UNIT I

Introduction: Network centric computing, Network centric content, peer-to –peer systems, cloud computing delivery models and services, Ethical issues, Vulnerabilities, Major challenges for cloud computing. Parallel and Distributed Systems: introduction, architecture, distributed systems, communication protocols, logical clocks, message delivery rules, concurrency, and model concurrency with Petri Nets.

CLOUD COMPUTING:

"**Cloud computing** is the on-demand availability of computer system resources, especially data storage (cloud storage) and computing power, without direct active management by the user."

"**Cloud computing** is an internet-based computing service in which large groups of remote servers are networked to allow centralized data storage, and online access to computer services or resources."

In utility computing the hardware and software resources are concentrated in large data centers and users can pay as they consume computing, storage, and communication resources. Utility computing often requires a cloud-like infrastructure, but its focus is on the business model for providing the computing services.

Cloud computing is a path to utility computing embraced by major IT companies such as Amazon, Apple, Google, HP, IBM, Microsoft, Oracle, and others.

Cloud computing delivery models, deployment models, defining attributes, resources, and organization of the infrastructure.

There are three cloud delivery models:

1. Software-as-a-Service (SaaS),

2. Platform-as-a-Service (PaaS),

3. Infrastructure-as-a-Service (IaaS),

Which deployed as public, private, community, and hybrid clouds.

SaaS is basically the application delivery over the Internet. The application is installed on to the cloud provider's servers and each user has a web browser interface to access the applications. The data that you store in this environment can be accessed from any device with an internet connection.

PaaS offers a platform over the cloud where each user can access resources such as databases, storage, and bandwidth with a single login. The platform enables users to develop and deploy applications in which they can use applications programming interfaces (API).

IaaS provides storage, processor power, memory, operating systems, and networking capabilities to customers so that they do not have to buy and maintain their own computer system infrastructure.

The defining attributes of the new philosophy for delivering computing services are as follows:

- Cloud computing uses Internet technologies to offer elastic services. The term elastic computing refers to the ability to dynamically acquire computing resources and support a variable workload.
- The resources used for these services can be metered and the users can be charged only for the resources they use.
- Maintenance and security are ensured by service providers.
- Economy of scale allows service providers to operate more efficiently due to specialization and centralization.
- Cloud computing is cost-effective due to resource multiplexing; lower costs for the service provider are passed on to the cloud users.
- The application data is stored closer to the site where it is used in a device- and location-independent manner; potentially, this data storage strategy increases reliability and security and, at the same time, it lowers communication costs.

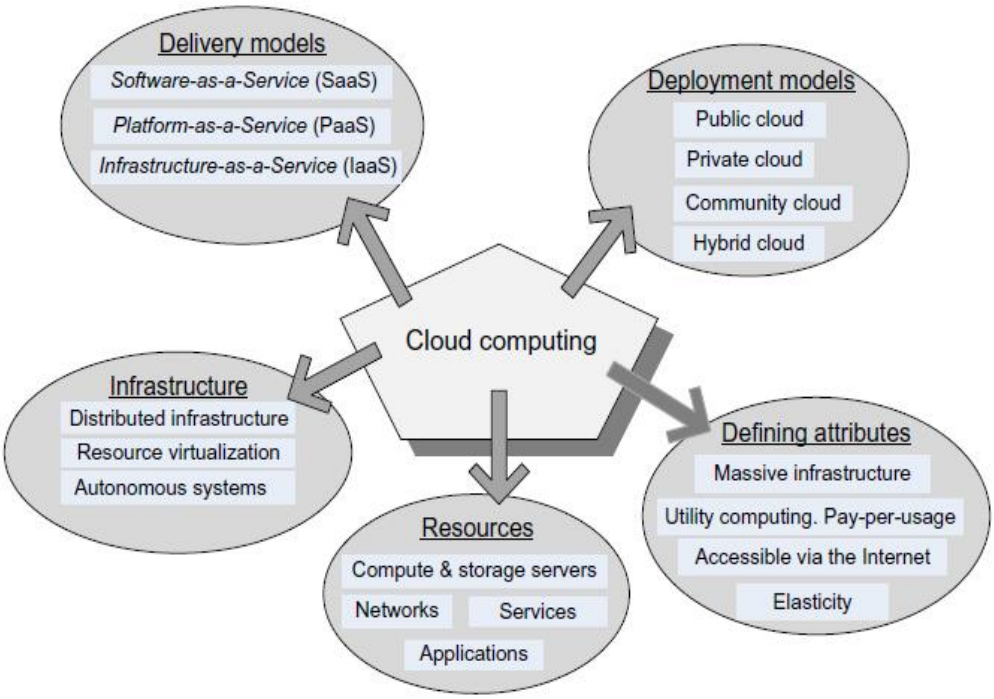


Figure : 1.1 Cloud computing: Delivery models, deployment models, defining attributes, resources, and organization of the infrastructure.

Network-centric computing and network-centric content:

Network-centric Computing focuses on large-scale distributed computing systems and applications that communicate through open, wide-area networks like the Internet. Typical examples of large-scale network-centric systems are the World-Wide Web and Computational Grids.

- Applications and data are downloaded from servers and exchanged with peers across a network.
- The Web and the semantic Web are expected to support composition of services.
- The Grid, initiated in the early 1990s by National Laboratories and Universities, is used primarily for applications in the area of science and engineering.
- Computer clouds, promoted since 2005 as a form of service-oriented computing by large IT companies, are used for enterprise computing, high-performance computing, Web hosting, and storage for network-centric content.

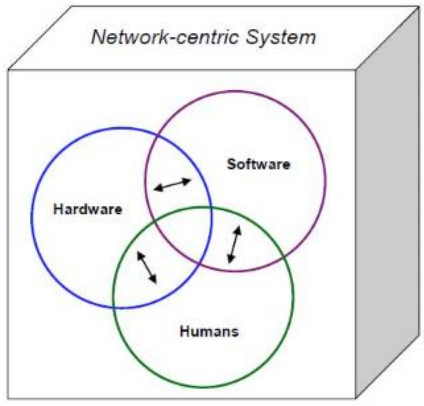
There are two major components of the World Wide Web:
1. Hyper Text Markup Language (HTML) for data description and
2. Hyper Text Transfer Protocol (HTTP) for data transfer.

The Web opened a new era in data sharing and ultimately led to the concept of network-centric content. The semantic Web2 is an effort to enable lay people to more easily find, share, and combine information available on the Web. machines can perform more of the tedious work involved in finding, combining, and acting upon information on the Web. Several technologies are necessary to provide a formal description of

1. Concepts,

2. Terms, and

3. Relationships within a given knowledge domain; they include the Resource description Framework (RDF), a variety of data interchange formats, and notations such as RDF Schema (RDFS) and the Web Ontology Language (OWL).



Major components of a Network-Centric system

Network – centric computing:

The idea to link such centers in an infrastructure resembling the power grid was born; the model known as **network-centric computing** was taking shape.

A ***computing grid*** is a distributed system consisting of a large number of loosely coupled, heterogeneous, and geographically dispersed systems in different administrative domains.

The term ***computing grid*** is a metaphor for accessing computer power with similar ease as we access power provided by the electric grid. Software libraries known as ***middleware*** have been furiously developed since the early 1990s to facilitate access to grid services.

- The companies promoting cloud computing seem to have learned the most important lessons from the grid movement.
- Computer clouds are typically homogeneous.
- An entire cloud shares the same security, resource management, cost and other policies, and last but not least, it targets enterprise computing.

S**ome of the reasons that several agencies of the US Government, including**
1. Health and Human Services (HHS),
2. the Centers for Disease Control (CDC),
3. the National Aeronautics and Space Administration (NASA),
4. the Navy's Next Generation Enterprise Network (NGEN), and
5. the Defense Information Systems Agency (DISA), have launched cloud computing initiatives and conduct actual system development intended to improve the efficiency and effectiveness of their information processing needs.

Network-centric content:

- The term *content* refers to any type or volume of media, be it static or dynamic, monolithic or modular, live or stored, produced by aggregation, or mixed.

- *Information* is the result of functions applied to content.

- The creation and consumption of audio and visual content are likely to transform the Internet to support increased quality in terms of resolution, frame rate, color depth, and stereoscopic information, and it seems reasonable to assume that the Future Internet3 will be content-centric.

Network-centric computing and network-centric content share a number of characteristics:

- Most applications are data-intensive.

- Computer simulation becomes a powerful tool for scientific research in virtually all areas of science, from physics, biology, and chemistry to archeology.

- Sophisticated tools for computer-aided design, such as Catia (Computer Aided Three-dimensional Interactive Application), are widely used in the aerospace and automotive industries.

- The wide spread use of sensors contributes to increases in the volume of data. Multimedia applications are increasingly popular; the ever-larger media increase the load placed on storage, networking, and processing systems.

- Virtually all applications are network-intensive. Indeed, transferring large volumes of data requires

- high-bandwidth networks; parallel computing, computation steering,4 and data streaming are examples of applications that can only run efficiently on low-latency networks.

- The systems are accessed using thin clients running on systems with limited resources. In June 2011

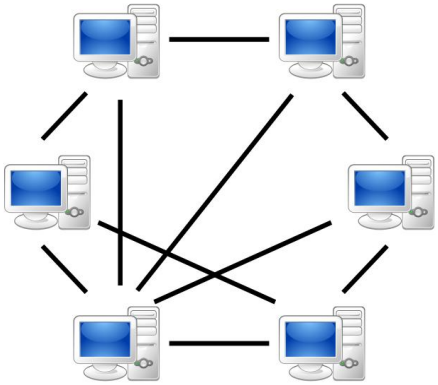
- Google released Google Chrome OS, designed to run on primitive devices and based on the browser with the same name.

- The infrastructure supports some form of workflow management. Indeed, complex computational
- tasks require coordination of several applications; composition of services is a basic tenet of Web 2.0.

Peer-to-peer systems:

Peer-To-Peer Network, the "peers" are computer systems that are connected to each other via the Internet. Files can be shared directly between systems on the network without the need for a central server.

Peer-to-Peer systems consist of interconnected peers of similar capabilities and responsibilities, where the peers can act as both servers and clients.



- P2P systems can be regarded as one of the precursors of today's clouds.

- New model for distributed computing promoted the idea of low-cost access to storage and central processing unit (CPU) cycles provided by participant systems;
- The resources are located in different administrative domains.
- The P2P systems are self-organizing and decentralized, whereas the servers in a cloud are in a single administrative domain and have a central management.
- P2P systems exploit the network infrastructure to provide access to distributed computing resources.

Decentralized applications developed in the 1980s, such as

- Simple Mail Transfer Protocol (SMTP), a protocol for email distribution, and
- Network News Transfer Protocol (NNTP), An application protocol for dissemination of news articles, are early examples of P2P systems.

P2P systems have several desirable properties:

- They require a minimally dedicated infrastructure, since resources are contributed by the participating systems.
- They are highly decentralized.
- They are scalable; the individual nodes are not required to be aware of the global state.
- They are resilient to faults and attacks, since few of their elements are critical for the delivery of service and the abundance of resources can support a high degree of replication.
- Individual nodes do not require excessive network bandwidth the way servers used in case of the client-server model do.
- Last but not least, the systems are shielded from censorship due to the dynamic and often unstructured system architecture.

Types of P2P networks :

1. **Unstructured P2P networks –**
   In this type of P2P network, each device is able to make an equal contribution. This network is easy to build as devices can be connected randomly in the network. But being unstructured, it becomes difficult to find content.
2. **Structured P2P networks –**
   It is designed using the software which creates a virtual layer in order to put the nodes in a specific structure. These are not easy to set-up but can give easy access to users to the content.
3. **Hybrid P2P networks –**
   It combines the features of both P2P network and client-server architecture. An example of such a network is to find a node using the central server.

Examples of P2P networks:

P2P networks can be basically categorized into three levels. The first level is the basic level which uses a USB to create a P2P network between two systems. The second is the intermediate level which involves the usage of copper wires in order to connect more than two systems. The third is the advanced level which uses software to establish protocols in order to manage numerous devices across the internet.

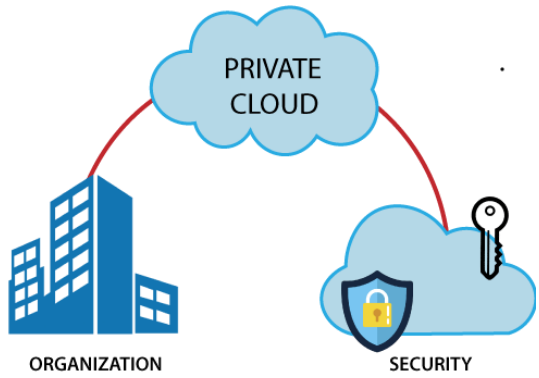
Cloud computing: an old idea whose time has come

- Once the technological elements were in place, it was only a matter of time until the economical advantages of cloud computing became apparent.
- Due to the economy of scale, large data centers – centers with more than 50,000 systems – are more economical to operate than medium-sized centers that have around 1,000 systems.
- Large data centers equipped with commodity computers experience a five to seven times decrease of resource consumption, including energy, compared to medium-sized centers .
- The networking costs, in dollars per Mbit/s/month, are 95/13 = 7.1 times larger, and the storage costs, in dollars per Gbyte/month, are 2.2/0.4 = 5.7 times larger for medium-sized centers.
- Medium-sized centers have a larger administrative overhead – one system administrator for 140 systems versus one for 1,000 systems for large centers.

Types of cloud Computing

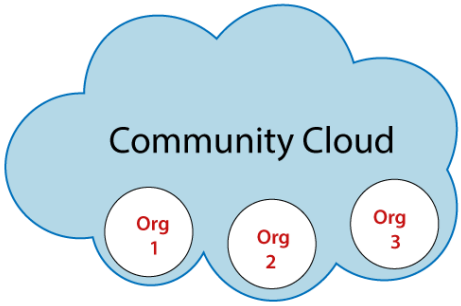
The term computer cloud is overloaded, since it covers infrastructures of different sizes,with different management and different user populations. Several types of cloud are envisioned:

1. Private cloud. The infrastructure is operated only for an organization. Private cloud is also known as an **internal cloud** or **corporate cloud**. It is used by organizations to build and manage their own data centers internally or by the third party. It can be deployed using Opensource tools such as Openstack and Eucalyptus.Based on the location and management,

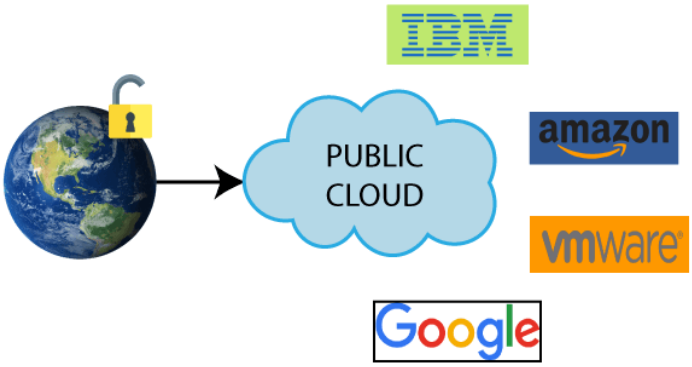


2. Community cloud. The infrastructure is shared by several organizations and supports a specific community that has shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be managed by the organizations or a third party and may exist on premises or off premises.

Example: Health Care community cloud



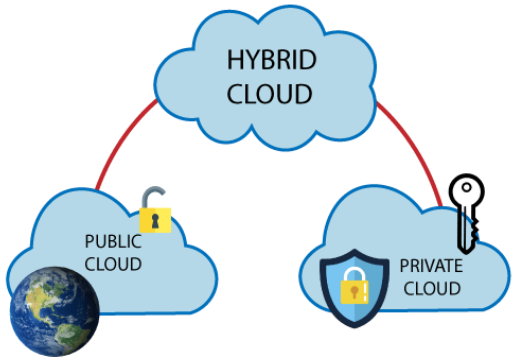
3. Public cloud. The infrastructure is made available to the general public or a large industry group and is owned by an organization selling cloud services. Public cloud is **open to all** to store and access information via the Internet using the pay-per-usage method.
In public cloud, computing resources are managed and operated by the Cloud Service Provider (CSP).



Example: **Amazon elastic compute cloud (EC2), IBM SmartCloud Enterprise, Microsoft, Google App Engine, Windows Azure Services Platform.**

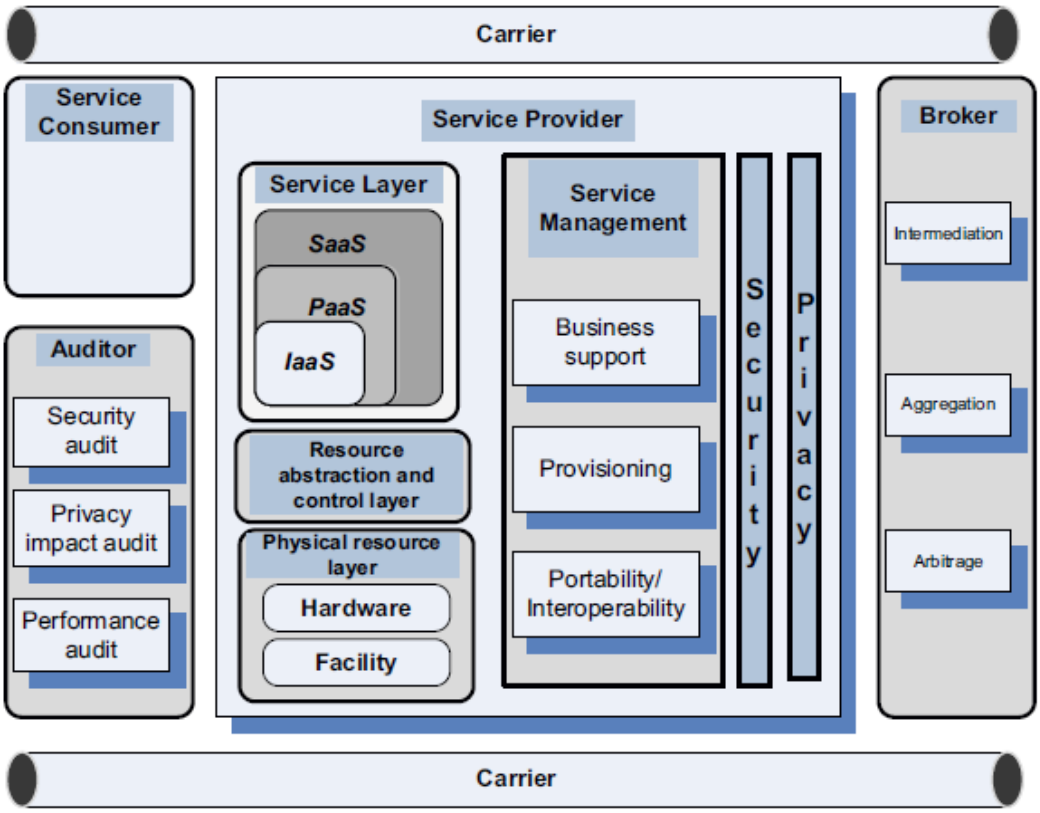
4. Hybrid cloud. The infrastructure is a composition of two or more clouds (private, community, or public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds).

Example: Google Application Suite (Gmail, Google Apps, and Google Drive), Office 365 (MS Office on the Web and One Drive), Amazon Web Services.



Cloud Computing Delivery Models and Services:

There are three entities
1. Service Consumer
2. Service provider
3. Broker



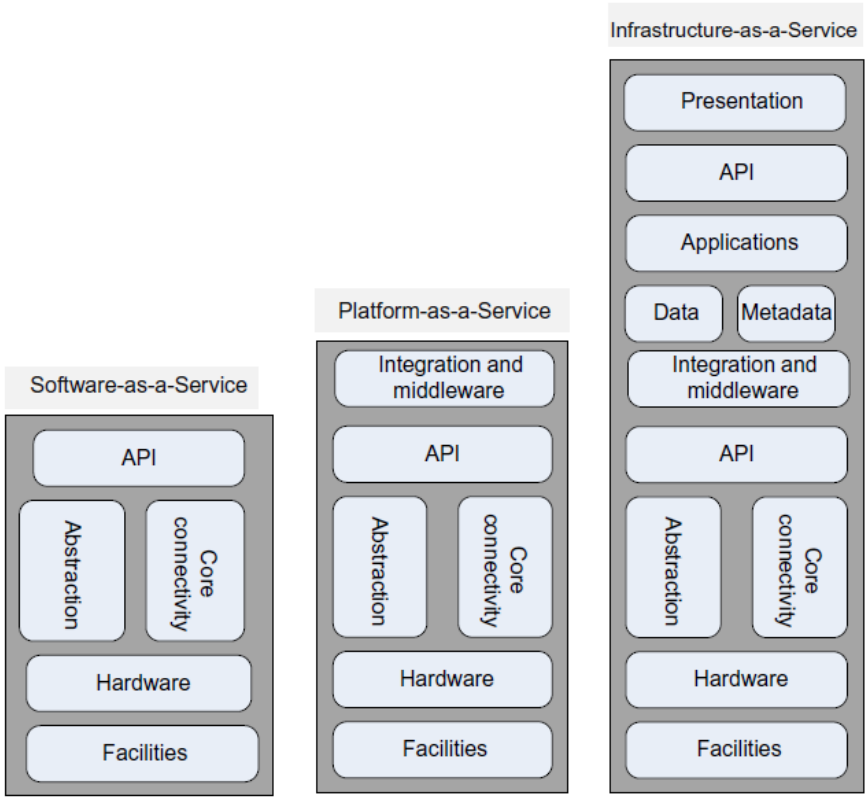
- The service consumer, the entity that maintains a business relationship with and uses service from service providers;
- The service provider, the entity responsible for making a service available to service consumers;

- The carrier, the intermediary that provides connectivity and transport of cloud services between providers and consumers;
- The broker, an entity that manages the use, performance, and delivery of cloud services and negotiates relationships between providers and consumers; and the auditor, a party that can conduct independent assessment of cloud services, information system operations, performance, and security of the cloud implementation.
- An audit is a systematic evaluation of a cloud system that measures how well it conforms to a set of established criteria. For example, a security audit evaluates cloud security, a privacy-impact audit evaluates cloud privacy assurance, and a performance audit evaluates cloud performance.

Structure of Delivery models:

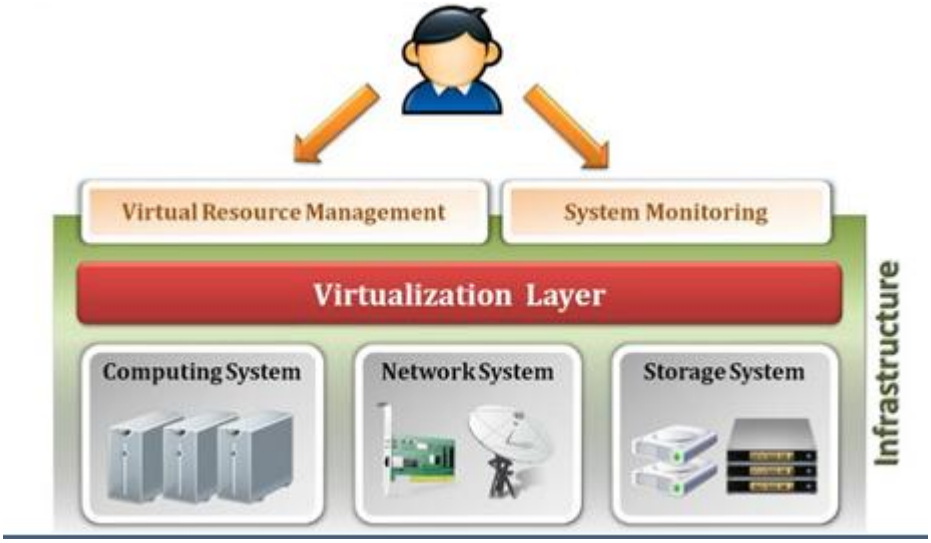
There are three delivery models

1. SaaS
2. PaaS
3. IaaS



Infrastructure as a Service (IaaS):

IaaS can deliver networking and a large database for storage and servers, and enables a business to reap the benefits of its own data center. IaaS providers will create an environment depending on a company's requirements, which allows businesses to only pay for the space they use.



IAAS Providers:

Amazon AWS, DigitalOcean, Microsoft Azure, Rackspace Open Cloud, Google Compute Engine, HP Enterprise, IBM SmartCloud Enterprise, Green Cloud Technologies. Etc…

Benefits Of IaaS:-

1) Cost savings:- As infrastructure is on the internet we don't have to maintain hardware and network devices or replacing old devices with new devices.

2) Multiple users:- More than one user can work on a single server from anywhere and anytime.

3) Server quality:- IaaS can run even if a server goes down. As data spread over multiple servers and if one particular hardware device fails then it will not affect on organization's infrastructure.

Drawbacks of IaaS:-

1) Data Loss:-

2) Vulnerability:-

Software as a Service | SaaS:

SaaS is a software distribution model in which services are hosted by a cloud service provider. These services are available to end-users over the internet so, the end-users do not need to install any software on their devices to access these services.

Business Services - SaaS Provider provides various business services to start-up the business. The SaaS business services include

- ERP (Enterprise Resource Planning),
- CRM (Customer Relationship Management), billing, and sales.

Document Management -

SaaS document management is a software application offered by a third party (SaaS providers) to create, manage, and track electronic documents.

Social Networks -

Social networking sites are used by the general public, so social networking service providers use SaaS for their convenience and handle the general public's information.

Mail Services -

To handle the unpredictable number of users and load on e-mail services, many e-mail providers offering their services using SaaS.

Top SaaS Providers:

Salesforce, Microsoft, Google Workspace, Xero, Zendesk, Cisco. Amazon Web Services (AWS) SaaS, Druva, NetSuit. Etc.

Advantages of SaaS

1) SaaS is easy to buy
2. One to Many
3. Less hardware required for SaaS
4. Low maintenance required for SaaS
5. No special software or hardware versions required

Disadvantages of SaaS

1) Security
2) Latency issue
3) Total Dependency on Internet
4) Switching between SaaS vendors is difficult

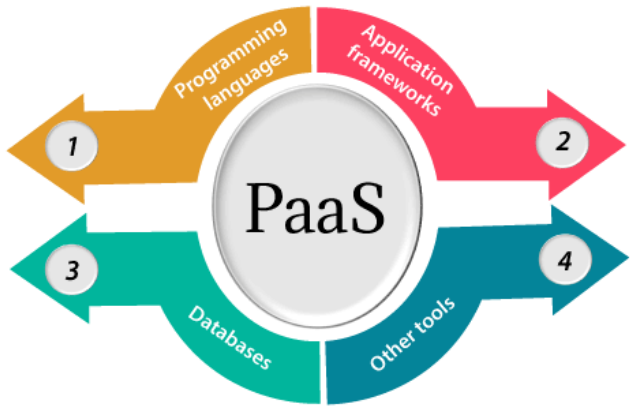
Platform as a Service | PaaS

Platform as a Service (PaaS) provides a runtime environment. It allows programmers to easily create, test, run, and deploy web applications. You can purchase these applications from a cloud service provider on a pay-as-per use basis and access them using the Internet connection.

PaaS includes infrastructure (servers, storage, and networking) and platform (middleware, development tools, database management systems, business intelligence, and more) to support the web application life cycle.

- Gives the capability to deploy consumer-created or acquired applications using programming languages and tools supported by the provider.
- The user does not manage or control the underlying cloud infrastructure, including network, servers, operating systems, or storage.
- The user has control over the deployed applications and, possibly, over the application hosting environment configurations.
- Such services include session management, device integration, sandboxes, instrumentation and testing, contents management, knowledge management, and Universal Description, Discovery, and Integration (UDDI), a platform-independent Extensible Markup Language (XML)-based
- Registry providing a mechanism to register and locate Web service applications.

PaaS is not particulary useful when the application must be portable, when proprietary programming languages are used, or when the original hardware and software must be customized to improve the performance of the application.



Paas Providers:

AWS, Engine Yard, Google Cloud, Heroku, IBM Cloud, Mendix aPaaS, Microsoft Azure, Red Hat OpenShift, Etc…

Advantages of PaaS

1) Simplified Development
2) Lower risk
3) Scalability

Disadvantages of PaaS

1) Data Privacy

Ethical issues in cloud computing:

Cloud computing is based on a paradigm shift with profound implications for computing ethics.

The main elements of this shift are:

(i) the control is relinquished to third-party services;
(ii) the data is stored on multiple sites administered by several organizations; and
(iii) multiple services interoperate across the network.

- Unauthorized access, data corruption, infrastructure failure, and service unavailability are some of the risks related to relinquishing the control to third-party services, whenever a problem occurs; it is difficult to identify the source and the entity causing it.

- Systems can span the boundaries of multiple organizations and cross security borders, a process called deperimeterization. As a result of deperimeterization, "not only the border of the organization's IT infrastructure blurs, also the border of the accountability becomes less clear".

- The complex structure of cloud services can make it difficult to determine who is responsible in case something undesirable happens.

- In a complex chain of events or systems, many entities contribute to an action, with undesirable consequences.

- Some of them have the opportunity to prevent these consequences, and therefore no one can be held responsible – the so-called "problem of many hands."

Some more Ethical issues:

1. Ubiquitous and unlimited data sharing and storage among organizations test the self-determination of information.
2. The right or ability of individuals to exercise personal control over the collection.
3. Use and disclosure of their personal data by others.
4. Confidence and trust in today's evolving information society.
5. Identity fraud and theft are made possible by the unauthorized access to personal data in circulation.
6. New forms of dissemination through social networks, which could also pose a danger to cloud computing.
7. Cloud service providers have already collected petabytes of sensitive personal information stored in data centers around the world.
8. The acceptance of cloud computing therefore will be determined by privacy issues addressed by these companies and the countries where the data centers are located.
9. Privacy is affected by cultural differences; though some cultures favor privacy, other cultures emphasize community.
10. Leads to an ambivalent attitude toward privacy on the Internet, which is a global system.

Cloud vulnerabilities:

- Clouds are affected by malicious attacks and failures of the infrastructure (e.g., power failures).
- Such events can affect Internet domain name servers and prevent access to a cloud or can directly affect
- the clouds. For example, an attack at Akamai on June 15, 2004 caused a domain name outage and a
- major blackout that affected Google, Yahoo!, and many other sites.
- In May 2009 Google was the target of a serious denial-of-service (DoS) attack that took down services such Google News and Gmail for several days.
- Lightning caused a prolonged downtime at Amazon on June 29 and 30, 2012; the *AWS* cloud in the Eastern region of the United States, which consists of 10 data centers across four availability zones, was initially troubled by utility power fluctuations, probably caused by an electrical storm.
- A June 29, 2012 storm on the East Coast took down some Virginia-based Amazon facilities and affected companies using systems exclusively in this region.
- *Instagram*, a photo-sharing service, was one of the victims of this outage, according to http://mashable.com/2012/06/30/aws-instagram/.
- The recovery from the failure took a very long time and exposed a range of problems. For example, one of the 10 centers failed to switch to backup generators before exhausting the power that could be supplied by *uninterruptible power supply* (UPS) units. *AWS* uses

"control planes" to allow users to switch to resources in a different region, and this software component also failed.

- The booting process was faulty and extended the time to restart *EC2* (*Elastic Computing*) and *EBS* (*Elastic Block Store*) services.
- Another critical problem was a bug in the elastic load balancer (ELB), which is used to route traffic to servers with available capacity.
- A similar bug affected the recovery process of the Relational Database Service (RDS). This event brought to light "hidden" problems that occur only under special circumstance

Parallel and Distributed Systems

Parallel Computing:

Parallel computing allows us to solve large problems by splitting them into smaller ones and solving them concurrently. In parallel computing multiple processors performs multiple tasks assigned to them simultaneously. Memory in parallel systems can either be shared or distributed. Parallel computing provides concurrency and saves time and money.

Parallel hardware and software systems allow us to solve problems demanding more resources than those provided by a single system and, at the same time, to reduce the time required to obtain a solution. The speed-up measures the effectiveness of parallelization; in the general case the speed-up of the parallel computation is defined as

$$S(N) = \frac{T(1)}{T(N)},$$

with T (1) the execution time of the sequential computation and T (N) the execution time when N parallel computations are carried out.

Amdahl's Law gives the potential speed-up of a parallel computation; it states that the portion of the computation that cannot be parallelized determines the overall speed-up. If α is the fraction of running time a sequential program spends on nonparallelizable segments of the computation, then

$$S = \frac{1}{\alpha}.$$

To prove this result, call σ the sequential time and π the parallel time and start from the definitions of $T(1)$, $T(N)$, and α:

$$T(1) = \sigma + \pi, \quad T(N) = \sigma + \frac{\pi}{N}, \quad \text{and} \quad \alpha = \frac{\sigma}{\sigma + \pi}.$$

Then

$$S = \frac{T(1)}{T(N)} = \frac{\sigma + \pi}{\sigma + \pi/N} = \frac{1 + \pi/\sigma}{1 + (\pi/\sigma) \times (1/N)}.$$

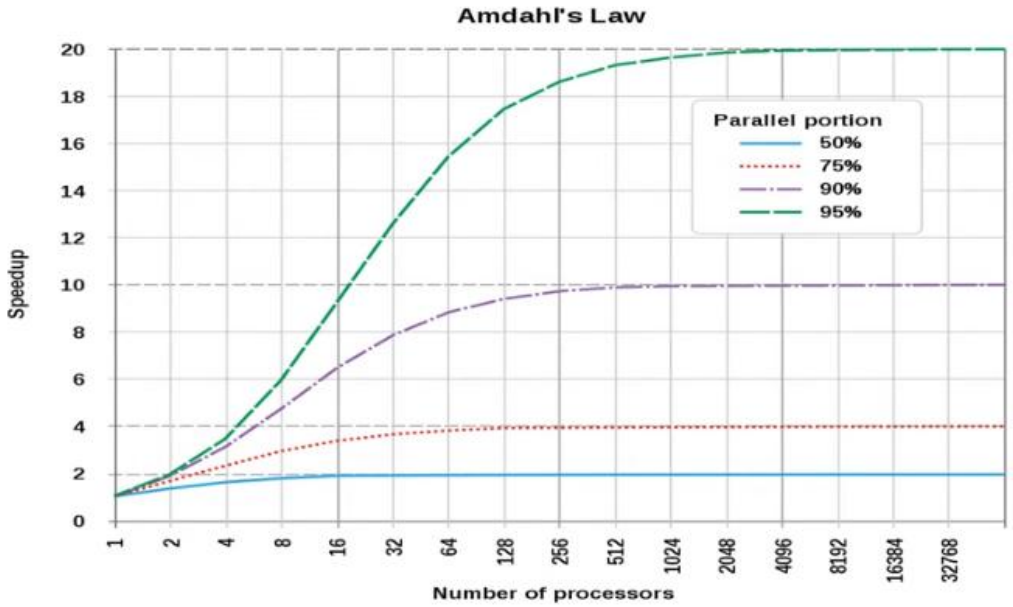
But

$$\pi/\sigma = \frac{1 - \alpha}{\alpha}.$$

Thus, for large N

$$S = \frac{1 + (1 - \alpha)/\alpha}{1 + (1 - \alpha)/(N\alpha)} = \frac{1}{\alpha + (1 - \alpha)/N} \approx \frac{1}{\alpha}.$$

Amdahl's law applies to a *fixed problem size*; in this case the amount of work assigned to each one of the parallel processes decreases when the number of processes increases, and this affects the efficiency of the parallel execution.



When the problem size is allowed to change, **Gustafson's Law** gives the scaled speed-up with N parallel processes as

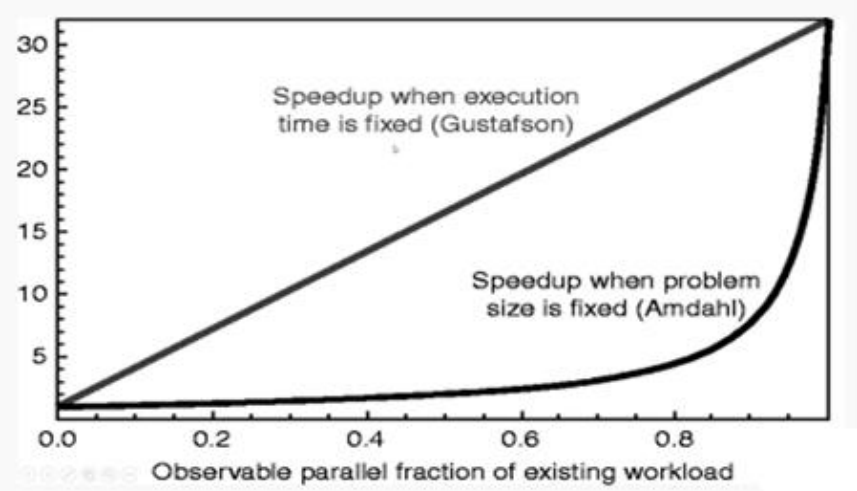
$$S(N) = N - \alpha(N - 1).$$

As before, we call σ the sequential time; now π is the fixed parallel time per process. The sequential execution time, T (1), and the parallel execution time with N parallel processes, T (N), are

$$T(1) = \sigma + N\pi \quad \text{and} \quad T(N) = \sigma + \pi$$

Then the scaled speed-up is

$$S(N) = \frac{T(1)}{T(N)} = \frac{\sigma + N\pi}{\sigma + \pi} = \frac{\sigma}{\sigma + \pi} + \frac{N\pi}{\sigma + \pi} = \alpha + N(1 - \alpha) = N - \alpha(N - 1).$$



- Concurrent execution can be challenging.
  - It could lead to race conditions, an undesirable effect when the results of concurrent execution depend on the sequence of events.
  - Shared resources must be protected by locks/ semaphores /monitors to ensure serial access.
  - Deadlocks and livelocks are possible.

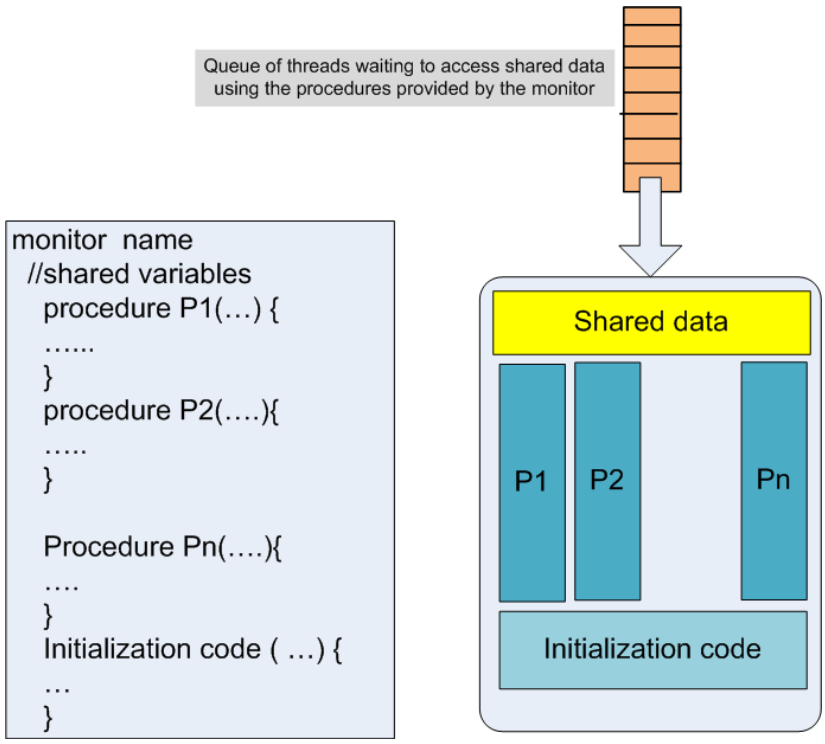
Deadlock:

Shared resources must be protected by locks to ensure serial access. Concurrent execution of multiple processes or threads is the presence of deadlocks; a deadlock occurs when processes or threads competing with one another for resources are forced to wait for additional resources held by other processes or threads and none of the processes or threads can finish.

The four Coffman conditions must hold simultaneously for a deadlock to occur:

1. Mutual exclusion. At least one resource must be nonsharable, and only one process/thread may use the resource at any given time.

2. Hold and wait. At least one process/thread must hold one or more resources and wait for others.
3. No preemption. The scheduler or a monitor should not be able to force a process/thread holding a resource to relinquish it.
4. Circular wait. Given the set of n processes/threads {P1, P2, P3, . . . , Pn}, P1 should wait for a resource held by P2, P2 should wait for a resource held by P3, and so on and Pn should wait for a resource held by P1.



More challenges:

- Livelock condition: Two or more processes/threads continually change their state in response to changes in the other processes; then none of the processes can complete its execution.
- Very often processes/threads running concurrently are assigned priorities and scheduled based on these priorities. Priority inversion, a higher priority process/task is indirectly preempted by a lower priority one.
- Discovering parallelism is often challenging and the development of parallel algorithms requires a considerable effort. For example, many numerical analysis problems, such as solving large systems of linear equations or solving systems of PDEs (Partial Differential Equations), require algorithms based on domain decomposition methods.

Parallelism:

- Fine-grain parallelism → relatively small blocks of the code can be executed in parallel without the need to communicate or synchronize with other threads or processes.
- Coarse-grain parallelism → large blocks of code can be executed in parallel.

- The speed-up of applications displaying fine-grain parallelism is considerably lower that those of coarse-grained applications; the processor speed is orders of magnitude larger than the communication speed even on systems with a fast interconnect.
- Data parallelism → the data is partitioned into several blocks and the blocks are processed in parallel.
- Same Program Multiple Data (SPMD) → data parallelism when multiple copies of the same program run concurrently, each one on a different data block.

Parallel Computer Architecture

Parallel computer architectures starts with the recognition that parallelism at different levels can be exploited. These levels are:

- **Bit level parallelism:**

  The number of bits processed per clock cycle, often called a word size, has increased gradually from 4-bit, to 8-bit, 16-bit, 32-bit, and to 64-bit. This has reduced the number of instructions required to process larger size operands and allowed a significant performance improvement. During this evolutionary process the number of address bits have also increased allowing instructions to reference a larger address space.

- **Instruction-level parallelism:**

  Today's computers use multi-stage processing pipelines to speed up execution. In a single CPU clock cycle, the processor decides in instruction-level parallelism how many instructions are implemented at the same time.

  A Complex Instruction Set Computing (CISC) architecture could have a much large number of pipelines stages;

  for example, an Intel Pentium 4 processor has a 35-stage pipeline.

An Example
1. e = a + b
2. f = c + d
3. g = e * f

Here, instruction 3 is dependent on instruction 1 and 2 .
However,instruction 1 and 2 can be independently processed.

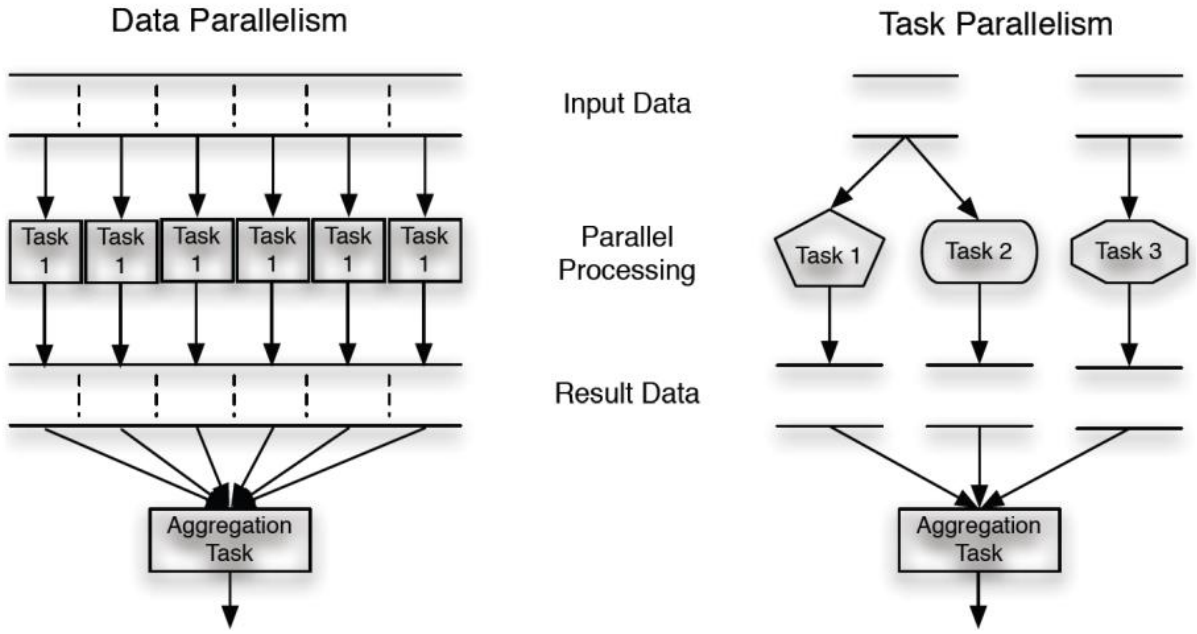
- **Data parallelism or loop parallelism:**

➢ The program loops can be processed in parallel. Data Parallelism means concurrent execution of the same task on each multiple computing core

- **Task parallelism**.

The problem can be decomposed into tasks that can be carried out concurrently. Task parallelism is the form of parallelism in which the tasks are decomposed into subtasks. Then, each subtask is allocated for execution. And, the execution of subtasks is performed concurrently by processors.

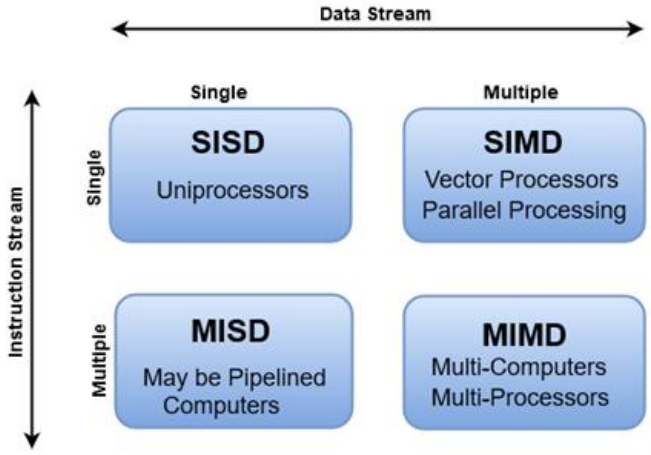
For example, SPMD. Note that data dependencies cause different flows of control in individual tasks.



Michael Flynn's classification of computer architectures is based on the number of concurrent control/instruction and data streams:

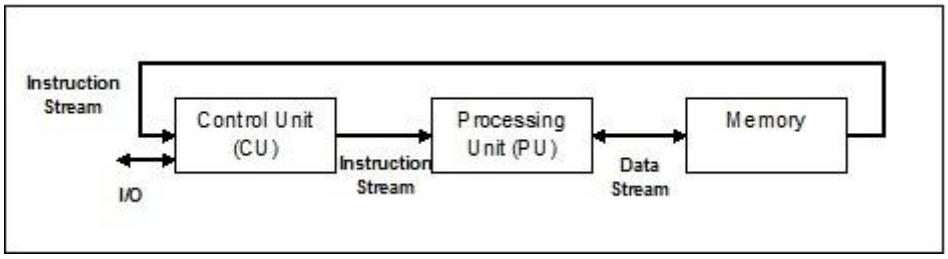
1. Single instruction stream, single data stream (SISD)
2. Single instruction stream, multiple data stream (SIMD)
3. Multiple instruction stream, single data stream (MISD)
4. Multiple instruction stream, multiple data stream (MIMD)

- The sequence of instructions read from memory constitutes an **instruction stream**.
- The operations performed on the data in the processor constitute a **data stream**.



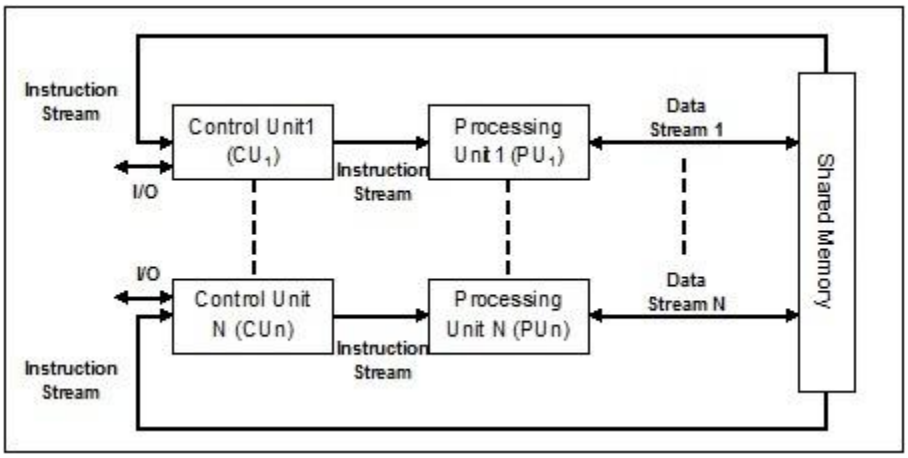
1. Single instruction stream, single data stream (SISD):

- It represents the organization of a single computer containing a control unit, a processor unit, and a memory unit. Instructions are executed sequentially, and the system may or may not have internal parallel processing capabilities.
- Parallel processing, in this case, may be achieved by means of multiple functional units or by pipeline processing.



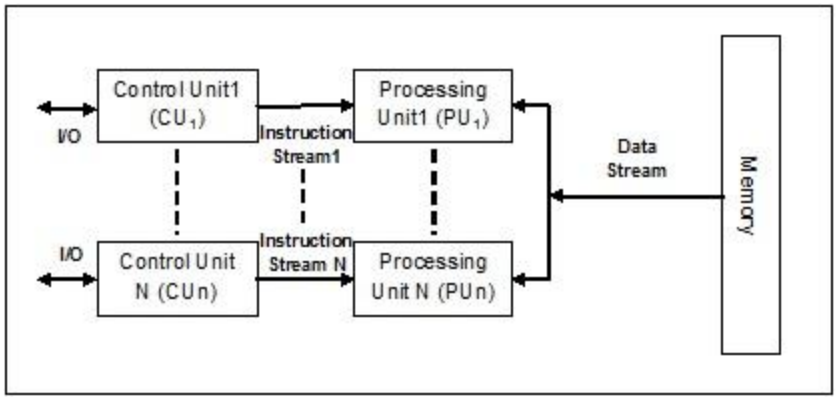
2. Single instruction stream, multiple data stream (SIMD)

- It represents an organization that includes many processing units under the supervision of a common control unit.
- The SIMD architecture supports vector processing. When an SIMD instruction is issued, the operations on individual vector components are carried out concurrently.
- The desire to support real-time graphics with vectors of two, three, or four dimensions led to the development of graphic processing units (GPUs).
- GPUs are very efficient at manipulating computer graphics, and their highly parallel structures based on SIMD execution support parallel processing of large blocks of data.



3. Multiple instruction stream, single data stream (MISD)

- In MISD, multiple processing units operate on one single-data stream. Each processing unit operates on the data independently via separate instruction stream.
- Systems with MISD stream have number of processing units performing different operations by executing different instructions on the same data set.



4. Multiple instruction stream, multiple data stream (MIMD)

MIMD architecture, each processor in a multiprocessor system can execute different sets of instructions independently on the different set of data set in parallel. The processors can share a common memory of an MIMD, Several types of systems: (1). Uniform Memory Access (UMA). (2). Cache Only Memory Access (COMA). (3). Non-Uniform Memory Access (NUMA).

UMA (Uniform Memory Access):

In this model, all the processors share the physical memory uniformly. All the processors have equal access time to all the memory words. Each processor may have a private cache memory. The peripheral devices follow a set of rules.

Non-uniform Memory Access (NUMA):

In the NUMA multiprocessor model, the access time varies with the location of the memory word. Here, the shared memory is physically distributed among all the processors, called local memories. The collection of all local memories forms a global address space which can be accessed by all the processors.

Cache Only Memory Architecture (COMA):

The COMA model is a specialized version of the NUMA model. Here, all the distributed main memories are converted to cache memories.

Distributed Systems

In distributed systems there is no shared memory and computers communicate with each other through message passing. In distributed computing a single task is divided among different computers. A distributed system is a collection of autonomous computers that are connected through a network and distribution software called middleware

A distributed system has several characteristics:

- The users perceive the system as a single, integrated computing facility.
- The components are autonomous.
- Scheduling and other resource management and security policies are implemented by each system.
- There are multiple points of control and multiple points of failure.
- The resources may not be accessible at all times.
- Can be scaled by adding additional resources.
- Can be designed to maintain availability even at low levels of hardware/software/network reliability.

The **remote procedure call (RPC)** supports inter-process communication and allows a procedure on a system to invoke a procedure running in a different address space, possibly on a remote system. Many programming languages support RPCs; for example, Java Remote Method Invocation (Java RMI) provides functionality similar to that of UNIX RPC methods, and XML-RPC uses XML to encode HTML-based calls.

The middleware should support a set of desirable properties of a distributed system:

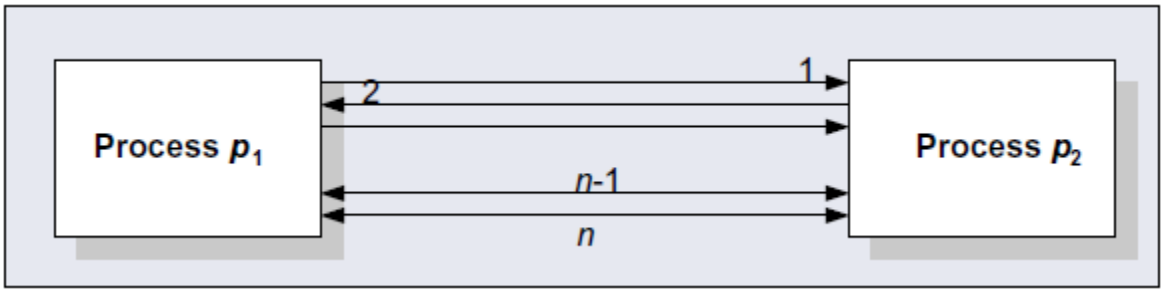
- **Access transparency.** Local and remote information objects are accessed using identical operations.
- **Location transparency.** Information objects are accessed without knowledge of their location.

- **Concurrency transparency.** Several processes run concurrently using shared information objects without interference among them.
- **Replication transparency.** Multiple instances of information objects are used to increase reliability without the knowledge of users or applications.
- **Failure transparency.** The concealment of faults.
- **Migration transparency.** The information objects in the system are moved without affecting the operation performed on them.
- **Performance transparency.** The system can be reconfigured based on the load and quality of service requirements.
- **Scaling transparency.** The system and the applications can scale without a change in the system structure and without affecting the applications.

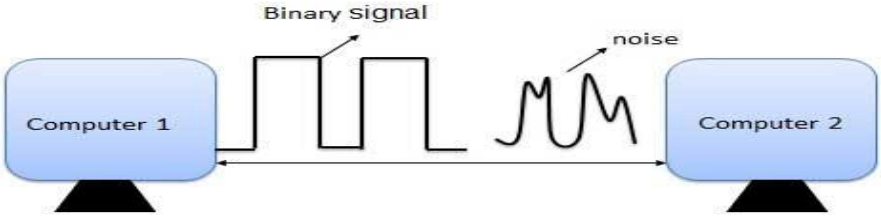
Communication protocols

- A major concern in any parallel and distributed system is communication in the presence of channel failures. There are multiple modes for a channel to fail, and some lead to messages being lost.

- There are multiple modes for a channel to fail, and some lead to messages being lost. In the general case, it is impossible to guarantee that two processes will reach an agreement in case of channel failures



Given two processes p1 and p2 connected by a communication channel that can lose a message with probability ε > 0, no protocol capable of guaranteeing that two processes will reach agreement exists, regardless of how small the probability ε is.



In practice, error detection and error correction codes allow processes to communicate reliably though noisy digital channels. The redundancy of a message is increased by more bits and packaging a message as a codeword.

Communication protocols implement:

- **Error control mechanisms** – using error detection and error correction codes.
- **Flow control** - provides feedback from the receiver, it forces the sender to transmit only the amount of data the receiver can handle.
- **Congestion control** - ensures that the offered load of the network does not exceed the network capacity.

Time and time intervals:

Process coordination requires:

- A global concept of time shared by cooperating entities.
- The measurement of time intervals, the time elapsed between two events

- Two events in the global history may be unrelated, neither one is the cause of the other; such events are said to be concurrent events.
- Local timers provide relative time measurements. An isolated systemν can be characterized by its history expressed as a sequence of events, each event corresponding to a change of the state of the system.
- Global agreement on time is necessary to trigger actions that shouldν occur concurrently. Timestamps are often used for event ordering using a global timeν base constructed on local virtual clocks.

System specification, design, and analysis require a clear understanding of cause-effect relationships. The activity of any process is modeled as a sequence of events; hence, the binary relation cause-effect relationship should be expressed in terms of events and should express our intuition that the cause must precede the effects.

The binary cause-effect relationship between two events has the following properties:

Two events in the global history may be unrelated. If so, neither one is the cause of the other; such events are said to be concurrent events.

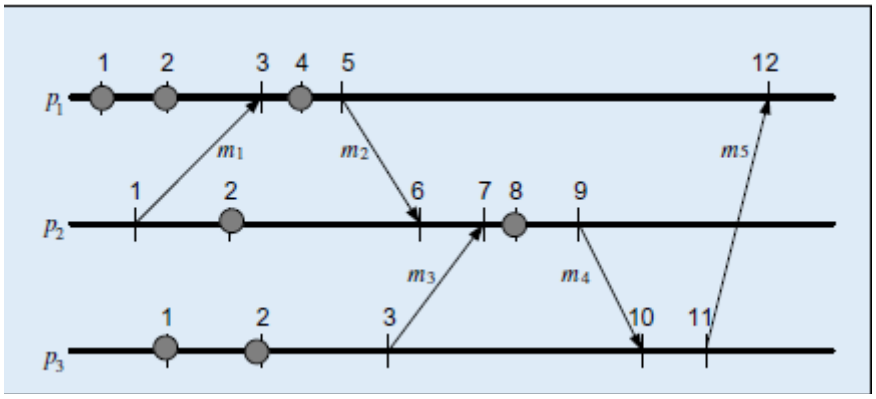
Logical clocks

A logical clock (LC) is an abstraction necessary to ensure the clock condition in the absence of a global clock. Each process pi maps events to positive integers.

Call LC(e) the local variable associated with event e. Each process time stamps each message m sent with the value of the logical clock at the time of sending, T S(m) = LC(send(m)). The rules to update the logical clock are specified by the following relationship:

$$LC(e) = \begin{cases} LC + 1 & \text{if} \quad e \text{ is a local event or a } send(m) \text{ event} \\ \max(LC, TS(m) + 1) & \text{if} \quad e = receive(m). \end{cases}$$

The concept of logical clocks is illustrated in below Figure using a modified space-time diagram in which the events are labeled with the logical clock value. Messages exchanged between processes are shown as lines from the sender to the receiver; the communication events corresponding to sending and receiving messages are marked on these diagrams.



Each process labels local events and sends events sequentially until it receives a message marked with a logical clock value larger than the next local logical clock value. It follows that logical clocks do not allow a global ordering of all events.

For example, there is no way to establish the ordering of events e^1_1, e^1_2, and e^1_3 in above Figure Nevertheless, communication events allow different processes to coordinate their logical clocks; for example, process p_2 labels the event e^3_2 as 6 because of message m_2, which carries the information about the logical clock value as 5 at the time message m_2 was sent. Recall that e^j_i is the j -th event in process p_i .

Logical clocks lack an important property, gap detection; given two events e and e' and their logical clock values, LC(e) and LC(e'), it is impossible to establish if an event e" exists such that

$$LC(e) < LC(e'') < LC(e').$$

Message delivery rules; causal delivery

Message receiving and message delivery are two distinct operations; a *delivery rule* is an additional assumption about the channel-process interface. This rule establishes when a message received is actually delivered to the destination process. The receiving of a message *m* and its delivery are two distinct events in a causal relation with one another.

$$receive(m) \rightarrow deliver(m)$$

First In, First Out (FIFO) delivery implies that messages are delivered in the same order in which they are sent. For each pair of source-destination processes (p_i , p_j), FIFO delivery requires that the following relation should be satisfied:

$$send_i(m) \rightarrow send_i(m') \Rightarrow deliver_j(m) \rightarrow deliver_j(m')$$

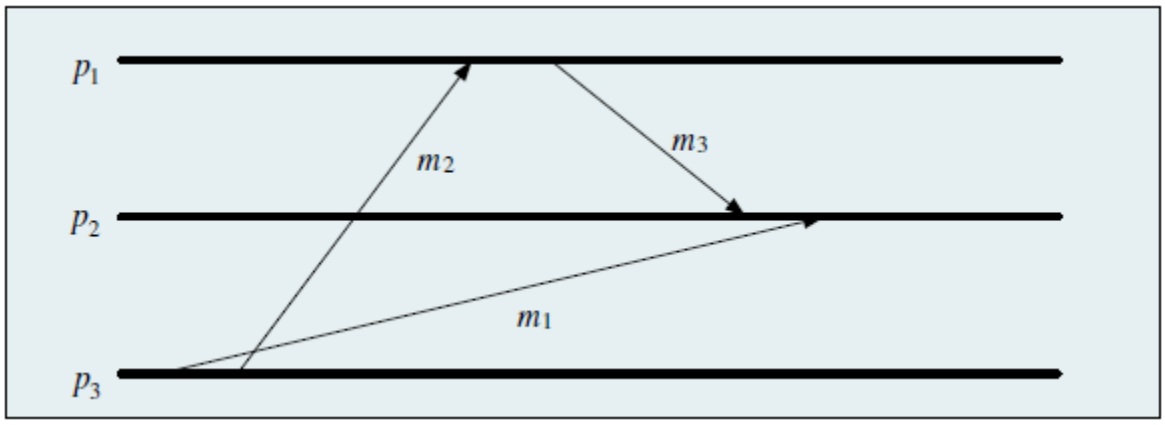
Even if the communication channel does not guarantee FIFO delivery, FIFO delivery can be enforced by attaching a sequence number to each message sent. The sequence numbers are also used to reassemble messages out of individual packets.

Causal delivery is an extension of the FIFO delivery to the case when a process receives messages from different sources. Assume a group of three processes, (p_i , p_j, p_k) and two messages m and m'. Causal delivery requires that

$$send_i(m) \rightarrow send_j(m') \Rightarrow deliver_k(m) \rightarrow deliver_k(m')$$

When more than two processes are involved in a message exchange, the message delivery may be FIFO but not causal

- $deliver(m_3) \rightarrow deliver(m_1)$, according to the local history of process p_2.
- $deliver(m_2) \rightarrow send(m_3)$, according to the local history of process p_1.
- $send(m_1) \rightarrow send(m_2)$, according to the local history of process p_3.
- $send(m_2) \rightarrow deliver(m_2)$.
- $send(m_3) \rightarrow deliver(m_3)$.



Call *T S(m)* the *time stamp* carried by message *m*. A message received by process *pi* is *stable* if no future messages with a time stamp smaller than *T S(m)* can be received by process *pi* . When logical clocks are used, a process *pi* can construct consistent observations of the system if it implements the following delivery rule: *Deliver all stable messages in increasing time-stamp order.*

For any two events, e and e', occurring in different processes, the so-called clock condition is satisfied if

$$e \rightarrow e' \Rightarrow RC(e) < RC(e'), \quad \forall e, e'.$$

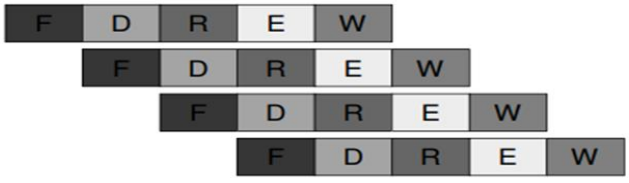
Concurrency

Concurrency means that several activities are executed simultaneously. Concurrency allows us to reduce the execution time of a data-intensive problem. Concurrency is a critical element of the design of system software. The kernel of an operating system exploits concurrency for virtualization of system resources such as the processor and the memory.

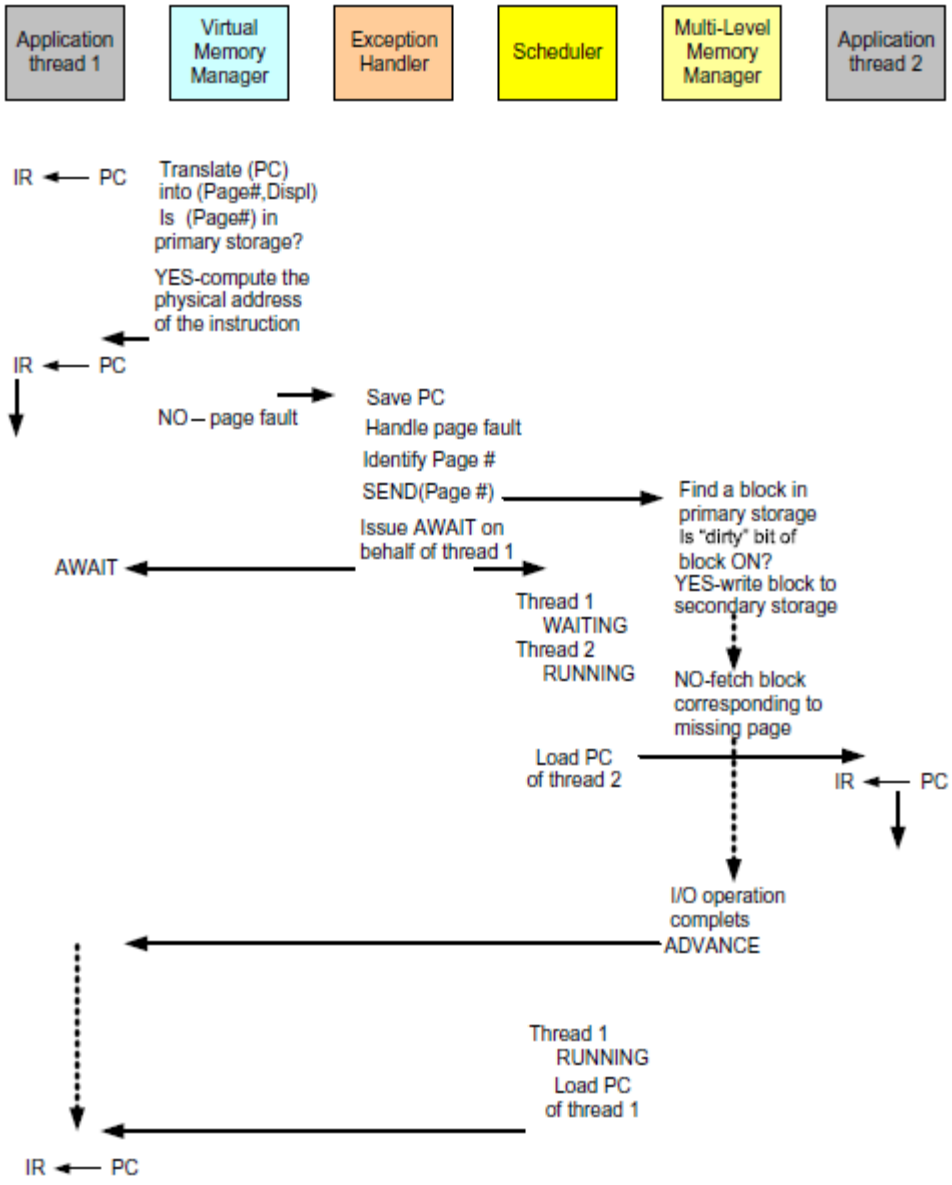
- Hiding latency and performance enhancement (e.g., schedule a ready-to-run thread when the current thread is waiting for the completion of an I/O operation).
- Avoiding limitations imposed by the physical resources (e.g., allow an application to run in a virtual address space of a standard size rather than be restricted by the physical memory available on a system).
- Enhancing reliability and performance, as in the case of RAID systems.

Concurrency is often motivated by the desire to enhance system performance. For example, in pipelined computer architecture, multiple instructions are in different phases of execution at any given time. Once the pipeline is full, a result is produced at every pipeline cycle; an *n*-stage pipeline could potentially lead to a speed-up by a factor of *n*.

- Fetch instruction from memory
- Decode the instruction
- Read data from registers
- Execute the instruction
- Write the result into a register



Context switching could involve multiple components of an OS kernel, including the Virtual Memory Manager (VMM), the Exception Handler (EH), the Scheduler (S), and the Multilevel Memory Manager (MLMM). When a page fault occurs during the fetching of the next instruction, multiple context switches are necessary,



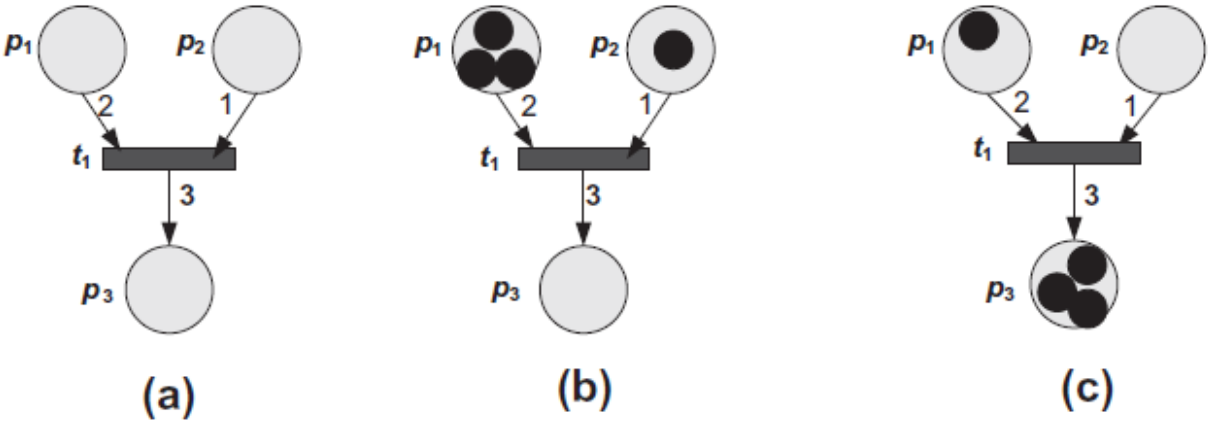
Communication channels allow concurrent activities to work in concert and to coordinate. Communication protocols allow us to transform noisy and unreliable channels into reliable ones that deliver messages in order.

Multiple instances of a cloud application, a server and the clients of the service it provides, and many other applications communicate via message passing. The Message Passing Interface (MPI) supports both synchronous and asynchronous communication, and it is often used by parallel and distributed applications.

Model concurrency with Petri Nets

A Petri net, also known as a place/transition (PT) net, is one of several mathematical modeling languages for the description of distributed systems. PNs are bipartite graphs populated with tokens that flow through the graph that are used to model the dynamic rather than static behavior of systems

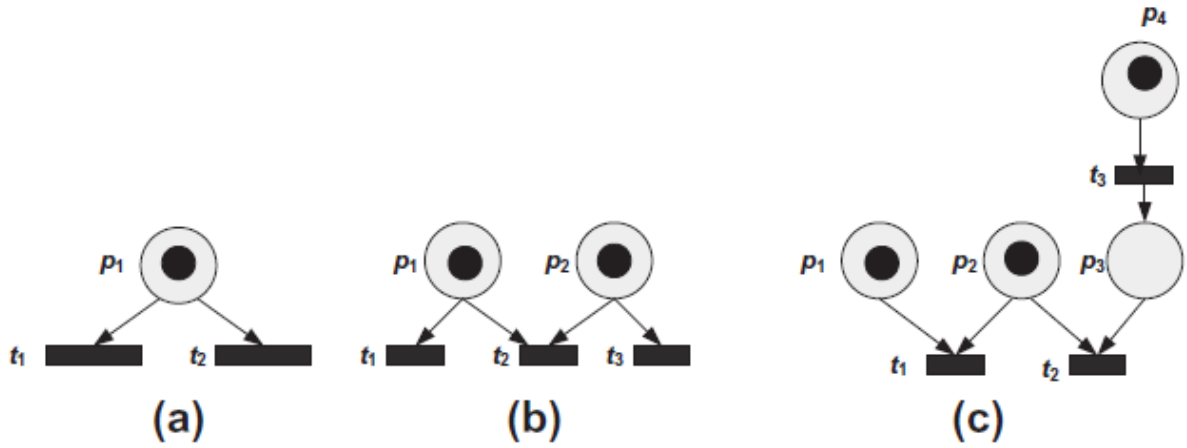
- A bipartite graph is one with two classes of nodes; arcs always connect a node in one class with one or more nodes in the other class.
- In the case of Petri nets the two classes of nodes are **places and transitions**; thus, the name place-transition (P/T) nets is often used for this class of bipartite graphs.
- Arcs connect one place with one or more transitions or a transition with one or more places.
- To model the dynamic behavior of systems, the places of a Petri net contain tokens. Firing of transitions removes tokens from the input places of the transition and adds them to its output places.
- Petri nets can model different activities in a distributed system. A transition may model the occurrence of an event, the execution of a computational task, the transmission of a packet, a logic statement, and so on.
- The input places of a transition model the pre-conditions of an event, the input data for the computational task, the presence of data in an input buffer, or the pre-conditions of a logic statement.
- The output places of a transition model the post-conditions associated with an event, the results of the computational task, the presence of data in an output buffer, or the conclusions of a logic statement.



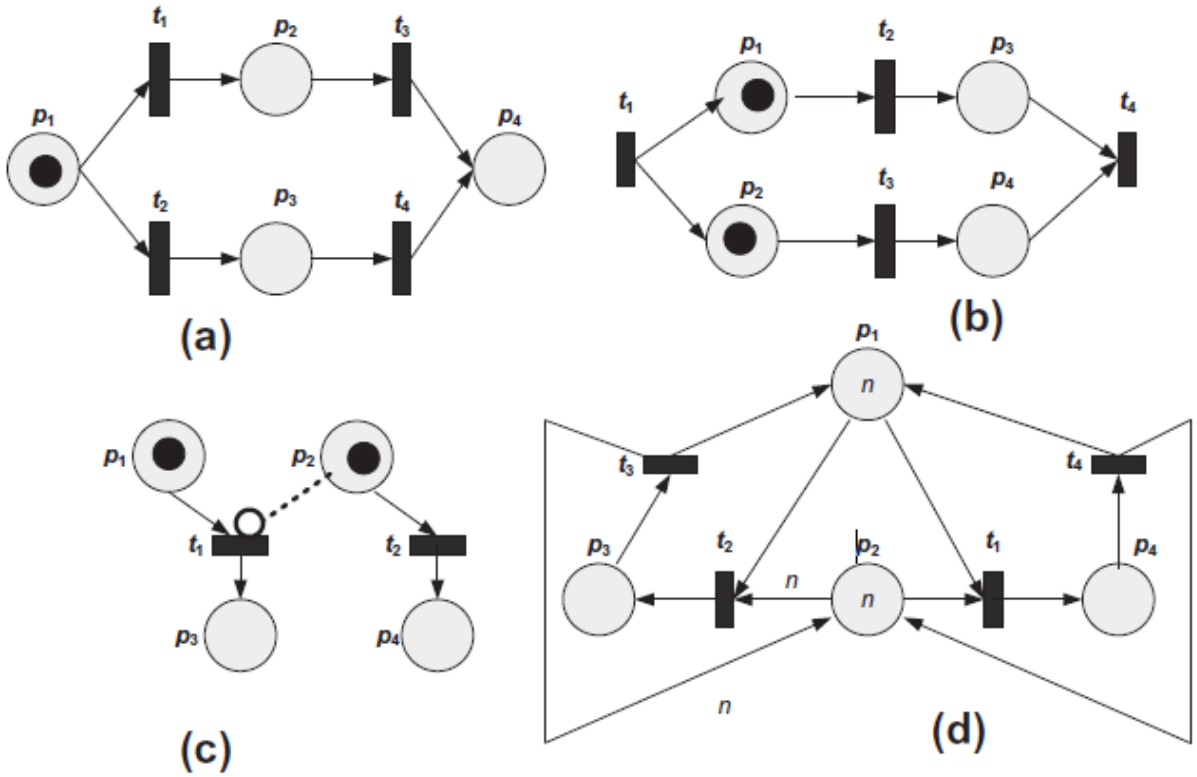
Petri nets, firing rules:

(a) An unmarked net with one transition *t*1 with two input places, *p*1 and *p*2, and one output place, *p*3.

(b) The marked net, the net with places populated by tokens; the net before firing the enabled transition t1.
(c) The marked net after firing transition t1. Two tokens from place p1 and one from place p2 are removed and transported to place p3.



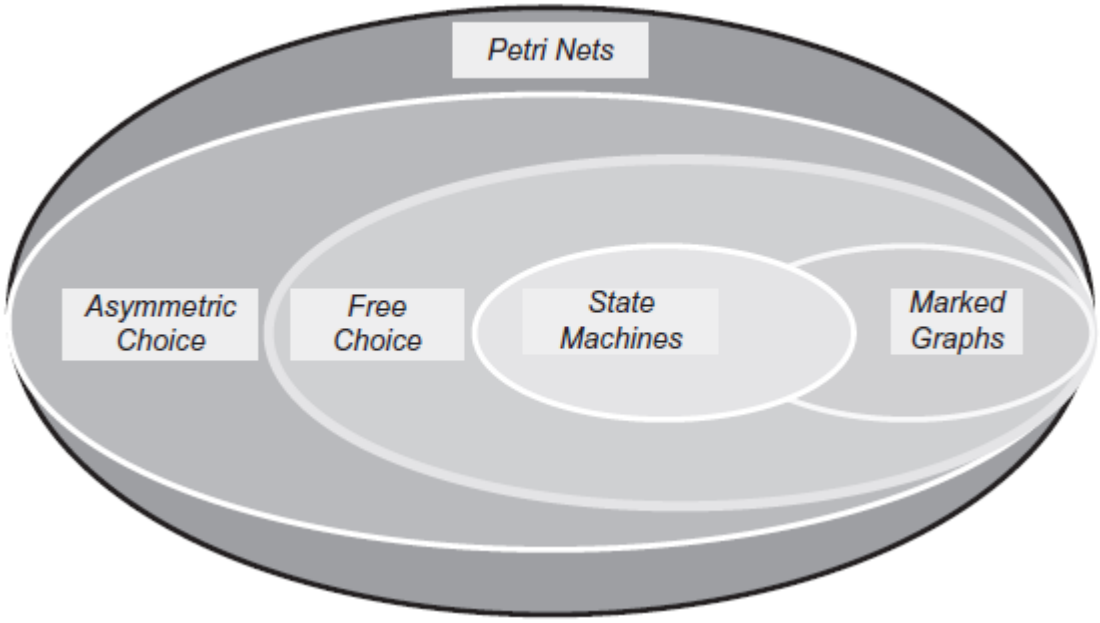
Petri nets modeling.
(a) Choice: Only one of transitions *t*1, or *t*2 may fire.
(b) Symmetric confusion: Transitions *t*1 and *t*3 are concurrent and, at the same time, they are in conflict with *t*2. If *t*2 fires, then *t*1 and/or *t*3 are disabled.
(c) Asymmetric confusion: Transition *t*1 is concurrent with *t*3 and is in conflict with *t*2 if *t*3 fires before *t*1.



The concurrent transitions *t*2 and *t*3 in Figure (a) model concurrent execution of two processes. A *marked graph* can model concurrency but not choice; transitions *t*2 and *t*3 in Figure (b) are concurrent, so there is no causal relationship between them. Transition *t*4 and its input places *p*3 and *p*4 in Figure (b) model synchronization; *t*4 can only fire if the conditions associated with *p*3 and *p*4 are satisfied.

Petri nets can be used to model *priorities*. The net in Figure (c) models a system with two processes modeled by transitions *t*1 and *t*2; the process modeled by *t*2 has a higher priority than the one modeled by *t*1. If both processes are ready to run, places *p*1 and *p*2 hold tokens. When the two processes are ready, transition *t*2 will fire first, modeling the activation of the second process. Only after *t*2 is activated will transition *t*1 fire, modeling the activation of the first process.

Petri nets are able to model exclusion. For example, the net in Figure (d), models a group of n concurrent processes in a shared-memory environment. At any given time only one processmay write, but any subset of the n processes may read at the same time, provided that no process writes. Place p3 models the process allowed to write, p4 the ones allowed to read, p2 the ones ready to access the shared memory, and p1 the running tasks.



State Machine:

A Petri net is a *state machine* if and only if

$$\forall t_i \in t \text{ then } (|\bullet t_i| = 1 \wedge |t_i \bullet| = 1)$$

All transitions of a state machine have exactly one incoming and one outgoing arc. This topological constraint limits the expressiveness of a state machine, so no concurrency is possible.

Marked Graph:

A Petri net is a marked graph if and only if ∀pi ∈ p then (|• pi| = 1∧|pi •| = 1). In a marked graph each place has only one incoming and one outgoing flow relation; thus, marked graphs do no not allow modeling of choice.

Free Choice:

Extended Free Choice, and Asymmetric Choice Petri Nets. The marked net, (N, s_0) with N = (p, t, f , l) is a free-choice net if and only if

$$(\bullet t_i) \cap (\bullet t_j) = \emptyset \Rightarrow |\bullet t_i| = |\bullet t_j| \quad \forall t_{i,j} \in t.$$

N is an *extended free-choice net* if $\forall t_i, t_j \in t$ then $(\bullet t_i) \cap (\bullet t_j) = \emptyset \Rightarrow \bullet t_i = \bullet t_j$.
N is an *asymmetric choice net* if and only if $(\bullet t_i) \cap (\bullet t_j) \neq \emptyset \Rightarrow (\bullet t_i \subseteq \bullet t_j)$ or $(\bullet t_i \supseteq \bullet t_j), \forall t_i, t_j \in t$.