Jisy Raju
Assistant Professor, CE Cherthala

Module 2

System models: Physical models - Architectural models -Fundamental models

2. System Models:

It describe **common properties** and **design choice of dispatcher** for distributed system in a single descriptive model.

Three types of models

- **Architectural Models**
- **Fundamental Models**
- **Physical Models**

2.1 Architectural Models:

Architecture models define the main components of the system, what their roles are and how they interact (software architecture), and how they are deployed in an underlying network of computers (system architecture). Architecture model is concerned with the placement of its parts, namely how components are mapped to underlying network and the relationship between them, that is, their functional roles and patterns of communication between them.

Architectural Model- including,

- System Architectures
- Architectural elements
- Software layers
- Variations on the client-server model

2.1.1 Architectural elements

To understand the fundamental building blocks of a distributed system, it is necessary to consider four key questions:

• What are the entities that are communicating in the distributed system?

• How do they communicate, or, more specifically, what communication paradigm is used?

• What (potentially changing) roles and responsibilities do they have in the overall architecture?

• How are they mapped on to the physical distributed infrastructure (what is their

placement)?

Communicating entities: what is communicating and how those entities communicate together define a rich design space for the distributed systems developer to consider. It is helpful to address the first question from a system-oriented and a problem-oriented perspective.

From a system perspective, the answer is normally very clear in that the entities that communicate in a distributed system are typically processes, leading to the prevailing view of a distributed system as processes coupled with appropriate inter-process communication paradigms.

From a programming perspective, however, this is not enough, and more problem-oriented abstractions have been proposed:

Objects: Objects have been introduced to enable and encourage the use of object oriented approaches in distributed systems (including both object-oriented design and object-oriented programming languages).

Components: Since their introduction a number of significant problems have been identified with distributed objects, and the use of component technology has emerged as a direct response to such weaknesses. Components resemble objects in that they offer problem-oriented abstractions for building distributed systems and are also accessed through interfaces.

Web services: Web services represent the third important paradigm for the development of distributed systems. Web services are closely related to objects and components, again taking an approach based on encapsulation of behavior and access through interfaces.

Communication paradigms: deals with how entities communicate in a distributed system, and consider three communication paradigm:

- Inter-process communication
- Remote invocation
- Indirect communication.

Inter-process communication refers to the relatively low-level support for communication between processes in distributed systems, including message-passing primitives, direct access to the API offered by Internet protocols (socket programming) and support for multicast communication.

Remote invocation represents the most common communication paradigm in distributed systems, covering a range of techniques based on a two-way exchange between communicating entities in a distributed system and resulting in the calling of a remote operation (*Request-reply protocols).*Request-reply protocols are effectively a pattern

imposed on an underlying procedure or method.

Request-reply protocols: Request-reply protocols are effectively a pattern imposed on an underlying message-passing service to support client-server computing. In particular, such protocols typically involve a pairwise exchange of messages from client to server and then from server back to client, with the first message containing an encoding of the operation to be executed at the server and also an array of bytes holding associated arguments and the second message containing any results of the operation,

-Remote procedure calls: In RPC, procedures in processes on remote computers can be called as if they are procedures in the local address space. The underlying RPC system then hides important aspects of distribution, including the encoding and decoding of parameters and results, the passing of messages and the preserving of the required semantics for the procedure call.

-Remote method invocation: Remote method invocation (RMI) strongly resembles remote procedure calls but in a world of distributed objects. With this approach, a calling object can invoke a method in a remote object. As with RPC, the underlying details are generally hidden from the user.

Indirect communication through a third entity, allowing a strong degree of decoupling between senders and receivers. In particular:

- Senders do not need to know who they are sending to (space uncoupling).
- Senders and receivers do not need to exist at the same time (time uncoupling).

Key techniques for indirect communication include:

Group communication: Group communication is concerned with the delivery of messages to a set of recipients and hence is a multiparty communication paradigm supporting one-to-many communication. Group communication relies on the abstraction of a group which is represented in the system by a group identifier.

Publish-subscribe systems: Many systems, such as the financial trading example can be classified as information-dissemination systems wherein a large number of producers (or publishers) distribute information items of interest (events) to a similarly large number of consumers (or subscribers).

Publish-subscribe systems all share the crucial feature of providing an intermediary service that efficiently ensures information generated by producers is routed to consumers who desire this information.

Message queues: Whereas publish-subscribe systems offer a one-to-many style of communication; message queues offer a point-to-point service whereby producer processes

can send messages to a specified queue and consumer processes can receive messages from the queue or be notified of the arrival of new messages in the queue.

Tuple spaces: Tuple spaces offer a further indirect communication service by supporting a model whereby processes can place arbitrary items of structured data, called tuples, in a persistent tuple space and other processes can either read or remove such tuples from the tuple space by specifying patterns of interest.

Distributed shared memory: Distributed shared memory (DSM) systems provide an abstraction for sharing data between processes that do not share physical memory. Programmers are nevertheless presented with a familiar abstraction of reading or writing (shared) data structures as if they were in their own local address spaces, thus presenting a high level of distribution transparency.

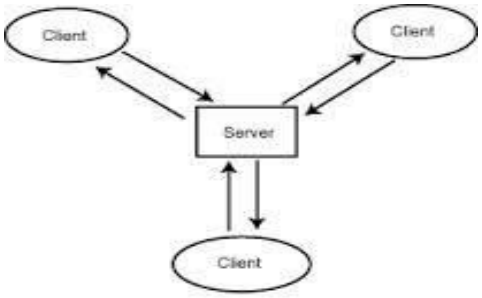
2.1.2 System Architectures: deals with the roles and responsibility

- Client- server model
- Services provided by multiple servers.
- Proxy servers and caches.
- Peer processes

Client- server model

The system is structured as a set of processes, called servers, that offer services to the users , called clients. The client-server model is usually based on a simple request/reply protocol , implemented with send/receive.

- The client sends a request message to the server asking for some service.
- The server does the work and return a result or error code if the work could not be performed.

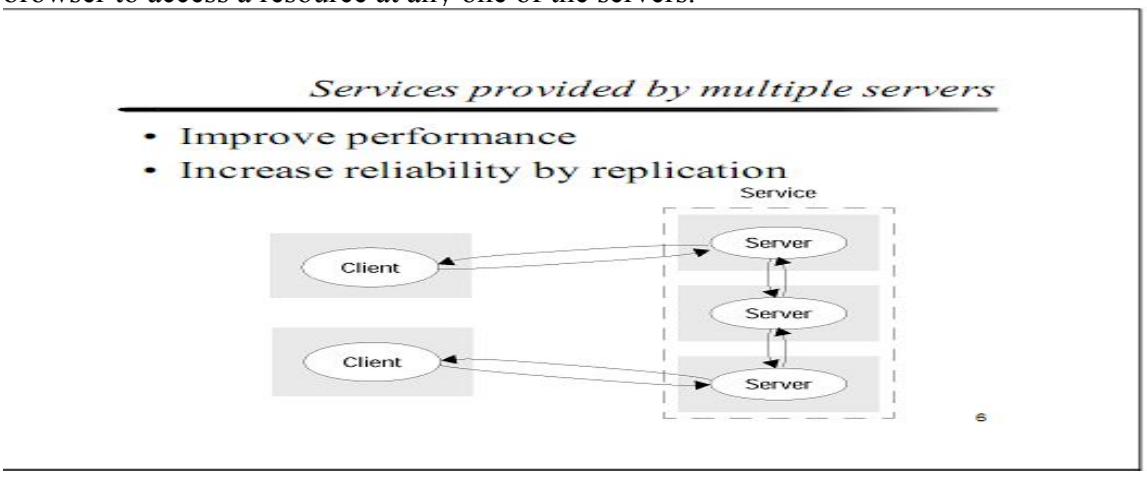


Web servers and most other Internet services are clients of the DNS service, which translates Internet domain names to network addresses. Another web-related example concerns search engines, which enable users to look up summaries of information available on web pages at sites throughout the Internet. These summaries are made by programs called web crawlers, which run in the background at a search engine site using HTTP requests to access web servers throughout the Internet.

Services provided by multiple servers

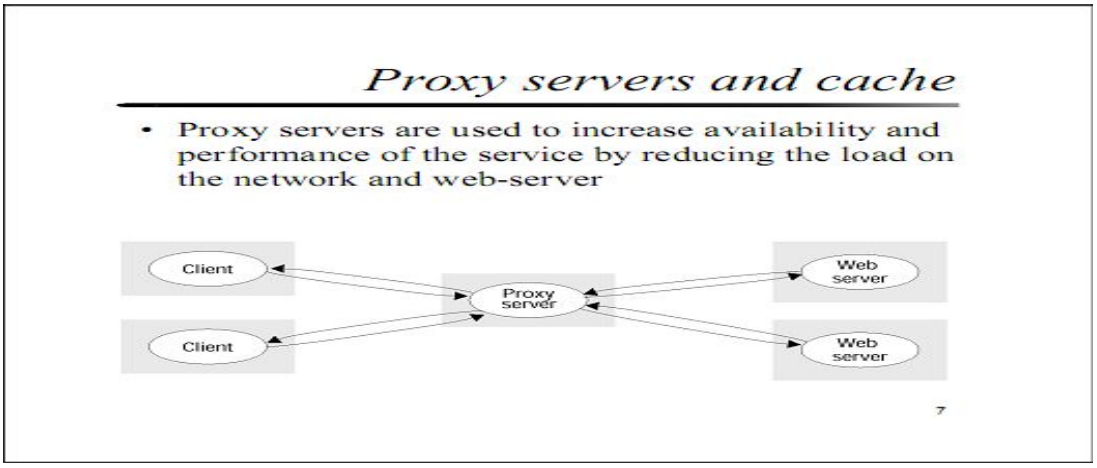
Services may be implemented as several server processes in separate host computers interacting as necessary to provide a service to client processes. The Web provides a common

example of partitioned data in which each web server manages its own set of resources. A user can employ a browser to access a resource at any one of the servers.



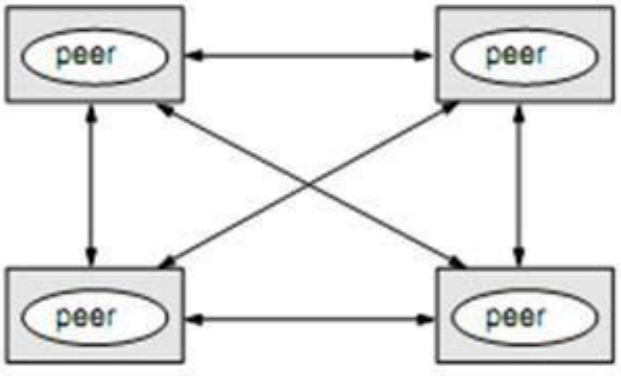
Proxy servers and caches

Proxy servers are used to increase availability and performance of the services by reducing the load on the network and web-server. Proxy server provides copies(replications) of resources which are managed by other server. Web browsers maintain a cache of recently visited web pages and other web resources in the client's local file system, using a special HTTP request to check with the original server that cached pages are up-to-date before displaying them.



Peer processes

All processes(objects) play similar roles without distinction between client or servers. It distributes shared resources widely and it share computing and communication loads.



2.1.3 Variations on the client-server model(mapping)

- Mobile code
- Mobile agent
- Thin client
- Network computers

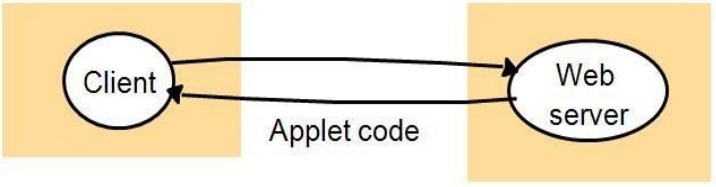
Mobile code: It is used to refer to code that can be sent from one computer to another and run at the destination. Its advantage is remote invocations are replaced by local ones so no need to suffer from the delays. Example : java applets

Step : 1

The user running a browser selects a link to an applets whose code is stored on a web server.

The code is downloaded to the browser and runs there.

client request results in the downloading of applet code



Step : 2

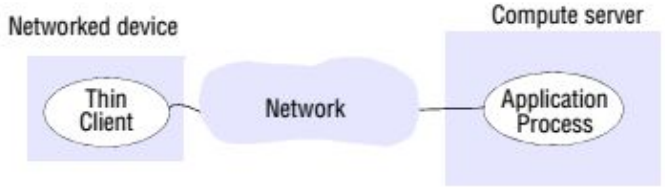
Client interacts with the applet.

client interacts with the applet

Mobile agent: It is a running program that travels from one computer to another carrying out a task to someone's behalf , such as collecting information, eventually returning with the results

Thin client: A thin client is a lightweight computer that establish a remote connection with a server-based computing environment. This architecture has low management and hardware cost. Here instead of downloading applications into user's computer, it runs on computer server.

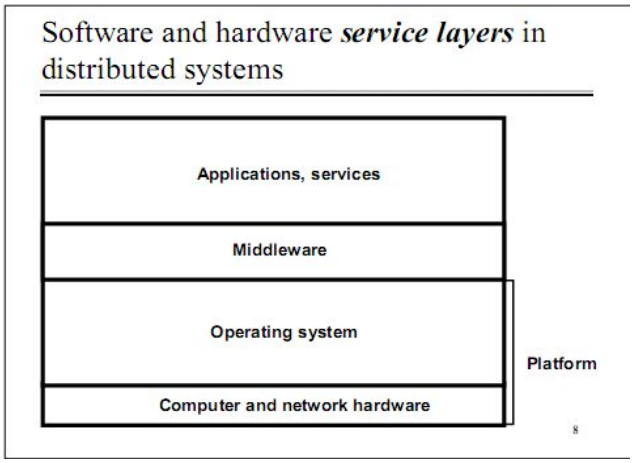
Figure 2.10 Thin clients and computer servers



Network computers: It downloads its operating system and any application software needed by the user from a remote server. Applications are run locally but files are managed by remote file server.

2.1.4 Software layers:

The layered architecture expressed in term of service layers offers a software abstraction, with higher layers being unaware of implementation details, or indeed of any other layers beneath them.



Platform for distributed systems and applications consists of the lowest-level hardware and software layers. These low-level layers provide services to the layers above them, which are implemented independently in each computer, bringing the system's programming interface up to a level that facilitates communication and coordination between processes.

Intel x86/Windows, Intel x86/Solaris, Intel x86/Mac OS X, Intel x86/Linux and ARM/Symbian are major examples.

Middleware was defined as a layer of software whose purpose is to mask heterogeneity and to provide a convenient programming model to application programmers. The activities of amiddleware such as:

- Remote method invocation
- Communication between a group of processes
- Notification of events
- The partitioning, placement and retrieval of shared data objects amongst cooperating computers;
- The replication of shared data objects;
- The transmission of multimedia data in real time.

Tiered architecture -Tiering is a technique to organize functionality of a given layer and place this functionality into appropriate servers and, as a secondary consideration, on to physical nodes.

The concepts of two- and three-tiered architecture are:

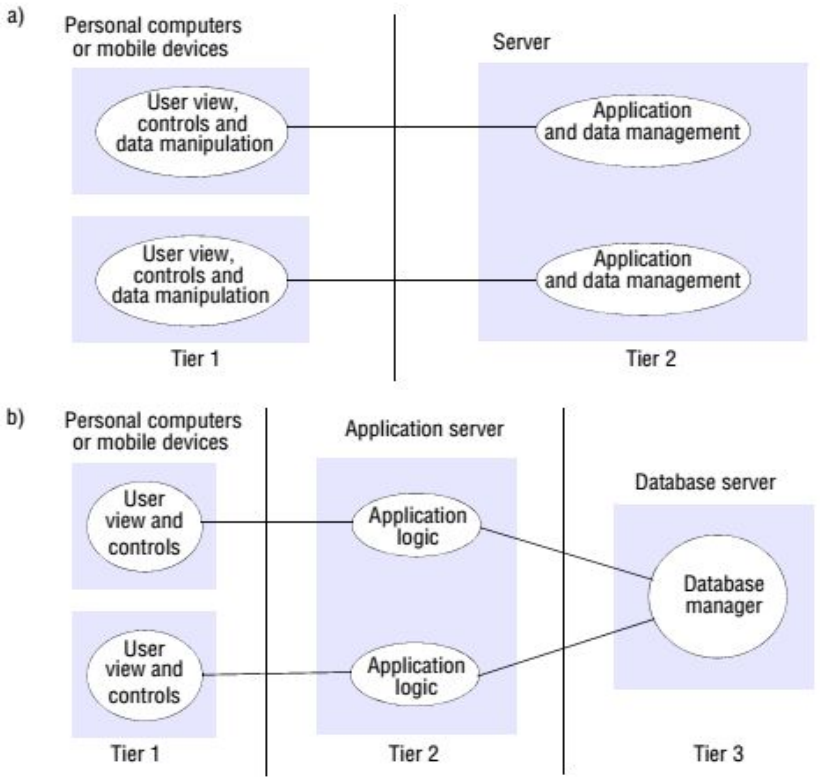
- **The presentation logic**, which is concerned with handling user interaction and updating the view of the application as presented to the user.
- **The application logic**, which is concerned with the detailed application-specific processing associated with the application (also referred to as the business logic, although the concept is

not limited only to business applications).

- **The data logic**, which is concerned with the persistent storage of the application, typically in a database management system.

Comparison between two-tier and three-tier architecture:

Figure 2.8 Two-tier and three-tier architectures



2.2 Fundamental Models:

Fundamental models deal with formal description of the properties that is common to architecture

models. Specific about the characteristic and the failures and security risks they might exhibit.

- Interaction Models – processes interact by passing msgs and coordination b/w them.
- Failure Models – defines and classifies the failures.
- Security Models – modular nature and openness of the DS exposes it to the external and internal attacks .

2.2.1 Interaction Models:

Computation occurs within processes; the processes interact by passing messages, resulting in communication (information flow) and coordination (synchronization and ordering of activities) between processes. Interacting processes in a distributed system are affected by two significant factors:

1. Performance of communication channels

- **Latency:** delay between sending and receipt of a message
- **Jitter:** *jitter* is the deviation from true periodicity of a presumably periodic signa
- **Throughput:** No. of Packet send per unit time
- **Bandwidth:** total no. of information send per unit time

2.Computer clocks:

Each computer in a distributed system has its own internal clock to supply the value of the current time to local processes. Therefore, two processes running on different computers read their clocks at the same time may take different time values. Clock drift rate refers to the relative amount a computer clock differs from a perfect reference clock.

Two variants of the interactive model:

-Synchronous distributed systems

-Asynchronous distributed systems

Synchronous distributed systems:

- The time to execute each step of a process has known lower and upper bounds.

• Each message transmitted over a channel is received within a known bounded time.

• Each process has a local clock whose drift rate from real time has a known bound.

Asynchronous distributed systems:A system in which there are no bounds on:

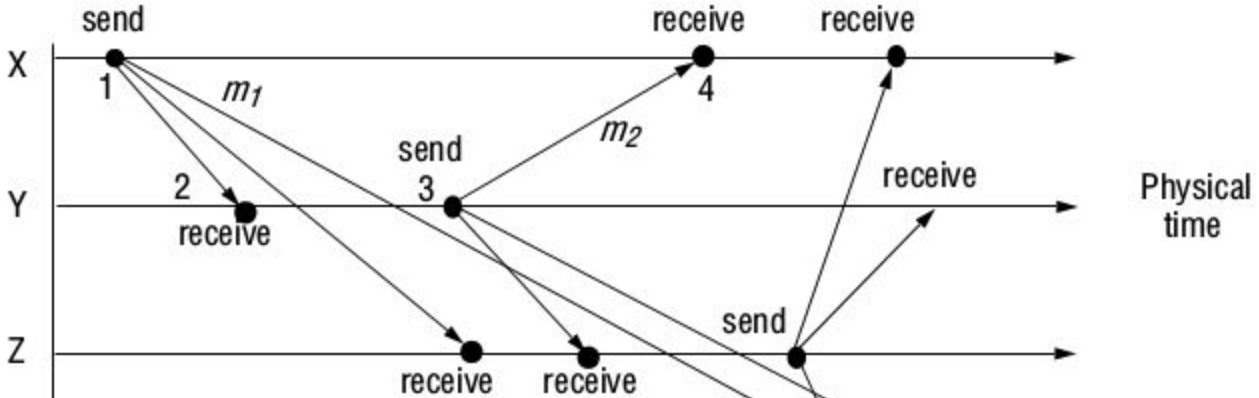
- process execution times.
- message delivery times.
- clock drift rate.

Event ordering:

Consider the following set of exchanges between a group of email

- users, X, Y, Z and A, on a mailing list:
    1. User X sends a message with the subject Meeting.
    2. Users Y and Z reply by sending a message with the subject Re: Meeting.

In real time, X's message is sent first, and Y reads it and replies; Z then reads both X's message and Y's reply and sends another reply, which references both X's and Y's messages. But due to the independent delays in message delivery, the messages may be delivered as shown in Figure.



		Inbox:
Item	*From*	*Subject*
23	Z	Re: Meeting
24	X	Meeting
25	Y	Re: Meeting

If clocks cannot be synchronized perfectly across a distributed system, Lamport proposed a model of logical time that can be used to provide an ordering among the events at processes running in different computers in a distributed system. Logically, we know that a message is received after it was sent. Therefore we can state a logical ordering for pairs of events shown in Figure 2.13, for example,

considering only the events concerning X and Y:

X sends m 1 before Y receives m 1 ;

Y sends m 2 before X receives m 2 .

We also know that replies are sent after receiving messages, so we have the following logical ordering for Y:

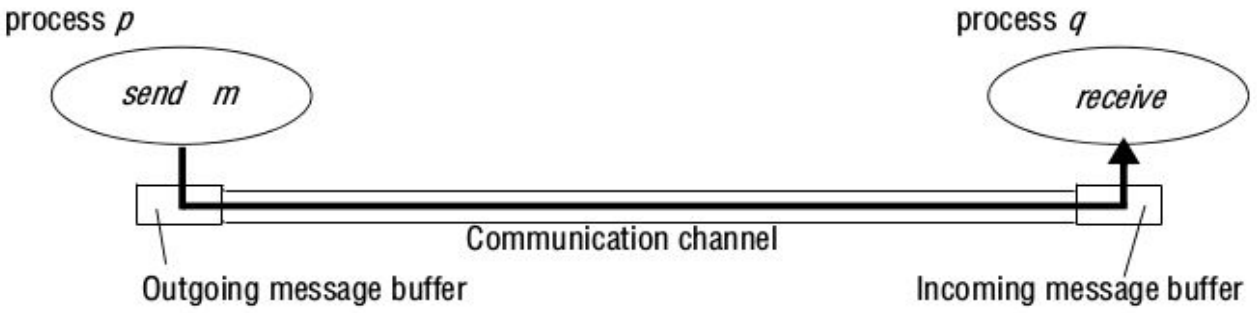
Y receives m 1 before sending m 2 .

Logical time takes this idea further by assigning a number to each event corresponding to its logical ordering, so that later events have higher numbers than earlier ones. For example, Figure shows the numbers 1 to 4 on the events at X and Y.

2.2.2 Failure model

The failure model attempts to give a precise specification of the faults that can be exhibited by processes and communication channels. Failure may occur in order to provide an understanding of its effects, Including**,**

1. Omission Failures : Process or channel failed to do something. The chief omission failure of a process is crash and dropping message. When we say that a process has crashed we mean that it has halted and will not execute any further steps of its program ever. The communication channel produces an omission failure if it does not transport a message from p's outgoing message buffer to q's incoming message buffer. This is known as 'dropping messages'



2. Arbitrary Failures/ Byzantine failure: Any Security Models type of error can occur in processes or channels. For example, a process may set wrong values in its data items, or it may return a wrong value in response to an invocation. An arbitrary failure of a process is one in which it arbitrarily omits intended processing steps or takes unintended processing steps

3. Timing Failures: Applicable only to synchronous distributed systems where time limits may not be met. Time limits are set to processes execution, communications and clock drifts rate. A timing faults occurs if any of this time limits exceeded.

4. Masking Failures:A service masks a failure by hiding it or converting it into a more acceptable type of failure. Checksums are used to mask corrupting messages -> a corrupted message is handled as a missing message. Message omission failures can be hidden by retransmitting messages.

The term reliable communication is defined in terms of validity and integrity as follows:

-Validity: Any message in the outgoing message buffer is eventually delivered to the incoming

message buffer.

-Integrity: The message received is identical to one sent, and no messages are delivered twice.

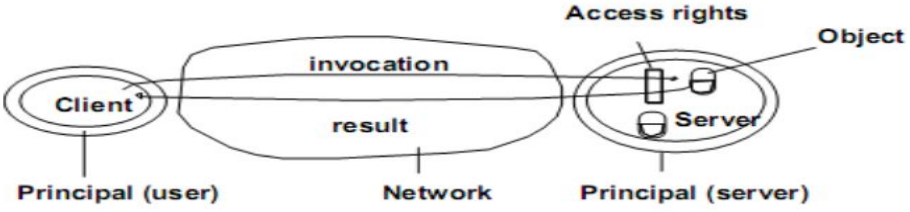
2.2.3 Security Models

Security of a distributed system can be achieved by securing the processes and the channels used for their interactions and by protecting the objects that they encapsulate against unauthorized access. Security model includes,

- **Protecting Objects**

1.Access Rights: who is allowed to perform the operations of an object.

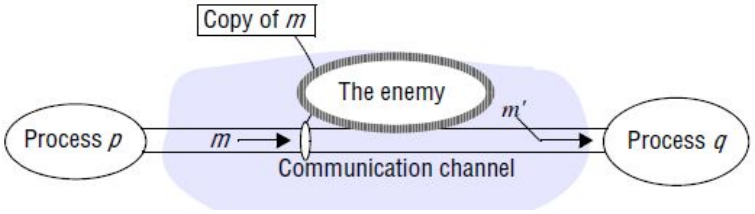
2. Principal: the authority who has some rights on the object.



- **Securing processes and their interactions.**

1.The enemy: The threats from a potential enemy include threats to processes and threats to

communication channels. Threats to processes: A process that is designed to handle incoming requests may receive a message from any other process in the distributed system, and it cannot necessarily determine the identity of the sender. Threats to communication channels: An enemy can copy, alter or inject messages as they travel across the network and its intervening gateways. Such attacks present a threat to the privacy and integrity of information as it travels over the network and to the integrity of the system.



2. Defeating security threats

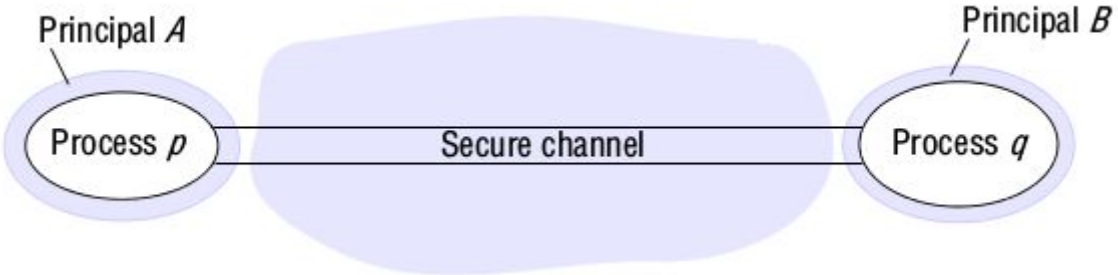
- **Cryptography and shared secrets:** cryptography is the science of keeping messages secure, and encryption is the process of scrambling a message in such a way as to hide its contents.
- **Authentication:** The use of shared secrets and encryption provides the basis for the

authentication of messages – proving the identities supplied by their senders.

- **Secure channels:** Encryption and authentication are used to build secure channels as a service layer on top of existing communication services. A secure channel is a communication channel connecting a pair of processes, each of which acts on behalf of a principal.

A secure channel has the following properties:

- Each of the processes knows reliably the identity of the principal on whose behalf the other process is executing. Therefore if a client and server communicate via a secure channel
- A secure channel ensures the privacy and integrity (protection against tampering) of the data transmitted across it.
- Each message includes a physical or logical timestamp to prevent messages from being replayed or reordered



3. Other possible threats from an enemy

Two further security threats – denial of service attacks and the deployment of mobile code.

- ***Denial of service:*** This is a form of attack in which the enemy interferes with the activities of authorized users by making excessive and pointless invocations on services or message transmissions in a network, resulting in overloading of physical resources
- ***Mobile code:*** Mobile code raises new and interesting security problems for any process that receives and executes program code from elsewhere, such as the email attachment. Such code may easily play a Trojan horse role, purporting to fulfil an innocent purpose .

The uses of security models

The use of security techniques such as encryption and access control incurs substantial processing and management costs.

2.3 Physical model

Baseline of physical model is, a distributed system one in which hardware or software components located at networked computers communicate and coordinate their actions only by passing messages. This leads to a minimal physical model of a distributed system as an extensible set of computer nodes interconnected by a computer network for the required passing of messages.

Three generations of distributed systems.

- Early distributed system
- Internet-scale distributed systems
- Contemporary distributed systems

2.3.1 Early distributed systems

Such systems emerged in the late 1970s and early 1980s in response to the emergence of local area networking technology. These systems typically consisted of between 10 and 100 nodes interconnected by a local area network, with limited Internet connectivity and supported a small range of services. Individual systems were largely homogeneous and openness was not a primary concern. Providing quality of service was still very much in its infancy and was a focal point for much of the research around such early systems.

2.3.2 Internet-scale distributed systems

Larger-scale distributed systems started to emerge in the 1990s in response to the dramatic growth of the Internet (for example, the Google search engine was first launched in 1996). In such systems, , an extensible set of nodes interconnected by a network of networks (the Internet). They incorporate large numbers of nodes and provide distributed system services for global organizations. The level of heterogeneity in such systems is significant in terms of networks, computer architecture, operating systems, languages employed and the development teams involved. This has led to an increasing emphasis on open standards

2.3.3 Contemporary distributed systems

In the above systems, nodes were typically desktop computers and therefore relatively static, discrete (not embedded within other physical entities) and autonomous.

- The emergence of mobile computing has led to physical models where nodes such as laptops or smart phones may move from location to location in a distributed system
- The emergence of ubiquitous computing has led to a move from discrete nodes to architectures where computers are embedded in everyday objects and in the surrounding environment
- The emergence of cloud computing and, in particular, cluster architectures has led to a move from autonomous nodes performing a given role to pools of nodes that together provide a given service.

The end result is a physical architecture with a significant increase in the level of heterogeneity embracing, openess and quality of service. Such systems potentially involve up to hundreds of thousands of nodes.

Distributed systems of systems : The emergence of ultra- large-scale (ULS) distributed systems. A system of systems can be defined as a complex system consisting of a series of subsystems that are systems in their own right and that come together to perform a particular task.

Distributed systems:	*Early*	*Internet-scale*	*Contemporary*
Scale	Small	Large	Ultra-large
Heterogeneity	Limited (typically relatively homogenous configurations)	Significant in terms of platforms, languages and middleware	Added dimensions introduced including radically different styles of architecture
Openness	Not a priority	Significant priority with range of standards introduced	Major research challenge with existing standards not yet able to embrace complex systems
Quality of service	In its infancy	Significant priority with range of services introduced	Major research challenge with existing services not yet able to embrace complex systems