MADHA ENGINEERING COLLEGE

(Affiliated to Anna University and Approved by AICTE, New Delhi) Madha Nagar, Kundrathur,
Chennai-600069

DEPARTMENT OF Master of Computer Application

R-2021

Lecture Notes

REFERENCES:

1. Dallas E Johnson, “Applied multivariate methods for data Analysis”, Thomson and Duxbury press, Singapore, 1998.
2. Richard A. Johnson and Dean W. Wichern, “Applied multivariate statistical Analysis”, Pearson Education, Fifth Edition, 6th Edition, New Delhi, 2013.
3. Bronson, R.,”Matrix Operation” Schaum’s outline series, Tata McGraw Hill, New York, 2011.
4. Oliver C. Ibe, “Fundamentals of Applied probability and Random Processes”, Academic Press, Boston, 2014.
5. Johnson R. A. and Gupta C.B., “Miller and Freund’s Probability and Statistics for Engineers”, Pearson India Education, Asia, 9th Edition, New Delhi, 2017.

CO-PO Mapping

CO	POs					
	PO1	PO2	PO3	PO4	PO5	PO6
1	2	-	2	2	1	2
2	2	-	2	2	1	1
3	2	-	2	1	1	2
4	3	1	2	2	1	2
5	3	-	2	3	2	3
Avg	2.5	0.2	2	2	1.2	2.4

RM4151 RESEARCH METHODOLOGY AND IPR L T P C
2 0 0 2

UNIT I RESEARCH DESIGN 6

Overview of research process and design, Use of Secondary and exploratory data to answer the research question, Qualitative research, Observation studies, Experiments and Surveys.

UNIT II DATA COLLECTION AND SOURCES 6

Measurements, Measurement Scales, Questionnaires and Instruments, Sampling and methods. Data - Preparing, Exploring, examining and displaying.

UNIT III DATA ANALYSIS AND REPORTING 6

Overview of Multivariate analysis, Hypotheses testing and Measures of Association. Presenting Insights and findings using written reports and oral presentation.

UNIT IV INTELLECTUAL PROPERTY RIGHTS 6

Intellectual Property – The concept of IPR, Evolution and development of concept of IPR, IPR development process, Trade secrets, utility Models, IPR & Biodiversity, Role of WIPO and WTO in IPR establishments, Right of Property, Common rules of IPR practices, Types and Features of IPR Agreement, Trademark, Functions of UNESCO in IPR maintenance.

UNIT V PATENTS 6

Patents – objectives and benefits of patent, Concept, features of patent, Inventive step, Specification, Types of patent application, process E-filling, Examination of patent, Grant of patent, Revocation, Equitable Assignments, Licences, Licensing of related patents, patent agents, Registration of patent agents.

TOTAL: 30 PERIODS

REFERENCES:

1. Cooper Donald R, Schindler Pamela S and Sharma JK, "Business Research Methods", Tata McGraw Hill Education, 11e (2012).
2. Catherine J. Holland, "Intellectual property: Patents, Trademarks, Copyrights, Trade Secrets", Entrepreneur Press, 2007.
3. David Hunt, Long Nguyen, Matthew Rodgers, "Patent searching: tools & techniques", Wiley, 2007.
4. The Institute of Company Secretaries of India, Statutory body under an Act of parliament, "Professional Programme Intellectual Property Rights, Law and practice", September 2013.

Course Outcomes:

At the end of this course, the students will have the ability to

1. Formulate and Design research problem
2. Understand and Comprehend the Data Collection Methods
3. Perform Data analysis and acquire Insights
4. Understand IPR and follow research ethics
5. Understand and Practice Drafting and filing a Patent in research and development

O-PO Mapping:

CO	PO					
	1	**2**	**3**	**4**	**5**	**6**
1	3	3	-	1	-	1
2	3	2	-	2	-	1
3	3	2	2	2	-	1
4	3	2	-	1	-	-
5	3	3	-	1	-	-
Avg.	3	2.4	0.4	1.4	-	0.6

MC4101 ADVANCED DATA STRUCTURES AND ALGORITHMS L T P C

3 0 0 3

COURSE OBJECTIVES:

- To understand the usage of algorithms in computing
- To learn and use hierarchical data structures and its operations
- To learn the usage of graphs and its applications
- To select and design data structures and algorithms that is appropriate for problems
- To study about NP Completeness of problems.

UNIT I ROLE OF ALGORITHMS IN COMPUTING & COMPLEXITY ANALYSIS 9

Algorithms – Algorithms as a Technology -Time and Space complexity of algorithms- Asymptotic analysis-Average and worst-case analysis-Asymptotic notation-Importance of efficient algorithms-Program performance measurement - Recurrences: The Substitution Method – The Recursion-

MADHA ENGINEERING COLLEGE

DEPARTMENT OF COMPUTER APPLICATION

RM4151 – Research Methodology & IPR

N.Vinodh MBA, M.Phil, Department of Management Studies

Unit-1

Overview of research process and design, Use of Secondary and exploratory data to answer the research question, Qualitative research, Observation studies, Experiments and Surveys.

Introduction and overview of Research process :

What is Research?

Definition: Research is defined as careful consideration of study regarding a particular concern or problem using scientific methods. According to the American sociologist Earl Robert Babbie, "research is a systematic inquiry to describe, explain, predict, and control the observed phenomenon. It involves inductive and deductive methods."

Inductive research methods analyze an observed event, while deductive methods verify the observed event. Inductive approaches are associated with qualitative research, and deductive methods are more commonly associated with quantitative analysis.

Research is conducted with a purpose to:

- Identify potential and new customers
- Understand existing customers
- Set pragmatic goals
- Develop productive market strategies
- Address business challenges
- Put together a business expansion plan
- Identify new business opportunities

What are the characteristics of research?

1. Good research follows a systematic approach to capture accurate data. Researchers need to practice ethics and a code of conduct while making observations or drawing conclusions.
2. The analysis is based on logical reasoning and involves both inductive and deductive methods.
3. Real-time data and knowledge is derived from actual observations in natural settings.
4. There is an in-depth analysis of all data collected so that there are no anomalies associated with it.
5. It creates a path for generating new questions. Existing data helps create more research opportunities.
6. It is analytical and uses all the available data so that there is no ambiguity in inference.
7. Accuracy is one of the most critical aspects of research. The information must be accurate and correct. For example, laboratories provide a controlled environment to collect data. Accuracy is measured in the instruments used, the calibrations of instruments or tools, and the experiment's final result.

What is the purpose of research?

There are three main purposes:

1. **Exploratory:** As the name suggests, researchers conduct exploratory studies to explore a group of questions. The answers and analytics may not offer a conclusion to the perceived problem. It is undertaken to handle new problem areas that haven't been explored before. This exploratory process lays the foundation for more conclusive data collection and analysis.
2. **Descriptive:** It focuses on expanding knowledge on current issues through a process of data collection. Descriptive research describe the behavior of a sample population. Only one variable is required to conduct the study. The three primary purposes of descriptive studies are describing, explaining, and validating the findings. For example, a study conducted to know if top-level management leaders in the 21st century possess the moral right to receive a considerable sum of money from the company profit.
3. **Explanatory:** Causal or explanatory research is conducted to understand the impact of specific changes in existing standard procedures. Running experiments is the most popular form. For example, a study that is conducted to understand the effect of rebranding on customer loyalty.

Research methods are broadly classified as **Qualitative** and **Quantitative**.

Both methods have distinctive properties and data collection methods.

Qualitative methods

Qualitative research is a method that collects data using conversational methods, usually open-ended questions. The responses collected are essentially non-numerical. This method helps a researcher understand what participants think and why they think in a particular way.

Types of qualitative methods include:

1. One-to-one Interview
2. Focus Groups
3. Ethnographic studies
4. Text Analysis
5. Case Study

Quantitative methods

Quantitative methods deal with numbers and measurable forms. It uses a systematic way of investigating events or data. It answers questions to justify relationships with measurable variables to either explain, predict, or control a phenomenon.

Types of quantitative methods include:

1. Survey research
2. Descriptive research
3. Correlational research

Remember, research is only valuable and useful when it is valid, accurate, and reliable. Incorrect results can lead to customer churn and a decrease in sales.

It is essential to ensure that your data is:

- Valid – founded, logical, rigorous, and impartial.
- Accurate – free of errors and including required details.
- Reliable – other people who investigate in the same way can produce similar results.
- Timely – current and collected within an appropriate time frame.
- Complete – includes all the data you need to support your business decisions.

8 Tips for Accurate Research Results:

1. Identify the main trends and issues, opportunities, and problems you observe. Write a sentence describing each one.
2. Keep track of the frequency with which each of the main findings appears.
3. Make a list of your findings from the most common to the least common.
4. Evaluate a list of the strengths, weaknesses, opportunities, and threats that have been identified in a SWOT analysis.
5. Prepare conclusions and recommendations about your study.
6. Act on your strategies
7. Look for gaps in the information, and consider doing additional inquiry if necessary
8. Plan to review the results and consider efficient methods to analyze and dissect results for interpretation.

Research design definition

Research design is the framework of research methods and techniques chosen by a researcher. The design allows researchers to hone in on research methods that are suitable for the subject matter and set up their studies up for success.

The design of a research topic explains the type of research (experimental, survey research, correlational, semi-experimental, review) and also its sub-type (experimental design, research problem, descriptive case-study).

There are three main types of designs for research: Data collection, measurement, and analysis.

The type of research problem an organization is facing will determine the research design and not vice-versa. The design phase of a study determines which tools to use and how they are used.

An impactful research usually creates a minimum bias in data and increases trust in the accuracy of collected data. A design that produces the least margin of error in experimental research is generally considered the desired outcome. The essential elements are:

1. Accurate purpose statement
2. Techniques to be implemented for collecting and analyzing research
3. The method applied for analyzing collected details
4. Type of research methodology

5. Probable objections for research
6. Settings for the research study
7. Timeline
8. Measurement of analysis

Proper research design sets your study up for success. Successful research studies provide insights that are accurate and unbiased. You'll need to create a survey that meets all of the main characteristics of a design. There are four key characteristics:

Neutrality: When you set up your study, you may have to make assumptions about the data you expect to collect. The results projected in the research should be free from bias and neutral. Understand opinions about the final evaluated scores and conclusions from multiple individuals and consider those who agree with the derived results.

Reliability: With regularly conducted research, the researcher involved expects similar results every time. Your design should indicate how to form research questions to ensure the standard of results. You'll only be able to reach the expected results if your design is reliable.

Validity: There are multiple measuring tools available. However, the only correct measuring tools are those which help a researcher in gauging results according to the objective of the research. The questionnaire developed from this design will then be valid.

Generalization: The outcome of your design should apply to a population and not just a restricted sample. A generalized design implies that your survey can be conducted on any part of a population with similar accuracy.

A researcher must have a clear understanding of the various types of research design to select which model to implement for a study. Like research itself, the design of your study can be broadly classified into quantitative and qualitative.

Qualitative: Qualitative research determines relationships between collected data and observations based on mathematical calculations. Theories related to a naturally existing phenomenon can be proved or disproved using statistical methods. Researchers rely on qualitative research methods that conclude "why" a particular theory exists along with "what" respondents have to say about it.

Quantitative: Quantitative research is for cases where statistical conclusions to collect actionable insights are essential. Numbers provide a better perspective to make critical business decisions. Quantitative research methods are necessary for the growth of any organization. Insights drawn from hard numerical data and analysis prove to be highly effective when making decisions related to the future of the business.

You can further break down the types of research design into five categories:

1. Descriptive research design: In a descriptive design, a researcher is solely interested in describing the situation or case under their research study. It is a theory-based design method which is created by gathering, analyzing, and presenting collected data. This allows a researcher to provide insights into the why and how of research. Descriptive design helps others better understand the need for the research. If the problem statement is not clear, you can conduct exploratory research.

2. Experimental research design: Experimental research establishes a relationship between the cause and effect of a situation. It is a causal design where one observes the impact caused by the independent variable on the dependent variable. For example, one

monitors the influence of an independent variable such as a price on a dependent variable such as customer satisfaction or brand loyalty. It is a highly practical research method as it contributes to solving a problem at hand.

The independent variables are manipulated to monitor the change it has on the dependent variable. It is often used in social sciences to observe human behavior by analyzing two groups. Researchers can have participants change their actions and study how the people around them react to gain a better understanding of social psychology.

3. Correlational research design: Correlational research is a non-experimental research technique that helps researchers establish a relationship between two closely connected variables. This type of research requires two different groups. There is no assumption while evaluating a relationship between two different variables, and statistical analysis techniques calculate the relationship between them.

A correlation coefficient determines the correlation between two variables, whose value ranges between -1 and +1. If the correlation coefficient is towards +1, it indicates a positive relationship between the variables and -1 means a negative relationship between the two variables.

4. Diagnostic research design: In diagnostic design, the researcher is looking to evaluate the underlying cause of a specific topic or phenomenon. This method helps one learn more about the factors that create troublesome situations.

This design has three parts of the research:

· Inception of the issue

· Diagnosis of the issue

· Solution for the issue

5. Explanatory research design: Explanatory design uses a researcher's ideas and thoughts on a subject to further explore their theories. The research explains unexplored aspects of a subject and details about what, how, and why of research questions.

Observation studies and experiment :

Observational studies

Observational studies are ones where researchers observe the effect of a risk factor, diagnostic test, treatment or other intervention without trying to change who is or isn't exposed to it. Cohort studies and case control studies are two types of observational studies.

Cohort study: For research purposes, a cohort is any group of people who are linked in some way. For instance, a birth cohort includes all people born within a given time frame. Researchers compare what happens to members of the cohort that have been exposed to a particular variable to what happens to the other members who have not been exposed.

Case control study: Here researchers identify people with an existing health problem ("cases") and a similar group without the problem ("controls") and then compare them with respect to an exposure or exposures.

Experimental studies

Experimental studies are ones where researchers introduce an intervention and study the effects. Experimental studies are usually randomized, meaning the subjects are grouped by chance.

Randomized controlled trial (RCT): Eligible people are randomly assigned to one of two or more groups. One group receives the intervention (such as a new drug) while the control group receives nothing or an inactive placebo. The researchers then study what happens to people in each group. Any difference in outcomes can then be linked to the intervention.

Strengths and weaknesses

The strengths and weaknesses of a study design should be seen in light of the kind of question the study sets out to answer. Sometimes, observational studies are the only way researchers can explore certain questions. For example, it would be unethical to design a randomized controlled trial deliberately exposing workers to a potentially harmful situation. If a health problem is a rare condition, a case control study (which begins with the existing cases) may be the most efficient way to identify potential causes. Or, if little is known about how a problem develops over time, a cohort study may be the best design.

Difference between Survey and Experiment :

S.No.	SURVEY	EXPERIMENT
01.	It refers to a way of gathering information regarding a variable under study from people.	It refers to the way of experimenting something practically with the help of scientific procedure/approach and the outcome is observed.
02.	Surveys are conducted in case of descriptive research.	Experiments are conducted in case of experimental research.
03.	Surveys are carried out to see something.	Experiments are carried out to experience something.
04.	These studies usually have larger samples.	These studies usually have smaller samples.
05.	The surveyor does not manipulate the variable or arrange for events to happen.	The researcher may manipulate the variable or arrange for events to happen.
06.	It is appropriate in case of social	It is appropriate in case of physical

	or behavioral science.	and natural science.
07.	It comes under field research.	It comes under laboratory research.
08.	Possible relationship between the data and the unknowns in the universe can be studied through surveys.	Experiments are meant to determine such relationships.
09.	Surveys can be performed in less cost than a experiments.	Experiments costs higher than the surveys.
10.	Surveys often deals with secondary data.	Experiments deal with primary data.
11.	In surveys there is no requirement of laboratory equipment or there is a very small requirement of equipment just to collect any sample of data.	In experiments usually laboratory equipment are used in various activities during the experiment process.
12.	It is vital in co-relational analysis.	It is vital in casual analysis.
13.	No manipulation is involved in surveys.	Manipulation is involved in experiments.
14.	In surveys data is collected through interview, questionnaire, case study etc.	In experiments data is collected through several readings of experiment.
15.	Surveys can focus on broad topics.	Experiments focuses on specific topic.

Exploratory research: Definition

Exploratory research is defined as a research used to investigate a problem which is not clearly defined. It is conducted to have a better understanding of the existing problem, but will not provide conclusive results. For such a research, a researcher starts with a general idea and uses this research as a medium to identify issues, that can be the focus for future research. An important aspect here is that the researcher should be willing to change his/her direction subject to the revelation of new data or insight. Such a research is usually carried out when the problem is at a preliminary stage. It is often

referred to as grounded theory approach or interpretive research as it used to answer questions like what, why and how.

For example: Consider a scenario where a juice bar owner feels that increasing the variety of juices will enable increase in customers, however he is not sure and needs more information. The owner intends to carry out an exploratory research to find out and hence decides to do an exploratory research to find out if expanding their juices selection will enable him to get more customers of if there is a better idea.

Another example of exploratory research is a podcast survey template that can be used to collect feedback about the podcast consumption metrics both from existing listeners as well as other podcast listeners that are currently not subscribed to this channel. This helps the author of the podcast create curated content that will gain a larger audience.

Types and methodologies of Exploratory research

While it may sound a little difficult to research something that has very little information about it, there are several methods which can help a researcher figure out the best research design, data collection methods and choice of subjects. There are two ways in which research can be conducted namely primary and secondary.. Under these two types, there are multiple methods which can used by a researcher. The data gathered from these research can be qualitative or quantitative. Some of the most widely used research designs include the following:

Primary research methods

Primary research is information gathered directly from the subject. It can be through a group of people or even an individual. Such a research can be carried out directly by the researcher himself or can employ a third party to conduct it on their behalf. Primary research is specifically carried out to explore a certain problem which requires an in-depth study.

- **Surveys/polls**: Surveys/polls are used to gather information from a predefined group of respondents. It is one of the most important quantitative method. Various types of surveys or polls can be used to explore opinions, trends, etc. With the advancement in technology, surveys can now be sent online and can be very easy to access. For instance, use of a survey app through tablets, laptops or even mobile phones. This information is also available to the researcher in real time as well. Nowadays, most organizations offer short length surveys and rewards to respondents, in order to achieve higher response rates.

For example: A survey is sent to a given set of audience to understand their opinions about the size of mobile phones when they purchase one. Based on such information organization can dig deeper into the topic and make business related decision.

- **Interviews:** While you may get a lot of information from public sources, but sometimes an in person interview can give in-depth information on the subject being studied. Such a research is a qualitative research method. An interview with a subject matter expert can give you meaningful insights that a generalized public source won't be able to provide. Interviews are carried out in person or on telephone which have open-ended questions to get meaningful information about the topic.

For example: An interview with an employee can give you more insights to find out the degree of job satisfaction, or an interview with a subject matter expert of quantum theory can give you in-depth information on that topic.

- **Focus groups:** Focus group is yet another widely used method in exploratory research. In such a method a group of people is chosen and are allowed to express their insights on the topic that is being studied. Although, it is important to make sure that while choosing the individuals in a focus group they should have a common background and have comparable experiences.

For example: A focus group helps a research identify the opinions of consumers if they were to buy a phone. Such a research can help the researcher understand what the consumer value while buying a phone. It may be screen size, brand value or even the dimensions. Based on which the organization can understand what are consumer buying attitudes, consumer opinions, etc.

- **Observations:** Observation research can be qualitative observation or quantitative observation. Such a research is done to observe a person and draw the finding from their reaction to certain parameters. In such a research, there is no direct interaction with the subject.

For example: An FMCG company wants to know how it's consumer react to the new shape of their product. The researcher observes the customers first reaction and collects the data, which is then used to draw inferences from the collective information.

Secondary research methods

Secondary research is gathering information from previously published primary research. In such a research you gather information from sources likes case studies, magazines, newspapers, books, etc.

- **Online research:** In today's world, this is one of the fastest way to gather information on any topic. A lot of data is readily available on the internet and the researcher can download it whenever he needs it. An important aspect to be noted for such a research is the genuineness and authenticity of the source websites that the researcher is gathering the information from.

For example: A researcher needs to find out what is the percentage of people that prefer a specific brand phone. The researcher just enters the information he needs in a search engine and gets multiple links with related information and statistics.

- **Literature research**: Literature research is one of the most inexpensive method used for discovering a hypothesis. There is tremendous amount of information available in libraries, online sources, or even commercial databases. Sources can include newspapers, magazines, books from library, documents from government agencies, specific topic related articles, literature, Annual reports, published statistics from research organizations and so on.

However, a few things have to be kept in mind while researching from these sources. Government agencies have authentic information but sometimes may come with a nominal cost. Also, research from educational institutions is generally overlooked, but in fact educational institutions carry out more number of research than any other entities.

Furthermore, commercial sources provide information on major topics like political agendas, demographics, financial information, market trends and information, etc.

For example: A company has low sales. It can be easily explored from available statistics and market literature if the problem is market related or organization related or if the topic being studied is regarding financial situation of the country, then research data can be accessed through government documents or commercial sources.

- **Case study research:** Case study research can help a researcher with finding more information through carefully analyzing existing cases which have gone through a similar problem. Such analysis are very important and critical especially in today's business world. The researcher just needs to make sure he analyses the case carefully in regards to all the variables present in the previous case against his own case. It is very commonly used by business organizations or social sciences sector or even in the health sector.

For example: A particular orthopedic surgeon has the highest success rate for performing knee surgeries. A lot of other hospitals or doctors have taken up this case to understand and benchmark the method in which this surgeon does the procedure to increase their success rate.

Exploratory research: Steps to conduct a research

- **Identify the problem**: A researcher identifies the subject of research and the problem is addressed by carrying out multiple methods to answer the questions.
- **Create the hypothesis**: When the researcher has found out that there are no prior studies and the problem is not precisely resolved, the researcher will create a hypothesis based on the questions obtained while identifying the problem.
- **Further research**: Once the data has been obtained, the researcher will continue his study through descriptive investigation. Qualitative methods are used to further study the subject in detail and find out if the information is true or not.

Characteristics of Exploratory research

- They are not structured studies
- It is usually low cost, interactive and open ended.
- It will enable a researcher answer questions like what is the problem? What is the purpose of the study? And what topics could be studied?
- To carry out exploratory research, generally there is no prior research done or the existing ones do not answer the problem precisely enough.
- It is a time consuming research and it needs patience and has risks associated with it.
- The researcher will have to go through all the information available for the particular study he is doing.
- There are no set of rules to carry out the research per se, as they are flexible, broad and scattered.
- The research needs to have importance or value. If the problem is not important in the industry the research carried out is ineffective.
- The research should also have a few theories which can support its findings as that will make it easier for the researcher to assess it and move ahead in his study
- Such a research usually produces qualitative data, however in certain cases quantitative data can be generalized for a larger sample through use of surveys and experiments.

Advantages of Exploratory research

- The researcher has a lot of flexibility and can adapt to changes as the research progresses.
- It is usually low cost.
- It helps lay the foundation of a research, which can lead to further research.
- It enables the researcher understand at an early stage, if the topic is worth investing the time and resources and if it is worth pursuing.

- It can assist other researchers to find out possible causes for the problem, which can be further studied in detail to find out, which of them is the most likely cause for the problem.

Disadvantages of Exploratory research

- Even though it can point you in the right direction towards what is the answer, it is usually inconclusive.
- The main disadvantage of exploratory research is that they provide qualitative data. Interpretation of such information can be judgmental and biased.
- Most of the times, exploratory research involves a smaller sample, hence the results cannot be accurately interpreted for a generalized population.
- Many a times, if the data is being collected through secondary research, then there is a chance of that data being old and is not updated.

Importance of Exploratory research

Exploratory research is carried out when a topic needs to be understood in depth, especially if it hasn't been done before. The goal of such a research is to explore the problem and around it and not actually derive a conclusion from it. Such kind of research will enable a researcher to set a strong foundation for exploring his ideas, choosing the right research design and finding variables that actually are important for the analysis. Most importantly, such a research can help organizations or researchers save up a lot of time and resources, as it will enable the researcher to know if it worth pursuing.

RM4151 RESEARCH METHODOLOGY AND IPR

N.Vinodh, Department of Management Studies

UNIT V

PATENTS

Patents – objectives and benefits of patent, Concept, features of patent, Inventive step, Specification, Types of patent application, process E-filling, Examination of patent, Grant of patent, Revocation, Equitable Assignments, Licenses, Licensing of related patents, patent agents, Registration of patent agents

What do you mean by patent?

A patent is **an exclusive right granted for an invention**. In other words, a patent is an exclusive right to a product or a process that generally provides a new way of doing something, or offers a new technical solution to a problem

What is a patent?

A patent is an exclusive right granted for an invention, which is a product or a process that provides, in general, a new way of doing something, or offers a new technical solution to a problem. To get a patent, technical information about the invention must be disclosed to the public in a patent application.

Benefits of Patents

A patent **gives you the right to stop others from copying, manufacturing, selling or importing your invention without your permission**. See protecting intellectual property. You get protection for a pre-determined period, allowing you to keep competitors at bay. You can then use your invention yourself.

Objectives of the global patent system

The core objective of the patent law is **to promote the progress of Science and useful arts**. It can be listed as: To encourage inventor: If a person puts efforts and resources in invention something that can be patented, he should have a provision that stops others from copying his work without his permission.

The global patent system should:

i. Be coherent and balanced:
    a. offering a fair level of protection to inventors/applicants from all backgrounds
    b. providing a fair balance between the rights of inventors/applicants and third parties
ii. Provide legal certainty to inventors/applicants and third parties alike
iii. Promote high quality patents by ensuring that patent protection is provided only to inventions that are new, involve an inventive step and are capable of industrial application
iv. Support economic growth:
    a. enabling global patent rights to be acquired in an efficient manner
    b. promoting consistent results in multiple jurisdictions
    c. promoting innovation and competition

Principles and commentary

The following principles and commentary have been prepared by the Chair, taking account of the objectives outlined above and the views of sub-group members. The principles are intended to encompass the views of all members of the sub-group, whilst recognising that differing views remain on how the principles should

best be implemented. The associated commentary takes account of these differing views, reflecting the various outcomes under consideration in respect of each principle, together with an indication of the level of support within the sub-group for each proposed solution.

1. Non-prejudicial disclosures / grace period

i. Inventors/applicants whose inventions have been disclosed prior to filing a patent application should, in certain circumstances, be given an opportunity to patent their invention

ii. Such circumstances should include breach of confidence or theft of information

iii. Any system which allows an invention to be patented after disclosure should take account of and balance the needs of:
   a. inventors/applicants, regardless of their level of IP expertise
   b. third parties (including those who could claim prior user rights)
   c. those whose primary focus is dissemination of knowledge and information

iv. Any system which allows an invention to be patented after disclosure should:
   a. provide a high level of legal certainty for applicants and third parties
   b. encourage early filing
   c. encourage research and development
   d. be applicable according to globally harmonised principles and rules so as to promote consistent results in multiple jurisdictions

Circumstances in which applicants should have the opportunity to patent a disclosed invention

While there is consensus that applicants should be given an opportunity to patent their invention where it has been disclosed due to breach of confidence or theft of information, there is no consensus on whether applicants should be given an opportunity to patent their invention where they have disclosed it themselves. There was more support, though not unanimity, for the opportunity to patent an invention which had been inadvertently disclosed[Footnote2].

Other characteristics of a potential grace period

Notwithstanding that there is no consensus on the introduction of a grace period covering disclosures by the applicant, there is agreement that, if such a system were to be introduced:

- It should be simple, with the same rules applying to all applicants and all types of disclosure deriving from the applicant, regardless of the intention or characteristics of the applicant.
- Encouraging transparency of the fact that the grace period has been invoked, for example through some form of declaration requirement, would increase legal certainty but place a burden on the applicant, and therefore further work should be conducted to explore how these factors could best be balanced.
- The duration of the grace period should be harmonised, and calculated from the priority date.

Regarding a declaration requirement, some believe that the increased legal certainty this would bring would not warrant the increased burden on the applicant. Others believe that an applicant who benefits from the grace period should bear some of the risk of using it, and therefore should be required to declare the graced disclosures of which they are aware. Some of those in favour of mandatory declaration believe that failure to declare a disclosure should result in administrative sanctions only, rather than loss of the benefit of the grace period for that disclosure.

There is no consensus on the optimal duration of the grace period, some believing the principles are best supported by duration of 6 months, others 12 months. However, there is agreement that, whatever the duration, it should itself be harmonized and should be calculated from the priority date in all jurisdictions.

Rights of third parties

The sub-group noted that the rights of third parties may have a significant effect on the way in which any grace period is used. All systems envisage certain circumstances in which the disclosure of third party inventions prior to the date of filing could affect the patentability of an application relying on a graced disclosure. To this extent the system incentivizes early filing.

Some members believe that it should be possible for prior user rights to arise where use of an invention in good faith is based on information derived from the applicant which has been disclosed to the public through a pre-filing disclosure during the grace period – to provide legal certainty to third parties and provide additional incentives for applicants to file early. Others believe that prior user rights should be a limited defence to patent infringement, and should not arise where knowledge of the invention has been derived from the applicant.

The sub-group members were open to further thought as to the possible interplay between third party rights and the grace period. Some felt that if it proved possible to reach agreement on the right balance between the interests of applicants and third parties, setting appropriate incentives to filing first, prior to disclosing the invention, and providing adequate protection for third parties, then the specific duration of the grace period might be less important, though it should still be harmonized.

2. Publication of applications

i. The global publication regime should be formulated to provide a clear time limit by which information about a potentially patented invention will be made public
ii. The timing of publication should provide for prompt dissemination of knowledge from all pending patent applications wherever filed
iii. Pending patent applications should be published promptly after the expiry of a **globally agreed timeframe**
iv. The globally agreed timeframe should balance the interests of inventors/applicants and those of third parties:
    a. inventors/applicants should be provided sufficient time to determine the likelihood of obtaining meaningful protection for their invention, and should they wish, to withdraw their application so as to retain the possibility of protecting their invention as a trade secret
    b. third parties should be provided with legal certainty regarding patent rights which are pending, so as to prevent duplication of R&D efforts and loss of investments
v. Patent office's should be able to delay or suppress publication of a pending application in exceptional circumstances
vi. Inventors/applicants should be able to request publication of an application prior to the globally agreed timeframe if they wish, as long as the requirements for publication under the applicable law are met
    - There is consensus that:

- 18 months from priority date is an appropriate timeframe to meet the principles outlined in paragraphs (iii) and (iv).
- Patent office's should be able to delay publication of a pending application beyond 18 months, or suppress publication of information within an application, in the following exceptional circumstances:
    - if publication would be prejudicial to public order, morality, or national security
    - if the application contains offensive or disparaging material
    - if a court order specifies that an application should not be published

The sub-group is open to considering any additional exceptional reasons which can be justified.

3. Conflicting applications

i. The grant of multiple patents for the same invention in the same jurisdiction should be prevented

ii. The patent system should allow for the protection of incremental inventions while ensuring that patent rights are not unjustifiably extended

iii. Any system which allows incremental inventions to be patented should:

a. balance the interests of inventors to protect incremental improvements on their own inventions with the interests of third parties to operate in the same field

b. promote innovation and competition

- There is consensus that:
    - Rules which govern conflicting applications should permit the patenting of incremental innovations, where this supports principle (iii).
    - Harmonised treatment of conflicting applications would be beneficial.
    - Further work should be conducted to compare various alternative approaches, bearing in mind the effects on innovation and competition.
    - There may be benefits to a harmonised system in which PCT applications apply as secret prior art upon international publication in any language.
    - Applications prosecuted directly through the national/regional route should apply as secret prior art only in those jurisdictions where they are or have been pending.

While there is agreement that rules which govern conflicting applications should support the principles outlined above, there is no consensus on how this should be achieved. In particular, there are differing views on what combination of features would best promote innovation and balance third party interests.

Some believe that innovation and competition are best supported by favouring the original applicant in respect of incremental inventions, by preventing their own earlier-filed applications ("secret prior art") being cited against them ("anti-self-collision"). However, among those members of the sub-group who consider that the original applicant should be favoured in this way, there are differing views as to the extent to which their secret prior art should count against other applicants – in particular whether it should count for the purposes of "enlarged novelty" or novelty and inventive step.

Other members of the sub-group believe that innovation and competition are best supported by providing equal access to the protection of incremental inventions for all applicants. They believe that this can be achieved by having no anti-self-collision, with secret prior art counting for novelty only against all applicants.

It is recognized that there could also be merit in considering new solutions which as yet do not exist – for example a system where secret prior art is applied for novelty and inventive step, and anti-self-collision applies for inventive step only.

The sub-group therefore agreed to carry out further work on these options.

The sub-group could see the logic underlying all of the present approaches regarding applications filed under the Patent Cooperation Treaty (PCT) – whether they should be applicable as secret prior art once they have been published in any language, once they have been published in an official language of the jurisdiction in which they are to be considered, or once they have entered the national/regional phase of the jurisdiction in which they are to be considered. The sub-group did not reach a definitive position on this issue. However, as patent systems become increasingly internationalized the sub-group could see there may be benefits to a harmonized system in which PCT applications apply as secret prior art once they have been published in any language. This would provide consistent legal certainty across different jurisdictions and respect the purpose of the PCT to give international applications the effect of a national filing in all designated member states. The sub-group agreed that this merited further discussion.

4. Prior user rights

i. A third party who has started using an invention in good faith prior to the filing of a patent application for that invention by another party should have a right to continue to use that invention

ii. The circumstances under which prior user rights arise, including the extent to which they rely on actual use having taken place, should balance the interests of third parties to protect their investments with the interests of the inventor/applicant

- There is consensus that:
  - Prior user rights should not arise through mere possession or knowledge of an invention by a third party.
  - Prior user rights should be limited to the territory in which the activity giving rise to prior user rights has taken place.

There is a degree of convergence, but not unanimity, that prior user rights should arise where a third party has, in good faith, made effective and serious preparations to use an invention. Those who hold this view believe that the process of innovation can be long and complex and it is arbitrary to use actual use of the invention as the threshold when substantial investments may have begun far before then. Those members believe, therefore, that it is fair, efficient and in the public interest that these investments should be protected whether or not actual use has taken place.

Others believe that prior user rights should arise only where actual use of the invention has taken place, noting that this is a clear test which avoids uncertainty regarding whether preparations are substantial enough, and ensures that prior user rights exist as a limited defence to infringement.

As far as the critical date is concerned, it was noted that in most, but not all, jurisdictions, prior user rights can arise up until the priority date of the invention. The sub-group recognised the benefits of harmonising the point in time by which prior user rights could arise.

As noted in section 1, some believe that it should be possible for prior user rights to arise where use of an invention by a third party in good faith is based on knowledge derived from a graced disclosure by the inventor/applicant. Others believe that prior user rights should be a limited defence to patent infringement, and should not arise where the information is derived from the inventor/applicant.

5. Prior art

i. Patents should only be granted for contributions that place in the hands of the public information that had not been previously known

ii. The scope of prior art should be properly defined to ensure that the subject matter for which exclusive rights are granted truly represents a contribution to, and not an encroachment on, the public domain

iii. Subject to agreed exceptions, prior art should consist of all information that has been made available to the public anywhere in the world before the earliest effective filing date of the claimed invention

- There is consensus that the principles outlined above underpin the patent system, and are therefore important for understanding how the principles in this document as a whole should operate.

Advantages of patents

- A patent gives you the right to stop others from copying, manufacturing, selling or importing your invention without your permission. See **protecting intellectual property**.
- You get protection for a pre-determined period, allowing you to keep competitors at bay.
- You can then use your invention yourself.
- Alternatively, you can license your patent for others to use it or you can sell it. This can provide an important source of revenue for your business. Indeed, some businesses exist solely to collect the royalties from a patent they have licensed - perhaps in combination with a registered design and trade mark. See **how to license a patent**.

Disadvantages of patents

- Your patent application means making certain technical information about your invention publicly available. It might be that keeping your invention secret may keep competitors at bay more effectively.
- Applying for a patent can be a very time-consuming and lengthy process (typically three to four years) - markets may change or technology may overtake your invention by the time you get a patent.
- Cost - it will cost you money whether you are successful or not - the application, searches for existing patents and a patent attorney's fees can all contribute to a reasonable outlay. The potential for making a profit should outweigh the time, effort and money it takes to get and maintain a patent. **Not all patents have financial value**.
- You'll need to remember to pay your annual fee or your patent will lapse.

- You'll need to be prepared to defend your patent. Taking action against an infringer can be very expensive. On the other hand, a patent can act as a deterrent, making defense unnecessary. Read more about **patent protection and enforcement**.

What are the features of patent rights?

In other words, a patent **provides its owner a 'right to exclude' others but not a 'right or freedom to use" the patented invention**. So, a patent gives its owner only an exclusive right to prevent or stop others, from making, using, offering for sale, selling or importing a product or process.

Patent applications: the three criteria

Patent applications must satisfy the following three criteria:

- **Novelty**

  This means that your invention must not have been made public – not even by yourself – before the date of the application.
- **Inventive step**

  This means that your product or process must be an inventive solution. It cannot be a solution that would be obvious to a manufacturer. Take the example of a different attachment method. Instead of welding the tubes of a swing together, they might be screwed together. This may well be a new method of making swings. But for someone involved in making them, it is too obvious a solution to be called an inventive step.
- **Industrial applicability**

  This criterion implies that it must be possible to actually manufacture the new invention. In other words, you can apply for a patent on a new kind of playing card that is easier to hold than existing cards. But you can't obtain a patent for an idea for a new card game.

What Is the Inventive Step?

The inventive step is used to find out if the patent is in fact for a new item or just an obvious improvement on an existing item. Inventive steps make sure patents aren't awarded to existing inventions that the "inventor" just improved upon. These patents could allow someone to make money off of an item just because they tweaked it. This patent could also allow them to sue companies that improve their own processes just because they made small changes as well.

The applicant must prove that the improvement isn't obvious to people within the industry and that there are actually improvements that come with patenting the idea.

One of the key words when talking about the inventive step is "obvious." Many people also refer to the inventive step as the "non-obviousness clause." The EPO defines this as going beyond the expectations of technology, instead of just following the next natural step.

For the most part, the term "inventive step" is used by our counterparts in Europe**; however, it is synonymous with the term "non-obviousness" that Americans use today.**

For example of an inventive step, it's a fairly common fact in the gardening community that plants need water and vitamins to grow. An inventive step would be mixing the two in a product, because it's easy to assume that gardeners have been doing that for decades.

Another term that is often used for the inventive step is novelty. In Future Science, Dr. Jonathan Atkinson and Dr. Rachel Jones define novelty as an idea that a patent should not be available to the public before it was filed. This includes:

- Discussing it at a conference or exhibition.
- Selling it or giving it away.
- Promoting the event in marketing materials.

By filing for a patent before it hits the market, the owner is able to prove that it was his idea, and that no one else has had the thought.

Why is the Inventive Step Rule Important?

The inventive step rule lets companies continue coming up with new ideas without worrying about running into a patent law. Instead of stopping natural progress (and creating a monopoly for the company that has the idea first), this clause allows companies to continue updating their systems to save money and resources.

In a paper presented at the Fordham Conference, John Richards explains that 'obvious' is Latin for 'upon the road,' or the next steps that companies or inventors would take in the process. Technology is an important example of this. Over the past several years, new technology has followed a pattern where it becomes lighter, cheaper, and smaller. Look at the first cell phone and how far it has improved to become the phones we use today. Patenting a model that is slightly lighter or smaller than a competitor (like an iPhone that's made of plastic or a lighter metal) would be an inventive step that follows an obvious progression – not an invention.

Reasons to Consider Not Using the Inventive Step

One of the biggest challenges faced in patent law and novelty is the subjective nature of the tests. It's hard to prove something is just an inventive step, so today's rules follow interviews and personal opinions.

The United States patent office uses the Teaching-Suggestion-Motivation (TSM) test to determine non-obviousness. Some say it's too controversial to use, but it proves that there must be some teaching or suggestion involved to form an idea. It is also referred to as a way to prevent hindsight bias.

Most legal teams compare the new idea with the existing item through interviews. They find people who are familiar with the industry and ask them about the differences between the two items. Some critics say this process is biased because of the person's background, education, and experience. This means the criteria changes from industry to industry, and also changes from person to person.

Reasons to Consider Using the Inventive Step

Even though some people might think the current rules are unfair, there are important reasons for keeping the inventive step rules on the books. The Omics Group says the inventive step rules follow the original goals of the patent system. Their main goal is to encourage people to come up with new ideas that they can protect and make money from. Trying to buy old ideas or making minor tweaks focuses on the money part of the patent office, not the invention part.

There are also people who are trying to improve the current rules and how lawyers prove them.

The European Journal of Law and Technology listed one study in which more than 200 students reviewed two products to see if it was an inventive step or actual invention. Based on this data, the patent judges had more information and opinions about whether something was obvious.

Currently, in the United States, determining obviousness requires three steps:

- **Evaluating the scope and content of the art.**
- **Determining the differences between the original art and the new invention.**
- **Resolving the skill level within the art.**

This provides a concrete way to determine obviousness that can be applied across most cases.

These steps are also known as Graham factors, based on the case *Graham et al. v. John Deere Co. of Kansas City et al*. They also consider three additional factors in the products:

- **Commercial success**
- **Long-felt but unsolved needs**
- **Failure of others**

This helps create a big picture view of how the product will affect society if it is patented.

Examples of the Inventive Step

According to the World Intellectual Propery Organization, inventive step rules have been around since the 15th century, but they became common in the 19th century. As patents become more advanced, the inventive step rules are used more often – especially in the tech field where software companies are making tools that common people aren't familiar with.

What is the difference between an inventive step and a non-obvious rule? It all depends on where you live. Simplicable says that most American legal systems use the term "inventive step," while most European systems use the term "non-obviousness." While this article has used both terms, it's important to stick to the one within your region to prevent confusion.

The inventive step clause will mostly be used in cases of patent challenges between two companies. One side will argue that the invention is new, while the other side says it was an inventive step that its clients came up with on their own. This shows the value of the non-obviousness rules even when a patent is approved.

The nature of the inventive step means companies have a chance to win their patents when they face the judge. However, if your company is not sure how to develop a case to prove the difference between the two, consider posting a job to hire a lawyer through UpCounsel who has experience in facing patent challenges and these non-obvious clauses. You can receive multiple free custom quotes from the top 5% of lawyers with an average of 14 years of experience.

Patent/ Types of Applications/ Specification - Reference

A patent is a right granted by the government to inventors in order to exclude others from making, using, offering for sale, or selling the patented invention in the United States or importing the invention into the U.S. This means that you get total control over your patented invention and get to decide who, if anyone, can use your invention.

Patents are valid for 20 years from the date you file your application, but patent rights are, in general, only enforceable from the date your patent is approved by the U.S. Patent and Trademark Office (USPTO).

According to the patent office, an invention is “any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof.” For technology product managers, just about any new product or feature is patentable: hardware, software, business methods, etc. Every new feature and product you create should be examined for patentability.

While just about anything can be patented, a few criteria must be met. Specifically, patented inventions must meet three characteristics: novel, useful and not obvious.

Novel. The novelty requirement is straightforward: Your invention must be new. Inventions that already exist cannot be patented. This gets a little tricky, because you can patent new uses for existing products. This frequently happens when a new use is discovered for an old pharmaceutical drug, where new testing shows the drug to effectively treat a completely unrelated disease. The new use has to be truly new and unrelated to the original use.

Useful. The usefulness requirement is two-fold: That your invention has a useful purpose and that it must actually perform its intended purpose. A useful purpose can be almost anything: The patent office, for example, issued Patent No. 6,368,227 to a boy who claimed the invention of swinging side-to-side, claiming the usefulness of joy. Along with good feelings or whatever other benefit you wish to claim, the invention must also work in order to be useful. If the patent office thinks your invention might not work, they may ask you to prove that it does.
Patent application number 20,030,114,313 claimed the invention of warp drive, which is "a system whose propulsion relies on warping space-time as opposed to the ejection of material to provide thrust." While the application packed more than400 paragraphs of technical detail, the patent office cast a skeptical eye, and asked the inventor to provide a working model before the patent application could be examined. (A model, working or otherwise, was unfortunately not delivered by the inventor.)

Not obvious. The "not obvious" requirement means that an inventive step is required. Your invention has to be different enough from what is already out there in the field in order to be patentable. This is an area, however, where the law is not very clear.Patents are regularly granted for surprisingly small improvements in a field.You should not let your thinking on this requirement prevent you from trying to patent something. If you have an idea that you think is worth patenting, explain it to your patent agent or attorney, and let them tell you if they think it meets the "not obvious" requirement.

In addition, you cannot patent the laws of nature, physical phenomena, and abstract ideas. Things like Isaac Newton's theory of universal gravitation or Albert Einstein's theory of general relativity are not patentable.

Three basic kinds of patents are allowed:

1. **Utility patents,** which cover inventions that function uniquely to produce a useful result.
2. **Design patents,** which cover the unique, ornamental, or visible shape or surface of an object.
3. **Plant patents**, which cover asexually reproducing plants.

Types of Patent Application

A **patent** is a statutory authorization or license which establishes a right or title over an invention for a particular period. It is primarily meant for the prevention of other businesses or its kind from making, using or selling an invention of a similar nature. In this article, we look at the different types of patent application in detail.

Patent Application

A patent application is a plea for the grant of a patent for the invention described and claimed by the applicant. An application for this purpose generally comprises of a description of the invention, added with

official forms and correspondence relevant to the application. Patent applications are of several types, and each one of them caters to a unique purpose.

Types of Patent

The types of patent application are:

1. Provisional Application
2. Ordinary or Non-Provisional Application
3. Convention Application
4. PCT International Application
5. PCT National Phase Application
6. Patent of Addition
7. Divisional Application

The rest of the article covers these types in detail.

Provisional Application

A provisional application, also known as a temporary application, is filed when an invention is under experimentation and isn't finalized. Moreover, it is a preliminary application which is filed before the patent office for claiming priority, as the Indian Patent Office follows the 'First to File' system (known popularly as the First-Come-First-Served-Basis). In technical terms, early filing of an invention will prevent the occurrence of any other related inventions from being designated as prior art to the inventor's application.

To add more, this type of patent application is filed when an invention requires additional time for development. If an application is supported by a provisional specification, the applicant is necessitated to file a complete specification within twelve months from the date of filing a provisional application. A failure in this part would render the application void.

An application for this purpose must include a brief explanation of the invention and must be drafted in a meticulous manner so as to ensure that the priority rights are secured for the invention.

Ordinary or Non-Provisional Application

This type of application is filed if the applicant doesn't have any priority to claim or if the application is not filed in pursuance of any preceding convention application. It must be supported by a complete specification, the likes of which must depict the invention in detail.

Complete specification can be filed through:

- **Direct Filing** – wherein complete specification is initially filed with the Indian Patent Office without filing any corresponding provisional specification.
- **Subsequent Filing** – wherein complete specification is filed subsequent to the filing of the corresponding provisional specification and claiming priority from the filed provisional specification.

A complete specification entails the following:

1. Title
2. A preamble to the invention.
3. The technical field of the invention.
4. Background of the invention.
5. Objects of the invention.
6. Statement of the invention.

7. A brief description of the drawings
8. A detailed description of the invention.
9. Claims
10. Abstract

Convention Application

A convention application is filed for claiming a priority date based on the same or substantially similar application filed in any of the convention countries. To avail a status of convention, an applicant is required to file an application in the Indian Patent Office within a year from the date of the initial filing of a similar application in the convention country. To re-iterate in simpler terms, a convention application entitles the applicant to claim priority in all the convention countries.

PCT International Application

As can be deciphered from its name, a PCT Application is an international application. Though the application does not provide for the grant of an international patent, it paves the way for a streamlined patent application process in many countries at one go. It is governed by the Patent Corporation Treaty and can be validated in up to 142 countries. Filing this application would protect an invention from being replicated in these designated countries.

Unlike other applications, it renders the application a time-frame of 30-31 months to enter into various countries from the international filing date or the priority date, thereby affording the applicant with additional time to access the viability of the invention.

Apart from this, it renders the following other benefits:

- The application provides an International Search Report citing prior art, which discloses whether or not the invention is novel.
- It provides an option for requesting an International Preliminary Examination Report, which is a report that contains an option on the patentability of the invention.
- The aforementioned reports facilitate the applicant to make more informed choices early in the patent process, as he/she can amend the application to deal with any conflicting material. Also, the applicant would receive a glimpse of the patentability of the invention before incurring charges for filing and prosecuting the application in each country.

An applicant from India can file this application at:

- The Indian Patent Office (IPO), which acts as the receiving office.
- The International Bureau of WIPO, either after availing a foreign filing permit from IPO or after six weeks and 12 months of filing an application in India.

PCT National Phase Application

It is considered essential for an applicant to file a national phase application in each of the country wherein protection is sought for. The time-frame for filing the same is scheduled within 31 months from the priority date or the international filing date, whichever is earlier. The time-limit could be enhanced through National Laws by each member country.

With respect to the National Phase Application, the title, description, abstract and claims as filed in the International Application under PCT shall be considered as the Complete Specification. Apart from this, the regulations applicable for filing and processing an ordinary patent application is also applied here.

Patent of Addition

This application must be filed if the applicant discovers that he has come across an invention which is a slight modification of the invention which has already been applied for or patented by the applicant. It can only be filed if the invention doesn't involve a substantial inventive step.

A patent of addition is only granted after the grant of the parent patent, and hence no separate renewal fee should be remitted during the term of the main patent. Moreover, it shall be granted for a term equal to that of the patent for the main invention, and therefore expires along with the main patent. The date of filing here shall be the date on which the application for patent of addition has been filed.

Divisional Application

An applicant may choose to divide an application and furnish two or more applications if a particular application claims for more than one invention. The priority date for these applications is similar to that of the parent application.

What are the five steps to filing a patent?

1. Understand Your Invention.
2. Research Your Invention.
3. Choose the Type of Protection.
4. Draft Your Patent Application.
5. Wait for a Formal Response.

1. Understand Your Invention

The first step in how to get a patent is to understand your invention. What aspect (or aspects) of the invention makes it new and useful? Suppose you made a custom pair of shears that are useful for cutting thin strips of fabric. The scissors have a custom-designed handle with different shaped finger loops and an additional set of pivots that allow the cutter to feel the slightest resistance when cutting through very thin fabric.

Once you identify the aspects that make your invention new and useful then you need to consider the scope. Are there other ways to make your invention? For example, could you use springs instead of pivots? Could you change the shape of the finger loops or the materials they are made out of? Find all the possible ways to make your invention work, even if they're not as good as the preferred way you make your invention.

Next, look at whether or not your invention has a broader application. Is there something special about thin fabric or could you use the scissors for any type of delicate work? Would the shears be useful for surgery or other precision applications? If so, would you need to make further modifications? Asking these types of questions early on will help you search, protect and benefit from the full scope of your invention. It will give you more strategic options and, most likely, a more valuable patent.

2. Research Your Invention

A patent requires absolute novelty. Part of the rationale of giving inventors exclusive rights to their inventions is to encourage the inventor to teach the public how to make the invention work. If a reasonably skilled person could make the invention work from information already available to the public, then what benefit is there to giving the inventor exclusive rights?

You have to search to find any relevant publication, presentation, sales brochure, **patent application** or issued patent that overlaps with your invention or some components of your invention. For example, you may

find a technical white paper that discusses precision robotic manufacturing. You notice that the robotic arms have the same kind of pivots as your super sensitive scissors. The reference is considered prior art, even if it is in a different field.

The United States Patent and Trademark Office (USPTO) requires you to disclose any publication, patent or other written document that you know to be relevant to your patent. Failure to do so could disqualify your patent, even after the USPTO issues it.

3. Choose the Type of Protection

Back to the pair of scissors that you made. Let's say you really like them, but they leave a lot to be desired. The pivots are too loose and you want to spend more time tinkering to get the tension just right. You also want to try out some new materials for the finger loops. If you want to get a patent on file but leave some room to tinker, you may want to file a provisional patent application.

A provisional application serves as proof that you are the inventor as of the date you file the patent. You can then take a year to file the actual patent application (what most people call a patent is actually called a utility patent). In that year, you can experiment and perfect the prototype that you built. You cannot, however, add anything new. If your new prototype includes super sensitive pressure sensors that you did not include in the provisional application then you will need to file a new patent application to get the benefit of the new sensors.

When considering what patent protection is available to you, do not rule anything out. For example, you may shape the finger loops on your scissors because they have a real functional improvement in improving sensitivity to resistance. They may also produce a very distinctive appearance. You could file a **design patent application** for the distinct appearance of the finger loops in addition to or in place of a **utility patent application**. Generally, a design patent protects the way an article looks, while a utility patent protects the way an article is used and works. Overlapping patent protection is incredibly important and makes your **intellectual property that much more valuable**.

4. Draft Your Patent Application

Drafting a patent application, even a provisional patent application, is tricky. Patent applications have several parts, each of which can be rejected for technical or formal reasons. If you are going to file it yourself, read the Manual of Patent Examining Procedure. Make checklists for each portion of your application and triple-check your work. If you are lucky, a mistake will only cost you time and money. If you are not, you can lose your filing date or cripple your ability to get an issued patent.

Drafting a patent is a skill that takes experience to develop and a team to execute. At all other steps, there is a lot of work the inventor can do. But when it comes to drafting the application itself, a professional will be very helpful.

5. Wait for a Formal Response

Do not expect to hear back from the patent office for a long time, usually a year or longer. When you do hear back, the examiner may argue that your invention is not novel in light of the prior art, that your invention is not the kind of thing that you can get a patent for, or that you have failed to fully explain how the invention works. If this does happen, you might want to seek professional advice to help craft your response.

In the meantime, while you are waiting for your response, get to work. Unless inventing is purely a hobby, you are investing your time and money in intellectual property that you think may have commercial value. If you are looking to license your scissor design to a leading tool company, for example, start talking to them now-and make sure your patent protects the features they consider most valuable. Because your patent is pending and your invention is protected, you can discuss whatever you need to seal the deal you want.

Patent Examination Process:

Once a patent application is filed in India and published after 18 months of filing the application, the next step involves the examination of the patent application. The examination process starts once the applicant files a request for examination within 48 months from the date of filing of the application or the priority date whichever is earlier.

During the first examination stage, the examiner prepares the examination report incorporating all the statutory objections for the given patent application. The examiner performs a patent search in order to identify the prior arts relevant to the invention. The objections are well communicated and properly defined so as to be understood by the addressee without seeking further clarification. The First

Examination Report (FER) is generally sent to the applicant along with the application and specification within six months from the date of publication or from the date of request for examination. The law in place requires that the objections are supported by correct legal provisions and proper reasoning. The objections once taken are maintained and are withdrawn only after justifying the proper reason for withdrawal.

The examination report may either be favorable or adverse to the applicant. If the report is favorable, the applicant has to put the application in order for grant within 12 months from its date. If the report is adverse, it normally includes formal objections relating to errors in the Forms or Fees and substantive objections relating to patentability requirements. During the process, the applicant can draft a response by amending the application in order to overcome the objections raised in the FER. The amendment can be allowed only if it is by way of disclaimer, correction or explanation. The amendment will not be allowed if the specification as amended describes matters which are not in substance disclosed or shown in the specification. Alternatively, the applicant may also request for a hearing within 1 month from the date of receiving the examination report to explain the reasons for non-acceptance of objections to the examiner.

The examiner can also withdraw the application any time after filing of application and before the grant of patent by filing a written request and paying the requisite fee.

Patentability: Applying For Patent

As we know that not every invention get patented, Patent is granted to the owner of the Patent when his/her invention satisfies the conditions for Patentability. Such conditions are as follows:

- Novelty
- Inventive step or non-obviousness
- Industrial Application

Section 3 and Section 4 deals with the list of exceptions that do not fall under the invention and hence are non-patentable.

Procedure for Grant of Patent

Persons entitled to apply for patents

—(1) Subject to the arrangements contained in section 134, an application for a patent for an invention might be made by any of the accompanying persons, in other words,—

(a) By any individual professing to be the valid and first creator of the invention;

(b) By any individual being the assignee of the individual professing to be the valid and first innovator in regard of the privilege to make such an application;

(c) By the legitimate agent of any deceased individual who is preceding his demise and is qualified to make such an application.

(2) An application under sub-section (1) might be made by any of the persons alluded to in that either alone or mutually with some other individual.

Filing of Application- Provisional/Complete: The Patent Application should be filed in ***form 1*** accompanied by either provisional or complete specification in ***form 2*** (If an applicant is not ready with the complete invention and need some more time for it then filing for the provisional application is recommended).

Publication of Application: The publication of the application is made after the expiry of 18 months from the priority date and no fees are required by the inventor. A prior- request for publication can be made **(Rule 24A)** under section **11A(2) in form 9 (optional step).**

Request for Examination(REF): The request for examination to examine the patent application is made in **form 18** (including fee) within **48 months** from the filing date by the applicant.

Examination issue of First Examination Report(FER): The controller sends the patent application to the examiner who checks for patentability as per the patentability criteria and creates the first examination report (FER).

Any objection raised regarding the patentability requirements during examining the patent application has to have complied within **12 months.**

Grant of Patent: Once the application meets all the requirements of patentability, the patent is granted to the inventor with the seal form patent office and is notified in the journal from time to time.

Opposition: Section 25 of the Act deals with the opposition to grant of patents and are of two types: **Pre Grant** (before the patent is granted) and **Post Grant** (after 1 year of grant of the patent). The opposition can be filed by anyone interested in the field of the invention in form 7 with the prescribed fee **within 12 months** from the date of publication of the patent.

Grounds of Opposition to Patent

- Obtained wrongly or fraudulently.
- The invention has been already published and known.
- Not involved in any of the inventive step.
- Not completed within 12 months.
- No clear and explicit description of the invention.
- Not considered an invention based on the subject matters for the invention.

Types of Patent Application

There are four types of Patent Application namely:

Provisional Application: This application is filed when the inventor is not quite ready with the invention and needs more time for the development of his invention and also don't want to lose the priority date. After

12 months of filing the provisional application, the complete application should be filed otherwise the patent application will not be considered. The provisional certificate may or may not have claimed.

Complete Application: Filing of the complete application, describes that the invention is complete. The complete application have claims.

Elements of complete application:

- Description of invention
- The best method of performing
- Claims
- Abstract

Convention Application: When an inventor or an applicant files the patent application in Indian Patent Office claiming for a priority date based on a similar application filed in convention countries, such applications are convention application.

Patent Cooperation Treaty (PCT) – International Application: A PCT application is an international application governed by the Patent Cooperation Treaty further administered by the World Intellectual Property Rights (WIPO).

Divisional Application: When an applicant feels that he has come across an invention which is a slight modification of the invention for which he has already applied for or has obtained the patent, the applicant can go for the patent of addition if the invention does not involve a substantial inventive step. There is no need to pay the separate renewal fee for the patent of addition during the term of the main patent and it expires along with the main patent.

Patent of Addition: when application made by applicant claims more than one invention, the applicant on his own files two or more applications, as applicable for each of the inventions. This type of application, divided out of the parent one, is called a Divisional Application. The priority date for all the divisional applications will be the same as that claimed by the Parent Application.

Documents Required For Patent Application

- Application for grant of the patent in **form 1**
- **Proof of right** to file the application from the inventor.
- Provisional/complete specification in **form -2**
- **Statement and undertaking** under **section 8 in Form 3**
- **Declaration as to inventorship**
- **Power of authority** in **form 26**
- **Applicant's signature** and an appropriate date
- **If application pertains to a biological material** obtained from India, submission of the **permission from the National Biodiversity Authority**
- Request for Examination- **Form 18**
- Requisite Statutory Fees

Rights of Patentee

Section 48 of the Act deals with the rights of conferring upon the patentee after the grant of the Patent.

- Exclusion of the third party to use, sell, or import the patented product without the patentee's consent.
- Exclusion of the third party from using, selling or importing the patented product (if the subject matter is the process) without the patentee's consent.

Infringement and Remedies for patent

Infringement of patent refers to the violation of the rights of the patent holder that is whenever a person exercises the rights of the patent holder without the patent owner's consent, he causes infringement.

Types of Patent Infringement:

- **Direct Infringement** – Directly selling, marketing, or using commercially, any product which is substantially close to the patented product without the consent of the patentee.
- **Indirect Infringement-** Deceitful and accidental patent infringement in any incident is an indirect infringement.
- **Contributory infringement-** If the person knowingly infringes the rights of the patent holder, it refers to contributory infringement.

Some acts that would not lead to infringement are as follows.

- Government use: As per Section 100 a patented invention can be used by the central government for its own use and as per section 47, the patented invention can be imported by the government.
- Exemption on experiments and research: the use of a patented invention for experiments and teaching purposes does not come under infringement.
- Patented inventions on drugs and medicines can be imported by the government.
- Any patented invention on foreign vessel/ aircraft/ vehicle comes to India is not an infringement.

Filing suits for infringement

Section 104 of the Patent Act, 1970 deals with the filing of the suit by the patentee against the infringement. The patentee can file a suit in a district court or directly in the high court. Suit for patent infringement can be filed after the grant of patent yet the patentee can also claim for damages committed between the publication of patent application to the grant of the patent.

The burden of proof is on the patentee after the grant of the patent but if the invention is a process than the burden to prove for infringement lies on the defendant.

Remedies for patent

Section 108 deals with relief or remedies against the infringement.

Injunction

The injunction is the most common form of Remedy granted in Patent infringement proceedings. The injunction is the order of the court restricting a person from beginning or continuing a course of action (infringing in this case) threatening or invading legal rights of a person.

Types of Injunction

There are two types of an injunction-

1. Interim Injunction
2. Permanent Injunction

Interim Injunction restricts the person temporarily from doing act and is granted before the full-fledged trial.

Permanent Injunction, on the other hand, restrains a person from doing a specified act and can be granted after the full-fledged trial.

Injunctions are preventive, prohibitive or restrictive that is restricting someone from doing a specified act or mandatory that is, they compel or orders a person to do something.

The plaintiff can obtain interlocutory order in the form of a temporary injunction whenever a case of patent infringement occurs from the court by proving the following facts:

1. The prima facie case of infringement
2. The balance of convenience in his favour
3. If the injunction is not granted he/she shall suffer irreparable damage.

Damages and account of profit

If the suit is in favour of the plaintiff, the court can award either damages or directs the defendant to render an account of profits but not both.

Exceptions and Limitations of Patent in India

Types of Exceptions & Limitations

Article 30 of TRIPS (Trade-Related Aspects of Intellectual Property Rights) allows for limited exceptions to the exclusive rights conferred by a patent.

- Exception on Non-Commercial use

The exclusive rights conferred by a patent does not allow the private or commercial activity.

The Government has the power to grant a license, known as Compulsory License (CL) , to a third party to use the patented invention (when the patentee is not using the invention for profit) so as to restrict the rights of the patentee for the purpose of preventing the abuse/ misuse of the rights by the property holder and to prevent the negative effect of such action on the public.

When the patented invention is not commercialized in India or the invention is not available to the public at reasonable prices or the invention is not manufactured in requisite amount, then the government grant such license.

- Exception on Experimental / Scientific Research

Section 47 of the Act subsection 3 deals with the exception on experimental and scientific use of Patented invention, the grant of a patent is subject to the condition that any product or process, in respect of which the patent is granted, may be made or used by any person for the purpose merely of experiment or research including the imparting of instructions to the students.

This form of exception grants third parties to carry out experiments and scientific processes for teaching students without infringing the rights of the patent holder.

- Exception on Regulatory use or Private use

Section 107A of the Indian Patent (Amendment) Act, 2005 deals with the exception of regulatory and private use also referred as bolar Provision, this exemption allows the manufacturers of generic drugs to undertake steps reasonably related to the development and submission of information required for obtaining

marketing approval anywhere in the world in respect of a patented product without the consent of the patentee.

This provision allows the generic producers to market and manufacture their goods before the expiration of the term of the patent. Bolar Provision has been upheld as conforming to the TRIPS agreement and is used in several countries to advance science and technology.

- Exception on Foreign Vessels, Aircraft or Land Vehicles

Section 49 the Indian Patents Act deals with the said exception, when the foreign vessels, aircraft, or land vehicles accidentally or temporarily comes to India, the patent rights are not infringed when the patented invention is used exclusively for the needs of foreign vessels, aircraft, or land vehicles and other accessories.
Conclusion

Patenting in India has protected the intellectual property of many innovators and has been useful in the growth of commerce and technology in India. One has to go through a certain process for the grant of Patent. Grant of Patent confers monopolistic rights upon the Patentee excluding the third party to sell, use, manufacture or import of the patented product without the consent of the patentee. If someone tries to use, sell, manufacture or import such patented products leading to the infringement of the rights of the patentee, the patentee can sue the person.

Can you revoke an assignment?

It can be revoked by an assignor later assigning the same right (the last assignment controls), the death or incapacity of the assignor, or by the delivery of notification of revocation to the assignee or obligor. Example: I verbally assign to you my rights to receive payment under a contract.

What is the meaning of equitable assignment?

An assignment which does not fulfil the statutory criteria for a legal assignment. An equitable assignment may be made in one of two ways: The assignor can inform the assignee that he transfers a right or rights to him.

Patent revocation means **cancellation of the rights granted to a person by the grant of a patent**. A patent can be revoked on petition of any person interested or of the Central Government or on a counter claim in a suit for infringement of the patent by the High Court.

Revocation of Patents in India

The term "Revocation" means **Cancellation** of the patent rights acquired to patentee. The revocation of **patent** can be applied by making a petition by any person interested or it can be also applied by the Central Government. In most of the cases revocation are filed on a counter claim against patent infringement suits in High Court or in IPAB (the Intellectual Property Appellate Board). The provision for Revocation of patents is mentioned under section 104.

Section 104 of the Patents Act, 1970 states that only IPAB or the High Court can be approached for revocation as no suit of infringement can be brought before a court inferior to the District Court having jurisdiction.

Section 64: Grounds for Revocation

Principally, Section 64 contains in-exhaustive grounds that dictate the conditions that warrant the revocation of patents. These grounds are following:

1. Invention is obvious, lacks an inventive step or utility
2. Invention isn't new and, has been publicly used or published in India before the priority date or it is foreseen in light of the knowledge available within any local or native community in India or elsewhere.
3. Either the party wasn't entitled to the patent, or the subject isn't patentable or doesn't amount to invention
4. The scope of patent specificationsis incomplete or the specifications have either been already claimed in a patent that is granted
5. The patent was wrongfully obtained in violation of another party's rights, such as through incorrect or false representation, or leave to modify specifications was obtained through fraudulent means
6. The information that has been disclosed under Section 8 is known to be false by the Applicant or he has been unable to furnish the required details
7. Complete specification omits or erroneously attributes geographical origin or biological matter used in the invention
8. The invention was either secretly used before the date or claim or the Applicant contravened secrecy instructions under Section 35
9. The complete specification neither describes the invention and method sufficiently nor does it disclose the best method of performing it which was known and entitled protection.

Other provisions for revocation of patents

A revocation petition can be filed undersection 65 if the patent granted relates to atomic energy.

If the Central Government find the facts that a granted patent has been exercising by wrong means and it is mischievous to the State or prejudicial to the public, the central government has rights in the case to revoke the concerned patent and such decision is also published in official journal of patents.

Moreover, to prevent the wastage of judicial machinery, it was laid down by the Supreme Court in the ***Enercon (India) Ltd and Ors. v. Enercon Gmbh*****,** that post grant opposition proceedings and petitions or counter-claims of revocation against the same patent, cannot be simultaneously instituted. Frivolous litigation shouldn't be encouraged as it is viewed as a tool for cash-rich litigants with dishonest interests.

Revocation of patent under Section 85 of the Patents Act

An application for revocation of patent can be filed after 2 years of grant of compulsory license. The grounds for revocation of a patent are same which are covered under Section 85 which mainly deal with following:-

- reasonable price,
- availability in the territory of India, and
- requirements of the public not met.

Section 85 reveals provision in the benefit of public. In these cases where the patentee does not take necessary actions that may help in better distribution of the product to the public and more profound availability at reasonable prices. Such revocations of patent are really useful to public.

Equitable Assignment: Everything You Need to Know

*An equitable assignment is one that does not fulfill the statutory criteria for a legal assignment, but is binding and upheld by the courts in the interest of equability, justice, and fairness.*3 min read

An equitable assignment is one that does not fulfill the statutory criteria for a legal assignment, but is binding and upheld by the courts in the interest of equability, justice, and fairness.

Equitable Assignment

An equitable assignment may not appear to be self-evident by the law's standard, but it presents the assignee with a title that is protected and recognized in equity. It's based on the essence of a declaration of trust; specifically, essential fairness and natural justice. As long as there is valuable consideration involved, it does not matter if a formal agreement is signed. There needs to be some sort of intent displayed from one party to assign and the other party to receive.

The evaluation of a righteous equitable assignment is completed by determining if a debtor would rationally pay the debt to another party alleging to be the assignee. Equitable assignments can be created by:

1. The assignor informing the assignee that they transferred a right to them
2. The assignor instructing the other party to release their obligation from the assignee and place it instead on the assignor

The only part of an agreement that can be assigned is the benefit. Generally speaking, there is no prerequisite for the written notice to be received or given. The significant characteristic that separates an equitable assignment from a legal assignment is that most of the time, an equitable assignee may not take action against a third party. Instead, it must rely on the guidelines governing equitable assignments. In other words, the equitable assignee must team up with the assignor to take action.

The Doctrine of Equitable Assignment in Wisconsin

In Dow Family LLC v. PHH Mortgage Corp., the Wisconsin Supreme Court issued in favor of the doctrine of equitable assignment. The case was similar to many other foreclosure cases, except this one came with a twist. Essentially, Dow Family LLC purchased a property and the property owner insisted the mortgage on the property had been paid off. However, in actuality, it wasn't.

Prior to the sale, the mortgage on the property was with PHH Mortgage Corp. When PHH went to foreclose on the mortgage, Dow Family LLC contested it. There was one specific rebuttal that caught the attention of the Wisconsin Supreme Court. The official mortgage on record was with MERS, an appointee for the original lender, U.S. Bank.

Dow argued that PHH couldn't foreclose on the property because the true owner was MERS. Essentially, Dow was stating that the mortgage was never assigned to PHH. Based on this argument, PHH utilized the doctrine of equitable assignment.

Based on a case from 1859, Croft v. Bunster, the court determined that the security for a note is equitably assigned when the note is assigned without a need for an independent, written assignment. Additionally, Dow contended that the statute of frauds prohibits the utilization of the doctrine, mainly because it claimed every assignment on a property must be formally recorded.

During the case, Dow argued that the MERS system, which stored the data regarding the mortgage, was fundamentally flawed. According to the court, the statute of frauds was satisfied because the equitable assignment was in accordance with the operation of law. Most importantly, the court avoided all consideration regarding the MERS system, concluding it was not significant in their decision.
The outcome was a major win for lenders, as they were relying on the doctrine specifically for these types of circumstances.

Most experts agree that this outcome makes sense in the current mortgage-lending environment. This is due to the fact that it is still quite common for mortgages to be bundled up into mortgage-backed securities and sold on the secondary market.
Many economists claim that by not requiring mortgages to be recorded each time a transfer is completed, the loans are more easily marketed to investors. Additionally, debtors know who their current mortgage company is because the new lender must always notify the current borrower in order to receive payment. It was determined that recording and documenting the mortgage merely provides a signal to the rest of the world that the property owner secures a debt.

RM4151 RESEARCH METHODOLOGY AND IPR

N.Vinodh, Department of Management Studies

UNIT IV

INTELLECTUAL PROPERTY RIGHTS

Intellectual Property – The concept of IPR, Evolution and development of concept of IPR, IPR development process, Trade secrets, utility Models, IPR & Bio diversity, Role of WIPO and WTO in IPR establishments, Right of Property, Common rules of IPR practices, Types and Features of IPR Agreement, Trademark, Functions of UNESCO in IPR maintenance.

Intellectual Property

Intellectual Property (IP) deals with any basic construction of human intelligence such as artistic, literary, technical or scientific constructions. Intellectual Property Rights (IPR) refers to the legal rights granted to the inventor or manufacturer to protect their invention or manufacture product. These legal rights confer an exclusive right on the inventor/manufacturer or its operator who makes full use of it's his invention/product for a limited period of time.

INTELLECTUAL PROPERTY RIGHTS

The intellectual property right is a kind of legal right that protects a person's artistic works, literary works, inventions or discoveries or a symbol or design for a specific period of time. Intellectual property owners are given certain rights by which they can enjoy their Property without any disturbances and prevent others from using them, although these rights are also called monopoly rights of exploitation, they are limited in geographical range, time and scope.

Nature of intellectual Property

- Intangible Rights over Tangible Property: The main Property that distinguishes IP from other forms of Property is its intangibility. While there are many important differences between different forms of IP, one factor they share is that they establish property protection over intangible things such as ideas, inventions, signs and information whereas intangible assets and close relationships are a tangible object. In which they are embedded. It allows creators or owners to benefit from their works when they are used commercially.
- Right to sue: In the language of the law, IP is an asset that can be owned and dealt with. Most forms of IP are contested in rights of action that are enforced only by legal action and by those who have rights. IP is a property right and can, therefore, be inherited, bought, gifted, sold, licensed, entrusted or pledged. The holder of an IPR owner has a type of Property that he can use the way he likes subject to certain conditions and takes legal action against the person who without his consent used his invention and can receive compensation against real Property.
- Rights and Duties: IP gives rise not only to property rights but also duties. The owner of the IP has the right to perform certain functions in relation to his work/product. He has the exclusive right to produce the work, make copies of the work, market work, etc. There is also a negative right to prevent third parties from exercising their statutory rights.
- Coexistence of different rights: Different types of IPRs can co-exist in relation to a particular function. For example, an invention may be patented, and the invention photograph may be copyrighted. A design can be protected under the Design Act, and the design can also be incorporated into a

trademark. There are many similarities and differences between the various rights that can exist together in IP. For example, there are common grounds between patent and industrial design; Copyright and neighboring rights, trademarks and geographical indications, and so on. Some intellectual property rights are positive rights; the rest of them are negative rights.

- Exhaustion of rights: Intellectual property rights are generally subject to the doctrine of exhaustion. Exhaustion basically means that after the first sale by the right holder or by its exhaustion authority, his right ceases and he is not entitled to stop further movement of the goods. Thus, once an IP rights holder has sold a physical product to which IPRs are attached, it cannot prevent subsequent resale of that product. The right terminates with the first consent. This principle is based on the concept of free movement of goods which is in force by consent or right of the rights holder. The exclusive right to sell goods cannot be exercised twice in relation to the same goods. The right to restrict further movements has expired as the right holder has already earned his share by the act of placing goods for the first sale in the market.
- Dynamism: IPR is in the process of continuous development. As technology is rapidly evolving in all areas of human activities, the field of IP is also growing. As per the requirement of scientific and technological progress, new items are being added to the scope of IPR, and the scope of its preservation is being expanded. Bio Patents, Software Copyrights, Plant Diversity Protection, these are few names which reflect contemporary developments in the field of IPR. The importance of intellectual property and its mobility is well established and reflected at all levels, including statutory, administrative and judicial.

Scope of intellectual Property

The scope of IP rights is broad; two classification modes are used to determine whether IP is copyright or Industrial Property. Industrial properties include patents or inventions, trademarks, trade names, biodiversity, plant breeding rights and other commercial interests. A patent gives its holder the exclusive right to use the Intellectual Property for the purposes of making money from the invention.

An invention is itself a new creation, process, machine or manufacture. Having copyright does not give you the exclusive right to an idea, but it protects the expression of ideas that are different from a patent. Copyright covers many fields, from art and literature to scientific works and software.

Even music and audio-visual works are covered by copyright laws. The duration of copyright protection exists 60 years after the death of the creator. In other words, an author's book is copyrighted for his entire life and then 60 years after his death. Unlike patent laws, there is no requirement of the administrative process in copyright laws.

Why promote and protect Intellectual Property?

There are several reasons for promoting and protecting intellectual property. Some of them are:

1. Progress and the good of humanity remain in the ability to create and invent new works in the field of technology and culture.
2. IP protection encourages publication, distribution, and disclosure of the creation to the public, rather than keeping it a secret.
3. Promotion and protection of intellectual Property promote economic development, generates new jobs and industries, and improves the quality of life.

Intellectual Property helps in balancing between the innovator's interests and public interest, provide an environment where innovation, creativity and invention can flourish and benefit all.

Kinds of intellectual Property

The subject of intellectual property is very broad. There are many different forms of rights that together make up intellectual property. IP can be basically divided into two categories, that is, industrial Property and intellectual property. Traditionally, many IPRs were collectively known as industrial assets.

It mainly consisted of patents, trademarks, and designs. Now, the protection of industrial property extends to utility models, service marks, trade names, passes, signs of source or origin, including geographical indications, and the suppression of unfair competition. It can be said that the term 'industrial property" is the predecessor of 'intellectual property".

Copyright

Copyright law deals with the protection and exploitation of the expression of ideas in a tangible form. Copyright has evolved over many centuries with respect to changing ideas about creativity and new means of communication and media. In the modern world, the law of copyright provides not only a legal framework for the protection of the traditional beneficiaries of copyright, the individual writer, composer or artist, but also the publication required for the creation of work by major cultural industries, film; Broadcast and recording industry; And computer and software industries.

It resides in literary, dramatic, musical and artistic works in "original' cinematic films, and in sound recordings set in a concrete medium. To be protected as the copyright, the idea must be expressed in original form. Copyright acknowledges both the economic and moral rights of the owner. The right to copyright is, by the principle of fair use, a privilege for others, without the copyright owner's permission to use copyrighted material. By the application of the doctrine of fair use, the law of copyright balances private and public interests.

Patent

Patent law recognizes the exclusive right of a patent holder to derive commercial benefits from his invention. A patent is a special right granted to the owner of an invention to the manufacture, use, and markets the invention, provided that the invention meets certain conditions laid down in law. Exclusive right means that no person can manufacture, use, or market an invention without the consent of the patent holder. This exclusive right to patent is for a limited time only.

To qualify for patent protection, an invention must fall within the scope of the patentable subject and satisfy the three statutory requirements of innovation, inventive step, and industrial application. As long as the patent applicant is the first to invent the claimed invention, the novelty and necessity are by and large satisfied. Novelty can be inferred by prior publication or prior use. Mere discovery 'can't be considered as an invention. Patents are not allowed for any idea or principle.

The purpose of patent law is to encourage scientific research, new technology, and industrial progress. The economic value of patent information is that it provides technical information to the industry that can be used for commercial purposes. If there is no protection, then there may be enough incentive to take a free ride at

another person's investment. This ability of free-riding reduces the incentive to invent something new because the inventor may not feel motivated to invent due to lack of incentives.

Trademark

A trademark is a badge of origin. It is a specific sign used to make the source of goods and services public in relation to goods and services and to distinguish goods and services from other entities. This establishes a link between the proprietor and the product. It portrays the nature and quality of a product. The essential function of a trademark is to indicate the origin of the goods to which it is attached or in relation to which it is used. It identifies the product, guarantees quality and helps advertise the product. The trademark is also the objective symbol of goodwill that a business has created.

Any sign or any combination thereof, capable of distinguishing the goods or services of another undertaking, is capable of creating a trademark. It can be a combination of a name, word, phrase, logo, symbol, design, image, shape, color, personal name, letter, number, figurative element and color, as well as any combination representing a graph. Trademark registration may be indefinitely renewable.

Geographical indication

It is a name or sign used on certain products which corresponds to a geographic location or origin of the product, the use of geographical location may act as a certification that the product possesses certain qualities as per the traditional method. Darjeeling tea and basmati rice are a common example of geographical indication. The relationship between objects and place becomes so well known that any reference to that place is reminiscent of goods originating there and vice versa.

It performs three functions. First, they identify the goods as origin of a particular region or that region or locality; Secondly, they suggest to consumers that goods come from a region where a given quality, reputation, or other characteristics of the goods are essentially attributed to their geographic origin, and third, they promote the goods of producers of a particular region. They suggest the consumer that the goods come from this area where a given quality, reputation or other characteristics of goods are essentially attributable to the geographic region.

It is necessary that the product obtains its qualities and reputation from that place. Since those properties depend on the geographic location of production, a specific link exists between the products and the place of origin. Geographical Indications are protected under the Geographical Indication of Goods (Registration and Protection) Act, 1999.

Industrial design

It is one of the forms of IPR that protects the visual design of the object which is not purely utilized. It consists of the creation of features of shape, configuration, pattern, ornamentation or composition of lines or colours applied to any article in two or three-dimensional form or combination of one or more features. Design protection deals with the outer appearance of an article, including decoration, lines, colours, shape, texture and materials. It may consist of three-dimensional features such as colours, shapes and shape of an article or two-dimensional features such as shapes or surface textures or other combinations.

Plant variety

A new variety of plant breeder is protected by the State. To be eligible for plant diversity protection, diversity must be novel, distinct and similar to existing varieties and its essential characteristics under the Plant Protection and Protection Act, 2001 should be uniform and stable. A plant breeder is given a license or special right to do the following in relation to different types of promotional material:

1. Produce and reproduce the material
2. Condition the material for the purpose of propagation
3. Offer material for sale
4. Sell the materials
5. Export the materials
6. Import the materials
7. The stock of goods for the above purposes

Typically, countries are protecting new plant varieties through the Sui Genis system. The general purpose of conservation is to encourage those who intend to manufacture, finance, or exploit such products to serve their purpose, particularly where they otherwise do not work at all.

The enactment of the Protection of Plant Varieties and 'Farmers' Rights Act 2001 is an outcome of the India's obligation which arose from article 27(3)(b) of the TRIPs Agreement of 2001 which obliges members to protect plant varieties either by patents or by effective sui generic system or by any combination thereof India declined to protect plant varieties by a sui generis law, i.e. the Plant Varieties Act.

The Concept of Intellectual Property

Intellectual property, very broadly, means the legal property which results from intellectual activity in the industrial, scientific and artistic fields. Countries have laws to protect intellectual property for two main reasons. One is to give statutory expression to the moral and economic rights of creators in their creations and such rights of the public in access to those creations. The second is to promote, as a deliberate act of government policy, creativity and the dissemination and application of its results and to encourage fair trading which would contribute to economic and social development.

Generally speaking, IP law aims at safeguarding creators and other producers of intellectual goods and services by granting them certain time- limited rights to control the use made of those productions. These rights do not apply to the physical object in which the creation may be embodied but instead to the intellectual creation as such. IP is traditionally divided into two branches: "industrial property and copyright". The convention establishing the World Intellectual Property Organization (WIPO), concluded in Stockholm on July 14, 1967 (Art. 2(viii) provides that

"Intellectual property shall include rights relating to:

1) Literary, artistic and scientific works:

2) Performances of performing artists, phonograms and broadcasts;

3) Inventions in all fields of human behavior;

4) Scientific discoveries;

5) Industrial designs;

6) Trademarks, service marks, and commercial names and designations;

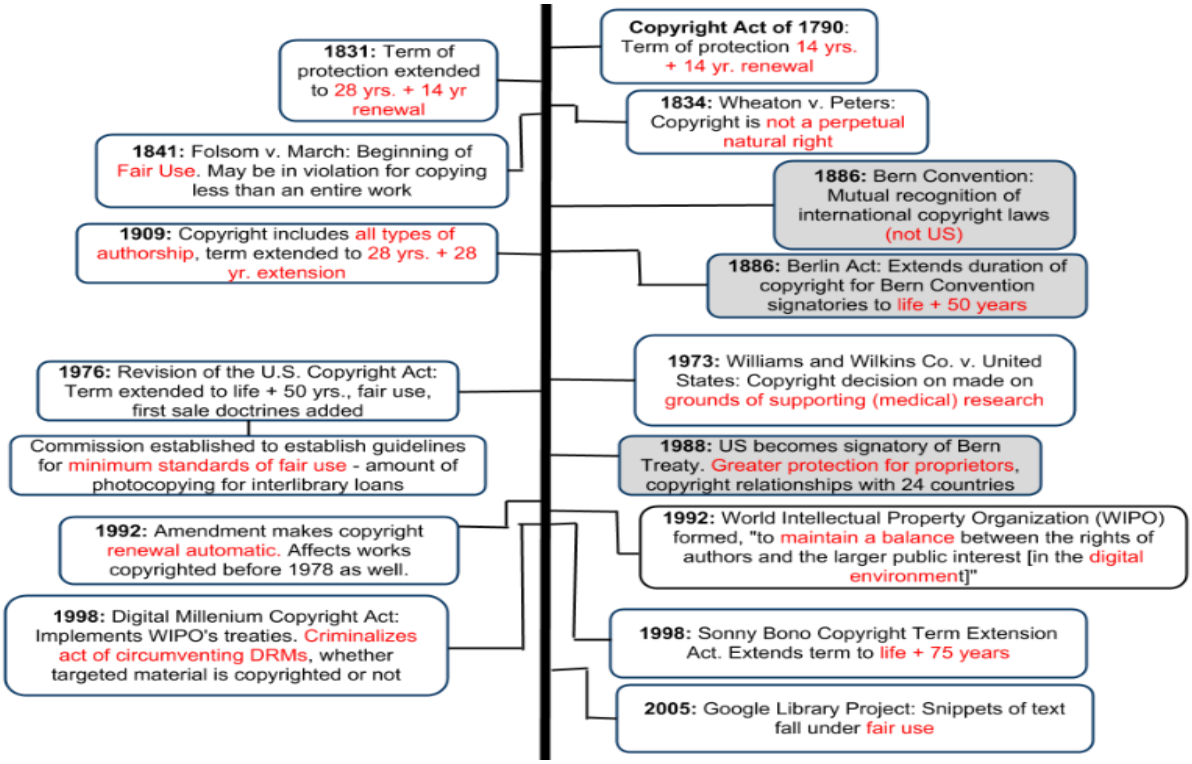
7) Protection against unfair competition and all other rights resulting from intellectual activity in industrial scientific, literary or artistic fields".

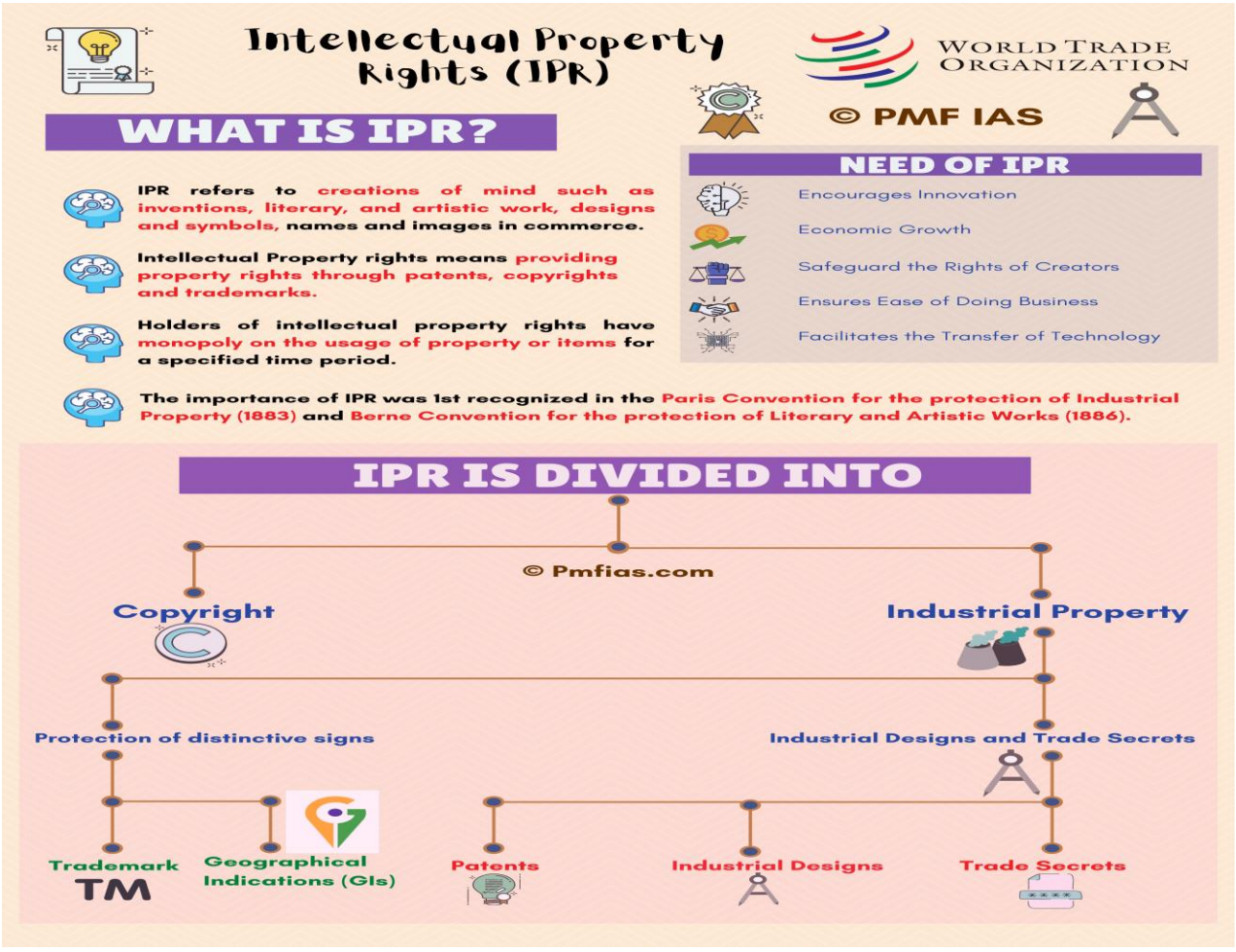
The areas mentioned under
(1) Belong to the copyright branch of intellectual property. The areas mentioned in
(2) Are usually called "neighboring rights", that is, rights neighboring on copyright?

The areas mentioned under 3, 5 and 6 constitute the industrial property branch of IP. The areas mentioned may also be considered as belonging to that branch.

The expression industrial property covers inventions and industrial designs. Simply stated, inventions are new solutions to technical problems, and industrial designs are aesthetic creations determining the appearance of industrial products. In addition, industrial property includes trademarks, service marks, commercial names and designations, including indications of source and appellations of origin, and protection against unfair competition. Hence the aspect of intellectual creations -although existent -is less prominent, but what counts here is that the object of industrial property typically consists of signs transmitting information to consumers, in particular, as regards products and services offered on the market, and that the protection is directed against unauthorized use of such signs which is likely to mislead consumers and misleading practices in general.

Evolution of IPR





TYPES OF IPR

Patent

- A Patent is granted for Invention which is a new product or process.
- Law: Patents Act, 1970, amended in 2006
- Ministry: DPIIT, Ministry of Commerce and industry
- Period: 20 Years

Trademark

TM

- A trademark is typically a name, word, phrase, logo, symbol, design, image, or a combination of these elements.
- Law: Trademark Act 1999
- Ministry: DPIIT, Ministry of Commerce and industry
- Period: 10 Years

Geographical Indication

- A geographical indication (GI) is a sign used on products that have a specific geographical origin and possess qualities or a reputation that are due to that origin.
- Law: Geographical Indications of Goods (Registration and Protection Act), 1999
- Ministry: DPIIT, Ministry of Commerce and industry
- Period: 10 Years

What Is a Trade Secret?

A trade secret is any practice or process of a company that is generally not known outside of the company. Information considered a trade secret gives the company a competitive advantage over its competitors and is often a product of internal research and development.

To be legally considered a trade secret in the United States, a company must make a reasonable effort in concealing the information from the public; the secret must intrinsically have economic value, and the trade secret must contain information. Trade secrets are a part of a company's intellectual property. Unlike a patent, a trade secret is not publicly known.

What are examples of trade secrets?

Examples of trade secrets can include engineering information; methods, processes, and know-how; tolerances and formulas; business and financial information; computer programs (particularly source code) and related information; pending, unpublished patent applications; business plans; budgets;

What is a utility model?

Similar to patents, utility models **protect new technical inventions through granting a limited exclusive right to prevent others from commercially exploiting the protected inventions without consents of the right holders**.

What is biological diversity in IPR?

"Biological Diversity" means **the variability among living organisms from**. **all sources and the ecological complexes of which they are part and includes**. **diversity within species or between species and of eco-systems** "Biological resources" means plants, animals and microorganisms or parts.

The WTO and World Intellectual Property Organization

The link between the WTO and the World Intellectual Property Organization (WIPO) is deeply rooted in the multilateral trading system. During the Uruguay Round, negotiators sought to connect the two institutions. The Preamble to the TRIPS Agreement encapsulates this connection by calling on the two organizations to establish a mutually supportive relationship. Further, the TRIPS Agreement legally requires Members to abide by certain rules of key conventions administered by WIPO

1995 WIPO-WTO Cooperation Agreement

The complementarily between the two organizations was further strengthened by the WIPO-WTO Cooperation Agreement. The Agreement covers transparency mechanisms, technical assistance and training and the implementation of a provision of WIPO's Paris Convention on state emblems.

The WIPO is an observer to the TRIPS Council, and the WTO enjoys observer status at the WIPO.

Joint Technical Assistance

Joint technical assistance and training activities, based, inter alia, on the joint initiatives of 1998 and 2001, aim to build the capacity of developing countries and LDCs in implementing the TRIPS Agreement, to enhance an understanding and participation in the global IP system and to help foster IP law and policy decision-making that coheres with broader public policy objectives.

The two flagship Geneva-based joint courses, the IP Advanced Course for policymakers and the Colloquium for IP teachers, are intensive programmes that give detailed and multi-stakeholder insight into the global IP system. The organizations also jointly publish the IP Colloquium Research Paper series – a collection of papers on IP issues directly relevant to developing countries.

WIPO's two main objectives are

(i) to promote the protection of intellectual property worldwide; and

(ii) to ensure administrative cooperation among the intellectual property Unions established by the treaties that WIPO administers.

In order to attain these objectives, WIPO, in addition to performing the administrative tasks of the Unions, undertakes a number of activities, including:

(i) normative activities, involving the setting of norms and standards for the protection and enforcement of intellectual property rights through the conclusion of international treaties;

(ii) program activities, involving legal and technical assistance to States in the field of intellectual property;

(iii) international classification and standardization activities, involving cooperation among industrial property offices concerning patent, trademark and industrial design documentation; and

(iv) Registration and filing activities, involving services related to international applications for patents for inventions and for the registration of marks and industrial designs.

Common rules of IPR

The concepts that underpin the protection of ideas and inventions are not new; such laws have been around for several hundred years and are discussed under the broad heading of intellectual property (IP). IP is easily misunderstood, but at the same time most scientists encounter it at some point in their career, as it is a necessary feature in the commercialization of research.

The term intellectual property includes such concepts and rights as copyright, trademarks, industrial design rights, and patents. It is important to remember that IP is a tool to help your endeavours, and not a goal in itself. Having IP for its own sake is pointless. IP can be crucial in commercializing research and running a successful science-based business, but having a patent and having a successful patented product are two very different things.

Above all, IP can only work for you if you understand what it is, why you want it, and what you are going to do with it. These ten simple rules are intended to provide an overview of these issues; however, we must start with a warning. Laws relating to IP change all the time, they are complex, sometimes rather obscure, and are very different from country to country. For example, research surrounding methods of treatment by surgery and therapy and diagnostic methods are patentable in the United States, but specifically excluded from patentability in Europe [1]. However, these boundaries seem to be shifting in both the US and Europe. In short, we are dealing with a complex and changing subject and restrict ourselves here to the guiding principles.

Rule 1: Get Professional Help

Although the process of obtaining IP looks deceptively simple, like many things the devil is in the detail. Let's consider patents as an example. The practicalities of patent application are straightforward; you simply file documents with the relevant body indicating that a patent is sought, and provide the identity of the person applying and a description of the "invention" for which a patent is sought. The patent office will then write back to you with an application number.

However, there is no guarantee that a patent application will become a granted patent. Indeed, at the application stage they do not even check that your description describes an invention at all. Even if you draft a description in as much detail as you would for an academic research paper and file it yourself, the prospect that it will be granted and enforceable is very low. There is skill and technique, even a language, that patent attorneys and patent agents have that allows them to describe and define inventions in the way a patent office requires. As an example, in everyday parlance, the terms "comprise" and "consist" could be considered to mean the same, whereas they have very distinct meanings in a patent application.

The dangers are possibly even greater with trademarks and registered designs (also known as "design patents")—these are generally granted with very little examination and patent offices are often even less inclined to suggest using a patent/trademark attorney for such "simpler" rights; however, the lack of examination means the validity of such a right is uncertain and they become open to challenge.
The costs of redrafting a self-filed application are invariably higher than the costs for drafting an application from scratch, and if there has been any disclosure it will probably not be possible to re-draft. So, in summary, if you want your IP to be valuable, you should seek professional advice at an early stage.

Rule 2: Know Your (Intellectual Property) Rights

IP rights come in various guises, and each is a defensive right to pursue legal action in the event that a third party infringes. In very basic terms:

- Patents protect inventions—broadly, things that are new and not obvious—and the way they work. Sometimes this is expressed as "everything under the sun made by man"; however, there are numerous local exceptions from patentability—we touched on the complexities of methods of treatment above—but there are similar issues in relation to genes, computer programs, and business methods, for example.
- Registered designs protect the appearance of products (not the function, which is protected by patents).
- Trademarks protect brands (e.g., trade names and logos).
- Copyright protects the expression of ideas—i.e., the words you choose to use to describe your idea—not an idea itself.

Most businesses do not need the trinity of patents, trademarks, and designs; in fact, trademarks are probably the only IP most companies have or need, however for a few companies the full house is required: for example, consider the Apple® iPad®: two registered trademarks, a registered design for its shape, and of course patents for the way it interacts with the user. Not to mention copyright covering the code and the packaging. A huge battle in the courts around the world is currently taking place over these rights that may well effect changes in the law. The *Wall Street Journal* calls the recent Apple/Samsung case "the patent trial of the century".

Rule 3: Think about Why You Want IP (i.e., What You Will Actually Do with It)

Any money spent on IP is capital that cannot be spent on production, marketing, etc., so think carefully about why you are investing in protecting your IP. There are many good reasons: to stop people from copying you; to add value to your company if you want to sell it; to sell or license to a third party; to hold it in your armory if you suspect you are going to be sued and want to countersue (for example, Google has spent a substantial amount of money buying patents recently; even to reduce your tax bill (in certain countries profits attributed to patents can be taxed at a lower rate).

However, in general, IP is a right to prevent other people from doing something; owning IP does not necessarily give you the right to do anything yourself.

One school of thought says that IP is only valuable if you are willing to enforce or defend it, and the cost of such an action can be prohibitive. Indeed, the business model of "patent trolls" is to purchase patents, sometimes from those who cannot afford to enforce them, not to use the invention, but just to enforce against infringing companies. On the other hand, the term "defensive IP" has been used to describe IP obtained, not to stop other people from competing, but to stop a competitor from patenting something that you may wish to use in the future. Thus a patent application may be filed, and published but allowed to lapse, with no intention of ever enforcing it, simply because the step of publication will mean that should a competitor apply to patent the same or a similar invention, the patent office will locate your application and it will anticipate the competitor's application.

Note also that while this article is titled "Ten Simple Rules to Protect *Your* IP", it is important not to be too introspective and to consider other people's IP. For example, successful strategies can be built around taking exclusive licenses—licenses that exclude even the IP owner from using the IP. One tactic to improve your competitive position can be to take an exclusive license under a patent, then either expand your range to include the patented product, or continue only to sell your own product, but use the exclusive license to prevent manufacture of the other by anybody else.

Rule 4: If You Don't Protect the IP, Your Innovation Is Less Likely to Happen

Maybe you are not an entrepreneur yourself, but have an idea that you would like to see it exploited—it could, after all, make the world a better place. You can publish it—then anyone who wishes can use it freely. But the big question here is, will they? Many inventors think that by publishing their ideas freely they are more likely to have them exploited; however, the converse is often true (for example, in health care, where lack of patent protection is often cited as a major reason for not following up an idea (T. Roberts, former president of the Chartered Institute of Patent Attorneys [UK]).

The reason is economic: most innovations require investment, and investors look for a return on their money. However, ideas that are released without any IP protection will often immediately attract competitors who can perhaps undercut the inventor (for example, with economies of scale). This decreases the likelihood of investment in the development of an invention (which is often more crucial than the invention itself) and increases the need for investment in marketing, etc. to obtain a competitive edge.

So what we have to consider here is that—even if you don't want to profit personally from the innovation—it may still pay to protect it so that it will see the light of day through other investors. Remember, IP can be licensed and what happens to the resulting income is up to the IP's owner. And this is a point where it gets

complex for scientists and others who invent as part of their employment. We will cover this in more detail in Rule 10.

Rule 5: What's in a Name?

You have a great idea but it's not patentable, or you have applied for patent protection but are worried that it may not cover everything, and of course the protection will expire after 20 years [5]. This is where trademarks come in to fill the gap in your protection. Unlike patents and designs, a trademark or brand can be protected with a registration at any time (unless someone else has got there before you)—you do not need to have kept your name a secret, and once registered the right will only expire if you stop using it or fail to renew it (generally every 10 years). So, you can protect your invention with a patent and sell it under your brand, which is also protected. Once the patent protection expires, customers are used to buying your product with reference to your brand, and will hopefully continue to do so even though competitors may start offering rival products. Just make sure your brand is something memorable and unique to you.

Viagra is just one example of a trademark so closely associated with the product (sildenafil) that a good proportion of the market should remain in the hands of the trademark owner well after the patent has expired (in this instance, if priced competitively). You do need to be careful here in selecting the name you are protecting: descriptive brands are easy to market but hard to protect because descriptive terms do not fulfil the requirement of “distinct character”. And you can be too successful: many people now use the trademark Hoover to mean a generic vacuum cleaner, Thermos for a vacuum flask to keep food hot, or Tannoy for a public address system. It can be very expensive in terms of lawyers fees to police such trademarks and keep protecting these names and prevent them becoming simply part of the language and hence devalued.

Rule 6: Be Realistic about What You Can, and Cannot, Protect

IP rights are, generally speaking, national rights provided by individual governments to regulate activity in that particular country. In some cases there are bilateral and multilateral agreements (for example, most of the world has signed up to the Berne Agreement, which accords the same level of copyright protection to foreign nationals of other Berne states that is provided to nationals of the state concerned [6]).
However, for most rights, it is a national issue. In an ideal world, each incremental improvement would be patented in each national jurisdiction (there are approximately 200 countries in the world), along with the name you trade under, and every brand would be the subject of a trademark, as would any color associated with your company and any sound you use, your products and their packaging would be the subject of registered designs, and your patent attorneys would be very wealthy!

In the real world it is essential to be realistic. A patenting regime covering more than the US, Europe, and a handful of other countries is a rare sight outside the realms of very large companies (such as big pharma), and even many big companies restrict themselves to key markets.

Rule 7: It's Big Business and Controversial

The world of IP is a big one. It's controversial, as it has a huge impact on international relations and trade. It's also controversial for political reasons, as many people feel that aggressive protection stifles the utility of products that have the potential to do good in the emerging world (again, for example, big pharma). The World Intellectual Property Organization (WIPO) is the United Nations agency dedicated to this area [7], and

it's worth considering its overarching aims, which include reducing the knowledge gap between developed and developing countries, and ensuring that the IP system continues to effectively serve its fundamental purpose of encouraging creativity and innovation in all countries.

Of course, many question the value to society of IP, or at least the expansion of IP, in promoting creativity and innovation. The Public Library of Science describes itself as a driving force of the open-access movement, and accordingly, unlike many copyrighted works, this article may be copied without seeking permission, provided that the original authors and source are cited.

It can be hard, for example, to defend the extension of copyright from 50 years after an author's death to 70 years on the grounds that the extra 20 years of protection is in any way likely to encourage creativity. Whatever your thoughts on IP, it is worth bearing in mind that others may disagree.

As a scientist and innovator you may be driven by many ideals: to make the world a better place, perhaps, or to buy yourself a yacht—we are all different. But like it or not, if you want to commercialize your ideas you cannot avoid the issue of IP, and we go back to Rule 1 here—get professional advice. Even if your aim is totally philanthropic you may still need to invest to protect your innovation, perversely because this is what will give it the biggest chance of actually succeeding. Simply make sure you tell your patent attorney what your ultimate aims are.

Rule 8: Keep Your Idea Secret until You Have Filed a Patent Application

Little upsets a patent attorney more than hearing “I have a great idea—it's selling really well” or “I've shown it to a few companies and they seem very interested”.

There is an old maxim that says a secret shared is not a secret anymore. While a secret shared under a non-disclosure agreement (NDA)—documents most people have heard about but probably never read—ought to stay secret, discussing an invention under the umbrella of confidentiality is no substitute for being able to freely discuss or publish an idea that is protected by a patent application.

Obviously, once your idea is published by a journal it is too late to file a patent application—your invention has been made available to the public. However, earlier in the publication cycle the situation is different. If you send a paper to a journal for submission, it will (excluding open review) be treated as a confidential disclosure to the publisher and the reviewers. Notwithstanding, the best advice is still to file a patent application before submitting a paper, either to avoid a potential “abusive disclosure” or hold up the publication of the paper.

In summary, novelty is key to patentability and your own disclosures count against you, so remember to file a patent application *before* telling anybody who is not bound by confidence.

Rule 9: Trade Secrets

Regarding patents, the economic reasoning behind the system is an exchange between you and the public. The government allows you a monopoly, and your side of the bargain is to disclose fully your invention so that once your 20 years of protection is up, it can be freely exploited for the good of society. A patent can provide you with a 20-year government approved monopoly. However, some ideas cannot be patented and indeed, some innovators don't want to patent their ideas. All is not lost here, however, as we fall back on an older idea and one much beloved of thriller writers: the trade secret.

If you really can keep a secret, your monopoly on an idea or product may never end. But once the genie's out of the bottle, like a champagne cork, you won't get it back in and you are unlikely to extract sufficient damages from whoever breaches confidentiality. Thus, if you have an idea that cannot be reverse engineered, you do not have to enter into the patent bargain. Trade secrets are free—just prevent the secret being disclosed. But bear in mind that that this can be very difficult indeed, but not impossible. Famous successful examples include the recipe for Coca-Cola and the formulation of the alcoholic beverage Chartreuse, which is only known by two monks.

Rule 10: Make Sure the IP Is Owned in a Way That Allows Development

Notice that we don't suggest "make sure *you* own the IP of your invention". If you discover something whilst working as an employee (e.g., of a company or an academic establishment), there will certainly be something in your contract about this. Generally, the employer will have first call on the invention, but may have clauses that will return rights to the individual if it is not exploited within a certain time—in some countries this is enshrined in law.

Ownership of IP is a minefield, and can be particularly difficult in an academic setting where numerous complicating features are involved. Universities, as employers, are likely to have a right to their employees' inventions; funding bodies may make their own claim; inventorship is not like authorship—the people whose names are on an academic paper are unlikely all to be inventors; and in cross-border collaborations, national laws on ownership may well be in competition with each other. One complicating factor that is often encountered is joint ownership: if you can, avoid joint ownership; instead, set up a company to own the IP and license it to partners if necessary (otherwise you face differing national rules on what joint owners can do with and without each other's permission).

If it is necessary to share IP, work out at the beginning who owns what, what rights each party has and importantly who will have the right to future inventions. In fact this is a common theme in several of our Ten Simple Rules: as soon as money rears it ugly head, strife follows, so it's as well to plan for dispute resolution right from the beginning.

In summary, first, you can never act too early, but it's very easy to act too late. Like many topics that involve the law, IP is a mind-numbingly complex topic and more so, perhaps, as it's not national, but international, so get the very best professional advice you can. If you are working as an employee, speak to your company at the earliest stage; they have a vested interest in helping get it right. Second, because significant sums of money are involved, plan for future discord. Finally, persevere: your invention can make the world a better place.

Types and Features of IPR Agreement

The areas of intellectual property that it covers are: copyright and related rights (i.e. the rights of performers, producers of sound recordings and broadcasting organizations); trademarks including service marks; geographical indications including appellations of origin; industrial designs; patents including the protection of new varieties of plants; the layout-designs of integrated circuits; and undisclosed information including trade secrets and test data.

The three main features of the Agreement are:

- Standards. In respect of each of the main areas of intellectual property covered by the TRIPS Agreement, the Agreement sets out the minimum standards of protection to be provided by each Member. Each of the main elements of protection is defined, namely the subject-matter to be protected, the rights to be conferred and permissible exceptions to those rights, and the minimum duration of protection. The Agreement sets these standards by requiring, first, that the substantive obligations of the main conventions of the WIPO, the Paris Convention for the Protection of Industrial Property (Paris Convention) and the Berne Convention for the Protection of Literary and Artistic Works (Berne Convention) in their most recent versions, must be complied with. With the exception of the provisions of the Berne Convention on moral rights, all the main substantive provisions of these conventions are incorporated by reference and thus become obligations under the TRIPS Agreement between TRIPS Member countries. The relevant provisions are to be found in Articles 2.1 and 9.1 of the TRIPS Agreement, which relate, respectively, to the Paris Convention and to the Berne Convention. Secondly, the TRIPS Agreement adds a substantial number of additional obligations on matters where the pre-existing conventions are silent or were seen as being inadequate. The TRIPS Agreement is thus sometimes referred to as a Berne and Paris-plus agreement.
- Enforcement. The second main set of provisions deals with domestic procedures and remedies for the enforcement of intellectual property rights. The Agreement lays down certain general principles applicable to all IPR enforcement procedures. In addition, it contains provisions on civil and administrative procedures and remedies, provisional measures, special requirements related to border measures and criminal procedures, which specify, in a certain amount of detail, the procedures and remedies that must be available so that right holders can effectively enforce their rights.
- Dispute settlement. The Agreement makes disputes between WTO Members about the respect of the TRIPS obligations subject to the WTO's dispute settlement procedures.

In addition the Agreement provides for certain basic principles, such as national and most-favored-nation treatment, and some general rules to ensure that procedural difficulties in acquiring or maintaining IPRs do not nullify the substantive benefits that should flow from the Agreement. The obligations under the Agreement will apply equally to all Member countries, but developing countries will have a longer period to phase them in. Special transition arrangements operate in the situation where a developing country does not presently provide product patent protection in the area of pharmaceuticals.

The TRIPS Agreement is a minimum standards agreement, which allows Members to provide more extensive protection of intellectual property if they so wish. Members are left free to determine the appropriate method of implementing the provisions of the Agreement within their own legal system and practice

REFERENCE

FOUR TYPES OF INTELLECTUAL PROPERTY PROTECTIONS

There are four types of intellectual property rights and protections (although multiple types of intellectual property itself). Securing the correct protection for your property is important, which is why consulting with a lawyer is a must. The four categories of intellectual property protections include:

TRADE SECRETS

Trade secrets refer to specific, private information that is important to a business because it gives the business a competitive advantage in its marketplace. If a trade secret is acquired by another company, it could harm the original holder.

Examples of trade secrets include recipes for certain foods and beverages (like Mrs. Fields' cookies or Sprite), new inventions, software, processes, and even different marketing strategies.

When a person or business holds a trade secret protection, others cannot copy or steal the idea. In order to establish information as a "trade secret," and to incur the legal protections associated with trade secrets, businesses must actively behave in a manner that demonstrates their desire to protect the information.

Trade secrets are protected *without* official registration; however, an owner of a trade secret whose rights are breached–i.e. someone steals their trade secret–may ask a court to ask against that individual and prevent them from using the trade secret.

PATENTS

As defined by the U.S. Patent and Trademark Office (USPTO), a patent is a type of limited-duration protection that can be used to protect inventions (or discoveries) that are new, non-obvious, and useful, such a new process, machine, article of manufacture, or composition of matter.

When a property owner holds a patent, others are prevented, under law, from offering for sale, making, or using the product.

COPYRIGHTS

Copyrights and patents are not the same things, although they are often confused. A copyright is a type of intellectual property protection that protects original works of *authorship*, which might include literary works, music, art, and more. Today, copyrights also protect computer software and architecture.

Copyright protections are *automatic*; once you create something, it is yours. However, if your rights under copyright protections are infringed and you wish to file a lawsuit, then registration of your copyright will be necessary.

TRADEMARKS

Finally, the fourth type of intellectual property protection is a trademark protection. Remember, patents are used to protect inventions and discoveries and copyrights are used to protect expressions of ideas and creations, like art and writing.

Trademarks, then, refer to phrases, words, or symbols that distinguish the source of a product or services of one party from another. For example, the Nike symbol–which nearly all could easily recognize and identify–is a type of trademark.

While patents and copyrights can expire, trademark rights come from the use of the trademark, and therefore can be held indefinitely. Like a copyright, registration of a trademark is not required, but registering can offer additional advantages.

UNESCO: United Nations Educational, Scientific and Cultural Organization

The main functions of UNESCO are to ensure that every child has access to a proper education, promoting cultural acceptance between nations while protecting historical sites, improving technology to aid in the distribution of resources and energy, and secure the safety of individual expression and basic human rights.

The United Nations Educational, Scientific and Cultural Organization (UNESCO) was born on 16 November 1945. UNESCO has 195 Members and 8 Associate Members and is governed by the General Conference and the Executive Board. The Secretariat, headed by the Director-General, implements the decisions of these two bodies. The Organization has more than 50 field offices around the world and its headquarters are located in Paris.

UNESCO's mission is to contribute to the building of a culture of peace, the eradication of poverty, sustainable development and intercultural dialogue through education, the sciences, culture, communication and information.

UNESCO works to create the conditions for dialogue among civilizations, cultures and peoples, based upon respect for commonly shared values. It is through this dialogue that the world can achieve global visions of sustainable development encompassing observance of human rights, mutual respect and the alleviation of poverty, all of which are at the heart of UNESCO's mission and activities.

UNESCO focuses on a set of objectives in the global priority areas "Africa" and "Gender Equality"

And on a number of overarching objectives:

- Attaining quality education for all and lifelong learning
- Mobilizing science knowledge and policy for sustainable development
- Addressing emerging social and ethical challenges
- Fostering cultural diversity, intercultural dialogue and a culture of peace
- Building inclusive knowledge societies through information and communication

MADHA Engineering College, Kundrathur, Chennai-69.
Master of Computer Application
N.Vinodh, MBA, M.Phil, Department of Management Studies
Unit-2 Research Methodology & IPR

Measurements, Measurement Scales, Questionnaires and Instruments, Sampling and methods. Data - Preparing, Exploring, examining and displaying.

Measurement:

Measurement is the process of observing and recording the observations that are collected as part of a research effort. There are two major issues that will be considered here.

First, to understand the **fundamental ideas** involved in measuring. Here we consider two of major measurement concepts. In **Levels of Measurement**, the meaning of the four major levels of measurement: nominal, ordinal, interval and ratio. Then we move on to the **reliability** of measurement, including consideration of true score theory and a variety of reliability estimators.

Second, to understand the different **types of measures** that you might use in social research. We consider four broad categories of measurements. **Survey research** includes the design and implementation of interviews and questionnaires. **Scaling** involves consideration of the major methods of developing and implementing a scale. **Qualitative research** provides an overview of the broad range of non-numerical measurement approaches. And **unobtrusive measures** presents a variety of measurement methods that don't intrude on or interfere with the context of the research.

LEVELS OF MEASUREMENT

There are different levels of measurement. These levels differ as to how closely they approach the structure of the number system we use. It is important to understand the level of measurement of variables in research, because the level of measurement determines the type of statistical analysis that can be conducted, and, therefore, the type of conclusions that can be drawn from the research.

Nominal Level

A nominal level of measurement uses symbols to classify observations into categories that must be both mutually exclusive and exhaustive. Exhaustive means that there must be enough categories that all the observations will fall into some category. Mutually exclusive means that the categories must be distinct enough that no observations will fall

into more than one category. This is the most basic level of measurement; it is essentially labeling. It can only establish whether two observations are alike or different, for example, sorting a deck of cards into two piles: red cards and black cards.

In a survey of boaters, one variable of interest was place of residence. It was measured by a question on a questionnaire asking for the zip code of the boater's principal place of residence. The observations were divided into zip code categories. These categories are mutually exclusive and exhaustive. All respondents live in one zip code category (exhaustive) but no boater lives in more than one zip code category (mutually exclusive). Similarly, the sex of the boater was determined by a question on the questionnaire. Observations were sorted into two mutually exclusive and exhaustive categories, male and female. Observations could be labeled with the letters M and F, or the numerals 0 and 1.

The variable of marital status may be measured by two categories, married and unmarried. But these must each be defined so that all possible observations will fit into one category but no more than one: legally married, common-law marriage, religious marriage, civil marriage, living together, never married, divorced, informally separated, legally separated, widowed, abandoned, annulled, etc.
In nominal measurement, all observations in one category are alike on some property, and they differ from the objects in the other category (or categories) on that property (e.g., zip code, sex). There is no ordering of categories (no category is better or worse, or more or less than another).

Ordinal Level
An ordinal level of measurement uses symbols to classify observations into categories that are not only mutually exclusive and exhaustive; in addition, the categories have some explicit relationship among them.

For example, observations may be classified into categories such as taller and shorter, greater and lesser, faster and slower, harder and easier, and so forth. However, each observation must still fall into one of the categories (the categories are exhaustive) but no more than one (the categories are mutually exclusive). Meats are categorized as regular, choice, or prime; the military uses ranks to distinguish categories of soldiers.

Most of the commonly used questions which ask about job satisfaction use the ordinal level of measurement. For example, asking whether one is very satisfied, satisfied,

neutral, dissatisfied, or very dissatisfied with one's job is using an ordinal scale of measurement.

Interval Level

An interval level of measurement classifies observations into categories that are not only mutually exclusive and exhaustive, and have some explicit relationship among them, but the relationship between the categories is known and exact. This is the first quantitative application of numbers.

In the interval level, a common and constant unit of measurement has been established between the categories. For example, the commonly used measures of temperature are interval level scales. We know that a temperature of 75 degrees is one degree warmer than a temperature of 74 degrees, just as a temperature of 42 degrees is one degree warmer than a temperature of 41 degrees.

Numbers may be assigned to the observations because the relationship between the categories is assumed to be the same as the relationship between numbers in the number system. For example, 74+1=75 and 41+1=42.

The intervals between categories are equal, but they originate from some arbitrary origin. that is, there is no meaningful zero point on an interval scale.

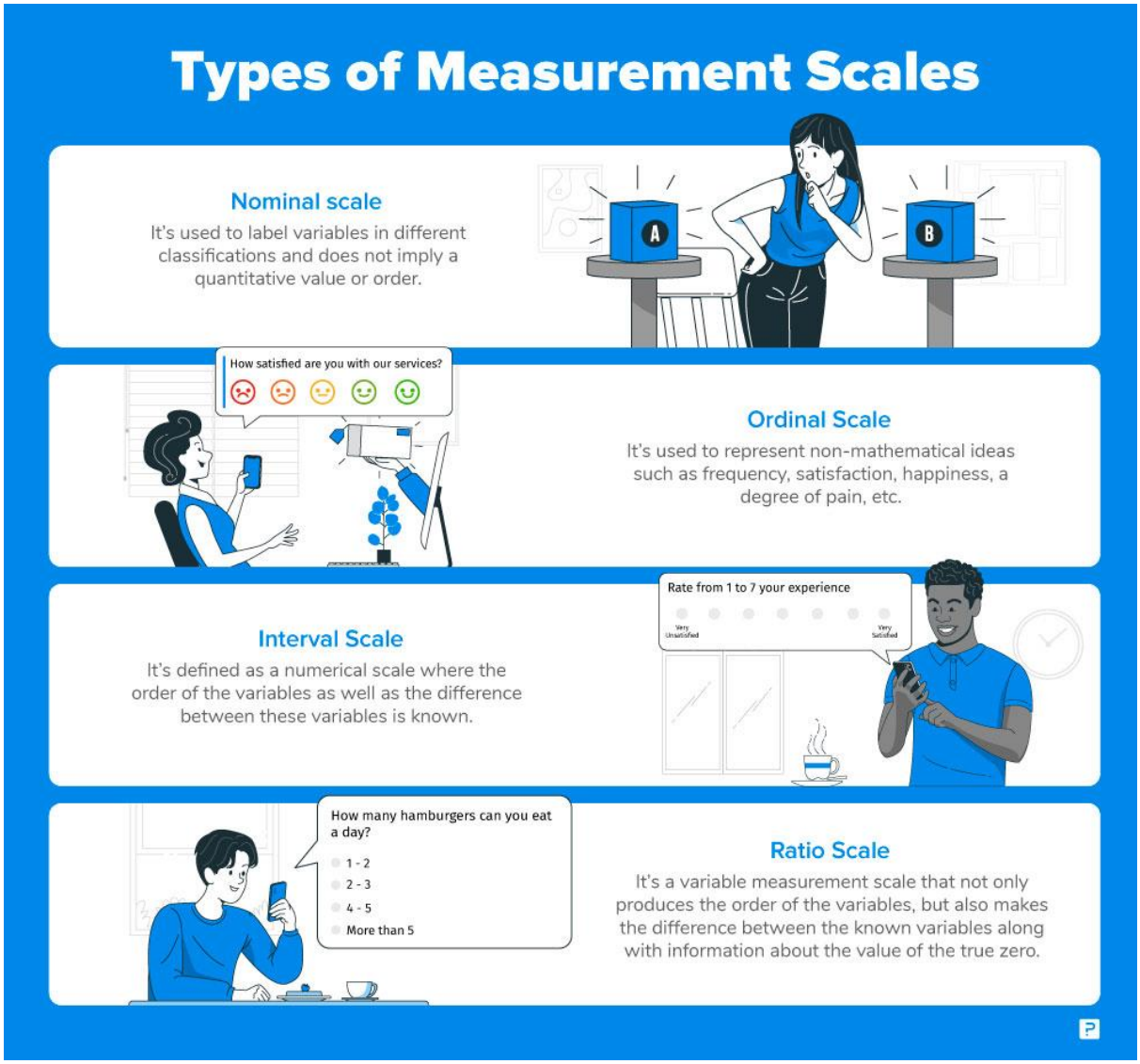
Ratio Level

The ratio level of measurement is the same as the interval level, with the addition of a meaningful zero point. There is a meaningful and non-arbitrary zero point from which the equal intervals between categories originate.

For example, weight, area, speed, and velocity are measured on a ratio level scale. In public policy and administration, budgets and the number of program participants are measured on ratio scales.

In many cases, interval and ratio scales are treated alike in terms of the statistical tests that are applied.

Variables measured at a higher level can always be converted to a lower level, but not vice versa. For example, observations of actual age (ratio scale) can be converted to categories of older and younger (ordinal scale), but age measured as simply older or younger cannot be converted to measures of actual age.



Questionaries & Instruments:

A questionnaire is a research tool featuring a series of questions used to collect useful information from respondents. These instruments include either written or oral questions and comprise an interview-style format. Questionnaires may be qualitative or quantitative and can be conducted online, by phone, on paper or face-to-face, and questions don't necessarily have to be administered with a researcher present.

Questionnaires feature either open or closed questions and sometimes employ a mixture of both. Open-ended questions enable respondents to answer in their own words in as much or as little detail as they desire. Closed questions provide respondents with a series of predetermined responses they can choose from.

Is a Questionnaire Just Another Word for "Survey"?

While the two terms seem synonymous, there are not quite the same. A questionnaire is a set of questions created for the purpose of gathering information; that information may not be used for a survey. However, all surveys *do* require questionnaires. If you are using

a questionnaire for survey sampling, it's important to ensure that it is designed to gather the most accurate answers from respondents.

Why Are Questionnaires Effective in Research?

Questionnaires are popular research methods because they offer a fast, efficient and inexpensive means of gathering large amounts of information from sizeable sample volumes. These tools are particularly effective for measuring subject behavior, preferences, intentions, attitudes and opinions. Their use of open and closed research questions enables researchers to obtain both qualitative and quantitative data, resulting in more comprehensive results.

Advantages of Questionnaires

Some of the many benefits of using questionnaires as a research tool include:

- **Practicality:** Questionnaires enable researchers to strategically manage their target audience, questions and format while gathering large data quantities on any subject.
- **Cost-efficiency:** You don't need to hire surveyors to deliver your survey questions — instead, you can place them on your website or email them to respondents at little to no cost.
- **Speed:** You can gather survey results quickly and effortlessly using mobile tools, obtaining responses and insights in 24 hours or less.
- **Comparability:** Researchers can use the same questionnaire yearly and compare and contrast research results to gain valuable insights and minimize translation errors.
- **Scalability:** Questionnaires are highly scalable, allowing researchers to distribute them to demographics anywhere across the globe.
- **Standardization:** You can standardize your questionnaire with as many questions as you want about any topic.
- **Respondent comfort:** When taking a questionnaire, respondents are completely anonymous and not subject to stressful time constraints, helping them feel relaxed and encouraging them to provide truthful responses.
- **Easy analysis:** Questionnaires often have built-in tools that automate analyses, making it fast and easy to interpret your results.

Disadvantages of Questionnaires

Questionnaires also have their disadvantages, such as:

- **Answer dishonesty:** Respondents may not always be completely truthful with their answers — some may have hidden agendas, while others may answer how they think society would deem most acceptable.

- **Question skipping:** Make sure to require answers for all your survey questions. Otherwise, you may run the risk of respondents leaving questions unanswered.
- **Interpretation difficulties:** If a question isn't straightforward enough, respondents may struggle to interpret it accurately. That's why it's important to state questions clearly and concisely, with explanations when necessary.
- **Survey fatigue:** Respondents may experience survey fatigue if they receive too many surveys or a questionnaire is too long.
- **Analysis challenges:** Though closed questions are easy to analyze, open questions require a human to review and interpret them. Try limiting open-ended questions in your survey to gain more quantifiable data you can evaluate and utilize more quickly.
- **Unconscientious responses:** If respondents don't read your questions thoroughly or completely, they may offer inaccurate answers that can impact data validity. You can minimize this risk by making questions as short and simple as possible.

Types of Questionnaires in Research

There are various types of questionnaires in survey research, including:

- **Postal:** Postal questionnaires are paper surveys that participants receive through the mail. Once respondents complete the survey, they mail them back to the organization that sent them.
- **In-house:** In this type of questionnaire, researchers visit respondents in their homes or workplaces and administer the survey in person.
- **Telephone:** With telephone surveys, researchers call respondents and conduct the questionnaire over the phone.
- **Electronic:** Perhaps the most common type of questionnaire, electronic surveys are presented via email or through a different online medium.

What are Research Instruments?

A research instrument is a tool used to collect, measure, and analyze data related to your subject.

Research instruments can be **tests**, **surveys**, **scales**, **questionnaires**, or even **checklists**.

A research instrument is a tool used to obtain, measure, and analyze data from subjects around the research topic.

To decide the instrument to use based on the type of study you are conducting: quantitative, qualitative, or mixed-method. For instance, for a quantitative study, you may decide to use a questionnaire, and for a qualitative study, you may choose to use a scale.

While it helps to use an established instrument, as its efficacy is already established, you may if needed use a new instrument or even create your own instrument.

What are the Different Types of Interview Research Instruments?

The general format of an interview is where the interviewer asks the interviewee to answer a set of questions which are normally asked and answered verbally. There are several different types of interview research instruments that may exist.

1. A structural interview may be used in which there are a specific number of questions that are formally asked of the interviewee and their responses recorded using a systematic and standard methodology.
2. An unstructured interview on the other hand may still be based on the same general theme of questions but here the person asking the questions (the interviewer) may change the order the questions are asked in and the specific way in which they're asked.
3. A focus interview is one in which the interviewer will adapt their line or content of questioning based on the responses from the interviewee.
4. A focus group interview is one in which a group of volunteers or interviewees are asked questions to understand their opinion or thoughts on a specific subject.
5. A non-directive interview is one in which there are no specific questions agreed upon but instead the format is open-ended and more reactionary in the discussion between interviewer and interviewee.

What is sampling?

Sampling is a technique of selecting individual members or a subset of the population to make statistical inferences from them and estimate characteristics of the whole population. Different sampling methods are widely used by researchers in market research so that they do not need to research the entire population to collect actionable insights.

It is also a time-convenient and a cost-effective method and hence forms the basis of any research design. Sampling techniques can be used in a research survey software for optimum derivation.

For example, if a drug manufacturer would like to research the adverse side effects of a drug on the country's population, it is almost impossible to conduct a research study that involves everyone. In this case, the researcher decides a sample of people from each demographic and then researches them, giving him/her indicative feedback on the drug's behavior.

Types of sampling: sampling methods

Sampling in market research is of two types – probability sampling and non-probability sampling. Let's take a closer look at these two methods of sampling.

1. **Probability sampling:** Probability sampling is a sampling technique where a researcher sets a selection of a few criteria and chooses members of a population randomly. All the members have an equal opportunity to be a part of the sample with this selection parameter.
2. **Non-probability sampling:** In non-probability sampling, the researcher chooses members for research at random. This sampling method is not a fixed or predefined selection process. This makes it difficult for all elements of a population to have equal opportunities to be included in a sample.

In this blog, we discuss the various probability and non-probability sampling methods that you can implement in any market research study.

Types of probability sampling with examples:

Probability sampling is a sampling technique in which researchers choose samples from a larger population using a method based on the theory of probability. This sampling method considers every member of the population and forms samples based on a fixed process.

For example, in a population of 1000 members, every member will have a 1/1000 chance of being selected to be a part of a sample. Probability sampling eliminates bias in the population and gives all members a fair chance to be included in the sample.

There are four types of probability sampling techniques:

- **Simple random sampling:** One of the best probability sampling techniques that helps in saving time and resources, is the Simple Random Sampling method. It is a reliable method of obtaining information where every single member of a population is chosen randomly, merely by chance. Each individual has the same probability of being chosen to be a part of a sample.

- For example, in an organization of 500 employees, if the HR team decides on conducting team building activities, it is highly likely that they would prefer picking chits out of a bowl. In this case, each of the 500 employees has an equal opportunity of being selected.

- **Cluster sampling:** Cluster sampling is a method where the researchers divide the entire population into sections or clusters that represent a population. Clusters are identified and included in a sample based on demographic parameters like age, sex, location, etc. This makes it very simple for a survey creator to derive effective inference from the feedback.

  For example, if the United States government wishes to evaluate the number of immigrants living in the Mainland US, they can divide it into clusters based on states such as California, Texas, Florida, Massachusetts, Colorado, Hawaii, etc. This way of conducting a survey will be more effective as the results will be organized into states and provide insightful immigration data.

- **Systematic sampling:** Researchers use the systematic sampling method to choose the sample members of a population at regular intervals. It requires the selection of a starting point for the sample and sample size that can be repeated at regular intervals. This type of sampling method has a predefined range, and hence this sampling technique is the least time-consuming.
  For example, a researcher intends to collect a systematic sample of 500 people in a population of 5000. He/she numbers each element of the population from 1-5000 and will choose every 10th individual to be a part of the sample (Total population/ Sample Size = 5000/500 = 10).

- **Stratified random sampling:** Stratified random sampling is a method in which the researcher divides the population into smaller groups that don't overlap but represent the entire population. While sampling, these groups can be organized and then draw a sample from each group separately.
  For example, a researcher looking to analyze the characteristics of people belonging to different annual income divisions will create strata (groups) according to the annual family income. Eg – less than $20,000, $21,000 – $30,000, $31,000 to $40,000, $41,000 to $50,000, etc. By doing this, the researcher concludes the characteristics of people belonging to different income groups. Marketers can analyze which income groups to target and which ones to eliminate to create a roadmap that would bear fruitful results.

Uses of probability sampling

There are multiple uses of probability sampling:

- **Reduce Sample Bias:** Using the probability sampling method, the bias in the sample derived from a population is negligible to non-existent. The selection of the sample mainly depicts the understanding and the inference of the researcher. Probability sampling leads to higher quality data collection as the sample appropriately represents the population.

- **Diverse Population:** When the population is vast and diverse, it is essential to have adequate representation so that the data is not skewed towards one demographic. For example, if Square would like to understand the people that could make their point-of-sale devices, a survey conducted from a sample of people across the US from different industries and socio-economic backgrounds helps.

- **Create an Accurate Sample:** Probability sampling helps the researchers plan and create an accurate sample. This helps to obtain well-defined data.

Types of non-probability sampling with examples

The non-probability method is a sampling method that involves a collection of feedback based on a researcher or statistician's sample selection capabilities and not on a fixed selection process. In most situations, the output of a survey conducted with a non-probable sample leads to skewed results, which may not represent the desired target population. But, there are situations such as the preliminary stages of research or cost constraints for conducting research, where non-probability sampling will be much more useful than the other type.

Four types of non-probability sampling explain the purpose of this sampling method in a better manner:

- **Convenience sampling:** This method is dependent on the ease of access to subjects such as surveying customers at a mall or passers-by on a busy street. It is usually termed as convenience sampling, because of the researcher's ease of carrying it out and getting in touch with the subjects. Researchers have nearly no authority to select the sample elements, and it's purely done based on proximity and not representativeness. This non-probability sampling method is used when there are time and cost limitations in collecting feedback. In situations where there are resource limitations such as the initial stages of research, convenience sampling is used.

For example, startups and NGOs usually conduct convenience sampling at a mall to distribute leaflets of upcoming events or promotion of a cause – they do that by standing at the mall entrance and giving out pamphlets randomly.

- **Judgmental or purposive sampling:** Judgemental or purposive samples are formed by the discretion of the researcher. Researchers purely consider the purpose of the study, along with the understanding of the target audience. For instance, when researchers want to understand the thought process of people interested in studying for their master's degree. The selection criteria will be: "Are you interested in doing your masters in …?" and those who respond with a "No" are excluded from the sample.

- **Snowball sampling:** Snowball sampling is a sampling method that researchers apply when the subjects are difficult to trace. For example, it will be extremely challenging to survey shelterless people or illegal immigrants. In such cases, using the snowball theory, researchers can track a few categories to interview and derive results. Researchers also implement this sampling method in situations where the topic is highly sensitive and not openly discussed—for example, surveys to gather information about HIV Aids. Not many victims will readily respond to the questions. Still, researchers can contact people they might know or volunteers associated with the cause to get in touch with the victims and collect information.

- **Quota sampling:** In Quota sampling, the selection of members in this sampling technique happens based on a pre-set standard. In this case, as a sample is formed based on specific attributes, the created sample will have the same qualities found in the total population. It is a rapid method of collecting samples.

Uses of non-probability sampling

Non-probability sampling is used for the following:

- **Create a hypothesis:** Researchers use the non-probability sampling method to create an assumption when limited to no prior information is available. This method helps with the immediate return of data and builds a base for further research.

- **Exploratory research:** Researchers use this sampling technique widely when conducting qualitative research, pilot studies, or exploratory research.

- **Budget and time constraints:** The non-probability method when there are budget and time constraints, and some preliminary data must be collected. Since the survey design is not rigid, it is easier to pick respondents at random and have them take the survey or questionnaire.

How do you decide on the type of sampling to use?

For any research, it is essential to choose a sampling method accurately to meet the goals of your study. The effectiveness of your sampling relies on various factors. Here are some steps expert researchers follow to decide the best sampling method.

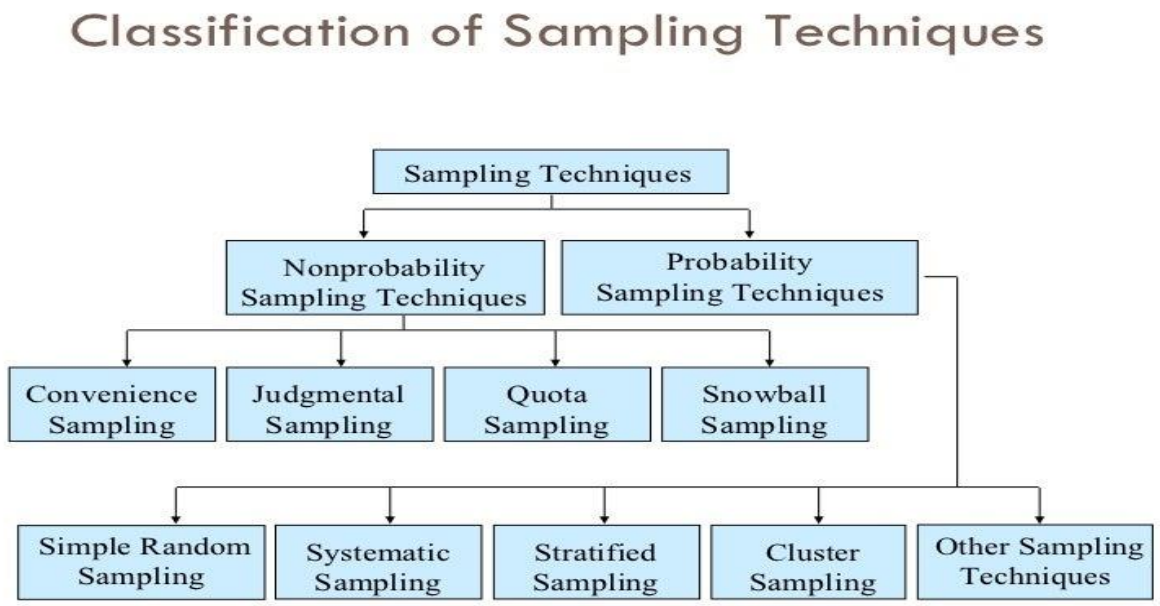
- Jot down the research goals. Generally, it must be a combination of cost, precision, or accuracy.
- Identify the effective sampling techniques that might potentially achieve the research goals.
- Test each of these methods and examine whether they help in achieving your goal.
- Select the method that works best for the research.

Difference between probability sampling and non-probability sampling methods

We have looked at the different types of sampling methods above and their subtypes. To encapsulate the whole discussion, though, the significant differences between probability sampling methods and non-probability sampling methods are as below:

	Probability Sampling Methods	**Non-Probability Sampling Methods**
Definition	Probability Sampling is a sampling technique in which samples from a larger population are chosen using a method based on the theory of probability.	Non-probability sampling is a sampling technique in which the researcher selects samples based on the researcher's subjective judgment rather than random selection.
Alternatively Known as	Random sampling method.	Non-random sampling method
Population selection	The population is selected randomly.	The population is selected arbitrarily.
Nature	The research is conclusive.	The research is exploratory.
Sample	Since there is a method for deciding the sample, the population demographics are conclusively represented.	Since the sampling method is arbitrary, the population demographics representation is almost always skewed.

Time Taken	Takes longer to conduct since the research design defines the selection parameters before the market research study begins.	This type of sampling method is quick since neither the sample or selection criteria of the sample are undefined.
Results	This type of sampling is entirely unbiased and hence the results are unbiased too and conclusive.	This type of sampling is entirely biased and hence the results are biased too, rendering the research speculative.
Hypothesis	In probability sampling, there is an underlying hypothesis before the study begins and the objective of this method is to prove the hypothesis.	In non-probability sampling, the hypothesis is derived after conducting the research study.



Data Preparation Steps

The specifics of the data preparation process vary by industry, organization and need, but the framework remains largely the same.

1. Gather data

The data preparation process begins with finding the right data. This can come from an existing data catalog or can be added ad-hoc.

2. Discover and assess data

After collecting the data, it is important to discover each dataset. This step is about getting to know the data and understanding what has to be done before the data becomes useful in a particular context.

Discovery is a big task, but Talend's data preparation platform offers visualization tools which help users profile and browse their data.

3. Cleanse and validate data

Cleaning up the data is traditionally the most time consuming part of the data preparation process, but it's crucial for removing faulty data and filling in gaps. Important tasks here include:

- Removing extraneous data and outliers.
- Filling in missing values.
- Conforming data to a standardized pattern.
- Masking private or sensitive data entries.

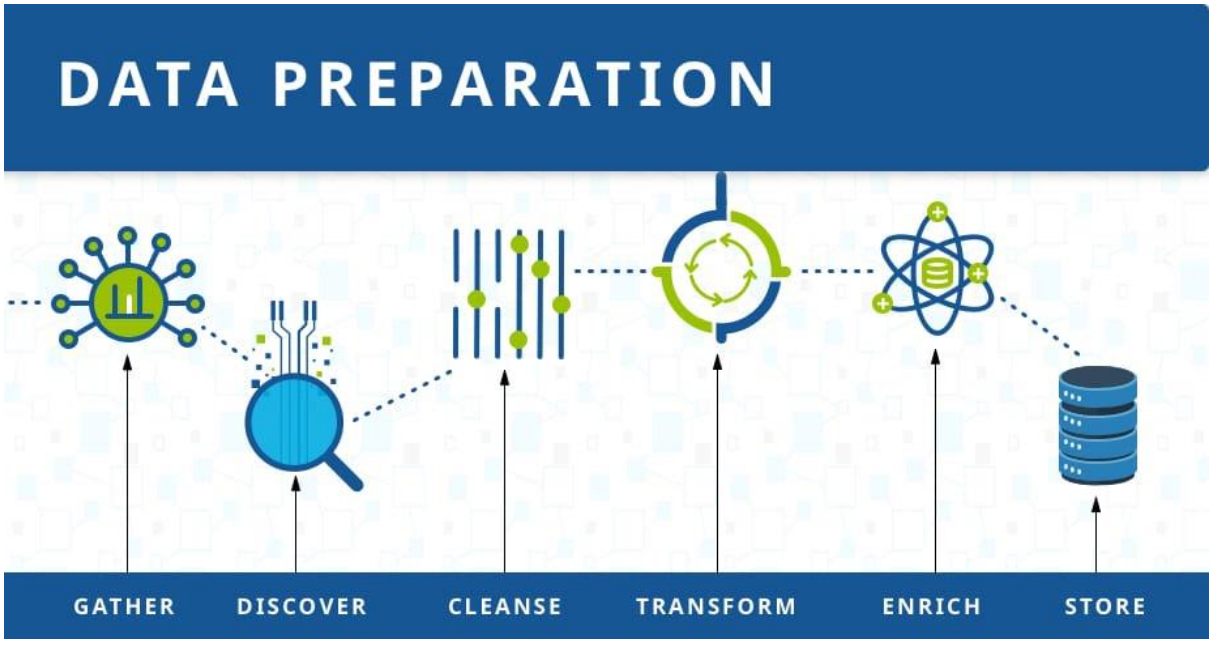
Once data has been cleansed, it must be validated by testing for errors in the data preparation process up to this point. Often times, an error in the system will become apparent during this step and will need to be resolved before moving forward.

4. Transform and enrich data

Transforming data is the process of updating the format or value entries in order to reach a well-defined outcome, or to make the data more easily understood by a wider audience. *Enriching* data refers to adding and connecting data with other related information to provide deeper insights.

5. Store data

Once prepared, the data can be stored or channeled into a third party application—such as a business intelligence tool—clearing the way for processing and analysis to take place.



What is Data Exploration?

Data exploration definition: Data exploration refers to the initial step in data analysis in which data analysts use data visualization and statistical techniques to describe dataset characterizations, such as size, quantity, and accuracy, in order to better understand the nature of the data.

Data exploration techniques include both manual analysis and automated data exploration software solutions that visually explore and identify relationships between different data variables, the structure of the dataset, the presence of outliers, and the distribution of data values in order to reveal patterns and points of interest, enabling data analysts to gain greater insight into the raw data.

Data is often gathered in large, unstructured volumes from various sources and data analysts must first understand and develop a comprehensive view of the data before extracting relevant data for further analysis, such as univariate, bivariate, multivariate, and principal components analysis.

Data Exploration Tools

Manual data exploration methods entail either writing scripts to analyze raw data or manually filtering data into spreadsheets. Automated data exploration tools, such as data visualization software, help data scientists easily monitor data sources and perform big data exploration on otherwise overwhelmingly large datasets. Graphical displays of data, such as bar charts and scatter plots, are valuable tools in visual data exploration.

A popular tool for manual data exploration is Microsoft Excel spreadsheets, which can be used to create basic charts for data exploration, to view raw data, and to identify the correlation between variables. To identify the correlation between two continuous variables in Excel, use the function CORREL() to return the correlation. To identify the correlation between two categorical variables in Excel, the two-way table method, the stacked column chart method, and the chi-square test are effective.

There is a wide variety of proprietary automated data exploration solutions, including business intelligence tools, data visualization software, data preparation software vendors, and data exploration platforms. There are also open source data exploration tools that include regression capabilities and visualization features, which can help businesses integrate diverse data sources to enable faster data exploration. Most data analytics software includes data visualization tools.

Why is Data Exploration Important?

Humans process visual data better than numerical data, therefore it is extremely challenging for data scientists and data analysts to assign meaning to thousands of rows and columns of data points and communicate that meaning without any visual components.

Data visualization in data exploration leverages familiar visual cues such as shapes, dimensions, colors, lines, points, and angles so that data analysts can effectively visualize and define the metadata, and then perform data cleansing. Performing the initial step of data exploration enables data analysts to better understand and visually identify anomalies and relationships that might otherwise go undetected.

What is Data Preparation?

Data preparation is the process of cleaning and transforming raw data prior to processing and analysis. It is an important step prior to processing and often involves reformatting data, making corrections to data and the combining of data sets to enrich data.

Data preparation is often a lengthy undertaking for data professionals or business users, but it is essential as a prerequisite to put data in context in order to turn it into insights and eliminate bias resulting from poor data quality.

For example, the data preparation process usually includes standardizing data formats, enriching source data, and/or removing outliers.

Benefits of Data Preparation + The Cloud

76% of data scientists say that data preparation is the worst part of their job, but the efficient, accurate business decisions can only be made with clean data. Data preparation helps:

- **Fix errors quickly** — Data preparation helps catch errors before processing. After data has been removed from its original source, these errors become more difficult to understand and correct.
- **Produce top-quality data** — Cleaning and reformatting datasets ensures that all data used in analysis will be high quality.

- **Make better business decisions** — Higher quality data that can be processed and analyzed more quickly and efficiently leads to more timely, efficient and high-quality business decisions.
-

Additionally, as data and data processes move to the cloud, data preparation moves with it for even greater benefits, such as:

- **Superior scalability** — Cloud data preparation can grow at the pace of the business. Enterprise don't have to worry about the underlying infrastructure or try to anticipate their evolutions.

- **Future proof** — Cloud data preparation upgrades automatically so that new capabilities or problem fixes can be turned on as soon as they are released. This allows organizations to stay ahead of the innovation curve without delays and added costs.

- **Accelerated data usage and collaboration** — Doing data prep in the cloud means it is always on, doesn't require any technical installation, and lets teams collaborate on the work for faster results.

What Is Data Analysis?

Although many groups, organizations, and experts have different ways to approach data analysis, most of them can be distilled into a one-size-fits-all definition. Data analysis is the process of cleaning, changing, and processing raw data, and extracting actionable, relevant information that helps businesses make informed decisions. The procedure helps reduce the risks inherent in decision-making by providing useful insights and statistics, often presented in charts, images, tables, and graphs.

It's not uncommon to hear the term "big data" brought up in discussions about data analysis. Data analysis plays a crucial role in processing big data into useful information. Neophyte data analysts who want to dig deeper by revisiting big data fundamentals should go back to the basic question, "What is data?"

Why is Data Analysis Important?

Here is a list of reasons why data analysis is such a crucial part of doing business today.

- Better Customer Targeting: You don't want to waste your business's precious time, resources, and money putting together advertising campaigns targeted at demographic groups that have little to no interest in the goods and services you

offer. Data analysis helps you see where you should be focusing your advertising efforts.

- You Will Know Your Target Customers Better: Data analysis tracks how well your products and campaigns are performing within your target demographic. Through data analysis, your business can get a better idea of your target audience's spending habits, disposable income, and most likely areas of interest. This data helps businesses set prices, determine the length of ad campaigns, and even help project the quantity of goods needed.

- Reduce Operational Costs: Data analysis shows you which areas in your business need more resources and money, and which areas are not producing and thus should be scaled back or eliminated outright.

- Better Problem-Solving Methods: Informed decisions are more likely to be successful decisions. Data provides businesses with information. You can see where this progression is leading. Data analysis helps businesses make the right choices and avoid costly pitfalls.

- You Get More Accurate Data: If you want to make informed decisions, you need data, but there's more to it. The data in question must be accurate. Data analysis helps businesses acquire relevant, accurate information, suitable for developing future marketing strategies, business plans, and realigning the company's vision or mission.

What Is the Data Analysis Process?

Answering the question "what is data analysis" is only the first step. Now we will look at how it's performed. The data analysis process, or alternately, data analysis steps, involves gathering all the information, processing it, exploring the data, and using it to find patterns and other insights. The process consists of:

- Data Requirement Gathering: Ask yourself why you're doing this analysis, what type of data analysis you want to use, and what data you are planning on analyzing.

- Data Collection: Guided by the requirements you've identified, it's time to collect the data from your sources. Sources include case studies, surveys,

interviews, questionnaires, direct observation, and focus groups. Make sure to organize the collected data for analysis.

- Data Cleaning: Not all of the data you collect will be useful, so it's time to clean it up. This process is where you remove white spaces, duplicate records, and basic errors. Data cleaning is mandatory before sending the information on for analysis.

- Data Analysis: Here is where you use data analysis software and other tools to help you interpret and understand the data and arrive at conclusions. Data analysis tools include Excel, Python, R, Looker, Rapid Miner, Chartio, Metabase, Redash, and Microsoft Power BI.

- Data Interpretation: Now that you have your results, you need to interpret them and come up with the best courses of action, based on your findings.

- Data Visualization: Data visualization is a fancy way of saying, "graphically show your information in a way that people can read and understand it." You can use charts, graphs, maps, bullet points, or a host of other methods. Visualization helps you derive valuable insights by helping you compare datasets and observe relationships.

What Is the Importance of Data Analysis in Research?

A huge part of a researcher's job is to sift through data. That is literally the definition of "research." However, today's Information Age routinely produces a tidal wave of data, enough to overwhelm even the most dedicated researcher.

Data analysis, therefore, plays a key role in distilling this information into a more accurate and relevant form, making it easier for researchers to do to their job.

Data analysis also provides researchers with a vast selection of different tools, such as descriptive statistics, inferential analysis, and quantitative analysis.
So, to sum it up, data analysis offers researchers better data and better ways to analyze and study said data.

What is Data Analysis: Types of Data Analysis

There are a half-dozen popular types of data analysis available today, commonly employed in the worlds of technology and business. They are:

- Diagnostic Analysis: Diagnostic analysis answers the question, "Why did this happen?" Using insights gained from statistical analysis (more on that later!), analysts use diagnostic analysis to identify patterns in data. Ideally, the analysts find similar patterns that existed in the past, and consequently, use those solutions to resolve the present challenges hopefully.

- Predictive Analysis: Predictive analysis answers the question, "What is most likely to happen?" By using patterns found in older data as well as current events, analysts predict future events. While there's no such thing as 100 percent accurate forecasting, the odds improve if the analysts have plenty of detailed information and the discipline to research it thoroughly.

- Prescriptive Analysis: Mix all the insights gained from the other data analysis types, and you have prescriptive analysis. Sometimes, an issue can't be solved solely with one analysis type, and instead requires multiple insights.

- Statistical Analysis: Statistical analysis answers the question, "What happened?" This analysis covers data collection, analysis, modeling, interpretation, and presentation using dashboards. The statistical analysis breaks down into two sub-categories:

1. Descriptive: Descriptive analysis works with either complete or selections of summarized numerical data. It illustrates means and deviations in continuous data and percentages and frequencies in categorical data.

2. Inferential: Inferential analysis works with samples derived from complete data. An analyst can arrive at different conclusions from the same comprehensive data set just by choosing different samplings.

- Text Analysis: Also called "data mining," text analysis uses databases and data mining tools to discover patterns residing in large datasets. It transforms raw data into useful business information. Text analysis is arguably the most straightforward and the most direct method of data analysis.

- Displaying data in research is the last step of the research process. It is important to display data accurately because it helps in presenting the findings of the research effectively to the reader. The purpose of displaying data in research is to make the findings more visible and make comparisons easy. When the researcher will present the research in front of the research committee, they will easily understand the findings of the research from displayed data. The readers of the research will also be able to understand it better. Without displayed data, the data looks too scattered and the reader cannot make inferences.

- There are basically two ways to display data: tables and graphs. The tabulated data and the graphical representation both should be used to give more accurate picture of the research. In quantitative research it is very necessary to display data, on the other hand in qualitative data the researcher decides whether there is a need to display data or not. The researcher can use an appropriate software to help tabulate and display the data in the form of graphs. Microsoft excel is one such example, it is a user-friendly program that you can use to help display the data.

Tables for displaying data in research

- The use of tables to display data is very common in research. Tables are very effective in presenting a large amount of data. They organize data very well and makes the data very visible. A badly tabulated data also occurs, in case, you do not have knowledge of tables and tabulating data consult a statistician to do this step effectively.
- Parts of a table
- To know the tables and to tabulate data in tables you should know the parts or structure of the tables. There are five parts of a tables, namely;
- ***Title***
- The title of the table speaks about the contents of the table. The title should have to be concise and precise, no extra details. The title should be written in sentence case.
- ***Stub***
- The column at the left-most of the table is called as stub. A stub has a stub-heading at the top of the column, not all tables have stub. The stub shows the subcategories that are listed along Y-axis.
- ***Caption***
- The caption is the column heading, the variable might have subcategories which are captioned. These subcategories are provided on the X-axis, the captions are provided on the top of each column.
- ***Body***
- The body of the table is the actual part of the table in which resides the whole values, results, and analysis.
- ***Footnotes***
- There can be many different types of notes that you may have to provide at the end of the table. The footnotes are provided just below the table and labeled as the source. The source generally are provided when the table has been taken from some other source. They are also provided for explaining some point in the table.

Sometimes there is some part of the table that is taken from a source so it should also be mentioned.

- Types of tables
- Tables are the most simple means to display data, they can be categorized into the following;
- Univariate
- Bivariate
- Polyvariate
- These categories are based on the numbers of variables that need to be tabulated in the table. A univariate table has one variable to be tabulated; a bivariate table, as the name suggests, has two variables to be tabulated and a polyvariate table has more than two variables to be tabulated.
- Graphs to display data
- The purpose of displaying data is to make the communications easier. Graphs should be used in displaying data when they can add to the visual beauty of the data. The researcher should decide whether there is a need for table only or he should also present data in the form of a suitable graph.
- Types of graphs
- You can use a suitable graph type depending on the type of data and the variables involved in the data.
- ***The histogram***
- The histogram is a graph that is highly used for displaying data. A histogram consists of rectangles that are drawn next to each other on the graph. The rectangles have no space in between them. A histogram can be drawn for a single variable as well as for two or more than two variables. The height of the bars in the histogram represent the frequency of each variable. It can be drawn for both categorical and continuous variables.
- ***The bar chart***
- The bar chart is similar to a histogram except in that it is drawn only for categorical variables. Since it is used for categorical variables, therefore, it is drawn with space between the rectangles.
- ***The frequency polygon***
- A frequency polygon is also very much like a histogram. A frequency polygon consists of frequency rectangles drawn next to each other but the values taken to draw the rectangles is the midpoint of the values. The height of the rectangles describes the frequency of each interval. A line is drawn that touches the midpoints at the highest frequency level on Y-axis and it touches the X-axis on each extreme end.

- ***The cumulative frequency polygon***
- The cumulative frequency polygon is also a frequency polygon, it is drawn using the cumulative frequencies on the Y-axis. The values on the X-axis are taken by using the endpoints of the interval. The endpoints of the interval are joined to each other the reason being that the cumulative frequency is always based on the upper limit of an interval.
- ***The stem and leaf display***
- The stem and leaf display is another easy way to display data. The stem and leaf display if rotated to 90 degrees become a histogram.
- ***The pie chart***
- The pie chart is a very different way to display data. The pie chart is a circle, as a circle has 360 degrees so it is taken in percentage and the whole pie or circle represent the whole population. The pie or circle is divided into slices or sections, each section represents the magnitude of the category or the sub-category.
- ***The trend curve***
- The trend curve is also called as the line diagram. It is drawn by plotting the midpoints on the X-axis and the frequencies commensurate with each interval on the Y-axis. The trend curve is drawn only for a set of data that has been measured on the continuous, interval or ratio scale. A trend diagram or the line diagram is most suitable for plotting values that show changes over a period of time.
- ***The area chart***
- The area chart is a variation of the trend curve. In area chart, the sub-categories of a variable can be displayed. The categories in the chart are displayed by shading them with different colors or patterns. For example, if there are both males and females category in the dataset both can be highlighted in this chart.
- ***The scattergram***
- A scattergram is a very simple way to plot the data on a chart. The scattergram is used for data where the change in one variable affects the change in the other variable. The frequency against each interval is plotted with the help of dots.

RM4151- Research Methodology and IPR

UNIT III

DATA ANALYSIS AND REPORTING

Overview of Multivariate analysis, Hypotheses testing and Measures of Association. Presenting Insights and findings using written reports and oral presentation.

Introduction:

There are three categories of analysis to be aware of:

- **Univariate analysis**, which looks at just one variable
- **Bivariate analysis**, which analyses two variables
- **Multivariate analysis**, which looks at more than two variables

1. **Univariate data –**

   This type of data consists of only one variable. The analysis of Univariate data is thus the simplest form of analysis since the information deals with only one quantity that changes. It does not deal with causes or relationships and the main purpose of the analysis is to describe the data and find patterns that exist within it. The example of a Univariate data can be height.

2. **Bivariate data –**

   This type of data involves two different variables. The analysis of this type of data deals with causes and relationships and the analysis are done to find out the relationship among the two variables. Example of bivariate data can be temperature and ice cream sales in summer season.

   Suppose the temperature and ice cream sales are the two variables of a bivariate data (figure 2). Here, the relationship is visible from the table that temperature and sales are directly proportional to each other and thus related because as the temperature increases, the sales also increase. Thus bivariate data analysis involves comparisons, relationships, causes and explanations. These variables are often plotted on X and Y axis on the graph for better understanding of data and one of these variables is independent while the other is dependent.

3. **Multivariate data**

   When the data involves three or more variables, it is categorized under multivariate. Example of this type of data is suppose an advertiser wants to compare the popularity of four advertisements on a website, then their click rates could be measured for both men and women and relationships between Variables can then be examined.

It is similar to bivariate but contains more than one dependent variable. The ways to perform analysis on this data depends on the goals to be achieved. Some of the techniques are regression analysis, path analysis, factor analysis and multivariate analysis of variance (MANOVA).

Overview of Multivariate analysis

Multivariate means involving multiple dependent variables resulting in one outcome. This explains that the majority of the problems in the real world are Multivariate. For example, we cannot predict the weather of any year based on the season. There are multiple factors like pollution, humidity, precipitation, etc.

Multivariate analysis (**MVA**) is a Statistical procedure for analysis of data involving more than one type of measurement or observation. It may also mean solving problems where more than one dependent variable is analysed simultaneously with other variables.

1. What is multivariate analysis?

In data analytics, we look at different variables (or factors) and how they might impact certain situations or outcomes. For example, in marketing, you might look at how the variable "money spent on advertising" impacts the variable "number of sales." In the healthcare sector, you might want to explore whether there's a correlation between "weekly hours of exercise" and "cholesterol level." This helps us to understand why certain outcomes occur, which in turn allows us to make informed predictions and decisions for the future.

Advantages and Disadvantages of Multivariate Analysis

Advantages

- The main advantage of multivariate analysis is that since it considers more than one factor of independent variables that influence the variability of dependent variables, the conclusion drawn is more accurate.
- The conclusions are more realistic and nearer to the real-life situation.

Disadvantages

- The main disadvantage of MVA includes that it requires rather complex computations to arrive at a satisfactory conclusion.

- Many observations for a large number of variables need to be collected and tabulated; it is a rather time-consuming process.

Classification Chart of Multivariate Techniques

Selection of the appropriate multivariate technique depends upon-

a) Are the variables divided into independent and dependent classification?

b) If yes, how many variables are treated as dependents in a single analysis?

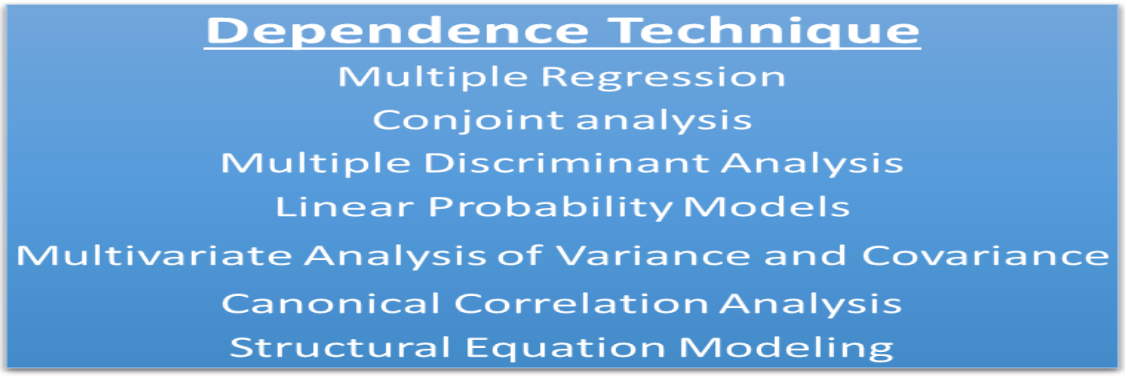
c) How are the variables, both dependent and independent measured?

Multivariate analysis technique can be classified into two broad categories viz., this classification depends upon the question: are the involved variables dependent on each other or not?

If the answer is yes: We have **Dependence methods.**

If the answer is no: We have **Interdependence methods.**

Dependence technique: Dependence Techniques are types of multivariate analysis techniques that are used when one or more of the variables can be identified as dependent variables and the remaining variables can be identified as independent.



Multiple Regression

Multiple Regression Analysis– Multiple regression is an extension of simple linear regression. It is used when we want to predict the value of a variable based on the value of two or more other variables. The variable we want to predict is called the dependent variable (or sometimes, the outcome, target, or criterion variable). Multiple regressions use multiple "x" variables for each independent variable: (x1)1, (x2)1, (x3)1, Y1)

Conjoint analysis

'**Conjoint analysis**' is a survey-based statistical technique used in market research that helps determine how people value different attributes (feature, function,

benefits) that make up an individual product or service. The objective of conjoint analysis is to determine the choices or decisions of the end-user, which drives the policy/product/service. Today it is used in many fields including marketing, product management, operations research, etc.

It is used frequently in testing consumer response to new products, in acceptance of advertisements and in-service design. Conjoint analysis techniques may also be referred to as multi-attribute compositional modelling, discrete choice modelling, or stated preference research, and is part of a broader set of trade-off analysis tools used for systematic analysis of decisions.

There are multiple conjoint techniques, few of them are CBC (Choice-based conjoint) or ACBC (Adaptive CBC).

Multiple Discriminant Analysis

The objective of discriminant analysis is to determine group membership of samples from a group of predictors by finding linear combinations of the variables which maximize the differences between the variables being studied, to establish a model to sort objects into their appropriate populations with minimal error.

Discriminant analysis derives an equation as a linear combination of the independent variables that will discriminate best between the groups in the dependent variable. This linear combination is known as the discriminant function.

The weights assigned to each independent variable are corrected for the interrelationships among all the variables. The weights are referred to as discriminant coefficients.

The discriminant equation:

$F = \beta 0 + \beta 1X1 + \beta 2X2 + \ldots + \beta pXp + \varepsilon$

where, F is a latent variable formed by the linear combination of the dependent variable, X1, X2,... XP is the p independent variable, ε is the error term and β0, β1, β2,..., βp is the discriminant coefficients.

A linear probability model

A linear probability model (LPM) is a regression model where the outcome variable is binary, and one or more explanatory variables are used to predict the outcome.

Explanatory variables can themselves be binary or be continuous. If the classification involves a binary dependent variable and the independent variables include non-metric ones, it is better to apply linear probability models.

Binary outcomes are everywhere: whether a person died or not, broke a hip has hypertension or diabetes, etc.

We typically want to understand what the probability of the binary outcome is given explanatory variables.

We could actually use our linear model to do so; it's very simple to understand why.

If Y is an indicator or dummy variable, then E[Y |X] is the proportion of 1s given X, which we interpret as a probability of Y given X.

$$died_i = \beta_0 + \beta_1 age_i + \epsilon_i$$

We can then interpret the parameters as the change in the probability of Y when X changes by one unit or for a small change in X For example, if we model , we could interpret β1 as the change in the probability of death for an additional year of age

Multivariate Analysis of Variance and Covariance

Multivariate analysis of variance (MANOVA) is an extension of a common analysis of variance (ANOVA). In ANOVA, differences among various group means on a single-response variable are studied. In MANOVA, the number of response variables is increased to two or more. The hypothesis concerns a comparison of vectors of group means. A MANOVA has one or more factors (each with two or more levels) and two or more dependent variables. The calculations are extensions of the general linear model approach used for ANOVA.

Canonical Correlation Analysis

Canonical correlation analysis is the study of the linear relations between two sets of variables. It is the multivariate extension of correlation analysis.

CCA is used for two typical purposes:-

- Data Reduction
- Data Interpretation

You could compute all correlations between variables from the one set (p) to the variables in the second set (q), however interpretation is difficult when pq is large.

Canonical Correlation Analysis allows us to summarize the relationships into a lesser number of statistics while preserving the main facets of the relationships. In a way, the motivation for canonical correlation is very similar to principal component analysis.

Structural Equation Modelling

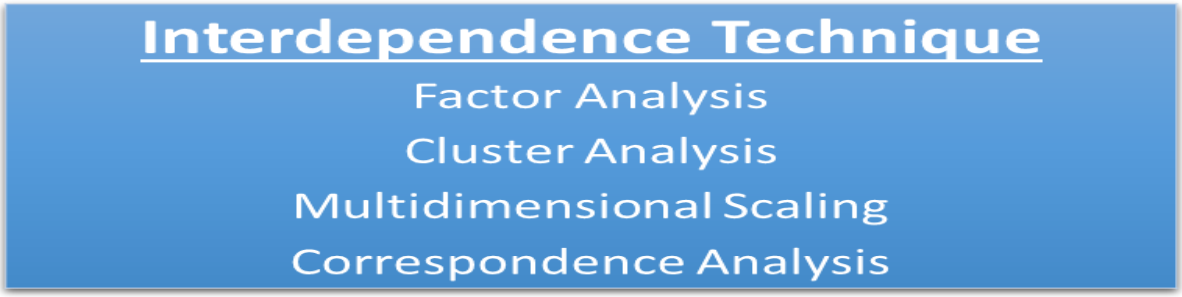
Structural equation modelling is a multivariate statistical analysis technique that is used to analyse structural relationships. It is an extremely broad and flexible framework for data analysis, perhaps better thought of as a family of related methods rather than as a single technique.

SEM in a single analysis can assess the assumed causation among a set of dependent and independent constructs i.e. validation of the structural model and the loadings of observed items (measurements) on their expected latent variables (constructs) i.e. validation of the measurement model. The combined analysis of the measurement and the structural model enables the measurement errors of the observed variables to be analysed as an integral part of the model, and factor analysis combined in one operation with the hypotheses testing.

Interdependence Technique

Interdependence techniques are a type of relationship that variables cannot be classified as either dependent or independent.

It aims to unravel relationships between variables and/or subjects without explicitly assuming specific distributions for the variables. The idea is to describe the patterns in the data without making (very) strong assumptions about the variables.



Factor Analysis

Factor analysis is a way to condense the data in many variables into just a few variables. For this reason, it is also sometimes called "dimension reduction". It

makes the grouping of variables with high correlation. Factor analysis includes techniques such as principal component analysis and common factor analysis.

This type of technique is used as a pre-processing step to transform the data before using other models. When the data has too many variables, the performance of multivariate techniques is not at the optimum level, as patterns are more difficult to find. By using factor analysis, the patterns become less diluted and easier to analyse.

Cluster analysis

Cluster analysis is a class of techniques that are used to classify objects or cases into relative groups called clusters. In cluster analysis, there is no prior information about the group or cluster membership for any of the objects.

- While doing cluster analysis, we first partition the set of data into groups based on data similarity and then assign the labels to the groups.
- The main advantage of clustering over classification is that it is adaptable to changes and helps single out useful features that distinguish different groups.

Cluster Analysis used in outlier detection applications such as detection of credit card fraud. As a data mining function, cluster analysis serves as a tool to gain insight into the distribution of data to observe the characteristics of each cluster.

Multidimensional Scaling

Multidimensional scaling (MDS) is a technique that creates a map displaying the relative positions of several objects, given only a table of the distances between them. The map may consist of one, two, three, or even more dimensions. The program calculates either the metric or the non-metric solution. The table of distances is known as the proximity matrix. It arises either directly from experiments or indirectly as a correlation matrix.

Correspondence analysis

Correspondence analysis is a method for visualizing the rows and columns of a table of non-negative data as points in a map, with a specific spatial interpretation. Data are usually counted in a cross-tabulation, although the method has been extended too many other types of data using appropriate data transformations. For cross-tabulations, the method can be considered to explain the association between the rows and columns of the table as measured by the Pearson chi-square statistic. The method has several similarities to principal component analysis, in that it situates

the rows or the columns in a high-dimensional space and then finds a best-fitting subspace, usually a plane, in which to approximate the points.

A correspondence table is any rectangular two-way array of non-negative quantities that indicates the strength of association between the row entry and the column entry of the table. The most common example of a correspondence table is a contingency table, in which row and column entries refer to the categories of two categorical variables, and the quantities in the cells of the table are frequencies.

The Objective of multivariate analysis

(1) **Data reduction or structural simplification**: This helps data to get simplified as possible without sacrificing valuable information. This will make interpretation easier.

(2) **Sorting and grouping**: When we have multiple variables, Groups of "similar" objects or variables are created, based upon measured characteristics.

(3**) Investigation of dependence among variables**: The nature of the relationships among variables is of interest. Are all the variables mutually independent or are one or more variables dependent on the others?

(4) **Prediction Relationships between variables**: must be determined for the purpose of predicting the values of one or more variables based on observations on the other variables.

(5**) Hypothesis construction and testing**. Specific statistical hypotheses, formulated in terms of the parameters of multivariate populations, are tested. This may be done to validate assumptions or to reinforce prior convictions.

Dependence methods

Dependence methods are used when one or some of the variables are dependent on others. Dependence looks at cause and effect; in other words, can the values of two or more independent variables be used to explain, describe, or predict the value of another, dependent variable? To give a simple example, the dependent variable of "weight" might be predicted by independent variables such as "height" and "age."

In machine learning, dependence techniques are used to build predictive models. The analyst enters input data into the model, specifying which variables are independent and which ones are dependent—in other words, which variables they want the model to predict, and which variables they want the model to use to make those predictions.

Interdependence methods

Interdependence methods are used to understand the structural makeup and underlying patterns within a dataset. In this case, no variables are dependent on others, so you're not looking for causal relationships. Rather, interdependence methods seek to give meaning to a set of variables or to group them together in meaningful ways.

Hypotheses testing and Measures of Association:

What Is Hypothesis Testing?

Hypothesis testing is an act in statistics whereby an analyst tests an assumption regarding a population parameter. The methodology employed by the analyst depends on the nature of the data used and the reason for the analysis.

Hypothesis testing is used to assess the plausibility of a hypothesis by using sample data. Such data may come from a larger population, or from a data-generating process. The word "population" will be used for both of these cases in the following descriptions.

How Hypothesis Testing Works

In hypothesis testing, an analyst tests a statistical sample, with the goal of providing evidence on the plausibility of the null hypothesis.

Statistical analysts test a hypothesis by measuring and examining a random sample of the population being analysed. All analysts use a random population sample to test two different hypotheses: the null hypothesis and the alternative hypothesis.

The null hypothesis is usually a hypothesis of equality between population parameters; e.g., a null hypothesis may state that the population mean return is equal to zero. The alternative hypothesis is effectively the opposite of a null hypothesis (e.g., the population mean return is not equal to zero). Thus, they are mutually exclusive, and only one can be true. However, one of the two hypotheses will always be true.

4 Steps of Hypothesis Testing

All hypotheses are tested using a four-step process:

1. The first step is for the analyst to state the two hypotheses so that only one can be right.
2. The next step is to formulate an analysis plan, which outlines how the data will be evaluated.
3. The third step is to carry out the plan and physically analyse the sample data.

4. The fourth and final step is to analyse the results and either reject the null hypothesis, or state that the null hypothesis is plausible, given the data.

Real-World Example of Hypothesis Testing

If, for example, a person wants to test that a penny has exactly a 50% chance of landing on heads, the null hypothesis would be that 50% is correct, and the alternative hypothesis would be that 50% is not correct.

Mathematically, the null hypothesis would be represented as Ho: P = 0.5. The alternative hypothesis would be denoted as "Ha" and be identical to the null hypothesis, except with the equal sign struck-through, meaning that it does not equal 50%.

A random sample of 100 coin flips is taken, and the null hypothesis is then tested. If it is found that the 100 coin flips were distributed as 40 heads and 60 tails, the analyst would assume that a penny does not have a 50% chance of landing on heads and would reject the null hypothesis and accept the alternative hypothesis.

If, on the other hand, there were 48 heads and 52 tails, then it is plausible that the coin could be fair and still produce such a result. In cases such as this where the null hypothesis is "accepted," the analyst states that the difference between the expected results (50 heads and 50 tails) and the observed results (48 heads and 52 tails) is "explainable by chance alone."

Hypothesis Testing _ (Alternate Content)

Hypothesis testing is the use of statistics to determine the probability that a given hypothesis is true. The usual process of hypothesis testing consists of four steps.

1. Formulate the null hypothesis (H NOT) (commonly, that the observations are the result of pure chance) and the alternative hypothesis Ha (commonly, that the observations show a real effect combined with a component of chance variation).
2. Identify a test statistic that can be used to assess the truth of the null hypothesis.
3. Compute the P-value, which is the probability that a test statistic at least as significant as the one observed would be obtained assuming that the null hypothesis were true. The smaller the P-value, the stronger the evidence against the null hypothesis.
4. Compare the P-value to an acceptable significance value alpha (sometimes called an alpha value). If P<=alpha, that the observed effect is statistically significant, the null hypothesis is ruled out, and the alternative hypothesis is valid.

Measures of Association:

Measure of association, in statistics, any of various factors or coefficients used to quantify a relationship between two or more variables. Measures of association are used in various fields of research but are especially common in the areas of epidemiology and psychology, where they frequently are used to quantify relationships between exposures and diseases or behaviours.

A measure of association may be determined by any of several different analyses, including correlation analysis and regression analysis.

(Although the terms *correlation* and *association* are often used interchangeably, *correlation* in a stricter sense refers to linear correlation, and *association* refers to any relationship between variables.) The method used to determine the strength of an association depends on the characteristics of the data for each variable. Data may be measured on an interval/ratio scale, an ordinal/rank scale, or a nominal/categorical scale. These three characteristics can be thought of as continuous, integer, and qualitative categories, respectively

Methods of analysis

Pearson's correlation coefficient

A typical example for quantifying the association between two variables measured on an interval/ratio scale is the analysis of relationship between a person's height and weight. Each of these two characteristic variables is measured on a continuous scale.

The appropriate measure of association for this situation is Pearson's correlation coefficient, *r* (rho), which measures the strength of the linear relationship between two variables on a continuous scale. The coefficient *r* takes on the values of −1 through +1. Values of −1 or +1 indicate a perfect linear relationship between the two variables, whereas a value of 0 indicates no linear relationship. (Negative values simply indicate the direction of the association, whereby as one variable increases, the other decreases.)

Correlation coefficients that differ from 0 but are not −1 or +1 indicate a linear relationship, although not a perfect linear relationship. In practice, ρ (the population correlation coefficient) is estimated by *r*, which is the correlation coefficient derived from sample data.

Spearman rank-order correlation coefficient

The Spearman rank-order correlation coefficient (Spearman rho) is designed to measure the strength of a monotonic (in a constant direction) association between two variables measured on an ordinal or ranked scale. Data that result from ranking and data collected on a scale that is not truly interval in nature (e.g., data obtained from Likert-scale administration) are subject to Spearman correlation analysis. In addition, any interval data may be transformed to ranks and analysed with the Spearman rho, although this results in a loss of information. Nonetheless, this approach may be used, for example, if one variable of interest is measured on an interval scale and the other is measured on an ordinal scale. Similar to Pearson's correlation coefficient, Spearman rho may be tested for its significance. A similar measure of strength of association is the Kendall tau, which also may be applied to measure the strength of a monotonic association between two variables measured on an ordinal or rank scale.

As an example of when Spearman rho would be appropriate, consider the case where there are seven substantial health threats to a community. Health officials wish to determine a hierarchy of threats in order to most efficiently deploy their resources. They ask two credible epidemiologists to rank the seven threats from 1 to 7, where 1 is the most significant threat. The Spearman rho or Kendall tau may be calculated to measure the degree of association between the epidemiologists' rankings, thereby indicating the collective strength of a potential action plan. If there is a significant association between the two sets of ranks, health officials may feel more confident in their strategy than if a significant association is not evident.

Chi-square test

The chi-square test for association (contingency) is a standard measure for association between two categorical variables. The chi-square test, unlike Pearson's correlation coefficient or Spearman rho, is a measure of the significance of the association rather than a measure of the strength of the association.

A simple and generic example follows. If scientists were studying the relationship between gender and political party, then they could count people from a random sample belonging to the various combinations: female-Democrat, female-Republican, male-Democrat, and male-Republican. The scientists could then perform a chi-square test to determine whether there was a significant disproportionate membership among those groups, indicating an association between gender and political party.

Presenting Insights and findings using written reports and oral presentation:

Oral presentation:

Two words that are capable of striking fear into the hearts of even the most confident student. But should they? Though not all of us can ever hope to reach the heady heights of oratory genius achieved by the likes of Barack Obama or Martin Luther King Jr, there are steps we can take to help us to present our point of view strongly.

Step 1: Research

Find out as much as you can about your chosen topic. The key skills for presenting argument in the VCE English Study Design clearly state that you need to 'conduct research to support the development of arguments on particular issues and acknowledge sources accurately and appropriately where relevant'. You are expected to research your chosen topic so that you have a deep and nuanced understanding of the issues and arguments. Read from multiple sources that present various points of view, and take notes on the arguments used.

Step 2: Plan your overall approach

Great speeches very rarely just happen; they are carefully crafted pieces of writing. Use your knowledge of argument and persuasive language as a basis for the development of your oral presentation. Remember that you are required to provide a written statement of intention to accompany your presentation. This statement of intention must outline the decisions you have made in the planning process, and explain how these demonstrate understanding of argument and persuasive language.

So, before you start writing, take the time to think carefully about the following aspects of your presentation.

Your contention

Where do you stand on the issue? Why? Express this in a clear and direct sentence. Avoid statements such as 'Greyhound racing is bad'. This a vague and general opinion, not a contention. A contention on this issue would be something like 'The cruel and abusive practice of greyhound racing should be banned immediately'.

Your context and audience

Who are you addressing? By that, I don't mean your teacher or your classmates. Rather, who is your imagined audience for the speech? This is important to keep in mind, as it will inform the language choices you make. Furthermore, consider in what context you

would be addressing your audience. Is your speech designed to be delivered on the steps of parliament at a rally or to a group of students at a graduation dinner? Decide this before you start writing. And don't be afraid to adopt a persona – this will allow you broader scope in selecting a particular context and audience.

Once you have decided on your contention and on your context and audience, it is time to consider some of the finer details of your presentation.

Your purpose

What do you want your imagined audience to think, feel or do? Do you wish to inform or educate them? To create alarm? To effect change? Your purpose should be closely related to your contention.

Your tone

What feelings are you seeking to communicate and to evoke in the audience? What mood are you trying to generate? Will you be using humour to relax your audience? Will you be hostile? Sympathetic? Will your tone change at any point and, if so, why?

All of the above are important factors to consider, as they will affect your language choices and the persuasive language techniques you employ.

Step 3: Plan your arguments

Now you need to decide on your supporting arguments. For each argument, ask

Yourself:

What persuasive language techniques will I use?

What evidence will I present?

Try to vary your chosen techniques, and remember Aristotle's principles of rhetoric – logos (appeal to logic and reason), ethos (character of the speaker) and pathos (emotional influence of the speaker). A strong argument will address all three elements in varying degrees.

Step 4: Write the introduction

Good speeches start strongly. You need to grab the audience's attention and make your point of view clear from the outset. The way you begin should be consistent with your audience and purpose. Strategies that you might consider are listed below.

Anecdote – this is a great way to highlight a personal connection to the issue or to strike a sympathetic tone.

Statistics – if your purpose is to shock your audience or to promote change, this is a great way to 'hit them hard' right from the outset.

Inclusive language – if you want to create a shared sense of purpose, make it clear to your audience that they are part of this issue, and that how they feel matters.

Once you have your audience's attention, introduce yourself (or your persona), clarify the issue, state your contention and signpost your main arguments.

Step 5: Write the body

This is where all you're planning from step 3 pays off!

For each body paragraph, ensure that you create strong topic sentences that clearly highlight your main arguments, and then develop each argument using your carefully selected language and evidence.

There are a few things that you should keep in mind as you write:

Cohesion is king! Keep your line of argument consistent and use connectives throughout.

Analyse the evidence! Don't just present a raft of statistics or evidence and expect them to make the argument for you. Analyse their importance in relation to the debate.

Include some rebuttal! An issue has two sides – you need to rebut some or all arguments from the opposing point of view.

Step 6: Write the conclusion

Aim to finish strongly. Reiterate your contention and then tell the audience what they should think, feel or do. (This should directly relate to the purpose you decided on in the planning stage).

To ensure that you finish on a powerful note, consider using an appeal, a rhetorical question, or a call to action.

Step 7: Proofread and practise

Read your speech to friends or family and get their feedback. Did the line of argument make sense to them? Did you persuade them? Did any parts of your speech lose their attention? Take note of these responses and edit your speech as required.

3. T.H. Cormen, C.E.Leiserson, R.L. Rivest and C.Stein, "Introduction to Algorithms", Prentice Hall of India, 3rd Edition, 2012.
4. Mark Allen Weiss, “Data Structures and Algorithms in C++”, Pearson Education, 3rd Edition, 2009.
5. E. Horowitz, S. Sahni and S. Rajasekaran, “Fundamentals of Computer Algorithms”, University Press, 2nd Edition, 2008.
6. Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman, “Data Structures and Algorithms”, Pearson Education, Reprint 2006.

CO-PO Mapping

CO	POs					
	PO1	PO2	PO3	PO4	PO5	PO6
1	2	1	3	2	1	2
2	2	1	3	2	2	2
3	2	1	3	2	2	2
4	3	1	3	2	2	2
5	3	1	3	2	2	2
Avg	2.5	1	3	2	1.8	2

MC4102 OBJECT ORIENTED SOFTWARE ENGINEERING L T P C
3 0 0 3

COURSE OBJECTIVES:

- To understand the phases in object oriented software development
- To gain fundamental concepts of requirements engineering and analysis.
- To know about the different approach for object oriented design and its methods
- To learn about how to perform object oriented testing and how to maintain software
- To provide various quality metrics and to ensure risk management.

UNIT I SOFTWARE DEVELOPMENT AND PROCESS MODELS 9

Introduction to Software Development – Challenges – An Engineering Perspective – Object Orientation – Software Development Process – Iterative Development Process – Process Models – Life Cycle Models – Unified Process – Iterative and Incremental – Agile Processes.

UNIT II MODELING OO SYSTEMS 9

Object Oriented Analysis (OOA / Coad-Yourdon), Object Oriented Design (OOD/Booch), Hierarchical Object Oriented Design (HOOD), Object Modeling Technique (OMT) – Requirement Elicitation – Use Cases – SRS Document – OOA - Identification of Classes and Relationships, Identifying State and Behavior – OOD - Interaction Diagrams – Sequence Diagram – Collaboration Diagrams - Unified Modeling Language and Tools.

UNIT III DESIGN PATTERNS 9

Design Principles – Design Patterns – GRASP – GoF – Dynamic Object Modeling – Static Object

Modeling.

UNIT IV SYSTEM TESTING 9

Software testing: Software Verification Techniques – Object Oriented Checklist :- Functional Testing – Structural Testing – Class Testing – Mutation Testing – Levels of Testing – Static and Dynamic Testing Tools - Software Maintenance – Categories – Challenges of Software Maintenance – Maintenance of Object Oriented Software – Regression Testing

UNIT V SOFTWARE QUALITY AND METRICS 9

Need of Object Oriented Software Estimation – Lorenz and Kidd Estimation – Use Case Points Method – Class Point Method – Object Oriented Function Point – Risk Management – Software Quality Models – Analyzing the Metric Data – Metrics for Measuring Size and Structure – Measuring Software Quality - Object Oriented Metrics

SUGGESTED ACTIVITIES:

1. Discuss the different phases in any domain like Health Monitoring System using extreme programming
2. Describe Business Requirement Specification (BRS) and SRS (Software Requirement Specification) for any Project like Automatic Intelligent Plant Watering System .using any one of requirement analysis tool
3. Identify the classes , relationship between classes and draw standard UML diagrams using any one UML modeling tool (eg: ArgoUML that supports UML 1.4 and higher)
4. for a system (eg: Conference Management System, student management system)
5. Test the above UML for all the scenarios identified using Selenium /JUnit / Apache JMeter
6. Perform COCOMO estimation for Book Management System to find effort and development time considering all necessary cost estimation factors. (Use GanttPRO Software for estimation)

COURSE OUTCOMES:

On completion of the course the student would be able to :

CO1: Design object oriented software using appropriate process models.
CO2: Differentiate software processes under waterfall and agile methodology.
CO3: Design and Develop UML diagrams for software projects.
CO4: Apply Design Patterns for a software process.
CO5: Categorize testing methods and compare different testing tools for software processes.
CO6: Analyze object oriented metrics and quality for software engineering processes.

TOTAL: 45 PERIODS

REFERENCES:

1. Yogesh Singh, RuchikaMalhotra, " Object – Oriented Software Engineering", PHI Learning Private Limited ,First edition,2012
2. Ivar Jacobson. Magnus Christerson, PatrikJonsson, Gunnar Overgaard, "Object Oriented Software Engineering, A Use Case Driven Approach", Pearson Education, Seventh Impression, 2009
3. Craig Larman, "Applying UML and Patterns, an Introduction to Object-Oriented Analysis and Design and Iterative Development", Pearson Education, Third Edition, 2008.
4. Grady Booch, Robert A. Maksimchuk, Michael W. Engle, Bobbi J. Young, Jim Conallen,

UNIT V SOFTWARE QUALITY AND METRICS 9

Need of Object Oriented Software Estimation – Lorenz and Kidd Estimation – Use Case Points Method – Class Point Method – Object Oriented Function Point – Risk Management – Software Quality Models – Analyzing the Metric Data – Metrics for Measuring Size and Structure – Measuring Software Quality - Object Oriented Metrics

1.1 A brief history of the metrics

Software metrics lies on the ancient discipline of measurement mainly developed by scientists (physicians). On this basis, some take up measurement principles in order to measure software activities.

So metrics origin goes back to the sixties with the Lines of Code (LOC) metric used to measure programmer's productivity and program quality (e.g. number of defects per KLOC1). The main aim was to provide information to support quantitative managerial decision-making during the software lifecycle

[20, p.357].

1.2 Defifinitions

There's two main concepts to defifine at this point : *the measurement activity*
and *the software metrics.*

1.2.1 Measurement activity

Norman Fenton gives the two following defifinitions of the *measurement activity*
[22, p.28]:
Formally, we defifine measurement as a mapping from the empirical world to the formal, relational world. Consequently, a measure is the number or symbol assigned to an entity by this mapping in order to characterize an attribute.
He gives this second defifinition introducing the numerical aspect [22, p.5]:
Measurement is the process by which numbers or symbols are as signed to attributes of entities in the real world in such a way as to describe them according to clearly defifined rules.

1.2.2 Software metrics

The fifirst defifinition of *software metrics* is proposed by Norman Fenton [20,
p.358]:
(...)*software metrics is a collective term used to describe the very*

wide range of activities concerned with measurement in software en gineering. These activities range from producing numbers that char acterize properties of software code (these are the classic software 'metrics') through to models that help predict software resource re quirement and software quality. The subject also includes the quan titative aspects of quality control and assurance - and this covers activities like recording and monitoring defects during development and testing.

An other defifinition of software metrics is due to Paul Goodman [25] :

The continuous application of measurement-based techniques to the software development process and its products to supply meaningful and timely management information, together with the use of those techniques to improve that process and its products". Applied To Engineering & Management Processes, Products & To Supply

Theory of Measure

2.1 Theory

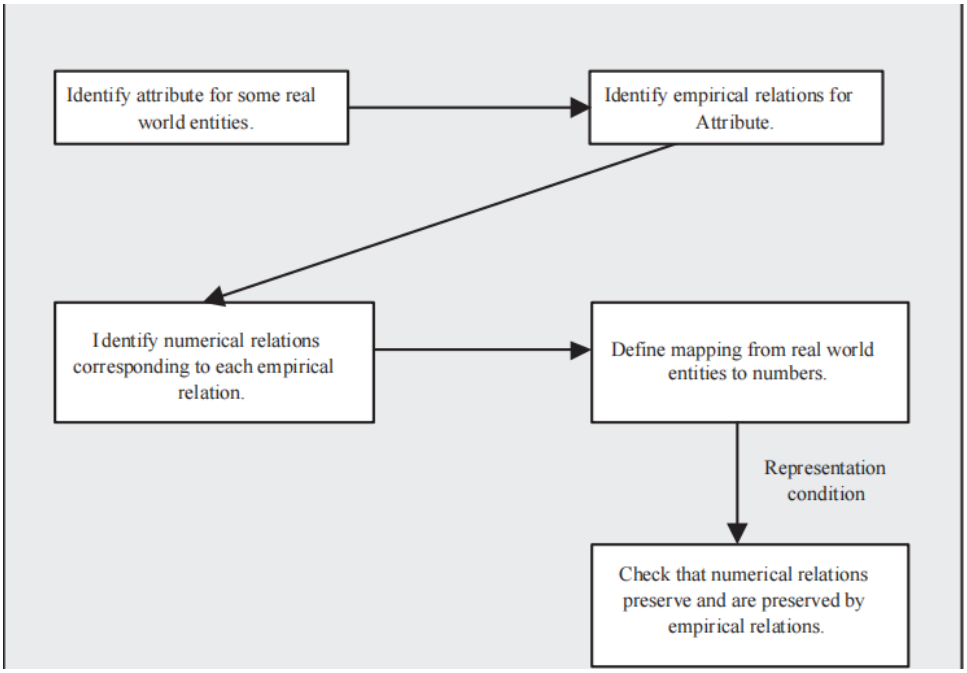
Measurement theory has fifirst been developed as a particular discipline of physics.

Physicians defifined fundamental rules in order to use correctly the new frame
work of measures. Further research led to develop several theories of measures.

Among them the *representational theory of measurement* has been used to build
the software metrics [22, p.24].

The representational theory of measurement aims at formalizing our intu
ition about the way the world works. We collect a set of data, the measures,
which should represent attributes of the entities observed. Manipulation of these
data must preserve the relationship observed among these entities [22, p.24-25].

Actually we collect measures about the real world and try to understand these
measures by comparing them. By example, we observe that certain people are
taller than others without measuring them. In fact, says Fenton, *our observa*
tion reflflects a set of rules that we are imposing on the set of people.[22, p.25].
So we defifine some binary relations between these entities.
Example 1 : When we say that *X* is taller than *Y* we defifine a binary relation between X and Y. In this case, "taller than" is an empirical relation for height.
The measurement activity is then defifined as the mapping between the em
pirical and the formal world1. So a measure is a number or symbol assigned to
an entity in order to characterize an attribute.
Basic rules are simple : the real world is the *domain* and the mathematical
world, the *range*. When we map an attribute to a mathematical system the
following rules must be respected :
(...)the representation condition asserts that a measurement map ping M must map entities into numbers and empirical relations into numerical relations in such a way that the empirical relations pre serve and are preserved by the numerical relations.[22, p.31]



To avoid traps (compare measures from entities which are not comparable)
several models have been introduced2. A model is an abstraction of reality, al
lowing us to strip details away and view an entity or concept from a particular
perspective. [22, p.36-37] Several models are likely to interest software metrics
: *cost-estimation model, quality models, capability-maturity model,....*
The use
for such models avoid to focus only on the formal and mathematical systems
and neglect the empirical one. A model will show fundamental characteris
tics and how they are articulate. This will allow to defifine a metric for each
characteristic.[22, p.38]
The model chosen, entities and attributes defifined, measures can be defifined
too. But when there are complex relationships among attributes, or when an

attributes must be measured by combining several of its aspects, then we need
a model for how to combine the related measures. For this reason, direct and
indirect measurements are distinguished.

2.1.1 Direct measurement

Direct measurement of an entity attribute involves no other attribute or entity.
For example, we can measure the length of a physical object without any other
object. Measures below are direct measures used in software engineering :

- Length of source code (LOC)
- Duration of testing process (Hours)
- Number of defects discovered (counting defects)
- Time a programmer spent on a project (Months)

2.1.2 Indirect measurement

Indirect measurement are measures of an attribute obtained by comparing dif
ferent measurements. For example, *Number of defects* divided by *module size*
gives the *Module defect density*.[22, p.40]

2.2 Measurement scales

Previous section shows how direct measurement assigns a representation or map
ping from observed relation system to numerical relation system. These kind
of measures are done in order to extract relationship between data and to draw
conclusion about them. However all the mappings are not the same. Diffffer
ences between them restrict the possible kind of comparison and analysis. To
avoid unappropriate analysis, ***measurement scale*** concept has been intro

duced as a principle by scientists.[22, 45-47] Five major measurement scales are identifified[37][22] :

- Nominal
- Ordinal
- Interval
- Ratio
- Absolute

A ***nominal*** scale puts each entity into a particular category, based on the value of the attributes. It's the same process when we identify a programming language. By reading the code you can recognize it and classify it. This scale has two major characteristics :

- The empirical relation system only consists of diffffferent classes ; there is no notion of ordering among the classes.
- Any distinct numbering or symbolic representation of the classes is an acceptable measure. But there isn't any notion of magnitude associated with the number or symbol.

Example : For instance, we try to classify the set of software faults in the code. We choose a measurement scale where faults are entities and their location are attributes. So fault location could be in three diffffferent sets : specifification, design or code. Then we can defifine a mapping *M* that assign the diffffferent classes to a particular number. 45 if x is a specifification fault, 2 if x is a design fault and 37 if x is a code fault. The value is not important here.

An ***ordinal*** scale ranks items in an order, such as when we assign failures a

progressive severity like minor, major, and catastrophic. This scale has three
characteristics :

- Empirical relation system consists of ordered classes with respect to the
attribute.
- Any mapping that preserves the ordering is acceptable.
- Numbers represent only ranks, so addition, and other mathematical oper
ations have no sense.

Example : You want to classify the diffffferent modules of your soft ware in three classes which denote the complexity (trivial, simple, complex). Then you choose a mapping *M* like in the nominal scale : 1 if x is trivial, 2 if x is simple and 3 if x is complex. The diffffference with the previous scale lies in the fact that the measurement map ping must preserve the complexity order. 3 is bigger than 1 preserve the relation more complex.

An ***interval*** scale defifines a distance from one point to another, so that there
are equal intervals between consecutive numbers. This property permits compu
tations not available with the ordinal scale, such as calculating the mean value.

However, there is no absolute zero point in an interval scale, and thus ratios do
not make sense. Care is thus needed when you make comparisons. The three
main characteristics are :

- Order are preserved.
- Diffffferences are preserved but not ratios.
- Addition and substraction are acceptable but not multiplication and divi
sion.

Example : Take the temperature measurement on a Celsius or Fahren heit where each degree is a class related to heat. We say that the

temperature in a place X is 20 degrees Celsius and, at the same time, 30 degrees Celsius in place Y. If the temperature move, in X, from 20 to 21 the heat will increase exactly in the same way if it change from 30 to 31 in place Y. So the relationship is preserved.
The scale with more information and flflexibility is the ***ratio*** scale, which incor
porates an absolute zero, preserves ratios, and permits the most sophisticated
analysis. Measures such as lines of codes or numbers of defects are ratio mea
sures. It is for this scale that we can say that A is twice the size of B. There
are four characteristics :
• Ordering, size of the intervals between entities and ratios are preserved.
• There is a zero element (represents total lack of attribute).
• Measurement mapping start at zero and increases at equal intervals (units).
• All arithmetic can be applied to the classes in the range of the mapping.
Example : You use a ratio scale when you measure the physical size of entities. The scale start at zero which represent the total lack of size (theoretical - no existence). You can measure size in centimeters, meters,...
The ***absolute*** scale of measurement is the more restrictive scale. For any
two measures, *M* and *M'*, there is only one admissible transformation : the
identity transformation. So there's only one way in which the measurement can
be made, so *M* and *M'* must be equal. The absolute scale respects the four
following properties :
• the entity set. The measurement is made simply by counting the number of elements in

• entity". The Attribute always takes the form "number of occurrences of *x* in the
• There's only one measurement mapping, namely the actual count.
• All arithmetic analysis of the resulting count is meaningful.

Example : Lines Of Codes (LOC) is an absolute scale measure ment of the attribute "number of lines of codes" of a program. But "number of centimeters" is not an absolute scale measurement of an person's size because you can also use inches, meters,...

2.3 Validation

Validation of the measures is necessary to do before analysis. It aims prove that metrics used are actually measuring what they claim they do. Pflfleeger[37] say that a measure is *valid* if it satisfifies the representation condition : if it captures in the mathematical world the behavior we perceive in the empirical world. For example, we must show that if **H** is a measure of height, and if **A** is taller than **B**, then **H(A)** is larger than **H(b)**. But such a proof must, by nature, be empirical and is diffiffifficult to demonstrate. So, we must consider wether we are using direct measure (size) or an indirect measure (number of decision points as measure of size) and which entity and attribute are being addressed.Currently there isn't any accepted standard for validating a measure [22, p.106-108].

Software measures

3.1 Classes and attributes

In the previous section measurement basis and rules have been presented. The fifirst activity to achieve in measurement is the entity and attribute identifification.

In software there are three classes :
• Processes : collection of software-related activities.
• Products : artifacts, deliverables and documents resulting from processes
• Resources : entities required by a process activity.
In each class of entity, we distinguish between internal and external at
tributes :
• Internal attributes
of a product, process or resource are those that can be measured purely in terms of the product, process or resource itself. In other words, an internal attribute can be measured by examining the product, process or resource on its own, separate from its behavior.[22, p.74]
• External attributes
of a product, process or resources are those that can be measured only with respect to how the product, process or resource relates to its environment. Here, the behavior of the process, product or resource is important, rather than the entity itself.[22, p.74]

3.2 Processes

Processes are measured to inform on duration, cost, effffectiveness and effiffifficiency
of software development activities. There is several internal process attributes
which can be measured directly :
• the duration of the process or activity
• the effffort associated with process or activity
• tivity the number of incidents of a specifified type arising during process or ac-

Example 1 of measure for a process [22, p.77]:
(...)we may be reviewing our requirements to ensure their quality

before turning them over to the designers. To measure the effffective ness of the review process, we can measure the number of require ments errors found during specifification. Likewise, we can measure the number of faults found during integration testing to determine how well we are doing. And the number of personnel working on the project between May 1 and September 30 can give us insight into resources needed for the development process.

Example 2 of measures from AT&T [22, p.77] :

AT&T developers wanted to know the effffectiveness of their software inspections. In particular, managers needed to evaluate the cost of the inspections against the benefifits received. To do this, they mea sured the average amount of effffort expended per thousand lines of code reviewed. As we will see later in this chapter, this information, combined with measures of the number of faults discovered during the inspections, allowed the managers to perform a cost-benefifit analysis.

3.3 Products

Products can be also measured. By products we mean not only items delivered to customer. All the artifacts, documents and prototypes produced during the process are considered as products. All these process outputs can be measured in term of quality, size,... For all of them we distinguish both external and internal attributes.

3.3.1 External attributes

External product attributes depend on product behavior and environment that inflfluence the measure. Example of external attributes are : *usability, integrity, effiffifficiency, testability, reusability, portability, operability*[22, p.78].

3.3.2 Internal attributes

Internal products attributes are easy to measure in terms of size, length, func
tionality, correctness.[22, p.78] Code clarity is an example of internal attribute
according to defifined rules like *"avoid GOTO"*.

3.4 Resources

Last measurable entities are the resources like personnel, materials and methods.[22,
p.82] Measuring resources help managers to understand and control the process.
Programmer's productivity is often measured in terms of lines of code.

Measures and Models

As explained in chapter 2 models must be used with metrics to avoid metrics
misuse. A *Model* is *an abstraction of reality, allowing us to strip away detail
and view an entity or concept from a particular perspective*[22, p.36]. Models
can take the form of equations, mapping or diagrams. It allow to understand
relationship between the component parts related one to another in the model.
Fenton gives an example of this kind of relation highlighted in a model :

To measure length of programs using lines of code, we need a model of a program. The model would specify how a program diffffers from a subroutine, wether or not to treat separate statements on the same line as distinct lines of code, wether or not to count comment lines, wether or not to count data declarations, and so on. The model would also tell us what to do when we have programs written in a combination of diffffferent languages. It might distinguish delivered operational programs from those under development, and it would tell

us how to handle situations where diffffferent versions run on diffffferent
platforms.[22, p.37]
A model gives the domain and range of the measure mapping and it describes
the entity and attribute being measured, the set of possible resulting measures,
and the relationship among several measures.
Another characteristic of models is that they distinguish the prediction from
the assessment (measure to estimate future characteristics from previous ones
or the determination of the current condition of a process, product or resource).

4.2 Goal-Question-Metric paradigm

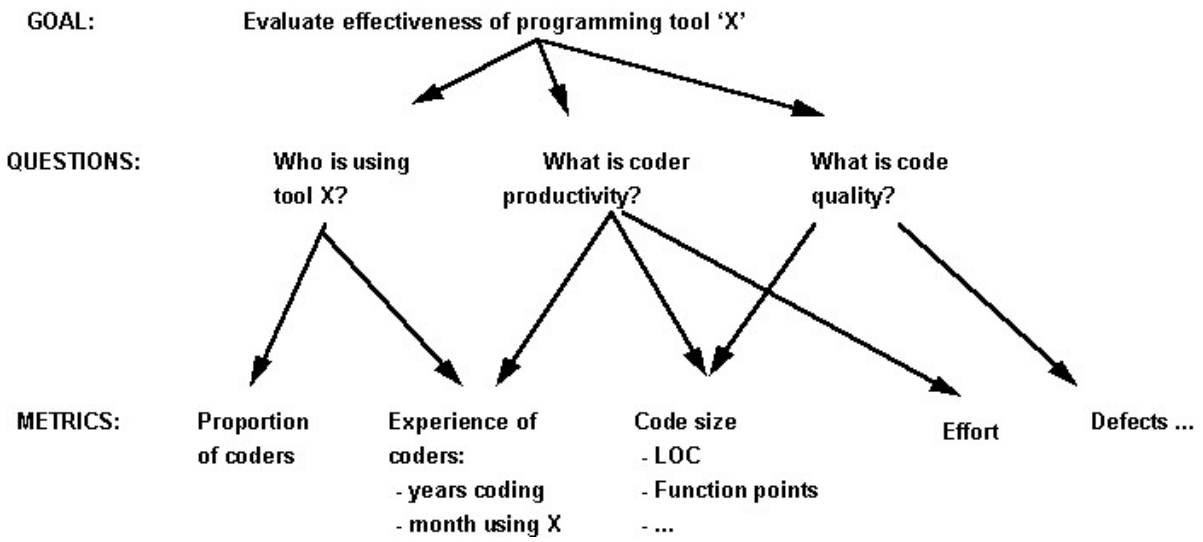
4.2.1 Origins

The Goal Question Metrics approach (GQM) has been suggested by Basili and
his colleagues in 1984.[5][4] They proposed an original approach to selecting and
implementing metrics. The GQM principle consists fifirst in expressing overall
goals of the organization. On this basis, questions whose answer to these g
are derived. Finally each question is analyzed in terms of what measurement is
needed to answer each question.

4.2.2 The paradigm

GQM provides a measurement framework involving three steps :
1. List the major goals of the development or maintenance projects.
2. Derive from each goal the questions that must be answered to to determine
if the goals are being met.
3. Decide what must be measured to answer the questions adequately.

The following fifigures illustrate how metrics are generated :



Fenton gives an example [22, p. 84]:
Suppose your overall goal is to evaluate the effffectiveness of using a coding standard (...). That is, you want to know if code produced by following the standard is superior in some way to code produced without it. To decide if the standard is effffective, you must ask sev eral key questions. fifirst, it is important to know who is using the standard, so that you can compare the productivity of the coders who use the standard with the productivity of those who do not. Like wise, you probably want to compare the quality of the code produced with the standard with the quality of non-standard code. Once these questions are identifified, you must analyse each question to deter mine what mst be measured in order to answer the question. For example, to understand who is using the standard, it is necessary to

know what proportion of coders is using the standard. However, it is also important to have an experience profifile of the coders, explaining
how long they have worked with the standard, the environment, the language, and other factors that will help to evaluate the effffective ness of the standard. The productivity question requires a defifinition of productivity, which is usually some measure of effffort divided by some measure of product size. (...) the metric can be in terms of lines of code, function points, or any other metrics that will be useful

to you. Similary, quality may be measured in terms of the number of errors found in the code, plus any other quality measures that you would like to use. In this way, you generate only those measures that are related to the goal. Notice that, in many cases, several mea surements may be needed to answer a single question. likewise, a single measurement may apply to more than one question; the goal provides the purpose for collecting the data, and the questions tell you and your project how to use the data.

4.3 Other models : quality models

Quality models aims at capturing the composite characteristics and their rela
tionship in order to measure quality. Among them, the McCall [32]and Boehm
[9]software quality model propose a decompositional approach. Since 1992 ISO [28] proposes the *Software Product Evaluation : Quality*
Characteristics and Guidelines for their use also know as ISO9126.

Goals of the Function Points

In 1979, Allan J. Albrecht proposed the fifirst function point's model and analysis
method called *Function Point Analysis*[1].1
This method's goals were to measure achievement and refifine valuation. Al
brecht proposes the three following defifinitions for the function points :

- Function points are a measurement of the software product based on the
user's function information treatment.
- Function points measure software by counting number of functionality of
the information's treatment associated with external and control data,
output and fifiles types.

•

This particular treatment is adjusted for the global function of information
treatment by applying an adjustment based on the software characteris
tics.
So function points are essentially based on the software's number of func
tionality proposed to the user. The goal was to obtain a technique to measure
productivity among diffffferent IBM's projects from 1974 to 1978. These projects
had been developed with diffffferent programming languages and tools. So the
objective was to provide a fifitted method to measure services provided to the
users.

5.2 History and evolution of function points

Albrecht began his research on this topic in the 70's. In the same time, Tom
DeMarco has also leading research on the same topic. His results are quite
the same in the concepts but not on the form. DeMarco's function points
have been recently used as basis for new research like : FFP (*Full Function
Points*), COSMIC-FFP (*Common Software Measurement International Con
sortium - Full Function Points*)[16].

ion points called *Mark II function points*.[45]
In 1984 IBM proposed a major review of the function points counting rules.
They added a evaluation procedure to assess complexity. This method became

the basic one to count function points taken by the IFPUG (*International Func*
*tion Points User Group*2). The function point's success and expansion and the
IFPUG creation contributed to the normalization of the function points mea
surement method.

5.3 Reliability of the function point's method

Navlaka proposes two fundamental rules that must be respected by measurement
method [34]:

• be obtained. Correctness : from the same data and rules, the same results must always

•

Repetitiveness : it doesn't matter the person who makes the measurement,
results must always be exactly the same at diffffferent time.

According industrial experiments the observed accuracy is more or less 95%.

Object Oriented
Measurement

6.1 Introduction

This section present several concepts on Object Oriented metrics. The impor
tance of the current research lead on this subject conduct us to devote it a
particular chapter.

6.2 Size measures

Object-Oriented systems generally grow in size between requirements analysis
and the testing phase. So diffffferent research have been done on this topics.

Pflfleeger used objects and methods as a basic size measurements which is more

accurate than COCOMO according to commercial applications [38, p.294].
An other method has been developed by Lorenz and Kidd [31]. They defifined
nine aspects of size that reflflect how the class characteristics affffect the product.
They propose the following aspects :
Number of scenario scripts (NSS) : It's the number of scenario scripts counted
in the use cases. This measure is correlated with application size and the
number of tests. NSS mainly allow to predict development and testing
effffforts.
Number of key classes : This measure evaluate the high-design effffort.
Number of support classes : This measures evaluates the low-level design.
Average number of support classes per key class : This measure gives an
idea of the system's structure.
Number of subsystems : This one provide more information on the system's
structure.
Class size : This measure include the number of operations and attributes.
Number of operations overridden by a class : Allow to evaluate inheri
tance effffects.
Number of operations added by a subclass : Measures also the inheritance
effffects.
Specialization index

6.3 Design measures

Chidamber and Kemerer have also provide a suite of metrics for object-oriented
developments [15]. They focuse their work more on design than on size so
they complement the Lorenz and Kidd's method. They focuse on the coupling
between objects, the response of a class and the lack of cohesion in methods[38,
p.297].
They calculate weighted methods per class in order to measure complexity.
They also defifine a class's depth of inheritance (It's the maximum length of the
path in the hierarchy from the class to the root of the inheritance tree). So more
deeper is a class in the hierarchy, more methods are inherited by this class.
Similarly, the number of children is the number of immediate subclasses
subordinated to the given class.

Collect, store, analyze and comment metrics

7.1 The data

7.1.1 Data properties

Before analyzing the data collection process there are several points to clarify.
Data that will be collected must satisfy several essential properties [22]:
Correctness The data were collected according to the exact rules of defifinition
of the metrics. For example, if comments are not supposed to be included
in the lines of codes count, then a check for correctness assures that no
comments were counted.

Accuracy This property refers to the diffffferences between the data and the
actual value. For example, time measurement will be less accurate on an
analog clock than on a digital one.
Precision It deals with the number of decimal places needed to express the
data.
Consistent In fact data must be consistent from one measuring device or per
son to another, without large diffffferences in value.
Time-Stamped We must know exactly when data has been collected in order
to allow comparison.
Replicated Last fundamental property hat assumes that the data defifinition
must be very accurate to allow other person to replicate the same mea
surement.

7.2 Data collection

There are two ways to collect data : the manual and the automatic. Manual
collection often conduct to bias, error, omission and delay. The automatic data
collection is preferable but often more diffiffifficult to implement. Fenton gives the
following guidelines to collect data :

- keep procedures simple;
- avoid unnecessary recording;
- train staffff in the need to record data and in the procedures to be used;
- provide the results of data capture and analysis to the original providers promptly and in a useful form that will assist them in their work;

• validate all data collected at a central point.

Data collection forms are interesting because they provide a frame to collect
data. But this form must be self-explanatory. 1
The GQM method previously
explained also provides methodology to collect data.
The data collection must be planned as ordinary project which is linked to
other projects to measure.
The collected data could be stored in database to allow further manipulations[22].

7.3 Analyze and comments

Data analysis and comment implies statistic methods and tools. They must be
used in an appropriate way. According to Fenton *Data sets of software attributes*
values must be analyzed with care, because software measures are not usually
normally distributed[22, p.235] There are diffffferent techniques that address a
wide variety of situations. Fenton gives the following advice :

- describe a set of attribute values using box plot statistics (based on median and quartiles) rather than on mean and variances;
- inspect a scatter plot visually when investigating the relation ship between two variables;
- use robust correlation coeffiffifficients to confifirm whether or not a relationship exists between two attributes;
- use robust regression in the presence of atypical values to iden tify a linear relationship between two attributes, or remove the atypical values before analysis;

• dant variable; always check the residuals by plotting them against the depen-

•

use Tukey's ladder to assist in the selection of transformations when faced with non-linear relationships;

• use principal component analysis to investigate the dimension ality of data sets with large numbers of correlated attributes.

In fact, the most important point, underlined by Fenton is the correlation

between the choice of the analysis technique and the goals of the investigation.

In this way you can support or refute the hypothesis you are testing.

UNIT - 2

A Brief History

The object-oriented paradigm took its shape from the initial concept of a new programming approach, while the interest in design and analysis methods came much later.

- The first object–oriented language was Simula (Simulation of real systems) that was developed in 1960 by researchers at the Norwegian Computing Center.
- In 1970, Alan Kay and his research group at Xerox PARK created a personal computer named Dynabook and the first pure object-oriented programming language (OOPL) - Smalltalk, for programming the Dynabook.
- In the 1980s, Grady Booch published a paper titled Object Oriented Design that mainly presented a design for the programming language, Ada. In the ensuing editions, he extended his ideas to a complete object–oriented design method.
- In the 1990s, Coad incorporated behavioral ideas to object-oriented methods.

The other significant innovations were Object Modelling Techniques (OMT) by James Rumbaugh and Object-Oriented Software Engineering (OOSE) by Ivar Jacobson.

Object-Oriented Analysis

Object–Oriented Analysis (OOA) is the procedure of identifying software engineering requirements and developing software specifications in terms of a software system's object model, which comprises of interacting objects.

The main difference between object-oriented analysis and other forms of analysis is that in object-oriented approach, requirements are organized around objects, which integrate both data and functions. They are modelled after real-world objects that the system interacts with. In traditional analysis methodologies, the two aspects - functions and data - are considered separately.

Grady Booch has defined OOA as, *"Object-oriented analysis is a method of analysis that examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain"*.

The primary tasks in object-oriented analysis (OOA) are −

- Identifying objects
- Organizing the objects by creating object model diagram
- Defining the internals of the objects, or object attributes
- Defining the behavior of the objects, i.e., object actions
- Describing how the objects interact

The common models used in OOA are use cases and object models.

Object-Oriented Design

Object–Oriented Design (OOD) involves implementation of the conceptual model produced during object-oriented analysis. In OOD, concepts in the analysis model, which are technology−independent, are mapped onto implementing classes, constraints are identified and interfaces are designed, resulting in a model for the

solution domain, i.e., a detailed description of how the system is to be built on concrete technologies.

The implementation details generally include −

- Restructuring the class data (if necessary),
- Implementation of methods, i.e., internal data structures and algorithms,
- Implementation of control, and
- Implementation of associations.

Grady Booch has defined object-oriented design as *"a method of design encompassing the process of object-oriented decomposition and a notation for depicting both logical and physical as well as static and dynamic models of the system under design"*.

Object-Oriented Programming

Object-oriented programming (OOP) is a programming paradigm based upon objects (having both data and methods) that aims to incorporate the advantages of modularity and reusability. Objects, which are usually instances of classes, are used to interact with one another to design applications and computer programs.

The important features of object–oriented programming are −

- Bottom–up approach in program design
- Programs organized around objects, grouped in classes
- Focus on data with methods to operate upon object's data
- Interaction between objects through functions
- Reusability of design through creation of new classes by adding features to existing classes

Some examples of object-oriented programming languages are C++, Java, Smalltalk, Delphi, C#, Perl, Python, Ruby, and PHP.

Grady Booch has defined object–oriented programming as *"a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships"*.

The object model visualizes the elements in a software application in terms of objects. In this chapter, we will look into the basic concepts and terminologies of object–oriented systems.

Objects and Classes

The concepts of objects and classes are intrinsically linked with each other and form the foundation of object–oriented paradigm.

Object

An object is a real-world element in an object–oriented environment that may have a physical or a conceptual existence. Each object has −

- Identity that distinguishes it from other objects in the system.
- State that determines the characteristic properties of an object as well as the values of the properties that the object holds.
- Behavior that represents externally visible activities performed by an object in terms of changes in its state.

Objects can be modelled according to the needs of the application. An object may have a physical existence,

like a customer, a car, etc.; or an intangible conceptual existence, like a project, a process, etc.

Class

A class represents a collection of objects having same characteristic properties that exhibit common behavior. It gives the blueprint or description of the objects that can be created from it. Creation of an object as a member of a class is called instantiation. Thus, object is an instance of a class.

The constituents of a class are −

- A set of attributes for the objects that are to be instantiated from the class. Generally, different objects of a class have some difference in the values of the attributes. Attributes are often referred as class data.
- A set of operations that portray the behavior of the objects of the class. Operations are also referred as functions or methods.

Example

Let us consider a simple class, Circle, that represents the geometrical figure circle in a two–dimensional space. The attributes of this class can be identified as follows −

- x–coord, to denote x–coordinate of the center
- y–coord, to denote y–coordinate of the center
- a, to denote the radius of the circle

Some of its operations can be defined as follows −

- findArea(), method to calculate area
- findCircumference(), method to calculate circumference
- scale(), method to increase or decrease the radius

During instantiation, values are assigned for at least some of the attributes. If we create an object my_circle, we can assign values like x-coord : 2, y-coord : 3, and a : 4 to depict its state. Now, if the operation scale() is performed on my_circle with a scaling factor of 2, the value of the variable a will become 8. This operation brings a change in the state of my_circle, i.e., the object has exhibited certain behavior.

Encapsulation and Data Hiding

Encapsulation

Encapsulation is the process of binding both attributes and methods together within a class. Through encapsulation, the internal details of a class can be hidden from outside. It permits the elements of the class to be accessed from outside only through the interface provided by the class.

Data Hiding

Typically, a class is designed such that its data (attributes) can be accessed only by its class methods and insulated from direct outside access. This process of insulating an object’s data is called data hiding or information hiding.

Example

In the class Circle, data hiding can be incorporated by making attributes invisible from outside the class and adding two more methods to the class for accessing class data, namely −

- setValues(), method to assign values to x-coord, y-coord, and a

- getValues(), method to retrieve values of x-coord, y-coord, and a

Here the private data of the object my_circle cannot be accessed directly by any method that is not encapsulated within the class Circle. It should instead be accessed through the methods setValues() and getValues().

Message Passing

Any application requires a number of objects interacting in a harmonious manner. Objects in a system may communicate with each other using message passing. Suppose a system has two objects: obj1 and obj2. The object obj1 sends a message to object obj2, if obj1 wants obj2 to execute one of its methods.

The features of message passing are −

- Message passing between two objects is generally unidirectional.
- Message passing enables all interactions between objects.
- Message passing essentially involves invoking class methods.
- Objects in different processes can be involved in message passing.

Inheritance

Inheritance is the mechanism that permits new classes to be created out of existing classes by extending and refining its capabilities. The existing classes are called the base classes/parent classes/super-classes, and the new classes are called the derived classes/child classes/subclasses. The subclass can inherit or derive the attributes and methods of the super-class(es) provided that the super-class allows so. Besides, the subclass may add its own attributes and methods and may modify any of the super-class methods. Inheritance defines an “is – a” relationship.

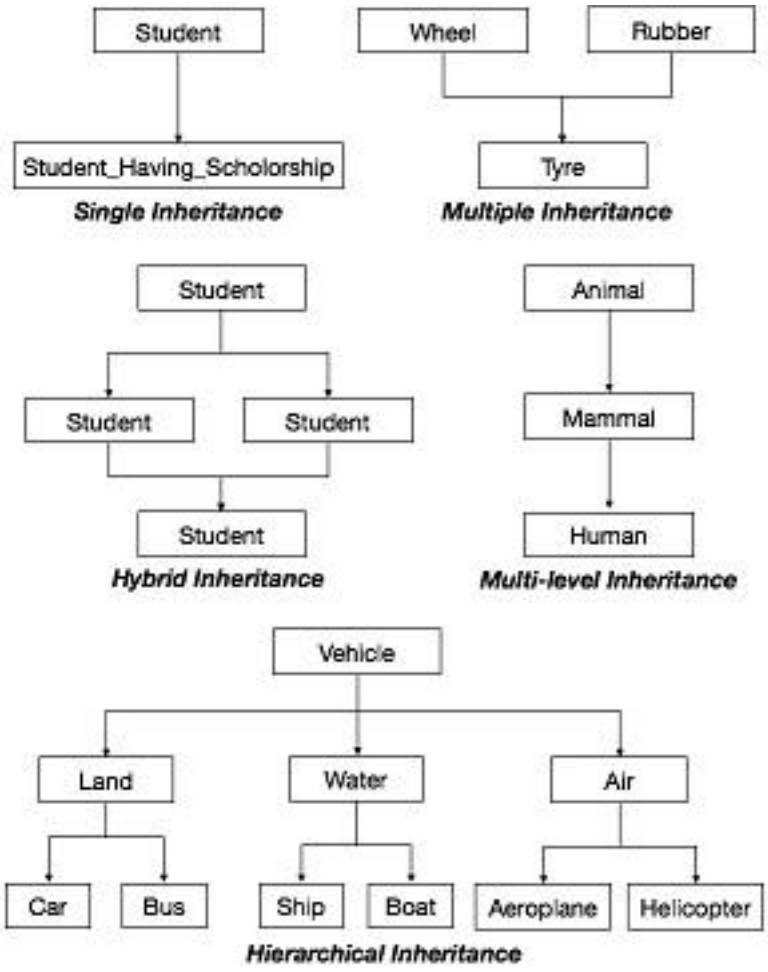
Example

From a class Mammal, a number of classes can be derived such as Human, Cat, Dog, Cow, etc. Humans, cats, dogs, and cows all have the distinct characteristics of mammals. In addition, each has its own particular characteristics. It can be said that a cow “is – a” mammal.

Types of Inheritance

- **Single Inheritance** − A subclass derives from a single super-class.
- **Multiple Inheritance** − A subclass derives from more than one super-classes.
- **Multilevel Inheritance** − A subclass derives from a super-class which in turn is derived from another class and so on.
- **Hierarchical Inheritance** − A class has a number of subclasses each of which may have subsequent subclasses, continuing for a number of levels, so as to form a tree structure.
- **Hybrid Inheritance** − A combination of multiple and multilevel inheritance so as to form a lattice structure.

The following figure depicts the examples of different types of inheritance.



Polymorphism

Polymorphism is originally a Greek word that means the ability to take multiple forms. In object-oriented paradigm, polymorphism implies using operations in different ways, depending upon the instance they are operating upon. Polymorphism allows objects with different internal structures to have a common external interface. Polymorphism is particularly effective while implementing inheritance.

Example

Let us consider two classes, Circle and Square, each with a method findArea(). Though the name and purpose of the methods in the classes are same, the internal implementation, i.e., the procedure of calculating area is different for each class. When an object of class Circle invokes its findArea() method, the operation finds the area of the circle without any conflict with the findArea() method of the Square class.

Generalization and Specialization

Generalization and specialization represent a hierarchy of relationships between classes, where subclasses inherit from super-classes.

Generalization

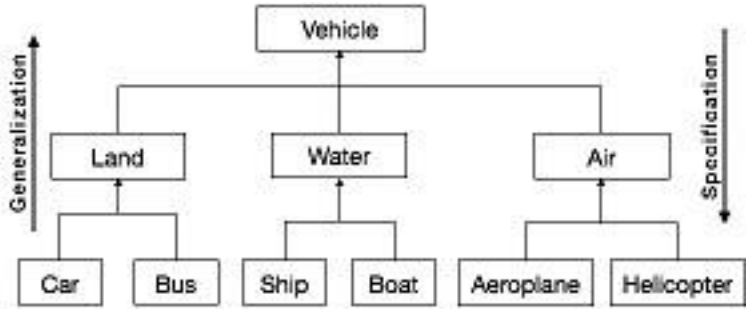
In the generalization process, the common characteristics of classes are combined to form a class in a higher level of hierarchy, i.e., subclasses are combined to form a generalized super-class. It represents an “is – a –

kind – of" relationship. For example, "car is a kind of land vehicle", or "ship is a kind of water vehicle".

Specialization

Specialization is the reverse process of generalization. Here, the distinguishing features of groups of objects are used to form specialized classes from existing classes. It can be said that the subclasses are the specialized versions of the super-class.

The following figure shows an example of generalization and specialization.



Links and Association

Link

A link represents a connection through which an object collaborates with other objects. Rumbaugh has defined it as "a physical or conceptual connection between objects". Through a link, one object may invoke the methods or navigate through another object. A link depicts the relationship between two or more objects.

Association

Association is a group of links having common structure and common behavior. Association depicts the relationship between objects of one or more classes. A link can be defined as an instance of an association.

Degree of an Association

Degree of an association denotes the number of classes involved in a connection. Degree may be unary, binary, or ternary.

- A **unary relationship** connects objects of the same class.
- A **binary relationship** connects objects of two classes.
- A **ternary relationship** connects objects of three or more classes.

Cardinality Ratios of Associations

Cardinality of a binary association denotes the number of instances participating in an association. There are three types of cardinality ratios, namely −

- **One–to–One** − A single object of class A is associated with a single object of class B.
- **One–to–Many** − A single object of class A is associated with many objects of class B.
- **Many–to–Many** − An object of class A may be associated with many objects of class B and conversely an object of class B may be associated with many objects of class A.

Aggregation or Composition

Aggregation or composition is a relationship among classes by which a class can be made up of any combination of objects of other classes. It allows objects to be placed directly within the body of other classes. Aggregation is referred as a “part–of” or “has–a” relationship, with the ability to navigate from the whole to its parts. An aggregate object is an object that is composed of one or more other objects.

Example

In the relationship, “a car has–a motor”, car is the whole object or the aggregate, and the motor is a “part–of” the car. Aggregation may denote −

- **Physical containment** − Example, a computer is composed of monitor, CPU, mouse, keyboard, and so on.
- **Conceptual containment** − Example, shareholder has–a share.

Benefits of Object Model

Now that we have gone through the core concepts pertaining to object orientation, it would be worthwhile to note the advantages that this model has to offer.

The benefits of using the object model are −

- It helps in faster development of software.
- It is easy to maintain. Suppose a module develops an error, then a programmer can fix that particular module, while the other parts of the software are still up and running.
- It supports relatively hassle-free upgrades.
- It enables reuse of objects, designs, and functions.
- It reduces development risks, particularly in integration of complex systems.

We know that the Object-Oriented Modelling (OOM) technique visualizes things in an application by using models organized around objects. Any software development approach goes through the following stages −

- Analysis,
- Design, and
- Implementation.

In object-oriented software engineering, the software developer identifies and organizes the application in terms of object-oriented concepts, prior to their final representation in any specific programming language or software tools.

Phases in Object-Oriented Software Development

The major phases of software development using object–oriented methodology are object-oriented analysis, object-oriented design, and object-oriented implementation.

Object–Oriented Analysis

In this stage, the problem is formulated, user requirements are identified, and then a model is built based upon real–world objects. The analysis produces models on how the desired system should function and how it must be developed. The models do not include any implementation details so that it can be understood and examined by any non–technical application expert.

Object–Oriented Design

Object-oriented design includes two main stages, namely, system design and object design.

System Design

In this stage, the complete architecture of the desired system is designed. The system is conceived as a set of interacting subsystems that in turn is composed of a hierarchy of interacting objects, grouped into classes. System design is done according to both the system analysis model and the proposed system architecture. Here, the emphasis is on the objects comprising the system rather than the processes in the system.

Object Design

In this phase, a design model is developed based on both the models developed in the system analysis phase and the architecture designed in the system design phase. All the classes required are identified. The designer decides whether −

- new classes are to be created from scratch,
- any existing classes can be used in their original form, or
- new classes should be inherited from the existing classes.

The associations between the identified classes are established and the hierarchies of classes are identified. Besides, the developer designs the internal details of the classes and their associations, i.e., the data structure for each attribute and the algorithms for the operations.

Object–Oriented Implementation and Testing

In this stage, the design model developed in the object design is translated into code in an appropriate programming language or software tool. The databases are created and the specific hardware requirements are ascertained. Once the code is in shape, it is tested using specialized techniques to identify and remove the errors in the code.

ANALYSIS

Analysis is a creative activity or an investigation of the problem and requirements.

Eg. To develop a Banking system

Analysis: How the system will be used?

Who are the users?

What are its functionalities?

DESIGN

Design is to provide a conceptual solution that satisfies the requirements of a given problem.

Eg. For a Book Bank System

Design: Bank(Bank name, No of Members, Address)

Student(Membership No,Name,Book Name, Amount Paid)

OBJECT ORIENTED ANALYSIS (OOA)

Object Oriented Analysis is a process of identifying classes that plays an important role in

achieving system goals and requirements.

Eg. For a Book Bank System, Classes or Objects identified are Book-details, Student-details, Membership-Details.

OBJECT ORIENTED DESIGN (OOD)

Object Oriented Design is to design the classes identified during analysis phase and to provide the relationship that exists between them that satisfies the requirements.

Eg. Book Bank System

Class name Book-Bank (Book-Name, No-of-Members, Address)

Student (Name, Membership No, Amount-Paid)

OBJECT ORIENTED ANALYSIS AND DESIGN (OOAD)

- OOAD is a Software Engineering approach that models an application by a set of Software Development Activities.

- OOAD emphasis on identifying, describing and defining the software objects and shows how they collaborate with one another to fulfill the requirements by applying the object oriented paradigm and visual modeling throughout the development life cycles.

UNIFIED PROCESS (UP)

The Unified Process has emerged as a popular iterative software development process for building object oriented systems. The Unified Process (UP) combines commonly accepted best practices, such as an iterative lifecycle and risk-driven development, into a cohesive and well-documented description. The best-known and extensively documented refinement of the Unified Process is the Rational Unified Process (RUP).

Reasons to use UP

- UP is an iterative process
- UP practices provide an example structure to talk about how to do, and how to learn OOA/D.

Best Practices and Key Concepts in UP

- Tackle high-risk and high-value issues in early iterations
- Engage users continuously for evaluation, feedback, and requirements
- Build a cohesive, core architecture in early iterations
- Apply use cases
- Provides visual modeling using UML
- Practice change request and configuration management.

UP PHASES

There are 4 phases in Unified Process,

1. Inception
2. Elaboration
3. Construction
4. Transition

INCEPTION

Inception is the initial stage of the project. Inception is not a requirements phase but it is a feasibility phase where complete investigation takes place to support a decision to continue or stop .It deals with

- Approximate vision
- Business case
- Scope
- Vague estimates

ELABORATION

In Elaboration phase the project team is expected to capture a healthy majority of the system requirements It deals with

- Refined vision,
- Iterative implementation of the core architecture,
- Resolution of high risks,
- Identification of most requirements and scope,
- Realistic estimates.

CONSTRUCTION

Construction phase encompasses on iterative implementation of the remaining lower risk and easier elements, and preparation for deployment.

TRANSITION

Transition phase focus on releasing the final product to the customers for usability.

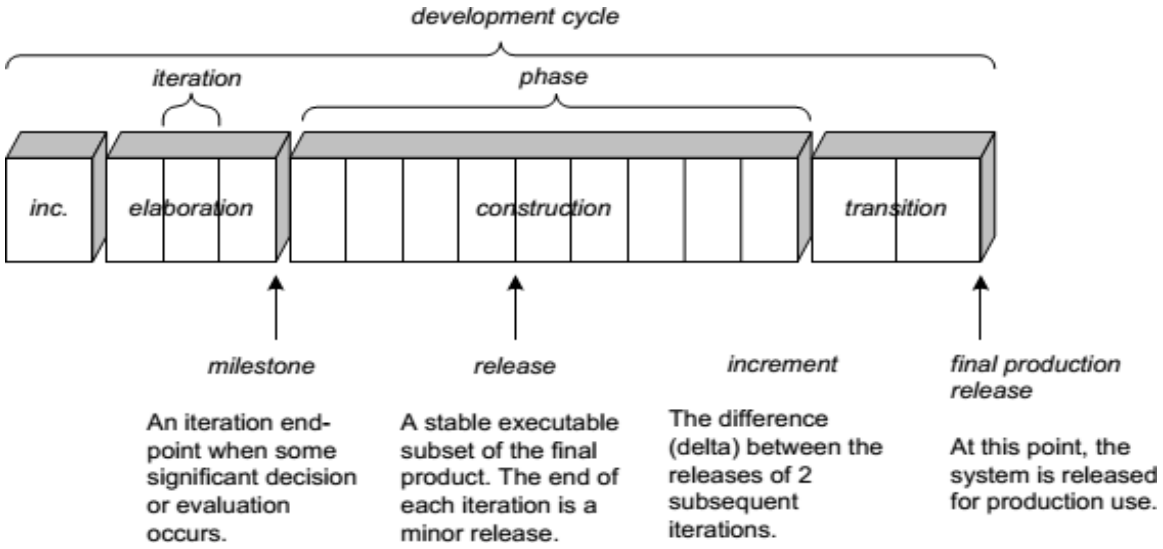


Fig: Phases of UP

UP DISCIPLINES

- UP describes work activities such as writing a use case within disciplines a set of activities and related artifacts in one subject area within requirement analysis.
- Artifact-any work such as code, web graphics, database schema, text documents, diagrams, models etc.

Several UP Disciplines

1. **Business Modeling-** Domain Model artifact to visualize concepts in the application domain.
2. **Requirements-** use case model and specification artifacts to capture functional and non-functional requirements.
3. **Design-** All aspects of design, including overall architecture, objects, databases, networking.

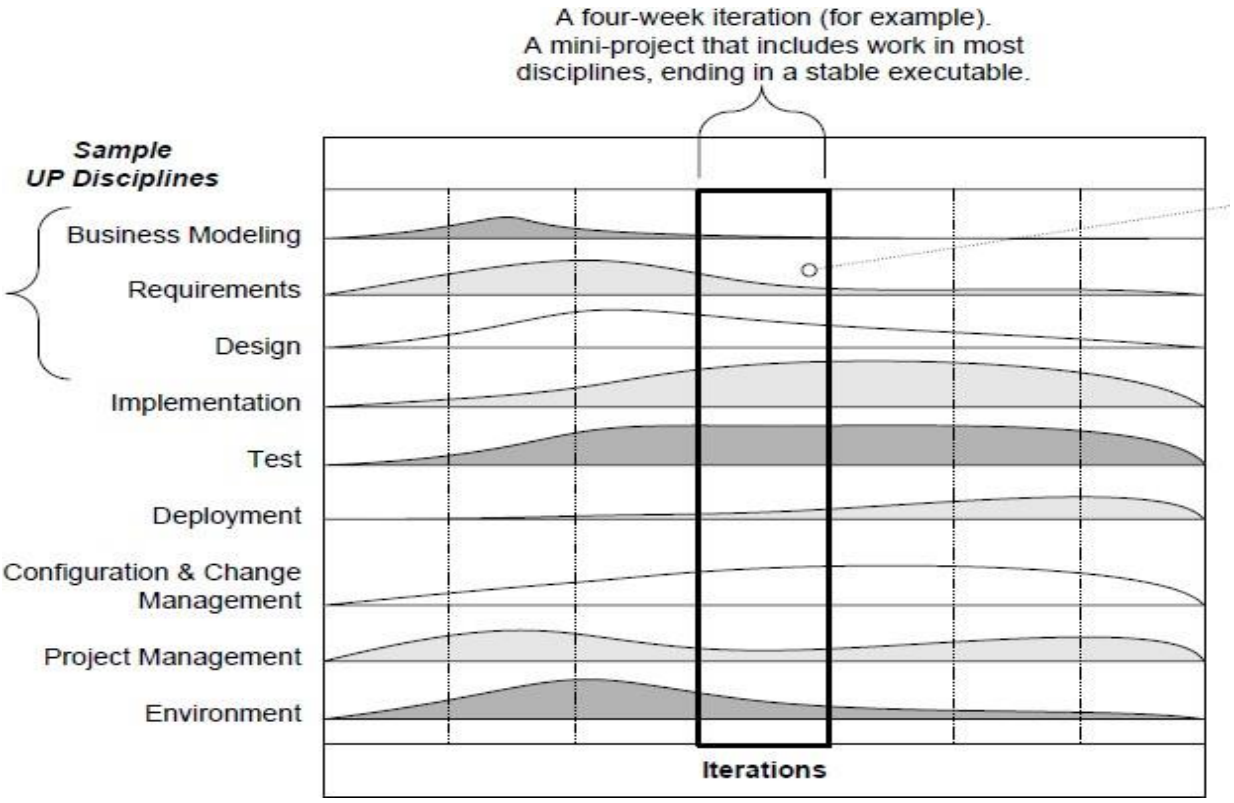


Fig: Sample UP Disciplines

UML DIAGRAMS

UML:

- Unified Modeling Language(UML) is a standard notation for the modeling of real-world objects as s first step in developing an object oriented design methodology.
- UML is a Visual language for specifying,constructing and documenting the artifacts of a system.
- The Various UML diagrams are as follows,
    i. Use Case Diagram
    ii. Class Diagram

iii. Interaction Diagram
    - Sequence Diagram
    - Collaboration Diagram or Communication Diagram
iv. State Diagram
v. Activity Diagram
vi. Package Diagram
vii. Component Diagram
viii. Deployment Diagram

Three ways to apply UML:

1. UML as sketch:
Informal and incomplete diagrams created to explore difficult parts of the problem.
2. UML as blueprint:
Detailed design diagram used for better understanding of code.
3. UML as programming language:
Complete executable specification of a software system in UML.

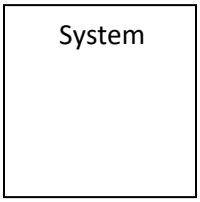
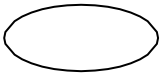
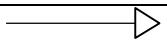
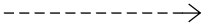
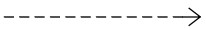
Three perspectives to apply UML:

1. Conceptual perspective: Diagrams describe the things of real world.
2. Specification perspective: Diagrams describe software abstractions or components with specifications and interfaces.
3. Implementation perspective: Diagrams describe software implementation in a particular technology.

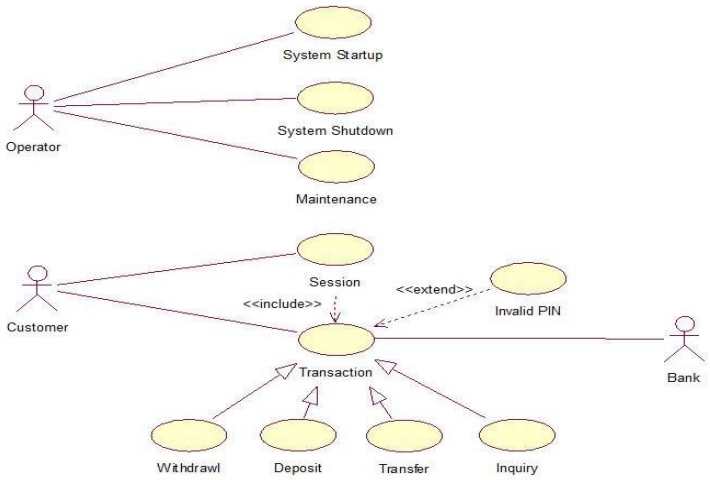
USE CASE DIAGRAM
Use case diagrams are used to describe a set of actions (use cases) that some system or systems should or can perform in collaboration with one or more external users of the system (actors). Each use case should provide some observable and valuable result to the actors or other stakeholders of the system.
Purpose:
1. Used to gather requirements of a system
2. Used to get an outside view of a system
3. Identify external and internal factors influencing the system
4. Show the interaction among the requirements through actors.

Uses:
1. Requirement analysis and high level design
2. Model the context of a system
3. Reverse engineering
4. Forward engineering

Notations:

S.No	Name	Notation	Description
1	**Actor**		Actors are the entities that interact with the system.
2	**System**		The use cases in the system make up the total requirements of the system.
3	**Use Case**		Use Case describes the actions performed by the user.
4	**Generalization**		A generalization relationship is used to represent inheritance relationship between model elements of same type.
5	**Include**	<<include>> 	An include relationship specifies how the behavior for the inclusion use case is inserted into the behavior defined for the base use case.
6	**Extend**	<<extend>> 	An extend relationship specifies how the behavior of the extension use case can be inserted into the behavior defined for the base use case.

Sample Example - ATM System



CLASS DIAGRAM:

Class diagram is a static diagram. It represents the static view of an application. The class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modeling of object oriented systems because they are the only UML diagrams which can be mapped directly with object oriented languages.

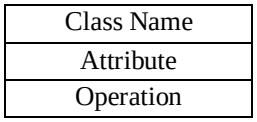
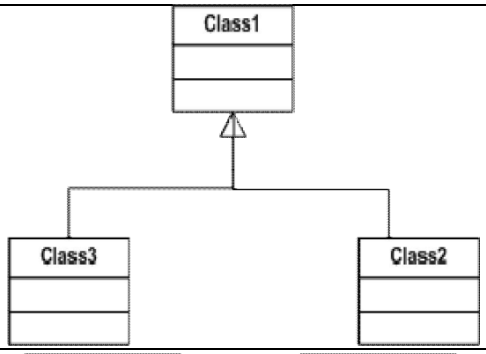
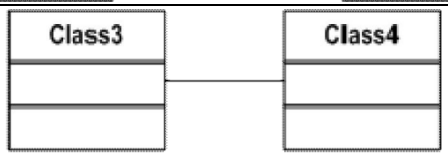
Purpose:

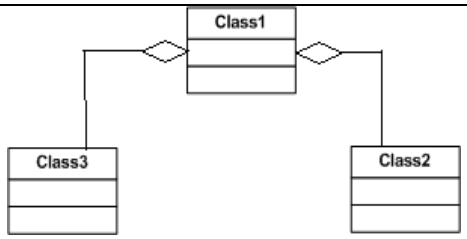
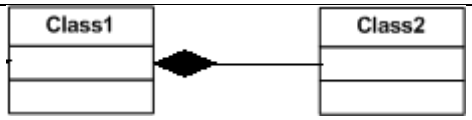
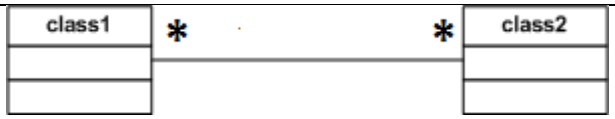
1. Analysis and design of the static view of an application
2. Describe responsibilities of a system
3. Base for Component and Deployment Diagrams
4. Forward and Reverse Engineering

Uses:

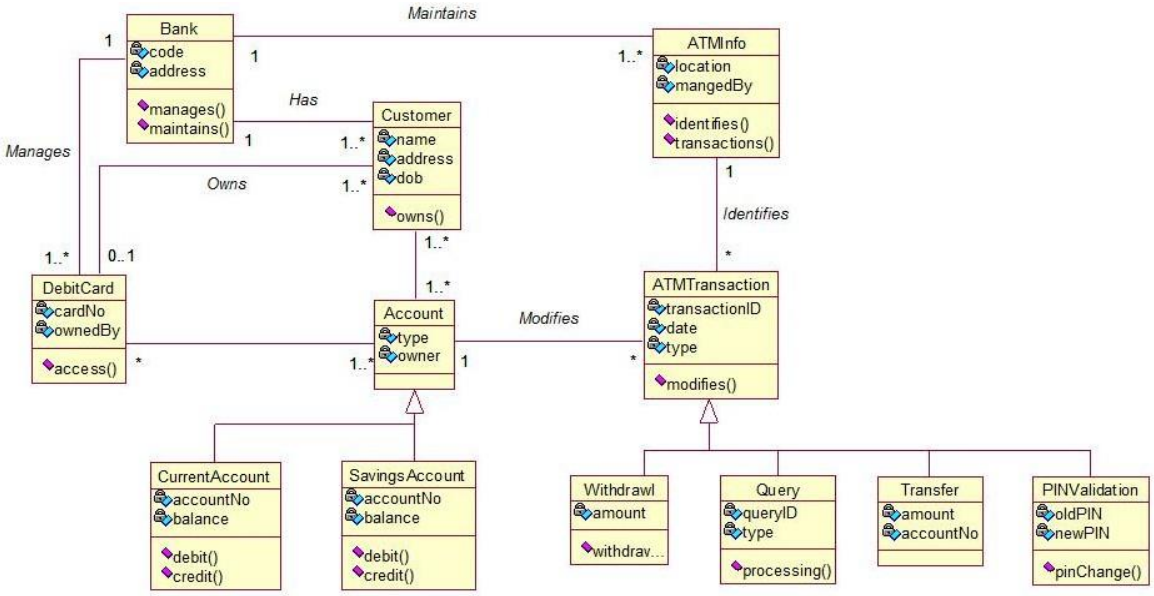
1. Describes the static view of the system
2. Shows the collaboration among the elements of the static view
3. Describes the functionalities performed by the system.
4. Construction of software applications using object oriented languages.

Notations:

S.No	Name	Notation	Description
1	**Class**		Class is an entity which describes a group of objects with same properties & behavior.
2	**Generalization**		Generalization refers to a relationship between two classes where one class is a specialized version of another.
3	**Association**		Association represent static relationships between classes.

4	**Aggregation**		Aggregation is a vague kind of association in the UML that loosely suggests whole-part relationships.
5	**Composition**		Composition is a strong kind of whole-part aggregation.
6	**Multiplicity**	 1 to 1 1 to * * to * * to 1 1 to 0… 2	Multiplicity specifies the number of instances of one class that may relate to a single instance of an associated class.

Sample Example – ATM System



INTERACTION DIAGRAM

Interaction diagrams are used to visualize the interactive behavior of the system. The Interactive behaviour is represented in UML by two diagrams namely,

- **Sequence Diagram**- It emphasizes on time sequence of messages
- **Collaboration Diagram**- It emphasizes on structural organization of the objects that send and receive messages.

Purpose:

1. To capture dynamic behaviour of a system
2. To describe the message flow in the system
3. To describe structural organization of the objects
4. To describe interaction among objects

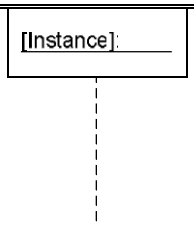
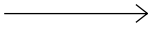
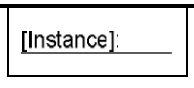
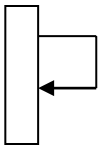
I. SEQUENCE DIAGRAM

Sequence diagram describes an interaction by focusing on the sequence of messages that are exchanged, along with their corresponding occurrence specifications on the lifelines.

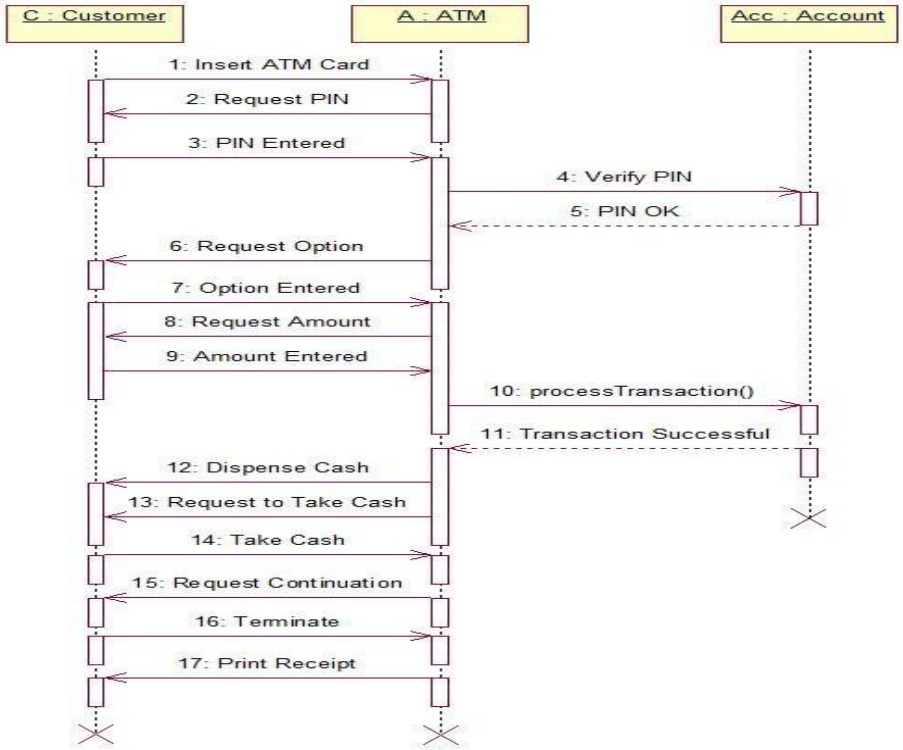
Uses:

1. To model flow of control by time sequence
2. To model flow of control by structural organizations
3. Forward engineering
4. Reverse engineering

Notations:

S.No	Name	Notation	Description
1	**Lifeline**		Lifeline represents the duration during which an object is alive and interacting with other objects in the system.
2	**Message**		To send message from one object to another.
3	**Object**		It represents the existence of an object of a particular time.
4	**Self message**		Self message is a message by the object to itself.

Sample Example – ATM System



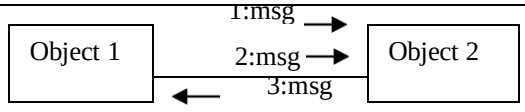
II. COLLABORATION DIAGRAM

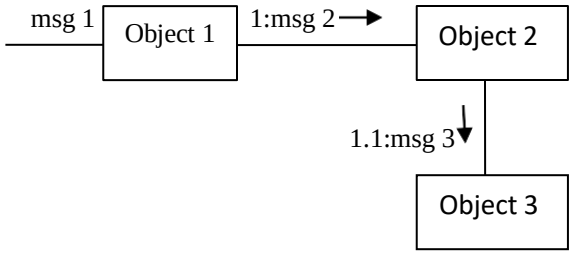
Collaboration or Communication diagram is also used to model the dynamic behaviour of the system. It emphasizes on structural organization of the objects that send and receive messages.

Uses:

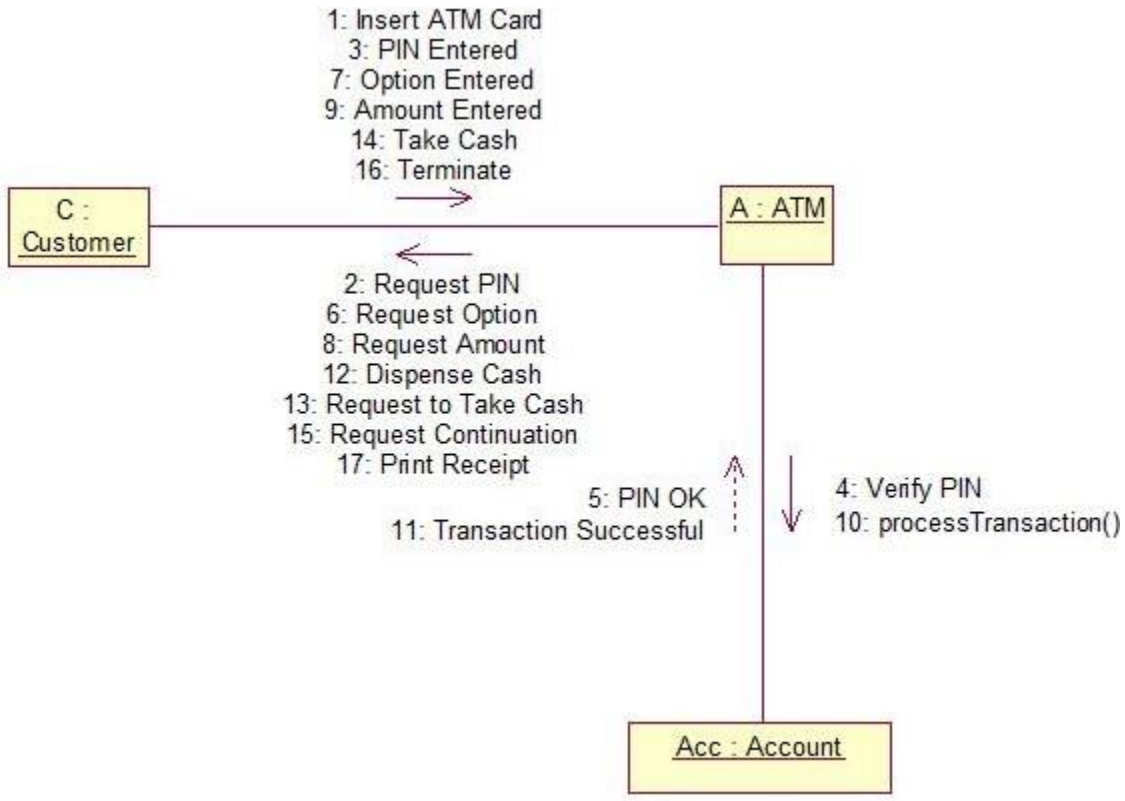
1. Used to show the messages that flow from one object to another within the system and the order in which they happen.
2. Used to track the source of the message from where it has been sent
3. Used to provide relationships and interactions among software objects

Notations:

S.No	Name	Notation	Description
1	**Link**	———	A Link is a connection path between two objects
2	**Message**		Communication between objects takes place through messages. A sequence number is added to show the sequential order of messages.

3	**Message Number Sequencing**		Numbers included along with the messages indicate the order of the message in an interaction.

Sample Example – ATM System



STATE DIAGRAM

- A State diagram is used to describe the behaviour of the systems. State diagrams require that the system described is composed of a finite number of states.
- State diagrams are used to give an abstract description of the behaviour of a system. This behaviour is analysed and represented in series of events, that could occur in one or more possible states.

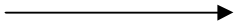
Purpose:

1. It describes dynamic behavior of the objects of the system.
2. It specifies the possible states, what transitions are allowed between states.
3. It is used to describe the dependence of the functionality on the state of the system
4. The state model describes those aspects of objects concerned with time and the sequencing of operations events.

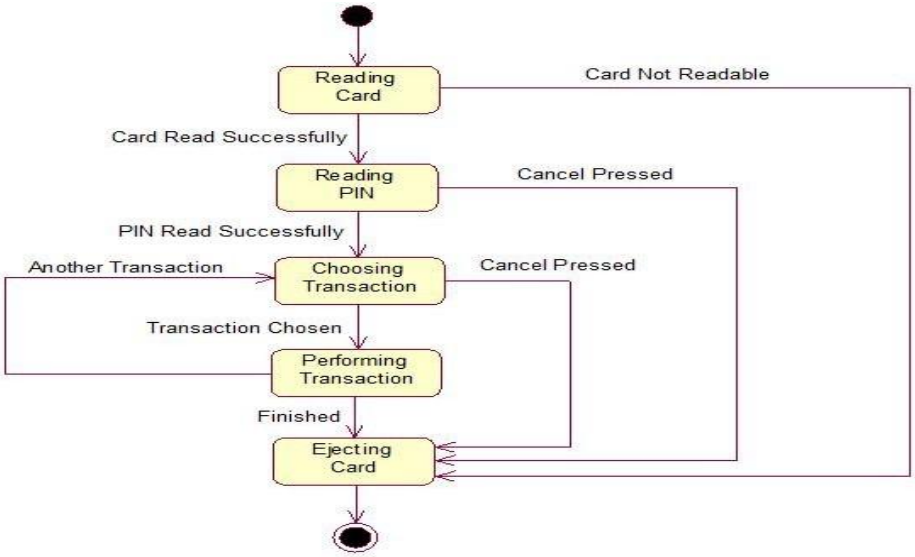
Uses:

1. To model the object states of a system.
2. To model the reactive system. Reactive system consists of reactive objects.
3. To identify the events responsible for state changes.
4. Forward and reverse engineering.

Notations:

S.No	Name	Notation	Description
1	**Initial State**		It shows the starting state of object.
2	**Final State**		It shows the terminating state of object.
3	**State**		Represents the state of object at an instant of time
4	**Transition**		A transition is a directed relationship between a source state and a target state.

Sample Example – ATM System



ACTIVITY DIAGRAM

An Activity diagram is basically a flowchart to represent the flow from one activity to another activity. Activity diagrams are typically used for business process modeling, for modeling the logic captured by a single use case or usage scenario or for modeling the detailed logic of a business rule.

Purpose:

1. Draw the activity flow of a system
2. Describe the sequence from one activity to another
3. Describe the parallel, branched and concurrent flow of the system.

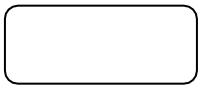
How to apply Activity Diagrams?

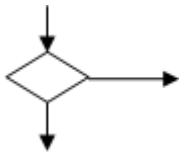
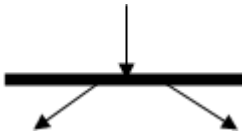
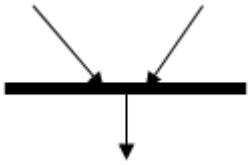
1. Activity diagrams show the flow of activities through the system.
2. Diagrams are read from top to bottom and have branches and forks to describe conditions and parallel activities.
3. A fork is used when multiple activities are occurring at the same time
4. The branch describes what activities will take place base on set of conditions
5. All branches at some point are followed by a merge to indicate the end of the conditional behavior started by that branch
6. After the merge all of the parallel activities must be combined by a join before transitioning into the final activity state.
7. Activity diagrams are applied to visualize business workflows and processes and use cases.

Uses:

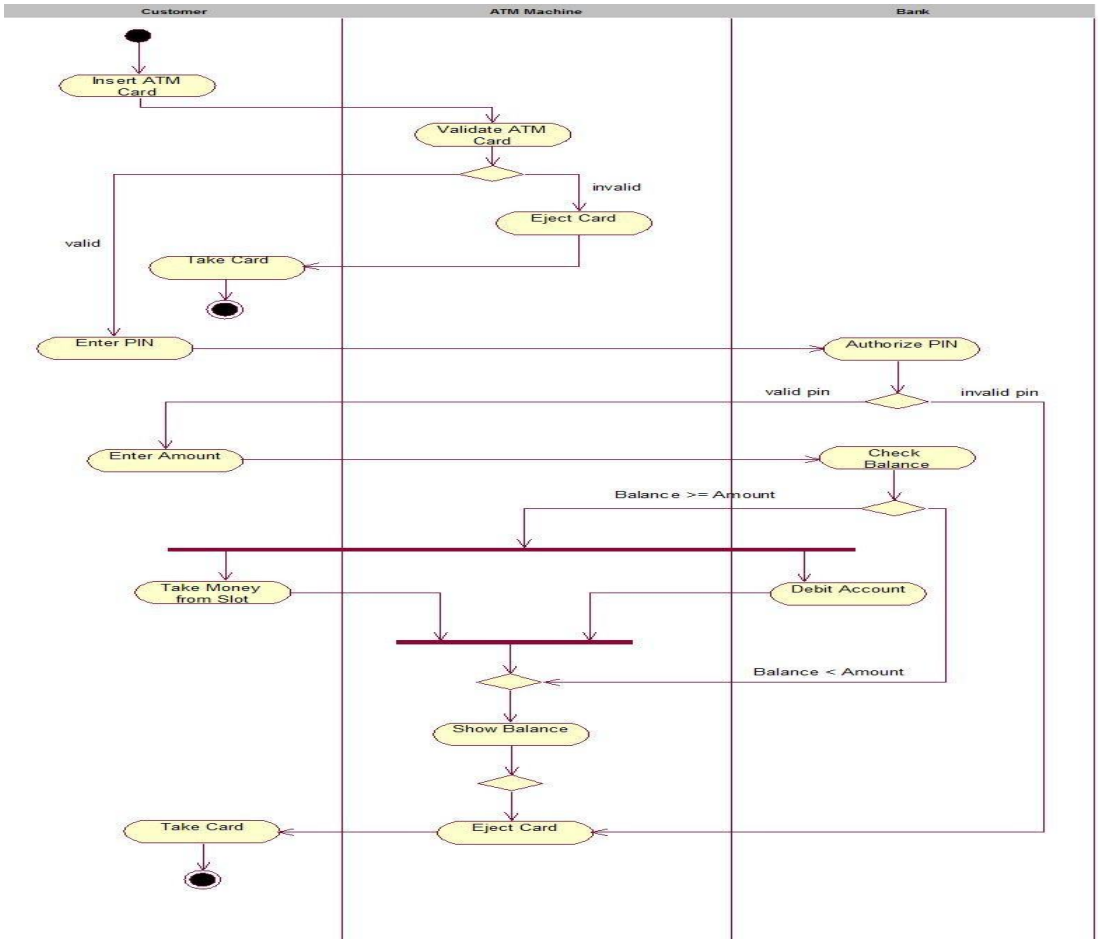
1. Visualize business processes and workflows.
2. Model work flow by using activities.
3. Model business requirements.
4. High level understanding of the system's functionalities.
5. Investigate business requirements at a later stage.

Notations:

S.No	Name	Notation	Description
1	**Activity**		Represents an individual activity of a system
2	**Initial State**		It shows the starting state of object.
3	**Final State**		It shows the terminating state of object.
4	**Transition**		Represents flow of data from one activity to another.

5	**Decision**		Decision node is a control node that accepts tokens on one or more incoming edges and selects outgoing edge from two or more outgoing flows.
6	**Fork**		A fork represents a single incoming transition and multiple outgoing transitions exhibiting parallel behavior
7	**Join**		A join in the activity diagram synchronizes the parallel behavior started at a fork.

Sample Example – ATM System



PACKAGE DIAGRAM

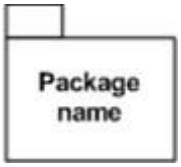
- Package diagrams organize the elements of a system into related groups to minimize dependencies among them.
- UML package diagrams are used to illustrate the logical architecture of a system, the layers, subsystems, packages etc.

Package is a namespace used to group together elements that are semantically related and might change together. It is a general purpose mechanism to organize elements into groups to provide better structure for system model.

Uses:

1. Package diagrams can use packages containing use cases to illustrate the functionality of a software system.
2. Package diagrams can use packages that represent the different layers of a software system to illustrate the layered architecture of a software system.

Notations:

S.No	Name	Notation	Description
1	**Package**		A package is a group of elements with common theme.

COMPONENT DIAGRAM

Component diagrams are used to model physical aspects of a system (elements like executables, libraries, files, documents etc.).Component diagrams are used to visualize the organization and relationships among the components in a system.

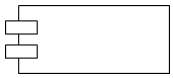
Purpose:

1. Visualize the components of a system
2. Construct executables by using forward and reverse engineering
3. Describe the organization and relationships of the components.

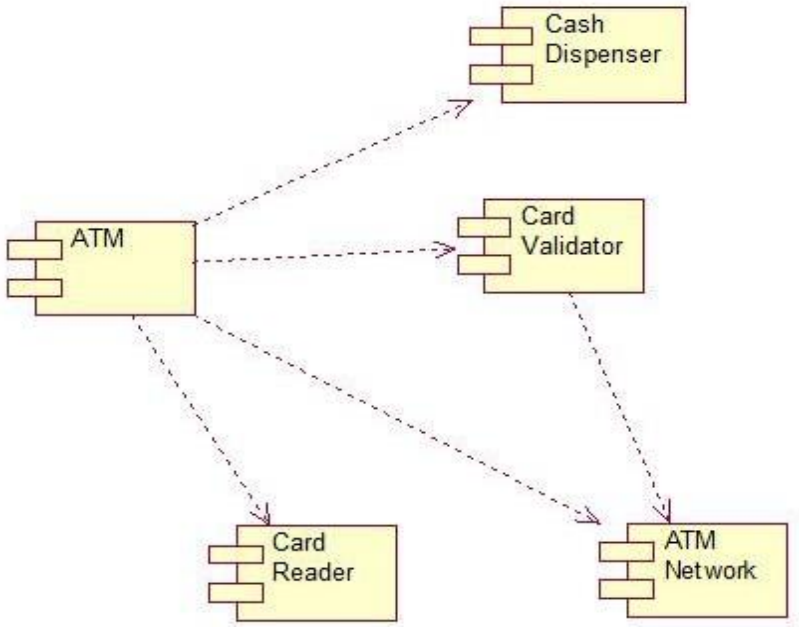
Uses:

1. Model the components of a system
2. Model database schema
3. Model executables of an application
4. Model system's source code.

Notations:

S.No	Name	Notation	Description
1	**Component**		A Component is a physical building block of the system

Sample Example – ATM System



DEPLOYMENT DIAGRAM

- Deployment diagram is defined as assignment of concrete software artifacts (executable files) to computational nodes (processing services).
- Deployment of software elements to the physical architecture and the communication (network) between physical elements.

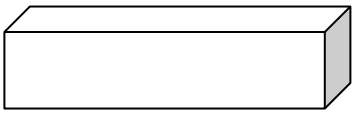
Purpose:

1. Visualize the hardware topology of a system.
2. Describe the hardware components used to deploy software components.
3. Describe the runtime processing nodes.

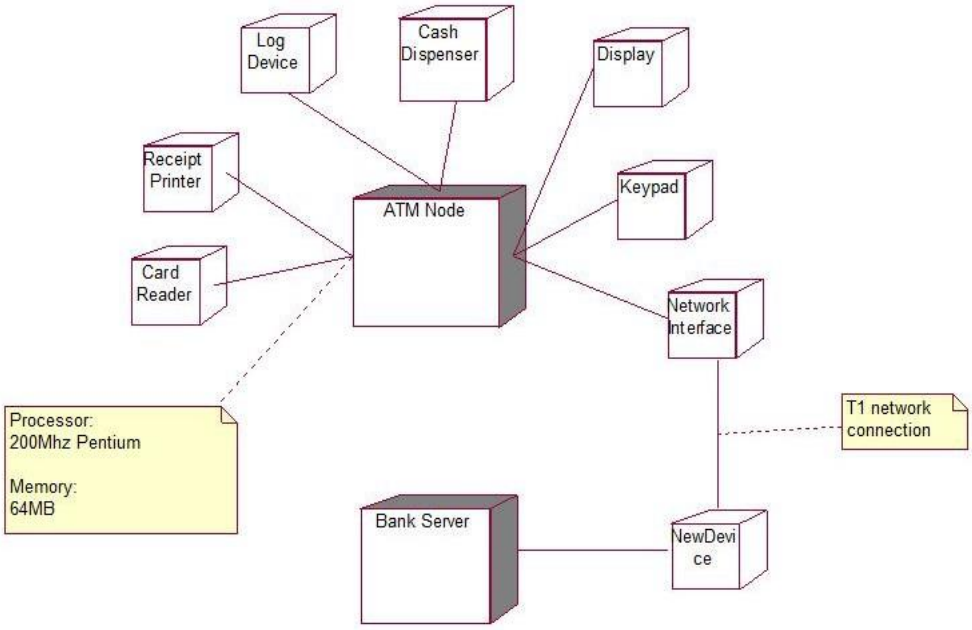
Uses:

1. To model the hardware topology of a system.
2. To model the embedded system.
3. To model the hardware details for a client/server system.
4. To model the hardware details of a distributed application.
5. Forward and Reverse engineering.

Notations:

S.No	Name	Notation	Description
1	**Node**		A single node in a deployment diagram represents multiple physical nodes, such as cluster of database servers.

Sample Example – ATM System



UNIT-4

SOFTWARE TESTING

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not.

Testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements.

This tutorial will give you a basic understanding on software testing, its types, methods, levels, and other related terminologies.

Why to Learn Software Testing?

In the IT industry, large companies have a team with responsibilities to evaluate the developed software in context of the given requirements. Moreover, developers also conduct testing which is called **Unit Testing**. In most cases, the following professionals are involved in testing a system within their respective capacities −

- Software Tester
- Software Developer
- Project Lead/Manager
- End User

Different companies have different designations for people who test the software on the basis of their experience and knowledge such as Software Tester, Software Quality Assurance Engineer, QA Analyst, etc.

Applications of Software Testing

- **Cost Effective Development** - Early testing saves both time and cost in many aspects, however reducing the cost without testing may result in improper design of a software application rendering the product useless.
- **Product Improvement** - During the SDLC phases, testing is never a time-consuming process. However diagnosing and fixing the errors identified during proper testing is a time-consuming but productive activity.
- **Test Automation** - Test Automation reduces the testing time, but it is not possible to start test automation at any time during software development. Test automaton should be started when the software has been manually tested and is stable to some extent. Moreover, test automation can never be used if requirements keep changing.
- **Quality Check** - Software testing helps in determining following set of properties of any software such as
    - Functionality
    - Reliability
    - Usability
    - Efficiency
    - Maintainability

- Portability

Audience

This tutorial is designed for software testing professionals who would like to understand the Testing Framework in detail along with its types, methods, and levels. This tutorial provides enough ingredients to start with the software testing process from where you can take yourself to higher levels of expertise.

Prerequisites

Before proceeding with this tutorial, you should have a basic understanding of the software development life cycle (SDLC). In addition, you should have a basic understanding of software programming using any programming language.

What is Testing?

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not. In simple words, testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements.

According to ANSI/IEEE 1059 standard, Testing can be defined as - A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item.

Who does Testing?

It depends on the process and the associated stakeholders of the project(s). In the IT industry, large companies have a team with responsibilities to evaluate the developed software in context of the given requirements. Moreover, developers also conduct testing which is called **Unit Testing**. In most cases, the following professionals are involved in testing a system within their respective capacities −

- Software Tester
- Software Developer
- Project Lead/Manager
- End User

Different companies have different designations for people who test the software on the basis of their experience and knowledge such as Software Tester, Software Quality Assurance Engineer, QA Analyst, etc.

It is not possible to test the software at any time during its cycle. The next two sections state when testing should be started and when to end it during the SDLC.

When to Start Testing?

An early start to testing reduces the cost and time to rework and produce error-free software that is delivered to the client. However in Software Development Life Cycle (SDLC), testing can be started from the Requirements Gathering phase and continued till the deployment of the software.

It also depends on the development model that is being used. For example, in the Waterfall model, formal testing is conducted in the testing phase; but in the incremental model, testing is performed at the end of every increment/iteration and the whole application is tested at the end.

Testing is done in different forms at every phase of SDLC −

- During the requirement gathering phase, the analysis and verification of requirements are also considered as testing.
- Reviewing the design in the design phase with the intent to improve the design is also considered as testing.
- Testing performed by a developer on completion of the code is also categorized as testing.

When to Stop Testing?

It is difficult to determine when to stop testing, as testing is a never-ending process and no one can claim that a software is 100% tested. The following aspects are to be considered for stopping the testing process −

- Testing Deadlines
- Completion of test case execution
- Completion of functional and code coverage to a certain point
- Bug rate falls below a certain level and no high-priority bugs are identified
- Management decision

Verification & Validation

These two terms are very confusing for most people, who use them interchangeably. The following table highlights the differences between verification and validation.

Sr.No.	Verification	Validation
1	Verification addresses the concern: "Are you building it right?"	Validation addresses the concern: "Are you building the right thing?"
2	Ensures that the software system meets all the functionality.	Ensures that the functionalities meet the intended behavior.
3	Verification takes place first and includes the checking for documentation, code, etc.	Validation occurs after verification and mainly involves the checking of the overall product.
4	Done by developers.	Done by testers.
5	It has static activities, as it includes collecting reviews, walkthroughs, and inspections to verify a software.	It has dynamic activities, as it includes executing the software against the requirements.

6	It is an objective process and no subjective decision should be needed to verify a software.	It is a subjective process and involves subjective decisions on how well a software works.

Many organizations around the globe develop and implement different standards to improve the quality needs of their software. This chapter briefly describes some of the widely used standards related to Quality Assurance and Testing.

ISO/IEC 9126

This standard deals with the following aspects to determine the quality of a software application −

- Quality model
- External metrics
- Internal metrics
- Quality in use metrics

This standard presents some set of quality attributes for any software such as −

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

The above-mentioned quality attributes are further divided into sub-factors, which you can study when you study the standard in detail.

ISO/IEC 9241-11

Part 11 of this standard deals with the extent to which a product can be used by specified users to achieve specified goals with Effectiveness, Efficiency and Satisfaction in a specified context of use.

This standard proposed a framework that describes the usability components and the relationship between them. In this standard, the usability is considered in terms of user performance and satisfaction. According to ISO 9241-11, usability depends on context of use and the level of usability will change as the context changes.

ISO/IEC 25000:2005

ISO/IEC 25000:2005 is commonly known as the standard that provides the guidelines for Software Quality Requirements and Evaluation (SQuaRE). This standard helps in organizing and enhancing the process related to software quality requirements and their evaluations. In reality, ISO-25000 replaces the two old ISO standards, i.e. ISO-9126 and ISO-14598.

SQuaRE is divided into sub-parts such as −

- ISO 2500n − Quality Management Division
- ISO 2501n − Quality Model Division
- ISO 2502n − Quality Measurement Division
- ISO 2503n − Quality Requirements Division
- ISO 2504n − Quality Evaluation Division

The main contents of SQuaRE are −

- Terms and definitions
- Reference Models
- General guide
- Individual division guides
- Standard related to Requirement Engineering (i.e. specification, planning, measurement and evaluation process)

ISO/IEC 12119

This standard deals with software packages delivered to the client. It does not focus or deal with the clients' production process. The main contents are related to the following items −

- Set of requirements for software packages.
- Instructions for testing a delivered software package against the specified requirements.

Miscellaneous

Some of the other standards related to QA and Testing processes are mentioned below −

Sr.No	Standard & Description
1	**IEEE 829** A standard for the format of documents used in different stages of software testing.
2	**IEEE 1061** A methodology for establishing quality requirements, identifying, implementing, analyzing, and validating the process, and product of software quality metrics.
3	**IEEE 1059** Guide for Software Verification and Validation Plans.
4	**IEEE 1008** A standard for unit testing.

5	**IEEE 1012** A standard for Software Verification and Validation.
6	**IEEE 1028** A standard for software inspections.
7	**IEEE 1044** A standard for the classification of software anomalies.
8	**IEEE 1044-1** A guide for the classification of software anomalies.
9	**IEEE 830** A guide for developing system requirements specifications.
10	**IEEE 730** A standard for software quality assurance plans.
11	**IEEE 1061** A standard for software quality metrics and methodology.
12	**IEEE 12207** A standard for software life cycle processes and life cycle data.
13	**BS 7925-1** A vocabulary of terms used in software testing.
14	**BS 7925-2** A standard for software component testing.

This section describes the different types of testing that may be used to test a software during SDLC.

Manual Testing

Manual testing includes testing a software manually, i.e., without using any automated tool or any script. In this type, the tester takes over the role of an end-user and tests the software to identify any unexpected behavior or bug. There are different stages for manual testing such as unit testing, integration testing, system testing, and user acceptance testing.

Testers use test plans, test cases, or test scenarios to test a software to ensure the completeness of testing. Manual testing also includes exploratory testing, as testers explore the software to identify errors in it.

Automation Testing

Automation testing, which is also known as Test Automation, is when the tester writes scripts and uses another software to test the product. This process involves automation of a manual process. Automation Testing is used to re-run the test scenarios that were performed manually, quickly, and repeatedly.

Apart from regression testing, automation testing is also used to test the application from load, performance, and stress point of view. It increases the test coverage, improves accuracy, and saves time and money in comparison to manual testing.

What to Automate?

It is not possible to automate everything in a software. The areas at which a user can make transactions such as the login form or registration forms, any area where large number of users can access the software simultaneously should be automated.

Furthermore, all GUI items, connections with databases, field validations, etc. can be efficiently tested by automating the manual process.

When to Automate?

Test Automation should be used by considering the following aspects of a software −

- Large and critical projects
- Projects that require testing the same areas frequently
- Requirements not changing frequently
- Accessing the application for load and performance with many virtual users
- Stable software with respect to manual testing
- Availability of time

How to Automate?

Automation is done by using a supportive computer language like VB scripting and an automated software application. There are many tools available that can be used to write automation scripts. Before mentioning the tools, let us identify the process that can be used to automate the testing process −

- Identifying areas within a software for automation
- Selection of appropriate tool for test automation
- Writing test scripts
- Development of test suits
- Execution of scripts
- Create result reports

- Identify any potential bug or performance issues

Software Testing Tools

The following tools can be used for automation testing −

- HP Quick Test Professional
- Selenium
- IBM Rational Functional Tester
- SilkTest
- TestComplete
- Testing Anywhere
- WinRunner
- LoadRunner
- Visual Studio Test Professional
- WATIR
- There are different methods that can be used for software testing. This chapter briefly describes the methods available.

Black-Box Testing

- The technique of testing without having any knowledge of the interior workings of the application is called black-box testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, while performing a black-box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.
- The following table lists the advantages and disadvantages of black-box testing.

Advantages	Disadvantages
Well suited and efficient for large code segments.	Limited coverage, since only a selected number of test scenarios is actually performed.
Code access is not required.	Inefficient testing, due to the fact that the tester only has limited knowledge about an application.
Clearly separates user's perspective from the developer's perspective through visibly defined roles.	Blind coverage, since the tester cannot target specific code segments or errorprone areas.
Large numbers of moderately skilled testers can test the application with no knowledge of implementation, programming language, or operating systems.	The test cases are difficult to design.

- White-Box Testing

- White-box testing is the detailed investigation of internal logic and structure of the code. White-box testing is also called **glass testing** or **open-box testing**. In order to perform **white-box** testing on an application, a tester needs to know the internal workings of the code.
- The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.
- The following table lists the advantages and disadvantages of white-box testing.

Advantages	Disadvantages
As the tester has knowledge of the source code, it becomes very easy to find out which type of data can help in testing the application effectively.	Due to the fact that a skilled tester is needed to perform white-box testing, the costs are increased.
It helps in optimizing the code.	Sometimes it is impossible to look into every nook and corner to find out hidden errors that may create problems, as many paths will go untested.
Extra lines of code can be removed which can bring in hidden defects.	It is difficult to maintain white-box testing, as it requires specialized tools like code analyzers and debugging tools.
Due to the tester's knowledge about the code, maximum coverage is attained during test scenario writing.	

- Grey-Box Testing

- Grey-box testing is a technique to test the application with having a limited knowledge of the internal workings of an application. In software testing, the phrase the more you know, the better carries a lot of weight while testing an application.
- Mastering the domain of a system always gives the tester an edge over someone with limited domain knowledge. Unlike black-box testing, where the tester only tests the application's user interface; in grey-box testing, the tester has access to design documents and the database. Having this knowledge, a tester can prepare better test data and test scenarios while making a test plan.

Advantages	Disadvantages
Offers combined benefits of black-box and white-box testing wherever possible.	Since the access to source code is not available, the ability to go over the code and test coverage is limited.
Grey box testers don't rely on the source code; instead they rely on interface definition	The tests can be redundant if the software

and functional specifications.	designer has already run a test case.
Based on the limited information available, a grey-box tester can design excellent test scenarios especially around communication protocols and data type handling.	Testing every possible input stream is unrealistic because it would take an unreasonable amount of time; therefore, many program paths will go untested.
The test is done from the point of view of the user and not the designer.	

- A Comparison of Testing Methods
- The following table lists the points that differentiate black-box testing, grey-box testing, and white-box testing.

Black-Box Testing	Grey-Box Testing	White-Box Testing
The internal workings of an application need not be known.	The tester has limited knowledge of the internal workings of the application.	Tester has full knowledge of the internal workings of the application.
Also known as closed-box testing, data-driven testing, or functional testing.	Also known as translucent testing, as the tester has limited knowledge of the insides of the application.	Also known as clear-box testing, structural testing, or code-based testing.
Performed by end-users and also by testers and developers.	Performed by end-users and also by testers and developers.	Normally done by testers and developers.
Testing is based on external expectations - Internal behavior of the application is unknown.	Testing is done on the basis of high-level database diagrams and data flow diagrams.	Internal workings are fully known and the tester can design test data accordingly.
It is exhaustive and the least time-consuming.	Partly time-consuming and exhaustive.	The most exhaustive and time-consuming type of testing.
Not suited for algorithm testing.	Not suited for algorithm testing.	Suited for algorithm testing.
This can only be done by	Data domains and internal boundaries can be tested, if	Data domains and internal boundaries can be better

trial-and-error method.	known.	tested.

There are different levels during the process of testing. In this chapter, a brief description is provided about these levels.

Levels of testing include different methodologies that can be used while conducting software testing. The main levels of software testing are −

- Functional Testing
- Non-functional Testing

Functional Testing

This is a type of black-box testing that is based on the specifications of the software that is to be tested. The application is tested by providing input and then the results are examined that need to conform to the functionality it was intended for. Functional testing of a software is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements.

There are five steps that are involved while testing an application for functionality.

Steps	Description
I	The determination of the functionality that the intended application is meant to perform.
II	The creation of test data based on the specifications of the application.
III	The output based on the test data and the specifications of the application.
IV	The writing of test scenarios and the execution of test cases.
V	The comparison of actual and expected results based on the executed test cases.

An effective testing practice will see the above steps applied to the testing policies of every organization and hence it will make sure that the organization maintains the strictest of standards when it comes to software quality.

Unit Testing

This type of testing is performed by developers before the setup is handed over to the testing team to formally execute the test cases. Unit testing is performed by the respective developers on the individual units of source code assigned areas. The developers use test data that is different from the test data of the quality assurance team.

The goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality.

Limitations of Unit Testing

Testing cannot catch each and every bug in an application. It is impossible to evaluate every execution path in every software application. The same is the case with unit testing.

There is a limit to the number of scenarios and test data that a developer can use to verify a source code. After having exhausted all the options, there is no choice but to stop unit testing and merge the code segment with other units.

Integration Testing

Integration testing is defined as the testing of combined parts of an application to determine if they function correctly. Integration testing can be done in two ways: Bottom-up integration testing and Top-down integration testing.

Sr.No.	Integration Testing Method
1	**Bottom-up integration** This testing begins with unit testing, followed by tests of progressively higher-level combinations of units called modules or builds.
2	**Top-down integration** In this testing, the highest-level modules are tested first and progressively, lower-level modules are tested thereafter.

In a comprehensive software development environment, bottom-up testing is usually done first, followed by top-down testing. The process concludes with multiple tests of the complete application, preferably in scenarios designed to mimic actual situations.

System Testing

System testing tests the system as a whole. Once all the components are integrated, the application as a whole is tested rigorously to see that it meets the specified Quality Standards. This type of testing is performed by a specialized testing team.

System testing is important because of the following reasons −

- System testing is the first step in the Software Development Life Cycle, where the application is tested as a whole.
- The application is tested thoroughly to verify that it meets the functional and technical specifications.
- The application is tested in an environment that is very close to the production environment where the application will be deployed.
- System testing enables us to test, verify, and validate both the business requirements as well as the application architecture.

Regression Testing

Whenever a change in a software application is made, it is quite possible that other areas within the application have been affected by this change. Regression testing is performed to verify that a fixed bug hasn't resulted in another functionality or business rule violation. The intent of regression testing is to ensure that a change, such as a bug fix should not result in another fault being uncovered in the application.

Regression testing is important because of the following reasons −

- Minimize the gaps in testing when an application with changes made has to be tested.
- Testing the new changes to verify that the changes made did not affect any other area of the application.
- Mitigates risks when regression testing is performed on the application.
- Test coverage is increased without compromising timelines.
- Increase speed to market the product.

Acceptance Testing

This is arguably the most important type of testing, as it is conducted by the Quality Assurance Team who will gauge whether the application meets the intended specifications and satisfies the client's requirement. The QA team will have a set of pre-written scenarios and test cases that will be used to test the application.

More ideas will be shared about the application and more tests can be performed on it to gauge its accuracy and the reasons why the project was initiated. Acceptance tests are not only intended to point out simple spelling mistakes, cosmetic errors, or interface gaps, but also to point out any bugs in the application that will result in system crashes or major errors in the application.

By performing acceptance tests on an application, the testing team will reduce how the application will perform in production. There are also legal and contractual requirements for acceptance of the system.

Alpha Testing

This test is the first stage of testing and will be performed amongst the teams (developer and QA teams). Unit testing, integration testing and system testing when combined together is known as alpha testing. During this phase, the following aspects will be tested in the application −

- Spelling Mistakes
- Broken Links
- Cloudy Directions
- The Application will be tested on machines with the lowest specification to test loading times and any latency problems.

Beta Testing

This test is performed after alpha testing has been successfully performed. In beta testing, a sample of the intended audience tests the application. Beta testing is also known as **pre-release testing**. Beta test versions of software are ideally distributed to a wide audience on the Web, partly to give the program a "real-world" test and partly to provide a preview of the next release. In this phase, the audience will be testing the following −

- Users will install, run the application and send their feedback to the project team.
- Typographical errors, confusing application flow, and even crashes.
- Getting the feedback, the project team can fix the problems before releasing the software to the actual users.
- The more issues you fix that solve real user problems, the higher the quality of your application will be.
- Having a higher-quality application when you release it to the general public will increase customer satisfaction.

Non-Functional Testing

This section is based upon testing an application from its non-functional attributes. Non-functional testing involves testing a software from the requirements which are nonfunctional in nature but important such as performance, security, user interface, etc.

Some of the important and commonly used non-functional testing types are discussed below.

Performance Testing

It is mostly used to identify any bottlenecks or performance issues rather than finding bugs in a software. There are different causes that contribute in lowering the performance of a software −

- Network delay
- Client-side processing
- Database transaction processing
- Load balancing between servers
- Data rendering

Performance testing is considered as one of the important and mandatory testing type in terms of the following aspects −

- Speed (i.e. Response Time, data rendering and accessing)
- Capacity
- Stability
- Scalability

Performance testing can be either qualitative or quantitative and can be divided into different sub-types such as **Load testing** and **Stress testing**.

Load Testing

It is a process of testing the behavior of a software by applying maximum load in terms of software accessing and manipulating large input data. It can be done at both normal and peak load conditions. This type of testing identifies the maximum capacity of software and its behavior at peak time.

Most of the time, load testing is performed with the help of automated tools such as Load Runner, AppLoader, IBM Rational Performance Tester, Apache JMeter, Silk Performer, Visual Studio Load Test, etc.

Virtual users (VUsers) are defined in the automated testing tool and the script is executed to verify the load testing for the software. The number of users can be increased or decreased concurrently or incrementally based upon the requirements.

Stress Testing

Stress testing includes testing the behavior of a software under abnormal conditions. For example, it may include taking away some resources or applying a load beyond the actual load limit.

The aim of stress testing is to test the software by applying the load to the system and taking over the resources used by the software to identify the breaking point. This testing can be performed by testing different scenarios such as −

- Shutdown or restart of network ports randomly
- Turning the database on or off
- Running different processes that consume resources such as CPU, memory, server, etc.

Usability Testing

Usability testing is a black-box technique and is used to identify any error(s) and improvements in the software by observing the users through their usage and operation.

According to Nielsen, usability can be defined in terms of five factors, i.e. efficiency of use, learn-ability, memory-ability, errors/safety, and satisfaction. According to him, the usability of a product will be good and the system is usable if it possesses the above factors.

Nigel Bevan and Macleod considered that usability is the quality requirement that can be measured as the outcome of interactions with a computer system. This requirement can be fulfilled and the end-user will be satisfied if the intended goals are achieved effectively with the use of proper resources.

Molich in 2000 stated that a user-friendly system should fulfill the following five goals, i.e., easy to Learn, easy to remember, efficient to use, satisfactory to use, and easy to understand.

In addition to the different definitions of usability, there are some standards and quality models and methods that define usability in the form of attributes and sub-attributes such as ISO-9126, ISO-9241-11, ISO-13407, and IEEE std.610.12, etc.

UI vs Usability Testing

UI testing involves testing the Graphical User Interface of the Software. UI testing ensures that the GUI functions according to the requirements and tested in terms of color, alignment, size, and other properties.

On the other hand, usability testing ensures a good and user-friendly GUI that can be easily handled. UI testing can be considered as a sub-part of usability testing.

Security Testing

Security testing involves testing a software in order to identify any flaws and gaps from security and vulnerability point of view. Listed below are the main aspects that security testing should ensure −

- Confidentiality
- Integrity

- Authentication
- Availability
- Authorization
- Non-repudiation
- Software is secure against known and unknown vulnerabilities
- Software data is secure
- Software is according to all security regulations
- Input checking and validation
- SQL insertion attacks
- Injection flaws
- Session management issues
- Cross-site scripting attacks
- Buffer overflows vulnerabilities
- Directory traversal attacks

Portability Testing

Portability testing includes testing a software with the aim to ensure its reusability and that it can be moved from another software as well. Following are the strategies that can be used for portability testing −

- Transferring an installed software from one computer to another.
- Building executable (.exe) to run the software on different platforms.

Portability testing can be considered as one of the sub-parts of system testing, as this testing type includes overall testing of a software with respect to its usage over different environments. Computer hardware, operating systems, and browsers are the major focus of portability testing. Some of the pre-conditions for portability testing are as follows −

- Software should be designed and coded, keeping in mind the portability requirements.
- Unit testing has been performed on the associated components.
- Integration testing has been performed.
- Test environment has been established.

Testing documentation involves the documentation of artifacts that should be developed before or during the testing of Software.

Documentation for software testing helps in estimating the testing effort required, test coverage, requirement tracking/tracing, etc. This section describes some of the commonly used documented artifacts related to software testing such as −

- Test Plan
- Test Scenario
- Test Case

- Traceability Matrix

Test Plan

A test plan outlines the strategy that will be used to test an application, the resources that will be used, the test environment in which testing will be performed, and the limitations of the testing and the schedule of testing activities. Typically the Quality Assurance Team Lead will be responsible for writing a Test Plan.

A test plan includes the following −

- Introduction to the Test Plan document
- Assumptions while testing the application
- List of test cases included in testing the application
- List of features to be tested
- What sort of approach to use while testing the software
- List of deliverables that need to be tested
- The resources allocated for testing the application
- Any risks involved during the testing process
- A schedule of tasks and milestones to be achieved

Test Scenario

It is a one line statement that notifies what area in the application will be tested. Test scenarios are used to ensure that all process flows are tested from end to end. A particular area of an application can have as little as one test scenario to a few hundred scenarios depending on the magnitude and complexity of the application.

The terms 'test scenario' and 'test cases' are used interchangeably, however a test scenario has several steps, whereas a test case has a single step. Viewed from this perspective, test scenarios are test cases, but they include several test cases and the sequence that they should be executed. Apart from this, each test is dependent on the output from the previous test.

Test Case

Test cases involve a set of steps, conditions, and inputs that can be used while performing testing tasks. The main intent of this activity is to ensure whether a software passes or fails in terms of its functionality and other aspects. There are many types of test cases such as functional, negative, error, logical test cases, physical test cases, UI test cases, etc.

Furthermore, test cases are written to keep track of the testing coverage of a software. Generally, there are no formal templates that can be used during test case writing. However, the following components are always available and included in every test case −

- Test case ID
- Product module
- Product version

- Revision history
- Purpose
- Assumptions
- Pre-conditions
- Steps
- Expected outcome
- Actual outcome
- Post-conditions

Many test cases can be derived from a single test scenario. In addition, sometimes multiple test cases are written for a single software which are collectively known as test suites.

Traceability Matrix

Traceability Matrix (also known as Requirement Traceability Matrix - RTM) is a table that is used to trace the requirements during the Software Development Life Cycle. It can be used for forward tracing (i.e. from Requirements to Design or Coding) or backward (i.e. from Coding to Requirements). There are many user-defined templates for RTM.

Each requirement in the RTM document is linked with its associated test case so that testing can be done as per the mentioned requirements. Furthermore, Bug ID is also included and linked with its associated requirements and test case. The main goals for this matrix are −

- Make sure the software is developed as per the mentioned requirements.
- Helps in finding the root cause of any bug.
- Helps in tracing the developed documents during different phases of SDLC.

What is Structural Testing ?

Structural testing, also known as glass box testing or white box testing is an approach where the tests are derived from the knowledge of the software's structure or internal implementation.

The other names of structural testing includes clear box testing, open box testing, logic driven testing or path driven testing.

Structural Testing Techniques:

- **Statement Coverage -** This technique is aimed at exercising all programming statements with minimal tests.
- **Branch Coverage -** This technique is running a series of tests to ensure that all branches are tested at least once.
- **Path Coverage -** This technique corresponds to testing all possible paths which means that each statement and branch are covered.

Calculating Structural Testing Effectiveness:

```
Statement Testing = (Number of Statements Exercised / Total Number of Statements)
x 100 %
```

```
Branch Testing = (Number of decisions outcomes tested / Total Number of decision
Outcomes) x 100 %

Path Coverage = (Number paths exercised / Total Number of paths in the program) x
100 %
```

Advantages of Structural Testing:

- Forces test developer to reason carefully about implementation
- Reveals errors in "hidden" code
- Spots the Dead Code or other issues with respect to best programming practices.

Disadvantages of Structural Box Testing:

- Expensive as one has to spend both time and money to perform white box testing.
- Every possibility that few lines of code is missed accidentally.
- Indepth knowledge about the programming language is necessary to perform white box testing.

What is Structured Walkthrough?

A structured walkthrough, a static testing technique performed in an organized manner between a group of peers to review and discuss the technical aspects of software development process. The main objective in a structured walkthrough is to find defects inorder to improve the quality of the product.

Structured walkthroughs are usually NOT used for technical discussions or to discuss the solutions for the issues found. As explained, the aim is to detect error and not to correct errors. When the walkthrough is finished, the author of the output is responsible for fixing the issues.

Benefits:

- Saves time and money as defects are found and rectified very early in the lifecycle.
- This provides value-added comments from reviewers with different technical backgrounds and experience.
- It notifies the project management team about the progress of the development process.
- It creates awareness about different development or maintenance methodologies which can provide a professional growth to participants.

Structured Walkthrough Participants:

- **Author -** The Author of the document under review.
- **Presenter -** The presenter usually develops the agenda for the walkthrough and presents the output being reviewed.
- **Moderator -** The moderator facilitates the walkthrough session, ensures the walkthrough agenda is followed, and encourages all the reviewers to participate.

- **Reviewers -** The reviewers evaluate the document under test to determine if it is technically accurate.
- **Scribe -** The scribe is the recorder of the structured walkthrough outcomes who records the issues identified and any other technical comments, suggestions, and unresolved questions.

What is Mutation Testing?

Mutation testing is a structural testing technique, which uses the structure of the code to guide the testing process. On a very high level, it is the process of rewriting the source code in small ways in order to remove the redundancies in the source code

These ambiguities might cause failures in the software if not fixed and can easily pass through testing phase undetected.

Mutation Testing Benefits:

Following benefits are experienced, if mutation testing is adopted:

- It brings a whole new kind of errors to the developer's attention.
- It is the most powerful method to detect hidden defects, which might be impossible to identify using the conventional testing techniques.
- Tools such as Insure++ help us to find defects in the code using the state-of-the-art.
- Increased customer satisfaction index as the product would be less buggy.
- Debugging and Maintaining the product would be more easier than ever.

Mutation Testing Types:

- **Value Mutations:** An attempt to change the values to detect errors in the programs. We usually change one value to a much larger value or one value to a much smaller value. The most common strategy is to change the constants.
- **Decision Mutations:** The decisions/conditions are changed to check for the design errors. Typically, one changes the arithmetic operators to locate the defects and also we can consider mutating all relational operators and logical operators (AND, OR , NOT)
- **Statement Mutations:** Changes done to the statements by deleting or duplicating the line which might arise when a developer is copy pasting the code from somewhere else.

Software Maintenance

Software maintenance is widely accepted part of SDLC now a days. It stands for all the modifications and updations done after the delivery of software product. There are number of reasons, why modifications are required, some of them are briefly mentioned below:

- **Market Conditions** - Policies, which changes over the time, such as taxation and newly introduced constraints like, how to maintain bookkeeping, may trigger need for modification.
- **Client Requirements** - Over the time, customer may ask for new features or functions in the software.
- **Host Modifications** - If any of the hardware and/or platform (such as operating system) of the target host changes, software changes are needed to keep adaptability.

- **Organization Changes** - If there is any business level change at client end, such as reduction of organization strength, acquiring another company, organization venturing into new business, need to modify in the original software may arise.

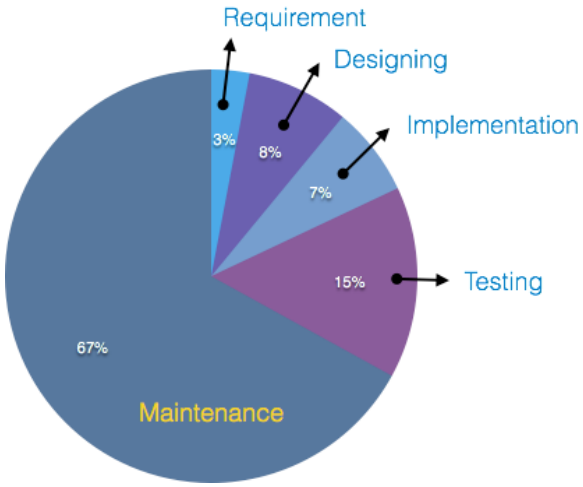
Types of maintenance

In a software lifetime, type of maintenance may vary based on its nature. It may be just a routine maintenance tasks as some bug discovered by some user or it may be a large event in itself based on maintenance size or nature. Following are some types of maintenance based on their characteristics:

- **Corrective Maintenance** - This includes modifications and updations done in order to correct or fix problems, which are either discovered by user or concluded by user error reports.
- **Adaptive Maintenance** - This includes modifications and updations applied to keep the software product up-to date and tuned to the ever changing world of technology and business environment.
- **Perfective Maintenance** - This includes modifications and updates done in order to keep the software usable over long period of time. It includes new features, new user requirements for refining the software and improve its reliability and performance.
- **Preventive Maintenance** - This includes modifications and updations to prevent future problems of the software. It aims to attend problems, which are not significant at this moment but may cause serious issues in future.

Cost of Maintenance

Reports suggest that the cost of maintenance is high. A study on estimating software maintenance found that the cost of maintenance is as high as 67% of the cost of entire software process cycle.



On an average, the cost of software maintenance is more than 50% of all SDLC phases. There are various factors, which trigger maintenance cost go high, such as:

Real-world factors affecting Maintenance Cost

- The standard age of any software is considered up to 10 to 15 years.

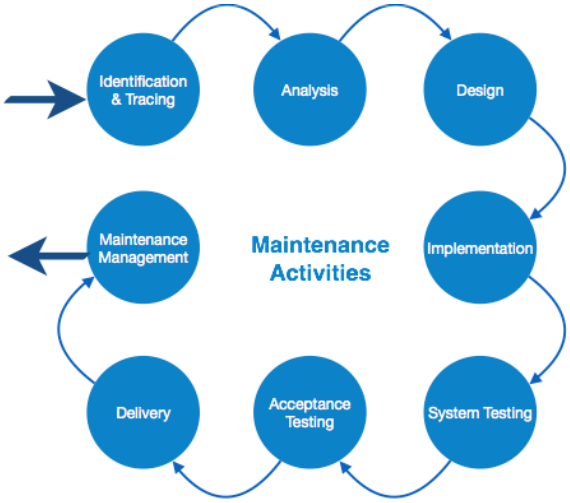
- Older softwares, which were meant to work on slow machines with less memory and storage capacity cannot keep themselves challenging against newly coming enhanced softwares on modern hardware.
- As technology advances, it becomes costly to maintain old software.
- Most maintenance engineers are newbie and use trial and error method to rectify problem.
- Often, changes made can easily hurt the original structure of the software, making it hard for any subsequent changes.
- Changes are often left undocumented which may cause more conflicts in future.

Software-end factors affecting Maintenance Cost

- Structure of Software Program
- Programming Language
- Dependence on external environment
- Staff reliability and availability

Maintenance Activities

IEEE provides a framework for sequential maintenance process activities. It can be used in iterative manner and can be extended so that customized items and processes can be included.



These activities go hand-in-hand with each of the following phase:

- **Identification & Tracing** - It involves activities pertaining to identification of requirement of modification or maintenance. It is generated by user or system may itself report via logs or error messages.Here, the maintenance type is classified also.
- **Analysis** - The modification is analyzed for its impact on the system including safety and security implications. If probable impact is severe, alternative solution is looked for. A set of required modifications is then materialized into requirement specifications. The cost of modification/maintenance is analyzed and estimation is concluded.
- **Design** - New modules, which need to be replaced or modified, are designed against requirement specifications set in the previous stage. Test cases are created for validation and verification.

- **Implementation** - The new modules are coded with the help of structured design created in the design step.Every programmer is expected to do unit testing in parallel.
- **System Testing** - Integration testing is done among newly created modules. Integration testing is also carried out between new modules and the system. Finally the system is tested as a whole, following regressive testing procedures.
- **Acceptance Testing** - After testing the system internally, it is tested for acceptance with the help of users. If at this state, user complaints some issues they are addressed or noted to address in next iteration.
- **Delivery** - After acceptance test, the system is deployed all over the organization either by small update package or fresh installation of the system. The final testing takes place at client end after the software is delivered.

  Training facility is provided if required, in addition to the hard copy of user manual.
- **Maintenance management** - Configuration management is an essential part of system maintenance. It is aided with version control tools to control versions, semi-version or patch management.

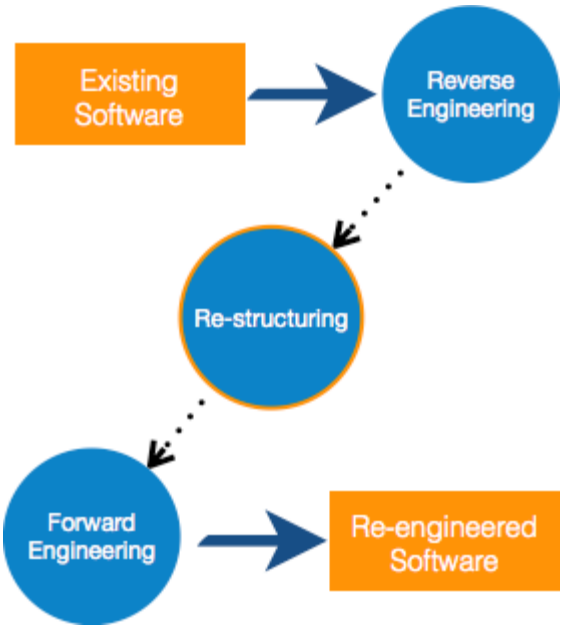
Software Re-engineering

When we need to update the software to keep it to the current market, without impacting its functionality, it is called software re-engineering. It is a thorough process where the design of software is changed and programs are re-written.

Legacy software cannot keep tuning with the latest technology available in the market. As the hardware become obsolete, updating of software becomes a headache. Even if software grows old with time, its functionality does not.

For example, initially Unix was developed in assembly language. When language C came into existence, Unix was re-engineered in C, because working in assembly language was difficult.

Other than this, sometimes programmers notice that few parts of software need more maintenance than others and they also need re-engineering.



Re-Engineering Process

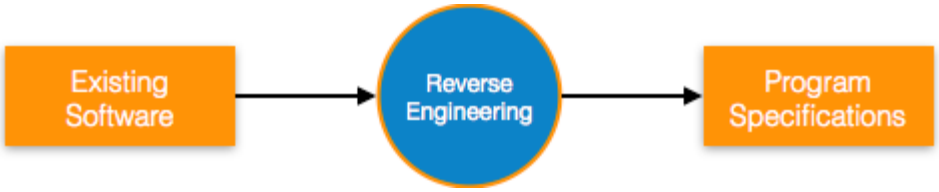
- **Decide** what to re-engineer. Is it whole software or a part of it?
- **Perform** Reverse Engineering, in order to obtain specifications of existing software.
- **Restructure Program** if required. For example, changing function-oriented programs into object-oriented programs.
- **Re-structure data** as required.
- **Apply Forward engineering** concepts in order to get re-engineered software.

There are few important terms used in Software re-engineering

Reverse Engineering

It is a process to achieve system specification by thoroughly analyzing, understanding the existing system. This process can be seen as reverse SDLC model, i.e. we try to get higher abstraction level by analyzing lower abstraction levels.

An existing system is previously implemented design, about which we know nothing. Designers then do reverse engineering by looking at the code and try to get the design. With design in hand, they try to conclude the specifications. Thus, going in reverse from code to system specification.



Program Restructuring

It is a process to re-structure and re-construct the existing software. It is all about re-arranging the source code, either in same programming language or from one programming language to a different one. Restructuring can have either source code-restructuring and data-restructuring or both.

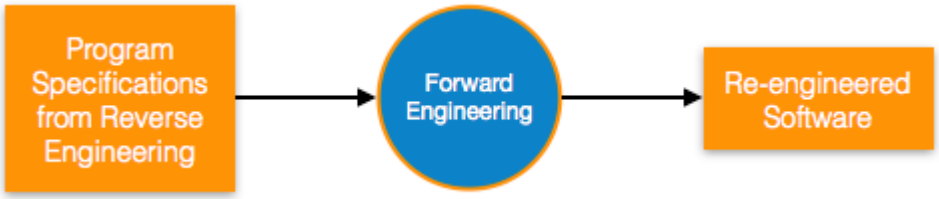
Re-structuring does not impact the functionality of the software but enhance reliability and maintainability. Program components, which cause errors very frequently can be changed, or updated with re-structuring.

The dependability of software on obsolete hardware platform can be removed via re-structuring.

Forward Engineering

Forward engineering is a process of obtaining desired software from the specifications in hand which were brought down by means of reverse engineering. It assumes that there was some software engineering already done in the past.

Forward engineering is same as software engineering process with only one difference – it is carried out always after reverse engineering.



Component reusability

A component is a part of software program code, which executes an independent task in the system. It can be a small module or sub-system itself.

Example

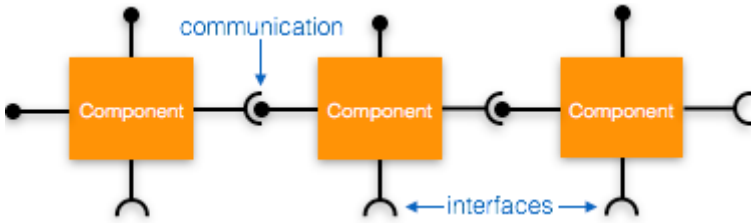
The login procedures used on the web can be considered as components, printing system in software can be seen as a component of the software.

Components have high cohesion of functionality and lower rate of coupling, i.e. they work independently and can perform tasks without depending on other modules.

In OOP, the objects are designed are very specific to their concern and have fewer chances to be used in some other software.

In modular programming, the modules are coded to perform specific tasks which can be used across number of other software programs.

There is a whole new vertical, which is based on re-use of software component, and is known as Component Based Software Engineering (CBSE).



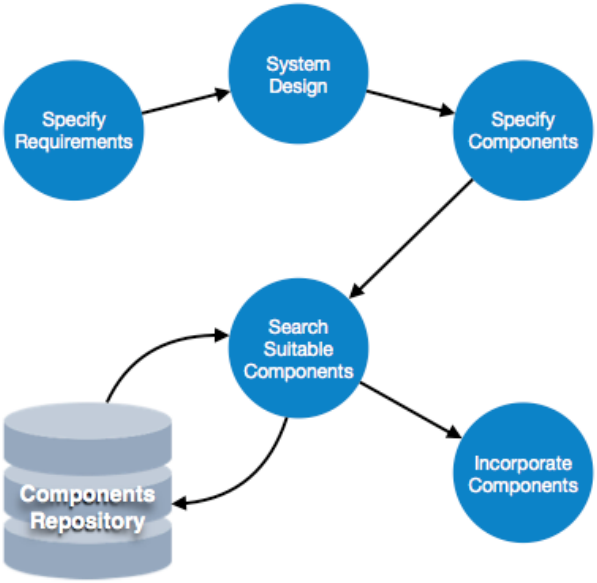
Re-use can be done at various levels

- **Application level** - Where an entire application is used as sub-system of new software.
- **Component level** - Where sub-system of an application is used.
- **Modules level** - Where functional modules are re-used.

  Software components provide interfaces, which can be used to establish communication among different components.

Reuse Process

Two kinds of method can be adopted: either by keeping requirements same and adjusting components or by keeping components same and modifying requirements.



- **Requirement Specification** - The functional and non-functional requirements are specified, which a software product must comply to, with the help of existing system, user input or both.
- **Design** - This is also a standard SDLC process step, where requirements are defined in terms of software parlance. Basic architecture of system as a whole and its sub-systems are created.
- **Specify Components** - By studying the software design, the designers segregate the entire system into smaller components or sub-systems. One complete software design turns into a collection of a huge set of components working together.
- **Search Suitable Components** - The software component repository is referred by designers to search for the matching component, on the basis of functionality and intended software requirements..
- **Incorporate Components** - All matched components are packed together to shape them as complete software.

What is Regression Testing?

Regression testing a black box testing technique that consists of re-executing those tests that are impacted by the code changes. These tests should be executed as often as possible throughout the software development life cycle.

Types of Regression Tests:

- **Final Regression Tests: -** A "final regression testing" is performed to validate the build that hasn't changed for a period of time. This build is deployed or shipped to customers.
- **Regression Tests: -** A normal regression testing is performed to verify if the build has NOT broken any other parts of the application by the recent code changes for defect fixing or for enhancement.

Selecting Regression Tests:

- Requires knowledge about the system and how it affects by the existing functionalities.
- Tests are selected based on the area of frequent defects.

- Tests are selected to include the area, which has undergone code changes many a times.
- Tests are selected based on the criticality of the features.

Regression Testing Steps:

Regression tests are the ideal cases of automation which results in better **R**eturn **O**n **I**nvestment (ROI).

- Select the Tests for Regression.
- Choose the apt tool and automate the Regression Tests
- Verify applications with Checkpoints
- Manage Regression Tests/update when required
- Schedule the tests
- Integrate with the builds
- Analyze the results

What is User Interface Testing?

User interface testing, a testing technique used to identify the presence of defects is a product/software under test by using Graphical user interface [GUI].

GUI Testing - Characteristics:

- GUI is a hierarchical, graphical front end to the application, contains graphical objects with a set of properties.
- During execution, the values of the properties of each objects of a GUI define the GUI state.
- It has capabilities to exercise GUI events like key press/mouse click.
- Able to provide inputs to the GUI Objects.
- To check the GUI representations to see if they are consistent with the expected ones.
- It strongly depends on the used technology.

GUI Testing - Approaches:

- **Manual Based -** Based on the domain and application knowledge of the tester.
- **Capture and Replay -** Based on capture and replay of user actions.
- **Model-based testing -** Based on the execution of user sessions based on a GUI model. Various GUI models are briefly discussed below.

Model Based Testing - In Brief:

- **Event-based model -** Based on all events of the GUI need to be executed at least once.
- **State-based model -** "all states" of the GUI are to be exercised at least once.

- **Domain model -** Based on the application domain and its functionality.

GUI Testing Checklist:

- Check Screen Validations
- Verify All Navigations
- Check usability Conditions
- Verify Data Integrity
- Verify the object states
- Verify the date Field and Numeric Field Formats

GUI Automation Tools

Following are some of the open source GUI automation tools in the market:

Product	Licensed Under	URL
AutoHotkey	GPL	http://www.autohotkey.com/
Selenium	Apache	http://docs.seleniumhq.org/
Sikuli	MIT	http://sikuli.org
Robot Framework	Apache	www.robotframework.org
watir	BSD	http://www.watir.com/
Dojo Toolkit	BSD	http://dojotoolkit.org/

Following are some of the Commercial GUI automation tools in the market.

Product	Vendor	URL
AutoIT	AutoIT	http://www.autoitscript.com/site/autoit/
EggPlant	TestPlant	www.testplant.com
QTP	Hp	http://www8.hp.com/us/en/software-solutions/

Rational Functional Tester	IBM	http://www-03.ibm.com/software/products/us/en/functional
Infragistics	Infragistics	www.infragistics.com
iMacros	iOpus	http://www.iopus.com/iMacros/
CodedUI	Microsoft	http://www.microsoft.com/visualstudio/
Sikuli	Micro Focus International	http://www.microfocus.com/

UNIT V PATENTS 6

Patents – objectives and benefits of patent, Concept, features of patent, Inventive step, Specification, Types of patent application, process E-filling, Examination of patent, Grant of patent, Revocation, Equitable Assignments, Licences, Licensing of related patents, patent agents, Registration of patent agents.

TOTAL: 30 PERIODS

REFERENCES:

1. Cooper Donald R, Schindler Pamela S and Sharma JK, "Business Research Methods", Tata McGraw Hill Education, 11e (2012).
2. Catherine J. Holland, "Intellectual property: Patents, Trademarks, Copyrights, Trade Secrets", Entrepreneur Press, 2007.
3. David Hunt, Long Nguyen, Matthew Rodgers, "Patent searching: tools & techniques", Wiley, 2007.
4. The Institute of Company Secretaries of India, Statutory body under an Act of parliament, "Professional Programme Intellectual Property Rights, Law and practice", September 2013.

Course Outcomes:

At the end of this course, the students will have the ability to

1. Formulate and Design research problem
2. Understand and Comprehend the Data Collection Methods
3. Perform Data analysis and acquire Insights
4. Understand IPR and follow research ethics
5. Understand and Practice Drafting and filing a Patent in research and development

O-PO Mapping:

CO	PO					
	1	**2**	**3**	**4**	**5**	**6**
1	3	3	-	1	-	1
2	3	2	-	2	-	1
3	3	2	2	2	-	1
4	3	2	-	1	-	-
5	3	3	-	1	-	-
Avg.	3	2.4	0.4	1.4	-	0.6

MC4101 ADVANCED DATA STRUCTURES AND ALGORITHMS L T P C

3 0 0 3

COURSE OBJECTIVES:

- To understand the usage of algorithms in computing
- To learn and use hierarchical data structures and its operations
- To learn the usage of graphs and its applications
- To select and design data structures and algorithms that is appropriate for problems
- To study about NP Completeness of problems.

UNIT I ROLE OF ALGORITHMS IN COMPUTING & COMPLEXITY ANALYSIS 9

Algorithms – Algorithms as a Technology -Time and Space complexity of algorithms- Asymptotic analysis-Average and worst-case analysis-Asymptotic notation-Importance of efficient algorithms-Program performance measurement - Recurrences: The Substitution Method – The Recursion-

Tree Method- Data structures and algorithms.

UNIT II HIERARCHICAL DATA STRUCTURES 9

Binary Search Trees: Basics – Querying a Binary search tree – Insertion and Deletion- Red Black trees: Properties of Red-Black Trees – Rotations – Insertion – Deletion -B-Trees: Definition of B - trees – Basic operations on B-Trees – Deleting a key from a B-Tree- Heap – Heap Implementation – Disjoint Sets - Fibonacci Heaps: structure – Mergeable-heap operations- Decreasing a key and deleting a node-Bounding the maximum degree.

UNIT III GRAPHS 9

Elementary Graph Algorithms: Representations of Graphs – Breadth-First Search – Depth-First Search – Topological Sort – Strongly Connected Components- Minimum Spanning Trees: Growing a Minimum Spanning Tree – Kruskal and Prim- Single-Source Shortest Paths: The Bellman-Ford algorithm – Single-Source Shortest paths in Directed Acyclic Graphs – Dijkstra‘s Algorithm; Dynamic Programming - All-Pairs Shortest Paths: Shortest Paths and Matrix Multiplication – The Floyd-Warshall Algorithm

UNIT IV ALGORITHM DESIGN TECHNIQUES 9

Dynamic Programming: Matrix-Chain Multiplication – Elements of Dynamic Programming – Longest Common Subsequence- Greedy Algorithms: – Elements of the Greedy Strategy- An Activity-Selection Problem - Huffman Coding.

UNIT V NP COMPLETE AND NP HARD 9

NP-Completeness: Polynomial Time – Polynomial-Time Verification – NP- Completeness and Reducibility – NP-Completeness Proofs – NP-Complete Problems.

TOTAL : 45 PERIODS

SUGGESTED ACTIVITIES:

1. Write an algorithm for Towers of Hanoi problem using recursion and analyze the complexity (No of disc-4)
2. Write any one real time application of hierarchical data structure
3. Write a program to implement Make_Set, Find_Set and Union functions for Disjoint Set Data Structure for a given undirected graph G(V,E) using the linked list representation with simple implementation of Union operation
4. Find the minimum cost to reach last cell of the matrix from its first cell
5. Discuss about any NP completeness problem

COURSE OUTCOMES:

CO1:Design data structures and algorithms to solve computing problems.
CO2: Choose and implement efficient data structures and apply them to solve problems.
CO3: Design algorithms using graph structure and various string-matching algorithms to solve real-life problems.
CO4: Design one’s own algorithm for an unknown problem.
CO5: Apply suitable design strategy for problem solving.

REFERENCES

1. S.Sridhar,” Design and Analysis of Algorithms”, Oxford University Press, 1st Edition, 2014.
2. Adam Drozdex, “Data Structures and Algorithms in C++”, Cengage Learning, 4th Edition, 2013.

Algorithm

Definition

An *algorithm* is a finite set of instructions that accomplishes a particular task.

Characteristics of an algorithm:-

- Must take an input.
- Must give some output(yes/no,valueetc.)
- Definiteness –each instruction is clear and unambiguous.
- Finiteness –algorithm terminates after a finite number of steps.
- Effectiveness –every instruction must be basic i.e. simple instruction.

Expectation from an algorithm

- Correctness:-
  - Correct: Algorithms must produce correct result.
  - Produce an incorrect answer: Even if it fails to give correct results all the time still there is a control on how often it gives wrong result. Eg.Rabin-Miller
  - PrimalityTest (Used in RSA algorithm): It doesn't give correct answer all the time.1 out of 250 times it gives incorrect result.
  - Approximation algorithm: Exact solution is not found, but near optimal solution can be found out. (Applied to optimization problem.)
- Less resource usage: Algorithms should use less resources (time and space)

Good Algorithms?

- Run in less time
- Consume less memory

But computational resources (time complexity) is usually more important

Algorithms as a Technology

- Algorithms are just like a technology. We all use latest and greatest processors but we need to run implementations of good algorithms on that computer in order to properly take benefits of our money that we spent to have the latest processor.
- Let's make this example more concrete by pitting a **faster computer(computer A)** running a sorting algorithm whose running time on n values grows like n^2 against a **slower computer (computer B)** running a sorting algorithm whose running time grows like $n \log n$

Algorithms as a Technology

- They each must sort an array of 10 million numbers. Suppose that computer A executes **10 billion** instructions per second (faster than any single sequential computer at the time of this writing) and computer B executes only **10 million** instructions per second, so that ***computer A is 1000 times faster than computer B in raw computing power.***
- To make the difference even more dramatic, suppose that the world's craftiest programmer codes in machine language for computer A, and the resulting code requires $2n^2$ instructions to sort n numbers. Suppose further that just an average programmer writes for computer B, using a high level language with an inefficient compiler, with the resulting code taking $50\ n \log n$ instructions.

Time and Space complexity of algorithms

Time and space complexity of algorithms

- To analyze an algorithm means:
  - developing a formula for predicting ***how fast* an algorithm** is, based on the size of the input (**time complexity**), and/or
  - developing a formula for predicting ***how much memory*** an algorithm requires, based on the size of the input (**space complexity**)
- Usually time is our biggest concern
  - Most algorithms require a fixed amount of space

Time Complexity of Algorithms

- If running time T(n) is O(f(n)) then the function f measures time complexity
  - **Polynomial** algorithms: T(n) is O(n^k); k = const
  - **Exponential** algorithm: otherwise
- **Intractable problem**: if no polynomial algorithm is known for its solution

Time Complexity

Problem :

1. Find the time complexity of sum of n numbers in an array?

```
for(i=0;i<n;i++)
{
sum=sum+a[i];
}
```

Time Complexity

Statement	Space occupied
i=0	1 (as it executes only one time)
i<n	The statement executes **n times and the statement executes once more when i<n is false. So execution may be n+1 times.**
i++	n times
Sum=sum+a[i]	n times
Total	**3n+2**

Big-Omega

- The function g(n) is Ω(f(n)) iff there exist a real positive constant $c > 0$ and a positive integer n_0 such that g(n) $\geq c$f(n) for all $n \geq n_0$
  - Big Omega is just opposite to Big Oh
  - It generalises the concept of "lower bound" ($\geq$) in the same way as Big Oh generalises the concept of "upper bound" ($\leq$)
  - If f(n) is O(g(n)) then g(n) is Ω(f(n))

Big-Theta

- The function g(n) is Θ(f(n)) iff there exist two real positive constants $c_1 > 0$ and $c_2 > 0$ and a positive integer n_0 such that:

$$c_1 f(n) \geq g(n) \geq c_2 f(n) \text{ for all } n \geq n_0$$

  - Whenever two functions, f and g, are of the same order, g(n) is Θ(f(n)), they are each Big-Oh of the other: g(n) is O(f(n)) AND f(n) is O(g(n))

Upper bounds of complexity

"Big-Oh" specifies an **upper bound of complexity** so that the following (and like) relationships hold:

$$1 = O(\log n) = O(n) = O(n \log n) = \ldots$$

$$\log n = O(n) = O(n \log n) = O(n^{\alpha});\ \alpha > 1 = \ldots$$

$$n = O(n \log n) = O(n^{\alpha});\ \alpha > 1 = O(2^{n}) = \ldots$$

$$n \log n = O(n^{\alpha});\ \alpha > 1 = O(n^{k});\ k > \alpha = \ldots$$

$$n^{k} = O(n^{\alpha});\ \alpha > k = O(2^{n}) = \ldots$$

Time complexity growth

$f(n)$	Number of data items processed per:			
	1 minute	1 day	1 year	1 century
n	**10**	**14,400**	**$5.26 \cdot 10^6$**	**$5.26 \cdot 10^8$**
$n \log n$	10	3,997	883,895	$6.72 \cdot 10^7$
$n^{1.5}$	10	1,275	65,128	$1.40 \cdot 10^6$
n^2	10	379	7,252	72,522
n^3	10	112	807	3,746
2^n	**10**	**20**	**29**	**35**

Beware exponential complexity

☺If a linear, O(n), algorithm processes **10** items per minute, then it can process **14,400** items per day, **5,260,000** items per year, and **526,000,000** items per century.

☻If an exponential, O(2^n), algorithm processes **10** items per minute, then it can process only **20** items per day and **35** items per century…

Big-Oh vs. Actual Running Time

- Example 1: let algorithms **A** and **B** have running times $T_A(n) = 20n$ ms and $T_B(n) = 0.1n \log_2 n$ ms
- In the "Big-Oh"sense, **A** is better than **B**...
- But: on which data volume can **A** outperform **B**?

  $T_A(n) < T_B(n)$ if $20n < 0.1n \log_2 n$, or

  $\log_2 n > 200$, that is, when $n > 2^{200} \approx 10^{60}$!

- Thus, in all practical cases **B** is better than **A**...

“Big-Oh” Feature 1: Scaling

- Constant factors are ignored. Only the powers and functions of *n* should be exploited:

$$\text{for all } c > 0 \rightarrow c \cdot \mathrm{f} = \mathrm{O}(\mathrm{f}) \text{ where } \mathrm{f} \equiv \mathrm{f}(n)$$

- It is this ignoring of constant factors that motivates for such a notation!
- ***Examples***: $50n$, $50000000n$, and $0.0000005n$ are $\mathrm{O}(n)$

“Big-Oh” Feature 2: Transitivity

- If h does not grow faster than g and g does not grow faster than f, then h does not grow faster than f:

$$h = O(g) \textbf{ AND } g = O(f) \rightarrow h = O(f)$$

- In other words, if f grows faster than g and g grows faster than h, then f grows faster than h
- ***Example***: $h = O(g)$; $g = O(n^2) \rightarrow h = O(n^2)$

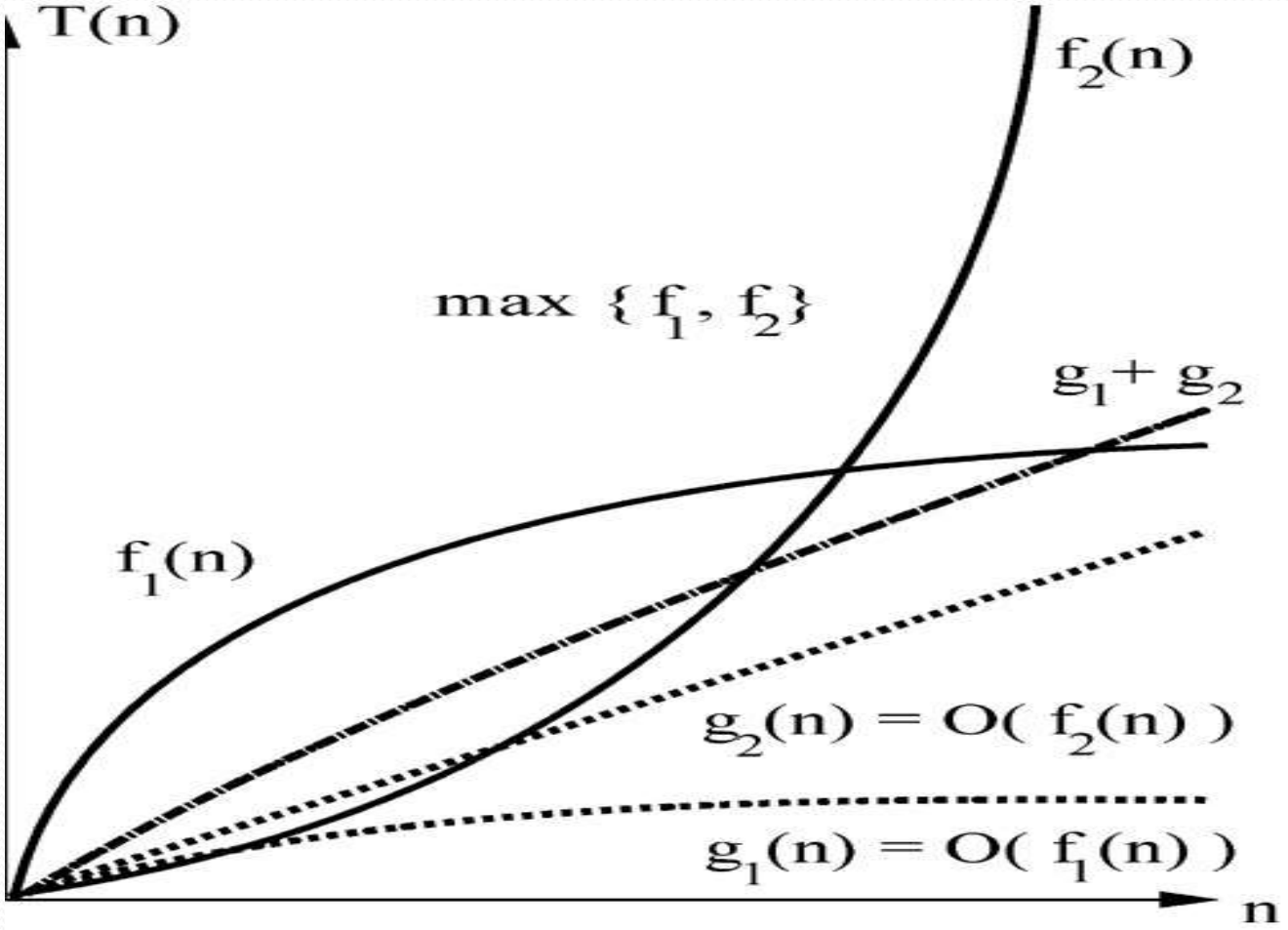
Feature 3: The Rule of Sums

- The sum grows as its fastest term:

$g_1 = O(f_1)$ AND $g_2 = O(f_2) \rightarrow g_1+g_2 = O(\max\{f_1,f_2\})$

  - if g = O(f) and h = O(f), then g + h = O (f)
  - if g = O(f), then g + f = O (f)
- ***Examples***:
  - if h = O(n) AND g = O(n^2), then g + h = O(n^2)
  - if h = O(n log n) AND g = O(n log log n), then g + h = O(n log n)

The Rule of Sums



Feature 4: The Rule of Products

- The upper bound for the product of functions is given by the product of the upper bounds for the functions:

  $g_1 = O(f_1)$ AND $g_2 = O(f_2) \rightarrow g_1 \cdot g_2 = O(f_1 \cdot f_2)$

  - if $g = O(f)$ and $h = O(f)$, then $g \cdot h = O(f^2)$
  - if $g = O(f)$, then $g \cdot h = O(f \cdot h)$
- ***Example:***

  if $h = O(n)$ AND $g = O(n^2)$, then $g \cdot h = O(n^3)$

Ascending order of complexity

$1 \leftarrow \log\log n \leftarrow \log n \leftarrow n \leftarrow n \log n$

$\leftarrow n^{\alpha};\ 1<\alpha<2 \leftarrow n^2 \leftarrow n^3 \leftarrow n^m;\ m>3 \leftarrow 2^n$ **…**

Questions:

- Where is the place of $n^2 \log n$?
- Where is the place of $n^{2.79}$?

- ***Answers:*** …$\leftarrow n^2 \leftarrow n^2 \log n \leftarrow n^{2.79} \leftarrow n^3 \leftarrow \ldots$

Answers to the questions

Running time T(n)	**Complexity O(n)**
$n^2 + 100\,n + 1$	$O(n^2)$
$0.001n^3 + n^2 + 1$	$O(n^3)$
$23\,n$	$O(n)$
2^{3n}	$O(8^n)$ as $2^{3n} \equiv (2^3)^n$
2^{3+n}	$O(2^n)$ as $2^{3+n} \equiv 2^3 \cdot 2^n$
$2 \cdot 3^n$	$O(3^n)$

Answers to the questions

Running time T(n)	**Complexity O(n)**
$0.0001\,n + 10000$	$O(n)$
$100000\,n + 10000$	$O(n)$
$0.0001\,n^2 + 10000\,n$	$O(n^2)$
$100000\,n^2 + 10000\,n$	$O(n^2)$
$30 \log_{20}(23n)$ **actually NOT that hard…**	$O(\log n)$ as $\log_c(ab)=\log_c a + \log_c b$

Space Complexity

The space complexity can be defined as amount of memory required by an algorithm to run

- To compute space complexity we use two factors constant and instance characteristics. The space requirement s(p) can be given as
- S(p) = c + sp
- Here c → constant
- Sp →space dependent upon instance characteristics

Space Complexity

Problem :

1. **Find the space complexity of measure of algorithm which returns (a+b+b*c+4.0)**

Solution:

```
sample alg(a,b,c)
{
return(a+b+c);
}
```

- Space complexity is calculated as S(p) = c + sp
- Assume each variable occupies one word size then the space is

Space Complexity

Statement	Space occupied
Space allocation is done for variable a	1
Space allocation is done for variable b	1
Space allocation is done for variable c	1
Total	**3**

- S(p)=3 + 0 =3 (Assuming a,b,c occupies 1 word size each)
- Because there are no instance characteristics and we know that we get a single output for the algorithm the space occupied by the algorithm is **3.**

Space Complexity

Problem :

1. **Find the space complexity for adding sum of n numbers in the array using recursive concept?**

```
recursivesum(a,n)
{
if(n<0)
return(0.0)
else return(rsum(a(n-1))+a[n]);
}
```

Space Complexity

Statement	Space occupied
Space allocation is done for variable a	1
Space allocation is done for a	1
Space allocation is done for output (returning value)	1
Total	**3**

- To know the depth of the recursion is **(n+1) so we need to multiply this depth of recursion with the total space. So the space occupied is given as follows**
- Total Space occupied is →3*(n+1)
  → 3(n+1)

Asymptotic analysis

Asymptotic analysis

Asymptotic analysis of an algorithm refers to defining the mathematical boundation /framing of its run-time performance.

Using asymptotic analysis, we can very well conclude the best case, average case, and worst case scenario of an algorithm.

Asymptotic analysis is input bound i.e., if there's no input to the algorithm, it is concluded to work in a constant time. Other than the "input" all other factors are considered constant.

Asymptotic analysis

Asymptotic analysis refers to computing the running time of any operation in mathematical units of computation. For example, *the running time of one operation is computed as f(n) and may be for another operation it is computed as* $g(n^2)$.

This means the first operation running time will increase linearly with the increase in **n** and the running time of the second operation will increase exponentially when **n** increases. Similarly, the running time of both operations will be nearly the same if **n** is significantly small.

Usually, the time required by an algorithm falls under three types:

- **Best Case** − ***Minimum time*** required for program execution.
- **Average Case** − ***Average time*** required for program execution.
- **Worst Case** − ***Maximum time*** required for program execution.

Average Case Analysis (Sometimes done)

In average case analysis, we take all possible inputs and calculate computing time for all of the inputs. Sum all the calculated values and divide the sum by the total number of inputs. We must know (or predict) the distribution of cases.

$$\sum_{i=1}^{n} \frac{\Theta(i)}{(n+1)}$$

$$\frac{\Theta((n+1)*(n+2)/2)}{(n+1)}$$

$$= \Theta(n)$$

Average Case Analysis (Sometimes done)

In average case analysis, we take all possible inputs and calculate computing time for all of the inputs. Sum all the calculated values and divide the sum by the total number of inputs. We must know (or predict) the distribution of cases.

For the linear search problem, let us assume that all cases are ***uniformly districuted*** (including the case of x not being present in the array). So we sum all the cases and divide the sum by (n+1). Following is the value of average-case time complexity.

Worst Case Analysis (Usually Done)

In the worst-case analysis, we calculate the upper bound on the running time of an algorithm. We must know the case that causes a maximum number of operations to be executed.

For Linear Search, the worst case happens when the element to be searched (x in the above code) is not present in the array. When x is not present, the search() function compares it with all the elements of arr[] one by one. Therefore, the worst-case time complexity of linear search would be $\Theta(n)$.

Asymptotic Notations

Asymptotic Complexity

- Running time of an algorithm as a function of input size n **for large n**.
- Expressed using only the **highest-order term** in the expression for the exact running time.
  - Instead of exact running time, say $\Theta(n^2)$.
- Describes behavior of function in the limit.
- Written using ***Asymptotic Notation***.

Asymptotic Notation

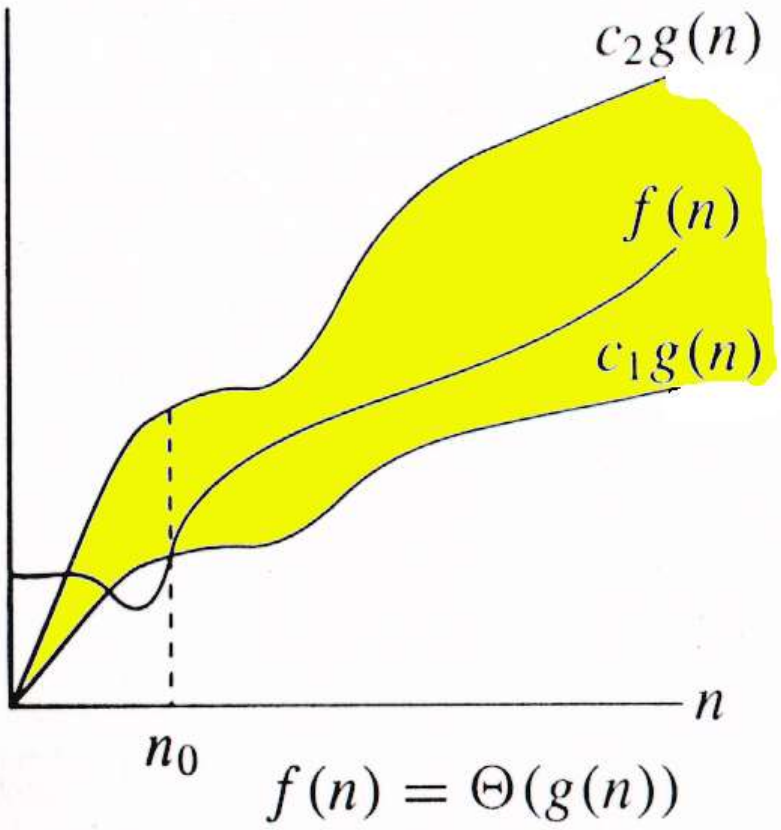
- **Θ, O, Ω, o, ω**
- Defined for functions over the natural numbers.
  - **Ex:** $f(n) = \Theta(n^2)$.
  - Describes how $f(n)$ grows in comparison to n^2.
- Define a ***set*** of functions; in practice used to compare two function sizes.
- The notations describe different rate-of-growth relations between the defining function and the defined set of functions.

Θ-notation

For function $g(n)$, we define $\Theta(g(n))$, big-Theta of n, as the set:

$\Theta(g(n)) = \{f(n) :$
∃ positive constants c_1, c_2, and n_0, such that $\forall n \geq n_0$,
we have $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$
$\}$

Intuitively: Set of all functions that have the same *rate of growth* as $g(n)$.



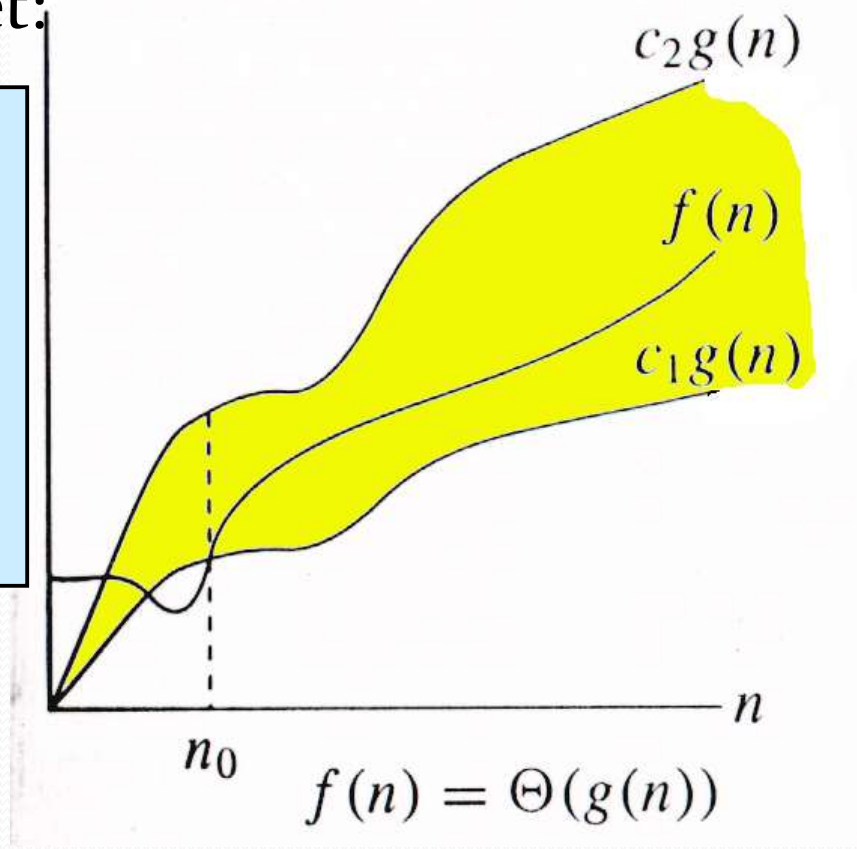
$g(n)$ is an *asymptotically tight bound* for $f(n)$.

Θ-notation

For function $g(n)$, we define $\Theta(g(n))$, big-Theta of n, as the set:

$\Theta(g(n)) = \{f(n) :$
$\exists$ **positive constants c_1, c_2, and n_0, such that** $\forall n \geq n_0$,
we have $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$
$\}$

Technically, $f(n) \in \Theta(g(n))$.
Older usage, $f(n) = \Theta(g(n))$.
I'll accept either…



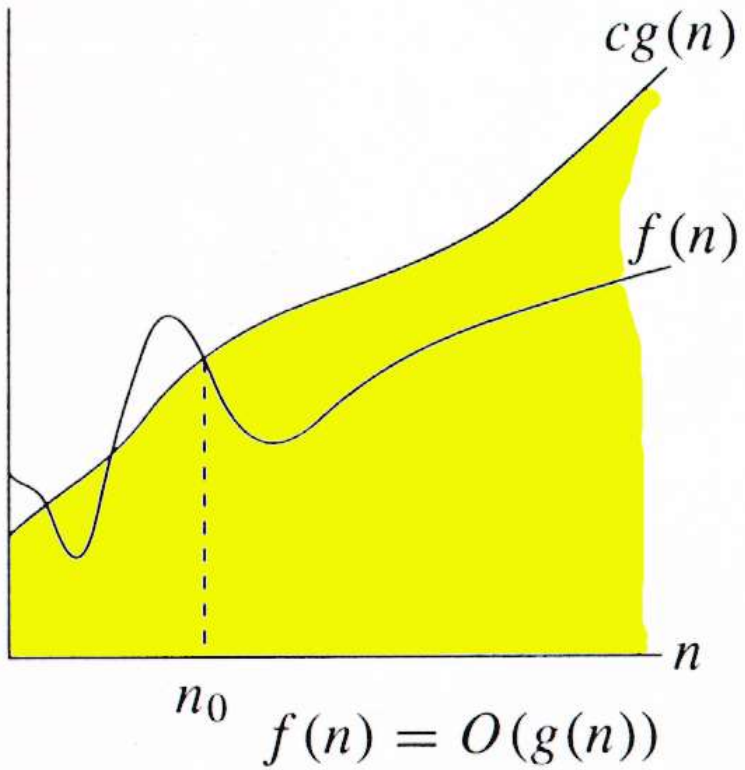
$f(n)$ and $g(n)$ are nonnegative, for large n.

O-notation

For function $g(n)$, we define $O(g(n))$, big-O of n, as the set:

$O(g(n)) = \{f(n) :$
$\exists$ **positive constants** c **and** n_0,
such that $\forall n \geq n_0$,
we have $0 \leq f(n) \leq cg(n)\}$

Intuitively: Set of all functions whose *rate of growth* is the same as or lower than that of $g(n)$.



$g(n)$ is an *asymptotic upper bound* for $f(n)$.

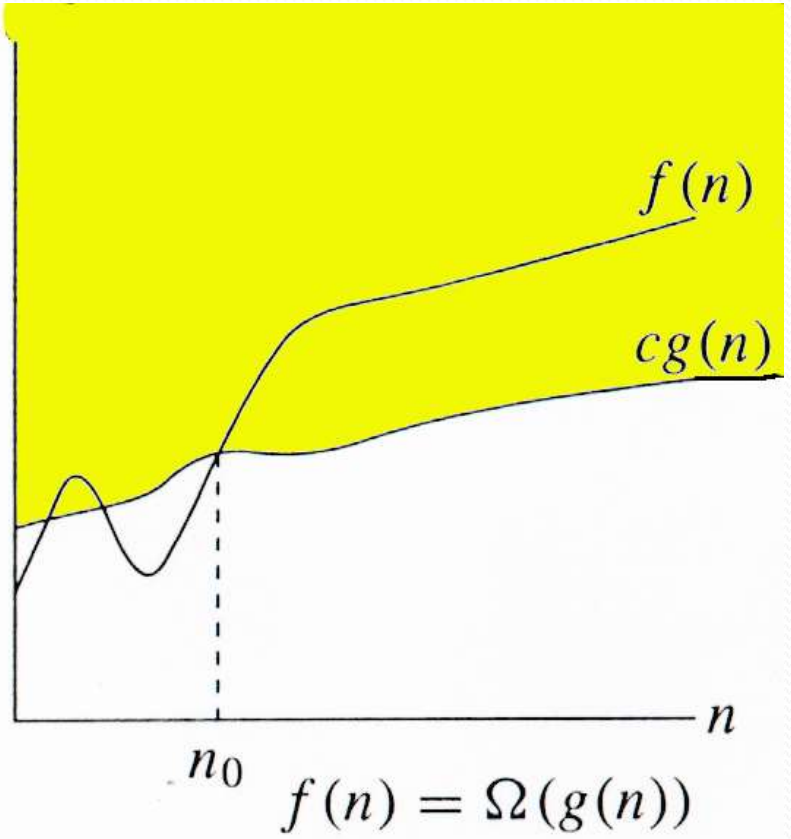
$f(n) = \Theta(g(n)) \Rightarrow f(n) = O(g(n))$.
$\Theta(g(n)) \subset O(g(n))$.

Ω -notation

For function $g(n)$, we define $\Omega(g(n))$, big-Omega of n, as the set:

$\Omega(g(n)) = \{f(n)$:
∃ **positive constants** c **and** n_0,
such that $\forall n \geq n_0$,
we have $0 \leq cg(n) \leq f(n)\}$

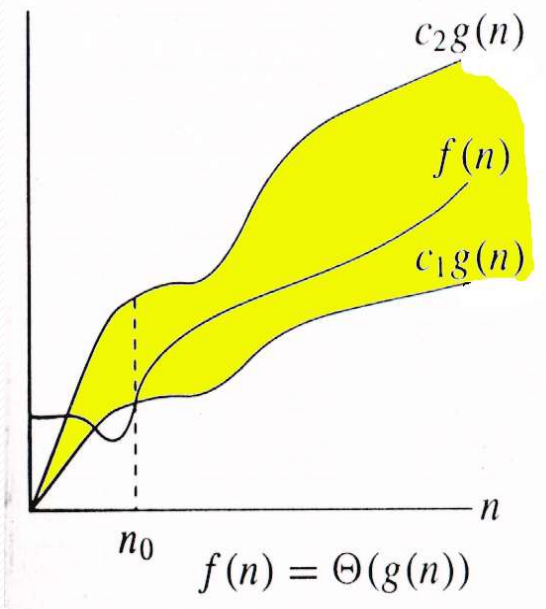
Intuitively: Set of all functions whose *rate of growth* is the same as or higher than that of $g(n)$.

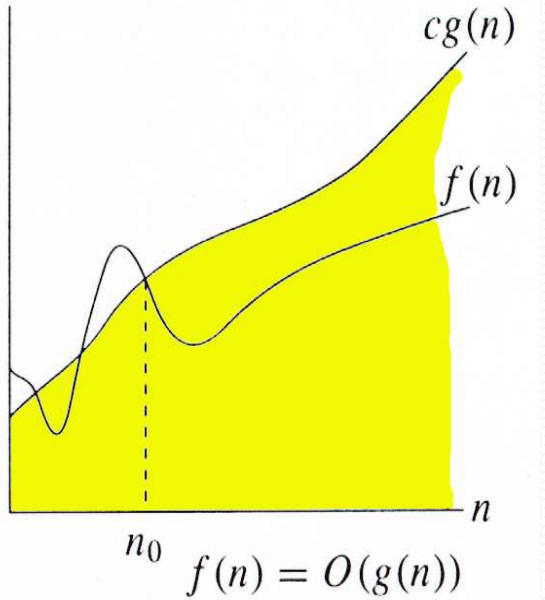


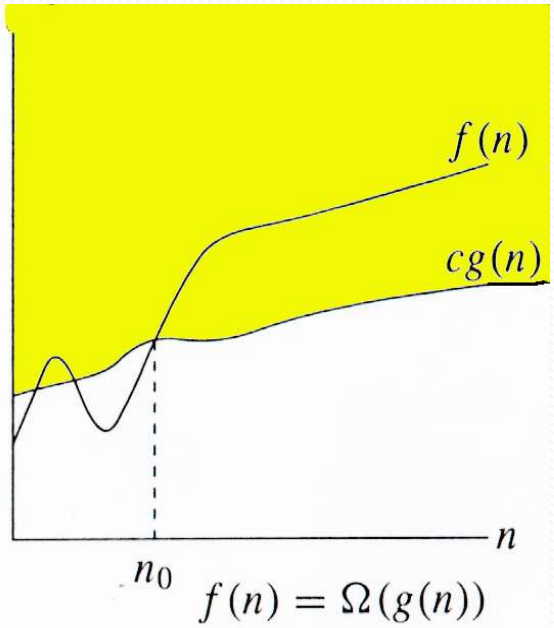
$g(n)$ is an *asymptotic lower bound* for $f(n)$.

$f(n) = \Theta(g(n)) \Rightarrow f(n) = \Omega(g(n))$.
$\Theta(g(n)) \subset \Omega(g(n))$.

Relations Between Θ, O, Ω







Relations Between Θ, Ω, O

Theorem : For any two functions $g(n)$ and $f(n)$,

$f(n) = \Theta(g(n))$ iff

$f(n) = O(g(n))$ **and** $f(n) = \Omega(g(n))$.

- I.e., $\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$
- In practice, asymptotically tight bounds are obtained from asymptotic upper and lower bounds.

Running Times

- "Running time is $O(f(n))$" $\Rightarrow$ Worst case is $O(f(n))$
- $O(f(n))$ bound on the worst-case running time $\Rightarrow$ $O(f(n))$ bound on the running time of every input.
- $\Theta(f(n))$ bound on the worst-case running time $\not\Rightarrow$ $\Theta(f(n))$ bound on the running time of every input.
- "Running time is $\Omega(f(n))$" $\Rightarrow$ Best case is $\Omega(f(n))$
- Can still say "Worst-case running time is $\Omega(f(n))$"
  - Means worst-case running time is given by some unspecified function $g(n) \in \Omega(f(n))$.

Example

- ***Insertion sort*** takes $\Theta(n^2)$ in the worst case, so sorting (as a *problem*) is $O(n^2)$. **Why?**
- Any sort algorithm must look at each item, so sorting is $\Omega(n)$.
- In fact, using (e.g.) merge sort, sorting is $\Theta(n \lg n)$ in the worst case.
  - Later, we will prove that we cannot hope that any comparison sort to do better in the worst case.

Asymptotic Notation in Equations

- Can use asymptotic notation in equations to replace expressions containing lower-order terms.
- For example,

  $4n^3 + 3n^2 + 2n + 1 = 4n^3 + 3n^2 + \Theta(n)$

  $= 4n^3 + \Theta(n^2) = \Theta(n^3)$. **How to interpret?**
- In equations, $\Theta(f(n))$ always stands for an ***anonymous function*** $g(n) \in \Theta(f(n))$
  - In the example above, $\Theta(n^2)$ stands for $3n^2 + 2n + 1$.

o-notation

For a given function $g(n)$, the set little-o:

$$o(g(n)) = \{f(n): \forall\, c > 0, \exists\, n_0 > 0 \text{ such that } \forall\, n \geq n_0, \text{ we have } 0 \leq f(n) < cg(n)\}.$$

$f(n)$ becomes insignificant relative to $g(n)$ as n approaches infinity:

$$\lim_{n \to \infty} [f(n) / g(n)] = 0$$

$g(n)$ is an ***upper bound*** for $f(n)$ that is not asymptotically tight.

Observe the difference in this definition from previous ones. **Why?**

For a given function $g(n)$, the set little-omega:

$$\omega(g(n)) = \{f(n): \forall\, c > 0, \exists\, n_0 > 0 \text{ such that } \forall\, n \geq n_0, \text{ we have } 0 \leq cg(n) < f(n)\}.$$

$f(n)$ becomes arbitrarily large relative to $g(n)$ as n approaches infinity:

$$\lim_{n \to \infty} [f(n) / g(n)] = \infty.$$

$g(n)$ is a ***lower bound*** for $f(n)$ that is not asymptotically tight.

Comparison of Functions

$$f \leftrightarrow g \approx a \leftrightarrow b$$

$$f(n) = O(g(n)) \approx a \leq b$$
$$f(n) = \Omega(g(n)) \approx a \geq b$$
$$f(n) = \Theta(g(n)) \approx a = b$$
$$f(n) = o(g(n)) \approx a < b$$
$$f(n) = \omega(g(n)) \approx a > b$$

Importance of efficient algorithms-

An algorithm is a method or a process followed to solve a problem. If the problem is viewed as a function, then an algorithm is an implementation for the function that transforms an input to the corresponding output.

A problem can be solved by many different algorithms. A given algorithm solves only one problem (i.e., computes a particular function).

Importance of efficient algorithms-

The advantage of knowing several solutions to a problem is that solution A might be more efficient than solution B for a specific variation of the problem, or for a specific class of inputs to the problem, while solution B might be more efficient than A for another variation or class of inputs.

- For example, one sorting algorithm might be the best for sorting a small collection of integers (which is important if you need to do this many times).
- Another might be the best for sorting a large collection of integers.
- A third might be the best for sorting a collection of variable-length strings

Importance of efficient algorithms-

By definition, something can only be called an algorithm if it has all of the following properties.

1. It must be **correct:** In other words, it must compute the desired function, converting each input to the correct output. Note that every algorithm implements some function, because every algorithm maps every input to some output (even if that output is a program crash).
2. It is **composed of a series of concrete steps:** Concrete means that the action described by that step is completely understood — and doable — by the person or machine that must perform the algorithm. Each step must also be doable in a finite amount of time

3. There can be no ambiguity as to which step will be performed next. Often it is the next step of the algorithm description.
4. It must be composed of a finite number of steps. Most languages for describing algorithms (including English and “pseudocode”). Data Structures and Algorithms way to perform repeated actions, known as iteration.
5. It must terminate. In other words, it may not go into an infinite loop.

Importance of efficient algorithms-

As applications are getting complex and data rich, there are three common problems that applications face now-a-days.

- **Data Search** – Consider an inventory of 1 million(10^6) items of a store. If the application is to search an item, it has to search an item in 1 million(10^6) items every time slowing down the search. As data grows, search will become slower.
- **Processor speed** – Processor speed although being very high, falls limited if the data grows to billion records.
- **Multiple requests** – As thousands of users can search data simultaneously on a web server, even the fast server fails while searching the data.
- To solve the above-mentioned problems, data structures come to rescue. Data can be organized in a data structure in such a way that all items may not be required to be searched, and the required data can be searched almost instantly.

Recurrences

Recursion is a particularly powerful kind of reduction, which can be described loosely as follows:

- If the given instance of the problem is small or simple enough, just solve it.
- Otherwise, reduce the problem to one or more simpler instances of the same problem.

- Recursion is generally expressed in terms of recurrences.
- In other words, when an algorithm calls to itself, we can often describe its running time by a **recurrence equation which** describes the overall running time of a problem of size n in terms of the running time on smaller inputs.

Recurrences

E.g.the worst case running time T(n) of the merge sort procedure by recurrence can be expressed as

$T(n) = \theta(1)$; if $n=1$

$2T(n/2) + \theta(n)$;if $n>1$

whose solution can be found as $T(n)=\theta(n \log n)$

There are various techniques to solve recurrences..

Recurrences – Substitution method

The substitution method comprises of 3 steps

i. Guess the form of the solution

ii. Verify by induction

iii. Solve for constants

- We substitute the guessed solution for the function when applying the inductive hypothesis to smaller values. Hence the name “substitution method”.
- This method is powerful, but we must be able to guess the form of the answer in order to apply it.

Recurrences – Substitution method

e.g.recurrence equation: $T(n)=4T(n/2)+n$

step 1: guess the form of solution

$T(n)=4T(n/2)$

$F(n)=4f(n/2)$

$F(2n)=4f(n)$

$F(n)=n^2$

So, $T(n)$ is order of n^2

Guess $T(n)=O(n^3)$

Step 2: verify the induction

Assume $T(k)<=ck^3$

$T(n)=4T(n/2)+n$

$<=4c(n/2)^3+n$

$<=cn^3/2+n$

$<=cn^3-(cn^3/2-n)$

$T(n)<=cn^3$ as $(cn^3/2-n)$ is always positive So what we assumed was true.

$T(n)=O(n^3)$

Recurrences – Substitution method

Step 3: solve for constants

$Cn^3/2-n>=0$

$n>=1$

$c>=2$

Now suppose we guess that $T(n)=O(n^2)$ which is tight upper bound

Assume,$T(k)<=ck^2$ so,we should prove that $T(n)<=cn^2$

$T(n)=4T(n/2)+n$

$4c(n/2)^2+n$

cn^2+n

- So,T(n) will never be less than cn^2. But if we will take the assumption of $T(k)=c1\ k^2-c2k$, then wecan find that $T(n) = O(n^2)$

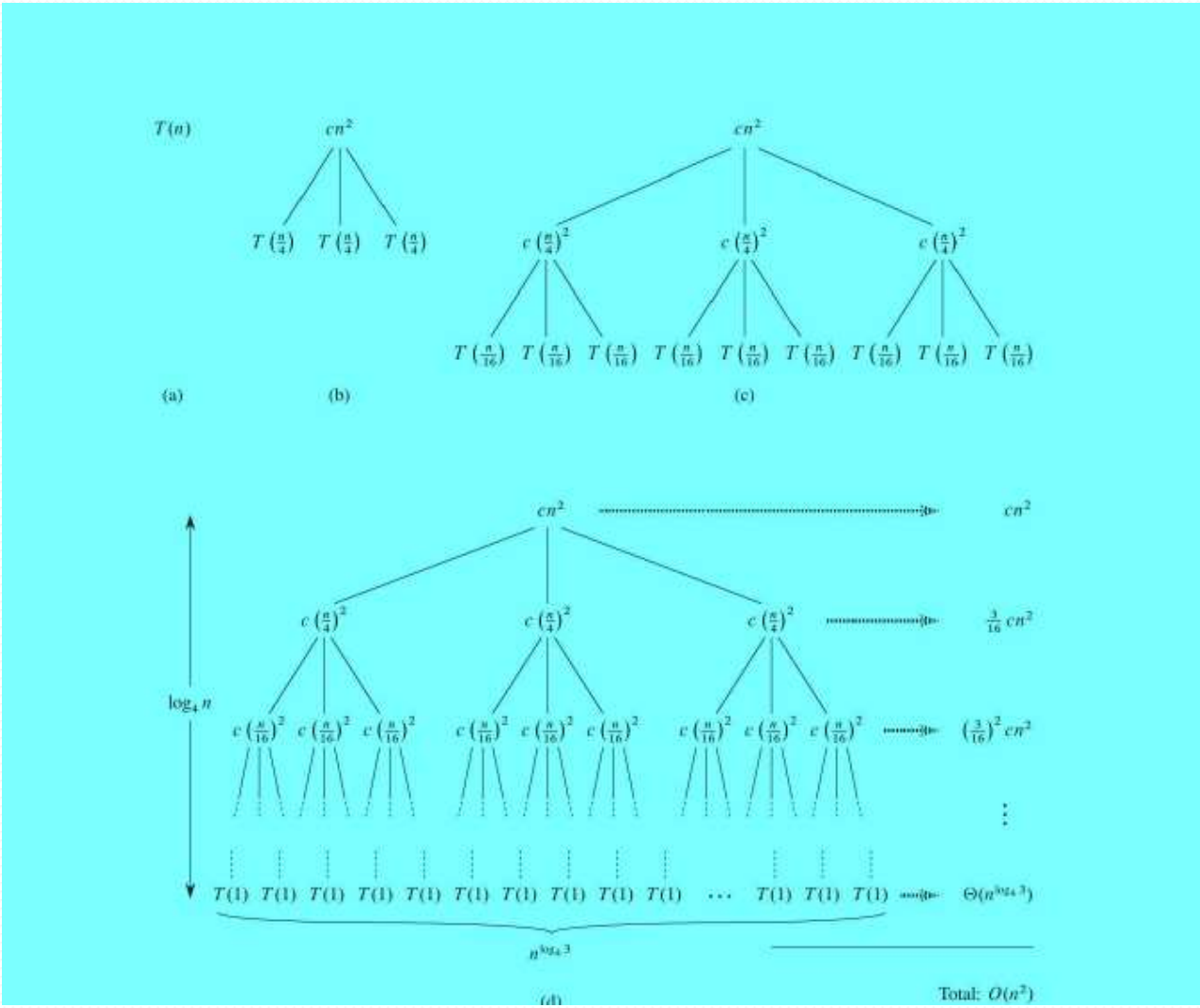
Recurrences – Reccursion Tree method

RECURSSION TREE METHOD:

- In a recursion tree ,each node represents the cost of a single sub-problem somewhere in the set of recursive problems invocations .we sum the cost within each level of the tree to obtain a set of per level cost,and then we sum all the per level cost to determine the total cost of all levels of recursion .
- Constructing a recursion tree for the recurrence

$$T(n)=3T(n/4)+cn^2$$

Recurrences – Reccursion Tree method



Recurrences – Reccursion Tree method

- Constructing a recursion tree for the recurrence T (n)= 3T (n=4) + cn².. Part (a) shows T (n),
- which progressively expands in (b)–(d) to form the recursion tree. The fully expanded tree in part (d) has height $\log_4 n$ (it has log4n + 1 levels).
  - Sub problem size at depth i =$n/4^i$
  - Sub problem size is 1 when $n/4^i=1$ => $i=\log_4 n$
  - So, no. of levels =1+ $\log_4 n$
  - Cost of each level = (no. of nodes) x (cost of each node)

Recurrences – Reccursion Tree method

No. Of nodes at depth $i=3^i$

Cost of each node at depth $i=c\ (n/4^i)^2$

Cost of each level at depth $i=3^i c\ (n/4^i)^2 = (3/16)^i cn^2$

$T(n)= \sum_{i=0}^{\log_4 n} cn^2(3/16)^i$

$T(n)= \sum_{i=0}^{\log_4 n -1} cn^2(3/16)^i$ + cost of last level

Cost of nodes in last level $=3^i T(1)$

⇨ $c3^{\log_4 n}$ (at last level $i=\log_4 n$)

⇨ $cn^{\log_4 3}$

$$T(n)= \sum_{i=0}^{\log_4 n -1} cn^2(3/16)^i + c\, n^{\log_4 3}$$

$$<= cn^2 \sum_{i=0}^{\infty} (3/16)^i + cn^{\log_4 3}$$

⇨ $<= cn^2*(16/13)+ cn^{\log_4 3}$ => $T(n)=O(n^2)$

- **Definition**
  - Data structure is a specified format for organizing and storing the data

- **Types of data structures**
  - Primitive data structures
  - Composite data structures with 2 types
- **Composite data structure**
  - Linear data structures
  - Non-Linear data structures

- **Primitive data structures**
  - Integer – Example: 1, 2, 3, -1, -5, 77
  - Float – Example: 1.3, 2.34, 3.5, -1.3, -5.6, 77.765
  - Character – Example: 'A', 'a', '2'
  - String – Example: "abc", "123", "ASD"
  - Boolean – Example: TRUE, FALSE

- **Composite - Linear data structures**
  - Arrays
  - Stack
  - Queue
  - Linked List

- **Composite - Non-Linear data structures**
  - Trees
  - Graphs

- **Algorithm**
  - Step by Step Process which defines set of instruction to be executed in certain order to get the desired output

- **Characteristics of Algorithm**
  - Unambiguous
  - Input
  - Output
  - Finiteness
  - Flexibility
  - Independent

- **Operations on data structures**
  - There are six operations can be obtained with Non-Primitive data structures. They are as follows
    - Insertion
    - Deletion
    - Searching
    - Traversing
    - Sorting
    - Merging

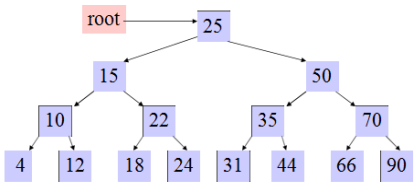
Binary Search Trees: Basics – Querying a Binary search tree – Insertion and Deletion- Red Black trees: Properties of Red-Black Trees – Rotations – Insertion – Deletion -B-Trees: Definition of B trees – Basic operations on B-Trees – Deleting a key from a B-Tree- Heap – Heap Implementation – Disjoint Sets - Fibonacci Heaps: structure – Mergeable-heap operations- Decreasing a key and deleting a node-Bounding the maximum degree.

BINARY SEARCH TREES

BASICS:

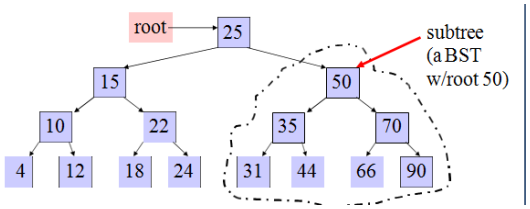
1. Hierarchical data structure with a single reference to root node
2. Each node has at most two child nodes (a left and a right child)
3. Nodes are organized by the Binary Search property:
   - Every node is ordered by some key data field(s)
   - For every node in the tree, its key is greater than its left child's key and less than its right child's key

Dia: Binary Search Trees



TERMINOLOGY:

1. The **Root** node is the top node in the hierarchy
2. A **Child** node has exactly one **Parent** node, a Parent node has at most two child nodes, **Sibling** nodes share the same Parent node (ex. node 22 is a child of node 15)
3. A **Leaf** node has no child nodes, an **Interior** node has at least one child node (ex. 18 is a leaf node)
4. Every node in the BST is a **Subtree** of the BST rooted at that node.



IMPLEMENTING BINARY SEARCH TREE:

Self-referential class is used to build Binary Search Trees

```
public class BSTNode
{
 Comparable data;
```

```
    BSTNode left;
    BSTNode right;
    public BSTNode(Comparable d)
    {
    data = d; left = right = null;
} }
```

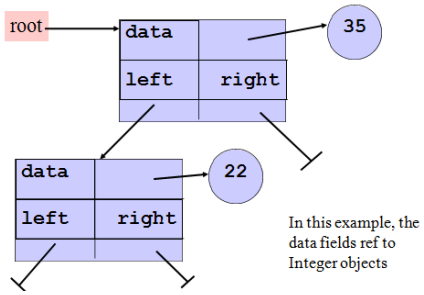
- left refers to the left child
- right refers to the left child
- data field refers to object that implements the Comparable interface, so that data fields can be compared to order nodes in the BST
- Single reference to the root of the BST
  - ➢ All BST nodes can be accessed through root reference by recursively accessing left or right child nodes

OPERATIONS:

- Naturally recursive:
  - Each node in the BST is itself a BST
- Some Operations:
  - Create a BST
  - Find node in BST using its key field
  - Add a node to the BST
  - Traverse the BST
    - ❖ visit all the tree nodes in some order

(1) Create a BST:

```
bst_node root = null;      // an empty BST
root = new BSTNode(new Integer(35));      // a BST w/1 node
Root.setLeft(new BSTNode(new Integer(22)); // add left child
```



(2) Find node in BST using its key field

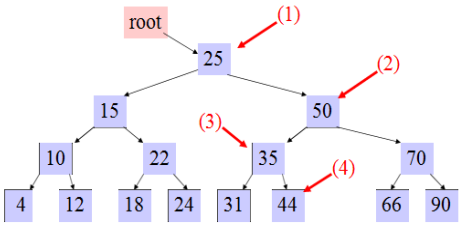
- Use the search key to direct a recursive binary search for a matching node
  1. Start at the root node as current node
  2. If the search key's value matches the current node's key then found a match
  3. If search key's value is greater than current node's
     1. If the current node has a right child, search right
     2. Else, no matching node in the tree

4. If search key is less than the current node's
    1. If the current node has a left child, search left
    2. Else, no matching node in the tree

Example: search for 45 in the tree

(key fields are show in node rather than in separate obj ref to by data field):

1. start at the root, 45 is greater than 25, search in right subtree
2. 45 is less than 50, search in 50's left subtree
3. 45 is greater than 35, search in 35's right subtree
4. 45 is greater than 44, but 44 has no right subtree so 45 is not in the BST



---

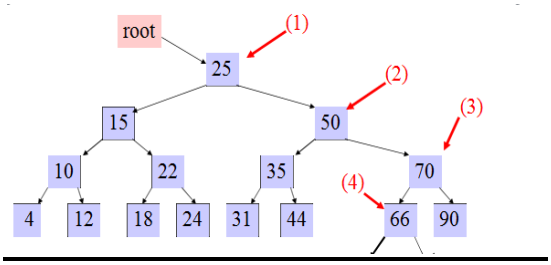
(3) Add a node to the BST

1. Always insert new node as leaf node
2. Start at root node as current node
3. If new node's key < current's key
    i. If current node has a left child, search left
    ii. Else add new node as current's left child
4. If new node's key > current's key
    i. If current node has a right child, search right
    ii. Else add new node as current's right child

Eg: Insert 60 in the tree

1. start at the root, 60 is greater than 25, search in right subtree
2. 60 is greater than 50, search in 50's right subtree
3. 60 is less than 70, search in 70's left subtree

    60 is less than 66, add 60 as 66's left child



(4) Traversal

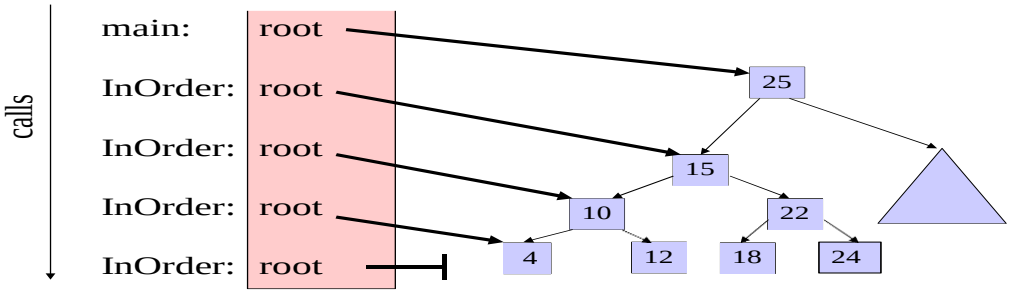
- Visit every node in the tree and perform some operation on it, (eg) print out the data fields of each node
- Three steps to a traversal
  - ❖ Visit the current node
  - ❖ Traverse its left subtree
  - ❖ Traverse its right subtree
- The order in which you perform these three steps results in different traversal orders:
  - ❖ Pre-order traversal:(1) (2) (3)
  - ❖ In-order traversal: (2) (1) (3)
  - ❖ Post-order traversal: (2) (3) (1)

Eg:

```
public void InOrder(bst_node root)
{
        // stop the recursion:
        if(root == null)
        {
          return;
        }
        // recursively traverse left subtree:  InOrder(root.leftChild());
        // visit the current node:  Visit(root);
        // recursively traverse right subtree:  InOrder(root.rightChild());
}
```

```
// in main: a call to InOrder passing root
InOrder(root);

// The call stack after the first few
// calls to InOrder(root.leftChild()):
```

Call Stack (drawn upside down):



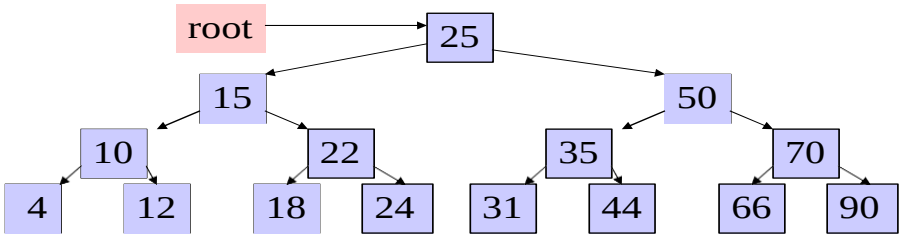
CS21, Tia Newhall

Traversal Examples

InOrder(root) visits nodes in the following order:
4, 10, 12, 15, 18, 22, 24, 25, 31, 35, 44, 50, 66, 70, 90

A Pre-order traversal visits nodes in the following order:
25, 15, 10, 4, 12, 22, 18, 24, 50, 35, 31, 44, 70, 66, 90

A Post-order traversal visits nodes in the following order:
4, 12, 10, 18, 24, 22, 15, 31, 44, 35, 66, 90, 70, 50, 25



CS21, Tia Newhall

QUERYING A BINARY SEARCH TREE

A common operation performed on a binary search tree is searching for a key stored in the tree. Besides the SEARCH operation, binary search trees can support such queries as MINIMUM, MAXIMUM, SUCCESSOR, and PREDECESSOR. In this section, we shall

examine these operations and show that each can be supported in time $O(h)$ on a binary search tree of height h.

Searching

We use the following procedure to search for a node with a given key in a binary search tree. Given a pointer to the root of the tree and a key k, TREE-SEARCH returns a pointer to a node with key k if one exists; otherwise, it returns NIL.

TREE-SEARCH (x, k)

1 **if** x= NIL or k = $key[x]$

2 **then return** x

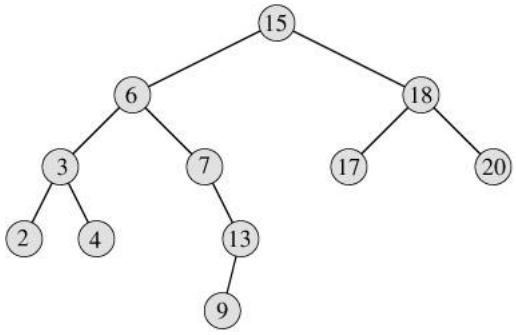
3 **if** $k < key[x]$

4 **then return** TREE-SEARCH($left[x]$, k)

5 **else return** TREE-SEARCH($right[x]$, k)

The procedure begins its search at the root and traces a path downward in the tree, as shown in diagram.

Dia: Binary Search Tree



For each node x it encounters, it compares the key k with $key[x]$. If the two keys are equal, the search terminates. If k is smaller than $key[x]$, the search continues in the left subtree of x, since the binary-search-tree property implies that k could not be stored in the right subtree. Symmetrically, if k is larger than $key[x]$, the search continues in the right subtree. The nodes encountered during the recursion form a path downward from the root of the tree, and thus the running time of TREE-SEARCH is $O(h)$, where h is the height of the tree.

Queries on a binary search tree. To search for the key 13 in the tree, we follow the path $15 \rightarrow 6 \rightarrow 7 \rightarrow 13$ from the root. The minimum key in the tree is 2, which can be found by

following *left* pointers from the root. The maximum key 20 is found by following *right* pointers from the root. The successor of the node with key 15 is the node with key 17, since it is the minimum key in the right subtree of 15. The node with key 13 has no right subtree, and thus its successor is its lowest ancestor whose left child is also an ancestor. In this case, the node with key 15 is its successor.

The same procedure can be written iteratively by "unrolling" the recursion into a **while** loop. On most computers, this version is more efficient.

ITERATIVE-TREE-SEARCH(x, k)

1 **while** $x \neq$ NIL and $k \neq key[x]$

2 **do if** $k < key[x]$

3 **then** $x \leftarrow left[x]$

4 **else** $x \leftarrow right[x]$

5 **return** x

Minimum and maximum

An element in a binary search tree whose key is a minimum can always be found by following *left* child pointers from the root until a NIL is encountered, as shown in diagram. The following procedure returns a pointer to the minimum element in the subtree rooted at a given node x.

TREE-MINIMUM (x)

1 **while** $left[x] \neq$ NIL

2 **do** $x \leftarrow left[x]$

3 **return** x

The binary-search-tree property guarantees that TREE-MINIMUM is correct. If a node x has no left subtree, then since every key in the right subtree of x is at least as large as $key[x]$, the minimum key in the subtree rooted at x is $key[x]$. If node x has a left subtree, then since no key in the right subtree is smaller than $key[x]$ and every key in the left subtree is not larger than $key[x]$, the minimum key in the subtree rooted at x can be found in the subtree rooted at $left[x]$.

The pseudocode for TREE-MAXIMUM is symmetric.

TREE-MAXIMUM(x)

1 **while** $right[x] \neq$ NIL

2 **do** $x \leftarrow right[x]$

3 **return** x

Both of these procedures run in $O(h)$ time on a tree of height h since, as in TREE-SEARCH, the sequence of nodes encountered forms a path downward from the root.

Successor and predecessor

Given a node in a binary search tree, it is sometimes important to be able to find its successor in the sorted order determined by an inorder tree walk. If all keys are distinct, the successor of a node x is the node with the smallest key greater than $key[x]$. The structure of a binary search tree allows us to determine the successor of a node without ever comparing keys. The following procedure returns the successor of a node x in a binary search tree if it exists, and NIL if x has the largest key in the tree.

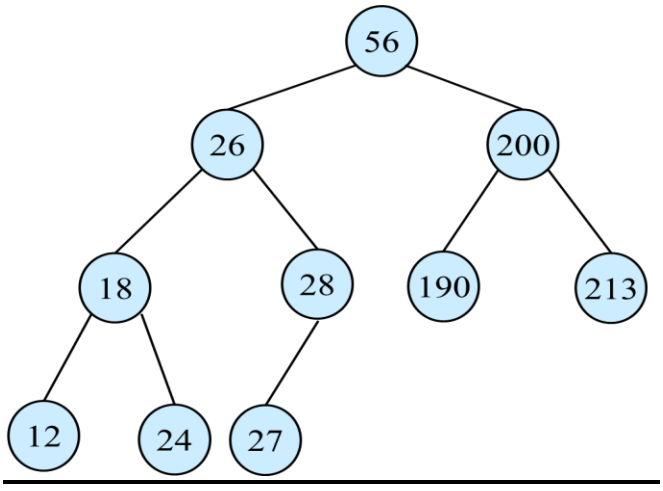
```
TREE-SUCCESSOR(x)
1  if right[x] ≠ NIL
2     then return TREE-MINIMUM (right[x])
3  y ← p[x]
4  while y ≠ NIL and x = right[y]
5     do x ← y
6        y ← p[y]
7  return y
```

The code for TREE-SUCCESSOR is broken into two cases. If the right subtree of node x is nonempty, then the successor of x is just the leftmost node in the right subtree, which is found in line 2 by calling TREE-MINIMUM($right[x]$). For example, the successor of the node with key 15 in diagram is the node with key 17.

INSERTION AND DELETION:

INSERTION

- Change the dynamic set represented by a BST.
- Ensure the binary-search-tree property holds after change.
- Insertion is easier than deletion.



Tree-Insert(T, z)

```
y ← NIL
x ← root[T]
while x ≠ NIL
   do y ← x
      if key[z] < key[x]
         then x ← left[x]
         else x ← right[x]
p[z] ← y
if y = NIL
   then root[t] ← z
   else if key[z] < key[y]
        then  left[y] ← z
        else right[y] ← z
```

- Initialization: $O(1)$
- While loop in lines 3-7 searches for place to insert z, maintaining parent y. This takes $O(h)$ time.
- Lines 8-13 insert the value: $O(1)$

  $\Rightarrow$ TOTAL: $O(h)$ time to insert a node.

DELETION:

- if x has no children ♦ case 0
  then remove x
- if x has one child ♦ case 1
  then make $p[x]$ point to child
- if x has two children (subtrees) ♦ case 2
  then swap x with its successor
  - perform case 0 or case 1 to delete it

TOTAL: O(h) time to delete a node

Tree-Delete(T, z)

```
/* Determine which node to splice out: either z or z's successor. */
1.  if left[z] = NIL or right[z] = NIL
2.      then y ← z
3.      else y ← Tree-Successor[z]
          a.  /* Set x to a non-NIL child of x, or to NIL if y has no children. */
4.  if  left[y] ≠ NIL
5.      then x ← left[y]
6.      else x ← right[y]
    /* y is removed from the tree by manipulating pointers of  p[y] and x */
7.  if x ≠ NIL
```

```
8.      then p[x] ← p[y]
    /* Continued on next slide */
9.  if p[y] = NIL
10.     then root[T] ← x
11.     else if  y ← left[p[i]]
12.          then left[p[y]] ← x
13.          else right[p[y]] ← x
    /* If z's successor was spliced out, copy its data into z */
14. if y ≠ z
15.     then  key[z] ← key[y]
16.           copy y's satellite data into z.
17. return y
```

Red Black Tree

Red black tree is another variant of binary search tree in which every node is colored either red or black we can define a red black tree as follows:

Red black tree is a binary search tree in which every node is colored either red or black.

A red-black tree's node structure would be:

```
struct t_red_black_node {
    enum { red, black } colour;
    void *item;
    struct t_red_black_node *left,
                    *right,
                    *parent;
    }
```

In red black tree the color of node is decided based on the properties of red black tree. Every red black tree has the following properties:

1. Red black tree must be a binary search tree.

2.The root node must be colored black.

3. The children of red color node must be colored black. There should not be two consecutive red nodes.

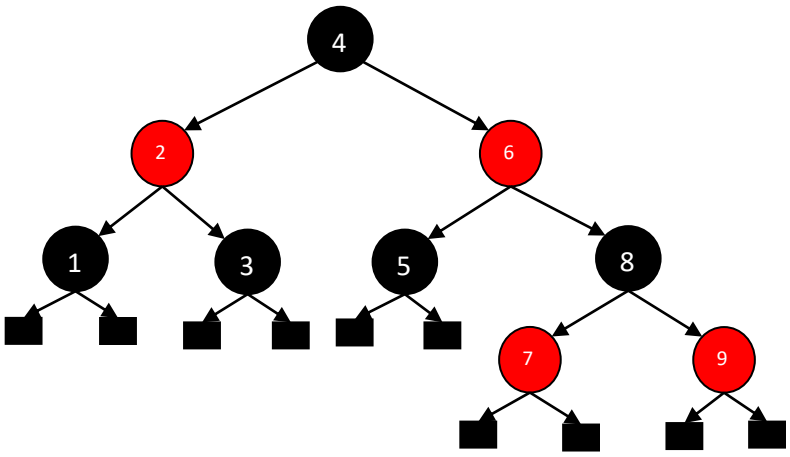
4. In all the paths of the tree there should be same number of black color nodes.

5. Every new node must be inserted with red color.

6. Every leaf(i.e null node) must be colored black.

Example:

Following is a red black tree which is created by inserting number from 1 to 9.



The above tree is a red black tree where every node is satisfying all the properties of red black tree.

Insertion into red black tree:

In a red black tree, every new node must be inserted with the color red. The insertion operation in red black tree is similar to insertion operation in binary search tree. But it is inserted with a color property. After every insertion operation, we need to check all the properties of red black tree. If all the properties are satisfied then we go to next operation otherwise we perform the following operation to make it red black tree.

The operations are

1. Recolor

2. Rotation

3. Rotation followed by recolor.

The insertion operation in red black tree is performed using the following steps:

Step 1: Check whether tree is empty.

Step2: If tree is empty when insert the **newnode** as root node with color black an exit from the operation.

Step3: If tree is not empty then insert the **newnode** as leaf node with color red.

Step4: If the parent of **newnode** is black then exit from the operation.

Step5: If the parent of **newnode** is r red then change the color of parent node's sibling of **newnode**.

Step6: If it is colored black or null then make suitable rotation and recolor it.

Step7: If it is colored red then perform recolor.

Repeat the same until tree becomes red black tree.

Example:

Create a red black tree by inserting following sequence of number :-

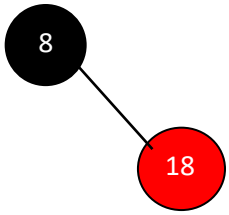
8, 18, 5, 15, 17, 25, 40, and 80

Insert (8)

Tree is empty. So insert newnode as root node with black color.

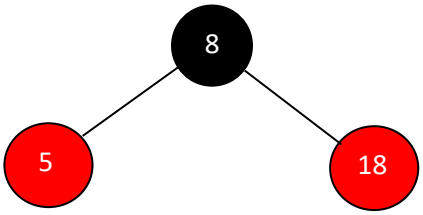
Insert (18)

Tree is not empty. So insert newnode with red color.



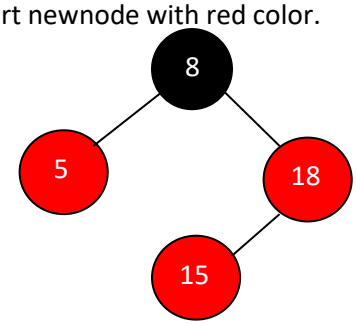
Insert (5)

Tree is not empty. So insert newnode with red color.



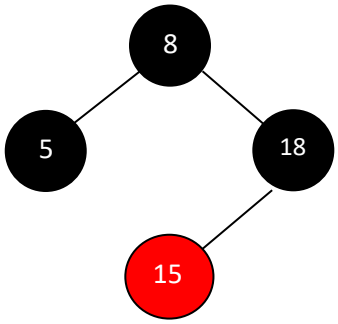
Insert (15)

Tree is not empty. So insert newnode with red color.



Here there are two consecutive red nodes 18 and 15. The newnode's parent sibling color is red and parent's parent is root node. So we use recolor to make it red black tree.

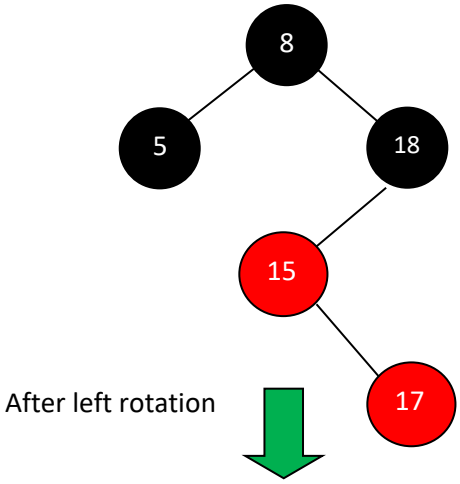
After recolor



After recolor operation, the trees satisfying all red black tree properties.

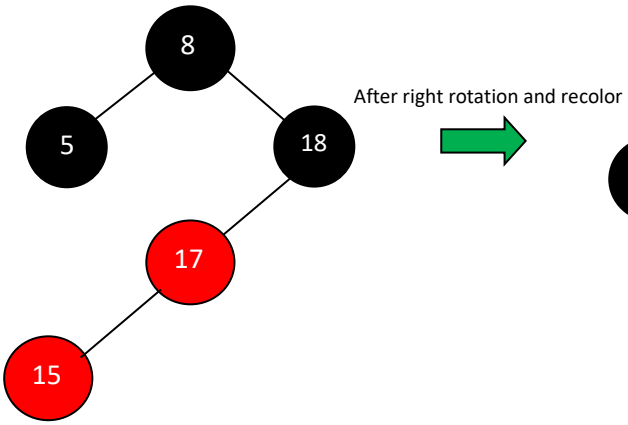
Insert (17)

The tree is not empty. So insert newnode with red color.

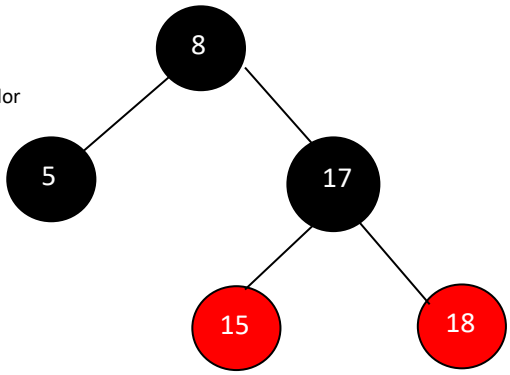


Here there two consecutive red nodes 15 and 17.The newnode's parent sibling is null. So we need rotation. Here we need LR rotation and recolor.

After left rotation

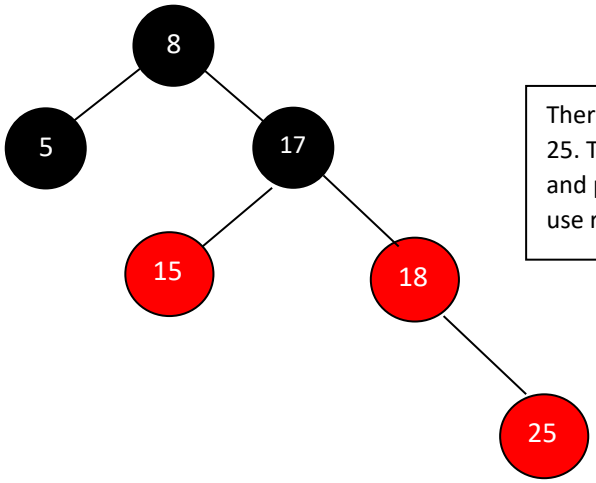


After right rotation and recolor



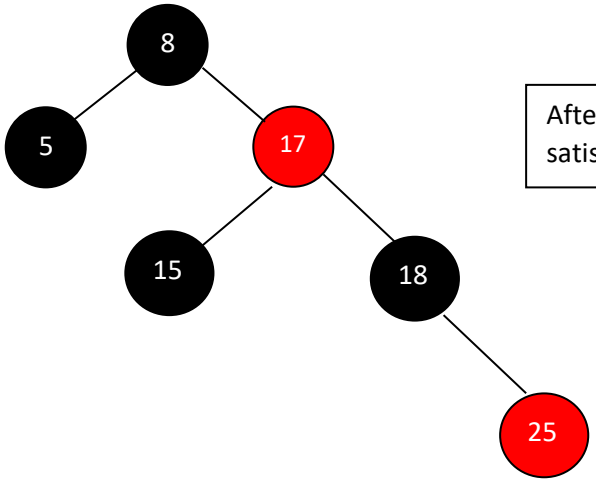
Insert (25)

Tree is not empty. So insert newnode with red color.



There are two consecutive red nodes 18 and 25. The newnode's parent sibling color is red and parent's parent is not root node. So we use recolor and recheck.

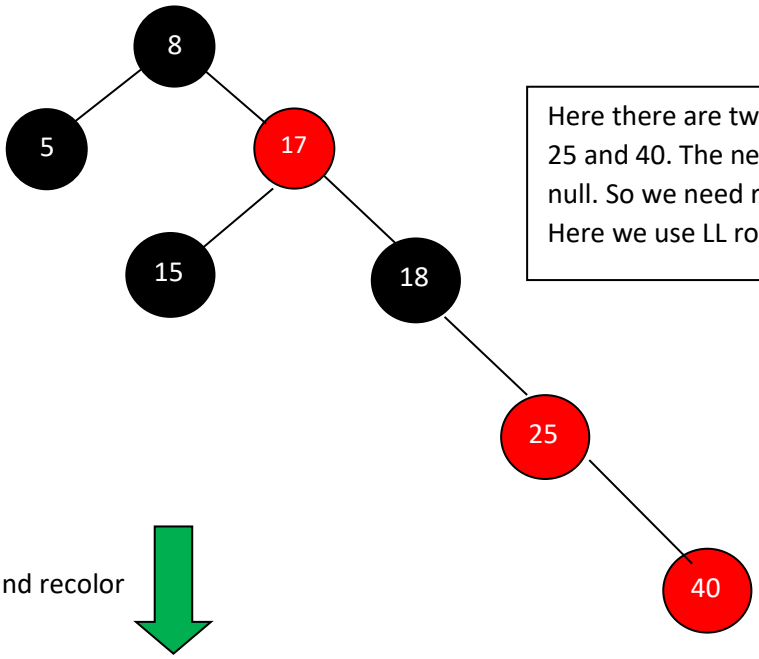
After recolor



After recolor operation, the tree is satisfying all red black tree properties.

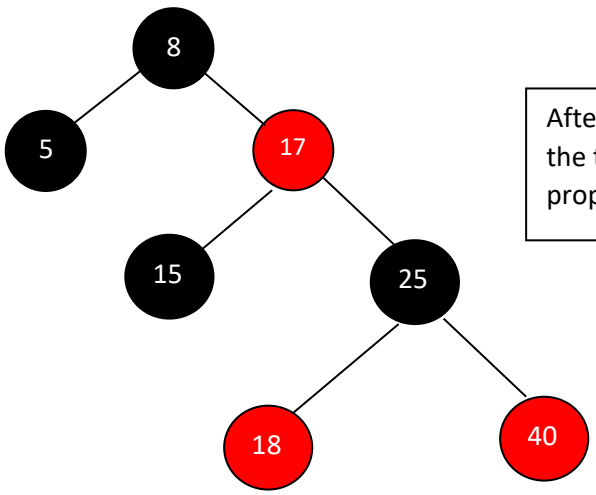
Insert(40)

Tree is not empty. So insert newnode with red color.



Here there are two consecutive red nodes 25 and 40. The newnode's parent sibling is null. So we need rotation and recolor. Here we use LL rotation and recheck.

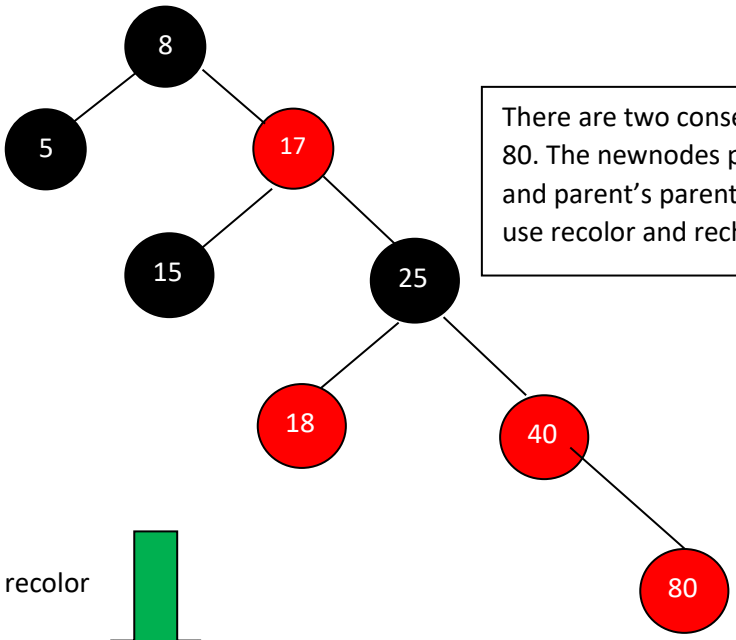
After LL rotation and recolor



After LL rotation and recolor operation, the tree satisfying all red black tree properties.

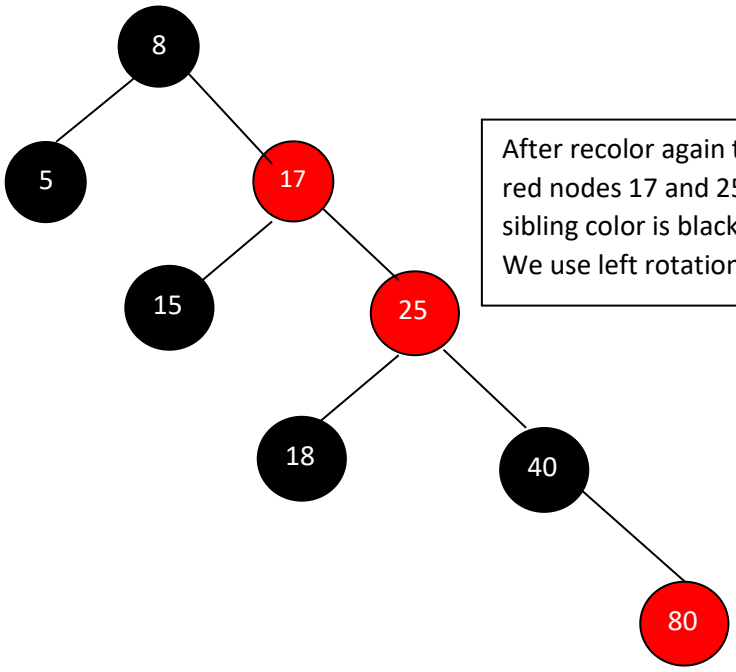
Insert(80)

Tree is not empty. So insert newnode with red color.



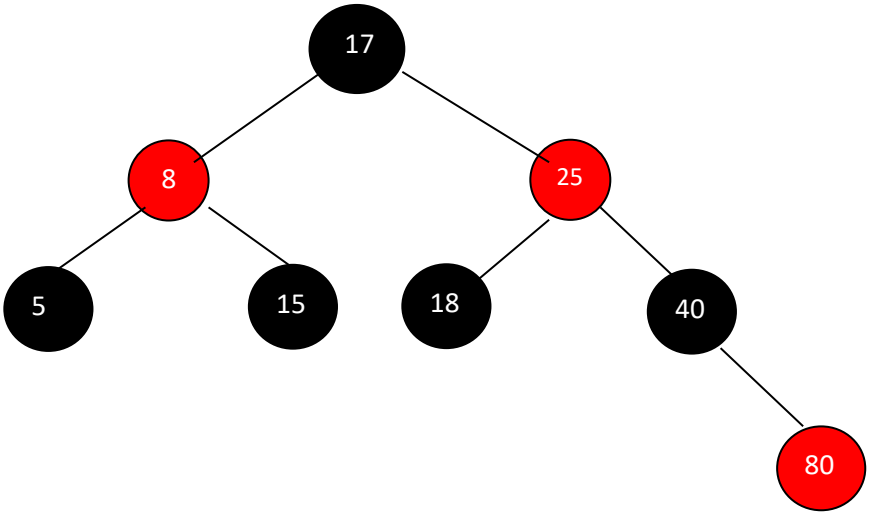
There are two consecutive red nodes 40 and 80. The newnodes parent sibling color is red and parent's parent is not root node. So we use recolor and recheck.

After recolor



After recolor again there are two consecutive red nodes 17 and 25. The newnode's parent sibling color is black. So we need rotation. We use left rotation And recolor.

After left rotation And recolor



Finally above tree is satisfying all the properties of red black tree and it is a perfect red black tree.

Deletion operation in red black tree

The deletion operation in red black tree is similar to deletion operation in BST. But after every deletion operation we need to check with the red black tree properties. If any of the properties violated then make suitable operations like recolor, rotation and rotation followed by recolor to make it red black tree.

Operations on Red Black Tree in details:

Rotations:

A rotation is a local operation in a search tree that preserves *in-order* traversal key ordering.

Note that in both trees, an in-order traversal yields:

A x B y C

The left_rotate operation may be encoded:

```
left_rotate( Tree T, node x ) {
    node y;
    y = x->right;
    /* Turn y's left sub-tree into x's right sub-tree */
    x->right = y->left;
    if ( y->left != NULL )
        y->left->parent = x;
    /* y's new parent was x's parent */
    y->parent = x->parent;
```

```
/* Set the parent to point to y instead of x */
/* First see whether we're at the root */
if ( x->parent == NULL ) T->root = y;
else
    if ( x == (x->parent)->left )
        /* x was on the left of its parent */
        x->parent->left = y;
    else
        /* x must have been on the right */
        x->parent->right = y;
/* Finally, put x on y's left */
y->left = x;
x->parent = y;
}
```

Insertion:

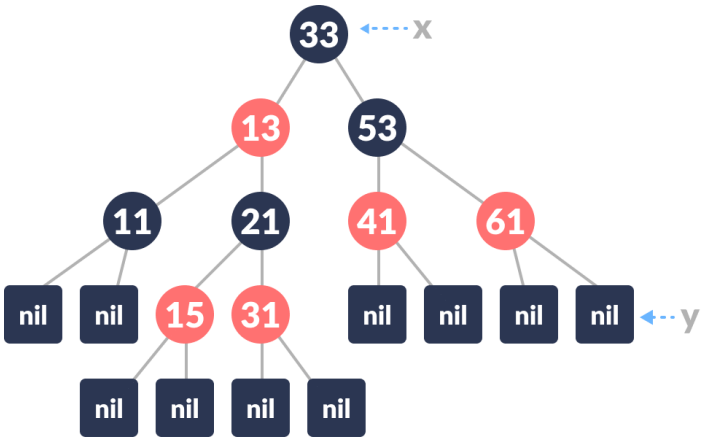
Insertion is somewhat complex and involves a number of cases. Note that we start by inserting the new node, `x`, in the tree just as we would for any other binary tree, using the `tree_insert` function. This new node is labelled red, and possibly destroys the red-black property. The main loop moves up the tree, restoring the red-black property.

Algorithm to Insert a New Node

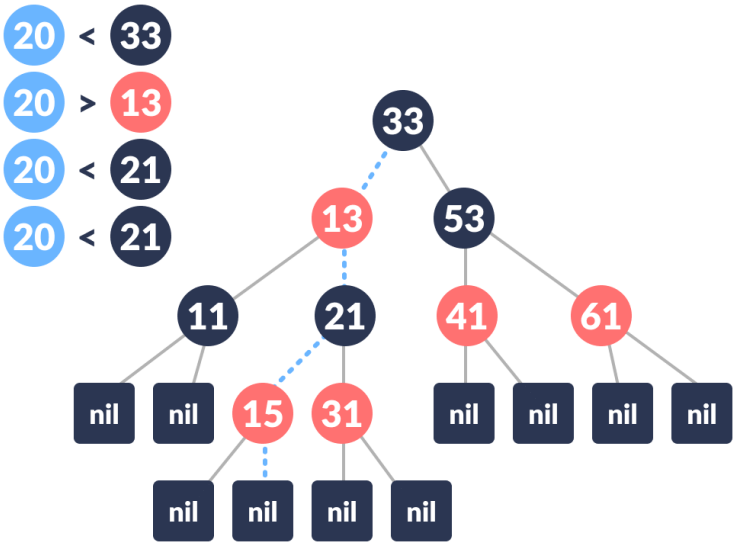
Following steps are followed for inserting a new element into a red-black tree:

1. The `newNode` be:

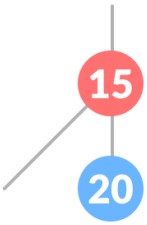
2. Let y be the leaf (ie. `NIL`) and `x` be the root of the tree. The new node is inserted in the following tree.



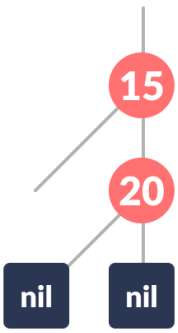
3. Check if the tree is empty (ie. whether `x` is `NIL`). If yes, insert `newNode` as a root node and color it black.
4. Else, repeat steps following steps until leaf (`NIL`) is reached.
    a. Compare `newKey` with `rootKey`.
    b. If `newKey` is greater than `rootKey`, traverse through the right subtree.
    c. Else traverse through the left subtree.



5. Assign the parent of the leaf as parent of `newNode`.
6. If `leafKey` is greater than `newKey`, make `newNode` as `rightChild`.
7. Else, make `newNode` as `leftChild`.



8. Assign `NULL` to the left and `rightChild` of `newNode`.
9. Assign RED color to `newNode`.



10. Call InsertFix-algorithm to maintain the property of red-black tree if violated.

Why newly inserted nodes are always red in a red-black tree?

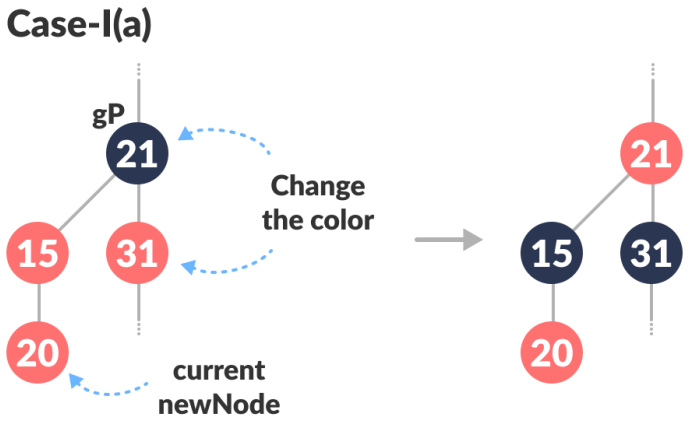
This is because inserting a red node does not violate the depth property of a red-black tree.

If you attach a red node to a red node, then the rule is violated but it is easier to fix this problem than the problem introduced by violating the depth property.

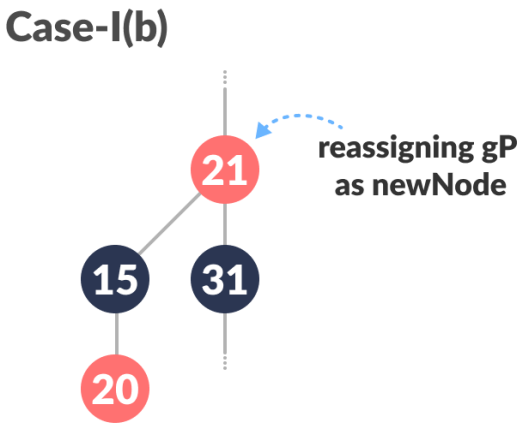
Algorithm to Maintain Red-Black Property After Insertion

This algorithm is used for maintaining the property of a red-black tree if insertion of a newNode violates this property.

1. Do the following until the parent of `newNode` `p` is RED.
2. If `p` is the left child of `grandParent` `gP` of `newNode`, do the following.
   Case-I:
   a. If the color of the right child of `gP` of `newNode` is RED, set the color of both the children of `gP` as BLACK and the color of `gP` as RED.

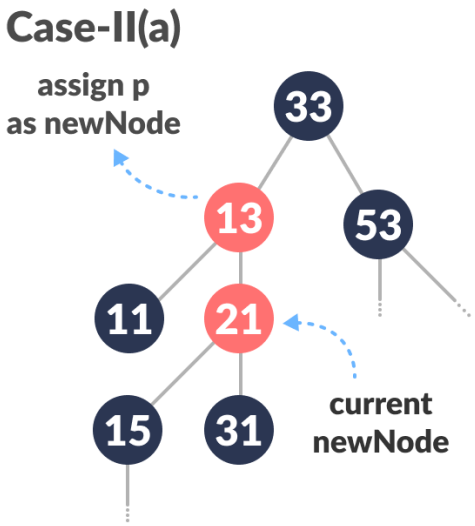


b. Assign `gP` to `newNode`.

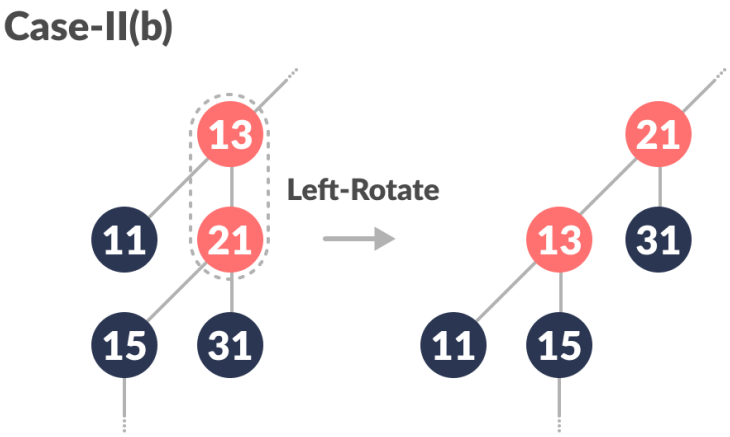


Case-II:

c. (Before moving on to this step, while loop is checked. If conditions are not satisfied, it the loop is broken.)
Else if `newNode` is the right child of `p` then, assign `p` to `newNode`.

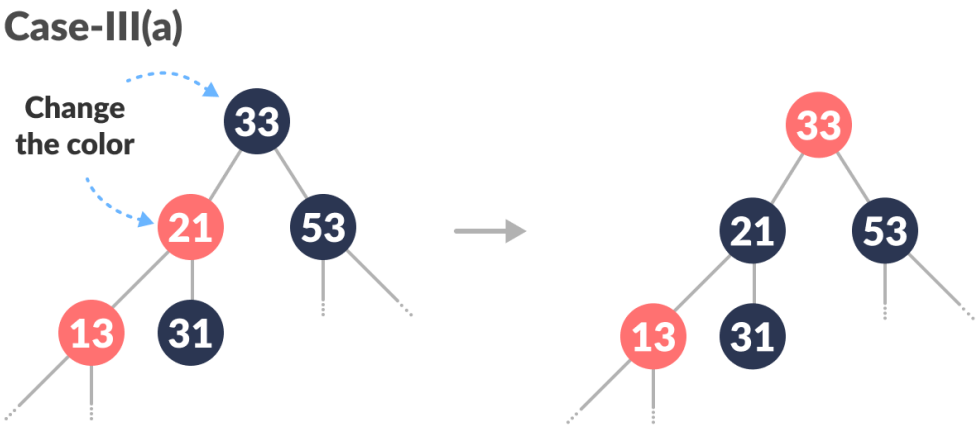


d. Left-Rotate `newNode`.

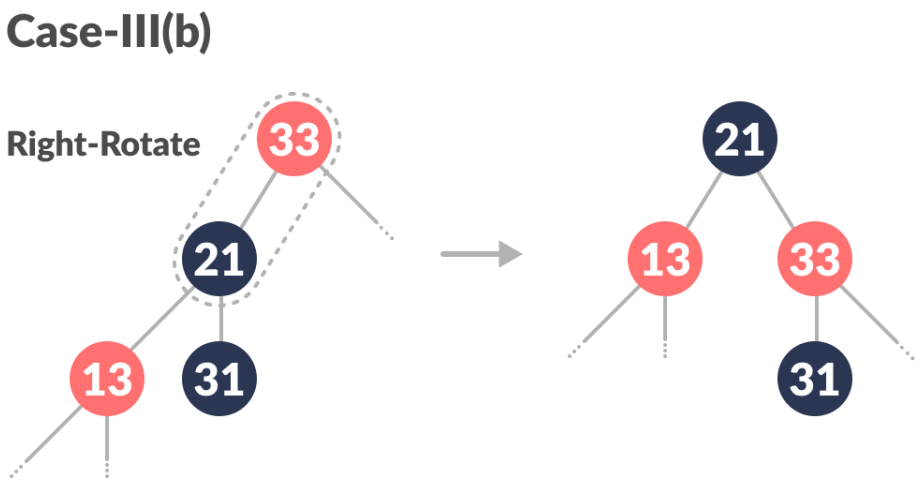


Case-III:

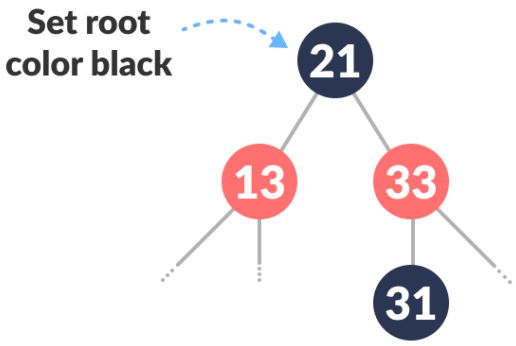
e. (Before moving on to this step, while loop is checked. If conditions are not satisfied, it the loop is broken.)
Set color of `p` as BLACK and color of `gP` as RED.



f. Right-Rotate `gP`.

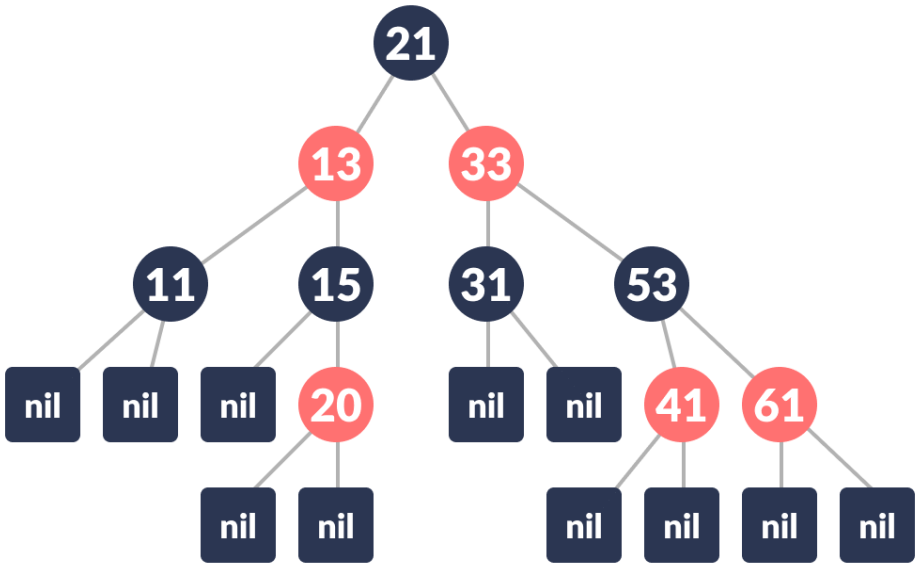


3. Else, do the following.
    a. If the color of the left child of `gP` of `z` is RED, set the color of both the children of `gP` as BLACK and the color of `gP` as RED.
    b. Assign `gP` to `newNode`.
    c. Else if `newNode` is the left child of `p` then, assign `p` to `newNode` and Right-Rotate `newNode`.
    d. Set color of `p` as BLACK and color of `gP` as RED.
    e. Left-Rotate `gP`.
4. (This step is perfomed after coming out of the while loop.)
   Set the root of the tree as BLACK.



The final tree look like this:

Final Tree



The insertion operation is encoded as:

```
rb_insert( Tree T, node x ) {
    /* Insert in the tree in the usual way */
    tree_insert( T, x );
    /* Now restore the red-black property */
    x->colour = red;
    while ( (x != T->root) && (x->parent->colour == red) ) {
       if ( x->parent == x->parent->parent->left ) {
           /* If x's parent is a left, y is x's right 'uncle' */
           y = x->parent->parent->right;
           if ( y->colour == red ) {
               /* case 1 - change the colours */
               x->parent->colour = black;
               y->colour = black;
               x->parent->parent->colour = red;
               /* Move x up the tree */
               x = x->parent->parent;
               }
           else {
               /* y is a black node */
               if ( x == x->parent->right ) {
                   /* and x is to the right */
                   /* case 2 - move x up and rotate */
                   x = x->parent;
                   left_rotate( T, x );
                   }
```

```
            /* case 3 */
            x->parent->colour = black;
            x->parent->parent->colour = red;
            right_rotate( T, x->parent->parent );
            }
        }
    else {
        /* repeat the "if" part with right and left
           exchanged */
        }
    }
/* Colour the root black */
T->root->colour = black;
}
```

Examination of the code reveals only one loop. In that loop, the node at the root of the sub-tree whose red-black property we are trying to restore, x, may be moved up the tree *at least one level* in each iteration of the loop. Since the tree originally has **O(log n)** height, there are **O(log n)** iterations. The tree_insert routine also has **O(log n)** complexity, so overall the rb_insert routine also has **O(log n)** complexity.

Red-black trees:

Trees which remain **balanced** - and thus guarantee **O(logn)** search times - in a dynamic environment. Or more importantly, since any tree can be re-balanced - but at considerable cost - can be re-balanced in **O(logn)** time.

Time complexity in big O notation

Algorithm	Average	Worst case
Space	O(*n*)	O(*n*)
Search	O(log *n*)[1]	O(log *n*)[1]
Insert	O(log *n*)[1]	O(log *n*)[1]
Delete	O(log *n*)[1]	O(log *n*)[1]

Applications:

Red black tree offer worst case guarantee for insertion time, deletion time and search time. Not only does this make them valuable in time sensitive applications such as real time applications but it makes them valuable building blocks in other data structures which provide worst case guarantees.

1. Most of the self-balancing BST library functions like map and set in C++ (OR TreeSet and TreeMap in Java) use Red Black Tree
2. It is used to implement CPU Scheduling Linux. Completely Fair Scheduler uses it.

B-TREES

DEFINITION:

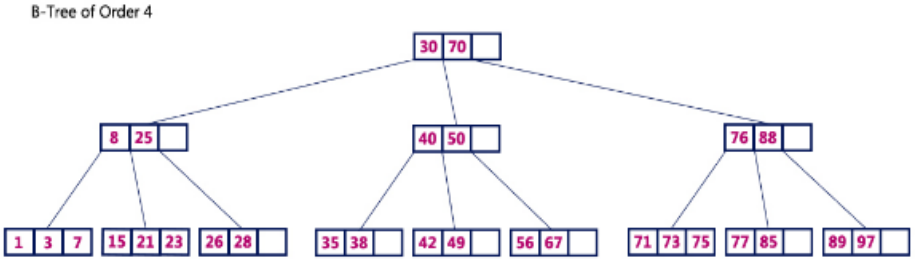
B-Tree is a self-balanced search tree in which every node contains multiple keys and has more than two children.

In search trees like binary search tree, AVL Tree, Red-Black tree, etc., every node contains only one value (key) and a maximum of two children. But there is a special type of search tree called B-Tree in which a node contains more than one value (key) and more than two children.

- Here, the number of keys in a node and number of children for a node depends on the order of B-Tree.
- Every B-Tree has an order.

For example, B-Tree of Order 4 contains a maximum of 3 key values in a node and maximum of 4 children for a node.

Example



B-Tree of Order m has the following properties...

- **Property #1** - All **leaf nodes** must be **at same level**.
- **Property #2** - All nodes except root must have at least **[m/2]-1** keys and maximum of **m-1** keys.
- **Property #3** - All non leaf nodes except root (i.e. all internal nodes) must have at least **m/2** children.
- **Property #4** - If the root node is a non leaf node, then it must have **atleast 2** children.
- **Property #5** - A non leaf node with **n-1** keys must have **n** number of children.
- **Property #6** - All the **key values in a node** must be in **Ascending Order**.

BASIC OPERATIONS ON B-TREES:

The following operations are performed on a B-Tree...

1. **Search**
2. **Insertion**
3. **Deletion**

Search Operation in B-Tree

The search operation in B-Tree is similar to the search operation in Binary Search Tree. In a Binary search tree, the search process starts from the root node and we make a 2-way decision every time (we go to either left subtree or right subtree). In B-Tree also search process starts from the root node but here we make an n-way decision every time. Where 'n' is the total number of children the node has. In a B-Tree, the search operation is performed with **O(log n)** time complexity. The search operation is performed as follows...

- **Step 1 -** Read the search element from the user.
- **Step 2 -** Compare the search element with first key value of root node in the tree.
- **Step 3 -** If both are matched, then display "Given node is found!!!" and terminate the function
- **Step 4 -** If both are not matched, then check whether search element is smaller or larger than that key value.
- **Step 5 -** If search element is smaller, then continue the search process in left subtree.
- **Step 6 -** If search element is larger, then compare the search element with next key value in the same node and repeate steps 3, 4, 5 and 6 until we find the exact match or until the search element is compared with last key value in the leaf node.
- **Step 7 -** If the last key value in the leaf node is also not matched then display "Element is not found" and terminate the function.

Insertion Operation in B-Tree

In a B-Tree, a new element must be added only at the leaf node. That means, the new keyValue is always attached to the leaf node only. The insertion operation is performed as follows...

- **Step 1 -** Check whether tree is Empty.
- **Step 2 -** If tree is **Empty**, then create a new node with new key value and insert it into the tree as a root node.

- **Step 3 -** If tree is **Not Empty**, then find the suitable leaf node to which the new key value is added using Binary Search Tree logic.
- **Step 4 -** If that leaf node has empty position, add the new key value to that leaf node in ascending order of key value within the node.
- **Step 5 -** If that leaf node is already full, **split** that leaf node by sending middle value to its parent node. Repeat the same until the sending value is fixed into a node.
- **Step 6 -** If the spilting is performed at root node then the middle value becomes new root node for the tree and the height of the tree is increased by one.

Example

Construct a **B-Tree of Order 3** by inserting numbers from 1 to 10.

Construct a B-Tree of order 3 by inserting numbers from 1 to 10.

insert(1)

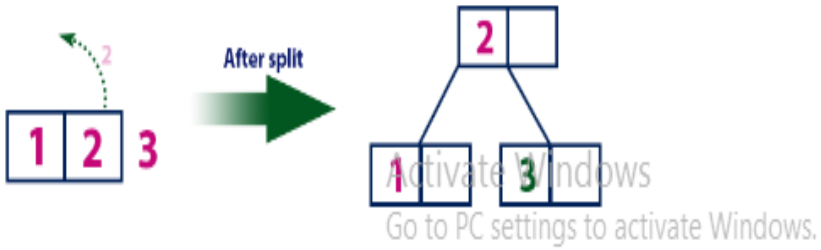
Since '1' is the first element into the tree that is inserted into a new node. It acts as the root node.

insert(2)

Element '2' is added to existing leaf node. Here, we have only one node and that node acts as root and also leaf. This leaf node has an empty position. So, new element (2) can be inserted at that empty position.

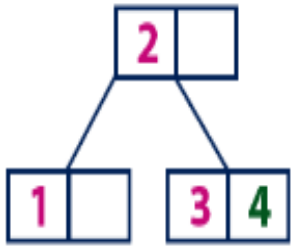
insert(3)

Element '3' is added to existing leaf node. Here, we have only one node and that node acts as root and also leaf. This leaf node doesn't has an empty position. So, we split that node by sending middle value (2) to its parent node. But here, this node doesn't has parent. So, this middle value becomes a new root node for the tree.



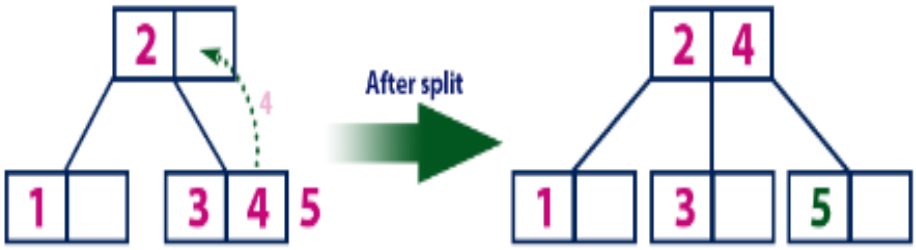
insert(4)

Element '4' is larger than root node '2' and it is not a leaf node. So, we move to the right of '2'. We reach to a leaf node with value '3' and it has an empty position. So, new element (4) can be inserted at that empty position.



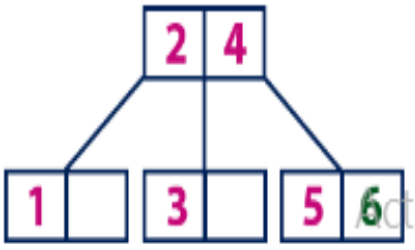
insert(5)

Element '5' is larger than root node '2' and it is not a leaf node. So, we move to the right of '2'. We reach to a leaf node and it is already full. So, we split that node by sending middle value (4) to its parent node (2). There is an empty position in its parent node. So, value '4' is added to node with value '2' and new element '5' added as new leaf node.



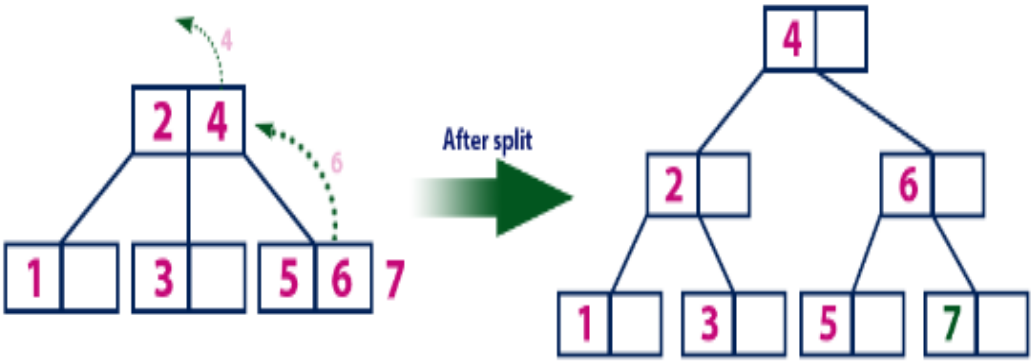
insert(6)

Element '6' is larger than root node '2' & '4' and it is not a leaf node. So, we move to the right of '4'. We reach to a leaf node with value '5' and it has an empty position. So, new element (6) can be inserted at that empty position.



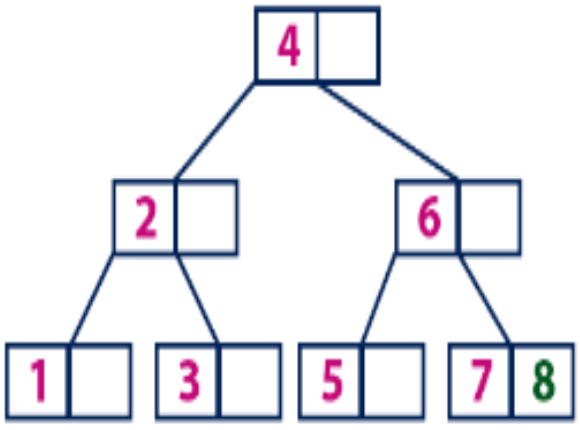
insert(7)

Element '7' is larger than root node '2' & '4' and it is not a leaf node. So, we move to the right of '4'. We reach to a leaf node and it is already full. So, we split that node by sending middle value (6) to its parent node (2&4). But the parent (2&4) is also full. So, again we split the node (2&4) by sending middle value '4' to its parent but this node doesn't have parent. So, the element '4' becomes new root node for the tree.



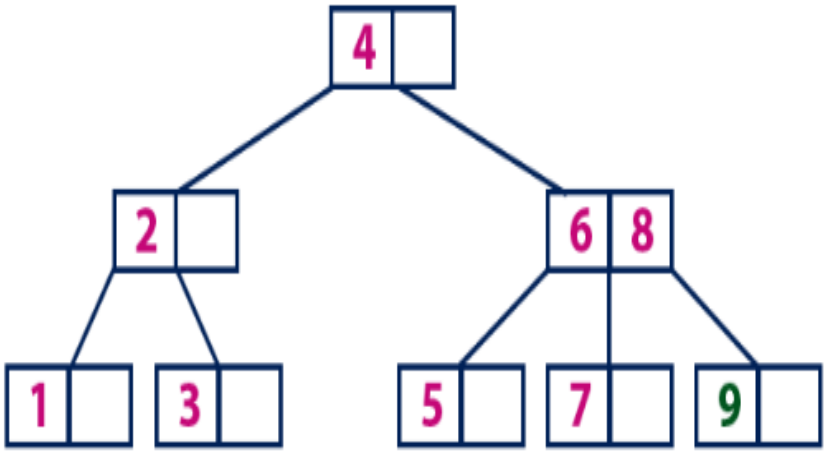
insert(8)

Element '8' is larger than root node '4' and it is not a leaf node. So, we move to the right of '4'. We reach to a node with value '6'. '8' is larger than '6' and it is also not a leaf node. So, we move to the right of '6'. We reach to a leaf node (7) and it has an empty position. So, new element (8) can be inserted at that empty position.



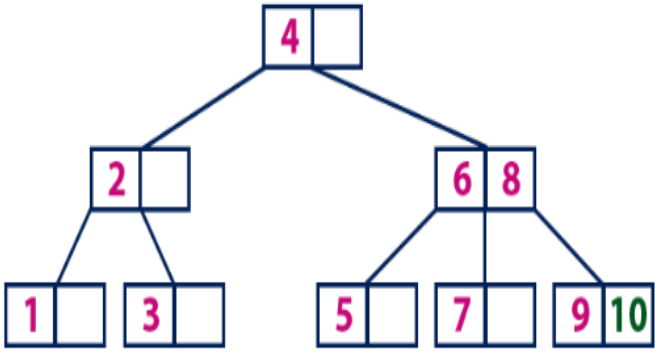
insert(9)

Element '9' is larger than root node '4' and it is not a leaf node. So, we move to the right of '4'. We reach to a node with value '6'. '9' is larger than '6' and it is also not a leaf node. So, we move to the right of '6'. We reach to a leaf node (7 & 8). This leaf node is already full. So, we split this node by sending middle value (8) to its parent node. The parent node (6) has an empty position. So, '8' is added at that position. And new element is added as a new leaf node.



insert(10)

Element '10' is larger than root node '4' and it is not a leaf node. So, we move to the right of '4'. We reach to a node with values '6 & 8'. '10' is larger than '6 & 8' and it is also not a leaf node. So, we move to the right of '8'. We reach to a leaf node (9). This leaf node has an empty position. So, new element '10' is added at that empty position.



DATA STRUCTURE - HEAP

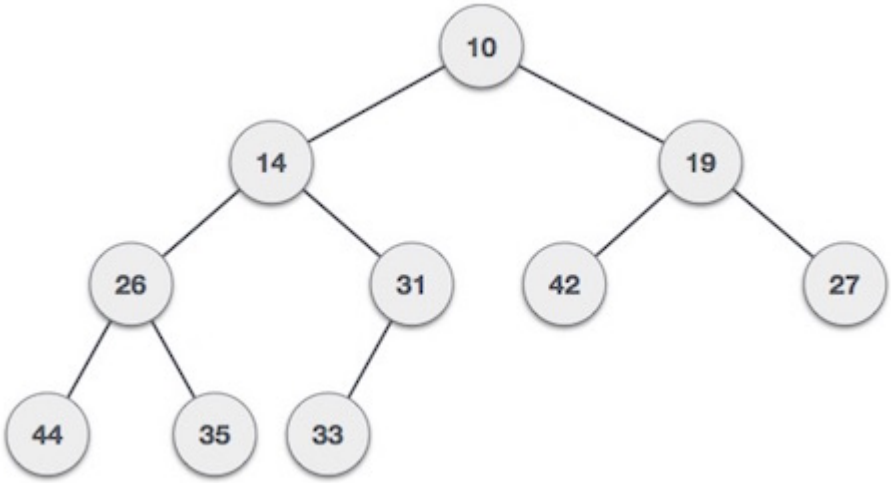
Heap is a special case of balanced binary tree data structure where root-node key is compared with its children and arranged accordingly. If α has child node β then −

key$_\alpha$ ≥ key$_\beta$

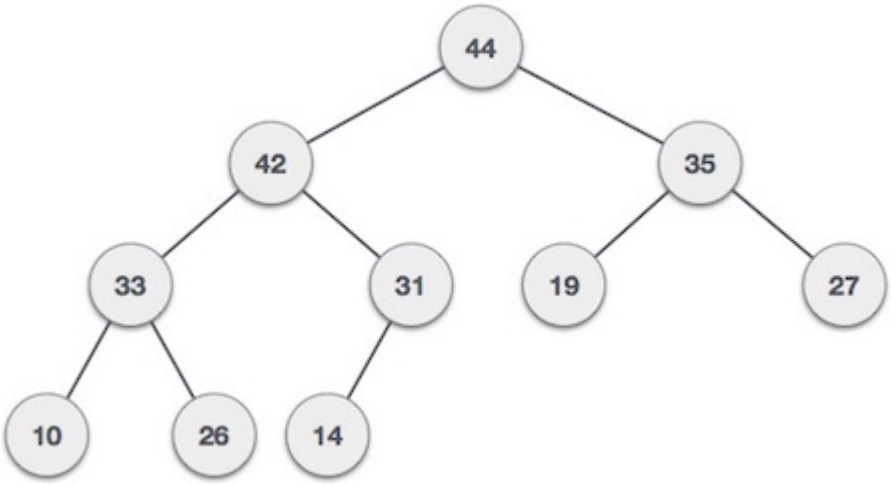
As the value of parent is greater than that of child, this property generates **Max Heap**. Based on this criteria a heap can be of two types −

```
For Input → 35 33 42 10 14 19 27 44 26 31
```

- **Min-Heap** − where the value of root node is less than or equal to either of its children.



- **Max-Heap** − where the value of root node is greater than or equal to either of its children.



Both trees are constructed using the same input and order of arrival.

Max Heap Construction Algorithm

We shall use the same example to demonstrate how a Max Heap is created. The procedure to create Min Heap is similar but we go for min values instead of max ones.

We are going to derive an algorithm for max-heap by inserting one element at a time. At any point of time, heap must maintain its property. While insertion, we also assume that we are inserting a node in already heapified tree.

```
Step 1 − Create a new node at the end of heap.
Step 2 − Assign new value to the node.
```

```
Step 3 - Compare the value of this child node with its parent.
Step 4 - If value of parent is less than child, then swap them.
Step 5 - Repeat step 3 & 4 until Heap property holds.
```

Note − In Min Heap construction algorithm we expect the value of parent node to be less than that of child node.

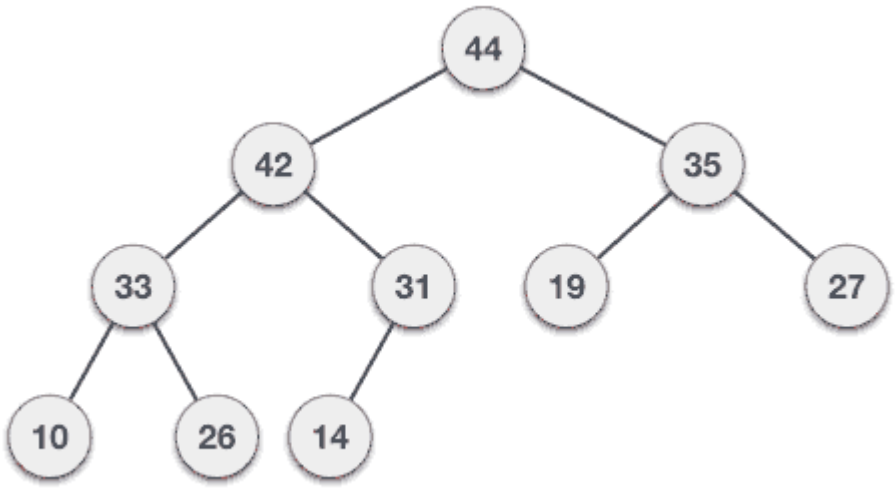
Let's understand Max Heap construction by an animated illustration. We take the same input sample that we use earlier.

Input 35 33 42 10 14 19 27 44 26 31

Max Heap Deletion Algorithm

Lets derive an algorithm to delete from max-heap. Deletion in Max *orMin* Heap is always happen at the root to remove the Maximum *orminimum* value.

```
Step 1 - Remove root node.
Step 2 - Move the last element of last level to root.
Step 3 - Compare the value of this child node with its parent.
Step 4 - If value of parent is less than child, then swap them.
Step 5 - Repeat step 3 & 4 until Heap property holds.
```



Fibonacci Heaps

Lecture slides adapted from:

- Chapter 20 of *Introduction to Algorithms* by Cormen, Leiserson, Rivest, and Stein.
- Chapter 9 of *The Design and Analysis of Algorithms* by Dexter Kozen.

Priority Queues Performance Cost Summary

Operation	Linked List	Binary Heap	Binomial Heap	Fibonacci Heap †	Relaxed Heap
make-heap	1	1	1	1	1
is-empty	1	1	1	1	1
insert	1	log *n*	log *n*	1	1
delete-min	*n*	log *n*	log *n*	log *n*	log *n*
decrease-key	*n*	log *n*	log *n*	1	1
delete	*n*	log *n*	log *n*	log *n*	log *n*
union	1	*n*	log *n*	1	1
find-min	*n*	1	log *n*	1	1

n = number of elements in priority queue † amortized

Theorem. Starting from empty Fibonacci heap, any sequence of a_1 *insert*, a_2 *delete-min*, and a_3 *decrease-key* operations takes $O(a_1 + a_2 \log n + a_3)$ time.

Priority Queues Performance Cost Summary

Operation	Linked List	Binary Heap	Binomial Heap	Fibonacci Heap †	Relaxed Heap
make-heap	1	1	1	1	1
is-empty	1	1	1	1	1
insert	1	log *n*	log *n*	1	1
delete-min	*n*	log *n*	log *n*	log *n*	log *n*
decrease-key	*n*	log *n*	log *n*	1	1
delete	*n*	log *n*	log *n*	log *n*	log *n*
union	1	*n*	log *n*	1	1
find-min	*n*	1	log *n*	1	1

n = number of elements in priority queue † amortized

Hopeless challenge. O(1) *insert*, *delete-min* and *decrease-key*. Why?

Fibonacci Heaps

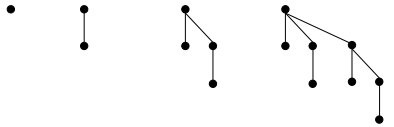
History. [Fredman and Tarjan, 1986]

- Ingenious data structure and analysis.
- Original motivation: improve Dijkstra's shortest path algorithm from $O(E \log V)$ to $O(E + V \log V)$.

V insert, V delete-min, E decrease-key

Basic idea.

- Similar to binomial heaps, but less rigid structure.
- Binomial heap: eagerly consolidate trees after each *insert*.



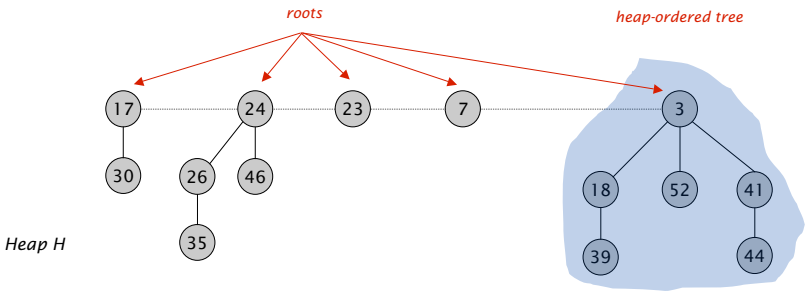
- Fibonacci heap: lazily defer consolidation until next *delete-min*.

Fibonacci Heaps: Structure

Fibonacci heap.

- Set of heap-ordered trees.
- Maintain pointer to minimum element.
- Set of marked nodes.

each parent larger than its children

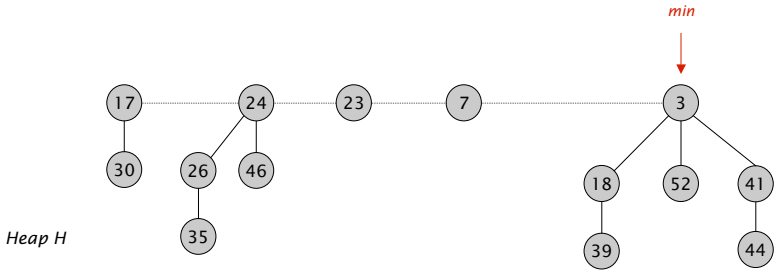


Fibonacci Heaps: Structure

Fibonacci heap.

- Set of heap-ordered trees.
- Maintain pointer to minimum element.
- Set of marked nodes.

find-min takes O(1) time

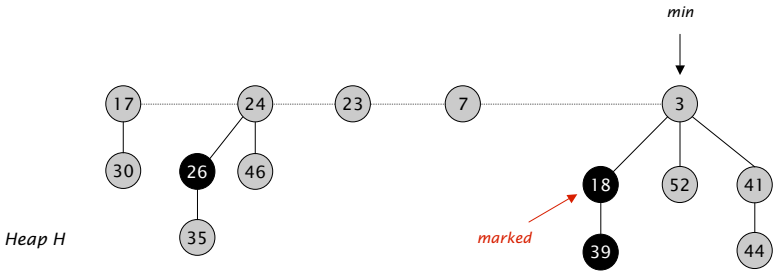


Fibonacci Heaps: Structure

Fibonacci heap.

- Set of heap-ordered trees.
- Maintain pointer to minimum element.
- Set of marked nodes.

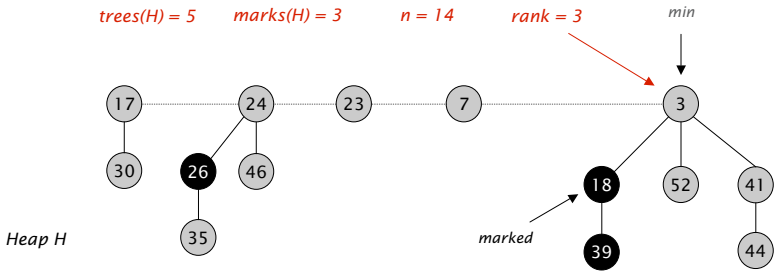
use to keep heaps flat (stay tuned)



Fibonacci Heaps: Notation

Notation.

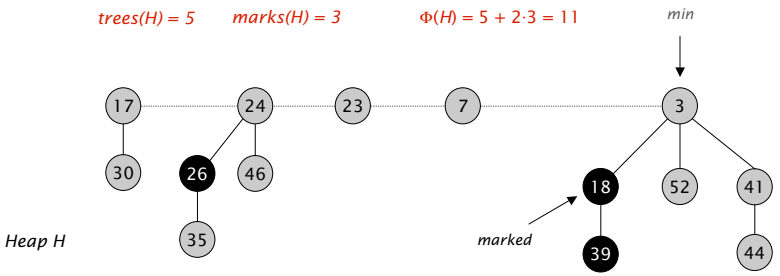
- *n* = number of nodes in heap.
- *rank(x)* = number of children of node *x*.
- *rank(H)* = max rank of any node in heap *H*.
- *trees(H)* = number of trees in heap *H*.
- *marks(H)* = number of marked nodes in heap *H*.



Fibonacci Heaps: Potential Function

$\Phi(H) = trees(H) + 2 \cdot marks(H)$

potential of heap H

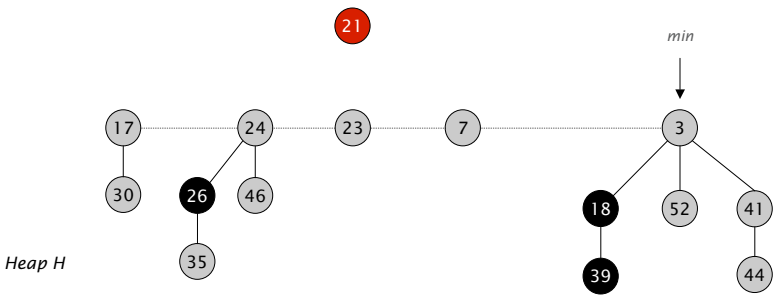


Insert

Fibonacci Heaps: Insert

Insert.
- Create a new singleton tree.
- Add to root list; update min pointer (if necessary).

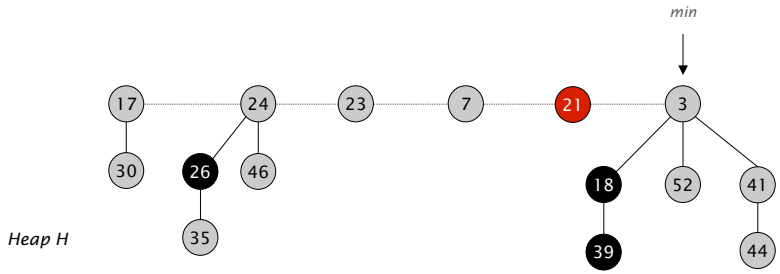
insert 21



Fibonacci Heaps: Insert

Insert.
- Create a new singleton tree.
- Add to root list; update min pointer (if necessary).

insert 21



Fibonacci Heaps: Insert Analysis

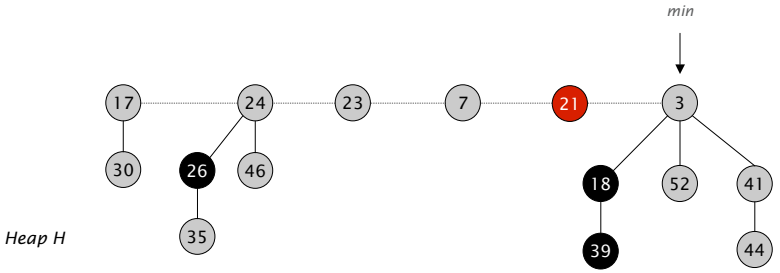
Actual cost. O(1)

Change in potential. +1

Amortized cost. O(1)

$$\Phi(H) = trees(H) + 2 \cdot marks(H)$$

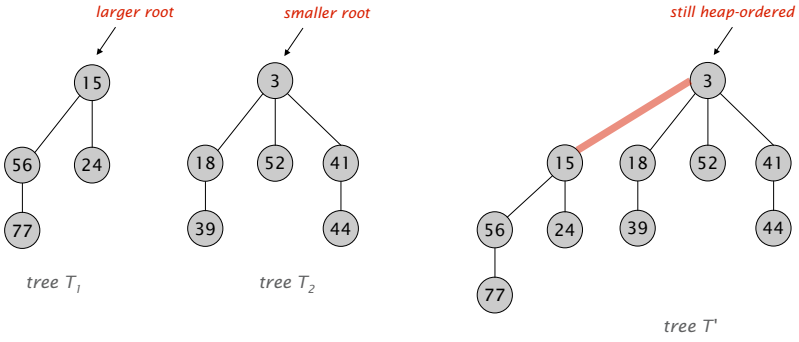
potential of heap H



Delete Min

Linking Operation

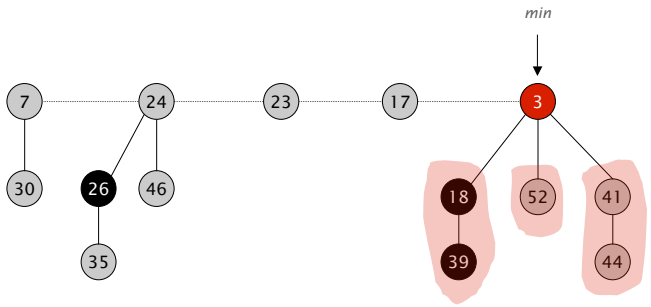
Linking operation. Make larger root be a child of smaller root.



Fibonacci Heaps: Delete Min

Delete min.

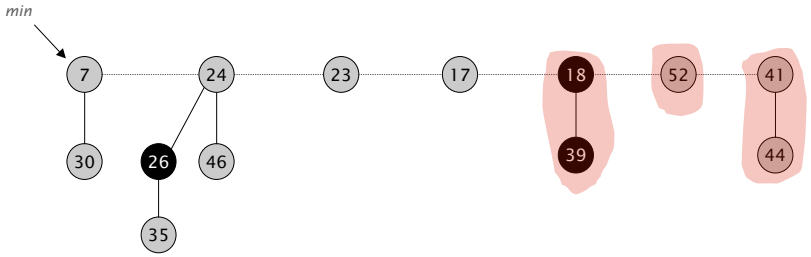
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

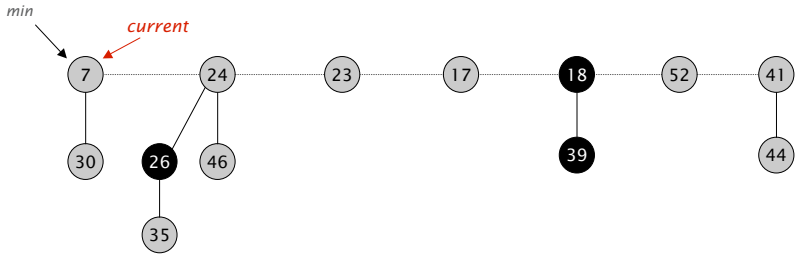
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

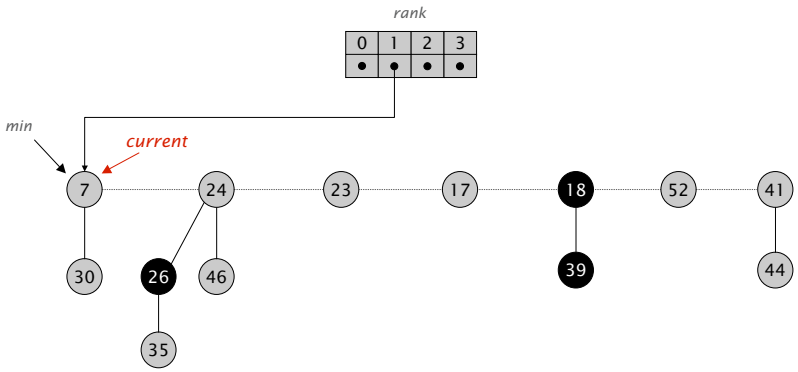
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

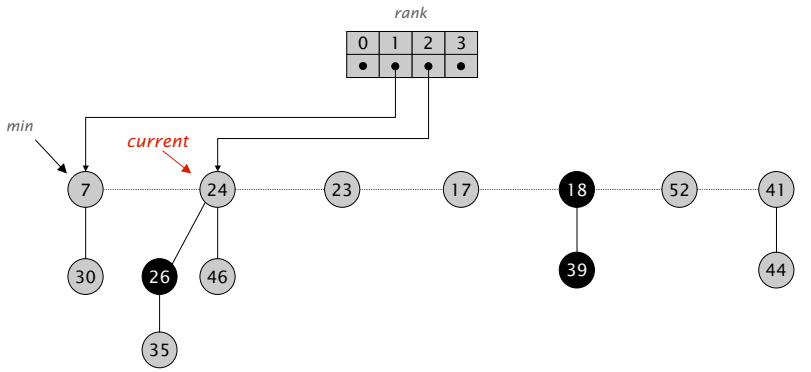
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

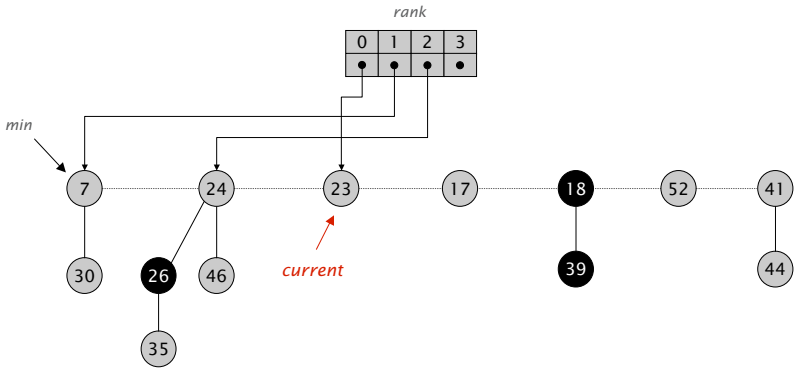
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

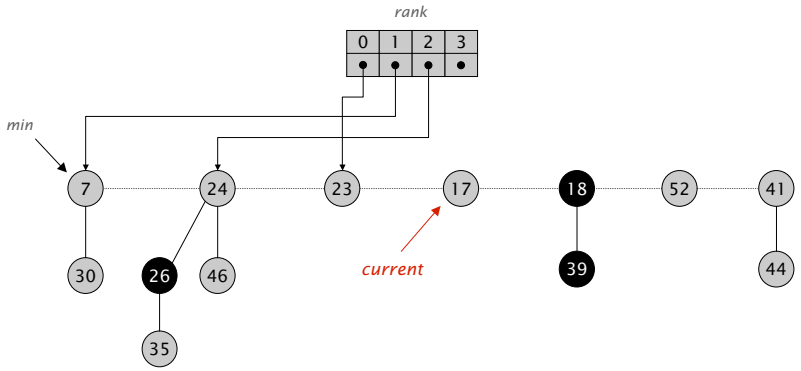
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.

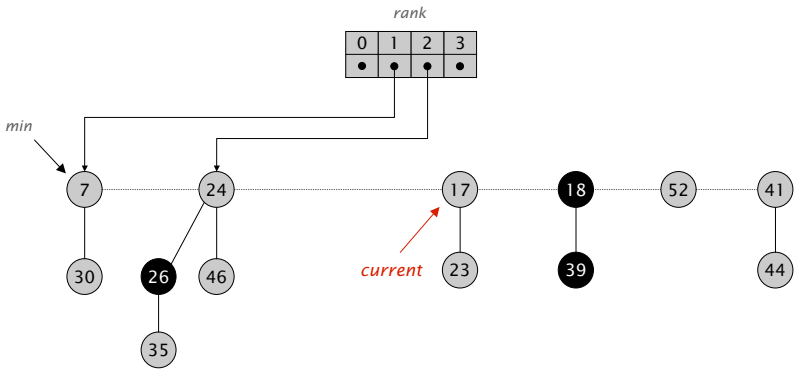


link 23 into 17

Fibonacci Heaps: Delete Min

Delete min.

- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.

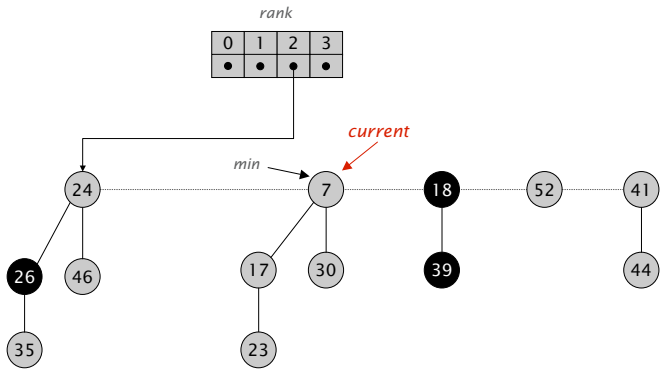


link 17 into 7

Fibonacci Heaps: Delete Min

Delete min.

- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.

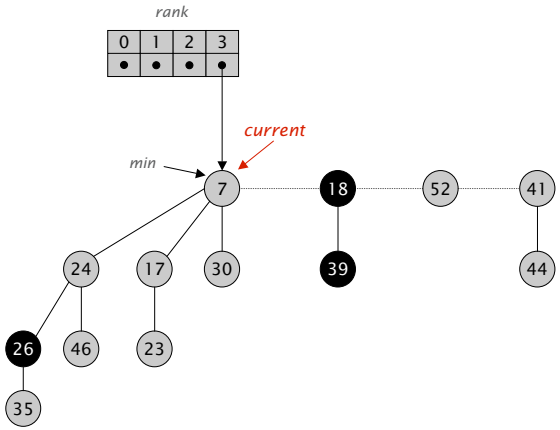


link 24 into 7

Fibonacci Heaps: Delete Min

Delete min.

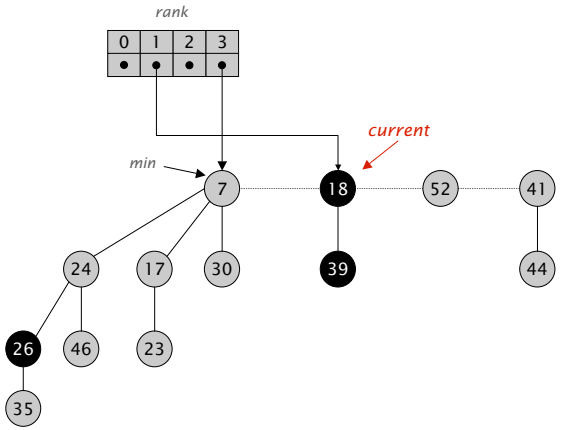
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

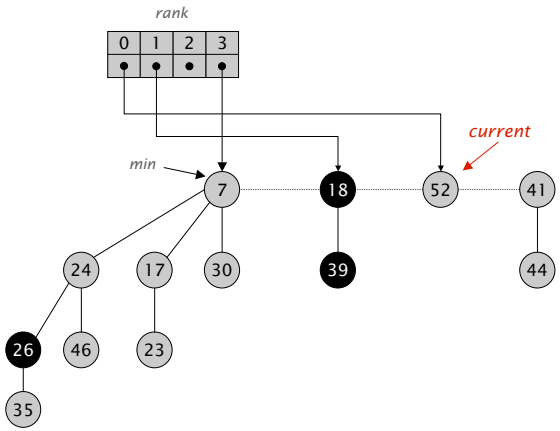
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

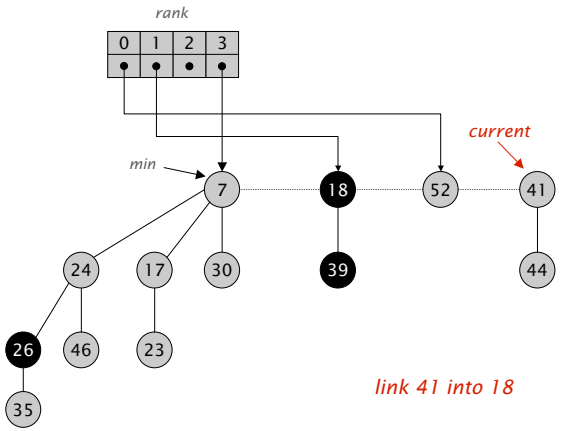
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

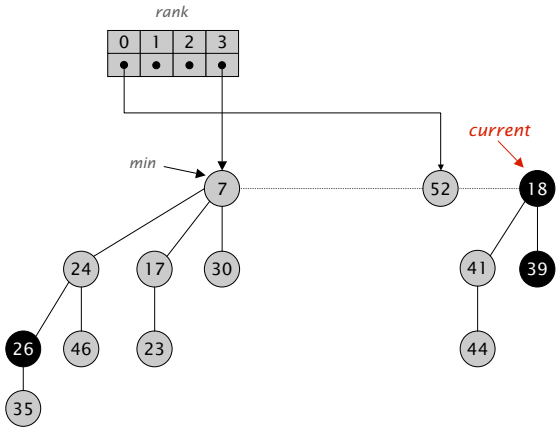
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

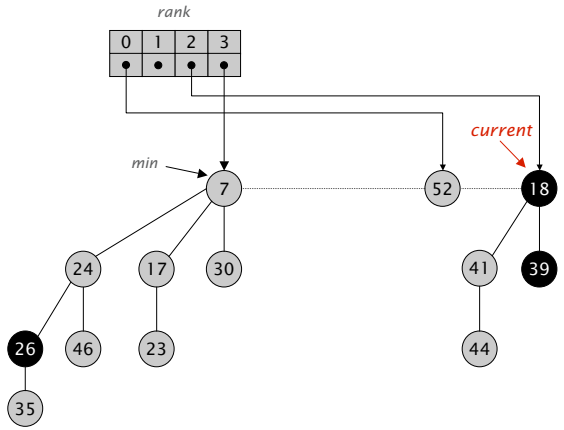
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

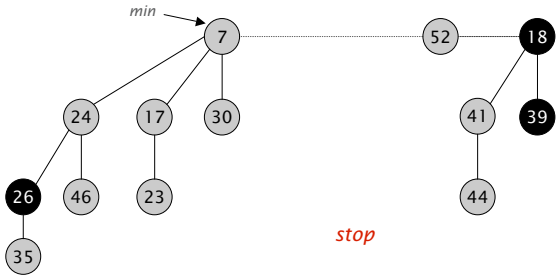
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min Analysis

Delete min.

$\Phi(H) = trees(H) + 2 \cdot marks(H)$

potential function

Actual cost. $O(rank(H)) + O(trees(H))$

- $O(rank(H))$ to meld min's children into root list.
- $O(rank(H)) + O(trees(H))$ to update min.
- $O(rank(H)) + O(trees(H))$ to consolidate trees.

Change in potential. $O(rank(H)) - trees(H)$

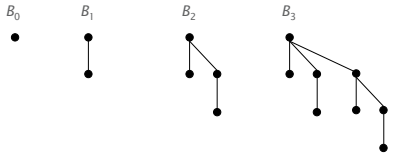
- $trees(H') \le rank(H) + 1$ since no two trees have same rank.
- $\Delta\Phi(H) \le rank(H) + 1 - trees(H)$.

Amortized cost. $O(rank(H))$

Q. Is amortized cost of O(*rank*(*H*)) good?

A. Yes, if only *insert* and *delete-min* operations.
- In this case, all trees are binomial trees.
- This implies *rank*(*H*) ≤ lg *n*.

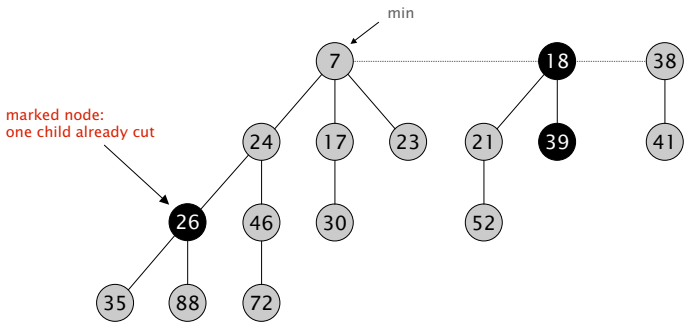
we only link trees of equal rank



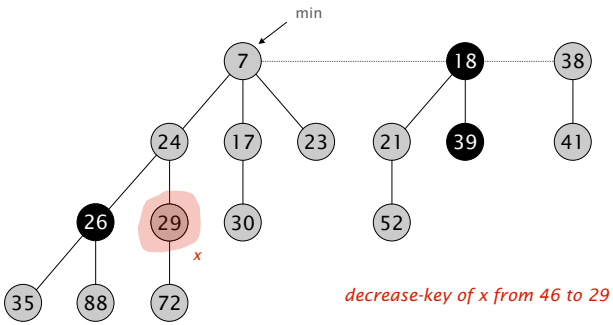
A. Yes, we'll implement *decrease-key* so that *rank*(*H*) = O(log *n*).

Decrease Key

Intuition for deceasing the key of node *x*.
- If heap-order is not violated, just decrease the key of *x*.
- Otherwise, cut tree rooted at *x* and meld into root list.
- To keep trees flat: as soon as a node has its second child cut, cut it off and meld into root list (and unmark it).



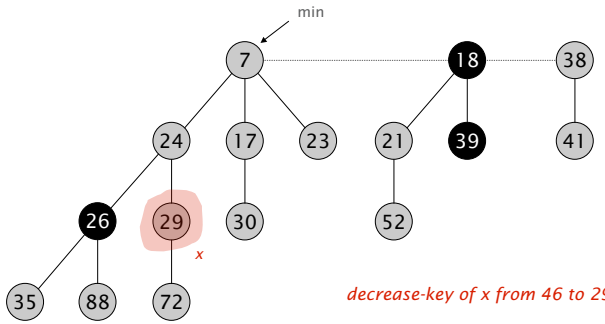
Case 1. [heap order not violated]
- Decrease key of *x*.
- Change heap min pointer (if necessary).



Fibonacci Heaps: Decrease Key

Case 1. [heap order not violated]

- Decrease key of *x*.
- Change heap min pointer (if necessary).

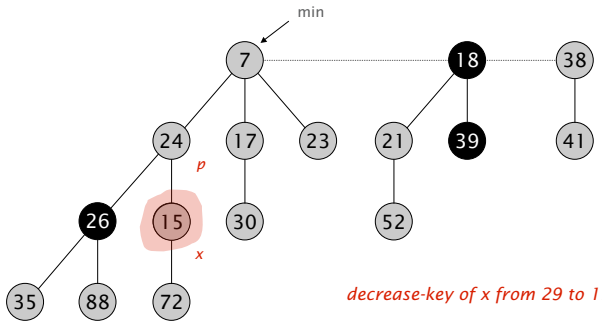


decrease-key of x from 46 to 29

Fibonacci Heaps: Decrease Key

Case 2a. [heap order violated]

- Decrease key of *x*.
- Cut tree rooted at *x*, meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut *p*, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).

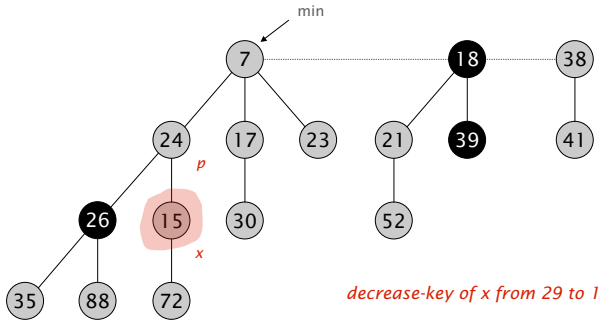


decrease-key of x from 29 to 15

Fibonacci Heaps: Decrease Key

Case 2a. [heap order violated]

- Decrease key of *x*.
- Cut tree rooted at *x*, meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut *p*, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).

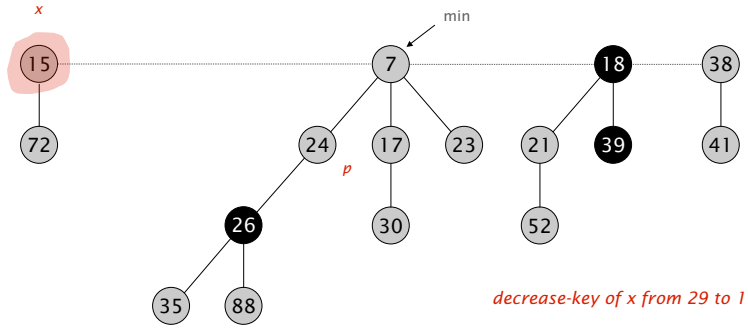


decrease-key of x from 29 to 15

Fibonacci Heaps: Decrease Key

Case 2a. [heap order violated]

- Decrease key of *x*.
- Cut tree rooted at *x*, meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut *p*, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).

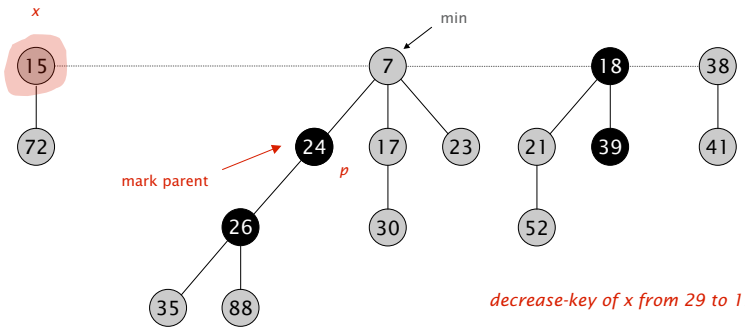


decrease-key of x from 29 to 15

Fibonacci Heaps: Decrease Key

Case 2a. [heap order violated]

- Decrease key of *x*.
- Cut tree rooted at *x*, meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut *p*, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).

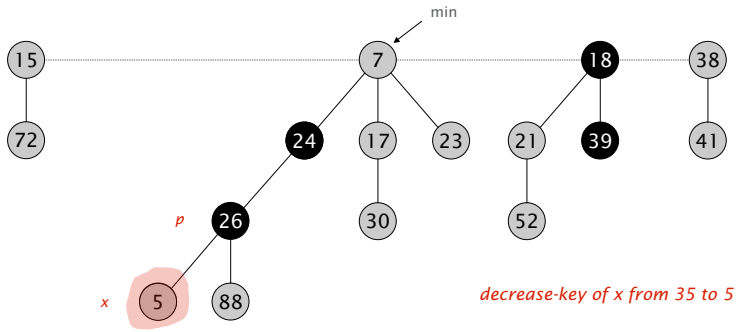


decrease-key of x from 29 to 15

Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

- Decrease key of *x*.
- Cut tree rooted at *x*, meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut *p*, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).

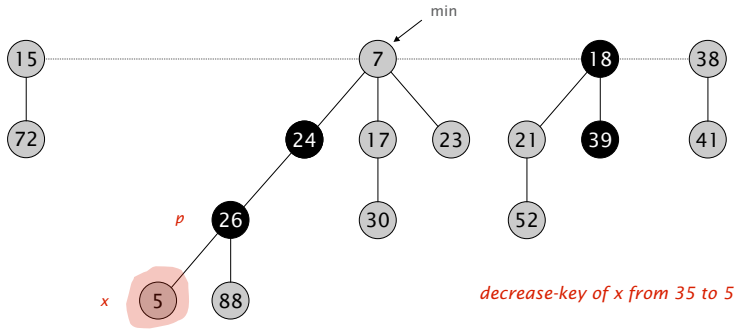


decrease-key of x from 35 to 5

Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

- Decrease key of *x*.
- Cut tree rooted at *x*, meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut *p*, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).

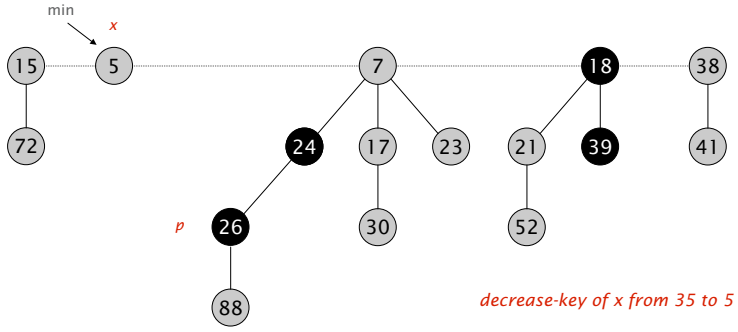


decrease-key of x from 35 to 5

Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

- Decrease key of *x*.
- Cut tree rooted at *x*, meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut *p*, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).

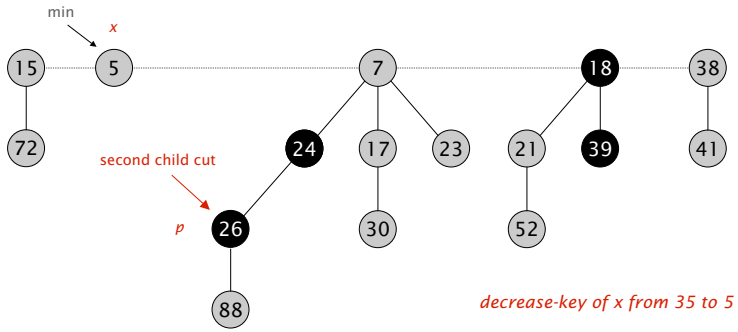


decrease-key of x from 35 to 5

Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

- Decrease key of x.
- Cut tree rooted at x, meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut p, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).

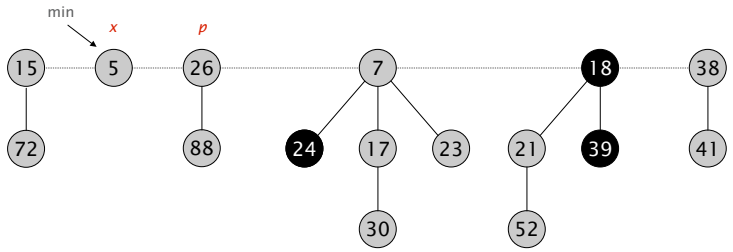


decrease-key of x from 35 to 5

Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

- Decrease key of x.
- Cut tree rooted at x, meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut p, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).

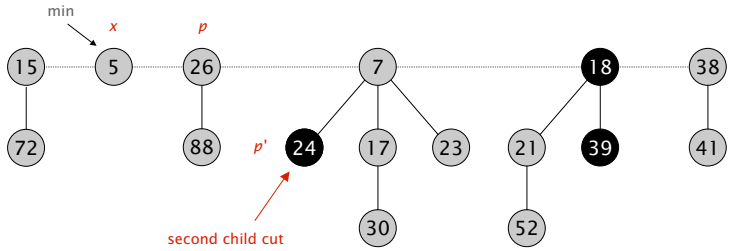


decrease-key of x from 35 to 5

Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

- Decrease key of x.
- Cut tree rooted at x, meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut p, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).

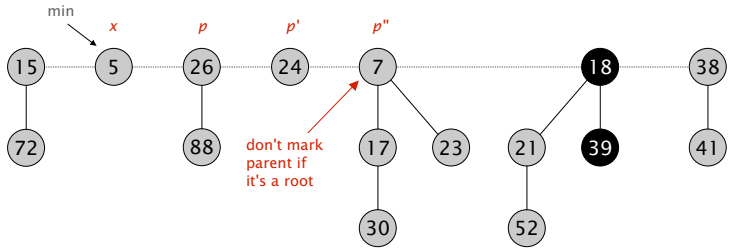


decrease-key of x from 35 to 5

Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

- Decrease key of x.
- Cut tree rooted at x, meld into root list, and unmark.
- If parent p of x is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut p, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



decrease-key of x from 35 to 5

Fibonacci Heaps: Decrease Key Analysis

Decrease-key.

$$\Phi(H) = trees(H) + 2 \cdot marks(H)$$

potential function

Actual cost. O(*c*)

- O(1) time for changing the key.
- O(1) time for each of *c* cuts, plus melding into root list.

Change in potential. O(1) - *c*

- $trees(H') = trees(H) + c$.
- $marks(H') \le marks(H) - c + 2$.
- $\Delta\Phi \le c + 2 \cdot (-c + 2) = 4 - c$.

Amortized cost. O(1)

Analysis

Analysis Summary

Insert.	O(1)
Delete-min.	O(*rank*(*H*)) †
Decrease-key.	O(1) †

† amortized

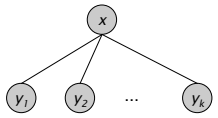
Key lemma. $rank(H) = O(\log n)$.

number of nodes is exponential in rank

Fibonacci Heaps: Bounding the Rank

Lemma. Fix a point in time. Let *x* be a node, and let $y_1, \ldots, y_k$ denote its children in the order in which they were linked to *x*. Then:

$$rank(y_i) \ge \begin{cases} 0 & \text{if } i = 1 \\ i - 2 & \text{if } i \ge 1 \end{cases}$$



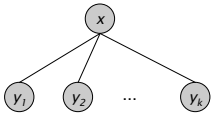
Pf.

- When y_i was linked into *x*, *x* had at least *i* -1 children $y_1, \ldots, y_{i-1}$.
- Since only trees of equal rank are linked, at that time $rank(y_i) = rank(x_i) \ge i - 1$.
- Since then, y_i has lost at most one child.
- Thus, right now $rank(y_i) \ge i - 2$. ■

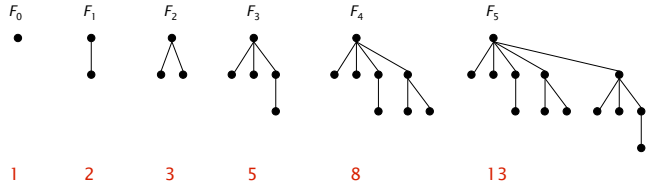
or y_i would have been cut

Lemma. Fix a point in time. Let x be a node, and let $y_1, \ldots, y_k$ denote its children in the order in which they were linked to x. Then:

$$rank(y_i) \geq \begin{cases} 0 & \text{if } i = 1 \\ i-2 & \text{if } i \geq 1 \end{cases}$$

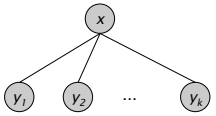


Def. Let F_k be smallest possible tree of rank k satisfying property.

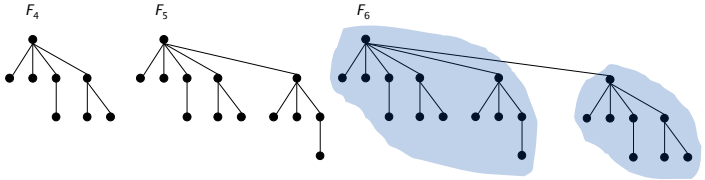


Lemma. Fix a point in time. Let x be a node, and let $y_1, \ldots, y_k$ denote its children in the order in which they were linked to x. Then:

$$rank(y_i) \geq \begin{cases} 0 & \text{if } i = 1 \\ i-2 & \text{if } i \geq 1 \end{cases}$$

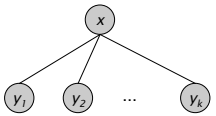


Def. Let F_k be smallest possible tree of rank k satisfying property.



Lemma. Fix a point in time. Let x be a node, and let $y_1, \ldots, y_k$ denote its children in the order in which they were linked to x. Then:

$$rank(y_i) \geq \begin{cases} 0 & \text{if } i = 1 \\ i-2 & \text{if } i \geq 1 \end{cases}$$



Def. Let F_k be smallest possible tree of rank k satisfying property.

Fibonacci fact. $F_k \geq \phi^k$, where $\phi = (1 + \sqrt{5}) / 2 \approx 1.618$.

Corollary. $rank(H) \leq \log_\phi n$.

golden ratio

Fibonacci Numbers

Fibonacci Numbers: Exponential Growth

Def. The Fibonacci sequence is: 1, 2, 3, 5, 8, 13, 21, ...

$$F_k = \begin{cases} 1 & \text{if } k = 0 \\ 2 & \text{if } k = 1 \\ F_{k-1} + F_{k-2} & \text{if } k \geq 2 \end{cases}$$

slightly non-standard definition

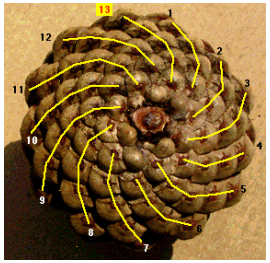
Lemma. $F_k \geq \phi^k$, where $\phi = (1 + \sqrt{5}) / 2 \approx 1.618$.

Pf. [by induction on k]

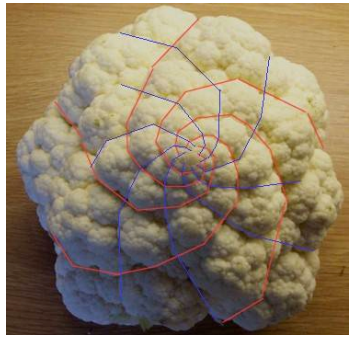
- Base cases: $F_0 = 1 \geq 1$, $F_1 = 2 \geq \phi$.
- Inductive hypotheses: $F_k \geq \phi^k$ and $F_{k+1} \geq \phi^{k+1}$

$$\begin{aligned} F_{k+2} &= F_k + F_{k+1} && \text{(definition)} \\ &\geq \phi^k + \phi^{k+1} && \text{(inductive hypothesis)} \\ &= \phi^k (1 + \phi) && \text{(algebra)} \\ &= \phi^k (\phi^2) && (\phi^2 = \phi + 1) \\ &= \phi^{k+2} && \text{(algebra)} \end{aligned}$$

Fibonacci Numbers and Nature



pinecone



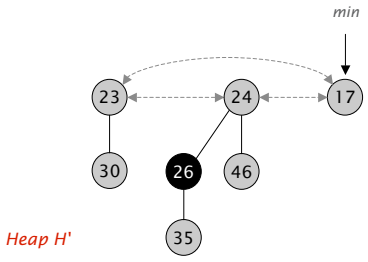
cauliflower

Union

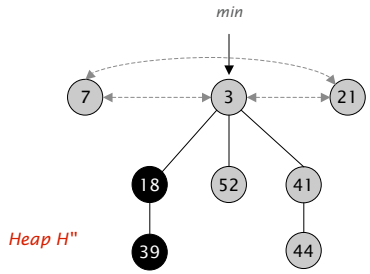
Fibonacci Heaps: Union

Union. Combine two Fibonacci heaps.

Representation. Root lists are circular, doubly linked lists.



Heap H'

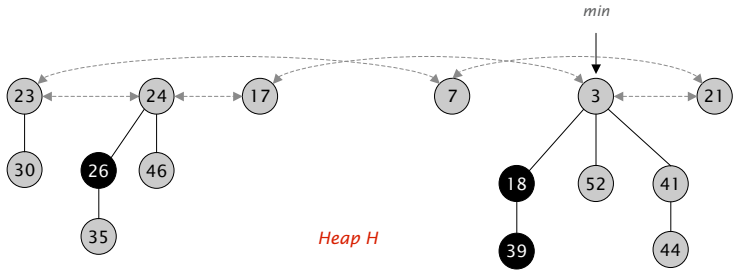


Heap H''

Fibonacci Heaps: Union

Union. Combine two Fibonacci heaps.

Representation. Root lists are circular, doubly linked lists.



Fibonacci Heaps: Union

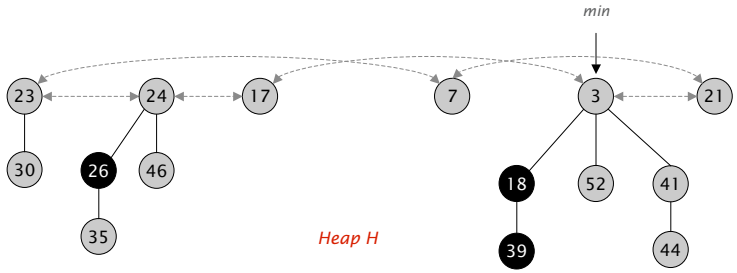
Actual cost. O(1)

Change in potential. 0

Amortized cost. O(1)

$\Phi(H) = trees(H) + 2 \cdot marks(H)$

potential function



Delete

Fibonacci Heaps: Delete

Delete node *x*.

- *decrease-key* of *x* to -∞.
- *delete-min* element in heap.

$\Phi(H) = trees(H) + 2 \cdot marks(H)$

potential function

Amortized cost. O(*rank*(*H*))

- O(1) amortized for *decrease-key*.
- O(*rank*(*H*)) amortized for *delete-min*.

19 Binomial Heaps

This chapter and Chapter 20 present data structures known as ***mergeable heaps***, which support the following five operations.

MAKE-HEAP() creates and returns a new heap containing no elements.

INSERT(H, x) inserts node x, whose *key* field has already been filled in, into heap H.

MINIMUM(H) returns a pointer to the node in heap H whose key is minimum.

EXTRACT-MIN(H) deletes the node from heap H whose key is minimum, returning a pointer to the node.

UNION(H_1, H_2) creates and returns a new heap that contains all the nodes of heaps H_1 and H_2. Heaps H_1 and H_2 are "destroyed" by this operation.

In addition, the data structures in these chapters also support the following two operations.

DECREASE-KEY(H, x, k) assigns to node x within heap H the new key value k, which is assumed to be no greater than its current key value.[1]

DELETE(H, x) deletes node x from heap H.

As the table in Figure 19.1 shows, if we don't need the UNION operation, ordinary binary heaps, as used in heapsort (Chapter 6), work well. Operations other than UNION run in worst-case time $O(\lg n)$ (or better) on a binary heap. If the UNION operation must be supported, however, binary heaps perform poorly. By concatenating the two arrays that hold the binary heaps to be merged and then running MIN-HEAPIFY (see Exercise 6.2-2), the UNION operation takes $\Theta(n)$ time in the worst case.

---

[1] As mentioned in the introduction to Part V, our default mergeable heaps are mergeable min-heaps, and so the operations MINIMUM, EXTRACT-MIN, and DECREASE-KEY apply. Alternatively, we could define a ***mergeable max-heap*** with the operations MAXIMUM, EXTRACT-MAX, and INCREASE-KEY.

Procedure	Binary heap (worst-case)	Binomial heap (worst-case)	Fibonacci heap (amortized)
MAKE-HEAP	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
INSERT	$\Theta(\lg n)$	$O(\lg n)$	$\Theta(1)$
MINIMUM	$\Theta(1)$	$O(\lg n)$	$\Theta(1)$
EXTRACT-MIN	$\Theta(\lg n)$	$\Theta(\lg n)$	$O(\lg n)$
UNION	$\Theta(n)$	$O(\lg n)$	$\Theta(1)$
DECREASE-KEY	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(1)$
DELETE	$\Theta(\lg n)$	$\Theta(\lg n)$	$O(\lg n)$

Figure 19.1 Running times for operations on three implementations of mergeable heaps. The number of items in the heap(s) at the time of an operation is denoted by n.

In this chapter, we examine "binomial heaps," whose worst-case time bounds are also shown in Figure 19.1. In particular, the UNION operation takes only $O(\lg n)$ time to merge two binomial heaps with a total of n elements.

In Chapter 20, we shall explore Fibonacci heaps, which have even better time bounds for some operations. Note, however, that the running times for Fibonacci heaps in Figure 19.1 are amortized time bounds, not worst-case per-operation time bounds.

This chapter ignores issues of allocating nodes prior to insertion and freeing nodes following deletion. We assume that the code that calls the heap procedures deals with these details.

Binary heaps, binomial heaps, and Fibonacci heaps are all inefficient in their support of the operation SEARCH; it can take a while to find a node with a given key. For this reason, operations such as DECREASE-KEY and DELETE that refer to a given node require a pointer to that node as part of their input. As in our discussion of priority queues in Section 6.5, when we use a mergeable heap in an application, we often store a handle to the corresponding application object in each mergeable-heap element, as well as a handle to corresponding mergeable-heap element in each application object. The exact nature of these handles depends on the application and its implementation.

Section 19.1 defines binomial heaps after first defining their constituent binomial trees. It also introduces a particular representation of binomial heaps. Section 19.2 shows how we can implement operations on binomial heaps in the time bounds given in Figure 19.1.

19.1 Binomial trees and binomial heaps

A binomial heap is a collection of binomial trees, so this section starts by defining binomial trees and proving some key properties. We then define binomial heaps and show how they can be represented.

19.1.1 Binomial trees

The ***binomial tree*** B_k is an ordered tree (see Section B.5.2) defined recursively. As shown in Figure 19.2(a), the binomial tree B_0 consists of a single node. The binomial tree B_k consists of two binomial trees B_{k-1} that are ***linked*** together: the root of one is the leftmost child of the root of the other. Figure 19.2(b) shows the binomial trees B_0 through B_4.

Some properties of binomial trees are given by the following lemma.

Lemma 19.1 (Properties of binomial trees)
For the binomial tree B_k,

1. there are 2^k nodes,
2. the height of the tree is k,
3. there are exactly $\binom{k}{i}$ nodes at depth i for $i = 0, 1, \ldots, k$, and
4. the root has degree k, which is greater than that of any other node; moreover if the children of the root are numbered from left to right by $k-1, k-2, \ldots, 0$, child i is the root of a subtree B_i.

Proof The proof is by induction on k. For each property, the basis is the binomial tree B_0. Verifying that each property holds for B_0 is trivial.

For the inductive step, we assume that the lemma holds for B_{k-1}.

1. Binomial tree B_k consists of two copies of B_{k-1}, and so B_k has $2^{k-1}+2^{k-1} = 2^k$ nodes.
2. Because of the way in which the two copies of B_{k-1} are linked to form B_k, the maximum depth of a node in B_k is one greater than the maximum depth in B_{k-1}. By the inductive hypothesis, this maximum depth is $(k-1)+1 = k$.
3. Let $D(k, i)$ be the number of nodes at depth i of binomial tree B_k. Since B_k is composed of two copies of B_{k-1} linked together, a node at depth i in B_{k-1} appears in B_k once at depth i and once at depth $i+1$. In other words, the number of nodes at depth i in B_k is the number of nodes at depth i in B_{k-1} plus

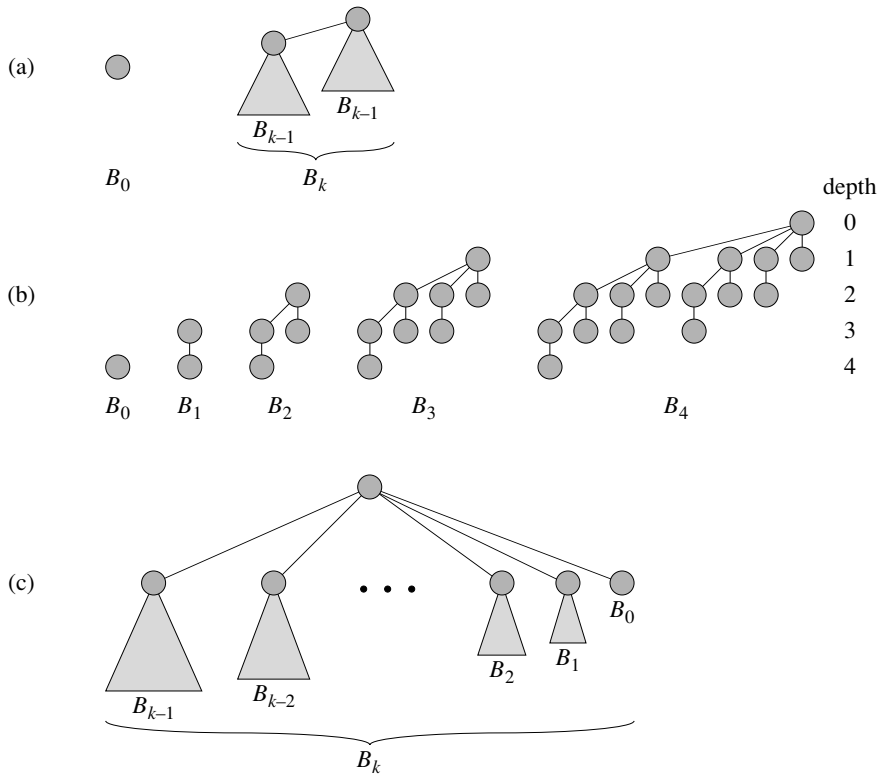


Figure 19.2 **(a)** The recursive definition of the binomial tree B_k. Triangles represent rooted subtrees. **(b)** The binomial trees B_0 through B_4. Node depths in B_4 are shown. **(c)** Another way of looking at the binomial tree B_k.

the number of nodes at depth $i - 1$ in B_{k-1}. Thus,

$$\begin{aligned} D(k,i) &= D(k-1,i) + D(k-1,i-1) && \text{(by the inductive hypothesis)} \\ &= \binom{k-1}{i} + \binom{k-1}{i-1} && \text{(by Exercise C.1-7)} \\ &= \binom{k}{i} . \end{aligned}$$

4. The only node with greater degree in B_k than in B_{k-1} is the root, which has one more child than in B_{k-1}. Since the root of B_{k-1} has degree $k - 1$, the root of B_k has degree k. Now, by the inductive hypothesis, and as Figure 19.2(c) shows, from left to right, the children of the root of B_{k-1} are roots of $B_{k-2}, B_{k-3}, \ldots, B_0$. When B_{k-1} is linked to B_{k-1}, therefore, the children of the resulting root are roots of $B_{k-1}, B_{k-2}, \ldots, B_0$. ■

Corollary 19.2
The maximum degree of any node in an n-node binomial tree is $\lg n$.

Proof Immediate from properties 1 and 4 of Lemma 19.1. ■

The term "binomial tree" comes from property 3 of Lemma 19.1, since the terms $\binom{k}{i}$ are the binomial coefficients. Exercise 19.1-3 gives further justification for the term.

19.1.2 Binomial heaps

A ***binomial heap*** H is a set of binomial trees that satisfies the following ***binomial-heap properties***.

1. Each binomial tree in H obeys the ***min-heap property***: the key of a node is greater than or equal to the key of its parent. We say that each such tree is ***min-heap-ordered***.
2. For any nonnegative integer k, there is at most one binomial tree in H whose root has degree k.

The first property tells us that the root of a min-heap-ordered tree contains the smallest key in the tree.

The second property implies that an n-node binomial heap H consists of at most $\lfloor \lg n \rfloor + 1$ binomial trees. To see why, observe that the binary representation of n has $\lfloor \lg n \rfloor + 1$ bits, say $\langle b_{\lfloor \lg n \rfloor}, b_{\lfloor \lg n \rfloor - 1}, \ldots, b_0 \rangle$, so that $n = \sum_{i=0}^{\lfloor \lg n \rfloor} b_i 2^i$. By property 1 of Lemma 19.1, therefore, binomial tree B_i appears in H if and only if bit $b_i = 1$. Thus, binomial heap H contains at most $\lfloor \lg n \rfloor + 1$ binomial trees.

Figure 19.3(a) shows a binomial heap H with 13 nodes. The binary representation of 13 is $\langle 1101 \rangle$, and H consists of min-heap-ordered binomial trees B_3, B_2, and B_0, having 8, 4, and 1 nodes respectively, for a total of 13 nodes.

Representing binomial heaps

As shown in Figure 19.3(b), each binomial tree within a binomial heap is stored in the left-child, right-sibling representation of Section 10.4. Each node has a *key* field and any other satellite information required by the application. In addition, each node x contains pointers $p[x]$ to its parent, $child[x]$ to its leftmost child, and $sibling[x]$ to the sibling of x immediately to its right. If node x is a root, then $p[x] = \text{NIL}$. If node x has no children, then $child[x] = \text{NIL}$, and if x is the rightmost child of its parent, then $sibling[x] = \text{NIL}$. Each node x also contains the field $degree[x]$, which is the number of children of x.

As Figure 19.3 also shows, the roots of the binomial trees within a binomial heap are organized in a linked list, which we refer to as the ***root list***. The degrees

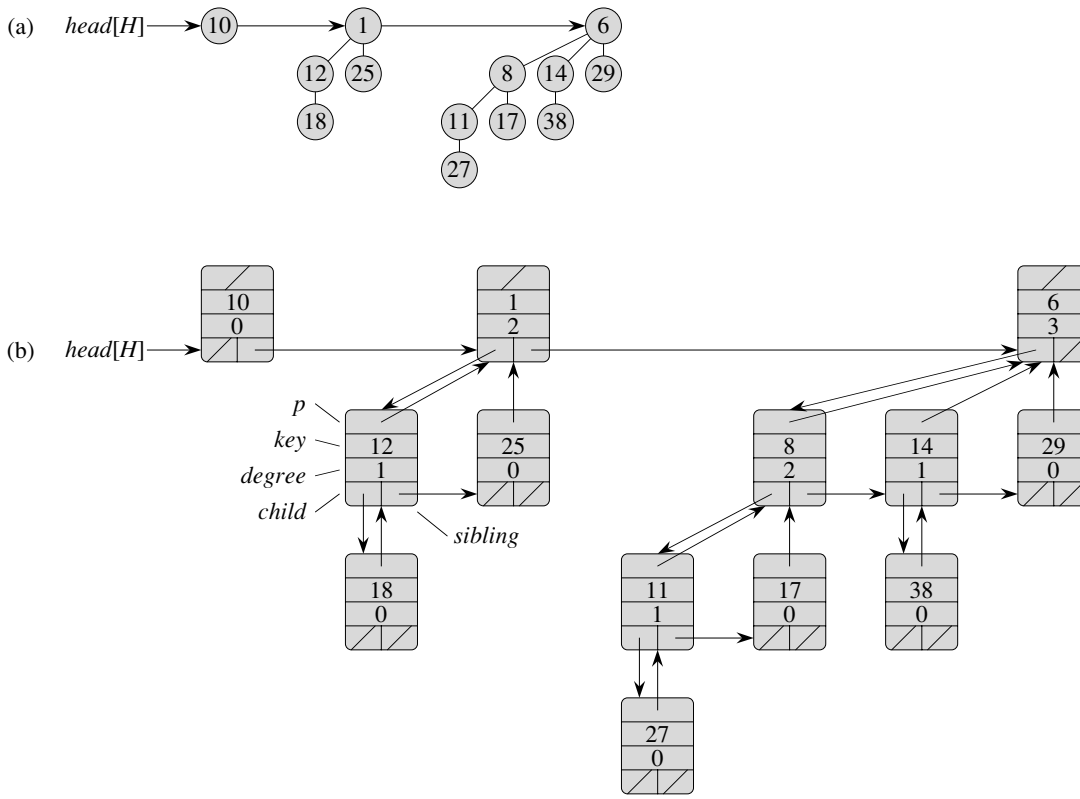


Figure 19.3 A binomial heap H with $n = 13$ nodes. **(a)** The heap consists of binomial trees B_0, B_2, and B_3, which have 1, 4, and 8 nodes respectively, totaling $n = 13$ nodes. Since each binomial tree is min-heap-ordered, the key of any node is no less than the key of its parent. Also shown is the root list, which is a linked list of roots in order of increasing degree. **(b)** A more detailed representation of binomial heap H. Each binomial tree is stored in the left-child, right-sibling representation, and each node stores its degree.

of the roots strictly increase as we traverse the root list. By the second binomial-heap property, in an n-node binomial heap the degrees of the roots are a subset of $\{0, 1, \ldots, \lfloor \lg n \rfloor\}$. The *sibling* field has a different meaning for roots than for nonroots. If x is a root, then $sibling[x]$ points to the next root in the root list. (As usual, $sibling[x] = \text{NIL}$ if x is the last root in the root list.)

A given binomial heap H is accessed by the field $head[H]$, which is simply a pointer to the first root in the root list of H. If binomial heap H has no elements, then $head[H] = \text{NIL}$.

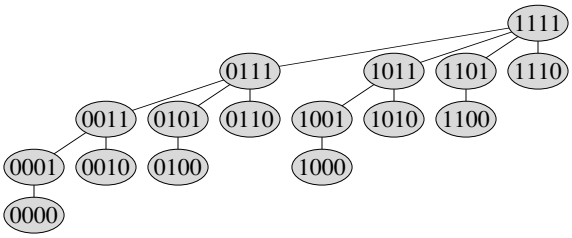


Figure 19.4 The binomial tree B_4 with nodes labeled in binary by a postorder walk.

Exercises

19.1-1
Suppose that x is a node in a binomial tree within a binomial heap, and assume that $sibling[x] \neq \text{NIL}$. If x is not a root, how does $degree[sibling[x]]$ compare to $degree[x]$? How about if x is a root?

19.1-2
If x is a nonroot node in a binomial tree within a binomial heap, how does $degree[x]$ compare to $degree[p[x]]$?

19.1-3
Suppose we label the nodes of binomial tree B_k in binary by a postorder walk, as in Figure 19.4. Consider a node x labeled l at depth i, and let $j = k - i$. Show that x has j 1's in its binary representation. How many binary k-strings are there that contain exactly j 1's? Show that the degree of x is equal to the number of 1's to the right of the rightmost 0 in the binary representation of l.

19.2 Operations on binomial heaps

In this section, we show how to perform operations on binomial heaps in the time bounds shown in Figure 19.1. We shall only show the upper bounds; the lower bounds are left as Exercise 19.2-10.

Creating a new binomial heap

To make an empty binomial heap, the MAKE-BINOMIAL-HEAP procedure simply allocates and returns an object H, where $head[H] = \text{NIL}$. The running time is $\Theta(1)$.

Finding the minimum key

The procedure BINOMIAL-HEAP-MINIMUM returns a pointer to the node with the minimum key in an n-node binomial heap H. This implementation assumes that there are no keys with value ∞. (See Exercise 19.2-5.)

```
BINOMIAL-HEAP-MINIMUM(H)
1  y ← NIL
2  x ← head[H]
3  min ← ∞
4  while x ≠ NIL
5        do if key[x] < min
6              then min ← key[x]
7                   y ← x
8           x ← sibling[x]
9  return y
```

Since a binomial heap is min-heap-ordered, the minimum key must reside in a root node. The BINOMIAL-HEAP-MINIMUM procedure checks all roots, which number at most $\lfloor \lg n \rfloor + 1$, saving the current minimum in *min* and a pointer to the current minimum in y. When called on the binomial heap of Figure 19.3, BINOMIAL-HEAP-MINIMUM returns a pointer to the node with key 1.

Because there are at most $\lfloor \lg n \rfloor + 1$ roots to check, the running time of BINOMIAL-HEAP-MINIMUM is $O(\lg n)$.

Uniting two binomial heaps

The operation of uniting two binomial heaps is used as a subroutine by most of the remaining operations. The BINOMIAL-HEAP-UNION procedure repeatedly links binomial trees whose roots have the same degree. The following procedure links the B_{k-1} tree rooted at node y to the B_{k-1} tree rooted at node z; that is, it makes z the parent of y. Node z thus becomes the root of a B_k tree.

```
BINOMIAL-LINK(y, z)
1  p[y] ← z
2  sibling[y] ← child[z]
3  child[z] ← y
4  degree[z] ← degree[z] + 1
```

The BINOMIAL-LINK procedure makes node y the new head of the linked list of node z's children in $O(1)$ time. It works because the left-child, right-sibling representation of each binomial tree matches the ordering property of the tree: in a B_k tree, the leftmost child of the root is the root of a B_{k-1} tree.

The following procedure unites binomial heaps H_1 and H_2, returning the resulting heap. It destroys the representations of H_1 and H_2 in the process. Besides BINOMIAL-LINK, the procedure uses an auxiliary procedure BINOMIAL-HEAP-MERGE that merges the root lists of H_1 and H_2 into a single linked list that is sorted by degree into monotonically increasing order. The BINOMIAL-HEAP-MERGE procedure, whose pseudocode we leave as Exercise 19.2-1, is similar to the MERGE procedure in Section 2.3.1.

```
BINOMIAL-HEAP-UNION(H_1, H_2)
 1  H ← MAKE-BINOMIAL-HEAP()
 2  head[H] ← BINOMIAL-HEAP-MERGE(H_1, H_2)
 3  free the objects H_1 and H_2 but not the lists they point to
 4  if head[H] = NIL
 5     then return H
 6  prev-x ← NIL
 7  x ← head[H]
 8  next-x ← sibling[x]
 9  while next-x ≠ NIL
10        do if (degree[x] ≠ degree[next-x]) or
                 (sibling[next-x] ≠ NIL and degree[sibling[next-x]] = degree[x])
11              then prev-x ← x                                  ▷ Cases 1 and 2
12                   x ← next-x                                  ▷ Cases 1 and 2
13              else if key[x] ≤ key[next-x]
14                      then sibling[x] ← sibling[next-x]        ▷ Case 3
15                           BINOMIAL-LINK(next-x, x)            ▷ Case 3
16                      else if prev-x = NIL                     ▷ Case 4
17                              then head[H] ← next-x            ▷ Case 4
18                              else sibling[prev-x] ← next-x    ▷ Case 4
19                           BINOMIAL-LINK(x, next-x)            ▷ Case 4
20                           x ← next-x                          ▷ Case 4
21           next-x ← sibling[x]
22  return H
```

Figure 19.5 shows an example of BINOMIAL-HEAP-UNION in which all four cases given in the pseudocode occur.

The BINOMIAL-HEAP-UNION procedure has two phases. The first phase, performed by the call of BINOMIAL-HEAP-MERGE, merges the root lists of binomial heaps H_1 and H_2 into a single linked list H that is sorted by degree into monotonically increasing order. There might be as many as two roots (but no more) of each degree, however, so the second phase links roots of equal degree until at most one root remains of each degree. Because the linked list H is sorted by degree, we can perform all the link operations quickly.

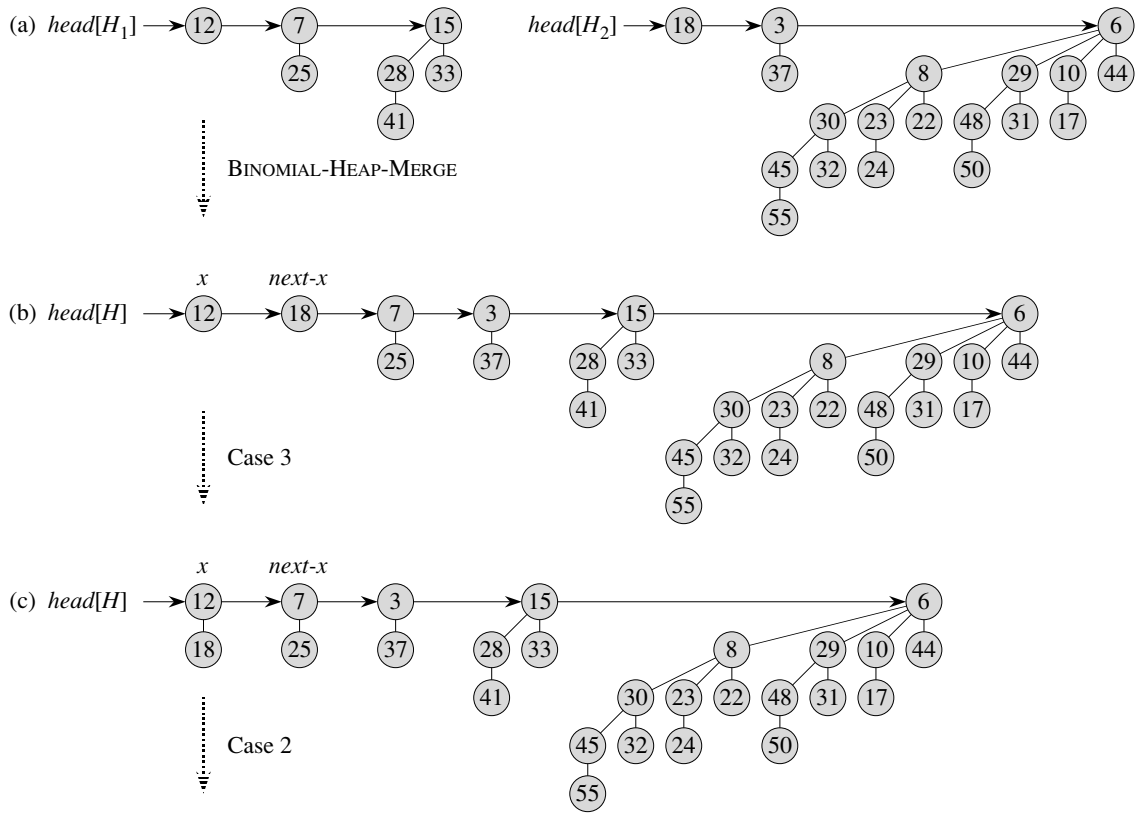
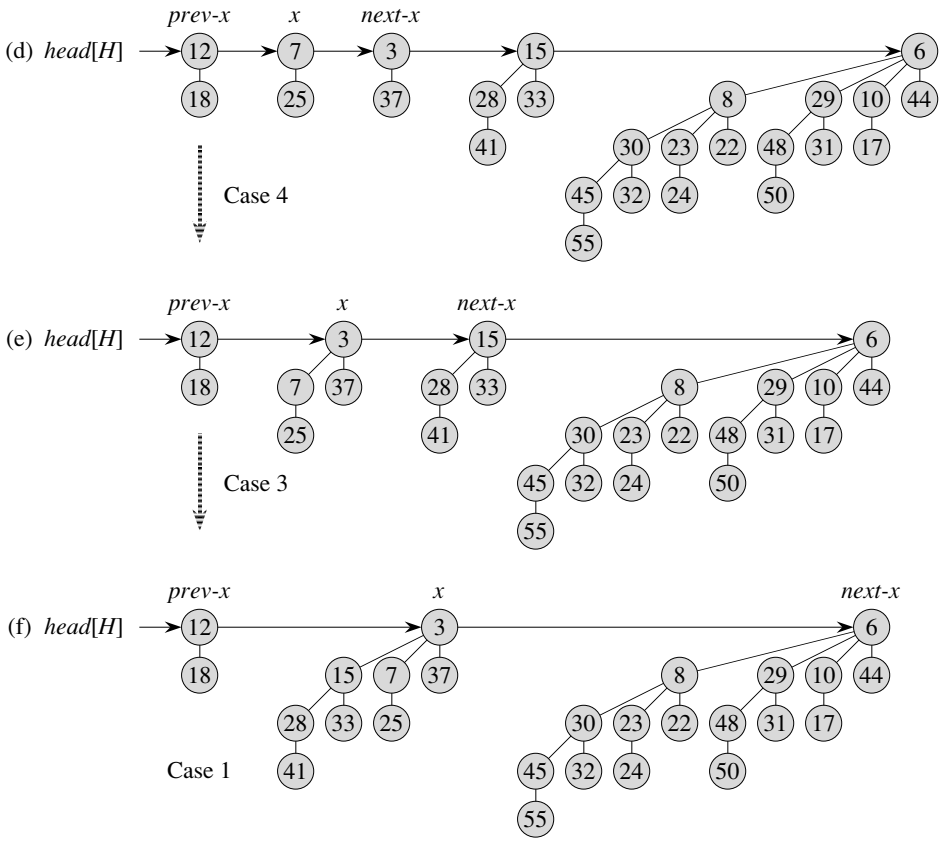


Figure 19.5 The execution of BINOMIAL-HEAP-UNION. **(a)** Binomial heaps H_1 and H_2. **(b)** Binomial heap H is the output of BINOMIAL-HEAP-MERGE(H_1, H_2). Initially, x is the first root on the root list of H. Because both x and *next-x* have degree 0 and $key[x] < key[next\text{-}x]$, case 3 applies. **(c)** After the link occurs, x is the first of three roots with the same degree, so case 2 applies. **(d)** After all the pointers move down one position in the root list, case 4 applies, since x is the first of two roots of equal degree. **(e)** After the link occurs, case 3 applies. **(f)** After another link, case 1 applies, because x has degree 3 and *next-x* has degree 4. This iteration of the **while** loop is the last, because after the pointers move down one position in the root list, *next-x* = NIL.

In detail, the procedure works as follows. Lines 1–3 start by merging the root lists of binomial heaps H_1 and H_2 into a single root list H. The root lists of H_1 and H_2 are sorted by strictly increasing degree, and BINOMIAL-HEAP-MERGE returns a root list H that is sorted by monotonically increasing degree. If the root lists of H_1 and H_2 have m roots altogether, BINOMIAL-HEAP-MERGE runs in $O(m)$ time by repeatedly examining the roots at the heads of the two root lists and appending the root with the lower degree to the output root list, removing it from its input root list in the process.



The BINOMIAL-HEAP-UNION procedure next initializes some pointers into the root list of H. First, it simply returns in lines 4–5 if it happens to be uniting two empty binomial heaps. From line 6 on, therefore, we know that H has at least one root. Throughout the procedure, we maintain three pointers into the root list:

- x points to the root currently being examined,
- *prev-x* points to the root preceding x on the root list: $sibling[prev\text{-}x] = x$ (since initially x has no predecessor, we start with *prev-x* set to NIL), and
- *next-x* points to the root following x on the root list: $sibling[x] = next\text{-}x$.

Initially, there are at most two roots on the root list H of a given degree: because H_1 and H_2 were binomial heaps, they each had at most one root of a given degree. Moreover, BINOMIAL-HEAP-MERGE guarantees us that if two roots in H have the same degree, they are adjacent in the root list.

In fact, during the execution of BINOMIAL-HEAP-UNION, there may be three roots of a given degree appearing on the root list H at some time. We shall see

in a moment how this situation could occur. At each iteration of the **while** loop of lines 9–21, therefore, we decide whether to link x and *next-x* based on their degrees and possibly the degree of $sibling[next\text{-}x]$. An invariant of the loop is that each time we start the body of the loop, both x and *next-x* are non-NIL. (See Exercise 19.2-4 for a precise loop invariant.)

Case 1, shown in Figure 19.6(a), occurs when $degree[x] \neq degree[next\text{-}x]$, that is, when x is the root of a B_k-tree and *next-x* is the root of a B_l-tree for some $l > k$. Lines 11–12 handle this case. We don't link x and *next-x*, so we simply march the pointers one position farther down the list. Updating *next-x* to point to the node following the new node x is handled in line 21, which is common to every case.

Case 2, shown in Figure 19.6(b), occurs when x is the first of three roots of equal degree, that is, when

$$degree[x] = degree[next\text{-}x] = degree[sibling[next\text{-}x]] .$$

We handle this case in the same manner as case 1: we just march the pointers one position farther down the list. The next iteration will execute either case 3 or case 4 to combine the second and third of the three equal-degree roots. Line 10 tests for both cases 1 and 2, and lines 11–12 handle both cases.

Cases 3 and 4 occur when x is the first of two roots of equal degree, that is, when

$$degree[x] = degree[next\text{-}x] \neq degree[sibling[next\text{-}x]] .$$

These cases may occur in any iteration, but one of them always occurs immediately following case 2. In cases 3 and 4, we link x and *next-x*. The two cases are distinguished by whether x or *next-x* has the smaller key, which determines the node that will be the root after the two are linked.

In case 3, shown in Figure 19.6(c), $key[x] \leq key[next\text{-}x]$, so *next-x* is linked to x. Line 14 removes *next-x* from the root list, and line 15 makes *next-x* the leftmost child of x.

In case 4, shown in Figure 19.6(d), *next-x* has the smaller key, so x is linked to *next-x*. Lines 16–18 remove x from the root list; there are two cases depending on whether x is the first root on the list (line 17) or is not (line 18). Line 19 then makes x the leftmost child of *next-x*, and line 20 updates x for the next iteration.

Following either case 3 or case 4, the setup for the next iteration of the **while** loop is the same. We have just linked two B_k-trees to form a B_{k+1}-tree, which x now points to. There were already zero, one, or two other B_{k+1}-trees on the root list resulting from BINOMIAL-HEAP-MERGE, so x is now the first of either one, two, or three B_{k+1}-trees on the root list. If x is the only one, then we enter case 1 in the next iteration: $degree[x] \neq degree[next\text{-}x]$. If x is the first of two, then we enter either case 3 or case 4 in the next iteration. It is when x is the first of three that we enter case 2 in the next iteration.

The running time of BINOMIAL-HEAP-UNION is $O(\lg n)$, where n is the total number of nodes in binomial heaps H_1 and H_2. We can see this as follows. Let H_1

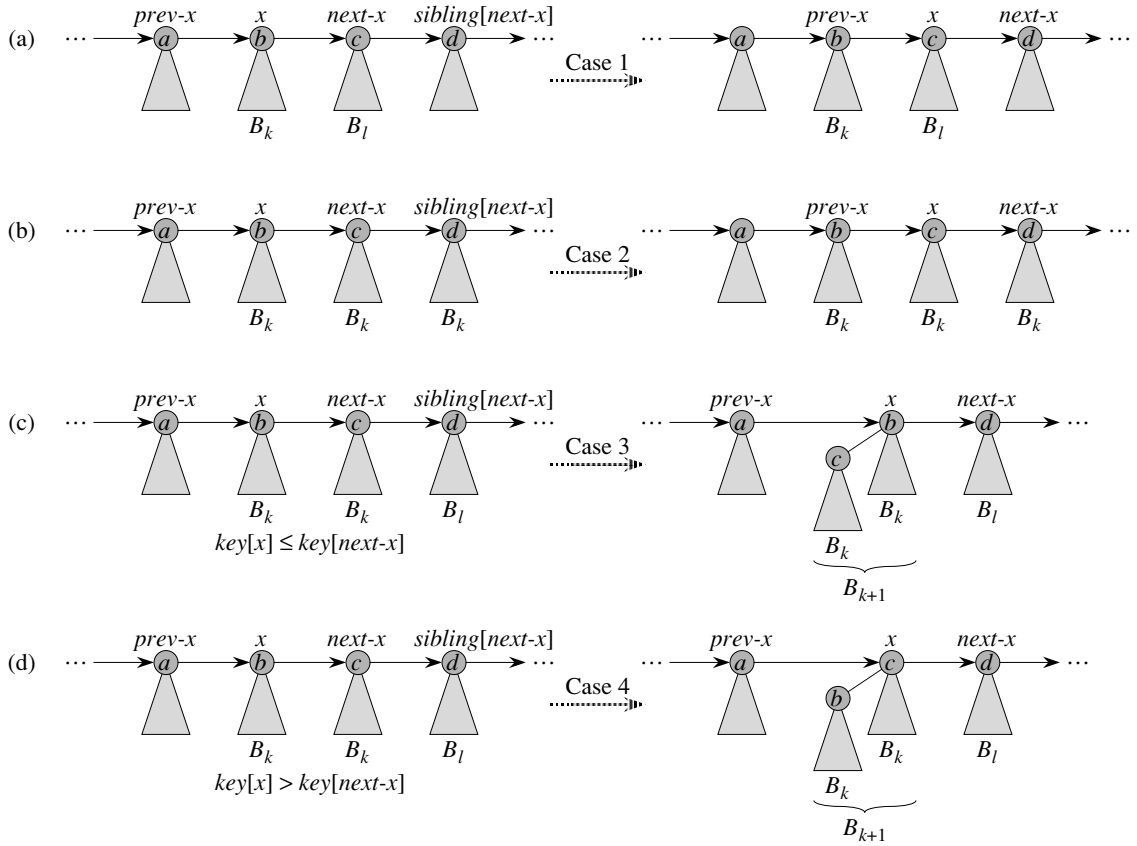


Figure 19.6 The four cases that occur in BINOMIAL-HEAP-UNION. Labels a, b, c, and d serve only to identify the roots involved; they do not indicate the degrees or keys of these roots. In each case, x is the root of a B_k-tree and $l > k$. **(a)** Case 1: $degree[x] \neq degree[next\text{-}x]$. The pointers move one position farther down the root list. **(b)** Case 2: $degree[x] = degree[next\text{-}x] = degree[sibling[next\text{-}x]]$. Again, the pointers move one position farther down the list, and the next iteration executes either case 3 or case 4. **(c)** Case 3: $degree[x] = degree[next\text{-}x] \neq degree[sibling[next\text{-}x]]$ and $key[x] \leq key[next\text{-}x]$. We remove *next-x* from the root list and link it to x, creating a B_{k+1}-tree. **(d)** Case 4: $degree[x] = degree[next\text{-}x] \neq degree[sibling[next\text{-}x]]$ and $key[next\text{-}x] \leq key[x]$. We remove x from the root list and link it to *next-x*, again creating a B_{k+1}-tree.

contain n_1 nodes and H_2 contain n_2 nodes, so that $n = n_1 + n_2$. Then H_1 contains at most $\lfloor \lg n_1 \rfloor + 1$ roots and H_2 contains at most $\lfloor \lg n_2 \rfloor + 1$ roots, and so H contains at most $\lfloor \lg n_1 \rfloor + \lfloor \lg n_2 \rfloor + 2 \leq 2 \lfloor \lg n \rfloor + 2 = O(\lg n)$ roots immediately after the call of BINOMIAL-HEAP-MERGE. The time to perform BINOMIAL-HEAP-MERGE is thus $O(\lg n)$. Each iteration of the **while** loop takes $O(1)$ time, and there are at most $\lfloor \lg n_1 \rfloor + \lfloor \lg n_2 \rfloor + 2$ iterations because each iteration either advances the

pointers one position down the root list of H or removes a root from the root list. The total time is thus $O(\lg n)$.

Inserting a node

The following procedure inserts node x into binomial heap H, assuming that x has already been allocated and $key[x]$ has already been filled in.

```
BINOMIAL-HEAP-INSERT(H, x)
1  H' ← MAKE-BINOMIAL-HEAP()
2  p[x] ← NIL
3  child[x] ← NIL
4  sibling[x] ← NIL
5  degree[x] ← 0
6  head[H'] ← x
7  H ← BINOMIAL-HEAP-UNION(H, H')
```

The procedure simply makes a one-node binomial heap H' in $O(1)$ time and unites it with the n-node binomial heap H in $O(\lg n)$ time. The call to BINOMIAL-HEAP-UNION takes care of freeing the temporary binomial heap H'. (A direct implementation that does not call BINOMIAL-HEAP-UNION is given as Exercise 19.2-8.)

Extracting the node with minimum key

The following procedure extracts the node with the minimum key from binomial heap H and returns a pointer to the extracted node.

```
BINOMIAL-HEAP-EXTRACT-MIN(H)
1  find the root x with the minimum key in the root list of H,
       and remove x from the root list of H
2  H' ← MAKE-BINOMIAL-HEAP()
3  reverse the order of the linked list of x's children,
       and set head[H'] to point to the head of the resulting list
4  H ← BINOMIAL-HEAP-UNION(H, H')
5  return x
```

This procedure works as shown in Figure 19.7. The input binomial heap H is shown in Figure 19.7(a). Figure 19.7(b) shows the situation after line 1: the root x with the minimum key has been removed from the root list of H. If x is the root of a B_k-tree, then by property 4 of Lemma 19.1, x's children, from left to right, are roots of B_{k-1}-, B_{k-2}-, ..., B_0-trees. Figure 19.7(c) shows that by reversing the list of x's children in line 3, we have a binomial heap H' that contains every node

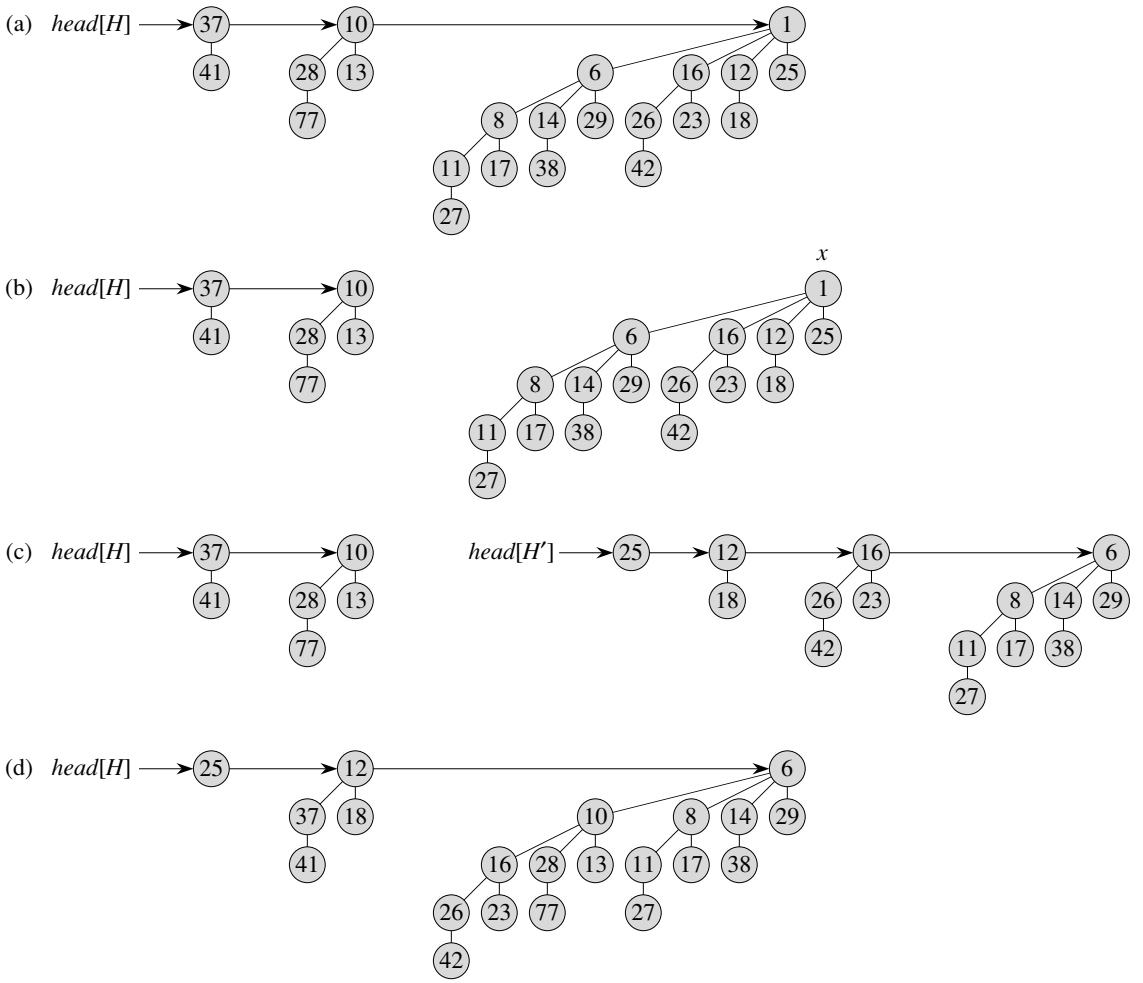


Figure 19.7 The action of BINOMIAL-HEAP-EXTRACT-MIN. **(a)** A binomial heap H. **(b)** The root x with minimum key is removed from the root list of H. **(c)** The linked list of x's children is reversed, giving another binomial heap H'. **(d)** The result of uniting H and H'.

in x's tree except for x itself. Because x's tree was removed from H in line 1, the binomial heap that results from uniting H and H' in line 4, shown in Figure 19.7(d), contains all the nodes originally in H except for x. Finally, line 5 returns x.

Since each of lines 1–4 takes $O(\lg n)$ time if H has n nodes, BINOMIAL-HEAP-EXTRACT-MIN runs in $O(\lg n)$ time.

Decreasing a key

The following procedure decreases the key of a node x in a binomial heap H to a new value k. It signals an error if k is greater than x's current key.

BINOMIAL-HEAP-DECREASE-KEY(H, x, k)

```
 1  if k > key[x]
 2     then error "new key is greater than current key"
 3  key[x] ← k
 4  y ← x
 5  z ← p[y]
 6  while z ≠ NIL and key[y] < key[z]
 7        do exchange key[y] ↔ key[z]
 8           ▷ If y and z have satellite fields, exchange them, too.
 9           y ← z
10           z ← p[y]
```

As shown in Figure 19.8, this procedure decreases a key in the same manner as in a binary min-heap: by "bubbling up" the key in the heap. After ensuring that the new key is in fact no greater than the current key and then assigning the new key to x, the procedure goes up the tree, with y initially pointing to node x. In each iteration of the **while** loop of lines 6–10, $key[y]$ is checked against the key of y's parent z. If y is the root or $key[y] \geq key[z]$, the binomial tree is now min-heap-ordered. Otherwise, node y violates min-heap ordering, and so its key is exchanged with the key of its parent z, along with any other satellite information. The procedure then sets y to z, going up one level in the tree, and continues with the next iteration.

The BINOMIAL-HEAP-DECREASE-KEY procedure takes $O(\lg n)$ time. By property 2 of Lemma 19.1, the maximum depth of x is $\lfloor \lg n \rfloor$, so the **while** loop of lines 6–10 iterates at most $\lfloor \lg n \rfloor$ times.

Deleting a key

It is easy to delete a node x's key and satellite information from binomial heap H in $O(\lg n)$ time. The following implementation assumes that no node currently in the binomial heap has a key of $-\infty$.

BINOMIAL-HEAP-DELETE(H, x)

```
1  BINOMIAL-HEAP-DECREASE-KEY(H, x, −∞)
2  BINOMIAL-HEAP-EXTRACT-MIN(H)
```

The BINOMIAL-HEAP-DELETE procedure makes node x have the unique minimum key in the entire binomial heap by giving it a key of $-\infty$. (Exercise 19.2-6

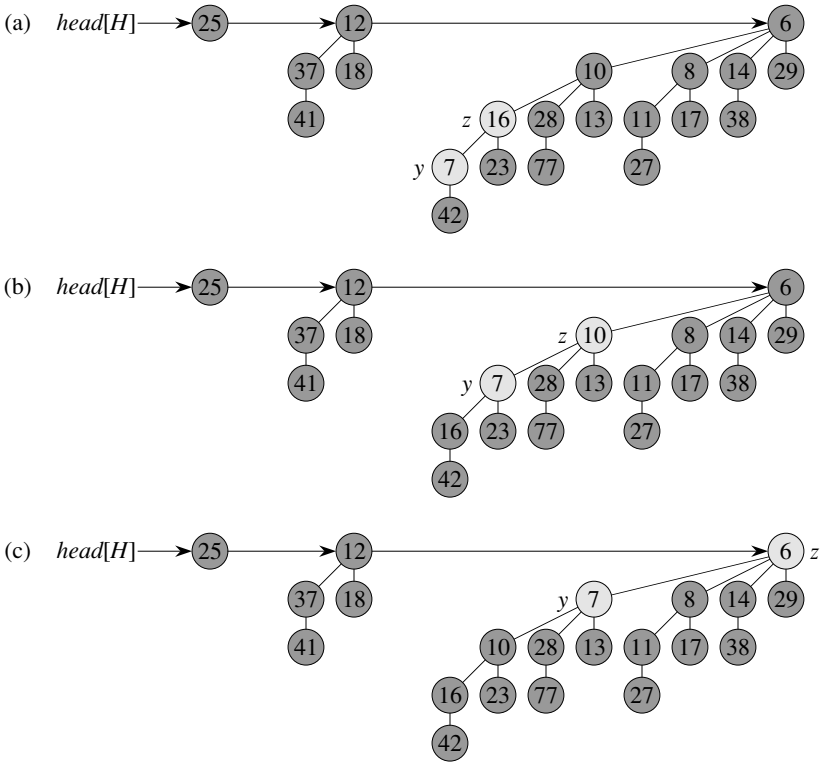


Figure 19.8 The action of BINOMIAL-HEAP-DECREASE-KEY. **(a)** The situation just before line 6 of the first iteration of the **while** loop. Node y has had its key decreased to 7, which is less than the key of y's parent z. **(b)** The keys of the two nodes are exchanged, and the situation just before line 6 of the second iteration is shown. Pointers y and z have moved up one level in the tree, but min-heap order is still violated. **(c)** After another exchange and moving pointers y and z up one more level, we find that min-heap order is satisfied, so the **while** loop terminates.

deals with the situation in which $-\infty$ cannot appear as a key, even temporarily.) It then bubbles this key and the associated satellite information up to a root by calling BINOMIAL-HEAP-DECREASE-KEY. This root is then removed from H by a call of BINOMIAL-HEAP-EXTRACT-MIN.

The BINOMIAL-HEAP-DELETE procedure takes $O(\lg n)$ time.

Exercises

19.2-1

Write pseudocode for BINOMIAL-HEAP-MERGE.

19.2-2
Show the binomial heap that results when a node with key 24 is inserted into the binomial heap shown in Figure 19.7(d).

19.2-3
Show the binomial heap that results when the node with key 28 is deleted from the binomial heap shown in Figure 19.8(c).

19.2-4
Argue the correctness of BINOMIAL-HEAP-UNION using the following loop invariant:

> At the start of each iteration of the **while** loop of lines 9–21, x points to a root that is one of the following:
>
> - the only root of its degree,
> - the first of the only two roots of its degree, or
> - the first or second of the only three roots of its degree.
>
> Moreover, all roots preceding x's predecessor on the root list have unique degrees on the root list, and if x's predecessor has a degree different from that of x, its degree on the root list is unique, too. Finally, node degrees monotonically increase as we traverse the root list.

19.2-5
Explain why the BINOMIAL-HEAP-MINIMUM procedure might not work correctly if keys can have the value ∞. Rewrite the pseudocode to make it work correctly in such cases.

19.2-6
Suppose there is no way to represent the key $-\infty$. Rewrite the BINOMIAL-HEAP-DELETE procedure to work correctly in this situation. It should still take $O(\lg n)$ time.

19.2-7
Discuss the relationship between inserting into a binomial heap and incrementing a binary number and the relationship between uniting two binomial heaps and adding two binary numbers.

19.2-8
In light of Exercise 19.2-7, rewrite BINOMIAL-HEAP-INSERT to insert a node directly into a binomial heap without calling BINOMIAL-HEAP-UNION.

19.2-9
Show that if root lists are kept in strictly decreasing order by degree (instead of strictly increasing order), each of the binomial heap operations can be implemented without changing its asymptotic running time.

19.2-10
Find inputs that cause BINOMIAL-HEAP-EXTRACT-MIN, BINOMIAL-HEAP-DECREASE-KEY, and BINOMIAL-HEAP-DELETE to run in $\Omega(\lg n)$ time. Explain why the worst-case running times of BINOMIAL-HEAP-INSERT, BINOMIAL-HEAP-MINIMUM, and BINOMIAL-HEAP-UNION are $\overset{\infty}{\Omega}(\lg n)$ but not $\Omega(\lg n)$. (See Problem 3-5.)

Problems

19-1 2-3-4 heaps
Chapter 18 introduced the 2-3-4 tree, in which every internal node (other than possibly the root) has two, three, or four children and all leaves have the same depth. In this problem, we shall implement ***2-3-4 heaps***, which support the mergeable-heap operations.

The 2-3-4 heaps differ from 2-3-4 trees in the following ways. In 2-3-4 heaps, only leaves store keys, and each leaf x stores exactly one key in the field $key[x]$. There is no particular ordering of the keys in the leaves; that is, from left to right, the keys may be in any order. Each internal node x contains a value $small[x]$ that is equal to the smallest key stored in any leaf in the subtree rooted at x. The root r contains a field $height[r]$ that is the height of the tree. Finally, 2-3-4 heaps are intended to be kept in main memory, so that disk reads and writes are not needed.

Implement the following 2-3-4 heap operations. Each of the operations in parts (a)–(e) should run in $O(\lg n)$ time on a 2-3-4 heap with n elements. The UNION operation in part (f) should run in $O(\lg n)$ time, where n is the number of elements in the two input heaps.

a. MINIMUM, which returns a pointer to the leaf with the smallest key.

b. DECREASE-KEY, which decreases the key of a given leaf x to a given value $k \leq key[x]$.

c. INSERT, which inserts leaf x with key k.

d. DELETE, which deletes a given leaf x.

e. EXTRACT-MIN, which extracts the leaf with the smallest key.

f. UNION, which unites two 2-3-4 heaps, returning a single 2-3-4 heap and destroying the input heaps.

19-2 Minimum-spanning-tree algorithm using binomial heaps

Chapter 23 presents two algorithms to solve the problem of finding a minimum spanning tree of an undirected graph. Here, we shall see how binomial heaps can be used to devise a different minimum-spanning-tree algorithm.

We are given a connected, undirected graph $G = (V, E)$ with a weight function $w : E \to \mathbf{R}$. We call $w(u, v)$ the weight of edge (u, v). We wish to find a minimum spanning tree for G: an acyclic subset $T \subseteq E$ that connects all the vertices in V and whose total weight

$$w(T) = \sum_{(u,v)\in T} w(u, v)$$

is minimized.

The following pseudocode, which can be proven correct using techniques from Section 23.1, constructs a minimum spanning tree T. It maintains a partition $\{V_i\}$ of the vertices of V and, with each set V_i, a set

$$E_i \subseteq \{(u, v) : u \in V_i \text{ or } v \in V_i\}$$

of edges incident on vertices in V_i.

```
MST(G)
 1  T ← ∅
 2  for each vertex v_i ∈ V[G]
 3       do V_i ← {v_i}
 4          E_i ← {(v_i, v) ∈ E[G]}
 5  while there is more than one set V_i
 6       do choose any set V_i
 7          extract the minimum-weight edge (u, v) from E_i
 8          assume without loss of generality that u ∈ V_i and v ∈ V_j
 9          if i ≠ j
10            then T ← T ∪ {(u, v)}
11                 V_i ← V_i ∪ V_j, destroying V_j
12                 E_i ← E_i ∪ E_j
```

Describe how to implement this algorithm using binomial heaps to manage the vertex and edge sets. Do you need to change the representation of a binomial heap? Do you need to add operations beyond the mergeable-heap operations given in Figure 19.1? Give the running time of your implementation.

Chapter notes

Binomial heaps were introduced in 1978 by Vuillemin [307]. Brown [49, 50] studied their properties in detail.

FIBONACCI HEAPS

12 Introduction

Priority queues are a classic topic in theoretical computer science. The search for a fast priority queue implementation is motivated primarily by two network optimization algorithms: Shortest Path (SP) and Minimum Spanning Tree (MST), i.e., the connector problem. As we shall see, Fibonacci Heaps provide a fast and elegant solution.
The following 3-step procedure shows that both Dijkstra's SP-algorithm or Prim's MST-algorithm can be implemented using a priority queue:

1. Maintain a priority queue on the vertices $V(G)$.
2. Put s in the queue, where s is the start vertex (Shortest Path) or any vertex (MST). Give s a *key* of 0. Add all other vertices and set their key to infinity.
3. Repeatedly delete the minimum-key vertex v from the queue and mark it *scanned*. For each neighbor w of v do: If w is not scanned (so far), decrease its key to the minimum of the value calculated below and w's current key:
   - SP: key(v) + length(vw),
   - MST: weight(vw).

The classical answer to the problem of maintaining a priority queue on the vertices is to use a *binary heap*, often just called a *heap*. Heaps are commonly used because they have good bounds on the time required for the following operations: insert $O(\log n)$, delete-min $O(\log n)$, and decrease-key $O(\log n)$, where n reflects the number of elements in the heap.
If a graph has n vertices and e edges, then running either Prim's or Dijkstra's algorithms will require $O(n \log n)$ time for inserts and deletes. However, in

the worst case, we will also perform e decrease-keys, because we may have to perform a key update every time we come across a new edge. This will take $O(e \log n)$ time. Since the graph is connected, $e \geq n$, and the overall time bound is given by $O(e \log n)$. As we shall see, Fibonacci heaps allow us to do much better.

13 Definition and Elementary Operations

The Fibonacci heap data structure invented by Fredman and Tarjan in 1984 gives a very efficient implementation of the priority queues. Since the goal is to find a way to minimize the number of operations needed to compute the MST or SP, the kind of operations that we are interested in are *insert*, *decrease-key*, *link*, and *delete-min* (we have not covered why *link* is a useful operation yet, but this will become clear later on). The method to achieve this minimization goal is laziness - *do work only when you must, and then use it to simplify the structure as much as possible so that your future work is easy.* This way, the user is forced to do many cheap operations in order to make the data structure complicated.
Fibonacci heaps make use of heap-ordered trees. A heap-ordered tree is one that maintains the heap property, that is, where $key(parent) \leq key(child)$ for all nodes in the tree.

DEFINITION 13.1: A Fibonacci heap H is a collection of heap-ordered trees that have the following properties:

1. The roots of these trees are kept in a doubly-linked list (the *root list* of H),

2. The root of each tree contains the minimum element in that tree (this follows from being a heap-ordered tree),

3. We access the heap by a pointer to the tree root with the overall minimum key,

4. For each node x, we keep track of the *degree* (also known as the order or *rank*) of x, which is just the number of children x has; we also keep track of the *mark* of x, which is a Boolean value whose role will be explained later.

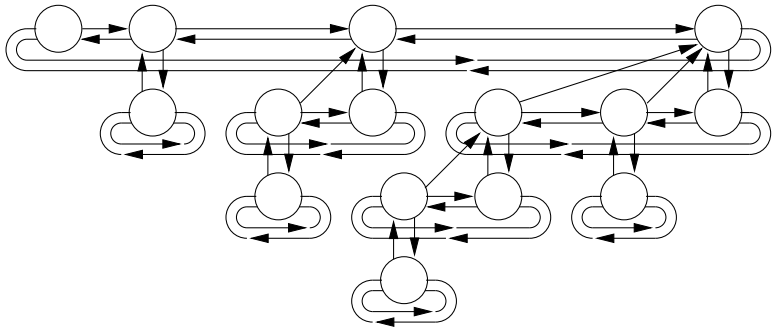


Fig. 9: A detailed view of a Fibonacci Heap. NULL pointers are omitted for clarity.

For each node, we have at most four pointers that respectively point to the node's parent, to one of its children, and to two of its siblings. The sibling pointers are arranged in a doubly-linked list (the *child list* of the parent node). We have not described how the operations on Fibonacci heaps are implemented, and their implementation will add some additional properties to H. The following are some elementary operations used in maintaining Fibonacci heaps:

Inserting a node x: We create a new tree containing only x and insert it into the root list of H; this is clearly an $O(1)$ operation.

Linking two trees x and y: Let x and y be the roots of the two trees we want to link; then if $key(x) \geq key(y)$, we make x the child of y; otherwise, we make y the child of x. We update the appropriate node's degrees and the appropriate child list; this takes $O(1)$ operations.

Cutting a node x: If x is a root in H, we are done. If x is not a root in H, we remove x from the child list of its parent, and insert it into the root list of H, updating the appropriate variables (the degree of the parent of x is decremented, etc.). Again, this takes $O(1)$ operations. We assume that when we want to cut/find a node, we have a pointer *hanging* around that accesses it directly, so actually finding the node takes $O(1)$ time.

```
CLEANUP:
newmin ← some root list node
for i ← 0 to ⌊log n⌋
    B[i] ← NULL
for all nodes v in the root list
   parent(v) ← NULL
   unmark v
   if key(newmin) > key(v)
      newmin ← v
   LINKDUPES(v)
```

```
LINKDUPES:
w ← B[deg(v)]
while w ≠ NULL
    B[deg(v)] ← NULL
    if key(w) ≤ key(v)
       swap v and w
    remove w from root list
    link w to v
    w ← B[deg(v)]
B[deg(v)] ← v
```

Fig. 10: The CLEANUP algorithm executed after performing a *delete-min*

Marking a node x: We say that x is marked if its mark is set to *true*, and that it is unmarked if its mark is set to *false*. A root is always unmarked. We mark x if it is not a root and it loses a child (i.e., one of its children is cut and put into the root-list). We unmark x whenever it becomes a root. We shall see later on that no marked node will lose a second child before it is cut itself.

13.1 The *delete-min* Operation

Deleting the minimum key node is a little more complicated. First, we remove the minimum key from the root list and splice its children into the root list. Except for updating the parent pointers, this takes $O(1)$ time. Then we scan through the root list to find the new smallest key and update the parent pointers of the new roots. This scan could take $O(n)$ time in the worst case. To bring down the *amortized* deletion time (see further on), we apply a CLEANUP algorithm, which links trees of equal degree until there is only one root node of any particular degree.
Let us describe the CLEANUP algorithm in more detail. This algorithm maintains a global array $B[1 \ldots \lfloor \log n \rfloor]$, where $B[i]$ is a pointer to some previously-visited root node of degree i, or NULL if there is no such previously-visited root node. Notice, the CLEANUP algorithm simultaneously resets the parent pointers of all the new roots and updates the pointer to the minimum key. The part of the algorithm that links possible nodes of equal degree is given in a separate subroutine LINKDUPES, see Figure 10. The subroutine

```
PROMOTE:
unmark v
if parent(v) ≠ NULL
    remove v from parent(v)'s child list
    insert v into the root list
    if parent(v) is marked
        PROMOTE(parent(v))
    else
        mark parent(v)
```

Fig. 11: The PROMOTE algorithm

ensures that no earlier root node has the same degree as the current. By the possible swapping of the nodes v and w, we maintain the heap property. We shall analyze the efficiency of the *delete-min* operation further on. The fact that the array B needs at most $\lfloor \log n \rfloor$ entries is proven in Section 15, where we prove that the degree of any (root) node in an n-node Fibonacci heap is bounded by $\lfloor \log n \rfloor$.

13.2 The *decrease-key* Operation

If we also need the ability to delete an arbitrary node. The usual way to do this is to decrease the node's key to $-\infty$ and then use *delete-min*. We start by describing how to decrease the key of a node in a Fibonacci heap; the algorithm will take $O(\log n)$ time in the worst case, but the *amortized* time will be only $O(1)$. Our algorithm for decreasing the key at a node v follows two simple rules:

1. If $newkey(v) < key(parent(v))$, promote v up to the root list (this moves the whole subtree rooted at v).

2. As soon as two children of any node w have been promoted, immediately promote w.

In order to enforce the second rule, we now mark certain nodes in the Fibonacci heap. Specifically, a node is marked if exactly one of its children has been promoted. If some child of a marked node is promoted, we promote (and unmark) that node as well. Whenever we promote a marked node, we

unmark it; this is the only way to unmark a node (if splicing nodes into the root list during a *delete-min* is not considered a promotion). A more formal description of the PROMOTE algorithm is given in Figure 11. This algorithm is executed if the new key of the node v is smaller than its parent's key.

14 Amortized Analysis

In an *amortized analysis*, time required to perform a sequence of data structure operations is averaged over all the operations performed. Amortized analysis can be used to show that the average cost of an operation is small, if one averages over a sequence of operations, even though a single operation might be expensive. Amortized analysis differs from average-case analysis in that probability is not involved; an amortized analysis guarantees that *average performance of each operation in the worst case.*

There are several techniques used to perform an amortized analysis, the method of amortized analysis used to analyze Fibonacci heaps is the potential method. When using this method we determine the the amortized cost of each operation and may overcharge operations early on to compensate for undercharges later. The potential method works as follows. We start with an initial data structure D_0 on which s operations are performed. For each $i = 1, \ldots, s$, we let c_i be the actual cost of the i-th operation and D_i be the data structure that results after applying the i-th operation to the data structure D_{i-1}. A potential function Φ maps each data structure D_i to a real number $\Phi(D_i)$, which is the *potential* (energy) associated with the data structure D_i. The *amortized cost* $\hat{c}_i$ of the i-th operation with respect to the potential function Φ is defined by:

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}). \tag{1}$$

The amortized cost of each operation is thus its actual cost plus the increase in potential due to the operation. The total amortized costs of the s operations is

$$\sum_i \hat{c}_i = \sum_i c_i + \Phi_{D_s} - \Phi_{D_0}, \tag{2}$$

If we can prove that $\Phi_{D_s} \geq \Phi_{D_0}$, then we have shown that the amortized costs bound the real costs. Thus, we can analyze the amortized costs to obtain a bound on the actual costs. In practice, we do not know how many

operations s might be performed. Therefore, if we require that $\Phi(D_i) \geq \Phi_{D_0}$ for all i, then we guarantee that we pay in advance. It is often convenient to define $\Phi(D_0) = 0$ and then to show that $\Phi(D_i) \geq 0$.

Intuitively, if the potential difference $\Phi(D_i) - \Phi(D_{i-1})$ of the i-th operation is positive, then the amortized cost $\hat{c}_i$ represents an overcharge to the i-th operation, and the potential of the data structure increases. If the potential difference is negative, then the amortized costs represents an undercharge and the actual cost of the operation is paid by a decrease in the potential.

14.1 Amortized Analysis of the *delete-min* and *decrease-key* Operation

Define $\Phi(H)$ as the number of root nodes $t(H)$ plus two times the number of marked nodes $m(H)$ in the Fibonacci heap H, i.e., $\Phi(H) = t(H) + 2m(H)$. We assume that a single unit of potential can pay for a constant amount of work, where the constant is sufficiently large to cover the cost of any of the specific constant-time pieces of work that we might encounter. Assume that a Fibonacci heap application begins with no heaps (this is the case for both the SP and MST algorithm). The initial potential, therefore, is 0, and obviously the potential is nonnegative at all subsequent times. Hence, the total amortized cost is thus an upper bound on the total actual cost for the sequence of operations (see Eq. (2)). We further assume that there is some upper bound $D(n)$ on the maximum degree of any node in an n-node Fibonacci heap. We derive this upper bound in Section 15.

The actual cost of a delete-min operation can be accounted for as follows. An $O(1)$ contribution comes from splicing the (at most $D(n)$) children of the minimum node in the root list (because setting the parent pointers to Null and unmarking the nodes is done by the Cleanup algorithm). The size of the root list upon calling the Cleanup algorithm is $D(n) + t(H) - 1$, since it consist of the original $t(H)$ root list nodes, minus the minimum node, plus the children of the extracted node. Meaning, at most $D(n) + t(H) - 1$ link operations are performed. Thus, the total amount of work performed is at most proportional to $D(n)+t(H)$, i.e., $O(D(n)+t(H))$. The potential before extracting the minimum node is $t(H) + 2m(H)$, and the potential afterward is at most $(D(n) + 1) + 2m(H)$, since at most $D(n) + 1$ roots remain and no nodes become marked during the operation. The amortized cost is thus at

most

$$\begin{aligned} & O(D(n) + t(H)) + ((D(n) + 1) + 2m(H)) - (t(H) + 2m(H)) \\ &= O(D(n)) + O(t(H)) - t(H) \\ &= O(D(n)), \end{aligned}$$

since we can scale up the units of the potential to dominate the hidden constant in $O(t(H))$. Intuitively, the cost of performing the link operations is paid by the decrease in the potential due to reducing the number of nodes on the root list.

Let us now consider the *decrease-key* operation. Decreasing the key has an actual cost of $O(1)$. Suppose that c recursive invocations of the PROMOTE function are called. Each recursive call takes O(1) time except for the recursive calls, hence, the actual cost of *decrease-key* is O(c). Next, we compute the change in potential. Each recursive call, except for the last, cuts a marked node and unmarks the node. Afterward, there are $t(H) + c$ trees (the original $t(H)$, $c - 1$ trees produced by the recursive calls, and the tree rooted at the node whose key was decreased). Whereas the maximum number of marked nodes equals $m(H) - c + 2$ ($c - 1$ were unmarked and the last call may have marked a node). The change in potential is therefore at most

$$((t(H) + c) + 2(m(H) - c + 2) - (t(H) + 2m(H)) = 4 - c. \tag{3}$$

Thus, the amortized cost of the *decrease-min* is at most

$$O(c) + 4 - c = O(1), \tag{4}$$

by scaling up the units of the potential to dominate the hidden constant in $O(c)$.

15 Bounding the Maximum Degree

In order to prove that the amortized time of the *delete-min* operation is $O(\log(n))$, we must show that $D(n)$ is bounded by $O(\log n)$. In this section we shall show that $D(n) \leq \lfloor \log_\phi n \rfloor$, where $\phi = (1 + \sqrt{5})/2$ is the golden ratio.

LEMMA 15.1: Let x be any node in a Fibonacci heap, and suppose that $d(x) = k$, where $d(x)$ denotes the degree of x. Let $y_1, y_2, \ldots, y_k$ denote the children of x in the order in which they were linked to x, from the earliest to the latest. Then, $d(y_1) \geq 0$ and $d(y_i) \geq i - 2$, for $i = 2, 3, \ldots, k$.

Proof: Obviously, $d(y_1) \geq 0$. For $i \geq 2$, we note that when y_i was linked to x, all of $y_1, y_2, \ldots, y_{i-1}$ were children of x, so we must have had $d(y_i) \geq i-1$, as y_i was only linked to x if $d(x) = d(y_i)$. Since then, node y_i has lost at most one child, otherwise y_i would have been cut. We may conclude $d(y_i) \geq i-2$. Q.E.D.

Let us now define the Fibonacci numbers F_k, for $k \geq 0$ as follows: $F_0 = 0$, $F_1 = 1$ and $F_k = F_{k-1} + F_{k-2}$, for $k \geq 2$. Then, we can easily show by induction on k that $F_{k+2} = 1 + \sum_{i=0}^{k} F_k$, for all $k \geq 0$ (simply apply induction on the term F_{k+1}). Moreover, $F_{k+2} \geq \phi^k$ (apply induction on both terms and use the fact that $(1 + \phi) = \phi^2$).

THEOREM 15.1: Let x be any node in a Fibonacci heap, and let $k = d(x)$. Then, $size(x) \geq F_{k+2} \geq \phi^k$, where $\phi = (1 + \sqrt{5})/2$.

Proof: Let s_k denote the minimum possible value of $size(z)$ over all nodes z such that $d(z) = k$. That is, s_k denotes the minimum number of nodes in a tree that is rooted by a degree k root node. Let $y_1, y_2, \ldots, y_k$ denote the children of x as in Lemma 15.1 in the order they were linked to x. To compute a lower bound on $size(x)$, we count one for x itself, one for the first child y_1 and then apply Lemma 15.1 for the remaining children. We have

$$size(x) \geq s_k \geq 2 + \sum_{i=2}^{k} s_{i-2}. \tag{5}$$

We now show by induction that $s_k \geq F_{k+2}$, for all $k \geq 0$. The cases for $k = 0$ and 1 are trivial. By induction, we have $s_i \geq F_{i+2}$ for $i = 0, \ldots, k-1$, therefore,

$$\begin{aligned} s_k &\geq 2 + \sum_{i=2}^{k} s_{i-2} \\ &\geq 1 + F_1 + \sum_{i=2}^{k} F_i = F_{k+2}. \end{aligned}$$

Thus, we have shown $size(x) \geq s_k \geq F_{k+2} \geq \phi^k$.

Q.E.D.

COROLLARY 15.1: The maximum degree $D(n)$ of any node in an n-node Fibonacci heap is bounded by $\lfloor \log_\phi n \rfloor$, meaning it is $O(\log n)$.

Proof: Let x be any node of an n-node Fibonacci heap and let k be its degree. By the previous theorem we have $n \geq size(x) \geq \phi^k$. Taking the base-$\phi$ logarithms yields $k \leq \log_\phi n$. Therefore, the maximum degree is $O(\log n)$.

Q.E.D.

As a results of this corollary, Prim's MST algorithm (or Dijkstra's SP) has an amortized cost of $O(n \log n + e)$, as a *decrease-key* operation is $O(1)$ and a *delete-min* operations is $O(\log n)$.

Lecture 5

Graphs

[Part 1]

Lecture Content

1. Graph Basics

1.1 Definitions and Terminologies

1.2 Data Structures Used to Store Graphs

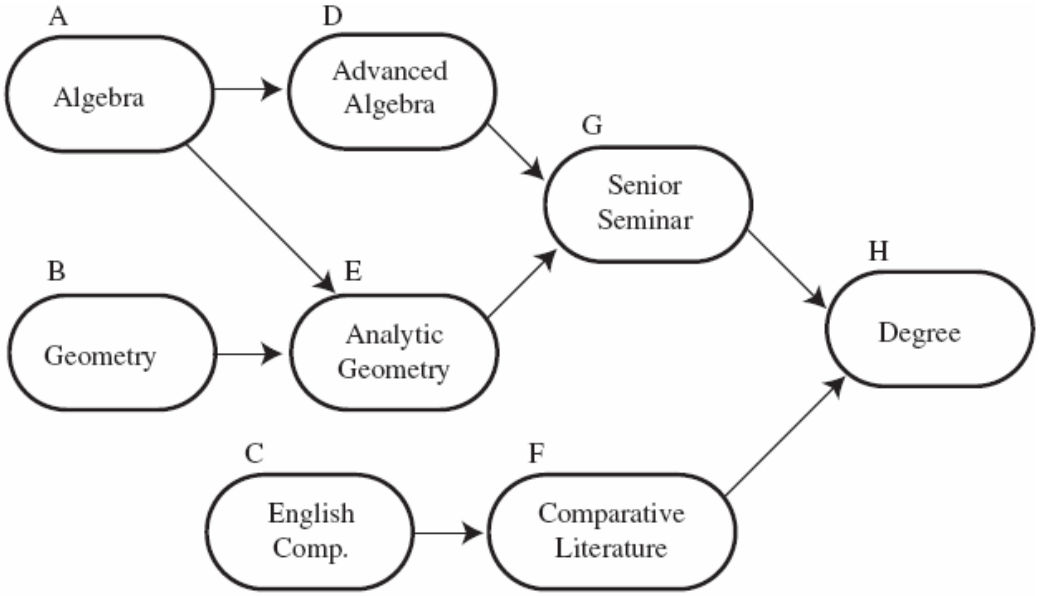
2. Graph Traversal

2.1 Depth-First Search (DFS)

2.2. Breadth-First Search (BFS)

Lecture Content

3. Topological Sorting

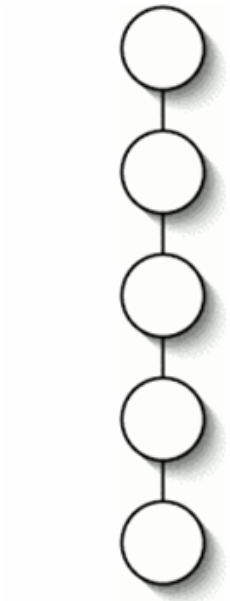


Course prerequisites

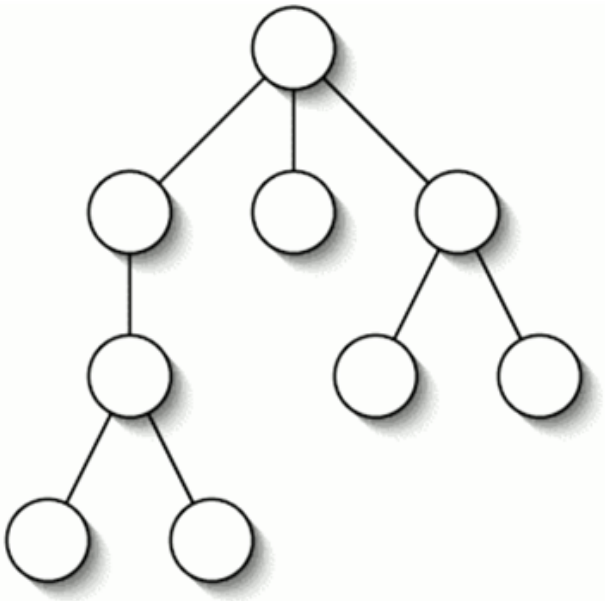
Topologically sorted order: C F B A E D G H

1. Graph Basics

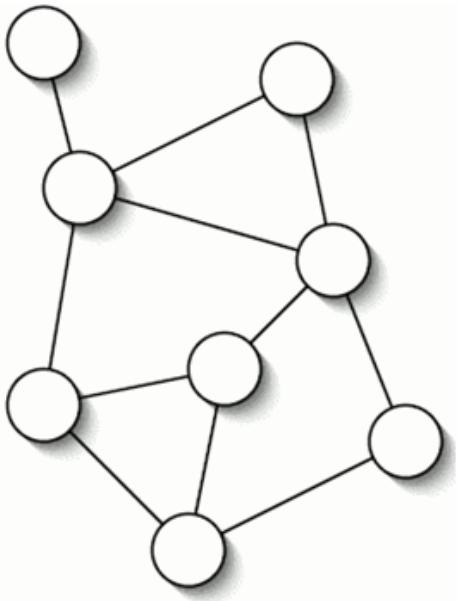
- Tree generalizes linear structures (i.e., singly linked list), graph generalizes tree.



Linear structure



Tree



Graph

1. Graph Basics

• The key difference between tree and graph is that, in a graph, there may be more than one path between two nodes.

• Many real-world problems can be modeled by using graphs. For example,

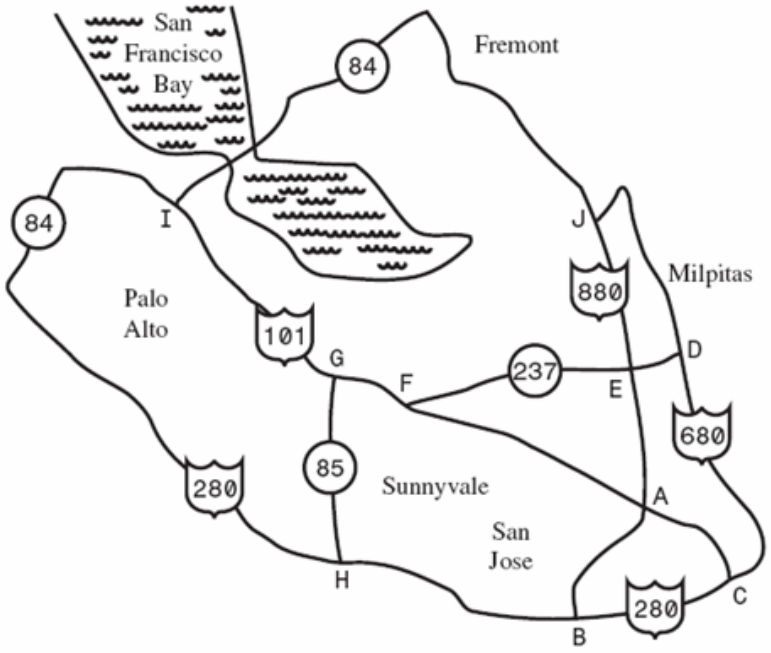
- finding the fastest routes for a mass transportation system (answers to the question: what is the shortest driving route from city *A* to city *B*),

1. Graph Basics

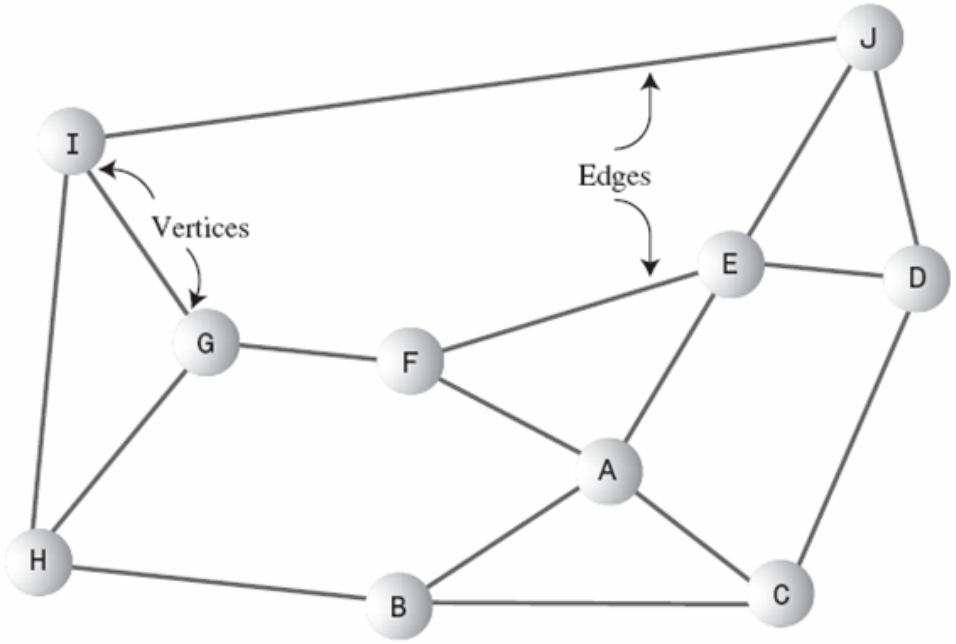
- finding a minimum spanning tree (answers to the question: how can computers be connected with the least amount of cable)

- routing electronic email through a computer network.

1. Graph Basics



Roadmap



Corresponding graph

1.1 Definitions and Terminologies

• A graph G consists of a set of **vertices** (also called nodes) V and a set of **edges** (also called arcs) E that connect the vertices.

• That is, $G = (V, E)$, where V is a set of n vertices $\{v_0, v_1, \ldots, v_{n-1}\}$ and E is a set of m edges $\{e_0, e_1, \ldots, e_{m-1}\}$.

• Each edge $e \in E$ is a pair (u, v), where $u, v \in V$ (i.e., $e = (u, v)$).

1.1 Definitions and Terminologies

• The number of vertices and edges of a graph G is denoted as $|V|$ and $|E|$, respectively (i.e., $|V| = n$ and $|E| = m$).

• If each edge $e = (u, v)$ in G is ordered (i.e., $(u, v) \neq (v, u)$), the graph is called **directed graph** (also called **digraph**). Otherwise, the graph is called **undirected graph**.

1.1 Definitions and Terminologies

- If a graph is directed, the **in-degree of a vertex** is the number of edges entering it.
- The **out-degree of a vertex** is the number of edges leaving it.
- The **degree of a vertex** is the sum of its in-degree and out-degree.

1.1 Definitions and Terminologies

• If each edge $e = (u, v)$ in G has a **weight** w (also called cost or length), the graph is called **weighted graph**.

• Vertex v is **adjacent** (also called **neighbor**) to vertex u if there is an edge from u to v.

• A **path** in a graph is a sequence of vertices connected by edges.

1.1 Definitions and Terminologies

• In **unweighted graphs**, a **path length** is the number of edges on the path.

• The **distance** between two vertices is the length of the shortest path between them.

• A **weighted path length** is the sum of weights (costs or lengths) on the path.

1.1 Definitions and Terminologies

- If $|E| = \Theta(|V|^2)$, then the graph G is called **dense graph**.

- If $|E| = \Theta(|V|)$, then the graph G is called **sparse graph**.

- A **cycle** is a path from a vertex back to itself.

1.1 Definitions and Terminologies

- A graph with no cycle is called **acyclic** graph. A directed acyclic graph is called a **DAG**.
- A graph in which every pair of vertices is connected by a path is said to be **connected**.
- Let G be a simple graph (i.e., no parallel edges and no self-loop/cycle) with n vertices and m edges. If G is undirected, then $m \leq n(n - 1)/2$. If G is directed, then $m \leq n(n - 1)$.

1.2 Data Structures Used to Store Graphs

• A graph can be stored by using an **adjacency matrix** (i.e., two-dimensional array, also called **neighbor matrix**) or an **adjacency list** (i.e., singly linked list).

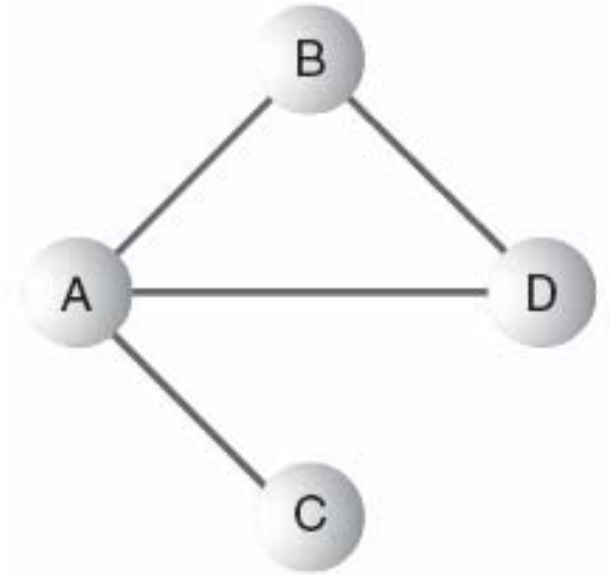
• Normally, dense and sparse graphs are represented by using adjacency matrix and an adjacency list, respectively.

Adjacency Matrix

- An adjacency matrix (e.g., `int adjMat[][]`) is a two-dimensional array in which the elements indicate whether an edge is present between two vertices. If a graph has n vertices, an $n \times n$ adjacency matrix is needed to store the graph.

Adjacency Matrix

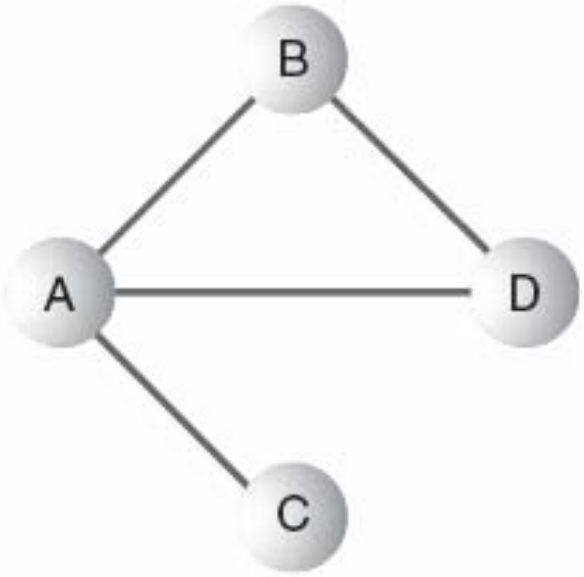
- **Example**: Consider the following graph



Adjacency Matrix

- The adjacency matrix for the graph is as follows.

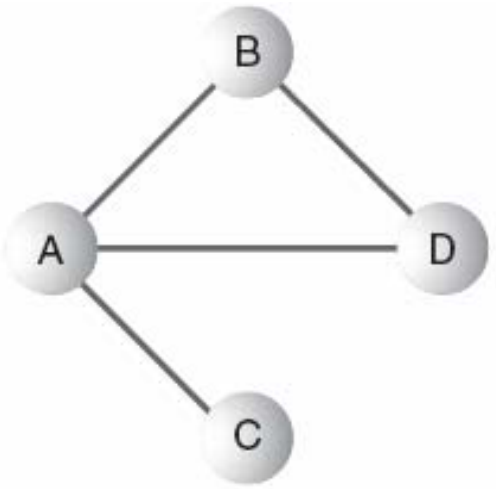
	A	B	C	D
A	0	1	1	1
B	1	0	0	1
C	1	0	0	0
D	1	1	0	0



Adjacency List

- An adjacency list is an array of singly linked lists. Each individual list shows **what vertices a given vertex is adjacent to**.
- The adjacency lists for the graph is given next.

A	B→C→D
B	A→D
C	A
D	A → B



Lecture Content

1. Graph Basics

   1.1 Definitions and Terminologies

   1.2 Data Structures Used to Store Graphs

2. Graph Traversal

   2.1 Depth-First Search (DFS)

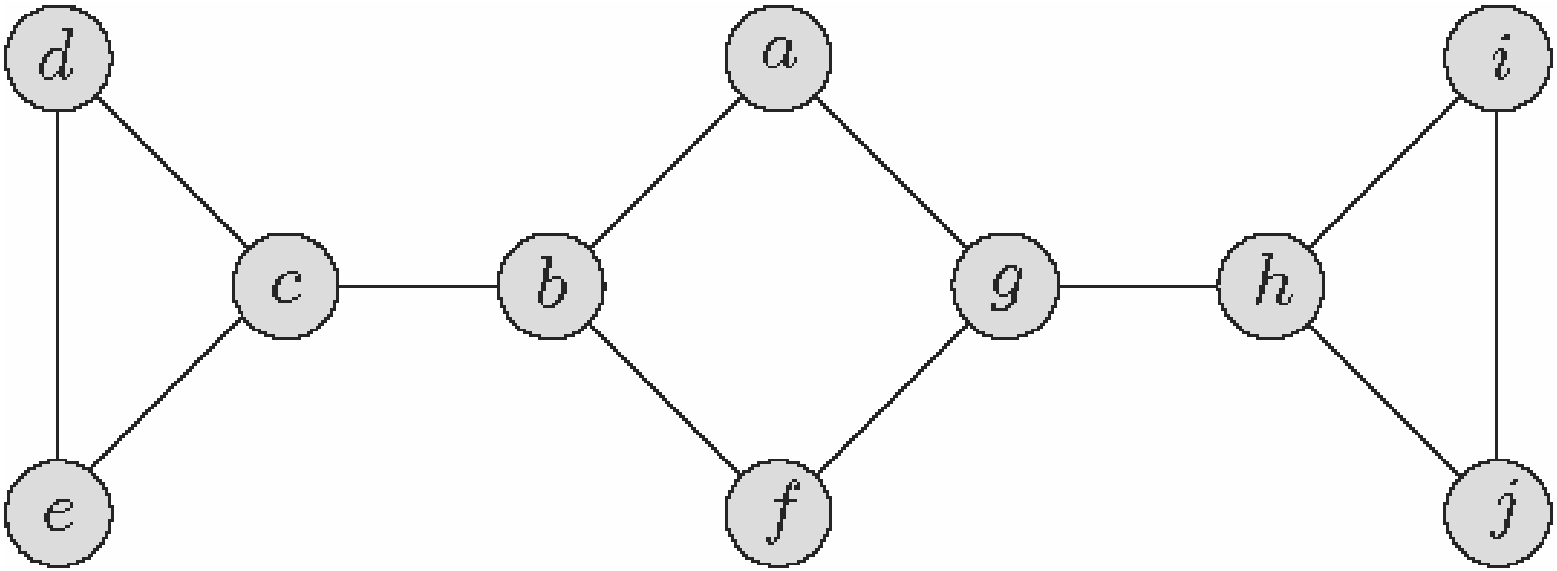
   2.2. Breadth-First Search (BFS)

2. Graph Traversal

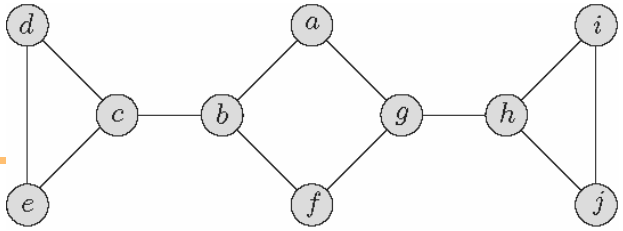
- Three traversals of a tree are preorder, inorder, and postorder. Tree traversal is always starts at the root of the tree.
- Two traversals of a graph are depth-first search (DFS) and breadth-first search (BFS). Since a graph has no root, we must specify a vertex at which to begin a traversal.
- Depth-first search is essentially a generalization of the preorder traversal of a rooted tree.

2.1 Depth-First Search (DFS)

• **Example**: List the order in which the nodes of the undirected graph shown in the figure below are visited by a *depth-first traversal* that starts from vertex *a*. Assume that we choose to visit adjacent vertices in alphabetical order.



2.1 Depth-First Search (DFS)



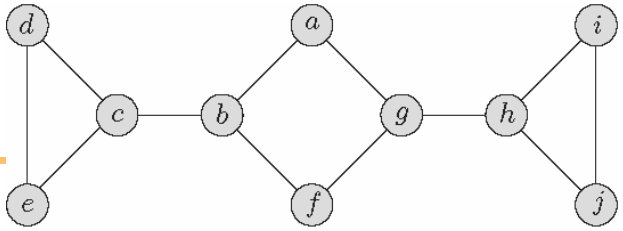
Algorithm DFS // M.H. Alsuwaiyel

Input: A directed or undirected graph $G = (V, E)$.

Output: Numbering of the vertices in depth-first search order.

1. *predfn* ← 1; *postdfn* ← 1
2. **for** each vertex $v \in V$
3. mark *v unvisited*
4. **end for**
5. **for** each vertex $v \in V$
6. **if** *v* is marked *unvisited* **then** *dfs*(*v*) // starting vertex
7. **end for**

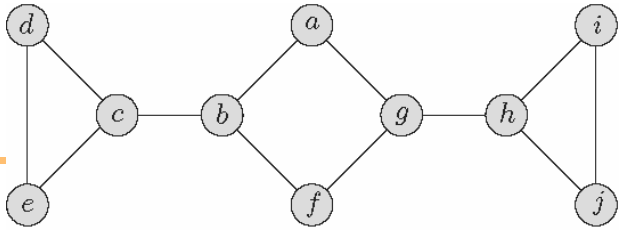
2.1 Depth-First Search (DFS)



Procedure *dfs*(*v*) // *v* is starting vertex, using stack

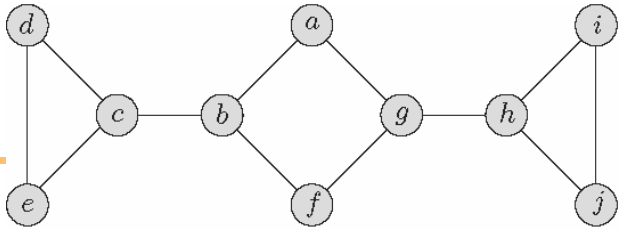
1. $S \leftarrow \{v\}$ // insert *v* into stack
2. mark *v visited*
3. **while** $S \neq \{\}$
4. $v \leftarrow$ Peek(S) // *v* is current vertex
5. find an unvisited neighbor *w* of *v*

2.1 Depth-First Search (DFS)




```
6.    if w exists then
7.        Push(w, S)
8.        mark w visited
9.        predfn ← predfn + 1
10.   else
11.       Pop(S); postdfn ← postdfn + 1
12.   end if
13. end while
```

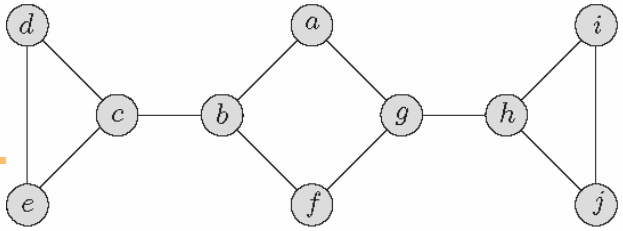
2.1 Depth-First Search (DFS)



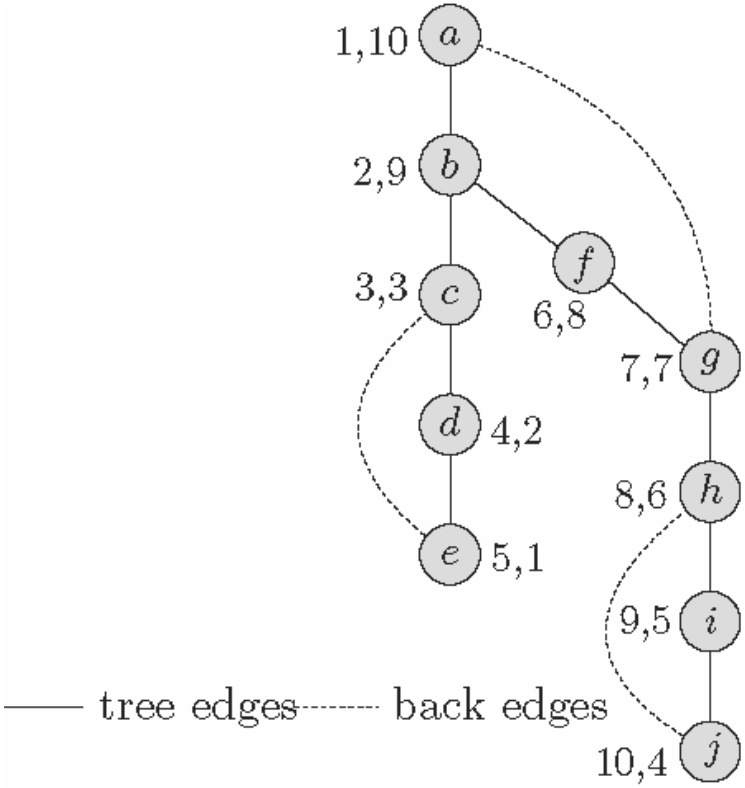
- The stack contents during DFS are given below.

Event	Stack Contents	Current vertex
Visit *a*	*a*	*a*
Visit *b*	*a*, *b*	*b*
Visit *c*	*a*, *b*, *c*	*c*
Visit *d*	*a*, *b*, *c*, *d*	*d*
Visit *e*	*a*, *b*, *c*, *d*, *e*	*e*
Pop *e*	*a*, *b*, *c*, *d*	*d*
Pop *d*	*a*, *b*, *c*	*c*
Pop *c*	*a*, *b*	*b*
Visit *f*	*a*, *b*, *f*	*f*
Visit *g*	*a*, *b*, *f*, *g*	*g*
Visit *h*	*a*, *b*, *f*, *g*, *h*	*h*
Visit *i*	*a*, *b*, *f*, *g*, *h*, *i*	*i*
Visit *j*	*a*, *b*, *f*, *g*, *h*, *i*, *j*	*j*
Pop *j*	*a*, *b*, *f*, *g*, *h*, *i*	*i*
Pop *i*	*a*, *b*, *f*, *g*, *h*	*h*
Pop *h*	*a*, *b*, *f*, *g*	*g*
Pop *g*	*a*, *b*, *f*	*f*
Pop *f*	*a*, *b*	*b*
Pop *b*	*a*	*a*
Pop *a*	Empty	

2.1 Depth-First Search (DFS)



- The order in which the nodes are visited by a DFS that starts from vertex *a* is *a*, *b*, *c*, *d*, *e*, *f*, *g*, *h*, *i*, *j*.
- The resulting tree (i.e., the *depth-first search tree*) is



2.1 Depth-First Search (DFS)

- Tree edges: edges in the depth-first search tree. An edge (v, w) is a tree edge if w was first visited when exploring the edge (v, w).
- Back edges: All other edges.

2.1 Depth-First Search (DFS)

In depth-first search traversal of directed graphs, however, the edges of G are classified into four types:

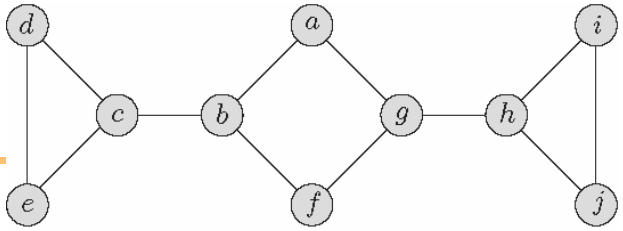
- Tree edges: edges in the depth-first search tree. An edge (v, w) is a tree edge if w was first visited when exploring the edge (v, w).
- Back edges: edges of the form (v, w) such that w is an ancestor of v in the depth-first search tree (constructed so far) and vertex w was marked visited when (v, w) was explored.

2.1 Depth-First Search (DFS)

• Forward edges: edges of the form (v, w) such that w is a descendant of v in the depth-first search tree (constructed so far) and vertex w was marked visited when (v, w) was explored.

• Cross edges: All other edges.

2.1 Depth-First Search (DFS)



Procedure *dfs*(*v*) // *v* is starting vertex, using recursion

1. mark *v* visited
2. *predfn* ← *predfn* + 1
3. **for** each edge (*v*, *w*) ∈ *E*
4. **if** *w* is marked *unvisited* **then** *dfs*(*w*)
5. **end for**
6. *postdfn* ← *postdfn* + 1

2.2 Breadth-First Search (BFS)

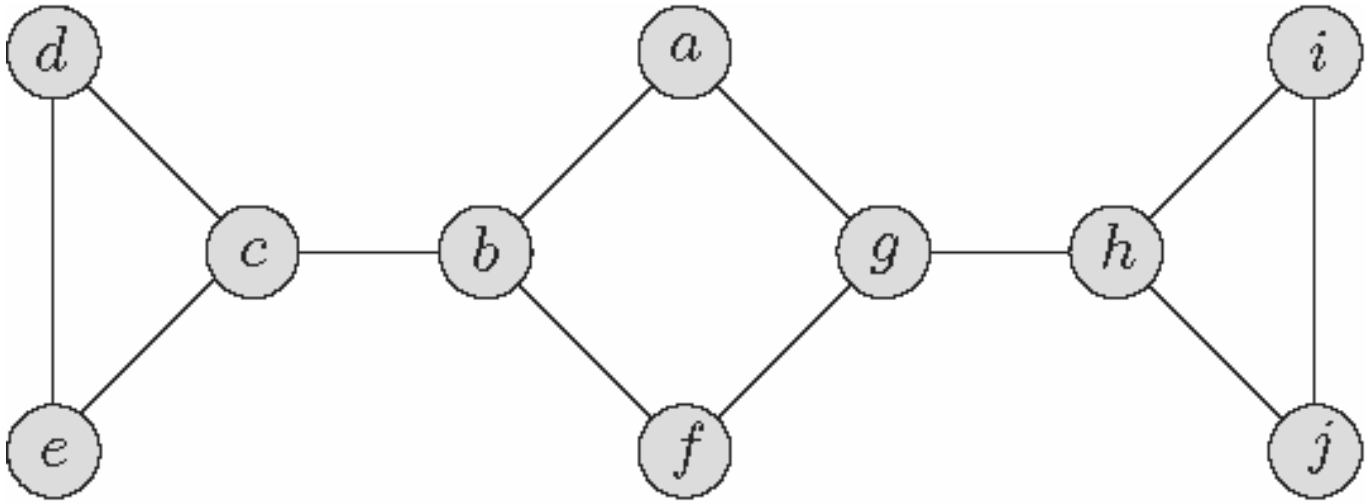
• Depth-first search algorithm gets as far away from the starting point as quickly as possible. DFS is implemented using a stack.

• In contrast, breadth-first search algorithm stays as close as possible to the starting point. BFS visits all the vertices adjacent to the starting vertex, and then goes further afield. BFS is implemented using a queue.

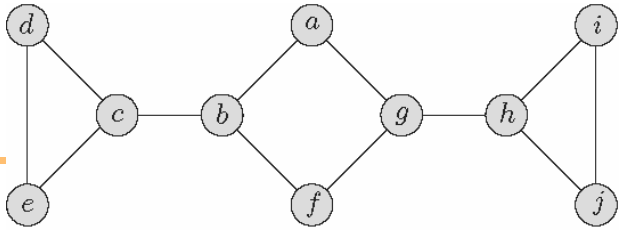
• The level-order traversal of a tree is an example of the breadth-first traversal.

2.2 Breadth-First Search (BFS)

• **Example**: List the order in which the nodes of the undirected graph shown in the figure below are visited by a *breadth-first traversal* that starts from vertex *a*. Assume that we choose to visit adjacent vertices in alphabetical order.



2.2 Breadth-First Search (BFS)



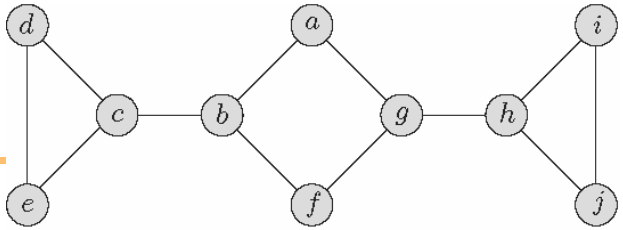
Algorithm BFS // M.H. Alsuwaiyel

Input: A directed or undirected graph $G = (V, E)$.

Output: Numbering of the vertices in BFS order.

1. $bfn \leftarrow 1$
2. **for** each vertex $v \in V$
3. mark v *unvisited*
4. **end for**
5. **for** each vertex $v \in V$
6. **if** v is marked *unvisited* **then** *bfs*(v) // starting vertex
7. **end for**

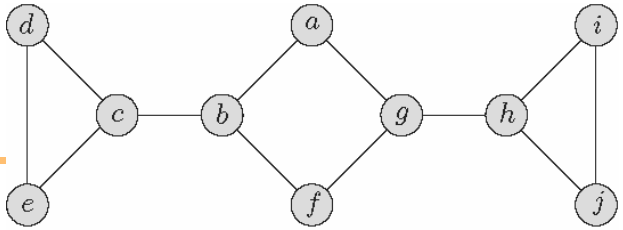
2.2 Breadth-First Search (BFS)



Procedure *bfs*(*v*) // *v* is starting vertex, using queue

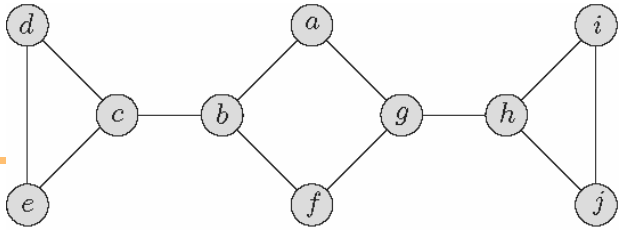
1. $Q \leftarrow \{v\}$ // insert *v* into queue
2. mark *v visited*
3. **while** $Q \neq \{\}$
4. $\quad v \leftarrow$ dequeue(Q) // *v* is current vertex
5. $\quad$ **for** each edge $(v, w) \in E$
6. $\quad\quad$ **if** *w* is marked *unvisited* **then**
7. $\quad\quad\quad$ enqueue(w, Q)
8. $\quad\quad\quad$ mark *w visited*
9. $\quad\quad\quad$ *bfn* $\leftarrow$ *bfn* + 1
10. $\quad\quad$ **end if**

2.2 Breadth-First Search (BFS)



11. **end for**
12. **end while**

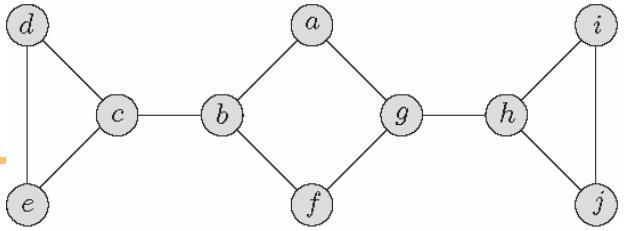
2.2 Breadth-First Search (BFS)



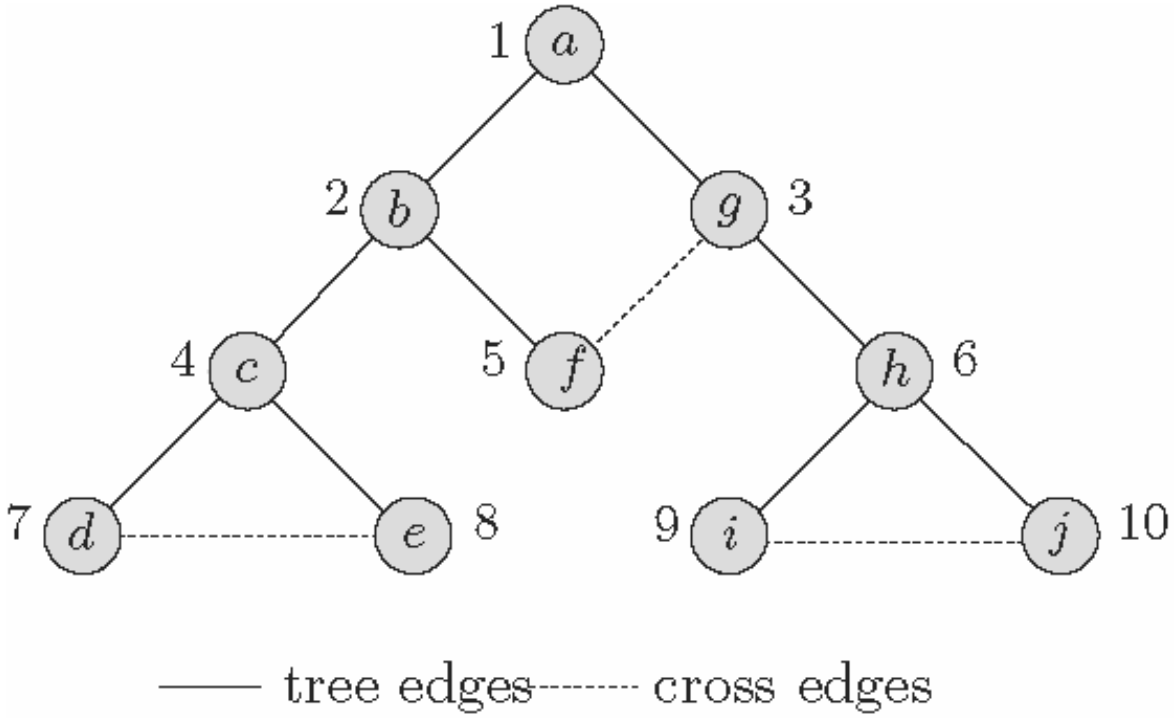
- The queue contents during BFS are given below.

Event	Queue Contents	Current vertex
Visit *a*	*a*	*a*
Visit *b*	*b*, *g*	*b*
Visit *g*	*g*, *c*, *f*	*g*
Visit *c*	*c*, *f*, *h*	*c*
Visit *f*	*f*, *h*, *d*, *e*	*f*
Visit *h*	*h*, *d*, *e*	*h*
Visit *d*	*d*, *e*, *i*, *j*	*d*
Visit *e*	*e*, *i*, *j*	*e*
Visit *i*	*i*, *j*	*i*
Visit *j*	*j*	*j*
Done	Empty	

2.2 Breadth-First Search (BFS)



- The order in which the nodes are visited by a BFS that starts from vertex *a* is *a*, *b*, *g*, *c*, *f*, *h*, *d*, *e*, *i*, *j*.
- The resulting tree (i.e., the *breadth-first search tree*) is

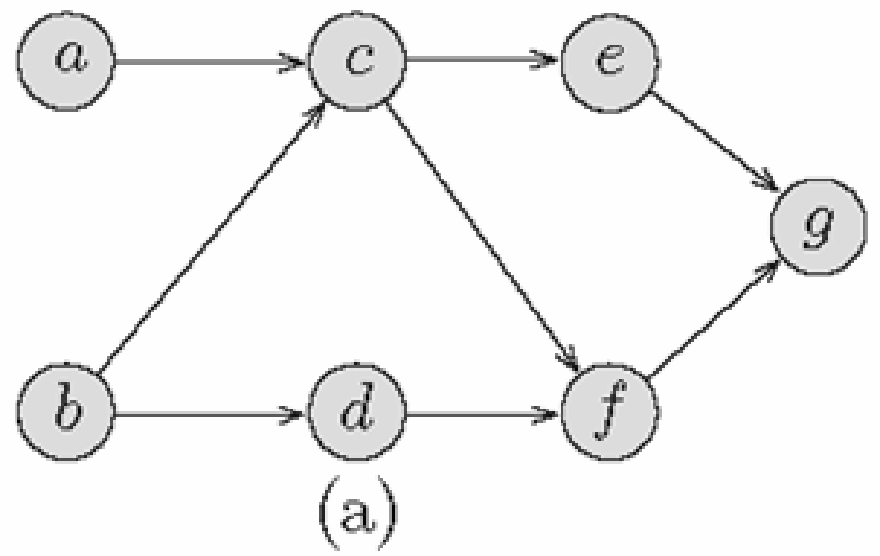


2.2 Breadth-First Search (BFS)

- Tree edges: edges in the breadth-first search tree. An edge (v, w) is a tree edge if w was first visited when exploring the edge (v, w).
- Cross edges: All other edges.

3. Topological Sorting

• Given a directed acyclic graph (dag for short) $G = (V, E)$, the problem of **topological sorting** is to find a linear ordering of its vertices in such a way that if $(v, w) \in E$, then v appears before w in the ordering.

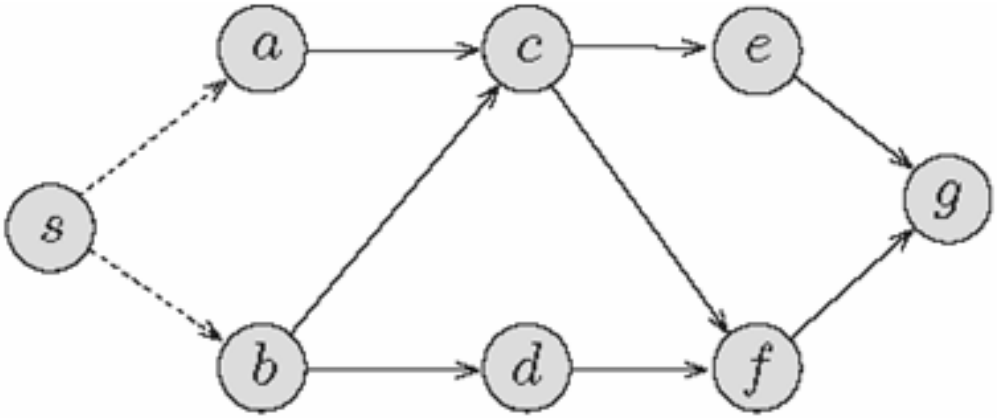


(a)

3. Topological Sorting

• For example, one possible topological sorting of the vertices in the dag shown in figure (a) above is b, d, a, c, f, e, g (or a, b, d, c, e, f, g)

• We will assume that the dag has only one vertex, say s, of indegree 0. If not, we may simply add a new vertex s and edges from s to all vertices of indegree 0.



(b)

3. Topological Sorting

- Next, we simply carry out a depth-first search on *G* starting at vertex *s*.
- When the traversal is complete, the values of the counter *postdfn* define a reverse topological ordering of the vertices in the dag.
- Thus, to obtain the ordering, we may add an output step to Algorithm DFS just after the counter *postdfn* is incremented. The resulting output is reversed to obtain the desired topological ordering.

Exercises

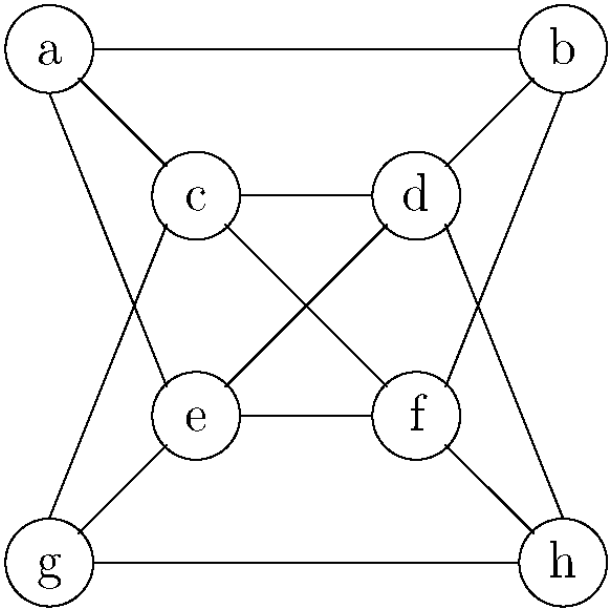
1. Write a complete program to implement the DFS.

2. Write a complete program to implement the BFS.

3. Modify the DFS program to find the topologically sorted order of a given dag.

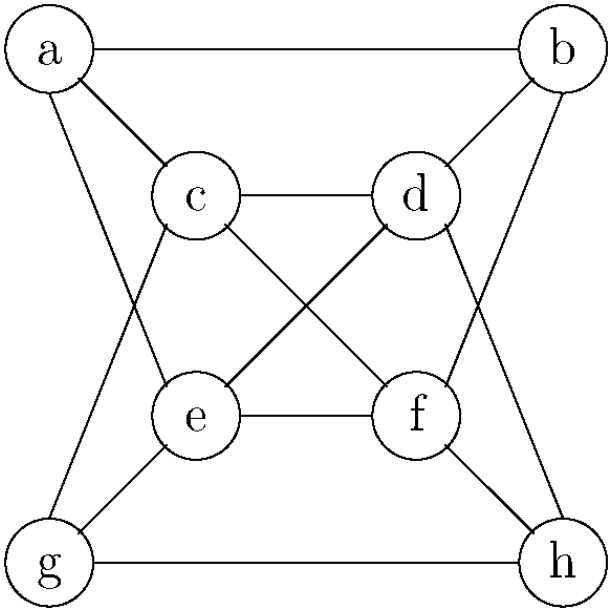
Exercises

4. List the order in which the nodes of the undirected graph shown in the figure below are visited by a depth-first traversal that starts from vertex *a*. Repeat this exercise for a depth-first traversal starting from vertex *d*.



Exercises

5. List the order in which the nodes of the undirected graph shown in the figure below are visited by a breadth-first traversal that starts from vertex *a*. Repeat this exercise for a breadth-first traversal starting from vertex *d*.



References

1. Noel Kalicharan. 2008. *Data Structures in Java*. CreateSpace. ISBN: 143827517X. Chapter 5

2. Noel Kalicharan. 2008. *Data Structures in C*. Createspace Press. ISBN: 1438253273. Chapter 7

3. Robert Lafore. 2002. *Data Structures and Algorithms in Java*. 2nd Ed, SAMS. ISBN: 0672324539.

References

4. M.H. Alsuwaiyel. 1999. *Algorithms Design Techniques and Analysis*. World Scientific Publishing. ISBN: 9810237405. Chapters 1, 4, 6

5. Anany V. Levitin. 2011. *Introduction to the Design and Analysis of Algorithms*. 3Ed. Addison-Wesley. ISBN: 0132316811.

Notes on Strongly Connected Components

Recall from Section 3.5 of the Kleinberg-Tardos book that the *strongly connected components* of a directed graph G are the equivalence classes of the following equivalence relation: $u \sim v$ if and only if there is a directed $u \rightsquigarrow v$ path and also there is a directed $v \rightsquigarrow u$ path. (Check that this is indeed an equivalence relation.) For example, in the directed graph in Figure 1, the strongly connected components are identified by the dashed circles.

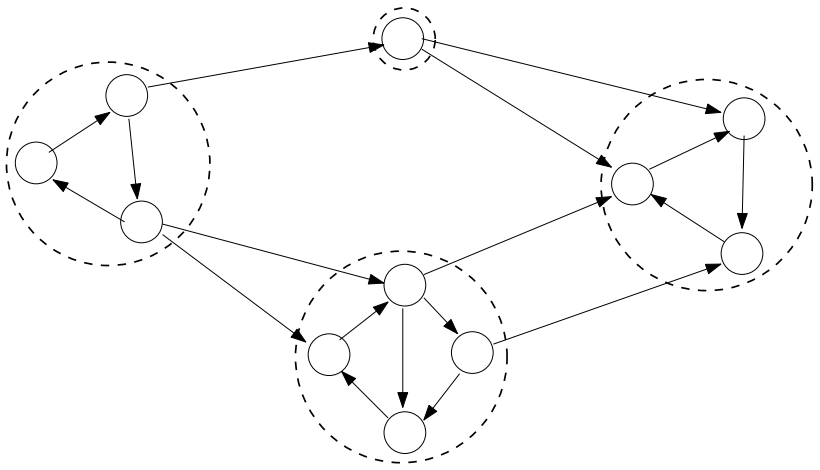


Figure 1: The strongly connected components of a directed graph.

1 The Algorithm

Goal of Lecture: to give a linear-time (i.e., $O(m+n)$-time) algorithm that computes the strongly connected components of a directed graph.

The algorithm we present is essentially two passes of depth-first search, plus some extremely clever additional book-keeping. The algorithm is described in a top-down fashion in Figures 2–4.

Input: a directed graph $G = (V, E)$, in adjacency list representation. Assume that the vertices V are labeled $1, 2, 3, \ldots, n$.

1. Let G^{rev} denote the graph G after the orientation of all arcs have been reversed.
2. Run the DFS-Loop subroutine on G^{rev}, processing vertices according to the given order, to obtain a finishing time $f(v)$ for each vertex $v \in V$.
3. Run the DFS-Loop subroutine on G, processing vertices in decreasing order of $f(v)$, to assign a leader to each vertex $v \in V$.
4. The strongly connected components of G correspond to vertices of G that share a common leader.

Figure 2: The top level of our SCC algorithm. The f-values and leaders are computed in the first and second calls to DFS-Loop, respectively (see below).

Input: a directed graph $G = (V, E)$, in adjacency list representation.

1. Initialize a global variable t to 0.

   [This keeps track of the number of vertices that have been fully explored.]

2. Initialize a global variable s to NULL.

   [This keeps track of the vertex from which the last DFS call was invoked.]

3. For $i = n$ downto 1:

   [In the first call, vertices are labeled $1, 2, \ldots, n$ arbitrarily. In the second call, vertices are labeled by their $f(v)$-values from the first call.]

   (a) if i not yet explored:

      i. set $s := i$

      ii. DFS(G, i)

Figure 3: The DFS-Loop subroutine.

Input: a directed graph $G = (V, E)$, in adjacency list representation, and a source vertex $i \in V$.

1. Mark i as explored.

   [It remains explored for the entire duration of the DFS-Loop call.]

2. Set leader$(i) := s$

3. For each arc $(i, j) \in G$:

   (a) if j not yet explored:

      i. DFS(G, j)

4. $t++$

5. Set $f(i) := t$

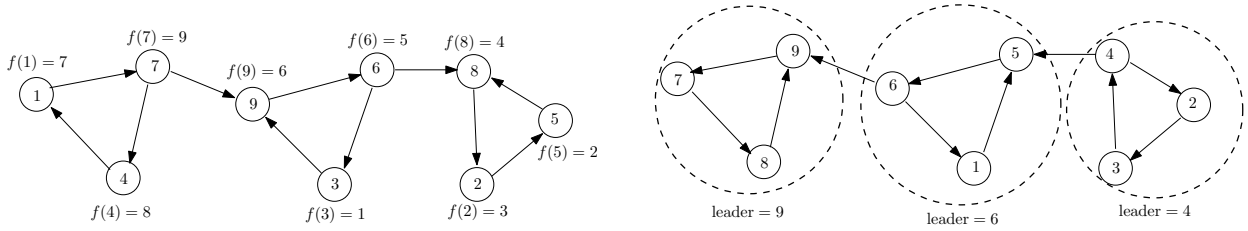
Figure 4: The DFS subroutine. The f-values only need to be computed during the first call to DFS-Loop, and the leader values only need to be computed during the second call to DFS-Loop.

As we've seen, each invocation of DFS-Loop can be implemented in linear time. You should think about how to implement the remaining details of the algorithm so that its overall running time is linear (i.e., $(O(m+n))$).

2 An Example

But why on earth should this algorithm work? An example should increase its plausibility (though it certainly doesn't constitute a proof of correctness). Figure 5(a) displays a reversed graph G^{rev}, with its vertices numbered arbitrarily, and the f-values computed in the first call to DFS-Loop. In more detail, the first DFS is initiated at node 9. The search must proceed next to node 6. DFS then has to make a choice between two different adjacent nodes; we have shown the f-values that ensue when DFS visits node 3 before node 8.[1] When DFS visits node 3 it gets stuck; at this point node 3 is assigned a finishing time of 1. DFS backtracks to node 6, proceeds to node 8, then node 2, and then node 5. DFS then backtracks all the way back to node 9, resulting in nodes 5, 2, 8, 6, and 9 receiving the finishing times 2, 3, 4, 5, and 6, respectively. Execution returns to DFS-Loop, and the next (and final) call to DFS begins at node 7.

Figure 5(b) shows the original graph (with all arcs now unreversed), with nodes labeled with their finishing times. The magic of the algorithm is now evident, as the SCCs of G present themselves to us in order: the first call to DFS discovers the nodes 7–9 (with leader 9); the second the nodes 1, 5, and 6 (with leader 6); and the third the remaining three nodes (with leader 4).



(a) First DFS-Loop on G^{rev}

(b) Second DFS-Loop on G

Figure 5: Example execution of the strongly connected components algorithm. In (a), nodes are labeled arbitrarily and their finishing times are shown. In (b), nodes are labeled by their finishing times and their leaders are shown.

3 Proof of Correctness

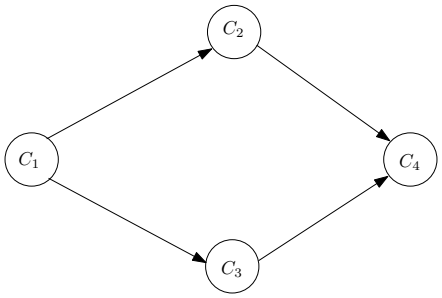
3.1 The Acyclic Meta-Graph of SCCs

First, observe that the strongly connected components of a directed graph form an acyclic "meta-graph", where the meta-nodes correspond to the SCCs $C_1, \ldots, C_k$, and there is an arc $C_h \to C_\ell$ with $h \neq \ell$ if and only if there is at least one arc (i, j) in G with $i \in C_h$ and $j \in C_\ell$. This directed graph must be acyclic: since within a SCC you can get from anywhere to anywhere else on a directed path, in a purported directed cycle of SCCs you can get from every node in a constituent SCC to every other node of every other SCC in the cycle. Thus the purported cycle of SCCs is actually just a single SCC. Summarizing, every directed graph has a useful "two-tier" structure: zooming out, one sees a DAG on the SCCs of the graph; zooming in on a particular SCC exposes its finer-grained structure. For example, the meta-graphs corresponding to the directed graphs in Figures 1 and 5(b) are shown in Figure 6.

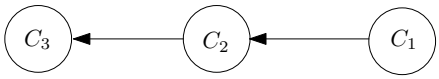
3.2 The Key Lemma

Correctness of the algorithm hinges on the following key lemma.

[1] Different choices of which node to visit next generate different sets of f-values, but our proof of correctness will apply to all ways of resolving these choices.



(a) SCC graph for Figure 1



(b) SCC graph for Figure 5(b)

Figure 6: The DAGs of the SCCs of the graphs in Figures 1 and 5(b), respectively.

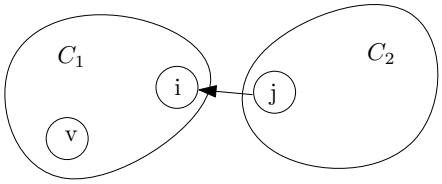
Key Lemma: Consider two "adjacent" strongly connected components of a graph G: components C_1 and C_2 such that there is an arc (i, j) of G with $i \in C_1$ and $j \in C_2$. Let $f(v)$ denote the finishing time of vertex v in some execution of DFS-Loop on the reversed graph G^{rev}. Then

$$\max_{v \in C_1} f(v) < \max_{v \in C_2} f(v).$$

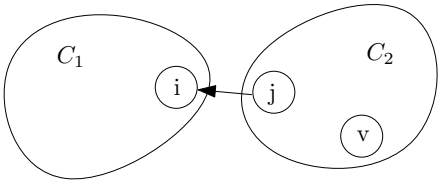
Proof of Key Lemma: Consider two adjacent SCCs C_1 and C_2, as they appear in the reversed graph G^{rev} — where there is an arc (j, i), with $j \in C_2$ and $i \in C_1$ (Figure 7). Because the equivalence relation defining the SCCs is symmetric, G and G^{rev} have the same SCCs; thus C_1 and C_2 are also SCCs of G^{rev}. Let v denote the first vertex of $C_1 \cup C_2$ visited by DFS-Loop in G^{rev}. There are now two cases.

First, suppose that $v \in C_1$ (Figure 7(a)). Since there is no non-trivial cycle of SCCs (Section 3.1), there is no directed path from v to C_2 in G^{rev}. Since DFS discovers everything reachable and nothing more, it will finish exploring all vertices in C_1 without reaching any vertices in C_2. Thus, *every* finishing time in C_1 will be smaller that *every* finishing time in C_2, and this is even stronger than the assertion of the lemma. (Cf., the left and middle SCCs in Figure 5.)

Second, suppose that $v \in C_2$ (Figure 7(b)). Since DFS discovers everything reachable and nothing more, the call to DFS at v will finish exploring all of the vertices in $C_1 \cup C_2$ before ending. Thus, the finishing time of v is the largest amongst vertices in $C_1 \cup C_2$, and in particular is larger than all finishing times in C_1. (Cf., the middle and right SCCs in Figure 5.) This completes the proof.



(a) All f-values in C_1 smaller than in C_2



(b) v has the largest f-value in $C_1 \cup C_2$

Figure 7: Proof of Key Lemma. Vertex v is the first in $C_1 \cup C_2$ visited during the execution of DFS-Loop on G^{rev}.

3.3 The Final Argument

The Key Lemma says that traversing an arc from one SCC to another (in the original, unreversed graph) strictly increases the maximum f-value of the current SCC. For example, if f_i denotes the largest f-value of a vertex in C_i in Figure 6(a), then we must have $f_1 < f_2, f_3 < f_4$. Intuitively, when DFS-Loop is invoked

on G, processing vertices in decreasing order of finishing times, the successive calls to DFS peel off the SCCs of the graph one at a time, like layers of an onion.

We now formally prove correctness of our algorithm for computing strongly connected components. Consider the execution of DFS-Loop on G. We claim that whenever DFS is called on a vertex v, the vertices explored — and assigned a common leader — by this call are precisely those in v's SCC in G. Since DFS-Loop eventually explores every vertex, this claim implies that the SCCs of G are precisely the groups of vertices that are assigned a common leader.

We proceed by induction. Let S denote the vertices already explored by previous calls to DFS (initially empty). Inductively, the set S is the union of zero or more SCCs of G. Suppose DFS is called on a vertex v and let C denote v's SCC in G. Since the SCCs of a graph are disjoint, S is the union of SCCs of G, and $v \notin S$, no vertices of C lie in S. Thus, this call to DFS will explore, at the least, all vertices of C. By the Key Lemma, every outgoing arc (i, j) from C leads to some SCC C' that contains a vertex w with a finishing time larger than $f(v)$. Since vertices are processed in decreasing order of finishing time, w has already been explored and belongs to S; since S is the union of SCCs, it must contain all of C'. Summarizing, every outgoing arc from C leads directly to a vertex that has already been explored. Thus this call to DFS explores the vertices of C and nothing else. This completes the inductive step and the proof of correctness.

Lecture 7: Minimum Spanning Trees and Prim's Algorithm

CLRS Chapter 23

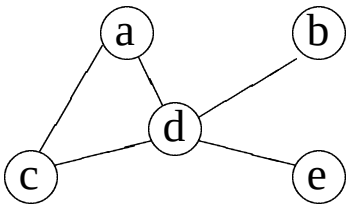
Outline of this Lecture

- Spanning trees and minimum spanning trees.
- The minimum spanning tree (MST) problem.
- The generic algorithm for MST problem.
- Prim's algorithm for the MST problem.
  - The algorithm
  - Correctness
  - Implementation + Running Time

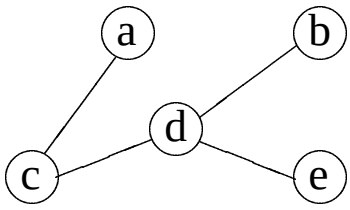
Spanning Trees

Spanning Trees: A subgraph T of a undirected graph $G = (V, E)$ is a spanning tree of G if it is a tree and contains every vertex of G.

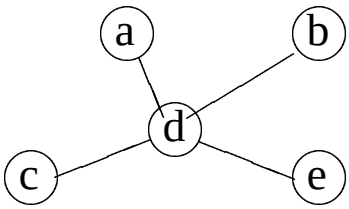
Example:



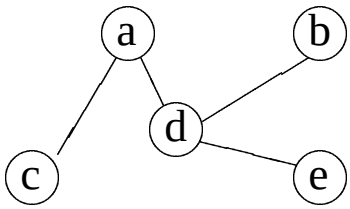
Graph



spanning tree 1



spanning tree 2



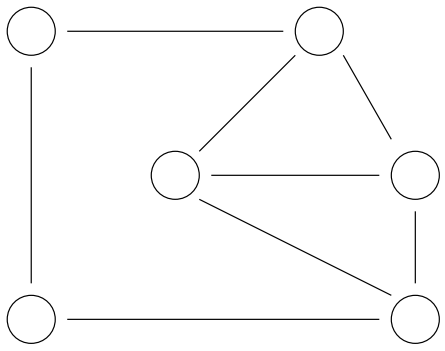
spanning tree 3

Spanning Trees

Theorem: Every connected graph has a spanning tree.

Question: Why is this true?

Question: Given a connected graph G, how can you find a spanning tree of G?

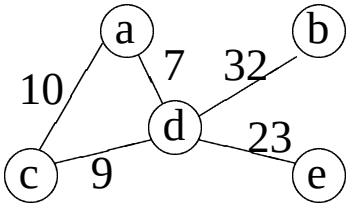


Weighted Graphs

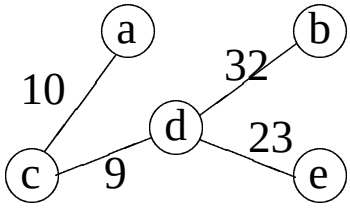
Weighted Graphs: A weighted graph is a graph, in which each edge has a weight (some real number).

Weight of a Graph: The sum of the weights of all edges.

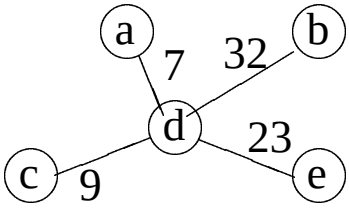
Example:



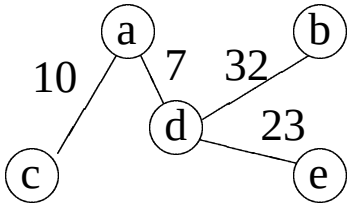
weighted graph



Tree 1. w=74



Tree 2, w=71



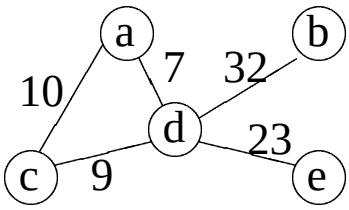
Tree 3, w=72

Minimum spanning tree

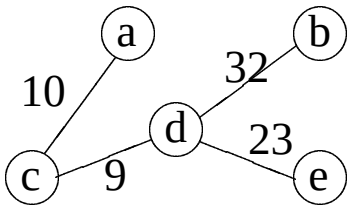
Minimum Spanning Trees

A **Minimum Spanning Tree** in an undirected connected weighted graph is a spanning tree of minimum weight (among all spanning trees).

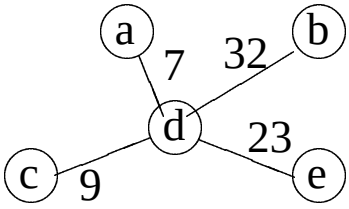
Example:



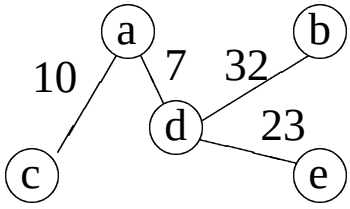
weighted graph



Tree 1. w=74



Tree 2, w=71



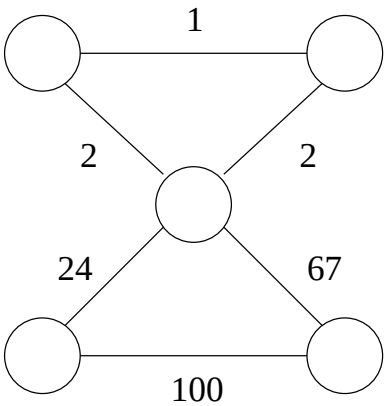
Tree 3, w=72

Minimum spanning tree

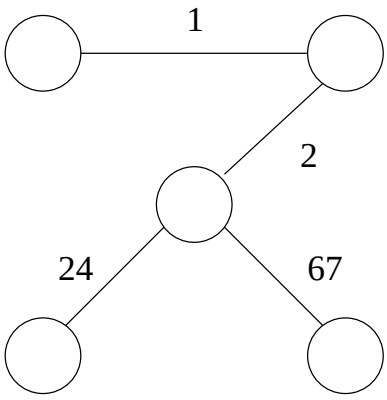
Minimum Spanning Trees

Remark: The minimum spanning tree may not be unique. However, if the weights of all the edges are pairwise distinct, it is indeed unique (we won't prove this now).

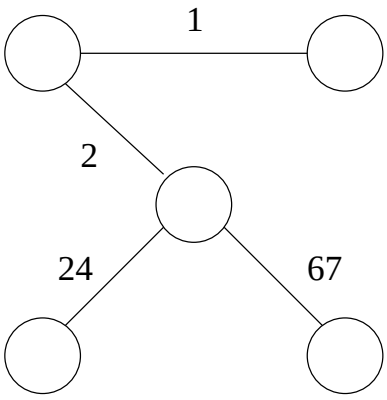
Example:



weighted graph



MST1



MST2

Minimum Spanning Tree Problem

MST Problem: Given a connected weighted undirected graph G, design an algorithm that outputs a minimum spanning tree (MST) of G.

Question: What is most intuitive way to solve?

Generic approach: A tree is an acyclic graph.
The idea is to start with an empty graph and try to add edges one at a time, always making sure that what is built remains acyclic. And if we are sure every time the resulting graph always is a subset of some minimum spanning tree, we are done.

Generic Algorithm for MST problem

Let A be a set of edges such that $A \subseteq T$, where T is a MST. An edge (u, v) is a *safe edge* for A, if $A \cup \{(u, v)\}$ is also a subset of some MST.

If at each step, we can find a safe edge (u, v), we can 'grow' a MST. This leads to the following generic approach:

```
Generic-MST(G, w)
 Let A=EMPTY;
 while A does not form a spanning tree
  find an edge (u, v) that is safe for A
  add (u, v) to A

 return A
```

How can we find a safe edge?

How to find a safe edge

We first give some definitions. Let $G = (V, E)$ be a connected and undirected graph. We define:

Cut A **cut** $(S, V - S)$ of G is a partition of V.

Cross An edge $(u, v) \in E$ **crosses** the cut $(S, V - S)$ if one of its endpoints is in S, and the other is in $V - S$.

Respect A cut **respects** a set A of edges if no edge in A crosses the cut.

Light edge An edge is a **light edge** crossing a cut if its weight is the minimum of any edge crossing the cut.

How to find a safe edge

Lemma

Let $G = (V, E)$ be a connected, undirected graph with a real-valued weight function w defined on E. Let A be a subset of E that is included in some minimum spanning tree for G, let $(S, V - S)$ be *any* cut of G that respects A, and let (u, v) be a light edge crossing the cut $(S, V - S)$. Then, edge (u, v) is safe for A.

It means that we can find a safe edge by

1. first finding a cut that respects A,

2. then finding the light edge crossing that cut.

That light edge is a safe edge.

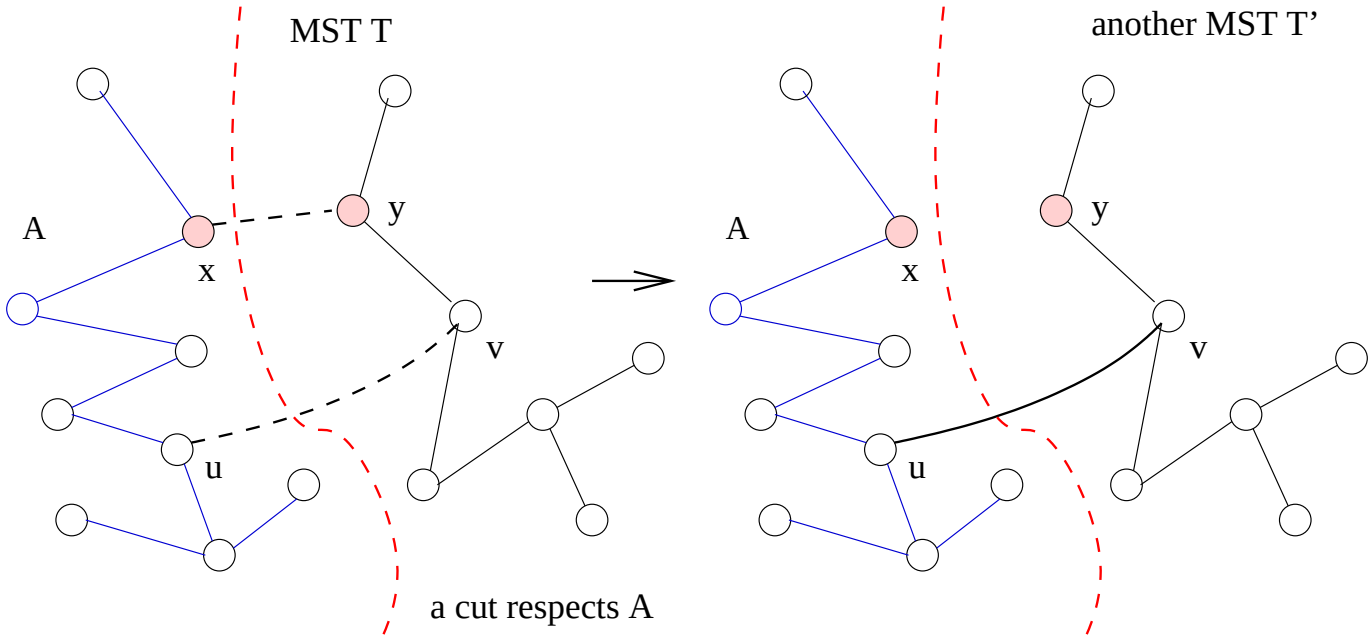
Proof

1. Let $A \subseteq T$, where T is a MST. Suppose $(u, v) \notin T$.

2. The trick is to construct *another* MST T' that contains both A and (u, v), thereby showing (u, v) is a safe edge for A.

3. Since u, and v are on opposite sides of the cut $(S, V - S)$, there is at least one edge in T on the path from u to v that *crosses* the cut. Let (x, y) be such edge. Since the cut respects A, $(x, y) \notin A$.

   Since (u, v) is a light edge crossing the cut, we have $w(x, y) \geq w(u, v)$.



4. Add (u, v) to T, it creates a cycle. By removing an edge from the cycle, it becomes a tree again. In particular, we remove (x, y) ($\notin A$) to make a new tree $T^{'}$.

5. The weight of $T^{'}$ is

$$\begin{aligned} w(T') &= w(T) - w(x, y) + w(u, v) \\ &\leq w(T) \end{aligned}$$

6. Since T is a MST, we must have $w(T) = w(T^{'})$, hence $T^{'}$ is also a MST.

7. Since $A \cup \{(u, v)\}$ is also a subset of $T^{'}$ (a MST), (u, v) is safe for A.

Prim's Algorithm

The generic algorithm gives us an idea how to 'grow' a MST.

If you read the theorem and the proof carefully, you will notice that the choice of a cut (and hence the corresponding light edge) in each iteration is immaterial. We can select *any cut* (that respects the selected edges) and find the light edge crossing that cut to proceed.

The *Prim's* algorithm makes a nature choice of the cut in each iteration – it grows a single tree and adds a light edge in each iteration.

Prim's Algorithm : How to grow a tree

Grow a Tree

- Start by picking any vertex r to be the root of the tree.

- While the tree does not contain
  all vertices in the graph
  find shortest edge leaving the tree
  and add it to the tree .

Running time is $O((|V| + |E|) \log |V|)$.

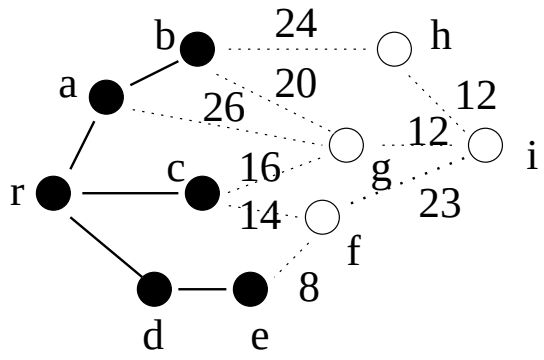
More Details

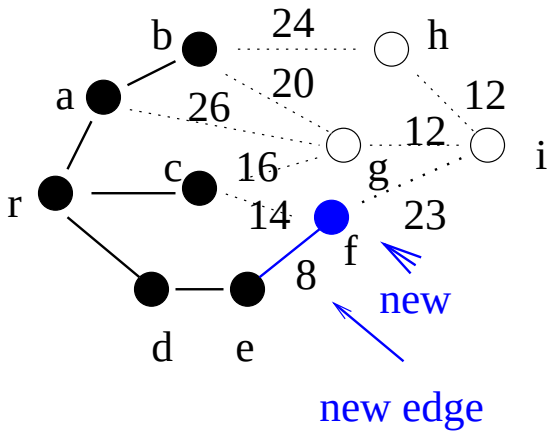
Step 0: Choose any element r; set $S = \{r\}$ and $A = \emptyset$. (Take r as the root of our spanning tree.)

Step 1: Find a lightest edge such that one endpoint is in S and the other is in $V \setminus S$. Add this edge to A and its (other) endpoint to S.

Step 2: If $V \setminus S = \emptyset$, then stop & output (minimum) spanning tree (S, A). Otherwise go to Step 1.

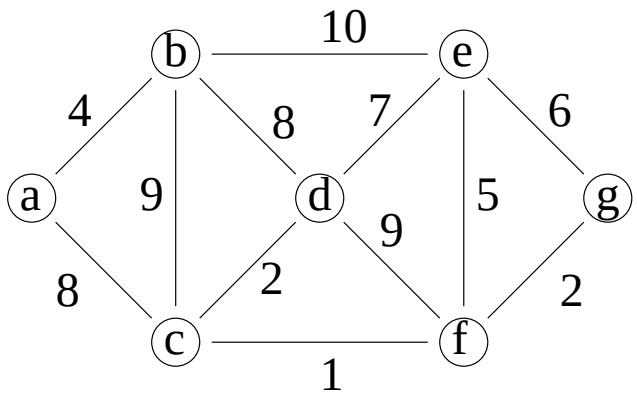
The idea: expand the current tree by adding the lightest (shortest) edge leaving it and its endpoint.



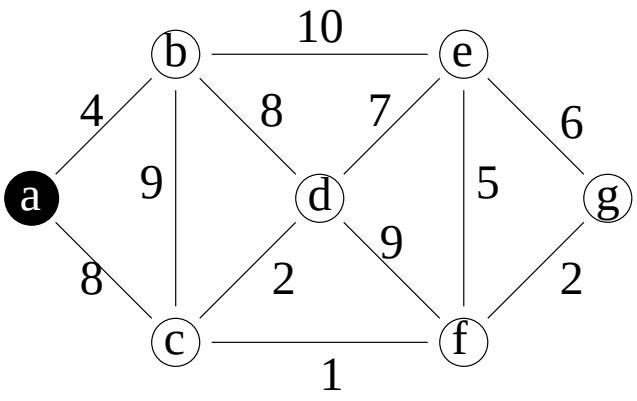


Prim's Algorithm

Worked Example



Connected graph



Step 0

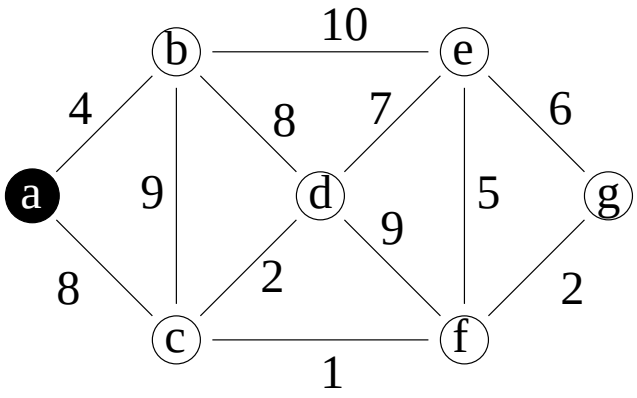
S={a}

V \ S = {b,c,d,e,f,g}

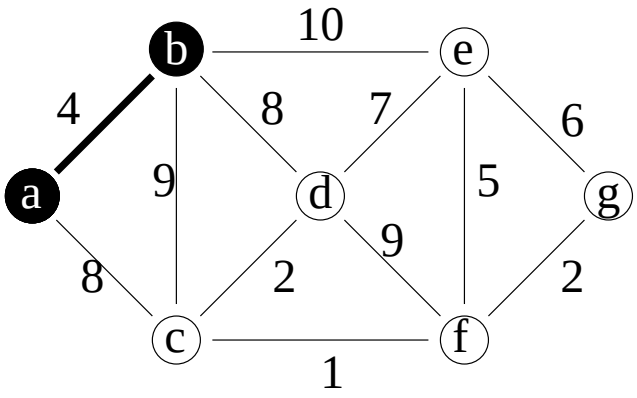
lightest edge = {a,b}

Prim's Algorithm

Prim's Example – Continued



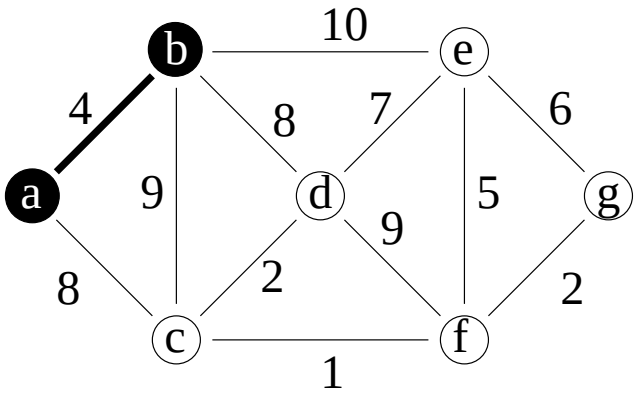
Step 1.1 before
S={a}
V \ S = {b,c,d,e,f,g}
A={}
lightest edge = {a,b}



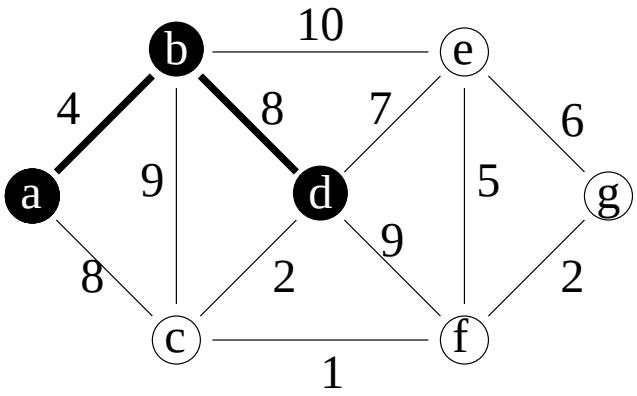
Step 1.1 after
S={a,b}
V \ S = {c,d,e,f,g}
A={{a,b}}
lightest edge = {b,d}, {a,c}

Prim's Algorithm

Prim's Example – Continued



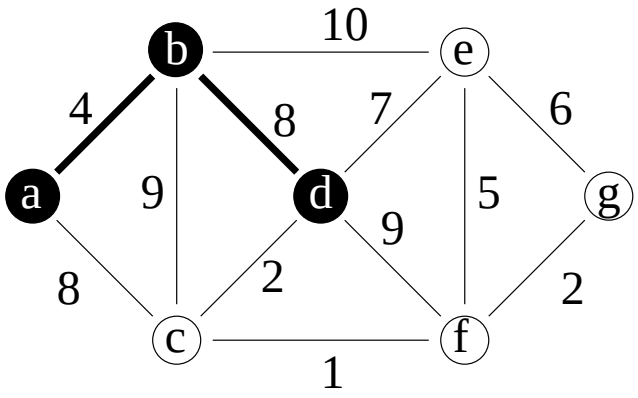
Step 1.2 before
S={a,b}
V \ S = {c,d,e,f,g}
A={{a,b}}
lightest edge = {b,d}, {a,c}



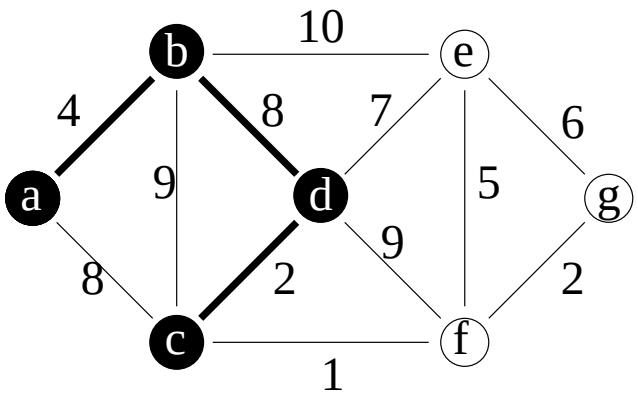
Step 1.2 after
S={a,b,d}
V \ S = {c,e,f,g}
A={{a,b},{b,d}}
lightest edge = {d,c}

Prim's Algorithm

Prim's Example – Continued



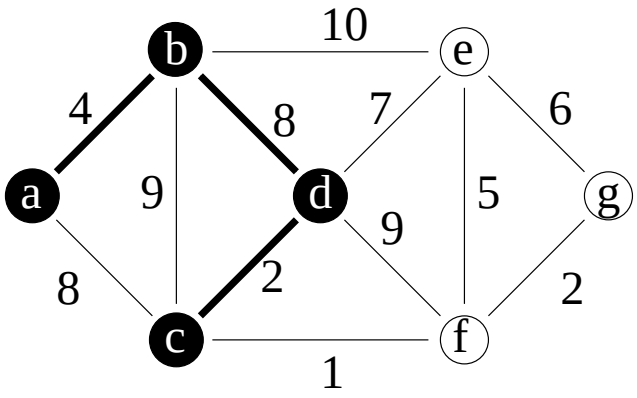
Step 1.3 before
S={a,b,d}
V \ S = {c,e,f,g}
A={{a,b},{b,d}}
lightest edge = {d,c}



Step 1.3 after
S={a,b,c,d}
V \ S = {e,f,g}
A={{a,b},{b,d},{c,d}}
lightest edge = {c,f}

Prim's Algorithm

Prim's Example – Continued



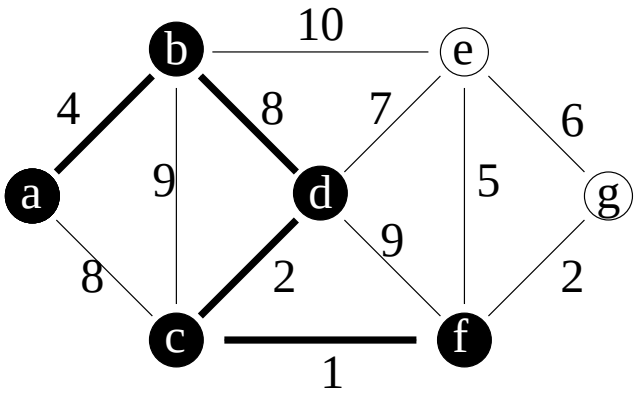
Step 1.4 before

S={a,b,c,d}

V \ S = {e,f,g}

A={{a,b},{b,d},{c,d}}

lightest edge = {c,f}



Step 1.4 after

S={a,b,c,d,f}

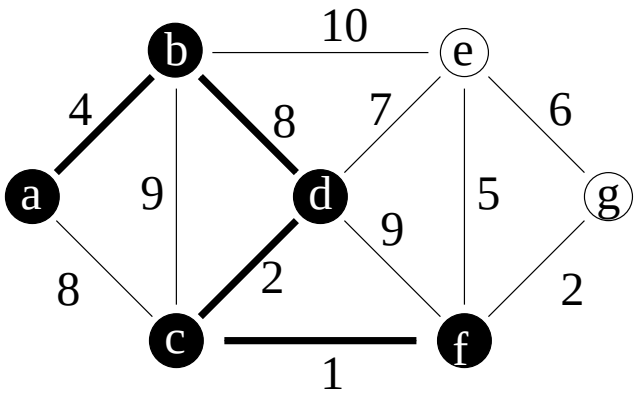
V \ S = {e,g}

A={{a,b},{b,d},{c,d},{c,f}}

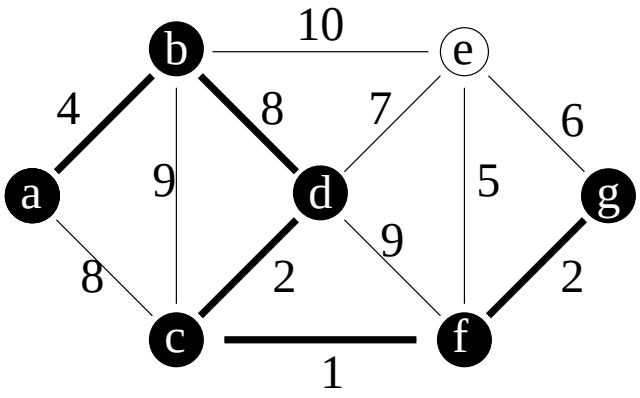
lightest edge = {f,g}

Prim's Algorithm

Prim's Example – Continued



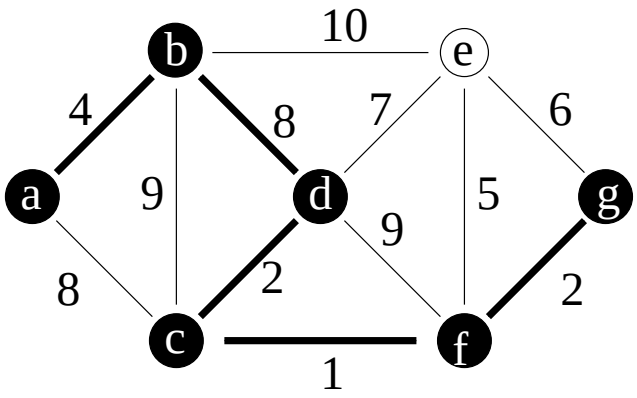
Step 1.5 before
S={a,b,c,d,f}
V \ S = {e,g}
A={{a,b},{b,d},{c,d},{c,f}}
lightest edge = {f,g}



Step 1.5 after
S={a,b,c,d,f,g}
V \ S = {e}
A={{a,b},{b,d},{c,d},{c,f},
{f,g}}
lightest edge = {f,e}

Prim's Algorithm

Prim's Example – Continued



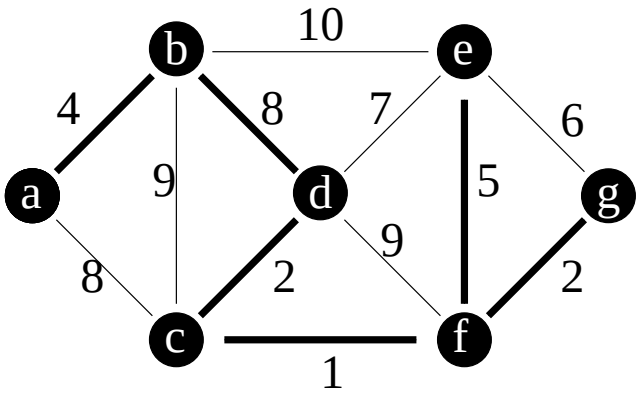
Step 1.6 before

S={a,b,c,d,f,g}

V \ S = {e}

A={{a,b},{b,d},{c,d},{c,f},
{f,g}}

lightest edge = {f,e}



Step 1.6 after

S={a,b,c,d,e,f,g}

V \ S = {}

A={{a,b},{b,d},{c,d},{c,f},
{f,g},{f,e}}

MST completed

Recall Idea of Prim's Algorithm

Step 0: Choose any element r and set $S = \{r\}$ and $A = \emptyset$. (Take r as the root of our spanning tree.)

Step 1: Find a lightest edge such that one endpoint is in S and the other is in $V \setminus S$. Add this edge to A and its (other) endpoint to S.

Step 2: If $V \setminus S = \emptyset$, then stop and output the minimum spanning tree (S, A).
Otherwise go to Step 1.

Questions:

- Why does this produce a **Minimum** Spanning Tree?
- How does the algorithm find the lightest edge and update A efficiently?
- How does the algorithm update S efficiently?

Prim's Algorithm

Question: How does the algorithm update S efficiently?

Answer: Color the vertices. Initially all are white. Change the color to black when the vertex is moved to S. Use color[v] to store color.

Question: How does the algorithm find the lightest edge and update A efficiently?

Answer:
(a) Use a priority queue to find the lightest edge.
(b) Use pred[v] to update A.

Reviewing Priority Queues

Priority Queue is a data structure (can be implemented as a heap) which supports the following operations:

insert(u, key):

Insert u with the key value key in Q.

u = extractMin():

Extract the item with the minimum key value in Q.

decreaseKey($u, new\text{-}key$):

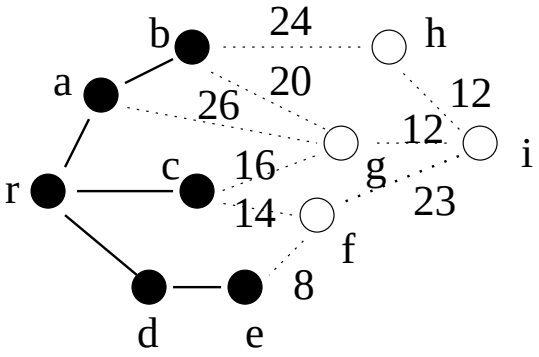
Decrease u's key value to $new\text{-}key$.

Remark: Priority Queues can be implemented so that each operation takes time $O(\log |Q|)$. See CLRS!

Using a Priority Queue to Find the Lightest Edge

Each item of the queue is a triple $(u, pred[u], key[u])$, where

- u is a vertex in $V \setminus S$,
- $key[u]$ is the weight of the lightest edge from u to any vertex in S, and
- $pred[u]$ is the endpoint of this edge in S. The array is used to build the MST tree.



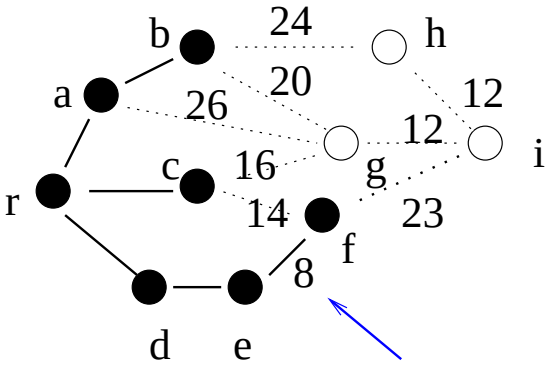
key[f] = 8, pred[f] = e

key[i] = infinity, pred[i] = nil

key[g] = 16, pred[g] = c

key[h] = 24, pred[h] = b

→ f has the minimum key



new edge

key[i] = 23, pred[i] = f

After adding the new edge and vertex f, update the key[v] and pred[v] for each vertex v adjacent to f

Description of Prim's Algorithm

Remark: G is given by adjacency lists. The vertices in $V \setminus S$ are stored in a priority queue with key=value of lightest edge to vertex in S.

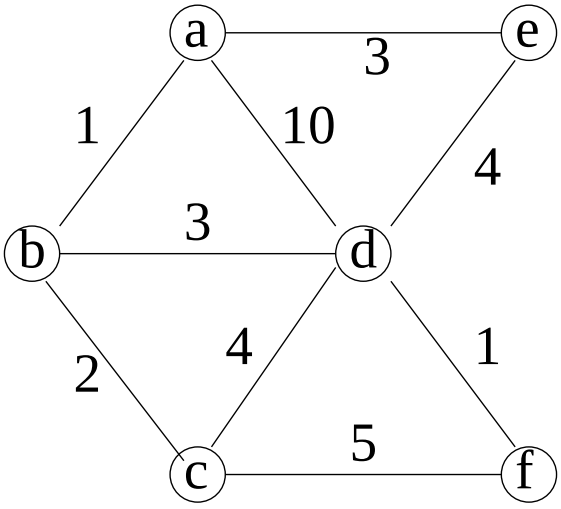
```
Prim(G, w, r)
{   for each u ∈ V                                  initialize
    {   key[u] = +∞;
        color[u] = W;
    }
    key[r] = 0;                                     start at root
    pred[r] = NIL;
    Q = new PriQueue(V);                            put vertices in Q
    while(Q is nonempty)                            until all vertices in MST
    {   u=Q.extraxtMin();                           lightest edge
        for each (v ∈ adj[u])
        {   if ((color[v] == W)&&(w[u, v] < key[v]))
            key[v] = w[u, v];                       new lightest edge
            Q.decreaseKey(v, key[v]);
            pred[v] = u;
        }
        color[u] = B;
    }
}
```

When the algorithm terminates, $Q = \emptyset$ and the MST is

$$T = \{\{v, pred[v]\} : v \in V \setminus \{r\}\}.$$

The pred pointers defi ne the MST as an inverted tree rooted at r.

Example for Running Prim's Algorithm



u	a	b	c	d	e	f
key[u]						
pred[u]						

Analysis of Prim's Algorithm

Let $n = |V|$ and $e = |E|$. The data structure PriQueue supports the following two operations: (See CLRS)

- $O(\log n)$ to extract each vertex from the queue.
  Done once for each vertex $= O(n \log n)$.

- $O(\log n)$ time to decrease the key value of neighboring vertex.
  Done at most once for each edge $= O(e \log n)$.

Total cost is then

$$O((n + e) \log n)$$

Analysis of Prim's Algorithm – Continued

```
Prim(G, w, r)  {
   for each (u in V)                                   2n
   {
      key[u] = +infinity;
      color[u] = white;
   }

   key[r] = 0;                                          1
   pred[r] = nil;                                       1
   Q = new PriQueue(V);                                 n

   while (Q. nonempty())                                1
   {
      u = Q.extractMin();                               O(log n)
      for each (v in adj[u])
      {
         if ((color[v] == white) &                      1
             (w(u,v) < key[v])                          1   O(deg(u) log n)
         {
            key[v] = w(u, v);                           1
            Q.decreaseKey(v, key[v]);                   O(log n)
            pred[v] = u;                                1
         }
      }
      color[u] = black;                                 1
   }
}
```

$$\sum_{u \text{ in } V} [O(\log n) + O(\deg(u) \log n)]$$

Analysis of Prim's Algorithm – Continued

So the overall running time is

$$
\begin{aligned}
& T(n, e) \\
&= 3n + 2 + \sum_{u \in V} [O(\log n) + O(\deg(u) \log n)] \\
&= 3n + 2 + O\left[(\log n) \sum_{u \in V} (1 + \deg(u))\right] \\
&= 3n + 2 + O[(\log n)(n + 2e)] \\
&= O[(\log n)(n + 2e)] \\
&= O[(\log n)(n + e)] \\
&= O[(|V| + |E|) \log |V|].
\end{aligned}
$$

Lecture 8: Kruskal's MST Algorithm

CLRS Chapter 23

Main Topics of This Lecture

- Kruskal's algorithm
  Another, but different, greedy MST algorithm

- Introduction to UNION-FIND data structure.
  Used in Kruskal's algorithm
  Will see implementation in next lecture.

Idea of Kruskal's Algorithm

The Kruskal's Algorithm is based directly on the generic algorithm. Unlike Prim's algorithm, we make a different choices of cuts.

Initially, trees of the forest are the vertices (no edges).

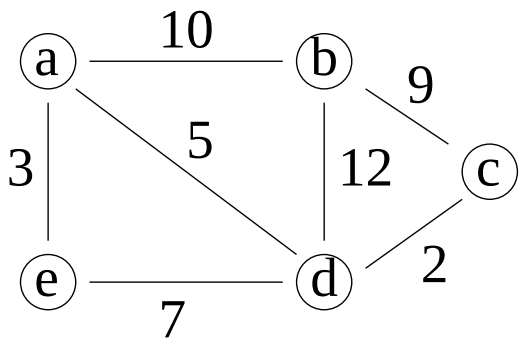
In each step add the cheapest edge that does not create a cycle.

Observe that unlike Prim's algorithm, which only grows one tree, Kruskal's algorithm grows a collection of trees (a forest).

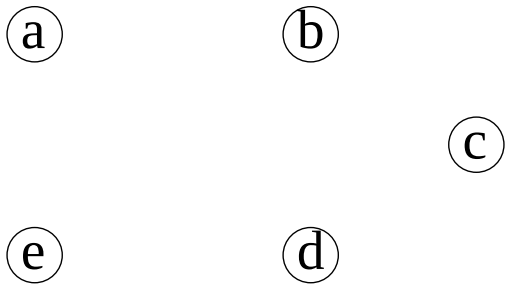
Continue until the forest 'merge to' a single tree.
(Why is a single tree created?)

This is a *minimum* spanning tree
(we must prove this).

Outline by Example



original graph



forest ⟶ MST

E

edge	weight
{d, c}	2
{a, e}	3
{a, d}	5
{e, d}	7
{b, c}	9
{a, b}	10
{b, d}	12

Forest (V, A)

A={ }

Outline of Kruskal's Algorithm

Step 0: Set $A = \emptyset$ and $F = E$, the set of all edges.

Step 1: Choose an edge e in F of minimum weight, and check whether adding e to A creates a cycle.

- If "yes", remove e from F.
- If "no", move e from F to A.

Step 2: If $F = \emptyset$, stop and output the minimal spanning tree (V, A). Otherwise go to Step 1.

Remark: Will see later, after each step, (V, A) is a subgraph of a MST.

Outline of Kruskal's Algorithm

Implementation Questions:

- How does algorithm choose edge $e \in F$ with minimum weight?

- How does algorithm check whether adding e to A creates a cycle?

How to Choose the Edge of Least Weight

Question:
How does algorithm choose edge $e \in F$ with minimum weight?

Answer: Start by sorting edges in E in order of increasing weight.
Walk through the edges in this order.
(Once edge e causes a cycle it will always cause a cycle so it can be thrown away.)

How to Check for Cycles

Observation: At each step of the outlined algorithm, (V, A) is acyclic so it is a forest.

If u and v are in the same tree, then adding edge $\{u, v\}$ to A creates a cycle.

If u and v are not in the same tree, then adding edge $\{u, v\}$ to A does not create a cycle.

Question: How to test whether u and v are in the same tree?

High-Level Answer: Use a disjoint-set data structure
Vertices in a tree are considered to be in same set.
Test if Find-Set(u) = Find-Set(v)?

Low -Level Answer:
The UNION-FIND data structure implements this:

The UNION-FIND Data Structure

UNION-FIND supports three operations on collections of **disjoint sets**: Let n be the size of the universe.

Create-Set(u): $O(1)$

Create a set containing the single element u.

Find-Set(u): $O(\log n)$

Find the set containing the element u.

Union(u, v): $O(\log n)$

Merge the sets respectively containing u and v into a common set.

For now we treat UNION-FIND as a black box.
Will see implementation in next lecture.

Kruskal's Algorithm: the Details

Sort E in increasing order by weight w; $O(|E| \log |E|)$

/* After sorting $E = \langle \{u_1, v_1\}, \{u_2, v_2\}, \ldots, \{u_{|E|}, v_{|E|}\} \rangle$ */

$A = \{\}$;

for (each u in V) CREATE-SET(u); $O(|V|)$

for $e_i = (u_i, v_i)$ from 1 to $|E|$ do $O(|E| \log |V|)$

    if (FIND-SET(u_i) != FIND-SET(v_i))

    { add $\{u_i, v_i\}$ to A;

      UNION(u_i, v_i);

    }

return(A);

Remark: With a proper implementation of UNION-FIND, Kruskal's algorithm has running time $O(|E| \log |E|)$.

Why Kruskal's Algorithm is correct?

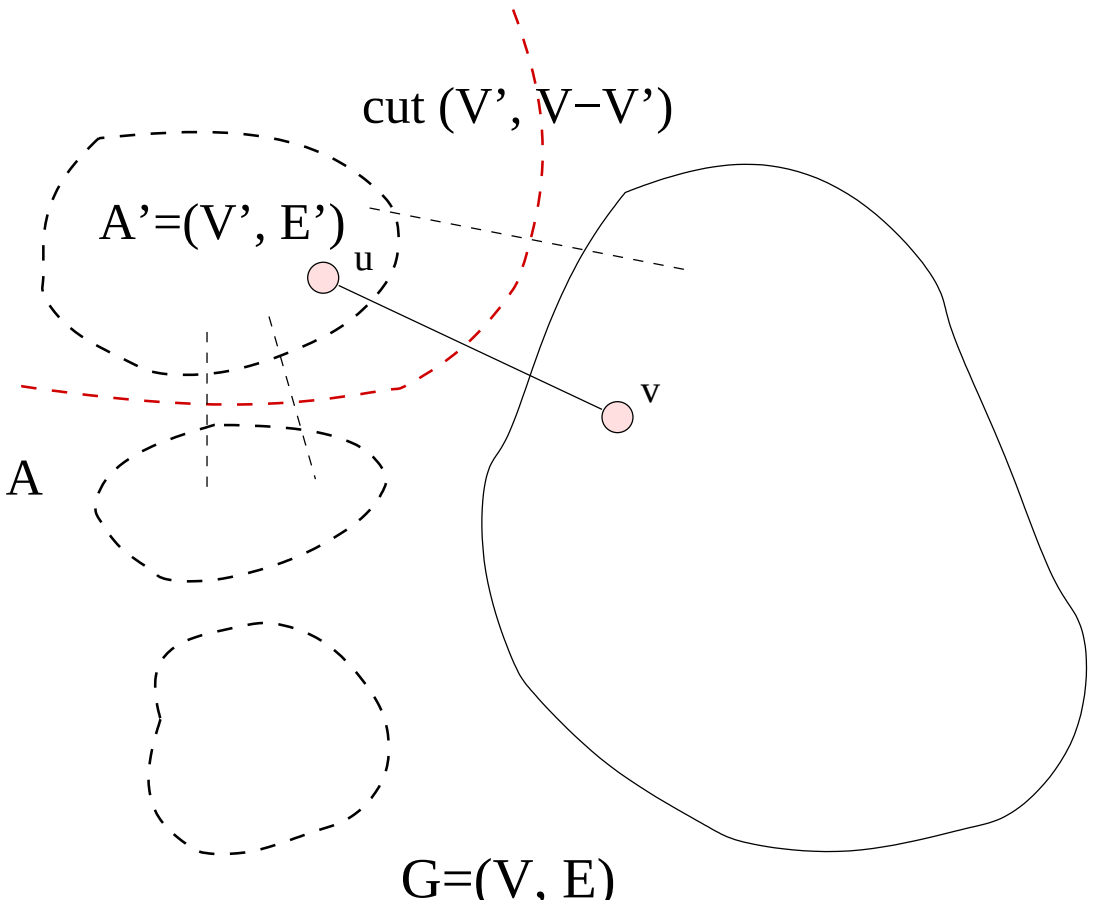
Let A be the edge set which has been selected by Kruskal's Algorithm, and (u, v) be the edge to be added next. It suffices to show there is a cut which respects A, and (u, v) is the light edge crossing that cut.

1. Let $A' = (V', E')$ denote the tree of the forest A that contains u. Consider the cut $(V', V - V')$.

2. Observe that there is no edge in A crosses this cut, so the cut respects A.

3. Since adding (u, v) to A' does not induce a cycle, (u, v) crosses the cut. Moreover, since (u, v) is currently the smallest edge, (u, v) is the light edge crossing the cut. This completes the correctness proof of Kruskal's Algorithm.

Why Kruskal's Algorithm is correct?

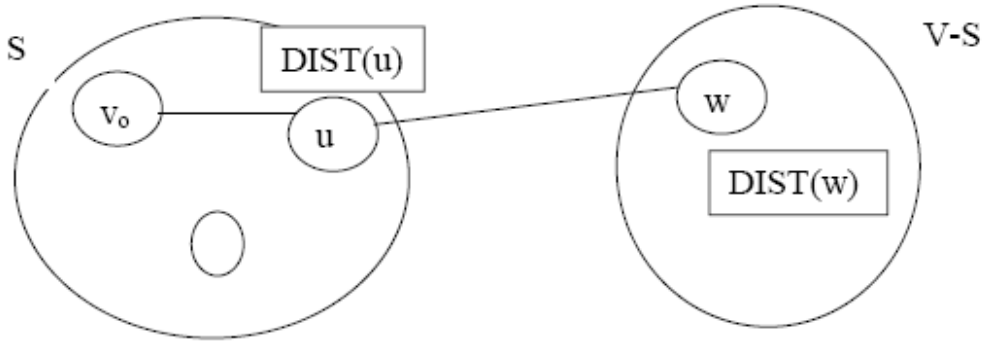


Single Source Shortest Paths

- Given a weighted graph G= (V,E) where the weights are >0.
- A source vertex, v_o belong to V.
- Find the shortest path from v_o to all other nodes in G.
- Shortest paths are generated in increasing order: 1,2,3,… ..

Dijkstra Algorithm

- S: Set of vertices (including v_o) whose final shortest paths from the source v_o have already been determined.
- For each node w∈V-S,
  Dist (w): the length of the shortest path starting from v_o going through only vertices which are in S and ending at w.
- The next path is generated as follows:
  It's the path of a vertex u which has Dist (u) minimum among all vertices in V-S
  Put u in S.
- Dist (w) for w in V-S may be decreased going though u.

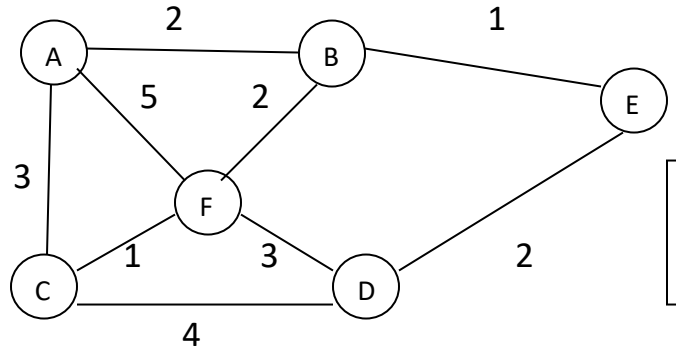


Compare Dist (u)+ cost (u,w) with Dist (w).

Algorithm:

```
Procedure SSSP (v_o, cost, n)
Array S (1:n);
        Begin
        /* initialization*/
                For i=1 to n do
                S(i)=0, Dist (i)= cost (v_o ,i)
                End for.
                S(v_o)=1, Dist (v_o)=o;
                For i=1 to n-1 do.
                        Choose u s.t. Dist (u)= min {Dist (w) } & S(w)=0
                        S(u)=1;
                        For all w with S(w)=0 do.
                                Dist (w)= min (Dist (w), Dist (u) +Cost (u,w))
                        End for.
                end for.
        end.
```

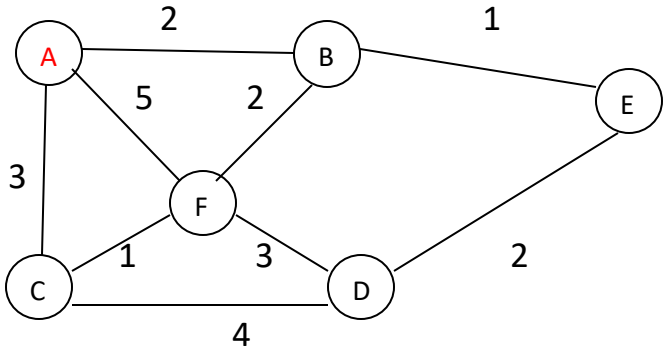
Example:

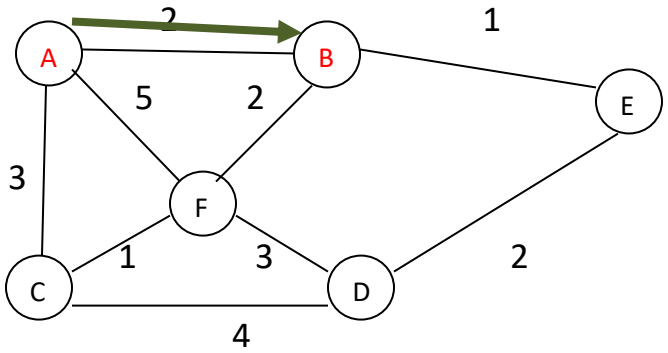


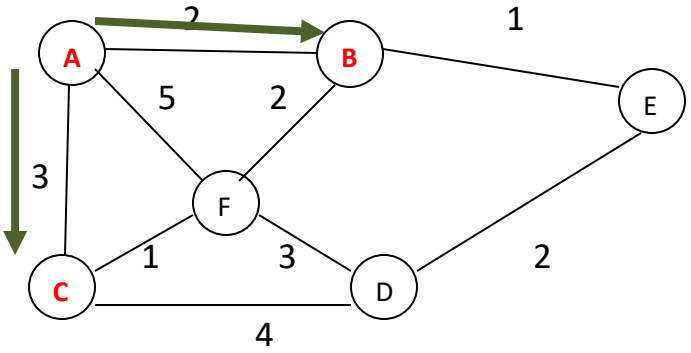
This will change to 5 to 4 because
Min{(A-F), (A-B-F)}
Min{5,4}

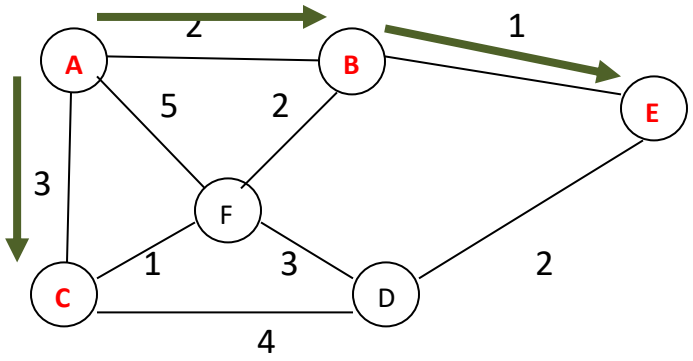
Iteration	N	D_B	D_C	D_D	D_E	D_F
Initial	{A}	2	3	∞	∞	5
1	{A,B}	2	3	∞	3	4
2	{A,B, C}	2	3	7(A-C-D)	3	4(no change because of same cost A-C-F)
3	{A,B, C,E}	2	3	5(min{(A-C-D,A-	3	4

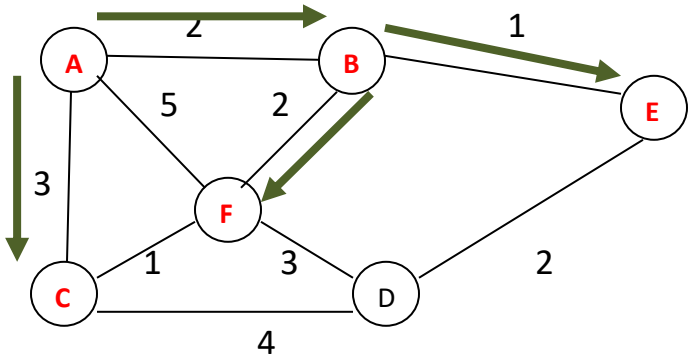
				B-E-D)}		
4	{ A,B, C,E,F}	2	3	5 no change	3	4
5	{ A,B, C,E,F,D}	2	3	5	3	4

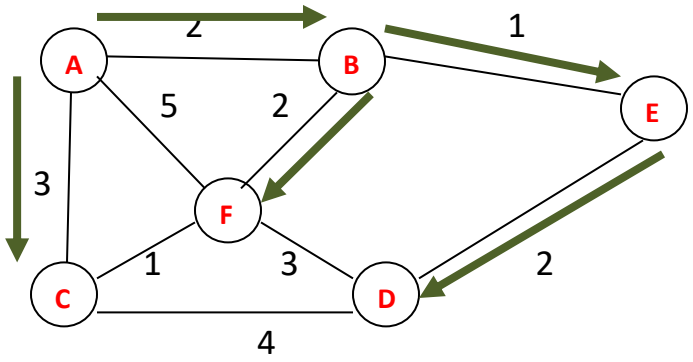












Implementation using min heap

- Build heap---------- O(v)
- Extracting min element from min-heap & Adjusting min heap v times-- v $\log_2$ v
- Decrease key operation:
    - Delete min key from heap---- O(1).

- Adjust root ------$\log_2 v$
- We have to perform decrease key operation on rest of the vertices at max. When the value change from infinite, we have adjust min heap which takes $\log_2 v$ (v time) So v $\log_2 v$. At max we have perform this decrease key operation v-1 times so decrease key operation v-1 times take $v^2 \log_2 v$
- $v^2 \log_2 v$ we can write it as **$e\log_2 v$** because e= v^2 in dense graph worst case.

Time complexity: O(v) + $v \log_2 v$ + $e \log_2 v$

Build heap

Extracting min element from min-heap & Adjusting min heap v times

Time complexity: $O(v^2)$ when adjacency matrix if the input is represented using adjacency list it can be reduced to **O((e+v) log v)** with the help of **binary heap.**

Drawback:

Dijkstra Algorithm will fail when there is negative weight cycle in the graph.

Shortest Paths

Dijkstra Bellman-Ford Floyd All-pairs paths

Lecturer: Georgy Gimel'farb

COMPSCI 220 Algorithms and Data Structures

Paths and Distances Revisited

Cost of a walk / path $v_0, v_1, \ldots, v_k$ in a digraph $G = (V, E)$ with edge weights $\{c(u, v) \mid (u, v) \in E\}$:

$$\text{cost}(v_0, v_1, \ldots, v_k) = \sum_{i=0}^{k-1} c(v_i, v_{i+1})$$

Distance $d(u, v)$ between two vertices u and v of $V(G)$: the minimum cost of a path between u and v.

Eccentricity of a node $u \in V$: $\text{ec}[u] = \max_{v \in V} d(u, v)$.

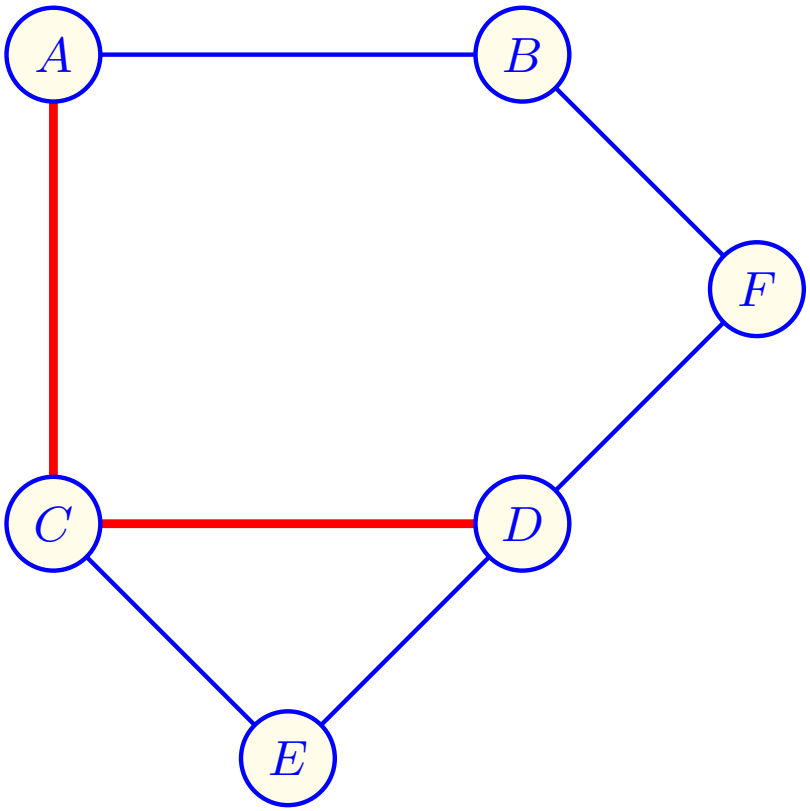
Radius of G: the minimum eccentricity of $u \in V$: $\min_{u \in V} \text{ec}[u]$.

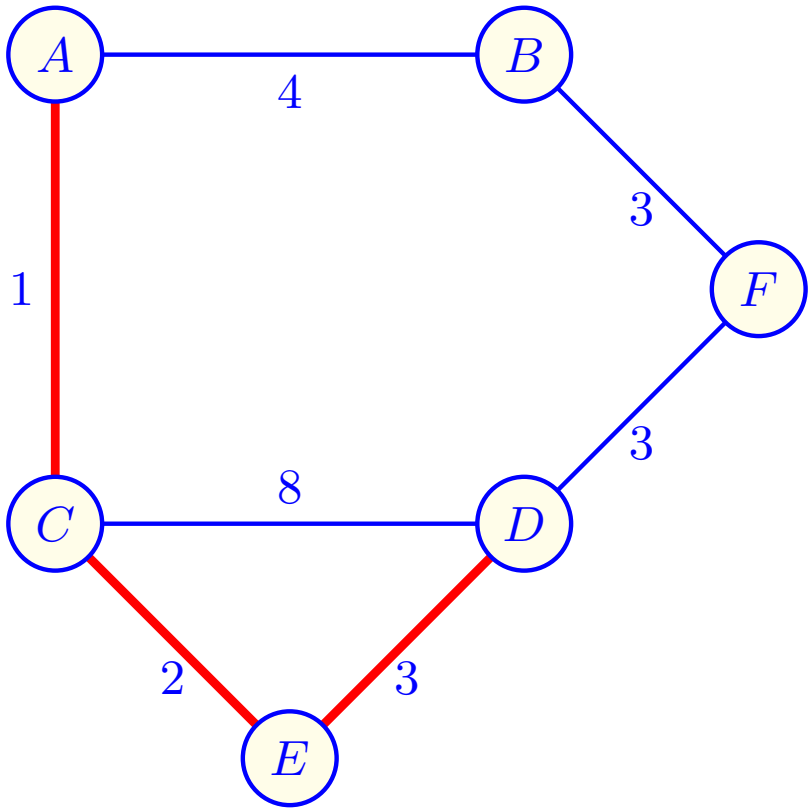
Diameter of G: the maximum eccentricity of $u \in V$: $\max_{u \in V} \text{ec}[u]$.

Note: there are analogous definitions for graphs.

Unweighted / Weighted Graphs: Shortest Paths

The shortest path from the vertex A to the vertex D:





$\min\{2_{A,C,D}, 3_{A,C,E,D}, 3_{A,B,F,D}\}$

$\min\{9_{A,C,D}, 6_{A,C,E,D}, 10_{A,B,F,D}\}$

Single-source Shortest Path (SSSP) in $G = (V, E, c)$

Given a source node v, find the shortest (minimum weight) path to each other node.

- Weight of a path: the sum of weights (costs) on the arcs.
- BFS works only if all weights $c(u, v)$; $(u, v) \in E$, are equal.
- **Dijkstra's algorithm** – one of the known solutions.
  - A **greedy** algorithm: each locally best choice is globally best.
  - Works only if all weights are non-negative.
  - Initial paths: one-arc paths from s to v of weight $\mathrm{cost}(s, v)$.
  - Each step compares the shortest paths with and without each new node.

Single-source Shortest Path (SSSP) in $G = (V, E, c)$

1. Build a list S of visited nodes (say, using a priority queue).
2. Iterative propagation of the shortest paths:
   1. Choose the closest unvisited node u being on a path with internal nodes in S.
   2. If adding the node u has established shorter paths, update distances of remaining unvisited nodes v from the source s.

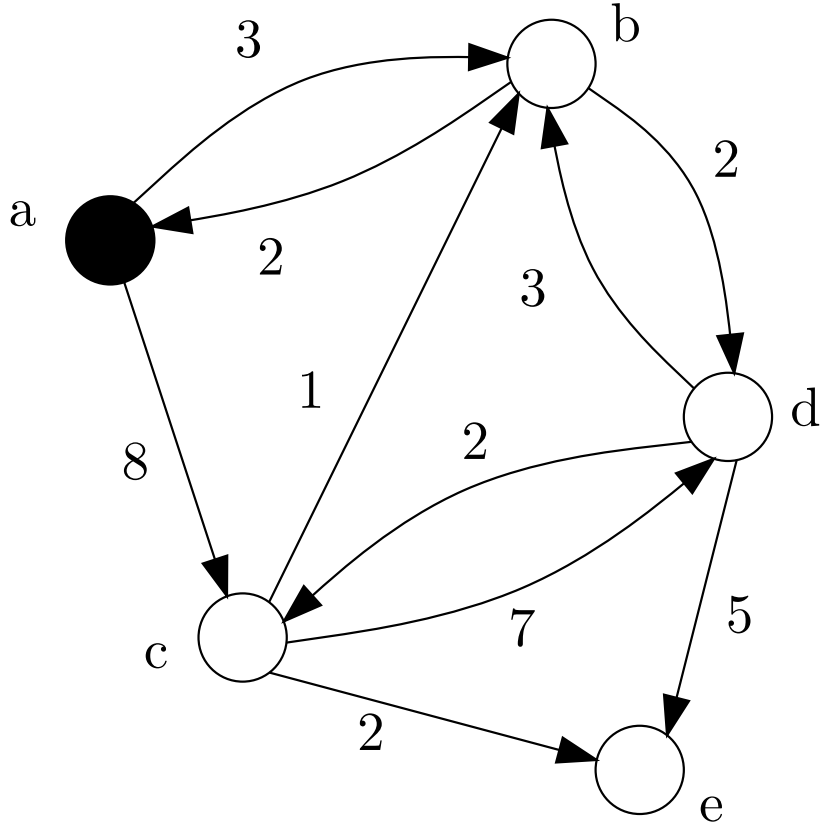
Complexity depends on data structures used.

- For a priority queue, such as a binary heap, running time $\mathrm{O}((m+n)\log n)$ is possible.
  - If every node is reachable from the source: $\mathrm{O}(m \log n)$.
- More sophisticated Fibonacci heaps lead to the best complexity of $\mathrm{O}(m + n \log n)$.

Dijkstra's Algorithm

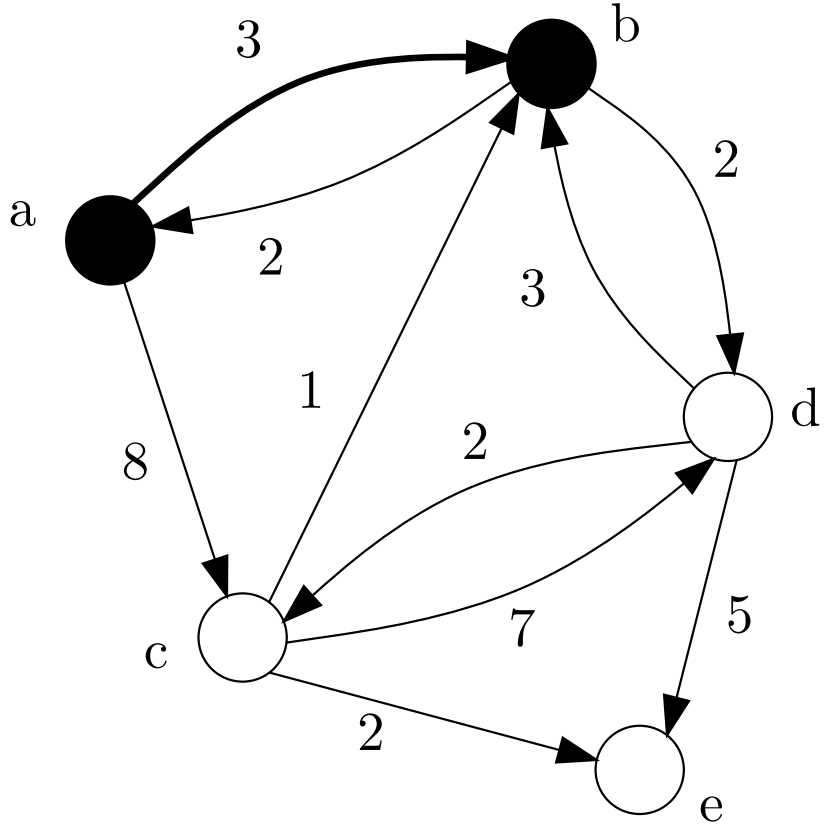
algorithm `Dijkstra`(weighted digraph (G, c), node $s \in V(G)$)
    array $colour[n] = \{\text{WHITE}, \ldots, \text{WHITE}\}$
    array $dist[n] = \{c[s, 0], \ldots, c[s, n-1]\}$
    $colour[s] \leftarrow$ BLACK
    while there is a WHITE node **do**
        pick a WHITE node u, such that $dist[u]$ is minimum
        $colour[u] \leftarrow$ BLACK
        for each x adjacent to u **do**
            if $colour[x] =$ WHITE **then**
                $dist[x] \leftarrow \min\{dist[x], dist[u] + c[u, x]\}$
            end if
        end for
    end while
    return $dist$
end

Dijkstra's Algorithm: Example 1



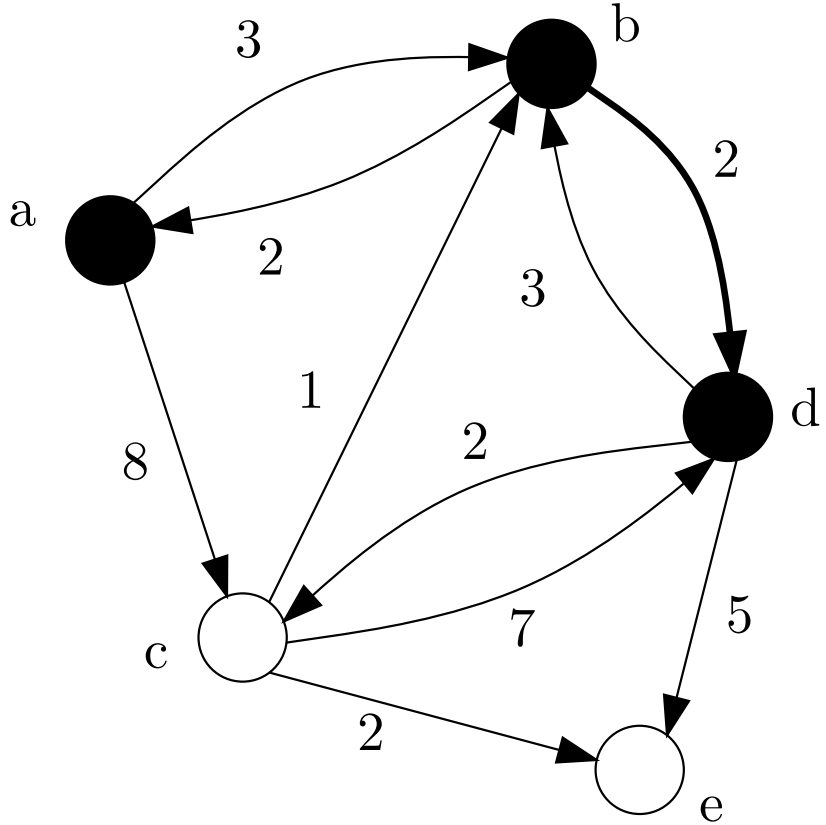
BLACK List S	$dist[x]$				
	a	b	c	d	e
a	0	3	8	∞	∞
$a\ b$	0	3	8	5	∞
$a\ b\ d$	0	3	7	5	10
$a\ b\ c\ d$	0	3	7	5	9
$a\ b\ c\ d\ e$	0	3	7	5	9

Dijkstra's Algorithm: Example 1



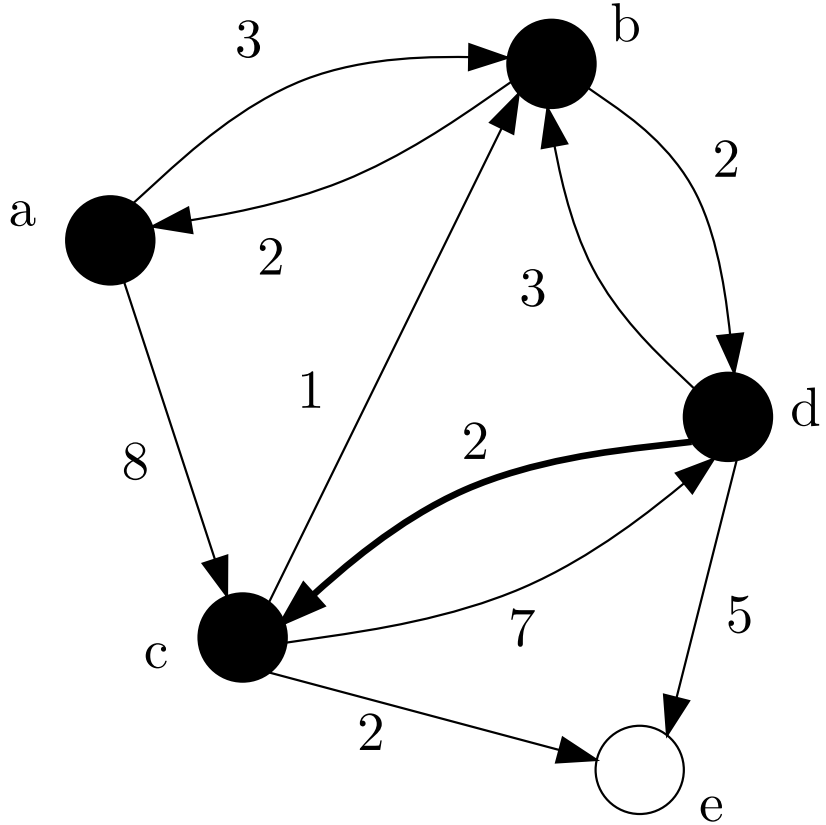
BLACK List S	$dist[x]$				
	a	b	c	d	e
a	0	3	8	∞	∞
$a\ b$	0	3	8	5	∞
$a\ b\ d$	0	3	7	5	10
$a\ b\ c\ d$	0	3	7	5	9
$a\ b\ c\ d\ e$	0	3	7	5	9

Dijkstra's Algorithm: Example 1



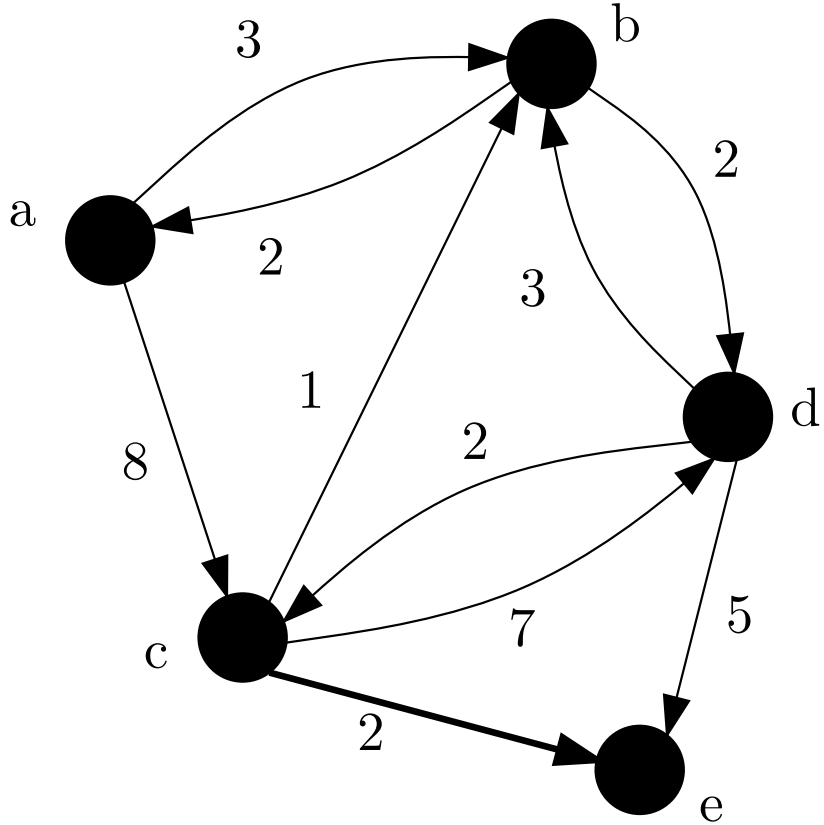
BLACK List S	$dist[x]$				
	a	b	c	d	e
a	0	3	8	∞	∞
$a\ b$	0	3	8	5	∞
$a\ b\ d$	0	3	7	5	10
$a\ b\ c\ d$	0	3	7	5	9
$a\ b\ c\ d\ e$	0	3	7	5	9

Dijkstra's Algorithm: Example 1



BLACK List S	$dist[x]$				
	a	b	c	d	e
a	0	3	8	∞	∞
$a\ b$	0	3	8	5	∞
$a\ b\ d$	0	3	7	5	10
$a\ b\ c\ d$	0	3	7	5	9
$a\ b\ c\ d\ e$	0	3	7	5	9

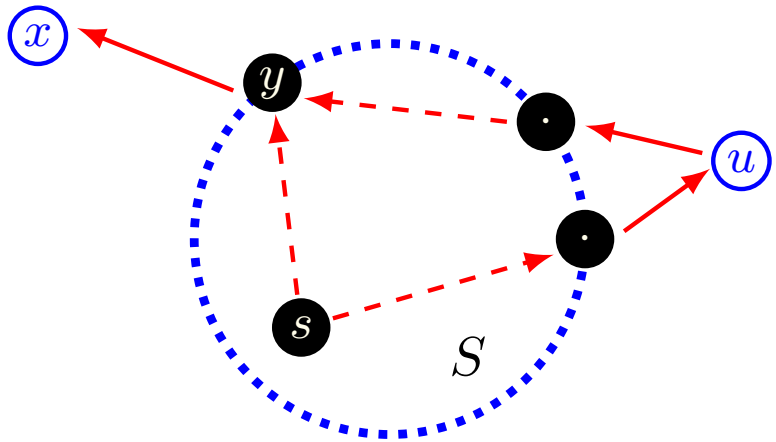
Dijkstra's Algorithm: Example 1



BLACK List S	$dist[x]$				
	a	b	c	d	e
a	0	3	8	∞	∞
$a\ b$	0	3	8	5	∞
$a\ b\ d$	0	3	7	5	10
$a\ b\ c\ d$	0	3	7	5	9
$a\ b\ c\ d\ e$	0	3	7	5	9

Why Does Dijkstra's Algorithm Work?

Let an S-**path** be a path starting at node s and ending at node x with all the intermediate nodes coloured BLACK, i.e., from the list S, except possibly x.



Theorem 6.8: Suppose that all arc weights are nonnegative.

Then these two properties hold at the top of **while**-loop:

P1: If $x \in V(G)$, then $dist[x]$ is the minimum cost of an S-path from s to x.

P2: If $colour[w] =$ BLACK (i.e., $w \in S$), then $dist[w]$ is the minimum cost of a path from s to w.

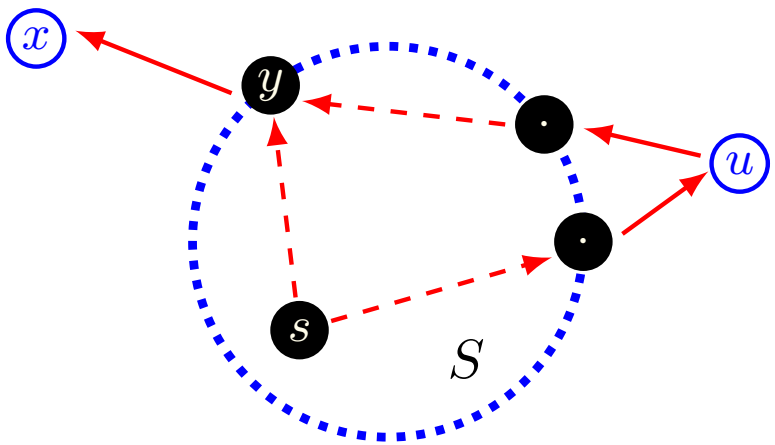
Once a node u is added to S and $dist[u]$ is updated, $dist[u]$ never changes in subsequent steps. After $S = V$, $dist$ holds the goal shortest distances.

Proving Why Dijkstra's Algorithm Works

The update rule: $dist[x] \leftarrow \min\{dist[x],\ dist[u] + c[u,x]\}$.

$dist[x]$ is the length of some path from s to x at every step.

- If $x \in S$, then it is an S-path.
- Updated $dist[v]$ never increases.



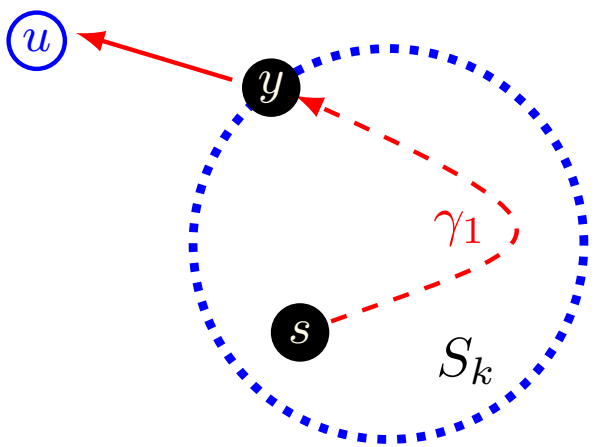
To prove P1 and P2: induction on the number of times k of going through the while-loop (S_k; $S_0 = \{s\}$; $dist[s] = 0$).

- $k = 0$: P1 and P2 hold as $dist[s] = 0$.
- Inductive hypothesis: P1 and P2 hold for $k \geq 0$; $S_{k+1} = S_k \bigcup \{u\}$.
- Inductive steps for P2 and P1:
  - Consider any s-to-w S_{k+1}-path $\gamma = (s, \ldots, y, u)$ of the weight $|\gamma|$.
  - If $w \in S_k$, consider the hypothesis.
  - If $w \notin S_k$, γ extends some s-to-y S_k-path $\gamma_1 = (s, \ldots, y)$.

Proving Why Dijkstra's Algorithm Works

Inductive step for P2:

- For $w \in S_{k+1}$ and $w \neq u$, P2 holds by inductive hypothesis.
- For $w = u$, P2 holds, too, because any S_{k+1}-path $\gamma = (s, \ldots, y, u)$ of weight $|\gamma|$ extends some S_k-path $\gamma_1 = (s, \ldots, y)$ of weight $|\gamma_1|$:
  - By the inductive hypothesis, $dist[y] \leq |\gamma_1|$.
  - By the update rule, $dist[u] \leq dist[y] + c(y, u)$.
  - Therefore, $dist[u] \leq |\gamma| = |\gamma_1| + c(y, u)$.



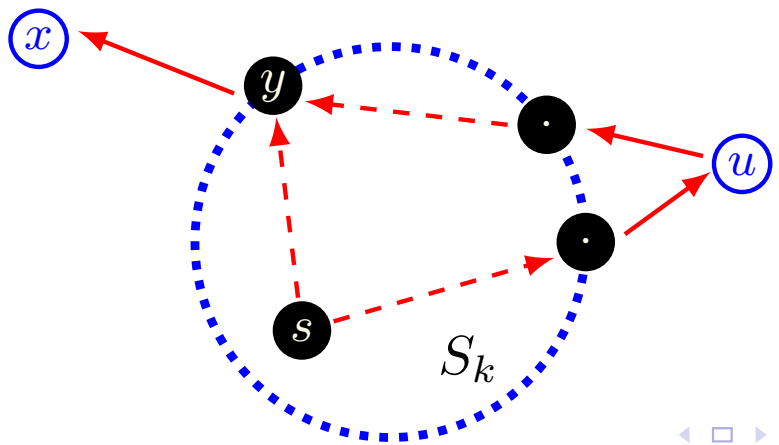
Proving Why Dijkstra's Algorithm Works

Inductive step for P1: $x \in V(G)$; γ – any s-to-x S_{k+1}-path; $S_{k+1} = S_k \bigcup \{u\}$:

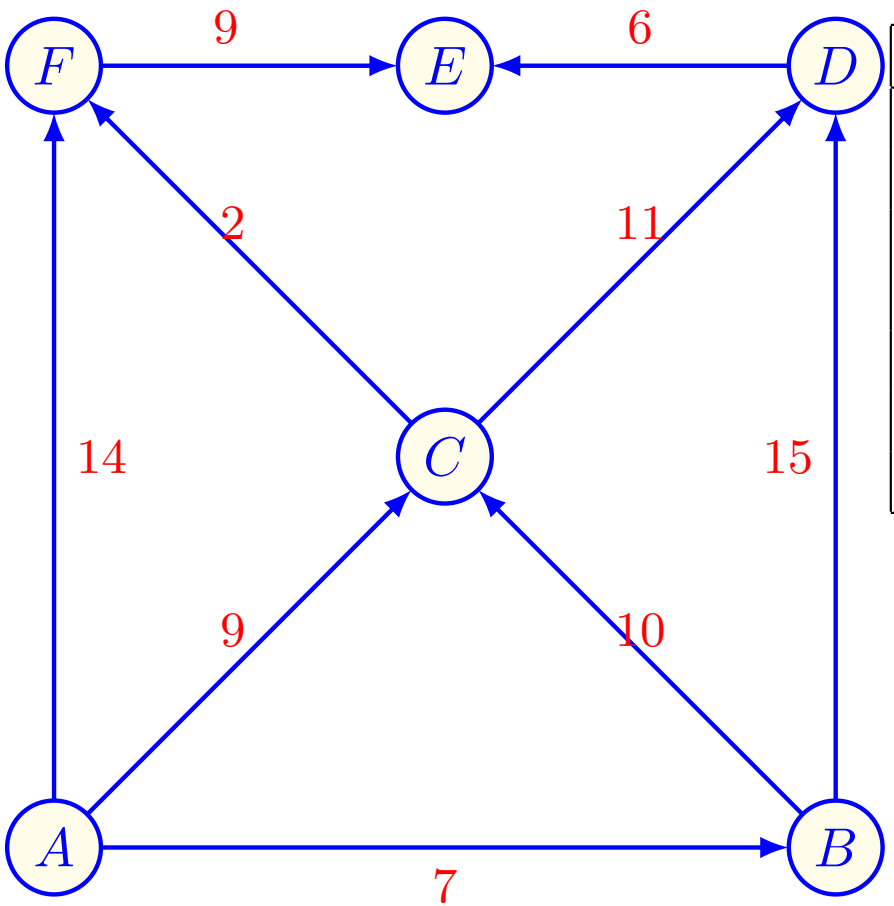
- $u \notin \gamma$: γ is an S_k-path and $|\gamma| \leq dist[x]$ by the inductive hypothesis.
- $u \in \gamma = (\overbrace{s, \ldots, u}^{\gamma_1}, x)$: by the update rule, $|\gamma| = |\gamma_1| + c(u, x) \geq dist[x]$.
- $u \in \gamma = (\overbrace{s, \ldots, u, \ldots . y}^{\gamma_1}, x)$, returning to S_k after u: by the update rule,

$$|\gamma| = |\gamma_1| + c(y, x) \geq |\beta| + c(y, x) \geq dist[y] + c(y, x) \geq dist[x]$$

where $|\beta|$ is the min weight of an s-to-y S_k-path.



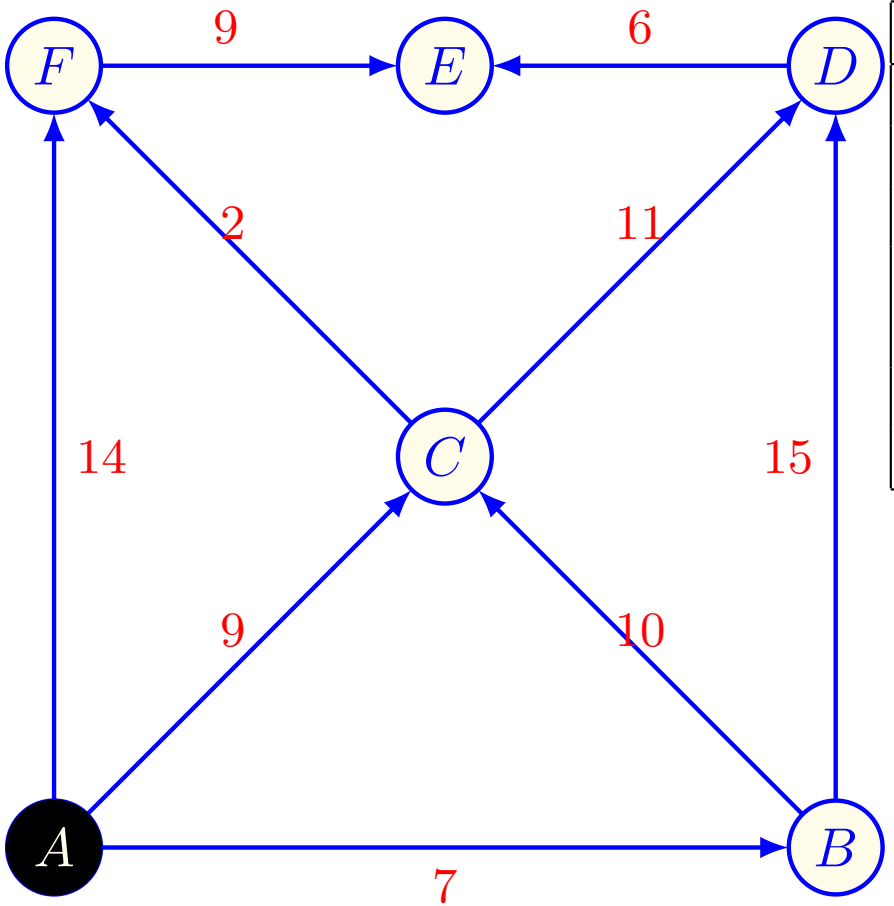
Dijksra's Algorithm: Example 2



Node u	A	B	C	D	E	F
	0	7	9	∞	∞	14
A	0	7	9	∞	∞	14
A B	0	7	9	22	∞	14
A B C	0	7	9	20	∞	11
A B C F	0	7	9	20	20	11
A B C D F	0	7	9	20	20	11
A B C D E F	0	7	9	20	20	11

for $u \in V(G)$ $dist[u] \leftarrow c[A, u]$

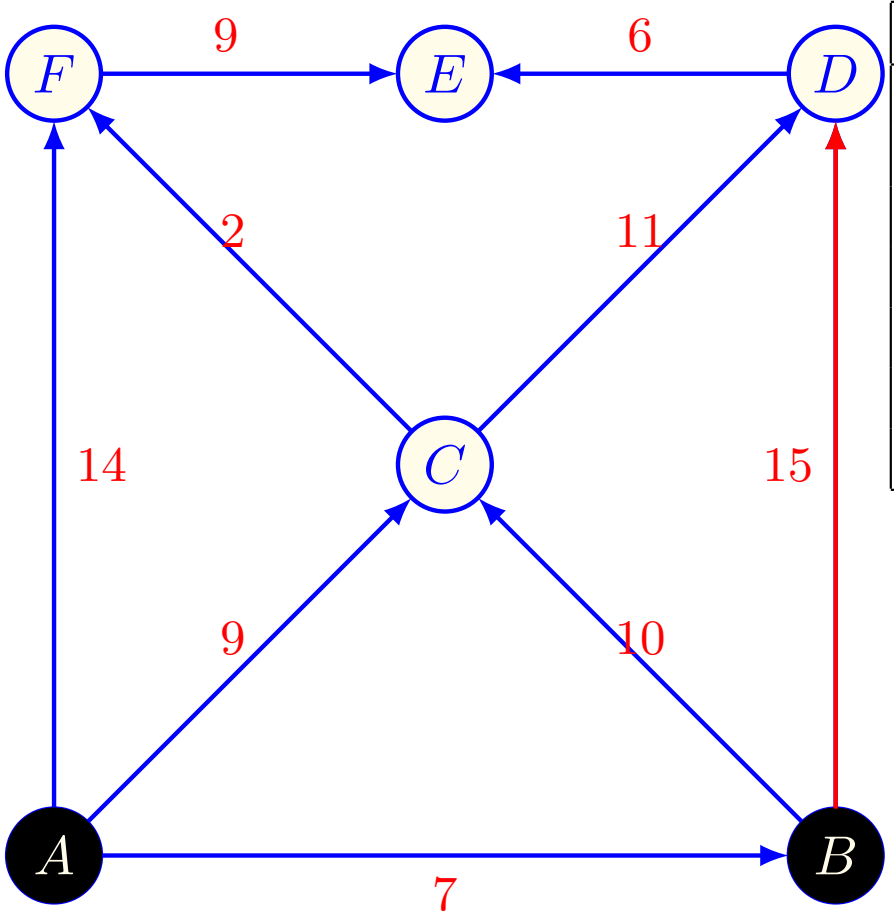
Dijksra's Algorithm: Example 2



Node u	A	B	C	D	E	F
	0	7	9	∞	∞	14
A	0	7	9	∞	∞	14
A B	0	7	9	22	∞	14
A B C	0	7	9	20	∞	11
A B C F	0	7	9	20	20	11
A B C D F	0	7	9	20	20	11
A B C D E F	0	7	9	20	20	11

$colour[A] \leftarrow \text{BLACK}$; $dist[A] \leftarrow 0$

Dijksra's Algorithm: Example 2



Node u	A	B	C	D	E	F
	0	7	9	∞	∞	14
A	0	7	9	∞	∞	14
A B	0	7	9	22	∞	14
A B C	0	7	9	20	∞	11
A B C F	0	7	9	20	20	11
A B C D F	0	7	9	20	20	11
A B C D E F	0	7	9	20	20	11

while-loop:

WHITE B, C, D, E, F: min $dist[B]$

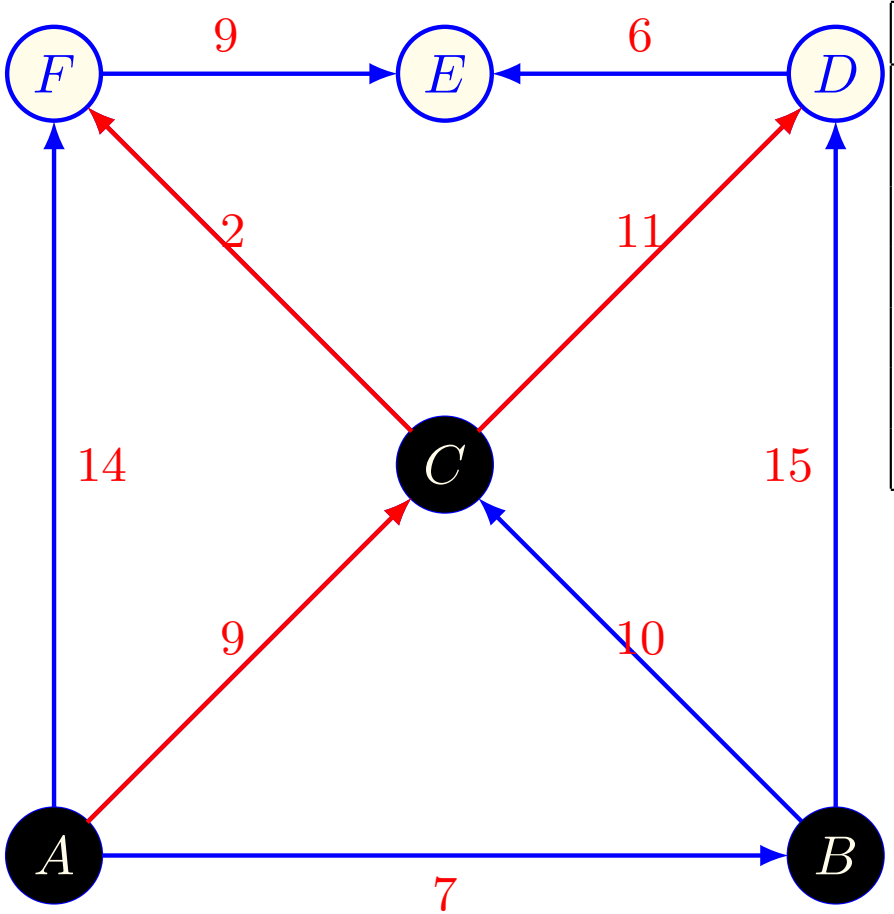
$colour[B] \leftarrow$ BLACK

for $x \in V(G)$

$dist[x] \leftarrow$

$\min \{dist[x], dist[B] + c[B, x]\}$

Dijksra's Algorithm: Example 2



Node u	A	B	C	D	E	F
	0	7	9	∞	∞	14
A	0	7	9	∞	∞	14
A B	0	7	9	22	∞	14
A B C	0	7	9	20	∞	11
A B C F	0	7	9	20	20	11
A B C D F	0	7	9	20	20	11
A B C D E F	0	7	9	20	20	11

while-loop:

WHITE C, D, E, F: min $dist[C]$

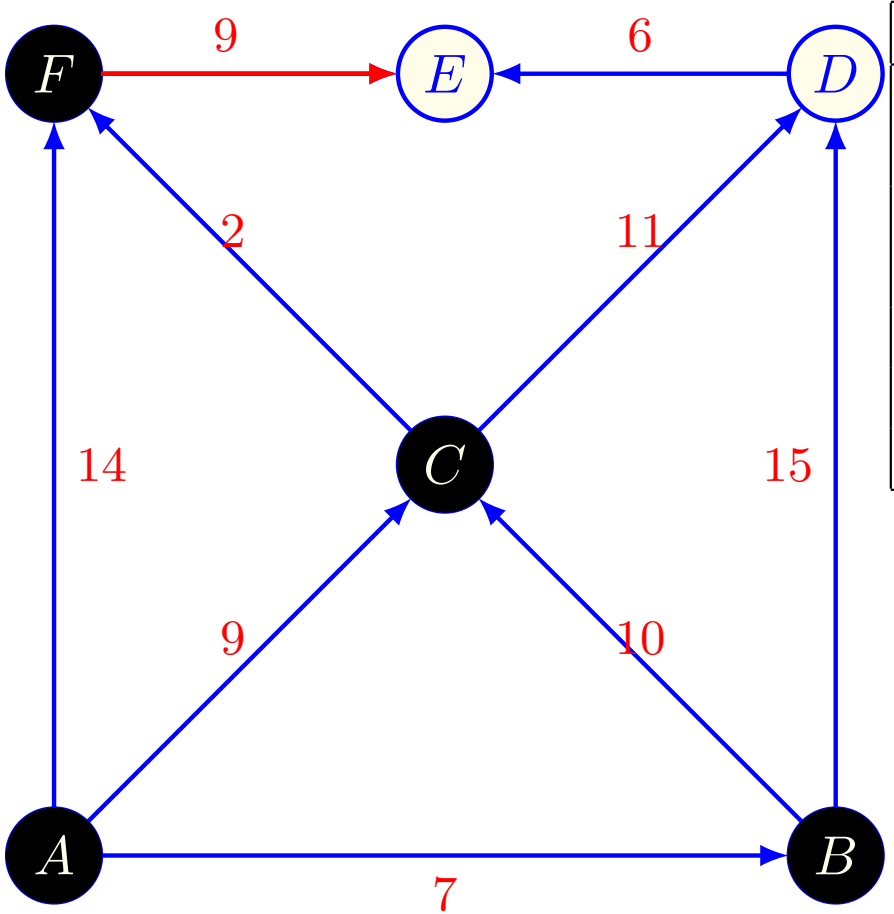
$colour[C] \leftarrow$ BLACK;

for $x \in V(G)$

$dist[x] \leftarrow$

$\min\{dist[x], dist[C] + c[C, x]\}$

Dijksra's Algorithm: Example 2



Node u	A	B	C	D	E	F
	0	7	9	∞	∞	14
A	0	7	9	∞	∞	14
A B	0	7	9	22	∞	14
A B C	0	7	9	20	∞	11
A B C F	0	7	9	20	20	11
A B C D F	0	7	9	20	20	11
A B C D E F	0	7	9	20	20	11

while-loop:

WHITE D, E, F: min $dist[F]$

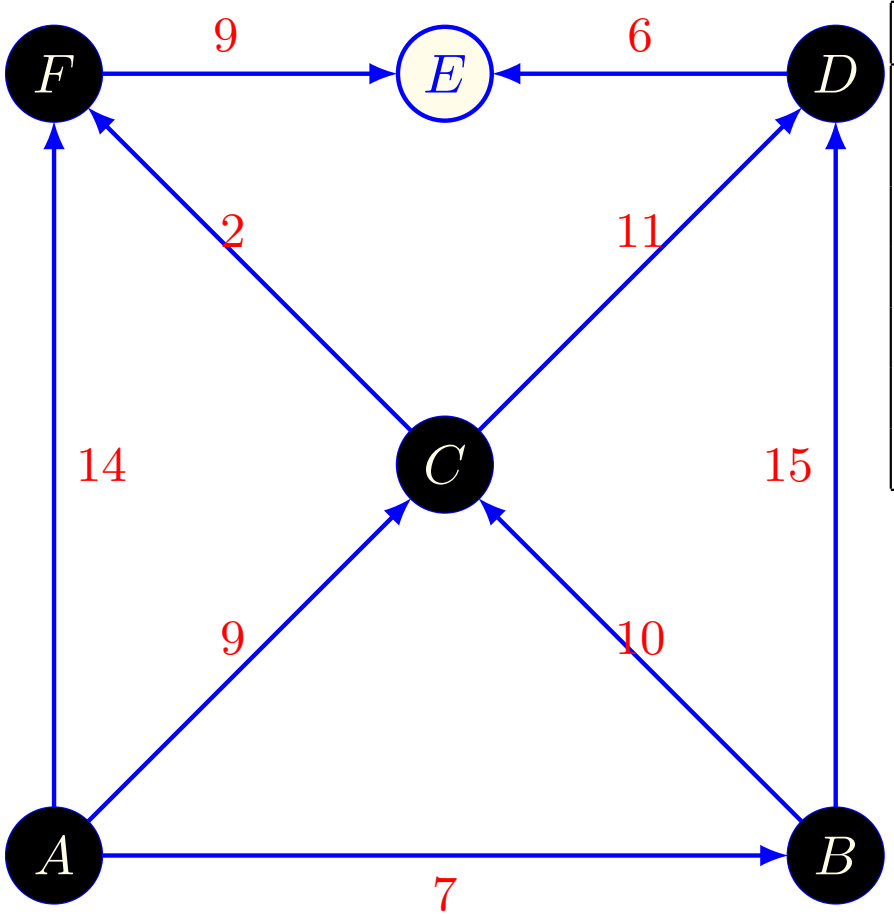
$colour[F] \leftarrow$ BLACK;

for $x \in V(G)$

$dist[x] \leftarrow$

$\min\{dist[x], dist[F] + c[F, x]\}$

Dijksra's Algorithm: Example 2



Node u	A	B	C	D	E	F
	0	7	9	∞	∞	14
A	0	7	9	∞	∞	14
A B	0	7	9	22	∞	14
A B C	0	7	9	20	∞	11
A B C F	0	7	9	20	20	11
A B C D F	0	7	9	20	20	11
A B C D E F	0	7	9	20	20	11

while-loop:

WHITE D, E: min $dist[D]$

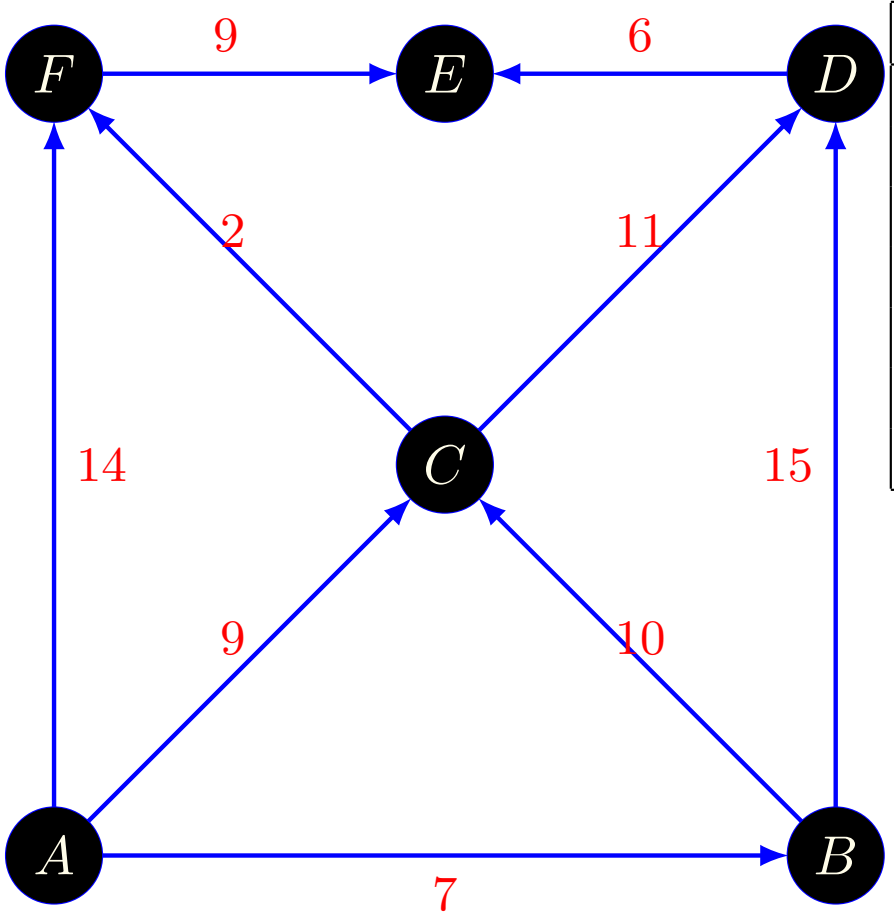
$colour[D] \leftarrow$ BLACK;

for $x \in V(G)$

$dist[x] \leftarrow$

$\min\{dist[x], dist[D] + c[D, x]\}$

Dijksra's Algorithm: Example 2



Node u	A	B	C	D	E	F
	0	7	9	∞	∞	14
A	0	7	9	∞	∞	14
A B	0	7	9	22	∞	14
A B C	0	7	9	20	∞	11
A B C F	0	7	9	20	20	11
A B C D F	0	7	9	20	20	11
A B C D E F	0	7	9	20	20	11

while-loop:

WHITE E: min $dist[E]$

$colour[E] \leftarrow$ BLACK;

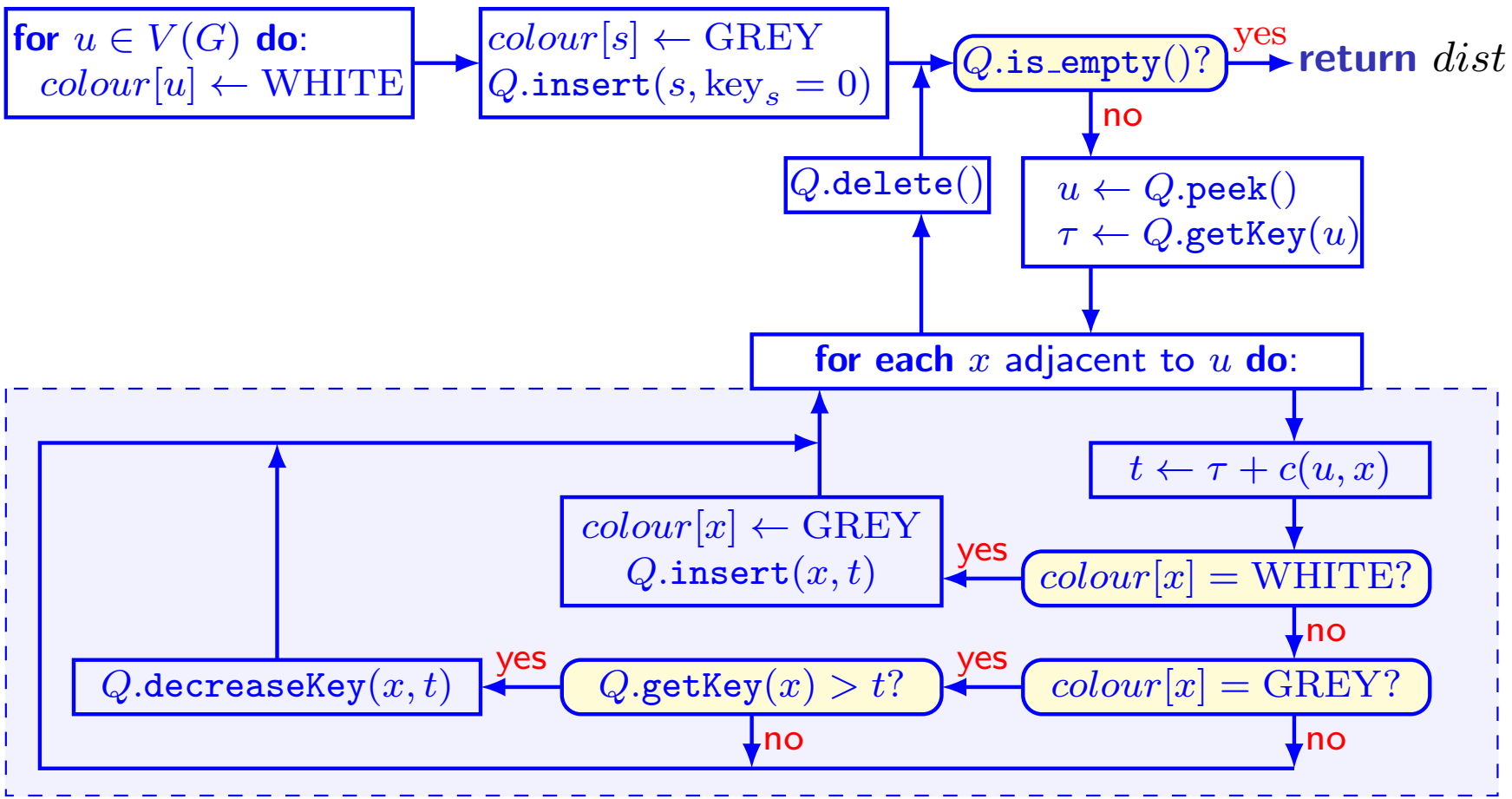
for $x \in V(G)$

$dist[x] \leftarrow$

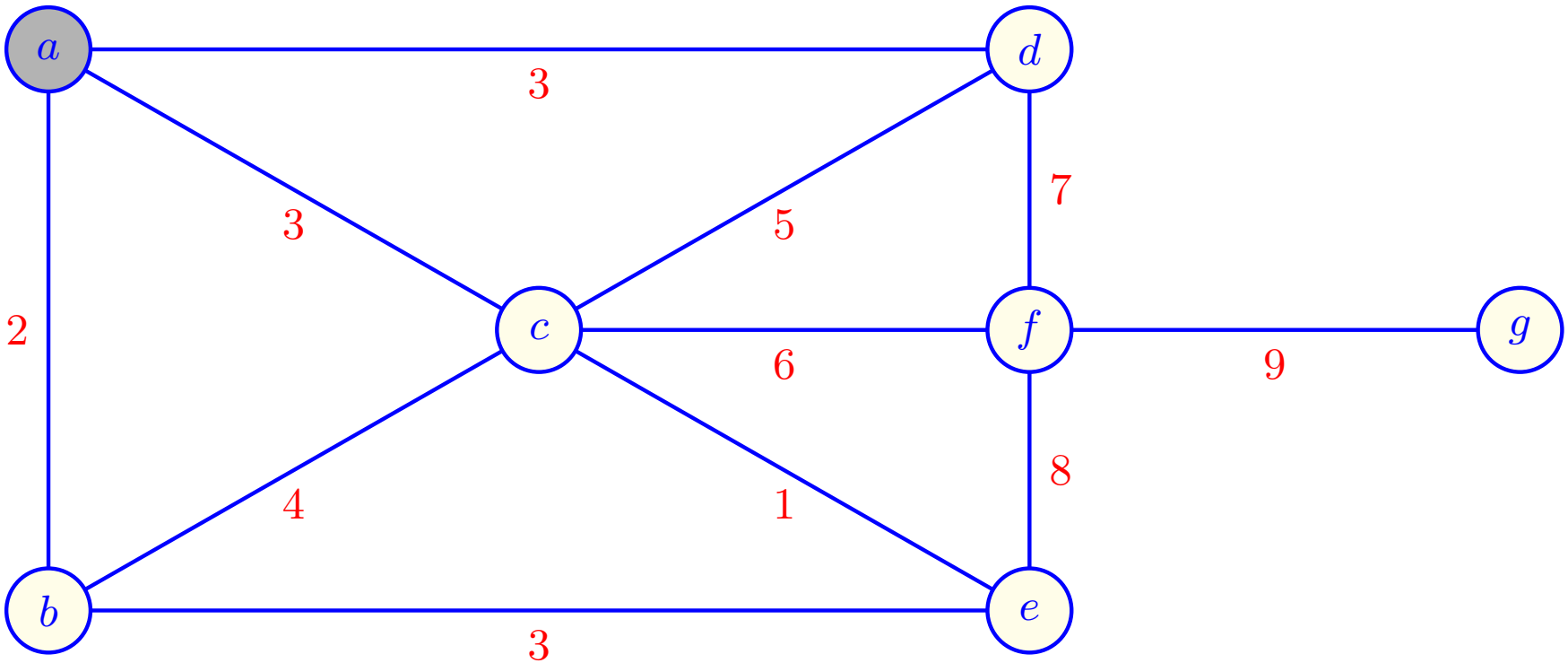
$\min\{dist[x], dist[E] + c[E, x]\}$

Dijksta's Algorithm: PFS Version

Input: weighted digraph (G, c); source node $s \in V(G)$;
priority queue Q; arrays $dist[0..n-1]$; $colour[0..n-1]$



Dijkstra's Algorithm: PFS Version: Start at a

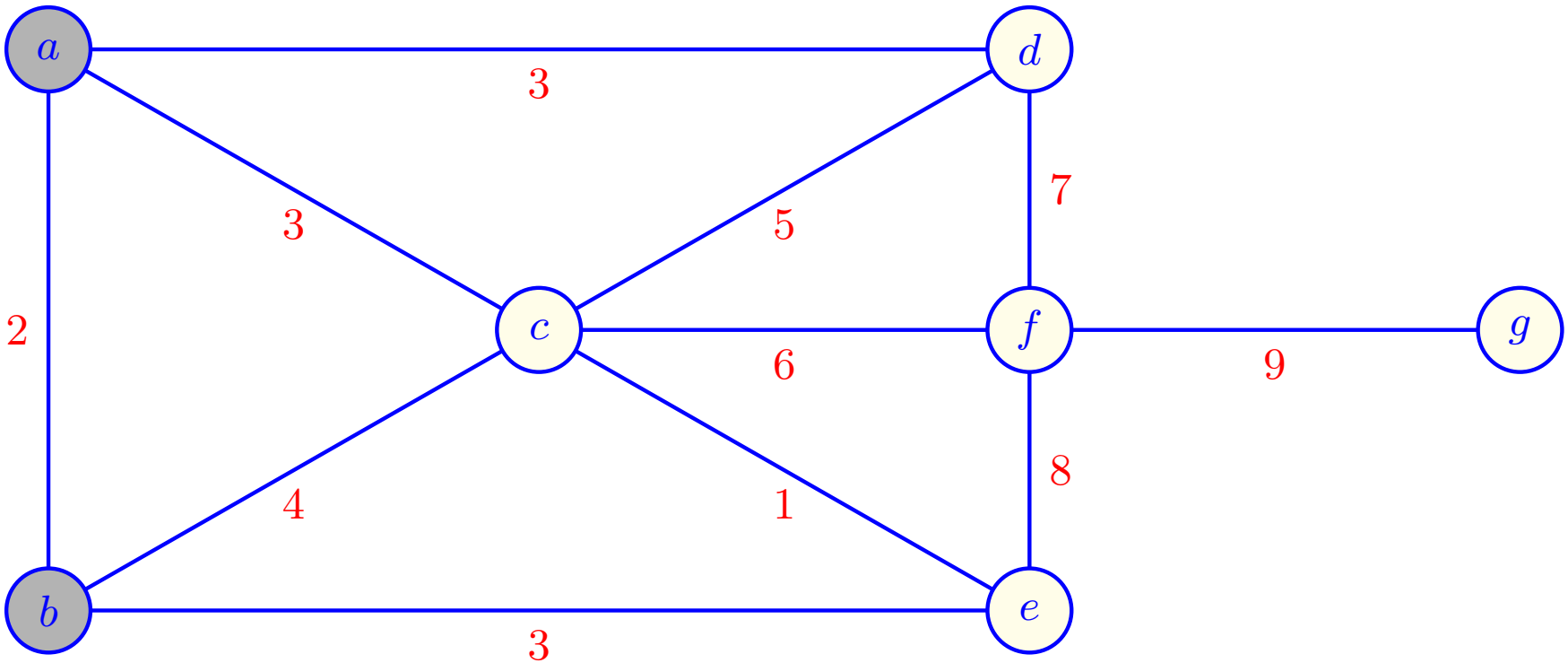


Initialisation:

Priority queue $Q = \{a_{\text{key}=0}\}$

$v \in V$	a	b	c	d	e	f	g
key_v	0						
$dist[v]$	–	–	–	–	–	–	–

Dijkstra's Algorithm: PFS Version: Steps 1 – 2

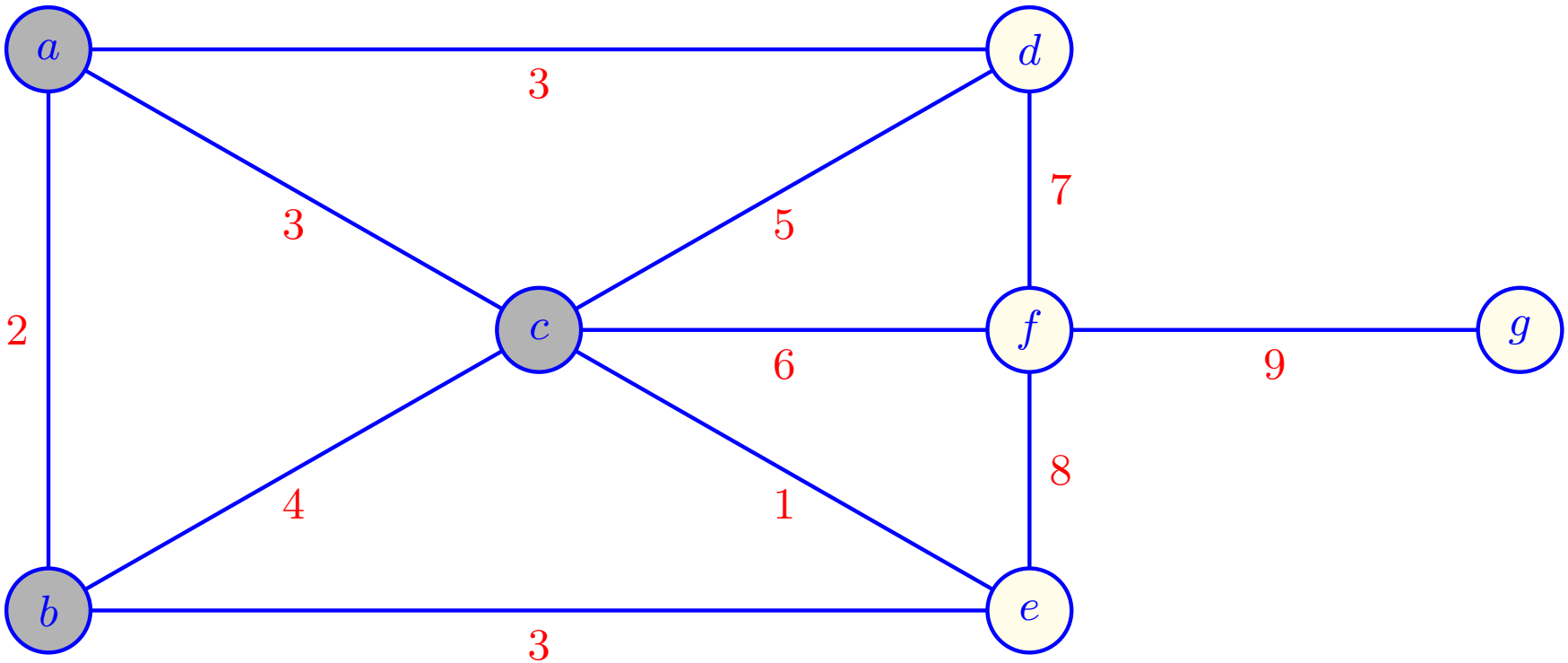


$u \leftarrow a$; $t_1 \leftarrow \text{key}_a = 0$; $x \in \{b, c, d\}$

$x \leftarrow b$: $t_2 = t_1 + \text{cost}(a, b) = 2$; $Q = \{a_0, b_2\}$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2					
$dist[v]$	–	–	–	–	–	–	–

Dijkstra's Algorithm: PFS Version: Step 3

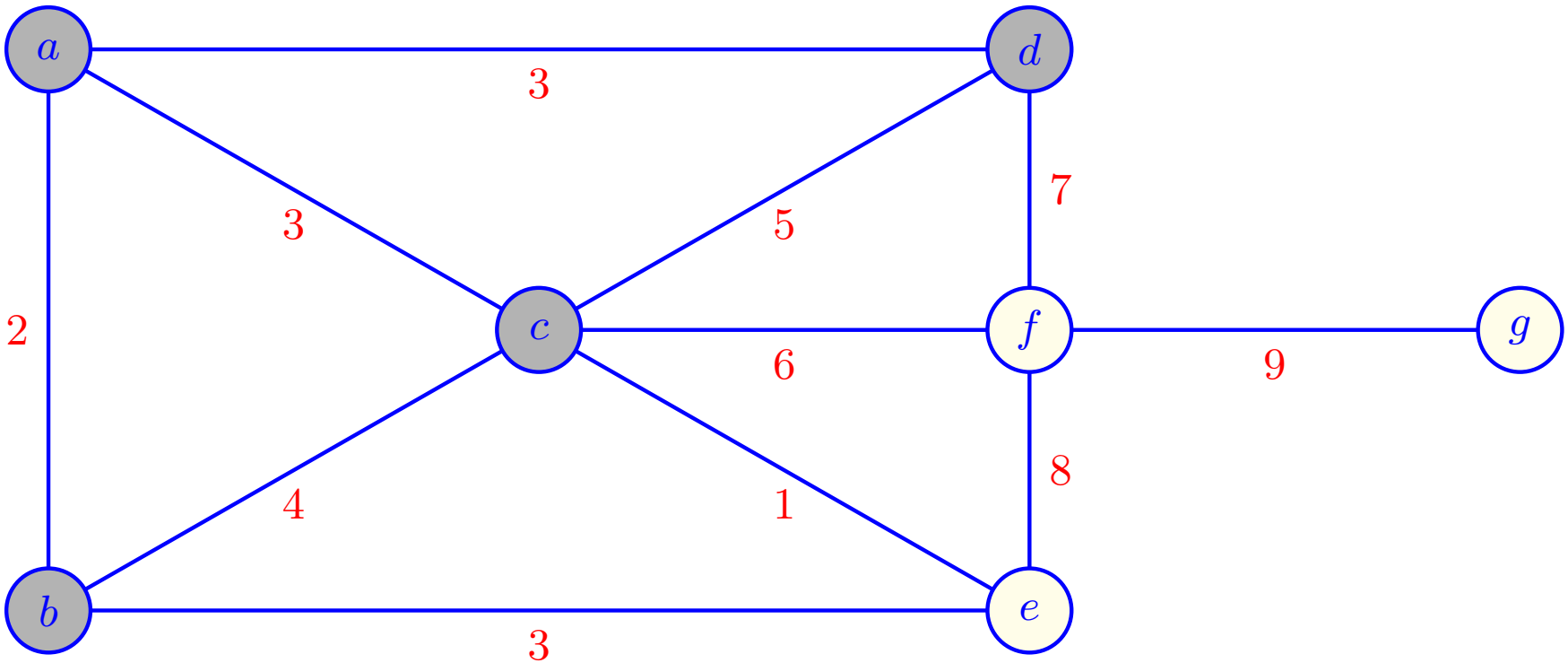


$u = a$; $t_1 = \text{key}_a = 0$; $x \in \{b, c, d\}$

$x \leftarrow c$: $t_2 = t_1 + \text{cost}(a, c) = 3$; $Q = \{a_0, b_2, c_3\}$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3				
$dist[v]$	–	–	–	–	–	–	–

Dijkstra's Algorithm: PFS Version: Step 4

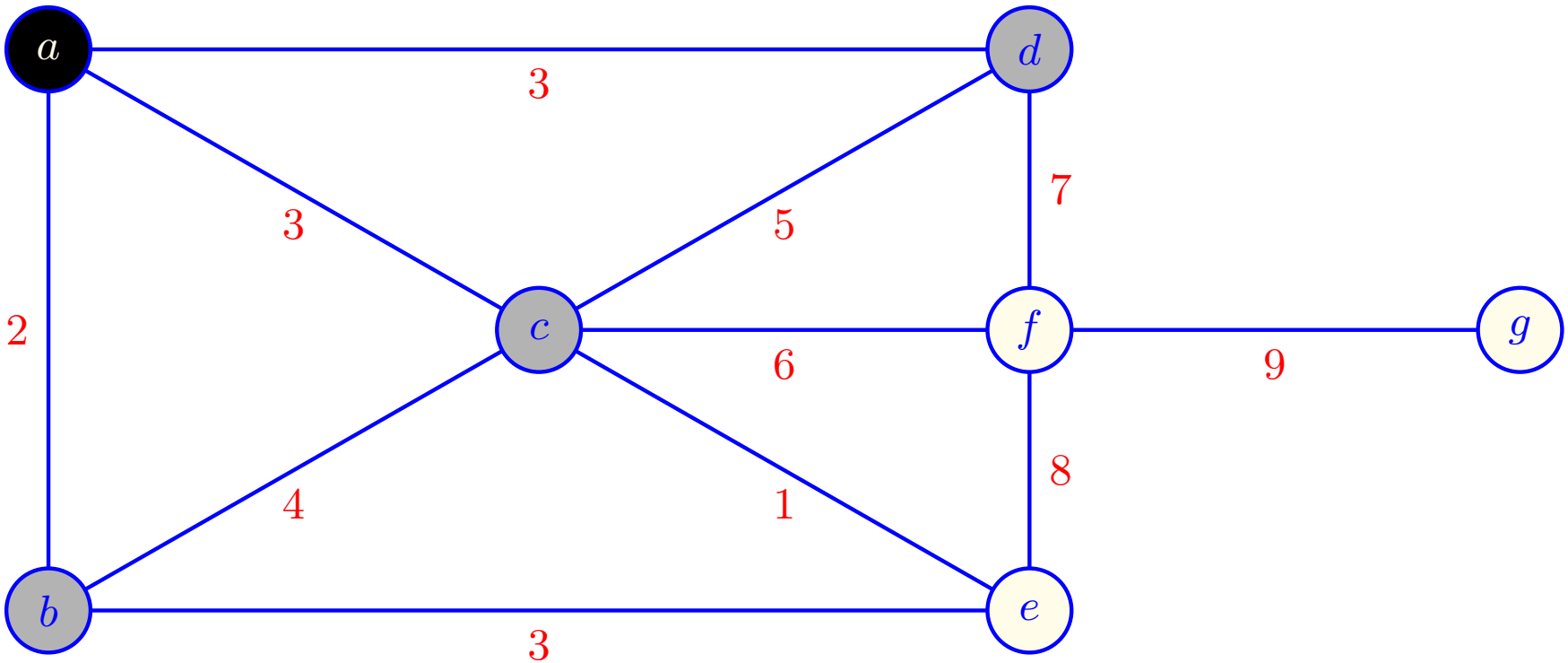


$u = a$; $t_1 = \text{key}_a = 0$; $x \in \{b, c, d\}$

$x \leftarrow d$: $t_2 = t_1 + \text{cost}(a, d) = 3$; $Q = \{a_0, b_2, c_3, d_3\}$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3	3			
$dist[v]$	–	–	–	–	–	–	–

Dijkstra's Algorithm: PFS Version: Step 5

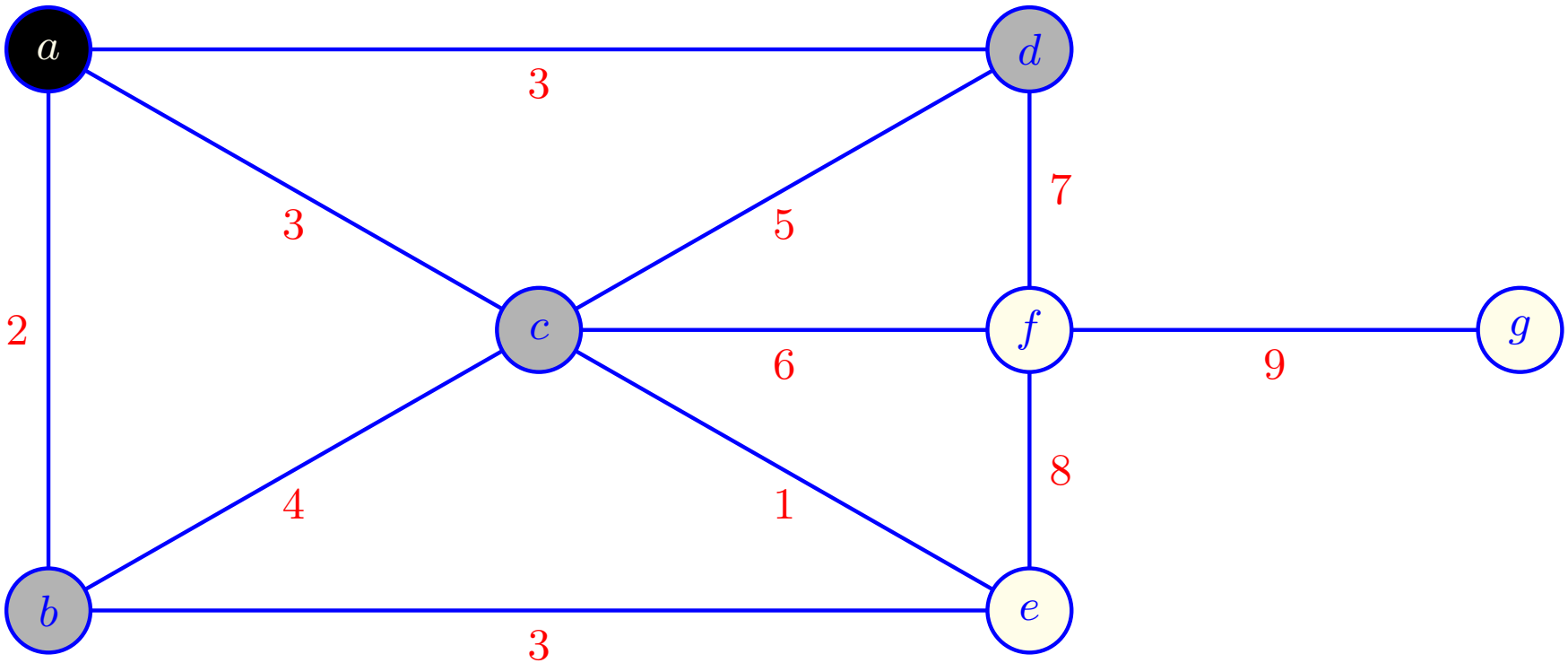


Completing the **while**-loop for $u = a$

$dist[a] \leftarrow t_1 = 0$; $Q = \{b_2, c_3, d_3\}$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3	3			
$dist[v]$	0	–	–	–	–	–	–

Dijkstra's Algorithm: PFS Version: Steps 6 – 7

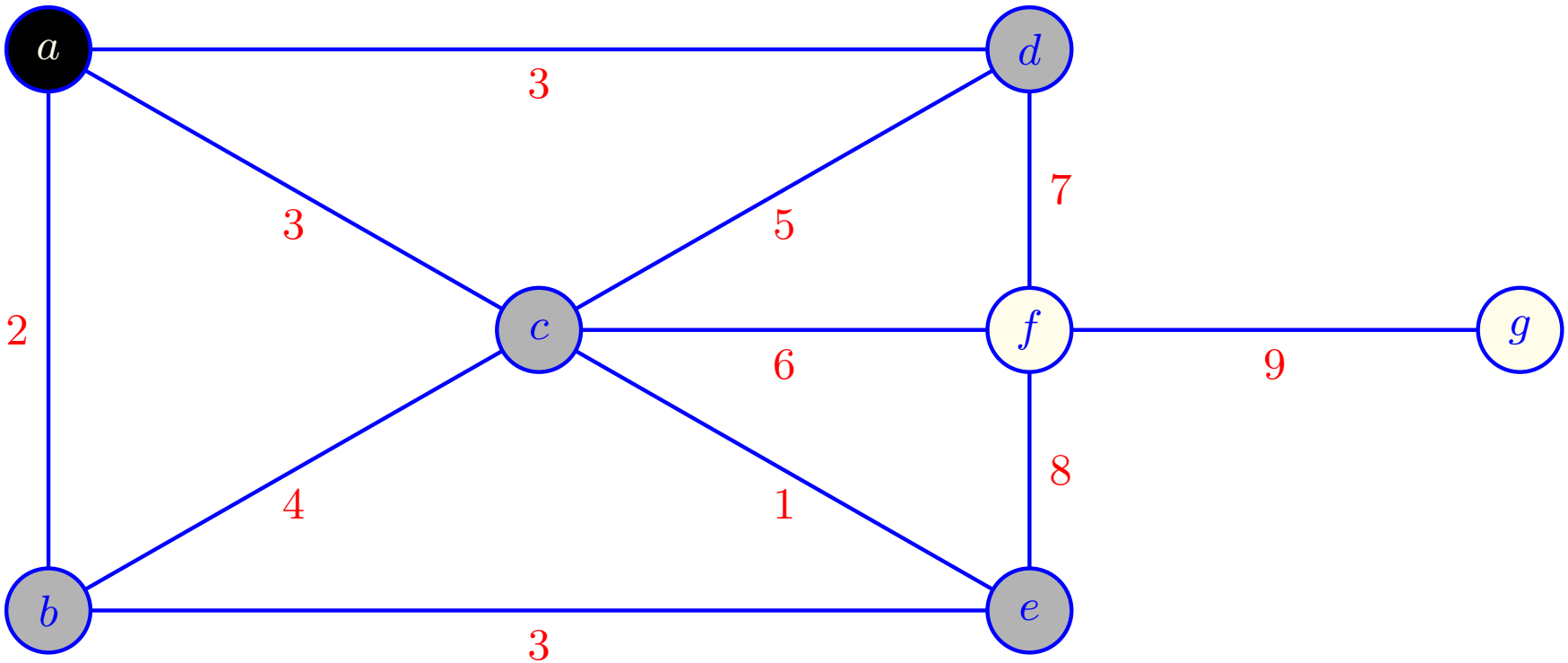


$u \leftarrow b$; $t_1 \leftarrow \text{key}_b = 2$; $x \in \{c, e\}$

$x \leftarrow c$: $t_2 = t_1 + \text{cost}(b, c) = 2 + 4 = 6$; $\text{key}_c = 3 < t_2 = 6$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3	3			
$dist[v]$	0	–	–	–	–	–	–

Dijkstra's Algorithm: PFS Version: Step 8

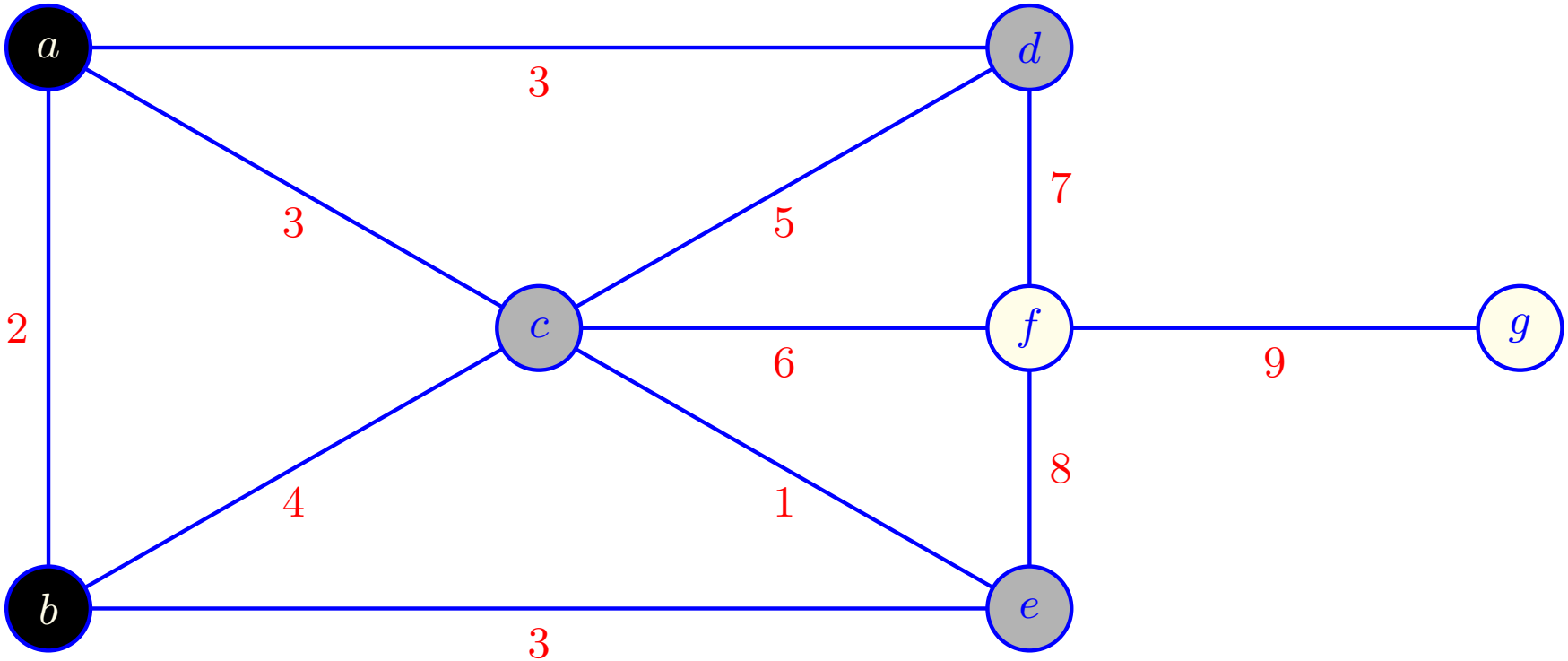


$u = b$; $t_1 = \text{key}_b = 2$; $x \in \{c, e\}$

$x \leftarrow e$: $t_2 = t_1 + \text{cost}(b, e) = 2 + 3 = 5$; $Q = \{b_2, c_3, d_3, e_5\}$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3	3	5		
$dist[v]$	0	–	–	–	–	–	–

Dijkstra's Algorithm: PFS Version: Step 9

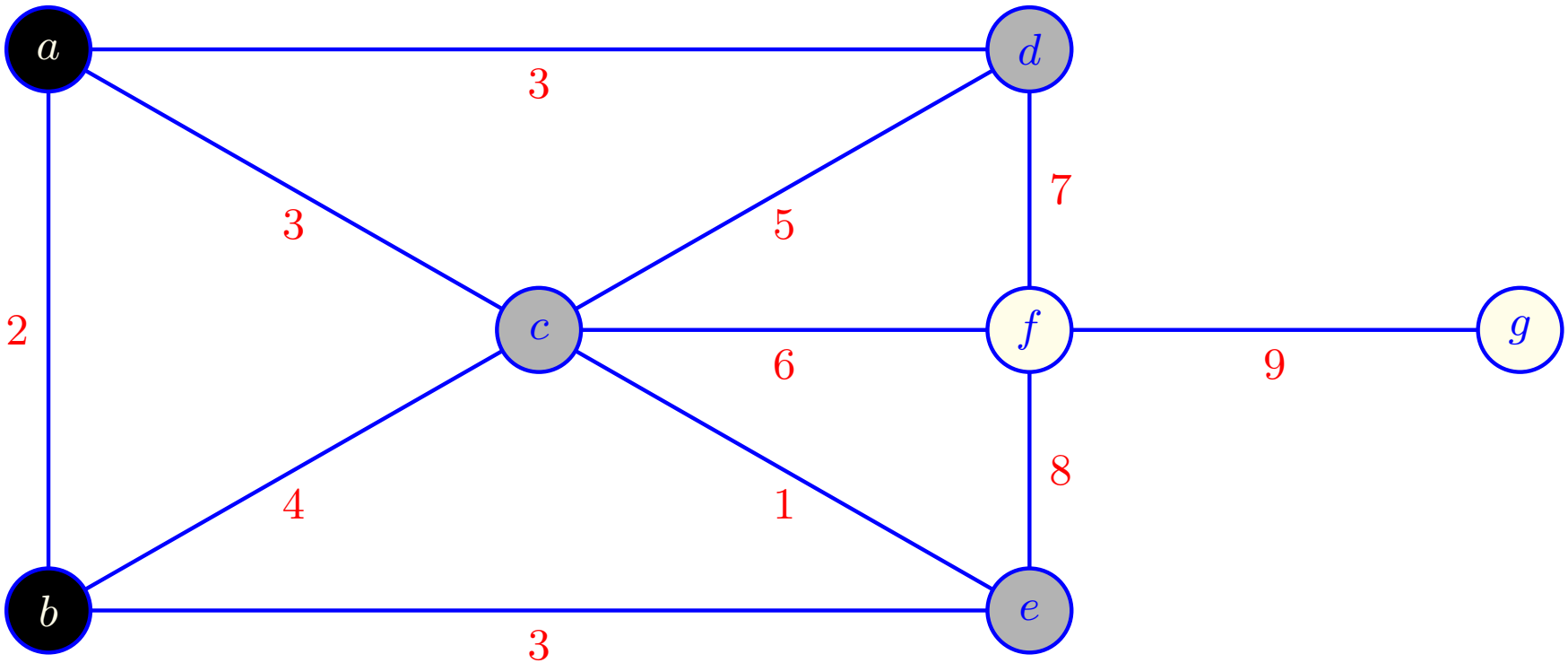


Completing the **while**-loop for $u = b$

$dist[b] \leftarrow t_1 = 2$; $Q = \{c_3, d_3, e_5\}$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3	3	5		
$dist[v]$	0	2	–	–	–	–	–

Dijkstra's Algorithm: PFS Version: Steps 10 – 11

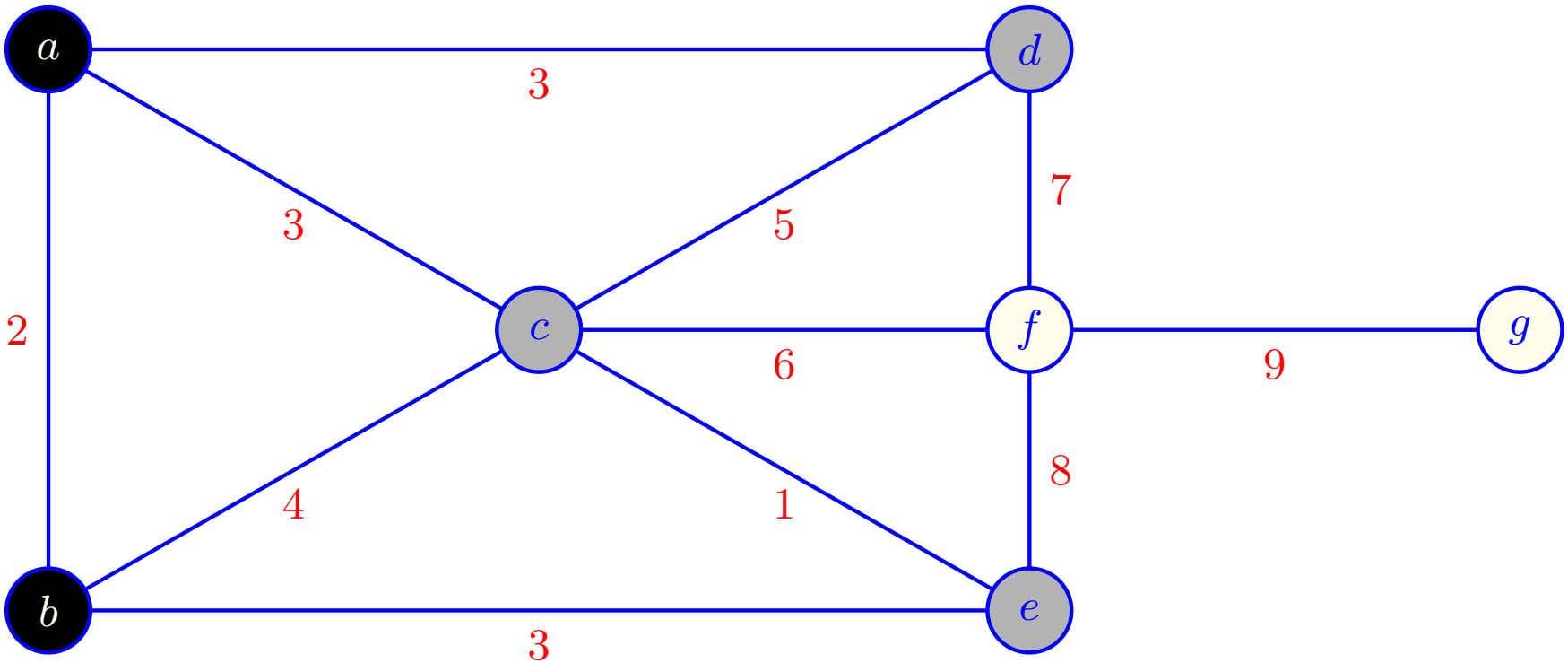


$u \leftarrow c$; $t_1 \leftarrow \text{key}_c = 3$; $x \in \{d, e, f\}$

$x \leftarrow d$: $t_2 = t_1 + \text{cost}(c, d) = 3 + 5 = 8$; $\text{key}_d = 3 < t_2 = 8$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3	3	5		
$dist[v]$	0	2	–	–	–	–	–

Dijkstra's Algorithm: PFS Version: Step 12

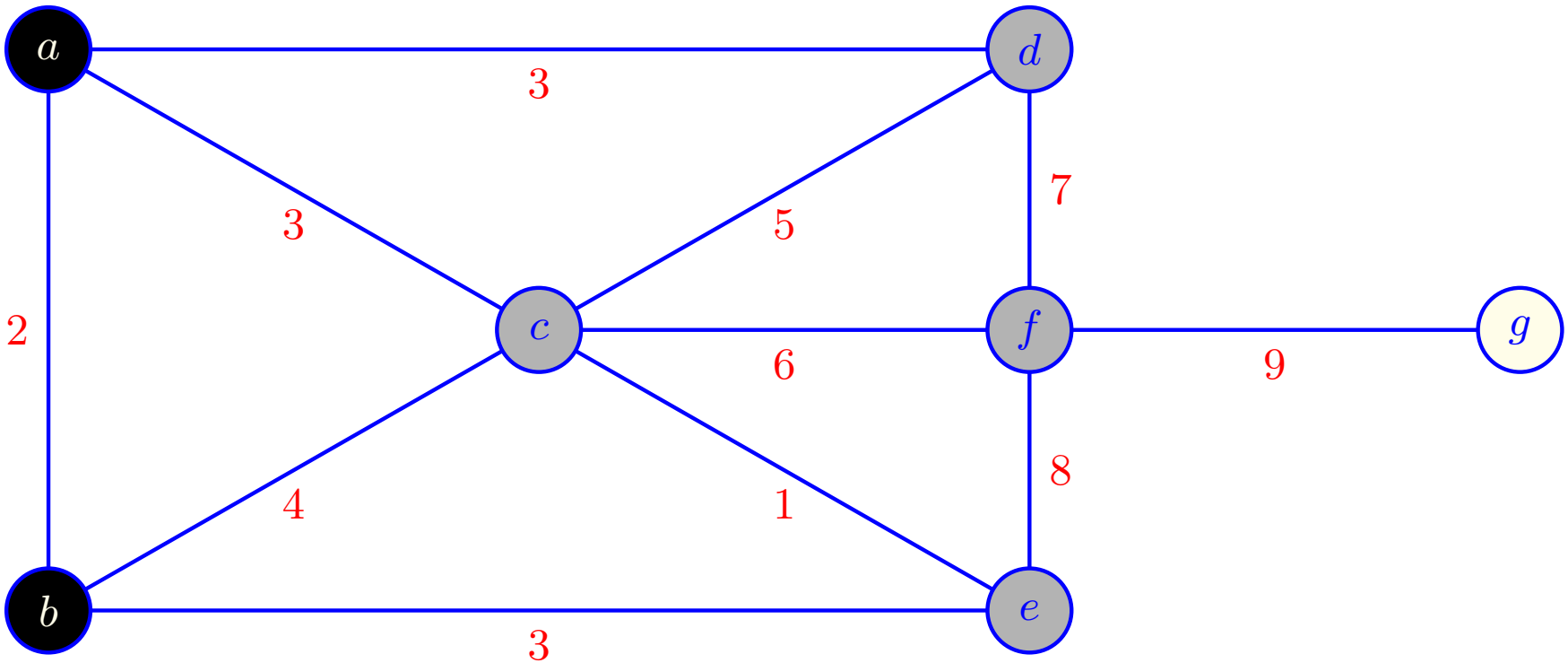


$u = c$; $t_1 = \text{key}_c = 3$; $x \in \{d, e, f\}$

$x \leftarrow e$: $t_2 = t_1 + \text{cost}(c, d) = 3 + 1 = 4$; $\text{key}_e = 5 < t_2 = 4$; $\text{key}_e \leftarrow 4$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3	3	4		
$dist[v]$	0	2	–	–	–	–	–

Dijkstra's Algorithm: PFS Version: Step 13

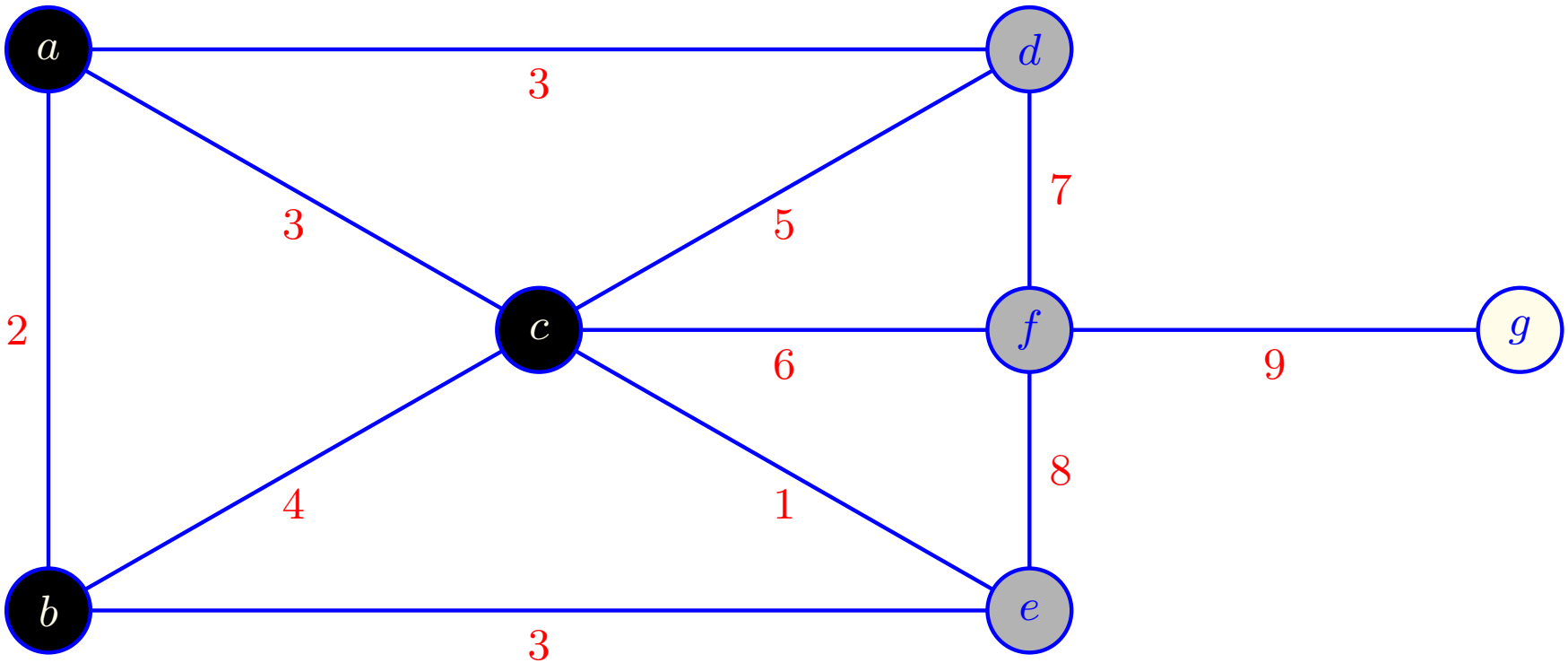


$u = c$; $t_1 = \text{key}_c = 3$; $x \in \{d, e, f\}$

$x \leftarrow f$: $t_2 = t_1 + \text{cost}(c, f) = 3 + 6 = 9$; $Q = \{c_3, d_3, e_4, f_9\}$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3	3	4	9	
$dist[v]$	0	2	–	–	–	–	–

Dijkstra's Algorithm: PFS Version: Step 14

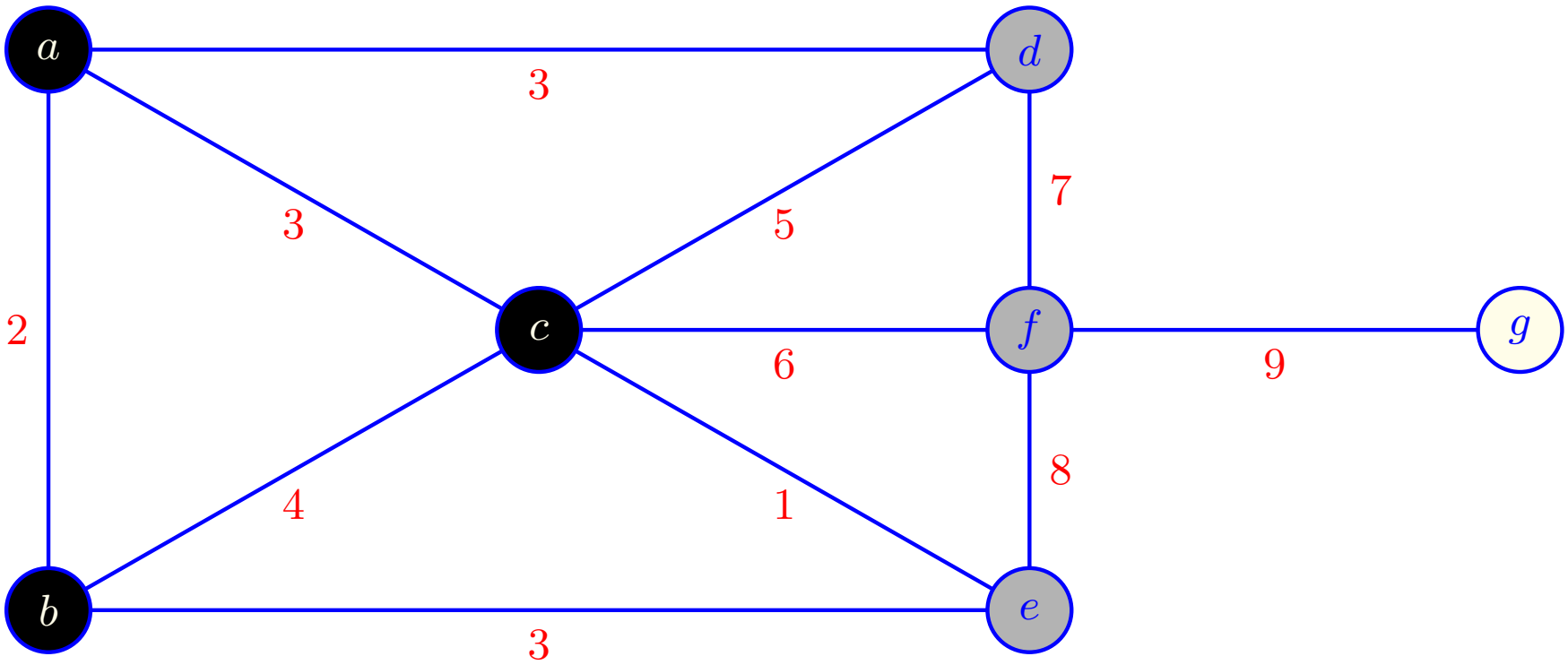


Completing the **while**-loop for $u = c$

$dist[c] \leftarrow t_1 = 3$; $Q = \{d_3, e_4, f_9\}$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3	3	4	9	
$dist[v]$	0	2	3	–	–	–	–

Dijkstra's Algorithm: PFS Version: Steps 15 – 16

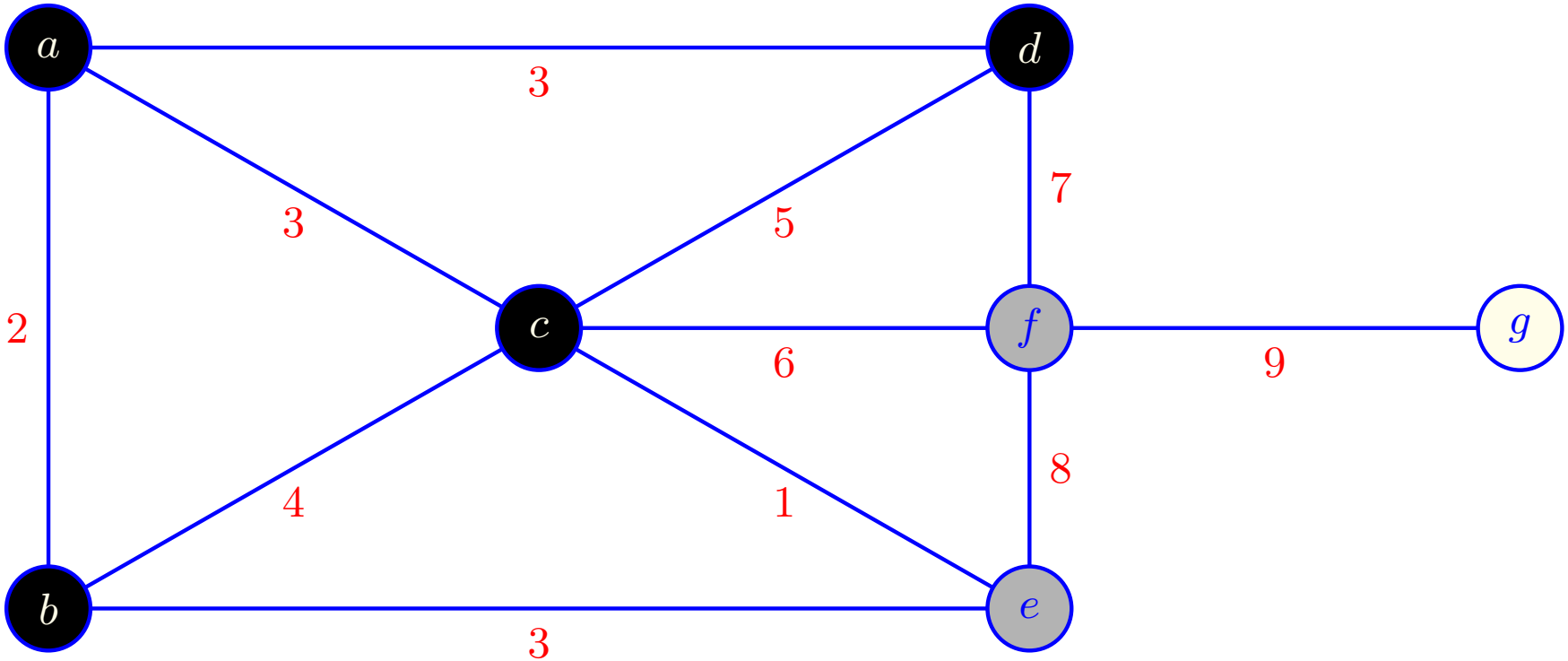


$u \leftarrow d$; $t_1 \leftarrow \text{key}_d = 3$; $x \in \{f\}$

$x \leftarrow f$: $t_2 = t_1 + \text{cost}(d, f) = 3 + 7 = 10$; $\text{key}_f = 9 < t_2 = 10$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3	3	4	9	
$dist[v]$	0	2	3	–	–	–	–

Dijkstra's Algorithm: PFS Version: Step 17

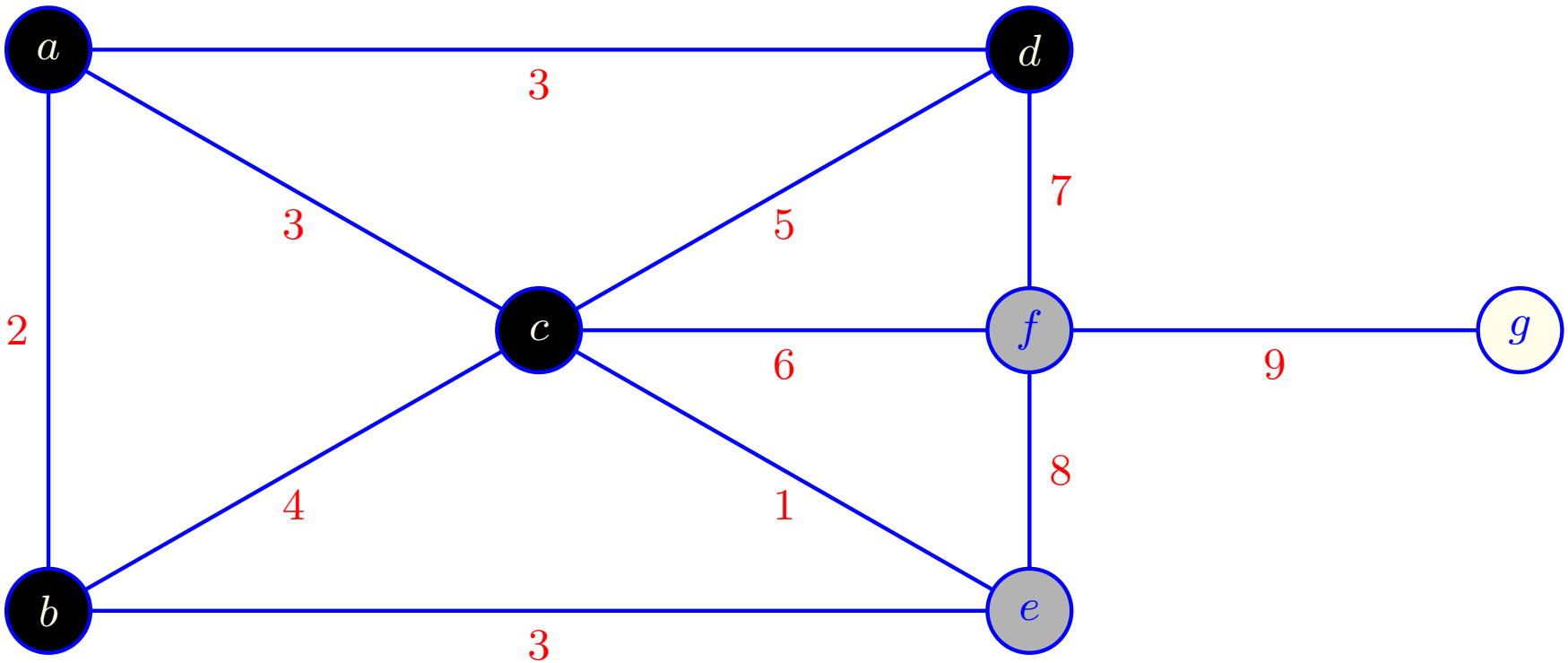


Completing the **while**-loop for $u = d$

$dist[d] \leftarrow t_1 = 3$; $Q = \{e_4, f_9\}$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3	3	4	9	
$dist[v]$	0	2	3	3	–	–	–

Dijkstra's Algorithm: PFS Version: Steps 18 – 19

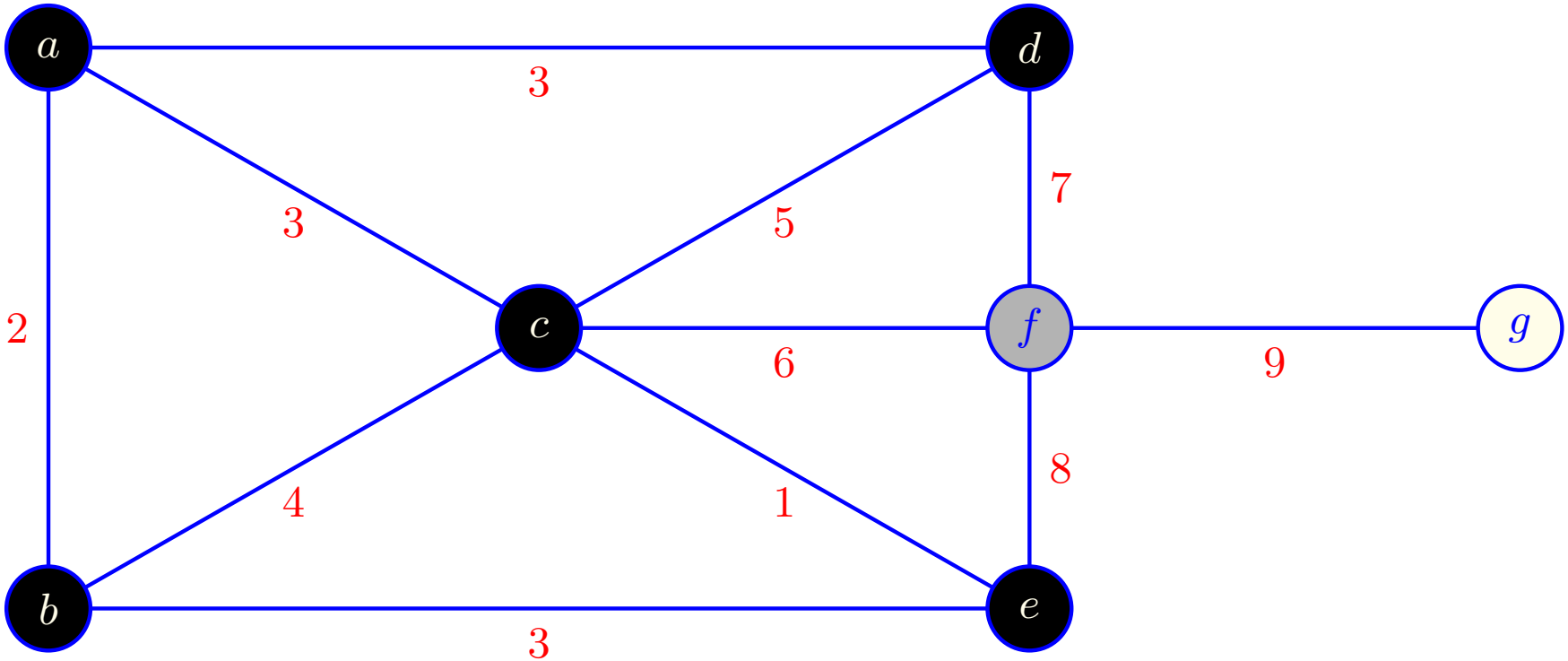


$u \leftarrow e$; $t_1 \leftarrow \text{key}_e = 4$; $x \in \{f\}$

$x \leftarrow f$: $t_2 = t_1 + \text{cost}(e, f) = 4 + 8 = 12$; $\text{key}_f = 9 < t_2 = 12$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3	3	4	9	
$dist[v]$	0	2	3	3	–	–	–

Dijkstra's Algorithm: PFS Version: Step 20

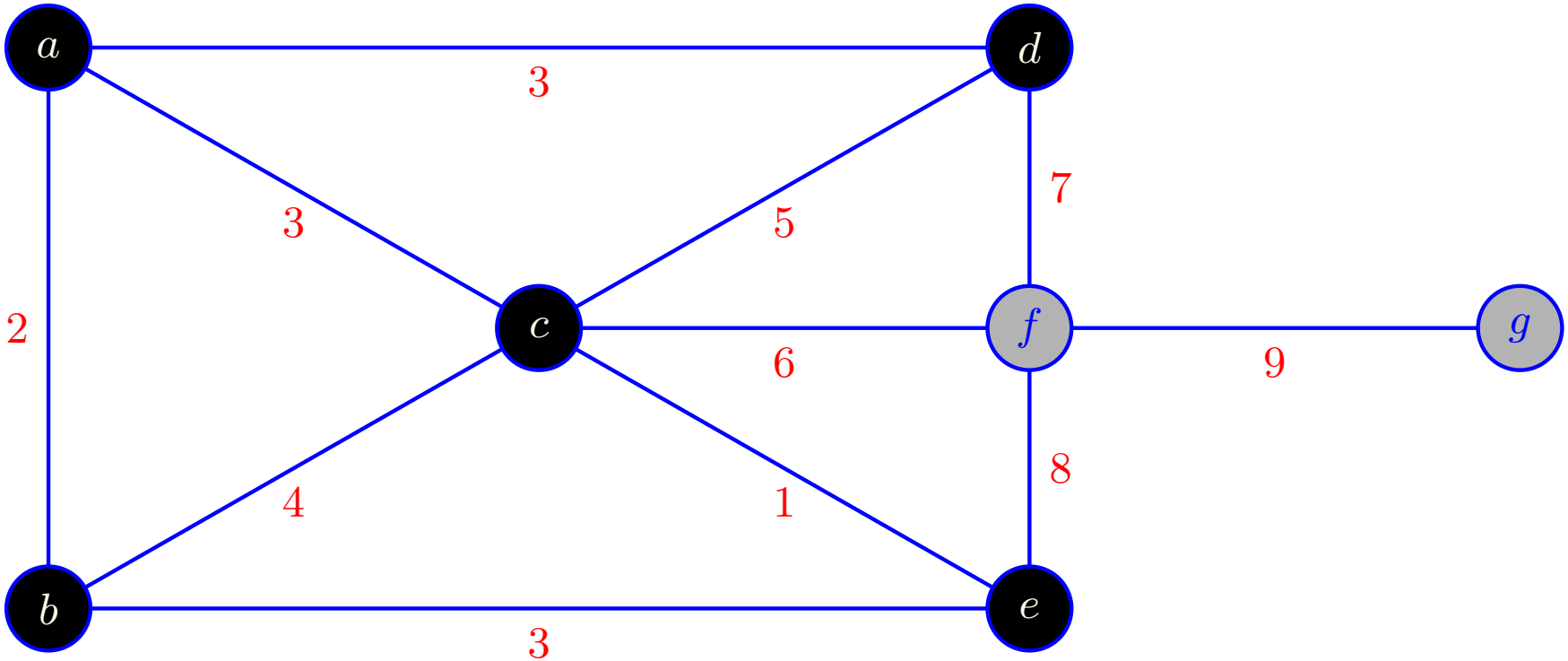


Completing the **while**-loop for $u = e$

$dist[e] \leftarrow t_1 = 4$; $Q = \{f_9\}$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3	3	4	9	
$dist[v]$	0	2	3	3	4	–	–

Dijkstra's Algorithm: PFS Version: Steps 21 – 22

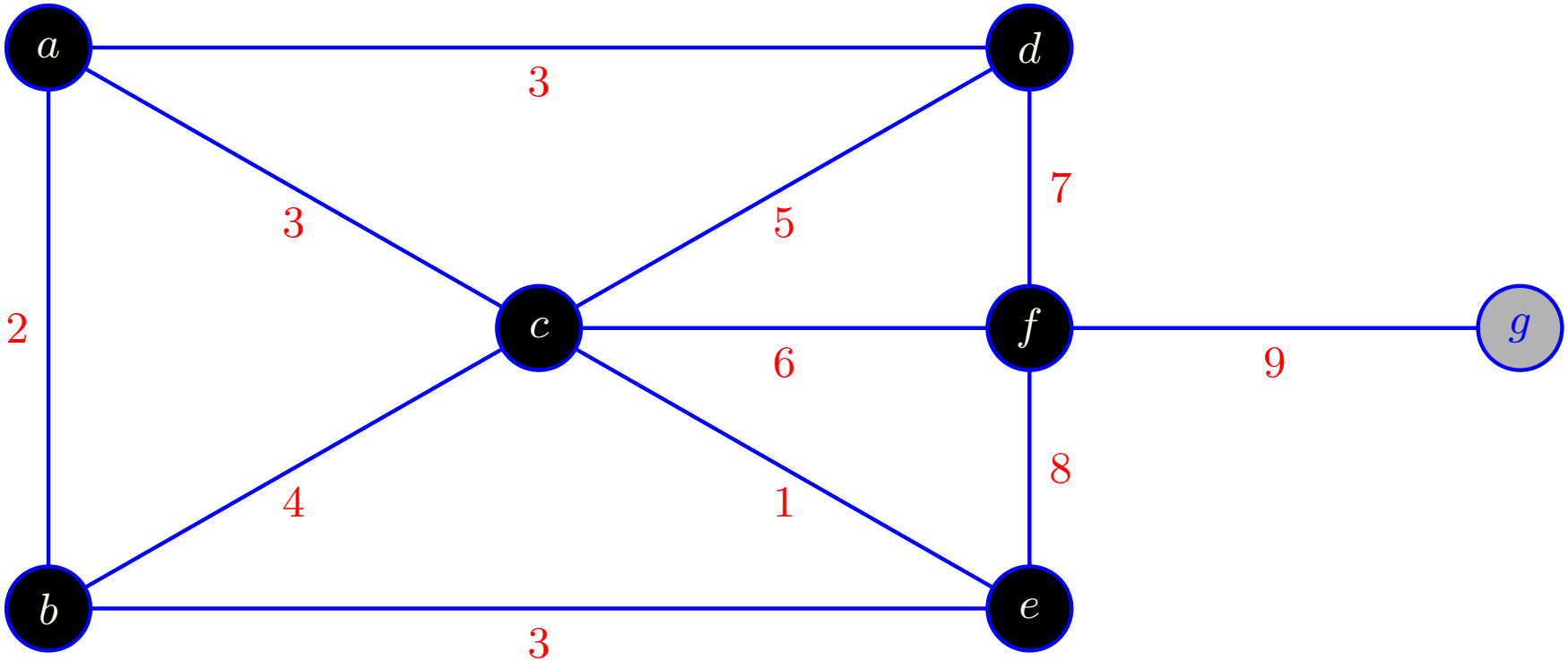


$u \leftarrow f$; $t_1 \leftarrow \text{key}_f = 9$; $x \in \{g\}$

$x \leftarrow g$: $t_2 = t_1 + \text{cost}(f, g) = 9 + 9 = 18$; $Q = \{f_9, g_{18}\}$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3	3	4	9	18
$dist[v]$	0	2	3	3	4	–	–

Dijkstra's Algorithm: PFS Version: Step 23

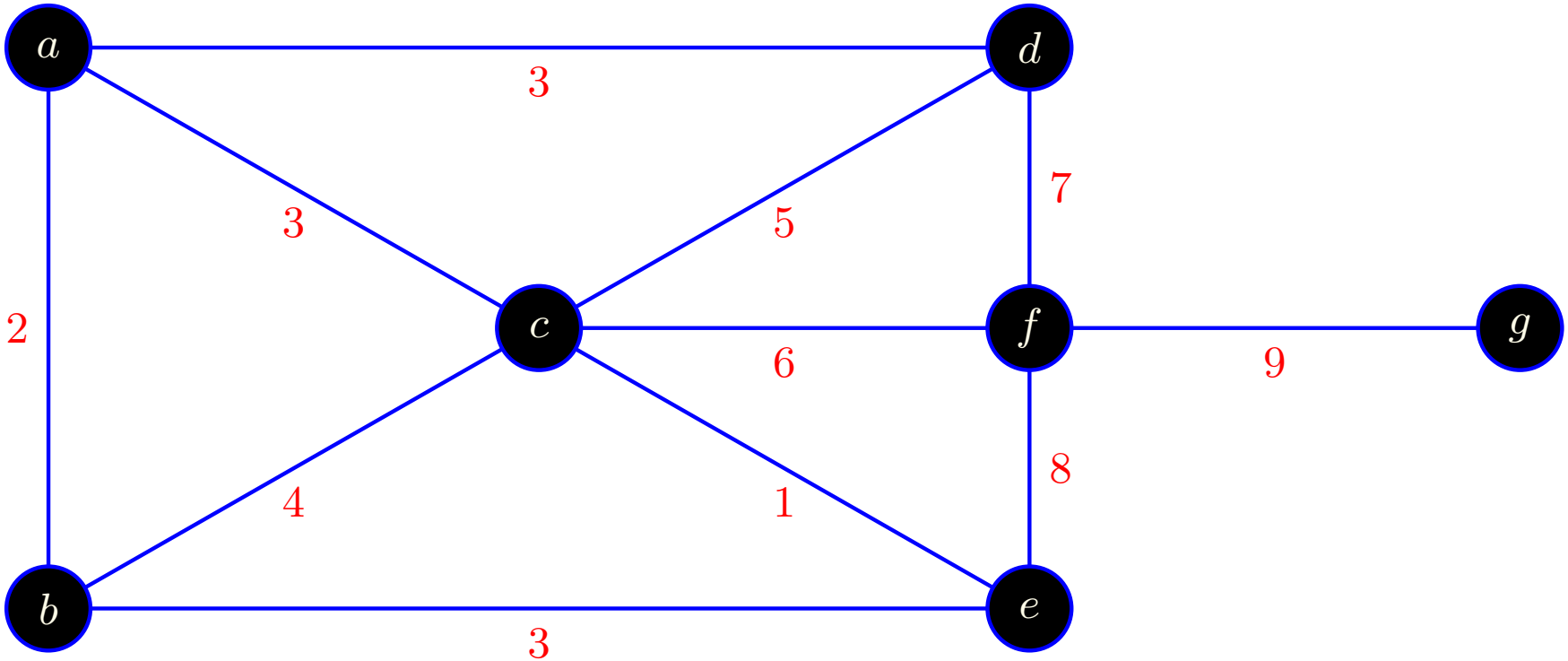


Completing the **while**-loop for $u = f$

$dist[f] \leftarrow t_1 = 9$; $Q = \{g_{18}\}$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3	3	4	9	18
$dist[v]$	0	2	3	3	4	9	–

Dijkstra's Algorithm: PFS Version: Steps 24 – 25



Completing the **while**-loop for $u = g$

$dist[g] \leftarrow t_1 = 18$; no adjacent verices for g; empty $Q = \{\}$

$v \in V$	a	b	c	d	e	f	g
key_v	0	2	3	3	4	9	18
$dist[v]$	0	2	3	3	4	9	18

SSSP: Bellman-Ford Algorithm

algorithm `Bellman-Ford`(weighted digraph (G, c); node s)

array $dist[n] = \{\infty, \infty, \ldots\}$
$dist[s] \leftarrow 0$
for i **from** 0 **to** $n - 1$ **do**
for $x \in V(G)$ **do**
for $v \in V(G)$ **do**
$dist[v] \leftarrow \min(dist[v], dist[x] + c(x, v))$
end for
end for
end for
return $dist$

end

Time complexity – $\Theta(n^3)$; unlike the Dijkstra's algorithm, it handles negative weight arcs (but no negative weight cycles making the SSSP senseless).

SSSP: Bellman-Ford Algorithm (Alternative Form)

algorithm `Bellman-Ford`(weighted digraph (G, c); node s)
    array $dist[n] = \{\infty, \infty, \ldots\}$
    $dist[s] \leftarrow 0$
    for i **from** 0 **to** $n - 1$ **do**
        for $(x, v) \in E(G)$ **do**
            $dist[v] \leftarrow \min(dist[v], dist[x] + c(x, v))$
        end for
    end for
    return $dist$
end

Replacing the two nested **for**-loops by the nodes $x, v \in V(G)$ with a single **for**-loop by the arcs $(x, v) \in E(G)$.

Time complexity: $\Theta(mn)$ using adjacency lists vs. $\Theta(n^3)$ using an adjacency matrix.

Bellman-Ford Algorithm

Slower than Dijkstra's algorithm when all arcs are nonnegative.

Basic idea as in Dijkstra's: to find the single-source shortest paths (SSSP) under progressively relaxing restrictions.

- Dikstra's: one node a time based on their current distance estimate.
- Bellman-Ford: all nodes at "level" $0, 1, \ldots, n-1$ in turn.
  - Level of a node v – the minimum possible number of arcs in a minimum weight path to that node from the source s.

Theorem 6.9

If a graph G contains no negative weight cycles, then after the i^{th} iteration of the outer **for**-loop, the element $dist[v]$ contains the minimum weight of a path to v for all nodes v with level at most i.

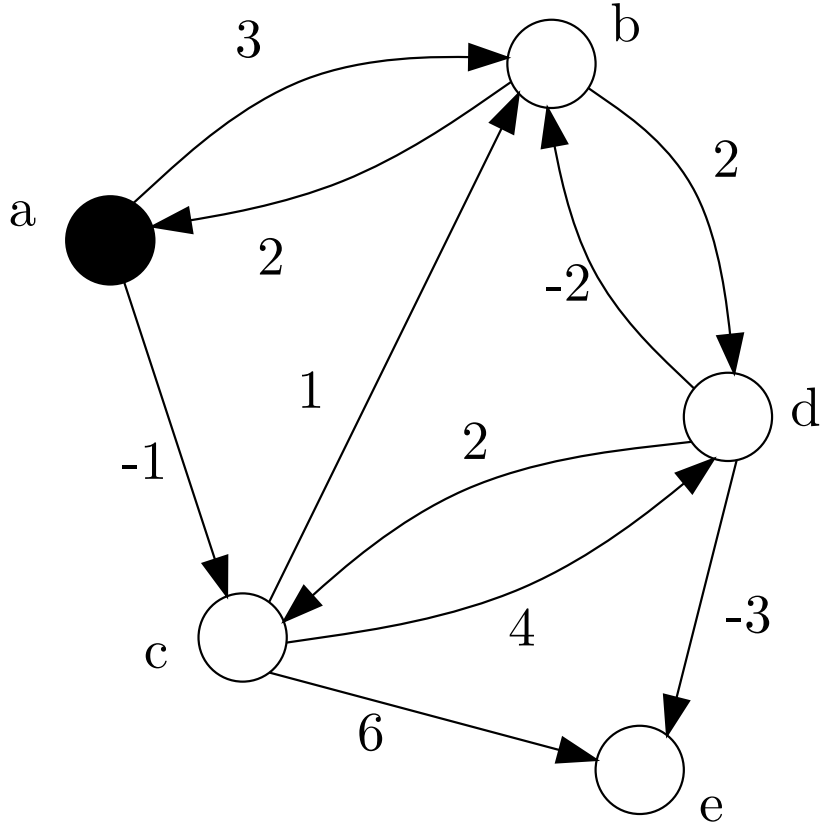
Proving Why Bellman-Ford Algorithm Works

Just as for Dijkstra's, the update ensures $dist[v]$ never increases.

Induction by the level i of the nodes:

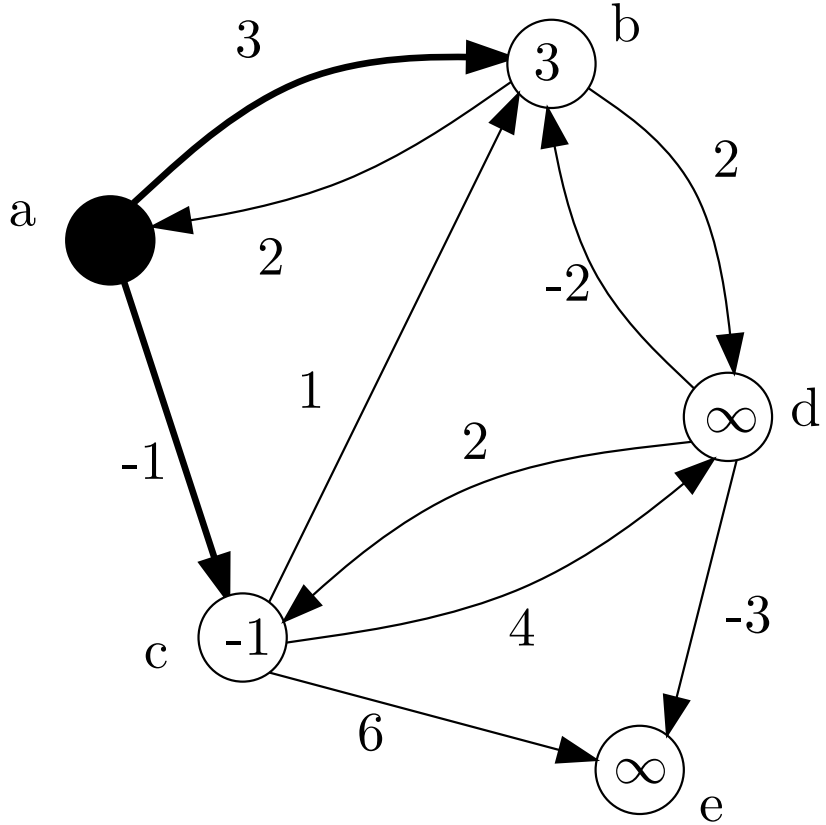
- **Base case**: $i = 0$; the result is true due to initialisation:
$$dist[s] = 0;\ dist[v] = \infty;\ v \in V \backslash s.$$
- **Induction hypothesis**: $dist[v]$; $v \in V$, are true for $i - 1$.
- **Induction step** for a node v at level i:
  - Due to no negative weight cycles, a min-weight s-to-v path, γ, has i arcs.
  - If y is the last node before v and γ_1 the subpath to y, then $dist[y] \leq |\gamma_1|$ by the induction hypothesis.
  - Thus by the update rule:
$$dist[v] \leq dist[y] + c(y, v) \leq |\gamma_1| + c(y, v) \leq |\gamma|$$
as required at level i.

Illustrating Bellman-Ford Algorithm



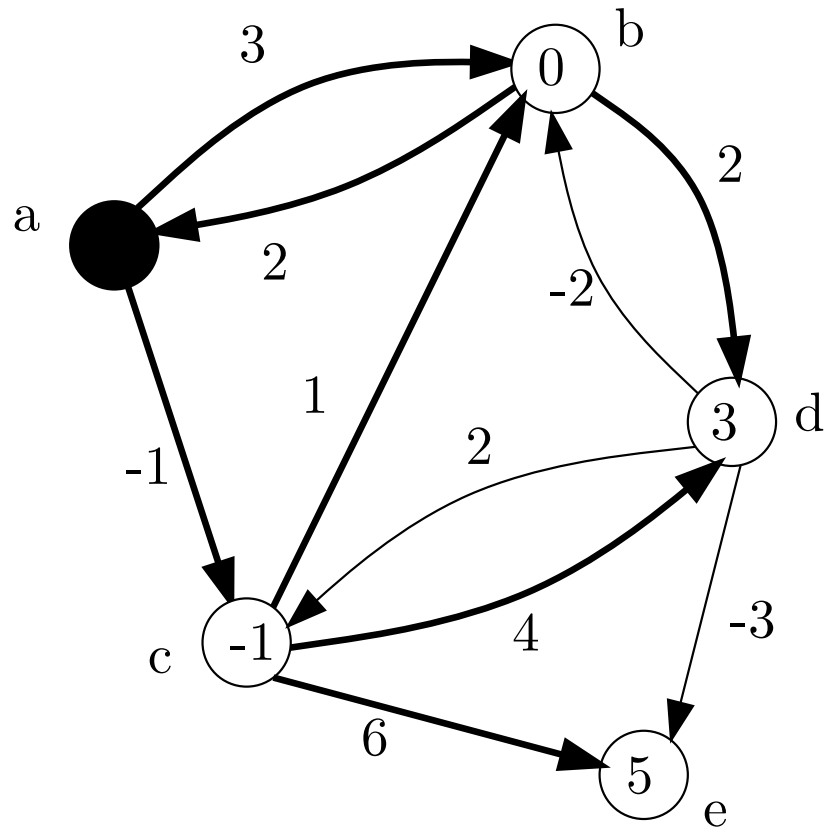
i	$dist[x]$				
	a	b	c	d	e
0	0	∞	∞	∞	∞
1	0	**3**	**−1**	∞	∞
2	0	**0**	−1	**3**	**5**
3	0	0	−1	**2**	**0**
4	0	0	−1	2	**−1**

Illustrating Bellman-Ford Algorithm



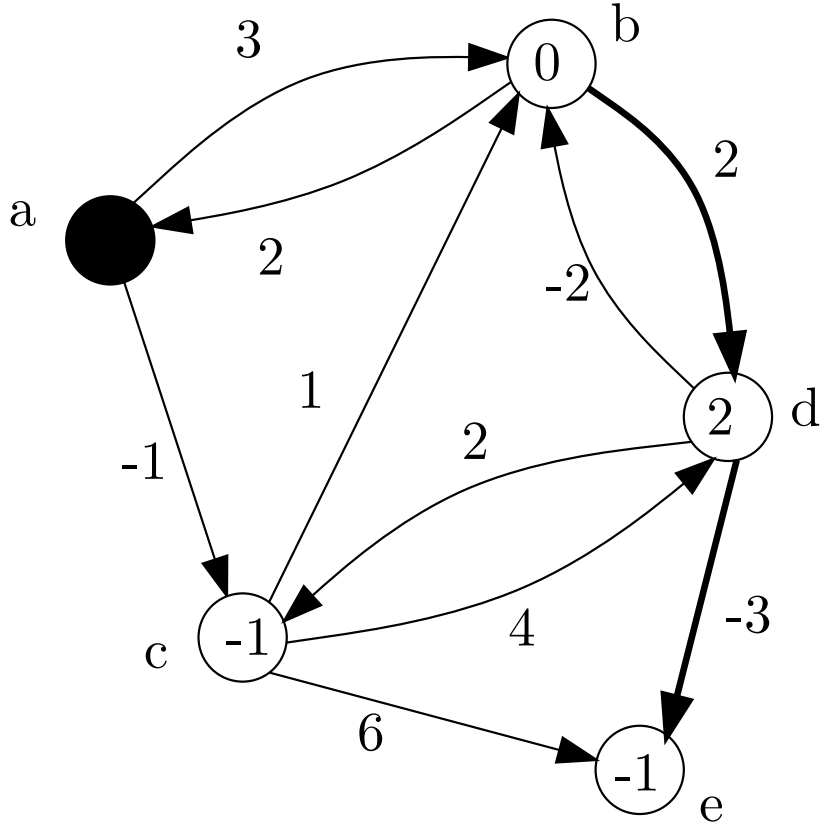
i	$dist[x]$				
	a	b	c	d	e
0	0	∞	∞	∞	∞
1	0	**3**	**−1**	∞	∞
2	0	**0**	−1	**3**	**5**
3	0	0	−1	**2**	**0**
4	0	0	−1	2	**−1**

Illustrating Bellman-Ford Algorithm



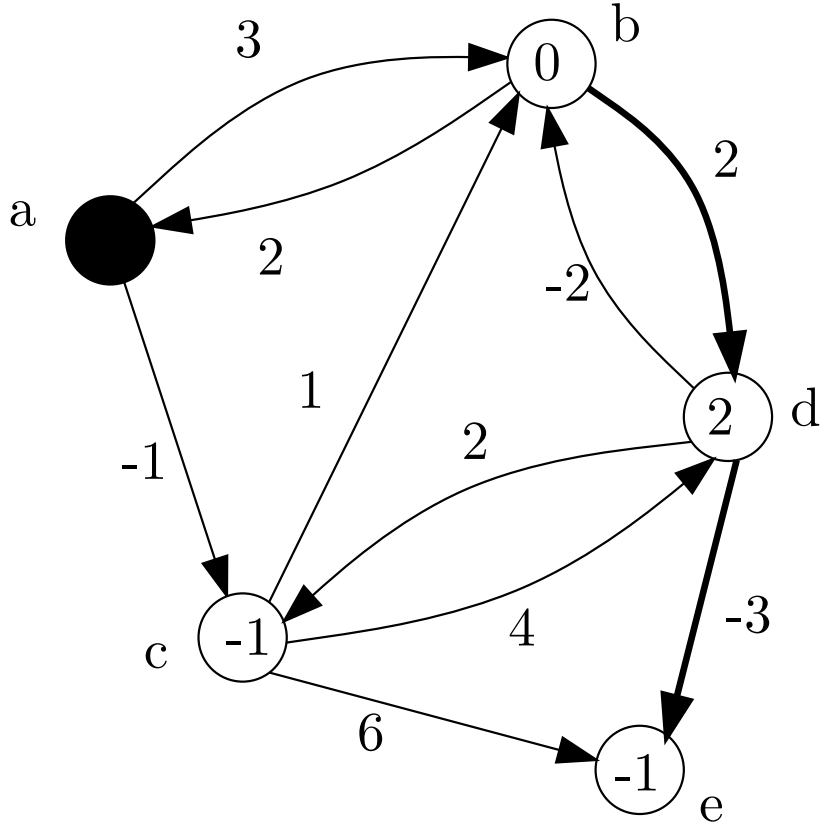
i	$dist[x]$				
	a	b	c	d	e
0	0	∞	∞	∞	∞
1	0	**3**	**−1**	∞	∞
2	0	**0**	−1	**3**	**5**
3	0	0	−1	**2**	**0**
4	0	0	−1	2	**−1**

Illustrating Bellman-Ford Algorithm



i	$dist[x]$				
	a	b	c	d	e
0	0	∞	∞	∞	∞
1	0	**3**	**−1**	∞	∞
2	0	**0**	−1	**3**	**5**
3	0	0	−1	**2**	**0**
4	0	0	−1	2	−1

Illustrating Bellman-Ford Algorithm



i	$dist[x]$				
	a	b	c	d	e
0	0	∞	∞	∞	∞
1	0	**3**	**−1**	∞	∞
2	0	**0**	−1	**3**	**5**
3	0	0	−1	**2**	**0**
4	0	0	−1	2	**−1**

Illustrating Bellman-Ford Algorithm (Alternative Form)

Arc (x, v):	a,b	a,c	b,a	b,d	c,b	c,d	c,e	d,b	d,c	d,e
$c(x, v)$:	3	-1	2	2	1	4	6	-2	2	-3

Iteration $i = 0$

x, v	Distance $d[v] \leftarrow \min\{d[v], d[x] + c(x, v)\}$						a	b	c	d	e
							0	∞	∞	∞	∞
a, b	d[b]	$\leftarrow$	$\min\{\infty,$	$0+3\}$	$=$	3	0	3	∞	∞	∞
a, c	d[c]	$\leftarrow$	$\min\{\infty,$	$0-1\}$	$=$	-1	0	3	-1	∞	∞
b, a	d[a]	$\leftarrow$	$\min\{0,$	$3+2\}$	$=$	0	0	3	-1	∞	∞
b, d	d[d]	$\leftarrow$	$\min\{\infty,$	$3+2\}$	$=$	5	0	3	-1	5	∞
c, b	d[b]	$\leftarrow$	$\min\{3,$	$-1+1\}$	$=$	0	0	0	-1	5	∞
c, d	d[d]	$\leftarrow$	$\min\{5,$	$-1+4\}$	$=$	3	0	0	-1	3	∞
c, e	d[e]	$\leftarrow$	$\min\{\infty,$	$-1+6\}$	$=$	5	0	0	-1	3	5
d, b	d[b]	$\leftarrow$	$\min\{0,$	$3-2\}$	$=$	0	0	0	-1	3	5
d, c	d[c]	$\leftarrow$	$\min\{-1,$	$3+2\}$	$=$	-1	0	0	-1	3	5
d, e	d[e]	$\leftarrow$	$\min\{5,$	$3-3\}$	$=$	0	0	0	-1	3	0

Illustrating Bellman-Ford Algorithm (Alternative Form)

Arc (x, v):	a,b	a,c	b,a	b,d	c,b	c,d	c,e	d,b	d,c	d,e
$c(x, v)$:	3	-1	2	2	1	4	6	-2	2	-3

Iteration $i = 1$

x, v	Distance $d[v] \leftarrow \min\{d[v], d[x] + c(x, v)\}$						a	b	c	d	e
							0	0	-1	3	0
a, b	d[b]	$\leftarrow$	$\min\{0,$	$0 + 3\}$	$=$	0	0	0	-1	3	0
a, c	d[c]	$\leftarrow$	$\min\{-1,$	$0 - 1\}$	$=$	-1	0	0	-1	3	0
b, a	d[a]	$\leftarrow$	$\min\{0,$	$0 + 2\}$	$=$	0	0	0	-1	3	0
b, d	d[d]	$\leftarrow$	$\min\{3,$	$0 + 2\}$	$=$	2	0	0	-1	2	0
c, b	d[b]	$\leftarrow$	$\min\{0,$	$-1 + 1\}$	$=$	0	0	0	-1	2	0
c, d	d[d]	$\leftarrow$	$\min\{2,$	$-1 + 4\}$	$=$	2	0	0	-1	2	0
c, e	d[e]	$\leftarrow$	$\min\{0,$	$-1 + 6\}$	$=$	0	0	0	-1	2	0
d, b	d[b]	$\leftarrow$	$\min\{0,$	$2 - 2\}$	$=$	0	0	0	-1	2	0
d, c	d[c]	$\leftarrow$	$\min\{-1,$	$2 + 2\}$	$=$	-1	0	0	-1	2	0
d, e	d[e]	$\leftarrow$	$\min\{0,$	$2 - 3\}$	$=$	-1	0	0	-1	2	-1

Illustrating Bellman-Ford Algorithm (Alternative Form)

Arc (x, v):	a,b	a,c	b,a	b,d	c,b	c,d	c,e	d,b	d,c	d,e
$c(x, v)$:	3	-1	2	2	1	4	6	-2	2	-3

Iteration $i = 2..4$

x, v	Distance $d[v] \leftarrow \min\{d[v], d[x] + c(x, v)\}$						a	b	c	d	e
							0	0	-1	2	-1
a, b	d[b]	$\leftarrow$	$\min\{0,$	$0 + 3\}$	$=$	0	0	0	-1	2	-1
a, c	d[c]	$\leftarrow$	$\min\{-1,$	$0 - 1\}$	$=$	-1	0	0	-1	2	-1
b, a	d[a]	$\leftarrow$	$\min\{0,$	$0 + 2\}$	$=$	0	0	0	-1	2	-1
b, d	d[d]	$\leftarrow$	$\min\{2,$	$0 + 2\}$	$=$	2	0	0	-1	2	-1
c, b	d[b]	$\leftarrow$	$\min\{0,$	$-1 + 1\}$	$=$	0	0	0	-1	2	-1
c, d	d[d]	$\leftarrow$	$\min\{2,$	$-1 + 4\}$	$=$	2	0	0	-1	2	-1
c, e	d[e]	$\leftarrow$	$\min\{-1,$	$-1 + 6\}$	$=$	-1	0	0	-1	2	-1
d, b	d[b]	$\leftarrow$	$\min\{0,$	$3 - 2\}$	$=$	0	0	0	-1	2	-1
d, c	d[c]	$\leftarrow$	$\min\{-1,$	$3 + 2\}$	$=$	-1	0	0	-1	2	-1
d, e	d[e]	$\leftarrow$	$\min\{-1,$	$3 - 3\}$	$=$	-1	0	0	-1	2	-1

Comments on Bellman-Ford Algorithm

- This (non-greedy) algorithm handles negative weight arcs, but not negative weight cycles.
- Running time with the two innermost nested **for**-loops: $\mathrm{O}(n^3)$.
  - Runs slower than the Dijkstra's algorithm since considers all nodes at "level" $i = 0, 1, \ldots, n-1$, in turn.
- The alternative form where the two inner-most **for**-loops are replaced with: **for** $(u, v) \in E(V)$ runs in time $O(nm)$.
  - The outer **for**-loop (by i) in this case can be terminated after no distance changes during the iteration (e.g., after $i = 2$ in the example on Slide 39).
- Bellman-Ford algorithm can be modified to detect negative weight cycle (see Textbook, Exercise 6.3.4)

All Pairs Shortest Path (APSP) Problem

Given a weighted digraph (G, c), determine for each pair of nodes $u, v \in V(G)$ (the length of) a minimum weight path from u to v.

Convenient output: a distance matrix $D = \big[D[u, v]\big]_{u,v \in V(G)}$

- Time complexity $\Theta(nA_{n,m})$ of computing the matrix D by finding the single-source shortest paths (SSSP) from each node as the source in turn.
  - $A_{n=|V(G)|,m=|E(G)|}$ – the complexity of the SSSP algorithm.
  - The APSP complexity $\Theta(n^3)$ for the adjacency matrix version of the Dijkstra's SSSP algorithm: $A_{n,m} = n^2$.
  - The APSP complexity $\Theta(n^2 m)$ for the Bellman-Ford SSSP algorithm: $A_{n,m} = mn$.

All Pairs Shortest Path (APSP) Problem

Floyd's algorithm – one of the known simpler algorithms for computing the distance matrix (three nested **for**-loops; $\Theta(n^3)$ time complexity):

1. Number all nodes (say, from 0 to $n-1$).
2. At each step k, maintain the matrix of shortest distances from node i to node j, not passing through nodes higher than k.
3. Update the matrix at each step to see whether the node k shortens the current best distance.

An alternative to running the SSSP algorithm from each node.

- Better than the Dijkstra's algorithm for dense graphs, probably not for sparse ones.
- Unlike the Dijkstra's algorithm, can handle negative costs.
- Based on Warshall's algorithm (just tells whether there is a path from node i to node j, not concerned with length).

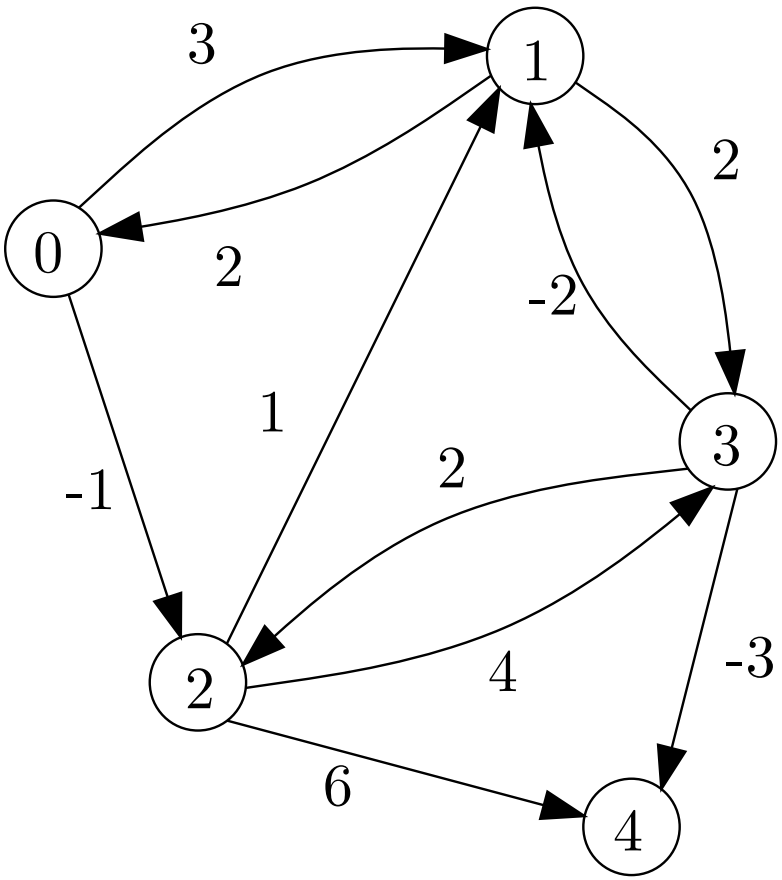
Floyd's Algorithm

algorithm `Floyd`(weighted digraph (G, c))
Initialisation: **for** $u, v \in V(G)$ **do** $D[u, v] \leftarrow c(u, v)$ **end for**
for $x \in V(G)$ **do**
  for $u \in V(G)$ **do**
    for $v \in V(G)$ **do**
      $D[u, v] \leftarrow \min\{D[u, v], D[u, x] + D[x, v]\}$
    end for
  end for
end for

This algorithm is based on **dynamic programming** principles.

At the bottom of the outer **for**-x-loop, $D[u, v]$ for each $u, v \in V(G)$ is the length of the shortest path from u to v passing through intermediate nodes x having been seen in that loop.

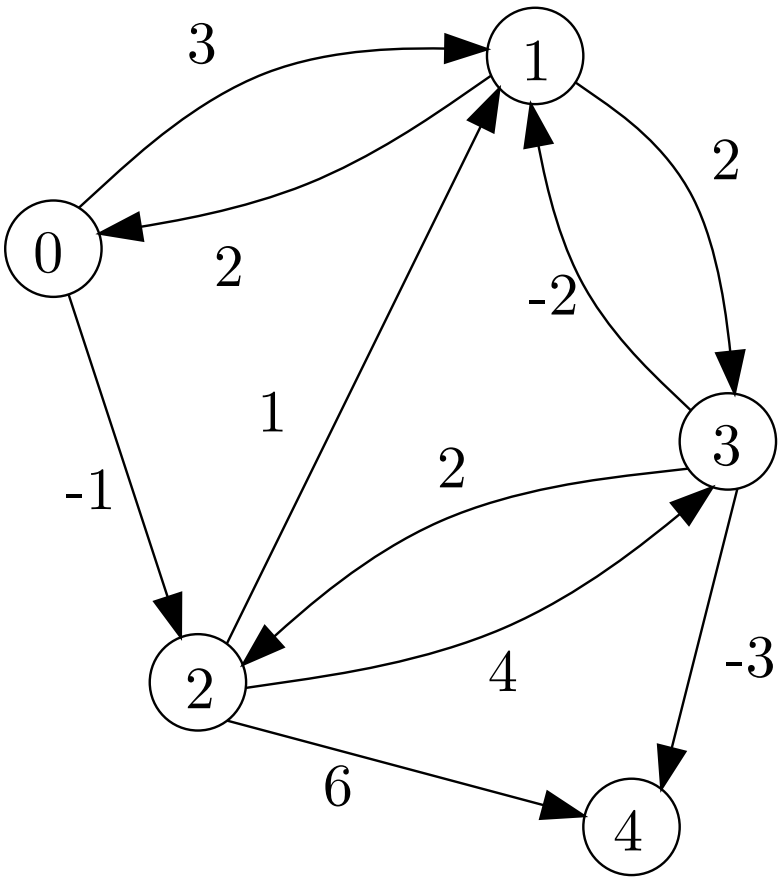
Illustrating Floyd's Algorithm



$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & -1 & \infty & \infty \\ 2 & 0 & \infty & 2 & \infty \\ \infty & 1 & 0 & 4 & 6 \\ \infty & -2 & 2 & 0 & -3 \\ \infty & \infty & \infty & \infty & 0 \end{bmatrix} \end{array}$$

Adjacency/cost matrix $c[u, v]$

Illustrating Floyd's Algorithm: $x = 0$

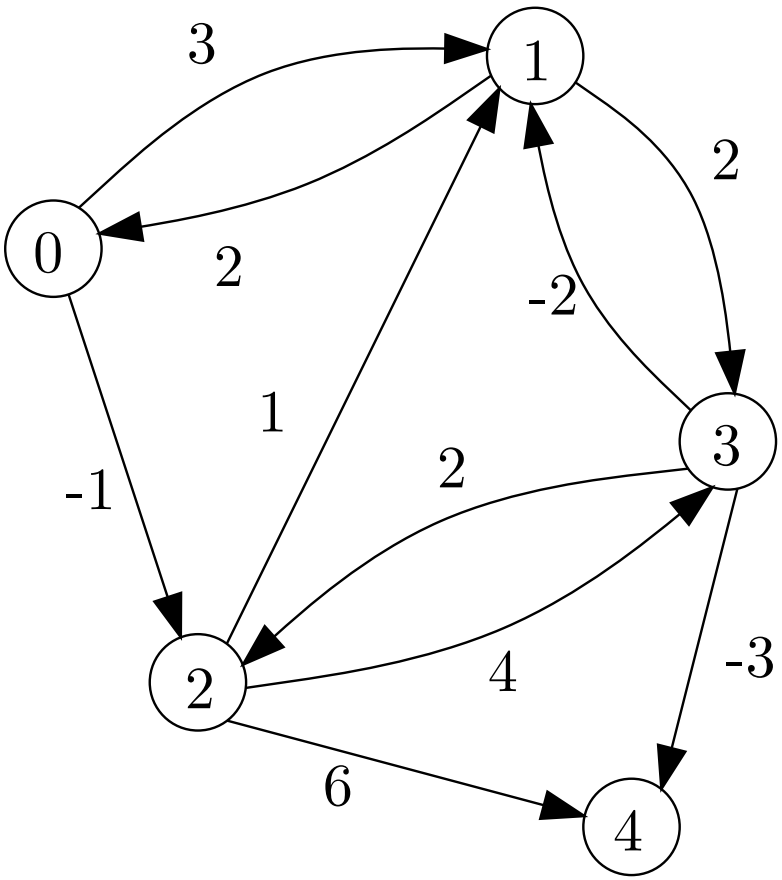


	0	1	2	3	4
0	0	3	-1	∞	∞
1	2	0	**1**	2	∞
2	∞	1	0	4	6
3	∞	-2	2	0	-3
4	∞	∞	∞	∞	0

Distance matrix $D_0[u, v]$

$D_0[1, 2] = \min\{\infty, 2_{c[1,0]} - 1_{c[0,1]}\} = 1$

Illustrating Floyd's Algorithm: $x = 1$

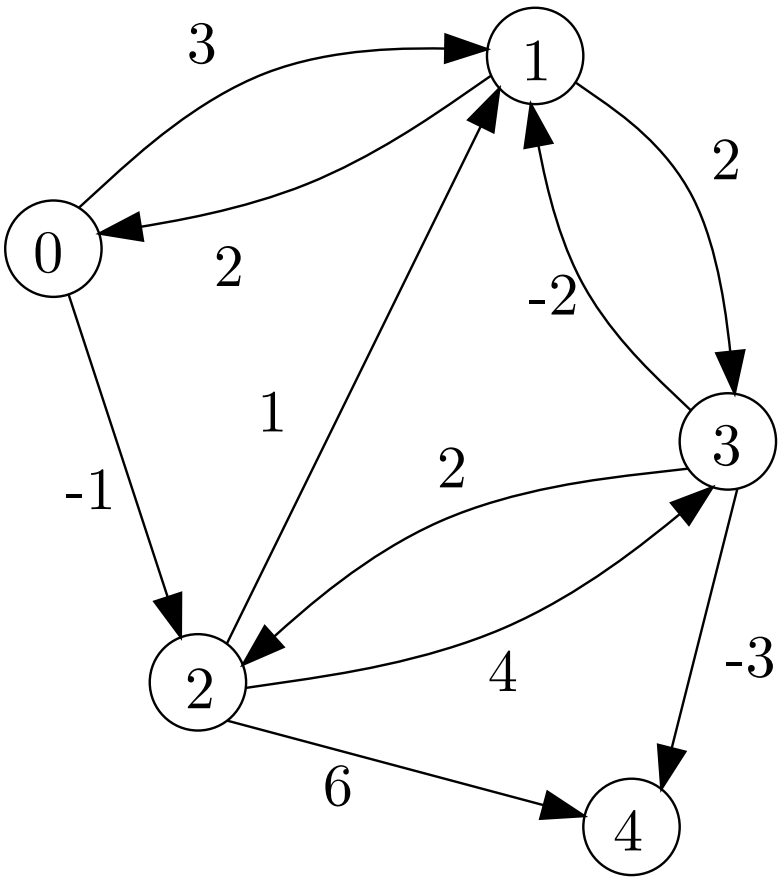


$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\ \left[\begin{array}{ccccc} 0 & 3 & -1 & \mathbf{5} & \infty \\ \hline 2 & 0 & 1 & 2 & \infty \\ \hline \mathbf{3} & 1 & 0 & \mathbf{3} & 6 \\ \mathbf{0} & -2 & \mathbf{-1} & 0 & -3 \\ \infty & \infty & \infty & \infty & 0 \end{array}\right] \end{array}$$

Distance matrix $D_1[u, v]$

$$D_1[0,3] = \min\{\infty, 3_{D_0[0,1]} + 2_{D_0[1,3]}\} = 5$$
$$D_1[2,3] = \min\{4, 1_{D_0[2,1]} + 2_{D_0[1,3]}\} = 3$$
$$D_1[3,2] = \min\{2, -2_{D_0[3,1]} + 1_{D_0[1,2]}\} = -1$$

Illustrating Floyd's Algorithm: $x = 2$

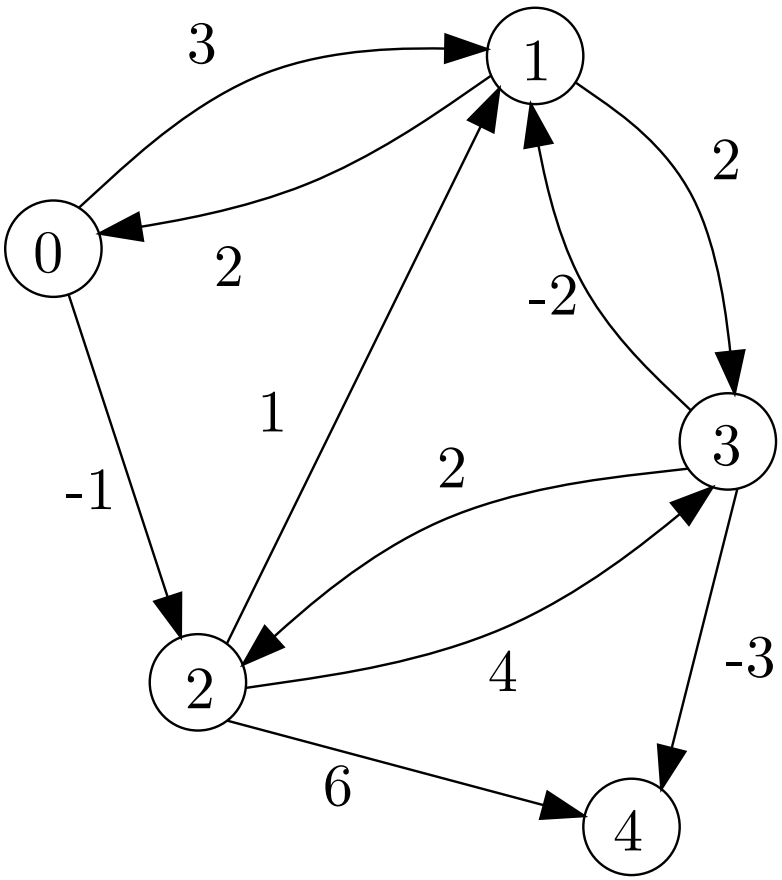


$$
\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array}
\begin{array}{c}
\begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\
\left[\begin{array}{ccc|cc}
0 & \mathbf{0} & -1 & \mathbf{2} & \mathbf{5} \\
2 & 0 & 1 & 2 & \mathbf{7} \\
\hline
3 & 1 & 0 & 3 & 6 \\
\hline
0 & -2 & -1 & 0 & -3 \\
\infty & \infty & \infty & \infty & 0
\end{array}\right]
\end{array}
$$

Distance matrix $D_2[u, v]$

$$D_2[0,1] = \min\{3, -1_{D_1[0,2]} + 1_{D_1[2,1]}\} = 0$$
$$D_2[0,3] = \min\{5, -1_{D_1[0,2]} + 3_{D_1[2,3]}\} = 2$$
$$D_2[0,4] = \min\{\infty, -1_{D_1[0,2]} + 6_{D_1[2,4]}\} = 5$$
$$D_2[1,4] = \min\{\infty, 1_{D_1[1,2]} + 6_{D_1[2,4]}\} = 7$$

Illustrating Floyd's Algorithm: $x = 3$

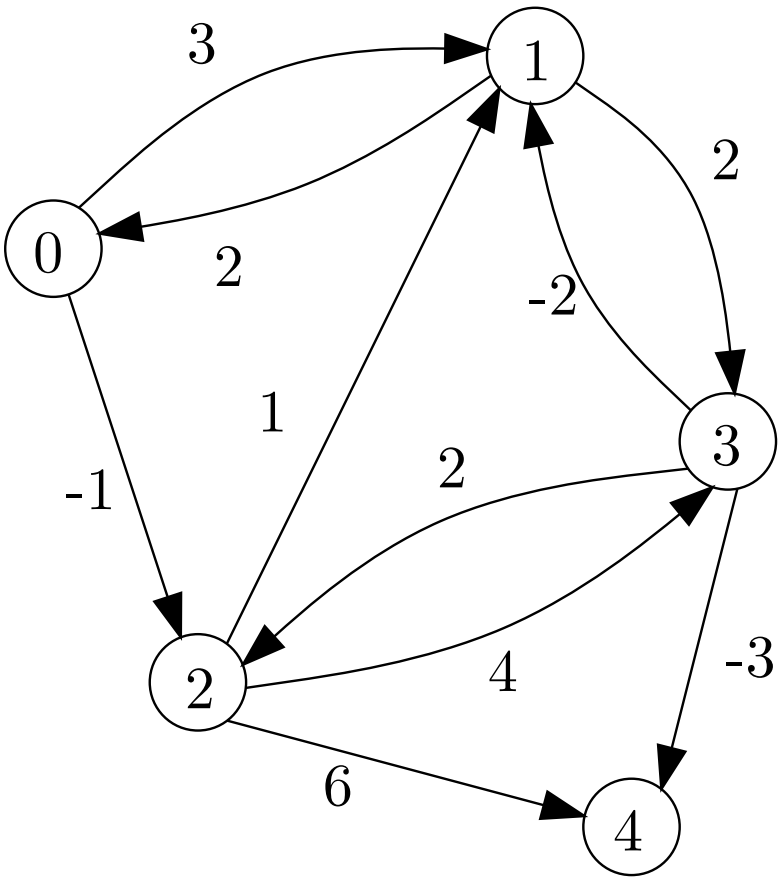


$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\ \left[\begin{array}{ccccc} 0 & 0 & -1 & 2 & \mathbf{-1} \\ 2 & 0 & 1 & 2 & \mathbf{-1} \\ 3 & 1 & 0 & 3 & \mathbf{0} \\ \hline 0 & -2 & -1 & 0 & -3 \\ \hline \infty & \infty & \infty & \infty & 0 \end{array}\right] \end{array}$$

Distance matrix $D_3[u, v]$

$$D_3[0,4] = \min\{5, 2_{D_2[0,3]} - 3_{D_2[3,4]}\} = -1$$
$$D_3[1,4] = \min\{7, 2_{D_1[1,3]} - 3_{D_1[3,4]}\} = -1$$
$$D_3[2,4] = \min\{6, 3_{D_1[2,3]} - 3_{D_1[3,4]}\} = 0$$

Illustrating Floyd's Algorithm: $x = 4$



$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 0 & -1 & 2 & -1 \\ 2 & 0 & 1 & 2 & -1 \\ 3 & 1 & 0 & 3 & 0 \\ 0 & -2 & -1 & 0 & -3 \\ \infty & \infty & \infty & \infty & 0 \end{bmatrix} \end{array}$$

Final distance matrix $D \equiv D_4[u, v]$

Proving Why Floyd's Algorithm Works

Theorem 6.12: At the bottom of the outer **for**-loop, for all nodes u and v, $D[u, v]$ contains the minimum length of all paths from u to v that are restricted to using only intermediate nodes that have been seen in the outer **for**-loop.

When algorithm terminates, all nodes have been seen and $D[u, v]$ is the length of the shortest u-to-v path.

Notation: S_k – the set of nodes seen after k passes through this loop; S_k-path – one with all intermediate nodes in S_k; D_k – the corresponding value of D.

Induction on the outer **for**-loop:

- **Base case**: $k = 0$; $S_0 = \emptyset$, and the result holds.
- **Induction hypothesis**: It holds after $k \geq 0$ times through the loop.
- **Inductive step**: To show that $D_{k+1}[u, v]$ after $k + 1$ passes through this loop is the minimum length of an u-to-v S_{k+1}-path.

Proving Why Floyd's Algorithm Works

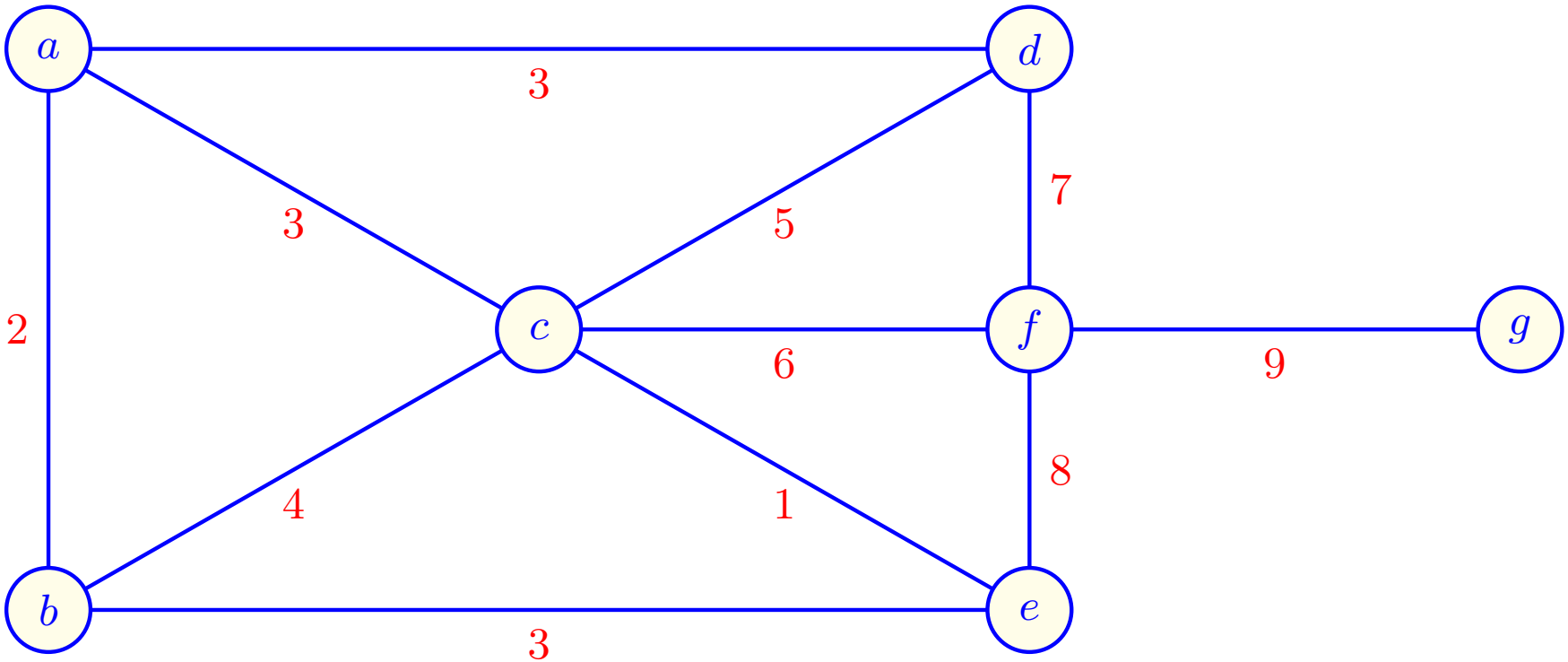
Inductive step:
Suppose that x is the last node seen in the loop, so $S_{k+1} = S_k \bigcup \{x\}$.

- Fix an arbitrary pair of nodes $u, v \in V(G)$ and let L be the min-length of an u-to-v S_{k+1}-path, so that obviously $L \leq D_{k+1}[u, v]$.
- To show that also $D_{k+1}[u, v] \leq L$, choose an u-to-v S_{k+1}-path γ of length L. If $x \notin \gamma$, the result follows from the induction hypothesis.
- If $x \in \gamma$, let γ_1 and γ_2 be, respectively, the u-to-x and x-to-v subpaths. Then γ_1 and γ_2 are S_k-paths and by the inductive hypothesis,

$$L \geq |\gamma_1| + |\gamma_2| \geq D_k[u, x] + D_k[x, v] \geq D_{k+1}[u, v]$$

Non-negativity of the weights is not used in the proof, and Floyd's algorithm works for negative weights (but negative weight cycles should not be present).

Floyd's Algorithm: Example 2



Computing all-pairs shortest paths

Floyd's Algorithm: Example 2

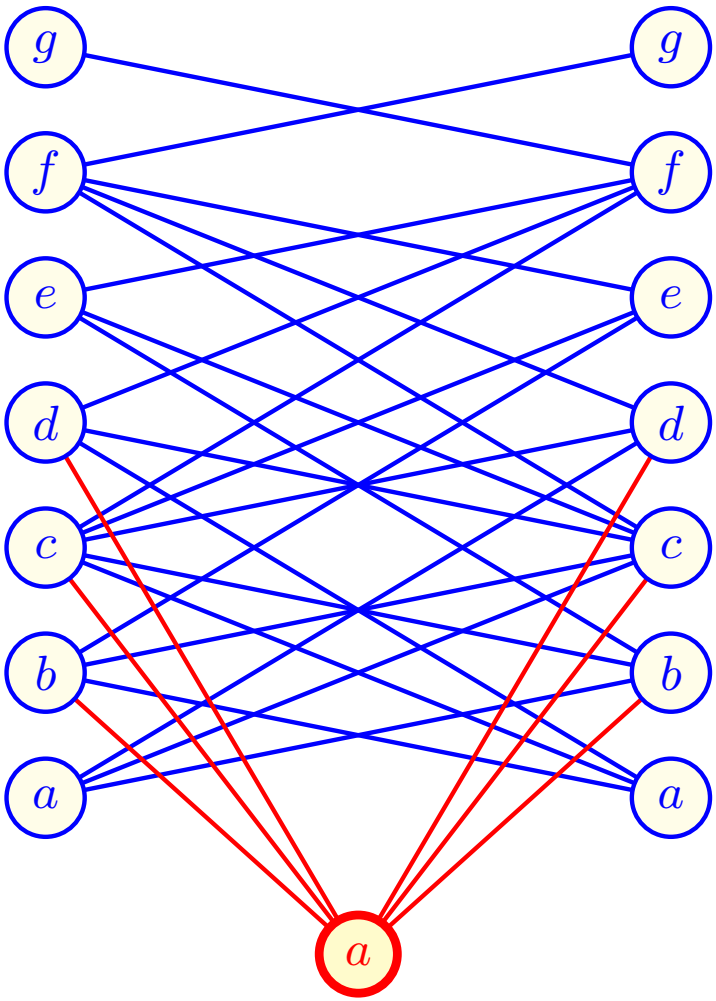
Initialisation

$$\big[D[u,v]\big]_{u,v\in V(G)} \leftarrow \overbrace{\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \end{array}\begin{bmatrix} 0 & 2 & 3 & 3 & \infty & \infty & \infty \\ 2 & 0 & 4 & \infty & 3 & \infty & \infty \\ 3 & 4 & 0 & 5 & 1 & 6 & \infty \\ 3 & \infty & 5 & 0 & \infty & 7 & \infty \\ \infty & 3 & 1 & \infty & 0 & 8 & \infty \\ \infty & \infty & 6 & 7 & 8 & 0 & 9 \\ \infty & \infty & \infty & \infty & \infty & 9 & 0 \end{bmatrix}}^{\text{Initialisation: } c(u,v)]}$$
$$\begin{array}{ccccccc} a & b & c & d & e & f & g \end{array}$$

for $x \in V = \{a, b, c, d, e, f, g\}$ **do**
  for $u \in V = \{a, b, c, d, e, f, g\}$ **do**
    for $v \in V = \{a, b, c, d, e, f, g\}$ **do**
      $D[u, v] \leftarrow \min\{D[u, v],\ D[u, x] + D[x, v]\}$
    end for
  end for
end for

Floyd's Algorithm: Example 2

$x \leftarrow a$



$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \end{array} \begin{array}{c} \begin{array}{ccccccc} a & b & c & d & e & f & g \end{array} \\ \underbrace{\begin{bmatrix} 0 & 2 & 3 & 3 & \infty & \infty & \infty \\ 2 & 0 & 4 & 5 & 3 & \infty & \infty \\ 3 & 4 & 0 & 5 & 1 & 6 & \infty \\ 3 & 5 & 5 & 0 & \infty & 7 & \infty \\ \infty & 3 & 1 & \infty & 0 & 8 & \infty \\ \infty & \infty & 6 & 7 & 8 & 0 & 9 \\ \infty & \infty & \infty & \infty & \infty & 9 & 0 \end{bmatrix}}_{} \end{array}$$

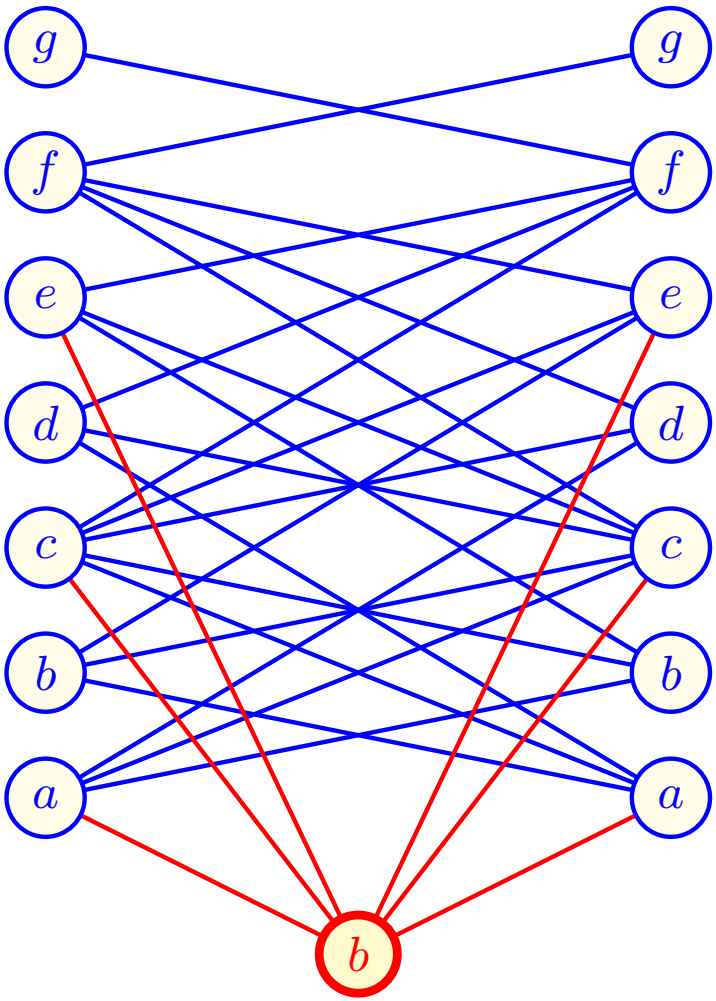
$$D[u,v] \leftarrow \min\{D[u,v],\ D[u,a] + D[a,v]\}\,; \quad (u,v) \in V^2$$

E.g.,

$$\begin{aligned} D[b,d] &\leftarrow \min\{D[b,d], D[b,a] + D[a,d]\} \\ &= \min\{\infty, 2+3\} = 5 \end{aligned}$$

Floyd's Algorithm: Example 2

$x \leftarrow b$



$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \end{array} \underbrace{\begin{array}{c} \begin{array}{ccccccc} a & b & c & d & e & f & g \end{array} \\ \left[\begin{array}{ccccccc} 0 & 2 & 3 & 3 & 5 & \infty & \infty \\ 2 & 0 & 4 & 5 & 3 & \infty & \infty \\ 3 & 4 & 0 & 5 & 1 & 6 & \infty \\ 3 & 5 & 5 & 0 & 8 & 7 & \infty \\ 5 & 3 & 1 & 8 & 0 & 8 & \infty \\ \infty & \infty & 6 & 7 & 8 & 0 & 9 \\ \infty & \infty & \infty & \infty & \infty & 9 & 0 \end{array}\right] \end{array}}$$

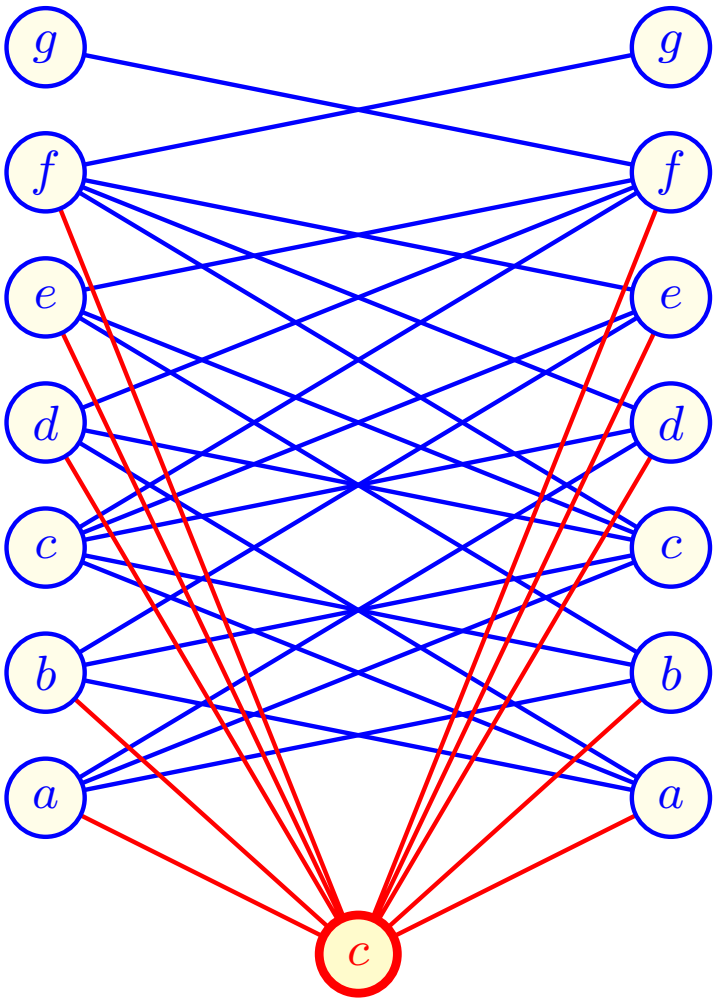
$$D[u,v] \leftarrow \min\{D[u,v],\ D[u,b]+D[b,v]\}\,; \quad (u,v) \in V^2$$

E.g.,

$$\begin{aligned} D[a,e] &\leftarrow \min\{D[a,e], D[a,b]+D[b,e]\} \\ &= \min\{\infty, 2+3\} = 5 \end{aligned}$$

Floyd's Algorithm: Example 2

$x \leftarrow c$



$$\underbrace{\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \end{array}\!\!\begin{array}{c} \begin{array}{ccccccc} a & b & c & d & e & f & g \end{array} \\ \left[\begin{array}{ccccccc} 0 & 2 & 3 & 3 & 4 & 9 & \infty \\ 2 & 0 & 4 & 5 & 3 & 10 & \infty \\ 3 & 4 & 0 & 5 & 1 & 6 & \infty \\ 3 & 5 & 5 & 0 & 6 & 7 & \infty \\ 4 & 3 & 1 & 6 & 0 & 7 & \infty \\ 9 & 10 & 6 & 7 & 7 & 0 & 9 \\ \infty & \infty & \infty & \infty & \infty & 9 & 0 \end{array}\right] \end{array}}$$

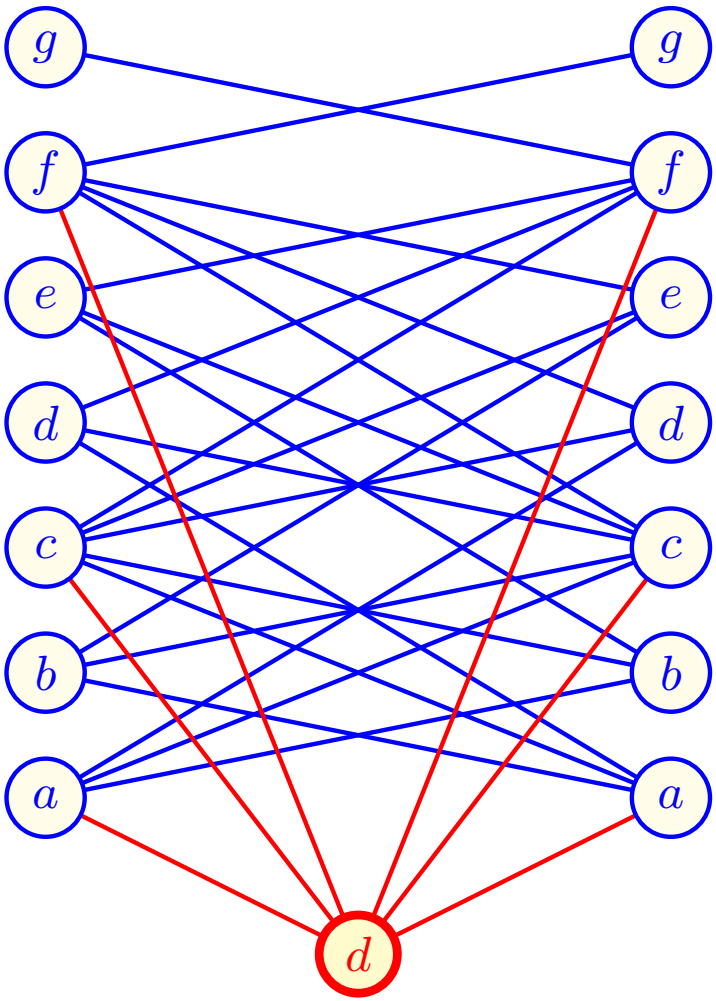
$$D[u,v] \leftarrow \min\{D[u,v],\ D[u,c]+D[c,v]\}\,; \quad (u,v) \in V^2$$

E.g.,

$$\begin{aligned} D[a,f] &\leftarrow \min\{D[a,f], D[a,c]+D[c,f]\} \\ &= \min\{\infty, 3+6\} = 9 \end{aligned}$$

Floyd's Algorithm: Example 2

$x \leftarrow d$



$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \end{array} \underbrace{\begin{array}{c} \begin{array}{ccccccc} a & b & c & d & e & f & g \end{array} \\ \left[\begin{array}{ccccccc} 0 & 2 & 3 & 3 & 4 & 9 & \infty \\ 2 & 0 & 4 & 5 & 3 & 10 & \infty \\ 3 & 4 & 0 & 5 & 1 & 6 & \infty \\ 3 & 5 & 5 & 0 & 8 & 7 & \infty \\ 4 & 3 & 1 & 8 & 0 & 7 & \infty \\ 9 & 10 & 6 & 7 & 7 & 0 & 9 \\ \infty & \infty & \infty & \infty & \infty & 9 & 0 \end{array}\right] \end{array}}$$

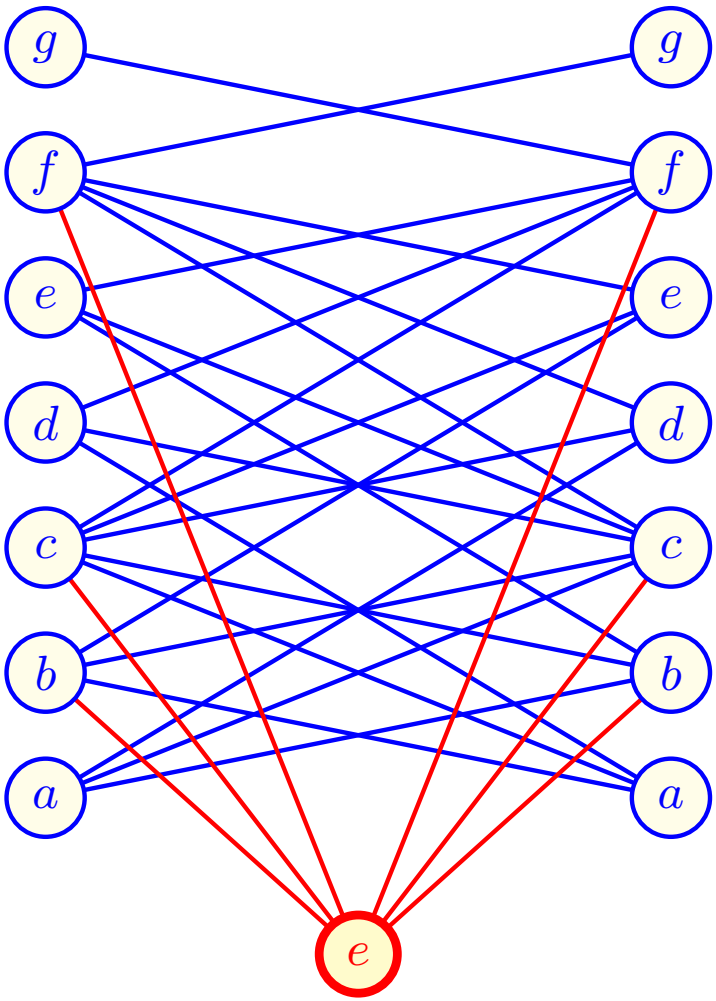
$$D[u,v] \leftarrow \min\{D[u,v],\ D[u,d] + D[d,v]\}\,; \quad (u,v) \in V^2$$

E.g.,

$$\begin{aligned} D[a,f] &\leftarrow \min\{D[a,f], D[a,d] + D[d,f]\} \\ &= \min\{9, 3+7\} = 9 \end{aligned}$$

Floyd's Algorithm: Example 2

$x \leftarrow e$



$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \end{array} \underbrace{\begin{array}{c} \begin{array}{ccccccc} a & b & c & d & e & f & g \end{array} \\ \left[\begin{array}{ccccccc} 0 & 2 & 3 & 3 & 4 & 9 & \infty \\ 2 & 0 & 4 & 5 & 3 & 10 & \infty \\ 3 & 4 & 0 & 5 & 1 & 6 & \infty \\ 3 & 5 & 5 & 0 & 8 & 7 & \infty \\ 4 & 3 & 1 & 8 & 0 & 7 & \infty \\ 9 & 10 & 6 & 7 & 7 & 0 & 9 \\ \infty & \infty & \infty & \infty & \infty & 9 & 0 \end{array}\right] \end{array}}$$

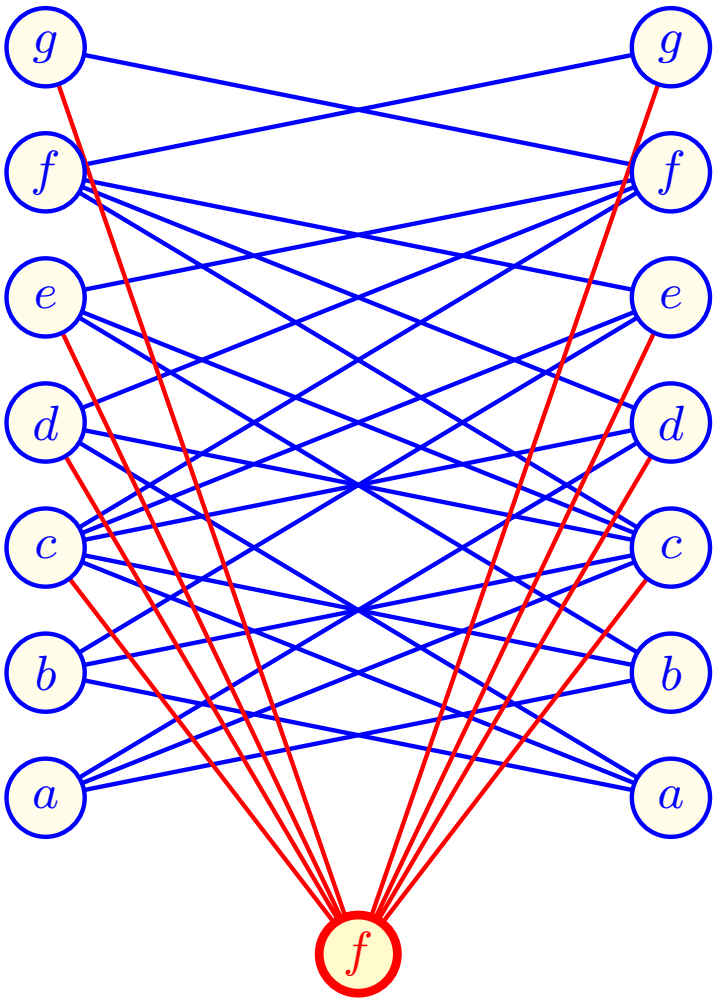
$$D[u,v] \leftarrow \min\{D[u,v],\ D[u,e] + D[e,v]\}\,; \quad (u,v) \in V^2$$

E.g.,

$$\begin{aligned} D[b,f] &\leftarrow \min\{D[b,f], D[b,e] + D[e,f]\} \\ &= \min\{9, 3+7\} = 9 \end{aligned}$$

Floyd's Algorithm: Example 2

$x \leftarrow f$



$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \end{array} \overbrace{\begin{array}{c} \begin{array}{ccccccc} a & b & c & d & e & f & g \end{array} \\ \left[\begin{array}{ccccccc} 0 & 2 & 3 & 3 & 4 & 9 & 18 \\ 2 & 0 & 4 & 5 & 3 & 10 & 19 \\ 3 & 4 & 0 & 5 & 1 & 6 & 15 \\ 3 & 5 & 5 & 0 & 8 & 7 & 16 \\ 4 & 3 & 1 & 8 & 0 & 7 & 16 \\ 9 & 10 & 6 & 7 & 7 & 0 & 9 \\ 18 & 19 & 15 & 16 & 16 & 9 & 0 \end{array}\right] \end{array}}$$

$$D[u,v] \leftarrow \min\{D[u,v],\ D[u,f] + D[f,v]\}\,; \quad (u,v) \in V^2$$

E.g.,

$$\begin{aligned} D[a,g] &\leftarrow \min\{D[a,g], D[a,f] + D[f,g]\} \\ &= \min\{\infty, 9+9\} = 18 \end{aligned}$$

Computing Actual Shortest Paths

- In addition to knowing the shortest distances, the shortest paths are often to be reconstructed.
- The Floyd's algorithm can be enhanced to compute also the **predecessor matrix** $\Pi = [\pi_{ij}]_{i,j=1,1}^{n,n}$ where vertex $\pi_{i,j}$ precedes vertex j on a shortest path from vertex i; $1 \leq i, j \leq n$.

Compute a sequence $\Pi^{(0)}, \Pi^{(1)}, \ldots \Pi^{(n)}$,

where vertex $\pi_{i,j}^{(k)}$ precedes the vertex j on a shortest path from vertex i with all intermediate vertices in $V_{(k)} = \{1, 2, \ldots, k\}$.

For case of no intermediate vertices:

$$\pi_{i,j}^{(0)} = \begin{cases} \text{NIL} & \text{if } i = j \text{ or } c[i,j] = \infty \\ i & \text{if } i \neq j \text{ and } c[i,j] < \infty \end{cases}$$

Computing Actual Shortest Paths

- In addition to knowing the shortest distances, the shortest paths are often to be reconstructed.
- The Floyd's algorithm can be enhanced to compute also the **predecessor matrix** $\Pi = [\pi_{ij}]_{i,j=1,1}^{n,n}$ where vertex $\pi_{i,j}$ precedes vertex j on a shortest path from vertex i; $1 \leq i, j \leq n$.

Compute a sequence $\Pi^{(0)}, \Pi^{(1)}, \ldots \Pi^{(n)}$,

where vertex $\pi_{i,j}^{(k)}$ precedes the vertex j on a shortest path from vertex i with all intermediate vertices in $V_{(k)} = \{1, 2, \ldots, k\}$.

For case of no intermediate vertices:

$$\pi_{i,j}^{(0)} = \begin{cases} \text{NIL} & \text{if } i = j \text{ or } c[i,j] = \infty \\ i & \text{if } i \neq j \text{ and } c[i,j] < \infty \end{cases}$$

Floyd's Algorithm with Predecessors

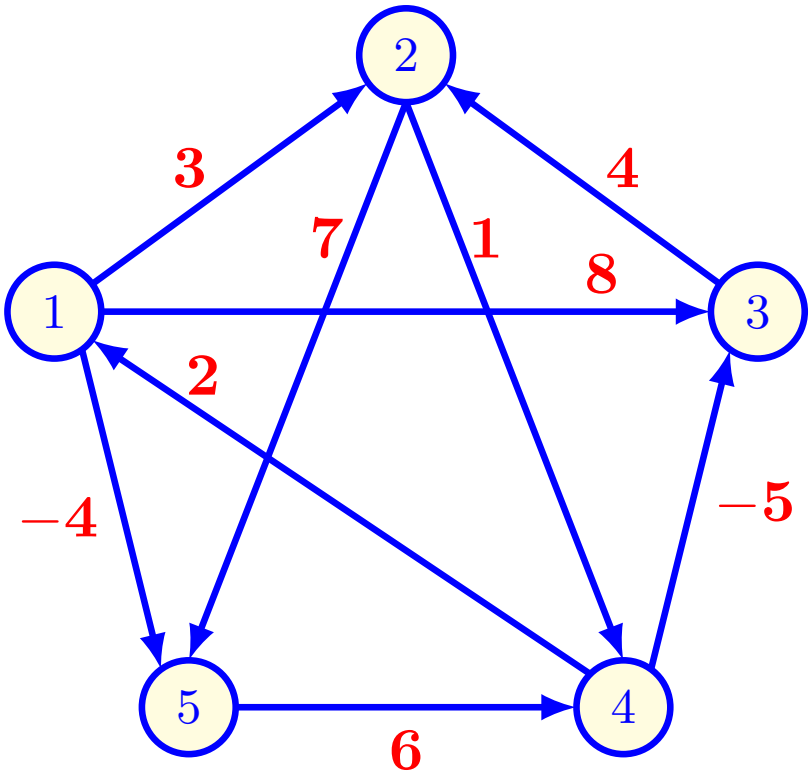
algorithm `FloydPred`(weighted digraph (G, c))

$D \leftarrow c$ Create initial distance matrix from weights.

$\Pi \leftarrow \Pi^{(0)}$ Initialize predecessors from c as in Slide 60.

for k **from** 1 **to** n **do**
  for i **from** 1 **to** n **do**
    for j **from** 1 **to** n **do**
      if $D[i,j] > D[i,k] + D[k,j]$ **then**
        $D[i,j] \leftarrow D[i,k] + D[k,j]; \quad \Pi[i,j] \leftarrow \Pi[k,j]$
      end if
    end for
  end for
end for

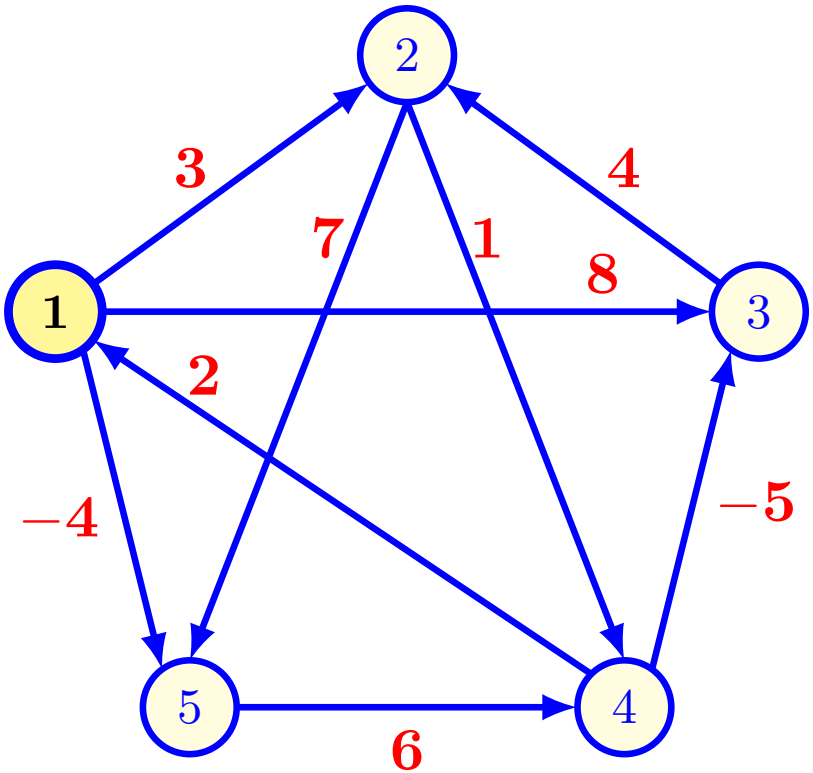
Illustrating Floyd's Algorithm with Predecessors



$$D^{(0)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \end{array}$$

$$\Pi^{(0)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{bmatrix} \end{array}$$

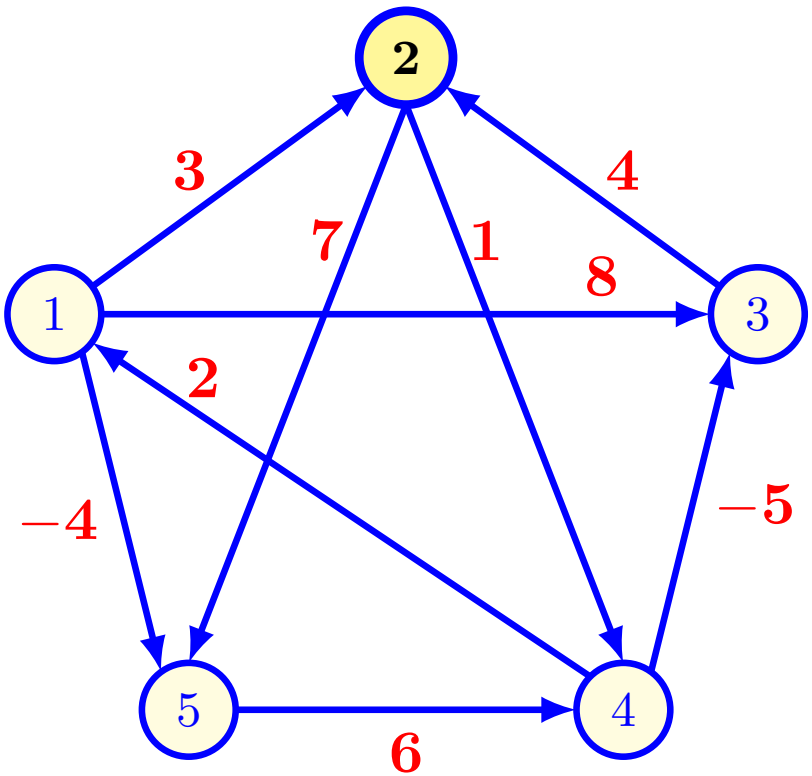
Illustrating Floyd's Algorithm with Predecessors: $k = 1$



$$D^{(1)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \end{array}$$

$$\Pi^{(1)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{bmatrix} \end{array}$$

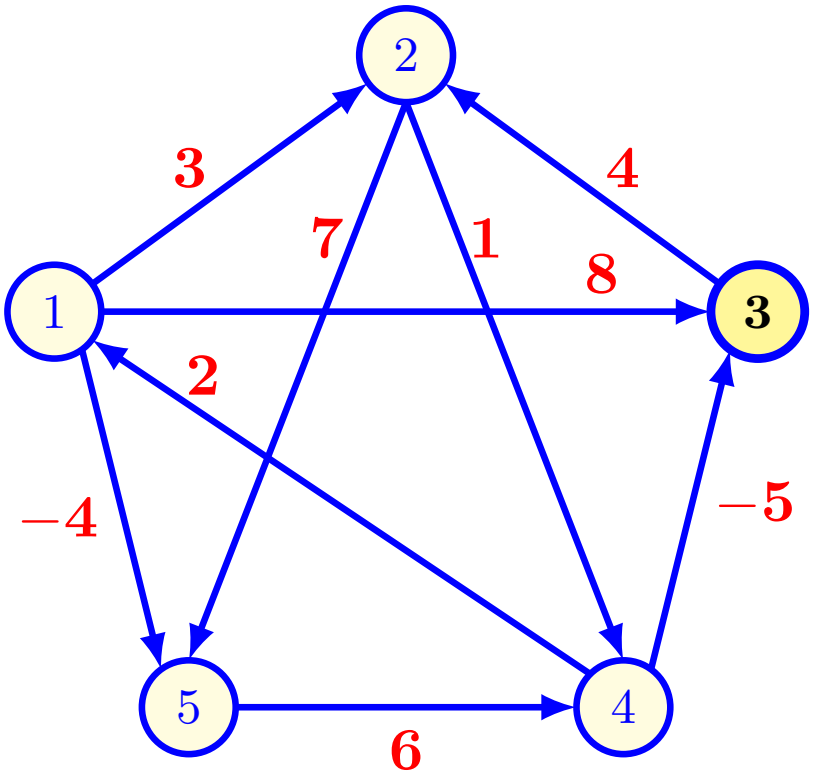
Illustrating Floyd's Algorithm with Predecessors: $k = 2$



$$D^{(2)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \end{array}$$

$$\Pi^{(2)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{bmatrix} \end{array}$$

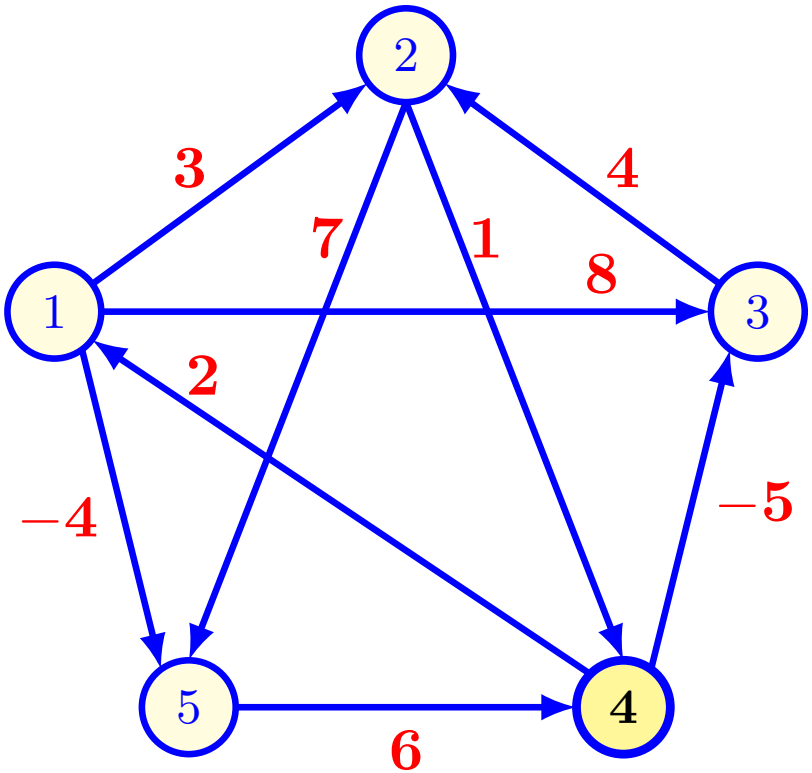
Illustrating Floyd's Algorithm with Predecessors: $k = 3$



$$D^{(3)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \end{array}$$

$$\Pi^{(3)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{bmatrix} \end{array}$$

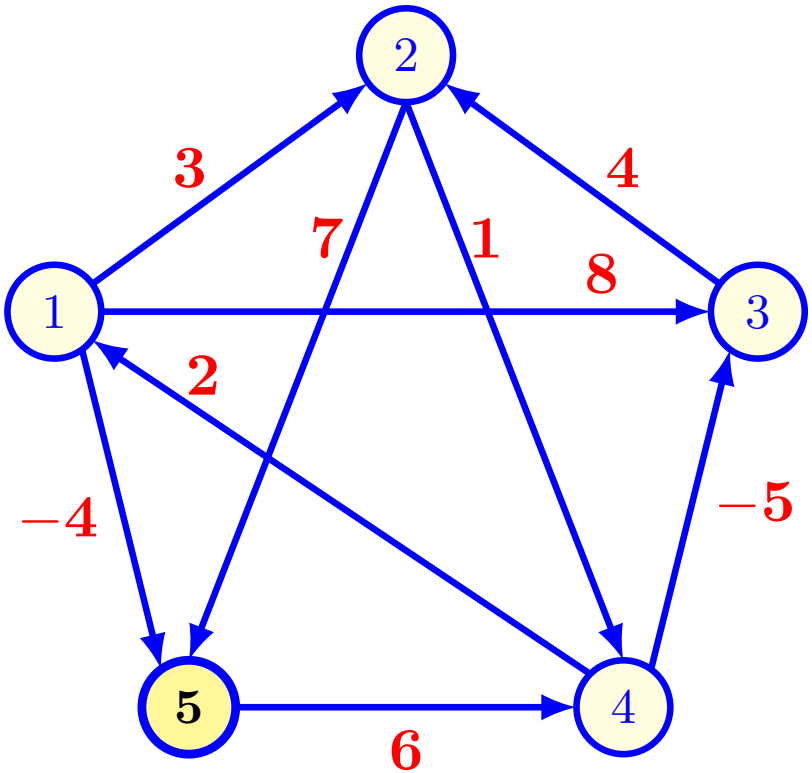
Illustrating Floyd's Algorithm with Predecessors: $k = 4$



$$D^{(4)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix} \end{array}$$

$$\Pi^{(4)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{bmatrix} \end{array}$$

Illustrating Floyd's Algorithm with Predecessors: $k = 5$



$$D^{(5)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix} \end{array}$$

$$\Pi^{(5)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{bmatrix} \end{array}$$

Getting Shortest Paths from Π Matrix

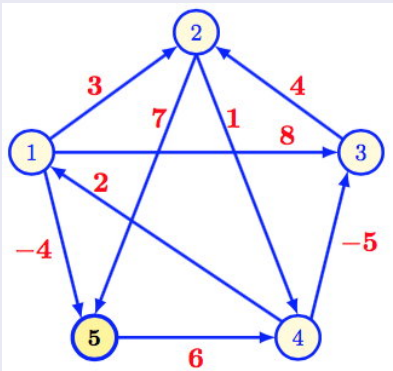
The recursive algorithm using the predecessor matrix $\Pi = \Pi^{(n)}$ to print **the shortest path** between vertices i and j:

algorithm `PrintPath`(Π, i, j)

if $i = j$ **then print** i
else
    if $\pi_{i,j} =$ NIL **then print** "no path from i to j"
    else
        `PrintPath`(Π, i, $\pi_{i,j}$)
        print j
    end if
end if

Illustrating PrintPath Algorithm

$$\Pi^{(5)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{bmatrix} \end{array}$$



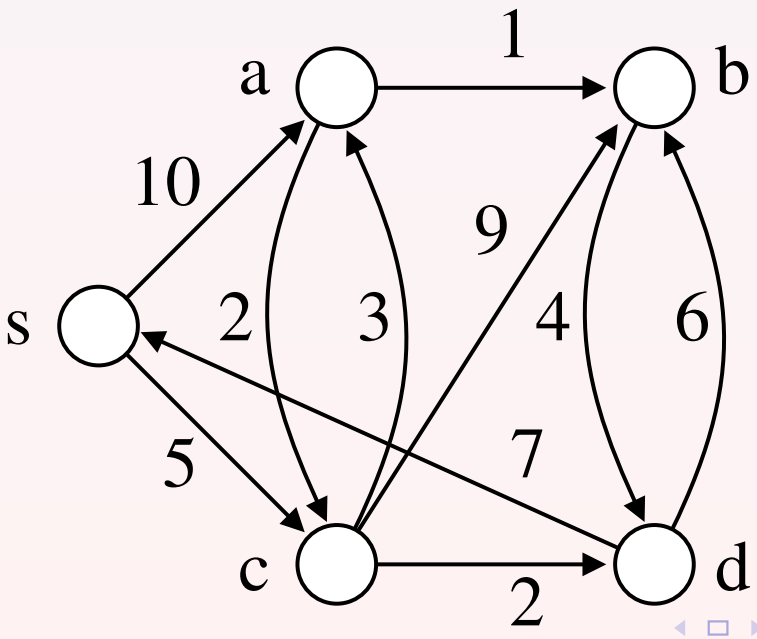
PrintPath($\Pi^{(5)}$, 5, 3)
→ PrintPath($\Pi^{(5)}$, 5, $\pi_{5,3} = 4$)
→ PrintPath($\Pi^{(5)}$, 5, $\pi_{5,4} = 5$)
print 5
print 4
print 3

PrintPath($\Pi^{(5)}$, 1, 2)
→ PrintPath($\Pi^{(5)}$, 1, $\pi_{1,2} = 3$)
→ PrintPath($\Pi^{(5)}$, 1, $\pi_{1,3} = 4$)
→ PrintPath($\Pi^{(5)}$, 1, $\pi_{1,4} = 5$)
→ PrintPath($\Pi^{(5)}$, 1, $\pi_{1,5} = 1$)
print 1
print 5
print 4
print 3
print 2

Single source shortest paths algorithms

Shortest paths

- Let $G = (V, E)$ be a directed graph, $w : E \to \mathbb{R}$ be a weight function.
- The weight of a path $P = (v_1, v_2, \ldots, v_k)$ is the sum of the weights of its edges: $w(P) = \sum_{i=2}^{k} w(v_{i-1}, v_i)$.
- A shortest path from u to v is a path from u to v with minimum weight. The shortest path is not necessarily unique.
- $\delta(u, v)$ is the weight of a shortest path from u to v. If v is not reachable from u then $\delta(u, v) = \infty$.

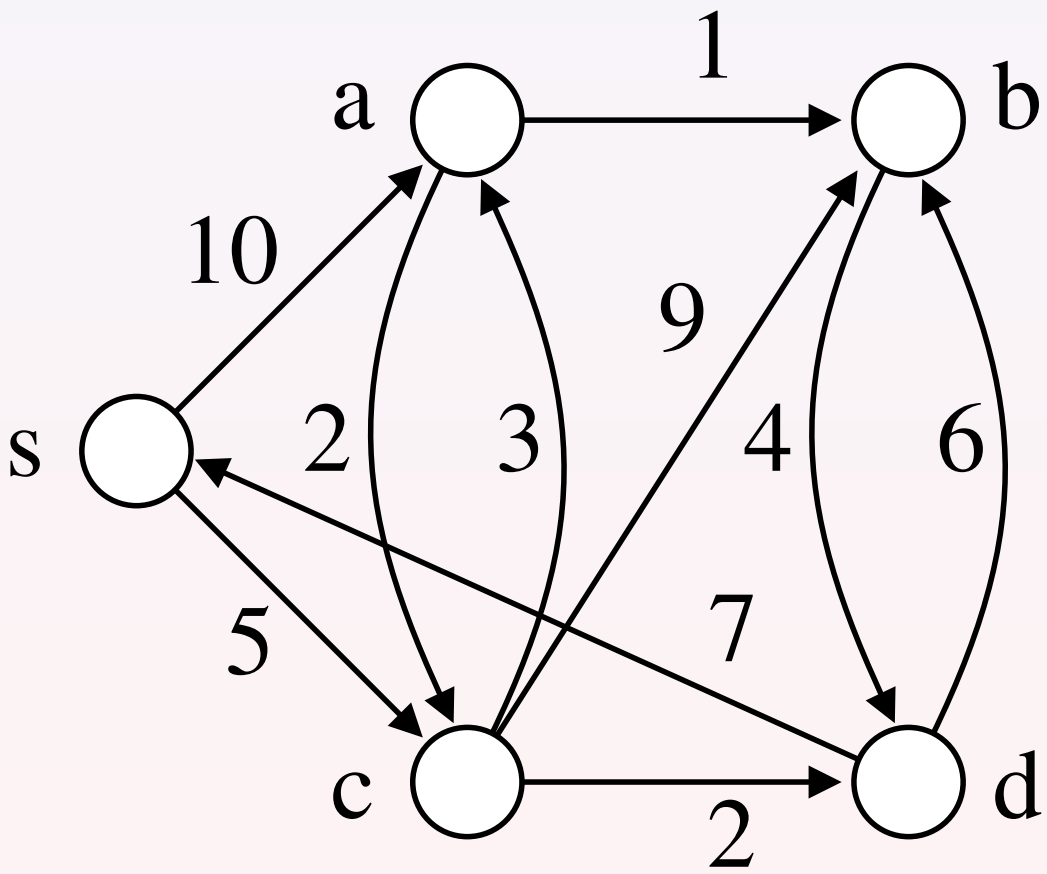


$\delta(s, b) = 9$

$\delta(b, s) = 11$

Single-source shortest paths

Given a weighted graph G and a vertex s, the goal is to compute $\delta(s, v)$ for every vertex v.



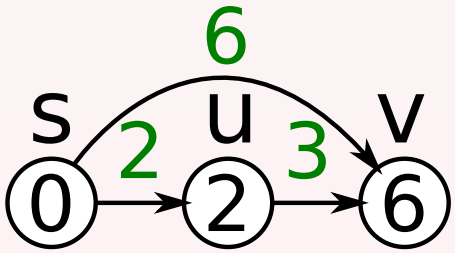
$\delta(s, s) = 0, \delta(s, a) = 8, \delta(s, b) = 9, \delta(s, c) = 5, \delta(s, d) = 7$

General structure for shortest paths algorithms

- $\delta(s, v) = \min_{u:(u,v)\in E} \delta(s, u) + w(u, v)$.
- Store values $v.d$ and $v.\pi$ in each vertex.
- $v.d$ is the length of the shortest path discovered so far from s to v ($v.d \geq \delta(s, v)$).
- $v.\pi$ is the vertex preceeding v on the shortest path discovered so far from s to v.
- When processing an edge (u, v), we update $v.d$ due to the newly discovered path $s \rightsquigarrow u \rightarrow v$.

RELAX(u, v, w)

(1) **if** $u.d + w(u, v) < v.d$
(2) $\quad v.d \leftarrow u.d + w(u, v)$
(3) $\quad v.\pi \leftarrow u$

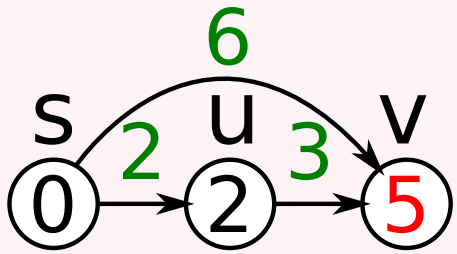


General structure for shortest paths algorithms

- $\delta(s, v) = \min_{u:(u,v)\in E} \delta(s, u) + w(u, v)$.
- Store values $v.d$ and $v.\pi$ in each vertex.
- $v.d$ is the length of the shortest path discovered so far from s to v ($v.d \geq \delta(s, v)$).
- $v.\pi$ is the vertex preceeding v on the shortest path discovered so far from s to v.
- When processing an edge (u, v), we update $v.d$ due to the newly discovered path $s \rightsquigarrow u \rightarrow v$.

RELAX(u, v, w)

(1) **if** $u.d + w(u, v) < v.d$
(2) $\quad v.d \leftarrow u.d + w(u, v)$
(3) $\quad v.\pi \leftarrow u$



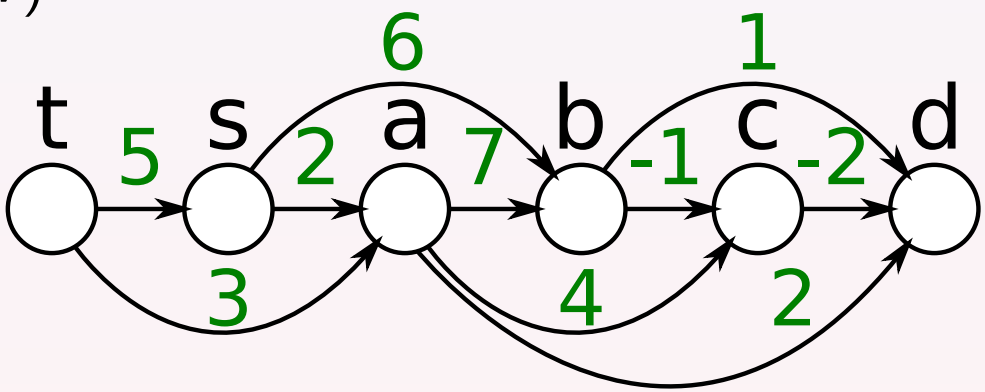
Directed acyclic graphs

DAG-SHORTEST-PATHS(G, w, s) // weights can be negative

(1) INITIALIZE(G, s)
(2) Topologically sort the vertices of G
(3) **foreach** $u \in G.V$ in topologically sorted order
(4) **foreach** $v \in G.\mathrm{Adj}[u]$
(5) RELAX(u, v, w)

INITIALIZE(G, s)

(1) **foreach** $v \in G.v$
(2) $v.d \leftarrow \infty$
(3) $v.\pi \leftarrow \mathrm{NULL}$
(4) $s.d \leftarrow 0$



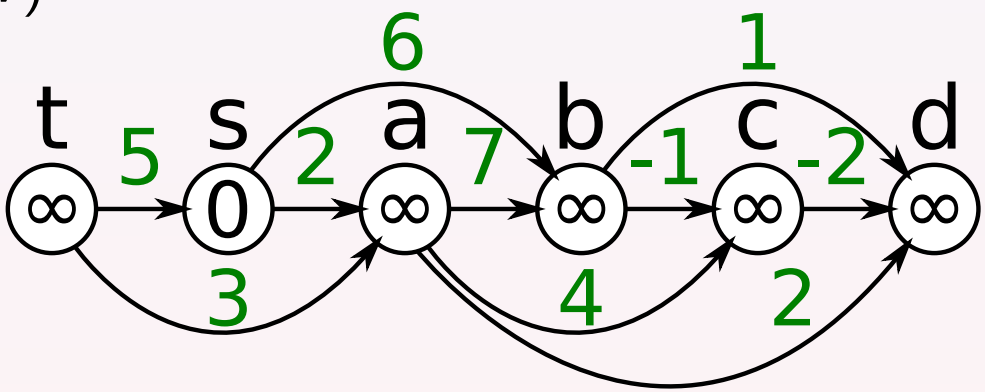
Directed acyclic graphs

DAG-SHORTEST-PATHS(G, w, s) // weights can be negative

(1) INITIALIZE(G, s)
(2) Topologically sort the vertices of G
(3) **foreach** $u \in G.V$ in topologically sorted order
(4) **foreach** $v \in G.\mathrm{Adj}[u]$
(5) RELAX(u, v, w)

INITIALIZE(G, s)

(1) **foreach** $v \in G.v$
(2) $v.d \leftarrow \infty$
(3) $v.\pi \leftarrow \mathrm{NULL}$
(4) $s.d \leftarrow 0$



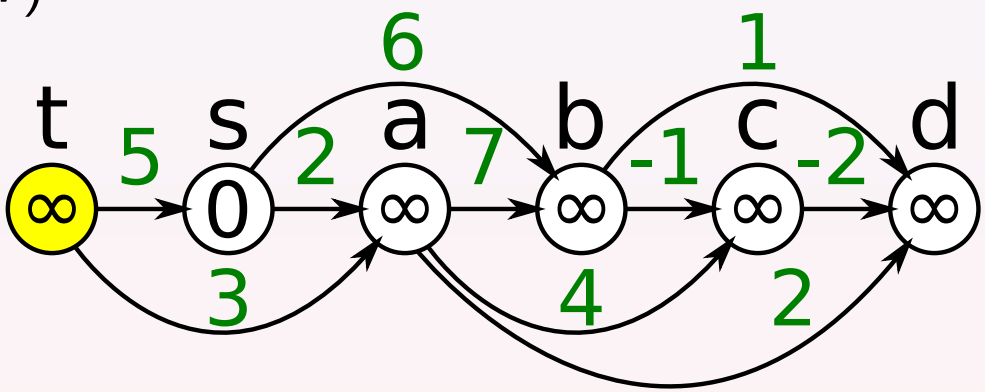
Directed acyclic graphs

DAG-SHORTEST-PATHS(G, w, s) // weights can be negative

(1) INITIALIZE(G, s)
(2) Topologically sort the vertices of G
(3) **foreach** $u \in G.V$ in topologically sorted order
(4) **foreach** $v \in G.\mathrm{Adj}[u]$
(5) RELAX(u, v, w)

INITIALIZE(G, s)

(1) **foreach** $v \in G.v$
(2) $v.d \leftarrow \infty$
(3) $v.\pi \leftarrow$ NULL
(4) $s.d \leftarrow 0$



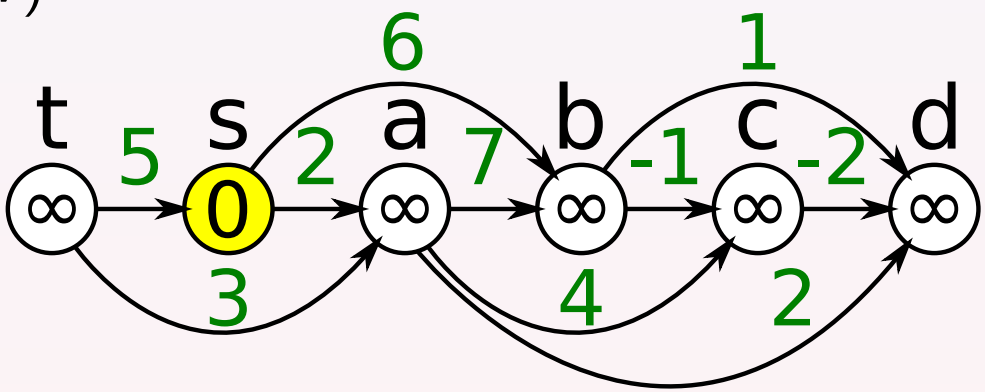
Directed acyclic graphs

DAG-SHORTEST-PATHS(G, w, s) // weights can be negative

(1) INITIALIZE(G, s)
(2) Topologically sort the vertices of G
(3) **foreach** $u \in G.V$ in topologically sorted order
(4) **foreach** $v \in G.\mathrm{Adj}[u]$
(5) RELAX(u, v, w)

INITIALIZE(G, s)

(1) **foreach** $v \in G.v$
(2) $v.d \leftarrow \infty$
(3) $v.\pi \leftarrow \mathrm{NULL}$
(4) $s.d \leftarrow 0$



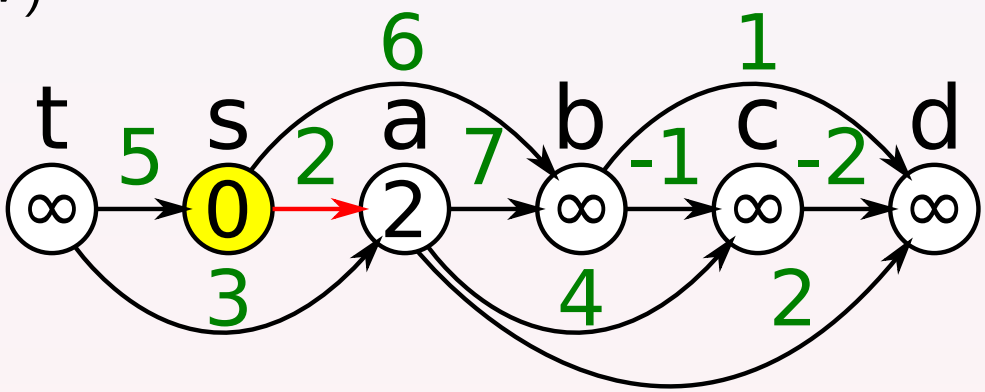
Directed acyclic graphs

DAG-SHORTEST-PATHS(G, w, s) // weights can be negative

(1) INITIALIZE(G, s)
(2) Topologically sort the vertices of G
(3) **foreach** $u \in G.V$ in topologically sorted order
(4) **foreach** $v \in G.\mathrm{Adj}[u]$
(5) RELAX(u, v, w)

INITIALIZE(G, s)

(1) **foreach** $v \in G.v$
(2) $v.d \leftarrow \infty$
(3) $v.\pi \leftarrow \mathrm{NULL}$
(4) $s.d \leftarrow 0$



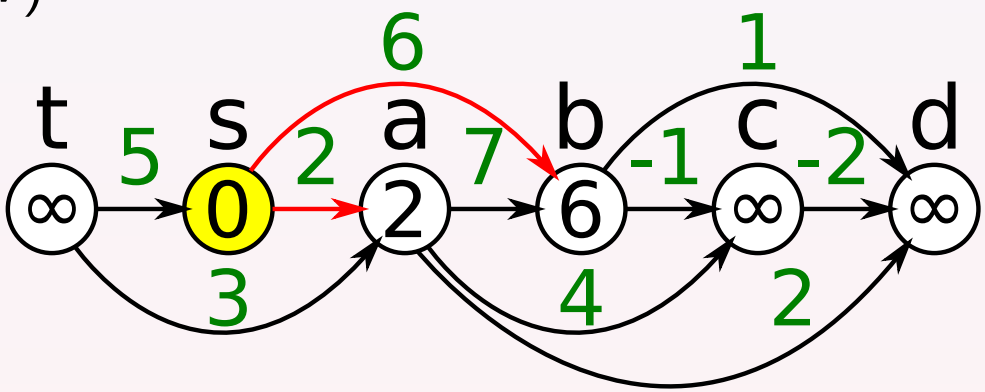
Directed acyclic graphs

DAG-SHORTEST-PATHS(G, w, s) // weights can be negative

(1) INITIALIZE(G, s)
(2) Topologically sort the vertices of G
(3) **foreach** $u \in G.V$ in topologically sorted order
(4) **foreach** $v \in G.\mathrm{Adj}[u]$
(5) RELAX(u, v, w)

INITIALIZE(G, s)

(1) **foreach** $v \in G.v$
(2) $v.d \leftarrow \infty$
(3) $v.\pi \leftarrow \mathrm{NULL}$
(4) $s.d \leftarrow 0$



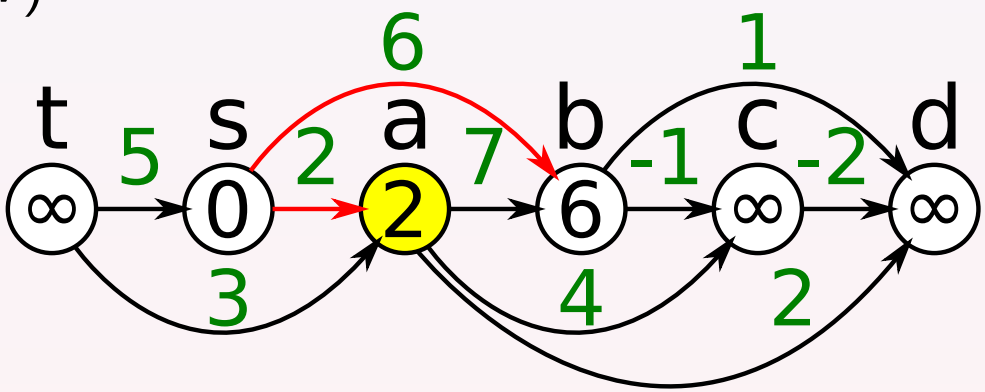
Directed acyclic graphs

DAG-SHORTEST-PATHS(G, w, s) // weights can be negative

(1) INITIALIZE(G, s)
(2) Topologically sort the vertices of G
(3) **foreach** $u \in G.V$ in topologically sorted order
(4) **foreach** $v \in G.\mathrm{Adj}[u]$
(5) RELAX(u, v, w)

INITIALIZE(G, s)

(1) **foreach** $v \in G.v$
(2) $v.d \leftarrow \infty$
(3) $v.\pi \leftarrow \mathrm{NULL}$
(4) $s.d \leftarrow 0$



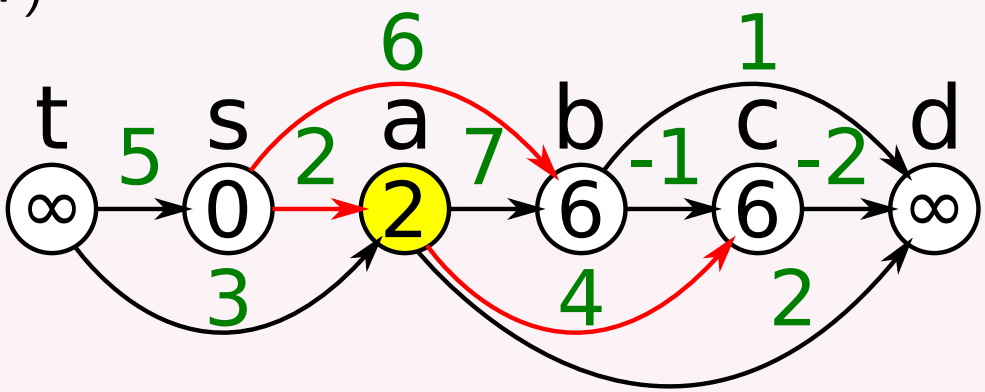
Directed acyclic graphs

DAG-SHORTEST-PATHS(G, w, s) // weights can be negative

(1) INITIALIZE(G, s)
(2) Topologically sort the vertices of G
(3) **foreach** $u \in G.V$ in topologically sorted order
(4) **foreach** $v \in G.\mathrm{Adj}[u]$
(5) RELAX(u, v, w)

INITIALIZE(G, s)

(1) **foreach** $v \in G.v$
(2) $v.d \leftarrow \infty$
(3) $v.\pi \leftarrow \mathrm{NULL}$
(4) $s.d \leftarrow 0$



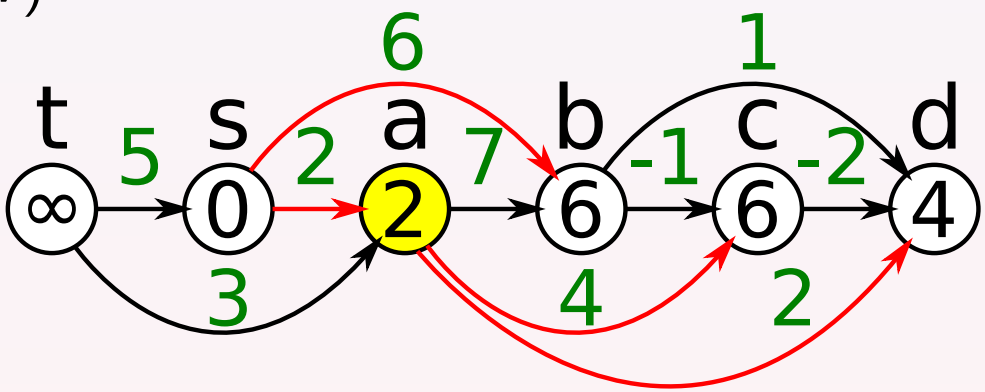
Directed acyclic graphs

DAG-SHORTEST-PATHS(G, w, s) // weights can be negative

(1) INITIALIZE(G, s)
(2) Topologically sort the vertices of G
(3) **foreach** $u \in G.V$ in topologically sorted order
(4) **foreach** $v \in G.\mathrm{Adj}[u]$
(5) RELAX(u, v, w)

INITIALIZE(G, s)

(1) **foreach** $v \in G.v$
(2) $v.d \leftarrow \infty$
(3) $v.\pi \leftarrow$ NULL
(4) $s.d \leftarrow 0$



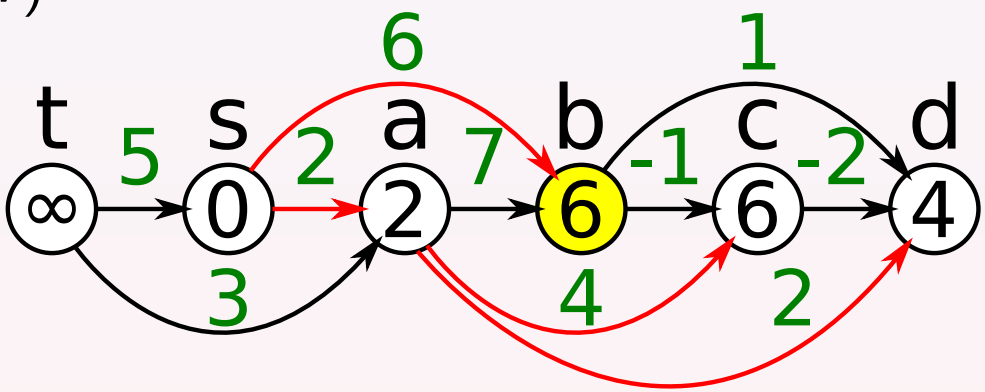
Directed acyclic graphs

DAG-SHORTEST-PATHS(G, w, s) // weights can be negative

(1) INITIALIZE(G, s)
(2) Topologically sort the vertices of G
(3) **foreach** $u \in G.V$ in topologically sorted order
(4) **foreach** $v \in G.\mathrm{Adj}[u]$
(5) RELAX(u, v, w)

INITIALIZE(G, s)

(1) **foreach** $v \in G.v$
(2) $v.d \leftarrow \infty$
(3) $v.\pi \leftarrow$ NULL
(4) $s.d \leftarrow 0$



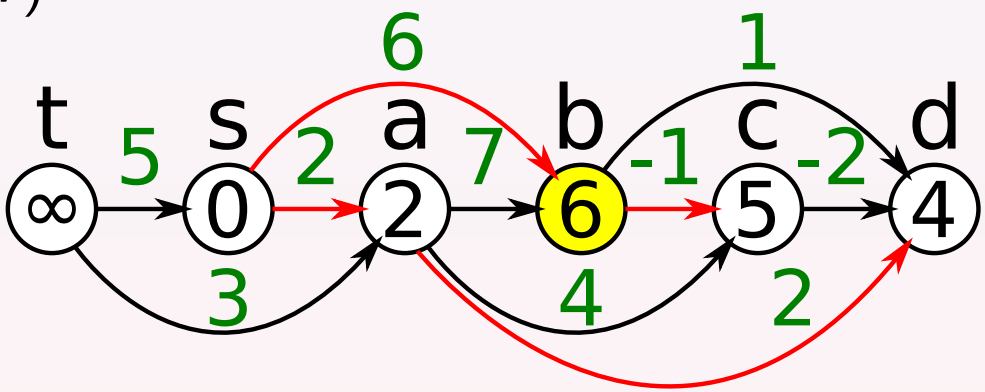
Directed acyclic graphs

DAG-SHORTEST-PATHS(G, w, s) // weights can be negative

(1) INITIALIZE(G, s)
(2) Topologically sort the vertices of G
(3) **foreach** $u \in G.V$ in topologically sorted order
(4) **foreach** $v \in G.\mathrm{Adj}[u]$
(5) RELAX(u, v, w)

INITIALIZE(G, s)

(1) **foreach** $v \in G.v$
(2) $v.d \leftarrow \infty$
(3) $v.\pi \leftarrow$ NULL
(4) $s.d \leftarrow 0$



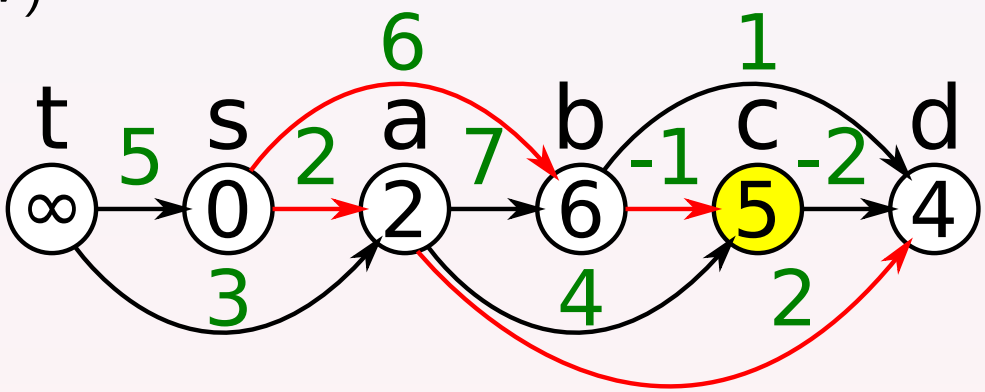
Directed acyclic graphs

DAG-SHORTEST-PATHS(G, w, s) // weights can be negative

(1) INITIALIZE(G, s)
(2) Topologically sort the vertices of G
(3) **foreach** $u \in G.V$ in topologically sorted order
(4) **foreach** $v \in G.\mathrm{Adj}[u]$
(5) RELAX(u, v, w)

INITIALIZE(G, s)

(1) **foreach** $v \in G.v$
(2) $v.d \leftarrow \infty$
(3) $v.\pi \leftarrow$ NULL
(4) $s.d \leftarrow 0$



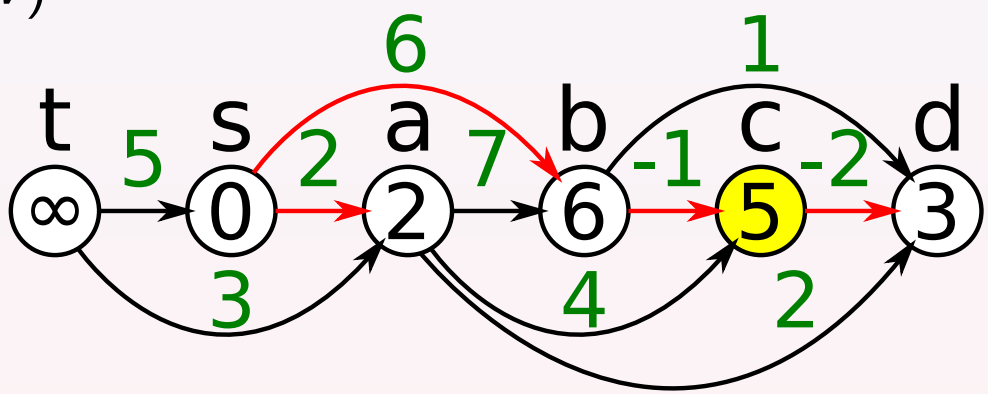
Directed acyclic graphs

DAG-SHORTEST-PATHS(G, w, s) // weights can be negative

(1) INITIALIZE(G, s)
(2) Topologically sort the vertices of G
(3) **foreach** $u \in G.V$ in topologically sorted order
(4) **foreach** $v \in G.\mathrm{Adj}[u]$
(5) RELAX(u, v, w)

INITIALIZE(G, s)

(1) **foreach** $v \in G.v$
(2) $v.d \leftarrow \infty$
(3) $v.\pi \leftarrow \mathrm{NULL}$
(4) $s.d \leftarrow 0$



Claim: When a vertex u is processed in line 3, $u.d = \delta(s, u)$.

Proof: Due to the topological order, we already called RELAX(x, u, w) for every edge (x, u) entering u.

By induction, when RELAX(x, u, w) is called, $x.d = \delta(s, x)$.

Thus, $u.d = \min_{x:(x,u)\in E} \delta(s, x) + w(x, u) = \delta(s, u)$.

Directed acyclic graphs

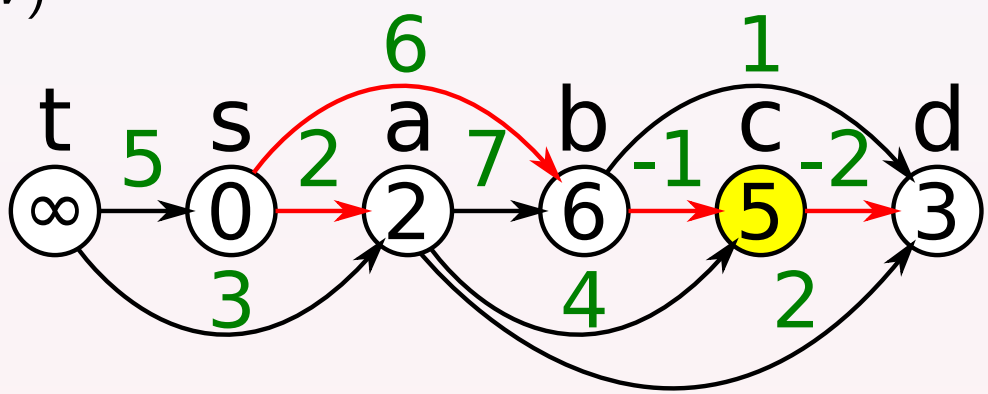
DAG-SHORTEST-PATHS(G, w, s) // weights can be negative

(1) INITIALIZE(G, s)
(2) Topologically sort the vertices of G
(3) **foreach** $u \in G.V$ in topologically sorted order
(4) **foreach** $v \in G.\mathrm{Adj}[u]$
(5) RELAX(u, v, w)

INITIALIZE(G, s)

(1) **foreach** $v \in G.v$
(2) $v.d \leftarrow \infty$
(3) $v.\pi \leftarrow \mathrm{NULL}$
(4) $s.d \leftarrow 0$

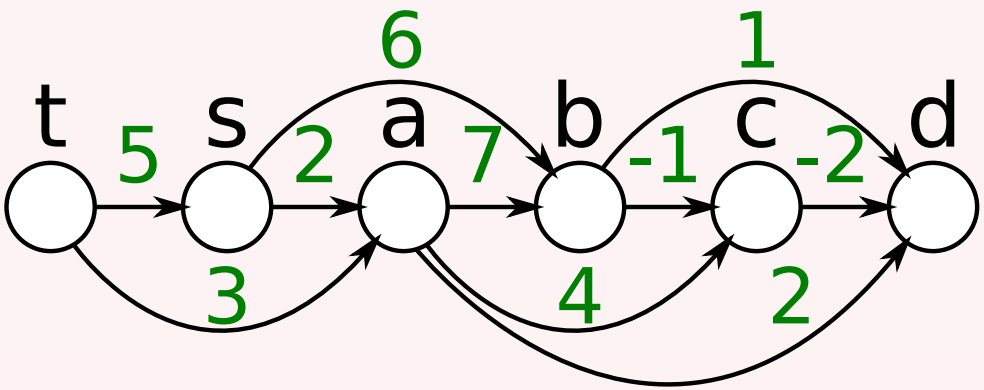
Line 1: $\Theta(V)$
Line 2: $\Theta(V + E)$
Lines 3–5: $\Theta(V + E)$
Total time: $\Theta(V + E)$



Backward dynamic programming

DAG-SHORTEST-PATHS(G, w, s)

(1) INITIALIZE(G, s)
(2) Initialize array Adj^R
(3) **foreach** $u \in G.V$
(4) **foreach** $v \in G.\mathrm{Adj}[u]$
(5) Add u to $\mathrm{Adj}^R[v]$
(6) Topologically sort the vertices of G
(7) **foreach** $v \in G.V$ in topologically sorted order
(8) **foreach** $u \in \mathrm{Adj}^R[v]$
(9) RELAX(u, v, w)



Problem variants

Variant 1: Given G, s compute the maximum weight of a path from s to every vertex v.
Variant 2: Given a directed graph G with weights on the vertices and a vertex s, compute the maximum weight of a path from s to every vertex v.

Variant 1 can be solved by changing $\textsc{Relax}(u, v, w)$ to

(1) **if** $v.d < u.d + w(u, v)$
(2) $\quad v.d \leftarrow u.d + w(u, v)$
(3) $\quad v.\pi \leftarrow u$

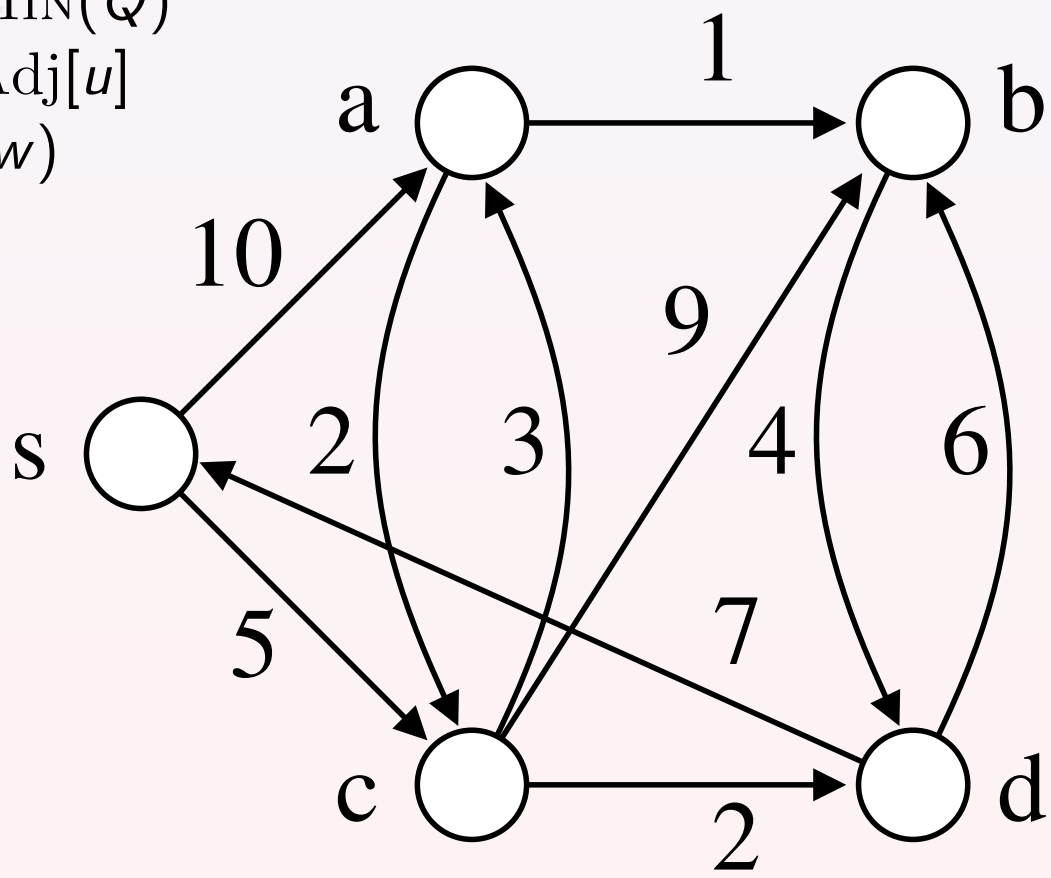
Variant 2 can be solved by changing $\textsc{Relax}(u, v, w)$ to

(1) **if** $v.d < u.d + w(v)$
(2) $\quad v.d \leftarrow u.d + w(v)$
(3) $\quad v.\pi \leftarrow u$

General graphs

DIJSKTRA(G, w, s) // weights are non-negative

(1) INITIALIZE(G, s)
(2) Build a priority queue Q on the vertices (keys are d values)
(3) **while** Q is not empty
(4) $u \leftarrow$ EXTRACTMIN(Q)
(5) **foreach** $v \in G.\mathrm{Adj}[u]$
(6) RELAX(u, v, w)

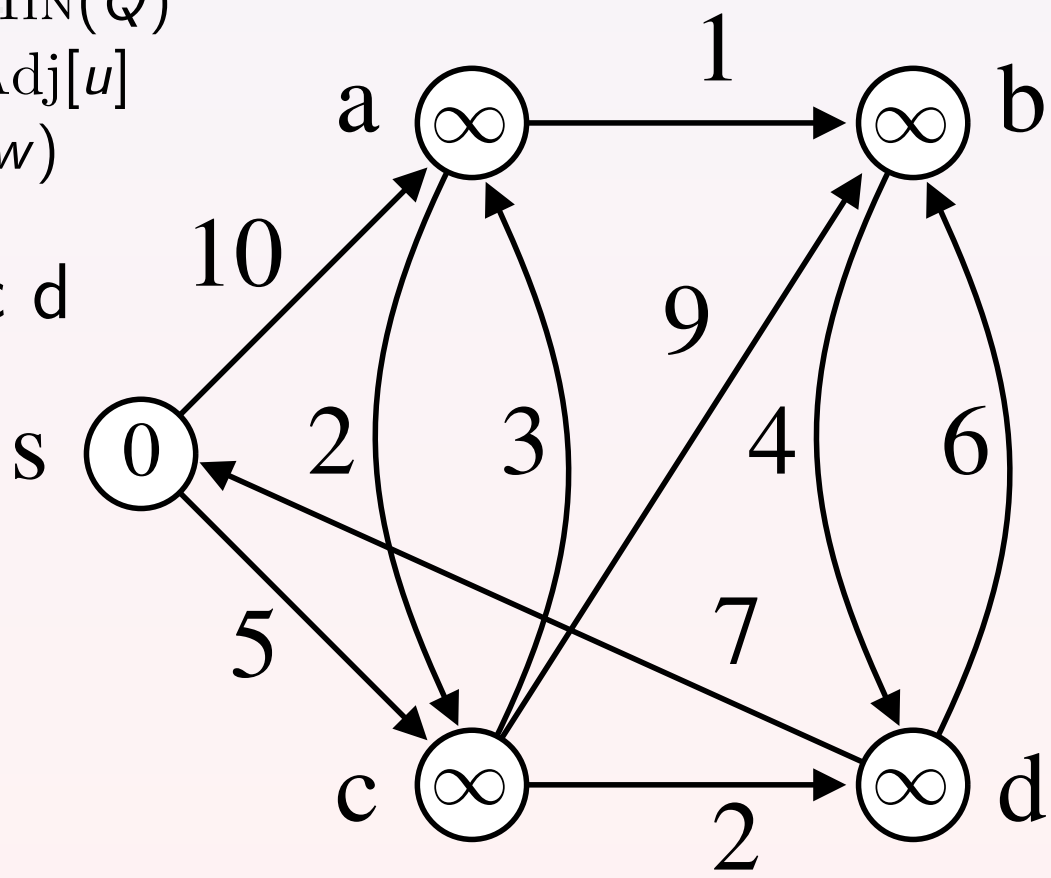


General graphs

DIJSKTRA(G, w, s) // weights are non-negative

(1) INITIALIZE(G, s)
(2) Build a priority queue Q on the vertices (keys are d values)
(3) **while** Q is not empty
(4) $\quad u \leftarrow$ EXTRACTMIN(Q)
(5) $\quad$ **foreach** $v \in G.\mathrm{Adj}[u]$
(6) $\quad\quad$ RELAX(u, v, w)

Q : s a b c d

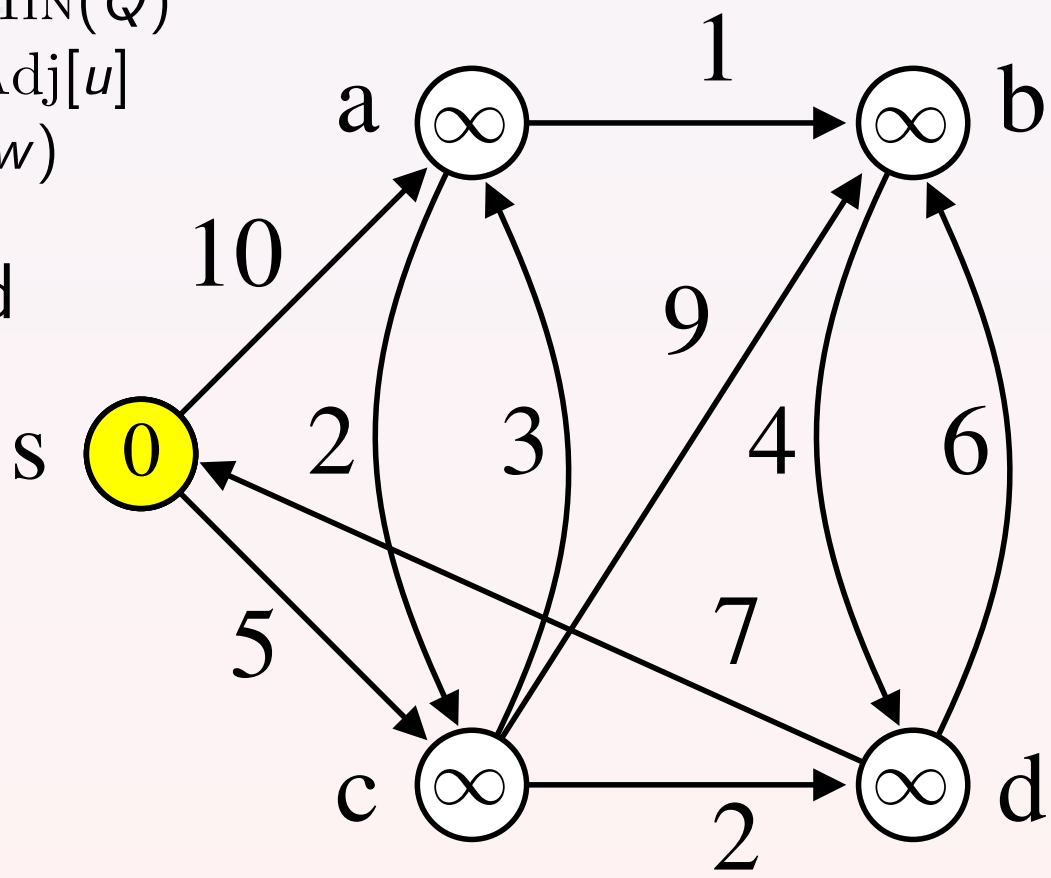


General graphs

DIJSKTRA(G, w, s) // weights are non-negative

(1) INITIALIZE(G, s)
(2) Build a priority queue Q on the vertices (keys are d values)
(3) **while** Q is not empty
(4) $\quad u \leftarrow$ EXTRACTMIN(Q)
(5) $\quad$ **foreach** $v \in G.\mathrm{Adj}[u]$
(6) $\quad\quad$ RELAX(u, v, w)

Q : a b c d

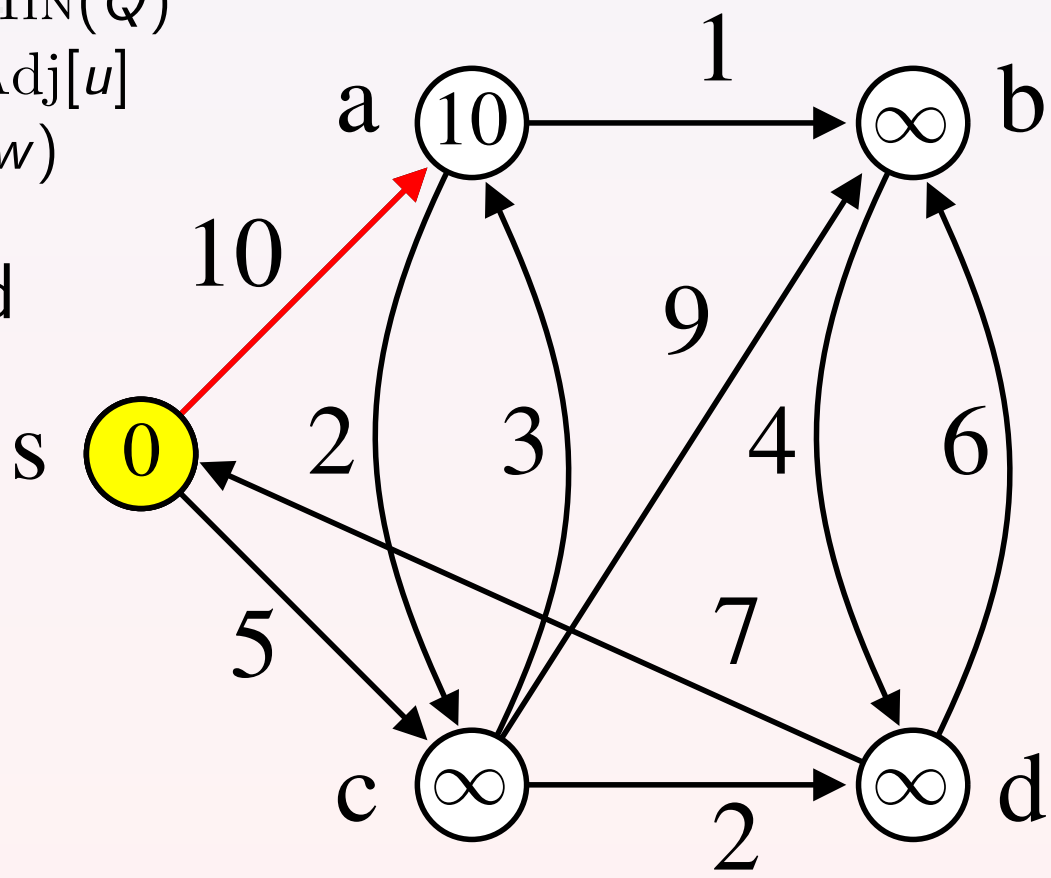


General graphs

DIJSKTRA(G, w, s) // weights are non-negative

(1) INITIALIZE(G, s)
(2) Build a priority queue Q on the vertices (keys are d values)
(3) **while** Q is not empty
(4) $u \leftarrow$ EXTRACTMIN(Q)
(5) **foreach** $v \in G.\mathrm{Adj}[u]$
(6) RELAX(u, v, w)

Q : a b c d

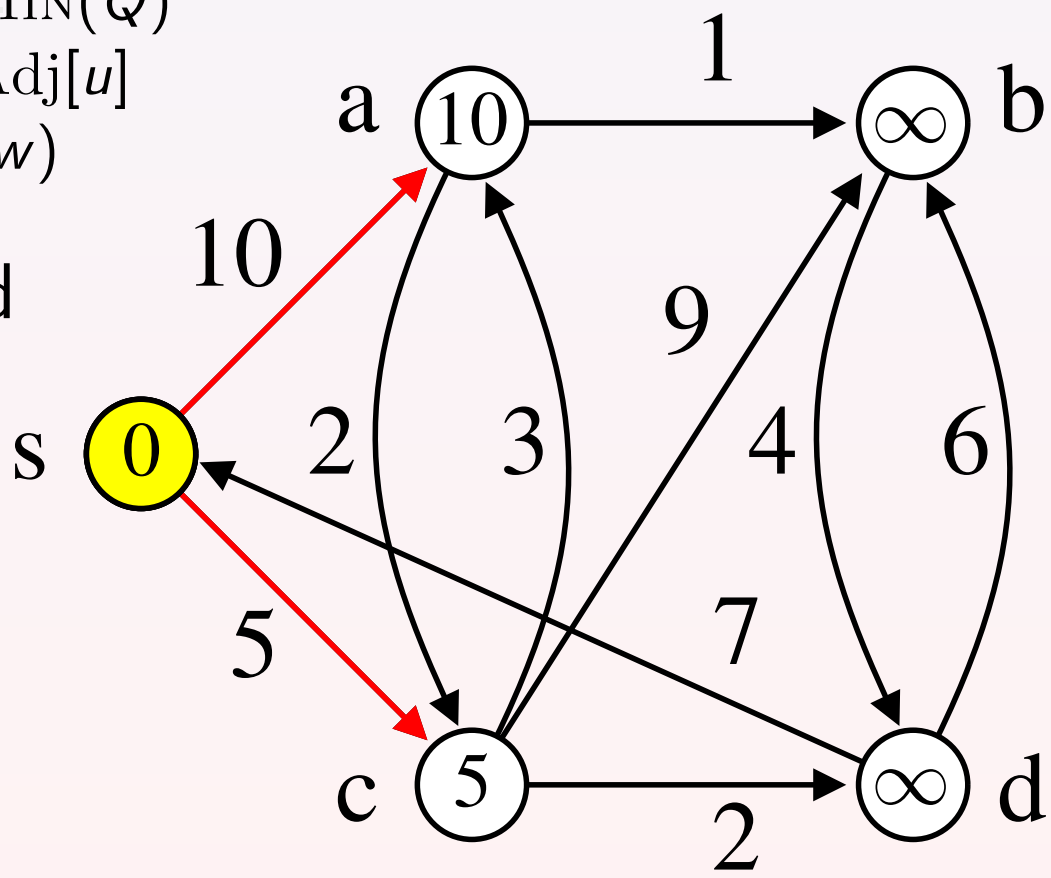


General graphs

DIJSKTRA(G, w, s) // weights are non-negative

(1) INITIALIZE(G, s)
(2) Build a priority queue Q on the vertices (keys are d values)
(3) **while** Q is not empty
(4) $u \leftarrow$ EXTRACTMIN(Q)
(5) **foreach** $v \in G.\mathrm{Adj}[u]$
(6) RELAX(u, v, w)

Q : a b c d

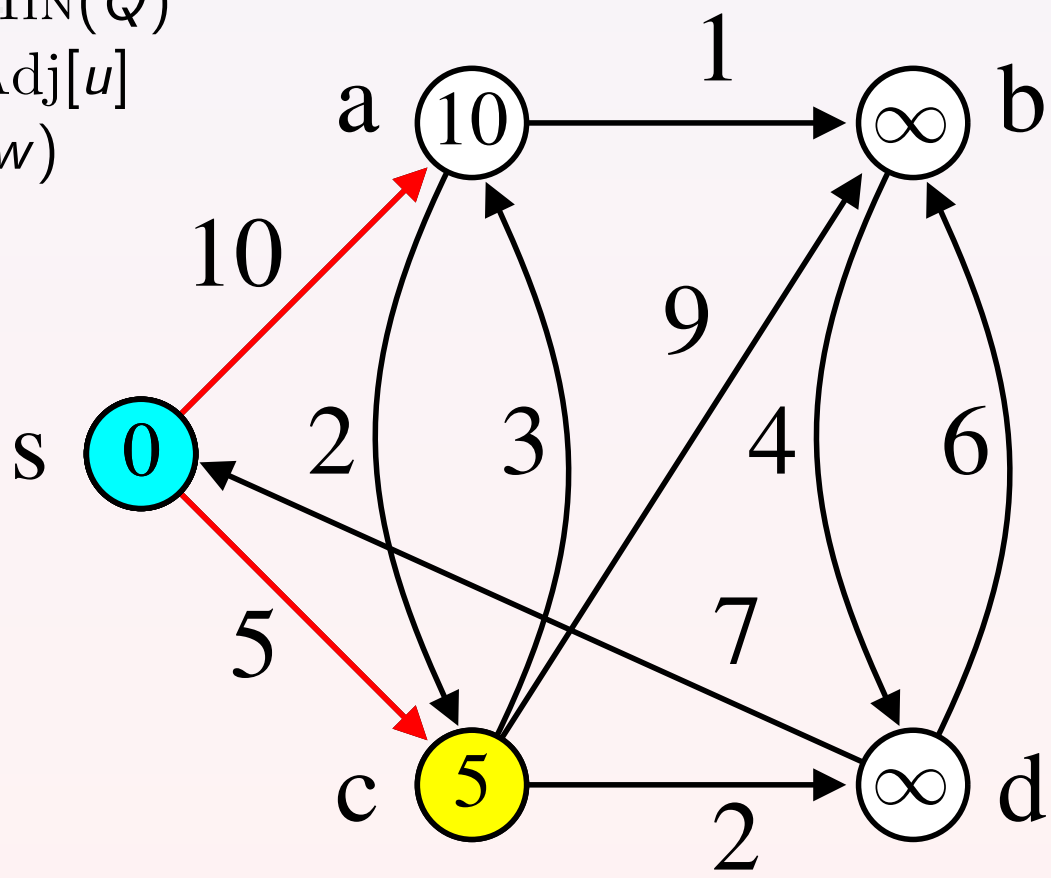


General graphs

DIJSKTRA(G, w, s) // weights are non-negative

(1) INITIALIZE(G, s)
(2) Build a priority queue Q on the vertices (keys are d values)
(3) **while** Q is not empty
(4) $u \leftarrow$ EXTRACTMIN(Q)
(5) **foreach** $v \in G.\mathrm{Adj}[u]$
(6) RELAX(u, v, w)

Q : a b d

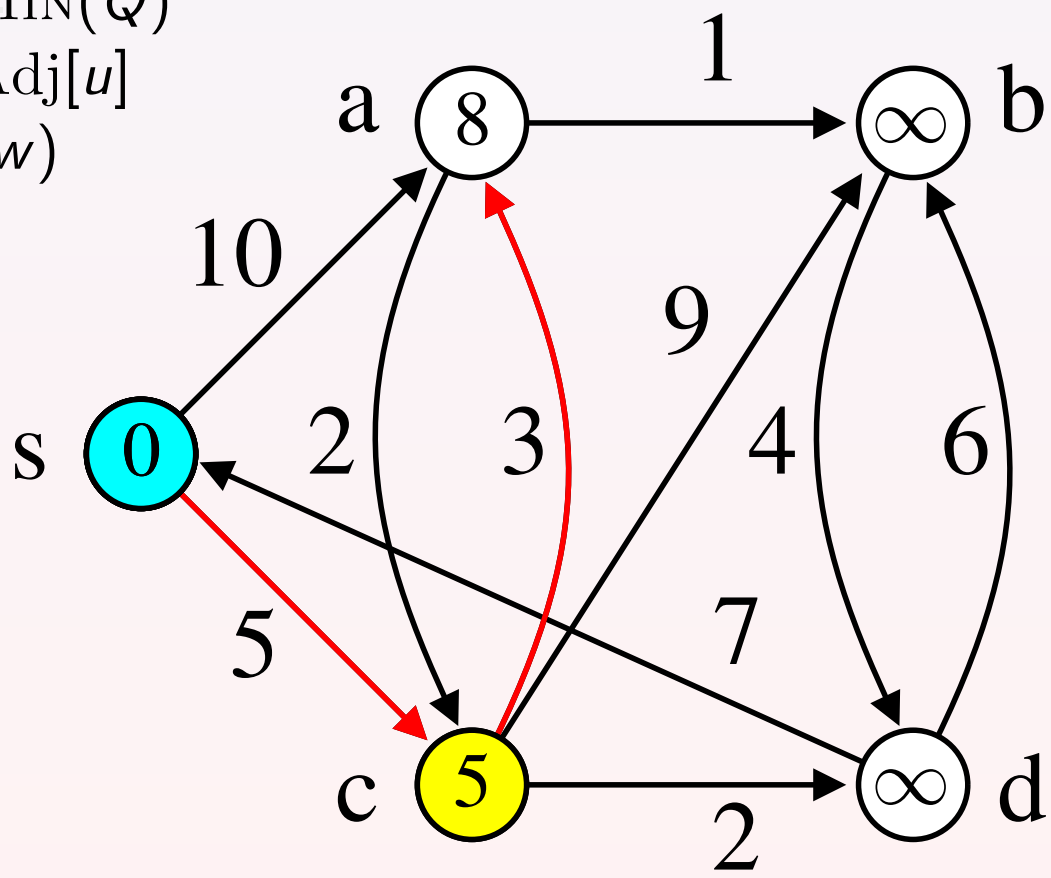


General graphs

DIJSKTRA(G, w, s) // weights are non-negative

(1) INITIALIZE(G, s)
(2) Build a priority queue Q on the vertices (keys are d values)
(3) **while** Q is not empty
(4) $\quad u \leftarrow$ EXTRACTMIN(Q)
(5) $\quad$ **foreach** $v \in G.\mathrm{Adj}[u]$
(6) $\quad\quad$ RELAX(u, v, w)

Q : a b d

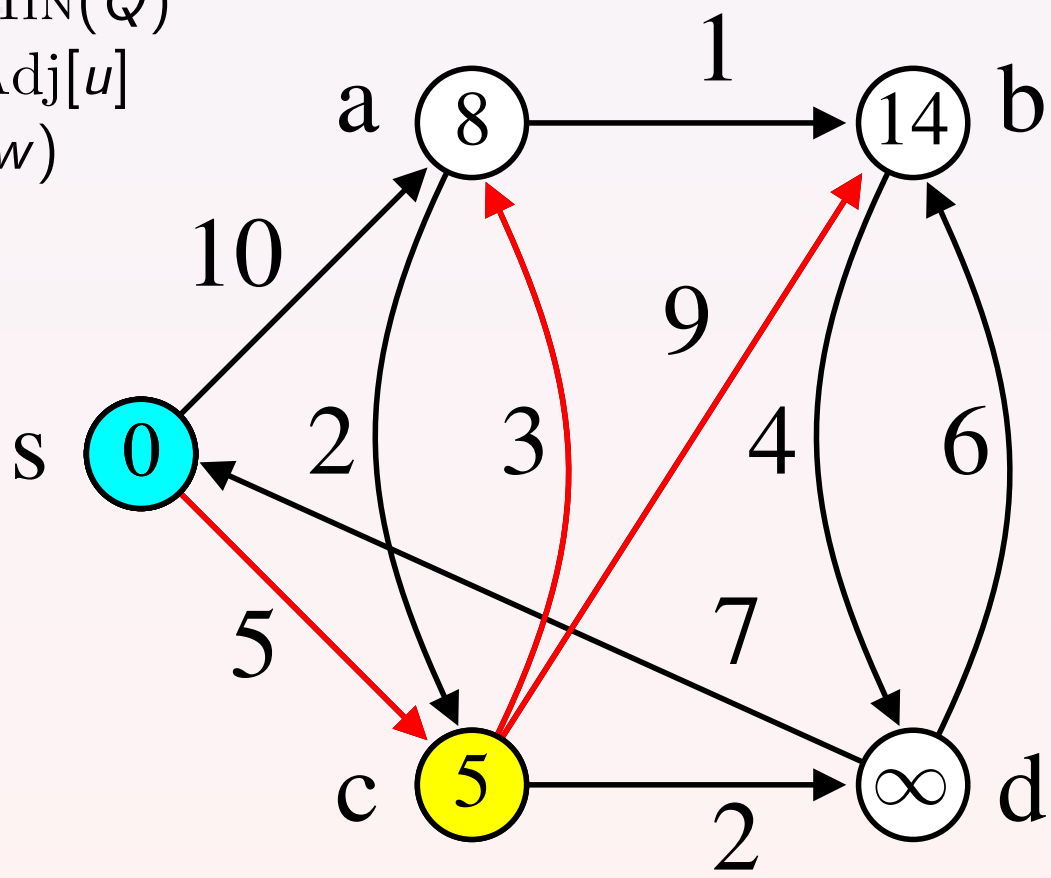


General graphs

DIJSKTRA(G, w, s) // weights are non-negative

(1) INITIALIZE(G, s)
(2) Build a priority queue Q on the vertices (keys are d values)
(3) **while** Q is not empty
(4) $\quad u \leftarrow$ EXTRACTMIN(Q)
(5) $\quad$ **foreach** $v \in G.\mathrm{Adj}[u]$
(6) $\quad\quad$ RELAX(u, v, w)

Q : a b d

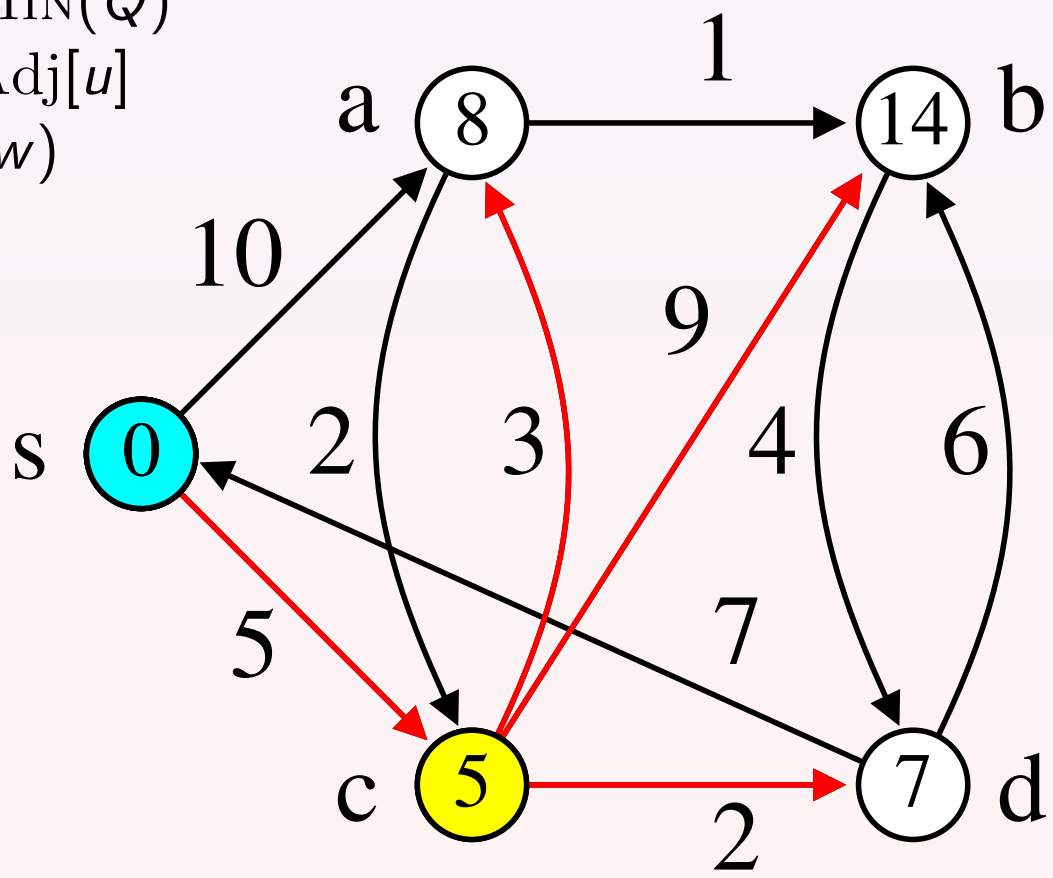


General graphs

DIJSKTRA(G, w, s) // weights are non-negative

(1) INITIALIZE(G, s)
(2) Build a priority queue Q on the vertices (keys are d values)
(3) **while** Q is not empty
(4) $\quad u \leftarrow$ EXTRACTMIN(Q)
(5) $\quad$ **foreach** $v \in G.\mathrm{Adj}[u]$
(6) $\quad\quad$ RELAX(u, v, w)

Q : a b d

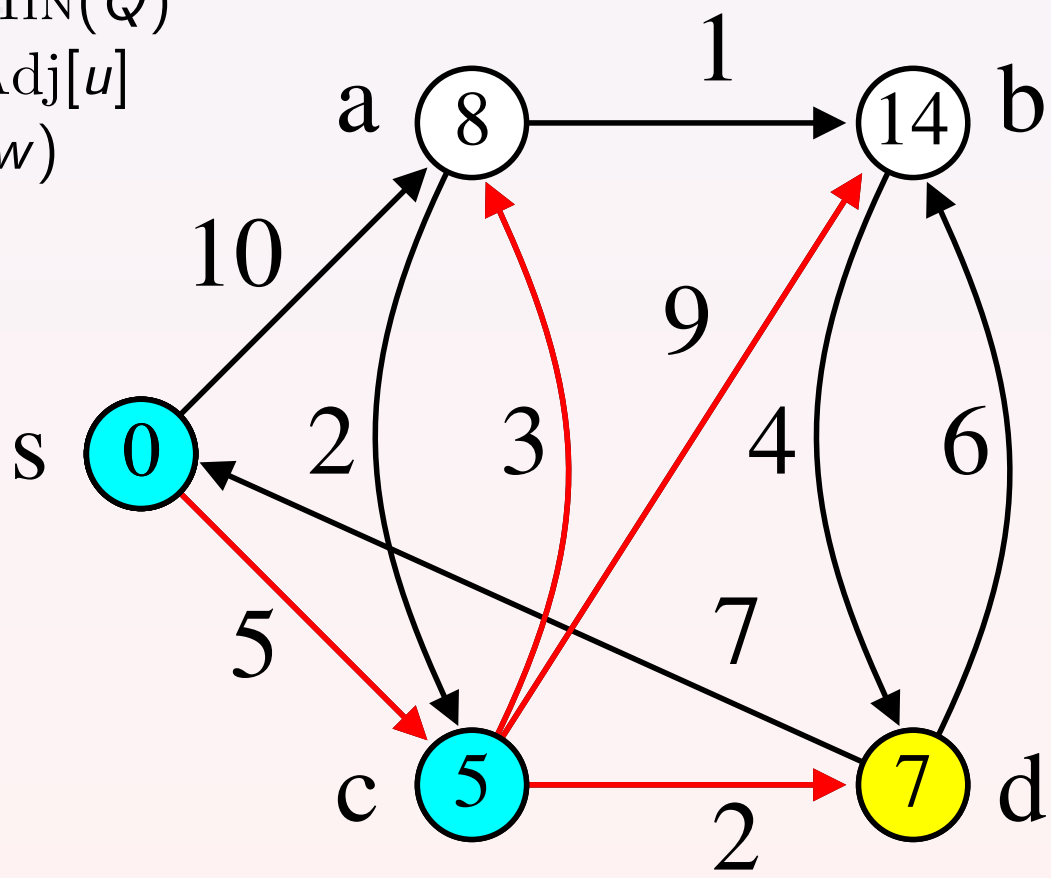


General graphs

DIJSKTRA(G, w, s) // weights are non-negative

(1) INITIALIZE(G, s)
(2) Build a priority queue Q on the vertices (keys are d values)
(3) **while** Q is not empty
(4) $\quad u \leftarrow$ EXTRACTMIN(Q)
(5) $\quad$ **foreach** $v \in G.\mathrm{Adj}[u]$
(6) $\quad\quad$ RELAX(u, v, w)

Q : a b

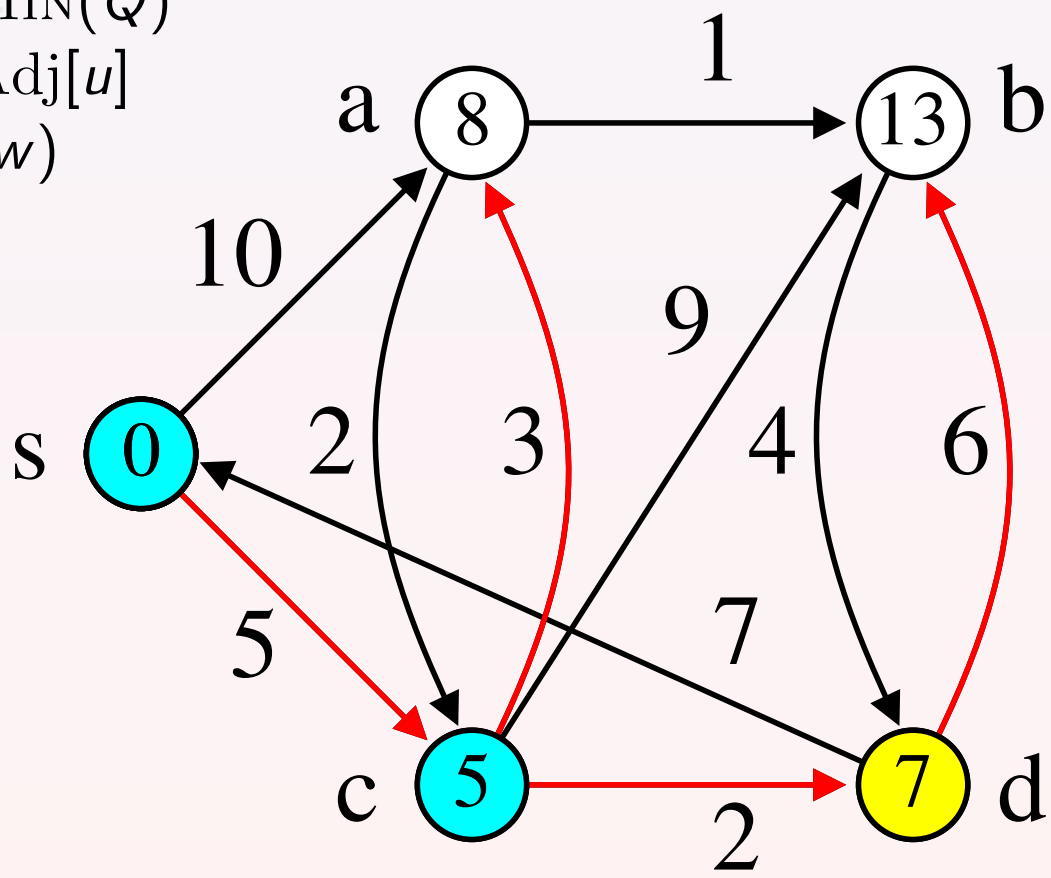


General graphs

DIJSKTRA(G, w, s) // weights are non-negative

(1) INITIALIZE(G, s)
(2) Build a priority queue Q on the vertices (keys are d values)
(3) **while** Q is not empty
(4) $u \leftarrow$ EXTRACTMIN(Q)
(5) **foreach** $v \in G.\mathrm{Adj}[u]$
(6) RELAX(u, v, w)

Q : a b

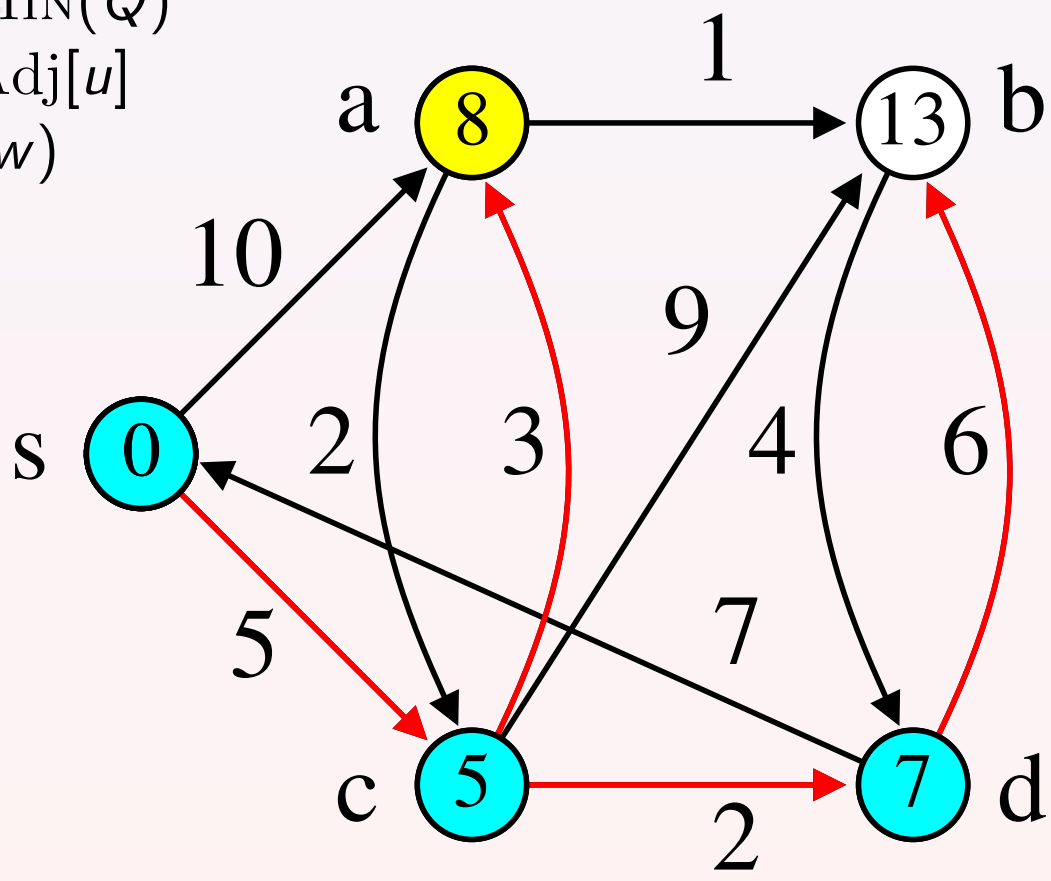


General graphs

DIJSKTRA(G, w, s) // weights are non-negative

(1) INITIALIZE(G, s)
(2) Build a priority queue Q on the vertices (keys are d values)
(3) **while** Q is not empty
(4) $\quad u \leftarrow$ EXTRACTMIN(Q)
(5) $\quad$ **foreach** $v \in G.\mathrm{Adj}[u]$
(6) $\quad\quad$ RELAX(u, v, w)

Q : b

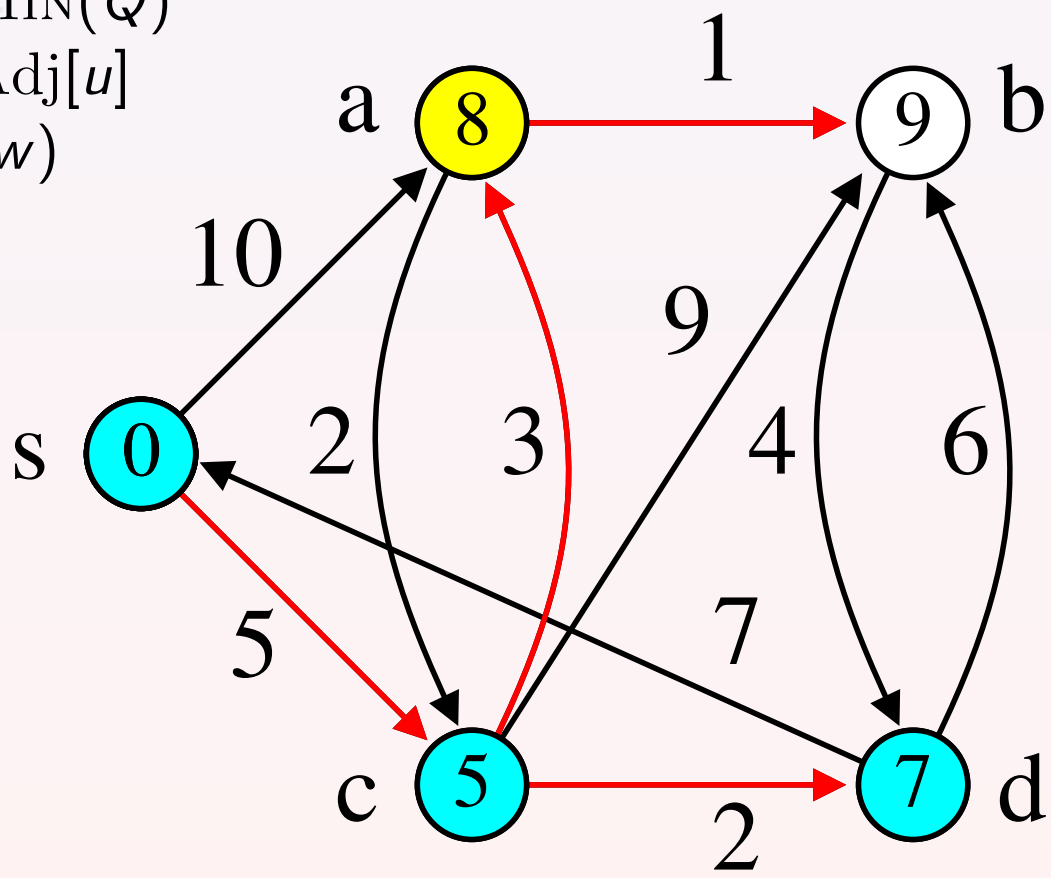


General graphs

DIJSKTRA(G, w, s) // weights are non-negative

(1) INITIALIZE(G, s)
(2) Build a priority queue Q on the vertices (keys are d values)
(3) **while** Q is not empty
(4) $u \leftarrow$ EXTRACTMIN(Q)
(5) **foreach** $v \in G.\mathrm{Adj}[u]$
(6) RELAX(u, v, w)

Q : b

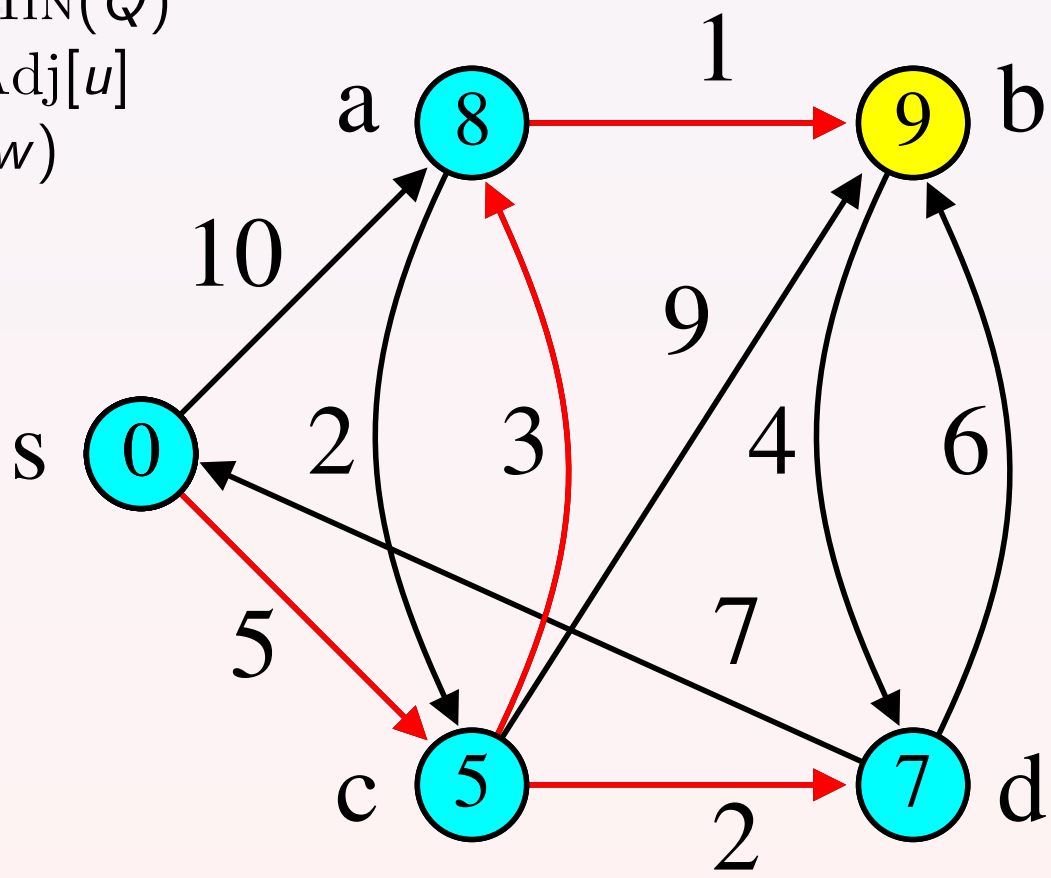


General graphs

DIJSKTRA(G, w, s) // weights are non-negative

(1) INITIALIZE(G, s)
(2) Build a priority queue Q on the vertices (keys are d values)
(3) **while** Q is not empty
(4) $\quad u \leftarrow$ EXTRACTMIN(Q)
(5) $\quad$ **foreach** $v \in G.\mathrm{Adj}[u]$
(6) $\quad\quad$ RELAX(u, v, w)

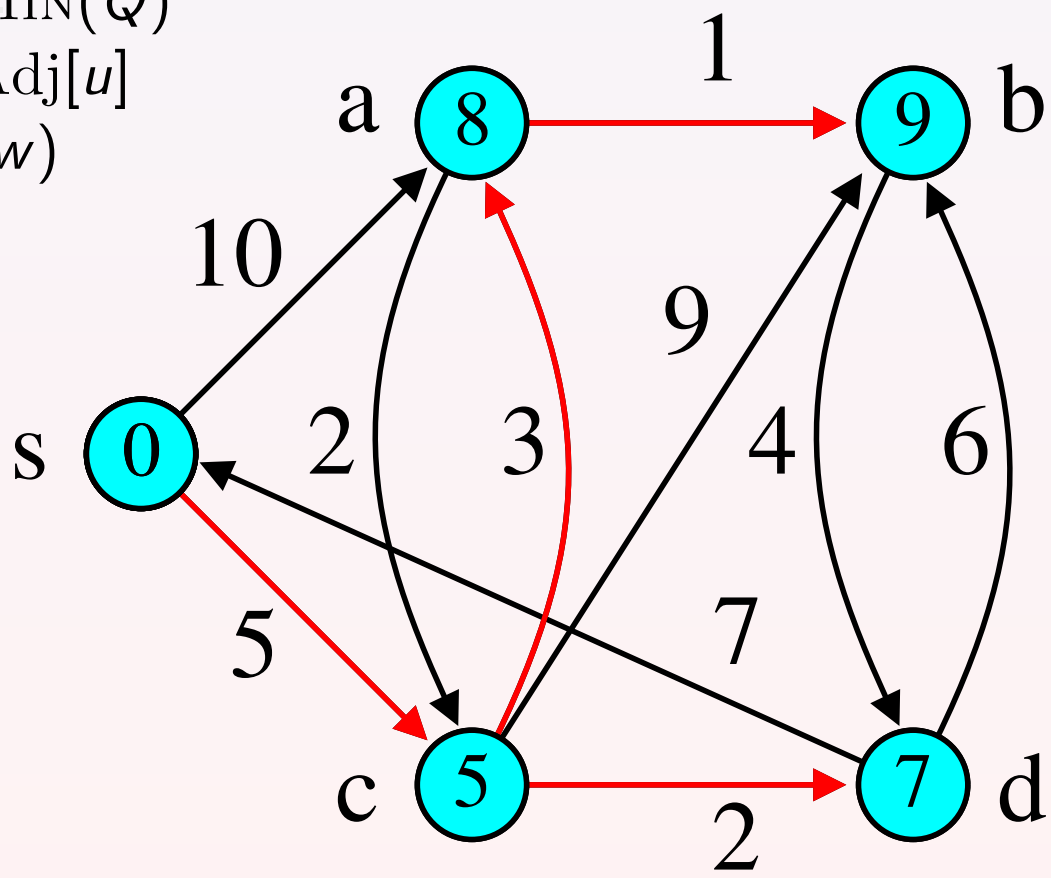
$Q : \emptyset$



General graphs

DIJSKTRA(G, w, s) // weights are non-negative

(1) INITIALIZE(G, s)
(2) Build a priority queue Q on the vertices (keys are d values)
(3) **while** Q is not empty
(4) $\quad u \leftarrow$ EXTRACTMIN(Q)
(5) $\quad$ **foreach** $v \in G.\mathrm{Adj}[u]$
(6) $\quad\quad$ RELAX(u, v, w)



General graphs

DIJSKTRA(G, w, s) // weights are non-negative

(1) INITIALIZE(G, s)
(2) Build a priority queue Q on the vertices (keys are d values)
(3) **while** Q is not empty
(4) $\quad u \leftarrow$ EXTRACTMIN(Q)
(5) $\quad$ **foreach** $v \in G.\mathrm{Adj}[u]$
(6) $\quad\quad$ RELAX(u, v, w)

RELAX(u, v, w)

(1) **if** $u.d + w(u, v) < v.d$
(2) $\quad$ DECREASEKEY$(Q, v, u.d + w(u, v))$
(3) $\quad v.\pi \leftarrow u$

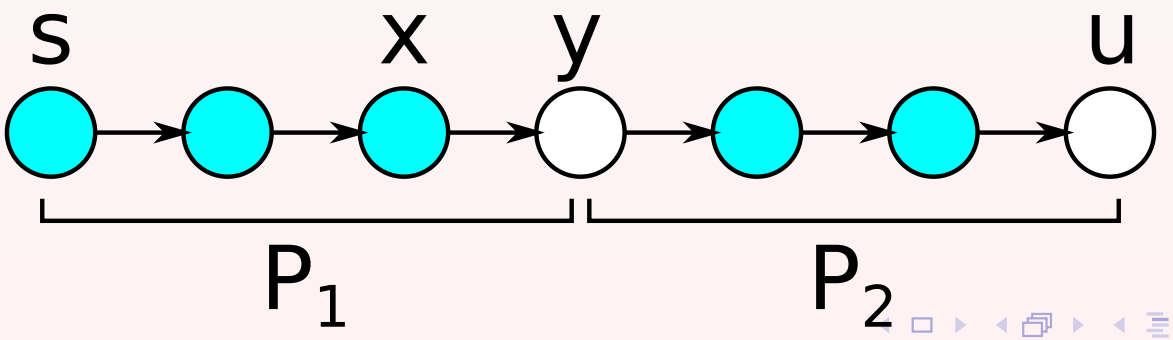
Line 1: $\Theta(V)$
Line 2: $\Theta(V)$
Line 4: $O(V \log V)$
Line 6: $O(E \log V)$
Total: $O((V + E) \log V)$

Correctness

- Claim: During the algorithm, $u.d = \delta(s, u)$ for every $u \notin Q$.
- The claim is proved using induction on time.
- Consider the time t just before a vertex u is removed from Q.

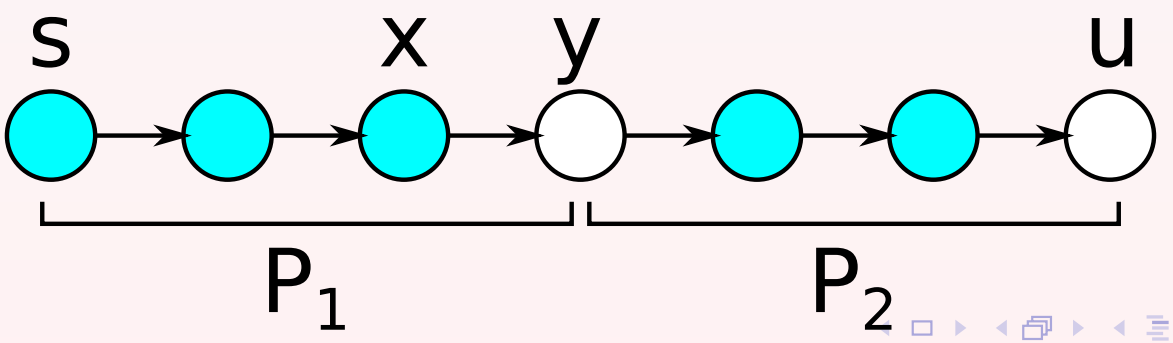
Let

- $P =$ a shortest path from s to u.
- $y =$ 1st vertex on P which is in Q.
- $x =$ the vertex preceding vertex on P ($x \notin Q$).
- $P_1 =$ prefix of P until y.
- $P_2 =$ suffix of P from y.



Correctness

1. P_1 is a shortest path from s to y.
2. By induction, $x.d = \delta(s, x)$ at the time x was removed from Q.
3. After x was removed from Q, $\textsc{Relax}(x, y, w)$ is called. From 1,2, $y.d \leq x.d + w(x, y) = \delta(s, x) + w(x, y) = \delta(s, y)$.
4. From 1, $\delta(s, y) = w(P_1) \leq w(P_1) + w(P_2) = w(P) = \delta(s, u)$.
5. From 3,4, $y.d \leq \delta(s, y) \leq \delta(s, u) \leq u.d$.
6. If $y = u$ then $y.d = u.d$. Otherwise, since u is removed from Q at time t, $y.d \geq u.d$.
7. From 5,6, $y.d = \delta(s, y) = \delta(s, u) = u.d$.



ALGORITHM DESIGN TECHNIQUES

Dynamic Programming: Matrix-Chain Multiplication – Elements of Dynamic Programming – Longest Common Subsequence- Greedy Algorithms: – Elements of the Greedy Strategy- An Activity-Selection Problem - Huffman Coding.

Dynamic Programming

Dynamic programming is a technique that breaks the problems into sub-problems, and saves the result for future purposes so that we do not need to compute the result again. The subproblems are optimized to optimize the overall solution is known as optimal substructure property.

The main use of dynamic programming is to solve optimization problems. Here, optimization problems mean that when we are trying to find out the minimum or the maximum solution of a problem. The dynamic programming guarantees to find the optimal solution of a problem if the solution exists.

The definition of dynamic programming says that it is a technique for solving a complex problem by first breaking into a collection of simpler subproblems, solving each subproblem just once, and then storing their solutions to avoid repetitive computations.

How does the dynamic programming approach work?

The following are the steps that the dynamic programming follows:

- It breaks down the complex problem into simpler subproblems.
- It finds the optimal solution to these sub-problems.
- It stores the results of subproblems (memoization). The process of storing the results of subproblems is known as memorization.
- It reuses them so that same sub-problem is calculated more than once.
- Finally, calculate the result of the complex problem.

The above five steps are the basic steps for dynamic programming. The dynamic programming is applicable that are having properties such as:

Those problems that are having overlapping subproblems and optimal substructures. Here, optimal substructure means that the solution of optimization problems can be obtained by simply combining the optimal solution of all the subproblems.

In the case of dynamic programming, the space complexity would be increased as we are storing the intermediate results, but the time complexity would be decreased.

Approaches of dynamic programming

There are two approaches to dynamic programming:

- Top-down approach
- Bottom-up approach

Top-down approach

The top-down approach follows the memorization technique, while bottom-up approach follows the tabulation method. Here memorization is equal to the sum of recursion and caching. Recursion means calling the function itself, while caching means storing the intermediate results.

Advantages

- It is very easy to understand and implement.
- It solves the subproblems only when it is required.
- It is easy to debug.

Disadvantages

It uses the recursion technique that occupies more memory in the call stack. Sometimes when the recursion is too deep, the stack overflow condition will occur.

It occupies more memory that degrades the overall performance.

Let's understand dynamic programming through an example.

```
1. int fib(int n)
2. {
3.    if(n<0)
4.    error;
5.  if(n==0)
```

```
6.  return 0;
7.  if(n==1)
8.  return 1;
9.  sum = fib(n-1) + fib(n-2);
10. }
```

In the above code, we have used the recursive approach to find out the Fibonacci series. When the value of 'n' increases, the function calls will also increase, and computations will also increase. In this case, the time complexity increases exponentially, and it becomes 2^n.

One solution to this problem is to use the dynamic programming approach. Rather than generating the recursive tree again and again, we can reuse the previously calculated value. If we use the dynamic programming approach, then the time complexity would be O(n).

When we apply the dynamic programming approach in the implementation of the Fibonacci series, then the code would look like:

```
1.  static int count = 0;
2.  int fib(int n)
3.  {
4.  if(memo[n]!= NULL)
5.  return memo[n];
6.  count++;
7.    if(n<0)
8.    error;
9.   if(n==0)
10.  return 0;
11.  if(n==1)
12. return 1;
13. sum = fib(n-1) + fib(n-2);
14. memo[n] = sum;
15. }
```

In the above code, we have used the memorization technique in which we store the results in an array to reuse the values. This is also known as a top-down approach in which we move from the top and break the problem into sub-problems.

Bottom-Up approach

The bottom-up approach is also one of the techniques which can be used to implement the dynamic programming. It uses the tabulation technique to implement the dynamic programming approach. It solves the same kind of problems but it removes the recursion. If we remove the recursion, there is no stack overflow issue and no overhead of the recursive functions. In this tabulation technique, we solve the problems and store the results in a matrix.

There are two ways of applying dynamic programming:

- **Top-Down**
- **Bottom-Up**

The bottom-up is the approach used to avoid the recursion, thus saving the memory space. The bottom-up is an algorithm that starts from the beginning, whereas the recursive algorithm starts from the end and works backward. In the bottom-up approach, we start from the base case to find the answer for the end. As we know, the base cases in the Fibonacci series are 0 and 1. Since the bottom approach starts from the base cases, so we will start from 0 and 1.

Key points

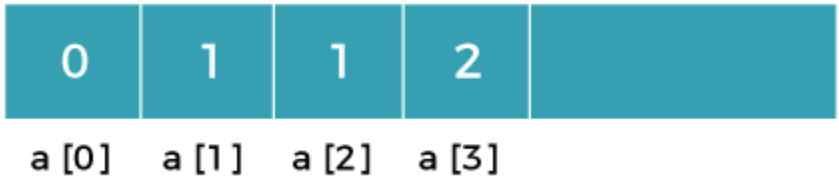
- We solve all the smaller sub-problems that will be needed to solve the larger sub-problems then move to the larger problems using smaller sub-problems.
- We use for loop to iterate over the sub-problems.
- The bottom-up approach is also known as the tabulation or table filling method.

Let's understand through an example.

Suppose we have an array that has 0 and 1 values at a[0] and a[1] positions, respectively shown as below:

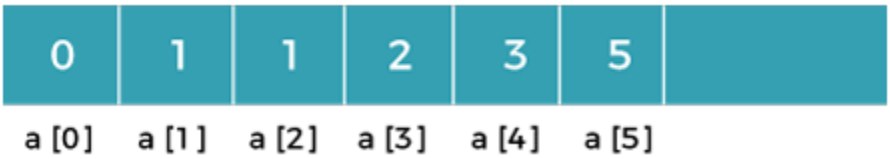
Since the bottom-up approach starts from the lower values, so the values at a[0] and a[1] are added to find the value of a[2] shown as below:

The value of a[3] will be calculated by adding a[1] and a[2], and it becomes 2 shown as below:



The value of a[4] will be calculated by adding a[2] and a[3], and it becomes 3 shown as below:

The value of a[5] will be calculated by adding the values of a[4] and a[3], and it becomes 5 shown as below:



The code for implementing the Fibonacci series using the bottom-up approach is given below:

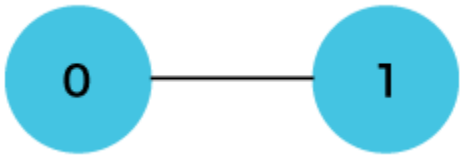
```
1. int fib(int n)
2. {
3.    int A[];
4.    A[0] = 0, A[1] = 1;
5.    for( i=2; i<=n; i++)
6.    {
7.        A[i] = A[i-1] + A[i-2]
8.    }
```

```
9.      return A[n];
10. }
```

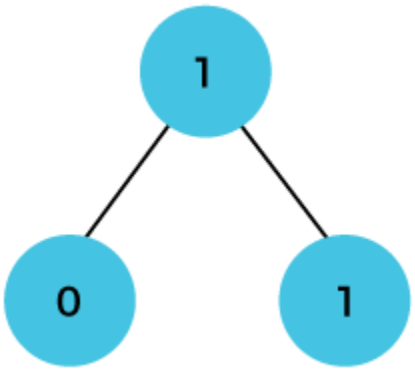
In the above code, base cases are 0 and 1 and then we have used for loop to find other values of Fibonacci series.

Let's understand through the diagrammatic representation.

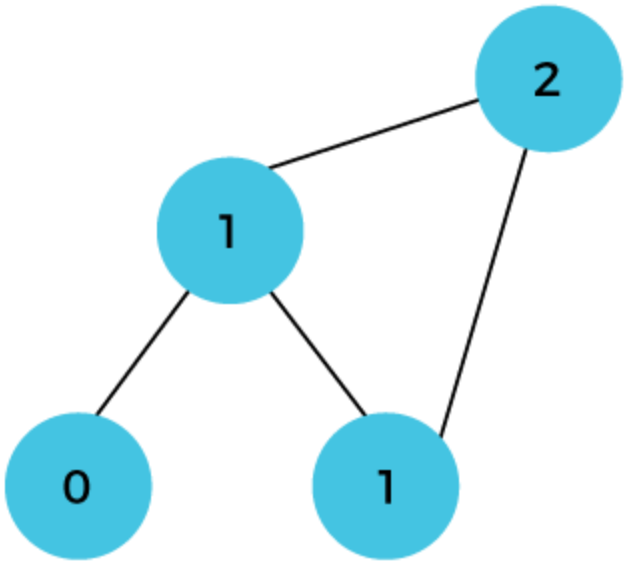
Initially, the first two values, i.e., 0 and 1 can be represented as:



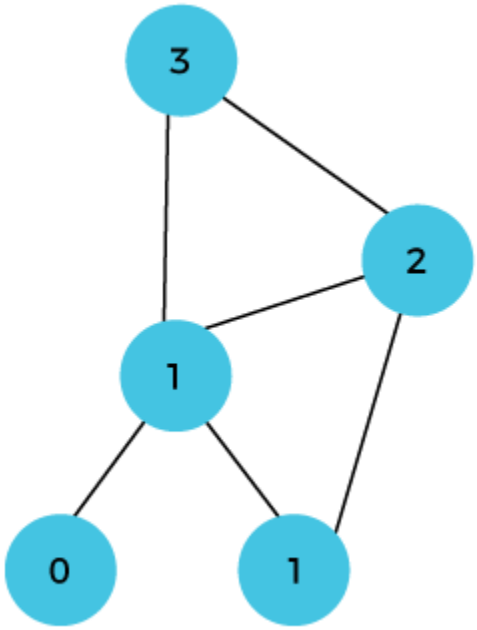
When i=2 then the values 0 and 1 are added shown as below:



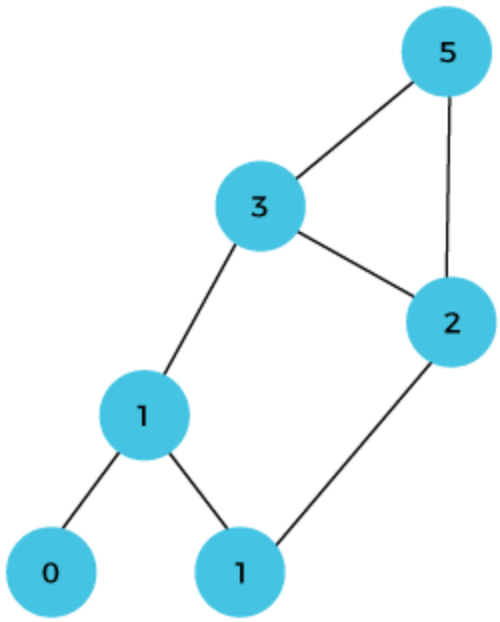
When i=3 then the values 1and 1 are added shown as below:



When i=4 then the values 2 and 1 are added shown as below:



When i=5, then the values 3 and 2 are added shown as below:



In the above case, we are starting from the bottom and reaching to the top.

Components of Dynamic Programming

The major components in any Dynamic Programming solution are:

1. Stages
2. States and state variables
3. State Transition
4. Optimal Choice

We now know how Dynamic Programming works, but how do we start building a Dynamic Programming solution? The major components needed to construct a Dynamic Programming solution are discussed below.

Stages

When a complex problem is divided into several subproblems, each subproblem forms a stage of the solution. After calculating the solution for each stage and choosing the best ones we get to the final optimized solution.

For example, in the previous instance of the climbing stairs problem, we needed the minimum cost to reach the **(i-1)** th and the **(i-2)** th stair first. That means to find ans[n], we had to calculate

the values of ans[n-1] and ans[n-2] first. Hence, here ans[n-1] and ans[n-2] are different stages whose solution we need to find to get to the value of ans[n] finally.

States and State Variables

Each subproblem can be associated with several states. States differ in their solutions because of different choices. A state for a subproblem is therefore defined more clearly based on a parameter, which is called the state variable. It is possible in some problems, that one than one variable is needed to define a state distinctly. In these cases, there are more than one state variables.

For example, we could reach the **nth** stair by either jumping up from the **(n-2)th** stair or by climbing one step from the **(n-1)th** stair. Hence, these are the two states related to reaching the **nth** stair in minimum cost and the variable which is the stair number from which we are climbing up to the current stair, is the state variable that defines a state uniquely.

State Transition

State Transition simply refers to how one subproblem relates to other subproblems. By using this state transition, we calculate our end solution.

For example, in the climbing stairs problem, the state transition was:

ans[n] = cost[n] + min(ans[n-1],ans[n-2])

Here ans[n] represents the minimum cost to reach the **nth** stair.

Using this state transition, we can know how different subproblems relate to each other, which can be used to compute the final optimal solution.

Optimal Choice

At each stage, we need to choose the option which leads to the most desirable solution. Choosing the most desirable option at every stage will eventually lead to an optimal solution in the end.

For example, in the climbing stairs problem, we need to make an optimal choice based on whether we get a minimum cost by clmbing from the stair directly beneath the current one, or by jumping from two stairs below. We need to make this optimal choice at each stair, to reach to the final solution at the end.

Elements Of Dynamic Programming

Three elements of the Dynamic Programming algorithm are :

1. Substructure
2. Table Structure

3. Bottom-Up Computation

The elements in a Dynamic Programming Solution are discussed below:

- To solve a given complex problem and to find its optimal solution, it is broken down into similar but smaller and easily computable problems called subproblems. Hence, the complete solution depends on many smaller problems and their solutions. We get to the final optimal solution after going through all subproblems and selecting the most optimal ones. This is the substructure element of any Dynamic Programming solution.
- Any Dynamic Programming solution involves storing the optimal solutions of the subproblems so that they don't have to be computed again and again. To store these solutions a table structure is needed. So, for example arrays in C++ or ArrayList in Java can be used. By using this structured table, the solutions of previous subproblems are reused.
- The solutions to subproblems need to be computed first to be reused again. This is called Bottom-Up Computation because we start storing values from the bottom and then consequently upwards. The solutions to the smaller subproblems are combined to get the final solution to the original problem.

Matrix Chain Multiplication

It is a Method under Dynamic Programming in which previous output is taken as input for next.

Here, Chain means one matrix's column is equal to the second matrix's row [always].

In general:

If $A = \lfloor a_{ij} \rfloor$ is a p x q matrix
$B = \lfloor b_{ij} \rfloor$ is a q x r matrix
$C = \lfloor c_{ij} \rfloor$ is a p x r matrix

Then

$$AB = C \text{ if } c_{ij} = \sum_{k=1}^{q} a_{ik}\ b_{kj}$$

Given following matrices $\{A_1, A_2, A_3, \ldots A_n\}$ and we have to perform the matrix multiplication, which can be accomplished by a series of matrix multiplications

$A_1 xA_2 x,A_3 x.....x A_n$

Matrix Multiplication operation is **associative** in nature rather commutative. By this, we mean that we have to follow the above matrix order for multiplication but we are free to **parenthesize** the above multiplication depending upon our need.

In general, for $1 \leq i \leq p$ and $1 \leq j \leq r$

$$C[i, j] = \sum_{k=1}^{q} A[i, k]B[k, j]$$

It can be observed that the total entries in matrix 'C' is 'pr' as the matrix is of dimension p x r Also each entry takes O (q) times to compute, thus the total time to compute all possible entries for the matrix 'C' which is a multiplication of 'A' and 'B' is proportional to the product of the dimension p q r.

It is also noticed that we can save the number of operations by reordering the parenthesis.

Example1: Let us have 3 matrices, A_1,A_2,A_3 of order (10 x 100), (100 x 5) and (5 x 50) respectively.

Three Matrices can be multiplied in two ways:

1. **$A_1,(A_2,A_3)$:** First multiplying(A_2 and A_3) then multiplying and resultant withA_1.
2. **$(A_1,A_2),A_3$:** First multiplying(A_1 and A_2) then multiplying and resultant withA_3.

No of Scalar multiplication in Case 1 will be:

1. (100 x 5 x 50) + (10 x 100 x 50) = 25000 + 50000 = 75000

No of Scalar multiplication in Case 2 will be:

1. (100 x 10 x 5) + (10 x 5 x 50) = 5000 + 2500 = 7500

To find the best possible way to calculate the product, we could simply parenthesis the expression in every possible fashion and count each time how many scalar multiplication are required.

Matrix Chain Multiplication Problem can be stated as "find the optimal parenthesization of a chain of matrices to be multiplied such that the number of scalar multiplication is minimized".

Number of ways for parenthesizing the matrices:

There are very large numbers of ways of parenthesizing these matrices. If there are n items, there are (n-1) ways in which the outer most pair of parenthesis can place.

$(A_1)\ (A_2,A_3,A_4,.....A_n)$
Or $(A_1,A_2)\ \ (A_3,A_4\ .....A_n)$
Or $(A_1,A_2,A_3)\ \ (A_4\ .....A_n)$
.....

$Or(A_1,A_2,A_3.....A_{n-1})\ (A_n)$

It can be observed that after splitting the kth matrices, we are left with two parenthesized sequence of matrices: one consist 'k' matrices and another consist 'n-k' matrices.

Now there are 'L' ways of parenthesizing the left sublist and 'R' ways of parenthesizing the right sublist then the Total will be L.R:

$$p(n) = \begin{cases} 1 & \text{if } n = 1 \\ \sum_{k=1}^{n-1} p(k)p(n-k) & \text{if } n \geq 2 \end{cases}$$

Also p (n) = c (n-1) where c (n) is the nth **Catalon number**

$$c(n) = \frac{1}{n+1}\binom{2n}{n}$$

On applying Stirling's formula we have

$$c(n) = \Omega\left(\frac{4^n}{n^{1.5}}\right)$$

Which shows that 4^n grows faster, as it is an exponential function, then $n^{1.5}$.

Development of Dynamic Programming Algorithm

1. Characterize the structure of an optimal solution.
2. Define the value of an optimal solution recursively.
3. Compute the value of an optimal solution in a bottom-up fashion.
4. Construct the optimal solution from the computed information.

Dynamic Programming Approach

Let $A_{i,j}$ be the result of multiplying matrices i through j. It can be seen that the dimension of $A_{i,j}$ is p_{i-1} x p_j matrix.

Dynamic Programming solution involves breaking up the problems into subproblems whose solution can be combined to solve the global problem.

At the greatest level of parenthesization, we multiply two matrices

$A_{1.....n}=A_{1....k}$ x $A_{k+1....n})$

Thus we are left with two questions:

- How to split the sequence of matrices?
- How to parenthesize the subsequence $A_{1.....k}$ and$A_{k+1.....n}$?

One possible answer to the first question for finding the best value of 'k' is to check all possible choices of 'k' and consider the best among them. But that it can be observed that checking all possibilities will lead to an exponential number of total possibilities. It can also be noticed that there exists only O (n^2) different sequence of matrices, in this way do not reach the exponential growth.

Step1: Structure of an optimal parenthesization: Our first step in the dynamic paradigm is to find the optimal substructure and then use it to construct an optimal solution to the problem from an optimal solution to subproblems.

Let $A_{i....j}$ where i≤ j denotes the matrix that results from evaluating the product

A_i A_{i+1}....A_j.

If i < j then any parenthesization of the product A_i A_{i+1} .....A_j must split that the product between A_k and A_{k+1} for some integer k in the range i ≤ k ≤ j. That is for some value of k, we first compute the matrices $A_{i.....k}$ & $A_{k+1....j}$ and then multiply them together to produce the final product $A_{i....j}$. The cost of computing $A_{i....k}$ plus the cost of computing $A_{k+1....j}$ plus the cost of multiplying them together is the cost of parenthesization.

Step 2: A Recursive Solution: Let m [i, j] be the minimum number of scalar multiplication needed to compute the matrix $A_{i....j}$.

If i=j the chain consist of just one matrix $A_{i....i}=A_i$ so no scalar multiplication are necessary to compute the product. Thus m [i, j] = 0 for i= 1, 2, 3....n.

If i<j we assume that to optimally parenthesize the product we split it between A_k and A_{k+1} where $i \leq k \leq j$. Then m [i,j] equals the minimum cost for computing the subproducts $A_{i....k}$ and $A_{k+1....j}$+ cost of multiplying them together. We know A_i has dimension p_{i-1} x p_i, so computing the product $A_{i....k}$ and $A_{k+1....j}$takes $p_{i-1}\ p_k\ p_j$ scalar multiplication, we obtain

$$m [i,j] = m [i, k] + m [k + 1, j] + p_{i-1}\ p_k\ p_j$$

There are only (j-1) possible values for 'k' namely k = i, i+1.....j-1. Since the optimal parenthesization must use one of these values for 'k' we need only check them all to find the best.

So the minimum cost of parenthesizing the product $A_i\ A_{i+1}.....A_j$ becomes

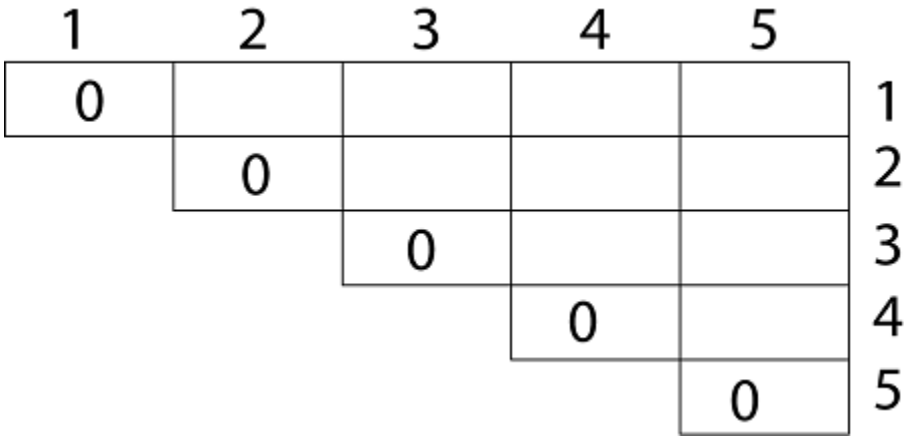
$$m [i,j] = \begin{cases} 0 & \text{if } i = j \\ \min\{m [i,k] + m [k+1,j] + p_{i-1}\ p_k p_j\} & \text{if } i < j \\ i \leq k < j & \end{cases}$$

To construct an optimal solution, let us define s [i,j] to be the value of 'k' at which we can split the product $A_i\ A_{i+1}$ .....A_j To obtain an optimal parenthesization i.e. s [i, j] = k such that

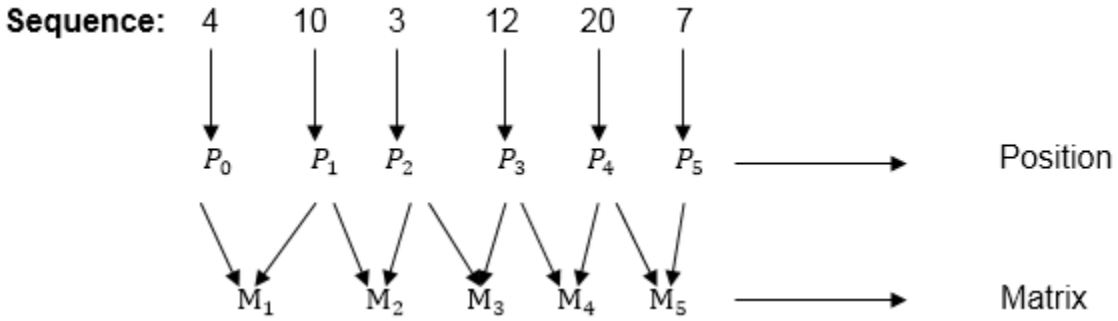
$$m [i,j] = m [i, k] + m [k + 1, j] + p_{i-1}\ p_k\ p_j$$

Example of Matrix Chain Multiplication

Example: We are given the sequence {4, 10, 3, 12, 20, and 7}. The matrices have size 4 x 10, 10 x 3, 3 x 12, 12 x 20, 20 x 7. We need to compute M [i,j], 0 ≤ i, j≤ 5. We know M [i, i] = 0 for all i.



Let us proceed with working away from the diagonal. We compute the optimal solution for the product of 2 matrices.



Here P_0 to P_5 are Position and M_1 to M_5 are matrix of size (p_i to p_{i-1})

On the basis of sequence, we make a formula

For M_i ⟶ p [i] as column

p [i-1] as row

In Dynamic Programming, initialization of every method done by '0'.So we initialize it by '0'.It will sort out diagonally.

We have to sort out all the combination but the minimum output combination is taken into consideration.

Calculation of Product of 2 matrices:

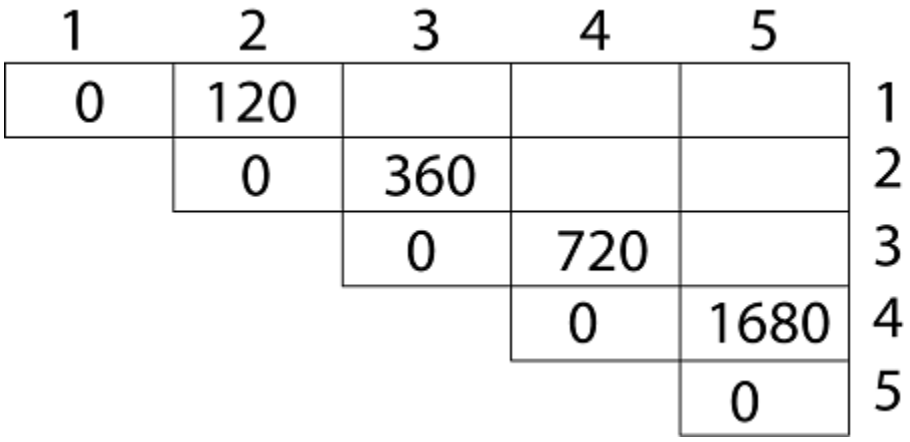
1. m (1,2) = m_1 x m_2
   = 4 x 10 x 10 x 3
   = 4 x 10 x 3 = 120

2. m (2, 3) = m_2 x m_3
   = 10 x 3 x 3 x 12
   = 10 x 3 x 12 = 360

3. m (3, 4) = m_3 x m_4
   = 3 x 12 x 12 x 20
   = 3 x 12 x 20 = 720

4. m (4,5) = m_4 x m_5
   = 12 x 20 x 20 x 7

= 12 x 20 x 7 = 1680



- We initialize the diagonal element with equal i,j value with '0'.
- After that second diagonal is sorted out and we get all the values corresponded to it

Now the third diagonal will be solved out in the same way.

Now product of 3 matrices:

M [1, 3] = $M_1 M_2 M_3$

1. There are two cases by which we can solve this multiplication: (M_1 x M_2) + M_3, M_1+ (M_2x M_3)
2. After solving both cases we choose the case in which minimum output is there.

$$M[1,3] = \min \begin{cases} M[1,2] + M[3,3] + p_0 p_2 p_3 = 120 + 0 + 4.3.12 = 264 \\ M[1,1] + M[2,3] + p_0 p_1 p_3 = 0 + 360 + 4.10.12 = 840 \end{cases}$$

M [1, 3] =264

As Comparing both output **264** is minimum in both cases so we insert **264** in table and (M_1 x M_2) + M_3 this combination is chosen for the output making.

M [2, 4] = $M_2 M_3 M_4$

1. There are two cases by which we can solve this multiplication: (M_2x M_3)+M_4, M_2+(M_3 x M_4)
2. After solving both cases we choose the case in which minimum output is there.

$$M[2,4] = \min \begin{cases} M[2,3] + M[4,4] + p_1 p_3 p_4 = 360 + 0 + 10.12.20 = 2760 \\ M[2,2] + M[3,4] + p_1 p_2 p_4 = 0 + 720 + 10.3.20 = 1320 \end{cases}$$

M [2, 4] = 1320

As Comparing both output **1320** is minimum in both cases so we insert **1320** in table and $M_2+(M_3 \times M_4)$ this combination is chosen for the output making.

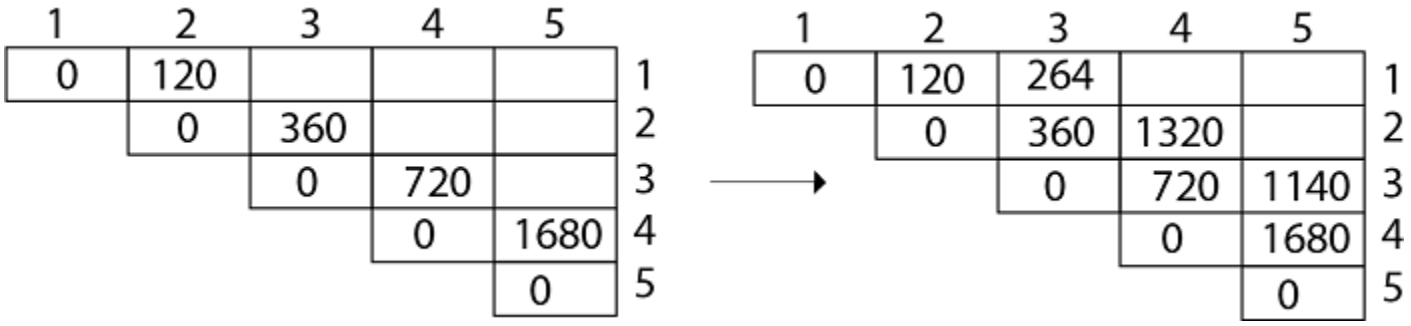
M [3, 5] = M_3 M_4 M_5

1. There are two cases by which we can solve this multiplication: ($M_3 \times M_4$) + M_5, M_3+ ($M_4 \times M_5$)
2. After solving both cases we choose the case in which minimum output is there.

$$M[3,5] = \min \begin{cases} M[3,4] + M[5,5] + p_2p_4p_5 = 720 + 0 + 3.20.7 = 1140 \\ M[3,3] + M[4,5] + p_2p_3p_5 = 0 + 1680 + 3.12.7 = 1932 \end{cases}$$

M [3, 5] = 1140

As Comparing both output **1140** is minimum in both cases so we insert **1140** in table and ($M_3 \times M_4$) + M_5this combination is chosen for the output making.



Now Product of 4 matrices:

M [1, 4] = M_1 M_2 M_3 M_4

There are three cases by which we can solve this multiplication:

1. ($M_1 \times M_2 \times M_3$) M_4
2. $M_1 \times (M_2 \times M_3 \times M_4)$
3. $(M_1 \times M_2) \times (M_3 \times M_4)$

After solving these cases we choose the case in which minimum output is there

$$M[1,4] = \min \begin{cases} M[1,3] + M[4,4] + p_0p_3p_4 = 264 + 0 + 4.12.20 = 1224 \\ M[1,2] + M[3,4] + p_0p_2p_4 = 120 + 720 + 4.3.20 = 1080 \\ M[1,1] + M[2,4] + p_0p_1p_4 = 0 + 1320 + 4.10.20 = 2120 \end{cases}$$

M [1, 4] =1080

As comparing the output of different cases then '**1080**' is minimum output, so we insert 1080 in the table and (M_1 xM_2) x (M_3 x M_4) combination is taken out in output making,

M [2, 5] = M_2 M_3 M_4 M_5

There are three cases by which we can solve this multiplication:

1. (M_2 x M_3 x M_4)x M_5
2. M_2 x(M_3 x M_4 x M_5)
3. (M_2 x M_3)x (M_4 x M_5)

After solving these cases we choose the case in which minimum output is there

$$M[2,5] = \min \begin{cases} M[2,4] + M[5,5] + p_1p_4p_5 = 1320 + 0 + 10.20.7 = 2720 \\ M[2,3] + M[4,5] + p_1p_3p_5 = 360 + 1680 + 10.12.7 = 2880 \\ M[2,2] + M[3,5] + p_1p_2p_5 = 0 + 1140 + 10.3.7 = 1350 \end{cases}$$

M [2, 5] = 1350

As comparing the output of different cases then '**1350**' is minimum output, so we insert 1350 in the table and M_2 x(M_3 x M_4 xM_5)combination is taken out in output making.

1	2	3	4	5	
0	120	264			1
	0	360	1320		2
		0	720	1140	3
			0	1680	4
				0	5

→

1	2	3	4	5	
0	120	264	1080		1
	0	360	1320	1350	2
		0	720	1140	3
			0	1680	4
				0	5

Now Product of 5 matrices:

M [1, 5] = M_1 M_2 M_3 M_4 M_5

There are five cases by which we can solve this multiplication:

1. (M_1 x M_2 xM_3 x M_4)x M_5
2. M_1 x(M_2 xM_3 x M_4 xM_5)
3. (M_1 x M_2 xM_3)x M_4 xM_5

4. $M_1 \times M_2 \times (M_3 \times M_4 \times M_5)$

After solving these cases we choose the case in which minimum output is there

$$M[1,5] = \min \begin{cases} M[1,4] + M[5,5] + p_0 p_4 p_5 = 1080 + 0 + 4.20.7 = 1544 \\ M[1,3] + M[4,5] + p_0 p_3 p_5 = 264 + 1680 + 4.12.7 = 2016 \\ M[1,2] + M[3,5] + p_0 p_2 p_5 = 120 + 1140 + 4.3.7 = 1344 \\ M[1,1] + M[2,5] + p_0 p_1 p_5 = 0 + 1350 + 4.10.7 = 1630 \end{cases}$$

M [1, 5] = 1344

As comparing the output of different cases then '**1344**' is minimum output, so we insert 1344 in the table and $M_1 \times M_2 \times (M_3 \times M_4 \times M_5)$combination is taken out in output making.

Final Output is:

1	2	3	4	5	
0	120	264	1080		1
	0	360	1320	1350	2
		0	720	1140	3
			0	1680	4
				0	5

→

1	2	3	4	5	
0	120	264	1080	1344	1
	0	360	1320	1350	2
		0	720	1140	3
			0	1680	4
				0	5

Step 3: Computing Optimal Costs: let us assume that matrix A_i has dimension $p_{i-1} \times p_i$ for i=1, 2, 3....n. The input is a sequence $(p_0, p_1, ...p_n)$ where length [p] = n+1. The procedure uses an auxiliary table m [1....n, 1.....n] for storing m [i, j] costs an auxiliary table s [1.....n, 1.....n] that record which index of k achieved the optimal costs in computing m [i, j].

The algorithm first computes m [i, j] ← 0 for i=1, 2, 3.....n, the minimum costs for the chain of length 1.

MATRIX-CHAIN-ORDER (p)

1. n length[p]-1
2. for i ← 1 to n
3. do m [i, i] ← 0
4. for l ← 2 to n // l is the chain length
5. do for i ← 1 to n-l + 1
6. do j ← i+ l -1
7. m[i,j] ← ∞
8. for k ← i to j-1
9. do q ← m [i, k] + m [k + 1, j] + $p_{i-1} p_k p_j$
10. If q < m [i,j]
11. then m [i,j] ← q
12. s [i,j] ← k

13. return m and s.

We will use table s to construct an optimal solution.

Step 1: Constructing an Optimal Solution:

PRINT-OPTIMAL-PARENS (s, i, j)

1. if i=j
2. then print "A"
3. else print "("
4. PRINT-OPTIMAL-PARENS (s, i, s [i, j])
5. PRINT-OPTIMAL-PARENS (s, s [i, j] + 1, j)
6. print ")"

Analysis: There are three nested loops. Each loop executes a maximum n times.

1. l, length, O (n) iterations.
2. i, start, O (n) iterations.
3. k, split point, O (n) iterations

Total Complexity is: O (n^3)

Algorithm with Explained Example

Question: P [7, 1, 5, 4, 2}

Solution: Here, P is the array of a dimension of matrices.

So here we will have 4 matrices:

A_{17x1} A_{21x5} A_{35x4} A_{44x2}
i.e.
First Matrix A_1 have dimension 7 x 1
Second Matrix A_2 have dimension 1 x 5
Third Matrix A_3 have dimension 5 x 4
Fourth Matrix A_4 have dimension 4 x 2

Let say,
From P = {7, 1, 5, 4, 2} - (Given)
And P is the Position
$p_0 = 7$, $p_1 = 1$, $p_2 = 5$, $p_3 = 4$, $p_4 = 2$.
Length of array P = number of elements in P
∴length (p)= 5

From step 3

Follow the steps in Algorithm in Sequence

According to Step 1 of Algorithm Matrix-Chain-Order

Step 1:

n ← length [p]-1

Where n is the total number of elements

And length [p] = 5

∴ n = 5 - 1 = 4

n = 4

Now we construct two tables m and s.

Table m has dimension [1.....n, 1.....n]

Table s has dimension [1.....n-1, 2.....n]

	1	2	3	4
1	0	35 ∞	48 ∞	42
2		0	20 ∞	28 ∞
3			0	
4				0

m-Table
[1....n, 1.....n]

	2	3	4
1	1	1	1
2		2	3
3			3

n-Table
[1....n-1, 2.....n]

Now, according to step 2 of Algorithm

1. **for** i ← 1 to n
2. **this** means: **for** i ← 1 to 4 (because n =4)
3. **for** i=1
4. m [i, i]=0
5. m [1, 1]=0
6. Similarly **for** i = 2, 3, 4
7. m [2, 2] = m [3,3] = m [4,4] = 0
8. i.e. fill all the diagonal entries "0" in the table m
9. Now,
10. l ← 2 to n
11. l ← 2 to 4 (because n =4)

Case 1:

1. When l - 2

for (i ← 1 to n - l + 1)
i ← 1 to 4 - 2 + 1
i ← 1 to 3

When i = 1
do j ← i + l - 1
j ← 1 + 2 - 1
j ← 2
i.e. j = 2
Now, m [i, j] ← ∞
i.e. m [1,2] ← ∞
Put ∞ in m [1, 2] table
for k ← i to j-1
k ← 1 to 2 - 1
k ← 1 to 1
k = 1
Now q ← m [i, k] + m [k + 1, j] + p_{i-1} p_k p_j
for l = 2
i = 1
j =2
k = 1
q ← m [1,1] + m [2,2] + p_0x p_1x p_2
and m [1,1] = 0
for i ← 1 to 4

$\therefore q \leftarrow 0 + 0 + 7 \times 1 \times 5$

$\mathbf{q \leftarrow 35}$

We have m [i, j] = m [1, 2] = ∞

Comparing q with m [1, 2]

q < m [i, j]

i.e. **35 < m [1, 2]**

35 < ∞

True

then, **m [1, 2] ← 35 (∴ m [i,j] ← q)**

s [1, 2] ← k

and the value of **k = 1**

s [1,2] ← 1

Insert "1" at dimension s [1, 2] in table s. And 35 at m [1, 2]

2. l remains 2

L = 2

i ← 1 to n - l + 1

i ← 1 to 4 - 2 + 1

i ← 1 to 3

for i = 1 done before

Now value of i becomes 2

i = 2

j ← i + l - 1

j ← 2 + 2 - 1

j ← 3

j = 3

m [i , j] ← ∞

i.e. **m [2,3]** ← ∞

Initially insert ∞ at m [2, 3]

Now, for k ← i to j - 1

k ← 2 to 3 - 1

k ← 2 to 2

i.e. k =2

Now, q ← m [i, k] + m [k + 1, j] + p_{i-1} p_k p_j

For l =2

i = 2

j = 3

k = 2

q ← m [2, 2] + m [3, 3] + $p_1 x\ p_2 \times p_3$

q ← 0 + 0 + 1 x 5 x 4

q ← 20

Compare q with m [i ,j]

If q < m [i,j]

i.e. 20 < m [2, 3]

20 < ∞

True

Then m [i,j] ← q

m [2, 3] ← 20

and s [2, 3] ← k

and k = 2

s [2,3] ← 2

3. Now i become 3

i = 3

l = 2

j ← i + l - 1

j ← 3 + 2 - 1

j ← 4

j = 4

Now, m [i, j] ← ∞

m [3,4] ← ∞

Insert ∞ at m [3, 4]

for k ← i to j - 1

k ← 3 to 4 - 1

k ← 3 to 3

i.e. k = 3

Now, q ← m [i, k] + m [k + 1, j] + p_{i-1} p_k p_j

i = 3

l = 2

j = 4

k = 3

q ← m [3, 3] + m [4,4] + p_2 x p_3 x p_4

q ← 0 + 0 + 5 x 2 x 4

q 40

Compare q with m [i, j]

If q < m [i, j]

40 < m [3, 4]

40 < ∞

True

Then, m [i,j] ← q

m [3,4] ← 40

and s [3,4] ← k

s [3,4] ← 3

Case 2: l becomes 3

L = 3

for i = 1 to n - l + 1

i = 1 to 4 - 3 + 1

i = 1 to 2

When i = 1

$j \leftarrow i + l - 1$

$j \leftarrow 1 + 3 - 1$

$j \leftarrow 3$

j = 3

Now, m [i,j] $\leftarrow \infty$

m [1, 3] $\leftarrow \infty$

for k $\leftarrow$ i to j - 1

k $\leftarrow$ 1 to 3 - 1

k $\leftarrow$ 1 to 2

Now we compare the value for both k=1 and k = 2. The minimum of two will be placed in m [i,j] or s [i,j] respectively.

(A) When k = 1	**(B)** When k =2
L =3,i=1,j=3	L =3,i=1,j=3
$q \leftarrow m[i, k] + m[k + 1, j] + p_{i-1}\ p_k\ p_j$	$q \leftarrow m[i, k] + m[k + 1, j] + p_{i-1}\ p_k\ p_j$
$q \leftarrow m[1,1] + m[2,3] + p_0 x\ p_1 x\ p_3$	$q \leftarrow m[1,2] + m[3,3] + p_0 x\ p_2 x\ p_3$
$q \leftarrow 0 + 20 + 7 \times 1 \times 4$	$q \leftarrow 35 + 0 + 7 \times 5 \times 3$
$q \leftarrow 48$	**$q \leftarrow 140$**

Now from above

Value of q become minimum for **k=1**

$\therefore$ m [i,j] $\leftarrow$ q

m [1,3] $\leftarrow$ **48**

Also m [i,j] > q

i.e. 48 $< \infty$

$\therefore$ m [i , j] $\leftarrow$ q

m [i, j] $\leftarrow$ 48

and s [i,j] $\leftarrow$ k

i.e. m [1,3] $\leftarrow$ **48**

s [1, 3] ← 1

Now i become 2

i = 2

l = 3

then j ← i + l -1

j ← 2 + 3 - 1

j ← 4

j = 4

so m [i,j] ← ∞

m [2,4] ← ∞

Insert initially ∞ at m [2, 4]

for k ← i to j-1

k ← 2 to 4 - 1

k ← 2 to 3

Here, also find the minimum value of m [i,j] for two values of k = 2 and k =3

(A) When k =2	**(B)** When k = 3
i=2 l=3, j=4	i=2,l=3, j=4
q ⟵ m [i, k] + m [k + 1, j] + p_{i-1} p_k p_j	q ⟵ m [i, k] + m [k + 1, j] + p_{i-1} p_k p_j
q ⟵m [2,2] + m [3,4] + p_1x p_2xp_4	q ⟵m [2,3] + m [4,4] +p_1xp_3x p_4
q ⟵0 + 40 +7 x 5 x 2	q ⟵20 + 0 + 1 x 4 x 2
q ⟵110	**q ⟵28**

1. But 28 < ∞
2. So m [i,j] ← q
3. And q ← 28
4. m [2, 4] ← 28
5. and s [2, 4] ← 3
6. e. It means in s table at s [2,4] insert 3 and at m [2,4] insert 28.

Case 3: l becomes 4

L = 4

For i ← 1 to n-l + 1

i ← 1 to 4 - 4 + 1

i ← 1

i = 1

do j ← i + l - 1

j ← 1 + 4 - 1

j ← 4

j = 4

Now m [i,j] ← ∞

m [1,4] ← ∞

for k ← i to j -1

k ← 1 to 4 - 1

k ← 1 to 3

When k = 1

q ← m [i, k] + m [k + 1, j] + p_{i-1} p_k p_j

q ← m [1,1] + m [2,4] + p_0xp_4x p_1

q ← 0 + 28 + 7 x 2 x 1

q ← 42

Compare q and m [i, j]

m [i,j] was ∞

i.e. m [1,4]

if q < m [1,4]

42< ∞

True

Then m [i,j] ← q

m [1,4] ← 42

and s [1,4] 1 ? k =1

When k = 2

L = 4, i=1, j = 4

q ← m [i, k] + m [k + 1, j] + p_{i-1} p_k p_j

q ← m [1, 2] + m [3,4] + p_0 xp_2 xp_4

q ← 35 + 40 + 7 x 5 x 2

q ← 145

Compare q and m [i,j]

Now m [i, j]

i.e. m [1,4] contains 42.

So if q < m [1, 4]

But 145 less than or not equal to m [1, 4]

So 145 less than or not equal to 42.

So no change occurs.

When k = 3

l = 4

i = 1

j = 4

$q \leftarrow m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j$

$q \leftarrow m[1, 3] + m[4,4] + p_0 x p_3 x p_4$

$q \leftarrow 48 + 0 + 7 \times 4 \times 2$

$q \leftarrow 114$

Again q less than or not equal to m [i, j]

i.e. 114 less than or not equal to m [1, 4]

114 less than or not equal to 42

So no change occurs. So the value of m [1, 4] remains 42. And value of s [1, 4] = 1

Now we will make use of only s table to get an optimal solution.

Use of step 4 Algorithm

Initial call of step 4 is (s, 1, n)

Where i=1

j = n and n= 4

i≠ j

else part

print "("

Print-optimal-Parens (s, i, s [i, j])

Print-optimal-Parens (s, 1, 4)

here i==j yes

print "A_1"

Print-optimal-Parens (s, s [i,j] + 1,j]

Print-optimal-Parens (s, 2, 4)

print "("

Print-optimal (s, 2, 3)

Print ")"

Print-optimal (s, 4, 4)

Print ")"

Starting from the beginning we are parenthesizing Matrix Chain Multiplication:

$(\mathbf{A_1})\ ((\mathbf{A_2}\ \mathbf{A_3})\mathbf{A_4}))$

Longest Common Sequence (LCS)

A subsequence of a given sequence is just the given sequence with some elements left out.

Given two sequences X and Y, we say that the sequence Z is a common sequence of X and Y if Z is a subsequence of both X and Y.

In the longest common subsequence problem, we are given two sequences X = (x_1 x_2....x_m) and Y = (y_1 y_2 y_n) and wish to find a maximum length common subsequence of X and Y. LCS Problem can be solved using dynamic programming.

Characteristics of Longest Common Sequence

A brute-force approach we find all the subsequences of X and check each subsequence to see if it is also a subsequence of Y, this approach requires exponential time making it impractical for the long sequence.

Given a sequence X = (x_1 x_2.....x_m) we define the ith prefix of X for i=0, 1, and 2...m as X_i= (x_1 x_2.....x_i). For example: if X = (A, B, C, B, C, A, B, C) then X_4= (A, B, C, B)

Optimal Substructure of an LCS: Let X = (x_1 x_2....x_m) and Y = (y_1 y_2.....) y_n) be the sequences and let Z = (z_1 z_2.....z_k) be any LCS of X and Y.

- If x_m = y_n, then z_k=x_m=y_n and Z_{k-1} is an LCS of $X_{m-1}$and Y_{n-1}
- If $x_m \neq y_n$, then $z_k \neq x_m$ implies that Z is an LCS of $X_{m-1}$and Y.
- If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is an LCS of X and Y_{n-1}

Step 2: Recursive Solution: LCS has overlapping subproblems property because to find LCS of X and Y, we may need to find the LCS of X_{m-1} and Y_{n-1}. If $x_m \neq y_n$, then we must solve two subproblems finding an LCS of X and Y_{n-1}.Whenever of these LCS's longer is an LCS of x and y. But each of these subproblems has the subproblems of finding the LCS of X_{m-1} and Y_{n-1}.

Let c [i,j] be the length of LCS of the sequence $X_i$and Y_j.If either i=0 and j =0, one of the sequences has length 0, so the LCS has length 0. The optimal substructure of the LCS problem given the recurrence formula

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ c[i-1, j-1]1 & \text{if } i,j > 0 \text{ and } x_i = y_j \\ \max(c[i, j-1], c[i-1, j]) & \text{if } i,j > 0 \text{ and } x_i \neq y_j \end{cases}$$

Step3: Computing the length of an LCS: let two sequences X = (x_1 x_2.....x_m) and Y = (y_1 y_2..... y_n) as inputs. It stores the c [i,j] values in the table c [0.....m,0.....n].Table b [1.....m, 1.....n] is maintained which help us to construct an optimal solution. c [m, n] contains the length of an LCS of X,Y.

Algorithm of Longest Common Sequence

LCS-LENGTH (X, Y)

1. m ← length [X]
2. n ← length [Y]
3. for i ← 1 to m

4. do c [i,0] ← 0
5. for j ← 0 to m
6. do c [0,j] ← 0
7. for i ← 1 to m
8. do for j ← 1 to n
9. do if $x_i = y_j$
10. then c [i,j] ← c [i-1,j-1] + 1
11. b [i,j] ← "↖"
12. else if c[i-1,j] ≥ c[i,j-1]
13. then c [i,j] ← c [i-1,j]
14. b [i,j] ← "↑"
15. else c [i,j] ← c [i,j-1]
16. b [i,j] ← "← "
17. return c and b.

Example of Longest Common Sequence

Example: Given two sequences X [1...m] and Y [1.....n]. Find the longest common subsequences to both.

x: A	B	C	B	D	A	B
y: B	D	C	A	B	A	

here X = (A,B,C,B,D,A,B) and Y = (B,D,C,A,B,A)

m = length [X] and n = length [Y]

m = 7 and n = 6

Here $x_1 = x[1] = A$ $y_1 = y[1] = B$

$x_2 = B$ $y_2 = D$

$x_3 = C$ $y_3 = C$

$x_4 = B$ $y_4 = A$

$x_5 = D$ $y_5 = B$

$x_6 = A$ $y_6 = A$

$x_7 = B$

Now fill the values of c [i, j] in m x n table

Initially, for i=1 to 7 c [i, 0] = 0

For j = 0 to 6 c [0, j] = 0

That is:

i	j	0	1	2	3	4	5	6
		y_1	B	D	C	A	B	A
0	X_1	0	0	0	0	0	0	0
1	A	0						
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Now for i=1 and j = 1

x_1 and y_1 we get $x_1 \neq y_1$ i.e. A ≠ B

And c [i-1,j] = c [0, 1] = 0

c [i, j-1] = c [1,0] = 0

That is, c [i-1,j]= c [i, j-1] so c [1, 1] = 0 and b [1, 1] = ' ↑ '

Now for i=1 and j = 2

x_1 and y_2 we get $x_1 \neq y_2$ i.e. A ≠ D

c [i-1,j] = c [0, 2] = 0

c [i, j-1] = c [1,1] = 0

That is, c [i-1,j]= c [i, j-1] and c [1, 2] = 0 b [1, 2] = ' ↑ '

Now for i=1 and j = 3

x_1 and y_3 we get $x_1 \neq y_3$ i.e. A ≠ C

c [i-1,j] = c [0, 3] = 0

c [i, j-1] = c [1,2] = 0

so c [1,3] = 0 b [1,3] = ' ↑'

Now for i=1 and j = 4

x_1 and y_4 we get. $x_1=y_4$ i.e A = A

c [1,4] = c [1-1,4-1] + 1

= c [0, 3] + 1

= 0 + 1 = 1

c [1,4] = 1

b [1,4] = ' ↖ '

Now for i=1 and j = 5

x_1 and y_5 we get $x_1 \neq y_5$

c [i-1,j] = c [0, 5] = 0

c [i, j-1] = c [1,4] = 1

Thus c [i, j-1] > c [i-1,j] i.e. c [1, 5] = c [i, j-1] = 1. So b [1, 5] = '←'

Now for i=1 and j = 6

x_1 and y_6 we get $x_1 = y_6$

c [1, 6] = c [1-1,6-1] + 1

= c [0, 5] + 1 = 0 + 1 = 1

c [1,6] = 1

b [1,6] = ' ↖ '

i	j	0	1	2	3	4	5	6
		y_1	B	D	C	A	B	A
0	X_1	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Now for i=2 and j = 1

We get x_2 and y_1 B = B i.e. $x_2 = y_1$

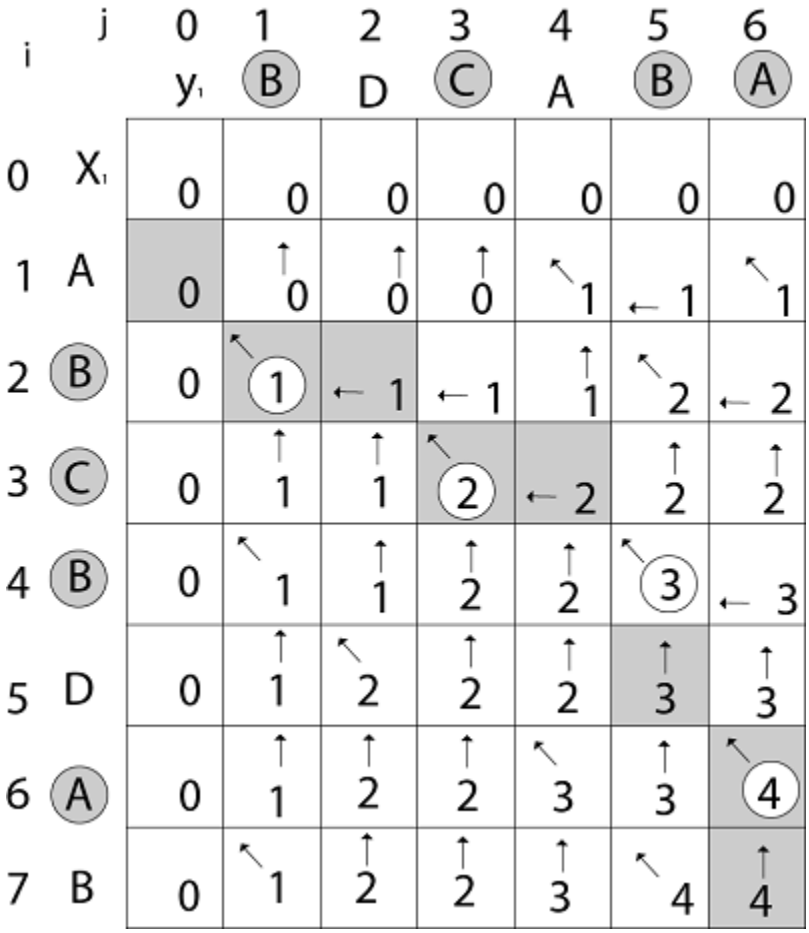
c [2,1] = c [2-1,1-1] + 1

= c [1, 0] + 1

= 0 + 1 = 1

c [2, 1] = 1 and b [2, 1] = ' ↖ '

Similarly, we fill the all values of c [i, j] and we get



Step 4: Constructing an LCS: The initial call is PRINT-LCS (b, X, X.length, Y.length)

PRINT-LCS (b, x, i, j)

1. if i=0 or j=0
2. then return
3. if b [i,j] = ' ↖ '
4. then PRINT-LCS (b,x,i-1,j-1)
5. print x_i
6. else if b [i,j] = ' ↑ '
7. then PRINT-LCS (b,X,i-1,j)
8. else PRINT-LCS (b,X,i,j-1)

Example: Determine the LCS of (1,0,0,1,0,1,0,1) and (0,1,0,1,1,0,1,1,0).

Solution: let X = (1,0,0,1,0,1,0,1) and Y = (0,1,0,1,1,0,1,1,0).

We are looking for c [8, 9]. The following table is built.

x =(1,0,0,1,0,1,0,1) y =(0,1,0,1,1,0,1,1,0)

i	j	0	1	2	3	4	5	6	7	8	9
		y_j	0	1	0	1	1	0	1	1	0
0	X_i	0	0	0	0	0	0	0	0	0	0
1	1	0	↑ 0	↖ (1)	↑ 1	↖ 1	← 1	↖ 1	↖ 0	↖ 1	← 1
2	0	0	↖ 1	↑ 1	↖ (2)	← 2	← 2	↖ 2	← 2	← 2	↖ 2
3	0	0	↖ 1	↑ 1	↖ 2	↑ 2	↑ 2	↖ (3)	← 3	← 3	↖ 3
4	1	0	↑ 1	↖ 2	↑ 2	↖ 3	↖ 3	↑ 3	↖ (4)	↖ 4	← 4
5	0	0	↖ 1	↑ 2	↖ 3	↑ 3	↑ 3	↖ 4	↑ 4	↑ 4	↖ 5
6	1	0	↑ 1	↖ 2	↑ 3	↖ 4	↖ 4	↑ 4	↖ 5	↖ (5)	↑ 5
7	0	0	↖ 1	↑ 2	↖ 3	↑ 4	↑ 4	↖ 5	↑ 5	↑ 5	↖ (6)
8	1	0	↑ 1	↖ 2	↑ 3	↖ 4	↖ 5	↑ 5	↖ 6	↖ 6	↑ 6

From the table we can deduct that LCS = 6. There are several such sequences, for instance (1,0,0,1,1,0) (0,1,0,1,0,1) and (0,0,1,1,0,1)

GREEDY ALGORITHMS

An algorithm is designed to achieve optimum solution for a given problem. In greedy algorithm approach, decisions are made from the given solution domain. As being greedy, the closest solution that seems to provide an optimum solution is chosen.

Greedy algorithms try to find a localized optimum solution, which may eventually lead to globally optimized solutions. However, generally greedy algorithms do not provide globally optimized solutions.

Counting Coins

This problem is to count to a desired value by choosing the least possible coins and the greedy approach forces the algorithm to pick the largest possible coin. If we are provided coins of ₹ 1, 2, 5 and 10 and we are asked to count ₹ 18 then the greedy procedure will be −

- **1** − Select one ₹ 10 coin, the remaining count is 8
- **2** − Then select one ₹ 5 coin, the remaining count is 3
- **3** − Then select one ₹ 2 coin, the remaining count is 1
- **4** − And finally, the selection of one ₹ 1 coins solves the problem

Though, it seems to be working fine, for this count we need to pick only 4 coins. But if we slightly change the problem then the same approach may not be able to produce the same optimum result.

For the currency system, where we have coins of 1, 7, 10 value, counting coins for value 18 will be absolutely optimum but for count like 15, it may use more coins than necessary. For example, the greedy approach will use 10 + 1 + 1 + 1 + 1 + 1, total 6 coins. Whereas the same problem could be solved by using only 3 coins (7 + 7 + 1)

Hence, we may conclude that the greedy approach picks an immediate optimized solution and may fail where global optimization is a major concern.

Examples

Most networking algorithms use the greedy approach. Here is a list of few of them −

- Travelling Salesman Problem
- Prim's Minimal Spanning Tree Algorithm
- Kruskal's Minimal Spanning Tree Algorithm
- Dijkstra's Minimal Spanning Tree Algorithm
- Graph - Map Coloring
- Graph - Vertex Cover
- Knapsack Problem
- Job Scheduling Problem

There are lots of similar problems that uses the greedy approach to find an optimum solution.

Components of Greedy Algorithm (Elements)

The components that can be used in the greedy algorithm are:

- **Candidate set:** A solution that is created from the set is known as a candidate set.
- **Selection function:** This function is used to choose the candidate or subset which can be added in the solution.
- **Feasibility function:** A function that is used to determine whether the candidate or subset can be used to contribute to the solution or not.
- **Objective function:** A function is used to assign the value to the solution or the partial solution.
- **Solution function:** This function is used to intimate whether the complete function has been reached or not.

Pseudo code of Greedy Algorithm

```
1.  Algorithm Greedy (a, n)
2.  {
3.    Solution : = 0;
4.   for i = 0 to n do
5.   {
6.      x: = select(a);
7.     if feasible(solution, x)
8.     {
9.        Solution: = union(solution , x)
10.    }
11.      return solution;
12.  } }
```

The above is the greedy algorithm. Initially, the solution is assigned with zero value. We pass the array and number of elements in the greedy algorithm. Inside the for loop, we select the element one by one and checks whether the solution is feasible or not. If the solution is feasible, then we perform the union.

Let's understand through an example.

Suppose there is a problem 'P'. I want to travel from A to B shown as below:

P : A → B

The problem is that we have to travel this journey from A to B. There are various solutions to go from A to B. We can go from A to B by **walk, car, bike, train, aeroplane**, etc. There is a constraint in the journey that we have to travel this journey within 12 hrs. If I go by train or aeroplane then only, I can cover this distance within 12 hrs. There are many solutions to this problem but there are only two solutions that satisfy the constraint.

If we say that we have to cover the journey at the minimum cost. This means that we have to travel this distance as minimum as possible, so this problem is known as a minimization problem. Till now, we have two feasible solutions, i.e., one by train and another one by air. Since travelling by train will lead to the minimum cost so it is an optimal solution. An optimal solution is also the feasible solution, but providing the best result so that solution is the optimal solution with the minimum cost. There would be only one optimal solution.

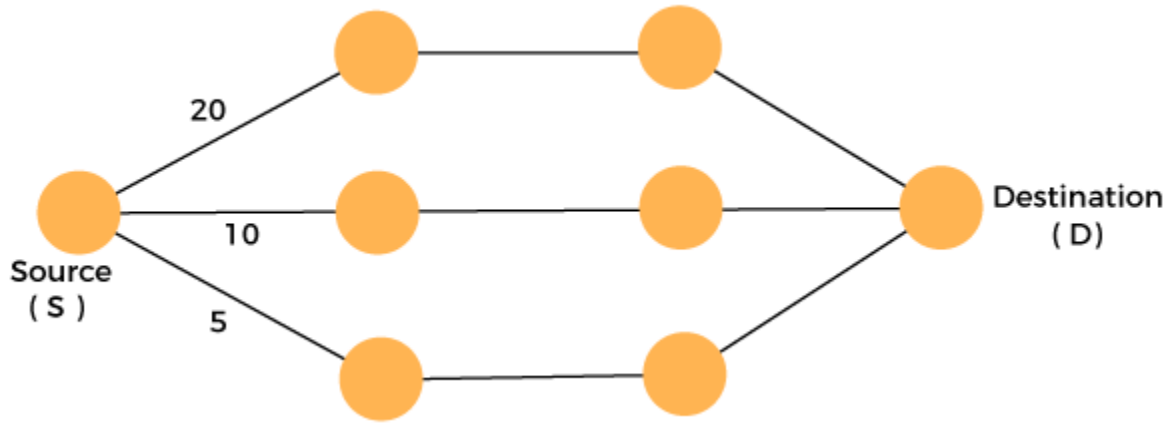
The problem that requires either minimum or maximum result then that problem is known as an optimization problem. Greedy method is one of the strategies used for solving the optimization problems.

Disadvantages of using Greedy algorithm

Greedy algorithm makes decisions based on the information available at each phase without considering the broader problem. So, there might be a possibility that the greedy solution does not give the best solution for every problem.

It follows the local optimum choice at each stage with a intend of finding the global optimum. Let's understand through an example.

Consider the graph which is given below:



We have to travel from the source to the destination at the minimum cost. Since we have three feasible solutions having cost paths as 10, 20, and 5. 5 is the minimum cost path so it is the optimal solution. This is the local optimum, and in this way, we find the local optimum at each stage in order to calculate the global optimal solution.

An Activity Selection Problem

The activity selection problem is a mathematical optimization problem. Our first illustration is the problem of scheduling a resource among several challenge activities. We find a greedy algorithm provides a well designed and simple method for selecting a maximum- size set of manually compatible activities.

Suppose S = {1, 2....n} is the set of n proposed activities. The activities share resources which can be used by only one activity at a time, e.g., Tennis Court, Lecture Hall, etc. Each Activity "i" has **start time** s_i and a **finish time** f_i, where $s_i \leq f_i$. If selected activity "i" take place meanwhile the half-open time interval $[s_i, f_i)$. Activities i and j are **compatible** if the intervals (s_i, f_i) and $[s_i, f_i)$ do not overlap (i.e. i and j are compatible if $s_i \geq f_i$ or $s_i \geq f_i$). The activity-selection problem chosen the maximum- size set of mutually consistent activities.

Algorithm Of Greedy- Activity Selector:

GREEDY- ACTIVITY SELECTOR (s, f)

1. n ← length [s]
2. A ← {1}
3. j ← 1.
4. for i ← 2 to n
5. do if $s_i \geq f_i$
6. then A ← A ∪ {i}
7. j ← i
8. return A

Example: Given 10 activities along with their start and end time as

S = (A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8 A9 A10)

Si = (1,2,3,4,7,8,9,9,11,12)

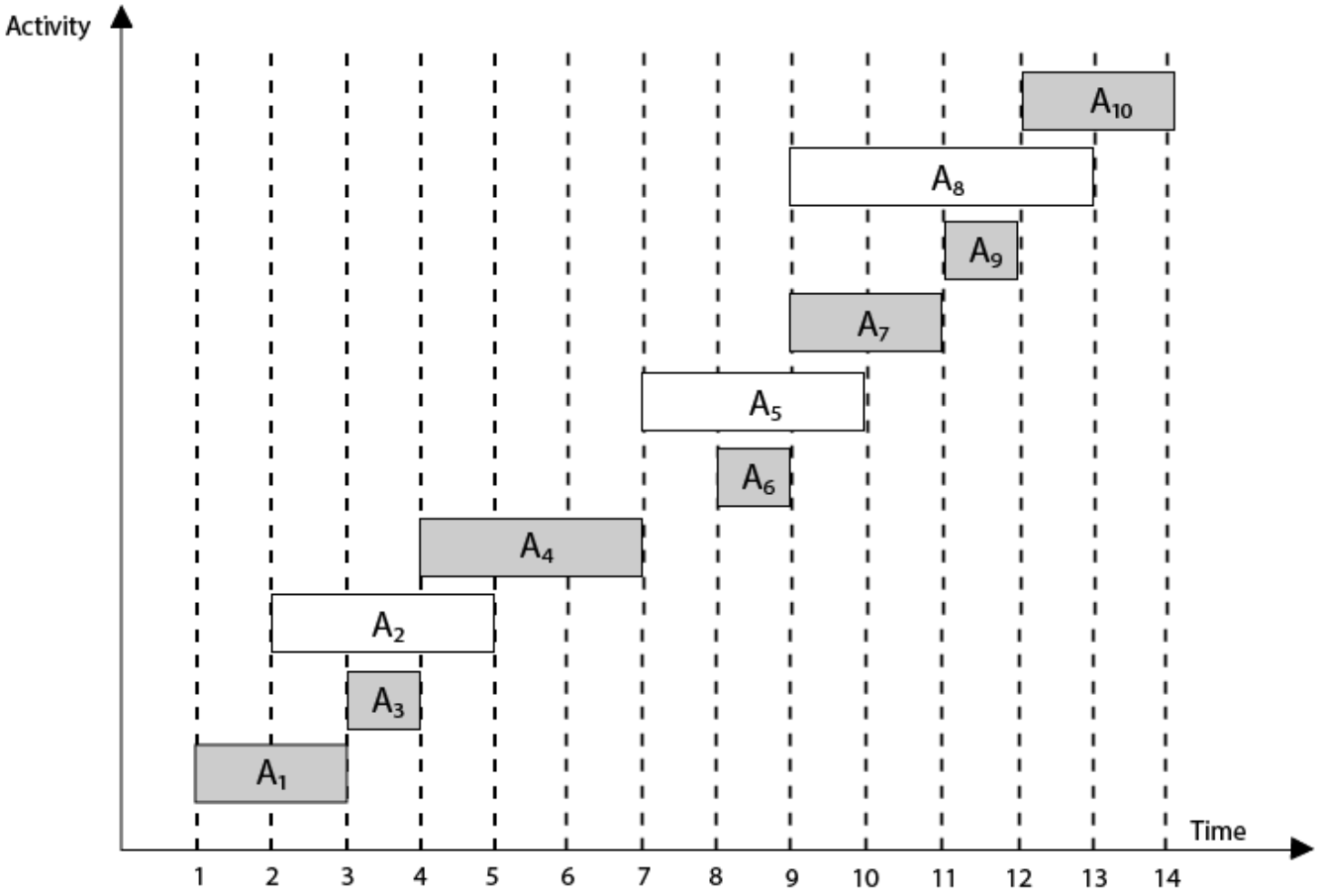
fi = (3,5,4,7,10,9,11,13,12,14)

Compute a schedule where the greatest number of activities takes place.

Solution: The solution to the above Activity scheduling problem using a greedy strategy is illustrated below:

Arranging the activities in increasing order of end time

Activity	A_1	A_3	A_2	A_4	A_6	A_5	A_7	A_9	A_8	A_{10}
Start	1	3	2	4	8	7	9	11	9	12
Finish	3	4	5	7	9	10	11	12	13	14



Now, schedule A_1

Next schedule A_3 as A_1 and A_3 are non-interfering.

Next **skip** A_2 as it is interfering.

Next, schedule A_4 as A_1 A_3 and A_4 are non-interfering, then next, schedule A_6 as A_1 A_3 A_4 and A_6 are non-interfering.

Skip A_5 as it is interfering.

Next, schedule A_7 as A_1 A_3 A_4 A_6 and A_7 are non-interfering.

Next, schedule A_9 as A_1 A_3 A_4 A_6 A_7 and A_9 are non-interfering.

Skip A_8 as it is interfering.

Next, schedule A_{10} as A_1 A_3 A_4 A_6 A_7 A_9 and A_{10} are non-interfering.

Thus the final Activity schedule is:

$$(A_1\ A_3\ A_4\ A_6\ A_7\ A_9\ A_{10})$$

Huffman Codes

- (i) Data can be encoded efficiently using Huffman Codes.
- (ii) It is a widely used and beneficial technique for compressing data.
- (iii) Huffman's greedy algorithm uses a table of the frequencies of occurrences of each character to build up an optimal way of representing each character as a binary string.

Suppose we have 10^5 characters in a data file. Normal Storage: 8 bits per character (ASCII) - 8 x 10^5 bits in a file. But we want to compress the file and save it compactly. Suppose only six characters appear in the file:

	a	b	c	d	e	f	Total
Frequency	45	13	12	16	9	5	100

How can we represent the data in a Compact way?

(i) Fixed length Code: Each letter represented by an equal number of bits. With a fixed length code, at least 3 bits per character:

For example:

```
a                   000

b                   001

c                   010

d                   011

e                   100

f                   101
```

For a file with 10^5 characters, we need 3 x 10^5 bits.

(ii) A variable-length code: It can do considerably better than a fixed-length code, by giving many characters short code words and infrequent character long codewords.

For example:

```
a       0

b      101

c      100

d      111

e      1101

f      1100
```

```
Number of bits = (45 x 1 + 13 x 3 + 12 x 3 + 16 x 3 + 9 x 4 + 5 x 4) x 1000
```

= 2.24 x 10^5bits

Thus, 224,000 bits to represent the file, a saving of approximately 25%.This is an optimal character code for this file.

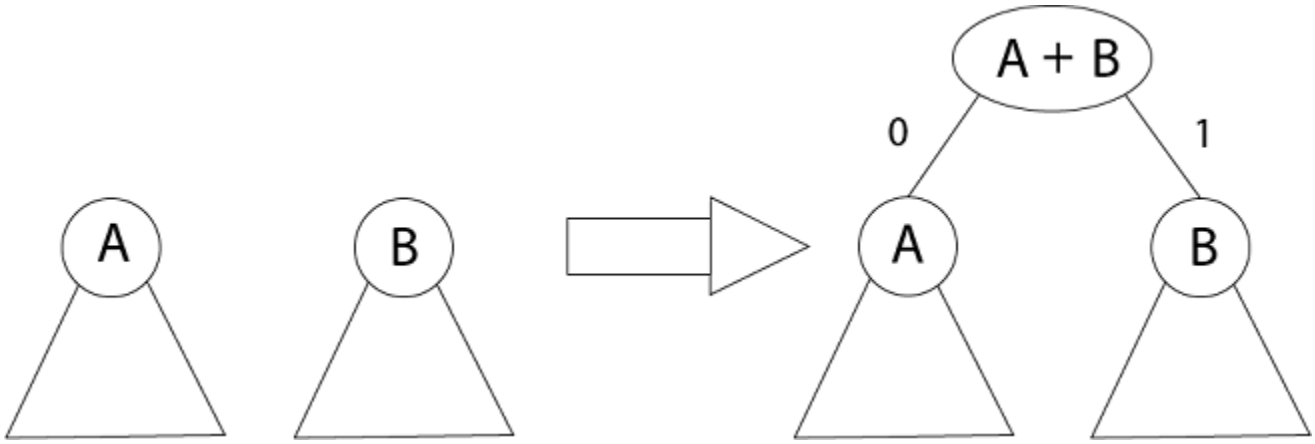
Prefix Codes:

The prefixes of an encoding of one character must not be equal to complete encoding of another character, e.g., 1100 and 11001 are not valid codes because 1100 is a prefix of some other code word is called prefix codes.

Prefix codes are desirable because they clarify encoding and decoding. Encoding is always simple for any binary character code; we concatenate the code words describing each character of the file. Decoding is also quite comfortable with a prefix code. Since no codeword is a prefix of any other, the codeword that starts with an encoded data is unambiguous.

Greedy Algorithm for constructing a Huffman Code:

Huffman invented a greedy algorithm that creates an optimal prefix code called a Huffman Code.



The algorithm builds the tree T analogous to the optimal code in a bottom-up manner. It starts with a set of |C| leaves (C is the number of characters) and performs |C| - 1 'merging' operations to create the final tree. In the Huffman algorithm 'n' denotes the quantity of a set of characters, z indicates the parent node, and x & y are the left & right child of z respectively.

Algorithm of Huffman Code

Huffman (C)

```
1. n=|C|
2. Q ← C
3. for i=1 to n-1
4. do
5. z= allocate-Node ()
6. x=  left[z]=Extract-Min(Q)
7. y= right[z] =Extract-Min(Q)
8. f [z]=f[x]+f[y]
9. Insert (Q, z)
10. return Extract-Min (Q)
```

Example: Find an optimal Huffman Code for the following set of frequencies:

1. a: 50 b: 25 c: 15 d: 40 e: 75

Solution:

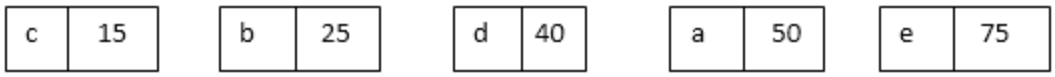
Given that: C = {a, b, c, d, e}

f (C) = {50, 25, 15, 40, 75}

n = 5

Q ← c

i.e.

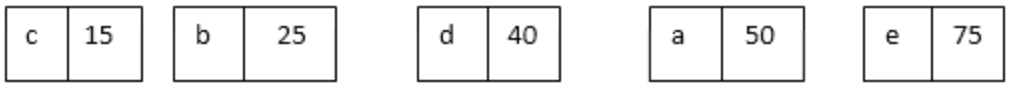


for i ← 1 to 4

i = 1 Z ← Allocate node

x ← Extract-Min (Q)

y ← Extract-Min (Q)

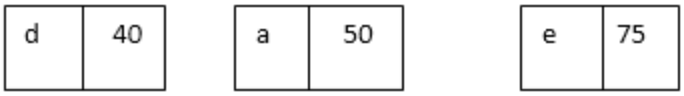


Left [z] ← x

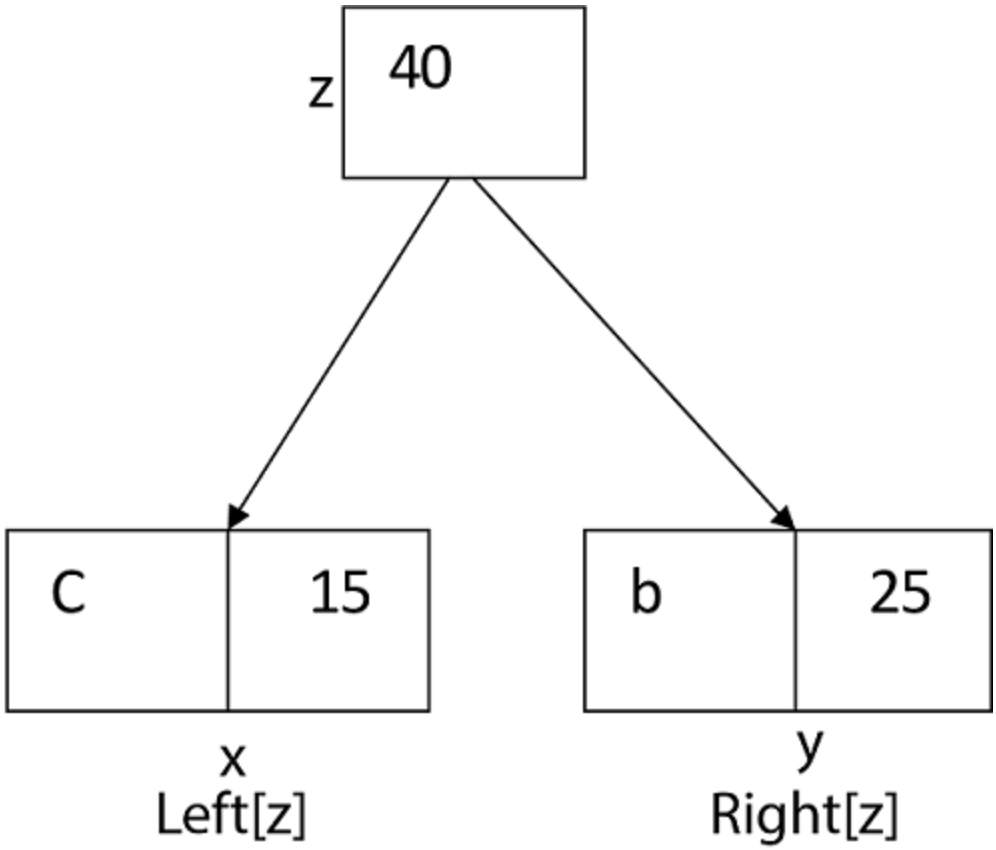
Right [z] ← y

f (z) ← f (x) + f (y) = 15 + 25

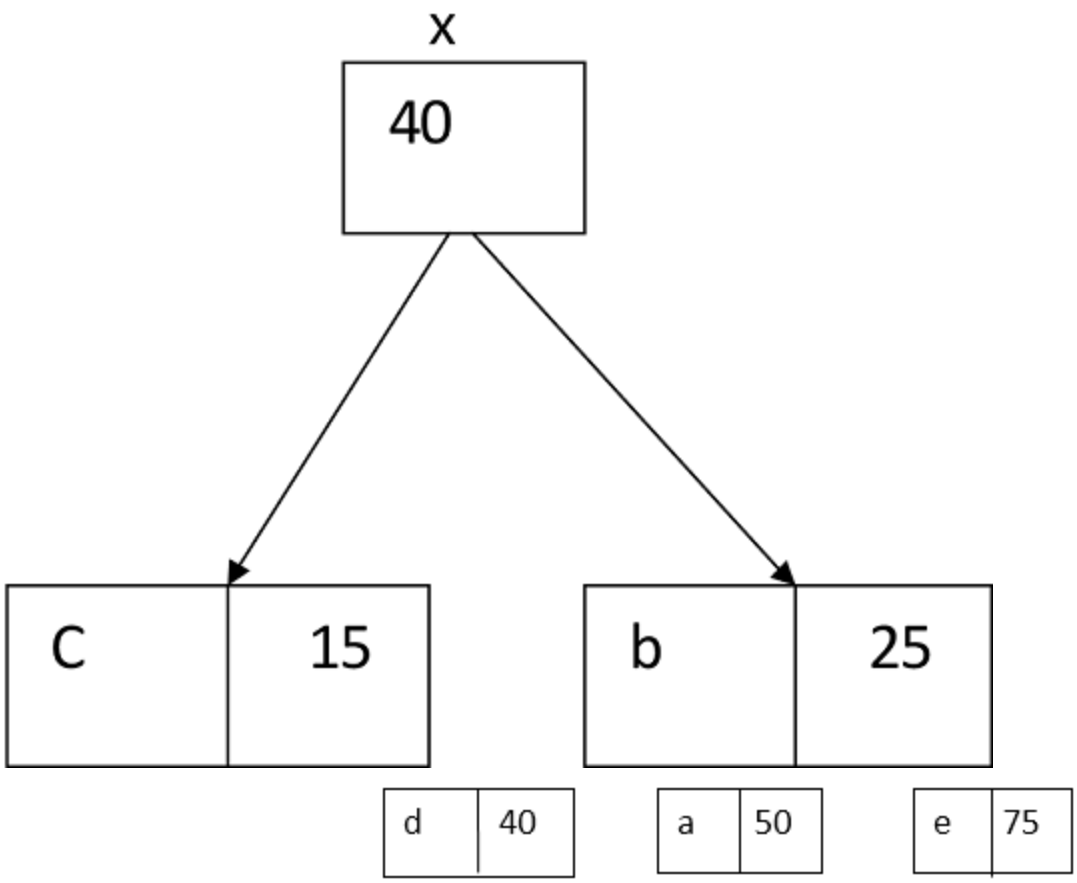
f (z) = 40



i.e.



Again for i=2



z ← Allocate node

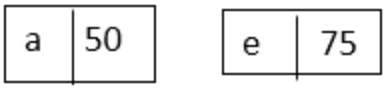
x ← 40

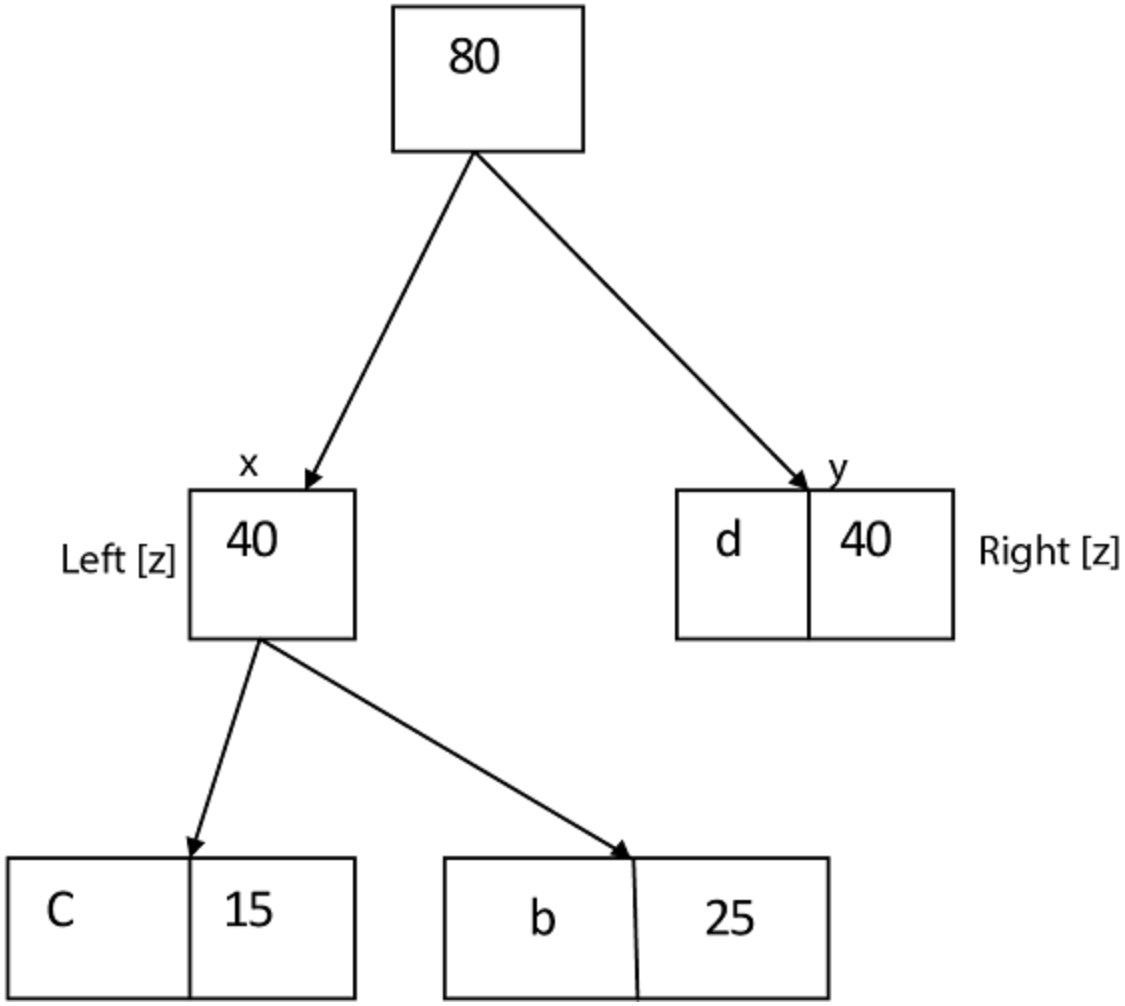
y ← 40

left [z] ← x

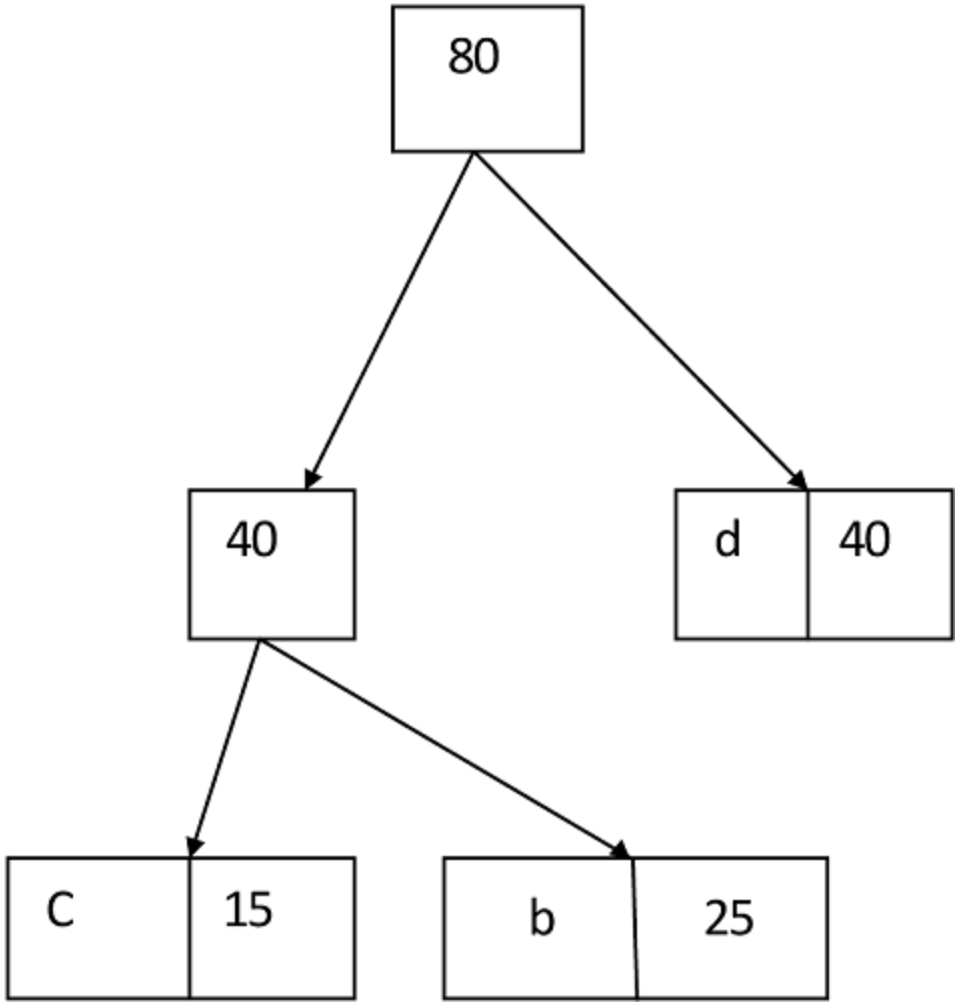
right [z] ← y

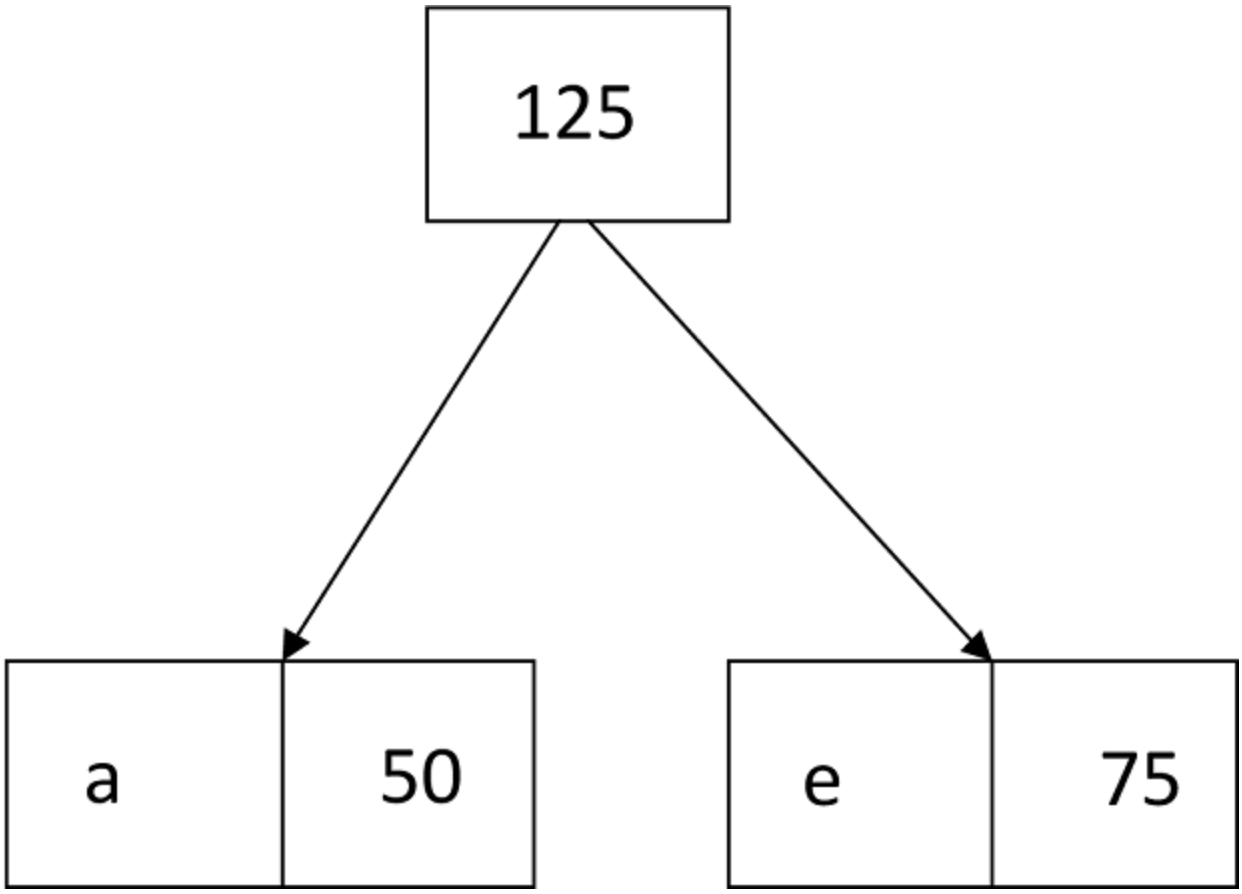
f (z) = 40 + 40 = 80



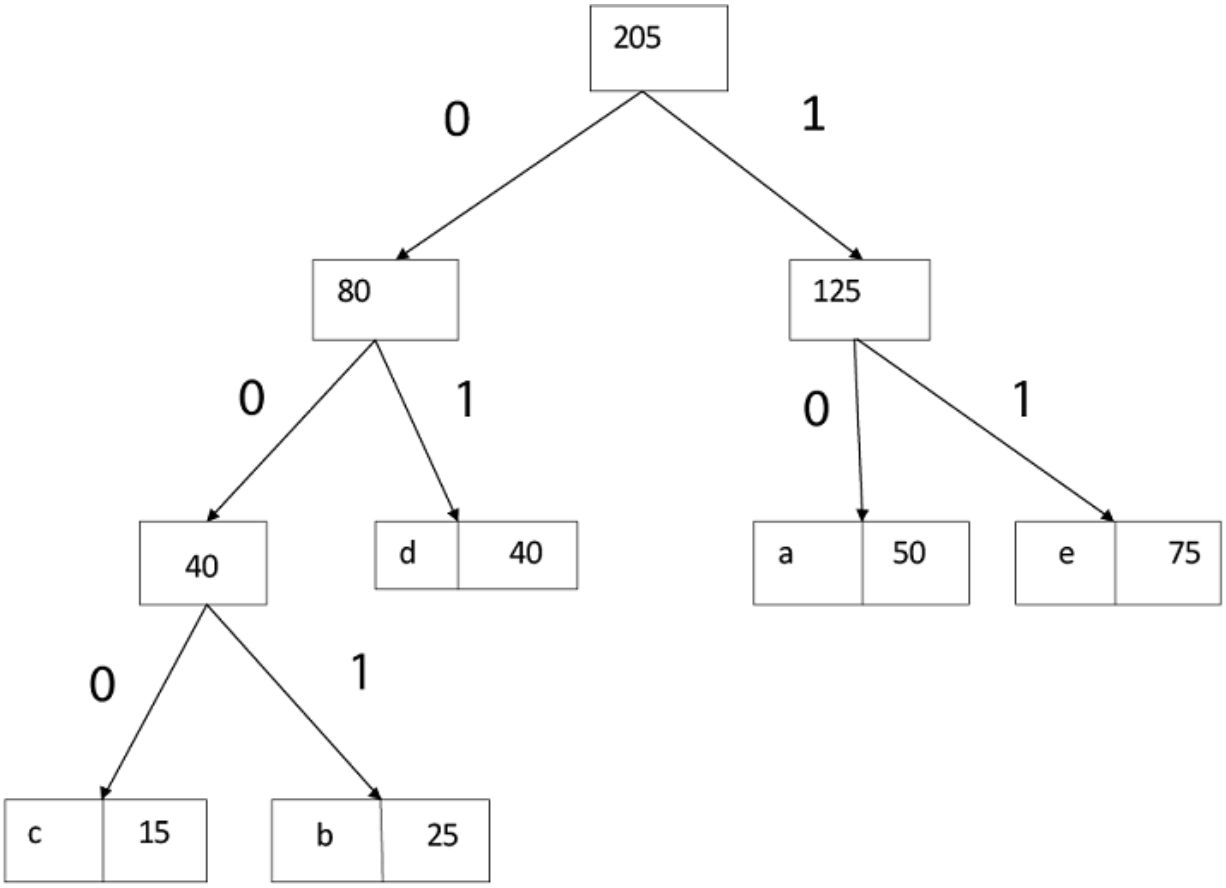


Similarly, we apply the same process we get


80
40
d
40
C
15
b
25



Thus, the final output is:


205
0
1
80
125
0
1
0
1
40
d
40
a
50
e
75
0
1
c
15
b
25

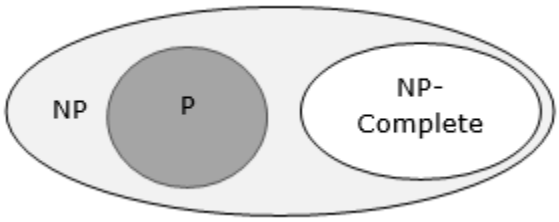
UNIT V

NP COMPLETE AND NP HARD

NP-Completeness: Polynomial Time – Polynomial-Time Verification – NP-Completeness and Reducibility – NP-Completeness Proofs – NP-Complete Problems.

NP-COMPLETENESS

A problem is in the class NPC if it is in NP and is as **hard** as any problem in NP. A problem is **NP-hard** if all problems in NP are polynomial time reducible to it, even though it may not be in NP itself.



If a polynomial time algorithm exists for any of these problems, all problems in NP would be polynomial time solvable. These problems are called **NP-complete**. The phenomenon of NP-completeness is important for both theoretical and practical reasons.

Definition of NP-Completeness

A language **B** is ***NP-complete*** if it satisfies two conditions

- **B** is in NP
- Every **A** in NP is polynomial time reducible to **B**.

If a language satisfies the second property, but not necessarily the first one, the language **B** is known as **NP-Hard**. Informally, a search problem **B** is **NP-Hard** if there exists some **NP-Complete** problem **A** that Turing reduces to **B**.

The problem in NP-Hard cannot be solved in polynomial time, until **P = NP**. If a problem is proved to be NPC, there is no need to waste time on trying to find an efficient algorithm for it. Instead, we can focus on design approximation algorithm.

POLYNOMIAL TIME:

NP class contains P class as a subset. NP problems being hard to solve.

Definition of P class Problem: - The set of decision-based problems come into the division of P Problems who can be solved or produced an output within polynomial time. P problems being easy to solve

Definition of Polynomial time: - If we produce an output according to the given input within a specific amount of time such as within a minute, hours. This is known as Polynomial time.

Definition of Non-Polynomial time: - If we produce an output according to the given input but there are no time constraints is known as Non-Polynomial time. But yes output will produce but time is not fixed yet.

Definition of Decision Based Problem: - A problem is called a decision problem if its output is a simple "yes" or "no" (or you may need this of this as true/false, 0/1, accept/reject.) We will phrase many optimization problems as decision problems. For example, Greedy method, D.P., given a graph G= (V, E) if there exists any Hamiltonian cycle.

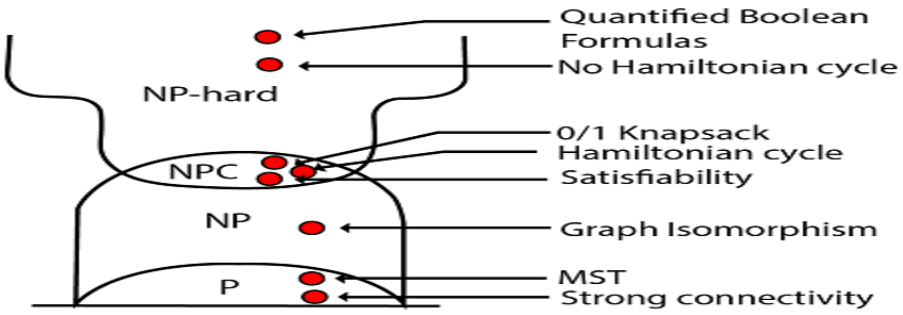
Definition of NP-hard class: - Here you to satisfy the following points to come into the division of NP-hard

1. If we can solve this problem in polynomial time, then we can solve all NP problems in polynomial time
2. If you convert the issue into one form to another form within the polynomial time

Definition of NP-complete class: - A problem is in NP-complete, if

1. It is in NP
2. It is NP-hard

Pictorial representation of all NP classes which includes NP, NP-hard, and NP-complete



NP-Complete Problems

Following are some NP-Complete problems, for which no polynomial time algorithm is known.

- Determining whether a graph has a Hamiltonian cycle
- Determining whether a Boolean formula is satisfiable, etc.

NP-Hard Problems

The following problems are NP-Hard

- The circuit-satisfiability problem
- Set Cover
- Vertex Cover
- Travelling Salesman Problem

In this context, now we will discuss TSP is NP-Complete

TSP is NP-Complete

The traveling salesman problem consists of a salesman and a set of cities. The salesman has to visit each one of the cities starting from a certain one and returning to the same city. The challenge of the problem is that the traveling salesman wants to minimize the total length of the trip

POLYNOMIAL TIME VERIFICATION

Before talking about the class of NP-complete problems, it is essential to introduce the notion of a verification algorithm.

Many problems are hard to solve, but they have the property that it easy to authenticate the solution if one is provided.

Hamiltonian cycle problem:-

Consider the Hamiltonian cycle problem. Given an undirected graph G, does G have a cycle that visits each vertex exactly once? There is no known polynomial time algorithm for this dispute.
Note: - It means you can't build a Hamiltonian cycle in a graph with a polynomial time even if there is no specific path is given for the Hamiltonian cycle with the particular vertex, yet you can't verify the Hamiltonian cycle within the polynomial time

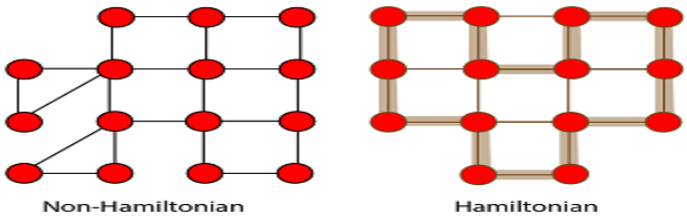


Fig: Hamiltonian Cycle

Let us understand that a graph did have a Hamiltonian cycle. It would be easy for someone to convince of this. They would similarly say: "the period is hv3, v7, v1....v13i.

We could then inspect the graph and check that this is indeed a legal cycle and that it visits all of the vertices of the graph exactly once. Thus, even though we know of no efficient way to solve the Hamiltonian cycle problem, there is a beneficial way to verify that a given cycle is indeed a Hamiltonian cycle.

Note:-For the verification in the Polynomial-time of an undirected Hamiltonian cycle graph G. There must be exact/specific/definite path must be given of Hamiltonian cycle then you can verify in the polynomial time.

Definition of Certificate: - A piece of information which contains in the given path of a vertex is known as certificate

Relation of P and NP classes

1. P contains in NP
2. P=NP

1. Observe that P contains in NP. In other words, if we can solve a problem in polynomial time, we can indeed verify the solution in polynomial time. More formally, we do not need to see a certificate (there is no need to specify the vertex/intermediate of the specific path) to solve the problem; we can explain it in polynomial time anyway.
2. However, it is not known whether P = NP. It seems you can verify and produce an output of the set of decision-based problems in NP classes in a polynomial time which is impossible because according to the definition of NP classes you can verify the solution within the polynomial time. So this relation can never be held.

NP-COMPLETENESS AND REDUCIBILITY

A decision problem L is NP-Hard if

L' ≤p L for all L' ϵ NP.

Definition: L is NP-complete if

1. L ϵ NP and

2. L' ≤ p L for some known NP-complete problem L.' Given this formal definition, the complexity classes are:

P: is the set of decision problems that are solvable in polynomial time.

NP: is the set of decision problems that can be verified in polynomial time.

NP-Hard: L is NP-hard if for all L' ϵ NP, L' ≤p L. Thus if we can solve L in polynomial time, we can solve all NP problems in polynomial time.

NP-Complete L is NP-complete if

1. L ϵ NP and
2. L is NP-hard

If any NP-complete problem is solvable in polynomial time, then every NP-Complete problem is also solvable in polynomial time. Conversely, if we can prove that any NP-Complete problem cannot be solved in polynomial time, every NP-Complete problem cannot be solvable in polynomial time.

Reductions:

The class NP-complete (NPC) problems consist of a set of decision problems (a subset of class NP) that no one knows how to solve efficiently. But if there were a polynomial solution for even a single NP-complete problem, then every problem in NPC will be solvable in polynomial time. For this, we need the concept of reductions.

Suppose there are two problems, A and B. You know that it is impossible to solve problem A in polynomial time. You want to prove that B cannot be explained in polynomial time. We want to show that (A ∉ P) => (B ∉ P)

Consider an example to illustrate reduction: The following problem is well-known to be NPC:

3-color: Given a graph G, can each of its vertices be labeled with one of 3 different colors such that two adjacent vertices do not have the same label (color).

Coloring arises in various partitioning issues where there is a constraint that two objects cannot be assigned to the same set of partitions. The phrase "coloring" comes from the original application which was in map drawing. Two countries that contribute a common border should be colored with different colors.

It is well known that planar graphs can be colored (maps) with four colors. There exists a polynomial time algorithm for this. But deciding whether this can be done with 3 colors is hard, and there is no polynomial time algorithm for it.

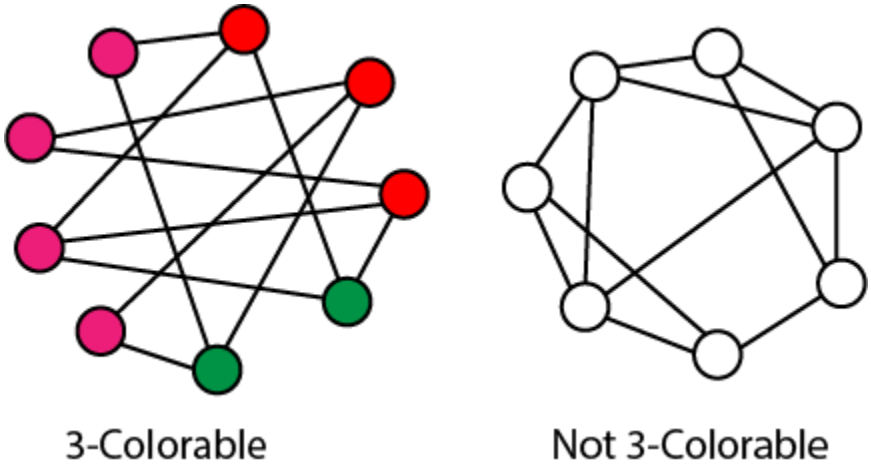


Fig: Example of 3-colorable and non-3-colorable graphs.

Polynomial Time Reduction:

We say that Decision Problem L_1 is Polynomial time Reducible to decision Problem L_2 ($L_1 \leq_p L_2$) if there is a polynomial time computation function f such that of all x, $x \epsilon L_1$ if and only if $x \epsilon L_2$

NP-COMPLETENESS PROOFS

To prove ***TSP is NP-Complete***, first we have to prove that ***TSP belongs to NP***. In TSP, we find a tour and check that the tour contains each vertex once. Then the total cost of the edges of the tour is calculated. Finally, we check if the cost is minimum. This can be completed in polynomial time. Thus ***TSP belongs to NP***.

Secondly, we have to prove that ***TSP is NP-hard***. To prove this, one way is to show that ***Hamiltonian cycle*** $\leq_p$ ***TSP*** (as we know that the Hamiltonian cycle problem is NPcomplete).

Assume ***G = (V, E)*** to be an instance of Hamiltonian cycle.

Hence, an instance of TSP is constructed. We create the complete graph $G' = (V, E')$, where

$E'=\{(i,j):i,j \in V and i \neq j E'=\{(i,j):i,j \in V and i \neq j$

Thus, the cost function is defined as follows −

$t(i,j)=\{01if(i,j) \in Eotherwise t(i,j)=\{0if(i,j) \in E1otherwise$

Now, suppose that a Hamiltonian cycle ***h*** exists in ***G***. It is clear that the cost of each edge in ***h*** is **0** in ***G***′ as each edge belongs to ***E***. Therefore, ***h*** has a cost of **0** in ***G***′. Thus, if graph ***G*** has a Hamiltonian cycle, then graph ***G***′ has a tour of **0** cost.

Conversely, we assume that ***G***′ has a tour ***h***′ of cost at most **0**. The cost of edges in ***E***′ are **0** and **1** by definition. Hence, each edge must have a cost of **0** as the cost of ***h***′ is **0**. We therefore conclude that ***h***′ contains only edges in ***E***.

We have thus proven that ***G*** has a Hamiltonian cycle, if and only if ***G***′ has a tour of cost at most **0**. TSP is NP-complete

NP-COMPLETE PROBLEMS

A problem is called NP (**nondeterministic polynomial**) if its solution can be guessed and verified in polynomial time; nondeterministic means that no particular rule is followed to make the guess. If a problem is NP and all other NP problems are polynomial-time reducible to it, the problem is NP-complete.

NP problem: - Suppose a DECISION-BASED problem is provided in which a set of inputs/high inputs you can get high output.

Criteria to come either in NP-hard or NP-complete.

1. The point to be noted here, the output is already given, and you can verify the output/solution within the polynomial time but can't produce an output/solution in polynomial time.
2. Here we need the concept of reduction because when you can't produce an output of the problem according to the given input then in case you have to use an emphasis on the concept of reduction in which you can convert one problem into another problem.

Note1:- If you satisfy both points then your problem comes into the category of NP-complete class

Note2:- If you satisfy the only 2nd points then your problem comes into the category of NP-hard class

So according to the given decision-based NP problem, you can decide in the form of yes or no. If, yes then you have to do verify and convert into another problem via reduction concept. If you are being performed, both then decision-based NP problems are in NP compete.

Kelli A. Houston, "Object Oriented Analysis & Design with Applications, Third Edition, Pearson Education,2010

5. Roger S. Pressman, "Software Engineering: A Practitioner's Approach, Tata McGraw-Hill Education, 8th Edition, 2015

CO-PO Mapping

CO	POs					
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**
1	2	1	3	3	2	2
2	2	1	3	2	2	2
3	2	1	3	3	2	2
4	2	1	3	3	1	2
5	2	1	3	3	3	2
6	1	1	3	2	2	2
Avg	1.83	1	3	2.66	2	2

MC4103 PYTHON PROGRAMMING L T P C
3 0 0 3

COURSE OBJECTIVES:

- To develop Python programs with conditionals, loops and functions.
- To use Python data structures – lists, tuples, dictionaries.
- To do input/output with files in Python
- To use modules, packages and frameworks in python
- To define a class with attributes and methods in python

UNIT I BASICS OF PYTHON 9

Introduction to Python Programming – Python Interpreter and Interactive Mode– Variables and Identifiers – Arithmetic Operators – Values and Types – Statements. Operators – Boolean Values – Operator Precedence – Expression – Conditionals: If-Else Constructs – Loop Structures/Iterative Statements – While Loop – For Loop – Break Statement-Continue statement – Function Call and Returning Values – Parameter Passing – Local and Global Scope – Recursive Functions

UNIT II DATA TYPES IN PYTHON 9

Lists, Tuples, Sets, Strings, Dictionary, Modules: Module Loading and Execution – Packages – Making Your Own Module – The Python Standard Libraries.

UNIT III FILE HANDLING AND EXCEPTION HANDLING 8

Files: Introduction – File Path – Opening and Closing Files – Reading and Writing Files –File Position –Exception: Errors and Exceptions, Exception Handling, Multiple Exceptions

UNIT IV MODULES, PACKAGES AND FRAMEWORKS 10

Modules: Introduction – Module Loading and Execution – Packages – Making Your Own Module – The Python Libraries for data processing, data mining and visualization- NUMPY, Pandas, Matplotlib, Plotly-Frameworks- -Django, Flask, Web2Py

UNIT V OBJECT ORIENTED PROGRAMMING IN PYTHON 9

Creating a Class, Class methods, Class Inheritance, Encapsulation, Polymorphism, class method vs. static methods, Python object persistence.

SUGGESTED ACTIVITIES:

1. Display a multiplication Table Both players are given the same string, S ; Both players have to make substrings using the letters of the string S.
2. Player A has to make words starting with consonants. Player B has to make words starting with vowels. The game ends when both players have made all possible substrings. Do Scoring
3. Write a function definition for JTOI() in Python that would display the corrected version of entire content of the file .TXT (has wrongly alphabet J in place of alphabet I) with all the alphabets "J" to be displayed as an alphabet "I" on screen.
4. Consider a CSV file of profit of 10 items in monthly sales of a year . Read this file using Pandas or NumPy or using the in-built matplotlib function. Perform the following task.

5. Read Total profit of all months and show it using a line plot
   Read all product sales data and show it using a multi-line plot
   Read each item sales data of each month and show it using a scatter plot
   Read each item product sales data and show it using the bar chart
   Read sales data of bathing soap of all months and show it using a bar chart.
   Calculate total sale data an year for each product and show it using a Pie chart
6. Create a Python class called Bank Account which represents a bank account, having as attributes: account Number (numeric type), name (name of the account owner as string type), balance. Create a constructor with parameters: account Number, name, balance. Create a Deposit() method which manages the deposit actions. Create a Withdrawal() method which manages withdrawals actions

COURSE OUTCOMES:

On completion of the course the student would be able to :

CO1: Develop algorithmic solutions to simple computational problems
CO2: Represent compound data using Python lists, tuples and dictionaries.
CO3: Read and write data from/to files in Python Programs
CO4: Structure simple Python programs using libraries, modules etc.
CO5: Structure a program by bundling related properties and behaviors into individual objects.

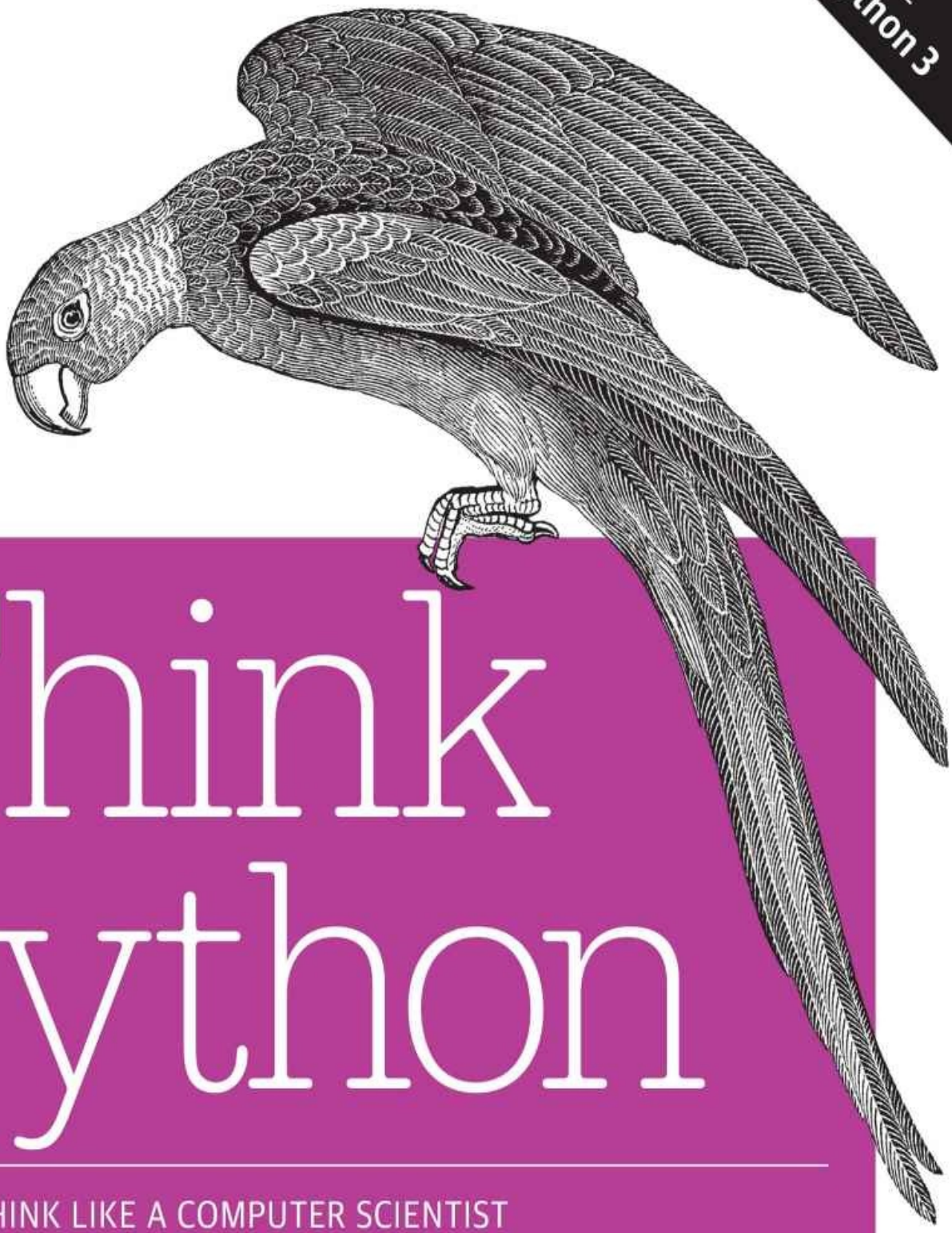
TOTAL : 45 PERIODS

REFERENCES

1. Reema Thareja, “Python Programming using Problem Solving Approach”, Oxford University Press, First edition, 2017
2. Allen B. Downey, “Think Python: How to Think Like a Computer Scientist”, Second Edition, Shroff, O‘Reilly Publishers, 2016 (http://greenteapress.com/wp/thinkpython/
3. Guido van Rossum, Fred L. Drake Jr., “An Introduction to Python – Revised and Updated for Python 3.2, Network Theory Ltd., First edition, 2011
4. John V Guttag, “Introduction to Computation and Programming Using Python”, Revised and Expanded Edition, MIT Press, 2013
5. Charles Dierbach, “Introduction to Computer Science using Python”, Wiley India Edition, First Edition, 2016

O'REILLY®

2nd Edition
Updated for Python 3



Think Python

HOW TO THINK LIKE A COMPUTER SCIENTIST

Allen B. Downey

Think Python

Second Edition

Allen B. Downey

Think Python

by Allen B. Downey

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (http://safaribooksonline.com). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

- Editor: Meghan Blanchette
- Production Editor: Kristen Brown
- Copyeditor: Nan Reinhardt
- Proofreader: Amanda Kersey
- Indexer: Allen Downey
- Interior Designer: David Futato
- Cover Designer: Karen Montgomery
- Illustrator: Rebecca Demarest
- August 2012: First Edition
- December 2015: Second Edition

Revision History for the Second Edition

- 2015-11-20: First Release

See http://oreilly.com/catalog/errata.csp?isbn=9781491939369 for release details.

978-1-491-93936-9

[LSI]

Preface

The Strange History of This Book

In January 1999 I was preparing to teach an introductory programming class in Java. I had taught it three times and I was getting frustrated. The failure rate in the class was too high and, even for students who succeeded, the overall level of achievement was too low.

One of the problems I saw was the books. They were too big, with too much unnecessary detail about Java, and not enough high-level guidance about how to program. And they all suffered from the trapdoor effect: they would start out easy, proceed gradually, and then somewhere around Chapter 5 the bottom would fall out. The students would get too much new material, too fast, and I would spend the rest of the semester picking up the pieces.

Two weeks before the first day of classes, I decided to write my own book. My goals were:

- Keep it short. It is better for students to read 10 pages than not read 50 pages.
- Be careful with vocabulary. I tried to minimize jargon and define each term at first use.
- Build gradually. To avoid trapdoors, I took the most difficult topics and split them into a series of small steps.
- Focus on programming, not the programming language. I included the minimum useful subset of Java and left out the rest.

I needed a title, so on a whim I chose *How to Think Like a Computer Scientist*.

My first version was rough, but it worked. Students did the reading, and they understood enough that I could spend class time on the hard topics, the interesting topics and (most important) letting the students practice.

I released the book under the GNU Free Documentation License, which allows users to copy, modify, and distribute the book.

What happened next is the cool part. Jeff Elkner, a high school teacher in Virginia, adopted my book and translated it into Python. He sent me a copy of his translation, and I had the unusual experience of learning Python by reading my own book. As Green Tea Press, I published the first Python version in 2001.

In 2003 I started teaching at Olin College and I got to teach Python for the first time. The contrast with Java was striking. Students struggled less, learned more, worked on more interesting projects, and generally had a lot more fun.

Since then I've continued to develop the book, correcting errors, improving some of the examples and adding material, especially exercises.

The result is this book, now with the less grandiose title *Think Python*. Some of the changes are:

- I added a section about debugging at the end of each chapter. These sections present

general techniques for finding and avoiding bugs, and warnings about Python pitfalls.

- I added more exercises, ranging from short tests of understanding to a few substantial projects. Most exercises include a link to my solution.
- I added a series of case studies — longer examples with exercises, solutions, and discussion.
- I expanded the discussion of program development plans and basic design patterns.
- I added appendices about debugging and analysis of algorithms.

The second edition of *Think Python* has these new features:

- The book and all supporting code have been updated to Python 3.
- I added a few sections, and more details on the Web, to help beginners get started running Python in a browser, so you don't have to deal with installing Python until you want to.
- For "The turtle Module" I switched from my own turtle graphics package, called Swampy, to a more standard Python module, `turtle`, which is easier to install and more powerful.
- I added a new chapter called "The Goodies", which introduces some additional Python features that are not strictly necessary, but sometimes handy.

I hope you enjoy working with this book, and that it helps you learn to program and think like a computer scientist, at least a little bit.

— Allen B. Downey

Olin College

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic

Indicates new terms, URLs, email addresses, filenames, and file extensions.

Bold

Indicates terms defined in the Glossary.

`Constant width`

Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`

Shows commands or other text that should be typed literally by the user.

`Constant width italic`

Shows text that should be replaced with user-supplied values or by values determined by context.

Using Code Examples

Supplemental material (code examples, exercises, etc.) is available for download at *http://www.greenteapress.com/thinkpython2/code*.

This book is here to help you get your job done. In general, if example code is offered with this book, you may use it in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Think Python*, 2nd Edition, by Allen B. Downey (O'Reilly). Copyright 2016 Allen Downey, 978-1-4919-3936-9."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

Safari® Books Online

Safari Books Online (*www.safaribooksonline.com*) is an on-demand digital library that delivers expert content in both book and video form from the world's leading authors in technology and business.

Technology professionals, software developers, web designers, and business and creative professionals use Safari Books Online as their primary resource for research, problem solving, learning, and certification training.

Safari Books Online offers a range of plans and pricing for enterprise, government, and education, and individuals.

Members have access to thousands of books, training videos, and prepublication manuscripts in one fully searchable database from publishers like O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, and hundreds more. For more information about Safari Books Online, please visit us online.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

- O’Reilly Media, Inc.
- 1005 Gravenstein Highway North
- Sebastopol, CA 95472
- 800-998-9938 (in the United States or Canada)
- 707-829-0515 (international or local)
- 707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *http://bit.ly/think-python_2E*.

To comment or ask technical questions about this book, send email to *bookquestions@oreilly.com*.

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgments

Many thanks to Jeff Elkner, who translated my Java book into Python, which got this project started and introduced me to what has turned out to be my favorite language.

Thanks also to Chris Meyers, who contributed several sections to *How to Think Like a Computer Scientist*.

Thanks to the Free Software Foundation for developing the GNU Free Documentation License, which helped make my collaboration with Jeff and Chris possible, and Creative Commons for the license I am using now.

Thanks to the editors at Lulu who worked on *How to Think Like a Computer Scientist*.

Thanks to the editors at O'Reilly Media who worked on *Think Python*.

Thanks to all the students who worked with earlier versions of this book and all the contributors (listed below) who sent in corrections and suggestions.

Contributor List

More than 100 sharp-eyed and thoughtful readers have sent in suggestions and corrections over the past few years. Their contributions, and enthusiasm for this project, have been a huge help.

If you have a suggestion or correction, please send email to *feedback@thinkpython.com*. If I make a change based on your feedback, I will add you to the contributor list (unless you ask to be omitted).

If you include at least part of the sentence the error appears in, that makes it easy for me to search. Page and section numbers are fine, too, but not quite as easy to work with. Thanks!

- Lloyd Hugh Allen sent in a correction to Section 8.4.
- Yvon Boulianne sent in a correction of a semantic error in Chapter 5.
- Fred Bremmer submitted a correction in Section 2.1.
- Jonah Cohen wrote the Perl scripts to convert the LaTeX source for this book into beautiful HTML.
- Michael Conlon sent in a grammar correction in Chapter 2 and an improvement in style in Chapter 1, and he initiated discussion on the technical aspects of interpreters.
- Benoit Girard sent in a correction to a humorous mistake in Section 5.6.
- Courtney Gleason and Katherine Smith wrote `horsebet.py`, which was used as a case study in an earlier version of the book. Their program can now be found on the website.
- Lee Harr submitted more corrections than we have room to list here, and indeed he should be listed as one of the principal editors of the text.
- James Kaylin is a student using the text. He has submitted numerous corrections.
- David Kershaw fixed the broken `catTwice` function in Section 3.10.
- Eddie Lam has sent in numerous corrections to Chapters 1, 2, and 3. He also fixed the Makefile so that it creates an index the first time it is run and helped us set up a versioning scheme.
- Man-Yong Lee sent in a correction to the example code in Section 2.4.
- David Mayo pointed out that the word “unconsciously” in Chapter 1 needed to be changed to “subconsciously”.
- Chris McAloon sent in several corrections to Sections 3.9 and 3.10.

- Matthew J. Moelter has been a long-time contributor who sent in numerous corrections and suggestions to the book.
- Simon Dicon Montford reported a missing function definition and several typos in Chapter 3. He also found errors in the `increment` function in Chapter 13.
- John Ouzts corrected the definition of “return value” in Chapter 3.
- Kevin Parks sent in valuable comments and suggestions as to how to improve the distribution of the book.
- David Pool sent in a typo in the glossary of Chapter 1, as well as kind words of encouragement.
- Michael Schmitt sent in a correction to the chapter on files and exceptions.
- Robin Shaw pointed out an error in Section 13.1, where the `printTime` function was used in an example without being defined.
- Paul Sleigh found an error in Chapter 7 and a bug in Jonah Cohen’s Perl script that generates HTML from LaTeX.
- Craig T. Snydal is testing the text in a course at Drew University. He has contributed several valuable suggestions and corrections.
- Ian Thomas and his students are using the text in a programming course. They are the first ones to test the chapters in the latter half of the book, and they have made numerous corrections and suggestions.
- Keith Verheyden sent in a correction in Chapter 3.
- Peter Winstanley let us know about a longstanding error in our Latin in Chapter 3.
- Chris Wrobel made corrections to the code in the chapter on file I/O and exceptions.
- Moshe Zadka has made invaluable contributions to this project. In addition to writing the first draft of the chapter on Dictionaries, he provided continual guidance in the early stages of the book.
- Christoph Zwerschke sent several corrections and pedagogic suggestions, and explained the difference between *gleich* and *selbe*.
- James Mayer sent us a whole slew of spelling and typographical errors, including two in the contributor list.
- Hayden McAfee caught a potentially confusing inconsistency between two examples.
- Angel Arnal is part of an international team of translators working on the Spanish

version of the text. He has also found several errors in the English version.

- Tauhidul Hoque and Lex Berezhny created the illustrations in Chapter 1 and improved many of the other illustrations.
- Dr. Michele Alzetta caught an error in Chapter 8 and sent some interesting pedagogic comments and suggestions about Fibonacci and Old Maid.
- Andy Mitchell caught a typo in Chapter 1 and a broken example in Chapter 2.
- Kalin Harvey suggested a clarification in Chapter 7 and caught some typos.
- Christopher P. Smith caught several typos and helped us update the book for Python 2.2.
- David Hutchins caught a typo in the Foreword.
- Gregor Lingl is teaching Python at a high school in Vienna, Austria. He is working on a German translation of the book, and he caught a couple of bad errors in Chapter 5.
- Julie Peters caught a typo in the Preface.
- Florin Oprina sent in an improvement in `makeTime`, a correction in `printTime`, and a nice typo.
- D. J. Webre suggested a clarification in Chapter 3.
- Ken found a fistful of errors in Chapters 8, 9 and 11.
- Ivo Wever caught a typo in Chapter 5 and suggested a clarification in Chapter 3.
- Curtis Yanko suggested a clarification in Chapter 2.
- Ben Logan sent in a number of typos and problems with translating the book into HTML.
- Jason Armstrong saw the missing word in Chapter 2.
- Louis Cordier noticed a spot in Chapter 16 where the code didn’t match the text.
- Brian Cain suggested several clarifications in Chapters 2 and 3.
- Rob Black sent in a passel of corrections, including some changes for Python 2.2.
- Jean-Philippe Rey at Ecole Centrale Paris sent a number of patches, including some updates for Python 2.2 and other thoughtful improvements.
- Jason Mader at George Washington University made a number of useful suggestions and corrections.

- Jan Gundtofte-Bruun reminded us that “a error” is an error.
- Abel David and Alexis Dinno reminded us that the plural of “matrix” is “matrices”, not “matrixes”. This error was in the book for years, but two readers with the same initials reported it on the same day. Weird.
- Charles Thayer encouraged us to get rid of the semicolons we had put at the ends of some statements and to clean up our use of “argument” and “parameter”.
- Roger Sperberg pointed out a twisted piece of logic in Chapter 3.
- Sam Bull pointed out a confusing paragraph in Chapter 2.
- Andrew Cheung pointed out two instances of “use before def”.
- C. Corey Capel spotted a missing word and a typo in Chapter 4.
- Alessandra helped clear up some Turtle confusion.
- Wim Champagne found a braino in a dictionary example.
- Douglas Wright pointed out a problem with floor division in `arc`.
- Jared Spindor found some jetsam at the end of a sentence.
- Lin Peiheng sent a number of very helpful suggestions.
- Ray Hagtvedt sent in two errors and a not-quite-error.
- Torsten Hübsch pointed out an inconsistency in Swampy.
- Inga Petuhhov corrected an example in Chapter 14.
- Arne Babenhauserheide sent several helpful corrections.
- Mark E. Casida is is good at spotting repeated words.
- Scott Tyler filled in a that was missing. And then sent in a heap of corrections.
- Gordon Shephard sent in several corrections, all in separate emails.
- Andrew Turner `spotted` an error in Chapter 8.
- Adam Hobart fixed a problem with floor division in `arc`.
- Daryl Hammond and Sarah Zimmerman pointed out that I served up `math.pi` too early. And Zim spotted a typo.
- George Sass found a bug in a Debugging section.

- Brian Bingham suggested Exercise 11-5.
- Leah Engelbert-Fenton pointed out that I used `tuple` as a variable name, contrary to my own advice. And then found a bunch of typos and a “use before def”.
- Joe Funke spotted a typo.
- Chao-chao Chen found an inconsistency in the Fibonacci example.
- Jeff Paine knows the difference between space and spam.
- Lubos Pintes sent in a typo.
- Gregg Lind and Abigail Heithoff suggested Exercise 14-3.
- Max Hailperin has sent in a number of corrections and suggestions. Max is one of the authors of the extraordinary *Concrete Abstractions* (Course Technology, 1998), which you might want to read when you are done with this book.
- Chotipat Pornavalai found an error in an error message.
- Stanislaw Antol sent a list of very helpful suggestions.
- Eric Pashman sent a number of corrections for Chapters 4–11.
- Miguel Azevedo found some typos.
- Jianhua Liu sent in a long list of corrections.
- Nick King found a missing word.
- Martin Zuther sent a long list of suggestions.
- Adam Zimmerman found an inconsistency in my instance of an “instance” and several other errors.
- Ratnakar Tiwari suggested a footnote explaining degenerate triangles.
- Anurag Goel suggested another solution for `is_abecedarian` and sent some additional corrections. And he knows how to spell Jane Austen.
- Kelli Kratzer spotted one of the typos.
- Mark Griffiths pointed out a confusing example in Chapter 3.
- Roydan Ongie found an error in my Newton’s method.
- Patryk Wolowiec helped me with a problem in the HTML version.

- Mark Chonofsky told me about a new keyword in Python 3.
- Russell Coleman helped me with my geometry.
- Wei Huang spotted several typographical errors.
- Karen Barber spotted the the oldest typo in the book.
- Nam Nguyen found a typo and pointed out that I used the Decorator pattern but didn't mention it by name.
- Stéphane Morin sent in several corrections and suggestions.
- Paul Stoop corrected a typo in `uses_only`.
- Eric Bronner pointed out a confusion in the discussion of the order of operations.
- Alexandros Gezerlis set a new standard for the number and quality of suggestions he submitted. We are deeply grateful!
- Gray Thomas knows his right from his left.
- Giovanni Escobar Sosa sent a long list of corrections and suggestions.
- Alix Etienne fixed one of the URLs.
- Kuang He found a typo.
- Daniel Neilson corrected an error about the order of operations.
- Will McGinnis pointed out that `polyline` was defined differently in two places.
- Swarup Sahoo spotted a missing semicolon.
- Frank Hecker pointed out an exercise that was under-specified, and some broken links.
- Animesh B helped me clean up a confusing example.
- Martin Caspersen found two round-off errors.
- Gregor Ulm sent several corrections and suggestions.
- Dimitrios Tsirigkas suggested I clarify an exercise.
- Carlos Tafur sent a page of corrections and suggestions.
- Martin Nordsletten found a bug in an exercise solution.
- Lars O.D. Christensen found a broken reference.

- Victor Simeone found a typo.
- Sven Hoexter pointed out that a variable named `input` shadows a build-in function.
- Viet Le found a typo.
- Stephen Gregory pointed out the problem with `cmp` in Python 3.
- Matthew Shultz let me know about a broken link.
- Lokesh Kumar Makani let me know about some broken links and some changes in error messages.
- Ishwar Bhat corrected my statement of Fermat's last theorem.
- Brian McGhie suggested a clarification.
- Andrea Zanella translated the book into Italian, and sent a number of corrections along the way.
- Many, many thanks to Melissa Lewis and Luciano Ramalho for excellent comments and suggestions on the second edition.
- Thanks to Harry Percival from PythonAnywhere for his help getting people started running Python in a browser.
- Xavier Van Aubel made several useful corrections in the second edition.

Chapter 1. The Way of the Program

The goal of this book is to teach you to think like a computer scientist. This way of thinking combines some of the best features of mathematics, engineering, and natural science. Like mathematicians, computer scientists use formal languages to denote ideas (specifically computations). Like engineers, they design things, assembling components into systems and evaluating tradeoffs among alternatives. Like scientists, they observe the behavior of complex systems, form hypotheses, and test predictions.

The single most important skill for a computer scientist is **problem solving**. Problem solving means the ability to formulate problems, think creatively about solutions, and express a solution clearly and accurately. As it turns out, the process of learning to program is an excellent opportunity to practice problem-solving skills. That's why this chapter is called "The Way of the Program".

On one level, you will be learning to program, a useful skill by itself. On another level, you will use programming as a means to an end. As we go along, that end will become clearer.

What Is a Program?

A **program** is a sequence of instructions that specifies how to perform a computation. The computation might be something mathematical, such as solving a system of equations or finding the roots of a polynomial, but it can also be a symbolic computation, such as searching and replacing text in a document or something graphical, like processing an image or playing a video.

The details look different in different languages, but a few basic instructions appear in just about every language:

input:

Get data from the keyboard, a file, the network, or some other device.

output:

Display data on the screen, save it in a file, send it over the network, etc.

math:

Perform basic mathematical operations like addition and multiplication.

conditional execution:

Check for certain conditions and run the appropriate code.

repetition:

Perform some action repeatedly, usually with some variation.

Believe it or not, that’s pretty much all there is to it. Every program you’ve ever used, no matter how complicated, is made up of instructions that look pretty much like these. So you can think of programming as the process of breaking a large, complex task into smaller and smaller subtasks until the subtasks are simple enough to be performed with one of these basic instructions.

Running Python

One of the challenges of getting started with Python is that you might have to install Python and related software on your computer. If you are familiar with your operating system, and especially if you are comfortable with the command-line interface, you will have no trouble installing Python. But for beginners, it can be painful to learn about system administration and programming at the same time.

To avoid that problem, I recommend that you start out running Python in a browser. Later, when you are comfortable with Python, I'll make suggestions for installing Python on your computer.

There are a number of web pages you can use to run Python. If you already have a favorite, go ahead and use it. Otherwise I recommend PythonAnywhere. I provide detailed instructions for getting started at *http://tinyurl.com/thinkpython2e*.

There are two versions of Python, called Python 2 and Python 3. They are very similar, so if you learn one, it is easy to switch to the other. In fact, there are only a few differences you will encounter as a beginner. This book is written for Python 3, but I include some notes about Python 2.

The Python **interpreter** is a program that reads and executes Python code. Depending on your environment, you might start the interpreter by clicking on an icon, or by typing `python` on a command line. When it starts, you should see output like this:

```
Python 3.4.0 (default, Jun 19 2015, 14:20:21)
[GCC 4.8.2] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

The first three lines contain information about the interpreter and the operating system it's running on, so it might be different for you. But you should check that the version number, which is `3.4.0` in this example, begins with 3, which indicates that you are running Python 3. If it begins with 2, you are running (you guessed it) Python 2.

The last line is a **prompt** that indicates that the interpreter is ready for you to enter code. If you type a line of code and hit Enter, the interpreter displays the result:

```
>>> 1 + 1
2
```

Now you're ready to get started. From here on, I assume that you know how to start the Python interpreter and run code.

The First Program

Traditionally, the first program you write in a new language is called “Hello, World!” because all it does is display the words “Hello, World!” In Python, it looks like this:

```
>>> print('Hello, World!')
```

This is an example of a **print statement**, although it doesn’t actually print anything on paper. It displays a result on the screen. In this case, the result is the words

```
Hello, World!
```

The quotation marks in the program mark the beginning and end of the text to be displayed; they don’t appear in the result.

The parentheses indicate that `print` is a function. We’ll get to functions in Chapter 3.

In Python 2, the print statement is slightly different; it is not a function, so it doesn’t use parentheses.

```
>>> print 'Hello, World!'
```

This distinction will make more sense soon, but that’s enough to get started.

Arithmetic Operators

After “Hello, World”, the next step is arithmetic. Python provides **operators**, which are special symbols that represent computations like addition and multiplication.

The operators `+`, `-`, and `*` perform addition, subtraction, and multiplication, as in the following examples:

```
>>> 40 + 2
42
>>> 43 - 1
42
>>> 6 * 7
42
```

The operator `/` performs division:

```
>>> 84 / 2
42.0
```

You might wonder why the result is `42.0` instead of `42`. I’ll explain in the next section.

Finally, the operator `**` performs exponentiation; that is, it raises a number to a power:

```
>>> 6**2 + 6
42
```

In some other languages, `^` is used for exponentiation, but in Python it is a bitwise operator called XOR. If you are not familiar with bitwise operators, the result will surprise you:

```
>>> 6 ^ 2
4
```

I won’t cover bitwise operators in this book, but you can read about them at *http://wiki.python.org/moin/BitwiseOperators*.

Values and Types

A **value** is one of the basic things a program works with, like a letter or a number. Some values we have seen so far are `2`, `42.0`, and `'Hello, World!'`

These values belong to different **types**: `2` is an **integer**, `42.0` is a **floating-point number**, and `'Hello, World!'` is a **string**, so-called because the letters it contains are strung together.

If you are not sure what type a value has, the interpreter can tell you:

```
>>> type(2)
<class 'int'>
>>> type(42.0)
<class 'float'>
>>> type('Hello, World!')
<class 'str'>
```

In these results, the word “class” is used in the sense of a category; a type is a category of values.

Not surprisingly, integers belong to the type `int`, strings belong to `str`, and floating-point numbers belong to `float`.

What about values like `'2'` and `'42.0'`? They look like numbers, but they are in quotation marks like strings:

```
>>> type('2')
<class 'str'>
>>> type('42.0')
<class 'str'>
```

They’re strings.

When you type a large integer, you might be tempted to use commas between groups of digits, as in `1,000,000`. This is not a legal *integer* in Python, but it is legal:

```
>>> 1,000,000
(1, 0, 0)
```

That’s not what we expected at all! Python interprets `1,000,000` as a comma-separated sequence of integers. We’ll learn more about this kind of sequence later.

Formal and Natural Languages

Natural languages are the languages people speak, such as English, Spanish, and French. They were not designed by people (although people try to impose some order on them); they evolved naturally.

Formal languages are languages that are designed by people for specific applications. For example, the notation that mathematicians use is a formal language that is particularly good at denoting relationships among numbers and symbols. Chemists use a formal language to represent the chemical structure of molecules. And most importantly:

> **Programming languages are formal languages that have been designed to express computations.**

Formal languages tend to have strict **syntax** rules that govern the structure of statements. For example, in mathematics the statement $3 + 3 = 6$ has correct syntax, but $3 + = 3\$6$ does not. In chemistry H_2O is a syntactically correct formula, but $_2Zz$ is not.

Syntax rules come in two flavors, pertaining to **tokens** and structure. Tokens are the basic elements of the language, such as words, numbers, and chemical elements. One of the problems with $3 + = 3\$6$ is that $\$$ is not a legal token in mathematics (at least as far as I know). Similarly, $_2Zz$ is not legal because there is no element with the abbreviation *Zz*.

The second type of syntax rule pertains to the way tokens are combined. The equation $3 + = 3$ is illegal because even though + and = are legal tokens, you can't have one right after the other. Similarly, in a chemical formula the subscript comes after the element name, not before.

This is @ well-structured Engli$h sentence with invalid t*kens in it. This sentence all valid tokens has, but invalid structure with.

When you read a sentence in English or a statement in a formal language, you have to figure out the structure (although in a natural language you do this subconsciously). This process is called **parsing**.

Although formal and natural languages have many features in common — tokens, structure, and syntax — there are some differences:

ambiguity:

> Natural languages are full of ambiguity, which people deal with by using contextual clues and other information. Formal languages are designed to be nearly or completely unambiguous, which means that any statement has exactly one meaning, regardless of context.

redundancy:

In order to make up for ambiguity and reduce misunderstandings, natural languages employ lots of redundancy. As a result, they are often verbose. Formal languages are less redundant and more concise.

literalness:

Natural languages are full of idiom and metaphor. If I say, “The penny dropped”, there is probably no penny and nothing dropping (this idiom means that someone understood something after a period of confusion). Formal languages mean exactly what they say.

Because we all grow up speaking natural languages, it is sometimes hard to adjust to formal languages. The difference between formal and natural language is like the difference between poetry and prose, but more so:

Poetry:

Words are used for their sounds as well as for their meaning, and the whole poem together creates an effect or emotional response. Ambiguity is not only common but often deliberate.

Prose:

The literal meaning of words is more important, and the structure contributes more meaning. Prose is more amenable to analysis than poetry but still often ambiguous.

Programs:

The meaning of a computer program is unambiguous and literal, and can be understood entirely by analysis of the tokens and structure.

Formal languages are more dense than natural languages, so it takes longer to read them. Also, the structure is important, so it is not always best to read from top to bottom, left to right. Instead, learn to parse the program in your head, identifying the tokens and interpreting the structure. Finally, the details matter. Small errors in spelling and punctuation, which you can get away with in natural languages, can make a big difference in a formal language.

Debugging

Programmers make mistakes. For whimsical reasons, programming errors are called **bugs** and the process of tracking them down is called **debugging**.

Programming, and especially debugging, sometimes brings out strong emotions. If you are struggling with a difficult bug, you might feel angry, despondent, or embarrassed.

There is evidence that people naturally respond to computers as if they were people. When they work well, we think of them as teammates, and when they are obstinate or rude, we respond to them the same way we respond to rude, obstinate people (Reeves and Nass, *The Media Equation: How People Treat Computers, Television, and New Media Like Real People and Places*).

Preparing for these reactions might help you deal with them. One approach is to think of the computer as an employee with certain strengths, like speed and precision, and particular weaknesses, like lack of empathy and inability to grasp the big picture.

Your job is to be a good manager: find ways to take advantage of the strengths and mitigate the weaknesses. And find ways to use your emotions to engage with the problem, without letting your reactions interfere with your ability to work effectively.

Learning to debug can be frustrating, but it is a valuable skill that is useful for many activities beyond programming. At the end of each chapter there is a section, like this one, with my suggestions for debugging. I hope they help!

Glossary

problem solving:

The process of formulating a problem, finding a solution, and expressing it.

high-level language:

A programming language like Python that is designed to be easy for humans to read and write.

low-level language:

A programming language that is designed to be easy for a computer to run; also called “machine language” or “assembly language”.

portability:

A property of a program that can run on more than one kind of computer.

interpreter:

A program that reads another program and executes it.

prompt:

Characters displayed by the interpreter to indicate that it is ready to take input from the user.

program:

A set of instructions that specifies a computation.

print statement:

An instruction that causes the Python interpreter to display a value on the screen.

operator:

A special symbol that represents a simple computation like addition, multiplication, or string concatenation.

value:

One of the basic units of data, like a number or string, that a program manipulates.

type:

A category of values. The types we have seen so far are integers (type `int`), floating-point numbers (type `float`), and strings (type `str`).

integer:

A type that represents whole numbers.

floating-point:

A type that represents numbers with fractional parts.

string:

A type that represents sequences of characters.

natural language:

Any one of the languages that people speak that evolved naturally.

formal language:

Any one of the languages that people have designed for specific purposes, such as representing mathematical ideas or computer programs; all programming languages are formal languages.

token:

One of the basic elements of the syntactic structure of a program, analogous to a word in a natural language.

syntax:

The rules that govern the structure of a program.

parse:

To examine a program and analyze the syntactic structure.

bug:

An error in a program.

debugging:

The process of finding and correcting bugs.

Exercises

Exercise 1-1.

It is a good idea to read this book in front of a computer so you can try out the examples as you go.

Whenever you are experimenting with a new feature, you should try to make mistakes. For example, in the "Hello, world!" program, what happens if you leave out one of the quotation marks? What if you leave out both? What if you spell `print` wrong?

This kind of experiment helps you remember what you read; it also helps when you are programming, because you get to know what the error messages mean. It is better to make mistakes now and on purpose than later and accidentally.

1. In a print statement, what happens if you leave out one of the parentheses, or both?

2. If you are trying to print a string, what happens if you leave out one of the quotation marks, or both?

3. You can use a minus sign to make a negative number like `-2`. What happens if you put a plus sign before a number? What about `2++2`?

4. In math notation, leading zeros are okay, as in `02`. What happens if you try this in Python?

5. What happens if you have two values with no operator between them?

Exercise 1-2.

Start the Python interpreter and use it as a calculator.

1. How many seconds are there in 42 minutes 42 seconds?

2. How many miles are there in 10 kilometers? Hint: there are 1.61 kilometers in a mile.

3. If you run a 10 kilometer race in 42 minutes 42 seconds, what is your average pace (time per mile in minutes and seconds)? What is your average speed in miles per hour?

Chapter 2. Variables, Expressions and Statements

One of the most powerful features of a programming language is the ability to manipulate **variables**. A variable is a name that refers to a value.

Assignment Statements

An **assignment statement** creates a new variable and gives it a value:

```
>>> message = 'And now for something completely different'
>>> n = 17
>>> pi = 3.141592653589793
```

This example makes three assignments. The first assigns a string to a new variable named `message`; the second gives the integer `17` to `n`; the third assigns the (approximate) value of π to `pi`.

A common way to represent variables on paper is to write the name with an arrow pointing to its value. This kind of figure is called a **state diagram** because it shows what state each of the variables is in (think of it as the variable's state of mind). Figure 2-1 shows the result of the previous example.

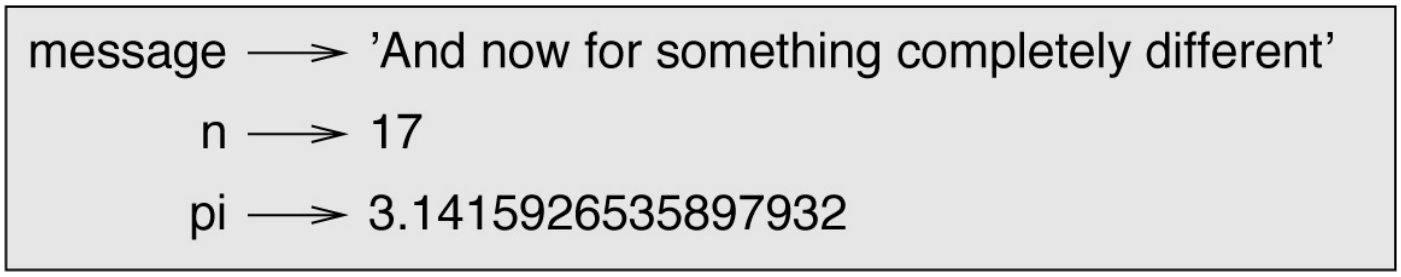


Figure 2-1. State diagram.

Variable Names

Programmers generally choose names for their variables that are meaningful — they document what the variable is used for.

Variable names can be as long as you like. They can contain both letters and numbers, but they can't begin with a number. It is legal to use uppercase letters, but it is conventional to use only lowercase for variables names.

The underscore character, _, can appear in a name. It is often used in names with multiple words, such as `your_name` or `airspeed_of_unladen_swallow`.

If you give a variable an illegal name, you get a syntax error:

```
>>> 76trombones = 'big parade'
SyntaxError: invalid syntax
>>> more@ = 1000000
SyntaxError: invalid syntax
>>> class = 'Advanced Theoretical Zymurgy'
SyntaxError: invalid syntax
```

`76trombones` is illegal because it begins with a number. `more@` is illegal because it contains an illegal character, `@`. But what's wrong with `class`?

It turns out that `class` is one of Python's **keywords**. The interpreter uses keywords to recognize the structure of the program, and they cannot be used as variable names.

Python 3 has these keywords:

```
False      class      finally    is         return
None       continue   for        lambda     try
True       def        from       nonlocal   while
and        del        global     not        with
as         elif       if         or         yield
assert     else       import     pass
break      except     in         raise
```

You don't have to memorize this list. In most development environments, keywords are displayed in a different color; if you try to use one as a variable name, you'll know.

Expressions and Statements

An **expression** is a combination of values, variables, and operators. A value all by itself is considered an expression, and so is a variable, so the following are all legal expressions:

```
>>> 42
42
>>> n
17
>>> n + 25
42
```

When you type an expression at the prompt, the interpreter **evaluates** it, which means that it finds the value of the expression. In this example, `n` has the value 17 and `n + 25` has the value 42.

A **statement** is a unit of code that has an effect, like creating a variable or displaying a value.

```
>>> n = 17
>>> print(n)
```

The first line is an assignment statement that gives a value to `n`. The second line is a print statement that displays the value of `n`.

When you type a statement, the interpreter **executes** it, which means that it does whatever the statement says. In general, statements don’t have values.

Script Mode

So far we have run Python in **interactive mode**, which means that you interact directly with the interpreter. Interactive mode is a good way to get started, but if you are working with more than a few lines of code, it can be clumsy.

The alternative is to save code in a file called a **script** and then run the interpreter in **script mode** to execute the script. By convention, Python scripts have names that end with `.py`.

If you know how to create and run a script on your computer, you are ready to go. Otherwise I recommend using PythonAnywhere again. I have posted instructions for running in script mode at *http://tinyurl.com/thinkpython2e*.

Because Python provides both modes, you can test bits of code in interactive mode before you put them in a script. But there are differences between interactive mode and script mode that can be confusing.

For example, if you are using Python as a calculator, you might type:

```
>>> miles = 26.2
>>> miles * 1.61
42.182
```

The first line assigns a value to `miles`, but it has no visible effect. The second line is an expression, so the interpreter evaluates it and displays the result. It turns out that a marathon is about 42 kilometers.

But if you type the same code into a script and run it, you get no output at all. In script mode an expression, all by itself, has no visible effect. Python actually evaluates the expression, but it doesn't display the value unless you tell it to:

```
miles = 26.2
print(miles * 1.61)
```

This behavior can be confusing at first.

A script usually contains a sequence of statements. If there is more than one statement, the results appear one at a time as the statements execute.

For example, the script

```
print(1)
x = 2
print(x)
```

produces the output

```
1
2
```

The assignment statement produces no output.

To check your understanding, type the following statements in the Python interpreter and see what they do:

```
5
x = 5
x + 1
```

Now put the same statements in a script and run it. What is the output? Modify the script by transforming each expression into a print statement and then run it again.

Order of Operations

When an expression contains more than one operator, the order of evaluation depends on the **order of operations**. For mathematical operators, Python follows mathematical convention. The acronym **PEMDAS** is a useful way to remember the rules:

- **P**arentheses have the highest precedence and can be used to force an expression to evaluate in the order you want. Since expressions in parentheses are evaluated first, `2 * (3-1)` is 4, and `(1+1)**(5-2)` is 8. You can also use parentheses to make an expression easier to read, as in `(minute * 100) / 60`, even if it doesn't change the result.

- **E**xponentiation has the next highest precedence, so `1 + 2**3` is 9, not 27, and `2 * 3**2` is 18, not 36.

- **M**ultiplication and **D**ivision have higher precedence than **A**ddition and **S**ubtraction. So `2*3-1` is 5, not 4, and `6+4/2` is 8, not 5.

- Operators with the same precedence are evaluated from left to right (except exponentiation). So in the expression `degrees / 2 * pi`, the division happens first and the result is multiplied by `pi`. To divide by 2π, you can use parentheses or write `degrees / 2 / pi`.

I don't work very hard to remember the precedence of operators. If I can't tell by looking at the expression, I use parentheses to make it obvious.

String Operations

In general, you can't perform mathematical operations on strings, even if the strings look like numbers, so the following are illegal:

```
'2'-'1'    'eggs'/'easy'    'third'*'a charm'
```

But there are two exceptions, `+` and `*`.

The `+` operator performs **string concatenation**, which means it joins the strings by linking them end-to-end. For example:

```
>>> first = 'throat'
>>> second = 'warbler'
>>> first + second
throatwarbler
```

The `*` operator also works on strings; it performs repetition. For example, `'Spam'*3` is `'SpamSpamSpam'`. If one of the values is a string, the other has to be an integer.

This use of `+` and `*` makes sense by analogy with addition and multiplication. Just as `4*3` is equivalent to `4+4+4`, we expect `'Spam'*3` to be the same as `'Spam'+'Spam'+'Spam'`, and it is. On the other hand, there is a significant way in which string concatenation and repetition are different from integer addition and multiplication. Can you think of a property that addition has that string concatenation does not?

Comments

As programs get bigger and more complicated, they get more difficult to read. Formal languages are dense, and it is often difficult to look at a piece of code and figure out what it is doing, or why.

For this reason, it is a good idea to add notes to your programs to explain in natural language what the program is doing. These notes are called **comments**, and they start with the # symbol:

```
# compute the percentage of the hour that has elapsed
percentage = (minute * 100) / 60
```

In this case, the comment appears on a line by itself. You can also put comments at the end of a line:

```
percentage = (minute * 100) / 60     # percentage of an hour
```

Everything from the # to the end of the line is ignored — it has no effect on the execution of the program.

Comments are most useful when they document non-obvious features of the code. It is reasonable to assume that the reader can figure out *what* the code does; it is more useful to explain *why*.

This comment is redundant with the code and useless:

```
v = 5     # assign 5 to v
```

This comment contains useful information that is not in the code:

```
v = 5     # velocity in meters/second.
```

Good variable names can reduce the need for comments, but long names can make complex expressions hard to read, so there is a trade-off.

Debugging

Three kinds of errors can occur in a program: syntax errors, runtime errors, and semantic errors. It is useful to distinguish between them in order to track them down more quickly.

Syntax error:

> "Syntax" refers to the structure of a program and the rules about that structure. For example, parentheses have to come in matching pairs, so `(1 + 2)` is legal, but `8)` is a **syntax error**.
>
> If there is a syntax error anywhere in your program, Python displays an error message and quits, and you will not be able to run the program. During the first few weeks of your programming career, you might spend a lot of time tracking down syntax errors. As you gain experience, you will make fewer errors and find them faster.

Runtime error:

> The second type of error is a runtime error, so called because the error does not appear until after the program has started running. These errors are also called **exceptions** because they usually indicate that something exceptional (and bad) has happened.
>
> Runtime errors are rare in the simple programs you will see in the first few chapters, so it might be a while before you encounter one.

Semantic error:

> The third type of error is "semantic", which means related to meaning. If there is a semantic error in your program, it will run without generating error messages, but it will not do the right thing. It will do something else. Specifically, it will do what you told it to do.
>
> Identifying semantic errors can be tricky because it requires you to work backward by looking at the output of the program and trying to figure out what it is doing.

Glossary

variable:

A name that refers to a value.

assignment:

A statement that assigns a value to a variable.

state diagram:

A graphical representation of a set of variables and the values they refer to.

keyword:

A reserved word that is used to parse a program; you cannot use keywords like `if`, `def`, and `while` as variable names.

operand:

One of the values on which an operator operates.

expression:

A combination of variables, operators, and values that represents a single result.

evaluate:

To simplify an expression by performing the operations in order to yield a single value.

statement:

A section of code that represents a command or action. So far, the statements we have seen are assignments and print statements.

execute:

To run a statement and do what it says.

interactive mode:

A way of using the Python interpreter by typing code at the prompt.

script mode:

A way of using the Python interpreter to read code from a script and run it.

script:

A program stored in a file.

order of operations:

Rules governing the order in which expressions involving multiple operators and operands are evaluated.

concatenate:

To join two operands end-to-end.

comment:

Information in a program that is meant for other programmers (or anyone reading the source code) and has no effect on the execution of the program.

syntax error:

An error in a program that makes it impossible to parse (and therefore impossible to interpret).

exception:

An error that is detected while the program is running.

semantics:

The meaning of a program.

semantic error:

An error in a program that makes it do something other than what the programmer intended.

Exercises

Exercise 2-1.

Repeating my advice from the previous chapter, whenever you learn a new feature, you should try it out in interactive mode and make errors on purpose to see what goes wrong.

- We've seen that `n = 42` is legal. What about `42 = n`?
- How about `x = y = 1`?
- In some languages every statement ends with a semicolon, `;`. What happens if you put a semicolon at the end of a Python statement?
- What if you put a period at the end of a statement?
- In math notation you can multiply *x* and *y* like this: xy. What happens if you try that in Python?

Exercise 2-2.

Practice using the Python interpreter as a calculator:

1. The volume of a sphere with radius *r* is $\frac{4}{3}\pi r^3$. What is the volume of a sphere with radius 5?
2. Suppose the cover price of a book is $24.95, but bookstores get a 40% discount. Shipping costs $3 for the first copy and 75 cents for each additional copy. What is the total wholesale cost for 60 copies?
3. If I leave my house at 6:52 am and run 1 mile at an easy pace (8:15 per mile), then 3 miles at tempo (7:12 per mile) and 1 mile at an easy pace again, what time do I get home for breakfast?

Chapter 3. Functions

In the context of programming, a **function** is a named sequence of statements that performs a computation. When you define a function, you specify the name and the sequence of statements. Later, you can “call” the function by name.

Function Calls

We have already seen one example of a **function call**:

```
>>> type(42)
<class 'int'>
```

The name of the function is `type`. The expression in parentheses is called the **argument** of the function. The result, for this function, is the type of the argument.

It is common to say that a function “takes” an argument and “returns” a result. The result is also called the **return value**.

Python provides functions that convert values from one type to another. The `int` function takes any value and converts it to an integer, if it can, or complains otherwise:

```
>>> int('32')
32
>>> int('Hello')
ValueError: invalid literal for int(): Hello
```

`int` can convert floating-point values to integers, but it doesn’t round off; it chops off the fraction part:

```
>>> int(3.99999)
3
>>> int(-2.3)
-2
```

`float` converts integers and strings to floating-point numbers:

```
>>> float(32)
32.0
>>> float('3.14159')
3.14159
```

Finally, `str` converts its argument to a string:

```
>>> str(32)
'32'
>>> str(3.14159)
'3.14159'
```

Math Functions

Python has a math module that provides most of the familiar mathematical functions. A **module** is a file that contains a collection of related functions.

Before we can use the functions in a module, we have to import it with an **import statement**:

```
>>> import math
```

This statement creates a **module object** named math. If you display the module object, you get some information about it:

```
>>> math
<module 'math' (built-in)>
```

The module object contains the functions and variables defined in the module. To access one of the functions, you have to specify the name of the module and the name of the function, separated by a dot (also known as a period). This format is called **dot notation**.

```
>>> ratio = signal_power / noise_power
>>> decibels = 10 * math.log10(ratio)

>>> radians = 0.7
>>> height = math.sin(radians)
```

The first example uses `math.log10` to compute a signal-to-noise ratio in decibels (assuming that `signal_power` and `noise_power` are defined). The math module also provides `log`, which computes logarithms base `e`.

The second example finds the sine of `radians`. The name of the variable is a hint that `sin` and the other trigonometric functions (`cos`, `tan`, etc.) take arguments in radians. To convert from degrees to radians, divide by 180 and multiply by π:

```
>>> degrees = 45
>>> radians = degrees / 180.0 * math.pi
>>> math.sin(radians)
0.707106781187
```

The expression `math.pi` gets the variable `pi` from the math module. Its value is a floating-point approximation of π, accurate to about 15 digits.

If you know trigonometry, you can check the previous result by comparing it to the square root of 2 divided by 2:

```
>>> math.sqrt(2) / 2.0
0.707106781187
```

Composition

So far, we have looked at the elements of a program — variables, expressions, and statements — in isolation, without talking about how to combine them.

One of the most useful features of programming languages is their ability to take small building blocks and **compose** them. For example, the argument of a function can be any kind of expression, including arithmetic operators:

```
x = math.sin(degrees / 360.0 * 2 * math.pi)
```

And even function calls:

```
x = math.exp(math.log(x+1))
```

Almost anywhere you can put a value, you can put an arbitrary expression, with one exception: the left side of an assignment statement has to be a variable name. Any other expression on the left side is a syntax error (we will see exceptions to this rule later).

```
>>> minutes = hours * 60                # right
>>> hours * 60 = minutes                # wrong!
SyntaxError: can't assign to operator
```

Adding New Functions

So far, we have only been using the functions that come with Python, but it is also possible to add new functions. A **function definition** specifies the name of a new function and the sequence of statements that run when the function is called.

Here is an example:

```
def print_lyrics():
    print("I'm a lumberjack, and I'm okay.")
    print("I sleep all night and I work all day.")
```

`def` is a keyword that indicates that this is a function definition. The name of the function is `print_lyrics`. The rules for function names are the same as for variable names: letters, numbers and underscore are legal, but the first character can't be a number. You can't use a keyword as the name of a function, and you should avoid having a variable and a function with the same name.

The empty parentheses after the name indicate that this function doesn't take any arguments.

The first line of the function definition is called the **header**; the rest is called the **body**. The header has to end with a colon and the body has to be indented. By convention, indentation is always four spaces. The body can contain any number of statements.

The strings in the print statements are enclosed in double quotes. Single quotes and double quotes do the same thing; most people use single quotes except in cases like this where a single quote (which is also an apostrophe) appears in the string.

All quotation marks (single and double) must be "straight quotes", usually located next to Enter on the keyboard. "Curly quotes", like the ones in this sentence, are not legal in Python.

If you type a function definition in interactive mode, the interpreter prints dots (. . .) to let you know that the definition isn't complete:

```
>>> def print_lyrics():
...     print("I'm a lumberjack, and I'm okay.")
...     print("I sleep all night and I work all day.")
...
```

To end the function, you have to enter an empty line.

Defining a function creates a **function object**, which has type `function`:

```
>>> print(print_lyrics)
<function print_lyrics at 0xb7e99e9c>
>>> type(print_lyrics)
<class 'function'>
```

The syntax for calling the new function is the same as for built-in functions:

```
>>> print_lyrics()
I'm a lumberjack, and I'm okay.
I sleep all night and I work all day.
```

Once you have defined a function, you can use it inside another function. For example, to repeat the previous refrain, we could write a function called `repeat_lyrics`:

```
def repeat_lyrics():
    print_lyrics()
    print_lyrics()
```

And then call `repeat_lyrics`:

```
>>> repeat_lyrics()
I'm a lumberjack, and I'm okay.
I sleep all night and I work all day.
I'm a lumberjack, and I'm okay.
I sleep all night and I work all day.
```

But that’s not really how the song goes.

Definitions and Uses

Pulling together the code fragments from the previous section, the whole program looks like this:

```
def print_lyrics():
    print("I'm a lumberjack, and I'm okay.")
    print("I sleep all night and I work all day.")

def repeat_lyrics():
    print_lyrics()
    print_lyrics()

repeat_lyrics()
```

This program contains two function definitions: `print_lyrics` and `repeat_lyrics`. Function definitions get executed just like other statements, but the effect is to create function objects. The statements inside the function do not run until the function is called, and the function definition generates no output.

As you might expect, you have to create a function before you can run it. In other words, the function definition has to run before the function gets called.

As an exercise, move the last line of this program to the top, so the function call appears before the definitions. Run the program and see what error message you get.

Now move the function call back to the bottom and move the definition of `print_lyrics` after the definition of `repeat_lyrics`. What happens when you run this program?

Flow of Execution

To ensure that a function is defined before its first use, you have to know the order statements run in, which is called the **flow of execution**.

Execution always begins at the first statement of the program. Statements are run one at a time, in order from top to bottom.

Function definitions do not alter the flow of execution of the program, but remember that statements inside the function don’t run until the function is called.

A function call is like a detour in the flow of execution. Instead of going to the next statement, the flow jumps to the body of the function, runs the statements there, and then comes back to pick up where it left off.

That sounds simple enough, until you remember that one function can call another. While in the middle of one function, the program might have to run the statements in another function. Then, while running that new function, the program might have to run yet another function!

Fortunately, Python is good at keeping track of where it is, so each time a function completes, the program picks up where it left off in the function that called it. When it gets to the end of the program, it terminates.

In summary, when you read a program, you don’t always want to read from top to bottom. Sometimes it makes more sense if you follow the flow of execution.

Parameters and Arguments

Some of the functions we have seen require arguments. For example, when you call `math.sin` you pass a number as an argument. Some functions take more than one argument: `math.pow` takes two, the base and the exponent.

Inside the function, the arguments are assigned to variables called **parameters**. Here is a definition for a function that takes an argument:

```
def print_twice(bruce):
    print(bruce)
    print(bruce)
```

This function assigns the argument to a parameter named `bruce`. When the function is called, it prints the value of the parameter (whatever it is) twice.

This function works with any value that can be printed:

```
>>> print_twice('Spam')
Spam
Spam
>>> print_twice(42)
42
42
>>> print_twice(math.pi)
3.14159265359
3.14159265359
```

The same rules of composition that apply to built-in functions also apply to programmer-defined functions, so we can use any kind of expression as an argument for `print_twice`:

```
>>> print_twice('Spam '*4)
Spam Spam Spam Spam
Spam Spam Spam Spam
>>> print_twice(math.cos(math.pi))
-1.0
-1.0
```

The argument is evaluated before the function is called, so in the examples the expressions `'Spam '*4` and `math.cos(math.pi)` are only evaluated once.

You can also use a variable as an argument:

```
>>> michael = 'Eric, the half a bee.'
>>> print_twice(michael)
Eric, the half a bee.
Eric, the half a bee.
```

The name of the variable we pass as an argument (`michael`) has nothing to do with the name of the parameter (`bruce`). It doesn't matter what the value was called back home (in the caller); here in `print_twice`, we call everybody `bruce`.

Variables and Parameters Are Local

When you create a variable inside a function, it is **local**, which means that it only exists inside the function. For example:

```
def cat_twice(part1, part2):
    cat = part1 + part2
    print_twice(cat)
```

This function takes two arguments, concatenates them, and prints the result twice. Here is an example that uses it:

```
>>> line1 = 'Bing tiddle '
>>> line2 = 'tiddle bang.'
>>> cat_twice(line1, line2)
Bing tiddle tiddle bang.
Bing tiddle tiddle bang.
```

When `cat_twice` terminates, the variable `cat` is destroyed. If we try to print it, we get an exception:

```
>>> print(cat)
NameError: name 'cat' is not defined
```

Parameters are also local. For example, outside `print_twice`, there is no such thing as `bruce`.

Stack Diagrams

To keep track of which variables can be used where, it is sometimes useful to draw a **stack diagram**. Like state diagrams, stack diagrams show the value of each variable, but they also show the function each variable belongs to.

Each function is represented by a **frame**. A frame is a box with the name of a function beside it and the parameters and variables of the function inside it. The stack diagram for the previous example is shown in Figure 3-1.

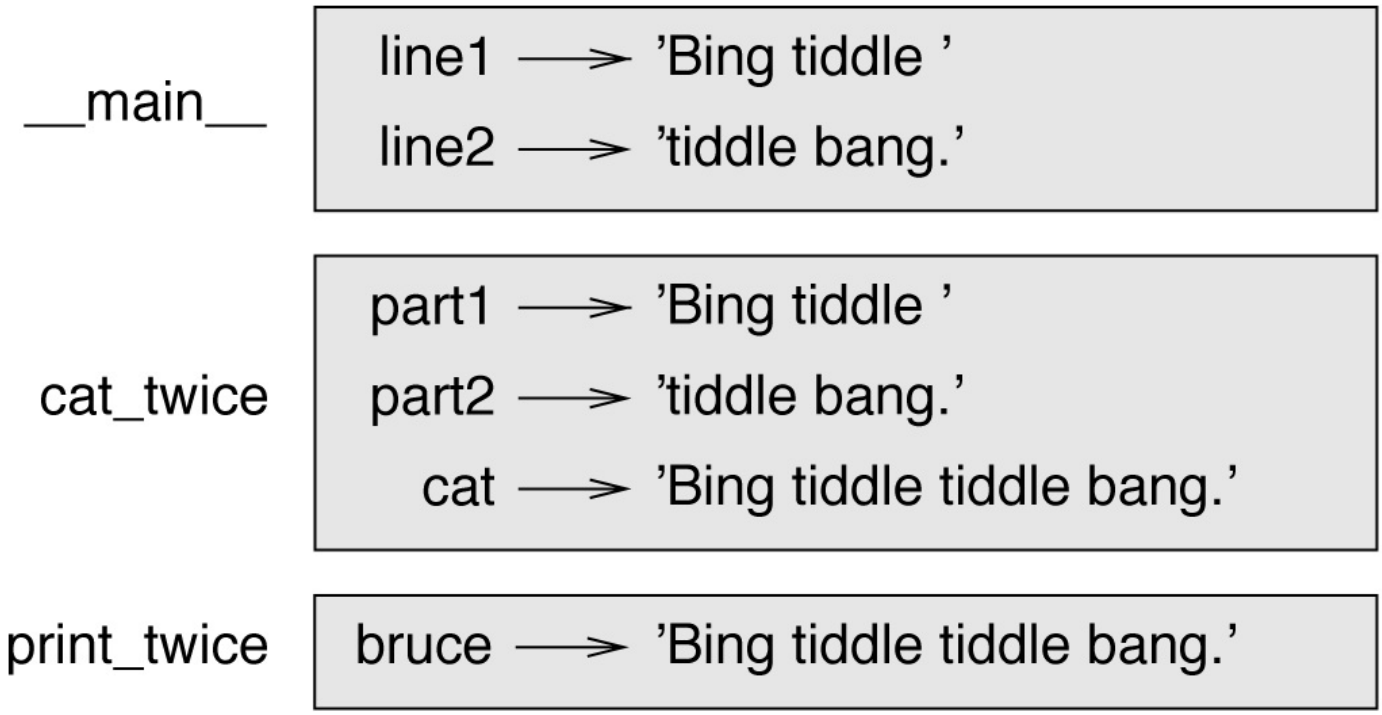


Figure 3-1. Stack diagram.

The frames are arranged in a stack that indicates which function called which, and so on. In this example, `print_twice` was called by `cat_twice`, and `cat_twice` was called by `__main__`, which is a special name for the topmost frame. When you create a variable outside of any function, it belongs to `__main__`.

Each parameter refers to the same value as its corresponding argument. So, `part1` has the same value as `line1`, `part2` has the same value as `line2`, and `bruce` has the same value as `cat`.

If an error occurs during a function call, Python prints the name of the function, the name of the function that called it, and the name of the function that called *that*, all the way back to `__main__`.

For example, if you try to access `cat` from within `print_twice`, you get a `NameError`:

```
Traceback (innermost last):
  File "test.py", line 13, in __main__
    cat_twice(line1, line2)
  File "test.py", line 5, in cat_twice
    print_twice(cat)
  File "test.py", line 9, in print_twice
```

```
    print(cat)
NameError: name 'cat' is not defined
```

This list of functions is called a **traceback**. It tells you what program file the error occurred in, and what line, and what functions were executing at the time. It also shows the line of code that caused the error.

The order of the functions in the traceback is the same as the order of the frames in the stack diagram. The function that is currently running is at the bottom.

Fruitful Functions and Void Functions

Some of the functions we have used, such as the math functions, return results; for lack of a better name, I call them **fruitful functions**. Other functions, like `print_twice`, perform an action but don't return a value. They are called **void functions**.

When you call a fruitful function, you almost always want to do something with the result; for example, you might assign it to a variable or use it as part of an expression:

```
x = math.cos(radians)
golden = (math.sqrt(5) + 1) / 2
```

When you call a function in interactive mode, Python displays the result:

```
>>> math.sqrt(5)
2.2360679774997898
```

But in a script, if you call a fruitful function all by itself, the return value is lost forever!

```
math.sqrt(5)
```

This script computes the square root of 5, but since it doesn't store or display the result, it is not very useful.

Void functions might display something on the screen or have some other effect, but they don't have a return value. If you assign the result to a variable, you get a special value called `None`:

```
>>> result = print_twice('Bing')
Bing
Bing
>>> print(result)
None
```

The value `None` is not the same as the string `'None'`. It is a special value that has its own type:

```
>>> print(type(None))
<class 'NoneType'>
```

The functions we have written so far are all void. We will start writing fruitful functions in a few chapters.

Why Functions?

It may not be clear why it is worth the trouble to divide a program into functions. There are several reasons:

- Creating a new function gives you an opportunity to name a group of statements, which makes your program easier to read and debug.
- Functions can make a program smaller by eliminating repetitive code. Later, if you make a change, you only have to make it in one place.
- Dividing a long program into functions allows you to debug the parts one at a time and then assemble them into a working whole.
- Well-designed functions are often useful for many programs. Once you write and debug one, you can reuse it.

Debugging

One of the most important skills you will acquire is debugging. Although it can be frustrating, debugging is one of the most intellectually rich, challenging, and interesting parts of programming.

In some ways debugging is like detective work. You are confronted with clues and you have to infer the processes and events that led to the results you see.

Debugging is also like an experimental science. Once you have an idea about what is going wrong, you modify your program and try again. If your hypothesis was correct, you can predict the result of the modification, and you take a step closer to a working program. If your hypothesis was wrong, you have to come up with a new one. As Sherlock Holmes pointed out, “When you have eliminated the impossible, whatever remains, however improbable, must be the truth.” (A. Conan Doyle, *The Sign of Four*).

For some people, programming and debugging are the same thing. That is, programming is the process of gradually debugging a program until it does what you want. The idea is that you should start with a working program and make small modifications, debugging them as you go.

For example, Linux is an operating system that contains millions of lines of code, but it started out as a simple program Linus Torvalds used to explore the Intel 80386 chip. According to Larry Greenfield, “One of Linus’s earlier projects was a program that would switch between printing AAAA and BBBB. This later evolved to Linux.” (*The Linux Users’ Guide* Beta Version 1).

Glossary

function:

A named sequence of statements that performs some useful operation. Functions may or may not take arguments and may or may not produce a result.

function definition:

A statement that creates a new function, specifying its name, parameters, and the statements it contains.

function object:

A value created by a function definition. The name of the function is a variable that refers to a function object.

header:

The first line of a function definition.

body:

The sequence of statements inside a function definition.

parameter:

A name used inside a function to refer to the value passed as an argument.

function call:

A statement that runs a function. It consists of the function name followed by an argument list in parentheses.

argument:

A value provided to a function when the function is called. This value is assigned to the corresponding parameter in the function.

local variable:

A variable defined inside a function. A local variable can only be used inside its function.

return value:

The result of a function. If a function call is used as an expression, the return value is the value of the expression.

fruitful function:

A function that returns a value.

void function:

A function that always returns `None`.

`None`:

A special value returned by void functions.

module:

A file that contains a collection of related functions and other definitions.

import statement:

A statement that reads a module file and creates a module object.

module object:

A value created by an `import` statement that provides access to the values defined in a module.

dot notation:

The syntax for calling a function in another module by specifying the module name followed by a dot (period) and the function name.

composition:

Using an expression as part of a larger expression, or a statement as part of a larger statement.

flow of execution:

The order statements run in.

stack diagram:

A graphical representation of a stack of functions, their variables, and the values they refer to.

frame:

A box in a stack diagram that represents a function call. It contains the local variables and parameters of the function.

traceback:

A list of the functions that are executing, printed when an exception occurs.

Exercises

Exercise 3-1.

Write a function named `right_justify` that takes a string named `s` as a parameter and prints the string with enough leading spaces so that the last letter of the string is in column 70 of the display:

```
>>> right_justify('monty')
                                                                monty
```

Hint: Use string concatenation and repetition. Also, Python provides a built-in function called `len` that returns the length of a string, so the value of `len('monty')` is 5.

Exercise 3-2.

A function object is a value you can assign to a variable or pass as an argument. For example, `do_twice` is a function that takes a function object as an argument and calls it twice:

```
def do_twice(f):
    f()
    f()
```

Here's an example that uses `do_twice` to call a function named `print_spam` twice:

```
def print_spam():
    print('spam')

do_twice(print_spam)
```

1. Type this example into a script and test it.

2. Modify `do_twice` so that it takes two arguments, a function object and a value, and calls the function twice, passing the value as an argument.

3. Copy the definition of `print_twice` from earlier in this chapter to your script.

4. Use the modified version of `do_twice` to call `print_twice` twice, passing `'spam'` as an argument.

5. Define a new function called `do_four` that takes a function object and a value and calls the function four times, passing the value as a parameter. There should be only two statements in the body of this function, not four.

Solution: *http://thinkpython2.com/code/do_four.py*.

Exercise 3-3.

Note: This exercise should be done using only the statements and other features we have learned so far.

1. Write a function that draws a grid like the following:

```
+ - - - - + - - - - +
|         |         |
|         |         |
|         |         |
|         |         |
```

```
+ - - - - + - - - - +
|         |         |
|         |         |
|         |         |
|         |         |
+ - - - - + - - - - +
```

Hint: to print more than one value on a line, you can print a comma-separated sequence of values:

```
print('+', '-')
```

By default, `print` advances to the next line, but you can override that behavior and put a space at the end, like this:

```
print('+', end=' ')
print('-')
```

The output of these statements is `'+ -'`.
A `print` statement with no argument ends the current line and goes to the next line.

2. Write a function that draws a similar grid with four rows and four columns.

Solution: *http://thinkpython2.com/code/grid.py*. Credit: This exercise is based on an exercise in Oualline, *Practical C Programming, Third Edition*, O'Reilly Media, 1997.

Chapter 4. Case Study: Interface Design

This chapter presents a case study that demonstrates a process for designing functions that work together.

It introduces the `turtle` module, which allows you to create images using turtle graphics. The `turtle` module is included in most Python installations, but if you are running Python using PythonAnywhere, you won't be able to run the turtle examples (at least you couldn't when I wrote this).

If you have already installed Python on your computer, you should be able to run the examples. Otherwise, now is a good time to install. I have posted instructions at *http://tinyurl.com/thinkpython2e*.

Code examples from this chapter are available from *http://thinkpython2.com/code/polygon.py*.

The turtle Module

To check whether you have the `turtle` module, open the Python interpreter and type:

```
>>> import turtle
>>> bob = turtle.Turtle()
```

When you run this code, it should create a new window with a small arrow that represents the turtle. Close the window.

Create a file named `mypolygon.py` and type in the following code:

```
import turtle
bob = turtle.Turtle()
print(bob)
turtle.mainloop()
```

The `turtle` module (with a lowercase *t*) provides a function called `Turtle` (with an uppercase *T*) that creates a Turtle object, which we assign to a variable named `bob`. Printing `bob` displays something like:

```
<turtle.Turtle object at 0xb7bfbf4c>
```

This means that `bob` refers to an object with type `Turtle` as defined in module `turtle`.

`mainloop` tells the window to wait for the user to do something, although in this case there's not much for the user to do except close the window.

Once you create a Turtle, you can call a **method** to move it around the window. A method is similar to a function, but it uses slightly different syntax. For example, to move the turtle forward:

```
bob.fd(100)
```

The method, `fd`, is associated with the turtle object we're calling `bob`. Calling a method is like making a request: you are asking `bob` to move forward.

The argument of `fd` is a distance in pixels, so the actual size depends on your display.

Other methods you can call on a Turtle are `bk` to move backward, `lt` for left turn, and `rt` right turn. The argument for `lt` and `rt` is an angle in degrees.

Also, each Turtle is holding a pen, which is either down or up; if the pen is down, the Turtle leaves a trail when it moves. The methods `pu` and `pd` stand for "pen up" and "pen down".

To draw a right angle, add these lines to the program (after creating `bob` and before calling `mainloop`):

```
bob.fd(100)
```

```
bob.lt(90)
bob.fd(100)
```

When you run this program, you should see `bob` move east and then north, leaving two line segments behind.

Now modify the program to draw a square. Don’t go on until you’ve got it working!

Simple Repetition

Chances are you wrote something like this:

```
bob.fd(100)
bob.lt(90)

bob.fd(100)
bob.lt(90)

bob.fd(100)
bob.lt(90)

bob.fd(100)
```

We can do the same thing more concisely with a `for` statement. Add this example to `mypolygon.py` and run it again:

```
for i in range(4):
    print('Hello!')
```

You should see something like this:

```
Hello!
Hello!
Hello!
Hello!
```

This is the simplest use of the `for` statement; we will see more later. But that should be enough to let you rewrite your square-drawing program. Don't go on until you do.

Here is a `for` statement that draws a square:

```
for i in range(4):
    bob.fd(100)
    bob.lt(90)
```

The syntax of a `for` statement is similar to a function definition. It has a header that ends with a colon and an indented body. The body can contain any number of statements.

A `for` statement is also called a **loop** because the flow of execution runs through the body and then loops back to the top. In this case, it runs the body four times.

This version is actually a little different from the previous square-drawing code because it makes another turn after drawing the last side of the square. The extra turn takes more time, but it simplifies the code if we do the same thing every time through the loop. This version also has the effect of leaving the turtle back in the starting position, facing in the starting direction.

Exercises

The following is a series of exercises using TurtleWorld. They are meant to be fun, but they have a point, too. While you are working on them, think about what the point is.

The following sections have solutions to the exercises, so don't look until you have finished (or at least tried).

1. Write a function called `square` that takes a parameter named `t`, which is a turtle. It should use the turtle to draw a square.
   Write a function call that passes `bob` as an argument to `square`, and then run the program again.

2. Add another parameter, named `length`, to `square`. Modify the body so length of the sides is `length`, and then modify the function call to provide a second argument. Run the program again. Test your program with a range of values for `length`.

3. Make a copy of `square` and change the name to `polygon`. Add another parameter named `n` and modify the body so it draws an n-sided regular polygon.
   Hint: The exterior angles of an n-sided regular polygon are *360/n* degrees.

4. Write a function called `circle` that takes a turtle, `t`, and radius, `r`, as parameters and that draws an approximate circle by calling `polygon` with an appropriate length and number of sides. Test your function with a range of values of `r`.
   Hint: figure out the circumference of the circle and make sure that `length * n = circumference`.

5. Make a more general version of `circle` called `arc` that takes an additional parameter `angle`, which determines what fraction of a circle to draw. `angle` is in units of degrees, so when `angle=360`, `arc` should draw a complete circle.

Encapsulation

The first exercise asks you to put your square-drawing code into a function definition and then call the function, passing the turtle as a parameter. Here is a solution:

```
def square(t):
    for i in range(4):
        t.fd(100)
        t.lt(90)

square(bob)
```

The innermost statements, `fd` and `lt`, are indented twice to show that they are inside the `for` loop, which is inside the function definition. The next line, `square(bob)`, is flush with the left margin, which indicates the end of both the `for` loop and the function definition.

Inside the function, `t` refers to the same turtle `bob`, so `t.lt(90)` has the same effect as `bob.lt(90)`. In that case, why not call the parameter `bob`? The idea is that `t` can be any turtle, not just `bob`, so you could create a second turtle and pass it as an argument to `square`:

```
alice = Turtle()
square(alice)
```

Wrapping a piece of code up in a function is called **encapsulation**. One of the benefits of encapsulation is that it attaches a name to the code, which serves as a kind of documentation. Another advantage is that if you reuse the code, it is more concise to call a function twice than to copy and paste the body!

Generalization

The next step is to add a `length` parameter to `square`. Here is a solution:

```
def square(t, length):
    for i in range(4):
        t.fd(length)
        t.lt(90)

square(bob, 100)
```

Adding a parameter to a function is called **generalization** because it makes the function more general: in the previous version, the square is always the same size; in this version it can be any size.

The next step is also a generalization. Instead of drawing squares, `polygon` draws regular polygons with any number of sides. Here is a solution:

```
def polygon(t, n, length):
    angle = 360 / n
    for i in range(n):
        t.fd(length)
        t.lt(angle)

polygon(bob, 7, 70)
```

This example draws a 7-sided polygon with side length 70.

If you are using Python 2, the value of `angle` might be off because of integer division. A simple solution is to compute `angle = 360.0 / n`. Because the numerator is a floating-point number, the result is floating point.

When a function has more than a few numeric arguments, it is easy to forget what they are, or what order they should be in. In that case it is often a good idea to include the names of the parameters in the argument list:

```
polygon(bob, n=7, length=70)
```

These are called **keyword arguments** because they include the parameter names as "keywords" (not to be confused with Python keywords like `while` and `def`).

This syntax makes the program more readable. It is also a reminder about how arguments and parameters work: when you call a function, the arguments are assigned to the parameters.

Interface Design

The next step is to write `circle`, which takes a radius, `r`, as a parameter. Here is a simple solution that uses `polygon` to draw a 50-sided polygon:

```
import math

def circle(t, r):
    circumference = 2 * math.pi * r
    n = 50
    length = circumference / n
    polygon(t, n, length)
```

The first line computes the circumference of a circle with radius `r` using the formula $2\pi r$. Since we use `math.pi`, we have to import `math`. By convention, `import` statements are usually at the beginning of the script.

`n` is the number of line segments in our approximation of a circle, so `length` is the length of each segment. Thus, `polygon` draws a 50-sided polygon that approximates a circle with radius `r`.

One limitation of this solution is that `n` is a constant, which means that for very big circles, the line segments are too long, and for small circles, we waste time drawing very small segments. One solution would be to generalize the function by taking `n` as a parameter. This would give the user (whoever calls `circle`) more control, but the interface would be less clean.

The **interface** of a function is a summary of how it is used: what are the parameters? What does the function do? And what is the return value? An interface is "clean" if it allows the caller to do what they want without dealing with unnecessary details.

In this example, `r` belongs in the interface because it specifies the circle to be drawn. `n` is less appropriate because it pertains to the details of *how* the circle should be rendered.

Rather than clutter up the interface, it is better to choose an appropriate value of `n` depending on `circumference`:

```
def circle(t, r):
    circumference = 2 * math.pi * r
    n = int(circumference / 3) + 1
    length = circumference / n
    polygon(t, n, length)
```

Now the number of segments is an integer near `circumference/3`, so the length of each segment is approximately 3, which is small enough that the circles look good, but big enough to be efficient, and acceptable for any size circle.

Refactoring

When I wrote `circle`, I was able to reuse `polygon` because a many-sided polygon is a good approximation of a circle. But `arc` is not as cooperative; we can't use `polygon` or `circle` to draw an arc.

One alternative is to start with a copy of `polygon` and transform it into `arc`. The result might look like this:

```
def arc(t, r, angle):
    arc_length = 2 * math.pi * r * angle / 360
    n = int(arc_length / 3) + 1
    step_length = arc_length / n
    step_angle = angle / n

    for i in range(n):
        t.fd(step_length)
        t.lt(step_angle)
```

The second half of this function looks like `polygon`, but we can't reuse `polygon` without changing the interface. We could generalize `polygon` to take an angle as a third argument, but then `polygon` would no longer be an appropriate name! Instead, let's call the more general function `polyline`:

```
def polyline(t, n, length, angle):
    for i in range(n):
        t.fd(length)
        t.lt(angle)
```

Now we can rewrite `polygon` and `arc` to use `polyline`:

```
def polygon(t, n, length):
    angle = 360.0 / n
    polyline(t, n, length, angle)

def arc(t, r, angle):
    arc_length = 2 * math.pi * r * angle / 360
    n = int(arc_length / 3) + 1
    step_length = arc_length / n
    step_angle = float(angle) / n
    polyline(t, n, step_length, step_angle)
```

Finally, we can rewrite `circle` to use `arc`:

```
def circle(t, r):
    arc(t, r, 360)
```

This process — rearranging a program to improve interfaces and facilitate code reuse — is called **refactoring**. In this case, we noticed that there was similar code in `arc` and `polygon`, so we "factored it out" into `polyline`.

If we had planned ahead, we might have written `polyline` first and avoided refactoring, but often you don't know enough at the beginning of a project to design all the interfaces. Once you start coding, you understand the problem better. Sometimes refactoring is a sign

that you have learned something.

A Development Plan

A **development plan** is a process for writing programs. The process we used in this case study is “encapsulation and generalization”. The steps of this process are:

1. Start by writing a small program with no function definitions.

2. Once you get the program working, identify a coherent piece of it, encapsulate the piece in a function and give it a name.

3. Generalize the function by adding appropriate parameters.

4. Repeat steps 1–3 until you have a set of working functions. Copy and paste working code to avoid retyping (and re-debugging).

5. Look for opportunities to improve the program by refactoring. For example, if you have similar code in several places, consider factoring it into an appropriately general function.

This process has some drawbacks — we will see alternatives later — but it can be useful if you don’t know ahead of time how to divide the program into functions. This approach lets you design as you go along.

docstring

A **docstring** is a string at the beginning of a function that explains the interface ("doc" is short for "documentation"). Here is an example:

```
def polyline(t, n, length, angle):
    """Draws n line segments with the given length and
    angle (in degrees) between them.  t is a turtle.
    """
    for i in range(n):
        t.fd(length)
        t.lt(angle)
```

By convention, all docstrings are triple-quoted strings, also known as multiline strings because the triple quotes allow the string to span more than one line.

It is terse, but it contains the essential information someone would need to use this function. It explains concisely what the function does (without getting into the details of how it does it). It explains what effect each parameter has on the behavior of the function and what type each parameter should be (if it is not obvious).

Writing this kind of documentation is an important part of interface design. A well-designed interface should be simple to explain; if you have a hard time explaining one of your functions, maybe the interface could be improved.

Debugging

An interface is like a contract between a function and a caller. The caller agrees to provide certain parameters and the function agrees to do certain work.

For example, `polyline` requires four arguments: `t` has to be a Turtle; `n` has to be an integer; `length` should be a positive number; and `angle` has to be a number, which is understood to be in degrees.

These requirements are called **preconditions** because they are supposed to be true before the function starts executing. Conversely, conditions at the end of the function are **postconditions**. Postconditions include the intended effect of the function (like drawing line segments) and any side effects (like moving the Turtle or making other changes).

Preconditions are the responsibility of the caller. If the caller violates a (properly documented!) precondition and the function doesn't work correctly, the bug is in the caller, not the function.

If the preconditions are satisfied and the postconditions are not, the bug is in the function. If your pre- and postconditions are clear, they can help with debugging.

Glossary

method:

A function that is associated with an object and called using dot notation.

loop:

A part of a program that can run repeatedly.

encapsulation:

The process of transforming a sequence of statements into a function definition.

generalization:

The process of replacing something unnecessarily specific (like a number) with something appropriately general (like a variable or parameter).

keyword argument:

An argument that includes the name of the parameter as a “keyword”.

interface:

A description of how to use a function, including the name and descriptions of the arguments and return value.

refactoring:

The process of modifying a working program to improve function interfaces and other qualities of the code.

development plan:

A process for writing programs.

docstring:

A string that appears at the top of a function definition to document the function’s interface.

precondition:

A requirement that should be satisfied by the caller before a function starts.

postcondition:

A requirement that should be satisfied by the function before it ends.

Exercises

Exercise 4-1.

Download the code in this chapter from *http://thinkpython2.com/code/polygon.py*.

1. Draw a stack diagram that shows the state of the program while executing `circle(bob, radius)`. You can do the arithmetic by hand or add `print` statements to the code.

2. The version of `arc` in "Refactoring" is not very accurate because the linear approximation of the circle is always outside the true circle. As a result, the Turtle ends up a few pixels away from the correct destination. My solution shows a way to reduce the effect of this error. Read the code and see if it makes sense to you. If you draw a diagram, you might see how it works.

Exercise 4-2.

Write an appropriately general set of functions that can draw flowers as in Figure 4-1.

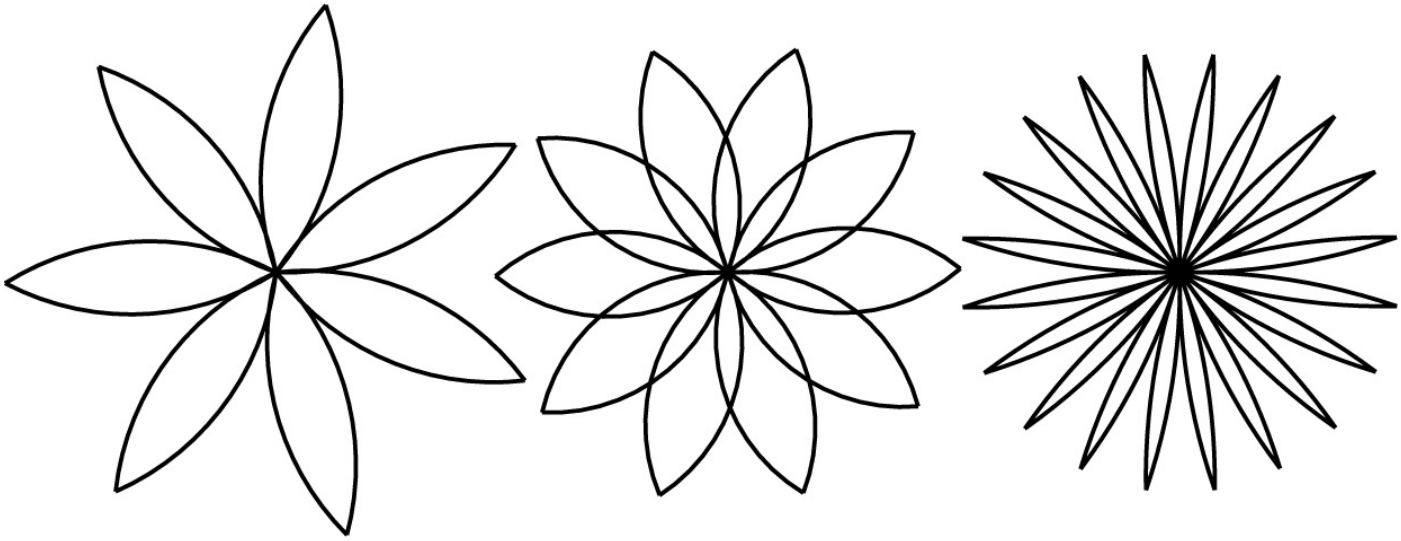


Figure 4-1. Turtle flowers.

Solution: *http://thinkpython2.com/code/flower.py*, also requires *http://thinkpython2.com/code/polygon.py*.

Exercise 4-3.

Write an appropriately general set of functions that can draw shapes as in Figure 4-2.

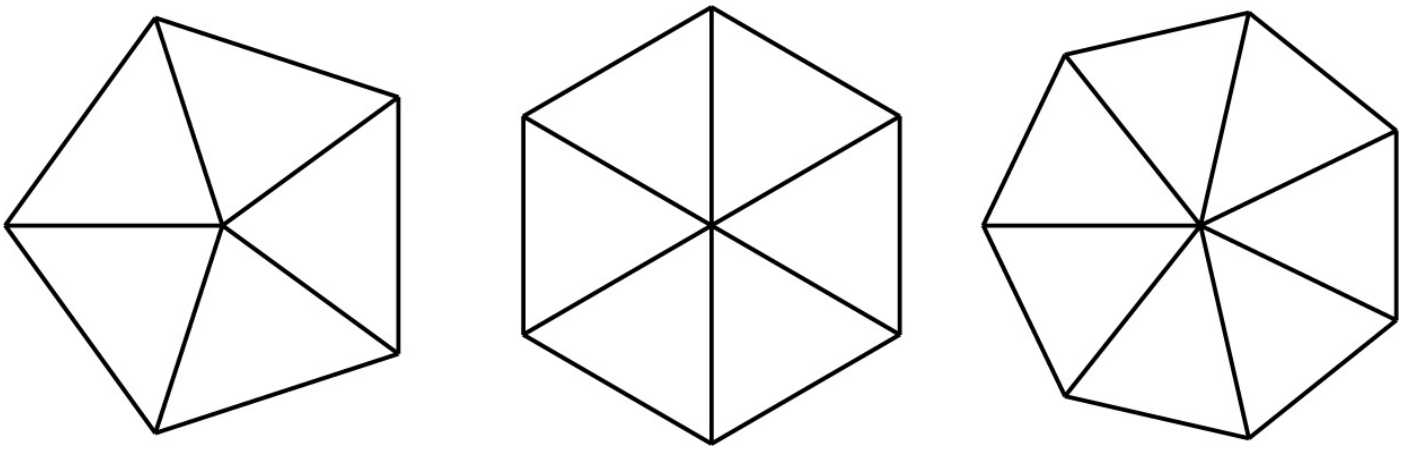


Figure 4-2. Turtle pies.

Solution: *http://thinkpython2.com/code/pie.py*.

Exercise 4-4.

The letters of the alphabet can be constructed from a moderate number of basic elements, like vertical and horizontal lines and a few curves. Design an alphabet that can be drawn with a minimal number of basic elements and then write functions that draw the letters.

You should write one function for each letter, with names `draw_a`, `draw_b`, etc., and put your functions in a file named `letters.py`. You can download a "turtle typewriter" from *http://thinkpython2.com/code/typewriter.py* to help you test your code.

You can get a solution from *http://thinkpython2.com/code/letters.py*; it also requires *http://thinkpython2.com/code/polygon.py*.

Exercise 4-5.

Read about spirals at *http://en.wikipedia.org/wiki/Spiral*; then write a program that draws an Archimedian spiral (or one of the other kinds).

Solution: *http://thinkpython2.com/code/spiral.py*.

Chapter 5. Conditionals and Recursion

The main topic of this chapter is the `if` statement, which executes different code depending on the state of the program. But first I want to introduce two new operators: floor division and modulus.

Floor Division and Modulus

The **floor division** operator, `//`, divides two numbers and rounds down to an integer. For example, suppose the run time of a movie is 105 minutes. You might want to know how long that is in hours. Conventional division returns a floating-point number:

```
>>> minutes = 105
>>> minutes / 60
1.75
```

But we don't normally write hours with decimal points. Floor division returns the integer number of hours, dropping the fraction part:

```
>>> minutes = 105
>>> hours = minutes // 60
>>> hours
1
```

To get the remainder, you could subtract off one hour in minutes:

```
>>> remainder = minutes - hours * 60
>>> remainder
45
```

An alternative is to use the **modulus operator**, `%`, which divides two numbers and returns the remainder:

```
>>> remainder = minutes % 60
>>> remainder
45
```

The modulus operator is more useful than it seems. For example, you can check whether one number is divisible by another — if `x % y` is zero, then `x` is divisible by `y`.

Also, you can extract the right-most digit or digits from a number. For example, `x % 10` yields the right-most digit of `x` (in base 10). Similarly `x % 100` yields the last two digits.

If you are using Python 2, division works differently. The division operator, `/`, performs floor division if both operands are integers, and floating-point division if either operand is a `float`.

Boolean Expressions

A **boolean expression** is an expression that is either true or false. The following examples use the operator ==, which compares two operands and produces `True` if they are equal and `False` otherwise:

```
>>> 5 == 5
True
>>> 5 == 6
False
```

`True` and `False` are special values that belong to the type `bool`; they are not strings:

```
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
```

The == operator is one of the **relational operators**; the others are:

```
x != y               # x is not equal to y
x > y                # x is greater than y
x < y                # x is less than y
x >= y               # x is greater than or equal to y
x <= y               # x is less than or equal to y
```

Although these operations are probably familiar to you, the Python symbols are different from the mathematical symbols. A common error is to use a single equal sign (=) instead of a double equal sign (==). Remember that = is an assignment operator and == is a relational operator. There is no such thing as =< or =>.

Logical Operators

There are three **logical operators**: `and`, `or`, and `not`. The semantics (meaning) of these operators is similar to their meaning in English. For example, `x > 0 and x < 10` is true only if `x` is greater than 0 *and* less than 10.

`n%2 == 0 or n%3 == 0` is true if *either or both* of the conditions is true, that is, if the number is divisible by 2 *or* 3.

Finally, the `not` operator negates a boolean expression, so `not (x > y)` is true if `x > y` is false, that is, if `x` is less than or equal to `y`.

Strictly speaking, the operands of the logical operators should be boolean expressions, but Python is not very strict. Any nonzero number is interpreted as `True`:

```
>>> 42 and True
True
```

This flexibility can be useful, but there are some subtleties to it that might be confusing. You might want to avoid it (unless you know what you are doing).

Conditional Execution

In order to write useful programs, we almost always need the ability to check conditions and change the behavior of the program accordingly. **Conditional statements** give us this ability. The simplest form is the `if` statement:

```
if x > 0:
    print('x is positive')
```

The boolean expression after `if` is called the **condition**. If it is true, the indented statement runs. If not, nothing happens.

`if` statements have the same structure as function definitions: a header followed by an indented body. Statements like this are called **compound statements**.

There is no limit on the number of statements that can appear in the body, but there has to be at least one. Occasionally, it is useful to have a body with no statements (usually as a place keeper for code you haven't written yet). In that case, you can use the `pass` statement, which does nothing.

```
if x < 0:
    pass          # TODO: need to handle negative values!
```

Alternative Execution

A second form of the `if` statement is “alternative execution”, in which there are two possibilities and the condition determines which one runs. The syntax looks like this:

```
if x % 2 == 0:
    print('x is even')
else:
    print('x is odd')
```

If the remainder when `x` is divided by 2 is 0, then we know that `x` is even, and the program displays an appropriate message. If the condition is false, the second set of statements runs. Since the condition must be true or false, exactly one of the alternatives will run. The alternatives are called **branches**, because they are branches in the flow of execution.

Chained Conditionals

Sometimes there are more than two possibilities and we need more than two branches. One way to express a computation like that is a **chained conditional**:

```
if x < y:
    print('x is less than y')
elif x > y:
    print('x is greater than y')
else:
    print('x and y are equal')
```

`elif` is an abbreviation of “else if”. Again, exactly one branch will run. There is no limit on the number of `elif` statements. If there is an `else` clause, it has to be at the end, but there doesn’t have to be one.

```
if choice == 'a':
    draw_a()
elif choice == 'b':
    draw_b()
elif choice == 'c':
    draw_c()
```

Each condition is checked in order. If the first is false, the next is checked, and so on. If one of them is true, the corresponding branch runs and the statement ends. Even if more than one condition is true, only the first true branch runs.

Nested Conditionals

One conditional can also be nested within another. We could have written the example in the previous section like this:

```
if x == y:
    print('x and y are equal')
else:
    if x < y:
        print('x is less than y')
    else:
        print('x is greater than y')
```

The outer conditional contains two branches. The first branch contains a simple statement. The second branch contains another `if` statement, which has two branches of its own. Those two branches are both simple statements, although they could have been conditional statements as well.

Although the indentation of the statements makes the structure apparent, **nested conditionals** become difficult to read very quickly. It is a good idea to avoid them when you can.

Logical operators often provide a way to simplify nested conditional statements. For example, we can rewrite the following code using a single conditional:

```
if 0 < x:
    if x < 10:
        print('x is a positive single-digit number.')
```

The `print` statement runs only if we make it past both conditionals, so we can get the same effect with the `and` operator:

```
if 0 < x and x < 10:
    print('x is a positive single-digit number.')
```

For this kind of condition, Python provides a more concise option:

```
if 0 < x < 10:
    print('x is a positive single-digit number.')
```

Recursion

It is legal for one function to call another; it is also legal for a function to call itself. It may not be obvious why that is a good thing, but it turns out to be one of the most magical things a program can do. For example, look at the following function:

```
def countdown(n):
    if n <= 0:
        print('Blastoff!')
    else:
        print(n)
        countdown(n-1)
```

If `n` is 0 or negative, it outputs the word, “Blastoff!” Otherwise, it outputs `n` and then calls a function named `countdown` — itself — passing `n-1` as an argument.

What happens if we call this function like this?

```
>>> countdown(3)
```

The execution of `countdown` begins with `n=3`, and since `n` is greater than 0, it outputs the value 3, and then calls itself…

> The execution of `countdown` begins with `n=2`, and since `n` is greater than 0, it outputs the value 2, and then calls itself…
>
> > The execution of `countdown` begins with `n=1`, and since `n` is greater than 0, it outputs the value 1, and then calls itself…
> >
> > > The execution of `countdown` begins with `n=0`, and since `n` is not greater than 0, it outputs the word, “Blastoff!” and then returns.
> >
> > The `countdown` that got `n=1` returns.
>
> The `countdown` that got `n=2` returns.

The `countdown` that got `n=3` returns.

And then you’re back in `__main__`. So, the total output looks like this:

```
3
2
1
Blastoff!
```

A function that calls itself is **recursive**; the process of executing it is called **recursion**.

As another example, we can write a function that prints a string `n` times:

```
def print_n(s, n):
    if n <= 0:
        return
    print(s)
    print_n(s, n-1)
```

If `n <= 0` the **return statement** exits the function. The flow of execution immediately

returns to the caller, and the remaining lines of the function don’t run.

The rest of the function is similar to `countdown`: it displays `s` and then calls itself to display `s` *n-1* additional times. So the number of lines of output is `1 + (n - 1)`, which adds up to `n`.

For simple examples like this, it is probably easier to use a `for` loop. But we will see examples later that are hard to write with a `for` loop and easy to write with recursion, so it is good to start early.

Stack Diagrams for Recursive Functions

In "Stack Diagrams", we used a stack diagram to represent the state of a program during a function call. The same kind of diagram can help interpret a recursive function.

Every time a function gets called, Python creates a frame to contain the function's local variables and parameters. For a recursive function, there might be more than one frame on the stack at the same time.

Figure 5-1 shows a stack diagram for `countdown` called with `n = 3`.

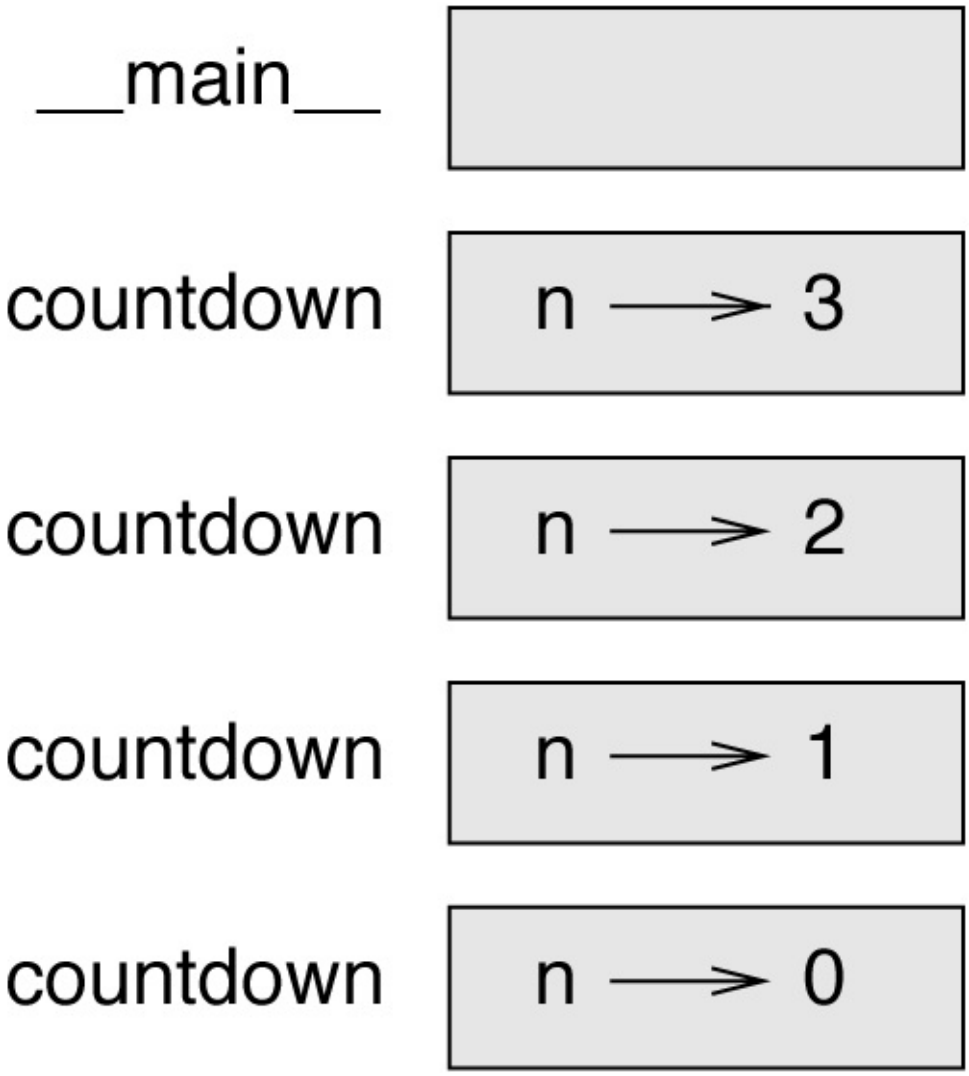


Figure 5-1. Stack diagram.

As usual, the top of the stack is the frame for `__main__`. It is empty because we did not create any variables in `__main__` or pass any arguments to it.

The four `countdown` frames have different values for the parameter `n`. The bottom of the stack, where `n=0`, is called the **base case**. It does not make a recursive call, so there are no more frames.

As an exercise, draw a stack diagram for `print_n` called with `s = 'Hello'` and `n=2`. Then write a function called `do_n` that takes a function object and a number, `n`, as arguments,

and that calls the given function `n` times.

Infinite Recursion

If a recursion never reaches a base case, it goes on making recursive calls forever, and the program never terminates. This is known as **infinite recursion**, and it is generally not a good idea. Here is a minimal program with an infinite recursion:

```
def recurse():
    recurse()
```

In most programming environments, a program with infinite recursion does not really run forever. Python reports an error message when the maximum recursion depth is reached:

```
  File "<stdin>", line 2, in recurse
  File "<stdin>", line 2, in recurse
  File "<stdin>", line 2, in recurse
                  .
                  .
                  .
  File "<stdin>", line 2, in recurse
RuntimeError: Maximum recursion depth exceeded
```

This traceback is a little bigger than the one we saw in the previous chapter. When the error occurs, there are 1,000 `recurse` frames on the stack!

If you write an infinite recursion by accident, review your function to confirm that there is a base case that does not make a recursive call. And if there is a base case, check whether you are guaranteed to reach it.

Keyboard Input

The programs we have written so far accept no input from the user. They just do the same thing every time.

Python provides a built-in function called `input` that stops the program and waits for the user to type something. When the user presses Return or Enter, the program resumes and `input` returns what the user typed as a string. In Python 2, the same function is called `raw_input`.

```
>>> text = input()
What are you waiting for?
>>> text
What are you waiting for?
```

Before getting input from the user, it is a good idea to print a prompt telling the user what to type. `input` can take a prompt as an argument:

```
>>> name = input('What…is your name?\n')
What…is your name?
Arthur, King of the Britons!
>>> name
Arthur, King of the Britons!
```

The sequence `\n` at the end of the prompt represents a **newline**, which is a special character that causes a line break. That's why the user's input appears below the prompt.

If you expect the user to type an integer, you can try to convert the return value to `int`:

```
>>> prompt = 'What…is the airspeed velocity of an unladen swallow?\n'
>>> speed = input(prompt)
What…is the airspeed velocity of an unladen swallow?
42
>>> int(speed)
42
```

But if the user types something other than a string of digits, you get an error:

```
>>> speed = input(prompt)
What…is the airspeed velocity of an unladen swallow?
What do you mean, an African or a European swallow?
>>> int(speed)
ValueError: invalid literal for int() with base 10
```

We will see how to handle this kind of error later.

Debugging

When a syntax or runtime error occurs, the error message contains a lot of information, but it can be overwhelming. The most useful parts are usually:

- What kind of error it was
- Where it occurred

Syntax errors are usually easy to find, but there are a few gotchas. Whitespace errors can be tricky because spaces and tabs are invisible and we are used to ignoring them.

```
>>> x = 5
>>>  y = 6
  File "<stdin>", line 1
    y = 6
    ^
IndentationError: unexpected indent
```

In this example, the problem is that the second line is indented by one space. But the error message points to `y`, which is misleading. In general, error messages indicate where the problem was discovered, but the actual error might be earlier in the code, sometimes on a previous line.

The same is true of runtime errors. Suppose you are trying to compute a signal-to-noise ratio in decibels. The formula is $SNR_{db} = 10\log_{10}(P_{signal}/P_{noise})$. In Python, you might write something like this:

```
import math
signal_power = 9
noise_power = 10
ratio = signal_power // noise_power
decibels = 10 * math.log10(ratio)
print(decibels)
```

When you run this program, you get an exception:

```
Traceback (most recent call last):
  File "snr.py", line 5, in ?
    decibels = 10 * math.log10(ratio)
ValueError: math domain error
```

The error message indicates line 5, but there is nothing wrong with that line. To find the real error, it might be useful to print the value of `ratio`, which turns out to be 0. The problem is in line 4, which uses floor division instead of floating-point division.

You should take the time to read error messages carefully, but don’t assume that everything they say is correct.

Glossary

floor division:

An operator, denoted `//`, that divides two numbers and rounds down (toward zero) to an integer.

modulus operator:

An operator, denoted with a percent sign (`%`), that works on integers and returns the remainder when one number is divided by another.

boolean expression:

An expression whose value is either `True` or `False`.

relational operator:

One of the operators that compares its operands: `==`, `!=`, `>`, `<`, `>=`, and `<=`.

logical operator:

One of the operators that combines boolean expressions: `and`, `or`, and `not`.

conditional statement:

A statement that controls the flow of execution depending on some condition.

condition:

The boolean expression in a conditional statement that determines which branch runs.

compound statement:

A statement that consists of a header and a body. The header ends with a colon (:). The body is indented relative to the header.

branch:

One of the alternative sequences of statements in a conditional statement.

chained conditional:

A conditional statement with a series of alternative branches.

nested conditional:

A conditional statement that appears in one of the branches of another conditional statement.

return statement:

A statement that causes a function to end immediately and return to the caller.

recursion:

The process of calling the function that is currently executing.

base case:

A conditional branch in a recursive function that does not make a recursive call.

infinite recursion:

A recursion that doesn’t have a base case, or never reaches it. Eventually, an infinite recursion causes a runtime error.

Exercises

Exercise 5-1.

The `time` module provides a function, also named `time`, that returns the current Greenwich Mean Time in "the epoch", which is an arbitrary time used as a reference point. On UNIX systems, the epoch is 1 January 1970.

```
>>> import time
>>> time.time()
1437746094.5735958
```

Write a script that reads the current time and converts it to a time of day in hours, minutes, and seconds, plus the number of days since the epoch.

Exercise 5-2.

Fermat's Last Theorem says that there are no positive integers *a*, *b*, and *c* such that

$$a^n + b^n = c^n$$

for any values of *n* greater than 2.

1. Write a function named `check_fermat` that takes four parameters — `a`, `b`, `c` and `n` — and checks to see if Fermat's theorem holds. If *n* is greater than 2 and

   $$a^n + b^n = c^n$$

   the program should print, "Holy smokes, Fermat was wrong!" Otherwise the program should print, "No, that doesn't work."

2. Write a function that prompts the user to input values for `a`, `b`, `c` and `n`, converts them to integers, and uses `check_fermat` to check whether they violate Fermat's theorem.

Exercise 5-3.

If you are given three sticks, you may or may not be able to arrange them in a triangle. For example, if one of the sticks is 12 inches long and the other two are one inch long, you will not be able to get the short sticks to meet in the middle. For any three lengths, there is a simple test to see if it is possible to form a triangle:

> If any of the three lengths is greater than the sum of the other two, then you cannot form a triangle. Otherwise, you can. (If the sum of two lengths equals the third, they form what is called a "degenerate" triangle.)

1. Write a function named `is_triangle` that takes three integers as arguments, and that prints either "Yes" or "No", depending on whether you can or cannot form a triangle from sticks with the given lengths.

2. Write a function that prompts the user to input three stick lengths, converts them to

integers, and uses `is_triangle` to check whether sticks with the given lengths can form a triangle.

Exercise 5-4.

What is the output of the following program? Draw a stack diagram that shows the state of the program when it prints the result.

```
def recurse(n, s):
    if n == 0:
        print(s)
    else:
        recurse(n-1, n+s)

recurse(3, 0)
```

1. What would happen if you called this function like this: `recurse(-1, 0)`?

2. Write a docstring that explains everything someone would need to know in order to use this function (and nothing else).

The following exercises use the `turtle` module, described in Chapter 4:

Exercise 5-5.

Read the following function and see if you can figure out what it does (see the examples in Chapter 4). Then run it and see if you got it right.

```
def draw(t, length, n):
    if n == 0:
        return
    angle = 50
    t.fd(length*n)
    t.lt(angle)
    draw(t, length, n-1)
    t.rt(2*angle)
    draw(t, length, n-1)
    t.lt(angle)
    t.bk(length*n)
```

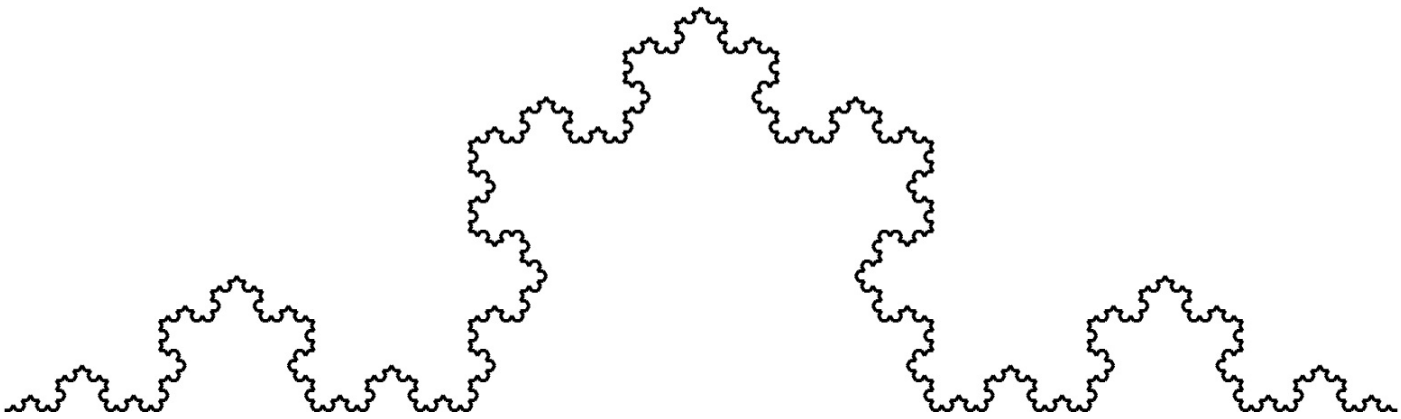


Figure 5-2. A Koch curve.

Exercise 5-6.

The Koch curve is a fractal that looks something like Figure 5-2. To draw a Koch curve with length *x*, all you have to do is:

1. Draw a Koch curve with length *x/3*.

2. Turn left 60 degrees.

3. Draw a Koch curve with length *x/3*.

4. Turn right 120 degrees.

5. Draw a Koch curve with length *x/3*.

6. Turn left 60 degrees.

7. Draw a Koch curve with length *x/3*.

The exception is if *x* is less than 3: in that case, you can just draw a straight line with length *x*.

1. Write a function called `koch` that takes a turtle and a length as parameters, and that uses the turtle to draw a Koch curve with the given length.

2. Write a function called `snowflake` that draws three Koch curves to make the outline of a snowflake.
   Solution: *http://thinkpython2.com/code/koch.py*.

3. The Koch curve can be generalized in several ways. See *http://en.wikipedia.org/wiki/Koch_snowflake* for examples and implement your favorite.

Chapter 6. Fruitful Functions

Many of the Python functions we have used, such as the math functions, produce return values. But the functions we've written are all void: they have an effect, like printing a value or moving a turtle, but they don't have a return value. In this chapter you will learn to write fruitful functions.

Return Values

Calling the function generates a return value, which we usually assign to a variable or use as part of an expression.

```
e = math.exp(1.0)
height = radius * math.sin(radians)
```

The functions we have written so far are void. Speaking casually, they have no return value; more precisely, their return value is `None`.

In this chapter, we are (finally) going to write fruitful functions. The first example is `area`, which returns the area of a circle with the given radius:

```
def area(radius):
    a = math.pi * radius**2
    return a
```

We have seen the `return` statement before, but in a fruitful function the `return` statement includes an expression. This statement means: “Return immediately from this function and use the following expression as a return value.” The expression can be arbitrarily complicated, so we could have written this function more concisely:

```
def area(radius):
    return math.pi * radius**2
```

On the other hand, **temporary variables** like a can make debugging easier.

Sometimes it is useful to have multiple return statements, one in each branch of a conditional:

```
def absolute_value(x):
    if x < 0:
        return -x
    else:
        return x
```

Since these `return` statements are in an alternative conditional, only one runs.

As soon as a return statement runs, the function terminates without executing any subsequent statements. Code that appears after a `return` statement, or any other place the flow of execution can never reach, is called **dead code**.

In a fruitful function, it is a good idea to ensure that every possible path through the program hits a `return` statement. For example:

```
def absolute_value(x):
    if x < 0:
        return -x
    if x > 0:
        return x
```

This function is incorrect because if `x` happens to be 0, neither condition is true, and the function ends without hitting a `return` statement. If the flow of execution gets to the end of a function, the return value is `None`, which is not the absolute value of 0:

```
>>> absolute_value(0)
None
```

By the way, Python provides a built-in function called `abs` that computes absolute values.

As an exercise, write a `compare` function takes two values, `x` and `y`, and returns `1` if `x > y`, `0` if `x == y`, and `-1` if `x < y`.

Incremental Development

As you write larger functions, you might find yourself spending more time debugging.

To deal with increasingly complex programs, you might want to try a process called **incremental development**. The goal of incremental development is to avoid long debugging sessions by adding and testing only a small amount of code at a time.

As an example, suppose you want to find the distance between two points, given by the coordinates (x_1, y_1) and (x_2, y_2). By the Pythagorean theorem, the distance is:

$$\text{distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

The first step is to consider what a `distance` function should look like in Python. In other words, what are the inputs (parameters) and what is the output (return value)?

In this case, the inputs are two points, which you can represent using four numbers. The return value is the distance represented by a floating-point value.

Immediately you can write an outline of the function:

```
def distance(x1, y1, x2, y2):
    return 0.0
```

Obviously, this version doesn't compute distances; it always returns zero. But it is syntactically correct, and it runs, which means that you can test it before you make it more complicated.

To test the new function, call it with sample arguments:

```
>>> distance(1, 2, 4, 6)
0.0
```

I chose these values so that the horizontal distance is 3 and the vertical distance is 4; that way, the result is 5, the hypotenuse of a 3-4-5 triangle. When testing a function, it is useful to know the right answer.

At this point we have confirmed that the function is syntactically correct, and we can start adding code to the body. A reasonable next step is to find the differences $x_2 - x_1$ and $y_2 - y_1$. The next version stores those values in temporary variables and prints them:

```
def distance(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    print('dx is', dx)
    print('dy is', dy)
    return 0.0
```

If the function is working, it should display `dx is 3` and `dy is 4`. If so, we know that the function is getting the right arguments and performing the first computation correctly. If

not, there are only a few lines to check.

Next we compute the sum of squares of `dx` and `dy`:

```
def distance(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    dsquared = dx**2 + dy**2
    print('dsquared is: ', dsquared)
    return 0.0
```

Again, you would run the program at this stage and check the output (which should be 25). Finally, you can use `math.sqrt` to compute and return the result:

```
def distance(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    dsquared = dx**2 + dy**2
    result = math.sqrt(dsquared)
    return result
```

If that works correctly, you are done. Otherwise, you might want to print the value of `result` before the return statement.

The final version of the function doesn't display anything when it runs; it only returns a value. The `print` statements we wrote are useful for debugging, but once you get the function working, you should remove them. Code like that is called **scaffolding** because it is helpful for building the program but is not part of the final product.

When you start out, you should add only a line or two of code at a time. As you gain more experience, you might find yourself writing and debugging bigger chunks. Either way, incremental development can save you a lot of debugging time.

The key aspects of the process are:

1. Start with a working program and make small incremental changes. At any point, if there is an error, you should have a good idea where it is.

2. Use variables to hold intermediate values so you can display and check them.

3. Once the program is working, you might want to remove some of the scaffolding or consolidate multiple statements into compound expressions, but only if it does not make the program difficult to read.

As an exercise, use incremental development to write a function called `hypotenuse` that returns the length of the hypotenuse of a right triangle given the lengths of the other two legs as arguments. Record each stage of the development process as you go.

Composition

As you should expect by now, you can call one function from within another. As an example, we'll write a function that takes two points, the center of the circle and a point on the perimeter, and computes the area of the circle.

Assume that the center point is stored in the variables `xc` and `yc`, and the perimeter point is in `xp` and `yp`. The first step is to find the radius of the circle, which is the distance between the two points. We just wrote a function, `distance`, that does that:

```
radius = distance(xc, yc, xp, yp)
```

The next step is to find the area of a circle with that radius; we just wrote that, too:

```
result = area(radius)
```

Encapsulating these steps in a function, we get:

```
def circle_area(xc, yc, xp, yp):
    radius = distance(xc, yc, xp, yp)
    result = area(radius)
    return result
```

The temporary variables `radius` and `result` are useful for development and debugging, but once the program is working, we can make it more concise by composing the function calls:

```
def circle_area(xc, yc, xp, yp):
    return area(distance(xc, yc, xp, yp))
```

Boolean Functions

Functions can return booleans, which is often convenient for hiding complicated tests inside functions. For example:

```
def is_divisible(x, y):
    if x % y == 0:
        return True
    else:
        return False
```

It is common to give boolean functions names that sound like yes/no questions; `is_divisible` returns either `True` or `False` to indicate whether `x` is divisible by `y`.

Here is an example:

```
>>> is_divisible(6, 4)
False
>>> is_divisible(6, 3)
True
```

The result of the == operator is a boolean, so we can write the function more concisely by returning it directly:

```
def is_divisible(x, y):
    return x % y == 0
```

Boolean functions are often used in conditional statements:

```
if is_divisible(x, y):
    print('x is divisible by y')
```

It might be tempting to write something like:

```
if is_divisible(x, y) == True:
    print('x is divisible by y')
```

But the extra comparison is unnecessary.

As an exercise, write a function `is_between(x, y, z)` that returns `True` if $x \leq y \leq z$ or `False` otherwise.

More Recursion

We have only covered a small subset of Python, but you might be interested to know that this subset is a *complete* programming language, which means that anything that can be computed can be expressed in this language. Any program ever written could be rewritten using only the language features you have learned so far (actually, you would need a few commands to control devices like the mouse, disks, etc., but that's all).

Proving that claim is a nontrivial exercise first accomplished by Alan Turing, one of the first computer scientists (some would argue that he was a mathematician, but a lot of early computer scientists started as mathematicians). Accordingly, it is known as the Turing Thesis. For a more complete (and accurate) discussion of the Turing Thesis, I recommend Michael Sipser's book *Introduction to the Theory of Computation* (Course Technology, 2012).

To give you an idea of what you can do with the tools you have learned so far, we'll evaluate a few recursively defined mathematical functions. A recursive definition is similar to a circular definition, in the sense that the definition contains a reference to the thing being defined. A truly circular definition is not very useful:

vorpal:

> An adjective used to describe something that is vorpal.

If you saw that definition in the dictionary, you might be annoyed. On the other hand, if you looked up the definition of the factorial function, denoted with the symbol *!*, you might get something like this:

$$0! = 1$$

$$n! = n(n-1)!$$

This definition says that the factorial of 0 is 1, and the factorial of any other value, *n*, is *n* multiplied by the factorial of *n-1*.

So *3!* is 3 times *2!*, which is 2 times *1!*, which is 1 times *0!*. Putting it all together, *3!* equals 3 times 2 times 1 times 1, which is 6.

If you can write a recursive definition of something, you can write a Python program to evaluate it. The first step is to decide what the parameters should be. In this case it should be clear that `factorial` takes an integer:

```
def factorial(n):
```

If the argument happens to be 0, all we have to do is return 1:

```
def factorial(n):
```

```
    if n == 0:
        return 1
```

Otherwise, and this is the interesting part, we have to make a recursive call to find the factorial of *n-1* and then multiply it by *n*:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        recurse = factorial(n-1)
        result = n * recurse
        return result
```

The flow of execution for this program is similar to the flow of `countdown` in "Recursion". If we call `factorial` with the value 3:

Since 3 is not 0, we take the second branch and calculate the factorial of `n-1`…

Since 2 is not 0, we take the second branch and calculate the factorial of `n-1`…

Since 1 is not 0, we take the second branch and calculate the factorial of `n-1`…

Since 0 equals 0, we take the first branch and return 1 without making any more recursive calls.

The return value, 1, is multiplied by *n*, which is 1, and the result is returned.

The return value, 1, is multiplied by *n*, which is 2, and the result is returned.

The return value (2) is multiplied by *n*, which is 3, and the result, 6, becomes the return value of the function call that started the whole process.

Figure 6-1 shows what the stack diagram looks like for this sequence of function calls.

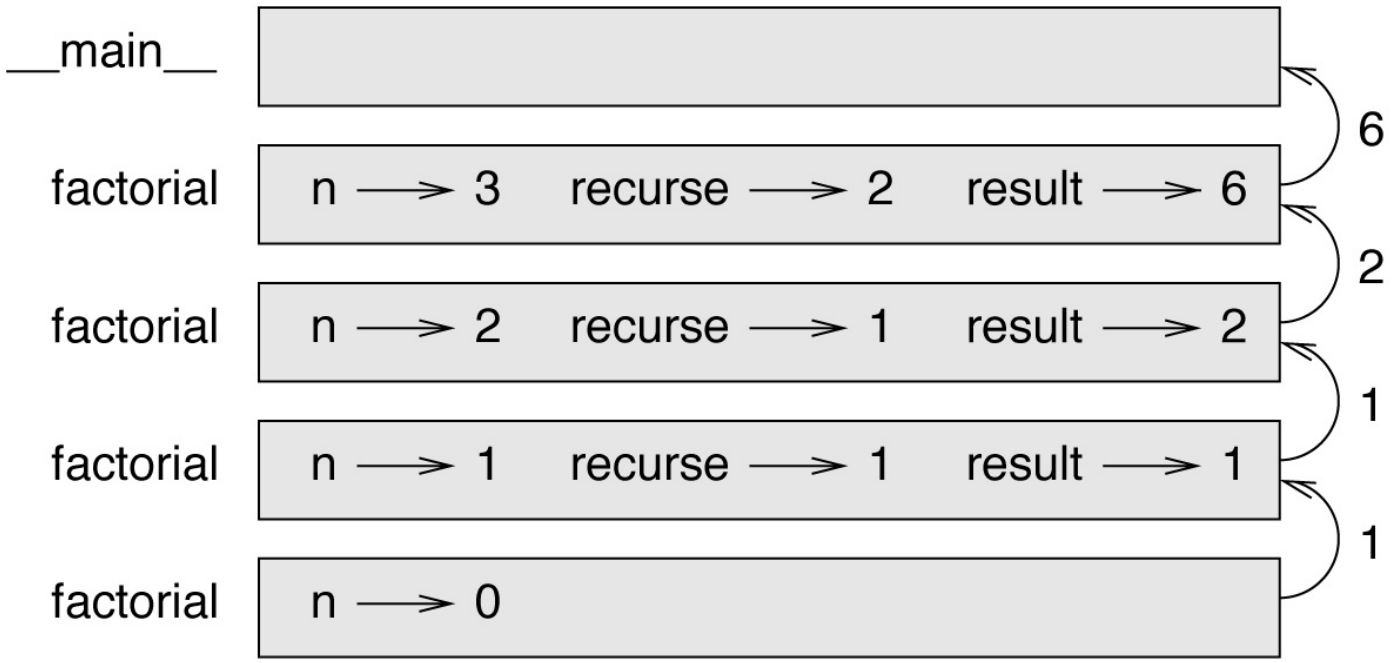


Figure 6-1. Stack diagram.

The return values are shown being passed back up the stack. In each frame, the return value is the value of `result`, which is the product of `n` and `recurse`.

In the last frame, the local variables `recurse` and `result` do not exist, because the branch that creates them does not run.

Leap of Faith

Following the flow of execution is one way to read programs, but it can quickly become overwhelming. An alternative is what I call the "leap of faith". When you come to a function call, instead of following the flow of execution, you *assume* that the function works correctly and returns the right result.

In fact, you are already practicing this leap of faith when you use built-in functions. When you call `math.cos` or `math.exp`, you don't examine the bodies of those functions. You just assume that they work because the people who wrote the built-in functions were good programmers.

The same is true when you call one of your own functions. For example, in "Boolean Functions", we wrote a function called `is_divisible` that determines whether one number is divisible by another. Once we have convinced ourselves that this function is correct — by examining the code and testing — we can use the function without looking at the body again.

The same is true of recursive programs. When you get to the recursive call, instead of following the flow of execution, you should assume that the recursive call works (returns the correct result) and then ask yourself, "Assuming that I can find the factorial of *n-1*, can I compute the factorial of *n*?" It is clear that you can, by multiplying by *n*.

Of course, it's a bit strange to assume that the function works correctly when you haven't finished writing it, but that's why it's called a leap of faith!

One More Example

After `factorial`, the most common example of a recursively defined mathematical function is `fibonacci`, which has the following definition (see *http://en.wikipedia.org/wiki/Fibonacci_number*):

$$\text{fibonacci}(0) = 0$$

$$\text{fibonacci}(1) = 1$$

$$\text{fibonacci}(n) = \text{fibonacci}(n-1) + \text{fibonacci}(n-2)$$

Translated into Python, it looks like this:

```
def fibonacci (n):
    if n == 0:
        return 0
    elif  n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)
```

If you try to follow the flow of execution here, even for fairly small values of *n*, your head explodes. But according to the leap of faith, if you assume that the two recursive calls work correctly, then it is clear that you get the right result by adding them together.

Checking Types

What happens if we call `factorial` and give it 1.5 as an argument?

```
>>> factorial(1.5)
RuntimeError: Maximum recursion depth exceeded
```

It looks like an infinite recursion. How can that be? The function has a base case — when `n == 0`. But if `n` is not an integer, we can *miss* the base case and recurse forever.

In the first recursive call, the value of `n` is 0.5. In the next, it is -0.5. From there, it gets smaller (more negative), but it will never be 0.

We have two choices. We can try to generalize the `factorial` function to work with floating-point numbers, or we can make `factorial` check the type of its argument. The first option is called the gamma function and it's a little beyond the scope of this book. So we'll go for the second.

We can use the built-in function `isinstance` to verify the type of the argument. While we're at it, we can also make sure the argument is positive:

```
def factorial (n):
    if not isinstance(n, int):
        print('Factorial is only defined for integers.')
        return None
    elif n < 0:
        print('Factorial is not defined for negative integers.')
        return None
    elif n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

The first base case handles nonintegers; the second handles negative integers. In both cases, the program prints an error message and returns `None` to indicate that something went wrong:

```
>>> factorial('fred')
Factorial is only defined for integers.
None
>>> factorial(-2)
Factorial is not defined for negative integers.
None
```

If we get past both checks, we know that *n* is positive or zero, so we can prove that the recursion terminates.

This program demonstrates a pattern sometimes called a **guardian**. The first two conditionals act as guardians, protecting the code that follows from values that might cause an error. The guardians make it possible to prove the correctness of the code.

In "Reverse Lookup" we will see a more flexible alternative to printing an error message: raising an exception.

Debugging

Breaking a large program into smaller functions creates natural checkpoints for debugging. If a function is not working, there are three possibilities to consider:

- There is something wrong with the arguments the function is getting; a precondition is violated.
- There is something wrong with the function; a postcondition is violated.
- There is something wrong with the return value or the way it is being used.

To rule out the first possibility, you can add a `print` statement at the beginning of the function and display the values of the parameters (and maybe their types). Or you can write code that checks the preconditions explicitly.

If the parameters look good, add a `print` statement before each `return` statement and display the return value. If possible, check the result by hand. Consider calling the function with values that make it easy to check the result (as in “Incremental Development”).

If the function seems to be working, look at the function call to make sure the return value is being used correctly (or used at all!).

Adding print statements at the beginning and end of a function can help make the flow of execution more visible. For example, here is a version of `factorial` with print statements:

```
def factorial(n):
    space = ' ' * (4 * n)
    print(space, 'factorial', n)
    if n == 0:
        print(space, 'returning 1')
        return 1
    else:
        recurse = factorial(n-1)
        result = n * recurse
        print(space, 'returning', result)
        return result
```

`space` is a string of space characters that controls the indentation of the output. Here is the result of `factorial(4)`:

```
                factorial 4
            factorial 3
        factorial 2
    factorial 1
factorial 0
returning 1
    returning 1
        returning 2
            returning 6
                returning 24
```

If you are confused about the flow of execution, this kind of output can be helpful. It takes some time to develop effective scaffolding, but a little bit of scaffolding can save a lot of

debugging.

Glossary

temporary variable:

A variable used to store an intermediate value in a complex calculation.

dead code:

Part of a program that can never run, often because it appears after a `return` statement.

incremental development:

A program development plan intended to avoid debugging by adding and testing only a small amount of code at a time.

scaffolding:

Code that is used during program development but is not part of the final version.

guardian:

A programming pattern that uses a conditional statement to check for and handle circumstances that might cause an error.

Exercises

Exercise 6-1.

Draw a stack diagram for the following program. What does the program print?

```
def b(z):
    prod = a(z, z)
    print(z, prod)
    return prod

def a(x, y):
    x = x + 1
    return x * y

def c(x, y, z):
    total = x + y + z
    square = b(total)**2
    return square

x = 1
y = x + 1
print(c(x, y+3, x+y))
```

Exercise 6-2.

The Ackermann function, $A(m, n)$, is defined:

$$A(m, n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1, 1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1, A(m, n-1)) & \text{if } m > 0 \text{ and } n > 0. \end{cases}$$

See *http://en.wikipedia.org/wiki/Ackermann_function*. Write a function named `ack` that evaluates the Ackermann function. Use your function to evaluate `ack(3, 4)`, which should be 125. What happens for larger values of `m` and `n`?

Solution: *http://thinkpython2.com/code/ackermann.py*.

Exercise 6-3.

A palindrome is a word that is spelled the same backward and forward, like “noon” and “redivider”. Recursively, a word is a palindrome if the first and last letters are the same and the middle is a palindrome.

The following are functions that take a string argument and return the first, last, and middle letters:

```
def first(word):
    return word[0]

def last(word):
    return word[-1]

def middle(word):
    return word[1:-1]
```

We’ll see how they work in Chapter 8.

1. Type these functions into a file named `palindrome.py` and test them out. What happens if you call `middle` with a string with two letters? One letter? What about the

empty string, which is written `''` and contains no letters?

2. Write a function called `is_palindrome` that takes a string argument and returns `True` if it is a palindrome and `False` otherwise. Remember that you can use the built-in function `len` to check the length of a string.

Solution: *http://thinkpython2.com/code/palindrome_soln.py*.

Exercise 6-4.

A number, *a*, is a power of *b* if it is divisible by *b* and *a/b* is a power of *b*. Write a function called `is_power` that takes parameters `a` and `b` and returns `True` if `a` is a power of `b`. Note: you will have to think about the base case.

Exercise 6-5.

The greatest common divisor (GCD) of *a* and *b* is the largest number that divides both of them with no remainder.

One way to find the GCD of two numbers is based on the observation that if *r* is the remainder when *a* is divided by *b*, then $gcd(a, b) = gcd(b, r)$. As a base case, we can use $gcd(a, 0) = a$.

Write a function called `gcd` that takes parameters `a` and `b` and returns their greatest common divisor.

Credit: This exercise is based on an example from Abelson and Sussman's *Structure and Interpretation of Computer Programs* (MIT Press, 1996).

Chapter 7. Iteration

This chapter is about iteration, which is the ability to run a block of statements repeatedly. We saw a kind of iteration, using recursion, in “Recursion”. We saw another kind, using a `for` loop, in “Simple Repetition”. In this chapter we’ll see yet another kind, using a `while` statement. But first I want to say a little more about variable assignment.

Reassignment

As you may have discovered, it is legal to make more than one assignment to the same variable. A new assignment makes an existing variable refer to a new value (and stop referring to the old value).

```
>>> x = 5
>>> x
5
>>> x = 7
>>> x
7
```

The first time we display `x`, its value is 5; the second time, its value is 7.

Figure 7-1 shows what **reassignment** looks like in a state diagram.

At this point I want to address a common source of confusion. Because Python uses the equal sign (=) for assignment, it is tempting to interpret a statement like `a = b` as a mathematical proposition of equality; that is, the claim that `a` and `b` are equal. But this interpretation is wrong.

First, equality is a symmetric relationship and assignment is not. For example, in mathematics, if *a=7* then *7=a*. But in Python, the statement `a = 7` is legal and `7 = a` is not.

Also, in mathematics, a proposition of equality is either true or false for all time. If *a=b* now, then *a* will always equal *b*. In Python, an assignment statement can make two variables equal, but they don't have to stay that way:

```
>>> a = 5
>>> b = a    # a and b are now equal
>>> a = 3    # a and b are no longer equal
>>> b
5
```

The third line changes the value of `a` but does not change the value of `b`, so they are no longer equal.

Reassigning variables is often useful, but you should use it with caution. If the values of variables change frequently, it can make the code difficult to read and debug.

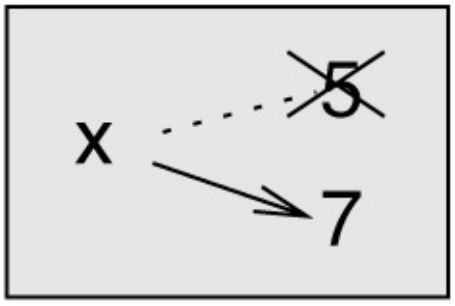


Figure 7-1. State diagram.

Updating Variables

A common kind of reassignment is an **update**, where the new value of the variable depends on the old.

```
>>> x = x + 1
```

This means “get the current value of `x`, add one, and then update `x` with the new value.”

If you try to update a variable that doesn’t exist, you get an error, because Python evaluates the right side before it assigns a value to `x`:

```
>>> x = x + 1
NameError: name 'x' is not defined
```

Before you can update a variable, you have to **initialize** it, usually with a simple assignment:

```
>>> x = 0
>>> x = x + 1
```

Updating a variable by adding 1 is called an **increment**; subtracting 1 is called a **decrement**.

The while Statement

Computers are often used to automate repetitive tasks. Repeating identical or similar tasks without making errors is something that computers do well and people do poorly. In a computer program, repetition is also called **iteration**.

We have already seen two functions, `countdown` and `print_n`, that iterate using recursion. Because iteration is so common, Python provides language features to make it easier. One is the `for` statement we saw in "Simple Repetition". We'll get back to that later.

Another is the `while` statement. Here is a version of `countdown` that uses a `while` statement:

```
def countdown(n):
    while n > 0:
        print(n)
        n = n - 1
    print('Blastoff!')
```

You can almost read the `while` statement as if it were English. It means, "While `n` is greater than 0, display the value of `n` and then decrement `n`. When you get to 0, display the word `Blastoff!`"

More formally, here is the flow of execution for a `while` statement:

1. Determine whether the condition is true or false.
2. If false, exit the `while` statement and continue execution at the next statement.
3. If the condition is true, run the body and then go back to step 1.

This type of flow is called a loop because the third step loops back around to the top.

The body of the loop should change the value of one or more variables so that the condition becomes false eventually and the loop terminates. Otherwise the loop will repeat forever, which is called an **infinite loop**. An endless source of amusement for computer scientists is the observation that the directions on shampoo, "Lather, rinse, repeat", are an infinite loop.

In the case of `countdown`, we can prove that the loop terminates: if `n` is zero or negative, the loop never runs. Otherwise, `n` gets smaller each time through the loop, so eventually we have to get to 0.

For some other loops, it is not so easy to tell. For example:

```
def sequence(n):
    while n != 1:
        print(n)
        if n % 2 == 0:        # n is even
            n = n / 2
        else:                 # n is odd
            n = n*3 + 1
```

The condition for this loop is `n != 1`, so the loop will continue until `n` is `1`, which makes the condition false.

Each time through the loop, the program outputs the value of `n` and then checks whether it is even or odd. If it is even, `n` is divided by 2. If it is odd, the value of `n` is replaced with `n*3 + 1`. For example, if the argument passed to `sequence` is 3, the resulting values of `n` are 3, 10, 5, 16, 8, 4, 2, 1.

Since `n` sometimes increases and sometimes decreases, there is no obvious proof that `n` will ever reach 1, or that the program terminates. For some particular values of `n`, we can prove termination. For example, if the starting value is a power of two, `n` will be even every time through the loop until it reaches 1. The previous example ends with such a sequence, starting with 16.

The hard question is whether we can prove that this program terminates for *all* positive values of `n`. So far, no one has been able to prove it *or* disprove it! (See *http://en.wikipedia.org/wiki/Collatz_conjecture*.)

As an exercise, rewrite the function `print_n` from “Recursion” using iteration instead of recursion.

break

Sometimes you don’t know it’s time to end a loop until you get halfway through the body. In that case you can use the `break` statement to jump out of the loop.

For example, suppose you want to take input from the user until they type `done`. You could write:

```
while True:
    line = input('> ')
    if line == 'done':
        break
    print(line)

print('Done!')
```

The loop condition is `True`, which is always true, so the loop runs until it hits the break statement.

Each time through, it prompts the user with an angle bracket. If the user types `done`, the `break` statement exits the loop. Otherwise the program echoes whatever the user types and goes back to the top of the loop. Here’s a sample run:

```
> not done
not done
> done
Done!
```

This way of writing `while` loops is common because you can check the condition anywhere in the loop (not just at the top) and you can express the stop condition affirmatively (“stop when this happens”) rather than negatively (“keep going until that happens”).

Square Roots

Loops are often used in programs that compute numerical results by starting with an approximate answer and iteratively improving it.

For example, one way of computing square roots is Newton's method. Suppose that you want to know the square root of *a*. If you start with almost any estimate, *x*, you can compute a better estimate with the following formula:

$$y = \frac{x + a / x}{2}$$

For example, if *a* is 4 and *x* is 3:

```
>>> a = 4
>>> x = 3
>>> y = (x + a/x) / 2
>>> y
2.16666666667
```

The result is closer to the correct answer ($\sqrt{4} = 2$). If we repeat the process with the new estimate, it gets even closer:

```
>>> x = y
>>> y = (x + a/x) / 2
>>> y
2.00641025641
```

After a few more updates, the estimate is almost exact:

```
>>> x = y
>>> y = (x + a/x) / 2
>>> y
2.00001024003
>>> x = y
>>> y = (x + a/x) / 2
>>> y
2.00000000003
```

In general we don't know ahead of time how many steps it takes to get to the right answer, but we know when we get there because the estimate stops changing:

```
>>> x = y
>>> y = (x + a/x) / 2
>>> y
2.0
>>> x = y
>>> y = (x + a/x) / 2
>>> y
2.0
```

When `y == x`, we can stop. Here is a loop that starts with an initial estimate, `x`, and improves it until it stops changing:

```
while True:
    print(x)
    y = (x + a/x) / 2
    if y == x:
        break
    x = y
```

For most values of `a` this works fine, but in general it is dangerous to test `float` equality. Floating-point values are only approximately right: most rational numbers, like *1/3*, and irrational numbers, like $\sqrt{2}$, can't be represented exactly with a `float`.

Rather than checking whether `x` and `y` are exactly equal, it is safer to use the built-in function `abs` to compute the absolute value, or magnitude, of the difference between them:

```
    if abs(y-x) < epsilon:
        break
```

Where `epsilon` has a value, like `0.0000001`, that determines how close is close enough.

Algorithms

Newton's method is an example of an **algorithm**: it is a mechanical process for solving a category of problems (in this case, computing square roots).

To understand what an algorithm is, it might help to start with something that is not an algorithm. When you learned to multiply single-digit numbers, you probably memorized the multiplication table. In effect, you memorized 100 specific solutions. That kind of knowledge is not algorithmic.

But if you were "lazy", you might have learned a few tricks. For example, to find the product of *n* and 9, you can write *n-1* as the first digit and *10-n* as the second digit. This trick is a general solution for multiplying any single-digit number by 9. That's an algorithm!

Similarly, the techniques you learned for addition with carrying, subtraction with borrowing, and long division are all algorithms. One of the characteristics of algorithms is that they do not require any intelligence to carry out. They are mechanical processes where each step follows from the last according to a simple set of rules.

Executing algorithms is boring, but designing them is interesting, intellectually challenging, and a central part of computer science.

Some of the things that people do naturally, without difficulty or conscious thought, are the hardest to express algorithmically. Understanding natural language is a good example. We all do it, but so far no one has been able to explain *how* we do it, at least not in the form of an algorithm.

Debugging

As you start writing bigger programs, you might find yourself spending more time debugging. More code means more chances to make an error and more places for bugs to hide.

One way to cut your debugging time is “debugging by bisection”. For example, if there are 100 lines in your program and you check them one at a time, it would take 100 steps.

Instead, try to break the problem in half. Look at the middle of the program, or near it, for an intermediate value you can check. Add a `print` statement (or something else that has a verifiable effect) and run the program.

If the mid-point check is incorrect, there must be a problem in the first half of the program. If it is correct, the problem is in the second half.

Every time you perform a check like this, you halve the number of lines you have to search. After six steps (which is fewer than 100), you would be down to one or two lines of code, at least in theory.

In practice it is not always clear what the “middle of the program” is and not always possible to check it. It doesn’t make sense to count lines and find the exact midpoint. Instead, think about places in the program where there might be errors and places where it is easy to put a check. Then choose a spot where you think the chances are about the same that the bug is before or after the check.

Glossary

reassignment:

Assigning a new value to a variable that already exists.

update:

An assignment where the new value of the variable depends on the old.

initialization:

An assignment that gives an initial value to a variable that will be updated.

increment:

An update that increases the value of a variable (often by one).

decrement:

An update that decreases the value of a variable.

iteration:

Repeated execution of a set of statements using either a recursive function call or a loop.

infinite loop:

A loop in which the terminating condition is never satisfied.

algorithm:

A general process for solving a category of problems.

Exercises

Exercise 7-1.

Copy the loop from "Square Roots" and encapsulate it in a function called `mysqrt` that takes `a` as a parameter, chooses a reasonable value of `x`, and returns an estimate of the square root of `a`.

To test it, write a function named `test_square_root` that prints a table like this:

```
a   mysqrt(a)     math.sqrt(a)  diff
-   ---------     ------------  ----
1.0 1.0           1.0           0.0
2.0 1.41421356237 1.41421356237 2.22044604925e-16
3.0 1.73205080757 1.73205080757 0.0
4.0 2.0           2.0           0.0
5.0 2.2360679775  2.2360679775  0.0
6.0 2.44948974278 2.44948974278 0.0
7.0 2.64575131106 2.64575131106 0.0
8.0 2.82842712475 2.82842712475 4.4408920985e-16
9.0 3.0           3.0           0.0
```

The first column is a number, *a*; the second column is the square root of *a* computed with `mysqrt`; the third column is the square root computed by `math.sqrt`; the fourth column is the absolute value of the difference between the two estimates.

Exercise 7-2.

The built-in function `eval` takes a string and evaluates it using the Python interpreter. For example:

```
>>> eval('1 + 2 * 3')
7
>>> import math
>>> eval('math.sqrt(5)')
2.2360679774997898
>>> eval('type(math.pi)')
<class 'float'>
```

Write a function called `eval_loop` that iteratively prompts the user, takes the resulting input and evaluates it using `eval`, and prints the result.

It should continue until the user enters `'done'`, and then return the value of the last expression it evaluated.

Exercise 7-3.

The mathematician Srinivasa Ramanujan found an infinite series that can be used to generate a numerical approximation of $1 / \pi$:

$$\frac{1}{\pi} = \frac{2\sqrt{2}}{9801} \sum_{k=0}^{\infty} \frac{(4k)!(1103 + 26390k)}{(k!)^4 396^{4k}}$$

Write a function called `estimate_pi` that uses this formula to compute and return an

estimate of π. It should use a `while` loop to compute terms of the summation until the last term is smaller than `1e-15` (which is Python notation for 10^{-15}). You can check the result by comparing it to `math.pi`.

Solution: *http://thinkpython2.com/code/pi.py*.

Chapter 8. Strings

Strings are not like integers, floats, and booleans. A string is a **sequence**, which means it is an ordered collection of other values. In this chapter you'll see how to access the characters that make up a string, and you'll learn about some of the methods strings provide.

A String Is a Sequence

A string is a sequence of characters. You can access the characters one at a time with the bracket operator:

```
>>> fruit = 'banana'
>>> letter = fruit[1]
```

The second statement selects character number 1 from `fruit` and assigns it to `letter`.

The expression in brackets is called an **index**. The index indicates which character in the sequence you want (hence the name).

But you might not get what you expect:

```
>>> letter
'a'
```

For most people, the first letter of `'banana'` is `b`, not `a`. But for computer scientists, the index is an offset from the beginning of the string, and the offset of the first letter is zero.

```
>>> letter = fruit[0]
>>> letter
'b'
```

So `b` is the 0th letter (“zero-eth”) of `'banana'`, `a` is the 1th letter (“one-eth”), and `n` is the 2th letter (“two-eth”).

As an index, you can use an expression that contains variables and operators:

```
>>> i = 1
>>> fruit[i]
'a'
>>> fruit[i+1]
'n'
```

But the value of the index has to be an integer. Otherwise you get:

```
>>> letter = fruit[1.5]
TypeError: string indices must be integers
```

len

`len` is a built-in function that returns the number of characters in a string:

```
>>> fruit = 'banana'
>>> len(fruit)
6
```

To get the last letter of a string, you might be tempted to try something like this:

```
>>> length = len(fruit)
>>> last = fruit[length]
IndexError: string index out of range
```

The reason for the `IndexError` is that there is no letter in `'banana'` with the index 6. Since we started counting at zero, the six letters are numbered 0 to 5. To get the last character, you have to subtract 1 from `length`:

```
>>> last = fruit[length-1]
>>> last
'a'
```

Or you can use negative indices, which count backward from the end of the string. The expression `fruit[-1]` yields the last letter, `fruit[-2]` yields the second to last, and so on.

Traversal with a for Loop

A lot of computations involve processing a string one character at a time. Often they start at the beginning, select each character in turn, do something to it, and continue until the end. This pattern of processing is called a **traversal**. One way to write a traversal is with a `while` loop:

```
index = 0
while index < len(fruit):
    letter = fruit[index]
    print(letter)
    index = index + 1
```

This loop traverses the string and displays each letter on a line by itself. The loop condition is `index < len(fruit)`, so when `index` is equal to the length of the string, the condition is false, and the body of the loop doesn't run. The last character accessed is the one with the index `len(fruit)-1`, which is the last character in the string.

As an exercise, write a function that takes a string as an argument and displays the letters backward, one per line.

Another way to write a traversal is with a `for` loop:

```
for letter in fruit:
    print(letter)
```

Each time through the loop, the next character in the string is assigned to the variable `letter`. The loop continues until no characters are left.

The following example shows how to use concatenation (string addition) and a `for` loop to generate an abecedarian series (that is, in alphabetical order). In Robert McCloskey's book *Make Way for Ducklings*, the names of the ducklings are Jack, Kack, Lack, Mack, Nack, Ouack, Pack, and Quack. This loop outputs these names in order:

```
prefixes = 'JKLMNOPQ'
suffix = 'ack'

for letter in prefixes:
    print(letter + suffix)
```

The output is:

```
Jack
Kack
Lack
Mack
Nack
Oack
Pack
Qack
```

Of course, that's not quite right because "Ouack" and "Quack" are misspelled. As an exercise, modify the program to fix this error.

String Slices

A segment of a string is called a **slice**. Selecting a slice is similar to selecting a character:

```
>>> s = 'Monty Python'
>>> s[0:5]
'Monty'
>>> s[6:12]
'Python'
```

The operator `[n:m]` returns the part of the string from the “n-eth” character to the “m-eth” character, including the first but excluding the last. This behavior is counterintuitive, but it might help to imagine the indices pointing *between* the characters, as in Figure 8-1.

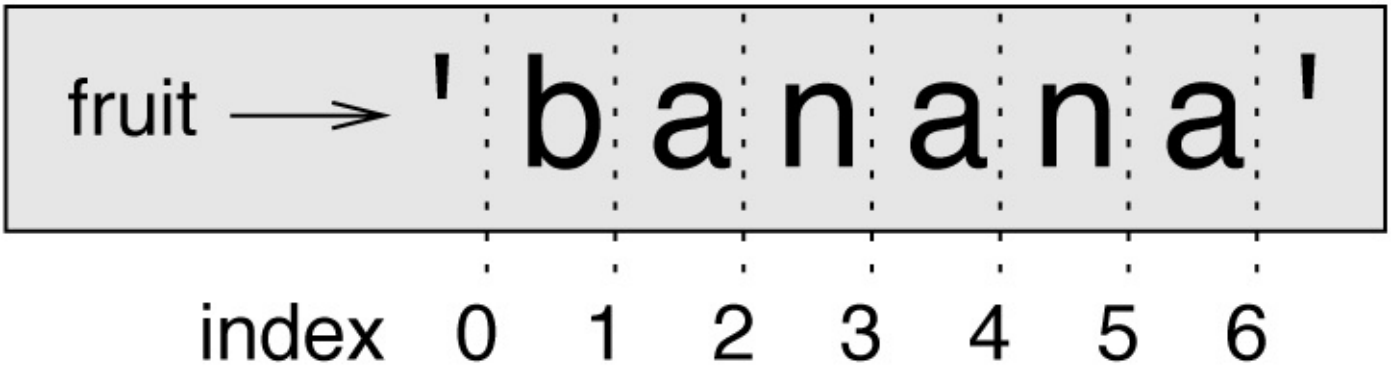


Figure 8-1. Slice indices.

If you omit the first index (before the colon), the slice starts at the beginning of the string. If you omit the second index, the slice goes to the end of the string:

```
>>> fruit = 'banana'
>>> fruit[:3]
'ban'
>>> fruit[3:]
'ana'
```

If the first index is greater than or equal to the second the result is an **empty string**, represented by two quotation marks:

```
>>> fruit = 'banana'
>>> fruit[3:3]
''
```

An empty string contains no characters and has length 0, but other than that, it is the same as any other string.

Continuing this example, what do you think `fruit[:]` means? Try it and see.

Strings Are Immutable

It is tempting to use the `[]` operator on the left side of an assignment, with the intention of changing a character in a string. For example:

```
>>> greeting = 'Hello, world!'
>>> greeting[0] = 'J'
TypeError: 'str' object does not support item assignment
```

The “object” in this case is the string and the “item” is the character you tried to assign. For now, an object is the same thing as a value, but we will refine that definition later (“Objects and Values”).

The reason for the error is that strings are **immutable**, which means you can’t change an existing string. The best you can do is create a new string that is a variation on the original:

```
>>> greeting = 'Hello, world!'
>>> new_greeting = 'J' + greeting[1:]
>>> new_greeting
'Jello, world!'
```

This example concatenates a new first letter onto a slice of `greeting`. It has no effect on the original string.

Searching

What does the following function do?

```
def find(word, letter):
    index = 0
    while index < len(word):
        if word[index] == letter:
            return index
        index = index + 1
    return -1
```

In a sense, `find` is the inverse of the `[]` operator. Instead of taking an index and extracting the corresponding character, it takes a character and finds the index where that character appears. If the character is not found, the function returns `-1`.

This is the first example we have seen of a `return` statement inside a loop. If `word[index] == letter`, the function breaks out of the loop and returns immediately.

If the character doesn't appear in the string, the program exits the loop normally and returns `-1`.

This pattern of computation — traversing a sequence and returning when we find what we are looking for — is called a **search**.

As an exercise, modify `find` so that it has a third parameter: the index in `word` where it should start looking.

Looping and Counting

The following program counts the number of times the letter `a` appears in a string:

```
word = 'banana'
count = 0
for letter in word:
    if letter == 'a':
        count = count + 1
print(count)
```

This program demonstrates another pattern of computation called a **counter**. The variable `count` is initialized to 0 and then incremented each time an `a` is found. When the loop exits, `count` contains the result — the total number of `a`'s.

As an exercise, encapsulate this code in a function named `count`, and generalize it so that it accepts the string and the letter as arguments.

Then rewrite the function so that instead of traversing the string, it uses the three-parameter version of `find` from the previous section.

String Methods

Strings provide methods that perform a variety of useful operations. A method is similar to a function — it takes arguments and returns a value — but the syntax is different. For example, the method `upper` takes a string and returns a new string with all uppercase letters.

Instead of the function syntax `upper(word)`, it uses the method syntax `word.upper()`:

```
>>> word = 'banana'
>>> new_word = word.upper()
>>> new_word
'BANANA'
```

This form of dot notation specifies the name of the method, `upper`, and the name of the string to apply the method to, `word`. The empty parentheses indicate that this method takes no arguments.

A method call is called an **invocation**; in this case, we would say that we are invoking `upper` on `word`.

As it turns out, there is a string method named `find` that is remarkably similar to the function we wrote:

```
>>> word = 'banana'
>>> index = word.find('a')
>>> index
1
```

In this example, we invoke `find` on `word` and pass the letter we are looking for as a parameter.

Actually, the `find` method is more general than our function; it can find substrings, not just characters:

```
>>> word.find('na')
2
```

By default, `find` starts at the beginning of the string, but it can take a second argument, the index where it should start:

```
>>> word.find('na', 3)
4
```

This is an example of an **optional argument**. `find` can also take a third argument, the index where it should stop:

```
>>> name = 'bob'
>>> name.find('b', 1, 2)
-1
```

This search fails because `b` does not appear in the index range from `1` to `2`, not including `2`. Searching up to, but not including, the second index makes `find` consistent with the slice operator.

The in Operator

The word `in` is a boolean operator that takes two strings and returns `True` if the first appears as a substring in the second:

```
>>> 'a' in 'banana'
True
>>> 'seed' in 'banana'
False
```

For example, the following function prints all the letters from `word1` that also appear in `word2`:

```
def in_both(word1, word2):
    for letter in word1:
        if letter in word2:
            print(letter)
```

With well-chosen variable names, Python sometimes reads like English. You could read this loop, “for (each) letter in (the first) word, if (the) letter (appears) in (the second) word, print (the) letter.”

Here’s what you get if you compare apples and oranges:

```
>>> in_both('apples', 'oranges')
a
e
s
```

String Comparison

The relational operators work on strings. To see if two strings are equal:

```
if word == 'banana':
    print('All right, bananas.')
```

Other relational operations are useful for putting words in alphabetical order:

```
if word < 'banana':
    print('Your word, ' + word + ', comes before banana.')
elif word > 'banana':
    print('Your word, ' + word + ', comes after banana.')
else:
    print('All right, bananas.')
```

Python does not handle uppercase and lowercase letters the same way people do. All the uppercase letters come before all the lowercase letters, so:

```
Your word, Pineapple, comes before banana.
```

A common way to address this problem is to convert strings to a standard format, such as all lowercase, before performing the comparison. Keep that in mind in case you have to defend yourself against a man armed with a Pineapple.

Debugging

When you use indices to traverse the values in a sequence, it is tricky to get the beginning and end of the traversal right. Here is a function that is supposed to compare two words and return `True` if one of the words is the reverse of the other, but it contains two errors:

```
def is_reverse(word1, word2):
    if len(word1) != len(word2):
        return False

    i = 0
    j = len(word2)

    while j > 0:
        if word1[i] != word2[j]:
            return False
        i = i+1
        j = j-1

    return True
```

The first `if` statement checks whether the words are the same length. If not, we can return `False` immediately. Otherwise, for the rest of the function, we can assume that the words are the same length. This is an example of the guardian pattern in “Checking Types”.

`i` and `j` are indices: `i` traverses `word1` forward while `j` traverses `word2` backward. If we find two letters that don’t match, we can return `False` immediately. If we get through the whole loop and all the letters match, we return `True`.

If we test this function with the words “pots” and “stop”, we expect the return value `True`, but we get an IndexError:

```
>>> is_reverse('pots', 'stop')
...
  File "reverse.py", line 15, in is_reverse
    if word1[i] != word2[j]:
IndexError: string index out of range
```

For debugging this kind of error, my first move is to print the values of the indices immediately before the line where the error appears.

```
    while j > 0:
        print(i, j)        # print here

        if word1[i] != word2[j]:
            return False
        i = i+1
        j = j-1
```

Now when I run the program again, I get more information:

```
>>> is_reverse('pots', 'stop')
0 4…
IndexError: string index out of range
```

The first time through the loop, the value of `j` is 4, which is out of range for the string

`'pots'`. The index of the last character is 3, so the initial value for `j` should be `len(word2)-1`.

If I fix that error and run the program again, I get:

```
>>> is_reverse('pots', 'stop')
0 3
1 2
2 1
True
```

This time we get the right answer, but it looks like the loop only ran three times, which is suspicious. To get a better idea of what is happening, it is useful to draw a state diagram. During the first iteration, the frame for `is_reverse` is shown in Figure 8-2.

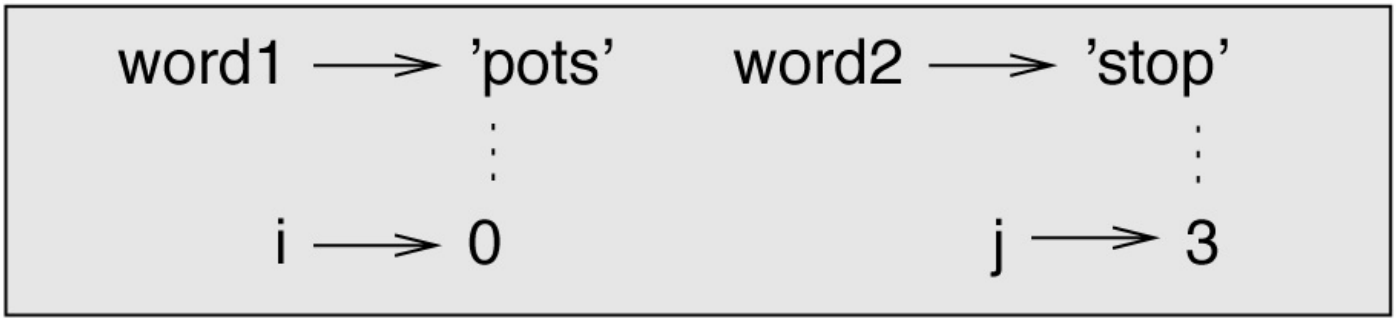


Figure 8-2. State diagram.

I took some license by arranging the variables in the frame and adding dotted lines to show that the values of `i` and `j` indicate characters in `word1` and `word2`.

Starting with this diagram, run the program on paper, changing the values of `i` and `j` during each iteration. Find and fix the second error in this function.

Glossary

object:

Something a variable can refer to. For now, you can use “object” and “value” interchangeably.

sequence:

An ordered collection of values where each value is identified by an integer index.

item:

One of the values in a sequence.

index:

An integer value used to select an item in a sequence, such as a character in a string. In Python indices start from 0.

slice:

A part of a string specified by a range of indices.

empty string:

A string with no characters and length 0, represented by two quotation marks.

immutable:

The property of a sequence whose items cannot be changed.

traverse:

To iterate through the items in a sequence, performing a similar operation on each.

search:

A pattern of traversal that stops when it finds what it is looking for.

counter:

A variable used to count something, usually initialized to zero and then incremented.

invocation:

A statement that calls a method.

optional argument:

A function or method argument that is not required.

Exercises

Exercise 8-1.

Read the documentation of the string methods at *http://docs.python.org/3/library/stdtypes.html#string-methods*. You might want to experiment with some of them to make sure you understand how they work. `strip` and `replace` are particularly useful.

The documentation uses a syntax that might be confusing. For example, in `find(sub[, start[, end]])`, the brackets indicate optional arguments. So `sub` is required, but `start` is optional, and if you include `start`, then `end` is optional.

Exercise 8-2.

There is a string method called `count` that is similar to the function in "Looping and Counting". Read the documentation of this method and write an invocation that counts the number of `a`'s in `'banana'`.

Exercise 8-3.

A string slice can take a third index that specifies the "step size"; that is, the number of spaces between successive characters. A step size of 2 means every other character; 3 means every third, etc.

```
>>> fruit = 'banana'
>>> fruit[0:5:2]
'bnn'
```

A step size of -1 goes through the word backwards, so the slice `[::-1]` generates a reversed string.

Use this idiom to write a one-line version of `is_palindrome` from Exercise 6-3.

Exercise 8-4.

The following functions are all *intended* to check whether a string contains any lowercase letters, but at least some of them are wrong. For each function, describe what the function actually does (assuming that the parameter is a string).

```
def any_lowercase1(s):
    for c in s:
        if c.islower():
            return True
        else:
            return False

def any_lowercase2(s):
    for c in s:
        if 'c'.islower():
            return 'True'
        else:
            return 'False'

def any_lowercase3(s):
    for c in s:
        flag = c.islower()
    return flag
```

```
def any_lowercase4(s):
    flag = False
    for c in s:
        flag = flag or c.islower()
    return flag

def any_lowercase5(s):
    for c in s:
        if not c.islower():
            return False
    return True
```

Exercise 8-5.

A Caesar cypher is a weak form of encryption that involves “rotating” each letter by a fixed number of places. To rotate a letter means to shift it through the alphabet, wrapping around to the beginning if necessary, so ‘A’ rotated by 3 is ‘D’ and ‘Z’ rotated by 1 is ‘A’.

To rotate a word, rotate each letter by the same amount. For example, “cheer” rotated by 7 is “jolly” and “melon” rotated by -10 is “cubed”. In the movie *2001: A Space Odyssey*, the ship computer is called HAL, which is IBM rotated by -1.

Write a function called `rotate_word` that takes a string and an integer as parameters, and returns a new string that contains the letters from the original string rotated by the given amount.

You might want to use the built-in function `ord`, which converts a character to a numeric code, and `chr`, which converts numeric codes to characters. Letters of the alphabet are encoded in alphabetical order, so for example:

```
>>> ord('c') - ord('a')
2
```

Because `'c'` is the two-eth letter of the alphabet. But beware: the numeric codes for uppercase letters are different.

Potentially offensive jokes on the Internet are sometimes encoded in ROT13, which is a Caesar cypher with rotation 13. If you are not easily offended, find and decode some of them.

Solution: *http://thinkpython2.com/code/rotate.py*.

Chapter 9. Case Study: Word Play

This chapter presents the second case study, which involves solving word puzzles by searching for words that have certain properties. For example, we'll find the longest palindromes in English and search for words whose letters appear in alphabetical order. And I will present another program development plan: reduction to a previously solved problem.

Reading Word Lists

For the exercises in this chapter we need a list of English words. There are lots of word lists available on the Web, but the one most suitable for our purpose is one of the word lists collected and contributed to the public domain by Grady Ward as part of the Moby lexicon project (see *http://wikipedia.org/wiki/Moby_Project*). It is a list of 113,809 official crosswords; that is, words that are considered valid in crossword puzzles and other word games. In the Moby collection, the filename is `113809of.fic`; you can download a copy, with the simpler name `words.txt`, from *http://thinkpython2.com/code/words.txt*.

This file is in plain text, so you can open it with a text editor, but you can also read it from Python. The built-in function `open` takes the name of the file as a parameter and returns a **file object** you can use to read the file.

```
>>> fin = open('words.txt')
```

`fin` is a common name for a file object used for input. The file object provides several methods for reading, including `readline`, which reads characters from the file until it gets to a newline and returns the result as a string:

```
>>> fin.readline()
'aa\r\n'
```

The first word in this particular list is “aa”, which is a kind of lava. The sequence `\r\n` represents two whitespace characters, a carriage return and a newline, that separate this word from the next.

The file object keeps track of where it is in the file, so if you call `readline` again, you get the next word:

```
>>> fin.readline()
'aah\r\n'
```

The next word is “aah”, which is a perfectly legitimate word, so stop looking at me like that. Or, if it’s the whitespace that’s bothering you, we can get rid of it with the string method `strip`:

```
>>> line = fin.readline()
>>> word = line.strip()
>>> word
'aahed'
```

You can also use a file object as part of a `for` loop. This program reads `words.txt` and prints each word, one per line:

```
fin = open('words.txt')
for line in fin:
    word = line.strip()
    print(word)
```

Exercises

There are solutions to these exercises in the next section. You should at least attempt each one before you read the solutions.

Exercise 9-1.

Write a program that reads `words.txt` and prints only the words with more than 20 characters (not counting whitespace).

Exercise 9-2.

In 1939 Ernest Vincent Wright published a 50,000-word novel called *Gadsby* that does not contain the letter "e". Since "e" is the most common letter in English, that's not easy to do.

In fact, it is difficult to construct a solitary thought without using that most common symbol. It is slow going at first, but with caution and hours of training you can gradually gain facility.

All right, I'll stop now.

Write a function called `has_no_e` that returns `True` if the given word doesn't have the letter "e" in it.

Modify your program from the previous section to print only the words that have no "e" and compute the percentage of the words in the list that have no "e".

Exercise 9-3.

Write a function named `avoids` that takes a word and a string of forbidden letters, and that returns `True` if the word doesn't use any of the forbidden letters.

Modify your program to prompt the user to enter a string of forbidden letters and then print the number of words that don't contain any of them. Can you find a combination of five forbidden letters that excludes the smallest number of words?

Exercise 9-4.

Write a function named `uses_only` that takes a word and a string of letters, and that returns `True` if the word contains only letters in the list. Can you make a sentence using only the letters `acefhlo`? Other than "Hoe alfalfa?"

Exercise 9-5.

Write a function named `uses_all` that takes a word and a string of required letters, and that returns `True` if the word uses all the required letters at least once. How many words are there that use all the vowels `aeiou`? How about `aeiouy`?

Exercise 9-6.

Write a function called `is_abecedarian` that returns `True` if the letters in a word appear in alphabetical order (double letters are okay). How many abecedarian words are there?

Search

All of the exercises in the previous section have something in common; they can be solved with the search pattern we saw in "Searching". The simplest example is:

```
def has_no_e(word):
    for letter in word:
        if letter == 'e':
            return False
    return True
```

The `for` loop traverses the characters in `word`. If we find the letter "e", we can immediately return `False`; otherwise we have to go to the next letter. If we exit the loop normally, that means we didn't find an "e", so we return `True`.

You could write this function more concisely using the `in` operator, but I started with this version because it demonstrates the logic of the search pattern.

`avoids` is a more general version of `has_no_e` but it has the same structure:

```
def avoids(word, forbidden):
    for letter in word:
        if letter in forbidden:
            return False
    return True
```

We can return `False` as soon as we find a forbidden letter; if we get to the end of the loop, we return `True`.

`uses_only` is similar except that the sense of the condition is reversed:

```
def uses_only(word, available):
    for letter in word:
        if letter not in available:
            return False
    return True
```

Instead of a list of forbidden letters, we have a list of available letters. If we find a letter in `word` that is not in `available`, we can return `False`.

`uses_all` is similar except that we reverse the role of the word and the string of letters:

```
def uses_all(word, required):
    for letter in required:
        if letter not in word:
            return False
    return True
```

Instead of traversing the letters in `word`, the loop traverses the required letters. If any of the required letters do not appear in the word, we can return `False`.

If you were really thinking like a computer scientist, you would have recognized that `uses_all` was an instance of a previously solved problem, and you would have written:

```
def uses_all(word, required):
    return uses_only(required, word)
```

This is an example of a program development plan called **reduction to a previously solved problem**, which means that you recognize the problem you are working on as an instance of a solved problem and apply an existing solution.

Looping with Indices

I wrote the functions in the previous section with `for` loops because I only needed the characters in the strings; I didn't have to do anything with the indices.

For `is_abecedarian` we have to compare adjacent letters, which is a little tricky with a `for` loop:

```
def is_abecedarian(word):
    previous = word[0]
    for c in word:
        if c < previous:
            return False
        previous = c
    return True
```

An alternative is to use recursion:

```
def is_abecedarian(word):
    if len(word) <= 1:
        return True
    if word[0] > word[1]:
        return False
    return is_abecedarian(word[1:])
```

Another option is to use a `while` loop:

```
def is_abecedarian(word):
    i = 0
    while i < len(word)-1:
        if word[i+1] < word[i]:
            return False
        i = i+1
    return True
```

The loop starts at `i=0` and ends when `i=len(word)-1`. Each time through the loop, it compares the *i*th character (which you can think of as the current character) to the *i+1*th character (which you can think of as the next).

If the next character is less than (alphabetically before) the current one, then we have discovered a break in the abecedarian trend, and we return `False`.

If we get to the end of the loop without finding a fault, then the word passes the test. To convince yourself that the loop ends correctly, consider an example like `'flossy'`. The length of the word is 6, so the last time the loop runs is when `i` is 4, which is the index of the second-to-last character. On the last iteration, it compares the second-to-last character to the last, which is what we want.

Here is a version of `is_palindrome` (see Exercise 6-3) that uses two indices: one starts at the beginning and goes up; the other starts at the end and goes down.

```
def is_palindrome(word):
    i = 0
    j = len(word)-1
```

```
    while i<j:
        if word[i] != word[j]:
            return False
        i = i+1
        j = j-1

    return True
```

Or we could reduce to a previously solved problem and write:

```
def is_palindrome(word):
    return is_reverse(word, word)
```

Using `is_reverse` from Figure 8-2.

Debugging

Testing programs is hard. The functions in this chapter are relatively easy to test because you can check the results by hand. Even so, it is somewhere between difficult and impossible to choose a set of words that test for all possible errors.

Taking `has_no_e` as an example, there are two obvious cases to check: words that have an 'e' should return `False`, and words that don't should return `True`. You should have no trouble coming up with one of each.

Within each case, there are some less obvious subcases. Among the words that have an "e", you should test words with an "e" at the beginning, the end, and somewhere in the middle. You should test long words, short words, and very short words, like the empty string. The empty string is an example of a **special case**, which is one of the non-obvious cases where errors often lurk.

In addition to the test cases you generate, you can also test your program with a word list like `words.txt`. By scanning the output, you might be able to catch errors, but be careful: you might catch one kind of error (words that should not be included, but are) and not another (words that should be included, but aren't).

In general, testing can help you find bugs, but it is not easy to generate a good set of test cases, and even if you do, you can't be sure your program is correct. According to a legendary computer scientist:

> Program testing can be used to show the presence of bugs, but never to show their absence!
>
> Edsger W. Dijkstra

Glossary

file object:

A value that represents an open file.

reduction to a previously solved problem:

A way of solving a problem by expressing it as an instance of a previously solved problem.

special case:

A test case that is atypical or non-obvious (and less likely to be handled correctly).

Exercises

Exercise 9-7.

This question is based on a Puzzler that was broadcast on the radio program *Car Talk* (*http://www.cartalk.com/content/puzzlers*):

> Give me a word with three consecutive double letters. I'll give you a couple of words that almost qualify, but don't. For example, the word committee, c-o-m-m-i-t-t-e-e. It would be great except for the 'i' that sneaks in there. Or Mississippi: M-i-s-s-i-s-s-i-p-p-i. If you could take out those i's it would work. But there is a word that has three consecutive pairs of letters and to the best of my knowledge this may be the only word. Of course there are probably 500 more but I can only think of one. What is the word?

Write a program to find it.

Solution: *http://thinkpython2.com/code/cartalk1.py*.

Exercise 9-8.

Here's another *Car Talk* Puzzler (*http://www.cartalk.com/content/puzzlers*):

> "I was driving on the highway the other day and I happened to notice my odometer. Like most odometers, it shows six digits, in whole miles only. So, if my car had 300,000 miles, for example, I'd see 3-0-0-0-0-0.
>
> "Now, what I saw that day was very interesting. I noticed that the last 4 digits were palindromic; that is, they read the same forward as backward. For example, 5-4-4-5 is a palindrome, so my odometer could have read 3-1-5-4-4-5.
>
> "One mile later, the last 5 numbers were palindromic. For example, it could have read 3-6-5-4-5-6. One mile after that, the middle 4 out of 6 numbers were palindromic. And you ready for this? One mile later, all 6 were palindromic!
>
> "The question is, what was on the odometer when I first looked?"

Write a Python program that tests all the six-digit numbers and prints any numbers that satisfy these requirements.

Solution: *http://thinkpython2.com/code/cartalk2.py*.

Exercise 9-9.

Here's another *Car Talk* Puzzler you can solve with a search (*http://www.cartalk.com/content/puzzlers*):

> “Recently I had a visit with my mom and we realized that the two digits that make up my age when reversed resulted in her age. For example, if she’s 73, I’m 37. We wondered how often this has happened over the years but we got sidetracked with other topics and we never came up with an answer.
>
> “When I got home I figured out that the digits of our ages have been reversible six times so far. I also figured out that if we’re lucky it would happen again in a few years, and if we’re really lucky it would happen one more time after that. In other words, it would have happened 8 times over all. So the question is, how old am I now?”

Write a Python program that searches for solutions to this Puzzler. Hint: you might find the string method `zfill` useful.

Solution: *http://thinkpython2.com/code/cartalk3.py*.

Chapter 10. Lists

This chapter presents one of Python’s most useful built-in types: lists. You will also learn more about objects and what can happen when you have more than one name for the same object.

A List Is a Sequence

Like a string, a **list** is a sequence of values. In a string, the values are characters; in a list, they can be any type. The values in a list are called **elements** or sometimes **items**.

There are several ways to create a new list; the simplest is to enclose the elements in square brackets (`[` and `]`):

```
[10, 20, 30, 40]
['crunchy frog', 'ram bladder', 'lark vomit']
```

The first example is a list of four integers. The second is a list of three strings. The elements of a list don't have to be the same type. The following list contains a string, a float, an integer, and (lo!) another list:

```
['spam', 2.0, 5, [10, 20]]
```

A list within another list is **nested**.

A list that contains no elements is called an empty list; you can create one with empty brackets, `[]`.

As you might expect, you can assign list values to variables:

```
>>> cheeses = ['Cheddar', 'Edam', 'Gouda']
>>> numbers = [42, 123]
>>> empty = []
>>> print(cheeses, numbers, empty)
['Cheddar', 'Edam', 'Gouda'] [42, 123] []
```

Lists Are Mutable

The syntax for accessing the elements of a list is the same as for accessing the characters of a string — the bracket operator. The expression inside the brackets specifies the index. Remember that the indices start at 0:

```
>>> cheeses[0]
'Cheddar'
```

Unlike strings, lists are mutable. When the bracket operator appears on the left side of an assignment, it identifies the element of the list that will be assigned:

```
>>> numbers = [42, 123]
>>> numbers[1] = 5
>>> numbers
[42, 5]
```

The one-eth element of `numbers`, which used to be 123, is now 5.

Figure 10-1 shows the state diagram for `cheeses`, `numbers` and `empty`.

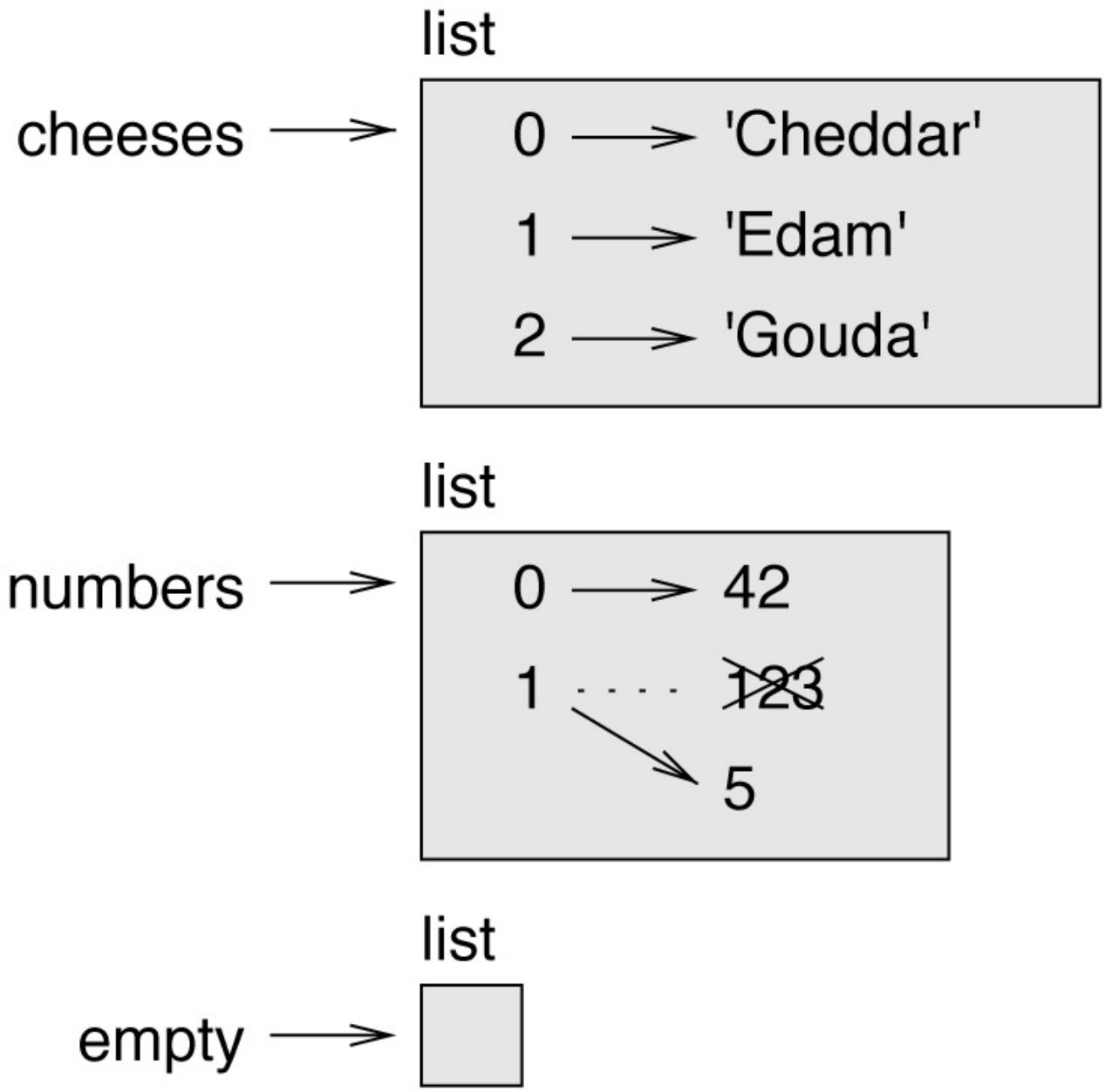


Figure 10-1. State diagram.

Lists are represented by boxes with the word “list” outside and the elements of the list inside. `cheeses` refers to a list with three elements indexed 0, 1 and 2. `numbers` contains two elements; the diagram shows that the value of the second element has been reassigned from 123 to 5. `empty` refers to a list with no elements.

List indices work the same way as string indices:

- Any integer expression can be used as an index.
- If you try to read or write an element that does not exist, you get an `IndexError`.
- If an index has a negative value, it counts backward from the end of the list.

The `in` operator also works on lists:

```
>>> cheeses = ['Cheddar', 'Edam', 'Gouda']
>>> 'Edam' in cheeses
True
>>> 'Brie' in cheeses
False
```

Traversing a List

The most common way to traverse the elements of a list is with a `for` loop. The syntax is the same as for strings:

```
for cheese in cheeses:
    print(cheese)
```

This works well if you only need to read the elements of the list. But if you want to write or update the elements, you need the indices. A common way to do that is to combine the built-in functions `range` and `len`:

```
for i in range(len(numbers)):
    numbers[i] = numbers[i] * 2
```

This loop traverses the list and updates each element. `len` returns the number of elements in the list. `range` returns a list of indices from 0 to *n-1*, where *n* is the length of the list. Each time through the loop, `i` gets the index of the next element. The assignment statement in the body uses `i` to read the old value of the element and to assign the new value.

A `for` loop over an empty list never runs the body:

```
for x in []:
    print('This never happens.')
```

Although a list can contain another list, the nested list still counts as a single element. The length of this list is four:

```
['spam', 1, ['Brie', 'Roquefort', 'Pol le Veq'], [1, 2, 3]]
```

List Operations

The `+` operator concatenates lists:

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> c
[1, 2, 3, 4, 5, 6]
```

The `*` operator repeats a list a given number of times:

```
>>> [0] * 4
[0, 0, 0, 0]
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

The first example repeats `[0]` four times. The second example repeats the list `[1, 2, 3]` three times.

List Slices

The slice operator also works on lists:

```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> t[1:3]
['b', 'c']
>>> t[:4]
['a', 'b', 'c', 'd']
>>> t[3:]
['d', 'e', 'f']
```

If you omit the first index, the slice starts at the beginning. If you omit the second, the slice goes to the end. So if you omit both, the slice is a copy of the whole list:

```
>>> t[:]
['a', 'b', 'c', 'd', 'e', 'f']
```

Since lists are mutable, it is often useful to make a copy before performing operations that modify lists.

A slice operator on the left side of an assignment can update multiple elements:

```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> t[1:3] = ['x', 'y']
>>> t
['a', 'x', 'y', 'd', 'e', 'f']
```

List Methods

Python provides methods that operate on lists. For example, `append` adds a new element to the end of a list:

```
>>> t = ['a', 'b', 'c']
>>> t.append('d')
>>> t
['a', 'b', 'c', 'd']
```

`extend` takes a list as an argument and appends all of the elements:

```
>>> t1 = ['a', 'b', 'c']
>>> t2 = ['d', 'e']
>>> t1.extend(t2)
>>> t1
['a', 'b', 'c', 'd', 'e']
```

This example leaves `t2` unmodified.

`sort` arranges the elements of the list from low to high:

```
>>> t = ['d', 'c', 'e', 'b', 'a']
>>> t.sort()
>>> t
['a', 'b', 'c', 'd', 'e']
```

Most list methods are void; they modify the list and return `None`. If you accidentally write `t = t.sort()`, you will be disappointed with the result.

Map, Filter and Reduce

To add up all the numbers in a list, you can use a loop like this:

```
def add_all(t):
    total = 0
    for x in t:
        total += x
    return total
```

`total` is initialized to 0. Each time through the loop, `x` gets one element from the list. The `+=` operator provides a short way to update a variable. This **augmented assignment statement**,

```
total += x
```

is equivalent to

```
total = total + x
```

As the loop runs, `total` accumulates the sum of the elements; a variable used this way is sometimes called an **accumulator**.

Adding up the elements of a list is such a common operation that Python provides it as a built-in function, `sum`:

```
>>> t = [1, 2, 3]
>>> sum(t)
6
```

An operation like this that combines a sequence of elements into a single value is sometimes called **reduce**.

Sometimes you want to traverse one list while building another. For example, the following function takes a list of strings and returns a new list that contains capitalized strings:

```
def capitalize_all(t):
    res = []
    for s in t:
        res.append(s.capitalize())
    return res
```

`res` is initialized with an empty list; each time through the loop, we append the next element. So `res` is another kind of accumulator.

An operation like `capitalize_all` is sometimes called a **map** because it “maps” a function (in this case the method `capitalize`) onto each of the elements in a sequence.

Another common operation is to select some of the elements from a list and return a sublist. For example, the following function takes a list of strings and returns a list that

contains only the uppercase strings:

```
def only_upper(t):
    res = []
    for s in t:
        if s.isupper():
            res.append(s)
    return res
```

`isupper` is a string method that returns `True` if the string contains only uppercase letters.

An operation like `only_upper` is called a **filter** because it selects some of the elements and filters out the others.

Most common list operations can be expressed as a combination of map, filter and reduce.

Deleting Elements

There are several ways to delete elements from a list. If you know the index of the element you want, you can use `pop`:

```
>>> t = ['a', 'b', 'c']
>>> x = t.pop(1)
>>> t
['a', 'c']
>>> x
'b'
```

`pop` modifies the list and returns the element that was removed. If you don't provide an index, it deletes and returns the last element.

If you don't need the removed value, you can use the `del` operator:

```
>>> t = ['a', 'b', 'c']
>>> del t[1]
>>> t
['a', 'c']
```

If you know the element you want to remove (but not the index), you can use `remove`:

```
>>> t = ['a', 'b', 'c']
>>> t.remove('b')
>>> t
['a', 'c']
```

The return value from `remove` is `None`.

To remove more than one element, you can use `del` with a slice index:

```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> del t[1:5]
>>> t
['a', 'f']
```

As usual, the slice selects all the elements up to but not including the second index.

Lists and Strings

A string is a sequence of characters and a list is a sequence of values, but a list of characters is not the same as a string. To convert from a string to a list of characters, you can use `list`:

```
>>> s = 'spam'
>>> t = list(s)
>>> t
['s', 'p', 'a', 'm']
```

Because `list` is the name of a built-in function, you should avoid using it as a variable name. I also avoid `l` because it looks too much like `1`. So that's why I use `t`.

The `list` function breaks a string into individual letters. If you want to break a string into words, you can use the `split` method:

```
>>> s = 'pining for the fjords'
>>> t = s.split()
>>> t
['pining', 'for', 'the', 'fjords']
```

An optional argument called a **delimiter** specifies which characters to use as word boundaries. The following example uses a hyphen as a delimiter:

```
>>> s = 'spam-spam-spam'
>>> delimiter = '-'
>>> t = s.split(delimiter)
>>> t
['spam', 'spam', 'spam']
```

`join` is the inverse of `split`. It takes a list of strings and concatenates the elements. `join` is a string method, so you have to invoke it on the delimiter and pass the list as a parameter:

```
>>> t = ['pining', 'for', 'the', 'fjords']
>>> delimiter = ' '
>>> s = delimiter.join(t)
>>> s
'pining for the fjords'
```

In this case the delimiter is a space character, so `join` puts a space between words. To concatenate strings without spaces, you can use the empty string, `''`, as a delimiter.

Objects and Values

If we run these assignment statements:

```
a = 'banana'
b = 'banana'
```

We know that `a` and `b` both refer to a string, but we don't know whether they refer to the *same* string. There are two possible states, shown in Figure 10-2.

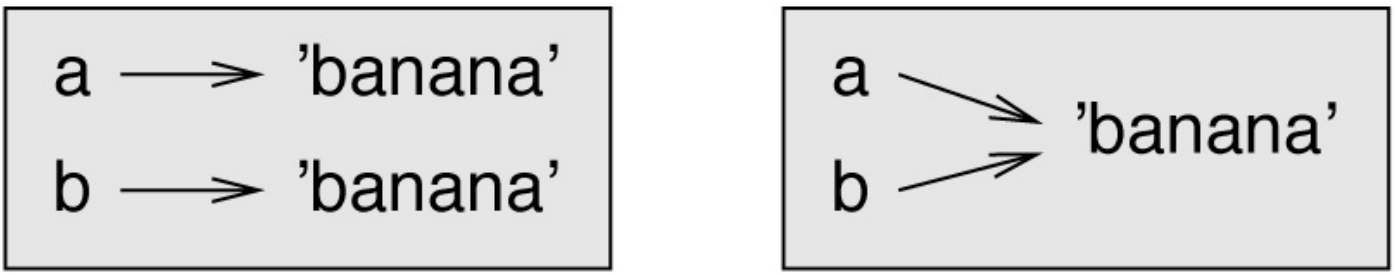


Figure 10-2. State diagram.

In one case, `a` and `b` refer to two different objects that have the same value. In the second case, they refer to the same object.

To check whether two variables refer to the same object, you can use the `is` operator:

```
>>> a = 'banana'
>>> b = 'banana'
>>> a is b
True
```

In this example, Python only created one string object, and both `a` and `b` refer to it. But when you create two lists, you get two objects:

```
>>> a = [1, 2, 3]
>>> b = [1, 2, 3]
>>> a is b
False
```

So the state diagram looks like Figure 10-3.

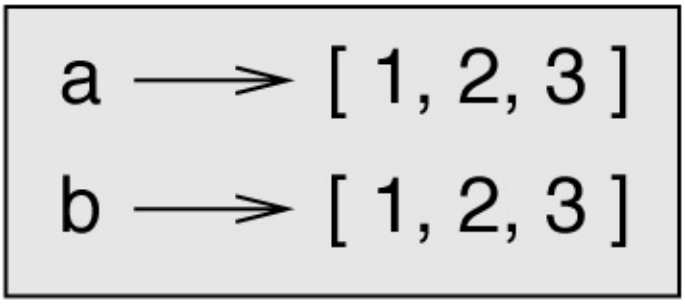


Figure 10-3. State diagram.

In this case we would say that the two lists are **equivalent**, because they have the same elements, but not **identical**, because they are not the same object. If two objects are

identical, they are also equivalent, but if they are equivalent, they are not necessarily identical.

Until now, we have been using “object” and “value” interchangeably, but it is more precise to say that an object has a value. If you evaluate `[1, 2, 3]`, you get a list object whose value is a sequence of integers. If another list has the same elements, we say it has the same value, but it is not the same object.

Aliasing

If a refers to an object and you assign `b = a`, then both variables refer to the same object:

```
>>> a = [1, 2, 3]
>>> b = a
>>> b is a
True
```

The state diagram looks like Figure 10-4.

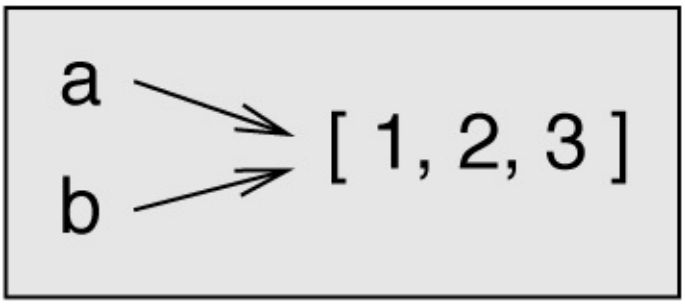


Figure 10-4. State diagram.

The association of a variable with an object is called a **reference**. In this example, there are two references to the same object.

An object with more than one reference has more than one name, so we say that the object is **aliased**.

If the aliased object is mutable, changes made with one alias affect the other:

```
>>> b[0] = 42
>>> a
[42, 2, 3]
```

Although this behavior can be useful, it is error-prone. In general, it is safer to avoid aliasing when you are working with mutable objects.

For immutable objects like strings, aliasing is not as much of a problem. In this example:

```
a = 'banana'
b = 'banana'
```

It almost never makes a difference whether `a` and `b` refer to the same string or not.

List Arguments

When you pass a list to a function, the function gets a reference to the list. If the function modifies the list, the caller sees the change. For example, `delete_head` removes the first element from a list:

```
def delete_head(t):
    del t[0]
```

Here's how it is used:

```
>>> letters = ['a', 'b', 'c']
>>> delete_head(letters)
>>> letters
['b', 'c']
```

The parameter `t` and the variable `letters` are aliases for the same object. The stack diagram looks like Figure 10-5.

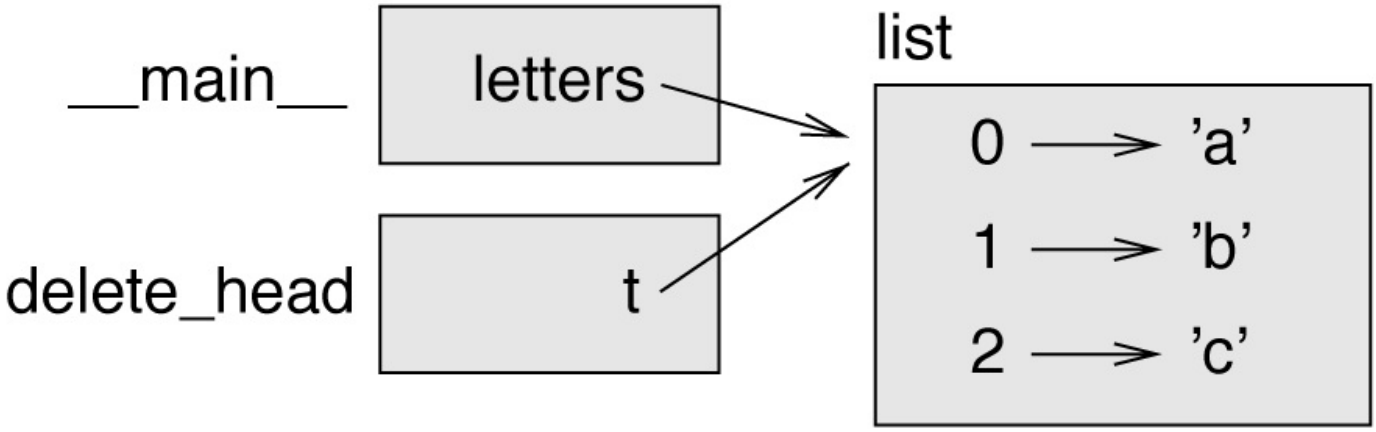


Figure 10-5. Stack diagram.

Since the list is shared by two frames, I drew it between them.

It is important to distinguish between operations that modify lists and operations that create new lists. For example, the `append` method modifies a list, but the `+` operator creates a new list:

```
>>> t1 = [1, 2]
>>> t2 = t1.append(3)
>>> t1
[1, 2, 3]
>>> t2
None
```

`append` modifies the list and returns `None`:

```
>>> t3 = t1 + [4]
>>> t1
[1, 2, 3]
>>> t3
[1, 2, 3, 4]
>>> t1
```

The + operator creates a new list and leaves the original list unchanged.

This difference is important when you write functions that are supposed to modify lists. For example, this function *does not* delete the head of a list:

```
def bad_delete_head(t):
    t = t[1:]               # WRONG!
```

The slice operator creates a new list and the assignment makes `t` refer to it, but that doesn't affect the caller.

```
>>> t4 = [1, 2, 3]
>>> bad_delete_head(t4)
>>> t4
[1, 2, 3]
```

At the beginning of `bad_delete_head`, `t` and `t4` refer to the same list. At the end, `t` refers to a new list, but `t4` still refers to the original, unmodified list.

An alternative is to write a function that creates and returns a new list. For example, `tail` returns all but the first element of a list:

```
def tail(t):
    return t[1:]
```

This function leaves the original list unmodified. Here's how it is used:

```
>>> letters = ['a', 'b', 'c']
>>> rest = tail(letters)
>>> rest
['b', 'c']
```

Debugging

Careless use of lists (and other mutable objects) can lead to long hours of debugging. Here are some common pitfalls and ways to avoid them:

1. Most list methods modify the argument and return `None`. This is the opposite of the string methods, which return a new string and leave the original alone.
   If you are used to writing string code like this:

   ```
   word = word.strip()
   ```

   It is tempting to write list code like this:

   ```
   t = t.sort()           # WRONG!
   ```

   Because `sort` returns `None`, the next operation you perform with `t` is likely to fail.
   Before using list methods and operators, you should read the documentation carefully and then test them in interactive mode.

2. Pick an idiom and stick with it.
   Part of the problem with lists is that there are too many ways to do things. For example, to remove an element from a list, you can use `pop`, `remove`, `del`, or even a slice assignment.
   To add an element, you can use the `append` method or the `+` operator. Assuming that `t` is a list and `x` is a list element, these are correct:

   ```
   t.append(x)
   t = t + [x]
   t += [x]
   ```

   And these are wrong:

   ```
   t.append([x])          # WRONG!
   t = t.append(x)        # WRONG!
   t + [x]                # WRONG!
   t = t + x              # WRONG!
   ```

   Try out each of these examples in interactive mode to make sure you understand what they do. Notice that only the last one causes a runtime error; the other three are legal, but they do the wrong thing.

3. Make copies to avoid aliasing.
   If you want to use a method like `sort` that modifies the argument, but you need to keep the original list as well, you can make a copy:

   ```
   >>> t = [3, 1, 2]
   ```

```
>>> t2 = t[:]
>>> t2.sort()
>>> t
[3, 1, 2]
>>> t2
[1, 2, 3]
```

In this example you could also use the built-in function `sorted`, which returns a new, sorted list and leaves the original alone:

```
>>> t2 = sorted(t)
>>> t
[3, 1, 2]
>>> t2
[1, 2, 3]
```

Glossary

list:

A sequence of values.

element:

One of the values in a list (or other sequence), also called items.

nested list:

A list that is an element of another list.

accumulator:

A variable used in a loop to add up or accumulate a result.

augmented assignment:

A statement that updates the value of a variable using an operator like +=.

reduce:

A processing pattern that traverses a sequence and accumulates the elements into a single result.

map:

A processing pattern that traverses a sequence and performs an operation on each element.

filter:

A processing pattern that traverses a list and selects the elements that satisfy some criterion.

object:

Something a variable can refer to. An object has a type and a value.

equivalent:

Having the same value.

identical:

Being the same object (which implies equivalence).

reference:

The association between a variable and its value.

aliasing:

A circumstance where two or more variables refer to the same object.

delimiter:

A character or string used to indicate where a string should be split.

Exercises

You can download solutions to these exercises from from *http://thinkpython2.com/code/list_exercises.py*.

Exercise 10-1.

Write a function called `nested_sum` that takes a list of lists of integers and adds up the elements from all of the nested lists. For example:

```
>>> t = [[1, 2], [3], [4, 5, 6]]
>>> nested_sum(t)
21
```

Exercise 10-2.

Write a function called `cumsum` that takes a list of numbers and returns the cumulative sum; that is, a new list where the *i*th element is the sum of the first *i+1* elements from the original list. For example:

```
>>> t = [1, 2, 3]
>>> cumsum(t)
[1, 3, 6]
```

Exercise 10-3.

Write a function called `middle` that takes a list and returns a new list that contains all but the first and last elements. For example:

```
>>> t = [1, 2, 3, 4]
>>> middle(t)
[2, 3]
```

Exercise 10-4.

Write a function called `chop` that takes a list, modifies it by removing the first and last elements, and returns `None`. For example:

```
>>> t = [1, 2, 3, 4]
>>> chop(t)
>>> t
[2, 3]
```

Exercise 10-5.

Write a function called `is_sorted` that takes a list as a parameter and returns `True` if the list is sorted in ascending order and `False` otherwise. For example:

```
>>> is_sorted([1, 2, 2])
True
>>> is_sorted(['b', 'a'])
False
```

Exercise 10-6.

Two words are anagrams if you can rearrange the letters from one to spell the other. Write a function called `is_anagram` that takes two strings and returns `True` if they are anagrams.

Exercise 10-7.

Write a function called `has_duplicates` that takes a list and returns `True` if there is any

element that appears more than once. It should not modify the original list.

Exercise 10-8.

This exercise pertains to the so-called Birthday Paradox, which you can read about at *http://en.wikipedia.org/wiki/Birthday_paradox*.

If there are 23 students in your class, what are the chances that two of you have the same birthday? You can estimate this probability by generating random samples of 23 birthdays and checking for matches. Hint: you can generate random birthdays with the `randint` function in the `random` module.

You can download my solution from *http://thinkpython2.com/code/birthday.py*.

Exercise 10-9.

Write a function that reads the file `words.txt` and builds a list with one element per word. Write two versions of this function, one using the `append` method and the other using the idiom `t = t + [x]`. Which one takes longer to run? Why?

Solution: *http://thinkpython2.com/code/wordlist.py*.

Exercise 10-10.

To check whether a word is in the word list, you could use the `in` operator, but it would be slow because it searches through the words in order.

Because the words are in alphabetical order, we can speed things up with a bisection search (also known as binary search), which is similar to what you do when you look a word up in the dictionary. You start in the middle and check to see whether the word you are looking for comes before the word in the middle of the list. If so, you search the first half of the list the same way. Otherwise you search the second half.

Either way, you cut the remaining search space in half. If the word list has 113,809 words, it will take about 17 steps to find the word or conclude that it's not there.

Write a function called `in_bisect` that takes a sorted list and a target value and returns the index of the value in the list if it's there, or `None` if it's not.

Or you could read the documentation of the `bisect` module and use that!

Solution: *http://thinkpython2.com/code/inlist.py*.

Exercise 10-11.

Two words are a "reverse pair" if each is the reverse of the other. Write a program that finds all the reverse pairs in the word list.

Solution: *http://thinkpython2.com/code/reverse_pair.py*.

Exercise 10-12.

Two words "interlock" if taking alternating letters from each forms a new word. For

example, “shoe” and “cold” interlock to form “schooled”.

Solution: *http://thinkpython2.com/code/interlock.py*. Credit: This exercise is inspired by an example at *http://puzzlers.org*.

1. Write a program that finds all pairs of words that interlock. Hint: don’t enumerate all pairs!

2. Can you find any words that are three-way interlocked; that is, every third letter forms a word, starting from the first, second or third?

Chapter 11. Dictionaries

This chapter presents another built-in type called a dictionary. Dictionaries are one of Python’s best features; they are the building blocks of many efficient and elegant algorithms.

A Dictionary Is a Mapping

A **dictionary** is like a list, but more general. In a list, the indices have to be integers; in a dictionary they can be (almost) any type.

A dictionary contains a collection of indices, which are called **keys**, and a collection of values. Each key is associated with a single value. The association of a key and a value is called a **key-value pair** or sometimes an **item**.

In mathematical language, a dictionary represents a **mapping** from keys to values, so you can also say that each key "maps to" a value. As an example, we'll build a dictionary that maps from English to Spanish words, so the keys and the values are all strings.

The function `dict` creates a new dictionary with no items. Because `dict` is the name of a built-in function, you should avoid using it as a variable name.

```
>>> eng2sp = dict()
>>> eng2sp
{}
```

The squiggly brackets, `{}`, represent an empty dictionary. To add items to the dictionary, you can use square brackets:

```
>>> eng2sp['one'] = 'uno'
```

This line creates an item that maps from the key `'one'` to the value `'uno'`. If we print the dictionary again, we see a key-value pair with a colon between the key and value:

```
>>> eng2sp
{'one': 'uno'}
```

This output format is also an input format. For example, you can create a new dictionary with three items:

```
>>> eng2sp = {'one': 'uno', 'two': 'dos', 'three': 'tres'}
```

But if you print `eng2sp`, you might be surprised:

```
>>> eng2sp
{'one': 'uno', 'three': 'tres', 'two': 'dos'}
```

The order of the key-value pairs might not be the same. If you type the same example on your computer, you might get a different result. In general, the order of items in a dictionary is unpredictable.

But that's not a problem because the elements of a dictionary are never indexed with integer indices. Instead, you use the keys to look up the corresponding values:

```
>>> eng2sp['two']
'dos'
```

The key `'two'` always maps to the value `'dos'` so the order of the items doesn't matter.

If the key isn't in the dictionary, you get an exception:

```
>>> eng2sp['four']
KeyError: 'four'
```

The `len` function works on dictionaries; it returns the number of key-value pairs:

```
>>> len(eng2sp)
3
```

The `in` operator works on dictionaries, too; it tells you whether something appears as a *key* in the dictionary (appearing as a value is not good enough).

```
>>> 'one' in eng2sp
True
>>> 'uno' in eng2sp
False
```

To see whether something appears as a value in a dictionary, you can use the method `values`, which returns a collection of values, and then use the `in` operator:

```
>>> vals = eng2sp.values()
>>> 'uno' in vals
True
```

The `in` operator uses different algorithms for lists and dictionaries. For lists, it searches the elements of the list in order, as in "Searching". As the list gets longer, the search time gets longer in direct proportion.

For dictionaries, Python uses an algorithm called a **hashtable** that has a remarkable property: the `in` operator takes about the same amount of time no matter how many items are in the dictionary. I explain how that's possible in "Hashtables", but the explanation might not make sense until you've read a few more chapters.

Dictionary as a Collection of Counters

Suppose you are given a string and you want to count how many times each letter appears. There are several ways you could do it:

1. You could create 26 variables, one for each letter of the alphabet. Then you could traverse the string and, for each character, increment the corresponding counter, probably using a chained conditional.

2. You could create a list with 26 elements. Then you could convert each character to a number (using the built-in function `ord`), use the number as an index into the list, and increment the appropriate counter.

3. You could create a dictionary with characters as keys and counters as the corresponding values. The first time you see a character, you would add an item to the dictionary. After that you would increment the value of an existing item.

Each of these options performs the same computation, but each of them implements that computation in a different way.

An **implementation** is a way of performing a computation; some implementations are better than others. For example, an advantage of the dictionary implementation is that we don't have to know ahead of time which letters appear in the string and we only have to make room for the letters that do appear.

Here is what the code might look like:

```
def histogram(s):
    d = dict()
    for c in s:
        if c not in d:
            d[c] = 1
        else:
            d[c] += 1
    return d
```

The name of the function is `histogram`, which is a statistical term for a collection of counters (or frequencies).

The first line of the function creates an empty dictionary. The `for` loop traverses the string. Each time through the loop, if the character `c` is not in the dictionary, we create a new item with key `c` and the initial value 1 (since we have seen this letter once). If `c` is already in the dictionary we increment `d[c]`.

Here's how it works:

```
>>> h = histogram('brontosaurus')
>>> h
{'a': 1, 'b': 1, 'o': 2, 'n': 1, 's': 2, 'r': 2, 'u': 2, 't': 1}
```

The histogram indicates that the letters `'a'` and `'b'` appear once; `'o'` appears twice, and

so on.

Dictionaries have a method called `get` that takes a key and a default value. If the key appears in the dictionary, `get` returns the corresponding value; otherwise it returns the default value. For example:

```
>>> h = histogram('a')
>>> h
{'a': 1}
>>> h.get('a', 0)
1
>>> h.get('b', 0)
0
```

As an exercise, use `get` to write `histogram` more concisely. You should be able to eliminate the `if` statement.

Looping and Dictionaries

If you use a dictionary in a `for` statement, it traverses the keys of the dictionary. For example, `print_hist` prints each key and the corresponding value:

```
def print_hist(h):
    for c in h:
        print(c, h[c])
```

Here's what the output looks like:

```
>>> h = histogram('parrot')
>>> print_hist(h)
a 1
p 1
r 2
t 1
o 1
```

Again, the keys are in no particular order. To traverse the keys in sorted order, you can use the built-in function `sorted`:

```
>>> for key in sorted(h):
...     print(key, h[key])
a 1
o 1
p 1
r 2
t 1
```

Reverse Lookup

Given a dictionary `d` and a key `k`, it is easy to find the corresponding value `v = d[k]`. This operation is called a **lookup**.

But what if you have `v` and you want to find `k`? You have two problems: first, there might be more than one key that maps to the value `v`. Depending on the application, you might be able to pick one, or you might have to make a list that contains all of them. Second, there is no simple syntax to do a **reverse lookup**; you have to search.

Here is a function that takes a value and returns the first key that maps to that value:

```
def reverse_lookup(d, v):
    for k in d:
        if d[k] == v:
            return k
    raise LookupError()
```

This function is yet another example of the search pattern, but it uses a feature we haven't seen before: `raise`. The **raise statement** causes an exception; in this case it causes a `LookupError`, which is a built-in exception used to indicate that a lookup operation failed.

If we get to the end of the loop, that means `v` doesn't appear in the dictionary as a value, so we raise an exception.

Here is an example of a successful reverse lookup:

```
>>> h = histogram('parrot')
>>> k = reverse_lookup(h, 2)
>>> k
'r'
```

And an unsuccessful one:

```
>>> k = reverse_lookup(h, 3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 5, in reverse_lookup
LookupError
```

The effect when you raise an exception is the same as when Python raises one: it prints a traceback and an error message.

The `raise` statement can take a detailed error message as an optional argument. For example:

```
>>> raise LookupError('value does not appear in the dictionary')
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
LookupError: value does not appear in the dictionary
```

A reverse lookup is much slower than a forward lookup; if you have to do it often, or if the dictionary gets big, the performance of your program will suffer.

Dictionaries and Lists

Lists can appear as values in a dictionary. For example, if you are given a dictionary that maps from letters to frequencies, you might want to invert it; that is, create a dictionary that maps from frequencies to letters. Since there might be several letters with the same frequency, each value in the inverted dictionary should be a list of letters.

Here is a function that inverts a dictionary:

```
def invert_dict(d):
    inverse = dict()
    for key in d:
        val = d[key]
        if val not in inverse:
            inverse[val] = [key]
        else:
            inverse[val].append(key)
    return inverse
```

Each time through the loop, `key` gets a key from `d` and `val` gets the corresponding value. If `val` is not in `inverse`, that means we haven't seen it before, so we create a new item and initialize it with a **singleton** (a list that contains a single element). Otherwise we have seen this value before, so we append the corresponding key to the list.

Here is an example:

```
>>> hist = histogram('parrot')
>>> hist
{'a': 1, 'p': 1, 'r': 2, 't': 1, 'o': 1}
>>> inverse = invert_dict(hist)
>>> inverse
{1: ['a', 'p', 't', 'o'], 2: ['r']}
```

Figure 11-1 is a state diagram showing `hist` and `inverse`. A dictionary is represented as a box with the type `dict` above it and the key-value pairs inside. If the values are integers, floats or strings, I draw them inside the box, but I usually draw lists outside the box, just to keep the diagram simple.

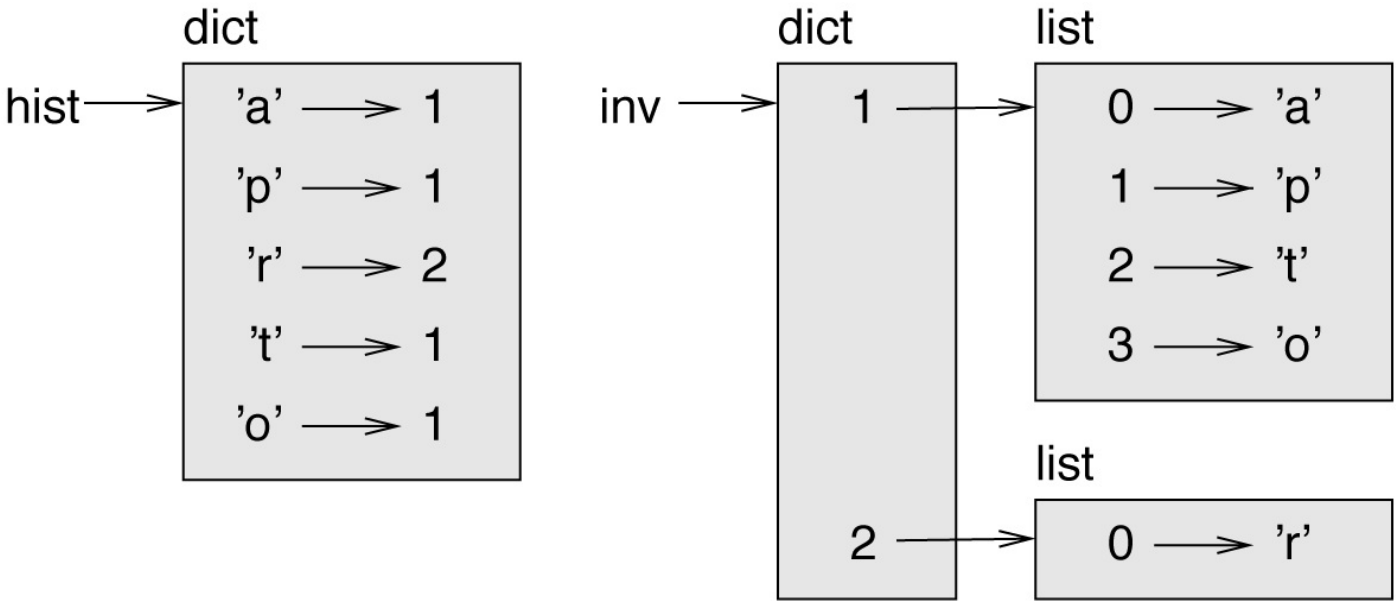


Figure 11-1. State diagram.

Lists can be values in a dictionary, as this example shows, but they cannot be keys. Here's what happens if you try:

```
>>> t = [1, 2, 3]
>>> d = dict()
>>> d[t] = 'oops'
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: list objects are unhashable
```

I mentioned earlier that a dictionary is implemented using a hashtable and that means that the keys have to be **hashable**.

A **hash** is a function that takes a value (of any kind) and returns an integer. Dictionaries use these integers, called hash values, to store and look up key-value pairs.

This system works fine if the keys are immutable. But if the keys are mutable, like lists, bad things happen. For example, when you create a key-value pair, Python hashes the key and stores it in the corresponding location. If you modify the key and then hash it again, it would go to a different location. In that case you might have two entries for the same key, or you might not be able to find a key. Either way, the dictionary wouldn't work correctly.

That's why keys have to be hashable, and why mutable types like lists aren't. The simplest way to get around this limitation is to use tuples, which we will see in the next chapter.

Since dictionaries are mutable, they can't be used as keys, but they *can* be used as values.

Memos

If you played with the `fibonacci` function from "One More Example", you might have noticed that the bigger the argument you provide, the longer the function takes to run. Furthermore, the runtime increases quickly.

To understand why, consider Figure 11-2, which shows the **call graph** for `fibonacci` with `n=4`.

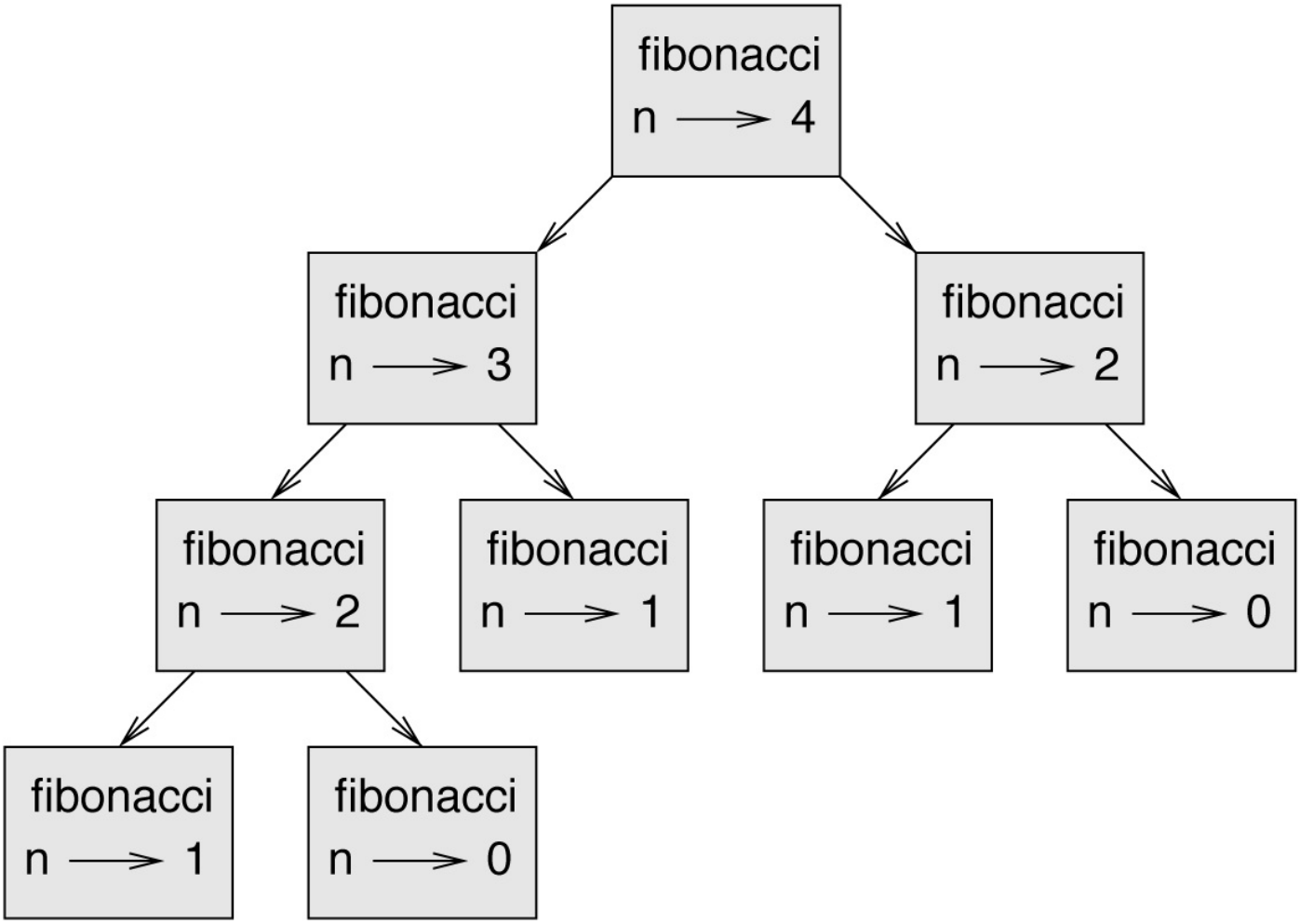


Figure 11-2. Call graph.

A call graph shows a set of function frames, with lines connecting each frame to the frames of the functions it calls. At the top of the graph, `fibonacci` with `n=4` calls `fibonacci` with `n=3` and `n=2`. In turn, `fibonacci` with `n=3` calls `fibonacci` with `n=2` and `n=1`. And so on.

Count how many times `fibonacci(0)` and `fibonacci(1)` are called. This is an inefficient solution to the problem, and it gets worse as the argument gets bigger.

One solution is to keep track of values that have already been computed by storing them in a dictionary. A previously computed value that is stored for later use is called a **memo**. Here is a "memoized" version of `fibonacci`:

```
known = {0:0, 1:1}
```

```
def fibonacci(n):
    if n in known:
        return known[n]

    res = fibonacci(n-1) + fibonacci(n-2)
    known[n] = res
    return res
```

`known` is a dictionary that keeps track of the Fibonacci numbers we already know. It starts with two items: 0 maps to 0 and 1 maps to 1.

Whenever `fibonacci` is called, it checks `known`. If the result is already there, it can return immediately. Otherwise it has to compute the new value, add it to the dictionary, and return it.

If you run this version of `fibonacci` and compare it with the original, you will find that it is much faster.

Global Variables

In the previous example, `known` is created outside the function, so it belongs to the special frame called `__main__`. Variables in `__main__` are sometimes called **global** because they can be accessed from any function. Unlike local variables, which disappear when their function ends, global variables persist from one function call to the next.

It is common to use global variables for **flags**; that is, boolean variables that indicate ("flag") whether a condition is true. For example, some programs use a flag named `verbose` to control the level of detail in the output:

```
verbose = True

def example1():
    if verbose:
        print('Running example1')
```

If you try to reassign a global variable, you might be surprised. The following example is supposed to keep track of whether the function has been called:

```
been_called = False

def example2():
    been_called = True          # WRONG
```

But if you run it you will see that the value of `been_called` doesn't change. The problem is that `example2` creates a new local variable named `been_called`. The local variable goes away when the function ends, and has no effect on the global variable.

To reassign a global variable inside a function you have to **declare** the global variable before you use it:

```
been_called = False

def example2():
    global been_called
    been_called = True
```

The **global statement** tells the interpreter something like, "In this function, when I say `been_called`, I mean the global variable; don't create a local one."

Here's an example that tries to update a global variable:

```
count = 0

def example3():
    count = count + 1           # WRONG
```

If you run it you get:

```
UnboundLocalError: local variable 'count' referenced before assignment
```

Python assumes that `count` is local, and under that assumption you are reading it before writing it. The solution, again, is to declare `count` global:

```
def example3():
    global count
    count += 1
```

If a global variable refers to a mutable value, you can modify the value without declaring the variable:

```
known = {0:0, 1:1}

def example4():
    known[2] = 1
```

So you can add, remove and replace elements of a global list or dictionary, but if you want to reassign the variable, you have to declare it:

```
def example5():
    global known
    known = dict()
```

Global variables can be useful, but if you have a lot of them, and you modify them frequently, they can make programs hard to debug.

Debugging

As you work with bigger datasets it can become unwieldy to debug by printing and checking the output by hand. Here are some suggestions for debugging large datasets:

Scale down the input:

> If possible, reduce the size of the dataset. For example if the program reads a text file, start with just the first 10 lines, or with the smallest example you can find. You can either edit the files themselves, or (better) modify the program so it reads only the first `n` lines.
>
> If there is an error, you can reduce `n` to the smallest value that manifests the error, and then increase it gradually as you find and correct errors.

Check summaries and types:

> Instead of printing and checking the entire dataset, consider printing summaries of the data: for example, the number of items in a dictionary or the total of a list of numbers.
>
> A common cause of runtime errors is a value that is not the right type. For debugging this kind of error, it is often enough to print the type of a value.

Write self-checks:

> Sometimes you can write code to check for errors automatically. For example, if you are computing the average of a list of numbers, you could check that the result is not greater than the largest element in the list or less than the smallest. This is called a "sanity check" because it detects results that are "insane".
>
> Another kind of check compares the results of two different computations to see if they are consistent. This is called a "consistency check".

Format the output:

> Formatting debugging output can make it easier to spot an error. We saw an example in "Debugging". The `pprint` module provides a `pprint` function that displays built-in types in a more human-readable format (`pprint` stands for "pretty print").

Again, time you spend building scaffolding can reduce the time you spend debugging.

Glossary

mapping:

A relationship in which each element of one set corresponds to an element of another set.

dictionary:

A mapping from keys to their corresponding values.

key-value pair:

The representation of the mapping from a key to a value.

item:

In a dictionary, another name for a key-value pair.

key:

An object that appears in a dictionary as the first part of a key-value pair.

value:

An object that appears in a dictionary as the second part of a key-value pair. This is more specific than our previous use of the word “value”.

implementation:

A way of performing a computation.

hashtable:

The algorithm used to implement Python dictionaries.

hash function:

A function used by a hashtable to compute the location for a key.

hashable:

A type that has a hash function. Immutable types like integers, floats and strings are hashable; mutable types like lists and dictionaries are not.

lookup:

A dictionary operation that takes a key and finds the corresponding value.

reverse lookup:

A dictionary operation that takes a value and finds one or more keys that map to it.

raise statement:

A statement that (deliberately) raises an exception.

singleton:

A list (or other sequence) with a single element.

call graph:

A diagram that shows every frame created during the execution of a program, with an arrow from each caller to each callee.

memo:

A computed value stored to avoid unnecessary future computation.

global variable:

A variable defined outside a function. Global variables can be accessed from any function.

global statement:

A statement that declares a variable name global.

flag:

A boolean variable used to indicate whether a condition is true.

declaration:

A statement like `global` that tells the interpreter something about a variable.

Exercises

Exercise 11-1.

Write a function that reads the words in `words.txt` and stores them as keys in a dictionary. It doesn't matter what the values are. Then you can use the `in` operator as a fast way to check whether a string is in the dictionary.

If you did Exercise 10-10, you can compare the speed of this implementation with the list `in` operator and the bisection search.

Exercise 11-2.

Read the documentation of the dictionary method `setdefault` and use it to write a more concise version of `invert_dict`.

Solution: *http://thinkpython2.com/code/invert_dict.py*.

Exercise 11-3.

Memoize the Ackermann function from Exercise 6-2 and see if memoization makes it possible to evaluate the function with bigger arguments. Hint: no.

Solution: *http://thinkpython2.com/code/ackermann_memo.py*.

Exercise 11-4.

If you did Exercise 10-7, you already have a function named `has_duplicates` that takes a list as a parameter and returns `True` if there is any object that appears more than once in the list.

Use a dictionary to write a faster, simpler version of `has_duplicates`.

Solution: *http://thinkpython2.com/code/has_duplicates.py*.

Exercise 11-5.

Two words are "rotate pairs" if you can rotate one of them and get the other (see `rotate_word` in Exercise 8-5).

Write a program that reads a wordlist and finds all the rotate pairs.

Solution: *http://thinkpython2.com/code/rotate_pairs.py*.

Exercise 11-6.

Here's another Puzzler from *Car Talk* (*http://www.cartalk.com/content/puzzlers*):

> This was sent in by a fellow named Dan O'Leary. He came upon a common one-syllable, five-letter word recently that has the following unique property. When you remove the first letter, the remaining letters form a homophone of the original word, that is a word that sounds exactly the same. Replace the first letter, that is, put it back and remove the second letter, and the result is yet another homophone of the original word. And the question is, what's the word?
>
> Now I'm going to give you an example that doesn't work. Let's look at the five-letter word, 'wrack.' W-R-A-C-K, you know like to 'wrack with pain.' If I remove the first letter, I am left with a four-letter word, 'R-A-C-K.' As in, 'Holy cow, did you see the rack on that buck! It must have been a nine-pointer!' It's a perfect homophone. If you put the 'w' back, and remove the 'r,' instead, you're left with the word, 'wack,' which is a real word, it's just not a homophone of the other two words.
>
> But there is, however, at least one word that Dan and we know of, which will yield two homophones if you remove either of the first two letters to make two, new four-letter words. The question is, what's the word?

You can use the dictionary from Exercise 11-1 to check whether a string is in the word list.

To check whether two words are homophones, you can use the CMU Pronouncing Dictionary. You can download it from *http://www.speech.cs.cmu.edu/cgi-bin/cmudict* or from *http://thinkpython2.com/code/c06d* and you can also download *http://thinkpython2.com/code/pronounce.py*, which provides a function named `read_dictionary` that reads the pronouncing dictionary and returns a Python dictionary that maps from each word to a string that describes its primary pronunciation.

Write a program that lists all the words that solve the Puzzler.

Solution: *http://thinkpython2.com/code/homophone.py*.

Chapter 12. Tuples

This chapter presents one more built-in type, the tuple, and then shows how lists, dictionaries, and tuples work together. I also present a useful feature for variable-length argument lists: the gather and scatter operators.

One note: there is no consensus on how to pronounce “tuple”. Some people say “tuh-ple”, which rhymes with “supple”. But in the context of programming, most people say “too-ple”, which rhymes with “quadruple”.

Tuples Are Immutable

A tuple is a sequence of values. The values can be any type, and they are indexed by integers, so in that respect tuples are a lot like lists. The important difference is that tuples are immutable.

Syntactically, a tuple is a comma-separated list of values:

```
>>> t = 'a', 'b', 'c', 'd', 'e'
```

Although it is not necessary, it is common to enclose tuples in parentheses:

```
>>> t = ('a', 'b', 'c', 'd', 'e')
```

To create a tuple with a single element, you have to include a final comma:

```
>>> t1 = 'a',
>>> type(t1)
<class 'tuple'>
```

A value in parentheses is not a tuple:

```
>>> t2 = ('a')
>>> type(t2)
<class 'str'>
```

Another way to create a tuple is the built-in function `tuple`. With no argument, it creates an empty tuple:

```
>>> t = tuple()
>>> t
()
```

If the argument is a sequence (string, list or tuple), the result is a tuple with the elements of the sequence:

```
>>> t = tuple('lupins')
>>> t
('l', 'u', 'p', 'i', 'n', 's')
```

Because `tuple` is the name of a built-in function, you should avoid using it as a variable name.

Most list operators also work on tuples. The bracket operator indexes an element:

```
>>> t = ('a', 'b', 'c', 'd', 'e')
>>> t[0]
'a'
```

And the slice operator selects a range of elements:

```
>>> t[1:3]
```

```
('b', 'c')
```

But if you try to modify one of the elements of the tuple, you get an error:

```
>>> t[0] = 'A'
TypeError: object doesn't support item assignment
```

Because tuples are immutable, you can’t modify the elements. But you can replace one tuple with another:

```
>>> t = ('A',) + t[1:]
>>> t
('A', 'b', 'c', 'd', 'e')
```

This statement makes a new tuple and then makes `t` refer to it.

The relational operators work with tuples and other sequences; Python starts by comparing the first element from each sequence. If they are equal, it goes on to the next elements, and so on, until it finds elements that differ. Subsequent elements are not considered (even if they are really big).

```
>>> (0, 1, 2) < (0, 3, 4)
True
>>> (0, 1, 2000000) < (0, 3, 4)
True
```

Tuple Assignment

It is often useful to swap the values of two variables. With conventional assignments, you have to use a temporary variable. For example, to swap `a` and `b`:

```
>>> temp = a
>>> a = b
>>> b = temp
```

This solution is cumbersome; **tuple assignment** is more elegant:

```
>>> a, b = b, a
```

The left side is a tuple of variables; the right side is a tuple of expressions. Each value is assigned to its respective variable. All the expressions on the right side are evaluated before any of the assignments.

The number of variables on the left and the number of values on the right have to be the same:

```
>>> a, b = 1, 2, 3
ValueError: too many values to unpack
```

More generally, the right side can be any kind of sequence (string, list or tuple). For example, to split an email address into a user name and a domain, you could write:

```
>>> addr = 'monty@python.org'
>>> uname, domain = addr.split('@')
```

The return value from `split` is a list with two elements; the first element is assigned to `uname`, the second to `domain`:

```
>>> uname
'monty'
>>> domain
'python.org'
```

Tuples as Return Values

Strictly speaking, a function can only return one value, but if the value is a tuple, the effect is the same as returning multiple values. For example, if you want to divide two integers and compute the quotient and remainder, it is inefficient to compute `x/y` and then `x%y`. It is better to compute them both at the same time.

The built-in function `divmod` takes two arguments and returns a tuple of two values: the quotient and remainder. You can store the result as a tuple:

```
>>> t = divmod(7, 3)
>>> t
(2, 1)
```

Or use tuple assignment to store the elements separately:

```
>>> quot, rem = divmod(7, 3)
>>> quot
2
>>> rem
1
```

Here is an example of a function that returns a tuple:

```
def min_max(t):
    return min(t), max(t)
```

`max` and `min` are built-in functions that find the largest and smallest elements of a sequence. `min_max` computes both and returns a tuple of two values.

Variable-Length Argument Tuples

Functions can take a variable number of arguments. A parameter name that begins with `*` **gathers** arguments into a tuple. For example, `printall` takes any number of arguments and prints them:

```
def printall(*args):
    print(args)
```

The gather parameter can have any name you like, but `args` is conventional. Here's how the function works:

```
>>> printall(1, 2.0, '3')
(1, 2.0, '3')
```

The complement of gather is **scatter**. If you have a sequence of values and you want to pass it to a function as multiple arguments, you can use the `*` operator. For example, `divmod` takes exactly two arguments; it doesn't work with a tuple:

```
>>> t = (7, 3)
>>> divmod(t)
TypeError: divmod expected 2 arguments, got 1
```

But if you scatter the tuple, it works:

```
>>> divmod(*t)
(2, 1)
```

Many of the built-in functions use variable-length argument tuples. For example, `max` and `min` can take any number of arguments:

```
>>> max(1, 2, 3)
3
```

But `sum` does not:

```
>>> sum(1, 2, 3)
TypeError: sum expected at most 2 arguments, got 3
```

As an exercise, write a function called `sumall` that takes any number of arguments and returns their sum.

Lists and Tuples

`zip` is a built-in function that takes two or more sequences and returns a list of tuples where each tuple contains one element from each sequence. The name of the function refers to a zipper, which joins and interleaves two rows of teeth.

This example zips a string and a list:

```
>>> s = 'abc'
>>> t = [0, 1, 2]
>>> zip(s, t)
<zip object at 0x7f7d0a9e7c48>
```

The result is a **zip object** that knows how to iterate through the pairs. The most common use of `zip` is in a `for` loop:

```
>>> for pair in zip(s, t):
...     print(pair)
...
('a', 0)
('b', 1)
('c', 2)
```

A zip object is a kind of **iterator**, which is any object that iterates through a sequence. Iterators are similar to lists in some ways, but unlike lists, you can’t use an index to select an element from an iterator.

If you want to use list operators and methods, you can use a zip object to make a list:

```
>>> list(zip(s, t))
[('a', 0), ('b', 1), ('c', 2)]
```

The result is a list of tuples; in this example, each tuple contains a character from the string and the corresponding element from the list.

If the sequences are not the same length, the result has the length of the shorter one:

```
>>> list(zip('Anne', 'Elk'))
[('A', 'E'), ('n', 'l'), ('n', 'k')]
```

You can use tuple assignment in a `for` loop to traverse a list of tuples:

```
t = [('a', 0), ('b', 1), ('c', 2)]
for letter, number in t:
    print(number, letter)
```

Each time through the loop, Python selects the next tuple in the list and assigns the elements to `letter` and `number`. The output of this loop is:

```
0 a
1 b
2 c
```

If you combine `zip`, `for` and tuple assignment, you get a useful idiom for traversing two (or more) sequences at the same time. For example, `has_match` takes two sequences, `t1` and `t2`, and returns `True` if there is an index `i` such that `t1[i] == t2[i]`:

```
def has_match(t1, t2):
    for x, y in zip(t1, t2):
        if x == y:
            return True
    return False
```

If you need to traverse the elements of a sequence and their indices, you can use the built-in function `enumerate`:

```
for index, element in enumerate('abc'):
    print(index, element)
```

The result from `enumerate` is an enumerate object, which iterates a sequence of pairs; each pair contains an index (starting from 0) and an element from the given sequence. In this example, the output is

```
0 a
1 b
2 c
```

Again.

Dictionaries and Tuples

Dictionaries have a method called `items` that returns a sequence of tuples, where each tuple is a key-value pair:

```
>>> d = {'a':0, 'b':1, 'c':2}
>>> t = d.items()
>>> t
dict_items([('c', 2), ('a', 0), ('b', 1)])
```

The result is a `dict_items` object, which is an iterator that iterates the key-value pairs. You can use it in a `for` loop like this:

```
>>> for key, value in d.items():
...     print(key, value)
...
c 2
a 0
b 1
```

As you should expect from a dictionary, the items are in no particular order.

Going in the other direction, you can use a list of tuples to initialize a new dictionary:

```
>>> t = [('a', 0), ('c', 2), ('b', 1)]
>>> d = dict(t)
>>> d
{'a': 0, 'c': 2, 'b': 1}
```

Combining `dict` with `zip` yields a concise way to create a dictionary:

```
>>> d = dict(zip('abc', range(3)))
>>> d
{'a': 0, 'c': 2, 'b': 1}
```

The dictionary method `update` also takes a list of tuples and adds them, as key-value pairs, to an existing dictionary.

It is common to use tuples as keys in dictionaries (primarily because you can't use lists). For example, a telephone directory might map from last-name, first-name pairs to telephone numbers. Assuming that we have defined `last`, `first` and `number`, we could write:

```
directory[last, first] = number
```

The expression in brackets is a tuple. We could use tuple assignment to traverse this dictionary:

```
for last, first in directory:
    print(first, last, directory[last,first])
```

This loop traverses the keys in `directory`, which are tuples. It assigns the elements of

each tuple to `last` and `first`, then prints the name and corresponding telephone number.

There are two ways to represent tuples in a state diagram. The more detailed version shows the indices and elements just as they appear in a list. For example, the tuple `('Cleese', 'John')` would appear as in Figure 12-1.

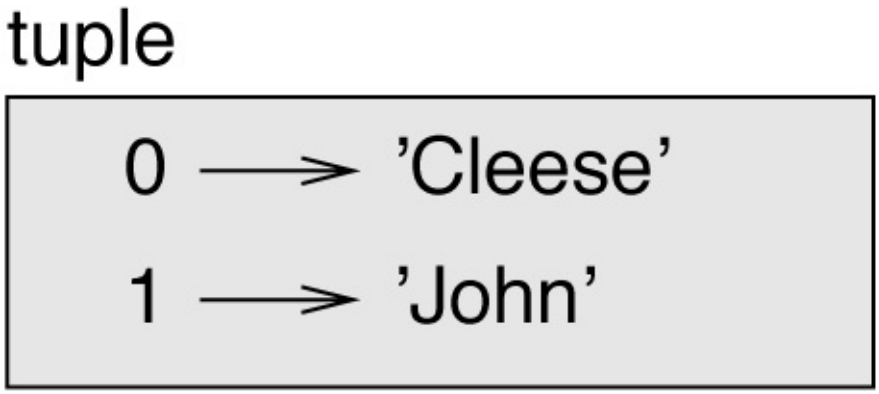


Figure 12-1. State diagram.

But in a larger diagram you might want to leave out the details. For example, a diagram of the telephone directory might appear as in Figure 12-2.

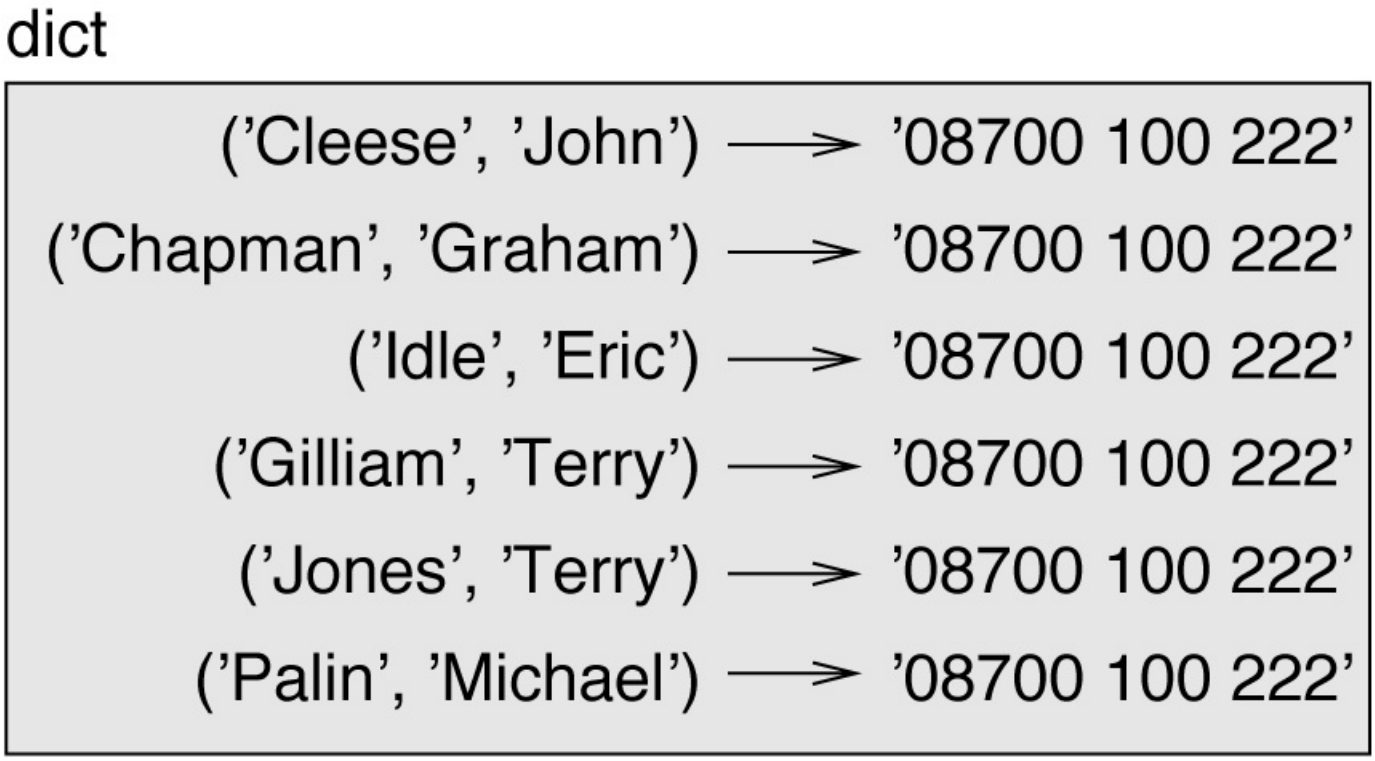


Figure 12-2. State diagram.

Here the tuples are shown using Python syntax as a graphical shorthand. The telephone number in the diagram is the complaints line for the BBC, so please don't call it.

Sequences of Sequences

I have focused on lists of tuples, but almost all of the examples in this chapter also work with lists of lists, tuples of tuples, and tuples of lists. To avoid enumerating the possible combinations, it is sometimes easier to talk about sequences of sequences.

In many contexts, the different kinds of sequences (strings, lists and tuples) can be used interchangeably. So how should you choose one over the others?

To start with the obvious, strings are more limited than other sequences because the elements have to be characters. They are also immutable. If you need the ability to change the characters in a string (as opposed to creating a new string), you might want to use a list of characters instead.

Lists are more common than tuples, mostly because they are mutable. But there are a few cases where you might prefer tuples:

1. In some contexts, like a `return` statement, it is syntactically simpler to create a tuple than a list.

2. If you want to use a sequence as a dictionary key, you have to use an immutable type like a tuple or string.

3. If you are passing a sequence as an argument to a function, using tuples reduces the potential for unexpected behavior due to aliasing.

Because tuples are immutable, they don't provide methods like `sort` and `reverse`, which modify existing lists. But Python provides the built-in function `sorted`, which takes any sequence and returns a new list with the same elements in sorted order, and `reversed`, which takes a sequence and returns an iterator that traverses the list in reverse order.

Debugging

Lists, dictionaries and tuples are examples of **data structures**; in this chapter we are starting to see compound data structures, like lists of tuples, or dictionaries that contain tuples as keys and lists as values. Compound data structures are useful, but they are prone to what I call **shape errors**; that is, errors caused when a data structure has the wrong type, size, or structure. For example, if you are expecting a list with one integer and I give you a plain old integer (not in a list), it won't work.

To help debug these kinds of errors, I have written a module called `structshape` that provides a function, also called `structshape`, that takes any kind of data structure as an argument and returns a string that summarizes its shape. You can download it from *http://thinkpython2.com/code/structshape.py*.

Here's the result for a simple list:

```
>>> from structshape import structshape
>>> t = [1, 2, 3]
>>> structshape(t)
'list of 3 int'
```

A fancier program might write "list of 3 ints", but it was easier not to deal with plurals. Here's a list of lists:

```
>>> t2 = [[1,2], [3,4], [5,6]]
>>> structshape(t2)
'list of 3 list of 2 int'
```

If the elements of the list are not the same type, `structshape` groups them, in order, by type:

```
>>> t3 = [1, 2, 3, 4.0, '5', '6', [7], [8], 9]
>>> structshape(t3)
'list of (3 int, float, 2 str, 2 list of int, int)'
```

Here's a list of tuples:

```
>>> s = 'abc'
>>> lt = list(zip(t, s))
>>> structshape(lt)
'list of 3 tuple of (int, str)'
```

And here's a dictionary with three items that map integers to strings:

```
>>> d = dict(lt)
>>> structshape(d)
'dict of 3 int->str'
```

If you are having trouble keeping track of your data structures, `structshape` can help.

Glossary

tuple:

An immutable sequence of elements.

tuple assignment:

An assignment with a sequence on the right side and a tuple of variables on the left. The right side is evaluated and then its elements are assigned to the variables on the left.

gather:

The operation of assembling a variable-length argument tuple.

scatter:

The operation of treating a sequence as a list of arguments.

zip object:

The result of calling a built-in function `zip`; an object that iterates through a sequence of tuples.

iterator:

An object that can iterate through a sequence, but which does not provide list operators and methods.

data structure:

A collection of related values, often organized in lists, dictionaries, tuples, etc.

shape error:

An error caused because a value has the wrong shape; that is, the wrong type or size.

Exercises

Exercise 12-1.

Write a function called `most_frequent` that takes a string and prints the letters in decreasing order of frequency. Find text samples from several different languages and see how letter frequency varies between languages. Compare your results with the tables at *http://en.wikipedia.org/wiki/Letter_frequencies*.

Solution: *http://thinkpython2.com/code/most_frequent.py*.

Exercise 12-2.

More anagrams!

1. Write a program that reads a word list from a file (see "Reading Word Lists") and prints all the sets of words that are anagrams.
   Here is an example of what the output might look like:
   ```
   ['deltas', 'desalt', 'lasted', 'salted', 'slated', 'staled']
   ['retainers', 'ternaries']
   ['generating', 'greatening']
   ['resmelts', 'smelters', 'termless']
   ```

   Hint: you might want to build a dictionary that maps from a collection of letters to a list of words that can be spelled with those letters. The question is, how can you represent the collection of letters in a way that can be used as a key?

2. Modify the previous program so that it prints the longest list of anagrams first, followed by the second longest, and so on.

3. In Scrabble, a "bingo" is when you play all seven tiles in your rack, along with a letter on the board, to form an eight-letter word. What collection of eight letters forms the most possible bingos? Hint: there are seven.
   Solution: *http://thinkpython2.com/code/anagram_sets.py*.

Exercise 12-3.

Two words form a "metathesis pair" if you can transform one into the other by swapping two letters; for example, "converse" and "conserve". Write a program that finds all of the metathesis pairs in the dictionary. Hint: don't test all pairs of words, and don't test all possible swaps.

Solution: *http://thinkpython2.com/code/metathesis.py*. Credit: This exercise is inspired by an example at *http://puzzlers.org*.

Exercise 12-4.

Here's another Car Talk Puzzler (*http://www.cartalk.com/content/puzzlers*):

> What is the longest English word, that remains a valid English word, as you remove its letters one at a time?
>
> Now, letters can be removed from either end, or the middle, but you can't rearrange any of the letters. Every time you drop a letter, you wind up with another English word. If you do that, you're eventually going to wind up with one letter and that too is going to be an English word — one that's found in the dictionary. I want to know what's the longest word and how many letters does it have?
>
> I'm going to give you a little modest example: Sprite. Ok? You start off with sprite, you take a letter off, one from the interior of the word, take the r away, and we're left with the word spite, then we take the e off the end, we're left with spit, we take the s off, we're left with pit, it, and I.

Write a program to find all words that can be reduced in this way, and then find the longest one.

This exercise is a little more challenging than most, so here are some suggestions:

1. You might want to write a function that takes a word and computes a list of all the words that can be formed by removing one letter. These are the "children" of the word.

2. Recursively, a word is reducible if any of its children are reducible. As a base case, you can consider the empty string reducible.

3. The wordlist I provided, `words.txt`, doesn't contain single letter words. So you might want to add "I", "a", and the empty string.

4. To improve the performance of your program, you might want to memoize the words that are known to be reducible.

Solution: *http://thinkpython2.com/code/reducible.py*.

Chapter 13. Case Study: Data Structure Selection

At this point you have learned about Python's core data structures, and you have seen some of the algorithms that use them. If you would like to know more about algorithms, this might be a good time to read Chapter 21. But you don't have to read it before you go on; you can read it whenever you are interested.

This chapter presents a case study with exercises that let you think about choosing data structures and practice using them.

Word Frequency Analysis

As usual, you should at least attempt the exercises before you read my solutions.

Exercise 13-1.

Write a program that reads a file, breaks each line into words, strips whitespace and punctuation from the words, and converts them to lowercase.

Hint: The `string` module provides a string named `whitespace`, which contains space, tab, newline, etc., and `punctuation` which contains the punctuation characters. Let's see if we can make Python swear:

```
>>> import string
>>> string.punctuation
'!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~'
```

Also, you might consider using the string methods `strip`, `replace` and `translate`.

Exercise 13-2.

Go to Project Gutenberg (*http://gutenberg.org*) and download your favorite out-of-copyright book in plain text format.

Modify your program from the previous exercise to read the book you downloaded, skip over the header information at the beginning of the file, and process the rest of the words as before.

Then modify the program to count the total number of words in the book, and the number of times each word is used.

Print the number of different words used in the book. Compare different books by different authors, written in different eras. Which author uses the most extensive vocabulary?

Exercise 13-3.

Modify the program from the previous exercise to print the 20 most frequently used words in the book.

Exercise 13-4.

Modify the previous program to read a word list (see "Reading Word Lists") and then print all the words in the book that are not in the word list. How many of them are typos? How many of them are common words that *should* be in the word list, and how many of them are really obscure?

Random Numbers

Given the same inputs, most computer programs generate the same outputs every time, so they are said to be **deterministic**. Determinism is usually a good thing, since we expect the same calculation to yield the same result. For some applications, though, we want the computer to be unpredictable. Games are an obvious example, but there are more.

Making a program truly nondeterministic turns out to be difficult, but there are ways to make it at least seem nondeterministic. One of them is to use algorithms that generate **pseudorandom** numbers. Pseudorandom numbers are not truly random because they are generated by a deterministic computation, but just by looking at the numbers it is all but impossible to distinguish them from random.

The `random` module provides functions that generate pseudorandom numbers (which I will simply call "random" from here on).

The function `random` returns a random float between 0.0 and 1.0 (including 0.0 but not 1.0). Each time you call `random`, you get the next number in a long series. To see a sample, run this loop:

```
import random

for i in range(10):
    x = random.random()
    print(x)
```

The function `randint` takes parameters `low` and `high` and returns an integer between `low` and `high` (including both):

```
>>> random.randint(5, 10)
5
>>> random.randint(5, 10)
9
```

To choose an element from a sequence at random, you can use `choice`:

```
>>> t = [1, 2, 3]
>>> random.choice(t)
2
>>> random.choice(t)
3
```

The `random` module also provides functions to generate random values from continuous distributions including Gaussian, exponential, gamma, and a few more.

Exercise 13-5.

Write a function named `choose_from_hist` that takes a histogram as defined in "Dictionary as a Collection of Counters" and returns a random value from the histogram, chosen with probability in proportion to frequency. For example, for this histogram:

```
>>> t = ['a', 'a', 'b']
>>> hist = histogram(t)
```

```
>>> hist
{'a': 2, 'b': 1}
```

your function should return `'a'` with probability *2/3* and `'b'` with probability *1/3*.

Word Histogram

You should attempt the previous exercises before you go on. You can download my solution from *http://thinkpython2.com/code/analyze_book1.py*. You will also need *http://thinkpython2.com/code/emma.txt*.

Here is a program that reads a file and builds a histogram of the words in the file:

```
import string

def process_file(filename):
    hist = dict()
    fp = open(filename)
    for line in fp:
        process_line(line, hist)
    return hist

def process_line(line, hist):
    line = line.replace('-', ' ')

    for word in line.split():
        word = word.strip(string.punctuation + string.whitespace)
        word = word.lower()
        hist[word] = hist.get(word, 0) + 1

hist = process_file('emma.txt')
```

This program reads `emma.txt`, which contains the text of *Emma* by Jane Austen.

`process_file` loops through the lines of the file, passing them one at a time to `process_line`. The histogram `hist` is being used as an accumulator.

`process_line` uses the string method `replace` to replace hyphens with spaces before using `split` to break the line into a list of strings. It traverses the list of words and uses `strip` and `lower` to remove punctuation and convert to lowercase. (It is shorthand to say that strings are “converted”; remember that strings are immutable, so methods like `strip` and `lower` return new strings.)

Finally, `process_line` updates the histogram by creating a new item or incrementing an existing one.

To count the total number of words in the file, we can add up the frequencies in the histogram:

```
def total_words(hist):
    return sum(hist.values())
```

The number of different words is just the number of items in the dictionary:

```
def different_words(hist):
    return len(hist)
```

Here is some code to print the results:

```
print('Total number of words:', total_words(hist))
```

```
print('Number of different words:', different_words(hist))
```

And the results:

```
Total number of words: 161080
Number of different words: 7214
```

Most Common Words

To find the most common words, we can make a list of tuples, where each tuple contains a word and its frequency, and sort it.

The following function takes a histogram and returns a list of word-frequency tuples:

```
def most_common(hist):
    t = []
    for key, value in hist.items():
        t.append((value, key))

    t.sort(reverse=True)
    return t
```

In each tuple, the frequency appears first, so the resulting list is sorted by frequency. Here is a loop that prints the 10 most common words:

```
t = most_common(hist)
print('The most common words are:')
for freq, word in t[:10]:
    print(word, freq, sep='\t')
```

I use the keyword argument `sep` to tell `print` to use a tab character as a “separator”, rather than a space, so the second column is lined up. Here are the results from *Emma*:

```
The most common words are:
to      5242
the     5205
and     4897
of      4295
i       3191
a       3130
it      2529
her     2483
was     2400
she     2364
```

This code can be simplified using the `key` parameter of the `sort` function. If you are curious, you can read about it at *https://wiki.python.org/moin/HowTo/Sorting*.

Optional Parameters

We have seen built-in functions and methods that take optional arguments. It is possible to write programmer-defined functions with optional arguments, too. For example, here is a function that prints the most common words in a histogram:

```
def print_most_common(hist, num=10):
    t = most_common(hist)
    print('The most common words are:')
    for freq, word in t[:num]:
        print(word, freq, sep='\t')
```

The first parameter is required; the second is optional. The **default value** of `num` is 10.

If you only provide one argument:

```
print_most_common(hist)
```

`num` gets the default value. If you provide two arguments:

```
print_most_common(hist, 20)
```

`num` gets the value of the argument instead. In other words, the optional argument **overrides** the default value.

If a function has both required and optional parameters, all the required parameters have to come first, followed by the optional ones.

Dictionary Subtraction

Finding the words from the book that are not in the word list from `words.txt` is a problem you might recognize as set subtraction; that is, we want to find all the words from one set (the words in the book) that are not in the other (the words in the list).

`subtract` takes dictionaries `d1` and `d2` and returns a new dictionary that contains all the keys from `d1` that are not in `d2`. Since we don't really care about the values, we set them all to None:

```
def subtract(d1, d2):
    res = dict()
    for key in d1:
        if key not in d2:
            res[key] = None
    return res
```

To find the words in the book that are not in `words.txt`, we can use `process_file` to build a histogram for `words.txt`, and then subtract:

```
words = process_file('words.txt')
diff = subtract(hist, words)

print("Words in the book that aren't in the word list:")
for word in diff:
    print(word, end=' ')
```

Here are some of the results from *Emma*:

```
Words in the book that aren't in the word list:
rencontre jane's blanche woodhouses disingenuousness
friend's venice apartment…
```

Some of these words are names and possessives. Others, like "rencontre", are no longer in common use. But a few are common words that should really be in the list!

Exercise 13-6.

Python provides a data structure called `set` that provides many common set operations. You can read about them in "Sets", or read the documentation at *http://docs.python.org/3/library/stdtypes.html#types-set*.

Write a program that uses set subtraction to find words in the book that are not in the word list.

Solution: *http://thinkpython2.com/code/analyze_book2.py*.

Random Words

To choose a random word from the histogram, the simplest algorithm is to build a list with multiple copies of each word, according to the observed frequency, and then choose from the list:

```
def random_word(h):
    t = []
    for word, freq in h.items():
        t.extend([word] * freq)

    return random.choice(t)
```

The expression `[word] * freq` creates a list with `freq` copies of the string `word`. The `extend` method is similar to `append` except that the argument is a sequence.

This algorithm works, but it is not very efficient; each time you choose a random word, it rebuilds the list, which is as big as the original book. An obvious improvement is to build the list once and then make multiple selections, but the list is still big.

An alternative is:

1. Use `keys` to get a list of the words in the book.

2. Build a list that contains the cumulative sum of the word frequencies (see Exercise 10-2). The last item in this list is the total number of words in the book, *n*.

3. Choose a random number from 1 to *n*. Use a bisection search (See Exercise 10-10) to find the index where the random number would be inserted in the cumulative sum.

4. Use the index to find the corresponding word in the word list.

Exercise 13-7.

Write a program that uses this algorithm to choose a random word from the book.

Solution: *http://thinkpython2.com/code/analyze_book3.py*.

Markov Analysis

If you choose words from the book at random, you can get a sense of the vocabulary, but you probably won't get a sentence:

```
this the small regard harriet which knightley's it most things
```

A series of random words seldom makes sense because there is no relationship between successive words. For example, in a real sentence you would expect an article like "the" to be followed by an adjective or a noun, and probably not a verb or adverb.

One way to measure these kinds of relationships is Markov analysis, which characterizes, for a given sequence of words, the probability of the words that might come next. For example, the song "Eric, the Half a Bee" begins:

- Half a bee, philosophically,
- Must, ipso facto, half not be.
- But half the bee has got to be
- Vis a vis, its entity. D'you see?
-
- But can a bee be said to be
- Or not to be an entire bee
- When half the bee is not a bee
- Due to some ancient injury?

In this text, the phrase "half the" is always followed by the word "bee", but the phrase "the bee" might be followed by either "has" or "is".

The result of Markov analysis is a mapping from each prefix (like "half the" and "the bee") to all possible suffixes (like "has" and "is").

Given this mapping, you can generate a random text by starting with any prefix and choosing at random from the possible suffixes. Next, you can combine the end of the prefix and the new suffix to form the next prefix, and repeat.

For example, if you start with the prefix "Half a", then the next word has to be "bee", because the prefix only appears once in the text. The next prefix is "a bee", so the next suffix might be "philosophically", "be" or "due".

In this example the length of the prefix is always two, but you can do Markov analysis with any prefix length.

Exercise 13-8.

Markov analysis:

1. Write a program to read a text from a file and perform Markov analysis. The result should be a dictionary that maps from prefixes to a collection of possible suffixes. The collection might be a list, tuple, or dictionary; it is up to you to make an appropriate choice. You can test your program with prefix length 2, but you should write the program in a way that makes it easy to try other lengths.

2. Add a function to the previous program to generate random text based on the Markov analysis. Here is an example from *Emma* with prefix length 2:

   > He was very clever, be it sweetness or be angry, ashamed or only amused, at such a stroke. She had never thought of Hannah till you were never meant for me?" "I cannot make speeches, Emma:" he soon cut it all himself.

   For this example, I left the punctuation attached to the words. The result is almost syntactically correct, but not quite. Semantically, it almost makes sense, but not quite.
   What happens if you increase the prefix length? Does the random text make more sense?

3. Once your program is working, you might want to try a mash-up: if you combine text from two or more books, the random text you generate will blend the vocabulary and phrases from the sources in interesting ways.

Credit: This case study is based on an example from Kernighan and Pike, *The Practice of Programming*, Addison-Wesley, 1999.

You should attempt this exercise before you go on; then you can can download my solution from *http://thinkpython2.com/code/markov.py*. You will also need *http://thinkpython2.com/code/emma.txt*.

Data Structures

Using Markov analysis to generate random text is fun, but there is also a point to this exercise: data structure selection. In your solution to the previous exercises, you had to choose:

- How to represent the prefixes.
- How to represent the collection of possible suffixes.
- How to represent the mapping from each prefix to the collection of possible suffixes.

The last one is easy: a dictionary is the obvious choice for a mapping from keys to corresponding values.

For the prefixes, the most obvious options are string, list of strings, or tuple of strings.

For the suffixes, one option is a list; another is a histogram (dictionary).

How should you choose? The first step is to think about the operations you will need to implement for each data structure. For the prefixes, we need to be able to remove words from the beginning and add to the end. For example, if the current prefix is “Half a”, and the next word is “bee”, you need to be able to form the next prefix, “a bee”.

Your first choice might be a list, since it is easy to add and remove elements, but we also need to be able to use the prefixes as keys in a dictionary, so that rules out lists. With tuples, you can’t append or remove, but you can use the addition operator to form a new tuple:

```
def shift(prefix, word):
    return prefix[1:] + (word,)
```

`shift` takes a tuple of words, `prefix`, and a string, `word`, and forms a new tuple that has all the words in `prefix` except the first, and `word` added to the end.

For the collection of suffixes, the operations we need to perform include adding a new suffix (or increasing the frequency of an existing one), and choosing a random suffix.

Adding a new suffix is equally easy for the list implementation or the histogram. Choosing a random element from a list is easy; choosing from a histogram is harder to do efficiently (see Exercise 13-7).

So far we have been talking mostly about ease of implementation, but there are other factors to consider in choosing data structures. One is runtime. Sometimes there is a theoretical reason to expect one data structure to be faster than other; for example, I mentioned that the `in` operator is faster for dictionaries than for lists, at least when the number of elements is large.

But often you don’t know ahead of time which implementation will be faster. One option is to implement both of them and see which is better. This approach is called

benchmarking. A practical alternative is to choose the data structure that is easiest to implement, and then see if it is fast enough for the intended application. If so, there is no need to go on. If not, there are tools, like the `profile` module, that can identify the places in a program that take the most time.

The other factor to consider is storage space. For example, using a histogram for the collection of suffixes might take less space because you only have to store each word once, no matter how many times it appears in the text. In some cases, saving space can also make your program run faster, and in the extreme, your program might not run at all if you run out of memory. But for many applications, space is a secondary consideration after runtime.

One final thought: in this discussion, I have implied that we should use one data structure for both analysis and generation. But since these are separate phases, it would also be possible to use one structure for analysis and then convert to another structure for generation. This would be a net win if the time saved during generation exceeded the time spent in conversion.

Debugging

When you are debugging a program, and especially if you are working on a hard bug, there are five things to try:

Reading:

> Examine your code, read it back to yourself, and check that it says what you meant to say.

Running:

> Experiment by making changes and running different versions. Often if you display the right thing at the right place in the program, the problem becomes obvious, but sometimes you have to build scaffolding.

Ruminating:

> Take some time to think! What kind of error is it: syntax, runtime, or semantic? What information can you get from the error messages, or from the output of the program? What kind of error could cause the problem you're seeing? What did you change last, before the problem appeared?

Rubberducking:

> If you explain the problem to someone else, you sometimes find the answer before you finish asking the question. Often you don't need the other person; you could just talk to a rubber duck. And that's the origin of the well-known strategy called **rubber duck debugging**. I am not making this up; see *https://en.wikipedia.org/wiki/Rubber_duck_debugging*.

Retreating:

> At some point, the best thing to do is back off and undo recent changes until you get back to a program that works and that you understand. Then you can start rebuilding.

Beginning programmers sometimes get stuck on one of these activities and forget the others. Each activity comes with its own failure mode.

For example, reading your code might help if the problem is a typographical error, but not if the problem is a conceptual misunderstanding. If you don't understand what your program does, you can read it 100 times and never see the error, because the error is in your head.

Running experiments can help, especially if you run small, simple tests. But if you run experiments without thinking or reading your code, you might fall into a pattern I call "random walk programming", which is the process of making random changes until the program does the right thing. Needless to say, random walk programming can take a long time.

You have to take time to think. Debugging is like an experimental science. You should have at least one hypothesis about what the problem is. If there are two or more

possibilities, try to think of a test that would eliminate one of them.

But even the best debugging techniques will fail if there are too many errors, or if the code you are trying to fix is too big and complicated. Sometimes the best option is to retreat, simplifying the program until you get to something that works and that you understand.

Beginning programmers are often reluctant to retreat because they can’t stand to delete a line of code (even if it’s wrong). If it makes you feel better, copy your program into another file before you start stripping it down. Then you can copy the pieces back one at a time.

Finding a hard bug requires reading, running, ruminating, and sometimes retreating. If you get stuck on one of these activities, try the others.

Glossary

deterministic:

Pertaining to a program that does the same thing each time it runs, given the same inputs.

pseudorandom:

Pertaining to a sequence of numbers that appears to be random, but is generated by a deterministic program.

default value:

The value given to an optional parameter if no argument is provided.

override:

To replace a default value with an argument.

benchmarking:

The process of choosing between data structures by implementing alternatives and testing them on a sample of the possible inputs.

rubber duck debugging:

Debugging by explaining your problem to an inanimate object such as a rubber duck. Articulating the problem can help you solve it, even if the rubber duck doesn’t know Python.

Exercises

Exercise 13-9.

The “rank” of a word is its position in a list of words sorted by frequency: the most common word has rank 1, the second most common has rank 2, etc.

Zipf’s law describes a relationship between the ranks and frequencies of words in natural languages (*http://en.wikipedia.org/wiki/Zipf’s_law*). Specifically, it predicts that the frequency, *f*, of the word with rank *r* is:

$$f = cr^{-s}$$

where *s* and *c* are parameters that depend on the language and the text. If you take the logarithm of both sides of this equation, you get:

$$\log f = \log c - s \log r$$

So if you plot log *f* versus log *r*, you should get a straight line with slope -*s* and intercept log *c*.

Write a program that reads a text from a file, counts word frequencies, and prints one line for each word, in descending order of frequency, with log *f* and log *r*. Use the graphing program of your choice to plot the results and check whether they form a straight line. Can you estimate the value of *s*?

Solution: *http://thinkpython2.com/code/zipf.py*. To run my solution, you need the plotting module `matplotlib`. If you installed Anaconda, you already have `matplotlib`; otherwise you might have to install it.

Chapter 14. Files

This chapter introduces the idea of “persistent” programs that keep data in permanent storage, and shows how to use different kinds of permanent storage, like files and databases.

Persistence

Most of the programs we have seen so far are transient in the sense that they run for a short time and produce some output, but when they end, their data disappears. If you run the program again, it starts with a clean slate.

Other programs are **persistent**: they run for a long time (or all the time); they keep at least some of their data in permanent storage (a hard drive, for example); and if they shut down and restart, they pick up where they left off.

Examples of persistent programs are operating systems, which run pretty much whenever a computer is on, and web servers, which run all the time, waiting for requests to come in on the network.

One of the simplest ways for programs to maintain their data is by reading and writing text files. We have already seen programs that read text files; in this chapter we will see programs that write them.

An alternative is to store the state of the program in a database. In this chapter I will present a simple database and a module, `pickle`, that makes it easy to store program data.

Reading and Writing

A text file is a sequence of characters stored on a permanent medium like a hard drive, flash memory, or CD-ROM. We saw how to open and read a file in “Reading Word Lists”.

To write a file, you have to open it with mode `'w'` as a second parameter:

```
>>> fout = open('output.txt', 'w')
```

If the file already exists, opening it in write mode clears out the old data and starts fresh, so be careful! If the file doesn’t exist, a new one is created.

`open` returns a file object that provides methods for working with the file. The `write` method puts data into the file:

```
>>> line1 = "This here's the wattle,\n"
>>> fout.write(line1)
24
```

The return value is the number of characters that were written. The file object keeps track of where it is, so if you call `write` again, it adds the new data to the end of the file:

```
>>> line2 = "the emblem of our land.\n"
>>> fout.write(line2)
24
```

When you are done writing, you should close the file:

```
>>> fout.close()
```

If you don’t close the file, it gets closed for you when the program ends.

Format Operator

The argument of `write` has to be a string, so if we want to put other values in a file, we have to convert them to strings. The easiest way to do that is with `str`:

```
>>> x = 52
>>> fout.write(str(x))
```

An alternative is to use the **format operator**, `%`. When applied to integers, `%` is the modulus operator. But when the first operand is a string, `%` is the format operator.

The first operand is the **format string**, which contains one or more **format sequences**, which specify how the second operand is formatted. The result is a string.

For example, the format sequence `'%d'` means that the second operand should be formatted as a decimal integer:

```
>>> camels = 42
>>> '%d' % camels
'42'
```

The result is the string `'42'`, which is not to be confused with the integer value `42`.

A format sequence can appear anywhere in the string, so you can embed a value in a sentence:

```
>>> 'I have spotted %d camels.' % camels
'I have spotted 42 camels.'
```

If there is more than one format sequence in the string, the second argument has to be a tuple. Each format sequence is matched with an element of the tuple, in order.

The following example uses `'%d'` to format an integer, `'%g'` to format a floating-point number, and `'%s'` to format a string:

```
>>> 'In %d years I have spotted %g %s.' % (3, 0.1, 'camels')
'In 3 years I have spotted 0.1 camels.'
```

The number of elements in the tuple has to match the number of format sequences in the string. Also, the types of the elements have to match the format sequences:

```
>>> '%d %d %d' % (1, 2)
TypeError: not enough arguments for format string
>>> '%d' % 'dollars'
TypeError: %d format: a number is required, not str
```

In the first example, there aren’t enough elements; in the second, the element is the wrong type.

For more information on the format operator, see *https://docs.python.org/3/library/stdtypes.html#printf-style-string-formatting*. A more

powerful alternative is the string format method, which you can read about at *https://docs.python.org/3/library/stdtypes.html#str.format*.

Filenames and Paths

Files are organized into **directories** (also called “folders”). Every running program has a “current directory”, which is the default directory for most operations. For example, when you open a file for reading, Python looks for it in the current directory.

The `os` module provides functions for working with files and directories (“os” stands for “operating system”). `os.getcwd` returns the name of the current directory:

```
>>> import os
>>> cwd = os.getcwd()
>>> cwd
'/home/dinsdale'
```

`cwd` stands for “current working directory”. The result in this example is `/home/dinsdale`, which is the home directory of a user named `dinsdale`.

A string like `'/home/dinsdale'` that identifies a file or directory is called a **path**.

A simple filename, like `memo.txt`, is also considered a path, but it is a **relative path** because it relates to the current directory. If the current directory is `/home/dinsdale`, the filename `memo.txt` would refer to `/home/dinsdale/memo.txt`.

A path that begins with `/` does not depend on the current directory; it is called an **absolute path**. To find the absolute path to a file, you can use `os.path.abspath`:

```
>>> os.path.abspath('memo.txt')
'/home/dinsdale/memo.txt'
```

`os.path` provides other functions for working with filenames and paths. For example, `os.path.exists` checks whether a file or directory exists:

```
>>> os.path.exists('memo.txt')
True
```

If it exists, `os.path.isdir` checks whether it’s a directory:

```
>>> os.path.isdir('memo.txt')
False
>>> os.path.isdir('/home/dinsdale')
True
```

Similarly, `os.path.isfile` checks whether it’s a file.

`os.listdir` returns a list of the files (and other directories) in the given directory:

```
>>> os.listdir(cwd)
['music', 'photos', 'memo.txt']
```

To demonstrate these functions, the following example “walks” through a directory, prints the names of all the files, and calls itself recursively on all the directories:

```
def walk(dirname):
    for name in os.listdir(dirname):
        path = os.path.join(dirname, name)

        if os.path.isfile(path):
            print(path)
        else:
            walk(path)
```

`os.path.join` takes a directory and a filename and joins them into a complete path.

The `os` module provides a function called `walk` that is similar to this one but more versatile. As an exercise, read the documentation and use it to print the names of the files in a given directory and its subdirectories. You can download my solution from *http://thinkpython2.com/code/walk.py*.

Catching Exceptions

A lot of things can go wrong when you try to read and write files. If you try to open a file that doesn't exist, you get an `IOError`:

```
>>> fin = open('bad_file')
IOError: [Errno 2] No such file or directory: 'bad_file'
```

If you don't have permission to access a file:

```
>>> fout = open('/etc/passwd', 'w')
PermissionError: [Errno 13] Permission denied: '/etc/passwd'
```

And if you try to open a directory for reading, you get

```
>>> fin = open('/home')
IsADirectoryError: [Errno 21] Is a directory: '/home'
```

To avoid these errors, you could use functions like `os.path.exists` and `os.path.isfile`, but it would take a lot of time and code to check all the possibilities (if "`Errno 21`" is any indication, there are at least 21 things that can go wrong).

It is better to go ahead and try — and deal with problems if they happen — which is exactly what the `try` statement does. The syntax is similar to an `if…else` statement:

```
try:
    fin = open('bad_file')
except:
    print('Something went wrong.')
```

Python starts by executing the `try` clause. If all goes well, it skips the `except` clause and proceeds. If an exception occurs, it jumps out of the `try` clause and runs the `except` clause.

Handling an exception with a `try` statement is called **catching** an exception. In this example, the `except` clause prints an error message that is not very helpful. In general, catching an exception gives you a chance to fix the problem, or try again, or at least end the program gracefully.

Databases

A **database** is a file that is organized for storing data. Many databases are organized like a dictionary in the sense that they map from keys to values. The biggest difference between a database and a dictionary is that the database is on disk (or other permanent storage), so it persists after the program ends.

The module `dbm` provides an interface for creating and updating database files. As an example, I'll create a database that contains captions for image files.

Opening a database is similar to opening other files:

```
>>> import dbm
>>> db = dbm.open('captions', 'c')
```

The mode `'c'` means that the database should be created if it doesn't already exist. The result is a database object that can be used (for most operations) like a dictionary.

When you create a new item, `dbm` updates the database file:

```
>>> db['cleese.png'] = 'Photo of John Cleese.'
```

When you access one of the items, `dbm` reads the file:

```
>>> db['cleese.png']
b'Photo of John Cleese.'
```

The result is a **bytes object**, which is why it begins with `b`. A bytes object is similar to a string in many ways. When you get farther into Python, the difference becomes important, but for now we can ignore it.

If you make another assignment to an existing key, `dbm` replaces the old value:

```
>>> db['cleese.png'] = 'Photo of John Cleese doing a silly walk.'
>>> db['cleese.png']
b'Photo of John Cleese doing a silly walk.'
```

Some dictionary methods, like `keys` and `items`, don't work with database objects. But iteration with a `for` loop works:

```
for key in db:
    print(key, db[key])
```

As with other files, you should close the database when you are done:

```
>>> db.close()
```

Pickling

A limitation of `dbm` is that the keys and values have to be strings or bytes. If you try to use any other type, you get an error.

The `pickle` module can help. It translates almost any type of object into a string suitable for storage in a database, and then translates strings back into objects.

`pickle.dumps` takes an object as a parameter and returns a string representation (`dumps` is short for “dump string”):

```
>>> import pickle
>>> t = [1, 2, 3]
>>> pickle.dumps(t)
b'\x80\x03]q\x00(K\x01K\x02K\x03e.'
```

The format isn’t obvious to human readers; it is meant to be easy for `pickle` to interpret. `pickle.loads` (“load string”) reconstitutes the object:

```
>>> t1 = [1, 2, 3]
>>> s = pickle.dumps(t1)
>>> t2 = pickle.loads(s)
>>> t2
[1, 2, 3]
```

Although the new object has the same value as the old, it is not (in general) the same object:

```
>>> t1 == t2
True
>>> t1 is t2
False
```

In other words, pickling and then unpickling has the same effect as copying the object.

You can use `pickle` to store non-strings in a database. In fact, this combination is so common that it has been encapsulated in a module called `shelve`.

Pipes

Most operating systems provide a command-line interface, also known as a **shell**. Shells usually provide commands to navigate the file system and launch applications. For example, in Unix you can change directories with `cd`, display the contents of a directory with `ls`, and launch a web browser by typing (for example) `firefox`.

Any program that you can launch from the shell can also be launched from Python using a **pipe object**, which represents a running program.

For example, the Unix command `ls -l` normally displays the contents of the current directory in long format. You can launch `ls` with `os.popen`[1]:

```
>>> cmd = 'ls -l'
>>> fp = os.popen(cmd)
```

The argument is a string that contains a shell command. The return value is an object that behaves like an open file. You can read the output from the `ls` process one line at a time with `readline` or get the whole thing at once with `read`:

```
>>> res = fp.read()
```

When you are done, you close the pipe like a file:

```
>>> stat = fp.close()
>>> print(stat)
None
```

The return value is the final status of the `ls` process; `None` means that it ended normally (with no errors).

For example, most Unix systems provide a command called `md5sum` that reads the contents of a file and computes a "checksum". You can read about MD5 at *http://en.wikipedia.org/wiki/Md5*. This command provides an efficient way to check whether two files have the same contents. The probability that different contents yield the same checksum is very small (that is, unlikely to happen before the universe collapses).

You can use a pipe to run `md5sum` from Python and get the result:

```
>>> filename = 'book.tex'
>>> cmd = 'md5sum ' + filename
>>> fp = os.popen(cmd)
>>> res = fp.read()
>>> stat = fp.close()
>>> print(res)
1e0033f0ed0656636de0d75144ba32e0  book.tex
>>> print(stat)
None
```

Writing Modules

Any file that contains Python code can be imported as a module. For example, suppose you have a file named `wc.py` with the following code:

```
def linecount(filename):
    count = 0
    for line in open(filename):
        count += 1
    return count

print(linecount('wc.py'))
```

If you run this program, it reads itself and prints the number of lines in the file, which is 7. You can also import it like this:

```
>>> import wc
7
```

Now you have a module object `wc`:

```
>>> wc
<module 'wc' from 'wc.py'>
```

The module object provides `linecount`:

```
>>> wc.linecount('wc.py')
7
```

So that's how you write modules in Python.

The only problem with this example is that when you import the module it runs the test code at the bottom. Normally when you import a module, it defines new functions but it doesn't run them.

Programs that will be imported as modules often use the following idiom:

```
if __name__ == '__main__':
    print(linecount('wc.py'))
```

`__name__` is a built-in variable that is set when the program starts. If the program is running as a script, `__name__` has the value `'__main__'`; in that case, the test code runs. Otherwise, if the module is being imported, the test code is skipped.

As an exercise, type this example into a file named `wc.py` and run it as a script. Then run the Python interpreter and `import wc`. What is the value of `__name__` when the module is being imported?

Warning: If you import a module that has already been imported, Python does nothing. It does not re-read the file, even if it has changed.

If you want to reload a module, you can use the built-in function `reload`, but it can be

tricky, so the safest thing to do is restart the interpreter and then import the module again.

Debugging

When you are reading and writing files, you might run into problems with whitespace. These errors can be hard to debug because spaces, tabs and newlines are normally invisible:

```
>>> s = '1 2\t 3\n 4'
>>> print(s)
1 2      3
 4
```

The built-in function `repr` can help. It takes any object as an argument and returns a string representation of the object. For strings, it represents whitespace characters with backslash sequences:

```
>>> print(repr(s))
'1 2\t 3\n 4'
```

This can be helpful for debugging.

One other problem you might run into is that different systems use different characters to indicate the end of a line. Some systems use a newline, represented `\n`. Others use a return character, represented `\r`. Some use both. If you move files between different systems, these inconsistencies can cause problems.

For most systems, there are applications to convert from one format to another. You can find them (and read more about this issue) at *http://en.wikipedia.org/wiki/Newline*. Or, of course, you could write one yourself.

Glossary

persistent:

Pertaining to a program that runs indefinitely and keeps at least some of its data in permanent storage.

format operator:

An operator, `%`, that takes a format string and a tuple and generates a string that includes the elements of the tuple formatted as specified by the format string.

format string:

A string, used with the format operator, that contains format sequences.

format sequence:

A sequence of characters in a format string, like `%d`, that specifies how a value should be formatted.

text file:

A sequence of characters stored in permanent storage like a hard drive.

directory:

A named collection of files, also called a folder.

path:

A string that identifies a file.

relative path:

A path that starts from the current directory.

absolute path:

A path that starts from the topmost directory in the file system.

catch:

To prevent an exception from terminating a program by using the `try` and `except` statements.

database:

A file whose contents are organized like a dictionary with keys that correspond to values.

bytes object:

An object similar to a string.

shell:

A program that allows users to type commands and then executes them by starting other programs.

pipe object:

An object that represents a running program, allowing a Python program to run commands and read the results.

Exercises

Exercise 14-1.

Write a function called `sed` that takes as arguments a pattern string, a replacement string, and two filenames; it should read the first file and write the contents into the second file (creating it if necessary). If the pattern string appears anywhere in the file, it should be replaced with the replacement string.

If an error occurs while opening, reading, writing or closing files, your program should catch the exception, print an error message, and exit.

Solution: *http://thinkpython2.com/code/sed.py*.

Exercise 14-2.

If you download my solution to Exercise 12-2 from *http://thinkpython2.com/code/anagram_sets.py*, you'll see that it creates a dictionary that maps from a sorted string of letters to the list of words that can be spelled with those letters. For example, `'opst'` maps to the list `['opts', 'post', 'pots', 'spot', 'stop', 'tops']`.

Write a module that imports `anagram_sets` and provides two new functions: `store_anagrams` should store the anagram dictionary in a "shelf"; `read_anagrams` should look up a word and return a list of its anagrams.

Solution: *http://thinkpython2.com/code/anagram_db.py*

Exercise 14-3.

In a large collection of MP3 files, there may be more than one copy of the same song, stored in different directories or with different filenames. The goal of this exercise is to search for duplicates.

1. Write a program that searches a directory and all of its subdirectories, recursively, and returns a list of complete paths for all files with a given suffix (like `.mp3`). Hint: `os.path` provides several useful functions for manipulating file- and path names.

2. To recognize duplicates, you can use `md5sum` to compute a "checksum" for each files. If two files have the same checksum, they probably have the same contents.

3. To double-check, you can use the Unix command `diff`.

Solution: *http://thinkpython2.com/code/find_duplicates.py*.

[1] `popen` is deprecated now, which means we are supposed to stop using it and start using the `subprocess` module. But for simple cases, I find `subprocess` more complicated than necessary. So I am going to keep using `popen` until they take it away.

Chapter 15. Classes and Objects

At this point you know how to use functions to organize code and built-in types to organize data. The next step is to learn “object-oriented programming”, which uses programmer-defined types to organize both code and data. Object-oriented programming is a big topic; it will take a few chapters to get there.

Code examples from this chapter are available from *http://thinkpython2.com/code/Point1.py*; solutions to the exercises are available from *http://thinkpython2.com/code/Point1_soln.py*.

Programmer-Defined Types

We have used many of Python's built-in types; now we are going to define a new type. As an example, we will create a type called `Point` that represents a point in two-dimensional space.

In mathematical notation, points are often written in parentheses with a comma separating the coordinates. For example, *(0,0)* represents the origin, and *(x,y)* represents the point *x* units to the right and *y* units up from the origin.

There are several ways we might represent points in Python:

- We could store the coordinates separately in two variables, `x` and `y`.
- We could store the coordinates as elements in a list or tuple.
- We could create a new type to represent points as objects.

Creating a new type is more complicated than the other options, but it has advantages that will be apparent soon.

A programmer-defined type is also called a **class**. A class definition looks like this:

```
class Point:
    """Represents a point in 2-D space."""
```

The header indicates that the new class is called `Point`. The body is a docstring that explains what the class is for. You can define variables and methods inside a class definition, but we will get back to that later.

Defining a class named `Point` creates a **class object**:

```
>>> Point
<class '__main__.Point'>
```

Because `Point` is defined at the top level, its "full name" is `__main__.Point`.

The class object is like a factory for creating objects. To create a Point, you call `Point` as if it were a function:

```
>>> blank = Point()
>>> blank
<__main__.Point object at 0xb7e9d3ac>
```

The return value is a reference to a Point object, which we assign to `blank`.

Creating a new object is called **instantiation**, and the object is an **instance** of the class.

When you print an instance, Python tells you what class it belongs to and where it is stored in memory (the prefix `0x` means that the following number is in hexadecimal).

Every object is an instance of some class, so “object” and “instance” are interchangeable. But in this chapter I use “instance” to indicate that I am talking about a programmer-defined type.

Attributes

You can assign values to an instance using dot notation:

```
>>> blank.x = 3.0
>>> blank.y = 4.0
```

This syntax is similar to the syntax for selecting a variable from a module, such as `math.pi` or `string.whitespace`. In this case, though, we are assigning values to named elements of an object. These elements are called **attributes**.

As a noun, “AT-trib-ute” is pronounced with emphasis on the first syllable, as opposed to “a-TRIB-ute”, which is a verb.

The following diagram shows the result of these assignments. A state diagram that shows an object and its attributes is called an **object diagram**; see Figure 15-1.

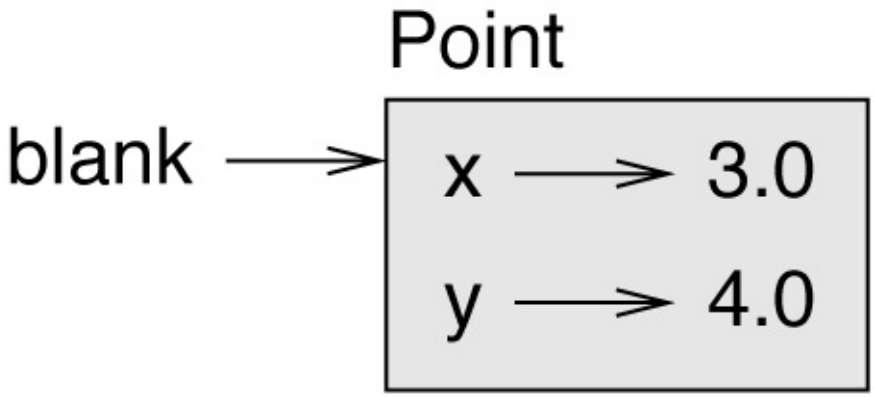


Figure 15-1. Object diagram.

The variable `blank` refers to a Point object, which contains two attributes. Each attribute refers to a floating-point number.

You can read the value of an attribute using the same syntax:

```
>>> blank.y
4.0
>>> x = blank.x
>>> x
3.0
```

The expression `blank.x` means, “Go to the object `blank` refers to and get the value of `x`.” In the example, we assign that value to a variable named `x`. There is no conflict between the variable `x` and the attribute `x`.

You can use dot notation as part of any expression. For example:

```
>>> '(%g, %g)' % (blank.x, blank.y)
'(3.0, 4.0)'
>>> distance = math.sqrt(blank.x**2 + blank.y**2)
>>> distance
5.0
```

You can pass an instance as an argument in the usual way. For example:

```
def print_point(p):
    print('(%g, %g)' % (p.x, p.y))
```

`print_point` takes a point as an argument and displays it in mathematical notation. To invoke it, you can pass `blank` as an argument:

```
>>> print_point(blank)
(3.0, 4.0)
```

Inside the function, `p` is an alias for `blank`, so if the function modifies `p`, `blank` changes.

As an exercise, write a function called `distance_between_points` that takes two Points as arguments and returns the distance between them.

Rectangles

Sometimes it is obvious what the attributes of an object should be, but other times you have to make decisions. For example, imagine you are designing a class to represent rectangles. What attributes would you use to specify the location and size of a rectangle? You can ignore angle; to keep things simple, assume that the rectangle is either vertical or horizontal.

There are at least two possibilities:

- You could specify one corner of the rectangle (or the center), the width, and the height.
- You could specify two opposing corners.

At this point it is hard to say whether either is better than the other, so we'll implement the first one, just as an example.

Here is the class definition:

```
class Rectangle:
    """Represents a rectangle.

    attributes: width, height, corner.
    """
```

The docstring lists the attributes: `width` and `height` are numbers; `corner` is a Point object that specifies the lower-left corner.

To represent a rectangle, you have to instantiate a Rectangle object and assign values to the attributes:

```
box = Rectangle()
box.width = 100.0
box.height = 200.0
box.corner = Point()
box.corner.x = 0.0
box.corner.y = 0.0
```

The expression `box.corner.x` means, "Go to the object `box` refers to and select the attribute named `corner`; then go to that object and select the attribute named `x`."

Figure 15-2 shows the state of this object. An object that is an attribute of another object is **embedded**.

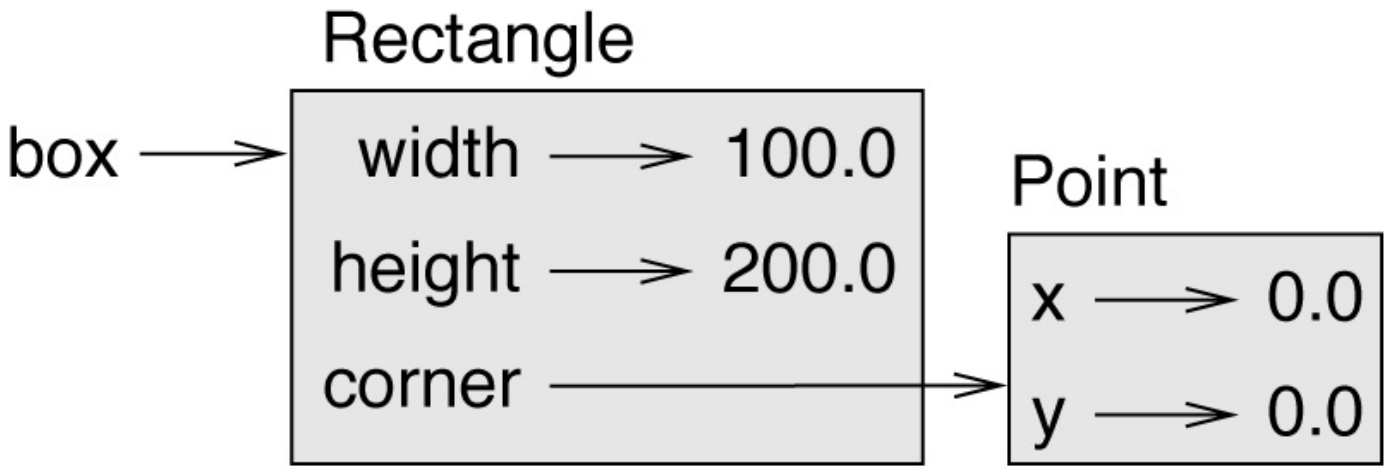


Figure 15-2. Object diagram.

Instances as Return Values

Functions can return instances. For example, `find_center` takes a `Rectangle` as an argument and returns a `Point` that contains the coordinates of the center of the `Rectangle`:

```
def find_center(rect):
    p = Point()
    p.x = rect.corner.x + rect.width/2
    p.y = rect.corner.y + rect.height/2
    return p
```

Here is an example that passes `box` as an argument and assigns the resulting Point to `center`:

```
>>> center = find_center(box)
>>> print_point(center)
(50, 100)
```

Objects Are Mutable

You can change the state of an object by making an assignment to one of its attributes. For example, to change the size of a rectangle without changing its position, you can modify the values of `width` and `height`:

```
box.width = box.width + 50
box.height = box.height + 100
```

You can also write functions that modify objects. For example, `grow_rectangle` takes a Rectangle object and two numbers, `dwidth` and `dheight`, and adds the numbers to the width and height of the rectangle:

```
def grow_rectangle(rect, dwidth, dheight):
    rect.width += dwidth
    rect.height += dheight
```

Here is an example that demonstrates the effect:

```
>>> box.width, box.height
(150.0, 300.0)
>>> grow_rectangle(box, 50, 100)
>>> box.width, box.height
(200.0, 400.0)
```

Inside the function, `rect` is an alias for `box`, so when the function modifies `rect`, `box` changes.

As an exercise, write a function named `move_rectangle` that takes a Rectangle and two numbers named `dx` and `dy`. It should change the location of the rectangle by adding `dx` to the `x` coordinate of `corner` and adding `dy` to the `y` coordinate of `corner`.

Copying

Aliasing can make a program difficult to read because changes in one place might have unexpected effects in another place. It is hard to keep track of all the variables that might refer to a given object.

Copying an object is often an alternative to aliasing. The `copy` module contains a function called `copy` that can duplicate any object:

```
>>> p1 = Point()
>>> p1.x = 3.0
>>> p1.y = 4.0

>>> import copy
>>> p2 = copy.copy(p1)
```

`p1` and `p2` contain the same data, but they are not the same Point:

```
>>> print_point(p1)
(3, 4)
>>> print_point(p2)
(3, 4)
>>> p1 is p2
False
>>> p1 == p2
False
```

The `is` operator indicates that `p1` and `p2` are not the same object, which is what we expected. But you might have expected `==` to yield `True` because these points contain the same data. In that case, you will be disappointed to learn that for instances, the default behavior of the `==` operator is the same as the `is` operator; it checks object identity, not object equivalence. That's because for programmer-defined types, Python doesn't know what should be considered equivalent. At least, not yet.

If you use `copy.copy` to duplicate a Rectangle, you will find that it copies the Rectangle object but not the embedded Point:

```
>>> box2 = copy.copy(box)
>>> box2 is box
False
>>> box2.corner is box.corner
True
```

Figure 15-3 shows what the object diagram looks like. This operation is called a **shallow copy** because it copies the object and any references it contains, but not the embedded objects.

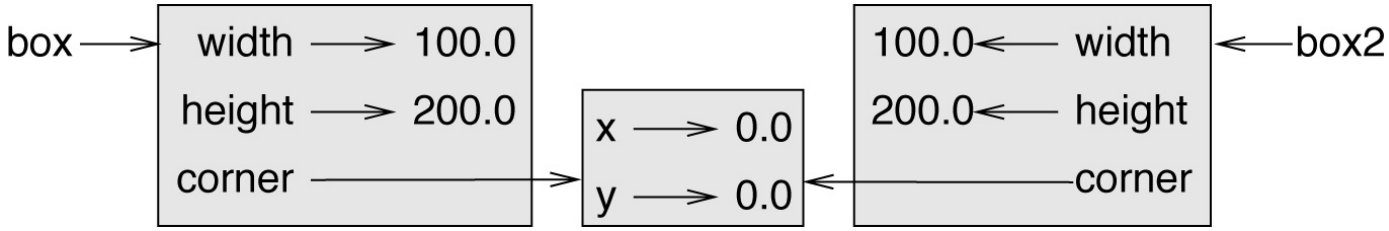


Figure 15-3. Object diagram.

For most applications, this is not what you want. In this example, invoking `grow_rectangle` on one of the Rectangles would not affect the other, but invoking `move_rectangle` on either would affect both! This behavior is confusing and error-prone.

Fortunately, the `copy` module provides a method named `deepcopy` that copies not only the object but also the objects it refers to, and the objects *they* refer to, and so on. You will not be surprised to learn that this operation is called a **deep copy**.

```
>>> box3 = copy.deepcopy(box)
>>> box3 is box
False
>>> box3.corner is box.corner
False
```

`box3` and `box` are completely separate objects.

As an exercise, write a version of `move_rectangle` that creates and returns a new Rectangle instead of modifying the old one.

Debugging

When you start working with objects, you are likely to encounter some new exceptions. If you try to access an attribute that doesn't exist, you get an `AttributeError`:

```
>>> p = Point()
>>> p.x = 3
>>> p.y = 4
>>> p.z
AttributeError: Point instance has no attribute 'z'
```

If you are not sure what type an object is, you can ask:

```
>>> type(p)
<class '__main__.Point'>
```

You can also use `isinstance` to check whether an object is an instance of a class:

```
>>> isinstance(p, Point)
True
```

If you are not sure whether an object has a particular attribute, you can use the built-in function `hasattr`:

```
>>> hasattr(p, 'x')
True
>>> hasattr(p, 'z')
False
```

The first argument can be any object; the second argument is a *string* that contains the name of the attribute.

You can also use a `try` statement to see if the object has the attributes you need:

```
try:
    x = p.x
except AttributeError:
    x = 0
```

This approach can make it easier to write functions that work with different types; more on that topic is coming up in "Polymorphism".

Glossary

class:

A programmer-defined type. A class definition creates a new class object.

class object:

An object that contains information about a programmer-defined type. The class object can be used to create instances of the type.

instance:

An object that belongs to a class.

instantiate:

To create a new object.

attribute:

One of the named values associated with an object.

embedded object:

An object that is stored as an attribute of another object.

shallow copy:

To copy the contents of an object, including any references to embedded objects; implemented by the `copy` function in the `copy` module.

deep copy:

To copy the contents of an object as well as any embedded objects, and any objects embedded in them, and so on; implemented by the `deepcopy` function in the `copy` module.

object diagram:

A diagram that shows objects, their attributes, and the values of the attributes.

Exercises

Exercise 15-1.

Write a definition for a class named `Circle` with attributes `center` and `radius`, where `center` is a Point object and radius is a number.

Instantiate a Circle object that represents a circle with its center at $(150, 100)$ and radius 75.

Write a function named `point_in_circle` that takes a Circle and a Point and returns True if the Point lies in or on the boundary of the circle.

Write a function named `rect_in_circle` that takes a Circle and a Rectangle and returns True if the Rectangle lies entirely in or on the boundary of the circle.

Write a function named `rect_circle_overlap` that takes a Circle and a Rectangle and returns True if any of the corners of the Rectangle fall inside the circle. Or as a more challenging version, return True if any part of the Rectangle falls inside the circle.

Solution: *http://thinkpython2.com/code/Circle.py*.

Exercise 15-2.

Write a function called `draw_rect` that takes a Turtle object and a Rectangle and uses the Turtle to draw the Rectangle. See Chapter 4 for examples using Turtle objects.

Write a function called `draw_circle` that takes a Turtle and a Circle and draws the Circle.

Solution: *http://thinkpython2.com/code/draw.py*.

Chapter 16. Classes and Functions

Now that we know how to create new types, the next step is to write functions that take programmer-defined objects as parameters and return them as results. In this chapter I also present “functional programming style” and two new program development plans.

Code examples from this chapter are available from *http://thinkpython2.com/code/Time1.py*. Solutions to the exercises are at *http://thinkpython2.com/code/Time1_soln.py*.

Time

As another example of a programmer-defined type, we'll define a class called `Time` that records the time of day. The class definition looks like this:

```
class Time:
    """Represents the time of day.

    attributes: hour, minute, second
    """
```

We can create a new `Time` object and assign attributes for hours, minutes, and seconds:

```
time = Time()
time.hour = 11
time.minute = 59
time.second = 30
```

The state diagram for the `Time` object looks like Figure 16-1.

As an exercise, write a function called `print_time` that takes a Time object and prints it in the form `hour:minute:second`. Hint: the format sequence `'%.2d'` prints an integer using at least two digits, including a leading zero if necessary.

Write a boolean function called `is_after` that takes two Time objects, `t1` and `t2`, and returns `True` if `t1` follows `t2` chronologically and `False` otherwise. Challenge: don't use an `if` statement.

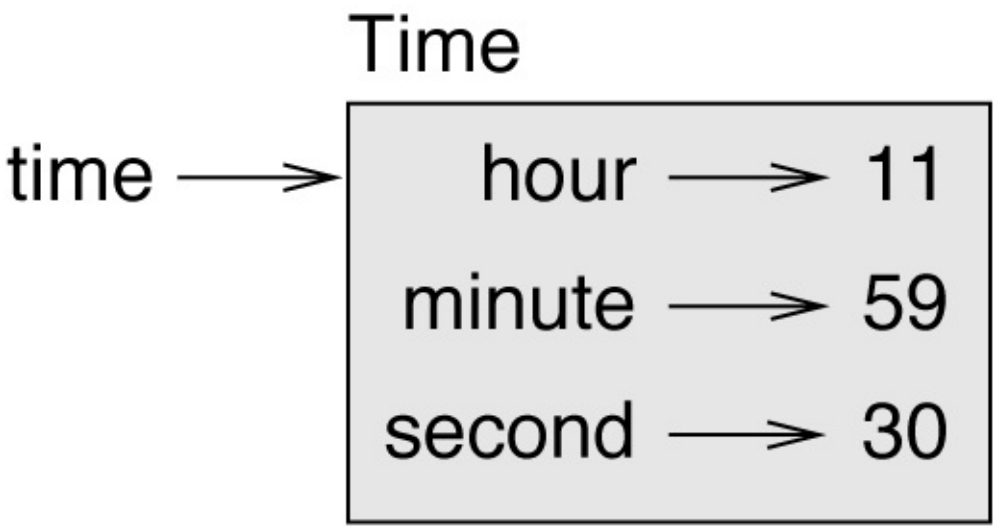


Figure 16-1. Object diagram.

Pure Functions

In the next few sections, we'll write two functions that add time values. They demonstrate two kinds of functions: pure functions and modifiers. They also demonstrate a development plan I'll call **prototype and patch**, which is a way of tackling a complex problem by starting with a simple prototype and incrementally dealing with the complications.

Here is a simple prototype of `add_time`:

```
def add_time(t1, t2):
    sum = Time()
    sum.hour = t1.hour + t2.hour
    sum.minute = t1.minute + t2.minute
    sum.second = t1.second + t2.second
    return sum
```

The function creates a new `Time` object, initializes its attributes, and returns a reference to the new object. This is called a **pure function** because it does not modify any of the objects passed to it as arguments and it has no effect, like displaying a value or getting user input, other than returning a value.

To test this function, I'll create two Time objects: `start` contains the start time of a movie, like *Monty Python and the Holy Grail*, and `duration` contains the runtime of the movie, which is 1 hour 35 minutes.

`add_time` figures out when the movie will be done:

```
>>> start = Time()
>>> start.hour = 9
>>> start.minute = 45
>>> start.second =  0

>>> duration = Time()
>>> duration.hour = 1
>>> duration.minute = 35
>>> duration.second = 0

>>> done = add_time(start, duration)
>>> print_time(done)
10:80:00
```

The result, `10:80:00`, might not be what you were hoping for. The problem is that this function does not deal with cases where the number of seconds or minutes adds up to more than sixty. When that happens, we have to "carry" the extra seconds into the minute column or the extra minutes into the hour column.

Here's an improved version:

```
def add_time(t1, t2):
    sum = Time()
    sum.hour = t1.hour + t2.hour
    sum.minute = t1.minute + t2.minute
    sum.second = t1.second + t2.second

    if sum.second >= 60:
```

```
        sum.second -= 60
        sum.minute += 1

    if sum.minute >= 60:
        sum.minute -= 60
        sum.hour += 1

    return sum
```

Although this function is correct, it is starting to get big. We will see a shorter alternative later.

Modifiers

Sometimes it is useful for a function to modify the objects it gets as parameters. In that case, the changes are visible to the caller. Functions that work this way are called **modifiers**.

`increment`, which adds a given number of seconds to a `Time` object, can be written naturally as a modifier. Here is a rough draft:

```
def increment(time, seconds):
    time.second += seconds

    if time.second >= 60:
        time.second -= 60
        time.minute += 1

    if time.minute >= 60:
        time.minute -= 60
        time.hour += 1
```

The first line performs the basic operation; the remainder deals with the special cases we saw before.

Is this function correct? What happens if `seconds` is much greater than 60?

In that case, it is not enough to carry once; we have to keep doing it until `time.second` is less than 60. One solution is to replace the `if` statements with `while` statements. That would make the function correct, but not very efficient. As an exercise, write a correct version of `increment` that doesn't contain any loops.

Anything that can be done with modifiers can also be done with pure functions. In fact, some programming languages only allow pure functions. There is some evidence that programs that use pure functions are faster to develop and less error-prone than programs that use modifiers. But modifiers are convenient at times, and functional programs tend to be less efficient.

In general, I recommend that you write pure functions whenever it is reasonable and resort to modifiers only if there is a compelling advantage. This approach might be called a **functional programming style**.

As an exercise, write a "pure" version of `increment` that creates and returns a new Time object rather than modifying the parameter.

Prototyping versus Planning

The development plan I am demonstrating is called “prototype and patch”. For each function, I wrote a prototype that performed the basic calculation and then tested it, patching errors along the way.

This approach can be effective, especially if you don’t yet have a deep understanding of the problem. But incremental corrections can generate code that is unnecessarily complicated (since it deals with many special cases) and unreliable (since it is hard to know if you have found all the errors).

An alternative is **designed development**, in which high-level insight into the problem can make the programming much easier. In this case, the insight is that a Time object is really a three-digit number in base 60 (see *http://en.wikipedia.org/wiki/Sexagesimal*.)! The `second` attribute is the “ones column”, the `minute` attribute is the “sixties column”, and the `hour` attribute is the “thirty-six hundreds column”.

When we wrote `add_time` and `increment`, we were effectively doing addition in base 60, which is why we had to carry from one column to the next.

This observation suggests another approach to the whole problem — we can convert Time objects to integers and take advantage of the fact that the computer knows how to do integer arithmetic.

Here is a function that converts Times to integers:

```
def time_to_int(time):
    minutes = time.hour * 60 + time.minute
    seconds = minutes * 60 + time.second
    return seconds
```

And here is a function that converts an integer to a Time (recall that `divmod` divides the first argument by the second and returns the quotient and remainder as a tuple):

```
def int_to_time(seconds):
    time = Time()
    minutes, time.second = divmod(seconds, 60)
    time.hour, time.minute = divmod(minutes, 60)
    return time
```

You might have to think a bit, and run some tests, to convince yourself that these functions are correct. One way to test them is to check that `time_to_int(int_to_time(x)) == x` for many values of `x`. This is an example of a consistency check.

Once you are convinced they are correct, you can use them to rewrite `add_time`:

```
def add_time(t1, t2):
    seconds = time_to_int(t1) + time_to_int(t2)
    return int_to_time(seconds)
```

This version is shorter than the original, and easier to verify. As an exercise, rewrite

`increment` using `time_to_int` and `int_to_time`.

In some ways, converting from base 60 to base 10 and back is harder than just dealing with times. Base conversion is more abstract; our intuition for dealing with time values is better.

But if we have the insight to treat times as base 60 numbers and make the investment of writing the conversion functions (`time_to_int` and `int_to_time`), we get a program that is shorter, easier to read and debug, and more reliable.

It is also easier to add features later. For example, imagine subtracting two Times to find the duration between them. The naive approach would be to implement subtraction with borrowing. Using the conversion functions would be easier and more likely to be correct.

Ironically, sometimes making a problem harder (or more general) makes it easier (because there are fewer special cases and fewer opportunities for error).

Debugging

A Time object is well-formed if the values of `minute` and `second` are between 0 and 60 (including 0 but not 60) and if `hour` is positive. `hour` and `minute` should be integral values, but we might allow `second` to have a fraction part.

Requirements like these are called **invariants** because they should always be true. To put it a different way, if they are not true, something has gone wrong.

Writing code to check invariants can help detect errors and find their causes. For example, you might have a function like `valid_time` that takes a Time object and returns `False` if it violates an invariant:

```
def valid_time(time):
    if time.hour < 0 or time.minute < 0 or time.second < 0:
        return False
    if time.minute >= 60 or time.second >= 60:
        return False
    return True
```

At the beginning of each function you could check the arguments to make sure they are valid:

```
def add_time(t1, t2):
    if not valid_time(t1) or not valid_time(t2):
        raise ValueError('invalid Time object in add_time')
    seconds = time_to_int(t1) + time_to_int(t2)
    return int_to_time(seconds)
```

Or you could use an **assert statement**, which checks a given invariant and raises an exception if it fails:

```
def add_time(t1, t2):
    assert valid_time(t1) and valid_time(t2)
    seconds = time_to_int(t1) + time_to_int(t2)
    return int_to_time(seconds)
```

`assert` statements are useful because they distinguish code that deals with normal conditions from code that checks for errors.

Glossary

prototype and patch:

A development plan that involves writing a rough draft of a program, testing, and correcting errors as they are found.

designed development:

A development plan that involves high-level insight into the problem and more planning than incremental development or prototype development.

pure function:

A function that does not modify any of the objects it receives as arguments. Most pure functions are fruitful.

modifier:

A function that changes one or more of the objects it receives as arguments. Most modifiers are void; that is, they return `None`.

functional programming style:

A style of program design in which the majority of functions are pure.

invariant:

A condition that should always be true during the execution of a program.

assert statement:

A statement that check a condition and raises an exception if it fails.

Exercises

Code examples from this chapter are available from *http://thinkpython2.com/code/Time1.py*; solutions to the exercises are available from *http://thinkpython2.com/code/Time1_soln.py*.

Exercise 16-1.

Write a function called `mul_time` that takes a Time object and a number and returns a new Time object that contains the product of the original Time and the number.

Then use `mul_time` to write a function that takes a Time object that represents the finishing time in a race, and a number that represents the distance, and returns a Time object that represents the average pace (time per mile).

Exercise 16-2.

The `datetime` module provides `time` objects that are similar to the Time objects in this chapter, but they provide a rich set of methods and operators. Read the documentation at *http://docs.python.org/3/library/datetime.html*.

1. Use the `datetime` module to write a program that gets the current date and prints the day of the week.

2. Write a program that takes a birthday as input and prints the user's age and the number of days, hours, minutes and seconds until their next birthday.

3. For two people born on different days, there is a day when one is twice as old as the other. That's their Double Day. Write a program that takes two birthdays and computes their Double Day.

4. For a little more challenge, write the more general version that computes the day when one person is *n* times older than the other.

Solution: *http://thinkpython2.com/code/double.py*.

Chapter 17. Classes and Methods

Although we are using some of Python's object-oriented features, the programs from the last two chapters are not really object-oriented because they don't represent the relationships between programmer-defined types and the functions that operate on them. The next step is to transform those functions into methods that make the relationships explicit.

Code examples from this chapter are available from *http://thinkpython2.com/code/Time2.py*, and solutions to the exercises are in *http://thinkpython2.com/code/Point2_soln.py*.

Object-Oriented Features

Python is an **object-oriented programming language**, which means that it provides features that support object-oriented programming, which has these defining characteristics:

- Programs include class and method definitions.
- Most of the computation is expressed in terms of operations on objects.
- Objects often represent things in the real world, and methods often correspond to the ways things in the real world interact.

For example, the `Time` class defined in Chapter 16 corresponds to the way people record the time of day, and the functions we defined correspond to the kinds of things people do with times. Similarly, the `Point` and `Rectangle` classes in Chapter 15 correspond to the mathematical concepts of a point and a rectangle.

So far, we have not taken advantage of the features Python provides to support object-oriented programming. These features are not strictly necessary; most of them provide alternative syntax for things we have already done. But in many cases, the alternative is more concise and more accurately conveys the structure of the program.

For example, in `Time1.py` there is no obvious connection between the class definition and the function definitions that follow. With some examination, it is apparent that every function takes at least one `Time` object as an argument.

This observation is the motivation for **methods**; a method is a function that is associated with a particular class. We have seen methods for strings, lists, dictionaries and tuples. In this chapter, we will define methods for programmer-defined types.

Methods are semantically the same as functions, but there are two syntactic differences:

- Methods are defined inside a class definition in order to make the relationship between the class and the method explicit.
- The syntax for invoking a method is different from the syntax for calling a function.

In the next few sections, we will take the functions from the previous two chapters and transform them into methods. This transformation is purely mechanical; you can do it by following a sequence of steps. If you are comfortable converting from one form to another, you will be able to choose the best form for whatever you are doing.

Printing Objects

In Chapter 16, we defined a class named `Time` and in "Time", you wrote a function named `print_time`:

```
class Time:
    """Represents the time of day."""

def print_time(time):
    print('%.2d:%.2d:%.2d' % (time.hour, time.minute, time.second))
```

To call this function, you have to pass a `Time` object as an argument:

```
>>> start = Time()
>>> start.hour = 9
>>> start.minute = 45
>>> start.second = 00
>>> print_time(start)
09:45:00
```

To make `print_time` a method, all we have to do is move the function definition inside the class definition. Notice the change in indentation.

```
class Time:
    def print_time(time):
        print('%.2d:%.2d:%.2d' % (time.hour, time.minute, time.second))
```

Now there are two ways to call `print_time`. The first (and less common) way is to use function syntax:

```
>>> Time.print_time(start)
09:45:00
```

In this use of dot notation, `Time` is the name of the class, and `print_time` is the name of the method. `start` is passed as a parameter.

The second (and more concise) way is to use method syntax:

```
>>> start.print_time()
09:45:00
```

In this use of dot notation, `print_time` is the name of the method (again), and `start` is the object the method is invoked on, which is called the **subject**. Just as the subject of a sentence is what the sentence is about, the subject of a method invocation is what the method is about.

Inside the method, the subject is assigned to the first parameter, so in this case `start` is assigned to `time`.

By convention, the first parameter of a method is called `self`, so it would be more common to write `print_time` like this:

```
class Time:
    def print_time(self):
        print('%.2d:%.2d:%.2d' % (self.hour, self.minute, self.second))
```

The reason for this convention is an implicit metaphor:

- The syntax for a function call, `print_time(start)`, suggests that the function is the active agent. It says something like, “Hey `print_time`! Here’s an object for you to print.”

- In object-oriented programming, the objects are the active agents. A method invocation like `start.print_time()` says “Hey `start`! Please print yourself.”

This change in perspective might be more polite, but it is not obvious that it is useful. In the examples we have seen so far, it may not be. But sometimes shifting responsibility from the functions onto the objects makes it possible to write more versatile functions (or methods), and makes it easier to maintain and reuse code.

As an exercise, rewrite `time_to_int` (from “Prototyping versus Planning”) as a method. You might be tempted to rewrite `int_to_time` as a method, too, but that doesn’t really make sense because there would be no object to invoke it on.

Another Example

Here's a version of `increment` (from "Modifiers") rewritten as a method:

```
# inside class Time:

    def increment(self, seconds):
        seconds += self.time_to_int()
        return int_to_time(seconds)
```

This version assumes that `time_to_int` is written as a method. Also, note that it is a pure function, not a modifier.

Here's how you would invoke `increment`:

```
>>> start.print_time()
09:45:00
>>> end = start.increment(1337)
>>> end.print_time()
10:07:17
```

The subject, `start`, gets assigned to the first parameter, `self`. The argument, `1337`, gets assigned to the second parameter, `seconds`.

This mechanism can be confusing, especially if you make an error. For example, if you invoke `increment` with two arguments, you get:

```
>>> end = start.increment(1337, 460)
TypeError: increment() takes 2 positional arguments but 3 were given
```

The error message is initially confusing, because there are only two arguments in parentheses. But the subject is also considered an argument, so all together that's three.

By the way, a **positional argument** is an argument that doesn't have a parameter name; that is, it is not a keyword argument. In this function call:

```
sketch(parrot, cage, dead=True)
```

`parrot` and `cage` are positional, and `dead` is a keyword argument.

A More Complicated Example

Rewriting `is_after` (from "Time") is slightly more complicated because it takes two Time objects as parameters. In this case it is conventional to name the first parameter `self` and the second parameter `other`:

```
# inside class Time:

    def is_after(self, other):
        return self.time_to_int() > other.time_to_int()
```

To use this method, you have to invoke it on one object and pass the other as an argument:

```
>>> end.is_after(start)
True
```

One nice thing about this syntax is that it almost reads like English: "end is after start?"

The init Method

The init method (short for “initialization”) is a special method that gets invoked when an object is instantiated. Its full name is `__init__` (two underscore characters, followed by `init`, and then two more underscores). An init method for the `Time` class might look like this:

```
# inside class Time:

    def __init__(self, hour=0, minute=0, second=0):
        self.hour = hour
        self.minute = minute
        self.second = second
```

It is common for the parameters of `__init__` to have the same names as the attributes. The statement

```
        self.hour = hour
```

stores the value of the parameter `hour` as an attribute of `self`.

The parameters are optional, so if you call `Time` with no arguments, you get the default values:

```
>>> time = Time()
>>> time.print_time()
00:00:00
```

If you provide one argument, it overrides `hour`:

```
>>> time = Time (9)
>>> time.print_time()
09:00:00
```

If you provide two arguments, they override `hour` and `minute`:

```
>>> time = Time(9, 45)
>>> time.print_time()
09:45:00
```

And if you provide three arguments, they override all three default values.

As an exercise, write an init method for the `Point` class that takes `x` and `y` as optional parameters and assigns them to the corresponding attributes.

The __str__ Method

__str__ is a special method, like __init__, that is supposed to return a string representation of an object.

For example, here is a str method for Time objects:

```
# inside class Time:

    def __str__(self):
        return '%.2d:%.2d:%.2d' % (self.hour, self.minute, self.second)
```

When you print an object, Python invokes the str method:

```
>>> time = Time(9, 45)
>>> print(time)
09:45:00
```

When I write a new class, I almost always start by writing __init__, which makes it easier to instantiate objects, and __str__, which is useful for debugging.

As an exercise, write a str method for the Point class. Create a Point object and print it.

Operator Overloading

By defining other special methods, you can specify the behavior of operators on programmer-defined types. For example, if you define a method named `__add__` for the `Time` class, you can use the `+` operator on Time objects.

Here is what the definition might look like:

```
# inside class Time:

    def __add__(self, other):
        seconds = self.time_to_int() + other.time_to_int()
        return int_to_time(seconds)
```

And here is how you could use it:

```
>>> start = Time(9, 45)
>>> duration = Time(1, 35)
>>> print(start + duration)
11:20:00
```

When you apply the `+` operator to Time objects, Python invokes `__add__`. When you print the result, Python invokes `__str__`. So there is a lot happening behind the scenes!

Changing the behavior of an operator so that it works with programmer-defined types is called **operator overloading**. For every operator in Python there is a corresponding special method, like `__add__`. For more details, see *http://docs.python.org/3/reference/datamodel.html#specialnames*.

As an exercise, write an `add` method for the Point class.

Type-Based Dispatch

In the previous section we added two Time objects, but you also might want to add an integer to a Time object. The following is a version of `__add__` that checks the type of `other` and invokes either `add_time` or `increment`:

```
# inside class Time:

    def __add__(self, other):
        if isinstance(other, Time):
            return self.add_time(other)
        else:
            return self.increment(other)

    def add_time(self, other):
        seconds = self.time_to_int() + other.time_to_int()
        return int_to_time(seconds)

    def increment(self, seconds):
        seconds += self.time_to_int()
        return int_to_time(seconds)
```

The built-in function `isinstance` takes a value and a class object, and returns `True` if the value is an instance of the class.

If `other` is a Time object, `__add__` invokes `add_time`. Otherwise it assumes that the parameter is a number and invokes `increment`. This operation is called a **type-based dispatch** because it dispatches the computation to different methods based on the type of the arguments.

Here are examples that use the `+` operator with different types:

```
>>> start = Time(9, 45)
>>> duration = Time(1, 35)
>>> print(start + duration)
11:20:00
>>> print(start + 1337)
10:07:17
```

Unfortunately, this implementation of addition is not commutative. If the integer is the first operand, you get

```
>>> print(1337 + start)
TypeError: unsupported operand type(s) for +: 'int' and 'instance'
```

The problem is, instead of asking the Time object to add an integer, Python is asking an integer to add a Time object, and it doesn't know how. But there is a clever solution for this problem: the special method `__radd__`, which stands for "right-side add". This method is invoked when a Time object appears on the right side of the `+` operator. Here's the definition:

```
# inside class Time:

    def __radd__(self, other):
        return self.__add__(other)
```

And here's how it's used:

```
>>> print(1337 + start)
10:07:17
```

As an exercise, write an `add` method for Points that works with either a Point object or a tuple:

- If the second operand is a Point, the method should return a new Point whose *x* coordinate is the sum of the *x* coordinates of the operands, and likewise for the *y* coordinates.
- If the second operand is a tuple, the method should add the first element of the tuple to the *x* coordinate and the second element to the *y* coordinate, and return a new Point with the result.

Polymorphism

Type-based dispatch is useful when it is necessary, but (fortunately) it is not always necessary. Often you can avoid it by writing functions that work correctly for arguments with different types.

Many of the functions we wrote for strings also work for other sequence types. For example, in “Dictionary as a Collection of Counters” we used `histogram` to count the number of times each letter appears in a word:

```
def histogram(s):
    d = dict()
    for c in s:
        if c not in d:
            d[c] = 1
        else:
            d[c] = d[c]+1
    return d
```

This function also works for lists, tuples, and even dictionaries, as long as the elements of `s` are hashable, so they can be used as keys in `d`:

```
>>> t = ['spam', 'egg', 'spam', 'spam', 'bacon', 'spam']
>>> histogram(t)
{'bacon': 1, 'egg': 1, 'spam': 4}
```

Functions that work with several types are called **polymorphic**. Polymorphism can facilitate code reuse. For example, the built-in function `sum`, which adds the elements of a sequence, works as long as the elements of the sequence support addition.

Since Time objects provide an `add` method, they work with `sum`:

```
>>> t1 = Time(7, 43)
>>> t2 = Time(7, 41)
>>> t3 = Time(7, 37)
>>> total = sum([t1, t2, t3])
>>> print(total)
23:01:00
```

In general, if all of the operations inside a function work with a given type, the function works with that type.

The best kind of polymorphism is the unintentional kind, where you discover that a function you already wrote can be applied to a type you never planned for.

Interface and Implementation

One of the goals of object-oriented design is to make software more maintainable, which means that you can keep the program working when other parts of the system change, and modify the program to meet new requirements.

A design principle that helps achieve that goal is to keep interfaces separate from implementations. For objects, that means that the methods a class provides should not depend on how the attributes are represented.

For example, in this chapter we developed a class that represents a time of day. Methods provided by this class include `time_to_int`, `is_after`, and `add_time`.

We could implement those methods in several ways. The details of the implementation depend on how we represent time. In this chapter, the attributes of a Time object are `hour`, `minute`, and `second`.

As an alternative, we could replace these attributes with a single integer representing the number of seconds since midnight. This implementation would make some methods, like `is_after`, easier to write, but it makes other methods harder.

After you deploy a new class, you might discover a better implementation. If other parts of the program are using your class, it might be time-consuming and error-prone to change the interface.

But if you designed the interface carefully, you can change the implementation without changing the interface, which means that other parts of the program don’t have to change.

Debugging

It is legal to add attributes to objects at any point in the execution of a program, but if you have objects with the same type that don't have the same attributes, it is easy to make mistakes. It is considered a good idea to initialize all of an object's attributes in the init method.

If you are not sure whether an object has a particular attribute, you can use the built-in function `hasattr` (see "Debugging").

Another way to access attributes is the built-in function `vars`, which takes an object and returns a dictionary that maps from attribute names (as strings) to their values:

```
>>> p = Point(3, 4)
>>> vars(p)
{'y': 4, 'x': 3}
```

For purposes of debugging, you might find it useful to keep this function handy:

```
def print_attributes(obj):
    for attr in vars(obj):
        print(attr, getattr(obj, attr))
```

`print_attributes` traverses the dictionary and prints each attribute name and its corresponding value.

The built-in function `getattr` takes an object and an attribute name (as a string) and returns the attribute's value.

Glossary

object-oriented language:

A language that provides features, such as programmer-defined types and methods, that facilitate object-oriented programming.

object-oriented programming:

A style of programming in which data and the operations that manipulate it are organized into classes and methods.

method:

A function that is defined inside a class definition and is invoked on instances of that class.

subject:

The object a method is invoked on.

positional argument:

An argument that does not include a parameter name, so it is not a keyword argument.

operator overloading:

Changing the behavior of an operator like + so it works with a programmer-defined type.

type-based dispatch:

A programming pattern that checks the type of an operand and invokes different functions for different types.

polymorphic:

Pertaining to a function that can work with more than one type.

information hiding:

The principle that the interface provided by an object should not depend on its implementation, in particular the representation of its attributes.

Exercises

Exercise 17-1.

Download the code from this chapter from *http://thinkpython2.com/code/Time2.py*. Change the attributes of `Time` to be a single integer representing seconds since midnight. Then modify the methods (and the function `int_to_time`) to work with the new implementation. You should not have to modify the test code in `main`. When you are done, the output should be the same as before.

Solution: *http://thinkpython2.com/code/Time2_soln.py*

Exercise 17-2.

This exercise is a cautionary tale about one of the most common, and difficult to find, errors in Python. Write a definition for a class named `Kangaroo` with the following methods:

1. An `__init__` method that initializes an attribute named `pouch_contents` to an empty list.

2. A method named `put_in_pouch` that takes an object of any type and adds it to `pouch_contents`.

3. A `__str__` method that returns a string representation of the `Kangaroo` object and the contents of the pouch.

Test your code by creating two `Kangaroo` objects, assigning them to variables named `kanga` and `roo`, and then adding `roo` to the contents of `kanga`'s pouch.

Download *http://thinkpython2.com/code/BadKangaroo.py*. It contains a solution to the previous problem with one big, nasty bug. Find and fix the bug.

If you get stuck, you can download *http://thinkpython2.com/code/GoodKangaroo.py*, which explains the problem and demonstrates a solution.

Chapter 18. Inheritance

The language feature most often associated with object-oriented programming is **inheritance**. Inheritance is the ability to define a new class that is a modified version of an existing class. In this chapter I demonstrate inheritance using classes that represent playing cards, decks of cards, and poker hands.

If you don't play poker, you can read about it at *http://en.wikipedia.org/wiki/Poker*, but you don't have to; I'll tell you what you need to know for the exercises.

Code examples from this chapter are available from *http://thinkpython2.com/code/Card.py*.

Card Objects

There are 52 cards in a deck, each of which belongs to 1 of 4 suits and 1 of 13 ranks. The suits are Spades, Hearts, Diamonds, and Clubs (in descending order in bridge). The ranks are Ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, and King. Depending on the game that you are playing, an Ace may be higher than King or lower than 2.

If we want to define a new object to represent a playing card, it is obvious what the attributes should be: `rank` and `suit`. It is not as obvious what type the attributes should be. One possibility is to use strings containing words like `'Spade'` for suits and `'Queen'` for ranks. One problem with this implementation is that it would not be easy to compare cards to see which had a higher rank or suit.

An alternative is to use integers to **encode** the ranks and suits. In this context, “encode” means that we are going to define a mapping between numbers and suits, or between numbers and ranks. This kind of encoding is not meant to be a secret (that would be “encryption”).

For example, this table shows the suits and the corresponding integer codes:

Spades	↦	3
Hearts	↦	2
Diamonds	↦	1
Clubs	↦	0

This code makes it easy to compare cards; because higher suits map to higher numbers, we can compare suits by comparing their codes.

The mapping for ranks is fairly obvious; each of the numerical ranks maps to the corresponding integer, and for face cards:

Jack	↦	11
Queen	↦	12
King	↦	13

I am using the ↦ symbol to make it clear that these mappings are not part of the Python program. They are part of the program design, but they don’t appear explicitly in the code.

The class definition for `Card` looks like this:

```
class Card:
    """Represents a standard playing card."""

    def __init__(self, suit=0, rank=2):
        self.suit = suit
        self.rank = rank
```

As usual, the init method takes an optional parameter for each attribute. The default card is

the 2 of Clubs.

To create a Card, you call `Card` with the suit and rank of the card you want:

```
queen_of_diamonds = Card(1, 12)
```

Class Attributes

In order to print Card objects in a way that people can easily read, we need a mapping from the integer codes to the corresponding ranks and suits. A natural way to do that is with lists of strings. We assign these lists to **class attributes**:

```
# inside class Card:

    suit_names = ['Clubs', 'Diamonds', 'Hearts', 'Spades']
    rank_names = [None, 'Ace', '2', '3', '4', '5', '6', '7',
              '8', '9', '10', 'Jack', 'Queen', 'King']

    def __str__(self):
        return '%s of %s' % (Card.rank_names[self.rank],
                             Card.suit_names[self.suit])
```

Variables like `suit_names` and `rank_names`, which are defined inside a class but outside of any method, are called class attributes because they are associated with the class object `Card`.

This term distinguishes them from variables like `suit` and `rank`, which are called **instance attributes** because they are associated with a particular instance.

Both kinds of attribute are accessed using dot notation. For example, in `__str__`, `self` is a Card object, and `self.rank` is its rank. Similarly, `Card` is a class object, and `Card.rank_names` is a list of strings associated with the class.

Every card has its own `suit` and `rank`, but there is only one copy of `suit_names` and `rank_names`.

Putting it all together, the expression `Card.rank_names[self.rank]` means “use the attribute `rank` from the object `self` as an index into the list `rank_names` from the class `Card`, and select the appropriate string.”

The first element of `rank_names` is `None` because there is no card with rank zero. By including `None` as a place-keeper, we get a mapping with the nice property that the index 2 maps to the string `'2'`, and so on. To avoid this tweak, we could have used a dictionary instead of a list.

With the methods we have so far, we can create and print cards:

```
>>> card1 = Card(2, 11)
>>> print(card1)
Jack of Hearts
```

Figure 18-1 is a diagram of the `Card` class object and one Card instance. `Card` is a class object; its type is `type`. `card1` is an instance of `Card`, so its type is `Card`. To save space, I didn’t draw the contents of `suit_names` and `rank_names`.

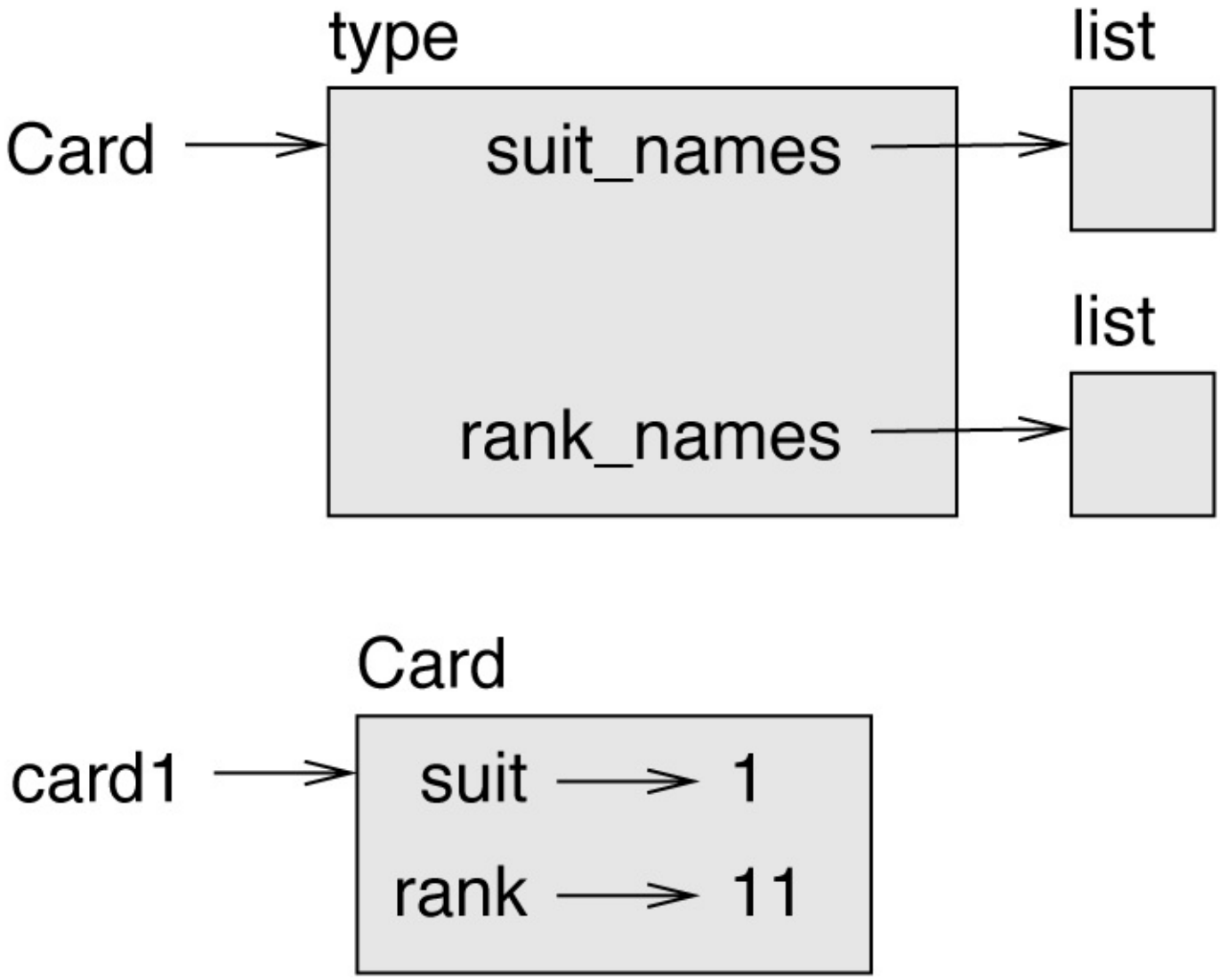


Figure 18-1. Object diagram.

Comparing Cards

For built-in types, there are relational operators (<, >, ==, etc.) that compare values and determine when one is greater than, less than, or equal to another. For programmer-defined types, we can override the behavior of the built-in operators by providing a method named `__lt__`, which stands for “less than”.

`__lt__` takes two parameters, `self` and `other`, and `True` if `self` is strictly less than `other`.

The correct ordering for cards is not obvious. For example, which is better, the 3 of Clubs or the 2 of Diamonds? One has a higher rank, but the other has a higher suit. In order to compare cards, you have to decide whether rank or suit is more important.

The answer might depend on what game you are playing, but to keep things simple, we’ll make the arbitrary choice that suit is more important, so all of the Spades outrank all of the Diamonds, and so on.

With that decided, we can write `__lt__`:

```
# inside class Card:

    def __lt__(self, other):
        # check the suits
        if self.suit < other.suit: return True
        if self.suit > other.suit: return False

        # suits are the same… check ranks
        return self.rank < other.rank
```

You can write this more concisely using tuple comparison:

```
# inside class Card:

    def __lt__(self, other):
        t1 = self.suit, self.rank
        t2 = other.suit, other.rank
        return t1 < t2
```

As an exercise, write an `__lt__` method for Time objects. You can use tuple comparison, but you also might consider comparing integers.

Decks

Now that we have Cards, the next step is to define Decks. Since a deck is made up of cards, it is natural for each Deck to contain a list of cards as an attribute.

The following is a class definition for `Deck`. The init method creates the attribute `cards` and generates the standard set of 52 cards:

```
class Deck:

    def __init__(self):
        self.cards = []
        for suit in range(4):
            for rank in range(1, 14):
                card = Card(suit, rank)
                self.cards.append(card)
```

The easiest way to populate the deck is with a nested loop. The outer loop enumerates the suits from 0 to 3. The inner loop enumerates the ranks from 1 to 13. Each iteration creates a new Card with the current suit and rank, and appends it to `self.cards`.

Printing the Deck

Here is a `__str__` method for `Deck`:

```
#inside class Deck:

    def __str__(self):
        res = []
        for card in self.cards:
            res.append(str(card))
        return '\n'.join(res)
```

This method demonstrates an efficient way to accumulate a large string: building a list of strings and then using the string method `join`. The built-in function `str` invokes the `__str__` method on each card and returns the string representation.

Since we invoke `join` on a newline character, the cards are separated by newlines. Here's what the result looks like:

```
>>> deck = Deck()
>>> print(deck)
Ace of Clubs
2 of Clubs
3 of Clubs…
10 of Spades
Jack of Spades
Queen of Spades
King of Spades
```

Even though the result appears on 52 lines, it is one long string that contains newlines.

Add, Remove, Shuffle and Sort

To deal cards, we would like a method that removes a card from the deck and returns it. The list method `pop` provides a convenient way to do that:

```
#inside class Deck:

    def pop_card(self):
        return self.cards.pop()
```

Since `pop` removes the *last* card in the list, we are dealing from the bottom of the deck.

To add a card, we can use the list method `append`:

```
#inside class Deck:

    def add_card(self, card):
        self.cards.append(card)
```

A method like this that uses another method without doing much work is sometimes called a **veneer**. The metaphor comes from woodworking, where a veneer is a thin layer of good quality wood glued to the surface of a cheaper piece of wood to improve the appearance.

In this case `add_card` is a "thin" method that expresses a list operation in terms appropriate for decks. It improves the appearance, or interface, of the implementation.

As another example, we can write a Deck method named `shuffle` using the function `shuffle` from the `random` module:

```
# inside class Deck:

    def shuffle(self):
        random.shuffle(self.cards)
```

Don't forget to import `random`.

As an exercise, write a Deck method named `sort` that uses the list method `sort` to sort the cards in a `Deck`. `sort` uses the `__lt__` method we defined to determine the order.

Inheritance

Inheritance is the ability to define a new class that is a modified version of an existing class. As an example, let's say we want a class to represent a "hand", that is, the cards held by one player. A hand is similar to a deck: both are made up of a collection of cards, and both require operations like adding and removing cards.

A hand is also different from a deck; there are operations we want for hands that don't make sense for a deck. For example, in poker we might compare two hands to see which one wins. In bridge, we might compute a score for a hand in order to make a bid.

This relationship between classes — similar, but different — lends itself to inheritance. To define a new class that inherits from an existing class, you put the name of the existing class in parentheses:

```
class Hand(Deck):
    """Represents a hand of playing cards."""
```

This definition indicates that `Hand` inherits from `Deck`; that means we can use methods like `pop_card` and `add_card` for Hands as well as Decks.

When a new class inherits from an existing one, the existing one is called the **parent** and the new class is called the **child**.

In this example, `Hand` inherits `__init__` from `Deck`, but it doesn't really do what we want: instead of populating the hand with 52 new cards, the init method for Hands should initialize `cards` with an empty list.

If we provide an init method in the `Hand` class, it overrides the one in the `Deck` class:

```
# inside class Hand:

    def __init__(self, label=''):
        self.cards = []
        self.label = label
```

When you create a Hand, Python invokes this init method, not the one in `Deck`.

```
>>> hand = Hand('new hand')
>>> hand.cards
[]
>>> hand.label
'new hand'
```

The other methods are inherited from `Deck`, so we can use `pop_card` and `add_card` to deal a card:

```
>>> deck = Deck()
>>> card = deck.pop_card()
>>> hand.add_card(card)
>>> print(hand)
King of Spades
```

A natural next step is to encapsulate this code in a method called `move_cards`:

```
#inside class Deck:

    def move_cards(self, hand, num):
        for i in range(num):
            hand.add_card(self.pop_card())
```

`move_cards` takes two arguments, a Hand object and the number of cards to deal. It modifies both `self` and `hand`, and returns `None`.

In some games, cards are moved from one hand to another, or from a hand back to the deck. You can use `move_cards` for any of these operations: `self` can be either a Deck or a Hand, and `hand`, despite the name, can also be a `Deck`.

Inheritance is a useful feature. Some programs that would be repetitive without inheritance can be written more elegantly with it. Inheritance can facilitate code reuse, since you can customize the behavior of parent classes without having to modify them. In some cases, the inheritance structure reflects the natural structure of the problem, which makes the design easier to understand.

On the other hand, inheritance can make programs difficult to read. When a method is invoked, it is sometimes not clear where to find its definition. The relevant code may be spread across several modules. Also, many of the things that can be done using inheritance can be done as well or better without it.

Class Diagrams

So far we have seen stack diagrams, which show the state of a program, and object diagrams, which show the attributes of an object and their values. These diagrams represent a snapshot in the execution of a program, so they change as the program runs.

They are also highly detailed; for some purposes, too detailed. A class diagram is a more abstract representation of the structure of a program. Instead of showing individual objects, it shows classes and the relationships between them.

There are several kinds of relationship between classes:

- Objects in one class might contain references to objects in another class. For example, each Rectangle contains a reference to a Point, and each Deck contains references to many Cards. This kind of relationship is called **HAS-A**, as in, "a Rectangle has a Point."

- One class might inherit from another. This relationship is called **IS-A**, as in, "a Hand is a kind of a Deck."

- One class might depend on another in the sense that objects in one class take objects in the second class as parameters, or use objects in the second class as part of a computation. This kind of relationship is called a **dependency**.

A **class diagram** is a graphical representation of these relationships. For example, Figure 18-2 shows the relationships between `Card`, `Deck` and `Hand`.

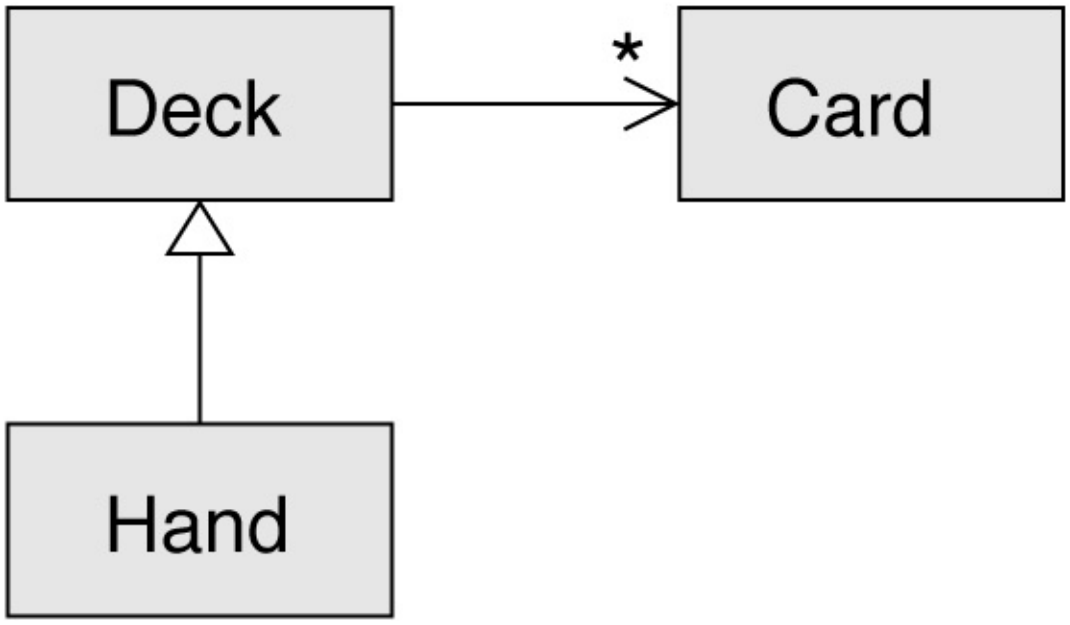


Figure 18-2. Class diagram.

The arrow with a hollow triangle head represents an IS-A relationship; in this case it indicates that Hand inherits from Deck.

The standard arrowhead represents a HAS-A relationship; in this case a Deck has references to Card objects.

The star (`*`) near the arrowhead is a **multiplicity**; it indicates how many Cards a Deck has. A multiplicity can be a simple number like `52`, a range like `5..7` or a star, which indicates that a Deck can have any number of Cards.

There are no dependencies in this diagram. They would normally be shown with a dashed arrow. Or if there are a lot of dependencies, they are sometimes omitted.

A more detailed diagram might show that a Deck actually contains a *list* of Cards, but built-in types like list and dict are usually not included in class diagrams.

Data Encapsulation

The previous chapters demonstrate a development plan we might call "object-oriented design". We identified objects we needed — like `Point`, `Rectangle` and `Time` — and defined classes to represent them. In each case there is an obvious correspondence between the object and some entity in the real world (or at least a mathematical world).

But sometimes it is less obvious what objects you need and how they should interact. In that case you need a different development plan. In the same way that we discovered function interfaces by encapsulation and generalization, we can discover class interfaces by **data encapsulation**.

Markov analysis, from "Markov Analysis", provides a good example. If you download my code from *http://thinkpython2.com/code/markov.py*, you'll see that it uses two global variables — `suffix_map` and `prefix` — that are read and written from several functions.

```
suffix_map = {}
prefix = ()
```

Because these variables are global, we can only run one analysis at a time. If we read two texts, their prefixes and suffixes would be added to the same data structures (which makes for some interesting generated text).

To run multiple analyses, and keep them separate, we can encapsulate the state of each analysis in an object. Here's what that looks like:

```
class Markov:

    def __init__(self):
        self.suffix_map = {}
        self.prefix = ()
```

Next, we transform the functions into methods. For example, here's `process_word`:

```
    def process_word(self, word, order=2):
        if len(self.prefix) < order:
            self.prefix += (word,)
            return

        try:
            self.suffix_map[self.prefix].append(word)
        except KeyError:
            # if there is no entry for this prefix, make one
            self.suffix_map[self.prefix] = [word]

        self.prefix = shift(self.prefix, word)
```

Transforming a program like this — changing the design without changing the behavior — is another example of refactoring (see "Refactoring").

This example suggests a development plan for designing objects and methods:

1. Start by writing functions that read and write global variables (when necessary).

2. Once you get the program working, look for associations between global variables and the functions that use them.

3. Encapsulate related variables as attributes of an object.

4. Transform the associated functions into methods of the new class.

As an exercise, download my Markov code from *http://thinkpython2.com/code/markov.py*, and follow the steps described above to encapsulate the global variables as attributes of a new class called `Markov`.

Solution: *http://thinkpython2.com/code/Markov.py* (note the capital M).

Debugging

Inheritance can make debugging difficult because when you invoke a method on an object, it might be hard to figure out which method will be invoked.

Suppose you are writing a function that works with Hand objects. You would like it to work with all kinds of Hands, like PokerHands, BridgeHands, etc. If you invoke a method like `shuffle`, you might get the one defined in `Deck`, but if any of the subclasses override this method, you'll get that version instead. This behavior is usually a good thing, but it can be confusing.

Any time you are unsure about the flow of execution through your program, the simplest solution is to add print statements at the beginning of the relevant methods. If `Deck.shuffle` prints a message that says something like `Running Deck.shuffle`, then as the program runs it traces the flow of execution.

As an alternative, you could use this function, which takes an object and a method name (as a string) and returns the class that provides the definition of the method:

```
def find_defining_class(obj, meth_name):
    for ty in type(obj).mro():
        if meth_name in ty.__dict__:
            return ty
```

Here's an example:

```
>>> hand = Hand()
>>> find_defining_class(hand, 'shuffle')
<class 'Card.Deck'>
```

So the `shuffle` method for this Hand is the one in `Deck`.

`find_defining_class` uses the `mro` method to get the list of class objects (types) that will be searched for methods. "MRO" stands for "method resolution order", which is the sequence of classes Python searches to "resolve" a method name.

Here's a design suggestion: when you override a method, the interface of the new method should be the same as the old. It should take the same parameters, return the same type, and obey the same preconditions and postconditions. If you follow this rule, you will find that any function designed to work with an instance of a parent class, like a Deck, will also work with instances of child classes like a Hand and PokerHand.

If you violate this rule, which is called the "Liskov substitution principle", your code will collapse like (sorry) a house of cards.

Glossary

encode:

To represent one set of values using another set of values by constructing a mapping between them.

class attribute:

An attribute associated with a class object. Class attributes are defined inside a class definition but outside any method.

instance attribute:

An attribute associated with an instance of a class.

veneer:

A method or function that provides a different interface to another function without doing much computation.

inheritance:

The ability to define a new class that is a modified version of a previously defined class.

parent class:

The class from which a child class inherits.

child class:

A new class created by inheriting from an existing class; also called a “subclass”.

IS-A relationship:

A relationship between a child class and its parent class.

HAS-A relationship:

A relationship between two classes where instances of one class contain references to instances of the other.

dependency:

A relationship between two classes where instances of one class use instances of the other class, but do not store them as attributes.

class diagram:

A diagram that shows the classes in a program and the relationships between them.

multiplicity:

A notation in a class diagram that shows, for a HAS-A relationship, how many references there are to instances of another class.

data encapsulation:

A program development plan that involves a prototype using global variables and a final version that makes the global variables into instance attributes.

Exercises

Exercise 18-1.

For the following program, draw a UML class diagram that shows these classes and the relationships among them.

```
class PingPongParent:
    pass

class Ping(PingPongParent):
    def __init__(self, pong):
        self.pong = pong


class Pong(PingPongParent):
    def __init__(self, pings=None):
        if pings is None:
            self.pings = []
        else:
            self.pings = pings

    def add_ping(self, ping):
        self.pings.append(ping)

pong = Pong()
ping = Ping(pong)
pong.add_ping(ping)
```

Exercise 18-2.

Write a Deck method called `deal_hands` that takes two parameters: the number of hands and the number of cards per hand. It should create the appropriate number of Hand objects, deal the appropriate number of cards per hand, and return a list of Hands.

Exercise 18-3.

The following are the possible hands in poker, in increasing order of value and decreasing order of probability:

pair:

Two cards with the same rank.

two pair:

Two pairs of cards with the same rank.

three of a kind:

Three cards with the same rank.

straight:

Five cards with ranks in sequence (aces can be high or low, so `Ace-2-3-4-5` is a straight and so is `10-Jack-Queen-King-Ace`, but `Queen-King-Ace-2-3` is not.)

flush:

Five cards with the same suit.

full house:

Three cards with one rank, two cards with another.

four of a kind:

Four cards with the same rank.

straight flush:

Five cards in sequence (as defined above) and with the same suit.

The goal of these exercises is to estimate the probability of drawing these various hands.

1. Download the following files from *http://thinkpython2.com/code*:

   `Card.py`:

   A complete version of the `Card`, `Deck` and `Hand` classes in this chapter.

   `PokerHand.py`:

   An incomplete implementation of a class that represents a poker hand, and some code that tests it.

2. If you run `PokerHand.py`, it deals seven 7-card poker hands and checks to see if any of them contains a flush. Read this code carefully before you go on.

3. Add methods to `PokerHand.py` named `has_pair`, `has_twopair`, etc. that return True or False according to whether or not the hand meets the relevant criteria. Your code should work correctly for "hands" that contain any number of cards (although 5 and 7 are the most common sizes).

4. Write a method named `classify` that figures out the highest-value classification for a hand and sets the `label` attribute accordingly. For example, a 7-card hand might contain a flush and a pair; it should be labeled "flush".

5. When you are convinced that your classification methods are working, the next step is to estimate the probabilities of the various hands. Write a function in `PokerHand.py` that shuffles a deck of cards, divides it into hands, classifies the hands, and counts the number of times various classifications appear.

6. Print a table of the classifications and their probabilities. Run your program with larger and larger numbers of hands until the output values converge to a reasonable degree of accuracy. Compare your results to the values at *http://en.wikipedia.org/wiki/Hand_rankings*.

Solution: *http://thinkpython2.com/code/PokerHandSoln.py*.

Chapter 19. The Goodies

One of my goals for this book has been to teach you as little Python as possible. When there were two ways to do something, I picked one and avoided mentioning the other. Or sometimes I put the second one into an exercise.

Now I want to go back for some of the good bits that got left behind. Python provides a number of features that are not really necessary — you can write good code without them — but with them you can sometimes write code that’s more concise, readable or efficient, and sometimes all three.

Conditional Expressions

We saw conditional statements in "Conditional Execution". Conditional statements are often used to choose one of two values; for example:

```
if x > 0:
    y = math.log(x)
else:
    y = float('nan')
```

This statement checks whether `x` is positive. If so, it computes `math.log`. If not, `math.log` would raise a `ValueError`. To avoid stopping the program, we generate a "NaN", which is a special floating-point value that represents "Not a Number".

We can write this statement more concisely using a **conditional expression**:

```
y = math.log(x) if x > 0 else float('nan')
```

You can almost read this line like English: "`y` gets log-`x` if `x` is greater than 0; otherwise it gets NaN".

Recursive functions can sometimes be rewritten using conditional expressions. For example, here is a recursive version of `factorial`:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

We can rewrite it like this:

```
def factorial(n):
    return 1 if n == 0 else n * factorial(n-1)
```

Another use of conditional expressions is handling optional arguments. For example, here is the init method from `GoodKangaroo` (see Exercise 17-2):

```
    def __init__(self, name, contents=None):
        self.name = name
        if contents == None:
            contents = []
        self.pouch_contents = contents
```

We can rewrite this one like this:

```
    def __init__(self, name, contents=None):
        self.name = name
        self.pouch_contents = [] if contents == None else contents
```

In general, you can replace a conditional statement with a conditional expression if both branches contain simple expressions that are either returned or assigned to the same

variable.

List Comprehensions

In "Map, Filter and Reduce" we saw the map and filter patterns. For example, this function takes a list of strings, maps the string method `capitalize` to the elements, and returns a new list of strings:

```
def capitalize_all(t):
    res = []
    for s in t:
        res.append(s.capitalize())
    return res
```

We can write this more concisely using a **list comprehension**:

```
def capitalize_all(t):
    return [s.capitalize() for s in t]
```

The bracket operators indicate that we are constructing a new list. The expression inside the brackets specifies the elements of the list, and the `for` clause indicates what sequence we are traversing.

The syntax of a list comprehension is a little awkward because the loop variable, `s` in this example, appears in the expression before we get to the definition.

List comprehensions can also be used for filtering. For example, this function selects only the elements of `t` that are uppercase, and returns a new list:

```
def only_upper(t):
    res = []
    for s in t:
        if s.isupper():
            res.append(s)
    return res
```

We can rewrite it using a list comprehension:

```
def only_upper(t):
    return [s for s in t if s.isupper()]
```

List comprehensions are concise and easy to read, at least for simple expressions. And they are usually faster than the equivalent for loops, sometimes much faster. So if you are mad at me for not mentioning them earlier, I understand.

But, in my defense, list comprehensions are harder to debug because you can't put a print statement inside the loop. I suggest that you use them only if the computation is simple enough that you are likely to get it right the first time. And for beginners that means never.

Generator Expressions

Generator expressions are similar to list comprehensions, but with parentheses instead of square brackets:

```
>>> g = (x**2 for x in range(5))
>>> g
<generator object <genexpr> at 0x7f4c45a786c0>
```

The result is a generator object that knows how to iterate through a sequence of values. But unlike a list comprehension, it does not compute the values all at once; it waits to be asked. The built-in function `next` gets the next value from the generator:

```
>>> next(g)
0
>>> next(g)
1
```

When you get to the end of the sequence, `next` raises a `StopIteration` exception. You can also use a `for` loop to iterate through the values:

```
>>> for val in g:
...     print(val)
4
9
16
```

The generator object keeps track of where it is in the sequence, so the `for` loop picks up where `next` left off. Once the generator is exhausted, it continues to raise `StopException`:

```
>>> next(g)
StopIteration
```

Generator expressions are often used with functions like `sum`, `max`, and `min`:

```
>>> sum(x**2 for x in range(5))
30
```

any and all

Python provides a built-in function, `any`, that takes a sequence of boolean values and returns `True` if any of the values are `True`. It works on lists:

```
>>> any([False, False, True])
True
```

But it is often used with generator expressions:

```
>>> any(letter == 't' for letter in 'monty')
True
```

That example isn't very useful because it does the same thing as the `in` operator. But we could use `any` to rewrite some of the search functions we wrote in "Search". For example, we could write `avoids` like this:

```
def avoids(word, forbidden):
    return not any(letter in forbidden for letter in word)
```

The function almost reads like English: "`word` avoids `forbidden` if there are not any forbidden letters in `word`."

Using `any` with a generator expression is efficient because it stops immediately if it finds a `True` value, so it doesn't have to evaluate the whole sequence.

Python provides another built-in function, `all`, that returns `True` if every element of the sequence is `True`. As an exercise, use `all` to rewrite `uses_all` from "Search".

Sets

In "Dictionary Subtraction" I use dictionaries to find the words that appear in a document but not in a word list. The function I wrote takes `d1`, which contains the words from the document as keys, and `d2`, which contains the list of words. It returns a dictionary that contains the keys from `d1` that are not in `d2`:

```
def subtract(d1, d2):
    res = dict()
    for key in d1:
        if key not in d2:
            res[key] = None
    return res
```

In all of these dictionaries, the values are `None` because we never use them. As a result, we waste some storage space.

Python provides another built-in type, called a `set`, that behaves like a collection of dictionary keys with no values. Adding elements to a set is fast; so is checking membership. And sets provide methods and operators to compute common set operations.

For example, set subtraction is available as a method called `difference` or as an operator, `-`. So we can rewrite `subtract` like this:

```
def subtract(d1, d2):
    return set(d1) - set(d2)
```

The result is a set instead of a dictionary, but for operations like iteration, the behavior is the same.

Some of the exercises in this book can be done concisely and efficiently with sets. For example, here is a solution to `has_duplicates`, from Exercise 10-7, that uses a dictionary:

```
def has_duplicates(t):
    d = {}
    for x in t:
        if x in d:
            return True
        d[x] = True
    return False
```

When an element appears for the first time, it is added to the dictionary. If the same element appears again, the function returns `True`.

Using sets, we can write the same function like this:

```
def has_duplicates(t):
    return len(set(t)) < len(t)
```

An element can only appear in a set once, so if an element in `t` appears more than once, the set will be smaller than `t`. If there are no duplicates, the set will be the same size as `t`.

We can also use sets to do some of the exercises in Chapter 9. For example, here's a version of `uses_only` with a loop:

```
def uses_only(word, available):
    for letter in word:
        if letter not in available:
            return False
    return True
```

`uses_only` checks whether all letters in `word` are in `available`. We can rewrite it like this:

```
def uses_only(word, available):
    return set(word) <= set(available)
```

The `<=` operator checks whether one set is a subset or another, including the possibility that they are equal, which is true if all the letters in `word` appear in `available`.

As an exercise, rewrite `avoids` using sets.

Counters

A Counter is like a set, except that if an element appears more than once, the Counter keeps track of how many times it appears. If you are familiar with the mathematical idea of a **multiset**, a Counter is a natural way to represent a multiset.

Counter is defined in a standard module called `collections`, so you have to import it. You can initialize a Counter with a string, list, or anything else that supports iteration:

```
>>> from collections import Counter
>>> count = Counter('parrot')
>>> count
Counter({'r': 2, 't': 1, 'o': 1, 'p': 1, 'a': 1})
```

Counters behave like dictionaries in many ways; they map from each key to the number of times it appears. As in dictionaries, the keys have to be hashable.

Unlike dictionaries, Counters don't raise an exception if you access an element that doesn't appear. Instead, they return 0:

```
>>> count['d']
0
```

We can use Counters to rewrite `is_anagram` from Exercise 10-6:

```
def is_anagram(word1, word2):
    return Counter(word1) == Counter(word2)
```

If two words are anagrams, they contain the same letters with the same counts, so their Counters are equivalent.

Counters provide methods and operators to perform set-like operations, including addition, subtraction, union and intersection. And they provide an often-useful method, `most_common`, which returns a list of value-frequency pairs, sorted from most common to least:

```
>>> count = Counter('parrot')
>>> for val, freq in count.most_common(3):
...     print(val, freq)
r 2
p 1
a 1
```

defaultdict

The `collections` module also provides `defaultdict`, which is like a dictionary except that if you access a key that doesn't exist, it can generate a new value on the fly.

When you create a `defaultdict`, you provide a function that's used to create new values. A function used to create objects is sometimes called a **factory**. The built-in functions that create lists, sets, and other types can be used as factories:

```
>>> from collections import defaultdict
>>> d = defaultdict(list)
```

Notice that the argument is `list`, which is a class object, not `list()`, which is a new list. The function you provide doesn't get called unless you access a key that doesn't exist:

```
>>> t = d['new key']
>>> t
[]
```

The new list, which we're calling `t`, is also added to the dictionary. So if we modify `t`, the change appears in `d`:

```
>>> t.append('new value')
>>> d
defaultdict(<class 'list'>, {'new key': ['new value']})
```

If you are making a dictionary of lists, you can often write simpler code using `defaultdict`. In my solution to Exercise 12-2, which you can get from *http://thinkpython2.com/code/anagram_sets.py*, I make a dictionary that maps from a sorted string of letters to the list of words that can be spelled with those letters. For example, `'opst'` maps to the list `['opts', 'post', 'pots', 'spot', 'stop', 'tops']`.

Here's the original code:

```
def all_anagrams(filename):
    d = {}
    for line in open(filename):
        word = line.strip().lower()
        t = signature(word)
        if t not in d:
            d[t] = [word]
        else:
            d[t].append(word)
    return d
```

This can be simplified using `setdefault`, which you might have used in Exercise 11-2:

```
def all_anagrams(filename):
    d = {}
    for line in open(filename):
        word = line.strip().lower()
        t = signature(word)
        d.setdefault(t, []).append(word)
```

```
    return d
```

This solution has the drawback that it makes a new list every time, regardless of whether it is needed. For lists, that's no big deal, but if the factory function is complicated, it might be.

We can avoid this problem and simplify the code using a `defaultdict`:

```
def all_anagrams(filename):
    d = defaultdict(list)
    for line in open(filename):
        word = line.strip().lower()
        t = signature(word)
        d[t].append(word)
    return d
```

My solution to Exercise 18-3, which you can download from *http://thinkpython2.com/code/PokerHandSoln.py*, uses `setdefault` in the function `has_straightflush`. This solution has the drawback of creating a `Hand` object every time through the loop, whether it is needed or not. As an exercise, rewrite it using a `defaultdict`.

Named Tuples

Many simple objects are basically collections of related values. For example, the Point object defined in Chapter 15 contains two numbers, `x` and `y`. When you define a class like this, you usually start with an init method and a str method:

```
class Point:

    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def __str__(self):
        return '(%g, %g)' % (self.x, self.y)
```

This is a lot of code to convey a small amount of information. Python provides a more concise way to say the same thing:

```
from collections import namedtuple
Point = namedtuple('Point', ['x', 'y'])
```

The first argument is the name of the class you want to create. The second is a list of the attributes Point objects should have, as strings. The return value from `namedtuple` is a class object:

```
>>> Point
<class '__main__.Point'>
```

`Point` automatically provides methods like `__init__` and `__str__` so you don't have to write them.

To create a Point object, you use the Point class as a function:

```
>>> p = Point(1, 2)
>>> p
Point(x=1, y=2)
```

The init method assigns the arguments to attributes using the names you provided. The str method prints a representation of the Point object and its attributes.

You can access the elements of the named tuple by name:

```
>>> p.x, p.y
(1, 2)
```

But you can also treat a named tuple as a tuple:

```
>>> p[0], p[1]
(1, 2)

>>> x, y = p
>>> x, y
(1, 2)
```

Named tuples provide a quick way to define simple classes. The drawback is that simple classes don’t always stay simple. You might decide later that you want to add methods to a named tuple. In that case, you could define a new class that inherits from the named tuple:

```
class Pointier(Point):
    # add more methods here
```

Or you could switch to a conventional class definition.

Gathering Keyword Args

In “Variable-Length Argument Tuples”, we saw how to write a function that gathers its arguments into a tuple:

```
def printall(*args):
    print(args)
```

You can call this function with any number of positional arguments (that is, arguments that don’t have keywords):

```
>>> printall(1, 2.0, '3')
(1, 2.0, '3')
```

But the `*` operator doesn’t gather keyword arguments:

```
>>> printall(1, 2.0, third='3')
TypeError: printall() got an unexpected keyword argument 'third'
```

To gather keyword arguments, you can use the `**` operator:

```
def printall(*args, **kwargs):
    print(args, kwargs)
```

You can call the keyword gathering parameter anything you want, but `kwargs` is a common choice. The result is a dictionary that maps keywords to values:

```
>>> printall(1, 2.0, third='3')
(1, 2.0) {'third': '3'}
```

If you have a dictionary of keywords and values, you can use the scatter operator, `**`, to call a function:

```
>>> d = dict(x=1, y=2)
>>> Point(**d)
Point(x=1, y=2)
```

Without the scatter operator, the function would treat `d` as a single positional argument, so it would assign `d` to `x` and complain because there’s nothing to assign to `y`:

```
>>> d = dict(x=1, y=2)
>>> Point(d)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: __new__() missing 1 required positional argument: 'y'
```

When you are working with functions that have a large number of parameters, it is often useful to create and pass around dictionaries that specify frequently used options.

Glossary

conditional expression:

An expression that has one of two values, depending on a condition.

list comprehension:

An expression with a `for` loop in square brackets that yields a new list.

generator expression:

An expression with a `for` loop in parentheses that yields a generator object.

multiset:

A mathematical entity that represents a mapping between the elements of a set and the number of times they appear.

factory:

A function, usually passed as a parameter, used to create objects.

Exercises

Exercise 19-1.

The following is a function that computes the binomial coefficient recursively:

```
def binomial_coeff(n, k):
    """Compute the binomial coefficient "n choose k".

    n: number of trials
    k: number of successes

    returns: int
    """
    if k == 0:
        return 1
    if n == 0:
        return 0

    res = binomial_coeff(n-1, k) + binomial_coeff(n-1, k-1)
    return res
```

Rewrite the body of the function using nested conditional expressions.

One note: this function is not very efficient because it ends up computing the same values over and over. You could make it more efficient by memoizing (see “Memos”). But you will find that it’s harder to memoize if you write it using conditional expressions.

Chapter 20. Debugging

When you are debugging, you should distinguish among different kinds of errors in order to track them down more quickly:

- Syntax errors are discovered by the interpreter when it is translating the source code into byte code. They indicate that there is something wrong with the structure of the program. Example: Omitting the colon at the end of a `def` statement generates the somewhat redundant message `SyntaxError: invalid syntax`.

- Runtime errors are produced by the interpreter if something goes wrong while the program is running. Most runtime error messages include information about where the error occurred and what functions were executing. Example: An infinite recursion eventually causes the runtime error `maximum recursion depth exceeded`.

- Semantic errors are problems with a program that runs without producing error messages but doesn't do the right thing. Example: An expression may not be evaluated in the order you expect, yielding an incorrect result.

The first step in debugging is to figure out which kind of error you are dealing with. Although the following sections are organized by error type, some techniques are applicable in more than one situation.

Syntax Errors

Syntax errors are usually easy to fix once you figure out what they are. Unfortunately, the error messages are often not helpful. The most common messages are `SyntaxError: invalid syntax` and `SyntaxError: invalid token`, neither of which is very informative.

On the other hand, the message does tell you where in the program the problem occurred. Actually, it tells you where Python noticed a problem, which is not necessarily where the error is. Sometimes the error is prior to the location of the error message, often on the preceding line.

If you are building the program incrementally, you should have a good idea about where the error is. It will be in the last line you added.

If you are copying code from a book, start by comparing your code to the book's code very carefully. Check every character. At the same time, remember that the book might be wrong, so if you see something that looks like a syntax error, it might be.

Here are some ways to avoid the most common syntax errors:

1. Make sure you are not using a Python keyword for a variable name.

2. Check that you have a colon at the end of the header of every compound statement, including `for`, `while`, `if`, and `def` statements.

3. Make sure that any strings in the code have matching quotation marks. Make sure that all quotation marks are straight quotes, not curly quotes.

4. If you have multiline strings with triple quotes (single or double), make sure you have terminated the string properly. An unterminated string may cause an `invalid token` error at the end of your program, or it may treat the following part of the program as a string until it comes to the next string. In the second case, it might not produce an error message at all!

5. An unclosed opening operator — `(`, `{`, or `[` — makes Python continue with the next line as part of the current statement. Generally, an error occurs almost immediately in the next line.

6. Check for the classic `=` instead of `==` inside a conditional.

7. Check the indentation to make sure it lines up the way it is supposed to. Python can handle space and tabs, but if you mix them it can cause problems. The best way to avoid this problem is to use a text editor that knows about Python and generates consistent indentation.

8. If you have non-ASCII characters in the code (including strings and comments), that might cause a problem, although Python 3 usually handles non-ASCII characters. Be careful if you paste in text from a web page or other source.

If nothing works, move on to the next section…

I keep making changes and it makes no difference.

If the interpreter says there is an error and you don't see it, that might be because you and the interpreter are not looking at the same code. Check your programming environment to make sure that the program you are editing is the one Python is trying to run.

If you are not sure, try putting an obvious and deliberate syntax error at the beginning of the program. Now run it again. If the interpreter doesn't find the new error, you are not running the new code.

There are a few likely culprits:

- You edited the file and forgot to save the changes before running it again. Some programming environments do this for you, but some don't.
- You changed the name of the file, but you are still running the old name.
- Something in your development environment is configured incorrectly.
- If you are writing a module and using `import`, make sure you don't give your module the same name as one of the standard Python modules.
- If you are using `import` to read a module, remember that you have to restart the interpreter or use `reload` to read a modified file. If you import the module again, it doesn't do anything.

If you get stuck and you can't figure out what is going on, one approach is to start again with a new program like "Hello, World!", and make sure you can get a known program to run. Then gradually add the pieces of the original program to the new one.

Runtime Errors

Once your program is syntactically correct, Python can read it and at least start running it. What could possibly go wrong?

My program does absolutely nothing.

This problem is most common when your file consists of functions and classes but does not actually invoke a function to start execution. This may be intentional if you only plan to import this module to supply classes and functions.

If it is not intentional, make sure there is a function call in the program, and make sure the flow of execution reaches it (see “Flow of execution” below).

My program hangs.

If a program stops and seems to be doing nothing, it is "hanging". Often that means that it is caught in an infinite loop or infinite recursion.

- If there is a particular loop that you suspect is the problem, add a `print` statement immediately before the loop that says "entering the loop" and another immediately after that says "exiting the loop".
  Run the program. If you get the first message and not the second, you've got an infinite loop. Go to the "Infinite loop" section below.

- Most of the time, an infinite recursion will cause the program to run for a while and then produce a "RuntimeError: Maximum recursion depth exceeded" error. If that happens, go to the "Infinite recursion" section below.
  If you are not getting this error but you suspect there is a problem with a recursive method or function, you can still use the techniques in the "Infinite recursion" section.

- If neither of those steps works, start testing other loops and other recursive functions and methods.

- If that doesn't work, then it is possible that you don't understand the flow of execution in your program. Go to the "Flow of execution" section below.

Infinite loop

If you think you have an infinite loop and you think you know what loop is causing the problem, add a `print` statement at the end of the loop that prints the values of the variables in the condition and the value of the condition.

For example:

```
while x > 0 and y < 0 :
    # do something to x
    # do something to y

    print('x: ', x)
    print('y: ', y)
    print("condition: ", (x > 0 and y < 0))
```

Now when you run the program, you will see three lines of output for each time through the loop. The last time through the loop, the condition should be `False`. If the loop keeps going, you will be able to see the values of `x` and `y`, and you might figure out why they are not being updated correctly.

Infinite recursion

Most of the time, infinite recursion causes the program to run for a while and then produce a `Maximum recursion depth exceeded` error.

If you suspect that a function is causing an infinite recursion, make sure that there is a base case. There should be some condition that causes the function to return without

making a recursive invocation. If not, you need to rethink the algorithm and identify a base case.

If there is a base case but the program doesn't seem to be reaching it, add a `print` statement at the beginning of the function that prints the parameters. Now when you run the program, you will see a few lines of output every time the function is invoked, and you will see the parameter values. If the parameters are not moving toward the base case, you will get some ideas about why not.

Flow of execution

If you are not sure how the flow of execution is moving through your program, add `print` statements to the beginning of each function with a message like "entering function `foo`", where `foo` is the name of the function.

Now when you run the program, it will print a trace of each function as it is invoked.

When I run the program I get an exception.

If something goes wrong during runtime, Python prints a message that includes the name of the exception, the line of the program where the problem occurred, and a traceback.

The traceback identifies the function that is currently running, and then the function that called it, and then the function that called *that*, and so on. In other words, it traces the sequence of function calls that got you to where you are, including the line number in your file where each call occurred.

The first step is to examine the place in the program where the error occurred and see if you can figure out what happened. These are some of the most common runtime errors:

NameError:

> You are trying to use a variable that doesn't exist in the current environment. Check if the name is spelled right, or at least consistently. And remember that local variables are local; you cannot refer to them from outside the function where they are defined.

TypeError:

> There are several possible causes:
>
> - You are trying to use a value improperly. Example: indexing a string, list, or tuple with something other than an integer.
>
> - There is a mismatch between the items in a format string and the items passed for conversion. This can happen if either the number of items does not match or an invalid conversion is called for.
>
> - You are passing the wrong number of arguments to a function. For methods, look at the method definition and check that the first parameter is `self`. Then look at the method invocation; make sure you are invoking the method on an object with the right type and providing the other arguments correctly.

KeyError:

> You are trying to access an element of a dictionary using a key that the dictionary does not contain. If the keys are strings, remember that capitalization matters.

AttributeError:

> You are trying to access an attribute or method that does not exist. Check the spelling! You can use the built-in function `vars` to list the attributes that do exist.
>
> If an AttributeError indicates that an object has `NoneType`, that means that it is `None`. So the problem is not the attribute name, but the object.
>
> The reason the object is none might be that you forgot to return a value from a function; if you get to the end of a function without hitting a `return` statement, it returns `None`. Another common cause is using the result from a list method, like `sort`, that returns `None`.

IndexError:

> The index you are using to access a list, string, or tuple is greater than its length minus one. Immediately before the site of the error, add a `print` statement to display the value of the index and the length of the array. Is the array the right size? Is the index the right value?

The Python debugger (`pdb`) is useful for tracking down exceptions because it allows you to examine the state of the program immediately before the error. You can read about `pdb` at *https://docs.python.org/3/library/pdb.html*.

I added so many print statements I get inundated with output.

One of the problems with using `print` statements for debugging is that you can end up buried in output. There are two ways to proceed: simplify the output or simplify the program.

To simplify the output, you can remove or comment out `print` statements that aren't helping, or combine them, or format the output so it is easier to understand.

To simplify the program, there are several things you can do. First, scale down the problem the program is working on. For example, if you are searching a list, search a *small* list. If the program takes input from the user, give it the simplest input that causes the problem.

Second, clean up the program. Remove dead code and reorganize the program to make it as easy to read as possible. For example, if you suspect that the problem is in a deeply nested part of the program, try rewriting that part with simpler structure. If you suspect a large function, try splitting it into smaller functions and testing them separately.

Often the process of finding the minimal test case leads you to the bug. If you find that a program works in one situation but not in another, that gives you a clue about what is going on.

Similarly, rewriting a piece of code can help you find subtle bugs. If you make a change that you think shouldn't affect the program, and it does, that can tip you off.

Semantic Errors

In some ways, semantic errors are the hardest to debug, because the interpreter provides no information about what is wrong. Only you know what the program is supposed to do.

The first step is to make a connection between the program text and the behavior you are seeing. You need a hypothesis about what the program is actually doing. One of the things that makes that hard is that computers run so fast.

You will often wish that you could slow the program down to human speed, and with some debuggers you can. But the time it takes to insert a few well-placed `print` statements is often short compared to setting up the debugger, inserting and removing breakpoints, and “stepping” the program to where the error is occurring.

My program doesn’t work.

You should ask yourself these questions:

- Is there something the program was supposed to do but which doesn’t seem to be happening? Find the section of the code that performs that function and make sure it is executing when you think it should.
- Is something happening that shouldn’t? Find code in your program that performs that function and see if it is executing when it shouldn’t.
- Is a section of code producing an effect that is not what you expected? Make sure that you understand the code in question, especially if it involves functions or methods in other Python modules. Read the documentation for the functions you call. Try them out by writing simple test cases and checking the results.

In order to program, you need a mental model of how programs work. If you write a program that doesn’t do what you expect, often the problem is not in the program; it’s in your mental model.

The best way to correct your mental model is to break the program into its components (usually the functions and methods) and test each component independently. Once you find the discrepancy between your model and reality, you can solve the problem.

Of course, you should be building and testing components as you develop the program. If you encounter a problem, there should be only a small amount of new code that is not known to be correct.

I’ve got a big hairy expression and it doesn’t do what I expect.

Writing complex expressions is fine as long as they are readable, but they can be hard to debug. It is often a good idea to break a complex expression into a series of assignments to temporary variables.

For example:

```
self.hands[i].addCard(self.hands[self.findNeighbor(i)].popCard())
```

This can be rewritten as:

```
neighbor = self.findNeighbor(i)
pickedCard = self.hands[neighbor].popCard()
self.hands[i].addCard(pickedCard)
```

The explicit version is easier to read because the variable names provide additional documentation, and it is easier to debug because you can check the types of the intermediate variables and display their values.

Another problem that can occur with big expressions is that the order of evaluation may not be what you expect. For example, if you are translating the expression $\frac{x}{2\pi}$ into Python, you might write:

```
y = x / 2 * math.pi
```

That is not correct because multiplication and division have the same precedence and are evaluated from left to right. So this expression computes $x\pi / 2$.

A good way to debug expressions is to add parentheses to make the order of evaluation explicit:

```
y = x / (2 * math.pi)
```

Whenever you are not sure of the order of evaluation, use parentheses. Not only will the program be correct (in the sense of doing what you intended), it will also be more readable for other people who haven’t memorized the order of operations.

I’ve got a function that doesn’t return what I expect.

If you have a `return` statement with a complex expression, you don’t have a chance to print the result before returning. Again, you can use a temporary variable. For example, instead of:

```
return self.hands[i].removeMatches()
```

you could write:

```
count = self.hands[i].removeMatches()
return count
```

Now you have the opportunity to display the value of `count` before returning.

I’m really, really stuck and I need help.

First, try getting away from the computer for a few minutes. Computers emit waves that affect the brain, causing these symptoms:

- Frustration and rage.
- Superstitious beliefs (“the computer hates me”) and magical thinking (“the program only works when I wear my hat backward”).
- Random walk programming (the attempt to program by writing every possible program and choosing the one that does the right thing).

If you find yourself suffering from any of these symptoms, get up and go for a walk. When you are calm, think about the program. What is it doing? What are some possible causes of that behavior? When was the last time you had a working program, and what did you do next?

Sometimes it just takes time to find a bug. I often find bugs when I am away from the computer and let my mind wander. Some of the best places to find bugs are on trains, in the shower, and in bed just before you fall asleep.

No, I really need help.

It happens. Even the best programmers occasionally get stuck. Sometimes you work on a program so long that you can't see the error. You need a fresh pair of eyes.

Before you bring someone else in, make sure you are prepared. Your program should be as simple as possible, and you should be working on the smallest input that causes the error. You should have `print` statements in the appropriate places (and the output they produce should be comprehensible). You should understand the problem well enough to describe it concisely.

When you bring someone in to help, be sure to give them the information they need:

- If there is an error message, what is it and what part of the program does it indicate?
- What was the last thing you did before this error occurred? What were the last lines of code that you wrote, or what is the new test case that fails?
- What have you tried so far, and what have you learned?

When you find the bug, take a second to think about what you could have done to find it faster. Next time you see something similar, you will be able to find the bug more quickly.

Remember, the goal is not just to make the program work. The goal is to learn how to make the program work.

Chapter 21. Analysis of Algorithms

This appendix is an edited excerpt from *Think Complexity*, by Allen B. Downey, also published by O'Reilly Media (2012). When you are done with this book, you might want to move on to that one.

Analysis of algorithms is a branch of computer science that studies the performance of algorithms, especially their runtime and space requirements. See *http://en.wikipedia.org/wiki/Analysis_of_algorithms*.

The practical goal of algorithm analysis is to predict the performance of different algorithms in order to guide design decisions.

During the 2008 United States presidential campaign, candidate Barack Obama was asked to perform an impromptu analysis when he visited Google. Chief executive Eric Schmidt jokingly asked him for "the most efficient way to sort a million 32-bit integers." Obama had apparently been tipped off, because he quickly replied, "I think the bubble sort would be the wrong way to go." See *http://bit.ly/1MpIwTf*.

This is true: bubble sort is conceptually simple but slow for large datasets. The answer Schmidt was probably looking for is "radix sort" (*http://en.wikipedia.org/wiki/Radix_sort*).[1]

The goal of algorithm analysis is to make meaningful comparisons between algorithms, but there are some problems:

- The relative performance of the algorithms might depend on characteristics of the hardware, so one algorithm might be faster on Machine A, another on Machine B. The general solution to this problem is to specify a **machine model** and analyze the number of steps, or operations, an algorithm requires under a given model.

- Relative performance might depend on the details of the dataset. For example, some sorting algorithms run faster if the data are already partially sorted; other algorithms run slower in this case. A common way to avoid this problem is to analyze the **worst-case** scenario. It is sometimes useful to analyze average-case performance, but that's usually harder, and it might not be obvious what set of cases to average over.

- Relative performance also depends on the size of the problem. A sorting algorithm that is fast for small lists might be slow for long lists. The usual solution to this problem is to express runtime (or number of operations) as a function of problem size, and group functions into categories depending on how quickly they grow as problem size increases.

The good thing about this kind of comparison is that it lends itself to simple classification of algorithms. For example, if I know that the runtime of Algorithm A tends to be

proportional to the size of the input, n, and Algorithm B tends to be proportional to n^2, then I expect A to be faster than B, at least for large values of n.

This kind of analysis comes with some caveats, but we'll get to that later.

Order of Growth

Suppose you have analyzed two algorithms and expressed their runtimes in terms of the size of the input: Algorithm A takes *100n+1* steps to solve a problem with size *n*; Algorithm B takes $n^2 + n + 1$ steps.

The following table shows the runtime of these algorithms for different problem sizes:

Input size	Runtime of Algorithm A	Runtime of Algorithm B
10	1 001	111
100	10 001	10 101
1 000	100 001	1 001 001
10 000	1 000 001	$> 10^{10}$

At *n=10*, Algorithm A looks pretty bad; it takes almost 10 times longer than Algorithm B. But for *n=100* they are about the same, and for larger values A is much better.

The fundamental reason is that for large values of *n*, any function that contains an n^2 term will grow faster than a function whose leading term is *n*. The **leading term** is the term with the highest exponent.

For Algorithm A, the leading term has a large coefficient, 100, which is why B does better than A for small *n*. But regardless of the coefficients, there will always be some value of *n* where $an^2 > bn$, for any values of *a* and *b*.

The same argument applies to the non-leading terms. Even if the runtime of Algorithm A were *n+1000000*, it would still be better than Algorithm B for sufficiently large *n*.

In general, we expect an algorithm with a smaller leading term to be a better algorithm for large problems, but for smaller problems, there may be a **crossover point** where another algorithm is better. The location of the crossover point depends on the details of the algorithms, the inputs, and the hardware, so it is usually ignored for purposes of algorithmic analysis. But that doesn't mean you can forget about it.

If two algorithms have the same leading order term, it is hard to say which is better; again, the answer depends on the details. So for algorithmic analysis, functions with the same leading term are considered equivalent, even if they have different coefficients.

An **order of growth** is a set of functions whose growth behavior is considered equivalent. For example, *2n*, *100n* and *n+1* belong to the same order of growth, which is written *O(n)* in **Big-Oh notation** and often called **linear** because every function in the set grows linearly with *n*.

All functions with the leading term n^2 belong to $O(n^2)$; they are called **quadratic**.

The following table shows some of the orders of growth that appear most commonly in

algorithmic analysis, in increasing order of badness.

Order of growth	Name
O(1)	constant
$O(\log_b n)$	logarithmic (for any *b*)
O(n)	linear
$O(n \log_b n)$	linearithmic
$O(n^2)$	quadratic
$O(n^3)$	cubic
$O(c^n)$	exponential (for any *c*)

For the logarithmic terms, the base of the logarithm doesn't matter; changing bases is the equivalent of multiplying by a constant, which doesn't change the order of growth. Similarly, all exponential functions belong to the same order of growth regardless of the base of the exponent. Exponential functions grow very quickly, so exponential algorithms are only useful for small problems.

Exercise 21-1.

Read the Wikipedia page on Big-Oh notation at *http://en.wikipedia.org/wiki/Big_O_notation* and answer the following questions:

1. What is the order of growth of $n^3 + n^2$? What about $1000000n^3 + n^2$? What about $n^3 + 1000000n^2$?

2. What is the order of growth of $(n^2 + n) \cdot (n + 1)$? Before you start multiplying, remember that you only need the leading term.

3. If *f* is in *O(g)*, for some unspecified function *g*, what can we say about *af+b*?

4. If f_1 and f_2 are in *O(g)*, what can we say about $f_1 + f_2$?

5. If f_1 is in *O(g)* and f_2 is in *O(h)*, what can we say about $f_1 + f_2$?

6. If f_1 is in *O(g)* and f_2 is *O(h)*, what can we say about $f_1 \cdot f_2$?

Programmers who care about performance often find this kind of analysis hard to swallow. They have a point: sometimes the coefficients and the non-leading terms make a real difference. Sometimes the details of the hardware, the programming language, and the characteristics of the input make a big difference. And for small problems, asymptotic behavior is irrelevant.

But if you keep those caveats in mind, algorithmic analysis is a useful tool. At least for large problems, the “better” algorithms is usually better, and sometimes it is *much* better. The difference between two algorithms with the same order of growth is usually a constant factor, but the difference between a good algorithm and a bad algorithm is unbounded!

Analysis of Basic Python Operations

In Python, most arithmetic operations are constant time; multiplication usually takes longer than addition and subtraction, and division takes even longer, but these runtimes don't depend on the magnitude of the operands. Very large integers are an exception; in that case the runtime increases with the number of digits.

Indexing operations — reading or writing elements in a sequence or dictionary — are also constant time, regardless of the size of the data structure.

A `for` loop that traverses a sequence or dictionary is usually linear, as long as all of the operations in the body of the loop are constant time. For example, adding up the elements of a list is linear:

```
total = 0
for x in t:
    total += x
```

The built-in function `sum` is also linear because it does the same thing, but it tends to be faster because it is a more efficient implementation; in the language of algorithmic analysis, it has a smaller leading coefficient.

As a rule of thumb, if the body of a loop is in $O(n^a)$ then the whole loop is in $O(n^{a+1})$. The exception is if you can show that the loop exits after a constant number of iterations. If a loop runs *k* times regardless of *n*, then the loop is in $O(n^a)$, even for large *k*.

Multiplying by *k* doesn't change the order of growth, but neither does dividing. So if the body of a loop is in $O(n^a)$ and it runs *n/k* times, the loop is in $O(n^{a+1})$, even for large *k*.

Most string and tuple operations are linear, except indexing and `len`, which are constant time. The built-in functions `min` and `max` are linear. The runtime of a slice operation is proportional to the length of the output, but independent of the size of the input.

String concatenation is linear; the runtime depends on the sum of the lengths of the operands.

All string methods are linear, but if the lengths of the strings are bounded by a constant — for example, operations on single characters — they are considered constant time. The string method `join` is linear; the runtime depends on the total length of the strings.

Most list methods are linear, but there are some exceptions:

- Adding an element to the end of a list is constant time on average; when it runs out of room it occasionally gets copied to a bigger location, but the total time for *n* operations is *O(n)*, so the average time for each operation is *O(1)*.

- Removing an element from the end of a list is constant time.

- Sorting is $O(n \log n)$.

Most dictionary operations and methods are constant time, but there are some exceptions:

- The runtime of `update` is proportional to the size of the dictionary passed as a parameter, not the dictionary being updated.

- `keys`, `values` and `items` are constant time because they return iterators. But if you loop through the iterators, the loop will be linear.

The performance of dictionaries is one of the minor miracles of computer science. We will see how they work in "Hashtables".

Exercise 21-2.

Read the Wikipedia page on sorting algorithms at *http://en.wikipedia.org/wiki/Sorting_algorithm* and answer the following questions:

1. What is a "comparison sort?" What is the best worst-case order of growth for a comparison sort? What is the best worst-case order of growth for any sort algorithm?

2. What is the order of growth of bubble sort, and why does Barack Obama think it is "the wrong way to go?"

3. What is the order of growth of radix sort? What preconditions do we need to use it?

4. What is a stable sort and why might it matter in practice?

5. What is the worst sorting algorithm (that has a name)?

6. What sort algorithm does the C library use? What sort algorithm does Python use? Are these algorithms stable? You might have to Google around to find these answers.

7. Many of the non-comparison sorts are linear, so why does does Python use an $O(n \log n)$ comparison sort?

Analysis of Search Algorithms

A **search** is an algorithm that takes a collection and a target item and determines whether the target is in the collection, often returning the index of the target.

The simplest search algorithm is a “linear search”, which traverses the items of the collection in order, stopping if it finds the target. In the worst case it has to traverse the entire collection, so the runtime is linear.

The `in` operator for sequences uses a linear search; so do string methods like `find` and `count`.

If the elements of the sequence are in order, you can use a **bisection search**, which is $O(\log n)$. Bisection search is similar to the algorithm you might use to look a word up in a dictionary (a paper dictionary, not the data structure). Instead of starting at the beginning and checking each item in order, you start with the item in the middle and check whether the word you are looking for comes before or after. If it comes before, then you search the first half of the sequence. Otherwise you search the second half. Either way, you cut the number of remaining items in half.

If the sequence has 1,000,000 items, it will take about 20 steps to find the word or conclude that it’s not there. So that’s about 50,000 times faster than a linear search.

Bisection search can be much faster than linear search, but it requires the sequence to be in order, which might require extra work.

There is another data structure called a **hashtable** that is even faster — it can do a search in constant time — and it doesn’t require the items to be sorted. Python dictionaries are implemented using hashtables, which is why most dictionary operations, including the `in` operator, are constant time.

Hashtables

To explain how hashtables work and why their performance is so good, I start with a simple implementation of a map and gradually improve it until it's a hashtable.

I use Python to demonstrate these implementations, but in real life you wouldn't write code like this in Python; you would just use a dictionary! So for the rest of this chapter, you have to imagine that dictionaries don't exist and you want to implement a data structure that maps from keys to values. The operations you have to implement are:

`add(k, v)`:

> Add a new item that maps from key `k` to value `v`. With a Python dictionary, `d`, this operation is written `d[k] = v`.

`get(k)`:

> Look up and return the value that corresponds to key `k`. With a Python dictionary, `d`, this operation is written `d[k]` or `d.get(k)`.

For now, I assume that each key only appears once. The simplest implementation of this interface uses a list of tuples, where each tuple is a key-value pair:

```
class LinearMap:

    def __init__(self):
        self.items = []

    def add(self, k, v):
        self.items.append((k, v))

    def get(self, k):
        for key, val in self.items:
            if key == k:
                return val
        raise KeyError
```

`add` appends a key-value tuple to the list of items, which takes constant time.

`get` uses a `for` loop to search the list: if it finds the target key it returns the corresponding value; otherwise it raises a `KeyError`. So `get` is linear.

An alternative is to keep the list sorted by key. Then `get` could use a bisection search, which is $O(\log n)$. But inserting a new item in the middle of a list is linear, so this might not be the best option. There are other data structures that can implement `add` and `get` in log time, but that's still not as good as constant time, so let's move on.

One way to improve `LinearMap` is to break the list of key-value pairs into smaller lists. Here's an implementation called `BetterMap`, which is a list of 100 LinearMaps. As we'll see in a second, the order of growth for `get` is still linear, but `BetterMap` is a step on the path toward hashtables:

```
class BetterMap:
```

```
    def __init__(self, n=100):
        self.maps = []
        for i in range(n):
            self.maps.append(LinearMap())

    def find_map(self, k):
        index = hash(k) % len(self.maps)
        return self.maps[index]

    def add(self, k, v):
        m = self.find_map(k)
        m.add(k, v)

    def get(self, k):
        m = self.find_map(k)
        return m.get(k)
```

`__init__` makes a list of `n` `LinearMaps`.

`find_map` is used by `add` and `get` to figure out which map to put the new item in, or which map to search.

`find_map` uses the built-in function `hash`, which takes almost any Python object and returns an integer. A limitation of this implementation is that it only works with hashable keys. Mutable types like lists and dictionaries are unhashable.

Hashable objects that are considered equivalent return the same hash value, but the converse is not necessarily true: two objects with different values can return the same hash value.

`find_map` uses the modulus operator to wrap the hash values into the range from 0 to `len(self.maps)`, so the result is a legal index into the list. Of course, this means that many different hash values will wrap onto the same index. But if the hash function spreads things out pretty evenly (which is what hash functions are designed to do), then we expect *n/100* items per LinearMap.

Since the runtime of `LinearMap.get` is proportional to the number of items, we expect BetterMap to be about 100 times faster than LinearMap. The order of growth is still linear, but the leading coefficient is smaller. That’s nice, but still not as good as a hashtable.

Here (finally) is the crucial idea that makes hashtables fast: if you can keep the maximum length of the LinearMaps bounded, `LinearMap.get` is constant time. All you have to do is keep track of the number of items and when the number of items per LinearMap exceeds a threshold, resize the hashtable by adding more LinearMaps.

Here is an implementation of a hashtable:

```
class HashMap:

    def __init__(self):
        self.maps = BetterMap(2)
        self.num = 0

    def get(self, k):
        return self.maps.get(k)

    def add(self, k, v):
        if self.num == len(self.maps.maps):
```

```
            self.resize()

        self.maps.add(k, v)
        self.num += 1

    def resize(self):
        new_maps = BetterMap(self.num * 2)

        for m in self.maps.maps:
            for k, v in m.items:
                new_maps.add(k, v)

        self.maps = new_maps
```

Each `HashMap` contains a `BetterMap`; `__init__` starts with just 2 LinearMaps and initializes `num`, which keeps track of the number of items.

`get` just dispatches to `BetterMap`. The real work happens in `add`, which checks the number of items and the size of the `BetterMap`: if they are equal, the average number of items per LinearMap is 1, so it calls `resize`.

`resize` make a new `BetterMap`, twice as big as the previous one, and then "rehashes" the items from the old map to the new.

Rehashing is necessary because changing the number of LinearMaps changes the denominator of the modulus operator in `find_map`. That means that some objects that used to hash into the same LinearMap will get split up (which is what we wanted, right?).

Rehashing is linear, so `resize` is linear, which might seem bad, since I promised that `add` would be constant time. But remember that we don't have to resize every time, so `add` is usually constant time and only occasionally linear. The total amount of work to run `add` *n* times is proportional to *n*, so the average time of each `add` is constant time!

To see how this works, think about starting with an empty HashTable and adding a sequence of items. We start with two LinearMaps, so the first two adds are fast (no resizing required). Let's say that they take one unit of work each. The next add requires a resize, so we have to rehash the first two items (let's call that two more units of work) and then add the third item (one more unit). Adding the next item costs one unit, so the total so far is six units of work for four items.

The next `add` costs five units, but the next three are only one unit each, so the total is 14 units for the first eight adds.

The next `add` costs nine units, but then we can add seven more before the next resize, so the total is 30 units for the first 16 adds.

After 32 adds, the total cost is 62 units, and I hope you are starting to see a pattern. After *n* adds, where *n* is a power of two, the total cost is *2n-2* units, so the average work per add is a little less than 2 units. When *n* is a power of two, that's the best case; for other values of *n* the average work is a little higher, but that's not important. The important thing is that it is *O(1)*.

Figure 21-1 shows how this works graphically. Each block represents a unit of work. The columns show the total work for each add in order from left to right: the first two `adds` cost one unit, the third costs three units, etc.

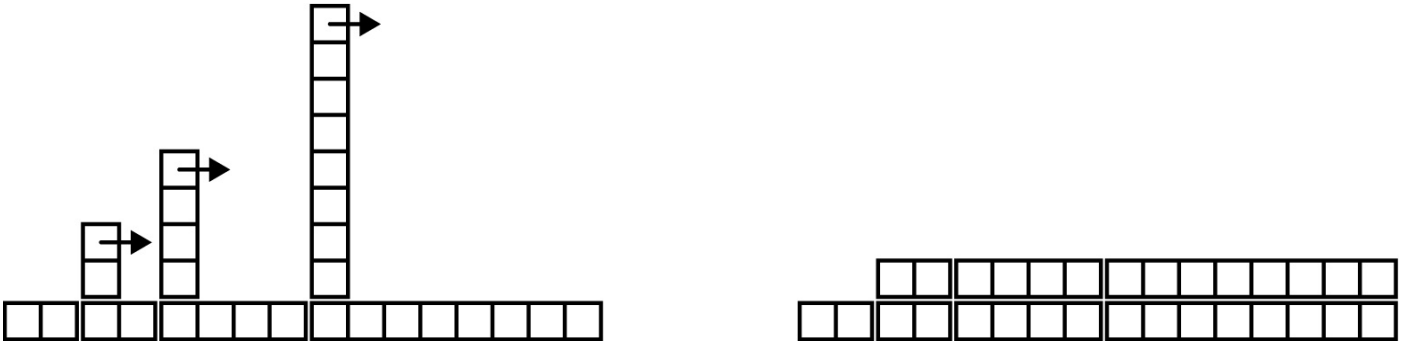


Figure 21-1. The cost of a hashtable add.

The extra work of rehashing appears as a sequence of increasingly tall towers with increasing space between them. Now if you knock over the towers, spreading the cost of resizing over all adds, you can see graphically that the total cost after *n* adds is $2n - 2$.

An important feature of this algorithm is that when we resize the HashTable it grows geometrically; that is, we multiply the size by a constant. If you increase the size arithmetically — adding a fixed number each time — the average time per `add` is linear.

You can download my implementation of HashMap from *http://thinkpython2.com/code/Map.py*, but remember that there is no reason to use it; if you want a map, just use a Python dictionary.

Glossary

analysis of algorithms:

A way to compare algorithms in terms of their runtime and/or space requirements.

machine model:

A simplified representation of a computer used to describe algorithms.

worst case:

The input that makes a given algorithm run slowest (or require the most space).

leading term:

In a polynomial, the term with the highest exponent.

crossover point:

The problem size where two algorithms require the same runtime or space.

order of growth:

A set of functions that all grow in a way considered equivalent for purposes of analysis of algorithms. For example, all functions that grow linearly belong to the same order of growth.

Big-Oh notation:

Notation for representing an order of growth; for example, *O(n)* represents the set of functions that grow linearly.

linear:

An algorithm whose runtime is proportional to problem size, at least for large problem sizes.

quadratic:

An algorithm whose runtime is proportional to n^2, where n is a measure of problem size.

search:

The problem of locating an element of a collection (like a list or dictionary) or determining that it is not present.

hashtable:

A data structure that represents a collection of key-value pairs and performs search in constant time.

[1] But if you get a question like this in an interview, I think a better answer is, “The fastest way to sort a million integers is to use whatever sort function is provided by the language I’m using. Its performance is good enough for the vast majority of applications, but if it turned out that my application was too slow, I would use a profiler to see where the time

was being spent. If it looked like a faster sort algorithm would have a significant effect on performance, then I would look around for a good implementation of radix sort.”

Index

S

About the Author

Allen Downey is a Professor of Computer Science at Olin College of Engineering. He has taught at Wellesley College, Colby College and U.C. Berkeley. He has a PhD in Computer Science from U.C. Berkeley and Master's and Bachelor's degrees from MIT.

Colophon

The animal on the cover of *Think Python* is the Carolina parrot, also known as the Carolina parakeet (*Conuropsis carolinensis*). This parrot inhabited the southeastern United States and was the only continental parrot with a habitat north of Mexico. At one time, it lived as far north as New York and the Great Lakes, although it was chiefly found from Florida to the Carolinas.

The Carolina parrot was mainly green with a yellow head and some orange coloring that appeared on the forehead and cheeks at maturity. Its average size ranged from 31–33 cm. It had a loud, riotous call and would chatter constantly while feeding. It inhabited tree hollows near swamps and riverbanks. The Carolina parrot was a very gregarious animal, living in small groups that could grow to several hundred parrots when feeding.

These feeding areas were, unfortunately, often the crops of farmers, who would shoot the birds to keep them away from the harvest. The birds' social nature caused them to fly to the rescue of any wounded parrot, allowing farmers to shoot down whole flocks at a time. In addition, their feathers were used to embellish ladies' hats, and some parrots were kept as pets. A combination of these factors led the Carolina parrot to become rare by the late 1800s, and poultry disease may have contributed to their dwindling numbers. By the 1920s, the species was extinct.

Today, there are more than 700 Carolina parrot specimens preserved in museums worldwide.

Many of the animals on O'Reilly covers are endangered; all of them are important to the world. To learn more about how you can help, go to *animals.oreilly.com*.

The cover image is from *Johnson's Natural History*. The cover fonts are URW Typewriter and Guardian Sans. The text font is Adobe Minion Pro; the heading font is Adobe Myriad Condensed; and the code font is Dalton Maag's Ubuntu Mono.

UNIT-II DATA TYPES IN PYTHON

UNIT II DATA TYPES IN PYTHON :

Lists, Tuples, Sets, Strings, Dictionary, Modules: Module Loading and Execution – Packages – Making Your Own Module – The Python Standard Libraries.

Lists :

- List is a sequence of values, which can be of different types. The values in list are called "elements" or "items"
- Each elements in list is assigned a number called "position" or "index"
- A list that contains no elements is called an empty list. They are created with empty brackets[]
- A list within another list is nested list

Creating a list :

The simplest way to create a new list is to enclose the elements in square brackets ([])

```
[10,20,30,40]
[100, "python" , 8.02]
```

1. LIST OPERATIONS:

1. Concatenation of list
2. Repetition of list

Concatenation: the '+' operator concatenate list

```
>>> a = [1,2,3]
>>> b = [4,5,6]
>>> c = a+b
>>> Print (a*2) => [1,2,3,1,2,3]
```

Repetition: the '*' operator repeats a list a given number of times

```
>>> a = [1,2,3]
>>> b = [4,5,6]
>>> print (a*2)= [1,2,3,1,2,3]
```

2. List looping: (traversing a list)

1. Looping in a list is used to access every element in list 2."for loop" is used to traverse the elements in list

```
eg: mylist =
    ["python","problem",100,6.28] for i
    in range (len (mylist)):
       print (mylist [i])
```

3.List Slices:

A subset of elements of list is called a slice of list. Eq: n = [1,2,3,4,5,6,7,8,9,10]

```
print (n[2:5])
print (n[-5])
print (n[5: ])
print (n[ : ])
```

4.Aliasing and cloning:

- when more than one variables refers to the same objects or list, then it is called aliasing.

```
a= [5,10,50,100]
b=a
b[0] = 80
print ("original list", a) = [5,10,50,100]
print ("Aliasing list", b) =
[80,5,10,50,100]
```

- Here both a & b refers to the same list. Thus, any change made with one object will affect other, since they are mutable objects.
- in general, it is safer to avoid aliasing when we are working with mutable objects

5. Cloning:

- Cloning creates a new list with same values under another name. Taking any slice of list create new list.
- Any change made with one object will not affect others. the easiest way to clone a new list is to use "slice operators"

```
a = [5,10,50,100]
b= a[ : ]
b[0] = 80
Print (" original list", a) =
[5,10,50,100] Print (" cloning list",
b) = [5,10,50,100]
```

List parameter:

- List can be passed as arguments to functions the list arguments are always passed by reference only.
- Hence, if the functions modifies the list the caller also changes. Eq: def head ():

```
        del t[ 0 ]
>>> letters = ['a','b','c']
>>> head (letters)
>>> letters
['b','c']
```

In above,

The parameters 't' and the variable 'letters' or aliases for the same objects An alternative way to write a function that creates and return a new list Eq: def tail (t):

```
        return t [1:]
>>> letters = ['a','b','c']
>>> result = tail (letters)
>>> result
    ['b','c']
```

In above,

The function leaves the original list unmodified and return all element in list except first element

TUPLES:

A tuple is a sequence of value which can be of any type and they are indexed by integers. Values in tuple are enclosed in parentheses and separated by comma. The elements in the tuple cannot be modified as in list (i.e) tuple are immutable objects

Creating tuple:

Tuple can be created by enclosing the element in parentheses separated by comma t = ('a','b','c','d')

To create a tuple with a single element we have to include a final comma

```
>>> t = 'a',
>>> type (t)
    < class 'tuple'>
```

Alternative way to create a tuple is the built-in function tuple which mean, it creates an empty tuple

```
>>> t = tuple ()
>>> t
>>> ( )
```

Accessing element in tuple:

If the argument in sequence, the result is a tuple with the elements of sequence.

```
>>>t= tuple('python')
>>> t
    ('p','y','t','h','o','n')
   t =
   ('a','b',100,8.02)
   print (t[0]) = 'a'
   print (t[1:3]) = ('b', 100 , 8.02)
```

Deleting and updating tuple:

Tuple are immutable, hence the elements in tuple cannot be updated / modified But we can delete the entire tuple by using keyword 'del'

```
Eg 1: a = (' programming', 200, 16.54, 'c',
'd') #Try changing an element.
      a[ 0 ] = 'python'    <------- Error,modifying not possible
      print (a [0])
Eg: # Deletion of
    tuple a =
    ('a','b','c','d')
    del (a) -------- delete entire tuple
    del a [1] < ------- error,deleting one element in tuple not possible
Eg: # replacing one tuple with
      another a = ('a','b','c','d')
      t = ('A',) + a[1: ]
      print (t) <-----('a','b','c','d')
```

Tuple Assignment:

- Tuple assignment is often useful to swap any number of values
- the number of variables in left and right of assignment operators must be equal
- A single assignment to paralleling assign value to all elements of tuple is the major benefit of tuple assignment

Eg: Tuple swapping in python

```
A= 100
B= 345
C= 450
print (" A & B:", A,"&",B)
    # Tuple assignments for two
variables A,B = B,A
print (" A&B after tuple assignment :
    ",A,"&",B) # Tuple assignment can be
    done for no of
variables A,B,C = C,A,B
print (" Tuple assignment for more variables:",
```

A,"&",B,"&",C) Output
A & B: 100 & 345
A&B after tuple assignment : 345 & 100
Tuple assignment for more variables: 450 & 345 & 100

Tuple as return value:

- Generally, function can only return one value but if the value is tuple the same as returning the multiple value
- Function can return tuple as return value

Eg: # the value of quotient & remainder are returned as
tuple def mod_div
(x,y): quotient
= x/y remainder
= x%y
return quotient, remainder
Input the seconds & get the hours minutes &
second sec = 4234
minutes,seconds= mod_div
(sec,60)
hours,minutes=mod_div(minut
es, 60)
print("%d seconds=%d hrs:: %d min:: %d
sec"%(sec,hours,minutes,seconds)) Output:
4234onds=1 hrs:: 10 min:: 34 sec

Set:

Sets are used to store multiple items in a single variable.

Set is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Tuple, and Dictionary, all with different qualities and usage.

A set is a collection which is *unordered*, *unchangeable**, and *unindexed*.

Example

Create a Set:

```
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

Set Items

Set items are unordered, unchangeable, and do not allow duplicate values.

Unordered

- Unordered means that the items in a set do not have a defined order.
- Set items can appear in a different order every time you use them, and cannot be referred to by index or key.

Unchangeable

- Set items are unchangeable, meaning that we cannot change the items after the set has been created.
- Duplicates Not Allowed
- Sets cannot have two items with the same value.

Example:

Duplicate values will be ignored:

```
thisset = {"apple", "banana", "cherry", "apple"}

print(thisset)
```

Get the Length of a Set

To determine how many items a set has, use the len() function.

Example

Get the number of items in a set:

```
thisset = {"apple", "banana", "cherry"}

print(len(thisset))
```

Set Items - Data Types

Set items can be of any data type:

Example

String, int and boolean data types:

```
set1 = {"apple", "banana", "cherry"}
set2 = {1, 5, 7, 9, 3}
set3 = {True, False, False}
```

A set can contain different data types:

Example

A set with strings, integers and boolean values:

```
set1 = {"abc", 34, True, 40, "male"}
```

Type()

From Python's perspective, sets are defined as objects with the data type 'set':

<class 'set'>

Example

What is the data type of a set?

```
myset = {"apple", "banana", "cherry"}
print(type(myset))
```

The set() Constructor

It is also possible to use the set() constructor to make a set.

Example

Using the set() constructor to make a set:

```
thisset = set(("apple", "banana", "cherry")) # note the double round-brackets
print(thisset)
```

Python Collections (Arrays)

There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered, unchangeable*, and unindexed. No duplicate members.
- **Dictionary** is a collection which is ordered** and changeable. No duplicate members.

*Set *items* are unchangeable, but you can remove items and add new items.

**As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

Access Items

You cannot access items in a set by referring to an index or a key.

But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

Example

Loop through the set, and print the values:

```
thisset = {"apple", "banana", "cherry"}

for x in thisset:
  print(x)
```

Set Methods

Python has a set of built-in methods that you can use on sets.

Method	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two other sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not

issuperset()	Returns whether this set contains another set or not
pop()	Removes an element from the set
remove()	Removes the specified element
symmetric_difference()	Returns a set with the symmetric differences of two sets
symmetric_difference_update()	inserts the symmetric differences from this set and another
union()	Return a set containing the union of sets
update()	Update the set with the union of this set and others

---

String

Python string is the collection of the characters surrounded by single quotes, double quotes, or triple quotes. The computer does not understand the characters; internally, it stores manipulated character as the combination of the 0's and 1's.

Each character is encoded in the ASCII or Unicode character. So we can say that Python strings are also called the collection of Unicode characters.

In Python, strings can be created by enclosing the character or the sequence of characters in the quotes. Python allows us to use single quotes, double quotes, or triple quotes to create the string.

Consider the following example in Python to create a string.

Syntax:

```
1. str = "Hi Python !"
```

Here, if we check the type of the variable **str** using a Python script

1. **print**(type(str)), then it will **print** a string (str).

In Python, strings are treated as the sequence of characters, which means that Python doesn't support the character data-type; instead, a single character written as 'p' is treated as the string of length 1.

Creating String in Python

We can create a string by enclosing the characters in single-quotes or double- quotes. Python also provides triple-quotes to represent the string, but it is generally used for multiline string or **docstrings**.

```
#Using single quotes
str1 = 'Hello Python'
print(str1)
#Using double quotes
str2 = "Hello Python"
print(str2)

#Using triple quotes
str3 = """Triple quotes are generally used for
    represent the multiline or
    docstring"""
print(str3)
```

Output:

```
Hello Python
Hello Python
Triple quotes are generally used for
    represent the multiline or
    docstring
```

Strings indexing and splitting

Like other languages, the indexing of the Python strings starts from 0. For example, The string "HELLO" is indexed as given in the below figure.

str = "HELLO"

H	E	L	L	O
0	1	2	3	4

str[0] = 'H'

str[1] = 'E'

str[2] = 'L'

str[3] = 'L'

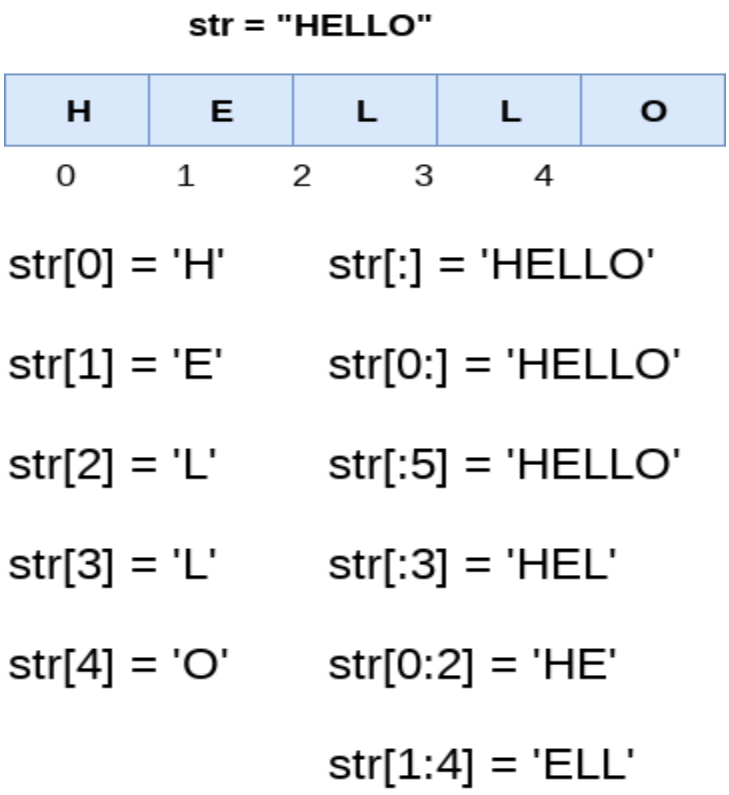
str[4] = 'O'

Consider the following example:

```
str = "HELLO"

print(str[0])

print(str[1])

print(str[2])

print(str[3])
print(str[4])
# It returns the IndexError because 6th index doesn't exist
print(str[6])
```

H

As shown in Python, the slice operator [] is used to access the individual characters of the string. However, we can use the : (colon) operator in Python to access the substring from the given string. Consider the following example.



Here, we must notice that the upper range given in the slice operator is always exclusive i.e., if str = 'HELLO' is given, then str[1:3] will always include str[1] = 'E', str[2] = 'L' and nothing else.

Consider the following example:

```
# Given String
str = "JAVATPOINT"
# Start Oth index to end
print(str[0:])
# Starts 1th index to 4th index
print(str[1:5])
# Starts 2nd index to 3rd index
print(str[2:4])
# Starts 0th to 2nd index
print(str[:3])
#Starts 4th to 6th index
print(str[4:7])
```

Output:

```
JAVATPOINT
AVAT
VA
JAV
TPO
```

Deleting the String

As we know that strings are immutable. We cannot delete or remove the characters from the string. But we can delete the entire string using the **del** keyword.

```
1. str = "JAVATPOINT"
2. del str[1]
```

Output:

```
TypeError: 'str' object doesn't support item deletion
```

Now we are deleting entire string.

```
1. str1 = "JAVATPOINT"
2. del str1
3. print(str1)
```

Output:

```
NameError: name 'str1' is not defined
```

String Operators

Operator	Description
+	It is known as concatenation operator used to join the strings given either side of the operator.
*	It is known as repetition operator. It concatenates the multiple copies of the same string.
[]	It is known as slice operator. It is used to access the sub-strings of a particular string.
[:]	It is known as range slice operator. It is used to access the characters from the specified range.

in	It is known as membership operator. It returns if a particular sub-string is present in the specified string.
not in	It is also a membership operator and does the exact reverse of in. It returns true if a particular substring is not present in the specified string.
r/R	It is used to specify the raw string. Raw strings are used in the cases where we need to print the actual meaning of escape characters such as "C://python". To define any string as a raw string, the character r or R is followed by the string.
%	It is used to perform string formatting. It makes use of the format specifiers used in C programming like %d or %f to map their values in python. We will discuss how formatting is done in python.

MODULES IN PYTHON

- A python module is a file that consists of python definition and statements. A module can define functions, classes and variables.
- It allows us to logically arrange related code and makes the code easier to understand and use.

1. **Import statement:**
   - An import statement is used to import python module in some python source file.

Syntax: import module1 [, module2 […module]]

Example:

```
>>>import math
>>>print (math.pi)
        3.14159265
```

2. **Import with renaming:**

The import a module by renaming it as follows,

```
>>>import math as a
>>>print("The value of pi is ",a.pi)
        The value of pi is 3.14159265
```

Writing modules:

- Any python source code file can be imported as a module into another python source file. For example, consider the following code named as support.py, which is python source file defining two function add(), display().

Support.py:

```
def add(a,b):
    print("The result is ",a+b)
    return
def display(p):
    print("welcome ",p)
```

return

The support.py file can be imported as a module into another python source file and its functions can be called from the new files as shown in the following code:

3. **Import file name**

```
import support          #import module support
support.add(3,4)        #calling add() of support module with two integers
support.add (3.5,4.7) #calling add() of support module with two real values
support.add ('a','b')   #calling add() of support module with two character
values support.add ("yona","alex")#calling add() of support module with two
string values support.display ('fleming')          #calling display() of support
module with a string value
```

Output:

```
The result is 7
The result is
8.2 The result
is ab
The result is yonaalex
Welcome, fleming
```

4. from……import statement:

- ✓ It allows us to import specific attributes from a module into the current namespace.

Syntax: from modulename import name1 [, name2[,……nameN]]

```
from support import add            #import module support
support.add(3,4)                   #calling add() of support module with two integers
support.add(3.5,4.7)               #calling add() of support module with two real
values support.add('a','b')        #calling add() of support module with two character
values support.add ("yona","alex")#calling add() of support module with two
string values support.display ('fleming')        #calling display() of support
module with a string value
```

Output:

```
The result is 7
The result is
8.2 The result
is ab
The result is yonaalex
Welcome, fleming
```

5. OS Module

- ✓ The OS module in python provide function for interacting with operating system
- ✓ To access the OS module have to import the OS module in our program

import os

method	example	description
name	Osname 'nt'	This function gives the name of the operating system
getcwd()	Os,getcwd() ,C;\\Python34'	Return the current working directory(CWD)of the file used to execute the code
mkdir(folder)	Os.mkdir("python")	Create a directory(folder) with the given name
rename(oldname,newname)	Os.rename("python","pspp")	Rename the directory or folder
remove("folder")	Os.remove("pspp")	Remove (delete)the directory or folder

getuid()	Os.getuid()	Return the current process's user id
environ	Os.nviron	Get the users environment

6. Sys Module

- ✓ Sys module provides information about constant, function and methods
- ✓ It provides access to some variables used or maintained by the interpreter

import sys

methods	example	description
sys.argv	sys.argv sys.argv(0) sys.argv(1)	Provides the list of command line arguments passed to a python script Provides to access the file name Provides to access the first input
sys.path	sys.path	It provide the search path for module
sys.path.append()	sys.path.append()	Provide the access to specific path to our program
sys.platform	sys.platform 'win32'	Provide information about the operating system platform
sys.exit	sys.exit <built.in function exit>	Exit from python

Steps to Create the Own Module

- ✓ Here we are going to create a calc module ; our module contains four functions

i.e add(),sub(),mul(),div()

Program for calculator module	output

Module name
;calc.py def
add(a,b); print(a+b)
def sub(a,b);
print(a-b)
def
mul(a,b);
print(a*b)
def
div(a,b);
print(a/b)
	import calculator
	calculator.add(2,3)
	
	Outcome
	>>>5

Package:

- ✓ A package is a collection of python module. Module is a single python file containing function definitions
- ✓ A package is a directory(folder)of python module containing an additional init py file, to differentiate a package from a directory
- ✓ Packages can be nested to any depth, provided that the corresponding directories contain their own init py file.
- ✓ __init py file is a directory indicates to the python interpreter that the directory should be treated like a python package init py is used to initialize the python package

Steps to Create a Package:

Step1: create the package directory

- ✓ Create the directory (folder)and give it your packages name
- ✓ Here the package name is calculator

Name	Data modified	Type
1. pycache___	05-12-2017	File folder
2.calculater	**08-12-2017**	**File folder**
3. DLLs	10-12-2017	File folder

Step2: write module for calculator directory add save the module in calculator directory

- ✓ Here four module have create for calculator directory

Local Disk (C)>Python34>Calculator

Name	Data modified	Type	Size
1. add	08-12-2017	File folder	1KB
2. div	08-12-2017	File folder	1KB
3. mul	08-12-2017	File folder	1KB
4. sub	08-12-2017	File folder	1KB

add.py	div.py	mul.py	sub.py
def add(a,b); print(a+b)	def div(a,b); print(a/b)	def mul(a,b); print(a*b)	def sub(a,b); print(a-b)

Step3: add the __init__.py file in the calculator directory

- ✓ A directory must contain the file named init__.py in order for python to consider it as a package

Add the following code in the init .py file

```
from * add import add
from * sub import sub
from * mul import mul
from * div import div
```

Local Disk (C):/Python34>Calculator

Name	Data modified	Type	Size
1. init_____	08-12-2017	File folder	1KB
2. add	08-12-2017	File folder	1KB
3. div	08-12-2017	File folder	1KB
4. mul	08-12-2017	File folder	1KB
5. sub	08-12-2017	File folder	1KB

Step4: To test your package

- ✓ Import calculator package in your program and add the path of your package in your program by using sys.path.append()

```
import calculator
importsys
sys.path.append("C:/Python34")
print ( calculator.add(10,5))
print ( calculator.sub(10,5))
print ( calculator.mul(10,5))
print ( calculator.div(10,5))
```

Example

Output :

```
>>> 15
    5
    50
```

UNIT-IV-Modules, Packages and Frameworks

UNIT IV MODULES, PACKAGES AND FRAMEWORKS

Modules: Introduction – Module Loading and Execution – Packages – Making Your Own Module – The Python Libraries for data processing, data mining and visualization- NUMPY, Pandas, Matplotlib, Plotly-Frameworks- -Django, Flask, Web2Py.

MODULES IN PYTHON

- A python module is a file that consists of python definition and statements. A module can define functions, classes and variables.
- It allows us to logically arrange related code and makes the code easier to understand and use.

1. Import statement:

- An import statement is used to import python module in some python source file.

Syntax: import module1 [, module2 […module]]

Example:

```
>>>import math
>>>print (math.pi)
        3.14159265
```

2. Import with renaming:

The import a module by renaming it as follows,

```
>>>import math as a
>>>print("The value of pi is ",a.pi)
        The value of pi is 3.14159265
```

Writing modules:

- Any python source code file can be imported as a module into another python source file. For example, consider the following code named as support.py, which is python source file defining two function add(), display().

Support.py:

```
def add(a,b):
    print("The result is ",a+b)
    return
def display(p):
    print("welcome ",p)
    return
```

The support.py file can be imported as a module into another python source file and its functions can be called from the new files as shown in the following code:

3. Import file name

```
import support          #import module support
```

```
support.add(3,4)        #calling add() of support module with two integers
support.add (3.5,4.7) #calling add() of support module with two real values
support.add ('a','b')   #calling add() of support module with two character
values support.add ("yona","alex")#calling add() of support module with two
string values support.display ('fleming')          #calling display() of support
module with a string value
```

Output:

```
` The result is 7
  The result is 8.2
  The result is ab
  The result is yonaalex
  Welcome, fleming
```

4. **from……import statement:**

- ✓ It allows us to import specific attributes from a module into the current namespace.

Syntax: from modulename import name1 [, name2[,……nameN]]

```
from support import add                #import module support
support.add(3,4)              #calling add() of support module with two integers
support.add(3.5,4.7)          #calling add() of support module with two real
values support.add('a','b')  #calling add() of support module with two character
values support.add ("yona","alex")#calling add() of support module with two
string values support.display ('fleming')          #calling display() of support
module with a string value
```

Output:

The result is 7

The result is 8.2

The result is ab

The result is yonaalex

Welcome, fleming

5. OS Module

- ✓ The OS module in python provide function for interacting with operating system
- ✓ To access the OS module have to import the OS module in our program

import os

method	example	description
name	Osname 'nt'	This function gives the name of the operating system
getcwd()	Os,getcwd() ,C;\\Python34'	Return the current working directory(CWD)of the file used to execute the code
mkdir(folder)	Os.mkdir("python")	Create a directory(folder) with the given name
rename(oldname,newname)	Os.rename("python","pspp")	Rename the directory or folder
remove("folder")	Os.remove("pspp")	Remove (delete)the directory or folder
getuid()	Os.getuid()	Return the current process's user id
environ	Os.nviron	Get the users environment

6. Sys Module

- ✓ Sys module provides information about constant, function and methods
- ✓ It provides access to some variables used or maintained by the interpreter

import sys

methods	example	description
sys.argv	sys.argv sys.argv(0) sys.argv(1)	Provides the list of command line arguments passed to a python script Provides to access the file name Provides to access the first input
sys.path	sys.path	It provide the search path for module
sys.path.append()	sys.path.append()	Provide the access to specific path to our program
sys.platform	sys.platform 'win32'	Provide information about the operating system platform
sys.exit	sys.exit <built.in function exit>	Exit from python

Steps to Create the Own Module

- ✓ Here we are going to create a calc module ; our module contains four functions i.e add(),sub(),mul(),div()

Program for calculator module	output

Module name
;calc.py def
add(a,b); print(a+b)
def sub(a,b);
print(a-b)
def
mul(a,b);
print(a*b)
def
div(a,b);
print(a/b)
	import calculator
	calculator.add(2,3)
	
	Outcome
	>>>5

Package:

- ✓ A package is a collection of python module. Module is a single python file containing function definitions
- ✓ A package is a directory(folder)of python module containing an additional init py file, to differentiate a package from a directory
- ✓ Packages can be nested to any depth, provided that the corresponding directories contain their own init py file.
- ✓ __init py file is a directory indicates to the python interpreter that the directory should be treated like a python package init py is used to initialize the python package

Steps to Create a Package:

Step1: create the package directory

- ✓ Create the directory (folder)and give it your packages name
- ✓ Here the package name is calculator

Name	Data modified	Type
1. pycache___	05-12-2017	File folder
2.calculater	**08-12-2017**	**File folder**
3. DLLs	10-12-2017	File folder

Step2: write module for calculator directory add save the module in calculator directory

- ✓ Here four module have create for calculator directory

Local Disk (C)>Python34>Calculator

Name	Data modified	Type	Size
1. add	08-12-2017	File folder	1KB
2. div	08-12-2017	File folder	1KB
3. mul	08-12-2017	File folder	1KB
4. sub	08-12-2017	File folder	1KB

add.py	div.py	mul.py	sub.py
def add(a,b); print(a+b)	def div(a,b); print(a/b)	def mul(a,b); print(a*b)	def sub(a,b); print(a-b)

Step3: add the __init__.py file in the calculator directory

- ✓ A directory must contain the file named init__.py in order for python to consider it as a package

Add the following code in the init .py file

```
from * add import add
from * sub import sub
from * mul import mul
from * div import div
```

Local Disk (C):/Python34>Calculator

Name	Data modified	Type	Size
1. init_____	08-12-2017	File folder	1KB
2. add	08-12-2017	File folder	1KB
3. div	08-12-2017	File folder	1KB
4. mul	08-12-2017	File folder	1KB
5. sub	08-12-2017	File folder	1KB

Step4: To test your package

- ✓ Import calculator package in your program and add the path of your package in your program by using sys.path.append()

```
import calculator
importsys
sys.path.append("C:/Python34")
print ( calculator.add(10,5))
print ( calculator.sub(10,5))
print ( calculator.mul(10,5))
print ( calculator.div(10,5))
```

Example

Output :

```
>>> 15
    5
    50
```

Python libraries for data processing:

Data processing services are available in various encodings, including CSV, XML, HTML, SQL, and JSON. Each situation requires a unique processing format. There are numerous programming languages. Python is frequently recommended as a viable alternative for machine learning applications due to its implementation of major libraries and cutting-edge technologies. Machine learning is built on data processing, and model success is highly dependent on the ability to read and transform data into the format required for the task at hand. Let us examine the various Python libraries in terms of the data types they provide.

Below, we have covered the Python libraries used for processing different types of data:

Tabular Data

Most of the large data is available in the tabular format, with rows referring to records and columns corresponding to features. Pandas in Python can handle such type data very perfectly. The advent of tabular data has evolved into a full-featured library that can handle both series and tabular data.

Text data

First, it's worth noting Python's extensive built-in text-processing capabilities. However, many natural language processing techniques, such as tokenization and lemmatization, may be done using NLTK. Along with that, Spacy is a good choice for advanced natural language processing and optimised pipelines.

Audio and musical data

Audio processing is enabled via libraries like librosa and essentia. Mido and pretty midi are good choices for symbolic music, like MIDI. Finally, music21 is a sophisticated library targeted at musicology analysis.

Images

Pillow is an image processing library in Python. Opencv is a computer vision library that can process videos or camera data. Because of its vast range of supported formats, imageio can give image data to the python script.

Python, in particular, is a highly regarded data processing language for a variety of reasons, including the following:

- Prototypes and experimentation with code are incredibly simple. Processing data, especially from less-than-clean sources, necessitates a great deal of tweaking, back and forth, and a struggle to capture all options.
- Python3 significantly improved multi-language support by making every string in the system UTF-8, which enables the processing of data encoded in different character sets by different languages.
- The standard library is quite strong and packed with essential modules that provide native support for common file types such as CSV files, zip files, and databases.
- The Python third-party library is enormous, and it has a wealth of excellent modules that enable it to increase the capabilities of a programme. There are also modules for geospatial data analysis, creating command-line interfaces, graphical interfaces, parsing data, and everything in between.
- Jupyter Notebooks allows you to execute code and receive immediate feedback. Python is quite agnostic about the development environment required, allowing it to function with anything from a simple text editor to more complex alternatives such as Visual Studio.

Numerical data

All above libraries have the power to read specific data formats. When this is converted in python objects and data structures, numpy usually comes into play, to manipulate these numerical values.

Before getting deep learning bazookas out, it is recommended to perform some analysis of the data, using sklearn, scipy and/or seaborn.

Data mining and visualization :

Data mining definition

The desired outcome from data mining is to create a model from a given data set that can have its insights generalized to similar data sets. A real-world example of a successful data mining application can be seen in automatic fraud detection from banks and credit institutions.

Your bank likely has a policy to alert you if they detect any suspicious activity on your account – such as repeated ATM withdrawals or large purchases in a state outside of your registered residence. How does this relate to data mining? Data scientists created this system by applying algorithms to classify and predict whether a transaction is fraudulent by comparing it against a historical pattern of fraudulent and non-fraudulent charges. The model "knows" that if you live in San Diego, California, it's highly likely that the thousand dollar purchases charged to a scarcely populated Russian province were not legitimate.

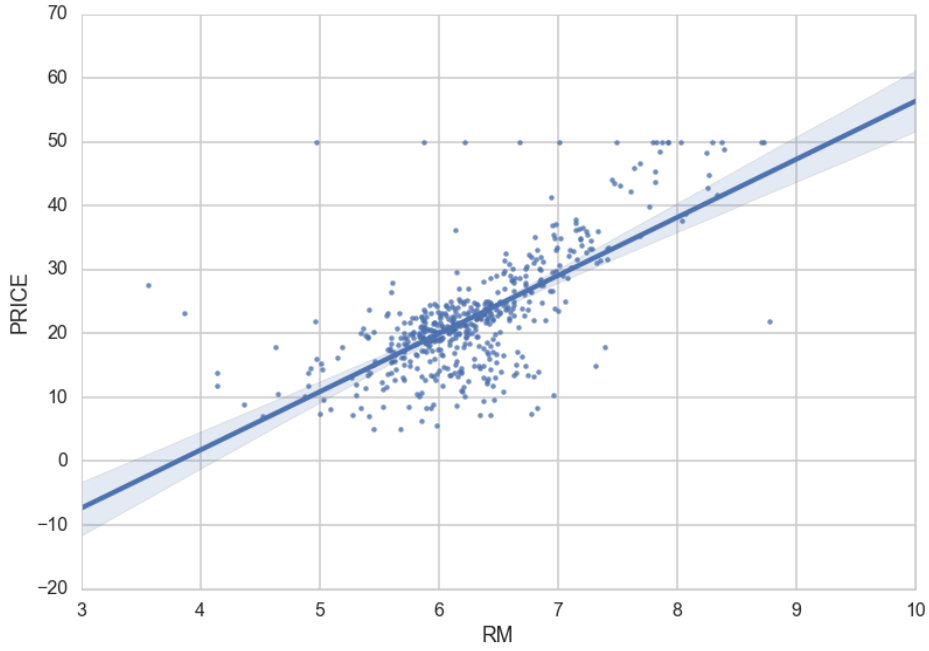
That is just one of a number of the powerful applications of data mining. Other applications of data mining include genomic sequencing, social network analysis, or crime imaging – but the most common use case is for analyzing aspects of the consumer life cycle. Companies use data mining to discover consumer preferences, classify different consumers based on their purchasing activity, and determine what makes for a well-paying customer – information that can have profound effects on improving revenue streams and cutting costs.

If you're struggling to find good data sets to begin your analysis, we've compiled 19 free data sets for your first data science project.

What are some data mining techniques?

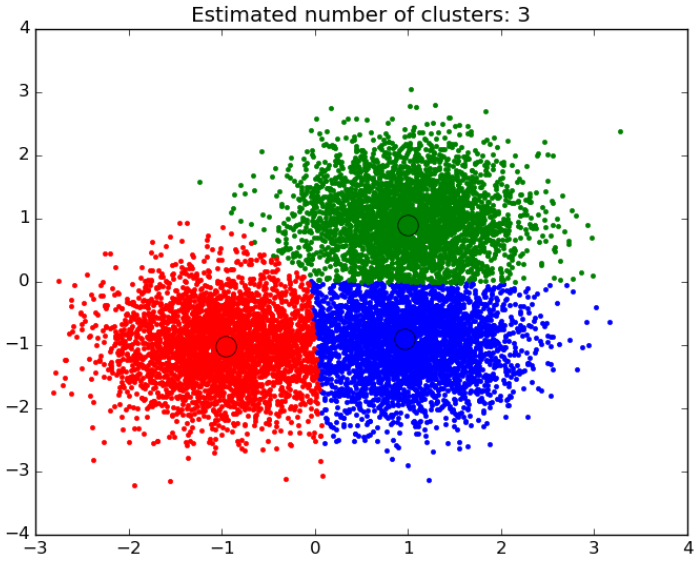
There are multiple ways to build predictive models from data sets, and a data scientist should understand the concepts behind these techniques, as well as how to use code to produce similar models and visualizations. These techniques include:

- **Regression** – Estimating the relationships between variables by optimizing the reduction of error.



An example of a scatterplot with a fitted linear regression model.

- **Classification** – Identifying what category an object belongs to. An example is classifying email as spam or legitimate, or looking at a person's credit score and approving or denying a loan request.
- **Cluster Analysis** – Finding natural groupings of data objects based upon the known characteristics of that data. An example could be seen in marketing, where analysis can reveal customer groupings with unique behavior – which could be applied in business strategy decisions.



An example of a scatter plot with the data segmented and colored by cluster.

- **Association and Correlation Analysis** – Looking to see if there are unique relationships between variables that are not immediately obvious. An example would be the famous case of beer and diapers: men who bought diapers at the end of the week were much more likely to buy beer, so stores placed them close to each other to increase sales.
- **Outlier analysis** – Examining outliers to examine potential causes and reasons for said outliers. An example of which is the use of outlier analysis in fraud detection, and trying to determine if a pattern of behavior outside the norm is fraud or not.

Data mining for business is often performed with a transactional and live database that allows easy use of data mining tools for analysis. One example of which would be an On-Line Analytical Processing server, or OLAP, which allows users to produce multi-dimensional analysis within the data server. OLAPs allow for business to query and analyze data without having to download static data files, which is helpful in situations where your database is growing on a daily basis. However, for someone looking to learn data mining and practicing on their own, an iPython notebook will be perfectly suited to handle most data mining tasks.

Let's walk through how to use Python to perform data mining using two of the data mining algorithms described above: regression and clustering.

Visualization :

Today's world, a lot of data is being generated on a daily basis. And sometimes to analyze this data for certain trends, patterns may become difficult if the data is in its raw format. To overcome this data visualization comes into play. Data visualization provides a good, organized pictorial representation of the data which makes it easier to understand, observe, analyze. In this tutorial, we will discuss how to visualize data using Python.

Python provides various libraries that come with different features for visualizing data. All these libraries come with different features and can support various types of graphs. In this tutorial, we will be discussing four such libraries.

- Matplotlib
- Seaborn
- Bokeh
- Plotly

Database Used

Tips Database

Tips database is the record of the tip given by the customers in a restaurant for two and a half months in the early 1990s. It contains 6 columns such as total_bill, tip, sex, smoker, day, time, size.

You can download the tips database from here

Example:

```
import pandas as pd

# reading the database

data = pd.read_csv("tips.csv")

# printing the top 10 rows

display(data.head(10))
```

Output:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
5	25.29	4.71	Male	No	Sun	Dinner	4
6	8.77	2.00	Male	No	Sun	Dinner	2
7	26.88	3.12	Male	No	Sun	Dinner	4
8	15.04	1.96	Male	No	Sun	Dinner	2
9	14.78	3.23	Male	No	Sun	Dinner	2

Matplotlib

Matplotlib is an easy-to-use, low-level data visualization library that is built on NumPy arrays. It consists of various plots like scatter plot, line plot, histogram, etc. Matplotlib provides a lot of flexibility.

To install this type the below command in the terminal.

```
pip install matplotlib
```

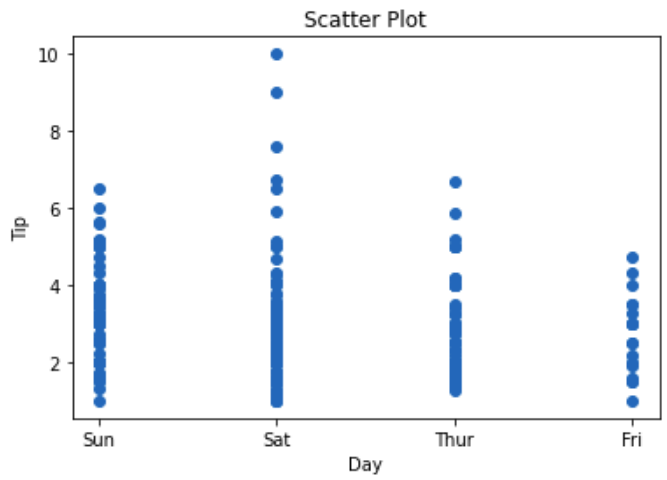
Scatter Plot

Scatter plots are used to observe relationships between variables and uses dots to represent the relationship between them. The **scatter()** method in the matplotlib library is used to draw a scatter plot.

Example:

```
import pandas as pd

import matplotlib.pyplot as plt

# reading the database

data = pd.read_csv("tips.csv")

# Scatter plot with day against tip

plt.scatter(data['day'], data['tip'])

# Adding Title to the Plot

plt.title("Scatter Plot")

# Setting the X and Y labels

plt.xlabel('Day')

plt.ylabel('Tip')

plt.show()
```

Output:



This graph can be more meaningful if we can add colors and also change the size of the points. We can do this by using the **c and s** parameter respectively of the scatter function. We can also show the color bar using the colorbar() method.

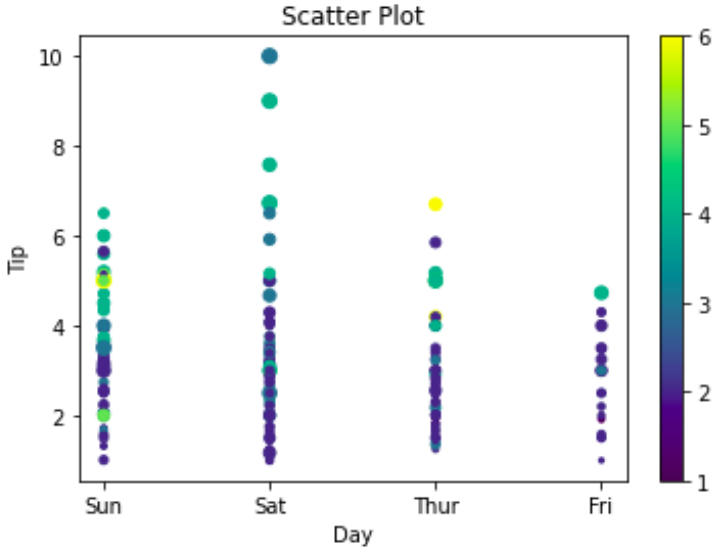
Example:

- Python3

```
import pandas as pd

import matplotlib.pyplot as plt

# reading the database

data = pd.read_csv("tips.csv")

# Scatter plot with day against tip

plt.scatter(data['day'], data['tip'], c=data['size'],

        s=data['total_bill'])

# Adding Title to the Plot
```

```
plt.title("Scatter Plot")

# Setting the X and Y labels

plt.xlabel('Day')

plt.ylabel('Tip')

plt.colorbar()

plt.show()
```

Output:



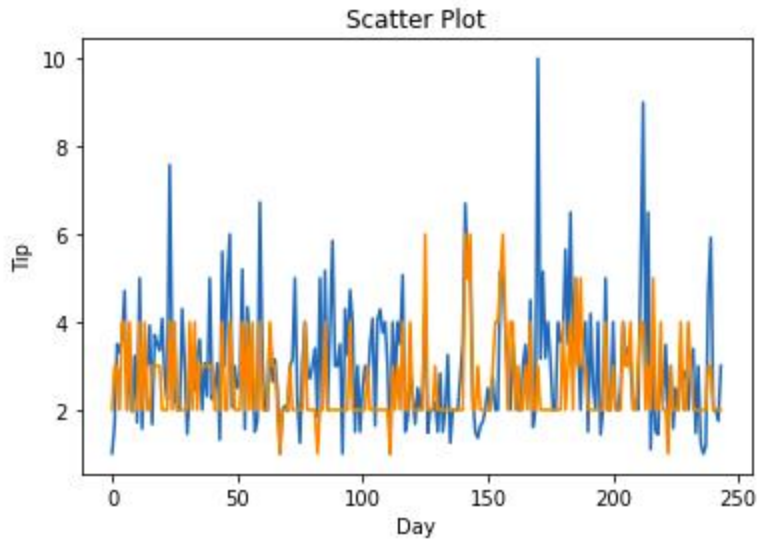
Line Chart

Line Chart is used to represent a relationship between two data X and Y on a different axis. It is plotted using the **plot()** function. Let's see the below example.

Example:

```
import pandas as pd

import matplotlib.pyplot as plt

# reading the database

data = pd.read_csv("tips.csv")

# Scatter plot with day against tip

plt.plot(data['tip'])

plt.plot(data['size'])

# Adding Title to the Plot

plt.title("Scatter Plot")

# Setting the X and Y labels

plt.xlabel('Day')

plt.ylabel('Tip')

plt.show()
```

Output:



Bar Chart

A bar plot or bar chart is a graph that represents the category of data with rectangular bars with lengths and heights that is proportional to the values which they represent. It can be created using the **bar()** method.

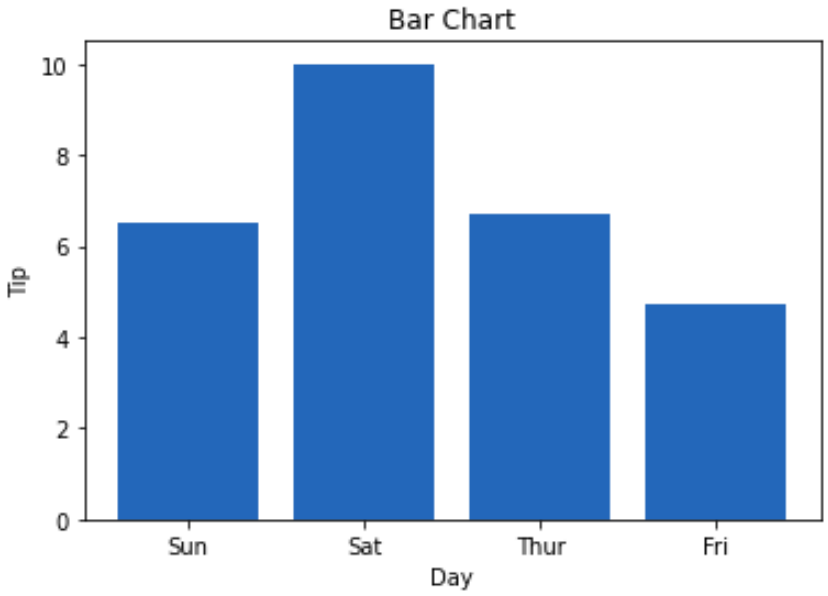
Example:

- Python3

```
import pandas as pd

import matplotlib.pyplot as plt

# reading the database

data = pd.read_csv("tips.csv")

# Bar chart with day against tip

plt.bar(data['day'], data['tip'])
```

```
plt.title("Bar Chart")

# Setting the X and Y labels

plt.xlabel('Day')

plt.ylabel('Tip')

# Adding the legends

plt.show()
```

Output:



Unit-V- Object oriented programming in python

UNIT V OBJECT ORIENTED PROGRAMMING IN PYTHON

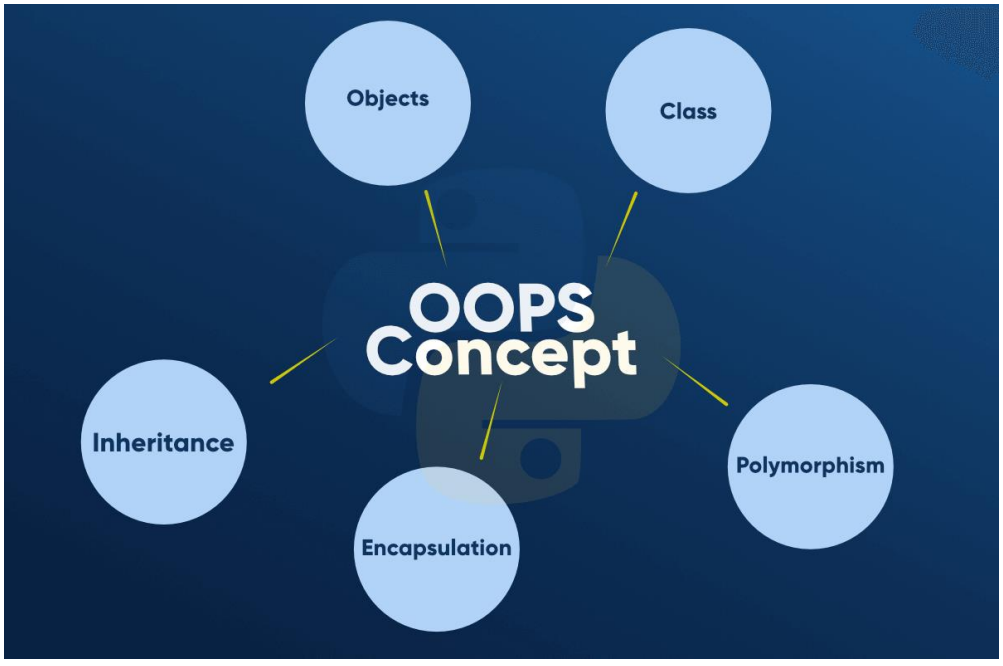
Creating a Class, Class methods, Class Inheritance, Encapsulation, Polymorphism, class method vs. static methods, Python object persistence.

Python OOPs Concepts:

In Python, object-oriented Programming (OOPs) is a programming paradigm that uses objects and classes in programming. It aims to implement real-world entities like inheritance, polymorphisms, encapsulation, etc. in the programming. The main concept of OOPs is to bind the data and the functions that work on that together as a single unit so that no other part of the code can access this data.

Main Concepts of Object-Oriented Programming (OOPs)

- Class
- Objects
- Polymorphism
- Encapsulation
- Inheritance



Class:

Class is a user-defined blueprint or prototype from which objects are created. Classes provide a means of bundling data and functionality together. Creating a new class creates a new type of object, allowing new instances of that type to be made. Each class instance can have attributes attached to it for maintaining its state. Class instances can also have methods (defined by their class) for modifying their state.

To understand the need for creating a class let's consider an example, let's say you wanted to track the number of dogs that may have different attributes like breed, age. If a list is used, the first element could be the dog's breed while the second element could represent its age. Let's suppose there are 100 different dogs, then how would you know which element is supposed to be which? What if you wanted to add other properties to these dogs? This lacks organization and it's the exact need for classes.

Class creates a user-defined data structure, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.

Some points on Python class:

- Classes are created by keyword class.
- Attributes are the variables that belong to a class.
- Attributes are always public and can be accessed using the dot (.) operator. Eg.: Myclass.Myattribute

Class Definition Syntax:

```
class ClassName:
# Statement-1
.
.
.
# Statement-N
```

Defining a class –

```
# Python  program to

# demonstrate defining

# a class

class Dog:
```

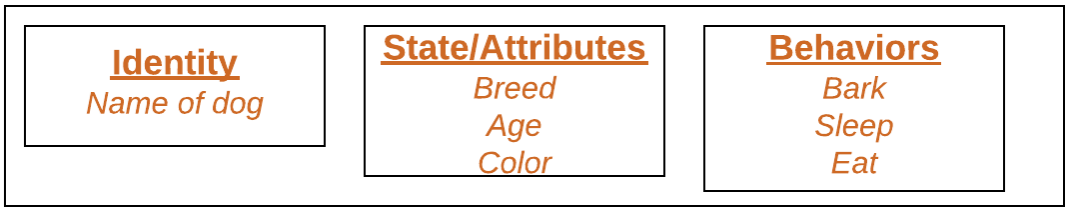
```
pass
```

In the above example, the class keyword indicates that you are creating a class followed by the name of the class (Dog in this case).

Class Objects:

An Object is an instance of a Class. A class is like a blueprint while an instance is a copy of the class with *actual values*. It's not an idea anymore, it's an actual dog, like a dog of breed pug who's seven years old. You can have many dogs to create many different instances, but without the class as a guide, you would be lost, not knowing what information is required. An object consists of :

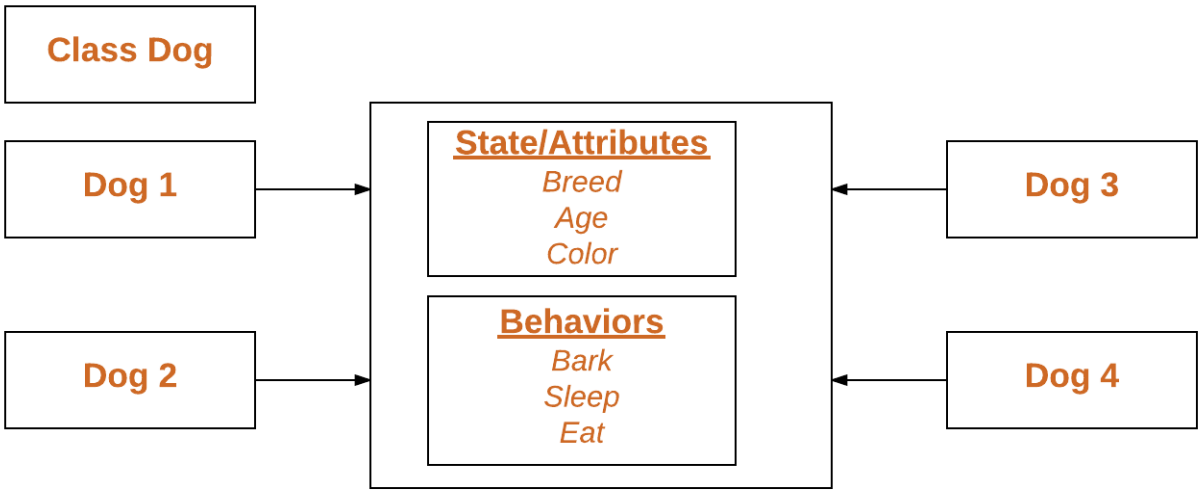
- **State:** It is represented by the attributes of an object. It also reflects the properties of an object.
- **Behavior:** It is represented by the methods of an object. It also reflects the response of an object to other objects.
- **Identity:** It gives a unique name to an object and enables one object to interact with other objects.



Declaring Objects (Also called instantiating a class)

When an object of a class is created, the class is said to be instantiated. All the instances share the attributes and the behavior of the class. But the values of those attributes, i.e. the state are unique for each object. A single class may have any number of instances.

Example:



Declaring an object –

```
# Python  program to

# demonstrate instantiating

# a class

class Dog

# A simple class

# attribute

attr1 = "mammal"

attr2 = "dog"

# A sample method
```

```
def fun(self):

print("I'm a", self.attr1)

print("I'm a", self.attr2)

# Driver code

# Object instantiation

Rodger = Dog()

# Accessing class attributes

# and method through objects

print(Rodger.attr1)

Rodger.fun()
```

Output:

```
mammal

I'm a mammal

I'm a dog
```

In the above example, an object is created which is basically a dog named Rodger. This class only has two class attributes that tell us that Rodger is a dog and a mammal.

The self

- Class methods must have an extra first parameter in the method definition. We do not give a value for this parameter when we call the method, Python provides it.
- If we have a method that takes no arguments, then we still have to have one argument.
- This is similar to this pointer in C++ and this reference in Java.

When we call a method of this object as myobject.method(arg1, arg2), this is automatically converted by Python into MyClass.method(myobject, arg1, arg2) – this is all the special self is about.

__init__ method

The __init__ method is similar to constructors in C++ and Java. Constructors are used to initializing the object's state. Like methods, a constructor also contains a collection of statements(i.e. instructions) that are executed at the time of Object creation. It runs as soon as an object of a class is instantiated. The method is useful to do any initialization you want to do with your object.

```
# A Sample class with init method

class Person:

# init method or constructor

def __init__(self, name):

self.name = name

# Sample Method

def say_hi(self):

print('Hello, my name is', self.name)

p = Person('Nikhil')

p.say_hi()
```

Output:

Hello, my name is Nikhil

Class and Instance Variables:

Instance variables are for data, unique to each instance and class variables are for attributes and methods shared by all instances of the class. Instance variables are variables whose value is assigned inside a constructor or method with self whereas class variables are variables whose value is assigned in the class.

Defining instance variable using a constructor.

```
# Python  program to show that the variables with a value

# assigned in the class declaration, are class variables and

# variables inside methods and constructors are instance

# variables.

# Class for Dog

class Dog:

# Class Variable

animal = 'dog'

# The init method or constructor

def __init__(self, breed, color):

# Instance Variable
```

```
self.breed = breed

self.color = color

# Objects of Dog class

Rodger = Dog("Pug", "brown")

Buzo = Dog("Bulldog", "black")

print('Rodger details:')

print('Rodger is a', Rodger.animal)

print('Breed: ', Rodger.breed)

print('Color: ', Rodger.color)

print('\nBuzo details:')

print('Buzo is a', Buzo.animal)

print('Breed: ', Buzo.breed)

print('Color: ', Buzo.color)

# Class variables can be accessed using class

# name also
```

```
print("\nAccessing class variable using class name")

print(Dog.animal)
```

Output:

```
Rodger details:
Rodger is a dog
Breed:  Pug
Color:  brown

Buzo details:
Buzo is a dog
Breed:  Bulldog
Color:  black

Accessing class variable using class name
dog
Defining instance variable using the normal method.
```

```
# Python  program to show that we can create

# instance variables inside methods

# Class for Dog

class Dog:
```

```
# Class Variable

animal = 'dog'

# The init method or constructor

def __init__(self, breed):

# Instance Variable

self.breed = breed

# Adds an instance variable

def setColor(self, color):

self.color = color

# Retrieves instance variable

def getColor(self):

return self.color

# Driver Code

Rodger = Dog("pug")

Rodger.setColor("brown")
```

```
print(Rodger.getColor())
```

Output:
brown

Inheritance:

Inheritance is the capability of one class to derive or inherit the properties from another class. The benefits of inheritance are:

1. It represents real-world relationships well.
2. It provides **reusability** of a code. We don’t have to write the same code again and again. Also, it allows us to add more features to a class without modifying it.
3. It is transitive in nature, which means that if class B inherits from another class A, then all the subclasses of B would automatically inherit from class A.

Below is a simple example of inheritance in Python

```
# A Python program to demonstrate inheritance

# Base or Super class. Note object in bracket.

# (Generally, object is made ancestor of all classes)

# In Python 3.x "class Person" is

# equivalent to "class Person(object)"

class Person(object):
```

```
# Constructor

def __init__(self, name):

self.name = name



# To get name

def getName(self):

return self.name



# To check if this person is an employee

def isEmployee(self):

return False




# Inherited or Subclass (Note Person in bracket)

class Employee(Person):



# Here we return true
```

```
def isEmployee(self):

return True

# Driver code

emp = Person("Geek1")  # An Object of Person

print(emp.getName(), emp.isEmployee())

emp = Employee("Geek2") # An Object of Employee

print(emp.getName(), emp.isEmployee())
```

Output:
False

True

What is object class?
1. Like Java Object class, in Python (from version 3.x), object is root of all classes.
2. In Python 3.x, "class Test(object)" and "class Test" are same.
3. In Python 2.x, "class Test(object)" creates a class with object as parent (called new style class) and "class Test" creates old style class (without object parent). Refer this for more details.

Subclassing (Calling constructor of parent class)

A child class needs to identify which class is its parent class. This can be done by mentioning the parent class name in the definition of the child class.
Eg: class **subclass_name (superclass_name)**:

_ _ _
_ _ _

```
# Python code to demonstrate how parent constructors
```

```
# are called.

# parent class

class Person( object ):

# __init__ is known as the constructor

def __init__(self, name, idnumber):

self.name = name

self.idnumber = idnumber

def display(self):

print(self.name)

print(self.idnumber)

# child class

class Employee( Person ):

def __init__(self, name, idnumber, salary, post):

self.salary = salary

self.post = post

# invoking the __init__ of the parent class

Person.__init__(self, name, idnumber)
```

```
# creation of an object variable or an instance

a = Employee('Rahul', 886012, 200000, "Intern")

# calling a function of the class Person using its instance

a.display()
```

Output:
Rahul

886012

‘a’ is the instance created for the class Person. It invokes the __init__() of the referred class. You can see ‘object’ written in the declaration of the class Person. In Python, every class inherits from a built-in basic class called ‘object’. The constructor i.e. the ‘__init__’ function of a class is invoked when we create an object variable or an instance of the class. The variables defined within __init__() are called as the instance variables or objects. Hence, ‘name’ and ‘idnumber’ are the objects of the class Person. Similarly, ‘salary’ and ‘post’ are the objects of the class Employee. Since the class Employee inherits from class Person, ‘name’ and ‘idnumber’ are also the objects of class Employee.

If you forget to invoke the __init__() of the parent class then its instance variables would not be available to the child class.

The following code produces an error for the same reason.

```
# Python program to demonstrate error if we

# forget to invoke __init__() of the parent.

class A:
```

```
def __init__(self, n = 'Rahul'):

self.name = n

class B(A):

def __init__(self, roll):

self.roll = roll

object = B(23)

print (object.name)
```

Output :

```
Traceback (most recent call last):

File "/home/de4570cca20263ac2c4149f435dba22c.py", line 12, in

print (object.name)

AttributeError: 'B' object has no attribute 'name'
```

Different forms of Inheritance:

1. Single inheritance: When a child class inherits from only one parent class, it is called single inheritance. We saw an example above.

2. Multiple inheritance: When a child class inherits from multiple parent classes, it is called multiple inheritance.

Unlike java, python shows multiple inheritance.

```
# Python example to show the working of multiple
```

```
# inheritance

class Base1(object):

def __init__(self):

self.str1 = "Geek1"

print("Base1")



class Base2(object):

def __init__(self):

self.str2 = "Geek2"

print("Base2")

class Derived(Base1, Base2):

def __init__(self):

# Calling constructors of Base1

# and Base2 classes

Base1.__init__(self)

Base2.__init__(self)

print("Derived")
```

```
def printStrs(self):

print(self.str1, self.str2)

ob = Derived()

ob.printStrs()
```

Output:

Base1

Base2

Derived

Geek1 Geek2

3.Multilevel inheritance:

When we have a child and grandchild relationship.

```
# A Python program to demonstrate inheritance

# Base or Super class. Note object in bracket.

# (Generally, object is made ancestor of all classes)

# In Python 3.x "class Person" is

# equivalent to "class Person(object)"

class Base(object):
```

```
# Constructor

def __init__(self, name):

self.name = name

# To get name

def getName(self):

return self.name

# Inherited or Sub class (Note Person in bracket)

class Child(Base):

# Constructor

def __init__(self, name, age):

Base.__init__(self, name)

self.age = age

# To get name

def getAge(self):

return self.age

# Inherited or Sub class (Note Person in bracket)

class GrandChild(Child):
```

```
# Constructor

def __init__(self, name, age, address):

Child.__init__(self, name, age)

self.address = address

# To get address

def getAddress(self):

return self.address

# Driver code

g = GrandChild("Geek1", 23, "Noida")

print(g.getName(), g.getAge(), g.getAddress())
```

Output:

```
23 oida
```

4.Hierarchical inheritance: More than one derived classes are created from a single base.

5.Hybrid inheritance: This form combines more than one form of inheritance. Basically, it is a blend of more than one type of inheritance.

Private members of parent class :

We don't always want the instance variables of the parent class to be inherited by the child class i.e. we can make some of the instance variables of the parent class private, which won't be available to the child class.

We can make an instance variable by adding double underscores before its name. For **example,**

Python program to demonstrate private members

```
# of the parent class

class C(object):

def __init__(self):

self.c = 21

# d is private instance variable

self.__d = 42

class D(C):

def __init__(self):

self.e = 84

C.__init__(self)

object1 = D()

# produces an error as d is private instance variable

print(object1.d)
```

Output :

```
File "/home/993bb61c3e76cda5bb67bd9ea05956a1.py", line 16, in
print (object1.d)
AttributeError: type object 'D' has no attribute 'd'
```

Polymorphism:

What is Polymorphism: The word polymorphism means having many forms. In programming, polymorphism means the same function name (but different signatures) being used for different types.

Example of inbuilt polymorphic functions :

```
# Python program to demonstrate in-built poly-

# morphic functions

# len() being used for a string

print(len("geeks"))

# len() being used for a list

print(len([10, 20, 30]))
```

Output:

```
5

3
```

Examples of user-defined polymorphic functions :

```
# A simple Python function to demonstrate

# Polymorphism
```

```
def add(x, y, z = 0):

return x + y+z

# Driver code

print(add(2, 3))

print(add(2, 3, 4))
```

Output:
5

9

Polymorphism with class methods:

The below code shows how Python can use two different class types, in the same way. We create a for loop that iterates through a tuple of objects. Then call the methods without being concerned about which class type each object is. We assume that these methods actually exist in each class.

```
class India():

def capital(self):

print("New Delhi is the capital of India.")

def language(self):
```

```
print("Hindi is the most widely spoken language of India.")

def type(self):

print("India is a developing country.")

class USA():

def capital(self):

print("Washington, D.C. is the capital of USA.")

def language(self):

print("English is the primary language of USA.")

def type(self):

print("USA is a developed country.")

obj_ind = India()

obj_usa = USA()

for country in (obj_ind, obj_usa):

country.capital()

country.language()
```

```
country.type()
```

Output:
New Delhi is the capital of India.

Hindi is the most widely spoken language of India.

India is a developing country.

Washington, D.C. is the capital of USA.

English is the primary language of USA.

USA is a developed country.

Polymorphism with Inheritance:

In Python, Polymorphism lets us define methods in the child class that have the same name as the methods in the parent class. In inheritance, the child class inherits the methods from the parent class. However, it is possible to modify a method in a child class that it has inherited from the parent class. This is particularly useful in cases where the method inherited from the parent class doesn't quite fit the child class. In such cases, we re-implement the method in the child class. This process of re-implementing a method in the child class is known as **Method Overriding**.

```
class Bird:

def intro(

many types of birds.")

self):
```

```
print("There are

def flight(self):

print("Most of the birds can fly but some cannot.")

class sparrow(Bird):

def flight(self):

print("Sparrows can fly.")

class ostrich(Bird):

def flight(self):

print("Ostriches cannot fly.")

obj_bird = Bird()

obj_spr = sparrow()

obj_ost = ostrich()

obj_bird.intro()

obj_bird.flight()

obj_spr.intro()
```

```
obj_spr.flight()

obj_ost.intro()

obj_ost.flight()
```

Output:

```
There are many types of birds.

Most of the birds can fly but some cannot.

There are many types of birds.

Sparrows can fly.

There are many types of birds.

Ostriches cannot fly.
```

Polymorphism with a Function and objects:

It is also possible to create a function that can take any object, allowing for polymorphism. In this example, let's create a function called "func()" which will take an object which we will name "obj". Though we are using the name 'obj', any instantiated object will be able to be called into this function. Next, let's give the function something to do that uses the 'obj' object we passed to it. In this case, let's call the three methods, viz., capital(), language() and type(), each of which is defined in the two classes 'India' and 'USA'. Next, let's create instantiations of both the 'India' and 'USA' classes if we don't have them already. With those, we can call their action using the same func() function:

```
def func(obj):

obj.capital()
```

```
obj.language()

obj.type()

obj_ind = India()

obj_usa = USA()

func(obj_ind)

func(obj_usa)
```

Code: Implementing Polymorphism with a Function

```
class India():

def capital(self):

print("New Delhi is the capital of India.")

def language(self):

print("Hindi is the most widely spoken language of India.")

def type(self):
```

```
print("India is a developing country.")

class USA():

def capital(self):

print("Washington, D.C. is the capital of USA.")

def language(self):

print("English is the primary language of USA.")

def type(self):

print("USA is a developed country.")

def func(obj):

obj.capital()

obj.language()

obj.type()

obj_ind = India()

obj_usa = USA()

func(obj_ind)
```

func(obj_usa)

Output:
New Delhi is the capital of India.

Hindi is the most widely spoken language of India.

India is a developing country.

Washington, D.C. is the capital of USA.

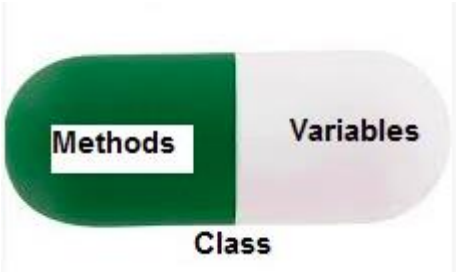
English is the primary language of USA.

USA is a developed country.

Encapsulation:

Encapsulation is one of the fundamental concepts in object-oriented programming (OOP). It describes the idea of wrapping data and the methods that work on data within one unit. This puts restrictions on accessing variables and methods directly and can prevent the accidental modification of data. To prevent accidental change, an object's variable can only be changed by an object's method. Those types of variables are known as **private variable.**

A class is an example of encapsulation as it encapsulates all the data that is member functions, variables, etc.



Consider a real-life example of encapsulation, in a company, there are different sections like the accounts section, finance section, sales section etc. The finance section handles all the financial transactions and keeps records of all the data related to finance. Similarly, the sales section handles all the sales-related activities and keeps records of all the sales. Now there may arise a situation when for some reason an official from the finance section needs all the data about sales in a particular month. In this case, he is not allowed to directly access the data of the sales section. He will first have to contact some other officer in the sales section and then request him to give the particular data. This is what encapsulation is. Here the data of the sales section

and the employees that can manipulate them are wrapped under a single name “sales section”. Using encapsulation also hides the data. In this example, the data of the sections like sales, finance, or accounts are hidden from any other section.

Protected members:

Protected members (in C++ and JAVA) are those members of the class that cannot be accessed outside the class but can be accessed from within the class and its subclasses. To accomplish this in Python, just follow **the convention** by prefixing the name of the member by a **single underscore “_”**.

Although the protected variable can be accessed out of the class as well as in the derived class(modified too in derived class), it is customary(convention not a rule) to not access the protected out the class body.

Note: The __init__ method is a constructor and runs as soon as an object of a class is instantiated.

```
# Python program to

# demonstrate protected members

# Creating a base class

class Base:

def __init__(self):

# Protected member

self._a = 2

# Creating a derived class
```

```
class Derived(Base):

def __init__(self):

# Calling constructor of

# Base class

Base.__init__(self)

print("Calling protected member of base class: ",

self._a)

# Modify the protected variable:

self._a = 3

print("Calling modified protected member outside class: ",

self._a)

obj1 = Derived()

obj2 = Base()

# Calling protected member

# Can be accessed but should not be done due to convention
```

```
print("Accessing protected member of obj1: ", obj1._a)

# Accessing the protected variable outside

print("Accessing protected member of obj2: ", obj2._a)
```

Output:
Calling protected member of base class: 2

Calling modified protected member outside class: 3

Accessing protected member of obj1: 3

Accessing protected member of obj2:

Python Classes and Methods:

Python is an "object-oriented programming language." This means that almost all the code is implemented using a special construct called classes. Programmers use classes to keep related things together. This is done using the keyword "class," which is a grouping of object-oriented constructs.

By the end of this tutorial you will be able to:

1. Define what is a class
2. Describe how to create a class
3. Define what is a method
4. Describe how to do object instantiation
5. Describe how to create instance attributes in Python

What is a class?

A class is a code template for creating objects. Objects have member variables and have behaviour associated with them. In python a class is created by the keyword class.

An object is created using the constructor of the class. This object will then be called the instance of the class. In Python we create instances in the following manner

```
Instance = class(arguments)
```

How to create a class:

The simplest class can be created using the class keyword. For example, let's create a simple, empty class with no functionalities.

```
>>> class Snake:
...     pass
...
>>> snake = Snake()
>>> print(snake)
<__main__.Snake object at 0x7f315c573550>
```

Attributes and Methods in class:

A class by itself is of no use unless there is some functionality associated with it. Functionalities are defined by setting attributes, which act as containers for data and functions related to those attributes. Those functions are called methods.

Attributes:

You can define the following class with the name Snake. This class will have an attribute name.

```
>>> class Snake:
...     name = "python" # set an attribute `name` of the class
```

```
...
```

You can assign the class to a variable. This is called object instantiation. You will then be able to access the attributes that are present inside the class using the dot . operator. For example, in the Snake example, you can access the attribute name of the class Snake.

```
>>> # instantiate the class Snake and assign it to variable snake
>>> snake = Snake()

>>> # access the class attribute name inside the class Snake.
>>> print(snake.name)
python
```

Methods:

Once there are attributes that “belong” to the class, you can define functions that will access the class attribute. These functions are called methods. When you define methods, you will need to always provide the first argument to the method with a self keyword.

For example, you can define a class Snake, which has one attribute name and one method change_name. The method change name will take in an argument new_name along with the keyword self.

```
>>> class Snake:
...     name = "python"
...
...     def change_name(self, new_name): # note that the first argument is self
...         self.name = new_name # access the class attribute with the self keyword
...
```

Now, you can instantiate this class Snake with a variable snake and then change the name with the method change_name.

```
>>> # instantiate the class
>>> snake = Snake()
```

```
>>> # print the current object name
>>> print(snake.name)
python

>>> # change the name using the change_name method
>>> snake.change_name("anaconda")
>>> print(snake.name)
anaconda
```

Instance attributes in python and the init method

You can also provide the values for the attributes at runtime. This is done by defining the attributes inside the init method. The following example illustrates this.

```
class Snake:

def __init__(self, name):
self.name = name

def change_name(self, new_name):
self.name = new_name
```

Now you can directly define separate attribute values for separate objects. For example,

```
>>> # two variables are instantiated
>>> python = Snake("python")
>>> anaconda = Snake("anaconda")

>>> # print the names of the two variables
>>> print(python.name)
python
>>> print(anaconda.name)
anaconda
```

Class method vs Static method in Python:

The class method in Python is a method, which is bound to the class but not the object of that class. The static methods are also same but there are some basic differences. For class methods, we need to specify @classmethod decorator, and for static method @staticmethod decorator is used.

Syntax for Class Method.

```
class my_class:
  @classmethod
 deffunction_name(cls, arguments):
    #Function Body
    return value
```

Syntax for Static Method.

```
class my_class:
  @staticmethod
  deffunction_name(arguments):
    #Function Body
    return value
```

differences between Classmethod and StaticMehtod?

Class Method	Static Method
The class method takes cls (class) as first argument.	The static method does not take any specific parameter.
Class method can access and modify the class state.	Static Method cannot access or modify the class state.
The class method takes the class as parameter to know about the state of that	Static methods do not know about class state. These methods are used to do some utility tasks by taking some

class.	parameters.
@classmethod decorator is used here.	@staticmethod decorator is used here.

The Static methods are used to do some utility tasks, and class methods are used for factory methods. The factory methods can return class objects for different use cases.

Example code

```
from datetime import date as dt
class Employee:
  def __init__(self, name, age):
    self.name = name
    self.age = age
  @staticmethod
  defisAdult(age):
    if age > 18:
      return True
    else:
      return False
  @classmethod
  defemp_from_year(emp_class, name, year):
    return emp_class(name, dt.today().year - year)
  def __str__(self):
    return 'Employee Name: {} and Age: {}'.format(self.name, self.age)
e1 = Employee('Dhiman', 25)
print(e1)
e2 = Employee.emp_from_year('Subhas', 1987)
print(e2)
print(Employee.isAdult(25))
print(Employee.isAdult(16))
```

Output

Employee Name: Dhiman and Age: 25

Employee Name: Subhas and Age: 31

True

False

Python object persistence:

The shelve module in Python's standard library is a simple yet effective tool for persistent data storage when using a relational database solution is not required. The shelf object defined in this module is dictionary-like object which is persistently stored in a disk file. This creates afile similar to dbm database on UNIX like systems. Only string data type can be used as key in this special dictionary object, whereas any picklable object can serve as value.

The shelve module defines three classes as follows −

Sr.No.	Module & Description
1	**Shelf** This is the base class for shelf implementations. It is initialized with dict-like object.
2	**BsdDbShelf** This is a subclass of Shelf class. The dict object passed to its constructor must support first(), next(), previous(), last() and set_location() methods.
3	**DbfilenameShelf** This is also a subclass of Shelf but accepts a filename as parameter to its constructor rather than dict object.

Easiest way to form a Shelf object is to use open() function defined in shelve module which return a DbfilenameShelf object.

open(filename, flag = 'c', protocol=None, writeback = False)

The filename parameter is assigned to the database created.

Default value for flag parameter is 'c' for read/write access. Other flags are 'w' (write only) 'r' (read only) and 'n' (new with read/write)

Protocol parameter denotes pickle protocol writeback parameter by default is false. If set to true, the accessed entries are cached. Every access calls sync() and close() operations hence process may be slow.

Following code creates a database and stores dictionary entries in it.

```
import shelve
s = shelve.open("test")
```

```
s['name'] = "Ajay"
s['age'] = 23
s['marks'] = 75
s.close()
```

This will create test.dir file in current directory and store key-value data in hashed form. The Shelf object has following methods available −

Sr.No.	Method & Description
1	**close()** synchronise and close persistent dict object.
2	**sync()** Write back all entries in the cache if shelf was opened with writeback set to True.
3	**get()** returns value associated with key
4	**items()** list of tuples – each tuple is key value pair
5	**keys()** list of shelf keys
6	**pop()** remove specified key and return the corresponding value.
7	**update()** Update shelf from another dict/iterable
8	**values()** list of shelf values

To access value of a particular key in shelf.

```
>>> s=shelve.open('test')
```

```
>>> s['age']
23
>>> s['age']=25
>>> s.get('age')
25
```

The items(), keys() and values() methods return view objects.

```
>>> list(s.items())
[('name', 'Ajay'), ('age', 25), ('marks', 75)]
>>> list(s.keys())
['name', 'age', 'marks']
>>> list(s.values())
['Ajay', 25, 75]
```

To remove a key-value pair from shelf

```
>>> s.pop('marks')
75
>>> list(s.items())
[('name', 'Ajay'), ('age', 25)]
```

Notice that key-value pair of marks-75 has been removed.

To merge items of another dictionary with shelf use update() method

```
>>> d={'salary':10000, 'designation':'manager'}
>>> s.update(d)
>>> list(s.items())
[('name', 'Ajay'), ('age', 25), ('salary', 10000), ('designation', 'manager')]
```

In this article we learned about shelve module which provides convenient mechanism for storing persistent dictionary object.

MA4151	APPLIED PROBABILITY AND STATISTICS FOR COMPUTER SCIENCE ENGINEERS	L T P C 3 1 0 4

COURSE OBJECTIVES:

- To encourage students to develop a working knowledge of the central ideas of Linear Algebra.
- To enable students to understand the concepts of Probability and Random Variables.
- To understand the basic probability concepts with respect to two dimensional random variables along with the relationship between the random variables and the significance of the central limit theorem.
- To apply the small / large sample tests through Tests of hypothesis.
- To enable the students to use the concepts of multivariate normal distribution and principal components analysis.

UNIT I LINEAR ALGEBRA 12

Vector spaces – norms – Inner Products – Eigenvalues using QR transformations – QR factorization – generalized eigenvectors – Canonical forms – singular value decomposition and applications – pseudo inverse – least square approximations.

UNIT II PROBABILITY AND RANDOM VARIABLES 12

Probability – Axioms of probability – Conditional probability – Bayes theorem – Random variables – Probability function – Moments – Moment generating functions and their properties – Binomial, Poisson , Geometric, Uniform, Exponential, Gamma and Normal distributions – Function of a random variable.

UNIT III TWO DIMENSIONAL RANDOM VARIABLES 12

Joint distributions – Marginal and conditional distributions – Functions of two dimensional random variables – Regression curve – Correlation.

UNIT IV TESTING OF HYPOTHESIS 12

Sampling distributions – Type I and Type II errors – Small and Large samples – Tests based on Normal, t, Chi square and F distributions for testing of mean , variance and proportions – Tests for independence of attributes and goodness of fit.

UNIT V MULTIVARIATE ANALYSIS 12

Random vectors and matrices – Mean vectors and covariance matrices – Multivariate normal density and its properties – Principal components – Population principal components – Principal components from standardized variables.

TOTAL : 60 PERIODS

COURSE OUTCOMES:

At the end of the course, students will be able to

- apply the concepts of Linear Algebra to solve practical problems.
- use the ideas of probability and random variables in solving engineering problems.
- be familiar with some of the commonly encountered two dimension random variables and be equipped for a possible extension to multivariate analysis.
- use statistical tests in testing hypothesis on data.
- develop critical thinking based on empirical evidence and the scientific approach to knowledge development.

Unit I : Linear Algebra

Vector spaces - norms - Inner products - Eigenvalues using QR transformations - QR factorization - Generalized Eigenvectors - Canonical forms - Singular value decomposition and applications - pseudo inverse - least Square approximations.

Vector spaces

notations

V - the given vector space,

x, y, z - vectors in V

F - the given number field

a, b, c - scalars in F

R - real field

C - complex field

R^3 - has dimension three

F^n - n-tuple of elements in F.

tr(M) - trace of matrix M.

Norms (length of a vector)

If $u = (a_1, a_2, \ldots, a_n)$ then $\|u\| = \sqrt{u \cdot u} = \sqrt{a_1^2 + a_2^2 + \cdots + a_n^2}$

Vector space (or) linear space

A vector space V over a field F consists of a set on which 2 operations (+ & ·) are defined so that for each pair of elements (x, y) in V there is a unique element $x+y$ in V and for each element $a \in V$ & each element $x \in V$ there is a unique element ax in V, s.t the following axioms hold

(i) $\forall x, y \in V$, $x+y = y+x$ (Commutative law)

(ii) $\forall x, y, z \in V$, $(x+y)+z = x+(y+z)$ (Associative law)

(iii) $\exists 0 \in V$ s.t $x+0 = x \quad \forall x \in V$.

(iv) $\forall x \in V$, $\exists y \in V$ s.t $x+y = 0$

(v) $\forall x \in V, \quad 1x = x$

(vi) $\forall a, b \in F$ & $\forall x \in V \quad (ab)x = a(bx)$

(vii) $\forall a \in F$ and $\forall x, y \in V, \quad a(x+y) = ax + ay$

(viii) $\forall a, b \in F$ and $\forall x \in V, \quad (a+b)x = ax + bx$

Ex: R, R^2, R^n are the vector spaces over R.

Note: The set of all n-tuples with entries from a field F is denoted by F^n.
vectors in F^n written as $\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}$

Theorem: Cancellation law for vector addition

If x, y, z are vectors in a vector space V s.t $x+z = y+z$ then $x = y$.

Proof: Given $x+z = y+z \quad \forall x, y, z \in V$ ——(1)

T.P: $x = y$

we have $\forall x \in V, \exists v \in V$ s.t $x+v = 0$ ——(2) by (iv)

Let $x = x+0$ by (iii)

$= x+(z+v)$ (ii)

$= (x+z)+v$ (ii)

$= (y+z)+v$

$= y+(z+v)$

$= y+0$

$= y$

$\Rightarrow x = y$

———

(1) Write the zero vector of $M_{3\times 4}$ from a field F.

Sol:

$$M = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}_{3\times 4}$$

If $M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ then what are M_{21}, M_{13}, M_{22}?

Sol:

$M_{21} = 4$, $M_{13} = 3$, $M_{22} = 5$

(2) Let V denote the set of all $m \times n$ matrices with real entries. Let F be the field of rational no's defined by $\forall A, B \in M_{m\times n}(F)$ and $c \in F$ $(A+B)_{ij} = A_{ij} + B_{ij}$ ——(1) & $(cA)_{ij} = cA_{ij}$ ——(2)

Is V a vector space over F?

Sol:

Let $A = [a_{ij}]_{m\times n}$, $B = [b_{ij}]_{m\times n}$, $C = [c_{ij}]_{m\times n} \in V(F)$

where $a_{ij}, b_{ij}, c_{ij} \in F$

(i) $A + B = [a_{ij}]_{m\times n} + [b_{ij}]_{m\times n}$

$= [a_{ij} + b_{ij}]_{m\times n} = [b_{ij} + a_{ij}]_{m\times n} = [b_{ij}]_{m\times n} + [a_{ij}]_{m\times n}$

$= B + A$

(ii) $(A+B)+C = \left([a_{ij}]_{m\times n} + [b_{ij}]_{m\times n}\right) + [c_{ij}]_{m\times n}$

$= [a_{ij} + b_{ij}]_{m\times n} + [c_{ij}]_{m\times n}$

$= [a_{ij} + b_{ij} + c_{ij}]_{m\times n}$

$= [a_{ij} + (b_{ij} + c_{ij})]_{m\times n}$

$= [a_{ij}]_{m\times n} + [b_{ij} + c_{ij}]_{m\times n}$

$= A + [B + C]$

(iii) $A + O = [a_{ij}]_{m\times n} + [0]_{m\times n}$

$= [a_{ij} + 0]_{m\times n} = [a_{ij}]_{m\times n} = A$

Also $O + A = A$

$\therefore O = O_{m\times n} = [0]_{m\times n}$ is the identity matrix

(iv) $A + (-A) = [a_{ij}]_{m\times n} + [-a_{ij}]_{m\times n}$

$= [a_{ij} - a_{ij}]_{m\times n} = [0]_{m\times n} = O$

Also $(-A) + A = O$

$\therefore A + (-A) = O = (-A) + A$

$-A$ is the additive inverse of A

(v) $1 \cdot A = 1\,[a_{ij}]_{m\times n} = [1 \cdot a_{ij}]_{m\times n} = [a_{ij}]_{m\times n} = A$

(vi) $(ab)A = (ab)\,[a_{ij}]_{m\times n} = [(ab)a_{ij}]_{m\times n}$

$= [a[ba_{ij}]]_{m\times n} = a[ba_{ij}]_{m\times n}$

$= a[b[a_{ij}]_{m\times n}]$

$= a(bA)$

(vii) $a(A+B) = a\left[\,[a_{ij}]_{m\times n} + [b_{ij}]_{m\times n}\,\right]$

$= a\,[a_{ij} + b_{ij}]_{m\times n}$

$= [a(a_{ij} + b_{ij})]_{m\times n}$

$= [aa_{ij} + ab_{ij}]_{m\times n}$

$= [aa_{ij}]_{m\times n} + [ab_{ij}]_{m\times n}$

$= a[a_{ij}]_{m\times n} + a[b_{ij}]_{m\times n}$

$= aA + aB$

(viii) $\forall\, a, b \in F$, $A \in V$

$(a+b)A = (a+b)\,[a_{ij}]_{m\times n}$

$= a[a_{ij}]_{m\times n} + b[a_{ij}]_{m\times n}$

$= aA + bA$

Space F^n

Let F be any field. If $u = (a_1, a_2, \dots a_n) \in F^n$

and $v = (b_1, b_2, \dots b_n) \in F^n$ then

$u+v = (a_1+b_1,\ a_2+b_2,\ \dots\ a_n+b_n)$ & $cu = (ca_1, ca_2, \dots ca_n)$

(1) In any vector space V, $1(x+y) = x+y$ for any $x, y \in V$ and any $a, b \in F$

(b)

in a vector space V, $\forall x, y \in V$ and $\forall a, b \in F$

$$(a+b)+(x+y) = (a+b)x + (a+b)y \quad \text{(vii)}$$

$$= ax + bx + ay + by \quad \text{(viii)}$$

$$= ax + ay + bx + by$$

(a) Let V and W be vector spaces over a field F. Let $Z = \{(v, w) \mid v \in V \text{ and } w \in W\}$ P.T Z is a vector space over F with the operations $(v_1, w_1) + (v_2, w_2) = (v_1 + v_2, w_1 + w_2)$ and $c(v_1, w_1) = (cv_1, cw_1)$

Proof:

Given $Z = \{(v, w) \mid v \in V \text{ \& } w \in W\}$

$$(v_1, w_1) + (v_2, w_2) = (v_1 + v_2, w_1 + w_2) \quad \text{---(1)}$$

$$c(v_1, w_1) = (cv_1, cw_1) \quad \text{---(2)}$$

(i) $(v_1, w_1), (v_2, w_2) \in Z$

$$\Rightarrow (v_1, w_1) + (v_2, w_2) = (v_1 + v_2, w_1 + w_2) \quad \text{---(3)}$$

$$(v_2, w_2) + (v_1, w_1) = (v_2 + v_1, w_2 + w_1) \quad \text{---(4)}$$

From (3) & (4), $(v_1, w_1) + (v_2, w_2) = (v_2, w_2) + (v_1, w_1)$

(ii) $(v_1, w_1), (v_2, w_2), (v_3, w_3) \in Z$

$$[(v_1, w_1) + (v_2, w_2)] + (v_3, w_3) = (v_1 + v_2, w_1 + w_2) + (v_3, w_3)$$

$$= (v_1 + v_2 + v_3, w_1 + w_2 + w_3) \quad \text{---(5)}$$

$$(v_1, w_1) + [(v_2, w_2) + (v_3, w_3)] = (v_1, w_1) + [(v_2 + v_3, w_2 + w_3)]$$

$$= (v_1 + v_2 + v_3, w_1 + w_2 + w_3) \quad \text{---(6)}$$

From (5) & (6),

$$[(v_1, w_1) + (v_2, w_2)] + (v_3, w_3) = (v_1, w_1) + [(v_2, w_2) + (v_3, w_3)]$$

(iii) $(0v, 0w) \in Z$

$$(v, w) + (0v, 0w) = (v + 0v, w + 0w) = (v, w)$$

$(0v, 0w)$ is the zero vector in Z.

(iv) $(v,w) \in Z$, $\exists\ (-v,-w) \in Z$

s.t $(v,w) + (-v,-w) = (v-v, w-w) = (0,0)$

$= (0v, 0w)$ is true.

$\Rightarrow \forall (v,w) \in Z,\ \exists\ (-v,-w)$ s.t $(v,w) + (-v,-w) = 0_Z$

(v) $\forall (v,w) \in Z \qquad 1(v,w) = (1v, 1w) = (v,w)$

(vi) $\forall a,b \in F \quad \forall (v,w) \in Z$

$(ab)(v,w) = (abv, abw) = a(bv, bw)$

$= a(b(v,w))$ is true.

(vii) $\forall c \in F \Rightarrow \forall (v_1,w_1), (v_2,w_2) \in Z$

$c((v_1,w_1)+(v_2,w_2)) = c(v_1+v_2) + c(w_1+w_2)$

$= (cv_1 + cv_2, cw_1 + cw_2)$

$= c(v_1, w_1) + c(v_2, w_2)$

(viii) $\forall (a,b) \in F \Rightarrow \forall (v,w) \in Z,$

$(a+b)(v,w) = (av + bv, aw + bw)$

$= (av, aw) + (bv, bw)$

$= a(v,w) + b(v,w)$ is true.

Hence Z is a vector space.

What is the zero vector in the vector space R^4?

Sol:

$(0\ 0\ 0\ 0)$ or $\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$

Function Space $\mathcal{F}[S,F]$

Let S be any non empty set and F be any field.

Let $\mathcal{F}(S,F)$ denote the set of all functions from S to F.

The set $\mathcal{F}(S,F)$ is a vector space with the operations of addition and scalar multiplication defined for $f, g \in \mathcal{F}(S,F)$ and $c \in F$ by

$$(f+g)(s) = f(s) + g(s) \quad \text{and} \quad (cf)(s) = c[f(s)]$$

for each $s \in S$

(1) Let $S = \{0,1\}$ and $F = R$. In $\mathcal{F}(S,R)$ show that $f = g$ and $f + g = h$ where $f(t) = 2t + 1$, $g(t) = 1 + 4t - 2t^2$, $h(t) = 5^t + 1$

Sol:

Given $S = \{0,1\}$ and $F = R$

$$f(t) = 2t + 1 \quad \text{---(1)}$$
$$g(t) = 1 + 4t - 2t^2 \quad \text{---(2)}$$
$$h(t) = 5^t + 1 \quad \text{---(3)}$$

T.P: $f = g$ and $f + g = h$

(i) $f(0) = 1$, $g(0) = 1$ | $f(1) = 2+1 = 3$, $g(1) = 1+4-2 = 3$

$\Rightarrow \boxed{f = g}$

(ii) $f(0) + g(0) = 2$, $h(0) = 5^0 + 1 = 1 + 1 = 2$ | $f(1) + g(1) = 6$, $h(1) = 5^1 + 1 = 6$

$\Rightarrow \boxed{f + g = h}$

~~Polynomial Space~~ $P(F)$ Sequence F

Note: Let F be any field. A sequence in F is a function σ from the +ive integers into F. If $\{a_n\}$ & $\{b_n\}$ are in V and $t \in F$, define

$$\{a_n\} + \{b_n\} = \{a_n + b_n\} \quad \text{and} \quad t\{a_n\} = \{ta_n\}.$$

---

① Let V be the set of sequence $\{a_n\}$ of real numbers. For $\{a_n\}, \{b_n\} \in V$ and any real no t. define $\{a_n\} + \{b_n\} = \{a_n + b_n\}$ & $t\{a_n\} = \{ta_n\}$ P.T with these operations, V is a vector space over R.

Proof: $\{a_n\}, \{b_n\} \in V$

(i) $\Rightarrow \{a_n\} + \{b_n\} = \{a_n + b_n\}$

$\{b_n\} + \{a_n\} = \{b_n + a_n\}$

$\Rightarrow \{a_n\} + \{b_n\} = \{b_n\} + \{a_n\}$ is true.

(ii) $\forall \{a_n\}, \{b_n\}, \{c_n\} \in V$

$\Rightarrow [\{a_n\} + \{b_n\}] + \{c_n\} = \{a_n + b_n\} + \{c_n\}$

$= \{a_n + b_n + c_n\}$

$\{a_n\} + [\{b_n\} + \{c_n\}] = \{a_n\} + \{b_n + c_n\}$

$= \{a_n + b_n + c_n\}$

$\therefore [\{a_n\} + \{b_n\}] + \{c_n\} = \{a_n\} + [\{b_n\} + \{c_n\}]$ is true.

(iii) $\{0\} \in V$, $\{a_n\} + \{0\} = \{a_n + 0\} = \{a_n\}$ $\forall \{a_n\} \in V$

(iv) ~~$\exists x \in V, \forall y \in V$ s.t $x + y = 0$~~ $\{a_n\} + \{-a_n\} = \{a_n + (-a_n)\}$

$= \{0\}$ is true.

(v) $1\{a_n\} = \{1a_n\} = \{a_n\}$

(vi) $a, b \in F$, $\{x_n\} \in V$

$\Rightarrow (ab)\{x_n\} = \{ab\, x_n\}$

$a\{bx_n\} = \{abx_n\}$

$\Rightarrow (ab)\{x_n\} = a\{bx_n\}$ is true.

$a \in F, \quad \{x_n\}, \{y_n\} \in V$

$$a[\{x_n\} + \{y_n\}] = a\{x_n + y_n\}$$
$$= \{ax_n + ay_n\}$$

$$a\{x_n\} + a\{y_n\} = \{ax_n\} + \{ay_n\}$$
$$= \{ax_n + ay_n\}$$

$$\Rightarrow a[\{x_n\} + \{y_n\}] = a\{x_n\} + a\{y_n\}$$

(viii) $a, b \in F, \quad \{x_n\} \in V$

$$\Rightarrow (a+b)\{x_n\} = \{(a+b)x_n\}$$
$$= \{ax_n + bx_n\}$$

$$a\{x_n\} + b\{x_n\} = \{ax_n\} + \{bx_n\}$$
$$= \{ax_n + bx_n\}$$

$$\Rightarrow (a+b)\{x_n\} = a\{x_n\} + b\{x_n\}$$

Hence V is a vector space over R.

Inner Products and Norms

Inner product spaces

Def:

Let V be a vector space over F. An inner product on V is a function that assigns to every ordered pair of vectors x & y in V, a scalar in F, denoted $\langle x, y \rangle$ such that for all x, y and z in V and all c in F, the following axioms hold.

(i) $\langle x+z, y \rangle = \langle x, y \rangle + \langle z, y \rangle$

(ii) $\langle cx, y \rangle = c \langle x, y \rangle$

(iii) $\langle \overline{x, y} \rangle = \langle y, x \rangle$ where the bar denotes complex conjugation

(iv) $\langle x, x \rangle > 0$ if $x \neq 0$

① Let $x = (2, 1+i, i)$ and $y = (2-i, 2, 1+2i)$ be vectors in C^3. Compute $\langle x, y \rangle$.

Sol:

$$\langle x, y \rangle = 2(\overline{2-i}) + (1+i)2 + i(\overline{1+2i})$$
$$= 2(2+i) + (1+i)2 + i(1-2i)$$
$$= 8+5i$$

---

② S.T $\langle (a,b), (c,d) \rangle = ac - bd$ on R^2 is not an inner product on given vector space.

Sol.

Let $(a,b) = (0,1) \in R^2$

$\langle (a,b), (c,d) \rangle = ac - bd$

$\langle (0,1), (0,1) \rangle = (0)(0) - (1)(1) = 0 - 1 = -1$

$\therefore \langle (0,1), (0,1) \rangle < 0$ which $\Rightarrow\Leftarrow$ to $\langle x, x \rangle > 0$

$\therefore \langle (a,b), (c,d) \rangle = ac - bd$ is not an inner product on R^2.

---

③ S.T the vector space H with $\langle \cdot, \cdot \rangle$ defined on the interval $[0, 2\pi]$ with the inner product $\langle f, g \rangle = \frac{1}{2\pi} \int_0^{2\pi} f(t)\overline{g(t)}\, dt$ is an inner product space.

Sol:

T.P: $\langle f, g \rangle$ is an inner product space.

(i) T.P: $\langle f_1 + f_2, g \rangle = \langle f_1, g \rangle + \langle f_2, g \rangle$

L.H.S $= \langle f_1 + f_2, g \rangle = \frac{1}{2\pi} \int_0^{2\pi} (f_1 + f_2)(t)\ \overline{g(t)}\, dt$

$= \frac{1}{2\pi} \int_0^{2\pi} f_1(t)\, \overline{g(t)}\, dt + \frac{1}{2\pi} \int_0^{2\pi} f_2(t)\, \overline{g(t)}\, dt$

$= \langle f_1, g \rangle + \langle f_2, g \rangle$

$=$ R.H.S

T.P: $\langle cf, g\rangle = c\langle f, g\rangle$

$$L.H.S = \langle cf, g\rangle = \frac{1}{2\pi}\int_0^{2\pi} (cf)(t)\ \overline{g(t)}\ dt$$

$$= c \times \frac{1}{2\pi}\int_0^{2\pi} f(t)\ \overline{g(t)}\ dt$$

$$= c\ \langle f, g\rangle$$

$$= R.H.S$$

(iii) T.P: $\langle f, g\rangle = \overline{\langle g, f\rangle}$

$$L.H.S = \langle f, g\rangle = \frac{1}{2\pi}\int_0^{2\pi} \overline{f(t)\ g(t)}\ dt$$

$$= \frac{1}{2\pi}\int_0^{2\pi} \overline{f(t)}\ g(t)\ dt$$

$$= \frac{1}{2\pi}\int_0^{2\pi} g(t)\cdot \overline{f(t)}\ dt$$

$$= \langle g, f\rangle$$

$$= R.H.S$$

(iv) T.P: $\langle f, f\rangle \geq 0$

$$\langle f, f\rangle = \frac{1}{2\pi}\int_0^{2\pi} f(t)\ \overline{f(t)}\ dt$$

$$= \frac{1}{2\pi}\int_0^{2\pi} |f(t)|^2\ dt$$

$$\geq 0$$

Hence $\langle f, g\rangle$ is an inner product &

$\therefore$ H is an inner product space.

Norm of a vector $\| \ \|$

Def: Let V be an inner product space. For $x \in V$, we define the norm (or) length of x by $\|x\| = \sqrt{\langle x, x\rangle}$

Result:

we define a norm $\|\cdot\|$ as a real-valued function on V satisfying the below conditions

$\forall\ x, y \in V$ & $a \in F$.

(i) $\|x\| \geq 0$ and $\|x\| = 0$ iff $x = 0$

(ii) $\|ax\| = |a| \cdot \|x\|$

(iii) $\|x+y\| \leq \|x\| + \|y\|$

(1) Let $x = (2, 1+i, i)$ and $y = (2-i, 2, 1+2i)$ be vectors in e^3. Compute $\|x\|$, $\|y\|$ & $\|x+y\|$

Sol:

$$\|x\| = \sqrt{\langle x, x\rangle} = \sqrt{2^2 + (\sqrt{1^2+i^2})^2 + (\sqrt{0^2+1^2})^2}$$

$$= \sqrt{4+2+1} = \sqrt{7}$$

$$\|y\| = \sqrt{\langle y, y\rangle} = \sqrt{(\sqrt{(2^2+(-1)^2})^2 + 2^2 + (\sqrt{1^2+2^2})^2}$$

$$= \sqrt{5+4+5} = \sqrt{14}$$

$$\|x+y\| = \sqrt{\langle x+y, x+y\rangle}$$

$$= \sqrt{(4-i)(4+i) + (3+i)(3-i) + (1+3i)(1-3i)}$$

$$= \sqrt{17+10+10} = \sqrt{37}$$

Cauchy-Schwarz inequality – Triangle inequality

$$|\langle x, y\rangle| \leq \|x\| \cdot \|y\|$$

Note: Triangle inequality $\|x+y\| \leq \|x\| + \|y\|$

(1.7)

Let $x = (2, 1+i, i)$ and $y = (2-i, 2, 1+2i)$ be vectors in C^3 verify Cauchy-schwarz and triangle inequality.

Sol:

(i) $|\langle f, g\rangle| \leq \|f\| \cdot \|g\|$

ie, $|\langle x, y\rangle| \leq \|x\| \cdot \|y\|$

$\langle x, y\rangle = 2\overline{(2-i)} + (1+i)2 + i\overline{(1+2i)}$

$= 2(2+i) + (1+i)2 + i(1-2i)$

$= 8+5i$

$\therefore |\langle x, y\rangle| = \sqrt{8^2+5^2}$

$= \sqrt{64+25}$

$= 9.43$

$\|x\| = \sqrt{\langle x, x\rangle} = \sqrt{(2)(2)+2+1} = \sqrt{7}$

$\|y\| = \sqrt{\langle y, y\rangle} = \sqrt{\left(\sqrt{2^2+1^2}\right)^2 + 2^2 + \left(\sqrt{1^2+2^2}\right)^2}$

$= \sqrt{5+4+5} = \sqrt{14}.$

$\|x\| \cdot \|y\| = 9.899$

$\therefore |\langle x, y\rangle| = \|x\| \cdot \|y\|$

(ii) $\|x+y\| \leq \|x\| + \|y\|$

L.H.S $= \sqrt{\langle x+y, x+y\rangle}$

$= \sqrt{(4-i)(4+i) + (3+i)(3-i) + (1+3i)(1-3i)}$

$= \sqrt{17+10+10} = 6.082$

R.H.S $= \|x\| + \|y\| = \sqrt{7} + \sqrt{14} = 6.386$

$\therefore \|x+y\| \leq \|x\| + \|y\|$

(2) Let V be an inner product space P.T

$\|x \pm y\|^2 = \|x\|^2 \pm 2 \cdot \text{Re} \langle x, y \rangle + \|y\|^2 \quad \forall\ x, y \in V$

where $\text{Re}\langle x, y \rangle$ denotes the real part of the complex number $\langle x, y \rangle$.

Sol:

$\forall\ x, y \in V \quad \|x+y\|^2 = \langle x+y, x+y \rangle$

$= \langle x, x+y \rangle + \langle y, x+y \rangle$

$= \langle x, x \rangle + \langle x, y \rangle + \langle y, x \rangle + \langle y, y \rangle$

$= \|x\|^2 + \langle x, y \rangle + \overline{\langle x, y \rangle} + \|y\|^2$

$= \|x\|^2 + 2 \cdot \text{Re} \langle x, y \rangle + \|y\|^2 \quad \text{---(1)}$

W.K.T $\|x-y\|^2 = \|x\|^2 - 2\,\text{Re}\langle x, y \rangle + \|y\|^2 \quad \text{---(2)}$

From (1) & (2), $\|x+y\|^2 = \|x^2 \pm 2\,\text{Re}\langle x, y \rangle + \|y\|^2$

---

(3) Let V be an inner product space. P.T

$\big|\, \|x\| - \|y\| \,\big| \leq \|x-y\| \quad \forall\ x, y \in V$

Sol:

Using Δ inequality,

$\|x\| - \|y\| = \|x - y + y\| - \|y\|$

$\leq \|x-y\| + \|y\| - \|y\|$

$\leq \|x-y\|$

$\Rightarrow \big|\, \|x\| - \|y\| \,\big| \leq \|x-y\| \quad \forall\ x, y \in V$

Vectors x and y in V are orthogonal (perpendicular)

Def: Let V be an inner product space. Vectors x & y in V are orthogonal if

$\langle x, y \rangle = 0$

Def:

A vector x in V is a unit vector if $\|x\| = 1$

Eigenvalues using QR transformation

Def: Let T be a linear operator on a vector space V. A non-zero vector $v \in V$ is called an Eigenvector of T if $\exists$ a scalar λ s.t $T(v) = \lambda v$. The scalar λ is called the Eigenvalue corresponding to the Eigenvector v.

(1) Prove that if a real matrix has one Eigenvector then it has an infinite no of Eigenvalues.

Proof:

Let λ be the Eigenvalue of the real matrix A and v be the Eigenvector of A corresponding to λ.

Then $(A - \lambda I) \cdot v = 0$ Let $c \in R$ be any constant

Let $(A - \lambda I)(cv) = c[(A - \lambda I)v] = c(0) = 0$

Hence cv is an Eigenvector of A, for any $c \in R$

$\therefore$ The real matrix A has infinitely many Eigenvectors.

Note: The QR iteration is an eigenvalue algorithm, i.e, a procedure to calculate the eigenvalues and eigenvectors of a matrix.

Use of QR factorization

The QR matrix decomposition allows one to express a matrix as a product of 2 seperate matrices Q and R. Q is an orthogonal matrix and R is a square ~~matrix~~ upper or right triangular matrix.

Q.R Method

The matrix A is described as $A = QR$.

where Q is orthogonal and R is upper triangular matrix.

Find Q and R for the QR factorization of

$$A = \begin{bmatrix} 1 & 2 & 3 \\ -1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Sol:

we want $A = QR$

Columns of A → orthonormal set

Q → orthogonal matrix (Orthonormal Columns)

R → upper triangular matrix

$$X = \left\{ \begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 5 \\ 7 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 8 \\ -4 \\ -6 \\ -6 \end{bmatrix} \right\} \text{ (orthogonal set)}$$

$$\begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \end{bmatrix} = \sqrt{1^2+(-1)^2+1^2+1^2} = 2$$

$\div 2$

$$\begin{bmatrix} 4 \\ -2 \\ -3 \\ -3 \end{bmatrix}$$

orthonormal set :

$$\left\{ \begin{bmatrix} 1/2 \\ -1/2 \\ 1/2 \\ 1/2 \end{bmatrix}, \begin{bmatrix} 5/\sqrt{76} \\ 7/\sqrt{76} \\ 1/\sqrt{76} \\ 1/\sqrt{76} \end{bmatrix}, \begin{bmatrix} 4/\sqrt{38} \\ -2/\sqrt{38} \\ -3/\sqrt{38} \\ -3/\sqrt{38} \end{bmatrix} \right\}$$

$$\therefore Q = \begin{bmatrix} 1/2 & 5/\sqrt{76} & 4/\sqrt{38} \\ -1/2 & 7/\sqrt{76} & -2/\sqrt{38} \\ 1/2 & 1/\sqrt{76} & -3/\sqrt{38} \\ 1/2 & 1/\sqrt{76} & -3/\sqrt{38} \end{bmatrix}$$

To find R:

Now $A = QR$

$$Q^T A = Q^T Q R$$

$$= IR$$

$$= R$$

$$\therefore R = \begin{bmatrix} 1/2 & -1/2 & 1/2 & 1/2 \\ 5/\sqrt{76} & 7/\sqrt{76} & 1/\sqrt{76} & 1/\sqrt{76} \\ 4/\sqrt{38} & -2/\sqrt{38} & -3/\sqrt{38} & -3/\sqrt{38} \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ -1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$\therefore R = \begin{bmatrix} 4/2 & 3/2 & 4/2 \\ 0 & 19/\sqrt{76} & 24/\sqrt{76} \\ 0 & 0 & 4/\sqrt{38} \end{bmatrix} = \begin{bmatrix} 2 & 3/2 & 2 \\ 0 & \sqrt{19}/2 & \frac{12\sqrt{19}}{19} \\ 0 & 0 & \frac{2\sqrt{38}}{19} \end{bmatrix}$$

$A = \begin{bmatrix} 3 & 2 \\ 1 & 4 \end{bmatrix}$ find the eigenvalues of the matrix correct to one decimal place

Sol:

$$A = \begin{bmatrix} 3 & 2 \\ 1 & 4 \end{bmatrix}$$

$$\tan\alpha = \frac{-a_{21}}{a_{11}} = -1/3$$

If $\tan\alpha < 0$ then $c = -1$

$$b = \sqrt{a_{11}^2 + a_{21}^2} = \sqrt{1^2 + 3^2} = 3.1623$$

$$\sin\alpha = c.\frac{|a_{21}|}{b} = \frac{-1.1}{3.1623} = -0.3162$$

$$\cos\alpha = \frac{|a_{11}|}{b} = \frac{3}{3.1623} = 0.9487$$

$$= a_{11}\cos\alpha - a_{21}\sin\alpha$$

$$= (3 \times 0.9487) - 1(-0.3162)$$

$$f_{11} = 3.1623$$

$$f_{12} = a_{12}\cos\alpha - a_{22}\sin\alpha = 3.1622$$

$$f_{21} = 0$$

$$f_{22} = a_{12}\sin\alpha + a_{22}\cos\alpha = 3.1622$$

$$\therefore Q_1 = \begin{bmatrix} 0.9487 & -0.3162 \\ 0.3162 & 0.9487 \end{bmatrix}$$

$$R_1 = \begin{bmatrix} 3.1623 & 3.1622 \\ 0 & 3.1624 \end{bmatrix}$$

Now $A_1 = R_1 Q_1 = \begin{bmatrix} 3 & 2 \\ 0.9 & 4 \end{bmatrix}$

$$A_2 = R_1 Q_1 = \begin{bmatrix} 4 & 2 \\ 1 & 3 \end{bmatrix}$$

$$\tan\alpha = -\frac{1}{4} = -0.2495, \quad \sin\alpha = \frac{-1}{\sqrt{17}}, \quad \cos\alpha = \frac{4}{\sqrt{17}} = 0.97$$

$$f_{11} = 4 \times 0.97 - 1 \times (-0.2495) = 4.1225$$

$$f_{12} = 2.6675, \quad f_{21} = 0, \quad f_{22} = 2.4250$$

$$A_3 = R_2 Q_2 = \begin{bmatrix} 4.1225 & 2.6675 \\ 0 & 2.4250 \end{bmatrix} \begin{bmatrix} 0.97 & -0.2425 \\ 0.2 & 0.97 \end{bmatrix}$$

$$= \begin{bmatrix} 4.6457 & 1.5978 \\ 0.5221 & 2.3522 \end{bmatrix}$$

Now $\tan\alpha = -\frac{0.521}{4.6457} =$

$\sin\alpha = -0.1256$, $\cos\alpha = 0.9921$

$f_{11} = 4.6229$, $f_{12} = 1.2707$, $f_{21} = 0$, $f_{22} = 2.1342$

$$A_4 = R_3 Q_3 = \begin{bmatrix} 4.6229 & 1.2707 \\ 0 & 2.1342 \end{bmatrix} \begin{bmatrix} 0.9921 & -0.1256 \\ 0.1256 & 0.9921 \end{bmatrix}$$

$$A_4 = \begin{bmatrix} 4.8808 & 1.2677 \\ 0.2681 & 2.1173 \end{bmatrix}$$

$\tan\alpha = -\frac{0.2621}{4.8808} =$ $\sin\alpha =$, $\cos\alpha =$

$f_{11} = 4.8808 \times 0.9985 - 0.2621 \times (0.084)$

$= 4.8882$, $f_{12} = 1.3214$, $f_{21} = 0$, $f_{22} = 2.0446$

$$A_5 = R_4 Q_4 = \begin{bmatrix} 4.8802 & 1.3814 \\ 0 & 2.0996 \end{bmatrix} \begin{bmatrix} 0.9905 & -0.05 \\ 0.054 & 0.9985 \end{bmatrix}$$

$$A_5 = \begin{bmatrix} 4.9566 & 1.0938 \\ 0.1120 & 2.0415 \end{bmatrix}$$

$\tan\alpha = -\frac{0.1120}{4.9566}$, $\sin\alpha = -0.0226$, $\cos\alpha = 0.9297$

$f_{11} = 4.9576$, $f_{12} = 1.1896$, $f_{21} = 0$, $f_{22} = 2.0166$

$$A_6 = \begin{bmatrix} 4.9819 & 1.0274 \\ 0.0456 & 2.0156 \end{bmatrix}$$

Now Comparing the diagonal element of A_5 & A_6

$|4.9566 - 4.9819| = 0.0253$

$|0.0156 - 2.0415| = 0.0269$

which is less than 0.05 → 4.9566

The eigen values of A are 5 & 2 → 2.0415

Singular value decomposition and applications

The SVD of a matrix is a factorization of that matrix into three matrices.

Formula: $A = U \Sigma V^T$ (A → input matrix, U → left singular vectors, Σ → diagonal (eigenvalues), V → right singular vectors)

(1) Find the singular value decomposition of the matrix $C = \begin{pmatrix} 5 & 5 \\ -1 & 7 \end{pmatrix}$.

Sol:

we want: $C = U \Sigma V^T$

* $C^T C = V \Sigma^T \Sigma V^T$
* $CV = U\Sigma$

$$C^T C = \begin{pmatrix} 5 & -1 \\ 5 & 7 \end{pmatrix}\begin{pmatrix} 5 & 5 \\ -1 & 7 \end{pmatrix} = \begin{pmatrix} 26 & 18 \\ 18 & 74 \end{pmatrix}$$

$$\det(C^T C - \lambda I) = \det\begin{pmatrix} 26-\lambda & 18 \\ 18 & 74-\lambda \end{pmatrix}$$

$$= \lambda^2 - 100\lambda + 1600$$

$$= (\lambda - 20)(\lambda - 80)$$

$$C^T C - 20I = \begin{pmatrix} 6 & 18 \\ 18 & 54 \end{pmatrix}, \quad v_1 = \begin{pmatrix} -3/\sqrt{10} \\ 1/\sqrt{10} \end{pmatrix}$$

$$C^T C - 80I = \begin{pmatrix} -54 & 18 \\ 18 & -6 \end{pmatrix}, \quad v_2 = \begin{pmatrix} 1/\sqrt{10} \\ 3/\sqrt{10} \end{pmatrix}$$

$$V = \begin{pmatrix} -3/\sqrt{10} & 1/\sqrt{10} \\ 1/\sqrt{10} & 3/\sqrt{10} \end{pmatrix}, \quad \Sigma = \begin{pmatrix} 2\sqrt{5} & 0 \\ 0 & 4\sqrt{5} \end{pmatrix}$$

$$\begin{pmatrix} 5 & 5 \\ -1 & 7 \end{pmatrix}\begin{pmatrix} -3/\sqrt{10} & 1/\sqrt{10} \\ 1/\sqrt{10} & 3/\sqrt{10} \end{pmatrix} = \begin{pmatrix} -\sqrt{10} & 2\sqrt{10} \\ 2\sqrt{10} & 2\sqrt{10} \end{pmatrix}$$

$$= \begin{pmatrix} -1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix}\begin{pmatrix} 2\sqrt{5} & \\ & 4\sqrt{5} \end{pmatrix}$$

$$U = \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix}$$

∴ $C = U \Sigma V^T$ =

Find all the Eigen values & Eigenvectors, SVD of the

matrix $A = \begin{bmatrix} 4 & 0 \\ 3 & -5 \end{bmatrix}$

Sol:

Step 1: $A^TA = \begin{bmatrix} 4 & 3 \\ 0 & -5 \end{bmatrix} \begin{bmatrix} 4 & 0 \\ 3 & -5 \end{bmatrix}$

$= \begin{bmatrix} 25 & -15 \\ -15 & 25 \end{bmatrix}$

Step 2: $A^TA - \lambda I = \begin{bmatrix} 25-\lambda & -15 \\ -15 & 25-\lambda \end{bmatrix}$

$\det(A^TA - \lambda I) = ((25-\lambda)(25-\lambda)) - (-15 \times -15)$

$= \lambda^2 - 50\lambda + 400$

Solving $\lambda^2 - 50\lambda + 400 = 0$

$\lambda^2 - 40\lambda - 10\lambda + 400 = 0$

$(\lambda - 10)(\lambda - 40) = 0$

∴ Two eigen values are 10, 40

Step 3: ~~For 10~~.

$A^TA - 10I = \begin{bmatrix} 25-10 & -15 \\ -15 & 25-10 \end{bmatrix} = \begin{bmatrix} 15 & -15 \\ -15 & 15 \end{bmatrix}$

$\begin{bmatrix} 15 & -15 \\ -15 & 15 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = 0$

Row 1 + Row 2 $\begin{bmatrix} 15 & -15 \\ 0 & 0 \end{bmatrix}$

Row 1 $\times \frac{1}{15}$ $\begin{bmatrix} 1 & -1 \\ 0 & 0 \end{bmatrix}$

$$\begin{bmatrix} 1 & -1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

solve: $1v_1 - 1v_2 = 0$

$1v_1 = v_2$ for $v_2 = 1$

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\sqrt{1^2 + 1^2} = \sqrt{2}$$

convert to unit vectors

$$\begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} = \text{Eigenvectors of Eigenvalue 10}$$

For 40:

$$A^TA - 40I = \begin{bmatrix} 25-40 & -15 \\ -15 & 25-40 \end{bmatrix} = \begin{bmatrix} -15 & -15 \\ -15 & -15 \end{bmatrix}$$

$$= \begin{bmatrix} -15 & -15 \\ 0 & 0 \end{bmatrix} \quad R_1 + (-1)R_2$$

$$\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \quad -1/15\, R_1$$

$$\therefore \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$1v_1 + 1v_2 = 0$$

$$v_1 = -v_2$$

$$\begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} = \text{Eigenvectors of Eigenvalue 40.}$$

$\therefore$ Eigenvectors & Eigenvalues are.

$$\begin{bmatrix} -1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} = V$$

$$\begin{bmatrix} \sqrt{40} & 0 \\ 0 & \sqrt{10} \end{bmatrix} = \Sigma$$

Step 4:

$$A = U \Sigma V^T$$

$$\Rightarrow AV\Sigma^T = U$$

$$AV = \begin{bmatrix} 4 & 0 \\ 3 & -5 \end{bmatrix} \begin{bmatrix} -1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$$

$$= \begin{bmatrix} -4/\sqrt{2} & 4/\sqrt{2} \\ -8/\sqrt{2} & -5/\sqrt{2} \end{bmatrix}$$

$$= \begin{bmatrix} -4/\sqrt{2} \times \frac{1}{3\sqrt{10}} & \frac{4}{\sqrt{2}} \times \frac{1}{\sqrt{10}} \\ \frac{-8}{\sqrt{2}} \times \frac{1}{3\sqrt{10}} & \frac{-8}{\sqrt{2}} \times \frac{1}{\sqrt{10}} \end{bmatrix}$$

$$= \begin{bmatrix} -1/\sqrt{5} & 2/\sqrt{5} \\ -2/\sqrt{5} & -1/\sqrt{5} \end{bmatrix}$$

$$\therefore AV\Sigma^T = \begin{bmatrix} -1/\sqrt{5} & 2/\sqrt{5} \\ -2/\sqrt{5} & -1/\sqrt{5} \end{bmatrix} \begin{bmatrix} \sqrt{40} & 0 \\ 0 & \sqrt{10} \end{bmatrix}$$

$$= \begin{bmatrix} -2\sqrt{2} & 2\sqrt{2} \\ -4\sqrt{2} & -\sqrt{2} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{-2\sqrt{2}}{\sqrt{40}} & \frac{2\sqrt{2}}{\sqrt{10}} \\ \frac{-4\sqrt{2}}{\sqrt{40}} & \frac{-\sqrt{2}}{\sqrt{10}} \end{bmatrix}$$

$$= \begin{bmatrix} -2/\sqrt{20} & 2/\sqrt{5} \\ -4/\sqrt{20} & -1/\sqrt{5} \end{bmatrix}$$

$$\therefore U = \begin{bmatrix} -1/\sqrt{5} & 2/\sqrt{5} \\ -2/\sqrt{5} & -1/\sqrt{5} \end{bmatrix}$$

Verification $A = U \Sigma V^T$

$$\begin{bmatrix} 4 & 0 \\ 3 & -5 \end{bmatrix} = \begin{bmatrix} -1/\sqrt{5} & 2/\sqrt{5} \\ -2/\sqrt{5} & -1/\sqrt{5} \end{bmatrix} \begin{bmatrix} \sqrt{40} & 0 \\ 0 & \sqrt{10} \end{bmatrix} \begin{bmatrix} -1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$$

Pseudo inverse (A^+)

The pseudo inverse is defined and unique for all matrices whose entries are real or complex numbers. It can be computed using the singular value decomposition.

Problems:

(1) $A = \begin{pmatrix} 1 & 2 \end{pmatrix}$

using SVD,
$A = U \Sigma V^T$

8. Open cloud architectures like Bluemix, Development platforms like Firebase

COURSE OUTCOMES:

On completion of the course, the students will be able to:

CO1: To understand the various IoT protocols
CO2: Test and experiment different sensors for application development
CO3: To develop applications using Arduino/Raspberry Pi/ Equivalent boards.
CO4: To develop applications that would read the sensor data and post it in Cloud
CO5: Develop IOT applications with different platforms and frameworks.

CO-PO Mapping

CO	POs					
	PO1	PO2	PO3	PO4	PO5	PO6
1	2	1	2	2	2	2
2	2	1	2	2	2	2
3	2	1	2	2	2	2
4	2	1	2	2	2	2
5	2	1	2	2	2	2
Avg	2	1	2	2	2	2

MC4001 SOFTWARE PROJECT MANAGEMENT L T P C
3 0 0 3

COURSE OBJECTIVES:

- To know how to do project planning for the software process.
- To learn the cost estimation techniques during the analysis of the project.
- To understand the quality concepts for ensuring the functionality of the software

UNIT SOFTWARE PROJECT MANAGEMENT CONCEPTS 9

Introduction to Software Project Management: An Overview of Project Planning: Select Project, Identifying Project scope and objectives, infrastructure, project products and Characteristics. Estimate efforts, Identify activity risks, and allocate resources- Six Sigma, Software Quality: defining software quality, ISO9126, External Standards.

UNIT II SOFTWARE EVALUATION AND COSTING 9

Project Evaluation: Strategic Assessment, Technical Assessment, cost-benefit analysis, Cash flow forecasting, cost-benefit evaluation techniques, Risk Evaluation. Selection of Appropriate Project approach: Choosing technologies, choice of process models, structured methods.

UNIT III SOFTWARE ESTIMATION TECHNIQUES 9

Software Effort Estimation: Problems with over and under estimations, Basis of software Estimation, Software estimation techniques, expert Judgment, Estimating by analogy. Activity Planning: Project schedules, projects and activities, sequencing and scheduling Activities, networks planning models, formulating a network model.

UNIT IV RISK MANAGEMENT 9

Risk Management: Nature of Risk, Managing Risk, Risk Identification and Analysis, Reducing the Risk. Resource Allocation: Scheduling resources, Critical Paths, Cost scheduling, Monitoring and Control: Creating Framework, cost monitoring, prioritizing monitoring.

UNIT V GLOBALIZATION ISSUES IN PROJECT MANAGEMENT 9

Globalization issues in project management: Evolution of globalization- challenges in building global teams-models for the execution of some effective management techniques for managing global teams. Impact of the internet on project management: Introduction – the effect of the internet on project management – managing projects for the internet – effect on project management activities. Comparison of project management software: dot Project, Launch pad, openProj. Case study: PRINCE2

SUGGESTED ACTIVITIES:

1. Reducing process variability using six-sigma model DMAIC on software company applications with respect to factors like quality aspects , production bugs classified and measured, the causes of the large number of production bugs leading to different improvement suggestions
2. Do cost benefit analysis using Ms-Excel for Selecting the project (from available data in the web like https://img.chandoo.org/a/24-cost-benefit-analysis.xlsx)
3. Frequencying and Scheduling the Project activities using open source Ms-Project
4. Risk analysis of any project with special reference to performance time cost trilogy
5. Set up a project and its tasks ; Communicate with everyone on the project team from within dotProject software.

TOTAL:45 PERIODS

COURSE OUTCOMES:

CO1: Understand the activities during the project scheduling of any software application.

CO2: Learn the risk management activities and the resource allocation for the projects.

CO3: Apply the software estimation and recent quality standards for evaluation of the software projects

CO4: Acquire knowledge and skills needed for the construction of highly reliable software project

CO5: Create reliable, replicable cost estimation that links to the requirements of project planning and managing

REFERENCES

1. Bob Hughes, Mike Cotterell & Rajib Mall “Software Project Management”, McGraw- Hill Publications, 6th Edition 2017.
2. Ian Somerville, “Software Engineering”, 10th Edition, Pearson Education, 2017.
3. Robert T. Futrell , “Quality Software Project Management”, Pearson Education India, 2008.
4. Gopalaswamy Ramesh, “Managing Global Software Projects: How to Lead Geographically Distributed Teams, Manage Processes and Use Quality Models”, McGraw Hill Education, 2017.
5. Richard H.Thayer “Software Engineering Project Management”, 2nd Edition, Wiley, 2006.
6. S. A. Kelkar,” Software Project Management” PHI, New Delhi, Third Edition ,2013

PROFESSIONAL ELECTIVES

MC4001 SOFTWARE PROJECT MANAGEMENT L T P C

3 0 0 3

UNIT I SOFTWARE PROJECT MANAGEMENT CONCEPTS

Introduction to Software Project Management: An Overview of Project Planning: Select Project, Identifying Project scope and objectives, infrastructure, project products and Characteristics. Estimate efforts, Identify activity risks, and allocate resources- Six Sigma, Software Quality: defining software quality, ISO9126, External Standards.

Introduction to Software Project Management:

Project management has been practiced since early civilization. Until the beginning of twentieth century civil engineering projects were actually treated as projects and were generally managed by creative architects and engineers. Project management as a discipline was not accepted. It was in the 1950s that organizations started to systematically apply project management tools and techniques to complex projects. As a discipline, Project Management developed from several fields of application including construction, engineering, and defense activity. Two forefathers of project management are commonly known: Henry Gantt, called the father of planning and control techniques who is famous for his use of the Gantt chart as a project management tool; and Henri Fayol for his creation of the five management functions which form the foundation of the body of knowledge associated with project and program management. The 1950s marked the beginning of the modern Project Management era. Project management became recognized as a distinct discipline arising from the management discipline.

WHAT IS A PROJECT

All of us have been involved in projects, whether they be our personal projects or in business and industry. Examples of typical projects are for example:

- **Personal projects:**
  - obtaining an MCA degree
  - writing a report
  - planning a party
  - planting a garden

- **Industrial projects:**
- Construction of a building
- provide electricity to an industrial estate
- building a bridge
- designing a new airplane

Projects can be of any size and duration. They can be simple, like planning a party, or complex like launching a space shuttle.

1.1.1 Project Definition:

A project can be defined in many ways :

A **project** is ―a temporary endeavor undertaken to create a unique product, service, or result.‖ Operations, on the other hand, is work done in organizations to sustain the business. Projects are different from operations in that they end when their objectives have been reached or the project has been terminated.

A project is temporary. A project's duration might be just one week or it might go on for years, but every project has an end date. You might not know that end date when the project begins, but it's there somewhere in the future. Projects are not the same as ongoing operations, although the two have a great deal in common.

A project is an endeavor. Resources, such as people and equipment, need to do work. The endeavor is undertaken by a team or an organization, and therefore projects have a sense of being intentional, planned events. Successful projects do not happen spontaneously; some amount of preparation and planning happens first.

Finally, every project creates a unique product or service. This is the **deliverable** for the project and the reason, why that project was undertaken.

Planning & Design:

After the initiation stage, the system is designed. Occasionally, a small prototype of the final product is built and tested. Testing is generally performed by a combination of testers and end users, and can occur after the prototype is built or concurrently. Controls should be in place that ensures that the final product will meet the specifications of the project charter. The results of the design stage should include a product design that:

- Satisfies the project sponsor (the person who is providing the project budget), end user, and business requirements.

-Functions as it was intended.

- Can be produced within acceptable quality standards.

- Can be produced within time and budget constraints.

Select Project:

Projects come in all shapes and sizes. The following attributes help us to define a project further:

A project has a unique purpose. Every project should have a well-defined objective. For example, many people hire firms to design and build a new house, but each house, like each person, is unique.

A project is temporary. A project has a definite beginning and a definite end. For a home construction project, owners usually have a date in mind when they'd like to move into their new homes.

A project is developed using progressive elaboration or in an iterative fashion. Projects are often defined broadly when they begin, and as time passes, the specific details of the project become clearer. For example, there are many decisions that must be made in planning and building a new house. It works best to draft preliminary plans for owners to approve before more detailed plans are developed.

A project requires resources, often from various areas. Resources include people, hardware, software, or other assets. Many different types of people, skill sets, and resources are needed to build a home.

A project should have a primary customer or sponsor. Most projects have many interested parties or stakeholders, but someone must take the primary role of sponsorship. The project sponsor usually provides the direction and funding for the project.

A project involves uncertainty. Because every project is unique, it is sometimes difficult to define the project's objectives clearly, estimate exactly how long it will take to complete, or determine how much it will cost. External factors also cause uncertainty, such as a supplier going out of business or a project team member needing unplanned time off. This uncertainty is one of the main reasons project management is so challenging.

Identifying Project scope and objectives:

Like any human undertaking, projects need to be performed and delivered under certain constraints. Traditionally, these constraints have been listed as scope, time, and cost. These are also referred to as the Project Management Triangle, where each side represents a constraint. One side of the triangle cannot be changed without impacting the others. A further refinement of the constraints separates product 'quality' or 'performance' from scope, and turns quality into a fourth constraint.

The time constraint refers to the amount of time available to complete a project. The cost constraint refers to the budgeted amount available for the project. The scope constraint refers to what must be done to produce the project's end result. These three constraints are often competing constraints: increased scope typically means increased time and increased cost, a tight time constraint could mean increased costs and reduced scope, and a tight budget could mean increased time and reduced scope.

The discipline of project management is about providing the tools and techniques that enable the project team (not just the project manager) to organize their work to meet these constraints.

Another approach to project management is to consider the three constraints as finance, time and human resources. If you need to finish a job in a shorter time, you can allocate more people at the problem, which in turn will raise the cost of the project, unless by doing this task quicker we will reduce costs elsewhere in the project by an equal amount.

Time:

For analytical purposes, the time required to produce a product or service is estimated using several techniques. One method is to identify tasks needed to produce the deliverables documented in a work breakdown structure or WBS. The work effort for each task is estimated and those estimates are rolled up into the final deliverable estimate

The tasks are also prioritized, dependencies between tasks are identified, and this information is documented in a project schedule. The dependencies between the tasks can affect the length of the overall project (dependency constraint), as can the availability of resources (resource constraint). Time is not considered a cost nor a resource since the project manager cannot control the rate at which it is expended. This makes it different from all other resources and cost categories.

Cost:

Cost to develop a project depends on several variables including : labor rates, material rates, risk management, plant (buildings, machines, etc.), equipment, and profit. When hiring an independent consultant for a project, cost will typically be determined by the consultant's or firm's per diem rate multiplied by an estimated quantity for completion.

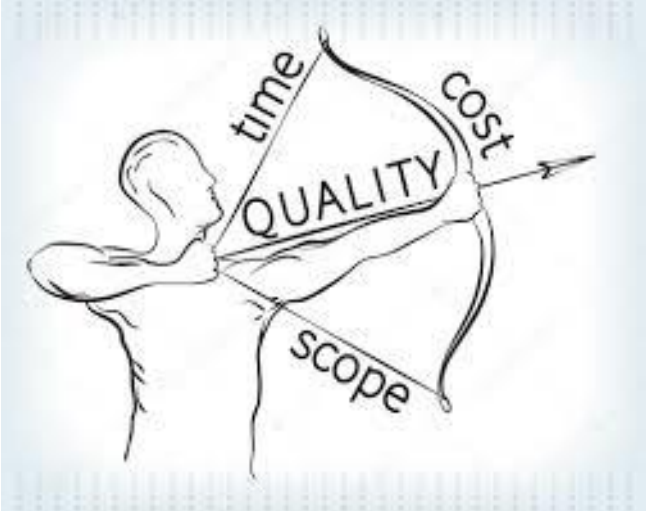


Figure 1.1 : The Project management Triangle

Scope:

Scope is requirement specified for the end result. The overall definition of what the project is supposed to accomplish, and a specific description of what the end result should be or accomplish can be said to be the scope of the project. A major component of scope is the quality of the final product. The amount of time put into individual tasks determines the overall quality of the project. Some tasks may require a given amount of time to complete adequately, but given more time could be completed exceptionally. Over the course of a large project, quality can have a significant impact on time and cost or vice versa.

Together, these three constraints viz. Scope, Schedule & Resources have given rise to the phrase "On Time, On Spec, On Budget". In this case, the term "scope" is substituted with "spec(ification)"

Infrastructure:

Project management is “the application of knowledge, skills, tools and techniques to project activities to meet the project requirements”. The effectiveness of project management is critical in assuring the success of any substantial activity. Areas of responsibility for the person handling the project include planning, control and implementation. A project should be initiated with a feasibility study, where a clear definition of the goals and ultimate benefits need to be determined. Senior managers' support for projects is important so as to ensure authority and direction throughout the project's progress and, also to ensure that the goals of the organization are effectively achieved in this process.

Knowledge, skills, goals and personalities are the factors that need to be considered within project management. The project manager and his/her team should collectively possess the

necessary and requisite interpersonal and technical skills to facilitate control over the various activities within the project.

The stages of implementation must be articulated at the project planning phase. Disaggregating the stages at its early point assists in the successful development of the project by providing a number of milestones that need to be accomplished for completion. In addition to planning, the control of the evolving project is also a prerequisite for its success. Control requires adequate monitoring and feedback mechanisms by which senior management and project managers can compare progress against initial projections at each stage of the project. Monitoring and feedback also enables the project manager to anticipate problems and therefore take pre□emptive and corrective measures for the benefit of the project.

Projects normally involve the introduction of a new system of some kind and, in almost all cases, new methods and ways of doing things. This impacts the work of others: the "users". User interaction is an important factor in the success of projects and, indeed, the degree of user involvement can influence the extent of support for the project or its implementation plan. A project manager is the one who is responsible for establishing a communication in between the project team and the user. Thus one of the most essential quality of the project manager is that of being a good communicator, not just within the project team itself, but with the rest of the organization and outside world as well.

Project products and Characteristics:

Features of projects:

- Projects are often carried out by a team of people who have been assembled for that specific purpose. The activities of this team may be co-ordinated by a project manager.

- Project teams may consist of people from different backgrounds and different parts of the organisation. In some cases project teams may consist of people from different organisations.

- Project teams may be inter-disciplinary groups and are likely to lie outside the normal organisation hierarchies.

- The project team will be responsible for delivery of the project end product to some sponsor within or outside the organisation. The full benefit of any project will not become available until the project as been completed.

project products and Characteristics.:

In recent years more and more activities have been tackled on a project basis. Project teams and a project management approach have become common in most organisations. The basic

approaches to project management remain the same regardless of the type of project being considered. You may find it useful to consider projects in relation to a number of major classifications:

a) Engineering and construction

The projects are concerned with producing a clear physical output, such as roads, bridges or buildings. The requirements of a project team are well defined in terms of skills and background, as are the main procedures that have to be undergone. Most of the problems which may confront the project team are likely to have occurred before and therefore their solution may be based upon past experiences.

b) Introduction of new systems

These projects would include computerisation projects and the introduction of new systems and procedures including financial systems. The nature and constitution of a project team may vary with the subject of the project, as different skills may be required and different end-users may be involved. Major projects involving a systems analysis approach may incorporate clearly defined procedures within an organisation.

c) Responding to deadlines and change

An example of responding to a deadline is the preparation of an annual report by a specified date. An increasing number of projects are concerned with designing organisational or environmental changes, involving developing new products and services.

What is Effort Estimation?

Effort estimation is the process of forecasting how much effort is required to develop or maintain a software application. This effort is traditionally measured in the hours worked by a person, or the money needed to pay for this work.

Effort estimation is used to help draft project plans and budgets in the early stages of the software development life cycle. This practice enables a project manager or product owner to accurately predict costs and allocate resources accordingly.

Identify activity risks:

Step 1; Identify and quantify activity-based risks

For brevity, we'll focus on the initial three steps as they cover risk identification specifically (while the remaining steps are about validating and formalizing findings against the overall project's scope).

Template specification

This is a risk statement based on feedback about causes, effects, impacts, areas of risk, and events. A structured template helps you capture this in a consistent way.

Basic identification

Answering two questions about potential risks: why or why not us and whether they have been experienced before. The former can be captured via SWOT analysis exercise while the latter is a statement, ideally referenced from a project post mortem or lessons learned library.

Detailed identification

This step is more time-consuming than the previous ones but also delivers the detail you need to properly assess risk. PMI identifies five tools to use:

- Interviewing
- Assumptions analysis
- Document reviews
- Delphi technique
- Brainstorming

Once you've completed these steps you'll need to categorize risk in the next one — the External cross-check step. We've covered this in our Understanding Risk Breakdown Structure article.

Step five is the **Internal Cross-check** which maps risks to corresponding elements in the scope of work. At this point you will start forming a view of what project elements are riskier than others, and what mitigation strategies to adopt.

The final step, **Statement Finalization**, packages findings in a series of diagrams covering *risky areas*, *causes,* and *impacts*.

Tip: Use a tool like Wrike to maintain a risk register spanning all of your projects which you can refer to whenever you start a new one.

Resource Allocation in Project:

Resource allocation is the process of assigning and scheduling available resources in the most effective and economical way possible. Projects will always need resources but they can often be scarce. The task, therefore, lies with the project manager to determine the proper timing and allocation of those resources within the project schedule.

So, what is resource allocation in project management? It is the management and delegation of resources throughout a project to ensure that it runs as smoothly and successfully as possible.

Six Sigma:

Six Sigma is a method of project management and is sometimes considered an alternative to project management.

It is a set of organisational tools that help improve the business processes. Six sigma aims to **reduce the variations in process** and therefore increase overall performance. It is given credit for reducing defective products and services, improving profits and employee morale and therefore **positively impacting the customer experience**.
If a process is said to have six sigma qualities, then this means that **the process is well controlled**, and the errors are well within tolerance.

There are many definitions of six sigma, but there is a common thread of organisation within statistical tolerance. It **requires projects to have clear outcomes** that are closely aligned to a business strategy, all of which are measurable.

The difference in opinion comes from whether Six Sigma is a philosophy, a set of tools and methodology or a set of metrics. The question is whether it matters and whether it is something of all these things.

Six Sigma is a philosophy because **it determines that processes can be measured** and that if you work on input, then you can control outputs. It is a set of qualitative and quantitative techniques which include statistical process control, control charts, failure mode and effect analysis and process mapping.

It can be considered a methodology because there are steps to follow: define, measure, analyse, improve and control. Finally, it is a set of performance metrics that allow you to account for shifts in process quality.

As well as Six Sigma there is **Lean Six Sigma**.
This is a method that requires the team to collaborate in efforts to improve performance by removing waste and reducing variation in quality. It is still a statistical process that gives quantitative analysis to improvements, rather than applying guesswork.

Software Quality:

Software quality product is defined in term of its fitness of purpose. That is, a quality product does precisely what the users want it to do. For software products, the fitness of use is generally explained in terms of satisfaction of the requirements laid down in the SRS document. Although "fitness of purpose" is a satisfactory interpretation of quality for many devices such as a car, a table fan, a grinding machine, etc.for software products, "fitness of purpose" is not a wholly satisfactory definition of quality.

Example: Consider a functionally correct software product. That is, it performs all tasks as specified in the SRS document. But, has an almost unusable user interface. Even though it may be functionally right, we cannot consider it to be a quality product.

The modern view of a quality associated with a software product several quality methods such as the following:

Portability: A software device is said to be portable, if it can be freely made to work in various operating system environments, in multiple machines, with other software products, etc

Usability: A software product has better usability if various categories of users can easily invoke the functions of the product.

Reusability: A software product has excellent reusability if different modules of the product can quickly be reused to develop new products.

Correctness: A software product is correct if various requirements as specified in the SRS document have been correctly implemented.

Maintainability: A software product is maintainable if bugs can be easily corrected as and when they show up, new tasks can be easily added to the product, and the functionalities of the product can be easily modified, etc.

Software Quality Management System:

A quality management system is the principal methods used by organizations to provide that the products they develop have the desired quality.

A quality system subsists of the following:

Managerial Structure and Individual Responsibilities: A quality system is the responsibility of the organization as a whole. However, every organization has a sever quality department to perform various quality system activities. The quality system of an arrangement should have the support of the top management. Without help for the quality system at a high level in a company, some members of staff will take the quality system seriously.

Quality System Activities: The quality system activities encompass the following:

Usability: A software product has better usability if various categories of users can easily invoke the functions of the product.

Reusability: A software product has excellent reusability if different modules of the product can quickly be reused to develop new products.

Correctness: A software product is correct if various requirements as specified in the SRS document have been correctly implemented.

Maintainability: A software product is maintainable if bugs can be easily corrected as and when they show up, new tasks can be easily added to the product, and the functionalities of the product can be easily modified, etc.

Software Quality Management System

A quality management system is the principal methods used by organizations to provide that the products they develop have the desired quality.

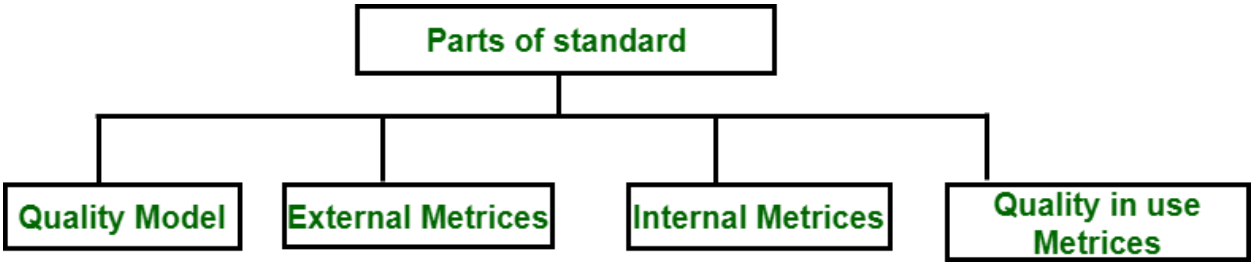
A quality system subsists of the following:

Managerial Structure and Individual Responsibilities: A quality system is the responsibility of the organization as a whole. However, every organization has a sever quality department to perform various quality system activities. The quality system of an arrangement should have the support of the top management. Without help for the quality system at a high level in a company, some members of staff will take the quality system seriously.

Quality System Activities: The quality system activities encompass the following:

ISO/IEC 9126 in Software Engineering:

ISO/IEC 9126 is an international standard proposed to make sure **'quality of all software-intensive products'** which includes a system like safety-critical where in case of failure of software lives will be in jeopardy. ISO i.e. International Organization for standardization and IEC i.e. International Electrotechnical Commission have developed ISO/IEC 9126 standards for software engineering –> **Product Quality** to provide an all-inclusive specification and **evaluation mode**l for the quality of the software product.
The standard is divided into 4 parts as depicted in the following figure :



Part-1 Software Engineering – Product Quality "Quality model" :

It describes quality model framework which explains relationships between different approaches to quality as well as identifying quality characteristics and sub-characteristics of software products.

Part-2 Software Engineering – Product Quality "External Metrices" :

It's use is to describes external metrices that are used to measure characteristics and sub-characteristics which are identifies in part 1.

Part-3 Software Engineering – Product Quality "Internal Metrices" :

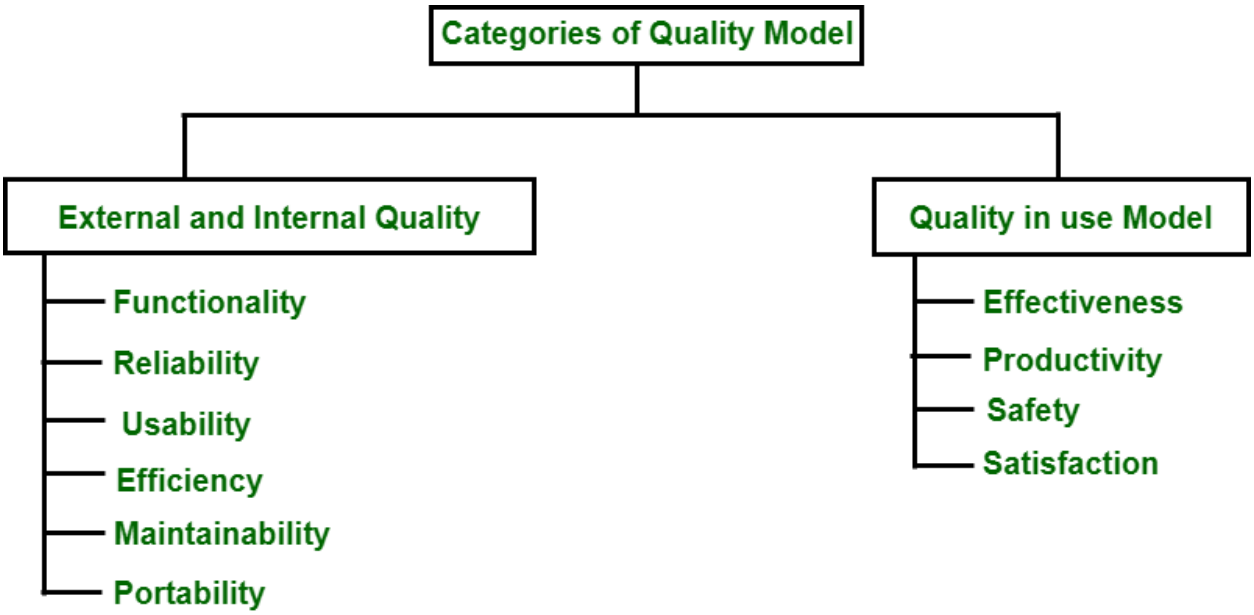
It's use is to describes internal metrices that are used to measure characteristics and sub-characteristics which are identifies in part 1.

Part-3 Software Engineering – Product Quality "Quality in use metrices" :

It's use is to identify metrices which are used to measure effects of combined quality characteristics for user.

As from above discussion, it is concluded that first three parts are concerned with describing and measuring quality of software product and fourth part concerned about quality of software product from user point of view.

Furthermore, first part i.e. Quality model is concerned classified into two categories as depicted in the following figure :



Internal External Quality Part : It determines the quality of a software product through six characteristics which are Functionality, Reliability, Usability, Efficiency, Maintainability and Portability. Each characteristics is subdivided into related sub-characteristics which are also depicted in the above example.

1. **Functionality:** The functions are those that will satisfy implied needs.
   - Suitability
   - Accuracy
   - Interoperability
   - Security
   - Functionality Compliance
   -
2. **Reliability:** A set of attributes that will bear on the capability of software to maintain the level of performance.
   - Maturity
   - Fault Tolerance
   - Recoverability
   - Reliability Compliance

3. **Usability:** A set of attributes that bear on the effort needed for use by a implied set of users.
    - Understandability
    - Learn ability
    - Operability
    - Attractiveness
    - Usability Compliance

4. **Efficiency:** A set of attributes that bear on the relationship between the level of performance of the software under stated conditions.
    - Time Behavior
    - Resource Utilization
    - Efficiency Compliance

5. **Maintainability:** A set of attributes that bear on the effort needed to make specified modifications.
    - Analyzability
    - Changeability
    - Stability
    - Testability
    - Maintainability Compliance

6. **Portability:** A set of attributes that bear on the ability of software to be transferred from one environment to another.
    - Adaptability
    - Installability
    - Co-existence
    - Replace ability
    - Portability Compliance

Quality in use Model : It identifies the four quality characteristics i.e. Effectiveness, Productivity, Safety, Satisfaction.

UNIT II SOFTWARE EVALUATION AND COSTING 9

Project Evaluation: Strategic Assessment, Technical Assessment, cost-benefit analysis, Cash flow forecasting,

cost-benefit evaluation techniques, Risk Evaluation. Selection of Appropriate

Project approach: Choosing technologies, choice of process models, structured methods.

Chapter 3

Project evaluation

OBJECTIVES

When you have completed this chapter you will be able to:

- carry out an evaluation and selection of projects against strategic, technical and economic criteria;
- use a variety of cost-benefit evaluation techniques for choosing among competing project proposals;
- evaluate the risk involved in a project and select appropriate strategies for minimizing potential costs.

3.1 Introduction

Deciding whether or not to go ahead with a project is really a case of comparing a proposed project with the alternatives and deciding whether to proceed with it. That evaluation will be based on strategic, technical and economic criteria and will normally be undertaken as part of strategic planning or a feasibility study for any information system development. The risks involved also need to be evaluated.

'Do nothing' is an option which should always be considered.

In this chapter we shall be using the term *project* in a broader sense than elsewhere in the book. Our decision as to whether or not to proceed with a project needs to be based upon whether or not it is desirable to carry out the development *and operation* of a software system. The term project may therefore be used, in this context, to describe the whole life cycle of a system from conception through to final decommissioning.

The BS 6079 guidelines (see Appendix B) use the term project in this way.

Project evaluation is normally carried out in Step 0 of Step Wise (Figure 3.1). The subsequent steps of Step Wise are then concerned with managing the development project that stems from this project selection.

Healy, Patrick L., *Project management : getting the job done on time and in budget*, Port Melbourne, Vic.; Newton, USA, Butterworth-Heinemann, 1997.
Although it presents itself as a general book on project management, it is relatively thin on the technical aspects of project planning. However, it makes many useful points about dealing with political and behavioural issues.

Charles B. Handy, *Understanding Organizations*, 4th edn, London, Penguin, 1993.
A general book on this topic which is not IS-specific.

Belbin, R. Meredith, *Management teams : why they succeed or fail*, Oxford; Boston, Butterworth-Heinemann, 1996.

Belbin, R. Meredith, *Team Roles at Work*, Oxford; Boston, Butterworth-Heinemann,1993.
This updates but does not replace the 1981 book

Project management standards

British Standards Institution, *BS 6079 Guide to project management*, London, BSI, 1996.

Central Computer and Telecommunications Agency, *PRINCE 2 : [project management for business]*, London, The Stationery Office, 1996.

Project Management Institute and PMI Standards Committee, *A Guide to the project management body of knowledge*, Upper Darby, PA: Project Management Institute, 1996: obtainable via WWW from http://www.pmi.org.

Contracts

David Bainbridge, *Introduction to computer law*, 3rd edn, London, Pitman, 1996.

Index

A

B

C

D

E

E

I

J

K

L

M

N

O

P

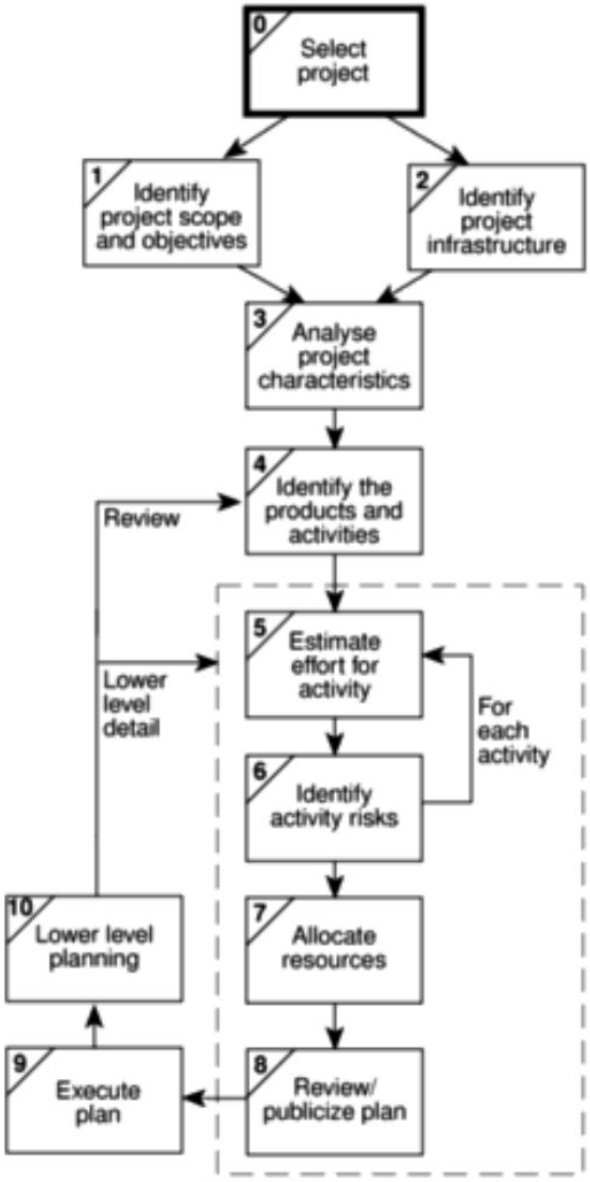


Figure 3.1 *Project evaluation is carried out in Step 0.*

D. C. Ferns defined a programme as 'a group of projects that are managed in a co-ordinated way to gain benefits that would not be possible were the projects to be managed independently' in a seminal article in the *International Journal of Project Management*, August 1991.

3.2 Strategic assessment

Programme management

It is being increasingly recognized that individual projects need to be seen as components of a *programme* and should be evaluated and managed as such. A programme, in this context, is a collection of projects that all contribute to the same overall organizational goals. Effective programme management requires that there is a well defined *programme goal* and that all the organization's projects are selected and tuned to contribute to this goal. A project must be evaluated according to how it contributes to this programme goal and its viability, timing, resourcing and final worth can be affected by the programme as a whole. It is to be expected

Q

R

S

T

U

V

W

Y

Z

that the value of any project is increased by the fact that it is part of a programme – the whole, as they say, being greater than the sum of the parts.

In order to carry out a successful strategic assessment of a potential project there should therefore be a strategic plan clearly defining the organization's objectives. This provides the context for defining the programme and programme goals and, hence, the context for assessing the individual project. It is likely, particularly in a large organization, that there will be an organizational structure for programme management and it will be, for example, the *programme director* and *programme executive*, rather than, say, a project manager, who will be responsible for the strategic assessment of a proposed project.

Even where there is no explicitly defined programme, any proposed project must be evaluated within the context of the organization's overall business objectives. Moreover, any potential software system will form part of the user organization's overall information system and must be evaluated within the context of the existing information system and the organization's information strategy. Table 3.1 illustrates typical issues that must be addressed as part of the strategic assessment of a project.

Table 3.1 *Typical issues and questions to be considered during strategic assessment*

Issue	*Typical questions*
Objectives	How will the proposed system contribute to the organization's stated objectives? How, for example, might it contribute to an increase in market share?
IS plan	How does the proposed system fit into the IS plan? Which existing system(s) will it replace/interface with? How will it interact with systems proposed for later development?
Organization structure	What effect will the new system have on the existing departmental and organization structure? Will, for example, a new sales order processing system overlap existing sales and stock control functions?
MIS	What information will the system provide and at what levels in the organization? In what ways will it complement or enhance existing management information systems?
Personnel	In what way will the proposed system affect manning levels and the existing employee skill base? What are the implications for the organization's overall policy on staff development?
Image	What, if any, will be the effect on customers' attitudes towards the organization? Will the adoption of, say, automated systems conflict with the objectives of providing a friendly service?

and operational costs, along with the value of the benefits of the new system are estimated. With a large system, the feasibility study could be treated as a project in its own right. This evaluation may be done as part of a strategic planning exercise where a whole range of potential software developments are evaluated and put into an order of priority. Sometimes an organization has a policy where a series of projects is planned as a *programme* of development.

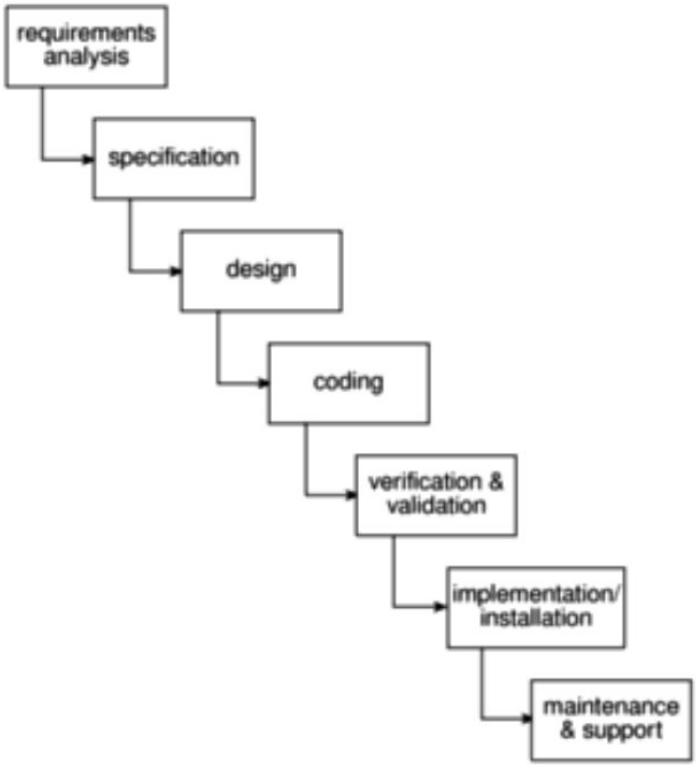


Figure 1.1 *A typical project life-cycle.*

The PRINCE 2 method, which is described in Appendix A takes this planning by stages approach.

2. **Planning** If the feasibility study produces results that indicate that the prospective project appears viable, then planning of the project can take place. In fact, for a large project, we would not do all our detailed planning right at the beginning. We would formulate an outline plan for the whole project and a detailed one for the first stage. More detailed planning of the later stages would be done as they approached. This is because we would have more detailed and accurate information upon which to base our plans nearer to the start of the later stages.

3. **Project execution** The project can now be executed. Individual projects are likely to differ considerably but a classic project life-cycle is shown in Figure 1.1.

The stages in the life-cycle illustrated in Figure 1.1 above are described in a little more detail below:

Requirements analysis This is finding out in detail what the users require of the system that the project is to implement. Some work along these lines will almost

Where a well-defined information systems strategy does not exist, system development and the assessment of project proposals will be based on a more piecemeal approach – each project being individually assessed early in its life cycle. In such cases it is likely that cost–benefit analysis will have more importance and some of the questions of Table 3.1 will be more difficult to answer.

Portfolio management

Third party developers must also carry out strategic and operational assessment of project proposals.

Where an organization such as a software house is developing a software system they could be asked to carry out a strategic and operational assessment on behalf of the customer. Whether or not this should be the case, they will require an assessment of any proposed project themselves. They will need to ensure that carrying out the development of a system is consistent with their own strategic plan – it is unlikely, for example, that a software house specializing in financial and accounting systems would wish to undertake development of a factory control system unless their strategic plan placed an emphasis on diversification.

The proposed project will form part of a *portfolio* of ongoing and planned projects and the selection of projects must take account of the possible effects on other projects in the portfolio (competition for resources, for example) and the overall portfolio profile (for example, specialization versus diversification).

3.3 Technical assessment

Technical assessment of a proposed system consists of evaluating the required functionality against the hardware and software available. Where an organization has a strategic information systems plan, this is likely to place limitations on the nature of solutions that might be considered. The constraints will, of course, influence the cost of the solution and this must be taken into account in the cost–benefit analysis.

3.4 Cost–benefit analysis

The most common way of carrying out an economic assessment of a proposed information system, or other development, is by comparing the expected costs of development and operation of the system with the benefits of having it in place.

Any project requiring an investment must, as a minimum, provide a greater benefit than putting that investment in, say, a bank.

Assessment is based upon the question of whether the estimated costs are exceeded by the estimated income and other benefits. Additionally, it is usually necessary to ask whether or not the project under consideration is the best of a number of options. There might be more candidate projects than can be undertaken at any one time and, in any case, projects will need to be prioritized so that any scarce resources may be allocated effectively.

The standard way of evaluating the economic benefits of any project is to carry out a cost–benefit analysis, which consists of two steps.

- **Identifying and estimating all of the costs and benefits of carrying out the project** This includes development costs of the system, the operating costs and the benefits that are expected to accrue from the operation of the system. Where the proposed system is replacing an existing one, these estimates should reflect the costs and benefits due to the new system. A sales order processing system, for example, could not claim to benefit an organization by the total value of sales – only by the increase due to the use of the new system.
- **Expressing these costs and benefits in common units** We must evaluate the net benefit, which is the difference between the total benefit and the total cost. To do this, we must express each cost and each benefit in monetary terms.

Most costs are relatively easy to identify and quantify in approximate monetary terms. It is helpful to categorize costs according to where they originate in the life of the project.

Many costs are easy to identify and measure in monetary terms.

- **Development costs** – include the salaries and other employment costs of the staff involved in the development project and all associated costs.
- **Setup costs** – include the costs of putting the system into place. These consist mainly of the costs of any new hardware and ancillary equipment but will also include costs of file conversion, recruitment and staff training.
- **Operational costs** – consist of the costs of operating the system once it has been installed.

Benefits, on the other hand, are often quite difficult to quantify in monetary terms even once they have been identified. Benefits may be categorized as follows.

Indirect benefits, which are difficult to estimate, are sometimes known as intangible benefits.

- **Direct benefits** – these accrue directly from the operation of the proposed system. These could, for example, include the reduction in salary bills through the introduction of a new, computerized system.
- **Assessable indirect benefits** – these are generally secondary benefits, such as increased accuracy through the introduction of a more user-friendly screen design where we might be able to estimate the reduction in errors, and hence costs, of the proposed system.
- **Intangible benefits** – these are generally longer term or benefits that are considered very difficult to quantify. Enhanced job interest can lead to reduced staff turnover and, hence, lower recruitment costs.

Exercise 3.1

Brightmouth College are considering the replacement of the existing payroll service, operated by a third party, with a tailored, off-the-shelf computer-based system. List some of the costs and benefits they might consider under each of the six headings given above. For each cost or benefit, explain how, in principle, it might be measured in monetary terms.

If a proposal shows an excess of benefits over costs then it is a candidate for further consideration.

Any project that shows an excess of benefits over costs is clearly worth considering for implementation. However, as we shall see later, it is not a sufficient justification for going ahead: we might not be able to afford the costs; there might be even better projects we could allocate our resources to instead; the project might be too risky.

3.5 Cash flow forecasting

As important as estimating the overall costs and benefits of a project is the forecasting of the cash flows that will take place and their timing. A cash flow forecast will indicate when expenditure and income will take place (Figure 3.2).

Typically products generate a negative cash flow during their development followed by a positive cash flow over their operating life. There might be decommissioning costs at the end of a product's life

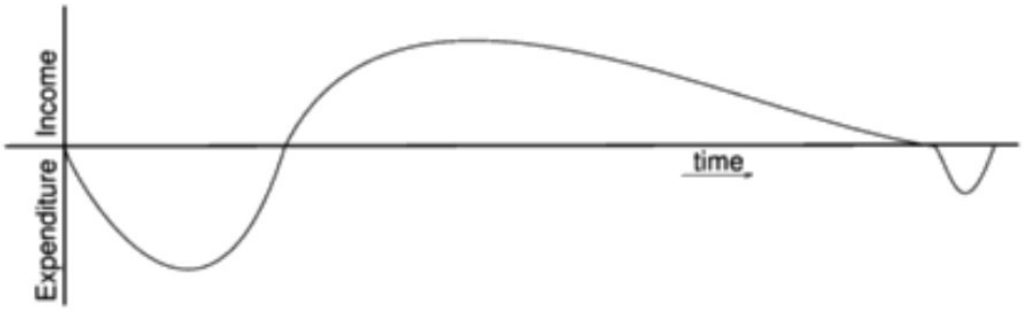


Figure 3.2 *Typical product life cycle cash flow.*

We need to spend money, such as staff wages, during the development stages of a project. Such expenditure cannot be deferred until income is received (either from using the software if it is being developed for in-house use or from selling it). It is important that we know that we can fund the development expenditure either from the company's own resources or by borrowing from the bank. In any event, it is vital to have some forecast of when expenditure such as the payment of salaries and bank interest will take place and when any income is to be expected, such as payment on completion or, possibly, stage payments.

The difficulty and importance of cash flow forecasting is evidenced by the number of companies that suffer bankruptcy because, although they are developing profitable products or services, they cannot sustain an unplanned negative cash flow.

Accurate cash flow forecasting is not easy, as it generally needs to be done early in the project's life cycle (at least before any significant expenditure is committed) and many items to be estimated (particularly the benefits of using software or decommissioning costs) might be some years in the future.

When estimating future cash flows, it is usual to ignore the effects of inflation. Trying to forecast the effects of inflation increases the uncertainty of the forecasts. Moreover, if expenditure is increased due to inflation it is likely that income will increase proportionately. However, measures to deal with increases in costs where work is being done for an external customer must be in place – for example index linked prices where work involves use of raw materials – see Chapter 10 on contract management.

Table 3.2 illustrates cash flow forecasts for four projects. In each case it is assumed that the cash flows take place at the end of each year. For short-term projects or where candidate projects demonstrate significant seasonal cash flow

patterns it can be advisable to produce quarterly, or even monthly, cash flow forecasts.

3.6 Cost–benefit evaluation techniques

We would consider proceeding with a project only where the benefits outweigh the costs. However, in order to choose among projects, we need to take into account the timing of the costs and benefits as well as the benefits relative to the size of the investment.

Exercise 3.2

Consider the project cash flow estimates for four projects at IOE shown in Table 3.2. Negative values represent expenditure and positive values income.

Rank the four projects in order of financial desirability and make a note of your reasons for ranking them in that way before reading further.

In the following sections we will take a brief look at some common methods for comparing projects on the basis of their cash flow forecasts.

Net profit

The net profit of a project is the difference between the total costs and the total income over the life of the project. Project 2 in Table 3.2 shows the greatest net profit but this is at the expense of a large investment. Indeed, if we had £1m to invest, we might undertake all of the other three projects and obtain an even greater net profit. Note also, that all projects contain an element of risk and we might not be prepared to risk £1m. We shall look at the effects of risk and investment later in this chapter.

Table 3.2 *Four project cash flow projections – figures are end of year totals (£)*

Year	*Project 1*	*Project 2*	*Project 3*	*Project 4*
0	−100,000	−1,000,000	−100,000	−120,000
1	10,000	200,000	30,000	30,000
2	10,000	200,000	30,000	30,000
3	10,000	200,000	30,000	30,000
4	20,000	200,000	30,000	30,000
5	100,000	300,000	30,000	75,000
Net profit	50,000	100,000	50,000	75,000

Cash flows take place at the end of each year. The year 0 figure represents the initial investment made at the start of the project.

Moreover, the simple net profit takes no account of the timing of the cash flows. Projects 1 and 3 each have a net profit of £50,000 and therefore, according to this selection criterion, would be equally preferable. The bulk of the income occurs late in the life of project 1, whereas project 3 returns a steady income throughout

its life. Having to wait for a return has the disadvantage that the investment must be funded for longer. Add to that the fact that, other things being equal, estimates in the more distant future are less reliable that short-term estimates and we can see that the two projects are not equally preferable.

Payback period

The *payback period* is the time taken to break even or pay back the initial investment. Normally, the project with the shortest payback period will be chosen on the basis that an organization will wish to minimize the time that a project is 'in debt'.

Exercise 3.3

Consider the four project cash flows given in Table 3.2 and calculate the payback period for each of them.

The advantage of the payback period is that it is simple to calculate and is not particularly sensitive to small forecasting errors. Its disadvantage as a selection technique is that it ignores the overall profitability of the project – in fact, it totally ignores any income (or expenditure) once the project has broken even. Thus the fact that projects 2 and 4 are, overall, more profitable than project 3 is ignored.

Return on investment

The *return on investment* (ROI), also known as the *accounting rate of return* (ARR), provides a way of comparing the net profitability to the investment required. There are some variations on the formula used to calculate the return on investment but a straightforward common version is

$$\text{ROI} = \frac{\text{average annual profit}}{\text{total investment}} \times 100$$

Exercise 3.4

Calculating the ROI for project 1, the net profit is £50,000 and the total investment is £100,000. The return on investment is therefore calculated as

$$\begin{aligned}\text{ROI} &= \frac{\text{average annual profit}}{\text{total investment}} \times 100 \\ &= \frac{10{,}000}{100{,}000} \times 100 = 10\%\end{aligned}$$

Calculate the ROI for each of the other projects shown in Table 3.2 and decide which, on the basis of this criterion, is the most worthwhile.

The return on investment provides a simple, easy to calculate measure of return on capital and is therefore quite popular. Unfortunately it suffers from two severe disadvantages. Like the net profitability, it takes no account of the timing of the cash flows. More importantly, it is tempting to compare the rate of return with current interest rates. However, this rate of return bears no relationship to the interest rates offered or charged by banks (or any other normal interest rate) since it takes no account of the timing of the cash flows or of the compounding of interest. It is therefore, potentially, very misleading.

Net present value

Net present value (NPV) and internal rate of return (IRR) are collectively known as discounted cash flow (DCF) techniques.

The calculation of *net present value* is a project evaluation technique that takes into account the profitability of a project and the timing of the cash flows that are produced. It does so by discounting future cash flows by a percentage known as the discount rate. This is based on the view that receiving £100 today is better than having to wait until next year to receive it, because the £100 next year is worth less than £100 now. We could, for example, invest the £100 in a bank today and have £100 plus the interest in a year's time. If we say that the present value of £100 in a year's time is £91, we mean that £100 in a year's time is the equivalent of £91 now.

Note that this example uses approximate figures – when you have finished reading this section you should be able to calculate the exact figures yourself.

The equivalence of £91 now and £100 in a year's time means we are discounting the future income by approximately 10% – that is, we would need an extra 10% to make it worthwhile waiting for a year. An alternative way of considering the equivalence of the two is to consider that, if we received £91 now and invested for a year at an annual interest rate of 10%, it would be worth £100 in a year's time. The annual rate by which we discount future earnings is known as the *discount rate* – 10% in the above example.

Similarly, £100 received in 2 years' time would have a present value of approximately £83 – in other words, £83 invested at an interest rate of 10% would yield approximately £100 in 2 years' time.

The present value of any future cash flow may be obtained by applying the following formula

$$\text{present value} = \frac{\text{value in year } t}{(1+r)^t}$$

where r is the discount rate, expressed as a decimal value and t is the number of years into the future that the cash flow occurs.

Alternatively, and rather more easily, the present value of a cash flow may be calculated by multiplying the cash flow by the appropriate discount factor. A small table of discount factors is given in Table 3.3.

The NPV for a project is obtained by discounting each cash flow (both negative and positive) and summing the discounted values. It is normally assumed that any initial investment takes place immediately (indicated as year 0) and is not discounted. Later cash flows are normally assumed to take place at the end of each year and are discounted by the appropriate amount.

Table 3.3 *Table of NPV discount factors*

	Discount rate (%)					
Year	*5*	*6*	*8*	*10*	*12*	*15*
1	0.9524	0.9434	0.9259	0.9091	0.8929	0.8696
2	0.9070	0.8900	0.8573	0.8264	0.7972	0.7561
3	0.8638	0.8396	0.7938	0.7513	0.7118	0.6575
4	0.8227	0.7921	0.7350	0.6830	0.6355	0.5718
5	0.7835	0.7473	0.6806	0.6209	0.5674	0.4972
6	0.7462	0.7050	0.6302	0.5645	0.5066	0.4323
7	0.7107	0.6651	0.5835	0.5132	0.4523	0.3759
8	0.6768	0.6274	0.5403	0.4665	0.4039	0.3269
9	0.6446	0.5919	0.5002	0.4241	0.3606	0.2843
10	0.6139	0.5584	0.4632	0.3855	0.3220	0.2472
15	0.4810	0.4173	0.3152	0.2394	0.1827	0.1229
20	0.3769	0.3118	0.2145	0.1486	0.1037	0.0611
25	0.2953	0.2330	0.1460	0.0923	0.0588	0.0304

More extensive or detailed tables may be constructed using the formula

$$\text{discount factor} = \frac{1}{(1+r)^t}$$

for various values of r (the discount rate) and t (the number of years from now)

Exercise 3.5

Assuming a 10% discount rate, the NPV for project 1 (Table 3.2) would be calculated as in Table 3.4. The net present value for Project 1, using a 10% discount rate is therefore £618. Using a 10% discount rate, calculate the net present values for projects 2, 3 and 4 and decide which, on the basis of this, is the most beneficial to pursue.

Table 3.4 *Applying the discount factors to project 1*

Year	*Project 1 cash flow (£)*	*Discount factor @ 10%*	*Discounted cash flow (£)*
0	−100,000	1.0000	−100,000
1	10,000	0.9091	9,091
2	10,000	0.8264	8,264
3	10,000	0.7513	7,513
4	20,000	0.6830	13,660
5	100,000	0.6209	62,090
Net Profit:	£50,000		NPV: £618

It is interesting to note that the net present values for projects 1 and 3 are significantly different – even though they both yield the same net profit and both have the same return on investment. The difference in NPV reflects the fact that, with project 1, we must wait longer for the bulk of the income.

The main difficulty with NPV for deciding between projects is selecting an appropriate discount rate. Some organizations have a standard rate but, where this is not the case, then the discount rate should be chosen to reflect available interest rates (borrowing costs where the project must be funded from loans) plus some premium to reflect the fact that software projects are inherently more risky than lending money to a bank. The exact discount rate is normally less important than ensuring that the same discount rate is used for all projects being compared. However, it is important to check that the ranking of projects is not sensitive to small changes in the discount rate – have a look at the following exercise.

Exercise 3.6

Calculate the net present value for each of the projects A, B and C shown in Table 3.5 using each of the discount rates 8%, 10% and 12%.

For each of the discount rates, decide which is the best project. What can you conclude from these results?

Alternatively, the discount rate can be thought of as a target rate of return. If, for example, we set a target rate of return of 15% we would reject any project that did not display a positive net present value using a 15% discount rate. Any project that displayed a positive NPV would be considered for selection – perhaps by using an additional set of criteria where candidate projects were competing for resources.

Table 3.5 *Three estimated project cash flows*

Year	*Project A (£)*	*Project B (£)*	*Project C (£)*
0	– 8,000	– 8,000	– 10,000
1	4,000	1,000	2,000
2	4,000	2,000	2,000
3	2,000	4,000	6,000
4	1,000	3,000	2,000
5	500	9,000	2,000
6	500	–6,000	2,000
Net Profit	4,000	5,000	6,000

Internal rate of return

One disadvantage of NPV as a measure of profitability is that, although it may be used to compare projects, it might not be directly comparable with earnings from other investments or the costs of borrowing capital. Such costs are usually quoted

as a percentage interest rate. The internal rate of return (IRR) attempts to provide a profitability measure as a percentage return that is directly comparable with interest rates. Thus, a project that showed an estimated IRR of 10% would be worthwhile if the capital could be borrowed for less than 10% or if the capital could not be invested elsewhere for a return greater than 10%.

The IRR is calculated as that percentage discount rate that would produce an NPV of zero. It is most easily calculated using a spreadsheet or other computer program that provides functions for calculating the IRR. Microsoft Excel and Lotus, for example, both provide IRR functions which, provided with an initial guess or seed value (which may be zero), will search for and return an IRR.

Manually, it must be calculated by trial-and-error or estimated using the technique illustrated in Figure 3.3. This technique consists of guessing two values

The IRR may be estimated by plotting a series of guesses: For a particular project, a discount rate of 8% gives a positive NPV of £7,898; a discount rate of 12% gives a negative NPV of −£5,829. The IRR is therefore somewhere between these two values. Plotting the two values on a chart and joining the points with a straight line suggests that the IRR is about 10.25%. The true IRR (calculated with a spreadsheet) is 10.167%.

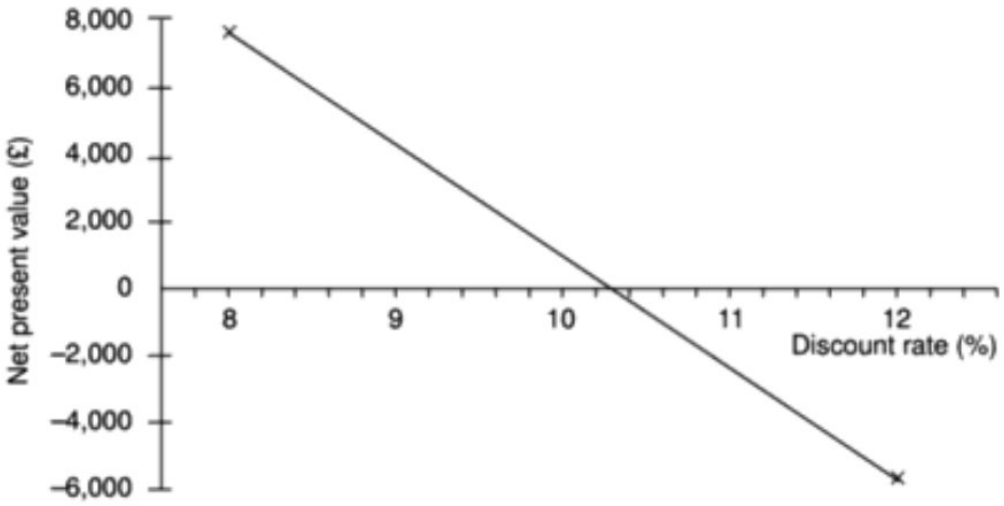


Figure 3.3 *Estimating the internal rate of return for project 1.*

(one either side of the true value) and using the resulting NPVs (one of which must be positive and the other negative) to estimate the correct value. Note that this technique will provide only an approximate value but, in many cases that can be sufficient to dismiss a project that has a small IRR or indicate that it is worth making a more precise evaluation.

The internal rate of return is a convenient and useful measure of the value of a project in that it is a single percentage figure that may be directly compared with rates of return on other projects or interest rates quoted elsewhere.

Table 3.6 illustrates the way in which a project with an IRR of 10% may be directly compared with other interest rates. The cash flow for the project is shown in column (a). Columns (b) to (e) show that if we were to invest £100,000 now at an annual interest rate of 10% in, say, a bank, we could withdraw the same amounts as we would earn from the project at the end of each year, column (e), and we would be left with a net balance of zero at the end. In other words, investing in a project that has an IRR of 10% can produce exactly the same cash flow as lending the money to a bank at a 10% interest rate. We can therefore reason

that a project with an IRR greater than current interest rates will provide a better rate of return than lending the investment to a bank. We can also say that it will be worth borrowing to finance the project if it has an IRR greater than the interest rate charged on the loan.

Table 3.6 *A project cash flow treated as an investment at 10%*

		Equivalent investment at 10%			
Year	*(a) Project cash flow forecast (£)*	*(b) Capital at start of year (£)*	*(c) Interest during year (£)*	*(d) Capital at end of year (£)*	*(e) End of year withdrawal (£)*
0	−100,000	—	—	—	—
1	10,000	100,000	10,000	110,000	10,000
2	10,000	100,000	10,000	110,000	10,000
3	10,000	100,000	10,000	110,000	10,000
4	20,000	100,000	10,000	110,000	20,000
5	99,000	90,000	9,000	99,000	99,000
6		0	0	0	0

£100,000 invested at 10% may be used to generate the cash flows shown. At the end of the 5-year period the capital and the interest payments will be entirely consumed leaving a net balance of zero.

One deficiency of the IRR is that it does not indicate the absolute size of the return. A project with an NPV of £100,000 and an IRR of 15% can be more attractive than one with an NPV of £10,000 and an IRR of 18% – the return on capital is lower but the net benefits greater.

An often quoted objection to the internal rate of return is that, under certain conditions, it is possible to find more than one rate that will produce a zero NPV. This is not a valid objection since, if there are multiple solutions, it is always appropriate to take the lowest value and ignore the others. Spreadsheets will normally always return the lowest value if provided with zero as a seed value.

NPV and IRR are not, however, a complete answer to economic project evaluation.

- A total evaluation must also take into account the problems of funding the cash flows – will we, for example, be able to repay the interest on any borrowed money and pay development staff salaries at the appropriate time?
- While a project's IRR might indicate a profitable project, future earnings from a project might be far less reliable than earnings from, say, investing with a bank. To take account of the risk inherent in investing in a project, we might require that a project earn a 'risk premium' (that is, it must earn, say, at least 15% more than current interest rates) or we might undertake a more detailed risk analysis as described in the following sections of this chapter.
- We must also consider any one project within the financial and economic framework of the organization as a whole – if we fund this one, will we also be able to fund other worthy projects?

certainly have been carried out when the project was evaluated but now the original information obtained needs to be updated and supplemented. Several different approaches to the users' requirements may be explored. For example, a small system that satisfies some, but not all, of the users' needs at a low price may be compared to a system with more functions but at a higher price.

Specification Detailed documentation of what the proposed system is to do.

Design A design that meets the specification has to be drawn up. This design activity will be in two stages. One will be the external or user design. This lays down what the system is to look like to the users in terms of menus, screen and report layouts and so on. The next stage produces the physical design, which tackles the way in which the data and software procedures are be structured internally.

Coding This might refer to writing code in a procedural language such as C or Ada, or might refer to the use of a high level application builder. Even where software is not being built from scratch, some modification to the base application might be required to meet the needs of the new application.

Verification and validation Whether software is developed specially for the current application or not, careful testing will be needed to check that the proposed system meets its requirements.

Implementation/installation Some system development practitioners refer to the whole of the project after design as 'implementation' (that is, the implementation of the design) while others insist that the term refers to the installation of the system after the software has been developed. In this case it encompasses such things as setting up data files and system parameters, writing user manuals and training users of the new system.

Maintenance and support Once the system has been implemented there will be a continuing need for the correction of any errors that may have crept into the system and for extensions and improvements to the system. Maintenance and support activities may be seen as a series of minor software projects. In many environments, most software development is in fact maintenance.

Exercise 1.2

Brightmouth College is a higher education institution which used to be managed by the local government authority but has now become autonomous. Its payroll is still administered by the local authority and pay slips and other output are produced in the local authority's computer centre. The authority now charges the college for this service. The college management are of the opinion that it would be cheaper to obtain an 'off-the-shelf' payroll application and do the payroll processing themselves.

What would be the main stages of the project to convert to independent payroll processing by the college? Bearing in mind that an off-the-shelf application is to

3.7 Risk evaluation

There is a risk that software might exceed the original specification and that a project will be completed early and under budget. That is not a risk that need concern us.

Every project involves risk of some form. When assessing and planning a project, we are concerned with the risk of the project's not meeting its objectives. In Chapter 8 we shall discuss ways of analysing and minimizing risk during the development of a software system. In this chapter, we are concerned with taking risk into account when deciding whether to proceed with a proposed project.

Risk identification and ranking

In any project evaluation we should attempt to identify the risks and quantify their potential effects. One common approach to risk analysis is to construct a project risk matrix utilizing a checklist of possible risks and to classify each risk according to its relative importance and likelihood. Note that the importance and likelihood need to be separately assessed – we might be less concerned with something that, although serious, is very unlikely to occur than with something less serious that is almost certain. Table 3.7 illustrates a basic project risk matrix listing some of the risks that might be considered for a project, with their importance and likelihood classified as high (H), medium (M), low (L) or exceedingly unlikely (—). So that projects may be compared the list of risks must be the same for each project being assessed. It is likely, in reality, that it would be somewhat longer than shown and more precisely defined.

The project risk matrix may be used as a way of evaluating projects (those with high risks being less favoured) or as a means of identifying and ranking the risks for a specific project. In Chapter 7 we shall consider a method for scoring the importance and likelihood of risks that may be used in conjunction with the project risk matrix to score and rank projects.

Risk and net present value

Where a project is relatively risky it is common practice to use a higher discount rate to calculate net present value. This addition or risk premium, might, for example, be an additional 2% for a reasonably safe project or 5% for a fairly risky one. Projects may be categorized as high, medium or low risk using a scoring method and risk premiums designated for each category. The premiums, even if arbitrary, provide a consistent method of taking risk into account.

Cost–benefit analysis

A rather more sophisticated approach to the evaluation of risk is to consider each possible outcome and estimate the probability of its occurring and the corresponding value of the outcome. Rather than a single cash flow forecast for a project, we will then have a set of cash flow forecasts, each with an associated probability of occurring. The value of the project is then obtained by summing the cost or benefit for each possible outcome weighted by its corresponding probability. Exercise 3.7 illustrates how this may be done.

Table 3.7 *A fragment of a basic project risk matrix*

Risk	*Importance*	*Likelihood*
Software never completed or delivered	H	—
Project cancelled after design stage	H	—
Software delivered late	M	M
Development budget exceeded ≤ 20%	L	M
Development budget exceeded > 20%	M	L
Maintenance costs higher than estimated	L	L
Response time targets not met	L	H

Exercise 3.7

BuyRight, a software house, is considering developing a payroll application for use in academic institutions and is currently engaged in a cost-benefit analysis. Study of the market has shown that, if they can target it efficiently and no competing products become available, they will obtain a high level of sales generating an annual income of £800,000. They estimate that there is a 1 in 10 chance of this happening. However, a competitor might launch a competing application before their own launch date and then sales might generate only £100,000 per year. They estimate that there is a 30% chance of this happening. The most likely outcome, they believe, is somewhere in between these two extremes – they will gain a market lead by launching before any competing product becomes available and achieve an annual income of £650,000. BuyRight have therefore calculated their expected sales income as in Table 3.8.

Total development costs are estimated at £750,000 and sales are expected to be maintained at a reasonably constant level for at least four years. Annual costs of marketing and product maintenance are estimated at £200,000, irrespective of the market share gained. Would you advise them to go ahead with the project?

This approach is frequently used in the evaluation of large projects such as the building of new motorways, where variables such as future traffic volumes, and hence the total benefit of shorter journey times, are subject to uncertainty. The technique does, of course, rely on our being able to assign probabilities of occurrence to each scenario and, without extensive study, this can be difficult.

When used to evaluate a single project, the cost–benefit approach, by 'averaging out' the effects of the different scenarios, does not take account an organization's reluctance to risk damaging outcomes. Because of this, where overall profitability is the primary concern, it is often considered more appropriate for the evaluation of a portfolio of projects.

Risk profile analysis

An approach which attempts to overcome some of the objections to cost–benefit averaging is the construction of risk profiles using sensitivity analysis.

Table 3.8 *BuyRight's income forecasts*

Sales	*Annual sales income (£)* i	*Probability* p	*Expected Value (£)* $i \times p$
High	800,000	0.1	80,000
Medium	650,000	0.6	390,000
Low	100,000	0.3	30,000
Expected Income			500,000

This involves varying each of the parameters that affect the project's cost or benefits to ascertain how sensitive the project's profitability is to each factor. We might, for example, vary one of our original estimates by plus or minus 5% and recalculate the expected costs and benefits for the project. By repeating this exercise for each of our estimates in turn we can evaluate the sensitivity of the project to each factor.

By studying the results of a sensitivity analysis we can identify those factors that are most important to the success of the project. We then need to decide whether we can exercise greater control over them or otherwise mitigate their effects. If neither is the case, then we must live with the risk or abandon the project.

For an explanation of the Monte Carlo technique see any textbook on operational research.

Sensitivity analysis demands that we vary each factor one at a time. It does not easily allow us to consider the effects of combinations of circumstances, neither does it evaluate the chances of a particular outcome occurring. In order to do this we need to use a more sophisticated tool such as Monte Carlo simulation. There are a number of risk analysis applications available (such as *@Risk* from Palisade) that use Monte Carlo simulation and produce risk profiles of the type shown in Figure 3.4.

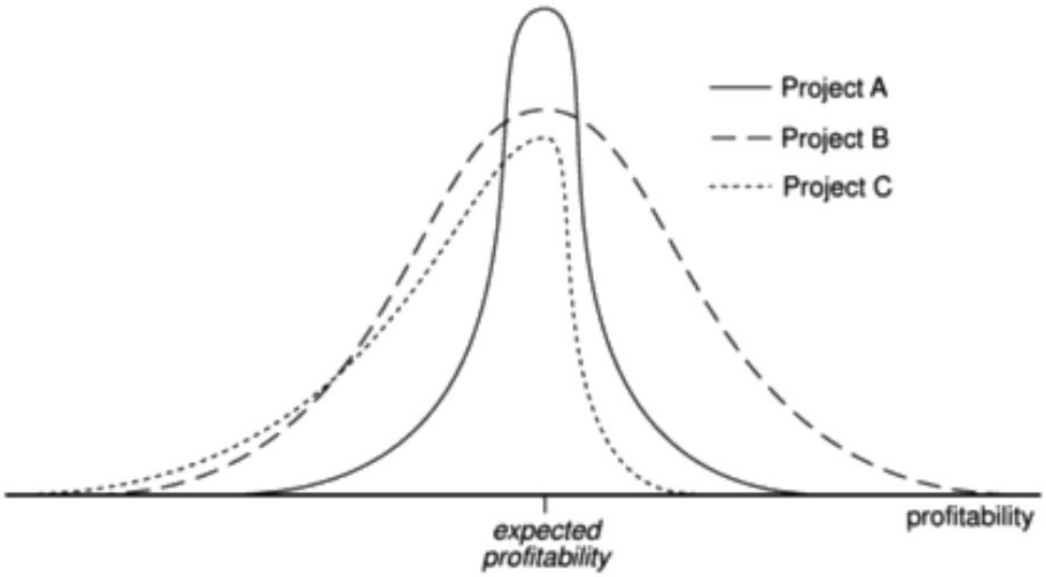
Projects may be compared as in Figure 3.4, which compares three projects with the same expected profitability. Project A is unlikely to depart far from this expected value compared to project B, which exhibits a larger variance. Both of these have symmetric profiles, which contrast with project C. Project C has a skewed distribution, which indicates that although it is unlikely ever to be much more profitable than expected, it is quite likely to be far worse.

Using decision trees

The approaches to risk analysis discussed previously rather assume that we are passive bystanders allowing nature to take its own course – the best we can do is to reject over-risky projects or choose those with the best risk profile. There are many situations, however, where we can evaluate whether a risk is important and, if it is, indicate a suitable course of action.

Many such decisions will limit or affect future options and, at any point, it is important to be able to see into the future to assess how a decision will affect the future profitability of the project.

Prior to giving Amanda the job of extending their invoicing system, IOE must consider the alternative of completely replacing the existing system – which they



All three projects have the same expected profitability.
The profitability of project A is unlikely to depart greatly from its expected value (indicated by the vertical axis) compared to the likely variations for project B. Project A is therefore less risky than project B.

Figure 3.4 *A risk analysis profile.*

will have to do at some point in the future. The decision largely rests upon the rate at which their equipment maintenance business expands – if their market share significantly increases (which they believe will happen if rumours of a competitor's imminent bankruptcy are fulfilled) the existing system might need to be replaced within 2 years. Not replacing the system in time could be an expensive option as it could lead to lost revenue if they cannot cope with the increase in invoicing demand. Replacing it immediately will, however, be expensive as it will mean deferring other projects that have already been scheduled.

They have calculated that extending the existing system will have an NPV of £57,000, although if the market expands significantly, this will be turned into a loss with an NPV of –£100,000 due to lost revenue. If the market does expand, replacing the system now has an NPV of £250,000 due to the benefits of being able to handle increased sales and other benefits such as improved management information. If sales do not increase, however, the benefits will be severely reduced and the project will suffer a loss with an NPV of –£50,000.

The company estimate the likelihood of the market increasing significantly at 20% – and, hence, the probability that it will not increase as 80%. This scenario can be represented as a tree structure as shown in Figure 3.5.

The analysis of a decision tree consists of evaluating the expected benefit of taking each path from a decision point (denoted by D in Figure 3.5). The expected value of each path is the sum of the value of each possible outcome multiplied by its probability of occurrence. The expected value of extending the system is therefore £40,000 ($75{,}000 \times 0.8 - 100{,}000 \times 0.2$) and the expected value of replacing the system £10,000 ($250{,}000 \times 0.2 - 50{,}000 \times 0.8$). IOE should therefore choose the option of extending the existing system.

This example illustrates the use of a decision tree to evaluate a simple decision at the start of a project. One of the great advantages of using decision trees to

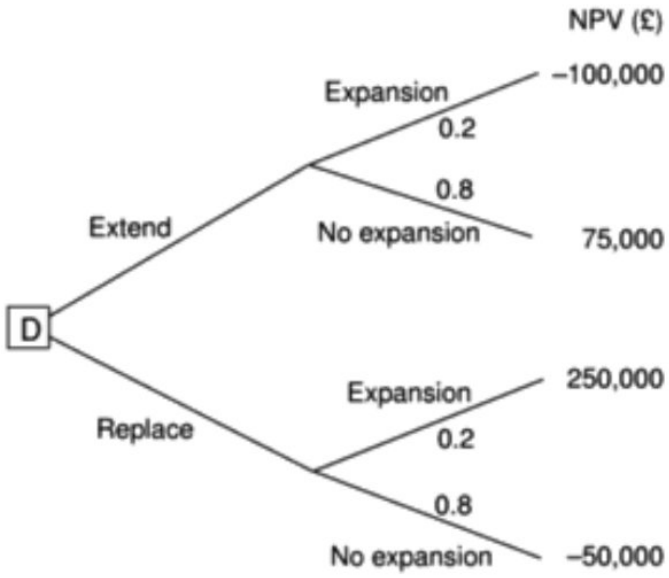


Figure 3.5 *A decision tree.*

model and analyse problems is the ease with which they can be extended. Figure 3.6 illustrates an extended version of Amanda's decision tree, which includes the possibility of a later decision should they decide to extend the system and then find there is an early market expansion.

The net present values shown in italic are those identified in Amanda's original decision tree shown in Figure 3.5.

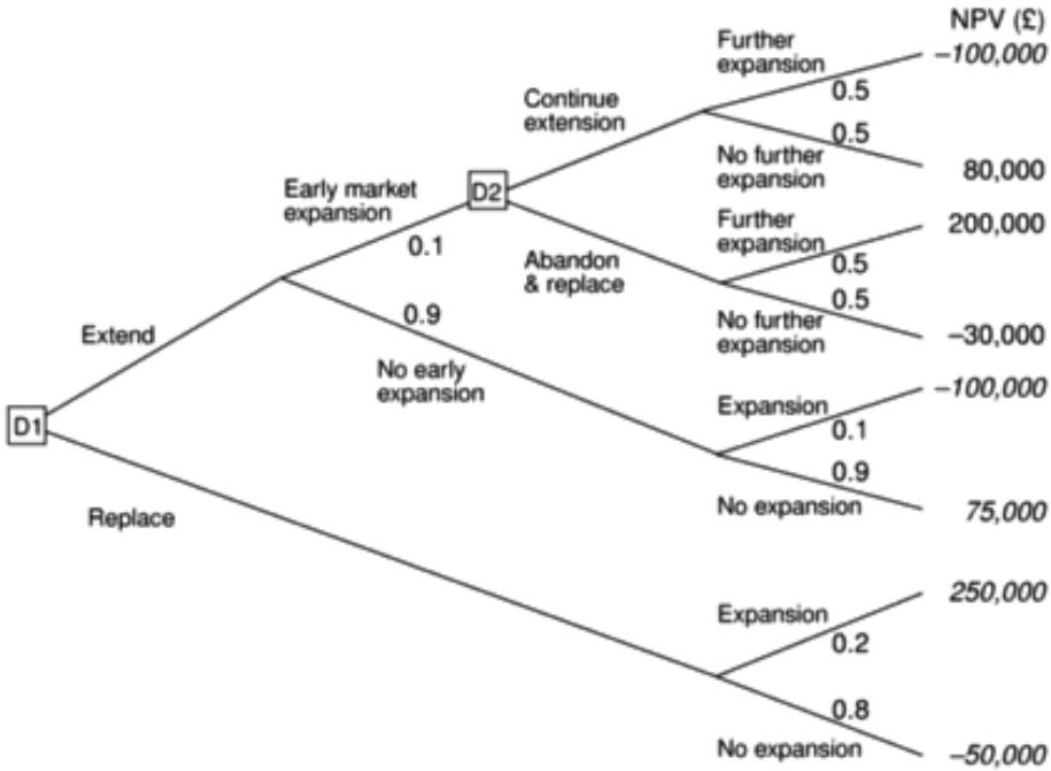


Figure 3.6 *An extension to Amanda's decision tree.*

3.8 Conclusion

Some of the key points in this chapter are:

- projects must be evaluated on strategic, technical and economic grounds;
- economic assessment involves the identification of all costs and income over the lifetime of the system, including its development and operation and checking that the total value of benefits exceeds total expenditure;
- money received in the future is worth less than the same amount of money in hand now, which may be invested to earn interest;
- the uncertainty surrounding estimates of future returns lowers their real value measured now;
- discounted cash flow techniques may be used to evaluate the present value of future cash flows taking account of interest rates and uncertainty;
- cost–benefit analysis techniques and decision trees provide tools for evaluating expected outcomes and choosing between alternative strategies.

3.9 Further exercises

1. Identify the major risks that could affect the success of the Brightmouth College Payroll project and try to rank them in order of importance.
2. Working in a group of three or four, imagine that you are about to embark upon a programming assignment as part of the assessed work for your course. Draw up a list of the risks that might affect the assignment outcome. Individually classify the importance and likelihood of each of those risk as high, medium or low. When you have done this compare your results and try to come up with an agreed project risk matrix.
3. Explain why discounted cash flow techniques provide better criteria for project selection than net profit or return on investment.
4. Consider the decision tree shown in Figure 3.6 and decide, given the additional possibilities, which option(s) IOE should choose.

4

PROJECT RISK MANAGEMENT

Unit Structure:

4.0 OBJECTIVES

After reading this chapter you will be able to:

1. Understand what risk is and the importance of good project risk management

2. Discuss the elements involved in risk management planning and contents of a risk management plan.
3. List common sources of risks on information technology projects
4. describe the risk identification process, tools and techniques to help identify project risks and the main output of risk identification-risk register
5. Discuss the qualitative risk analysis process and explain how to calculate risk factors, create probability impact matrixes, apply the top ten risk item tracking techniques, and use expert judgment to rank risks
6. Explain quantitative risk analysis process and how to apply decision trees, simulation, and sensitivity analysis to quantify risks
7. Provide examples of using different risk response planning strategies to address both negative and positive risks
8. Discuss the components of risk monitoring and control
9. Describe how software can assist in project risk management

4.1 INTRODUCTION

Managing risk is an integral part of good management and is something many managers do already in one form or another.

Project risk is an uncertain event or condition that, if it occurs, has a positive or a negative effect on at least one project objective. A risk may have one or more causes and, if it occurs, one or more impacts. Risk management is the systematic process of planning for, identifying, analyzing, responding to, and monitoring project risks. It involves processes, tools, and techniques that will help the project manager maximize the probability and results of positive events and minimize the probability and consequences of adverse events as indicated and appropriate within the context of risk to the overall project objectives of cost, time, scope and quality. Project risk management is most effective when first performed early in the life of the project and is a continuing responsibility throughout the project's life cycle.

4.2 THE IMPORTANCE OF PROJECT RISK MANAGEMENT

The project risk management process helps project sponsors and project teams make informed decisions regarding

alternative approaches to achieving their objectives and the relative risk involved in each, in order to increase the likelihood of success in meeting or exceeding the most important objectives (e.g. time) sometimes at the expense of other objectives (e.g. cost).

Risk Management provides a structured way of identifying and analyzing potential risks, and devising and implementing responses appropriate to their impact. These responses generally draw on strategies of risk prevention, risk transfer, impact mitigation or risk acceptance. Within a single project or proposal each of these strategies may have application for different individual risks.
Risk management encourages the project team to take appropriate measures to:

1. Minimize adverse impacts to project scope, cost, and schedule (and quality, as a result).
2. Maximize opportunities to improve the project's objectives with lower cost, shorter schedules, enhanced scope and higher quality.
3. Minimize management by crisis.

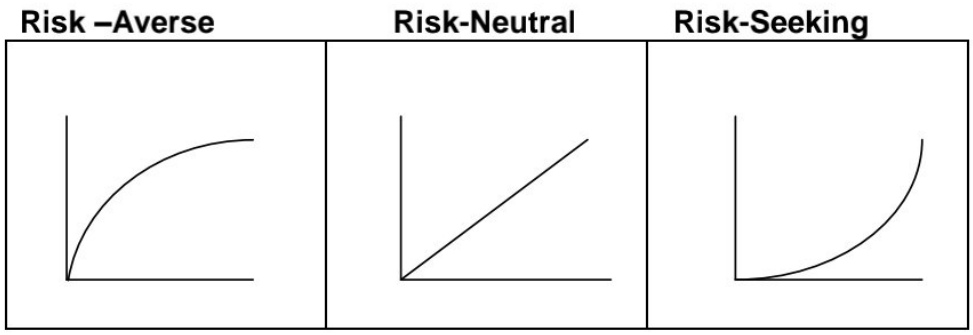
Project risk management is the art and science of identifying, analyzing and responding to risk throughout the life of a project and in the best interest of meeting the project objectives. A frequently overlooked aspect of project management, risk management can often result in significant improvements in the ultimate success of projects .Risk management can have a positive impact on selecting projects, determining scope of projects and developing realistic schedules and cost estimates. It helps project stakeholders understand the nature of the project, involves team members in defining the strengths and weakness , and helps to integrate the other project management knowledge areas.

Before you can improve project risk management, you must understand what risk is .A basic dictionary definition says that risk is "the possibility of loss or injury". This definition highlights the negativity often associated with risk and suggests that uncertainty is involved. Project risk management involves understanding potential problems that might occur on the project and how they might impede project success. The *PMBOK Guide 2004* refers to this type of risk as a negative risk. However, there are also positive risks, which can result in good things happening on a project. A general definition of a project risk , therefore , is an uncertainty that can have a negative or positive effect on meeting project objectives.

Some organizations or people have a neutral tolerance for risk, some have an aversion to risk, and others are risk seeking. These three preferences for risk are part of the utility theory of risk.

Risk utility or risk tolerance is the amount of satisfaction or pleasure received from a potential payoff. The following figure shows the basic difference between risk averse, risk neutral, and risk seeking preferences. The y-axis represents utility, or the amount of pleasure received from taking a risk. The x-axis shows the amount of potential payoff, opportunity, or dollar value of the opportunity at stake.

Utility rises at a decreasing rate for a risk averse person. That is when more payoff or money is at stake ,a person or organization that is risk averse gains less satisfaction from risk, or has lower tolerance for the risk. Those who are risk seeking have a higher tolerance for the risk, and their satisfaction increases when more payoff is at stake. A risk seeking person prefers outcomes that are more uncertain and is often willing to pay penalty to take risk. A risk neutral person achieves balance between risk and payoff..



Risk utility function and risk preference

The goal of project risk management can be viewed as minimizing potential negative risks while maximizing potential positive risks. The term known risks is some times used to describe risks that the project team has identified and analyzed .

4.2.1 Processes and outputs:

This matrix shows the six main processes and all of the deliverables associated with project risk management

Process	Output(deliverables)
Risk management planning	Risk management plan(RMP)

Risk identification	Risk Register (Register)
Qualitative risk analysis	Risk Register (updates) Prioritized list of risks classified as high, moderate, or low
Quantitative risk analysis	Quantitative Risk Analysis Reports Numerical analysis of the project's likelihood of achieving its overall objectives (Risk Register updates)
Risk response planning	1- Risk Register (updates) 2- Project Management Plan (updates) 3- Project Risk Management Plan (updates) 4- Risk-related contractual agreements The outcome may result in one or more of the following: residual risks, secondary risks, change control, contingency reserve (amounts of time or budget needed).
Risk monitoring and control	Risk Register (updates) The outcome may result in workaround plans, corrective actions, programming change request (PCR), and updates to risk identification checklists for future projects

4.3 RISK MANAGEMENT PLANNING

Risk management planning is the process of deciding how to approach and plan for risk management activities for a project, and the main output of this process is a risk management plan A risk management plan documents the procedure for managing risk throughout the project.

The project team should hold several planning meeting early in the project's life cycle to help develop the risk management plan. The project team should review the project documents as well as corporate risk management policies, risk categories, lessons learned reports from past projects and templates for creating risk management plan.

Careful and explicit planning enhances the possibility of success of the other risk management processes. Risk Management Planning is the process of deciding how to approach

and conduct the risk management activities for a project. Planning of risk management processes is important to ensure that the level, type, and visibility of risk management are commensurate with both the risk and importance of the project to the organization, to provide sufficient resources and time for risk management activities, and to establish an agreed-upon basis for evaluating risks. The Risk Management Planning process should be completed early during project planning, since it is crucial to successfully performing the other processes described in this handbook.

The result of Risk Management Planning is a Risk Management Plan. The risk management plan identifies and establishes the activities of risk management for the project in the project plan (RMP)

A risk management plan summarizes how risk management will be performed on a particular project. Like other specific knowledge area plans it becomes a subset of project management plan The following table lists the general topics that a risk management plan should address It is important to clarify roles and responsibilities, prepare budget and schedule estimates for risk-related work, and .identify risk categories for consideration. It is also important to describe how risk management will be done, including assessment of risk probabilities and impacts as well as creation of risk related documentation.

Methodology: How will risk management will be performed on this project?. What tools and data sources are available and applicable?
Roles and responsibilities: who are the individuals responsible for implementing specific tasks and providing deliverables related to risk management
Budget and schedule: What are the estimated costs and schedules for performing risk related activities?
Risk Categories: What are the main categories of risk that should be addressed on this project?. Is there a risk breakdown structure for the project
Risk Probability and Impact: How will the probabilities and impacts of risk items be assessed?. What scoring and interpretation methods will be used for the qualitative and quantitative analysis of risks?
Risk Documentation: What reporting formats and processes will used for risk management activities?

In addition to risk management plan many projects also include contingency plans, fallback plans, and contingency reserves. Contingency plans are predefined actions that the project team will take if an identified risk event occurs. For example ,if the project team knows that a new release of a software package may not be available in time for them to use it for their project, they might have a contingency plan to use the older version of the software.

Fallback plans are developed for risks that have a high impact on meeting project objectives and are put in to effect if attempts to reduce risks are not effective. For example , a new college graduate might have a main plan and several contingency plans on where to live after graduation, but if none of the plans work out a fallback plan might be to live at home for a while. Sometimes contingency plans and fallback plans are used interchangeably.

Contingency reserves or contingency allowances are provisions held by the project sponsor or organization to reduce the risk of cost or schedule overruns to an acceptable level. For example if a project appears to be off course because the the staff is inexperienced with some new technology and the team had not identified it as a risk ,the project sponsor may provide additional funds from contingency reserves to hire an outside consultant to train and advise the project staff in using the new technology.

4.4 COMMON SOURCES OF RISK IN INFORMATION TECHNOLOGY PROJECTS

Several studies show that IT projects share some common sources of risk. The Standish Group developed an IT success potential scoring sheet based on potential risks. Other broad categories of risk help identify potential risks.

Information Technology Success Potential Scoring Sheet

Success Criterion	Relative Importance
User Involvement	19
Executive Management support	16
Clear Statement of Requirements	15
Proper Planning	11
Realistic Expectations	10
Smaller Project Milestones	9
Competent Staff	8

Ownership	6
Clear Visions and Objectives	3
Hard-Working, Focused Staff	3
Total	100

The Standish Group provides specific questions for each success criterion to help decide the number of points to assign to a project. For example the five questions related to user involvement include the following
Do I have the right user(s)?
Did I involve the users early and often?
Do I have a quality relationship with user(s)?
Do I make involvement easy
Did I find out what the user(s) need(s)?

The number of questions corresponding to each success criterion determines the number of points each positive response is assigned. For example in the case of user involvement there are five questions . For each positive reply , you would get(19/5) 3.8 points . !9 represents the weight of the criterion and five represents the number of questions. Therefore ,you would assign a value to the user involvement criterion by adding 3.8 points to the score for each question you can answer positively.

4.4.1 Categories of Risk:

A broad categories of risks are described on the questionnaires developed by many organizations. Some of them are given below.

 Market risk: If the information technology project is to produce a new product or service will it be useful to the organization or marketable to others?. Will user accept the product or service?. Will someone else make a better product or service faster, making the project a waste of time and money.

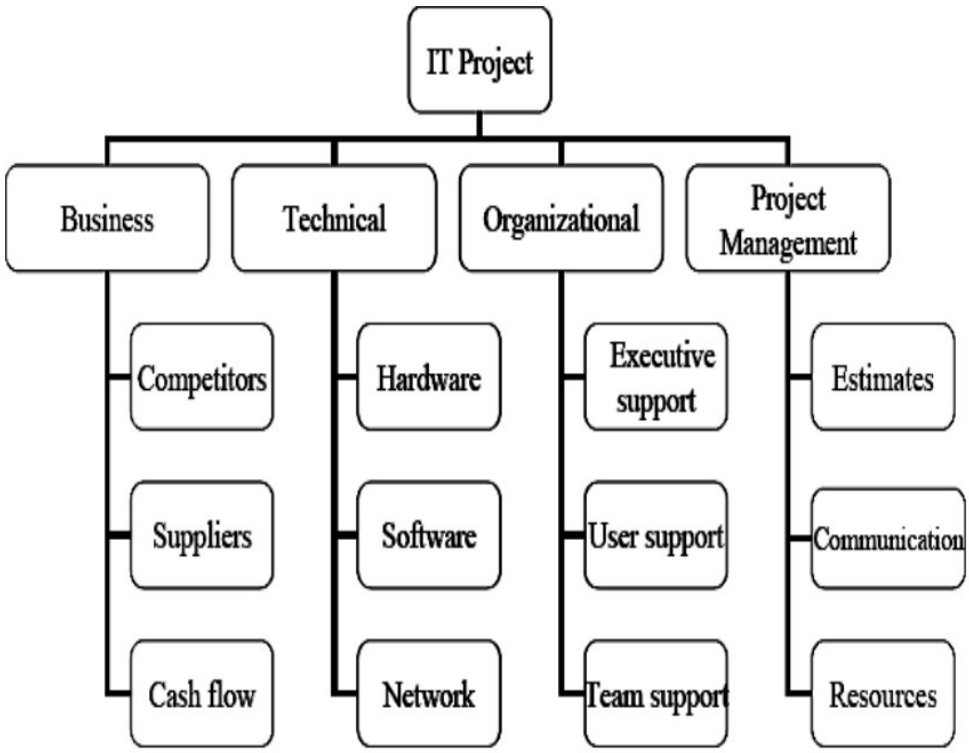
 Financial risk: Can the organization afford to undertake the project?. How confident are the stakeholders in the financial projections?. Will the project meet NPV,ROI, and payback estimates?. If not van the organization afford to proceed the project?. Is this project the best way to use the organization's financial resources?

 Technology risk: Is the project technically feasible?. Will it use mature, leading edge or bleeding edge technologies? When will decisions be made on which technology to use? Will H/w, S/w and network function properly?. You can also breakdown the technology risk into h/w, s/w, and network technology if required.

 People risk: Does the organization have or can they find people with appropriate skills to complete the project successfully?. Do they have enough experience?. Does senior management support the project?. Is the organization familiar sponsor/customer for the project?. How good is the relationship with the sponsor/customer?
 Structure/process risk: What is the degree of change the new project will introduce into user areas and business procedures? How many distinct user groups does the project need to satisfy? With how many other systems does the project need to interact? Does the organization have processes in place to complete the project successfully?

4.4.2 Risk Breakdown Structure:

 A **risk breakdown structure** is a hierarchy of potential risk categories for a project. Similar to a work breakdown structure but used to identify and categorize risks. A sample shown below.



A risk break down structure is a useful tool that can help project managers consider potential risks in different categories. The highest level categories are business, technical, and organizational and project management. Competitors suppliers, and cash flow are categories that fall under business risks. Under technical risk are the categories h/w, s/w, and network.

A risk break down structure provides a simple, one page chart to help ensure a project team is considering important risk categories related to all information technology projects.

The following table shows the potential negative risk conditions that can exist within each knowledge area.

Potential Risk Conditions Associated With Each Knowledge Area

Knowledge Area	Risk Conditions
Integration	Inadequate planning; poor resource allocation; poor integration management; lack of post-project review
Scope	Poor definition of scope or work packages; incomplete definition of quality requirements; inadequate scope control
Time	Errors in estimating time or resource availability; poor allocation and management of float; early release of competitive products
Cost	Estimating errors; inadequate productivity, cost, change, or contingency control; poor maintenance, security, purchasing, etc.
Quality	Poor attitude toward quality; substandard design/materials/workmanship; inadequate quality assurance program
Human Resources	Poor conflict management; poor project organization and definition of responsibilities; absence of leadership
Communications	Carelessness in planning or communicating; lack of consultation with key stakeholders
Risk	Ignoring risk; unclear assignment of risk; poor insurance management
Procurement	Unenforceable conditions or contract clauses; adversarial relations

4.5 RISK IDENTIFICATION

Risk identification involves identifying potential project risks. Risk Identification produces a deliverable — the project Risk Register – where risks are identified that may affect the project's ability to achieve its objectives. Risk Identification documents which risks might affect the project and documents their characteristics. The Risk Register is subsequently amended with the results from qualitative risk analysis and risk response planning, and is reviewed and updated throughout the project.

Participants in risk identification activities can include the following, where appropriate: project manager, project team members, risk management team (if assigned), subject matter experts both from the project and from outside the project team, customers, end users, other project managers, stakeholders, and risk management experts. While these personnel are often key

participants for risk identification, all project personnel should be encouraged to identify risks.

4.5.1 Suggestions For Identifying Risks:

The assigned team members identify the potential risks (threats and opportunities), using

- The risk breakdown structure, suitably tailored to the project.
- The sample risk list
- Their own knowledge of the project or similar projects.
- Consultation with others who have significant knowledge of the project or its environment.
- Consultation with others who have significant knowledge of similar projects.

There are several other tools and techniques also for identifying risks Five common information gathering techniques for risk identification include brainstorming ,Delphi technique ,interviewing ,root cause analysis, and SWOT analysis.

1. Brain Storming:

It is a technique by which a team attempt to generate ideas or find solutions for a specific by amassing ideas spontaneously and with out judgment . This approach can help the group create a comprehensive list of risks to address later in the qualitative and quantitative risk analysis process. An experienced facilitator should run the brainstorming session and introduce new categories of potential risks to keep the ideas flowing . After the ideas are collected ,the facilitator can group and categorize the ideas to make them more manageable

2. Delphi Technique:

The Delphi Technique is used to derive a consensus among a panel of experts who make predictions about future developments. It Provides independent and anonymous input regarding future events. Uses repeated rounds of questioning and written responses and avoids the biasing effects possible in oral methods, such as brainstorming.

3. Interviewing:

Interviewing is a fact-finding technique for collecting information in face-to-face, phone, e-mail, or instant messaging discussions. Interviewing people with similar project experience is an important tool for identifying potential risks.

4. SWOT Analysis:

- SWOT analysis (strengths, weaknesses, opportunities,and threats) can also be used during risk identification.

- Helps identify the broad negative risks that apply to a project.

Applying SWOT to specific potential projects can help identify the broad risks and opportunities that apply in that scenario Some other techniques for risk identification are

5. Use of checklists :

The list of risks that have been encountered in previous projects provide meaningful template for understanding risks in current projects.

It is important to analyze project assumptions to make sure that they are valid. Incomplete, inaccurate or inconsistent assumptions might lead to identifying more risks.

6. Diagramming Technique:

This method include using cause and effect diagrams or fishbone diagrams ,flow charts and influence diagrams .Fishbone diagrams help you trace problems back to their root cause. Process flow charts are diagrams that show how different parts of the system interrelate.

4.5.2 The Risk Register:

The main output of the risk identification process is a list of identified risks and other information needed to begin creating a risk register.

A risk register is:

- A document that contains the results of various risk management processes and that is often displayed in a table or spreadsheet format.
- A tool for documenting potential risk events and related information.

Risk Register Contents

- An identification number for each risk event.
- A rank for each risk event.
- The name of each risk event.
- A description of each risk event.
- The category under which each risk event falls.
- The root cause of each risk.

- Triggers for each risk; triggers are indicators or symptoms of actual risk events.
- Potential responses to each risk.
- The risk owner or person who will own or take responsibility for each risk.
- The probability and impact of each risk occurring.
- The status of each risk.

Sample Risk Register

No.	Rank	Risk	Description	Category	Root Cause	Triggers	Potential Responses	Risk Owner	Probability	Impact	Status
R44	1										
R21	2										
R7	3										

4.6 QUALITATIVE RISK ANALYSIS

- Assess the likelihood and impact of identified risks to determine their magnitude and priority.
- Risk quantification tools and techniques include:
- Probability/impact matrixes
- The Top Ten Risk Item Tracking
- Expert judgment

4.6.1 Using Probability/Impact Matrix To Calculate Risk Factors:

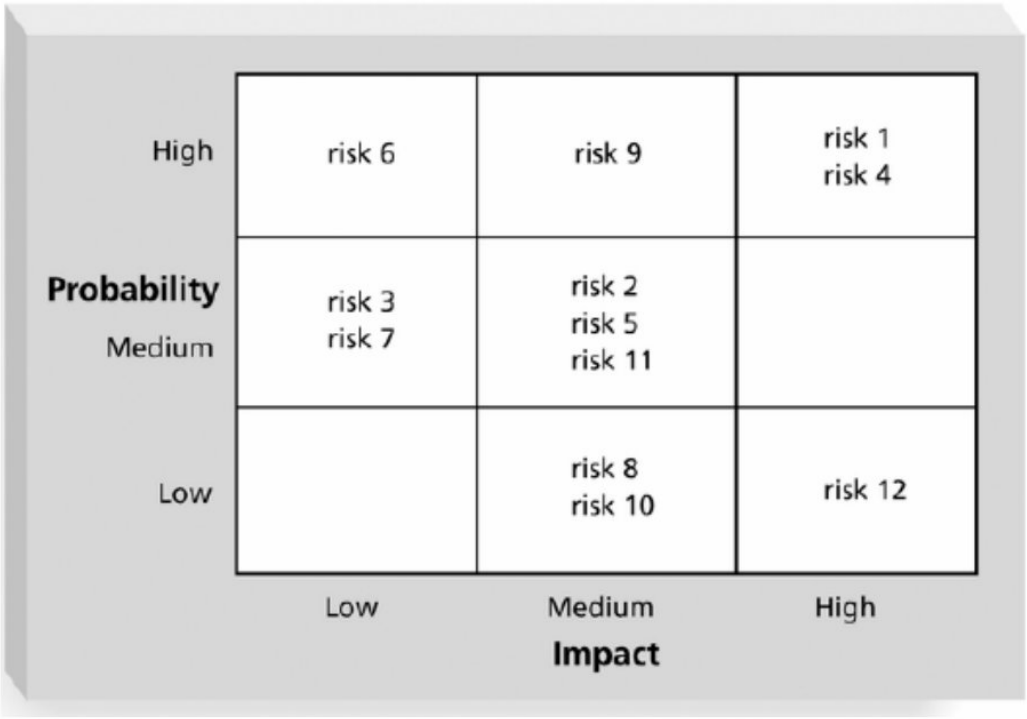
- A probability/impact matrix or chart lists the relative probability of a risk occurring on one side of a matrix or axis on a chart and the relative impact of the risk occurring on the other.

- List the risks and then label each one as high, medium, or low in terms of its probability of occurrence and its impact if it did occur.

It may be useful to create separate Probability/Impact Matrix or chart for negative risks and positive risks to make sure both types of risks are adequately addressed. Qualitative analysis is normally done quickly so that the project team has to decide what type of approach makes the most sense for their project. To quantify risk probability and consequence, the Defense Systems Management College developed a technique for calculating risk factors – the numbers that represent the overall risk of specific events ,based on their probability of occurring and consequences to the project if they do occur. The technique makes use of Probability/Impact Matrix that shows the probability of risks occurring and the impact or consequences of the risks.

Probability of a risk occurring can be estimated based on several factors as determined by the unique nature of each project . For example factors evaluated for potential H/W or S/W technology risks could include the technology not being mature, the technology being too complex, and an inadequate support base for developing the technology. The impact of a risk occurring could include factors such as availability of fallback solutions or the consequences of not meeting performance , cost and schedule estimates

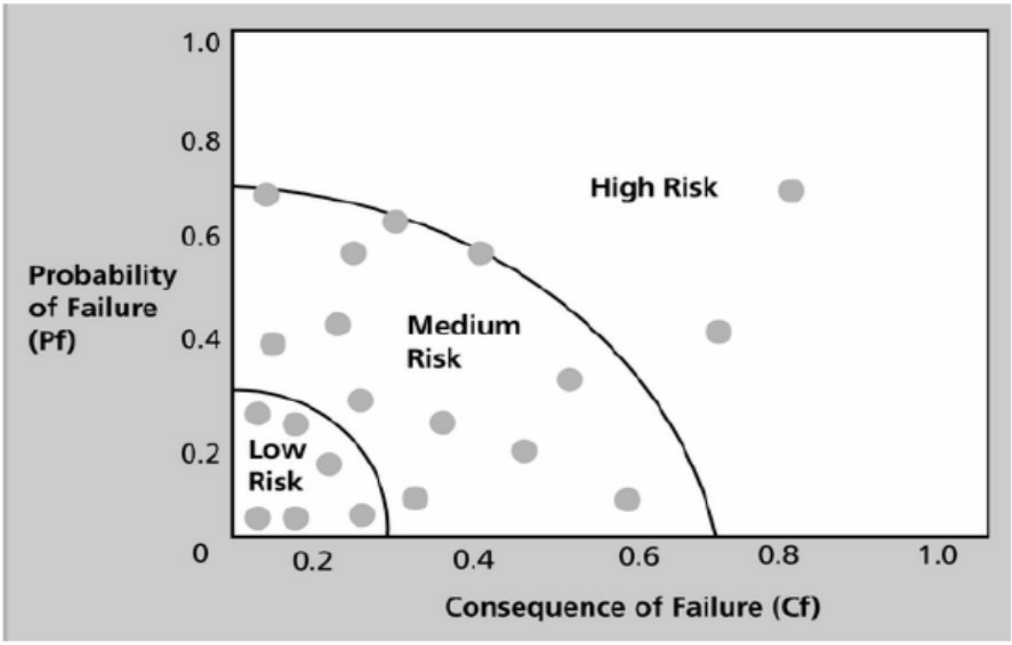
Sample Probability/Impact Matrix



The following figure gives an example of how the risk factors were used to graph the probability of failure and consequence of failure for proposed technologies. The figure classifies potential

technologies (dots on the charts) as high, medium, or low risk based on the probability of failure and consequence of failure. The researchers for this study highly recommended that the US Air Force invest in the low to medium risk technologies and suggested that it not pursue the high risk technologies. It can be seen that the rigor behind using Probability/Impact Matrix and risk factors provides a much stronger argument than simply stating the risk probabilities or consequences are high, medium, or low

Chart Showing High-, Medium-, and Low-Risk Technologies



4.6.2 Top Ten Risk Item Tracking:

Top Ten Risk Item Tracking is a qualitative risk analysis tool that helps to identify risks and maintain an awareness of risks throughout the life of a project. Establish a periodic review of the top ten project risk items.

The review begins with a summary of the status of top ten sources of risk on the project. The summary includes each item's current ranking previous ranking, number of times it appears on the list over a period of time, and a summary of progress made in resolving the risk item since the previous review.

List the current ranking, previous ranking, number of times the risk appears on the list over a period of time, and a summary of progress made in resolving the risk item.

The following figure provides an example of Top Ten Risk Item Tracking chart that could be used at a management review meeting for a project. This includes only the top five negative risk events. Each risk event is ranked based on the current month, previous month, and how many months it has been in the top ten. The last column briefly describes the progress for resolving each particular risk item

Example of Top Ten Risk Item Tracking

	Monthly Ranking			
Risk Item	**This Month**	**Last Month**	**Number of Months**	**Risk Resolution Progress**
Inadequate planning	1	2	4	Working on revising the entire project plan
Poor definition of scope	2	3	3	Holding meetings with project customer and sponsor to clarify scope
Absence of leadership	3	1	2	Just assigned a new project manager to lead the project after old one quit
Poor cost estimates	4	4	3	Revising cost estimates
Poor time estimates	5	5	3	Revising schedule estimates

4.6.3 Expert Judgment:

Many organizations rely on the intuitive feelings and past experience of experts to help identify potential project risks. Experts can categorize risks as high, medium, or low with or without more sophisticated techniques.

The main output of qualitative risk analysis is updating the risk register .The ranking column of the risk register should be filled in along with numeric value or high, medium, low for the probability and impact of the risk event. Additional information is often added for risk events, such as identification of risks that need more attention in the near term or those that can be placed on a watch list. A watch list is a list of risks that are low priority , but are still

identified as potential risks. Qualitative analysis can also identify risks that should be evaluated on a quantitative basis.

4.7 QUANTITATIVE RISK ANALYSIS

Often follows qualitative risk analysis, but both can be done together.

Large, complex projects involving leading edge technologies often require extensive quantitative risk analysis.
Main techniques include:

- Decision tree analysis
- Simulation
- Sensitivity analysis

Quantitative risk analysis is a way of numerically estimating the probability that a project will meet its cost and time objectives. Quantitative analysis is based on a simultaneous evaluation of the impact of all identified and quantified risks. The result is a probability distribution of the project's cost and completion date based on the identified risks in the project.

Quantitative risk analysis involves statistical techniques, primarily Monte Carlo simulation that is most widely and easily used with specialized software.

Quantitative risk analysis starts with the model of the project, either its project schedule or its cost estimate depending on the objective. The degree of uncertainty in each schedule activity and each line-item cost element is represented by a probability distribution. The probability distribution is usually specified by determining the optimistic, the most likely and the pessimistic values for the activity or cost element – this is typically called the "3-point estimate." The three points are estimated during an interview with subject matter experts who usually focus on the schedule or cost elements one at a time. The risks that lead to the three points are recorded for the quantitative risk analysis report and for risk response planning. For each activity or cost element a probability distribution type is chosen that best represents the risks discussed in the interview. Typical distributions usually include the triangular, beta, normal and uniform.

4.7.1 Decision Trees and Expected monetary Value:

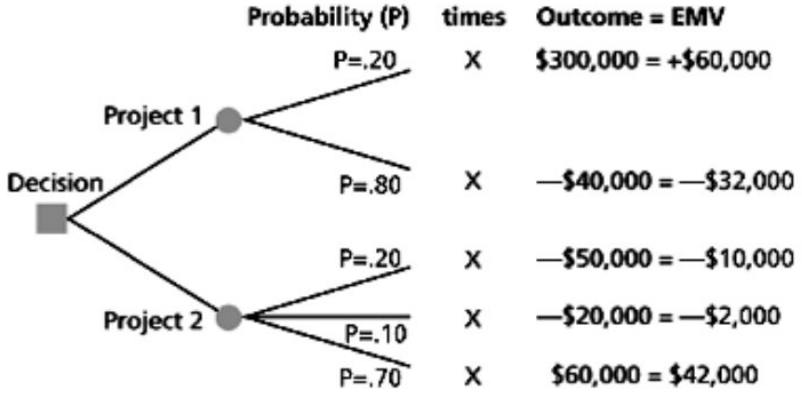
A decision tree is a diagramming analysis technique used to help select the best course of action in situations in which future outcomes are uncertain.

- Estimated monetary value (EMV) is the product of a risk event probability and the risk event's monetary value.

 You can draw a decision tree to help find the EMV.

To create a decision tree and to calculate expected monetary value specifically, you must estimate the probabilities, of certain events occurring. For example in the following figure there is a 20 percent probability(P=.20) that Cliff's firm will win the contract project1, which is estimated to be $300,000 in profits- the outcome of the top branch in the figure. There is an 80 percent probability that it will not win the contract for the project, and the outcome is estimated to be -$40,000 meaning that the firm has to invest $40,000 into project1with no reimbursement if it is not awarded the contract.

To calculate EMV for each project, multiply the probability by the outcome value for each potential outcome for each project .
The EMV for project 1 is 0.2($300,000)+0.8(-40,000)=$60,000-$32000=28,000



Project 1's EMV = $60,000 —32,000 = $28,000
Project 2's EMV = —$10,000 —2,000 + 42,000 = $30,000

4.7.2 Simulation:

A specialized Monte Carlo simulation software program runs (iterates) the project schedule or cost estimate many times, drawing duration or cost values for each iteration at random from the probability distribution derived from the 3-point estimates and probability distribution types selected for each element. The Monte Carlo software develops from the results of the simulation a probability distribution of possible completion dates and project costs. From this distribution it is possible to answer such questions as:

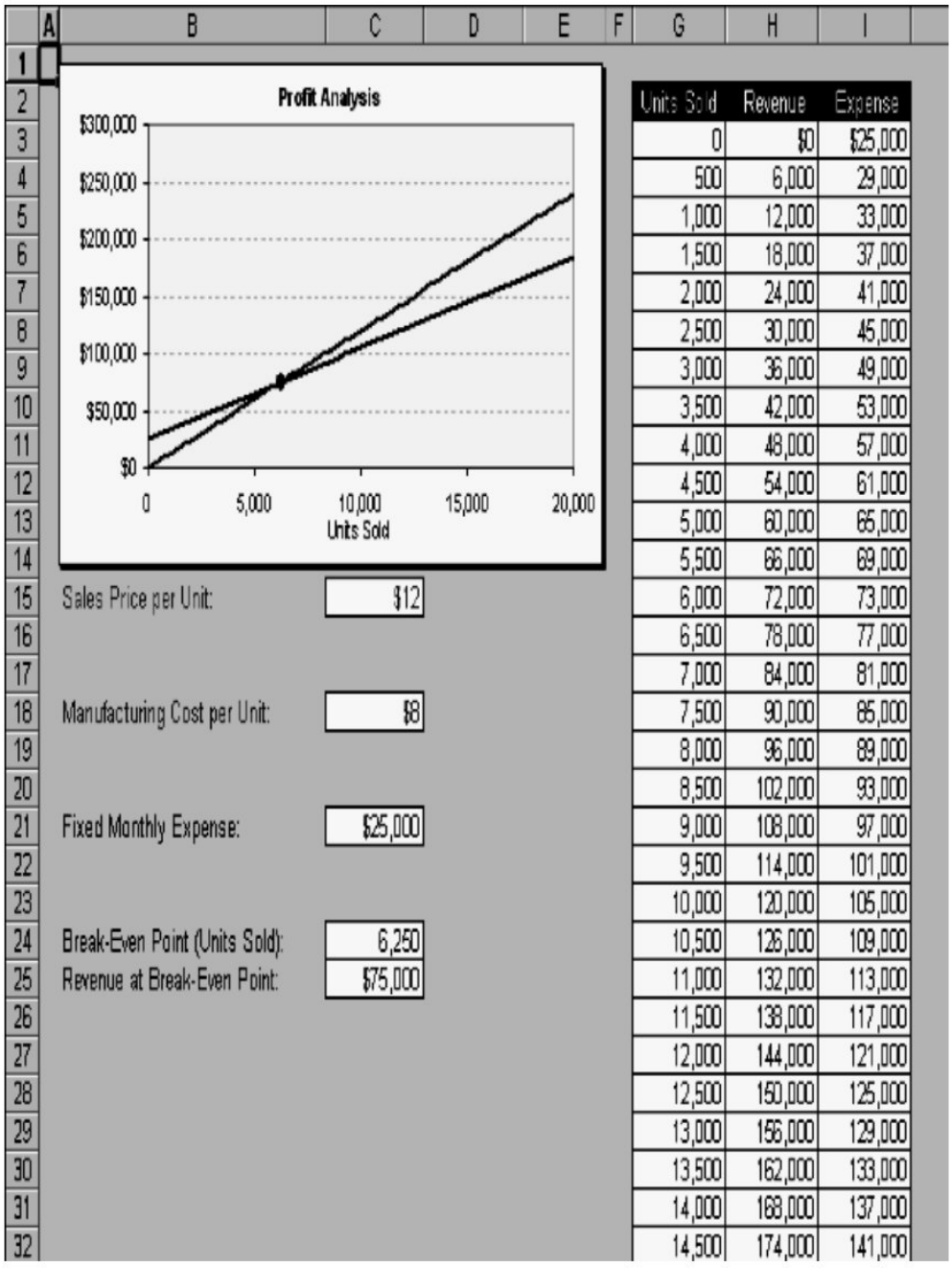
- How likely is the current plan to come in on schedule or on budget?
- How much contingency reserve of time or money is needed to provide the agency with a sufficient degree of certainty?

• Using sensitivity analysis, which activities or line-item cost elements contribute the most to the possibility of overrunning schedule or cost targets?

4.7.3 Sensitivity Analysis:

- Sensitivity analysis is a technique used to show the effects of changing one or more variables on an outcome.
- For example, many people use it to determine what the monthly payments for a loan will be given different interest rates or periods of the loan, or for determining break-even points based on different assumptions.
- Spreadsheet software, such as Excel, is a common tool for performing sensitivity analysis.

The following figure shows an example Excel file created to quickly show the break-even point for a product based on various inputs-the sales price per unit, manufacturing cost per unit, and fixed monthly expenses. The current inputs result I a break-even point of 6,250 units sold. Users of this spreadsheet can change inputs and see the effects on the break-even point in chart format. Project teams often create similar models to determine the sensitivity of various project variables.



The main outputs of quantitative risk analysis are updates to the risk register, such as revised risk rankings or detailed information behind those rankings. The quantitative analysis also provides high level information in terms of the probabilities of achieving certain projects objectives. This information might cause the project manager to suggest changes in contingency reserves .

4.8 RISK RESPONSE PLANNING

Risk Response Planning is the process of developing options, and determining actions to enhance opportunities and reduce threats to the project's objectives. It focuses on the high-risk items evaluated in the qualitative and/or quantitative risk analysis. In Risk Response Planning parties are identified and assigned to take responsibility for each risk response. This process ensures that each risk requiring a response has an owner monitoring the responses, although a different party may be responsible for implementing the risk handling action itself.

The project manager and the PDT identify which strategy is best for each risk, and then design specific action(s) to implement that strategy.

Strategies for Negative Risks or Threats include:

 Avoid: Risk avoidance involves changing the project plan to eliminate the risk or to protect the project objectives (time, cost, scope, quality) from its impact. The team might achieve this by changing scope, adding time, or adding resources (thus relaxing the so-called "triple constraint").

These changes may require a Programming Change Request (PCR). Some negative risks (threats) that arise early in the project can be avoided by clarifying requirements, obtaining information, improving communication, or acquiring expertise.

 Transfer: Risk transference requires shifting the negative impact of a threat, along with ownership of the response, to a third party. An example would be the team transfers the financial impact of risk by contracting out some aspect of the work.

Transference reduces the risk only if the contractor is more capable of taking steps to reduce the risk and does so. Risk transference nearly always involves payment of a risk premium to the party taking on the risk.

Transference tools can be quite diverse and include, but are not limited to the use of: insurance, performance bonds, warranties, guarantees, incentive/disincentive clauses, A+B Contracts, etc.

 Mitigate. Risk mitigation implies a reduction in the probability and/or impact of an adverse risk event to an acceptable threshold. Taking early action to reduce the probability and/or impact of a risk is often more effective than trying to repair the damage after the risk has occurred.

Risk mitigation may take resources or time and hence may represent a tradeoff of one objective for another. However, it may

still be preferable to going forward with an unmitigated risk. Monitoring the deliverables closely, increasing the number of parallel activities in the schedule, early involvement of regulatory agencies in the project, early and continuous outreach to communities/advocacy groups, implementing value engineering, performing corridor studies, adopting less complex processes, conducting more tests, or choosing a more stable supplier are examples of mitigation actions.

General Risk Mitigation Strategies for Technical, Cost, and Schedule Risks

Technical Risks	Cost Risks	Schedule Risks
Emphasize team support and avoid stand-alone project structure	Increase the frequency of project monitoring	Increase the frequency of project monitoring
Increase project manager authority	Use WBS and CPM	Use WBS and CPM
Improve problem handling and communication	Improve communication, project goals understanding, and team support	Select the most experienced project manager
Increase the frequency of project monitoring	Increase project manager authority	
Use WBS and CPM		

Strategies for Positive Risks or Opportunities include:

 Exploit. The organization wishes to ensure that the opportunity is realized. This strategy seeks to eliminate the uncertainty associated with a particular upside risk by making the opportunity definitely happen. Examples include securing talented resources that may become available for the project.

 Share. Allocating ownership to a third party who is best able to capture the opportunity for the benefit of the project. Examples include: forming risk-sharing partnerships, teams, working with elected officials, special-purpose companies, joint ventures, etc.

 Enhance. This strategy modifies the size of an opportunity by increasing probability and/or positive impacts, and by identifying and maximizing key drivers of these positive-impact risks. Seeking to facilitate or strengthen the cause of the opportunity, and proactively targeting and reinforcing its trigger conditions, might increase probability. Impact drivers can also be targeted, seeking to increase the project's susceptibility to the opportunity.

 Acceptance. A strategy that is adopted because it is either not possible to eliminate that risk from a project or the cost in time or money of the response is not warranted by the importance of the risk. When the project manager and the project team decide to accept a certain risk(s), they do not need to change the project plan to deal with that certain risk, or identify any response strategy other than agreeing to address the risk if and when it occurs. A workaround plan may be developed for that eventuality.

There are two types of acceptance strategy:

1. Active acceptance. The most common active acceptance strategy is to establish a contingency reserve, including amounts of time, money, or resources to handle the threat or opportunity.

2- Passive acceptance. Requires no action leaving the project team to deal with the threats or opportunities as they occur.

i. **Workaround**:

Workaround is distinguished from contingency plan in that a workaround is a recovery plan that is implemented if the event occurs, whereas a contingency plan is to be implemented if a trigger event indicates that the risk is very likely to occur.

As with risk identification process, the team should also consider residual risks, secondary risks, and risk interaction in the risk response planning process. See page 10 for details.

4.9 RISK MONITORING AND CONTROL

Risk monitoring and control keeps track of the identified risks, residual risks, and new risks. It also monitors the execution of planned strategies on the identified risks and evaluates their effectiveness.

Risk monitoring and control continues for the life of the project. The list of project risks changes as the project matures, new risks develop, or anticipated risks disappear.

Typically during project execution there should be regularly held risk meetings during which all or a part of the Risk Register is reviewed for the effectiveness of their handling and new risks are discussed and assigned owners. Periodic project risk reviews repeat the process of identification, analysis, and response planning. The project manager ensures that project risk is an agenda item at all PDT meetings. Risk ratings and prioritization commonly change during the project lifecycle.

If an unanticipated risk emerges, or a risk's impact is greater than expected, the planned response may not be adequate. The project manager and the PDT must perform additional response planning to control the risk.

Risk control involves:

- Choosing alternative response strategies
- Implementing a contingency plan
- Taking corrective actions
- Re-planning the project, as applicable

The individual or a group assigned to each risk (risk owner) reports periodically to the project manager and the risk team leader on the status of the risk and the effectiveness of the response plan. The risk owner also reports on any unanticipated effects, and any mid-course correction that the PDT must consider in order to mitigate the risk.

4.10. USING SOFTWARE TO ASSIST IN PROJECT RISK MANAGEMENT

Most organizations use software to create , update , and distribute informations in their risk registers.The risk register is often a word or excel file but it can also be part of a more sophisticated database. Spreadsheets can aid in tracking and quantifying risk , preparing charts and graphs , and performing sensitivity analysis. Software can be used to create decision trees and estimated monitary values.

More sophisticated risk management software such as Monte Carlo Simulation s/w can help you develop models and use simulations to analyze and respond to various risks. There are also several s/w packages created specifically for project risk management . If a risk is not identified.

Software should be used as a tool to help make good decisions in project risk management, not as a scapegoat for when things go wrong.

4.11 SUMMARY

Risk Management is always forgotten when managing projects but the irony is that all projects have risk. People in general think that risk management is just a blaming session to uncover flaws in a particular project. This perception has to be abolished.

Management and Project managers have to understand that Risk Management is the one of the few practical way to manage uncertainties and doubts towards a particular project.

Risk can never be abolished, but can only be reduced to an acceptable level. Risk Management is a must for any projects and it has to be done from the initiation phase throughout the project lifecycle. Risk Management is not free, and it isn't cheap. There may need to have third party audits which incur cost. There must always be continual management support and commitment to ensure the success of projects.

This chapter we discussed the importance of risk management in the projects and also were able to understand the different processes in the risk management, which consists of the following actions

 Project risk management is the art and science of identifying, analyzing, and responding to risk throughout the life of a project and in the best interests of meeting project objectives.

Main processes include:

- Risk management planning
- Risk identification
- Qualitative risk analysis
- Quantitative risk analysis
- Risk response planning
- Risk monitoring and control

Sample Questions

1. Discuss the common sources of risk on information technology projects and suggestions for managing them**. (Ans:section 4.4)**

2. Explain how to use decision trees and Monte Carlo analysis for quantifying risk. **(Ans Hint: section 4.7.1)**

Suggested Readings:

1. Boehm ,Barry W. " Software Risk management: Principles and practices"
2. Kathy Schwalbe : Information Technology project management
3. Hillson , David. : The risk breakdown structure as an aid to effective risk management
4. DeMarco,Tom and Timothy Lister: Managing Risk on software projects

GLOBALIZATION IMPACT ON PROJECT MANAGEMENT

1. Introduction

We live in a projectified society. The term coined in the mid-1990s has been gaining relevance ever since, as projects permeate not just economies, but nearly all spheres of human society. From the 'traditional' project-based industries, such as construction (serving the earliest examples of projects – Kozak-Holland 2010; Morris 2013; Chiu 2010), and later defense (with the US Department of Defense and 'McNamara's Revolution widely recognized as the cradle of classic project management – Lenfle and Loch 2010; Morris 2013), projects as organizational form spilled over to other economic spheres, including the rather novel ones (notably IT and information systems), as well as the public sector (Crawford and Helm 2009; Godenhjelm, Lundin, and Sjoblom 2015), education (Austin *et al.* 2013) and scientific research (Fowler, Lindahl, and Sköld 2015), healthcare (Shirley 2020; Suhonen and Paasivaara 2011), media (Bouncken, Lekse, and Koch 2008). It is not uncommon to see people describing their nonwork-related activities (hobbies, sports, studies and many other things) in 'project' terms (be it making paintings, working on some elaborate embroidery, going in forlearning to cook recipes from a specific cuisine, reaching a particular fitness or sports goal, *etc*.).

As regards the extent of projects in the developed economies, already in the 1992–1996 a massive panel survey of 3,500 European firms highlighted a marked increase in the use of projects in firm activity – up from 13 to 42 per cent over the course of just four years (Whittington *et al.* 1999). In 2004, a survey by PricewaterhouseCoopers provided new evidence supporting the trend – each of the 200 firms in the sample was engaged in some form of project activity, while a quarter of them had large (100+) project portfolios (Nieto-Rodriguez *et al.* 2004). Scranton (2014) used World Bank data for the estimate of about 22 per cent of world GDP in 2009 came from project-based activities, with some countries like India and China having much higher estimates (34 % and 45 % respectively). A more recent study of Germany, Norway and Iceland revealed that the shares of project work in total working hours reached as high as 34.7 %, 32.6 % and 27.7 % respectively in 2014–2017, with the majority of projects being internal and thus largely invisible to the customer (Schoper *et al.* 2018). With these figures taken into account, expressions like 'projectification of everything' (Jensen *et al.* 2016: 22), or statements on the omnipresence of projects and their becoming a human condition (Lundin *et al.* 2015; see also Lundin and Söderholm 1998; Hobday 1998, 2000; Cicmil and Hodgson 2006) do not sound as an exaggeration.

Increasing importance of projects and their spread into various spheres of economy, as well as into social and political activities, thus appears to be an established trend of the latest decades. Is has been running simultaneous to a number of major changes in the life of humanity, of which globalization is likely among the first to come to mind. This raises a question of globalization impact on the essential features of projects and the sphere of project management.

2. Globalization, Technological Paradigms, and Project Management: Concept Evolution

(2002), Richard R. Nelson (2008), Nick von Tunzelmann (von Tunzelmann *et al.* 2008), Leonid Grinin and Andrey Korotayev (Akaev *et al.* 2012; Grinin L., Grinin A., Korotayev 2017; Grinin, Korotayev 2016). The K-waves are considered to have emerged in the economy with the Industrial Revolution, which fundamentally changed the means of production. Each new cycle would be started by a number of breakthrough technologies forming a new technological paradigm (the spread of these technologies allows the emergence of new spheres of production, thus causing an economic upswing). Their impact on globalization differs from one technology to the other; it was shown, for example, that the 'golden age' of the nineteenth-century globalization was largely related to the technological breakthroughs forming the essence of the second technological paradigm, such as the telegraph, railways and steamships, which allowed for a rocketing increase in the speed and volume of communications over distance, most notably between Europe and Americas (Zinkina *et al.* 2019).

The fifth technological paradigm, which, according to researchers, is currently completing its final stage, is based on technological breakthroughs in microelectronics (invention and worldwide spread of personal computers), information technologies (the Internet, email, social networks, instant messengers, *etc*.) and various software that have created new industries and changed the already existing ones. These technologies allowed for a dramatic increase in the world connectivity, which formed the essence of the most recent decades of globalization. In addition, the fifth technological paradigm witnessed the development and massive use of cyber-physical systems, that is, systems encompassing complex interconnections of computing devices, physical objects, people and physical environments. One would naturally expect such profound changes to have had reflection in the nature of projects and project management. To understand them better, let us take a brief look at the history of project management and classic style of project management developed in the twentieth century.

Historians of project management believe that the first projects emerged in construction. The largest and best-known examples would include Egyptian pyramids, Mesopotamian temples, the Great Chinese Wall (to name a few), but these 'megaprojects' co-existed with smaller ones needed in everyday life such as construction of temples, roads, bridges, and irrigation systems (Kozak-Holland 2010; Chiu 2010; Morris 2013). The Hammurabi code of laws includes a clause setting legal responsibility for the architect and construction workers if their building collapsed (Chiu 2010: 170). The construction of Egyptian pyramids involved such basic project management practices as keeping accounts of materials, labor resources, and money spent. For a long time in history, project organization was mostly applied for the tasks of constructing buildings and infrastructure (Hall 2012; Garel 2013).

The first turning point in the history of project management was likely associated with the Industrial Revolution. As the range of projects increased, so did the complexity of management technologies; the hierarchy of 'general contractor – contractor – subcontractors' is widely established, the sphere of responsibility is defined for each level in this hierarchy, control for the project timing and the quality of the result is getting formalized (Chiu 2010: 201). Railways, mining, petrochemicals, medicine, cargo transportation,

banking were all actively developing using the form of projects, although formal positions and roles of project manager, chief project engineer, project coordinator were only just beginning to appear (Morris 2013: 19). At the same time, the first

Globalization receives unabating attention not only from the research community, but from numerous decision-makers, state leaders, entrepreneurs, media and general public all over the world. Multi-dimensional as it is, this phenomenon invariably impacts nearly all spheres of human life, even though the extent and character of its influence can differ. Globalization has had a significant effect on global economy, global financial system, politics, statehood, society, *etc*. (Grinin 2011). There are different views regarding the history of globalization. Sometimes it is associated with the spread of PCs, the Internet, and the huge increase in global communications and other types of connection; another line of research puts globalization in the context of Macrohistory/Big History, tracing its start back to ancient times and viewing the recent surge of globalization as yet another stage in its long history which has seen both periods of high integration and periods of weakening ties between various parts of the world (Zinkina *et al.* 2019; Grinin 2011). Some researchers claim the most recent decades of globalization to be a fundamentally new phenomenon, others see it as an intensification of the previously existing trends and connections. However, regardless of the approach we take, it can hardly be disputed globalization

has been closely interrelated with technological change.

The concept of technological paradigms, closely related to the concept of long economic cycles / Kondratieff waves (1922), was developed in the works of Carlota Perez

transnational corporations appeared which had to develop the first tools for trans-border project management, heavily relying on the most advanced transport and communication technologies of the time (Wilkins 1988).

Researchers differ in their opinions on the emergence of modern project management. Sometimes it attributed to the ‘taylorism’ at the very beginning of the twentieth century. More frequently, the Gantt Chart of 1910 is recalled, which formally set one of the ‘golden triad’ of modern project management indicators – timing (concentrating on the timing of project tasks and sub-tasks and how long they took to be completed). Both approaches were frequently used before the World War II, which saw a surge in the number of projects and their variety. The most controversial case of the time was the Manhattan project. Some see it as the cradle of classic project management where its principles of organization, planning, and control were developed (Shenhar and Dvir 2007). Others emphasize that the principles underlying the greatest part of Manhattan project (trial-and-error, parallel trials) got excluded from classical project management, and its very essence contradicted these principles (Lenfle and Loch 2010). The very term of ‘project management’ appeared after the War, in 1952–53, in the USA Air Force, followed by a formal position of project manager, the concept of project office and other important concepts (Morris 2013). Classical project management was to a large extent developed in the course of two military projects, Atlas and Polaris (the latter is more known for the development of PERT tool for network calendar planning), and then got widely accepted through the so-called ‘McNamara Revolution’. While at his post of the Secretary of Defense, Robert

McNamara transformed the processes of analysis, planning, decision-making and project management. First, the phased approach has become the main project management model for the Department of Defense and the newly created NASA. In the evaluation procedures, special attention was paid to the stages of developing the concept and determining the project contract. This was supported by the proliferation of management tools such as PERT. Secondly, starting from 1963, the Department of Defense moved from cost-recovery contracts (plus a flat-rate fee) to fixed-price contracts, which increased the responsibility of contractors to achieve the goals of the project while strictly sticking to the pre-planned budget. This helped consolidate project planning and control as key elements of project management. The two key assumptions that underlie the phased approach can be identified as follows: (1) project management is focused exclusively on implementing the decisions, not making or discussing them; (2) uncertainty management and control are feasible. Thus, classical project management was limited to the effective implementation of routine initiatives and tasks, being cut off from the two main areas of management inherent in the Manhattan project – strategy development and strategic search (Lenfle and Loch 2010).

3. Data and Methods

The last decades witnessed a growing failure rate in projects, varying from one sphere to another but particularly high in the new spheres, such as IT/IS, where in the mid-1990s more than a half of projects failed to meet at least one success criteria of the 'golden triad' of classic project management (time, budget, and scope), and about onethird were considered complete failures (Ewusi-Mensah and Przasnyski 1997). Projects would frequently overrun their schedules, require additional financial resources, get cancelled before completion or survive to be completed but fail to produce a good/service popular with customers. These issues caught the attention of a number of researchers trying to understand and conceptualize the reasons behind the project failures and to transfer this knowledge into methodological improvements aimed at fixing the most commonly experienced problems. We review this literature to reveal the most frequently mentioned mismatches between the classical project management and modern projects. We then proceed to analyze the role of globalization in the emergence of these mismatches. We use areas of project management knowledge developed in the Guide to the Project Management Body of Knowledge developed by Project Management Institution in order to classify the channels of globalization impact on project management.

4. Impact of Globalization on Project Management

Globalization has been influencing project management through a large number of channels. In order to classify and organize this variety, let us follow the structure proposed in the Guide to the Project Management Body of Knowledge (PMBOK Guide) developed by Project Management Institution. This classification includes the following categories: project environment; project leadership; managing project integration; managing project scope; managing project time; managing project cost; managing project quality; managing human resources of the project; managing project communications; managing project risk; managing project procurement; and managing project stakeholders. Below we will list the channels of globalization impact on project management structured according to these categories. The list does not claim to be comprehensive and only contains the most notable effects of globalization on the corresponding areas of project management.

- project environment – globalization exerts a huge impact upon the environment where the projects are carried out, first and foremost, by increasing its volatility and penetrability to external shocks. A vivid example from rather recent past can be presented here, namely, the global financial and economic crisis of 2008–2009. What started as mortgage market crisis in the USA quickly evolved into a full-scale global economic recession, causing a slowdown, or a temporary halt, or even a cancellation in numerous projects worldwide. The impact of the global crisis was of such scale that these slowdowns and cancellations frequently could not be averted or mitigated by regular project management. Globalization-driven environmental shocks to project management can come in a variety of forms, such as global financial and economic crises (as in the example viewed above), commodity price spikes, sudden sociopolitical shocks and destabilizations (including ones caused by external influence), changes in the international political arena, technological breakthroughs made in other countries *etc.*
- project leadership – much depends on the educational and professional background of the leader, including their involvement in the global sphere of knowledge related to the project managed, as well as the presence or absence of international connections which could ease communications with foreign colleagues, knowledge of foreign competitors and their experience, access to international funding *etc.*; 'globalized' leaders may have a number of important advantages here;
- managing project integration involves project planning, project knowledge management, monitoring, and integrated change control; at this stage it is important for project managers to take into account the possible impact of globalization upon the project scope, time, and cost – see below. Integrated change control is related to monitoring potential risks and reacting properly if one or several of the risks become practically relevant – see below 'managing project risk'; international laws and regulations applied to global projects must be well understood in order to effectively manage the relationship of the professional practices to governments and political conditions as well as build key relationship to global government decision-makers;
- managing project scope – project scope should be defined taking into account the global sphere of knowledge related to the project managed; on the one hand, globalization of knowledge makes it easier to plan the scope of the project as a lot of details can be clarified through previous experience and some potential pitfalls can be avoided, both for deterministic and non-deterministic projects; on the other hand, it makes it more difficult, as a large amount of knowledge needs to be taken into account for project success; thus, high-qualified human resources are necessary. For non-deterministic projects, globalization of knowledge is of particular importance, as it serves for defining the project goal itself and the project can be carried out 'standing on the shoulders of the giants';
- managing project time – this aspect used to be much more transparent at the project planning phase in the epoch of classical project management and can be affected by a multiplicity of various factors in the epoch of globalization (see 'project environment'); project overruns are currently typical and have been so for a while, reaching 50 per cent – 200 per cent of the planned project time (Morris and Hough 1987: 7;

Reichelt and Lyneis 1999); we presume that this is due to uncertainty and complexity rising with globalization – see below Section 5;

- managing project cost – on the one hand, globalization presents some opportunities to carry out projects at smaller costs, for example, economizing on human resources through outsourcing tasks to different countries with cheaper labor force, or saving on the cost of materials by gaining access to information on cheaper producers and reaching out to these very producers through modern information technologies; on the other hand, globalization makes projects vulnerable to commodity price shocks and, generally speaking, to all the global economic and financial volatility, including crisis phenomena; moreover, increased price awareness of customers' needs to be taken into account when defining the costs of project and the price of project output;
- managing project quality – global quality standards regarding the output of the project need to be taken into account for project success; globalization increases international competition for project output quality;
- managing human resources of the project – one of the prominent challenges posed by globalization for project managers is managing virtual project teams consisting of outsourced members; advantages stemming from globalization here include 'access to a wider pool of talent, potential cost reductions by cheaper labor in developing countries, the enforcement of internal competition and possible quality improvements. External advantages for customers are follow-the-sun development and extended service times' (Eberlein 2008: 29; Gurung and Prater 2006). More challenging aspects also stemming from globalization include the need to address significant cultural heterogeneity within the virtual teams, which tends to rise with the increase in the number of sites involved. Mismanagement of cultural differences and intercultural miscommunications can possibly lead to project failure.
- managing project communications – this aspect could be understood in two aspects, namely communication as literal technically enabled interaction and communication as transmission of messages. For the first aspect, project communications are largely sustained with ICT technologies of the fifth Kondratieff cycle underlying the most recent wave of globalization; globalization increases the importance of communication among the members of the project team situated at different places, quite often remote from each other. For the second aspect, it is not the physical distance between the team members that matters, but rather cultural distance. Indeed, project members may belong to different organizational cultures and different cultures in general, and same notions may have different implications for them, and same questions may retrieve very different answers (Shulgin *et al.* 2017). Failure to take into account intercultural differences may lead to miscommunications and eventual project failure.
- managing project risk – project risk management in a globalizing society turns out to be a much more versatile task than classical project management ever implied. This largely has to deal with the two key notions of uncertainty and complexity – uncertainty relates to the external conditions wherein the project is carried out, while complexity relates both to the external environment and to the structure of a project itself, as well as the integration of project into the environment (see Section 5 below). Risks

are generally reflected in project time (project failure due to untimely delivery of intermediate or final results), project costs (failure due to overrunning the budget), and project scope (partial or full incompletion of the project). Causes of the risks are multiple and a huge number of them are related to globalization, be it financial (sudden limitation or abrupt ending of financial resources due to global price hikes or global financial dynamics or economic shocks), technological (global appearance of new technologies that make technologies used in the project of the very goal of the project obsolete), cultural (managing successful intercultural communication within a multi-cultural project team, or securing successful communication with multi-cultural society of consumers), political (accommodating for global geopolitical changes affecting the project course and landscape), legal (accommodating for international legal provisions affecting the project course and landscape) *etc*.

- managing project procurement – global procurement knowledge and skills are needed; global procurement is influenced by price fluctuations at global commodity markets, as well as dealing with international laws;
- managing project stakeholders – one of the challenges posed here by globalization is the necessity to manage the interests of stakeholders probably belonging to different cultures (same as with project human resources). Advantages here again include the possibility to build on a richer variety of cultures and knowledge; however, mismanagement of cultural differences and intercultural miscommunications can possibly lead to project failure.

Below we will try to summarize the challenges posed by globalization for various areas of project management and changes brought to project management by globalizing environment.

5. Research and Discussion

One of the fundamental differences between classical project management and modern projects lies in the sphere of planning and the extent of pre-determination. As mentioned above, classical project management would imply that uncertainty management and control are feasible; thus, it was well suited for projects with clearly pre-defined goals and methods and a detailed pre-defined plan, realized in rather stable environment that was unlikely to require any changes to initial project features in the course of its implementation. However, with modern projects one would rather expect this not to be the case. In this line of thought, Hall (2012) distinguishes between deterministic and non-deterministic projects (the ones with and without a detailed and exact outline of the project goal preceding the start of the project).

Another important point touches on dealing with innovations in methods. Planning is one of the pillars of classical project management and, indeed, a cornerstone of phased approach, as the manager basically controlled that project implementation strictly followed the initially developed plan. This approach is suitable for projects where technologies and environment are unlikely to experience significant changes in the course of project implementation. However, it will hardly be efficient in rapidly changing spheres, especially ones close to the technological frontier, such as IT/IS, biotechnology, some branches of medicine and pharmaceutical industry, *etc*. Here, sticking to the initial plan and refusing any changes in it may lead to overlooking important technological innovations emerging during the project implementation; this, in turn, may

make the project results obsolete and unable to stand competition in the market – sometimes before they are even fully obtained.

Let us view another closely related point, namely, the technological uncertainty. Classical project management and modern project are nearly opposite in their stance in relation to uncertainty. The first implies that there is little to no uncertainty, which makes project activities and schedule easy to plan; the latter frequently exist in very high degrees of uncertainty, to the point that methods of the projects are very vaguely understood by the time it starts, or turn out to be non-existent yet (look at the development of a new pharmaceutical formula for a drug that would cure a previously incurable disease or condition). This is where classical project management is reasonably set aside in favor of parallel trial and trial-and-error approaches, which were so important in Manhattan project (see above). In terms of project goals, classical management is set to work with a clearly outlined goal, where scope and specifications are clearly known before the beginning; a parallel trial and trial-and-error work are better for a goal which has yet to be specified in the course of the project, as initial knowledge is insufficient for its detailed outline because it is innovative in its kind and cannot build upon similar results of earlier projects. As Lenfle and Loch (2010) put it, in such projects the detailsof goal description can be seen as hypotheses that need to be tested and supported/ rejected by trials in the course of the project.

Related to uncertainty, but in no way equal to it is the concept of complexity. Albert Hirschman (1967) was among the first researchers to view projects as systems. In this view, the complexity of the project is coming not from its size and the numbers of parts, but rather from the necessity to develop methods for coordinating these parts into a single whole. In modern research, a number of approaches have been developed to the problem of classifying projects depending on their levels of complexity (and, of course, conceptually defining these levels). Hobday (1998, 2000) specifies four levels of complexity, from extremely complex to simple projects (depending on such variables as the number of parts and components, complexity of systemic architecture, range and depth of required knowledge and skills, as well as variability of required materials and information). He notes that technical progress and new industrial requirements significantly widened the functional capacity, spread, and productivity of complex industrial products and systems, and projects aimed at their creation are increasing in complexity. Similar logic leads Shenhar and Dvir (2007) to single out three types of projects: simple (assembly of systemic parts), systemic (creation of systems), and massive (creation of metasystems).

However, these approaches limit project complexity to the internal features of the projects. To take into account complexity generated by external factors, we can turn to TOE approach comprising technical, organizational, and environmental complexity. Technical complexity was largely described above; organizational complexity deals with resource availability and human factors, while environmental complexity looks at project location, market conditions *etc*. (Bosch-Rekveldt *et al.* 2011). Another noteworthy approach is to distinguish between structural and dynamic complexity. The importance of accounting for structural complexity in project management was first noted by Baccarini (1996), who sub-divided it into organizational and technological complexity. Somewhat later, Williams (1999) pointed at complexity caused by uncertainty in project methods and project goals. Brady and Davies (2014) used the structuraldynamic

dichotomy to generalize the existing knowledge on complexity types and their implications for project management. Structural complexity includes hierarchy of systems and interdependencies of project components, system integration, and interactions between team members. Foreseen and unforeseen uncertainty, innovations, market changes are subsumed under dynamic complexity. In other words, structural complexity, according to Brady and Davies, is related to the interactions within the project, while dynamic complexity is engendered in the interactions between the project and its environment.
A somewhat different approach is taken by Geraldi, Maylor, and Williams (2011), who differentiate between not two but five types of complexity: structural (technological and organizational), uncertainty-related (goals, methods), time-related, dynamic, and sociopolitical (legal issues, security issues, ethical issues *etc*.).
As for the impact of globalization on projects and project management, which is frequently viewed through the prism of particular changes posing new challenges for the managers (such as the development of outsourcing supported by the ICT innovations of the fifth technological paradigm and their spread into developing countries withcheaper labor force), it can be proved to be much more fundamental when the issues of uncertainty and complexity are taken into account.

6. Conclusion

Thus, in terms of uncertainty, globalization has clearly been making the environment where the projects are realized more uncertain in various aspects. As regards internal uncertainty (arising in the interactions within the project), one can single out an increase in technological uncertainty, which makes it harder to define project methods and goals before its start. However, it is far from being the only effect of globalization on project management. An increase in organizational uncertainty can also largely be attributed to globalization (with newly arising challenges of managing multinational teams, outsourced tasks, *etc.*). An even greater increase can be observed in the uncertainty of the environment and various external conditions under which a project is being realized.
Thus, globalization has boosted the extent of market interconnection for the majority of goods and services. Interconnection brings with it increasing vulnerability to global crises; this was clearly visible during the 2008–2009 global financial and economic crisis, when the repercussions of what started as a downfall of the USA mortgage banking were eventually felt in almost all countries of the world. Moreover, interconnection makes national markets very sensitive to price fluctuations in global markets, and this sensitivity, in turn, may influence national sociopolitical stability – not to touch on the more obvious matter of global perspective increasing competition in the markets, which also makes it harder to set a truly competitive project goal, manage the project efficiently, and obtain a successful and competitive result. If a project is implemented on a multi-national scale, its complexity is likely to reach a whole next level, as project managers are to simultaneously take into account the changes in legal sphere, political leanings, sociopolitical stability, investment climate, markets, and numerous other aspects in a number of countries.
Even a rather simple assembly-type project will have its complexity increase due to growing uncertainty in the growing number of aspects – goals (how do we make our goal competitive?), time (will the product/service still be relevant by the time the project is complete?), methods (what are the best technologies, knowledge, skills to be used in the development of the product/service, and are they existent yet? are they likely

to change while the project is implemented?), relevance (is the result likely to become obsolete due to fast technological progress, even if the best knowledge in this particular sphere is used?), and numerous external conditions in markets, potential customers, technological development, law, sociopolitical stability, *etc*. These factors increase both the uncertainty and complexity of contemporary projects and appear to be two main factors channeling globalization's impact on projects and project management.

Project management was still in the cradle during the ‘golden epoch’ of early globalization in the late nineteenth century and up to World War I. Classical project management took several decades for its tenets to be properly developed, but its spread was to a large extent provided by new technologies of the fourth technological paradigm – namely, the first generation of large-scale computer-system architectures (mainframe computers). The newest surge of globalization, to which many researchers, practitioners, and members of general public attribute the term and concept of globalization itself, has been heavily relying on the ICT technologies of the fifth technological paradigmwhich greatly increased the degree of global interconnectedness and brought about fundamental changes in many spheres of economy – project management being no exception.

The ‘golden triad’ of success measurement in classical project management (timebudget-scope) has largely lost its relevance to project success. On the one hand, more projects tend to break at least one condition of the triad – still, their results may be deemed successful. On the other hand, strict sticking to the initial schedule, budgeting, and scope is no guarantee of success for the result of the project. We show that this is to a great extent related to the changes brought about by globalization, which can be mostly subsumed under increasing uncertainty and complexity. While rapid technological changes and moving technological frontiers definitely contribute to both, they are not exclusively responsible for the whole increase in uncertainty and complexity, as numerous other aspects both within the projects and outside (in the environment where the projects are implemented) have become more uncertain and more complex, and things are not going back to the simpler way of life.

M.A. M. COLLEGE OF ENGINEERING, TRICHY

DEPARTMENT OF MCA

SOFTWARE PROJECT MANAGEMENT

UNIT - I

INTRODUCTION TO SOFTWARE PROJECT MANAGEMENT

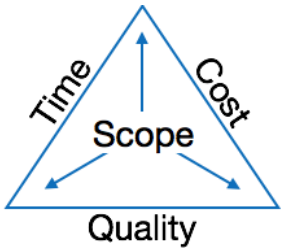
- ✓ The job pattern of an IT company engaged in software development can be seen split in two parts:
  - Software Creation
  - Software Project Management
- ✓ **A project** is well-defined task, which is a collection of several operations done in order to achieve a goal (for example, software development and delivery).
- ✓ A Project can be **characterized** as:
  - Every project may have a **unique and distinct goal**.
  - Project is **not routine activity** or day-to-day operations.
  - Project comes with a **start time** and **end time.**
  - Project ends when its goal is achieved hence it is a temporary phase in the lifetime of an organization.
  - Project **needs adequate resources** in terms of time, manpower, finance, material and knowledge-bank.

Software Project

- ✓ A Software Project is the **complete procedure of software development** from requirement gathering to testing and maintenance, carried out according to the execution methodologies, in a specified period of time and cost to achieve intended software product.

Need of software project management

- ✓ Software is said to be an intangible product. Software development is a kind of all new stream in world business and there's very little experience in building software products.
- ✓ Most software products are tailor made to **fit client's requirements**. The most important is that the underlying technology changes and advances so frequently and rapidly that experience of one product may not be applied to the other one. All such business and environmental constraints bring risk in software development hence it is essential to manage software projects efficiently.
- ✓ The image above **shows triple constraints** for software projects. It is an essential part of software organization to deliver quality product, keeping the cost within client's budget constrain and deliver the project as per scheduled.



- ✓ There are several factors, both internal and external, which may impact this triple constrain triangle. Any of three factors can severely impact the other two.
- ✓ Therefore, software project management is essential to incorporate user requirements along with budget and time constraints.

What Is Software Development Project Management?

- ✓ Project management in software development is the process of planning, scheduling, executing, monitoring, and delivering software projects.
- ✓ Ssoftware development projects can be better seen as complex undertakings, where **leaders must analyze** the cost-benefits and optimization problems between business value and the software development pipeline and processes.
- ✓ In an ever-evolving landscape where business management and tech are, essentially, both close family relatives, software development projects are complex tasks led by two or more persons. Bounded by time, budget, and staffing resources to produce novel or enhanced computer code with resource allocation and execution in mind.
- ✓ More generally, software projects are defined by a comprehensive development pipeline, all the way from initial gathering to testing and maintenance, carried within a given timeline to achieve the intended final product.

Why Is Project Management Important in Software Development Projects?

- ✓ Software project management is important because it ensures that there is strategic method towards accomplishing software-related objectives. This **adds significant business value to new/existing business processes and models**. In practice, your business should see benefits such as:
  - A greater competitive advantage
  - Improved resource allocation
  - Tighter budgeting
  - Better communication
  - Clearer and more effective documentation
- ✓ In contrast to traditional project management models, software development is relatively unique, as software projects require a distinct lifecycle and phased development process that demands several rounds of user and internal testing, updating, and analysis of customer feedback.
- ✓ As a result, project management in software development is a crucial part of **delivering a quality product**, and no development model would be much without it.

What Does a Project Manager Do in Software Development?

- ✓ Project managers are **responsible** for efficiently executing all the details of the software project from start to finish.

- ✓ Most importantly, they are aware of all phases in the Software Development Life Cycle (SDLC) that the software project should undergo, streamlining the initial planning and long-term maintenance of the project at hand.
- ✓ By closely monitoring the project management in the software development process, experimenting with various plans, allocating resources/budget, and solidifying communication amongst the team, software project managers **ensure that final product goals are met under all constraints** while still maximizing customer satisfaction. All in all, project managers perform the following activities:

1. Project Estimation

- ✓ Perhaps the most vital aspect of the initial project planning phase, project size estimation is crucial in determining the cost, duration, and trajected efforts necessary for the project. These estimations include:
  - **Time:** Total projected time to complete the project and, if necessary, its subparts.
  - **Cost:** Summary of total expenses to develop the software product.
  - **Effort**: Estimated effort demanded to complete the project under any given conditions and constraints.
- ✓ As a project manager, it is crucial that you accurately specify these time, cost, and effort estimations, as all future planning and executions are dependent on these projections.

2. Scheduling and Resource Allocation

- ✓ In the context of project management in software development, project managers **should schedule and allocate all required 'manpower' and resources** for software project development after finalizing the estimations above.

3. Staffing

- ✓ Establish and **decide on a team structure** and staffing plan. **Gain feedback** from the team on work burdens and their task progress to reallocate/reorganize the staff accordingly.

4. Risk Management

- ✓ As a software project manager, you must **identify and analyze any unanticipated risks** that might arise during the project development lifecycle. Devise your own risk reduction method to minimize/eliminate potential risks and unintended consequences during software development.

5. Miscellaneous Planning

- ✓ During this stage, a project manager may establish several other plans including quality assurance, configuration management, etc. Additional planning is conditioned on project monitoring and control, a cycle where the PM actively diagnoses potential risks and obstacles during development and uses planning to resolve those risks.

Most Important Skills a Project Manager Needs to Succeed

1. Leadership

- ✓ Naturally, anyone managing a team should **have leadership skills**. But what does be a strong leader mean? Strong leaders motivate their teams, coach them through task completion, and give them guidance on any issues that may arise.
- ✓ Much of leadership is psychological. Technically, everyone on the team knows what they're supposed to do. But setting actionable goals, measuring team performance, and giving constructive feedback makes a great impact on how the job gets done. And a project manager is responsible for these duties.

2. Communication

- ✓ Communication is an important skill in just about every industry. But it becomes increasingly important if you're leading a team of software developers.
- ✓ Project managers must have the technical expertise to communicate the details of the software project as well as the interpersonal skills to build and manage meaningful relationships with team members.

3. Time Management

- ✓ It should go without saying that time management is an essential part of the project manager role. Having a deadline isn't always enough and having someone behind the scenes to ensure that everyone's on task is often the missing gear in an otherwise slow-churning machine.

4. Organization

- ✓ Organization is perhaps the **most important skill for project managers.** But it's a broad term and various factors play into one's ability to organize, from communication skills to time management.
- ✓ Simply put, a project manager should be able to see the big picture and break that down into the tiny details that make up a software development project. They should also be able to map those details out into workable tasks.

5. Critical Thinking / Problem-Solving

- ✓ Often project managers are the first-responders when it comes to addressing bottlenecks and crises that pop up during development. In order to successfully resolve these issues, they must be quick on their feet and think outside of the box to come up with solutions.

Software Development Project Management Methodologies

- ✓ Software development methodologies **provide a structure for how you will go about building your software product**. These methodologies are also called project management methodologies as they serve as guidance for organizing your projects and delivering optimal performance.

1. Agile

- ✓ Agile is by far the most popular software development methodology there is. Hub Staff found that 36% of software development teams are using Agile, with almost 40% of teams using a hybrid solution of methodologies. The central points of the Agile method are iterative development and consumer feedback. Based on an Agile manifesto written in 2001, developers collaborate with customers as they build software, iteratively implementing and testing software adjustments as they go.

2. Scrum

- ✓ Scrum is an extension of Agile. It features a distinct system that helps developers approach the development process. Much of what happens on a srum team is managed via the concept of sprints, a short, time-boxed period allotted for completing a set amount of work. The Scrum methodology similarly includes sprint retrospectives, sprint reviews, sprint planning, and daily scrums.

3. Waterfall

- ✓ The waterfall methodology emphasizes a linear progression of development. It's merely a **step-by-step process of creating software from planning requirements to deployment**. However, the integrity of this model is also its downfall. Professional developers rarely use this model because it does not adapt well to customer input, changing requirement, or unforeseen circumstances.

4. Lean

- ✓ The Lean methodology stresses the optimization of resources within a business. It stems from an operational thinking strategy that gave Toyota Motor Company its success.
- ✓ In short, the fundamentals of the Lean methodology are **to eliminate waste and create value for the customer**. Within software development, Lean thinking means foregoing extra features and preventing delays whenever possible.
- ✓ Note that Lean and Agile are often confused. While some consider Lean to be a framework of Agile, comparing Lean vs. Agile reveals they are not as similar as some may think.

5. Feature-Driven Development

- ✓ Feature-Driven Development (FDD) is an Agile methodology for developing software. Like Agile, **FDD is iterative, incremental, and customer-centric**. As the name suggests, FDD involves developing software based on its features and creating feature-specific teams. Frequent status reports are also a mainstay in feature-driven development.

5 Stages of Project Management in Software Development

- ✓ Whether you're a junior or senior project manager, a common principle in the software development and management pipeline will always hold true: following and maintaining the Software Development Life Cycle (SDLC).

- ✓ By adapting the SDLC process to your project timeline, your project initiation, planning, monitoring, and closure will be streamlined, enabling the team to finish in time without sacrificing the integrity of the given project.
- ✓ Project management in software development essentially uses the general project management template with software-specific goals that streamline the process.

The 5 stages of project management in software development are outlined as follows:

1. Project Initiation

- ✓ It may seem intimidating, but do not worry—by establishing a **simple foundation of ideas and preliminary goals for your project,** you're bound to create a surefire template for the next four phases of your project management timeline.
- ✓ Project initiation involves morphing an abstract idea into a meaningful goal with actionable future steps. During project initiation, a PM should develop a business case and define the project on a broad level. Project managers can easily initiate this with a project charter.
- ✓ A project charter is a document consisting of critical details, including project constraints, goals, appointment of the project manager, expected timeline, budget(s), staffing, etc.
- ✓ Once a manager has illustrated a clear path forward with their charter, they should identify key project stakeholders (i.e. future members involved in the project). This can be easily accomplished by creating a stakeholder register.
- ✓ In a project charter, **two evaluation tools** are used to decide whether a project is worth pursuing.
- ✓ These tools can be tailored to optimize the process of project management in software development.
    - **Business Case Document:** This document justifies the necessities of the project and includes an estimate of potential financial benefits.
    - **Feasibility Study:** PMs use this to evaluate a project's objectives, timeline, and costs to determine whether a project is worth executing. Feasibility studies simply allow you to assess a project based on:
        - Requirements of the project,
        - Available resources.
- ✓ It's important to note that although clear objectives of the project are established during the initiation phase, a project charter should not include complex technical details that are discussed in phase two (project planning).

2. Project Planning

- ✓ It is crucial that a project manager diligently lays out the project planning stage, as it is a decisive factor for the project's roadmap. Typically, this can be fulfilled with agile project management, effectively breaking down weeks of planning into mere days.

- ✓ The PM should identify technical requirements and develop a detailed project schedule by creating a **clear communication plan and establishing goals/deliverables.**
- ✓ Additionally, requirement analysis can be used during planning with inputs from the customer, sales department, market surveys and domain experts in the industry.
- ✓ For project management in software development, cross-collaboration is crucial, as it links the development team with the business management teams.

3. Project Execution

- ✓ During project execution, team members begin completing the actual work and subtasks of the project. A project manager should look forward to establishing efficient workflows while diligently monitoring the collective progress of the team.
- ✓ Additionally, a project manager must maintain effective collaboration between stakeholders and all team members involved. Ultimately, this will ensure that everyone is one the same page and the project runs without any glaring issues.

4. Project Monitoring and Controlling

- ✓ Although project monitoring is a process that associates with each and every stage of the project management timeline, project managers should use this time to specifically **ensure those project objectives and deliverables are fulfilled.**
- ✓ A project manager will ensure that no one deviates from the original course of the project by setting both Critical Success Factors (CSF) and Key Performance Indicators (KPI).
- ✓ Finally, the manager will also quantitatively record and track the effort/cost during the process, ensuring that constraints such as budget and time are met with long-term sustainability in mind.

5. Project Closure

- ✓ This final phase of the **project management process** typically follows after the final delivery of the product. Occasionally, external talent is hired specifically for the project on contract.
- ✓ Additionally, a **PM will be responsible** for terminating these contracts and completing any necessary paperwork and documentation.
- ✓ Oftentimes, teams will hold a reflection meeting upon project completion in order to contemplate their comprehensive successes and failures before, during, and after development.
- ✓ This method of reflection provides a sense of continuous improvement for the team, enhancing the overall productivity and output of the team for the company.
- ✓ Finally, a project manager should review the entirety of the project (from start to finish), complete a detailed report that covers after facet (e.g. five stages of project management in software development) of the project, and securely store it for future reference.

Software Project vs Other Types of Project

- ✓ Many techniques of general project management also apply to Software Project Management. Fred Brooks identified some characteristics of software projects which make them particularly difficult:

1. Invisibility

- ✓ With Software, **progress is not immediately visible** since work is logical; however, for physical artifacts like bridges, work progress can be seen from time to time. In Software Development, there is a level of uncertainty. It **isn't easy to accurately define detailed requirements** for software projects before starting the development. However, with the introduction of approaches like Agile and Scrum, we can overcome this limitation.

2. Complexity

- ✓ Software projects contain more complexity than other engineered artifacts. For example, in a bridge, there is a clear structural relationship between parts, whereas software **component relationships are much more complicated**. We can't measure the complexity of a software project until we work on it.

3. Conformity

- ✓ Physical systems are governed by consistent physical law, while Software developers have to **conform to the requirements of human clients.**

4. Flexibility

- ✓ Software systems are particularly **subject to change.** A bridge has to be built in a specific order, whereas We can make Software much more flexibly and **restructure parts quite freely.**

Activities covered by software project management

l. The feasibility study

- ✓ This is an investigation to **decide whether a prospective project is worth starting**. Information will be gathered about the general requirements of the proposed system.
- ✓ The probable developmental and operational costs, along with the value of the benefits of the new system are estimated. With a large system, the feasibility study could be treated as a project in its own right. This evaluation may be done as part of a strategic planning exercise where a whole range of potential software developments are evaluated and put into an order of priority.
- ✓ Sometimes an organization has a policy where a series of projects is planned as a programmed of development.

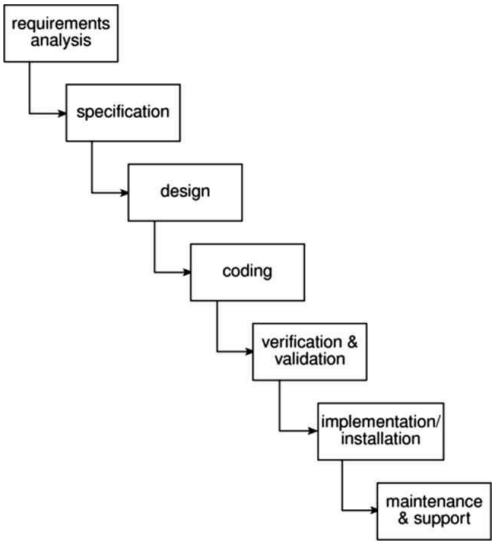
2. ***Planning***

- ✓ If the feasibility study produces results that indicate that the prospective project appears viable, then planning of the project can take place.

- ✓ In fact, for a large project, we would not do all our detailed planning right at the beginning. We would formulate an outline plan for the whole project and a detailed one for the first stage. More detailed planning of the later stages would be done as they approached.
- ✓ This is because we would have more detailed and accurate information upon which to base our plans nearer to the start of the later stages.

3. ***Project execution***

- ✓ The project can now be executed. Individual projects are likely to differ considerably but a classic project life-cycle is shown in Figure 1.1. The stages in the life-cycle illustrated in Figure 1.1 above are described in a little more detail below:
  - **Requirements analysis** This is finding out in detail what the users require of the system that the project is to implement. Several different approaches to the users' requirements may be explored. For example, a small system that satisfies some, but not all, of the users' needs at a low price may be compared to a system with more functions but at a higher price.
  - **Specification** Detailed documentation of **what the proposed system is to do.**
  - **Design** A design that meets the specification has to be drawn up. This design activity will be in two stages. One will be the **external or user design.** This lays down what the system is to look like to the users in terms of menus, screen and report layouts and so on. The next stage produces the physical design, which tackles the way in which the data and software procedures are be structured internally.
  - **Coding** This might refer to writing code in a procedural language such as C or Ada, or might refer to the use of a high-level application builder. Even where software is not being built from scratch, some modification to the base application might be required to meet the needs of the new application.



  - **Verification and validation** Whether software is developed specially for the current application or not, careful testing will be needed to check that the proposed system meets its requirements.
  - **Implementation/installation** Some system development practitioners refer to the whole of the project after design as 'implementation' (that is, the implementation of the design) while others insist that the term refers to the installation of the system after the software has been developed. In this case it encompasses such things as setting up data files and system parameters, writing user manuals and training users of the new system.

- **Maintenance and support** Once the system has been implemented there will be a continuing need for the correction of any errors that may have crept into the system and for extensions and improvements to the system. Maintenance and support activities may be seen as a series of minor software projects. In many environments, most software development is in fact maintenance.

SOME WAYS OF CATEGORIZING SOFTWARE PROJECTS

- ✓ It is important to distinguish between the main types of software project because what is appropriate in one context might not be so in another. For example, SSADM, the Structured Systems Analysis and Design Method, is suitable for developing information systems but not necessarily other types of system.

Information systems versus embedded systems

- ✓ A distinction may be made between information systems and embedded systems (Embedded systems are also called real-time or industrial systems). Very crudely, the difference is that in the former case the system interfaces with the organization, whereas in the latter case the system interfaces with a machine. A stock control system would be an information system that controls when the organization reorders stock. An embedded, or process control, system might control the air conditioning equipment in a building. Some systems may have elements of both so that the stock control system might also control an automated warehouse.

Objectives versus products

- ✓ Projects may be distinguished by whether their aim is to produce a product or to meet certain objectives. A project might be to create a product the details of which have been specified by the client. The client has the responsibility for justifying the product.
- ✓ On the other hand, the project might be required to meet certain objectives. There might be several ways of achieving these objectives in contrast to the constraints of the product-driven project. One example of this is where a new information system is implemented to improve some service to users inside or outside an organization. The subject of an agreement would be the level of service rather than the characteristics of a particular information system.
- ✓ Many software projects have two stages. The **first stage** is an objectives-driven project, which results in a recommended course of action and may even specify a new software application to meet identified requirements. The **next stage** is a project actually to create the software product.

The project as a system

- ✓ A project is concerned with creating a new system and/or transforming an old one and is itself a system.

Systems, subsystems and environments

- ✓ A simple definition of the term system is **'a set of interrelated parts'**. A system will normally be part of a larger system and will itself comprise subsystems. Outside the system there will be the system's environment. This will be made up of things that can affect the system but over which the system has no direct control.

Open versus closed systems

- ✓ Open systems are those that interact with the environment. Nearly all systems are open. One reason that engineered systems and the projects to construct them often fail is that the technical staff involved do not appreciate the extent to which systems are open and are liable to be affected by outside changes.

Sub-optimization

- ✓ This is where a subsystem is working at its optimum but is having a detrimental effect on the overall system. An example of this might be where software developers deliver to the users a system that is very efficient in its use of machine resources, but is also very difficult to modify.

Sociotechnical systems

- ✓ Software projects belong to this category of systems. Any software project requires both technological organization and also the organization of people. Software project managers therefore need to have both technical competence and the ability to interact persuasively with other people.

What is management

The Open University suggest that management involves the following activities:

- planning - deciding what is to be done;
- organizing - making arrangements;
- staffing - selecting the right people for the job, for example;
- directing - giving instructions;
- monitoring - checking on progress;
- controlling - taking action to remedy hold-ups;
- innovating - coming up with new solutions;

representing - liaising with users etc.

- ✓ Another way of looking at the management task is to ask managers what their most frequent challenges are. A survey of software project managers produced the following list:
  - coping with resource constraints (83%);
  - communicating effectively among task groups (80%);
  - gaining commitment from team members (74%);
  - establishing measurable milestones (70%);

- working out project plan agreement with their team (57%);
- gaining commitment from management (45%);
- managing vendors and sub-contractors (38%).

✓ The percentages relate to the numbers of managers identifying each challenge. A manager could identify more than one.

Problems with software projects

✓ One way of deciding what ought to be covered in 'software project management' is to consider what problems need to be addressed.

✓ Traditionally, management has been seen as the preserve of a distinct class within the organization. As technology has made the tasks undertaken by an organization more sophisticated, many management tasks seem to have become dispersed throughout the organization: there are management systems rather than managers.

✓ Nevertheless, the successful project will normally have one person who is responsible for its success. Such people are likely to be concerned with the key areas that arc most likely to prevent success - they arc primarily trouble-shooters and their job is likely to be molded by the problems that confront the project. A survey of managers published by Thayer, Pyster and Wood identified the following commonly experienced problems:

- poor estimates and plans;
- lack of quality standards and measures;
- lack of guidance about making organizational decisions;
- lack of techniques to make progress visible;
- poor role definition - who does what?
- incorrect success criteria.

✓ The above list looks at the project from the manager's point of view. What about the staff who make up the members of the project team? Below is a list of the problems identified by a number of students on a degree course in Computing and Information Systems who had just completed a year's industrial placement:

- inadequate specification of work;
- management ignorance of IT;
- lack of knowledge of application area;
- lack of standards;
- lack of up-to-date documentation;
- preceding activities not completed on time - including late delivery of equipment;
- lack of communication between users and technicians;
- lack of communication leading to duplication of work;

- lack of commitment - especially when a project is tied to one person who then moves;
- narrow scope of technical expertise;
- changing statutory requirements;
- changing software environment;
- deadline pressure;
- lack of quality control;
- remote management;

✓ Note how many of the problems identified by the students stemmed from poor communications. Another common problem identified by this and other groups of students is the wide range of IT specialisms - an organization may be made up of lots of individuals or groups who will be expert in one set of software techniques and tools but ignorant of those used by their colleagues. Communication problems are therefore bound to arise.

✓ What about the problems faced by the customers of the products of computer projects? Here are some recent stories in the press:

- the United States Internal Revenue System was to abandon its tax system modernization programmed after having spent S4 billion:
- the state of California spent SI billion on its non-functional welfare database system:
- the £339 million United Kingdom air traffic control system was reported as being two years behind schedule;
- a discount stock brokerage company had 50 people working 14 hours or more a day to correct three months of records clerically—the report commented that the new system had been rushed into operation without adequate testing;
- in the United Kingdom, a Home Office immigration service computerization project was reported as having missed two deadlines and was nine months late;
- the Public Accounts Committee of the House of Commons in the United Kingdom blamed software bugs and management errors for £12 million of project costs in relation to an implementation of a Ministry of Agriculture computer system to administer farm subsidies.

✓ Most of the stories above relate to public sector organizations. This may be misleading—private sector organizations tend to conceal their disasters and in any case many of the public projects above were actually being carried out by private sector contractors. Any lingering faith by users in the innate ability of IT people to plan ahead properly will have been removed by consideration of the 'millennium bug", a purely self-inflicted IT problem. On balance it might be a good idea not to survey users about their problems with IT projects!

Stakeholders

- ✓ Stakeholders are persons and organizations such as customers, sponsors, performing organization and the public, who are actively involved in the project or those whose interests may be positively or negatively affected by the execution, completion, or cancellation of the project.
- ✓ Stakeholders may also exert influence over the project and its deliverables. The project management team must identify both internal and external stakeholders in order to determine the requirements and expectations of all parties involved.
- ✓ Furthermore, the project manager must manage the influence of the various stakeholders in relation to the project requirements to ensure a successful outcome. Figure 2-6 illustrates the relationship between the project, the project team, and other common stakeholders.

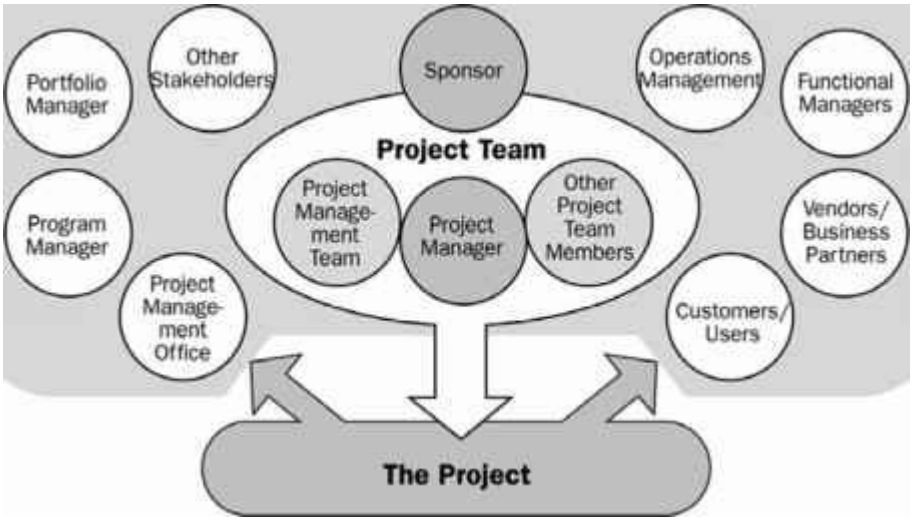


Figure 2-6. The Relationship Between Stakeholders and the Project

- ✓ Stakeholders have varying levels of responsibility and authority when participating on a project, and these can change over the course of the project's life cycle. Their responsibility and authority may range from occasional contributions in surveys and focus groups to full project sponsorship, which includes providing financial and political support.
- ✓ Stakeholders can have an adverse impact on the project objectives. Likewise, project managers who ignore stakeholders can expect a negative impact on project outcomes.
- ✓ Stakeholder identification can be difficult at times. For instance, it could be argued that an assembly-line worker whose future employment depends on the outcome of a new product-design project is a stakeholder. Identifying stakeholders and understanding their relative degree of influence on a project is critical. Failure to do so can extend the timeline and raise costs substantially.
- ✓ An example is late recognition that the legal department is a significant stakeholder; this could cause delays and increase expenses because of the additional documentation requirements necessary to carry out project tasks.

- ✓ A project can have both positive and negative stakeholders. Positive stakeholders arc those who would normally benefit from a successful outcome from the projcct, while negative stakeholders are those who perceive negative outcomes from the project's success. For example, business leaders from a community that will benefit from an industrial expansion project may be positive stakeholders because they sec economic benefit to the community.
- ✓ Conversely, environmental groups could be negative stakeholders if they view the projcct as harmful to the environment. In the case of positive stakeholders, their interests are best served by helping the project succeed. The interests of negative stakeholder's arc served by impeding the project's progress.
- ✓ Overlooking negative stakeholders can result in an increased likelihood of failure. An important part of a project manager's responsibility is to manage stakeholder expectations. This can be difficult because stakeholders often have very different or conflicting objectives.
- ✓ Part of the project manager's responsibility is to balance these interests and ensure that the project team interacts with stakeholders in a professional and cooperative manner.

Customers/Users

- ✓ The customers/users are the persons or organizations that will use the project's product or service or result. Customers/users may be internal or external. There may also be multiple layers of customers. For example, the customers for a new pharmaceutical product can include the doctors who prescribe it, the patients who use it, and the insurers who pay for it.
- ✓ In some application areas, customers and users are synonymous, while in others, customers refer to the entity acquiring the project's product and users refer to those who will directly utilize the project's product.
- ✓ These customer/users are key sources of information for the project team because it is typically for customers or end users that the project has been created. The customers/users therefore play a significant role in determining the scope of the project, influencing how the project is carried out, and testing the product or service ultimately delivered by the project team.
- ✓ As a result of this close partnership, the customers/users carry significant responsibility for providing accurate and timely data to the project team as well as identifying risks and responding to other issues that arise. Customers and users deal primarily with the project team, but they may also have direct involvement with vendors, business partners, or other operational stakeholders involved with the work necessary to complete the project deliverable.

Sponsor

- ✓ The sponsor is the person or group that **provides the financial resources,** in cash or in kind, for the project. When a project is first conceived, the person or group acting as the sponsor champions

the cause of the project. This includes serving as spokesperson to higher levels of management to garner support throughout the organization and promote the benefits that the project will bring.

- ✓ The sponsor leads the project through the engagement or selection process until formally authorized, and therefore plays a significant role in the development of the initial scope and charter. Another key attribute of the sponsor is to provide financial resources, in cash or in kind, for the project.
- ✓ Sponsors have a major stake in the project's success, and therefore, at times, may take an active role on the project team. For issues that are beyond the control of the project manager, the sponsor serves as an escalation path. The sponsor may also be involved in other important issues such as authorizing changes in scope, phase-end reviews, and go/no-go decisions when risks are particularly high.
- ✓ The sponsor deals directly, and most often, with the project manager. For some projects, the sponsor may also interact with the project team and other key stakeholders, particularly when resolving issues that have been escalated. Some projects have multiple sponsors who fulfill this role.

Portfolio Managers/Portfolio Review Board

- ✓ Portfolio managers are responsible for the high-level governance of a collection of projects or programs, which may or may not be interdependent. Portfolio managers could receive direction from the organization's business strategy or from the guidance of a director within the organization.
- ✓ Portfolio review boards are a committee usually made up of the organization's executives who act as a project selection panel. They review each project for its return on investment, the value of the project, risks associated with taking on the project, and other attributes of the project.
- ✓ Portfolio review boards are not essential in every organization but, when used, provide additional support for project selection and prioritization. Portfolio review boards can also assist with responding to Request for Proposals (RFPs), tenders, or other opportunities that are developed externally.

Program Managers

- ✓ Program managers are responsible for managing multiple projects when projects gain some measure of benefit by being managed collectively. The program manager ensures that all related projects are integrated, on schedule, on budget, and ultimately serve the common goal for which the program was created. The program manager interacts with each project manager to provide support and guidance on the individual projects.

Project Management Office

- ✓ A project management office (PMO) organizes and manages control over projects within an organization, providing a uniform approach regardless of the discipline, technology, or purpose. The PMO can be a stakeholder if it has direct or indirect responsibility for the outcome of the project. The PMO can provide but is not limited to:
  - Administrative support services such as policies, methodology and templates,
  - Mentoring to project managers,
  - Project support, guidance and training on how to manage projects and the use of tools,
  - Resource alignment of project staff, or
  - Centralized communication among project managers, project sponsors, managers, and other stakeholders.

Project Managers

- ✓ The project manager is the person assigned by the performing organization to achieve the project objectives. This is a challenging, high-profile role with significant responsibility and constantly shifting priorities. It requires flexibility, good judgment, and strong negotiating skills. The project manager must be able to understand project detail, but manage from the big-picture perspective.
- ✓ As the person responsible for the success of the project, the project manager is in charge of all aspects of the project including, but not limited to:
  - Developing the project plan and all related component plans,
  - Keeping the project on track in terms of budget and schedule,
  - Identifying, monitoring, and responding to risk, or
  - Providing accurate and timely reporting of project metrics.
- ✓ The project manager is the lead person responsible for interfacing with all stakeholders, particularly the project sponsor, project team, and other key stakeholders.

Project Team

- ✓ A project team is comprised of the project manager, project management team, and other team members who carry out the work but who are not necessarily involved with management of the project. On some projects, the sponsor may also be part of the project team.
- ✓ A properly functioning project team can mean the difference between a highly successful project and one that fails. While a team should be staffed with people who have the skills and talents necessary to carry out their respective roles, an effective team is one that demonstrates the ability to work well together and accept team members' strengths and weaknesses.
- ✓ This requires a willingness to communicate clearly, accurately, and fully and to commit to quality work and meeting deadlines. Individual team members must also be aware of how their work affects the work of other team members.

Functional Managers

- ✓ Functional managers are individuals who play a management role within an administrative or functional area of the business, such as human resources, finance, accounting, or procurement. They are assigned their own permanent staff to carry out the ongoing work, and they have a clear directive to manage all tasks within their functional area of responsibility.
- ✓ A project manager may need to work with a functional manager to make use of the services the functional group typically provides. For instance, a project manager may deal with a human resources manager to hire a new employee or a contractor with the right skill set for carrying out necessary project tasks. Likewise, a functional manager in finance may be a resource regarding funding of the project and other budgetary details.
- ✓ Functional managers deal most often with the project manager or other members of the project management team. Functional managers typically have little interaction with other stakeholders with regard to project work.

Vendors/Business Partners

- ✓ Vendors, also called suppliers or contractors, are external companies that enter into a contractual agreement to provide components or services necessary for the project. Business partners are also external companies, but they have a special relationship with the enterprise, sometimes attained through a certification process.
- ✓ Business partners provide specialized expertise or fill a specified role such as installation, customization, training, or support. When the project work is contracted almost entirely to a series of vendors and business partners the resulting group is referred to as a network organization.
- ✓ As independent contractors, vendors/business partners often deal with the project management team and customers/users. They may also interface with project team members from operations who are involved with tasks related to the contracted product or service.
- ✓ It is the responsibility of vendors/business partners to carry out all contracted duties with the same standards of quality and professionalism as the enterprise itself.

Requirement specification

Very often, especially in the case of product-driven projects, the objectives of the project are carefully defined in terms of functional requirements, quality requirements, and resource requirements.

• **Functional requirements** These define what the system that will be the end product of the project is to do. Systems analysis and design methods, such as SADT and Information Engineering, are designed primarily to provide functional requirements.

• **Quality requirements** There will be other attributes of the system to be implemented that do not relate so much to what the system is to do but how it is to do it. These are still things that the user will

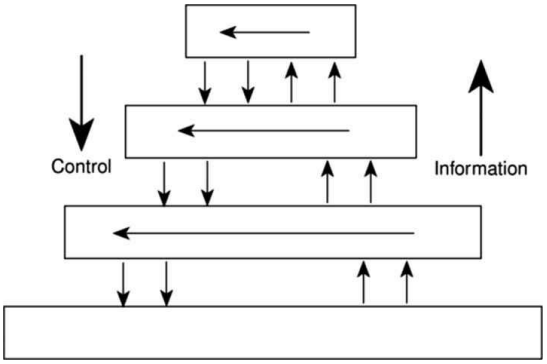
be able to experience. They include, for example, response time, the ease of using the system and its reliability.

• **Resource requirements** A record of how much the organization is willing to spend on the system. There will usually be a trade-off between this and the time it takes to implement the system. In general, it costs disproportionately more to implement a system by an earlier date than a later one. There might also be a trade-off between the functional and quality requirements and cost. We would all like exceptionally reliable and user-friendly systems that do exactly what we want but we might not be able to afford them.

Information and control in organizations

Hierarchical information and control systems

- ✓ With small projects, the project leaders are likely to be working very closely with the other team members and might even be carrying out many non-managerial tasks themselves. Therefore, they should have a pretty good idea of what is going on. When projects are larger, many separate teams will be working on different aspects of the project and the overall managers of the project are not going to have day-to-day direct contact with all aspects of the work.
- ✓ Larger projects are likely to have a hierarchical management structure (Figure 1.3). Project team members will each have a group leader who allocates them work and to whom they report progress. In turn the group leader, along with several other group leaders, will report to a manager at the next higher level. That manager might have to report to another manager at a higher level, and so on.
- ✓ There might be problems that cannot be resolved at a particular level. For example, additional resources might be needed for some task, or there might be a disagreement with another group. These will be referred to the next higher level of management.
- ✓ At each higher level more information will be received by fewer people. There is thus a very real danger that managers at the higher levels might be overloaded with too much information. To avoid this, at each level the information will have to be summarized.



- ✓ The larger the project, the bigger the communication problems. The referral of disagreements to a higher level is sometimes known as escalation.
- ✓ As a result of examining the progress information and comparing it against what was planned, some remedial action might need to be taken. Instructions may be formulated and passed down to a lower level of management. The lower level

managers will have to interpret what needs to be done and formulate more detailed plans to fulfil the directive. As the directives filter down the hierarchy, they will be expanded into more detail at each level.

- ✓ Not all information flows concerning a project will be going up and down the hierarchy. There will also be lateral flows between groups and individuals on the same level.

Levels of decision making and information

- ✓ Each decision made in a project environment should be based on adequate information of the correct sort. The type of information needed depends on the level of decision making. Decisions can be grouped at three levels: strategic, tactical, and operational.
- ✓ Strategic decision making is essentially about deciding objectives. In the case of the Bright mouth College payroll, the decision to become administratively independent could be regarded as a strategic decision.
- ✓ In our example we were interested only in the payroll, but this might have been part of a wider programmed which may have affected many other administrative functions.
- ✓ Tactical decision making is needed to ensure that the objectives will be fulfilled. The project leader who has the responsibility for achieving objectives will have to formulate a plan of action to meet those objectives.
- ✓ The project leader will need to monitor progress to see whether these objectives are likely to be met and to take action where needed to ensure that the things remain on course.
- ✓ Operational decisions relate to the day-to-day work of implementing the project. Deciding the content of the acceptance tests might come under this heading.

Differences in types of information

- ✓ Table 1.1 gives some idea of the differences in the kind of information needed. There is a kind of continuum for most of the qualities suggested and what is needed for tactical decision making comes somewhere in the middle. Effectiveness is concerned with doing the right thing. Efficiency is carrying out a task making the best possible use of the resources.

Table 1.1 The types of information required for decision making

CHARACTERISTIC	OPERATIONAL	STRATEGIC
Motivation	Efficiency	effectiveness
Orientation	Internal	internal and external
Focus	specific to a function	specific to organization
Detail	Detailed	summarized
Response	Fast	not so fast
access paths	Standard	flexible
up-to-datedness	Essential	desirable

Accuracy	Essential	approximate
Certainty	Essential	often predictive
Objectivity	High	more subjective
information type	mainly quantitative	often qualitative

Measurement

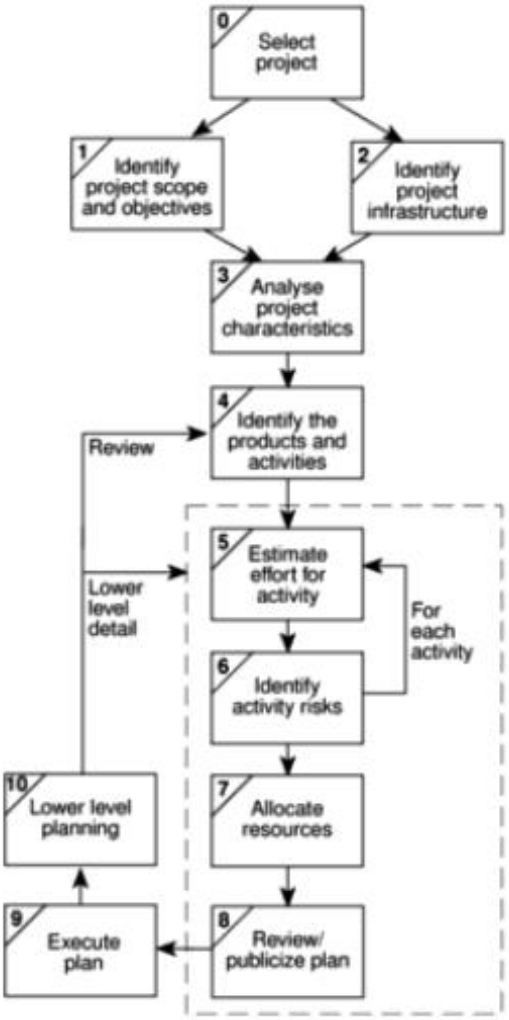
- ✓ The quantification of the leader of a small project will have direct contact with many aspects of the measures of effectiveness project. With larger projects, project leaders would have to depend on information reduces ambiguity being supplied to them. This information should not be vague and ideally should be quantitative. This ties in with our need for unambiguous measures of effectiveness. Software development deals largely with intangibles and does not easily lend itself to quantitative measures, but attempts are increasingly being made to introduce measurement into the software process.
- ✓ Software measurements can be divided into performance measures and predictive measures.
  - **Performance measures** These measure the characteristics of a system that has been delivered. They are important when we are trying to specify unambiguously the quality requirements of a proposed system.
  - **Predictive measures** the trouble with performance measures is that you need to have a system actually up and running before you can take measurements. As a project leader, what you want to be able to do is to get some idea of the likely characteristics of the final system during its development. Predictive measures are taken during development and indicate what the performance of the final system is likely to be.

AN OVERVIEW OF PROJECT PLANNING

Planning is the most difficult process in project management. A major step in project planning is to plan in outline first and then in more detail. Each step of project planning has different activities to perform. Following are the major steps in project planning Steps in Project Planning,

Step 0: Select project
Step 1: Identify project scope and objectives
Step 2: Identify project infrastructure
Step 3: Analyze project characteristics
Step 4: Identify project products and activities
Step 5: Estimate effort for each activity
Step 6: Identify activity risks
Step 7: Allocate resources
Step 8: Review / Publicize plan
Step 9 & 10: Execute plan / lower level of planning

STEP WISE PROJECT PLANNING



Step 0: Select project

✓ This is called step 0 because in a way of project planning, it is **outside the main project planning process**. Feasibility study suggests us that the **project is worthwhile or not.** This project evaluation may be done on an individual basis or as part of strategic planning.

Step 1: Identify project scope and objectives

✓ The activities in this step **ensure that all the parties to the project agree on the objectives** and are committed to the success of the project. A danger to be avoided is overlooking people who are affected by the project.

Step 1.1: Identify objectives and practical measures of the effectiveness in meeting those objectives

✓ The need for agreed objectives for a project and ways of measuring the success in achieving those objectives.

Step 1.2: Establish a project authority

✓ A single overall project authority needs to be established so that there is unity of purpose among all those concerned.

Step 1.3: Identify all stakeholders in the project and their interests

✓ Essentially all the parties who have an interest in the project need to be identified.

Step 1.4: Modify objectives in the light of stakeholder analysis

✓ In order to gain the full cooperation of all concerned, it might be necessary to modify the project objectives. This can mean adding new features to the system giving a benefit to some stakeholder group as a means of assuring their commitment to the project.

✓ This is potentially dangerous, since the system size might be increased and the original objectives obscured. Because of these dangers, this process must be done consciously and in a controlled manner.

Step 1.5: Establish methods of communication with all parties

✓ For internal staff, this should be fairly straightforward, but a project leader implementing a payroll system would need to find a contact point with BACS (Bankers Automated Clearing Scheme) for instance.

Step 2: Identify project infrastructure

- ✓ Projects are rarely initiated in a vacuum. There is usually some kind of existing infrastructure into which the project can fit. The project leader who does not already know about this structure needs to find out its precise nature.

Step 2.1: Identify relationship between the project and strategic planning

- ✓ As well as identifying projects to be carried out, an organization needs to decide the order in which these projects are to be carried out.
- ✓ It also needs to establish the framework within which the proposed new systems are to fit. Hardware and software standards, for example, are needed so that various systems can communicate with each other. These strategic decisions must be documented in a strategic business plan or in an information technology plan that is developed from the business plan.

Step 2.2: Identify installation standards and procedures

- ✓ Any organization that develops software should define its development procedures. As a minimum, the normal stages in the software life cycle to be carried out should be documented along with the products created at each stage. Change control and configuration management standards should be in place to on ensure that changes to requirements are implemented in a safe and orderly way.
- ✓ The procedural standards may lay down the quality checks that need to be done at each point of the project life cycle or these may be documented in a separate quality standards and procedures manual.
- ✓ The organization, as part of its monitoring and control policy must have in place a measurement programmed that dictates that certain statistics have to be collected at various stages of a project.
- ✓ Finally the project manager should be aware of any project planning and control standards. These will relate to the way that the project is controlled: for example, the way that the hours spent by team members on individual tasks are recorded on time-sheets.

Step 2.3: Identify project team organization

- ✓ Project leaders, especially in the case of large projects, will often have some control over the organizational structure of the project team. More often, though, the organizational structure will be dictated to them. For example, there might have been a high-level managerial decision that code developers and systems analysts will be in different groups, or that the development of PC applications will not be done within the same group as that responsible for 'legacy' main-frame applications.
- ✓ If the project leader does have some control over the project team organization then this would best be considered at a later stage.

Step3: Analyze project characteristics

- ✓ The general purpose of this part of the planning operation is to ensure that the Chapter 4 elaborates on appropriate methods are used for the project. the process of analyzing project characteristics.

Step 3.1: Distinguish the project as either objective- or product-driven

- ✓ This has already been discussed in the first chapter. A general point to note is that as system development advances, it tends to become more product-driven, although the underlying objectives always remain and must be respected.

Step 3.2: Analyze other project characteristics (including quality-based ones)

- ✓ For example, is this an information system that is being developed or a process control system, or does it have elements of both? Is it a safety-critical system, that is, where human life could be threatened by a malfunction?

Step 3.3: Identify high level project risks

- ✓ Consideration must be given to the risks that threaten the successful outcome of the project. Generally speaking, most risks can be attributed to the operational or development environment, the technical nature of the project or the type of product being created.

Step 3.4: Take into account user requirements concerning implementation

- ✓ The clients will usually have their own procedural requirements. For example, work for government departments usually requires the use of SSADM.

Step 3.5: Select general lifecycle approach in the light of the above

- ✓ The project life cycle to be used for the project will be influenced by the issues Chapter 4 discusses life raised above. For example, a prototyping approach might be used where the user cycles in more detail, requirements are not clear.

Step 3.6: Review overall resource estimates

- ✓ Once the major risks have been identified and the broad project approach has been decided upon, this would be a good point at which to re-estimate the effort and other resources required to implement the project. Where enough information is available, an estimate based on function points might be appropriate.

Step 4: Identify project products and activities

The more detailed planning of the individual activities that will be needed now takes place. The longer-term planning is broad and in outline, while the more immediate tasks are planned in some detail.

Step 4.1: Identify and describe project products (or deliverables)

- ✓ In general, there can be no project products that do not have activities that create them. Wherever possible, we ought also to ensure the reverse: that there are no activities that do not produce a

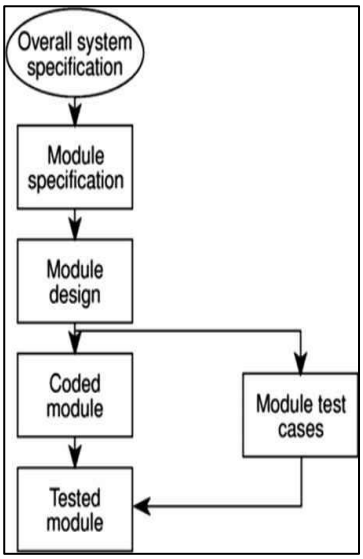
tangible product. Making sure we have identified all the things the project is to create helps us to ensure that all the activities we need to carry out are accounted for.

- ✓ These products will include a large number of technical products including training material and operating instructions, but also products to do with the management and the quality of the project. Planning documents would, for example, be management products.
- ✓ The products will form a hierarchy. The main products will have sets of component products, which in turn might have sub-component products and so on. These relationships can be documented in a Product Breakdown Structure (PBS).
- ✓ This part of the planning process draws heavily on the standards laid down in PRINCE 2. These specify that products at the bottom of the PBS should be documented by Product Descriptions, which contain:
  - the name/identity of the product;
  - the purpose of the product;
  - the derivation of the product (that is, the other products from which it is derived);
  - the composition of the product;
  - the relevant standards;

  the quality criteria that should apply to it.

Step 4.2: Document generic product flows

- ✓ Some of the products will need some other product to exist first before they can be created.



- ✓ For example, a program design must be created before the program can be written and the program specification must exist before the design can be commenced. These relationships can be portrayed in a Product Flow Diagram (PFD). Figure 2.3 gives an example.

Step 4.3: Recognize product instances

Where the same generic PFD fragment relates to more than one instance of a particular type of product, an attempt should be made to identify each of those instances.

Step 4.4: Produce ideal activity network

- ✓ In order to generate one product from another there must be one or more activities that carry out the transformation. By identifying these activities, we can create an activity network, which shows the tasks that have to be carried out and the order in which they have to be executed.
- ✓ The activity networks are 'ideal' in the sense that no account has been taken of resource constraints. For example, in Figure 2.4, it is assumed that resources are available for all four software modules to be developed in parallel.

Step 4.5: Modify the ideal to take into account need for stages and checkpoints

- ✓ The approach to sequencing activities described above encourages the formulation of a plan that will minimize the overall duration, or 'elapsed time', for the project. It assumes that an activity will start as soon as the preceding ones upon which it depends have been completed.
- ✓ There might, however, be a need to modify this by dividing the project into stages and introducing checkpoint activities.
- ✓ These are activities that draw together the products of preceding activities to check that they are compatible. These checkpoints are sometimes referred to as milestone events.
- ✓ A checkpoint could potentially delay work on some elements of the project - there has to be a trade-off between efficiency and quality.

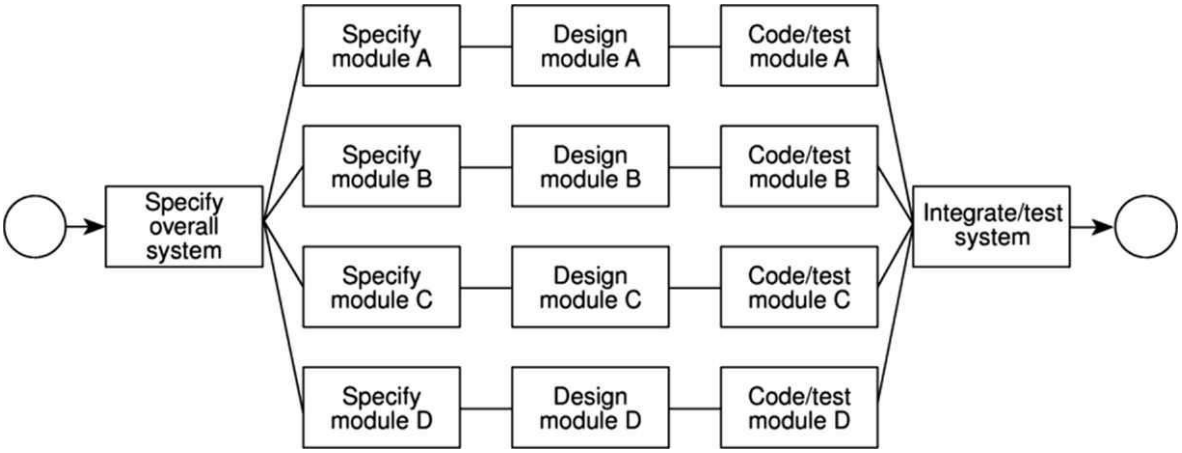


Figure 2.4 An activity network fragment for the IOE Maintenance Group Accounts project.

Step 5: Estimate effort for each activity

Step 5.1: Carry out bottom-up estimates

- ✓ Some top-down estimates of effort, cost and duration will already have been done (see Step 3.6). At this point, estimates of the staff effort and other resources required, and the probable elapsed time needed for each activity will need to be produced. The method of arriving at each of these estimates will vary depending on the type of activity.
- ✓ The individual activity estimates of effort should be summed to get an overall bottom-up estimate, which can be reconciled with the previous top-down estimate.
- ✓ The activities on the activity network can be annotated with their elapsed times so that the overall duration of the project can be calculated.

Step 5.2: Revise plan to create controllable activities

- ✓ The estimates for individual activities might reveal that some are going to take quite a long time. Long activities often make a project difficult to control.
- ✓ If an activity involving system testing is to take 12 weeks, it might be difficult after six weeks to judge accurately whether 50% of the work is completed.
- ✓ It would be better to break this down into a series of smaller sub-tasks.

Step 6: Identify activity risks

Step 6.1: Identify and quantify activity-based risks

- ✓ Risks inherent in the overall nature of the project have already been considered in Step 3. We now want to look at each activity in turn and assess the risks to its successful outcome. The seriousness of each risk and likelihood of it occurring have to be gauged.
- ✓ At individual task level some risks are unavoidable, and the general effect if a problem materializes is to make the task longer or more costly. A range of estimates can be produced to take into account the possible occurrence of the risks.

Step 6.2: Plan risk reduction and contingency measures where appropriate

- ✓ It is possible to avoid or at least reduce some of the identified risks. Contingency plans specify action that is to be taken if a risk materializes. For example, a contingency plan could be to use contract staff if a member of the project team is unavailable at a key time because of illness.

Step 6.3: Adjust overall plans and estimates to take account of risks

- ✓ We can change our plans, perhaps by adding new activities which reduce risks. For example, a new programming language could mean that we schedule training courses and time for the programmers to practice their new programming skills on some non-essential work.

Step 7: Allocate resources

Step 7.1: Identify and allocate resources

- ✓ The type of staff needed for each activity is recorded. The staff available for the project are identified and are provisionally allocated to tasks.

Step 7.2: Revise plans and estimates to take into account resource constraints

- ✓ Some staff might be needed for more than one task at the same time and. in this case, an order of priority is established. The decisions made here can have an effect on the overall duration of the project when some tasks are delayed while waiting for staff to become free.
- ✓ Ensuring someone is available to start work on an activity as soon as the preceding activities have been completed might mean that they are idle while waiting for the job to start and are therefore used inefficiently.

Step 8: Review/publicize plan

Step 8.1: Review quality aspects of the project plan

- ✓ A danger when controlling any project is that an activity can reveal that an earlier activity was not properly completed and needs to be reworked. This, at a stroke, can transform a project that appears to be progressing satisfactorily into one that is badly out of control.
- ✓ It is important to know that when a task is reported as completed, it really is- hence the importance of quality reviews. Each task should have 'exit requirements. These are quality checks that have to be passed before the activity can be 'signed off as completed.

Steps 9 and 10: Execute plan and Lower levels of planning

- ✓ Once the project is under way, plans will need to be drawn up in greater detail for each activity as it becomes due. Detailed planning of the later stages will have to be delayed because more information will be available nearer the start of the stage.
- ✓ Of course, it is necessary to make provisional plans for the more distant tasks, because thinking about what has to be done can help unearth potential problems, but sight should not be lost of the fact that these plans are provisional.

SIX SIGMA

- ✓ **Digital transformation** has become the hottest word of this world. New technologies and tools are supporting the transformation journey of companies big and small as they compete to get a bigger slice of business in a fast-paced competitive environment.
- ✓ Yet, is it enough to smooth a company's transformative process? Can a standalone technology implementation **remove a bottleneck in the production process** or support troubleshooting a service design flaw?
- ✓ Over the years, it has been refined and polished into a sound theory of principles and methods, aimed at business transformation through a **clearly defined process.** This finished product is Six Sigma.

Six Sigma

- ✓ 6σ is concept to **achieve Quality and process improvement,** by eliminating defects through analysing variance with statistical tools and making necessary improvements in a process /product. It **aims to reduce variance.** This was first introduced by Motorola to reduce defects.
- ✓ Motorola was producing TV Sets. It was observed that their market share was reduced due to defective products. And they found that their quality management systems were weak and they wanted to improve, thus they found Six Sigma (6σ).
- ✓ σ means standard deviation. σ is a Greek small Alphabet. Like in English, capital and small alphabets, Greek alphabets has capital and small letters.
- ✓ Σ (sigma) Capital Greek letter indicating summation.
- ✓ σ (sigma) Small Greek letter indicating standard deviation.
- ✓ It can only be implemented wherever the variance is measurable. Variance is analysed through statistical tools. So statistical and probability knowledge is necessary.
- ✓ What is variance? Variance can be termed as deviation from specification.
- ✓ Can we produce a product or process without variation? No, because several factors influence deviation. We can see in engineering drawing that always some tolerance limits are given to the specification (Upper specification limit (USL) and Lower specification limit (LSL)).

- ✓ If the product made is exactly as per specification or within tolerance limits, Then the product is accepted or else rejected.
- ✓ **What does 6σ do?** 6σ is aimed to reduce the variance to the minimum. Variance is the **main cause of defects.** **How it is done?** By studying variation through statistical tools. And making necessary improvements / Corrections/Design changes to reduce the variation.
- ✓ Quality can be termed as internal and external quality. **External quality** - Product or service quality given to customer. we always concentrate on external quality. Because product with defects cannot be sold to customer. **Internal quality** - Quality at different stages in producing a product at manufacturing facility.
- ✓ If the percentage of defects increases externally or internally or in both areas. The company would be affected drastically. It has to spend a lot for reworking.
- ✓ 6σ aims that no defective product is produced at different process in manufacturing. It is same applicable to service industry also. Is it really possible to produce products without defects? Very difficult. If a company is on 6σ level the probability of defects is 3.4 or less than 3.4 defects per million opportunities. That is If we produce 1 million products the chances of defects are 3.4 products. It can also be said that 6σ yields 99.9997% good products.
- ✓ There are several sigma levels also please find the below.
  - 6σ. (99.9997% good parts) (3.4 DPMO)
  - 5σ. (99.98% good parts) (244 DPMO)
  - 4σ. (99.38% good parts) (6210 DPMO)
  - 3σ. (93.32% good parts) (66807 DPMO)
  - 2σ. (69.13% good parts) (308538 DPMO)
  - 1σ. (30.23% good parts) (697700 DPMO)

*DPMO - Defects per million opportunities.

Six Sigma Levels

White Belt

- ✓ This is the simplest stage, where: Any newcomer can join. People work with teams on problem-solving projects. The participant is required to **understand the basic Six Sigma concepts.**

Yellow Belt

- ✓ Here, the participant: Takes part as a project team member. Reviews process improvements. Gains **understanding of the various methodologies, and DMAIC.**

Green level

- ✓ This level of expertise requires the following criteria: Minimum of three years of full-time employment. Understand the tools and methodologies used for problem-solving. Hands-on

experience on projects involving some level of business transformation. Guidance for Black Belt projects in data collection and analysis. Lead Green Belt projects or teams.

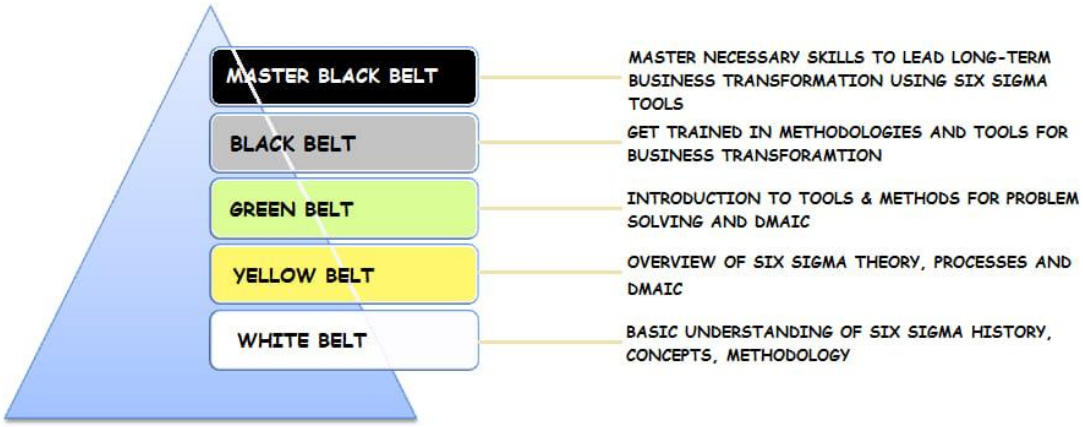
Black Level

- ✓ This level includes the following: Minimum of three years of full-time employment Work experience in a core knowledge area. Proof of completion of a **minimum of two Six Sigma projects** Demonstration of expertise at applying multivariate metrics to diverse business change settings. Leading diverse teams in problem-solving projects. Training and coaching project teams.

Master Black Belt

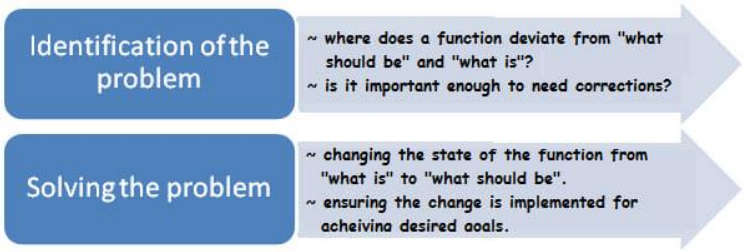
- ✓ To reach this level, a candidate must: Be in possession of a Black Belt certification Have a minimum of five years of full-time employment, or Proof of completion of a minimum of 10 Six Sigma projects A proven work portfolio, with individual specific requirements, as given here, for instance. Have coached and trained Green Belts and Black Belts. Develop key metrics and strategies. Have worked as an organization's Six Sigma technologist and internal business transformation advisor.

<u>The Six Sigma Certification Levels</u>



What is Six Sigma?

- ✓ Six Sigma is a **set of management tools and techniques** designed to improve business by reducing the likelihood of error. It is a data-driven approach that **uses a statistical methodology for eliminating defects.** The etymology is based on the Greek symbol "sigma" or "σ," a statistical term for measuring process deviation from the process mean or target.
- ✓ "Six Sigma" **comes from the bell curve** used in statistics, where one Sigma symbolizes a single standard deviation from the mean.
- ✓ If the process has six Sigma's, three above and three below the mean, the defect rate is classified as "extremely low." So, processes, where the mean is minimum 6σ away from the closest specification limit, are aimed at Six Sigma. The concept of Six Sigma has a simple goal – delivering near-perfect goods and services for business transformation for optimal customer satisfaction (CX).
- ✓ Goals are achieved through a two-pronged approach:



The Six Sigma Methodology

- ✓ The two main Six Sigma methodologies are **DMAIC and DMADV.** Each has its own set of recommended procedures to be implemented for business transformation.
- ✓ **DMAIC** is a data-driven method used to improve existing products or services for better customer satisfaction. It is the acronym for the five phases: **D – Define, M – Measure, A – Analyse, I – Improve, C – Control.**
- ✓ DMAIC is applied in the manufacturing of a product or delivery of a service.
- ✓ **DMADV** is a part of the Design for Six Sigma (DFSS) process used to design or re-design different processes of product manufacturing or service delivery.
- ✓ The five phases of DMADV are: **D – Define, M – Measure, A – Analyse, D – Design, V –Validate.**
- ✓ DMADV is employed when existing processes do not meet customer conditions, even after optimization, or when it is required to develop new methods. It is executed by Six Sigma Green Belts and Six Sigma Black Belts and under the supervision of Six Sigma Master Black Belts.
- ✓ The two methodologies are used in different business settings, and professionals seeking to master these methods and application scenarios would do well to take an online certificate program taught by industry experts.

The Six Sigma Process of Business Transformation

- ✓ Although Six Sigma uses various methods to discover deviations and solve problems, the DMAIC is the standard methodology used by Six Sigma practitioners. Six Sigma uses a data-driven management process used for optimizing and improving business processes. The underlying framework is a strong customer focus and robust use of data and statistics to conclude.
- ✓ The Six Sigma Process of the DMAIC method has five phases:

- ✓ Each of the above phases of business transformation has several steps:

DEFINE

- ✓ The Six Sigma process begins with a customer-centric approach.

Step 1: The **business problem is defined** from the customer perspective.

Step 2: **Goals are set.** What do you want to achieve?

Step 3: **Map the process.** Verify with the stakeholders that you are on the right track.

MEASURE

- ✓ The second phase is focused on the metrics of the project and the tools used in the measurement. How can you improve? How can you quantify this?

Step 1: Measure your problem in numbers or with supporting data.

Step 2: Define performance yardstick. Fix the limits for "Y. "

Step 3: Evaluate the measurement system to be used. Can it help you achieve your outcome?

ANALYZE

- ✓ The third phase analyzes the process to discover the influencing variables.

Step 1: Determine if your process is efficient and effective.

Step 2: Quantify your goals in numbers. For instance, reduce defective goods by 20%.

Step 3: Identify variations using historical data.

IMPROVE

- ✓ This process investigates how the changes in "X" impact "Y." This phase is where you identify how you can improve the process implementation.

Step 1: Identify possible reasons. Test to identify which of the "X" variables identified in influence "Y".

Step 2: Discover relationships between the variables

Step 3: Establish process tolerance, defined as the precise values that certain variables can have, and still fall within acceptable boundaries, for instance, the quality of any given product.

CONTROL

- ✓ In this final phase, you determine that the performance objective identified in the previous phase is well implemented and that the designed improvements are sustainable.

Step 1: Validate the measurement system to be used.

Step 2: Establish process capability. Is the goal being met?

Step 3: Once the previous step is satisfied, implement the process.

The Six Sigma Tools (2M)

- ✓ Cause and Effect Analysis
- ✓ Flow Chart
- ✓ Pareto Chart
- ✓ Histogram
- ✓ Check Sheet
- ✓ Scatter Plot
- ✓ Control Chart

Six Sigma Techniques

- ✓ The Six Sigma methodology also uses a mix of statistical and data analysis tools such as process mapping and design and proven qualitative and quantitative techniques, to achieve the desired outcome.

Brainstorming

- ✓ Brainstorming is the key process of any problem-solving method and is often utilized in the "improve" phase of the DMAIC methodology. It is a necessary process before anyone starts using any tools. Brainstorming involves bouncing ideas and generating creative ways to approach a problem through intensive freewheeling group discussions. A facilitator, who is typically the lead Black Belt or Green Belt, moderates the open session among a group of participants.

Root Cause Analysis/The 5 Whys

- ✓ This technique helps to get to the root cause of the problems under consideration and is used in the "analyze" phase of the DMAIC cycle. In the 5 Whys technique, the question "why" is asked, again and again, finally leading up to the core issue. Although "five" is a rule of thumb, the actual number of questions can be greater or fewer, whatever it takes to gain clarity.

Voice of the Customer

- ✓ This is the process used to capture the "voice of the customer" or customer feedback by either internal or external means. The technique is aimed at giving the customer the best products and services. It captures the changing needs of the customer through direct and indirect methods. The voice of the customer technique is used in the "define' phase of the DMAIC method, usually to further define the problem to be addressed.

The 5S System

- ✓ This technique has its roots in the Japanese principle of workplace energies. The 5S System is aimed at removing waste and eliminating bottlenecks from inefficient tools, equipment, or resources in the workplace. The five steps used are Seiri (Sort), Seiton (Set in Order), Seiso (Shine), Seiketsu (Standardize), and Shitsuke (Sustain).

Kaizen (Continuous Improvement)

✓ The Kaizen technique is a powerful strategy that powers a continuous engine for business improvement. It is the practice continuously monitoring, identifying, and executing improvements. This is a particularly useful practice for the manufacturing sector. Collective and ongoing improvements ensure a reduction in waste, as well as immediate change whenever the smallest inefficiency is observed.

Benchmarking

✓ Benchmarking is the technique that employs a set standard of measurement. It involves making comparisons with other businesses to gain an independent appraisal of the given situation. Benchmarking may involve comparing important processes or departments within a business (internal benchmarking), comparing similar work areas or functions with industry leaders (functional benchmarking), or comparing similar products and services with that of competitors (competitive benchmarking).

Poka-yoke (Mistake Proofing)

✓ This technique's name comes from the Japanese phrase **meaning "to avoid errors,"** and entails preventing the chance of mistakes from occurring. In the poka-yoke technique, employees spot and remove inefficiencies and human errors during the manufacturing process.

Value Stream Mapping

✓ The value stream mapping technique charts the current flow of materials and information to design a future project. The objective is to remove waste and inefficiencies in the value stream and create leaner operations. It identifies seven different types of waste and three types of waste removal operations.

DEFINING SOFTWARE QUALITY

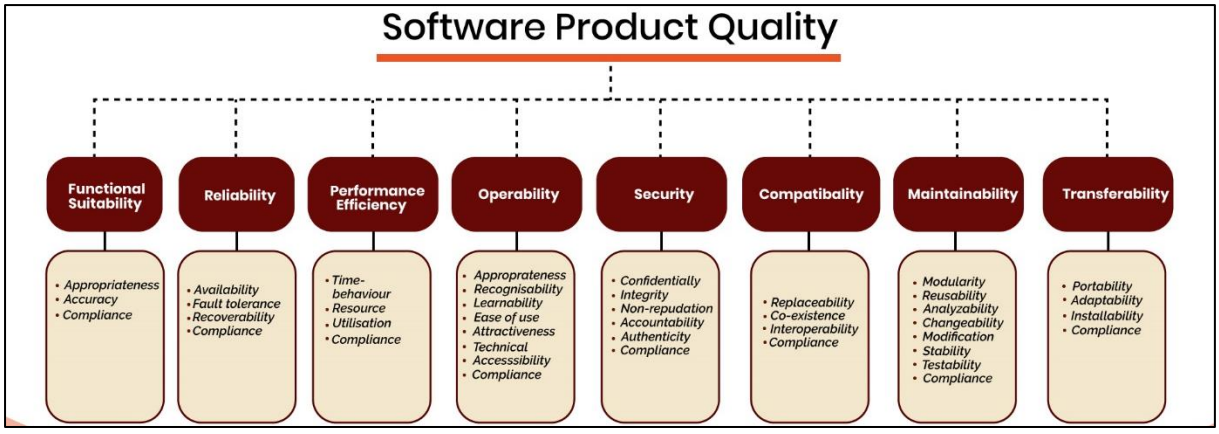
What is software quality?

✓ The quality of software can be defined as the **ability of the software to function as per user requirement.** When it comes to software products it must satisfy all the functionalities written down in the SRS document.

Key aspects that conclude software quality include,

- **Good design** – It's always important to have a good and aesthetic design to please users
- **Reliability** – Be it any software it should be able to perform the functionality impeccably without issues
- **Durability-** Durability is a confusing term, in this context, durability means the ability of the software to work without any issue for a long period of time.
- **Consistency** – Software should be able to perform consistently over platform and devices

- **Maintainability** – Bugs associated with any software should be able to capture and fix quickly and news tasks and enhancement must be added without any trouble
- **Value for money –** customer and companies who make this app should feel that the money spent on this app has not fine to waste.



ISO/IEC 25010:2011 Software Quality Model

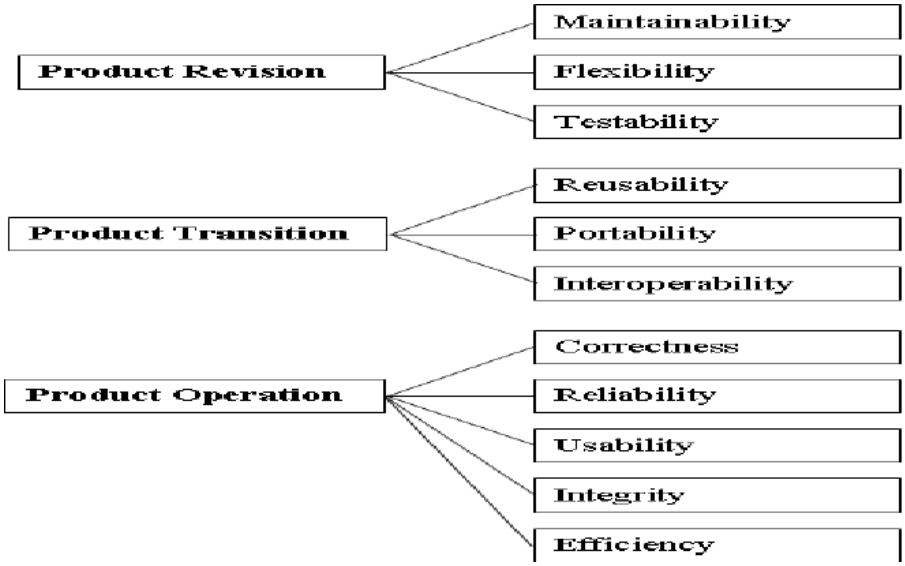
What is Software Quality Model?

✓ Software quality models were proposed to measure the quality of any software model. There are three widely accepted models when it comes to measuring software quality

- McCall's Quality Model
- Boehm quality model
- Dromey's quality model

<u>Mc call's Model</u>

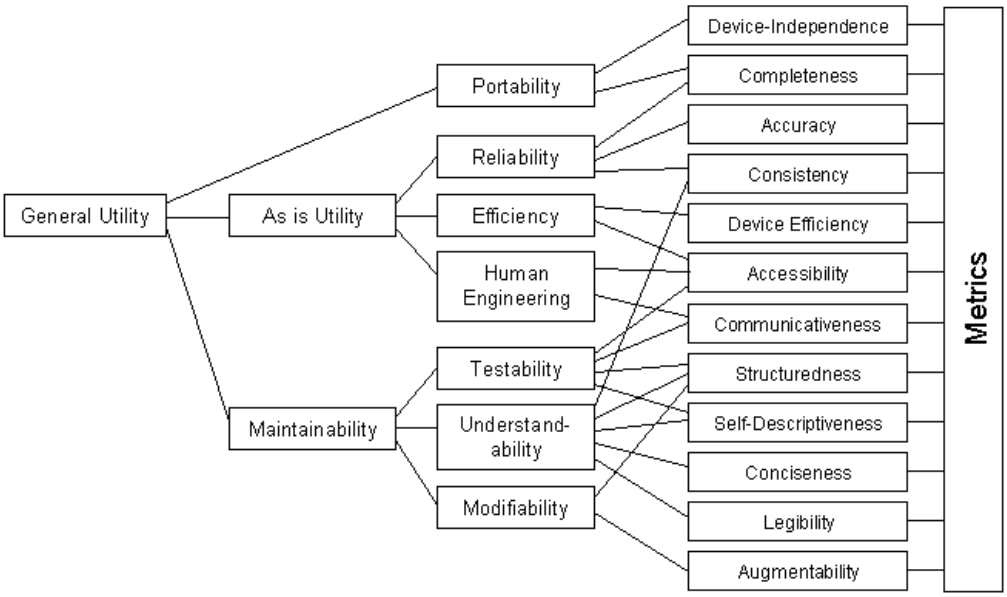
✓ Mc Call's model was first introduced in the US Airforce in the year 1977. The main intention of this model was to maintain harmony between users and developers.



Boehm Quality Model

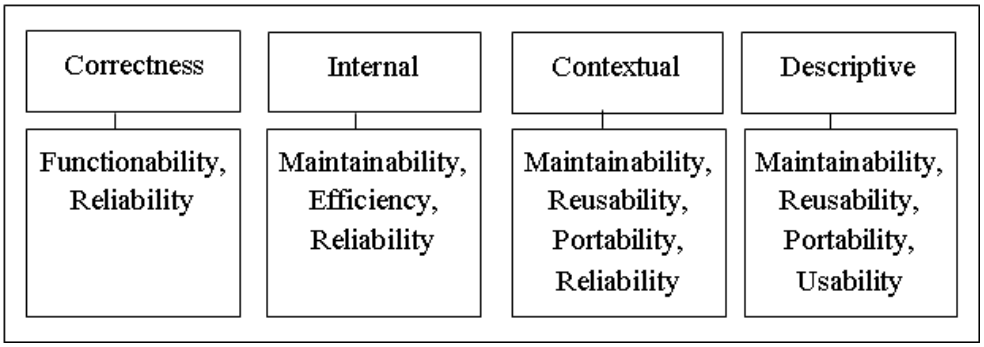
✓ Boehm model was introduced in the year 1978. It was a kind of hierarchical model that's structured around high-level characteristics. Boehm model measures software quality on the basis of certain

characteristics.



Dromey's quality model

- ✓ Dromey's model is mainly focused on the attributes and sub-attributes to connect properties of the software to the quality attributes. There are three principal elements to this model,
    - Product properties that affect the quality
    - High-level quality attributes
    - Linking the properties with quality attributes



Dromeys software quality model

How can software engineers acquire software quality?

Management plan – Have a clear idea about how the quality assurance process will be carried out through the project. Quality engineering activities required should also be set at the beginning along with team skill check.

Proper checkpoints – Checkpoints at required intervals should be set

How do we achieve Software quality?

- ✓ Achieving quality will ensure maximum profit for your software business. But the biggest hurdle is to achieve quality and here are some of the ways.
    - o Define characteristics that define quality for a product
    - o Decide how to measure each of that quality characteristic

- Set standards for each quality characteristic
- Do quality control with respect to the standards
- Find out the reasons that are hindering quality
- Make necessary improvements

What are software quality metrics?

- ✓ In any software project, you can go on building the code but at some point, you need to take a break and check if the work you are doing is right, if the process you followed is correct and so on. Metrics help you in exactly that.
- ✓ Metrics are pointers or numbers which help you understand the attributes of a **p**roduct, (like its complexity, its size, it's quality, etc.), the attributes of the **p**rocess (which can be used to improve the quality and speed of development) and the attributes of the **p**roject (which includes the number of resources, costs, productivity and timeline among others), popularly known as the three P's.

Why are software quality metrics important?

Software quality metrics are an indicator of the health of the product, process, and project. Good metrics with accurate data can help in

- Developing a strategy and giving the right direction to the process/project
- Recognizing the areas of focus
- Making strategic decisions
- Driving Performance and many others.

Important Software Quality Metrics

For any metrics to truly serve the purpose, there are 2 parts. One is the data accuracy and the second is metrics selection. All metrics will not be suitable for all processes and projects. So, the selection of the metrics needs to be done carefully. Let us now look at some very important and most commonly used Software Quality Metrics and how they are helpful in driving a better code

Defect Density

- ✓ The first measure of the quality of any products is the number of defects found and fixed. Though there a many "conditions apply" cases this is the first ballpark estimate of the quality of the software. The more the number of defects found, would be the quality of development is poor. So, the management should strive hard to improve development and do an RCA (Root Cause Analysis) to find why the quality is taking the hit.

Defect Density = No. of Defects Found / Size of AUT or module

Defect Removal Efficiency (DRE)

This is an important metric for assessing the effectiveness of a testing team. DRE is an indicator of the number of defects the tester or the testing team was able to remove from going into a production environment. Every quality team wants to ensure a 100% DRE.

DRE = A/(A+B) x 100

A – number of defects found before production

B – Number of defects found in production

Product operation quality factors

• **Correctness** The extent to which a program satisfies its specifications and fulfils the user's objectives.

• **Reliability** The extent to which a program can be expected to perform its intended function with required precision.

• **Efficiency** The amounts of computer resources required by the software.

• **Integrity** The extent to which access to software or data by unauthorized persons can be controlled.

• **Usability** The effort required to learn, operate, prepare input and interpret output.

Product revision quality factors

• **Maintainability** The effort required to locate and fix an error in an operational program.

• **Testability** The effort required to test a program to ensure it performs its intended function.

• **Flexibility** The effort required to modify an operational program. Product transition quality factors

• **Portability** The effort required to transfer a program from one hardware configuration and/or software system environment to another.

• **Reusability** The extent to which a program can be used in other applications.

• **Interoperability** The effort required to couple one system to another.

ISO 9126 SOFTWARE QUALITY CHARACTERISTICS

- ✓ ISO 9126 is an international standard for the evaluation of software. The standard is divided into four parts which addresses, respectively, the following subjects: quality model; external metrics; internal metrics; and quality in use metrics.
- ✓ ISO9126–1 represents the latest (and ongoing) research into characterizing software for the purposes of software quality control, software quality assurance and software process improvement (SPI). The ISO 9126 documentation itself, from the official ISO 9126 documentation, can only be purchased and is subject to copyright. SQA.net only reproduces the basic structure of the ISO 9126 standard and any descriptions, commentary or guidance are original material based on public domain information as well as our own experience.
- ✓ The ISO 9126–1 software quality model identifies **6 main quality characteristics**, namely:
  - Functionality
  - Reliability
  - Usability
  - Efficiency
  - Maintainability
  - Portability

- ✓ These characteristics are broken down into sub characteristics. It is at the sub characteristic level that measurement for SPI will occur. The main characteristics of the ISO9126–1 quality model, can be defined as follows:

Functionality

- ✓ Functionality is the essential purpose of any product or service. For certain items this is relatively easy to define, for example *a ship's anchor* has the function of holding a ship at a given location.
- ✓ The more functions a product has, e.g. an ATM machine, then the more complicated it becomes to define its functionality. For software a list of functions can be specified, i.e. a sales order processing system should be able to *record customer information* so that it can be used to reference a sales order. A sales order system should also provide the following functions:
  - Record sales order product, price and quantity.
  - Calculate total price.
  - Calculate appropriate sales tax.
  - Calculate date available to ship, based on inventory.
  - Generate purchase orders when stock falls below a given threshold.

Reliability

- ✓ Once a software system is functioning, as **specified, and delivered the reliability characteristic defines the capability of the system to** maintain its service provision under defined conditions for defined periods of time. One aspect of this characteristic is *fault tolerance* that is the ability of a system to withstand component failure. For example, if the network goes down for 20 seconds then comes back the system should be able to recover and continue functioning.

Usability

- ✓ Usability only exists with regard to functionality and refers to **the ease of use for a given function.** For example, a function of an ATM machine is to dispense cash as requested. Placing common amounts on the screen for selection, i.e. $20.00, $40.00, $100.00 etc., does not impact the function of the ATM but addresses the Usability of the function. The ability to learn how to use a system (learnability) is also a major sub characteristic of usability.

Efficiency

- ✓ This characteristic is concerned with the **system resources used when providing the required functionality**. The amount of disk space, memory, network etc. provides a good indication of this characteristic. As with a number of these characteristics, there are overlaps. For example, the usability of a system is influenced by the system's Performance, in that if a system takes 3 hours to respond the system would not be easy to use although the essential issue is a performance or efficiency characteristic.

Maintainability

- ✓ The ability to **identify and fix a fault within a software component** is what the maintainability characteristic addresses. In other software quality models this characteristic is referenced as supportability. Maintainability is impacted by code readability or complexity as well as modularization. Anything that helps with identifying the cause of a fault and then fixing the fault is the concern of maintainability. Also, the ability to verify (or test) a system, i.e. testability, is one of the sub characteristics of maintainability.

Portability

- ✓ This characteristic refers to **how well the software can adopt to changes in its environment** or with its requirements. The sub characteristics of this characteristic includes adaptability. Object oriented design and implementation practices can contribute to the extent to which this characteristic is present in a given system.

UNIT - II (SOFTWARE EVALUVATION AND COSTING)

STRATEGIC ASSESSMENT

Programme management

- ✓ It is being increasingly recognized that individual projects need to be seen as components of a programme and should be evaluated and managed as such. A programme, in this context, is a collection of projects that all contribute to the same overall organizational goals.
- ✓ Effective programme management requires that there is a **well-defined programme goal** and that all the organization's projects are selected and tuned to contribute to this goal. A project must be evaluated according to how it contributes to this programme goal and its viability, timing, resourcing and final worth can be affected by the programme as a whole.
- ✓ It is to be expected that the value of any project is increased by the fact that it is part of a programme - the whole, as they say, being greater than the sum of the parts.
- ✓ In order to carry out a successful strategic assessment of a potential project there should therefore be a strategic plan clearly defining the organization's objectives. This provides the context for defining the programme and programme goals and, hence, the context for assessing the individual project.
- ✓ It is likely, particularly in a large organization, that there will be an organizational structure for programme management and it will be, for example, the programme director and programme executive, rather than, say, a project manager, who will be responsible for the strategic assessment of a proposed project.
- ✓ Even where there is no explicitly defined programme, any proposed project must be evaluated within the context of the organization's overall business objectives.

✓ Moreover, any potential software system will form part of the user organization's overall information system and must be evaluated within the context of the existing information system and the organization's information strategy. Table 2.1 illustrates typical issues that must be addressed as part of the strategic assessment of a project.

Table 3.1 *Typical issues and questions to be considered during strategic assessment*

Issue	*Typical questions*
Objectives	How will the proposed system contribute to the organization's stated objectives? How, for example, might it contribute to an increase in market share?
IS plan	How does the proposed system fit into the IS plan? Which existing system(s) will it replace/interface with? How will it interact with systems proposed for later development?
Organization structure	What effect will the new system have on the existing departmental and organization structure? Will, for example, a new sales order processing system overlap existing sales and stock control functions?
MIS	What information will the system provide and at what levels in the organization? In what ways will it complement or enhance existing management information systems?
Personnel	In what way will the proposed system affect manning levels and the existing employee skill base? What are the implications for the organization's overall policy on staff development?
Image	What, if any, will be the effect on customers' attitudes towards the organization? Will the adoption of, say, automated systems conflict with the objectives of providing a friendly service?

✓ Where a well-defined information systems strategy does not exist, system development and the assessment of project proposals will be based on a more piecemeal approach - each project being individually assessed early in its life cycle. In such cases it is likely that cost-benefit analysis will have more importance and some of the questions of Table 3.1 will be more difficult to answer.

Portfolio management

✓ Where an organization such as a **software house is developing a software system,** they could be asked to carry out a strategic and operational assessment on behalf of the customer. Whether or not this should be the case, they will require an assessment of any proposed project themselves.

✓ They will need to ensure that carrying out the development of a system is consistent with their own strategic plan - it is unlikely, for example, that a software house specializing in financial and accounting systems would wish to undertake development of a factory control system unless their strategic plan placed an emphasis on diversification.

✓ The proposed project will form part of a portfolio of ongoing and planned projects and the selection of projects must take account of the possible effects on other projects in the portfolio (competition for resources, for example) and the overall portfolio profile (for example, specialization versus diversification).

Technical assessment

✓ Technical assessment of a proposed system consists of evaluating the required functionality against the hardware and software available.

✓ Where an organization has a strategic information systems plan, this is likely to place limitations on the nature of solutions that might be considered. The constraints will, of course, influence the cost of the solution and this must be taken into account in the cost benefit analysis.

COST BENEFIT ANALYSIS

- ✓ The most common way of carrying out an economic assessment of a proposed information system, or other development, is by comparing the expected costs of development and operation of the system with the benefits of having it in place.
- ✓ Assessment is based upon the question of whether the estimated costs are exceeded by the estimated income and other benefits. Additionally, it is usually necessary to ask whether or not the project under consideration is the best of a number of options. There might be more candidate projects than can be undertaken at any one time and, in any case, projects will need to be prioritized so that any scarce resources may be allocated effectively.
- ✓ The standard way of evaluating the economic benefits of any project is to carry out a cost-benefit analysis, which consists of two steps.
    - **Identifying and estimating all of the costs and benefits of carrying out the project** This includes development costs of the system, the operating costs and the benefits that are expected to accrue from the operation of the system. Where the proposed system is replacing an existing one, these estimates should reflect the costs and benefits due to the new system. A sales order processing system, for example, could not claim to benefit an organization by the total value of sales - only by the increase due to the use of the new system.
    - **Expressing these costs and benefits in common units** We must evaluate the net benefit, which is the difference between the total benefit and the total cost. To do this, we must express each cost and each benefit in monetary terms.
- ✓ Most costs are relatively easy to identify and quantify in approximate monetary. It is helpful to categorize costs according to where they originate in the life of the project.
    - **Development costs -** include the salaries and other employment costs of the staff involved in the development project and all associated costs.
    - **Setup costs -** include the costs of putting the system into place. These consist mainly of the costs of any new hardware and ancillary equipment but will also include costs of file conversion, recruitment and staff training.
    - **Operational costs -** consist of the costs of operating the system once it has been installed.
- ✓ Benefits, on the other hand, are often quite difficult to quantify in monetary terms even once they have been identified. Benefits may be categorized as follows.
    - **Direct benefits -** these accrue directly from the operation of the proposed system. These could, for example, include the reduction in salary bills through the introduction of a new, computerized system.

- **Assessable indirect benefits -** these are generally secondary benefits, such as increased accuracy through the introduction of a more user-friendly screen design where we might be able to estimate the reduction in errors, and hence costs, of the proposed system.
- **Intangible benefits -** these are generally longer term or benefits that are considered very difficult to quantify. Enhanced job interest can lead to reduced staff turnover and, hence, lower recruitment costs.

Any project that shows an excess of benefits over costs is clearly worth considering for implementation. However, as we shall see later, it is not a sufficient justification for going ahead: we might not be able to afford the costs; there might be even better projects we could allocate our resources to instead; the project might be too risky.

CASH FLOW FORECASTING

- ✓ As important as estimating the overall costs and benefits of a project is the forecasting of the cash flows that will take place and their timing. A cash flow forecast will indicate when expenditure and income will take place (Figure 3.2).

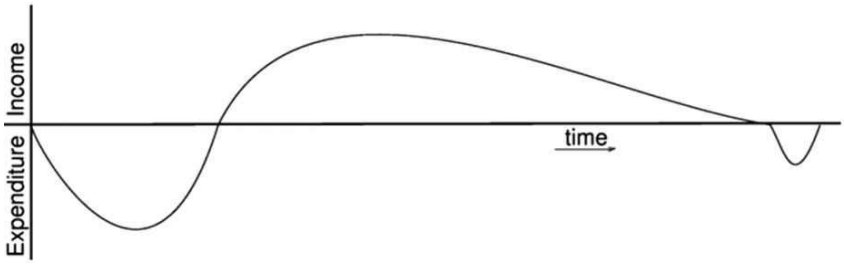


Figure 3.2 Typical product life cycle cash flow.

- ✓ We need to spend money, such as staff wages, during the development stages of a project. Such expenditure cannot be deferred until income is received (either from using the software if it is being developed for in-house use or from selling it). It is important that we know that we can fund the development expenditure either from the company's own resources or by borrowing from the bank.
- ✓ In any event, it is vital to have some forecast of when expenditure such as the payment of salaries and bank interest will take place and when any income is to be expected, such as payment on completion or, possibly, stage payments.
- ✓ Accurate cash flow forecasting is not easy, as it generally needs to be done early in the project's life cycle (at least before any significant expenditure is committed) and many items to be estimated (particularly the benefits of using software or decommissioning costs) might be some years in the future.
- ✓ When estimating future cash flows, it is usual to ignore the effects of inflation. Trying to forecast the effects of inflation increases the uncertainty of the forecasts. Moreover, if expenditure is increased due to inflation it is likely that income will increase proportionately.

- ✓ However, measures to deal with increases in costs where work is being done for an external customer must be in place - for example index linked prices where work involves use of raw.
- ✓ Table 3.2 illustrates cash flow forecasts for four projects. In each case it is assumed that the cash flows take place at the end of each year. For short-term projects or where candidate projects demonstrate significant seasonal cash flow patterns it can be advisable to produce quarterly, or even monthly, cash flow forecasts.

Table 3.2 *Four project cash flow projections – figures are end of year totals (£)*

Year	*Project 1*	*Project 2*	*Project 3*	*Project 4*
0	−100,000	−1,000,000	−100,000	−120,000
1	10,000	200,000	30,000	30,000
2	10,000	200,000	30,000	30,000
3	10,000	200,000	30,000	30,000
4	20,000	200,000	30,000	30,000
5	100,000	300,000	30,000	75,000
Net profit	50,000	100,000	50,000	75,000

COST BENEFIT EVALUATION TECHNIQUES

- ✓ We would consider proceeding with a project only where the benefits outweigh the costs. However, in order to choose among projects, we need to take into account the timing of the costs and benefits as well as the benefits relative to the size of the investment. In the following sections we will take a brief look at some common methods for comparing projects on the basis of their cash flow forecasts.

Net profit

- ✓ The net profit of a project is the difference between the total costs and the total income over the life of the project. Project 2 in Table 3.2 shows the greatest net profit but this is at the expense of a large investment. Indeed, if we had £lm to invest, we might undertake all of the other three projects and obtain an even greater net profit. Note also, that all projects contain an element of risk and we might not be prepared to risk £1 m.
- ✓ Moreover, the simple net profit takes no account of the timing of the cash flows. Projects 1 and 3 each have a net profit of £50,000 and therefore, according to this selection criterion, would be equally preferable. The bulk of the income occurs late in the life of project 1, whereas project 3 returns a steady income throughout its life. Having to wait for a return has the disadvantage that the investment must be funded for longer.
- ✓ Add to that the fact that, other things being equal, estimates in the more distant future are less reliable that short-term estimates and we can see that the two projects are not equally preferable.

Payback period

- ✓ The payback period is the time taken to break even or pay back the initial investment. Normally, the project with the shortest payback period will be chosen on the basis that an organization will wish to minimize the time that a project is 'in debt'.
- ✓ The advantage of the payback period is that it is simple to calculate and is not particularly sensitive to small forecasting errors. Its disadvantage as a selection technique is that it ignores the overall

profitability of the project - in fact, it totally ignores any income (or expenditure) once the project has broken even. Thus, the fact that projects 2 and 4 are, overall, more profitable than project 3 is ignored.

Return on investment

- ✓ The return on investment (ROI), also known as the accounting rate of return (ARR), provides a way of comparing the net profitability to the investment required. There are some variations on the formula used to calculate the return on investment but a straightforward common version is

$$\text{ROI} = \frac{\text{average annual profit}}{\text{total investment}} \times 100$$

- ✓ The main difficulty with NPV for deciding between projects is selecting an appropriate discount rate. Some organizations have a standard rate but, where this is not the case, then the discount rate should be chosen to reflect available interest rates (borrowing costs where the project must be funded from loans) plus some premium to reflect the fact that software projects are inherently more risky than lending money to a bank. The exact discount rate is normally less important than ensuring that the same discount rate is used for all projects being compared. However, it is important to check that the ranking of projects is not sensitive to small changes in the discount rate - have a look at the following exercise.

Internal rate of return

- ✓ One disadvantage of NPV as a measure of profitability is that, although it may be used to compare projects, it might not be directly comparable with earnings from other investments or the costs of borrowing capital. Such costs are usually quoted as a percentage interest rate.
- ✓ The internal rate of return (IRR) attempts to provide a profitability measure as a percentage return that is directly comparable with interest rates. Thus, a project that showed an estimated IRR of 10% would be worthwhile if the capital could be borrowed for less than 10% or if the capital could not be invested elsewhere for a return greater than 10%.
- ✓ The IRR is calculated as that percentage discount rate that would produce an NPV of zero. It is most easily calculated using a spreadsheet or other computer program that provides functions for calculating the IRR. Microsoft Excel and Lotus, for example, both provide IRR functions which, provided with an initial guess or seed value (which may be zero), will search for and return an IRR.
- ✓ Manually, it must be calculated by trial-and-error or estimated using the technique illustrated in Figure 3.3. This technique consists of guessing two values.

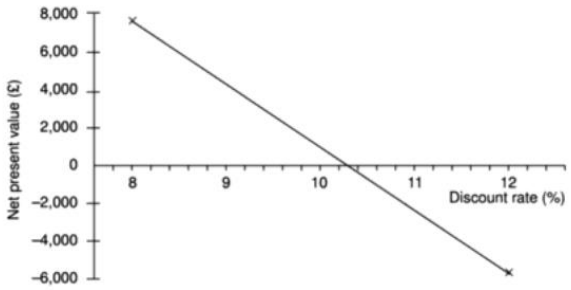


Figure 3.3 *Estimating the internal rate of return for project 1.*

RISK MANAGEMENT IN SOFTWARE DEVELOPMENT

- ✓ Every project involves risk of some form. When assessing and planning a project, we are concerned with the risk of the project's not meeting its objectives. We are concerned with taking risk into account when deciding whether to proceed with a proposed project.

Risk identification and ranking

- ✓ In any project evaluation we should attempt to identify the risks and quantify their potential effects. One common approach to risk analysis is to construct a project risk matrix utilizing a checklist of possible risks and to classify each risk according to its relative importance and likelihood.
- ✓ Note that the importance and likelihood need to be separately assessed - we might be less concerned with something that, although serious, is very unlikely to occur than with something less serious that is almost certain.
- ✓ Table 3.7 illustrates a basic project risk matrix listing some of the risks that might be considered for a project, with their importance and likelihood classified as high (H), medium (M), low (L) or exceedingly unlikely (—). So that projects may be compared the list of risks must be the same for each project being assessed. It is likely, in reality, that it would be somewhat longer than shown and more precisely defined.
- ✓ The project risk matrix may be used as a way of evaluating projects (those with high risks being less favoured) or as a means of identifying and ranking the risks for a specific project. In Chapter 7 we shall consider a method for scoring the importance and likelihood of risks that may be used in conjunction with the project risk matrix to score and rank projects.

Risk and net present value

- ✓ Where a project is relatively risky it is common practice to use a higher discount rate to calculate net present value. This addition or risk premium, might, for example, be an additional 2% for a reasonably safe project or 5% for a fairly risky one. Projects may be categorized as high, medium or low risk using a scoring method and risk premiums designated for each category. The premiums, even if arbitrary, provide a consistent method of taking risk into account.

Cost-benefit analysis

- ✓ A rather more sophisticated approach to the evaluation of risk is to consider each possible outcome and estimate the probability of its occurring and the corresponding value of the outcome. Rather

than a single cash flow forecast for a project, we will then have a set of cash flow forecasts, each with an associated probability of occurring. The value of the project is then obtained by summing the cost or benefit for each possible outcome weighted by its corresponding probability.

A fragment of a basic project risk matrix

Risk	Importance	Likelihood
Software never completed or delivered	H	-
Project cancelled after design stage	H	-
Software delivered late	M	M
Development budget exceeded < 20%	L	M
Development budget exceeded > 20%	M	L
Maintenance costs higher than estimated	L	L
Response time targets not met	L	H

- ✓ This approach is frequently used in the evaluation of large projects such as the building of new motorways, where variables such as future traffic volumes, and hence the total benefit of shorter journey times, are subject to uncertainty. The technique does, of course, rely on our being able to assign probabilities of occurrence to each scenario and, without extensive study, this can be difficult.
- ✓ When used to evaluate a single project, the cost-benefit approach, by 'averaging out' the effects of the different scenarios, does not take account an organization's reluctance to risk damaging outcomes. Because of this, where overall profitability is the primary concern, it is often considered more appropriate for the evaluation of a portfolio of projects.

Risk profile analysis

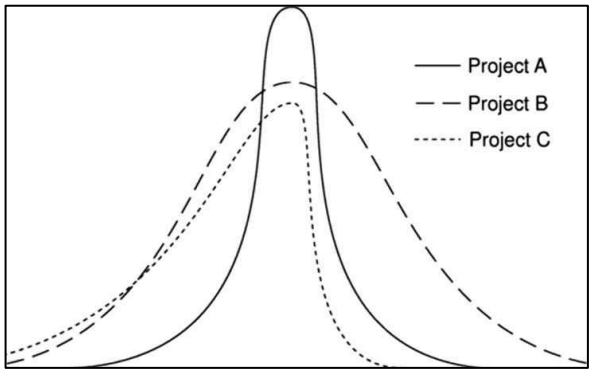
- ✓ An approach which attempts to overcome some of the objections to cost-benefit averaging is the construction of risk profiles using sensitivity analysis.
- ✓ This involves varying each of the parameters that affect the project's cost or benefits to ascertain how sensitive the project's profitability is to each factor. We might, for example, vary one of our original estimates by plus or minus 5% and recalculate the expected costs and benefits for the project. By repeating this exercise for each of our estimates in turn we can evaluate the sensitivity of the project to each factor.
- ✓ By studying the results of a sensitivity analysis we can identify those factors that are most important to the success of the project. We then need to decide whether we can exercise greater control over them or otherwise mitigate their effects.
- ✓ If neither is the case, then we must live with the risk or abandon the project. For an explanation of the Sensitivity analysis demands that we vary each factor one at a time. It does not Monte Carlo

technique easily allow us to consider the effects of combinations of circumstances, neither see any textbook on does it evaluate the chances of a particular outcome occurring.

- ✓ In order to do this operational research. we need to use a more sophisticated tool such as Monte Carlo simulation. There are a number of risk analysis applications available (such as @Risk from Palisade) that use Monte Carlo simulation and produce risk profiles.
- ✓ Projects may be compared as in Figure 3.4, which compares three projects with the same expected profitability. Project A is unlikely to depart far from this expected value compared to project B, which exhibits a larger variance. Both of these have symmetric profiles, which contrast with project C. Project C has a skewed distribution, which indicates that although it is unlikely ever to be much more profitable than expected, it is quite likely to be far worse.

Using decision trees

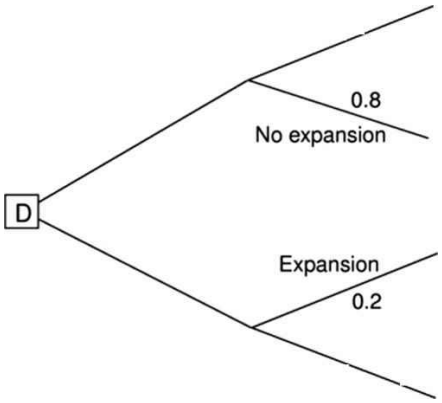
- ✓ The approaches to risk analysis discussed previously rather assume that we are passive bystanders allowing nature to take its own course - the best we can do is to reject over-risky projects or choose those with the best risk profile. There are many situations, however, where we can evaluate whether a risk is important and, if it is, indicate a suitable course of action.
- ✓ Many such decisions will limit or affect future options and, at any point, it is important to be able to see into the future to assess how a decision will affect the future profitability of the project.
- ✓ Prior to giving Amanda the job of extending their invoicing system, IOE must consider the alternative of completely replacing the existing system - which they will have to do at some point in the future.
- ✓ The decision largely rests upon the rate at which their equipment maintenance business expands - if their market share significantly increases (which they believe will happen if rumours of a competitor's imminent bankruptcy are fulfilled) the existing system might need to be replaced within 2 years.
- ✓ Not replacing the system in time could be an expensive option as it could lead to lost revenue if they cannot cope with the increase in invoicing demand. Replacing it immediately will, however, be expensive as it will mean deferring other projects that have already been scheduled.
- ✓ All three projects have the same expected profitability. The profitability of project A is unlikely to depart greatly from its expected value (indicated by the vertical axis) compared to the likely variations for project B. Project A is therefore less risky than project B.



- ✓ They have calculated that extending the existing system will have an NPV of £57,000, although if the market expands significantly, this will be turned

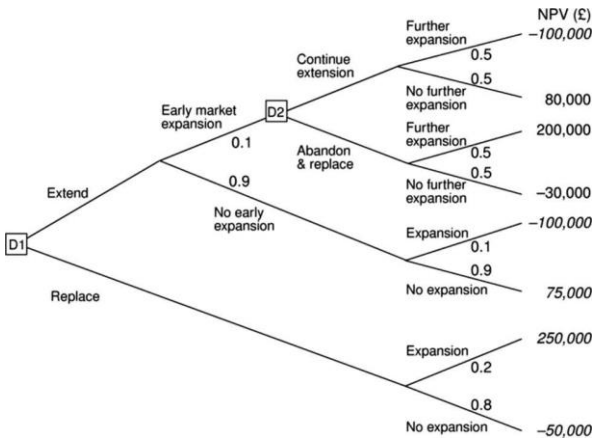
into a loss with an NPV of -£100,000 due to lost revenue. If the market does expand, replacing the system now has an NPV of £250,000 due to the benefits of being able to handle increased sales and other benefits such as improved management information. If sales do not increase, however, the benefits will be severely reduced and the project will suffer a loss with an NPV of -£50,000.

- ✓ The company estimate the likelihood of the market increasing significantly at 20% - and, hence, the probability that it will not increase as 80%. This scenario can be represented as a tree structure as shown in Figure 3.5.
- ✓ The analysis of a decision tree consists of evaluating the expected benefit of taking each path from a decision point (denoted by D in Figure 3.5). The expected value of each path is the sum of the value of each possible outcome multiplied by its probability of occurrence. The expected value of extending the system is therefore £40,000 (75,000 x 0.8 - 100,000 x 0.2) and the expected value of replacing the system £ 10,000 (250,000 x 0.2 - 50,000 x 0.8). IOE should therefore choose the option of extending the existing system.
- ✓ This example illustrates the use of a decision tree to evaluate a simple decision at the start of a project. One of the great advantages of using decision trees to model and analyse problems is the ease with which they can be extended. Figure 3.6 illustrates an extended version of Amanda's decision tree, which includes the possibility of a later decision should they decide to extend the system and then find there is an early market expansion.



The net present values shown in italic are those identified in Amanda's original decision tree.

An extension to Amanda's decision tree.



SELECTION OF AN APPROPRIATE PROJECT APPROACH

Choosing technologies

- ✓ An outcome of project analysis will be the selection of the most appropriate methodologies and technologies. Methodologies include techniques like the various flavours of object-oriented (OO) development, SSADM and JSP (Jackson Structured Programming) while technologies might include an appropriate application-building environment, or the use of knowledge-based system tools.
- ✓ As well as the products and activities, the chosen technology will influence the following aspects of a project:
  - the training requirements for development staff;
  - the types of staff to be recruited;
  - the development environment - both hardware and software;
  - system maintenance arrangements.
- ✓ We are now going to describe some of the steps of project analysis.

Identify project as either objectives-driven or product-driven

- ✓ You will recall from Chapter 1 that we distinguished between objectives-driven and product-driven projects. Very often a product-driven project will have been preceded by an objectives-driven project which chose the general software solution that is to be implemented.
- ✓ There will be cases where things are so vague that even the objectives of the project are uncertain or are the subject of disagreement. People may be experiencing a lot of problems but no-one knows exactly what the solution to the problems might be. It could be that the IT specialists can provide help in some places but assistance from other specialisms is needed in others. In these kinds of situation, a soft systems approach might need to be considered.

Analyse other project characteristics

- ✓ The sorts of question that would need to be asked include the following.
  - **Is a data orientated or a control orientated system to be implemented?'** Data orientated* systems generally mean information systems that will have a considerable database. 'Control orientated' systems refer to embedded control systems. These days it is not uncommon to have systems with components of both types.
  - **Will the software that is to be produced be a general package or application specific?** An example of a general package would be a spreadsheet or a word processing package. An application specific package could be, for example, an airline seat reservation system.
  - **Is the system to be implemented of a particular type for which specific tools have been developed?** For example:

    • does it involve concurrent processing? - if so the use of techniques appropriate to the analysis and design of such systems would be considered;

- will the system to he created he knowledge-based? - expert systems have a set of rules which result in some 'expert advice' when applied to a problem domain (sets of methods and tools have been developed to assist in the creation of such systems); or
- will the system to be produced make heavy use of computer graphics?

- **Is the system to be created safety-critical?** For instance, could a malfunction in the system endanger human life?
- **What is the nature of the hardware/software environment in which the system will operate?** It might be that the environment in which the final software will operate is different from that in which it will be developed. Embedded software may be developed on a large development machine that has lots of supporting software tools in the way of compilers, debuggers, static analyzers and so on, but might then be down-loaded to a small processor in the larger engineered product. A system destined for a personal computer will need a different approach to one destined for a main-frame or a client-server environment.

Identify high level project risks

- ✓ When we first embark on a project, we might be expected to work out elaborate plans even though we are at least partially ignorant of many important factors that will affect the project.
- ✓ For example, until we do a detailed investigation of the users' requirements, we will not be able to estimate how much effort will be needed to build a system to meet those requirements.
- ✓ The greater the uncertainties at the beginning of the project, the greater the risk that the project will be unsuccessful. Once we recognize a particular area of uncertainty we can, however, take steps to reduce its uncertainty.
- ✓ One suggestion is that uncertainty can be associated with the products, processes, or resources associated with the project.
  - **Product uncertainty** Here we ask how well the requirements are understood. It might be that the users themselves are uncertain about what a proposed information system is to do. The government, say, might introduce a new form of taxation but the way this is going to operate in detail will not be known until a certain amount of case law has been built up. Some environments can change so quickly that what was a precise and valid statement of requirements rapidly becomes out of date.
  - **Process uncertainty** It might be that the project under consideration is the first OMT is an object-oriented where an organization has tried to use a method, such as SSADM or OMT, that is design approach, new to them. Perhaps a new application building tool is being used. Any change in the way that the systems are developed is going to introduce uncertainty.
  - **Resource uncertainty** the main area of uncertainty here will almost surely be the availability of staff of the right ability and experience. A major influence on the degree of

uncertainty in a project will be the sheer size of a project. The larger the number of resources needed or the longer the duration of the project, the more inherently risky it is likely to be.

Take into account user requirements concerning implementation

- ✓ A user organization lays down standards that have to be adopted by any contractor providing software for them. For example, the UK Civil Service favors the SSADM standard where information systems are being developed.

It is common for organizations to specify that suppliers of software have BS EN 9001:1994 or Tick IT accreditation. This will affect the way projects are conducted.

Select general life cycle approach

- ✓ **Control systems** A real-time system will have to be implemented using an appropriate methodology, for example, Mascot. Real-time systems that employ concurrent processing will use techniques such as Petri nets.
- ✓ **Information systems** Similarly, an information system will need a methodology, such as SSADM or Information Engineering, that matches that type of environment. SSADM will be especially appropriate where the project will employ a large number of development staff whose work will need to be coordinated: the method lays down in detail what needs to be done and what products need to be created at each step. Team members would therefore know exactly what is expected of them.
- ✓ **General applications** Where the software to be produced is for the general market rather than for a specific application and user, then a methodology such as SSADM would have to be thought about very carefully. This is because the framers of the method make the assumption that a specific user exists. Some parts in the method also assume that there is an existing system that can be analyzed to yield the logical features of the new, computer-based, system.
- ✓ **Specialized techniques** These have been invented to expedite the development of, for example, knowledge-based systems where there are a number of specialized tools and logic-based programming languages that can be used to implement this type of system. Similarly, a number of specialized techniques and tools have been developed to assist in the development of graphics-based systems.
- ✓ **Hardware environment** the environment in which the system is to operate can put constraints on the way it is to be implemented. For instance, the need for a fast response time or for the software to take up only a small part of computer memory may mean that only low-level programming languages can be used - particularly in real-time and embedded systems.
- ✓ **Safety-critical systems** Where safety and reliability are of the essence, it might be possible to justify the additional expense of a formal specification using a notation such as Z or VDM. Really

critical systems call for expensive measures such as having independent teams develop parallel systems with the same functionality. The parallel systems can then run concurrently when the application is in operation so that the results of each of the parallel systems can be crosschecked.

- ✓ **Imprecise requirements** Uncertainties or a novel hardware/software platform may mean that a prototyping approach should be considered. If the environment in which the system is to be implemented is a rapidly changing one, then serious consideration would need to be given to incremental delivery. If the users have uncertain objectives in connection with the project, then a soft systems approach might be desirable.

Choice of process models

- ✓ The word 'process' is sometimes used to emphasize the idea of a system in action. In order to achieve an outcome, the system will have to execute one or more activities: this is its process. This idea can be applied to the development of computer-based systems where a number of interrelated activities have to be undertaken to create a final product. These activities can be organized in different ways and we can call these process models.
- ✓ A major part of the planning will be the choosing of the development methods to be used and the slotting of these into an overall process model.
- ✓ The planner needs not only to select methods but also to specify how the method is to be applied. With methods such as SSADM, there is a considerable degree of choice about how it is to be applied: not all parts of SSADM are compulsory. Many student projects have the rather basic failing that at the planning stage they claim that, say, SSADM is to be used: in the event, all that is produced are a few SSADM fragments such as a top-level data flow diagram and a preliminary logical data structure diagram. If this is all the particular project requires, it should be stated at the outset.

Structured methods

- ✓ Although some "object-oriented' specialists may object(!), we include the 00 approach as a structured method - after all, we hope it is not unstructured. Structured methods are made up of sets of steps and rules, which, when applied produce system products such as data flow diagrams. Each of these products is carefully documented.
- ✓ Such methods are often time consuming compared to more intuitive approaches and this implies some additional cost. The pay-off is such things as a less error prone and more maintainable final system. This balance of costs and benefits is more likely to be justified on a large project involving many developers and users. This is not to say that smaller projects cannot justify the use of such methods.

UNIT – 3

SOFTWARE EFFORT ESTIMATION

Where are estimates done?

- ✓ Estimates are carried out at various stages of a software project. At each stage, the reasons for the estimate and the methods used will vary.

Strategic planning: The costs of computerizing potential applications as well as the benefits of doing so might need to be estimated to help decide what priority to give to each project. Such estimates might also influence the numbers of various, types of development staff to be recruited by the organization.

Feasibility study: This ascertains that the benefits of the potential system will justify the costs.

System specification: Most system development methodologies usefully distinguish between the definition of the users' requirements and the design that documents how those requirements are to be fulfilled. The effort needed to implement different design proposals will need to be estimated. Estimates at the design stage will also confirm that the feasibility study is still valid, taking into account all that has been learnt during detailed requirements analysis.

Project planning: As the planning and implementation of the project progresses to greater levels of detail, more detailed estimates of smaller work components will be made. As well as confirming the earlier and more broad-brush estimates, these will help answer questions about, for example, when staff will have completed particular tasks and be available for new activities. Two general points can be made here:

• as the project proceeds, so the accuracy of the estimates should improve as knowledge about the nature of the project increases;

• conventional wisdom is that at the beginning of the project the user requirement (that is. a logical model of the required system) is of paramount importance and that premature consideration of the physical implementation is to be avoided. In order to do an estimate, however, the estimator will have to speculate about this physical implementation, for instance about the number of software modules to be written.

Evaluation of suppliers' proposals: In the case of the IOE maintenance group accounts subsystem, for example. IOE might consider putting the actual system-building out to tender. Staff in the software houses that are considering a bid would need to scrutinize the system specification and produce estimates on which to base proposals. Amanda might still be required to carry out her own estimate to help judge the bids received. IOE might wish to question a proposal that seems too low: they might wonder, for example, whether the proposer had properly understood the requirements. If. on the other hand, the bids seem too high, they might reconsider in-house development.

Problems with over and underestimates

- ✓ A project leader such as Amanda will need to be aware that the estimate itself, if known to the development team, will influence the time required to implement the system. An over-estimate might cause the project to take longer than it would otherwise. This can be explained by the application of two 'laws'.

Parkinson's Law 'Work expands to fill the time available', which implies that given an easy target staff will work less hard.

Brooks' Law The effort required to implement a project will go up disproportionately with the number of staff assigned to the project. As the project team grows in size so will the effort that has to go into management, co-ordination and communication. This has given rise, in extreme cases, to the notion of Brooks'.

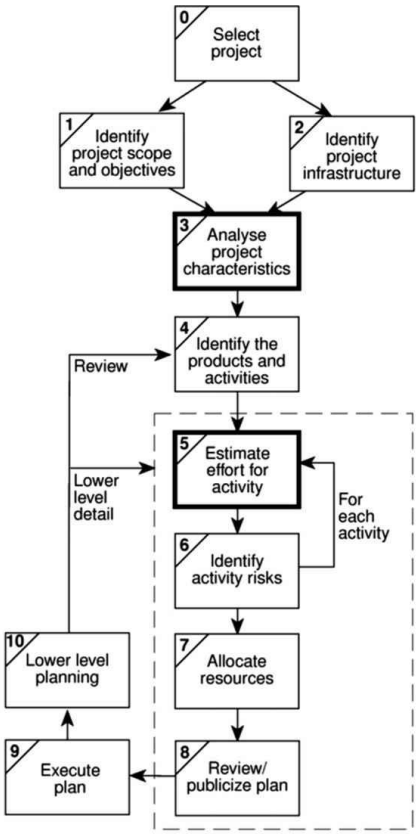


Figure 5.1 *Software estimation takes place in Steps 3 and 5 in particular.*

- ✓ Law: 'putting more people on a late job makes it later'. If there is an over-estimate of the effort required then this might lead to more staff being allocated than are needed and managerial overheads will be increased. This is more likely to be of significance with large projects.
- ✓ Some have suggested that while the under-estimated project might not be completed on time or to cost, it might still be implemented in a shorter time than a project with a more generous estimate. There must, however, be limits to this phenomenon where all the slack in the project is taken up.
- ✓ The danger with the under-estimate is the effect on quality. Staff, particularly those with less experience, might respond to pressing deadlines by producing work which is sub-standard. Since we are into laws, this might be seen as a manifestation of Weinberg's zeroth law of reliability: 'if a system does not have to be reliable, it can meet any other objective'.
- ✓ In other words, if there is no need for the program actually to work, you can meet any programming deadline that might be set! Sub-standard work might only become visible at the later, testing, phases of a project, which are particularly difficult to control and where extensive rework can have catastrophic consequences for the project completion date.

- ✓ Because of the possible effects on the behaviour of development staff caused by the size of estimates, they might be artificially reduced by their managers to increase pressure on staff. This will work only where staff are unaware that this has been done.
- ✓ Research has found that motivation and morale are enhanced where targets are achievable. If, over a period of time, staff become aware that the targets set are unattainable and that projects are routinely not meeting their published targets, then this will help to destroy motivation.
- ✓ Furthermore, people like to think of themselves as winners and there is a general tendency to put success down to our own efforts, while failure is blamed on the organization.
- ✓ In the end, an estimate is not really a prediction, it is a management goal. Barry Boehm has suggested that if a software development cost is within 20% of the estimated cost estimate for the job then a good manager can turn it into a self-fulfilling prophecy.
- ✓ A project leader like Amanda will work hard to make the actual performance conform to the estimate.

The basis for software estimating

The need for historical data

- ✓ Nearly all estimating methods need information about how projects have been implemented in the past. However, care needs to be taken in judging the applicability of data to the estimator's own circumstances because of possible differences in environmental factors such as the programming languages used, the software tools available, the standards enforced and the experience of the staff.

Measure of work

- ✓ It is normally not possible to calculate directly the actual cost or time required to implement a project. The time taken to write a program might vary according to the competence or experience of the programmer. Implementation times might also vary because of environmental factors such as the software tools available. The usual practice is therefore to express the work content of the system to be implemented independently of effort, using a measure such as source lines of code (SLOC). The reader might also come across the abbreviation KLOC which refers to thousands of lines of code.

Complexity

- ✓ Two programs with the same KLOC will not necessarily take the same time to write, even if done by the same developer in the same environment. One program might be more complex. Because of this, SLOC estimates have to be modified to take complexity into account. Attempts have been made to find objective measures of complexity, but often it will depend on the subjective judgement of the estimator.

Software effort estimation techniques

- ✓ Barry Boehm, in his classic work on software effort models, identified the main ways of deriving estimates of software development effort as:

  • algorithmic models - which use 'effort drivers' representing characteristics of the target system and the implementation environment to predict effort;

  • expert judgement - where the advice of knowledgeable staff is solicited;

  • analogy - where a similar, completed, project is identified and its actual effort is used as a basis for the new project;

  • Parkinson - which identifies the staff effort available to do a project and uses that as the 'estimate';

  • price to win - where the 'estimate' is a figure that appears to be sufficiently low to win a contract;

  • top-down - where an overall estimate is formulated for the whole project and is then broken down into the effort required for component tasks;

  • bottom-up - where component tasks are identified and sized and these individual estimates are aggregated.
- ✓ Clearly, the 'Parkinson' method is not really an effort prediction method, but a method of setting the scope of a project. Similarly, 'price to win' is a way of deciding a price and not a prediction method. On these grounds, Boehm rejects them as prediction techniques although they might have some value as management techniques. There is, for example, a perfectly acceptable engineering practice of 'design to cost' which is one example of the broader approach of 'design by objectives'.
- ✓ We will now look at some of these techniques more closely. First we will examine the difference between top-down and bottom-up estimating.

Bottom-up estimating

- ✓ Estimating methods can be generally divided into bottom-up and top-down approaches. With the bottom-up approach, the estimator breaks the project into its component tasks and then estimates how much effort will be required to carry out each task.
- ✓ With a large project, the process of breaking down into tasks would be a repetitive one: each task would be analysed into its component sub-tasks and these in turn would be further analysed.
- ✓ This is repeated until you get to components that can be executed by a single person in about a week or two. The reader might wonder why this is not called a top-down approach: after all you are starting from the top and working down! Although this top-down analysis is an essential precursor to bottom-up estimating, it is really a separate one - that of producing a Work Breakdown Structure (WBS). The bottom-up part comes in adding up the calculated effort for each activity to get an overall estimate.
- ✓ The bottom-up approach is most appropriate at the later, more detailed, stages of project planning. If this method is used early on in the project cycle then the estimator will have to make some

assumptions about the characteristics of the final system, for example the number and size of software modules. These will be working assumptions that imply no commitment when it comes to the actual design of the system.

- ✓ Where a project is completely novel or there is no historical data available, the estimator would be well advised to use the bottom-up approach.

The top-down approach and parametric models

- ✓ The top-down approach is normally associated with parametric (or algorithmic) models. These may be explained using the analogy of estimating the cost of rebuilding a house. This would be of practical concern to a house-owner who needs sufficient insurance cover to allow for rebuilding the property if it were destroyed.
- ✓ Unless the house-owner happens to be in the building trade it is unlikely that he or she would be able to work out how many bricklayer-hours, how many carpenter-hours, electrician-hours and so on would be required. Insurance companies, however, produce convenient tables where the house-owner can find an estimate of rebuilding costs based on such parameters as the number of storeys and the floor space that a house has. This is a simple parametric model.
- ✓ The effort needed to implement a project will be related mainly to variables associated with characteristics of the final system. The form of the parametric model will normally be one or more formulae in the form:

effort = (system size) x (productivity rate)

- ✓ For example, system size might be in the form 'thousands of lines of code' (KLOC) and the productivity rate 40 days per KLOC. The values to be used will often be matters of subjective judgement.
- ✓ A model to forecast software development effort therefore has two key components. The first is a method of assessing the size of the software development task to be undertaken. The second assesses the rate of work at which the task can be done. For example, Amanda at IOE might estimate that the first software module to be constructed is 2 KLOC. She might then judge that if Kate undertook the development of the code, with her expertise she could work at a rate of 40 days per KLOC and complete the work in 2 x 40 days, that is, 80 days, while Ken, who is less experienced, would need 55 days per KLOC and take 2 x 55 that is, 110 days to complete the task.
- ✓ Some parametric models, such as that implied by function points, are focused on system or task size, while others, such are COCOMO, are more concerned with productivity factors.
- ✓ Having calculated the overall effort required, the problem is then to allocate proportions of that effort to the various activities within that project.
- ✓ The top-down and bottom-up approaches are not mutually exclusive. Project managers will probably try to get a number of different estimates from different people using different methods.

Some parts of an overall estimate could be derived using a top-down approach while other parts could be calculated using a bottom-up method.

Expert judgement

- ✓ This is asking someone who is knowledgeable about either the application area or the development environment to give an estimate of the effort needed to carry out a task. This method will most likely be used when estimating the effort needed to change an existing piece of software. The estimator would have to carry out some kind of impact analysis in order to judge the proportion of code that would be affected and from that derive an estimate. Someone already familiar with the software would be in the best position to do this.
- ✓ Some have suggested that expert judgement is simply a matter of guessing, but our own research has shown that experts tend to use a combination of an informal analogy approach where similar projects from the past are identified and bottom-up estimating.

Estimating by analogy

- ✓ The use of analogy is also called case-based reasoning. The estimator seeks out projects that have been completed (source cases) and that have similar characteristics to the new project (the target case). The effort that has been recorded for the matching source case can then be used as a base estimate for the target. The estimator should then try to identify any differences between the target and the source and make adjustments to the base estimate for the new project.
- ✓ This might be a good approach where you have information about some previous projects but not enough to draw generalized conclusions about what variables might make good size parameters.
- ✓ A problem here is how you actually identify the similarities and differences between the different systems. Attempts have been made to automate this process. One software application that has been developed to do this is ANGEL.
- ✓ This identifies the source case that is nearest the target by measuring the Euclidean distance between cases. The source case that is at the shortest Euclidean distance from the target is deemed to be the closest match. The Euclidean distance is calculated:

$$distance = \text{square-root of } ((target\_parameter_1 - source\_parameter_1)^2 + \ldots + (target\_parameter_n - source\_parameter_n)^2)$$

ACTIVITY PLANNING

- ✓ We looked at methods for forecasting the effort required for a project - both for the project as a **whole and for individual activities**. A detailed plan for the project, however, must also include a schedule indicating the start and completion times for each activity. This will enable us to:
  - ensure that the appropriate resources will be available precisely when required;
  - avoid different activities competing for the same resources at the same time;
  - produce a detailed schedule showing which staff carry out each activity;

- produce a detailed plan against which actual achievement may be measured;
- produce a timed cash flow forecast;
- replan the project during its life to correct drift from the target.

✓ To be effective, a plan must be stated as a set of targets, the achievement or non-achievement of which can be unambiguously measured. The activity plan does in this by providing a target start and completion date for each activity. within which each activity may be carried out).

✓ The starts and completions of activities must be clearly visible and this is one of the reasons why it is advisable to ensure that each and every project activity produces some tangible product or 'deliverable'.

✓ Monitoring the project's progress is then, at least in part, a case of ensuring that the products of each activity are delivered on time. As a project progresses it is unlikely that everything will go according to plan. Much of the job of project management concerns recognizing when something has gone wrong, identifying its causes and revising the plan to mitigate its effects.

✓ The activity plan should provide a means of evaluating the consequences of not meeting any of the activity target dates and guidance as to how the plan might most effectively be modified to bring the project back to target. We shall see that the activity plan may well also offer guidance as to which components of a project should be most closely monitored. This co-ordination will normally form part of Programme Management.

The objectives of activity planning

✓ In addition to providing project and resource schedules, activity planning aims to achieve a number of other objectives which may be summarized as follows.

Feasibility assessment Is the project possible within required timescales and resource constraints? It is not until we have constructed a detailed plan that we can forecast a completion date with any reasonable knowledge of its achievability. The fact that a project may have been estimated as requiring two work-years effort might not mean that it would be feasible to complete it within, say, three months were eight people to work on it - that will depend upon the availability of staff and the degree to which activities may be undertaken in parallel.

Resource allocation What are the most effective ways of allocating resources to the project and when should they be available? The project plan allows us to investigate the relationship between timescales and resource availability (in general, allocating additional resources to a project shortens its duration) and the efficacy of additional spending on resource procurement.

Detailed costing How much will the project cost and when is that expenditure likely to take place? After producing an activity plan and allocating specific resources, we can obtain more detailed estimates of costs and their timing.

Motivation Providing targets and being seen to monitor achievement against targets is an effective way of motivating staff, particularly where they have been involved in setting those targets in the first place.

Co-ordination When do the staff in different departments need to be available to work on a particular project and when do staff need to be transferred between projects? The project plan, particularly with large projects involving more than a single project team, provides an effective vehicle for communication and co-ordination among teams. In situations where staff may need to be transferred between project teams (or work concurrently on more than one project), a set of integrated project schedules should ensure that such staff are available when required and do not suffer periods of enforced idleness.

Activity planning and scheduling techniques place an emphasis on completing the project in a minimum time at an acceptable cost or, alternatively, meeting an arbitrarily set target date at minimum cost. These are not, in themselves, concerned with meeting quality targets, which generally impose constraints on the scheduling process.

One effective way of shortening project durations is to carry out activities in parallel. Clearly we cannot undertake all the activities at the same time - some require the completion of others before they can start and there are likely to be resource constraints limiting how much may be done simultaneously. Activity scheduling will, however, give us an indication of the cost of these constraints in terms of lengthening timescales and provide us with an indication of how timescales may be shortened by relaxing those constraints. It is up to us, if we try relaxing precedence constraints by, for example, allowing a program coding task to commence before the design has been completed, to ensure that we are clear about the potential effects on product quality.

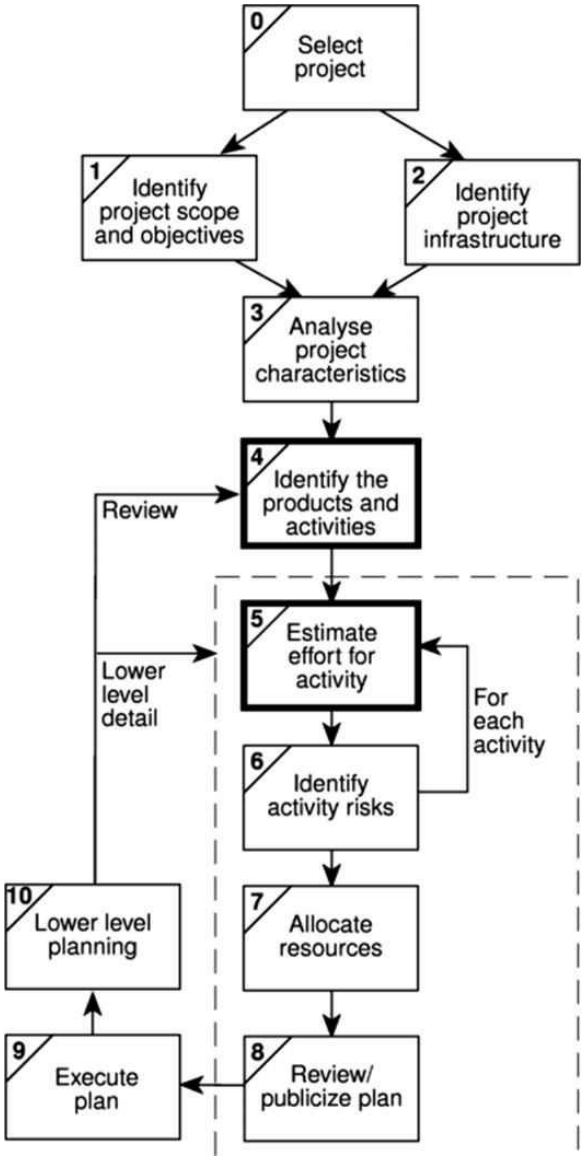
When to plan

- ✓ Planning is an ongoing process of refinement, each iteration becoming more detailed and more accurate than the last. Over successive iterations, the emphasis and purpose of planning will shift.
- ✓ During the feasibility study and project start-up, the main purpose of planning will be to estimate timescales and the risks of not achieving target completion dates or keeping within budget. As the project proceeds beyond the feasibility study, the emphasis will be placed upon the production of activity plans for ensuring resource availability and cash flow control.
- ✓ Throughout the project, until the final deliverable has reached the customer, monitoring and replanning must continue to correct any drift that might prevent meeting time or cost targets.

Project schedules

- ✓ Before work commences on a project or, possibly, a stage of a larger project, the project plan must be developed to the level of showing dates when each activity should start and finish and when and

how much of each resource will be required. Once the plan has been refined to this level of detail we call it a project schedule. Creating a project schedule comprises four main stages.



✓ The first step in producing the plan is to decide what activities need to be carried out and in what order they are to be done. From this we can construct an ideal activity plan - that is, a plan of when each activity would ideally be undertaken were resources not a constraint. It is the creation of the ideal activity plan that we shall discuss in this chapter. This activity plan is generated by Steps 4 and 5 of Step Wise. Activity planning is carried out in Steps 4 and 5.

✓ The ideal activity plan will then be the subject of an activity risk analysis, aimed at identifying potential problems. This might suggest alterations to the ideal activity plan and will almost certainly have implications for resource allocation.

✓ The third step is resource allocation. The expected availability of resources might place constraints on when certain activities can be carried out, and our ideal plan might need to be adapted to take account of this.

✓ The final step is schedule production. Once resources have been allocated to each activity, we will be in a position to draw up and publish a project schedule, which indicates planned start and completion dates and a resource requirements statement for each activity.

Projects and activities

Defining activities

✓ Before we try to identify the activities that make up a project it is worth reviewing what we mean by a project and its activities and adding some assumptions that will be relevant when we start to produce an activity plan.

- a project is composed of a number of inter-related activities;
- a project may start when at least one of its activities is ready to start;
- a project will be completed when all of the activities it encompasses have been completed;

• an activity must have a clearly defined start and a clearly defined end-point, normally marked by the production of a tangible deliverable;

• if an activity requires a resource (as most do) then that resource requirement must be forecastable and is assumed to be required at a constant level throughout the duration of the activity;

• the duration of an activity must be forecastable - assuming normal circumstances, and the reasonable availability of resources;

• some activities might require that others are completed before they can begin (these are known as precedence requirements).

Identifying activities

- ✓ Essentially there are three approaches to identifying the activities or tasks that make up a project - we shall call them the activity-based approach, the product-based approach and the hybrid approach.

The activity-based approach The activity-based approach consists of creating a list of all the activities that the project is thought to involve. This might involve a brainstorming session involving the whole project team or it might stem from an analysis of similar past projects. When listing activities, particularly for a large project, it might be helpful to subdivide the project into the main life style stages and consider each of these separately.

- ✓ Rather than doing this in an ad hoc manner, with the obvious risks of omitting or double-counting tasks, a much favoured way of generating a task list is to create a Work Breakdown Structure (WBS). This involves identifying the main (or high-level) tasks required to complete a project and then breaking each of these down into a set of lower-level tasks. Figure 6.2 shows a fragment of a WBS where the design task has been broken down into three tasks and one of these has been further decomposed into two tasks.
- ✓ Activities are added to a branch in the structure if they directly contribute to the task immediately above - if they do not contribute to the parent task, then they should not be added to that branch. The tasks at each level in any branch should
- ✓ Activities must be defined so that they meet these criteria. Any activity that does not meet these criteria must be redefined.
- ✓ A complete task catalogue will normally include task definitions along with task input and output products and other task-related information.
- ✓ include everything that is required to complete the task at the higher level - if they are not a comprehensive definition of the parent task, then something is missing.
- ✓ When preparing a WBS, consideration must be given to the final level of detail or depth of the structure. Too great a depth will result in a large number of small tasks that will be difficult to

manage, whereas a too shallow structure will provide insufficient detail for project control. Each branch should, however, be broken down at least to a level where each leaf may be assigned to an individual or responsible section within the organization.

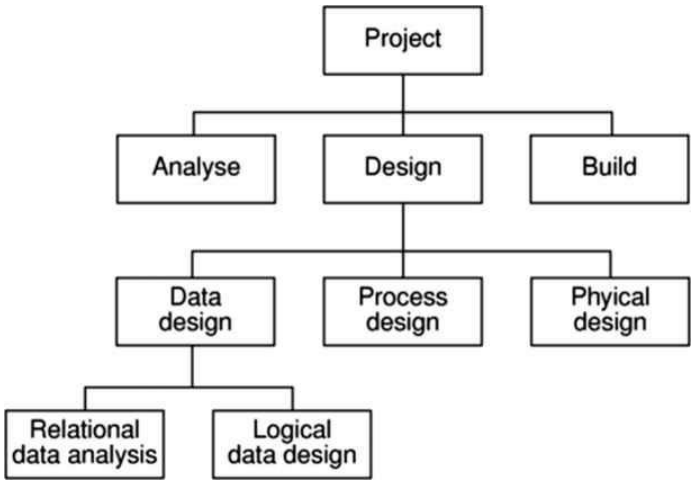


Figure 6.2 A fragment of an activity-based Work Breakdown Structure.

- ✓ Advantages claimed for the WBS approach include the belief that it is much more likely to result in a task catalogue that is complete and is composed of non-overlapping activities. Note that it is only the leaves of the structure that comprise the list of activities comprising the project - higher-level nodes merely represent collections of activities.
- ✓ The WBS also represents a structure that may be refined as the project proceeds. In the early part of a project we might use a relatively high-level or shallow WBS, which can be developed as information becomes available, typically during the project's analysis and specification phases.
- ✓ Once the project's activities have been identified (whether or not by using a WBS) they need to be sequenced in the sense of deciding which activities need to be completed before others can start.

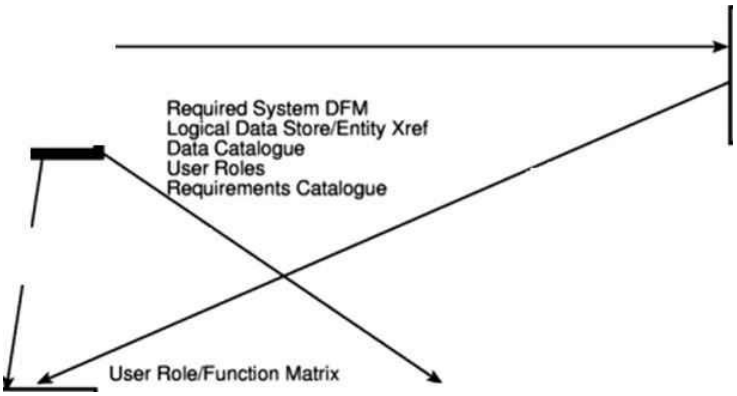
The product-based approach It consists of producing a Product Breakdown Structure and a Product Flow Diagram. The PFD indicates, for each product, which other products are required as inputs. The PFD can therefore be easily transformed into an ordered list of activities by identifying the transformations that turn some products into others. Proponents of this approach claim that it is less likely that a product will be left out of a PBS than that an activity might be omitted from an unstructured activity list.

- ✓ This approach is particularly appropriate if using a methodology such as SSADM, which clearly specifies, for each step or task, each of the products required and the activities required to produce it. The SSADM Reference Manual provides a set of generic PBSs for each stage in SSADM (such as that shown in Figure 6.3), which can be used as a basis for generating a project-specific PBS.
- ✓ The SSADM Reference Manual also supplies generic activity networks and, using the project-specific PBS and derived PFD, these may be used as a basis for developing a project-specific activity network. Figure 6.4 illustrates an activity network for the activities required to create the products in Figure 6.3.

Figure 6.3 *SSADM Product Breakdown Structure for Requirements Specification (adapted from Goodland and Slater).*

- ✓ Notice how the development of a PFD leads directly to an activity network that indicates the sequencing of activities - in Figure 6.4, activity 340 (Enhance required data model) requires products from activity 330 and activity 360 needs products from both activities 330 and 340.

The hybrid approach The WBS illustrated in Figure 6.2 is based entirely on a structuring of activities. Alternatively, and perhaps more commonly, a WBS may be based upon the project's products as illustrated in Figure 6.5, which is in turn based on a simple list of final deliverables and, for each deliverable, a set of activities required to produce that product. Figure 6.5 illustrates a flat WBS and it is likely that, in a project of any size, it would be beneficial to introduce additional levels - structuring both products and activities. The degree to which the structuring is product-based or activity-based might be influenced by the nature of the project and the particular development method adopted. As with a purely activity-based WBS, having identified the activities we are then left with the task of sequencing them.



- ✓ A framework dictating the number of levels and the nature of each level in the structure may be imposed on a WBS. For example, in their MITP methodology, IBM recommend that the following five levels should be used in a WBS:
  - Level 1: Project
  - Level 2: Deliverables such as software, manuals and training courses.
  - Level 3: Components which are the key work items needed to produce deliverables, such as the modules and tests required to produce the system software.

- Level 4: Work-packages which are major work items, or collections of related tasks, required to produce a component.
- Level 5: Tasks which are tasks that will normally be the responsibility of a single person.

Sequencing and scheduling activities

- ✓ Throughout a project, we will require a schedule that clearly indicates when each of the project's activities is planned to occur and what resources it will need. We shall be considering scheduling in more detail in Chapter 8, but let us consider in outline how we might present a schedule for a small project. One way of presenting such a plan is to use a bar chart as shown in Figure 6.6.
- ✓ The chart shown has been drawn up taking account of the nature of the development process (that is, certain tasks must be completed before others may start) and the resources that are available (for example, activity C follows activity B because Andy cannot work on both tasks at the same time). In drawing up the chart, we have therefore done two things - we have sequenced the tasks (that is, identified the dependencies among activities dictated by the development process) and scheduled them (that is, specified when they should take place). The scheduling has had to take account of the availability of staff and the ways in which the activities have been allocated to them. The schedule might look quite different were there a different number of staff or were we to allocate the activities differently.
- ✓ The bar chart does not show why certain decisions have been made. It is not clear, for example, why activity H is not scheduled to start until week 9. It could be that it cannot start until activity F has been completed or it might be because Charlie is going to be on holiday during week 8.

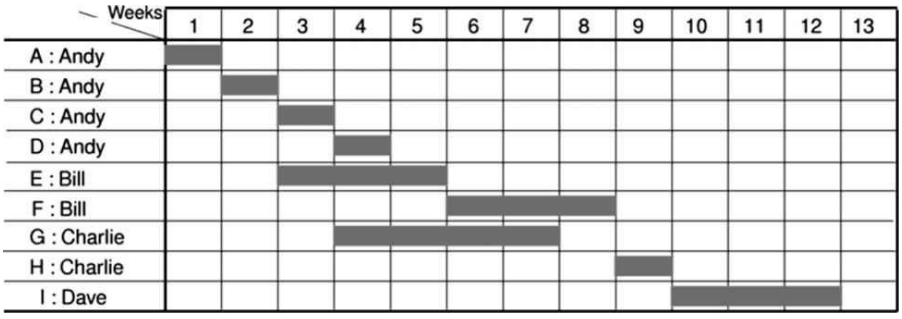


Figure 6.6 A project plan as a bar chart.

- ✓ Activity key:

A: Overall design
B: Specify module 1
C: Specify module 2
D: Specify module 3
E: Code module 1
F: Code module 3
G: Code module 2
H: Integration testing
I: System testing

- ✓ In the case of small projects, this combined sequencing-scheduling approach might be quite suitable, particularly where we wish to allocate individuals to particular tasks at an early planning stage. However, on larger projects it is better to separate out these two activities: to sequence the task according to their logical relationships and then to schedule them taking into account resources and other factors.
- ✓ Approaches to scheduling that achieve this separation between the logical and the physical use networks to model the project and it is these approaches that we will consider in subsequent sections of this chapter.

Network Planning Models

- ✓ These project scheduling techniques model the project's activities and their relationships as a network. In the network, time flows from left to right. These techniques were originally developed in the 1950s - the two best known being CPM (Critical Path Method) and PERT (Program Evaluation Review Technique). More recently a variation on these techniques, called precedence networks, has become popular and it is this method that is adopted in the majority of computer applications currently available. All three methods are very similar and it must be admitted that many people use the same name (particularly CPM) indiscriminately to refer to any or all of the methods.

Formulating a network model

- ✓ The first stage in creating a network model is to represent the activities and their interrelationships as a graph. In CPM we do this by representing activities as links (arrowed lines) in the graph - the nixies (circles) representing the events of activities starting and finishing.

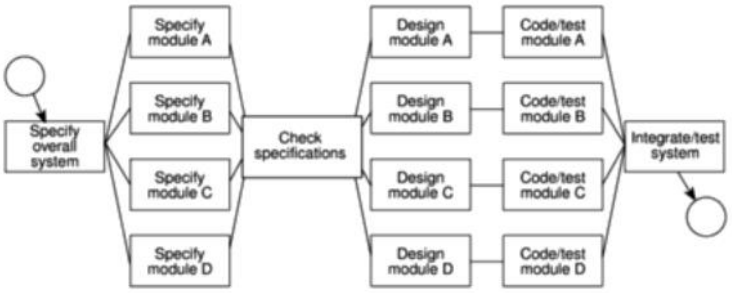


Figure 6.7 *The IOE maintenance group accounts project activity network fragment with a checkpoint activity added.*

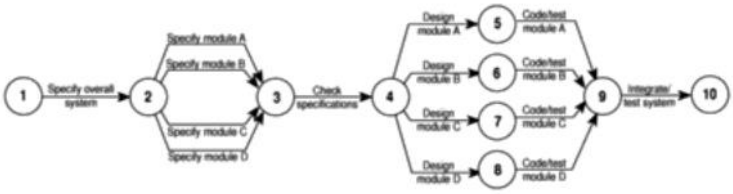


Figure 6.8 *The IOE maintenance group accounts project activity network fragment represented as a CPM network.*

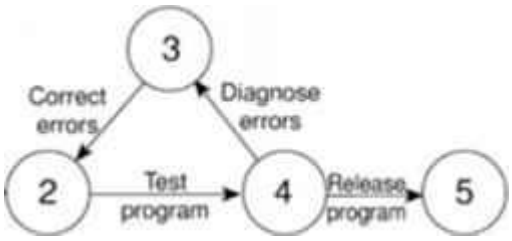
Constructing CPM networks

- ✓ Before we look at how CPM networks are used, it is worth spending a few moments considering the rules for their construction.
  - **A project network may have only one start node** The start node designates the point at which the project may start. All activities coming from that node may start immediately resources are available - that is. they do not have to wait for any other activities to be completed.
  - **A project network may have only one end node** The end node designates the completion of the project and a project may only finish once! The end node for the project fragment shown in Figure 6.8 is the one numbered 10.
  - **A link has duration** A link represents an activity and, in general, activities take time to execute. Notice, however, that the network in Figure 6.8 does not contain any reference to durations. The links are not drawn in any way to represent the activity durations. The network drawing merely represents the logic of the project - the rules governing the order in which activities are to be carried out.
  - **Nodes have no duration** Nodes are events and. as such, are instantaneous points in time. The source node is the event of the project becoming ready to start and the sink node is the event of the project becoming completed. Intermediate nodes represent two simultaneous events - the event of all activities leading in to a node having been completed and the event of all activities leading out of that node being in a position to be started. In Figure 6.9 node 3 is the event that both coding and data take-on have been completed and activity program testing is free to start. Installation may be started only when event 4 has been achieved, that is, as soon as program testing has been completed.

Figure 6.9 *Fragment of a CPM network.*

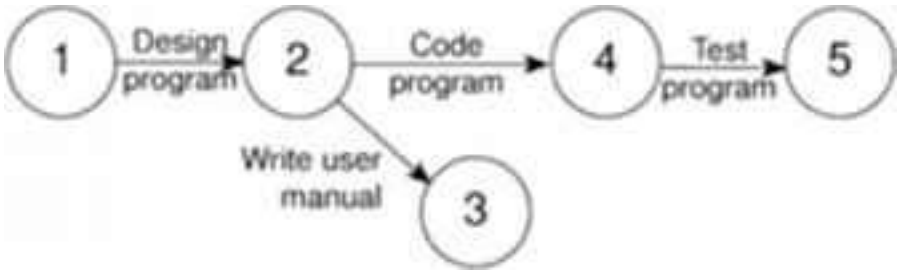
  - **Time moves from left to right** If at all possible, networks are drawn so that time moves from left to right. It is rare that this convention needs to be flouted but. in any case, the arrow s on the activity lines give a strong visual indication of the lime flow of the project.
  - **Nodes are numbered sequentially** There are no precise rules about node numbering but nodes should be numbered so that head nodes (those at the "arrow' end of an activity) always have a higher number than tail events (those at the "non-arrow' end of an activity. This convention makes it easy to spot loops.
  - **A network may not contain loops** Figure 6.10 demonstrates a loop in a CPM network. A loop is an error in that it represents a situation that cannot occur in practice. While loops, in

the sense of iteration, may occur in practice, they cannot be directly represented in a project network. Note that the logic of Figure 6.10 suggests that program testing cannot start until the errors have been corrected.



If we know the number of times we expect to repeat a set of activities, a test-diagnose-correct sequence, for example, then we can draw that set of activities as a straight sequence, repeating it the appropriate number of times. If we do not know how many times a sequence is going to be repeated then we cannot calculate the duration of the project unless we adopt an alternative strategy such as redefining the complete sequence as a single activity and estimating how long it will take to complete it.

- **A network may not contain dangles** A dangling activity such as Write user manual in Figure 6.11 cannot exist, as it would suggest there are two completion points for the project. If. in Figure 6.11 node 5 represents the true project completion point and there are no activities dependent on activity Write user manual, then the network should be redrawn so that activity Write user manual starts at node 2 and terminates at node 5 - in practice, we would need to insert a dummy activity between nodes 3 and 5 as described in Section 6.9. In other words, all events, except the first and the last, must have at least one activity entering them and at least one activity leaving them and all activities must start and end with an event.
- **Precedents are the immediate preceding activities** In Figure 6.9. the activity Program test cannot start until both Code and Data take-on have been completed and activity Install cannot start until Program test has finished. Code and Data take-on can therefore be said to be precedents of Program test, and Program test is a precedent of Install. Note that we do not speak of Code and Data take-on as precedents of Install - that relationship is implicit in the previous statement.



UNIT - IV - RISK MANAGEMENT

RISK MANAGEMENT

Introduction

- ✓ In project evaluation, including assessment of the risk of the project's **not delivering the expected benefits**. we are concerned with the risk of the development project's not proceeding according to plan. We are primarily concerned with the **risks of the project's running late or over budget** and with the identification of the steps that can be taken to **avoid or minimize those risks.**
- ✓ Some risks are more important than others. Whether or not a particular risk is important depends on the nature of the risk, its likely effects on a particular activity and the criticality of the activity.
- ✓ High risk activities on a project's critical path are a cause for concern. To reduce these dangers, we must **ensure that risks are minimized or, at least**, distributed over the project and, ideally, **removed from critical path activities.**
- ✓ The risk of an activity's running over time is likely to depend, at least in part, on **who is doing or managing it.** The evaluation of risk and the allocation of staff and other resources are therefore closely connected.

THE NATURE OF RISK

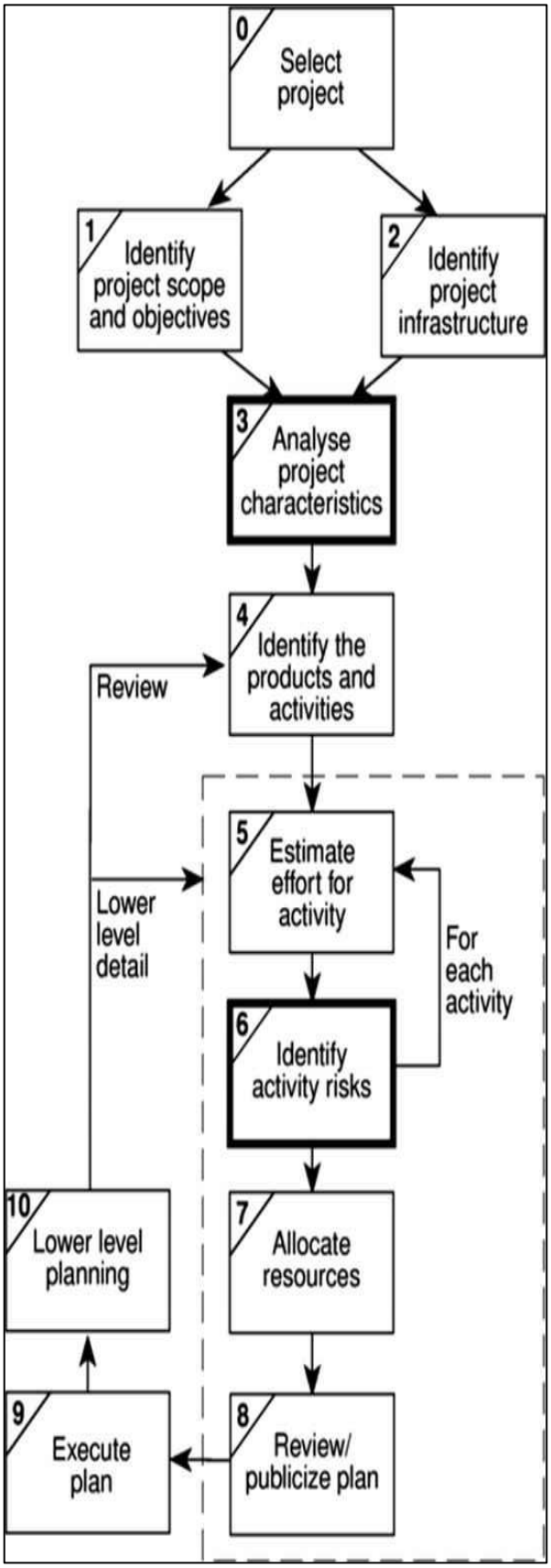
- ✓ For the purpose of identifying and managing those risks that may cause a project to overrun its time-scale or budget, it is convenient to identify three types of risk:

  - those caused by the inherent difficulties of **estimation;**
  - those due to assumptions made during the **planning process;**
  - those of unforeseen (or at least **unplanned) events** occurring.

Estimation errors

- ✓ Some tasks are harder to estimate than others because of the **lack of experience of similar tasks** or because of the nature of a task.
- ✓ Producing a set of user manuals is reasonably straightforward and, given that we have carried out similar tasks previously, we should be able to estimate with some degree of accuracy **how long it will take** and **how much it will cost.**
- ✓ On the other hand, **the time required** for program testing and debugging, might be **difficult to predict with a similar degree of accuracy** - even if we have written similar programs in the past.

✓ Estimation can be improved by **analyzing historic data** for similar activities and similar systems. Keeping records **comparing our original estimates** with the final methods of estimation, outcome will reveal the type of tasks that are **difficult to estimate correctly.**



Planning assumptions:

✓ At every stage during planning, **assumptions are made which, if not valid, may put the plan at risk.**

✓ Our activity network, for example, is likely to be built on the assumption of using a particular design methodology - which may be subsequently changed.

✓ We generally assume that, following coding, a module will be tested and then integrated with others - we might not plan for module testing showing up the need for changes in the original design but, in the event, it might happen.

✓ At each stage in the planning process, it is important to list **explicitly all of the assumptions** that have been made and **identify what effects they might have on the plan** if they are inappropriate.

Eventualities:

✓ Some eventualities might never be foreseen and we can only resign ourselves to the fact that **unimaginable things do,** sometimes, happen. They are, however, very rare.

✓ The majority of unexpected events can, in fact, be identified - the **requirements specification might be altered** after some of the modules have been coded, the **senior programmer might take maternity leave**, the required **hardware might not be delivered** on time. Such events do happen from time to time and, although the likelihood of any one of them happening during a particular project may be relatively low, they must be considered and planned for.

MANAGING RISK

- ✓ The objective of risk management is to avoid or minimize the adverse effects of unforeseen events by avoiding the risks or drawing up contingency plans for dealing with them.
- ✓ There are a number of models for risk management, but most are similar, in that they identify two main components - **risk identification and risk management**. An example of an often-used model is that in Figure 7.2, which shows a task breakdown structure for what Barry Boehm calls risk engineering.
  - **Risk identification** consists of listing all of the risks that can adversely affect the successful execution of the project.
  - **Risk estimation** consists of assessing the likelihood and impact of each hazard.

Figure 7.2 *Boehm's risk engineering task breakdown.*

- **Risk evaluation** consists of ranking the risks and determining risk aversion strategies.
- **Risk planning** consists of drawing up contingency plans and, where appropriate, adding these to the project's task structure. With **small projects,** risk planning is likely to be the responsibility of the **project manager** but **medium or large projects** will benefit from the appointment of a **full-time risk manager.**
- **Risk control** concerns the main functions of the risk manager in **minimizing and reacting to problems throughout the project**. This function will include aspects of quality control in addition to dealing with problems as they occur.
- **Risk monitoring** must be an **ongoing activity,** as the importance and likelihood of particular risks can change as the project proceeds.
- **Risk directing and risk staffing** are concerned with the day-to-day management of risk. Risk aversion and problem-solving strategies frequently involve the use of additional staff and this must be planned for and directed.

- ✓ Whatever task model or whichever techniques are used, ***risk management will not be effective unless all project staff are risk-oriented*** and are provided with an environment where they can freely discuss the risks that might affect a project. All too often, team members who identify potential risks at an early stage are **seen as having a negative attitude.**
- ✓ Writing about attitudes to risk, Dwayne Phillips remarks that ***'I have seen a room get suddenly quiet when someone brings up a "concern"'*** but says that ***'pretending that problems will not occur will not prevent them'.***
- ✓ For **effective risk management**, it is important that the project team are **encouraged to identify and discuss risks as early as possible** in the project's life.

RISK IDENTIFICATION

- ✓ The first stage in any risk assessment exercise is **to identify the hazards** that might affect the duration or resource costs of the project.
- ✓ A hazard is **an event** that might occur and will, if it does occur, create a problem for the successful completion of the project.
- ✓ In identifying and analyzing risks, we can usefully distinguish between the cause (or hazard), its immediate effect (the problem that it creates) and the risk that it will pose to the project.
- ✓ For example, the **Illness of a team member is** a **hazard** that might result in the **problem of late delivery of a component.** The late delivery of that component is likely having an effect on other activities and might, particularly if it is on the **critical path,** put the project **completion date at risk.**
- ✓ A common way of identifying hazards is to use a checklist listing all the possible hazards and factors that influence them. Typical checklists list many, even hundreds, of factors and there are, today, a number of knowledge-based software products available to assist in this analysis.
- ✓ Some hazards are **generic risks** - that is, they are **relevant to all software projects** and **standard checklists can be used** and augmented from an analysis of past projects to identify them. These will include risks such as **misunderstanding the requirements or key personnel** being ill.
- ✓ There will also be **specific risks** that are relevant to an individual project and these are likely to be more difficult to identify without an involvement of the members of the project team and a working environment that encourages risk assessment.
- ✓ The categories of factors that will need to be considered include the following.

  - **Application factors:** The nature of the application - whether it is a simple data processing application, a safety-critical system or a large distributed system with real-time elements - is likely to be a **critical factor**. The **expected size** of the application is also important - the larger

the system, the greater is the likelihood of errors and communication and management problems.

- **Staff factors:** The experience and skills of the staff involved are clearly major factors - an experienced programmer is, one would hope, less likely to make errors than one with little experience. We must, however, also consider the appropriateness of the experience - experience in coding small data processing, modules in Cobol may be of little value if we are developing a complex real-time control system using C++. Such factors as the level of staff satisfaction and the staff turn-over rates are also important to the success of any project - **demotivated staff or key personnel** leaving unexpectedly have caused many a project to fail.
- **Project factors:** It is important that the **project and its objectives are well defined** and that they are **absolutely clear to all members** of the project team and all key stakeholders. Any possibility that this is not the case will pose a risk to the success of the project. Similarly, an agreed and formal quality plan must be in place and adhered to by all participants and any possibility that the quality plan is inadequate or not adhered to will jeopardize the project.
- **Project methods:** Using **well specified and structured methods** for project management and system development will **decrease the risk** of delivering a system that is unsatisfactory or late. Using such methods for the first time, though, may cause problems and delays - it is only with experience that the benefits accrue.
- **Hardware/software factors:** A project that requires new hardware for development is likely to pose a higher risk than one where the software can be developed on existing (and familiar) hardware. Where a system is developed on one type of hardware or software **platform to be used on another** there might be additional (and high) **risks at installation**.
- **Changeover factors:** The need for an 'all-in-one' changeover to the new system poses particular risks. Incremental or gradual changeover minimizes the risks involved but is not always practical. Parallel running can provide a safety net but might be impossible or too costly.
- **Supplier factors:** The extent to which a project relies on external organizations that cannot be directly controlled often influences the project's success. Delays in, for example, the installation of telephone lines or delivery of equipment may be difficult to avoid - particularly if the project is of little consequence to the external supplier.
- **Environment factors:** Changes in the environment can affect a project's success. A significant change in the taxation regulations could, for example, have serious consequences for the development of a payroll application.
- **Health and safety factors:** While not generally a major issue for software projects (compared, say, to civil engineering projects), the possible effects of project activities on the **health and**

safety of the participants and the environment should be considered. BS 6079 states that 'every project should include an audit of these specific risks before work starts' and that 'audit updates should be scheduled as part of the overall project plan'.

Although some factors might influence the project as a whole, it is necessary to consider them individually for each activity - a key member of staff being ill during fact-finding might, for example, be far less serious than a similar absence during user training. Within a PRINCE 2 environment it can be appropriate to list the factors for each of the products identified in the product breakdown structure.

RISK ANALYSIS

- ✓ Having identified the risks that might affect our project we need some way of assessing their importance. Some risks will be relatively **unimportant** (for example, the risk that some of the documentation is delivered a day late), whereas some will be of **major significance** (such as the risk that the software is delivered late). Some are **quite likely to occur** (it is quite likely, for example, that one of the software developers in a team will take a few days sick leave during a lengthy project), whereas others are **relatively unlikely** (hardware failure causing loss of completed code, perhaps).
- ✓ The probability of a hazard's occurring is known as the **risk likelihood;** the effect that the resulting problem will have on the project, if it occurs, is known as the risk impact and the importance of the risk is known as the risk value or risk exposure. The risk value is calculated as:

risk exposure = risk likelihood x risk impact

- ✓ Ideally the risk impact is estimated in monetary terms and the likelihood assessed as a probability. In that case the risk exposure will represent an expected cost in the same sense that we calculated expected costs and benefits when discussing cost-benefit analysis.
- ✓ The risk exposures for various risks can then be compared with each other to assess the relative importance of each risk and they can be directly compared with the costs and likelihoods of success of various contingency plans.
- ✓ However, estimation of these costs and probabilities is likely to be difficult, subjective, time-consuming and costly. In spite of this, it is valuable to obtain some quantitative measure of risk likelihood and impact because, without these, it is difficult to compare or rank risks in a meaningful way. Moreover, the effort put into obtaining a good quantitative estimate can provide a deeper and valuable understanding of the problem.

- ✓ Many risk managers use a **simple scoring method** to provide a quantitative measure for assessing each risk. Some just categorize likelihoods and impacts as **high, medium or low,** but this form of ranking does not allow the calculation of a risk exposure.
- ✓ A better and popular approach is to score the likelihood and impact on a scale of, say, 1 to 10 where the hazard that is most likely to occur receives a score of 10 and the least likely a score of 1.
- ✓ Ranking likelihoods and impacts on a scale of 1 to 10 is relatively easy, but most risk managers will attempt to assign scores in a more meaningful way such that, for example, a likelihood scoring 8 is considered twice as likely as one with a score of 4.
- ✓ Impact measures, scored on a similar scale, must take into account the total risk to the project. This must include the following potential costs:

    - the cost of delays to scheduled dates for deliverables;
    - cost overruns caused by using additional or more expensive resources;
    - the costs incurred or implicit in any compromise to the system's quality or functionality.

Table 7.1 illustrates part of Amanda's risk value assessment. Notice that the hazard with the highest risk value might not be the one that is most likely nor the one with the greatest potential impact.

risk	Hazard	Likelihood	Impact	Risk exposure
R1	Changes to requirements specification during coding	1	8	8
R2	Specification takes longer than expected	3	7	21
R3	Staff sickness affecting critical path activities	5	7	35
R4	Staff sickness affecting non-critical activities	10	3	30
R5	Module coding takes longer than expected	4	5	20
R6	Module testing demonstrates errors or deficiencies in design	1	10	10

Prioritizing the risks

- ✓ Managing risk involves the use of two strategies:

    - reducing the risk exposure by reducing the likelihood or impact;
    - drawing up contingency plans to deal with the risk should it occur.

- ✓ Any attempt to reduce a risk exposure or put a contingency plan in place will have a cost associated with it. It is therefore important to ensure that this effort is applied in the most effective

way and we need a way of prioritizing the risks so that the more important ones can receive the greatest attention.

- ✓ Risk exposures based on scoring methods must be treated with some caution. Amanda's assessment shown in Table 7.1 does not indicate, for example, that risk R5 is twice as important as R6. Nor can it be taken as necessarily meaning that R2 is more important than R5.
- ✓ In the first case, this is because we cannot interpret the risk exposure values quantitatively because they are based on a non-cardinal scoring method. In the second case, the exposure values are far too close for us to be able to distinguish between them - particularly in view of the somewhat approximate and subjective way in which Amanda is likely to have assessed the likelihoods and, perhaps to a lesser extent, the impacts.
- ✓ The risk exposures will, however, allow us to obtain an approximate ranking in order of importance. Considering just the risks in Table 7.1, R3 and R4 are, on this basis, clearly the most important and we could classify them as being high risk concerns.
- ✓ There is a significant difference between the exposure scores of these two and one with the next highest exposure, R2. R2 and R5 have similar scores and might be thought of as medium priority risks. The two remaining risks, R1 and R6 have quite low exposure values and can therefore be classified as low risk items.
- ✓ In practice, there are generally other factors, in addition to the risk exposure value, that must also be taken into account when prioritizing risks.

  - **Confidence of the risk assessment** Some of our risk exposure assessments will be relatively poor. Where this is the case, there is a need for further investigation before action can be planned.
  - **Compound risks** Some risks will be dependent on others. Where this is the case, they should be treated together as a single risk.
  - **The number of risks** There is a limit to the number of risks that can be effectively considered and acted on by a project manager. We might therefore wish to limit the size of the prioritized list.
  - **Cost of action** Some risks, once recognized, can be reduced or avoided immediately with very little cost or effort and it is sensible to take action on these regardless of their risk value. For other risks we need to compare the costs of taking action with the benefits of reducing the risk. One method for doing this is to calculate the risk reduction leverage (RRL) using the equation

Classifying risks into these three categories is clearly not always as easy as in this example although, in practice, risks do frequently cluster and break points are often quite distinct.

$$RRL = \frac{RE_{before} - RE_{after}}{risk\ reduction\ cost}$$

The RRL is used as a factor in prioritizing risks and for evaluating alternative courses of action in dealing with a particular risk.

where RE_{before}, fon, is the original risk exposure value, RE_{after} is the expected risk exposure value after taking action and the risk reduction cost is the cost of implementing the risk reduction action. Risk reduction costs must be expressed in the same units as risk values - that is, expected monetary values or score values.

REDUCING THE RISKS

Broadly, there are five strategies for risk reduction.

- ✓ **Hazard prevention** Some hazards can be prevented from occurring or their likelihood reduced to insignificant levels. The risk of key staff being unavailable for meetings can be minimized by early scheduling, for example.
- ✓ **Likelihood reduction** Some risks, while they cannot be prevented, can have their likelihoods reduced by prior planning. The risk of **late changes to a requirements specification** can, for example, be reduced by prototyping. Prototyping will not eliminate the risk of late changes and will need to be supplemented by contingency planning.
- ✓ **Risk avoidance** A project can, for example, be protected from the risk of overrunning the schedule by increasing duration estimates or reducing functionality.
- ✓ **Risk transfer** the impact of some risks can be transferred away from the project by, for example, contracting out or taking out insurance.
- ✓ **Contingency planning** Some risks are not preventable and contingency plans will need to be drawn up to reduce the impact should the hazard occur. A project manager should draw up contingency plans for using agency programmers to minimize the impact of any unplanned absence of programming staff.

The use of checklists for hazard identification.

Many of these generic checklists, as well as listing common generic hazards, list typical actions for risk reduction. The checklist in Table 7.2 is based upon an often-quoted list produce by Barry Boehm.

Risk	Risk reduction techniques
Personnel shortfalls	staffing with top talent; job matching; team building; training and career development; early scheduling of key personnel.
Unrealistic time and cost estimates	multiple estimation techniques; design to cost; incremental development; recording and analysis of past projects; standardization of methods.
Developing the wrong software functions	improved project evaluation; formal specification methods; user surveys; prototyping; early users' manuals.
Developing the wrong user interface	prototyping; task analysis; user involvement.
Gold plating	requirements scrubbing; prototyping; cost-benefit analysis; design to cost.
Late changes to requirements	stringent change control procedures; high change threshold; incremental prototyping; incremental development (defer changes).
Shortfalls in external supplied components	benchmarking; inspections; formal specifications; contractual agreements; quality assurance procedures and certification.
Shortfalls in externally performed tasks	quality assurance procedures; competitive design or prototyping; teambuilding; contract incentives.
Real-time performance shortfalls	simulation; benchmarking; prototyping; tuning; technical analysis.
Development technically too difficult	technical analysis; cost-benefit analysis; prototyping; staff training and development.

RESOURCE ALLOCATION

✓ In general, the allocation of resources to activities will lead us to **review and modify the ideal activity plan.** It may cause us to revise stage or project completion dates. In any event, it is likely to lead to a narrowing of the time-spans within which activities may be scheduled. The final result of resource allocation will normally be a number of schedules including:

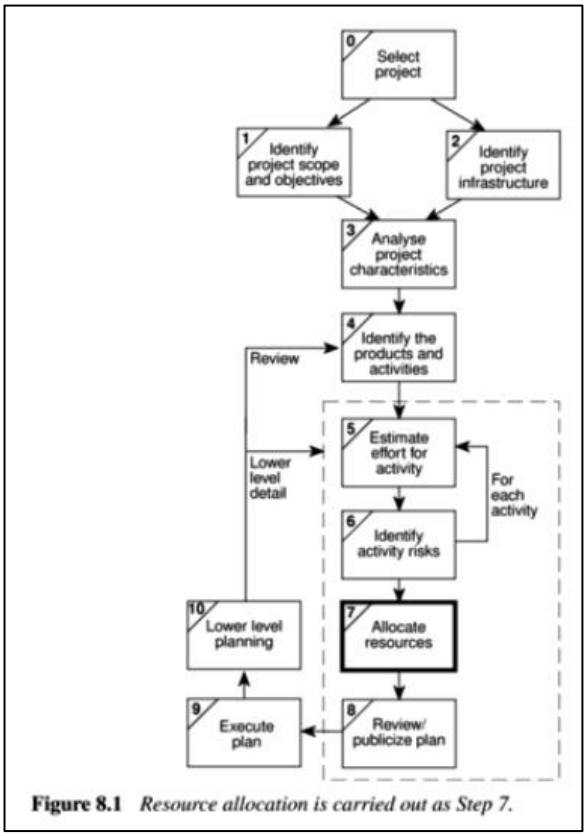


Figure 8.1 *Resource allocation is carried out as Step 7.*

- **activity schedule** indicating the planned start and completion dates for each activity.
- **resource schedule** showing the dates on which each resource will be required and the level of that requirement.
- **cost schedule** showing the planned cumulative expenditure incurred by the use of resources over time.

The nature of resources

✓ A resource is **any item or person** required for the execution of the project. This covers many things-from

paper clips to key personnel- and it is unlikely that we would wish to itemize every resource required, let alone draw up a schedule for their use!

- ✓ Stationery and other standard office supplies, for example, need not normally be the concern of the project manager - ensuring there is always an adequate supply is the role of the office manager.
- ✓ The project manager must concentrate on those resources where there is a possibility that, without planning. they might not be sufficiently available when required. Individual programmers, for example, might be committed to working on a number of projects and it will be important to book their time well in advance. In general, resources will fall into one of seven categories.
  - **Labor** The main items in this category will be members of the development project team such as the project manager, systems analysts and software developers. Equally important will be the quality assurance team and other support staff and any employees of the client organization who might be required to undertake or participate in specific activities.
  - **Equipment** Obvious items will include workstations and other computing and office equipment. We must not forget that staff also need basic equipment such as desks and chairs.
  - **Materials** Materials are items that are consumed, rather than **equipment that is used**. They are of little consequence in most software projects but can be important for some software that is to be widely distributed might, for example, require supplies of floppy disks to be specially obtained.
  - **Space** For projects that are undertaken with existing staff, **space is normally readily available**. If any additional staff (recruited or contracted) should be needed then office space will need to be found.
  - **Services** Some projects will require procurement of specialist services development of a wide area distributed system, for example, requires scheduling of telecommunications services.
  - **Time** Time is the resource that is being offset against the other primary evaluation resources - project time-scales can sometimes be reduced by increasing other resources and will almost certainly be extended if they are unexpectedly reduced.
  - **Money** Money is **a secondary resource**-it is used to buy other resources and will be consumed as other resources are used. It is similar to other resources in that it is available at a cost- in this case interest charges.

Identifying resource requirements

- ✓ The first step in producing a resource allocation plan is to **list the resources that will be required** along with the expected level of demand. This will normally be done by considering each activity in turn and identifying the resources required.

- ✓ It is likely, however, that there will also be resources required that are not activity specific but are part of the project's infrastructure (such as the project manager) or required to support other resources (office space, for example, might be required to house contract software developers).
- ✓ At this stage, it is necessary that the resource requirements list be as comprehensive as possible - it is better that something is included that may later be deleted as unnecessary than to omit something essential.

SCHEDULING RESOURCES

Table 8.1 *Part of Amanda's resource requirements list*

Stage	Activity	Resource	Days	Quantity	Notes
ALL		Project manager	104 F/T		
1	All	Workstation	34		Check software availability
	IoE/P/1	Senior analyst	34 F/T		
2	All	Workstation	—	3	1 per person would be ideal
	IoE/P/2	Analyst–designer	20 F/T		
	IoE/P/3	Analyst–designer	15 F/T		
	IoE/P/4	Analyst–designer	25 F/T		
	IoE/P/5	Analyst–designer	15 F/T		Could use analyst–programmer
3	All	Workstation	2 F/T		
	IoE/P/6	Senior analyst*	2 F/T		
4	All	Workstation	—	3	As stage 2
	IoE/P/7	Analyst–designer	7 F/T		
	IoE/P/8	Analyst–designer	6 F/T		
	IoE/P/9	Analyst–designer	4 F/T		
	IoE/P/10	Analyst–designer	4 F/T		
5	All	Workstation	—	4	1 per programmer
	All	Office space	—		If contract programmers used
	IoE/P/11	Programmer	30 F/T		
	IoE/P/12	Programmer	28 F/T		
	IoE/P/13	Programmer	15 F/T		
	IoE/P/14	Programmer	25 F/T		
6	All	Full machine access	—		Approx. 16 hours for full system test
	IoE/P/15	Analyst–designer	6 F/T		

* In reality, this would normally be done by a review involving all the analysts working on stage 2.

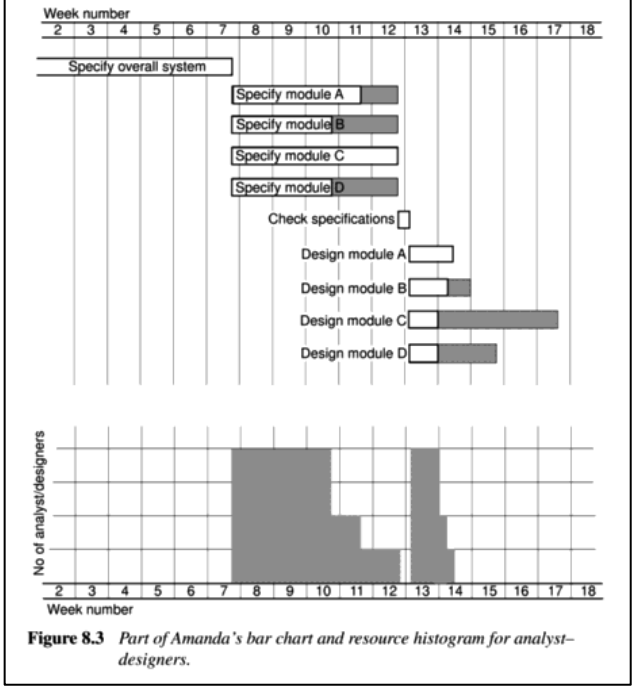


Figure 8.3 *Part of Amanda's bar chart and resource histogram for analyst–designers.*

✓ Having produced the resource requirements list, the next stage is to map this onto the activity plan to assess the distribution of resources required over the duration of the project.

✓ This is best done by representing the activity plan as a bar chart and using this to produce a resource histogram for each resource. Figure 8.3 illustrates Amanda's activity plan as a bar chart and a resource histogram for analyst-designers.

✓ Each activity has been scheduled to start at its **earliest start date** - a sensible initial strategy, since we would, other things being equal, wish to save any float to allow for contingencies. Earliest start date scheduling, as is the case with Amanda's project, frequently creates resource histograms that start with a peak and then tail off.

✓ Changing the level of resources on a project over time, particularly personnel, generally adds to the cost of a project. Recruiting staff has costs and even where staff are transferred internally, time will be needed for familiarization with the new project environment.

✓ The resource histogram in Figure 8.3 poses particular problems in that it calls for two

analyst-designers to be idle for eleven days, one for six days and one for two days between the specification and design stage.

- ✓ It is unlikely that lOE would have another project requiring their skills for exactly those periods of time and this raises the question as to whether this idle time should be charged to Amanda's project. The ideal resource histogram will he smooth with. perhaps an initial buildup and a staged run-down.

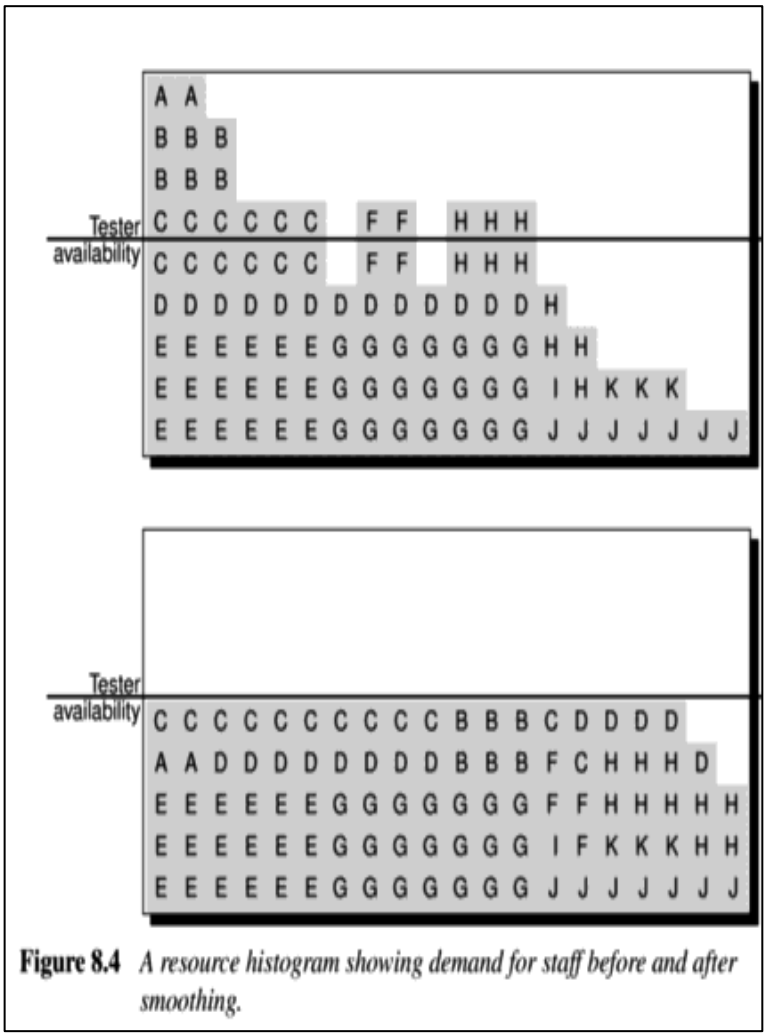


Figure 8.4 *A resource histogram showing demand for staff before and after smoothing.*

- ✓ An additional problem with an uneven resource histogram is that it is more likely to call for levels of resource beyond those available.
- ✓ Figure 8.4 illustrates how, by adjusting the start date of some activities and splitting others, a resource histogram can. subject to constraints such as precedence requirements, be smoothed to contain resource demand at available levels.
- ✓ The different letters represent staff working on a series of module testing tasks, that is. one person working on task A, two on tasks B and C etc.
- ✓ In Figure 8.4. the original histogram was created by scheduling the activities at their earliest start dates. The resource histogram shows the typical peaked shape caused by earliest start date scheduling and calls for a total of nine staff where only five are available for the project.
- ✓ By delaying the start of some of the activities, it has been possible to smooth the histogram and reduce the maximum level of demand for the resource. Notice that some activities, such as C and I), have been split. Where non-critical activities can be split they can provide a useful way of filling troughs in the demand for a resource, but in software projects it is difficult to split tasks without increasing the time they take.
- ✓ Some of the activities call for more than one unit of the resource at a time -activity F. for example, requires two programmers, each working for two weeks. It might be possible to reschedule this activity to use one programmer over four weeks although that has not been considered in this case.
- ✓ In practice, resources have to be allocated to a project on an activity-by-activity basis and finding the 'best' allocation can be time consuming and difficult. As soon as a member of the project team

is allocated to an activity that activity acquires a scheduled start and finish date and the team member becomes unavailable for other activities for that period. Thus. allocating a resource to one activity limits the flexibility for resource allocation and scheduling of other activities.

- ✓ It is therefore helpful to prioritize activities so that resources can be allocated to competing activities in some rational order. The priority must always be to allocate resources to critical path activities and then to those activities that are most likely to affect others. In that way. lower priority activities are made to lit around the more critical, already scheduled activities. Of the various ways of prioritizing activities, two are described below.

  - **Total float priority** Activities are ordered according to their total float, those with the smallest total float having the highest priority. In the simplest application of this method, activities are allocated resources in ascending order of total float. However, as scheduling proceeds, activities will be delayed (if resources are not available at their earliest start dates) and total floats will be reduced. It is therefore desirable to recalculate floats (and hence reorder the list) each time an activity is delayed.
  - **Ordered list priority** with this method, activities that can proceed at the same time are ordered according to a set of simple criteria. An example of this is Barman's priority list, which takes into account activity duration as well as total float:

    1. shortest critical activity;
    2. critical activities;
    3. shortest non-critical activity;
    4. non-critical activity with least float;
    5. non-critical activities.

Unfortunately, resource smoothing, or even containment of resource demand to available levels, is not always possible within planned time-scales - deferring activities to smooth out resource peaks often puts back project completion. Where that is the case, we need to consider ways of increasing the available resource levels or altering working methods.

CRITICAL PATH METHOD

- ✓ The critical path method calculates the theoretical early start and finish dates, and late start and finish dates, for all activities without regard for any resource limitations, by performing a forward pass analysis and a backward pass analysis through the project schedule network paths.

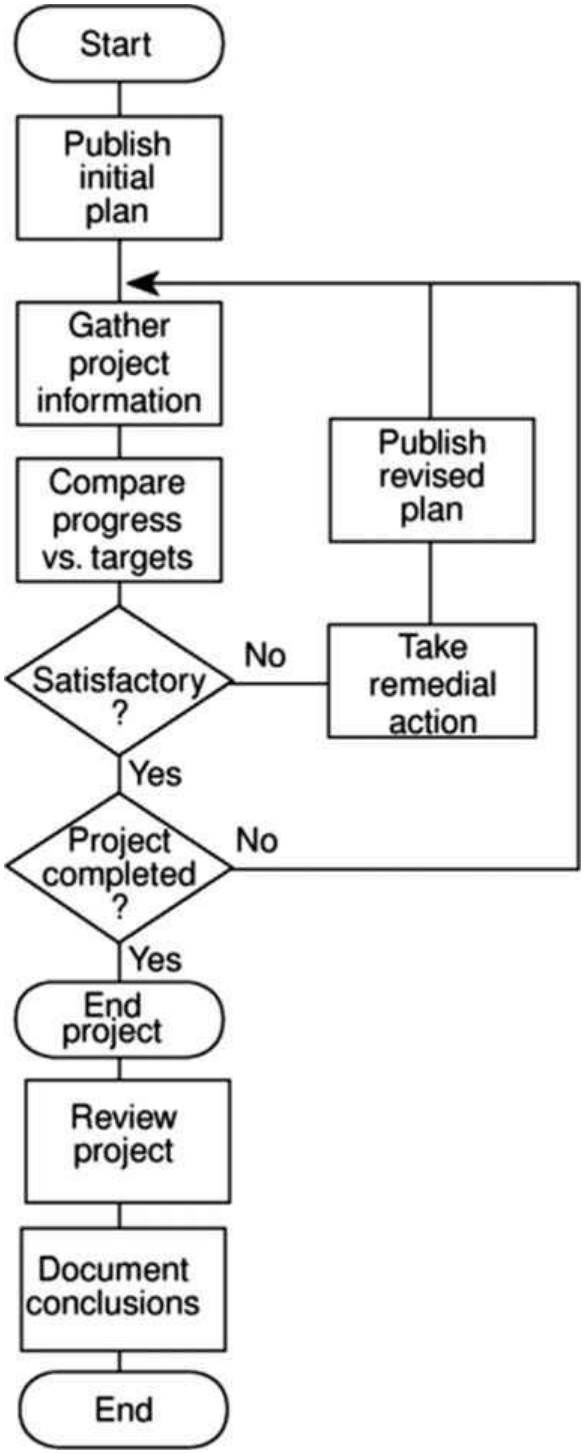
- ✓ The resulting early and late start and finish dates are not necessarily the project schedule; rather, they indicate the time periods within which the schedule activity should be scheduled, given activity durations, logical relationships, leads, lags, and other known constraints.
- ✓ Calculated early start and finish dates, and late start and finish dates, may or may not be the same on any network path since total float, which provides schedule flexibility, may be positive, negative, or zero. On any network path, the schedule flexibility is measured by the positive difference between early and late dates, and is termed "total float." Critical paths have either a zero or negative total float, and schedule activities on a critical path are called "critical activities.
- ✓ " A critical path is normally characterized by zero total/free float across the critical path and network paths can have multiple near critical paths. Adjustments to activity durations, logical relationships, leads and lags, or other schedule constraints may be necessary to produce network paths with a zero or positive total float.
- ✓ Once the total float for a network path is zero or positive, then the free float—the amount of time that a schedule activity can be delayed without delaying the early start date of any immediate successor activity within the network path—can also be determined.

COST SCHEDULES

- ✓ It is now time to produce a detailed cost schedule showing weekly or monthly costs over the life of the project. This will provide a more detailed and accurate estimate of costs and will serve as a plan against which project progress can be monitored. Calculating cost is straightforward where the organization has standard cost figures for staff and other resources. Where this is not the case, then the project manager will have to calculate the costs. In general, costs are categorized as follows.
    - **Staff costs** These will include staff salaries as well as the other direct costs of employment such as the employer's contribution to social security funds, pension scheme contributions, holiday pay and sickness benefit. These are commonly charged to projects at hourly rates based on weekly work records completed by staff. Note that contract staff are usually charged by the week or month - even when they are idle.
    - **Overheads** Overheads represent expenditure that an organization incurs, which cannot be directly related to individual projects or jobs including space rental, interest charges and the costs of service departments (such as personnel). Overhead costs can be recovered by making a fixed charge on development departments (in which case they usually appear as a weekly or monthly charge for a project), or by an additional percentage charge on direct staff employment costs. These additional charges or on costs can easily equal or exceed the direct employment costs.

- **Usage charges** in some organizations, projects are charged directly for use of resources such as computer time (rather than their cost being recovered as an overhead). This will normally be on an 'as used' basis.

CREATING THE FRAMEWORK



✓ Exercising control over a project and ensuring that targets are met is a matter of regular monitoring, finding out what is happening, and comparing it with current targets. If there is a mismatch between the planned outcomes and the actual ones then either replanning is needed to bring the project back on target or the target will have to be revised.

✓ Figure 9.1 illustrates a model of the project control cycle and shows how, once the initial project plan has been published, project control is a continual process of monitoring progress against that plan and, where necessary, revising the plan to take account of deviations.

✓ It also illustrates the important steps that must be taken after completion of the project so that the experienced gained in any one project can feed into the planning stages of future projects, thus allowing us to learn from past mistakes.

✓ In practice we are normally concerned with departures from the plan in four dimensions - delays in meeting target dates, shortfalls in quality, inadequate functionality, and costs going over target. In this chapter we are mainly concerned with the first and last of these.

Responsibility

✓ The overall responsibility for ensuring satisfactory progress on a project is often the role of the project steering committee or Project Board. Day-to-day responsibility will rest with the project manager and, in all but the smallest of projects, aspects of this can be delegated to team leaders.

- ✓ Figure 9.2 illustrates the typical reporting structure found with medium and large projects. With small projects (employing around half a dozen or fewer staff) individual team members usually report directly to the project manager, but most cases team leaders will collate reports on their section's progress and forward summaries to the project manager. These, in turn, will be incorporated into project-level reports for the steering committee and, via them or directly, progress reports for the client.
- ✓ Reporting may be oral or written, formal or informal, or regular or ad hoc and some examples of each type are given in Table 9.1. While any effective team leader or project manager will be in touch with team members and available to discuss problems, any such informal reporting of project progress must be complemented by formal reporting procedures.

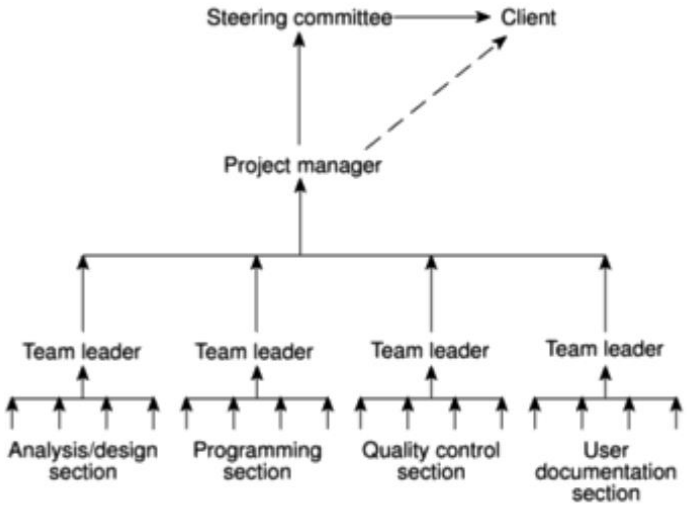


Figure 9.2 *Project reporting structures.*

Assessing progress

- ✓ Progress assessment will normally be made on the basis of information collected and collated at regular intervals or when specific events occur.
- ✓ Wherever possible, this information will be objective and tangible - whether or not a particular report has been delivered, for example. However, such end-of-activity deliverables might not occur sufficiently frequently throughout the life of the project. Here progress assessment will have to rely on the judgement of the team members who are carrying out the project activities.

Table 9.1 *Categories of reporting*

Report type	*Examples*	*Comment*
Oral formal regular	weekly or monthly progress meetings	while reports may be oral formal written minutes should be kept
Oral formal ad hoc	end-of-stage review meetings	while largely oral, likely to receive and generate written reports
Written formal regular	job sheets, progress reports	normally weekly using forms
Written formal ad hoc	exception reports, change reports	
Oral informal ad hoc	canteen discussion, social interaction	often provides early warning; must be backed up by formal reporting

Setting checkpoints

- ✓ It is essential to set a series of checkpoints in the initial activity plan. Checkpoints may be:
  - regular (monthly, for example);
  - tied to specific events such as the production of a report or other deliverable.

Taking snap-shots

- ✓ The frequency with which a manager needs to receive information about progress will depend upon the size and degree of risk of the project or that part of the project under their control. Team leaders, for example, need to assess progress daily (particularly when employing inexperienced staff) whereas project managers may find weekly or monthly reporting appropriate. In general, the higher the level, the less frequent and less detailed the reporting needs to be.
- ✓ There are, however, strong arguments in favour of formal weekly collection of information from staff carrying out activities. Collecting data at the end of each week ensures that information is provided while memories are still relatively fresh and provides a mechanism for individuals to review and reflect upon their progress during the past few days.
- ✓ Major, or project-level, progress reviews will generally take place at particular points during the life of a project - commonly known as review points or control points. PRINCE 2, for example, designates a series of checkpoints where the status of work in a project or for a team is reviewed. At the end of each project Stage, PRINCE 2 provides for an End Stage Assessment where an assessment of the project and consideration of its future are undertaken.

COST MONITORING

- ✓ Expenditure monitoring is an important component of project control. Not only in itself, but also because it provides an indication of the effort that has gone into (or at least been charged to) a project. A project might be on time but only because more money has been spent on activities than originally budgeted. A cumulative expenditure chart such as that shown in Figure 9.9 provides a simple method of comparing actual and planned expenditure. By itself it is not particularly meaningful - Figure 9.9 could, for example, illustrate a project that is running late or one that is on time hut has shown substantial costs savings! We need to take account of the current status of the project activities before attempting to interpret the meaning of recorded expenditure. Project costs may be monitored by a company's accounting system. By themselves, they provide little information about project status.

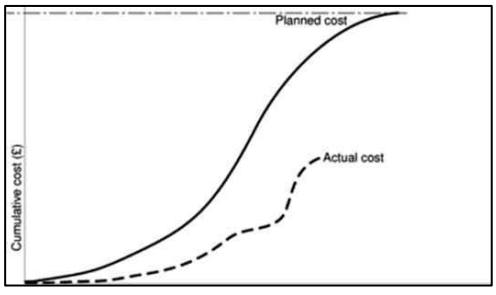


Figure 9.9 Tracking cumulative expenditure.

- ✓ Cost charts become much more useful if we add projected future costs calculated by adding the estimated costs of uncompleted work to the costs already incurred. Where a computer-based planning tool is used, revision of cost schedules is generally provided automatically once actual expenditure has been recorded.

Prioritizing monitoring

- ✓ So far we have assumed that all aspects of a project will receive equal treatment in terms of the degree of monitoring applied. We must not forget, however, that monitoring takes time and uses resources that might sometimes be put to better use! In this section we list the priorities we might apply in deciding levels of monitoring.
    - **Critical path activities** Any delay in an activity on the critical path will cause a delay in the completion date for the project. Critical path activities are therefore likely to have a very high priority for close monitoring.
    - **Activities with no free float** A delay in any activity with no free float will delay at least some subsequent activities even though, if the delay is less than the total float, it might not delay the project completion date. These subsequent delays can have serious effects on our resource schedule as a delay in a subsequent activity could mean that the resources for that activity will become unavailable before that activity is completed because they are committed elsewhere.
    - **Free float** is the amount of time an activity may be delayed without affecting any subsequent activity.
    - **Activities with less than a specified float** If any activity has very little float it might use up this float before the regular activity monitoring brings the problem to the project manager's attention. It is common practice to monitor closely those activities with less than, say, one-week free float.
    - **High risk activities** A set of high-risk activities should have been identified as part of the initial risk profiling exercise. If we are using the PERT three-estimate approach we will designate as high risk those activities that have a high estimated duration variance. These activities will be given close attention because they are most likely to overrun or overspend.
    - **Activities using critical resources** Activities can be critical because they are very expensive (as in the case of specialized contract programmers). Staff or other resources might be available only for a limited period, especially if they are controlled outside the project team. In any event, an activity that demands a critical resource requires a high level of monitoring.

UNIT - V

GLOBALIZATION ISSUES IN PROJECT

What is globalization?

- ✓ Globalization is the process by which ideas, knowledge, information, goods and services **spread around the world**.
- ✓ In business, the term is used in an economic context to **describe integrated economies marked by free trade**, the **free flow of capital among countries** and **easy access to foreign resources,** including labor markets, to maximize returns and benefit for the common good.
- ✓ Globalization, or globalization as it is known in some parts of the world, is driven by the convergence of cultural and economic systems.
- ✓ This convergence promotes -- and in some cases necessitates -- **increased interaction, integration and interdependence among nations.**
- ✓ The more countries and regions of the world become intertwined politically, culturally and economically, the more globalized the world becomes.

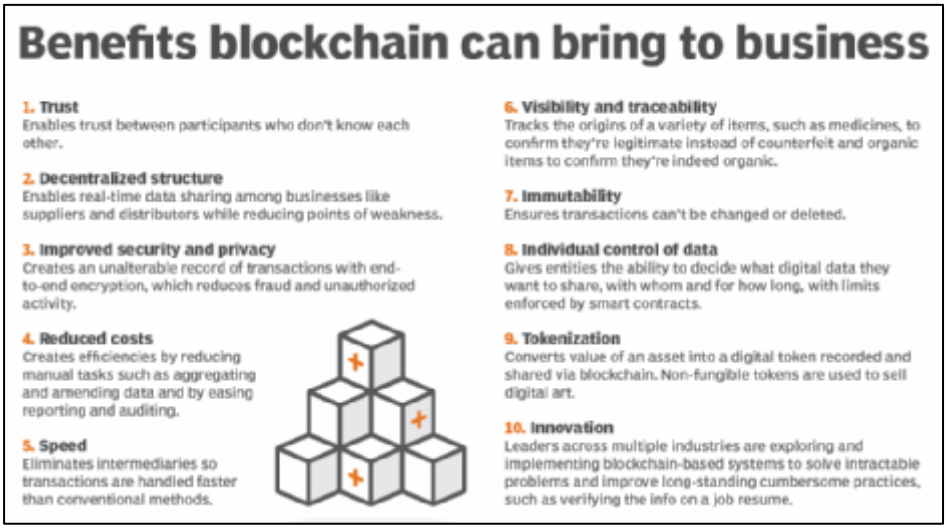
How globalization works

- ✓ In a globalized economy, countries specialize in the products and services they have a competitive advantage in. This generally means what they can produce and provide most efficiently, with the least amount of resources, at a **lower cost** than competing nations.
- ✓ If all countries are specializing in **what they do best**, production should be more efficient **worldwide, prices should be lower**, economic growth widespread and all countries should benefit -- in theory.
- ✓ Policies that promote free trade, **open borders** and international cooperation all drive economic globalization. They enable businesses to **access lower priced raw materials** and parts, take advantage of lower cost labor markets and access larger and growing markets around the world in which to sell their goods and services.
- ✓ ***Money, products, materials, information and people flow*** more swiftly across national boundaries today than ever.
- ✓ Advances in technology have **enabled** and accelerated this flow and the **resulting international interactions and dependencies**. These technological advances have been especially pronounced in transportation and telecommunications.
- ✓ Among the **recent technological changes** that have played a role in globalization are the following:
    - **Internet and internet communication.** The internet has **increased the sharing and flow of information and knowledg**e, access to ideas and exchange of culture among people of

different countries. It has contributed to closing the digital divide between more and less advanced countries.

- **Communication technology.** The introduction of 4G and 5G technologies has dramatically increased the speed and responsiveness of mobile and wireless networks. Increased speed and bandwidth are among the benefits of 5G technology.

- **IoT and AI.** These technologies are enabling the **tracking of assets** in transit and as they move across borders, making cross-border product management more efficient.
- **Blockchain.** This technology is **enabling the development of decentralized databases and storage** that support the tracking of materials in the supply chain. Blockchain facilitates the secure access to data required in industries such as healthcare and banking. For example, blockchain provides a **transparent ledger** that centrally records and vets' transactions in a way that **prevents corruption and breaches.**



- **Transportation.** Advances in air and fast rail technology have facilitated the movement of people and products. And changes in shipping logistics technology moves raw materials, parts and finished products around the globe more efficiently.
- **Manufacturing.** Advances such as **automation and 3D printing** have reduced geographic constraints in the manufacturing industry. 3D printing enables digital designs to be sent

anywhere and physically printed, making distributed, smaller-scale production near the point of consumption easier. Automation speeds up processes and supply chains, giving workforces more flexibility and improving output.

Why is globalization important?

- ✓ Globalization changes the way nations, businesses and people interact. Specifically, it changes the nature of economic activity among nations, expanding trade, opening global supply chains and **providing access to natural resources and labor markets.**
- ✓ Changing the way trade and financial exchange and interaction occurs among nations also **promotes the cultural exchange of ideas.** It removes the barriers set by geographic constraints, political boundaries and political economies.
- ✓ For example, globalization enables businesses in one nation to **access another nation's resources**. More open access changes the way products are developed, supply chains are managed and organizations communicate. Businesses **find cheaper raw materials and parts, less expensive or more skilled labor and more efficient ways to develop products.**
- ✓ With ***fewer restrictions*** on trade, globalization creates opportunities to expand. Increased trade **promotes international competition.** This, in turn, spurs innovation and, in some cases, the exchange of ideas and knowhow. In addition, people coming from other nations to do business and work bring with them their **own cultures,** which ***influence and mix with other cultures.***
- ✓ The many types of exchange that globalization facilitates can have positive and negative effects. For instance, the exchange of people and goods across borders can bring fresh ideas and help business. However, this movement can also heighten the **spread of disease** and promote ideas that might destabilize political economies.

EVOLUTION OF GLOBALIZATION

- Globalization is an historical process that began with the first movement of people out of Africa into other parts of the world. Traveling short or long distances, migrants' merchants and others have delivered their ideas, customs and products to new lands. The melding, borrowing and adaptation of outside influences are found in many areas of human life.

History of globalization

- ✓ Although many people consider globalization a twentieth century phenomenon, the process has been happening for millennia. Examples include the following:
  - **The Roman Empire.** Going back to 600 B.C., the Roman Empire spread its economic and governing systems through significant portions of the ancient world for centuries.
  - **Silk Road trade.** These trade routes, which date from 130 B.C. to 1453 A.D., represented another wave of globalization. They brought merchants, goods and travelers from **China through Central Asia and the Middle East to Europe.**

- **Pre-World War I**. ***European*** countries made significant investments overseas in the decades before World War I. The period from 1870 to 1914 is called the golden age of globalization.
- **Post-World War II.** ***The United States*** led the effort to create a global economic system with a set of broadly accepted international rules. Multinational institutions were established such as the United Nations (UN), International Monetary Fund, World Bank and World Trade Organization to promote international cooperation and free trade.

✓ The term globalization as it's used today came to prominence in the 1980s, reflecting several technological advancements that increased international interactions. IBM's introduction of the personal computer in 1981 and the subsequent evolution of the modern internet are two examples of technology that helped drive international communication, commerce and globalization.

✓ Globalization has ebbed and flowed throughout history, with periods of expansion and retrenchment. The 21st century has witnessed both. Global stock markets plummeted after the Sept. 11, 2001, terrorist attacks in the United States, but rebounded in subsequent years.

✓ More recently, nationalist political movements have slowed immigration, closed borders and increased trade protectionism. The pandemic has had similar effects on borders and immigration and also disrupted supply chains.

✓ However, overall, the early **21st century** has seen a dramatic **increase** in the pace of global integration. **Rapid advances** in technology and telecommunications are responsible for much of this change.

Globalization in the modem era

✓ Globalization, in the modem sense of the term, came into existence after the Second World War. One of the **main factors** for this was the plan by the ***world leaders to break down the borders for fostering trade relations between nations.*** It was also in this that major countries like India, So Lanka, Indonesia and some countries in South America gained independence.

✓ As a result, these countries too started having their own economic systems and made established trade relations with the rest of the world. The establishment of the United Nations Organization (UNO) was also a major step in this regard.

✓ Gradually, the economic scenario of the world strengthened and it led to better trade relations and communication. Some other factors which have put a positive impact on globalization are:

- Promotion of free commerce and trade
- Abolition of various double taxes, tariffs, and capital controls
- Reduction of transport cost and development of infrastructure
- Creation of global corporations
- Blend of culture and tradition across the countries

- ✓ Another milestone in the history of globalization is the creation of the World Trade Organization which led to the growth of a uniform platform to settle trade and commercial disputes.
- ✓ According to economic surveys the world exports ***improved from 8.5% to around 16.2% due to globalization***.

India and globalization

- ✓ The wake of globalization was first felt in the 1990s in India when the then **finance minister Dr Manmohan Singh** initiated the economic liberalization plan.
- ✓ Since then, India has gradually become one of the economic giants in the world. Today, it has one of the fastest growing economies in the world with an **average growth rate of around 67%.** There has also been a rise in the per capital income and the standard of living Poverty has also reduced by around 10%.
- ✓ The service industry has a share of around 54% of the annual Gross Domestic Product while the industrial and agricultural sectors share around 29% and 17% respectively. Due to the process of globalization, the exports have also improved significantly.
- ✓ Globalization has really out a positive impact on today's economy and it is expected to develop in the years to come.

What is the G20?

- ✓ The G20, or Group of Twenty, is an international forum that aims to foster international cooperation by addressing global economic issues, such as financial stability and climate change.
- ✓ The G20 is **made up of 19 countries** and the European Union, including most of the world's largest economies.
- ✓ The nations involved account for 60% of the planet's population, 75% of global trade and 80% of world GDP. It was founded in 1999, following the 1997 financial crisis, and has met every year since then.
- ✓ Since 2008, the G20 has held an annual summit that brings together heads of state to discuss important economic issues. The G20's president is selected annually on a rotating basis, and that person's home country hosts the summit.
- ✓ In 2019, the summit was held in Osaka, Japan, and it addressed issues such as women's empowerment, climate change and artificial intelligence. The 2020 summit was to be in Riyadh, Saudi Arabia, but was held virtually because of the pandemic.
- ✓ Three of the main themes addressed were empowering people, especially women and youth; safeguarding the planet; and long-term strategies to share the benefits of innovation and technological advancement. The 2021 summit will be held in Rome, Italy, and will focus on recovery from the pandemic and climate change.

- ✓ The **members of G20** are Argentina, Australia, Brazil, Canada, China, France, Germany, Japan, India, Indonesia, Italy, Mexico, Russia, South Africa, Saudi Arabia, South Korea, Turkey, the United Kingdom, the United States and the European Union. Spain is a permanent guest of the organization.

Types of globalization: Economic, political, cultural

- ✓ **Economic globalization**. Here, the focus is on the **integration of international financial markets and the coordination of financial exchange**. Free trade agreements, such the North American Free Trade Agreement and the Trans-Pacific Partnership are examples of economic globalization. Multinational corporations, which operate in two or more countries, play a large role in economic globalization.
- ✓ **Political globalization.** This type **covers the national policies that bring countries together politically, economically and culturally**. Organizations such as NATO and the UN are part of the political globalization effort.
- ✓ **Cultural globalization.** This aspect of globalization **focuses in a large part on the technological and societal factors that are causing cultures to converge**. These include increased ease of communication, the pervasiveness of social media and access to faster and better transportation.

These three types influence one another. For example, liberalized national trade policies drive economic globalization. Political policies also affect cultural globalization, enabling people to communicate and move around the globe more freely. Economic globalization also affects cultural globalization through the import of goods and services that expose people to other cultures.

Effects of globalization

- ✓ The effects of globalization can be felt locally and globally, touching the lives of individuals as well as the broader society in the following ways:
  - **Individuals.** Here, a variety of international influences affect ordinary people. Globalization affects their access to goods, the **prices they pay and their ability to travel** to or even move to other countries.
  - **Communities.** This level encompasses the impact of globalization on local or regional organizations, businesses and economies. It affects who lives in communities, where they work, who they work for, their ability to move out of their community and into one in another country, among other things. Globalization also changes the way **local cultures develop within communities.**
  - **Institutions.** Multinational corporations, national governments and other organizations such as colleges and universities are **all affected by their country's approach** to and acceptance of globalization. Globalization affects the ability of companies to grow and expand, a university's

ability to diversify and grow its student body and a government's ability to pursue specific economic policies.

While the effects of globalization can be observed, analyzing the net impact is more complex. Proponents often see specific results as **positive** and critics of globalization view the same results as **negative**. A relationship that benefits one entity may damage another, and whether globalization benefits the world at large remains a point of contention.

Examples of globalization

Multinational corporations are a tangible example of globalization. Some examples include the following:

- **McDonald's** had 39,198 fast-food restaurants in 119 countries and territories, according to its Securities and Exchange Commission filing at the end of 2020. It employed more than 2.2 million people at that time, the filing said.
- **Ford Motor Company** reported in 2021 that it works with about 1,200 tier 1 suppliers around the globe.
- **Amazon's** recent expansion has it using tens of thousands of suppliers and employing more than nearly 1.3 million full- and part-time employees.
- Through their influence on social and economic development in the countries that host them, multinational corporations embody the contradictions of globalization. They bring jobs, skills and wealth to the region they are investing or doing business in. But they also can **destroy local businesses**, exploit cheap labor and **threaten indigenous cultures**. The benefits they offer are often unsustainable because the loyalty of multinationals is to their investors and bottom lines and not to the local people, economies and cultures where they are doing business.
- Another example of globalization is the **response to the COVID-19 pandemic**. Because the world was able to communicate across boundaries, nations were able to work together to quickly produce vaccines for the virus.
- In addition, doctors traveled where they were needed. For example, Cuba sent doctors to Italy at the beginning of the pandemic to assist with the crisis as it developed there.
- However, countries also enacted strict travel restrictions and many closed their borders to cut down on the free movement of people and spread of the virus.

Benefits of globalization

✓ Globalization enables countries to access **less expensive natural resources** and **lower cost labor**.

✓ As a result, they can produce lower cost goods that can be sold globally. Proponents of globalization argue that it improves the state of the world in many ways, such as the following:

- **Solves economic problems.** Globalization moves jobs and capital to places that need these resources. It gives rich countries access to lower cost resources and labor and poorer countries access to jobs and the investment funds they need for development.
- **Promotes free trade.** Globalization puts pressure on nations to reduce tariffs, subsidies and other barriers to free trade. This consequently promotes economic growth, creates jobs, makes companies more competitive and lowers prices for consumers.
- **Spurs economic development.** Theoretically, globalization gives poorer countries access to foreign capital and technology they would not otherwise have. Foreign investment can result in an improved standard of living for the citizens of those nations.
- **Encourages positive trends in human rights and the environment.** Advocates of globalization point to improved attention to human rights on a global scale and a shared understanding of the impact of people and production on the environment.
- **Promotes shared cultural understanding.** Advocates view the increased ability to travel and experience new cultures as a positive part of globalization that can contribute to international cooperation and peace.

Negative consequences of globalization

✓ Many proponents view globalization as way to solve systemic economic problems. But critics see it as increasing global inequality. Among the critiques of globalization are the following issues:

- **Destabilizes markets.** Critics of globalization blame the elimination of trade barriers and the freer movement of people for ***undermining national policies and local cultures***. Labor markets in particular are affected when people move across borders in search of higher paying jobs or companies outsource work and jobs to lower cost labor markets.
- **Damages the environment.** The transport of goods and people among nations generates greenhouse gas and all the negative effects it has on the environment. Global travel and trade also can introduce, sometimes inadvertently, invasive species to foreign ecosystems. Industries such as fishing and logging tend to go where business is most lucrative or regulations are less strict, which has resulted in overfishing and deforestation in some parts of the world.
- **Lowers living standards.** When companies move operations overseas to minimize costs, such moves can eliminate jobs and increase unemployment in sectors of the home country.
- **Facilitates global recessions.** Tightly integrated global markets carry a greater risk of global recessions. The 2007-2009 financial crisis and Great Recession is a good example of how intertwined global markets are and how financial problems in one country or region can quickly affect other parts of the world. Globalization reduces the ability of individual nations to effectively use monetary and fiscal policy to control the national economy.

- **Damages cultural identities.** Critics of globalization decry the decimation of unique cultural identities and languages that comes with the international movement of businesses and people. At the same time, the internet and social media are driving this trend even without the movement of people and commerce.
- **Increases the likelihood of pandemics.** Increased travel, critics say, has the potential to increase the risk of pandemics. The H1N1 (swine flu) outbreak of 2009 and coronavirus in 2020 and 2021 are two examples of serious diseases that spread to multiple nations quickly.

Future of globalization

✓ Technological advances, particularly **blockchain, mobile communication and banking, are fueling economic globalization.** Nonetheless, rising levels of protectionism and anti-globalization sentiment in several countries could slow or even reverse the rapid pace of globalization.

✓ Nationalism and increasing trends toward conservative economic policies are driving these anti-globalization efforts.

✓ Global trade is also made more difficult and facing rising threats from other factors, such as these:

- climate change
- decaying infrastructure
- cyber attacks
- human rights abuses
- The takeaway

✓ Globalization is a longstanding trend that is in the process of changing and possibly slowing. There are advantages to the more open border and free trade that globalization promotes, as well as negative consequences.

✓ In a modern, post-pandemic world, individuals, businesses and countries must consider both sides of the globalization issue. Learn **how companies are rethinking global supply chains to avoid disruption and reap the benefits of globalization.**

CHALLENGES IN BUILDING GLOBAL TEAMS

✓ One of the biggest requests and agenda items that arises when coaching chief executives, chief marketing officers and other global leaders today is: "How do I lead my global team effectively?"

✓ As a result of globalization, the traditional boundaries that existed within organizations are becoming increasingly blurred. This is having an impact on how top teams are being organized in effect, how they must be led. This can also create greater challenges.

✓ Many executives say that **leading a team spanning border can be difficult**, if not frustrating.

✓ Getting their teams in shape can be a **slow process** and take a **great deal of patience,** due to the **complications of time-zone differences, loss of face-to-face Contact, language barriers, technology issues** or having **different ways of conducting business** in different regions.

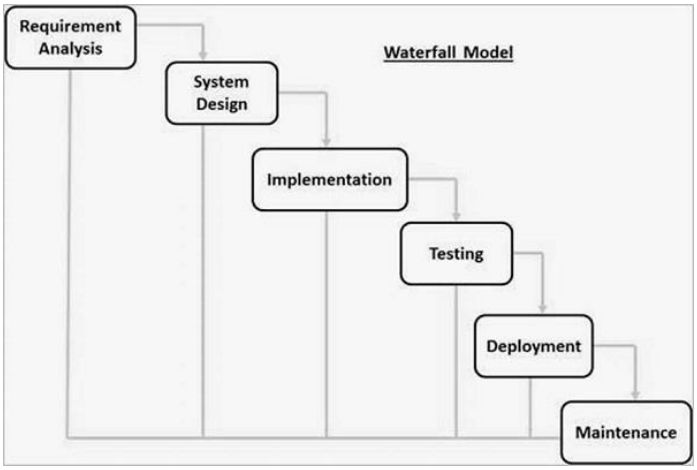
Five challenges can emerge within global teams:

- ✓ "***Cannot trust what I cannot see***", where a lack of face-to-face contact makes it difficult to build relationships.
- ✓ "***Out of sight, out of mind***", where real work takes priority over virtual work.
- ✓ "***Stifled diversity***", when force-fitting universal processes can kill creativity.
- ✓ "***Old-school leadership***", where the command-and-control style may not be appropriate.
- ✓ "***Time-zone resentment***" conducting meetings consistently in the "middle" zone can create resentment from those having to dial in during the early morning or late evening.

MODELS FOR THE EXECUTION OF SOME EFFECTIVE MANAGEMENT TECHNIQUES FOR MANAGING GLOBAL TEAMS

Waterfall Model - Design

- ✓ Waterfall approach was first SDLC Model to be used widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially. The following illustration is a representation of the different phases of the Waterfall Model.
- ✓ The sequential phases in Waterfall model are –
    - **Requirement Gathering and analysis** – All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.
    - **System Design** – The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.
    - **Implementation** – With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.



- **Integration and Testing** – All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
- **Deployment of system** – Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.
- **Maintenance** – There are some issues which come up in the client environment. To fix those issues, patches are released. Also, to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

✓ All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases. The next phase is started only after the defined set of goals are achieved for previous phase and it is signed off, so the name "Waterfall Model". In this model, phases do not overlap.

Spiral model

✓ The spiral model combines the idea of iterative development with the systematic, controlled aspects of the waterfall model.

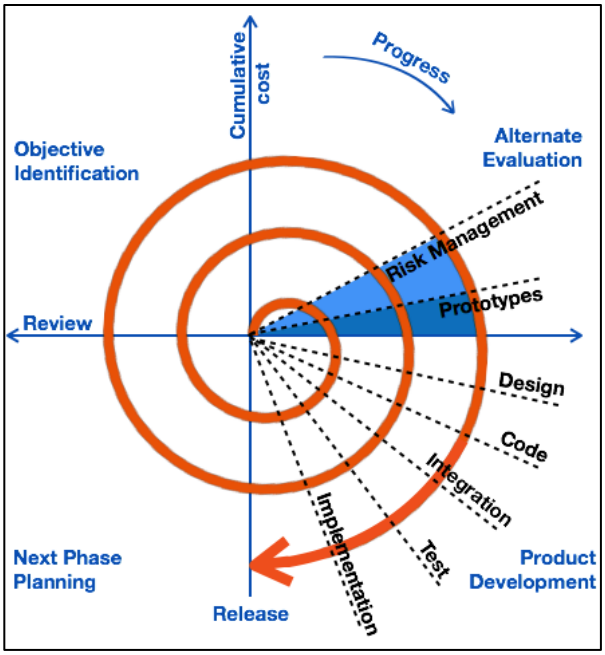
✓ This Spiral model is a **combination of iterative development process model and sequential linear development model** i.e. the waterfall model with a very high emphasis on risk analysis. It allows incremental releases of the product or incremental refinement through each iteration around the spiral.

✓ The spiral model has four phases. A software project repeatedly passes through these phases in iterations called Spirals.

- **Identification:** This phase starts with gathering the business requirements in the baseline spiral. In the subsequent spirals as the product matures, identification of system requirements, subsystem requirements and unit requirements are all done in this phase. This phase also includes understanding the system requirements by continuous communication between the customer and the system analyst. At the end of the spiral, the product is deployed in the identified market.
- **Design:** The Design phase starts with the conceptual design in the baseline spiral and involves architectural design, logical design of modules, physical product design and the final design in the subsequent spirals.
- **Construct or Build:** The Construct phase refers to production of the actual software product at every spiral. In the baseline spiral, when the product is just thought of and the design is being developed a POC (Proof of Concept) is developed in this phase to get customer feedback. Then in the subsequent spirals with higher clarity on requirements and design details a working

model of the software called build is produced with a version number. These builds are sent to the customer for feedback.

- **Evaluation and Risk Analysis:** Risk Analysis includes identifying, estimating and monitoring the technical feasibility and management risks, such as schedule slippage and cost overrun. After testing the build, at the end of first iteration, the customer evaluates the software and provides feedback.



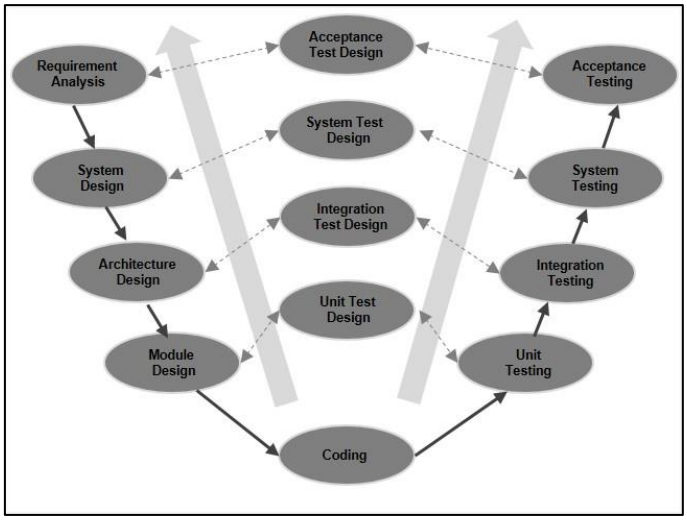
✓ The following illustration is a representation of the Spiral Model, listing the activities in each phase. Based on the customer evaluation, the software development process enters the next iteration and subsequently follows the linear approach to implement the feedback suggested by the customer. The process of iterations along the spiral continues throughout the life of the software.

V-model

✓ The V-model is an SDLC model where execution of processes happens in a sequential manner in a V-shape. It is also known as **Verification and Validation model**.

✓ The V-Model is an extension of the waterfall model and is based on the association of a testing phase for each corresponding development stage. This means that for every single phase in the development cycle, there is a directly associated testing phase.

✓ This is a highly-disciplined model and the next phase starts only after completion of the previous phase.



V-Model - Design

✓ Under the V-Model, the corresponding testing phase of the development phase is planned in parallel. So, there are Verification phases on one side of the 'V' and Validation phases on the other side. The Coding Phase joins the two sides of the V-Model. The following illustration depicts the different phases in a V-Model of the SDLC.

V-Model - Verification Phases

There are several Verification phases in the V-Model, each of these are explained in detail below.

Business Requirement Analysis

This is the first phase in the development cycle where the product **requirements are understood from the customer's perspective.** This phase involves detailed communication with the customer to understand his expectations and exact requirement. This is a very important activity and needs to be managed well, as most of the customers are not sure about what exactly they need. The **acceptance test design planning** is done at this stage as business requirements can be used as an input for acceptance testing.

System Design

Once you have the clear and detailed product requirements, it is time to design the complete system. The system design will have the understanding and detailing the complete hardware and communication setup for the product under development. The system test plan is developed based on the system design. Doing this at an earlier stage leaves more time for the actual test execution later.

Architectural Design

Architectural specifications are understood and designed in this phase. Usually more than one technical approach is proposed and based on the technical and financial feasibility the final decision is taken. The system design is broken down further into modules taking up different functionality. This is also referred to as **High Level Design (HLD)**.

The data transfer and communication between the internal modules and with the outside world (other systems) is clearly understood and defined in this stage. With this information, integration tests can be designed and documented during this stage.

Module Design

In this phase, the detailed internal design for all the system modules is specified, referred to as **Low Level Design (LLD)**. It is important that the design is compatible with the other modules in the system architecture and the other external systems. The unit tests are an essential part of any development process and helps eliminate the maximum faults and errors at a very early stage. These unit tests can be designed at this stage based on the internal module designs.

Coding Phase

The actual coding of the system modules designed in the design phase is taken up in the Coding phase. The best suitable programming language is decided based on the system and architectural requirements.

The coding is performed based on the coding guidelines and standards. The code goes through numerous code reviews and is optimized for best performance before the final build is checked into the repository.

Validation Phases

The different Validation Phases in a V-Model are explained in detail below.

Unit Testing

Unit tests designed in the module design phase are executed on the code during this validation phase. Unit testing is the testing at code level and helps eliminate bugs at an early stage, though all defects cannot be uncovered by unit testing.

Integration Testing

Integration testing is associated with the architectural design phase. Integration tests are performed to test the coexistence and communication of the internal modules within the system.

System Testing

System testing is directly associated with the system design phase. System tests check the entire system functionality and the communication of the system under development with external systems. Most of the software and hardware compatibility issues can be uncovered during this system test execution.

Acceptance Testing

Acceptance testing is associated with the business requirement analysis phase and involves testing the product in user environment. Acceptance tests uncover the compatibility issues with the other systems available in the user environment. It also discovers the non-functional issues such as load and performance defects in the actual user environment.

IMPACT OF THE INTERNET ON PROJECT MANAGEMENT

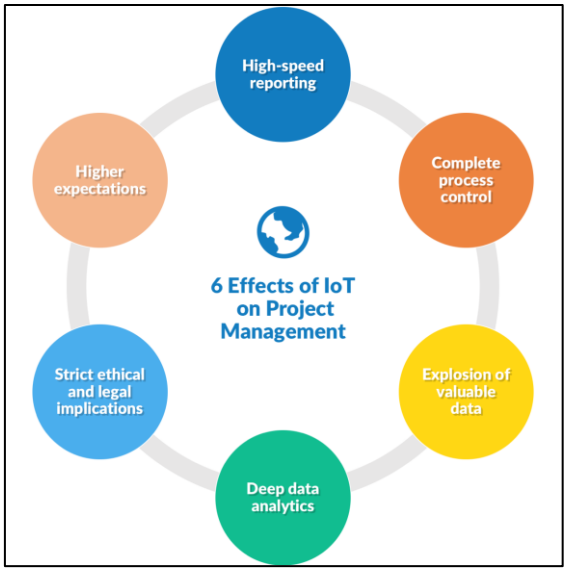
What is Internet of Things (IoT)?

- ✓ The Internet of Things (IoT) can be defined as "***a network of physical objects or people called things***" that are embedded with software electronics, network, and sensors which allows these objects to collect and exchange data.
- ✓ The goal of lot is to extend to internet connectivity from standard devices like computer, mobile, tablet to relatively dumb devices like a toaster.
- ✓ lot makes virtually everything **"smart,"** by improving aspects of our life with the power of data collection, Al algorithm, and networks
- ✓ The thing in lot can also be a person with a diabetes monitor implant, an animal with tracking devices, etc.
- ✓ The IoT is essentially the global network of devices that can communicate with one another and end users through the internet.
- ✓ *"Gartner, Inc. forecasts that 8.4 billion connected things will be in use worldwide in 2017, up 31 percent from 2016, and will reach 20.4 billion by 2020."*
- ✓ That's almost three devices per person, and each of those devices will be measuring data and facilitating communication.

- ✓ Many major technology firms are **developing their own IoT platforms**, such as **Amazon Web Services**, **Microsoft Azure**, and **Google Cloud**.
- ✓ But preparing for the IoT isn't just a concern for mega corporations. Project managers and small business leaders also need to be ready for a connected workplace.
- ✓ The IoT intersects with project management on everything from team collaboration to data collection. You can expect real-time status reporting via IoT to user in a new era of dynamic planning and revolutionized project execution.
- ✓ Data collection will happen seamlessly and constantly, allowing leaders to make more informed decisions. Inventory and resources will be **easily monitored at all times.**
- ✓ **Devices can automatically sense** and **respond to what is happening around them** or in their network, reducing the need for human intervention, lowering operating costs, increasing response times, and minimizing errors. Moreover, customers can expect to receive better and faster service.
- ✓ In terms of **project management technology**, the IoT will fundamentally **alter the speed of project execution**. Organizations that capitalize on the IoT will complete projects faster than those that don't, and organizations that fail to adapt to the IoT revolution will be left hopelessly behind.
- ✓ At least six things will change, which will require project managers to adapt both technically and systematically.

THE EFFECT OF INTERNET ON PROJECT MANAGEMENT

IoT enables hyper speed reporting



- ✓ lot substantially reduces the cost of communication.
- ✓ The hyper connected devices and constant flow of data that automate systems will speed things up considerably.
- ✓ **No more idle times are required** in between activities.
- ✓ No more silos from support systems such as databases, storage, and its operations.

IMPACT Say you're an IT project manager, and you need to run a status report on all of your organization's desktop and laptop computers and tablets and mobile devices. In the past, this might take weeks. But with the lot, a project manager could ***run a report on the quantity and condition of all of those pieces in an instant.***

IoT allows complete monitoring and process control

- ✓ loT allows project managers, management, and stakeholders to monitor and control activities in real time. The overall snapshot of a comprehensive system is monitored on a single screen, which allows overseers to immediately attend to any interruptions.

IMPACT Using equipment as an example, **sensors** will be used for monitoring and predicting maintenance needs throughout a project's lifetime. The scope of devices, activities and conditions that need to be tested will increase exponentially as projects become more complex. Ease of use and environments suddenly become critical.

IOT creates an explosion of valuable project data

- ✓ In the past, archiving historical data was a time and labour-intensive process. With the lot, **historical data will become available immediately,** which is extremely helpful for current and future projects. Everything from budgeting to individual meetings with team members will be recorded in great detail, providing a solid foundation for future decisions.

IMPACT Pro Management tool will need to be more responsive and scalable to accommodate this data explosion. Organizations need to make sure that their project management software package is capable of growing to accommodate this incoming of data. They also need to know what's time to pored-for example, if your team is capping out on your storage allowance each month.

IoT allows super-deep data analytics

- ✓ With the loT comes advanced data analytics, and advanced data analytics require advanced interpretations and management.

IMPACT Project managers must upgrade their skills related to data handling, which could mean increasing spend and resources toward data hiring experienced data analysts, and accounting for data analysis when creating the project timeline. In other words, the more familiar project managers are with the importance advanced data analysis, the better the chances for project success.

lOT users in stricter ethical and legal implications

- ✓ Today's internet-connected devices send data to each other extremely fast. We're not dealing with dial-up modems anymore. One error could create a domino effect that could topple an entire project or, in extreme cases, an entire career before you can say "Enron".

IMPACT Businesses of all sizes need to impose stricter ethical and legal implications on any slight mistake or oversight, Project managers and team members should be aware of this early on so that the project can be completed with **minimal ethical and legal risks.**

IOT raises expectations for all stakeholders

- ✓ Once companies adopt loT, the marketplace will be transformed into a level playing field. Only the strongest and the fittest will survive. No can organizations hide behind old excuses such as, "**We don't have to that data**" or "**We need a few weeks to get that report back**"

IMPACT Project managers need to lead the charge when it comes to raising standards in the late era. As a project manager, your job is to be aware of the **most useful technology available** and **enable your team to use it.**

MANAGING PROJECTS FOR THE INTERNET

- ✓ The entire IOT process starts with the devices themselves like smartphones, smartwatches, electronic appliances like TV, Washing Machine which helps you to communicate with the IOT platform. Here, are four fundamental components of an IoT system:

1)Sensors/Devices: Sensors or devices are a key component that helps you to **collect live data from the surrounding environment.** All this data may have various levels of complexities. It could be a simple temperature monitoring sensor, or it may be in the form of the video feed. A device may have various types of sensors which **performs multiple tasks apart from sensing.** Example. A mobile phone is a device which has multiple sensors like GPS, camera but your smartphone is not able to sense these things.

2)Connectivity: All the **collected data is sent to a cloud infrastructure.** The sensors should be connected to the doUD (Department of Urban Development) using various mediums of communications. These **communication mediums** include mobile or satellite networks, Bluetooth, WI-FI, WAN, etc

3) Data Processing: Once that data is collected, and it gets to the cloud, the software performs **processing on the gathered data.** This process can be just checking the temperature, reading on devices like AC or heaters. However, it can sometimes also be very complex like identifying objects, using computer vision on video.

4)User Interface: The **information needs to be available to the end-user** in some way which can be achieved by triggering alarms on their phones or sending them notification through email or text message. The user sometimes might need an interface which actively checks their IOT system. **For example**, the user has a **camera installed in his home**. He wants to access video recording and all the feeds with the help of a web server. However, it's not always one-way communication.

- ✓ Depending on the lot application and complexity of the system, the user may also be **able to perform an action** which may create cascading effects. **For example,** if a user detects any changes in the temperature of the refrigerator, with the help of IOT technology the user should **able to adjust the temperature** with the help of their mobile phone.

EFFECTS OF PROJECT MANAGEMENT ACTIVITIES

- ✓ With the advent of globalization, project management is no longer a local issue, but an international affair that is risky in nature. Changes in the global environment are presenting organizations with both opportunities and challenges.

- ✓ However, a review of the results of project monitoring and evaluation on World Bank projects indicates that many of the key problems of implementation lie in the general environment of the project, and are not under the direct control of the project manager.
- ✓ The project management (PM) environment for international development projects is also much more complicated than domestic projects in industrialized countries. Project managers should understand the social, economic, political and cultural factors that affect the project environment.
- ✓ International projects are more complicated and riskier than domestic projects. Some risks encountered in international projects are not the same as those in domestic projects.
- ✓ The **cultural differences** issue has been recognized as one of the **main concerns** in international projects management. Although there may also be cultural differences in a domestic project team because of the team members' difference in origin, international project teams seem to be more easily influenced by cultural differences.
- ✓ Kwak (2002) states that the culture issue is the least known but the most hazardous in the context of international development projects.

Some Issues in Managing International Projects

Many researchers and practitioners are aware of the challenge of managing international projects, since international projects **face uncertainties** caused by host country conditions. Researchers have previously identified some key factors that constrain the success of international projects.

Cultural differences

Large-scale international projects are of a global nature. Therefore, a **high degree of coordination and communication is needed**. Communication in the international environment is complicated by different languages, cultures and etiquette.

- ✓ The internationalization in project management creates intercultural communication problems that result in significant **misunderstanding and conflict.** Pheng and Leong (2000) conducted research on international construction in China, and determined that cultural differences are a critical factor that can actually **affect the outcome of an international project.**
- ✓ For an international project manager, ***understanding key concepts in cross-cultural management*** and project management is the basic requirement in the era of globalization.
- ✓ Muriithi and Crawford (2003) also argue that Western management concepts may not be applicable to other cultures that are not so deeply rooted in the Western philosophy. They suggest that appropriate modifications can be made to current management theories by studying cultural differences.

Political factors

Khattab, Anchor and Davies (2007) did a study to examine the vulnerability of international projects to political risks. Their study results showed that political risks are ranked first by respondents. Other

authors also mention that political interventions can sometimes decide the success of foreign-invested firms (Buckley, Clegg & Hui, 2006).

- ✓ Political risks are the key risks to successful international construction contracting. For international projects, these factors can produce problems that may not be problematic in domestic projects. Dikmen, Birgonul and Han (2007) state that political risk factors receive the most attention from researchers in international projects.

Legal factors.

One of the more difficult aspects of doing business globally is **dealing with vast differences in legal and regulatory environments.** The United States, for example, has an established set of laws and regulations that provide direction to businesses operating within its borders.

But because there is no global legal system, key areas of business law, for example, ***contract provisions and copyright protection, can be treated in different ways in different countries.*** Companies doing international business often face many inconsistent laws and regulations (Buckley, Clegg & Hui, 2006).

Economic factors

If you plan to do business in a foreign country, you need to know its level of economic development. You also should be aware of factors **influencing the value of its currency** and the **impact that changes in that value will have on your profits.** If you don't understand a nation's level of economic development, you'll have trouble answering some basic questions, such as; will consumers in this country be able to afford the product I want to sell? How many units can I expect to sell? Will it be possible to make a reasonable profit? Researchers pay a great deal of attention to economic risk factors in international projects.

COMPARISON OF PROJECT MANAGEMENT SOFTWARE

Dot Project

- ✓ DotProject is a web-based, multi-user and multi-language project management application. It is free and open so investitware, and is maintained by an community of volunteer programmers.
- ✓ Dot Project is an Open Source Project Management application supported free of charge by web developers from all over the world.
- ✓ DotProject is in PHP and is free to anyone who would like to download and use it. It is easy to work with dotProject because it has cleaned and simple user interface.
- ✓ dotProject is mostly a **task-oriented project management system**, predating contemporary tools addressing methodologies such as **Agile software development.** Instead, it uses the "waterfall" model to manage tasks, sequentially and/or in parallel, assigned to different members of a team or teams, and establishing dependencies between tasks and milestones. It can display such relationships visually using **Gantt charts.**

- ✓ It is not specifically designed for software project management but can be used by most kinds of project-oriented service companies (such as design studios, architects, media producers, lawyer offices, and the like), all of **which organize their work conceptually in similar ways.**
- ✓ Unlike most contemporary software project management tools, dotProject cannot be easily integrated with the usual constellation of 'business tools'; instead, it is a complete, standalone application, not requiring anything else besides a platform that supports PHP (it is web server agnostic) and MySQL/MariaDB. Except for drawing Gantt graphics, it has a reasonably small footprint in terms of memory and disk space requirements.
- ✓ While dotProject is self-contained in terms of user authentication and management, it can also integrate with an external LDAP server, as well as synchronize its users with a phpNuke installation. Further authentication methods are possible to be developed separately but are currently not part of the core software.
- ✓ The **core of dotProject focuses on Companies,** which may have subunits known as Departments, which, in turn, have Users. Companies can be internal or external; thus, a project can be shared/viewed by customers, by giving them access via a special Role. Roles have a reasonably complex permissions system, allowing a certain degree of fine-tuning of what kind of information can be viewed and/or edited by the users. There is even the possibility of having a 'public' role with no access to any information but nevertheless able to submit tickets via the integrated ticketing system.
- ✓ Projects, in turn, are linked to one company and (optionally) one or more departments in that company; users assigned to a specific project, however, may come from any company or department — thus allowing cross-company development, or the involvement of external users.
- ✓ Projects are **divided into Tasks,** which can have all sorts of dependencies between them; tasks can also have subtasks, and they can be assigned to specific milestones. This allows the establishment of complex relationships between the team members, the many projects they might be involved in, and the amount of work to be distributed among all.
- ✓ As is common with other project management tools, tasks can be created as mere stubs and completed later; assigned and reassigned to team members; or even moved across projects (or becoming subtasks of other tasks).
- ✓ Team members are expected to register the amount of time they spend on each task, which is **accomplished via Logs.** These are often one-line comments with an estimate of the time consumed (but can optionally have much more information); dotProject will take those logs into account when calculating the workload, the overall cost of the project so far (and compare it to the budget), as well as figuring out what tasks are being completed in due time or are overdue.

- ✓ Depending on the company style and its level of activity tracking — according to their business culture — time-tracking can be as simple as just closing a task, or it might involve several logs until a supervisor deems that the task can be safely closed.
- ✓ All these activities are tracked and made part of the overall project history.
- ✓ Optionally, dotProject can **send emails to the involved parties,** triggered by special conditions — such as a task being overdue, or having been completed so that a customer can be invoiced.
- ✓ While dotProject is not a fully-fledged invoicing system, it can produce enough data output to send reasonably detailed invoices to customers.
- ✓ At the same time, via its reporting facility, the management or the board can get properly formatted reports about ongoing projects, besides having access to the Gantt charts.
- ✓ Communication between team members can be as simple as leaving comments on tasks and/or logs, but dotProject also includes a minimalistic Forum facility. These are usually assigned to a single project (but each project can have several separate forums, with separate moderators, serving different purposes).
- ✓ And while dotProject is **not a sophisticated document management system**, it nevertheless allows files to be uploaded to a special directory, also assigned to specific projects/tasks, and under control of the permission system (file names get hashed, and only someone with the proper permission will be able to retrieve those files).
- ✓ There is a very simple built-in file management system to allow for file uploading and categorizing with metadata. The file folder can theoretically be mounted on an external file system on a cloud storage provider — so long as this is achieved at the operating system level; dotProject, by itself, does not connect directly to any storage provider. dotProject also includes a very simple versioning system.
- ✓ Tasks and milestones are also integrated into the built-in Calendar module, which is usually the preset entry point of the user — allowing them to keep up with the tasks they're involved in, or those that they supervise.
- ✓ There is some flexibility in how the information is presented. It is unknown if there is a way to automatically subscribe to a specific calendar; by contrast, Contacts, a module that allows editing the data related to each user, also permits exports using the vCard format.
- ✓ Its features include,
    - User Management
    - Ticketing Support System provided via email
    - Client Management
    - Task Listing
    - File Archive

- Contact List
- Calendar
- Discussion Forum

Launchpad

✓ Launchpad is a collaboration and project management software solution designed to be a unique collaboration and hosting platform for developing software projects. Some of the goals Launchpad was created for are:

- Encourage project contribution
- Develop, endorse and publish software
- Build communities through teams and mailing lists
- Improve collaboration with other developers and projects
- Share cross-project code, bug reports, ideas and translations

✓ Utilizing Launchpad removes the need for external mail hosting and provides users with a single account to manage their involvement in multiple mailing lists.

✓ Launchpad mailing lists are user-friendly and team-oriented. They things by allowing teams to **manage one central mailing list**. All team are able to subscribe to the team mailing list.

✓ **TECHNOLOGY:** Once you register as a user, Launchpad software can be assessed by logging online through their website. Launchpad provides convenient 24/7 access.

✓ **USERS**: Launchpad is designed primarily for small businesses of any type.

✓ **PRICING:** Launchpad is free to use for open source software projects and is available at a low annual subscription price for all other projects. It can be accessed from anywhere at any time.

FEATURES

• Bugs by email • Bug tracking • Share bug reports across projects • Code hosting and review • Direct links to code • Mailing lists • Blueprint specification tracking • Transform patches into fixes • Team PPAs • Multiple architectures • Web services API	• People profiles • Team branches • Team responsibility • Python library • Karma • Multi-language • Ubuntu package hosting • Automatic software updates and distribution • Information on publishing your software • Commercial project subscriptions • Quick and easy account authentication

open Proj

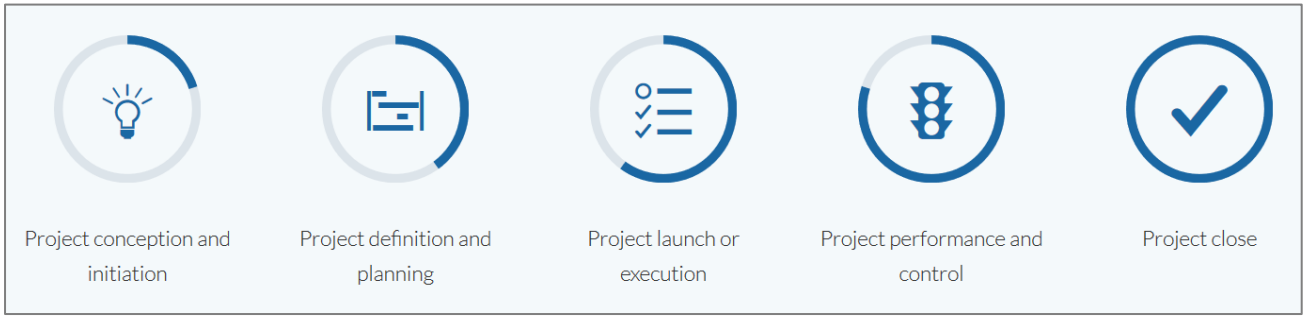
- ✓ Open Proj is open source project management software intended as a complete desktop replacement for Microsoft Project, being able to open existing native Project files. OpenProj runs on the Java Platform, allowing it to run on a variety of different operating systems
- ✓ OpenProject is free and open source software for classical as well as agile project management to support your team along the entire project life-cycle. OpenProject is available in more than 30 languages.
- ✓ OpenProject is licensed under GNU GPL V3. The source code is freely published on GitHub. We understand free as in free speech. We do offer paid subscriptions for our software.
- ✓ OpenProject exists since 2011 and is a fork of the deprecated ChiliProject which was a fork of Redmine.

First steps to get started

- ✓ To get started with OpenProject, there are a few easy steps to follow:
  - Get an account and sign in
  - Create a new project
  - Invite team members to collaborate
  - Create work packages
  - Set up a project plan

The entire Project Management Life-Cycle

- ✓ OpenProject offers a full feature set to support project teams along the entire project management process:



- ✓ OpenProject **enables project collaboration and communication without system interruption** from the initial project idea until project closure and documentation. The following sections provide links to the documentation for each project phase:

PROJECT PHASE	DOCUMENTATION FOR
Project concept and initiation	Collect ideas and specify project scope and deliverables: set up a project, document initial ideas, project description, invite members.

PROJECT PHASE	DOCUMENTATION FOR
Project definition and planning	Create a project overview with detailed information, set up a project plan, create your roadmap.
Project launch or execution	Manage all project activities, such as tasks, deliverables, risks, features, bugs, change requests. Use agile boards with your teams, document meetings, share news.
Project performance and control	Create and manage project budgets, track and evaluate time and costs. Have custom reports for accurate, current insight into project performance and allocated resources.
Project close	Document project achievements, lessons learned, best practices and easily summarize the main results of a project. Archive projects for later reference and lessons learned.

Project concept and initiation OpenProject supports the initial set-up and configuration of a project structure.

FEATURES	DOCUMENTATION FOR
Create a new project	Create and set up a new project in OpenProject
Set up a project structure	Create a project hierarchy to structure your work in OpenProject
Project settings	Create first ideas, tasks, rough milestones.
Add members	Invite your team to collaborate in OpenProject.

Project definition and planning Create a project overview with more detailed information, set up your project plan, structure your work, create a roadmap.

FEATURES	DOCUMENTATION FOR
Global projects overview	Create a project overview with important project information.

FEATURES	DOCUMENTATION FOR
Structure your work	Create work packages and structure your work
Roadmap planning	Create a roadmap for your project.

Project launch or execution Manage all project activities, such as tasks, deliverables, risks, features, bugs, change requests in the work packages. Use agile boards with your teams. Document meetings, share news.

FEATURES	DOCUMENTATION FOR
Work packages	Create and manage all project deliverables, tasks, features, risks, and more.
Boards	Manage your work with an Agile approach in the flexible boards view.
Meetings	Plan and document your project meetings and share minutes with all your team.
News	Share project news with your team.
Wiki	Document all important project information and keep it up to date with your team.

Project performance and control Create and manage project budgets, track and evaluate time and costs. Have custom reports for accurate, current insight into project performance and allocated resources.

FEATURES	DOCUMENTATION FOR
Dashboard	Visualize your progress within a project or project overarching.
Budgets	Create and manage budgets in your project.
Time tracking	Track time for any work within your project.
Track unit costs	Track unit costs for your project.
Time and cost reporting	Have accurate detailed reports of current spent time and costs within your project.

Project close Document project achievements, lessons learned, best practices and easily summarize the main results of a project. Archive projects for later reference and lessons learned.

FEATURES	DOCUMENTATION FOR
Wiki	Document all relevant project information, lessons learned, best practices, results, and more.
Project archive	Archive your project for further reference and documentation.

CASE STUDY PRINCE2 (refer internet)

6. Jim Lewis, “DEVOPS: A complete beginner’s guide to DevOps best practices”, ISBN-13:978-1673259148, ISBN-10: 1673259146, First Edition,2019

CO-PO Mapping

CO	POs					
	PO1	PO2	PO3	PO4	PO5	PO6
1	2	1	2	2	2	2
2	2	1	2	2	2	2
3	3	1	3	2	2	2
4	2	1	2	2	2	2
5	2	1	2	2	2	2
Avg	2.2	1	2.2	2	2	2

MC4204 MOBILE APPLICATION DEVELOPMENT L T P C
3 0 2 4

COURSE OBJECTIVES:

- To understand the need and characteristics of mobile applications.
- To design the right user interface for mobile applications.
- To understand the design issues in the development of mobile applications.
- To understand the development procedure for mobile applications.
- To develop mobile applications using various tools and platforms.

UNIT I INTRODUCTION 15

Mobile Application Model – Infrastructure and Managing Resources – Mobile Device Profiles – Frameworks and Tools

- Installation of necessary components and software

UNIT II USER INTERFACE 15

Generic UI Development - Multimodal and Multichannel UI –Gesture Based UI – Screen Elements and Layouts – Voice XML.

Lab Component:

i. Implement mobile applications using UI toolkits and frameworks.
ii. Design an application that uses Layout Managers and event listeners.

UNIT III APPLICATION DESIGN 15

Memory Management – Design Patterns for Limited Memory – Workflow for Application development – Java API – Dynamic Linking – Plugins and rule of thumb for using DLLs – Multithreading in Java - Concurrency and Resource Management.

Lab Component:

i. Design a mobile application that is aware of the resource constraints of mobile devices.
ii. Design an application that uses Dynamic Linking

UNIT IV MOBILE OS **15**

Mobile OS: Android, iOS – Android Application Architecture – Understanding the anatomy of a mobile application - Android basic components –Intents and Services – Storing and Retrieving data – Packaging and Deployment – Security and Hacking.

Lab Component:

i. Develop an application that makes use of mobile database
ii. Implement an android application that writes data into the SD card.

UNIT V APPLICATION DEVELOPMENT **15**

Communication via the Web – Notification and Alarms – Graphics and Multimedia: Layer Animation, Event handling and Graphics services – Telephony – Location based services

Lab Component:

i. Develop a web based mobile application that accesses internet and location data.
ii. Develop an android application using telephony to send SMS.

TOTAL:75 PERIODS

COURSE OUTCOMES:

On completion of the course, the student will be able to

CO1: Understand the basics of mobile application development frameworks and tools.
CO2: Develop a UI for mobile applications.
CO3: Design mobile applications that manage memory dynamically.
CO4: Build applications based on mobile OS like Android, iOs.
CO5: Build location based services.

SOFTWARE REQUIREMENTS

1. JDK, ECLIPSE IDE / equivalent, ANDROID STUDIO

REFERENCES

1. Reto Meier, Ian Lake, “Professional Android”, 4th Edition, Wrox, 2018.
2. Zigurd Mednieks, Laird Dornin, G. Blake Meike, Masumi Nakamura, “Programming Android”, O’Reilly, 2nd Edition, 2012.
3. Alasdair Allan, “Learning iOS Programming”, O’Reilly, Third Edition, 2013.
4. Bill Phillips, Chris Stewart, Brian Hardy, and Kristin Marsicano, Android Programming: The Big Nerd Ranch Guide, 4th edition, 2019.
5. Christian Keur, Aaron Hillegass, iOS Programming: The Big Nerd Ranch Guide, 6th Edition, O’Reilly, 2016.
6. Barry Burd, “Android Application Development All-In-One for Dummies”, 3rd Edition, 2021.

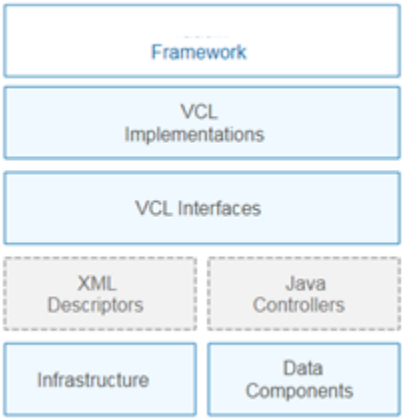
CO-PO Mapping

CO	POs					
	PO1	PO2	PO3	PO4	PO5	PO6
1	2	1	2	2	2	2

UNIT II USER INTERFACE

2.1 Generic UI Development - 2.2 Multimodal and Multichannel UI –2.3 Gesture Based UI – 2.4 Screen Elements and 2.5 Layouts – 2.6 Voice XML.

The **Generic User Interface** (**Generic UI**, **GUI**) framework allows you to create **UI** screens using Java and XML. XML is optional but it provides a declarative approach to the screen layout and reduces the amount of code which is required for building the **user interface**.



The application screens consist of the following parts:

- Descriptors – XML files for declarative definition of the screen layout and data components.
- Controllers – Java classes for handling events generated by the screen and its UI controls and for programmatic manipulation with the screen components.

The code of application screens interacts with visual component interfaces (VCL Interfaces). These interfaces are implemented using the Vaadin framework components.
Visual Components Library (VCL) contains a large set of ready-to-use components.
Data components provide a unified interface for binding visual components to entities and for working with entities in screen controllers.
Infrastructure includes the main application window and other common client mechanisms.

Screens and Fragments

A screen is a main unit of the generic UI. It contains visual components, data containers and non-visual components. A screen can be displayed inside the main application window either in the tab or as a modal dialog.

The main part of the screen is a Java class called controller. Layout of the screen is usually defined in an XML file called descriptor.

In order to show a screen, the framework creates a new instance of the Window visual component, connects the window with the screen controller and loads the screen layout components as child components of the window. After that, the screen's window is added to the main application window.

A fragment is another UI building block which can be used as part of screens and other fragments. It is very similar to screen internally, but has a specific lifecycle and the Fragment visual component instead of Window at the root of the components tree. Fragments also have controllers and XML descriptors.

Android UI Controls

There are number of UI controls provided by Android that allow you to build the graphical user interface for your app.

Sr.No.	UI Control & Description
1	TextView This control is used to display text to the user.
2	EditText EditText is a predefined subclass of TextView that includes rich editing capabilities.
3	AutoCompleteTextView The AutoCompleteTextView is a view that is similar to EditText, except that it shows a list of completion suggestions automatically while the user is typing.
4	Button A push-button that can be pressed, or clicked, by the user to perform an action.
5	ImageButton An ImageButton is an AbsoluteLayout which enables you to specify the exact location of its children. This shows a button with an image (instead of text) that can be pressed or clicked by the user.
6	CheckBox An on/off switch that can be toggled by the user. You should use check box when presenting users with a group of selectable options that are not mutually exclusive.
7	ToggleButton An on/off button with a light indicator.
8	RadioButton The RadioButton has two states: either checked or unchecked.
9	RadioGroup A RadioGroup is used to group together one or more RadioButtons.
10	ProgressBar The ProgressBar view provides visual feedback about some ongoing tasks, such as when you are performing a task in the background.
11	Spinner A drop-down list that allows users to select one value from a set.
12	TimePicker The TimePicker view enables users to select a time of the day, in either 24-hour mode or AM/PM mode.

13	DatePicker The DatePicker view enables users to select a date of the day.

Create UI Controls

Input controls are the interactive components in your app's user interface. Android provides a wide variety of controls you can use in your UI, such as buttons, text fields, seek bars, check box, zoom buttons, toggle buttons, and many more.

To create a UI Control/View/Widget you will have to define a view/widget in the layout file and assign it a unique ID as follows −

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical" >

  <TextView android:id="@+id/text_id"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="I am a TextView" />
</LinearLayout>
```

Events are a useful way to collect data about a user's interaction with interactive components of Applications.

There are following three concepts related to Android Event Management −

- **Event Listeners** − an event listener is an interface in the View class that contains a single callback method. These methods will be called by the Android framework when the View to which the listener has been registered is triggered by user interaction with the item in the UI.

- **Event Listeners Registration** − Event Registration is the process by which an Event Handler gets registered with an Event Listener so that the handler is called when the Event Listener fires the event.

- **Event Handlers** − When an event happens and we have registered an event listener for the event, the event listener calls the Event Handlers, which is the method that actually handles the event.

Event Listeners & Event Handlers

Event Handler	Event Listener & Description
onClick()	**OnClickListener()** This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. You will use onClick() event handler to handle such event.

onLongClick()	**OnLongClickListener()** This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. for one or more seconds. You will use onLongClick() event handler to handle such event.
onFocusChange()	**OnFocusChangeListener()** This is called when the widget looses its focus ie. user goes away from the view item. You will use onFocusChange() event handler to handle such event.
onKey()	**OnFocusChangeListener()** This is called when the user is focused on the item and presses or releases a hardware key on the device. You will use onKey() event handler to handle such event.
onTouch()	**OnTouchListener()** This is called when the user presses the key, releases the key, or any movement gesture on the screen. You will use onTouch() event handler to handle such event.
onMenuItemClick()	**OnMenuItemClickListener()** This is called when the user selects a menu item. You will use onMenuItemClick() event handler to handle such event.
onCreateContextMenu()	**onCreateContextMenuItemListener()** This is called when the context menu is being built(as the result of a sustained "long click)

2.2 Multichannel and Multimodal UIs

A multimodal interface for mobile devices requires the integration of several recognition technologies together with sophisticated user interface and distinct tools for input and output of data

The “smart mobile” has become an essential and inseparable part of our lives. This powerful tool enables us to perform multi-tasks in different modalities of voice, text, gesture, etc. The user plays an important role in the mode of operation, so multimodal interaction provides the user with new complex multiple modalities of interfacing with a system, such as speech, touch, type and more.

- Multimodality in mobile computing has become a very active field of research in the past few years.
- Mobile devices will allow smooth and smart interaction with everyday life‘s objects, thanks to natural and multimodal interactions.
- The multi-modal, multi-channel and multidevice notions are presented and are referenced by the name and partial acronym ―multi-DMC (Dynamic Multichannel)
- A multi-DMC referential is explained, in order to understand what kind of notions have to be sustained in such systems.
- To support at the same time different modalities including voice or gesture, different devices, like PC or smartphone and different channels such as web or telephone.
- However, in recent years, numerous scientific researches focus on post-WIMP interfaces.

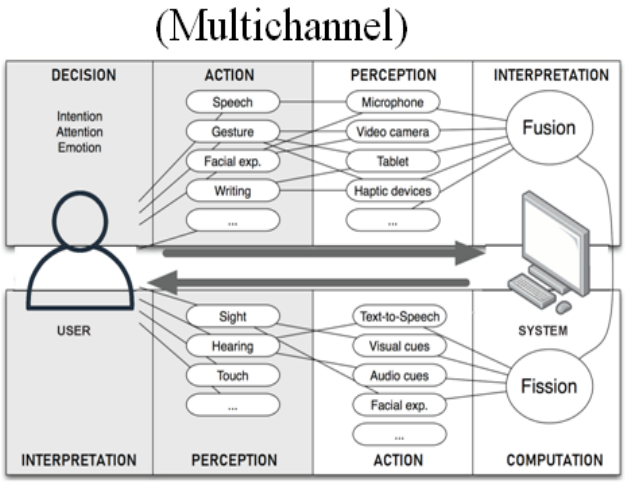
"windows, icons, menus, pointer"(WIMP) denoting a style of interaction using these elements of the user interface.

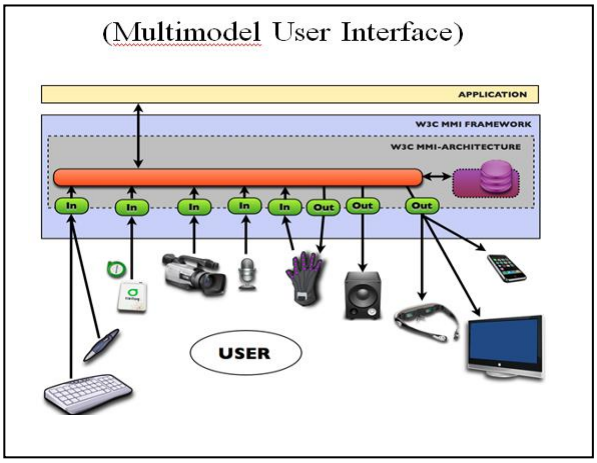
- In computing, a **window** is a graphical control element. It consists of a visual area containing some of the graphical user interface of the program it belongs to and is framed by a window decoration. It usually has a rectangular shape that can overlap with the area of other windows. It displays the *output* of and may allow *input* to one or more processes.
- It is no longer limited to a single way of interacting with a computer system, but considering the different solutions to offer user interfaces as natural as possible.
- With the introduction of many types of mobile devices, such as cellphones, Personal Digital Assistant (PDA), pocket PC, and the rise of their capabilities (Wifi, GPS, RFID, NFC(Near Field Communication)...) designing and deploying mobile interactive software that optimize the human-computer interaction has become a fundamental challenge.
- Modern terminals are natively equipped with many input and output resources needed for multimodal interactions, such as camera, vibration, accelerometer, stylus, etc.
- However, the main difference between multimedia and multimodal interaction lies in the semantic interpretation and the time management.
- Multimodality in mobile computing appears as an important trend, but a very few applications allow a real synergic (working together) multimodality.
- Multimodality tries to combine interaction means to enhance the ability of the user interface adaptation to its context of use, without requiring costly redesign and reimplementation.
- Blending multiple access channels provides new possibilities of interaction to users.
- Users have the possibility to switch between interaction means or to multiple available modes of interaction in parallel.
- In the context of ubiquitous and mobile computing, this situation of independent and collocated users performing unrelated tasks is however very likely to occur.
- Even if there is a risk of overlapping categories, design issues are often classified into management, technical and social issues.

- **Management:** mainly deals with registration and later identification of users and devices as they enter and leave the workspace environment.
- **Technical:** issue occurs with the control of specific device features and also the technical management of services offering the possibility to introduce (discover) or remove specific components from an interaction.
- **The design problems** is then related to fusion (i.e. combining multiple input types) and fission (i.e. combining multiple output types) mechanisms, synchronization and rules management between heterogeneous devices.
- **Social issues** are more related to social rules and privacy matters. As we know, some devices are inherently unsuitable for supporting privacy, such as microphones, speakers and public displays. The ubiquitous role of the computer makes each day more unsuitable for the screen-keyboard mouse model posed on a corner of a desk.

- In fact, the large success and rise of the Internet networks have complemented computing communication due to the technical standards used and their adoption of languages such as HTML, WML, or VoiceXML.
- An example of this incompatibility can be found in computers with different operating systems that process various types of media (texts, graphics, sounds, and video).
- Though the information can be easily transmitted through the networks, the formats of the coded data are incompatible.
- As a direct result the end-user bears additional cost and time lost when trying to obtain or utilize product and service based on their particular platform.
- This creates the urgent need for easier access to information — whether at the office, home, or on the train, etc.

- This need is felt all the more with the constant new arrival of soft/hardware materials, the success of the pocket computers and mobile telephones.
- With the multiplicity of the means of connecting to Internet, it is necessary to conceive generic interfaces and mechanisms of transformation to obtain concrete interfaces for each platform.





2.3 Gesture Based User Interface

Gestural UI refers to using specific gestures, like scrolling, pinching, and tapping to operate an interface. It also refers to gesture recognition, including tipping, tilting, eye motion, and shaking.

Tap and swipe are two common gestures that allow the user to perform primary actions on their mobile devices. The tap gesture is essentially a brief touch of the mobile screen surface with the fingertip. Common uses of this gesture in iOS and Android devices include: Select or submit.

Gestures are important for application of any category to **boost the app user interface** and provide comfort and ease of use to the users.

Gestures are **always linked to animations** in a mobile app. And these animations are essential to maintain an **illusion of interactivity** for the app users. So when they are paired with app gestures, they make the brain believe it's interacting with the tangible objects.

Uses of Gesture

Gestural UI is also commonplace in **the gaming, automotive, and medical industries**.
Popular consoles, such as Xbox, use cameras and sensors to track player movements and gestures for many of their interactive games.

Hand gestures

Hand gestures offer an inspiring field of research because they **can facilitate communication** and provide a natural means of interaction that can be used across a variety of applications. Previously, hand gesture recognition was achieved with wearable sensors attached directly to the hand with gloves.

Mobile device provides GestureDetector class to receive motion events and tell us that these events correspond to gestures or not.

Android provides special types of touch screen events such as pinch , double tap, scrolls , long presses and flinch. These are all known as gestures.

Android provides GestureDetector class to receive motion events and tell us that these events correspond to gestures or not. To use it , you need to create an object of GestureDetector and then extend another class with **GestureDetector.SimpleOnGestureListener** to act as a listener and override some methods.

```
GestureDetector myG;
myG = new GestureDetector(this,new Gesture());

class Gesture extends GestureDetector.SimpleOnGestureListener{
   public boolean onSingleTapUp(MotionEvent ev) {
   }

   public void onLongPress(MotionEvent ev) {
   }

   public boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceX,
   float distanceY) {
   }

   public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX,
   float velocityY) {
   }
}
```

Handling Pinch Gesture

Android provides **ScaleGestureDetector** class to handle gestures like pinch e.t.c. In order to use it, you need to instantiate an object of this class. Its syntax is as follow −

```
ScaleGestureDetector SGD;
SGD = new ScaleGestureDetector(this,new ScaleListener());
```

The first parameter is the context and the second parameter is the event listener.
To define the event listener and override a function **OnTouchEvent** to make it working.

```
public boolean onTouchEvent(MotionEvent ev) {
   SGD.onTouchEvent(ev);
   return true;
}

private class ScaleListener extends ScaleGestureDetector.SimpleOnScaleGestureListener {
   @Override
   public boolean onScale(ScaleGestureDetector detector) {
      float scale = detector.getScaleFactor();
      return true;
   }
}
```

There are other methods available that notify more about touch events. They are listed below −

Sr.No	Method & description
1	**getEventTime()** This method get the event time of the current event being processed..
2	**getFocusX()** This method get the X coordinate of the current gesture's focal point.
3	**getFocusY()** This method get the Y coordinate of the current gesture's focal point.
4	**getTimeDelta()** This method return the time difference in milliseconds between the previous accepted scaling event and the current scaling event.
5	**isInProgress()** This method returns true if a scale gesture is in progress..
6	**onTouchEvent(MotionEvent event)** This method accepts MotionEvents and dispatches events when appropriate.

Example main activity file - **src/MainActivity.java**.

```
public boolean onTouchEvent(MotionEvent ev) {
    SGD.onTouchEvent(ev);
    return true;
  }

  private class ScaleListener extends ScaleGestureDetector.
    SimpleOnScaleGestureListener {

    @Override
    public boolean onScale(ScaleGestureDetector detector) {
      scale *= detector.getScaleFactor();
      scale = Math.max(0.1f, Math.min(scale, 5.0f));
      matrix.setScale(scale, scale);
      iv.setImageMatrix(matrix);
      return true;
    }
  }
```

2.4 Screen elements / User Interface

The basic unit of mobile application is the activity. A UI is defined in an xml file. During compilation, each element in the XML is compiled into equivalent Android GUI class with attributes represented by methods.

Units of Measurement

When you are specifying the size of an element on an Android UI, you should remember the following units of measurement.

SNo	Unit & description
1	dp
	Density-independent pixel. 1 dp is equivalent to one pixel on a 160 dpi screen.
2	sp
	Scale-independent pixel. This is similar to dp and is recommended for specifying font sizes
3	pt
	Point. A point is defined to be 1/72 of an inch, based on the physical screen size.
4	px
	Pixel. Corresponds to actual pixels on the screen

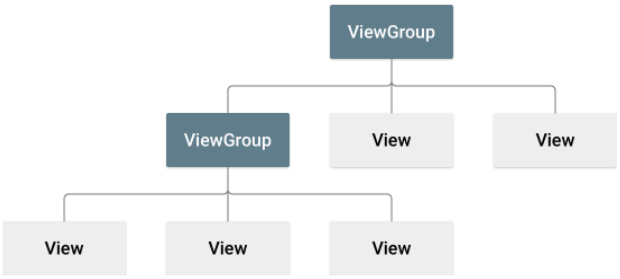
Screen Densities

Sr.No	Density & DPI
1	Low density (ldpi)
	120 dpi
2	Medium density (mdpi)
	160 dpi
3	High density (hdpi)
	240 dpi
4	Extra High density (xhdpi)
	320 dpi

View and ViewGroups

An activity contains Views and ViewGroups. A view is a widget that has an appearance on screen. Examples of views are buttons, labels, and text boxes. A view derives from the base class androidOne or more views can be grouped together into one ViewGroup.view.View.

A ViewGroup (which is itself a special type of view) provides the layout in which you can order the appearance and sequence of views. A ViewGroup derives from the base class android.view.ViewGroup.



The View objects are usually called "widgets" and can be one of many subclasses, such as Button or TextView. The ViewGroup objects are usually called "layouts" can be one of many types that provide a different layout structure, such as LinearLayout or ConstraintLayout .

The View is a base class for all UI components in android it is used to create interactive UI components such as TextView, EditText, Checkbox, Radio Button, etc. and it is responsible for event handling and drawing. For example, the **EditText** class is used to accept the input from users in android apps, which is a subclass of View.
The **View** is a base class for all UI components in android and

ViewGroup
The ViewGroup is a subclass of View and it will act as a base class for **layouts** and **layouts parameters**. The ViewGroup will provide an invisible containers to hold other **Views** or **ViewGroups** and to define the layout properties.

For example, Linear Layout is the ViewGroup that contains a UI controls like button, textview, etc. and other layouts also.

It can be defined a UI or input controls in two ways

Declare UI Elements in XML
The layout file must contain only one root element, which must be a View or ViewGroup object. Once we define the root element, then we can add additional layout objects or widgets as a child elements to build View hierarchy that defines our layout.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"

   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <TextView
      android:id="@+id/fstTxt"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Enter Name" />
   <EditText
      android:id="@+id/name"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:ems="10"/>
   <Button
      android:id="@+id/getName"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
```

```
        android:text="Get Name" />
</LinearLayout>
```

Create UI elements at runtime

```
public class MainActivity extends AppCompatActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      TextView textView1 = new TextView(this);
      textView1.setText("Name:");
      EditText editText1 = new EditText(this);
      editText1.setText("Enter Name");
      Button button1 = new Button(this);
      button1.setText("Add Name");
      LinearLayout linearLayout = new LinearLayout(this);
      linearLayout.addView(textView1);
      linearLayout.addView(editText1);
      linearLayout.addView(button1);
      setContentView(linearLayout);
   }
}
```

Types of UI Controls

Different type of UI controls available in android to implement the user interface for our android applications.

Following are the commonly used UI or input controls in android applications.

- TextView
- EditText
- AutoCompleteTextView
- Button
- ImageButton
- ToggleButton
- CheckBox
- RadioButton
- RadioGroup
- ProgressBar
- Spinner
- TimePicker
- DatePicker
- SeekBar
- AlertDialog
- Switch
- RatingBar

TextView

TextView is a user interface control that is used to display the text to the user.

EditText

EditText is a user interface control which is used to allow the user to enter or modify the text.

AutoCompleteTextView

AutoCompleteTextView is an editable text view which is used to show the list of suggestions based on the user typing text. The list of suggestions will be shown as a dropdown menu from which the user can choose an item to replace the content of the textbox.

Android Button

Button is a user interface control that is used to perform an action when the user clicks or tap on it.

Image Button

Image Button is a user interface control that is used to display a button with an image to perform an action when the user clicks or tap on it.
Generally, the Image button in android looks similar as regular Button and perform the actions same as regular button but only difference is for image button we will add an image instead of text.

Toggle Button

Toggle Button is a user interface control that is used to display ON (Checked) or OFF (Unchecked) states as a button with a light indicator.

CheckBox

Checkbox is a two-states button that can be either checked or unchecked.

Radio Button

Radio Button is a two-states button that can be either checked or unchecked and it cannot be unchecked once it is checked.

Radio Group

Radio Group is used to group one or more radio buttons into separate groups based on our requirements.

In case if we group radio buttons using the radio group, at a time only one item can be selected from the group of radio buttons.

ProgressBar

ProgressBar is a user interface control which is used to indicate the progress of an operation.

Spinner

Spinner is a drop-down list which allows a user to select one value from the list.

TimePicker

TimePicker is a widget for selecting the time of day, either in 24-hour or AM/PM mode.

DatePicker

DatePicker is a widget for selecting a date.

2.5 Layouts

A layout defines the structure for a user interface in your app, such as in an activity. All elements in the layout are built using a hierarchy of View and ViewGroup objects. A View usually draws something the user can see and interact with. Whereas a ViewGroup is an invisible container that defines the layout structure for View and other ViewGroup objects.

Each View and ViewGroup has a set of common attributes.

ATTRIBUTE	DESCRIPTION
layout_width	Specifies the width of the View or ViewGroup
layout_height	Specifies the height of the View or ViewGroup
layout_marginTop	Specifies extra space on the top side of the View or ViewGroup
layout_marginBottom	Specifies extra space on the bottom side of the View or ViewGroup
layout_marginLeft	Specifies extra space on the left side of the View or ViewGroup
layout_marginRight	Specifies extra space on the right side of the View or ViewGroup
layout_gravity	Specifies how child Views are positioned
layout_weight	Specifies how much of the extra space in the layout should be allocated to the View
layout_x	Specifies the x-coordinate of the View or ViewGroup
layout_y	Specifies the y-coordinate of the View or ViewGroup

Types of Layout

- LinearLayout
- AbsoluteLayout
- TableLayout
- RelativeLayout
- FrameLayout
- ScrollView

Linear Layout

Linear layout is further divided into horizontal and vertical layout. It means it can arrange views in a single column or in a single row. Here is the code of linear layout(vertical) that includes a text view.

```
<?xml version=”1.0” encoding=”utf-8”?>
<LinearLayout xmlns:android=”http://schemas.android.com/apk/res/android”
  android:layout_width=”fill_parent”
  android:layout_height=”fill_parent”
  android:orientation=”vertical” >

  <TextView
    android:layout_width=”fill_parent”
    android:layout_height=”wrap_content”
    android:text=”@string/hello” />
</LinearLayout>
```

AbsoluteLayout

The AbsoluteLayout enables you to specify the exact location of its children. It can be declared like this.

```
<AbsoluteLayout
  android:layout_width=”fill_parent”
  android:layout_height=”fill_parent”
  xmlns:android=”http://schemas.android.com/apk/res/android” >

  <Button
    android:layout_width=”188dp”
    android:layout_height=”wrap_content”
    android:text=”Button”
    android:layout_x=”126px”
    android:layout_y=”361px” />
</AbsoluteLayout>
```

TableLayout

The TableLayout groups views into rows and columns. You use the <TableRow> element to designate a row in the table. Each row can contain one or more views. Each view you place within a row forms a cell. The width of each column is determined by the largest width of each cell in that column.

```
<TableLayout
  xmlns:android=”http://schemas.android.com/apk/res/android”
  android:layout_height=”fill_parent”
  android:layout_width=”fill_parent” >

  <TableRow>
    <TextView
      android:text=”User Name:”
      android:width =”120dp”
      />

    <EditText
      android:id=”@+id/txtUserName”
      android:width=”200dp” />
  </TableRow>

</TableLayout>
```

RelativeLayout

The RelativeLayout enables you to specify how child views are positioned relative to each other.It can be declared like this.

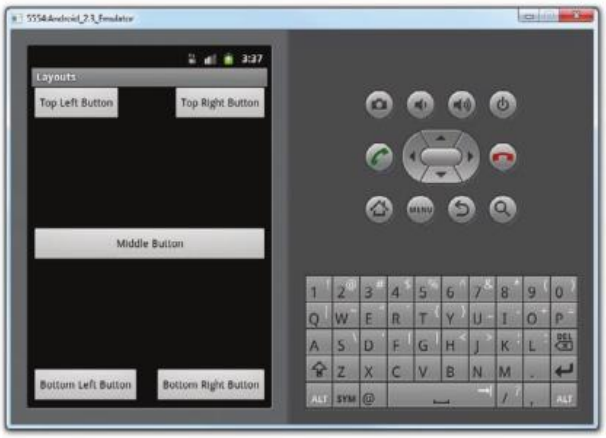
```
<RelativeLayout
  android:id=”@+id/RLayout”
  android:layout_width=”fill_parent”
  android:layout_height=”fill_parent”
  xmlns:android=”http://schemas.android.com/apk/res/android” >
</RelativeLayout>
```

Each view embedded within the RelativeLayout has attributes that enable it to align with another view. These attributes are as follows:

Observe the following attributes found in the various `Button` views:

- `layout_alignParentLeft` — Aligns the view to the left of the parent view
- `layout_alignParentRight` — Aligns the view to the right of the parent view
- `layout_alignParentTop` — Aligns the view to the top of the parent view
- `layout_alignParentBottom` — Aligns the view to the bottom of the parent view
- `layout_centerVertical` — Centers the view vertically within its parent view
- `layout_centerHorizontal` — Centers the view horizontally within its parent view

When the screen orientation changes to landscape mode, the four buttons are aligned to the four edges of the screen, and the center button is centered in the middle of the screen with its width fully stretched.



FrameLayout

The FrameLayout is a placeholder on screen that you can use to display a single view. It can be declared like this.

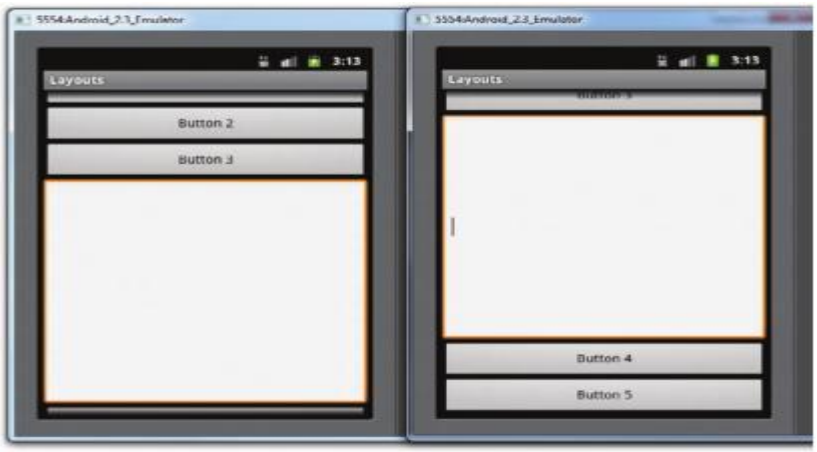
```
<?xml version=”1.0” encoding=”utf-8”?>
<FrameLayout
  android:layout_width=”wrap_content”
  android:layout_height=”wrap_content”
  android:layout_alignLeft=”@+id/lblComments”
  android:layout_below=”@+id/lblComments”
  android:layout_centerHorizontal=”true” >

  <ImageView
    android:src = “@drawable/droid”
    android:layout_width=”wrap_content”
    android:layout_height=”wrap_content” />
</FrameLayout>
```

Scrollview

A ScrollView is a special type of FrameLayout in that it enables users to scroll through a list of views that occupy more space than the physical display. The ScrollView can contain only one child view or ViewGroup, which normally is a LinearLayout.

```
<?xmlversion=”1.0”encoding=”utf-8”?>
<ScrollView android:layout_width=”fill_parent”
android:layout_height=”fill_parent”
xmlns:android=”http://schemas.android.com/apk/res/android”
>
```

```
<LinearLayout
android:layout_width=”fill_parent”
android:layout_height=”wrap_content”
android:orientation=”vertical”
>
<Button
android:id=”@+id/button1”
android:layout_width=”fill_parent”
android:layout_height=”wrap_content”
android:text=”Button1”
/>
<Button
android:id=”@+id/button2”
android:layout_width=”fill_parent”
android:layout_height=”wrap_content”
android:text=”Button2”
/>
<Button
android:id=”@+id/button3”
android:layout_width=”fill_parent”
android:layout_height=”wrap_content”
android:text=”Button3”
/>
<EditText
android:id=”@+id/txt”
android:layout_width=”fill_parent”
android:layout_height=”300px”
/>
<Button
android:id=”@+id/button4”
android:layout_width=”fill_parent”
android:layout_height=”wrap_content”
android:text=”Button4”
/>
<Button
android:id=”@+id/button5”
android:layout_width=”fill_parent”
android:layout_height=”wrap_content”
android:text=”Button5”
/>
</LinearLayout>
</ScrollView>
```



Resizing and Repositioning

Apart from anchoring your views to the four edges of the screen, an easier way to customize the UI based on screen orientation is to create a separate res/layout folder containing the XML files for the UI of each orientation. To support landscape mode, you can create a new folder in the res folder and name it as layout-land (representing landscape).

Basically, the main.xml file contained within the layout folder defines the UI for the activity in portrait mode, whereas the main.xml file in the layout-land folder defines the UI in landscape mode.

CREATING THE USER INTERFACE

```
packagenet.learn2develop.UICode;
importandroid.app.Activity;
importandroid.os.Bundle;
import android.view.ViewGroup.LayoutParams;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.TextView;

public class MainActivity extends Activity{
/**Calledwhentheactivityisfirstcreated.*/
@Override
public void onCreate(BundlesavedInstanceState){
super.onCreate(savedInstanceState);
//setContentView(R.layout.main);
//---param forviews--
      LayoutParams params = new LinearLayout.LayoutParams(
            LayoutParams.FILL_PARENT,
            LayoutParams.WRAP_CONTENT);
       //---create a layout--
      LinearLayout layout = new LinearLayout(this);
      layout.setOrientation(LinearLayout.VERTICAL);
       //---create a textview--
      TextView tv = new TextView(this);
      tv.setText("This is a TextView");
      tv.setLayoutParams(params);
       //---create a button--
      Button btn = new Button(this);
      btn.setText("This is a Button");
      btn.setLayoutParams(params);

      //---adds the textview--
      layout.addView(tv);

   //---adds the button--
      layout.addView(btn);
       //---create a layout param for the layout--
      LinearLayout.LayoutParams layoutParam = new LinearLayout.LayoutParams(
            LayoutParams.FILL_PARENT,
            LayoutParams.WRAP_CONTENT
);
       this.addContentView(layout, layoutParam);
}
}
```

UNIT III APPLICATION DESIGN

3.1 Memory Management – 3.2 Design Patterns for Limited Memory – 3.3 Work Flow for Applicationdevelopment –3.4 Java API – 3.5 Dynamic Linking – 3.6 Plugins and rule of thumb for using DLLs –3.7 Concurrency and 3.8 Resource Management.

3.1 Memory management

Memory management is part of an operating system which allocates memory among competing processes, maximizing memory utilization and system throughput.

It provides a convenient high level abstraction of low level hardware for programmers and compilers.

Tasks of Memory Management

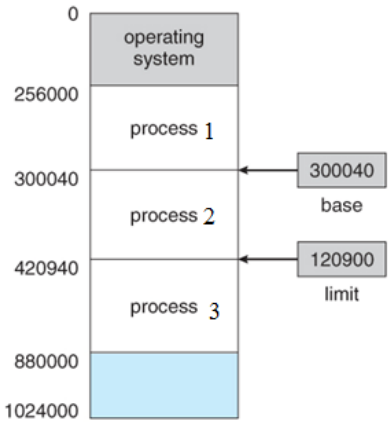
- It keeps track of each memory location either it is allocated to some process or it is free
- It checks how much memory should be allocated to processes
- It decides which process will get memory at what time
- It tracks whenever some memory gets unallocated and correspondingly it updates the status

Major Memory Management Techniques

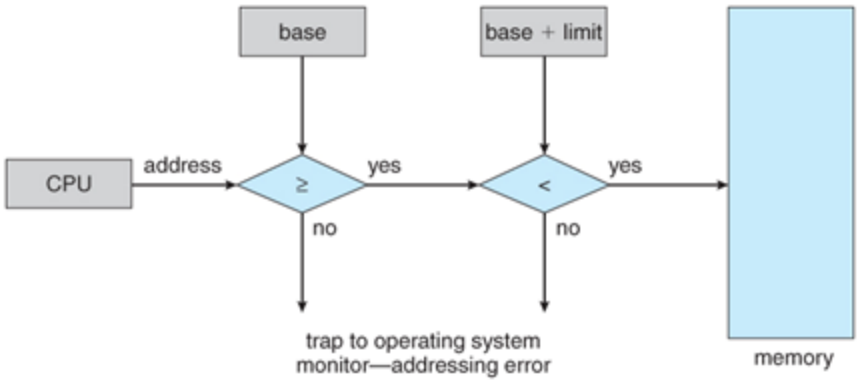
Base and limit registers

- Processes must be restricted so that they can only access memory locations that belong to that particular process
- Each process has a base register and limit register
  - The base register holds the smallest valid memory address
  - The limit register specifies the size of the range

- For example, the valid memory address for process 2 is from 300040 to 420940



- Every memory access from a user process is checked against these two registers.

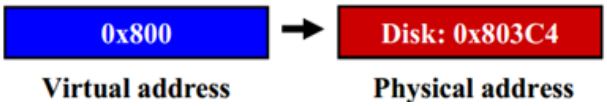


- The OS kernel has access to all memory locations as it needs to manage the whole memory

Virtual memory

Virtual memory (VM) is the basic abstraction that OS provides for memory management. "Virtual" means "using a level of indirection"

- ☐ All programs use virtual memory addresses
- ☐ Virtual address is converted to a physical address
- ☐ Physical address indicates the real physical location of data
- ☐ Physical location can be memory or disk

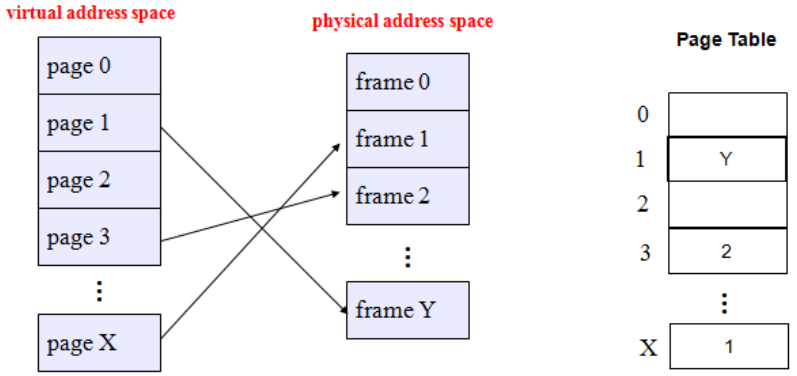


- The translation of virtual to physical addresses is handled by the memory-management unit (MMU).
- MMU uses a relocation register whose value is added to every memory request at the hardware level

Paging

- Paging is a memory management technique that allows the process's physical memory to be discontinuous
- It eliminates the fragmentation problem by allocating memory in equal sized blocks known as pages
- It is the predominant memory management technique now

- Paging divides physical memory into a number of equal sized blocks called frames and divides a process's logical memory space into equal sized blocks called pages
- Any page from any process can be placed into any available frame
- A page table is used to look up what frame a particular page is stored in at the moment



Garbage collection

The mechanism for reclaiming unused memory within a managed memory environment is known as garbage collection.

Garbage collection has two goals:

1. Find data objects in a program that cannot be accessed in the future
2. Reclaim the resources used by those objects.

Android's memory heap is a generational one, meaning that there are different buckets of allocations that it tracks, based on the expected life and size of an object being allocated.

Dalvik garbage collection:

- Each Android app runs in a Dalvik virtual machine and has its own Dalvik garbage collector
- The garbage collector maintains a free list which contains all the free memory blocks
- If a process requests a free block and the free list is empty, the garbage collector will be triggered to work:
  - Each block has a bit to indicate if it is in use
  - Mark the bits for blocks that are in use and cannot be collected. These mark bits are stored in a separate memory area
  - Sweep and collect all unmarked blocks and put them back to the free list

Share memory

- In order to fit everything it needs in RAM, Android tries to share RAM pages across processes.
  It can do so in the following ways:

- Each app process is forked (create new process) from an existing process called Zygote. The Zygote process starts when the system boots and loads common framework code and resources (such as activity themes).

- To start a new app process, the system forks the Zygote process then loads and runs the app's code in the new process.

- This approach allows most of the RAM pages allocated for framework code and resources to be shared across all app processes.

- Most static data is mmapped into a process. This technique allows data to be shared between processes, and also allows it to be paged out (To transfer memory contents to auxiliary storage) when needed.
- Example static data include: Dalvik code (by placing it in a pre-linked .odex file for direct mmapping), app resources (by designing the resource table to be a structure that can be mmapped and by aligning the zip entries of the APK), and traditional project elements like native code in .so files.

- APKs contain certain .odex files whose supposed function is to save space. Doing so speeds up the boot process, as it preloads part of an application.

In many places, Android shares the same dynamic RAM across processes using explicitly allocated shared memory regions (either with ashmem or gralloc).

Allocate and reclaim app memory

- The Dalvik heap is constrained to a single virtual memory range for each app process.
- This defines the logical heap size, which can grow as it needs to but only up to a limit that the system defines for each app.
- The logical size of the heap is not the same as the amount of physical memory used by the heap. When inspecting your app's heap, Android computes a value called the Proportional Set Size (PSS), which accounts for both dirty and clean pages that are shared with other processes—but only in an amount that's proportional to how many apps share that RAM.
- This (PSS) total is what the system considers to be your physical memory footprint. For more information about PSS, see the Investigating Your RAM Usage guide.
- Android can only shrink the logical heap size when there is unused space at the end of the heap.
- However, the system can still reduce physical memory used by the heap.

Switch apps

- When users switch between apps, Android keeps apps that are not foreground—that is, not visible to the user or running a foreground service like music playback— in a cache.
- For example, when a user first launches an app, a process is created for it; but when the user leaves the app, that process does not quit.
- The system keeps the process cached. If the user later returns to the app, the system reuses the process, thereby making the app switching faster.
- If your app has a cached process and it retains resources that it currently does not need, then your app—even while the user is not using it— affects the system's overall performance.
- As the system runs low on resources like memory, it kills processes in the cache.
- The system also accounts for processes that hold onto the most memory and can terminate them to free up RAM.

Restrict app memory

- To maintain a functional multi-tasking environment, Android sets a hard limit on the heap size for each app.
- The exact heap size limit varies between devices based on how much RAM the device has available overall.
- If your app has reached the heap capacity and tries to allocate more memory, it can receive an OutOfMemoryError.
- In some cases, you might want to query the system to determine exactly how much heap space you have available on the current device—for example, to determine how much data is safe to keep in a cache.
- You can query the system for this figure by calling getMemoryClass().
- This method returns an integer indicating the number of megabytes available for your app's heap.

iOS Memory Management

iOS include a fully-integrated virtual memory system that is always on
Like Android, iOS does not support swapping for the same reason

The principles to free up memory

- ❖ Read-only data which has a copy on the flash is simply removed from memory and reloaded from flash as needed
- ❖ Modified data is never removed from memory by the OS
- ❖ The system asks the running apps to free up memory voluntarily to make room for new data.
- ❖ Apps that fail to free up sufficient memory will be terminated by the OS
- ❖ iOS also uses the concept of paging
- ❖ Besides, it divides the virtual address space of a process into a number of regions. Each region contains a known number of pages
- ❖ The kernel associates a virtual memory (VM) object with each region and uses VM objects to track and manage the memory

---

3.2 Design Patterns for Limited Memory

A design pattern is just a convenient way of reusing object-oriented (OO) code between projects and programmers.

1. Design patterns are recurring solutions to design problems you see over and over.
2. Design patterns constitute a set of rules describing how to accomplish certain tasks in the realm of software development.
3. Design patterns focus more on reuse of recurring architectural design themes, while frameworks focus on detailed design and implementation
4. A pattern addresses a recurring design problem that arises in specific design situations and presents a solution to it.
5. Patterns identify and specify abstractions that are above the level of single classes and instances, or of components.
6. Design patterns are also about interaction between objects. One possible view of some of these patterns is to consider them as communication patterns.

The design patterns divides into three types:

• Creational patterns: create objects for you, rather than your having to instantiate objects directly. Your program gains more flexibility in deciding which objects need to be created for a given case.

• Structural patterns: help you compose groups of objects into larger structures, such as complex user interfaces and accounting data.

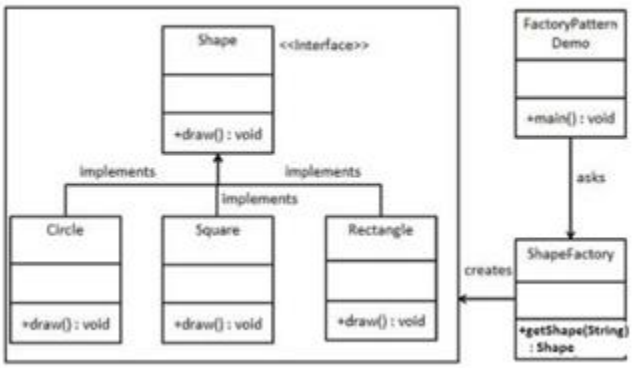
• Behavioral patterns: help you to define the communication between objects in your system and how the flow is controlled in a complex program.

Singleton Pattern

The singleton pattern is one of the simplest design patterns: it involves only one class which is responsible to instantiate itself, to make sure it creates not more than one instance; in the same time it provides a global point of access to that instance. In this case the same instance can be used from everywhere, being impossible to invoke directly the constructor each time.

```
class Singleton
{
        private static Singleton instance = new Singleton();
        private Singleton()
        {
                ...
        }
        public static synchronized Singleton getInstance()
        {
                return instance;
        }
        public void doSomething()
        {
                ...
        }
}
```

• When a interface does not know which sub-class of objects initilize it



```
public class SimplePizzaFactory {
    public Pizza createPizza(Strin
        Pizza pizza = null;

        if (type.equals("cheese"))
            pizza = new CheesePizz
        } else if (type.equals("pe
            pizza = new PepperoniP
        } else if (type.equals("cl
            pizza = new ClamPizza(
        } else if (type.equals("ve
            pizza = new VeggiePizz
        }
        return pizza;
    }
}
```

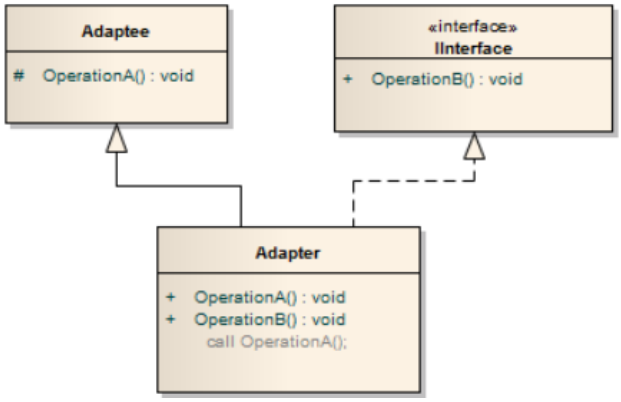
Structural Patterns

Adapter Design Pattern

•The adapter pattern is adapting between classes and objects. Like any adapter in the real world it is used to be an interface, a bridge between two objects.

•It's used to identifying a simple way to realize relationships between entities.

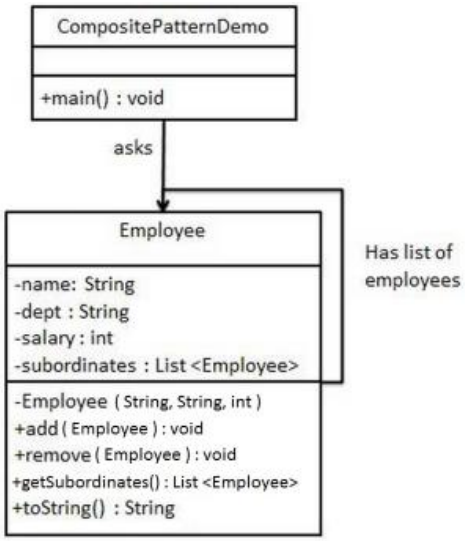
•The main use of this pattern is when a class that you need to use doesn't meet the requirements of an interface.



Composite Design Pattern

• Composite pattern is used where we need to treat a group of objects in similar way as a single object. Composite pattern composes objects in term of a tree structure to represent part as well as whole hierarchy

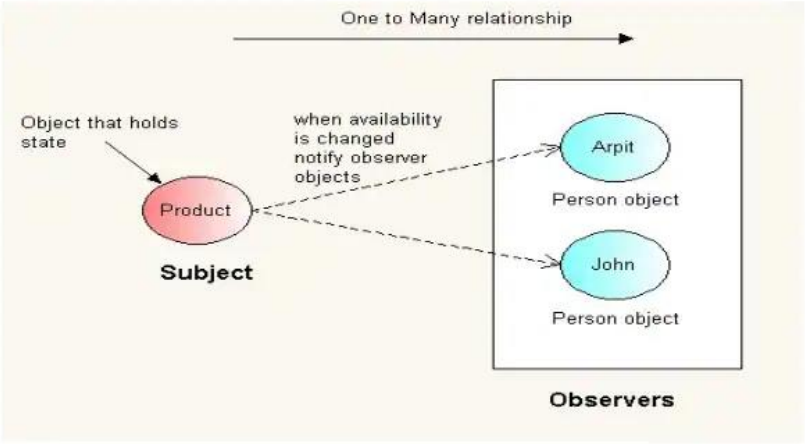
• This pattern creates a class contains group of its own objects. This class provides ways to modify its group of same objects.



Behavioral patterns

Observer Design Pattern

- Observer pattern is one of the behavioral design pattern.
- Observer design pattern is useful when you are interested in the state of an object and want to get notified whenever there is any change in observer pattern.
- The object that watch on the state of another object are called Observer and the object that is being watched is called Subject



Design Pattern use in Android

Three most common Design Patterns use in Android

1)MVC (Model View Controller)

2)MVP (Model View Presenter)

MVC (Model View Control)

1. Data Model, which contains the computational parts of the program
2. View, which presents the user interface
3. Controller, which interacts between the user and the view
4. Each aspect of the problem is a separate object , and each has its own rules for managing its data.
5. Communication between the user, the graphical user interface

(GUI), and the data should be carefully controlled

• Controllers are the activities themselves which contain all the business logic done in the application.

• Models are our entities which describe the components of our apps.

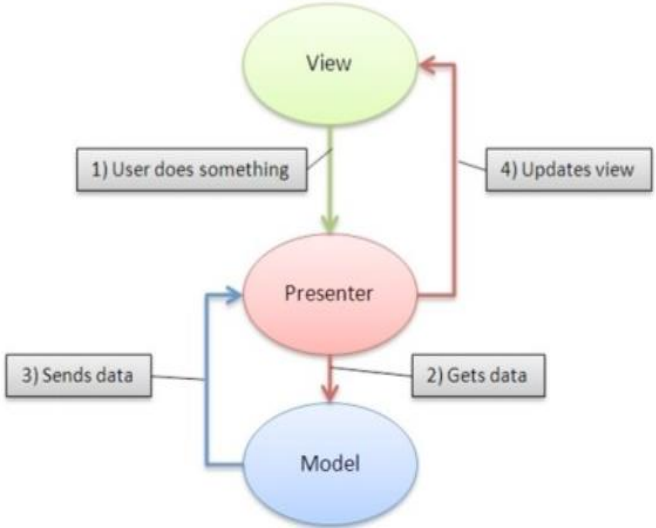
• Views can be done in XML layouts.

MVP (Model View Presenter)

• The MVP pattern allows **separate the presentation layer from the logic.**

• **Model-View-Presenter** pattern is a perfect fit for android development. Since the Views role in this pattern are:

Serving as a entry point
Rendering components
Routing user events to the presenter



The presenter

The presenter is responsible to act as **the middle man between view and model**. It retrieves data from the model and returns it formatted to the view.

The View

The view, usually implemented by an Activity (it may be a Fragment, a View… depending on how the app is structured), will contain a reference to the presenter.

The only thing that the view will do is calling a method from the presenter every time there is an interface action (a button click for example).

The model

In an application with a good layered architecture, this model would only be the gateway to the domain layer or business logic.

Write the Difference B/W MVC & MVP

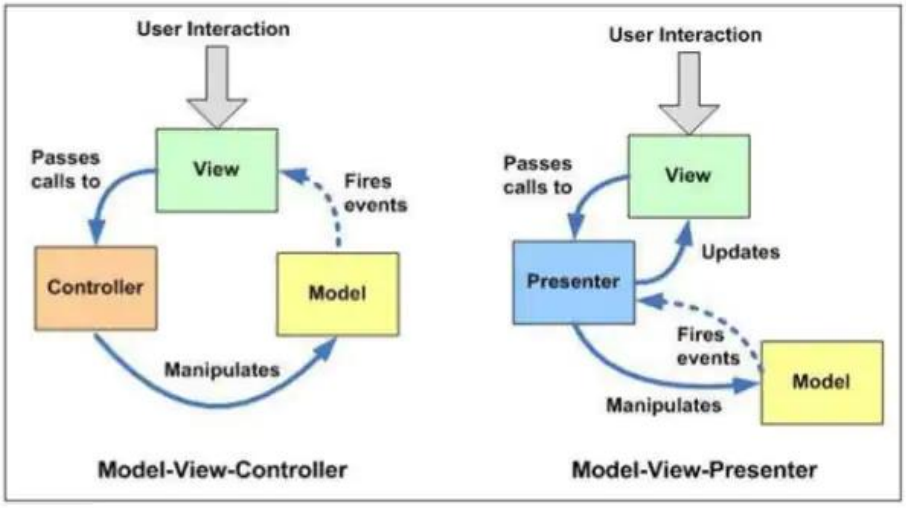
- MVC & MVP UI Presentation Pattern focus on separation of view with Model Based on MVC (UI Presentation Pattern)

- Separation of responsibility among three components:
  1.View - responsible for rendering UI elements
  2.Controller - responsible for responding to view actions
  3.Model - responsible for business behavior and state management

* Separation of responsibility among four components:
  1.View - Responsible for rendering UI elements
  2.View Interface - Responsible for loose coupling between view and model
  (Loose coupling is an approach to interconnecting the components in a system or network so that those components, also called elements, depend on each other to the least extent practicable.)
  3.Presenter - Responsible for view and model interaction
  4.Model - Responsible for business behavior and state management

- MVC MVP contains high degree of loose coupling
- Multiple views can share single controller Usually one view has one presenter (1-1 mapping).

Multiple presenters would be associated with a complex view Identifies which view to update Presenter will update its associated view



What are the classes uses in Design pattern?

The Following Android Classes uses Design Patterns

1) View Holder uses Singleton Design Pattern
   (ViewHolder describes **an item view and metadata about its place within the RecyclerView**.)
2) Intent uses Factory Design Pattern
3) Adapter uses Adapter Design Pattern
4) Broadcast Receiver uses Observer Design Pattern
5) View uses Composite Design Pattern
6) Media Framework uses Façade Design Pattern

3.3 WORK FLOW FOR APPLICATIONDEVELOPMENT

Mobile Applicaion development process is divided into several phases, these phases include-

1. The Requirement Analysis / Identification Phase

In the first phase, ideas are collected and categorized. The main objective of this phase is to come out with a new idea or improvements to the existing application. The ideas can come from the customer or from the developers.
With this, they would like to know about your purpose of the app and the audience you wish to provide to your app.
If no similar application exists on any mobile platform, then the idea with its core functionality should be documented. The other important task in this phase is to define the time required to develop the application. The initial requirement gathering should also be completed. The work done by the mobile application idea team should then be documented and forwarded to the design team.

2. The Project Scope Finalisation

This phase is about defining the actual project scope in detail. Here you as a business owner and your technology partner (Android app development company) stay in continuous coordination to finalise the detailed project scope. This project scope lists down the various functionalities of the app along with deciding the applications feature list. Based on the total feature list, the project scope is finalised and the final project scope decides the budget estimations and the time estimations for the development of your app. After the scope finalisation, you get a clear idea of the **mobile app development cost**.

3. The Wireframe Development Phase

This phase is the first phase where the work comes in progress and the project managers and developers design the wireframe of the app solution based on the project scope that was finalised. This wireframe decides the user journey. It includes all the basic functionalities that the user will get while he browses through the app, this will give a detailed idea of how the app will work and what would be the main functionalities of the app.

4. The Design Phase

In this phase, the idea from the mobile application team is developed into an initial design of the application. The feasibility of developing the application on all mobile platform is determined. Alternatively, the specific target mobile platform is identified.

The application functionality is broken down into modules and into prototypes i.e., combination of modules which are to be released in the prototype fashion. The functional requirements are defined.
The software architecture of the application is created. Then the prototypes and associated modules are defined. A very important part of the design phase is to create the storyboard for the user interface interaction: this storyboard describes the flow of the application. The design team's work is documented and forwarded to the development team for coding.

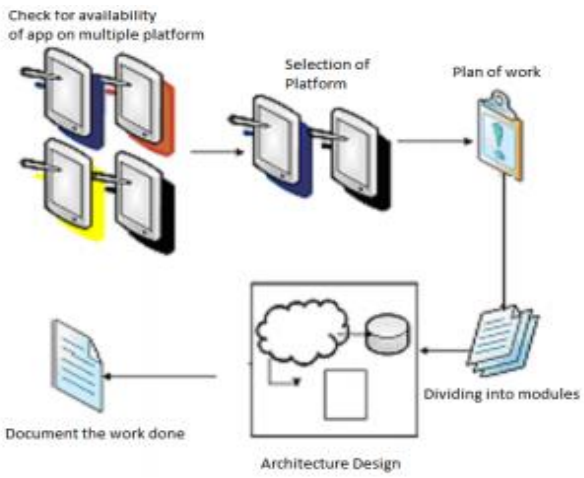


Fig 2. Design Phase

5. The Development Phase

In this phase, the application is coded. Coding for different modules of the same prototype can proceed in parallel. The development process can be in two stages: Coding for Functional Requirement and Coding for UI requirements. The code is developed first for the core functionalities. Parallel development can be done for modules of the same prototype that are independent of each other. Subsequently, these modules can be integrated. In the second stage, user interface is designed so that it can be supported on as many mobile operating system platforms as possible; Finally, the documentation of the development phase is then forwarded to the prototyping phase.

6. Prototyping Phase

In this phase, the functional requirements of each prototype are analyzed; the prototypes are tested and sent to the client for feedback. After feedback is received from the client, the required changes are implemented through the development phase. When the second prototype is ready, it is integrated with the first prototype, tested and then sent to the client. The development, prototyping and testing phases are repeated until the final prototype is ready. The final prototype is sent to the client for a final feedback.
The work done in this prototyping phase is documented and then forwarded to the testing phase.

7. The Quality Testing Phase

This phase involves the quality testing process, in this process the finally developed version is given to the dedicated QA (Quality Analysis) team and they test the entire application for its authenticity and its quality. Whether the app is developed along with the project scope and does it involve all the functionalities and features that are discussed in the project scope.
The quality analysists check the application for the various bugs and give back to the development team to resolve the bugs if any Testing is one of the most important phases of any development lifecycle model.
The testing of the prototype types is performed on an emulator/simulator followed by testing on the real device. The emulator/simulator is often provided in the SDK.
The test cases are documented and forwarded to the client for feedback.

8. Deployment Phase

Deployment is the final phase of the development process. After the testing is completed and the final feedback is obtained from the client, the application is ready for the deployment. The application is uploaded to the appropriate application store/market for user consumption. Before the application is deployed, the following steps are to be checked. Register as a developer on the respective

- Application developer's website by paying the annual fee, if necessary, for respective OS.
- Check the rules and regulations of the application store for the deployment of an application.
- Design the icon and wallpaper to be used on the application store.
- Create the file format required on operating system platform.

9. Maintenance / The Client Feedback Phase

In this phase, the quality tested app is delivered to the client and he checks the app on his part for its functionality and if it is according to the project scope he finalised. With this, he gets feedback for the app across his team and based on this he shares his feedback report, which may include minor modifications, corrections according to the scope, and he may highlight some bugs which he thinks might be crucial in application's performance.

Based on this the app comes back to the Android app development company along with the feedback shared by the client. Now according to the client's feedback, the final version of the app is prepared.

10. The Final Delivery Phase

After resolving all the feedbacks received from your side, the **mobile application development company** hands over the final deliverables (the final version of the app) to you. The app is delivered to you within the timeline estimates shared by you to the app development company and now the app is ready for the launch based on the launch schedule decided by you in the Google Play Store.

11. The App Marketing Phase

This phase is not included in the app solution development process but it comes just after the app launch and you as a business owner have to market your app on different marketing platforms like the web world, on your official company's website, your social media channels, through paid marketing channels, and through in-app promotions as well. This will help you get your app to your actual customer base and boost its popularity.

3.4 JAVA API

Mobile operating system using Java language as the simply for android apps. Most of android developer using java of any kind will put you in top standing for android apps coding.

Java Key Features for Mobile Apps

- Java portable code execute in all platforms
- Java support OOPS concept
- Easy APIs offer for tool for every conceivable task
- Open Source Libraries
- Huge Global Community with Android Support
- Easy to learn, implement and execute

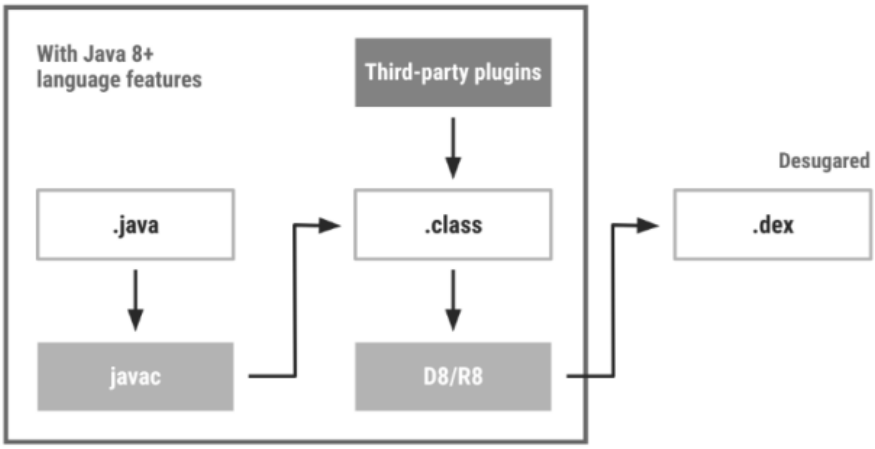


Figure 1. Java 8 language feature support using `desugar` bytecode transformations.

The Android Gradle plugin provides built-in support for using certain Java 8 language features and third-party libraries that use them.

What Are Java APIs?

APIs are important software components bundled with the JDK. APIs in Java include classes, interfaces, and user Interfaces. They enable developers to integrate various applications and websites and offer real-time information.

Need for Java APIs

Java developers use APIs to:

Streamline Operating Procedures

Social media applications like Twitter, Facebook, LinkedIn, and Instagram provide users with multiple options on one screen. Java APIs make this functionality possible.

Improve Business Techniques

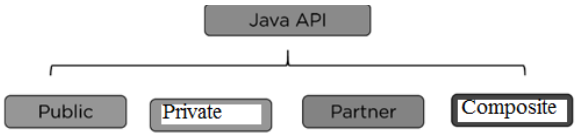
Introducing APIs to the public leads many companies to release private data to generate new ideas, fix existing bugs, and receive new ways to improve operations. The Twitter developer account is an example of an API that gives programmers private API keys to access Twitter data and develop applications.

Create Powerful Applications

Online banking has changed the industry forever, and APIs offer customers the ability to manage their finances digitally with complete simplicity.

Types of Java APIs

There are four types of APIs in Java:



- Public
- Private
- Partner
- Composite

Public

Public (or open) APIs are Java APIs that come with the JDK. They do not have strict restrictions about how developers use them.

Private

Private (or internal) APIs are developed by a specific organization and are accessible to only employees who work for that organization.

Partner

Partner APIs are considered to be third-party APIs and are developed by organizations for strategic business operations.

Composite

Composite APIs are microservices, and developers build them by combining several service APIs.

Data and API Services

Data and API services are another way to categorize Java APIs other than public, private, partner, and composite. APIs are also classified based on their data-manipulation capabilities and the variety of services they offer, including:

- ❖ Internal API services
- ❖ External API services
- ❖ CRUD
- ❖ User interface services

Internal API Services

Internal API services are developed to offer organizations services specific to that organization. These services include only complex data operations and internal processes.

External API Services

External APIs are open-source APIs that developers integrate into an existing application or website.

CRUD

CRUD APIs provide data manipulation operations over various data storage units such as software as a service (SaaS) and relational database management systems (RDBMS), using standard storage-unit connecting tools like Java Database Connectivity (JDBC).

User Interface Services

User interface service APIs are open-source APIs that allow developers to build user interfaces for mobile devices, computers, and other electronics.

API Service Protocols

The rules and protocols guide the functionality of the Java API. Different APIs have different service protocols.
For a typical RESTful API, developers must follow these rules:

Stateless

A RESTful API follows client-server architecture so it must be stateless.
Uniform Interface
The entities in a RESTful API are the server and clients. Applications that run on a global scale need a uniform client and server interface through the Hypertext Transfer Protocol (HTTP). Uniform Resource Identifiers (URIs) allocate the required resources.

Client-Server

The client-server model used in the RESTful API should be fault-tolerant. Both the client and server are expected to operate independently. The changes made at the client end should not affect the server end and vice versa.

Cache

Including a cache memory allows the application to record intermediate responses and run faster in real-time. A RESTful API also includes the cache memory.

Layered

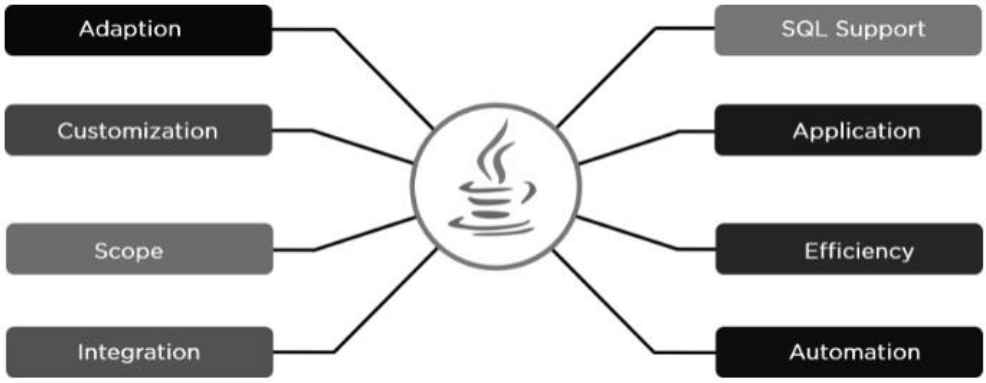
A RESTful API is built using layers. Layers in the API are loosely coupled, or independent, from each other. Each layer contributes to a different level of hierarchy and also supports encapsulation.

The Most Commonly Used Java APIs

API	Java Media Frameworks
RESTful API	Java Persistence API
Web API	Java Speech API
Facebook.4j	Java 3D
Twitter.4j	Java USB for Windows
JavaHelp	Android API
Java Advanced Imaging	Association of the Standardization of Embedded Platforms
Java Data Objects	Java Naming and Directory Interface

The Advantages of APIs

Some of the main advantages of using Java APIs include:



Extensive SQL Support

APIs in Java enable a wide range of SQL support services in user applications through a component-based interface.

Application

APIs in Java provide effortless access to all of an application's major software components and easily deliver services.

Efficiency

Java APIs are highly efficient because they enable rapid application deployment. Also, the data that the application generates is always available online.

Automation

APIs allow computers to automatically upload, download, update and delete data automatically without human interaction.

Integration

Java APIs can integrate into any application and website and provide a fluid user experience with dynamic data delivery.

Scope

Java APIs easily make websites, applications, and information available to a wide range of users and audiences.

Customization

Java APIs enable developers and businesses to build applications that personalize the user interface and data.

Adaptability

Java APIs are highly flexible and adaptable because they can easily accept feature updates and changes to frameworks and operating environments.

3.5 DYNAMIC / DEEP LINKS

Dynamic or Deep links are a type of connection meant to send users to an application directly rather than a store or a website. These links are under use to send users to a specific in-app location straight.

It will let the users save their plenty of time and effort required to search a particular element or reach a specific page on an application.

Deep linking can do this by specifying a custom URL scheme or an internal URL to open the application if it is installed already.

Dynamic Links in Firebase

Firebase Dynamic Links are a tool provided by Google' new Firebase platform to allow marketers to create a single link that will work the way you want, on multiple platforms, and whether or not your app is installed.

Dynamic links are smart URLs which allow us to send existing and potential users to any location within our Android and iOS app.

Dynamic Links Features

Here are some key features of Firebase Dynamic links that you must know:

- **Survive app install process**

Dynamic links are just like smart URLs that will help you to retarget your existing users to any of your preferred locations. These links can survive the process of application install which will let users see their desired content. More amazingly, these are free to use at any level and scale.

- **Add user to user sharing**

When a user is going to share content from your application, the final objective of this must be to convert his or her friends into active application users. The best way to achieve this is by presenting personalized content via deep linking through Firebase Dynamic Links.

- **Drive more installs**

You can add deep links to your promotional campaigns on every platform. It will let you enhance installs via effective social media, SMS, and email marketing campaigns.

- **Turn more users to mobile app**

Firebase Dynamic Links can let you help in migrating your desktop users to mobile applications. By this, you can give your users an easier way to send deep links on a mobile app.

Key benefits of Dynamic Links

There are the following benefits of **Dynamic Links**:

1. It is helpful in the conversion of web users to native app users.
2. It increases the conversion of user-to-user sharing.
3. It drives more installs with email, social, and SMS marketing campaigns.
4. It also helps to turn desktop users into mobile app users.

1. Converting web users to the app user

If a mobile web user installs our app by opening an app install link without a dynamic link, then they have to navigate again to where they were. With the help of dynamic links, we can make sure that after web users install our app, they can continue where they left off.

2. User-to-user sharing

Make it easy for our users to share our app's content with their friends. There is no need to worry about the platform, or either their friends already use our app or not.

3. Email, social, and SMS campaigns

Sending promotional offers using links which work on any platform. Current and future users can redeem our offer; either they use iOS, Android, or web browsers, or not and either they already install our app or not.

4. Real-world app promotion

We use Eddystone beacons and QR codes, which encode a Dynamic Link in our physical displays to promote our app at venues and events.

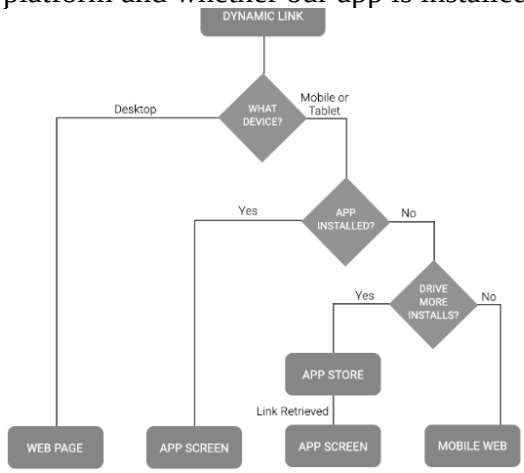
5. Converting desktop users to app users

Dynamic links are generated when web users bookmark a page or send themselves a link. If they open the link on a different device, they can get the best experience for the device.

- You can direct your web users to a specific part of your application with Firebase Dynamic Links.
- It can also help you to create a more effective, simple, and engaging social and email campaign in a more effective way.
- Email marketing is still an essential marketing tactic, and more amazingly, Firebase Dynamic Links can help you run successful and personalize email marketing campaigns with ease.
- Deep links via Firebase Dynamic links can also help you enable the user to share your application to increase your user base.
- Firebase Dynamic links can let you use short URLs for affiliate marketing as well.

Functioning of Dynamic Link

The dynamic link is created either by forming a URL by adding Dynamic Link parameters to a domain-specific to our app or by using the Firebase console, iOS, using a REST API or Android Builder API. These parameters specify the links we want to open, depending on the user's platform and whether our app is installed.



Many retailers will understand Firebase's dynamic linking capability as deep linking, i.e. retailers will be able to link to specific content within the app or on the web. Utilising a single link, Firebase can detect whether the user has the app and therefore direct the user to this deep link in-app. In the instance the user does not have the app installed, the retailer, on set-up, will have the option to choose whether to direct the user to the specific content on the web or to the App/Google Play Store to download the app.

3.7 CONCURRENCY IN MOBILE APPLICATION DEVELOPMENT

A scalable application in a multicore device environment, the Android developer should be capable of creating concurrent lines of execution that combine and aggregate data from multiple resources.

What is Concurrency

- Concurrency is the ability to run several programs or several parts of a program in parallel.
- If a time consuming task can be performed asynchronously or in parallel, this improve the throughput and the interactivity of the program

Concurrency issues

- Threads have their own call stack, but can also access shared data.
- Therefore you have two basic problems:-
  - visibility and
  - access problems

- A visibility problem occurs if thread A reads shared data which is later changed by thread B and thread A is unaware of this change.
- An access problem can occur if several thread access and change the same shared data at the same time.

Locks and thread synchronization

Visibility and access problem can lead to

- **Live ness failure**: The program does not react anymore due to problems in the concurrent access of data, e.g. deadlocks.
- **Safety failure**: The program creates incorrect data.

- Java provides *locks* to protect certain parts of the code to be executed by several threads at the same time.
- The simplest way of locking a certain method or Java class is to define the method or class with the **synchronized** keyword.

The ***synchronized*** keyword in Java ensures:

- that only a single thread can execute a block of code at the same time
- that each thread entering a synchronized block of code sees the effects of all previous modifications that were guarded by the same lock

- Synchronization is necessary for mutually exclusive access to blocks of and for reliable communication between threads.

```
public synchronized void critial() {
  // some thread critical stuff  here
.....
}
```

- You can also use the synchronized keyword to protect blocks of code within a method.

```
public void add(String site) {
   synchronized (this) {
     if (!crawledSites.contains(site)) {
       linkedSites.add(site);
     }
   }
 }
```

The Java memory model

- The *Java memory model* also defines the situations in which a thread re-fresh its own memory from the main memory.
- It also describes which operations are atomic and the ordering of the operations.

Atomic operation

- An atomic operation is an operation which is performed as a single unit of work without the possibility of interference from other operations
- reading or writing a variable is an atomic operation(unless the variable is of typelong or double).

- Operations variables of type long or double are only atomic if they declared with the volatile keyword.

Concurrent Processing in Android

- If input events or tasks are not handled concurrently, whole code of an Android application runs in the main thread and each line of code is executed one after each other.

- Assume if you perform a long lasting operation, for example accessing resource (like MP3, JSON, Image) from the Internet, the application goes hung state until the corresponding operation is finished.
- To bring good user experience in Android applications, all potentially slow running operations or tasks in an Android application should be made to run asynchronously.

AsyncTask in Android

- AsyncTask is an abstract Android class which helps the Android applications to handle the Main UI thread in efficient way.
- AsyncTask class allows us to perform long lasting tasks/background operations and show the result on the UI thread without affecting the main thread.

How to implement AsyncTask in Android applications?

- Create a new class inside Activity class and subclass it by extending AsyncTask

```
private class DownloadMp3Task extends AsyncTask<URL, Integer, Long> {
 protected Long doInBackground(URL... urls) {
    //Yet to code
    }
 protected void onProgressUpdate(Integer... progress) {
    //Yet to code
    }
 protected void onPostExecute(Long result) {
    //Yet to code
    }
}
```

onPreExecute:

- Invoked before the task is executed ideally before doInBackground method is called on the UI thread. This method is normally used to setup the task like showing progress bar in the UI.

doInBackground:

- Code running for long lasting time should be put in doInBackground method. When execute method is called in UI main thread, this method is called with the parameters passed.

onProgressUpdate:

- Invoked by calling publishProgress at anytime from doInBackground. This method can be used to display any form of progress in the user interface.

onPostExecute:

- Invoked after background computation in doInBackground method completes processing. Result of the doInBackground is passed to this method.

Concurrent package constructs

Other Java concurrent constructs provided by java.util.concurrent, which are also available on Android SDK are as follows:

Lock objects (java.util.concurrent): They implement locking behaviors with a higher level idiom.
Executors: These are high-level APIs to launch and manage a group of thread executions (ThreadPool, and so on).
Concurrent collections: These are the collections where the methods that change the collection are protected from synchronization issues.
Synchronizers: These are high-level constructs that coordinate and control thread execution (Semaphore, Cyclic Barrier, and so on).
Atomic variables (java.util.concurrent.atomic): These are classes that provide thread-safe operations on single variables. One example of it is AtomicInteger that could be used in our example to solve the correctness issue.

Some Android-specific constructs use these classes as basic building blocks to implement their concurrent behavior, although they could be used by a developer to build custom concurrent constructs to solve a specific use case.

Executor framework

The Executor framework is another framework available on java.util.concurrent that provides an interface to submit Runnable tasks, decoupling the task submission from the way the task will run:

```
public interface Executor {
  void execute(Runnable command);
}
```

Each Executor, which implements the interface that we defined earlier, can manage the asynchronous resources, such as thread creation destruction and caching, and task queueing in a variety of ways to achieve the perfect behavior to a specific use case.

The java.util.concurrent comes with a group of implementations available out of the box that cover most generic use cases, as follows:

Executors.newCachedThreadPool(): This is a thread poll that could grow and reuse previously created threads
Executors.newFixedThreadPool (nThreads): This is a thread pool with a fixed number of threads and a message queue for store work
Executors.newSingleThreadPool(): This is similar to newFixedThreadPool, but with only one working thread
To run a task on Executor, the developer has to invoke execute() by passing Runnable as an argument:

```
public class MyRunnable implements Runnable {
   public void run() {
      Log.d("Generic", "Running From Thread " +
          Thread.currentThread().getId());
          // Your Long Running Computation Task
   }
}
public void startWorking(){
   Executor executor = Executors.newFixedThreadPool(5);
   for ( int i=0; i < 20; i++ ) {
      executor.execute(new MyRunnable());
   }
}
```

Multithreading in Java using Android

The most basic one, java.lang.Thread, is the class that is mostly used and is the construct that creates a new independent line of execution in a Java program:

MyThread myThread = new MyThread();

myTread.start();

At this time, create an instance of our MyThread, and when start it in the second line, the system creates a thread inside the process and executes the run() method.

Thread.currentThread(): This retrieves the current running instance of the thread

Thread.sleep(time): This pauses the current thread from execution for the given period of time

Thread.getName() and Thread.getId(): These get the name and TID, respectively so that they can be useful for debugging purposes

Thread.isAlive(): This checks whether the thread is currently running or it has already finished its job

Thread.join(): This blocks the current thread and waits until the accessed thread finishes its execution or dies

Created the Runnable subclass so that it implements the run() method and can be passed and executed by a thread:

```
public class MyRunnable implements Runnable {

   public void run(){
      Log.d("Generic","Running in the Thread " +
                  Thread.currentThread().getId());
         // Do your work here
         ...
   }
}
```

Now our Runnable subclass can be passed to Thread and is executed independently in the concurrent line of execution:

```
Thread thread = new Thread(new MyRunnable());
thread.start();
```

3.8 RESOURCE MANAGEMENT IN MOBILE APPLICATION DEVELOPMENT

In addition to coding the application, to build an excellent application, you need to focus on a variety of resources, such as you use a variety of static content, such as bitmaps, colors, layout definition, user interface strings, animation and so on. These resources are generally placed in the project's res / standalone subdirectory.

res / directory contains all of the resources in various subdirectories. Here's a picture resource, two layout resources and a string resource file. The following table gives a detailed in the project res / directory support resources.

Resources	Resource Type
anim /	XML file that defines the animation property. They are saved in res / anim / folder, by type of access R.anim
color /	XML file that defines the color status list. They are saved in res / color / folder, by type of access R.color
drawable /	Image files, such as .png, .jpg, .gif, or XML file, is compiled as a bitmap, state list, shapes, animated images. They are saved in res / drawable / folder, by type of access R.drawable
layout /	Custom UI XML file layout. They are saved in res / layout / folder, by type of access R.layout
menu /	Custom application menu XML files, such as the Options menu, context menus, sub-menus. They are saved in res / menu / folder, by type of access R.menu
raw /	Any files are saved in their original form. We need to R.raw.filename named resource ID, to open the raw file by calling Resource.openRawResource ()
values /	XML files contain simple values (such as strings, integers, color, etc.).

Alternative Resources

Your application needs to provide resources to support an alternative configuration for a specific device. For example, you need to provide an alternative picture of resources for different screen resolutions, providing an alternative string resources for different languages. At runtime, Android detects the current device configuration, and load the appropriate resources for an application.

To identify a set of alternative resources for specific configuration, following the steps of:

Create a new directory res / down to <resource_name> _ <config_qualifier> are named.

Saving alternative resources in response to this directory.

Access to resources

In application development, we need access to defined resources, either through code or through XML files.

When the Android application is compiled to generate a class R, which contains the ID and all res / directory resources. You can use the class R, + by subclass resource name or directly use the resource ID to access resources.

Examples

Access res / drawable / myimage.png, and set it to the ImageView, you would use the following code:

```
ImageView imageView = (ImageView) findViewById(R.id.myimageview);
imageView.setImageResource(R.drawable.myimage);
```

The first line of code in here to get R.id.myimageview defined myimageview of ImageView in the layout file. The second line with R.drawable.myimage to get the image in res / drawable in a subdirectory called myimage.

Examples

Consider an example in which res / values / strings.xml defined as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string  name="hello">Hello, World!</string>
</resources>
```

Now you can use the ID of the object msg of TextView resource ID to set the text, as follows:

```
TextView msgTextView = (TextView) findViewById(R.id.msg);
msgTextView.setText(R.string.hello);
```

Access in XML

Consider the following XML resource file res / values / strings.xml, which contains a color resource and a string resource -

```
<?xml version="1.0" encoding="utf-8"?>
<EditText xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:textColor="@color/opaque_red"
   android:text="@string/hello" />
```

Now, you can use these resources in the following layout file to set the text color and text content:

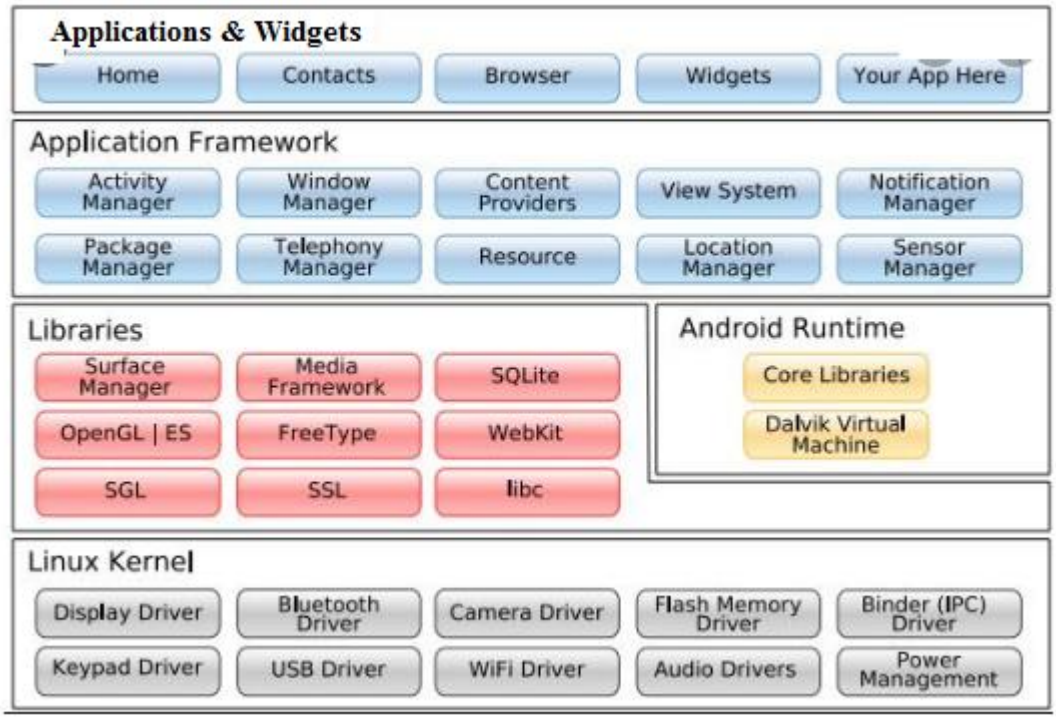
```
<code>&lt;?xml version="1.0" encoding="utf-8"?&gt;
&lt;EditText xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:textColor="@color/opaque_red"
   android:text="@string/hello" /&gt;
</code>
```

UNIT IV MOBILE OS

Mobile OS: Android, iOS – Android Application Architecture – Android basic components – Intents and Services – Storing and Retrieving data – Packaging and Deployment – Security and Hacking.

Android operating system is a stack of software components which is roughly divided into five sections and four main layers as shown below in the architecture diagram.

ANDROID ARCHITECTURE



Linux kernel

Linux Kernel is heart of the android architecture. It manages all the available drivers such as display drivers, camera drivers, Bluetooth drivers, audio drivers, memory drivers, etc. which are required during the runtime.

The Linux Kernel will provide an abstraction layer between the device hardware and the other components of android architecture. It is responsible for management of memory, power, devices etc.

The features of Linux kernel are:

- **Security:** The Linux kernel handles the security between the application and the system.
- **Memory Management:** It efficiently handles the memory management thereby providing the freedom to develop our apps.
- **Process Management:** It manages the process well, allocates resources to processes whenever they need them.
- **Network Stack:** It effectively handles the network communication.
- **Driver Model:** It ensures that the application works properly on the device and hardware manufacturers responsible for building their drivers into the Linux build.

Android Libraries

This category encompasses those Java-based libraries that are specific to Android development. Examples of libraries in this category include the application framework libraries in addition to those that facilitate user interface building, graphics drawing and database access. A summary of some key core Android libraries available to the Android developer is as follows −

- **android.app** − Provides access to the application model and is the cornerstone of all Android applications.
- **android.content** − Facilitates content access, publishing and messaging between applications and application components.

Platform libraries –

The Platform Libraries includes various C/C++ core libraries and Java based libraries such as Media, Graphics, Surface Manager, OpenGL etc. to provide a support for android development.

- **Media** library provides support to play and record an audio and video formats.
- **Surface manager** responsible for managing access to the display subsystem.
- **SGL** and **OpenGL** both cross-language, cross-platform application program interface (API) are used for 2D and 3D computer graphics.
- **SQLite** provides database support. It is used to access data published by content providers and includes SQLite database management classes.and **FreeType** provides font support.
- **Web-Kit** This open source web browser engine provides all the functionality to display web content and to simplify page loading.
- widget − A rich collection of pre-built user interface components such as buttons, labels, list views, layout managers, radio buttons etc.
- Text − Used to render and manipulate text on a device display.
- View − The fundamental building blocks of application user interfaces.
- **SSL (Secure Sockets Layer)** is security technology to establish an encrypted link between a web server and a web browser.

Android Runtime

Android Runtime environment is one of the most important part of Android. It contains components like core libraries and the Dalvik virtual machine(DVM). Mainly, it provides the base for the application framework and powers our application with the help of the core libraries.

Like Java Virtual Machine (JVM), **Dalvik Virtual Machine (DVM)** is a register-based virtual machine and specially designed and optimized for android to ensure that a device can run multiple instances efficiently. It depends on the layer Linux kernel for threading and low-level memory management.

Android uses DVM to optimize battery life, memory and performance. The byte code generated by the **Java compiler** has to be converted to .dex file by DVM, as it has its own byte code. Also, multiple class files are created as one .dex file and the compressed .jar file is greater than the uncompressed .dex file.

The Android runtime also provides a set of core libraries which enable Android application developers to write Android applications using standard Java programming language.

Application Framework

The Application Framework layer provides many higher-level services to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications.

The Android framework includes the following key services −

- Activity Manager − Controls all aspects of the application lifecycle and activity stack.
- Content Providers − Allows applications to publish and share data with other applications.
- Resource Manager − Provides access to non-code embedded resources such as strings, color settings and user interface layouts.
- Notifications Manager − Allows applications to display alerts and notifications to the user.
- View System − An extensible set of views used to create application user interfaces.

Applications

You will find all the Android application at the top layer. The pre-installed applications like home, contacts, camera, gallery etc and third party applications downloaded from the play store like chat applications, games etc. will be installed on this layer only.

It runs within the Android run time with the help of the classes and services provided by the application framework.

---

iOS

iOS is a mobile operating system developed and distributed by Apple Inc. It was originally released in 2007 for the iPhone, iPod Touch, and Apple TV. iOS is derived from Mac OS X, with which it shares the Darwin foundation. iOS is Apple's mobile version of the OS X operating system used in Apple computers.

It is developed by objective-C and C language.Not opensource.

Features of iOS

The power of iOS can be felt with some of the following features provided as a part of the device.

- Maps- web mapping service for Location information.
- Siri - Siri is the voice assistant on Apple devices Google's Google Assistant.
- Facebook and Twitter
- Multi-Touch- It's an entirely new interface based on a large **multi-touch** display and innovative new software that lets you control everything using only your fingers.
- Accelerometer- You access the raw **accelerometer** data using the classes of the Core Motion framework.
- GPS - **GPS**-enabled **iOS** device will also periodically send **GPS** locations, travel speed etc
- High end processor -
- Camera -
- Safari - **Safari** browser is the best way to experience the Internet on all your Apple devices. It brings robust customisation options, powerful privacy protections and industry-leading battery life.

- Powerful APIs -
- Game center - **Game Center** is a service by **Apple** that allows users to play and challenge friends when playing online multiplayer social gaming network games.
- In-App Purchase
- Reminders
- Wide Range of gestures

iOS Architecture

The iOS is the operating system created by Apple Inc. for mobile devices. The iOS is used in many of the mobile devices for apple such as iPhone, iPod, iPad etc.

The iOS architecture is layered. It contains an intermediate layer between the applications and the hardware so they do not communicate directly.
The lower layers in iOS provide the basic services and the higher layers provide the user interface and sophisticated graphics.

Apple iOS Architecture

Layers in iOS Architecture

The different layers as shown in the above diagram are given as follows −

Core OS

All the iOS technologies are built on the low level features provided by the Core OS layer. These technologies include Core Bluetooth Framework, External Accessory Framework, Accelerate Framework, Security Services Framework, Local Authorization Framework etc.

Core Services

There are many frameworks available in the core services layer. Details about some of these are given as follows −

Cloud kit Framework

The data can be moved between the app the iCloud using the Cloudkit Framework.

Core Foundation Framework

This provides the data management and service features for the iOS apps.

Core Data Framework

The data model of the model view controller app is handled using the Core Data Framework.

Address Book Framework

The address book framework provides access to the contacts database of the user.

Core Motion Framework

All the motion based data on the device is accessed using core motion framework.

Health kit Framework

The health related information of the user can be handled by this new framework.

Core Location Framework

This framework provides the location and heading information to the various apps.

Core Location is a framework that's included in the iOS SDK that can be used to determine location and heading with a device. Location is found using GPS or assisted GPS on the device. Assisted GPS helps retrieve your location quicker by using the cellular or Wi-Fi network to triangulate your location.

Media

The media layer enables all the graphics, audio and video technology of the system. The different frameworks are:

UIKit Graphics

This provides support for designing images and animating the view content.

Core Graphics Framework

This provides support for 2-D vector and image based rendering and is the native drawing engine for iOS apps.

Core Animation

The Core Animation technology optimizes the animation experience of the apps.

Media Player Framework

This framework provides support for playing playlists and enables the user to use their iTunes library.

AV Kit

This provides various easy to use interfaces for video presentation.

Cocoa Touch

The cocoa touch layer provides the following frameworks −

Event Kit Framework

This shows the standard system interfaces using view controllers for viewing and changing calendar related events.

Game Kit Framework

This provides support for users to share their game related data online using Game center.

Map Kit Framework

This provides a scrollable map which can be included into the app user interface.

ANDROID - BASIC COMPONENTS

Application components are the essential building blocks of an Android application.

There are following four main components that can be used within an Android application.

Sr.No	Components & Description
1	**Activities -** They dictate the UI and handle the user interaction to the smart phone screen.
2	**Services -** They handle background processing associated with an application.
3	**Broadcast Receivers -** They handle communication between Android OS and applications.
4	**Content Providers -** They handle data and database management issues.

Activities

An Android *activity* **represents a screen with which a user can interact. The** Activity **class is also used to respond to user input. An activity can transition to another activity as the user navigates between screens.**
An activity is running when it's in the foreground of the screen.
For example, a contacts app that is having multiple activities like showing a list of contacts, add a new contact, and another activity to search for the contacts. All these activities in the contact app are independent of each other but will work together to provide a better user experience.
This handles the user interactions with the smartphone display.

An activity is implemented as a subclass of Activity class as follows −

```
public class MainActivity extends Activity {
}
```

Services

A service is a component that runs in the background to perform long-running operations. For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network (download files) without blocking user interaction with an activity.
A service is implemented as follows −

```
public class MyService extends Service {
}
```

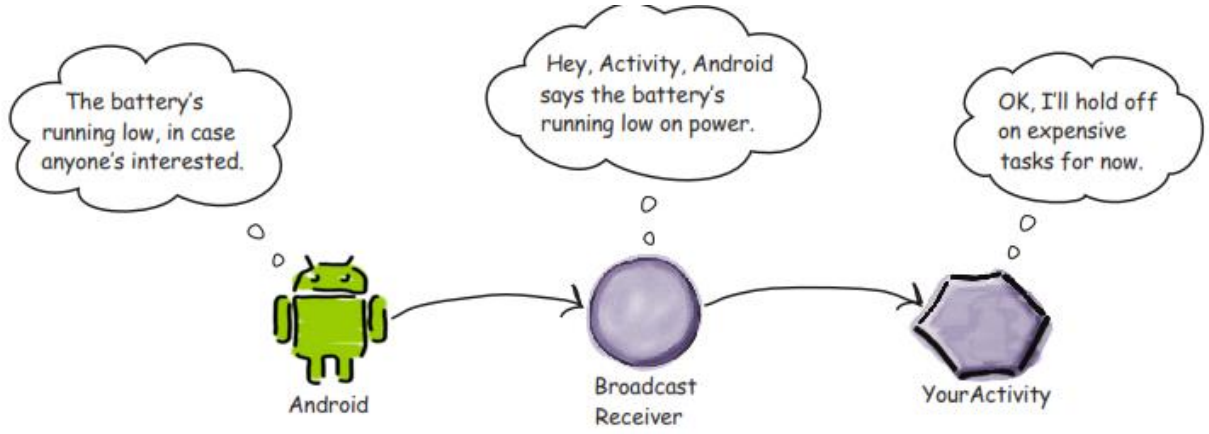
Broadcast Receivers

The broadcast receivers handle the communication between the applications and the android operating system.
Android broadcasts system events when they occur, including things like the device running low battery power, a new incoming phone call, or the system getting booted.

Broadcast Receivers simply respond to broadcast messages from other applications or from the system. For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, and will initiate appropriate action.
If you insert headset into mobile, this ∩ headset symbol is displayed through this broadcast receiver. Likewise if you charge mobile, mobile charging is shown by this receiver.

A broadcast receiver is implemented as a subclass of **BroadcastReceiver** class.

```
public class MyReceiver  extends  BroadcastReceiver {
  public void onReceive(context,intent){}
}
```

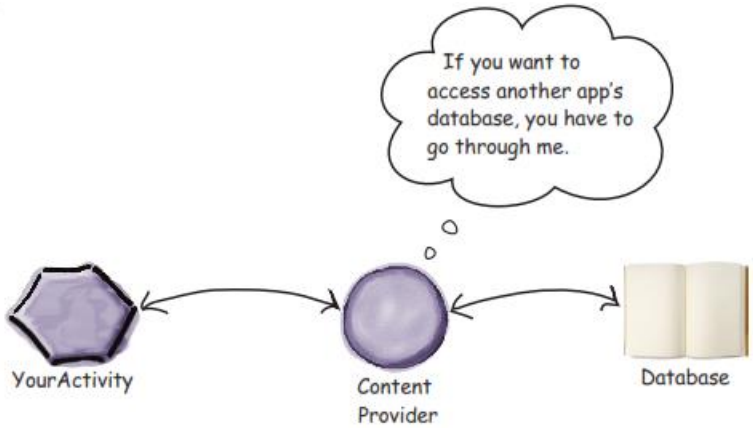


Content Providers

You can't access another app's data by interrogating its database, Instead, you use a content provider, which is an interface that allows apps to share data in a controlled way. It allows you to perform queries to read the data, insert new records, and update or delete existing records.
A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the *ContentResolver* class. The data may be stored in the file system, the database or somewhere else entirely.

A content provider is implemented as a subclass of **ContentProvider** class.

```
public class MyContentProvider extends  ContentProvider {
  public void onCreate(){}
}
```



S.No	Components & Description
1	Fragments
	Represents a portion of user interface in an Activity.
2	Views
	UI elements that are drawn on-screen including buttons, lists forms etc.
3	Layouts
	View hierarchies that control screen format and appearance of the views.
4	Intents
	Messages wiring components together.
5	Resources
	External elements, such as strings, constants and drawable pictures.
6	Manifest
	Configuration file for the application.

5.Intends:

It is a powerful inter-application message-passing framework. They are extensively used throughout Android. Intents can be used to start and stop Activities and Services, to broadcast messages system-wide or to an explicit Activity, Service or Broadcast Receiver or to request action be performed on a particular piece of data.

6. Widgets

These are the small visual application components that you can find on the home screen of the devices. They are a special variation of Broadcast Receivers that allow us to create dynamic, interactive application components for users to embed on their Home Screen.

7. Notifications

Notifications are the application alerts that are used to draw the user's attention to some particular app event without stealing focus or interrupting the current activity of the user. Eg Wifi availability, facebook notifications etc. They are generally used to grab user's attention when the application is not visible or active, particularly from within a Service or Broadcast Receiver. Examples: E-mail popups, Messenger popups, etc.

Android Libraries

A set of standard Java development libraries (providing support for such general purpose tasks as string handling, networking and file manipulation), the Android development environment also includes the Android Libraries.

- **android.app** – Provides access to the application model and is the cornerstone of all Android applications.
- **android.content** – Facilitates content access, publishing and messaging between applications and application components.
- **android.database** – Used to access data published by content providers and includes SQLite database management classes.
- **android.graphics** – A low-level 2D graphics drawing API including colors, points, filters, rectangles and canvases.
- **android.hardware** – Presents an API providing access to hardware such as the accelerometer and light sensor.
- **android.opengl** – A Java interface to the OpenGL ES 3D graphics rendering API.
- **android.os** – Provides applications with access to standard operating system services including messages, system services and interprocess communication.
- **android.media** – Provides classes to enable playback of audio and video.
- **android.net** – A set of APIs providing access to the network stack. Includes *android.net.wifi*, which provides access to the device's wireless stack.
- **android.print** – Includes a set of classes that enable content to be sent to configured printers from within Android applications.
- **android.provider** – A set of convenience classes that provide access to standard Android content provider databases such as those maintained by the calendar and contact applications.
- **android.text** – Used to render and manipulate text on a device display.
- **android.util** – A set of utility classes for performing tasks such as string and number conversion, XML handling and date and time manipulation.
- **android.view** – The fundamental building blocks of application user interfaces.
- **android.widget** - A rich collection of pre-built user interface components such as buttons, labels, list views, layout managers, radio buttons *etc.*
- **android.webkit** – A set of classes intended to allow web-browsing capabilities to be built into applications.

Android Activity Lifecycle

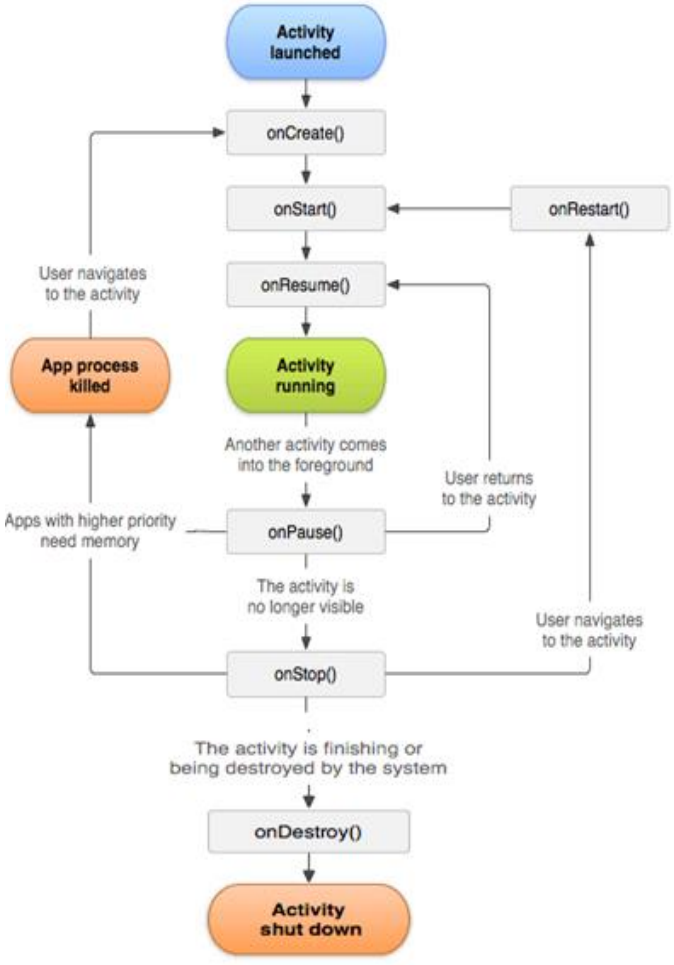
In android, **Activity** represents a single screen with a user interface (UI) of an application and it will acts an entry point for users to interact with an app.

Generally, the android apps will contain multiple screens and each screen of our application will be an extension of Activity class. By using activities, we can place all our android application UI components in a single screen.

From the multiple activities in android app, one activity can be marked as a **main activity** and that is the first screen to appear when we launch the application. In android app each activity can start another activity to perform different actions based on our requirements.

For example, a contacts app which is having multiple activities, in that the main activity screen will show a list of contacts and from the main activity screen we can launch other activities that provide screens to perform tasks like add a new contact and search for the contacts. All these activities in the contact app are loosely bound to other activities but will work together to provide a better user experience.

Generally, in android there is a minimal dependency between the activities in an app. To use activities in application we need to register those activities information in our app's manifest file (**AndroidMainfest.xml**) and need to manage activity life cycle properly.



To use activities in our application we need to define an activities with required attributes in manifest file (**AndroidMainfest.xml**) like as shown below

```
<?xml version="1.0" encoding="utf-8"?>
<manifest …..>
<application …..>
<activity android:name=".MainActivity" >
        …….

        …….

</activity>

    …….

</application>
</manifest>
```

The activity attribute **android:name** will represent the name of class and we can also add multiple attributes like icon, label, theme, permissions, etc. to an activity element based on our requirements.

In android application, activities can be implemented as a subclass of **Activity** class like as shown below.

```
public class MainActivity extends Activity {

}
```

This is how we can activities in android application based on our requirements.

Generally, the activities in our android application will go through a different stages in their life cycle. In android, **Activity** class have 7 callback methods like onCreate(), onStart(), onPause(), onRestart(), onResume(), onStop() and onDestroy() to describe how the activity will behave at different stages.

By using activity call-back methods we can define how our activity can behave when the user enter or leaves our application.

Android Activity Lifecycle Callback Methods

In android, an activity goes through a series of states during its lifetime. By using callback methods we can get the activity transitions between the states.

Android system initiates its program within an **Activity** starting with a call on onCreate() callback method. There is a sequence of callback methods that start up an activity and a sequence of callback methods that tear down an activity.

This section will give you detailed information about callback methods to handle activity transitions between states during the lifecycle.

onCreate()

onCreate() gets called when the activity is first created. This is the first callback method.. During the activity creation, activity entered into a **Created** state.

If we have an application start-up logic that needs to perform only once during the life cycle of an activity, then we can write that logic in onCreate() method.

Following is the example of defining a onCreate() method in android activity.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
}
```

Once onCreate() method execution is finished, the activity will enter into Started state and the system calls the onStart() method.

onStart()
The onStart() callback method will invoke when an activity entered into **Started** State by completing onCreate() method. The onStart() method will make an activity visible to the user and this method execution will finish very quickly.

Following is the example of defining a onStart() method in android activity.

// @**Override** is a Java annotation. It tells the compiler that the following method **overrides** a method of its superclass. For instance, say you implement a Person class.

```
@Override
protected void onStart()
{
super.onStart()
}
```

After completion of onStart() method execution, the activity enters into **Resumed** state and system invoke the onResume() method.

onResume()
When an activity entered into **Resumed** state, the system invokes onResume() call back method. In this state activity start interacting with the user that means user can see the functionality and designing part of an application on the single screen.

Mostly the core functionality of an app is implemented in onResume() method.

The app will stay in this **Resumed** state until an another activity happens to take focus away from the app like getting a phone call or screen turned off, etc.

In case if any interruption events happen in **Resumed** state, the activity will enter into **Paused** state and the system will invoke onPause() method.

After an activity returned from **Paused** state to **Resumed** state, the system again will call onResume() method due to this we need to implement onResume() method to initialize the components that we release during onPause() method

Following is the example of defining a onResume() method in android activity.

```
@Override
public void onResume() {
super.onResume();
if (mCamera == null) {
initializeCamera();
   }
}
```

onPause()
Whenever the user leaves an activity or the current activity is being Paused then the system invokes onPause() method. The onPause() method is used to pause operations like stop playing

the music when the activity is in a paused state or pass an activity while switching from one app to another app because every time only one app can be focused.

Following is the example of defining a onPause() method in android activity.

```
@Override
public void onPause()
{
super.onPause();
if (mCamera != null) {
mCamera.release();
mCamera = null;
   }
}
```

After completion of onPause() method execution, the next method is either onStop() or onResume() depending on what happens after an activity entered into a **Paused** state.

onStop()
The system will invoke onStop() callback method when an activity no longer visible to the user, the activity will enter into Stopped state. This happens due to current activity entered into **Resumed** state or newly launched activity covers complete screen or it's been destroyed.

The onStop() method is useful to release all the app resources which are no longer needed to the user.

Following is the example of defining a onStop() method in android activity.

```
@Override
protected void onStop()
{
super.onStop();
}
```

The next callback method which raised by the system is either onRestart(), in case if the activity coming back to interact with the user or onDestroy(), in case if the activity finished running.
onRestart()
The system will invoke onRestart() method when an activity restarting itself after stopping it. The onRestart() method will restore the state of activity from the time that is being stopped.

The onRestart() callback method in android activity will always be followed by onStart() method.
onDestroy()
The system will invoke onDestroy() method before an activity is destroyed and this is the final callback method received by the android activity.

The system will invoke this onDestory() callback method either the activity is finishing or system destroying the activity to save space.

Following is the example of defining a onDestroy() method in android activity.

```
@Override
public void onDestroy()
{
super.onDestroy();
}
```

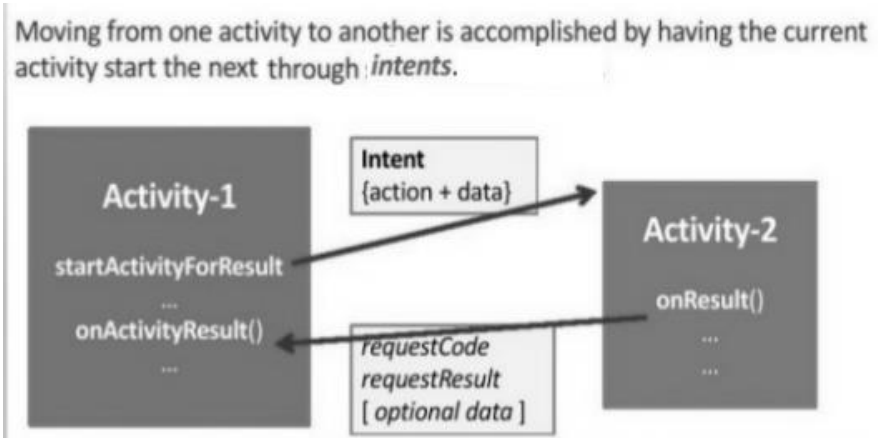
The onDestroy() method will release all the resources which are not released by previous callback onStop() method.

INTENTS

Intents are asynchronous messages which allow Android components to request functionality from other components of the Android system.
For example an Activity can send an Intents to the Android system which starts another Activity. An Intent can also contain data. This data can be used by the receiving component.

Intent is a messaging object for run-time binding between components in the same or different applications. The components are Activities, Services, Broadcast receivers.

- Call a component from another component.
- Possible to pass data between components.



In android, **Intent** is a messaging object which is used to request an action from another app or same app through components such as activities, services, broadcast receivers, and content providers.

Generally, in android, **Intents** will help us to maintain the communication between app components from the same application as well as with the components of other applications.

In android, Intents are the objects of **android.content.Intent** types and intents are mainly useful to perform the following things.

Component	Description
Starting an Activity	By sending an Intent object to startActivity() method we can start a new Activity or existing Activity to perform required things.
Starting a Service	By sending an Intent object to startService() method we can start a new Service or send required instructions to an existing Service.
Delivering a Broadcast	By sending an **Intent** object to sendBroadcast() method we can deliver our messages to other app broadcast receivers.

Component Name

It defines the name of the component (.class file) to start and by using the component name android system will deliver intent to the specific app component defined by the component name. In case if we didn't define component name then the android system will decide which component should receive intent based on other intent information such as action, data, etc.

In android, we can specify the component name for intent by using a fully qualified class name of the target component and package name, for example, edu.annauniv.resultActivity. We can set the component name by using setComponent(), setClass(), setClassName() or by using the Intent constructor.

Action

It defines the name of the action to be performed to start an activity. The following are some of the common actions to start an activity.

Action	Description
ACTION_VIEW	We can use this action in intent with startActivity(), when we have information that activity can show to the user.
ACTION_SEND	We can use this action in intent with startActivity(), when we have some data that the user can share through another app such as an email app, social sharing app.

We can specify the action name of intent by using setAction() or with an Intent constructor.

Data

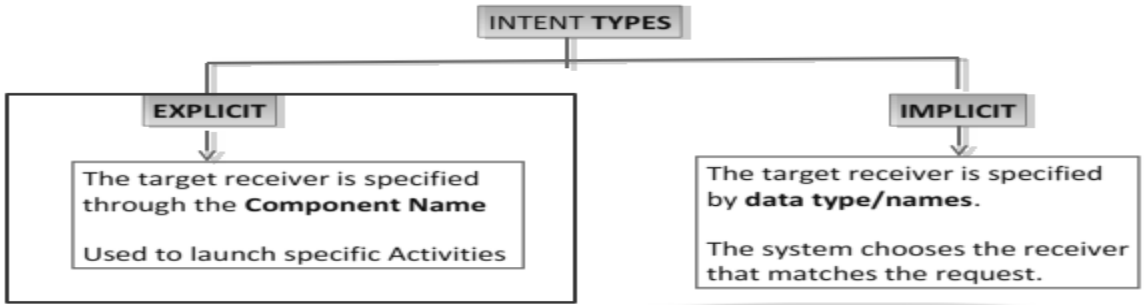
It specifies a type of data to an intent filter. When we create an intent, it's important to specify the type of data (MIME type) in addition to its URI. By specifying a MIME type of data, it helps the android system to decide which is the best component to receive our intent.

Category

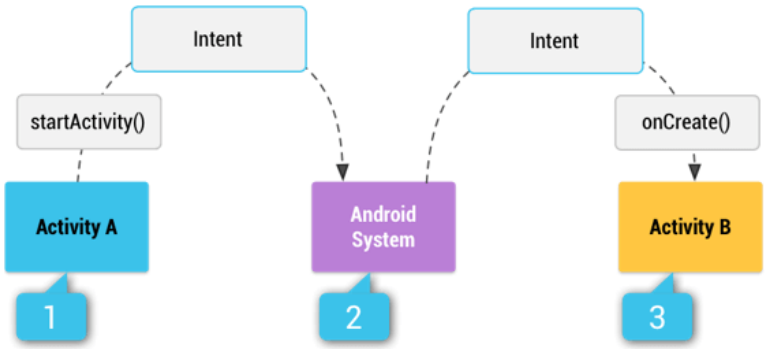
Generally, the android category is optional for intents and it specifies the additional information about the type of component that should handle an intent.

We can specify a category for intent by using addCategory().

The above properties (Component Name, Action, Data, and Category) will represent the characteristics of an intent. By using these properties, the android system will easily decide which app component to start.



Implicit Intent



If you observe the above image **Activity A** creates an intent with the required action and sends it to an android system using the **startActivity()** method. The android system will search for an intent filter that matches the intent in all apps. Whenever the match found the system starts matching activity (**Activity B**) by invoking the **onCreate()** method.

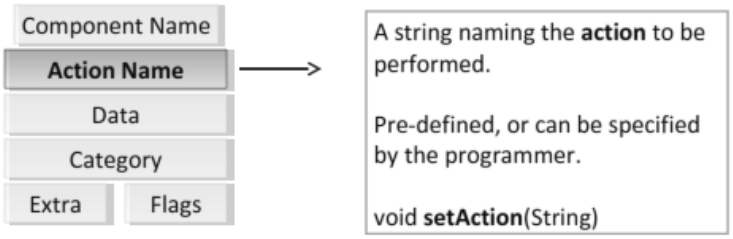
When an Intent is launched, Android checks out which activities might answer to the Intent …
If at least one is found, then that activity is started!
The Binding does not occur at compile time, nor at install time, but at run-time.

Implicit Intents do not directly specify the Android components which should be called.
For example the following tells the Android system to view a webpage.

```
Intent i = new Intent(Intent.ACTION_VIEW,
           Uri.parse("http://www.google.com"));
```

or telling the Android system to open the camera:

```
Intent i = new Intent(android.provider.MediaStore.
    ACTION_IMAGE_CAPTURE);
```



Intent Components
(Predefined Actions)

Action Name	Description
ACTION_EDIT	Display data to edit
ACTION_MAIN	Start as a main entry point, does not expect to receive data.
ACTION_PICK	Pick an item from the data, returning what was selected.
ACTION_VIEW	Display the data to the user
ACTION_SEARCH	Perform a search

Special actions

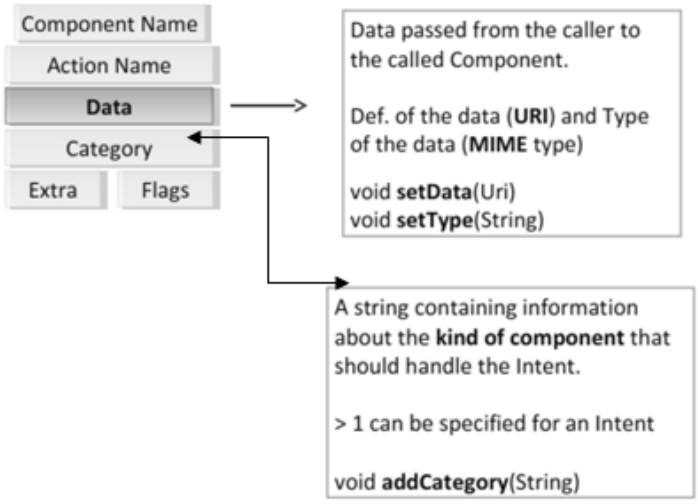
Action Name	Description
ACTION_IMAGE_CAPTION	Open the camera and receive a photo
ACTION_VIDEO_CAPTION	Open the camera and receive a video
ACTION_DIAL	Open the phone app and dial a phone number
ACTION_SENDTO	Send an email (email data contained in the extra)
ACTION_SETTINGS	Open the system setting
ACTION_WIRELESS_SETTINGS	Open the system setting of the wireless interfaces
ACTION_DISPLAY_SETTINGS	Open the system setting of the display

1. Example of Implicit Intent that initiates a web search.

```
public void doSearch(String query) {
Intent intent =new Intent(Intent.ACTION_SEARCH);
Intent.putExtra(SearchManager.QUERY,query);
if (intent.resolveActivity(getPackageManager()) !=null)
startActivity(intent)
}
```

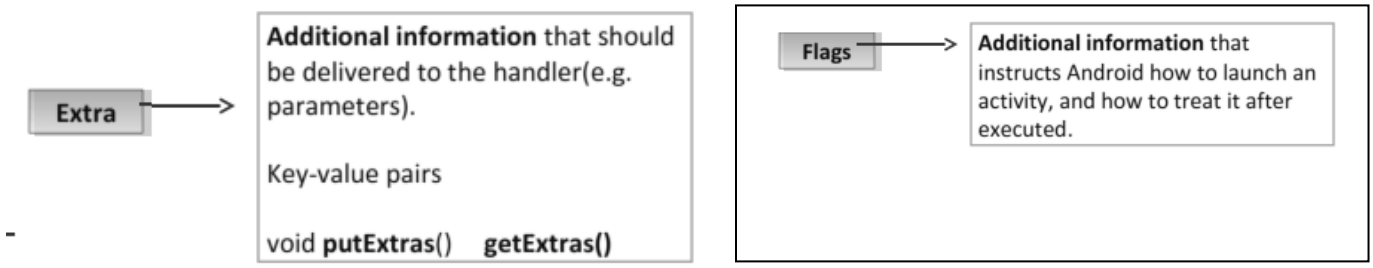
2. Example of Implicit Intent that plays a music file.

```
public void playMedia(Uri file) {
Intent intent =new Intent(Intent.ACTION_VIEW);
if (intent.resolveActivity(getPackageManager()) !=null)
startActivity(intent)
}
```



Category: string describing the **kind of component** that should handle the intent.

Category Name	Description
CATEGORY_HOME	The activity displays the HOME screen.
CATEGORY_LAUNCHER	The activity is listed in the top-level application launcher, and can be displayed.
CATEGORY_PREFERENCE	The activity is a preference panel.
CATEGORY_BROWSABLE	The activity can be invoked by the browser to display data referenced by a link.



-

Explicit Intent: (Programmer decide)

Explicit Intents explicitly defines the component which should be called by the Android system, by using the Java class as identifier. The following shows an explicit Intent.

```
Intent i = new Intent(this, ActivityTwo.class);
```

Explicit Intents are typically used within one application as the classes in an application are controlled by the application developer.

Specify the name of the Activity that will handle the intent.

```
Intent intent=new Intent();
ComponentName component=new ComponentName(this,SecondActivity.class);
intent.setComponent(component);
startActivity(intent);
```

Activities can return results (e.g. data)

Sender side: invoke the startActivityForResult()

onActivityResult(int requestCode, int resultCode, Intent data)
startActivityForResult(Intent intent, int requestCode);

Intent intent = new Intent(this, SecondActivity.class);
startActivityForResult(intent, CHOOSE_ACTIVITY_CODE);
…

public void onActivityResult(int requestCode, int resultCode, Intent data) {
// Invoked when SecondActivity completes its operations …
}

Activities can return results (e.g. data)
Receiver side: invoke the setResult()

void setResult(int resultCode, Intent data)

Intent intent=new Intent();
setResult(RESULT_OK, intent);
intent.putExtra(”result", resultValue);
finish();

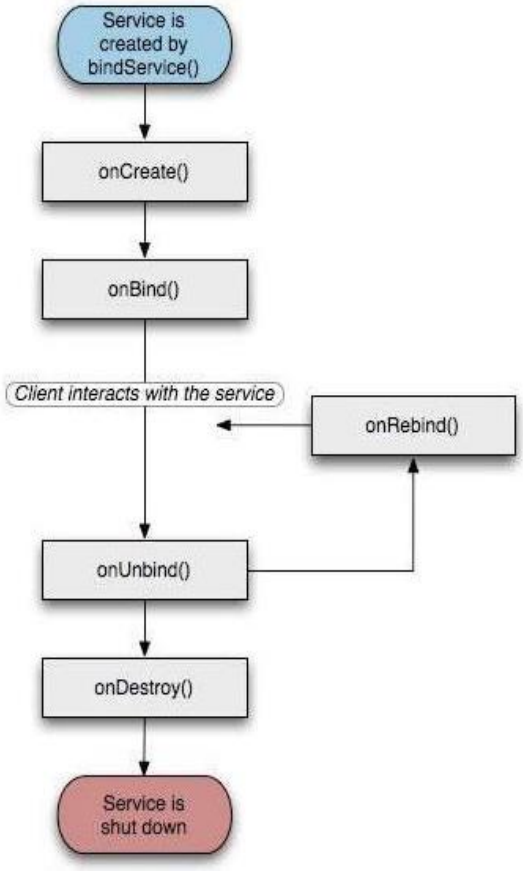
The result is delivered to the caller component only after invoking the finish() method.

SERVICES

A service is a component that runs in the background to perform long-running operations without needing to interact with the user and it works even if application is destroyed. A service can essentially take two states −

Sr.No.	State & Description
1	Started A service is started when an application component, such as an activity, starts it by calling startService(). Once started, a service can run in the background indefinitely, even if the component that started it is destroyed.
2	Bound A service is bound when an application component binds to it by calling bindService(). A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with interprocess communication (IPC).

A service has life cycle callback methods that you can implement to monitor changes in the service's state and you can perform work at the appropriate stage. The following diagram on the left shows the life cycle when the service is created with startService() and the diagram on the right shows the life cycle when the service is created with bindService(): (image courtesy : android.com)



onStartCommand()

The system calls this method when another component, such as an activity, requests that the service be started, by calling *startService()*. If you implement this method, it is your responsibility to stop the service when its work is done, by calling *stopSelf()* or *stopService()* methods.

onBind()

The system calls this method when another component wants to bind with the service by calling *bindService()*. If you implement this method, you must provide an interface that clients use to communicate with the service, by returning an *IBinder* object.

onUnbind()

The system calls this method when all clients have disconnected from a particular interface published by the service.

onRebind()

The system calls this method when new clients have connected to the service, after it had previously been notified that all had disconnected in its *onUnbind(Intent)*.

onCreate()

The system calls this method when the service is first created using *onStartCommand()* or *onBind()*. This call is required to perform one-time set-up.

onDestroy()

The system calls this method when the service is no longer used and is being destroyed. Your service should implement this to clean up any resources such as threads, registered listeners, receivers, etc.

```
package com.AndroidTut;

import android.app.Service;
import android.os.IBinder;
import android.content.Intent;
import android.os.Bundle;

public class HelloService extends Service {

  /** indicates how to behave if the service is killed */
  int mStartMode;

  /** interface for clients that bind */
  IBinder mBinder;

  /** indicates whether onRebind should be used */
  boolean mAllowRebind;

  /** Called when the service is being created. */
  @Override
  public void onCreate() {

  }

  /** The service is starting, due to a call to startService() */
  @Override
  public int onStartCommand(Intent intent, int flags, int startId) {
    return mStartMode;
  }

  /** A client is binding to the service with bindService() */
  @Override
  public IBinder onBind(Intent intent) {
    return mBinder;
  }

  /** Called when all clients have unbound with unbindService() */
  @Override
  public boolean onUnbind(Intent intent) {
    return mAllowRebind;
  }

  /** Called when a client is binding to the service with bindService()*/
  @Override
  public void onRebind(Intent intent) {

  }
```

```
  /** Called when The service is no longer used and is being destroyed */
  @Override
  public void onDestroy() {

  }
}
```

MainActivity.java

```
package com.example.AndroidTut.myapplication;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

import android.os.Bundle;
import android.app.Activity;
import android.util.Log;
import android.view.View;

public class MainActivity extends Activity {
  String msg = "Android : ";

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Log.d(msg, "The onCreate() event");
  }

  public void startService(View view) {
    startService(new Intent(getBaseContext(), MyService.class));
  }

  // Method to stop the service
  public void stopService(View view) {
    stopService(new Intent(getBaseContext(), MyService.class));
  }
}
```

Following is the content of MyService.java. This file can have implementation of one or more methods associated with Service based on requirements. For now we are going to implement only two methods onStartCommand() and onDestroy() −

```
package com.example.AndroidTut.myapplication;

import android.app.Service;
import android.content.Intent;
```

```
import android.os.IBinder;
import android.support.annotation.Nullable;
import android.widget.Toast;

public class MyService extends Service {
  @Nullable
  @Override
  public IBinder onBind(Intent intent) {
    return null;
  }

  @Override
  public int onStartCommand(Intent intent, int flags, int startId) {
    // Let it continue running until it is stopped.
    Toast.makeText(this, "Service Started", Toast.LENGTH_LONG).show();
    return START_STICKY;
  }

  @Override
  public void onDestroy() {
    super.onDestroy();
    Toast.makeText(this, "Service Destroyed", Toast.LENGTH_LONG).show();
  }
}
```

DATA STORAGE IN ANDROID

1. Shared Preferences:-

Android shared preference is used to store and retrieve primitive information. In android, string, integer, long, number etc. are considered as primitive data type.

Android Shared preferences are used to store data in key and value pair so that we can retrieve the value on the basis of key.

It is widely used to get information from user such as in settings.
Typical usage of SharedPreferences is for saving application such as username and password, auto login flag, remember-user flag etc.

The shared preferences information is stored in an XML file on the device Lightweight usage, such as saving application settings.

using getpreferences()
You used the SharedPreferences object by supplying it with a name, like this:

```
prefs=getSharedPreferences(prefName,MODE_PRIVATE);
SharedPreferences.Editoreditor=prefs.edit();
```

The information saved inside the SharedPreferences object is visible to all the activities (page)

within the same application. (axisbank app)

The MODE_PRIVATE constant indicates that the shared preference file can only be opened by the application that created it.

The Editor class allows you to save key/value pairs to the preferences file by exposing methods such as the following:

- putString()
- putBoolean()
- putLong()
- putInt()
- putFloat()

When you are done saving the values, call the commit() method to save the changes:

```
editor.putFloat(FONT_SIZE_KEY,editText.getTextSize());
editor.putString(TEXT_VALUE_KEY,editText.getText().toString());
//---savesthevalues--editor.commit();
```

Commit – stores as permanently.

When the activity is loaded, you first obtain the SharedPreferences object and then retrieve all the values saved earlier:

```
//---loadtheSharedPreferencesobject—
SharedPreferences  prefs=getSharedPreferences(prefName,MODE_PRIVATE);
```

```
//---settheTextViewfontsize to the previously saved values –
floatfontSize=prefs.getFloat(FONT_SIZE_KEY,12);
editText.setText(prefs.getString(TEXT_VALUE_KEY,
“”));
```

The shared preferences file is saved as an XML file in the data/<package_name>/shared_prefs folder.

using getpreferences()

prefs.xml

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
   <PreferenceCategory
      android:summary="Username and password information"
      android:title="Login information" >
      <EditTextPreference
         android:key="username"
         android:summary="Please enter your login username"
         android:title="Username" />
      <EditTextPreference
         android:key="password"
         android:summary="Enter your password"
         android:title="Password" />
   </PreferenceCategory>

   <PreferenceCategory
```

```
    android:summary="Username and password information"
    android:title="Settings" >
    <CheckBoxPreference
      android:key="checkBox"
      android:summary="On/Off"
      android:title="Keep me logged in" />

    <ListPreference
      android:entries="@array/listOptions"
      android:entryValues="@array/listValues"
      android:key="listpref"
      android:summary="List preference example"
      android:title="List preference" />
  </PreferenceCategory>
</PreferenceScreen>
```

SharedPreferences can be associated with the entire application, or to a specific activity.
Use the getSharedPrefernces() method to get access to the preferences.
If the preferences XML file exist, it is opened, otherwise it is created.

To Control access permission to the file:
MODE_PRIVATE: private only to the application
MODE_WORLD_READABLE: all application can read XML file
MODE_WORLD_WRITABLE: all application can write XML file

To retrieve shared preferences data:

```
String   username = prefsEditor.getString("username"." ");
String   password = prefsEditor.getString("password"." ");
```

2. Internal Storage:-

We are able to save or read data from the device internal memory. Android can save files directly to the device internal storage. FileInputStream and FileOutputStream classes are used to read and write data into the file. These files are private to the application and will be removed if you uninstall the application.

We can create a file using openFileOutput with parameter as file name and the operating mode.

Similarly, we can open the file using openFileOutput() passing the parameter as the filename with extension.

- Files are used to store large amount of data
- Use I/O interfaces provided by java.io.* libraries to read/write files.
- Only local files can be accessed.

File Operation(Read)
Use context.openFileInput(string name) to open a private input file stream related to a program.
Throw FileNotFoundException when file does not exist.
Syntax: fileinputStream.in=this.openfileinput("xyz.txt")

File Operation (Write)
Use context.openFileOutput(string name,int mode) to open a private output file stream related to a program. The file will be created if it does not exist.
Output stream can be opened in append mode, which means new data will be appended to end of the file.

String mystring="Hello World"

Syntax:-
FileOutputStream outfile = this.openFileOutput("myfile.txt",MODE_APPEND)
//Open and Write in"myfile.txt",using append mode.
Outfile.write(mystring.getBytes());
Outfile.close();//close output stream

Activity class
Let's write the code to write and read data from the internal storage.
File: MainActivity.java

```
package example.javatpoint.com.internalstorage;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;

public class MainActivity extends AppCompatActivity {
   EditText  editTextFileName,editTextData;
   Button saveButton,readButton;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      editTextFileName=findViewById(R.id.editText1);
      editTextData=findViewById(R.id.editText2);
      saveButton=findViewById(R.id.button1);
      readButton=findViewById(R.id.button2);

     //Performing Action on Write Button
      saveButton.setOnClickListener(new View.OnClickListener(){
           @Override
         public void onClick(View arg0) {
            String filename=editTextFileName.getText().toString();
            String data=editTextData.getText().toString();

            FileOutputStream fos;
            try {
               fos = openFileOutput(filename, Context.MODE_PRIVATE);
```

```
            //default mode is PRIVATE, can be APPEND etc.
            fos.write(data.getBytes());
            fos.close();

            Toast.makeText(getApplicationContext(),filename + " saved",
                  Toast.LENGTH_LONG).show();

          } catch (FileNotFoundException e) {e.printStackTrace();}
          catch (IOException e) {e.printStackTrace();}
        }
      });

    //Performing Action on Read Button
    readButton.setOnClickListener(new View.OnClickListener(){

      @Override
      public void onClick(View arg0) {
        String filename=editTextFileName.getText().toString();
        StringBuffer stringBuffer = new StringBuffer();
        try {
          BufferedReader inputReader = new BufferedReader(new InputStreamReader(
                openFileInput(filename)));
          String inputString;
          //Reading data line by line and storing it into the stringbuffer
          while ((inputString = inputReader.readLine()) != null) {
            stringBuffer.append(inputString + "\n");
          }

        } catch (IOException e) {
          e.printStackTrace();
        }
        //Displaying data on the toast

Toast.makeText(getApplicationContext(),stringBuffer.toString(),Toast.LENGTH_LONG).show(
);

      }

    });
  }
}
```

3. External Storage:-

Store public data on the shared external storage. (SD Card)

Every Android-compatible device supports a shared “external storage” so that you can save files. The Removable storage media has larger capacity, as well as the capability to share the files easily with other users.

File saved to the external storage are world readable and can be modified by the user and transfer files on computer.
These files are private to the application and will be removed when the application is uninstalled.
Must check whether external storage is available first by calling getExternalStorageState()

```
//Performing action on save button
saveBtn.setOnClickListener(newView.OnClickListener(){
public void onClick(View v){
String str=textBox.getText().toString();


try
{
//---SD Card Storage---
File sdCard = Environment.getExternalStorageDirectory();
File directory = new File (sdCard.getAbsolutePath() + "/MyFiles");
directory.mkdirs();
File file = new File(directory, "textfile.txt");
FileOutputStream fOut = new FileOutputStream(file);
OutputStreamWriter  osw=new OutputStreamWriter(fOut);
//---write the string to the file---
osw.write(str);
osw.flush();
osw.close();

//---display file saved message---
Toast.makeText(getBaseContext(), "Filesavedsuccessfully!",
Toast.LENGTH_SHORT).show();
//---clears the Edit Text---
textBox.setText("");
}

catch(IOExceptionioe)
{
ioe.printStackTrace();
}
}
});
```

To load the file from the external storage, modify the onClick() method for the Load button:

```
loadBtn.setOnClickListener(newView.OnClickListener(){
public void onClick(View v){
try
{
//---SD Storage--File
sdCard = Environment.getExternalStorageDirectory();
```

```
File directory = new File (sdCard.getAbsolutePath() + "/MyFiles");
File file = new File(directory, "textfile.txt");
FileInputStream fIn = new FileInputStream(file);
InputStreamReader isr = new InputStreamReader(fIn);
char[]inputBuffer=newchar[READ_BLOCK_SIZE];
Strings="";
intcharRead;


while((charRead=isr.read(inputBuffer))>0)
{
//---convertthecharstoaString--StringreadString= String.copyValueOf(inputBuffer,0,charRead);
s+=readString;
inputBuffer=newchar[READ_BLOCK_SIZE];
}
textBox.setText(s);
Toast.makeText(getBaseContext(), "Fileloadedsuccessfully!",
Toast.LENGTH_SHORT).show();
}
catch(IOExceptionioe){
ioe.printStackTrace();
}
}
});
```

SQLite Databases:-Store structured data in a private database.
android.database.sqlite Contains the SQLite database management classes that an application would use to manage its own private database.
android.database.sqlite.SQLiteDatabase Contains the methods for: creating, opening, closing, inserting, updating, deleting and quering an SQLite database.

android.database.sqlite – Classes

SQLiteCloseable - An object created from a SQLiteDatabase that can be closed.
SQLiteCursor - A Cursor implementation that exposes results from a query on a SQLiteDatabase.
SQLiteDatabase - Exposes methods to manage a SQLite database.
SQLiteOpenHelper - A helper class to manage database creation and version management.
SQLiteProgram - A base class for compiled SQLite programs.
SQLiteQuery - A SQLite program that represents a query that reads the resulting rows into a CursorWindow.
SQLiteQueryBuilder - a convenience class that helps build SQL queries to be sent to SQLiteDatabase objects.
SQLiteStatement - A pre-compiled statement against a SQLiteDatabase that can be reused.
Open Or CreateDatabase
This method will open an existing database or create one in the application data area

```
import android.database.sqlite.SQLiteDatabase;
   SQLiteDatabase  myDatabase;
```

```
    myDatabase = openOrCreateDatabase ("my_sqlite_database.db" ,

SQLiteDatabase.CREATE_IF_NECESSARY , null);
```

Creating Tables

Create a static string containing the SQLite CREATE statement, use the execSQL( ) method to execute it.

```
String  createAuthor = "CREAT TABLE   authors (
                            id   INTEGER PRIMARY KEY AUTOINCREMENT, fname   TEXT,
lname  TEXT);
myDatabase.execSQL(createAuthor);
```

insert( )

```
long  insert(String table, String nullColumnHack, ContentValues values)

import  android.content.ContentValues;
ContentValues  values = new  ContentValues( );
values.put("firstname" , "J.K.");
values.put("lastname" , "Rowling");
long newAuthorID =  myDatabase.insert("tbl_authors" , "" , values);
int  delete(String table, String whereClause, String[] whereArgs)
public void deleteBook(Integer  bookId)  {
    myDatabase.delete("tbl_books" , "id=?" ,
         new String[ ]   { bookId.toString( )  } ) ;
 }
```

update()

```
int  update(String table, ContentValues values, String whereClause, String[ ] whereArgs)
public void updateBookTitle(Integer bookId, String newTitle) {
  ContentValues  values = new ContentValues();
     values.put("title" , newTitle);
```

Network Connection:-Store data on the web with your own network server.

ANDROID PACKAGES

A package is a namespace that combines a set of relevant classes and interfaces. Conceptually one can consider packages as being similar to various folders on your computer.

A namespace is a declarative region that provides a scope to the identifiers (the names of types, functions, variables, etc) inside it. Namespaces are used to organize code into logical groups and to prevent name collisions that can occur especially when your code base includes multiple libraries

Android application package. Android Package (APK) is the package file format used by the Android operating system for distribution and installation of mobile apps and middleware.

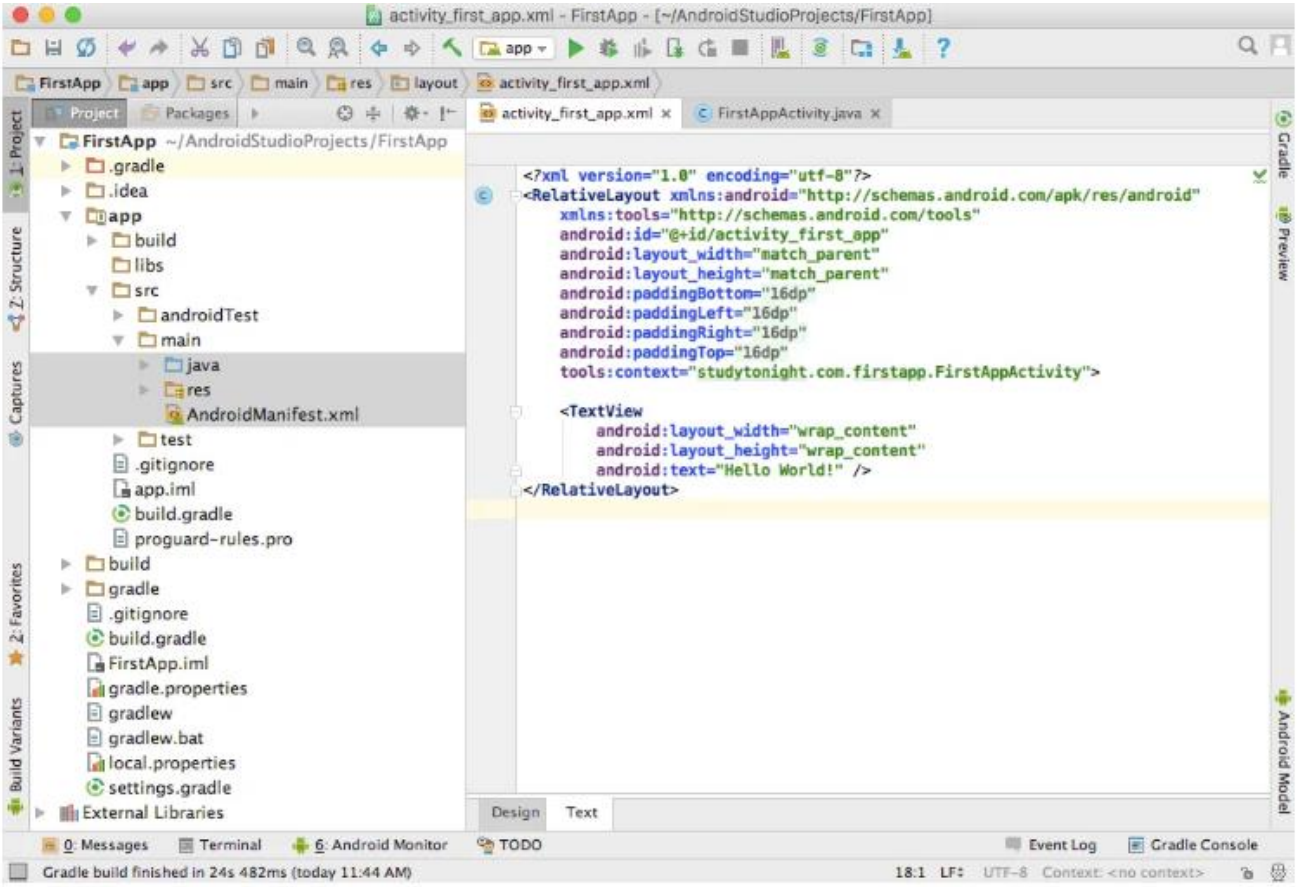
A package is basically the directory (folder) in which the source code resides. Normally, this is a directory structure that uniquely distinguishes the android application; such as com. example. app. Then the developer can build packages within the application package that divides the code;

Package Structure in Android Studio

In Android Studio, the modern day Android App package structure has changed a little bit, but the change is very subtle.

Here is a list of folders created, when you create an Android App project in Android Studio and see it in Project View Mode:

NOTE: The packages which are written in bold font are important packages.



src folder

The src folder holds two most important folders on any Android project, namely, androidTest and main.

The androidTest package is created to hold Test cases for testing the application code and running.

While the main folder, has 2 folders and 1 file. They are:

- java directory
  This folder contains .java (JAVA) files. Here, you can create interface(s), activity(s), fragment(s) or adapter(s) for your application. This folder contains java code only. You can create separate packages for each of these and create classes inside them to give your application project a well defined structure.

- AndroidManifest.xml
  This file is a mandatory file for any android application. In this file we provide information about all the application's Activities, Services, Broadcast Receivers etc and all the permissions like Internet, Contacts, Camera etc that our application will require when it is installed on any device. Just keep in mind that this file is the heart of any Android Application.

- res directory

This folder contains all the resources like icons, images etc related to our application project and it contains the following sub-folders:

- drawable
  This folder contains xml, png, jpg and jpeg files. In this folder you can store images which are used in your app and other .xml files which are used for many purposes like creating button background, or shadow effect etc.
- layout
  This folder contain only the layout .xml files for different screens and parts of your application.
- values
  This folder contains default files
  like strings.xml, dimens.xml, colors.xml, styles.xml.
  - In strings.xml, you can specify all the string constants like title of screen or any other tag which are constantly used in your app.
  - In dimens.xml, you can create different .xml files to define dimensions as per resolution of screen and dimension of it and give any dimensions for padding, height, width, margin in this file.
  - In color.xml you can mention the list of colors using their hashcodes, used in your application. If this is not created by default, you can create this yourself.
  - In styles.xml you can define different readymade styles to use them directly anywhere in your Android App.

Build Folder

This folder contain R.java file, which is an auto generated file. This file indexes all the resources of the android application project like layout xml, strings xml etc and it is auto generated.

In build folder, you will have application's debug and release .apk files, created when you build your project.

libs folder

If you want to use any external library, all you have to do is copy-paste the .jar file into the libs folder and then you can directly use it in you .java code files.

gradle folder

In this folder there are files related to gradle which can be modified to alter the project building process. For example: If you wish to run all the available test cases before building the .apk file, you can do so by mentioning it in the gradle file.

To Import Module

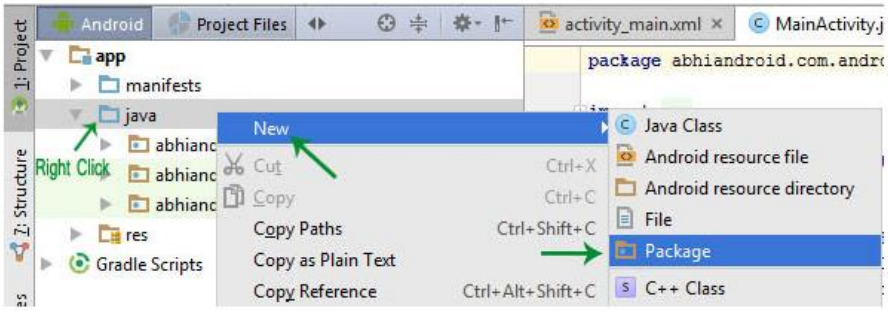
1. Go to File > New > Import Module...
2. Select the source directory of the Module you want to import and click Finish.
3. Open Project Structure and open Module Settings for your project.
4. Open the Dependencies tab.
5. Click the (+) icon and select Module Dependency. Select the module and click Ok.
6. Open your build.gradle file and check that the module is now listed under dependencies.(implementation project(path: ':ViewPagerIndicator')

Steps to import Module in Android Studio 3.4 and higher (See attached image).

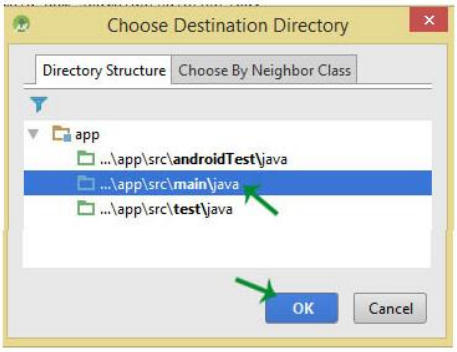
1. Go to File > New > Import Module...
2. Select the source directory of the Module you want to import and click Finish.
3. Open Project Structure Dialog (You can open the PSD by selecting File > Project Structure) and from the left panel click on Dependencies. (PSD file represent image file types that are created with the most commonly used professional image editing program)
4. Select the module from the Module(Middle) section In which you want to add module dependency.
5. Click the (+) icon from the Declared Dependencies section and click Module Dependency.
6. Select the module and click Ok.
7. Open your build.gradle file and check that the module is now listed under dependencies.(implementation project(path: ':ViewPagerIndicator')

<u>To Create/Add new Package inside src folder in Android Studio:</u>

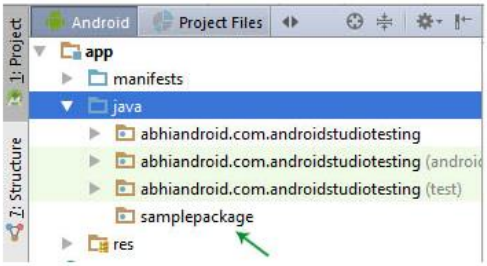
The following are the steps to add new Package name inside src folder:
Step 1: Open Android Studio and Navigate to any view(Android or Project)
Step 2: In Android View you will have two folders: app and Gradle Scripts
Step 3: Open App folder then open Java folder. Right click on Java folder and select New > Package.



Step 4: Choose directory destination which main\java and click OK.



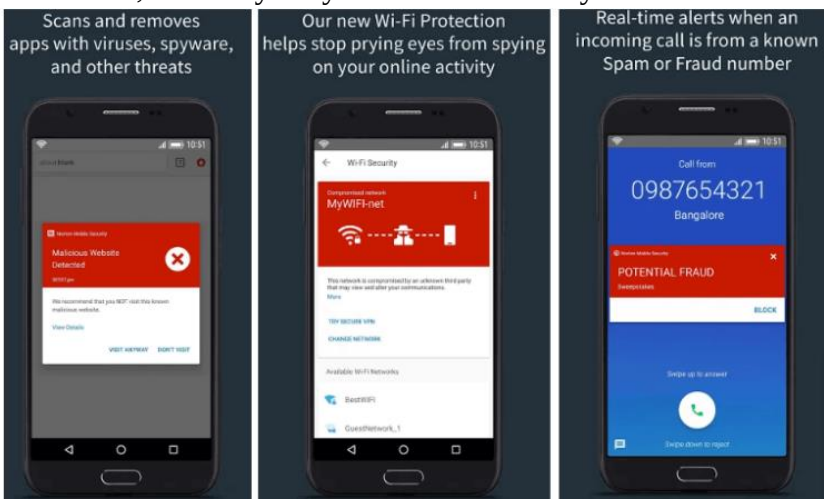
Step 5: Give a name to new Package(For example: samplePackage). Click Ok.



SECURITY AND HACKING.

Norton Mobile Security and Antivirus is a free app for Android that protects your device from malware, viruses, spyware, etc. The app provides you the privacy against websites that trap your information, malicious apps, and from common viruses, spyware, fraudulent robocalls, and malware.

Android device keeps lots of private information, and they may be unsafe for a cyber attack. Norton Mobile Security actively protects your personal information and Android device against online scammers and mobile attackers. The app comes with various features such as Wi-Fi security, app advisor, web protection, and device security.



Features of Norton Mobile Security and Antivirus

Wi-Fi Security

Wi-Fi Security helps to protect your device and keep safe when you connect to an open Wi-Fi network. The app alerts you if your connected Wi-Fi is insecure or unknown people on the same network surveillance your online activity.

App Advisor

App advisor of Norton Security automatically analysis the apps for high privacy risks, ransomware, high battery usage, data usage, malware, or invasive behavior before you download anything from the web.

Web Protection

Web protection helps you to protect and detect malicious and fraud websites and links from navigation while using your favorite apps, browser, text messages, MMS, email, social networking sites, and QR codes.

Device Security

Device security lets you know when your operating system becomes unsafe and weak so that you can take appropriate protection for your device against cybercriminals that can control your device and steal your personal information.

Android Antivirus

Norton Security scans Android virus and helps to remove apps, file, and folder that contains a virus and slow down or harm your device. It is proactive antivirus protection.

Surveillance App Protection

Surveillance App Protection feature protects your online privacy and security. This app lets you know if any app is sharing your contacts, device's location, messages, and photos without your permission.

Spam and Fraud Call Alerts

Spam and Fraud Call Alerts feature helps you to identify the unknown robocall numbers that makes you to give your personal information. You can block such calls.

Call Blocking

Call blocking feature provides facility to automatically block calls from any phone numbers or unknown or unnamed numbers only in smartphones.

Safe Search

The safe search feature correctly detects unsafe websites from your search result before visiting a website.

Norton 360 Deluxe

Norton 360 Deluxe is a complete antivirus software that protects your internet connection with a Secure VPN and your devices against viruses, malware, ransomware, and other online threats. Norton 360 Deluxe covered up to 4 devices PCs, Mac, smartphones, and tablets.

Norton 360 Deluxe antivirus provides the following features:

- Online Threat Protection
- Secure VPN
- 50 GB PC Cloud Backup
- Password Manager
- Parental Control
- SafeCam for PC
- Smart Firewall for PC or Firewall for Mac
- 100% Virus Protection Promise

UNIT V APPLICATION DEVELOPMENT

Communication via the Web – Notification and Alarms – Graphics and Multimedia: Layer Animation, Event handling and Graphics services – Telephony – Location based services

NOTIFIATIONS

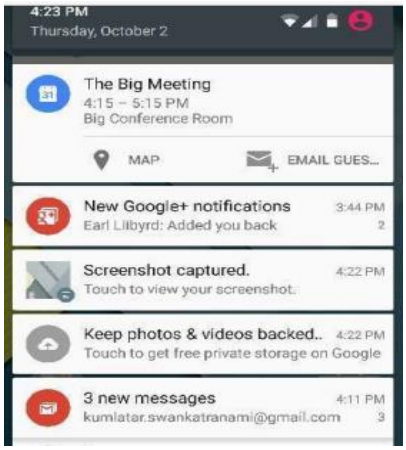
A notification is a message that Android displays outside your app's UI to provide the user with reminders, communication from other people(Whatsapp or Mail message), or other timely information (Update software information) from your app. Users can tap the notification to open your app or take an action directly from the notification.
Notification should at least have small icon, content title and content text information.

To define at least one action that needs to be performed when user clicks notification text
To handle the action events from notifications, you need to create activity class for each action.

Notification message is shown on the top of the screen, to alert users that events have occurred that may require attention.

The NotificationManager class is responsible for handling the Notifications- Its capability are
1. Create new status bar icons (Our own coding to create status bar)
2. Display additional information in the extended status bar window
3. Flash the lights/LEDs
4. Vibrate the phone
5. Sound audible alerts (ringtones, Media Store audio)

Android notifications are displayed as icons in notification area. Users can view details of notification by opening notification drawer.
Notification feature allows apps to notify event or latest information to users when app is open or closed.



Notification Manager
A notification is a message you can display to the user outside of your application's normal UI. When you tell the system to issue a notification, it first appears as an icon in the notification area.
To see the details of the notification, the user opens the notification drawer. Both the notification area and the notification drawer are system-controlled areas that the user can view at any time

Android Toast class provides a handy way to show users alerts but problem is that these alerts are not persistent (permenant) which means alert flashes on the screen for a few seconds and then disappears.

Toast Notification
A toast notification is a message that pops up on the surface of the window.
It only fills the amount of space required for the message and the user's current activity remains visible and interactive.

The notification automatically fades in and out, and does not accept interaction events.
Because a toast can be created from a background Service, it appears even if the application isn't visible or Opened.

For example, when user creates an event using calendar application it will notify the user as "Event Created" after the create action is completed.
Toast notification is best suited for one way information to the user where we don't expect any response. Toast message does not stop or disturb the current activity, just the message is shown in parallel.

Example for Android Toast Notification
Toast notification can be created from an Activity or Service. Toast is the class to be used as below.

```
Context appContext = getApplicationContext();
Toast mailMessage = Toast.makeText(appContext, "Email Received.", Toast. LENGTH_LONG);
mailMessage.setGravity(Gravity.TOP, 0, 0); //optional
mailMessage.show();
```

duration – can be either LENGTH_SHORT or LENGTH_LONG
setGravity – is used to position the message in screen. By default it shows at bottom centered. First parameter is Gravity a constant identifying location in container broadly like TOP | BOTTOM | LEFT … , second and third parameters are x, y-offset.

Status Bar Notification
A status bar notification adds an icon to the system's status bar and an expanded message (notification detail) in the "Notifications" window.
Status notification is used to display rich notification information especially from a (background) Service where user can interact. It will be shown as an icon with an alert in the status bar.
When the user pulls down the status bar, the list of notification will be in the notification window.
You can also configure the notification to alert the user with a sound, a vibration, and flashing lights on the device.

For example when a SMS message is received a message icon is shown in the status bar. On pull down, the list of unread messages will be shown in the notification window.

- Create a simple notification with an icon alert. Alert can be a ticker text message or sound or vibration or flashlight.
- Associate notification message with details shown on message expansion to activity/intent. Notification message can be a list and it is identified using a unique identifier. Existing messages can be updated too.
- Register the notification message with notification manager. NotificationManager is a system service that manages all the notifications.

Notification Alerts

Sound:

```
notification.defaults |= Notification.DEFAULT_SOUND;
notification.sound = Uri.parse("file:///sdcard/notification/robo_dance.mp3”)
```

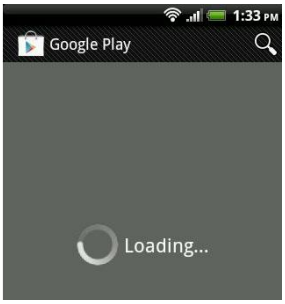
Vibration:

```
notification.defaults |= Notification.DEFAULT_VIBRATE;
//use the above default or set custom valuse as below
long[] vibrate = {0,200,100,200};
notification.vibrate = vibrate;
```

Flash Light:

```
notification.defaults |= Notification.DEFAULT_LIGHTS;
//use the above default or set custom valuse as below
notification.ledARGB = 0xffff0000;//red color
notification.flags |= Notification.FLAG_SHOW_LIGHTS;
```

Dialog Notification



Dialog notification is not an exact type of notification. Dialog is common in window based UIs. A small panel that appears on top of an active window and user will not be able to do any other activity other than acting on the dialog.
From an android Activity a dialog will be launched and the Activity loses focus. User should give input and work on the dialog. Once the user action is completed the dialog is closed. Dialog has many uses and one among them is notification to user.

For all these purposes dialog notification is used. There are many types of dialogs available such as,

- AlertDialog
- ProgressDialog
- DatePickerDialog
- TimePickerDialog

Pending Intent (Software downloading updation using wifi)

By giving a **PendingIntent** to another application, we are granting it the right to perform the operation .
This is used in setting the notification

```
Intent intent = new Intent(this, MyActivity.class);
PendingIntent launchIntent = PendingIntent.getActivity(context, 0, intent, 0);
setLatestEventInfo(context,expandedTitle,expandedText,launchIntent);
```

Create and Send Notifications-1

Create Notification Builder

As a first step is to create a notification builder using NotificationCompat.Builder.build(). You will use Notification Builder to set various Notification properties like its small and large icons, title, priority etc.

NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this)

Create and Send Notifications-2

Once you have Builder object, you can set its Notification properties using Builder object as per your requirement. But this is mandatory to set at least following

− – A small icon, set by setSmallIcon()

– A title, set by setContentTitle()

– Detail text, set by setContentText()

mBuilder.setSmallIcon(R.drawable.notification_icon);

mBuilder.setContentTitle("Notification Alert, Click Me!");

mBuilder.setContentText("Hi, This is Android Notification Detail!")

Create and Send Notifications-3

Intent resultIntent = new Intent(this, ResultActivity.class);

• TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);

• stackBuilder.addParentStack(ResultActivity.class);

• // Adds the Intent that starts the Activity to the top of the stack

• stackBuilder.addNextIntent(resultIntent);

The NotificationCompat.Builder Class

Notification build()

– Combine all of the options that have been set and return a new Notification object.

• NotificationCompat.Builder setAutoCancel (boolean autoCancel)

– Setting this flag will make it so the notification is automatically canceled when the user clicks it in the panel.

• NotificationCompat.Builder setContent (RemoteViews views)

– Supply a custom RemoteViews to use instead of the standard one.

• NotificationCompat.Builder setContentInfo (CharSequence info)

– Set the large text at the right-hand side of the notification.

• NotificationCompat.Builder setContentIntent (PendingIntent intent)

– Supply a PendingIntent to send when the notification is clicked

ANDROID ALARMMANAGER

Android AlarmManager allows you to access system alarm.

By the help of **Android AlarmManager** in android, you can *schedule your application to run at a specific time* in the future. It works whether your phone is running or not.
The Android AlarmManager holds a CPU wake lock that *provides guarantee not to sleep the phone* until broadcast (Notificattion) is handled.

To create a simple alarm clock app in Android you will need to follow these steps.

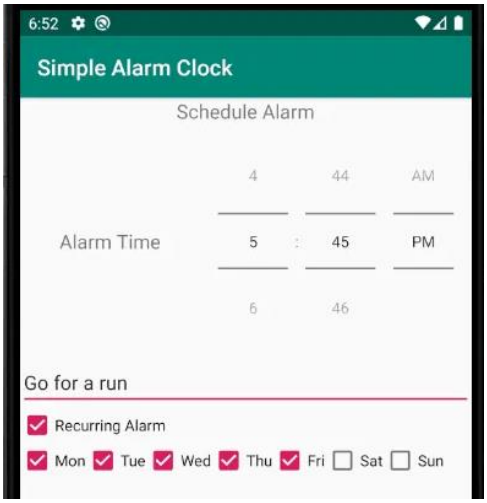
1. Capture the alarm time using a TimePicker
2. Schedule an alarm using Alarm Manager.
3. Start the Alarm Service using a Broadcast Receiver
4. Use a Notification, Media Player and Vibrator to activate the alarm
5. Managing Dismissal and Snoozing of an Alarm
6. Cancelling an Alarm using the Alarm Manager and a Pending Intent
7. Handle Rescheduling Alarm Service
8. Handle Enabling and Disabling of Alarms

Features of the Simple Alarm Clock Android App

The simple alarm clock Android app the following steps to be done.

1. Ability to set a once off alarm
2. Ability to set a recurring alarm on set days of the week
3. Ability to disable and re-enable an alarm
4. Ability to play a looped audio track for the alarm that is active
5. Ability to play a vibration effect for the alarm that is active
6. Ability to show a notification for the alarm this is active
7. Ability to dismiss an alarm
8. Ability to snooze (Ringing Postpone next small interval) an alarm

Structure of the Simple Alarm Clock Android App



In addition to fragments and activities the simple alarm clock Android app we will be building will contain a BroadcastReceiver and Service.
The BroadcastReceiver will be used to trigger (execute/fire)the alarm to start after the alarm manager generates a broadcast once the system time hits the scheduled alarm time.
The BroadcastReceiver will start a Service that will be used for the alarm. This Service will display a notification and will play an audio track for the alarm sound on loop and produce a vibration effect until the alarm is dismissed or snoozed.
The app will be structured using the Model View ViewModel (MVVM) software design pattern.

Step 1: Capturing the Alarm Time using a TimePicker

Capturing the time for the alarm to go off using a TimePicker widget.

Creating a fragment called “CreateAlarmFragment” which will be used for capturing all the details needed for the alarm such as:

- The time of day of the alarm
- A title for the alarm which will be shown in the notification when the alarm is playing
- Whether it is a recurring alarm or a once off alarm, and if it is a recurring alarm which days of the week the alarm will be active

This class provides access to the system alarm services. These allow you to schedule your application to be run at some point in the future.

When an alarm goes off, the Intent that had been registered for it is broadcast by the system, automatically starting the target application if it is not already running. Registered alarms are retained while the device is asleep (and can optionally wake the device up if they go off during that time), but will be cleared if it is turned off and rebooted.

The Alarm Manager holds a CPU wake lock as long as the alarm receiver's onReceive() method is executing. This guarantees that the phone will not sleep until you have finished handling the broadcast. Once onReceive() returns, the Alarm Manager releases this wake lock.

Nested classes

class	AlarmManager.AlarmClockInfo An immutable description of a scheduled "alarm clock" event.
interface	AlarmManager.OnAlarmListener Direct-notification alarms: the requester must be running continuously from the time the alarm is set to the time it is delivered, or delivery will fail.
String	ACTION_NEXT_ALARM_CLOCK_CHANGED Broadcast Action: Sent after the value returned by getNextAlarmClock() has changed.
String	ACTION_SCHEDULE_EXACT_ALARM_PERMISSION_STATE_CHANGED Broadcast Action: An app is granted the Manifest.permission.SCHEDULE_EXACT_ALARM permission.
long	INTERVAL_DAY Available inexact recurrence interval recognized by setInexactRepeating(int, long, long, android.app.PendingIntent) when running on Android prior to API 19.
void	cancel(AlarmManager.OnAlarmListener listener)

	Remove any alarm scheduled to be delivered to the given OnAlarmListener. getNextAlarmClock()
AlarmManager.AlarmClockInfo	Gets information about the next alarm clock currently scheduled.

void setTime(long millis)Set the system wall clock time.
void setTimeZone(String timeZone) Sets the system's persistent default time zone.

ACTION_NEXT_ALARM_CLOCK_CHANGED
public static final **String** ACTION_NEXT_ALARM_CLOCK_CHANGED

```
package example.javatut.com.alarmmanager;

import android.app.AlarmManager;
import android.app.PendingIntent;
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity {
   Button start;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      start= findViewById(R.id.button);

      start.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View view) {
            startAlert();
         }
      });
   }

public void startAlert(){
      EditText text = findViewById(R.id.time);
      int i = Integer.parseInt(text.getText().toString());
      Intent intent = new Intent(this, MyBroadcastReceiver.class);
      PendingIntent pendingIntent = PendingIntent.getBroadcast(
            this.getApplicationContext(), 234324243, intent, 0);
      AlarmManager  alarmManager = (AlarmManager) getSystemService(ALARM_SERVICE);
```

```
        alarmManager.set(AlarmManager.RTC_WAKEUP, System.currentTimeMillis()
                + (i * 1000), pendingIntent);
        Toast.makeText(this, "Alarm set in " + i + " seconds",Toast.LENGTH_LONG).show();
    }
}
```

Let's create BroadcastReceiver class that starts alarm.

File: MyBroadcastReceiver.java

```
package example.javatpoint.com.alarmmanager;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.media.MediaPlayer;
import android.widget.Toast;

public class MyBroadcastReceiver extends BroadcastReceiver {
    MediaPlayer mp;
    @Override
    public void onReceive(Context context, Intent intent) {
        mp=MediaPlayer.create(context, R.raw.alarm);
        mp.start();
        Toast.makeText(context, "Wakeup ", Toast.LENGTH_LONG).show();
    }
}
```

Graphics and animations for a rather static application should be implemented much differently than graphics and animations for an interactive game.

Graphics – are **visual images or designs on some surface**, such as a wall, canvas, screen, paper, or stone to inform.

Scalar Vector graphics are (such as **.svg**), **Raster** graphics formats (such **bmp**, **png**, **jpeg…**).

Canvas and Drawables

Android provides a set of View widgets that provide general functionality for a wide array of user interfaces. You can also extend these widgets to modify the way they look or behave. In addition, you can do your own custom 2D rendering using the various drawing methods .

The **android.graphics.Canvas** can be used to draw graphics in android. It provides methods to draw oval, rectangle, picture, text, line etc.

The **android.graphics.Paint** class is used with canvas to draw objects on canvas. It holds the information of color and style.

Android provides a huge set of 2D-drawing APIs that allow you to create graphics.

Android has got visually appealing graphics and mind blowing animations.

The Android framework provides a rich set of powerful APIS for applying animation to UI elements and graphics as well as drawing custom 2D and 3D graphics.

In this example, going to display 2D graphics in android.
File: activity_main.xml

```
<RelativeLayout xmlns:androclass="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

</RelativeLayout>
```

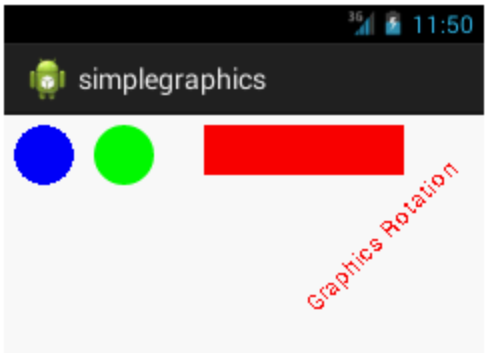
MainActivity.java

```
package com.example.simplegraphics;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.view.View;

public class MainActivity extends Activity {

    DemoView demoview;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        demoview = new DemoView(this);
        setContentView(demoview);
    }

    private class DemoView extends View{
        public DemoView(Context context){
```

```
    super(context);
}

@Override protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);

    // custom drawing code here
    Paint paint = new Paint();
    paint.setStyle(Paint.Style.FILL);

    // make the entire canvas white
    paint.setColor(Color.WHITE);
    canvas.drawPaint(paint);

    // draw blue circle with anti aliasing turned off
    paint.setAntiAlias(false);
    paint.setColor(Color.BLUE);
    canvas.drawCircle(20, 20, 15, paint);

    // draw green circle with anti aliasing turned on
    paint.setAntiAlias(true);
    paint.setColor(Color.GREEN);
    canvas.drawCircle(60, 20, 15, paint);

    // draw red rectangle with anti aliasing turned off
    paint.setAntiAlias(false);
```

```
        paint.setColor(Color.RED);
        canvas.drawRect(100, 5, 200, 30, paint);

        // draw the rotated text
        canvas.rotate(-45);

        paint.setStyle(Paint.Style.FILL);
        canvas.drawText("Graphics Rotation", 40, 180, paint);

        //undo the rotate
        canvas.restore();
    }
}
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}
```

Output:



Canvas

Android graphics provides low level graphics tools such as canvases, color, filters, points and rectangles which handle drawing to the screen directly.

The Android framework provides a set of 2D-DRAWING APIs which allows user to provide own custom graphics onto a canvas or to modify existing views to customize their look and feel.

There are two ways to draw 2D graphics,

1. Draw your animation into a View object from your layout.

2. Draw your animation directly to a Canvas.

Some of the important methods of Canvas Class are as follows

i) drawText()

ii) drawRoundRect()

iii) drawCircle()

iv) drawRect()

v) drawBitmap()

vi) drawARGB()

You can use these methods in onDraw() method to create your own custom user interface.

Drawing an animation with a View is the best option to draw simple graphics that do not need to change dynamically and are not a part of a performance-intensive game. It is used when user wants to display a static graphic or predefined animation.

Drawing an animation with a Canvas is better option when your application needs to re-draw itself regularly. For example video games should be drawing to the Canvas on its own.

MyView.java

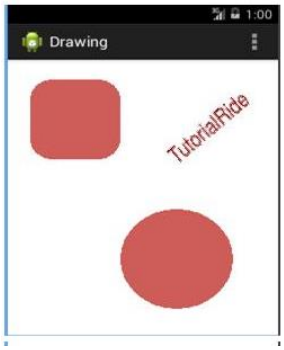
```
public class MyView extends View
{
    public MyView(Context context)
    {
        super(context);
        // TODO Auto-generated constructor stub
    }
    @Override
    protected void onDraw(Canvas canvas)
    {
        // TODO Auto-generated method stub
        super.onDraw(canvas);
        int radius;
        radius = 50;
        Paint paint = newPaint();
        paint.setStyle(Paint.Style.FILL);
```

```
        paint.setColor(Color.parseColor("#CD5C5C"));
        canvas.drawCircle(150,200, radius, paint);
        canvas.drawRoundRect(newRectF(20,20,100,100), 20, 20, paint);
        canvas.rotate(-45);
        canvas.drawText("TutorialRide", 40, 180, paint);
        canvas.restore();
    }
}
```

MainActivity.java

To pass the object of subclass that extends from View class in setContentView() method as given below. In our case the name of the subclass is MyView.

Output:




```
Public class MainActivity extends Activity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(new MyView(this));
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
```

```
        return true;
    }
}
```

ANIMATION

An animation is, is to think of it as a series of images being drawn one after another on the screen.

Tween Animation

Tween Animation takes some parameters such as start value , end value, size , time duration , rotation angle e.t.c and perform the required animation on that object.

```
Animation animation = AnimationUtils.loadAnimation(getApplicationContext(),
  R.anim.myanimation);
```

Sr.No	Method & Description
1	start() This method starts the animation.
2	setDuration(long duration) This method sets the duration of an animation.
3	getDuration() This method gets the duration which is set by above method
4	end() This method ends the animation.
5	cancel() This method cancels the animation.

To apply this animation to an object , we will just call the startAnimation() method of the object. Its syntax is –

```
ImageView image1 = (ImageView)findViewById(R.id.imageView1);
image.startAnimation(animation);
```

Steps	Description
1	You will use Android studio IDE to create an Android application and name it as My Application under a package com.example.sairam.myapplication.
2	Modify src/MainActivity.java file to add animation code
3	Modify layout XML file res/layout/activity_main.xml add any GUI component if required.
4	Create a new folder under res directory and call it anim. Confirm it by visiting res/anim
5	Right click on anim and click on new and select Android XML file You have to create different files that are listed below.
6	Create files myanimation.xml,clockwise.xml,fade.xml,move.xml,blink.xml,slide.xml and add the XML code.
7	No need to change default string constants. Android studio takes care of default constants at values/string.xml.
8	Run the application and choose a running android device and install the application on it and verify the results.

```
package com.example.sairamkrishna.myapplication;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.ImageView;
import android.widget.Toast;
```

```
public class MainActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }

public void clockwise(View view){
    ImageView image = (ImageView)findViewById(R.id.imageView);
    Animation animation = AnimationUtils.loadAnimation(getApplicationContext(),
      R.anim.myanimation);
    image.startAnimation(animation);
  }

  public void zoom(View view){
    ImageView image = (ImageView)findViewById(R.id.imageView);
    Animation animation1 = AnimationUtils.loadAnimation(getApplicationContext(),
      R.anim.clockwise);
    image.startAnimation(animation1);
  }

  public void fade(View view){
    ImageView image = (ImageView)findViewById(R.id.imageView);
    Animation animation1 =
      AnimationUtils.loadAnimation(getApplicationContext(),
        R.anim.fade);
    image.startAnimation(animation1);
  }

  public void blink(View view){
    ImageView image = (ImageView)findViewById(R.id.imageView);
    Animation animation1 =
      AnimationUtils.loadAnimation(getApplicationContext(),
        R.anim.blink);
    image.startAnimation(animation1);
  }

  public void move(View view){
    ImageView image = (ImageView)findViewById(R.id.imageView);
    Animation animation1 =
      AnimationUtils.loadAnimation(getApplicationContext(), R.anim.move);
    image.startAnimation(animation1);
  }

  public void slide(View view){
    ImageView image = (ImageView)findViewById(R.id.imageView);
    Animation animation1 =
      AnimationUtils.loadAnimation(getApplicationContext(), R.anim.slide);
```

```
    image.startAnimation(animation1);
  }
}
```

Following are the three animation systems used in Android applications:

1. Property Animation

2. View Animation

3. Drawable Animation

1. Property Animation

Property animation is the preferred method of animation in Android.

This animation is the robust framework which lets you animate any properties of any objects, view or non-view objects.

The android.animation provides classes which handle property animation.

2. View Animation

View Animation is also called as Tween Animation.

The android.view.animation provides classes which handle view animation.

This animation can be used to animate the content of a view.

It is limited to simple transformation such as moving, re-sizing and rotation, but not its background color.

3. Drawable Animation

Drawable animation is implemented using the AnimationDrawable class.

This animation works by displaying a running sequence of 'Drawable' resources that is images, frame by frame inside a view object.

EVENT HANDLING

An Event is a response to an user's interaction with input controls, such as press of a button or touch of the screen. Android framework places each occurring Event in a queue based on **FIFO** (first-in first-out) logic.

When an Event triggers, the Event Listener that is involved with the View object should be registered. Then, the registered Event Listener should implement a corresponding callback method (Event Handler) in order to handle the Event.

There are following three concepts related to Android Event Management −

- **Event Listeners** − An event listener is an interface in the View class that contains a single callback method. These methods will be called by the Android framework when the View to which the listener has been registered is triggered by user interaction with the item in the UI.(Using mouse click or Keyboard, mouseover on text or button, change text, press which button-customer care which button is pressed) and google search.
- **Event Listeners Registration** − Event Registration is the process by which an Event Handler gets registered with an Event Listener so that the handler is called when the Event Listener fires the event. Key_change( ), button_click( ), mouseover( )
- **Event Handlers** − When an event happens and we have registered an event listener for the event, the event listener calls the Event Handlers, which is the method that actually handles the event. (Which event should be fired)

There are many more event listeners available as a part of **View** class like OnHoverListener, OnDragListener etc which may be needed for your application.

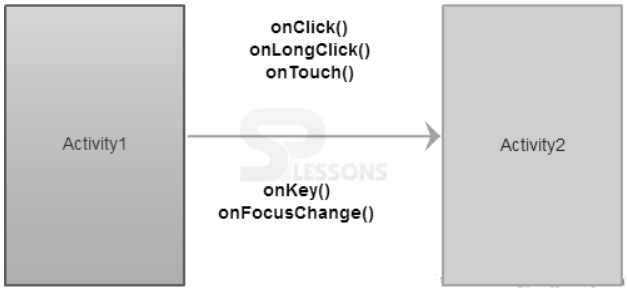
Event Listeners Registration

Event Registration is the process by which an Event Handler gets registered with an Event Listener so that the handler is called when the Event Listener fires the event.

There are different ways to register an Event Listener.

• Implement the Event Listener interface into the Activity class. (mainActivity.java)
• Use an anonymous inner Event Listener class. (General activity)
• Use a separate Event Listener class.
• Declare it into the layout XML file.

Some of the most common Event Handlers with the respective Event Listeners are as follows:



The following are the call-back methods included in the event listener interface.

Method	Description
onClick()	The system calls this method when a user clicks a View component.

Method	Description
onLongClick()	This method is called when the user touches and holds the item or focuses on the item using navigation-keys or trackball and presses and holds "enter" key or presses and holds down on the trackball (for one second).
onFocusChange()	This method is called when the user navigates onto or away from the item.
onKey()	This method is called when the user is focused on the item and presses or releases a hardware key on the device.
onTouch()	This method is called when the user performs a touch event, including a press, a release, or any movement gesture on the screen.
onCreateContextMenu()	This method is called when a Context Menu is being built.

Following is the example of registering a Button **onClick** event listener in android application.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
   ….
   Button button = (Button)findViewById(R.id.btnShow);
   // Register onClick listener with the below implementation
   button.setOnClickListener(btnListener);
….
}

// Anonymous implementation of OnClickListener
private View.OnClickListener btnListener = new View.OnClickListener() {
   public void onClick(View v) {
      // do something when the button is clicked
   }
};
```

Android Event Handlers

In android, **Event Handlers** are useful to define several callback methods when we are building custom components from view.

Following are the some of commonly used callback methods for event handling in android applications.

Method	Description
onKeyDown()	This method is called when a new key event occurs.
onKeyUp()	This method is called when a key up event occurs.
onTrackballEvent()	This method is called when a trackball motion event occurs.
onTouchEvent()	This method is called when a touch screen motion event occurs.
onFocusChanged()	This method is called when the view gains or loses focus.

Touch Mode

Users can interact with their devices by using hardware keys or buttons or touching the screen. Touching the screen puts the device into touch mode. The user can then interact with it by touching the on-screen virtual buttons(Virtual Keyboard), images, etc

Focus

A view or widget is usually highlighted or displays a flashing cursor when it's in focus. This indicates that it's ready to accept input from the user.

- **isFocusable()** − it returns true or false
- **isFocusableInTouchMode()** − checks to see if the view is focusable in touch mode. (A view may be focusable when using a hardware key but not when the device is in touch mode)

```
android:foucsUp="@=id/button_l"
```

onTouchEvent()

```
public boolean onTouchEvent(motionEvent event){
  switch(event.getAction()){
    case TOUCH_DOWN:
    Toast.makeText(this,"you have clicked down Touch button",Toast.LENTH_LONG).show();
    break();

    case TOUCH_UP:
    Toast.makeText(this,"you have clicked up touch button",Toast.LENTH_LONG).show();
    break;

    case TOUCH_MOVE:
    Toast.makeText(this,"you have clicked move touch button"Toast.LENTH_LONG).show();
    break;
  }
  return super.onTouchEvent(event) ;
}
```

Create a new android application using android studio and give names as **InputEventsExample**.
Now open an activity_main.xml file from \res\layout path and write the code like as shown below

activity_main.xml

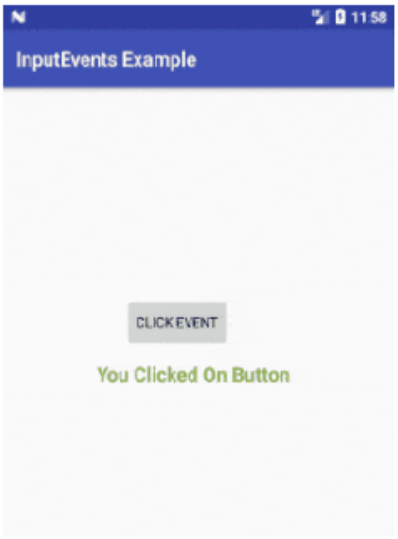
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical" >
   <Button
      android:id="@+id/btnClick"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Click Event"
      android:layout_marginTop="200dp" android:layout_marginLeft="130dp"/>
   <TextView
      android:id="@+id/txtResult"
      android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:layout_marginLeft="100dp"
    android:textColor="#86AD33"
    android:textSize="20dp"
    android:textStyle="bold"
    android:layout_marginTop="12dp"/>
</LinearLayout>
```

Once we are done with the creation of layout with required controls, need to load the XML layout resource from our activity onCreate() callback method, for that open main activity file MainActivity.java from \java\com.tutlane.inputeventsexample path and write the code like as shown below.

MainActivity.java

```
package com.tutlane.inputeventsexample;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
   Button btn;
   TextView tView;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      btn = (Button)findViewById(R.id.btnClick);
      tView = (TextView)findViewById(R.id.txtResult);
      btn.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View v) {
            tView.setText("You Clicked On Button");
         }
      });
   }
}
```



AdapterView Event Listeners

AdapterView descendants have a few more key event listeners having to do with their items:

- setOnItemClickListener - Callback when an item contained is clicked
- setOnItemLongClickListener - Callback when an item contained is clicked and held
- setOnItemSelectedListener - Callback when an item is selected

```
@Override
   public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
     // Do something here
     // The position is the index of the item pressed
     // If the third item in a list was pressed, position is `2`
   }
```

EditText Common Listeners

In addition to the listeners described above, there are a few other common listeners for input fields in particular.

- addTextChangedListener - Fires each time the text in the field is being changed
- setOnEditorActionListener - Fires when an "action" button on the soft keyboard is pressed

---

ANDROID TELEPHONYMANAGER

Android Telephony framework provides us the functionalities of the mobile. It gives us information about functionalities like calls, SMS, MMS, network, data services, IMEI number, and so on.
SMS - Short Message Service
MMS – Multimedia Message Service

The **android.telephony.TelephonyManager** class provides information about the telephony services such as subscriber id, sim serial number, phone network type etc. Moreover, you can determine the phone state etc.

Android Telephony architecture works in 4 layers that are :

1. **Communication Processor**
2. **Radio Interface Layer (RIL)**
3. **Framework Services**
4. **Applications**

Let us try to understand them briefly one by one :
1. Communication Processor
It is an input/output processor to distribute and collect data from a number of remote terminals. It is a specialized processor designed to communicate with the data communication network.

2. Radio Interface Layer
It is a bridge between the hardware and Android phone framework services. Rather we say, it is a protocol stack for Telephone. It has two main components that are:

- **RIL Daemon**– It starts when the android system starts. It reads the system properties to find a library that is to be used for Vendor RIL.
- **Vendor RIL**– It is also known as RIL Driver. It can be understood as a library that is specific to each modem.

3. Framework Services
The telephony Framework starts and initializes along with the system. All the queries by Application API are directed to RIL using these services.

4. Application
These are the Application UI related to telephony such as Dialer, SMS, MMS, Call tracker, etc. These applications start with the android system boot up. These are tied with framework services of telephony.

Android Telephony Framework consists of two types of packages that are:

1. Internal Telephony Packages: This is generally the used for default telephony app.apk.
2. Open Technology Packages: This is for third-party apps.
These are 4 layers of Android Telephony.

Implementation of Android Telephony
We will now implement it in ***Android Studio*** using the following steps:
1. At first, create a new project and name it.
2. Now, open the layout file, and define the following:

MainActivity.java

```
package com.DataFlair.androidtelephony;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.telephony.TelephonyManager;
import android.widget.TextView;

public class MainActivity extends Activity {
package com.DataFlair.androidtelephony;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.telephony.TelephonyManager;
import android.widget.TextView;

public class MainActivity extends Activity {
  TextView tv;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    tv = findViewById(R.id.textView);


    //instance of TelephonyManager
    TelephonyManager tele_man = (TelephonyManager)
getSystemService(Context.TELEPHONY_SERVICE);

    String nwcountryISO =       tele_man.getNetworkCountryIso();
    String SIMCountryISO = tele_man.getSimCountryIso();

    String PhoneType = ""; // it'll hold the type of phone i.e CDMA / GSM/ None
    int phoneType = tele_man.getPhoneType();

    switch (phoneType) {
      case (TelephonyManager.PHONE_TYPE_CDMA):
        PhoneType = "CDMA";
        break;
      case (TelephonyManager.PHONE_TYPE_GSM):
        PhoneType = "GSM";
        break;
      case (TelephonyManager.PHONE_TYPE_NONE):
```

```
            PhoneType = "NONE";
            break;
        }

        // true or false for roaming or not
        boolean checkRoaming = tele_man.isNetworkRoaming();

        String data = "Your Mobile Details are enlisted below: \n";
        data += "\n Network Country ISO is - " + nwcountryISO;
        data += "\n SIM Country ISO is - " + SIMCountryISO;
        data += "\n Network type is - " + PhoneType;
        data += "\n Roaming on is - " + checkRoaming;
        //Now we'll display the information
        tv.setText(data);
    }
}
```

Android TelephonyManager Example

Let's see the simple example of TelephonyManager that prints information of the telephony services.

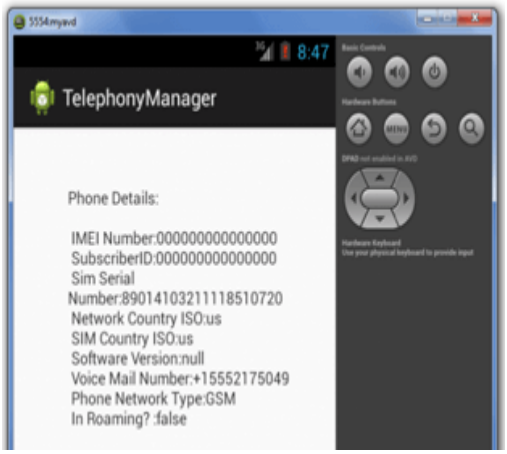
activity_main.xml

```
<RelativeLayout xmlns:androclass="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="38dp"
        android:layout_marginTop="30dp"
        android:text="Phone Details:" />

</RelativeLayout>
```

File: MainActivity.java

```
package com.javatpoint.telephonymanager;

import android.os.Bundle;
import android.app.Activity;
import android.content.Context;
import android.telephony.TelephonyManager;
import android.view.Menu;
import android.widget.TextView;
```

```
public class MainActivity extends Activity {
  TextView textView1;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      textView1=(TextView)findViewById(R.id.textView1);

      //Get the instance of TelephonyManager
      TelephonyManager  tm=(TelephonyManager)getSystemService(Context.TELEPHONY_SERVICE);

      //Calling the methods of TelephonyManager the returns the information
      String IMEINumber=tm.getDeviceId();
      String subscriberID=tm.getDeviceId();
      String SIMSerialNumber=tm.getSimSerialNumber();
      String networkCountryISO=tm.getNetworkCountryIso();
      String SIMCountryISO=tm.getSimCountryIso();
      String softwareVersion=tm.getDeviceSoftwareVersion();
      String voiceMailNumber=tm.getVoiceMailNumber();

      //Get the phone type
      String strphoneType="";

      int phoneType=tm.getPhoneType();

      switch (phoneType)
      {
            case (TelephonyManager.PHONE_TYPE_CDMA):
                  strphoneType="CDMA";
                     break;
            case (TelephonyManager.PHONE_TYPE_GSM):
                  strphoneType="GSM";
                     break;
            case (TelephonyManager.PHONE_TYPE_NONE):
                   strphoneType="NONE";
                      break;
       }

      //getting information if phone is in roaming
      boolean isRoaming=tm.isNetworkRoaming();

      String info="Phone Details:\n";
      info+="\n IMEI Number:"+IMEINumber;
      info+="\n SubscriberID:"+subscriberID;
      info+="\n Sim Serial Number:"+SIMSerialNumber;
      info+="\n Network Country ISO:"+networkCountryISO;
      info+="\n SIM Country ISO:"+SIMCountryISO;
      info+="\n Software Version:"+softwareVersion;
      info+="\n Voice Mail Number:"+voiceMailNumber;
      info+="\n Phone Network Type:"+strphoneType;
      info+="\n In Roaming? :"+isRoaming;
```



```
        textView1.setText(info);//displaying the information in the textView
    }

}
```

Interfaces

TelephonyCallback.ActiveDataSubscriptionIdListener	Interface for active data subscription ID listener.
TelephonyCallback.CallDisconnectCauseListener	Interface for call disconnect cause listener.
TelephonyCallback.CallForwardingIndicatorListener	Interface for call-forwarding indicator listener.
TelephonyCallback.CallStateListener	Interface for call state listener.
TelephonyCallback.CarrierNetworkListener	Interface for carrier network listener.
TelephonyCallback.CellInfoListener	Interface for cell info listener.
TelephonyCallback.CellLocationListener	Interface for device cell location listener.
TelephonyCallback.DataActivationStateListener	Interface for SIM data activation state listener.
TelephonyCallback.DataActivityListener	Interface for data activity state listener.
TelephonyCallback.DataConnectionStateListener	Interface for data connection state listener.
TelephonyCallback.DisplayInfoListener	Interface for display info listener.
TelephonyCallback.EmergencyNumberListListener	Interface for the current emergency number list listener.
TelephonyCallback.ImsCallDisconnectCauseListener	Interface for IMS call disconnect cause listener.
TelephonyCallback.MessageWaitingIndicatorListener	Interface for message waiting indicator listener.
TelephonyCallback.SignalStrengthsListener	Interface for network signal strengths listener.
TelephonyCallback.UserMobileDataStateListener	Interface for user mobile data state listener.

Classes

AccessNetworkConstants	Contains access network related constants.
AvailableNetworkInfo	Defines available network information which includes
	corresponding subscription id, network plans.
	TelephonyManager#updateAvailableNetworks
BarringInfo	Provides the barring configuration for a particular service type.
CarrierConfigManager	Provides access to telephony configuration values that are carrier-specific.
CarrierConfigManager.Gps	GPS configs.
CellIdentity	CellIdentity represents the identity of a unique cell.
CellIdentityCdma	CellIdentity is to represent a unique CDMA cell
CellIdentityGsm	CellIdentity to represent a unique GSM cell
CellSignalStrengthCdma	Signal strength related information.
CellSignalStrengthGsm	GSM signal strength related information.
CellSignalStrengthLte	LTE signal strength related information.
CellSignalStrengthNr	5G NR signal strength related information.
DataFailCause	DataFailCause collects data connection failure causes code from different sources.
DisconnectCause	Describes the cause of a disconnected call.
NetworkScanRequest	Defines a request to peform a network scan.
PhoneNumberUtils	Various utilities for dealing with phone number strings.
PreciseDataConnectionState	Contains precise data connection state.
RadioAccessSpecifier	Describes a particular radio access network to be scanned.
ServiceState	Contains phone state and service related information.
SignalStrength	Contains phone signal strength related information.

SignalStrengthUpdateRequest.Builder	Builder class to create SignalStrengthUpdateRequest object.
SignalThresholdInfo	Defines the threshold value of the signal strength.
SignalThresholdInfo.Builder	Builder class to create SignalThresholdInfo objects.
SmsManager	Manages SMS operations such as sending data, text, and pdu SMS messages.
SmsManager.FinancialSmsCallback	callback for providing asynchronous sms messages for financial app.
SmsMessage	A Short Message Service message.
SubscriptionInfo	A Parcelable class for Subscription Information.
SubscriptionManager	SubscriptionManager is the application interface to SubscriptionController and provides information about the current Telephony Subscriptions.
SubscriptionManager.OnOpportunisticSubscriptionsChangedListener	A listener class for monitoring changes to SubscriptionInfo records of opportunistic subscriptions.
SubscriptionManager.OnSubscriptionsChangedListener	A listener class for monitoring changes to SubscriptionInfo records.
SubscriptionPlan	Description of a billing relationship plan between a carrier and a specific subscriber.
SubscriptionPlan.Builder	Builder for a SubscriptionPlan.
TelephonyCallback	A callback class for monitoring changes in specific telephony states on the device, including service state, signal strength, message waiting indicator (voicemail), and others.
TelephonyDisplayInfo	TelephonyDisplayInfo contains telephony-related information used for display purposes only.
TelephonyManager	Provides access to information about the telephony services on the device.
TelephonyManager.CellInfoCallback	Callback for providing asynchronous CellInfo on request

TelephonyManager.UssdResponseCallback	Used to notify callers of TelephonyManager#sendUssdRequest(String, UssdResponseCallback, Handler) when the network either successfully executes a USSD request, or if there was a failure while executing the request.
TelephonyScanManager	Manages the radio access network scan requests and callbacks.
TelephonyScanManager.NetworkScanCallback	The caller of TelephonyManager.requestNetworkScan(android.telephony.NetworkScanRequest, java.util.concurrent.Executor, android.telephony.TelephonyScanManager.NetworkScanCallback) should implement and provide this callback so that the scan results or errors can be returned.
VisualVoicemailService	This service is implemented by dialer apps that wishes to handle OMTP or similar visual voicemails.
VisualVoicemailService.VisualVoicemailTask	Represents a visual voicemail event which needs to be handled.

Exceptions

TelephonyManager.CallComposerException	Exception that may be supplied to the callback in TelephonyManager.uploadCallComposerPicture(InputStream, String, Executor, OutcomeReceiver) if something goes awry.
TelephonyManager.ModemErrorException	Exception that is supplied to the callback in TelephonyManager.getNetworkSlicingConfiguration(Executor, OutcomeReceiver) if the modem returned a failure.
TelephonyManager.NetworkSlicingException	Exception that may be supplied to the callback in TelephonyManager.getNetworkSlicingConfiguration(Executor, OutcomeReceiver) if something goes awry.
TelephonyManager.TimeoutException	Exception that is supplied to the callback in TelephonyManager.getNetworkSlicingConfiguration(Executor, OutcomeReceiver) if the system timed out waiting for a response from the Radio.

MC4204 MOBILE APPLICATION DEVELOPMENT

UNIT I INTRODUCTION

Mobile Application Model – Infrastructure and Managing Resources – Mobile Device Profiles – Frameworks and Tools

Mobile application development

Mobile application development is **the process of creating software applications that run on a mobile device**, and a typical mobile application utilizes a network connection to work with remote computing resources.

Mobile Application Model

Types of Mobile Apps by Technology

There are three basic types of mobile apps if we categorize them by the technology used to code them:

- **Native apps** are created for one specific platform or operating system.
- **Web apps** are responsive versions of websites that can work on any mobile device or OS because they're delivered using a mobile browser.
- **Hybrid apps** are combinations of both native and web apps, but wrapped within a native app, giving it the ability to have its own icon or be downloaded from an app store.

01. Native Apps

Native apps are built specifically for a mobile device's operating system (OS). Thus, you can have native Android mobile apps or native iOS apps, not to mention all the other platforms and devices. Because they're built for just one platform, you cannot mix and match

– say, use a Blackberry app on an Android phone or use an iOS app on a Windows phone.

Technology Used: Native apps are coded using a variety of programming languages. Some examples include: Java, Kotlin, Python, Swift, Objective-C, C++, and React.

02. Web Apps

Web apps behave similarly to native apps but are accessed via a web browser on your mobile device. They're not standalone apps in the sense of having to download and install code into your device. They're actually responsive websites that adapt its user interface to the device the user is on. In fact, when you come across the option to "install" a web app, it often simply bookmarks the website URL on your device.

One kind of web app is the progressive web app (PWA), which is basically a native app running inside a browser. For a deeper discussion on PWAs

Technology Used: Web apps are designed using HTML5, CSS, JavaScript, Ruby, and similar programming languages used for web work.

03. Hybrid Apps

And then there are the hybrid apps. These are web apps that look and feel like native apps. They might have a home screen app icon, responsive design, fast performance, even be able to function offline, but they're really web apps made to look native.

Technology Used: Hybrid apps use a mixture of web technologies and native APIs. They're developed using: Ionic, Objective C, Swift, HTML5, and others.

- Listening to short talks and lectures and completing listening comprehension exercises
- Listening to TED Talks

2. Speaking:
   - Giving one minute talks
   - Participating in small Group Discussions
   - Making Presentations

3. Reading:
   - Reading Comprehension
   - Reading subject specific material
   - Technical Vocabulary

4. Writing:
   - Formal vs Informal Writing
   - Paragraph Writing
   - Essay Writing
   - Email Writing

REFERENCES / MANUALS / SOFTWARE: Open Sources / websites

TOTAL: 30 PERIODS

COURSE OUTCOMES:

On completion of the course, the students will be able to:

- Listen and comprehend lectures in English
- Articulate well and give presentations clearly
- Participate in Group Discussions successfully
- Communicate effectively in formal and informal writing
- Write proficient essays and emails

CO-PO Mapping

CO	POs					
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**
1	1	2	2	1	1	1
2	1	3	2	1	1	1
3	1	2	3	1	1	1
4	1	3	2	1	1	1
5	1	3	2	1	1	1
Avg	1	2.6	2.2	1	1	1

MC4201 **FULL STACK WEB DEVELOPMENT** **L T P C**

3 0 0 3

COURSE OBJECTIVES:

- To understand the fundamentals of web programming and client side scripting.
- To learn server side development using NodeJS.

- To understand API development with Express Framework.
- To understand and architect databases using NoSQL and SQL databases.
- To learn the advanced client side scripting and ReactJS framework

UNIT I INTRODUCTION TO CSS and JAVASCRIPT 9

Introduction to Web: Server - Client - Communication Protocol (HTTP) – Structure of HTML Documents – Basic Markup tags – Working with Text and Images with CSS– CSS Selectors – CSS Flexbox - JavaScript: Data Types and Variables - Functions - Events – AJAX: GET and POST

UNIT II SERVER SIDE PROGRAMMING WITH NODE JS 9

Introduction to Web Servers – Javascript in the Desktop with NodeJS – NPM – Serving files with the http module – Introduction to the Express framework – Server-side rendering with Templating Engines – Static Files - async/await - Fetching JSON from Express

UNIT III ADVANCED NODE JS AND DATABASE 9

Introduction to NoSQL databases – MongoDB system overview - Basic querying with MongoDB shell – Request body parsing in Express – NodeJS MongoDB connection – Adding and retrieving data to MongoDB from NodeJS – Handling SQL databases from NodeJS – Handling Cookies in NodeJS – Handling User Authentication with NodeJS

UNIT IV ADVANCED CLIENT SIDE PROGRAMMING 9

React JS: ReactDOM - JSX - Components - Properties – Fetch API - State and Lifecycle - -JS Localstorage - Events - Lifting State Up - Composition and Inheritance

UNIT V APP IMPLEMENTATION IN CLOUD 9

Cloud providers Overview – Virtual Private Cloud – Scaling (Horizontal and Vertical) – Virtual Machines, Ethernet and Switches – Docker Container – Kubernetes

TOTAL: 45 PERIODS

SUGGESTED ACTIVITIES:

1. Build an online MCQ quiz app. The questions and options should be fetched based on the chosen topic from a NodeJS server. The questions can be stored in a JSON file in the backend. Once the user has answered the questions, the frontend must send the chosen options to the backend and the backend must identify the right answers and send the score back to the front end. The frontend must display the score in a separate neatly designed page.
2. Build a blog website where you can add blog posts through a simple admin panel and the users can view the blog posts. The contents of the blog posts can be stored in either MongoDB or MySQL database. The home page should contain the titles of the blog post and the full post can be viewed by clicking the title. Frontend can be built either using React or through template engines served by the NodeJS server.
3. Take any ecommerce or social media website/app. Analyze what the API endpoints would have been used for and how the frontend interacts with the backend. The networks tab in the browser's developer tools can be used if required.
4. Architect an entire database structure for an E-Commerce application for MongoDB. Discuss how the database would have been structured if you were using a SQL database.
5. Build a simple calculator app with React. The user should be able to add numbers and operations to the app by clicking on buttons, just like you would do in a mobile phone. The moment the operation and the two operations are defined, the answer should be displayed

M.A.M COLLEGE OF ENGINEERING

MASTER OF COMPUTER APPLICATION

MC4201 - FULL STACK WEB DEVELOPMENT

UNIT I INTRODUCTION TO CSS and JAVASCRIPT

Introduction to Web: Server - Client - Communication Protocol (HTTP) – Structure of HTML Documents – Basic Markup tags – Working with Text and Images with CSS– CSS Selectors – CSS Flexbox - JavaScript: Data Types and Variables - Functions - Events – AJAX: GET and POST

UNIT II SERVER SIDE PROGRAMMING WITH NODE JS

Introduction to Web Servers – Javascript in the Desktop with NodeJS – NPM – Serving files with the http module – Introduction to the Express framework – Server-side rendering with Templating Engines – Static Files - async/await - Fetching JSON from Express

UNIT III ADVANCED NODE JS AND DATABASE

Introduction to NoSQL databases – MongoDB system overview - Basic querying with MongoDB shell – Request body parsing in Express – NodeJS MongoDB connection – Adding and retrieving data to MongoDB from NodeJS – Handling SQL databases from NodeJS – Handling Cookies in NodeJS – Handling User Authentication with NodeJS

UNIT IV ADVANCED CLIENT SIDE PROGRAMMING

React JS: ReactDOM - JSX - Components - Properties – Fetch API - State and Lifecycle - -JS Localstorage - Events - Lifting State Up - Composition and Inheritance

UNIT V APP IMPLEMENTATION IN CLOUD

Cloud providers Overview – Virtual Private Cloud – Scaling (Horizontal and Vertical) – Virtual Machines, Ethernet and Switches – Docker Container – Kubernetes

UNIT I

INTRODUCTION TO CSS and JAVASCRIPT

Introduction to CSS

CSS stands for Cascading Style Sheets. CSS describes how HTML elements are to be displayed on screen, paper, or in other media. CSS saves a lot of work. It can control the layout of multiple web pages all at once. External stylesheets are stored in CSS files.

Cascading **S**tyle **S**heets, fondly referred to as **CSS**, is a simply designed language intended to simplify the process of making web pages presentable. CSS allows you to apply styles to web pages.

More importantly, CSS enables you to do this independent of the HTML that makes up each web page. It describes how a webpage should look: it prescribes colors,fonts, spacing, and much more.

while html uses tags, css uses rulesets.

WHY CSS?

- **CSS saves time:** write CSS once and reuse the same sheet in multiple HTML pages.
- **Easy Maintenance:** To make a global change simply change the style, and all elements in all the webpages will be updated automatically.
- **Search Engines:** CSS is considered a clean coding technique, which means search engines won't have to struggle to "read" its content.
- **Superior styles to HTML:** CSS has a much wider array of attributes than HTML, so you can give a far better look to your HTML page in comparison to HTML attributes.
- **Offline Browsing:** CSS can store web applications locally with the help of an offline cache. Using this we can view offline websites.

CSS Syntax:

A CSS comprises style rules that are interpreted by the browser and then applied to the corresponding elements in your document.

A style rule set consists of a selector and declaration block.

Selector -- h1

Declaration -- {color:blue;font size:12px;}

- The selector points to the HTML element you want to style.
- The declaration block contains one or more declarations separated by semicolons.
- Each declaration includes a CSS property name and a value, separated by a colon.For Example:

–; color is property and blue is value.

–; font-size is property and 12px is value.

A CSS declaration always ends with a semicolon, and declaration blocks aresurrounded by curly braces.

Example :

In the following example all p elements will be center-aligned, with a blue text color:

- CSS

```
p {
        color: blue;
        text-align: center;
}
```

Hello World!

These paragraphs are styled with CSS.

CSS Versions

1. CSS1
2. CSS2
3. CSS3
4. CSS4

Version 4 comes with:-

- CSS-Pro
- CSS-Mobile

Supported Browser:

- Google Chrome
- Microsoft Edge
- Firefox
- Opera
- Safari

What is JavaScript introduction?

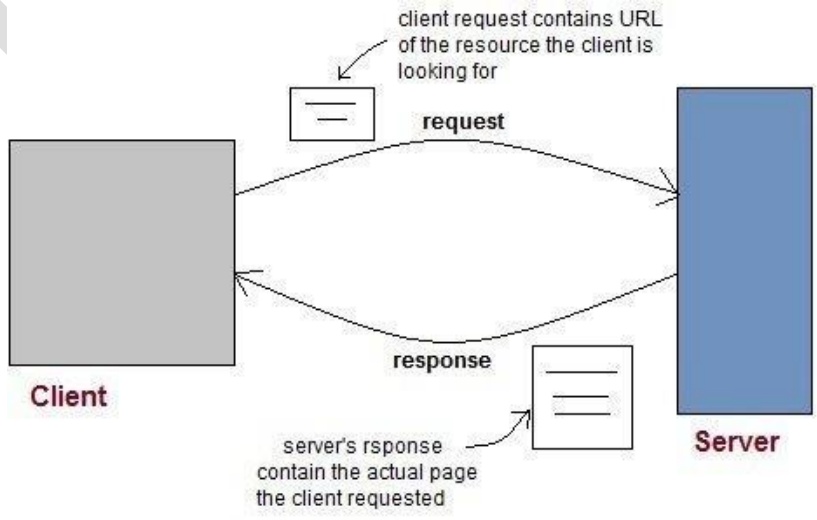
JavaScript is **a lightweight, cross-platform, and interpreted compiled programming language which is also known as the scripting language for webpages**. It is well-known for the development of web pages, many non-browser environments also use it.

- CSS is a styling language used to style HTML pages so that they can be used to attract users.
- JavaScript is a programming language that changes the appearance of web pages, and it is dynamic.
- CSS is static and is related to the colour, position, size and style of the web pages, and the appearance is made beautiful.

Introduction to Web

Web consists of billions of clients and server connected through wires and wireless networks. The web clients make requests to web server. The web server receives the request, finds the resources and return the response to the client. When a server answers a request, it usually sends some type of content to the client.

The client uses web browser to send request to the server. The server often sends response to the browser with a set of instructions written in HTML(HyperText Markup Language). All browsers know how to display HTML page to the client.



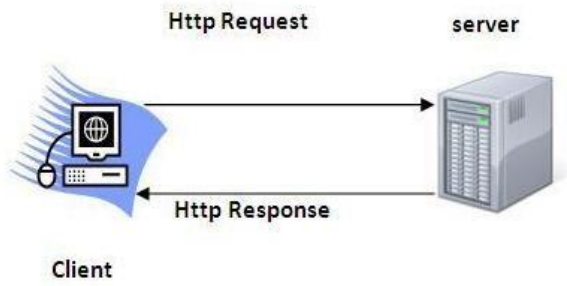
Web Application

A website is a collection of static files(webpages) such as HTML pages, images, graphics etc. A Web application is a web site with dynamic functionality on the server. Google, Facebook, Twitter are examples of web applications.

Server - Client - Communication Protocol (HTTP)

The Hypertext Transfer Protocol (HTTP) is application-level protocol for collaborative, distributed, hypermedia information systems. It is the data communication protocol used to establish communication between client and server.

HTTP is TCP/IP based communication protocol, which is used to deliver the data like image files, query results, HTML files etc on the World Wide Web (WWW) with the default port is TCP 80. It provides the standardized way for computers to communicate with each other.



The Basic Characteristics of HTTP (Hyper Text Transfer Protocol):

- It is the protocol that allows web servers and browsers to exchange data over theweb.
- It is a request response protocol.
- It uses the reliable TCP connections by default on TCP port 80.
- It is stateless means each request is considered as the new request. In other words, server doesn't recognize the user by default.

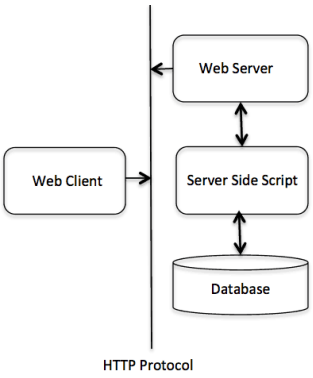
The Basic Features of HTTP (Hyper Text Transfer Protocol):

There are three fundamental features that make the HTTP a simple and powerful protocol used for communication:

- **HTTP is media independent:** It specifies that any type of media content can be sent by HTTP as long as both the server and the client can handle the data content.
- **HTTP is connectionless:** It is a connectionless approach in which HTTP client i.e., a browser initiates the HTTP request and after the request is sent the client disconnects from server and waits for the response.
- **HTTP is stateless:** The client and server are aware of each other during a current request only. Afterwards, both of them forget each other. Due to the stateless nature of protocol, neither the client nor the server can retain the information about different request across the web pages.

The Basic Architecture of HTTP (Hyper Text Transfer Protocol):

The below diagram represents the basic architecture of web application and depictswhere HTTP stands:



HTTP is request/response protocol which is based on client/server based architecture. In this protocol, web browser, search engines, etc. behave as HTTP clients and the Web server like Servlet behaves as a server

Clients and servers communicate by exchanging individual messages (as opposed to a stream of data). The messages sent by the client, usually a Web browser,are called requests and the messages sent by the server as an answer are called responses.

A request is made by an entity called a user-agent, which is typically a web browser however can be a bot or scraper. The server answer with a response. In between can be any number of proxies or caches that can act as gateways.

HTTP is stateless, which means inherently data isn't saved. HTTP cookies allow use of stateful sessions. This might be used for example with an e-commerce website asyou click from page to page.

HTTP Request

Requests consists of the following elements:

- An HTTP method, usually a verb like GET, POST or a noun like OPTIONS or HEAD that defines the operation the client wants to perform. Typically, a client wants to fetch a resource (using GET) or post the value of an HTML form (using POST), though more operations may be needed in othercases.
- The path of the resource to fetch; the URL of the resource stripped from elementsthat are obvious from the context, for example without the protocol (http://), the domain (here, developer.mozilla.org), or the TCP port (here, 80).
- The version of the HTTP protocol.
- Optional headers that convey additional information for the servers.
- Or a body, for some methods like POST, similar to those in responses, which contain the resource sent.

HTTP Response

Responses consist of the following elements:

- The version of the HTTP protocol they follow.
- A status code, indicating if the request was successful, or not, and why.
- A status message, a non-authoritative short description of the status code.
- HTTP headers, like those for requests.
- Optionally, a body containing the fetched resource.

Structure of HTML Documents

INTRODUCTION

HTML is a language of the web. It's used to design the web pages or we can say structure the page layouts of a website. HTML stands for HYPERTEXT MARKUP LANGUAGE, as its full form suggests it's not any programming language, a markup language.

So, while the execution of HTML code we can't face any such error. In real HTML code wasn't compiled or interpreted because HTML code was rendered by thebrowser. which is similar to the compilation of a program. Html content is parched through the browser to display the content of HTML.

HTML DOCUMENTS STRUCTURE

Html used predefined tags and attributes to tell the browser how to display content, means in which format, style, font size, and images to display. Html is a case insensitive language. **Case insensitive** means there is no difference in upper case andlower case (capital and small letters) both treated as the same, for r example 'D' and 'd' both are the same here.

There are generally two types of tags in HTML:

1. **Paired Tags**: These tags come in pairs. That is they have both opening(< >) and closing(</ >) tags.
2. **Empty Tags**: These tags do not require to be closed.

Below is an example of a () tag in HTML, which tells the browser to bold the textinside it.

 I MCA

OUTPUT:

I MCA

Tags and attributes: Tags are individuals of html structure, we have to open and close any tag with a forward slash like this <h1> </h1>. There are some variations with the tag some of them are self-closing tag which isn't required to close and someare empty tag where we can add any attributes in it.

Attributes are additional properties of html tags that define the property of any html tags. i.e. width, height, controls, loops, input, and autoplay. These attributesalso help us to store information in meta tags by using name, content, and type attributes. Html documents structured mentioned below:

Structure of an HTML Document

An HTML Document is mainly divided into two parts:

- **HEAD**: This contains the information about the HTML document. For Example, the Title of the page, version of HTML, Meta Data, etc.
- **BODY**: This contains everything you want to display on the Web Page.

Let us now have a look at the basic structure of HTML. That is the code that is a mustfor every webpage to have:

```
<!DOCTYPE html>
<html>

<head>
    <title>Page Title</title>
</head>

<body>
    <h2>Heading Content</h2>
    <p>Paragraph Content</p>
</body>

</html>
```

<!DOCTYPE html>: This tag is used to tells the HTML version. This currentlytells that the version is HTML 5.0

<html> </html> : <html> is a root element of html. It's a biggest and main element in complete html language, all the tags , elements and attributes enclosed init or we can say wrap init , which is used to structure a web page. <html> tag is parenttag of <head> and <body> tag , other tags enclosed within <head > and <body>.

In <html > tag we use "lang" attributes to define languages of html page suchas <html lang="en"> here en represents English language. some of them are : es = Spanish , zh-Hans = Chinese, fr= french and el= Greek etc.

<head>: Head tag contains metadata, title, page CSS etc. Data stored in the <head> tag is not displayed to the user, it is just written for reference purposes and asa watermark of the owner.

Note: for better understanding refer above code of html.

<tittle> = to store website name or content to be displayed.

<link> = To add/ link css(cascading style sheet) file.

<meta> = 1. to store data about website, organisation , creator/ owner
2. for responsive website via attributes
3. to tell compatibility of html with browser

<script> = to add javascript file.

<body>: A body tag is used to enclose all the data which a web page has from texts tolinks. All the content that you see rendered in the browser is contained within this element.

Following tags and elements used in the body.1 .

<h1> ,<h2> ,<h3> to <h6>
2. <p>
3. and
4. <b>, <i> and<u>
5. ,<ul>and<ol>.
6. <img> , <audio> , <video> and<iframe>
7. <table> <th> , <thead>and<tr>.
8. <form>
9. <label> and <input>

HTML is the foundation of web pages, and is used for webpage development bystructuring websites and web apps.

Basic Markup tags

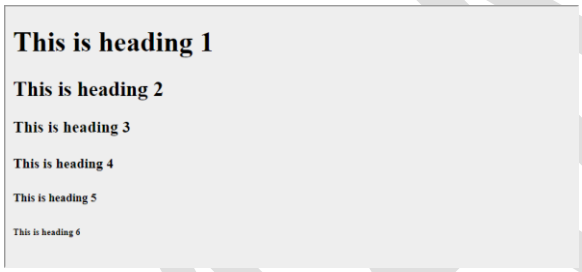
1. Heading Tags

Any document starts with a heading. You can use different sizes for your headings. HTML also has six levels of headings, which use the elements **<h1>, <h2>, <h3>, <h4>, <h5>,** and **<h6>**. While displaying any heading, browser adds one linebefore and one line after that heading.

Example

```
<!DOCTYPE html>
<html>
 <head>
   <title>Heading Example</title>
 </head>
 <body>
   <h1>This is heading 1</h1>
   <h2>This is heading 2</h2>
   <h3>This is heading 3</h3>
   <h4>This is heading 4</h4>
   <h5>This is heading 5</h5>
   <h6>This is heading 6</h6>
 </body>
</html>
```

This will produce the following result −



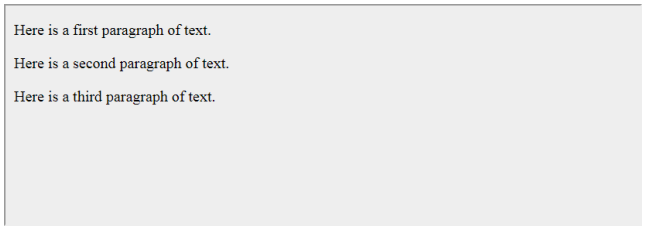
2. Paragraph Tag

The ** ** tag offers a way to structure your text into different paragraphs. Each paragraph of text should go in between an opening and a closing tag as shown below in the example −

Example

```
<!DOCTYPE html>
<html>
 <head>
   <title>Paragraph Example</title>
 </head>
 <body>
   <p>Here is a first paragraph of text.</p>
   <p>Here is a second paragraph of text.</p>
   <p>Here is a third paragraph of text.</p>
 </body>
</html>
```

This will produce the following result −



3. Line Break Tag

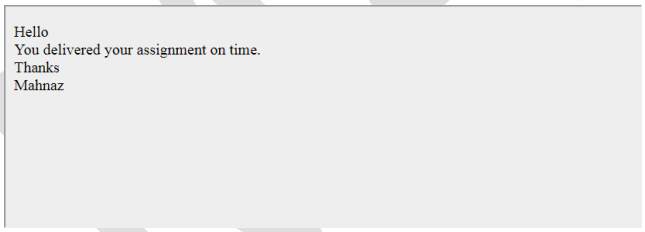
Whenever you use the **
** element, anything following it starts from the next line. This tag is an example of an **empty** element, where you do not need opening and closing tags, as there is nothing to go in between them.

The
 tag has a space between the characters **br** and the forward slash. If you omit this space, older browsers will have trouble rendering the line break, while if you miss the forward slash character and just use
 it is not valid in XHTML.

Example

```
<!DOCTYPE html>
<html>
  <head>
    <title>Line Break  Example</title>
  </head>
  <body>
    <p>Hello<br />
      You delivered your assignment ontime.<br />
      Thanks<br />
      Mahnaz</p>
  </body>
</html>
```

This will produce the following result −



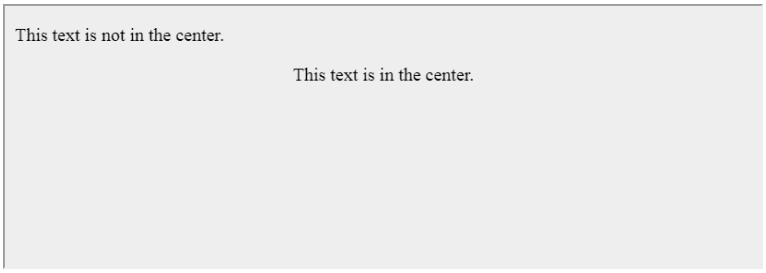
4. Centering Content

You can use ** ** tag to put any content in the center of the page or any table cell.

Example

```
<!DOCTYPE html>
<html>
  <head>
    <title>Centring Content Example</title>
  </head>
  <body>
    <p>This text is not in the center.</p>
    <center>
      <p>This text is in the center.</p>
    </center>
```

```
  </body>
</html>
```



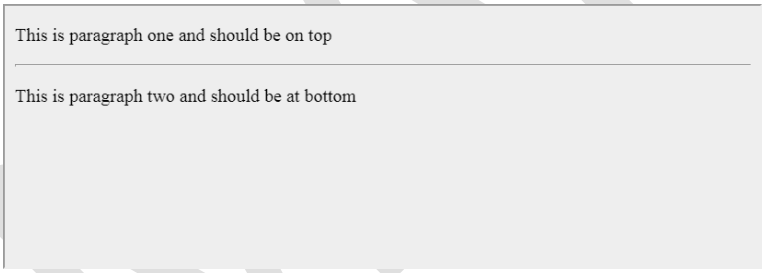
5. Horizontal Lines

Horizontal lines are used to visually break-up sections of a document. The **<hr>** tag creates a line from the current position in the document to the right margin and breaks the line accordingly.

Example

```
<!DOCTYPE html>
<html>
  <head>
    <title>Horizontal Line Example</title>
  </head>
  <body>
    <p>This is paragraph one and should be on top</p>
    <hr />
    <p>This is paragraph two and should be at bottom</p>
  </body>
</html>
```

This will produce the following result −



Again **<hr />** tag is an example of the **empty** element, where you do not need opening and closing tags, as there is nothing to go in between them.

The **<hr />** element has a space between the characters **hr** and the forward slash. If you omit this space, older browsers will have trouble rendering the horizontal line, while if you miss the forward slash character and just use **<hr>** it is not valid in XHTML

6. Preserve Formatting

Sometimes, you want your text to follow the exact format of how it is written inthe HTML document. In these cases, you can use the preformatted tag **<pre>**.

Any text between the opening **<pre>** tag and the closing **</pre>** tag willpreserve the formatting of the source document.

Example

```
<!DOCTYPE html>
<html>
  <head>
    <title>Preserve Formatting Example</title>
```

```
  </head>
  <body>
    <pre>
      function testFunction( strText ){
        alert (strText)
       }
     </pre>
   </body>
</html>
```

```
function testFunction( strText ){
   alert (strText)
}
```

Working with Text and Images with CSS

1. *CSS Text*

CSS has a lot of properties for formatting text.

Text Color

The color property is used to set the color of the text. The color is specified by:

- a color name - like "red"
- a HEX value - like "#ff0000"
- an RGB value - like "rgb(255,0,0)"

Look at CSS Color Values for a complete list of possible color values.

The default text color for a page is defined in the body selector.

Example

```
body {
  color: blue;
}

h1 {
  color: green;
}
```

Text Color and Background Color

In this example, we define both the background-color property and the color property:

Example

```
body {
  background-color: lightgrey;
  color: blue;
}

h1 {
  background-color: black;
  color: white;
}

div {
  background-color: blue;
  color: white;
}
```

CSS Text Alignment

CSS Text Alignment and Text Direction properties:

- text-align
- text-align-last
- direction
- unicode-bidi
- vertical-align

Text Alignment

- The text-align property is used to set the horizontal alignment of a text.
- A text can be left or right aligned, centered, or justified.
- The following example shows center aligned, and left and right aligned text (left alignment is default if text direction is left-to-right, and right alignment is default if text direction is right-to-left):

When the text-align property is set to "justify", each line is stretched so that every line has equal width,

Example

```
h1 {
  text-align: center;
}

h2 {
  text-align: left;
}

h3 {
  text-align: right;
}
```

and the left and right margins are straight (like in magazines and newspapers):

Example

```
div {
  text-align: justify;
}
```

Text Align Last

The text-align-last property specifies how to align the last line of a text.

Example

Align the last line of text in three <p> elements:

```
p.a {
  text-align-last: right;
}

p.b {
  text-align-last: center;
}

p.c {
  text-align-last: justify;
}
```

Text Direction

The direction and unicode-bidi properties can be used to change the text direction of anelement:

Example

```
p {
  direction: rtl;
  unicode-bidi: bidi-override;
}
```

Vertical Alignment

The vertical-align property sets the vertical alignment of an element.

EXAMPLE:

Set the vertical alignment of an image in a text:

```
img.a {
 vertical-align: baseline;
}
img.b {
 vertical-align: text-top;
}
img.c {
 vertical-align: text-bottom;
}
img.d {
 vertical-align: sub;
}
img.e {
 vertical-align: super;
}
```

The CSS Text Alignment/Direction Properties

Property	Description
direction	**Specifies the text direction/writing direction**
text-align	**Specifies the horizontal alignment of text**
text-align-last	**Specifies how to align the last line of a text**
unicode-bidi	**Used together with the direction property to set or return whether the text should be overridden to support multiple languages in the same document**
vertical-align	**Sets the vertical alignment of an element**

2. CSS Text Decoration

CSS Text Decoration properties:

- text-decoration-line
- text-decoration-color
- text-decoration-style
- text-decoration-thickness
- text-decoration

Add a Decoration Line to Text

The text-decoration-line property is used to add a decoration line to text.

```
h1 {
  text-decoration-line: overline;
}

h2 {
  text-decoration-line: line-through;
}

h3 {
  text-decoration-line: underline;
}

p {
  text-decoration-line: overline underline;
}
```

Specify a Color for the Decoration Line

The text-decoration-color property is used to set the color of the decoration line.

Example

```
h1 {
  text-decoration-line: overline;
  text-decoration-color: red;
}
h2 {
  text-decoration-line: line-through;
  text-decoration-color: blue;
}
h3 {
  text-decoration-line: underline;
  text-decoration-color: green;
}
p {
  text-decoration-line: overline underline;
  text-decoration-color: purple;

}
```

All CSS text-decoration Properties

Property	Description
text-decoration	**Sets all the text-decoration properties inone declaration**
text-decoration-color	**Specifies the color of the text-decoration**
text-decoration-line	**Specifies the kind of text decoration to beused (underline, overline, etc.)**
text-decoration-style	**Specifies the style of the text decoration(solid, dotted, etc.)**
text-decoration-thickness	**Specifies the thickness of the text decorationline**

3. *CSS Text Transformation*

- ✓ Text Transformation
- ✓ The text-transform property is used to specify uppercase and lowercase lettersin a text.
- ✓ It can be used to turn everything into uppercase or lowercase letters, or capitalize the first letter of each word:

Example

```
p.uppercase {
  text-transform: uppercase;
}

p.lowercase {
  text-transform: lowercase;
}

p.capitalize {
  text-transform: capitalize;
}
```

4. *CSS Text Spacing*

CSS Text Indentation, Letter Spacing, Line Height, Word Spacing, and White Space properties:

- text-indent
- letter-spacing
- line-height
- word-spacing
- white-space

Text Indentation

The text-indent property is used to specify the indentation of the first line of a text:

Letter Spacing

Example

```
p {
  text-indent: 50px;
}
```

The letter-spacing property is used to specify the space between the characters in a text.

Line Height

Example

```
h1 {
  letter-spacing: 5px;
}

h2 {
  letter-spacing: -2px;
}
```

The line-height property is used to specify the space between lines:

Word Spacing

Example

```
p.small {
  line-height: 0.8;
}

p.big {
  line-height: 1.8;
}
```

The word-spacing property is used to specify the space between the words in a text.

The following example demonstrates how to increase or decrease the space betweenwords:

Example

```
p.one {
  word-spacing: 10px;
}

p.two {
  word-spacing: -2px;
}
```

White Space

The white-space property specifies how white-space inside an element is handled. This example demonstrates how to disable text wrapping inside an element:

Example

```
p {
  white-space: nowrap;
}
```

The CSS Text Spacing Properties

Property	Description
letter-spacing	**Specifies the space between characters in a text**
line-height	**Specifies the line height**
text-indent	**Specifies the indentation of the first line in a text-block**
white-space	**Specifies how to handle white-space inside an element**
word-spacing	**Specifies the space between words in a text**

5. *CSS Text Shadow*

Text Shadow

The text-shadow property adds shadow to text.

In its simplest use, you only specify the horizontal shadow (2px) and the vertical shadow (2px):

Text shadow effect!

Example

```
h1 {
  text-shadow: 2px 2px;
}
```

Next, add a color (red) to the shadow:

Text shadow effect!

Example

```
h1 {
  text-shadow: 2px 2px red;
}
```

Then, add a blur effect (5px) to the shadow:

Text shadow effect!

Example

```
h1 {
  text-shadow: 2px 2px 5px red;
}
```

IMAGES WITH CSS

1.Thumbnail Images

Use the border property to create thumbnail images.Thumbnail Image:

Example

```
img {
  border: 1px solid #ddd;
  border-radius: 4px;
  padding: 5px;
  width: 150px;
}

<img src="paris.jpg" alt="Paris">
```

Thumbnail Image:

Thumbnail Image as Link:

Example

```
img {
  border: 1px solid #ddd;
  border-radius: 4px;
  padding: 5px;
  width: 150px;
}
img:hover {
  box-shadow: 0 0 2px 1px rgba(0, 140, 186, 0.5);
}
<a href="paris.jpg">
  <img src="paris.jpg" alt="Paris">
</a>
```

Thumbnail Image as Link:

---

1. **Responsive Images:**

The responsive image is used to adjust the image automatically to the specifiedbox.

Example: This example illustrates the use of the **Styling image** property for creating responsive images.

<!DOCTYPE html>

```
<html>
<head>
  <style>
  img {
     max-width: 100%;
     height: auto;
  }
  </style>
</head>
 <body>
  <img src=
"https://sparc.org.in/courses/university/it_courses/mca/ mca-course-in-gtb- nagar-
sparc-academy "
      alt="Responsive-image"
      width="1000"
      height="300">
</body>
</html>
```

Output:

2. Transparent Image:

The opacity property is used to set the image transparent. The opacity value liesbetween 0.0 to 1.0.

Example: This example illustrates the use of the **Styling image** property for creating transparent images.

```
<!DOCTYPE html>
<html>
<head>
  <title>style image</title>
  <style>
  img {
     opacity: 0.5;
  }
  </style>
</head>
 <body>
  <img src=
"https://sparc.org.in/courses/university/it_courses/mca/mca-course-
in-gtb-nagar-sparc-academy"
      alt="Transparent-image"
      width="100%">
</body>
</html>
```

Output:

Supported Browsers: The browsers supported by *Styling Images* are listed below:

- Google Chrome
- Internet Explorer
- Microsoft Edge
- Firefox
- Opera

CSS Selectors

CSS selectors are used to "find" (or select) HTML elements based on their element name, id, class, attribute, and more. We can divide CSS selectors into five categories:

- Simple selectors (select elements based on name, id, class)
- Combinator selectors (select elements based on a specific relationship between them)
- Pseudo-class selectors (select elements based on a certain state)
- Pseudo-elements selectors (select and style a part of an element)
- Attribute selectors (select elements based on an attribute or attribute value)

1. The CSS element Selector

The element selector selects HTML elements based on the element name.

Example

Here, all <p> elements on the page will be center-aligned, with a red text color:

```
p {
  text-align: center;
  color: red;
}
```

2. The CSS id Selector

The id selector uses the id attribute of an HTML element to select aspecific element.

The id of an element is unique within a page, so the id selector is used to select one unique element!

To select an element with a specific id, write a hash (#) character,followed by the id of the element.

Example

The CSS rule below will be applied to the HTML element with id="para1":

```
#para1 {
  text-align: center;
  color: red;
}
```

3. The CSS class Selector

The class selector selects HTML elements with a specific class attribute. To select elements with a specific class, write a period (.) character, followed by the class name.HTML elements can also refer to more than one class.

Example

In this example all HTML elements with class="center" will be red and center-aligned:

```
.center {
  text-align: center;
  color: red;
}
```

Example

In this example the <p> element will be styled according to class="center" and to class="large":

```
<p class="center large">This paragraph refers to two classes.</p>
```

4. The CSS Universal Selector

The universal selector (*) selects all HTML elements on the page.

Example

The CSS rule below will affect every HTML element on the page:

```
* {
  text-align: center;
  color: blue;
}
```

5. The CSS Grouping Selector

The grouping selector selects all the HTML elements with the same style definitions.

Look at the following CSS code (the h1, h2, and p elements have the same style definitions):

It will be better to group the selectors, to minimize the code.

```
h1 {
  text-align: center;
  color: red;
}

h2 {
  text-align: center;
  color: red;
}

p {
  text-align: center;
  color: red;
}
```

To group selectors, separate each selector with a comma.

Example

In this example we have grouped the selectors from the code above:

```
h1, h2, p {
  text-align: center;
  color: red;
}
```

CSS Flexbox

The **flexbox** or flexible box model in CSS is a one-dimensional layout model that has flexible and efficient layouts with distributed spaces among items to control their alignment structure ie., it is a layout model that provides an easy and clean way to arrange items within a container. Flexbox can be useful for creating small-scales layouts & is responsive and mobile-friendly.

Features of flexbox:

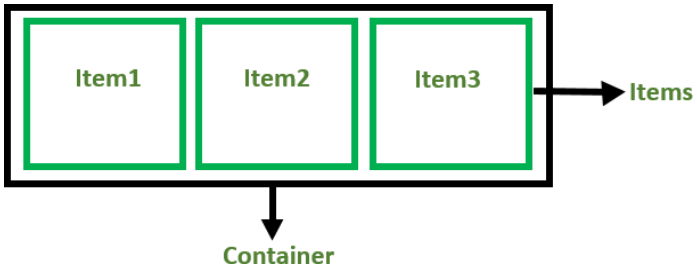
- A lot of flexibility is given.
- Arrangement & alignment of items.
- Proper spacing
- Order & Sequencing of items.
- Bootstrap 4 is built on top of the flex layout.

Before the flexbox model, we had 4 layout modes:

- **Block:** It is used to make sections in web pages.
- **Inline:** It is used for text.
- **Table:** It is used for two-dimensional table data.
- **Positioned:** It is used for the explicit position of an element.

There are 2 main components of the Flexbox:

- **Flex Container**: The parent "div" which contains various divisions is called a flex container.
- **Flex Items**: The items inside the container "div" are flex items.



For creating the flexbox, we need to create a flex container along with setting thedisplay property to flex.

- HTML

```
<!DOCTYPE html>
<html>
 <head>
   <title>Flexbox Tutorial</title>
   <style>
   .flex-container {
      display: flex;
      background-color: #32a852;
   }
    .flex-container div {
      background-color: #c9d1cb;
      margin: 10px;
      padding: 10px;
   }
   </style>
</head>
 <body>
   <h2>COMPUTER SCIENCE</h2>
   <h4> Flexbox</h4>
   <div class="flex-container">
      <div>Item1</div>
      <div>Item2</div>
      <div>Item3</div>
    </div>
 </body>
  </html>
```

Output:

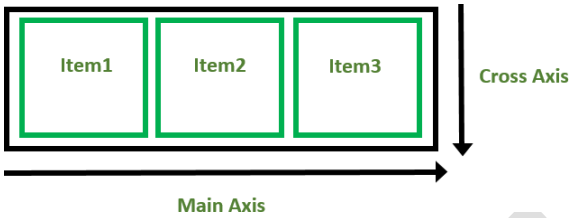
COMPUTER SCIENCE

Flexbox

Item1 Item2 Item3

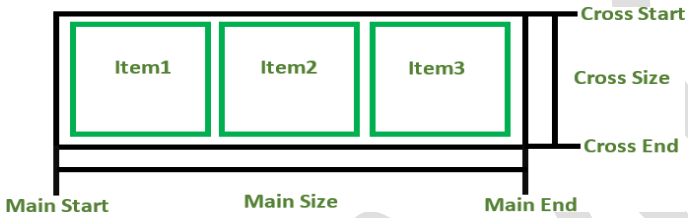
Flexbox Axes: While working with Flexbox, we deal with 2 axes:

- Main Axis
- Cross Axis



Main Axis:

- By default, the main axis runs from left to right.
- **Main Start:** The start of the main axis is called Main Start.
- **Main Size:** The length between Main Start and Main End is called Main Size.
- **Main End:** The endpoint is called Main End.
- Main And Cross Axis



left to right:
flex-direction: row;
right to left:
flex-direction: row-reverse;
top to bottom:
flex-direction: column;
bottom to top:
flex-direction: column;

Cross Axis: The cross axis will be perpendicular to the main axis.

- By default, Cross Axis runs perpendicular to the Main Axis i.e. from top to bottom.
- **Cross Start:** The start of the Cross axis is called Cross Start.
- **Cross Size:** The length between Cross Start and Cross End is called Cross Size.
- **Cross End:** The endpoint is called Cross End.

Supported Browsers:

- Google Chrome 29.0
- Firefox 22.0
- Microsoft Edge 11.0
- Opera 48.0
- Safari 10.0

JavaScript: Data Types and Variables -Functions – Events

1. Datatypes in JavaScript

There are majorly two types of languages. First, one is **Statically typed language** where each variable and expression type is already known at compile time. Once a variable is declared to be of a certain data type, it cannot hold values of other data types.
Example: C, C++, Java.

// Java(Statically typed)
int x = 5 // variable x is of type int and it will not store any other type.string y = 'abc' // type string and will only accept string values

Other, **Dynamically typed languages**: These languages can receive different datatypes over time. For example- Ruby, Python, JavaScript, etc.

// Javascript(Dynamically typed)
var x = 5; // can store an integer
var name = 'string'; // can also store a string.

JavaScript is a dynamically typed (also called loosely typed) scripting language. That is, in JavaScript variables can receive different data types over time. Datatypes are basically typed data that can be used and manipulated in a program.

The latest ECMAScript(ES6) standard defines following data types: Out of which six data types are Primitive(predefined).

- **Numbers**: Represent both integer and floating-point numbers. Example: 5, 6.5, 7etc.
- **String**: A string is a sequence of characters. In JavaScript, strings can be enclosed within the single or double quotes. Example: "Hello GeeksforGeeks" etc.
- **Boolean**: Represent a logical entity and can have two values: true or false.
- **Null**: This type has only one value : *null.*
- **Undefined**: A variable that has not been assigned a value is *undefined.*
- **Symbol:** Unlike other primitive data types, it does not have any literal form. It is a built-in object whose constructor returns a symbol-that is unique.
- **bigint:** The bigint type represents the whole numbers that are larger than 2^{53}-1. Toform a bigint literal number, you append the letter n at the end of the number.
- **Object**: It is the most important data-type and forms the building blocks formodern JavaScript. We will learn about these data types in detail in further articles.

2. Variables in JavaScript:

Variables in JavaScript are containers that hold reusable data. It is thebasic unit of storage in a program.

- The value stored in a variable can be changed during program execution.
- A variable is only a name given to a memory location, all the operations done on the variable effects that memory location.
- In JavaScript, all the variables must be declared before they can be used.

Before ES2015, JavaScript variables were solely declared using the *var* keyword followed by the name of the variable and semi-colon. Below is the syntax to create variables in JavaScript:

***var* var_name;**
var x;

The var_name is the name of the variable which should be defined by the user and should be unique. These types of names are also known as **identifiers**.

The rules for creating an identifier in JavaScript are, the name of the identifier should not be any pre-defined word(known as keywords), the first character must be a letter, an underscore (_), or a dollar sign ($).

Subsequent characters may be any letter or digit or an underscore or dollar sign.

We can initialize the variables either at the time of declaration or also later when we want to use them. Below are some examples of declaring and initializingvariables in JavaScript:

// declaring single variablevar name;
// declaring multiple variables

```
var name, title, num;
// initializing variablesvar name
= "Harsh";
name = "Rakesh";
```

JavaScript is also known as **untyped** language. This means, that once a variable is created in JavaScript using the keyword var, we can store any type of value in this variable supported by JavaScript.

```
// storing a mathematical expressionvar x = 5 +
10 + 1;
console.log(x); // 16
```

After ES2015, we now have two new variable containers: let and const. Now we shall look at both of them one by one. The variable type **Let** shares lots of similarities with var but unlike var, it has scope constraints. To know more about them visit let vs var. Let's make use of the let variable:

```
// let variable let x; //
undefinedlet name =
'Mukul';
// can also declare multiple valueslet
a=1,b=2,c=3;
// assignmentlet a
=3;
a = 4; // works same as var.
```

Variable Scope in Javascript

Scope of a variable is the part of the program from where the variable may directly be accessible.

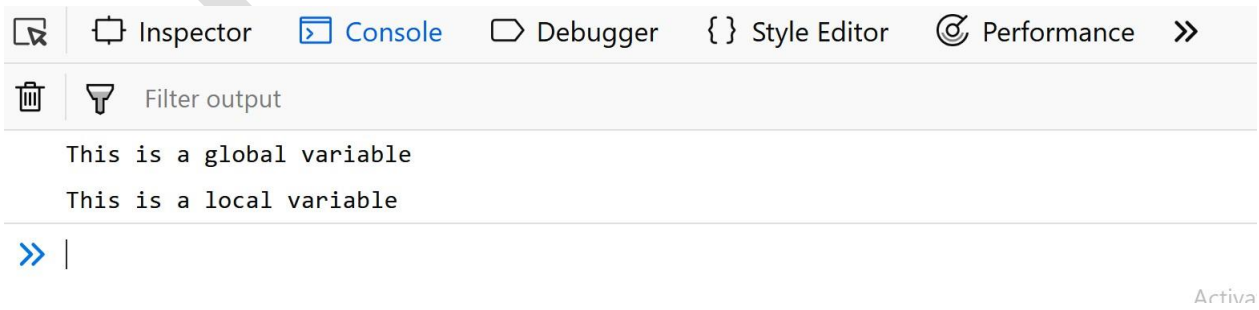
In JavaScript, there are two types of scopes:

1. **Global Scope** – Scope outside the outermost function attached to Window.
2. **Local Scope** – Inside the function being executed.

Let's look at the code below. We have a global variable defined in the first line in the global scope. Then we have a local variable defined inside the function fun().

```
let globalVar = "This is a global variable";
function fun() {
let localVar = "This is a local variable";
console.log(globalVar);
console.log(localVar);
}
fun();
```

OUTPUT:

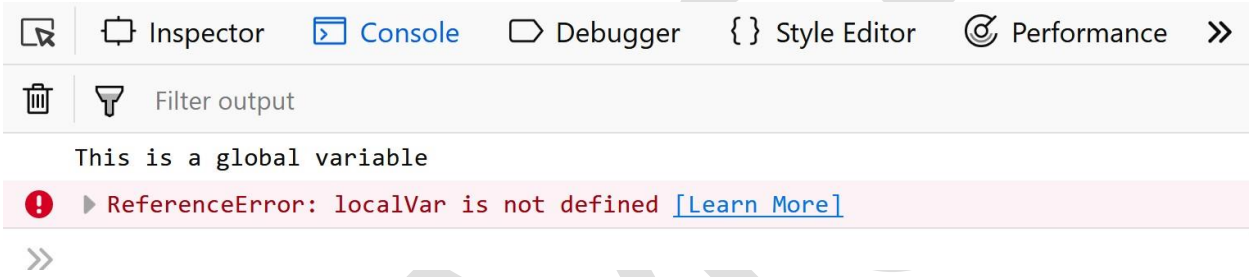


When we execute the function fun(), the output shows that both global, and local variables, are accessible inside the function as we are able to console.log them. This shows that inside the function we have access to both global variables (declared outside the function) and local variables (declared inside the function).

Let's move the console.log statements outside the function and put them just after calling the function.

```
let globalVar = "This is a global variable";
function fun() {
let localVar = "This is a local variable";
}
fun();
console.log(globalVar);
console.log(localVar);
```

OUTPUT:



3. *JavaScript function*

A JavaScript function is a block of code designed to perform a particular task. A JavaScript function is executed when "something" invokes it (calls it).

Example

```
function myFunction(p1, p2) {
  return p1 * p2;   // The function returns the product of p1 and p2
}
```

JavaScript Function Syntax

A JavaScript function is defined with the function keyword, followed by a **name**, followed by parentheses **()**.

Function names can contain letters, digits, underscores, and dollar signs (same rules asvariables).

The parentheses may include parameter names separated by commas: ***(parameter1, parameter2, ...)***

The code to be executed, by the function, is placed inside curly brackets: **{}**

```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

Function **parameters** are listed inside the parentheses () in the function definition.
Function **arguments** are the **values** received by the function when it is invoked.
Inside the function, the arguments (the parameters) behave as local variables.

Function Invocation
The code inside the function will execute when "something" **invokes** (calls) thefunction:

- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self invoked)

Function Return

When JavaScript reaches a return statement, the function will stop executing.

If the function was invoked from a statement, JavaScript will "return" to executethe code after the invoking statement.

Functions often compute a **return value**. The return value is "returned" back tothe "caller":

Example

Calculate the product of two numbers, and return the result:

```
let x = myFunction(4, 3);   // Function is called, return value will end up
in x

function myFunction(a, b) {
  return a * b;             // Function returns the product of a and b
}
```

The result in x will be:

```
12
```

Why Functions?

You can reuse code: Define the code once, and use it many times.

You can use the same code many times with different arguments, to produce differentresults.

Example

Convert Fahrenheit to Celsius:

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}
document.getElementById("demo").innerHTML = toCelsius(77);
```

The () Operator Invokes the Function

Using the example above, toCelsius refers to the function object, and toCelsius() refersto the function result.

Accessing a function without () will return the function object instead of the functionresult.

Example

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}
document.getElementById("demo").innerHTML = toCelsius;
```

Functions Used as Variable Values

Functions can be used the same way as you use variables, in all types of formulas, assignments, and calculations.

Example

Instead of using a variable to store the return value of a function:

```
let x = toCelsius(77);
let text = "The temperature is " + x + " Celsius";
```

You can use the function directly, as a variable value:

```
let text = "The temperature is " + toCelsius(77) + " Celsius";
```

Local Variables

Variables declared within a JavaScript function, become **LOCAL** to the function.
Local variables can only be accessed from within the function.

Example

```
// code here can NOT use carName

function myFunction() {
  let carName = "Volvo";
  // code here CAN use carName
}

// code here can NOT use carName
```

4. JavaScript Events

HTML events are **"things"** that happen to HTML elements.
When JavaScript is used in HTML pages, JavaScript can **"react"** on these events.

HTML Events
An HTML event can be something the browser does, or something a user does.EXAMPLE:

- An HTML web page has finished loading
- An HTML input field was changed
- An HTML button was clicked

Often, when events happen, you may want to do something.
JavaScript lets you execute code when events are detected.
HTML allows event handler attributes, **with JavaScript code**, to be added to HTML elements.

With single quotes:
<element event='some JavaScript'>
With double quotes:
<element event="some JavaScript">
In the following example, an onclick attribute (with code), is added toa <button> element:
In the example above, the JavaScript code changes the content of the element withid="demo".

Example

```
<button onclick="document.getElementById('demo').innerHTML = Date()">The time is?</button>
```

In the next example, the code changes the content of its own element (using **this**.innerHTML):
Common HTML Events

Example

```
<button onclick="this.innerHTML = Date()">The time is?</button>
```

Example

```
<button onclick="displayDate()">The time is?</button>
```

Here is a list of some common HTML events:

Event	Description
Onchange	An HTML element has been changed
Onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

JavaScript Event Handlers

Event handlers can be used to handle and verify user input, user actions, and browseractions:

- Things that should be done every time a page loads
- Things that should be done when the page is closed
- Action that should be performed when a user clicks a button
- Content that should be verified when a user inputs data
- And more ...

Many different methods can be used to let JavaScript work with events:

- HTML event attributes can execute JavaScript code directly
- HTML event attributes can call JavaScript functions

AJAX: GET and POST

The jQuery get() and post() methods are used to request data from the server with anHTTP GET or POST request.

HTTP Request: GET vs. POST

Two commonly used methods for a request-response between a client and server are:GET and POST.

- **GET** - Requests data from a specified resource
- **POST** - Submits data to be processed to a specified resource

GET is basically used for just getting (retrieving) some data from the server. **Note:** TheGET method may return cached data.

POST can also be used to get some data from the server. However, the POST methodNEVER caches data, and is often used to send data along with the request.

jQuery $.get() Method

The $.get() method requests data from the server with an HTTP GET request.

Syntax:

***$.get(*URL,callback);**

The required URL parameter specifies the URL you wish to request.

The optional callback parameter is the name of a function to be executed if the requestsucceeds.

The following example uses the $.get() method to retrieve data from a file on the server:

Example

```
$("button").click(function(){
  $.get("demo_test.asp", function(data, status){
    alert("Data: " + data + "\nStatus: " + status);
  });
});
```

The first parameter of $.get() is the URL we wish to request ("demo_test.asp").

The second parameter is a callback function. The first callback parameter holds the content of the page requested, and the second callback parameter holds the status of therequest.

jQuery $.post() Method

The $.post() method requests data from the server using an HTTP POST request.

Syntax:

$.post(*URL,data,callback*);

- ✓ The required URL parameter specifies the URL you wish to request.
- ✓ The optional data parameter specifies some data to send along with the request.
- ✓ The optional callback parameter is the name of a function to be executed if the request succeeds.

- ✓ The following example uses the $.post() method to send some data along with the request:

Example

```
$("button").click(function(){
  $.post("demo_test_post.asp",
  {
    name: "Donald Duck",
    city: "Duckburg"
  },
  function(data, status){
    alert("Data: " + data + "\nStatus: " + status);
  });
});
```

- ✓ The first parameter of $.post() is the URL we wish to request ("demo_test_post.asp").
- ✓ Then we pass in some data to send along with the request (name and city).
- ✓ The ASP script in "demo_test_post.asp" reads the parameters, processes them, and returns a result.
- ✓ The third parameter is a callback function. The first callback parameter holds the content of the page requested, and the second callback parameter holds the status of the request.

UNIT II

SERVER SIDE PROGRAMMING WITH NODE JS

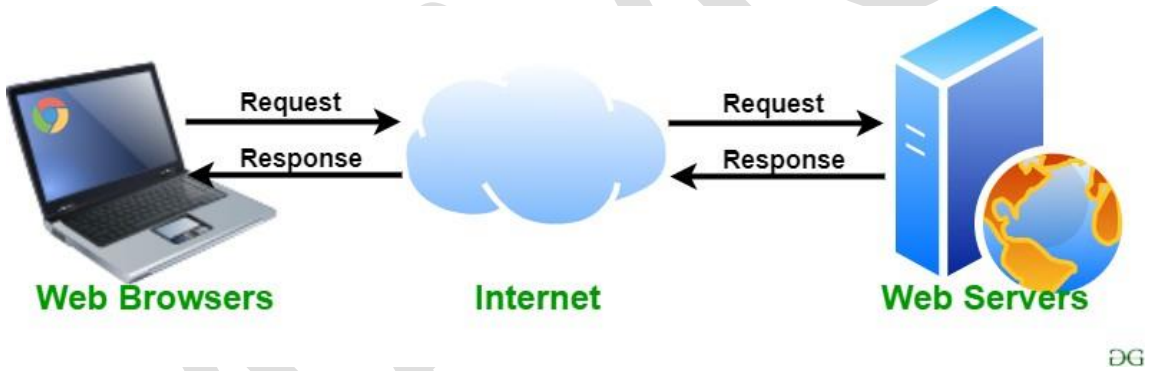
Introduction to Web Servers

Definition: A web server is a computer that runs websites. It's a computer program that distributes web pages as they are requisitioned. The basic objective of the web server is to store, process and deliver web pages to the users.

This intercommunication is done using Hypertext Transfer Protocol (HTTP). These web pages are mostly static content that includes HTML documents, images, style sheets, test etc. Apart from HTTP, a web server also supports SMTP (Simple Mail transfer Protocol) and FTP (File Transfer Protocol) protocol for emailing and for file transfer and storage.

A web server is a software program that serves web pages to web users (browsers).

A web server delivers requested web pages to users who enter the URL in a web browser. Every computer on the internet that contains a web site must have a web server program.



Characteristics of web servers

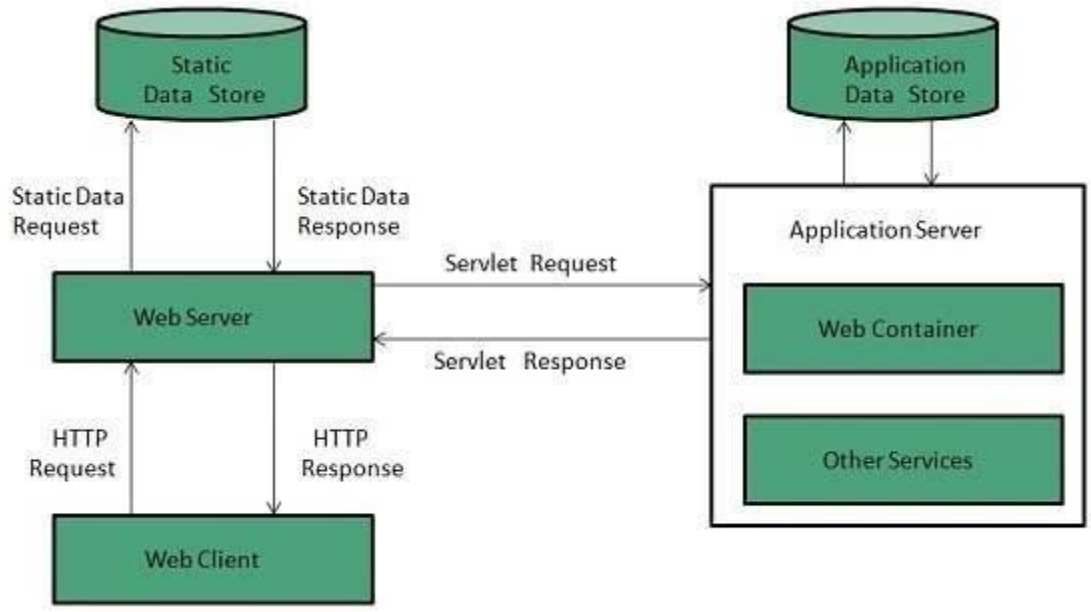
A web server computer is just like any other computer. The basic characteristics of web servers are:

- It is always connected to the internet so that clients can access the web pages hosted by the web server.
- It always has an application called "web server" running.

Web Server Working

Web server respond to the client request in either of the following two ways:

- Sending the file to the client associated with the requested URL.
- Generating response by invoking a script and communicating with database



- When client sends request for a web page, the web server search for the requested page if requested page is found then it will send it to client with an HTTP response.
- If the requested web page is not found, web server will the send an **HTTP response:Error 404 Not found.**
- If client has requested for some other resources then the web server will contact to the application server and data store to construct the HTTP response.

Architecture

Concurrent Approach

Concurrent approach allows the web server to handle multiple client requests at the same time. It can be achieved by following methods:

1. Multi-process
2. Multi-threaded
3. Hybrid method.

1. Multi-processing

In this a single process (parent process) initiates several single-threaded child processes and distribute incoming requests to these child processes. Each of the child processes are responsible for handling single request.

It is the responsibility of parent process to monitor the load and decide if processes should be killed or forked.

2. Multi-threaded

Unlike Multi-process, it creates multiple single-threaded process.

3. Hybrid

It is combination of above two approaches. In this approach multiple process are created and each process initiates multiple threads. Each of the threads handles one connection. Using multiple threads in single process results in less load on system resources.

There are many web servers available in the market both free and paid. Some of them are described below:

- **Apache HTTP server:** The Apache HTTP web server was developed by the Apache Software Foundation. It is an open-source software which means that we can access and make changes to its code and mold it according to our preference.
  The Apache Web Server can be installed and operated easily on almost all operating systems like Linux, MacOS, Windows, etc.

- **Microsoft Internet Information Services (IIS):** IIS (Internet Information Services) is a high performing web server developed by Microsoft. It is strongly united with the operating system and is therefore relatively easier to administer.
  It has all the features of the Apache HTTP Server except that it is not an open-source software and therefore its code is inaccessible which means that we cannot make changes in the code to suit our needs. It can be easily installed in any Windows device.

Javascript in the Desktop with NodeJS

Node.js is known for being a server-side platform, interest in using it for building desktop applications is growing rapidly. Bindings exist for GUI toolkits such as GTK+, QT and Cocoa. However, one of the advantages of using Node.js for web development is the ability to use the same language on both the server and client.

It's possible to use Node.js together with a web browser shell to develop cross-platform desktop applications–and games using WebGL–using HTML, CSS and JavaScript.

The Contenders

Chrome Applications

The most basic way of running a "desktop" application is to run a local server and use the web browser as the user interface. Chrome's command line exposes an extra flag to run itself in "application" mode, hiding everything but the web view.

One example of this is Morkdown, a cross-platform application to edit GitHub Flavoured Markdown. It starts a Node.js HTTP server locally on a random port, then launches Chrome using `'--app=http://localhost:' + port` as a command-line flag.

There are a few downsides to this approach. To be able to use the application, the user will need to have both Node.js and Chrome (or Chromium) installed on their computer. Ideally, desktop applications should be self-contained, allowing the user to launch it and use it straight away without runtime pre-requisites.

Chrome applications don't feel entirely like desktop applications. Outside of the web view, operating-system-specific features and UI items can't be modified, and it isn't possible to brand the application (it will appear as another Chrome window).

Since the application is running in Chrome, users still have access to regular web browser features, and can open new windows and tabs and even the Chrome Developer Tools.

The need to have a server running to access the Node.js APIs means having two parts to the application: a HTTP API or WebSocket interface for the browser to talk to, and a web frontend to communicate with the server. This results in an undesirable layer whereby you have to write the server and the client separately, without the ability to run Node.js within the frontend.

node-webkit

node-webkit is a web browser shell built on Chromium, allowing for the Node.js API to be used alongside the DOM API within the same context. As a basic example, you could replace the contents of `body` with a file read via `fs.readFile` in a `script` tag like so:

https://github.com/rvagg/morkdown

It's also possible to use modules from npm and require them the exact same way. Native addons are also supported to extend both node and node-webkit, however they must be built using nw-gyp. node-webkit comes with a library to manipulate external parts of the shell, including the menu bar, tray icons and clipboard.

node-webkit applications are configured via a `window` key in `package.json`, outlining various properties of the application such as the entry document, width and height amongst others.

node-webkit has a major advantage to Chrome applications, as both DOM manipulation and Node.js API calls can be used in the same context without needing to run a separate server.

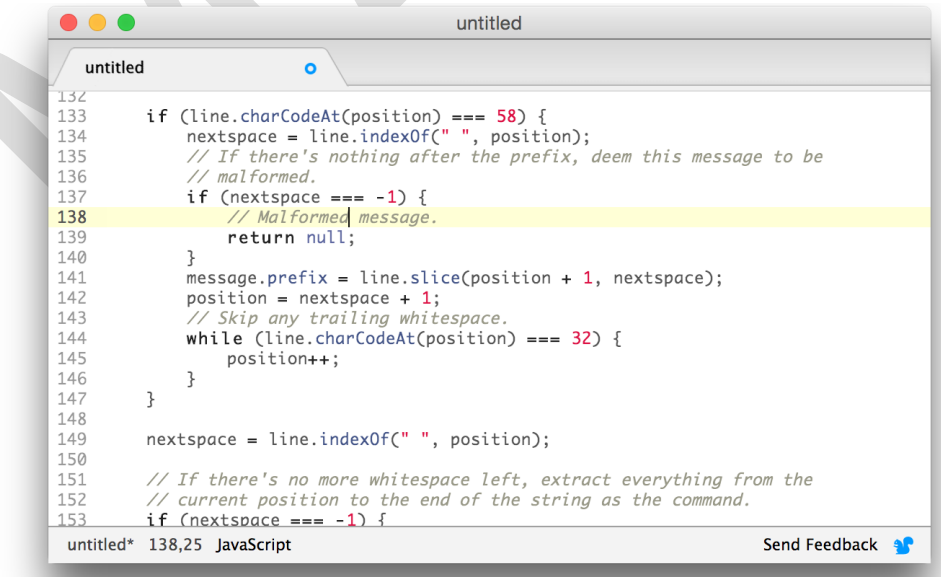
The only caveat to this is that modules pulled into the application via `require` only have access to the Node.js context, not the web view's. This means that Node.js modules must exclusively use functions and classes provided by Node.js or modules from npm, as the DOM is off limits. To get around this, you can include JavaScript using `script` tags.

atom-shell

atom-shell–similarl to node-webkit–is a shell built using components from Chromium. Needing the entire Chromium source, atom-shell only builds libchromiumcontent.

Building atom-shell is much faster than building node-webkit as a result. atom-shell uses an alternative method of integrating libuv with Chromium's event loop, as well as using an upcoming feature in Node.js 0.12, multi-context.

There haven't been many large applications built with atom-shell apart from Atom itself, however atom-shell is fully documented.



The distinction between the browser shell and its runtime allow for cleaner organisation of code that deals with application state management and the logic needed to power the UI, compared with node-webkit.

Unlike node-webkit, application configuration is done via an entry script rather than an object in package.json. As a result, atom-shell is more flexible than node-webkit for application state customisation at startup.

Unlike node-webkit, atom-shell uses APIs exposed by libchromiumcontent instead of modifying Chromium directly, allowing easier upgrading of the renderer. This results in new Blink features being brought into atom-shell at a quicker pace than node-webkit.

Pros

- Developing desktop applications using HTML, CSS and JavaScript, as a web developer, allows you to quickly push out functional, cross-platform applications using the frontend frameworks and npm modules you already know
- Access to the latest web technologies available in Blink
- Easy to package the application for end users
- If you already have an remote web application, you can already reuse most of that codebase to build a desktop application

Cons

- When packaging applications using either shell, the resulting executable contains an almost complete version of Chromium and Node.js as well as your HTML, CSS and JavaScript.
- Depending on the target system, the entire packaged application can become almost 100Mb, whilst the size of an application using native UI libraries can start at a mere few kilobytes in size.
- Compared to native applications, desktop web applications typically require a much larger amount of RAM and CPU power to run and render

NPM

NPM is the world's largest **Software Registry**.

The registry contains over 800,000 **code packages**.

Open-source developers use **npm** to **share** software.

Many organizations also use npm to manage private development.

npm is two things: first and foremost, it is an online repository for the publishing of open-source Node.js projects; second, it is a command-line utility for interacting with said repository that aids in package installation, version management, and dependency management. A plethora of Node.js libraries and applications are published on npm, and many more are added every day. These applications can be searched for on https://www.npmjs.com/. Once you have a package you want to install, it can be installed with a single command-line command.

npm is free to use.

You can download all npm public software packages without any registration or logon.

Command Line Client

npm includes a **CLI** (Command Line Client) that can be used to download and install software:

Windows Example

C:\>npm install <package>

Mac OS Example

>npm install <package>

Installing npm

npm is installed with **Node.js**

This means that you have to install Node.js to get npm installed on your computer.

Download Node.js from the official Node.js web site: https://nodejs.org

Software Package Manager

The name **npm** (Node Package Manager) stems from when npm first was created as a package manager for Node.js.

All **npm** packages are defined in files called **package.json**. The content of package.json must be written in **JSON**. At least two fields must be present in the definition file: **name** and **version**.

Example

```
{
"name" : "foo",
"version" : "1.2.3",
"description" : "A package for fooing things",
"main" : "foo.js",
"keywords" : ["foo", "fool", "foolish"],
"author" : "John Doe",
"licence" : "ISC"
}
```

Managing Dependencies

npm can manage **dependencies**.**npm** can (in one command line) install all the dependencies of a project.Dependencies are also defined in **package.json**.

Sharing Your Software

If you want to share your own software in the **npm registry**, you can sign in at:

https://www.npmjs.com

Publishing a Package

You can publish **any directory** from your computer as long as the directory has a **package.json file**.

Check if npm is installed:

```
C:\>npm
```

Check if you are logged in:

```
C:\>npm whoami
```

If not, log in:

```
C:\>npm login
Username: <your username>
Password: <your password>
```

Navigate to your project and publish your project:

```
C:\Users\myuser>cd myproject
C:\Users\myuser\myproject>npm publish
```

Serving files with the http module

A basic necessity for most http servers is to be able to serve static files. It is not that hard to do in Node.js. First you read the file, then you serve the file. Here is an example of a script that will serve the files in the current directory:

```
var fs = require('fs'),
    http = require('http');
http.createServer(function (req, res) {
  fs.readFile(__dirname + req.url, function (err,data) {
  if (err) {
     res.writeHead(404);
     res.end(JSON.stringify(err));
     return;
    }
    res.writeHead(200);
    res.end(data);
  });
}).listen(8080);
```

This example takes the path requested and it serves that path, relative to the local directory. This works fine as a quick solution; however, there are a few problems with this approach.

First, this code does not correctly handle mime types. Additionally, a proper static file server should really be taking advantage of client side caching, and should send a "Not Modified" response if nothing has changed. Furthermore, there are security bugs that can enable a malicious user to break out of the current directory. (for example, GET /../../../).

```
var static = require('node-static');
var http = require('http');
var file = new(static.Server)(__dirname);
http.createServer(function (req, res) {
file.serve(req, res);
}).listen(8080);
```

Introduction to the Express framework

What is Express?

Express is a small framework that sits on top of Node.js's web server functionality to simplify its APIs and add helpful new features.It makes it easier to organize your application's functionality with middle ware and routing; it adds helpful utilities to Node.js's HTTP objects;it facilitates the rendering of dynamic HTTP objects.

Express is a part of **MEAN** stack, a full stack JavaScript solution used in building fast, robust, and maintainable production web applications.

MongoDB(Database)
ExpressJS(Web Framework)
AngularJS(Front-end Framework)
NodeJS(Application Server)

Node. js - Express Framework

1. Allows to set up middlewares to respond to HTTP Requests.
2. Defines a routing table which is used to perform different actions based on HTTP Method and URL.
3. Allows to dynamically render HTML Pages based on passing arguments to templates.

Express is a minimal and flexible Node.js web application framework that provides a robust set of features to develop web and mobile applications. It facilitates the rapid development of Node based Web applications.

Installing Express

Firstly, install the Express framework globally using NPM so that it can be used to create a web application using node terminal.

$ npm install express --save

The above command saves the installation locally in the **node_modules** directory and creates a directory express inside node_modules. You should install the following important modules along with express –

- **body-parser** – This is a node.js middleware for handling JSON, Raw, Text and URL encoded form data.

- **cookie-parser** – Parse Cookie header and populate req.cookies with an object keyed by the cookie names.
- **multer** – This is a node.js middleware for handling multipart/form-data.

$ npm install body-parser --save
$ npm install cookie-parser --save
$ npm install multer –save

Installing Express on Windows (WINDOWS 10)

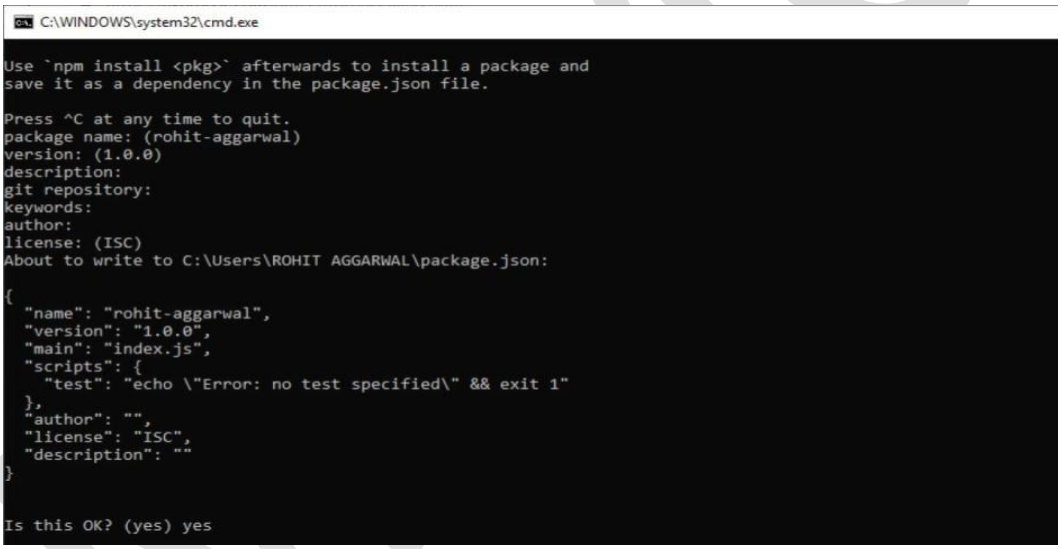
Assuming that you have installed node.js on your system, the following steps should be followed to install express on your Windows:

STEP-1: Creating a directory for our project and make that our working directory.
$ mkdir gfg

$ cd gfg

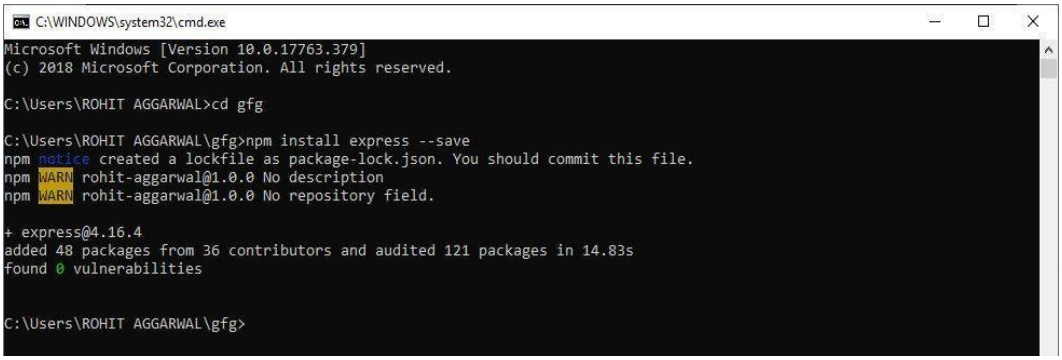
STEP-2: Using npm init command to create a package.json file for our project.
$ npm init

This command describes all the dependencies of our project. The file will be updated when adding further dependencies during the development process, for example when you set up your build system.



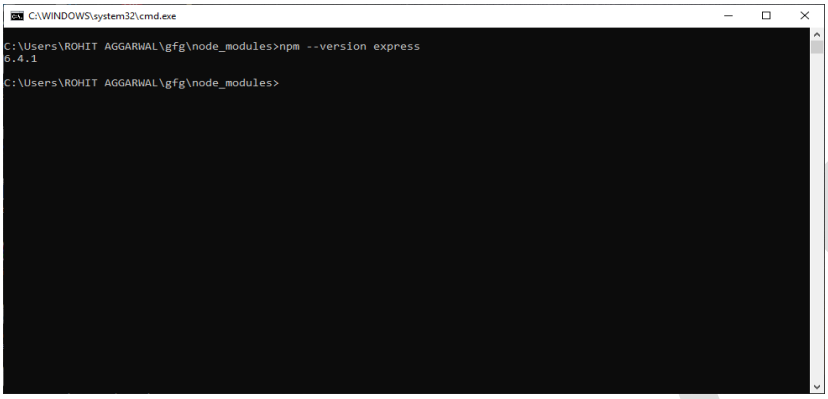
Keep pressing enter and enter “yes/no” accordingly at the terminus line.

STEP-3: Installing Express
Now in your *gfg(name of your folder)* folder type the following command line:
$ npm install express --save



NOTE- Here ***"WARN"*** indicates the fields that must be entered in STEP-2.
STEP-4: Verify that Express.js was installed on your Windows:
To check that express.js was installed on your system or not, you can run the following command line on cmd:

C:\Users\Admin\gfg\node_modules>npm --version express



The version of express.js will be displayed on successful installation.

Hello world Example

Following is a very basic Express app which starts a server and listens on port 8081 for connection. This app responds with **Hello World!** for requests to the homepage. For every other path, it will respond with a **404 Not Found.**

```
var express = require('express');
var app = express();
app.get('/', function (req, res) {
  res.send('Hello World');
})
var server = app.listen(8081, function () {
  var host = server.address().address
  var port = server.address().port
  console.log("Example app listening at http://%s:%s", host, port)
})
```

Save the above code in a file named server.js and run it with the following command.

$ node server.js

You will see the following output –

Example app listening at http://0.0.0.0:8081

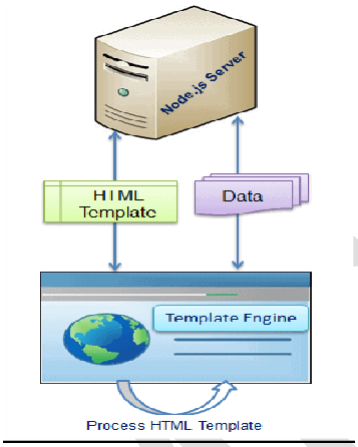
Open http://127.0.0.1:8081/ in any browser to see the following result.

Server-side rendering with Templating Engines

Template Engines for Node.js

Template engine helps us to create an HTML template with minimal code. Also, it can inject data into HTML template at client side and produce the final HTML.

The following figure illustrates how template engine works in Node.js.



Template Engine

As per the above figure, client-side browser loads HTML template, JSON/XML data and template engine library from the server. Template engine produces the final HTML using template and data in client's browser. However, some HTML templates process data and generate final HTML page at server side also.

There are many template engines available for Node.js. Each template engine uses a different language to define HTML template and inject data into it.

The following is a list of important template engines for Node.js

- ***Jade***
- ***Vash***
- ***EJS***
- ***Mustache***
- ***Dust.js***
- ***Nunjucks***
- ***Handlebars***
- ***atpl***
- ***haml***

Advantages of Template engine in Node.js

1. Improves developer's productivity.
2. Improves readability and maintainability.
3. Faster performance.
4. Maximizes client side processing.
5. Single template for multiple pages.
6. Templates can be accessed from CDN (Content Delivery Network).

Server-side rendering (SSR) is a popular technique for rendering a normally client-side only single page app (SPA) on the server and then sending a fully rendered page to the client.

The client's JavaScript bundle can then take over and the SPA can operate as normal.

SSR technique is helpful in situations like the client has a slow internet connection and then rendering of the whole page on client-side takes too much time in certain situations Server Side Rendering might come as handy. One of the widely used modules used to do Server Side Rendering in Node.js is EJS Module. EJS stands for **Embedded JavaScript template.**

Feature of EJS Module:

1. ***Use plain javascript.***
2. ***Fast Development time.***
3. ***Simple syntax.***
4. ***Faster execution.***
5. ***Easy Debugging.***
6. ***Active Development.***

Installation of request module:

1. First of all install express js and ejs using npm install. You also can visit this link to know more about EJS.

npm install ejs

2. The require() method is used to load and cache JavaScript modules.

const ejs = require('ejs');

3. Next step is to create a folder and add a file name app.js and a file named index.ejs. Be careful, about syntax of index file, here it is ejs which denotes it is an ejs file.

node app.js

Render file using EJS renderFIle() method

To perform Server Side Rendering we use renderFile() method of the ejs module, which helps us to render the ejs file on the server-side.

Syntax:

ejs.renderFile(fileName, { }, { }, callback);

Here, callback function takes two arguments first is an error (if there is an error occurs then the renderFile returns an error) and on successful rendering it returns a template.

Filename: app.js

```
JAVASCRIPT
// Requiring modules
const express = require('express');
const app = express();
const ejs = require('ejs');
var fs = require('fs');
const port = 8000;

// Render index.ejs file
app.get('/', function (req, res) {
        // Render page using renderFile method
        ejs.renderFile('index.ejs', {},
                {}, function (err, template) {
                if (err) {
                        throw err;
                } else {
                        res.end(template);}});
                        });
// Server setup
        app.listen(port, function (error) {
                if (error)
                        throw error;
                        else
                         console.log("server id running");

                        });
```

Filename: index.ejsHTML

```
        <!DOCTYPE html>
        <html lang="en">
        <head>
                <meta charset="UTF-8">
                <meta name="viewport" content=
                        "width=device-width, initial-scale=1.0">
        </head>
        <body>
                <h1>Hello World</h1>
        </body></html>
```

Steps to run the program:

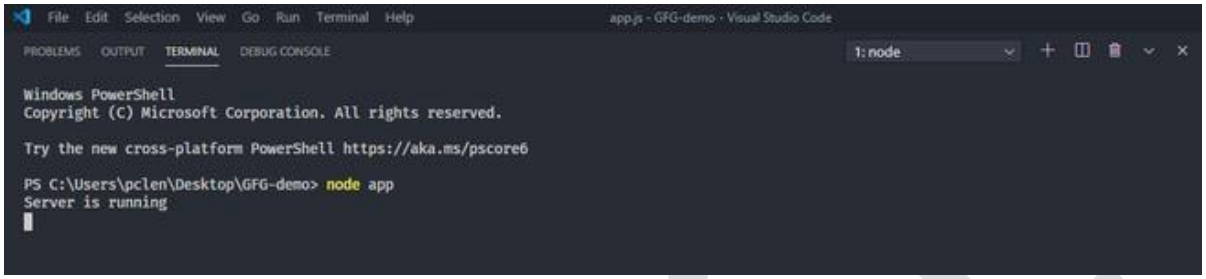
1. Folder Structure:

2. Make sure you have installed the express and request module using the following commands:

npm install express

npm install ejs

3. Run **app.js** using the below command:
node app.js



Starting Node Server

4. Now type *localhost:8000* in your browser to display the ejs page to see the below result:

Static Files - async/await

"async and await make promises easier to write"

async makes a function return a Promise

await makes a function wait for a Promise

Async Syntax

The keyword async before a function makes the function return a promise

EXAMPLE:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript async / await</h2>
<p id="demo"></p>
<script>
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}
```

```
async function myFunction() {return "MCA Students";}
myFunction().then(
 function(value) {myDisplayer(value);},
 function(error) {myDisplayer(error);}
);</script></body></html>
```

OUTPUT:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript async / await</h2>
<p id="demo"></p>
<script>
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}
async function myFunction() {return "MCA Students";}
myFunction().then(
  function(value) {myDisplayer(value);},
  function(error) {myDisplayer(error);}
);</script>
</body>
</html>
```

JavaScript async / await

MCA Students

Await Syntax

The keyword await before a function makes the function wait for a promise:

let value = await promise;

The await keyword can only be used inside an async function.

EXAMPLE:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript async / await</h2>
<p>Wait 3 seconds (3000 milliseconds) for this page to change.</p>
<h1 id="demo"></h1>
<script>
async function myDisplay() {
 let myPromise = new Promise(function(resolve) {

   setTimeout(function() {resolve(" I MCA Students :)");}, 3000);

 });
 document.getElementById("demo").innerHTML = await myPromise;
}
myDisplay();
</script></body></html>
```

OUTPUT:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript async / await</h2>
<p>Wait 3 seconds (3000 milliseconds) for this page to
change.</p>
<h1 id="demo"></h1>
<script>
async function myDisplay() {
  let myPromise = new Promise(function(resolve) {
    setTimeout(function() {resolve(" I MCA Students :)");},
3000);
  });
  document.getElementById("demo").innerHTML = await
myPromise;
}
myDisplay();
</script></body></html>
```

JavaScript async / await

Wait 3 seconds (3000 milliseconds) for this page to change.

I MCA Students :)

Fetching JSON from Express

What does JSON () do in Express?

json() is a built-in middleware function in Express. This method is used to **parse the incoming requests with JSON payloads** and is based upon the bodyparser. This method returns the middleware that only parses JSON and only looks at the requests where the content-type header matches the type option.

When you listen for connections on a route in Express, the callback function will be invoked on every network call with a Request object instance and a Response object instance.

Example:

Here we used the Response.send() method, which accepts any string.

```
app.get('/', (req, res) => res.send('Hello World!'))
```

You can send JSON to the client by using Response.json(), a useful method.

It accepts an object or array, and converts it to JSON before sending it:

```
res.json({ username: 'Flavio' })
```

"how to get json data from json file in node js" Code Answer's

read json file node js

```
function readJsonFile(file) {
    let bufferData = fs.readFileSync(file)
    let stData = bufferData.toString()
    let data = JSON.parse(stData)
    return data
}
```

how to get json data from json file in node js

```
const fs = require("fs");
var posts = [];

fs.readFile('./data/posts.json', 'utf8', (err, data) => {
                if (err) throw err;
                posts = JSON.parse(data);
            });
```

UNIT III

ADVANCED NODE JS AND DATABASE

Introduction to NoSQL databases

A **NoSQL** originally referring to non SQL or non relational is a database that provides a mechanism for storage and retrieval of data. This data is modeled in means other than the tabular relations used in relational databases.

NoSQL databases are used in real-time web applications and big data and their use are increasing over time.

A NoSQL database includes simplicity of design, simpler horizontal scaling to clusters of machines and finer control over availability. The data structures used by NoSQL databases are different from those used by default in relational databases which makes some operations faster in NoSQL.

The suitability of a given NoSQL database depends on the problem it should solve. Data structures used by NoSQL databases are sometimes also viewed as more flexible than relational database tables.

Many NoSQL stores compromise consistency in favor of availability, speed and partition tolerance. Barriers to the greater adoption of NoSQL stores include the use of low-level query languages, lack of standardized interfaces, and huge previous investments in existing relational databases.

Most NoSQL stores lack true ACID(Atomicity, Consistency, Isolation, Durability) transactions but a few databases, such as MarkLogic, Aerospike, FairCom c-treeACE, Google Spanner (though technically a NewSQL database), Symas LMDB, and OrientDB have made them central to their designs.

Advantages of NoSQL:

There are many advantages of working with NoSQL databases such as MongoDB and Cassandra. The main advantages are high scalability and high availability.

1. High scalability –

- ✓ NoSQL database use sharding for horizontal scaling. Partitioning of data and placing it on multiple machines in such a way that the order of the data is preserved is sharding.
- ✓ Vertical scaling means adding more resources to the existing machine whereas horizontal scaling means adding more machines to handle the data.
- ✓ Vertical scaling is not that easy to implement but horizontal scaling is easy to implement.
- ✓ Examples of horizontal scaling databases are MongoDB, Cassandra etc.

2. High availability –

Auto replication feature in NoSQL databases makes it highly available because in case of any failure data replicates itself to the previous consistent state.

Disadvantages of NoSQL:

1. **Narrow focus –**
   NoSQL databases have very narrow focus as it is mainly designed for storage but it provides very little functionality. Relational databases are a better choice in the field of Transaction Management than NoSQL.
2. **Open-source –**
   NoSQL is open-source database. There is no reliable standard for NoSQL yet.
3. **Management challenge –**
   Data management in NoSQL is much more complex than a relational database. NoSQL, in particular, has a reputation for being challenging to install and even more hectic to manage on a daily basis.
4. **GUI is not available –**
   GUI mode tools to access the database is not flexibly available in the market.
5. **Backup –**
   Backup is a great weak point for some NoSQL databases like MongoDB. MongoDB has no approach for the backup of data in a consistent manner.
6. **Large document size –**
   Some database systems like MongoDB and CouchDB store data in JSON format. Which means that documents are quite large (BigData, network bandwidth, speed), and having descriptive key names actually hurts, since they increase the document size.

Types of NoSQL database:

1. MongoDB falls in the category of NoSQL document based database.
2. **Key value store:** Memcached, Redis, Coherence
3. **Tabular:** Hbase, Big Table, Accumulo
4. **Document based:** MongoDB, CouchDB, Cloudant
5. RDBMS (Relational Database Management System)
6. OLAP (Online Analytical Processing)
7. NoSQL (recently developed database)

MongoDB system overview

MongoDB is based on a NoSQL database that is used for storing data in a key-value pair. Its working is based on the concept of document and collection. It is also an open-source, a document-oriented, cross-platform database system that is written using C++.

Mongo DB can be defined as a document-oriented database system that uses the concept of NoSQL. It also provides high availability, high performance, along with automatic scaling.

What is a Database?

In MongoDB, a database can be defined as a physical container for collections of data. Here, on the file system, every database has its collection of files residing. Usually, a MongoDB server contains numerous databases.

What are Collections?

Collections can be defined as a cluster of MongoDB documents that exist within a single database. MongoDB collections do not implement the concept of schema. Documents that have collection usually contain different fields. Typically, all the documents residing within a collection are meant for a comparable or related purpose.

What is a Document?

A document can be defined as a collection of key-value pairs that contain dynamic schema. Dynamic schema is something that documents of the equal collection do not require for having the same collection of fields or construction, and a common field is capable of holding various types of data.

Here is a table showing the relation between the terminologies used in RDBMS and MongoDB:

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple or Row	Document
Column	Field
Table Join	Embedded Documents
Primary Key	Primary key / Default key
Mysqld / Oracle	Mongod

Popular Organizations That Use MongoDB

- Adobe
- McAfee
- LinkedIn
- FourSquare
- MetLife
- eBay

Why Use MongoDB?

- Document-Oriented data storage, i.e., data, is stored in a JSON style format, which increases the readability of data as well.
- Replication and high availability of data.
- MongoDB provides Auto-sharding.
- Ad hoc queries are supported by MongoDB, which helps in searching by range queries, field, or using regex terms.
- Indexing of values can be used to create and improve the overall search performance in MongoDB. MongoDB allows any field to be indexed within a document.
- MongoDB has a rich collection of queries.
- Updating of data can be done at a faster pace.
- It can be integrated with other popular programming languages also to handle structured as well as unstructured data within various types of applications.

Advantages of Using MongoDB

- It is easy to set up, i.e., install the MongoDB.
- Since MongoDB is a schema-less database, so there is no hassle of schema migration.
- Since it is a document-oriented language, document queries are used, which plays a vital role in supporting dynamic queries.
- Easily scalable.
- It is easy to have a performance tuning as compared to other relational databases.
- It helps in providing fast accessing of data because of its nature of implementing the internal memory to store the data.
- MongoDB is also used as a file system that can help in easy management of load balancing.
- MongoDB also supports the searching using the concept of regex (regular expression) as well as fields.
- Users can run MongoDB as a windows service also.
- It does not require any VM to run on different platforms.
- It also supports sharding of data.

Basic querying with MongoDB shell

A query in a database system is a command that is used for extracting data from a database and display it in a readable form. Every query associated with the database system is associated with any particular language (such as SQL for structured data, MongoDB for unstructured data).

Methods for Performing Queries in MongoDB

1. **The find() method:** This method is used for querying data from a MongoDB collection.
   The basic syntax for using this method is:
   Syntax:
   db.collection_name.find()
   Example:
   db.writers.find()
2. **The pretty() method:** This method is used for giving a proper format to the output extracted by the query.
   The basic syntax for using this method is:
   Syntax:
   db.collection_name.find().pretty()
   Example:
   db.writers.find().pretty()

Here is how they can be implemented:

Filtering Criteria in MongoDB Queries

It is also possible to filter your results by giving or adding some specific criteria in which you are interested to. For example, if you wish to see the Gaurav Mandes data, you can add a specific attribute to the find() to fetch the data of Gaurav Mandes from that particular database.

Example:
db.writers.find({ author: "Gaurav Mandes" })

Output:

```
> db.writers.find( { author: "Gaurav Mandes" } )
{ "_id" : ObjectId("5d16f6a1e198c897a4105d0d"), "title" : "Networking", "descriptio
n" : "Networking Essentials", "author" : "Gaurav Mandes" }
{ "_id" : ObjectId("5d16f6a1e198c897a4105d0e"), "title" : "Game Engineering", "desc
ription" : "Game Engineering and Development", "author" : "Gaurav Mandes" }
>
```

MongoDB Query Which Specify "AND" Condition

MongoDB also allows you in specifying data values of the documents holding two or more specified values to be fetched from the query. Here are two examples showing the use of specifying queries using AND.

Example:
db.writers.find({ tools: "Visual Studio", born: 1948})

MongoDB Query Which Specify "OR" Condition

MongoDB allows users to specify either one or multiple values to be true. According to this, till one of the conditions is true, the document data will get returned. Here is an example showing the use of OR condition:

Example:
db.musicians.find({$or: [{ instrument: "Drums" }, { born: 1945 }] })

Output:

```
> db.musicians.find({$or: [ { instrument: "Drums" }, { born: 1945 } ]})
{ "_id" : 2, "name" : "Ian Paice", "instrument" : "Drums", "born" : 1948 }
{ "_id" : 3, "name" : "Roger Glover", "instrument" : "Bass", "born" : 1945 }
{ "_id" : 7, "name" : "Jeff Burrows", "instrument" : "Drums", "born" : 1968 }
>
```

$in operator

The $in operator is another special operator used in queries for providing a list of values in the query. When your document holds any of those provided values, it gets returned. Here is an example:

Example:
db.musicians.find({ "instrument": { $in: ["Keyboards", "Bass"] } })

Output:

```
> db.musicians.find( { "instrument": { $in: [ "Keyboards", "Bass" ] } } )
{ "_id" : 3, "name" : "Roger Glover", "instrument" : "Bass", "born" : 1945 }
{ "_id" : 5, "name" : "Don Airey", "instrument" : "Keyboards", "born" : 1948 }
>
```

Using $explain

The **$explain** operator provides information on the query, indexes used in a query and other statistics. It is very useful when analyzing how well your indexes are optimized.

```
>db.users.find({gender:"M"},{user_name:1,_id:0}).explain()
```

Using $hint

The **$hint** operator forces the query optimizer to use the specified index to run a query. This is particularly useful when you want to test performance of a query with different indexes. For example, the following query specifies the index on fields **gender** and **user_name** to be used for this query –

```
>db.users.find({gender:"M"},{user_name:1,_id:0}).hint({gender:1,user_name:1})
{ "user_name" : "tombenzamin" }
```

Request body parsing in Express

- Node.js body parsing middleware.
- Parse incoming request bodies in a middleware before your handlers, available under the req.body property.
- **Note** As req.body's shape is based on user-controlled input, all properties and values in this object are untrusted and should be validated before trusting.
- For example, req.body.foo.toString() may fail in multiple ways, for example the foo property may not be there or may not be a string, and toString may not be a function and instead a string or other user input.

This does not handle multipart bodies, due to their complex and typically large nature. For multipart bodies, you may be interested in the following modules:

- ***busboy and connect-busboy***
- ***multiparty and connect-multiparty***
- ***formidable***
- ***multer***

This module provides the following parsers:

- ***JSON body parser***
- ***Raw body parser***
- ***Text body parser***
- ***URL-encoded form body parser***

Other body parsers you might be interested in:

- ***body***
- ***co-body***

Installation

```
$ npm install body-parser
```

API

- var bodyParser = require('body-parser')
- The bodyParser object exposes various factories to create middlewares.
- All middlewares will populate the req.body property with the parsed body when the Content-Type request header matches the type option, or an empty object ({}) if there was no body to parse, the Content-Type was not matched, or an error occurred.
- The various errors returned by this module are described in the errors section.

bodyParser.json([options])

- Returns middleware that only parses json and only looks at requests where the Content-Type header matches the type option.
- This parser accepts any Unicode encoding of the body and supports automatic inflation of gzip and deflate encodings.
- A new body object containing the parsed data is populated on the request object after the middleware (i.e. req.body).

Options

The json function takes an optional options object that may contain any of the following keys:

inflate

When set to true, then deflated (compressed) bodies will be inflated; when false, deflated bodies are rejected. Defaults to true.

limit

Controls the maximum request body size. If this is a number, then the value specifies the number of bytes; if it is a string, the value is passed to the bytes library for parsing. Defaults to '100kb'.

reviver

The reviver option is passed directly to JSON.parse as the second argument.

strict

When set to true, will only accept arrays and objects; when false will accept anything JSON.parse accepts. Defaults to true.

type

The type option is used to determine what media type the middleware will parse. This option can be a string, array of strings, or a function. If not a function, type option is passed directly to the type-is library and this can be an extension name (like json), a mime type (like application/json), or a mime type with a wildcard (like */* or */json). If a function, the type option is called as fn(req) and the request is parsed if it returns a truthy value. Defaults to application/json.

verify

The verify option, if supplied, is called as verify(req, res, buf, encoding), where buf is a Buffer of the raw request body and encoding is the encoding of the request. The parsing can be aborted by throwing an error.

bodyParser.raw([options])

- Returns middleware that parses all bodies as a Buffer and only looks at requests where the Content-Type header matches the type option.
- This parser supports automatic inflation of gzip and deflate encodings.
- A new body object containing the parsed data is populated on the request object after the middleware (i.e. req.body). This will be a Buffer object of the body.

Options

The raw function takes an optional options object that may contain any of the following keys:

inflate

When set to true, then deflated (compressed) bodies will be inflated; when false, deflated bodies are rejected. Defaults to true.

limit

Controls the maximum request body size. If this is a number, then the value specifies the number of bytes; if it is a string, the value is passed to the bytes library for parsing. Defaults to '100kb'.

type

The type option is used to determine what media type the middleware will parse. This option can be a string, array of strings, or a function. If not a function, type option is passed directly to the type-is library and this can be an extension name (like bin), a mime type (like application/octet-stream), or a mime type with a wildcard (like */* or application/*). If a function, the type option is called as fn(req) and the request is parsed if it returns a truthy value. Defaults to application/octet-stream.

verify

The verify option, if supplied, is called as verify(req, res, buf, encoding), where buf is a Buffer of the raw request body and encoding is the encoding of the request. The parsing can be aborted by throwing an error.

bodyParser.text([options])

- Returns middleware that parses all bodies as a string and only looks at requests where the Content-Type header matches the type option.
- This parser supports automatic inflation of gzip and deflate encodings.
- A new body string containing the parsed data is populated on the request object after the middleware (i.e. req.body). This will be a string of the body.

Options

The text function takes an optional options object that may contain any of the following keys:

defaultCharset

Specify the default character set for the text content if the charset is not specified in the Content-Type header of the request. Defaults to utf-8.

inflate

When set to true, then deflated (compressed) bodies will be inflated; when false, deflated bodies are rejected. Defaults to true.

limit

Controls the maximum request body size. If this is a number, then the value specifies the number of bytes; if it is a string, the value is passed to the bytes library for parsing. Defaults to '100kb'.

type

The type option is used to determine what media type the middleware will parse. This option can be a string, array of strings, or a function. If not a function, type option is passed directly to the type-is library and this can be an extension name (like txt), a mime type (like text/plain), or a mime type with a wildcard (like */* or text/*). If a function, the type option is called as fn(req) and the request is parsed if it returns a truthy value. Defaults to text/plain.

verify

The verify option, if supplied, is called as verify(req, res, buf, encoding), where buf is a Buffer of the raw request body and encoding is the encoding of the request. The parsing can be aborted by throwing an error.

bodyParser.urlencoded([options])

- Returns middleware that only parses urlencoded bodies and only looks at requests where the Content-Type header matches the type option.
- This parser accepts only UTF-8 encoding of the body and supports automatic inflation of gzip and deflate encodings.
- A new body object containing the parsed data is populated on the request object after the middleware (i.e. req.body). This object will contain key-value pairs, where the value can be a string or array (when extended is false), or any type (when extended is true).

Options

The urlencoded function takes an optional options object that may contain any of the following keys:

extended

The extended option allows to choose between parsing the URL-encoded data with the querystring library (when false) or the qs library (when true). The "extended" syntax allows for rich objects and arrays to be encoded into the URL-encoded format, allowing for a JSON-like experience with URL-encoded.
Defaults to true, but using the default has been deprecated.

inflate

When set to true, then deflated (compressed) bodies will be inflated; when false, deflated bodies are rejected. Defaults to true.

limit

Controls the maximum request body size. If this is a number, then the value specifies the number of bytes; if it is a string, the value is passed to the bytes library for parsing. Defaults to '100kb'.

parameterLimit

The parameterLimit option controls the maximum number of parameters that are allowed in the URL-encoded data. If a request contains more parameters than this value, a 413 will be returned to the client. Defaults to 1000.

type

The type option is used to determine what media type the middleware will parse. This option can be a string, array of strings, or a function. If not a function, type option is passed directly to the type-is library and this can be an extension name (like urlencoded), a mime type (like application/x-www-form-urlencoded), or a mime type with a wildcard (like */x-www-form-urlencoded). If a function, the type option is called as fn(req) and the request is parsed if it returns a truthy value. Defaults to application/x-www-form-urlencoded.

verify

The verify option, if supplied, is called as verify(req, res, buf, encoding), where buf is a Buffer of the raw request body and encoding is the encoding of the request. The parsing can be aborted by throwing an error.

Examples

Express/Connect top-level generic

This example demonstrates adding a generic JSON and URL-encoded parser as a top-level middleware, which will parse the bodies of all incoming requests. This is the simplest setup.

```
var express = require('express')
var bodyParser = require('body-parser')
var app = express()
// parse application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({ extended: false }))
// parse application/json
app.use(bodyParser.json())
app.use(function (req, res) {
res.setHeader('Content-Type', 'text/plain')
res.write('you posted:\n')
res.end(JSON.stringify(req.body, null, 2))
})
```

Express route-specific

This example demonstrates adding body parsers specifically to the routes that need them. In general, this is the most recommended way to use body-parser with Express.

```
var express = require('express')
var bodyParser = require('body-parser')
var app = express()
// create application/json parser
var jsonParser = bodyParser.json()
// create application/x-www-form-urlencoded parser
var urlencodedParser = bodyParser.urlencoded({ extended: false })
// POST /login gets urlencoded bodies
app.post('/login', urlencodedParser, function (req, res) {
  res.send('welcome, ' + req.body.username)
})
// POST /api/users gets JSON bodies
app.post('/api/users', jsonParser, function (req, res) {
  // create user in req.body
})
```

Change accepted type for parsers

All the parsers accept a type option which allows you to change the Content-Type that the middleware will parse.

```
var express = require('express')
var bodyParser = require('body-parser')
var app = express()
// parse various different custom JSON types as JSON
app.use(bodyParser.json({ type: 'application/*+json' }))
// parse some custom thing into a Buffer
app.use(bodyParser.raw({ type: 'application/vnd.custom-type' }))
// parse an HTML body into a string
app.use(bodyParser.text({ type: 'text/html' }))
```

NodeJS MongoDB connection

Install Node.js

First, make sure you have a supported version of Node.js installed. The current version of MongoDB Node.js Driver requires Node 4.x or greater.

Install the MongoDB Node.js Driver

The MongoDB Node.js Driver allows you to easily interact with MongoDB databases from within Node.js applications. You'll need the driver in order to connect to your database and execute the queries described in this Quick Start series.

If you don't have the MongoDB Node.js Driver installed, you can install it with the following command.

```
npm install mongodb
```

At the time of writing, this installed version 3.6.4 of the driver. Running npm listmongodb will display the currently installed driver version number.

Create a free MongoDB Atlas cluster and load the sample data

Next, you'll need a MongoDB database. The easiest way to get started with MongoDB is to use Atlas, MongoDB's fully-managed database-as-a-service.

Head over to Atlas and create a new cluster in the free tier. At a high level, a cluster is a set of nodes where copies of your database will be stored. Once your tier is created, load the sample data.

Get your cluster's connection info

The final step is to prep your cluster for connection.

In Atlas, navigate to your cluster and click CONNECT. The Cluster Connection Wizard will appear.

The Wizard will prompt you to add your current IP address to the IP Access List and create a MongoDB user if you haven't already done so. Be sure to note the username and password you use for the new MongoDB user as you'll need them in a later step.

Next, the Wizard will prompt you to choose a connection method. Select Connect Your Application. When the Wizard prompts you to select your driver version, select Node.js and 3.6 or later. Copy the provided connection string.

Connect to your database from a Node.js application

Now that everything is set up, it's time to code! Let's write a Node.js script that connects to your database and lists the databases in your cluster.

Import MongoClient

The MongoDB module exports MongoClient, and that's what we'll use to connect to a MongoDB database. We can use an instance of MongoClient to connect to a cluster, access the database in that cluster, and close the connection to that cluster.

```
const {MongoClient} = require('mongodb');
```

Create our main function

Let's create an asynchronous function named main() where we will connect to our MongoDB cluster, call functions that query our database, and disconnect from our cluster.

```
async function main() {
        // we'll add code here soon
}
```

The first thing we need to do inside of main() is create a constant for our connection URI. The connection URI is the connection string you copied in Atlas in the previous section.

When you paste the connection string, don't forget to update <username> and <password> to be the credentials for the user you created in the previous section. The connection string includes a <dbname> placeholder.

For these examples, we'll be using the sample_airbnb database, so replace <dbname> with sample_airbnb.

```

/**
 * Connection URI. Update <username>, <password>, and <your-cluster-url> to reflect your cluster.
 * See https://docs.mongodb.com/ecosystem/drivers/node/ for more details
 */
const uri = "mongodb+srv://<username>:<password>@<your-cluster-url>/test?retryWrites=true&w=majority";
```

Now that we have our URI, we can create an instance of MongoClient.

```
const client = new MongoClient(uri);
```

Note: When you run this code, you may see DeprecationWarnings around the URL string parserand the Server Discover and Monitoring engine. If you see these warnings, you can remove them by passing options to the MongoClient.

For example, you could instantiate MongoClient by calling new MongoClient(uri, { useNewUrlParser: true, useUnifiedTopology: true }).

Now we're ready to use MongoClient to connect to our cluster. client.connect() will return a promise. We will use the await keyword when we call client.connect() to indicate that we should block further execution until that operation has completed.

```
await client.connect();
```

Now we are ready to interact with our database. Let's build a function that prints the names of the databases in this cluster. It's often useful to contain this logic in well named functions in order to improve the readability of your codebase.

Throughout this series, we'll create new functions similar to the function we're creating here as we learn how to write different types of queries. For now, let's call a function named listDatabases().

```
await listDatabases(client);
```

Let's wrap our calls to functions that interact with the database in a try/catch statement so that we handle any unexpected errors.

```
try {
    await client.connect();
    await listDatabases(client);
} catch (e) {
    console.error(e);
}
```

We want to be sure we close the connection to our cluster, so we'll end our try/catch with a finally statement.

```
finally {
    await client.close();
}
```

Once we have our main() function written, we need to call it. Let's send the errors to the console.

```
main().catch(console.error);
```

Putting it all together, our main() function and our call to it will look something like the following.

```
async function main(){
    /**
     * Connection URI. Update <username>, <password>, and <your-cluster-url> to
reflect your cluster.
     * See https://docs.mongodb.com/ecosystem/drivers/node/ for more details
     */
    const uri = "mongodb+srv://<username>:<password>@<your-cluster-
url>/test?retryWrites=true&w=majority";
    const client = new MongoClient(uri);
    try {
        // Connect to the MongoDB cluster
        await client.connect();
        // Make the appropriate DB calls
        await  listDatabases(client);
    } catch (e) {
        console.error(e);
    } finally {
        await client.close();
    }
}
main().catch(console.error);
```

This function will retrieve a list of databases in our cluster and print the results in the console.

```
async function listDatabases(client){
    databasesList = await client.db().admin().listDatabases();
     console.log("Databases:");
    databasesList.databases.forEach(db => console.log(` - ${db.name}`));
};
```

Save Your File

You've been implementing a lot of code. Save your changes, and name your file something like connection.js.

Execute Your Node.js Script

Now you're ready to test your code! Execute your script by running a command like the following in your terminal: **node connection.js**

You will see output like the following:

```
Databases:
- sample_airbnb
- sample_geospatial
- sample_mflix
- sample_supplies
- sample_training
- sample_weatherdata
- admin
- local
```

Adding and retrieving data to MongoDB from NodeJS

- ✓ **MongoDB**, the most popular NoSQL database, is an open-source document-oriented database.
- ✓ The term 'NoSQL' means 'non-relational'.
- ✓ It means that MongoDB isn't based on the table-like relational database structure but provides an altogether different mechanism for storage and retrieval of data.
- ✓ This format of storage is called BSON (similar to JSON format).

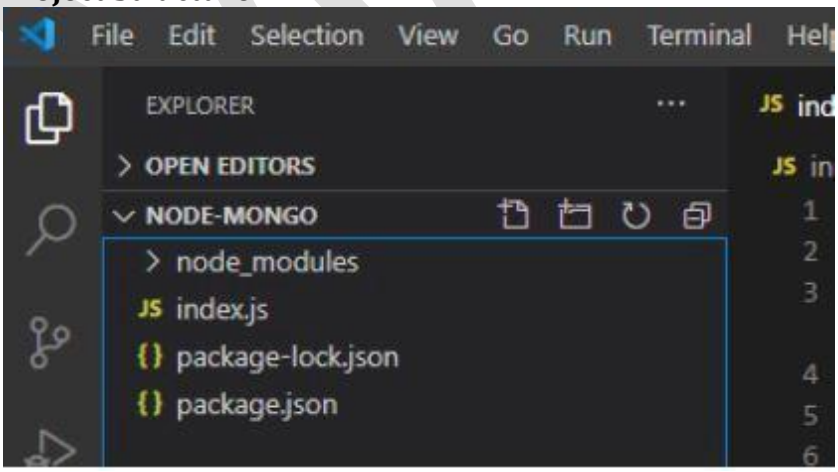
MongoDB module:

- ➢ This module of Node.js is used for connecting the MongoDB database as well as used for manipulating the collections and databases in MongoDB.
- ➢ The mongodb.connect() method is used for connecting the MongoDB database which is running on a particular server on your machine.
- ➢ We can also use promises, in this method in resolve the object contains all the methods and properties required for collection manipulation and in reject the error occurs during connection.

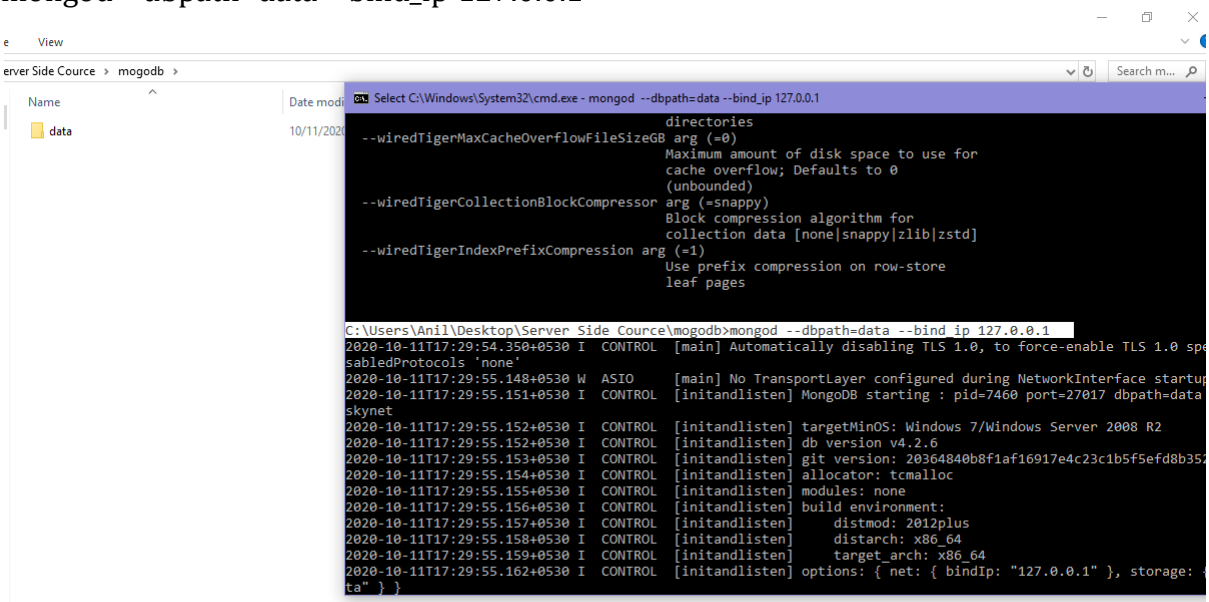
Installing module:

node install mongodb

Project Structure:



Running the server on Local IP: data is folder name
mongod --dbpath=data --bind_ip 127.0.0.1



MongoDB Database:
Database:GFG
Collection:GFGcollections

```
> use GFG
switched to db GFG
> show tables
GFGcollections
> db.GFGcollections.find().pretty()
{
        "_id" : ObjectId("5fa6dbe4776dcc3140b180da"),
        "name" : "GfGnew",
        "class" : "GFG"
}
{
        "_id" : ObjectId("5faac0b70337755c2fd679b1"),
        "name" : "aayush",
        "class" : "GFG"
}
{
        "_id" : ObjectId("5faac0cc0337755c2fd679b2"),
        "name" : "saini",
        "class" : "bye"
}
>
```

Index.js
1. Fetching single document of GFGcollections
JAVASCRIPT:

```
const MongoClient = require("mongodb");
const url = 'mongodb://localhost:27017/';
const databasename = "GFG"; // Database name
MongoClient.connect(url).then((client) => {
        const connect = client.db(databasename);
```

```
        // Connect to collection
        const collection = connect
                        .collection("GFGcollections");
        // Fetching the records having
        // name as saini
        collection.find({ "name": "saini" })
                .toArray().then((ans) => {
                        console.log(ans);
                });
}).catch((err) => {
        // Printing the error message
        console.log(err.Message);
})
```

OUTPUT:

```
Anil@skynet MINGW64 ~/Desktop/Server Side Cource/node-mongo
$ node index.js
(node:11564) DeprecationWarning: current Server Discovery and Monitoring engine is deprecated, and will be removed in a future version. To use the new Server Discover and Monitoring
engine, pass option { useUnifiedTopology: true } to the MongoClient constructor.
(Use `node --trace-deprecation ...` to show where the warning was created)
[ { _id: 5faac0cc0337755c2fd679b2, name: 'saini', class: 'bye' } ]
```

2. <u>Fetching all documents of the GFGcollections</u>

JAVASCRIPT:

```
const MongoClient = require("mongodb");
const url = 'mongodb://localhost:27017/';
const databasename = "GFG"; // Database name
MongoClient.connect(url).then((client) => {
        const connect = client.db(databasename);
        // Connect to collection
        const collection = connect
                .collection("GFGcollections");
        collection.find({}).toArray().then((ans) => {
                console.log(ans);
        });
}).catch((err) => {
        // Printing the error message
        console.log(err.Message);
})
```

OUTPUT:

```
$ node index.js
(node:3232) DeprecationWarning: current Server Discovery and Monitoring engine is deprecated, and will be removed in a future version. To use the new Server Discover and Monitoring e
ngine, pass option { useUnifiedTopology: true } to the MongoClient constructor.
(Use `node --trace-deprecation ...` to show where the warning was created)
[
  { _id: 5fa6dbe4776dcc3140b180da, name: 'GFGnew', class: 'GFG' },
  { _id: 5faac0b70337755c2fd679b1, name: 'aayush', class: 'GFG' },
  { _id: 5faac0cc0337755c2fd679b2, name: 'saini', class: 'bye' }
]
```

Handling SQL databases from NodeJS

- ✓ Node.js can be used in database applications.
- ✓ One of the most popular databases is MySQL.

MySQL Database

To be able to experiment with the code examples, you should have MySQL installed on your computer.

You can download a free MySQL database at https://www.mysql.com/downloads/.

Install MySQL Driver

Once you have MySQL up and running on your computer, you can access it by using Node.js.

To access a MySQL database with Node.js, you need a MySQL driver.

To download and install the "mysql" module, open the Command Terminal and execute the following:

C:\Users\Your Name>npm install mysql

Now you have downloaded and installed a mysql database driver.

Node.js can use this module to manipulate the MySQL database:

```
var mysql = require('mysql');
```

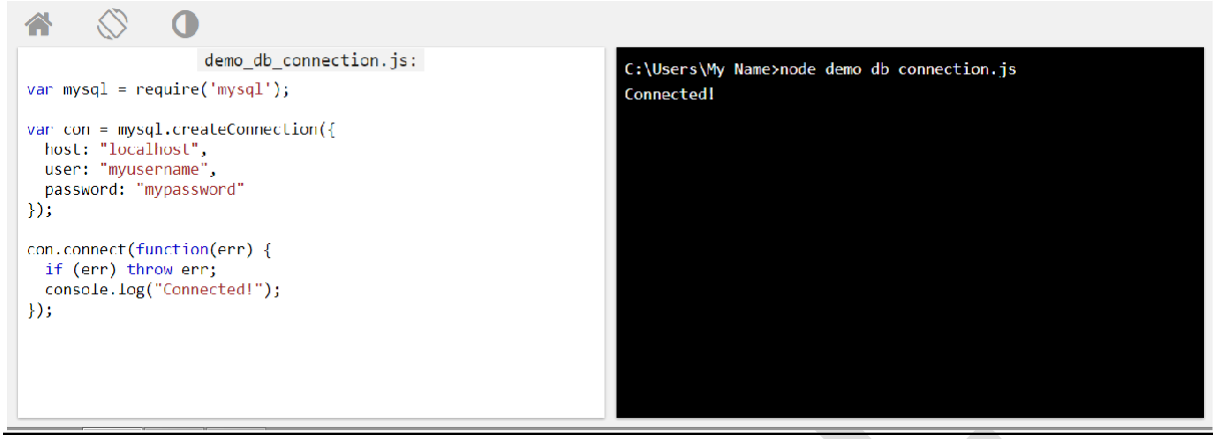
Create Connection

Start by creating a connection to the database.

Use the username and password from your MySQL database.

demo_db_connection.js

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword"
});
con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
});
```



Handling Cookies in NodeJS

- ✓ A cookie is a mechanism that allows the server to store its own information about a user on the user's own computer.
- ✓ You can view the cookies that have been stored on your hard disk
- ✓ The location of the cookies depends on the browser.
- ✓ To use cookies in nodejs express application, we use cookie-parser package of npm, By using this package we can easily manage the express application cookies.

Install package

```
npm install  cookie-parser
```

Use this middleware

```
const express = require('express');
const cookieParser = require('cookie-parser');
const app = express();
// adding cookieParser to middleware stack
app.use(cookieParser());
```

The module gives us access to req.cookies with an object keyed with the cookie name. We can also enable signed cookie support by passing a secret string, Which assigns req.secret.

Set Cookie

```
app.get('/cookie',function(req, res){
let minute = 60 * 1000;
res.cookie(cookie_name, 'cookie_value', { maxAge: minute });
return res.send('cookie has been set!');
});
```

In the above code, We have set the maximum age of a Cookie, Which is optional.

We can also set the expire time in milliseconds like below
res.cookie(cookie_name , 'cookie_value', {expire : 24 * 60 * 60 * 1000 });

We can also set cookie only over HttpOnly.

This flag tells the browsers to not allow client-side script access to the Cookie.

res.cookie(cookie_name , 'cookie_value', { HttpOnly: true});

We can tell express to use https encrypted channel to exchange cookie data with secure flag.

```
res.cookie(cookie_name , 'cookie_value', { secure: true});
```

Read Cookies

We can access Cookies via request object, req.cookies.cookie_nameor req.cookies.

Delete cookies

We can also easily delete Cookies by using res.clearCookie function, which accepts the name of the Cookie which we want to delete.

We can also delete Cookies from browser developers tools.

```
app.get('/deletecookie', function(req,res){
res.clearCookie('cookie_name');
res.send('Cookie deleted');
});
```

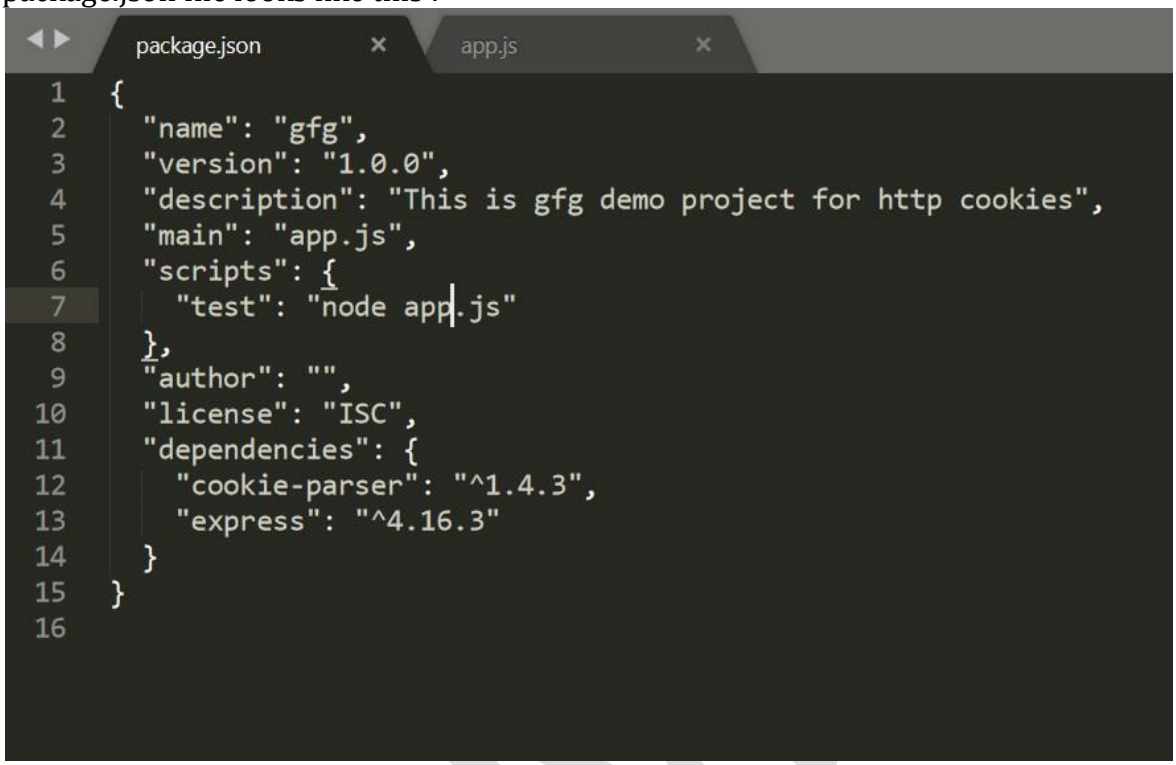
HTTP Cookies in Node.js

- ✓ Cookies are small data that are stored on a client side and sent to the client along with server requests.
- ✓ Cookies have various functionality, they can be used for maintaining sessions and adding user-specific features in your web app.
- ✓ For this, we will use **cookie-parser** module of npm which provides middleware for parsing of cookies.

First set your directory of the command prompt to root folder of the project and run the following command: **npm init**

This will ask you details about your app and finally will create a **package.json** file.

After that run the following command and it will install the required module and add them in your package.json file npm install express cookie-parser --save package.json file looks like this :



After that we will setup basic express app by writing following code in our app.js file in root directory .

```
let express = require('express');
//setup express app
let app = express()
//basic route for homepage
app.get('/', (req, res)=>{
res.send('welcome to express app');
});
//server listens to port 3000
app.listen(3000, (err)=>{
if(err)
throw err;
console.log('listening on port 3000');
});
```

After that if we run the command : **node app.js**

Handling User Authentication with NodeJS

Introduction

- Creating a user registration form employs the management of the registered user.
- This is where user role authentication comes into play. Role authentication ensures that non-admin users cannot make changes or access exclusive information.
- It grants administrative privileges to admin users and basic privileges to basic users.
- You can build your own authentication functionality with web tokens like JSON Web Token (JWT) or use a trusted third-party customer identity and access management (CIAM) software like LoginRadius.

Set Up a Mongo Database

You'll store all your user data — which includes username, password, and role — in MongoDB.

Install a node package called Mongoose that will connect to MongoDB. Then create a user schema for your application.

```
npm init
npm install mongoose
```

- ✓ npm init sets up your new project and creates a package.json file with the credentials.
- ✓ After installing mongoose, create a new file db.js in the project's directory and require mongoose.

```
const Mongoose = require("mongoose")
```

With the help of mongoose, you can connect your application to MongoDB:

```
// db.js
const Mongoose = require("mongoose")
const localDB = `mongodb://localhost:27017/role_auth`
const connectDB = async () => {
  await Mongoose.connect(localDB, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
  })
  console.log("MongoDB Connected")
```

```
}
module.exports = connectDB
```

- ✓ The code snippet here connects to mongodb://localhost:27017 and then specifies the name of the database /role_auth.
- ✓ The function connectDB awaits for the connection, which contains the URI and options as a second parameter.
- ✓ If it connects without errors, it will log out MongoDB Connected. Error issues will be fixed while connecting to the database.
- ✓ After this, it exported the function for use in the server.

Set Up the Server

```
npm i express nodemon
```

- ✓ Express.js is a Node.js framework for building web applications quickly and easily.
- ✓ Nodemon is a tool that watches the file system and automatically restarts the server when there is a change.
- ✓ You require express in your application to listen for a connection on port 5000.
- ✓ Create a new file server.js in the root directory and create the listening event:

```
const express = require("express")
const app = express()
const PORT = 5000
app.listen(PORT, () => console.log(`Server Connected to port ${PORT}`))
```

The next step is to test your application. Open up your package.json file and add the following to scripts:

```
"scripts": {
  "start": "node server.js",
  "dev": "nodemon server.js"
}
```

Open your terminal and run npm run dev to start the server.

Connect to the Database

Earlier, you've created a function that connects to MongoDB and exported that function. Now import that function into your server.js:

```
const connectDB = require("./db");
...
//Connecting the Database
connectDB();
```

You also need to create an error handler that catches every unhandledRejection error.

```
const server = app.listen(PORT, () =>
  console.log(`Server Connected to port ${PORT}`)
)
// Handling Error
process.on("unhandledRejection", err => {
console.log(`An error occurred: ${err.message}`)
server.close(() => process.exit(1))
})
```

- ✓ The listening event is assigned to a constant server.
- ✓ If an unhandledRejection error occurs, it logs out the error and closes the server with an exit code of 1.

Create User Schema

- ✓ Schema is like a blueprint that shows how the database will be constructed.
- ✓ You'll structure a user schema that contains username, password, and role.
- ✓ Create a new folder model in the project's directory, and create a file called User.js.

Now open User.js and create the user schema:

```
// user.js
const Mongoose = require("mongoose")
const UserSchema = new Mongoose.Schema({
username: {
   type: String,
   unique: true,
   required: true,
 },
 password: {
   type: String,
   minlength: 6,
   required: true,
 },
 role: {
   type: String,
   default: "Basic",
   required: true,
 },
})
```

- ✓ In the schema, the username will be unique, required, and will accept strings.
- ✓ You've specified the minimum characters(6) the password field will accept. The role field grants a default value (basic) that you can change if needed.

Now, you need to create a user model and export it:

```
const User = Mongoose.model("user", UserSchema)
module.exports = User
```

You've created the user model by passing the UserSchema as the second argument while the first argument is the name of the model user.

Perform CRUD Operations

You'll create functions that handle:

- adding users;
- getting all users;
- updating the role of users; and,
- deleting users.

Register Function

- As the name implies, this function will handle the registrations of users.
- Let's create a new folder named Auth. It will contain the Authentication file and the Route set-up file.
- After creating the Auth folder, add two files — Auth.js and Route.js.

Now open up our Auth.js file and import that User model:

```
const User = require("../model/User")
```

- The next step is to create an async express function that will take the user's data and register it in the database.
- You need to use an Express middleware function that will grant access to the user's data from the body.

You'll use this function in the server.js file:

```
const app = express()
app.use(express.json())
```

Let's go back to your Auth.js file and create the register function:

```
// auth.js
exports.register = async (req, res, next) => {
 const { username, password } = req.body
 if (password.length < 6) {
  return res.status(400).json({ message: "Password less than 6 characters" })
 }
 try {
  await User.create({
   username,
   password,
  }).then(user =>
   res.status(200).json({
    message: "User successfully created",
    user,
   })
  )
 } catch (err) {
  res.status(401).json({
   message: "User not successful created",
   error: error.mesage,
  })
 }
}
```

The exported register function will be used to set up the routes.

You got the username and password from the req.body and created a tryCatch block that will create the user if successful; else, it returns status code 401 with the error message.

Set Up Register Route

You'll create a route to /register using express.Router.

Import the register function into your route.js file, and use it as the route's function:

```
const express = require("express")
const router = express.Router()
const { register } = require("./auth")
```

```
router.route("/register").post(register)
module.exports = router
```

The last step is to import your route.js file as middleware in server.js:

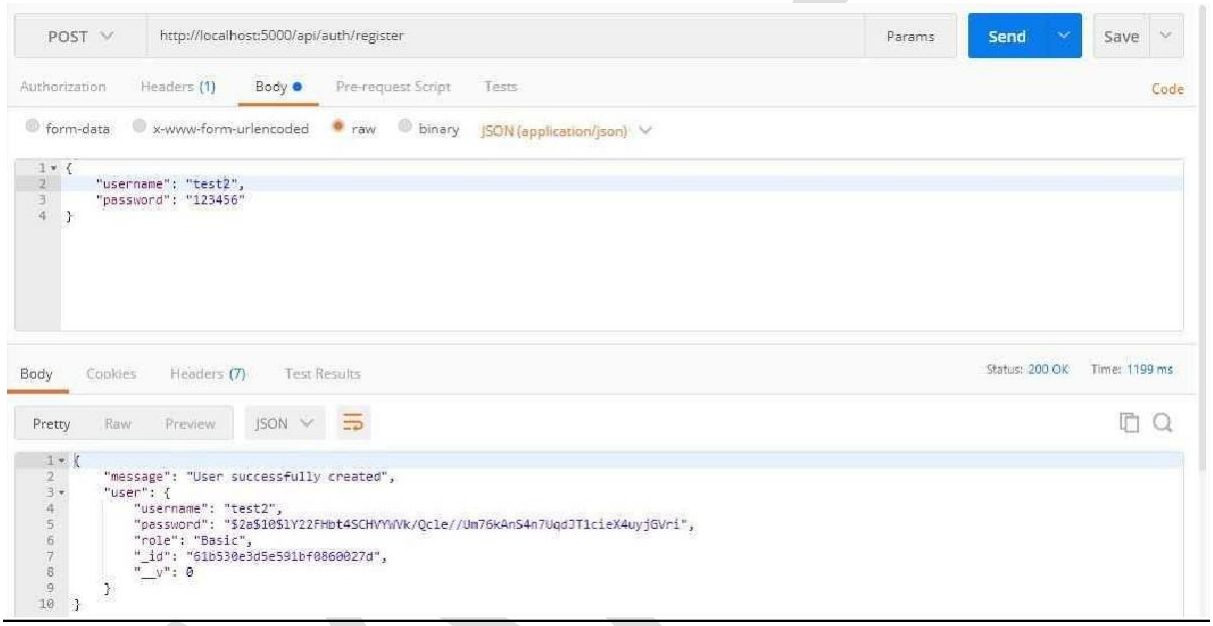
```
app.use("/api/auth", require("./Auth/route"))
```

The server will use the router middleware function if there is a request to /api/auth.

Test the Register Route

You'll use Postman to test all the routes.

Open up Postman to send a POST request to http://localhost:5000/api/auth/register and pass the username and password to the body:



Login Function

- You've created a function that adds registered users to the database.
- You have to create another function that will authenticate user credentials and check if the user is registered.

Open the Auth.js file and create the Login function, as follows:

```
// auth.js
exports.login = async (req, res, next) => {
  const { username, password } = req.body
  // Check if username and password is provided
  if (!username || !password) {
    return res.status(400).json({
      message: "Username or Password not present",
    })
  }
}
```

The login function returns status code 400 if the username and password were not provided.

You need to find a user with the provided username and password:

```
exports.login = async (req, res, next) => {
  try {
    const user = await User.findOne({ username, password })
    if (!user) {
```

```
    res.status(401).json({
      message: "Login not successful",
      error: "User not found",
    })
  } else {
    res.status(200).json({
      message: "Login successful",
      user,
    })
  }
 } catch (error) {
  res.status(400).json({
  message: "An error occurred",
  error: error.message,
  })
 }
}
```

Here, it returns status code 401 when a user isn't found and 200 when a user is found. The code snippet wrapped all this in a tryCatch block to detect and output errors, if any.

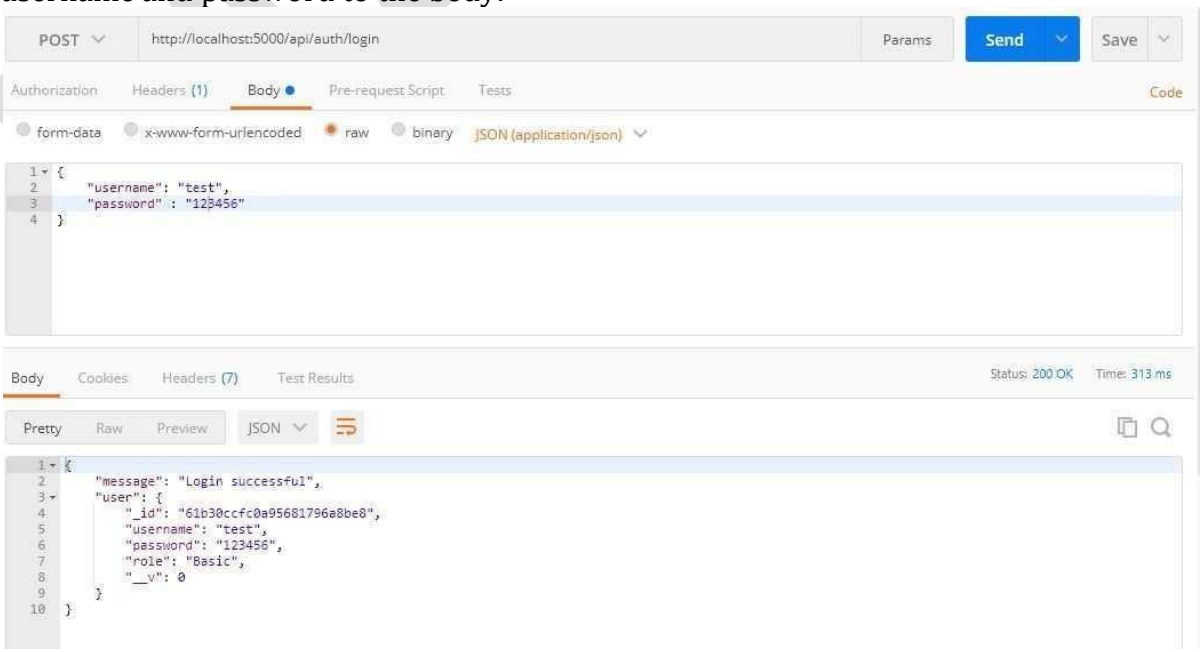
Set Up Login Route

To set up the login route, import the login function into your route.js:

```
const express = require("express");
const router = express.Router();
const { register, login } = require("./auth");
...
router.route("/login").post(login);
module.exports = router;
```

Test the Login Route

Make a POST request at http://localhost:5000/api/auth/login and pass a valid username and password to the body:



Update Function

This function will be responsibile for updating the role of a basic user to an admin user. Open the auth.js file and create the update function, as follows:

```
//auth.js
exports.update = async (req, res, next) => {
  const { role, id } = req.body
  // Verifying if role and id is presnt
  if (role && id) {
    // Verifying if the value of role is admin
    if (role === "admin") {
      await User.findById(id)
    } else {
      res.status(400).json({
        message: "Role is not admin",
      })
    }
  } else {
    res.status(400).json({ message: "Role or Id not present" })
  }
}
```

- ✓ The first if statement verifies if role and id are present in the request body.
- ✓ The second if statement checks if the value of role is admin.
- ✓ You should do this to avoid having over two roles.
- ✓ After finding a user with that ID, you'll create a third if block that will check for the role of the user:

```
exports.update = async (req, res, next) => {
  const { role, id } = req.body;
  // First - Verifying if role and id is presnt
  if (role && id) {
    // Second - Verifying if the value of role is admin
    if (role === "admin") {
      // Finds the user with the id
      await User.findById(id)
        .then((user) => {
          // Third - Verifies the user is not an admin
          if (user.role !== "admin") {
            user.role = role;
            user.save((err) => {
              //Monogodb error checker
              if (err) {
                res
                  .status("400")
                  .json({ message: "An error occurred", error: err.message });
                process.exit(1);
              }
              res.status("201").json({ message: "Update successful", user });
            });
          } else {
            res.status(400).json({ message: "User is already an Admin" });
```

```
      }
    })
    .catch((error) => {
      res
        .status(400)
        .json({ message: "An error occurred", error: error.message });
    });

  ...
```

The third if block prevents assigning an admin role to an admin user, while the last if block checks if an error occurred when saving the role in the database.

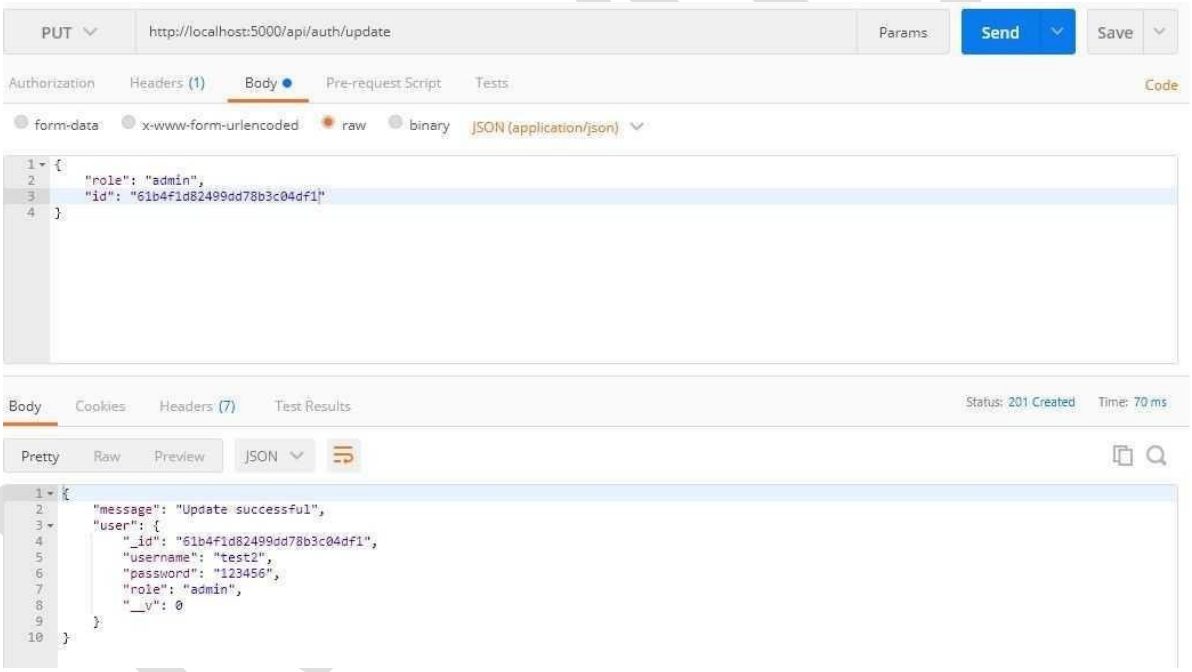
Set Up Update Route

Import the update function in your route.js, as follows:

```
const { register, login, update } = require("./auth");
...
router.route("/update").put(update);
```

Testing the Update Route

Send a put request to http://localhost:5000/api/auth/update:



Delete Function

The deleteUser function will remove a specific user from the database. Let's create this function in our auth.js file:

```
exports.deleteUser = async (req, res, next) => {
  const { id } = req.body
  await User.findById(id)
    .then(user => user.remove())
    .then(user =>
      res.status(201).json({ message: "User successfully deleted", user })
    )
    .catch(error =>
      res
```

```
      .status(400)
      .json({ message: "An error occurred", error: error.message })
    )
}
```

You remove the user based on the id you get from req.body.

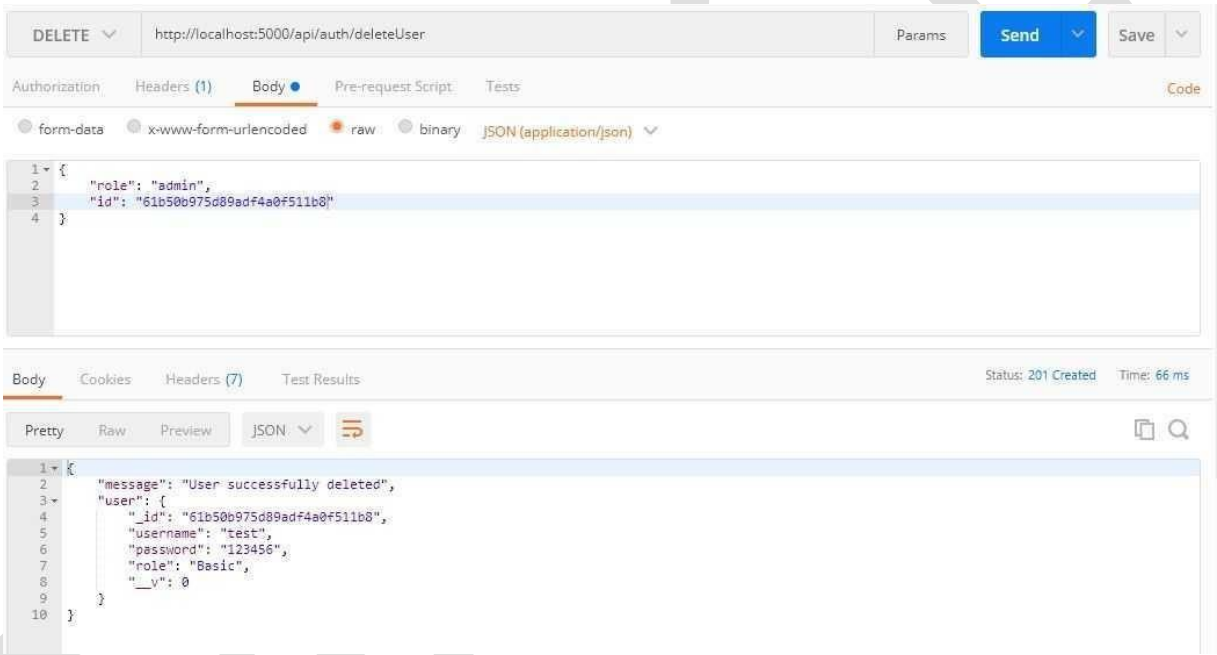
Set up the deleteUser Route

Open your route.js file to create a delete request to /deleteUser, using the deleteUser as its function:

```
const { register, login, update, deleteUser } = require("./auth");
...
router.route("/deleteUser").delete(deleteUser);
```

Test the deleteUser Route

Send a delete request to http://localhost:5000/api/auth/deleteUser by passing a valid id to the body:



Hash User Passwords

- Saving user passwords in the database in plain text format is reckless.
- It is preferable to hash your password before storing it.
- For instance, it will be tough to decipher the passwords in your database if they are leaked.
- Hashing passwords is a cautious and reliable practice.
- Let's use bcryptjs to hash your user passwords.

Lets install bcryptjs:

```
npm i bcryptjs
```

After installing bcryptjs, import it into your auth.js

```
const bcrypt = require("bcryptjs")
```

Refactor Register Function

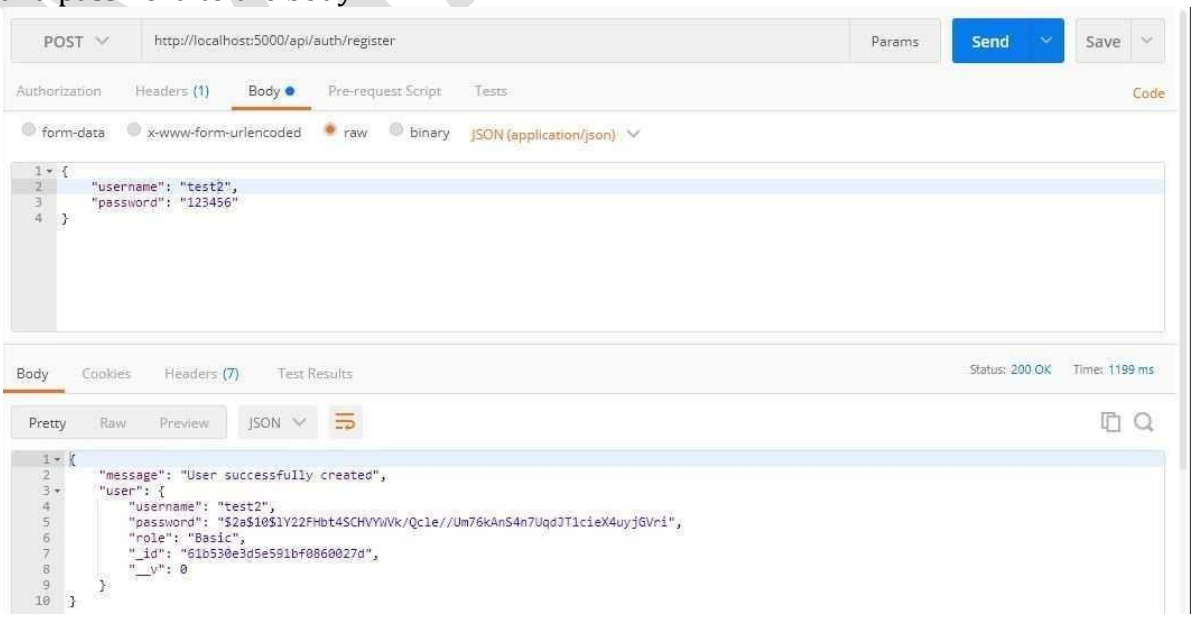
Instead of sending a plain text format to your database, lets hash the password using bcrypt:

```
exports.register = async (req, res, next) => {
 const { username, password } = req.body;
 bcrypt.hash(password, 10).then(async (hash) => {
  await User.create({
   username,
   password: hash,
  })
   .then((user) =>
    res.status(200).json({
     message: "User successfully created",
      user,
    })
   )
   .catch((error) =>
    res.status(400).json({
     message: "User not successful created",
     error: error.message,
    })
   );
 });
};
```

- bcrypt takes in your password as the first argument and the number of times it will hash the password as the second argument.
- Passing a large number will take bcrypt a long time to hash the password, so give a moderate number like 10.
- bcrypt will return a promise with the hashed password; then, send that hashed password to the database.

Test the Register Function

Send a POST request to http://localhost:5000/api/auth/register and pass the username and password to the body:



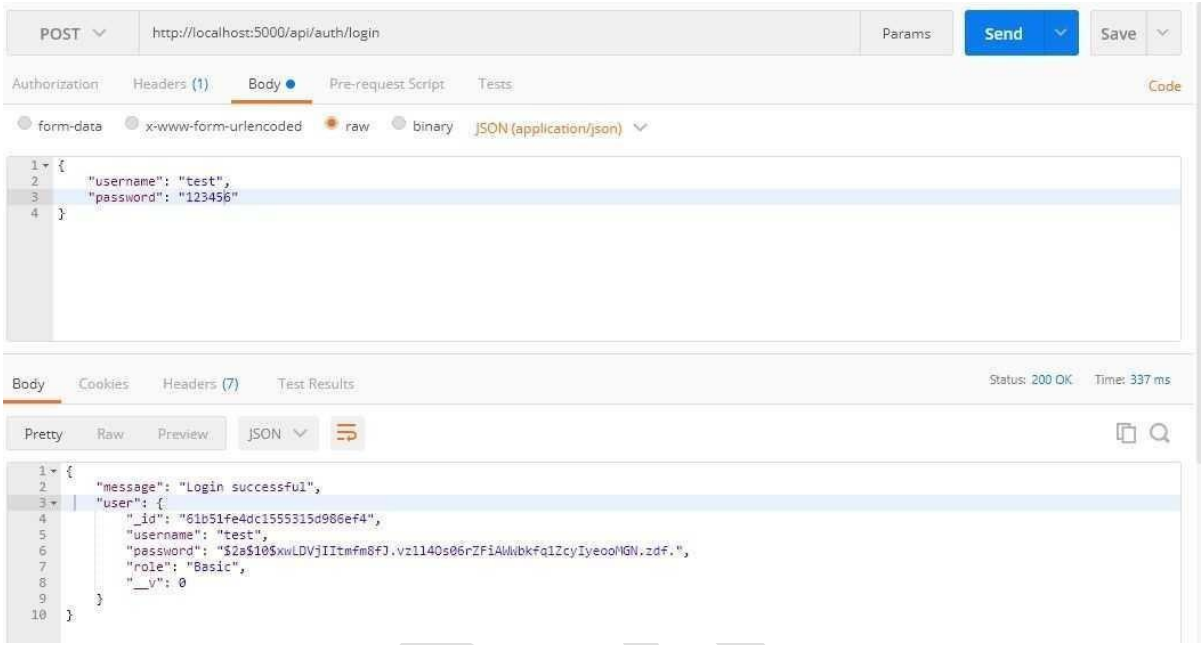
Refactor the Login Function

```
exports.login = async (req, res, next) => {
  const { username, password } = req.body
  // Check if username and password is provided
  if (!username || !password) {
    return res.status(400).json({
      message: "Username or Password not present",
    })
  }
  try {
    const user = await User.findOne({ username })
    if (!user) {
      res.status(400).json({
        message: "Login not successful",
        error: "User not found",
      })
    } else {
      // comparing given password with hashed password
      bcrypt.compare(password, user.password).then(function (result) {
      result
          ? res.status(200).json({
              message: "Login successful",
              user,
            })
          : res.status(400).json({ message: "Login not succesful" })
      })
    }
  } catch (error) {
    res.status(400).json({
    message: "An error occurred",
    error: error.message,
    })
  }
}
```

bcrypt.compare checks if the given password and the hashed password in the database are the same.

Test the Login Function

Send a POST request to http://localhost:5000/api/auth/login and pass a valid username and password to the body:



Authenticate Users with JSON Web Token (JWT)

- JSON Web Token helps shield a route from an unauthenticated user.
- Using JWT in your application will prevent unauthenticated users from accessing your users' home page and prevent unauthorized users from accessing your admin page.
- JWT creates a token, sends it to the client, and then the client uses the token for making requests.
- It also helps verify that you're a valid user making those requests.

You've to install JWT before using it in your application:

```
npm i jsonwebtoken
```

Refactor the Register Function

- ✓ When a user registers, you'll send a token to the client using JWT as a cookie.
- ✓ To create this token, you need to set a secret string.

You'll use the node's in-built package called crypto to create random strings:

```
node
require("crypto").randomBytes(35).toString("hex")
```

Output:

```
$ node
Welcome to Node.js v14.17.0.
Type ".help" for more information.
> require('crypto').randomBytes(35).toString('hex')
'4715aed3c946f7b0a38e6b534a9583628d84e96d10fbc04700770d572af3dce43625dd'
>
```

- ✓ Storing this secret string in an environment variable is a safe practice.
- ✓ If this secret string is leaked, unauthenticated users can create fake tokens to access the route.

Store your secret string in a variable:

```
const jwtSecret =

"4715aed3c946f7b0a38e6b534a9583628d84e96d10fbc04700770d572af3dce43
625dd"
```

Once you've created your jwtSecret, import jsonwebtoken as the token in the register function:

```
const jwt = require('jsonwebtoken')
const jwtSecret =
'4715aed3c946f7b0a38e6b534a9583628d84e96d10fbc04700770d572af3dce43
625dd'
exports.register = async (req, res, next) => {
 const { username, password } = req.body;
 bcrypt.hash(password, 10).then(async (hash) => {
  await User.create({
   username,
   password: hash,
  })
   .then((user) => {
    const maxAge = 3 * 60 * 60;
    const token = jwt.sign(
     { id: user._id, username, role: user.role },
     jwtSecret,
     {
      expiresIn: maxAge, // 3hrs in sec
     }
    );
    res.cookie("jwt", token, {
     httpOnly: true,
     maxAge: maxAge * 1000, // 3hrs in ms
    });
    res.status(201).json({
     message: "User successfully created",
     user: user._id,
    });
   })
   .catch((error) =>
    res.status(400).json({
     message: "User not successful created",
     error: error.message,
    })
   );
 });
};
```

The code snippet created the token using JWT's sign function. This function takes in three parameters:

- the payload is the first parament that you'll pass to the function. This payload holds data concerning the user, and this data should not contain sensitive information like passwords;
- you passed your jwtSecret as the second parameter; and,
- how long the token will last as the third parameter.

After passing all these arguments, JWT will generate a token. After the token is generated, send it as a cookie to the client.

Refactor the Login Function

Also, generate a token for logged in users:

```
exports.login = async (req, res, next) => {
    bcrypt.compare(password, user.password).then(function (result) {
      if (result) {
        const maxAge = 3 * 60 * 60;
        const token = jwt.sign(
          { id: user._id, username, role: user.role },
          jwtSecret,
          {
            expiresIn: maxAge, // 3hrs in sec
          }
        );
        res.cookie("jwt", token, {
          httpOnly: true,
          maxAge: maxAge * 1000, // 3hrs in ms
        });
        res.status(201).json({
          message: "User successfully Logged in",
          user: user._id,
        });
      } else {
        res.status(400).json({ message: "Login not succesful" });
      }
    });
  }
 } catch (error) {
  res.status(400).json({
  message: "An error occurred",
  error: error.message,
  });
 }
};
```

Protect the Routes

To prevent unauthenticated users from accessing the private route, take the token from the cookie, verify the token, and redirect users based on role.

You'll get the token from the client using a node package called cookie-parser.

Let's install the package before using it:

```
npm i cookie-parser
```

After installing it, import it into your server.js file and use it as a *middleware*:

```
const cookieParser = require("cookie-parser");

app.use(cookieParser());
```

You'll create your middleware that verifies the token and grants access to your private route.

Let's create a new folder in the project's folder named middleware and create a file called auth.js.

Admin Authentication

Open the auth.js file and create the middleware:

```
const jwt = require("jsonwebtoken")
const jwtSecret =

"4715aed3c946f7b0a38e6b534a9583628d84e96d10fbc04700770d572af3dce43
625dd"
exports.adminAuth = (req, res, next) => {
  const token = req.cookies.jwt
  if (token) {
    jwt.verify(token, jwtSecret, (err, decodedToken) => {
      if (err) {
        return res.status(401).json({ message: "Not authorized" })
      } else {
        if (decodedToken.role !== "admin") {
          return res.status(401).json({ message: "Not authorized" })
        } else {
          next()
        }
      }
    })
  } else {
    return res
      .status(401)
      .json({ message: "Not authorized, token not available" })
  }
}
```

- ✓ The code snippet requests a token from the client, checks if a token is available, and verifies that token.
- ✓ JWT verifies your token with your jwtSecret and returns a callback function.
- ✓ This function returns status code 401 if the token fails the authentication test.
- ✓ When you've created the token, you passed a payload that contained the user's credentials.
- ✓ You'll get the role from the credentials and check if the user's role is admin.
- ✓ If the user is not an admin, you return status code 401, but you'll call the next function if the user is an admin.

Basic User Authentication

You'll also authenticate basic users before granting them access to the users route.
Let's create another middleware in your auth.js file that will authenticate basic users:

```
exports.userAuth = (req, res, next) => {
  const token = req.cookies.jwt
  if (token) {
    jwt.verify(token, jwtSecret, (err, decodedToken) => {
      if (err) {
        return res.status(401).json({ message: "Not authorized" })
      } else {
        if (decodedToken.role !== "Basic") {
          return res.status(401).json({ message: "Not authorized" })
        } else {
          next()
        }
      }
    })
  } else {
    return res
      .status(401)
      .json({ message: "Not authorized, token not available" })
  }
}
```

Protect the Routes

You'll have two routes: one for the user and the other for the admin.
Let's import this middleware into your server.js file and protect your routes:

```
const { adminAuth, userAuth } = require("./middleware/auth.js");

app.get("/admin", adminAuth, (req, res) => res.send("Admin Route"));
app.get("/basic", userAuth, (req, res) => res.send("User Route"));
```

Updating user roles and deleting users should be done by an Admin, so you need to import this auth.js middleware into your route.js file to protect the update and delete routes.
route.js:

```
const { adminAuth } = require("../middleware/auth")
router.route("/update").put(adminAuth, update)
router.route("/deleteUser").delete(adminAuth, deleteUser)
```

Populate the Database with Admin User

You need to create an admin user in your database.
Open up your terminal, and let's run some MongoDB methods:

```
mongo
```

After mongo is started, you need to use the role_auth database:

```
use role_auth
```

Before adding your admin user to the database, you need to hash the password using bcrypt in the node terminal. Open node terminal in your project's directory:

```
const password = require("bcryptjs").hash("admin", 10)
password
```

After you've created the constant password, you need to enter the password in the node terminal to get your hashed password.

```
> const password = require("bcryptjs").hash("admin", 10)
undefined
> password
Promise {
  '$2a$10$mZwU9AbYSyX7E1A6fu/ZO.BDhmCOIK7k6jXvKcuJm93PyYuH2eZ3K'
}
```

You'll use the hashed password to create your admin:

```
db.users.insert({
  username: "admin",
  password:
"$2a$10$mZwU9AbYSyX7E1A6fu/ZO.BDhmCOIK7k6jXvKcuJm93PyYuH2eZ3K",
  role: "admin",
})
```

To check if it was successfully created, run db.users.find().pretty() — this will output all users in the database.

UNIT IV

ADVANCED CLIENT SIDE PROGRAMMING

React JS

React. js is an open-source JavaScript library that is used for **building user interfaces specifically for single-page applications**. It's used for handling the view layer for web and mobile apps. React also allows us to create reusable UI components.

Let us understand this with a practical example.

Let's say one of your friends posted a photograph on Facebook. Now you go and like the image and then you started checking out the comments too. Now while you are browsing over comments you see that the likes count has increased by 100, since you liked the picture, even without reloading the page. This magical count change is because of ReactJS.

React is a declarative, efficient, and flexible JavaScript library for building user interfaces. 'V' denotes the view in MVC. ReactJS is an open-source, component-based front end library responsible only for the view layer of the application. It is maintained by Facebook.

React uses a declarative paradigm that makes it easier to reason about your application and aims to be both efficient and flexible. It designs simple views for each state in your application, and React will efficiently update and render just the right component when your data changes. The declarative view makes your code more predictable and easier to debug.

A React application is made of multiple components, each responsible for rendering a small, reusable piece of HTML. Components can be nested within other components to allow complex applications to be built out of simple building blocks. A component may also maintain an internal state – for example, a TabList component may store a variable corresponding to the currently open tab.

React is not a framework. It is just a library developed by Facebook to solve some problems that we were facing earlier.

Prerequisites: Download Node packages with their latest version.

Example: Create a new React project by using the command below:

npx create-react-app myapp

ReactDOM

ReactJS is a library to build active User Interfaces thus rendering is one of the integral parts of ReactJS. React provides the developers with a package **react-dom** a.k.a ReactDOM to access and modify the DOM.

What is DOM?

DOM, abbreviated as Document Object Model, is a World Wide Web Consortium standard logical representation of any webpage. In easier words, DOM is a tree-like structure that contains all the elements and it's properties of a website as its nodes. DOM provides a language-neutral interface that allows accessing and updating of the content of any element of a webpage.

Before React, Developers directly manipulated the DOM elements which resulted in frequent DOM manipulation, and each time an update was made the browser had to recalculate and repaint the whole view according to the particular CSS of the page, which made the total process to consume a lot of time.

As a betterment, React brought into the scene the virtual DOM. The **Virtual DOM** can be referred to as a copy of the actual DOM representation that is used to hold the updates made by the user and finally reflect it over to the original Browser DOM at once consuming much lesser time.

What is ReactDOM?

ReactDOM is a package that provides DOM specific methods that can be used at the top level of a web app to enable an efficient way of managing DOM elements of the web page. ReactDOM provides the developers with an API containing the following methods and a few more.

- render()
- findDOMNode()
- unmountComponentAtNode()
- hydrate()
- createPortal()

Pre-requisite: To use the ReactDOM in any React web app we must first import ReactDOM from the react-dom package by using the following code snippet:

import ReactDOM from 'react-dom'

1. render() Function

This is one of the most important methods of ReactDOM. This function is used to render a single React Component or several Components wrapped together in a Component or a div element. This function uses the efficient methods of React for updating the DOM by being able to change only a subtree, efficient diff methods, etc.

Syntax:

ReactDOM.render(element, container, callback)

Parameters: This method can take a maximum of three parameters as described below.

- **element:** This parameter expects a JSX expression or a React Element to be rendered.
- **container:** This parameter expects the container in which the element has to be rendered.
- **callback:** This is an optional parameter that expects a function that is to be executed once the render is complete.

Return Type: This function returns a reference to the component or null if a stateless component was rendered.

2. findDOMNode() Function

This function is generally used to get the DOM node where a particular React component was rendered. This method is very less used like the following can be done by adding a ref attribute to each component itself.

Syntax:
ReactDOM.findDOMNode(component)

Parameters: This method takes a single parameter component that expects a React Component to be searched in the Browser DOM.
Return Type: This function returns the DOM node where the component was rendered on success otherwise null.

3. unmountComponentAtNode() Function

This function is used to unmount or remove the React Component that was rendered to a particular container. As an example, you may think of a notification component, after a brief amount of time it is better to remove the component making the web page more efficient.

Syntax:
ReactDOM.unmountComponentAtNode(container)

Parameters: This method takes a single parameter container which expects the DOM container from which the React component has to be removed.
Return Type: This function returns true on success otherwise false.

4. hydrate() Function

This method is equivalent to the render() method but is implemented while using server-side rendering.

Syntax:
ReactDOM.hydrate(element, container, callback)

Parameters: This method can take a maximum of three parameters as described below.

- **element:** This parameter expects a JSX expression or a React Component to be rendered.
- **container:** This parameter expects the container in which the element has to be rendered.
- **callback:** This is an optional parameter that expects a function that is to be executed once the render is complete.

Return Type: This function attempts to attach event listeners to the existing markup and returns a reference to the component or null if a stateless component was rendered.

5. createPortal() Function

Usually, when an element is returned from a component's render method, it's mounted on the DOM as a child of the nearest parent node which in some cases may not be desired. Portals allow us to render a component into a DOM node that resides outside the current DOM hierarchy of the parent component.

Syntax:
ReactDOM.createPortal(child, container)

Parameters: This method takes two parameters as described below.

- **child:** This parameter expects a JSX expression or a React Component to be rendered.
- **container:** This parameter expects the container in which the element has to be rendered.

Return Type: This function returns nothing.

Important Points to Note:

- ReactDOM.render() replaces the child of the given container if any. It uses a highly efficient diff algorithm and can modify any subtree of the DOM.
- findDOMNode() function can only be implemented upon mounted components thus Functional components can not be used in findDOMNode() method.
- ReactDOM uses observables thus provides an efficient way of DOM handling.
- ReactDOM can be used on both the client-side and server-side.

JSX

React is a declarative, efficient, and flexible JavaScript library for building user interfaces. But instead of using regular JavaScript, React code should be written in something called JSX.

sample JSX code:

```
const ele = <h1>This is sample JSX</h1>;
```

The above code snippet somewhat looks like HTML and it also uses a JavaScript-like variable but is neither HTML nor JavaScript, it is JSX.

JSX is basically a syntax extension of regular JavaScript and is used to create React elements. These elements are then rendered to the React DOM.

Why JSX?

- It is faster than normal JavaScript as it performs optimizations while translating to regular JavaScript.
- It makes it easier for us to create templates.
- Instead of separating the markup and logic in separated files, React uses *components* for this purpose. We will learn about components in detail in further articles.

Using JavaScript expressions in JSX: In React we are allowed to use normal JavaScript expressions with JSX. To embed any JavaScript expression in a piece of code written in JSX we will have to wrap that expression in curly braces {}. Consider the below program, written in the **index.js** file:

```
import React from 'react';
import ReactDOM from 'react-dom';
const name = "Learner";
const element = <h1>Hello,{ name }.Welcome to MAMCE.< /h1>;
ReactDOM.render( element,document.getElementById("root"));
```

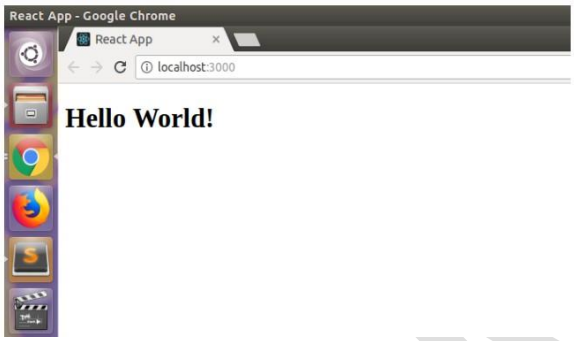
OUTPUT:

Hello, Learner.Welcome to MAMCE.

In the above program we have embedded the javascript expression const name = “Learner”; in our JSX code. We embed the use of any JavaScript expression in JSX by wrapping them in curly braces except if-else statements. But we can use conditional statements instead of if-else statements in JSX. Below is the example where conditional expressing is embedded in JSX:

```
import React from 'react';
import ReactDOM from 'react-dom';
let i = 1;
const element = <h1>{ (i == 1) ? 'Hello World!' : 'False!' } < /h1>;
ReactDOM.render( element,document.getElementById("root"));
```

OUTPUT:



In the above example, the variable i is checked if for the value 1. As it equals 1 so the string ‘Hello World!’ is returned to the JSX code. If we modify the value of the variable i then the string ‘False’ will be returned.

Attributes in JSX:

JSX allows us to use attributes with the HTML elements just like we do with normal HTML. But instead of the normal naming convention of HTML, JSX uses camelcase convention for attributes.

For example, *class* in HTML becomes *className* in JSX. The main reason behind this is that some attribute names in HTML like ‘class’ are reserved keywords in JavaScripts. So, in order to avoid this problem, JSX uses the camel case naming convention for attributes. We can also use custom attributes in JSX. For custom attributes, the names of such attributes should be prefixed by **data-**. In the below example, we have used a custom attribute with name **data-sampleAttribute** for the <h2> tag.

```
import React from 'react';
import ReactDOM from 'react-dom';
const element = <div><h1 className = "hello">Hello MCA</h1>
                    <h2 data-sampleAttribute="sample">Custom attribute</h2>
< /div>;
ReactDOM.render( element,document.getElementById("root"));
```

Specifying attribute values:

JSX allows us to specify attribute values in two ways:

1. **As for string literals:** We can specify the values of attributes as hard-coded strings using quotes:

```
const ele = <h1 className = "firstAttribute">Hello!</h1>;
```

1. **As expressions:** We can specify attributes as expressions using curly braces {}:

```
const ele = <h1 className = {varName}>Hello!</h1>;
```

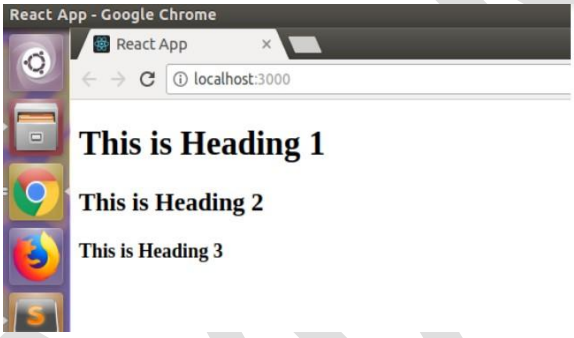
Wrapping elements or Children in JSX:

Consider a situation where you want to render multiple tags at a time. To do this we need to wrap all of this tag under a parent tag and then render this parent element to the HTML. All the subtags are called child tags or children of this parent element.

Notice in the below example how we have wrapped h1, h2, and h3 tags under a single div element and rendered them to HTML:

```
import React from 'react';
import ReactDOM from 'react-dom';
const element = <div><h1>This is Heading 1 < /h1>
                    <h2>This is Heading 2</h2 >
                    <h3>This is Heading 3 < /h3>
                    </div > ;
ReactDOM.render( element,document.getElementById("root"));
```

OUTPUT:



Comments in JSX:

JSX allows us to use comments as it allows us to use JavaScript expressions. Comments in JSX begins with **/*** and ends with ***/**. We can add comments in JSX by wrapping them in curly braces {} just like we did in the case of expressions.

Below example shows how to add comments in JSX:

```
import React from 'react';

import ReactDOM from 'react-dom';

const element = <div><h1>Hello World !</h1>

                    {/ * This is a comment in JSX * /}

                </div>;

ReactDOM.render( element,document.getElementById("root"));
```

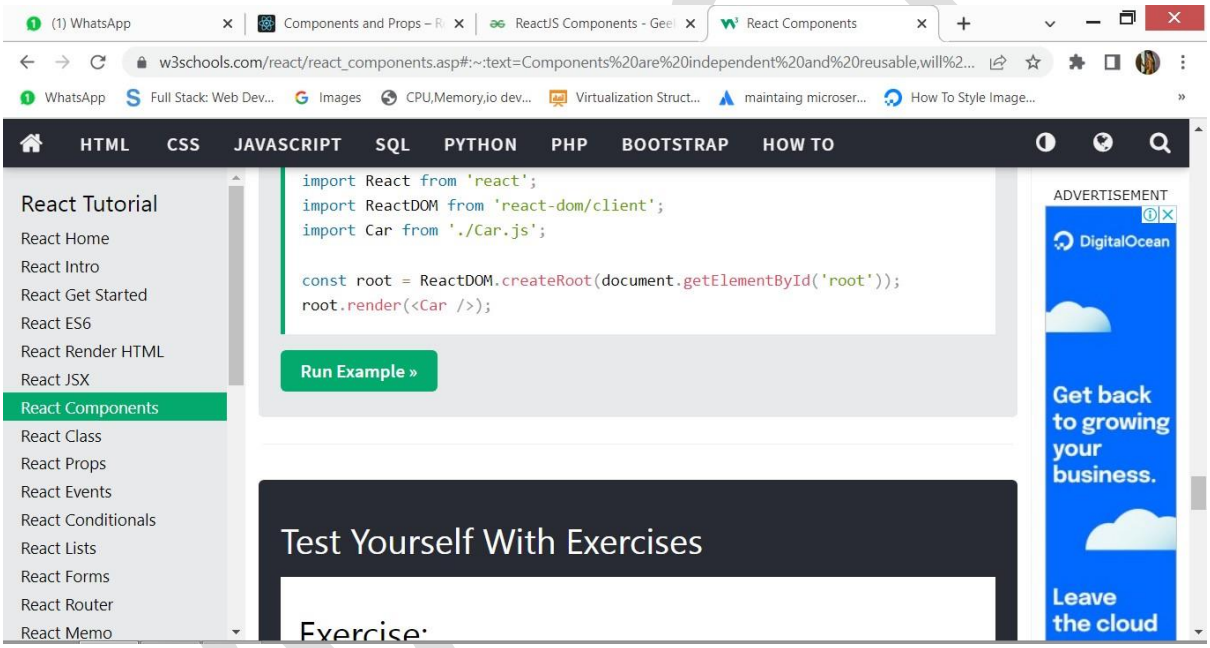
Components

React Components

Components are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and return HTML.

A **Component** is one of the core building blocks of React. In other words, we can say that every application you will develop in React will be made up of pieces called components.

Components make the task of building UIs much easier. You can see a UI broken down into multiple individual pieces called components and work on them independently and merge them all in a parent component which will be your final UI.



Google's custom search at the top can be seen as an individual component, the navigation bar can be seen as an individual component, the sidebar is an individual component, the list of articles or post is also an individual component and finally, we can merge all of these individual components to make a parent component which will be the final UI for the homepage.
Components in React basically return a piece of JSX code that tells what should be rendered on the screen.

1. **Functional Components**: Functional components are simply javascript functions. We can create a functional component in React by writing a javascript function. These functions may or may not receive data as parameters, Below example shows a valid functional component in React:

```
const Democomponent=()=>
{
  return <h1>Welcome Message!</h1>;
}
```

2. **Class Components**: The class components are a little more complex than the functional components. The functional components are not aware of the other components in your program whereas the class components can work with each other. We can pass data from one class component to other class components. We can use JavaScript ES6 classes to create class-based components in React. Below example shows a valid class-based component in React:

```
class Democomponent extends React.Component
{
  render(){
      return <h1>Welcome Message!</h1>;
  }
}
```

The components we created in the above two examples are equivalent, and we also have stated the basic difference between a functional component and class component

Rendering Components

React is also capable of rendering user-defined components. To render a component in React we can initialize an element with a user-defined component and pass this element as the first parameter to ReactDOM.render() or directly pass the component as the first argument to the ReactDOM.render() method.
Below syntax shows how to initialize a component to an element:

```
const elementName = <ComponentName />;
```

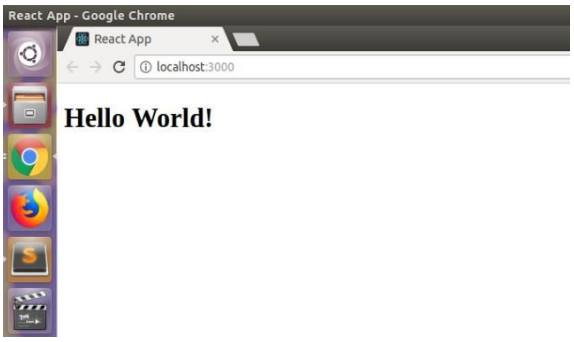
In the above syntax, the *ComponentName* is the name of the user-defined component.

The name of a component should always start with a capital letter. This is done to differentiate a component tag with html tags.
Below example renders a component named Welcome to the screen:
Open your **index.js** file from your project directory, and make the given below changes:
src index.js:

- javascript

```
import React from 'react';
import ReactDOM from 'react-dom';
 // This is a functional component
const Welcome=()=>
{
    return <h1>Hello World!</h1>
}
 ReactDOM.render(
  <Welcome />,
  document.getElementById("root")
);
```

Output:



Let us see step-wise what is happening in the above example:

1. We call the ReactDOM.render() as the first parameter.
2. React then calls the component Welcome, which returns <h1>Hello World!</h1>; as the result.
3. Then the ReactDOM efficiently updates the DOM to match with the returned element and renders that element to the DOM element with id as "root".

Properties

React enables developers to create dynamic and advanced component using properties. Every component can have attributes similar to HTML attributes and each attribute's value can be accessed inside the component using properties (props).

Props are arguments passed into React components.

Props are passed to components via HTML attributes.

props stands for properties.

For example, *Hello* component with a name attribute can be accessed inside the component through this.props.name variable.

<Hello name="React" />
// value of name will be "Hello* const name = this.props.name

React properties supports attribute's value of different types. They are as follows,

- String
- Number
- Datetime
- Array
- List
- Objects

React Props

- Props stand for "Properties." They are read-only components. It is an object which stores the value of attributes of a tag and work similar to the HTML attributes.

- It gives a way to pass data from one component to other components. It is similar to function arguments. Props are passed to the component in the same way as arguments passed in a function.
- Props are immutable so we cannot modify the props from inside the component. Inside the components, we can add attributes called props. These attributes are available in the component as this.props and can be used to render dynamic data in our render method.
- When you need immutable data in the component, you have to add props to reactDom.render() method in the main.js file of your ReactJS project and used it inside the component in which you need.

```
import React from 'react';
import ReactDOM from 'react-dom/client';
function Car(props) {
 return <h2>I am a { props.brand }!</h2>;
}
const myElement = <Car brand="Ford" />;
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

output:

Fetch API

What is fetch API in JavaScript?

The Fetch API **provides a JavaScript interface for accessing and manipulating parts of the HTTP pipeline, such as requests and responses**. It also provides a global fetch() method that provides an easy, logical way to fetch resources asynchronously across the network.

ReactJS: ReactJS is a declarative, efficient, and flexible JavaScript library for building user interfaces. It's 'V' in MVC. ReactJS is an open-source, component-based front-end library responsible only for the view layer of the application. It is maintained by Facebook.

API: API is an abbreviation for Application Programming Interface which is a collection of communication protocols and subroutines used by various programs to communicate between them. A programmer can make use of various API tools to make its program easier and simpler. Also, an API facilitates the programmers with an efficient way to develop their software programs.

Approach: used the API endpoint
from http://jsonplaceholder.typicode.com/users we have created the component in App.js and styling the component in App.css. From the API we have target "id", "name", "username", "email" and fetch the data from API endpoints.

Below is the stepwise implementation of how we fetch the data from an API in react. We will use the fetch function to get the data from the API.
Step by step implementation to fetch data from an api in react.

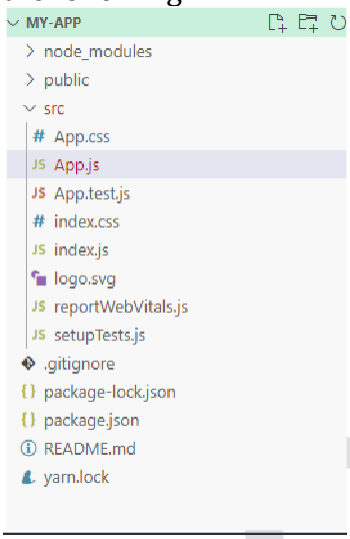
- **Step 1:** Create React Project
  npm create-react-app MY-APP
- **Step 2:** Change your directory and enter your main folder charting as
  cd MY-APP
- **Step 3:** API endpoint
  https://jsonplaceholder.typicode.com/users


```
// 20210815205448
// https://jsonplaceholder.typicode.com/users

[
  {
    "id": 1,
    "name": "Leanne Graham",
    "username": "Bret",
    "email": "Sincere@april.biz",
    "address": {
      "street": "Kulas Light",
      "suite": "Apt. 556",
      "city": "Gwenborough",
      "zipcode": "92998-3874",
      "geo": {
        "lat": "-37.3159",
        "lng": "81.1496"
      }
    },
    "phone": "1-770-736-8031 x56442",
    "website": "hildegard.org",
    "company": {
      "name": "Romaguera-Crona",
      "catchPhrase": "Multi-layered client-server
      "bs": "harness real-time e-markets"
    }
  },
```


API

- **Step 4:** Write code in App.js to fetch data from API and we are using fetch function.

Project Structure: It will look the following.



Project Structure

Example:

App.js

```
import React from "react";
import './App.css';
class App extends React.Component {
        // Constructor
        constructor(props) {
                super(props);
this.state = {
                        items: [],
                        DataisLoaded: false
                };
        }
        // ComponentDidMount is used to
        // execute the code
        componentDidMount() {
                fetch("https://jsonplaceholder.typicode.com/users")
                        .then((res) => res.json())
                        .then((json) => {
                                this.setState({
                                        items: json,
                                        DataisLoaded: true
                                });
                        })
        }
        render() {
                const { DataisLoaded, items } = this.state;
                if (!DataisLoaded) return <div>
                        <h1> Pleses wait some time ....</h1> </div> ;
                return (
                <div className = "App">
                        <h1> Fetch data from an api in react </h1> {
```

```
                        items.map((item) => (
                        <ol key = { item.id } >
                                User_Name: { item.username },
                                Full_Name: { item.name },
                                User_Email: { item.email }
                                </ol>
                        ))
                    }
                </div>
        );
}
}
export default App;
```

Write code in App.css for styling the app.js file.

App.CSS

```
.App {
        text-align: center;
        color: Green;
}
.App-header {
        background-color: #282c34;
        min-height: 100vh;
        display: flex;
        flex-direction: column;
        align-items: center;
        justify-content: center;
        font-size: calc(10px + 2vmin);
        color: white;
}
.App-link {
        color: #61dafb;
}
@keyframes App-logo-spin {
        from {
                transform: rotate(0deg);
        }
        to {
                transform: rotate(360deg);
        }
}
```

Step to run the application: Open the terminal and type the following command.
npm start
Output: Open the browser and our project is shown in the
URL **http://localhost:3000/**

Using the JavaScript Fetch API

The Fetch API through the fetch() method allows us to make an HTTP request to the backend. With this method, we can perform different types of operations using HTTP methods like the GET method to request data from an endpoint, POST to send data to an endpoint, and more.

Since we are fetching data, our focus is the GET method.

fetch() requires the URL of the resource we want to fetch and an optional parameter:

fetch(url, options)

We can also specify the HTTP method in the optional parameter. For the GET method, we have the following:

```
fetch(url, {
  method: "GET" // default, so we can ignore
})
```

Or, we can simply ignore the optional parameter because GET is the default:

```
fetch(url)
```

As mentioned earlier, we will fetch data from a REST API. We could use any API, but here we will use a free online API called JSONPlaceholder to fetch a list of posts into our application; here is a list of the resources we can request

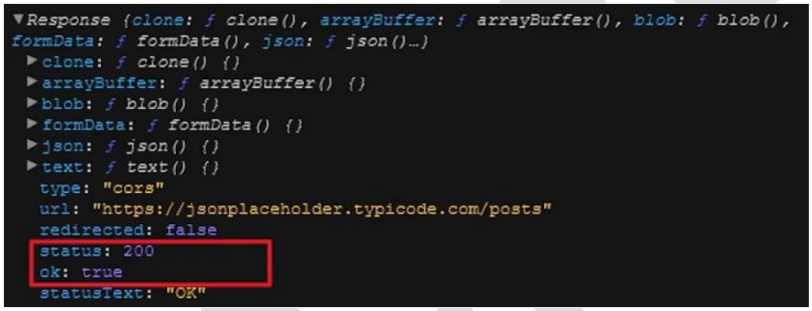
By applying what we've learned so far, a typical fetch() request looks like the following:

```
import { useState, useEffect } from "react";
export default function App() {
 const [data, setData] = useState(null);
 const [loading, setLoading] = useState(true);
 const [error, setError] = useState(null);

 useEffect(() => {
  fetch(`https://jsonplaceholder.typicode.com/posts`)
```

```
  .then((response) => console.log(response));
 }, []);
 return <div className="App">App</div>;
}
```

In the code, we are using the fetch() method to request post data from the resource endpoint as seen in the useEffect Hook. This operation returns a promise that could either resolve or reject.

If it resolves, we handle the response using .then(). But at this stage, the returned data is a Response object, which is not the actual format that we need, although it is useful to check for the HTTP status and to handle errors.



State and Lifecycle

What is state in react JS?

The state is **a built-in React object that is used to contain data or information about the component**. A component's state can change over time; whenever it changes, the component re-renders.

React doesn't recommend using multiple renders instead it uses a stateful approach where the page is re-rendered once a state is altered.

What are the lifecycle of react JS?

Each component in React has a lifecycle which you can monitor and manipulate during its three main phases.

The three phases are: 1. **Mounting 2. Updating 3. Unmounting**.

1.Mounting

Mounting means putting elements into the DOM.

React has four built-in methods that gets called, in this order, when mounting a component:

1. constructor()
2. getDerivedStateFromProps()
3. render()
4. componentDidMount()

The render() method is required and will always be called, the others are optional and will be called if you define them.

constructor

The constructor() method is called before anything else, when the component is initiated, and it is the natural place to set up the initial state and other initial values.

The constructor() method is called with the props, as arguments, and you should always start by calling the super(props) before anything else, this will initiate the parent's constructor method and allows the component to inherit methods from its parent (React.Component).

```
import React from 'react';
import ReactDOM from 'react-dom/client';
class Header extends React.Component {
constructor(props) {
  super(props);
  this.state = {favoritecolor: "red"};
 }
 render() {
  return (
    <h1>My Favorite Color is {this.state.favoritecolor}</h1>
  );
 }
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Header />);
```

OUTPUT:

getDerivedStateFromProps

The getDerivedStateFromProps() method is called right before rendering the element(s) in the DOM.

This is the natural place to set the state object based on the initial props.

It takes state as an argument, and returns an object with changes to the state.

The example below starts with the favorite color being "red", but the getDerivedStateFromProps() method updates the favorite color based on the favcol attribute:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
class Header extends React.Component {
constructor(props) {
  super(props);
  this.state = {favoritecolor: "red"};
 }
 static getDerivedStateFromProps(props, state) {
  return {favoritecolor: props.favcol };
 }
 render() {
  return (
    <h1>My Favorite Color is {this.state.favoritecolor}</h1>
  );
 }
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Header favcol="yellow"/>);
```

OUTPUT:

render

The render() method is required, and is the method that actually outputs the HTML to the DOM.

```
import React from 'react';
import ReactDOM from 'react-dom/client';
class Header extends React.Component {
render() {
  return (
    <h1>This is the content of the Header component</h1>
  );
 }}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Header />);
```

OUTPUT:

componentDidMount

The componentDidMount() method is called after the component is rendered.

This is where you run statements that requires that the component is already placed in the DOM.

```
import React from 'react';
import ReactDOM from 'react-dom/client';
class Header extends React.Component {
constructor(props) {
  super(props);
  this.state = {favoritecolor: "red"};
 }
 componentDidMount() {
  setTimeout(() => {
   this.setState({favoritecolor: "yellow"})
  }, 1000)
 }
 render() {
  return (
   <h1>My Favorite Color is {this.state.favoritecolor}</h1>
  );
 }
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Header />);
```

OUTPUT:

2.Updating

A component is updated whenever there is a change in the component's state or props.

React has five built-in methods that gets called, in this order, when a component is updated:

1. getDerivedStateFromProps()
2. shouldComponentUpdate()
3. render()
4. getSnapshotBeforeUpdate()
5. componentDidUpdate()

The render() method is required and will always be called, the others are optional and will be called if you define them.

getDerivedStateFromProps

Also at *updates* the getDerivedStateFromProps method is called. This is the first method that is called when a component gets updated.

This is still the natural place to set the state object based on the initial props.

The example below has a button that changes the favorite color to blue, but since the getDerivedStateFromProps() method is called, which updates the state with the color from the favcol attribute, the favorite color is still rendered as yellow:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
class Header extends React.Component {
constructor(props) {
  super(props);
  this.state = {favoritecolor: "red"};
 }
 static getDerivedStateFromProps(props, state) {
  return {favoritecolor: props.favcol };
 }
 changeColor = () => {
  this.setState({favoritecolor: "blue"});
 }
 render() {
  return (
    <div>
    <h1>My Favorite Color is {this.state.favoritecolor}</h1>
    <button type="button" onClick={this.changeColor}>Change color</button>
    </div>
  ); }}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Header favcol="yellow" />);
```

OUTPUT:

shouldComponentUpdate

In the shouldComponentUpdate() method you can return a Boolean value that specifies whether React should continue with the rendering or not.

The default value is true.

The example below shows what happens when the shouldComponentUpdate() method returns false:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
class Header extends React.Component {
constructor(props) {
  super(props);
  this.state = {favoritecolor: "red"};
 }
 shouldComponentUpdate() {
  return false;
 }
 changeColor = () => {
  this.setState({favoritecolor: "blue"});
 }
 render() {
  return (
    <div>
    <h1>My Favorite Color is {this.state.favoritecolor}</h1>
    <button type="button" onClick={this.changeColor}>Change color</button>
    </div>
  );
 }
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Header />);
```

OUTPUT:

render

The render() method is of course called when a component gets *updated*, it has to re-render the HTML to the DOM, with the new changes.

The example below has a button that changes the favorite color to blue:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
class Header extends React.Component {
constructor(props) {
  super(props);
  this.state = {favoritecolor: "red"};
 }
 changeColor = () => {
  this.setState({favoritecolor: "blue"});
 }
 render() {
  return (
    <div>
    <h1>My Favorite Color is {this.state.favoritecolor}</h1>
    <button type="button" onClick={this.changeColor}>Change color</button>
    </div>
  );
 }
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Header />);
```

OUTPUT:

getSnapshotBeforeUpdate

In the getSnapshotBeforeUpdate() method you have access to the props and state *before* the update, meaning that even after the update, you can check what the values were *before* the update.

If the getSnapshotBeforeUpdate() method is present, you should also include the componentDidUpdate() method, otherwise you will get an error.

The example below might seem complicated, but all it does is this:

When the component is *mounting* it is rendered with the favorite color "red".

When the component *has been mounted,* a timer changes the state, and after one second, the favorite color becomes "yellow".

This action triggers the *update* phase, and since this component has a getSnapshotBeforeUpdate() method, this method is executed, and writes a message to the empty DIV1 element.

Then the componentDidUpdate() method is executed and writes a message in the empty DIV2 element:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
class Header extends React.Component {
constructor(props) {
  super(props);
  this.state = {favoritecolor: "red"};
 }
 componentDidMount() {
  setTimeout(() => {
    this.setState({favoritecolor: "yellow"})
  }, 1000)
 }
 getSnapshotBeforeUpdate(prevProps, prevState) {
  document.getElementById("div1").innerHTML =
  "Before the update, the favorite was " + prevState.favoritecolor;
 }
 componentDidUpdate() {
  document.getElementById("div2").innerHTML =
  "The updated favorite is " + this.state.favoritecolor;
 }
 render() {
  return (
    <div>
    <h1>My Favorite Color is {this.state.favoritecolor}</h1>
    <div id="div1"></div>
    <div id="div2"></div>
```

```
    </div>
  );
 }
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Header />);
```

OUTPUT:

componentDidUpdate

The componentDidUpdate method is called after the component is updated in the DOM.

The example below might seem complicated, but all it does is this:

When the component is *mounting* it is rendered with the favorite color "red".

When the component *has been mounted,* a timer changes the state, and the color becomes "yellow".

This action triggers the *update* phase, and since this component has a componentDidUpdate method, this method is executed and writes a message in the empty DIV element:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
class Header extends React.Component {
constructor(props) {
  super(props);
  this.state = {favoritecolor: "red"};
 }
 componentDidMount() {
  setTimeout(() => {
   this.setState({favoritecolor: "yellow"})
  }, 1000)
 }
 componentDidUpdate() {
  document.getElementById("mydiv").innerHTML =
```

```
    "The updated favorite is " + this.state.favoritecolor;
  }
  render() {
    return (
      <div>
      <h1>My Favorite Color is {this.state.favoritecolor}</h1>
      <div id="mydiv"></div>
      </div>
    );
  }
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Header />);
```

OUTPUT:

3.Unmounting

The next phase in the lifecycle is when a component is removed from the DOM, or *unmounting* as React likes to call it.

React has only one built-in method that gets called when a component is unmounted:

- componentWillUnmount()

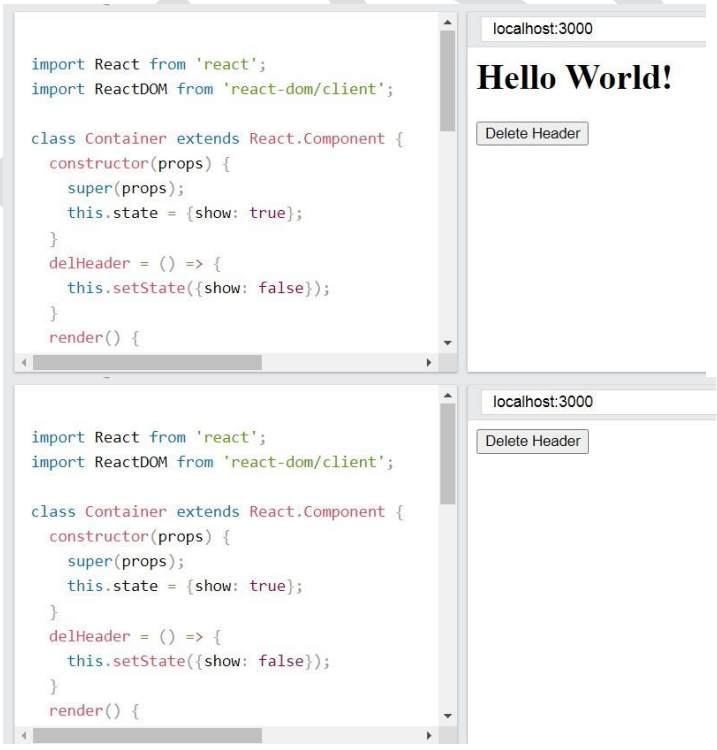
3.1componentWillUnmount

The componentWillUnmount method is called when the component is about to be removed from the DOM.

```
import React from 'react';
import ReactDOM from 'react-dom/client';
class Container extends React.Component {
constructor(props) {
  super(props);
  this.state = {show: true};
}
 delHeader = () => {
  this.setState({show: false});
```

```
  }
  render() {
    let myheader;
    if (this.state.show) {
      myheader = <Child />;
    };
    return (
      <div>
      {myheader}
      <button type="button" onClick={this.delHeader}>Delete Header</button>
      </div>
    );
  }
}
class Child extends React.Component {
  componentWillUnmount() {
    alert("The component named Header is about to be unmounted.");
  }
  render() {
    return (
      <h1>Hello World!</h1>
    );
  }
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Container />);
```

OUTPUT:



JS Localstorage

JavaScript localStorage

LocalStorage is a data storage type of web storage. This allows the JavaScript sites and apps to store and access the data without any expiration date. This means that the data will always be persisted and will not expire. So, data stored in the browser will be available even after closing the browser window.

In short, all we can say is that the localStorage holds the data with no expiry date, which is available to the user even after closing the browser window. It is useful in various ways, such as remembering the shopping cart data or user login on any website.

In the past days, cookies were the only option to remember this type of temporary and local information, but now we have localStorage as well. Local storage comes with a higher storage limit than cookies (5MB vs 4MB). It also does not get sent with every HTTP request. So, it is a better choice now for client-side storage. Some essential points of localStorage need to be noted:

- localStorage is not secure to store sensitive data and can be accessed using any code. So, it is quite insecure.
- It is an advantage of localStorage over cookies that it can store more data than cookies. You can store 5MB data on the browser using localStorage.
- localStorage stores the information only on browser instead in database. Thereby the localStorage is not a substitute for a server-based database.
- localStorage is synchronous, which means that each operation executes one after another.

localStorage Methods

The localStorage offers some methods to use it. Before that, a basic overview of these methods are as follows:

Methods	Description
setItem()	This method is used to add the data through key and value to localStorage.
getItem()	It is used to fetch or retrieve the value from the storage using the key.
removeItem()	It removes an item from storage by using the key.
clear()	It is used to gets clear all the storage.

Each of these methods is used with localStorage keyword connecting with dot(.) character.

For Example: localStorage.setItem().

LocalStorage property is read-only.

Following some codes given, which are used to add, retrieve, remove, and clear the data in localStorage. Use them in your code accordingly whenever needed. You need a key-value pair to add some data in localStorage. So, let key is city and its value is Noida, **i.e.,** key: value = city: Noida.

Add data

To add the data in localStorage, both key and value are required to pass in setItem() function.

```
localStorage.setItem("city", "Noida");
```

Retrieve data

It requires only the key to retrieve the data from storage and a JavaScript variable to store the returned data.

```
const res = localStorage.getItem("city");
```

Remove data

It also requires only the key to remove the value attached with it.

```
localStorage.removeItem("city");
```

Clear localStorage

It is a simple clear() function of localStorage, which is used to remove all the localStorage data:

```
localStorage.clear()
```

Advantage of localStorage

- The data collected by localStorage is stored in the browser. You can store 5 MB data in the browser.
- There is no expiry date of data stored by localStorage.
- You can remove all the localStorage item by a single line code, i.e., **clear()**.
- The localStorage data persist even after closing the browser window, like items in shopping cart.
- It also has advantages over cookies because it can store more data than cookies.

Browser compatibility

The localStorage has specified in HTML 5, which is supported by several browsers, like Chrome. Below is a list of different browsers and their versions that supports JavaScript localStorage.

Browser	Chrome	Internet Explorer	Firefox	Opera	Safari	Edge
Version support	4.0	8.0	3.5	11.5	4	12

Example

It is a basic example of adding a key and value to localStorage and retrieving back by the key. See the code below how localStorage methods work:

```
<html>
<body>
<div id="result"></div>
<script>
// Check browser support
if (typeof(Storage) !== "undefined") {
  // Store an item to localStorage
  localStorage.setItem("firstname", "Alen");
  // Retrieve the added item
  document.getElementById("result").innerHTML = localStorage.getItem("firstname");
}
else {
  //display this message if browser does not support localStorage
  document.getElementById("result").innerHTML = "Sorry, your browser does not support Web Storage.";
}
</script> </body></html>
```

Output

Alen

Clear all records

Clear() method of localStorage is used to clear the entire storage data. When this method invokes, it clears all the records for that domain from the storage. It does not contain any parameters. See the syntax to clear the localStorage:

window.localStorage.clear();

Or

localStorage.clear();

We will use this clear code in below example.

Check localStorage

On the JavaScript console, you can check what is in local storage by typing **localStorage** command on it. Even if there nothing in localStorage, it has length =

0 inside it.

Command

LocalStorage

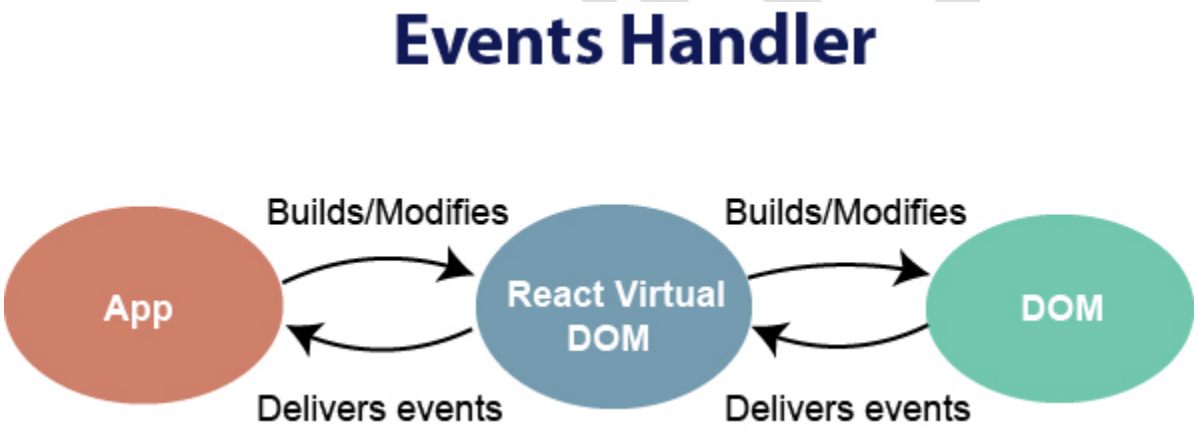
Output

```
Storage {length: 0}
```

Events

An event is an action that could be triggered as a result of the user action or system generated event. For example, a mouse click, loading of a web page, pressing a key, window resizes, and other interactions are called events.

React has its own event handling system which is very similar to handling events on DOM elements. The react event handling system is known as Synthetic Events. The synthetic event is a cross-browser wrapper of the browser's native event.



Handling events with react have some syntactic differences from handling events on DOM. These are:

1. React events are named as **camelCase** instead of **lowercase**.
2. With JSX, a function is passed as the **event handler** instead of a **string**. For example:

Event declaration in plain HTML:

```
<button onclick="showMessage()">
    Hello MCA
</button>
```

Event declaration in React:

```
<button onClick={showMessage}>
    Hello MCA
</button>
```

3. In react, we cannot return **false** to prevent the **default** behavior. We must call **preventDefault** event explicitly to prevent the default behavior. For example:

In plain HTML, to prevent the default link behavior of opening a new page, we can write:

```
<a href="#" onclick="console.log('You had clicked a Link.'); return false">
   Click_Me
</a>
```

In React, we can write it as:

```
function ActionLink() {
   function handleClick(e) {
      e.preventDefault();
   console.log('You had clicked a Link.');5.
   }
   return (
      <a href="#" onClick={handleClick}>
      Click_Me9.
      </a>
   );
}
```

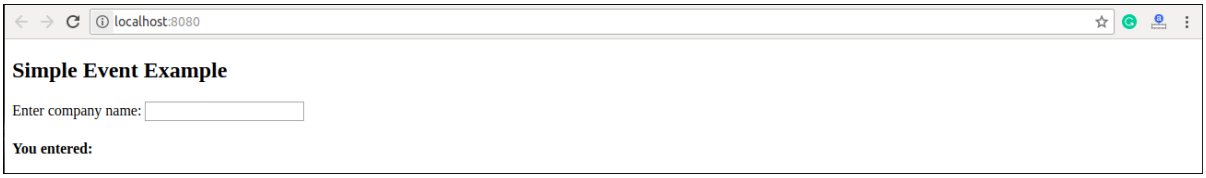
Example

In the below example, we have used only one component and adding an onChange event. This event will trigger the **changeText** function, which returns the company name.

```
import React, { Component } from 'react';
class App extends React.Component {
   constructor(props) {
      super(props);
      this.state = {
      companyName: ''7.
      };
   }
   changeText(event) {
      this.setState({
      companyName: event.target.value
      });
   }
   render() {
      return (
         <div>
            <h2>Simple Event Example</h2>
            <label htmlFor="name">Enter company name: </label>
<input type="text" id="companyName" onChange={this.changeText.bind(this)}/>
            <h4>You entered: { this.state.companyName }</h4>
      </div>
      );
   }
}
export default App;
```

Output

When you execute the above code, you will get the following output.



After entering the name in the textbox, you will get the output as like below screen.

Just like HTML DOM events, React can perform actions based on user events.

React has the same events as HTML: click, change, mouseover etc.

Adding Events

React events are written in camelCase syntax:

onClick instead of onclick.

React event handlers are written inside curly braces:

onClick={shoot} instead of onClick="shoot()".

React:

```
<button onClick={shoot}>Take the Shot!</button>
```

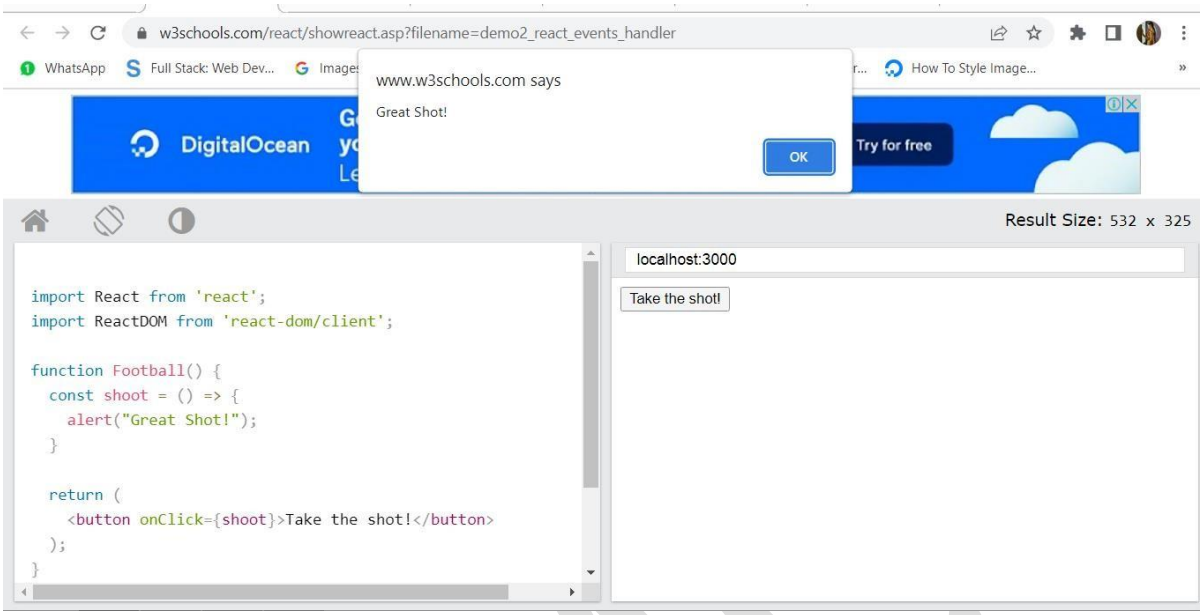
HTML:

```
<button onclick="shoot()">Take the Shot!</button>
```

EXAMPLE:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
function Football() {
 const shoot = () => {
  alert("Great Shot!");
 }
 return (
  <button onClick={shoot}>Take the shot!</button>
 );
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Football />);
```

OUTPUT:



Lifting State Up

Lifting up the State: As we know, every component in React has its own state. Because of this sometimes data can be redundant and inconsistent. So, by Lifting up the state we make the state of the parent component as a single source of truth and pass the data of the parent in its children.

Time to use Lift up the State: If the data in "parent and children components" or in "cousin components" is Not in Sync.

Example 1: If we have 2 components in our App. **A -> B** where, A is parent of B. keeping the same data in both Component A and B might cause inconsistency of data.

Example 2: If we have 3 components in our App.

A

/ \

B C

Where A is the parent of B and C. In this case, If there is some Data only in component B but, component C also wants that data. We know Component C cannot access the data because a component can talk only to its parent or child (Not cousins).

Problem: Let's Implement this with a simple but general example. We are considering the second example.

Complete File Structure:

Approach: To solve this, we will Lift the state of component B and component C to component A. Make A.js as our Main Parent by changing the path of App in the index.js file

Before:
import App from './App';

After:
import App from './A';

Filename- A.js:

```
import React,{ Component } from 'react';
import B from './B'
import C from './C'
class A extends Component {
constructor(props) {
super(props);
this.handleTextChange =
this.handleTextChange.bind(this);
  this.state = {text: ''};
 }
 handleTextChange(newText) {
  this.setState({text: newText});
 }
 render() {
  return (
     <React.Fragment>
      <B text={this.state.text}
        handleTextChange={this.handleTextChange}/>
      <C text={this.state.text} />
     </React.Fragment>
  );
 }}
export default A;
```

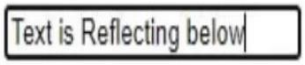
Filename- B.js:

```
import React,{ Component } from 'react';
class B extends Component {
constructor(props) {
        super(props);
        this.handleTextChange = this.handleTextChange.bind(this);
}
handleTextChange(e){
        this.props.handleTextChange(e.target.value);
}
render() {
        return (
                <input value={this.props.text}
                        onChange={this.handleTextChange} />
        ); }}
export default B;
```

Filename- C.js:

```
import React,{ Component } from 'react';
class C extends Component {
render() {
        return (
                <h3>Output: {this.props.text}</h3>
        );
}
}
export default C;
```

Output: ***Now, component C can Access text in component B through component A.***



Output: Text is Reflecting below

Composition and Inheritance

React Composition is a development pattern based on React's original component model where we build components from other components using explicit defined props or the implicit children prop.

Composition Patterns

1. Lifting State & Container/Presenter
2. Higher-Order Component (HOC)
3. Render Prop/Function-as-Child
4. "Renderless" State Provider

1.Lifting State & Container/Presenter

The simplest way to compose React components is to follow these rules:

1. Divide components into stateful "*containers*" and stateless "*presenters*".
2. Pass functions ("callbacks") that change the container state as props to children
3. If two components need access to the same state, move the state into their common parent

EXAMPLE:

```
class Container extends Component {
 state = {
  text: 'foo'
 }
 changeText = newText => this.setState({text: newText})
 render() {
  <div>
   <Presenter1 text={this.state.text} changeText={this.changeText} />
```

```
    <Presenter2 text={this.state.text} />
   <div>
  }
}
```

2. Higher-Order Component (HOC)

A higher-order component is a function that takes a component and returns a component. One use case is to inject additional props or context.

Think of an HOC as a component factory or a stage in a "component assembly line".

```
// JSX
const withFooProp = WrappedComponent => props => <WrappedComponent foo={2}
{...props} />
// createElement
const withFooProp = WrappedComponent => props => h(WrappedComponent, { foo: 2,
...props })
const Bar // ... a component
export default withFooProp(Bar) // export an augmented version of Bar
// elsewhere...
import Bar from './bar'
// props {baz: 2, foo: 2}
<Bar baz={2} />
```

Without JSX it is technically possible to use HOCs inline as well, but this will cause a re-paint every render because a new component class is created each time and React uses the identity of the component constructor as part of its reconciliation.

Examples

- React-Redux uses an HOC called connect to map store state to props
- React-Router's withRouter HOC provides route context to components needing access to history APIs

Disadvantages

- Creates a wrapper around components, allocating a new function and taking up space in the tree when debugging
- Higher-order function composition doesn't always work inline with JSX, depending on what you're trying to do.

3. Render Prop/Function-as-Child

Components that use the "render prop" pattern invert control of rendering from the component itself to the consumer by accepting the render function as a prop. This prop is commonly called "render", although since JSX children are turned into the 3rd argument to createElement, the children prop can be repurposed as a render prop, too (the function-as-child pattern).

FileName: function-as-child.js

```
const InjectTimestamp = ({children, ...props}) => {
  const time = new Date()
  return children(time)
}
InjectTimestamp.propTypes = {
  children: PropTypes.func,
}
const Foo = () => {
  return (
    <div>
      <InjectTimestamp>
        { time => <div>time.toLocaleString()</div> }
      </InjectTimestamp>
    </div>
  )
}
```

FileName: render-prop.js

```
const InjectTimestamp = ({render, ...props}) => {
  const time = new Date()
  return render(time)
}
InjectTimestamp.propTypes = {
  render: PropTypes.func,
}
const Foo = () => {
  return (
    <div>
      <InjectTimestamp render={time => (
        <div>{ time.toLocaleString() }</div>
      )} />
    </div>
  )
}
```

Disadvantages

- Function signature is difficult to enforce (static checking works, using PropTypes at runtime is impossible)
- For the function-as-child pattern, implicitly redefining the meaning of the 3rd argument (*children*) in *createElement* can be confusing

4. "Renderless" State Provider

The "renderless" component pattern takes the container-presenter pattern to the extreme and forces state management to be implemented completely separately from render logic. React Powerplug is one library that provides utilities for composing components with this pattern.

What is inheritance in react JS?

Inheritance **allows the app to do the coupling between the parent-child component and reuse properties such as state values and function in its child components**. React does not use inheritance except in the initial component class, which extends from the react package

Two classes exist are:

- Superclass(Parent Class)
- Subclass(Child Class)

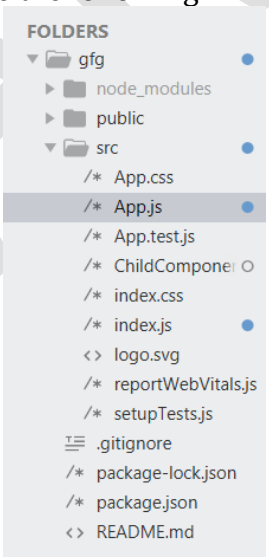
In React, the composition model is used instead of inheritance, so that code can be re-used again between the components. In react **extends** keyword is used on the main function i.e the constructor function.

By using the extends keyword you can have the present component have all the component properties from the already existing component. The composition model uses the super-sub class relationship by passing the state and props. The sub-class segment can access any progressions to one another.

Creating React Application:

- **Step 1:** Create a React application using the following command in the terminal/ command prompt:
  create-react-app foldername
- **Step 2:** After creating your project folder i.e. foldername, move to it using the following command:
  cd foldername

Project Structure: It will look like the following:



Project Structure

Here, you have two components i.e. *AppComponent* and a *ChildComponent*, and the child component takes over all the app properties.

Example: Now write down the following code in the App.js file. Here, App is our default(parent) component where we have written our code. In the below code, this.state.message is passed to ChildComponent.

FileName :App.js

```
import logo from './logo.svg';
import React from 'react';
import './App.css';
import ChildComponent from "./ChildComponent";
class App extends React.Component {
        constructor(props) {
                super(props);
                this.state = {
                        message: " Geeks for Geeks message"
                };
        }
        render() {
                return (
                        <div>
                                <ChildComponent message={this.state.message} />
                        </div>
                );
        }}
export default App;
```

Now write down the following code in the ChildComponent.js file. The child component accepts all the app component properties.

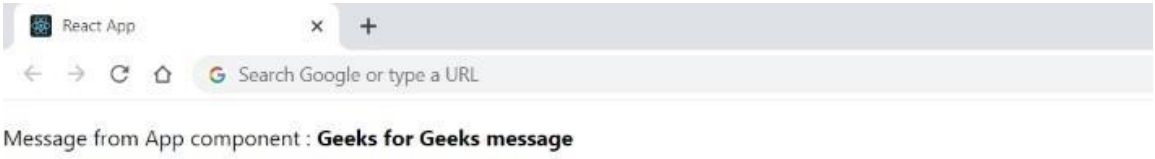
FileName:ChildComponent.js

```
import React from "react";
class ChildComponent extends React.Component {
        render() {
                const { message } = this.props;
                return (
                        <div>
                        <p> Message from App component : <b>{message}</b> </p>
                        </div>
                );
        }}
export default ChildComponent;
```

Step to Run Application: Run the application using the following command from the root directory of the project:

npm start

OUTPUT:



UNIT V

APP IMPLEMENTATION IN CLOUD

Cloud providers Overview

Cloud computing is Web-based computing that allows businesses and individuals to consume computing resources such as virtual machines, databases, processing, memory, services, storage, messaging, events, and pay-as-you-go. Cloud services often improve upon older ones.

Advantages of cloud computing

1. Cloud computing allows a business to cut their operational and fixed monthly costs of hardware, databases, servers, software licenses.
2. Cloud computing offers 24/7 uptime (99.99% uptime). Cloud servers and data centers are managed by the cloud service provided.
3. Cloud computing is scalable and reliable. There is no limit to the number of users or resources.
4. Cloud computing provides maintainability and automatic updates of new software, OS, databases, and third-party software.
5. Cloud service providers have data centers in various locations, which makes them faster and more reliable. Larger companies such as Microsoft and AWS even have data centers around the world.

Cloud computing can be divided into three major categories, Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS).

- Infrastructure: The foundation of every computing environment. This infrastructure could include networks, database services, data management, data storage (known in this context as cloud storage), servers (cloud is the basis for serverless computing), and virtualization.
- Platforms: The tools needed to create and deploy applications. These platforms could include operating systems like Linux®, middleware, and runtime environments.
- Software: Ready-to-use applications. This software could be custom or standard applications provided by independent service providers.

The two leaders in cloud computing are Amazon and Microsoft, followed by Google, Alibaba, and IBM.

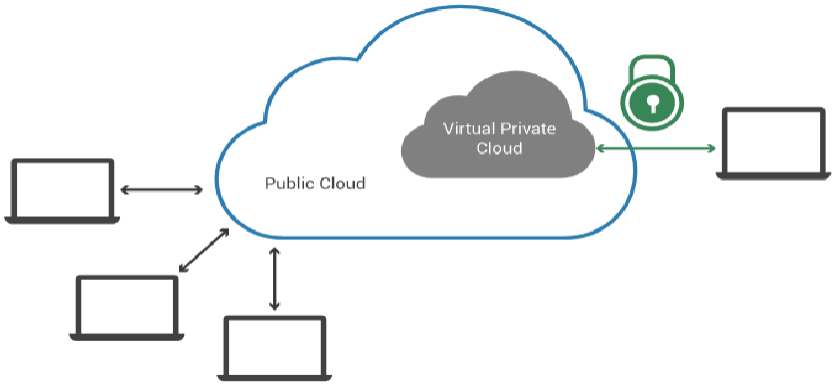
Here is a list of my top 10 cloud service providers:

1. Amazon Web Services (AWS)
2. Microsoft Azure
3. Google Cloud
4. Alibaba Cloud
5. IBM Cloud
6. Oracle
7. Salesforce
8. SAP
9. Rackspace Cloud
10. VMWare

The following table summarizes the top 3 key players and their offerings in the cloud computing world:

	AWS	**Azure**	**Google Cloud**
Company	AWS Inc.	Microsoft	Google
Launch year	2006	2010	2008
Geographical Regions	25	54	21
Availability Zones	78	140 (countries)	61
Key offerings	Compute, storage, database, analytics, networking, machine learning, and AI, mobile, developer tools, IoT, security, enterprise applications, blockchain.	Compute, storage, mobile, data management, messaging, media services, CDN, machine learning and AI, developer tools, security, blockchain, functions, IoT.	Compute, storage, databases, networking, big data, cloud AI, management tools, Identity and security, IoT, API platform
Compliance Certificates	46	90	
Annual Revenue	$33 billion	$35 billion	$8 billion

Virtual Private Cloud



A virtual private cloud (VPC) is a secure, isolated private cloud hosted within a public cloud. VPC customers can run code, store data, host websites, and do anything else they could do in an ordinary private cloud, but the private cloud is hosted remotely by a public cloud provider.

VPCs combine the scalability and convenience of public cloud computing with the data isolation of private cloud computing.

What is a public cloud? What is a private cloud?

A public cloud is shared cloud infrastructure. Multiple customers of the cloud vendor access that same infrastructure, although their data is not shared – just like every person in a restaurant orders from the same kitchen, but they get different dishes. Public cloud service providers include AWS, Google Cloud Platform, and Microsoft Azure, among others.

A private cloud, however, is single-tenant. A private cloud is a cloud service that is exclusively offered to one organization. A virtual private cloud (VPC) is a private cloud within a public cloud; no one else shares the VPC with the VPC customer.

How is a VPC isolated within a public cloud?

A VPC isolates computing resources from the other computing resources available in the public cloud.

Key technologies for isolating a VPC from the rest of the public cloud:

Subnets: A subnet is a range of IP addresses within a network that are reserved so that they're not available to everyone within the network, essentially dividing part of the network for private use.

VLAN: A VLAN is a virtual LAN. Like a subnet, a VLAN is a way of partitioning a network, but the partitioning takes place at a different layer within the OSI model.

VPN: A virtual private network (VPN) uses encryption to create a private network over the top of a public network. VPN traffic passes through publicly shared Internet infrastructure – routers, switches, etc. – but the traffic is scrambled and not visible to anyone.

What are the advantages of using a VPC instead of a private cloud?

Scalability: VPC is hosted by a public cloud provider, customers can add more computing resources on demand.

Easy hybrid cloud deployment: It's relatively simple to connect a VPC to a public cloud or to on-premises infrastructure via the VPN.

Better performance: Cloud-hosted websites and applications typically perform better than those hosted on local on-premises servers.

Better security: The public cloud providers that offer VPCs often have more resources for updating and maintaining the infrastructure, especially for small and mid-market businesses.

Features

- **Availability:** Redundant resources and highly fault-tolerant availability zone architectures mean your applications and workloads are highly available.
- **Security:** VPC is a logically isolated network, your data and applications won't share space or mix with those of the cloud provider's other customers.
- **Affordability**: VPC customers can take advantage of the public cloud's cost-effectiveness, such as saving on hardware costs, labor times, and other resources.

Benefits

- **Flexible business growth:** Because cloud infrastructure resources—including virtual servers, storage, and networking—can be deployed dynamically, VPC customers can easily adapt to changes in business needs.
- **Satisfied customers:** The high availability of VPC environments enables reliable online experiences that build customer loyalty and increase trust in your brand.

Three-tier architecture in a VPC

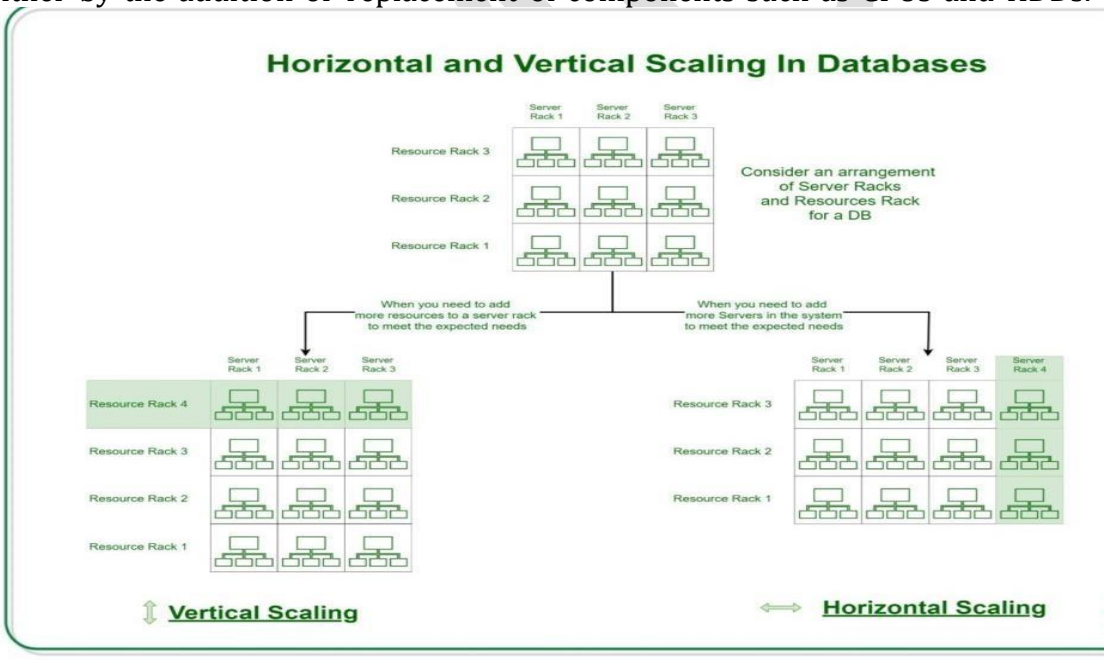
The majority of today's applications are designed with a three-tier architecture comprised of the following interconnected tiers:

- The **web or presentation tier**, which takes requests from web browsers and presents information created by, or stored within, the other layers to end users.
- The **application tier**, which houses the business logic and is where most processing takes place.
- The **database tier**, comprised of database servers that store the data processed in the application tier.

Scaling (Horizontal and Vertical)

Vertical Scaling

- **Vertical scaling** refers to adding more resources (CPU/RAM/DISK) to your server (database or application server is still remains one) as on demand.
- Vertical Scaling is most commonly used in applications and products of middle-range as well as small and middle-sized companies. One of the most common examples of **Virtual Scaling** is to buy an expensive hardware and use it as a Virtual Machine hypervisor (VMWare ESX).
- **Vertical Scaling** usually means upgrade of server hardware. Some of the reasons to scale vertically includes increasing IOPS (Input / Ouput Operations), amplifying CPU/RAM capacity, as well as disk capacity.
- However, even after using virtualization, whenever an improved performance is targeted, the risk for downtimes with it is much higher than using horizontal scaling.
- Vertical scaling of cloud resources is the enhancement of memory, processing power, networking, and other technical capabilities of an existing cloud server, either by the addition or replacement of components such as CPUs and HDDs.



- Vertical scaling is also referred to as scaling 'up' (or 'down'). Compared to horizontal scaling (scaling 'out' or 'in'), vertical cloud scaling entails an increase in power and thereby throughput of a single server or other machine.
- When you scale up, your data and applications continue to exist on a single node. However, the load of processing them spreads through more powerful components, mostly to handle increased workloads.

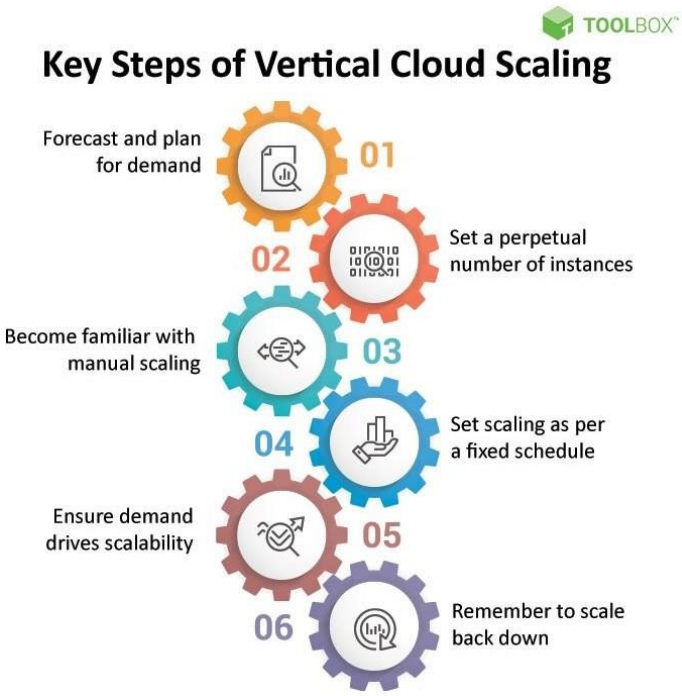
- Leading cloud service providers such as AWS and Azure have many instance sizes available for users. Therefore, vertical scaling of cloud resources is available for several solutions, including RDS databases and EC2 instances.

How does vertical cloud scaling work?

- Cloud scalability is primarily of two types: vertical scaling (upgrading of existing remote infrastructure) and horizontal scaling (deploying new infrastructure).
- In traditional data centers, vertical scaling is normally achieved by purchasing new, powerful servers or components.
- The old servers or components are replaced and either resold, discarded as scrap, or repurposed for less intensive processing- or memory-based applications.
- However, vertical scaling in a cloud environment is much simpler for the end-user. In most cases, all it takes is the modification of instance size on a web portal.
- Ensuring the availability of adequate infrastructure and resources to meet scalability demands is the cloud vendor's responsibility.

Vertical Cloud Scaling Process: 6 Key Steps

The scaling process differs for every leading cloud platform. However, it is generally easy to understand and implement. The following steps are the most relevant in a cloud environment powered by AWS.



Step I: Forecast and plan for demand

This is best achieved through in-depth research to understand past demand and forecast future demand trends.

This strategy should involve stakeholders from across the organization, your cloud vendor(s), and consultants (if necessary). Together, work to identify all your cloud resource needs, as well as the metrics and constraints that are critical for success with vertical scaling.

Step II: Set a perpetual number of instances

Demand levels, related scalability, and flexibility requirements, it is time to set your vertical cloud scaling parameters accordingly. Begin by configuring the auto-scaling feature to preserve a fixed number of instances perpetually.

EC2 auto scaling scans the health of running instances routinely and replaces bad instances if it detects any. This way, you will have the number of instances you need to scale up or down whenever required.

AWS will automatically manage the creation and termination of instances to ensure that stable capacity is maintained according to the fixed value specified. This enables the maintenance of a minimum and maximum capacity for automatic vertical scaling.

Step III: Become familiar with manual scaling

However, it is helpful to keep this information handy if the need to regulate performance without the help of scaling rules arises.

Manual vertical scaling is straightforward in the case of RDS or EC2 servers; simply change the size of the instance.

Be warned, however, that this is just a contingency step with some drawbacks, including the need for manual intervention, potentially higher costs due to architecture being left larger than required, and downtime during the changing of instance size.

Step IV: Set scaling as per a fixed schedule

It's time to put this knowledge to use. As long as your demand forecast from step 1 is accurate, you should be able to scale instances automatically at fixed dates and times whenever required.

For instance, during the launch of a new product or a seasonal shopping event. This will enable all stakeholders to predict the availability of resources at any given moment in advance and even increase the number of perpetual instances if required.

Step V: Ensure demand is driving scalability

It is wise to check back often to ensure that your cloud platform scales as per demand, even if your demand forecast is not perfect.

The demand-based scaling capabilities of leading cloud vendors are comfortably responsive to fluctuations in traffic.

If everything is set up correctly, traffic spikes that you haven't scheduled for according to step 4 should still be addressed.

After all, one of the key benefits of automatic vertical scaling is its ability to address demand that cannot be predicted.

Step VI: Remember to scale back down

It is critical to keep this final step in mind, not only in manual intervention but also otherwise. Familiarize yourself with the pricing structure of your cloud provider, especially the bits specific to vertical scaling (or scaling in general).

Generally speaking, scaling beyond a limit will cost you, even if it is fully automatic. Keep an eye out for opportunities to scale back down whenever feasible.

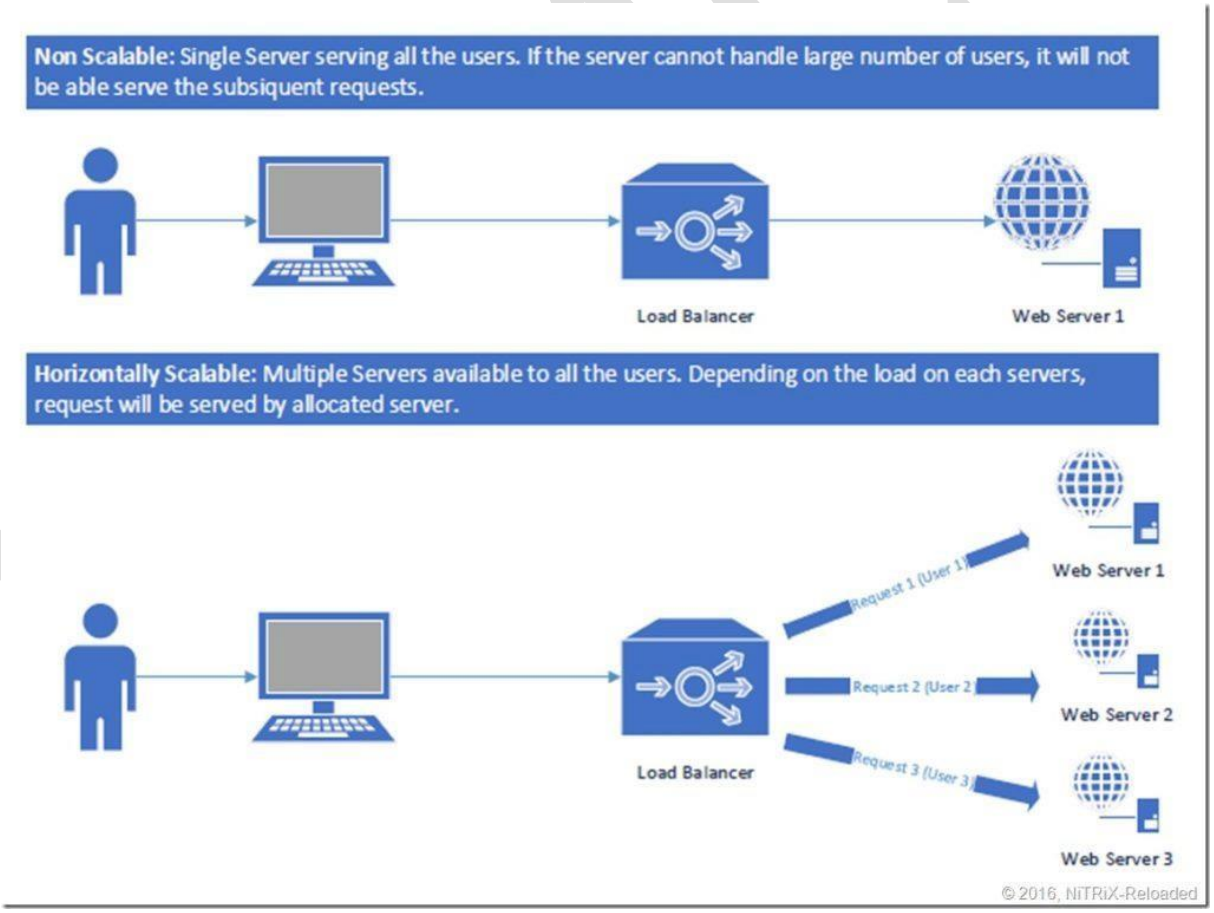
Horizontal Scaling

- ✓ **Horizontal Scaling** is a must use technology – whenever a high availability of (server) services are required **Scaling horizontally** involves adding more processing units or phyiscal machines to your server or database.

- ✓ It involves growing the number of nodes in the cluster, reducing the responsibilities of each member node by spreading the key space wider and providing additional end-points for client connections.
- ✓ Horizontal Scaling has been historically much more used for high level of computing and for application and services.
- ✓ Although this does not alter the capacity of each individual node, the load is decreased due to the distribution between separate server nodes.

The Internet and particular **web services** have boosted the use of **Horizontal Scaling**. Most giant companies that provide well known web services like Google (Gmail, YouTube), Yahoo, Facebook, EBay, Amazon etc. are using heavily horizontal scaling.

Horizontal cloud scaling, also known as scaling out, is the enhancement of cloud bandwidth by adding new computing nodes or machines. In simple terms, horizontal cloud scaling means adding a new server to a data center to help the existing servers handle the increased workload.



Horizontal cloud scaling is normally used for websites, applications, and other internet platforms hosted on the cloud that see wide fluctuations in network traffic.

Through horizontal scaling, the servers that host and process the data for these services are increased or decreased as needed.

This enables the service to adapt to fluctuations in demand, prevent downtime, and ensure high-quality service delivery.

Disadvantages of horizontal scaling

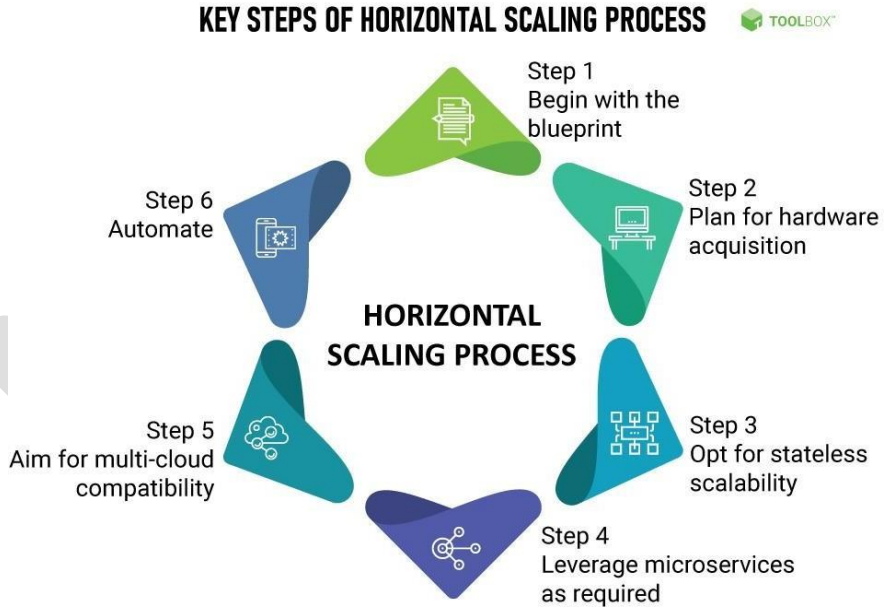
- ✓ **Increase in the complexity of operations** and maintenance.
- ✓ When a data center has multiple servers, it becomes **harder to manage and maintain**.
- ✓ **Things become more complicated on the software** front as well because the need to add load balancing and virtualization may arise.
- ✓ **Backup also becomes difficult** as the synchronization and communication among numerous machines need to be addressed.

Horizontal Scaling Process: 6 Key Steps

Step 1: Begin with the blueprint

Start by including all relevant stakeholders in the study of past demand trends for your organization.

Next, brainstorm with them to determine the short-, medium-, and long-term bandwidth requirements your organization would need to fulfill in the foreseeable future.



Step 2: Plan for hardware acquisition

At this stage of the process, it is important to include representatives from the IT department, as they will be responsible for the ongoing upkeep of this hardware.

Work with all concerned parties to identify how you'll go about scaling out. Aspects to be considered here should include hardware cost, compatibility, and the amount and type of infrastructure required.

Step 3: Opt for stateless scalability

Businesses generally have multiple instances of their applications running on the cloud at the same time. As you scale out, it becomes more difficult to guarantee that client requests will be received by one instance every time.

This makes stateful horizontal scaling (scaling that involves the storage of data between sessions for later use) unsuitable for most business applications.

Step 4: Leverage microservices as required

After setting up the hardware and preparing it for horizontal scaling, the next step is to optimize the process. This can be achieved with the help of microservices.

Microservices allow the splitting of services among various business functions.

Step 5: Aim for multi-cloud compatibility

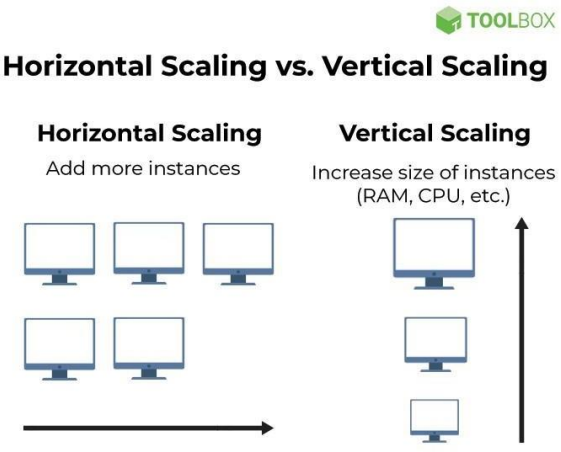
It would make sense to host all your business applications in your own data centers for now. However, there might come a time when you need to expand beyond the workload that your in-house servers can handle, such as a big sale or any other popular event.

That's when ensuring multi-cloud compatibility can prove to be immensely useful. By ensuring that your applications are capable of running on any cloud platform, you can scale out into the infrastructure of a third-party cloud vendor as required.

Step 6: Finally, automate!

Automation makes it easy and cost-effective to create and replicate workloads. It also helps ensure that your servers scale out whenever there is a spike in demand, without the need for manual intervention.

The Difference



Difference between Horizontal and Vertical Scaling

Horizontal Scaling	Vertical Scaling
Horizontal Scaling is defined as the ability to extend capacity by interfacing different hardware or software entities	Vertical Scaling is defined as the ability to increase an existing system's capacity by adding resources
It is based on partitioning where each node contains a single part of data	The data is present on a single node and is scaled through multicore
It is referred to as Scale-out	It is referred to as Scale-up
The licensing fee is costly	The licensing is cost-effective
It requires a load balancer to distribute load among the servers within a system	Scaling the server capacity enhances the load capacity of the server
It implies boosting the power of individual server with the existing server	It implies boosting the power of the individual server

Virtual Machines, Ethernet and Switches

What is a virtual machine (VM)?

- A virtual machine is a virtual representation, or emulation, of a physical computer. They are often referred to as a guest while the physical machine they run on is referred to as the host.
- Virtualization makes it possible to create multiple virtual machines, each with their own operating system (OS) and applications, on a single physical machine. A VM cannot interact directly with a physical computer.
- Instead, it needs a lightweight software layer called a hypervisor to coordinate between it and the underlying physical hardware. The hypervisor allocates physical computing resources—such as processors, memory, and storage—to each VM. It keeps each VM separate from others so they don't interfere with each other.
- While this technology can go by many names, including virtual server, virtual server instance (VSI) and virtual private server (VPS), this article will simply refer to them as virtual machines.

How virtualization works

When a hypervisor is used on a physical computer or server, (also known as bare metal server), it allows the physical computer to separate its operating system and applications from its hardware. Then, it can divide itself into several independent "virtual machines."

Each of these new virtual machines can then run their own operating systems and applications independently while still sharing the original resources from the bare metal server, which the hypervisor manages. Those resources include memory, RAM, storage, etc.

There are two primary types of hypervisors.

Type 1 hypervisors run directly on the physical hardware (usually a server), taking the place of the OS. Typically, you use a separate software product to create and manipulate VMs on the hypervisor.

Type 2 hypervisors run as an application within a host OS and usually target single-user desktop or notebook platforms. With a Type 2 hypervisor, you manually create a VM and then install a guest OS in it. You can use the hypervisor to allocate physical resources to your VM, manually setting the amount of processor cores and memory it can use. Depending on the hypervisor's capabilities, you can also set options like 3D acceleration for graphics.

Advantages and benefits of VMs

- **Resource utilization and improved ROI:** Because multiple VMs run on a single physical computer, customers don't have to buy a new server every time they want to run another OS, and they can get more return from each piece of hardware they already own.
- **Scale:** With cloud computing, it's easy to deploy multiple copies of the same virtual machine to better serve increases in load.
- **Portability:** VMs can be relocated as needed among the physical computers in a network. This makes it possible to allocate workloads to servers that have spare computing power.
- **Flexibility**: Creating a VM is faster and easier than installing an OS on a physical server. Developers and software testers can create new environments on demand to handle new tasks as they arise.
- **Security:** A VM is a file that can be scanned for malicious software by an external program. The fast, easy creation of VMs also makes it possible to completely delete a compromised VM and then recreate it quickly, hastening recovery from malware infections.

Types of VMs

- Windows virtual machines
- Android virtual machines
- Mac virtual machines
- iOS virtual machines
- Java virtual machines
- Python virtual machines
- Linux virtual machines
- VMware virtual machines
- Ubuntu virtual machines

Windows virtual machines

- Most hypervisors support VMs running the Windows OS as a guest. Microsoft's Hyper-V hypervisor comes as part of the Windows operating system.
- When installed, it creates a parent partition containing both itself and the primary Windows OS, each of which gets privileged access to the hardware.
- Other operating systems, including Windows guests, run in child partitions that communicate with the hardware via the parent partition.

Android virtual machines

- Google's open-source Android OS is common on mobile devices and connected home devices such as home entertainment devices.
- The Android OS runs only on the ARM processor architecture that is common to these devices, but enthusiasts, Android gamers, or software developers might want to run it on PCs.

Mac virtual machines

- Apple only allows its macOS system to run on Apple hardware, prohibiting people from running it on non-Apple hardware as a VM or otherwise under its end user license agreement.

iOS virtual machines

- It is not possible to run iOS in a VM today because Apple strictly controls its iOS OS and doesn't allow it to run on anything other than iOS devices.
- The closest thing to an iOS VM is the iPhone simulator that ships with the Xcode integrated development environment, which simulates the entire iPhone system in software.

Java virtual machines

- The Java platform is an execution environment for programs written in the Java software development language.
- Java's promise was "write once, run anywhere" functionality.
- This meant that any Java program could run on any hardware running the Java platform.
- To achieve that, the Java platform includes a Java virtual machine (JVM).

Python virtual machines

- Like the JVM, the Python VM doesn't run on a hypervisor, and it doesn't contain a guest OS.
- It is a tool that enables programs written in the Python programming language to run on a variety of CPUs.
- Similar to Java, Python translates its programs into an intermediate format called bytecode, storing it in a file ready for execution.
- When the program runs, the Python VM translates the bytecode into machine code for fast execution.

Linux virtual machines

- It is also a common host OS used to run VMs and even has its own hypervisor called the kernel-based virtual machine (KVM).

VMware virtual machines

- VMware was an early virtualization software vendor and is now a popular provider of both Type 1 and Type 2 hypervisor and VM software to enterprise customers.

Ubuntu virtual machines

- Ubuntu can be deployed as a guest OS on Microsoft Hyper-V.
- It provides an optimized version of Ubuntu Desktop that works well in Hyper-V's Enhanced Session Mode, providing tight integration between the Windows host and Ubuntu VM.
- It includes support for clipboard integration, dynamic desktop resizing, shared folders, and moving the mouse between the host and guest desktops.

Docker Container

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly.

The Docker platform

- Docker provides the ability to package and run an application in a loosely isolated environment called a container.
- The isolation and security allows you to run many containers simultaneously on a given host.
- Containers are lightweight and contain everything needed to run the application, so you do not need to rely on what is currently installed on the host.
- You can easily share containers while you work, and be sure that everyone you share with gets the same container that works in the same way.

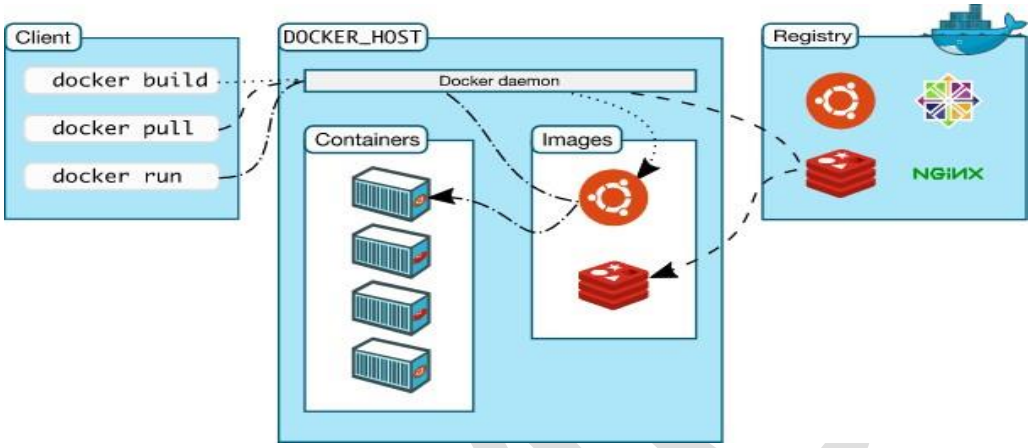
Docker provides tooling and a platform to manage the lifecycle of your containers:

- Develop your application and its supporting components using containers.
- The container becomes the unit for distributing and testing your application.
- When you're ready, deploy your application into your production environment, as a container or an orchestrated service. This works the same whether your production environment is a local data center, a cloud provider, or a hybrid of the two.

Docker architecture

- ✓ Docker uses a client-server architecture.
- ✓ The Docker *client* talks to the Docker *daemon*, which does the heavy lifting of building, running, and distributing your Docker containers.

- ✓ The Docker client and daemon *can* run on the same system, or you can connect a Docker client to a remote Docker daemon.
- ✓ The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.
- ✓ Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers.



The Docker daemon

The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes.

A daemon can also communicate with other daemons to manage Docker services.

The Docker client

The Docker client (docker) is the primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to dockerd, which carries them out.

The docker command uses the Docker API. The Docker client can communicate with more than one daemon.

Docker Desktop

Docker Desktop is an easy-to-install application for your Mac or Windows environment that enables you to build and share containerized applications and microservices.

Docker Desktop includes the Docker daemon (dockerd), the Docker client (docker), Docker Compose, Docker Content Trust, Kubernetes, and Credential Helper.

Docker registries

A Docker *registry* stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default.

When you use the docker pull or docker run commands, the required images are pulled from your configured registry.

When you use the docker push command, your image is pushed to your configured registry.

Docker objects

When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects.

Images

An *image* is a read-only template with instructions for creating a Docker container. Often, an image is *based on* another image, with some additional customization.

Example: you may build an image which is based on the ubuntu image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.

You might create your own images or you might only use those created by others and published in a registry. To build your own image, you create a *Dockerfile* with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Dockerfile creates a layer in the image.

When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.

Containers

A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.

You can control how isolated a container's network, storage, or other underlying subsystems are from other containers or from the host machine.

When a container is removed, any changes to its state that are not stored in persistent storage disappear.

Example docker run command

The following command runs an ubuntu container, attaches interactively to your local command-line session, and runs /bin/bash.

$ docker run **-i -t** ubuntu /bin/bash

Kubernetes

Kubernetes is an open source container orchestration platform that automates many of the manual processes involved in deploying, managing, and scaling containerized applications.

Modern software is increasingly run as fleets of containers, sometimes called microservices.

A complete application may comprise many containers, all needing to work together in specific ways.

Kubernetes is software that turns a collection of physical or virtual hosts (servers) into a platform that:

- Hosts containerized workloads, providing them with compute, storage, and network resources, and

- Automatically manages large numbers of containerized applications — keeping them healthy and available by adapting to changes and challenges

Kubernetes work

1. When developers create a multi-container application, they plan out how all the parts fit and work together, how many of each component should run, and roughly what should happen when challenges (e.g., lots of users logging in at once) are encountered.

2. They store their containerized application components in a container registry (local or remote) and capture this thinking in one or several text files comprising a*configuration*. To start the application, they "apply" the configuration to Kubernetes.

3. Kubernetes job is to evaluate and implement this configuration and maintain it until told otherwise. It:

    - Analyzes the configuration, aligning its requirements with those of all the other application configurations running on the system

    - Finds resources appropriate for running the new containers (e.g., some containers might need resources like GPUs that aren't present on every host)

    - Grabs container images from the registry, starts up the new containers, and helps them connect to one another and to system resources (e.g., persistent storage), so the application works as a whole

4. Then Kubernetes monitors everything, and when real events diverge from desired states, Kubernetes tries to fix things and adapt.

Use of Kubernetes

One of the benefits of Kubernetes is that it makes building and running complex applications much simpler. Here's a handful of the many Kubernetes features:

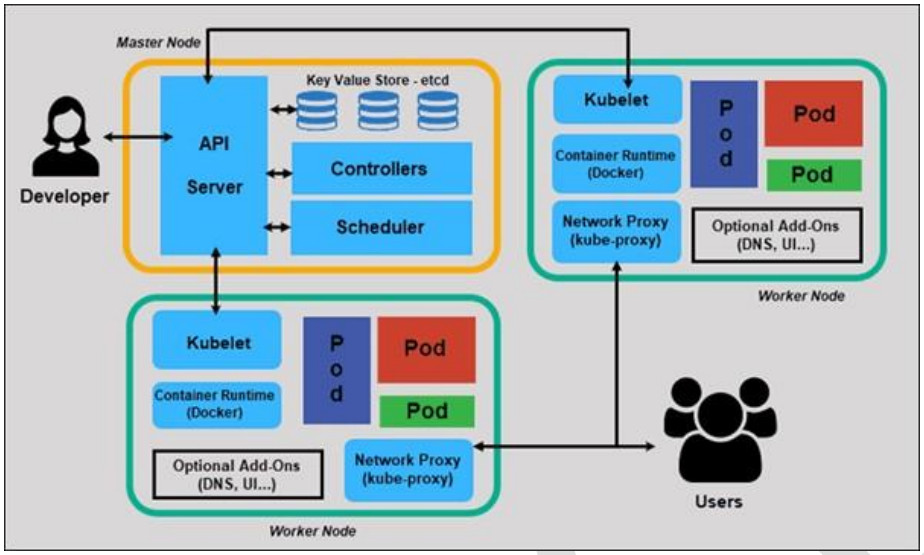
- Standard services like local DNS and basic load-balancing that most applications need, and are easy to use.

- Standard behaviors (e.g., restart this container if it dies) that are easy to invoke, and do most of the work of keeping applications running, available, and performant.

- A standard set of abstract "objects" (called things like "pods," "replicasets," and "deployments") that wrap around containers and make it easy to build configurations around collections of containers.

- A standard API that applications can call to easily enable more sophisticated behaviors, making it much easier to create applications that manage other applications.

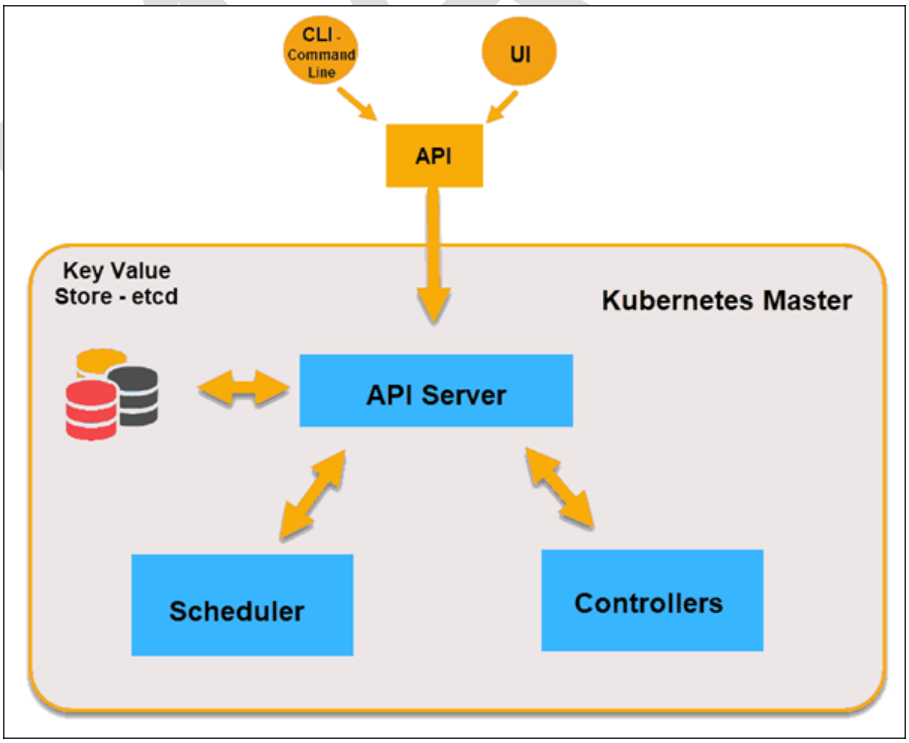
components :



The master node controls and manages a set of worker nodes and contains the Kubernetes cluster. We can talk to the master node via CLI, GUI, or API, and more than one master node can be used for fault tolerance. Kubernetes uses the etcd, and all master nodes are connected to etcd, which is a distributed key-value store.

API Server

API servers perform all administrative tasks on the master nodes. Users send the command to the API server, which then validates the request process and executes them. The API server determines if the request is valid or not and then processes it.



Key-Value Store (etcd)

Etcd is an open-source distributed Key-Value Store used to hold and manage the critical information that distributed systems need to keep running. The Key-Value Store, also called etcd, is a database Kubernetes uses to back up all cluster data. It stores the entire configuration and state of the cluster.

Controller

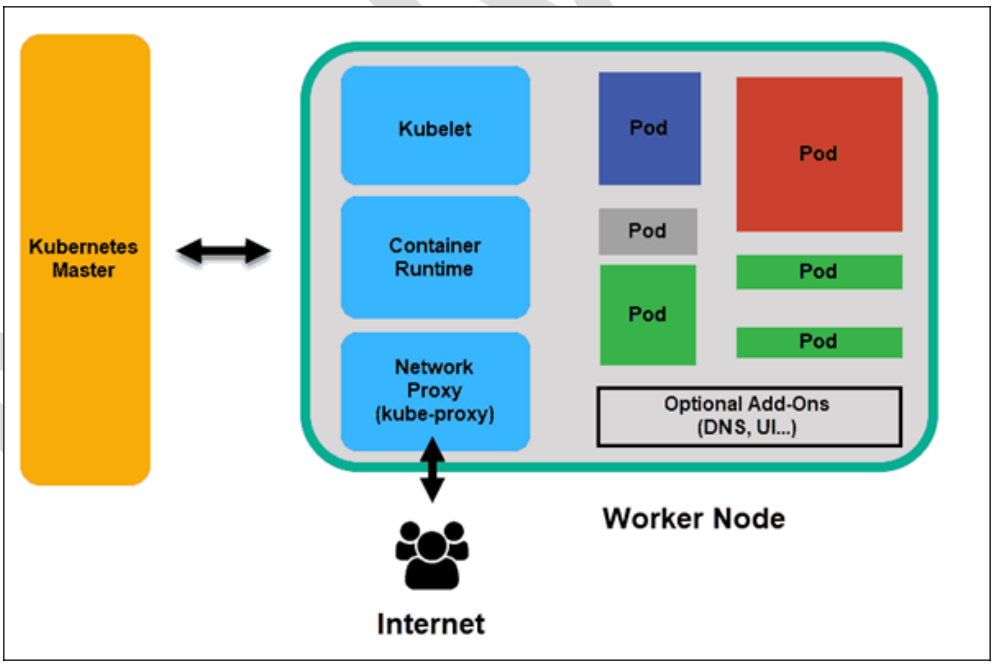
The role of the Controller is to obtain the desired state from the API Server. It checks the current state of the nodes it is tasked to control, and determines if there are any differences, and resolves them if any.

Scheduler

The Scheduler's main job is to watch for new requests coming from the API Server and assign them to healthy nodes. It ranks the quality of the nodes and deploys pods to the best-suited node.

Worker Node

Worker nodes listen to the API Server for new work assignments; they execute the work assignments and then report the results to the Kubernetes Master node.



Kubelet

The kubelet runs on every node in the cluster. It is the principal Kubernetes agent. By installing kubelet, the node's CPU, RAM, and storage become part of the broader cluster. It watches for tasks sent from the API Server, executes the task, and reports back to the Master.

Container Runtime

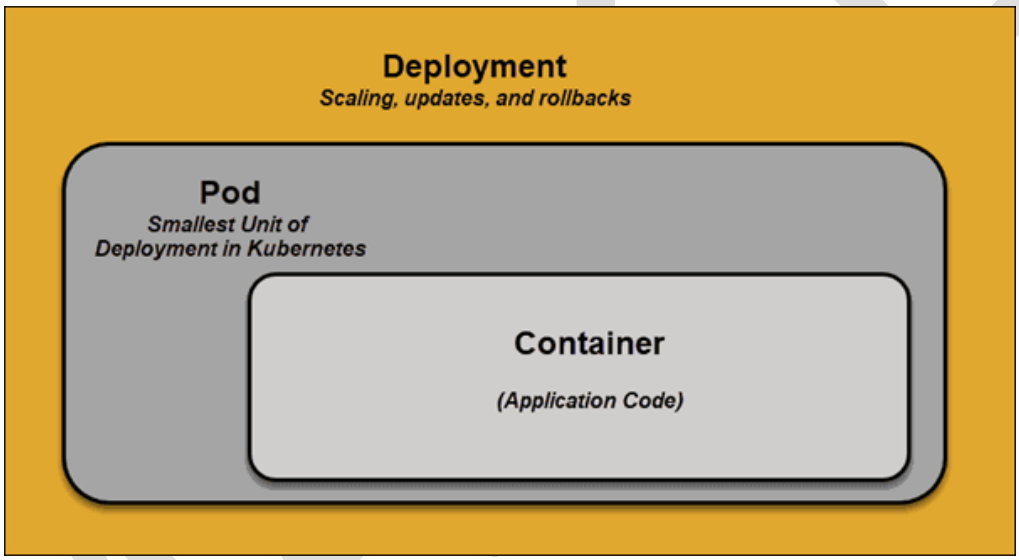
The container runtime pulls images from a container image registry and starts and stops containers. A 3rd party software or plugin, such as Docker, usually performs this function.

Kube-proxy

The kube-proxy makes sure that each node gets its IP address.

Pod

A pod is the smallest element of scheduling in Kubernetes. Without it, a container cannot be part of a cluster.



Kubernetes Services

One of the best features Kubernetes offers is that non-functioning pods get replaced by new ones automatically. The new pods have a different set of IPs. It can lead to processing issues and IP churn as the IPs no longer match. If left unattended, this property would make pods highly unreliable.

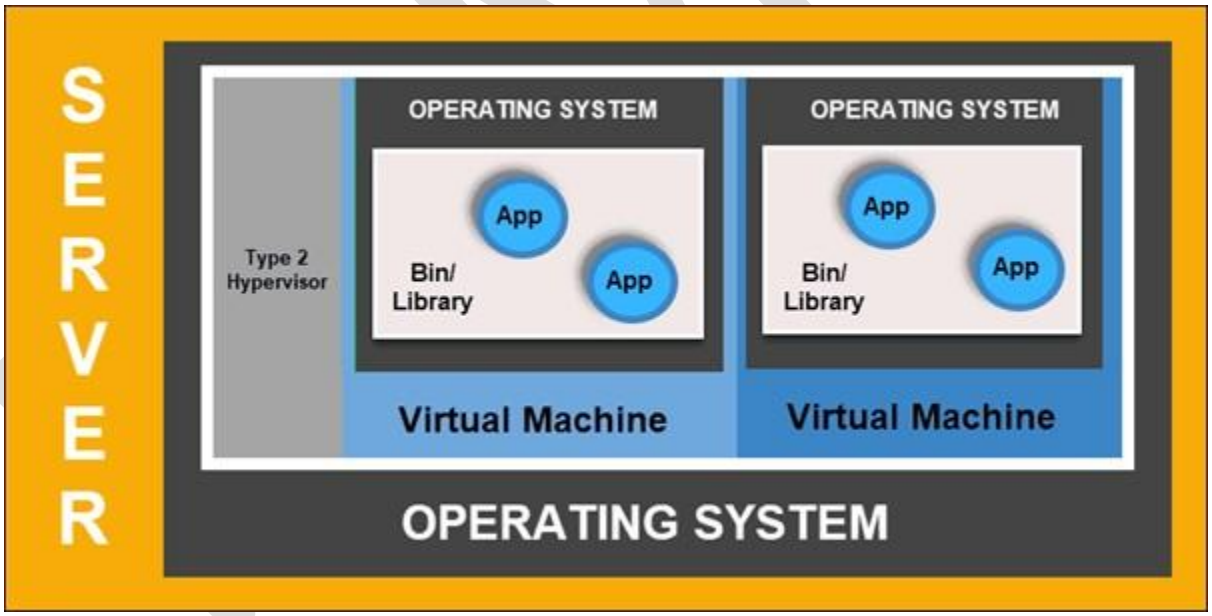
1. **Traditional Deployment**

Initially, developers deployed applications on individual physical servers. This type of deployment posed several challenges. The sharing of physical resources meant that one application could take up most of the processing power, limiting the performance of other applications on the same machine.

It takes a long time to expand hardware capacity, which in turn increases costs. To resolve hardware limitations, organizations began virtualizing physical machines.

2. Virtualized Deployment

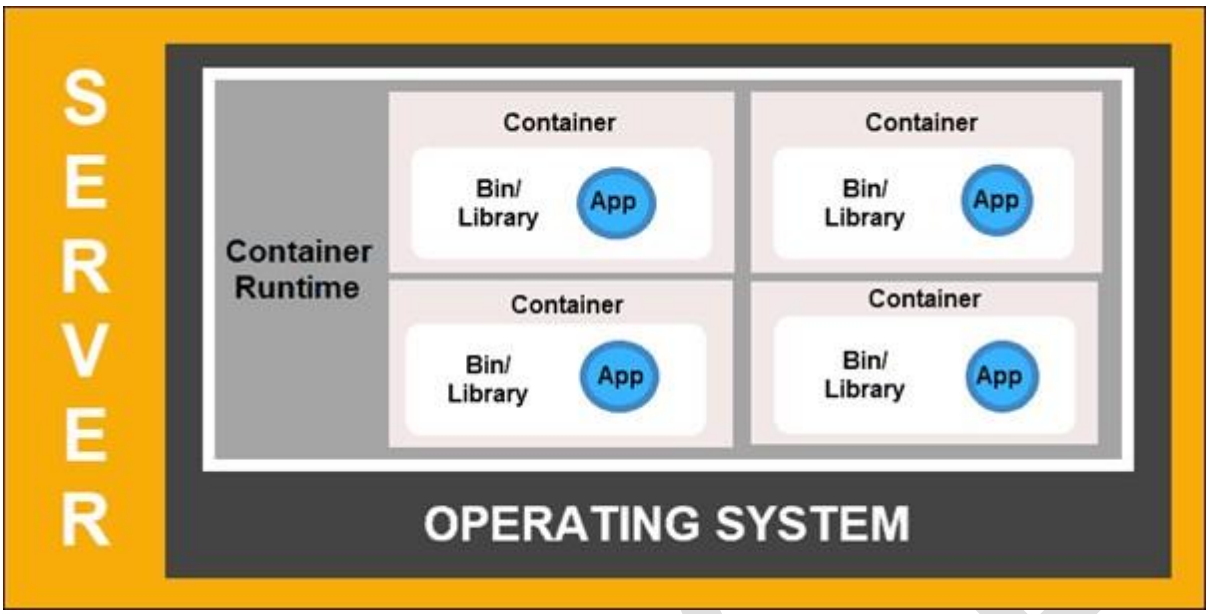
Virtualized deployments allow you to scale quickly and spread the resources of a single physical server, update at will, and keep hardware costs in check. Each VM has its operating system and can run all critical systems on top of the virtualized hardware.



3. Container Deployment

Container Deployment is the next step in the drive to create a more flexible and efficient model. Much like VMs, containers have individual memory, system files, and processing space.

Multiple applications can now share the same underlying operating system. This feature makes containers much more efficient than full-blown VMs. They are portable across clouds, different devices, and almost any OS distribution.



The container structure also allows for applications to run as smaller, independent parts. These parts can then be deployed and managed dynamically on multiple machines. The elaborate structure and the segmentation of tasks are too complex to manage manually.

****

UNIT V APP IMPLEMENTATION IN CLOUD

Cloud providers Overview – Scaling (Horizontal and vertical) – Virtual Machines, Ethernet and Switches – Docker Container – Kubernetes

Cloud Providers Overview:

In Spinnaker, providers are integrations to the Cloud platforms you deploy your applications to.

In this section, you'll register credentials for your Cloud platforms. Those credentials are known as *Accounts* in Spinnaker, and Spinnaker deploys your applications via those accounts.

Supported providers:

All of Spinnaker's abstractions and capabilities are built on top of the Cloud Providers that it supports. So, for Spinnaker to do anything you must enable at least one provider, with one Account added for it.

Add as many of the following providers as you need. When you're done, return to this page.

- App Engine
- Amazon Web Services
- Azure
- Cloud Foundry
- DC/OS
- Google Compute Engine
- Kubernetes
- Oracle

CPI - Cloud Provider Interface:

Why do we need it?

The Kubernetes Controller Manager (KCM) is a daemon that embeds the core control loops shipped with Kubernetes. The Cloud Provider Interface is responsible for running all the platform specific control loops that were previously run in core Kubernetes components like the KCM and the kubelet, but have been moved out-of-tree to allow cloud and infrastructure providers to implement integrations that can be developed, built and released independent of Kubernetes core.

Since Kubernetes is a declarative system, the purpose of these control loops is to watch the actual state of the system through the API server. If the actual state is different from the desired/declared state, it initiates operations to rectify the situation by making changes to try to move the current state towards the desired state.

When an application is deployed in Kubernetes, the application definition (the desired end-state of the application) is persisted in etcd via the API server on the K8s master node.

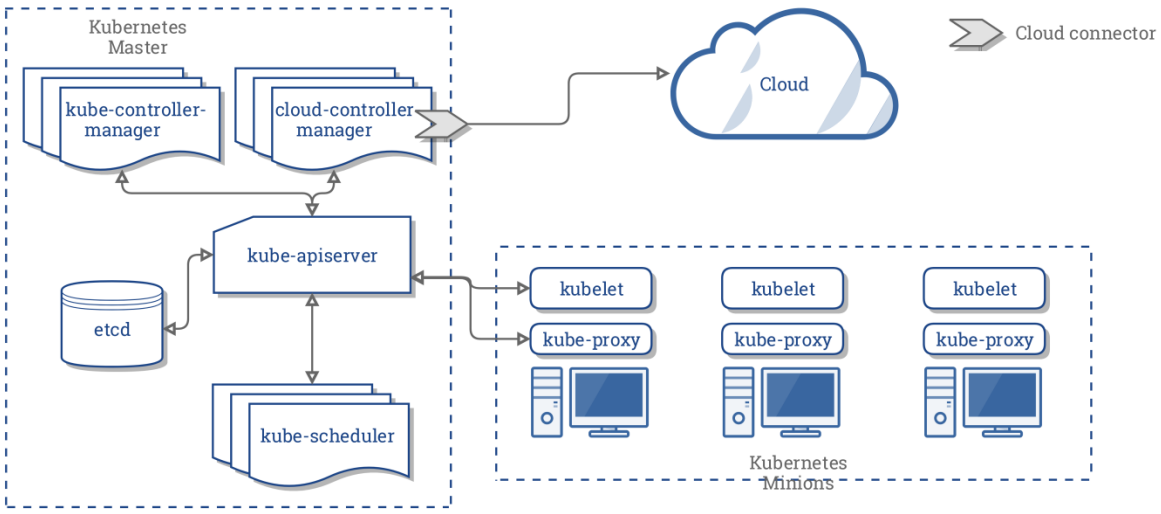
The API server holds both a record of the desired state and another record of the actual state (real world observed state). When these records differ, a controller is responsible for initiating tasks to rectify the difference. This could be something as simple as a request to add a PersistentVolume to a Pod. In this case the desired state is different from the actual state, so the controllers will initiate tasks to make them the same, i.e. whatever is needed to attach the correct PV to the Pod.

The Cloud Provider Interface (CPI) replaces the Kubernetes Controller Manager for only the cloud specific control loops. The Cloud Provider breaks away some of the functionality of Kubernetes controller manager (KCM) and handles it separately. Note that in many cases, some of these interfaces are not relevant for some CPIs, so you may only see a subset of these interfaces implemented for your cloud provider.

Interfaces (optionally) implemented in the Cloud Provider are as follows:

- **Node control loops**, provide cloud specific info about nodes in your cluster. It does a number of tasks:
  - Initialize a node with cloud specific zone/region labels
  - Initialize a node with cloud specific instance details, for example, type and size
  - Obtain the node's IP addresses and hostname
  - In case a node becomes unresponsive, check the cloud to see if the node has been deleted from the cloud. If the node has been deleted from the cloud, delete the Kubernetes Node object.
- **Route control loops**, provide cloud specific info about networking. It is responsible for configuring network routes in the infrastructure so that containers on different nodes in the Kubernetes cluster can communicate with each other. At the time of writing, the route controller is only applicable for Google Compute Engine clusters, and is not applicable to vSphere.
- **Service control loops** - responsible for listening to K8s Service type create, update, and delete events. This is also known as a Load Balancer control loop, a cloud specific ingress controller. Based on the current state of the services in Kubernetes, it configures load balancers (such as Amazon ELB , Google LB, or Oracle Cloud Infrastructure LB) to reflect the state of the services in Kubernetes. Additionally, it ensures that service backends for load balancers are up to date. vSphere does not have a native load balancer per-se. However, VMware's network virtualization product, NSX, can be used to provide such functionality to K8s.

- There is no volume controller now. This responsibility has been taken over by the CSI, the Container Storage Initiative. So for vSphere, when transitioning from in-tree to out-of tree, you need both a CPI and CSI to get the same level of functionalist as the previous vSphere Cloud Provider (VCP), e.g. zones for placement.
- **Custom control loops** - implementations of the CPI can also run custom controllers that enhance the cluster's capabilities specific to the underlying infrastructure platform. vSphere today does not run any custom controllers but may introduce them in the future.



The vSphere Cloud Provider does not implement LoadBalancer, Clusters or Routes - these networking features are not available in native vSphere. The vSphere CPI is only implementing node instances in the vSphere CPI. The credentials to connect to vSphere is managed with a Kubernetes secrets file. Zone support is significant because vSphere has its own concept of zones/fault domains. The CPI maps those vSphere Zone concepts to Kubernetes Zone concepts. vSphere tags are used to identify zones and regions in vSphere datacenter objects. These same tags are mapped to labels in Kubernetes, allowing placement of Nodes and thus Pods and Persistent Volumes in the appropriate Zone or Region.

CPI Integration Detailed:

Once the vSphere Cloud Provider is fully functional on your cluster, your cluster will have access to new integration points with vSphere. Below are the key integrations that are enabled by the vSphere cloud provider.

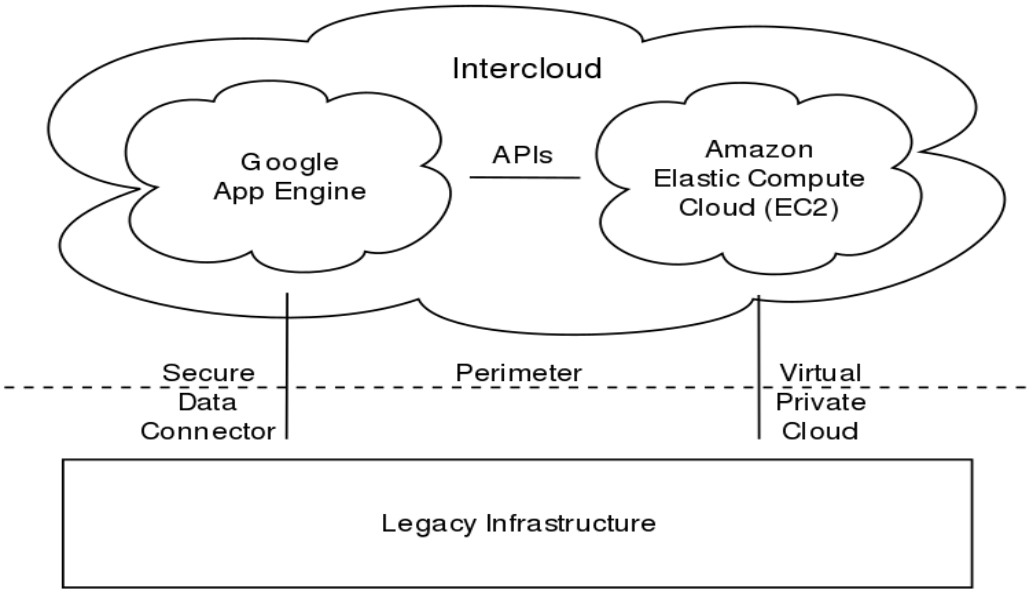
Kubernetes Nodes:

When a Kubernetes node registers itself with the Kubernetes API server, it will request additional information about itself from the cloud provider. As of today, the cloud provider will provide a new Node object in the cluster with it's node addresses, instance type and zone/region topology.

Virtual private cloud:

A **virtual private cloud (VPC)** is an on-demand configurable pool of shared resources allocated within a **public** cloud environment, providing a certain level of isolation between the different organizations (denoted as *users* hereafter) using the resources. The isolation between one VPC user and all other users of the same cloud (other VPC users as well as other public cloud users) is achieved normally through allocation of a private IP subnet and a virtual communication construct (such as a VLAN or a set of encrypted communication channels) per user. In a VPC, the previously described mechanism, providing isolation within the cloud, is accompanied with a virtual private network (VPN) function (again, allocated per VPC user) that secures, by means of authentication and encryption, the remote access of the organization to its VPC resources. With the introduction of the described isolation levels, an organization using this service is in effect working on a '**virtually private**' cloud (that is, as if the cloud infrastructure is not shared with other users), and hence the name VPC.

VPC is most commonly used in the context of cloud infrastructure as a service. In this context, the infrastructure provider, providing the underlying public cloud infrastructure, and the provider realizing the VPC service over this infrastructure, may be different vendors.



Virtual Private Cloud (VPC)

Virtual private cloud (VPC) is a cloud computing standard for securing cloud apps in a public cloud environment.

If you are a cloud computing enthusiast or technology learner, you must have heard of the term Virtual private cloud (VPC.) It refers to the booming resource-sharing cloud computing technology.

Gone are the days when IT enterprises used to depend on the private cloud for data security. The scenario has changed with the emergence of advanced cloud security protocols. Now, many users go for the public cloud for savings and scalability when they need to use cloud resources.

Continue reading this blog to know more about the technology and its functionalities

Defining a Virtual Private Cloud (VPC):

First, let's look at its detailed definition to understand the concept. Virtual private cloud or VPC is a cloud resource-sharing standard. VPC will ensure security and data operations throughput like a private cloud, even if you use any crowded public cloud environment.

For example, imagine the public cloud as a restaurant with hundreds of others dining there. Here, a VPC is equivalent to a private chamber that you can use with your family and have privacy and security within a public premise.

The reason behind its popularity is manifold. Some of the perks of using VPC are minimal IT infrastructure cost, on-demand resource scaling (up and down), and virtually zero downtime due to maintenance.

Moreover, The process for setting up a private cloud-like computing system for various types of websites and applications (web, mobile, or desktop) is super easy. You can also set it up on a public cloud ecosystem managed by a third party.

A VPC is a perfect solution for cloud users who want complete control over their software-as-a-service (SaaS) apps and prefer powerful customization options. The reason is the segregation of their cloud assets from other cloud users. Furthermore, modifications that VPC users make don't interfere with the settings or assets of non-VPC users.

VPC ensures the security of VPC users by allowing access to their digital assets only through some particular IP addresses. The users can control access permission and assign apps or website access permission to a trusted group of public cloud users.

How to Construct a VPC:

A VPC architecture consists of multiple cloud resources. It's more or less similar to a home computing system. The only difference is it exists in the cloud.

Here are the cloud resources of a VPC:

- A **virtual server instance (VSI)** creates the compute resources. It comes with specific processing power and memory.
- The **logical instance** assists in data communication. This instance is known as networking. It allows end-users to access the cloud apps or tools they own.
- A **flexible storage** resource for data storage. You may scale up or scale down your storage quota based on the changing needs.

The VPC networking system uses various complicated and robust protocols to perform different actions. For instance, tool users can access the public gateways using the internet, which is a public-facing network.

In the VPC architecture, there are also elements like load balancers that distribute the incoming traffic to different VSIs. As a result, the network can positively impact the operation and performance of the tool. VPC also uses routers for internal communication between segmented works and bring traffic. It ensures that the external traffic gets to the public-facing apps.

The logical instances or cloud resources of a VPC remain segregated from the remaining public cloud. To keep the logical instances isolated, the cloud computing vendor uses various virtual and physical mechanisms.

The data within a VPC operates through a three-tier architecture where all the tiers need their own subnet. For this reason, each tier gets dedicated IP address ranges.

The VPC networking system uses various complicated and robust protocols to perform different actions. For instance, tool users can access the public gateways using the internet, which is a public-facing network.

In the VPC architecture, there are also elements like load balancers that distribute the incoming traffic to different VSIs. As a result, the network can positively impact the operation and performance of the tool. VPC also uses routers for internal communication between segmented works and bring traffic. It ensures that the external traffic gets to the public-facing apps.

The logical instances or cloud resources of a VPC remain segregated from the remaining public cloud. To keep the logical instances isolated, the cloud computing vendor uses various virtual and physical mechanisms.

The data within a VPC operates through a three-tier architecture where all the tiers need their own subnet. For this reason, each tier gets dedicated IP address ranges.

How to Isolate a VPC:

By generating a virtual layer on public cloud hardware, a VPC keeps partial processing storage, capability, and memory for itself. It's possible to create several VPCs based on the power of public cloud infrastructure. Thus, a virtual private network (VPN) keeps its user data and the data processing methods separated from other public cloud users.

You can ensure the secured and isolated data transmission by these networking components:

#1. Network Address Translation (NAT)

All the cloud-based tools located on the VPC uses separate private IP addresses to transmit data. Thus, the public-facing connection becomes impossible for any VPC-hosted app unless you use NAT to overcome this obstacle.

When someone attempts to access your app, NAT will ensure safe communication between private and public domains. It does so by matching the predefined public IP with the private IP.

#2. Virtual Local Area Network (VLANs)

The function of VLAN is to split the public cloud network into an isolated and reserved network. The data link layer of the OSI model is the place where splitting occurs.

In a VPC environment, Your systems will communicate via a VLAN system. Hence, it'll stay separated from the other components of the public cloud.

#3. Virtual Private Networks (VPNs)

VPN is the most popular of the protocols that develop a private networking channel on a public network.

VPC uses the public cloud network to transmit data between the nodes or cloud assets. But, it utilizes the VPN technology for data encryption while transmitting through shared resources such as routers, switches, etc.

#4. Private Internet Protocols (Subnet IPs)

A VPC also contains private IP addresses which you can use for internal application connectivity for security and performance. Data transmitted through private IPs within VPC won't be traveling through the public Internet.

Conclusion

To sum up, a virtual private cloud (VPC) is a private cloud computing environment located within a public cloud. Besides flexibility, it offers you scalability, reduced risk, and minimum downtime. If you prefer cloud-based service, VPC is an affordable option for you if you want your network infrastructure to expand along with your business growth.

Cloud computing is becoming more secure as time passes and new technologies emerge. Learn more about challenges and risks in cloud computing to keep your cloud assets safe

Horizontal Scaling vs Vertical Scaling & Multi-Tenancy:

In this section we will learn about various Cloud Concepts:- a) Scaling on the cloud b) Degradation of Services c) Availability Vs Durability on the Cloud d) Single & Multi-tenancy Applications e) Types of Cloud Deployments

Discover the differences between horizontal and vertical scaling — both in the cloud and on-premise

What Is Scalability?

Scalability describes a system's elasticity. While we often use it to refer to a system's ability to grow, it is not exclusive to this definition. We can scale down, scale up, and scale out accordingly.

If you are running a website, web service, or application, its success hinges on the amount of network traffic it receives. It is common to underestimate just how much traffic your system will incur, especially in the early stages. This could result in a crashed server and/or a decline in your service quality.

Thus, scalability describes your system's ability to adapt to change and demand. Good scalability protects you from future downtime and ensures the quality of your service.

But what options do you have when it comes to implementing scaling and ensuring your business's scalability? That's where horizontal and vertical scaling come in.

What Is Horizontal Scaling?

Horizontal scaling (aka scaling out) refers to adding additional nodes or machines to your infrastructure to cope with new demands. If you are hosting an application on a

server and find that it no longer has the capacity or capabilities to handle traffic, adding a server may be your solution.

It is quite similar to delegating workload among several employees instead of one. However, the downside of this may be the added complexity of your operation. You must decide which machine does what and how your new machines work with your old machines.

You can consider this the opposite of vertical scaling.

What Is Vertical Scaling?

Vertical scaling (aka scaling up) describes adding additional resources to a system so that it meets demand. How is this different from horizontal scaling?

While horizontal scaling refers to adding additional nodes, vertical scaling describes adding more power to your current machines. For instance, if your server requires more processing power, vertical scaling would mean upgrading the CPUs. You can also vertically scale the memory, storage, or network speed.

Additionally, vertical scaling may also describe replacing a server entirely or moving a server's workload to an upgraded one.

Horizontal Vs. Vertical Scaling

Once again, the biggest central functional difference between the two is that horizontal scaling often forces you to rework how you implement your services or layers. For instance, let's look at simple three-tier architecture.

You have your presentation tier (user interface/client), logic tier (virtual server/services), and data tier (storage/databases). In the case of horizontal scaling, you can delegate each tier (or the functions responsible for them) to a different node.

However, you may already be running these tiers on different servers but find that one of these servers is underperforming or no longer meets demands. Once again, you can choose to scale this server vertically or horizontally. You may upgrade it with more resources or add another server to share the workload.

For further illustration, let's consider databases. If you host your database on a single dedicated server and it gets too large, horizontal scaling would mean adding a new node, partitioning, and sharing the data between the old server and the new.

In our "lifting weights" analogy, horizontal scaling would mean buying new clothes while vertical scaling would be modifying your old clothes to handle your new gains.

With that being said, let's look at a simple breakdown of the advantages and disadvantages of vertical and horizontal scaling.

Advantages of horizontal scaling

- **Scaling is easier from a hardware perspective** - All horizontal scaling requires you to do is add additional machines to your current pool. It eliminates the need to analyze which system specifications you need to upgrade.
- **Fewer periods of downtime** - Because you're adding a machine, you don't have to switch the old machine off while scaling. If done effectively, there may never be a need for downtime and clients are less likely to be impacted.
- **Increased resilience and fault tolerance** - Relying on a single node for all your data and operations puts you at a high risk of losing it all when it fails. Distributing it among several nodes saves you from losing it all.
- **Increased performance** - If you are using horizontal scaling to manage your network traffic, it allows for more endpoints for connections, considering that the load will be delegated among multiple machines.

Disadvantages of horizontal scaling

- **Increased complexity of maintenance and operation** - Multiple servers are harder to maintain than a single server is. Additionally, you will need to add software for load balancing and possibly virtualization. Backing up your machines may also become a little more complex. You will need to ensure that nodes synchronize and communicate effectively.
- **Increased Initial costs** - Adding new servers is far more expensive than upgrading old ones.

Advantages of vertical scaling

- **Cost-effective** - Upgrading a pre-existing server costs less than purchasing a new one. Additionally, you are less likely to add new backup and virtualization software when scaling vertically. Maintenance costs may potentially remain the same too.
- **Less complex process communication** - When a single node handles all the layers of your services, it will not have to synchronize and communicate with other machines to work. This may result in faster responses.
- **Less complicated maintenance** - Not only is maintenance cheaper but it is less complex because of the number of nodes you will need to manage.
- **Less need for software changes** - You are less likely to change how the software on a server works or how it is implemented.

Disadvantages of vertical scaling

- **Higher possibility for downtime** - Unless you have a backup server that can handle operations and requests, you will need some considerable downtime to upgrade your machine.
- **Single point of failure** - Having all your operations on a single server increases the risk of losing all your data if a hardware or software failure was to occur.
- **Upgrade limitations** - There is a limitation to how much you can upgrade a machine. Every machine has its threshold for RAM, storage, and processing power.

Which Should You Choose And When?

Both horizontal and vertical scaling have their own benefits and limitations. Since there isn't a one-size-fits-all solution for organizations, you need to scale according to your needs and resources. Here are a few factors to consider along with which type of scaling suits the situation best:

- **Cost** - Initial hardware costs for horizontal upgrades are higher. If you are working on a tight budget and need to add more resources to your infrastructure quickly and cheaply, then vertical scaling may be the best option for you.
- **Future-proofing** - Adding additional updated machines through horizontal scaling will increase the overall performance threshold of your organization. There is a limit to how much you can vertically scale a single node and it may not be able to handle the demands of the future.
- **Topographic distribution** - If you plan to have nationwide or global clients, it is unreasonable to expect them all to access your services from a single machine in a single location. In a situation like this, you'll need to horizontally scale your resources to maintain your service level agreement (SLA).
- **Reliability** - Horizontal scaling may offer you a more reliable system. It increases redundancy and ensures that you are not relying on a single machine. If one machine fails, another may be able to pick up the slack temporarily.
- **Upgradability and flexibility** - If you are running your application's tiers on individual machines, they are easier to decouple and upgrade without any downtime.
- **Performance and complexity** - Performance will depend on how your services work and how they are interconnected. Simple straightforward applications won't benefit much from being run on multiple machines. In fact, it may degrade its quality. Sometimes it's better to leave the application as is and upgrade the hardware to meet demand. Horizontal scaling may require you to rewrite the code or add a virtual machine that unifies all the servers.

On-Premise Vs. Cloud Scaling

For the majority of this guide, we've chosen to keep things simple by using on-premise non-cloud scaling for our examples. However, cloud scaling works much the same.

A cloud service provider (CSP) may implement hyper-converged infrastructure-based horizontal scaling or choose to use virtual distributed services.

The former is quite common among private and hybrid cloud solutions. In most cases, your cloud provider will handle the scaling. This means you or your IT management won't have to worry as much about what new hardware is required to meet new demands.

Service providers such as Azure and AWS have automatic scaling.

They can increase and decrease resources according to your requirements at any given time. They can scale up or out when traffic to your application is at its peak and scale down when demand is lessened. This provides organizations with more efficient and cost-effective scaling. This is another reason to consider cloud migration.

Cost: The Grand Determinant

Despite your aspirations or organization's needs, what may determine your decision, in the end, is cost. While horizontal scaling sounds great from a functional standpoint, you may not be able to afford it (right now). Nevertheless, it is still important to note that on-premise vertical and horizontal scaling may not be the only options available to you.

You can integrate both or migrate your organization's infrastructure to a cloud service provider and allow them to handle scaling for you. The latter may be more financially and pragmatically feasible for you, especially in the long run.

However, how do you actually prove this? If you migrate to a cloud solution, how do you determine your present and future cloud expenditure?

A cloud cost management platform may be the best way to do this. You can determine and prove that migration and cloud auto-scaling will ultimately be more cost-effective than on-premise scaling.

CloudZero has assisted companies such as ResponseTap to improve cost predictability and scale more efficiently by allowing them to see exactly which features and products impact their AWS spend. CloudZero allows companies to map and view a detailed breakdown of their total cloud spend – from the highest level down to the most basic components.

What is a virtual machine?

A **Virtual Machine** (VM) is a compute resource that uses software instead of a physical computer to run programs and deploy apps. One or more virtual "guest" machines run on a physical "host" machine. Each virtual machine runs its own operating

system and functions separately from the other VMs, even when they are all running on the same host. This means that, for example, a virtual MacOS virtual machine can run on a physical PC.

Virtual machine technology is used for many use cases across on-premises and cloud environments. More recently, public cloud services are using virtual machines to provide virtual application resources to multiple users at once, for even more cost efficient and flexible compute.

What are virtual machines used for?

Virtual machines (VMs) allow a business to run an operating system that behaves like a completely separate computer in an app window on a desktop. VMs may be deployed to accommodate different levels of processing power needs, to run software that requires a different operating system, or to test applications in a safe, sandboxed environment.

Virtual machines have historically been used for server virtualization, which enables IT teams to consolidate their computing resources and improve efficiency. Additionally, virtual machines can perform specific tasks considered too risky to carry out in a host environment, such as accessing virus-infected data or testing operating systems. Since the virtual machine is separated from the rest of the system, the software inside the virtual machine cannot tamper with the host computer.

How do virtual machines work?

The virtual machine runs as a process in an application window, similar to any other application, on the operating system of the physical machine. Key files that make up a virtual machine include a log file, NVRAM setting file, virtual disk file and configuration file. \

Advantages of virtual machines

Virtual machines are easy to manage and maintain, and they offer several advantages over physical machines:

- VMs can run multiple operating system environments on a single physical computer, saving physical space, time and management costs.

- Virtual machines support legacy applications, reducing the cost of migrating to a new operating system. For example, a Linux virtual machine running a distribution of Linux as the guest operating system can exist on a host server that is running a non-Linux operating system, such as Windows.
- VMs can also provide integrated disaster recovery and application provisioning options.

Disadvantages of virtual machines

While virtual machines have several advantages over physical machines, there are also some potential disadvantages:

- Running multiple virtual machines on one physical machine can result in unstable performance if infrastructure requirements are not met.
- Virtual machines are less efficient and run slower than a full physical computer. Most enterprises use a combination of physical and virtual infrastructure to balance the corresponding advantages and disadvantages.

The two types of virtual machines

Users can choose from two different types of virtual machines—process VMs and system VMs:

A process virtual machine allows a single process to run as an application on a host machine, providing a platform-independent programming environment by masking the information of the underlying hardware or operating system. An example of a process VM is the Java Virtual Machine, which enables any operating system to run Java applications as if they were native to that system.

A system virtual machine is fully virtualized to substitute for a physical machine. A system platform supports the sharing of a host computer's physical resources between multiple virtual machines, each running its own copy of the operating system. This virtualization process relies on a hypervisor, which can run on bare hardware, such as VMware ESXi, or on top of an operating system.

What are 5 types of virtualization?

All the components of a traditional data center or IT infrastructure can be virtualized today, with various specific types of virtualization:

- **Hardware virtualization**: When virtualizing hardware, virtual versions of computers and operating systems (VMs) are created and consolidated into a single, primary, physical server. A hypervisor communicates directly with a physical server's disk space and CPU to manage the VMs. Hardware virtualization, which is also known as server virtualization, allows hardware resources to be utilized more efficiently and for one machine to simultaneously run different operating systems.
- **Software virtualization**: Software virtualization creates a computer system complete with hardware that allows one or more guest operating systems to run on a physical host machine. For example, Android OS can run on a host machine that is natively using a Microsoft Windows OS, utilizing the same hardware as the host machine does. Additionally, applications can be virtualized and delivered from a server to an end user's device, such as a laptop or smartphone. This allows employees to access centrally hosted applications when working remotely.
- **Storage virtualization**: Storage can be virtualized by consolidating multiple physical storage devices to appear as a single storage device. Benefits include increased performance and speed, load balancing and reduced costs. Storage virtualization also helps with disaster recovery planning, as virtual storage data can be duplicated and quickly transferred to another location, reducing downtime.
- **Network virtualization**: Multiple sub-networks can be created on the same physical network by combining equipment into a single, software-based virtual network resource. Network virtualization also divides available bandwidth into multiple, independent channels, each of which can be assigned to servers and devices in real time. Advantages include increased reliability, network speed, security and better monitoring of data usage. Network virtualization can be a good choice for companies with a high volume of users who need access at all times.

- **Desktop virtualization**: This common type of virtualization separates the desktop environment from the physical device and stores a desktop on a remote server, allowing users to access their desktops from anywhere on any device. In addition to easy accessibility, benefits of virtual desktops include better data security, cost savings on software licenses and updates, and ease of management.

Container vs virtual machine

Like virtual machines, container technology such as Kubernetes is similar in the sense of running isolated applications on a single platform. While virtual machines virtualize the hardware layer to create a "computer," containers package up just a single app along with its dependencies. Virtual machines are often managed by a hypervisor, whereas container systems provide shared operating system services from the underlying host and isolate the applications using virtual-memory hardware.

A key benefit of containers is that they have less overhead compared to virtual machines. Containers include only the binaries, libraries and other required dependencies, and the application. Containers that are on the same host share the same operating system kernel, making containers much smaller than virtual machines. As a result, containers boot faster, maximize server resources, and make delivering applications easier. Containers have become popluar for use cases such as web applications, DevOps testing, microservices and maximizing the number of apps that can be deployed per server.

Virtual machines are larger and slower to boot than containers. They are logically isolated from one another, with their own operating system kernel, and offer the benefits of a completely separate operating system. Virtual machines are best for running multiple applications together, monolithic applications, isolation between apps, and for legacy apps running on older operating systems. Containers and virtual machines may also be used together.

Setting up a virtual machine

Virtual machines can be simple to set up, and there are many guides online that walk users through the process. VMware offers one such useful virtual machine set-up guide.

What is an Ethernet Switch?

Ethernet switching connects wired devices such as computers, laptops, routers, servers, and printers to a local area network (LAN). Multiple Ethernet switch ports allow for faster connectivity and smoother access across many devices at once.

An Ethernet switch creates networks and uses multiple ports to communicate between devices in the LAN. Ethernet switches differ from routers, which connect networks and use only a single LAN and WAN port. A full wired and wireless corporate infrastructure provides wired connectivity and Wi-Fi for wireless connectivity.

Hubs are similar to Ethernet switches in that connected devices on the LAN will be wired to them, using multiple ports. The big difference is that hubs share bandwidth equally among ports, while Ethernet switches can devote more bandwidth to certain ports without degrading network performance. When many devices are active on a network, Ethernet switching provides more robust performance.

Routers connect networks to other networks, most commonly connecting LANs to wide area networks (WANs). Routers are usually placed at the gateway between networks and route data packets along the network.

Most corporate networks use combinations of switches, routers, and hubs, and wired and wireless technology.

What Ethernet Switches Can Do For Your Network

Ethernet switches provide many advantages when correctly installed, integrated, and managed. These include:

1. Reduction of network downtime
2. Improved network performance and increased available bandwidth on the network
3. Relieving strain on individual computing devices
4. Protecting the overall corporate network with more robust security
5. Lower IT capex and opex costs thanks to remote management and consolidated wiring
6. Right-sizing IT infrastructure and planning for future expansion using modular switches

Most corporate networks support a combination of wired and wireless technologies, including Ethernet switching as part of the wired infrastructure. Dozens of devices can connect to a

network using an Ethernet switch, and administrators can monitor traffic, control communications among machines, securely manage user access, and rapidly troubleshoot.

The switches come in a wide variety of options, meaning organizations can almost always find a solution right-sized for their network. These range from basic unmanaged network switches offering plug-and-play connectivity, to feature-rich Gigabit Ethernet switches that perform at higher speeds than wireless options.

How Ethernet Switches Work: Terms and Functionality

Frames are sequences of information, travel over Ethernet networks to move data between computers. An Ethernet frame includes a destination address, which is where the data is traveling to, and a source address, which is the location of the device sending the frame. In a standard seven-layer Open Systems Interconnection (OSI) model for computer networking, frames are part of Layer 2, also known as the data-link layer. These are sometimes known as "link layer devices" or "Layer 2 switches."

Transparent Bridging is the most popular and common form of bridging, crucial to Ethernet switch functionality. Using transparent bridging, a switch automatically begins working without requiring any configuration on a switch or changes to the computers in the network (i.e. the operation of the switch is transparent).

Address Learning -- Ethernet switches control how frames are transmitted between switch ports, making decisions on how traffic is forwarded based on 48-bit media access control (MAC) addresses that are used in LAN standards. An Ethernet switch can learn which devices are on which segments of the network using the source addresses of the frames it receives.

Every port on a switch has a unique MAC address, and as frames are received on ports, the software in the switch looks at the source address and adds it to a table of addresses it constantly updates and maintains. (This is how a switch "discovers" what devices are reachable on which ports.) This table is also known as a forwarding database, which is used by the switch to make decisions on how to filter traffic to reach certain destinations. That the Ethernet switch can "learn" in this manner makes it possible for network administrators to add new connected endpoints to the network without having to manually configure the switch or the endpoints.

Traffic Filtering -- Once a switch has built a database of addresses, it can smoothly select how it filters and forwards traffic. As it learns addresses, a switch checks frames and makes decisions based on the destination address in the frame. Switches can also isolate traffic to only those segments needed to receive frames from senders, ensuring that traffic does not unnecessarily flow to other ports.

Frame Flooding -- Entries in a switch's forwarding database may drop from the list if the switch doesn't see any frames from a certain source over a period of time. (This keeps the forwarding database from becoming overloaded with "stale" source information.) If an entry is dropped—meaning it once again is unknown to the switch—but traffic resumes from that entry at a later time, the switch will forward the frame to all switch ports (also known as frame flooding) to search for its correct destination. When it connects to that destination, the switch once again learns the correct port, and frame flooding stops.

Multicast Traffic -- LANs are not only able to transmit frames to single addresses, but also capable of sending frames to multicast addresses, which are received by groups of endpoint destinations. Broadcast addresses are a specific form of multicast address; they group all of the endpoint destinations in the LAN. Multicasts and broadcasts are commonly used for functions such as dynamic address assignment, or sending data in multimedia applications to multiple users on a network at once, such as in online gaming. (Streaming applications such as video, which send high rates of multicast data and generate a lot of traffic, can hog network bandwidth.

Managed vs. Unmanaged Ethernet Switches

Unmanaged Ethernet switching refers to switches that have no user configuration; these can just be plugged in and turned on.

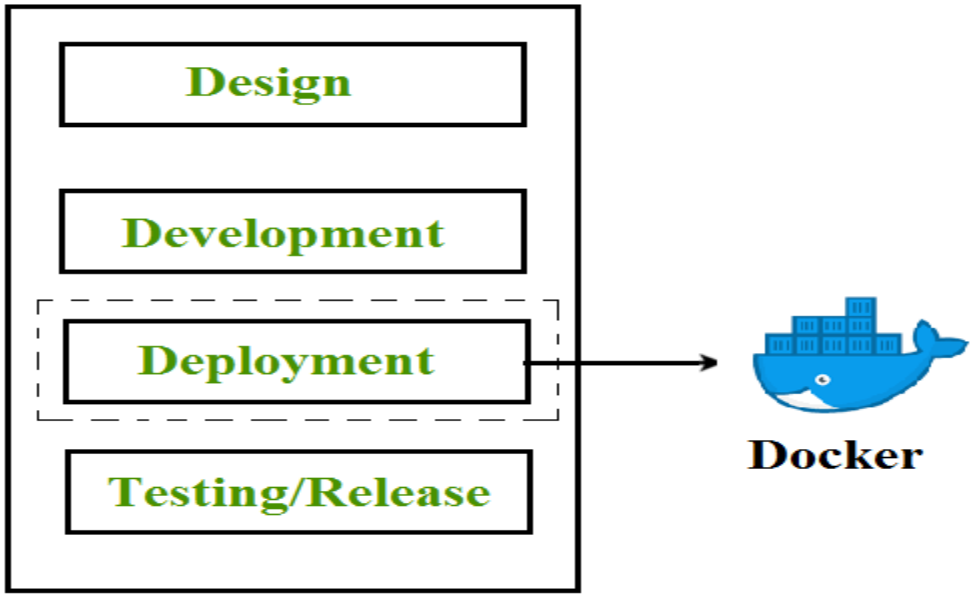
Managed Ethernet switching refers to switches that can be managed and programmed to deliver certain outcomes and perform certain tasks, from adjusting speeds and combining users into subgroups, to monitoring network traffic.

Containerization using Docker

Docker is the containerization platform that is used to package your application and all its dependencies together in the form of containers to make sure that your application works seamlessly in any environment which can be developed or tested or in production. Docker is a

tool designed to make it easier to create, deploy, and run applications by using containers.

Docker is the world's leading software container platform. It was launched in 2013 by a company called Dotcloud, Inc which was later renamed Docker, Inc. It is written in the Go language. It has been just six years since Docker was launched yet communities have already shifted to it from VMs. Docker is designed to benefit both developers and system administrators making it a part of many DevOps toolchains. Developers can write code without worrying about the testing and production environment. Sysadmins need not worry about infrastructure as Docker can easily scale up and scale down the number of systems. Docker comes into play at the deployment stage of the software development cycle.

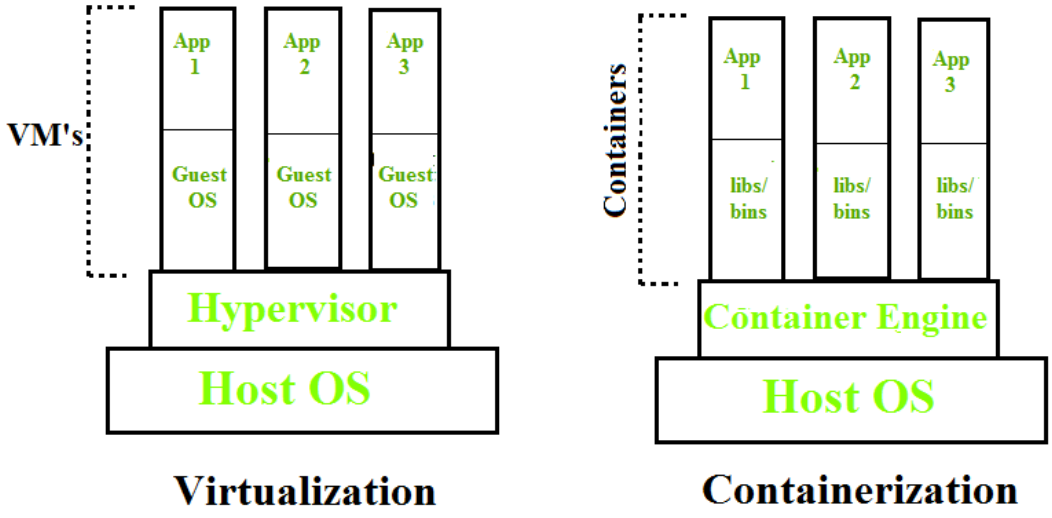


Containerization

Containerization is OS-based virtualization that creates multiple virtual units in the userspace, known as Containers. Containers share the same host kernel but are isolated from each other through private namespaces and resource control mechanisms at the OS level. Container-

based Virtualization provides a different level of abstraction in terms of virtualization and isolation when compared with hypervisors. Hypervisors use a lot of hardware which results in overhead in terms of virtualizing hardware and virtual device drivers. A full operating system (e.g -Linux, Windows) runs on top of this virtualized hardware in each virtual machine instance.
But in contrast, containers implement isolation of processes at the operating system level, thus avoiding such overhead. These containers run on top of the same shared operating system kernel of the underlying host machine and one or more processes can be run within each container. In containers you don't have to pre-allocate any RAM, it is allocated dynamically during the creation of containers while in VMs you need to first pre-allocate the memory and then create the virtual machine. Containerization has better resource utilization compared to VMs and a short boot-up process. It is the next evolution in virtualization.

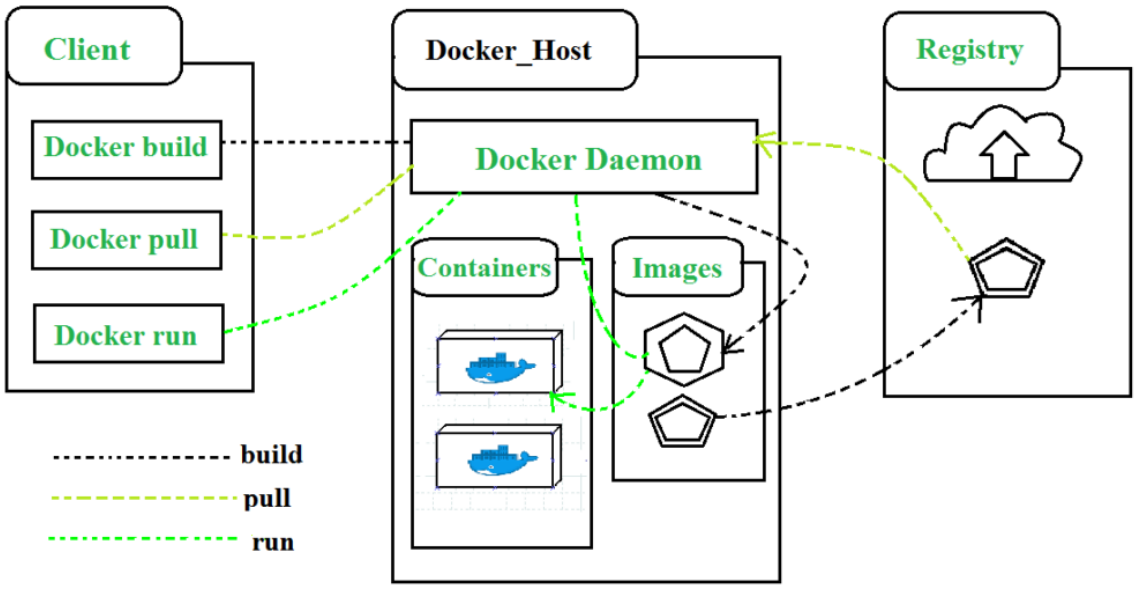
Containers can run virtually anywhere, greatly easy development and deployment: on Linux, Windows, and Mac operating systems; on virtual machines or bare metal, on a developer's machine or in data centers on-premises; and of course, in the public cloud. Containers virtualize CPU, memory, storage, and network resources at the OS level, providing developers with a sandboxed view of the OS logically isolated from other applications. Docker is the most popular open-source container format available and is supported on Google Cloud Platform and by Google Kubernetes Engine.



Docker Architecture

Docker architecture consists of Docker client, Docker Daemon running on Docker Host, and Docker Hub repository. Docker has client-server architecture in which the client communicates with the Docker Daemon running on the Docker Host using a combination of REST APIs, Socket IO, and TCP. If we have to build the Docker image, then we use the client to execute the build command to Docker Daemon then Docker Daemon builds an image based

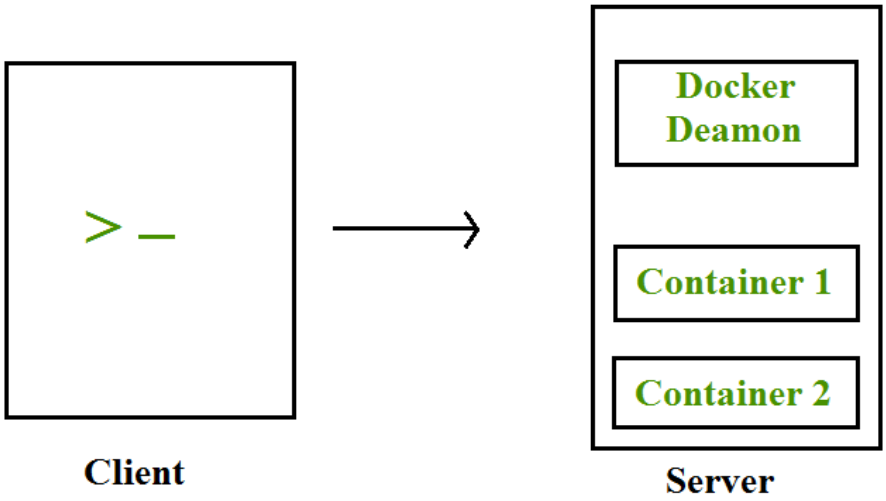
on given inputs and saves it into the Docker registry. If you don't want to create an image then just execute the pull command from the client and then Docker Daemon will pull the image from the Docker Hub finally if we want to run the image then execute the run command from the client which will create the container.



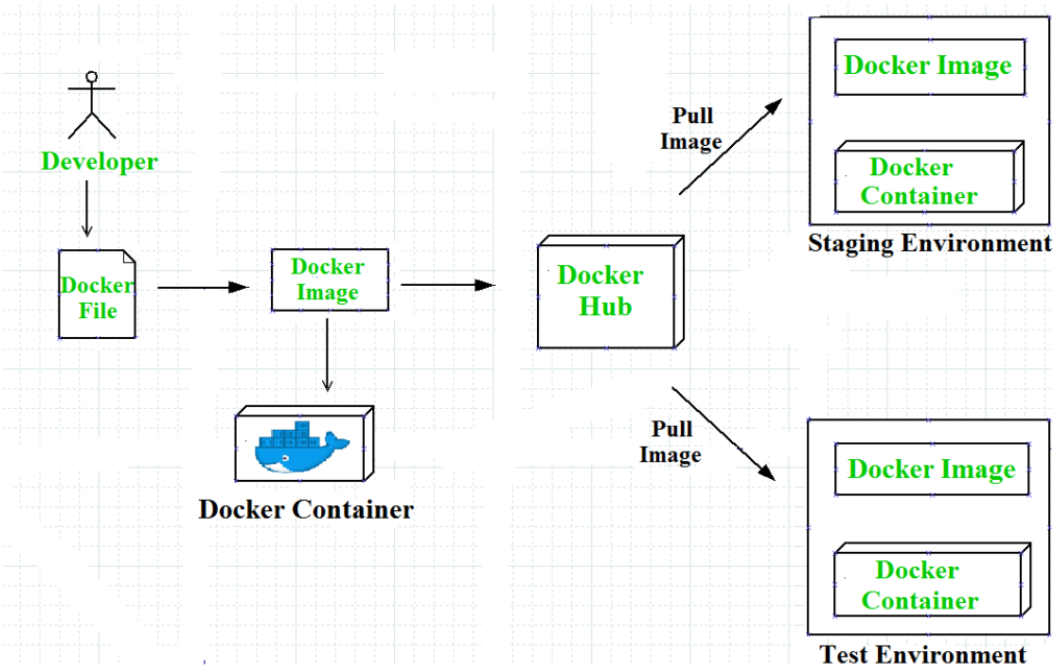
Components of Docker

The main components of Docker include – Docker clients and servers, Docker images, Dockerfile, Docker Registries, and Docker containers. These components are explained in detail in the below section :

1. **Docker Clients and Servers**– Docker has a client-server architecture. The Docker Daemon/Server consists of all containers. The Docker Daemon/Server receives the request from the Docker client through CLI or REST APIs and thus processes the request accordingly. Docker client and Daemon can be present on the same host or different host.



1. **Docker Images**– Docker images are used to build docker containers by using a read-only template. The foundation of every image is a base image eg. base images such as – ubuntu14.04 LTS, and Fedora 20. Base images can also be created from scratch and then required applications can be added to the base image by modifying it thus this process of creating a new image is called “committing the change”.
2. **Docker File**– Dockerfile is a text file that contains a series of instructions on how to build your Docker image. This image contains all the project code and its dependencies. The same Docker image can be used to spin ‘n’ number of containers each with modification to the underlying image. The final image can be uploaded to Docker Hub and shared among various collaborators for testing and deployment. The set of commands that you need to use in your Docker File is FROM, CMD, ENTRYPOINT, VOLUME, ENV, and many more.
3. **Docker Registries**– Docker Registry is a storage component for Docker images. We can store the images in either public/private repositories so that multiple users can collaborate in building the application. Docker Hub is Docker’s cloud repository. Docker Hub is called a public registry where everyone can pull available images and push their images without creating an image from scratch.
4. **Docker Containers**– Docker Containers are runtime instances of Docker images. Containers contain the whole kit required for an application, so the application can be run in an isolated way. For eg.- Suppose there is an image of Ubuntu OS with NGINX SERVER when this image is run with the docker run command, then a container will be created and NGINX SERVER will be running on Ubuntu OS.



Docker Compose

Docker Compose is a tool with which we can create a multi-container application. It makes it easier to configure and
run applications made up of multiple containers. For example, suppose you had an application that required WordPress and MySQL, you could create one file which would start both the containers as a service without the need to start each one separately. We define a multi-container application in a YAML file. With the docker-compose-up command, we can start the application in the foreground. Docker-compose will look for the docker-compose. YAML file in the current folder to start the application. By adding the -d option to the docker-compose-up command, we can start the application in the background. Creating a docker-compose. YAML file for WordPress application :

#cat docker-compose.yaml

version: ’2’

services:

db:

image: mysql:5.7

volumes:db_data:/var/lib/mysql

restart: always

environment:

MYSQL_ROOT_PASSWORD: WordPress

MYSQL_DATABASE: WordPress

MYSQL_USER: WordPress

MYSQL_PASSWORD: WordPress

WordPress:

depends_on:

- DB

image: WordPress:latest

ports:

- "8000:80"

restart: always

environment:

WORDPRESS_DB_HOST: db:3306

WORDPRESS_DB_PASSWORD: wordpress

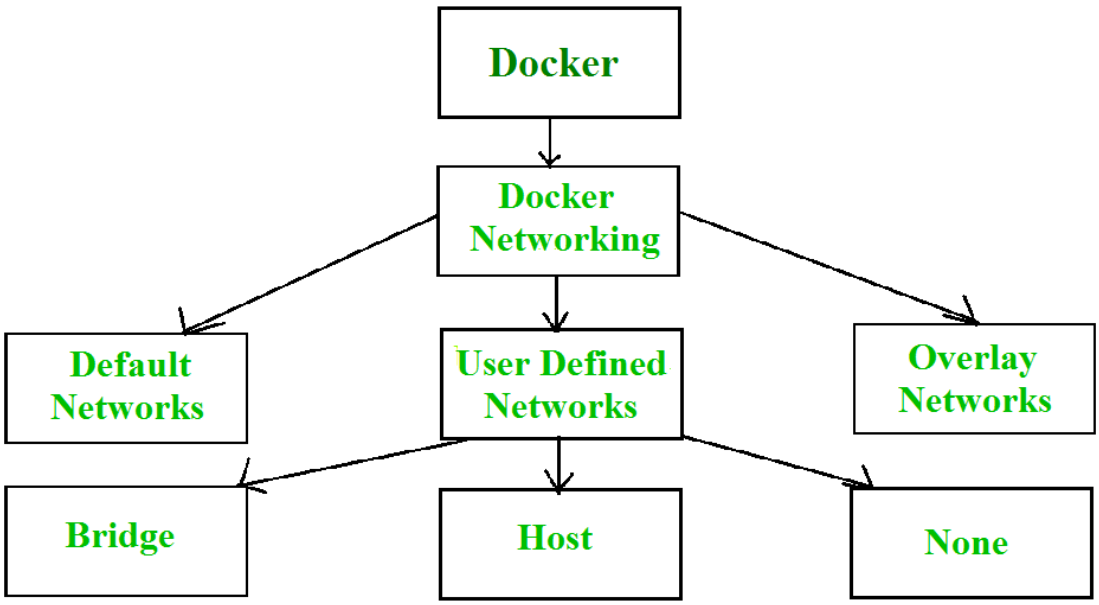
volumes:

db_data:

In this docker-compose. YAML file, we have the following ports section for the WordPress container, which means that we are going to map the host's 8000 port with the container's 80 port. So that host can access the application with its IP and port no.

Docker Networks

When we create and run a container, Docker by itself assigns an IP address to it, by default. Most of the time, it is required to create and deploy Docker networks as per our needs. So, Docker let us design the network as per our requirements. There are three types of Docker networks- default networks, user-defined networks, and overlay networks.



\

To get a list of all the default networks that Docker creates, we run the command shown below –

```
$ docker network ls

NETWORK ID          NAME                DRIVER
b48564s6sf7bd       bridge              bridge
fe56b3e2dd73d       docker_gwbridge     bridge
5d989edbddbs9       host                host
98absh67bs8m2       none                null
```

There are three types of networks in Docker –

1. **Bridged network**: When a new Docker container is created without the –network argument, Docker by default connects the container with the bridge network. In bridged networks, all the containers in a single host can connect through their IP addresses. A Bridge network is created when the span of Docker hosts is one i.e.

when all containers run on a single host. We need an overlay network to create a network that has a span of more than one Docker host.

2. **Host network**: When a new Docker container is created with the –network=host argument it pushes the container into the host network stack where the Docker daemon is running. All interfaces of the host are accessible from the container which is assigned to the host network.
3. **None network**: When a new Docker container is created with the –network=none argument it puts the Docker container in its network stack. So, in this none network, no IP addresses are assigned to the container, because of which they cannot communicate with each other.

We can assign any one of the networks to the Docker containers. The –network option of the 'docker run' command is used to assign a specific network to the container.

```
$docker run --network ="network name"
```

To get detailed information about a particular network we use the command-

```
$docker network inspect "network name"
```

Advantages of Docker –

Docker has become popular nowadays because of the benefits provided by Docker containers. The main advantages of Docker are:

1. **Speed** – The speed of Docker containers compared to a virtual machine is very fast. The time required to build a container is very fast because they are tiny and lightweight. Development, testing, and deployment can be done faster as containers are small. Containers can be pushed for testing once they have been built and then from there on to the production environment.
2. **Portability** – The applications that are built inside docker containers are extremely portable. These portable applications can easily be moved anywhere as a single element and their performance also remains the same.
3. **Scalability** – Docker has the ability that it can be deployed on several physical servers, data servers, and cloud platforms. It can also be run on every Linux machine. Containers can easily be moved from a cloud environment to a local host and from there back to the cloud again at a fast pace.
4. **Density** – Docker uses the resources that are available more efficiently because it does not use a hypervisor. This is the reason that more containers can be run on a single host as compared to virtual machines. Docker Containers have higher performance because of their high density and no overhead wastage of resources.

What is Kubernetes?

Kubernetes is software that automatically manages, scales, and maintains multi-container workloads in desired states

Modern software is increasingly run as fleets of containers, sometimes called microservices. A complete application may comprise many containers, all needing to work together in specific ways. Kubernetes is software that turns a collection of physical or virtual hosts (servers) into a platform that:

- Hosts containerized workloads, providing them with compute, storage, and network resources, and

- Automatically manages large numbers of containerized applications — keeping them healthy and available by adapting to changes and challenges

How does Kubernetes work?

1. When developers create a multi-container application, they plan out how all the parts fit and work together, how many of each component should run, and roughly what should happen when challenges (e.g., lots of users logging in at once) are encountered.

2. They store their containerized application components in a container registry (local or remote) and capture this thinking in one or several text files comprising a*configuration*. To start the application, they "apply" the configuration to Kubernetes.

3. Kubernetes job is to evaluate and implement this configuration and maintain it until told otherwise. It:

    - Analyzes the configuration, aligning its requirements with those of all the other application configurations running on the system

    - Finds resources appropriate for running the new containers (e.g., some containers might need resources like GPUs that aren't present on every host)

    - Grabs container images from the registry, starts up the new containers, and helps them connect to one another and to system resources (e.g., persistent storage), so the application works as a whole

4. Then Kubernetes monitors everything, and when real events diverge from desired states, Kubernetes tries to fix things and adapt. For example, if a container crashes, Kubernetes restarts it. If an underlying server fails, Kubernetes finds resources elsewhere to run the containers that node was hosting. If traffic to an application suddenly spikes, Kubernetes can scale out containers to handle the additional load, in conformance to rules and limits stated in the configuration.

Why use Kubernetes?

One of the benefits of Kubernetes is that it makes building and running complex applications much simpler. Here's a handful of the many Kubernetes features:

- Standard services like local DNS and basic load-balancing that most applications need, and are easy to use.

- Standard behaviors (e.g., restart this container if it dies) that are easy to invoke, and do most of the work of keeping applications running, available, and performant.

- A standard set of abstract "objects" (called things like "pods," "replicasets," and "deployments") that wrap around containers and make it easy to build configurations around collections of containers.

- A standard API that applications can call to easily enable more sophisticated behaviors, making it much easier to create applications that manage other applications.

The simple answer to "what is Kubernetes used for" is that it saves developers and operators a great deal of time and effort, and lets them focus on building features for their applications, instead of figuring out and implementing ways to keep their applications running well, at scale.

By keeping applications running despite challenges (e.g., failed servers, crashed containers, traffic spikes, etc.) Kubernetes also reduces business impacts, reduces the need for fire drills to bring broken applications back online, and protects against other liabilities, like the costs of failing to comply with Service Level Agreements (SLAs).

Where can I run Kubernetes?

Kubernetes also runs almost anywhere, on a wide range of Linux operating systems (worker nodes can also run on Windows Server). A single Kubernetes cluster can span hundreds of bare-metal or virtual machines in a datacenter, private, or any public cloud. Kubernetes can also run on developer desktops, edge servers, microservers like Raspberry Pis, or very small mobile and IoT devices and appliances.

With some forethought (and the right product and architectural choices) Kubernetes can even provide a functionally-consistent platform across all these infrastructures. This means that applications and configurations composed and initially tested on a desktop Kubernetes can move seamlessly and quickly to more-formal testing, large-scale production, edge, or IoT deployments. In principle, this means that enterprises and organizations can build “hybrid” and “multi-clouds” across a range of platforms, quickly and economically solving capacity problems without lock-in.

What is a Kubernetes cluster?

The K8s architecture is relatively simple. You never interact directly with the nodes hosting your application, but only with the control plane, which presents an API and is in charge of scheduling and replicating groups of containers named Pods. Kubectl is the command line interface that allows you to interact with the API to share the desired application state or gather detailed information on the infrastructure’s current state.

Let’s look at the various pieces.

Nodes

Each node that hosts part of your distributed application does so by leveraging Docker or a similar container technology, such as Rocket from CoreOS. The nodes also run two additional pieces of software: kube-proxy, which gives access to your running app, and kubelet, which receives commands from the k8s control plane. Nodes can also run flannel, an etcd backed network fabric for containers.

Master

The control plane itself runs the API server (kube-apiserver), the scheduler (kube-scheduler), the controller manager (kube-controller-manager) and etcd, a highly available key-value store for shared configuration and service discovery implementing the Raft consensus Algorithm.

What is “enterprise Kubernetes?”

Kubernetes, by itself, provides a core software framework for container and resource management, default services, plus an API. It’s engineered to be extensible via standard interfaces to provide important capabilities like:

- Running containers – a container runtime or ‘engine’

- Letting containers communicate – a container network
- Providing persistent storage – a container storage solution
- Routing inbound traffic to containers in a secure and orderly way – an ingress solution
- Full-featured load balancing – distributing inbound traffic evenly to container workloads – via integration with an external load-balancing solution

… and many other components essential for efficient use and operations at scale. To make Kubernetes work at all — you or someone else needs to choose and integrate solutions to fill these critical slots.

Kubernetes alternatives made available free of charge typically select from among open source alternatives to provide these capabilities. These are often very good solutions for learning and small-scale use.

Organizations that want to use Kubernetes to run production software at scale need more, and more-mature functionality:

- They need Kubernetes that's feature-complete, hardened and secure, and easily integrated with centralized IT resources like directory services, monitoring and observability, notifications and ticketing, and so on.
- They need Kubernetes that can be deployed, scaled, managed, and updated in consistent ways, perhaps across many different kinds of infrastructure.
- They need all the different parts of Kubernetes to be validated together, and supported by a single vendor.

"Enterprise Kubernetes" refers to products and suites of products that answer these needs: that fill all of Kubernetes' feature slots with best-of-breed solutions, solve problems of Kubernetes management across multiple infrastructures, enable consistency, and provide complete support.

How do I start using Kubernetes?

Mirantis makes several Kubernetes solutions, appropriate for different uses. Our open source products can be used free of charge, with community support. Our flagship products can be trialed free of charge and are available with tiered support up to fully-managed services.

2	2	1	2	2	2	2
3	2	1	2	2	2	2
4	3	1	3	3	2	3
5	3	1	3	3	2	3
Avg	2.4	1	2.4	2.4	2	2.4

MC4205 **CYBER SECURITY** **L T P C**
3 0 0 3

COURSE OBJECTIVES:

- To learn the principles of cyber security and to identify threats and risks.
- To learn how to secure physical assets and develop system security controls.
- To understand how to apply security for Business applications and Network Communications.
- To learn the technical means to achieve security.
- To learn to monitor and audit security measures.

UNIT I PLANNING FOR CYBER SECURITY 9

Best Practices-Standards and a plan of Action-Security Governance Principles, components and Approach-Information Risk Management-Asset Identification-Threat Identification-Vulnerability Identification-Risk Assessment Approaches-Likelihood and Impact Assessment-Risk Determination, Evaluation and Treatment-Security Management Function-Security Policy-Acceptable Use Policy-Security Management Best Practices - Security Models: Bell La Padula model, Biba Integrity Model - Chinese Wall model

UNIT II SECURITY CONTROLS 9

People Management-Human Resource Security-Security Awareness and Education-Information Management- Information Classification and handling-Privacy-Documents and Record Management-Physical Asset Management-Office Equipment-Industrial Control Systems-Mobile Device Security-System Development-Incorporating Security into SDLC - Disaster management and Incident response planning.

UNIT III CYBER SECURITY FOR BUSINESS APPLICATIONS AND NETWORKS 9

Business Application Management-Corporate Business Application Security-End user Developed Applications-System Access- Authentication Mechanisms-Access Control-System Management-Virtual Servers-Network Storage Systems-Network Management Concepts-Firewall-IP Security-Electronic Communications - Case study on OWASP vulnerabilities using OWASP ZAP tool.

UNIT IV TECHNICAL SECURITY 9

Supply Chain Management-Cloud Security-Security Architecture-Malware Protection-Intrusion Detection-Digital Rights Management-Cryptographic Techniques-Threat and Incident Management-Vulnerability Management-Security Event Management-Forensic Investigations-Local Environment Management-Business Continuity.

UNIT V SECURITY ASSESSMENT 9

Security Monitoring and Improvement-Security Audit-Security Performance-Information Risk Reporting-Information Security Compliance Monitoring-Security Monitoring and Improvement Best

UNIT I PLANNING FOR CYBER SECURITY

Best Practices-Standards and a plan of Action-Security Governance Principles, components and Approach-Information Risk Management-Asset Identification-Threat Identification-Vulnerability Identification-Risk Assessment Approaches-Likelihood and Impact Assessment-Risk Determination, Evaluation and Treatment-Security Management Function-Security Policy- Acceptable Use Policy-Security Management Best Practices - Security Models: Bell La Padula model, Biba Integrity Model -Chinese Wall model

Practices-Standards and a plan of Action

Defining Cyberspace and Cyber security

Cyberspace consists of artifacts based on or dependent on computer and communications technology; the information that these artifacts use, store, handle, or process; and the interconnections among these various elements.

Cyber security

- Cyber security is the collection of tools, policies, security concepts, security safeguards, guidelines, risk management approaches, actions, training, best practices, assurance and technologies that are used to protect the cyberspace environment and organization and user's assets.

Two related terms should be mentioned:

Information security: Preservation of confidentiality, integrity, and availability of information. In addition, other properties—such as authenticity, accountability, non-repudiation, and reliability—can also be involved.

Network security: Protection of networks and their services from unauthorized modification, destruction, or disclosure and provision of assurance that the network performs its critical functions correctly and that there are no harmful side effects.

Cyber security encompasses information security, with respect to electronic information, and network security. Information security also is concerned with physical (for example, paper-based) information. However, in practice, the terms cyber security and information security are often used interchangeably.

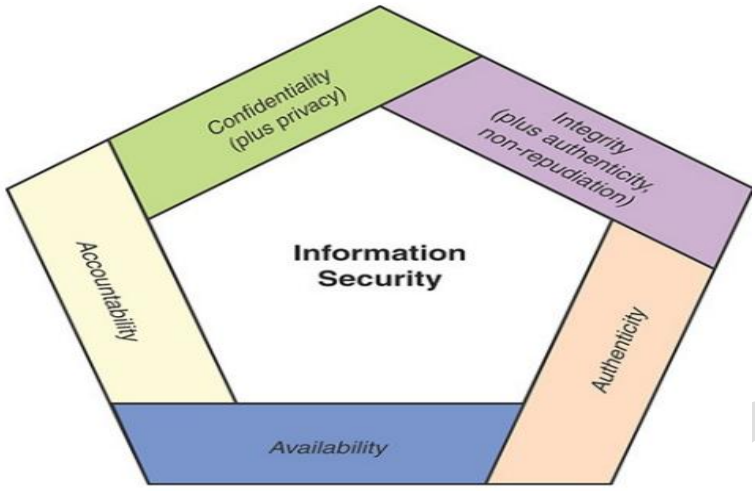


FIGURE 1.1 Essential Cybersecurity Objectives

A more extensive list of cyber security objectives includes the following:

Availability: The property of a system or a system resource being accessible or usable or operational upon demand, by an authorized system entity, according to performance specifications for the system; that is, a system is available if it provides services according to the system design whenever users request them.

Integrity: The property that data has not been changed, destroyed, or lost in an unauthorized or accidental manner.

Authenticity: The property of being genuine and being able to be verified and trusted. This means verifying that users are who they say they are and that each input arriving at the system came from a trusted source.

Non-repudiation: Assurance that the sender of information is provided with proof of delivery and the recipient is provided with proof of the sender's identity, so neither can later deny having processed the information.

Confidentiality: The property that data is not disclosed to system entities unless they have been authorized to know the data.

Accountability: The property of a system or system resource ensuring that the actions of a system entity may be traced uniquely to that entity, which can then be held responsible for its actions.

The Value of Standards and Best Practices Documents

- The development, implementation, and management of a cybersecurity system for an organization are extraordinarily complex and difficult.
- A wide variety of technical approaches are involved, including cryptography, network security protocols, operating system mechanisms, database security schemes, and malware identification.
- The areas of concern are broad, including stored data, data communications, human factors, physical asset and property security, and legal, regulatory, and contractual concerns.
- And there is an ongoing need to maintain high confidence in the cyber security capability in the face of evolving IT systems, relationships with outside parties, personnel turnover, changes to the physical plant, and the ever-evolving threat landscape.

- On the standards side, the most prominent player is the National Institute of Standards and Technology (NIST).
- IST has a huge number of security publications, including nine Federal Information Processing Standards (FIPS) and well 100 active Special Publications (SP) that provide guidance on virtually all aspects of cyber security.
- Other organization that have produced cyber security standards and guidelines include the ITU-T, International Organization for Standardization (ISO), and the Internet Society (ISOC).
- In addition, a number of professional and industry groups have produced best practices documents and guidelines. The most important such document is the Standard of Good Practice for Information Security, produced by the Information Security Forum (ISF).
- This 300-plus-page document provides a wide range of best practices representing the consensus of industry and government organizations.
- Other respected organizations, including the Information Systems Audit and Control Association (ISACA) and the Payment Card Industry (PCI), have produced a number of similar documents.

Table 1.1 lists the most prominent best practices and standards documents that are discussed in this book.

TABLE 1.1 Important Best Practices and Standards Documents

Source	Title	Date
ISF	Standard of Good Practice for Information Security	2016
ISO	ISO 27002: Code of Practice for Information Security Controls	2013
NIST	Framework for Improving Critical Infrastructure Cybersecurity	2017
Center for Internet Security (CIS)	CIS Critical Security Controls for Effective Cyber Defense Version 7	2018
ISACA	COBIT 5 for Information Security	2012
PCI Security Standards Council	Data Security Standard v3.2: Requirements and Security Assessment Procedures	2016

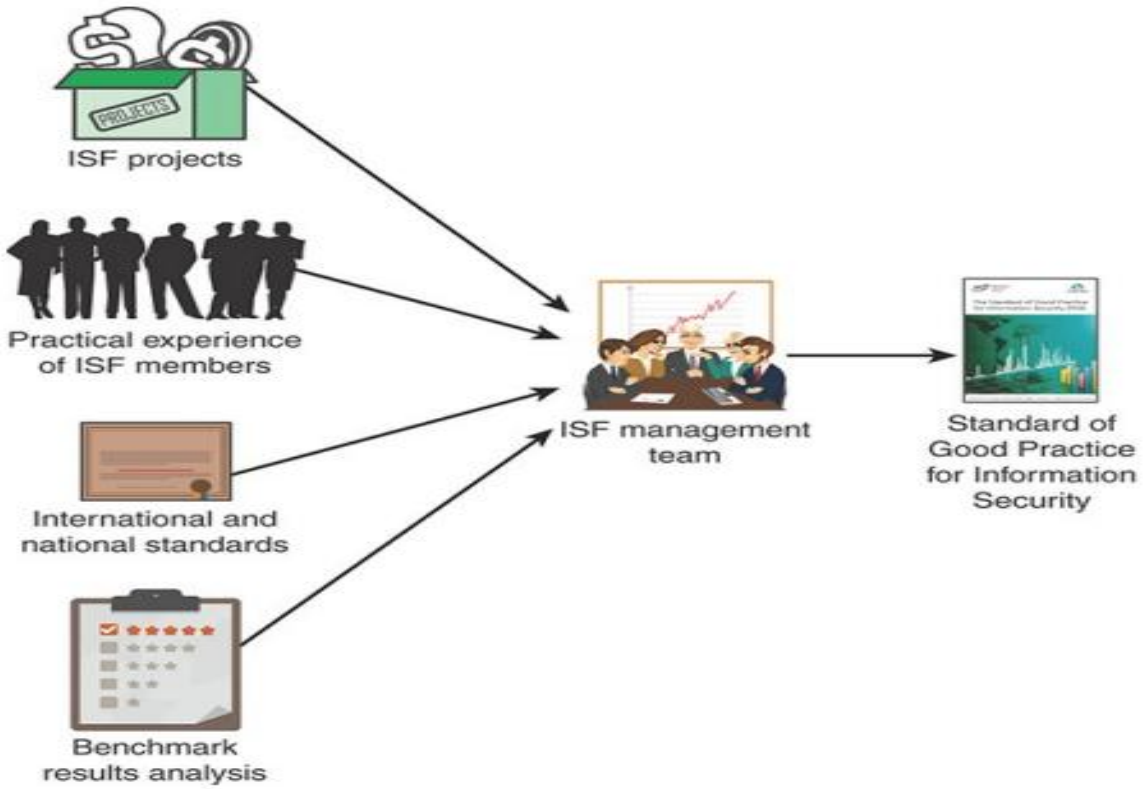


FIGURE 1.2 Basis for the ISF Standard of Good Practice for Information

The SGP is of particular interest to the following individuals:

Chief information security officers (or equivalent): Responsible for developing policy and implementing sound information security governance and information security assurance.

Information security managers (as well as security architects, local security coordinators, and information protection champions): Responsible for promoting or implementing an information security assurance **program**

Business managers: Responsible for ensuring that critical business applications, processes, and local environments on which an organization's success depends are effectively managed and controlled

IT managers and technical staff: Responsible for designing, planning, developing, deploying, and maintaining key business applications, information systems, or facilities security policy A set of rules and practices that specify or regulate how a system or organization provides security services to protect sensitive and critical system resources.

Internal and external auditors: Responsible for conducting security audits

IT service providers: Responsible for managing critical facilities (for example, computer installations, networks) on behalf of the organization

Procurement and vendor management teams: Responsible for defining appropriate information security requirements in contracts

TABLE 1.2 ISF Standard of Good Practice for Information Security: Categories and Areas

Category	Areas
Security Governance (SG)	Security Governance Approach Security Governance Components
Information Risk Assessment (IR)	Information Risk Assessment Framework Information Risk Assessment Process
Security Management (SM)	Security Policy Management Information Security Management
People Management (PM)	Human Resource Security Security Awareness/Education
Information Management (IM)	Information Classification and Privacy Information Protection
Physical Asset Management (PA)	Equipment Management Mobile Computing
System Development (SD)	System Development Management System Development Life Cycle
Business Application Management (BA)	Corporate Business Applications End User Developed Applications
System Access (SA)	Access Management Customer Access
System Management (SY)	System Configuration System Maintenance
Networks and Communications (NC)	Network Management Electronic Communication
Supply Chain Management (SC)	External Supplier Management Cloud Computing
Technical Security Management (TS)	Security Solutions Cryptography
Threat and Incident Management (TM)	Cybersecurity Resilience Security Incident Management
Local Environment Management (LE)	Local Environments Physical and Environmental Security
Business Continuity (BC)	Business Continuity Framework Business Continuity Process
Security Monitoring and Improvement (SI)	Security Audit Security Performance

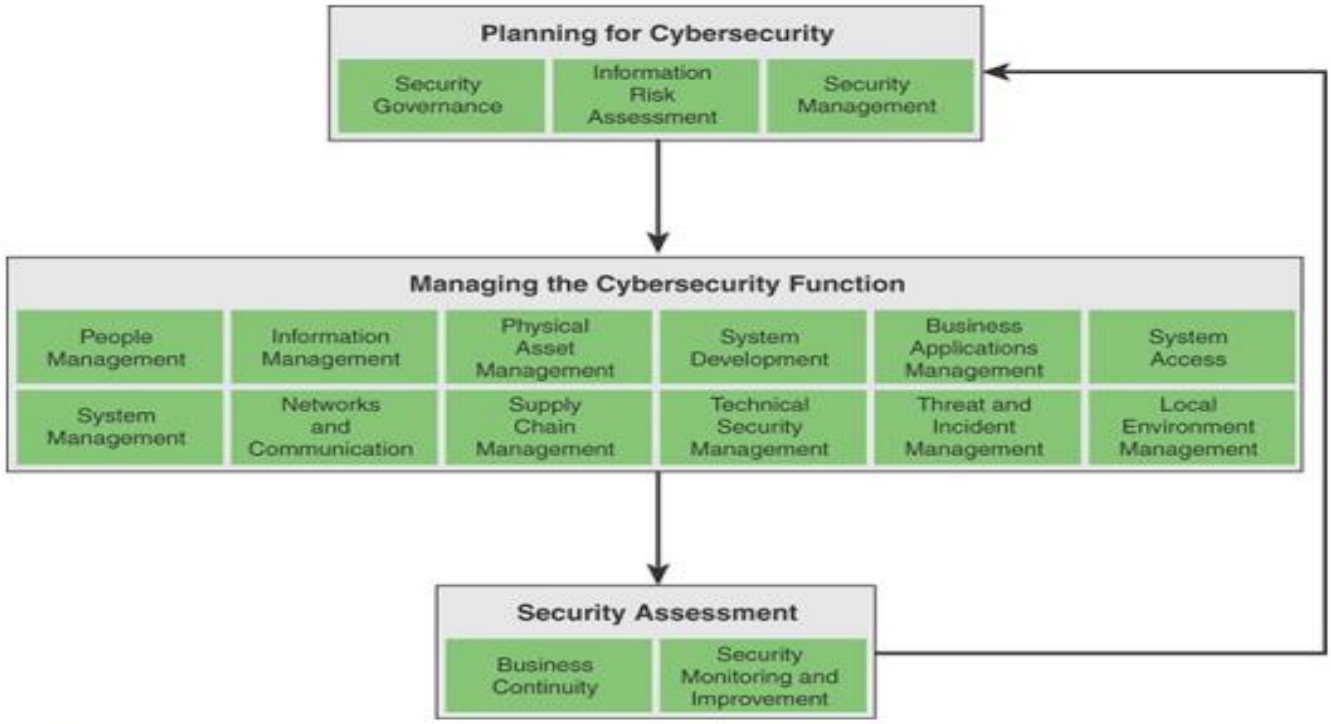


FIGURE 1.3 Categories in the Standard of Good Practice for Information Security

It is informative to consider the 17 SGP categories as being organized into three principal activities (see Figure 1.3):

1. **Planning for cybersecurity**: Developing approaches for managing and controlling the cybersecurity function(s); defining the requirements specific to a given IT environment; and developing policies and procedures for managing the security function

2. **Managing the cybersecurity function**: Deploying and managing the security controls to satisfy the defined security requirements

3. **Security assessment:** Assuring that the security management function enables business continuity; monitoring, assessing, and improving the suite of cybersecurity controls

The ISO 27000 series deals with all aspects of an ISMS. It helps small, medium, and large businesses in any sector keep information assets secure. This growing collection of standards falls into four categories (see Figure 1.4):

Overview and vocabulary: Provide an overview and relevant vocabulary for ISMS

Requirements: Discuss normative standards that define requirements for an ISMS and for those certifying such systems

Guidelines: Provide direct support and detailed guidance and/or interpretation for the overall process of establishing, implementing, maintaining, and improving an ISMS

Sector-specific guidelines: Address sector-specific guidelines for an ISM

ISMS overview and vocabulary	ISMS requirements		ISMS guidelines		ISMS sector-specific guidelines	
27000 ISMS overview	27001 ISMS requirements	27006 Audit and certification of ISMS	27002 Code of practice for IS controls	27003 ISMS implementation	27010 Intersector/ interorganiza-tional comms	27011 Telecomms organizations
	27009 Sector-specific application		27004 ISM measurement	27005 IS risk management	27015 Financial services	27017 IS controls for cloud services
			27007 ISMS auditing	TR 27008 Auditors on IS control	27018 Protection of PII in public clouds	27019 Energy utility industry PCS
			27013 Integrated implementation of 27001/20000	27014 Governance of IS		
			TR 27016 Organizational economics	27036 IS for supplier relationships		

ISMS = Information Security Management System
PII = personally identifiable information
PCS = process control systems

FIGURE 1.4 ISO 27000 ISMS Family of Standards

Security Governance

NIST SP 800-100, Information Security Handbook: A Guide for Managers, defines information security governance as follows:

Information security governance The process of establishing and maintaining a framework and supporting management structure and processes to provide assurance that information security strategies are aligned with and support business objectives, are consistent with applicable laws and regulations through adherence to policies and internal controls, and provide assignment of responsibility, all in an effort to manage risk.

More generally, the term security governance encompasses governance concerns for cyber security, information security, and network security.

Security Governance and Security Management

To better understand the role of security governance, it is useful to distinguish between information security governance (previously defined), information security management, and information security implementation/operations.

ISO 27000 defines information security management as follows:

The supervision and making of decisions necessary to achieve business objectives through the protection of the organization's information assets.

Management of information security is expressed through the formulation and use of information security policies, procedures and guidelines, which are then applied throughout the organization by all individuals associated with the organization.

And information security implementation/operations can be defined in this fashion:

The implementation, deployment and ongoing operation of security controls defined within a cyber security framework.

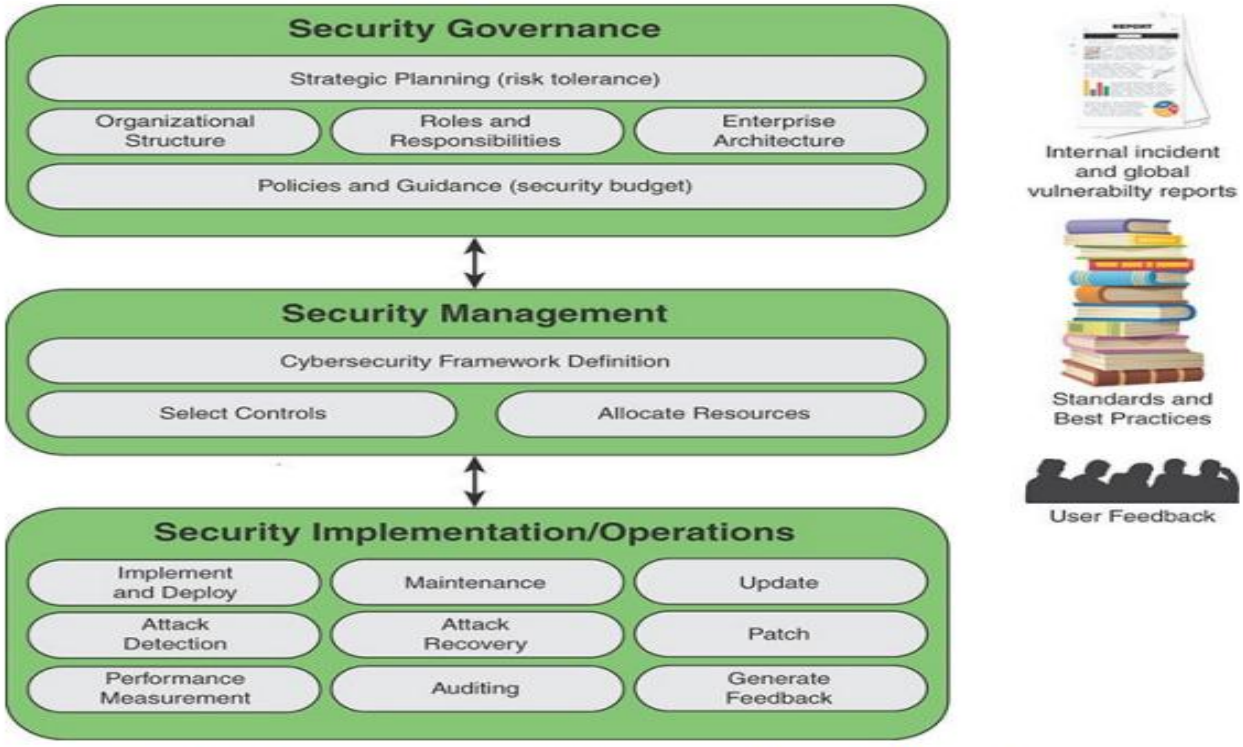


FIGURE 2.1 Information Security Management System Element

Figure 2.1 illustrates the key responsibilities at each level. As indicated, there is interaction among the three layers in the ongoing evolution of the information security management system (ISMS). In addition, three supplemental factors play roles.

Internal security incident reports and global vulnerability reports from various sources help define the threat and level of risk that the organization faces in protecting its information assets. The numerous standards and best practices documents provide guidance on managing risk.

User feedback comes from both internal users and external users who have access to the organization's information assets. This feedback helps improve the effectiveness of policies, procedures, and technical mechanisms.

Depending on the organization and its cybersecurity approach, each of the three factors plays a role to a greater or lesser extent at each level.

Security Governance Principles

Principles X.1054 provides concepts and guidance on principles and processes for information security governance, by which organizations evaluate, direct, and monitor the management of information security. X.1054 lays out as a key objective of information security governance the alignment of information security objectives and strategy with overall business objectives and strategy.

X.1054 lists six principles for achieving this objective:

Establish organizationwide information security: Information security, or cybersecurity, concerns should permeate the organization's structure and functions. Management at all levels should ensure that information security is integrated with information technology (IT) and other activities. Toplevel management should ensure that information security serves overall business objectives and should establish responsibility and accountability throughout the organization.

Adopt a risk-based approach:. Security governance, including allocation of resources and budgets, should be based on the risk appetite of an organization, considering loss of competitive advantage, compliance and liability risks, operational disruptions, reputational harm, and financial loss.

Set the direction of investment decisions: Information security investments are intended to support organizational objectives. Security governance entails ensuring that information security is integrated with existing organization processes for capital and operational expenditure, for legal and regulatory compliance, and for risk reporting.

Ensure conformance with internal and external requirements.:External requirements include mandatory legislation and regulations, standards leading to certification, and contractual requirements. Internal requirements comprise broader organizational goals and objectives. Independent security audits are the accepted means of determining and monitoring conformance.

Foster a security-positive environment for all stakeholders:Security governance should be responsive to stakeholder expectations, keeping in mind that various stakeholders can have different values and needs. The governing body should take the lead in promoting a positive information security culture, which includes requiring and supporting security education, training, and awareness programs.

Security Governance Components

The following key activities, or components that constitute effective security governances

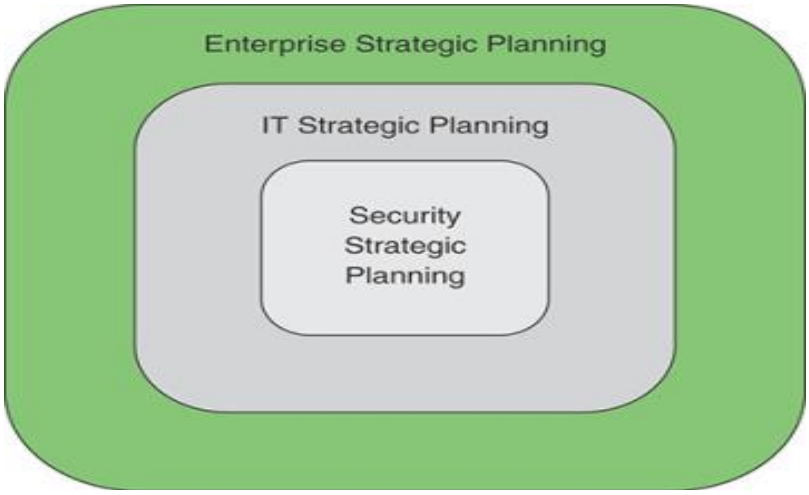
- Strategic planning
- Organizational structure
- Establishment of roles and responsibilities
- Integration with the enterprise architecture
- Documentation of security objectives in policies and guidance

The following sections examine each of these components in turn.

Strategic Planning

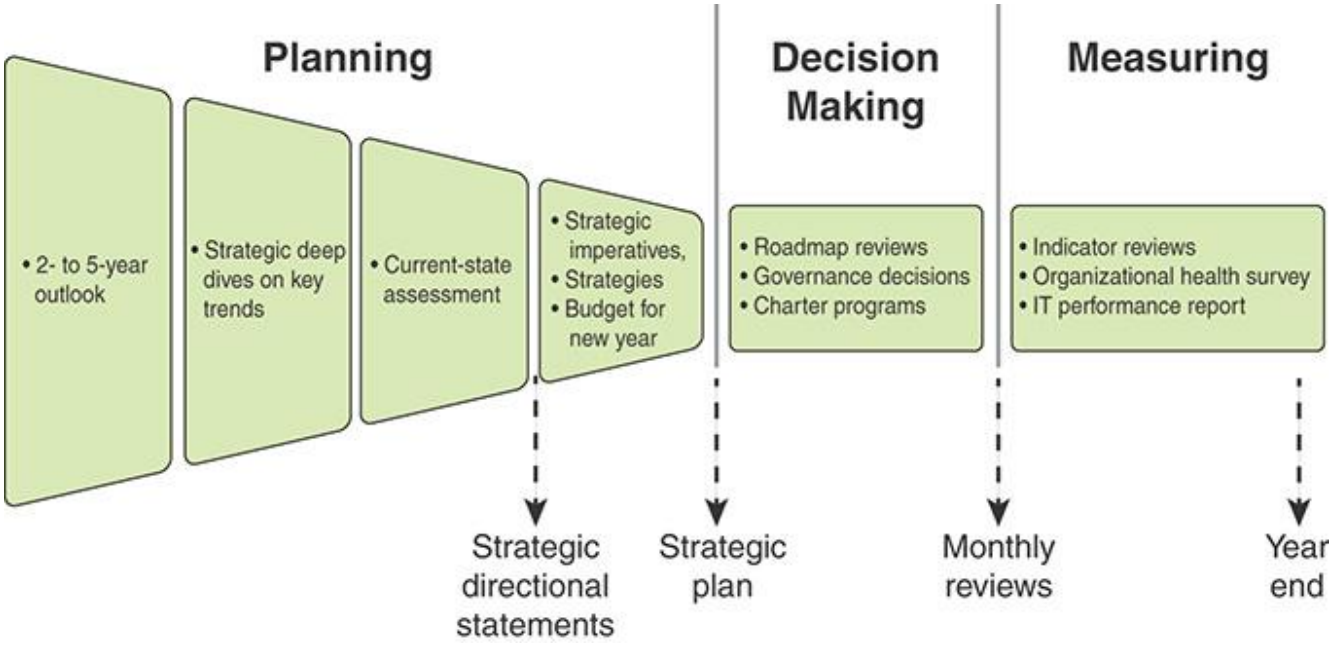
It is useful for this discussion to define three hierarchically related aspects of strategic planning (see Figure 2.2):

- Enterprise strategic planning
- Information technology (IT) strategic planning
  - Cybersecurity or information security strategic planning



Enterprise strategic planning involves defining long-term goals and objectives for an organization (for example, business enterprise, government agency, or nonprofit organization) and the development of plans to achieve these goals and objectives.

The management activity involved in enterprise strategic planning is described in the Strategic Management Group's Strategic Planning Basics [SMG17] as an activity used to set priorities, focus energy and resources, strengthen operations, ensure that employees and other stakeholders are working toward common goals, establish agreement around intended outcomes/results, and assess and adjust the organization's direction in response to a changing environment. It involves the development of a strategic plan and the ongoing oversight of the implementation of that plan.



The six phases are as follows:

1. **Two- to five-year business and technology outlook:** At the beginning of the year, the planning team takes as input an overall vision and mission statement developed at the enterprise level. During this phase, the team reviews the enterprise strategies, technology trends, employee trends, and so on to better understand the future environment that will shape the IT organization and its deliverables. IT subject matter experts from throughout the organization are recruited to help define the major trends that may be critical in shaping the organization and its decision making in the next few years.

2. **Strategic deep dive:** The team identifies a small number of high-impact areas that require more in-depth analysis to inform the overall strategic planning process. Depending on circumstances at a given point in time, these may include IoT, social media trends, and changing regulatory compliance rules.

3. **Current-state assessment:** The planning team analyzes the current state of all the IT-related systems and policies and compares these with the long-range outlook, paying special attention to the key drivers developed in the preceding

phase. The result is a set of recommendations for adjustments to IT's focus areas and spending plans.

4. **Imperatives, roadmaps, and finances:** The next phase is the development of a strategic plan for IT. The plan includes a discussion of strategic objectives and a budget and investment plan. The plan reflects IT's highest-priority items and provides an outcome framework for defining success. Each item includes a roadmap that can influence budget and organization decisions in the upcoming year.

5. **Governance process and decision making:** Once the annual budget is approved, the information from the preceding phases is used to guide the governance process and the many decisions made across the organization to implement the strategic plan and one-year strategic objectives. These decisions include project chartering, supplier selection, sourcing, investment trade-off decisions, and so on.

6. **Regular reviews:** Monthly reviews based on a wide variety of input help ensure that the strategic plan and governance decisions are followed. This culminates in a year-end assessment. Reviews continue into the following year until a new strategic plan and new governance decisions provide input for modifying the review process.

Information security strategic planning: is alignment of information security management and operation with enterprise and IT strategic planning. The pervasive use and value of IT within organizations has resulted in an expanded notion of IT's delivery of value to the organization to include mitigation of the organization's risk [ZIA15]. Accordingly, IT security is a concern at all levels of an organization's governance and decision-making processes, and information security strategic planning is an essential component of strategic planning.

TABLE 2.1 Elements of a Strategic Plan Document

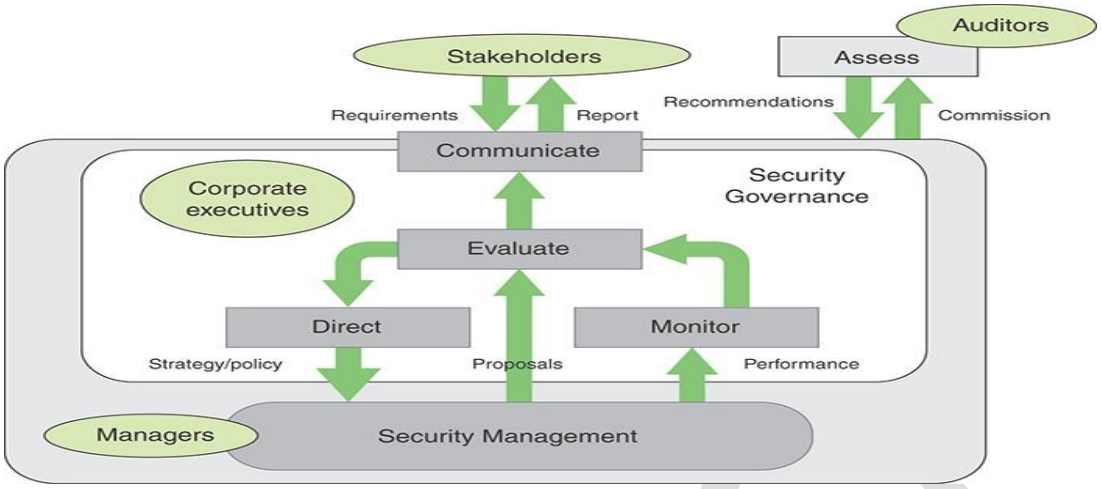
Section	Description

Section	Description
Definition	
Mission, vision, and objectives	Defines the strategy for aligning the information security program with organizational goals and objectives, including the role of individual security projects in enabling specific strategic initiatives.
Priorities	Describes factors that determine strategy and the priorities of objectives.
Success criteria	Defines success criteria for the information security program. Includes risk management, resilience, and protection against adverse business impacts.
Integration	Strategy for integrating the security program with the organization's business and IT strategy.
Threat defense	Describes how the security program will help the organization defend against security threats.
Execution	
Operations plan	An annual plan to achieve agreed objectives that involves agreeing on budgets, resources, tools, policies, and initiatives. This plan (a) can be used for monitoring progress and communicating with stakeholders and (b) ensures that information security is included from the outset in each relevant project.

Section	Description
Monitoring plan	This plan involves planning and maintaining a stakeholder feedback loop, measuring progress against objectives, and ensuring that strategic objectives remain valid and in line with business needs.
Adjustment plan	This plan involves ensuring that strategic objectives remain valid and in line with business needs as well as procedures to communicate the value.
Review	
Review plan	This plan describes procedures and individuals/committees involved in regular review of the information security strategy.

Organizational Structure

The organizational structure to deal with cybersecurity depends, in large part, on the size of the organization, its type (for example, government agency, business, nonprofit), and the organization's degree of dependence on IT. But the essential security governance functions to be performed are in essence the same across organizations. Figure 2.4, which is based on a figure in X.1054, illustrates these basic functions within a broader context.



The basic security governance functions are as follows:

- **Direct:** Guiding security management from the point of view of enterprise strategies and risk management. This function involves developing an information security policy.
- **Monitor:** Monitoring the performance of security management with measurable indicators.
- **Evaluate:** Assessing and verifying the results of security performance monitoring in order to ensure that objectives are met and to determine future changes to the ISMS and its management.
- **Communicate:** Reporting enterprise security status to stakeholders and evaluating stakeholder requirements.

FIGURE 2.6 Security Governance Roles and Responsibilities Example

Security Governance Approach

Security Governance Framework The definition, monitoring, and maintenance of a security governance framework entails a number of tasks:

- Appoint a single executive to be ultimately responsible for security governance, whose duties including implementing the framework and developing and monitoring an information security strategy and security assurance program.

- Decide and communicate to top executives the objectives of the security governance framework, including ensuring alignment with overall organization policies and goals, enhancing business value, and adequately managing risk.

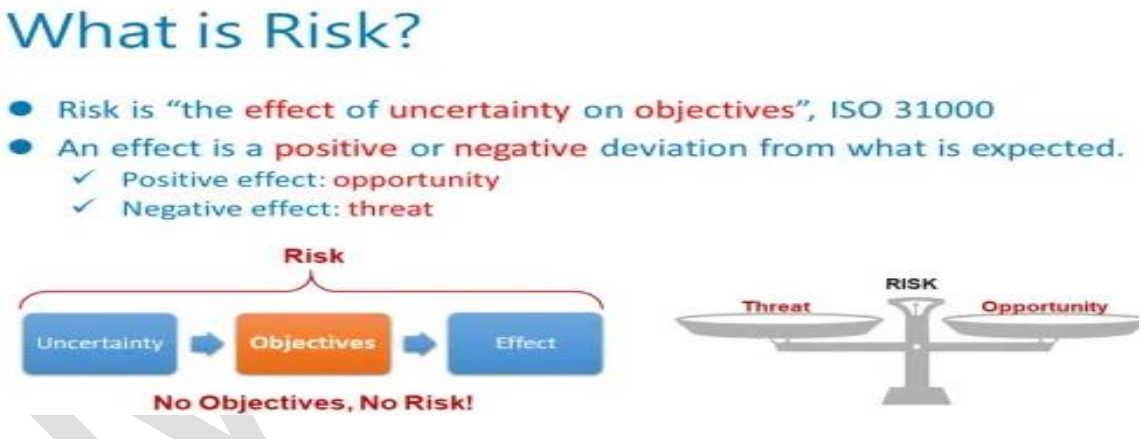
- Ensure integration of the security architecture with the enterprise architecture.

- Include a process that enables the governing body to evaluate the operation of the information security strategy to ensure that it aligns with business needs the organization's current risk appetite.

- Regularly review the organization's risk appetite to ensure that it is appropriate for the current environment in which the organization operates.

Information Risk Management

Information Security *is a discipline of protecting information* ***assets*** *from* ***threats*** *through* ***safeguards*** *to achieve the* ***objectives*** *of confidentiality, integrity, and availability (Tier 3), or CIA for short, support business (Tier 2), and create and deliver values (Tier 1).*



- Management implies someone proactively, deliberately, explicitly and systematically identifying, assessing, evaluating and dealing with risks on an ongoing basis (coping with any changes), along with related governance aspects such as direction, control, authorization and resourcing of the process, risk treatments etc.;

- Risk, in this context, is the possibility, the potential occurrence of events or incidents that might materially harm the organisation's interests or interfere with the realisation of business objectives;

- Information is the valuable meaning, knowledge and insight deriving from raw data such as the content of computer files, paperwork, conversations, expertise, intellectual property, art, concepts and so forth.

Figure 3.1 illustrates in general terms a universally accepted method for determining the level of risk.

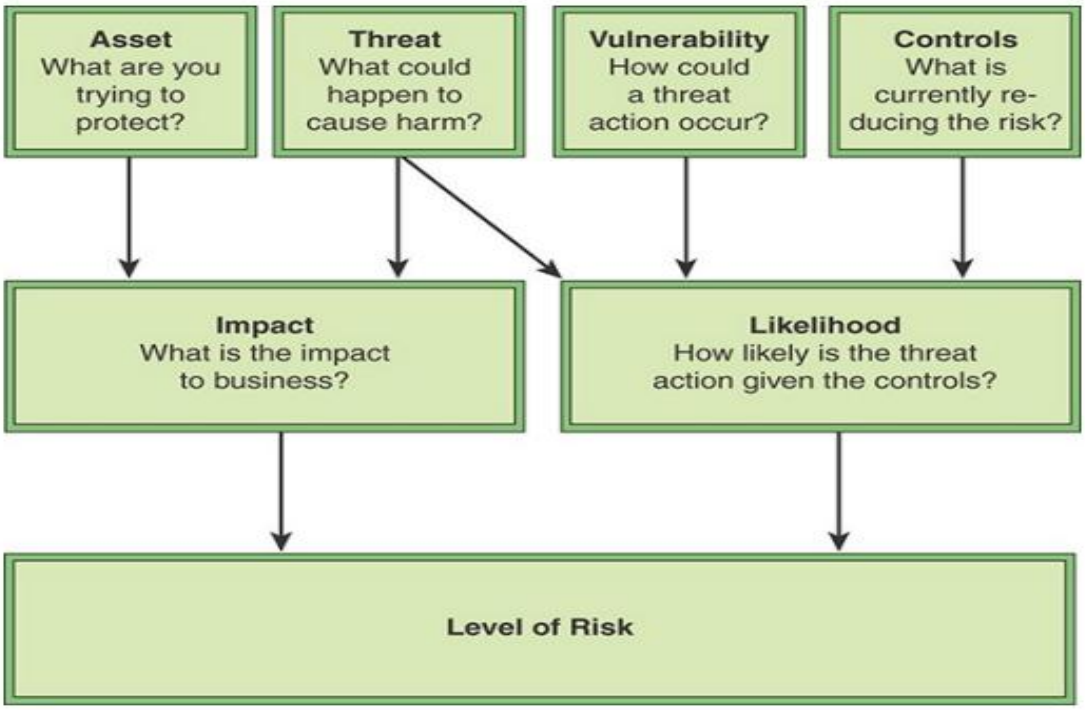


FIGURE 3.1 Determining Information Security Risk

Two main threads, impact and likelihood, should be pursued in parallel. An organization pursues the following tasks related to these threads:

- **Impact:** Consider these two elements in determining impact:
  - **Assets:** Develop an inventory of the organization's assets, which includes an itemization of the assets and an assigned value for each asset. These include intangible assets such as reputation and goodwill, as well as tangible assets, such as databases, equipment, business plans, and personnel.
  - **Threat:** For each asset, determine the possible threats that could reduce the value of that asset.

  Then, for each asset, determine the impact to the business, in terms of cost or lost value, of a threat action occurring.
- **Likelihood:** Consider the three elements in determining likelihood:
  - **Threat:** For each asset, determine which threats are relevant and need to be considered.
  - **Vulnerability:** For each threat to an asset, determine the level of vulnerability to the threat. That is, determine specifically for an asset how a threat action could be achieved.
  - **Controls:** Determine what security controls are currently in place to reduce the risk.

FIGURE 3.2 Risk Management Life Cycle

The steps are as follows:

1. Assess risk based on assets, threats, vulnerabilities, and existing controls. From these inputs, determine impact and likelihood and then the level of risk.
2. Identify potential security controls to reduce risk and prioritize the use of these controls.
3. Allocate resources, roles, and responsibilities and implement controls.
4. Monitor and evaluate risk treatment effectiveness.

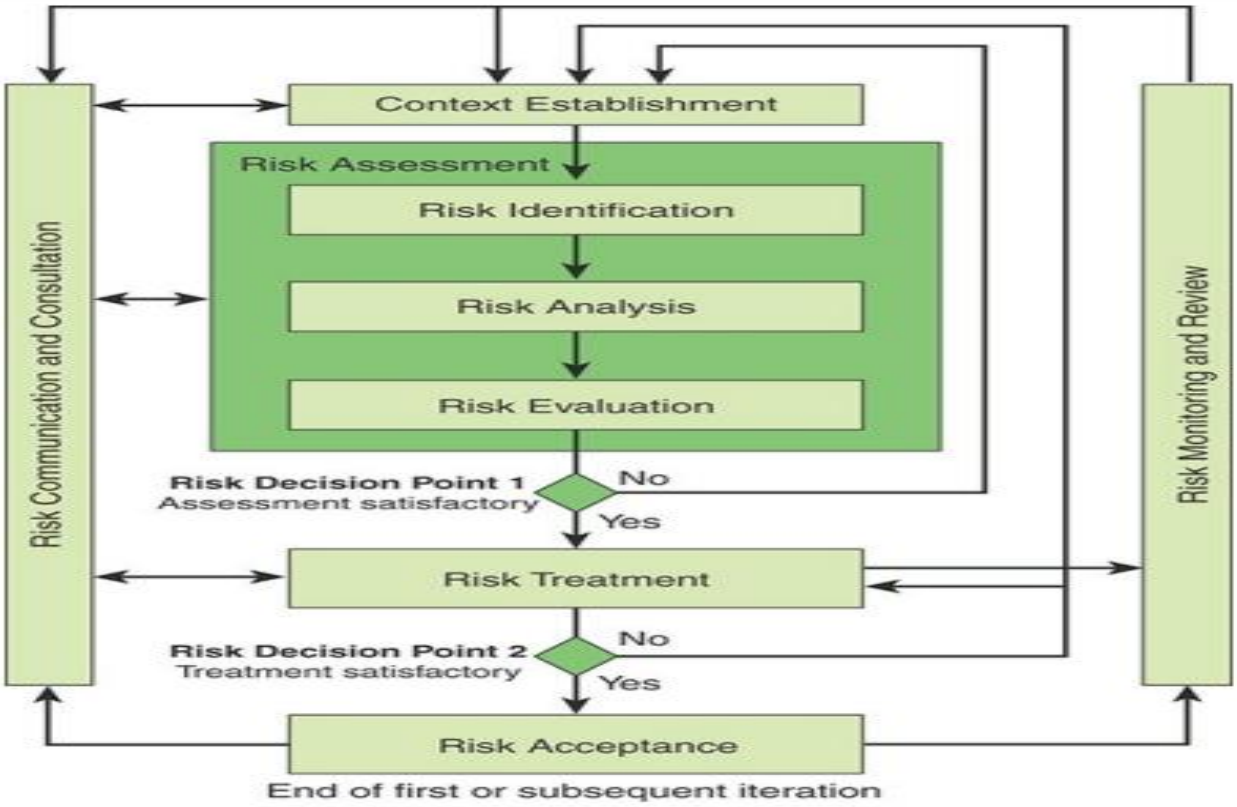


FIGURE 3.3 ISO 27005 Risk Management Process

TABLE 3.2 ISO 27005 Risk Management Context Establishment

Category	Consideration or Criteria
Purpose of risk management	▪ Legal compliance and evidence of due diligence ▪ Preparation of a business continuity plan ▪ Preparation of an incident response plan ▪ Description of the information security requirements for a product, a service or a mechanism
Risk evaluation criteria	▪ The strategic value of the business information proces ▪ The criticality of the information assets involved ▪ Legal and regulatory requirements and contractual obligations ▪ Operational and business importance of availability, confidentiality, and integrity ▪ Stakeholder expectations and perceptions and negative consequences for goodwill and reputation
Impact criteria	▪ Level of classification of the impacted information asset ▪ Breaches of information security (for example, loss of confidentiality, integrity, and availability)

- **Risk assessment:** ISO 27001 defines risk assessment as consisting of three activities:
  - **Risk identification:** Involves the identification of risk sources, events their causes, and their potential consequences. It involves historical data, theoretical analysis, informed and expert opinions, and stakeholders' needs.
  - **Risk analysis:** Provides the basis for risk evaluation and decisions about risk treatment. Risk analysis includes risk estimation.
  - **Risk evaluation:** Assists in the decision about risk treatment by comparing the results of risk analysis with risk criteria to determine whether the risk and/or its magnitude are acceptable or tolerable.
- **Risk treatment:** Involves the following:
  - Avoiding the risk by deciding not to start or continue with an activity that gives rise to the risk
  - Taking or increasing risk in order to pursue an opportunity
  - Removing the risk source
  - Changing the likelihood

- **Risk communication and consultation:** Encompasses the continual and iterative processes that an organization conducts to provide, share, or obtain information and to engage in dialogue with stakeholders regarding the management of risk.
- **Risk monitoring and review:** Includes ongoing monitoring and review of all risk information obtained from the risk management activities.

3.3 Threat Identification

Threat identification is the process of identifying threat sources with the potential to harm system assets. Threat sources are categorized into three areas:

- **Environmental:** Examples include floods, earthquakes, tornadoes, landslides, avalanches, electrical storms, and power failure.
- **Business resources:** Examples include equipment failure, supply chain disruption, and unintentional harm caused by employees.
- **Hostile actors:** Examples include hackers, hacktivists, insider threats, criminals, and nation-state actors.

Both environmental and business resource threats must be recognized and addressed, but the bulk of the effort of threat identification—and indeed of risk assessment and risk management—involves dealing with threats from hostile actors. That is the focus of this section.

Threat	Example
Compromises to intellectual property	Piracy, copyright infringement
Espionage or trespass	Unauthorized access and/or data collection
Forces of nature	Fire, flood, earthquake, lightning
Human error or failure	Accidents, employee mistakes, failure to follow policy
Information extortion	Blackmail of information disclosure
Missing, inadequate, or incomplete controls	Software controls, physical security
Missing, inadequate, or incomplete organizational policy or planning	Training issues, privacy, lack of effective policy
Quality of service deviations from service providers	Power and WAN quality of service issues
Sabotage or vandalism	Destruction of systems or information
Software attacks	Viruses, worms, macros, denial of service
Technical hardware failures or errors	Equipment failure
Technical software failures or errors	Bugs, code problems, unknown loopholes
Technological obsolescence	Antiquated or outdated technologies
Theft	Illegal confiscation of property

Asset Identification

- identification is the identification of the assets, threats, existing controls, vulnerabilities, and impacts relevant to the organization and that serve as inputs
- A first step in risk assessment is to document and determine values for the organization's assets. An asset is anything of value to the business that requires protection, including hardware, software, information, and business assets.
- Many assets of various types can be identified, and the challenge is to develop a uniform way of documenting the assets, the security implications of each, and the costs associated with security incidents related to each.
- Asset valuation relates directly to business needs. Accordingly, the input for asset valuation needs to be provided by owners and custodians of assets, not by members of the risk assessment team.

Hardware Assets

- Hardware assets include servers, workstations, laptops, mobile devices, removable media, networking and telecommunications equipment, and peripheral equipment. Key concerns are loss of a device, through theft or damage, and lack of availability of the device for an extended period.
- Another concern is device malfunction, due to deliberate malfunction or other causes. Asset valuation needs to take into account the replacement cost of the hardware, disruption losses, and recovery expenses.

Software Assets

Software assets include applications, operating systems and other system software, virtual machine and container virtualization software, software for software-defined networking (SDN) and network function virtualization (NFV), database management systems, file systems, and client and server software. Availability is a key consideration here, and asset valuation must take account of disruption losses and recovery expenses.

Information Assets

Information assets comprise the information stored in databases and file systems, both on-premises and remotely in the cloud.

As an example, ITU-T X.1055 lists the following as types of information assets in a telecommunications or network environment:

- Routing information
- Subscriber information
- Blacklist information
- Registered service information
- Operational information
- Trouble information
- Configuration information
- Customer information
- Billing information
- Customer calling patterns
- Customer geographic locations
- Traffic statistical information
- Contracts and agreements
- System documentation
- Research information

- User manuals
- Training materials
- Operational or support procedures
- Business continuity plans
- Emergency plan fallback arrangements
- Audit trails and archived information

Asset valuation needs to take into account the impact of threats to confidentiality, privacy, integrity, and authenticity. As an example of a

Business Assets

The business assets category includes organization assets that don't fit into the other categories, including human resources, business processes, and physical plant. This category also includes intangible assets, such as organization control, know-how, reputation, and image of the organization.

Asset Register

In order to effectively protect assets, an organization needs to provide a systematic method of documenting assets and their security implications. This is done in an asset register that documents important security-related information for each asset. Examples of items that may be included for each asset are as follows:

- **Asset name/description:** This information uniquely identifies an asset.
- **Asset type:** This denotes the type of asset it is, such as physical/infrastructure assets, software, information, service, or human resource.
- **Asset class:** For purposes of risk assessment, an organization should group assets into classes so risks are measured against classes of assets rather than against individual assets. Asset class examples include desktops/workstations, servers, Payment Card Industry (PCI) devices, restricted/sensitive file shares, and restricted printers.
- **Information assets:** An information asset defines specifically what kind of information is processed, transmitted, or stored by the asset (for example, customer personally identifiable information [PII], PCI data). This item does not apply to all assets.
- **Asset owner:** An organization should define the department/company function that owns an asset and is responsible for risk associated with the

TABLE 3.4 Example Asset Register

Asset Name/Description	Asset Classification	Disaster Recovery Priority	Description	Exposure Level
Personnel	High	1	Employees	Medium
Client PII	High	1	Personally identifiable information	Low
Production web server	Medium	1	Company primary website (no sensitive data)	High

3.5 Vulnerability Identification

Vulnerability identification is the process of identifying vulnerabilities that can be exploited by threats to cause harm to assets. A vulnerability is a weakness or a flaw in a system's security procedures, design, implementation, or internal controls that could be accidentally triggered or intentionally exploited when a threat is manifested.

The following sections develop categories of vulnerabilities, discuss approaches to identifying and documenting vulnerabilities, and discuss the use of the National Vulnerability Database.

Vulnerability Categories

Vulnerabilities occur in the following areas:

- **Technical vulnerabilities:** Flaws in the design, implementation, and/or configuration of software and/or hardware components, including application software, system software, communications software, computing equipment, communications equipment, and embedded devices.
- **Human-caused vulnerabilities:** Key person dependencies, gaps in awareness and training, gaps in discipline, and improper termination of access.
- **Physical and environmental vulnerabilities:** Insufficient physical access controls, poor siting of equipment, inadequate temperature/humidity controls, and inadequately conditioned electrical power.
- **Operational vulnerabilities:** Lack of change management, inadequate separation of duties, lack of control over software installation, lack of control over media handling and storage, lack of control over system communications, inadequate access control or weaknesses in access control procedures, inadequate recording and/or review of system activity records, inadequate control over encryption keys, inadequate reporting, handling and/or resolution of security incidents, and inadequate monitoring and evaluation of the effectiveness of security controls.
- **Business continuity and compliance vulnerabilities:** Misplaced, missing, or inadequate processes for appropriate management of business risks; inadequate business continuity/contingency planning; and inadequate monitoring and evaluation for compliance with governing policies and

Risk assessment

A risk assessment is a process to identify potential hazards and analyze what could happen if a hazard occurs. A business impact analysis (BIA) is the process for determining the potential impacts resulting from the interruption of time sensitive or critical business processes.

Quantitative Versus Qualitative Risk Assessment

Two factors of risk assessment can be treated either quantitatively or qualitatively: impact and likelihood.

- For impact, if it seems feasible to assign a specific monetary cost to each of the impact areas, then the overall impact can be expressed as a monetary cost. Otherwise, qualitative terms, such as low, moderate, and high, are used.
- Similarly, the likelihood of a security incident may be determined quantitatively or qualitatively. The quantitative version of likelihood is simply a probability value, and again the qualitative likelihood can be expressed in such categories as low, medium, and high.

	Quantitative	Qualitative
Benefits	▪ Risks are prioritized by financial impact; assets are prioritized by financial values. ▪ Results facilitate management of risk by return on security investment. ▪ Results can be expressed in management-specific terminology (for example, monetary values and probability expressed as a specific percentage). ▪ Accuracy tends to increase over time as the organization builds historic record of data while gaining experience.	▪ It enables visibility and understanding of risk ranking. ▪ It is easier to reach consensus. ▪ It is not necessary to quantify threat frequency. ▪ It is not necessary to determine financial values of assets. ▪ It is easier to involve people who are not experts on security or computers.
Drawbacks	▪ Impact values assigned to risks are based on subjective opinions of participants. ▪ The process to reach credible results and consensus is very time-consuming. ▪ Calculations can be complex and time-consuming. ▪ Results are presented in monetary terms only, and they may be difficult for nontechnical people to interpret. ▪ The process requires expertise, so participants cannot be easily coached through it.	▪ There is insufficient differentiation between important risks. ▪ It is difficult to justify investing in control implementation because there is no basis for a cost/benefit analysis. ▪ Results are dependent upon the quality of the risk management team that is created.

An organization needs some clearly defined categories of impact, threat, and vulnerability. For impact, FIPS 199, Standards for Security Categorization of Federal Information and Information Systems, defines three security categories based on

the potential impact on an organization should certain events occur that jeopardize the IT assets needed by the organization to accomplish its assigned mission, protect its assets, fulfill its legal responsibilities, maintain its day-to-day functions, and protect individuals.

- **Low:** Expected to have a limited adverse effect on organizational operations, organizational assets, or individuals, including the following:
  - Cause a degradation in mission capability to an extent and duration

that the organization is able to perform its primary functions, but the effectiveness of the functions is noticeably reduced

- Result in minor damage to organizational assets
- Result in minor financial loss
- Result in minor harm to individuals

- **Moderate or medium:** Expected to have a serious adverse effect on organizational operations, organizational assets, or individuals, including the following:
  - Cause a significant degradation in mission capability to an extent and duration that the organization is able to perform its primary functions, but the effectiveness of the functions is significantly reduced
  - Result in significant damage to organizational assets
  - Result in significant financial loss
  - Result in significant harm to individuals that does not involve loss of life or serious life-threatening injuries
- **High:** Expected to have a severe or catastrophic adverse effect on organizational operations, organizational assets, or individuals, including the following:
  - Cause a severe degradation in or loss of mission capability to an extent and duration that the organization is not able to perform one or more of its primary functions
  - Result in major damage to organizational assets
  - Result in major financial loss

Security Management Function

What is Security Management?

- Security management covers all aspects of protecting an organization's assets – including computers, people, buildings, and other assets – against risk.

- A security management strategy begins by identifying these assets, developing and implementing policies and procedures for protecting them, and maintaining and maturing these programs over time.

Below, we discuss what security management means to organizations, types of security management, and review some considerations for security management when choosing a cyber security solution.

Purpose of Security Management

- The goal of security management procedures is to provide a foundation for an organization's cybersecurity strategy.
- The information and procedures developed as part of security management processes will be used for data classification, risk management, and threat detection and response.
- These procedures enable an organization to effectively identify potential threats to the organization's assets, classify and categorize assets based on their importance to the organization, and to rate vulnerabilities based on their probability of exploitation and the potential impact to the organization.

Types of Security Management : Security management can come in various different forms. Three common types of security management strategies include information, network, and cyber security management.

1. Information Security Management

- Information security management includes implementing security best practices and standards designed to mitigate threats to data like those found in the ISO/IEC 27000 family of standards.

- Information security management programs should ensure the confidentiality, integrity, and availability of data.

- Many organizations have internal policies for managing access to data, but some industries have external standards and regulations as well.

The following as areas of responsibility:

- ✓ Application information security Infrastructure
- ✓ information security

- ✓ Access management
- ✓ Threat and incident management
- ✓ Risk management
- ✓ Awareness program
- ✓ Metrics
- ✓ Vendor assessments

- For example, healthcare organizations are governed by the Health Insurance Portability and Accessibility Act (HIPAA), and the Payment Card Industry Data Security Standard (PCI DSS) protects payment card information.

2. Network Security Management

- Network security management is a vital component of a network management strategy. The network is the vector by which most cyber attacks reach an organization's systems and its first line of defense against cyber threats.

- Network security management includes deploying network monitoring and defense solutions, implementing network segmentation, and controlling access to the network and the devices connected to it.

3. Cyber security Management

- Cyber security management refers to a more general approach to protecting an organization and its IT assets against cyber threats.

- This form of security management includes protecting all aspects of an organization's IT infrastructure, including the network, cloud infrastructure, mobile devices, Internet of Things (IoT) devices, and applications and APIs.

- NISTIR 7359, **Information Security Guide for Government Executives**, provides a useful summary of the tasks that comprise information security management Although addressed to government executives, NISTIR 7359 discusses the general functional areas of an information security or

cybersecurity program that should be the responsibility of the CISO in any organization.

The key security program areas include the following:

Security planning: Security planning includes strategic security planning, But it also includes more detailed planning for the organization, coordination, and implementation of security. Key actors within the organization, such as department heads and project managers, need to be consulted and brought into the ongoing process of planning.

Capital planning : Capital planning is designed to facilitate and control the expenditure of the organization's funds. Part of the planning process, and part of the CISO's responsibility, is to prioritize potential IT security investments for allocating available funding.

Awareness and training: Awareness and training programs ensure that personnel at all levels of the organization understand their information security responsibilities to properly use and protect the information resources entrusted to them.

Information security governance: The CISO should advise C-level executives and the board concerning the development

System development life cycle: This is the overall process of developing, implementing, and retiring information systems.

Security products and services acquisition: Management supervision of the acquisition of security-related products and services includes considering the costs involved, the underlying security requirements, and the impact on the organizational mission, operations, strategic functions, personnel, and service-provider arrangements.

Risk management

Configuration management: The CISO should employ configuration management to ensure adequate consideration of the potential security impacts due to specific changes to an information system or its surrounding environment.

Incident response: Incident response, which occurs after the detection of a security event, seeks to minimize the damage of the event and facilitate rapid recovery.

Contingency planning: Information system contingency planning involves management policies and procedures designed to maintain or restore business operations, including computer operations, possibly at an alternate location, in the event of emergencies, system failures, or disasters.

Performance measures:

The CISO should ensure that an organizationwide performance measures are defined and used. Performance measures are a key feedback mechanism for an effective information security program.

Support function: The CISO should:

- ✓ Act as a clearing house for security advice, making experts available to business unit managers and project managers, as needed
- ✓ Promote security awareness throughout the organization
- ✓ Develop standard terms and agreements in contracts to ensure that suppliers and other external relationships meet the security standards of the organization
- ✓ Evaluate the security implications of new business initiatives
- ✓ Oversee the risk assessment process
- ✓ Set standards for use of cryptographic algorithms and security protocols

Monitor function: The CISO should monitor trends and developments to be aware of how they may affect the organization's security strategy and implementation, including in the area of business trends, new technical developments, security solutions, standards, legislation, and regulation.

Projects function: The CISO should be responsible for overseeing security-related projects.
External requirements function: The CISO should manage the implications of laws, regulations, and contracts.

security plan

- Security plan is to provide an overview of the security requirements of the system and describe the controls in place or planned for meeting those requirements.
- The system security plan also delineates responsibilities and expected behavior of all individuals who access the system. The system security plan is basically documentation of the structured process of planning adequate, cost-effective security protection for a system.

information system in an organization have a separate plan document with the following elements:

Information system name/identifier: A name or identifier uniquely assigned to each system. Assignment of a unique identifier supports the organization's ability to easily collect information and security metrics specific to the system as well as facilitate complete traceability to all requirements related to system implementation and performance. The identifier should remain the same throughout the life of the system and retained in audit logs related to system use.

Information system owner: The person responsible for managing this asset.

Authorizing individual: The senior management official or executive with the authority to formally assume responsibility for operating an information system at an acceptable level of risk to agency operations, agency assets, or individuals.

Assignment of security responsibility: The individual responsible for the security of the information system.

Security categorization: Using the FIPS 199, Standards for Security Categorization of Federal Information and Information Systems, categories, the acceptable level of

risk (low, moderate, or high) for confidentiality, integrity, and availability (for each distinct element of the system, if necessary).

Information system operational status: Status, such as operational, under development, or undergoing major modification.

Information system type: Type, such as major application or support system.

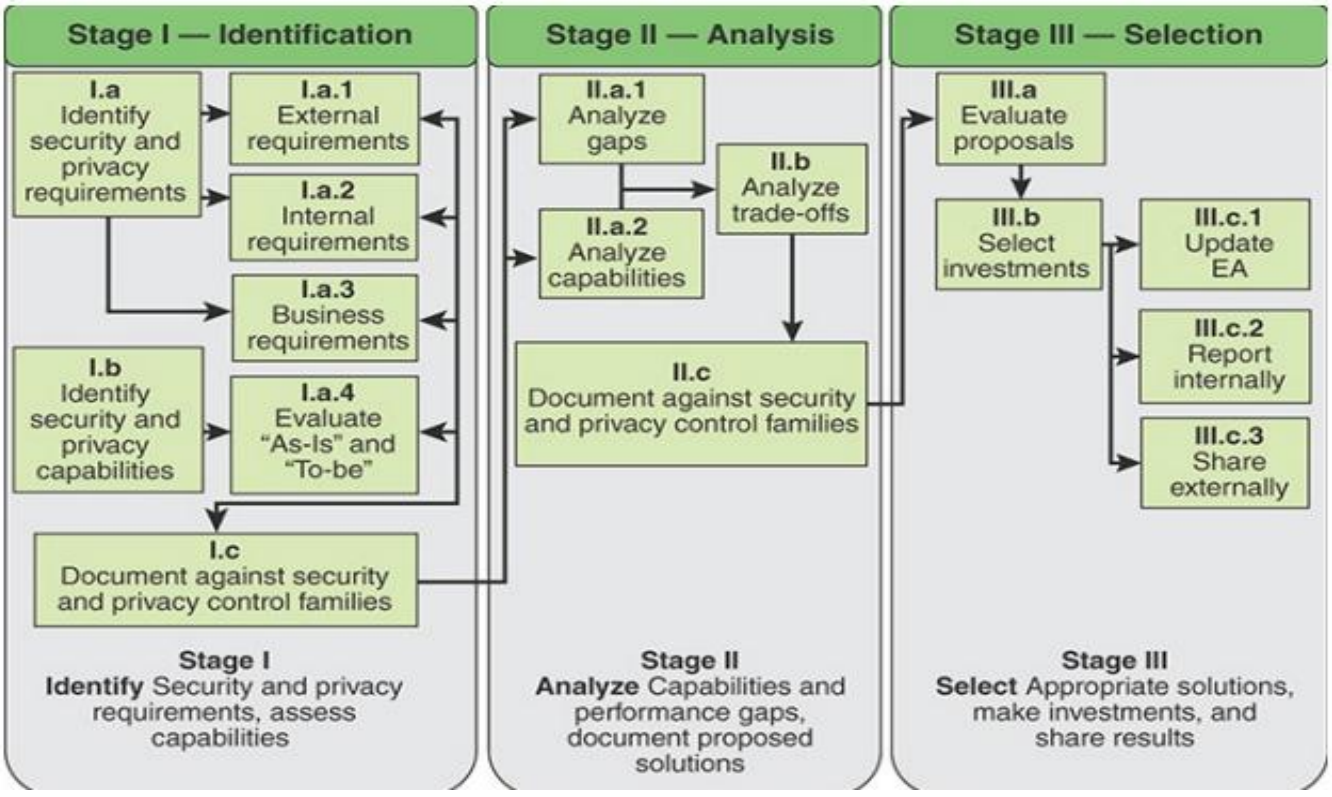
Description/purpose: A brief description (one to three paragraphs) of the function and purpose of the system.

Existing security controls: Description of each control.

Planned security controls: Description of each control plus implementation plan.

Information system security plan completion date: The target date.

Information system security plan approval date: The data plan approved date.



This process involves three steps, each of which has goals, objectives, implementing activities, and output products for formal inclusion in agency enterprise architecture and capital planning processes:

1**. Identify**: Encompasses the research and documentation activities necessary to identify security and privacy requirements in support of the mission objectives so that they can be incorporated into the enterprise architecture.

2. Analyze: Involves an analysis of organization security and privacy requirements and the existing or planned capabilities that support security and privacy.

3. Select: Involves an enterprise evaluation of the solutions proposed in the preceding phase and the selection of major investments.

Step 1 refers to three types of requirements, defined as follows:

External requirements: These are security requirements imposed from outside the organization, such as laws, regulations, and contractual commitments.

Internal requirements: These are security requirements developed as part of the security policy, such as the acceptable degree of risk and confidentiality, integrity, availability, and privacy guidelines.

Business requirements: This refers to requirements other than security requirements that are related to the overall business mission. Examples include finance, accounting, and audit requirements. In general, these requirements refer to the organization's need to discharge business responsibilities.

Capital Planning

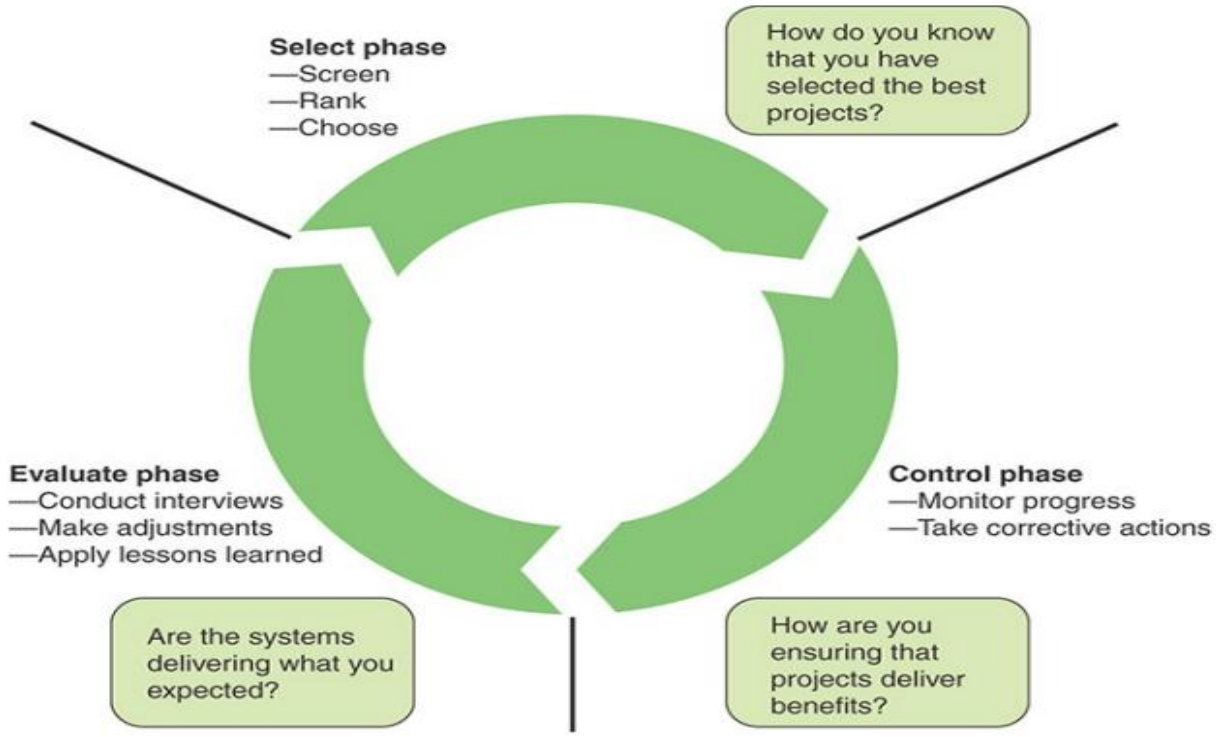


FIGURE 4.2 Capital Planning and Investment Lifecycle

The Select/Control/Evaluate framework defines a cyclical process consisting of three steps for deciding which projects to pursue or which investments to make:

1. Select: Identify and analyze each project's risks and returns before committing significant funds to any project. The organization then selects the IT projects that best support its mission needs. The organization repeats this process each time funds are allocated to projects.

2. Control: Ensure that as projects develops and investment expenditures continue, the project continues to meet mission needs at the expected levels of cost and risk. If the project is not meeting expectations or if problems have arisen, steps must be quickly taken to address the deficiencies. If mission needs have changed, the organization needs to adjust its objectives for the project and appropriately modify expected project outcomes.

3. Evaluate: Compare actual results and expected results after a project was fully implemented. This is done for the following reasons: To assess the project's impact

on mission performance To identify any necessary changes or modifications to the project To revise the investment management process based on lessons learned

Security Policy

Information security policy is an aggregate of directives, rules, and practices that prescribes how an organization manages, protects, and distributes information.

It is helpful to distinguish four types of documents before proceeding:

- **Information security strategic plan**: Relates to the long-term goals for maintaining security for assets.
- **Security plan**: Relates to security controls in place and planned to meet strategic security objectives.
- **Security policy**: Relates to the rules and practices that enforce security.
- **Acceptable use policy**: Relates to how users are allowed to use assets.
- **Security-Related Documents**

Document Type	Description	Primary Audience
Information security strategic plan	A document used to communicate with the organization the organization's long-term goals with respect to information security, the actions needed to achieve those goals, and all the other critical elements developed during the planning exercise.	C-level executives
Security plan	A formal document that provides an overview of the security requirements for the information system and describes the security controls in place or planned for meeting those requirements.	C-level executives, security managers, other

		managers
Security policy	A set of laws, rules, and practices that regulate how an organization manages and protects assets and the rules for distribution of sensitive information. It includes associated responsibilities and the information security principles to be followed by all relevant individuals.	All employees, especially those with some responsibility for an asset or assets
Acceptable use policy	A policy that defines for all parties the ranges of use that are approved for use of information, systems, and services within the organization.	All employees

-

Security Policy Categories

- **Access control policy**: How information is accessed
- **Contingency planning policy**: How availability of data is provided 24/7
- **Data classification policy**: How data are classified
- **Change control policy**: How changes are made to directories or the file server
- **Wireless policy**: How wireless infrastructure devices need to be configured
- **Incident response policy**: How incidents are reported and investigated
- **Termination of access policy**: How employee access to organization assets is handled during termination
- **Backup policy**: How data is backed up
- **Virus policy**: How virus infections need to be dealt with
- **Retention policy**: How data can be stored
- **Physical access policy**: How access to the physical area is obtained
- **Security awareness policy**: How security awareness is carried out
- **Audit trail policy**: How audit trails are analyzed Firewall policy: How firewalls are named, configured, and so on
- **Network security policy**: How network systems are secured
- **Encryption policy**: How data are encrypted, the encryption method used, and so on
- **BYOD policy**: What devices an employee may use both on premises and off to access organization assets

- **Cloud computing policy**: Security aspects of using cloud computing resources and service

Security Policy Document Content

Whether a single document or a set of documents, each security policy document should include the following sections:

- **Overview**: Background information on what issue the policy addresses
- **Purpose**: Why the policy was created
- **Scope**: What areas the policy covers
- **Targeted audience**: To whom the policy is
- **applicable Policy**: A complete but concise description of the policy
- **Noncompliance**: Consequences for violating the policy
- **Definitions**: Technical terms used in the document
- **Version:** Version number to keep track of the changes made to the document

Management Guidelines for Security Policies The SGP provides a useful set of guidelines for the creation, content, and use of security policy documents, which can be categorized as follows

Responsibilities: Identify the following:

- Those responsible for ratifying policy document (for example, the board)
- Responsibilities of all relevant individuals to comply with the policy
- Individuals responsible for protecting specific assets
- That all individuals must confirm the understanding of, acceptance of, and compliance with relevant policies and understand that disciplinary action will follow policy violation

Principles: Specify the following:

- All relevant assets to be identified and classified by value/importance
- All assets protected with respect to CIA (confidentiality, integrity, and availability) and other security requirements
- All laws, regulations, and standards complied with

Actions: Specify the following:

- That all individuals are made aware of the security policy and their responsibilities
- That all assets are subject to risk assessment periodically and before a major change
- That all breaches are reported in a systematic fashion
- That auditing occurs periodically and as needed
- That policy documents are reviewed regularly and as needed

Acceptable use: Policies that include the following:

- Documentation of what behaviors are required, acceptable, and prohibited with Respect various assets
- Responsibility for establishing, approving, and monitoring acceptable use policies.

Acceptable Use Policy

- An acceptable use policy (AUP) is a type of security policy targeted at all employees who have access to one or more organization assets.
- It defines what behaviors are acceptable and what behaviors are not acceptable. The policy should be clear and concise, and it should be a condition of employment for each employee to sign a form indicating that he or she has read and understood the policy and agrees to abide by its conditions.

suggests the following process for developing an AUP:

1. **Conduct a risk assessment to identify areas of concern**.:As part of the risk assessment process, identify the elements that need to go into an AUP.

2. **Create the policy**: The policy should be tailored to the specific risks identified, including liability costs. For example, the organization is exposed to liability if customer data is exposed. If the failure to protect the data is due to an employee's action or inaction, and if this behavior violates the AUP, and if this policy is clear and enforced, then this may mitigate the liability of the organization.

3. **Distribute the A**UP: This includes educating employees on why an AUP is necessary.

4. **Monitor complianc** .:A procedure is needed to monitor and report on AUP compliance.

5. **Enforce the po**licy: The AUP must be enforced consistently and fairly when it is breached.

The document is the policy section, which covers the following areas:

General use and ownership: Key points in this section include:

- Employees must ensure that proprietary information is protected.
- Access to sensitive information is allowed only to the extent authorized and necessary to fulfill duties.
- Employees must exercise good judgment regarding the reasonableness of personal use.

Security and proprietary information: Key points in this section include:

- Mobile devices must comply with the company's BYOD policies.
- System- and user-level passwords must comply with the company's password policy.
- Employees must use extreme caution when opening email attachments.

Unacceptable use—system and network activities: Key points in this section include:

- Unauthorized copying of copyrighted material
- The prohibition against accessing data, a server, or an account for any purpose other than conducting company business, even with authorized access
- Revealing your account password to others or allowing use of your account by others
- Making statements about warranty unless it is a part of normal job duties

- Circumventing user authentication or security of any host, network, or account
- Providing information about, or lists of, company employees to outside parties.

Unacceptable use—email and communication activities: Key points in this section include:

- Any form of harassment
- Any form of spamming
- Unauthorized use, or forging, of email header information

Unacceptable use—blogging and social media: Key points in this section include:

- Blogging is acceptable, provided that it is done in a professional and responsible manner, does not otherwise violate company policy, is not detrimental to company's best interests, and does not interfere with an employee's regular work duties.
- Any blogging that may harm or tarnish the image, reputation, and/or goodwill of company and/or any of its employees is prohibited.
- Employees may not attribute personal statements, opinions, or beliefs to the company.

Security Management Best Practices

The SGP breaks down the best practices in the security management category into two areas and five topics and provides detailed checklists for each topic. The areas and topics are as follows:

Security policy management: Discusses a specialist information security function, led by a sufficiently senior manager (e.g., a CISO), that is assigned adequate authority and resources to run information securityrelated projects; promote information security throughout the organization; and manage the implications of relevant laws, regulations and contracts.

- **Information security policy**: Documents the governing body's direction on and commitment to information security and communicate it to all relevant individuals.
- **Acceptable use policies**: Lists recommended actions for establishing AUPs, which define the organization's rules on how each individual (for example, an employee, a contractor) may use information and systems, including software, computer equipment, and connectivity.

Information security management: Provides guidance for developing a comprehensive, approved information security policy (including supporting policies, standards, and procedures) and communicating it to all individuals who have access to the organization's information and systems.

- **Information security function**: Ensures that good practice in information security is applied effectively and consistently throughout the organization.
- **Information security projects**: Lists recommended actions for ensuring that all information security projects apply common project management practices, meet security requirements, and are aligned with the organization's business objectives.
- **Legal and regulatory compliance**: Describes a process that should be established to identify and interpret the information security implications of relevant laws and regulations.

Information security management:

- ✓ Provides guidance for developing a comprehensive, approved information security policy.
  - Information security function
  - Information security projects
  - Legal and regulatory compliance

5. SECURITY MODEL

- ✓ A model describes the system
  - e.g., a high level specification or an abstract machine description of what the system does .
- ✓ A security policy
  - defines the security requirements for a given system
  - Verification techniques that can be used to show that a policy is satisfied by a system

✓ System Model + Security Policy = Security Model

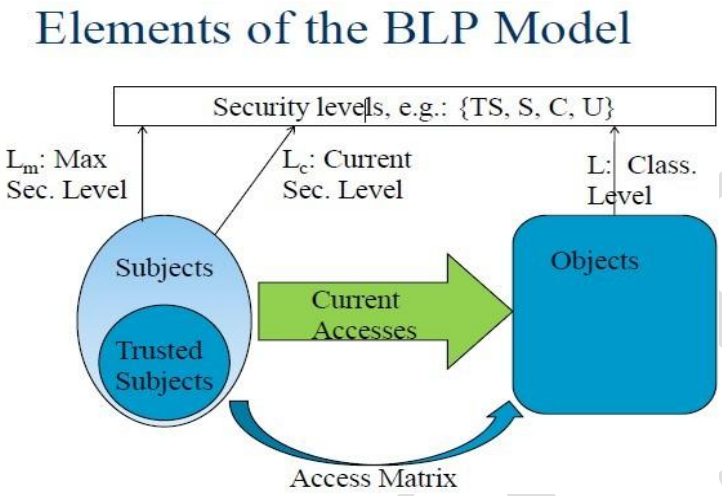
BELL-LAPADULA MODEL

- Bell-LaPadula model is a security method created for the US government to preserve the confidentiality of information

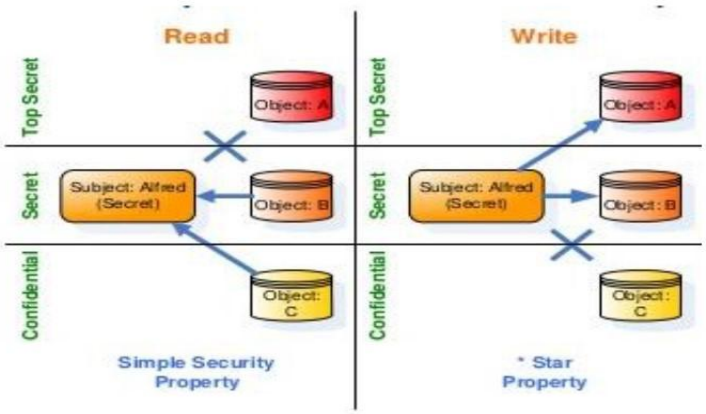
The BLP Security Model

- A computer system is modeled as a state-transition system
  - There is a set of subjects; some are designated as trusted.
  - Each state has objects, an access matrix, and the current access information.
  - There are state transition rules describing how a system can go from one state to another
  - Each subject s has a maximal security level Lm(s), and a current security level Lc(s)
  - Each object has a classification level
- **Example of security levels**
  - Top Secret
  - Secret
  - Confidential
  - Unclassified

Bell-LaPadula is a form of multilevel security



The Bell-LaPadula model is defined by the following properties



- If a system is initially in a secure state, and every transition of the system satisfies the simple security condition, and the * property, then every state of the system is secure

- **Example**

security level	*subject*	*object*
Top Secret	Tamara	Personnel Files
Secret	Samuel	E-Mail Files
Confidential	Claire	Activity Logs
Unclassified	Alice	Telephone Lists

- Tamara can read all files
- Claire cannot read Personnel or E-Mail Files
- Alice can only read Telephone Lists

Bell-LaPadula model has two major limitations:

- It provides confidentiality only. (no integrity, authentication ,etc.)
- It provides no method for management of classifications

BIBA MODEL

- The Biba integrity model was published in 1977 at the Mitre Corporation, one year after the Bell La-Padula model was published.
- The Biba model supports the access control of both subjects and objects.
  - Subjects are the active elements in the system that can access information (processes acting on behalf of the users).
  - Objects are the passive system elements for which access can be requested (files, programs, etc.).

Access Modes

- The Biba model consists of the following access modes:

✓ **Modify**
  - This mode is similar to the write mode in other models.

✓ **Observe**
  - The observe right allows a subject to read an object.

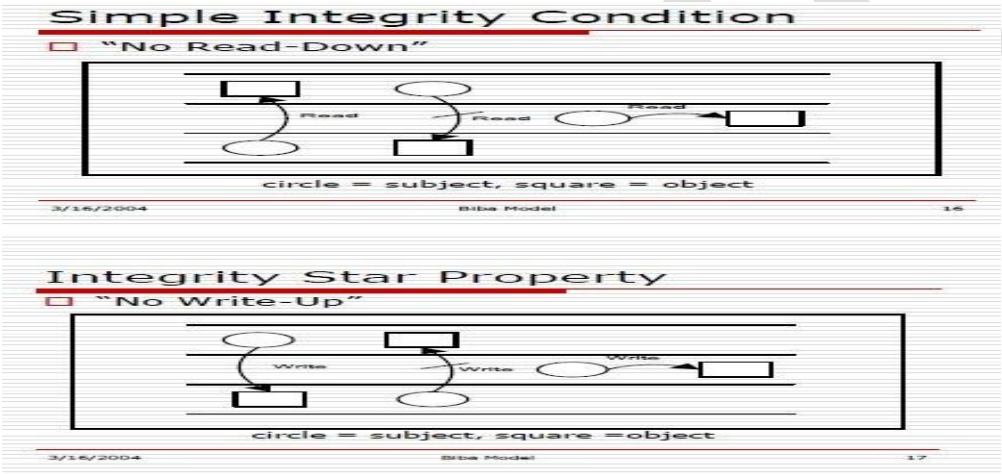
✓ **Invoke**
  - The invoke right allows a subject to communicate with another subject.

✓ **Execute**
  - The execute right allows a subject to execute an object

Biba Policies

- The model supports both mandatory and discretionary policies
  - **The Mandatory Policies**:
    - Strict Integrity Policy
    - Low-Watermark Policy for Subjects
    - Low-Watermark Policy for Objects
  - **The Discretionary Policies**:
    - Access Control Lists
    - Object Hierarchy
    - Ring
- **Strict Integrity Policy** The Strict Integrity Policy is the first part of the Biba model. The policy consists of:

- **1. Simple Integrity Condition**: s ∈ S can observe o∈ O if and only if i(s) ≤ i(o) (—no read-down‖). That means high integrity subject cannot read low integrity objects.
- **2. Integrity Star Property**: s ∈ S can modify o∈ O if and only if i(o) ≤ i(s) (—no write-up‖). Subject cannot move low integrity data to high integrity environment.
- **3. Invocation Property**: s_1 ∈ S can invoke s_2 ∈ S if and only if i(s_2) ≤ i(s_1). That means a subject at one integrity level is prohibited from invoking or calling up a subject at higher level of integrity.



Low-Watermark Policy for Subjects

– The low-watermark policy for subjects is a relaxed no read-down

Low-Watermark Policy for Objects

– The low-watermark policy for objects is a relaxed no write-down.

Ring Policy

- Any subject can observe any object, regardless of integrity levels

Advantages:

- The Biba model is it simple and easy to implement.
- The Biba model provides a number of different policies that can be selected based on need.

Disadvantages:

- The model does nothing to enforce confidentiality. The Biba model doesn‘t support the granting and revocation of authorization.
- To use this model all computers in the system must support the labeling of integrity for both subjects and objects

5.3 CHINESE WALL MODEL

- Chinese Wall Model which addresses a conflicts of interest problem.
- Strictly speaking, this is not an integrity policy, but an access control confidentiality policy.
- The main idea is that you are able to access any information you want from any company but once you access that information, you are no longer allowed to access information from another company within that class of companies.
- The security policy builds on three levels of abstraction.
    - **Objects** such as files. Objects contain information about only one company.
    - **Company groups** collect all objects concerning a particular company.
    - **Conflict classes** cluster the groups of objects for competing companies. That means contain the CDs of companies in competition

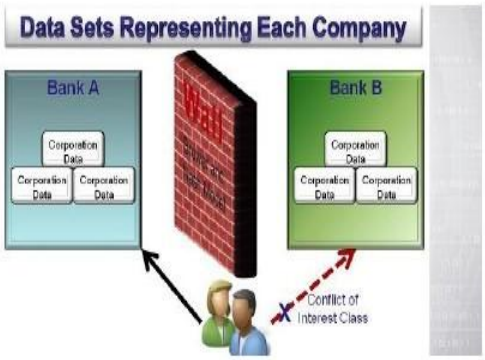
- For example, consider the following conflict classes:

{ Dialog, Mobitel, Airtel }
{ Central Bank, HNB, HSBC }
{ Microsoft }

    - For example, if you access a file from Dialog, you subsequently will be blocked from accessing any files from Mobitel or Airtel
- Chinese wall model is mainly focus on **conflicts of interest (COI) COI- conflict classes contains CDs of competitive companies**

- Principle: Users should not access the confidential information of both a client organization and one or more of its competitors.
- How it works
    - Users have no —wall initially.
    - Once any given file is accessed, files with competitor information become inaccessible.
    - Unlike other models, access control rules change with user behavior

- A "wall" is defined by a set of rules that ensures no subject from one side of the wall can access objects on the other side of the wall.



- **Simple security rule:**
  - Access is only granted if the object requested:
    - is in the same company dataset as an object already accessed by that subject, i.e. within the Wall, or
    - belongs to an entirely different conflict of interest class

- ***-property rule**:
  - Write access is only permitted if
    - access is permitted by the simple security rule, and
    - no object can be read which is in a different company dataset to the one for which write access

UNIT II SECURITY CONTROLS

People Management -Human Resource Security-Security Awareness and Education-Information Management- Information Classification and handling-Privacy-Documents and Record Management-Physical Asset Management-Office Equipment-Industrial Control Systems-Mobile Device Security- System Development-Incorporating Security into SDLC - Disaster management and Incident response planning.

People Management

- The Information Security Forum's (ISF's) Standard of Good Practice for Information Security (SGP) uses the term people management to refer to all aspects of security related to the behavior of employees and others who have access to the organization's information and systems.
- As many security experts have pointed out, superb technical solutions for ensuring security are bound to fail if employees do not understand their security responsibilities and are trained and motivated to fulfill those responsibilities.

Human Resource Security

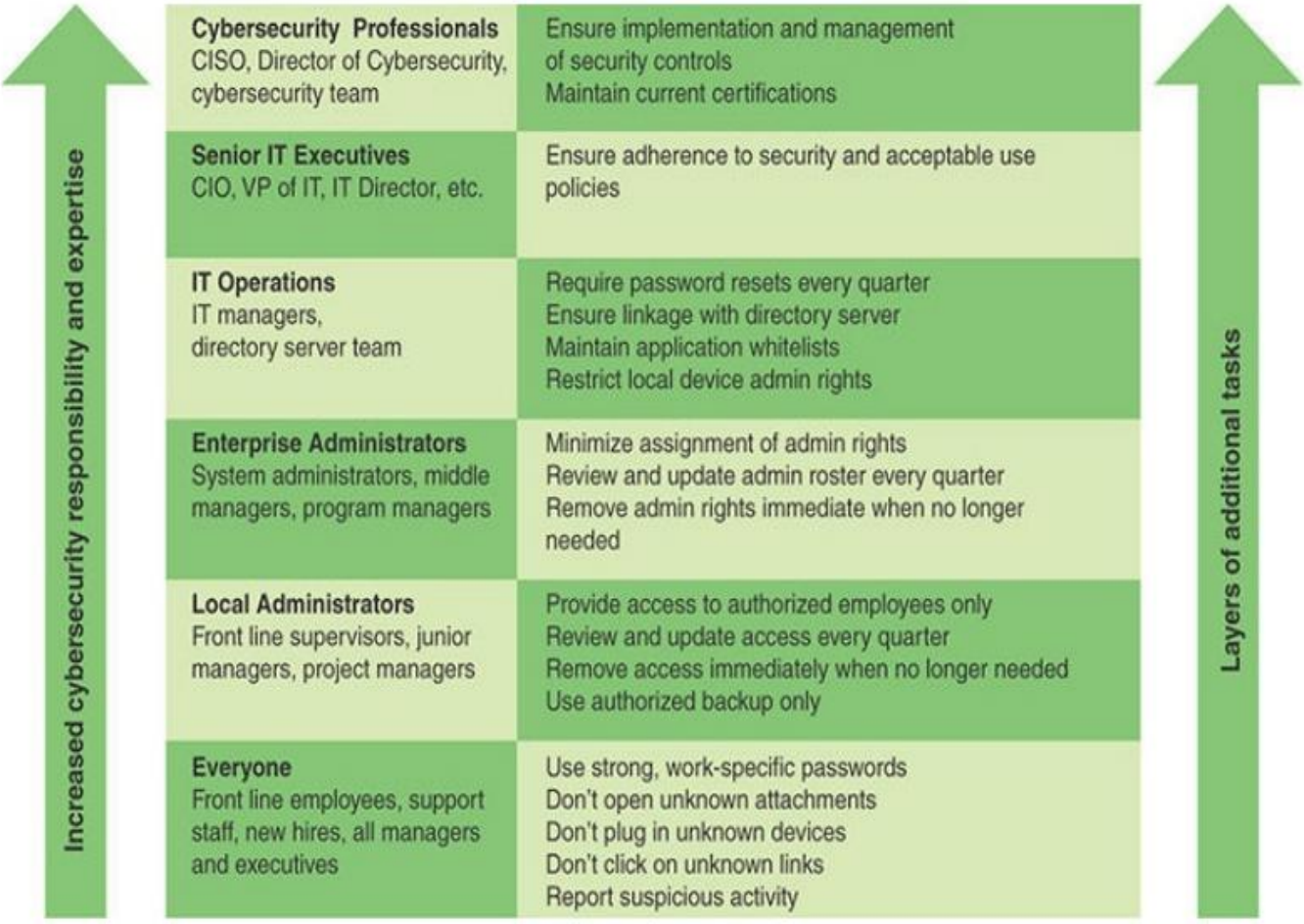
- Human Resource Management (HRM) is an operation in companies designed to maximize employee performance in order to meet the employer's strategic goals and objectives. More precisely, HRM focuses on management of people within companies, emphasizing on policies and systems.
- Sound security practice dictates that information security requirements be embedded into each stage of the employment life cycle, specifying security-related actions required during the induction of each individual, the employee's ongoing management, and termination of his or her employment.
  - Hiring new employees
  - Training employees
  - Monitoring employee behavior
  - Handling employee departure/termination

Especially important is the management of personnel with privileged user access to information and IT assets.

General guidelines for checking applicants include the following:

1) Security in the Hiring Process

- ✓ ISO 27002, Code of Practice for Information Security Controls, lists the following security objective of the hiring process: to ensure that employees and contractors understand their responsibilities and are suitable for the roles for which they are considered.
  - Background Checks and Screening
  - Employment Agreements
  - Job Descriptions



Security-Related Tasks by Job Description

2) During Employment

- ✓ ISO 27002 lists the following security objective with respect to current employees: to ensure that employees and contractors are aware of and fulfill their information security responsibilities.
  - Least privilege
  - Separation of duties
  - Limited reliance on key employees
  - Dual operator policy

3) Termination of Employment

- ✓ ISO 27002 lists the following security objective with respect to termination of employment: to protect the organization's interests as part of the process of changing or terminating employment.
  - Removing the person's name from all lists of authorized access to applications and systems
  - For IT personnel, ensuring that no rogue admin accounts were created
  - Explicitly informing guards that the ex-employee is not allowed into the building without special authorization by named employees
  - Removing all personal access codes
  - Recovering all assets, including employee ID, disks, documents, and equipment
  - Notifying, by memo or email, appropriate departments so that they are aware of the change in employment status

Security Awareness and Education

- A critical element of an information security program is the security awareness and training program.
- It is the means for disseminating security information to all employees, including IT staff, IT security staff, and management, as well as IT users and other employees.
- A workforce that has a high level of security awareness and appropriate security training for each individual's role is as important as, if not more important than, any other security countermeasure or control.
  - ✓ Two key National Institute of Standards and Technology (NIST) publications, SP 800-16.

- ✓ A Role-Based Model for Federal Information Technology/Cyber security Training. SP 800-50, Building an Information Technology Security Awareness and Training Program.

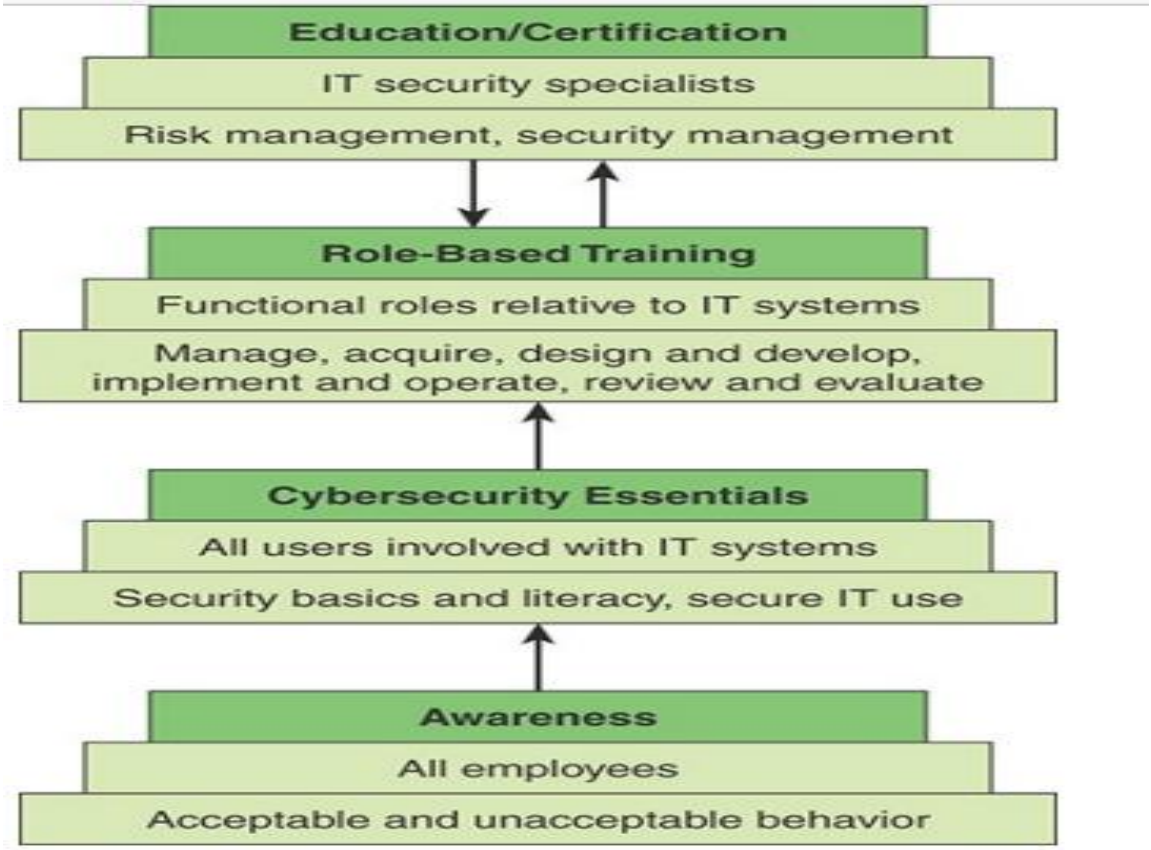


FIGURE 5.2 Cybersecurity Learning Continuum

1. <u>Security Awareness</u>

- ✓ All employees have security responsibilities; all employees must have suitable awareness training. Awareness seeks to focus an individual's attention on an issueor a set of issues
  - o security awareness
  - o security culture
  - o negligent behavior
  - o accidental behavior
  - o malicious behavior
  - o change management
    - ▪ Change management is designed to minimize resistance to

organizational change through involvement of key players and stakeholders.

- Plan, assess, and design
- Execute and manage
- Evaluate and adjust

Awareness Program Communication Materials

- ✓ An awareness training program are the communication materials and methodsused to convey security awareness.
  - ***Use in-house materials***
  - ***Use externally obtained materials***
- ✓ In-house materials that are effectively used include the following:
  - Brochures, leaflets, and fact sheets
  - Security handbook:
  - Regular email or newsletter:
  - Distance learning:
  - Workshop and training sessions:
  - Formal classes
  - Video:
  - Website
  - Emphasizing the difference between critical information and sensitive information, which must be treated differently.
    - **Critical information** -Information that needs to be available andhave integrity
    - **Sensitive information** -Information that can be disclosed only toauthorized individual

Awareness Program Evaluation

- Number of security incidents due to human behavior
- Audit findings
- Results of staff surveys
- Tests of whether staff follow correct procedures
- Number of staff completing training

2. **Cyber security Essentials Program**
   - ✓ Its principal function is to target users of IT systems and applications, including company-supplied mobile devices and bring your own device (BYOD) policies, and develop sound security practices for these employees.
   - ✓ Secondarily, it provides the foundation for subsequent specialized or role-based training by providing a universal baseline of key security terms and concepts.

Bring your own device (BYOD)

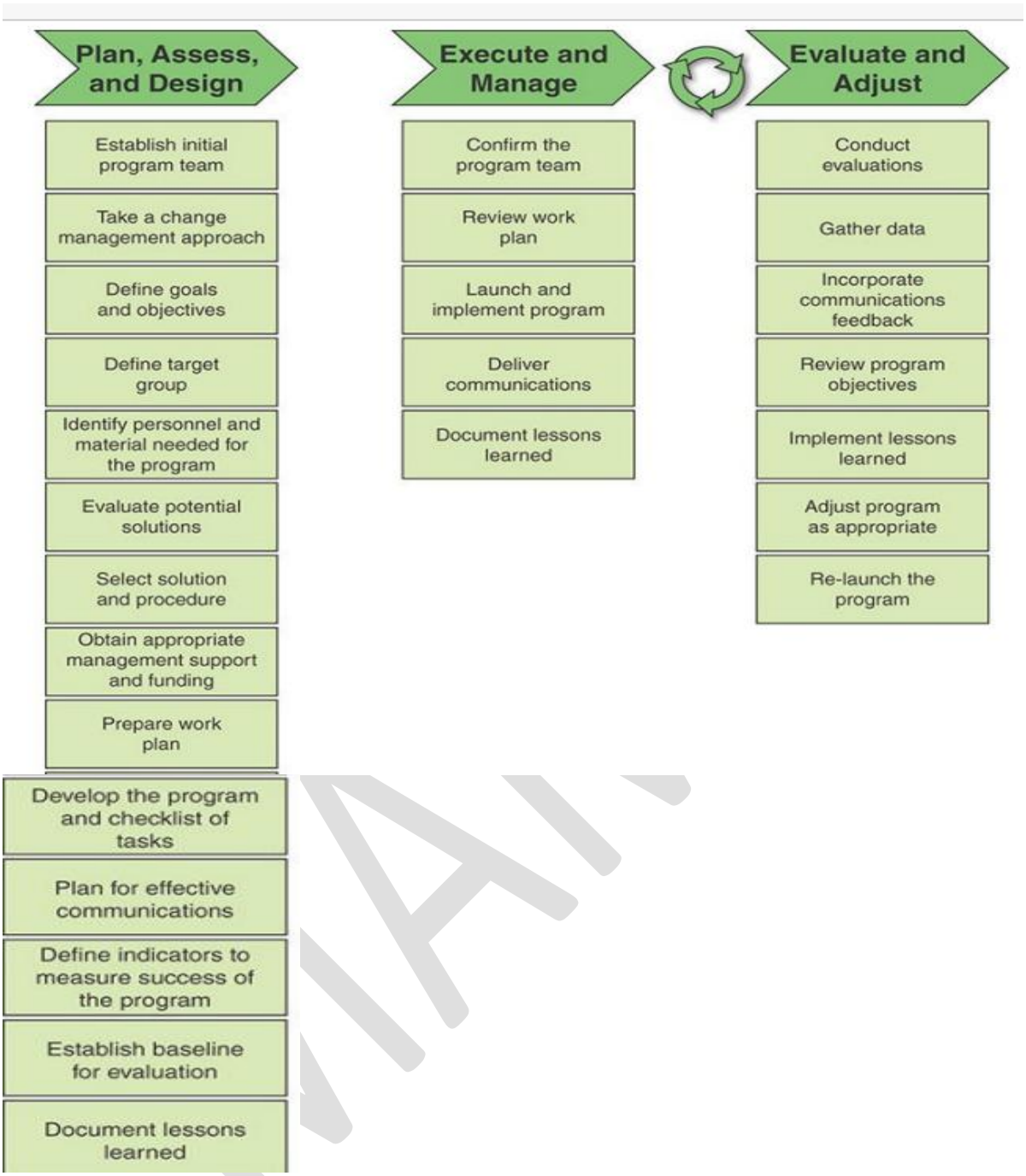
- ✓ An IT strategy in which employees, business partners, and others use their personally selected and purchased client devices to execute enterprise applications and access data and the corporate network.
- ✓ Key topics that should be covered include:
  - Technical underpinnings of cybersecurity and its taxonomy, terminology, and challenges
  - Common information and computer system security vulnerabilities
  - Common cyberattack mechanisms, their consequences, and motivations foruse
  - Different types of cryptographic algorithms
  - Firewalls and other means of intrusion prevention
  - Fundamental security design principles and their role in limiting points ofvulnerability

3. **Role-Based Training**
   - ✓ Role-based training is targeted at individuals who have functional rather than userroles with respect to IT systems and applications.
     - Manage
     - Design
     - Implement
     - Evaluate

4. **Education and Certification**
   - ✓ An education and certification program is targeted at those who have specific security responsibilities, as opposed to IT workers who have

some other.

- Global Information Assurance Certification (GIAC) Security Essentials (GSEC)
- Systems Security Certified Practitioner (SSCP):
- Information Systems Audit and Control Association (ISACA) CertifiedInformation Security Manager (CISM)
- SANS computer security training and certification



Security Awareness Processes

The three main processes are as follows:

- **Plan, assess, and design:** Awareness programs must be designed with the organization mission in mind. They should support the business needs of the organization and be relevant to the organization's culture and IT architecture. The most successful programs are those that users feel are relevant to the subject matter and issues presented. In the design step of the program, the awareness needs are identified, an effective awareness plan is developed, organizational buy-in is sought and secured, and priorities are established.
- **Execute and manage:** This process includes activities necessary to implement an information security awareness program. The initiative is executed and managed only when:
  - A needs assessment has been conducted
  - A strategy has been developed
  - An awareness program plan for implementing that strategy has been completed
  - Material has been developed
- **Evaluate and adjust:** Formal evaluation and feedback mechanisms are critical components of any security awareness program. The feedback mechanism must be designed to address objectives initially established for the program. Once the baseline requirements are solidified, design and implement a feedback strategy.

INFORMATION MANAGEMENT

The area of information management, according to the Information Security Forum's (ISF's) Standard of Good Practice for Information Security (SGP), encompasses four topics,

- ✓ **Information classification and handling:** Deals with methods of classifying and protecting an organization's
- ✓ **information assets Privacy** : Is concerned with threat, controls, and policies related to the privacy of personally identifiable information .
- ✓ **Document and records management**: Is concerned with the protection and handling of the documents and records maintained by an organization Sensitive

✓ **physical information**: Covers specific issues related to the security of information assets in physical form.

INFORMATION CLASSIFICATION AND HANDLING

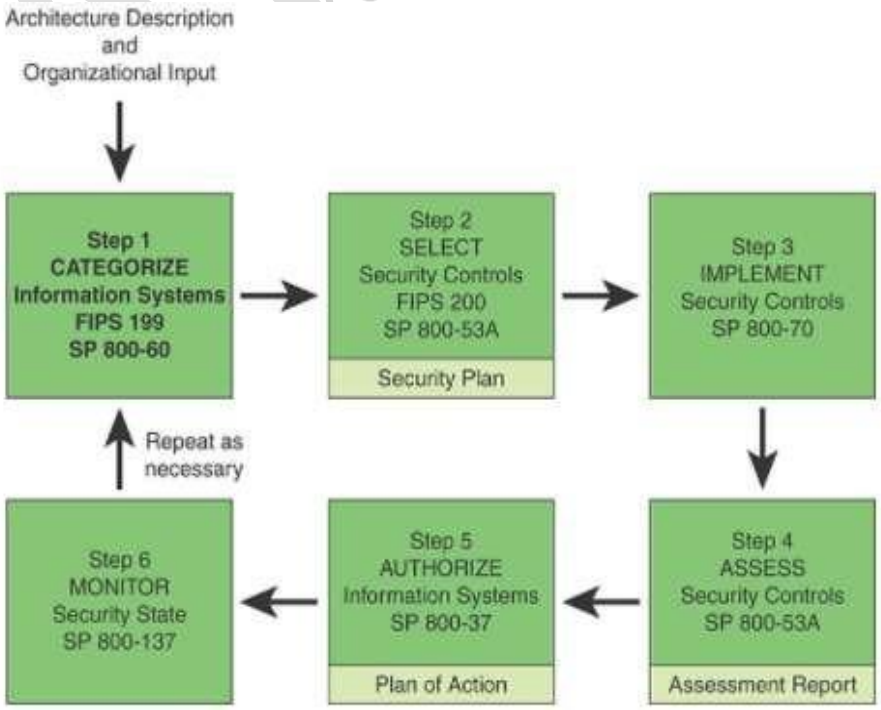
✓ A necessary preliminary step to the development of security controls and policies for protecting information is that all the information assets of the organization must be classified according to their importance and according to the impact of security breaches involving the information.

- Classification of information
- Labeling of information:
- Handling of assets:

1. Information Classification

✓ It is useful to view information classification and handling in the overall context of risk management.

- Categorize
- Select
- Implement
- Assess
- Authorize
- Monitor



- That are essential to information classification:

- Information type
- Security objective
- Impact
- Security classification

2. Information Labeling

- ✓ A label needs to be associated with each instance of an information type sothat its classification is clearly and unambiguously known.
- ✓ Methods are needed to ensure that a label is not separated from the information and that the content of the label is secure from unauthorized modification.

Radio-frequency identification (RFID)

- ✓ A data collection technology that uses electronic tags attached to items to allow the items to be identified and tracked with a remote system. The tag consists of an RFID chip attached to an antenna.

3. Information Handling

- ✓ Information handling refers to processing, storing, communicating, or otherwise handling information consistent with its classification.ISO 27002 lists the following relevant considerations:
- ✓ Access restrictions supporting the protection requirements for each level of classification
  - Maintenance of a formal record of the authorized recipients of assets
  - Protection of temporary or permanent copies of information to alevel consistent with the protection of the original information
  - Storage of IT assets in accordance with manufacturers' specifications

✓ An organization can also take advantage of broader information management tools, such as a document management system (DMS) or a records management system (RMS)

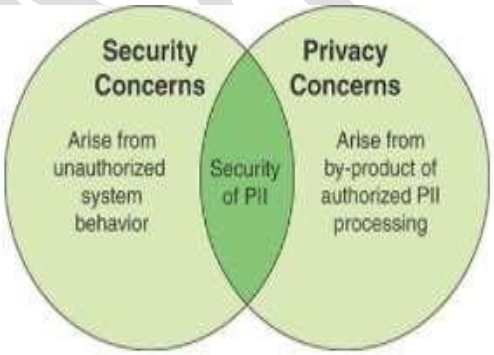
Data Loss Prevention (DLP)

✓ The automated tools should facilitate integration with other security tools, such as encryption and digital signature modules and data loss prevention (DLP) packages.

- ✓ A set of technologies and inspection techniques used to classify information content contained within an object—such as a file, an email, a packet, an application, or a data store—while at rest (in storage), in use (during an operation), or in transit (across a network).

PRIVACY

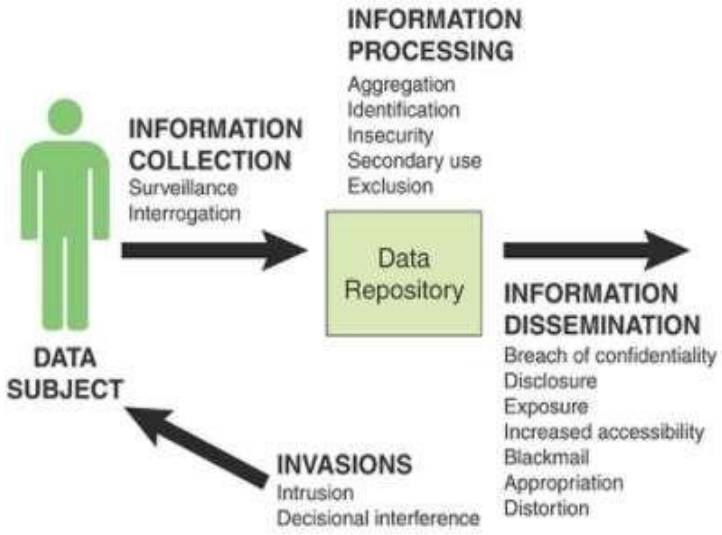
- ✓ The term privacy usually refers to making ostensibly private information about an individual unavailable to parties who should not have that information.
- ✓ Privacy interests attach to the gathering, control, protection, and use of information about individual.
- ✓ Examples
  - Personal identification number (PIN), such as Social Security number (SSN), passport number, driver's license number, taxpayer identification number, patient identification number, or financial account or credit card number
  - Telephone numbers, including mobile, business, and personal numbers



Relationship between Information Security and Privacy

1. Privacy Threats

- ✓ To understand the requirements for privacy, the threats must first beidentified



- ✓ Information collection is not necessarily harmful but can in some cases constitutea privacy threat.
  - Surveillance
  - Interrogation
  - Information processing refers to the use, storage, and manipulation of datathat have been collected.
    - Aggregation
    - Identification
    - Insecurity
    - Secondary use
- ✓ The area of information dissemination encompasses the revelation of personalinformation or the threat of such revelation.
  - Disclosure
  - Breach of confidentiality
  - Exposure
  - Blackmail
  - Appropriation
- ✓ The fourth area of privacy threats is referred to as invasions, and it involvesimpingements directly on the individual
  - Intrusion
  - Decisional interference

Privacy Principles and Policies

- ✓ A number of international organizations and national governments have

introduced standards, laws, and regulations intended to protect individual privacy.

ISO 29100

- ✓ ISO 29100, Privacy Framework, lists the following 11 privacy principles that form the bases of this international standard

European Union's GDPR

- ✓ One of the most comprehensive initiatives is European Union's (EU's) General Data Protection Regulation (GDPR), approved by the European Parliament in 2016,

U.S. Privacy Laws and Regulations

- ✓ There is no single law or regulation covering privacy in the United States. Rather,a collection of federal privacy laws cover various aspects of privacy;

U.S. Privacy Laws and Regulations

- ✓ There is no single law or regulation covering privacy in the United States. Rather,a collection of federal privacy laws cover various aspects of privacy;

2. Privacy Controls

- ✓ To counter privacy threats and comply with government laws and regulations, organizations need a set privacy controls that encompass their privacy requirements and that respond to legal requirements
  - Authority and purpose:
  - Accountability, audit, and risk management
  - Data quality and integrity
  - Data minimization and retention:
  - Individual participation and redress
  - Security:
  - Transparency
  - Use limitation

DOCUMENT AND RECORDS MANAGEMENT

- ✓ A specific class of information consists of documents and records.
- ✓ **Document:** A set of information pertaining to a topic, structured for human comprehension, represented by a variety of symbols, and stored and handled as a unit. A document may be modified.
- ✓ **Record:** A subclass of documents that clearly delineates terms and conditions, statements, or claims or that provides an official record. Generally a record, once created, is not modified.
- **Document management system**: Software that manages documents for electronic publishing. It generally supports a large variety of document formats and provides extensive access control and searching capabilities across networks.
- **Records management system:** Software that provides tools for and aids in records management.

1. Document Management

✓ Document management is a key business function because of the importance ofdocuments to the operation of an organization.

- To record or to document contracts and agreements
- To record policies, standards, and procedures
- To represent a view of reality at a point in time
- To create an image or impression
- To generate revenue as a product

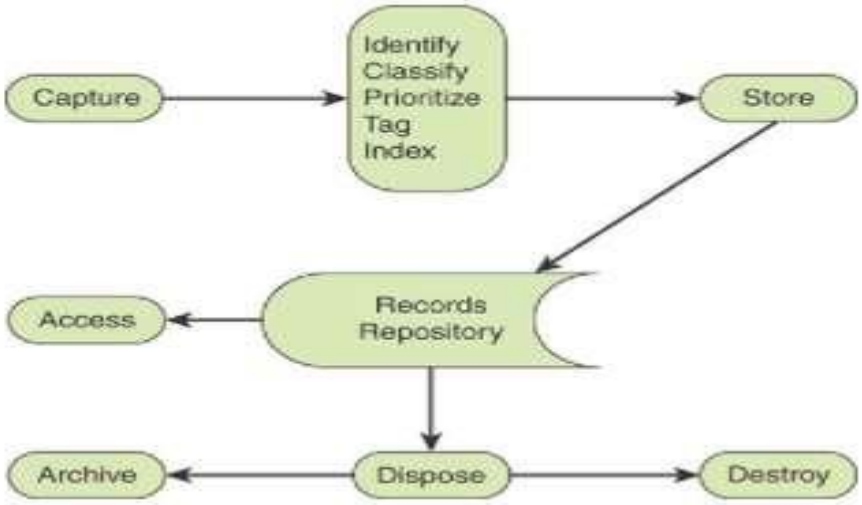
Document Management Life Cycle

2. Records Management

✓ Records management is a vital business function that requires stronger

security measures than those for document management.

✓



- ✓ The Information Security Guide developed by the Higher Education Information Security Council (HEISC).
  - Establish a strategic planning group that includes top managers/executives to support the program.
  - Identify a records manager to oversee all aspects of the program, including ongoing management and use of internal personnel or outside consultants.
  - Complete a records inventory.
  - Develop a records management manual with defined policies and procedures.
  - Make records management a high priority.
- ✓ A key aspect of records management is the retention and disposition policy.
  - Active
  - Semi-active
  - Inactive
  -

Physical Asset Management

The Information Security Forum's (ISF's) Standard of Good Practice for Information Security (SGP) uses the term physical asset to refer to all information and communications technology (ICT) hardware, including systems and network equipment; office equipment (such as, network printers and multifunction devices); mobile devices; and specialist equipment (for example, industrial control systems).

OFFICE EQUIPMENT

- ✓ Office equipment includes printers, photocopiers, facsimile machines, scanners, and multifunction devices (MFDs).
- ✓ Office equipment often contains the same components as a server (e.g., operating system, hard disk drives, and network interface cards) and runs services such as web, mail, and ftp.
- ✓ A multifunction device (MFD) is generally defined as a network-attached document production device that combines two or more of these functions: copy, print, scan, and fax.

1. Threats and Vulnerabilities

There are numerous potential threats to office equipment

1. Network Services
   - Management protocols:
   - Services protocols:
2. Information Disclosure
   - Print, fax, and copy/scan logs
   - Address books
   - Mailboxes

3.Denial-of-Service Attacks

   - Physical Security
   - Operating System Security

2. Security Controls

- ✓ A useful checklist of security measures an organization can take to protect MFDs isprovided in the SANS.
  - Network protocols and services
  - Management
  - Security updates
  - Physical security
  - The choice of measures should depend on the organization's risk assessment relative to a particular office device asset.
    - Physical device management
    - Remote device management:
    - Job access and processing
    - Application development platforms

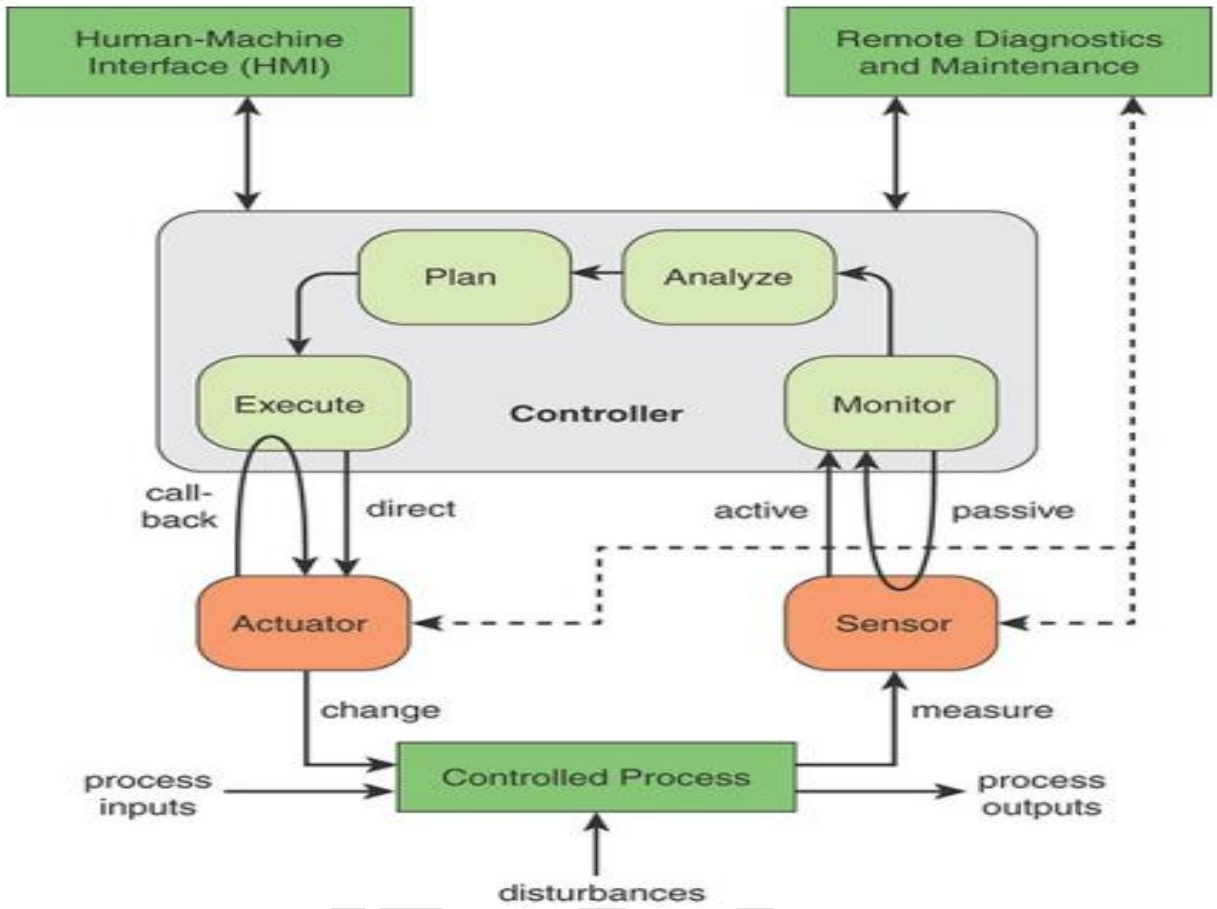
- User management:
- Logging and monitoring
- Miscellaneous

3. Equipment Disposal

- ✓ The SGP recommends that sensitive information stored on office equipmentbe securely destroyed.
- ✓ Three increasingly secure actions for sanitization are defined:
  - Clear
  - Purge
- ✓ **Self-Encrypting Drive (SED)**
  - A hard drive with a circuit built into the disk drive controller chip that encrypts all data to the magnetic media and decrypts all the data from the media automatically.

INDUSTRIAL CONTROL SYSTEMS

- ✓ An industrial control system (ICS) is used to control industrial processes such as manufacturing, product handling, production, and distribution.
- ✓ Industrial control systems include supervisory control and data acquisition (SCADA) systems used to control geographically dispersed assets, as well as distributed control systems (DCSs) and smaller control systems, using programmable logic controllers to control localized processes.



- ✓ **Sensor**: A sensor measures some parameter of a physical, chemical, or biological entity and delivers an electronic signal proportional to the observed characteristic, either in the form of an analog voltage level or a digital signal. In both cases, the sensor output is typically input to a microcontroller or other management element.
- ✓ **Actuator**: An actuator receives an electronic signal from a controller and responds by interacting with its environment to produce an effect on some parameter of a physical, chemical, or biological entity.
- ✓ **Controller**: The controller interprets the signals and generates corresponding manipulated variables, based on a control algorithm and target set points, which it transmits to the actuators. The controller may have very limited intelligence and may rely on the human–machine interface for direction. But typically, a controller adjusts the directives given to actuators automatically, based on sensor input.
- ✓ **Human–machine interface**: Operators and engineers use human interfaces to monitor and configure set points, control algorithms, and adjust and establish parameters in the controller.

- ✓ **The human interface** also displays process status information and historical information. Remote diagnostics and maintenance: Diagnostics and maintenance utilities are used to prevent, identify, and recover from abnormal operation or failures.

Differences Between IT Systems and Industrial Control Systems

IT Systems	ICSs
Response must be consistent	Response is time-critical.
Responses such as rebooting are acceptable	Responses such as rebooting may not be acceptable because of process availability requirements
Manage data.	Control physical world
Systems are specified with enough resources to support the addition of third- party applications such as security solutions	Systems are designed to support the intended industrial process and may not have enough memory and computing resources to support the addition of security capabilities.
Standard communications protocols are used.	Many proprietary and standard communication protocols are used
Component lifetime is on the order of 3 to 5years	Component lifetime is on the order of 10 to15 years.

ICS Security

- ✓ The Department of Homeland Security's (DHS's) Recommended Practice: Improving Industrial Control System Cyber security with Defense-in-Depth Strategies.
- ✓

IT Systems	ICSs
Easily deployed and updated	Ability to protect legacy systems with after-marketsolutions
Multiple vendors	Usually the same vendor over time
Modern methods	Modern methods possibly inappropriate
Regular and scheduled	Strategic scheduling

Integral part of the development process	Historically not an integral part of the development process

Typical Security Threats to ICSs

- ✓ Blocked or delayed flow of information through ICS networks, which could disruptICS operation.
- ✓ Unauthorized changes to instructions, commands, or alarm thresholds, whichcould damage, disable, or shut down equipment
- ✓ Inaccurate information sent to system operators.
- ✓ ICS software or configuration settings modified, or ICS software infected withmalware, which could have various negative effects.

Key Security Measures

- The DHS has offered these recommendations for protecting ICSs.
  - Implement application white listing:
  - Ensure configuration management and patch management
  - Reduce attack surface areas

Attack surface

- The reachable and exploitable vulnerabilities in a system.
  - Build a defendable environment
  - Manage authentication
  - multifactor authentication (MFA)

Resources for ICS Security

- Recommended Practice: Improving Industrial Control System Cybersecurity with Defense-in-Depth Strategies
- Cyber Security Assessments of Industrial Control Systems
- SP 800-82, Guide to Industrial Control Systems Security
- Catalog of Control Systems Security: Recommendations for StandardsDevelopers

MOBILE DEVICE SECURITY

- ✓ A mobile device as a "portable computing and communications device withinformation storage capability".
  - Growing use of new devices
  - Cloud-based applications
  - External business requirements

Growing use of new devices: Organizations are experiencing significant growth in employee use of mobile devices. In many cases, employees are allowed to use a combination of endpoint devices as part of their day-to-day activities.

Cloud-based applications: Applications no longer run solely on physical servers in corporate data centers. Today applications can run anywhere—on traditional physical servers, on mobile virtual servers, or in the cloud. In addition, end users can now take advantage of a wide variety of cloudbased applications and IT services for personal and professional use.

External business requirements: An enterprise must also provide guests, third-party contractors, and business partners network access using various devices from a multitude of locations

Mobile Device Technology

Providing enterprise security for mobile devices is extraordinarily complex, both because of the technology of these devices and the ecosystem in which they operate.

Hardware:

The base layer of the technology stack is the device hardware. This includes an application processor, with the ARM family of processors being the most common. There is also a separate processor that runs the cellular network processor, typically referred to as the baseband processor.

Firmware: The firmware necessary to boot the mobile operating system (that is, boot loader) may verify additional device initialization code, device drivers used for peripherals, and portions of the mobile operating system— all before a user can use the device. I

Mobile operating system: The most common operating systems for mobile devices are Android and iOS. An operating system includes a sandbox facility for isolating third-party applications in some manner to prevent unexpected or unwanted interaction between the system, its applications, and applications' respective data (including user data).

Application: The application layer includes third-party applications, various apps and services provided by the mobile device vendor, and facilities for defining permissions

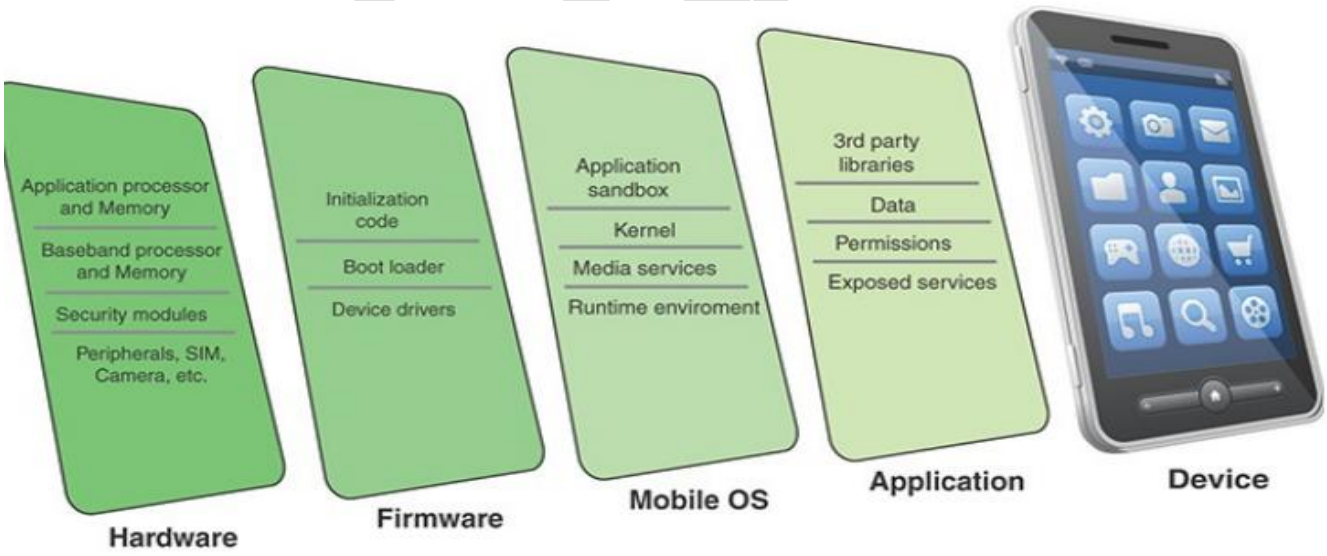


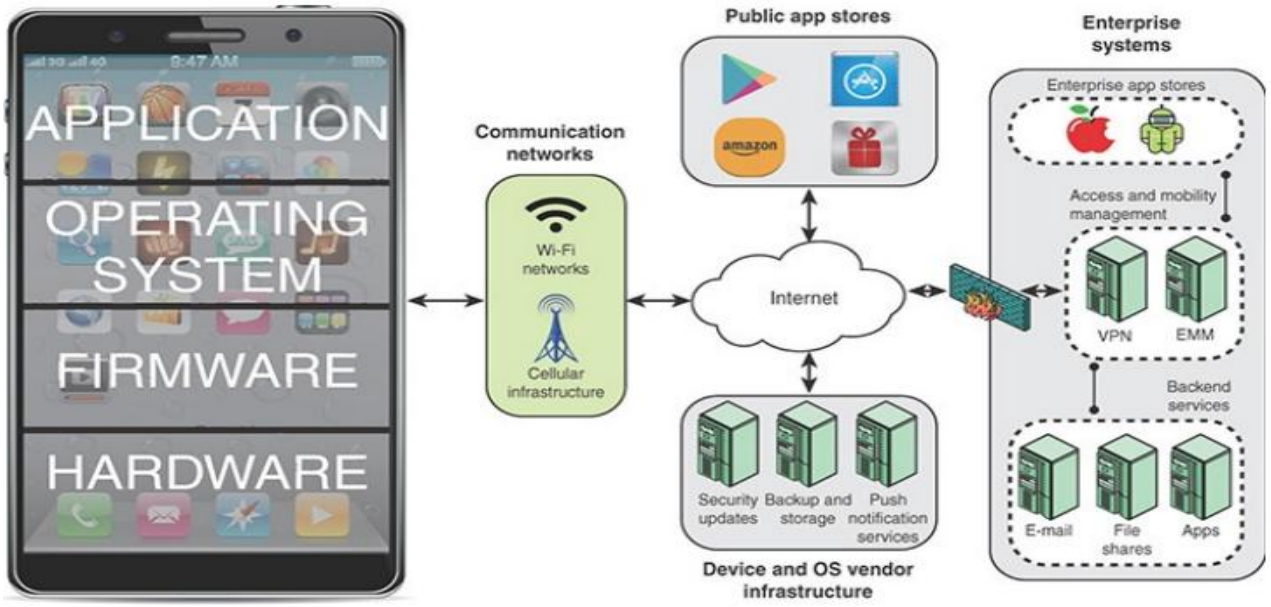
FIGURE 7.4 Mobile Device Technology Stack

1. **Mobile Ecosystem**
   - ✓ The execution of mobile applications on a mobile device may involve communication across a number of networks and interaction with a number

of systems owned and operated by a variety of parties.

- ✓ This ecosystem makes the achievement of effective security challenging.
  - Cellular and Wi-Fi infrastructure:
  - Public application stores
  - Private application stores
  - Device and OS vendor infrastructure:
  - Enterprise mobility management systems
  - Enterprise mobile services
  - app store

- ✓ **Cellular and Wi-Fi infrastructure**: Modern mobile devices are typically equipped with the capability to use cellular and Wi-Fi networks to access the Internet and to place telephone calls. Cellular network cores also rely on authentication servers to use and store customer authentication information.
- ✓ **Public application stores (public app store):** Public app stores include native app stores, which are digital distribution services operated and developed by mobile operating system vendors. For Android, the official app store is Google Play, and for iOS, it is simply called the App Store.
- ✓ **Device and OS vendor infrastructure**: Mobile device and operating system vendors host servers to provide updates and patches to operating systems and apps. Other cloud-based services may be offered, such as storing user data and wiping missing devices.
- ✓ **Enterprise mobility management systems**: Enterprise mobility management (EMM) is a general term that refers to everything involved in managing mobile devices and related components (such as wireless networks). EMM is much broader than just information security; it includes mobile application management, inventory management, and cost management.
- ✓ **Enterprise mobile services:** These back-end services are accessible from authorized users' mobile devices, including email, file sharing, and other applications



Mobile Eco system

2. Vulnerabilities

- ✓ Mobile devices need additional specialized protection measures beyond those implemented for other client devices, such as desktop and laptop devices that are used only within the organization's facilities and on the organization's networks.
  - Lack of Physical Security Controls
  - Use of Untrusted Mobile Devices
  - Use of Untrusted Networks
  - Use of Applications Created by Unknown Parties
  - Use of Untrusted Content
  - Use of Location Services

3. Mobile Device Security Strategy

- ✓ The recent DHS report Study on Mobile Device Security groups security threats and defenses into five primary components of the mobile ecosystem and their associated attack surface: the mobile device technology stack, mobile applications, mobile network protocols and services, physical access to the device, and enterprise mobile infrastructure.
  - *Rooting*
  - *Side loading*

Common Mobile Device Threats

- Mobile device technology stack
- Mobile applications
- Mobile networks
- Device physical systems
- Mobile enterprise

4. Resources for Mobile Device Security

- ✓ A number of documents for U.S. agencies are valuable resources for any organization.
  - Study on Mobile Device Security
  - Assessing Threats to Mobile Devices & Infrastructure
  - Guidelines for Managing and Securing Mobile Devices in the Enterprise
  - Test the Security of Mobile Applications:

- Guidelines on Hardware-Rooted Security in Mobile Devices
- Mobile Device Security: Cloud and Hybrid Builds

System Development

- ***The system development life cycle (SDLC)*** **is the overall process of developing, implementing, and retiring information systems. Various SDLC models have been developed to guide the processes involved, and some methods work better than others for specific types of projects.**

INCORPORATING SECURITY INTO THE SDLC

- ✓ Enterprise security considerations dictate that security-focused activities and deliverables be a part of every phase of the SDLC in order to ensure that the developed system is able to withstand malicious attacks.
- ✓ Whatever SDLC methodology is used, an enterprise should make sure both that the SDLC methodology lends itself to a comprehensive security component and that security policies and procedures are well defined for each phase.
- ✓ The following elements in defining the security considerations applied during each phase:
  - Major security activities
  - Expected outputs
  - Synchronization
  - Control gates

1. Initiation Phase

- ✓ The relationships among five key security-related activities that comprise the initiation phase. During this initial phase, the security focus is on identifying and assessing risks.

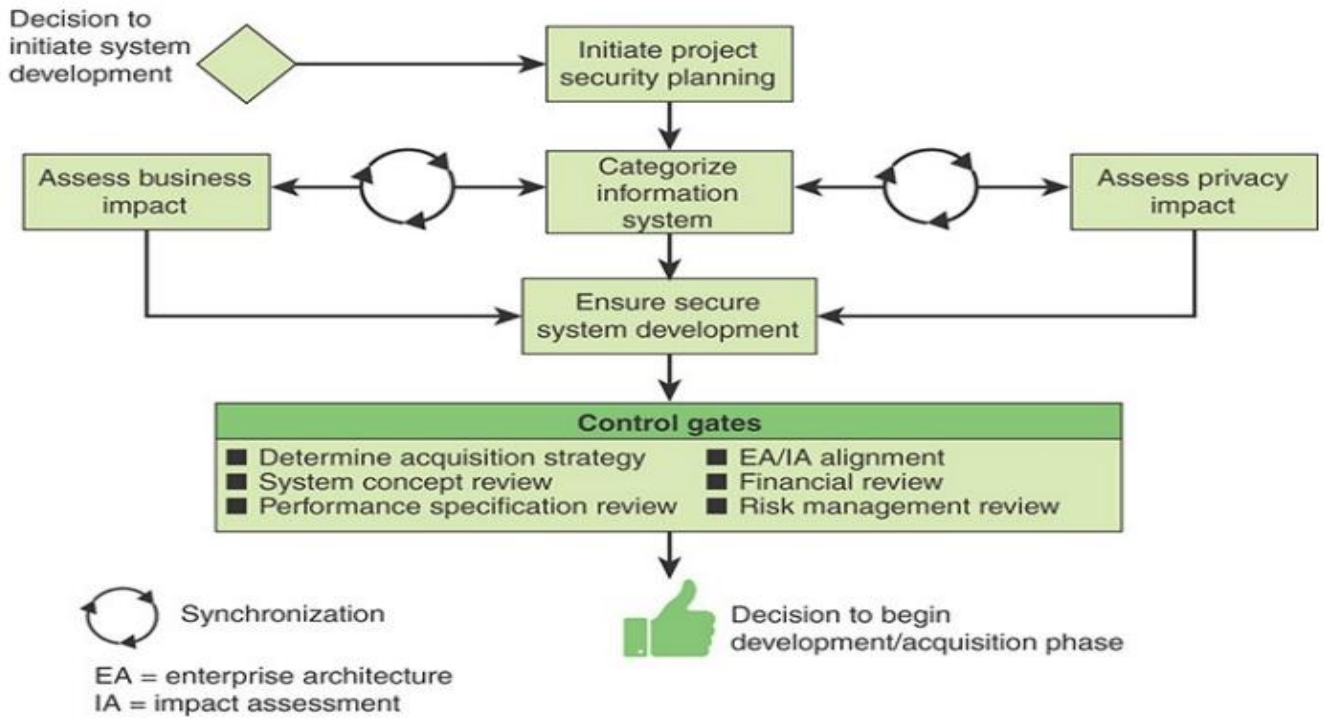


FIGURE 8.5 Security in the Initiation Phase

- Initiating Project Security Planning
  - system owner
  - Categorizing Information Systems and Assessing Impact
    - Ensuring Secure System Development
    - Secure concept of operations:
    - Standards and processes
    - Security training for development team
    - Quality management:
    - source code repository
    - Control Gates
      - Determine acquisition strategy
      - System concept review
      - Performance specification review
      - Financial review
      - Risk management review

2. Development/Acquisition Phase

- ✓ The relationships among six key security-related activities that comprise thedevelopment/acquisition phase.
- ✓ A key security activity in this phase is conducting a risk assessment and using theresults to supplement the baseline security controls.

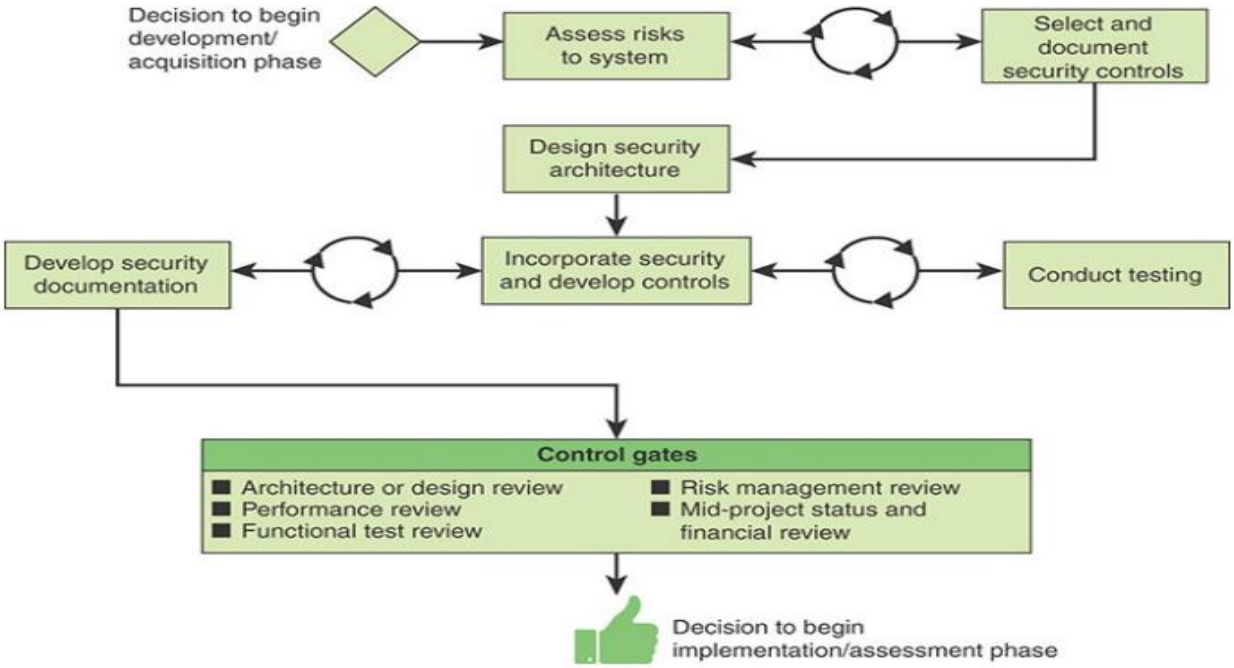


FIGURE 8.6 Security in the Development/Acquisition Phase

- ✓ Assessing Risks and Selecting Controls
- ✓ Developing, Testing, and Documenting Security Controls and Features
  - functional testing
  - penetration testing
  - user testing
- ✓ Control Gates
  - Architecture/design review:
  - Performance review
  - Functional test review:
  - Risk management review

3. Implementation/Assessment Phase

- ✓ The relationship among four key security-related activities that comprise theimplementation/assessment phase.

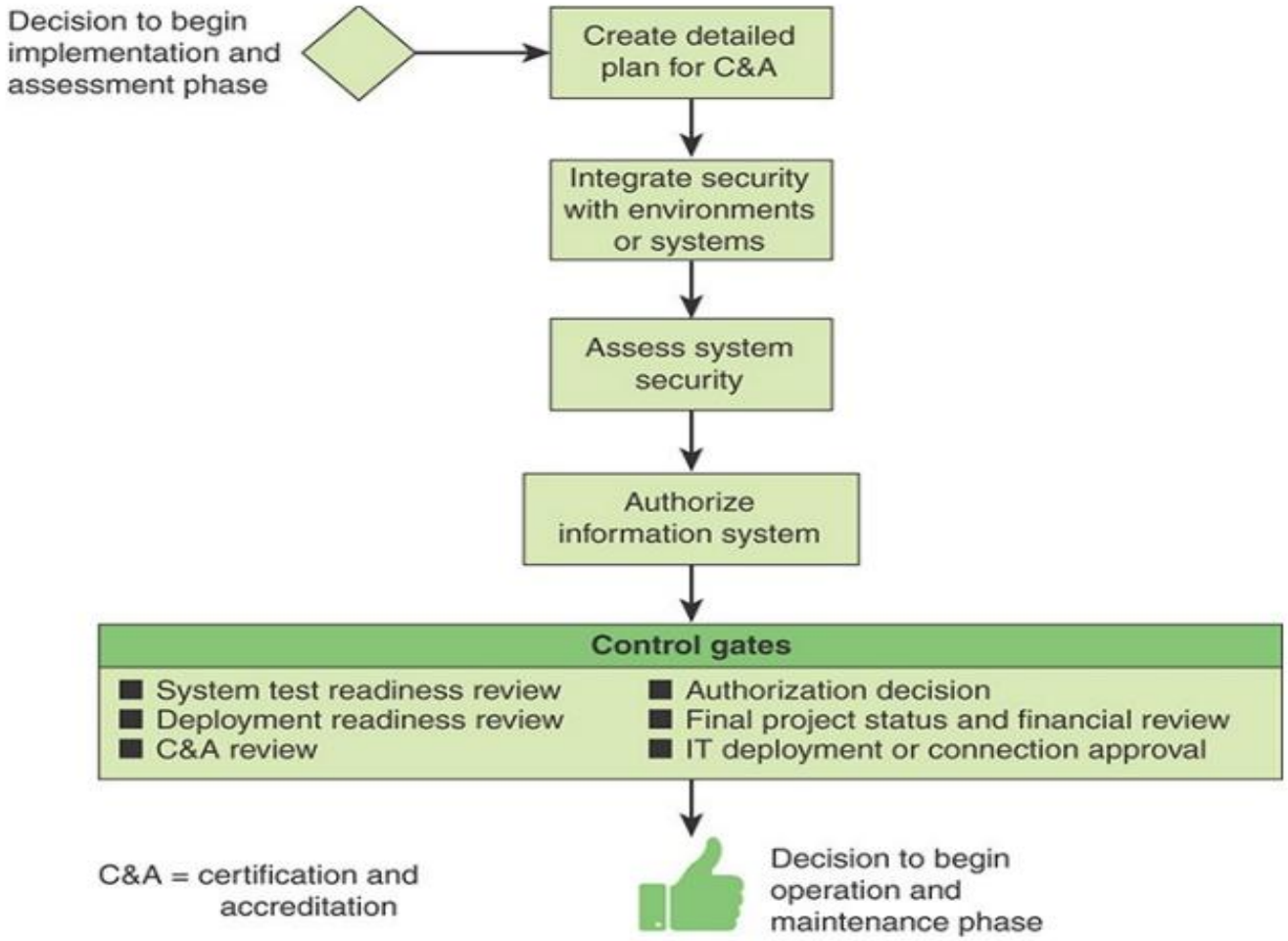


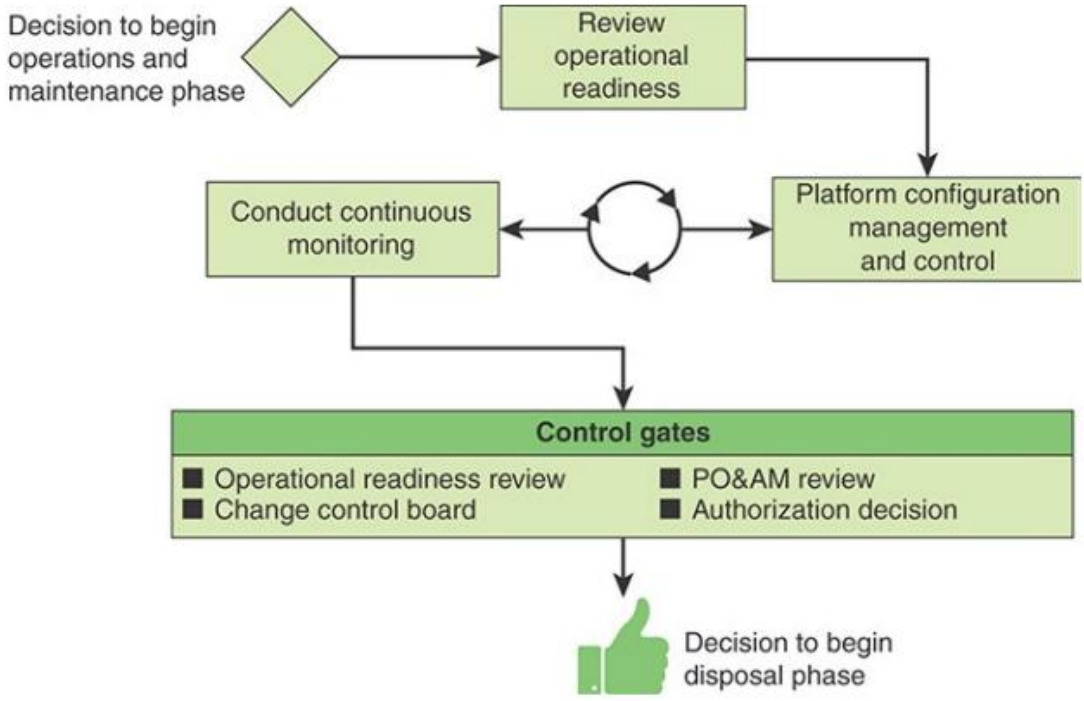
FIGURE 8.7 Security in the Implementation/Assessment Phase

- ✓ Creating a Detailed Plan for C&A
  - Certification and Accreditation (C&A)
- ✓ Integrating Security with Environments or Systems
- ✓ Assessing System Security
- ✓ Authorizing Information System
- ✓ Control Gates
  - System test readiness review
  - Deployment readiness review
  - Certification and accreditation (C&A) review

- Authorization decision
- Final project status and financial review

4. Operations and Maintenance Phase

✓ The relationship among three key security-related activities that comprise theoperations and maintenance phase.



✓ Reviewing Operational Readiness
✓ Ensuring Configuration Management and Control
✓ change control
✓ change control board (CCB)
✓ Conducting Monitoring Continuously
✓ Control Gates
- Operational readiness review:
- Change control board
- Authorization decision

5. Disposal Phase

- ✓ The disposal phase is the process of preserving (if applicable) and discarding system information, hardware, and software. During this phase, information, hardware, and software are moved to another system, archived, discarded, or destroyed.
- ✓ The disposal phase involves the following control gates:
  - System closure review
  - Change control board
  - Security review of closure
  -

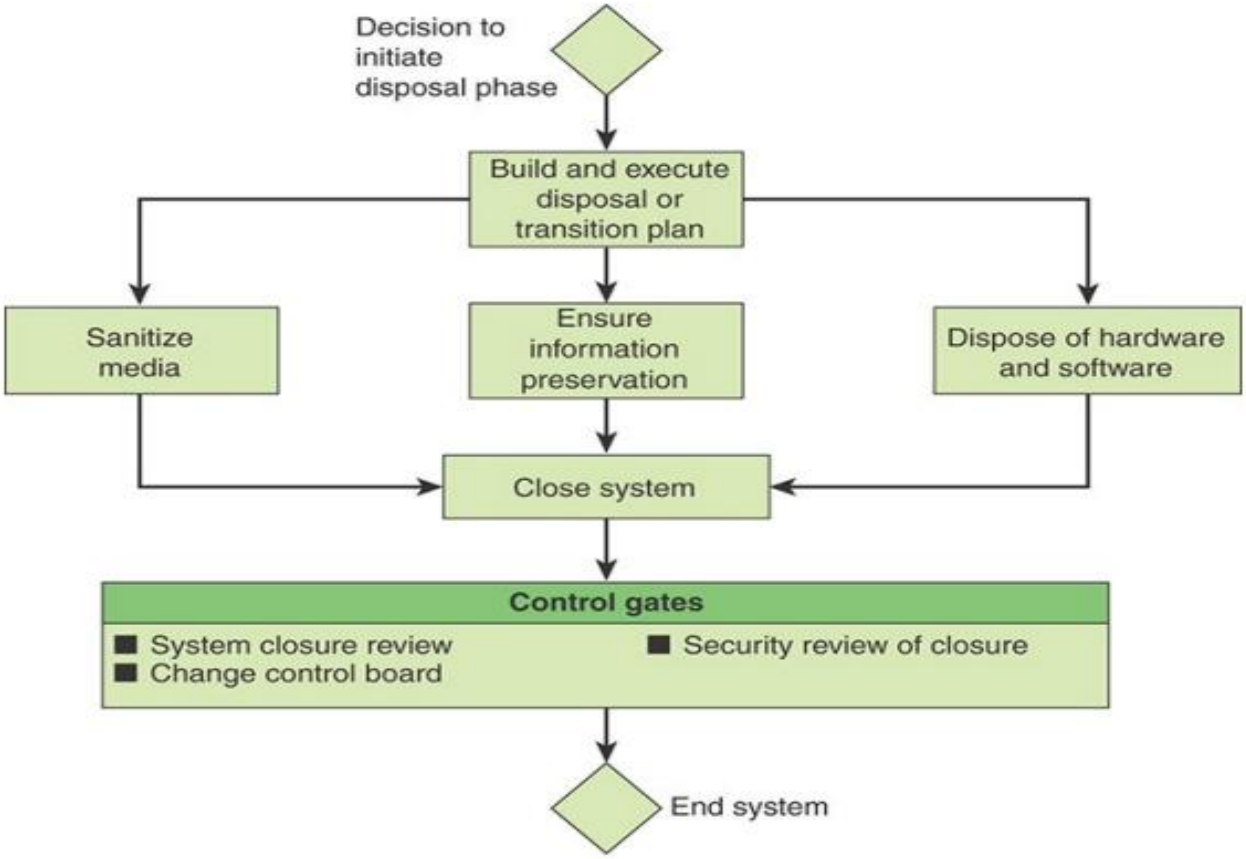


FIGURE 8.9 Security in the Disposal Phase

- ✓ The following key activities:
  - Create a disposal/transition plan
  - Ensure information protection
  - Dispose of hardware and software
  - Close system

DISASTER MANAGEMENT

- Disaster recovery is generally a planning process and it produces a document which ensures businesses to solve critical events that affect their activities.
- Such events can be a natural disaster (earthquakes, flood, etc.), cyber–attack or hardware failure like servers or routers.

Cyber attack in disaster management

- Cyber attacks aim to disable, disrupt, destroy or control computer systems or to alter, block, delete, manipulate or steal the data held within these systems.
- A cyber attack can be launched from anywhere by any individual or group using one or more various attack strategies.

Steps to creating a disaster recovery plan

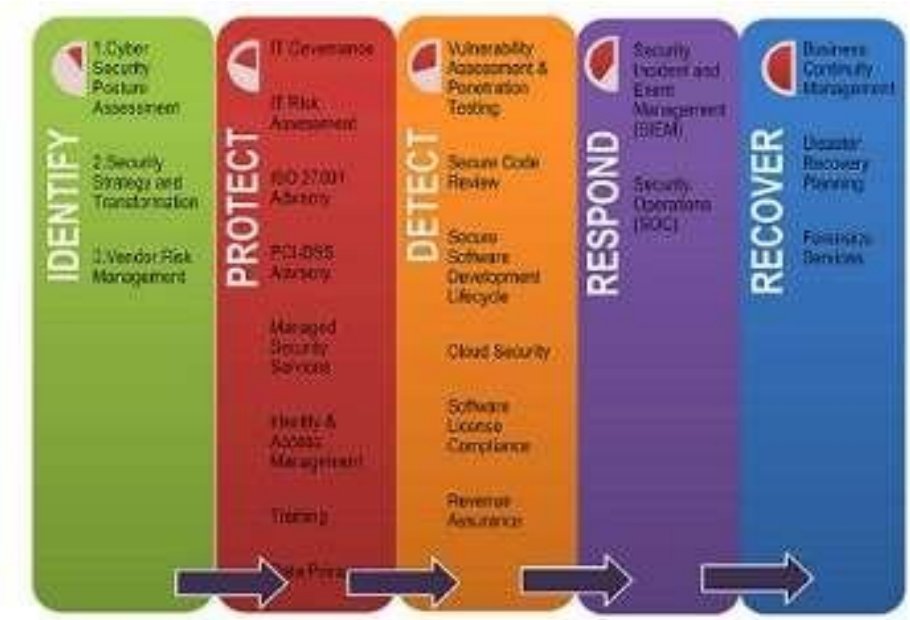
- Identify representatives from each area of the business
- Document your risks
- Specify which data, technologies, and tools are most critical
- Maintain an inventory of physical assets
- Determine where and how critical business information will be backed up
- Protection for every computer
- Taps the benefits of cloud backup
- Runs automatically
- Prioritizes easy recovery

Cyber security Disaster Recovery Plan

- Requirements to Have a Disaster Recovery Plan
- Disaster recovery starts with an inventory of all assets like computers, network equipment, server

Example

Systems	Down Time	Disaster type	Preventions	Solution strategy	Recover fully
Payroll system	8 hours	Server damaged	We take backup daily	Restore the backups in the Backup Server	Fix the primary server and restore up to date data



Preventive steps to be taken for Disaster Recovery

- The server room should have an authorized level. For example: only IT personnelshould enter at any given point of time.
- In the server room there should be a fire alarm, humidity sensor, flood sensor anda temperature sensor.
- At the server level, RAID systems should always be used and there should alwaysbe a spare Hard Disk in the server room.
- You should have backups in place, this is generally recommended for local and off-site backup, so a NAS should be in your server room.
- Backup should be done periodically.

INCIDENT RESPONSE PLANNING

- An incident response plan is a set of written instructions that outline your organization's response to data breaches, data leaks, cyber attacks and security incidents.
- Incident response planning contains specific directions for specific attack scenarios, avoiding further damages, reducing recovery time and mitigating cyber security risk.

- Incident response procedures focus on planning for security breaches and howorganizations will recover from them.

Responsible for Incident Response Planning

- Organizations should form a computer security incident response team (CSIRT) who is responsible for analyzing, categorizing and responding to security incidents.
    - Incident response manager
    - Security analysts
    - Threat researchers

Different Types of Security Incidents

- Types of malware
- Man-in-the-middle attacks
- Social engineering
- Data breaches
- Data leaks
- Email spoofing
- Domain hijacking
- Denial of service (DoS)

Tools are Available for Incident Response

- There are tools and industry standards that can be helpful to incident responseteams.
- Tools can be split into three categories:
    - Prevention
    - Detection
    - Response

What is the Industry Standard for Incident Response?

- There are two frameworks that have become industry standard, the NIST Incident Response Process and the SANS Incident Response Process.

- The NIST Incident Response Process is four steps:
  - Preparation
  - Detection and analysis
  - Containment, eradication and recovery
  - Post-incident activity
- The SANS Incident Response Process is six:
  - Preparation
  - Identification
  - Containment
  - Eradication
  - Recovery
  - Lessons learned

UNIT III CYBER SECURITY FOR BUSINESS APPLICATIONS AND NETWORKS

Business Application Management-Corporate Business Application Security-

End user DevelopedApplications-System Access-

Authentication Mechanisms-Access Control-System Management- Virtual Servers-Network Storage Systems-

Network Management Concepts-Firewall-IP Security- Electronic Communications

- Case study on OWASP vulnerabilities using OWASP ZAP tool.

Business Application Management

- Business application management and security is a complex field. Applications encompass purpose-built applications developed in-house or by contractors, applications supplied by application and operating system vendors, and open source application software. Applications may operate on a variety of platforms, including workstations, PCs, mobile devices, and web based. They may also need to access and generate a wide variety of data files and databases.

CORPORATE BUSINESS APPLICATION SECURITY

Application security overlaps with many of the topics covered in other chapters but needs to be considered as a separate security concern as well.

The aim of web application security is to identify the following:

- Critical assets of the organization
- Genuine users who may access the data
- The level of access provided to each user
- Various vulnerabilities that may exist in the application
- Data criticality and risk analysis on data exposure
- Appropriate remediation measures

Application security

- The use of software, hardware, and procedural solutions to protect applications from external threats.
- This includes adding features or functionality to application software to prevent a range of different threats.

- It also includes security features outside the application, such as firewalls, antivirus software, and access control methods.

1. Business Application Register

- Application portfolio management, there should be an inventory, or register, of all applications, with details concerning the application, including security- related aspects.

TABLE 9.3 Information to Be Included in a Business Application Register

Category	Information
Type	▪ Developed in-house, **commercial-off-the-shelf (COTS) software**, cloud-based software, mobile-based software, end user-developed software
Operational	▪ Business purpose ▪ Business processes supported by the application
	▪ Relative importance to the organization (for example, critical, important, nonessential) ▪ Owner
Users	▪ Type and number of users ▪ Type and volume of connections
Access security	▪ Method of user authentication ▪ Network security barriers (for example, firewall, IPsec)
Type of data accessed	▪ Personally identifiable information ▪ Sensitive information (requires strong confidentiality) ▪ Requires strong availability ▪ Requires strong integrity
Technical	▪ Application version ▪ Supplier and licensing requirements ▪ Technical support contact

Commercial-off-the-shelf (COTS) software

- Software that is commercially available, leased, licensed, or sold to the general public and that requires no special modification or maintenance over the life cycle of the product to meet the needs of the procuring agency.

2.Business Application Protection

- The business assets, sound security architecture principles should be appliedto business applications.
- The considerations are somewhat different for two categories: *internally developed applications and externally-developed applications*.

Internal Application Security

For any application that is developed within the organization, it is essential to incorporate security into all stages of the SDLC

- Document security requirements.
- Develop standardized procedures for evaluating application security products and services.
- Enforce compliance with government and industry standards and regulations.
- Develop a policy for pre-deployment application testing and validation
- Construct a policy for documentation of application code review.

External Application Security

- The external environment, including the host operating system or virtual operating system, the hardware platform, and network connections
  - Protection against unauthorized access using access control measuresat the operating system level
  - Enforcement of virtual platform security
  - Encryption of network traffic using Transport Layer Security (TLS) orInternet Protocol Security (IPsec)

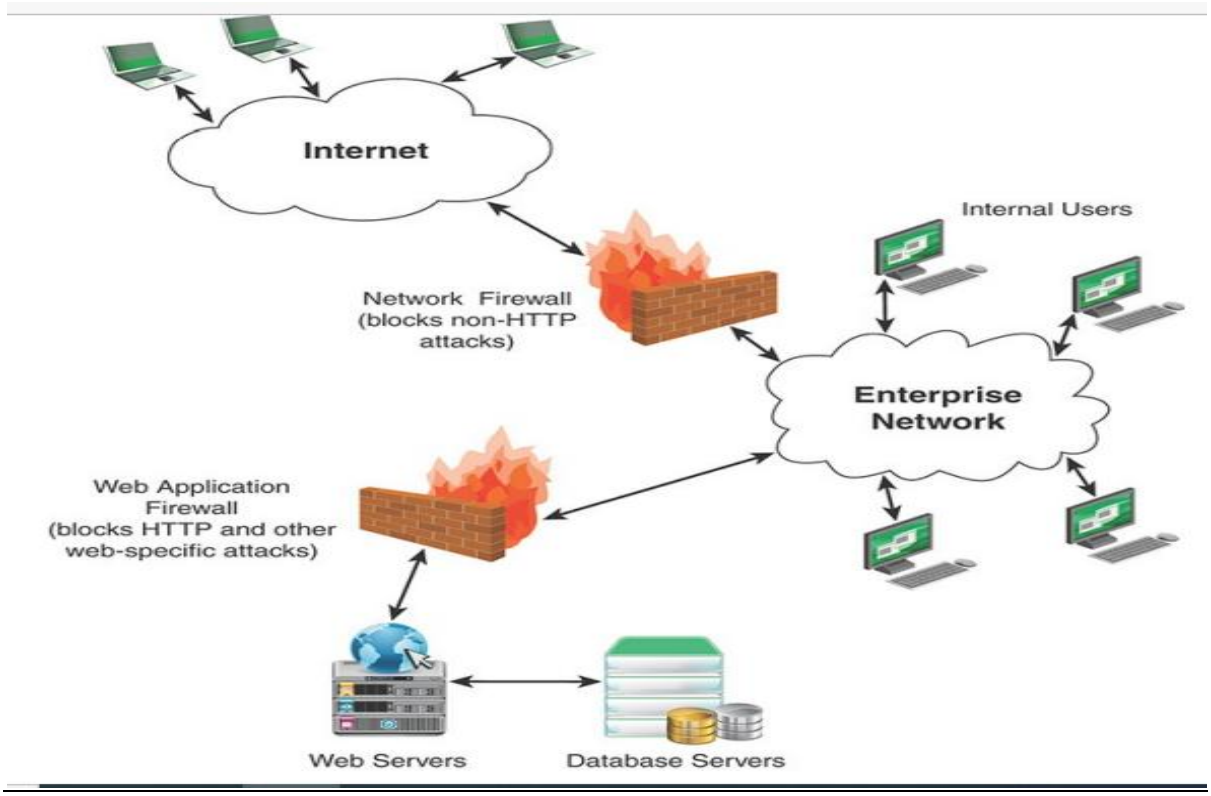
3. Browser-Based Application Protection

- As enterprises move applications online, both for internal use and for external users, such as customers and vendors, web application security becomes an increasing concern.

- Web Application Security Risks
  - Injection
  - Broken authentication
  - Sensitive data exposure
  - Security misconfiguration
  - XML external entity
  - Insecure deserialization
  - Broken access control
  - Using components with known vulnerabilities
  - Cross-site scripting (XSS)
  - Insufficient logging and monitoring

Web Application Firewall

- The most important tool in countering web application threats is a web application firewall (WAF), a firewall that monitors, filters, or blocks data packets as they travel to and from a web application.



Context for a Web Application Firewall

- There are a number of hosting options for WAFs, including the following:
  - Network-based
  - Local hardware:
  - Local software

Network-based:

- A network-based firewall is a hardware firewall incorporated with a router at the edge of an enterprise network, acting as a filter to all traffic to and from network devices, including web-based application servers.
- Because there may be a variety of web applications on a number of servers, this approach can be complex to maintain. In addition, a network-based firewall may not be placed so as to catch internal traffic.

Local hardware:

- A local hardware firewall is placed between the application server and its network connection or connections.
- This type of firewall is much simpler than a network-based firewall because it only has to have logic for filtering traffic specific to the local server.

Local software:

- A local software firewall is built on the server host operating system or virtual machine operating system.
- This approach can be as effective as a local hardware firewall and is easier to configure and modify

Key features

- Real-time application security monitoring and access control:
- Virtual patching
- Full HTTP traffic logging
- Web application hardening

Real-time application security monitoring and access control:

- All HTTP traffic in both directions passes through Mod Security, where it can be inspected and filtered. Mod Security also has a persistent storage

mechanism, which enables tracking of events over time to perform event correlation.

Virtual patching:

- This is the ability to apply web application patching without making changes directly to the application.
- Virtual patching is applicable to applications that use any communication protocol, but it is particularly useful with HTTP because the traffic can generally be well understood by an intermediary device.

Full HTTP traffic logging:

- Web servers traditionally do very little when it comes to logging for security purposes. Mod Security gives you the ability to log events, including raw transaction data, which is essential for forensics.
- In addition, the system manager gets to choose which transactions are logged, which parts of a transaction are logged, and which parts are sanitized.

Web application hardening:

This is a method of attack surface reduction in which the system manager selectively narrows down the HTTP features that will be accepted (for example, request methods, request headers, and content types).

Web Application Security Policy

- The European Union Agency for Network and Information Security (ENISA').
- Data Breach Investigations Report web application vulnerabilities accountfor the largest portion of attack vectors after malware.

END USER-DEVELOPED APPLICATIONS (EUDAS)

- Some business processes are supported by user-developed applications referred to as EUDAs that are outside the formal systems supported by the IT organization
- Examples of activities by end user programmers include:
  - Using spreadsheets for accounting
  - Using Mat Lab for analysis
  - Creating a web page
  - Recording macros in Word

- Automating office tasks
- Creating business software (SAP programming)

1. Benefits of EUDAs

- Convenience and ease of use
- Powerful tools and technology-aware end users
- Demand for information

Benefit of EUDAs include the following:

Convenience and ease of use:

EUDAs can be developed easily and quickly by non-IT staff. Business users frequently become frustrated with the amount of time taken by the IT department to service their requests.

They therefore often resort to developing their own solutions, using applications such as Microsoft Excel to meet their reporting needs. EUDAs allow businesses and users to quickly deploy solutions in response to shifting market and economic conditions, industry changes, or evolving regulations.

Powerful tools and technology-aware end users:

End-user tools offer rich functionality, including the ability to connect to corporate data sources.

As a result, technology-savvy users can perform powerful data processing from their desktops. This can help plug functionality gaps for business systems.

Demand for information:

Traditionally, managers were often constrained by standard reports in IT systems that failed to meet all management information and reporting requirements.

The lack of flexibility in these systems and increasing demand for different views of the data have resulted in an increase in the level of end-user computing in organizations.

2. Risks of EUDAs

- User-developed and user-controlled applications are generally not subject to the same development, monitoring, and reporting rigor and

control as traditional applications.

- Errors
- Poor version and change control
- Poor documentation
- Lack of security
- Lack of an audit trail
- Regulatory and compliance violations:
- Risk of the unknown

This leads to a number of disadvantages and risks related to EUDAs, including the following:

Errors:

- ✓ Errors can occur at data entry, in formulas, in application logic, or with links to other applications or data sources.
- ✓ Without a sound SDLC discipline, such errors are bound to occur. This could result in poor decision making or inaccurate financial reporting.

Poor version and change control:

- ✓ EUDAs can be more difficult to control than traditional IT-developed applications.
- ✓ Even where change control policies exist, they can be difficult to enforce.

Poor documentation:

- ✓ Files that have not been properly documented may be used incorrectly after a change in ownership of the EUDA, or they may just be used improperly in general.
- ✓ Again, this can lead to unintended and undetected errors.

Lack of security:

- ✓ Unsecured files may be easily traded among users, which introduces the risk of changes to portions of data that should remain constant.
- ✓ This can lead to increased errors or might allow sensitive and confidential information to be seen by unauthorized users. An EUDA could possibly be used to perpetrate fraud or hide losses.

Lack of an audit trail:

- ✓ The ability to audit and control changes to key data is essential both for internal governance and for compliance with external regulation.
- ✓ For critical applications, managing this risk effectively is crucial, and in many instances, it requires monitoring and controlling changes at a detailed level.

Regulatory and compliance violations:

- ✓ A host of regulations deal with security and privacy for which an enterprise is responsible.

Risk of the unknown:

- ✓ The greatest operational risk with EUDA usage is in not knowing the size of the potential problem.
- ✓ The use of EUDAs is so widespread that it may be extremely difficult to assess just how many EUDAs exist, how many are used in critical business applications, how they are linked together, and where data is fed into or extracted from other IT applications.
- ✓ To quantify this risk, it is necessary to carry out a full inventory of EUDA usage and a detailed risk assessment of all businesscritical spreadsheets.

Opportunity cost:

- ✓ Scarce resources (money or employee time) may be wasted on developing these applications.

3. EUDA Security Framework

- To deal with the many risks associated with the use of EUDAs, enterprises need a comprehensive security framework that formalizes procedures for managing EUDAs and clarifies organizational policy.

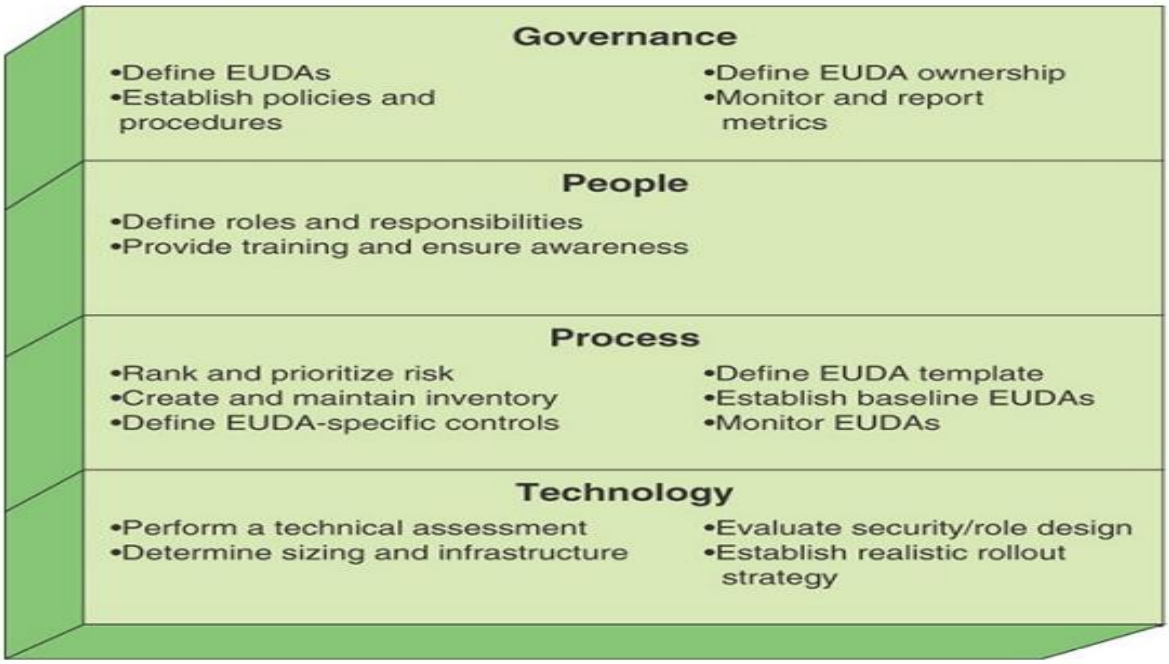


FIGURE 9.5 EUDA Security Framework

Governance

- The first set of considerations is in the area of governance. Senior executives must define what constitutes an EUDA. This involves distinguishing EUDAs from IT-developed and supported applications and specifying which types of EUDAs should be placed under management control.

People

- Proper management and control of EUDAs requires identifying the key stakeholders in the EUDA management program. Once the key stakeholders are identified, the next step is to establish the roles and responsibilities.

Process

- Management's top concerns with respect to EUDAs are the potential risks ofany given application.
  - Version control
  - Change control
  - Data integrity control
  - Access control

For each EUDA, based on its risk and impact assessment, the **EUDA owner should select the appropriate controls, which should include the following:**

- **Version control**: Helps ensure that the latest and approved version of an EUDA is used throughout the organization
- **Change control:** Helps ensure that the changes to EUDAs are appropriately tracked and reviewed
- **Data integrity control**: Helps ensure data integrity Access control: Helps ensure that only authorized users can access EUDAs and in what manner (for example, view, change, delete)
- **Availability control**: Helps ensure that EUDAs are available in the event of disaster, accidental deletion, and so on

Technology

- The organization should perform an assessment to see what sorts of tools and enablers exist or should be acquired to support the development of EUDAs.
- Specific EUDA management software tools can be deployed, or native functionality (such as Microsoft SharePoint) can be used, with various degrees of functionality available.

Authentication Mechanisms

System Access

System access is the capability that restricts access to business applications, mobile devices, systems, and networks to authorized individuals for specific business purposes. System access comprises three distinct functions:

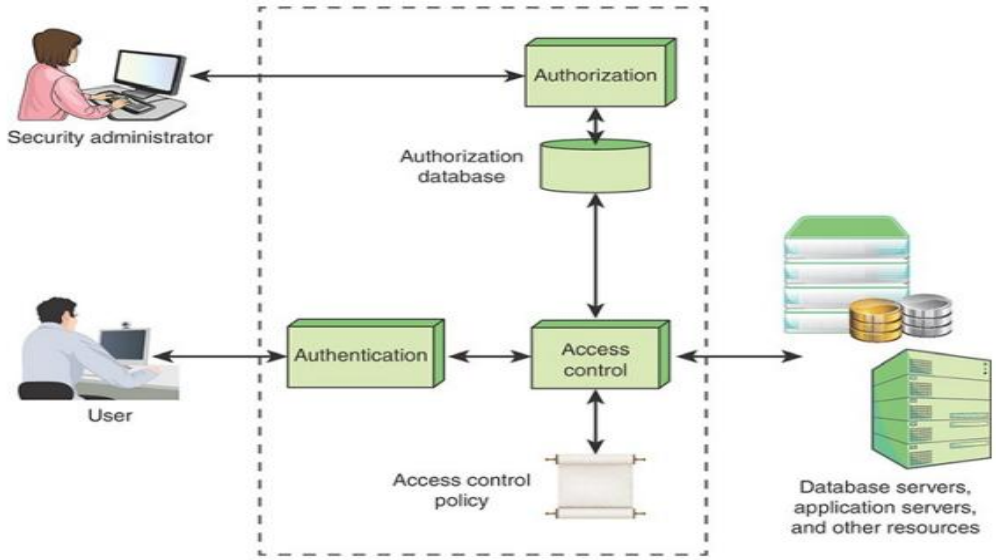
Authentication: Verifying the identity of a user, process, or device, often as a prerequisite to allowing access to resources in an information system. This function is often referred to as user authentication, to distinguish it from message authentication or data authentication.

Authorization: In the context of system access, authorization is the granting of access or other rights to a user, program, or process to access system resources. Authorization defines what an individual or program can do after successful authentication.

Access control: The process of granting or denying specific requests for accessing and using information and related information processing services and for entering

specific physical facilities. Access control ensures that access to assets is authorized and restricted based on business and security requirements.

The three functions are shown in Figure 10.1.



Authorization:

A designated security administrator is responsible for creating and maintaining the authorization database. The administrator sets these authorizations on the basis of the security policy of the organization and the roles and responsibilities of individual employees.

The process for authorizing users should include the following:

- Associating access privileges with uniquely defined individuals, for example by using unique identifiers, such as user IDs.
- Maintaining a central record of access rights granted to a user ID to access information systems and services.
- Obtaining authorization from the owner of the information system or service for the use of the information system or service. Separate approval for access rights from management may also be appropriate.
- Applying the principle of least privilege to give each person the minimum access necessary to do his or her job.

- Assigning individual access privileges for resources based on information security levels and classification of information.
- Specifying the networks and networked services to be accessed, such as files and databases.
- Defining requirements for expiration of privileged access rights.
- Ensuring that identifiers are not reused. This means deleting authorizations associated with a user ID when the individual assigned that user ID changes roles or leaves the organization.

User Authentication

- User authentication is one of the most complex and challenging security functions. There are a wide variety of methods of authentication, with associated threats, risks, and countermeasures. This section provides an overview of them.
- The following three sections look at the three general authentication factors: password, hardware token, and biometric.

The following three sections look at the three general authentication factors:

In most computer security contexts, user authentication is a fundamental building block and the primary line of defense. User authentication is the basis for most types of access control and for user accountability.

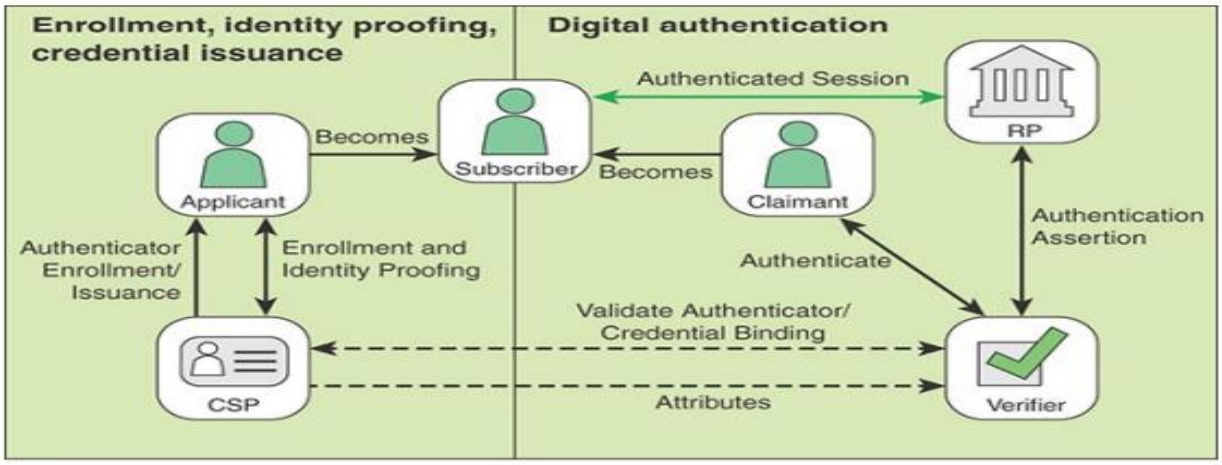
User authentication encompasses two functions:

Identification step: This step involves presenting an identifier to the security system. (Assign identifiers carefully because authenticated identities are the basis for other security services, such as access control service.)

Verification step: This step involves presenting or generating authentication information that corroborates the binding between the entity and the identifier.

A Model for Electronic User Authentication

National Institute of Standards and Technology (NIST) SP 800-63, *Digital Identity Guidelines*, defines a general model for user authentication that involves a number of entities and procedures, as shown in Figure 10.2.



CSP = credential service provider
RP = relying party

FIGURE 10.2 The NIST 800-63 Digital Identity Model

Three concepts are important in understanding this model:

Digital identity: The digital identity is the unique representation of a subject engaged in an online transaction. The representation consists of an attribute or set of attributes that uniquely describe a subject within a given context of a digital service but does not necessarily uniquely identify the subject in all contexts.

Identity proofing: This process establishes that a subject is who he or she claims to be to a stated level of certitude. This process involves collecting, validating, and verifying information about a person.

Digital authentication: This process involves determining the validity of one or more authenticators used to claim a digital identity. Authentication establishes that a subject attempting to access a digital service is in control of the technologies used to authenticate.

Successful authentication provides reasonable risk-based assurances that the subject accessing the service today is the same as the subject that previously accessed the service.

Six entities are defined in Figure 10.2:

Credential service provider (CSP): A trusted entity that issues or registers subscriber authenticators. For this purpose, the CSP establishes a digital credential for each subscriber and issues electronic credentials to subscribers. A CSP may be an independent third party or may issue credentials for its own use.

Verifier: An entity that verifies the claimant's identity by verifying the claimant's possession and control of one or two authenticators, using an authentication protocol. To do this, the verifier may also need to validate credentials that link the authenticator(s) to the subscriber's identifier and check their status.

Relying party (RP): An entity that relies upon the subscriber's authenticator(s) and credentials or a verifier's assertion of a claimant's identity, typically to process a transaction or grant access to information or a system.

Applicant: A subject undergoing the processes of enrollment and identity proofing.

Claimant: A subject whose identity is to be verified using one or more authentication protocols.

Subscriber: A party who has received a credential or authenticator from a CSP.

TABLE 10.1 Authentication Factors

Factor	Examples	Properties
Knowledge	User ID Password PIN	Can be shared Many passwords are easy to guess Can be forgotten
Possession	Smart card Electronic badge Electronic key	Can be shared Can be duplicated (cloned) Can be lost or stolen
Inherence	Fingerprint Face Iris Voice print	Not possible to share False positives and false negatives possible Forging difficult

Multifactor Authentication :

Multifactor authentication refers to the use of more than one of the authentication means in the preceding list (see Figure 10.3). The strength of an authentication system is largely determined by the number of factors incorporated by the system.

A system that requires two factors is generally stronger than a system requiring a single factor, assuming that the individual factors are reasonably strong.

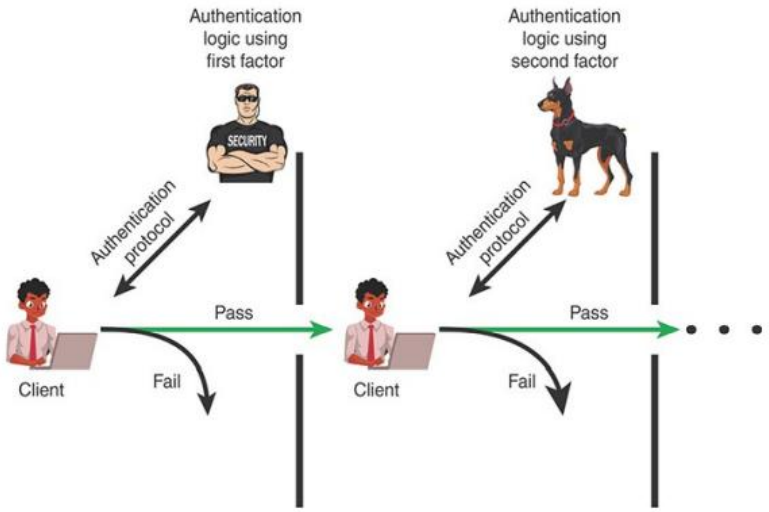


FIGURE 10.3 Multifactor Authentication

Password-Based Authentication :What you know is a widely used line of defense against intruders is a password system. Virtually all multiuser systems, network-based servers, web-based ecommerce sites, and other similar services require that a user provide not only a name or identifier (ID) but also a password.

- ✓ The system compares the password to a previously stored password for that user ID, maintained in a system password file. The password serves to authenticate the ID of the individual logging on to the system.
- ✓ In turn, the ID provides security in the following ways: The ID determines whether the user is authorized to gain access to a system. In some systems, only those who already have an ID filed on the system are allowed to gain access.
- ✓ The ID determines the privileges accorded to the user. A few users may have supervisory or "superuser" status that enables them to read files and perform functions that are especially protected by the operating system. Some systems have guest or anonymous accounts, and users of these accounts have

more limited privileges than others.

- ✓ The ID is used in what is referred to as discretionary access control. For example, by listing the IDs of the other users, a user may grant permission to them to read files owned by that user.

Possession-Based Authentication :Objects that a user possesses for the purpose of user authentication are sometimes called hardware tokens

Card Type	Defining Feature	Example
Embossed	Raised characters only, on front	Traditional credit card
Magnetic stripe	Magnetic bar on back, characters on front	Bank card
Memory	Electronic memory inside	Prepaid phone card
Smart ▪ Contact ▪ Contactless	Electronic memory and processor inside ▪ Electrical contacts exposed on surface ▪ Radio antenna embedded inside	Biometric ID card

Types of Cards Used as Possession Factor

eID Functions

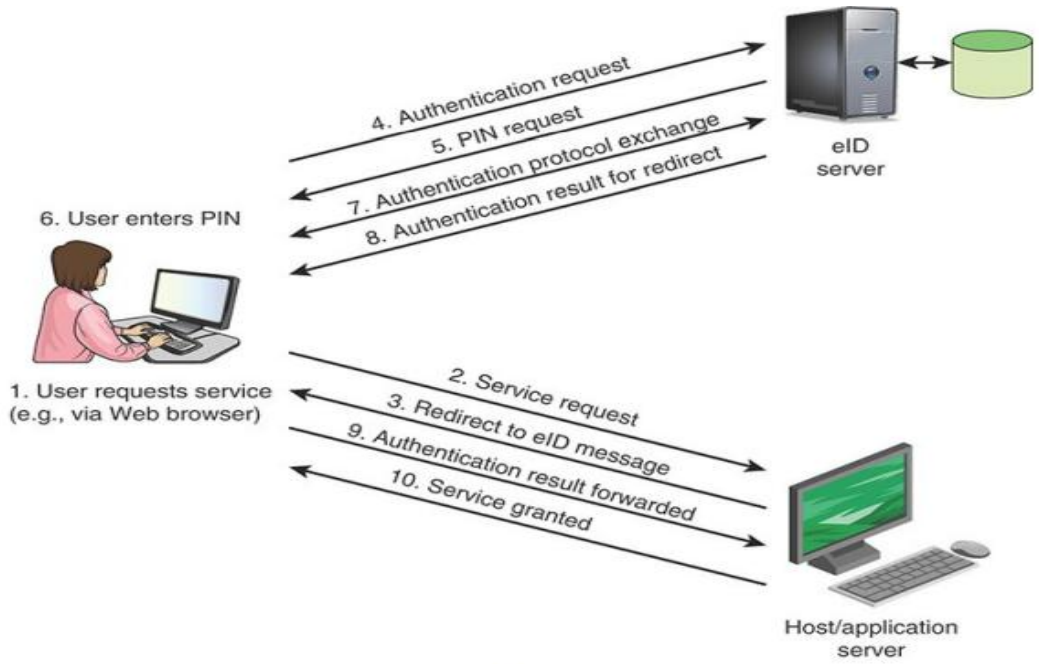


FIGURE 10.6 User Authentication with eID

Biometric Authentication :

A biometric authentication system attempts to authenticate an individual based on his or her unique physical characteristics.

These include both static characteristics (for example, fingerprints, hand geometry, facial characteristics, retinal and iris patterns) and dynamic characteristics (for example, voiceprint, signature).

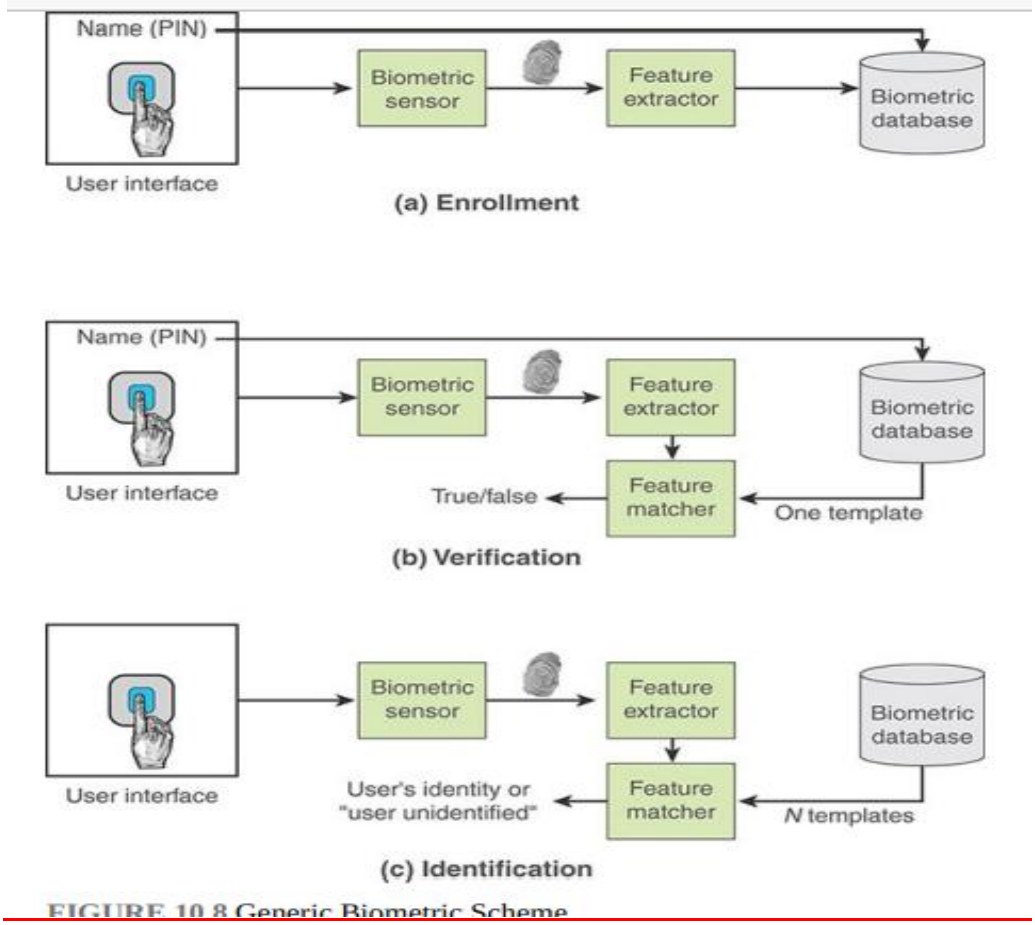


FIGURE 10.8 Generic Biometric Scheme

Access Control

This section provides an overview of important aspects of access control. It is useful to begin by defining the following terms:

Access: Ability and means to communicate with or otherwise interact with a system, to use system resources to handle information, to gain knowledge of the information the system contains, or to control system components and functions.

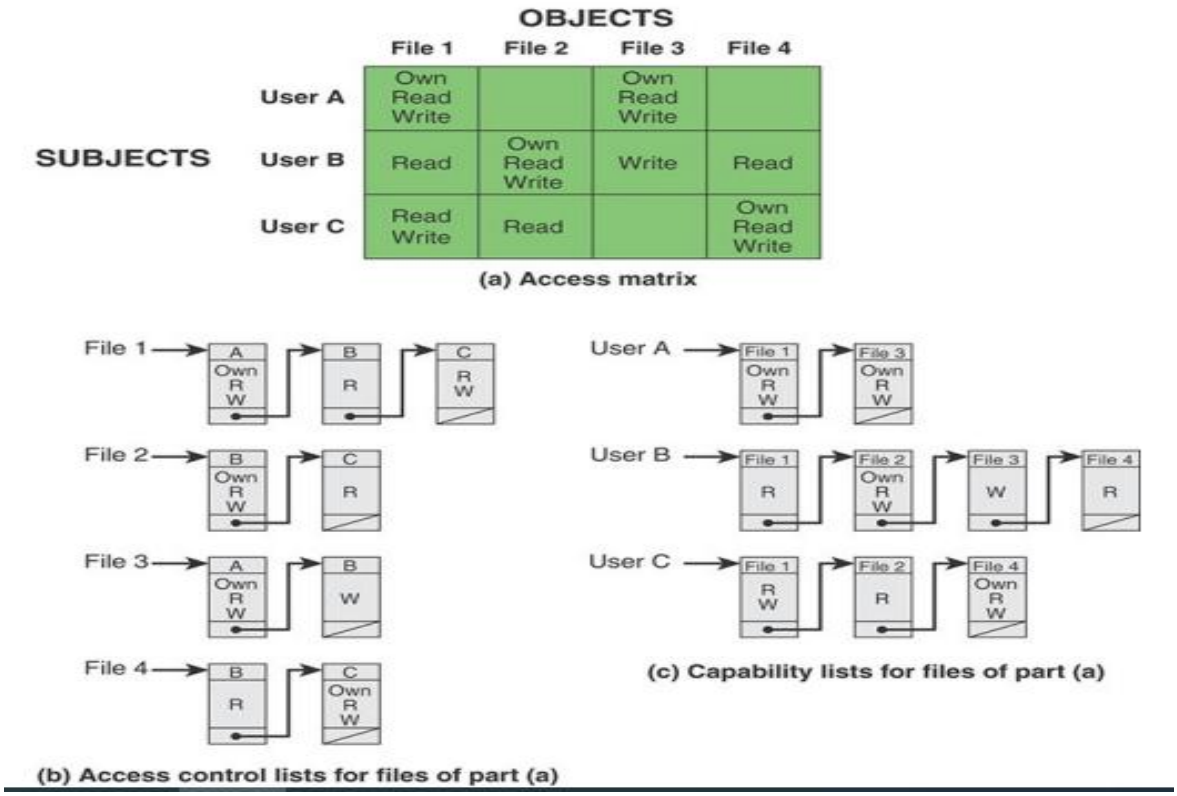
Access control: The process of granting or denying specific requests for obtaining and using information and related information processing services to enter specific physical facilities.

Access control mechanism: Security safeguards (that is, hardware and software features, physical controls, operating procedures, management procedures, and various combinations of these) designed to detect and deny unauthorized access and permit authorized access to an information system.

Access control service: A security service that protects against a system entity using a system resource in a way not authorized by the system's security policy.

Basic access control systems typically define three classes of subject, with different access rights for each class:

- **Owner**: This can be the creator of a resource, such as a file. For system resources, ownership can belong to a system administrator. For project resources, a project administrator or leader can be assigned ownership.
- **Group:** In addition to the privileges assigned to an owner, a named group of users can also be granted access rights, such that membership in the group is sufficient to exercise these access rights. In most schemes, a user may belong to multiple groups.
- **World:** The least amount of access is granted to users who are able to access the system but are not included in the categories owner and group for this resource.



Examples of Access Control Structures

Access Control Policies

Access Control Policies An access control policy dictates what types of access are permitted, under what circumstances, and by whom. Access control policies are generally grouped into the following categories:

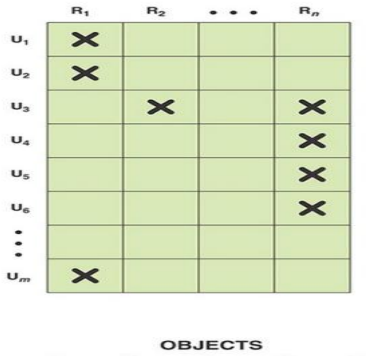
Access control policies are generally grouped into the following categories:

Discretionary access control (DAC): Access control based on the identity of the requestor and on access rules (authorizations) stating what requestors are (or are not) allowed to do. The controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject.

Mandatory access control (MAC): Access control based on comparing security labels (which indicate how sensitive or critical system resources are) with security

clearances (which indicate system entities are eligible to access certain resources). This policy is termed mandatory because an entity that has clearance to access a resource may not, just by its own volition, enable another entity to access that resource.

Role-based access control (RBAC): Access control based on user roles (that is, a collection of access authorizations a user receives based on an explicit or implicit assumption of a given role). Role permissions can be inherited through a role hierarchy and typically reflect the permissions needed to perform defined functions within an organization. A given role can apply to a single individual or to several individuals.



ROLES	OBJECTS								
	R_1	R_2	R_n	F_1	F_1	P_1	P_2	D_1	D_2
R_1	control	owner	owner control	read *	read owner	wakeup	wakeup	seek	owner
R_2		control		write *	execute			owner	seek *
•									
•									
•									
R_n			control		write	stop			

FIGURE 10.14 Access Control Matrix Representation of RBAC

Attribute-based access control (ABAC): Access control based on attributes associated with and about subjects, objects, targets, initiators, resources, or the environment. An access control rule set defines the combination of attributes under which an access takes place.

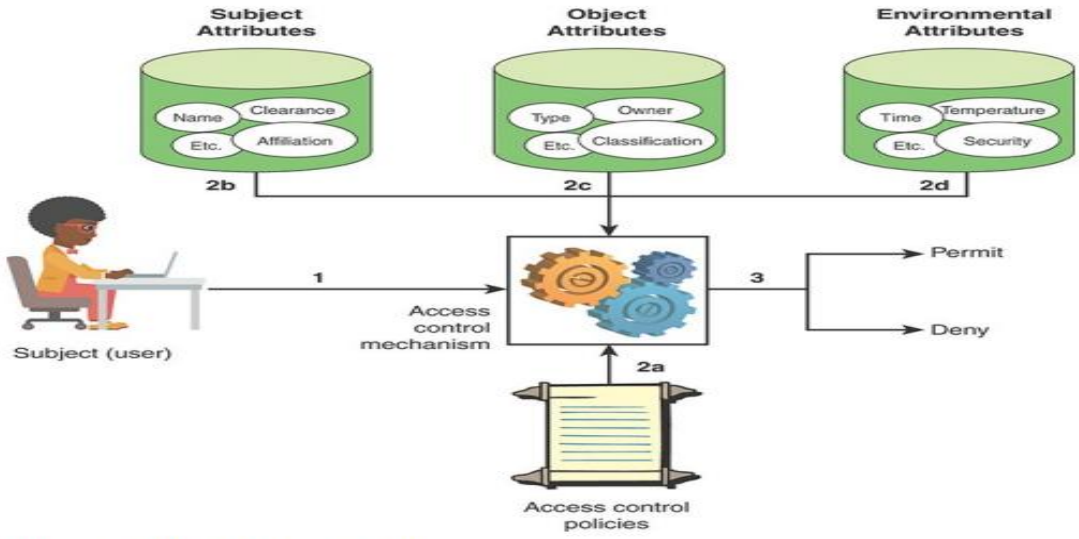


FIGURE 10.15 ABAC Scenario

TABLE 10.7 Evaluation Metrics for Access Control Systems

Administrative Properties	Enforcement Properties
Auditing Privileges/capabilities discovery Ease of privilege assignments Syntactic and semantic support for specifying AC rules Policy management Delegation of administrative capabilities Flexibilities of configuration into existing systems The horizontal scope (across platforms and applications) of control The vertical scope (between application, DBMS, and OS) of control	Policy combination, composition, and constraint Bypass Least privilege principle support Separation of Duty (SoD) Safety (confinements and constraints) Conflict resolution or prevention Operational/situational awareness Granularity of control Expression (policy/model) properties Adaptable to the implementation and evolution of AC policies
Support Properties	**Performance Properties**
Policy import and export OS compatibility Policy source management User interfaces and API Verification and compliance function support	Response time Policy repository and retrieval Policy distribution Integrated with authentication function

Virtual Servers

- Virtualization refers to a technology that provides an abstraction of the computing resources used by some software, which thus runs in a simulated

environment called a virtual machine (VM).

- Benefits arising from using virtualization include better efficiency in the use of the physical system resources than is typically seen using a single operating system instance.
- This is particularly evident in the provision of virtualized server systems.
- Virtualization also provides support for multiple distinct operating systems and associated applications on the one physical system. This is more commonly seen on client systems.

Virtualization Alternatives :

- A hypervisor is software that sits between hardware and VMs and acts as a resource broker. It allows multiple VMs to safely coexist on a single physical server host and share that host's resources.
- The virtualizing software provides abstraction of all physical resources (such as processor, memory, network, and storage resources) and thus enables multiple computing stacks, called VMs, to be run on a single physical host.

A hypervisor performs the following functions:

Execution management of VMs: This includes scheduling VMs for execution, virtual memory management to ensure VM isolation from other VMs, and context switching between various processor states. It also includes isolation of VMs to prevent conflicts in resource usage and emulation of timer and interrupt mechanisms.

Devices emulation and access control: A hypervisor emulates all network and storage (block) devices that different native drivers in VMs are expecting, mediating access to physical devices by different VMs.

Execution of privileged operations by hypervisor for guest VMs: Instead of being executed directly by the host hardware, certain operations invoked by guest operating systems, may have to be executed on its behalf by the hypervisor because of their privileged nature.

Management of VMs (also called VM life cycle management): A hypervisor configures guest VMs and controls VM states (for example Start, Pause, Stop).

Administration of hypervisor platform and hypervisor software: This involves setting parameters for user interactions with the hypervisor host as well as hypervisor software.

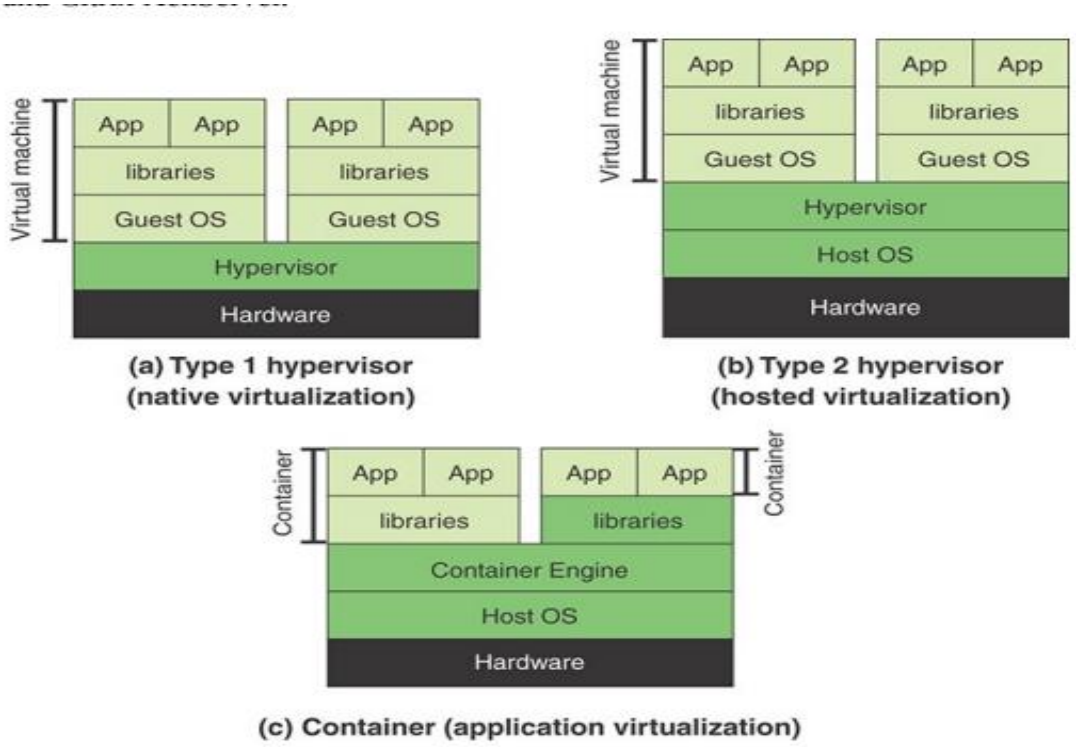


FIGURE 11.2 Comparison of Hypervisors and Containers

There are two types of hypervisors, distinguished by whether there is an operating system between the hypervisor and the host.

A type 1 hypervisor (see Figure 11.2a) is loaded as a software layer directly onto a physical server, much as an operating system is loaded; this is referred to as native virtualization.

The type 1 hypervisor directly controls the physical resources of the host. Once it is installed and configured, the server is then capable of supporting virtual machines as guests.

A type 2 hypervisor exploits the resources and functions of a host operating system and runs as a software module on top of the operating system (see Figure 11.2b); this is referred to as hosted virtualization, or nested virtualization.

A type 2 hypervisor relies on the operating system to handle all the hardware interactions on the hypervisor's behalf.

Virtualization Security Issues

- A number of security concerns that result from the use of virtualized systems, including the following: **Guest operating system isolation:**
- It is important to ensure that programs executing within a guest operating system can only access and use the resources allocated to it and cannot covertly interact with programs or data in either of the guest operating system's or in the hypervisor.

Guest operating system monitoring by the hypervisor: The hypervisor has privileged access to the programs and data in each guest operating system and must be trusted as secure from subversion and compromised use of this access.

Virtualized environment security: It is important to ensure security of the environment, particularly in regard to image and snapshot management, which attackers can attempt to view or modify.

Securing Virtualization Systems

SP 800-125, which provides guidance for appropriate security in virtualized systems, states that organizations using virtualization should do the following:

- ✓ Plan the security of the virtualized system carefully.
- ✓ Secure all elements of a full virtualization solution, including the hypervisor, guest operating systems, and virtualized infrastructure—and also maintain their security
- ✓ Ensure that the hypervisor is properly secured
- ✓ Restrict and protect administrator access to the virtualization solution

Network Storage Systems :

- Organizations make use of two broad categories of computer storage for files, databases, and other data: local and networked.
- Local storage, commonly called direct access storage (DAS), is a dedicated digital storage device attached directly to a server or PC via a cable or residing as an internal drive.
- Most users' computers and most servers have DAS. DAS creates data islands because data cannot be easily shared with other servers.
- Networked storage is a term used to describe a storage device (usually many devices paired together) that is available over a network. This kind of storage maintains copies of data across high-speed local area network (LAN) connections and is designed to back up files, databases, and other data to a central location that can be easily accessed via standard network protocols and tools.

Networked storage comes in the following topologies:

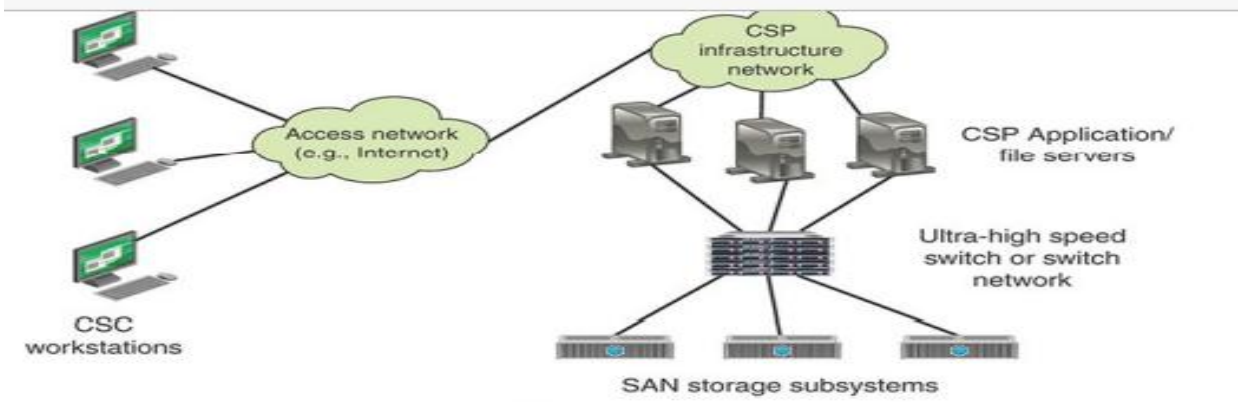
Storage area network (SAN):

- A SAN is a dedicated network that provides access to various types of storage devices, including tape libraries, optical jukeboxes, and disk arrays. To servers and other devices in the network, a SAN's storage devices look like locally attached devices.
- A disk block– based storage technology, SAN is probably the most pervasive form of storage for very large data centers and is a de facto staple for database intensive applications. These applications require shareable storage, large bandwidth, and support for the distances from rack to rack within the data center.

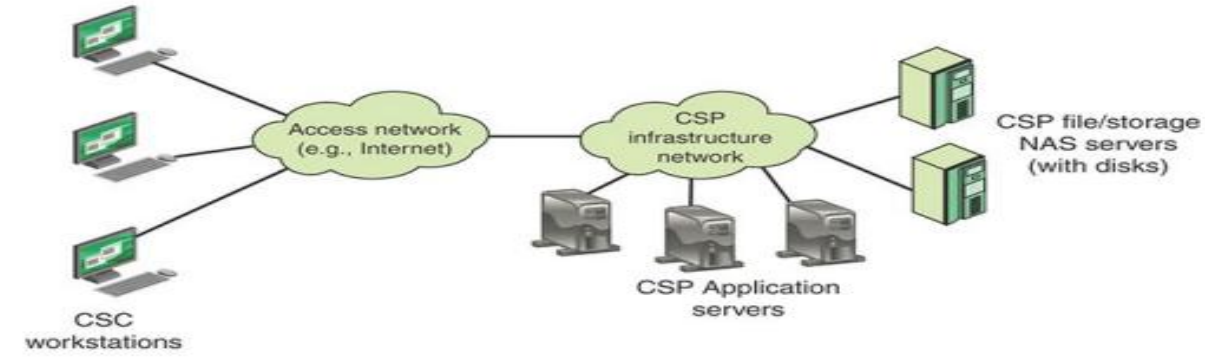
Network attached storage (NAS):

- NAS systems are networked appliances that contain one or more hard drives that are shared with multiple, heterogeneous computers. Their specialized role in networks is to store and serve files.
- NAS disk drives typically support built-in data protection mechanisms, including redundant storage containers or redundant arrays of independent disks (RAID). NAS enables file serving responsibilities to be separated from

other servers on the network and typically provides faster data access than traditional file servers.



(a) Configuration with storage area network (SAN)



(b) Configuration with network attached storage (NAS)

FIGURE 11.3 SAN and NAS in a Cloud Infrastructure

The SGP recommends the following security measures:

- ✓ Follow the system development and configuration security policies for design and configuration of network storage systems.
- ✓ Be sure that SANs and NASs are subject to standard security practices (for example, configuration, malware protection, change management, patch management).
- ✓ Ensure that the IT facility provides protection of network storage management consoles and administration interfaces.

- ✓ Store encryption information on network storage systems.
- ✓ Allow for additional security arrangements specific to NAS and SAN. Security arrangements specific to NAS and SAN depend on the type of server configuration, whether virtualization is used, and network configuration.

Network Management Concepts: Fiewall -IP Security

- This section provides an overview of network management. Let's begin by looking at the requirements for network management.
- This will provide an idea of the scope of the task to be accomplished. To manage a network, it is fundamental to know something about the current status and behavior of that network.
- Effective management requires a network management system that includes a comprehensive set of data gathering and control tools and that is integrated with the network hardware and software.

Let's look at the general architecture of a network management system.

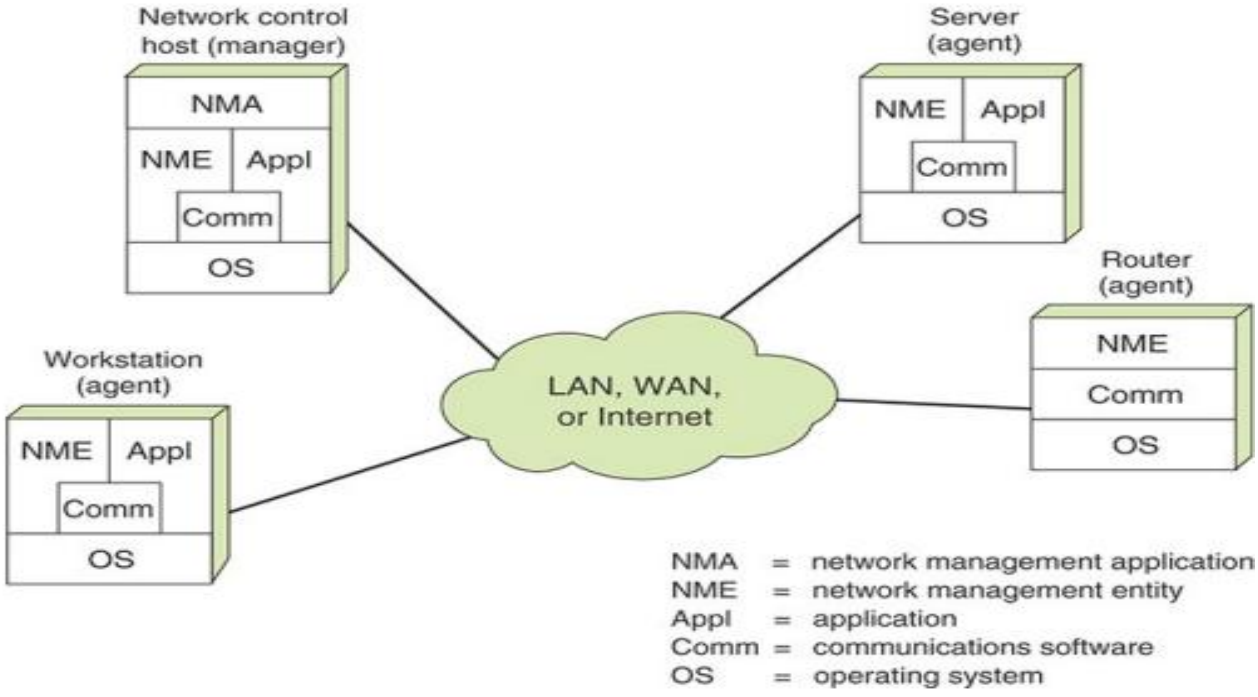


FIGURE 12.1 Components of a Network Management System

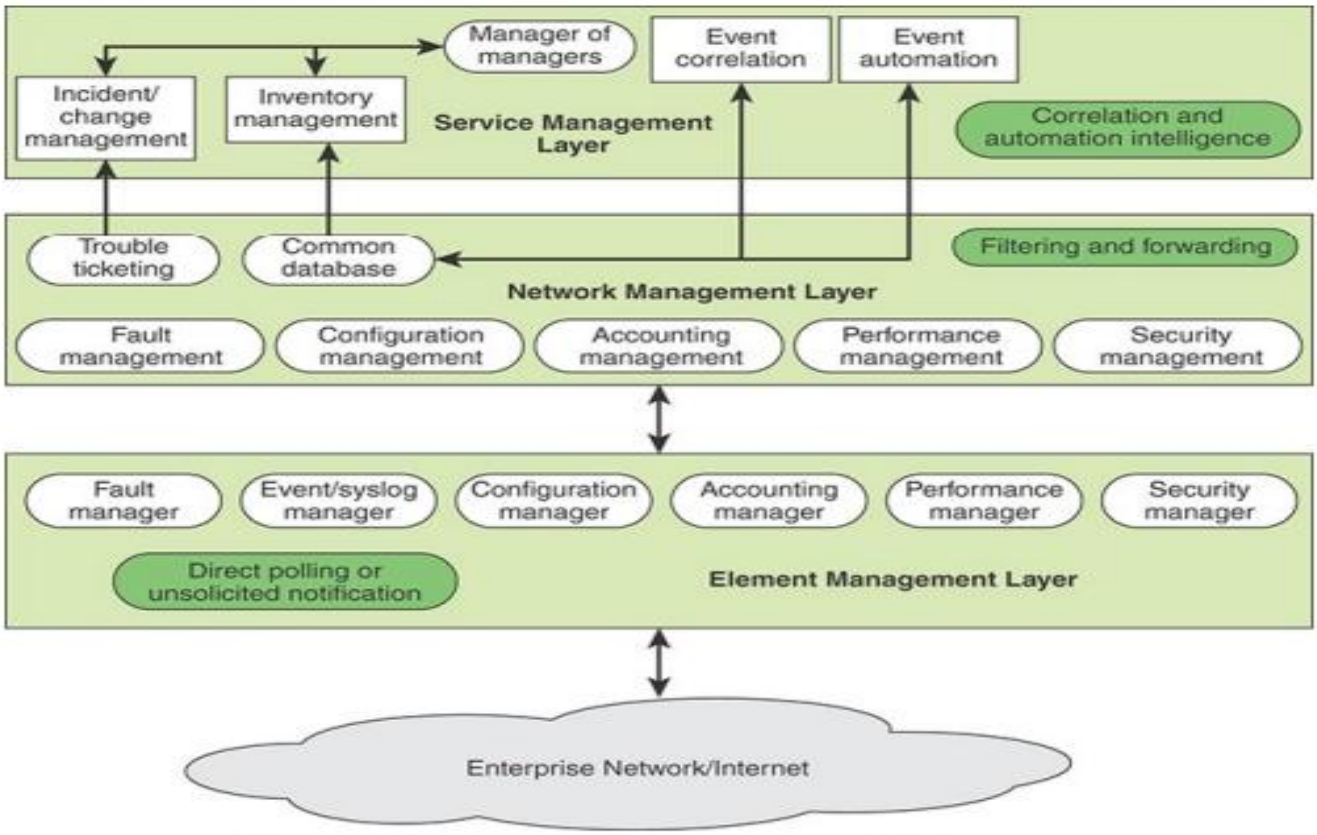


FIGURE 12.3 Network Management System Logical Architecture

Firewalls

- ✓ The firewall is an important complement to host-based security services such as intrusion detection systems. Typically, a firewall is inserted between the premises network and the Internet to establish a controlled link and to erect an outer security wall or perimeter.
- ✓ The aim of this perimeter is to protect the premises network from Internet-based attacks and to provide a single choke point where security and auditing are imposed.
- ✓ Firewalls are also deployed internally in an enterprise network to segregate portions of the network.
- ✓ A firewall provides an additional layer of defense, insulating internal systems from external networks or other parts of the internal network.

- ✓ This follows the classic military doctrine of "defense in depth," which is applicable to IT security.

Firewall Characteristics "Network Firewalls" [BELL94] lists the following design goals for a firewall:

- All traffic from inside to outside, and vice versa , must pass through the firewall. This is achieved by physically blocking all access to the local network except via the firewall. Various configurations are possible, as explained later in this chapter.

- Only authorized traffic, as defined by the local security policy, is allowed to pass. Various types of firewalls are used, and they implement various types of security policies, as explained later in this chapter. The firewall itself is immune to penetration.

- This implies the use of a hardened system with a secured operating system. Trusted computer systems are suitable for hosting a firewall and often required in government applications.

Originally, firewalls focused primarily on service control, but they have since evolved to provide all four techniques:

Service control: Determines the types of Internet services that can be accessed—inbound or outbound. The firewall can filter traffic on the basis of IP address, protocol, or port number; provide proxy software that receives and interprets each service request before passing it on; or host the server software itself, such as a web or mail service.

Direction control: Determines the direction in which particular service requests are initiated and allowed to flow through the firewall.

User control: Controls access to a service according to which user is attempting to access it. This feature is typically applied to users inside the firewall perimeter (local users). It can also be applied to incoming traffic from external users, though this requires some form of secure authentication technology, such as that provided in IP Security (IPsec).

Behavior control: Controls how particular services are used. For example, the firewall can filter email to eliminate spam or enable external access to only a portion of the information on a local web serve.

Firewalls have limitations, including the following:

- Firewalls cannot stop users from accessing malicious websites, making it vulnerable to internal threats or attacks.
- Firewalls cannot protect against the transfer of virus-infected files or software.
- Firewalls cannot prevent misuse of passwords.
- Firewalls cannot protect if security rules are misconfigured.
- Firewalls cannot protect against non-technical security risks, such as social engineering.
- Firewalls cannot stop or prevent attackers with modems from dialing in to or out of the internal network.
- Firewalls cannot secure the system which is already infected.

Types of Firewalls

- Three types of firewalls, such as **software firewalls, hardware firewalls, or both**, depending on their structure. Each type of firewall has different functionality but the same purpose. However, it is best practice to have both to achieve maximum possible protection.

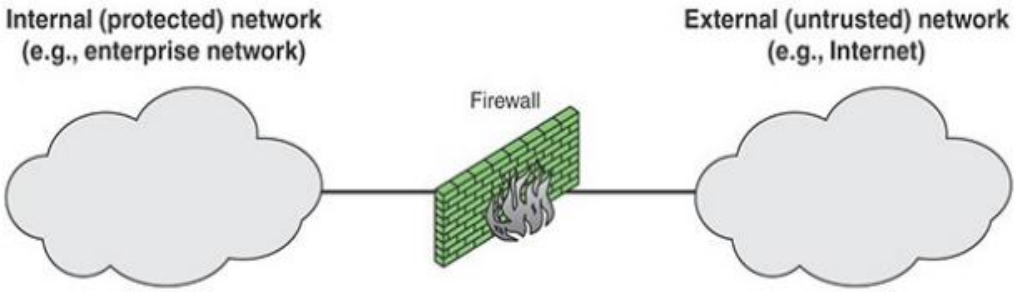
- A hardware firewall is a physical device that attaches between a computer network and a gateway. For example- a broadband router. A hardware firewall is sometimes referred to as an **Appliance Firewall**.

- On the other hand, a software firewall is a simple program installed on a computer that works through port numbers and other installed software. This type of firewall is also called a **Host Firewall**.

The following are types of firewall techniques that can be implemented as software or hardware:

- Packet-filtering Firewalls
- Circuit-level Gateways

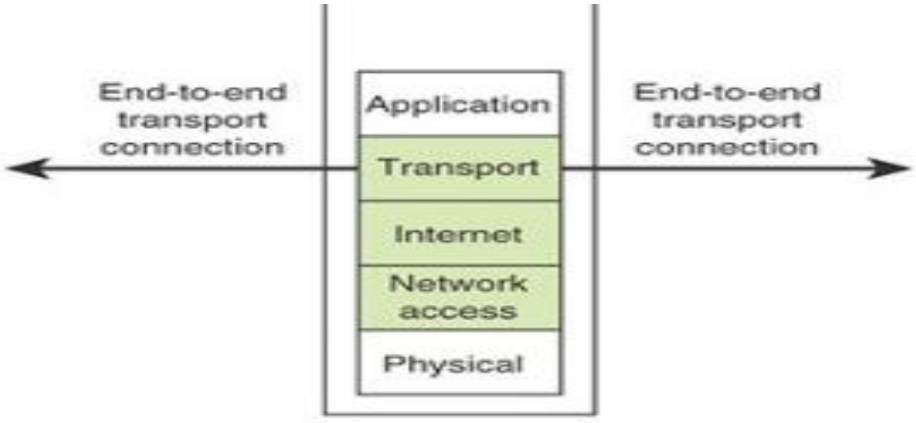
- Application-level Gateways (Proxy Firewalls)
- Stateful Multi-layer Inspection (SMLI) Firewalls
- Next-generation Firewalls (NGFW)
- Threat-focused NGFW
- Network Address Translation (NAT) Firewalls
- Cloud Firewalls
- Unified Threat Management (UTM) Firewalls



(a) General model

Packet-filtering Firewalls

A packet filtering firewall is the most basic type of firewall. It acts like a management program that monitors network traffic and filters incoming packets based on configured security rules. These firewalls are designed to block network traffic IP Protocols, an IP address, and a port number if a data packet does not match the established rule-set.

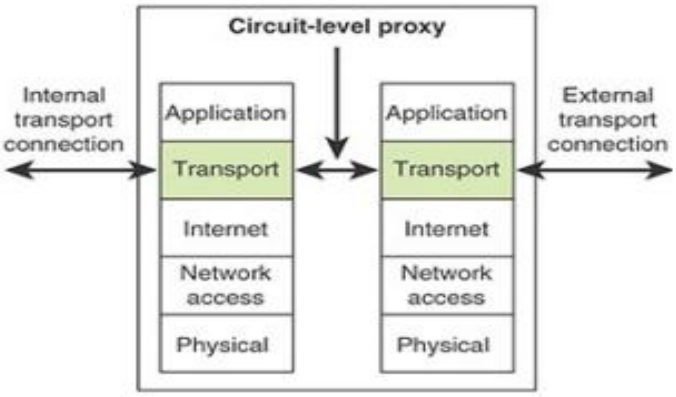


(b) Packet filtering firewall

While packet-filtering firewalls can be considered a fast solution without many resource requirements, they also have some limitations. Because these types of firewalls do not prevent web-based attacks, they are not the safest.

Circuit-level Gateways

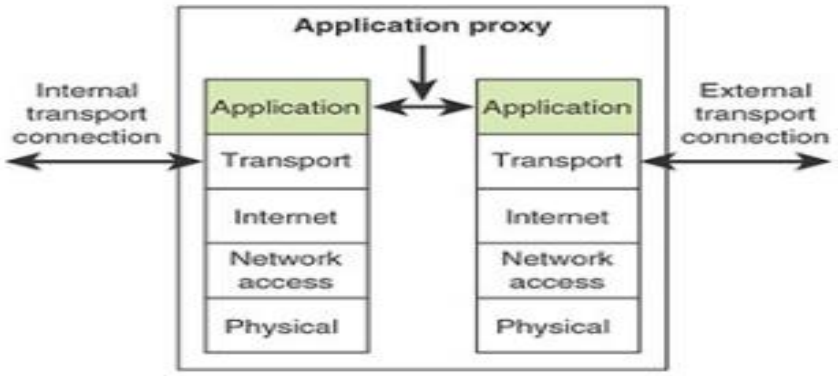
Circuit-level gateways are another simplified type of firewall that can be easily configured to allow or block traffic without consuming significant computing resources. These types of firewalls typically operate at the session-level of the OSI model by verifying **TCP (Transmission Control Protocol)**



(e) Circuit-level proxy firewall

Application-level Gateways (Proxy Firewalls)

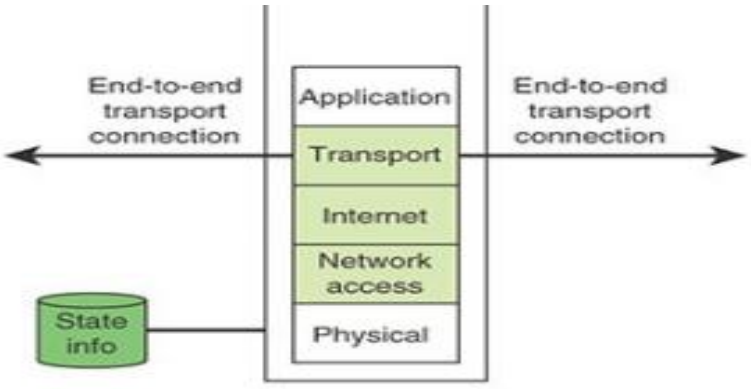
Proxy firewalls operate at the application layer as an intermediate device to filter incoming traffic between two end systems (e.g., network and traffic systems). That is why these firewalls are called **'Application-level Gateways'**.



(d) Application proxy firewall

Stateful Multi-layer Inspection (SMLI) Firewalls

- Stateful multi-layer inspection firewalls include both packet inspection technology and TCP handshake verification, making SMLI firewalls superior to packet-filtering firewalls or circuit-level gateways. Additionally, these types of firewalls keep track of the status of established connections.
- In simple words, when a user establishes a connection and requests data, the SMLI firewall creates a database (state table). The database is used to store session information such as source IP address, port number, destination IP address, destination port number, etc.



(c) Stateful inspection firewall

Next-generation Firewalls (NGFW)

- Many of the latest released firewalls are usually defined as **'next-generation firewalls'**. However, there is no specific definition for next-generation firewalls. This type of firewall is usually defined as a security device

combining the features and functionalities of other firewalls. These firewalls include **deep-packet inspection (DPI),** surface-level packet inspection, and TCP handshake testing, etc.

- In simple words, when a user establishes a connection and requests data, the SMLI firewall creates a database (state table). The database is used to store session information such as source IP address, port number, destination IP address, destination port number, etc.
- Connection information is stored for each session in the state table. Using stateful inspection technology, these firewalls create security rules to allow anticipated traffic.

Network Address Translation (NAT) Firewalls

- Network address translation or NAT firewalls are primarily designed to access Internet traffic and block all unwanted connections.
- These types of firewalls usually hide the IP addresses of our devices, making it safe from attackers.

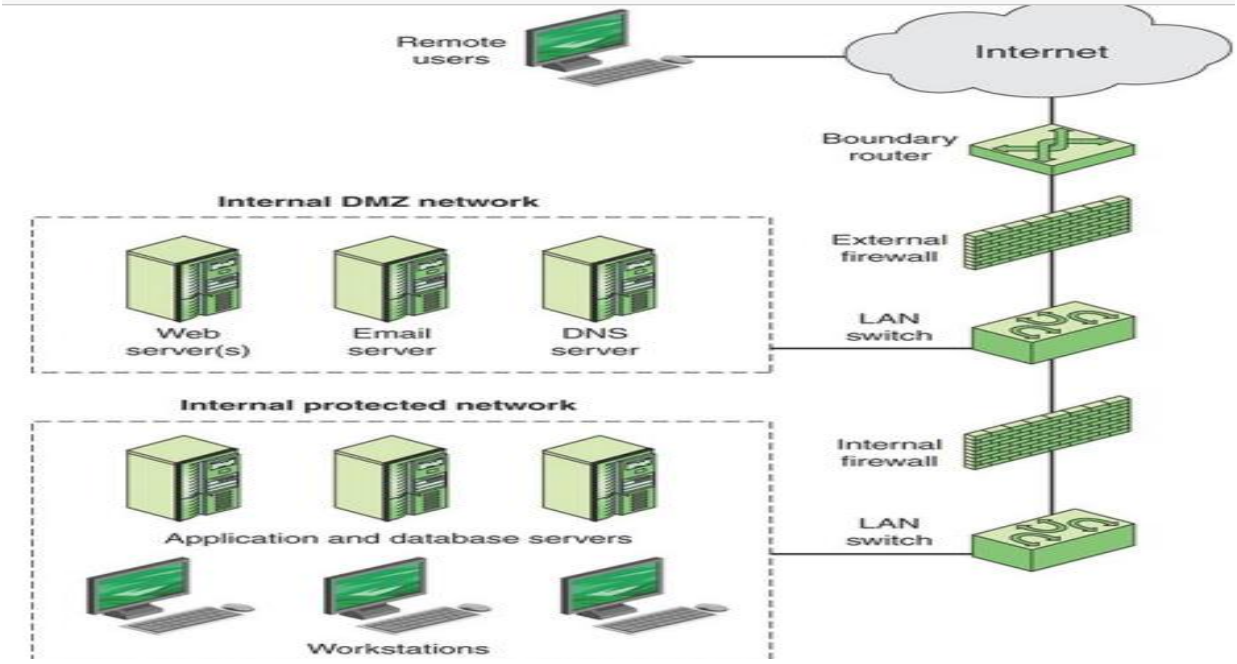
Cloud Firewalls

- Whenever a firewall is designed using a cloud solution, it is known as a cloud firewall or **FaaS (firewall-as-service)**. Cloud firewalls are typically maintained and run on the Internet by third-party vendors.
- This type of firewall is considered similar to a proxy firewall. The reason for this is the use of cloud firewalls as proxy servers. However, they are configured based on requirements.

Unified Threat Management (UTM) Firewalls

- UTM firewalls are a special type of device that includes features of a stateful inspection firewall with anti-virus and intrusion prevention support.
- Such firewalls are designed to provide simplicity and ease of use. These firewalls can also add many other services, such as cloud management, etc.

.



In this type of configuration, internal firewalls serve three purposes

- ❖ An internal firewall adds more stringent filtering capability, compared to the external firewall, in order to protect enterprise servers and workstations from external attack.

- ❖ An internal firewall provides two-way protection with respect to the DMZ. First, the internal firewall protects the remainder of the network from attacks launched from DMZ systems. Such attacks might originate from worms, rootkits, bots, or other malware lodged in a DMZ system. Second, an internal firewall protects the DMZ(demilitarized zone) systems from attack from the internal protected network.

- ❖ Multiple internal firewalls are used to protect portions of the internal network from each other. For example, firewalls are configured so that internal servers are protected from internal workstations and vice versa. A common practice is to place the DMZ on a different network interface on the external firewall from that used to access the internal networks.

IPSECURITY

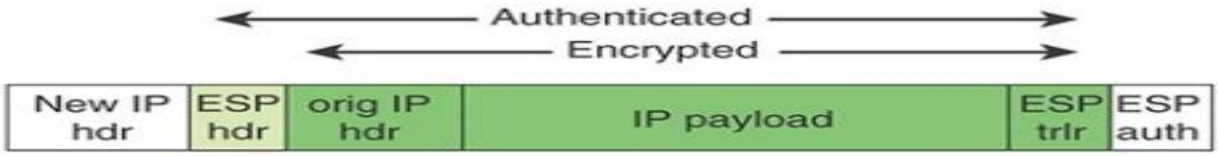
- IPsec is a set of Internet standards that augment both versions of IP that are

in current use (IPv4 and IPv6) with security features.

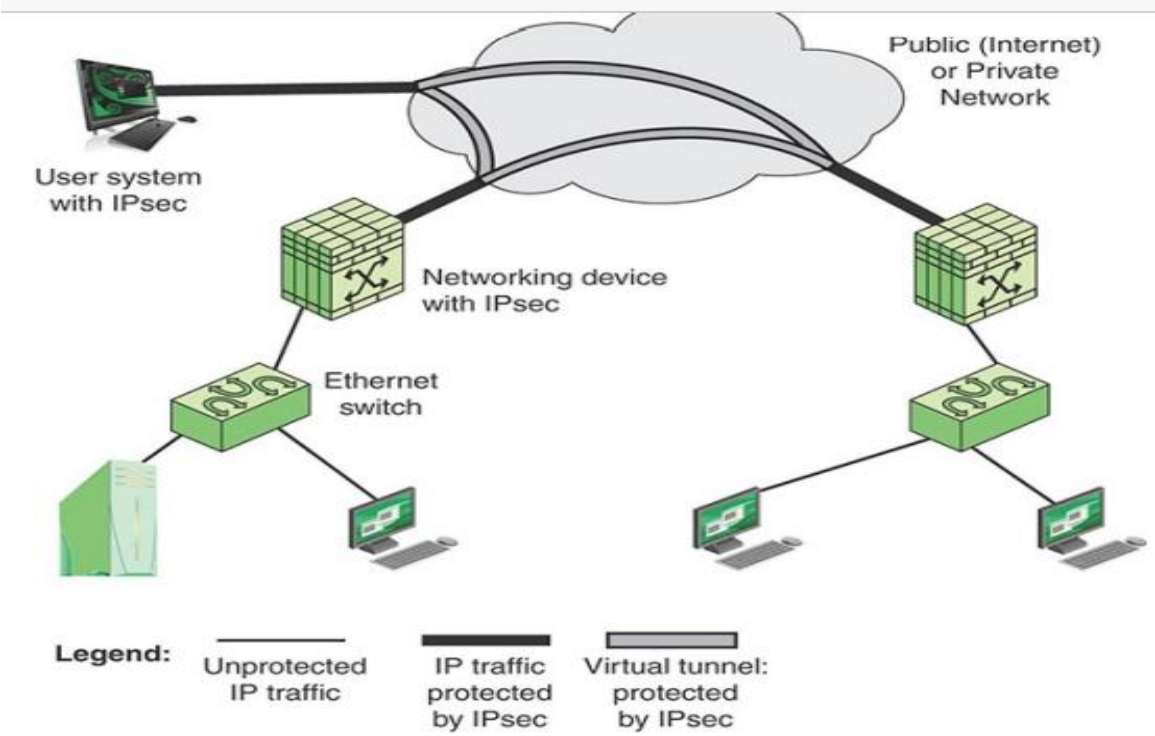
- The principal feature of IPsec is that it encrypts and/or authenticates all traffic at the IP level. Thus, all distributed applications—including remote logon, client/server, email, file transfer, web access, and so on—are secured.

IPsec provides three main facilities:

- ❖ An authentication-only function referred to as Authentication Header (AH),
- ❖ a combined authentication/encryption function called Encapsulating Security Payload (ESP), and
- ❖ a key exchange function.



(a) Tunnel-mode format



(b) Example configuration

FIGURE 12.6 An IPsec Tunnel Mode Scenario

- ✓ Figure 12.6a shows a simplified packet format for an IPsec option known as tunnel mode, using ESP and a key exchange function.
- ✓ Figure 12.6b shows a typical IPsec usage scenario. An organization maintains local area networks (LANs) at dispersed locations. Insecure IP traffic is conducted on each LAN.
- ✓ For traffic offsite, through some sort of private or public WAN, IPsec protocols are used. These protocols operate in networking devices, such as a router or firewall, that connect each LAN to the outside world.
- ✓ The IP sec networking device Typically encrypts and compresses all traffic going into the WAN and decrypts and decompresses traffic coming from the WAN; these operations are transparent to workstations and servers on the LAN.
- ✓ Secure transmission is also possible with individual users who dial in to the WAN. Such user workstations must implement the IPsec protocols to provide security.

- ✓ Electronic Communications Often the focus of enterprise security is protecting stored information and server facilities, as well as client/server communication from the wide variety of threats on the landscape.
- ✓ It is important not to overlook security related to electronic communications that may not involve server or database access but that is between individuals.

Electronic communications

Often the focus of enterprise security is protecting stored information and server facilities, as well as client/server communication from the wide variety of threats on the landscape.

It is important not to overlook security related to electronic communications that may not involve server or database access but that is between individuals.

Email:

- ✓ **I**t is useful to have a basic grasp of the Internet mail architecture, as defined in RFC 5598, Internet Mail Architecture. At its most fundamental level, the Internet mail architecture consists of a user world, in the form of message user agents (MUAs), and a transfer world, in the form of the Message Handling System (MHS), which is composed of message transfer agents (MTAs).

This section looks at four types of electronic communications that need to be protected.

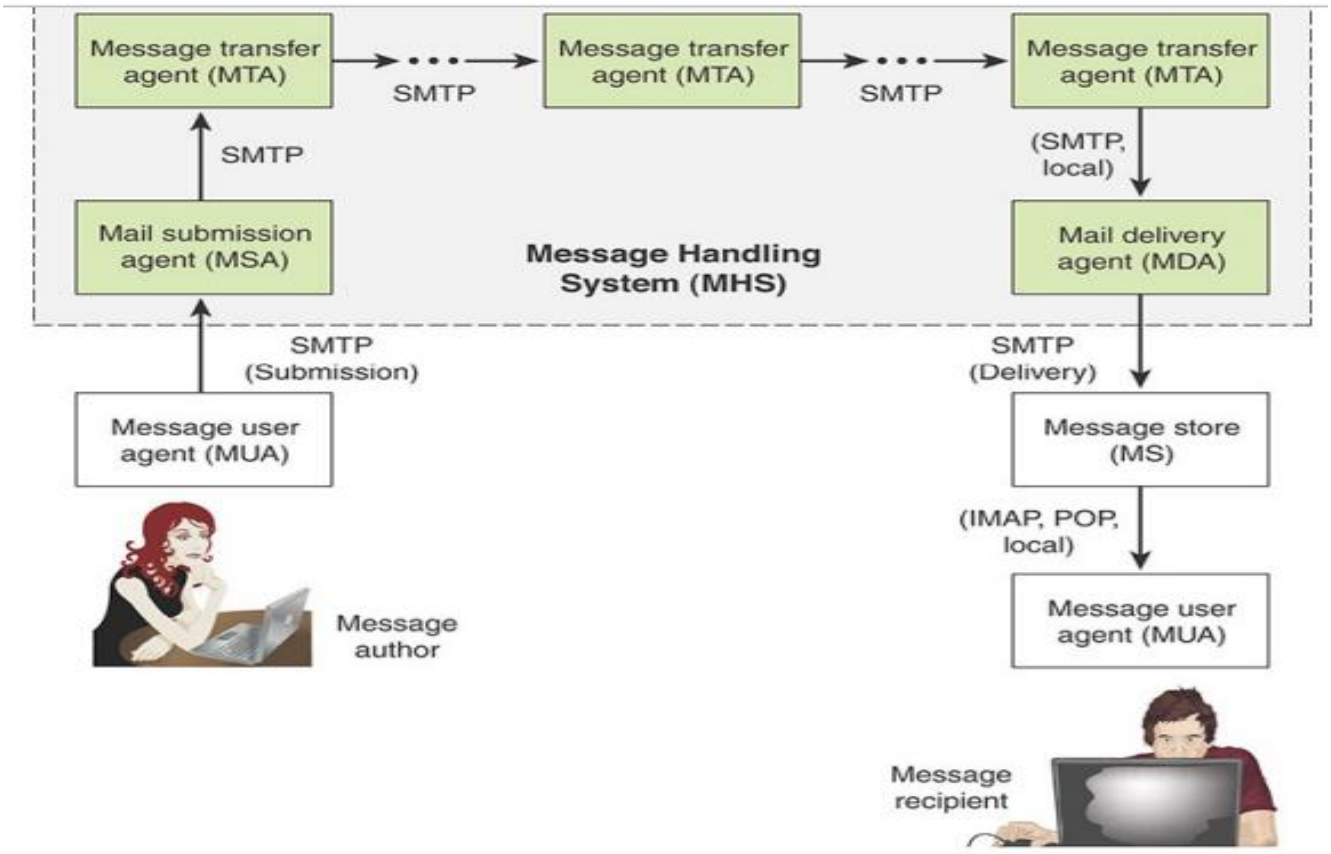


FIGURE 12.9 Function Modules and Standardized Protocols Used Between Them in the Internet Mail Architecture

OWASP

What is OWASP?

- The Open Web Application Security Project (OWASP) is an open, online community that creates methodologies, tools, technologies and guidance on how to deliver secure web applications.
- It is an international collaborative initiative comprised of both individuals and corporations.
- The project aims to standardise security approaches in web development and spread associated knowledge.

OWASP ZAP

What is OWASP ZAP?

OWASP ZAP (ZAP) is one of the world's most popular free security tools and is actively maintained by hundreds of international volunteers. It can help to find security vulnerabilities in web applications. It's also a great tool for experienced pen testers and beginners.

ZAP can scan through the web application and detect issues related to:

- SQL injection
- Broken Authentication
- Sensitive data exposure
- Broken Access control
- Security misconfiguration
- Cross Site Scripting (XSS)
- Insecure Deserialization
- Components with known vulnerabilities
- Missing security headers

Why we chose OWASP ZAP?

- ✓ As it is designed to be used by people with a wide range of pen testing experience, it was ideal for our team who were new to penetration testing.
- ✓ ZAP is a free open-source tool which is easy to setup and use. As it is used by the wider community, there is a lot of help available online through the ZAP blog and other articles to help you setup and use the tool.
- ✓ ZAP is cross platform i.e. you can install it in Windows, Linux or Mac OS.
- ✓ ZAP can be run in a Docker container, which suited our project tech stack. Also, its functionality is scalable with many diverse extensions published on GitHub.

- ✓ ZAP Jenkins plugin can be setup to run the scans as part of CI / CD pipelines.

How to Use the OWASP ZAP Vulnerability Scanner to Plan A Vulnerability Test?

The OWASP ZAP tool captures the request just before hitting the network, which allows to analyze the various parameters, header values in the request. It then explores and attacks it to find security issues that need redressal. In the process, it records the requests and responses on every page and sends out alerts when it encounters an issue.

Below are the steps on how to initiate the OWASP ZAP penetration testing using a Windows system:

1. Starting the OWASP ZAP UI

To start a vulnerability test using the OWASP ZAP web application scanner, you need to download the tool and install it.

It is platform agnostic and hence you can set it up on either Windows, Mac OS, or Linux. However, if you are using Windows or Linux, you should also have Java 8+ already installed on your system.

After installation, click on the OWASP ZAP icon on your desktop. Now, click on the 'start' button on the start-up dialog box, to launch the ZAP UI.

Upon running the interface, a pop-up window will ask if you want to save the session. For a new session, choose the default option 'No, I do not want to persist the session'.

2. Initiating a Scan

- ✓ You can start scanning your web application by using the QuickStart automated scan. With QuickStart, you can scan an application just by entering its URL and pushing the 'attack' button, which makes it quite simple to execute.
- ✓ You can use passive scanning as well, which is one of the most interesting features of the OWASP ZAP scanner. The tool records all the requests received by the application and its responses.

- ✓ It then issues an alert if any anomaly is observed with either the request or the response. However, it cannot detect an issue such as an SQL injection attack.
- ✓ Instead, you can use the active scanning feature to find out the vulnerabilities not found through passive scanning.
- ✓ During an active scan, ZAP can simulate a real attack against some specific areas of your application to understand the response.

Additionally, the ZAP scanner can be used in different modes like:

- The standard mode which allows you to use every feature of the tool
- You can also use attack mode to run active scans.
- The safe mode turns off the harmful features while the protected mode lets you scan chosen websites within a defined scope.

UNIT VI **TECHNICAL SECURITY**

Supply ChainManagement -Cloud Security-Security Architecture-Malware Protection-Intrusion Detection-Digital Rights Management-Cryptographic Techniques-Threat and Incident Management-Vulnerability Management-Security Event Management-Forensic Investigations- Local Environment Management -Business Continuity.

Supply ChainManagement

- A supply chain was defined as the network of all the individuals, organizations, resources, activities, and technology involved in the creation and sale of a product, from the delivery of source materials from the supplier to the manufacturer, through to its eventual delivery to the end user.
- In this traditional use, the term applies to the entire chain of production and use of physical products. The chain can link a number of entities, beginning with raw materials suppliers, through manufacturers, wholesalers, retailers, and consumers.
- More recently the term supply chain has been used in connection with information and communications technology (ICT).

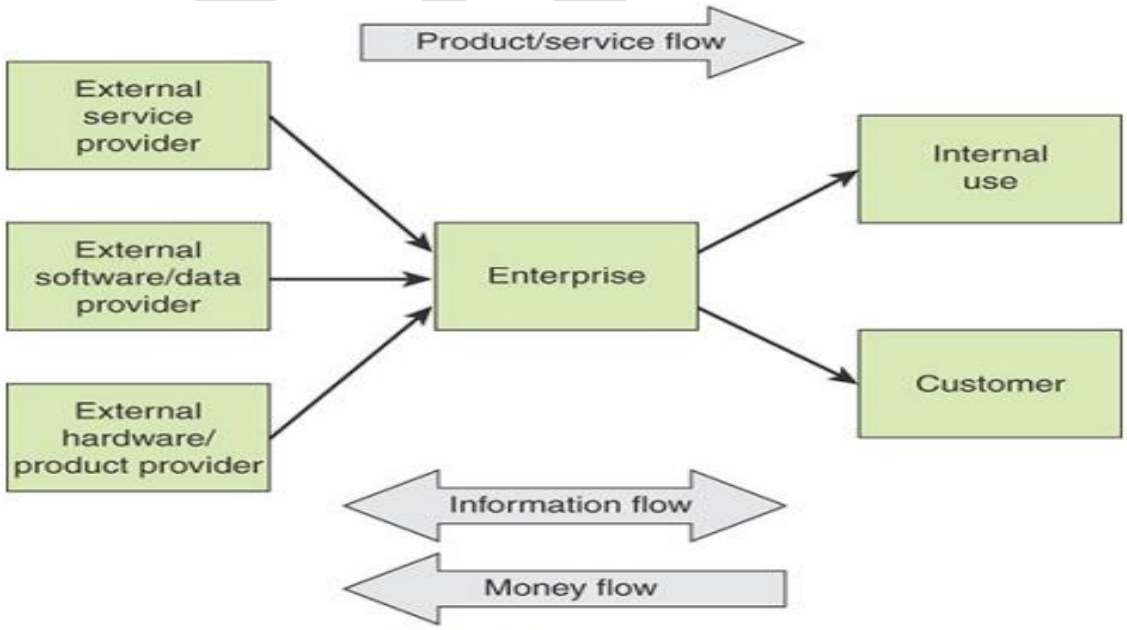


FIGURE 13.1 Supply Chain Flows

An enterprise procures the following from external sources:

- ✓ **Services:** Examples include cloud computing services, data center services, network services, and external auditing services.
- ✓ **Software/data:** Examples include operating system and application software and databases of information, such as threat information.
- ✓ **Hardware/products**: Examples include computer and networking equipment.

Indicates three types of flows associated with a supply chain:

- ✓ **Product/service flow**: A key requirement is a smooth flow of an item from the provider to the enterprise and then on to the internal user or external customer. The quicker the flow, the better it is for the enterprise, as it minimizes the cash cycle.
- ✓ **Information flow**: Information flow comprises the request for quotation, purchase order, monthly schedules, engineering change requests, quality complaints, and reports on supplier performance from the customer side to the supplier. From the producer's side to the consumer's side, the information flow consists of the presentation of the company, offer, confirmation of purchase order, reports on action taken on deviation, dispatch details, report on inventory, invoices, and so on.
- ✓ **Money flow**: On the basis of the invoice raised by the producer, the clients examine the order for correctness. If the claims are correct, money flows from the clients to the respective producer. Flow of money is also observed from the producer side to the clients in the form of debit notes.

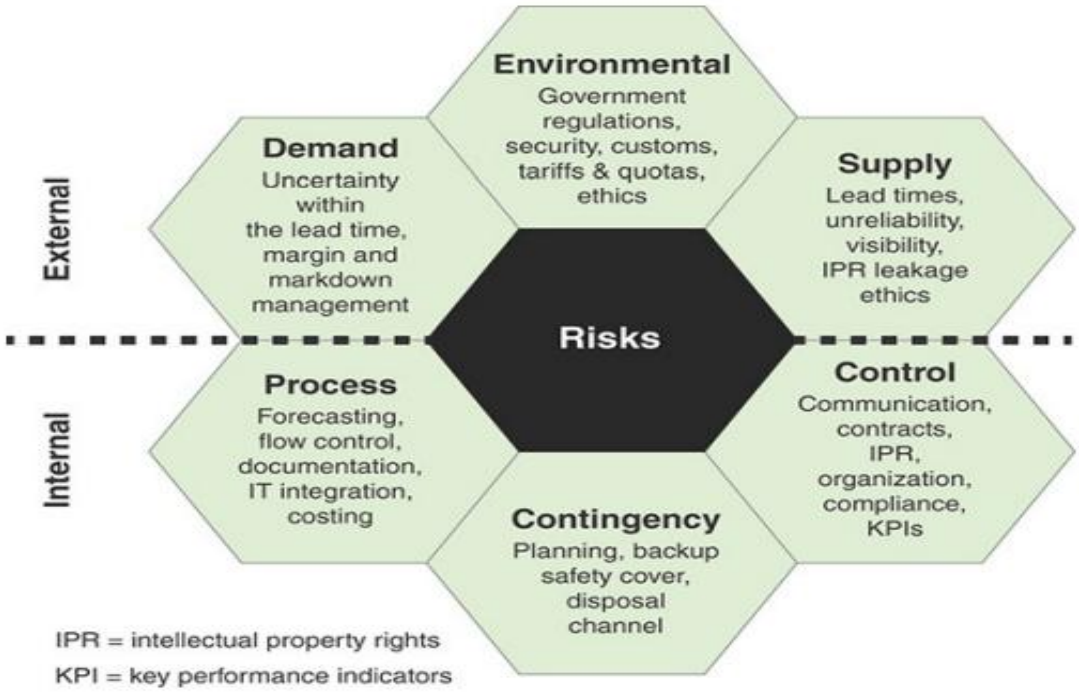
FIGURE 13.2 Supply Chain Management l

The 0068==xyz.xcements of supply chain management include the following:

- ✓ **Demand management**: This function recognizes all demands for goods and services to support the marketplace. It involves prioritizing demand when supply is lacking. Proper demand management facilitates the planning and use of resources for profitable business results.
- ✓ **Supplier qualification**: This function provides an appropriate level of confidence that suppliers, vendors, and contractors are able to supply consistent quality of materials, components, and services in compliance with customer and regulatory requirements. An integrated supplier qualification process should also identify and mitigate the associated risks of materials, components, and services.
- ✓ **Supplier negotiation**: In this process of formal communication, two or more people come together to seek mutual agreement on an issue or issues. Negotiation is particularly appropriate when issues besides price are important for the buyer or when competitive bidding does not satisfy the buyer's requirements on those issues.

- ✓ **Sourcing, procurement, and contract management**: Sourcing refers to the selection of a supplier or suppliers. Procurement is the formal process of purchasing goods or services.
- ✓ **Logistics and inventory control:** In this context, logistics refers to the process of strategically managing the procurement, movement, and storage of materials, parts, and finished inventory (and the related information flows) through the organization and its marketing channels. Inventory control is the tracking and accounting of procured items.
- ✓ **Invoice, reconciliation, and payment**: This is the process of paying for goods and services.

- ✓ **Supplier performance monitoring**: This function includes the methods and techniques for collecting information to be used to measure, rate, or rank supplier performance on a continuous basis. Performance refers to the ability of the supplier to meet stated contractual commitments and enterprise objectives.



✓ FIGURE 13.5 Supply Chain Risk Areas

The external risks are as follows:

- ✓ **Demand**: Refers to disturbances to the flow of product, information, or cash from within the supply chain between the organization and its market. For example, disruptions in the cash resource within the supply chain needed to pay the organization can have a major impact on the operating capability of organizations.

- ✓ **Supply:** The upstream equivalent of demand risk; it relates to potential or actual disturbances to the flow of product or information from within the supply chain between the organization and its suppliers. In a similar way to demand risk, the disruption of key resources coming into the organization can have a significant impact on the organization's ability to perform

- ✓ **Environmental:** The risk associated with external and, from the firm's perspective, uncontrollable events. The risks can impact the firm directly or through the firm's suppliers and customers. Environmental risk is broader

than just natural events like earthquakes or storms. It also includes, for example, changes created by governing bodies such as changes in legislation or customs procedures, as well as changes in the competitive climate.

The internal risks are as follows:

- ✓ **Processes**: The sequences of value-adding and managerial activities undertaken by the firm. Process risk relates to disruptions to key business processes that enable the organization to operate. Some processes are key to maintaining the organization's competitive advantage, while others can underpin the organization's activities.
- ✓ **Controls:** The rules, systems, and procedures that govern how an organization exerts control over processes and resources. In terms of the supply chain, controls may relate to order quantities, batch sizes, safety stock policies, and so on, plus the policies and procedures that govern asset and transportation management. Control risk is therefore the risk arising from the application or misapplication of these rules.
- ✓ **Contingency:** The existence of a prepared plan and the identification of resources that are mobilized in the event of a risk being identified. Contingency plans may encompass inventory, capacity, dual sourcing, distribution and logistics alternatives, and backup arrangements.

cloud security

What is cloud security?

Cloud security is the set of control-based security measures and technology protection, designed to protect online stored resources from **leakage, theft**, and **data loss**. Protection includes data from **cloud infrastructure, applications**, and **threats.** Security applications uses a software the same as **SaaS (Software as a Service)**

model.

Security Considerations for Cloud Computing

The following key issues that need to be addressed when an organization moves data and/or applications into the cloud:

Confidentiality and privacy: An organization has commitments to its employees and customers in the areas of data confidentiality and privacy. Further, any breach of confidentiality or privacy can have adverse business impacts. Finally, regulations and legal restrictions apply. Placing data in the cloud introduces new risks that must be assessed.

Data breach responsibilities: Placing data and services in the cloud amplifies concerns about data breaches, yet security is not under the direct control of the customer.

The following are some issues in this regard:

- ✓ **Responsibility for notifying**: Data breach generally carries with it an obligation to notify. Who is responsible for notification (customer, vendor, third party) and how quickly?
- ✓ **Risks to intellectual property**: Risks include authorization, terms and conditions that (inappropriately) assert ownership over intellectual property held by third parties, and weakening of ability for organizations to assert "work made for hire" for creations that are developed "without use of organizational resources.
- ✓ **Export controls**: Does the vendor house data at foreign sites? Are the systems managed by foreign nationals?

E-discovery: Institutions and their legal counsel can be obligated to keep records needed for legal discovery. But these records are not under direct organizational control; the organization no longer has the record in the same way that it formerly did. How does one handle discovery in this externalized infrastructure?

Risk assessment: To perform effective risk assessment, the customer must have considerable information about the security policies and controls in effect at the cloud service provider.

Business continuity: Plans are needed to deal with the suspension or termination of the cloud service. The customer needs to have the portability capability to move data to a different cloud service provider.

Legal issues: Legal risks and obligations must be clarified and documented.

Threats for Cloud Service Users

The use of cloud services and resources introduces a novel set of threats to enterprise cybers ecurity

Responsibility ambiguity: The enterprise-owned system relies on services from the cloud provider. The level of the service provided (SaaS, PaaS, IaaS) determines the magnitude of resources that are offloaded from IT systems on to the cloud systems. Regardless of the level of service, it is difficult to define precisely the security responsibilities of the customer and those of the cloud service provider. If there is any ambiguity, this complicates risk assessment, security control design, and incident response.

Loss of governance: The migration of a part of the enterprises IT resources to the cloud infrastructure gives partial management control to the cloud service provider. The degree of loss of governance depends on the cloud service model (SaaS, PaaS, IaaS). In any case, the enterprise no longer has complete governance and control of IT operations.

Loss of trust: It is sometimes difficult for a cloud service user to assess the provider's trust level due to the black-box feature of the cloud service. There is no way to obtain and share the provider's security level in a formalized manner.

Service provider lock-in: A consequence of the loss of governance could be a lack of freedom in terms of how to replace one cloud provider with another. An example of a difficulty in transitioning is if a cloud provider relies on nonstandard hypervisors or virtual machine image format and does not provide tools to convert virtual machines to a standardized format.

Nonsecure cloud service user access: As most of the resource deliveries are through remote connections, unprotected application programming interfaces (APIs) (mostly management APIs and PaaS services) are among the easiest attack vectors. Attack methods such as phishing, fraud, and exploitation of software vulnerabilities pose significant threats.

Lack of asset management: The cloud service user may have difficulty in assessing and monitoring asset management by the cloud service provider. Key elements of interest include location of sensitive asset/information, degree of physical control for data storage, reliability of data backup (data retention issues), and countermeasures for business continuity and disaster.

Data loss and leakage: This threat can be strongly related to the preceding item. However, loss of an encryption key or a privileged access code brings serious problems to cloud service users.

Risk Evaluation

It is useful to have a detailed questionnaire for performing risk evaluation for cloud services. The Information Security Council developed a template to be used for this purpose. **The template includes the questions in the following areas:**

- ✓ High-level description
- ✓ Authentication Authorization-logical access
- ✓ control Data security

- ✓ Recoverability
- ✓ Operational controls
- ✓ Incident response
- ✓ Application security
- ✓ Testing and validation

Malware Protection Activities:

Activities Malicious software (malware) is perhaps the most significant security threat to organizations

What is Malware?

As software designed to interfere with a computer's normal functioning, malware is a blanket term for viruses, trojans, and other destructive computer programs threat actors use to infect systems and networks in order to gain access to sensitive information.

Malware Definition

- Malware (short for "malicious software") is a file or code, typically delivered over a network, that infects, explores, steals or conducts virtually any behavior an attacker wants. And because malware comes in so many variants, there are numerous methods to infect computer systems.

Though varied in type and capabilities, malware usually has one of the following objectives:

- Provide remote control for an attacker to use an infected machine.
- Send spam from the infected machine to unsuspecting targets.
- Investigate the infected user's local network.
- Steal sensitive data.

Types of Malware:

Although the terminology related to malware is not consistent, the following list provides a useful guide to the various types of malware:

- ✓ **Adware:** Advertising that is integrated into software. It can result in pop-up ads or redirection of a browser to a commercial site.
- ✓ **Auto-rooter**: A malicious hacker tool used to break in to new machines remotely. Backdoor (trapdoor): Any mechanisms that bypasses a normal security check; it may allow unauthorized access to functionality.
- ✓ **Exploit:** Code specific to a single vulnerability or set of vulnerabilities. Downloader: A program that installs other items on a machine that is under attack. Usually, a downloader is sent in an email message.
- ✓ **Droppe**r: A malware installer that surreptitiously carries viruses, backdoors, and other malicious software to be executed on the compromised machine. Droppers don't cause harm directly but deliver a malware payload onto a target machine without detection.
- ✓ **Polymorphic dropper**: Also called a polymorphic packer, a software exploit tool that bundles several types of malware into a single package, such as an email attachment, and can force its "signature" to mutate over time, making it difficult to detect and remove.
- ✓ **Flooder:** A tool used to attack networked computer systems with a large volume of traffic to carry out a denial-of-service (DoS) attack.
- ✓ **Keyloggers**: A software tool that captures keystrokes on a compromised system. Kit (virus generator): A set of tools for generating new viruses automatically.
- ✓ **Logic bomb:** A program inserted into software by an intruder. A logic bomb lies dormant until a predefined condition is met, at which point the program triggers an unauthorized act.

- ✓ **Malware as a Service (MaaS):** A web-based provider of malware. MaaS may provide access to botnets, support hotlines, and servers that regularly update and test malware strains for efficacy. Mobile code: Software (for example, script, macro, or other portable instructions) that can be shipped unchanged to a heterogeneous collection of platforms and execute with identical semantics.
- ✓ **Potentially unwanted program (PUP**): A program that may be unwanted, despite the possibility that users consented to download it. PUPs include spyware, adware, and dialers and are often downloaded in conjunction with programs that users actually want.
- ✓ **Ransomware**: A type of malware in which the data on a victim's computer is locked, typically by encryption, and payment is demanded before the ransomed data is decrypted and access returned to the victim.
- ✓ **Remote access Trojan (RAT):** A malware program that includes a backdoor for administrative control over the target computer. RATs are usually downloaded invisibly with user-requested programs—such as games—or sent as email attachments.
- ✓ **Rootkit:** A set of hacker tools used after attacker has broken into a computer system and gained root-level access.
- ✓ **Scraper**: A simple program that searches a computer's memory for sequences of data that match particular patterns, such as credit card numbers.
- ✓ Point-of-sale terminals and other computers usually encrypt payment card data when storing and transmitting it, and attackers often use scrapers to locate card numbers in memory before they are encrypted or after they are decrypted for processing

- ✓ **Spammer programs**: Programs used to send large volumes of unwanted email. Spyware: Software that collects information from a computer and transmits it to another system.
- ✓ **Trojan horse**: A computer program that appears to have a useful function but also has a hidden and potentially malicious function that evades security mechanisms, sometimes by exploiting legitimate authorizations of a system entity that invokes the Trojan horse program.
- ✓ **Virus:** Malware that, when executed, tries to replicate itself into other executable code; when it succeeds, the code is infected. When the infected code is executed, the virus also executes. Web drive-by: An attack that infects a user system when the user visits a web page.
- ✓ **Worm**: A computer program that runs independently and propagates a complete working version of itself onto other hosts on a network.
- ✓ **Zombie, bot**: A program that is activated on an infected machine to launch attacks on other machines.

How to Prevent Malware:

- ❖ A variety of security solutions are used to detect and **prevent malware**. These include firewalls, next-generation firewalls, network intrusion prevention systems (IPS), deep packet inspection (DPI) capabilities, unified threat management systems, antivirus and anti-spam gateways, virtual private networks, content filtering and data leak prevention systems. In order to prevent malware, all security solutions should be tested using a wide range of malware-based attacks to ensure they are working properly.
- ❖ A robust, up-to-date library of malware signatures must be used to ensure testing is completed against the latest attacks

- The Cortex XDR agent combines **multiple methods of prevention** at critical phases within the attack lifecycle to halt the execution of malicious programs and stop the exploitation of legitimate applications, regardless of operating system, the endpoint's online or offline status, and whether it is connected to an organization's network or roaming.
- Because the Cortex XDR agent does not depend on signatures, it can prevent zero-day malware and unknown exploits through a combination of prevention methods.

Malware Detection:

- Advanced malware analysis and detection tools exist such as firewalls, Intrusion Prevention Systems (IPS), and sandboxing solutions.
- Some malware types are easier to detect, such as **ransomware**, which makes itself known immediately upon encrypting your files. Other malware like spyware, may remain on a target system silently to allow an adversary to maintain access to the system.
- Regardless of the malware type or malware meaning, its detectability or the person deploying it, the intent of malware use is always malicious.
- When you enable behavioral threat protection in your endpoint security policy, the Cortex XDR agent can also continuously monitor endpoint activity for malicious event chains identified by Palo Alto Networks.

Malware Removal:

- Antivirus software can remove most standard infection types and many options exist for off-the-shelf solutions.
- Cortex XDR enables remediation on the endpoint following an alert or investigation giving administrators the option to begin a variety of mitigation steps starting with isolating endpoints by disabling all network access on compromised endpoints except for traffic to the Cortex XDR console, terminating processes to stop any running malware from continuing to perform malicious activity on the endpoint, and blocking additional

executions, before quarantining malicious files and removing them from their working directories if the Cortex XDR agent has not already done so.

Malware Protection:

- To protect your organization against malware, you need a holistic, enterprise-wide malware protection strategy.
- Commodity threats are exploits that are less sophisticated and more easily detected and prevented using a combination of antivirus, anti-spyware, and vulnerability protection features along with URL filtering and Application identification capabilities on the firewall.

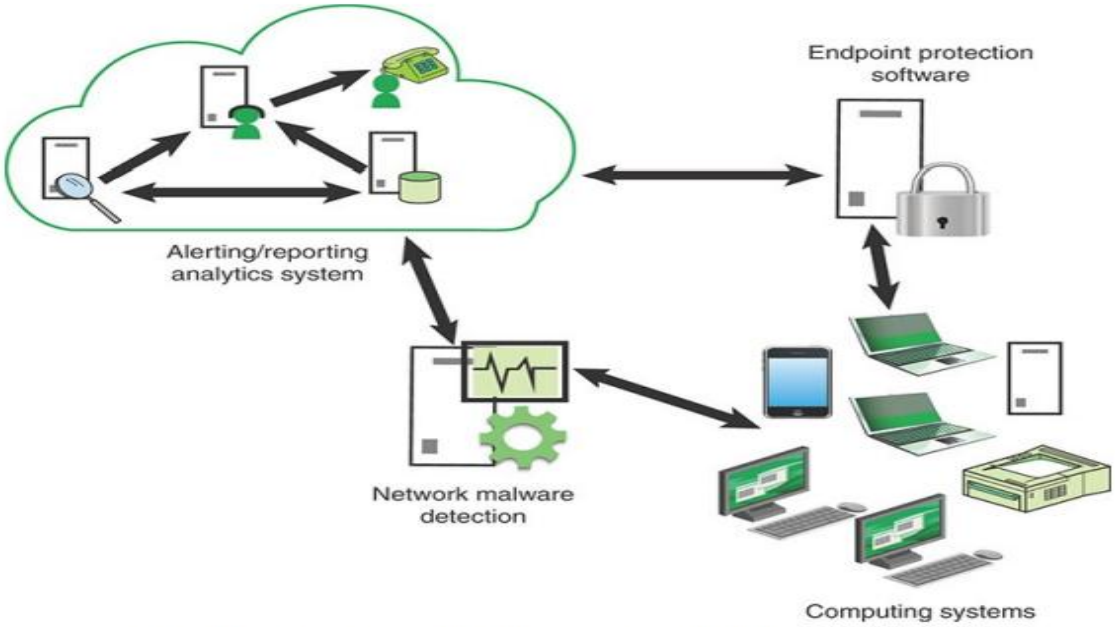


FIGURE 14.2 Malware System Entity Relationship Diagram

Intrusion Detection

Intrusion Detection It is useful to begin by defining the following terms:

Intrusion: Violations of security policy, usually characterized as attempts to affect the confidentiality, integrity, or availability of a computer or network. These violations come from attackers accessing systems from the Internet or from

authorized users of the systems attempting to overstep their legitimate authorization levels or using their legitimate access to the system to conduct unauthorized activity.

Intrusion detection: The process of collecting information about events occurring in a computer system or network and analyzing them for signs of intrusions.

Intrusion detection system (IDS): Hardware or software products that gather and analyze information from various areas within a computer or a network for the purpose of finding and providing real-time or near-real-time warning of attempts to access system resources in an unauthorized manner.

Intrusion detection systems employ two detection methods –

- ***Signature-based detection*** matches data activity to a signature or pattern in a signatures database. A new harmful behavior that is not in the database, for example, is overlooked when using signature-based detection.

- **Unlike signature-based detection**, *behavior-based detection* recognizes any abnormality and issues alarms, making it capable of identifying new sorts of threats. It's referred to as an expert system since it learns what regular behavior looks like in your system.

Any of the following can be considered an intrusion –

- Malware, sometimes known as ransomware, is a type of computer virus.
- Attempts to obtain unauthorized access to a system
- DDOS (Distributed Denial of Service) attacks
- Destruction of cyber-enabled equipment
- Employee security breaches that are unintentional (like moving a secure file into a shared folder)
- Untrustworthy users, both within and external to your company
- Phishing campaigns and other methods of deceiving consumers with ostensibly genuine communication are examples of social engineering assaults.

Network Intrusion Attack Techniques

- When it comes to compromising networks, attackers are increasingly relying on existing tools and procedures as well as stolen credentials.
- Operating system utilities, commercial productivity software, and scripting languages, **for example, are clearly not malware and have a wide range of lawful applications.**

- **Asymmetric Routing** – Attackers will typically employ several routes to gain access to the targeted device or network if the network allows for asymmetric routing.
- **Buffer Overwriting** – Attackers can substitute regular data in specified parts of computer memory on a network device with a barrage of commands that can subsequently be utilized as a part of a network incursion by overwriting certain memory locations.
- **Covert CGI Scripts** – The Common Gateway Interface (CGI), which allows servers to relay user requests to appropriate programs and get data back to then forward to users, unfortunately, provides an easy mechanism for attackers to gain access to network system files.
- **Enormous traffic loads** – Attackers can cause chaos and congestion in network settings by producing traffic loads that are too enormous for systems to fully filter, allowing them to carry out assaults without being discovered.
- **Worms** – The typical, isolated computer virus, or worm, is one of the easiest and most dangerous network penetration tools. Worms, which are commonly distributed by email attachments or instant messaging, use a considerable amount of network resources, preventing permitted activities from taking place.

Approaches to Intrusion Detection

Intrusion detection assumes that the behavior of the intruder differs from that of a legitimate user in ways that are quantifiable..

There are two general approaches to intrusion detection: misuse detection and anomaly detection

FIGURE 14.5 Approaches to Intrusion Detection

1) Misuse detection : is based on rules that specify system events, sequences of events, or observable properties of a system that are believed to be symptomatic of security incidents.

Misuse detectors use various pattern-matching algorithms, operating on large databases of attack patterns, or signatures.

- ✓ **Advantage of misuse detection**: is that it is accurate and generates few false alarms.
- ✓ **Disadvantage of misuse detection** :it that it cannot detect novel or unknown attacks.

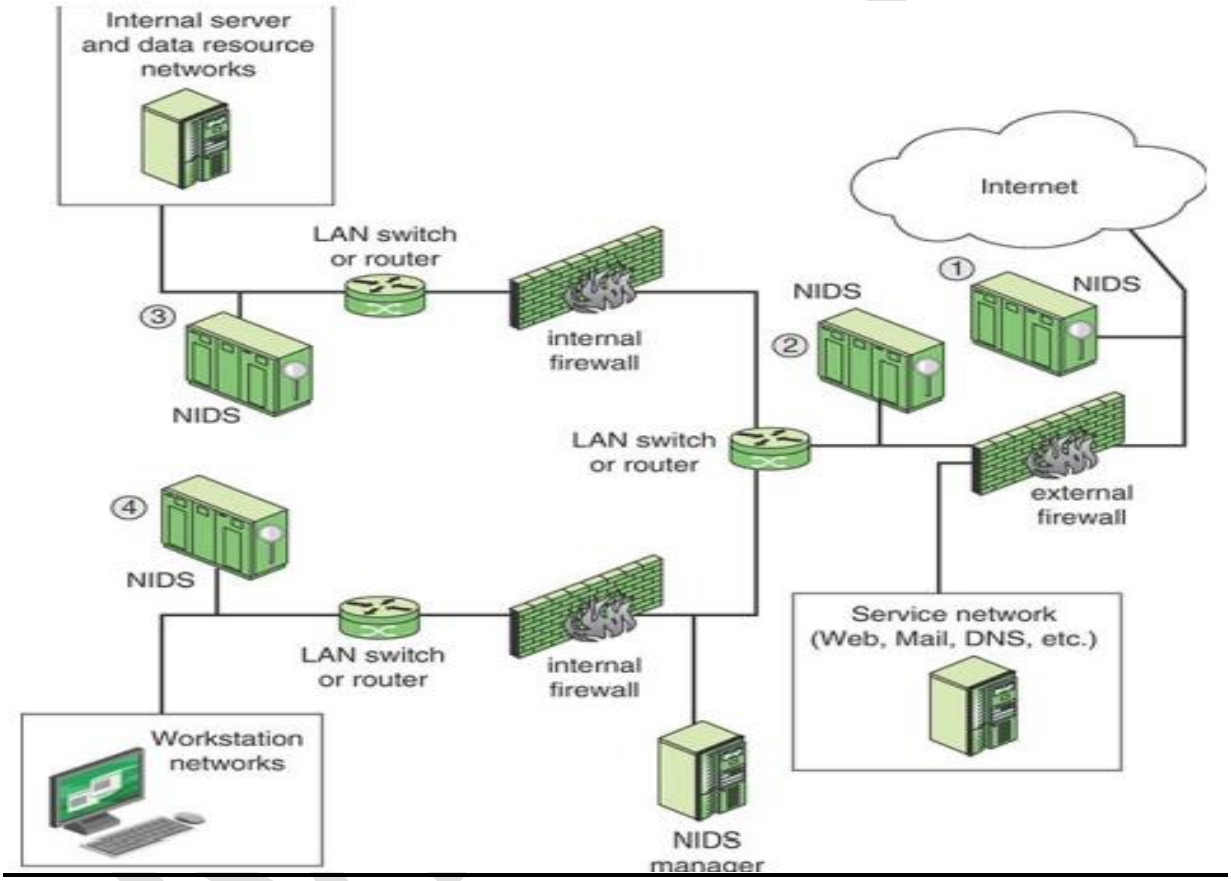
2) Anomaly detection : involves searching for activity that is different from the normal behavior of system entities and system resources.

- ✓ **Advantage of anomaly detection** is that it is able to detect previously unknown attacks based on an audit of activity.
- ✓ **Disadvantage of anomaly detection** :is that there is a significant trade-off between false positives and false negatives.

Network-Based Intrusion Detection Systems:

- ✓ A network-based IDS (NIDS) monitors the traffic on the network segment as a data source.

- ✓ This is generally accomplished by placing the network interface card in promiscuous mode to capture all network traffic that crosses its network segment.
- ✓ Network traffic on other segments and traffic on other means of communication (such as phone lines) can't be monitored by a single NIDS.



NIDS Sensor Deployment Example

<u>There are four types of locations for the sensors</u>:

<u>Outside the main enterprise firewall</u>: This placement is useful for establishing the level of threat for a given enterprise network. Those responsible for winning management support for security efforts find this placement valuable. In the network demilitarized zone (DMZ), inside the main firewall but outside

internal firewalls: This location monitors for penetration attempts that target web and other services that are generally open to outsiders.

Behind internal firewalls: A sensor can be positioned to monitor major backbone networks, such as those that support internal servers and database resources.

Behind internal firewalls: A sensor can be positioned to monitor LANs that support user workstations and servers specific to single departments. Figure NSD EXAMPLE can monitor for more specific attacks at network segments, as well as attacks originating from inside the organization.

Digital Rights Management

- The use of technology to limit and manage access to intellectual material is known as digital rights management(DRM).
- Another definition of DRM is giving over control of digital content to a computer program rather than the person who owns it.
- DRM protects the copyright holder's rights by preventing unlawful distribution and alteration of content.
- In simple words, DRM is a method of safeguarding copyrights in digital material. This strategy entails employing technologies that restrict the copying and use of copyrighted content as well as proprietary software.
- As digital material expands via peer-to-peer file sharing, torrent sites, and online piracy, DRM is becoming more significant.
- It assists entertainment and media enterprises in addressing cybersecurity issues that all businesses confront, such as securing consumer data, assuring and showing compliance, improving operational efficiency, and avoiding

downtime. Authors, musicians, filmmakers, and other content creators can use DRM to specify and limit what users can and cannot do with their work.

- It also enables them to preserve their copyrighted content protect the creative and financial commitment they make in their work, and prevent their media from being stolen or redistributed unlawfully.
- They can, for example, block users from accessing certain assets, avoiding any legal difficulties that may arise from illicit use. This is critical for copyright and intellectual property protection.

Benefits of Digital Rights Management

Apart from protecting copyright holders and content creators against piracy, DRM has a number of additional advantages.

- **Provides Privacy**: Businesses may use DRM technology to encrypt critical documents ranging from contracts and strategic plans to secret personnel information. It allows users to restrict access to files and trace who has accessed them, as well as prohibit them from being changed, saved, duplicated, or printed.
- **Securing ownership**: DRM is significant for authors and writers who want to safeguard their work. They can utilize technology to keep control of their material and prevent it from being changed or rebranded. This is also beneficial to scientists who wish to safeguard their discoveries and innovations.
- **Prevent unauthorized, unintended usage**.: DRM technology aids material buyers in adhering to the license information that governs how, when, and even where they can use it and avoids financial penalties.
- **Ensuring appropriate content access**: DRM limits content to targeted audiences and restricts it to certain audiences. Content aimed at those

above the age of 18 will, for example, be restricted to adults who can prove their age.

- **File privacy**: DRM facilitates companies in securing sensitive files and safeguarding their privacy. Intruders are unable to access or view confidential or sensitive information as a consequence of this.

DRM Structure and Components :DRM is best understood in terms of the key components of a DRM system and their interconnections.

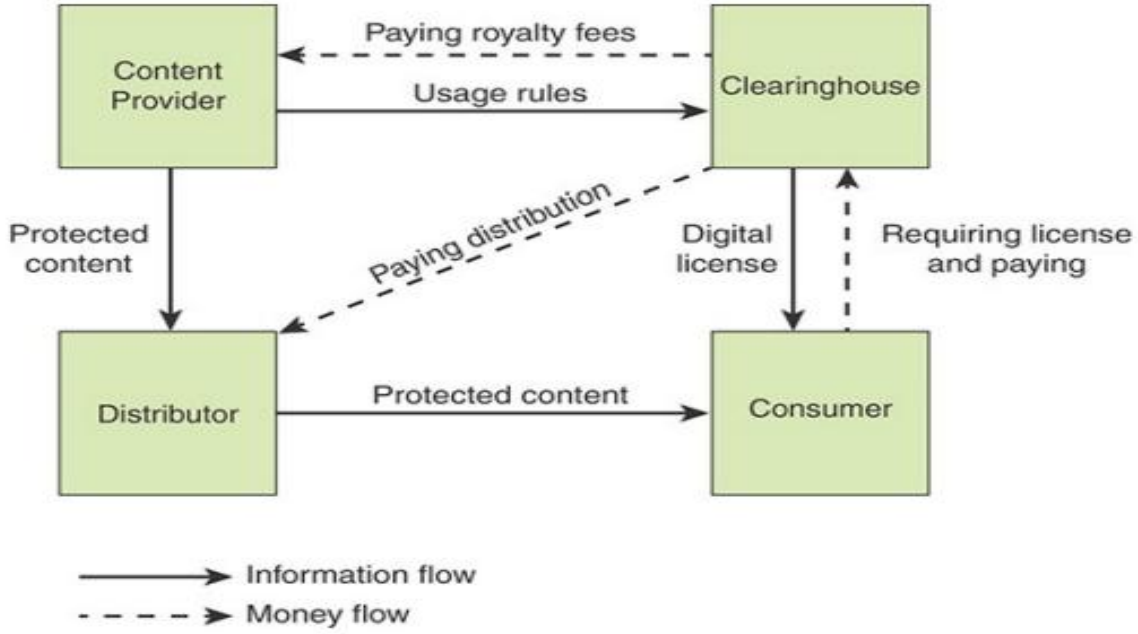


FIGURE 14.9 DRM Components

The principal users of DRM systems are as follows:

Content provider: Holds the digital rights to the content and wants to protect these rights. Examples are a music record label and a movie studio.

Distributor: Provides distribution channels, such as an online shop or a web retailer. For example, an online distributor receives the digital content from the

content provider and creates a web catalog presenting the content and rights metadata for the content promotion.

Consumer: Uses the system to access the digital content by retrieving downloadable or streaming content through the distribution channel and then paying for the digital license. The player/viewer application used by the consumer takes charge of initiating license request to the clearinghouse and enforcing the content usage rights.

Clearinghouse: Handles the financial transaction for issuing the digital license to the consumer and pays royalty fees to the content provider and distribution fees to the distributor accordingly. The clearinghouse is also responsible for logging license consumptions for the individual consumers.

DRM SYSTEM ARCHITECTURE

- ✓ The system is access by parties in three roles. Rights holders are the content providers, who either created the content or have acquired rights to the content. Service providers include distributors and clearinghouses. Consumers are those who purchase the right to access to content for specific uses.

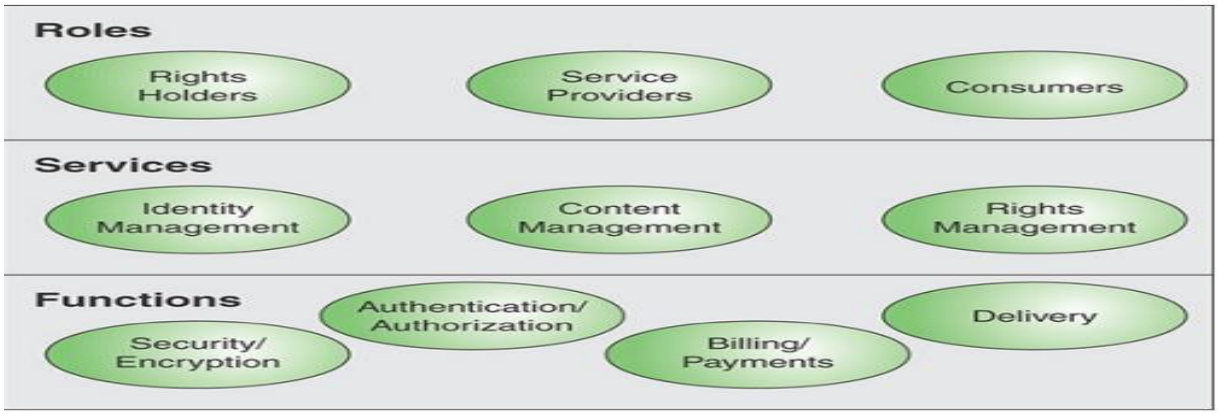


FIGURE 14.10 DRM System Architecture

The following services are provided by a DRM system:

Identity management: Mechanisms to uniquely identify entities, such as parties and content

Content management: Processes and functions needed to manage the content lifestyle

Rights management: Processes and functions needed to manage rights, rights holders, and associated requirements.

Cryptographic Solutions

Definition(s):

The generic term for a cryptographic device, COMSEC equipment, or combination of such devices/equipment containing either a classified algorithm or an unclassified algorithm.

Four uses for cryptography predominate:

Data encryption: Data encryption is a powerful and cost-effective means of providing data confidentiality and integrity. Once data are encrypted, the ciphertext does not have to be protected against disclosure. Further, if ciphertext is modified, it does not decrypt correctly. Data encryption is especially useful for transmitting data over the Internet or other network outside the control of the enterprise and also for storage in the cloud.

Data integrity: Cryptographic algorithms provide an effective way to determine whether a block of data (for example, email text, message, file, database record) was altered in an unauthorized manner.

Digital signature: The digital signature, or electronic signature, is the electronic equivalent of a written signature that is recognized as having the same legal status as a written signature. In addition to ensuring data integrity, digital signature algorithms provide a means of linking a document with a particular entity, as is done with a written signature.

User authentication: Cryptography is the basis for several advanced authentication methods. Instead of communicating passwords over an open network, authentication involves demonstrating knowledge of a cryptographic key. Using such a method, a one-time password that is not susceptible to eavesdropping is used.

Cryptographic Algorithms : Cryptographic algorithms fall into three broad categories: encryption/decryption algorithms, secure hash algorithms, and digital signature algorithms.

Symmetric Encryption: Symmetric encryption, also referred to as conventional encryption, is a cryptographic scheme in which encryption and decryption are performed using the same key.

Plaintext: This is the original message or data block that is fed into the algorithm as input. Encryption algorithm: The encryption algorithm performs various substitutions and transformations on the plaintext.

Secret key: The secret key is also input to the encryption algorithm. The exact substitutions and transformations performed by the algorithm depend on the key.

Cipher text: This is the scrambled message produced as output. It depends on the plaintext and the secret key. For a given data block, two different keys will produce two different cipher texts.

Decryption algorithm: This is the inverse of the encryption algorithm. It takes the ciphertext and the secret key and produces the original plaintext.

Public Key Encryption

Public key cryptography, also called asymmetric cryptography, involves the use of two separate keys, in contrast to symmetric encryption, which uses only one key.

The use of two keys has profound consequences in the areas of confidentiality, key distribution, and authentication.

A public key encryption scheme has several ingredients:

Plaintext: This is the readable message or data block that is fed into the algorithm as input.

Encryption algorithm: The encryption algorithm performs various transformations on the plaintext.

Public key and private key: This is a pair of keys that were selected so that if one is used for encryption, the other is used for decryption. The exact transformations performed by the encryption algorithm depend on the public or private key that is provided as input.

Ciphertext: This is the scrambled block produced as output. It depends on the plaintext and the key. For a given message, two different keys produce two different ciphertexts.

Decryption algorithm: This algorithm accepts the ciphertext and the matching key and produces the original plaintext.

L7L7ll

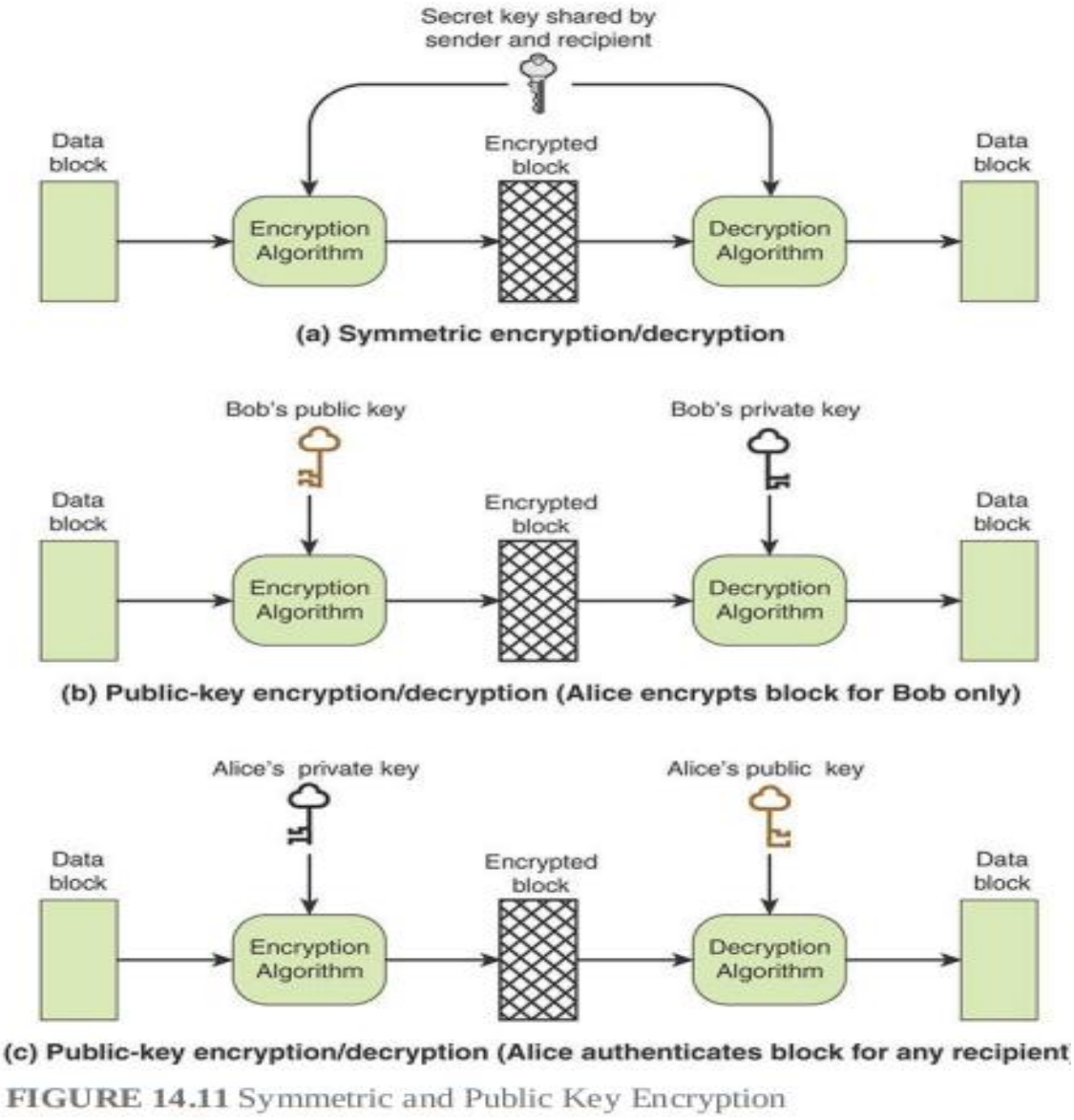


FIGURE 14.11 Symmetric and Public Key Encryption

Threat and Incident Management :

Technical Vulnerability Management: usually referred to simply as vulnerability management, is a security practice specifically designed to proactively mitigate or prevent the exploitation of technical vulnerabilities that exist in a system or an organization.

The process involves the identification, classification, remediation, and mitigation of various vulnerabilities in a system. It is an integral part of cybersecurity and is practiced together with risk management as well as other security practices.

- ✓ technical vulnerability: A hardware, firmware, communication, or software flaw that leaves an information processing system open to potential exploitation either externally or internally, resulting in risk for the system.

- ✓ FIGURE 15.1 Vulnerability Management Steps

five key steps involved in vulnerability management

1)Plan Vulnerability Management

Effective management of technical vulnerabilities begins with planning.

Key aspects of the planning process include the following:

- ❖ **Risk and process integration**: Technical vulnerability review is an operational aspect of an overall information security risk management

strategy. A vulnerability analysis must consider the relative risk impacts, including those related to the potential for operational disruption. These risks must also have a clear reporting path that allows for appropriate management awareness of risk factors and exposure. Vulnerability management should also provide input into change management and incident management processes.

- **Integration with asset inventory:** "Information Risk Assessment," asset identification is an integral part of risk assessment. The resulting asset inventory allows for action to be taken once a technical vulnerability is reviewed and a mitigation strategy agreed on. By integrating the asset inventory with the vulnerability management system, an enterprise can prioritize high-risk systems where the impact of technical vulnerabilities can be greatest.
- **Establishment of clear authority to review vulnerabilities**: Because probing a network for vulnerabilities can disrupt systems and expose private data, an enterprise needs to have in place a policy and buy-in from top management before performing vulnerability assessments.
- **System and application life cycle integration**: The review of vulnerabilities must be integrated in system release and software development planning to ensure that potential weaknesses are identified early to both lower risks and manage costs of finding these issues prior to identified release dates.

2)Discover Known Vulnerabilities

The discover step involves monitoring sources of information about known vulnerabilities to hardware, software, and network equipment. Key sources of information include the following:

- National Institute of Standards and Technology (NIST) National Vulnerability Database (NVDB) and Common Vulnerability Scoring System (CVSS)
- Computer emergency response (or readiness) team (CERT): Such a team is a cooperative venture that collects information about system vulnerabilities and disseminates it to systems managers.
- **Packet Storm:** Packet Storm provides around-the-clock information and tools to help mitigate both personal data and fiscal loss on a global scale. As new information surfaces, Packet Storm releases everything immediately through its RSS (Rich Site Summary) feeds, Twitter, and Facebook.
- **Security Focus** : The SecurityFocus Vulnerability Database provides security professionals with up-to-date information on vulnerabilities for all platforms and services.
- **Internet Storm Center (ISC):** Maintained by the SANS Technology Institute, the ISC provides a free analysis and warning service to thousands of Internet users and organizations and is actively working with Internet service providers to fight back against the most malicious attackers.

3) Scan for Vulnerabilities:

- Scanning can cause disruptions. The scanning process can impact performance. This is especially true with legacy systems, which can have problems even with simple network port scans. IT operations staff need to be in the loop. Make them aware of the importance and relevance of scans.
- Also, timing needs to be resolved to ensure that scanning does not conflict with regular maintenance schedules.
- Scanning can generate huge amounts of data and numerous false positives. Technical vulnerability management practices produce very large data sets.
- Accordingly, use frequent follow-up evaluations to validate the findings. Reviewing all these vulnerabilities is infeasible.

4) Log and Report :

When a vulnerability scan is completed, the organization should log the results so that personnel can verify the activity of the regular vulnerability scanning tools.

An organization should rank discovered vulnerabilities, such as attaching a score to each vulnerability that reflects the following:

- The skill required to exploit the vulnerability
- The availability of the exploit to potential attackers
- The privilege gained upon successful exploitation
- The risk and impact of this vulnerability if exploitation is successful
- The resulting vulnerability scoring and metrics provide a valuable guide in

the remediation process.

5) Remediate Vulnerabilities:

An organization should deploy automated patch management tools and software update tools for operating system and software/applications on all systems for which such tools are available and safe.

Security event management (SEM)

- ✓ The process of identifying , gathering, monitoring, analyzing, and reporting security-related events. The objective of SEM is to extract from a large volume of security events those events that qualify as incidents.
- ✓ SEM takes data input from all devices/nodes and other similar applications, such as log management software.
- ✓ The collected events data is analyzed with security algorithms and statistical computations to trace out any vulnerability, threat, or risk.

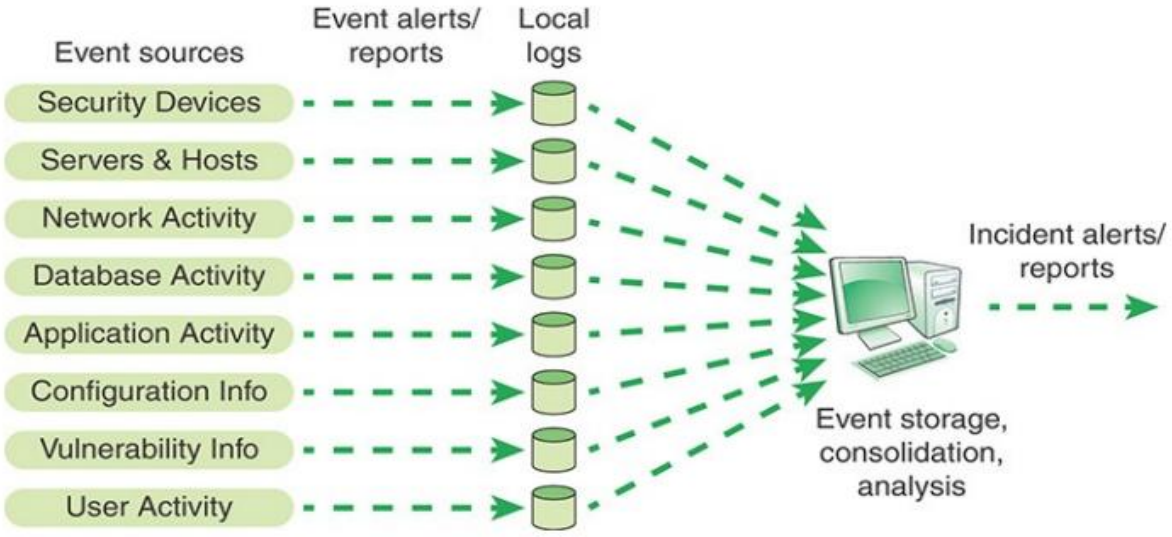


FIGURE 15.3 Security Event Management

SEM Functions :The first phase of event management is the collection of event data in the form of logs, as discussed in the preceding section. As event data are generated, they are generally stored in logs local to the devices that generate them.

A number of steps need to be taken at this point:

1. Normalization: For effective management, the log data needs to be in a common format to enable further processing.

2. Filtering: This step includes assigning priorities to various types of events. On the basis of priority, large number of events can be set aside and not subject to further analysis, or they can be archived in case there is a need to review them later.

3. Aggregation: The IT facility of a large enterprise generates millions of events per day. It is possible to aggregate them by categories into a more manageable amount of data. For example, if a particular type of traffic is blocked a number of times, it is

sufficient to record as a single aggregate event the type of traffic and the number of times it was blocked over a particular time frame.

Analysis includes the following aspects:

Pattern matching: It is important to look for data patterns within the fields of stored event records. A collection of events with a given pattern can signal a security incident.

Scan detection: Often, an attack begins with a scan of IT resources by the attacker, such as port scans, vulnerability scans, or other types of pings. A substantial number of scans being found from a single source or a small number of sources can signal a security incident.

Threshold detection: A straightforward form of analysis is the detection of a threshold being crossed. For example, if the number of occurrences of a type of event exceeds a given threshold in a certain time period, that constitutes an incident.

Event correlation: Correlation consists of using multiple events from a number of sources to determine that an attack or suspicious activity occurred. For example, if a particular type of attack proceeds in multiple stages, the separate events that record those multiple activities need to be correlated in order to see the attack. Another aspect of correlation is to correlate particular events with known system vulnerabilities, which might result in a high-priority incident.

Forensic Investigations

Forensic Techniques into Incident Response, defines computer forensics, or digital forensics, as the identification, collection, examination, and analysis of data while

preserving the integrity of the information and maintaining a strict chain of custody for the data.

Computer forensics seeks to answer several critical questions, including the following:

- ✓ What happened?
- ✓ When did the events occur?
- ✓ In what order did the events occur?
- ✓ What was the cause of these events?
- ✓ Who caused these events to occur?
- ✓ What enabled these events to take place?
- ✓ What was affected?
- ✓ How much was it affected?

Phases of Digital Forensics Process

Prepare : Preparation involves the planning and policy-making activities related to forensic investigation. A section of the security policy should deal with computer forensics. SP 800-86, Guide to Integrating Forensic Techniques into Incident Response, recommends the following considerations:

- ✓ Ensure that policies contain clear statements addressing all major forensic considerations, such as contacting law enforcement, performing monitoring, and conducting regular reviews of forensic policies and procedures
- ✓ Create and maintain procedures and guidelines for performing forensic tasks, based on the organization's policies and all applicable laws and regulations

- ✓ Ensure that policies and procedures support the reasonable and appropriate use of forensic tools
- ✓ Ensure that IT professionals are prepared to participate in forensic activities.

Key actions include the following:

- ✓ Creating a file system baseline to help detect changes
- ✓ Utilizing a central system log server
- ✓ Maintaining network-level logging at key control points on the network
- ✓ Synchronizing system clocks and log timestamps using central Network Time Protocol (NTP) servers for systems that generate logs.

identification

The identification : phase is initiated when there is a request for a forensic analysis.

- ✓ This phase involves understanding the purpose of the request and the scope of the investigation, such as type of case, subjects involved, and system involved.
- ✓ A forensic analyst must determine if a request contains sufficient information to start the process. If not, the analyst must coordinate with the requester to determine the next step.

Collect

- ✓ The data collection process includes one or more of the following, depending on the purpose of the forensic analysis:
- ✓ Capturing data from system logs, event logs, and incident logs
- ✓ Discovering data on computer systems
- ✓ Recovering deleted, encrypted, or damaged file information

- ✓ Monitoring online activity
- ✓ Making a bit-image copy of an affected system's hard drive
- ✓ Detecting violations of corporate policy

Preserve

Preserve Several actions comprise the preservation of data process, including the following:

- ✓ Creating a log that documents when, from where, how, and by whom data were collected
- ✓ Storing the data in a secure fashion to prevent tampering or contamination
- ✓ Logging each access to the data made for forensic analysis.

Analyze

Analysis depends on the specifics of each job. The examiner usually provides feedback to the client during analysis, and from this dialogue the analysis may take a different path or be narrowed to specific areas. Examples of analysis tasks include:

- ✓ Checking for changes to the system such as new programs, files, services, and users
- ✓ Looking at running processes and open ports for anomalous behavior
- ✓ Checking for Trojan horse programs and toolkits
- ✓ Checking for other malware
- ✓ Looking for illegal content
- ✓ Looking for indicators of compromise
- ✓ Determining the who , when, where, what, and how details of a security incident.

Report

The nature of any report resulting from a forensic investigation depends on the original purpose of the investigation.

- ✓ **Alternative explanations**: The available information may not provide a definitive explanation of the cause and nature of an incident.
- ✓ **Audience consideration**: An incident requiring law enforcement involvement requires highly detailed reports of all information gathered and can also require copies of all evidentiary data obtained.
- ✓ **Actionable information**: Reporting also includes identifying actionable information gained from data that allows an analyst to collect new sources of information.

Local Environment Management

- ❖ local environment In the context of cybersecurity, a physically distinct and separate area, which may be a single office space, building, or building complex. The local environment may have unique physical security, personnel security, and information security requirements that are distinct from those of the rest of the enterprise.

The following are some of the factors that require the development of strategies tailored to the local environment:

- ❖ Most organizations have many different end-user environments, often across physical locations and comprising individuals who use a wide range of technologies to handle information.
- ❖ There are significant differences in the knowledge, behavior, and actions of end users in different environments.
- ❖ End users employ a variety of corporate-issued and personally owned devices (in organizations that have bring your own device [BYOD] policies).

- End users sometimes blur the boundaries between work and personal computing (for example, with mobile computing).
- End users typically want to configure their own user environments and install personal software such as applications for social networking, instant messaging, peer-to-peer networking, and voice over IP (VoIP).

Local Environment Profile :Security management and senior executives may not have a good grasp on the security issues in a local environment, such as the value of information that employees have access to and use, the threats this information is exposed to when not adequately protected, and the potential business impact if this information is compromised in the end-user environment.

Key elements of the profile include:

- **Individuals:** Each local environment should have one or more staff members with specific information security responsibilities, as discussed subsequently. The profile should detail the types of users at the location, in terms of their application and data usage, level of security awareness training, security privileges, and whether they use mobile devices and, if so, what type.
- **Business processes and information**: This area includes the types of information used and whether any sensitive information is accessible. The profile should include descriptions of business processes that involve user information access, as well as descriptions of any external suppliers (for example, cloud service providers).
- **Technology use**: This topic comprises the applications and IT equipment used. Location: The profile should provide a description of the location housing the users and equipment. The profile should indicate to what degree the location is accessible to the public or to others who are not part

of the organization, whether the physical space is shared with other organizations .

Local Security Coordination: An enterprise must manage the twofold concern of ensuring that the enterprise wide information security policy is applied in the local environment and that policy elements are adapted to the local profile.

Information Security Coordinat: An information security coordinator is responsible for developing and maintaining information security in the local environment and coordinating this with the organization's security executives and managers.

The information security coordinator should be responsible for the following:

- Developing the local environment profile
- Determining the best way to implement enterprise security policy in the local environment
- Providing oversight of implementation of the information security policy in the local environment Ensuring that physical security arrangements are in place and adequate
- Assisting with communicating security policies and requirements to local end users and local management

Business Continuity

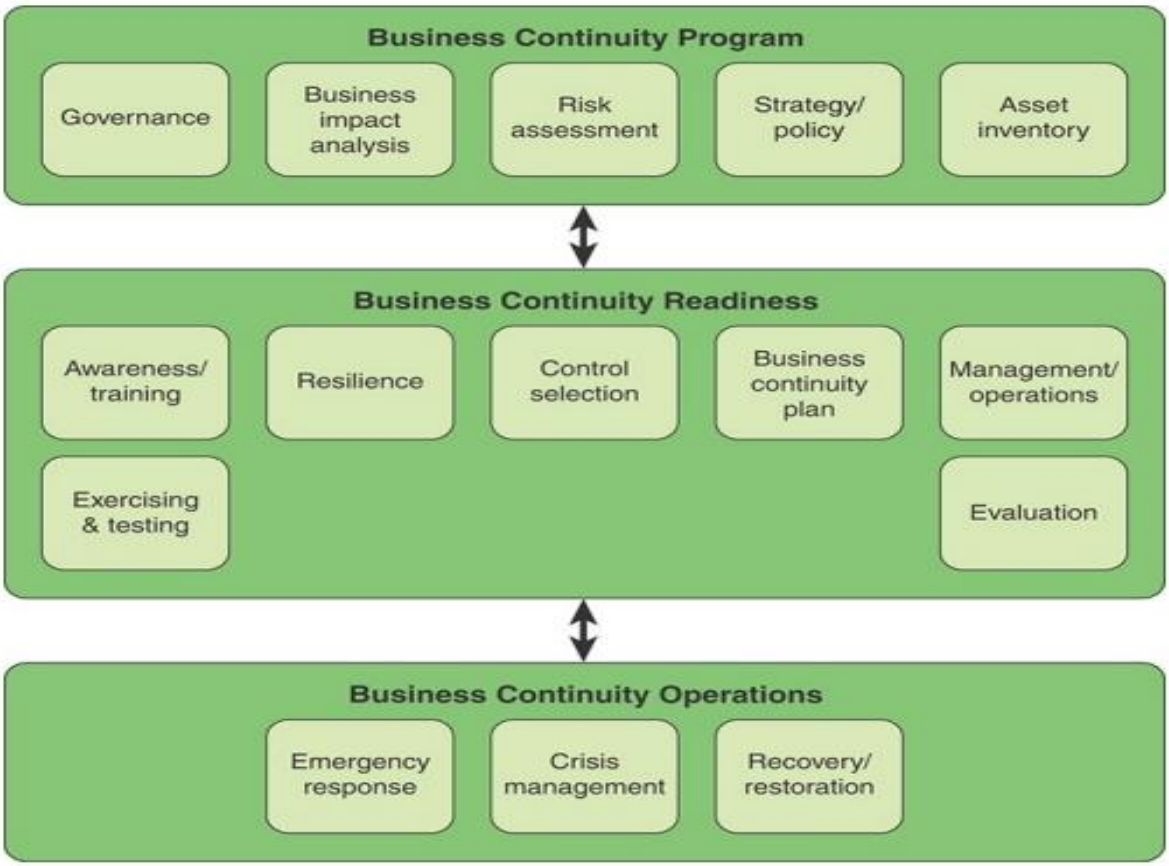
- A fundamental concern for all organizations is business continuity. An organization needs to perform essential functions during an emergency situation that disrupts normal operations and resume normal operations in a timely manner after the emergency has ended.

The International Organization for Standardization (ISO) has published a family of standards for business continuity management that enterprise security managers should be familiar with:

- ISO 22300, Security and Resilience—Vocabulary: Provides a glossary of relevant terms.
- ISO 22301, Business Continuity Management Systems—Requirements: Specifies requirements for setting up and managing an effective business continuity management system (BCMS). This is the first international standard focused exclusively on business continuity.
- ISO 22313, Business Continuity Management Systems—Guidance: Provides guidance, where appropriate, on the requirements specified in ISO 22301 and provides recommendations ("should") and permissions ("may") in relationship to them.
- ISO 22317, Business Continuity Management Systems: Guidelines for Business Impact Analysis (BIA): Provides guidelines (based on good international practice) for performing a business impact analysis (BIA), which is a requirement of ISO 22301 .It provides guidance for establishing, implementing, and maintaining a formal and documented process for business impact analysis. It is applicable to all organizations, regardless of type, location, size, and nature of the organization.
- ISO 22318, Business Continuity Management Systems: Guidelines for Business Impact Analysis (BIA): Provides guidelines for supply chain continuity.

Two additional useful guidance documents are:

- ✓ National Institute of Standards and Technology (NIST) SP 800-34, Contingency Planning Guide for Federal Information Systems: Provides a detailed description of the planning process.
- ✓ European Union Agency for Network and Information Security's (ENISA's) IT Business Continuity Management: An Approach for Small and Medium Sized Organizations: Provides a detailed list of controls for implementing business continuity plans.

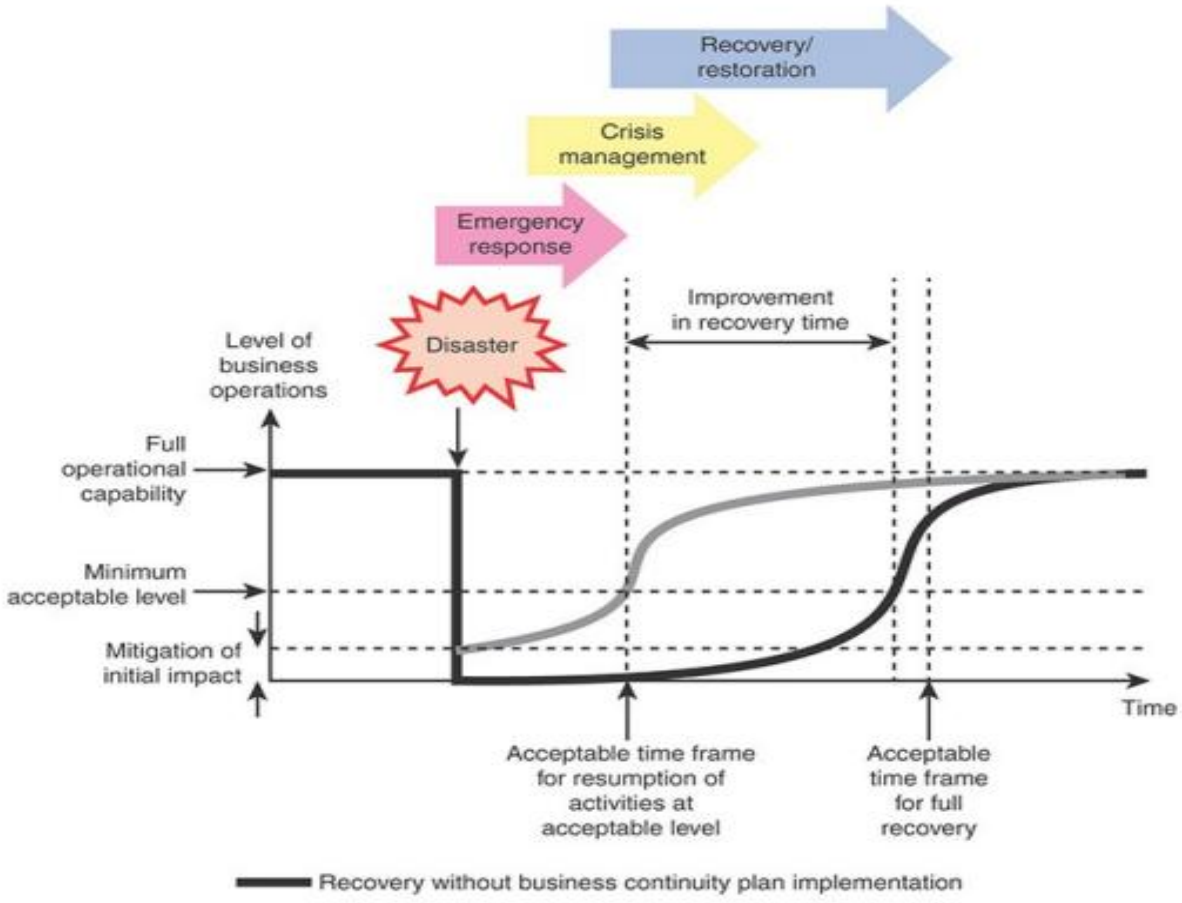


- ✓ FIGURE 17.1 Elements of Business Continuity Management

Business Continuity Concepts This section provides an overview of business continuity.

- **Business:** For purposes of discussing business continuity, the operations and services performed by an organization in pursuit of its objectives, goals, or mission. As such, it is equally applicable to large, medium, and small organizations operating in industrial, commercial, public, and not-for-profit sectors.
- **Business continuity**: The capability of an organization to continue delivering products or services at acceptable predefined levels following a disruptive incident. Business continuity embraces all the operations in a company, including how employees function in compromised situations.
- **Business continuity management (BCM):** A holistic management process that identifies potential threats to an organization and the impacts to business operations those threats, if realized, might cause, and that provides a framework for building organizational resilience with the capability of an effective response that safeguards the interests of its key stakeholders, reputation, brand, and value-creating activities.
- **Business continuity management system (BCMS):** Part of an overall management system that establishes, implements, operates, monitors, reviews, maintains, and improves business continuity. The management system includes organizational structure, policies, planning activities, responsibilities, procedures, processes, and resources.
- **Business continuity manage**r: An individual who manages, designs, oversees, and/or assesses an enterprise's business continuity capability to ensure that the enterprise's critical functions continue to operate following disruptive events.
- **Business continuity plan (BCP)**: The documentation of a predetermined set of instructions or procedures that describe how an organization's mission/business processes will be sustained during and after a significant disruption.

- **Business continuity** program: An ongoing management and governance process supported by top management and appropriately resourced to implement and maintain business continuity management.



Business Continuity Objectives

- Minimize loss of life, injury, and property damage.
- Mitigate the duration, severity, or pervasiveness of disruptions that do occur.
- Achieve timely and orderly resumption of essential functions and the return to normal operations. Protect essential facilities, equipment, records, and assets. Be executable with or without warning.

following key components that are essential to maintaining business continuity:

- Management
- Staff
- CT systems
- Buildings and equipment

UNIT-V **SECURITY ASSESSMENT**

Security Monitoring and Improvement- Security Audit-Security Performance-Information Risk Reporting-Information Security Compliance Monitoring-Security Monitoring and Improvement Best Practices.

Security Monitoring and Improvement

Two aspects of security monitoring that lead to improvement in organizational security:
The security audit and security performance.

Security Audit

- ✓ A security audit relates to security policies and the mechanisms and procedures used to enforce that policy.
- ✓ A security audit trail is an important component of a security audit.

security audit trail

- ✓ An independent review and examination of a system's records and activities to determine the adequacy of system controls , ensure compliance with established security policy and procedures, detect breaches in security services, and recommend any changes that are indicated for countermeasures.

security audit trail

- ✓ A chronological record of system activities that is sufficient to enable the reconstruction and examination of the sequence of environments and activities surrounding or leading to an operation, procedure, or event in a security-relevant transaction from inception to final results.

What is a security audit?

A security audit is a systematic evaluation of the security of a company's information system by measuring how well it conforms to an established set of criteria. A thorough audit typically assesses the security of the system's physical configuration and environment, software, information handling processes and user practices.

Why are security audits important?

There are several reasons to do a security audit. They include these six goals:

1. Identify security problems and gaps, as well as system weaknesses.
2. Establish a security baseline that future audits can be compared with.
3. Comply with internal organization security policies.
4. Comply with external regulatory requirements.
5. Determine if security training is adequate.
6. Identify unnecessary resources.

Types of security audits

Security audits come in two forms, internal and external audits , that involve the following procedures:

- **Internal audits:** In these audits, a business uses its own resources and internal audit department.

  Internal audits are used when an organization wants to validate business systems for policy and procedure compliance.

The objectives of an internal security audit include the following:

- Identify security weaknesses
- Provide an opportunity to improve the information security management system
- Provide management with information about the status of security
- Deliver information about the status of security to management
- Review compliance of security systems with the information security policy of the organization
- Find and resolve noncompliance

- **External audits:** With these audits, an outside organization is brought in to conduct an audit.

  External audits are also conducted when an organization needs to confirm it is conforming to industry standards or government regulations.

The objectives of the external security include the following:

- Assess the process of the internal audit
- Determine the commonality and frequency of recurrence of various types of security violations
- Identify the common causes of various types of security violations
- Provide advisory and training inputs to tackle the neglect of procedures
- Review and update the policy

X.816, Security Audit and Alarms Framework, lists the following objectives for a security audit:

- Allows the adequacy of the security policy to be evaluated
- Aids in the detection of security violations
- Facilitates making individuals accountable for their actions (or for actions by entities acting on their behalf)
- Assists in the detection of misuse of resources
- Acts as a deterrent to individuals who might attempt to damage the system

Steps involved in a security audit

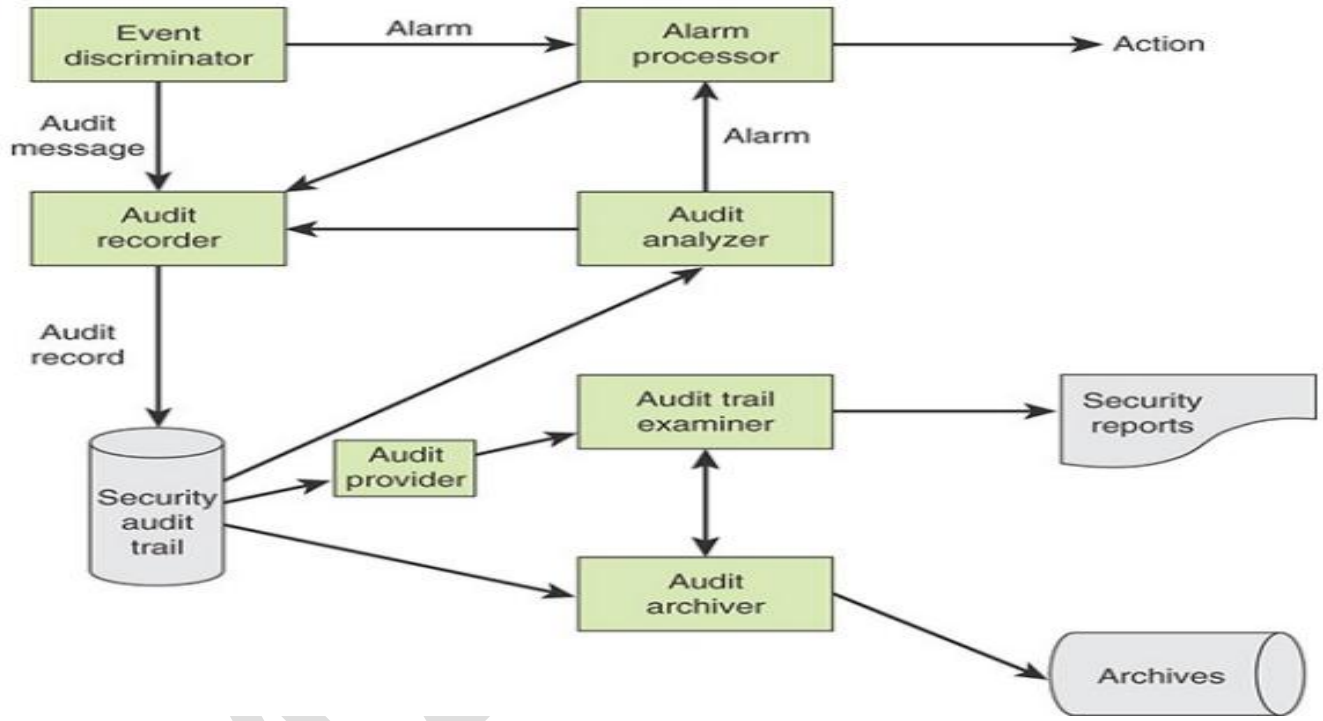
These five steps are generally part of a security audit:

1. **Agree on goals.** Include all stakeholders in discussions of what should be achieved with the audit.
2. **Define the scope of the audit.** List all assets to be audited, including computer equipment, internal documentation and processed data.
3. **Conduct the audit and identify threats.** List potential threats related to each Threats can include the loss of data, equipment or records through natural disasters, malware or unauthorized users.
4. **Evaluate security and risks.** Assess the risk of each of the identified threats happening, and how well the organization can defend against them.

5. **Determine the needed controls.** Identify what security measures must be implemented or improved to minimize risks.

Security Audit and Alarms Model

X.816 has developed a model that shows the elements of the security auditing function and their relationships to security alarms



Security Audit and Alarms Model

The key elements of this model are as follows:

Event discriminator: This logic embedded in the software of the system monitors system activity and detects security-related events that it was configured to detect.

Audit recorder: For each detected event, the event discriminator transmits the information to an audit recorder. The model depicts this transmission in the form of a message. The audit could also be done by recording the event in a shared memory area.

Alarm processor:

Some of the events detected by the event discriminator are defined to be alarm events. For such events, an alarm is issued to an alarm processor. The alarm processor takes some action based on the alarm. This action is itself an auditable event and so is transmitted to the audit recorder.

Security audit trail:

The audit recorder creates a formatted record of each event and stores it in the security audit trail.

Audit analyzer:

The security audit trail is available to the audit analyzer, which, based on a pattern of activity, may define a new auditable event that is sent to the audit recorder and may generate an alarm.

Audit archiver:

This software module periodically extracts records from the audit trail to create a permanent archive of auditable events.

Archives:

The audit archives are a permanent store of security-related events on this system. Audit provider: The audit provider is an application and/or user interface to the audit trail.

Audit trail examiner:

The audit trail examiner is an application or a user who examines the audit trail and the audit archives for historical trends, for computer forensic purposes, and for other analyses.

Security reports: The audit trail examiner prepares human-readable security reports.

Data to Collect for Auditing

- The choice of what data to collect should be based on a number of requirements. One issue is the amount of data to collect, which is determined by the range of areas of interest and by the granularity of data collection.

- The more data collected, the greater the performance penalty on the system. Larger amounts of data may also unnecessarily burden the various algorithms used to examine and analyze the data. Further, the presence of large amounts of data creates a temptation to generate security reports excessive in number or length.

Security audit trail design is the selection of data items to capture, including the following:

- Events related to the use of the auditing software
- Events related to the security mechanisms on the system
- Any events that are collected for use by the various security detection and prevention mechanisms, including items related to intrusion detection and items related to firewall operation
- Events related to system management and operation
- Events related to operating system access (for example, via system calls) Events related to application access for selected applications
- Events related to remote access

The following sections look at categories for audit trail design. System-Level Audit Trail

System-Level Audit Trails :

System-level audit trails are generally used to monitor and optimize system performance but serve a security audit function as well. The system enforces certain aspects of security policy, such as access to the system itself.

Application-Level Audit Trails:

Application-level audit trails are used to detect security violations in an application or to detect flaws in the application's interaction with the system. For critical applications, or those that deal with sensitive data, an application-level audit trail provides the desired level of detail to assess security threats and impacts.

User-Level Audit Trails:

A user-level audit trail traces the activity of an individual user over time. It is used to hold a user accountable for his or her actions. Such audit trails are also useful as

input to an analysis program that attempts to define normal versus anomalous behavior.

A user-level audit trail records user interactions with the system, such as commands issued, identification and authentication attempts, and files and resources accessed. The audit trail also captures the user's use of applications.

Network-Level Audit Trails :

Network-level audit trails encompass a wide variety of network activity. Enterprises use such audit trails to evaluate system performance and perform load balancing. These audit trails can also include security-related data, such as that generated by firewalls, virtual private network managers, and IPsec traffic.

Physical Access Audit Trails :

Physical access audit trails are generated by equipment that controls physical access and are then transmitted to a central host for subsequent storage and analysis. Examples are card-key systems and alarm systems.

Security Performance

- Security performance is the measurable result of security controls applied to information systems and supporting information security programs.
- The Information Security Forum's (ISF's) Standard of Good Practice for Information Security (SGP) defines the security performance function as comprising three areas:

Security monitoring and reporting: Consists of monitoring security performance regularly and reporting to specific audiences, such as executive management.

Information risk reporting: Consists of producing reports relating to information risk and presenting reporting to executive management on a regular basis.

Information security compliance monitoring: Consists of information security controls derived from regulatory and legal drivers and contracts, used to monitor security compliance.

Security Performance Measurement Two terms are relevant to this discussion:

Security performance:

The measurable result of security controls applied to information systems and supporting information security programs.

Security performance metric:

A variable related to security performance to which a value is assigned as the result of measurement. Also called a security performance measure.

National Institute of Standards and Technology (NIST) IR 7564, Directions in Security Metrics Research, lists the following as the main broad uses of security metrics:

Strategic support:

Assessments of security properties can be used to aid in different kinds of decision making, such as program planning, resource allocation, and product and service selection.

Quality assurance:

Security metrics can be used during the software development life cycle to eliminate vulnerabilities, particularly during code production, by performing functions such as measuring adherence to secure coding standards, identifying vulnerabilities that are likely to exist, and tracking and analyzing security flaws that are eventually discovered.

Tactical oversight:

Monitoring and reporting of the security status or posture of an IT system can be carried out to determine compliance with security requirements (for example, policies, procedures, regulations), gauge the effectiveness of security controls and manage risk, provide a basis for trend analysis, and identify specific areas for improvement.

Sources of Security:

Metrics A security officer or a group responsible for developing a set of metrics for security performance assessment draws on several authoritative sets, some of which are described here.

Three processes comprise this domain:

Performance and conformance:

Collect, validate, and evaluate business, IT, and process goals and metrics. Monitor to ensure that processes are performing against agreed-on performance and conformance goals and metrics and provide reporting that is systematic and timely.

System of internal control:

Continuously monitor and evaluate the control environment, including self-assessments and independent assurance reviews. Enable management to identify control deficiencies and inefficiencies and to initiate improvement actions. Plan, organize, and maintain standards for internal control assessment and assurance activities.

Compliance with external requirements: Evaluate whether IT processes and IT-supported business processes are compliant with laws, regulations, and contractual requirements. Obtain assurance that the requirements were identified and complied with and integrate IT compliance with overall enterprise compliance.

lists the metrics defined for these three processes.

Goal	Metrics
Performance and Conformance	
Information security performance is monitored on an ongoing basis	▪ Percentage of business processes that meet defined information security requirements
Information security and information risk practices conform to internal compliance requirements.	▪ Percentage of information security practices that satisfy internal compliance requirements
System of Internal Control	
Information security controls are deployed and operating effectively	▪ Percentage of processes that satisfy information security control requirements ▪ Percentage of controls in which information security control requirements are met
Monitoring processes for information security controls are in place and results are reported	▪ Percentage of information security controls appropriately monitored and results reported and reviewed
Compliance with External Requirements	
Information security and information risk practices conform to external compliance requirements	▪ Percentage of information security practices that satisfy external compliance requirements
Monitoring is conducted for new or revised external requirements with an impact on information security	▪ Number or percentage of projects initiated by information security to implement new external requirements

Suggested Security Performance Metrics (COBIT 5 for Information Security)

TABLE 18.3 Examples of Security Performance Metrics (NIST 800-55)

Area	Metric
Security budget	Percentage of the agency's information system budget devoted to information security
Vulnerability management	Percentage of high vulnerabilities mitigated within organizationally defined time periods after discovery
Access control	Percentage of remote access points used to gain unauthorized access
Awareness and training	Percentage of information system security personnel that have received security training
Audit and accountability	Average frequency of audit records review and analysis for inappropriate activity
Certification, accreditation, and security assessments	Percentage of new systems that have completed certification and accreditation (C&A) prior to their implementation
Configuration management	Percentage approved and implemented configuration changes identified in the latest automated baseline configuration
Contingency planning	Percentage of information systems that have conducted annual contingency plan testing
Identification and authentication	Percentage of users with access to shared accounts

Information Risk Reporting

- Risk reporting is a process that produces information systems reports that address threats, capabilities, vulnerabilities, and inherent risk changes. Risk reporting describes any information security events that the institution faces and the effectiveness of management's response to and resilience in the face of those events. An organization needs to have a method of disseminating those reports to appropriate members of management. T

The Information Systems Audit and Control Association (ISACA) has developed useful guidance on information risk reporting, based on COBIT 5 [ISAC09].

The guidance makes use of two key concepts in COBIT 5:

Process: A collection of practices influenced by the enterprise's policies and procedures that takes inputs from a number of sources (including other processes), manipulates the inputs, and produces outputs (for example, products, services).

Processes have clear business reasons for existing, accountable owners, clear roles and responsibilities around the execution of the process, and the means to measure performance.

Activity: The main action taken to operate the process, which provides guidance to achieve management practices for successful governance and management of enterprise IT.

Activities:

- Describe a set of necessary and sufficient action-oriented implementation steps to achieve a governance practice or management practice.
- Consider the inputs and outputs of the process
- Are based on generally accepted standards and good practices
- Support establishment of clear roles and responsibilities
- Are non prescriptive and need to be adapted and developed into specific procedures appropriate for the enterprise.

TABLE 18.5 Risk Reporting Goals and Metrics

Category	Goals	Metrics
Process	▪ Ensure that information on the true state of IT-related exposures and opportunities is made available in a timely manner and to the right people for appropriate response	▪ Percentage of risk issues inappropriately distributed too high or too low in the enterprise hierarchy ▪ Number of IT-related events with business impact not earlier reported as in IT risk ▪ Percentage of critical assets/resources covered by monitoring activities ▪ Timeliness of reports on IT exposures relative to the next expected threat or loss event. ▪ Potential business impact of exposures discovered by assurance groups
Activity	▪ Communicate IT risk analysis results ▪ Report IT risk management	▪ Percentage of risk analysis reports accepted on initial delivery

	compliance ▪ Interpret independent IT assessment findings	management reports ▪ Frequency of risk management activity reporting ▪ Number of IT-related events with business impact not previously reported as an IT risk. ▪ Number of IT risk issues identified by outside parties yet to be interpreted and mapped into the risk profile
Risk reporting	▪ Ensure that IT-related risk issues, opportunities and events are addressed in a cost-effective manner and in line with business priorities	▪ The cumulative business impact from IT-related incidents and events anticipated by risk evaluation processes but not yet addressed by mitigation or event action planning

-

Information Security Compliance Monitoring

- The objective of information security compliance monitoring is to ensure that information security controls are consistently prioritized and addressed according to information security obligations associated with legislation, regulations, contracts, industry standards, or organizational policies.
- COBIT 5 Guidelines COBIT 5 provides specific guidance on security monitoring and reporting for compliance with external requirements.

For the process of ensuring compliance with external requirements, COBIT 5 defines the following steps:

1.**Identify external compliance requirements**: On a continuous basis, identify and monitor for changes in local and international laws, regulations, and other external requirements that the organization must comply with from an IT perspective.

2. **Optimize response to external requirements**: Review and adjust policies, principles, standards, procedures, and methodologies to ensure that legal,

regulatory, and contractual requirements are addressed and communicated. Consider industry standards, codes of good practice, and good practice guidance for adoption and adaptation.

3. **Confirm external compliance**: Confirm compliance of policies, principles, standards, procedures, and methodologies with legal, regulatory, and contractual requirements.

4. **Obtain assurance of external compliance**: Obtain and report assurance of compliance and adherence with policies, principles, standards, procedures, and methodologies. Confirm that corrective actions to address compliance gaps are closed in a timely manner.

Compliance Strategy

The following steps constitute a general approach to information security compliance monitoring:

1. Identify key stakeholders and/or partners across the organization who regularly deal with institutional compliance issues (for example, legal, risk management, privacy, audit).

2. Identify key standards, regulations, contractual commitments, and other areas that address specific requirements for security and privacy.

3. Perform a high-level gap analysis of each compliance requirement that is applicable to determine where progress needs to be made.

4. Develop a prioritized action plan that will help organize remedial efforts.

5. Develop a compliance policy, standard, roles and responsibilities, and/or procedures in collaboration with other key stakeholders

Security Monitoring and Improvement Best Practices

The SGP breaks down the best practices in the security monitoring and improvement category into two areas and eight topics and provides detailed checklists for each topic. **The areas and topics are as follows:**

- ***Security audit***: This area provides guidance for conducting thorough, independent, and regular audits of the security status of target environments

(critical business environments, processes, applications, and supporting systems/networks).

- ***Security audit management***: The objective of this topic is to ensure that security controls have been implemented effectively and that risk is being adequately managed and to provide the owners of target environments and executive management with an independent assessment of their security status.
  - **Security audit process—planning**: Provides guidance on a methodology for security audits.
  - **Security audit process—fieldwork**: Provides a checklist of actions related to collecting relevant background material, performing security audit tests, and recording the results of the tests.
  - **Security audit process—reporting**: Provides a checklist of items that should be in the security audit report, as well as guidance on the reporting process.
  - **Security audit process—monitoring**: Provides a checklist of actions to ensure the risks identified during security audits are treated effectively , compliance requirements are being met, and agreed security controls are being implemented within agreed time scales.
- **Security performance**:

  This area provides guidance for monitoring information risks; compliance with the security-related elements of legal, regulatory, and contractual requirements; and the overall information security condition of the organization on a regular basis, reporting the results to specific audiences, such as executive management.
- **Security monitoring and performance**:

  The objective of this topic is to ensure that there is a reporting function that provides selected audiences with a relevant, accurate, comprehensive, and coherent assessment of information security performance.
- **Information risk reporting**:

The objective of this topic is to ensure that there is a reporting function that provides executive management with an accurate, comprehensive, and coherent view of information risk across the organization.

- **Information security compliance monitoring**:
  This topic provides guidelines for a security management process that should be established, which comprises information security controls derived from regulatory and legal drivers and contracts.

REFERENCE QUESTIONS

1. Briefly define the terms security audit and security audit trail.

2. What are the key elements of the X.816 security audit model's relationship with security alarms?

3. What are some of the auditable items suggested in the X.816 model of security audits and alarms?

4. What are the four different types of audit trails?

5. What are the key objectives of an external security audit?

6. How does the SGP define the security performance function?

7. NIST IR 7564 defines three broad uses of security metrics. Enumerate them.

8.What are the three key processes for the COBIT 5 Monitor, Evaluate, and Assess domain?

9. What guidelines does COBIT 5 define for the performance and conformance process?

10. Describe the monitoring and reporting function, as per SP 800-55.

11. ISACA's guidance on information risk reporting is based on which two concepts of COBIT 5?

12. What are the generic steps for security compliance monitoring?

4. Jiawei Han, Micheline Kamber, Jian Pei, “Data Mining: Concepts and Techniques”, Third Edition, Morgan Kaufmann, 2012.
5. Brad Dayley, “Teach Yourself NoSQL with MongoDB in 24 Hours”, Sams Publishing, First Edition, 2014.
6. C. J. Date, A. Kannan, S. Swamynathan, “An Introduction to Database Systems”, Eighth Edition, Pearson Education, 2006

CO-PO Mapping

CO	POs					
	PO1	PO2	PO3	PO4	PO5	PO6
1	2	1	2	2	2	2
2	2	1	3	2	2	2
3	2	1	3	2	2	3
4	2	1	3	2	3	3
5	2	1	3	2	2	2
Avg	2	1	2.8	2	2.2	2.4

MC4203 CLOUD COMPUTING TECHNOLOGIES L T P C
3 0 0 3

COURSE OBJECTIVES:

- To understand the basic concepts of Distributed systems.
- To learn about the current trend and basics of Cloud computing.
- To be familiar with various Cloud concepts.
- To expose with the Server, Network and storage virtualization.
- To be aware of Microservices and DevOps.

UNIT I DISTRIBUTED SYSTEMS 9

Introduction to Distributed Systems – Characterization of Distributed Systems – Distributed Architectural Models –Remote Invocation – Request-Reply Protocols – Remote Procedure Call – Remote Method Invocation – Group Communication – Coordination in Group Communication – Ordered Multicast – Time Ordering – Physical Clock Synchronization – Logical Time and Logical Clocks.

UNIT II BASICS OF CLOUD COMPUTING 9

Cloud Computing Basics – Desired features of Cloud Computing – Elasticity in Cloud – On demand provisioning - Applications – Benefits – Cloud Components: Clients, Datacenters & Distributed Servers – Characterization of Distributed Systems – Distributed Architectural Models - Principles of Parallel and Distributed computing - Applications of Cloud computing – Benefits – Cloud services – Open source Cloud Software: Eucalyptus, Open Nebula, Open stack, Aneka, Cloudsim.

UNIT III CLOUD INFRASTRUCTURE 9

Cloud Architecture and Design – Architectural design challenges – Technologies for Network based system - NIST Cloud computing Reference Architecture – Public, Private and Hybrid

clouds – Cloud Models : IaaS, PaaS and SaaS – Cloud storage providers - Enabling Technologies for the Internet of Things – Innovative Applications of the Internet of Things.

UNIT IV CLOUD ENABLING TECHNOLOGIES 9

Service Oriented Architecture – Web Services – Basics of Virtualization – Emulation – Types of Virtualization – Implementation levels of Virtualization – Virtualization structures – Tools & Mechanisms – Virtualization of CPU, Memory & I/O Devices – Desktop Virtualization – Server Virtualization – Google App Engine – Amazon AWS - Federation in the Cloud.

UNIT V MICROSERVICES AND DEVOPS 9

Defining Microservices - Emergence of Microservice Architecture – Design patterns of Microservices – The Mini web service architecture – Microservice dependency tree – Challenges with Microservices - SOA vs Microservice – Microservice and API – Deploying and maintaining Microservices – Reason for having DevOps – Overview of DevOps – Core elements of DevOps – Life cycle of DevOps –Adoption of DevOps - DevOps Tools – Build, Promotion and Deployment in DevOps.

SUGGESTED ACTIVITIES:

1. Write a client and server program to calculate the value of PI, in which server calls the remote procedure of the client side (C programming)
2. Create an word document of your class time table and store locally and also on cloud and share it (use www.zoho.com , docs.google.com)
3. Create your resume in a neat format using google and zoho cloud Programs on PaaS
4. Discuss processor virtualization, memory virtualization, I/O virtualization in VMWare
5. Set up Azure DevOps, Import Code and Create the Azure DevOps Build Pipeline

COURSE OUTCOMES:

Upon completion of the course, the students will be able to

CO1: Use Distributed systems in Cloud Environment.

CO2: Articulate the main concepts, key technologies, strengths and limitations of Cloud computing.

CO3: Identify the Architecture, Infrastructure and delivery models of Cloud computing.

CO4: Install, choose and use the appropriate current technology for the implementation of Cloud.

CO5: Adopt Microservices and DevOps in Cloud environments.

TOTAL:45 PERIODS

REFERENCES

1. Kai Hwang, Geoffrey C. Fox & Jack J.Dongarra, "Distributed and Cloud Computing, From Parallel Processing to the Internet of Things", Morgan Kaufmann Publishers, First Edition, 2012
2. Andrew S. Tanenbaum & Maarten Van Steen,“Distributed Systems - Principles and Paradigms”, Third Edition, Pearson, 2017.
3. Thomas Erl, Zaigham Mahood & Ricardo Puttini, “Cloud Computing, Concept, Technology & Architecture”, Prentice Hall, SecondEdition, 2013.
4. Richard Rodger, “The Tao of Microservices”, ISBN 9781617293146, Manning Publications, First Edition, December 2017.
5. Magnus Larsson, “Hands-On Microservices with Spring Boot and Spring Cloud: Build and deploy microservices using spring cloud, Istio and kubernetes”, Packt Publishing Ltd, First Edition, September 2019.

UNIT I INTRODUCTION

Introduction to Cloud Computing – Definition of Cloud – Evolution of Cloud Computing – Underlying Principles of Parallel and Distributed Computing – Cloud Characteristics – Elasticity in Cloud – On-demand Provisioning.

INTRODUCTION

EVOLUTION OF DISTRIBUTED COMPUTING

Grids enable access to shared computing power and storage capacity from your desktop. Clouds enable access to leased computing power and storage capacity from your desktop.

- Grids are an **open source** technology. Resource users and providers alike can understand and contribute to the management of their grid
- Clouds are a **proprietary** technology. Only the resource provider knows exactly how their cloud manages data, job queues, security requirements and so on.
- The concept of grids was proposed in 1995. The Open science grid (OSG) started in 1995 The EDG (European Data Grid) project began in 2001.
- In the late 1990`s Oracle and EMC offered early private cloud solutions . However the term cloud computing didn't gain prominence until 2007.

SCALABLE COMPUTING OVER THE INTERNET

Instead of using a centralized computer to solve computational problems, a parallel and distributed computing system uses multiple computers to solve large-scale problems over the Internet. Thus, distributed computing becomes data-intensive and network-centric.

The Age of Internet Computing

- high-performance computing (HPC) applications is no longer optimal for measuring system performance
- The emergence of computing clouds instead demands high-throughput computing (HTC) systems built with parallel and distributed computing technologies
- We have to upgrade data centers using fast servers, storage systems, and high-bandwidth networks.

The Platform Evolution

- From 1950 to 1970, a handful of mainframes, including the IBM 360 and CDC 6400

- From 1960 to 1980, lower-cost minicomputers such as the DEC PDP 11 and VAX Series
- From 1970 to 1990, we saw widespread use of personal computers built with VLSI microprocessors.
- From 1980 to 2000, massive numbers of portable computers and pervasive devices appeared in both wired and wireless applications
- Since 1990, the use of both HPC and HTC systems hidden in clusters, grids, or Internet clouds has proliferated

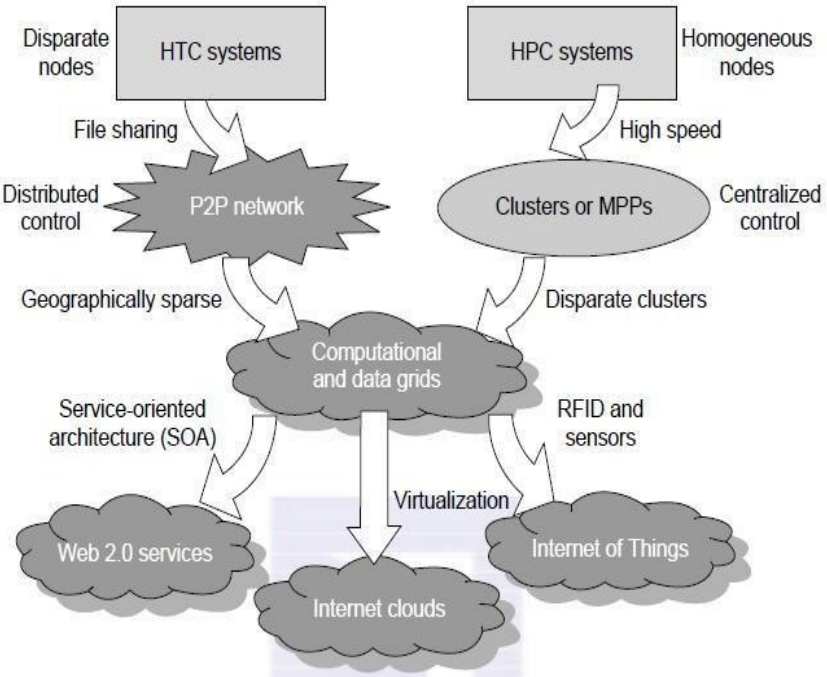


FIGURE 1.1

Evolutionary trend toward parallel, distributed, and cloud computing with clusters, MPPs, P2P networks, grids, clouds, web services, and the Internet of Things.

On the HPC side, supercomputers (massively parallel processors or MPPs) are gradually replaced by clusters of cooperative computers out of a desire to share computing resources. The cluster is often a collection of homogeneous compute nodes that are physically connected in close range to one another.

- On the HTC side, peer-to-peer (P2P) networks are formed for distributed file sharing and content delivery applications. A P2P system is built over many client machines (a concept we will discuss further in Chapter 5). Peer machines are globally distributed in nature. P2P, cloud computing, and web service platforms are more focused on

HTC applications than on HPC applications. Clustering and P2P technologies lead to the development of computational grids or data grids.

- For many years, HPC systems emphasize the raw speed performance. The speed of HPC systems has increased from Gflops in the early 1990s to now Pflops in 2010.
- The development of market-oriented high-end computing systems is undergoing a strategic change from an HPC paradigm to an HTC paradigm. This HTC paradigm pays more attention to high-flux computing. The main application for high-flux computing is in Internet searches and web services by millions or more users simultaneously. The performance goal thus shifts to measure high throughput or the number of tasks completed per unit of time. HTC technology needs to not only improve in terms of batch processing speed, but also address the acute problems of cost, energy savings, security, and reliability at many data and enterprise computing centers.
- Advances in virtualization make it possible to see the growth of Internet clouds as a new computing paradigm. The maturity of radio-frequency identification (RFID), Global Positioning System (GPS), and sensor technologies has triggered the development of the Internet of Things (IoT). These new paradigms are only briefly introduced here.
- The high-technology community has argued for many years about the precise definitions of centralized computing, parallel computing, distributed computing, and cloud computing. In general, distributed computing is the opposite of centralized computing. The field of parallel computing overlaps with distributed computing to a great extent, and cloud computing overlaps with distributed, centralized, and parallel computing.

Terms

Centralized computing

This is a computing paradigm by which all computer resources are centralized in one physical system. All resources (processors, memory, and storage) are fully shared and tightly coupled within one integrated OS. Many data centers and supercomputers are centralized systems, but they are used in parallel, distributed, and cloud computing applications.

• Parallel computing

In parallel computing, all processors are either tightly coupled with centralized shared memory or loosely coupled with distributed memory. Inter processor communication is accomplished through shared memory or via message passing. Acomputer system capable of parallel computing is commonly known as a parallel computer. Programs running in a parallel computer are called parallel programs. The process of writing parallel programs is often referred to as parallel programming.

- **Distributed computing** This is a field of computer science/engineering that studies distributed systems. A distributed system consists of multiple autonomous computers, each having its own private memory, communicating through a computer network. Information exchange in a distributed system is accomplished through message passing. A computer program that runs in a distributed system is known as a distributed program. The process of writing distributed programs is referred to as distributed programming.
- **Cloud computing** An Internet cloud of resources can be either a centralized or a distributed computing system. The cloud applies parallel or distributed computing, or both. Clouds can be built with physical or virtualized resources over large data centers that are centralized or distributed. Some authors consider cloud computing to be a form of utility computing or service computing . As an alternative to the preceding terms, some in the high-tech community prefer the term concurrent computing or concurrent programming. These terms typically refer to the union of parallel computing and distributing computing, although biased practitioners may interpret them differently.
- **Ubiquitous computing** refers to computing with pervasive devices at any place and time using wired or wireless communication. The Internet of Things (IoT) is a networked connection of everyday objects including computers, sensors, humans, etc. The IoT is supported by Internet clouds to achieve ubiquitous computing with any object at any place and time. Finally, the term Internet computing is even broader and covers all computing paradigms over the Internet. This book covers all the aforementioned computing paradigms, placing more emphasis on distributed and cloud computing and their working systems, including the clusters, grids, P2P, and cloud systems.

Internet of Things

- The traditional Internet connects machines to machines or web pages to web pages. The concept of the IoT was introduced in 1999 at MIT .

- The IoT refers to the networked interconnection of everyday objects, tools, devices, or computers. One can view the IoT as a wireless network of sensors that interconnect all things in our daily life.
- It allows objects to be sensed and controlled remotely across existing network infrastructure

SYSTEM MODELS FOR DISTRIBUTED AND CLOUD COMPUTING

- Distributed and cloud computing systems are built over a large number of autonomous computer nodes.
- These node machines are interconnected by SANs, LANs, or WANs in a hierarchical manner. With today's networking technology, a few LAN switches can easily connect hundreds of machines as a working cluster.
- A WAN can connect many local clusters to form a very large cluster of clusters.

Clusters of Cooperative Computers

A computing cluster consists of interconnected stand-alone computers which work cooperatively as a single integrated computing resource.

- In the past, clustered computer systems have demonstrated impressive results in handling heavy workloads with large data sets.

Cluster Architecture

cluster built around a low-latency, high bandwidth interconnection network. This network can be as simple as a SAN or a LAN (e.g., Ethernet).

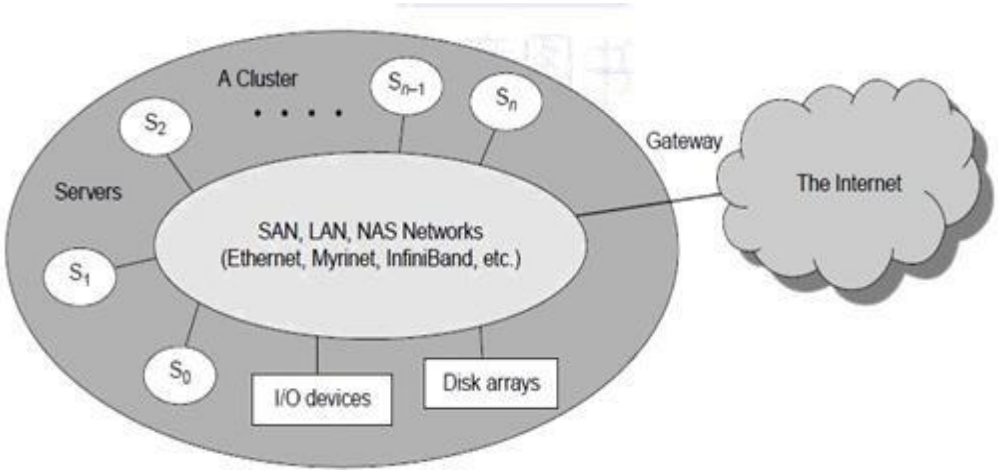


Figure 1.2 Clusters of Servers

Figure 1.2shows the architecture of a typical server cluster built around a low-latency, high bandwidth interconnection network. This network can be as simple as a SAN (e.g., Myrinet) or a LAN (e.g., Ethernet).

- To build a larger cluster with more nodes, the interconnection network can be built with multiple levels of Gigabit Ethernet, or InfiniBand switches.
- Through hierarchical construction using a SAN, LAN, or WAN, one can build scalable clusters with an increasing number of nodes. The cluster is connected to the Internet via a virtual private network (VPN) gateway.
- The gateway IP address locates the cluster. The system image of a computer is decided by the way the OS manages the shared cluster resources.

Most clusters have loosely coupled node computers. All resources of a server node are managed by their own OS. Thus, most clusters have multiple system images as a result of having many autonomous nodes under different OS control.

Single-System Image(SSI)

- Ideal cluster should merge multiple system images into a single-system image (SSI).
- Cluster designers desire a cluster operating system or some middleware to support SSI at various levels, including the sharing of CPUs, memory, and I/O across all cluster nodes.

An SSI is an illusion created by software or hardware that presents a collection of resources as one integrated, powerful resource. SSI makes the cluster appear like a single machine to the user. A cluster with multiple system images is nothing but a collection of independent computers.

Hardware, Software, and Middleware Support

- Clusters exploring massive parallelism are commonly known as MPPs. Almost all HPC clusters in the Top 500 list are also MPPs.
- The building blocks are computer nodes (PCs, workstations, servers, or SMP), special communication software such as PVM, and a network interface card in each computer node.

Most clusters run under the Linux OS. The computer nodes are interconnected by a high-bandwidth network (such as Gigabit Ethernet, Myrinet, InfiniBand, etc.). Special cluster middleware supports are needed to create SSI or high availability (HA). Both sequential and parallel applications can run on the cluster, and special parallel environments are needed to facilitate use of the cluster resources. For example, distributed memory has multiple images. Users may want all distributed memory to be shared by all servers by forming distributed shared

memory (DSM). Many SSI features are expensive or difficult to achieve at various cluster operational levels. Instead of achieving SSI, many clusters are loosely coupled machines. Using virtualization, one can build many virtual clusters dynamically, upon user demand.

Cloud Computing over the Internet

- A cloud is a pool of virtualized computer resources.
- A cloud can host a variety of different workloads, including batch-style backend jobs and interactive and user-facing applications.
- A cloud allows workloads to be deployed and scaled out quickly through rapid provisioning of virtual or physical machines.
- The cloud supports redundant, self-recovering, highly scalable programming models that allow workloads to recover from many unavoidable hardware/software failures.
- Finally, the cloud system should be able to monitor resource use in real time to enable rebalancing of allocations when needed.

a. Internet Clouds

- Cloud computing applies a virtualized platform with elastic resources on demand by provisioning hardware, software, and data sets dynamically .The idea is to move desktop computing to a service-oriented platform using server clusters and huge databases at data centers.
- Cloud computing leverages its low cost and simplicity to benefit both users and providers.
- Machine virtualization has enabled such cost-effectiveness. Cloud computing intends to satisfy many user applications simultaneously.

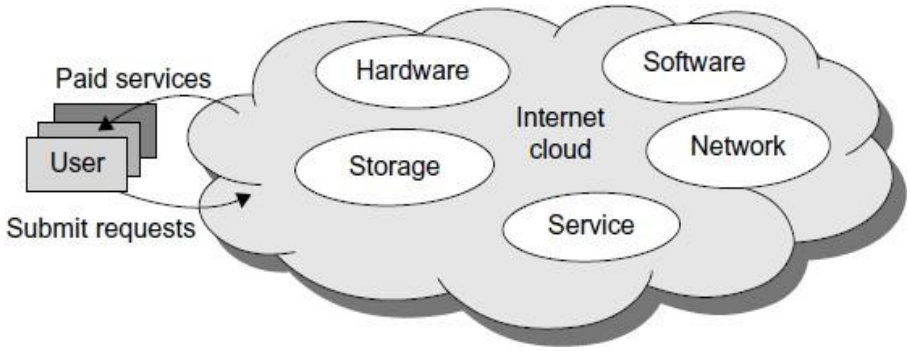


Figure 1.3 Internet Cloud

b. The Cloud Landscape

- The cloud ecosystem must be designed to be secure, trustworthy, and dependable. Some computer users think of the cloud as a centralized resource pool. Others consider the cloud to be a server cluster which practices distributed computing over all the servers Traditionally, a distributed computing system tends to be owned and operated by an autonomous administrative domain (e.g., a research laboratory or company) for on-premises computing needs.
- Cloud computing as an on-demand computing paradigm resolves or relieves us from these problems.

Three Cloud service Model in a cloud landscape

Infrastructure as a Service (IaaS)

- This model puts together infrastructures demanded by users—namely servers, storage, networks, and the data center fabric.
- The user can deploy and run on multiple VMs running guest OS on specific applications.
- The user does not manage or control the underlying cloud infrastructure, but can specify when to request and release the needed resources.

Platform as a Service (PaaS)

• This model enables the user to deploy user-built applications onto a virtualized cloud platform. PaaS includes middleware, databases, development tools, and some runtime support such as Web 2.0 and Java.

• The platform includes both hardware and software integrated with specific programming interfaces.

• The provider supplies the API and software tools (e.g., Java, Python, Web 2.0, .NET). The user is freed from managing the cloud infrastructure.

Software as a Service (SaaS)

• This refers to browser-initiated application software over thousands of paid cloud customers. The SaaS model applies to business processes, industry applications, consumer relationship management (CRM), enterprise resources planning (ERP), human resources (HR), and collaborative applications. On the customer side, there is no upfront investment in servers or software licensing. On the provider side, costs are rather low, compared with conventional hosting of user applications.

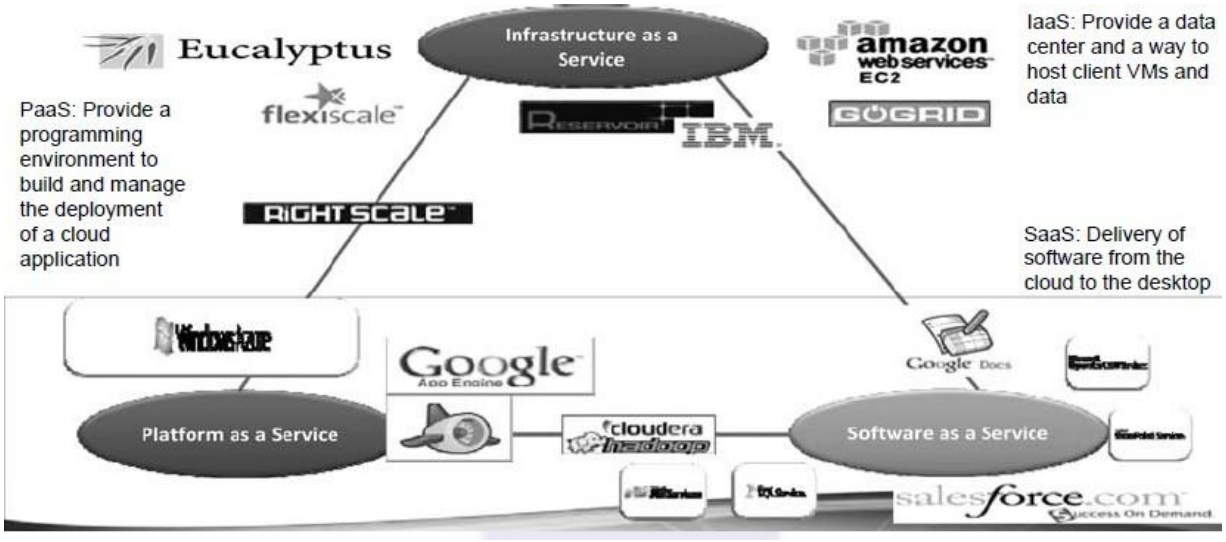


Figure 1.4 The Cloud Landscape in an application

Internet clouds offer four deployment modes: private, public, managed, and hybrid . These modes demand different levels of security implications. The different SLAs imply that the security responsibility is shared among all the cloud providers, the cloud resource consumers, and the third party cloud-enabled software providers. Advantages of cloud computing have been advocated by many IT experts, industry leaders, and computer science researchers.

Reasons to adapt the cloud for upgraded Internet applications and web services:

1. Desired location in areas with protected space and higher energy efficiency
2. Sharing of peak-load capacity among a large pool of users, improving overall utilization
3. Separation of infrastructure maintenance duties from domain-specific application development
4. Significant reduction in cloud computing cost, compared with traditional computing paradigms
5. Cloud computing programming and application development
6. Service and data discovery and content/service distribution
7. Privacy, security, copyright, and reliability issues
8. Service agreements, business models, and pricing policies

- Cloud computing is using the internet to access someone else's software running on someone else's hardware in someone else's data center.
- The user sees only one resource (HW, Os) but uses virtually multiple os. HW resources etc..
- Cloud architecture effectively uses virtualization
- A model of computation and data storage based on “pay as you go” access to “unlimited” remote data center capabilities
- A cloud infrastructure provides a framework to manage scalable, reliable, on-demand access to applications
- Cloud services provide the “invisible” backend to many of our mobile applications
- High level of elasticity in consumption
- Historical roots in today’s Internet apps
    - Search, email, social networks, e-com sites
    - File storage (Live Mesh, Mobile Me)

Definition

- **“The National Institute of Standards and Technology (NIST) defines cloud computing as a "pay-per-use model for enabling available, convenient and on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."**

Cloud Computing Architecture

- Architecture consists of 3 tiers
    - Cloud Deployment Model
    - Cloud Service Model
    - Essential Characteristics of Cloud Computing

Essential Characteristics 1

- On-demand self-service.

- A consumer can unilaterally provision computing capabilities such as server time and network storage as needed automatically, without requiring human interaction with a service provider.

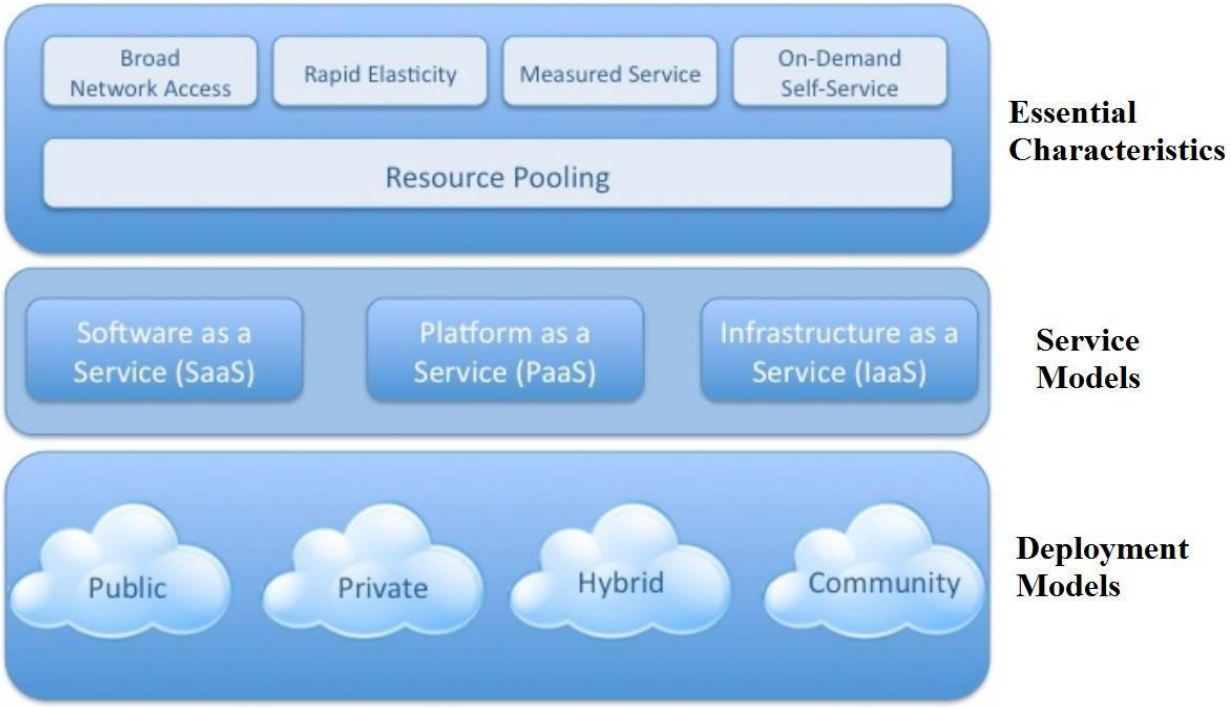


Figure 1.5 Cloud Computing Architecture

Essential Characteristics 2

- Broad network access.
  - Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, laptops, and PDAs) as well as other traditional or cloudbased software services.

Essential Characteristics 3

- Resource pooling.
  - The provider's computing resources are pooled to serve multiple consumers using a **multi-tenant model**, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand.

Essential Characteristics 4

- **Rapid elasticity.**
  - Capabilities can be rapidly and elastically provisioned - in some cases automatically - to quickly scale out; and rapidly released to quickly scale in.
  - To the consumer, the capabilities available for provisioning often appear to be unlimited and can be purchased in any quantity at any time.

Essential Characteristics 5

- **Measured service.**
  - Cloud systems automatically control and optimize resource usage by leveraging a metering capability at some level of abstraction appropriate to the type of service.
  - Resource usage can be monitored, controlled, and reported - providing transparency for both the provider and consumer of the service.

Cloud Service Models

- Cloud **S**oftware as a Service (**S**aaS)
- Cloud **P**latform as a Service (**P**aaS)
- Cloud **I**nfrastructure as a Service (**I**aaS)

SaaS

- SaaS is a licensed software offering on the cloud and pay per use
- SaaS is a software delivery methodology that provides licensed multi-tenant access to software and its functions remotely as a Web-based service. Usually billed based on usage
  - Usually multi tenant environment
  - Highly scalable architecture
- Customers do not invest on software application programs
- The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure.
- The applications are accessible from various client devices through a thin client interface such as a web browser (e.g., web-based email).

- The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, data or even individual application capabilities, with the possible exception of limited user specific application configuration settings.

SaaS providers

- Google's Gmail, Docs, Talk etc
- Microsoft's Hotmail, Sharepoint
- SalesForce,
- Yahoo, Facebook

Infrastructure as a Service (IaaS)

- IaaS is the delivery of technology infrastructure (mostly hardware) as an on demand, scalable service
  - Usually billed based on usage
  - Usually multi tenant virtualized environment
  - Can be coupled with Managed Services for **OS** and application support
  - User can choose his OS, storage, deployed app, networking components
  -

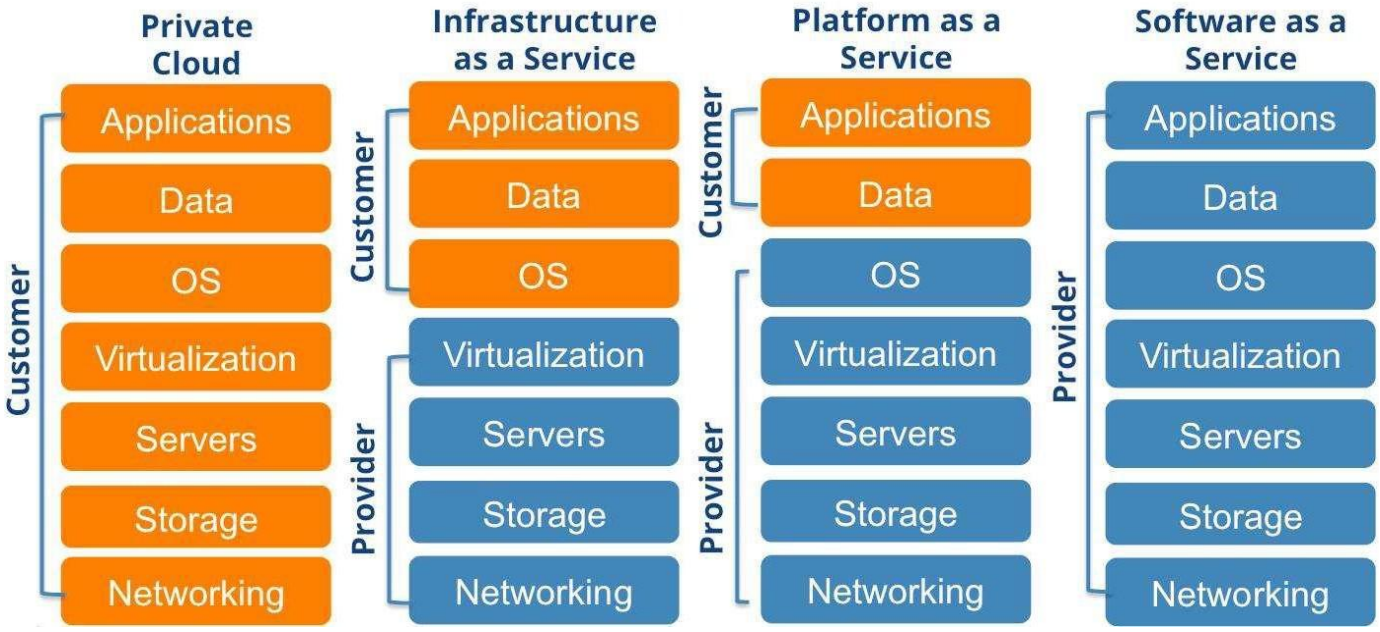


Figure 1.6 Cloud Service Model

- The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources.

- Consumer is able to deploy and run arbitrary software, which may include operating systems and applications.
- The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications, and possibly limited control of select networking components (e.g., host firewalls).

IaaS providers

- Amazon Elastic Compute Cloud (EC2)
  - Each instance provides 1-20 processors, upto 16 GB RAM, 1.69TB storage
- RackSpace Hosting
  - Each instance provides 4 core CPU, upto 8 GB RAM, 480 GB storage
- Joyent Cloud
  - Each instance provides 8 CPUs, upto 32 GB RAM, 48 GB storage
- Go Grid
  - Each instance provides 1-6 processors, upto 15 GB RAM, 1.69TB storage

Platform as a Service (PaaS)

- PaaS provides all of the facilities required to support the complete life cycle of building, delivering and deploying web applications and services entirely from the Internet. Typically applications must be developed with a particular platform in mind
  - Multi tenant environments
  - Highly scalable multi tier architecture
- The capability provided to the consumer is to deploy onto the cloud infrastructure consumer created or acquired applications created using programming languages and tools supported by the provider.
- The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly application hosting environment configurations.

PaaS providers

- Google App Engine
  - Python, Java, Eclipse

- Microsoft Azure
  - .Net, Visual Studio
- Sales Force
  - Apex, Web wizard
- TIBCO,
- VMware,
- Zoho

Cloud Computing - Opportunities and Challenges

- It enables services to be used without any understanding of their infrastructure.
- Cloud computing works using economies of scale
- It potentially lowers the outlay expense for start up companies, as they would no longer need to buy their own software or servers.
- Cost would be by on-demand pricing.
- Vendors and Service providers claim costs by establishing an ongoing revenue stream.
- Data and services are stored remotely but accessible from "anywhere"

Cloud Computing – Pros

- Lower computer costs
- Instant software updates:
  - When the application is web-based, updates happen automatically
- Improved document format compatibility
- e capacity:
  - Cloud computing offers virtually limitless storage
  - • Increased data reliability:

Cloud Computing – Cons

- Need of Internet :
  - A dead Internet connection means no work and in areas where Internet connections are few or inherently unreliable, this could be a deal-breaker.
  - Requires a constant Internet connection

- Can be slow:
  - Even with a fast connection, web-based applications can sometimes be slower than accessing a similar software program on your desktop PC.
- Disparate Protocols :
  - Each cloud systems uses different protocols and different APIs – Standards yet to evolve.

Evolution of Cloud Computing

Evolution of Cloud Computing

- Cloud Computing Leverages dynamic resources to deliver a large number of services to end users.
- It is High Throughput Computing(HTC) paradigm
- It enables users to share access to resources from anywhere at any time

II Hardware Evolution

- In 1930, binary arithmetic was developed
  - computer processing technology, terminology, and programming languages.

• In 1939,Electronic computer was developed

- Computations were performed using vacuum-tube technology.

• In 1941, Konrad Zuse's Z3 was developed

- Support both floating-point and binary arithmetic.

There are four generations

- First Generation Computers
- Second Generation Computers
- Third Generation Computers
- Fourth Generation Computers

a.First Generation Computers

Time Period : 1942 to 1955

Technology : Vacuum Tubes

Size : Very Large System

Processing : Very Slow

Examples:

1.ENIAC (Electronic Numerical Integrator and Computer)

2.EDVAC(Electronic Discrete Variable Automatic Computer)

Advantages:

• It made use of vacuum tubes which was the advanced technology at that time

• Computations were performed in milliseconds.

Disadvantages:

• very big in size, weight was about 30 tones.

• very costly.

• Requires more power consumption

•Large amount heat was generated.

b.Second Generation Computers

Time Period : 1956 to 1965.

Technology : Transistors

Size : Smaller

Processing : Faster

o Examples

Honeywell 400

IBM 7094

Advantages

- Less heat than first generation.
- Assembly language and punch cards were used for input.
- Low cost than first generation computers.
- Computations was performed in microseconds.
- Better Portability as compared to first generation

Disadvantages:

- A cooling system was required.
- Constant maintenance was required.
- Only used for specific purposes

c.Third Generation Computers

Time Period : 1966 to 1975

Technology : ICs (Integrated Circuits)

Size : Small as compared to 2nd generation computers

Processing : Faster than 2nd generation computers

Examples

• PDP-8 (Programmed Data Processor)

• PDP-11

Advantages

• These computers were cheaper as compared to generation computers.

• They were fast and reliable.

• IC not only reduce the size of the computer but it also improves the performance of the computer

• Computations was performed in nanoseconds

Disadvantages

• IC chips are difficult to maintain.

• The highly sophisticated technology required for the manufacturing of IC chips.

•Air Conditioning is required

d.Fourth Generation Computers

Time Period : 1975 to Till Date

Technology : Microprocessor

Size : Small as compared to third generation computer

Processing : Faster than third generation computer

Examples

• IBM 4341

• DEC 10

Advantages:

- Fastest in computation and size get reduced as compared to the previous generation of computer. Heat generated is small.
- Less maintenance is required.

Disadvantages:

- The Microprocessor design and fabrication are very complex.
- Air Conditioning is required in many cases

III Internet Hardware Evolution

- Internet Protocol is the standard communications protocol used by every computer on the Internet.
- The conceptual foundation for creation of the Internet was significantly developed by three individuals.
  - Vannevar Bush — MEMIX (1930)
  - Norbert Wiener
  - Marshall McLuhan
- Licklider was founder for the creation of the AR PANET (Advanced Research Projects Agency Network)
- Clark deployed a minicomputer called an Interface Message Processor (IMP) at each site.
- Network Control Program (NCP)- first networking protocol that was used on the ARPANET

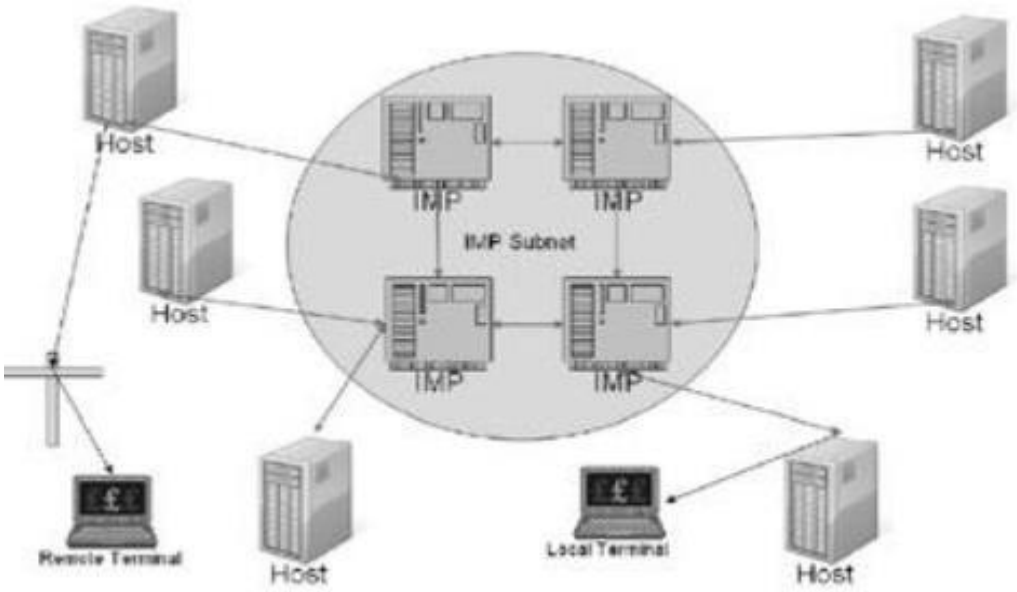


Figure 1.7 **IMP Architecture**

Internet Hardware Evolution

- Establishing a Common Protocol for the Internet
- Evolution of Ipv6
- Finding a Common Method to Communicate Using the Internet Protocol
- Building a Common Interface to the Internet
- The Appearance of Cloud Formations From One Computer to a Grid of Many

a.Establishing a Common Protocol for the Internet

- NCP essentially provided a transport layer consisting of the ARPANET Host-to-Host Protocol (AIIIIP) and the Initial Connection Protocol (ICP)
- Application protocols
    - File Transfer Protocol (FTP), used for file transfers,
    - Simple Mail Transfer Protocol (SMTP), used for sending email

Four versions of TCP/IP

- TCP vl
- TCP v2
- TCP v3 and IP v3,
- TCP v4 and IP v4

b.Evolution of Ipv6

- IPv4 was never designed to scale to global levels.
- To increase available address space, it had to process large data packets (i.e., more bits of data).
- To overcome these problems, Internet Engineering Task Force (IETF) developed IPv6, which was released in January 1995.
- Ipv6 is sometimes called the Next Generation Internet Protocol (IPNG) or TCP/IP v6.

c.Finding a Common Method to Communicate Using the Internet Protocol

- In the 1960s,the word ktpertext was created by Ted Nelson.
- In 1962, Engelbart's first project was Augment, and its purpose was to develop computer tools to augment human capabilities.
- He developed the mouse, Graphical user interface (GUI), and the first working hypertext system, named NLS (oN-Line System).

- NLS was designed to cross-reference research papers for sharing among geographically distributed researchers.
- In the 1980s, Web was developed in Europe by Tim Berners-Lee and Robert Cailliau

uilding a Common Interface to the Internet

- Betters-Lee developed the first web browser featuring an integrated editor that could create hypertext documents.
- Following this initial success, Berners-Lee enhanced the server and browser by adding support for the FTP (File Transfer protocol)

Figure 1.8 First Web Browser

- Mosaic was the first widely popular web browser available to the general public. Mosaic support for graphics, sound, and video clips.
- In October 1994, Netscape released the first beta version of its browser, Mozilla 0.96b, over the Internet.
- In 1995, Microsoft Internet Explorer was developed that supports both a graphical Web browser and the name for a set of technologies.
- Mozilla Firefox. released in November 2004, became very popular almost immediately.

e.The Appearance of Cloud Formations From One Computer to a Grid of Many

- Two decades ago, computers were clustered together to form a single larger computer in order to simulate a supercomputer and greater processing power.
- In the early 1990s, Ian Foster and Carl Kesselman presented their concept of "The Grid." They used an analogy to the electricity grid, where users could plug in and use a (metered) utility service.
- A major problem in clustering model was data residency. Because of the distributed nature of a grid, computational nodes could be anywhere in the world.

- The Globus Toolkit is an open source software toolkit used for building grid systems and applications

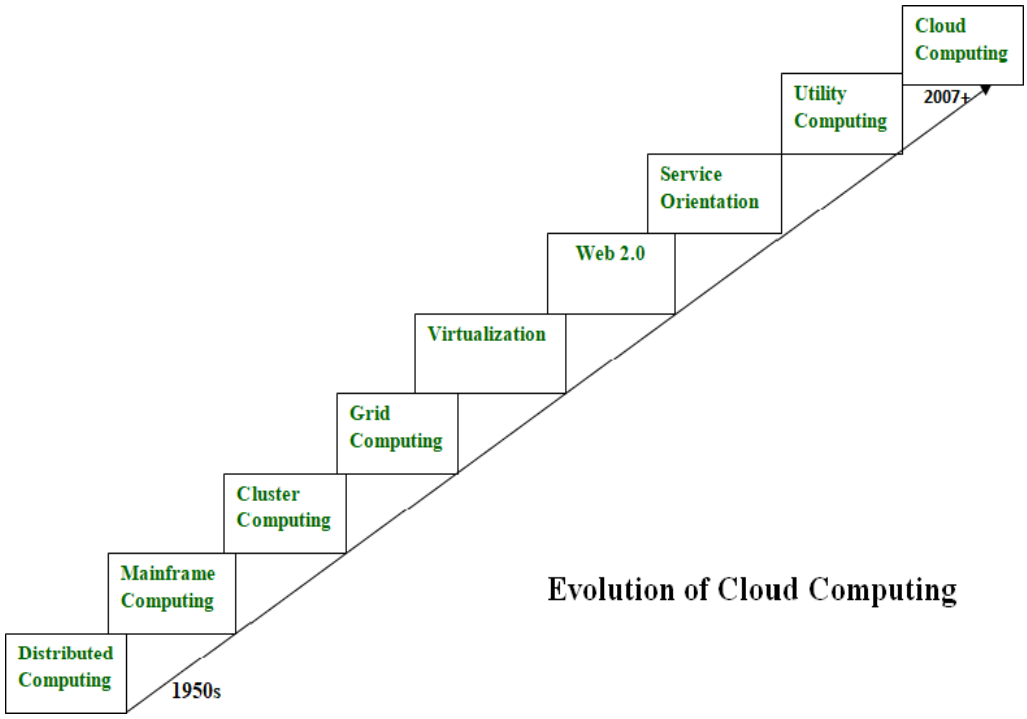


Figure 1.9 Evolution

Evolution of Cloud Services

2008-2009	**Google Application Engine** **Microsoft Azure**
2006	**S3 launches EC2**
2002	**Launch of Amazon Web Services**
1990	**The first milestone of cloud computing arrival of salesforce.com**
1960	**Super Computers** **Mainframes**

IV. SERVER VIRTUALIZATION

- Virtualization is a method of running multiple independent virtual operating systems on a single physical computer.
- This approach maximizes the return on investment for the computer.

- Virtualization technology is a way of reducing the majority of hardware acquisition and maintenance costs, which can result in significant savings for any company.
    - Parallel Processing
    - Vector Processing
    - Symmetric Multiprocessing Systems
    - Massively Parallel Processing Systems

a.Parallel Processing

- Parallel processing is performed by the simultaneous execution of program instructions that have been allocated across multiple processors.
- Objective: running a progran in less time.
- The next advancement in parallel processing-multiprogramming
- In a multiprogramming system, multiple programs submitted by users but each allowed to use the processor for a short time.
- This approach is known as "round-robin scheduling"(RR scheduling)

b.Vector Processing

- Vector processing was developed to increase processing performance by operating in a multitasking manner.
- Matrix operations were added to computers to perform arithmetic operations.
- This was valuable in certain types of applications in which data occurred in the form of vectors or matrices.
- In applications with less well-formed data, vector processing was less valuable.

c.Symmetric Multiprocessing Systems

- Symmetric multiprocessing systems (SMP) was developed to address the problem of resource management in master/slave models.
- In SMP systems, each processor is equally capable and responsible for managing the workflow as it passes through the system.
- The primary goal is to achieve sequential consistency

d.Massively Parallel Processing Systems

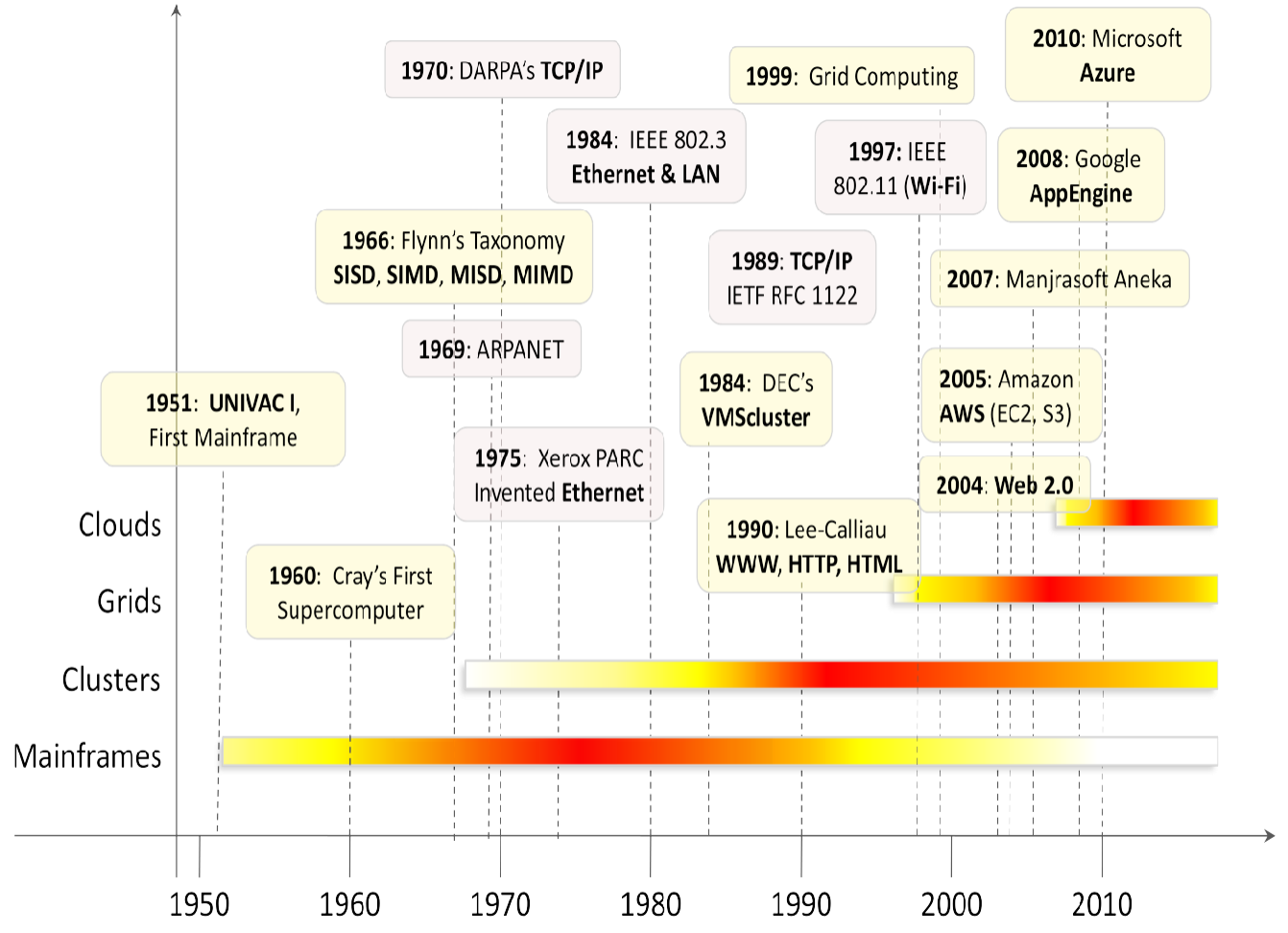
- In Massively Parallel Processing Systems, a computer system with many independent arithmetic units, which run in parallel.
- All the processing elements are interconnected to act as one very large computer.

- Early examples of MPP systems were the Distributed ArrayProcessor, the Goodyear MPP, the Connection Machine, and the Ultracomputer
- MPP machines are not easy to program, but for certain applications, such as data mining, they are the best solution

Principles of Parallel and Distributed Computing

- **Three major milestones have led to cloud computing evolution**
  - Mainframes: Large computational facilities leveraging multiple processing units. Even though mainframes cannot be considered as distributed systems, they offered large computational power by using multiple processors, which were presented as a single entity to users.

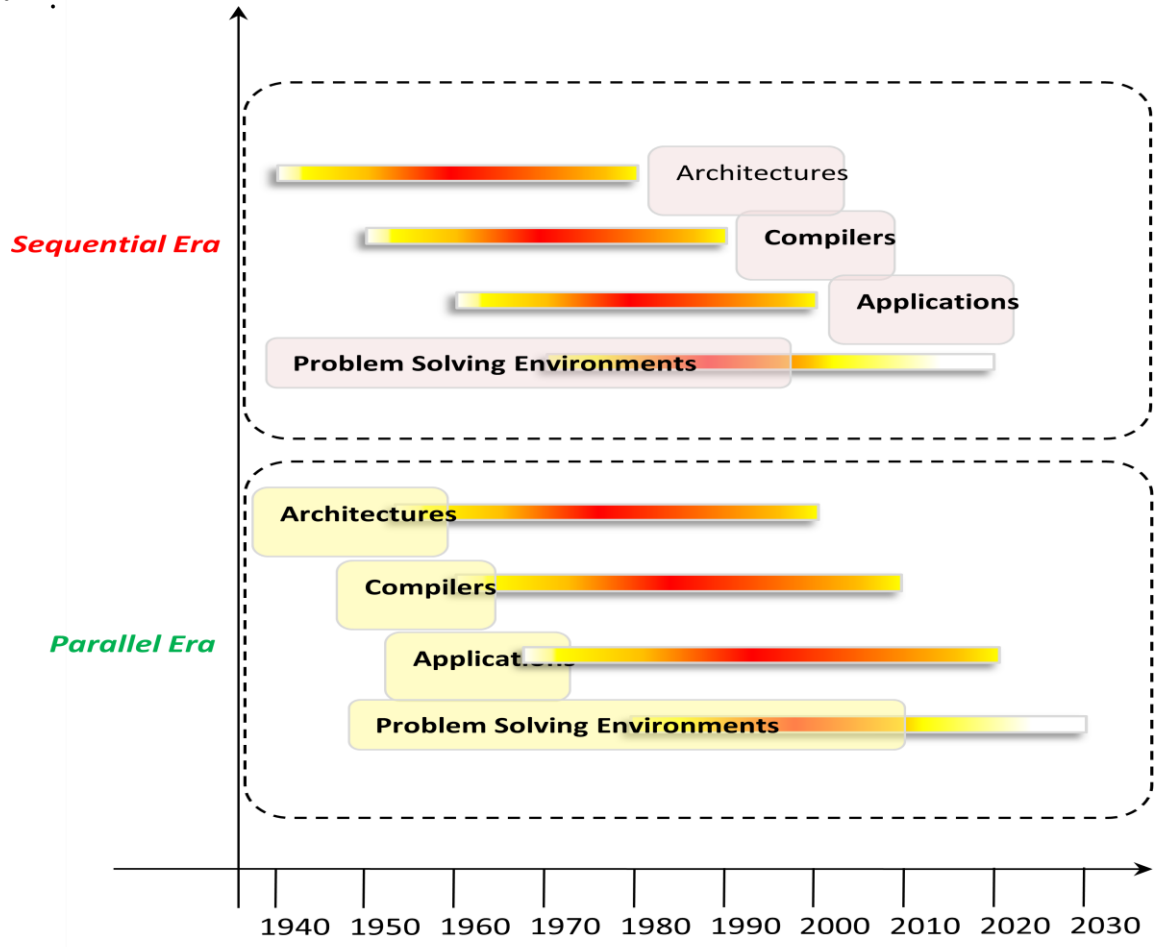
Mile Stones to Cloud computing Evolution



  - Clusters: An alternative technological advancement to the use of mainframes and super computers.
  - Grids
- Clouds

Eras of Computing

- Two fundamental and dominant models of computing are ***sequential*** and ***parallel***.
  - The sequential era began in the 1940s, and Parallel(and distributed) computing era followed it within a decade.
- Four key elements of computing developed during three eras are
  - Architecture
  - Compilers
  - Applications
  - Problem solving environments
- .



- The computing era started with development in ***hardware architectures***, which actually enabled the creation of ***system software*** – particularly in the area of **compilers and operating systems** – which support the management of such systems and the development of ***applications***

- The term parallel computing and distributed computing are **often used interchangeably**, even though they mean **slightly different things**.
- The term **parallel implies a tightly coupled system**, where as **distributed systems refers to a wider class of system, including those that are tightly coupled**.
- More precisely, the term **parallel computing** refers to a model in which **the computation is divided among several processors sharing the same memory**.
- The architecture of **parallel computing system** is often characterized by the **homogeneity of components**: **each processor is of the same type and it has the same capability as the others.**

- The shared memory has a single address space, which is accessible to all the processors.
- Parallel programs are then broken down into several units of execution that can be allocated to different processors and can communicate with each other by means of shared memory.
- Originally parallel systems are considered as those architectures that featured multiple processors sharing the same physical memory and that were considered a single computer.
  - Over time, these restrictions have been relaxed, and parallel systems now include all architectures that are based on the concept of shared memory, whether this is physically present or created with the support of libraries, specific hardware, and a highly efficient networking infrastructure.
  - For example: a cluster of which of the nodes are connected through an InfiniBand network and configured with distributed shared memory system can be considered as a parallel system.
- The term distributed computing encompasses any architecture or system that allows the computation to be broken down into units and executed concurrently on different

computing elements, whether these are processors on different nodes, processors on the same computer, or cores within the same processor.

- Distributed computing includes a wider range of systems and applications than parallel computing and is often considered a more general term.
- Even though it is not a rule, the term distributed often implies that the locations of the computing elements are not the same and such elements might be heterogeneous in terms of hardware and software features.
- Classic examples of distributed computing systems are
  - Computing Grids
  - Internet Computing Systems

Elements of Parallel computing

- Silicon-based processor chips are reaching their physical limits. Processing speed is constrained by the speed of light, and the density of transistors packaged in a processor is constrained by thermodynamics limitations.
- A viable solution to overcome this limitation is to connect multiple processors working in coordination with each other to solve "Grand Challenge" problems.
- The first step in this direction led
  - To the development of parallel computing, which encompasses techniques, architectures, and systems for performing multiple activities in parallel.

a.Parallel Processing

- Processing of multiple tasks simultaneously on multiple processors is called ***parallel processing***.
- The parallel program consists of multiple active processes (tasks) simultaneously solving a given problem.
- A given task is divided into multiple subtasks using a divide-and-conquer technique, and each subtask is processed on a different central processing unit (CPU).
- Programming on multi processor system using the divide-and-conquer technique is called ***parallel programming***.
- Many applications today require more computing power than a traditional sequential computer can offer.

- Parallel Processing provides a cost effective solution to this problem by increasing the number of CPUs in a computer and by adding an efficient communication system between them.
- The workload can then be shared between different processors. This setup results in higher computing power and performance than a single processor a system offers.

Parallel Processing influencing factors

- The development of parallel processing is being influenced by many factors. The prominent among them include the following:
  - Computational requirements are ever increasing in the areas of both scientific and business computing. The technical computing problems, which require high-speed computational power, are related to
    - life sciences, aerospace, geographical information systems, mechanical design and analysis etc.
  - Sequential architectures are reaching mechanical physical limitations as they are constrained by the speed of light and thermodynamics laws.
    - The speed which sequential CPUs can operated is reaching saturation point (no more vertical growth), and hence an alternative way to get high computation speed is to connect multiple CPUs (opportunity for horizontal growth).
  - Hardware improvements in pipelining , super scalar, and the like are non scalable and require sophisticated compiler technology.
    - Developing such compiler technology is a difficult task.
  - Vector processing works well for certain kinds of problems. It is suitable mostly for scientific problems (involving lots of matrix operations) and graphical processing.
    - It is not useful for other areas, such as databases.
  - The technology of parallel processing is mature and can be exploited commercially
    - here is already significant R&D work on development tools and environments.
  - Significant development in networking technology is paving the way for

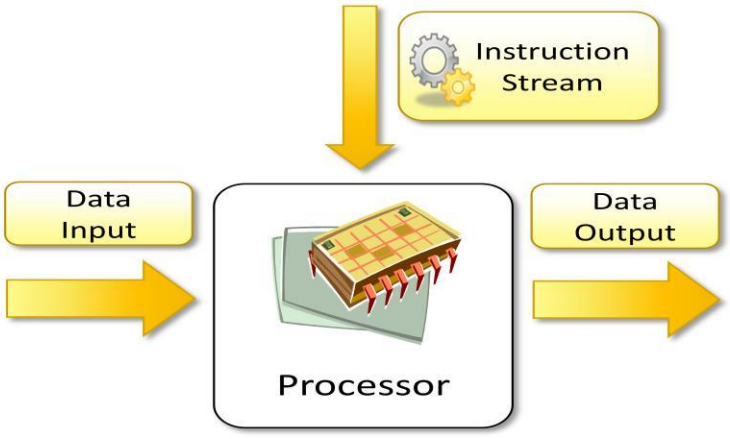
- heterogeneous computing.

ardware architectures for parallel Processing

- The core elements of parallel processing are CPUs. Based on the number of instructions and data streams, that can be processed simultaneously, computing systems are classified into the following four categories:
  - Single-instruction, Single-data (SISD) systems
  - Single-instruction, Multiple-data (SIMD) systems
  - Multiple-instruction, Single-data (MISD) systems
  - Multiple-instruction, Multiple-data (MIMD) systems

(i) Single – Instruction , Single Data (SISD) systems

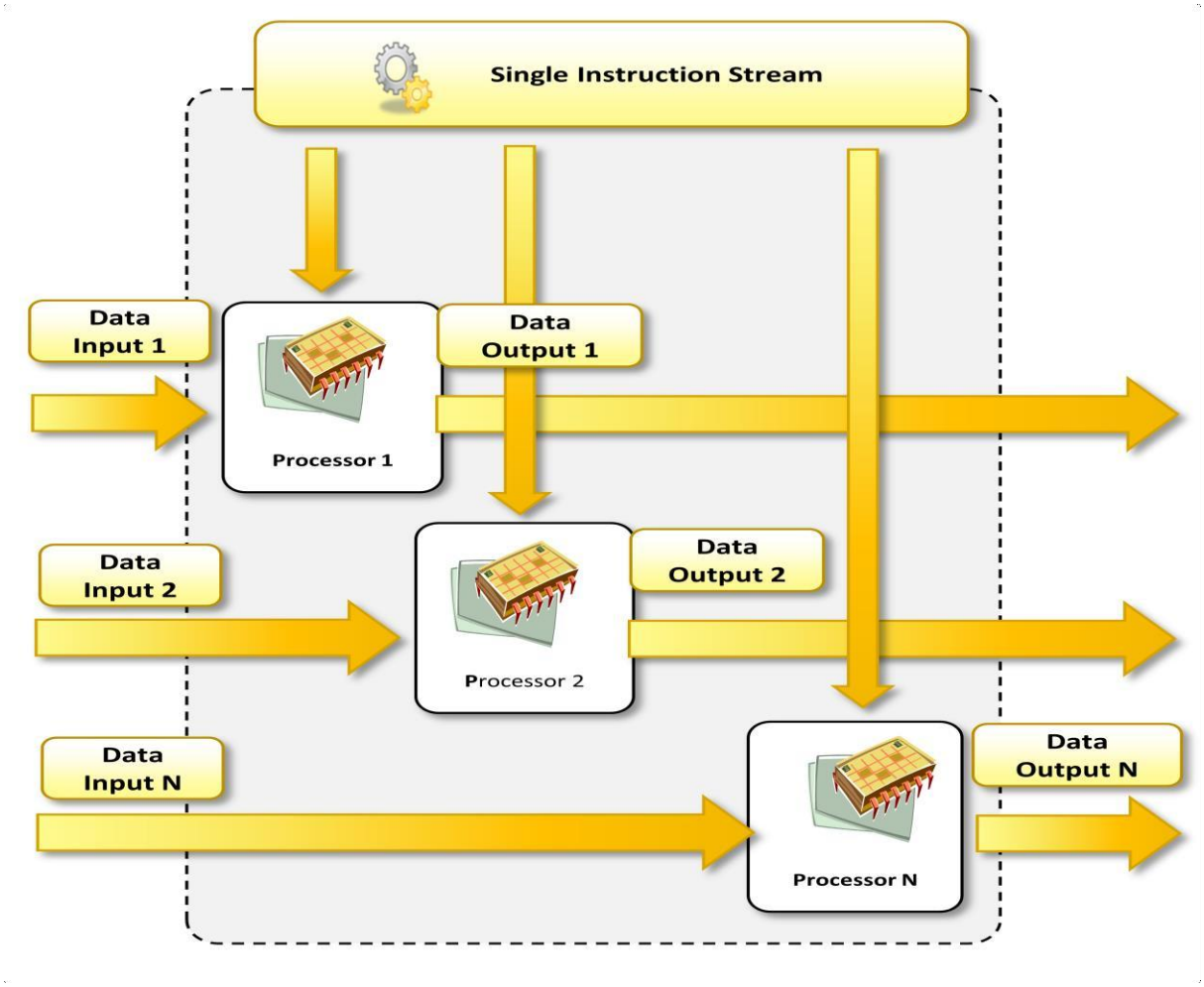
- SISD computing system is a uni-processor machine capable of executing a single instruction, which operates on a single data stream.
- Machine instructions are processed sequentially, hence computers adopting this model are popularly called sequential computers.
- Most conventional computers are built using SISD model.
- All the instructions and data to be processed have to be stored in primary memory.
- The speed of processing element in the SISD model is limited by the rate at which the computer can transfer information internally.
- Dominant representative SISD systems are IBM PC, Macintosh, and workstations.



(ii) Single – Instruction , Multiple Data (SIMD) systems

- SIMD computing system is a multiprocessor machine capable of executing the same instruction on all the CPUs but operating on different data streams.
- Machines based on this model are well suited for scientific computing since they involve lots of vector and matrix operations.
- For instance statement Ci = Ai * Bi, can be passed to all the processing elements (PEs), organized data elements of vectors A and B can be divided into multiple sets (N- sets for N PE systems), and each PE can process one data set.

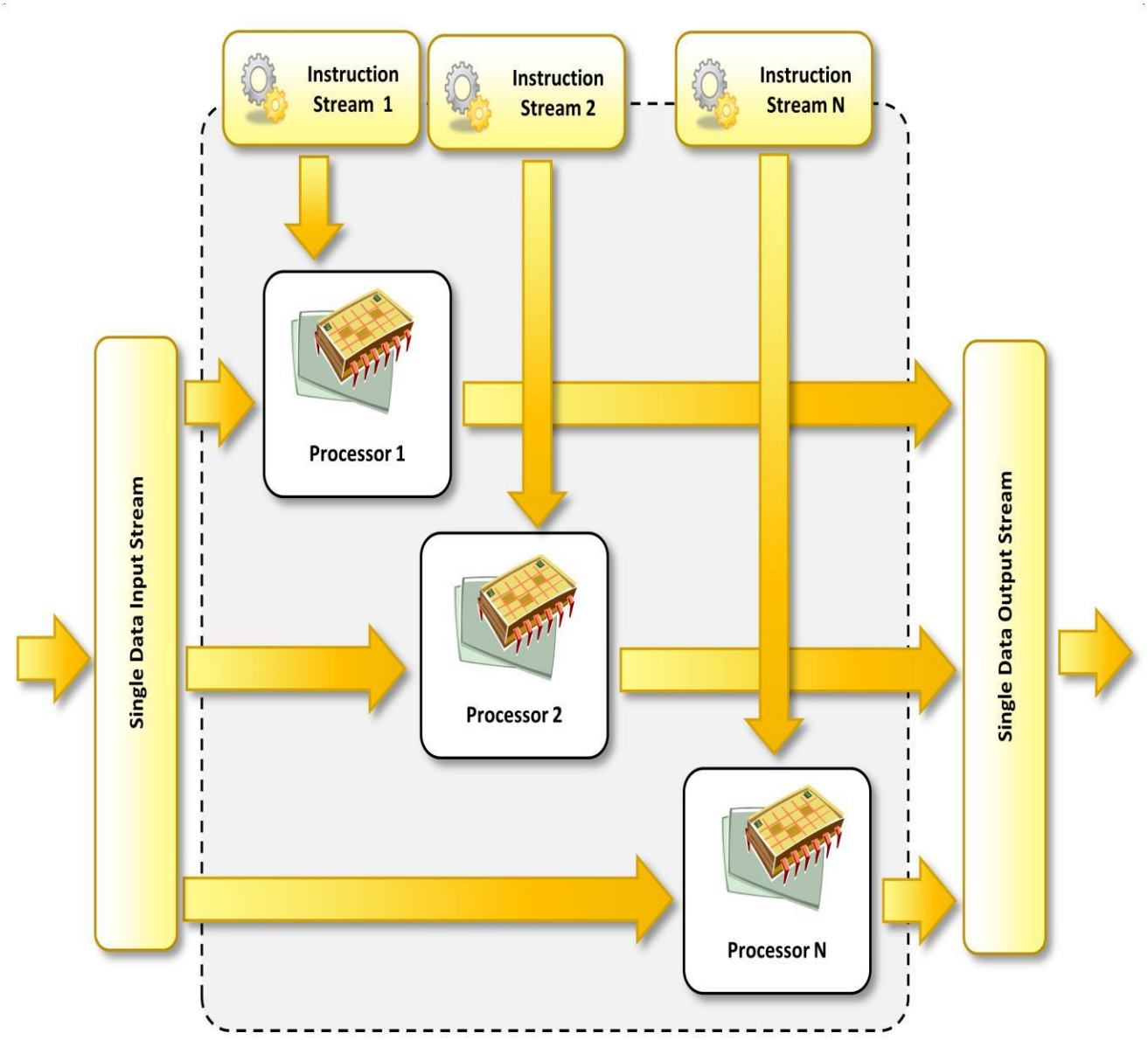
Dominant representative SIMD systems are Cray's Vector processing machine and Thinking Machines Cm*, and GPGPU accelerators



-

(iii) Multiple – Instruction , Single Data (MISD) systems

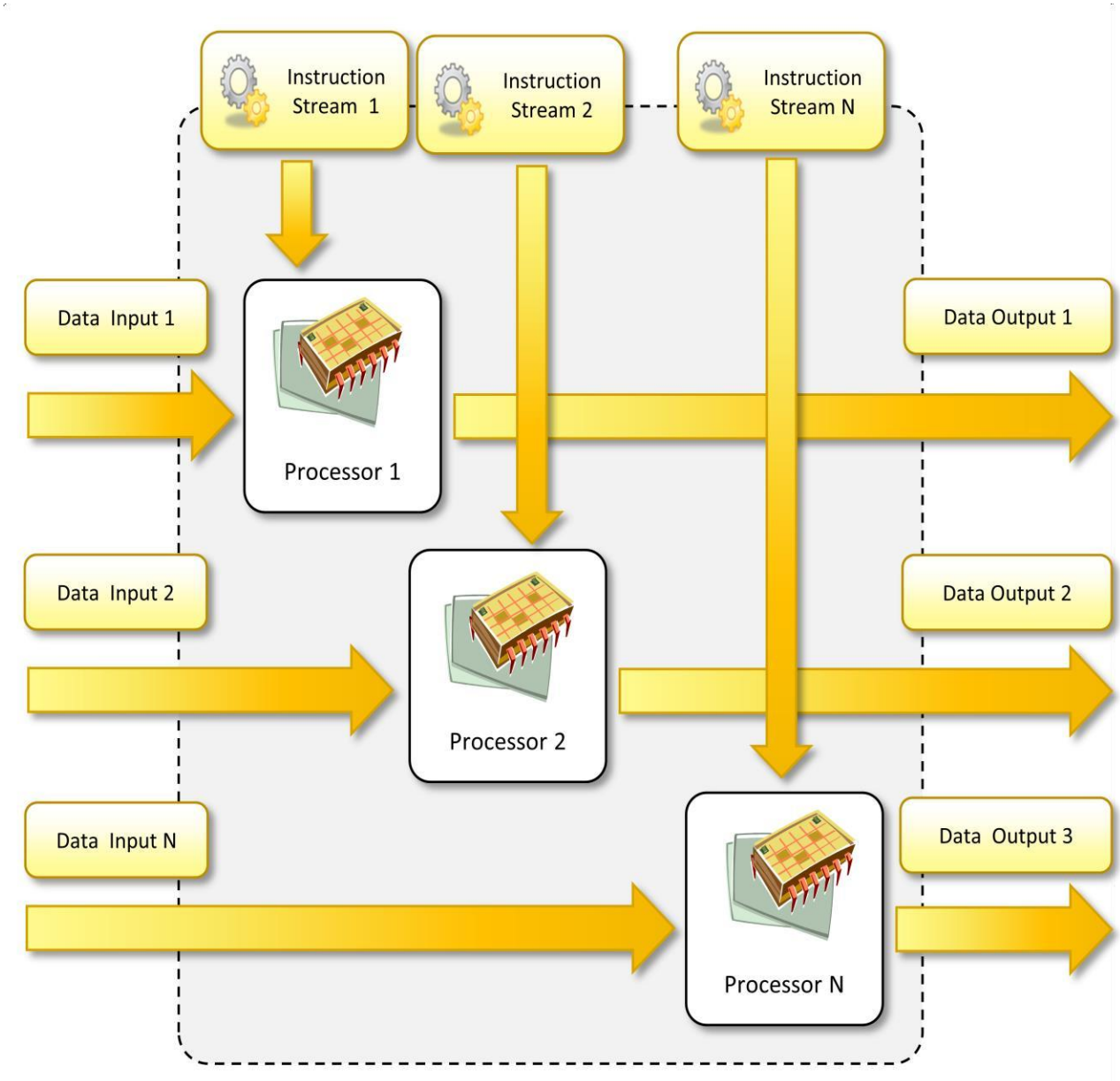
- MISD computing system is a multi processor machine capable of executing different instructions on different Pes all of them operating on the same data set.
- Machines built using MISD model are not useful in most of the applications.
- Few machines are built but none of them available commercially.
- This type of systems are more of an intellectual exercise than a practical configuration.



(iv) Multiple – Instruction , Multiple Data (MIMD) systems

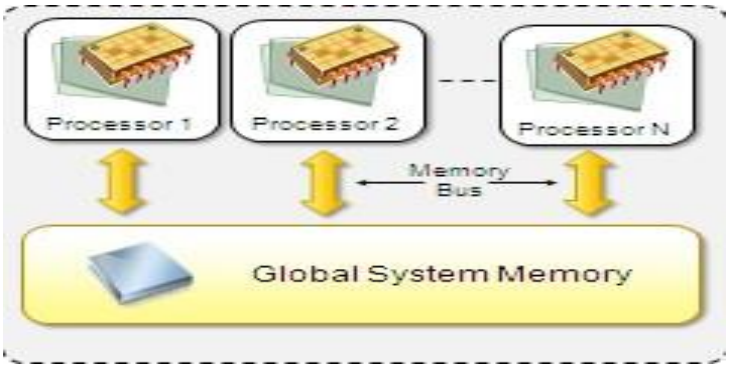
- MIMD computing system is a multi processor machine capable of executing multiple instructions on multiple data sets.
- Each PE in the MIMD model has separate instruction and data streams, hence machines built using this model are well suited to any kind of application.
- Unlike SIMD, MISD machine, PEs in MIMD machines work asynchronously,

MIMD machines are broadly categorized into shared-memory MIMD and distributed memory MIMD based on the way PEs are coupled to the main memory



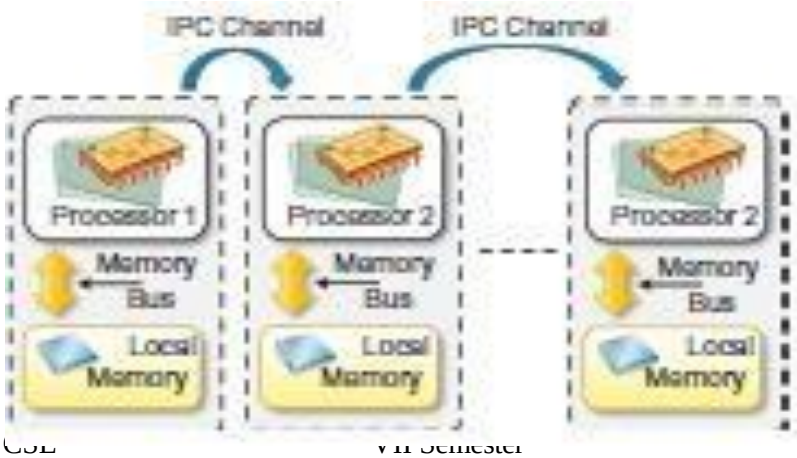
Shared Memory MIMD machines

- All the PEs are connected to a single global memory and they all have access to it.
- Systems based on this model are also called tightly coupled multi processor systems.
- The communication between PEs in this model takes place through the shared memory.
- Modification of the data stored in the global memory by one PE is visible to all other PEs.
- Dominant representative shared memory MIMD systems are silicon graphics machines and Sun/IBM SMP (Symmetric Multi-Processing).



Distributed Memory MIMD machines

- All PEs have a local memory. Systems based on this model are also called loosely coupled multi processor systems.
- The communication between PEs in this model takes place through the interconnection network, the inter process communication channel, or IPC.
- The network connecting PEs can be configured to tree, mesh, cube, and so on.
- Each PE operates asynchronously, and if communication/synchronization among tasks is necessary, they can do so by exchanging messages between them.



Shared Vs Distributed MIMD model

- The shared memory MIMD architecture is easier to program but is less tolerant to failures and harder to extend with respect to the distributed memory MIMD model.
- Failures, in a shared memory MIMD affect the entire system, whereas this is not the case of the distributed model, in which each of the PEs can be easily isolated.
- Moreover, shared memory MIMD architectures are less likely to scale because the addition of more PEs leads to memory contention.
- This is a situation that does not happen in the case of distributed memory, in which each PE has its own memory.

As a result, distributed memory MIMD architectures are most popular today

pproaches to Parallel Programming

- A sequential program is one that runs on a single processor and has a single line of control.
- To make many processors collectively work on a single program, the program must be divided into smaller independent chunks so that each processor can work on separate chunks of the problem.
- The program decomposed in this way is a parallel program.
- A wide variety of parallel programming approaches are available.
- The most prominent among them are the following.
- Data Parallelism
- Process Parallelism
- Farmer-and-worker model
- The above said three models are suitable for task-level parallelism. In the case of data level parallelism, the divide-and-conquer technique is used to split data into multiple sets, and each data set is processed on different PEs using the same instruction.
- This approach is highly suitable to processing on machines based on the SIMD model.
- In the case of Process Parallelism, a given operation has multiple (but distinct) activities that can be processed on multiple processors.
- In the case of Farmer-and-Worker model, a job distribution approach is used, one processor is configured as master and all other remaining PEs are designated as slaves,

the master assigns the jobs to slave PEs and, on completion, they inform the master, which in turn collects results.

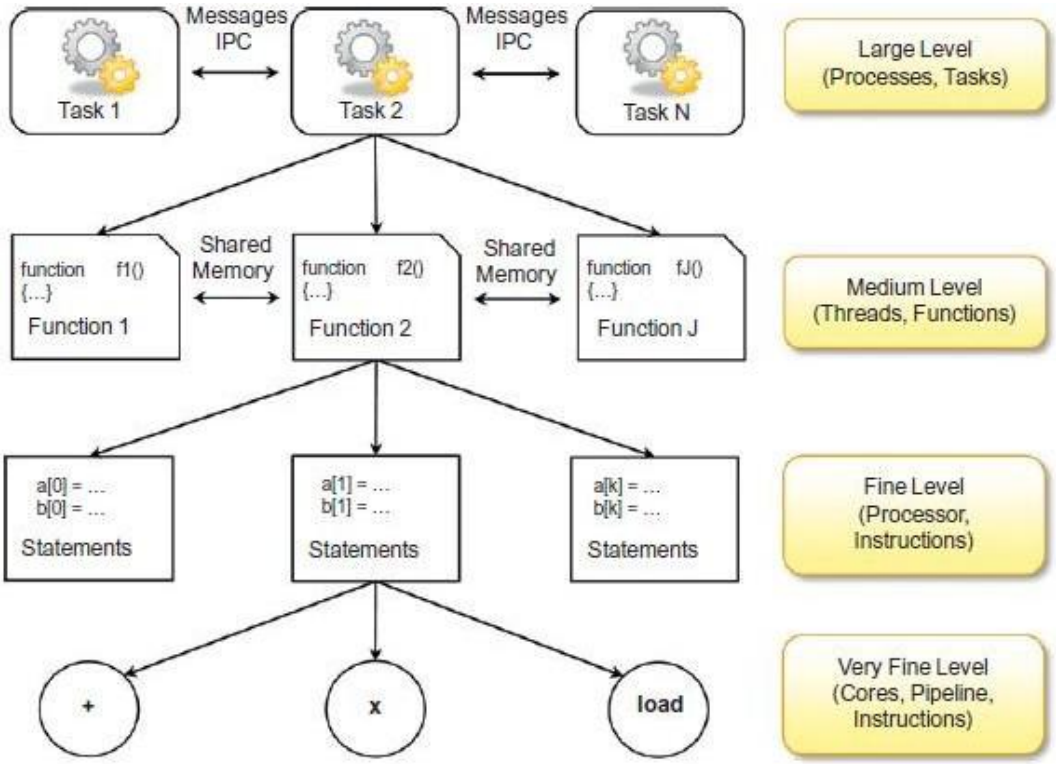
- These approaches can be utilized in different levels of parallelism.

d. Levels of Parallelism

- Levels of Parallelism are decided on the lumps of code (grain size) that can be a potential candidate of parallelism.
- The table shows the levels of parallelism.
- All these approaches have a common goal
  - To boost processor efficiency by hiding latency.
  - To conceal latency, there must be another thread ready to run whenever a lengthy operation occurs.
- The idea is to execute concurrently two or more single-threaded applications. Such as compiling, text formatting, database searching, and device simulation.

Grain Size	Code Item	Parallelized By
Large	Separate and heavy weight process	Programmer
Medium	Function or procedure	Programmer
Fine	Loop or instruction block	Parallelizing compiler
Very Fine	Instruction	Processor

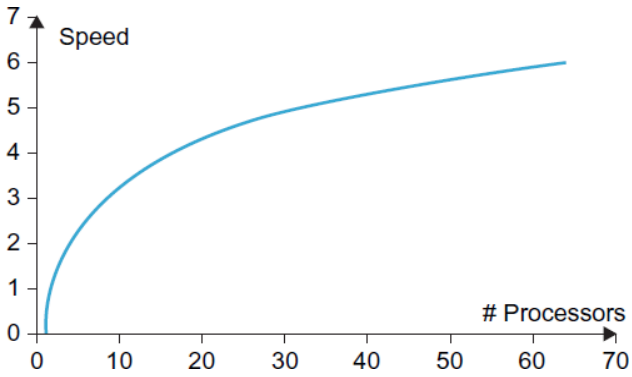
Levels of Parallelism



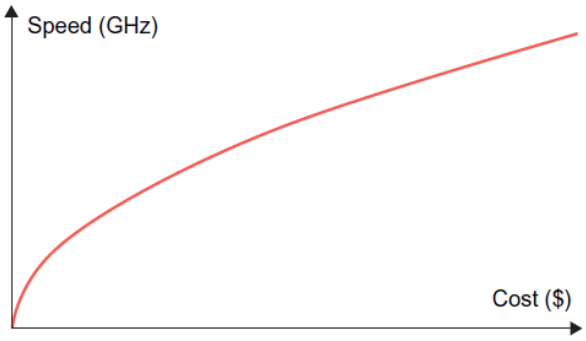
e. Laws of Caution

- Studying how much an application or a software system can gain from parallelism.
- In particular what need to keep in mind is that parallelism is used to perform multiple activities together so that the system can increase its throughput or its speed.
- But the relations that control the increment of speed are not linear.
- For example: for a given n processors, the user expects speed to be increase by in times. This is an ideal situation, but it rarely happens because of the communication overhead.
- Here two important guidelines to take into account.
  - Speed of computation is proportional to the square root of the system cost; they never increase linearly. Therefore, the faster a system becomes, the more expensive it is to increase its speed

– Speed by a parallel computer increases as the logarithm of the number of processors (i.e. y=k*log(N)).



Number processors versus speed



Cost versus speed

Elements of Distributed Computing

a.General concepts and definitions

- Distributed computing studies the models, architectures, and algorithms used for building and managing distributed systems.
- As general definition of the term distributed system, we use the one proposed by Tanenbaum
  - A distributed system is a collection of independent computers that appears to its users as a single coherent system.
- This definition is general enough to include various types of distributed computing systems that are especially focused on unified usage and aggregation of distributed resources.
- Communications is another fundamental aspect of distributed computing. Since distributed systems are composed of more than one computer that collaborate together, it is necessary to provide some sort of data and information exchange between them, which generally occurs through the network.
  - A distributed system is one in which components located at networked computers communicate and coordinate their action only by passing messages.
- As specified in this definition, the components of a distributed system communicate with some sort of message passing. This is a term that encompasses several communication models.

omponents of distributed System

- A distributed system is the result of the interaction of several components that traverse the entire computing stack from hardware to software.
- It emerges from the collaboration of several elements that- by working together- give users the illusion of a single coherent system.
- The figure provides an overview of the different layers that are involved in providing the services of a distributed system.

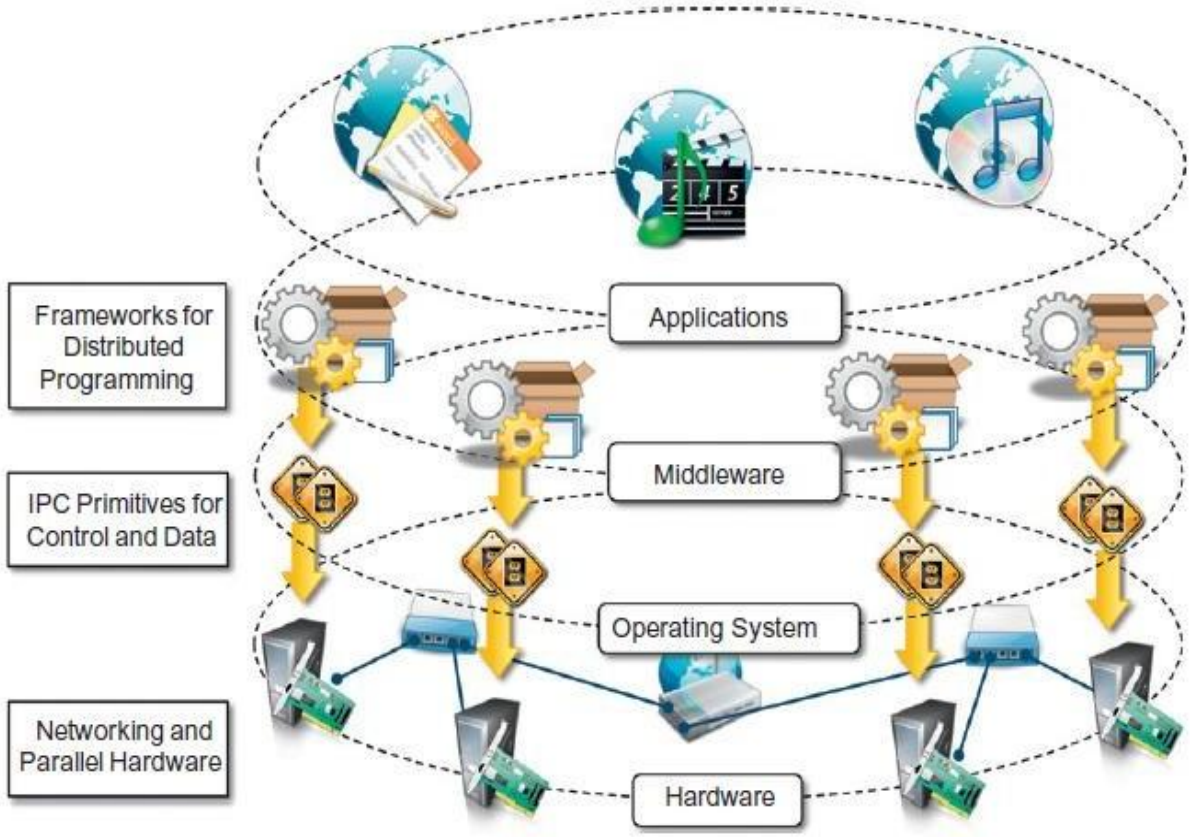


Figure 1.10 A layered view of a distributed system.

c.Architectural styles for distributed computing

- At the very bottom layer, computer and network hardware constitute the physical infrastructure; these components are directly managed by the operating system, which

provides the basic services for inter process communication (IPC), process scheduling and management, and resource management in terms of file system and local devices.

- Taken together these two layers become the platform on top of which specialized software is deployed to turn a set of networked computers into a distributed system
- Although a distributed system comprises the interaction of several layers, the middleware layer is the one that enables distributed computing, because it provides a coherent and uniform runtime environment for applications.
- There are many different ways to organize the components that, taken together, constitute such an environment.
- The interactions among these components and their responsibilities give structure to the middleware and characterize its type or, in other words, define its architecture.
- Architectural styles aid in understanding the classifying the organization of the software systems in general and distributed computing in particular.
- The use of well-known standards at the operating system level and even more at the hardware and network levels allows easy harnessing of heterogeneous components and their organization into a coherent and uniform system.
- For example; network connectivity between different devices is controlled by standards, which allow them into interact seamlessly.
- Design patterns help in creating a common knowledge within the community of software engineers and developers as to how to structure the relevant of components within an application and understand the internal organization of software applications.
- Architectural styles do the same for the overall architecture of software systems.
- The architectural styles are classified into two major classes
    - Software Architectural styles : Relates to the logical organization of the software.
    - System Architectural styles: styles that describe the physical organization of distributed software systems in terms of their major components.

Software Architectural Styles

- Software architectural styles are based on the logical arrangement of software components.
- They are helpful because they provide an intuitive view of the whole system, despite its physical deployment.

- They also identify the main abstractions that are used to shape the components of the system and the expected interaction patterns between them.

Data Centered Architectures

- These architectures **identify the data as the fundamental element of the software system,** and access to shared data is the core characteristics of the data-centered architectures.
- Within the context of distributed and parallel computing systems, **integrity of data is overall goal for such systems.**
- The **repository architectural style** is the most relevant reference model in this category. It is characterized by two main components – the central data structure, which represents the current state of the system, and a collection of independent component, which operate on the central data.
- The ways in which the independent components interact with the central data structure can be very heterogeneous.
- In particular repository based architectures differentiate and specialize further into subcategories according to the choice of control discipline to apply for the shared data structure. Of particular interest are databases and blackboard systems.

Black board Architectural Style

- The black board architectural style is characterized by three main components:
  - Knowledge sources: These are entities that update the knowledge base that is maintained in the black board.
  - Blackboard: This represents the data structure that is shared among the knowledge sources and stores the knowledge base of the application.
  - Control: The control is the collection of triggers and procedures that govern the interaction with the blackboard and update the status of the knowledge base.

Data Flow Architectures

- Access to data is the core feature; data-flow styles explicitly incorporate the pattern of data-flow, since their design is determined by an orderly motion of data from component to component, which is the form of communication between them.
- Styles within this category differ in one of the following ways: how the control is exerted, the degree of concurrency among components, and the topology that describes the flow of data.

- **Batch Sequential:** The batch sequential style is characterized by an ordered sequence of separate programs executing one after the other. These programs are chained together by providing as input for the next program the output generated by the last program after its completion, which is most likely in the form of a file. This design was very popular in the mainframe era of computing and still finds applications today. For example, many distributed applications for scientific computing are defined by jobs expressed as sequence of programs that, for example, pre-filter, analyze, and post process data. It is very common to compose these phases using the batch sequential style.
- **Pipe-and-Filter Style:** It is a variation of the previous style for expressing the activity of a software system as sequence of data transformations. Each component of the processing chain is called a filter, and the connection between one filter and the next is represented by a data stream.

Virtual Machine architectures

- The virtual machine class of architectural styles is characterized by the presence of an abstract execution environment (generally referred as a virtual machine) that simulates features that are not available in the hardware or software.
- Applications and systems are implemented on top of this layer and become portable over different hardware and software environments.
- The general interaction flow for systems implementing this pattern is – the program (or the application) defines its operations and state in an abstract format, which is interpreted by the virtual machine engine. The interpretation of a program constitutes its execution. It is quite common in this scenario that the engine maintains an internal representation of the program state.
- Popular examples within this category are rule based systems, interpreters, and command language processors.
- **Rule-Based Style:**
    - This architecture is characterized by representing the abstract execution environment as an inference engine. Programs are expressed in the form of rules or predicates that hold true. The input data for applications is generally represented by a set of assertions or facts that the inference engine uses to activate rules or to apply predicates, thus transforming data. The examples of rule-based systems can be found in the networking domain: Network Intrusion Detection

Systems (NIDS) often rely on a set of rules to identify abnormal behaviors connected to possible intrusion in computing systems.

- Interpreter Style: The presence of engine to interpret the style.

Call and return architectures

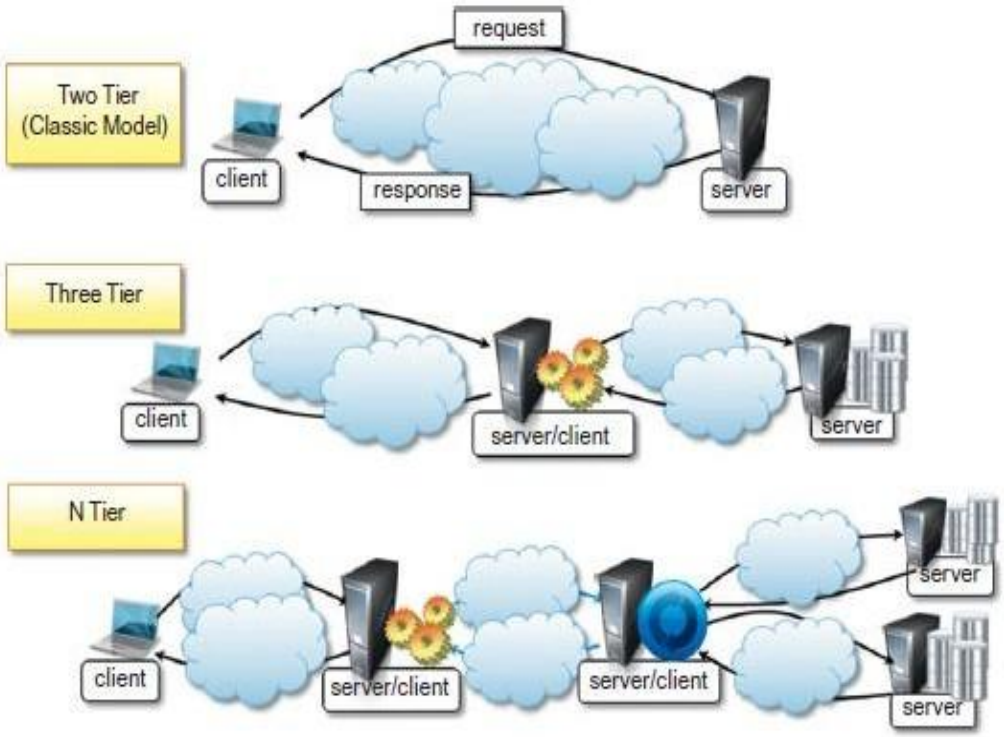
- This identifies all systems that are organized into components mostly connected together by method calls.
- The activity of systems modeled in this way is characterized by a chain of method calls whose overall execution and composition identify the execution one or more operations.
- There are three categories in this
  - Top down Style : developed with imperative programming
  - Object Oriented Style: Object programming models
  - Layered Style: provides the implementation in different levels of abstraction of the system.

System Architectural Styles

- System architectural styles cover the physical organization of components and processes over a distributed infrastructure.
- Two fundamental reference style
  - Client / Server
  - Peer- to -Peer
  - The information and the services of interest can be centralized and accessed through a single access point: the server.
  - Multiple clients are interested in such services and the server must be appropriately designed to efficiently serve requests coming from different clients.

Client / Server architectural Styles

-
-
-
-
-



- Symmetric architectures in which all the components, called peers, play the same role and incorporate both client and server capabilities of the client/server model.
- More precisely, each peer acts as a server when it processes requests from other peers and as a client when it issues requests to other peers.

Peer-to-Peer architectural Style

d.Models for Inter process Communication

- Distributed systems are composed of a collection of concurrent processes interacting with each other by means of a network connection.
- IPC is a fundamental aspect of distributed systems design and implementation.
- IPC is used to either exchange data and information or coordinate the activity of processes.
- IPC is what ties together the different components of a distributed system, thus making them act as a single system.
- There are several different models in which processes can interact with each other – these maps to different abstractions for IPC.
- Among the most relevant that we can mention are shared memory, remote procedure call (RPC), and message passing.
- At lower level, IPC is realized through the fundamental tools of network programming.
- Sockets are the most popular IPC primitive for implementing communication channels between distributed processes.

Message-based communication

- The abstraction of message has played an important role in the evolution of the model and technologies enabling distributed computing.
- The definition of distributed computing – is the one in which components located at networked computers communicate and coordinate their actions only by passing messages. The term messages, in this case, identify any discrete amount of information that is passed from one entity to another. It encompasses any form of data representation that is limited in size and time, whereas this is an invocation to a remote procedure or a serialized object instance or a generic message.
- The term message-based communication model can be used to refer to any model for IPC.
- Several distributed programming paradigms eventually use message-based communication despite the abstractions that are presented to developers for programming the interactions of distributed components.
- Here are some of the most popular and important:

Message Passing: This paradigm introduces the concept of a message as the main abstraction of the model. The entities exchanging information explicitly encode in the form of a message the data to be exchanged. The structure and the content of a message vary according to the model. Examples of this model are the Message-Passing-Interface (MPI) and openMP.

- **Remote Procedure Call (RPC):** This paradigm extends the concept of procedure call beyond the boundaries of a single process, thus triggering the execution of code in remote processes.
- **Distributed Objects:** This is an implementation of the RPC model for the object-oriented paradigm and contextualizes this feature for the remote invocation of methods exposed by objects. Examples of distributed object infrastructures are Common Object Request Broker Architecture (CORBA), Component Object Model (COM, DCOM, and COM+), Java Remote Method Invocation (RMI), and .NET Remoting.
- **Distributed agents and active Objects:** Programming paradigms based on agents and active objects involve by definition the presence of instances, whether they are agents of objects, despite the existence of requests.
- **Web Service**: An implementation of the RPC concept over HTTP; thus allowing the interaction of components that are developed with different technologies. A Web service is exposed as a remote object hosted on a Web Server, and method invocation are transformed in HTTP requests, using specific protocols such as Simple Object Access Protocol (SOAP) or Representational State Transfer (REST).

e. Technologies for distributed computing

- Remote Procedure Call (RPC)
  - RPC is the fundamental abstraction enabling the execution procedures on clients' request.
  - RPC allows extending the concept of a procedure call beyond the boundaries of a process and a single memory address space.
  - The called procedure and calling procedure may be on the same system or they may be on different systems..
- Distributed Object Frameworks

– Extend object-oriented programming systems by allowing objects to be distributed across a heterogeneous network and provide facilities so that they can be coherently act as though they were in the same address space.

Web Services

- Web Services are the prominent technology for implementing SOA systems and applications.
- They leverage Internet technologies and standards for building distributed systems.
- Several aspects make Web Services the technology of choice for SOA.
- First, they allow for interoperability across different platforms and programming languages.
- Second, they are based on well-known and vendor-independent standards such as HTTP, SOAP, and WSDL.
- Third, they provide an intuitive and simple way to connect heterogeneous software systems, enabling quick composition of services in distributed environment.

ELASTICITY IN CLOUD COMPUTING

- Elasticity is defined as the ability of a system to add and remove resources (such as CPU cores, memory, VM and container instances) to adapt to the load variation in real time.
- Elasticity is a dynamic property for cloud computing.
- Elasticity is the degree to which a system is able to adapt to workload changes by provisioning and deprovisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible.

Elasticity = Scalability + Automation + Optimization

- Elasticity is built on top of scalability.
- It can be considered as an automation of the concept of scalability and aims to optimize at best and as quickly as possible the resources at a given time.
- Another term associated with elasticity is the efficiency, which characterizes how cloud resource can be efficiently utilized as it scales up or down.
- It is the amount of resources consumed for processing a given amount of work, the lower this amount is, the higher the efficiency of a system.

- Elasticity also introduces a new important factor, which is the speed.
- Rapid provisioning and deprovisioning are key to maintaining an acceptable performance in the context of cloud computing
- Quality of service is subjected to a service level agreement

Classification

Elasticity solutions can be arranged in different classes based on

- Scope
- Policy
- Purpose
- Method

a.Scope

- Elasticity can be implemented on any of the cloud layers.
- Most commonly, elasticity is achieved on the IaaS level, where the resources to be provisioned are virtual machine instances.
- Other infrastructure services can also be scaled
- On the PaaS level, elasticity consists in scaling containers or databases for instance.
- Finally, both PaaS and IaaS elasticity can be used to implement elastic applications, be it for private use or in order to be provided as a SaaS
- The elasticity actions can be applied either at the infrastructure or application/platform level.
- The elasticity actions perform the decisions made by the elasticity strategy or management system to scale the resources.
- Google App Engine and Azure elastic pool are examples of elastic Platform as a Service (PaaS).
- Elasticity actions can be performed at the infrastructure level where the elasticity controller monitors the system and takes decisions.
- The cloud infrastructures are based on the virtualization technology, which can be VMs or containers.
- In the embedded elasticity, elastic applications are able to adjust their own resources according to runtime requirements or due to changes in the execution flow.
- There must be a knowledge of the source code of the applications.

- Application Map: The elasticity controller must have a complete map of the application components and instances.
- Code embedded: The elasticity controller is embedded in the application source code.
- The elasticity actions are performed by the application itself.
- While moving the elasticity controller to the application source code eliminates the use of monitoring systems
- There must be a specialized controller for each application.

b.Policy

- Elastic solutions can be either manual or automatic.
- A manual elastic solution would provide their users with tools to monitor their systems and add or remove resources but leaves the scaling decision to them.

Automatic mode: All the actions are done automatically, and this could be classified into reactive and proactive modes.

Elastic solutions can be either reactive or predictive

Reactive mode: The elasticity actions are triggered based on certain thresholds or rules, the system reacts to the load (workload or resource utilization) and triggers actions to adapt changes accordingly.

- An elastic solution is reactive when it scales a posteriori, based on a monitored change in the system.
- These are generally implemented by a set of Event-Condition-Action rules.

Proactive mode: This approach implements forecasting techniques, anticipates the future needs and triggers actions based on this anticipation.

- A predictive or proactive elasticity solution uses its knowledge of either recent history or load patterns inferred from longer periods of time in order to predict the upcoming load of the system and scale according to it.

c.Purpose

- An elastic solution can have many purposes.
- The first one to come to mind is naturally performance, in which case the focus should be put on their speed.
- Another purpose for elasticity can also be energy efficiency, where using the minimum amount of resources is the dominating factor.

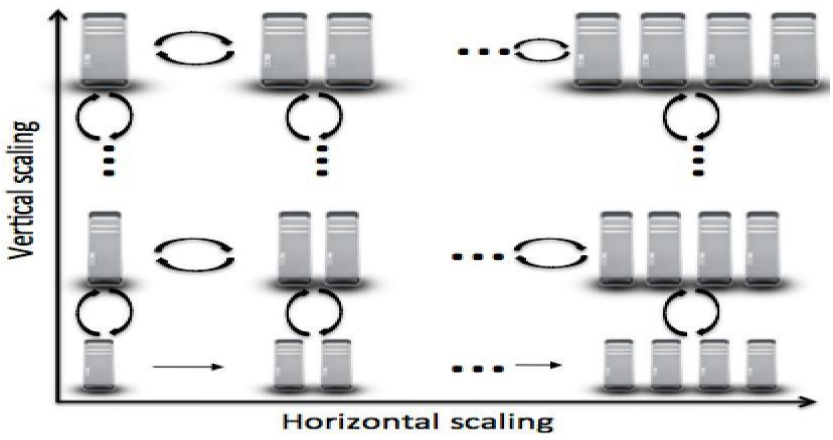
- Other solutions intend to reduce the cost by multiplexing either resource providers or elasticity methods
- Elasticity has different purposes such as improving performance, increasing resource capacity, saving energy, reducing cost and ensuring availability.
- Once we look to the elasticity objectives, there are different perspectives.
- Cloud IaaS providers try to maximize the profit by minimizing the resources while offering a good Quality of Service (QoS),
- PaaS providers seek to minimize the cost they pay to the

Cloud.

- The customers (end-users) search to increase their Quality of Experience (QoE) and to minimize their payments.
- QoE is the degree of delight or annoyance of the user of an application or service

d.Method

- Vertical elasticity, changes the amount of resources linked to existing instances on-the-fly.
- This can be done in two manners.
- The first method consists in explicitly redimensioning a virtual machine instance, i.e., changing the quota of physical resources allocated to it.
- This is however poorly supported by common operating systems as they fail to take into account changes in CPU or memory without rebooting, thus resulting in service interruption.
- The second vertical scaling method involves VM migration: moving a virtual machine instance to another physical machine with a different overall load changes its available resources



- Horizontal scaling is the process of adding/removing instances, which may be located at different locations.
- Load balancers are used to distribute the load among the different instances.
- Vertical scaling **is** the process of modifying resources (CPU, memory, storage or both) size for an instance at run time.
- It gives more flexibility for the cloud systems to cope with the varying workloads

Migration

- Migration can be also considered as a needed action to further allow the vertical scaling when there is no enough resources on the host machine.
- It is also used for other purposes such as migrating a VM to a less loaded physical machine just to guarantee its performance.
- Several types of migration are deployed such as live migration and no-live migration.
- Live migration has two main approaches
    - post-copy
    - pre-copy
- Post-copy migration suspends the migrating VM, copies minimal processor state to the target host, resumes the VM and then begins fetching memory pages from the source.
- In pre-copy approach, the memory pages are copied while the VM is running on the source.
- If some pages are changed (called dirty pages) during the memory copy process, they will be recopied until the number of recopied pages is greater than dirty pages, or the source VM will be stopped.
- The remaining dirty pages will be copied to the destination VM.

Architecture

- The architecture of the elasticity management solutions can be either centralized or decentralized.
- Centralized architecture has only one elasticity controller, i.e., the auto scaling system that provisions and deprovisions resources.

- In decentralized solutions, the architecture is composed of many elasticity controllers or application managers, which are responsible for provisioning resources for different cloud-hosted platforms

Provider

- Elastic solutions can be applied to a single or multiple cloud providers.
- A single cloud provider can be either public or private with one or multiple regions or datacenters.
- Multiple clouds in this context means more than one cloud provider.
- It includes hybrid clouds that can be private or public, in addition to the federated clouds and cloud bursting.
- Most of the elasticity solutions support only a single cloud provider

On-demand Provisioning.

- Resource Provisioning means the selection, deployment, and run-time management of software (e.g., database server management systems, load balancers) and hardware resources (e.g., CPU, storage, and network) for ensuring guaranteed performance for applications.
- Resource Provisioning is an important and challenging problem in the large-scale distributed systems such as Cloud computing environments.
- There are many resource provisioning techniques, both static and dynamic each one having its own advantages and also some challenges.
- These resource provisioning techniques used must meet Quality of Service (QoS) parameters like availability, throughput, response time, security, reliability etc., and thereby avoiding Service Level Agreement (SLA) violation.
- Over provisioning and under provisioning of resources must be avoided.
- Another important constraint is power consumption.
- The ultimate goal of the cloud user is to minimize cost by renting the resources and from the cloud service provider's perspective to maximize profit by efficiently allocating the resources.

- In order to achieve the goal, the cloud user has to request cloud service provider to make a provision for the resources either statically or dynamically.
- So that the cloud service provider will know how many instances of the resources and what resources are required for a particular application.
- By provisioning the resources, the QoS parameters like availability, throughput, security, response time, reliability, performance etc must be achieved without violating SLA

There are two types

- **Static Provisioning**
- **Dynamic Provisioning**

Static Provisioning

- For applications that have predictable and generally unchanging demands/workloads, it is possible to use “static provisioning" effectively.
- With advance provisioning, the customer contracts with the provider for services.
- The provider prepares the appropriate resources in advance of start of service.
- The customer is charged a flat fee or is billed on a monthly basis.

Dynamic Provisioning

- In cases where demand by applications may change or vary, “dynamic provisioning" techniques have been suggested whereby VMs may be migrated on-the-fly to new compute nodes within the cloud.
- The provider allocates more resources as they are needed and removes them when they are not.
- The customer is billed on a pay-per-use basis.
- When dynamic provisioning is used to create a hybrid cloud, it is sometimes referred to as cloud bursting.

Parameters for Resource Provisioning

- Response time
- Minimize Cost
- Revenue Maximization
- Fault tolerant
- Reduced SLA Violation
- Reduced Power Consumption

Response time: The resource provisioning algorithm designed must take minimal time to respond when executing the task.

Minimize Cost: From the Cloud user point of view cost should be minimized.

Revenue Maximization: This is to be achieved from the Cloud Service Provider's view.

Fault tolerant: The algorithm should continue to provide service in spite of failure of nodes.

Reduced SLA Violation: The algorithm designed must be able to reduce SLA violation.

Reduced Power Consumption: VM placement & migration techniques must lower power consumption

Dynamic Provisioning Types

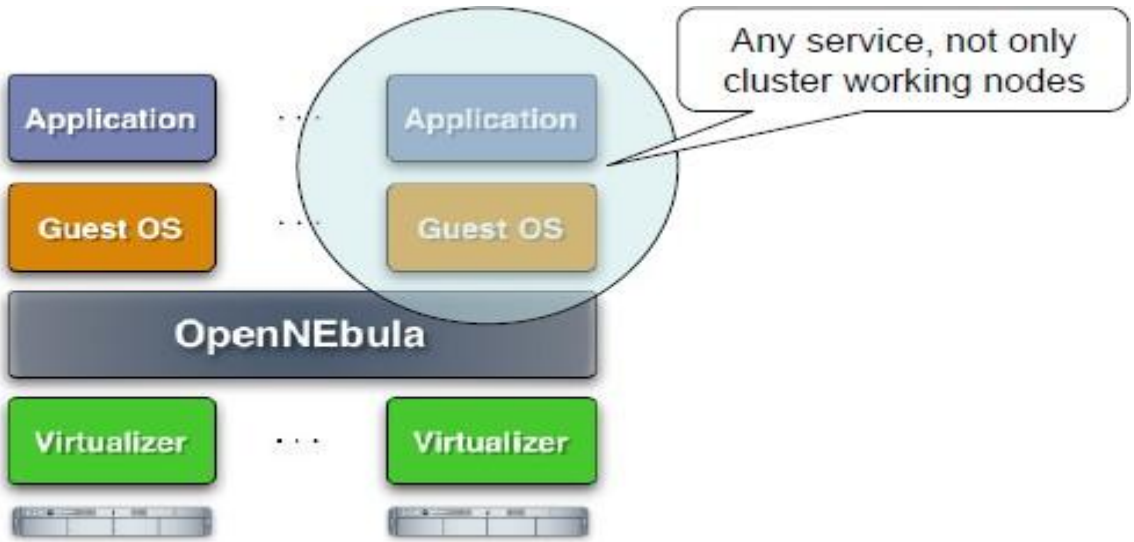
1. Local On-demand Resource Provisioning
2. Remote On-demand Resource Provisioning

Local On-demand Resource Provisioning

1. The Engine for the Virtual Infrastructure

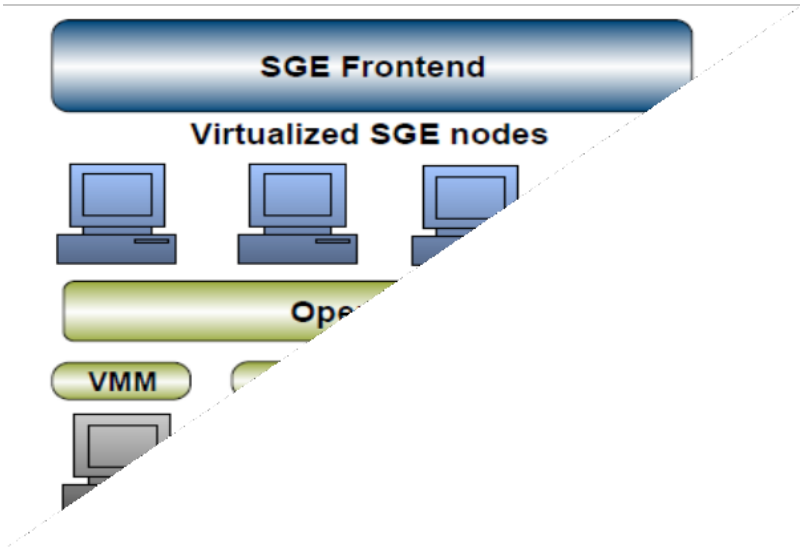
The OpenNebula Virtual Infrastructure Engine

- OpenNEbula creates a distributed virtualization layer
    - Extend the benefits of VM Monitors from one to multiple resources
    - Decouple the VM (service) from the physical location
- Transform a distributed physical infrastructure into a flexible and elastic virtual infrastructure, which adapts to the changing demands of the VM (service) workloads



Separation of Resource Provisioning from Job Management

- New virtualization layer between the service and the infrastructure layers
- Seamless integration with the existing middleware stacks.
- Completely transparent to the computing service and so end users



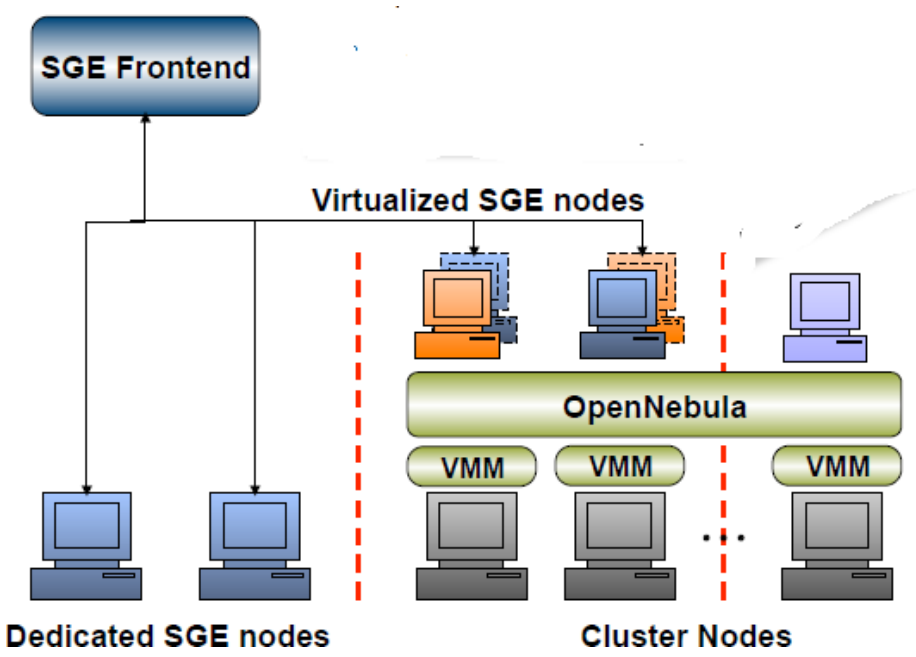
Cluster Partitioning

- Dynamic partition of the infrastructure
- Isolate workloads (several computing clusters)
- Dedicated HA partitions

Benefits for Existing Grid Infrastructures

- The **virtualization of the local infrastructure** supports a virtualized alternative to contribute resources to a Grid infrastructure
    - Simpler deployment and operation of new middleware distributions
    - Lower operational costs
    - Easy provision of resources to more than one infrastructure
    - Easy support for VO-specific worker nodes

  Performance partitioning between local and grid clusters



•

Other Tools for VM Management

- VMware DRS, Platform Orchestrator, IBM Director, Novell ZENworks, Enomalism, Xenoserver
- **Advantages**:
  - Open-source (Apache license v2.0)
  - Open and flexible architecture to integrate new virtualization technologies
  - Support for the definition of any scheduling policy (consolidation, workload balance, affinity, SLA)
  - LRM-like CLI and API for the integration of third-party tools

Remote on-Demand Resource Provisioning

Access to Cloud Systems

- Provision of virtualized resources as a service

VM Management Interfaces

The processes involved are

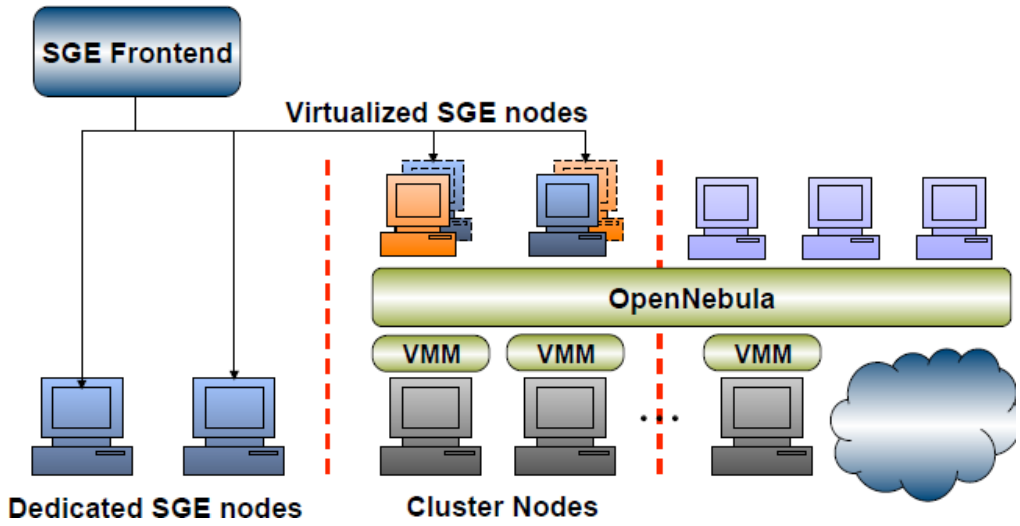
- Submission
- Control
- Monitoring

Infrastructure Cloud Computing Solutions

- Commercial Cloud: Amazon EC2
- Scientific Cloud: Nimbus (University of Chicago)
- Open-source Technologies
  - Globus VWS (Globus interfaces)
  - Eucalyptus (Interfaces compatible with Amazon EC2)
  - OpenNEbula (Engine for the Virtual Infrastructure)

On-demand Access to Cloud Resources

- Supplement local resources with cloud resources to satisfy peak or fluctuating demands



UNIT III CLOUD ARCHITECTURE, SERVICES AND STORAGE

Layered Cloud Architecture Design – NIST Cloud Computing Reference Architecture – Public, Private and Hybrid Clouds - laaS – PaaS – SaaS – Architectural Design Challenges – Cloud Storage – Storage-as-a-Service – Advantages of Cloud Storage – Cloud Storage Providers – S3.

LAYERED ARCHITECTURE:

Generic Cloud Architecture Design:

An Internet cloud is envisioned as a public cluster of servers provisioned on demand to perform collective web services or distributed applications using data-center resources.

- Cloud Platform Design Goals
- Enabling Technologies for Clouds
- A Generic Cloud Architecture

Cloud Platform Design Goals

- Scalability
- Virtualization
- Efficiency
- Reliability
- Security

Cloud management receives the user request and finds the correct resources. Cloud calls the provisioning services which invoke the resources in the cloud. Cloud management software needs to support both physical and virtual machines

Enabling Technologies for Clouds

- Cloud users are able to demand more capacity at peak demand, reduce costs, experiment with new services, and remove unneeded capacity.
- Service providers can increase system utilization via multiplexing, virtualization and dynamic resource provisioning.
- Clouds are enabled by the progress in hardware, software and networking technologies
- Cloud users are able to demand more capacity at peak demand, reduce costs, experiment with new services, and remove unneeded capacity.
- Service providers can increase system utilization via multiplexing, virtualization and dynamic resource provisioning.

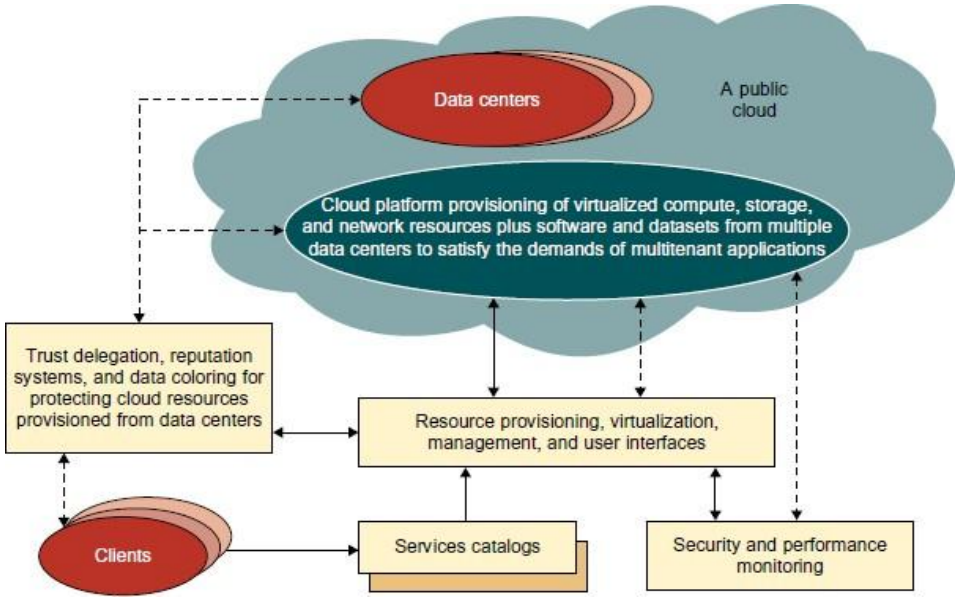
- Clouds are enabled by the progress in hardware, software and networking technologies

Table 4.3 Cloud-Enabling Technologies in Hardware, Software, and Networking

Technology	Requirements and Benefits
Fast platform deployment	Fast, efficient, and flexible deployment of cloud resources to provide dynamic computing environment to users
Virtual clusters on demand	Virtualized cluster of VMs provisioned to satisfy user demand and virtual cluster reconfigured as workload changes
Multitenant techniques	SaaS for distributing software to a large number of users for their simultaneous use and resource sharing if so desired
Massive data processing	Internet search and web services which often require massive data processing, especially to support personalized services
Web-scale communication	Support for e-commerce, distance education, telemedicine, social networking, digital government, and digital entertainment applications
Distributed storage	Large-scale storage of personal records and public archive information which demands distributed storage over the clouds
Licensing and billing services	License management and billing services which greatly benefit all types of cloud services in utility computing

A Generic Cloud Architecture

- The Internet cloud is envisioned as a massive cluster of servers.
- Servers are provisioned on demand to perform collective web services using data-center resources.
- The cloud platform is formed dynamically by provisioning or deprovisioning servers, software, and database resources.
- Servers in the cloud can be physical machines or VMs.
- User interfaces are applied to request services.

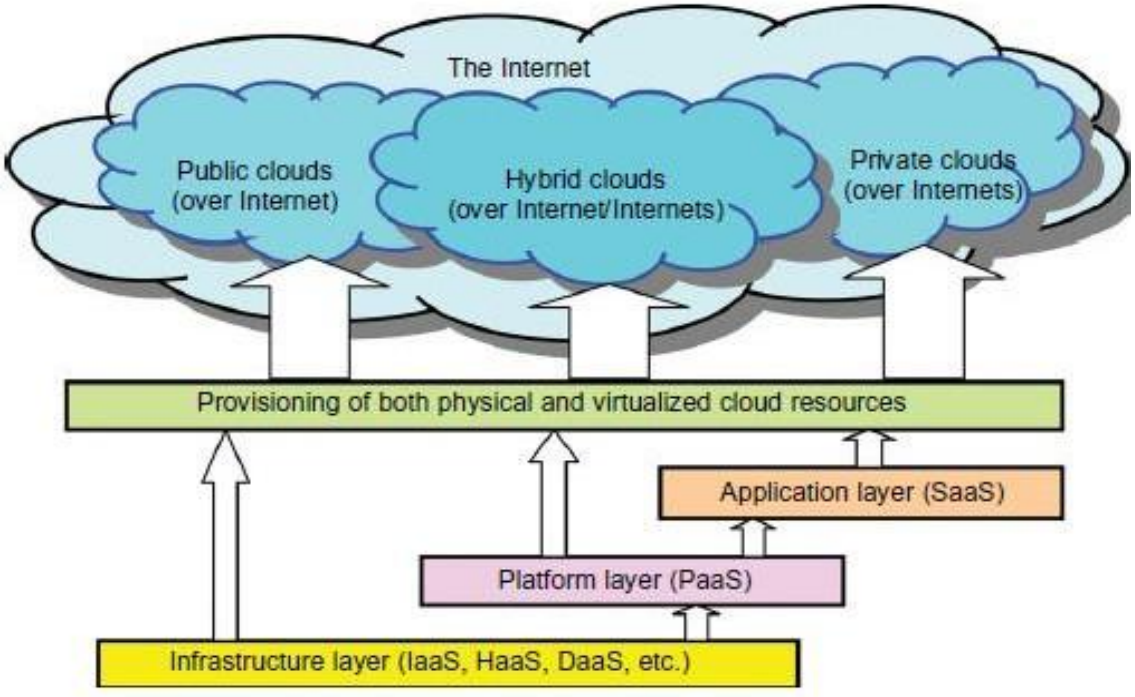


- The cloud computing resources are built into the data centers.
- Data centers are typically owned and operated by a third-party provider.

Consumers do not need to know the underlying technologies

- In a cloud, software becomes a service.
- Cloud demands a high degree of trust of massive amounts of data retrieved from large data centers.
- The software infrastructure of a cloud platform must handle all resource management and maintenance automatically.
- Software must detect the status of each node server joining and leaving.
- Cloud computing providers such as Google and Microsoft, have built a large number of data centers.
- Each data center may have thousands of servers.
- The location of the data center is chosen to reduce power and cooling costs.

Layered Cloud Architectural Development



- The architecture of a cloud is developed at three layers
  - Infrastructure
  - Platform
  - Application

- Implemented with virtualization and standardization of hardware and software resources provisioned in the cloud.

The services to public, private and hybrid clouds are conveyed to users through networking support

Infrastructure Layer

- Foundation for building the platform layer.
- Built with virtualized compute, storage, and network resources.
- Provide the flexibility demanded by users.
- Virtualization realizes automated provisioning of resources and optimizes the infrastructure management process.

Platform Layer

- Foundation for implementing the application layer for SaaS applications.
- Used for general-purpose and repeated usage of the collection of software resources.
- Provides users with an environment to develop their applications, to test operation flows, and to monitor execution results and performance.

The platform should be able to assure users that they have scalability, dependability, and security protection

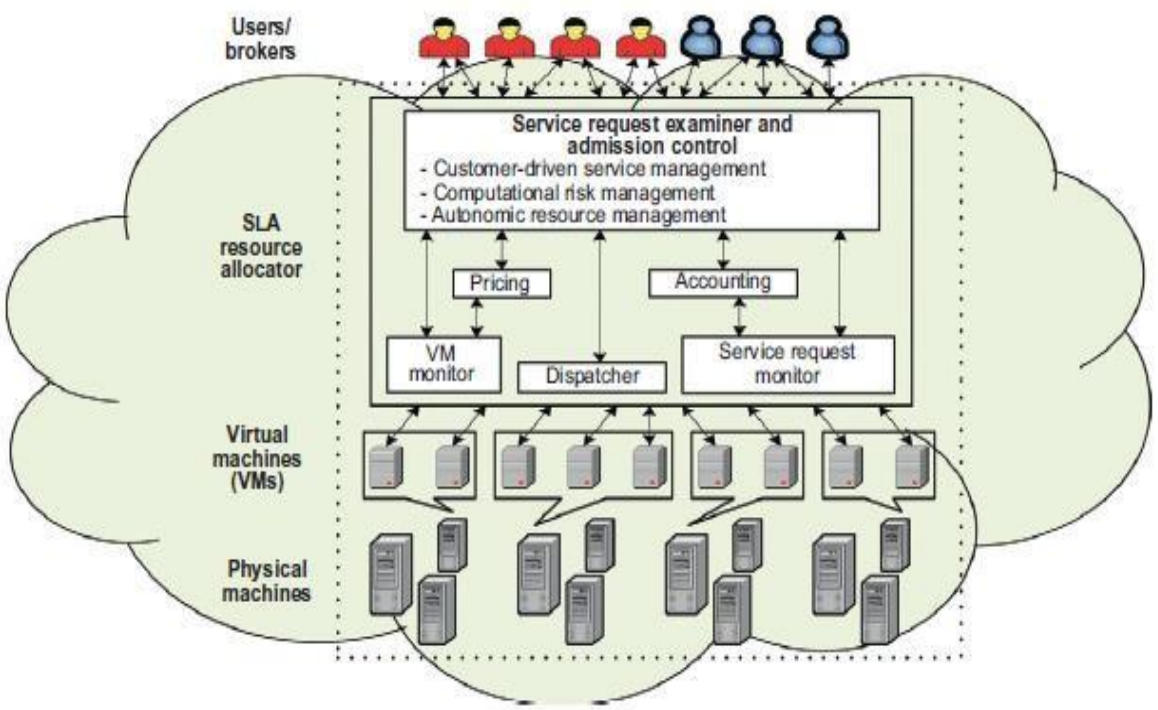
Application Layer

- Collection of all needed software modules for SaaS applications.
- Service applications in this layer include daily office management work, such as information retrieval, document processing, and authentication services.
- The application layer is also heavily used by enterprises in business marketing and sales, consumer relationship management (CRM) and financial transactions.
- Not all cloud services are restricted to a single layer.
- Many applications may apply resources at mixed layers.
- Three layers are built from the bottom up with a dependence relationship.

Market-Oriented Cloud Architecture

- High-level architecture for supporting market-oriented resource allocation in a cloud computing environment.
- Users or brokers acting on user's behalf submit service requests to the data center.
- When a service request is first submitted, the service request examiner interprets the submitted request for QoS requirements.

Accept or Reject the request.



- **VM Monitor**: Latest status information regarding resource availability.
- **Service Request Monitor**: Latest status information workload processing
- **Pricing mechanism:**Decides how service requests are charged.
- **Accounting mechanism:**Maintains the actual usage of resources by requests to compute the final cost.
- VM Monitor mechanism keeps track of the availability of VMs and their resource entitlements.
- Dispatcher starts the execution of accepted service requests on allocated VMs.

  Service Request Monitor mechanism keeps track of the execution progress of service requests.

  Multiple VMs can be started and stopped on demand

Quality of Service Factors

QoS parameters

- Time
- Cost
- Reliability
- Trust/security

  QoS requirements cannot be static and may change over time.

CLOUD REFERENCE ARCHITECTURE

Definitions

- A model of computation and data storage based on “pay as you go” access to “unlimited” remote data center capabilities.
- A cloud infrastructure provides a framework to manage scalable, reliable, on-demand access to applications.
- Cloud services provide the “invisible” backend to many of our mobile applications.

High level of elasticity in consumption.

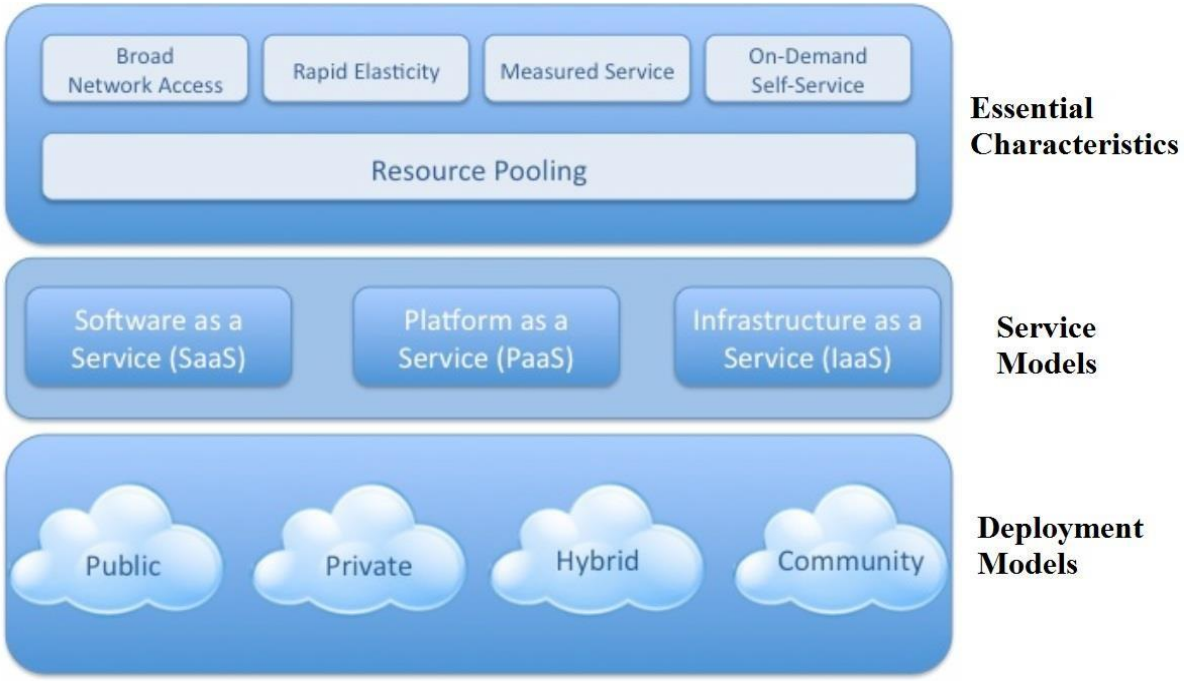
NIST Cloud Definition:

The National Institute of Standards and Technology (NIST) defines cloud computing as a

> **"pay-per-use model for enabling available, convenient and on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.**"

Architecture

- Architecture consists of 3 tiers
  - Cloud Deployment Model
  - Cloud Service Model
  - Essential Characteristics of Cloud Computing .



Essential Characteristics 1

- On-demand self-service.
  - A consumer can unilaterally provision computing capabilities such as server time and network storage as needed automatically, without requiring human interaction with a service provider.

Essential Characteristics 2

- Broad network access.
  - Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, laptops, and PDAs) as well as other traditional or cloudbased software services.

Essential Characteristics 3

- Resource pooling.
  - The provider's computing resources are pooled to serve multiple consumers using a **multi-tenant model**, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand.

Essential Characteristics 4

- **Rapid elasticity.**
  - Capabilities can be rapidly and elastically provisioned - in some cases automatically - to quickly scale out; and rapidly released to quickly scale in.
  - To the consumer, the capabilities available for provisioning often appear to be unlimited and can be purchased in any quantity at any time.

Essential Characteristics 5

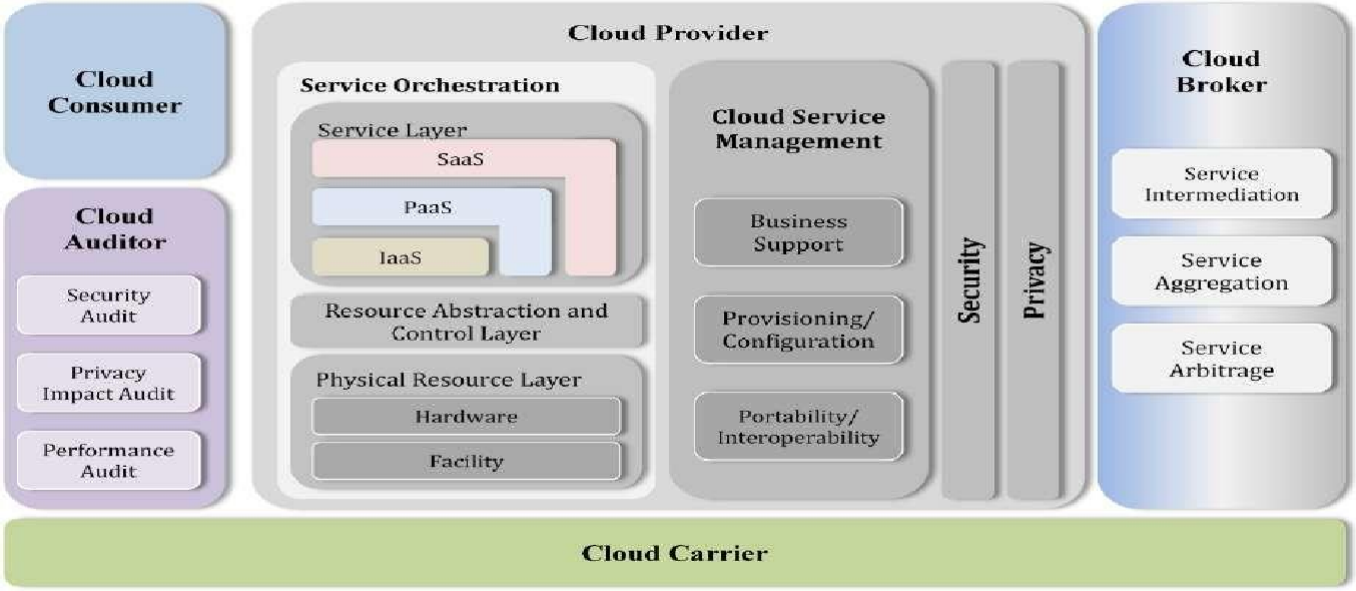
- **Measured service.**
  - Cloud systems automatically control and optimize resource usage by leveraging a metering capability at some level of abstraction appropriate to the type of service.

Resource usage can be monitored, controlled, and reported - providing transparency for both the provider and consumer of the service.

NIST (National Institute of Standards and Technology Background)

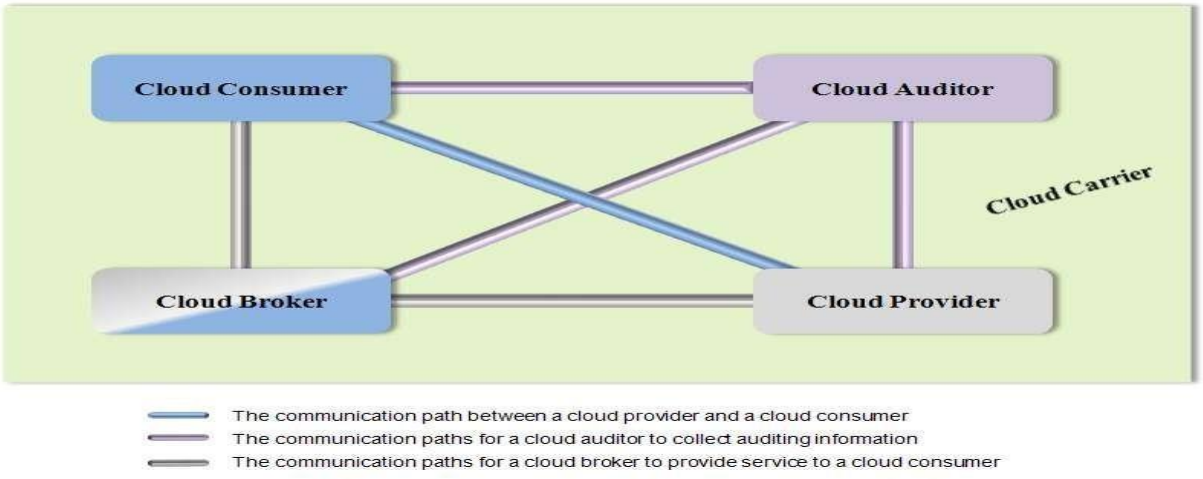
The goal is to accelerate the federal government's adoption of secure and effective cloud computing to reduce costs and improve services.

Cloud Computing Reference Architecture:



Actor	Definition
Cloud Consumer	A person or organization that maintains a business relationship with, and uses service from, *Cloud Providers*.
Cloud Provider	A person, organization, or entity responsible for making a service available to interested parties.
Cloud Auditor	A party that can conduct independent assessment of cloud services, information system operations, performance and security of the cloud implementation.
Cloud Broker	An entity that manages the use, performance and delivery of cloud services, and negotiates relationships between *Cloud Providers* and *Cloud Consumers*.
Cloud Carrier	An intermediary that provides connectivity and transport of cloud services from *Cloud Providers* to *Cloud Consumers*.

Interactions between the Actors in Cloud Computing

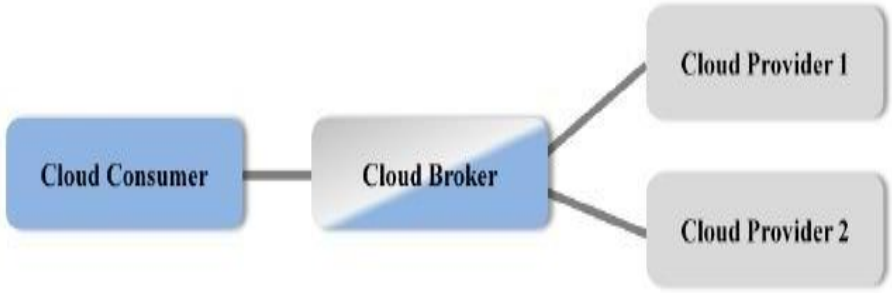


Example Usage Scenario 1:

- A cloud consumer may request service from a cloud broker instead of contacting a cloud provider directly.
- The cloud broker may create a new service by combining multiple services or by enhancing an existing service.

Usage Scenario- Cloud Brokers

- In this example, the actual cloud providers are invisible to the cloud consumer.
- The cloud consumer interacts directly with the cloud broker.

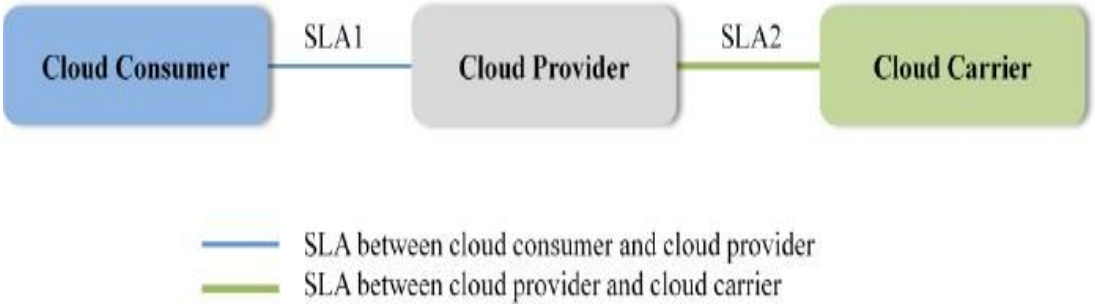


Example Usage Scenario 2

- Cloud carriers provide the connectivity and transport of cloud services from cloud providers to cloud consumers.
- A cloud provider participates in and arranges for two unique service level agreements (SLAs), one with a cloud carrier (e.g. SLA2) and one with a cloud consumer (e.g. SLA1).

Usage Scenario for Cloud Carriers

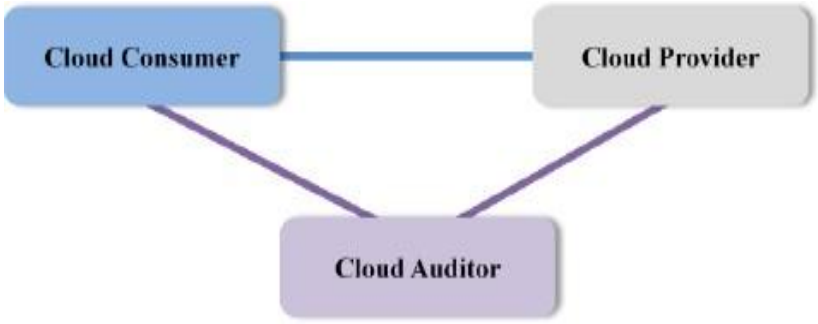
- A cloud provider arranges service level agreements (SLAs) with a cloud carrier.
- Request dedicated and encrypted connections to ensure the cloud services.



Example Usage Scenario 3

- For a cloud service, a cloud auditor conducts independent assessments of the operation and security of the cloud service implementation.

- The audit may involve interactions with both the Cloud Consumer and the Cloud Provider.

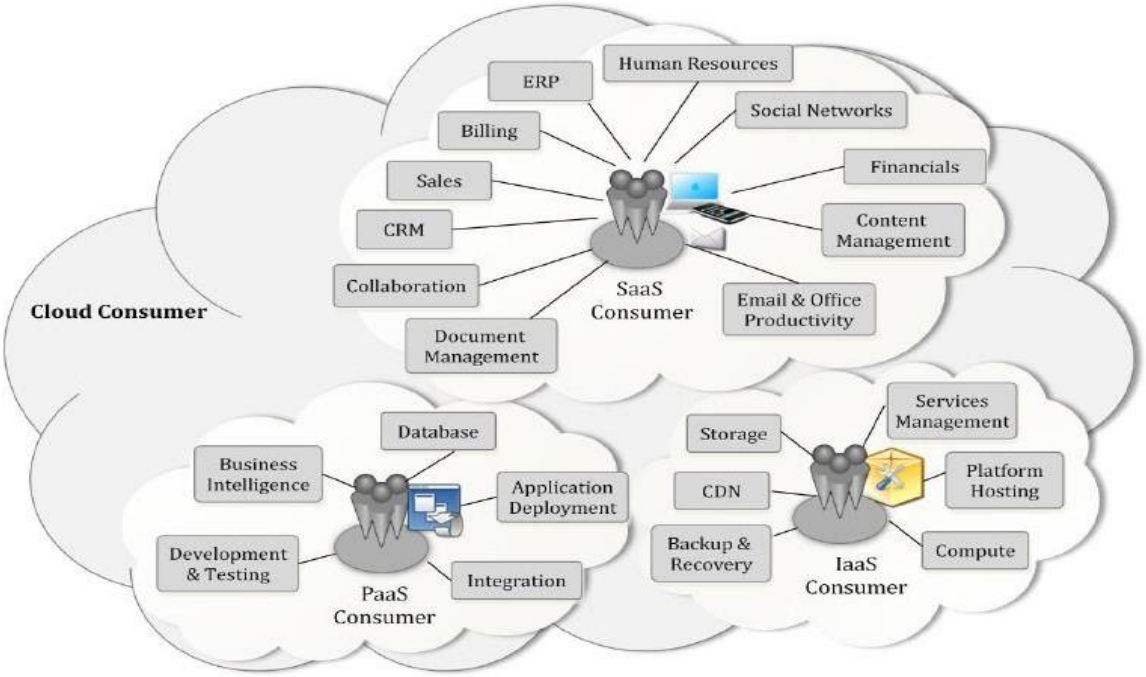


Cloud Consumer

- The cloud consumer is the principal stakeholder for the cloud computing service.
- A cloud consumer represents a person or organization that maintains a business relationship with, and uses the service from a cloud provider.

The cloud consumer may be billed for the service provisioned, and needs to arrange payments accordingly.

Example Services Available to a Cloud Consumer



- The consumers of SaaS can be organizations that provide their members with access to software applications, end users or software application administrators.
- SaaS consumers can be billed based on the number of end users, the time of use, the network bandwidth consumed, the amount of data stored or duration of stored data.

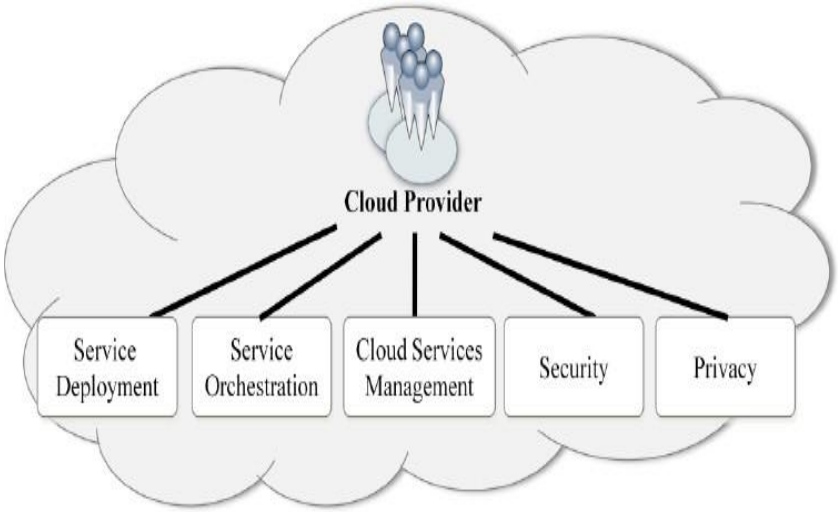
- Cloud consumers of PaaScan employ the tools and execution resources provided by cloud providers to develop, test, deploy and manage the applications.
- PaaS consumers can be application developers or application testers who run and test applications in cloud-based environments,.
- PaaS consumers can be billed according to, processing, database storage and network resources consumed.
- Consumers of IaaS have access to virtual computers, network-accessible storage & network infrastructure components.
- The consumers of IaaS can be system developers, system administrators and IT managers.
- IaaS consumers are billed according to the amount or duration of the resources consumed, such as CPU hours used by virtual computers, volume and duration of data stored.

Cloud Provider

- A cloud provider is a person, an organization;
- It is the entity responsible for making a service available to interested parties.
- A Cloud Provider acquires and manages the computing infrastructure required for providing the services.
- Runs the cloud software that provides the services.

Makes arrangement to deliver the cloud services to the Cloud Consumers through network access.

Cloud Provider - Major Activities



Cloud Auditor

- A cloud auditor is a party that can perform an independent examination of cloud service controls.
- Audits are performed to verify conformance to standards through review of objective evidence.
- A cloud auditor can evaluate the services provided by a cloud provider in terms of security controls, privacy impact, performance, etc.

Cloud Broker

- Integration of cloud services can be too complex for cloud consumers to manage.
- A cloud consumer may request cloud services from a cloud broker, instead of contacting a cloud provider directly.
- A cloud broker is an entity that manages the use, performance and delivery of cloud services. Negotiates relationships between cloud providers and cloud consumers.

Services of cloud broker

Service Intermediation:

- A cloud broker enhances a given service by improving some specific capability and providing value-added services to cloud consumers.

Service Aggregation:

- A cloud broker combines and integrates multiple services into one or more new services.
- The broker provides data integration and ensures the secure data movement between the cloud consumer and multiple cloud providers.

Services of cloud broker

Service Arbitrage:

- Service arbitrage is similar to service aggregation except that the services being aggregated are not fixed.
- Service arbitrage means a broker has the flexibility to choose services from multiple agencies.

Eg: The cloud broker can use a credit-scoring service to measure and select an agency with the best score.

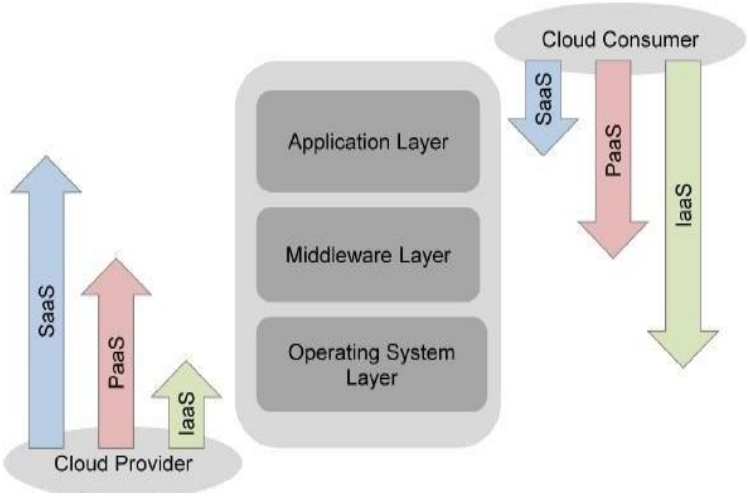
Cloud Carrier

- A cloud carrier acts as an intermediary that provides connectivity and transport of cloud services between cloud consumers and cloud providers.

- Cloud carriers provide access to consumers through network.
- The distribution of cloud services is normally provided by network and telecommunication carriers or a *transport agent*
- A transport agent refers to a business organization that provides physical transport of storage media such as high-capacity hard drives and other access devices.

Scope of Control between Provider and Consumer

The Cloud Provider and Cloud Consumer share the control of resources in a cloud system



- The application layer includes software applications targeted at end users or programs.

The applications are used by SaaS consumers, or installed/managed/maintained by PaaS consumers, IaaS consumers and SaaS providers.

- The middleware layer provides software building blocks (e.g., libraries, database, and Java virtual machine) for developing application software in the cloud.
- Used by PaaS consumers, installed/ managed/ maintained by IaaS consumers or PaaS providers, and hidden from SaaS consumers.
- The OS layer includes operating system and drivers, and is hidden from SaaS consumers and PaaS consumers.
- An IaaS cloud allows one or multiple guest OS to run virtualized on a single physical host.

The IaaS consumers should assume full responsibility for the guest OS, while the IaaS provider controls the host OS,

Cloud Deployment Model

- Public Cloud
- Private Cloud
- Hybrid Cloud
- Community Cloud

Public cloud

- A public cloud is one in which the cloud infrastructure and computing resources are made available to the general public over a public network.
- A public cloud is meant to serve a multitude(huge number) of users, not a single customer.
- A fundamental characteristic of public clouds is multitenancy.
- Multitenancy allows multiple users to work in a software environment at the same time, each with their own resources.
- Built over the Internet (i.e., service provider offers resources, applications storage to the customers over the internet) and can be accessed by any user.
- Owned by service providers and are accessible through a subscription.
- Best Option for small enterprises, which are able to start their businesses without large up-front(initial) investment.
- By renting the services, customers were able to dynamically upsize or downsize their IT according to the demands of their business.
- Services are offered on a price-per-use basis.
- Promotes standardization, preserve capital investment
- Public clouds have geographically dispersed datacenters to share the load of users and better serve them according to their locations
- Provider is in control of the infrastructure

Examples:

o Amazon EC2 is a public cloud that provides Infrastructure as a Service

o Google AppEngine is a public cloud that provides Platform as a Service

o SalesForce.com is a public cloud that provides software as a service.

Advantage

- **Offers unlimited scalability** – on demand resources are available to meet your business needs.
- **Lower costs**—no need to purchase hardware or software and you pay only for the service you use.
- **No maintenance -** Service provider provides the maintenance.
- **Offers reliability:** Vast number of resources are available so failure of a system will not interrupt service.
- Services like SaaS, PaaS, IaaS are easily available on Public Cloud platform as it can be accessed from anywhere through any Internet enabled devices.
- **Location independent** – the services can be accessed from any location

Disadvantage

- No control over privacy or security
- Cannot be used for use of sensitive applications(Government and Military agencies will not consider Public cloud)
- Lacks complete flexibility(since dependent on provider)
- No stringent (strict) protocols regarding data management

Private Cloud

- Cloud services are used by a single organization, which are not exposed to the public
- Services are always maintained on a private network and the hardware and software are dedicated only to single organization
- Private cloud is physically located at
    - Organization's premises [On-site private clouds] **(or)**
    - Outsourced(Given) to a third party[Outsource private Clouds]
- It may be managed either by
- Cloud Consumer organization (or)
    - By a third party
- Private clouds are used by

- government agencies
- financial institutions
- Mid size to large-size organisations.

- On-site private clouds

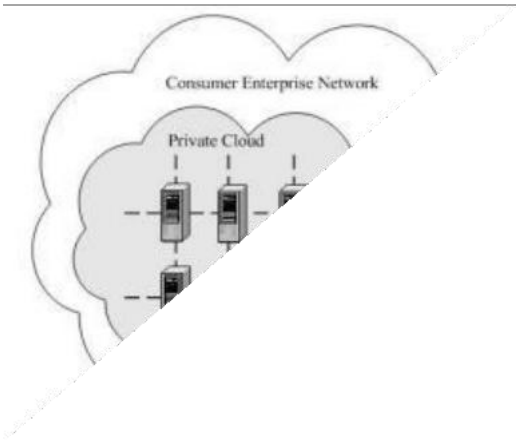
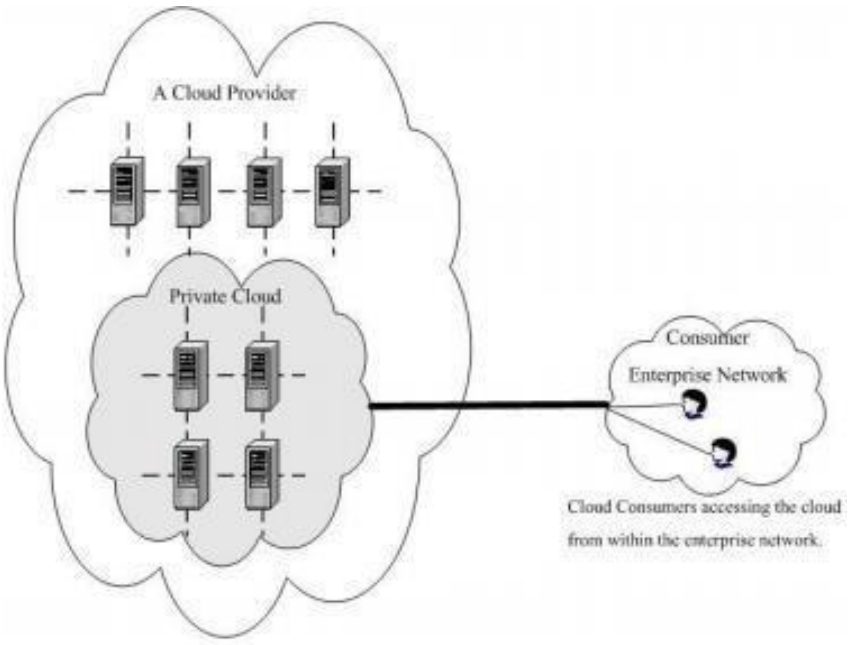


Fig: On-site private clouds

Out-sourced Private Cloud

- Supposed to deliver more efficient and convenient cloud



- Offers higher efficiency, resiliency(to recover quickly), security, and privacy
- **Customer information protection**: In-house security is easier to maintain and rely on.

- Follows its own(private organization) standard procedures and operations(where as in public cloud standard procedures and operations of service providers are followed)

Advantage

- Offers greater Security and Privacy
- Organization has control over resources
- Highly reliable
- Saves money by virtualizing the resources

Disadvantage

- Expensive when compared to public cloud
- Requires IT Expertise to maintain resources.

Hybrid Cloud

- Built with both public and private clouds
- It is a heterogeneous cloud resulting from a private and public clouds.
- Private cloud are used for
  - sensitive applications are kept inside the organization's network
  - business-critical operations like financial reporting
- Public Cloud are used when
  - Other services are kept outside the organization's network
  - high-volume of data
  - Lower-security needs such as web-based email(gmail,yahoomail etc)
- The resources or services are temporarily leased for the time required and then released. This practice is also known as **cloud bursting.**

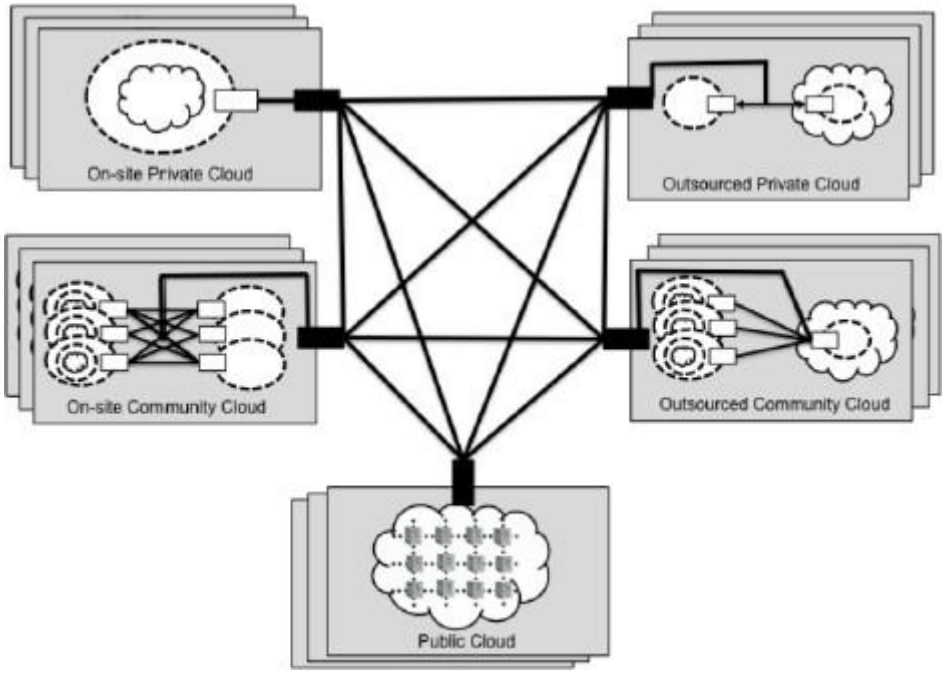


Fig:Hybrid Cloud

Advantage

- It is scalable
- Offers better security
- Flexible-Additional resources are availed in public cloud when needed
- Cost-effectiveness—we have to pay for extra resources only when needed.
- Control - Organisation can maintain a private infrastructure for sensitive application

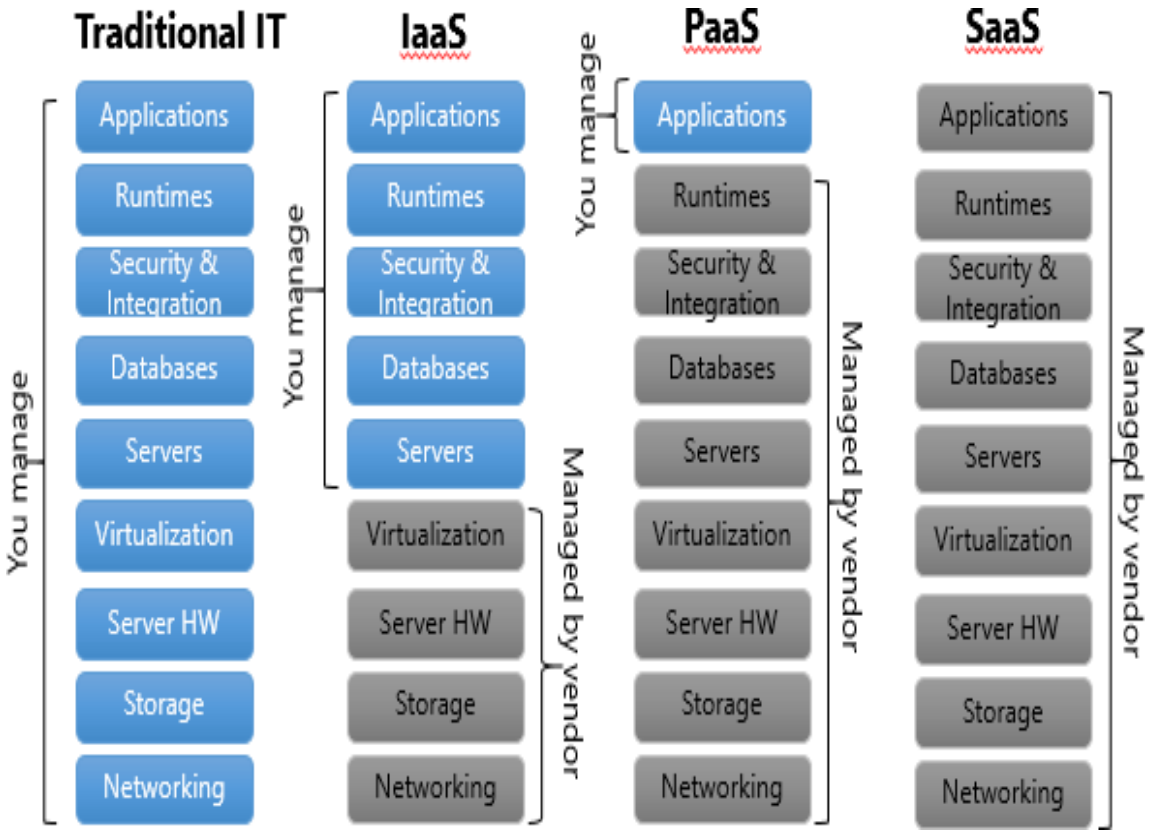
Disadvantage

- Infrastructure Dependency
- Possibility of security breach(violate) through public cloud

Difference	Public	Private	Hybrid
Tenancy	**Multi-tenancy:** the data of multiple organizations in stored in a shared environment.	**Single tenancy:** Single organizations data is stored in the cloud.	□ Data stored in the public cloud is multi-tenant. □ Data stored in private cloud is Single Tenancy.
Exposed to the Public	Yes: anyone can use the public cloud services.	No: Only the organization itself can use the private cloud services.	□ Services on private cloud can be accessed only by the organization's users □Services on public cloud can be Accessed by anyone.
Data Center Location	Anywhere on the Internet	Inside the organization's network.	□ Private Cloud- Present in organization's network. □ Public Cloud - anywhere on the Internet.
Cloud Service Management	Cloud service provider manages the services.	Organization has their own administrators managing services	□ Organization manages the private cloud. □ Cloud Service Provider(CSP) manages the public cloud.
Hardware Components	CSP provides all the hardware.	Organization provides hardware.	□ Private Cloud – organization provides resources. □ Public Cloud – Cloud service Provider provides.
Expenses	Less Cost	Expensive when compared to public cloud	Cost required for setting up private cloud.

Cloud Service Models

- **S**oftware as a Service (**S**aaS)
- **P**latform as a Service (**P**aaS)
- **I**nfrastructure as a Service (**I**aaS)



These models are offered based on various SLAs between providers and users

- SLA of cloud computing covers

o service availability

o performance

- data protection

o Security

Software as a Service(SaaS)(Complete software offering on the cloud)

- SaaS is a licensed software offering on the cloud and pay per use
- SaaS is a software delivery methodology that provides licensed multi-tenant access to software and its functions remotely as a Web-based service. Usually billed based on usage
  - Usually multi tenant environment

- Highly scalable architecture

- Customers do not invest on software application programs.
- The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure.
- The applications are accessible from various client devices through a thin client interface such as a web browser (e.g., web-based email).
- The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, data or even individual application capabilities, with the possible exception of limited user specific application configuration settings.
- On the customer side, there is no upfront investment in servers or software licensing.
- It is a "one-to-many" software delivery model, whereby an application is shared across multiple users
- Characteristic of Application Service Provider(ASP)
  - o Product sold to customer is application access.
  - o Application is centrally managed by Service Provider.
  - o Service delivered is one-to-many customers
  - o Services are delivered on the contract
    - E.g. Gmail and docs, Microsoft SharePoint, and the CRM software(Customer Relationship management)
- **SaaS providers**
- Google's Gmail, Docs, Talk etc
- Microsoft's Hotmail, Sharepoint
- SalesForce,
- Yahoo
- Facebook

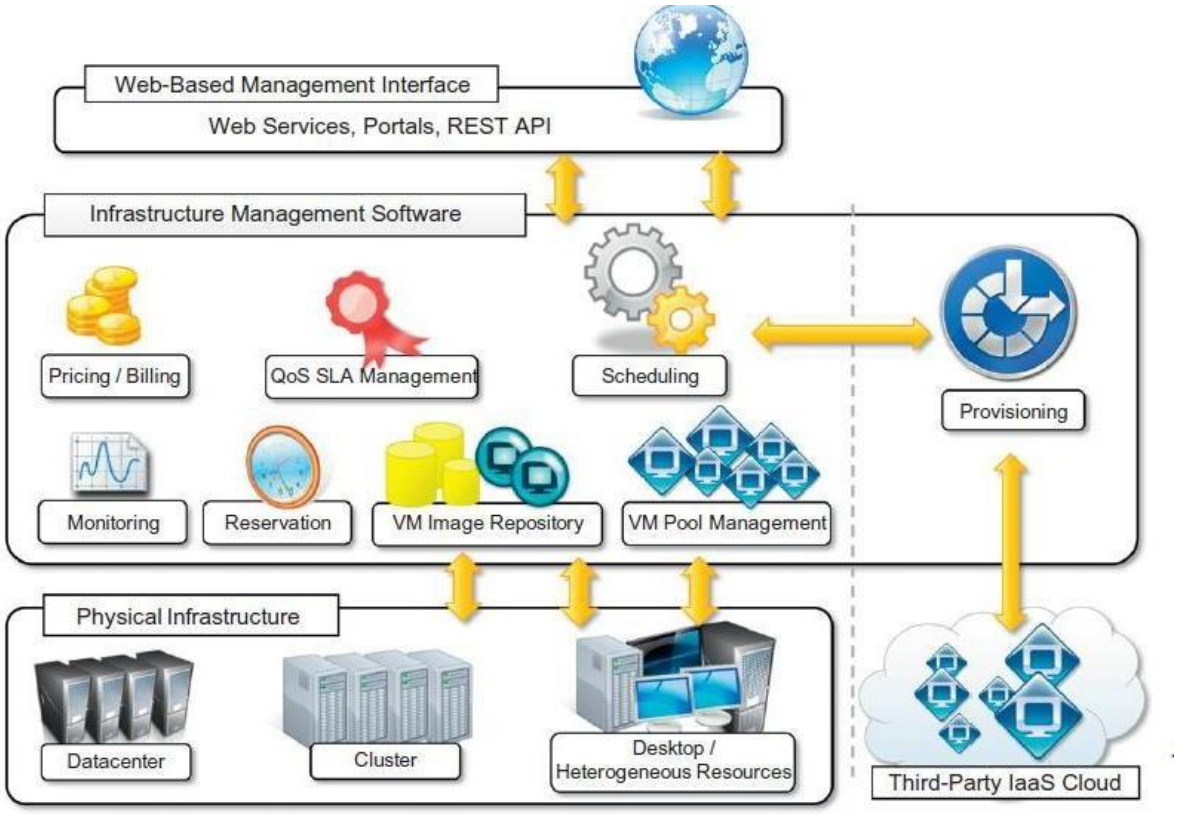
Infrastructure as a Service (IaaS) (Hardware offerings on the cloud)

IaaS is the delivery of technology infrastructure (mostly hardware) as an on demand, scalable service .

- Usually billed based on usage
- Usually multi tenant virtualized environment
- Can be coupled with Managed Services for OS and application support
- User can choose his OS, storage, deployed app, networking components

- The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources.
- Consumer is able to deploy and run arbitrary software, which may include operating systems and applications.
- The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage and deployed applications.

- IaaS/HaaS solutions bring all the benefits of hardware virtualization: workload partitioning, application isolation, sandboxing, and hardware tuning
- **Sandboxing:** A program is set aside from other programs in a separate environment so that if errors or security issues occur, those issues will not spread to other areas on the computer.
- **Hardware tuning:** To improve the performance of system
- The user works on multiple VMs running guest OSes
- the service is performed by rented cloud infrastructure
- The user does not manage or control the cloud infrastructure, but can specify when to request and release the needed resources.



IaaS providers

- Amazon Elastic Compute Cloud (EC2)
  - Each instance provides 1-20 processors, upto 16 GB RAM, 1.69TB storage
- RackSpace Hosting
  - Each instance provides 4 core CPU, upto 8 GB RAM, 480 GB storage
- Joyent Cloud
  - Each instance provides 8 CPUs, upto 32 GB RAM, 48 GB storage
- Go Grid
  - Each instance provides 1-6 processors, upto 15 GB RAM, 1.69TB storage

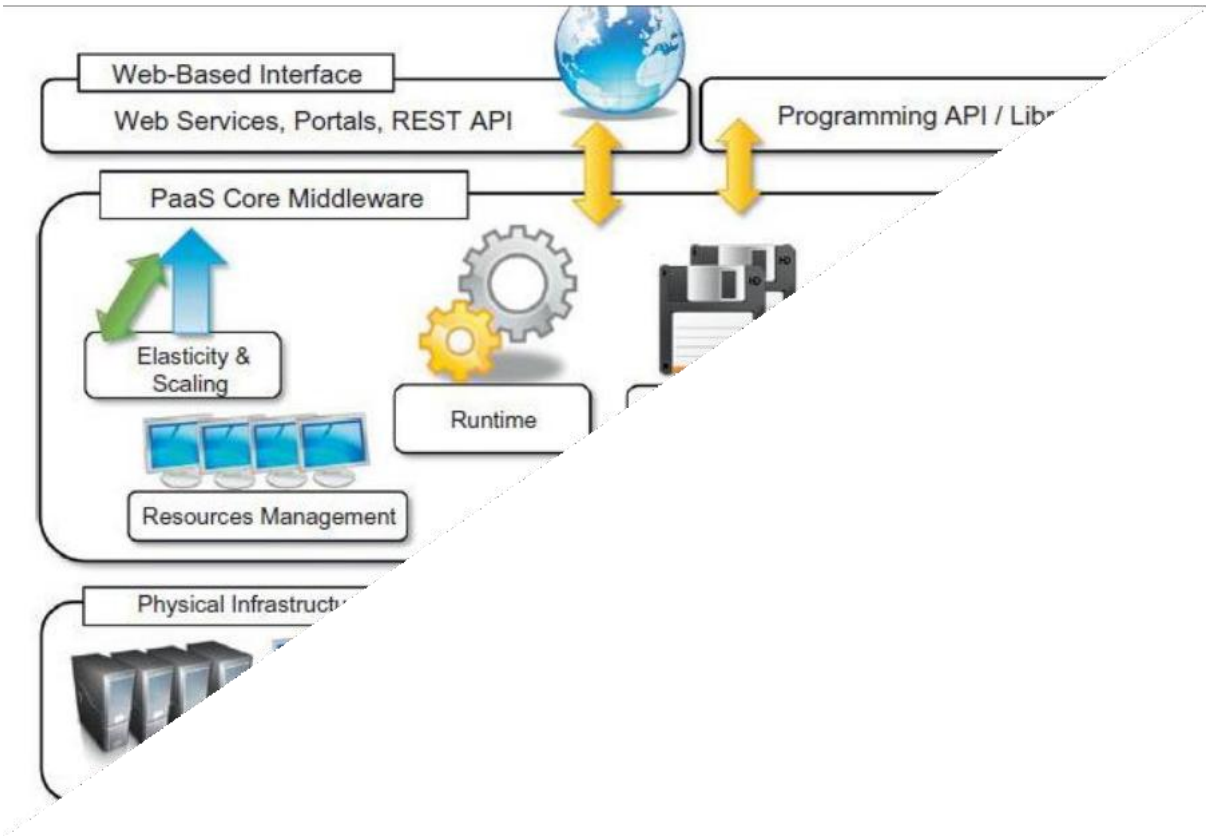
Platform as a Service (PaaS) (Development platform)

- PaaS provides all of the facilities required to support the complete life cycle of building, delivering and deploying web applications and services entirely from the Internet.
- Typically applications must be developed with a particular platform in mind
  - Multi tenant environments
  - Highly scalable multi tier architecture
- The capability provided to the consumer is to deploy onto the cloud infrastructure consumer created or acquired applications created using programming languages and tools supported by the provider.
- The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage.

Have control over the deployed applications and possibly application hosting environment configurations.

Customers are provided with execution platform for developing applications.

- Execution platform includes operating system, programming language execution environment, database, web server, hardware etc.
- This acts as **middleware** on top of which applications are built
- The user is freed from managing the cloud infrastructure



Application management is the core functionality of the middleware

 Provides runtime(execution) environment

 Developers design their applications in the execution environment.

 Developers need not concern about hardware (physical or virtual), operating systems, and other resources.

 PaaS core middleware manages the resources and scaling of applications on demand.

 PaaS offers

o Execution environment and hardware resources (infrastructure) **(or)**

o software is installed on the user premises

 PaaS: Service Provider provides Execution environment and hardware resources (infrastructure)

Characteristics of PaaS

 Runtime framework: Executes end-user code according to the policies set by the user and the provider.

 Abstraction: PaaS helps to deploy(install) and manage applications on the cloud.

 Automation: Automates the process of deploying applications to the infrastructure, additional resources are provided when needed.

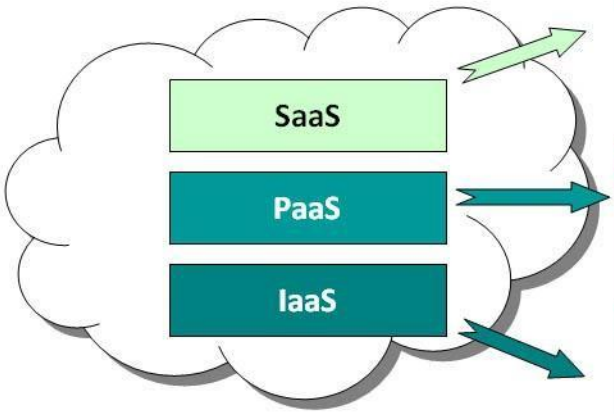
 Cloud services: helps the developers to simplify the creation and delivery cloud applications.

PaaS providers

- Google App Engine
  - Python, Java, Eclipse
- Microsoft Azure
  - .Net, Visual Studio
- Sales Force
  - Apex, Web wizard
- TIBCO,
- VMware,
- Zoho

Cloud Computing – Services

- Software as a Service - SaaS
- Platform as a Service - PaaS
- Infrastructure as a Service - IaaS



Who Uses It	What Services are available	Why use it?
Business Users	EMail, Office Automation, CRM, Website Testing, Wiki, Blog, Virtual Desktop ...	To complete business tasks
Developers and Deployers	Service and application test, development, integration and deployment	Create or deploy applications and services for users
System Managers	Virtual machines, operating systems, message queues, networks, storage, CPU, memory, backup services	Create platforms for service and application test, development, integration and deployment

Category	Description	Product Type	Vendors and Products
PaaS-I	Execution platform is provided along with hardware resources (infrastructure)	Middleware + Infrastructure	Force.com, Longjump
PaaS -II	Execution platform is provided with additional components	Middleware + Infrastructure, Middleware	Google App Engine
PaaS- III	Runtime environment for developing any kind of application development	Middleware + Infrastructure, Middleware	Microsoft Azure

Architectural Design Challenges

Challenge 1 : Service Availability and Data Lock-in Problem

Service Availability

- Service Availability in Cloud might be affected because of
- Single Point Failure
- Distributed Denial of Service
- Single Point Failure
  - o Depending on single service provider might result in failure.
  - o In case of single service providers, even if company has multiple data centres located in different geographic regions, it may have **common software infrastructure and accounting systems**.

Solution:

o Multiple cloud providers may provide more protection from failures and they provide High Availability(HA)

o Multiple cloud Providers will rescue the loss of all data.

Distributed Denial of service (DDoS) attacks.

o Cyber criminals, attack target websites and online services and makes services unavailable to users.

o DDoS tries to overwhelm (disturb) the services unavailable to user by having more traffic than the server or network can accommodate.

Solution:

o Some SaaS providers provide the opportunity to defend against DDoS attacks by using quick scale-ups.

Customers cannot easily extract their data and programs from one site to run on another.

Solution:

o Have standardization among service providers so that customers can deploy (install) services and data across multiple cloud providers.

Data Lock-in

 It is a situation in which a customer using service of a provider cannot be moved to another service provider because technologies used by a provider will be incompatible with other providers.

 This makes a customer dependent on a vendor for services and makes customer unable to use service of another vendor.

Solution:

o Have standardization (in technologies) among service providers so that customers can easily move from a service provider to another.

Challenge 2: Data Privacy and Security Concerns

 Cloud services are prone to attacks because they are accessed through internet.

Security is given by

o Storing the encrypted data in to cloud.

o Firewalls, filters.

 Cloud environment attacks include

o Guest hopping

o Hijacking

o VM rootkits.

 Guest Hopping: Virtual machine hyper jumping (VM jumping) is an attack method that exploits(make use of) hypervisor's weakness that allows a virtual machine (VM) to be accessed from another.

 Hijacking: Hijacking is a type of network security attack in which the attacker takes control of a communication

 VM Rootkit: is a collection of malicious (harmful) computer software, designed to enable access to a computer that is not otherwise allowed.

 A **man-in-the-middle (MITM)** attack is a form of eavesdroppping(Spy) where communication between two users is monitored and modified by an unauthorized party.

o Man-in-the-middle attack may take place **during VM migrations** [virtual machine (VM) migration - VM is moved from one physical host to another host].

 Passive attacks steal sensitive data or passwords.

 Active attacks may manipulate (control) kernel data structures which will cause major damage to cloud servers.

Challenge 3: Unpredictable Performance and Bottlenecks

 Multiple VMs can share CPUs and main memory in cloud computing, but I/O sharing is problematic.

 Internet applications continue to become more data-intensive (handles huge amount of data).

 Handling huge amount of data (data intensive) is a bottleneck in cloud environment.

 Weak Servers that does not provide data transfers properly must be removed from cloud environment

Challenge 4: Distributed Storage and Widespread Software Bugs

 The database is always growing in cloud applications.

 There is a need to create a storage system that meets this growth.

 This demands the design of efficient distributed SANs (Storage Area Network of Storage devices).

 Data centres must meet

o Scalability

o Data durability

o HA(High Availability)

o Data consistence

 Bug refers to errors in software.

 Debugging must be done in data centres.

Challenge 5: Cloud Scalability, Interoperability and Standardization

Cloud Scalability

 Cloud resources are scalable. Cost increases when storage and network bandwidth scaled(increased)

Interoperability

 Open Virtualization Format (OVF) describes an open, secure, portable, efficient, and extensible format for the packaging and distribution of VMs.

 OVF defines a transport mechanism for VM, that can be applied to different virtualization platforms

Standardization

 Cloud standardization, should have ability for virtual machine to run on any virtual platform.

Challenge 6: Software Licensing and Reputation Sharing

 Cloud providers can use both pay-for-use and bulk-use licensing schemes to widen the business coverage.

 Cloud providers must create reputation-guarding services similar to the “trusted e-mail” services

 Cloud providers want legal liability to remain with the customer, and vice versa.

Cloud Storage

 Storing your data on the storage of a cloud service provider rather than on a local system.

 Data stored on the cloud are accessed through Internet.

 Cloud Service Provider provides Storage as a Service

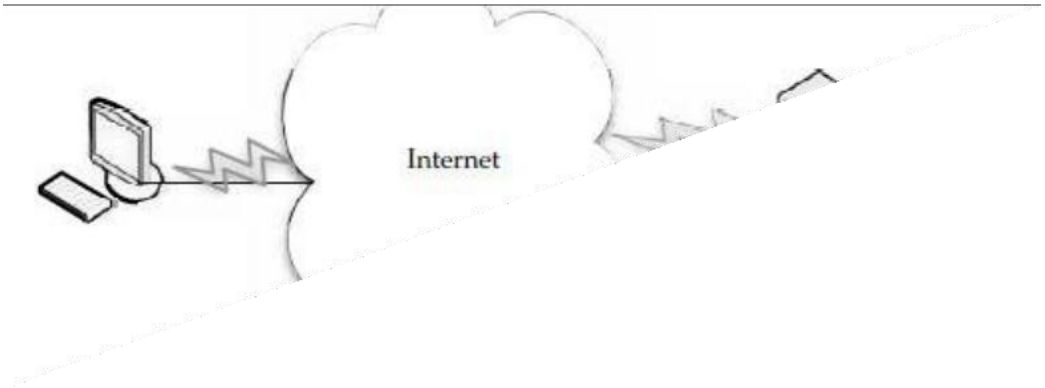
Storage as a Service

- Third-party provider rents space on their storage to cloud users.
- Customers move to cloud storage when they lack in budget for having their own storage.
- Storage service providers takes the responsibility of taking current backup, replication, and disaster recovery needs.
- Small and medium-sized businesses can make use of Cloud Storage
- Storage is rented from the provider using a

o cost-per-gigabyte-stored **(or)**

o cost-per-data-transferred

- The end user doesn't have to pay for infrastructure (resources), they have to pay only for how much they transfer and save on the provider's storage.



5.2 Providers

- Google Docs allows users to upload documents, spreadsheets, and presentations to Google's data servers.
- Those files can then be edited using a Google application.
- Web email providers like Gmail, Hotmail, and Yahoo! Mail, store email messages on their own servers.
- Users can access their email from computers and other devices connected to the Internet.
- Flicker and Picasa host millions of digital photographs, Users can create their own online photo albums.
- YouTube hosts millions of user-uploaded video files.
- Hostmonster and GoDaddy store files and data for many client web sites.
- Facebook and MySpace are social networking sites and allow members to post pictures and other content. That content is stored on the company's servers.
- MediaMax and Strongspace offer storage space for any kind of digital data.

Data Security

- To secure data, most systems use a combination of techniques:

o Encryption

o Authentication

o Authorization

Encryption

o Algorithms are used to encode information. To decode the information keys are required.

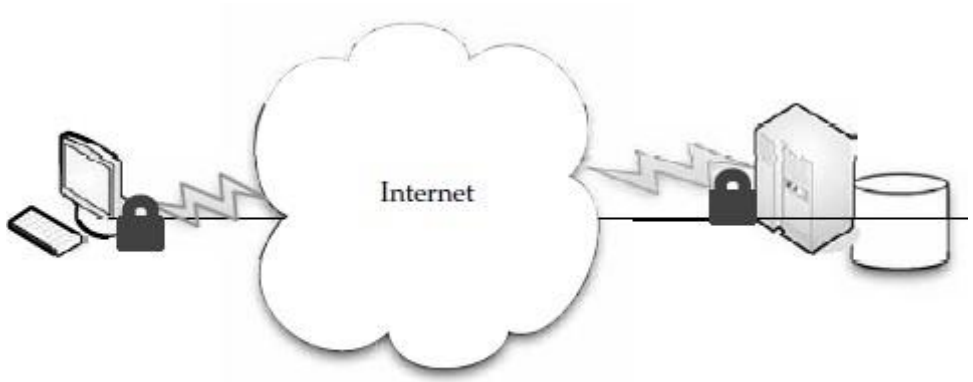
Authentication processes

o This requires a user to create a name and password.

Authorization practices

o The client lists the people who are authorized to access information stored on the cloud system.

If information stored on the cloud, the head of the IT department might have complete and free access to everything.



Encryption and authentication are two security measures you can use to keep your data safe on a cloud storage provider.

Reliability

- Service Providers gives reliability for data through redundancy (maintaining multiple copies of data).

Reputation is important to cloud storage providers. If there is a perception that the provider is unreliable, they won't have many clients.

Advantages

- Cloud storage providers balance server loads.
- Move data among various datacenters, ensuring that information is stored close and thereby available quickly to where it is used.
- It allows to protect the data in case there's a disaster.
- Some products are agent-based and the application automatically transfers information to the cloud via FTP

Cautions

- Don't commit everything to the cloud, but use it for a few, noncritical purposes.
- Large enterprises might have difficulty with vendors like Google or Amazon.
- Forced to rewrite solutions for their applications.

- Lack of portability.

Theft (Disadvantage)

- User data could be stolen or viewed by those who are not authorized to see it.
- Whenever user data is let out of their own datacenter, risk trouble occurs from a security point of view.
- If user store data on the cloud, make sure user encrypts data and secures data transit with technologies like SSL.

Cloud Storage Providers

Amazon Simple Storage Service (S3)

- The best-known cloud storage service is Amazon's Simple Storage Service (S3), launched in 2006.
- Amazon S3 is designed to make computing easier for developers.
- Amazon S3 provides an interface that can be used to store and retrieve any amount of data, at any time, from anywhere on the Web.
- Amazon S3 is intentionally built with a minimal feature set that includes the following functionality:

  • Write, read, and delete objects containing from 1 byte to 5 gigabytes of data each.

  The number of objects that can be stored is unlimited.

  • Each object is stored and retrieved via a unique developer-assigned key.

  • Objects can be made private or public, and rights can be assigned to specific users.

  • Uses standards-based REST and SOAP interfaces designed to work with any Internet-development toolkit.

Design Requirements

Amazon built S3 to fulfill the following design requirements:

• **Scalable** Amazon S3 can scale in terms of storage, request rate, and users to support an unlimited number of web-scale applications.

- **Reliable** Store data durably, with 99.99 percent availability. Amazon says it does not allow any downtime.

• **Fast** Amazon S3 was designed to be fast enough to support high-performance applications. Server-side latency must be insignificant relative to Internet latency. Any performance bottlenecks can be fixed by simply adding nodes to the system.

• **Inexpensive** Amazon S3 is built from inexpensive commodity hardware components. As a result, frequent node failure is the norm and must not affect the overall system. It must be hardware-agnostic, so that savings can be captured as Amazon continues to drive down infrastructure costs.

• **Simple** Building highly scalable, reliable, fast, and inexpensive storage is difficult. Doing so in a way that makes it easy to use for any application anywhere is more difficult. Amazon S3 must do both.

Design Principles

Amazon used the following principles of distributed system design to meet Amazon S3 requirements:

• **Decentralization** It uses fully decentralized techniques to remove scaling bottlenecks and single points of failure.

• **Autonomy** The system is designed such that individual components can make decisions based on local information.

• **Local responsibility** Each individual component is responsible for achieving its consistency; this is never the burden of its peers.

• **Controlled concurrency** Operations are designed such that no or limited concurrency control is required.

• **Failure toleration** The system considers the failure of components to be a normal mode of operation and continues operation with no or minimal interruption.

• **Controlled parallelism** Abstractions used in the system are of such granularity that parallelism can be used to improve performance and robustness of recovery or the introduction of new nodes.

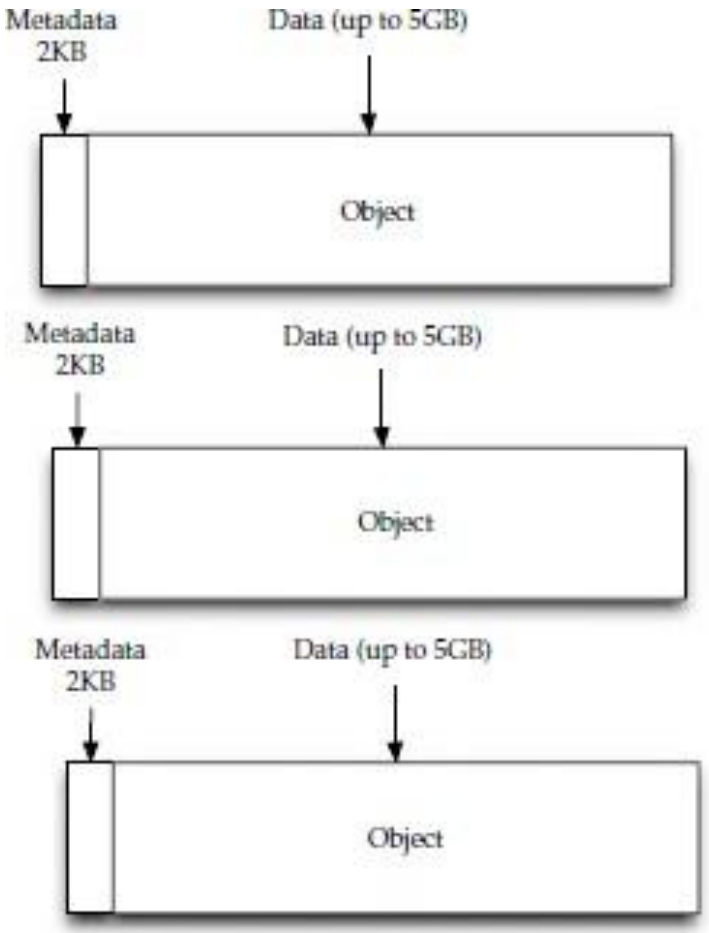
• **Small, well-understood building blocks** Do not try to provide a single service that does everything for everyone, but instead build small components that can be used as building blocks for other services.

• **Symmetry** Nodes in the system are identical in terms of functionality, and require no or minimal node-specific configuration to function.

• **Simplicity** The system should be made as simple as possible, but no simpler.

How S3 Works

Amazon keeps its lips pretty tight about how S3 works, but according to Amazon, S3's design aims to provide scalability, high availability, and low latency at commodity costs. S3 stores arbitrary objects at up to 5GB in size, and each is accompanied by up to 2KB of metadata. Objects are organized by *buckets*. Each bucket is owned by an AWS account and the buckets are identified by a unique, user-assigned key.



Multiple objects are stored in buckets in Amazon S3.

Buckets and objects are created, listed, and retrieved using either a REST-style or SOAP interface.

Objects can also be retrieved using the HTTP GET interface or via BitTorrent. An access control list restricts who can access the data in each bucket. Bucket names and keys are formulated so that they can be accessed using HTTP. Requests are authorized using an access control list associated with each bucket and object, for instance:

http://s3.amazonaws.com/examplebucket/examplekey

http://examplebucket.s3.amazonaws.com/examplekey

The Amazon AWS Authentication tools allow the bucket owner to create an authenticated URL with a set amount of time that the URL will be valid.

UNIT IV RESOURCE MANAGEMENT AND SECURITY IN CLOUD 10

Inter Cloud Resource Management – Resource Provisioning and Resource Provisioning Methods – Global Exchange of Cloud Resources – Security Overview – Cloud Security Challenges – Software-as-a-Service Security – Security Governance – Virtual Machine Security – IAM – Security Standards.

INTER-CLOUD RESOURCE MANAGEMENT

Cloud of Clouds (Inter cloud)

- Inter cloud or 'cloud of clouds'-refer to a theoretical model for cloud computing services.
- Combining many different individual clouds into one seamless mass in terms of on-demand operations.
- The inter cloud would simply make sure that a cloud could use resources beyond its reach.
- Taking advantage of pre-existing contracts with other cloud providers.
- Each single cloud does not have infinite physical resources or ubiquitous geographic footprint.
- A cloud may be saturated to the computational and storage resources of its infrastructure.
- It would still be able satisfy such requests for service allocations sent from its clients.
- A single cloud cannot always fulfill the requests or provide required services.
- When two or more clouds have to communicate with each other, or another intermediary comes into play and federates the resources of two or more clouds.
- In inter cloud, intermediary is known as "cloud broker" or simply "broker."
- Broker is the entity which introduces the cloud service customer (CSC) to the cloud service provider (CSP)

Inter-Cloud Resource Management Consists of

- Extended Cloud Computing Services
- Resource Provisioning and Platform Management
- Virtual Machine Creation and Management
- Global Exchange of Cloud Resources

4.1.1 Extended Cloud Computing Services

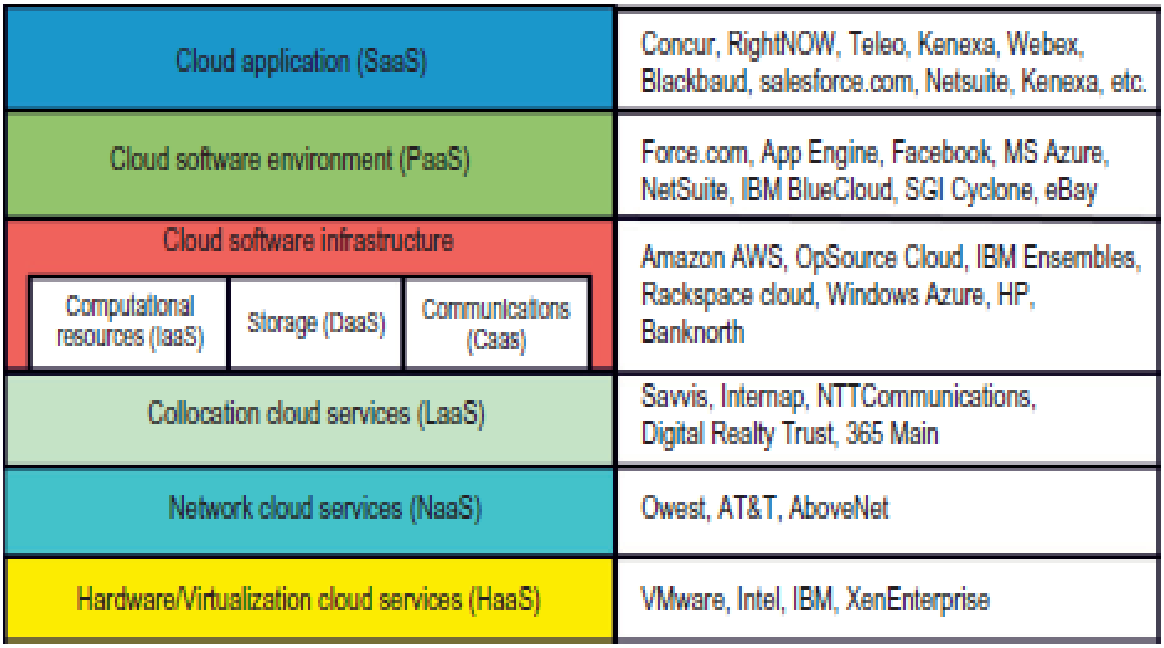


Fig: Six layers of cloud services and their providers

Six layers of cloud services

- Software as a Service(SaaS)
- Platform as a Service(PaaS)
- Infrastructure as a Service(IaaS)
- Hardware / Virtualization Cloud Services(HaaS)
- Network Cloud Services (NaaS)
- Collocation Cloud Services(LaaS)

- The top layer offers SaaS which provides cloud application.
- PaaS sits on top of IaaS infrastructure.
- The bottom three layers are more related to physical requirements.
- The bottommost layer provides Hardware as a Service (HaaS).
- NaaS is used for interconnecting all the hardware components.

- Location as a Service (LaaS), provides security to all the physical hardware and network resources. This service is also called as Security as a Service.
- The cloud infrastructure layer can be further subdivided as
  - Data as a Service (DaaS)
  - Communication as a Service (CaaS)
  - Infrastructure as a Service(IaaS)
- Cloud players are divided into three classes:
  - Cloud service providers and IT administrators
  - Software developers or vendors
  - End users or business users.

Cloud Players	IaaS	PaaS	SaaS
IT administrators/ Cloud Providers	Monitor SLAs	Monitor SLAs and enable service platforms	Monitor SLAs and deploy software
Software developers (Vendors)	To deploy and store data	Enabling platforms	Develop and deploy software
End users or business users	To deploy and store data	To develop and test software	Use business software

Table: Cloud Differences in Perspective of Providers, Vendors, and Users

Cloud Service Tasks and Trends

- SaaS is mostly used for Business Applications
- Eg: CRM (Customer Relationship Management) used for business promotion, direct sales, and marketing services
- PaaS is provided by Google, Salesforce.com, and Facebook etc.
- IaaS is provided by Amazon, Windows Azure, and RackRack etc.
- Collocation services Provides security to lower layers.
- Network cloud services provide communications.

Software Stack for Cloud Computing

- The software stack structure of cloud computing software can be viewed as layers.
- Each layer has its own purpose and provides the interface for the upper layers.
- The lower layers are not completely transparent to the upper layers.

Runtime Support Services

- Runtime support refers to software needed in applications.
- The SaaS provides the software applications as a service, rather than allowing users purchase the software.
- On the customer side, there is no upfront investment in servers.

Resource Provisioning (Providing) and Platform Deployment There are techniques to provision computer resources or VMs. Parallelism is exploited at the cluster node level.

Provisioning of Compute Resources (VMs)

- Providers supply cloud services by signing SLAs with end users.
- The SLAs must specify resources such as
  - CPU
  - Memory
  - Bandwidth

Users can use these for a preset (fixed) period.

- Under provisioning of resources will lead to broken SLAs and penalties.
- Over provisioning of resources will lead to resource underutilization, and consequently, a decrease in revenue for the provider.
- Provisioning of resources to users is a challenging problem. The difficulty comes from the following
  - Unpredictability of consumer demand
  - Software and hardware failures
  - Heterogeneity of services

- Power management
- Conflict in signed SLAs between consumers and service providers.

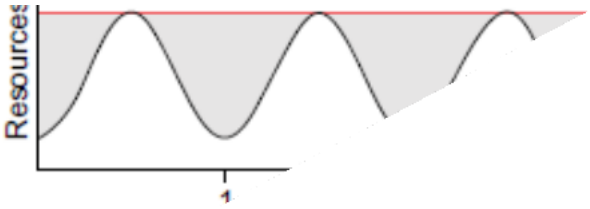
Provisioning Methods

Three cases of static cloud resource provisioning policies are considered.

Static cloud resource provisioning

case (a)

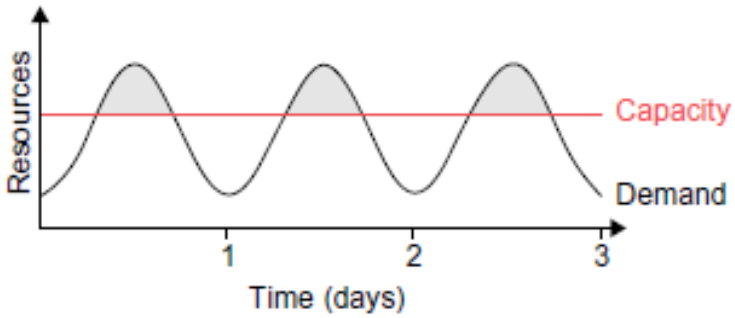
- over provisioning(Providing) with the peak load causes heavy resource waste (shaded area).



..

case (b)

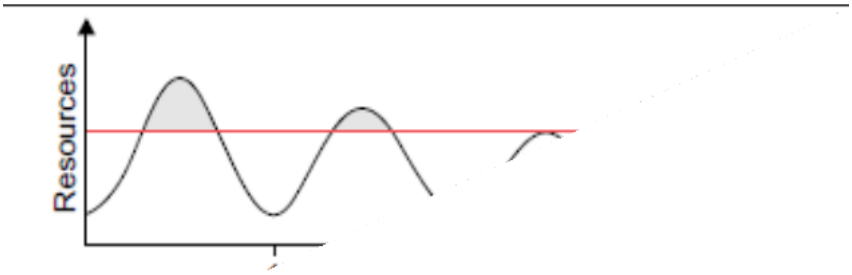
Under provisioning of resources results in losses by both user and provider. Users have paid for the demand (the shaded area above the capacity) is not used by users.



(b) Underprovisioning 1

case (c)

Declining in user demand results in worse resource waste.



Constant provisioning

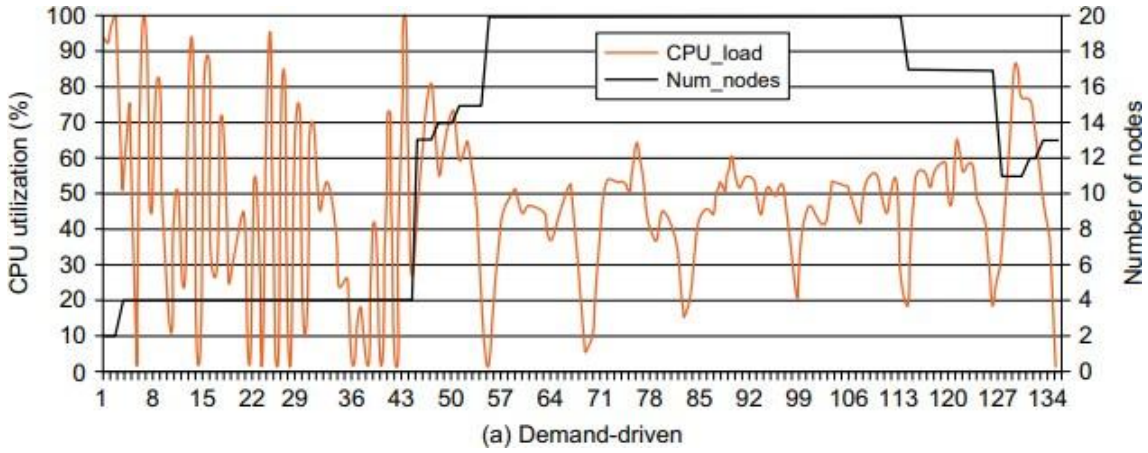
- Fixed capacity to a declining user demand could result in even worse resource waste.
- The user may give up the service by canceling the demand, resulting in reduced revenue for the provider.
- Both the user and provider may be losers in resource provisioning without elasticity.

Resource-provisioning methods are

- Demand-driven method - Provides static resources and has been used in grid computing
- Event-driven method - Based on predicted workload by time.
- Popularity-Driven Resource Provisioning – Based on Internet traffic monitored

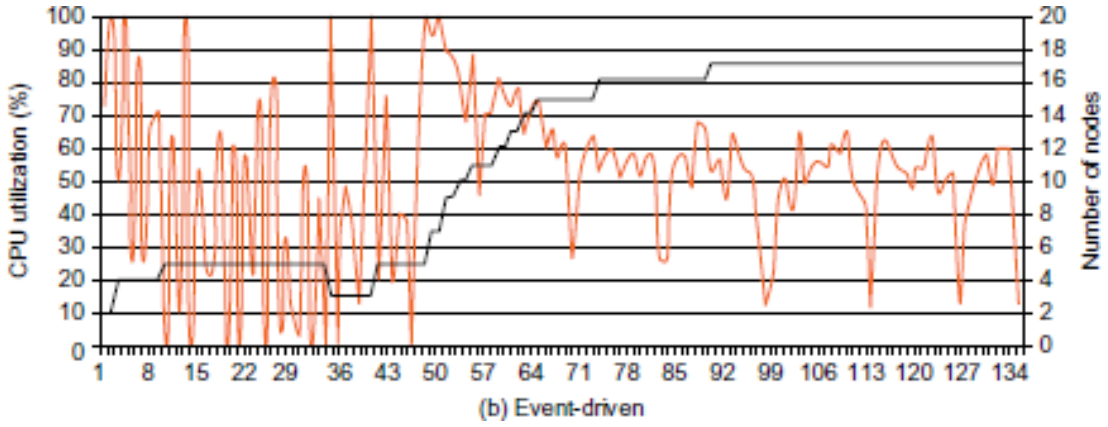
Demand Driven Methods

- Provides Static resources
- This method adds or removes nodes (VM) based on the current utilization(Use) level of the allocated resources.
- When a resource has surpassed (exceeded) a threshold (Upperlimit) for a certain amount of time, the scheme increases the resource (nodes) based on demand.
- When a resource is below a threshold for a certain amount of time, then resources could be decreased accordingly.
- This method is easy to implement.
- The scheme does not work out properly if the workload changes abruptly.



Event-Driven Resource Provisioning

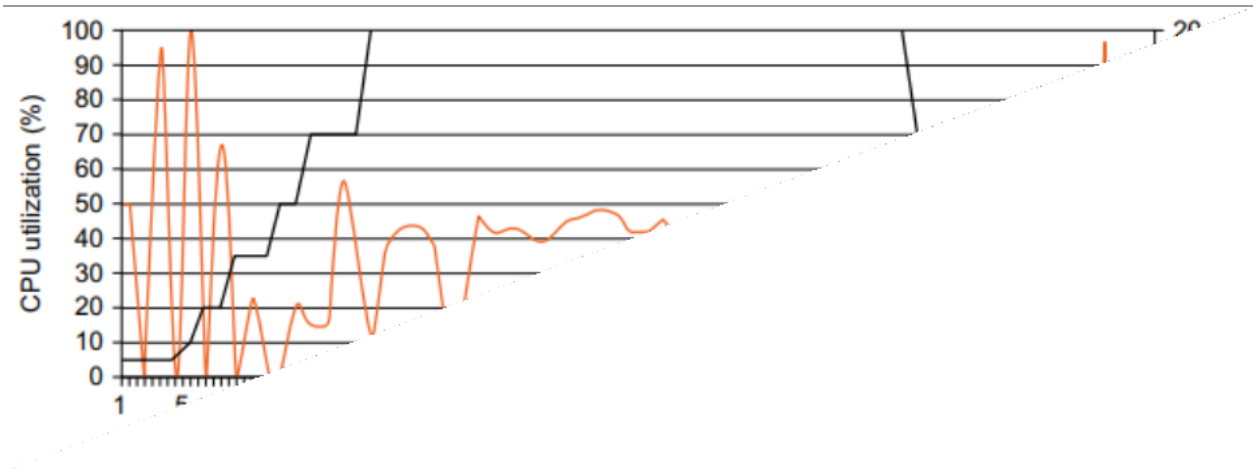
- This scheme adds or removes machine instances based on a specific time event.
- The scheme works better for seasonal or predicted events such as Christmastime in the West and the Lunar New Year in the East.
- During these events, the number of users grows before the event period and then decreases during the event period. This scheme anticipates peak traffic before it happens.
- The method results in a minimal loss of QoS, if the event is predicted correctly



Popularity-Driven Resource Provisioning

- Internet searches for popularity of certain applications and allocates resources by popularity demand.

- This scheme has a minimal loss of QoS, if the predicted popularity is correct.
- Resources may be wasted if traffic does not occur as expected.
- Again, the scheme has a minimal loss of QoS, if the predicted popularity is correct.
- Resources may be wasted if traffic does not occur as expected.



Dynamic Resource Deployment

- The cloud uses VMs as building blocks to create an execution environment across multiple resource sites.
- Dynamic resource deployment can be implemented to achieve scalability in performance.
- Peering arrangements established between gateways enable the allocation of resources from multiple grids to establish the execution environment.
- Dynamic resource deployment can be implemented to achieve scalability in performance.
- InterGrid is used for interconnecting distributed computing infrastructures.
- InterGrid provides an execution environment on top of the interconnected infrastructures.
- IGG(InterGridGateway) allocates resources from an
  - Organization's local cluster (Or)
  - Cloud provider.
- Under peak demands, IGG interacts with another IGG that can allocate resources from a cloud computing provider.
- Component called the DVE manager performs resource allocation and management.
- Intergrid gateway (IGG) allocates resources from a local cluster three steps:

(1) Requesting the VMs(Resources)

(2) Enacting (Validate) the leases

(3) Deploying (install) the VMs as requested.

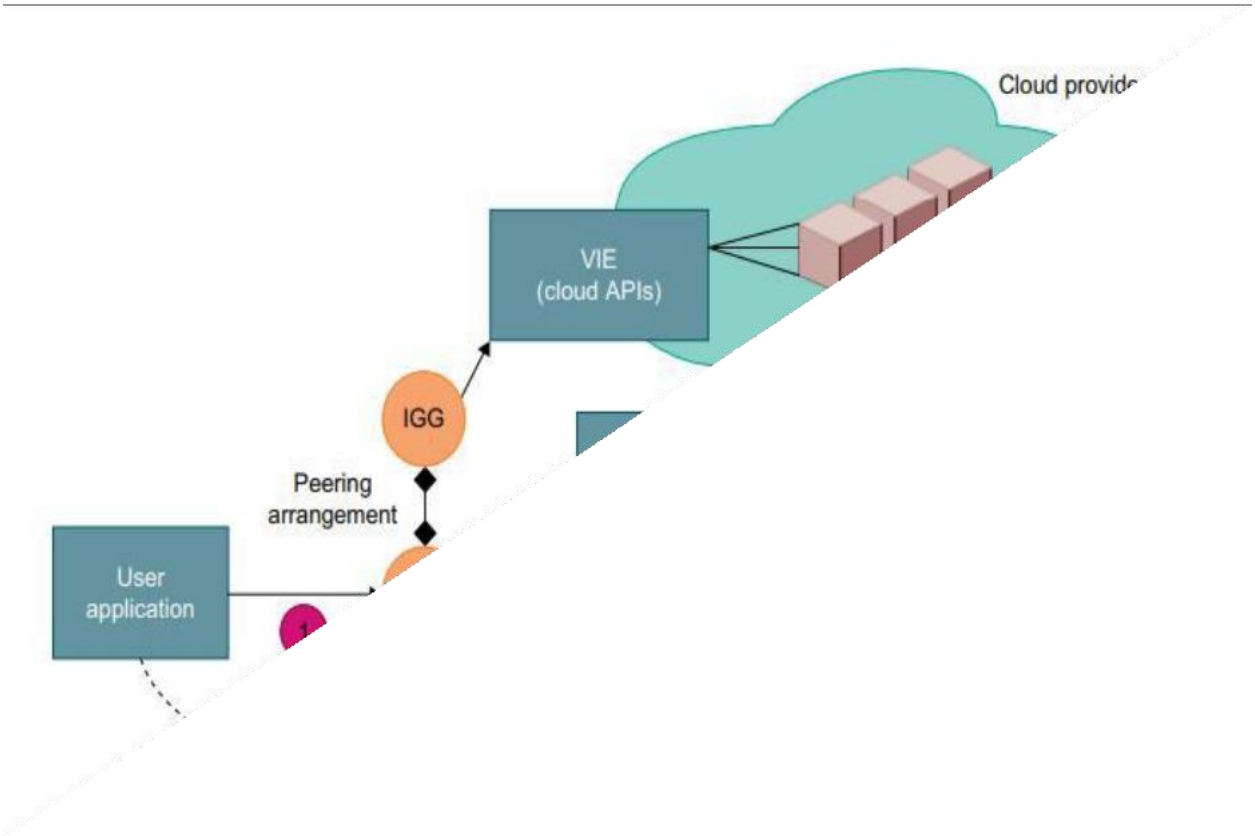


Fig: Cloud resource deployment using an IGG (intergrid gateway) to allocate the VMs from a Local cluster to interact with the IGG of a public cloud provider.

- Under peak demand, this IGG interacts with another IGG that can allocate resources from a cloud computing provider.
- A grid has predefined peering arrangements with other grids, which the IGG manages.
- Through multiple IGGs, the system coordinates the use of InterGrid resources.
- An IGG is aware of the peering terms with other grids, selects suitable grids that can provide the required resources, and replies to requests from other IGGs.
- Request redirection policies determine which peering grid InterGrid selects to process a request and a price for which that grid will perform the task.
- An IGG can also allocate resources from a cloud provider.
- The InterGrid allocates and provides a distributed virtual environment (DVE).

- This is a virtual cluster of VMs that runs isolated from other virtual clusters.
- A component called the DVE manager performs resource allocation and management on behalf of specific user applications.
- The core component of the IGG is a scheduler for implementing provisioning policies and peering with other gateways.
- The communication component provides an asynchronous message-passing mechanism.

Provisioning of Storage Resources

- Storage layer is built on top of the physical or virtual servers.
- Data is stored in the clusters of the cloud provider.
- The service can be accessed anywhere in the world.

- Eg:
  - E-mail system might have millions of users and each user can have thousands of e-mails and consume multiple gigabytes of disk space.
  - Web searching application.
  - To store huge amount of information solid-state drives are used instead of hard disk drives

In storage technologies, hard disk drives may be augmented (increased) with solid-state drives in the future.

Table 4.8 Storage Services in Three Cloud Computing Systems

Storage System	Features
GFS: Google File System	Very large sustainable reading and writing bandwidth, mostly continuous accessing instead of random accessing. The programming interface is similar to that of the POSIX file system accessing interface.
HDFS: Hadoop Distributed File System	The open source clone of GFS. Written in Java. The programming interfaces are similar to POSIX but not identical.
Amazon S3 and EBS	S3 is used for retrieving and storing data from/to remote servers. EBS is built on top of S3 for using virtual disks in running EC2 instances.

Virtual Machine Creation and Management

The managers provide a public API for users to submit and control the VMs

.

Fig. Virtual Machine Creation and Management

Independent Service Management:

- Independent services request facilities to execute many unrelated tasks.
- Commonly, the APIs provided are some web services that the developer can use conveniently.

Running Third-Party Applications

- Cloud platforms have to provide support for building applications that are constructed by third-party
  application providers or programmers.

- The APIs are often in the form of services.
- Web service application engines are often used by programmers for building applications.
- The web browsers are the user interface for end users.

Virtual Machine Manager

The manager manage VMs deployed on a set of physical resources

- VIEs(Virtual Infrastructure Engine) can create and stop VMs on a physical cluster
- Users submit VMs on physical machines using different kinds of hypervisors

- To deploy a VM, the manager needs to use its template.
- Virtual Machine Templates contains a description for a VM with the following static information:
    - The number of cores or processors to be assigned to the VM
    - The amount of memory the VM requires
    - The kernel used to boot the VM's operating system.
    - The price per hour of using a VM
- OAR/Kadeploy is a deployment tool
- API(Application Programming Interface) - An API is a software intermediary that makes it possible for application programs to interact with each other and share data

Virtual Machine Templates

- A VM template is analogous to a computer's configuration and contains a description for a VM with the following static information:
- The number of cores or processors to be assigned to the VM
- The amount of memory the VM requires
- The kernel used to boot the VM's operating system
- The disk image containing the VM's file system
- The price per hour of using a VM

Distributed VM Management

- A distributed VM manager makes requests for VMs and queries their status.
- This manager requests VMs from the gateway on behalf of the user application.
- The manager obtains the list of requested VMs from the gateway.
- This list contains a tuple of public IP/private IP addresses for each VM with Secure Shell (SSH) tunnels.

Global Exchange of Cloud Resources

- Cloud infrastructure providers (i.e., IaaS providers) have established data centers in multiple geographical locations to provide redundancy and ensure reliability in case of site failures.
- Amazon does not provide seamless/automatic mechanisms for scaling its hosted services across multiple geographically distributed data centers.
- This approach has many shortcomings
- First, it is difficult for cloud customers to determine in advance the best location for hosting their services as they may not know the origin of consumers of their services.
- Second, SaaS providers may not be able to meet the QoS expectations of their service consumers originating from multiple geographical locations.
- The figure the high-level components of the Melbourne group's proposed InterCloud architecture

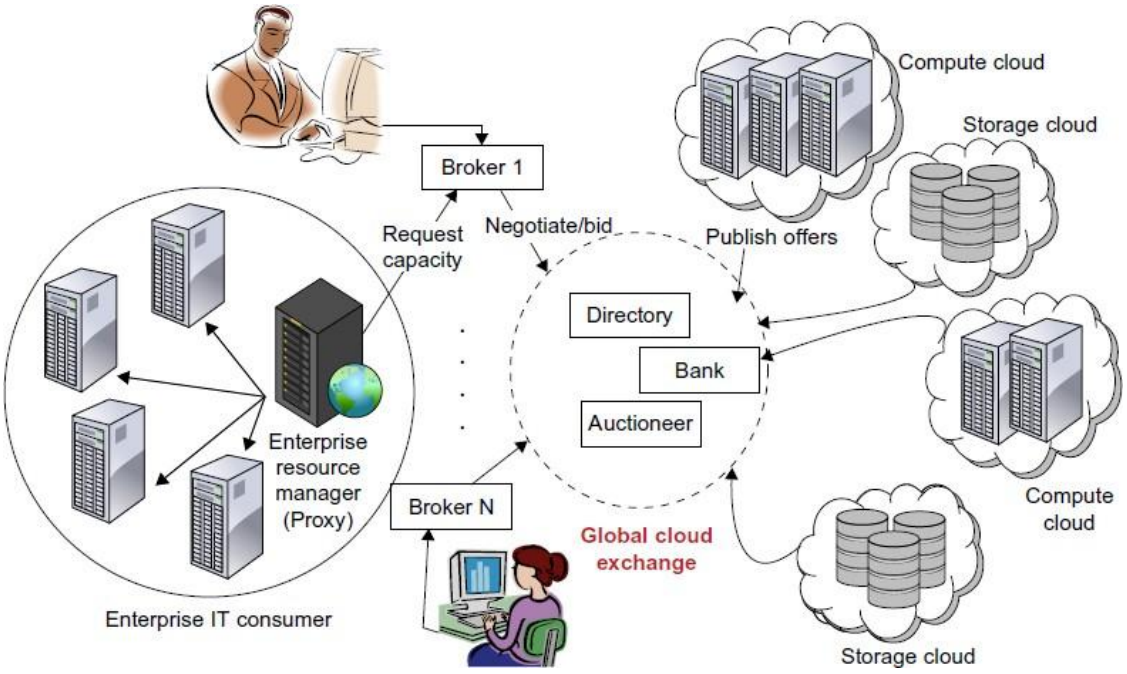


Fig: Inter-cloud exchange of cloud resources through brokering

- It is **not possible** for a cloud infrastructure provider to establish its **data centers at all possible locations** throughout the world.
- This results in **difficulty** in meeting the **QOS expectations** of their customers.

- Hence, services of **multiple cloud infrastructure** service providers are used.
- **Cloud coordinator** evaluates the available resources.
- The availability of a banking system ensures that financial transactions related to SLAs are carried out in a securely.
- By realizing InterCloud architectural principles in mechanisms in their offering, cloud providers will be able to dynamically expand or resize their provisioning capability based on sudden spikes in workload demands by leasing available computational and storage capabilities from other cloud.
- They consist of client brokering and coordinator services that support utility-driven federation of clouds:
    - application scheduling
    - resource allocation
    - migration of workloads.
- The architecture cohesively couples the administratively and topologically distributed storage and compute capabilities of clouds as part of a single resource leasing abstraction.
- The system will ease the crossdomain capability integration for on-demand, flexible, energy-efficient, and reliable access to the infrastructure based on virtualization technology
- The Cloud Exchange (CEx) acts as a market maker for bringing together service producers and consumers.
- It aggregates the infrastructure demands from application brokers and evaluates them against the available supply currently published by the cloud coordinators.
- It supports trading of cloud services based on competitive economic models such as commodity markets and auctions.
- CEx allows participants to locate providers and consumers with fitting offers.

Security

- Virtual machines from multiple organizations have to be co-located on the same physical server in order to maximize the efficiencies of virtualization.

- Cloud service providers must learn from the managed service provider (MSP) model and ensure that their customers' applications and data are secure if they hope to retain their customer base and competitiveness.
- Cloud environment should be free from abuses, cheating, hacking, viruses, rumors, and privacy and copyright violations.

Cloud Security Challenges

- In cloud model users lose control over physical security.
- In a public cloud, users are sharing computing resources with other companies.
- When users share the environment in the cloud, it results in data at risk of seizure (attack).
- Storage services provided by one cloud vendor may be incompatible with another vendor's services; this results in unable to move from one to the other.
- Vendors create "sticky services".
- Sticky services are the services which makes end user, in difficulty while transporting from one cloud vendor to another.

Example: Amazon's "Simple Storage Service" [S3] is incompatible with IBM's Blue Cloud, or Google, or Dell).

- Customers want their data encrypted while **data is at rest** (data stored) in the cloud vendor's storage pool.
- Data integrity means ensuring that data is identically maintained during any operation (such as transfer, storage, or retrieval).
- Data integrity is assurance that the data is consistent and correct.
- One of the key challenges in cloud computing is data-level security.
- It is difficult for a customer to find where its data resides on a network controlled by its provider.
- Some countries have strict limits on what data about its citizens can be stored and for how long.
- Banking regulators require that customers' financial data remain in their home country.

- Security managers will need to pay particular attention to systems that contain critical data such as corporate financial information.
- Outsourcing (giving rights to third party) loses control over data and not a good idea from a security perspective.
- Security managers have to interact with company's legal staff to ensure that appropriate contract terms are in place to protect corporate data.
- Cloud-based services will result in many mobile IT users accessing business data and services without traversing the corporate network.
- This will increase the need for enterprises to place security controls between mobile users and cloud-based services.
- Placing large amounts of sensitive data in a globally accessible cloud leaves organizations open to large distributed threats—attackers no longer have to come onto the premises to steal data, and they can find it all in the one "virtual" location.
- Virtualization efficiencies in the cloud require virtual machines from multiple organizations to be collocated on the same physical resources.
- Although traditional data center security still applies in the cloud environment, physical segregation and hardware-based security cannot protect against attacks between virtual machines on the same server.
- The dynamic and fluid nature of virtual machines will make it difficult to maintain the consistency of security and ensure the auditability of records.
- The ease of cloning and distribution between physical servers could result in the propagation of configuration errors and other vulnerabilities.
- Localized virtual machines and physical servers use the same operating systems as well as enterprise and web applications in a cloud server environment, increasing the threat of an attacker or malware exploiting vulnerabilities in these systems and applications remotely.
- Virtual machines are vulnerable as they move between the private cloud and the public cloud.
- Operating system and application files are on a shared physical infrastructure in a virtualized cloud environment and require system, file, and activity monitoring to provide

confidence and auditable proof to enterprise customers that their resources have not been compromised or tampered with.

- The **Intrusion Detection System(IDS)** and **Intrusion Prevention Systems(IPS)** detects malicious activity at virtual machine level.
- The co-location of multiple virtual machines increases the threat from attacker.
- If Virtual machines and physical machine use the same operating systems in a cloud environment, increases the threat from an attacker.
- A fully or partially shared cloud environment is expected to have a greater attack than own resources environment.
- Virtual machines must be self-defending.
- Cloud computing provider is incharge of customer data security and privacy.

Software as a Service Security (Or) Data Security (Or) Application Security (Or) Virtual Machine Security.

Cloud computing models of the future will likely combine the use of SaaS (and other XaaS's as appropriate), utility computing, and Web 2.0 collaboration technologies to leverage the Internet to satisfy their customers' needs. New business models being developed as a result of the move to cloudcomputing are creating not only new technologies and business operational processes but also newsecurity requirements and challenges

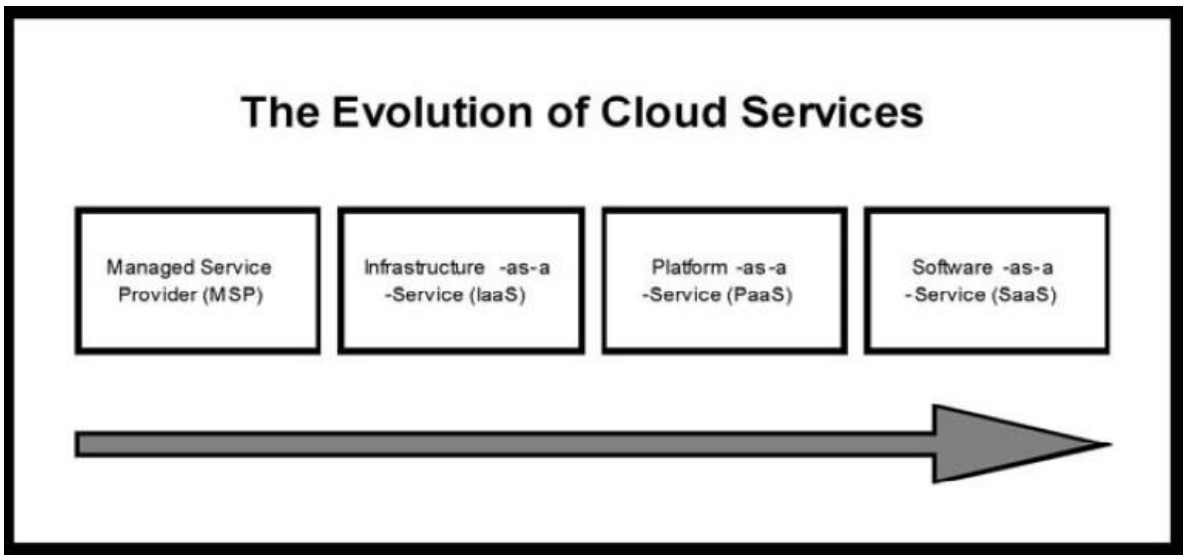


Fig: Evolution of Cloud Services

SaaS plays the dominant cloud service model and this is the area where the most critical need for security practices are required

 Security issues that are discussed with cloud-computing vendor:

1. **Privileged user access**—Inquire about who has specialized access to data, and about the hiring and management of such administrators.
2. **Regulatory compliance**—Make sure that the vendor is willing to undergo external audits and/or security certifications.
3. **Data location**—Does the provider allow for any control over the location of data?
4. **Data segregation**—Make sure that encryption is available at all stages, and that these encryption schemes were designed and tested by experienced professionals.
5. **Recovery**—Find out what will happen to data in the case of a disaster. Do they offer complete restoration? If so, how long would that take?
6. **Investigative support**—Does the vendor have the ability to investigate any inappropriate or illegal activity?
7. **Long-term viability**—What will happen to data if the company goes out of business? How will data be returned, and in what format?

The security practices for the SaaS environment are as follows:

Security Management (People)

- One of the most important actions for a security team is to develop a formal charter for the security organization and program.
- This will foster a shared vision among the team of what security leadership is driving toward and expects, and will also foster "ownership" in the success of the collective team.
- The charter should be aligned with the strategic plan of the organization or company the security team works for.

Security Governance

- A security committee should be developed whose objective is to focus on providing guidance about security initiatives with business and IT strategies.
- A charter for the security team is typically one of the first deliverables from the steering committee.

- This charter must clearly define the roles and responsibilities of the security team and other groups involved in performing information security functions.
- Lack of a formalized strategy can lead to an unsustainable operating model and
- security level as it evolves.
- In addition, lack of attention to security governance can result in key needs of the business not being met, including but not limited to, risk management, security monitoring, applicationsecurity, and sales support.
- Lack of proper governance and management of duties can also result in potential security risks being left unaddressed and opportunities to improve the business being missed.
- The security team is not focused on the key security functions and activities that are critical to the business.

Cloud security governance refers to the management model that facilitates effective and efficient security management and operations in the cloud environment so that an enterprise's business targets are achieved. This model incorporates a hierarchy of executive mandates, performance expectations, operational practices, structures, and metrics that, when implemented, result in the optimization of business value for an enterprise. Cloud security governance helps answer leadership questions such as:

- Are our security investments yielding the desired returns?
- Do we know our security risks and their business impact?
- Are we progressively reducing security risks to acceptable levels?
- Have we established a security-conscious culture within the enterprise?

Strategic alignment, value delivery, risk mitigation, effective use of resources, and performance measurement are key objectives of any IT-related governance model, security included. To successfully pursue and achieve these objectives, it is important to understand the operational culture and business and customer profiles of an enterprise, so that an effective security governance model can be customized for the enterprise.

Cloud Security Governance Challenges

Whether developing a governance model from the start or having to retrofit one on existing investments in cloud, these are some of the common challenges:

Lack of senior management participation and buy-in

The lack of a senior management influenced and endorsed security policy is one of the common challenges facing cloud customers. An enterprise security policy is intended to set the executive tone, principles and expectations for security management and operations in the cloud. However, many enterprises tend to author security policies that are often laden with tactical content, and lack executive input or influence. The result of this situation is the ineffective definition and communication of executive tone and expectations for security in the cloud.

Lack of embedded management operational controls

Another common cloud security governance challenge is lack of embedded management controls into cloud security operational processes and procedures. Controls are often interpreted as an auditor's checklist or repackaged as procedures, and as a result, are not effectively embedded into security operational processes and procedures as they should be, for purposes of optimizing value and reducing day-to-day operational risks. This lack of embedded controls may result in operational risks that may not be apparent to the enterprise. For example, the security configuration of a device may be modified (change event) by a staffer without proper analysis of the business impact (control) of the modification. The net result could be the introduction of exploitable security weaknesses that may not have been apparent with this modification.

Lack of operating model, roles, and responsibilities

Many enterprises moving into the cloud environment tend to lack a formal operating model for security, or do not have strategic and tactical roles and responsibilities properly defined and operationalized. This situation stifles the effectiveness of a security management and operational function/organization to support security in the cloud. Simply, establishing a hierarchy that includes designating an accountable official at the top, supported by a stakeholder committee, management team, operational staff, and third-party provider support (in that order) can help an enterprise to better manage and control security in the cloud, and protect associated investments in accordance with enterprise business goals.

Lack of metrics for measuring performance and risk

Another major challenge for cloud customers is the lack of defined metrics to measure security performance and risks – a problem that also stifles executive visibility into the real security risks in the cloud. This challenge is directly attributable to the combination of other challenges discussed above. For example, a metric that quantitatively measures the number of exploitable security vulnerabilities on host devices in the cloud over time can be leveraged as an indicator of risk in the host device environment. Similarly, a metric that measures the number of user-reported security incidents over a given period can be leveraged as a performance indicator

of staff awareness and training efforts. Metrics enable executive visibility into the extent to which security tone and expectations (per established policy) are being met within the enterprise and support prompt decision-making in reducing risks or rewarding performance as appropriate. The challenges described above clearly highlight the need for cloud customers to establish a framework to effectively manage and support security in cloud management, so that the pursuit of business targets are not potentially compromised. Unless tone and expectations for cloud security are established (via an enterprise policy) to drive operational processes and procedures with embedded management controls, it is very difficult to determine or evaluate business value, performance, resource effectiveness, and risks regarding security operations in the cloud. Cloud security governance facilitates the institution of a model that helps enterprises explicitly address the challenges described above.

Key Objectives for Cloud Security Governance

Building a cloud security governance model for an enterprise requires strategic-level security management competencies in combination with the use of appropriate security standards and frameworks (e.g., NIST, ISO, CSA) and the adoption of a governance framework (e.g., COBIT). The first step is to visualize the overall governance structure, inherent components, and to direct its effective design and implementation. The use of appropriate security standards and frameworks allow for a minimum standard of security controls to be implemented in the cloud, while also meeting customer and regulatory compliance obligations where applicable. A governance framework provides referential guidance and best practices for establishing the governance model for security in the cloud. The following represents key objectives to pursue in establishing a governance model for security in the cloud. These objectives assume that appropriate security standards and a governance framework have been chosen based on the enterprise's business targets, customer profile, and obligations for protecting data and other information assets in the cloud environment.

1. **Strategic Alignment**

   Enterprises should mandate that security investments, services, and projects in the cloud are executed to achieve established business goals (e.g., market competitiveness, financial, or operationalperformance).

2. **Value Delivery**

   Enterprises should define, operationalize, and maintain an appropriate security function/organization with appropriate strategic and tactical representation, and charged with the

responsibility to maximize the business value (Key Goal Indicators, ROI) from the pursuit of security initiatives in the cloud.

3. **Risk Mitigation**

   Security initiatives in the cloud should be subject to measurements that gauge effectiveness in mitigating risk to the enterprise (Key Risk Indicators). These initiatives should also yield results that progressively demonstrate a reduction in these risks over time.

4. **Effective Use of Resources**

   It is important for enterprises to establish a practical operating model for managing and performing security operations in the cloud, including the proper definition and operationalization of due processes, the institution of appropriate roles and responsibilities, and use of relevant tools for overall efficiency and effectiveness.

5. **Sustained Performance**

   Security initiatives in the cloud should be measurable in terms of performance, value and risk to the enterprise (Key Performance Indicators, Key Risk Indicators), and yield results that demonstrate attainment of desired targets (Key Goal Indicators) over time.

Risk Management

- Effective risk management entails identification of technology assets; identification of data and its links to business processes, applications, and data stores; and assignment of ownership and custodial responsibilities.
- Actions should also include maintaining a repository of information assets
- A risk assessment process should be created that allocates security resources related to business continuity.

Risk Assessment

- Security risk assessment is critical to helping the information security organization make informed decisions when balancing the dueling priorities of business utility and protection of assets.
- Lack of attention to completing formalized risk assessments can contribute to an increase in information security audit findings, can jeopardize certification goals, and can lead to

inefficient and ineffective selection of security controls that may not adequately mitigate information security risks to an acceptable level.

Security Portfolio(selection) Management

- Security portfolio management ensures efficient and effective operation of any information.

Security Awareness

- Not providing proper awareness and training to the people who may need them can expose the company to a variety of security risks

Policies, Standards, and Guidelines

- Policies, standards, and guidelines are developed that can ensure consistency of performance.

Secure Software Development Life Cycle (SecSDLC)

- The SecSDLC involves identifying specific threats and the risks. The SDLC consists of six phases

Phase 1.Investigation:

-Define project goals, and document them.

Phase 2.Analysis:

-Analyze current threats and perform risk analysis.

Phase 3.Logical design:

-Develop a security blueprint(plan) and business responses to disaster.

Phase 4.Physical design:

-Select technologies to support the security blueprint(plan).

Phase 5.Implementation:

- Buy or develop security solutions.

Phase 6.Maintenance:

-Constantly monitor, test, modify, update, and repair to respond to changing threats.

Security Monitoring and Incident Response

- Centralized security management systems should be used to provide notification of security vulnerabilities and to monitor systems continuously.

Business Continuity Plan

Business continuity plan, ensures uninterrupted operations of business.

Forensics

Forensics includes recording and analyzing events to determine the nature and source of information abuse, security attacks, and other such incidents.

Security Architecture Design

A security architecture framework should be established with the following consideration

1. Authentication
2. Authorization
3. Availability
4. Confidentiality
5. Integrity
6. Privacy

Vulnerability Assessment

- Vulnerability assessment classifies network assets to more efficiently prioritize vulnerability-mitigation programs, such as patching and system upgrading.
- It measures the effectiveness of risk mitigation by setting goals of reduced vulnerability exposure and faster mitigation

Password Assurance Testing

- If the SaaS security team or its customers want to periodically test password strength by running
- password "crackers," they can use cloud computing to decrease crack time and pay only for what they use.
-

Security Images:

- Virtualization-based cloud computing provides the ability to create "Gold image" VM secure builds and to clone multiple copies.
- Gold image VMs also provide the ability to keep security up to date and reduce exposure by patching offline.

Data Privacy

- Depending on the size of the organization and the scale of operations, either an individual or a team should be assigned and given responsibility for maintaining privacy.
- A member of the security team who is responsible for privacy or security compliance team should collaborate with the company legal team to **address data privacy issues and concerns.**

- **Hiring a consultant** in privacy area, will ensure that your organization is prepared to meet the data privacy demands of its customers and regulators.

Data Governance

The data governance framework should include:

_ Data inventory

_ Data classification

_ Data analysis (business intelligence)

_ Data protection

_ Data privacy

_ Data retention/recovery/discovery

_ Data destruction

Data Security

The challenge in cloud computing is data-level security.

Security to data is given by

 Encrypting the data

 Permitting only specified users to access the data.

 Restricting the data not to cross the countries border.

For example, with data-level security, the enterprise can specify that this data is not allowed to go outside of the India.

Application Security

 This is collaborative effort between the security and product development team.

 Application security processes

o Secure coding guidelines

o Training

o Testing scripts

o Tools

 Penetration Testing is done to a System or application.

 Penetration Testing is defined as a type of Security Testing used to test the **insecure areas of the system or application**.

 The goal of this testing is to **find all the security vulnerabilities** that are present in the system being tested.

 SaaS providers should secure their web applications by following **Open Web Application Security Project (OWASP) guidelines** for secure application development, **by locking down ports** and unnecessary commands

5.3 Virtual Machine Security

In the cloud environment, physical servers are consolidated (combined) to multiple virtual machine instances.

Following are deployed on virtual machines to ensure security

 Firewalls

 Intrusion detection and prevention

 Integrity monitoring

 Log inspection

Virtual servers have security requirements identical to those of physical servers. The same applies to the applications and services they host. Virtualization provides security benefits: each virtual machine has a private security context, potentially with separate authentication and authorization rules, and with separate process, name and file system spaces. Deploying applications onto separate virtual machines provides better security control compared to running multiple applications on the same host operating system: penetrating one virtual machine's OS doesn't necessarily compromise workload and data residing in other virtual machines. Nonetheless, some practices should be kept in mind to prevent virtualization from introducing security vulnerabilities.

One aspect is physical security. Virtual infrastructure is not as 'visible' as physical infrastructure: there is no sticky label on a virtual machine to indicate its purpose and security classification. If a datacenter identifies servers with extremely high security requirements, and physically isolates them in a locked room or cage to prevent tampering or theft of data, then the physical machines hosting their virtualized workloads should be isolated in a similar way. Even without secured areas, many institutions keep workloads of different security classes on different servers. Those same isolation rules apply for virtual machines. Care should be taken to ensure

that the protected virtual machines are not migrated to a server in a less secure location. In the context of Oracle VM, this implies maintaining separate server pools, each with their own group of servers.

These rules of isolation should also be applied to networking: there are no color coded network cables to help staff identify and isolate different routes, segments and types network traffic to and from virtual machines or between them. There are no visual indicators that help ensure that application, management, and backup traffic are kept separate. Rather than plug network cables into different physical interfaces and switches, the Oracle VM administrator must ensure that the virtual network interfaces are connected to separate virtual networks. Specifically, use VLANs to isolate virtual machines from one another, and assign virtual networks for virtual machine traffic to different physical interfaces from those used for management, storage or backup. These can all be controled from the Oracle VM Manager user interface. Ensure that secure live migration is selected to guarantee that virtual machine memory data is not sent across the wire unencrypted.

Additional care must be given to virtual machine disk images. In most cases the virtual disks are made available over the network for migration and failover purposes. In many cases they are files, which could easily be copied and stolen if the security of network storage is compromised. Therefore it is essential to lock down the NAS or SAN environments and prevent unauthorized access. An intruder with root access to a workstation on the storage network could mount storage assets and copy or alter their contents. Use a separate network for transmission between the storage servers and the Oracle VM hosts to ensure its traffic is not made public and subject to being snooped. Make sure that unauthorized individuals are not permitted to log into the Oracle VM Servers, as that would give them access to the guests' virtual disk images, and potentially much more.

All of these steps require controlling access to the Oracle VM Manager and Oracle VM Server domain 0 instances. Network access to these hosts should be on a private network, and the user accounts able to log into any of the servers in the Oracle VM environment should be rigorously controlled, and limited to the smallest possible number of individuals.

Identity and access management architecture(IAM)

Basic concept and definitions of IAM functions for any service:

Authentication – is a process of verifying the identity of a user or a system.Authentication usually connotes a more roburst form of identification. In some use cases such as service – to- service interaction, authentication involves verifying the network service.

Authorization – is a process of determining the privileges the user or system is entitled to once the identity is established. Authorization usually follows theauthentication step and is used to determine whether the user or service has the necessary privileges to perform certain operations.

Auditing – Auditing entails the process of review and examination ofauthentication, authorization records and activities to determine the adequacy of IAMsystem controls, to verify complaints with established security policies and procedure,to detect breaches in security services and to recommend any changes that areindicated for counter measures

IAM Architecture and Practice

IAM is not a monolithic solution that can be easily deployed to gain capabilitiesimmediately. It is as much an aspect of architecture as it is acollection of technology components, processes, and standard practices. Standardenterprise IAM architecture encompasses several layers of technology, services, andprocesses. At the core of the deployment architecture is a directory service (such as LDAP or Active Directory) that acts as a repository for the identity, credential, and user attributes of the organization's user pool. The directory interacts with IAM technology components such as authentication, user management, provisioning, and federation services that support the standard IAM practice and processes within the organization.

The IAM processes to support the business can be broadly categorized as follows:

User management: Activities for the effective governance and management of identity life cycles

Authentication management: Activities for the effective governance and management of the process for determining that an entity is who or what it claims tobe.

Authorization management: Activities for the effective governance and management of the process for determining entitlement rights that decide what resources an entity is permitted to access in accordance with the organization's policies.

Access management: Enforcement of policies for access control in response to a request from an entity (user, services) wanting to access an IT resource within the organization.

Data management and provisioning: Propagation of identity and data for authorization to IT resources via automated or manual processes.

Monitoring and auditing: Monitoring, auditing, and reporting compliance by users regarding access to resources within the organization based on the defined policies.

IAM processes support the following operational activities:

Provisioning: Provisioning can be thought of as a combination of the duties of the human resources and IT departments, where users are given access to data repositories or systems, applications, and databases based on a unique user identity. Deprovisioning works in the opposite manner, resulting in the deletion or deactivation of an identity or of privileges assigned to the user identity.

Credential and attribute management: These processes are designed to manage the life cycle of credentials and user attributes— create, issue, manage, revoke—to inappropriate account use. Credentials are usually bound to an individual and are verified during the authentication process. The processes include provisioning of attributes, static (e.g., standard text password) and dynamic (e.g., one-time password) credentials that comply with a password standard (e.g., passwords resistant to dictionary attacks), handling password expiration, encryption management of credentials during transit and at rest, and access policies of user attributes (privacy and handling of attributes for various regulatory reasons).Minimize the business risk associated with

identityimpersonation

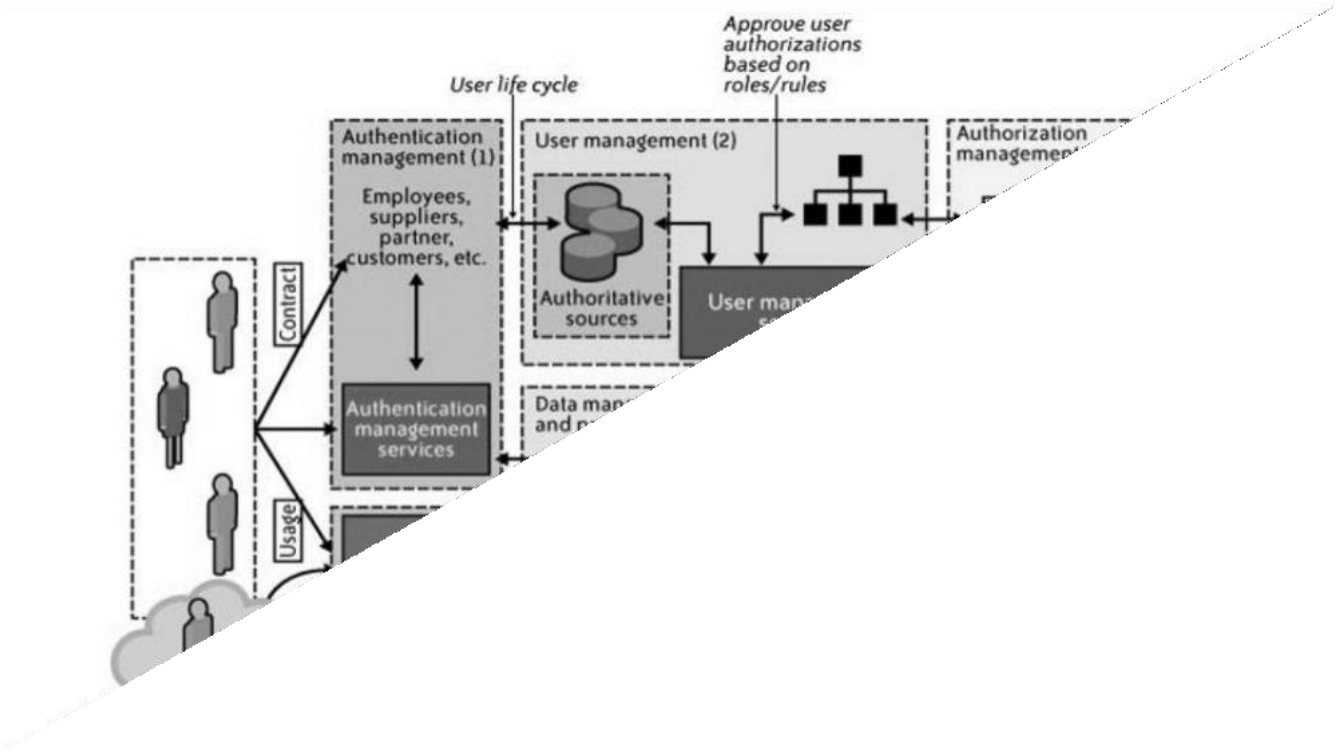


Figure 5.7 Enterprise IAM functional architecture

Entitlement management: Entitlements are also referred to as authorization policies. The processes in this domain address the provisioning and deprovisioning of privileges needed for the user to access resources including systems, applications, and databases. Proper entitlement management ensures that users are assigned only the required privileges.

Compliance management: This process implies that access rights and privileges are monitored and tracked to ensure the security of an enterprise's resources. The process also helps auditors verify compliance to various internal access control policies, and standards that include practices such as segregation of duties, access monitoring, periodic auditing, and reporting. An example is a user certification process that allows application owners to certify that only authorized users have the privileges necessary to access business-sensitive information.

Identity federation management: Federation is the process of managing the trust relationships established beyond the internal network boundaries or administrative domain boundaries among distinct organizations. A federation is an association of organizations that come together to exchange information about their users and resources to enable collaborations and transactions.

Centralization of authentication (authN) and authorization (authZ): A central authentication and authorization infrastructure alleviates the need for application developers to build custom authentication and authorization features into their applications. Furthermore, it promotes a loose coupling architecture where applications become agnostic to the authentication methods and policies. This approach is also called an ―externalization of authN and authZ from applications

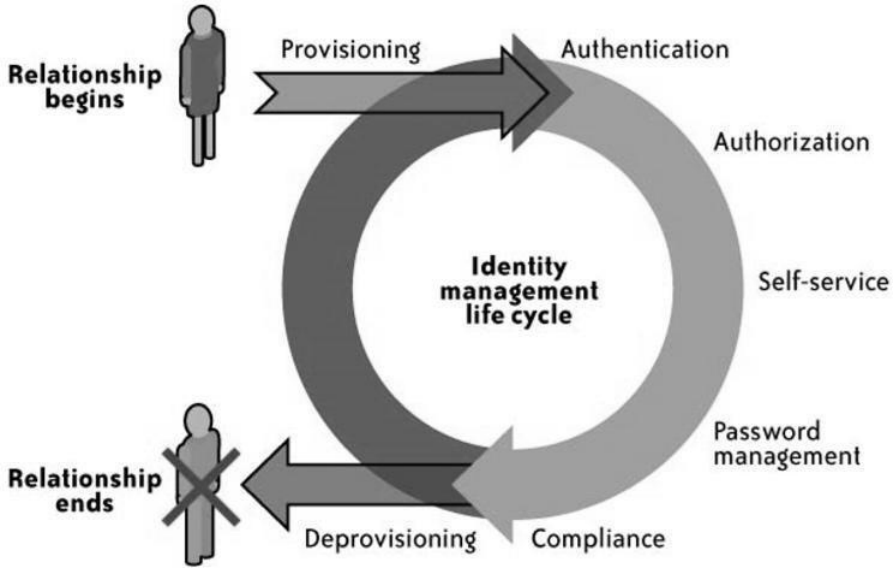


Figure 5.8 Identity Life cycle

IAM Standards and Specifications for Organisations

The following IAM standards and specifications will help organizations implement effective and efficient user access management practices and processes inthe cloud. These sections are ordered by four major challenges in user and access management faced by cloud users:

1. How can I avoid duplication of identity, attributes, and credentials and provide a single sign-on user experience for my users? SAML.
2. How can I automatically provision user accounts with cloud services and automate the process of provisoning and deprovisioning? SPML.

IAM Practices in the Cloud

When compared to the traditional applications deployment model within the enterprise, IAM practices in the cloud are still evolving. In the current state of IAM technology, standards support by CSPs (SaaS, PaaS, and IaaS) is not consistent across providers. Although large providers such as Google, Microsoft, and Salesforce.com seem to demonstrate basic IAM

capabilities, our assessment is that they still fall short of enterprise IAM requirements for managing regulatory, privacy, and data protection requirements. The maturity model takes into account the dynamic nature of IAM users, systems, and applications in the cloud and addresses the four key components of the IAM automation process:

- User Management, New Users
- User Management, User Modifications
- Authentication Management
- Authorization Management

IAM practices and processes are applicable to cloud services; they need to be adjusted to the cloud environment. Broadly speaking, user management functions in the cloud can be categorized as follows:

- Cloud identity administration, Federation or SSO
- Authorization management
- Compliance management

Cloud Identity Administration: Cloud identity administrative functions should focus on life cycle management of user identities in the cloud—provisioning, deprovisioning, identity federation, SSO, password or credentials management, profile management, and administrative management. Organizations that are not capable of supporting federation should explore cloud-based identity management services. This new breed of services usually synchronizes an organization's internal directories with its directory (usually multitenant) and acts as a proxy IdP for the organization.

Federated Identity (SSO): Organizations planning to implement identity federation that enables SSO for users can take one of the following two paths (architectures):

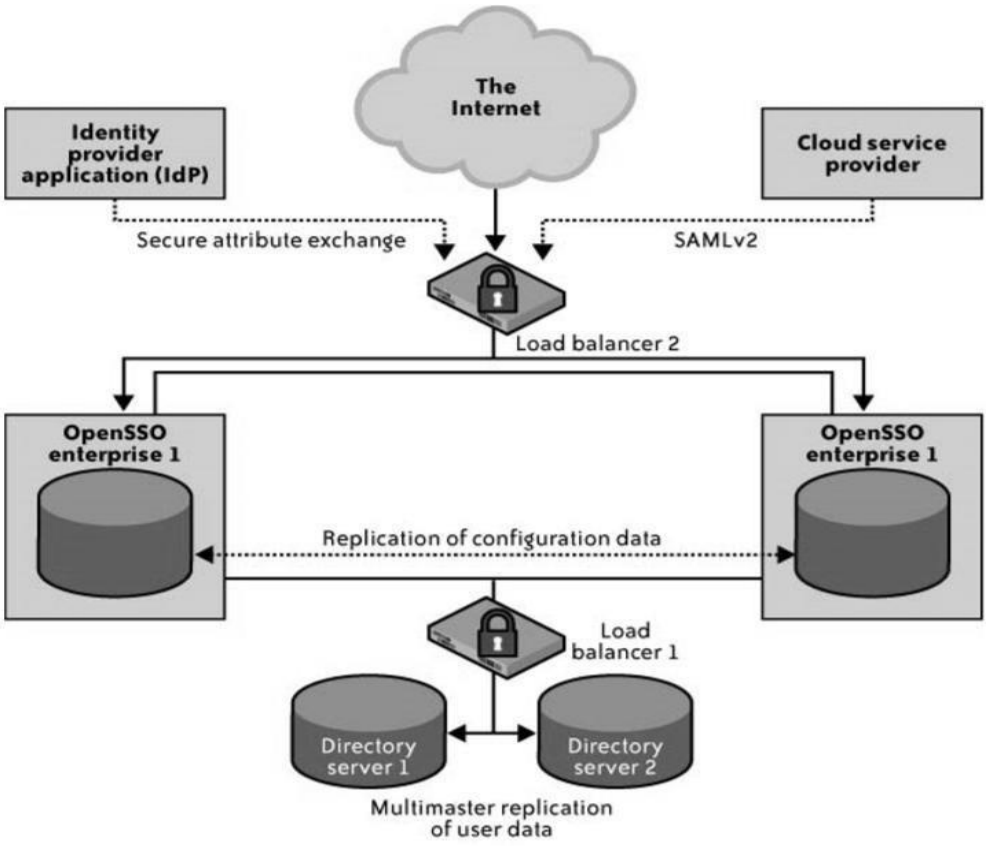
- Implement an enterprise IdP within an organization perimeter.
- Integrate with a trusted cloud-based identity management service provider.

Both architectures have pros and cons.

Enterprise identity provider: In this architecture, cloud services will delegate authentication to an organization's IdP. In this delegated authentication architecture, the organization federates identities within a trusted circle of CSP domains. A circle of trust can be created with all the domains that are authorized to delegate authentication to the IdP. In this deployment architecture,

where the organization will provide and support an IdP, greater control can be exercised over user identities, attributes, credentials, and policies for authenticating and authorizing users to a cloud service.

IdP deployment architecture.



Security standards

Security standards define the processes, procedures, and practices necessary for implementing a security program. These standards also apply to cloud-related IT activities and include specific steps that should be taken to ensure a secure environment is maintained that provides privacy and security of confidential information in a cloud environment. Security standards are based on a set of key principles intended to protect this type of trusted environment. Messaging standards, especially for security in the cloud, must also include nearly all the same considerations as any other IT security endeavor.

Security (SAML ,OAuth, OpenID, SSL/TLS)

A basic philosophy of security is to have layers of defense, a concept known as *defense in depth*. This means having overlapping systems designed to provide security even if one system fails. An example is a firewall working in conjunction with an intrusion-detection system (IDS). Defense in depth provides security because there is no single point of failure and no single-entry vector at which an attack can occur. No single security system is a solution by itself, so it is far better to secure all systems. This type of layered security is precisely what we are seeing develop in cloud computing. Traditionally, security was implemented at the endpoints, where the user controlled access. An organization had no choice except to put firewalls, IDSs, and antivirus software inside its own network. Today, with the advent of managed security services offered by cloud providers, additional security can be provided inside the cloud.

Security Assertion Markup Language (SAML)

SAML is an XML-based standard for communicating authentication, authorization, and attribute information among online partners. It allows businesses to securely send assertions between partner organizations regarding the identity and entitlements of a principal. The Organization for the Advancement of Structured Information Standards (OASIS) Security Services Technical Committee is in charge of defining, enhancing, and maintaining the SAML specifications.

SAML is built on a number of existing standards, namely, SOAP, HTTP, and XML. SAML relies on HTTP as its communications protocol and specifies the use of SOAP (currently, version 1.1). Most SAML transactions are expressed in a standardized form of XML. SAML assertions and protocols are specified using XML schema. Both SAML 1.1 and SAML 2.0 use digital signatures (based on the XML Signature standard) for authentication and message integrity. XML encryption is supported in SAML 2.0, though SAML 1.1 does not have encryption capabilities. SAML defines XML-based assertions and protocols, bindings, and profiles. The term SAML Core refers to the general syntax and semantics of SAML assertions as well as the protocol used to request and transmit those assertions from one system entity to another. SAML protocol refers to what is transmitted, not how it is transmitted. A SAML binding determines how SAML requests and responses map to standard messaging protocols. An important (synchronous) binding is the SAML SOAP binding.

SAML standardizes queries for, and responses that contain, user authentication, entitlements, and attribute information in an XML format. This format can then be used to request security information about a principal from a SAML authority. A SAML authority, sometimes called the asserting party, is a platform or application that can relay security information. The relying party (or assertion consumer or requesting party) is a partner site that receives the security information.

The exchanged information deals with a subject's authentication status, access authorization, and attribute information. A subject is an entity in a particular domain. A person identified by an email address is a subject, as might be a printer.

SAML assertions are usually transferred from identity providers to service providers. Assertions contain statements that service providers use to make access control decisions. Three types of statements are provided by SAML: authentication statements, attribute statements, and authorization decision statements. SAML assertions contain a packet of security information in this form:

<saml:Assertion A...>
<Authentication>
...
</Authentication>
<Attribute>
...
</Attribute>
<Authorization>
...
</Authorization>
</saml:Assertion A>

The assertion shown above is interpreted as follows:

Assertion A, issued at time T by issuer I, regarding subject
S, provided conditions C are valid.

Authentication statements assert to a service provider that the principal did indeed authenticate with an identity provider at a particular time using a particular method of authentication. Other information about the authenticated principal (called the authentication

context) may be disclosed in an authentication statement. An attribute statement asserts that a subject is associated with certain attributes. An attribute is simply a name-value pair. Relying parties use attributes to make access control decisions. An authorization decision statement asserts that a subject is permitted to perform action A on resource R given evidence E. The expressiveness of authorization decision statements in SAML is intentionally limited.

A SAML protocol describes how certain SAML elements (including assertions) are packaged within SAML request and response elements. It provides processing rules that SAML entities must adhere to when using these elements. Generally, a SAML protocol is a simple request-response protocol. The most important type of SAML protocol request is a query. A service provider makes a query directly to anidentity provider over a secure back channel. For this reason, query messages are typically bound to SOAP. Corresponding to the three types of statements, there are three types of SAML queries: the authentication query, the attribute query, and the authorization decision query. Of these, the attribute query is perhaps most important. The result of an attribute query is a SAML response containing anassertion, which itself contains an attribute statement.

Open Authentication (OAuth)

OAuth is an open protocol, initiated by Blaine Cook and Chris Messina, to allow secure API authorization in a simple, standardized method for various types of web applications. Cook and Messina had concluded that there were no open standards for API access delegation. The OAuth discussion group was created in April 2007, for the small group of implementers to write the draft proposal for an open protocol. DeWitt Clinton of Google learned of the OAuth project and expressed interest in supporting the effort. In July 2007 the team drafted an initial specification, and it was released in October of the same year. OAuth is a method for publishing and interacting with protected data. For developers, OAuth provides

users access to their data while protecting account credentials. OAuth allows users to grant access to their information, which is shared by the service provider and consumers without sharing all of their identity. The Core designation is used to stress that this is the baseline, and other extensions and protocols can build on it. By design, OAuth Core 1.0 does not provide many desired features (e.g., automated discovery of endpoints, language support, support for XML-RPC and SOAP, standard definition of resource access, OpenID integration, signing algorithms, etc.). This intentional lack of feature support is viewed by the authors as a significant

benefit. The Core deals with fundamental aspects of the protocol, namely, to establish a mechanism for exchanging a user name and password for a token with defined rights and to provide tools to protect the token. . In fact, OAuth by itself *provides no privacy at all* and depends on other protocols such as SSL to accomplish that.

OpenID

OpenID is an open, decentralized standard for user authentication and access control that allows users to log onto many services using the same digital identity. It is a single-sign-on (SSO) method of access control. As such, it replaces the common log-in process (i.e., a log-in name and a password) by allowing users to log in once and gain access to resources across participating systems. The original OpenID authentication protocol was developed in May 2005 by Brad Fitzpatrick, creator of the popular community web site Live-Journal. In late June 2005, discussions began between OpenID developers and other developers from an enterprise software company named Net-Mesh. These discussions led to further collaboration on interoperability between OpenID and NetMesh's similar Light-Weight Identity (LID) protocol. The direct result of the collaboration was the Yadis discovery protocol, which was announced on October 24, 2005.

The Yadis specification provides a general-purpose identifier for a person and any other entity, which canbe used with a variety of services. It provides a syntax for a resource description document identifying services available using that identifier and an interpretation of the elements of that document. Yadis discovery protocol is used for obtaining a resource description document, given that identifier. Together these enable coexistence and interoperability of a rich variety of services using a single identifier. The identifier uses a standard syntax and a well-established namespace and requires no additional namespace administration infrastructure.

An OpenID is in the form of a unique URL and is authenticated by the entity hosting the OpenID URL.The OpenID protocol does not rely on a central authority to authenticate a user's identity. Neither the OpenID protocol nor any web sites requiring identification can mandate that a specific type of authentication be used; nonstandard forms of authentication such as smart cards, biometrics, or ordinary passwords are allowed. A typical scenario for using OpenID might be something like this: A user visits a web site that displays an OpenID log-in form somewhere on the page. Unlike a typical log-in form, which has fields for user name and password, the OpenID

log-in form has only one field for the OpenID identifier (which is an OpenID URL). This form is connected to an implementation of an OpenID client library.

A user will have previously registered an OpenID identifier with an OpenID identity provider. The user types this OpenID identifier into the OpenID log-in form. The relying party then requests the web page located at that URL and reads an HTML link tag to discover the identity provider service URL. With OpenID 2.0, the client discovers the identity provider service URL by requesting the XRDS document (also called the Yadis document) with the content type **application/xrds+xml,** which may be available at the target URL but is always available for a target XRI.

There are two modes by which the relying party can communicate with the identity provider: **checkid_immediate** and **checkid_setup.** In **checkid_immediate,** the relying party requests that the provider not interact with the user. All communication is relayed through the user's browser without explicitly notifying the user. In **checkid_setup,** the user communicates with the provider server directly using the same web browser as is used to access the relying party site. The second option is more popular on the web.

To start a session, the relying party and the identity provider establish a shared secret—referenced by an associate handle—which the relying party then stores. Using **checkid_setup,** the relying party redirects the user's web browser to the identity provider so that the user can authenticate with the provider. The method of authentication varies, but typically, an OpenID identity provider prompts the user for a password, then asks whether the user trusts the relying party web site to receive his or her credentials and identity details. If the user declines the identity provider's request to trust the relying party web site, the browser is redirected to the relying party with a message indicating that authentication was rejected.

The site in turn refuses to authenticate the user. If the user accepts the identity provider's request to trust the relying party web site, the browser is redirected to the designated return page on the relying party web site along with the user's credentials. That relying party must then confirm that the credentials really came from the identity provider. If they had previously established a shared secret, the relying party can validate the shared secret received with the credentials against the one previously stored. In this case, the relying party is considered to be stateful, because it stores the shared secret between sessions (a process sometimes referred to as

persistence). In comparison, a stateless relying party must make background requests using the **check_authentication** method to be sure that the data came from the identity provider.

SSL/TLS

Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are cryptographically secure protocols designed to provide security and data integrity for communications over TCP/IP. TLS and SSL encrypt the segments of network connections at the transport layer. Several versions of the protocols are in general use in web browsers, email, instant messaging, and voice-over-IP. TLS is an IETF standard protocol which was last updated in RFC 5246.

The TLS protocol allows client/server applications to communicate across a network in a way specifically designed to prevent eavesdropping, tampering, and message forgery. TLS provides endpoint authentication and data confidentiality by using cryptography. TLS authentication is one-way—the server is authenticated, because the client already knows the server's identity. In this case, the client remains unauthenticated. At the browser level, this means that the browser has validated the server's certificate—more specifically, it has checked the digital signatures of the server certificate's issuing chain ofCertification Authorities (CAs).

Validation does not identify the server to the end user. For true identification, the end user must verify the identification information contained in the server's certificate (and, indeed, its whole issuing CA chain).This is the only way for the end user to know the "identity" of the server, and this is the only way identity can be securely established, verifying that the URL, name, or address that is being used is specified inthe server's certificate. Malicious web sites cannot use the valid certificate of another web site becausethey have no means to encrypt the transmission in a way that it can be decrypted with the valid certificate.

Since only a trusted CA can embed a URL in the certificate, this ensures that checking the apparent URL with the URL specified in the certificate is an acceptable way of identifying the site.TLS also supports a more secure bilateral connection mode whereby both ends of the connection can be assured that they are communicating with whom they believe they are connected. This is known asmutual (assured) authentication. Mutual authentication requires the TLS client-side to also maintain a certificate.

TLS involves three basic phases:

1. Peer negotiation for algorithm support
2. Key exchange and authentication
3. Symmetric cipher encryption and message authentication

During the first phase, the client and server negotiate cipher suites, which determine which ciphers are used; makes a decision on the key exchange and authentication algorithms to be used; and determines the message authentication codes. The key exchange and authentication algorithms are typically public key algorithms. The message authentication codes are made up from cryptographic hash functions. Once these decisions are made, data transfer may begin.

UNIT II CLOUD ENABLING TECHNOLOGIES

Service Oriented Architecture – REST and Systems of Systems – Web Services – Publish-Subscribe Model – Basics of Virtualization – Types of Virtualization – Implementation Levels of Virtualization – Virtualization Structures – Tools and Mechanisms – Virtualization of CPU – Memory – I/O Devices –Virtualization Support and Disaster Recovery.

Introduction

Web Service

- **Generic definition**

- Any application accessible to other applications over the Web.

- **Definition of the UDDI consortium**

- Web services are self-contained, modular business applications that have open, Internet-oriented, standards-based interfaces.

- **Definition of the World Wide Web Consortium (W3C)**

- A Web service is a software system designed to support interoperable machine-to-machine interaction over a network.
- It has an interface described in a machine-processable format (specifically WSDL).
- Other systems interact with the Web service using SOAP messages.

What is a Web Service?

- Web Services are Classes/Methods, NOT Servlets
- Loosely-coupled
- Encapsulate functionality to logical entities
- Reuse of code and functionality
- Distributed Architecture
- Standardized interface & established Internet protocols

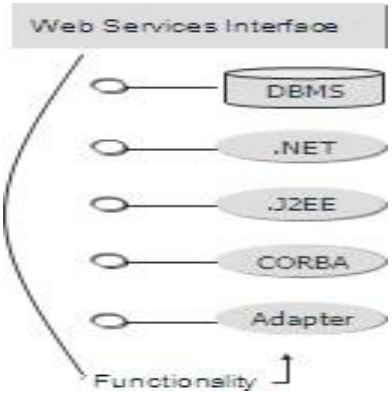


Figure 2.1 Service Interfaces

Characteristics of a Web Service

- A web service interface generally consists of a collection of operations that can be used by a client over the Internet.
- The operations in a web service may be provided by a variety of different resources, for example, programs, objects, or databases.
- The key characteristic of (most) web services is that they can process XML-formatted SOAP messages. An alternative is the REST approach.
- Each web service uses its own service description to deal with the service-specific characteristics of the messages it receives. Commercial examples include Amazon, Yahoo, Google and eBay.

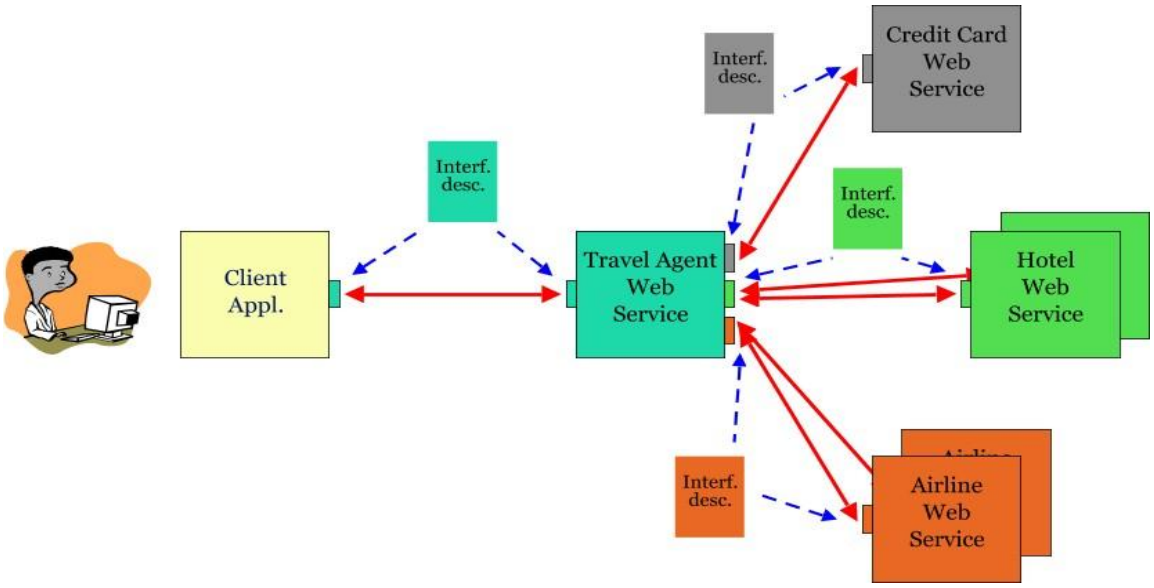


Figure 2.2 Example-Travel Agent Service

Remote Access

- IP Address - Locate a Computer
- URI - Uniform Resource Identifier
  - Locate a file in that Computer
- Socket and Port- Binding to a Method

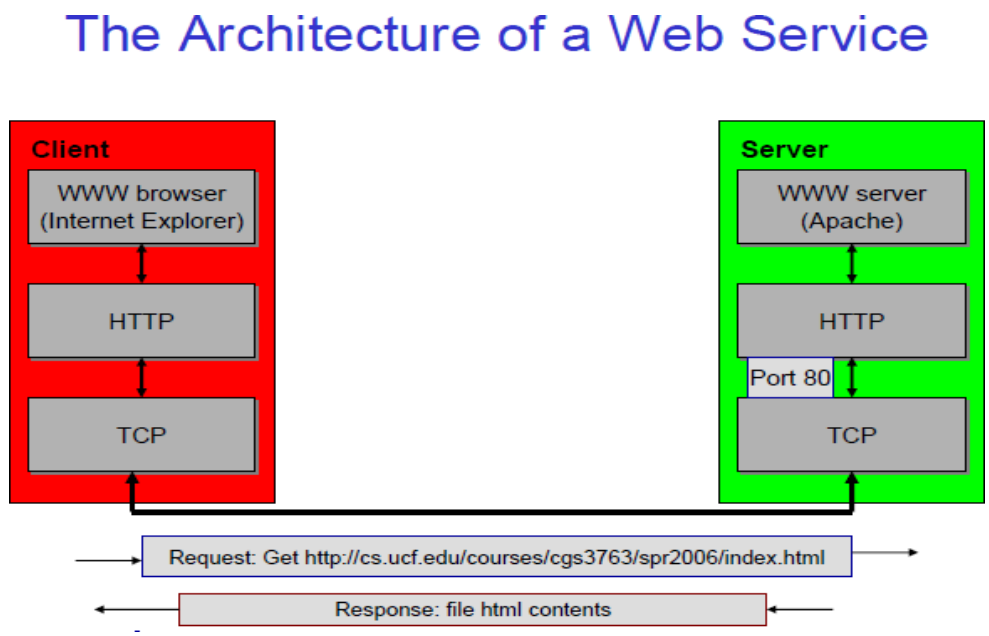
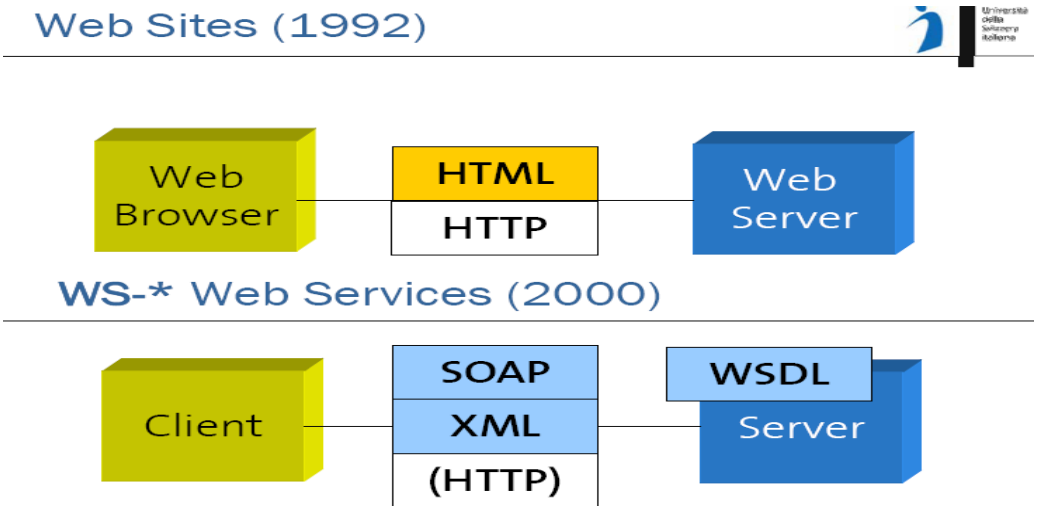


Figure 2.3 Web Service Architecture



SOA – Service Oriented Architecture

- Service provider publishes service description (WSDL), e.g. on a service broker
- Service Requester finds service (on service broker) and dynamically binds to service
- Enables ad-hoc collaboration and Enterprise Application Integration (EAI) within web-based information systems
- SOA is about how to design a software system that makes use of services of new or legacy applications through their published or discoverable interfaces.

- These applications are often distributed over the networks.
- SOA also aims to make service interoperability extensible and effective.
- It prompts architecture styles such as loose coupling, published interfaces and a standard communication model in order to support this goal.

Properties of SOA

- Logical view
- Message orientation
- Description orientation

Logical view

- The SOA is an abstracted, logical view of actual programs, databases, business processes.
- Defined in terms of what it does, typically carrying out a business-level operation.
- The service is formally defined in terms of the messages exchanged between provider agents and requester agents.

Message Orientation

- The internal structure of providers and requesters include the implementation language, process structure, and even database structure.
- These features are deliberately abstracted away in the SOA
- Using the SOA discipline one does not and should not need to know how an agent implementing a service is constructed.
- The key benefit of this concerns legacy systems.
- By avoiding any knowledge of the internal structure of an agent, one can incorporate any software component or application to adhere to the formal service definition.

Description orientation

- A service is described by machine-executable metadata.
- The description supports the public nature of the SOA.
- Only those details that are exposed to the public and are important for the use of the service should be included in the description.

- The semantics of a service should be documented, either directly or indirectly, by its description.

SOA Realization (Two ways)

- XML - SOAP Based Web Services
- Extensible Markup Language (XML) is a markup language designed as a standard way to encode documents and data
- SOAP is an acronym for Simple Object Access Protocol. It is an XML-based messaging protocol for exchanging information among computers
- RESTful Web services

Web service is the terminology used everywhere

Service Oriented Architecture model implemented by XML Web Services

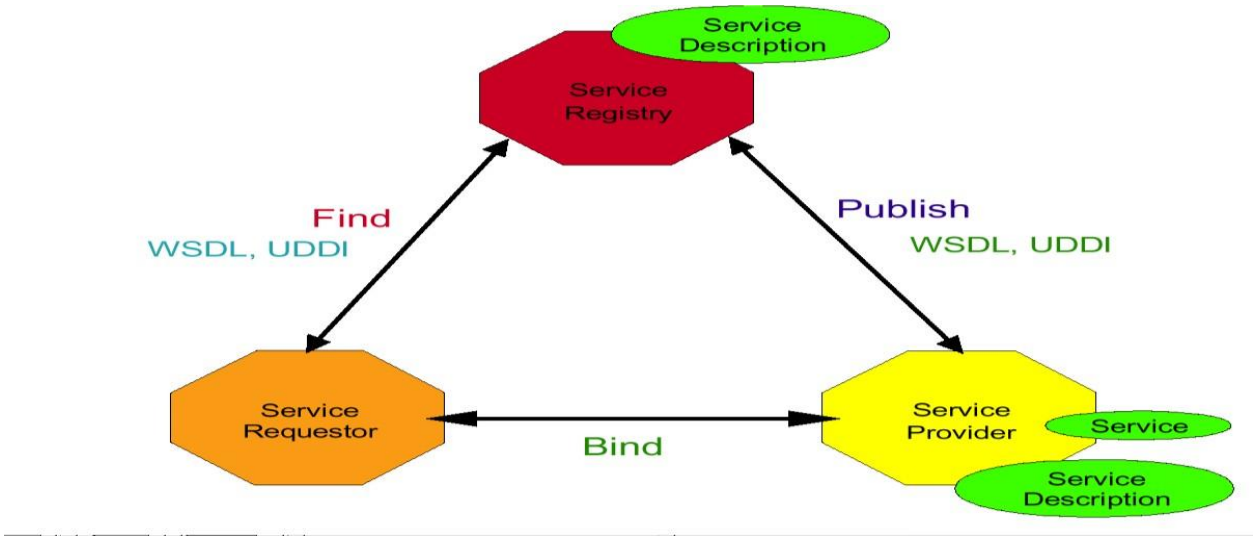


Figure 2.4 SOA Model

WSDL – Web Services Description Languages

- Provides a machine-readable description of how the service can be called, what parameters it expects, and what data structures it returns.
- Used in combination with SOAP and an XML Schema to provide Web services over the Internet.

UDDI – Universal Description, Discovery and Integration

- White Pages — address, contact, and known identifiers;

- Yellow Pages — industrial categorizations based on standard taxonomies;
- Green Pages — technical information about services exposed by the business.

Representational State Transfer (REST)

- APIs are for software components; a way for software to interact with other software.
- Web Services are a set of rules and technologies that enable two or more components on the web to talk to each other.
- Not every API is a web service.
- REST API is a web service.
- REST API is an API that follows the rules of REST specification.
- A web service is defined by rules:

a) How software components will talk?

b) What kind of messages they will send to each other?

c) How requests and responses will be handled?

HTTP and REST

- HTTP is an application layer protocol for sending and receiving messages over a network.
- REST is a specification that dictates how distributed systems on the web should communicate.
- REST is a way to implement and use the HTTP protocol.

REST and Systems of Systems

- SOA focuses on loosely coupled software applications running across different administrative domains, based on common protocols and technologies, such as HTTP and XML.
- SOA is related to early efforts on the architecture style of large scale distributed systems, particularly Representational State Transfer (REST).
- REST still provides an alternative to the complex standard-driven web services technology.
- Used in many Web 2.0 services.

- REST is a software architecture style for distributed systems, particularly distributed hypermedia systems, such as the World Wide Web.

Applications:

Google, Amazon, Yahoo, Facebook and Twitter

Advantage:

- Simplicity
- Ease of being published and consumed by clients.

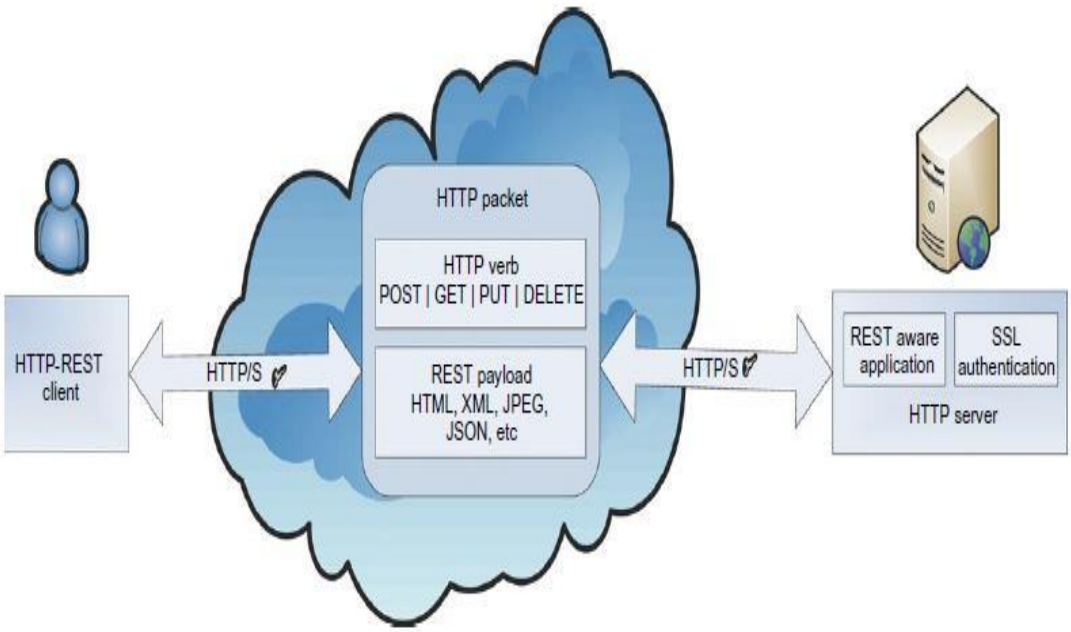


Figure 2.5 A simple REST interaction between user and server in HTTP specification

REST Principles

The REST architectural style is based on four principles:

- **Resource Identification through URIs**
- **Uniform, Constrained Interface**
- **Self-Descriptive Message**
- **Stateless Interactions**

Resource Identification through URIs

- The RESTful web service exposes a set of resources which identify targets of interaction with its clients.
- The key abstraction of information in REST is a resource.
- Any information that can be named can be a resource, such as a document or image or a temporal service.
- A resource is a conceptual mapping to a set of entities.
- Each particular resource is identified by a unique name, or more precisely, a Uniform Resource Identifier (URI).
- URI is of type URL, providing a global addressing space for resources involved in an interaction between components as well as facilitating service discovery.
- The URIs can be bookmarked or exchanged via hyperlinks.
- URIs provide more readability and the potential for advertisement.

Uniform, Constrained Interface

- Interaction with RESTful web services is done via the HTTP standard, client/server cacheable protocol.
- Resources are manipulated using a fixed set of four CRUD

  (create, read, update, delete) verbs or operations:

  - PUT
  - GET
  - POST
  - DELETE

- **PUT** creates a new resource.
- The resource can then be destroyed by using **DELETE.**
- **GET retrieves the current state of a resource.**
- **POST** transfers a new state onto a resource**.**

Self-Descriptive Message

- **A** REST message includes enough information to describe how to process the message.
- This enables intermediaries to do more with the message without parsing the message contents.
- In REST, resources are decoupled from their representation so that their content can be accessed in a variety of standard formats

  Eg:- HTML, XML, MIME, plain text, PDF, JPEG, JSON, etc.
- REST provides multiple/alternate representations of each resource.
- Metadata about the resource is available and can be used for various purposes.
  - Cache control
  - Transmission error detection
  - Authentication or authorization
  - Access control.

Stateless Interactions

- The REST interactions are "stateless"
- Message does not depend on the state of the conversation.
- Stateless communications improve visibility, reliability and increases scalability
- Decrease network performance by increasing the repetitive data

REST - Advantages

- RESTful web services can be considered an alternative to SOAP stack or "big web services
- Simplicity
- Lightweight nature
- Integration with HTTP

REST Architectural Elements

REST Elements	Elements	Example
Data elements	Resource	The intended conceptual target of a hypertext reference
	Resource identifier	URL
	Representation	HTML document, JPEG image, XML, etc.
	Representation metadata	Media type, last-modified time
	Resource metadata	Source link, alternates, vary
	Control data	If-modified-since, cache-control
Connectors	Client	libwww, libwww-perl
	Server	libwww, Apache API, NSAPI
	Cache	Browser cache, Akamai cache network
	Resolver	Bind (DNS lookup library)
	Tunnel	SSL after HTTP CONNECT
Components	Origin server	Apache httpd, Microsoft IIS
	Gateway	Squid, CGI, Reverse Proxy
	Proxy	CERN Proxy, Netscape Proxy, Gauntlet
	User agent	Netscape Navigator, Lynx, MOMspider

Web Services

- Web service is often referred to a self-contained, self-describing, modular application designed to be used and accessible by other software applications across the web.
- This allows client software to dynamically determine what a service does, the data types that a service uses, how to invoke operations on the service, and the responses that the service may return.
- Once a web service is deployed, other applications and other web services can discover and invoke the deployed service.
- A web service is defined as a software system designed to support interoperable machine-to-machine interaction over a network
- A web service has an interface described in a machine-executable format (specifically Web Services Description Language or WSDL).
- Web services are remotely executed, they do not depend on resources residing on the client system that calls them.

- Other systems interact with the web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization

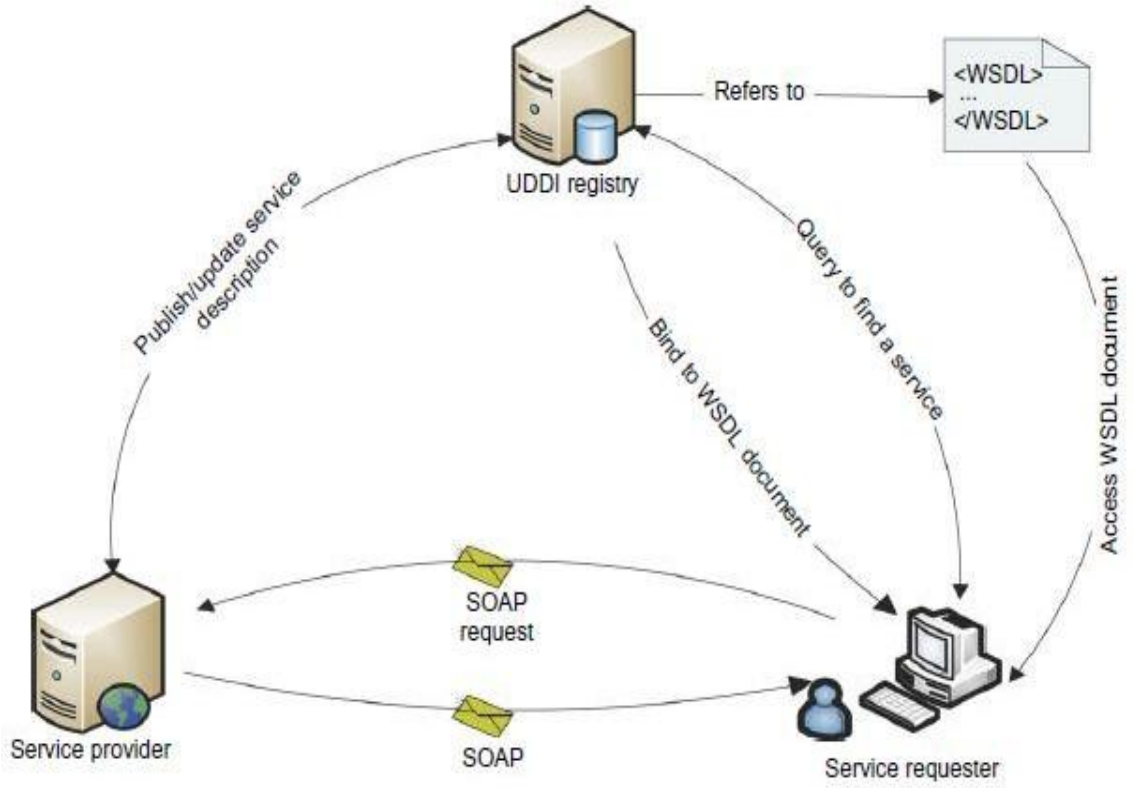


Figure 2.6 Web Services

Web Services- Technologies

- Simple Object Access Protocol (SOAP)
- Web Services Description Language (WSDL)
- Universal Description, Discovery and Integration (UDDI)

Simple Object Access Protocol (SOAP)

- SOAP provides a standard packaging structure for transmission of XML documents over various Internet protocols, such as SMTP, HTTP, and FTP.
- A SOAP message consists of a root element called envelope. Envelope contains a header: a container that can be extended by intermediaries.

- Additional application-level elements such as routing information, authentication, transaction management, message parsing instructions and Quality of Service (QoS) configurations are also included.
- Body element that carries the payload of the message.
- The content of the payload will be marshaled by the sender's SOAP engine and unmarshaled at the receiver side, based on the XML schema that describes the structure of the SOAP message.

Web Services Description Language (WSDL)

WSDL describes the interface, a set of operations supported by a web service in a standard format.

- It standardizes the representation of input and output parameters of its operations as well as the service's protocol binding, the way in which the messages will be transferred on the wire.
- Using WSDL enables disparate clients to automatically understand how to interact with a web service.

Universal Description, Discovery, and Integration (UDDI)

- UDDI provides a global registry for advertising and discovery of web services.
- Performed by searching for names, identifiers, categories or the specification implemented by the web service

SOAP is an extension, and an evolved version of XML-RPC.

A simple and effective remote procedure call protocol which uses XML for encoding its calls and HTTP as a transport mechanism.

- A procedure executed on the server and the value it returns was formatted in XML.

- SOAP mainly describes the protocols between interacting parties
  - Data format of exchanging messages is left to XML schema to be handled.

Figure 2.7 WS-I Protocol Stack

- **Business Process Execution Language for Web Services (BPEL4WS**), a standard executable language for specifying interactions between web services.
  - Web services can be composed together to make more complex web services and workflows.
  - BPEL4WS is an XML-based language, built on top of web service specifications, which is used to define and manage long-lived service orchestrations or processes.
  - In BPEL, a business process is a large-grained stateful service, which executes steps to complete a business goal.
  - That goal can be the completion of a business transaction, or fulfillment of the job of a service.
- **Web Service WS-Notification** enables web services to use the publish and subscribe messaging pattern.
- **Web Services Security (WS-Security) are** set of protocols that ensure security for SOAP-based messages by implementing the principles of confidentiality, integrity and authentication.

- **Web Services Reliable Messaging** (**WS-Reliable Messaging**) describes a protocol that allows **messages** to be delivered reliably between distributed applications in the presence of software component, system, or network failures
- **WS-ResourceLifetime** specification standardizes the means by which a WS-Resource can be destroyed.
- **WS-Policy** is a specification that allows web services to use XML to advertise their **policies** (on security, quality of service, etc.) and for web service consumers to specify their **policy** requirements.
- **WS-ResourceProperties** defines a standard set of message exchanges that allow a requestor to query or update the property values of the WS-Resource.
- **WS-Addressing** is a specification of transport-neutral mechanism that allows web services to communicate addressing information.
- **WS-Transaction WS-Transaction** is a specification developed that indicates how transactions will be handled and controlled in Web Services.
- The transaction specification is divided into two parts - short atomic transactions (AT) and long business activity (BA).
- **Web Services Coordination (WS-Coordination)** describes an extensible framework for providing protocols that coordinate the actions of distributed applications.
- **The Java Message Service (JMS) API** is a messaging standard that allows application components based on the Java Platform Enterprise Edition (Java EE) to create, send, receive, and read messages.
- **Internet Inter-ORB Protocol (IIOP)** is an object-oriented protocol
- Used to facilitate network interaction between distributed programs written in different programming languages.
- IIOP is used to enhance Internet and intranet communication for applications and services.

Table 5.3 Sample SOAP Request-Response for Creating an S3 Bucket

SOAP Request	SOAP Response
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" soap:encodingStyle= "http://www.w3.org/2001/12/soap-encoding"> <soap:Body> <CreateBucket xmlns="http://doc.s3.amazonaws .com/2010-03-15"> <Bucket>SampleBucket</Bucket> <AWSAccessKeyId> 1B9FVRAYCP1VJEXAMPLE= </AWSAccessKeyId> <Timestamp>2010-03-15T14:40:00.165Z </Timestamp> <Signature>luyz3d3P0aTou39dzbqaEXAMPLE =</Signature> </CreateBucket> </soap:Body> </soap:Envelope>	<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" soap:encodingStyle= "http://www.w3.org/2001/12/soap-encoding"> <soap:Body> <CreateBucket xmlns="http://doc.s3.amazonaws .com/2010-03-15"> <Bucket>SampleBucket</Bucket> <AWSAccessKeyId>1B9FVRAYCP1VJEXAMPLE= </AWSAccessKeyId> <Timestamp>2010-03-15T14:40:00.165Z </Timestamp> <Signature>luyz3d3P0aTou39dzbqaEXAMPLE =</Signature> </CreateBucket> </soap:Body> </soap:Envelope>

- A SOAP message consists of an envelope used by the applications to enclose information that need to be sent.
- An envelope contains a header and a body block.
- The EncodingStyle element refers to the URI address of an XML schema for encoding elements of the message.
- Each element of a SOAP message may have a different encoding, but unless specified, the encoding of the whole message is as defined in the XML schema of the root element.
- The header is an optional part of a SOAP message that may contain auxiliary information.
- The body of a SOAP request-response message contains the main information of the conversation, formatted in one or more XML blocks.
- In example, the client is calling CreateBucket of the Amazon S3 web service interface.
- In case of an error in service invocation, a SOAP message including a Fault element in the body.

Table 5.4 The 10 Areas Covered by the Core WS-* Specifications

WS-* Specification Area	Examples
1. Core Service Model	XML, WSDL, SOAP
2. Service Internet	WS-Addressing, WS-MessageDelivery, Reliable WSRM, Efficient MOTM
3. Notification	WS-Notification, WS-Eventing (Publish-Subscribe)
4. Workflow and Transactions	BPEL, WS-Choreography, WS-Coordination
5. Security	WS-Security, WS-Trust, WS-Federation, SAML, WS-SecureConversation
6. Service Discovery	UDDI, WS-Discovery
7. System Metadata and State	WSRF, WS-MetadataExchange, WS-Context
8. Management	WSDM, WS-Management, WS-Transfer
9. Policy and Agreements	WS-Policy, WS-Agreement
10. Portals and User Interfaces	WSRP (Remote Portlets)

PUBLISH SUBSCRIBE MODEL

- Publish/Subscribe systems are nowadays considered a key technology for information diffusion.
- Each participant in a publish/subscribe communication system can play the role of a publisher or a subscriber of information.
- Publishers produce information in form of events, which are then consumed by subscribers.
- Subscribers can declare their interest on a subset of the whole information issuing subscriptions.
- There are two major roles:
  - Publisher
  - Subscriber
- The former provides facilities for the later to register its interest in a specific topic or event.

- Specific conditions holding true on the publisher side can trigger the creation of messages that are attached to a specific event.
- Message will be available to all the subscribers that registered for the corresponding event.
- There are two major strategies for dispatching the event to the subscribers.

Push strategy:

- It is the responsibility of the publisher to notify all the subscribers. Eg: Method invocation.

Pull strategy :

- The publisher simply makes available the message for a specific event.
- It is the responsibility of the subscribers to check whether there are messages on the events that are registered.
- Subscriptions are used to filter out part of the events produced by publishers.
- In Software Architecture, Publish/Subscribe pattern is a message pattern and a network oriented architectural pattern
- It describes how two different parts of a message passing system connect and communicate with each other.
- There are three main components to the Publish Subscribe Model:
    - Publishers
    - Eventbus/broker
    - Subscribers

Publishers:

- Broadcast messages, with no knowledge of the subscribers.

Subscribers:

- They ‘listen’ out for messages regarding topic/categories that they are interested in without any knowledge of who the publishers are.

Event Bus:

- Transfers the messages from the publishers to the subscribers.

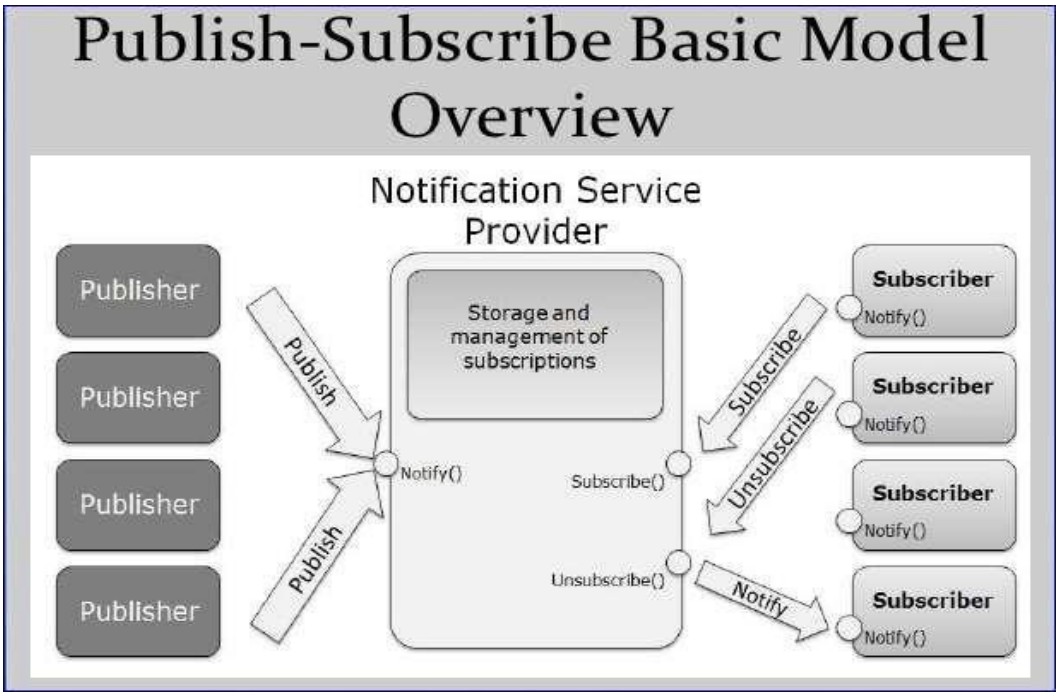


Figure 2.8 Publish Subscribe Model

- Each subscriber only receives a subset of the messages that have been sent by the Publisher.
- Receive the message topics or categories they have subscribed to.
- There are two methods of filtering out unrequired messages:
  - Topic based filter
  - Content based filter
  -

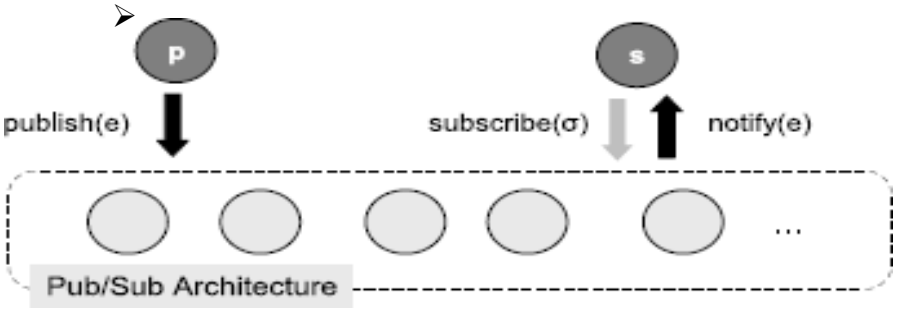


Figure 2.9 High Level View of A Publish/Subscribe System

- A generic pub/sub communication system is often referred as Event Service or Notification Service.

- System composed of a set of nodes distributed over a communication network.
- The clients of this system are divided according to their role into publishers and subscribers.
- Clients are not required to communicate directly among themselves.
- The interaction takes place through the nodes of the pub/sub system.

Elements of a Publish/Subscribe System

- A publisher submits a piece of information e (i.e., an event) to the pub/sub system by executing the publish(e) operation.
- An event is structured as a set of attribute-value pairs.
- Each attribute has a *name, a simple character* string, and a *type.*
- The type is generally one of the common primitive data types defined in programming languages or query languages (e.g. integer, real, string, etc.).
- On the subscriber's side, interest in specific events is expressed through subscriptions.
- *A* subscription is a filter over a portion of the event content (or the whole of it).
- Expressed through a set of constraints that depend on the subscription language.
- A subscriber installs and removes a subscription from the pub/sub system by executing the subscribe() and unsubscribe() operations respectively.
- An event e matches a subscription if it satisfies all the declared constraints on the corresponding attributes.
- The task of verifying whenever an event e matches a subscription is called matching.

Semantics of a Publish/subscribe System

- When a process issues a subscribe/unsubscribe operation, the pub/sub system is not immediately aware of the occurred event.
- The registration (resp. cancellation) of a subscription takes a certain amount of time, denoted as Tsub, to be stored into the system.

This time encompass the update of the internal data structures of the pub/sub system and the network delay due to the routing of the subscription.

Three properties:

- Safety (Legality): A subscriber cannot be notified for an information it is not interested in.
- Safety (Validity): A subscriber cannot be notified for an event that has not been previously published.
- Liveness: The delivery of a notification for an event is guaranteed only for those subscribers that subscribed at a time at least Tsub before the event was published.

Quality of Service in Publish/Subscribe Systems

- Reliable delivery
- Timeliness
- Security and trust

Reliable delivery

- Reliable delivery of an event means determining the subscribers that have to receive a published event, as stated by the liveness property and delivering the event to all of them.

Timeliness

- Real-time applications often require strict control over the time elapsed by a piece of information to reach all its consumers.
- They are typically deployed over dedicated infrastructures or simply managed environments where synchronous message delivery can be safely assumed.

Security and trust

- A subscriber wants to trust authenticity of the events it receives from the system.
- Generated by a trusty publisher and the information they contains have not been corrupted.
- Subscribers have to be trusted for what concerns the subscriptions they issue.
- Since an event is in general delivered to several subscribers, the producer/consumer trust relationship that commonly occur in a point-to-point communication, in pub/sub system must involve multiple participants

Subscription Models

- Topic based Model
- Type based Model
- Concept based Model
- Content based Model

Topic-based Model

- Events are grouped in topics.
- A subscriber declares its interest for a particular topic to receive all events pertaining to that topic.
- Each topic corresponds to a logical channel ideally connecting each possible publisher to all interested subscribers.
- Requires the messages to be broadcasted into logical channels.
- Subscribers only receive messages from logic channels they care about (and have subscribed to).

Type based Model

- Pub/sub variant events are actually objects belonging to a specific type, which can thus encapsulate attributes as well as methods.
- Types represent a more robust data model for application developer.
- Enforce type-safety at the pub/sub system, rather than inside the application.
- The declaration of a desired type is the main discriminating attribute.

Concept based Model

- Allows to describe event schema at a higher level of abstraction by using ontologies.
- Provide a knowledge base for an unambiguous interpretation of the event structure, by using metadata and mapping functions.

Content based Model

- System allows subscribers to receive messages based on the content of the messages.

Subscribers themselves must sort out junk messages from the ones they want.

Benefits

Loose coupling

- The publisher is not aware of the number of subscribers, of the identities of the subscribers, or of the message types that the subscribers are subscribed to.

Improved security

- The communication infrastructure transports the published messages only to the applications that are subscribed to the corresponding topic.
- Specific applications can exchange messages directly, excluding other applications from the message exchange.

Improved testability.

- Topics usually reduce the number of messages that are required for testing.

Separation of concerns

- Due to the simplistic nature of the architecture, developers can exercise fine grained separation of concerns by dividing up message types to serve a single simple purpose each.
- Eg. data with a topic "/cats" should only contain information about cats.

Reduced cognitive load for subscribers

- Subscribers need not concern themselves with the inner workings of a publisher.
- They do not even have to access to the source code.
- Subscribers only interact with the publisher through the public API exposed by the publisher.

Drawbacks

Increased complexity.

Publish/Subscribe requires you to address the following:

- To design a message classification scheme for topic implementation.
- To implement the subscription mechanism.
- To modify the publisher and the subscribers.

Increased maintenance effort.

- Managing topics requires maintenance work.
- Organizations that maintain many topics usually have formal procedures for their use.

Decreased performance

- Subscription management adds overhead.
- This overhead increases the latency of message exchange, and this latency decreases performance.

Inflexibility of data sent by publisher

- The publish/subscribe model introduces high semantic coupling in the messages passed by the publishers to the subscribers.
- Once the structure of the data is established, it becomes difficult to change.
- In order to change the structure of the messages, all of the subscribers must be altered to accept the changed format

Instability of Delivery

- The publisher does not have perfect knowledge of the status of the systems listening to the messages.
- For instance, publish/subscribe is commonly used for logging systems.
- If a logger subscribing to the 'Critical' message type crashes or gets stuck in an error state, then the 'Critical' messages may be lost!
- Then any services depending on the error messages will be unaware of the problems with the publisher.

Applications

Used in a wide range of group communication applications including

- Software Distribution
- Internet TV
- Audio or Video-conferencing
- Virtual Classroom

- Multi-party Network Games
- Distributed Cache Update

It can also be used in even larger size group communication applications, such as broadcasting and content distribution.

- News and Sports Ticker Services
- Real-time Stock Quotes and Updates
- Market Tracker
- Popular Internet Radio Sites

VIRTUALIZATION

- Virtualization is a technique, which allows sharing single physical instance of an application or resource among multiple organizations or tenants (customers).
- Virtualization is a proved technology that makes it possible to run multiple operating system and applications on the same server at same time.
- Virtualization is the process of creating a logical(virtual) version of a server operating system, a storage device, or network services.
- The technology that work behind virtualization is known as a virtual machine monitor(VM), or virtual manager which separates compute environments from the actual physical infrastructure.
- Virtualization -- the abstraction of computer resources.
- Virtualization hides the physical characteristics of computing resources from their users, applications, or end users.
- This includes making a single physical resource (such as a server, an operating system, an application, or storage device) appear to function as multiple virtual resources.
- It can also include making multiple physical resources (such as storage devices or servers) appear as a single virtual resource.
- In computing, virtualization refers to the act of creating a virtual (rather than actual) version of something, like computer hardware platforms, operating systems, storage devices, and computer network resources

- Creation of a virtual machine over existing operating system and hardware.
- **Host machine:** The machine on which the virtual machine is created.
- **Guest machine:** virtual machines referred as a guest machine.
- **Hypervisor:** Hypervisor is a firmware or low-level program that acts as a Virtual Machine Manager.

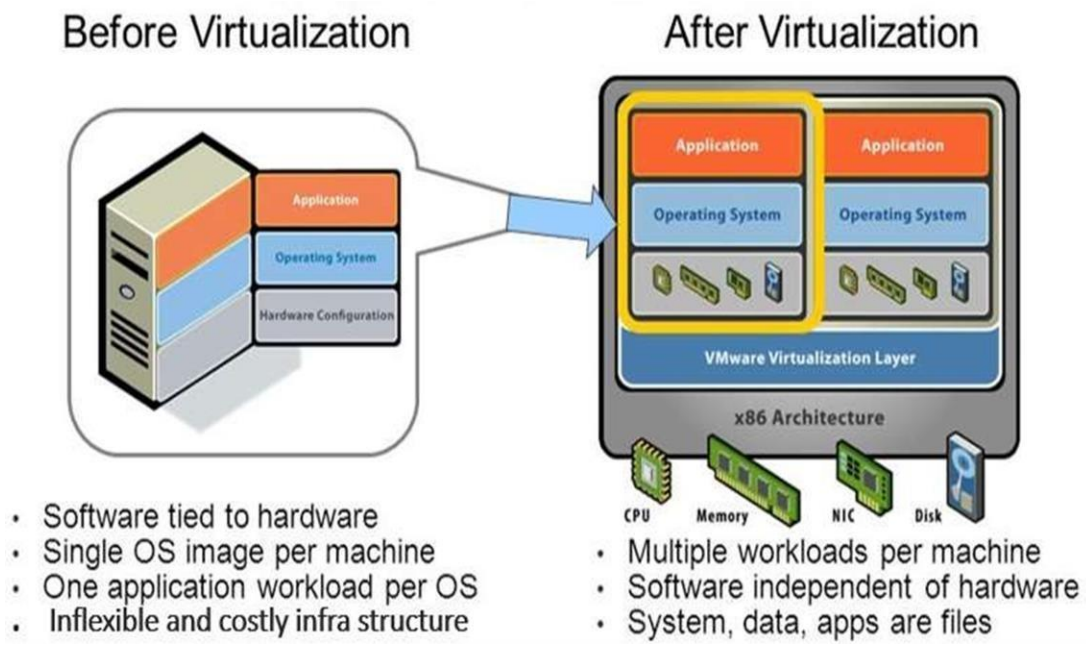


Figure 2.10 Virtualization Example

Advantages of Virtualization:

1. Reduced Costs.
2. Efficient hardware Utilization.
3. Virtualization leads to better resource Utilization and increase performance
4. Testing for software development.
5. Increase Availability
6. Save energy
7. Shifting all your Local Infrastructure to Cloud in a day
8. Possibility to Divide Services
9. Running application not supported by the host.

Disadvantages of Virtualization:

1. Extra Costs.
2. Software Licensing.

IMPLEMENTATION LEVELS OF VIRTUALIZATION

Virtualization is a computer architecture technology by which multiple virtual machines (VMs) are multiplexed in the same hardware machine. The purpose of a VM is to enhance resource sharing by many users and improve computer performance in terms of resource utilization and application flexibility.

Hardware resources (CPU, memory, I/O devices, etc.) or software resources(operating system and software libraries) can be virtualized in various functional layers.

The idea is to separate the hardware from the software to yield better system efficiency. For example, computer users gained access to much enlarged memory space when the concept of virtual memory was introduced. Similarly, virtualization techniques can be applied to enhance the use of compute engines, networks and storage.

Levels of Virtualization:

A traditional computer runs with host operating system specially tailored for its hardware architecture, as shown in Figure 2.11 (a). After virtualization, different user applications managed by their own operating systems (guest OS) can run on the same hardware, independent of the host OS.

This is often done by adding additional software, called a virtualization layer as shown in Figure 2.11 (b). This virtualization layer is known as hypervisor or virtual machine monitor (VMM) .The VMs are shown in the upper boxes, where applications run with their own guest OS over the virtualized CPU, memory, and I/O resources. The main function of the software layer for virtualization is to virtualize the physical hardware of a host machine into virtual resources to be used by the VMs, exclusively. The virtualization software creates the abstraction of VMs by interposing a virtualization layer at various levels of a computer system. Common virtualization layers include the instruction set architecture (ISA) level, hardware level, operating system level, library support level, and application level.

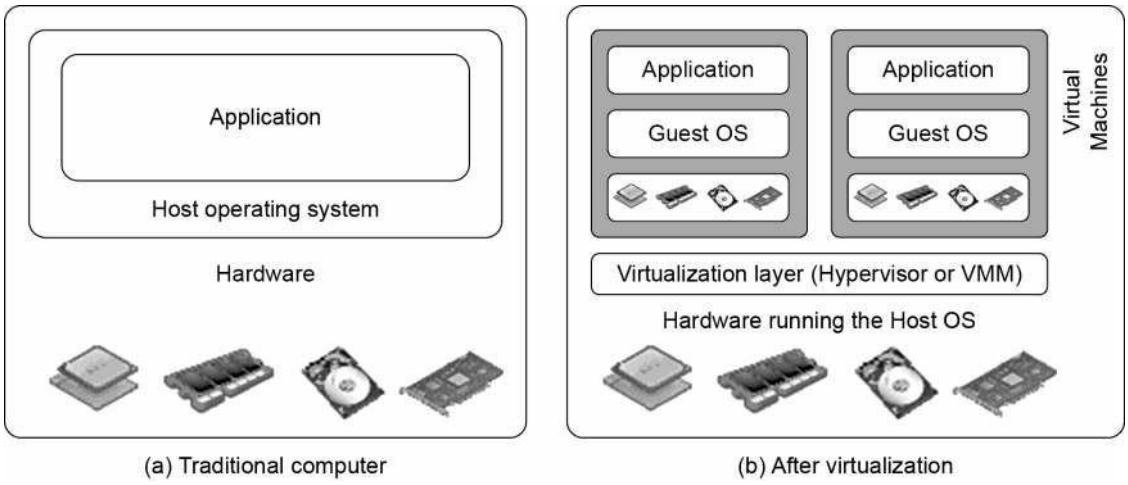


Figure 2.11 The architecture of a computer system before and after Virtualization

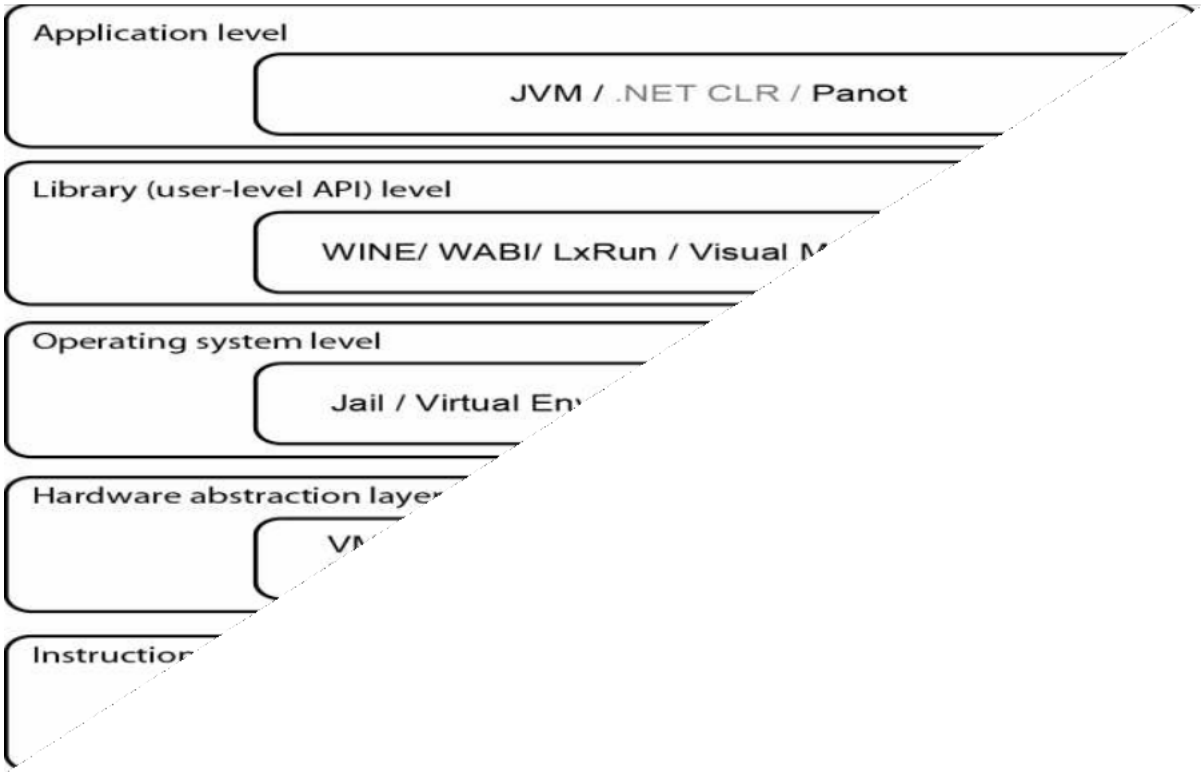


Figure 2.12 Virtualization ranging from hardware to applications in five abstraction levels.

Instruction Set Architecture Level:

At the ISA level, virtualization is performed by emulating a given ISA by the ISA of the host machine. For example, MIPS binary code can run on an x86-based host machine with the help of ISA emulation. With this approach, it is possible to run a large amount of legacy binary

code written for various processors on any given new hardware host machine. Instruction set emulation leads to virtual ISAs created on any hardware machine.

The basic emulation method is through code interpretation. An interpreter program interprets the source instructions to target instructions one by one. OneSource instruction may require tens or hundreds of native target instructions to perform its function. Obviously, this process is relatively slow. For better performance, dynamic binary translation is desired.

This approach translates basic blocks of dynamic source instructions to target instructions. The basic blocks can also be extended to program traces or super blocks to increase translation efficiency. Instruction set emulation requires binary translation and optimization. A virtual instruction set architecture (V-ISA) thus requires adding a processor-specific software translation layer to the compiler.

Hardware Abstraction Level:

Hardware-level virtualization is performed right on top of the bare hardware. The idea is to virtualize a computer's resources, such as its processors, memory, and I/O devices. The intention is to upgrade the hardware utilization rate by multiple users concurrently.

Operating System Level:

This refers to an abstraction layer between traditional OS and user applications. OS-level virtualization creates isolated containers on a single physical server and the OS instances to utilize the hardware and software in datacenters.

The containers behave like real servers. OS-level virtualization is commonly used in creating virtual hosting environments to allocate hardware resources among a large number of mutually distrusting users. It is also used, to a lesser extent, in consolidating server hardware by moving services on separate hosts into containers or VMs on one server.

Library Support Level:

Most applications use APIs exported by user level libraries rather than using lengthy system calls by the OS. Since most systems provide well documented APIs, such an interface becomes another candidate for virtualization.

Virtualization with library interfaces is possible by controlling the communication link between applications and the rest of a system through API hooks. The software tool WINE has implemented this approach to support Windows applications on top of UNIX hosts. Another

example is the vCUDA which allows applications executing within VMs to leverage GPU hardware acceleration.

User-Application Level:

Virtualization at the application level virtualizes an application as a VM. On a traditional OS, an application often runs as a process. Therefore, application-level virtualization is also known as process-level virtualization. The most popular approach is to deploy high level language (HLL)VMs.

VMM Design Requirements and Providers

Hardware-level virtualization inserts a layer between real hardware and traditional operating systems. This layer is commonly called the Virtual Machine Monitor (VMM) and it manages the hardware resources of a computing system. Each time programs access the hardware the VMM captures the process. VMM acts as a traditional OS.

One hardware component, such as the CPU, can be virtualized as several virtual copies. Therefore, several traditional operating systems which are the same or different can sit on the same set of hardware simultaneously.

Three requirements for a VMM

- First, a VMM should provide an environment for programs which is essentially identical to the original machine.
- Second, programs run in this environment should show, at worst, only minor decreases in speed.
- Third, a VMM should be in complete control of the system resources

Table 3.2 Comparison of Four VMM and Hypervisor Software Packages

Provider and References	Host CPU	Host OS	Guest OS	Architecture
VMware Workstation [71]	x86, x86-64	Windows, Linux	Windows, Linux, Solaris, FreeBSD, Netware, OS/2, SCO, BeOS, Darwin	Full Virtualization
VMware ESX Server [71]	x86, x86-64	No host OS	The same as VMware Workstation	Para-Virtualization
Xen [7,13,42]	x86, x86-64, IA-64	NetBSD, Linux, Solaris	FreeBSD, NetBSD, Linux, Solaris, Windows XP and 2003 Server	Hypervisor
KVM [31]	x86, x86-64, IA-64, S390, PowerPC	Linux	Linux, Windows, FreeBSD, Solaris	Para-Virtualization

Virtualization Support at the OS Level

With the help of VM technology, a new computing mode known as cloud computing is emerging. Cloud computing is transforming the computing landscape by shifting the hardware and staffing costs of managing a computational center to third parties, just like banks. However, cloud computing has at least two challenges.

- The first is the ability to use a variable number of physical machines and VM instances depending on the needs of a problem.
- The second challenge concerns the slow operation of instantiating new VMs.

Currently, new VMs originate either as fresh boots or as replicates of a template VM, unaware of the current application state. Therefore, to better support cloud computing, a large amount of research and development should be done.

Why OS-Level Virtualization?

To reduce the performance overhead of hardware-level virtualization, even hardware modification is needed. OS-level virtualization provides a feasible solution for these hardware-level virtualization issues. Operating system virtualization inserts a virtualization layer inside an operating system to partition a machine's physical resources. It enables multiple isolated VMs within a single operating system kernel. This kind of VM is often called a virtual execution environment (VE), Virtual Private System (VPS), or simply container. From the user's point of view, VEs look like real servers. This means a VE has its own set of processes, file system, user accounts, network interfaces with IP addresses, routing tables, firewall rules, and other personal settings. Although VEs can be customized for different people, they share the same operating system kernel.

Advantages of OS Extensions

(1) VMs at the operating system level have minimal startup/shutdown costs, low resource requirements, and high scalability.

(2) For an OS-level VM, it is possible for a VM and its host environment to synchronize state changes when necessary.

These benefits can be achieved via two mechanisms of OS-level virtualization:

(1) All OS-level VMs on the same physical machine share a single operating system kernel

(2) The virtualization layer can be designed in a way that allows processes in VMs to access as many resources of the host machine as possible, but never to modify them.

Virtualization on Linux or Windows Platforms

Virtualization support on the Windows-based platform is still in the research stage. The Linux kernel offers an abstraction layer to allow software processes to work with and operate on resources without knowing the hardware details. New hardware may need a new Linux kernel to support. Therefore, different Linux platforms use patched kernels to provide special support for extended functionality.

Table 3.3 Virtualization Support for Linux and Windows NT Platforms

Virtualization Support and Source of Information	Brief Introduction on Functionality and Application Platforms
Linux vServer for Linux platforms (http://linux-vserver.org/)	Extends Linux kernels to implement a security mechanism to help build VMs by setting resource limits and file attributes and changing the root environment for VM isolation
OpenVZ for Linux platforms [65]; http://ftp.openvz.org/doc/OpenVZ-Users-Guide.pdf)	Supports virtualization by creating *virtual private servers (VPSes)*; the VPS has its own files, users, process tree, and virtual devices, which can be isolated from other VPSes, and checkpointing and live migration are supported
FVM (Feather-Weight Virtual Machines) for virtualizing the Windows NT platforms [78])	Uses system call interfaces to create VMs at the NY kernel space; multiple VMs are supported by virtualized namespace and copy-on-write

Middleware Support for Virtualization

Library-level virtualization is also known as user-level Application Binary Interface (ABI) or API emulation. This type of virtualization can create execution environments for running alien programs on a platform rather than creating a VM to run the entire operating system. API call interception and remapping are the key functions performed. This provides an overview of several library-level virtualization systems: namely the Windows Application Binary Interface (WABI), lxrun, WINE, Visual MainWin, and Vcuda.

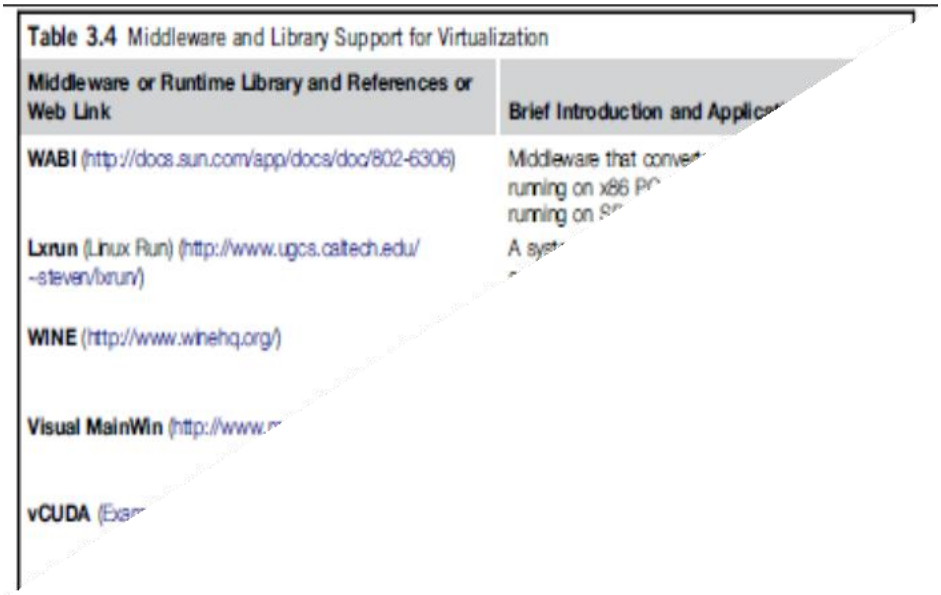


Table 3.4 Middleware and Library Support for Virtualization

Middleware or Runtime Library and References or Web Link	Brief Introduction and Applic
WABI (http://docs.sun.com/app/docs/doc/802-6306)	Middleware that conver running on x86 P running on S
Lxrun (Linux Run) (http://www.ugcs.caltech.edu/~steven/lxrun/)	A syst
WINE (http://www.winehq.org/)	
Visual MainWin (http://www.	
vCUDA (Exa	

Virtualization Structures/Tools and Mechanisms

There are three typical classes of VM architecture. Before virtualization, the operating system manages the hardware. After virtualization, a virtualization layer is inserted between the hardware and the operating system. In such a case, the virtualization layer is responsible for converting portions of the real hardware into virtual hardware. Therefore, different operating systems such as Linux and Windows can run on the same physical machine, simultaneously.

Depending on the position of the virtualization layer, there are several classes of VM architectures, namely the hypervisor architecture, para-virtualization, and host based virtualization. The hypervisor is also known as the VMM (Virtual Machine Monitor). They both perform the same virtualization operations.

Hypervisor and Xen Architecture:

The hypervisor supports hardware-level virtualization on bare metal devices like CPU, memory, disk and network interfaces. The hypervisor software sits directly between the physical hardware and its OS. This virtualization layer is referred to as either the VMM or the hypervisor. The hypervisor provides hypercalls for the guest OSes and applications. Depending on the functionality, a hypervisor can assume a micro-kernel architecture like the Microsoft Hyper-V. Or it can assume monolithic hypervisor architecture like the VMware ESX for server virtualization.

A micro-kernel hypervisor includes only the basic and unchanging functions (such as physical memory management and processor scheduling). The device drivers and other changeable components are outside the hypervisor. A monolithic hypervisor implements all the aforementioned functions, including those of the device drivers.

Therefore, the size of the hypervisor code of a micro-kernel hypervisor is smaller than that of a monolithic hypervisor. Essentially, a hypervisor must be able to convert physical devices into virtual resources dedicated for the deployed VM to use.

The Xen Architecture:

The core components of a Xen system are the hypervisor, kernel, and applications. The organization of the three components is important. Like other virtualization systems, many guest OSes can run on top of the hypervisor. However, not all guest OSes are created equal, and one in particular controls the others.

The guest OS, which has control ability, is called Domain 0, and the others are called Domain U. Domain 0 is a privileged guest OS of Xen. It is first loaded when Xen boots without any file system drivers being available. Domain 0 is designed to access hardware directly and manage devices. Therefore, one of the responsibilities of Domain 0 is to allocate and map hardware resources for the guest domains (the Domain U domains).

Binary Translation with Full Virtualization:

Depending on implementation technologies, hardware virtualization can be classified into two categories: full virtualization and host-based virtualization. Full virtualization does not need to modify the host OS. It relies on binary translation to trap and to virtualize the execution of certain sensitive, non virtualizable instructions. The guest OSes and their applications consist of noncritical and critical instructions. In a host-based system, both a host OS and a guest OS are used. A virtualization software layer is built between the host OS and guest OS.

Full Virtualization:

With full virtualization, noncritical instructions run on the hardware directly while critical instructions are discovered and replaced with traps into the VMM to be emulated by software. Both the hypervisor and VMM approaches are considered full virtualization.

Binary Translation of Guest OS Requests Using a VMM :

VMware puts the VMM at Ring 0 and the guest OS at Ring 1. The VMM scans the instruction stream and identifies the privileged, control- and behavior-sensitive instructions. When these instructions are identified, they are trapped into the VMM, which emulates the behavior of these instructions.

Figure 2.13 Indirect execution of complex instructions via binary translation of guest OS requests using the VMM plus direct execution of simpleinstructions on the same host.

The method used in this emulation is called binary translation. Therefore, full virtualization combines binary translation and direct execution. The guest OS is completely decoupled from the underlying hardware. Consequently, the guest OS is unaware that it is being virtualized. Binary translation employs a code cache to store translated hot instructions to improve performance, but it increases the cost of memory usage.

Host-Based Virtualization:

An alternative VM architecture is to install a virtualization layer on top of the host OS. This host OS is still responsible for managing the hardware. The guest OSes are installed and run on top of the virtualization layer. Dedicated applications may run on the VMs. Certainly, some other applications can also run with the host OS directly. This host-based architecture has some distinct advantages, as enumerated next. First, the user can install this VM architecture without modifying the host OS. The virtualizing software can rely on the host OS to provide device drivers and other low level services. This will simplify the VM design and ease its deployment. Second, the host-based approach appeals to many host machine configurations.

Compared to the hypervisor/VMM architecture, the performance of the host based architecture may also be low. When an application requests hardware access, it involves four layers of mapping which downgrades performance significantly.

Para-Virtualization with Compiler Support:

Para-virtualization needs to modify the guest operating systems. A para-virtualized VM provides special APIs requiring substantial OS modifications in user applications. Performance degradation is a critical issue of a virtualized system. No one wants to use a VM if it is much slower than using a physical machine.

The virtualization layer can be inserted at different positions in a machine software stack. However, para-virtualization attempts to reduce the virtualization overhead, and thus improve performance by modifying only the guest OS kernel. The guest operating systems are para-virtualized. The traditional x86 processor offers four instruction execution rings: Rings 0,1, 2, and 3. The lower the ring number, the higher the privilege of instruction being executed. The OS is responsible for managing the hardware and the privileged instructions to execute at Ring 0, while user-level applications run at Ring 3.

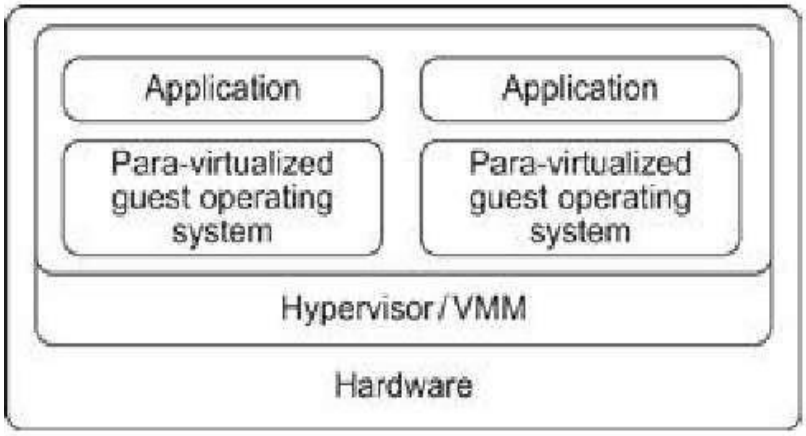


Figure 2.14 Para-virtualized VM architecture

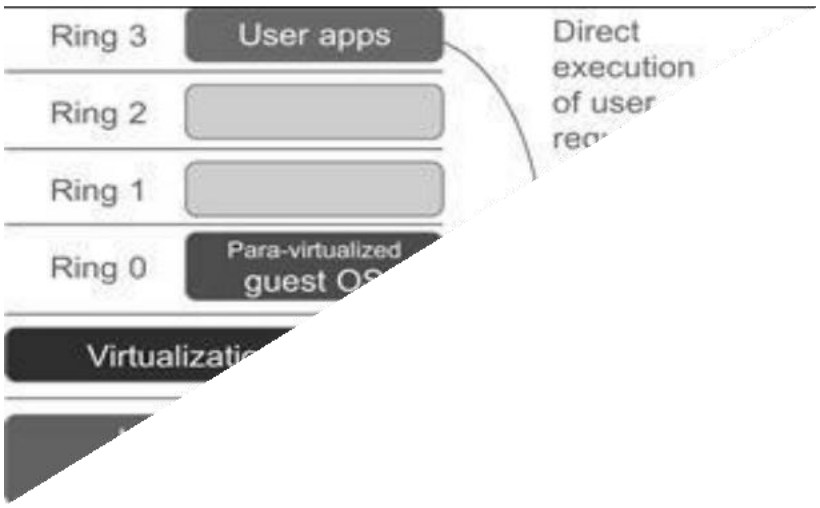


Figure 2.15 The use of a para-virtualized guest OS assisted by an intelligent compiler to replace non virtualizable OS instructions by hyper calls.

Para-Virtualization Architecture:

When the x86 processor is virtualized, a virtualization layer is inserted between the hardware and the OS. According to the x86 ring definitions, the virtualization layer should also be installed at Ring 0. The para-virtualization replaces non virtualizable instructions with hyper calls that communicate directly with the hypervisor or VMM. However, when the guest OS kernel is modified for virtualization, it can no longer run on the hardware directly.

Although para-virtualization reduces the overhead, it has incurred other problems. First, its compatibility and portability may be in doubt, because it must support the unmodified OS as well. Second, the cost of maintaining para-virtualized OSes is high, because they may require deep OS kernel modifications. Finally, the performance advantage of para virtualization varies greatly due to workload variations.

KVM (Kernel-Based VM):

This is a Linux para-virtualization system—a part of the Linux version 2.6.20 kernel. Memory management and scheduling activities are carried out by the existing Linux kernel. The KVM does the rest, which makes it simpler than the hypervisor that controls the entire machine. KVM is a hardware-assisted para-virtualization tool, which improves performance and supports unmodified guest OSes such as Windows, Linux, Solaris, and other UNIX variants. Unlike the

full virtualization architecture which intercepts and emulates privileged and sensitive instructions at runtime, para-virtualization handles these instructions at compile time.

The guest OS kernel is modified to replace the privileged and sensitive instructions with hyper calls to the hypervisor or VMM. Xen assumes such a para virtualization architecture. The guest OS running in a guest domain may run at Ring 1instead of at Ring 0. This implies that the guest OS may not be able to execute some privileged and sensitive instructions. The privileged instructions are implemented by hypercalls to the hypervisor. After replacing the instructions with hyper calls, the modified guest OS emulates the behavior of the original guest OS.

VIRTUALIZATION OF CPU, MEMORY, AND I/O DEVICES

To support virtualization, processors such as the x86 employ a special running mode and instructions, known as hardware-assisted virtualization. In this way, the VMM and guest OS run in different modes and all sensitive instructions of the guest OS and its applications are trapped in the VMM. To save processor states, modes witching are completed by hardware. For the x86architecture, Intel and AMD have proprietary technologies for hardware-assisted virtualization.

Hardware Support for Virtualization: Modern operating systems and processors permit multiple processes to run simultaneously. If there is no protection mechanism in a processor, all instructions from different processes will access the hardware directly and cause a system crash. Therefore, all processors have at least two modes, user mode and supervisor mode, to ensure controlled access of critical hardware. Instructions running in supervisor mode are called privileged instructions.

Other instructions are unprivileged instructions. In a virtualized environment, it is more difficult to make OSes and applications run correctly because there are more layers in the machine stack.

CPU Virtualization:

A VM is a duplicate of an existing computer system in which a majority of the VM instructions are executed on the host processor in native mode. Thus, unprivileged instructions of VMs run directly on the host machine for higher efficiency. Other critical instructions should be handled carefully for correctness and stability. The critical instructions are divided into three categories:

Privileged instructions - Privileged instructions execute in a privileged mode and will be trapped if executed outside this mode.

Control sensitive instructions - Control-sensitive instructions attempt to change the configuration of resources used.

Behavior-sensitive instructions - Behavior-sensitive instructions have different behaviors depending on the configuration of resources, including the load and store operations over the virtual memory.

A CPU architecture is virtualizable if it supports the ability to run the VM's privileged and privileged instructions in the CPU's user mode while the VMM runs in supervisor mode. When the privileged instructions including control- and behavior sensitive instructions of a VM are executed, they are trapped in the VMM. In this case, the VMM acts as a unified mediator for hardware access from different VMs to guarantee the correctness and stability of the whole system. RISC CPU architectures can be naturally virtualized because all control- and behavior-sensitive instructions are privileged instructions.

Hardware-Assisted CPU Virtualization:

This technique attempts to simplify virtualization because full or para virtualization is complicated. Intel and AMD add an additional mode called privilege mode level (some people call it Ring-1) to x86 processors. Therefore, operating systems can still run at Ring 0 and the hypervisor can run at Ring -1.All the privileged and sensitive instructions are trapped in the hypervisor automatically. This technique removes the difficulty of implementing binary translation of full virtualization. It also lets the operating system run in VMs without modification.

Memory Virtualization:

Virtual memory virtualization is similar to the virtual memory support provided by modern operating systems. In a traditional execution environment, the operating system maintains mappings of virtual memory to machine memory using page tables, which is a one-stage mapping from virtual memory to machine memory. All modern x86 CPUs include a memory management unit (MMU) and a translation lookaside buffer (TLB) to optimize virtual memory performance.

However, in a virtual execution environment, virtual memory virtualization involves sharing the physical system memory in RAM and dynamically allocating it to the physical

memory of the VMs. That means a two-stage mapping process should be maintained by the guest OS and the VMM, respectively: virtual memory to physical memory and physical memory to machine memory. Furthermore, MMU virtualization should be supported, which is transparent to the guest OS. The guest OS continues to control the mapping of virtual addresses to the physical memory addresses of VMs. But the guest OS cannot directly access the actual machine memory. The VMM is responsible for mapping the guest physical memory to the actual machine memory. Figure 2.16 shows the two-level memory mapping procedure.

I/O Virtualization:

I/O virtualization involves managing the routing of I/O requests between virtual devices and the shared physical hardware. There are three ways to implement I/O virtualization:

- Full device emulation
- Para virtualization
- Direct I/O

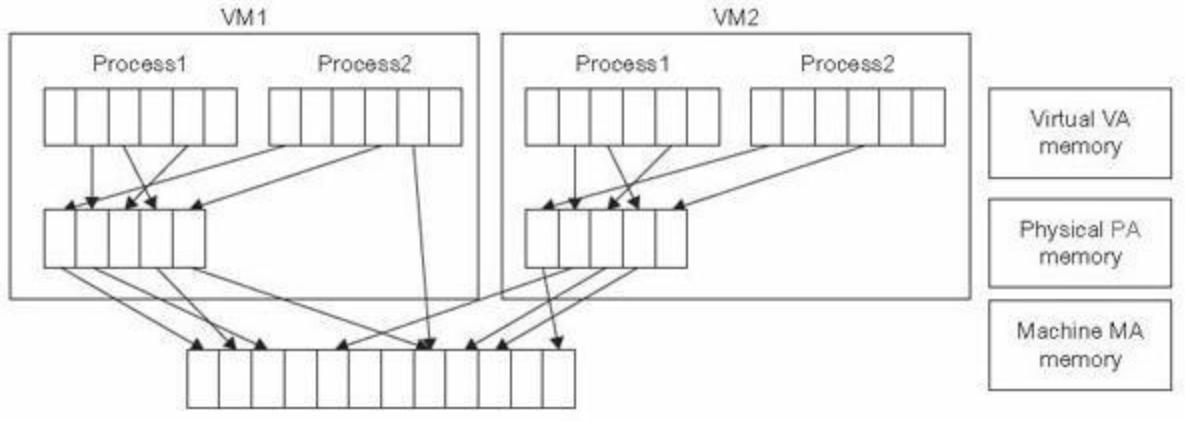


Figure 2.16 Two-level memory mapping procedure.

Full device emulation is the first approach for I/O virtualization. Generally, this approach emulates well known, real-world devices. All the functions of a device or bus infrastructure, such as device enumeration, identification, interrupts, and DMA, are replicated in software. This software is located in the VMM and acts as a virtual device. The I/O access requests of the guest OS are trapped in the VMM which interacts with the I/O devices.

A single hardware device can be shared by multiple VMs that run concurrently. However, software emulation runs much slower than the hardware it emulates. The para

virtualization method of I/O virtualization is typically used in Xen. It is also known as the split driver model consisting of a frontend driver and a backend driver. The frontend driver is running in Domain U and the backend driver is running in Domain 0. They interact with each other via a block of shared memory. The frontend driver manages the I/O requests of the guest OSes and the backend driver is responsible for managing the real I/O devices and multiplexing the I/O data of different VMs. Although para I/O-virtualization achieves better device performance than full device emulation, it comes with a higher CPU overhead.

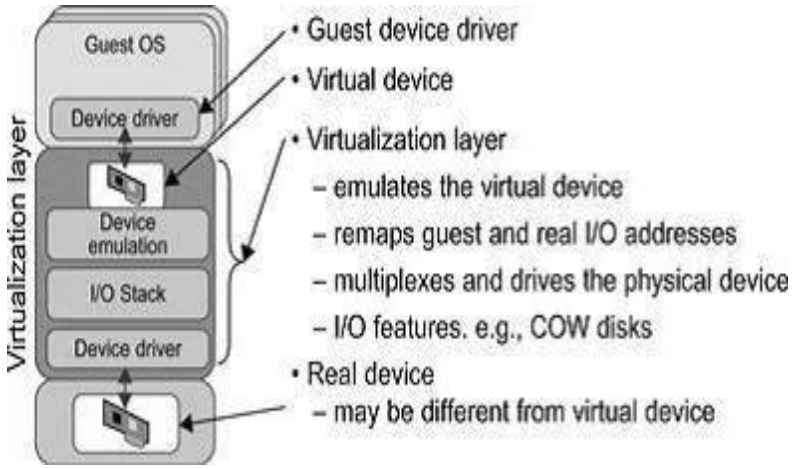


Figure 2.17 Device emulation for I/O virtualization implemented inside the middle layer that maps real I/O devices into the virtual devices for the guest device driver to use.

Virtualization in Multi-Core Processors:

Virtualizing a multi-core processor is relatively more complicated than virtualizing a unicore processor. Though multicore processors are claimed to have higher performance by integrating multiple processor cores in a single chip, muti-core virtualization has raised some new challenges to computer architects, compiler constructors, system designers, and application programmers.

There are mainly two difficulties: Application programs must be parallelized to use all cores fully, and software must explicitly assign tasks to the cores, which is a very complex problem.

Virtualization Support and Disaster Recover:

One very distinguishing feature of cloud computing infrastructure is the use of system virtualization and the modification to provisioning tools. Virtualizations of servers on a shared cluster can consolidate web services. As the VMs are the containers of cloud services, the provisioning tools will first find the corresponding physical machines and deploy the VMs to those nodes before scheduling the service to run on the virtual nodes.

In addition, in cloud computing, virtualization also means the resources and fundamental infrastructure are virtualized. The user will not care about the computing resources that are used for pro-viding the services. Cloud users do not need to know and have no way to discover physical resources that are involved while processing a service request.

Also, application developers do not care about some infrastructure issues such as scalability and fault tolerance (i.e., they are virtualized). Application developers focus on service logic. Figure 2.18 shows the infrastructure needed to virtualize the servers in a data center for implementing specific cloud applications.

Hardware Virtualization

In many cloud computing systems, virtualization software is used to virtualize the hardware. System virtualization software is a special kind of software which simulates the execution of hardware and runs even unmodified operating systems. Cloud computing systems use virtualization software as the running environment for legacy software such as old operating systems and unusual applications. Virtualization software is also used as the platform for developing new cloud applications that enable developers to use any operating systems and programming environments they like.

The development environment and deployment environment can now be the same, which eliminates some runtime problems. Some cloud computing providers have used virtualization technology to provide this service for developers. As mentioned before, system virtualization software is considered the hardware analog mechanism to run an unmodified operating system, usually on bare hardware directly, on top of software. Table 4.4 lists some of the system virtualization software in wide use at the time of this writing. Currently, the VMs installed on a cloud computing platform are mainly used for hosting third-party programs. VMs provide flexible runtime services to free users from worrying about the system environment

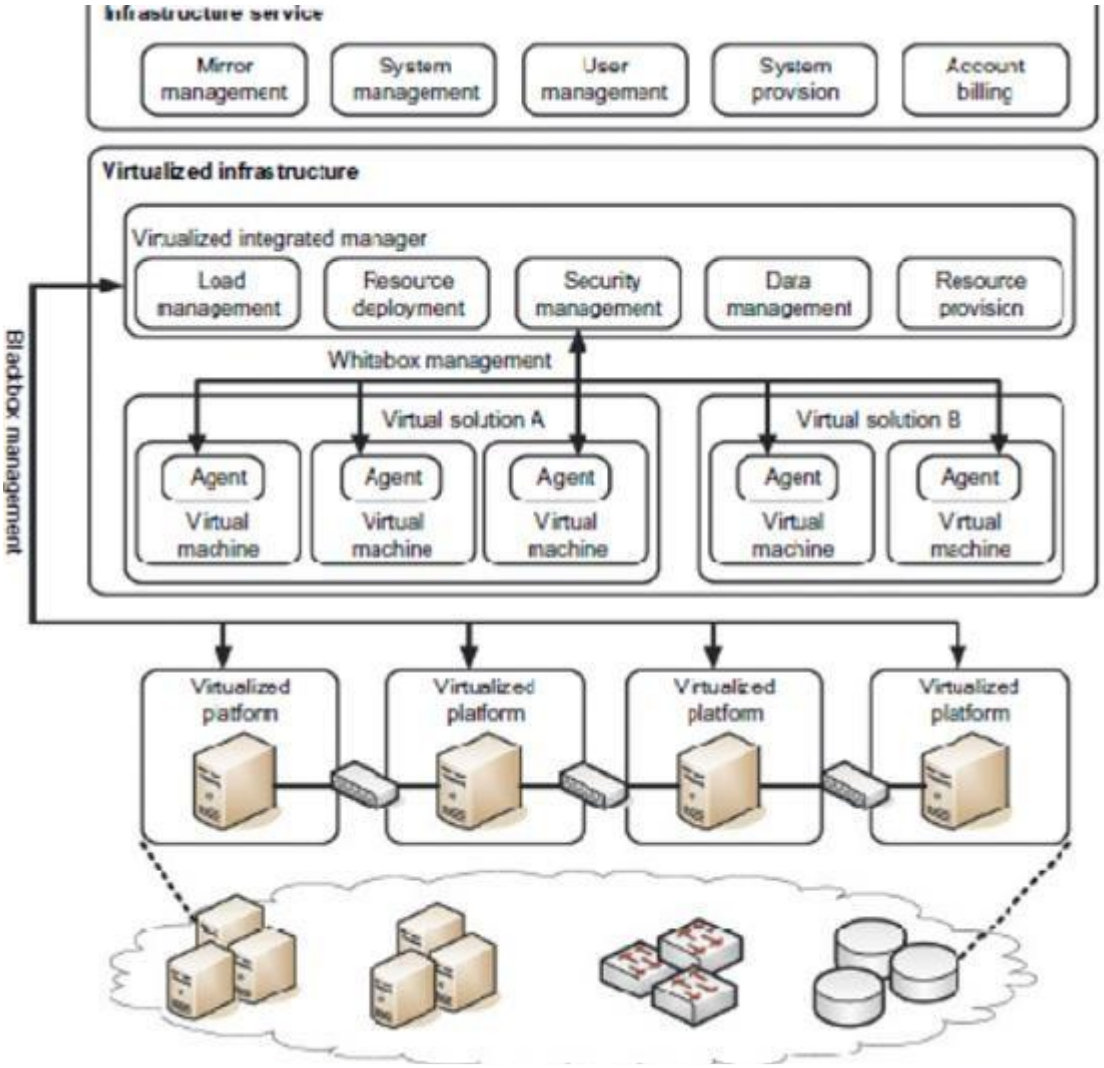
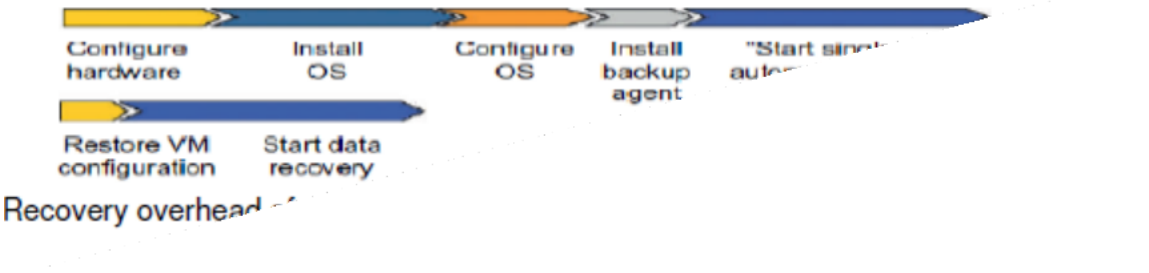


Figure 2.18 Virtualized Storage server and

Using VMs in a cloud computing platform ensures extreme flexibility for users. As the computing resources are shared by many users, a method is required to maximize the users' privileges and still keep them separated safely. Traditional sharing of cluster resources depends on the user and group mechanism on a system. Such sharing is not flexible. Users cannot customize the system for their special purposes. Operating systems cannot be changed. The separation is not complete.

Table 4.4 Virtualized Resources in Compute, Storage, and Network Clouds [4]

Provider	AWS	Microsoft Azure	GAE
Compute cloud with virtual cluster of servers	x86 instruction set, Xen VMs, resource elasticity allows scalability through virtual cluster, or a third party such as RightScale must provide the cluster	Common language runtime VMs provisioned by declarative descript	Predefi fr
Storage cloud with virtual storage	Models for block store (EBS) and augmented key/blob store (SimpleDB), automati scaling varies from EB fuly automatic (S	SO	
Network cloud services	Declarative plac		



An environment that meets one user's requirements often cannot satisfy another user. Virtualization allows users to have full privileges while keeping them separate. Users have full access to their own VMs, which are completely separate from other users' VMs. Multiple VMs can be mounted on the same physical server. Different VMs may run with different OSes. We also need to establish the virtual disk storage and virtual networks needed by the VMs. The virtualized resources form a resource pool.

The virtualization is carried out by special servers dedicated to generating the virtualized resource pool. The virtualized infrastructure (black box in the middle) is built with many virtualizing integration managers. These managers handle loads, resources, security, data, and provisioning functions.

Virtualization Support in Public Clouds

AWS provides extreme flexibility (VMs) for users to execute their own applications. GAE provides limited application-level virtualization for users to build applications only based on the services that are created by Google. Microsoft provides programming-level virtualization (.NET virtualization) for users to build their applications. The VMware tools apply to workstations, servers, and virtual infrastructure. The Microsoft tools are used on PCs and some special servers. The Xen Enterprise tool applies only to Xen-based servers.

Everyone is interested in the cloud; the entire IT industry is moving toward the vision of the cloud. Virtualization leads to HA, disaster recovery, dynamic load leveling, and rich provisioning support. Both cloud computing and utility computing leverage the benefits of virtualization to provide a scalable and autonomous computing environment.

Storage Virtualization for Green Data Centers

IT power consumption in the United States has more than doubled to 3 percent of the total energy consumed in the country. The large number of data centers in the country has contributed to this energy crisis to a great extent. More than half of the companies in the Fortune 500 are actively implementing new corporate energy policies. Recent surveys from both IDC and Gartner confirm the fact that virtualization had a great impact on cost reduction from reduced power consumption in physical computing systems. This alarming situation has made the IT industry become more energy aware.

With little evolution of alternate energy resources, there is an imminent need to con-serve power in all computers. Virtualization and server consolidation have already proven handy in this aspect. Green data centers and benefits of storage virtualization are considered to further strengthen the synergy of green computing.

Virtualization for IaaS

VM technology has increased in ubiquity. This has enabled users to create customized environments atop physical infrastructure for cloud computing.

Use of VMs in clouds has the following distinct benefits:

(1) System administrators consolidate workloads of underutilized servers in fewer servers;

(2) VMs have the ability to run legacy code without interfering with other APIs;

(3) VMs can be used to improve security through creation of sandboxes for running applications with questionable reliability;

(4) Virtualized cloud platforms can apply performance isolation, letting providers offer some guarantees and better QoS to customer applications.

2.9.5.5 VM Cloning for Disaster Recovery

VM technology requires an advanced disaster recovery scheme. One scheme is to recover one physical machine by another physical machine. The second scheme is to recover one VM by another VM. Traditional disaster recovery from one physical machine to another is rather slow, complex, and expensive. Total recovery time is attributed to the hardware configuration, installing and configuring the OS, installing the backup agents, and the long time to restart the physical machine. To recover a VM platform, the installation and configuration times for the OS and backup agents are eliminated.

Therefore, we end up with a much shorter disaster recovery time, about 40 percent of that to recover the physical machines. Virtualization aids in fast disaster recovery by VM encapsulation. The cloning of VMs offers an effective solution. The idea is to make a clone VM on a remote server for every running VM on a local server. Among all the clone VMs, only one needs to be active. The remote VM should be in a suspended mode.

A cloud control center should be able to activate this clone VM in case of failure of the original VM, taking a snapshot of the VM to enable live migration in a minimal amount of time. The migrated VM can run on a shared Internet connection.

Only updated data and modified states are sent to the suspended VM to update its state. The Recovery Property Objective (RPO) and Recovery Time Objective (RTO) are affected by the number of snapshots taken. Security of the VMs should be enforced during live migration of VMs.

COURSE OUTCOMES:

Upon completion of the course the students should be able to:

CO1: Write client side scripting HTML, CSS and JS.

CO2: Implement and architect the server side of the web application.

CO3: Implement Web Application using NodeJS.

CO4: Architect NoSQL databases with MongoDB.

CO5: Implement a full-stack Single Page Application using React, NodeJS and MongoDB and deploy on Cloud.

REFERENCES

1. David Flanagan, “Java Script: The Definitive Guide”, O’Reilly Media, Inc, 7th Edition, 2020
2. Matt Frisbie, "Professional JavaScript for Web Developers", Wiley Publishing, Inc, 4th Edition, ISBN: 978-1-119-36656-0, 2019
3. Alex Banks, Eve Porcello, "Learning React", O’Reilly Media, Inc, 2nd Edition, 2020
4. Marc Wandschneider, “Learning Node”, Addison-Wesley Professional, 2nd Edition, 2016
5. Joe Beda, Kelsey Hightower, Brendan Burns, “Kubernetes: Up and Running”, O’Reilly Media, 1st edition, 2017
6. Paul Zikopoulos, Christopher Bienko, Chris Backer, Chris Konarski, Sai Vennam, “Cloud Without Compromise”, O’Reilly Media, 1st edition, 2021

CO-PO Mapping

CO	POs					
	PO1	PO2	PO3	PO4	PO5	PO6
1	1	1	2	2	2	2
2	2	1	2	2	2	2
3	2	1	3	2	3	2
4	2	1	3	2	3	3
5	2	1	3	2	3	3
Avg	1.8	1	2.6	2	2.6	2.4

MC4202 ADVANCED DATABASE TECHNOLOGY L T P C
3 0 0 3

COURSE OBJECTIVES:

- To understand the working principles and query processing of distributed databases.
- To understand the basics of spatial, temporal and mobile databases and their applications.
- To distinguish the different types of NoSQL databases.
- To understand the basics of XML and create well-formed and valid XML documents.
- To gain knowledge about information retrieval and web search.

UNIT I DISTRIBUTED DATABASES 9

Distributed Systems – Introduction – Architecture – Distributed Database Concepts – Distributed Data Storage – Distributed Transactions – Commit Protocols – Concurrency Control – Distributed Query Processing

UNIT II SPATIAL AND TEMPORAL DATABASES 9

Active Databases Model – Design and Implementation Issues - Temporal Databases - Temporal Querying - Spatial Databases: Spatial Data Types, Spatial Operators and Queries – Spatial Indexing and Mining – Applications -– Mobile Databases: Location and Handoff Management, Mobile Transaction Models – Deductive Databases - Multimedia Databases.

UNIT III NOSQL DATABASES 9

NoSQL – CAP Theorem – Sharding - Document based – MongoDB Operation: Insert, Update, Delete, Query, Indexing, Application, Replication, Sharding–Cassandra: Data Model, Key Space, Table Operations, CRUD Operations, CQL Types – HIVE: Data types, Database Operations, Partitioning – HiveQL – OrientDB Graph database – OrientDB Features

UNIT IV XML DATABASES 9

Structured, Semi structured, and Unstructured Data – XML Hierarchical Data Model – XML Documents – Document Type Definition – XML Schema – XML Documents and Databases – XML Querying – XPath – XQuery

UNIT V INFORMATION RETRIEVAL AND WEB SEARCH 9

IR concepts – Retrieval Models – Queries in IR system – Text Preprocessing – Inverted Indexing – Evaluation Measures – Web Search and Analytics – Current trends.

TOTAL: 45 PERIODS

Suggested Activities:

1. Create a distributed database for any application (ex. book store) and access it using PHP and Python
2. Create spatial database of any place and perform query operations
3. Creating Databases and writing simple queries using MongoDB, DynamoDB, Voldemort Key-Value Distributed Data Store Hbase and Neo4j.
4. Creating XML Documents, Document Type Definition and XML Schema for any e-commerce website and perform XML Querying
5. Perform sentiment analysis for any web document using text preprocessing techniques

COURSE OUTCOMES:

On completion of the course, the student will be able to:

CO1: Design a distributed database system and execute distributed queries.

CO2: Manage Spatial and Temporal Database systems and implement it in corresponding applications.

CO3: Use NoSQL database systems and manipulate the data associated with it.

CO4: Design XML database systems and validate with XML schema.

CO5: Apply knowledge of information retrieval concepts on web databases.

REFERENCES:

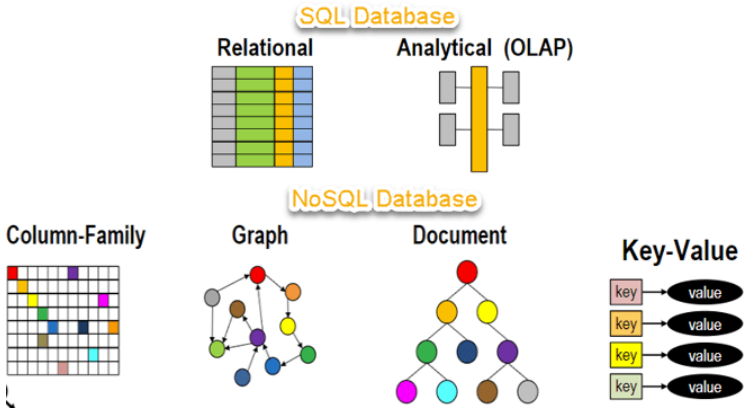
1. Abraham Silberschatz, Henry F Korth, S. Sudharshan, “Database System Concepts”, Seventh Edition, McGraw Hill, 2019.
2. R. Elmasri, S.B. Navathe, “Fundamentals of Database Systems”, Seventh Edition, Pearson Education/Addison Wesley, 2017.
3. Guy Harrison, “Next Generation Databases, NoSQL, NewSQL and Big Data”, First Edition, Apress publishers, 2015

UNIT II NOSQL DATABASES

NoSQL – CAP Theorem – Sharding - Document based – MongoDB Operation: Insert, Update, Delete, Query, Indexing, Application, Replication, Sharding, Deployment – Using MongoDB with PHP / JAVA – Advanced MongoDB Features – Cassandra: Data Model, Key Space, Table Operations, CRUD Operations, CQL Types – HIVE: Data types, Database Operations, Partitioning – HiveQL – OrientDB Graph database – OrientDB Features

Introduction to NOSQL Systems

NoSQL database stands for "Not Only SQL" or "Not SQL."



NoSQL Database is a non-relational Data Management System, that does not require a fixed schema. It avoids joins, and is easy to scale. The major purpose of using a NoSQL database is for distributed data stores with humongous data storage needs. NoSQL is used for Big data and real-time web apps. For example, companies like Twitter, Facebook and Google collect terabytes of user data every single day.

Why NoSQL?

The concept of NoSQL databases became popular with Internet giants like Google, Facebook, Amazon, etc. who deal with huge volumes of data. The system response time becomes slow when you use RDBMS for massive volumes of data.

To resolve this problem, we could "scale up" our systems by upgrading our existing hardware. This process is expensive.

The alternative for this issue is to distribute database load on multiple hosts whenever the load increases. This method is known as "scaling out."

Brief History of NoSQL Databases

1998- Carlo Strozzi use the term NoSQL for his lightweight, open-source relational database
2000- Graph database Neo4j is launched
2004- Google BigTable is launched
2005- CouchDB is launched
2007- The research paper on Amazon Dynamo is released
2008- Facebooks open sources the Cassandra project
2009- The term NoSQL was reintroduced

Emergence of NOSQL Systems

Many companies and organizations are faced with applications that store vast amounts of data. Consider a free e-mail application, such as Google Mail or Yahoo Mail or other similar service—this application can have millions of users, and each user can have thousands of e-mail messages. There is a need for a storage system that can manage all these e-mails; a structured relational SQL system may not be appropriate.

Because (1) SQL systems offer too many services (powerful query language, concurrency control, etc.), which this application may not need; and (2) a structured data model such the traditional relational model may be too restrictive.

Another example, consider an application such as Facebook, with millions of users who submit posts, many with images and videos; then these posts must be displayed on pages of other users using the social media relationships among the users. User profiles, user relationships, and posts must all be stored in a huge collection of data stores, and the appropriate posts must be made available to the sets of users that have signed up to see these posts

Google developed a proprietary NOSQL system known as BigTable, which is used in many of Google's applications that require vast amounts of data storage, such as Gmail, Google Maps, and Web site indexing s. Google's innovation led to the category of NOSQL systems known as **column-based** or **wide column** stores; they are referred to as **column family** stores.

Apache Hbase is an open source NOSQL system based on similar concepts.

Google's innovation led to the category of NOSQL systems known as column-based or wide column stores;
Amazon developed a NOSQL system called DynamoDB that is available through Amazon's cloud services. This innovation led to the category known as key-value data stores or sometimes key-tuple or key-object data stores.

Facebook developed a NOSQL system called Cassandra, which is now open source and known as Apache Cassandra. This NOSQL system uses concepts from both key-value stores and column-based systems.

MongoDB and CouchDB, which are classified as document-based NOSQL systems or document stores.

Another category of NOSQL systems is the graph-based NOSQL systems, or graph databases; these include Neo4J and GraphBase.

Features of NoSQL

- Non-relational
- NoSQL databases never follow the relational model
- Never provide tables with flat fixed-column records
- Work with self-contained aggregates or BLOBs
- Doesn't require object-relational mapping and data normalization
- No complex features like query languages, query planners, referential integrity joins, ACID.

Characteristics of NOSQL Systems

1. Scalability:

There are two kinds of scalability in distributed systems: horizontal and vertical. In NOSQL systems, **horizontal scalability** is generally used, where the distributed system is expanded by adding more nodes for data storage and processing as the volume of data grows.
Vertical scalability, on the other hand, refers to expanding the storage and computing power of existing nodes. In NOSQL systems, horizontal scalability is employed.

Scalability = the **ability** of a **system** to handle a ***growing*** **amount** of **work**

2 ways to achieve scalability:
Vertical scalability: Handle more load by using faster processors
Horizontal scalability: Handle more load by using more processors

NOSQL systems are always horizontally scalable:
Processing/storage capacity in NOSQL systems are increased by adding more processing/storage nodes.

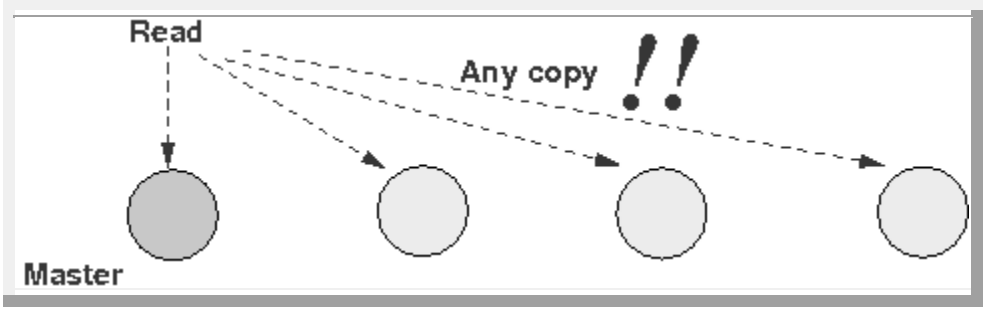
2. Availability, Replication and Eventual Consistency: Many applications that use NOSQL systems require continuous system availability. To accomplish this, data is replicated over two or more nodes in a transparent manner, so that if one node fails, the data is still available on other nodes.
Replication improves data availability and can also improve read performance.

3. Replication Models: Two major replication models are used in NOSQL systems:
master-slave and master-master replication. **Master-slave replication** requires one copy to be the master copy; all write operations must be applied to the master copy and then propagated to the slave copies.
The **master-master replication** allows reads and writes at any of the replicas but may not guarantee that reads at nodes that store different copies see the same values.

Master-slave replication:

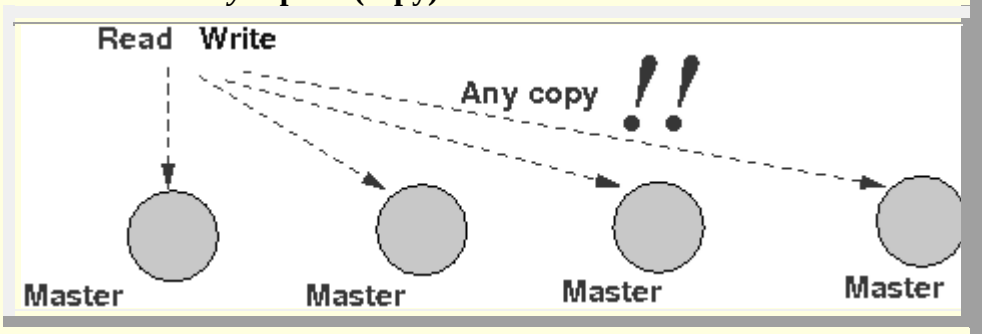
- ***One*** of the **copies** is the **master copy**
- ***All*** **write operations must** apply to the **master copy**
- The **slave copies** will ***eventually*** receive the **updates**

Read operations usually can access ***any*** **data copy** for **performance.**



Master-master replication:

- Allows **read/write** to ***any* replica (copy)**



- The **write** operation includes a ***time stamp.***

4. Sharding of Files: In many NOSQL applications, files (or collections of data objects) can have many millions of records (or documents or objects), and these records can be accessed concurrently by thousands of users. It is not practical to store the whole file in one node.
Sharding (horizontal partitioning) is often employed in NOSQL systems. This serves to distribute the load of accessing the file records to multiple nodes.

5. High-Performance Data Access: In many NOSQL applications, it is necessary to find individual records or objects (data items) from among the millions of data records or objects in a file. To achieve this, most systems use one of two techniques: hashing or range partitioning on object keys.
In **hashing**, a hash function *h(K)* is applied to the key *K*, and the location of the object with key *K* is determined by the value of *h(K)*. In **range partitioning**, the location is determined via a range of key values;

for example, location *i* would hold the objects whose key values K are in the range $Ki_{min} \leq K \leq Ki_{max}$.

Types / Categories of NoSQL Databases
NoSQL Databases are mainly categorized into four types: Key-value pair, Column-oriented, Graph-based and Document-oriented. Every category has its unique attributes and limitations. None of the above-specified database is better to solve all the problems. Users should select the database based on their product needs.

- Key-value Pair Based
- Column-oriented Graph
- Graphs based
- Document-oriented

1. Document-based NOSQL systems: These systems store data in the form of documents using well-known formats, such as JSON (JavaScript Object Notation).
Documents are accessible via their document id, but can also be accessed rapidly using other indexes.
Amazon SimpleDB, CouchDB, MongoDB, Riak, Lotus Notes, MongoDB, are popular Document originated DBMS systems.

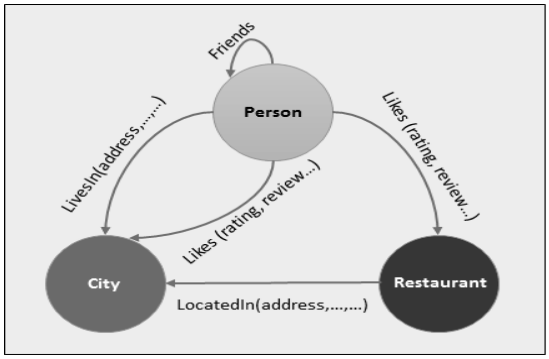
2. NOSQL key-value stores: These systems have a simple data model based on fast access by the key to the value associated with the key; the value can be a record or an object or a document or even have a more complex data structure.
It is designed in such a way to handle lots of data and heavy load. Key-value pair storage databases store data as a hash table where each key is unique, and the value can be a JSON, BLOB(Binary Large Objects), string, etc.
It is one of the most basic NoSQL database example. This kind of NoSQL database is used as a collection, dictionaries, associative arrays, etc. Key value stores help the developer to store schema-less data. They work best for shopping cart contents.
Redis, Dynamo, Riak are some NoSQL examples of key-value store DataBases.

3. Column-based or wide column NOSQL systems: These systems partition a table by column into column families (a form of vertical partitioning), where each column family is stored in its own files. They also allow versioning of data values.

ColumnFamily			
Row Key	Column Name		
	Key	Key	Key
	Value	Value	Value
	Column Name		
	Key	Key	Key
	Value	Value	Value

Column-oriented databases work on columns and are based on BigTable paper by Google. Every column is treated separately. Values of single column databases are stored contiguously.
They deliver high performance on aggregation queries like SUM, COUNT, AVG, MIN etc. as the data is readily available in a column.
Column-based NoSQL databases are widely used to manage data warehouses, business intelligence, CRM, Library card catalogs,
HBase, Cassandra, HBase, Hypertable are NoSQL query examples of column based database.

4. Graph-based NOSQL systems: A graph type database stores entities as well the relations amongst those entities. The entity is stored as a node with the relationship as edges. An edge gives a relationship between nodes. Every node and edge has a unique identifier.
Data is represented as graphs, and related nodes can be found by traversing the edges using path expressions.



5. Hybrid NOSQL systems: These systems have characteristics from two or Key Value Pair Based

Advantages of NoSQL

- Can be used as Primary or Analytic Data Source
- Big Data Capability
- No Single Point of Failure
- Easy Replication
- No Need for Separate Caching Layer
- It provides fast performance and horizontal scalability.
- Can handle structured, semi-structured, and unstructured data with equal effect
- Object-oriented programming which is easy to use and flexible

- NoSQL databases don't need a dedicated high-performance server
- Support Key Developer Languages and Platforms
- Simple to implement than using RDBMS
- It can serve as the primary data source for online applications.
- Handles big data which manages data velocity, variety, volume, and complexity
- Excels at distributed database and multi-data center operations
- Eliminates the need for a specific caching layer to store data
- Offers a flexible schema design which can easily be altered without downtime or service disruption

Disadvantages of NoSQL

- No standardization rules
- Limited query capabilities
- RDBMS databases and tools are comparatively mature
- It does not offer any traditional database capabilities, like consistency when multiple transactions are performed simultaneously.
- When the volume of data increases it is difficult to maintain unique values as keys become difficult
- Doesn't work as well with relational data
- The learning curve is stiff for new developers
- Open source options so not so popular for enterprises.

CAP Theorem

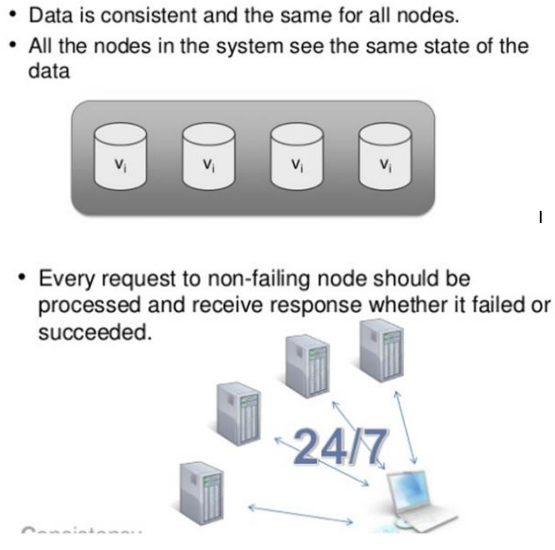
It is very important to understand the limitations of NoSQL database. NoSQL can not provide consistency and high availability together. This was first expressed by Eric Brewer in CAP Theorem.

CAP theorem states that we can only achieve at most two out of three guarantees for a database: Consistency, Availability and Partition Tolerance.

1. Consistency
2. Availability
3. Partition Tolerance

Consistency:

The data should remain consistent even after the execution of an operation. This means once data is written, any future read request should contain that data. For example, after updating the order status, all the clients should be able to see the same data.



Availability:

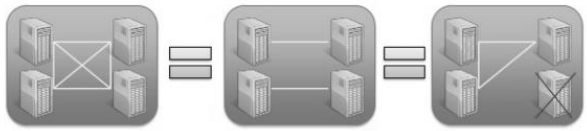
The database should always be available and responsive. It should not have any downtime.

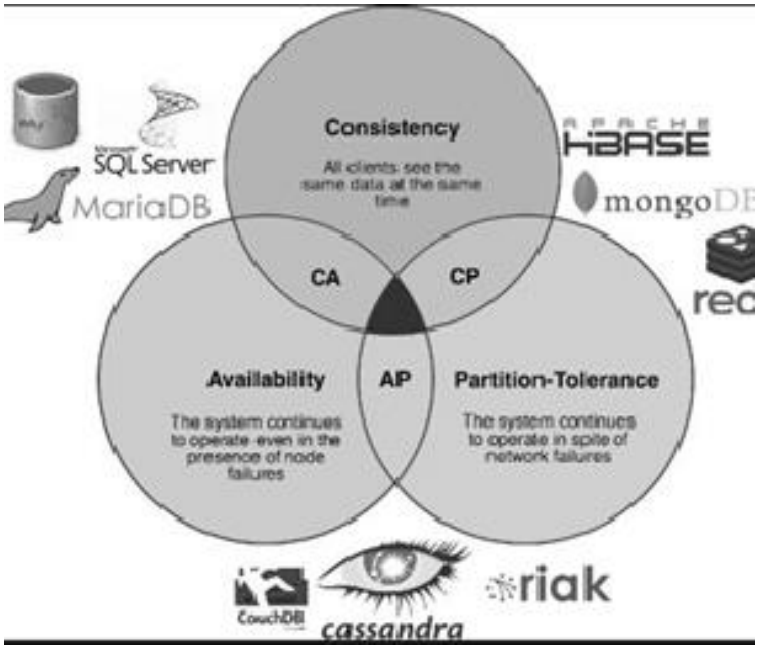
Availability means the system should always perform reads/writes on any non-failing node of the cluster successfully without any error.

Partition Tolerance:

Partition Tolerance means that the system should continue to function even if the communication among the servers is not stable. For example, the servers can be partitioned into multiple groups which may not communicate with each other. Here, if part of the database is unavailable, other parts are always unaffected.

- **P**artition-Tolerance
  - If some nodes crash / communication fails, service still performs as expected





Eventual Consistency

The term "eventual consistency" means to have copies of data on multiple machines to get high availability and scalability. Thus, changes made to any data item on one machine has to be propagated to other replicas.

Data replication may not be instantaneous as some copies will be updated immediately while others in due course of time. These copies may be mutually, but in due course of time, they become consistent. Hence, the name eventual consistency.

Sharding

Technique to reduce the processing load per node

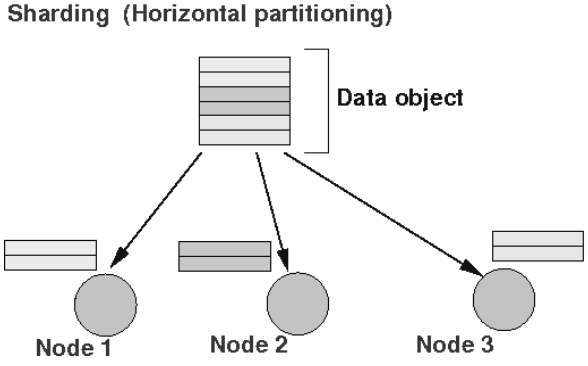
Sharding (= horizontal partitioning)

How the data is stored in sharding:

Each "shard is stored at a different node

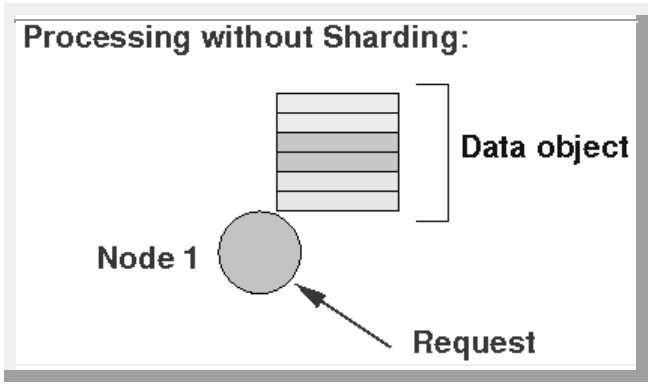
Graphically: Load distribution using sharding:

Processing is distributed over multiple nodes:

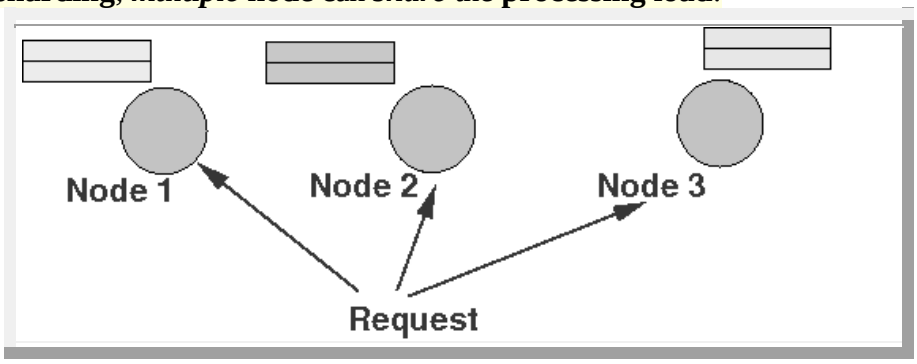


Without sharding,

a ***single*** **node** must **process** the **entire data object**:

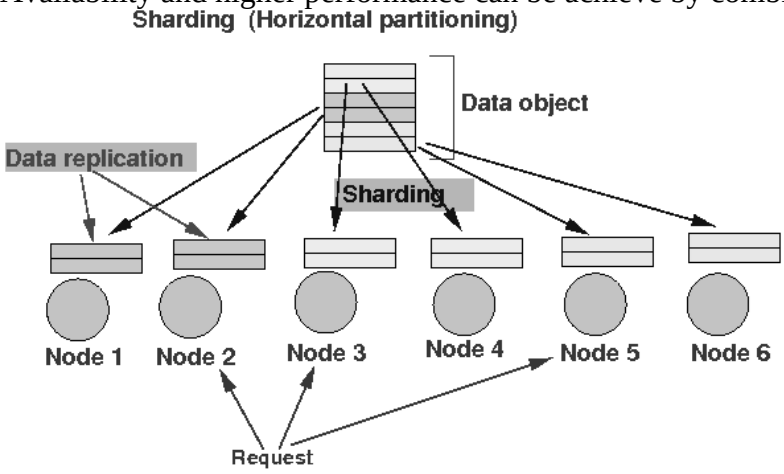


With* sharding**, ***multiple* node** can ***share the **processing load**:



Sharding combined with data replication

Availability and higher performance can be achieve by combining sharding and data replication



Commonly used sharding techniques

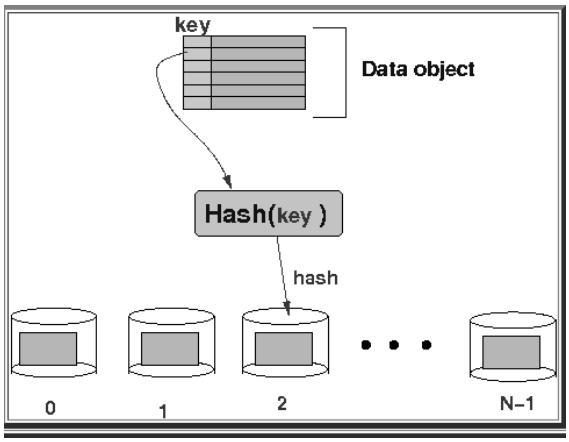
Hashing

Range partitioning

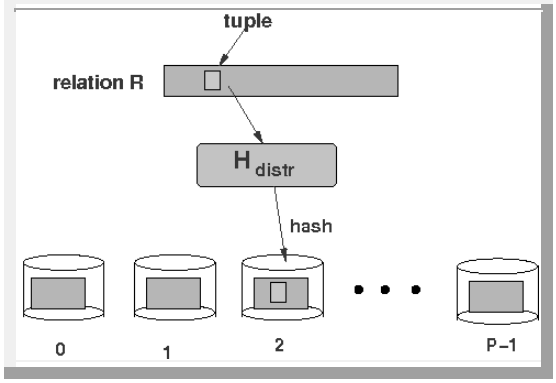
Hashing-based sharding:

Used in key-value NOSQL systems

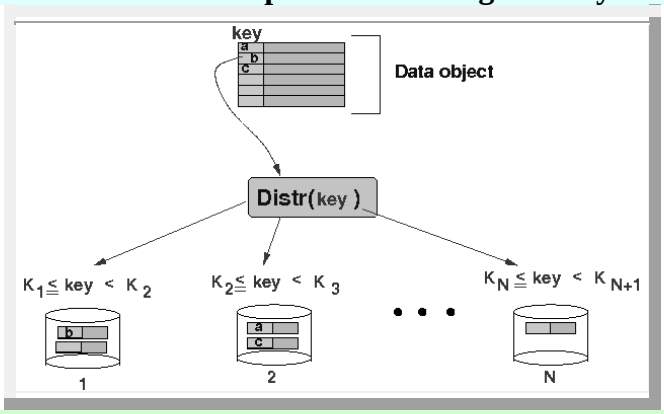
Data is sharded using the key:



This **approach** is ***similar*** to the **tuple storage** in **parallel database processing**:



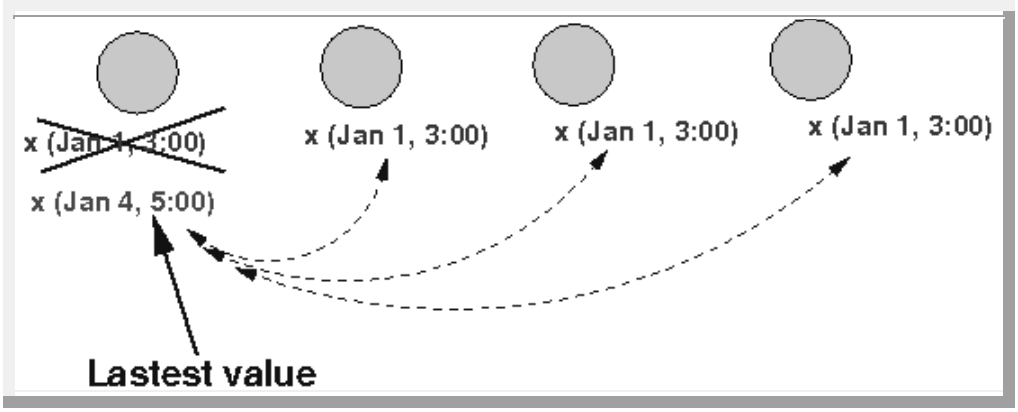
Range-based sharding:
Data is **sharded** based on **pre-defined ranges** of **key values**:



Range-based sharding can handle **range-based *queries*** !!!

Increasing parallelism in NOSQL systems

- NOSQL system adopt *relaxed* data consistency *constraints*:
- An update is applied to *one* copy without locking the other copies:

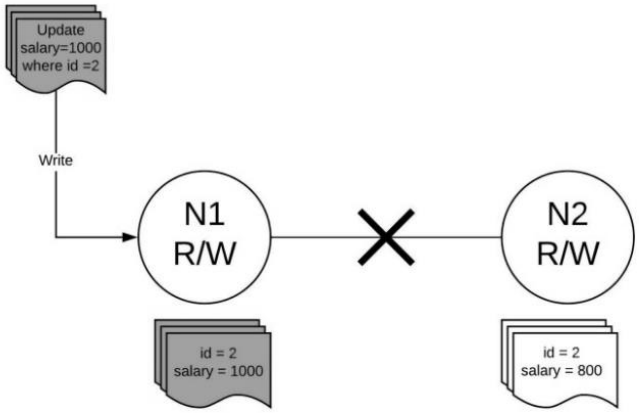


So: the **data** can be **temporally *inconsistent*** (see above figure) !!!!

NOSQL characteristics related to distributed databases and distributed systems.

How a Distributed System breaks Consistency or Availability?

Scenario 1: Failing to propagate update request to other nodes.
Say, we have two nodes(N1 & N2) in a cluster and both nodes can accept read and write requests.



In the above diagram, the N1 node gets an update request for **id 2** and updates the salary from 800 to 1000. But, since there is network partition, hence, N1 can not send the latest update to N2.

So, when a read request comes to N2, it can do either of two things:

1. Respond with whatever data it has, in this case, salary = 800, and update the data when the network partition resolves — **making the system Available but Inconsistent.**
2. Simply return with an error, saying "I do not have updated data" — **making a system Consistent but Unavailable** by not returning inconsistent data.

Document-Based NOSQL Systems and MongoDB

Document-based or document-oriented NOSQL systems typically store data as **collections** of similar **documents**. These types of systems are also sometimes known as **document stores**. The individual documents somewhat resemble *complex objects* or XML documents.

Although the documents in a collection should be similar, they can have different data elements (attributes).
The system basically extracts the data element names from the self-describing documents in the collection, and the user can request that the system create indexes on some of the data elements. Documents can be specified in various formats, such as XML. A popular language to specify documents in NOSQL systems is JSON (JavaScript Object Notation).
There are many document-based NOSQL systems, including MongoDB and CouchDB, among many others.

Database

Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

Collection

Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

Document

A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

The following table shows the relationship of RDBMS terminology with MongoDB.

RDBMS (SQL)	**MongoDB** (NOSQL)
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by MongoDB itself)
Database Server and Client	
mysqld/Oracle	mongod
mysql/sqlplus	mongo

Why Use MongoDB?

- **Document Oriented Storage** − Data is stored in the form of JSON style documents.
- Index on any attribute
- Replication and high availability
- Auto-Sharding
- Rich queries
- Fast in-place updates
- Professional support by MongoDB

Where to Use MongoDB?

- Big Data
- Content Management and Delivery
- Mobile and Social Infrastructure
- User Data Management
- Data Hub

MongoDB Data Model

MongoDB is an open-source document database and leading NoSQL database.
MongoDB is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability. MongoDB works on concept of collection and document.

MongoDB documents are stored in BSON (Binary JSON) format, which is a variation of JSON with some additional data types and is more efficient for storage than JSON.
Individual documents are stored in a collection. The operation createCollection is used to create each collection.
Theoperation createCollection is used to create each collection. For example, the following command can be used to create a collection called **project** to hold PROJECT objects from the COMPANY database ;
db.createCollection("project", { capped : true, size : 1310720, max : 500 })
The first parameter "project" is the **name** of the collection, which is followed by an optional document that specifies **collection options**. In our example, the collection is **capped**; this means it has upper limits on its storage space (**size**) and number ofdocuments (**max**).

For our example, we will create another document collection called **worker** to hold information about the EMPLOYEEs who work on each project;
For example:
db.createCollection("worker", { capped : true, size : 5242880, max : 2000 }))

Each document in a collection has a unique **ObjectId** field, called **_id**, which is automatically indexed in the collection unless the user explicitly requests no index for the _id field. The value of ObjectId can be *specified by the user*, or it can be *system-generated* if the user does not specify an _id field for a particular document.
System-generated ObjectIds have a specific format, which combines the timestamp when the object is created (4 bytes, in an internal MongoDB format), the node id (3 bytes), the process id (2 bytes), and a counter (3 bytes) into a 16-byte Id value.
A collection does not have a schema. The structure of the data fields in documents is chosen based on how documents will be accessed and used, and the user can choose a normalized design or denormalized form.

Sample Document

Following example shows the document structure of a blog site, which is simply a comma separated key value pair.

Example of simple documents in MongoDB.
(a) Denormalized document design with embedded subdocuments.
(b) Embedded array of document references.
(c) Normalized documents.

(a) project document with an array of embedded workers:

```
{
    _id:          "P1",
    Pname:        "ProductX",
    Plocation:    "Bellaire",
    Workers: [
                { Ename: "John Smith",
                  Hours: 32.5
                },
                { Ename: "Joyce English",
                  Hours: 20.0
                }
            ]
);
```

(b) project document with an embedded array of worker ids:

```
{
    _id:          "P1",
    Pname:        "ProductX",
    Plocation:    "Bellaire",
    WorkerIds:    [ "W1", "W2" ]
}
    { _id:        "W1",
    Ename:        "John Smith",
    Hours:        32.5
}
    { _id:        "W2",
    Ename:        "Joyce English",
    Hours:        20.0
}
```

.

(c) normalized project and worker documents (not a fully normalized design for M:N relationships):

```
{
    _id:          "P1",
    Pname:        "ProductX",
    Plocation:    "Bellaire"
}
{   _id:          "W1",
    Ename:        "John Smith",
    ProjectId:    "P1",
    Hours:        32.5
}
```

```
{   _id:        "W2",
    Ename:      "Joyce English",
    ProjectId:  "P1",
    Hours:      20.0
}
```

Example of simple documents in MongoDB. (d) Inserting the documents in Figure 24.1(c) into their collections.

(d) inserting the documents in (c) into their collections "project" and "worker":

```
db.project.insert( { _id: "P1", Pname: "ProductX", Plocation: "Bellaire" } )
db.worker.insert( [ { _id: "W1", Ename: "John Smith", ProjectId: "P1", Hours: 32.5 },
                    { _id: "W2", Ename: "Joyce English", ProjectId: "P1",
                      Hours: 20.0 } ] )
```

Advantages of MongoDB over RDBMS

- **Schema less** − MongoDB is a document database in which one collection holds different documents. Number of fields, content and size of the document can differ from one document to another.
- Structure of a single object is clear.
- No complex joins.
- Deep query-ability. MongoDB supports dynamic queries on documents using a document-based query language that's nearly as powerful as SQL.
- Tuning.
- **Ease of scale-out** − MongoDB is easy to scale.
- Conversion/mapping of application objects to database objects not needed.
- Uses internal memory for storing the (windowed) working set, enabling faster access of data.

MongoDB CRUD Operations

MongoDb has several **CRUD operations**, where CRUD stands for (create, read, update, delete). Documents can be *created* and inserted into their collections using the **insert** operation, whose format is:

db.<collection_name>.insert(<document(s)>)

The parameters of the insert operation can include either a single document or an array of documents, as shown in Figure 24.1(d). The *delete* operation is called **remove**, and the format is:

db.<collection_name>.remove(<condition>)

The documents to be removed from the collection are specified by a Boolean condition on some of the fields in the collection documents. There is also an **update** operation, which has a condition to select certain documents, and a *$set* clause to specify the update. It is also possible to use the update operation to replace an existing document with another one but keep the same ObjectId.

For *read* queries, the main command is called **find**, and the format is:

db.<collection_name>.find(<condition>)

The use Command

MongoDB **use DATABASE_NAME** is used to create database. The command will create a new database if it doesn't exist, otherwise it will return the existing database.

use DATABASE_NAME

Example

If you want to use a database with name **<mydb>**, then **use DATABASE** statement would be as follows −

```
>use mydb
switched to db mydb
```

To check your currently selected database, use the command **db**

```
>db
mydb
```

If you want to check your databases list, use the command **show dbs**.

```
>show dbs
local     0.78125GB
test      0.23012GB
```

The dropDatabase() Method

MongoDB **db.dropDatabase()** command is used to drop a existing database.

Basic syntax of **dropDatabase()** command is as follows −

```
db.dropDatabase()
```

This will delete the selected database. If you have not selected any database, then it will delete default 'test' database.

```
>use mydb
switched to db mydb
>db.dropDatabase()
>{ "dropped" : "mydb", "ok" : 1 }
>
```

The createCollection() Method

MongoDB **db.createCollection(name, options)** is used to create collection.

Basic syntax of **createCollection()** command is as follows −

```
db.createCollection(name, options)
```

In the command, **name** is name of collection to be created. **Options** is a document and is used to specify configuration of collection.

Parameter	Type	Description
Name	String	Name of the collection to be created
Options	Document	(Optional) Specify options about memory size and indexing

```
>use test
switched to db test
>db.createCollection("mycollection")
{ "ok" : 1 }
>
```

MongoDB supports many datatypes. Some of them are −

- **String** − This is the most commonly used datatype to store the data. String in MongoDB must be UTF-8 valid.
- **Integer** − This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.
- **Boolean** − This type is used to store a boolean (true/ false) value.
- **Double** − This type is used to store floating point values.
- **Min/ Max keys** − This type is used to compare a value against the lowest and highest BSON elements.
- **Arrays** − This type is used to store arrays or list or multiple values into one key.
- **Timestamp** − ctimestamp. This can be handy for recording when a document has been modified or added.
- **Object** − This datatype is used for embedded documents.
- **Null** − This type is used to store a Null value.
- **Symbol** − This datatype is used identically to a string; however, it's generally reserved for languages that use a specific symbol type.
- **Date** − This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.
- **Object ID** − This datatype is used to store the document's ID.
- **Binary data** − This datatype is used to store binary data.
- **Code** − This datatype is used to store JavaScript code into the document.
- **Regular expression** − This datatype is used to store regular expression.

The insert() Method

To insert data into MongoDB collection, you need to use MongoDB's **insert()** or **save()** method.

The basic syntax of **insert()** command is as follows −

```
>db.COLLECTION_NAME.insert(document)
```

Example

```
> db.users.insert({
... _id : ObjectId("507f191e810c19729de860ea"),
... title: "MongoDB Overview",
... description: "MongoDB is no sql database",
... by: "tutorials point",
... url: "http://www.tutorialspoint.com",
... tags: ['mongodb', 'database', 'NoSQL'],
... likes: 100
... })
WriteResult({ "nInserted" : 1 })
>
```

The insertOne() method

If you need to insert only one document into a collection you can use this method.

```
> db.createCollection("empDetails")
{ "ok" : 1 }
> db.empDetails.insertOne(
        {
                First_Name: "Radhika",
                Last_Name: "Sharma",
                Date_Of_Birth: "1995-09-26",
                e_mail: "radhika_sharma.123@gmail.com",
                phone: "9848022338"
        })
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5dd62b4070fb13eec3963bea")
}
>
```

The insertMany() method

You can insert multiple documents using the insertMany() method. To this method you need to pass an array of documents.

Following example inserts three different documents into the empDetails collection using the insertMany() method.

```
> db.empDetails.insertMany(
        [
                {
                        First_Name: "Radhika",
                        Last_Name: "Sharma",
                        Date_Of_Birth: "1995-09-26",
                        e_mail: "radhika_sharma.123@gmail.com",
                        phone: "9000012345"
                },
                {
                        First_Name: "Rachel",
                        Last_Name: "Christopher",
                        Date_Of_Birth: "1990-02-16",
                        e_mail: "Rachel_Christopher.123@gmail.com",
                        phone: "9000054321"
                },
                {
                        First_Name: "Fathima",
                        Last_Name: "Sheik",
                        Date_Of_Birth: "1990-02-16",
                        e_mail: "Fathima_Sheik.123@gmail.com",
                        phone: "9000054321"
                }
        ]
)
```

The find() Method

To query data from MongoDB collection, you need to use MongoDB's **find()** method.

The basic syntax of **find()** method is as follows −

```
>db.COLLECTION_NAME.find()
```

find() method will display all the documents in a non-structured way.

The findOne() method

Apart from the find() method, there is **findOne()** method, that returns only one document.

```
>db.COLLECTIONNAME.findOne()
```

Example

Following example retrieves the document with title MongoDB Overview.

```
> db.mycol.findOne({title: "MongoDB Overview"})
{
        "_id" : ObjectId("5dd6542170fb13eec3963bf0"),
        "title" : "MongoDB Overview",
        "description" : "MongoDB is no SQL database",
        "by" : "tutorials point",
        "url" : "http://www.tutorialspoint.com",
        "tags" : [
                "mongodb",
                "database",
                "NoSQL"
        ],
        "likes" : 100
}
```

MongoDB's **update()** and **save()** methods are used to update document into a collection. The update() method updates the values in the existing document while the save() method replaces the existing document with the document passed in save() method.

MongoDB Update() Method

The update() method updates the values in the existing document.

Syntax

The basic syntax of **update()** method is as follows −

```
>db.COLLECTION_NAME.update(SELECTION_CRITERIA, UPDATED_DATA)
```

MongoDB Save() Method

The **save()** method replaces the existing document with the new document passed in the save() method.

Syntax

The basic syntax of MongoDB **save()** method is shown below −

```
>db.COLLECTION_NAME.save({_id:ObjectId(),NEW_DATA})
```

The remove() Method

MongoDB's **remove()** method is used to remove a document from the collection. remove() method accepts two parameters. One is deletion criteria and second is justOne flag.

- **deletion criteria** − (Optional) deletion criteria according to documents will be removed.
- **justOne** − (Optional) if set to true or 1, then remove only one document.

Syntax

Basic syntax of **remove()** method is as follows −

```
>db.COLLECTION_NAME.remove(DELLETION_CRITTERIA)
```

Remove Only One

If there are multiple records and you want to delete only the first record, then set **justOne** parameter in **remove()** method.

```
>db.COLLECTION_NAME.remove(DELETION_CRITERIA,1)
```

Remove All Documents

If you don't specify deletion criteria, then MongoDB will delete whole documents from the collection. **This is equivalent of SQL's truncate command.**

```
> db.mycol.remove({})
WriteResult({ "nRemoved" : 2 })
> db.mycol.find()
>
```

MC5105 - ADVANCED DATABASE TECHNOLOGY

UNIT I DISTRIBUTED DATABASES

Distributed Systems – Introduction – Architecture – Distributed Database Concepts – DistributedData Storage – Distributed Transactions – Commit Protocols – Concurrency Control – DistributedQuery Processing

A **distributed computing system** consists of a number of Processing sites (or) Nodes or CPUs that are interconnected by a computer network and that cooperate in performing certain assigned tasks.

A distributed system is a collection of autonomous computers linked by a computer network that appear to the users of the system as a single computer.
By running distributed system software the computers are enabled to:
1. Coordinate their activities
2. Share resources: hardware, software, data.

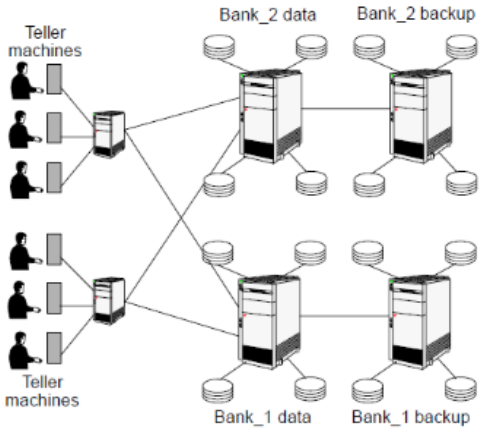
Examples of distributed systems

Finance and commerce	eCommerce e.g. Amazon and eBay, PayPal, online banking and trading
The information society	Web information and search engines, ebooks, Wikipedia; social networking: Facebook and MySpace.
Creative industries and entertainment	online gaming, music and film in the home, user-generated content, e.g. YouTube, Flickr
Healthcare	health informatics, on online patient records, monitoring patients
Education	e-learning, virtual learning environments; distance learning
Transport and logistics	GPS in route finding systems, map services: Google Maps, Google Earth
Science	The Grid as an enabling technology for collaboration between scientists
Environmental management	sensor technology to monitor earthquakes, floods or tsunamis

Example:

Automatic banking (teller machine) system

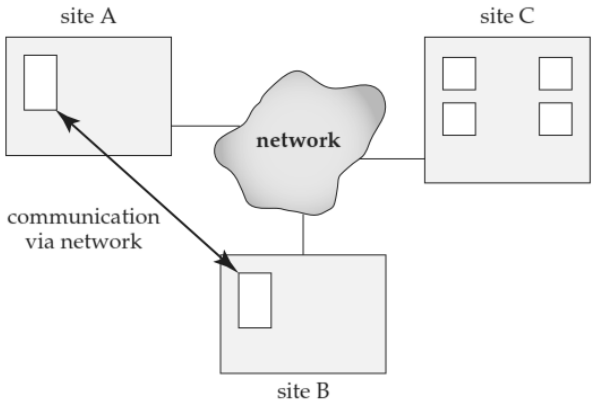
- Primary requirements: security and reliability.
- Consistency of replicated data.
- Concurrent transactions (operations which involve accounts in different banks; simultaneous access from several users, etc).
- Fault tolerance



DISTRIBUTED SYSTEMS

In a **distributed database system**, the database is stored on several computers. The computers in a distributed system communicate with one another through various communication media, such as high-speed private networks or the Internet.
The computers in a distributed system may vary in size and function, ranging from workstations up to mainframe systems.

The computers in a distributed system are referred to by a number of different names, such as **sites** or **nodes**, depending on the context in which they are mentioned. We mainly use the term **site**, to emphasize the physical distribution of these systems.
The main differences between shared-nothing parallel databases and distributed databases are that distributed databases are typically geographically separated, are separately administered, and have a slower interconnection.



A **local transaction** is one that accesses data only from sites where the transaction was initiated.
A **global transaction** is one that either accesses data in several different sites.

Reasons for building distributeddatabase systems

Sharing data - The major advantage in building a distributed database system is the provision of an environment where users at one site may be able to access the data residing at other sites. For instance, in a distributed university system, where each campus stores data related to that campus, it is possible for a user in one campus to access data in another campus.

Autonomy - In a centralized system, the database administrator of the central site controls the database. In a distributed system, there is a global database administrator responsible for the entire system.
A part of these responsibilities is delegated to the local database administrator for each site.
Depending on the design of the distributed database system, each administrator may have a different degree of **local autonomy**.
Availability - If one site fails in a distributed system, the remaining sites may be able to continue operating. In particular, if data items are **replicated** in several sites, the failure of a site does not necessarily imply the shutdown of the system.

Recovery - The failure of one site must be detected by the system, and appropriate action may be needed to recover from the failure. The system must no longer use the services of the failed site. Finally, when the failed site recovers or is repaired, mechanisms must be available to integrate it smoothly back into the system.

Example of Distributed Sysem

To illustrate the difference between the two types of transactions—local and global—at the sites, consider a transaction to add $50 to account number A-177 located at the Valleyview branch.
If the transaction was initiated at the Valleyview branch, then it is considered local; otherwise, it is considered global.
A transaction to transfer $50 from account A-177 to account A-305, which is located at the Hillside branch, is a global transaction, since accounts in two different sites are accessed as a result of its execution.
In an ideal distributed database system, the sites would share a common global schema (although some relations may be stored only at some sites), all sites would run the same distributed database-management software, and the sites would be aware of each other's existence.

Advantages of Distributed System

Performance: very often a collection of processors can provide higher performance than a centralized computer. Many tasks can be executed concurrently at different computers.
Distribution: many applications involve, by their nature, spatially separated machines (banking, commercial, automotive system).
Reliability (fault tolerance): If few computers fail others are available and hence the system continues.
Incremental growth: As requirements on processing power grow, new hardware and software resources can be added without replacing the existing resources.
Sharing of data/resources: Due to communication between connected computers resources can be shared among computers. Shared data is essential to many applications (banking, computer supported cooperative work, reservation systems); other resources can be also shared (e.g. expensive printers).

Disadvantage of Distributed System

Difficulties of developing distributed software: how should operating systems, programming languages and applications look like?
Networking problems: several problems are created by the network infrastructure, which have to be dealt with: loss of messages, overloading, ...
Security problems: sharing generates the problem of data security.

Difference between Centralized Database & Distributed database

Centralized Database	Distributed Database
• A type of database that contains a single database located at one location in the network.	• A type of database that contains two or more database files located at different location in the network.
• Managing, updating and taking in backups of data is easier because there is only one database file.	• As there are multiple database files in a distributed database it require time to synchronize data.
• Requires time for accessing data because multiple users access the database file.	• Speed in accessing the data is higher because the data is recieved from the nearest database file.
• If the database fails the users do not have access to a database.	• If one database fails the users can still access other database files.
• Has more data consistency and it provides the complete view to the user.	• Can have data replications and there can be some data inconsistency.

Design Issues / Characteristics of Distributed Systems

Design issues that arise specifically from the distributed nature of the application:

1. Transparency
2. Communication
3. Performance
4. Scalability
5. Heterogeneity
6. Openness
7. Reliability & fault tolerance
8. Security

1. Transparancy

Access transparency – Hiding how the resources are accessed.

Location transparency - users cannot tell where hardware and software resources (CPUs, files, data bases) are located; the name of the resource shouldn't encode the location of the resource.

Migration (mobility) transparency - resources should be free to move from one location to another without having their names changed.

Replication transparency - the system is free to make additional copies of files and other resources (for purpose of performance and/or reliability), without the users noticing.

Example: several copies of a file; at a certain request that copy is accessed which is the closest to the client.

Concurrency transparency - the users will not notice the existence of other users in the system (even if they access the same resources).

Failure transparency - applications should be able to complete their task despite failures occurring in certain components of the system.

Performance transparency - load variation should not lead to performance degradation.

Type of Transparency

Transparency	Description
Access	Hide how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource is replicated
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource

2. Communication:

Components of a distributed system have to communicate in order to interact. This implies support at two levels:

1. Networking infrastructure (interconnections & network software).
2. Appropriate communication primitives and models and their implementation:
    a. communication primitives:
        i. send
        ii. receive
        iii. remote procedure call (RPC)

b. communication models

i. Client-Server communication: implies a message exchange between two processes:

the process which requests a service and the one which provides it;

ii. Group Multicast: the target of a message is a set of processes, which are members of a given group.

3. Performance

Several factors are influencing the performance of a distributed system:
The performance of individual workstations.
The speed of the communication infrastructure.
Extent to which reliability (fault tolerance) is provided (replication and preservation of coherence imply large overheads).
Flexibility in workload allocation: for example, idle processors (workstations) could be allocated automatically to a user's task.

4. Scalability

The system should remain efficient even with a significant increase in the number of users and resources connected:

- Cost of adding resources should be reasonable;
- Performance loss with increased number of users and resources should be controlled;

5. Heterogeneity

Distributed applications are typically heterogeneous

- Different hardware: mainframes, workstations, PCs, servers, etc.;
- Different software: UNIX, MS-Windows, IBM OS/2, Real-time OSs, etc.;
- Unconventional devices: teller machines, telephone switches, robots, manufacturing systems
- Diverse networks and protocols: Ethernet, FDDI, ATM, TCP/IP, Novell Netware, etc.

An additional software layer called middleware used to mask heterogeneity.

6. Openness

One of the important features of distributed systems is openness and flexibility:

- Every service is equally accessible to every client (local or remote);
- It is easy to implement, install and debug new services;
- Users can write and install their own services.

Key aspect of openness:
Standard interfaces and protocols (like Internet communication protocols)
Support of heterogeneity (by adequate middleware, like CORBA)

7. Reliability and Fault Tolerance

One of the main goals of building distributed systems is improvement of reliability.
Availability: If machines go down, the system should work with the reduced amount of resources.
There should be a very small number of critical resources;
Critical resources: resources which have to be up in order the distributed system to work.
Key pieces of hardware and software (critical resources) should be replicated i.e. if one of them fails another one takes up - redundancy.
Data on the system must not be lost, and copies stored redundantly on different servers must be kept consistent.
The more copies kept, the better the availability, but keeping consistency becomes more difficult.

Fault-tolerance is a main issue related to reliability: the system has to detect faults and act in a reasonable way:
Mask the fault: continue to work with possibly reduced performance but without loss of data/ information.
Fail gracefully: react to the fault in a predictable way and possibly stop functionality for a short period, but without loss of data/information.
1)Node failure -Hardware or software failure.
2)Malicious Error-Caused by unauthorized Access.

8. Security
Security of information resources:
Confidentiality: Protection against disclosure to un-authorized person
Integrity: Protection against alteration and corruption
Availability: Keep the resource accessible

Characteristics of Distributed Systems
A Distributed System has the following characteristics:

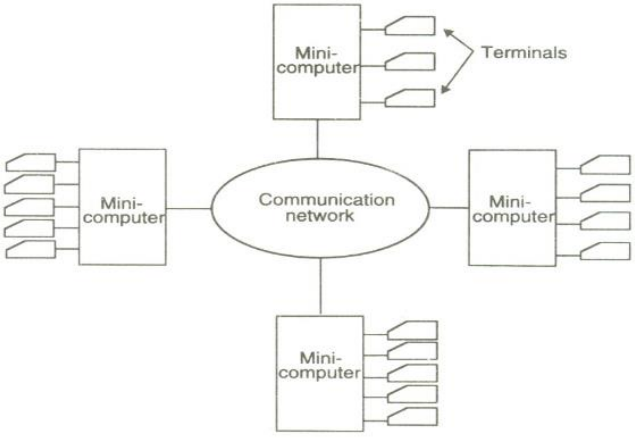
- It consists of several independent computers connected through communication network.
- The computers communicate with each other by exchanging message over a communication
- network.
- Each computer has its own memory, clock and runs its own operating system.
- Each computer has its own resources, called local resources.
- Remote resources are accessed through the network.

…….. Additionally Write, Design Issues with Distributed System.

System Architecture Types

Distributed systems can be modeled into several types. Various models are used for building distributed computing systems. These models can be broadly classified into following categories, described below:
1. Mini Computer Model,
2. Workstation Model,
3. Workstation Server Model,
4. Processor Pool Model,
5. Hybrid Model.

1. Mini Computer Model
In this model, the distributed system consists of several minicomputers. Each computer supports multiple users and provides access to remote resources. The ratio of processors to users is normally less than one.



Minicomputer model is a simple extension of the centralized time-sharing system. As shown in Figure, a distributed computing system based on this model consists of a few minicomputers.
They may be large supercomputers as well interconnected by a communication network. Each minicomputer usually has multiple users simultaneously logged on to it.
Several interactive terminals are connected to each minicomputer. Each user is logged on to one specific minicomputer, with remote access to other minicomputers.

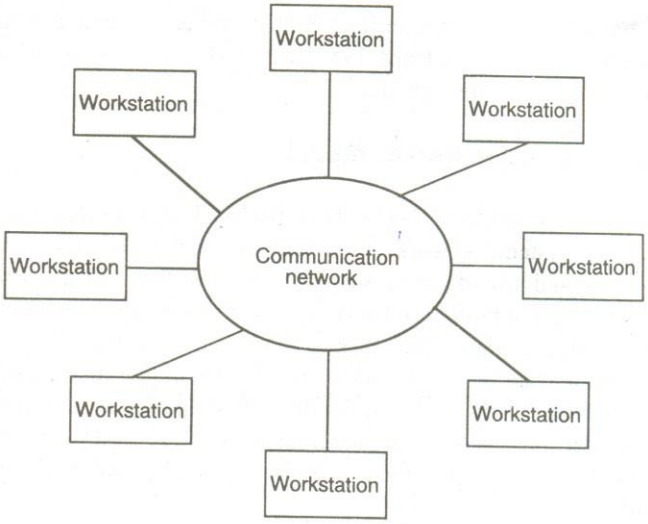
The minicomputer model may be used when resource sharing (such as sharing of information databases of different types, with each type of database located on a different machine) with remote users is desired.

2. Workstation Model

In this model, the distributed system consists of several workstations; every user has a workstation where user's work is performed. With the help of distributed file system, a user can access data regardless of the location of the data. The ratio of processors to users is normally one. The workstations are independent computers with memory, hard disks, keyboard and console. Workstations are connected with each other through communication network.

A company's office or a university department may have several workstations scattered throughout a building or compass each workstation equipped with its own disk and serving as a single-user computer.

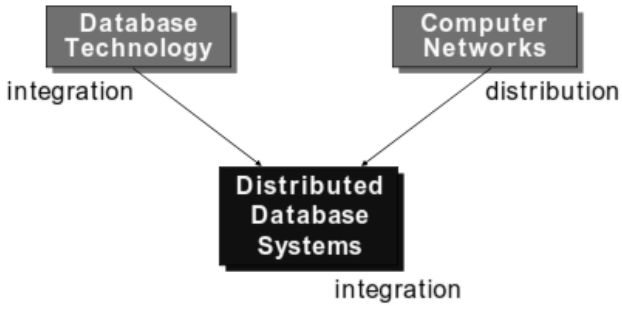
In this model a user logs onto one of the workstations called his or her *home workstation* and submits jobs for execution. When the system finds that the user's workstation does not have sufficient processing power for executing the processes of the submitted jobs efficiently, it transfers one or more of the processes from the user's workstation to some other workstation that is currently idle and gets the process executed there, and finally the result of execution is returned to the user's, in. workstation.



DISTRIBUTED DATABASE CONCEPTS

A distributed database is a collection of multiple interconnected databases, which are spread physically across various locations that communicate via a computer network.

Distributed database management system (DDBMS) is as a software system that manages a distributed database while making the distribution transparent to the user .



The sites may all be located in physically within the same building or a group of adjacent buildings—and connected via a local area network, or they may be geographically distributed over large distances and connected via wide area network. Local area networks typically use wireless hubs or cables, whereas long-haul or wide area networks use telephone lines, cables, wireless communication infrastructures or satellites.

Networks may have different topologies that define the direct communication paths among sites. The type and topology of the network used for distributed query processing and distributed database design.

Features

- Databases in the collection are logically interrelated with each other. Often they represent a single logical database.
- Data is physically stored across multiple sites. Data in each site can be managed by a DBMS independent of the other sites.
- The processors in the sites are connected via a network. They do not have any multiprocessor configuration.
- A distributed database is not a loosely connected file system.
- A distributed database incorporates transaction processing, but it is not synonymous with a transaction processing system.

Rules of Distributed database

Rule 1: Local Site Independence
Each local site can act as an independent, autonomous, centralized DBMS. Each site is responsible for security, concurrency control, backup, and recovery.

Rule 2: Central Site Independence
No site in the network relies on a central site or any other site. All sites have the same capabilities.

Rule 3: Failure Independence
The system is not affected by node failures.The system is in continuous operation even in the case of a node failure or an expansion of the network.

Rule 4: Location Transparency
The user does not need to know the location of the data in order to retrieve those data.

Rule 5: Fragmentation Transparency
The user sees only one single logical database. Data fragmentation is transparent to the user. The user does not need to know the name of the database fragments in order to retrieve them.

Rule 6: Replication Transparency
The user sees only one single logical database. The DDBMS transparently selects the database fragment to access.The DDBMS manages all fragments transparently to the user

Rule 7: Distributed Query Processing
A distributed query may be executed at several different DP sites. Query optimization is performed transparently by the DDBMS.

Rule 8: Distributed Transaction Processing
A transaction may update data at several different sites. The transaction is transparently executed at several different DP sites.

Rule 9: Hardware Independence
The system must run on any hardware platform.

Rule 1O: Operating System Independence
The system must run on any operating system software platform.

Rule 11: Network Independence
The system must run on any network platform.

Rule 12: Database Independence
The system must support any vendor's database product.

Distributed Database Management System

A distributed database management system (DDBMS) is a centralized software system that manages a distributed database in a manner as if it were all stored in a single location.

Features

- It is used to create, retrieve, update and delete distributed databases.
- It synchronizes the database periodically and provides access mechanisms by the virtue of which the distribution becomes transparent to the users.
- It ensures that the data modified at any site is universally updated.
- It is used in application areas where large volumes of data are processed and accessed by numerous users simultaneously.
- It is designed for heterogeneous database platforms.
- It maintains confidentiality and data integrity of the databases.

Factors Encouraging DDBMS

The following factors encourage moving over to DDBMS −

- **Distributed Nature of Organizational Units** − Most organizations in the current times are subdivided into multiple units that are physically distributed over the globe. Each unit requires its own set of local data. Thus, the overall database of the organization becomes distributed.
- **Need for Sharing of Data** − The multiple organizational units often need to communicate with each other and share their data and resources. This demands common databases or replicated databases that should be used in a synchronized manner.
- **Support for Both OLTP and OLAP** − Online Transaction Processing (OLTP) and Online Analytical Processing (OLAP) work upon diversified systems which may have common data. Distributed database systems aid both these processing by providing synchronized data.
- **Database Recovery** − One of the common techniques used in DDBMS is replication of data across different sites. Replication of data automatically helps in data recovery if database in any site is damaged. Users can access data from other sites while the damaged site is being reconstructed. Thus, database failure may become almost inconspicuous to users.

- **Support for Multiple Application Software** − Most organizations use a variety of application software each with its specific database support. DDBMS provides a uniform functionality for using the same data among different platforms.

Components of Distributed Data Base Management System:

*Computer workstations (sites or nodes) that form the network system. The distributed database system must be independent of the computer system hardware.
• Network hardware and software components that reside in each workstation. The network components allow all sites to interact and exchange data. Network system independence is a desirable distributed database system attribute.
*Communications media that carry the data from one workstation to another.
*The DDBMS must be communications media-independent; that is, it must be able to support several types of communications media.
*The transaction processor (TP), which is the software component found in each computer that requests data. The transaction processor receives and processes the application's data requests (remote and local). The TP is also known as the application processor (AP) or the transaction manager (TM).
*The data processor (DP), which is the software component residing on each computer that stores and retrieves data located at the site. The DP is also known as the data manager (DM). A data processor may even be a centralized DBMS.

The protocols determine how the distributed database system will:
Interface with the network to transport data and commands between data processors (DPs) and transaction processors (TPs).
• Synchronize all data received from DPs (TP side) and route retrieved data to the appropriate TPs (DP side).
Ensure common database functions in a distributed system. Such functions include security, Concurrency control, backup, and recovery.

Advantages of Distributed Databases

Following are the advantages of distributed databases over centralized databases.

Improved ease and flexibility of application development. Developing and maintaining applications at geographically distributed sites of an organization is facilitated owing to transparency of data distribution and control.

Modular Development − If the system needs to be expanded to new locations or new units, in centralized database systems, adding new computers and local data to the new site and finally connecting them to the distributed system, with no interruption in current functions.

More Reliable − In case of database failures, the total system of centralized databases comes to a halt. However, in distributed systems, when a component fails, the functioning of the system continues may be at a reduced performance. Hence DDBMS is more reliable.

Better Response − A distributed DBMS fragments the database by keeping the data closer to where it is needed most. Data localization reduces the contention for CPU and I/O services and simultaneously reduces access delays involved in wide area networks.

Lower Communication Cost − In distributed database systems, if data is located locally where it is mostly used, then the communication costs for data manipulation can be minimized. This is not feasible in centralized systems.

Additional Functions of Distributed Databases

Keeping track of data distribution. The ability to keep track of the data distribution, fragmentation, and replication by expanding the DDBMS catalog.

Distributed query processing. The ability to access remote sites and transmit Queries and data among the various sites via a communication network.

Distributed transaction management. The ability to devise execution strategies for queries and transactions that access data from more than one site and to synchronize the access to distributed data and maintain the integrity of the overall database.

Replicated data management. The ability to decide which copy of a replicated data item to access and to maintain the consistency of copies of a replicated data item.

Distributed database recovery. The ability to recover from individual site crashes and from new types of failures, such as the failure of communication links.

Security. Distributed transactions must be executed with the proper management Of the security of the data and the authorization/access privileges of users.

Distributed directory (catalog) management. A directory contains information (metadata) about data in the database.

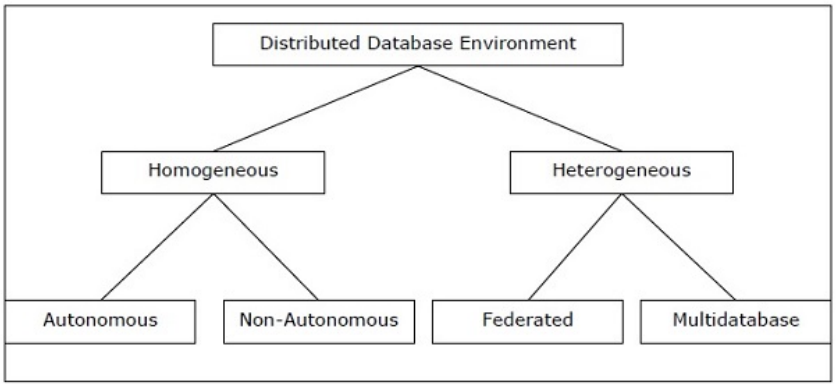
The directory may be global for the entire DDB, or local for each site. The placement and distribution of the directory are design and policy issues.

Disadvantages of Distributed Database System

- The distributed database is quite complex and it is difficult to make sure that a user gets a uniform view of the database because it is spread across multiple locations.
- It is difficult to provide security in a distributed database as the database needs to be secured at all the locations it is stored. Moreover, the infrastructure connecting all the nodes in a distributed database also needs to be secured.
- It is difficult to maintain data integrity in the distributed database because of its nature. There can also be data redundancy in the database as it is stored at multiple locations.

Types of Distributed Databases

Distributed databases can be broadly classified into homogeneous and heterogeneous distributed database environments, each with further sub-divisions, as shown in the following illustration.

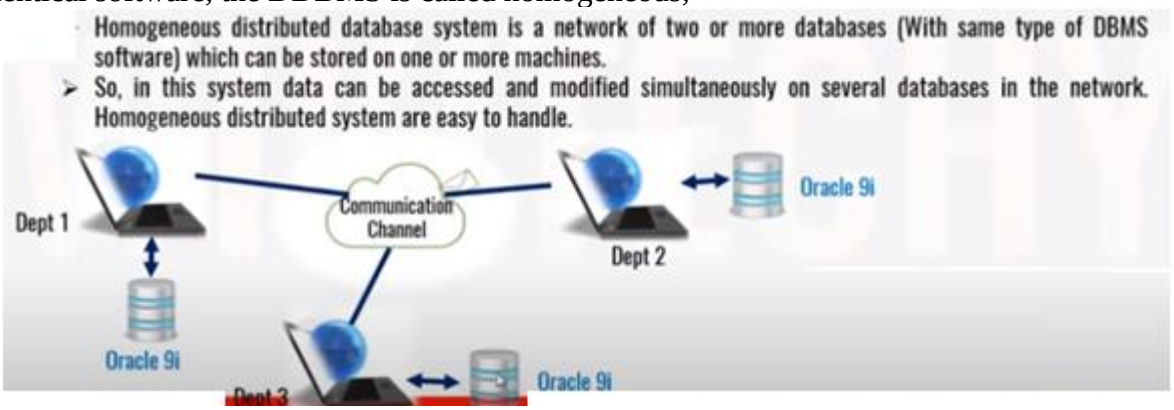


Homogeneous Distributed Databases

In a homogeneous distributed database, all the sites use identical DBMS software and Operating systems and same data model.

These database management systems are much easier to handle and the database can even be scaled if required.

If all servers (or individual local DBMSs) use identical software and all users (clients) use identical software, the DDBMS is called homogeneous;



Its Properties / Advantages are −

- The sites use very similar software.
- Much easier to design and manage.
- The sites use identical DBMS or DBMS from the same vendor.
- Each site is aware of all other sites and cooperates with other sites to process user requests.
- The database is accessed through a single interface as if it is a single database.

Disadvantages:

Difficult for most organizations to force a homogenous environment.

Types of Homogeneous Distributed Database

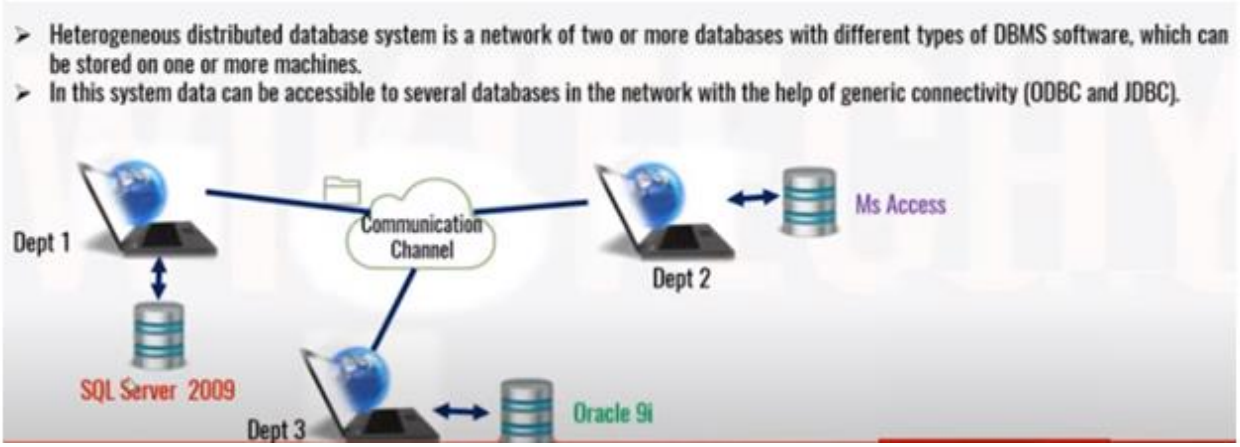
There are two types of homogeneous distributed database −

- **Autonomous** − Each database is independent that functions on its own. They are integrated by a controlling application and use message passing to share data updates.
- **Non-autonomous** − Data is distributed across the homogeneous nodes and a central or master DBMS co-ordinates data updates across the sites.

Heterogeneous Distributed Databases

In a heterogeneous distributed database, different sites have different operating systems, DBMS products and data models. Its properties are −

- Different sites use dissimilar schemas and software.
- The system may be composed of a variety of DBMSs like relational, network, hierarchical or object oriented.
- Query processing is complex due to dissimilar schemas.
- Transaction processing is complex due to dissimilar software.
- A site may not be aware of other sites and so there is limited co-operation in processing user requests.



Types of Heterogeneous Distributed Databases

- **Federated** − The heterogeneous database systems are independent in nature and integrated together so that they function as a single database system.
- **Un-federated** − The database systems employ a central coordinating module through which the databases are accessed.

In this type of database, different data center may run different DBMS products with possibly different underlying data models.
Occurs when sites have implemented their own database and integration is considered later.

Advantages :
Huge data can be stored in one global center from different databases center.
Remote access is done using the global schema.
Different DBMS may be used at each node.

Disadvantage
1. Difficult to design and manage.

Students, Heterogenous DBMS for understanding purpose

Platform	DBMS	Operating System	Network Communication Protocol
IBM 3090	DB2	MVS	APPC LU 6.2
DEC VAX	VAX Rdb	VMS	DECnet
IBM AS/400	SQL/400	OS/400	3270
RISC computer	Informix	Unix	TCP/IP
Pentium CPU	ORACLE	OS/2	NETBIOS

DISTRIBUTED DATA STORAGE

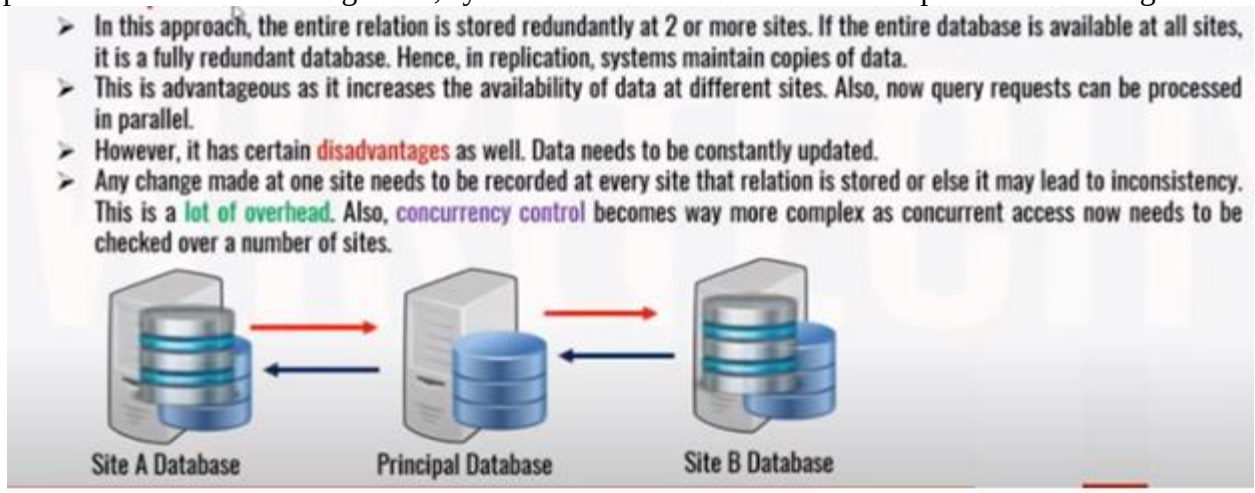
A distributed DBMS manages the distributed database in a manner so that it appears as on one single database to users.

A distributed database is a collection of multiple interconnected databases, which are spread physically across various locations that communicate via computer network.

1. Data Replication

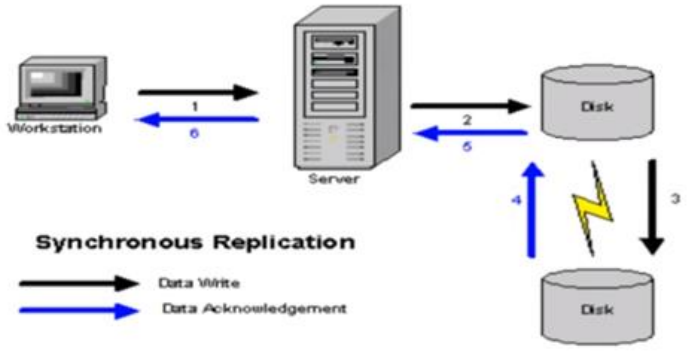
Data Replication is the process of storing data in more than one site or node for faster retrieval and fault tolerance.
It is useful in improving the availability of data.
It is simply copying data from one server to another server so that all user's can share the same data without any inconsistency.
User's can access data relevant to their tasks without interfering of other's tasks.

Replication & Fragmentation – Replication and Fragmentation can be combined . Relation is partitioned into several fragments, system maintains several identical replicas of each fragment.



➢Synchronous Replication:

- In synchronous replication, the replica will be modified immediately after some changes are made in the relation table.
- So there is no difference between original data and replica.



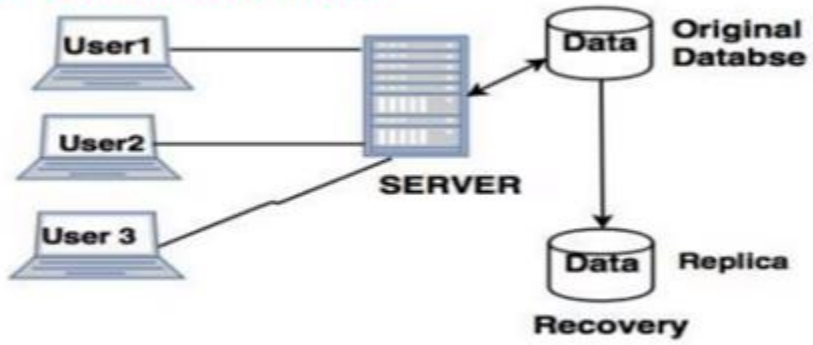
➢Asynchronous Replication:

- In asynchronous replication, the replica will be modified after commit is fired on to the database.

Replication Schemes:

1. Full Replication:

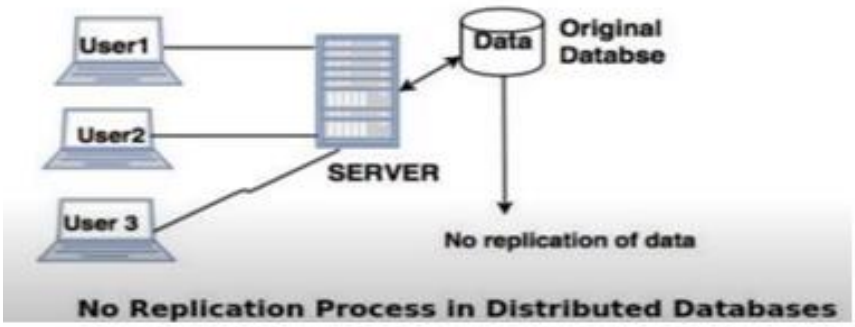
In full replication scheme, the database is available to almost every location or user in communication network.



Full Replication Process In Distributed System

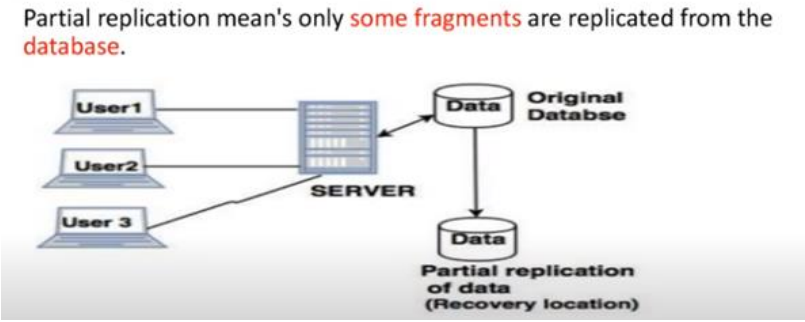
2. No Replication:

No replication means, each fragment is stored exactly at one location.



No Replication Process in Distributed Databases

3. Partial Replication



Advantages of Repliation

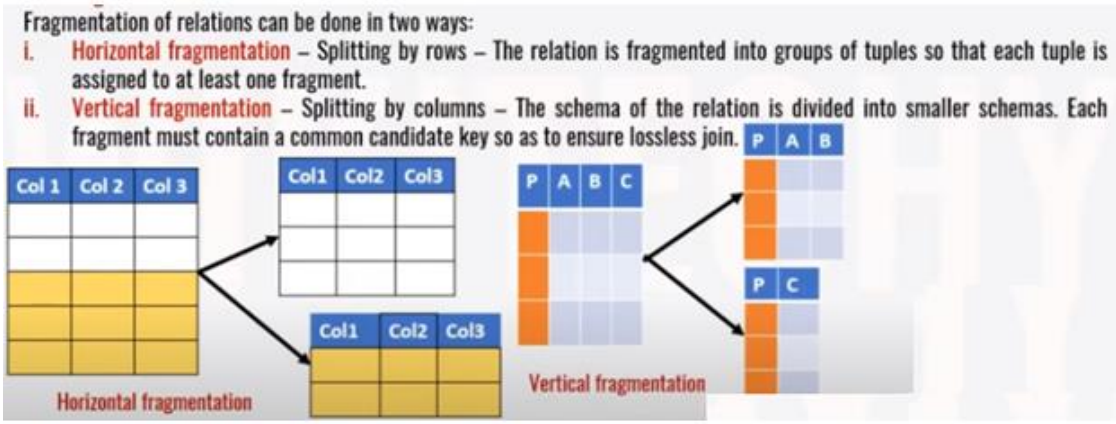
- **Reliability** − In case of failure of any site, the database system continues to work since a copy is available at another site(s).
- **Reduction in Network Load** /Data Transfer− Since local copies of data are available, query processing can be done with reduced network usage, particularly during prime hours. Data updating can be done at non-prime hours.
- **Quicker Response** − Availability of local copies of data ensures quick query processing and consequently quick response time.
- **Simpler Transactions** − Transactions require less number of joins of tables located at different sites and minimal coordination across the network. Thus, they become simpler in nature.
- Data Replication supports multiple user's and gives high performance.
- To perform faster execution of queries.

Disadvantages of Data Replication

- **Increased Storage Requirements** − Maintaining multiple copies of data is associated with increased storage costs. It consumes more space.
- **Increased Cost and Complexity of Data Updating** − Each time a data item is updated, the update needs to be reflected in all the copies of the data at the different sites. This requires complex synchronization techniques and protocols.
-

2. Fragmentation (Sharding)

Fragmentation is the task of dividing a table into a set of smaller tables. The subsets of the table are called **fragments** and each of the fragment is stored in different sites when they are required.
Fragmentation can be of three types: horizontal, vertical, and hybrid (combination of horizontal and vertical).
Fragments must be used to reconstruct the original relation (database).
Fragmentation does not create copies of data, consistency is not a problem.



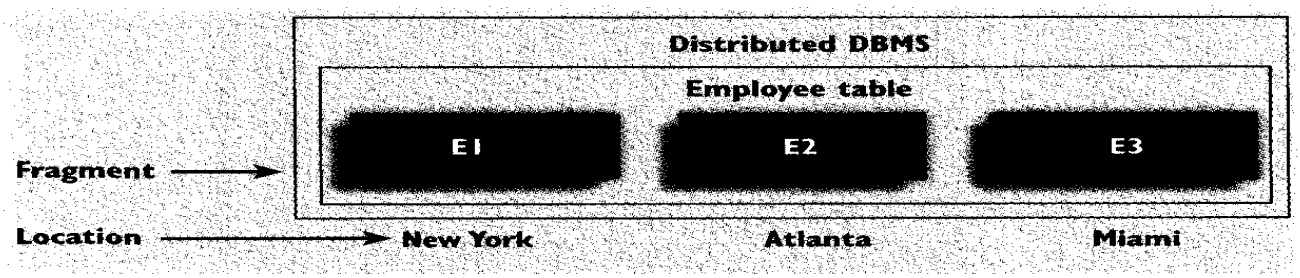
In **horizontal fragmentation**, a relation r is partitioned into a number of subsets, $r_1, r_2, \ldots, r_n$. Each tuple of relation r must belong to at least one of the fragments, so that the original relation can be reconstructed, if needed.

As an illustration, the *account* relation can be divided into several different fragments, each of which consists of tuples of accounts belonging to a particular branch. If the banking system has only two branches—Hillside and Valleyview—then there are two different fragments:

$$account_1 = \sigma_{branch\_name = \text{"Hillside"}}(account)$$
$$account_2 = \sigma_{branch\_name = \text{"Valleyview"}}(account)$$

Horizontal fragmentation is usually used to keep tuples at the sites where they are used the most, to minimize data transfer.

To illustrate the use of various transparency levels, let us suppose we have an EMPLOYEE table containing the attributes EMP_NAME, EMP_DOB, EMP_ADDRESS, EMP_DEPARTMENT, and EMP_SALARY The EMPLOYEE data are distributed over three different locations: New York, Atlanta, and Miami. The table is divided by location; that is, all New York employee data are stored in fragment El, Atlanta employee data are stored in fragment E2, and Miami employee data are stored in fragment E3.



Case 1: The Database Supports Fragmentation Transparency

The query conforms to a nondistributed database query format; that is, it does not specify fragment names or locations. The query reads:

SELECT * FROM EMPLOYEE WHERE EMP_DOB < 01/01/40;

Case 2: The Database Supports Location Transparency

Fragment names must be specified in the query, but fragment location is not specified. The query reads:

```
SELECT * FROM EI WHERE EMP_DOB < 01/01/40;
UNION
SELECT * FROM E2 WHERE EMP_DOB < 01/01/40;
UNION
SELECT * FROM E3 WHERE EMP_DOB < 01/01/40;
```

Case 3: The Database Supports Local Mapping Transparency

Both the fragment name and location must be specified in the query. Using pseudo-SQL:

```
SELECT * FROM El  NODE  NY  WHERE EMP_DOB < 01/01/40;
UNION
SELECT * FROM E2  NODE ATL WHERE EMP_DOB < 01/01/40;
UNION
SELECT * FROM E3 NODE MIA WHERE EMP_DOB < 01/01/40;
```

NODE indicates the .location of the database fragment.

For 2nd example, in the student schema, if the details of all students of Computer Science Course needs to be maintained at the School of Computer Science, then the designer will horizontally fragment the database as follows −

```
CREATE COMP_STD AS
  SELECT * FROM STUDENT
  WHERE COURSE = "Computer Science";
```

Vertical Fragmentation

In vertical fragmentation, the fields or columns of a table are grouped into fragments. Vertical Fragmentation. Each site may not need all the attributes of a relation, which would indicate the need for a different type of fragmentation.

Vertical fragmentation divides a relation "vertically" by columns. A vertical fragment of a relation keeps only certain attributes of the relation. For example, we may want to fragment the EMPLOYEE relation into two vertical fragments.

The first vertical fragment
includes personal information—
Empno, Name, Bdate, Address, and Gender—and the second fragment
includes work-related information—
Empno, Salary, DA, HRA, Netpay

In Second example, let us consider that a University database keeps records of all registered students in a Student table having the following schema.

STUDENT

Regd_No	Name	Course	Address	Semester	Fees	Marks

Now, the fees details are maintained in the accounts section. In this case, the designer will fragment the database as follows −

```
CREATE TABLE STD_FEES AS
  SELECT Regd_No, Fees
  FROM STUDENT;
```

Mixed (Hybrid) Fragmentation. In hybrid fragmentation, a combination of horizontal and vertical fragmentation techniques are used. This is the most flexible fragmentation technique since it generates fragments with minimal extraneous information. However, reconstruction of the original table is often an expensive task.

Advantages of Fragmentation:

Horizontal :

Allows parallel processing on fragments of a relation.

Since data is stored close to the site of usage, efficiency of the database system is increased.

Vertical :

Allows tuples to be split up so that each part of the tuple is stored where it is most frequently accessed.

Tuple-id attribute allows efficient joining of vertical fragments.

Allows parallel processing on a relation.

- Local query optimization techniques are sufficient for most queries since data is locally available.
- Since irrelevant data is not available at the sites, security and privacy of the database system can be maintained.

Disadvantages of Fragmentation

- When data from different fragments are required, the access speeds may be very high.
- In case of recursive fragmentations, the job of reconstruction will need expensive techniques.
- Lack of back-up copies of data in different sites may render the database ineffective in case of failure of a site.

Distribution Transparency

Distribution transparency is the property of distributed databases by the virtue of which the internal details of the distribution are hidden from the users. The DDBMS designer may choose to fragment tables, replicate the fragments and store them at different sites. However, since users are oblivious of these details, they find the distributed database easy to use like any centralized database.

The three dimensions of distribution transparency are −

- Location transparency
- Fragmentation transparency
- Replication transparency

Location Transparency

Location transparency ensures that the user can query on any table(s) or fragment(s) of a table as if they were stored locally in the user's site. The fact that the table or its fragments are stored at remote site in the distributed database system, should be completely oblivious to the end user. The address of the remote site(s) and the access mechanisms are completely hidden.

Users cannot tell where hardware and software resources (CPUs, files, data bases) are located; the name of the resource shouldn't encode the location of the resource.

Fragmentation Transparency

Fragmentation transparency enables users to query upon any table as if it were unfragmented. Thus, it hides the fact that the table the user is querying on is actually a fragment or union of some fragments. It also conceals the fact that the fragments are located at diverse sites.

This is somewhat similar to users of SQL views, where the user may not know that they are using a view of a table instead of the table itself.

Replication Transparency

Replication transparency ensures that replication of databases are hidden from the users. It enables users to query upon a table as if only a single copy of the table exists.

The system is free to make additional copies of files and other resources (for purpose of performance and/or reliability), without the users noticing.
Example: several copies of a file; at a certain request that copy is accessed which is the closest to the client.

Replication transparency is associated with concurrency transparency and failure transparency. Whenever a user updates a data item, the update is reflected in all the copies of the table. However, this operation should not be known to the user. This is concurrency transparency. Also, in case of failure of a site, the user can still proceed with his queries using replicated copies without any knowledge of failure. This is failure transparency.

Transaction Transparency

Transaction transparency is a DDBMS property that ensures that database transactions will maintain the distributed database's integrity and consistency. Transaction transparency ensures that the transaction will be completed only if all database sites involved in the transaction complete their part of the transaction.

Distributed database systems require complex mechanisms to manage transactions and ensure the database's consistency and integrity.

Combination of Transparencies

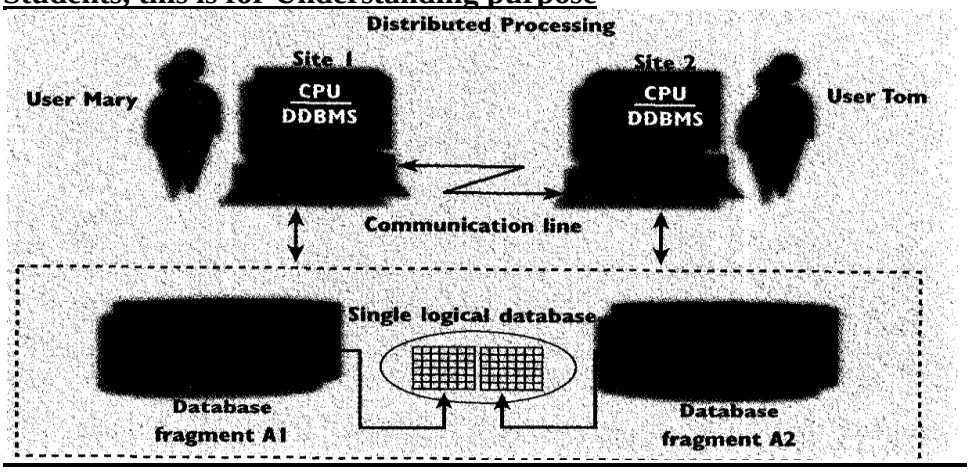
In any distributed database system, the designer should ensure that all the stated transparencies are maintained to a considerable extent. The designer may choose to fragment tables, replicate them and store them at different sites; all oblivious to the end user. However, complete distribution transparency is a tough task and requires considerable design efforts.

Access transparency – Hiding how the resources are accessed.
Concurrency transparency - the users will not notice the existence of other users in the system (even if they access the same resources).
Failure transparency - applications should be able to complete their task despite failures occurring in certain components of the system.
Performance transparency - load variation should not lead to performance degradation.

Type of Transparency

Transparency	Description
Access	Hide how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource is replicated
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource

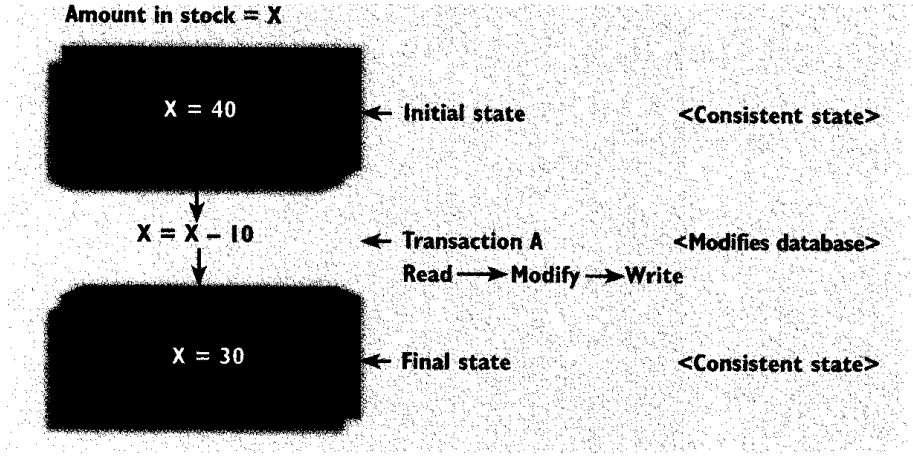
Students, this is for Understanding purpose



The single logical database consists of two database fragments AI and A2, located at sites I and 2, respectively. Mary can query the database as if it were a local database; and so can Tom. Both users "see" only one logical database and do not need to know the names of the fragments. In fact, the end users need not even know that the database is divided into separate fragments, nor do they need to know where the fragments are located.

TRANSACTIONS

A transaction is a logical unit of work that must be either entirely completed or aborted. A transaction that changes the contents of the database must alter the database from one consistent state to another. A consistent database state is one in which all data integrity constraints are satisfied.



To ensure consistency of the database, every transaction must begin with the database in a known consistent state. If the database is not in a consistent state, the transaction will yield an inconsistent database that violates its integrity and business rules.

Once a transaction has been guaranteed access to the database, it may manipulate the database contents. Using Figure of example, the data item X must be read from permanent storage to primary memory;
the data value is then updated by subtracting 10;
the new value of X is written back from primary memory to permanent storage.

As this transaction is taking place, the DBA/IS must ensure that no other transaction accesses X. When the transaction is concluded, all data used by the transaction are released to give subsequent transactions data access. The database is returned to a consistent state.

Example:
An example illustrates a more complex set of transactions. Suppose that an accountant wishes to register the credit sale of 100 units of product X to customer Y in the amount of $500.00.The required transactions can be completed by;

- Reducing product X's Quantity on hand (QOH) by 100.
- Adding $500.00 to customer Y's accounts receivable.

These 2 SQL Commands to be completed for above transactions correctly

```
UPDATE PRODUCT SET QOH = QOH - 100
        WHERE PROD_CODE = 'X';

UPDATE  BANK  SET BALANCE = BALANCE + 500
        WHERE ACC_NUM = 'Y';
```

But, if both transactions are not completely executed, the transaction yields an inconsistent database.
Now suppose that the DBMS completes the first transaction by updating the quantity on hand for product X Then, during the execution of the second part of the transaction (the UPDATE of the accounts receivable table),the computer system experiences a loss of electrical power.
If the computer does not have a backup power supply, the second transaction cannot be completed. Therefore, while the PRODUCT table was updated to represent the sale of product X, customerY was not charged.
The database is now in an inconsistent state, and it is not usable for subsequent transactions. The DBA/IS must be able to recover the database to a previous consistent state.

Transaction Properties:

Atomicity :
Requires that all operations of a transaction be completed; if not, the transaction is aborted.

Durability: Once DBMS informs the user that a transaction has been successfully completed, its effect should persist even if system crashes.

Serializability :
Describes the result of the concurrent execution of several transactions. More specifically, the concurrent transactions are treated as though they were executed in serial order (one after another).This property is important in multi-user and distributed databases, where several transactions are likely to be executed concurrently.

Isolation:
Means that the data used during the execution of a transaction cannot be used by a second transaction until the first one is completed. In other words, if a transaction Tl is being executed and is using the data item X, that data item cannot be accessed by any other transaction (T2...Tn) until TI ends.This property is particularly useful in multi-user database environments because several different users can access and update the database at the same time.

Transaction Management with SQL
Transaction support is provided by two SQL statements:

COMMIT and ROLLBACK. When a transaction sequence is initiated by a user or an application program, it must continue through all succeeding SQL statements until one of the following four events occurs:

1. A COMMIT statement is reached, in which case all changes are permanently recorded within the database.The COMMIT statement automatically ends the SQL transaction.
2. A ROLLBACK statement is reached, in which case all the changes are aborted and the database is rolled back to its previous consistent state.
3. The end of a program is successfully reached, in which case all changes are permanently recorded within the database.This action is equivalent to COMMIT
4. The program is abnormally terminated, in which case the changes made in the database are aborted and the database is rolled back to its previous consistent state.This action is equivalent to ROLLBACK.

Example of Commit:
The use of COMMIT is illustrated in the following sales sequence, which updates a product's quantity on hand (PROD_QOHJ and the customer's accounts receivable:

```
UPDATE PRODUCT SET QOH = QOH - 100
        WHERE PROD_CODE = 'X';

UPDATE  BANK  SET BALANCE = BALANCE + 500
        WHERE ACC_NUM = ‘Y’;

COMMIT;
```

The Transaction Log
A DBMS uses a transaction log to keep track of all transactions that update the database. The information stored in this log is used by the DBMS for a recovery requirement triggered by a ROLLBACK statement, a program's abnormal termination, or a system failure such as a network discrepancy or a disk crash. Some RDBMSs use the transaction log to recover a database forward to a currently consistent state.
After a server failure, for example, ORACLE automatically rolls back uncommitted transactions and rolls forward transactions that were committed but not yet written to the physical database.

While the DBMS executes transactions that modify the database, it also automatically updates the transaction log.The transaction log stores before-and-after data about the database and any of the tables, rows, and attribute values that participated in the transaction. The beginning and ending (COMMIT) of the transaction are also recorded. Although using a transaction log increases the processing overhead of a DBMS, the ability to restore a corrupted database is worth the price.

TID	TABLE	ROW ID	ATTRIBUTE	BEFORE	AFTER
101	**** Start Transaction				
101	PRODUCT	345TYX	PROD_QOH	243	143
101	ACCREC	60120010	AR_BALANCE	1200	4700
101	**** End Transaction : COMMITTED				

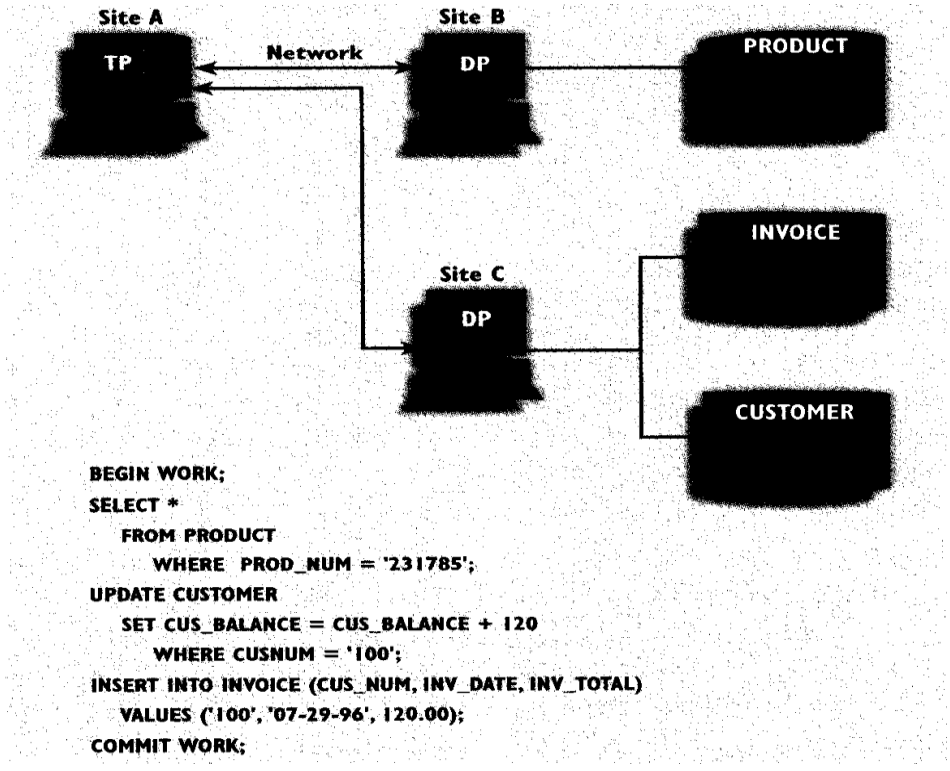
Transaction ID automatically assigned by the DBMS

If a ROLLBACK is issued before the termination of a transaction, the DBMS will restore the database only for that particular transaction, rather than for all transactions, in order to maintain the durability of the previous transactions. In other words, committed transactions are not rolled back.

The transaction log is itself a database, and it is managed by the DBMS like any other database. The transaction log is subject to such common database dangers as disk-full conditions and disk crashes.

Distributed Transaction

A distributed transaction allows a transaction to reference several different (local or remote) DP sites. Although each single request can reference only one remote DP site, the transaction as a whole can reference multiple DP sites because each request can reference a different site.



DISTRIBUTED REQUEST

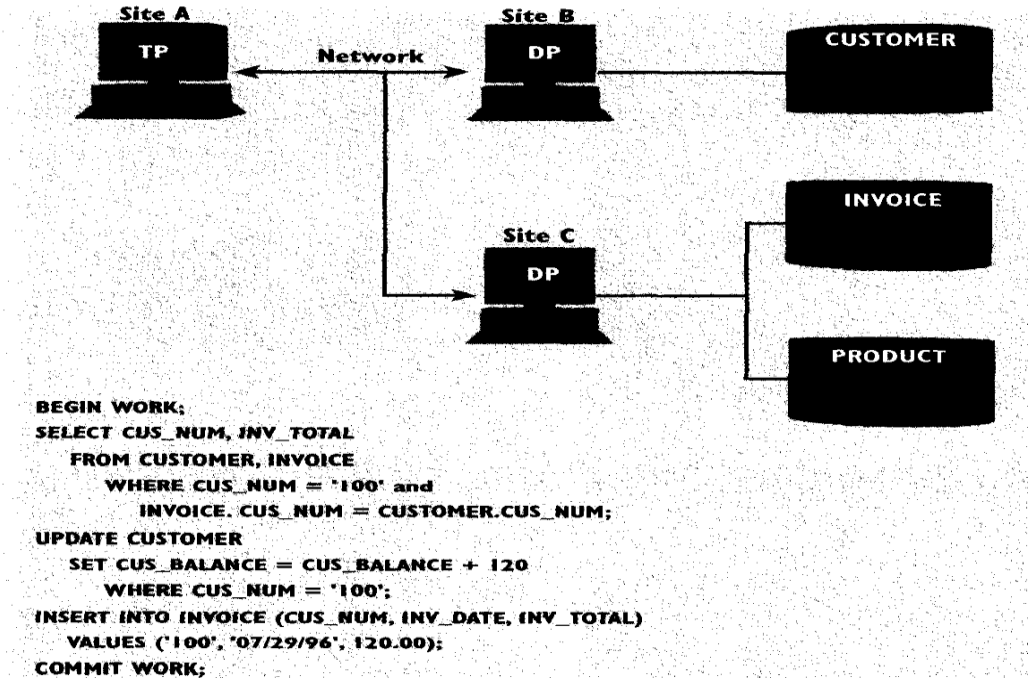
A distributed request reference data from several remote DP sites.

Because each request can access data from more than one DP site, a transaction can access several sites.

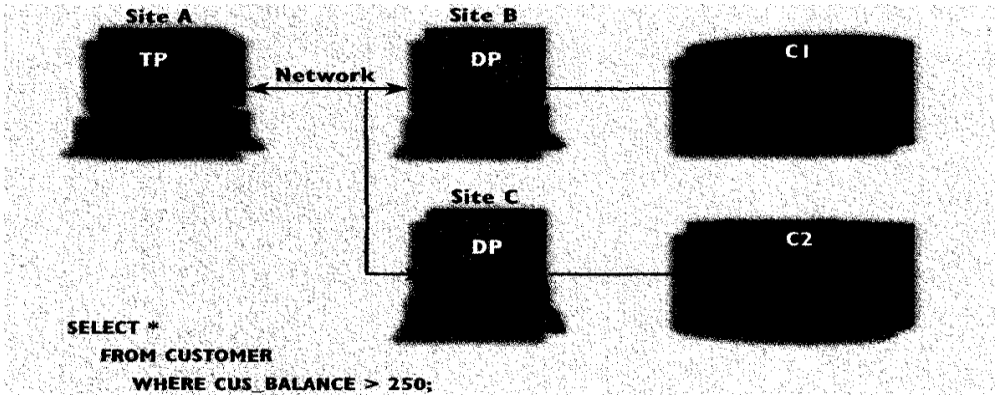
The ability to execute a distributed request provides fully distributed database processing capabilities because we are able to:

1. Partition a database table into several fragments.
2. Reference one or more of those fragments with only one request. In other words, we have fragmentation transparency.

The location and partition of the data should be transparent to the end user.



The distributed request feature also allows a single request to reference a physically partitioned table. For example, suppose that a CUSTOMER table is divided into two fragments, CI and C2, located at sites B and C, respectively. Let us further suppose that the end user wants to obtain a list of all customers whose balances exceed $250.00.



UNIT IV XML AND DATAWAREHOUSE

XML Database: XML – XML Schema – XML DOM and SAX Parsers – XSL – XSLT – XPath and XQuery – Data Warehouse: Introduction – Multidimensional Data Modeling – Star and

What is a **Data Warehouse**?

A single, complete and consistent store of data obtained from a variety of different sources made available to end users in a what they can understand and use in a business context.

- Data should be integrated across the enterprise
- Summary data has a real value to the organization
- Historical data holds the key to understanding data over time
- What-if capabilities are required

A process of transforming data into information and making it available to users in a timely.

A data warehouse is a large centralized repository of data that contains information from many sources within an organization. The collated data is used to guide business decisions through analysis, reporting, and data mining tools.

Definition : "A warehouse is a subject-oriented, integrated, time-variant and non-volatile collection of data in support of management's decision making process".

Subject Oriented: Data that gives information about a particular subject instead of about a company's ongoing operations.

Integrated: Data that is gathered into the data warehouse from a variety of sources and merged into a coherent whole.

Time-variant: All data in the data warehouse is identified with a particular time period.

Non-volatile: Data is stable in a data warehouse. More data is added but data is never removed.

Characteristics of data warehousing

1) A data warehouse can be viewed as an information system with the following attributes:
   - It is a database designed for analytical tasks.
   - It's content is periodically updated.
   - It contains current and historical data to provide a historical perspective of information.
2) It is separate from operational database.
3) It integrates from heterogeneous systems.
4) It stores huge amount of data, more historical than current data.
5) Does not require data to be highly accurate.
6) Queries are generally complex.
7) Goal is to execute statistical queries and provide results which can influence decision making in favor of the Enterprise.
8) These systems are thus called Online Analytical Processing Systems (OLAP).

Benefits of data warehousing

- Data warehouses are designed to perform well with aggregate queries running on large amounts of data.
- The structure of data warehouses is easier for end users to navigate, understand and query against unlike the relational databases primarily designed to handle lots of transactions.
- Data warehouses enable queries that cut across different segments of a company's operation. E.g. production data could be compared against inventory data even if they were originally stored in different databases with different structures.
- Queries that would be complex in very normalized databases could be easier to build and maintain in data warehouses, decreasing the workload on transaction systems.
- Data warehousing is an efficient way to manage and report on data that is from a variety of sources, non uniform and scattered throughout a company.
- It is an efficient way to manage demand for lots of information from lots of users.
- Data warehousing provides the capability to analyze large amounts of historical data for that can provide an organization with competitive advantage.
- Data warehouses facilitate decision support system applications such as trend reports, exception reports, and reports that show actual performance versus goals

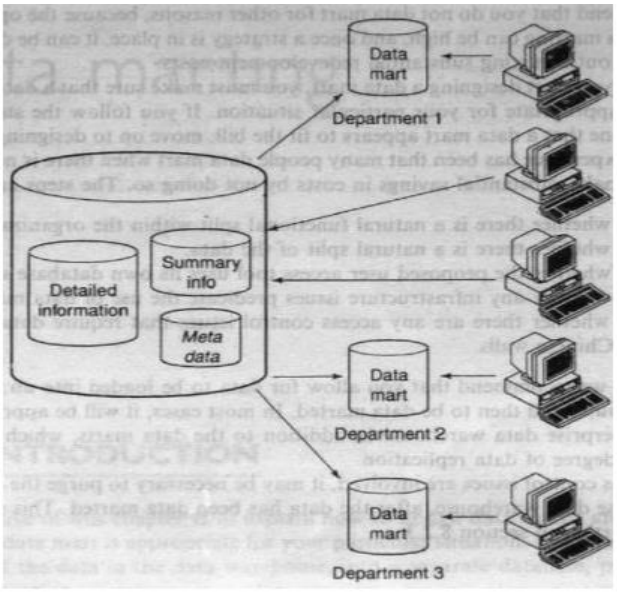
Very Large Data Bases

- Terabytes -- Walmart -- 24 Terabytes
- Petabytes -- Geographic Information Systems
- Exabytes -- National Medical Records
- Zettabytes - Weather images
- Zottabytes - Intelligence Agency Videos

Data Mart

A data mart is a subset of a data warehouse oriented to a specific business line. Data marts contain repositories of summarized data collected for analysis on a specific section or unit within an organization, for example, the sales department.

- Data mart is a subset of the information content of a data warehouse that is stored in its own database, summarized or in detail.
- Data marting can improve query performance, simply by reducing the volume of data.



When is a data mart appropriate

Data marts are created for the following reasons

- to speed up queries by reducing the volume of data to be scanned;
- to structure data in a form suitable for a user access tool;
- to partition data in order to impose access control strategies;
- to segment data into different hardware platforms

Identify Functional Splits

- we must determine whether the business is structured in such a way as to benefit from functionally splitting the data
- Look for departmental splits, and then determine whether the way in which the departments use information tends to be in isolation from the rest of the organization.

A data-marting strategy in the retail sector

Example

- a retail organization in which each merchant is responsible for maximizing the sales of a group of products
- information in the data warehouse
  - sales transactions on a daily level, to monitor actual sales;
  - sales forecasts on a weekly basis;
  - stock positions on a daily basis, to monitor stock levels
  - stock movements on a daily basis, to monitor supplier or shrinkage issues
- the merchant is not interested in products that he or she is not responsible
- the merchant is extremely unlikely to query information about other products

Identify User Access Tool Requirements

- Data marts are required in order to support any user access tools that require internal data structures
- Data within those structures is outside the control of the data warehouse . they need to be populated and updated on a regular basis.

Sno	Feature	DATA MART	DATA WAREHOUSE
1	Focus	A single subject or functional organization area.	Enterprise-wide repository of disparate data sources.
2	Data Sources	Relatively few sources linked to one line of business.	Many external and internal sources from different areas of an organization.
3	Size	Less than 100 GB	100 GB minimum but often in the range of terabytes for large organizations.
4	Normalization	No preference between a normalized and denormalized structure.	Modern warehouses are mostly denormalized for quicker data querying and read performance.
5	Decision Types	Tactical decisions pertaining to particular business lines and ways of doing things.	Strategic decisions that affect the entire enterprise.
6	Setup Time	3-6 months	At least a year for on-premise warehouses; cloud data warehouses are much quicker to set up.

7	Data Held	Typically summarized data.	Raw data, metadata, and summary data.

1.2 Operational Database Systems

An operational system is a system that is used daily or constantly to perform routine operations or part of normal business processes. For example, Order Entry, Purchasing, Stock/Trading, Bank operations.

Features of Operational Systems

- They are OLTP systems
- They generally use the E-R data model.
- Need to work for routine tasks
- It is used to store transactional data.
- The information content is generally recent.
- Optimized to handle large numbers of simple read/write transactions
- Optimized for fast response to predefined transactions
- Used by people who deal with customers, products -- clerks, salespeople etc.
- Their goals are data accuracy & consistency , Concurrency , Recoverability, Reliability (ACID Properties).

Database Systems have been used traditionally for OLTP

- clerical data processing tasks
- detailed, up to date data
- structured repetitive tasks
- read/update a few records
- isolation, recovery and integrity are critical

OLTP vs. Data Warehouse (or) Operational data Vs Datawarehouse

- OLTP
  - Application Oriented
  - Used to run business
  - Detailed data
  - Current up to date
  - Isolated Data
  - Repetitive access
  - Clerical User
  - Performance Sensitive
  - Few Records accessed at a time (tens)
  - Read/Update Access
  - No data redundancy
  - Database Size 100MB -100 GB
- Warehouse (DSS)
  - Subject Oriented
  - Used to analyze business
  - Summarized and refined
  - Snapshot data
  - Integrated Data
  - Ad-hoc access
  - Knowledge User (Manager)
  - Performance relaxed
  - Large volumes accessed at a time(millions)
  - Mostly Read (Batch Update)
  - Redundancy present
  - Database Size 100 GB - few terabytes

- Transaction throughput is the performance metric
- Thousands of users
- Managed in entirety

- Query throughput is the performance metric
- Hundreds of users
- Managed by subsets

1.3 Multidimensional DataModel. (Data cubes)

In relational databases constructed from E-R diagrams, relational tables represent either entity sets or relationship sets. Then, relational tables representing such relationship sets correspond to 2D data cubes: a mapping between two dimensions.
In multidimensional data model, 3- and higher dimensional data cubes correspond to relationship sets that connect more 3 or more different entity sets.

Data Cubes. A data cube is a relational table that consists of two types of attributes:
1. Measure attributes. These attributes measure certain quantities within the data. They can be aggregate (e.g., number of items purchased) or nonaggregate (e.g., price).
2. Dimension attributes. These attributes specify the features over which the measures are established/ computed.
Data cube tables also often are called fact tables.

Example. Consider a video-rentals
Locations(Id, Street, City, State, Zip, County);
Movies(Id, Title, Year, Studio, RentalPrice);
Customers(MemberId, Name, Address, City, State, Zip, Phone);
InStock(Location, Movie, NumCopies);
Rentals(Location, Movie, Customer, RentalDate, Due, Returned);

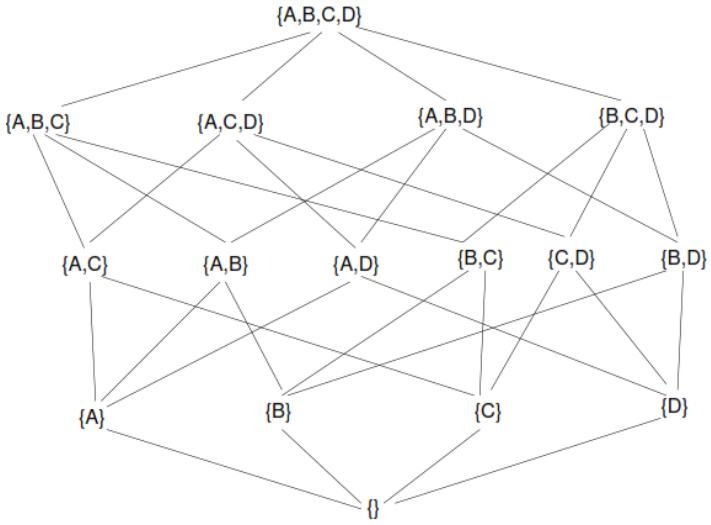
create a data cube (fact) table based on the table of rentals by aggregating the customer information:

```
CREATE TABLE RentalCube3D AS
(SELECT Location, Movie, RentalDate, COUNT(*) AS NumRentals FROM Rentals
GROUP BY Location, Movie, RentalDate);
```

The statement above generates a 3D data cube with the dimensions Movie Title, Store Location and Time.

Lattice of Cuboids

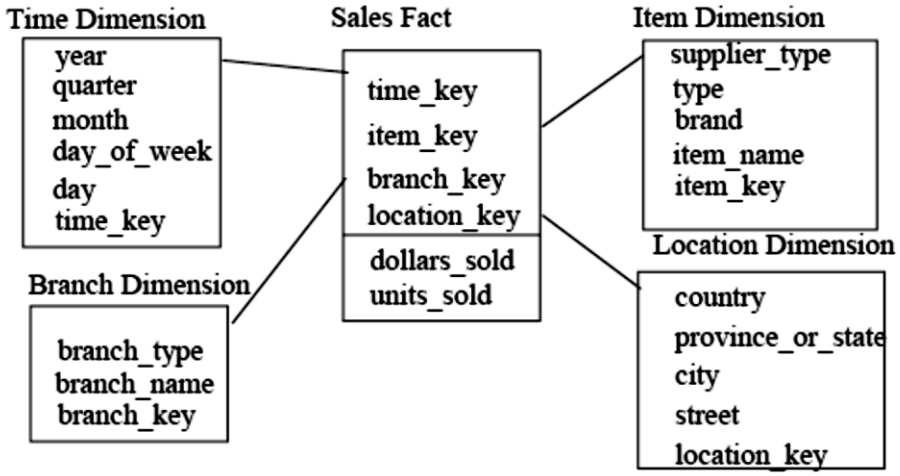
The Lattics of Cuboids is a graph G whose vertices are all subsets of a (cuboids). There is an edge from a node A to a node B is $A \subset B$.
The Lattice of cuboids is typically shown in layers, with the apex cuboid at the top (or bottom) layer and the base cuboid - at the bottom (top).



1.4 Schemas for Multidimensional Databases

Star schema: The star schema is a modeling paradigm in which the data warehouse contains (1) a large central table (fact table), and (2) a set of smaller attendant tables (dimension tables), one for each dimension. The schema graph resembles a starburst, with the dimension tables displayed in a radial pattern around the central fact table.

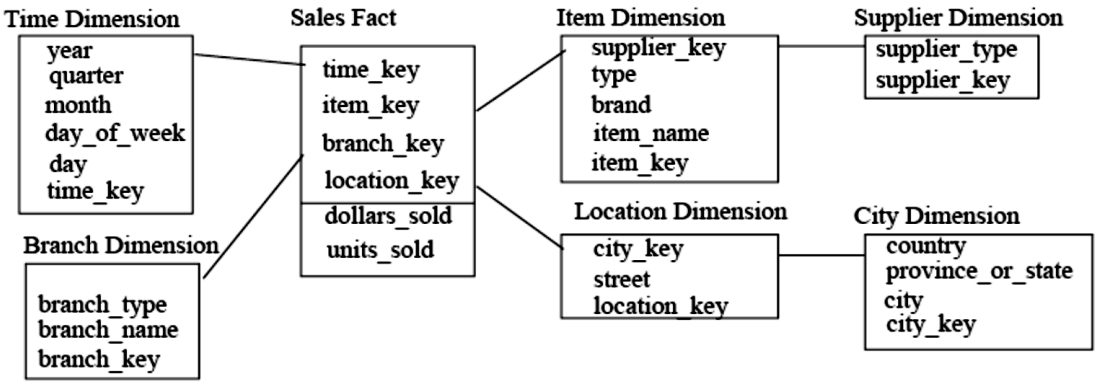
Figure Star schema of a data warehouse for sales.



Defining a Star Schema in DMQL

define cube sales_star [time, item, branch, location]:
dollars_sold = sum(sales_in_dollars), avg_sales = avg(sales_in_dollars), units_sold = count(*)
define dimension time as (time_key, day, day_of_week, month, quarter, year)
define dimension item as (item_key, item_name, brand, type, supplier_type)
define dimension branch as (branch_key, branch_name, branch_type)
define dimension location as (location_key, street, city, province_or_state, country)

Snowflake schema: The snowflake schema is a variant of the star schema model, where some dimension tables are normalized, thereby further splitting the data into additional tables. The resulting schema graph forms a shape similar to a snowflake. The major difference between the snowflake and star schema models is that the dimension tables of the snowflake model may be kept in normalized form. Such a table is easy to maintain and also saves storage space because a large dimension table can be extremely large when the dimensional structure is included as columns.



Defining a Snowflake Schema in DMQL

define cube sales_snowflake [time, item, branch, location]:

dollars_sold = sum(sales_in_dollars), avg_sales = avg(sales_in_dollars), units_sold = count(*)

define dimension time as (time_key, day, day_of_week, month, quarter, year)

define dimension item as (item_key, item_name, brand, type, supplier(supplier_key, supplier_type))

define dimension branch as (branch_key, branch_name, branch_type)

define dimension location as (location_key, street, city(city_key, province_or_state, country))

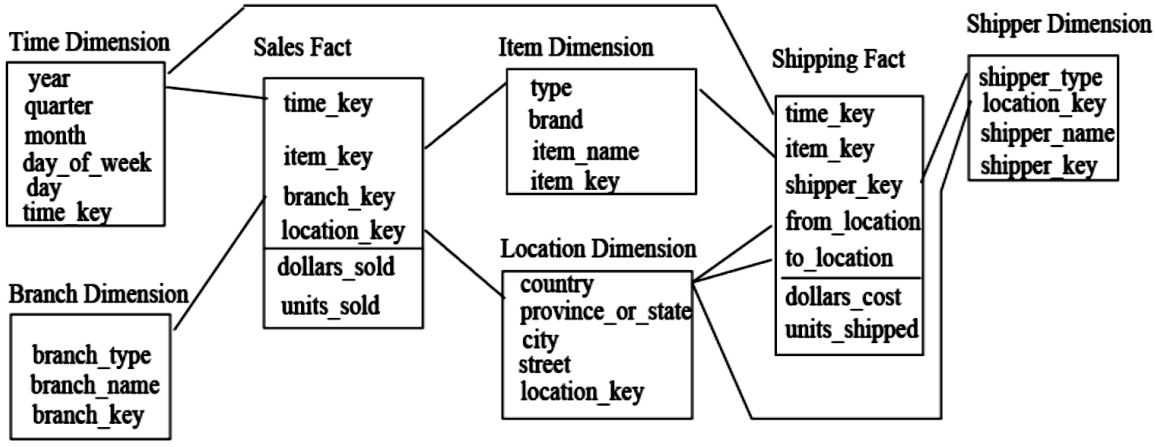
Advantage of Snowflake Schema

- The main advantage of Snowflake Schema is the improvement of query performance due to minimized disk storage requirements and joining smaller lookup tables.
- It is easier to maintain.
- Increase flexibility.

Disadvantage of Snowflake Schema

- The main disadvantage of the Snowflake Schema is the additional maintenance efforts needed to the increase number of lookup tables.
- Makes the queries much more difficult to create because more tables need to be joined.

Fact constellation: Sophisticated applications may require multiple fact tables to share dimension tables. This kind of schema can be viewed as a collection of stars, and hence is called a galaxy schema or a fact constellation.



What is Measure?

Measure: a function evaluated on aggregated data corresponding to given dimension-value pairs.

Measures can be:

- distributive: if the measure can be calculated in a distributive manner.
  - E.g., count(), sum(), min(), max().
- algebraic: if it can be computed from arguments obtained by applying distributive aggregate functions.
  - E.g., avg()=sum()/count(), min_N(), standard_deviation().
- holistic: if it is not algebraic.
  - E.g., median(), mode(), rank().

Data Cube

Cube Computation: ROLAP-Based Method

- Efficient cube computation methods
  - ROLAP-based cubing algorithms
  - Array-based cubing algorithm
  - Bottom-up computation method
- ROLAP-based cubing algorithms
  - Sorting, hashing, and grouping operations are applied to the dimension attributes in order to reorder and cluster related tuples
    - Grouping is performed on some subaggregates as a "partial grouping step"
    - Aggregates may be computed from previously computed aggregates, rather than from the base fact table

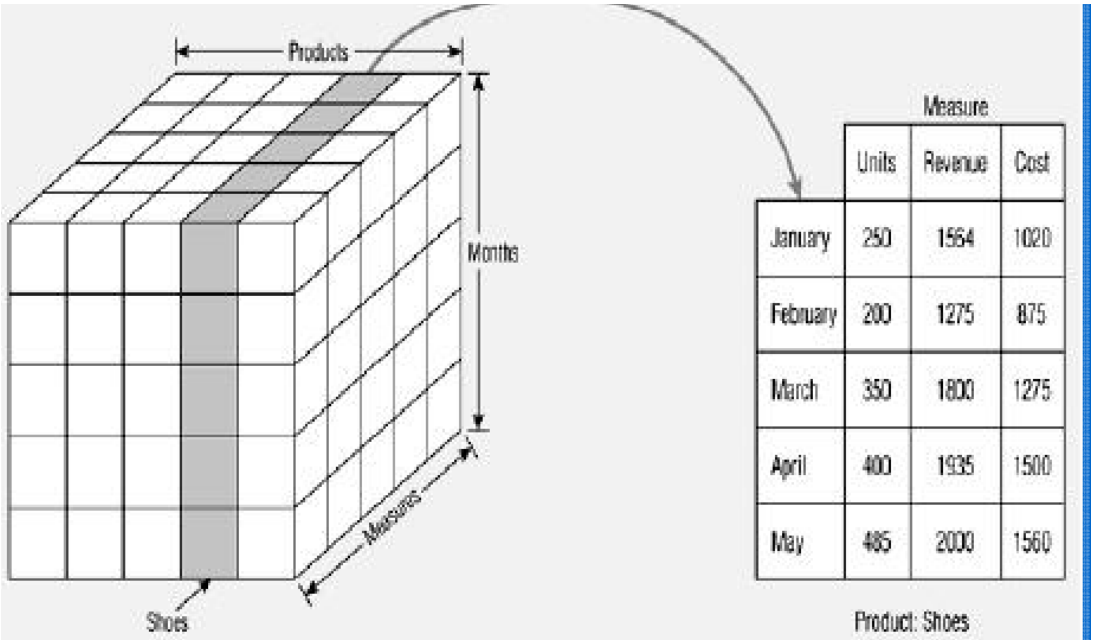
Cube Operations; Write OLAP operations here.

1.5 OLAP

OLAP is the application that use of a set of powerful graphical visualization tools that provides users with multidimensional views of their data and allows them to analyze the data.

OLAP allows users to analyze database information from multiple database systems at one time.

OLAP data is stored in multidimensional databases.



Some popular OLAP server software programs

Oracle Express Server
Hyperion Solutions Essbase

OLAP (Online Analytical Processing) is a term used to describe the analysis of complex data from the data warehouse. OLAP is the dynamic analysis, and consolidation of large volumes of multi-dimensional data.
It views of aggregate data to provide quick access to strategic information for the purposes of advanced analysis.
OLAP enables users to gain a deeper understanding and knowledge about various aspects of their corporate data through fast, consistent, interactive access to a variety of possible views of data.

Examples of OLAP

Comparisons (this period v.s. last period)
- Show me the sales per region for this year and compare it to that of the previous year to identify discrepancies

Multidimensional ratios (percent to total)
- Show me the contribution to weekly profit made by all items sold in the northeast stores between may 1 and may 7

Ranking and statistical profiles (top N/bottom N)
- Show me sales, profit and average call volume per day for my 10 most profitable salespeople

Custom consolidation (market segments, ad hoc groups)
- Show me an abbreviated income statement by quarter for the last four quarters for my northeast region operations

Features of OLAP
- Multi-dimensional views of data.
- Support for complex calculations.
- Time Intelligence.

OLAP Applications

Finance: Budgeting, activity-based costing, financial performance analysis, and financial modeling.
Sales: Sales analysis and sales forecasting.
Marketing: Market research analysis, sales forecasting, promotions analysis, customer analysis, and market/customer segmentation.
Manufacturing: Production planning and defect analysis.

OLAP Operations

OLAP operations on multidimensional data.

1. **Roll-up**: The roll-up operation performs aggregation by climbing-up a concept hierarchy for a dimension (or) by reducing dimensions. The following example shows the result of a roll-up operation performed on the central cube by climbing up the concept hierarchy for location. This hierarchy was defined as the total order street < city < state <. country

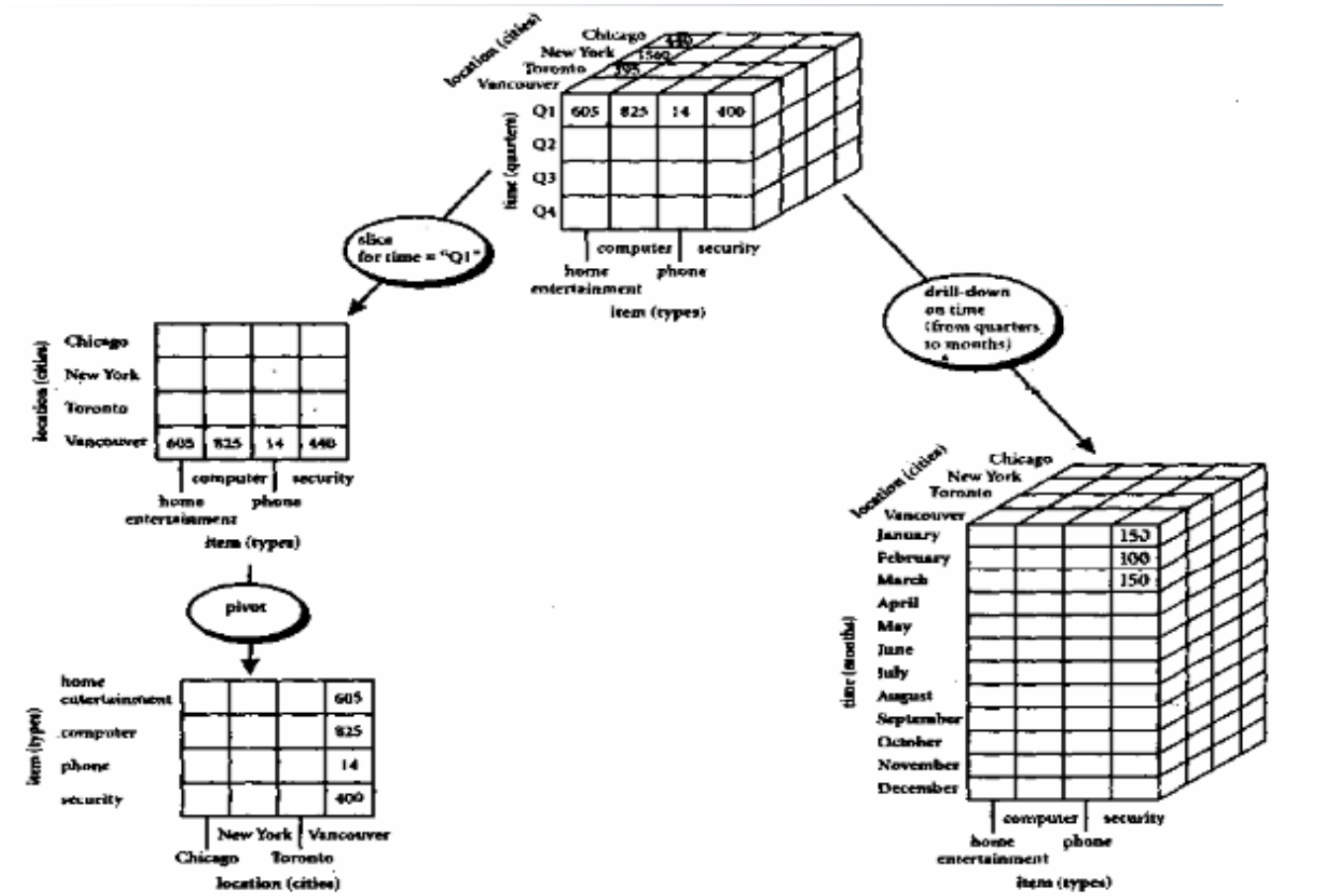
```
SELECT productname, Location, SUM(sales)FROM market
GROUP BY productname, Location;
```

2. **Drill-down**: Drill-down is the reverse of roll-up. It navigates from less detailed data to more detailed data. Drill-down can be realized by either stepping-down a concept hierarchy for a dimension or introducing additional dimensions. Figure shows the result of a drill-down operation performed on the central cube by stepping down a concept hierarchy for time defined as Year > Quarter > Month > Day. Drill-down occurs by descending the time hierarchy from the level of quarter to the more detailed level of month.

3. **Slice and dice:** The slice operation performs a selection on one dimension of the given cube which returns subcube. Figure shows a slice operation where the sales data are selected from the central cube for the dimension time using the criteria location=”NewDelhi". The dice operation defines a subcube by performing a selection on two or more dimensions.

```
SELECT productname, Location, sales FROM market where location=’NewDelhi’;
```

4. **Pivot (rotate):** Pivot is a visualization operation which rotates the data axes so the data cube does not change, but the viewpoint from which it is viewed is shifted.



OLTP vs OLAP

SNO	OLTP	OLAP
1.	On-Line Transaction Processing	On-Line Analytical Processing
2.	Short transactions both queries and updates.	Long transactions usually complex queries.
3.	Queries are simple.	Queries are complex.
4.	Updates are frequent. (eg. Reservation ticket, bankbalance.	Infrequent updates.
5.	User is IT Professional.	User is Knowledge worker.
6.	Its function is daily task.	Its function is decision making.
7.	Its DB design is application oriented.	Its design is subject oriented.
8.	Data – upto date, detail, relational	Historical, Multidimensional, Integrated.
9.	Access – Read / Write	Access Read-only.
10.	DB Size – 100 mb to 100 gb	100 gb – 1 tb

1.6 Data warehouse Architecture and its seven components

1. Warehouse/database technology
2. Data sourcing, cleanup, transformation, and migration tools
3. Metadata repository
4. Data marts
5. Data query, reporting, analysis, and mining tools
6. Data warehouse administration and management
7. Information delivery system

1. Data warehouse database

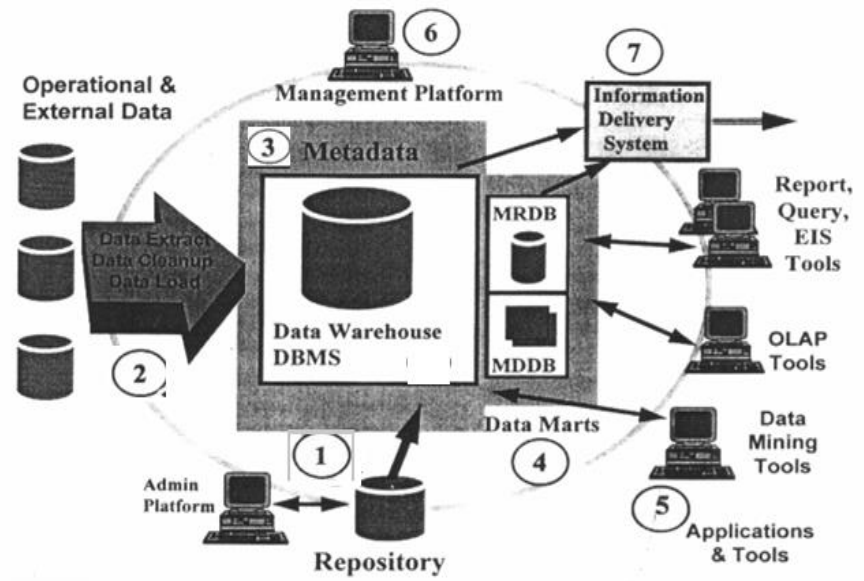
Data warehouse is an environment, not a product which is based on relational database management system that functions as the central storage area for informational data.

The central storage area information is surrounded by number of key components designed to make the environment is functional, manageable and accessible.

The data source for data warehouse is coming from operational applications. The data entered into the data warehouse transformed into an integrated structure and format. The transformation process involves conversion, summarization, filtering and condensation. The data warehouse must be capable of holding and managing large volumes of data as well as different structure of data structures over the time.

2. Sourcing, Acquisition, Clean up, and Transformation Tools

They perform conversions, summarization, key changes, structural changes and condensation. The data transformation is required so that the information can by used by decision support tools. The transformation produces programs, control statements, COBOL code, UNIX scripts, and SQL DDL code etc., to move the data into data warehouse from multiple operational systems.



Data warehouse environment.

The functionalities of these tools are listed below:

- To remove unwanted data from operational db.
- Converting to common data names and attributes.
- Calculating summaries and derived data.
- Establishing defaults for missing data.
- Accommodating source data definition changes.

Issues to be considered while data sourcing, cleanup, extract and transformation:

Data heterogeneity: It refers to the different way the data is defined and used in different modules, it may have different access languages, it may have data navigation methods, operations, concurrency, integrity and recovery processes etc.,

Some experts involved in the development of such tools:

Prism Solutions, Evolutionary Technology Inc., Vality, Praxis and Carleton

3. Meta data

It is data about data. It is used for maintaining, managing and using the data warehouse. It is classified into two:

Technical Meta data: It contains information about data warehouse data used by warehouse designer, administrator to carry out development and management tasks. It includes,

- Info about data stores
- Transformation descriptions.- That is mapping methods from operational db to warehouse db
- Warehouse- Object and data structure definitions for target data
- The rules used to perform clean up, and data enhancement
- Data mapping operations
- Access authorization, backup history, archive history, info delivery history, data acquisition history, data access etc.,

Business Meta data: It contains information that gives information stored in data warehouse to users. It includes,

- Subject areas, and info object type including queries, reports, images, video, audio clips etc.
- Internet home pages
- Info related to info delivery system
- Data warehouse operational info such as ownerships, audit trails etc.,

Meta data helps the users to understand content and find the data. Meta data are stored in a separate data stores which is known as informational directory or Meta data repository which helps to integrate, maintain and view the contents of the data warehouse. The following lists the characteristics of info directory/ Meta data:

- It is the gateway to the data warehouse environment.
- It supports easy distribution and replication of content for high performance and availability
- It should be searchable by business oriented key words
- It should act as a launch platform for end user to access data and analysis tools
- It should support the sharing of info.
- It should support scheduling options for request
- It should support and provide interface to other applications
- It should support end user monitoring of the status of the data warehouse environment

4 Access tools

Its purpose is to provide info to business users for decision making. There are five categories:

- Data Query and Reporting tools
- Application Development tools
- Executive info system tools (EIS)
- OLAP tools
- Data mining tools

Query and reporting tools are used to generate query and report. There are two types of reporting tools. They are:

- Production reporting tool used to generate regular operational reports
- Desktop report writer are inexpensive desktop tools designed for end users.

Managed Query tools: used to generate SQL query. It uses Meta layer software in between users and databases which offers a point-and-click creation of SQL statement. This tool is a preferred choice of users to perform segment identification, demographic analysis, territory management and preparation of customer mailing lists etc.

Application development tools: This is a graphical data access environment which integrates OLAP tools with data warehouse and can be used to access all db systems

OLAP Tools: are used to analyze the data in multi dimensional and complex views. To enable multidimensional properties it uses MDDB and MRDB where MDDB refers multi dimensional data base and MRDB refers multi relational data bases.

Data mining tools: are used to discover knowledge from the data warehouse data also can be used for data visualization and data correction purposes.

5. Data marts

Departmental subsets that focus on selected subjects. They are independent used by dedicated user group. They are used for rapid delivery of enhanced decision support functionality to end users. Data mart is used in the following situation:

- Extremely urgent user requirement.
- The absence of a budget for a full scale data warehouse strategy.
- The decentralization of business needs.
- The attraction of easy to use tools and mind sized project.

Data mart presents two problems:

1. Scalability: A small data mart can grow quickly in multi dimensions. So that while designing it, the organization has to pay more attention on system scalability, consistency and manageability issues
2. Data integration

6. Data warehouse admin and management

The management of data warehouse includes,

- Security and priority management
- Monitoring updates from multiple sources
- Data quality checks
- Managing and updating meta data
- Auditing and reporting data warehouse usage and status
- Removing data
- Replicating, sub setting and distributing data
- Backup and recovery
- Data warehouse storage management which includes capacity planning, hierarchical storage management and purging of aged data etc.,

7 Information delivery system

- It is used to enable the process of subscribing for data warehouse info.
- Delivery to one or more destinations according to specified scheduling algorithm.