CHAPTER 2

SYSTEM MODELS

Types of Models

- **Physical models** - consider the types of computers and devices that constitute a system and their interconnectivity, without details of specific technology.

- **Architectural models** describe a system in terms of the computational and communication tasks performed by its computational elements; the computational elements being individual computers or aggregates of them supported by appropriate network interconnections. Client-server and peer-to-peer are two of the most commonly used forms of architectural model for distributed systems.

- **Fundamental models** take an abstract perspective in order to describe solutions to individual issues faced by most distributed systems.

Difficulties and threats for distributed systems

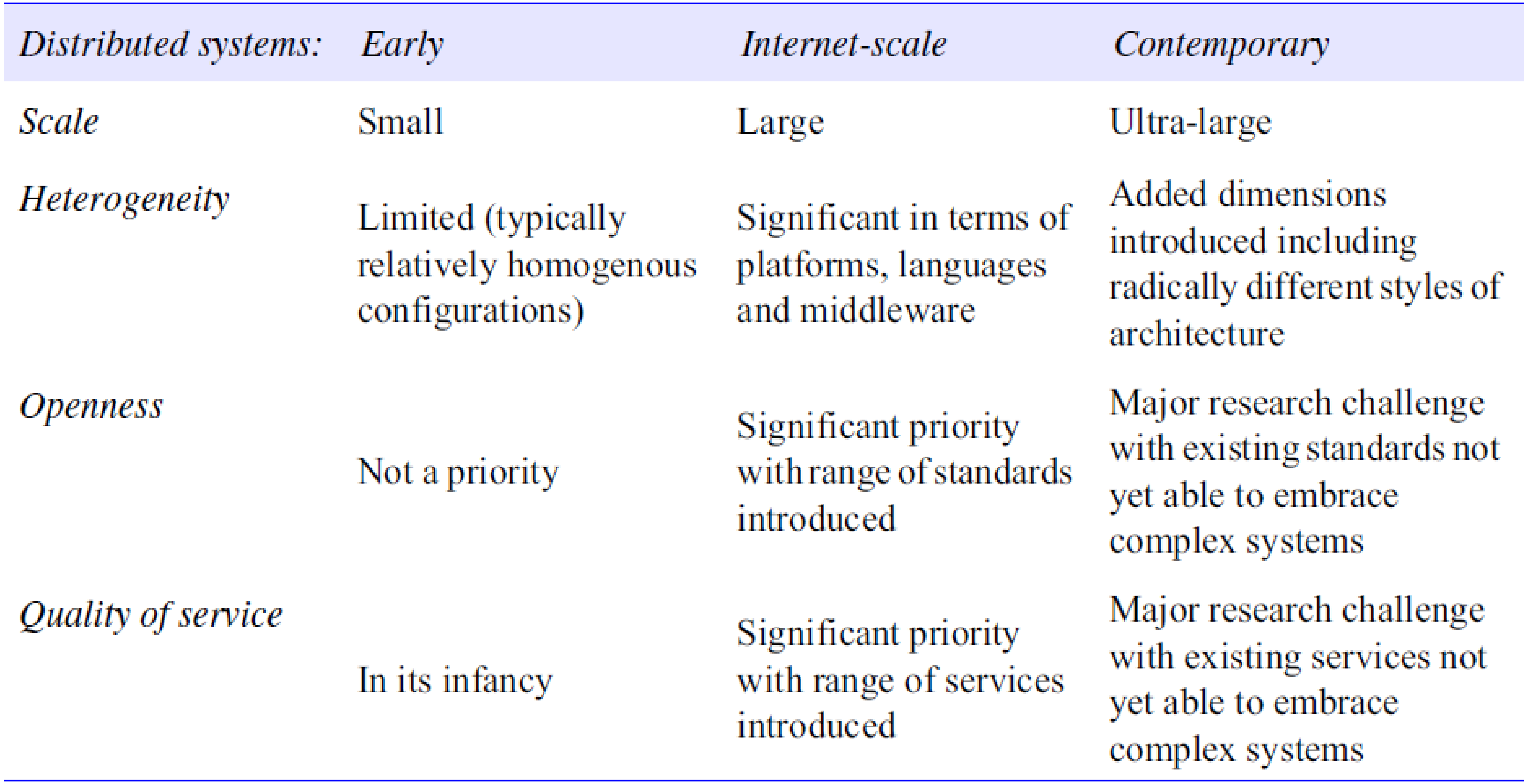
- Widely varying modes of use: The component parts of systems are subject to wide variations in workload. Some applications have special requirements for high communication bandwidth and low latency

- Wide range of system environments: A distributed system must accommodate heterogeneous hardware, operating systems and networks. The networks may differ widely in performance.

- Internal problems: Non-synchronized clocks, conflicting data updates and many modes of hardware and software failure involving the individual system components.

- External threats: Attacks on data integrity and secrecy, denial of service attacks.

Physical models



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

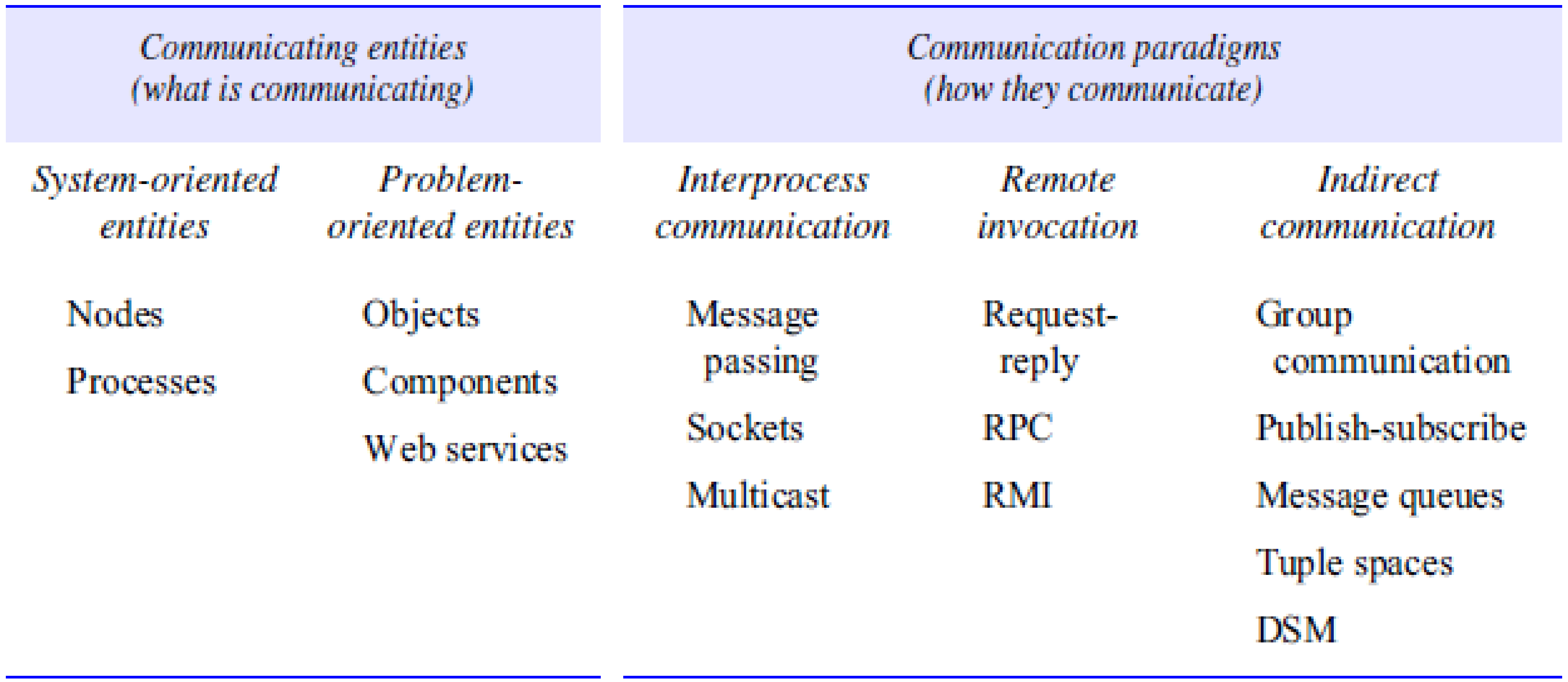
Distributed systems of systems

- Modern distributed systems are also being referred as ***systems of systems*** (mirroring the view of the Internet as a network of networks). 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 or tasks.

- As an example of a system of systems, consider an environmental management system for flood prediction. In such a scenario, there will be sensor networks deployed to monitor the state of various environmental parameters relating to rivers, flood plains, tidal effects and so on.

Architectural models

Figure 2.2 Communicating entities and communication paradigms



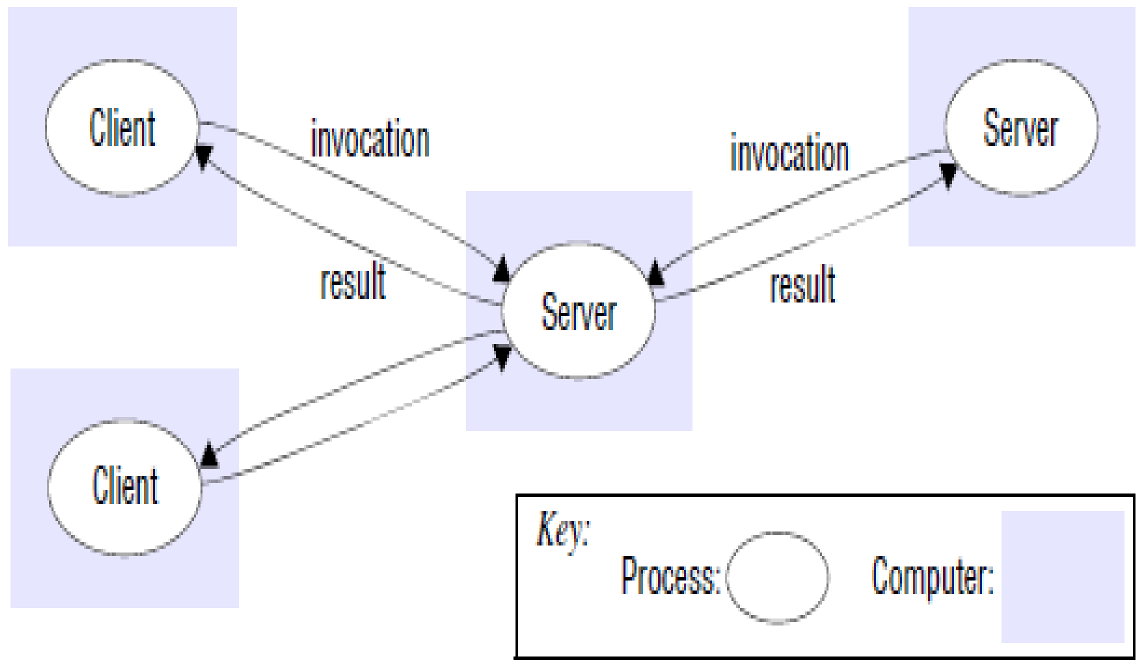
Communicating entities (what is communicating)		*Communication paradigms (how they communicate)*		
System-oriented entities	*Problem-oriented entities*	*Interprocess communication*	*Remote invocation*	*Indirect communication*
Nodes	Objects	Message passing	Request-reply	Group communication
Processes	Components	Sockets	RPC	Publish-subscribe
	Web services	Multicast	RMI	Message queues
				Tuple spaces
				DSM

Architectural Styles

- Client-server:

- Client processes interact with individual server processes in potentially separate host computers in order to access the shared resources that they manage.
- Servers may in turn be clients of other servers, as the figure indicates.
- For example, a web server is often a client of a local file server that manages the files in which the web pages are stored.
- Web servers and most other Internet services are clients of the DNS service, which translates Internet domain names to network addresses.

Clients invoke individual servers



Search Engines

- Is another web-related example which enable users to look up and 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.

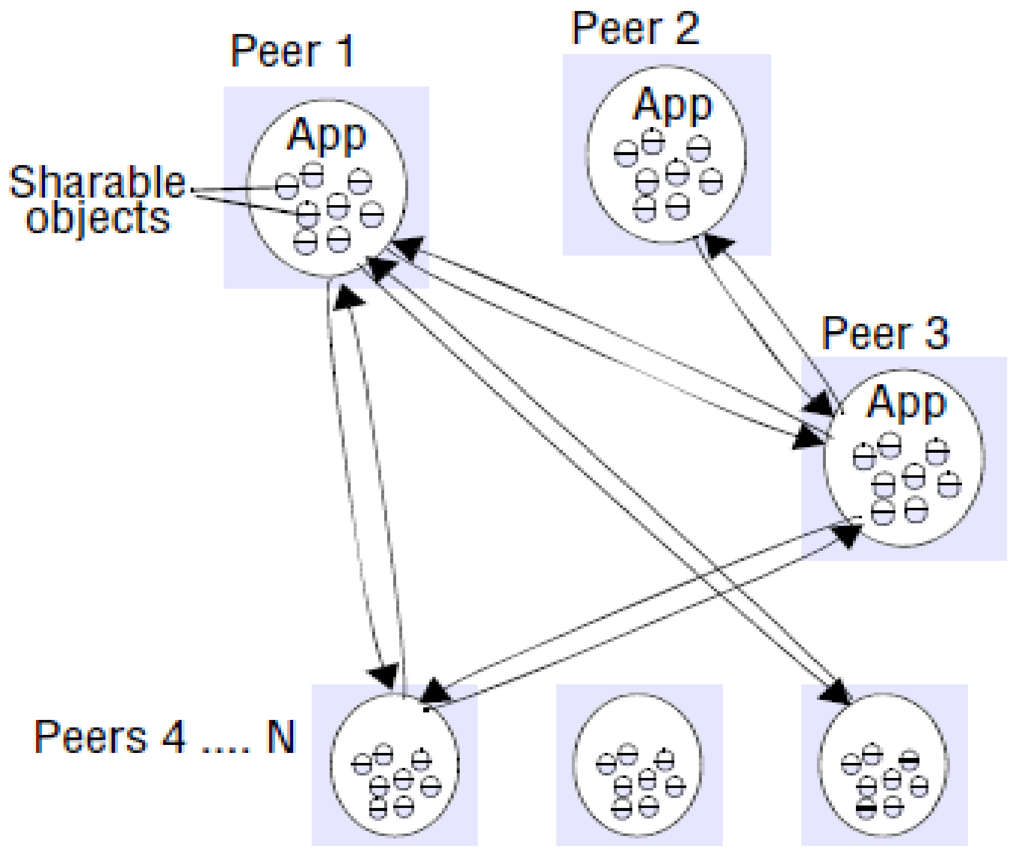
- Thus a search engine is both a server and a client: it responds to queries from browser clients and it runs web crawlers that act as clients of other web servers.

- In this example, the server tasks (responding to user queries) and the crawler tasks (making requests to other web servers) are entirely independent; there is little need to synchronize them and they may run concurrently.

- In fact, a typical search engine would normally include many concurrent threads of execution, some serving its clients and others running web crawlers.

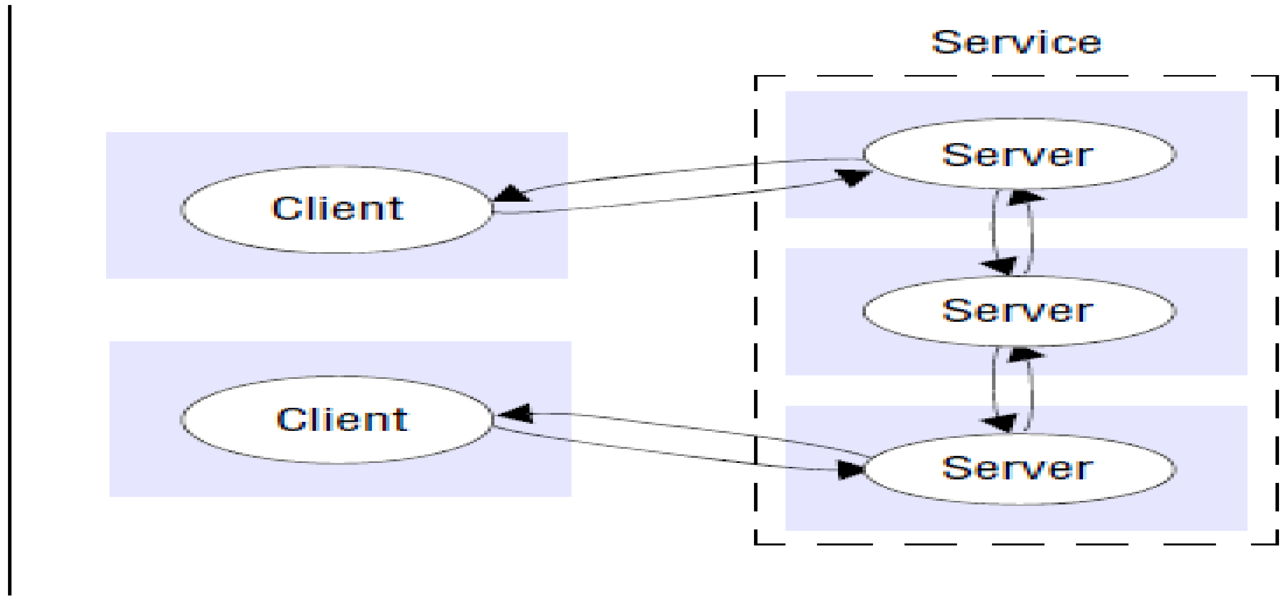
Peer-to-peer:

Figure 2.4a Peer-to-peer architecture



- ✓ The key insight that led to the development of peer-to-peer systems is that the network and computing resources owned by the users of a service could also be put to use to support that service.
- ✓ This has the useful consequence that the resources available to run the service grow with the number of users.
- ✓ The aim of the peer-to-peer architecture is to exploit the resources (both data and hardware) in a large number of participating computers for the fulfilment of a given task or activity.
- ✓ Peer-to-peer applications and systems have been successfully constructed that enable tens or hundreds of thousands of computers to provide access to data and other resources that they collectively store and manage. Examples – Naster and BitTorrent.

Figure 2.4b A service provided by multiple servers



- ✓ Each object is replicated in several computers to further distribute the load and to provide resilience in the event of disconnection of individual computers (as is inevitable in the large, heterogeneous networks at which peer-to-peer systems are aimed).
- ✓ The need to place individual objects and retrieve them and to maintain replicas amongst many computers renders this architecture substantially more complex than the client-server architecture.

Placement •

- ✓ The final issue to be considered is how entities such as objects or services map on to the underlying physical distributed infrastructure which will consist of a potentially large number of machines interconnected by a network of arbitrary complexity.
- ✓ Placement is crucial in terms of determining the properties of the distributed system, most obviously with regard to performance but also to other aspects, such as reliability and security.

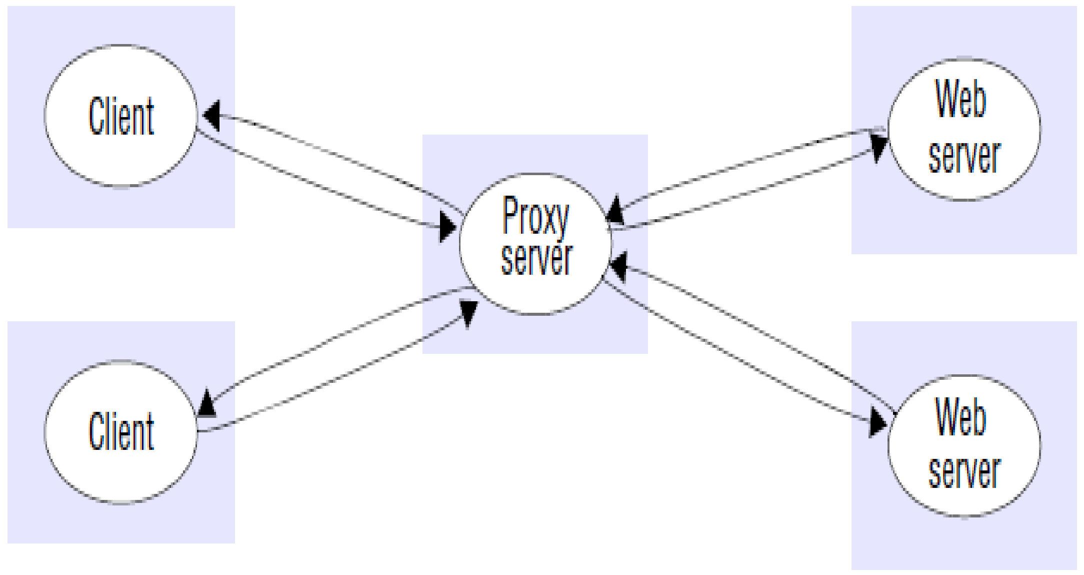
mapping of services to multiple servers;

- caching;
- mobile code;
- mobile agents

Caching:

- A *cache* is a store of recently used data objects that is closer to one client or a particular set of clients than the objects themselves.
- When a new object is received from a server it is added to the local cache store, replacing some existing objects if necessary.
- When an object is needed by a client process, the caching service first checks the cache and supplies the object from there if an up-to-date copy is available.
- If not, an up-to-date copy is fetched. Caches may be co-located with each client or they may be located in a proxy server that can be shared by several clients.
- 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.

Figure 2.5 Web proxy server

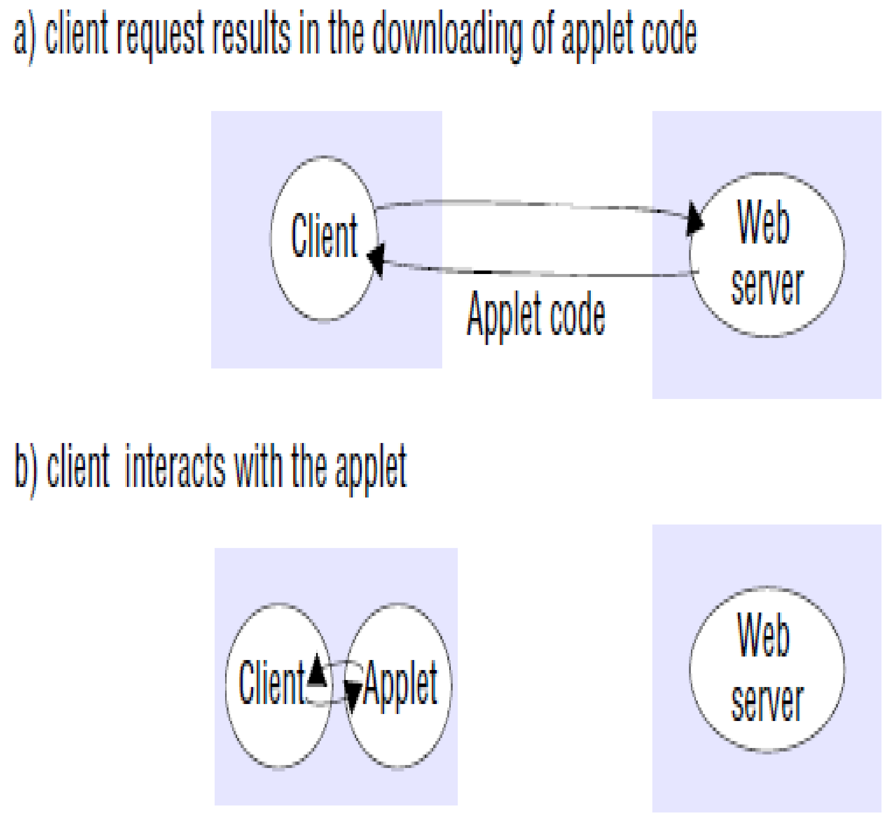


- ✓ Web proxy servers (Figure 2.5) provide a shared cache of web resources for the client machines at a site or across several sites.
- ✓ The purpose of proxy servers is to increase the availability and performance of the service by reducing the load on the wide area network and web servers. Proxy servers can take on other roles; for example, they may be used to access remote web servers through a firewall.

Mobile code

- Applets are a well-known and widely used example of mobile code – the user running a browser selects a link to an applet whose code is stored on a web server; the code is downloaded to the browser and runs there, as shown in Figure 2.6.
- An advantage of running the downloaded code locally is that it can give good interactive response since it does not suffer from the delays or variability of bandwidth associated with network communication

igure 2.6 Web applets


a) client request results in the downloading of applet code
Client
Web server
Applet code
b) client interacts with the applet
Client
Applet
Web server

Mobile agents:

- A mobile agent is a running program (including both code and data) that travels from one computer to another in a network carrying out a task on someone’s behalf, such as collecting information, and eventually returning with the results.
- Mobile agents might be used to install and maintain software on the computers within an organization or to compare the prices of products from a number of vendors by visiting each vendor’s site and performing a series of database operations.
- Mobile agents (like mobile code) are a potential security threat to the resources in computers that they visit. The environment receiving a mobile agent should decide which of the local resources it should be allowed to use, based on the identity of the user on whose behalf the agent is acting – their identity must be included in a secure way with the code and data of the mobile agent.
- mobile agents can themselves be vulnerable – they may not be able to complete their task if they are refused access to the information they need

Architectural patterns

- **Layering –**

  In terms of distributed systems, this equates to a vertical organization of services into service layers. A distributed service can be provided by one or more server processes, interacting with each other and with client processes in order to maintain a consistent system-wide view of the service's resources.

- **Tiered architecture-**

  Tiered architectures are complementary to layering. Whereas layering deals with the vertical organization of services into layers of abstraction, 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

Figure 2.7 Software and hardware service layers in distributed systems

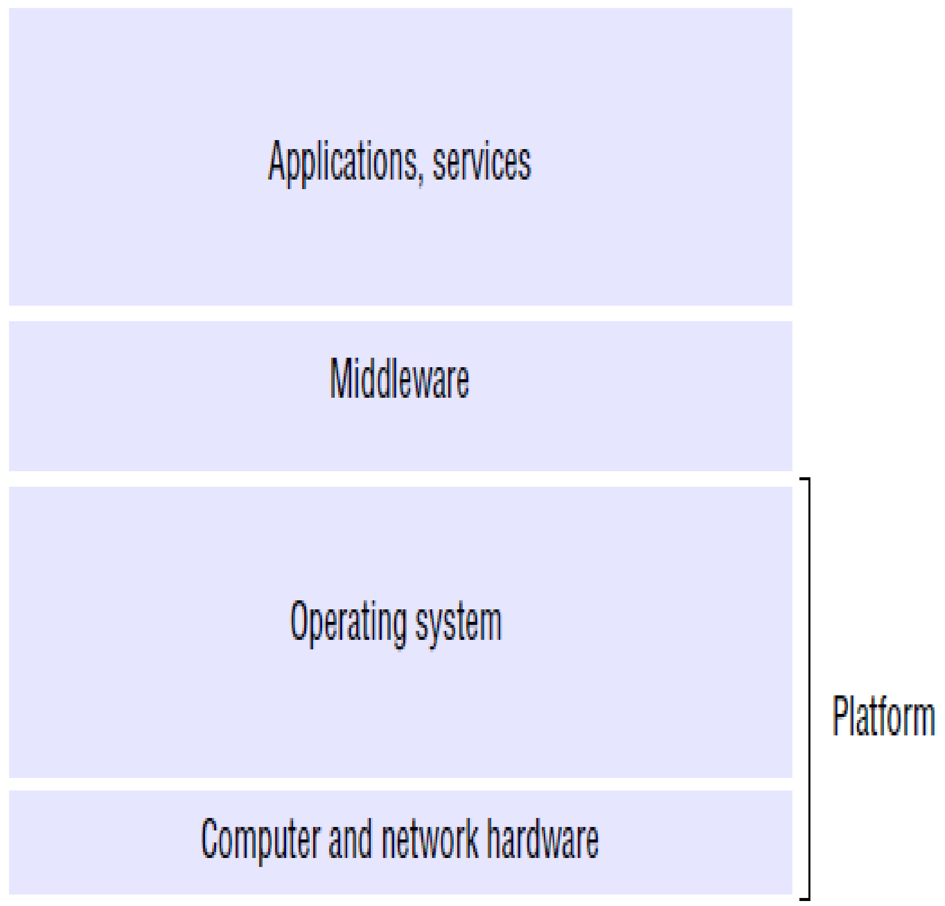


Figure 2.7 introduces the important terms *platform* and *middleware as seen in Layering .*

platform for distributed systems and applications consists of the lowest-level hardware and software layers.

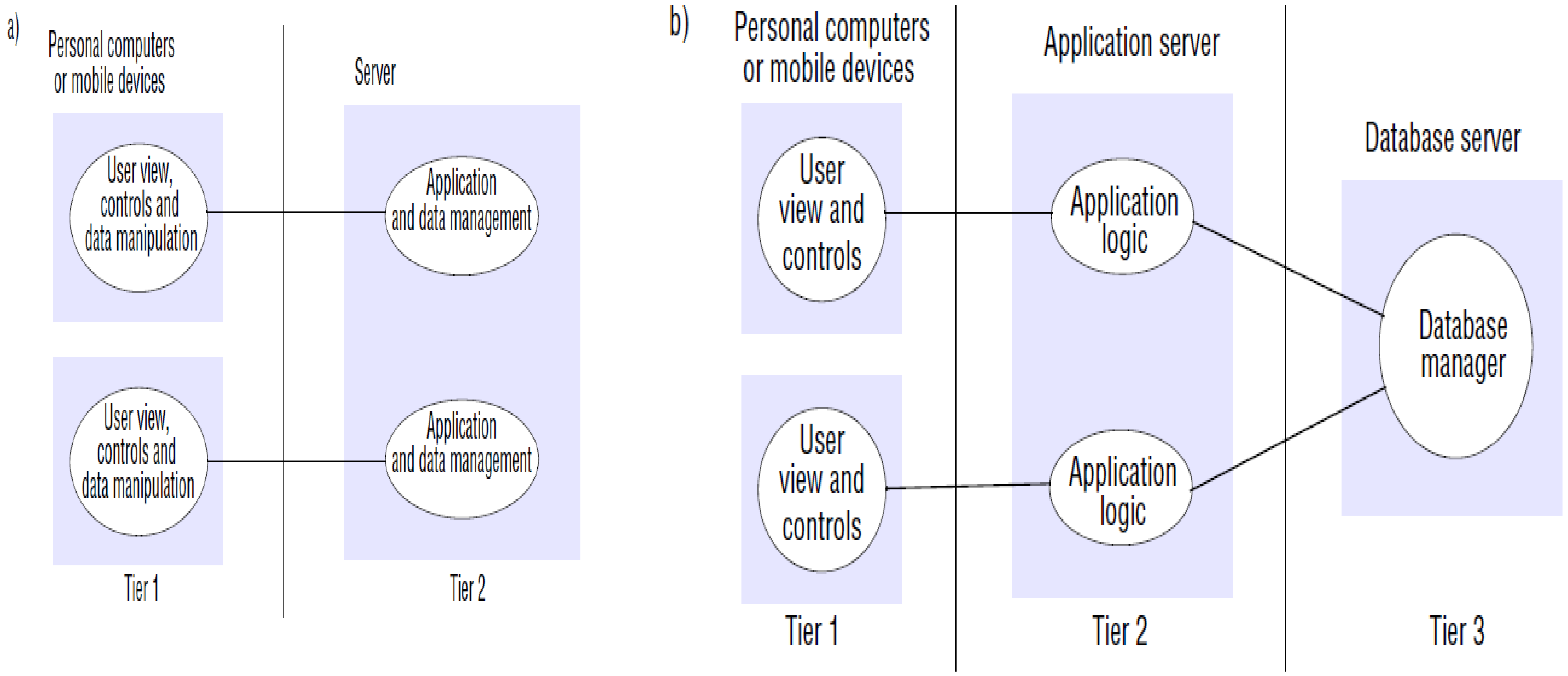
These low-level layers provide services to the layers above them, which are implemented independently in each computer, bringing the system's programming interface up to a level that facilitates communication and coordination between processes.

Middleware is represented by processes or objects in a set of computers that interact with each other to implement communication and resource-sharing support for distributed applications.
It is concerned with providing useful building blocks for the construction of software components that can work with one another in a distributed system.

Figure 2.12 Categories of middleware

Major categories:	*Subcategory*	*Example systems*
Distributed objects (Chapters 5, 8)	Standard	RM-ODP
	Platform	CORBA
	Platform	Java RMI
Distributed components (Chapter 8)	Lightweight components	Fractal
	Lightweight components	OpenCOM
	Application servers	SUN EJB
	Application servers	CORBA Component Model
	Application servers	JBoss
Publish-subscribe systems (Chapter 6)	-	CORBA Event Service
	-	Scribe
	-	JMS
Message queues (Chapter 6)	-	Websphere MQ
	-	JMS
Web services (Chapter 9)	Web services	Apache Axis
	Grid services	The Globus Toolkit
Peer-to-peer (Chapter 10)	Routing overlays	Pastry
	Routing overlays	Tapestry
	Application-specific	Squirrel
	Application-specific	OceanStore
	Application-specific	Ivy
	Application-specific	Gnutella

Figure 2.8 Two-tier and three-tier architectures


a)
Personal computers or mobile devices
Server
User view, controls and data manipulation
Application and data management
User view, controls and data manipulation
Application and data management
Tier 1
Tier 2
b)
Personal computers or mobile devices
Application server
Database server
User view and controls
Application logic
Database manager
User view and controls
Application logic
Tier 1
Tier 2
Tier 3

- The introduction of **Javascript,** a cross-platform and cross-browser programming language that is downloaded and executed in the browser, constituted a first step towards the removal of those constraints.

- Javascript is a general-purpose language enabling both user interface and application logic to be programmed and executed in the context of a browser window.

- **AJAX** is the second innovative step that was needed to enable major interactive web applications to be developed and deployed. It enables Javascript front-end programs to request new data directly from server programs. Any data items can be requested and the current page updated selectively to show the new values. Indeed, the front end can react to the new data in any way that is useful for the application.

- AJAX is the ‘glue’ that supports the construction of such applications; it provides a communication mechanism enabling front-end components running in a browser to issue requests and receive results from back-end components running on a server.

- Clients issue requests through the Javascript *XmlHttpRequest* object, which manages an HTTP exchange with a server process. Because *XmlHttpRequest* has a complex API that is also somewhat browser-dependent, it is usually accessed through one of the many Javascript libraries that are available to support the development of web applications.

- The AJAX mechanism constitutes an effective technique for the construction of responsive web applications in the context of the indeterminate latency of the Internet, and it has been very widely deployed. The Google Maps application is an outstanding example.

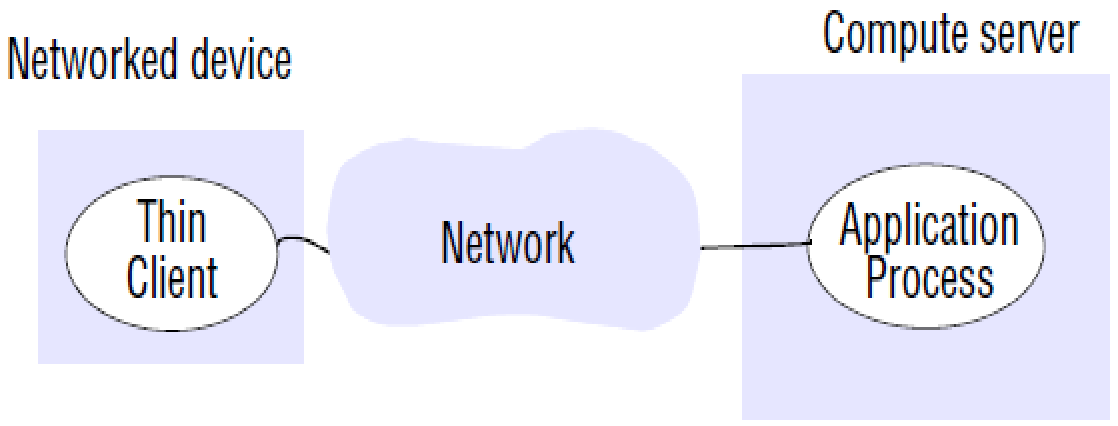
- Maps are displayed as an array of contiguous 256 x 256 pixel images (called *tiles*). When the map is moved the visible tiles are repositioned by Javascript code in the browser and additional tiles needed to fill the visible area are requested with an AJAX call to a Google server. They are displayed as soon as they are received, but the browser continues to respond to user interaction while they are awaited.

Thin clients

- The term thin client refers to a software layer that supports a window-based user interface that is local to the user while executing application programs or, more generally, accessing services on a remote computer.

- The main drawback of the thin client architecture is in highly interactive graphical activities such as CAD and image processing, where the delays experienced by users are increased to unacceptable levels by the need to transfer image and vector information between the thin client and the application process, due to both network and operating system latencies.

Figure 2.10 Thin clients and computer servers



Virtual network computing

- The concept is straightforward, providing remote access to graphical user interfaces.
- In this solution, a VNC client (or viewer) interacts with a VNC server through a VNC protocol. The protocol operates at a primitive level in terms of graphics support, based on framebuffers and featuring one operation: the placement of a rectangle of pixel data at a given position on the screen (some solutions, such as XenApp from Citrix operate at a higher level in terms of window operations.
- Virtual network computing has superseded network computers, a previous attempt to realise thin client solutions through simple and inexpensive hardware devices that are completely reliant on networked services, downloading their operating system and any application software needed by the user from a remote file server.
- Since all the application data and code is stored by a file server, the users may migrate from one network computer to another.

Fundamental Models

Fundamental Models- Basics

- A fundamental model should contain only the essential ingredients that we need to consider in order to understand and reason about some aspects of a system's behaviour.
- Purpose of the model:
  - To make explicit all the relevant assumptions about the system.
  - To make generalizations concerning what is possible or impossible, given those assumptions. The generalizations may take the form of general-purpose algorithms or desirable properties that are guaranteed. The guarantees are dependent on logical analysis and, where appropriate, mathematical proof.
- There is much to be gained by knowing what our designs do, and do not, depend upon. It allows us to decide whether a design will work if we try to implement it in a particular system: we need to only ask whether our assumptions hold in that system.

Fundamental Models (cont.)

- The aspects of distributed systems that we wish to capture in our fundamental models are intended to help us to discuss and reason about:
  - **Interaction:** Computation occurs within processes; the processes interact by passing messages, resulting in communication (information flow) and coordination (synchronization and ordering of activities) between processes. In the analysis and design of distributed systems we are concerned especially with these interactions.
  - The interaction model must reflect the facts that communication takes place with delays that are often of considerable duration, and that the accuracy with which independent processes can be coordinated is limited by these delays and by the difficulty of maintaining the same notion of time across all the computers in a distributed system.

Fundamental Models (cont.)

- **Failure:** The correct operation of a distributed system is threatened whenever a fault occurs in any of the computers on which it runs (including software faults) or in the network that connects them. Our model defines and classifies the faults. This provides a basis for the analysis of their potential effects and for the design of systems that are able to tolerate faults of each type while continuing to run correctly.

- **Security:** The modular nature of distributed systems and their openness exposes them to attack by both external and internal agents. Our security model defines and classifies the forms that such attacks may take, providing a basis for the analysis of threats to a system and for the design of systems that are able to resist them.

Fundamental Models (cont.)

1. Interaction Model:
   - Deals with communication details among the components and their timing and performance details.
2. Failure Model:
   - Provides specification and effects of faults relating to processes and communication channels.
3. Security Model:
   - Specifies possibilities of attacks to processes and communication channels due to openness and modular nature of distributed systems.
   - Provides a basis for the analysis of threats to a system.

Basics of (1) Interaction Model

- Multiple server processes may cooperate to provide service. For example, DNS, which partitions and replicates its data at servers throughout the Internet, and Sun’s Network Information Service, which keeps replicated copies of password files at several servers in a local area network.

- A set of peer processes may cooperate to achieve common goal. For example, Audio conferencing, a voice conferencing system that distributes streams of audio data in a similar manner, but with strict real-time constraints.

- Their behaviour and state can be defined by distributed algorithm.

Basics of (1) Interaction Model (cont.)

- **Distributed Algorithm**
  - Definition of the steps to be taken by each of the processes of which DS is made of, including the transmission of messages.
  - Rate at which each process proceed and the timing of transmission of messages can not be predicted in general.
  - Each process has its own state.
- The rate at which each process proceeds and the timing of the transmission of messages between them cannot in general be predicted. It is also difficult to describe all the states of a distributed algorithm, because it must deal with the failures of one or more of the processes involved or the failure of message transmissions.
- Significant factors affecting interacting processes in a distributed system:
  - Communication performance is often a limiting characteristic.
  - It is impossible to maintain a single global notion of time.

Performance of communication channels:

- The communication channels in our model are realized in a variety of ways in distributed systems – for example, by an implementation of streams or by simple message passing over a computer network.

- **Delay** between start of a message's transmission from one process and beginning of its receipt by another process. It includes:

  - Time taken for the first of a string of bits transmitted through a network to reach its destination.

  - Delay in accessing the network increases with increase in network load. For example, for Ethernet transmission the sending station waits for the network to be free of traffic.

  - Time taken by operating system communication services at both sending and receiving processes, which varies according to current load on operating system.

Performance of communication channels (cont.)

- The **bandwidth** of a computer network is the total amount of information that can be transmitted over it in a given time. When a large number of communication channels are using the same network, they have to share the available bandwidth.

- **Jitter** is the variation in the time taken to deliver a series of messages. Jitter is relevant to multimedia data. For example, if consecutive samples of audio data are played with differing time intervals, the sound will be badly distorted.

Computer clocks and timing events:

- Each computer in DS has its own internal clock, which can be used by local processes.
- Processes running on different computers associate timestamps with their events.
- If two processes read their clocks at the same time, their local clocks may supply different time values.
- **Clock drift rate:** The relative amount of time with which a computer clock deviates from a perfect reference clock. Even if the clocks on all the computers in a distributed system are set to the same time initially, their clocks will eventually vary quite significantly unless corrections are applied.
- For example, computers may use radio receivers to get time readings from the Global Positioning System with an accuracy of about 1 micro-second.

Variants of Interaction Model- Synchronous vs. Asynchronous

- In a DS, it is hard to set limits on the time that can be taken for process execution, message delivery or clock drift. Two opposing extreme positions provide a pair of simple models. The first has a strong assumption of time and the second makes no assumptions about time:
- **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.

Synchronous distributed systems (cont.):

- It is possible to suggest likely upper and lower bounds for process execution time, message delay and clock drift rates in a distributed system, but it is difficult to arrive at realistic values and to provide guarantees of the chosen values.
- Unless the values of the bounds can be guaranteed, any design based on the chosen values will not be reliable.
- However, modelling an algorithm as a synchronous system may be useful for giving some idea of how it will behave in a real distributed system. In a synchronous system it is possible to use timeouts, for example, to detect the failure of a process.

Synchronous distributed systems (cont.):

- Synchronous distributed systems can be built. What is required is for the processes to perform tasks with known resource requirements for which they can be guaranteed sufficient processor cycles and network capacity, and for processes to be supplied with clocks with bounded drift rates.

Asynchronous distributed systems:

- Many distributed systems, such as the Internet, are very useful without being able to qualify as synchronous systems. Therefore we need an alternative model. An asynchronous distributed system is one in which there are no bounds on:
  - Process execution speeds- for example, one process step may take only a picosecond and another a century; all that can be said is that each step may take an arbitrarily long time.
  - Message transmission delays- for example, one message from process A to process B may be delivered in negligible time and another may take several years. In other words, a message may be received after an arbitrarily long time.
  - Clock drift rates- the drift rate of a clock is arbitrary.

Asynchronous distributed systems (cont.)

- The asynchronous model allows no assumptions about the time intervals involved in any execution. This exactly models the Internet, in which there is no intrinsic bound on server or network load and therefore on how long it takes, for example, to transfer a file using FTP. Sometimes an email message can take days to arrive.
- The Web cannot always provide a particular response within a reasonable time limit, browsers have been designed to allow users to do other things while they are waiting. Any solution that is valid for an asynchronous distributed system is also valid for a synchronous one.

Asynchronous distributed systems (cont.)

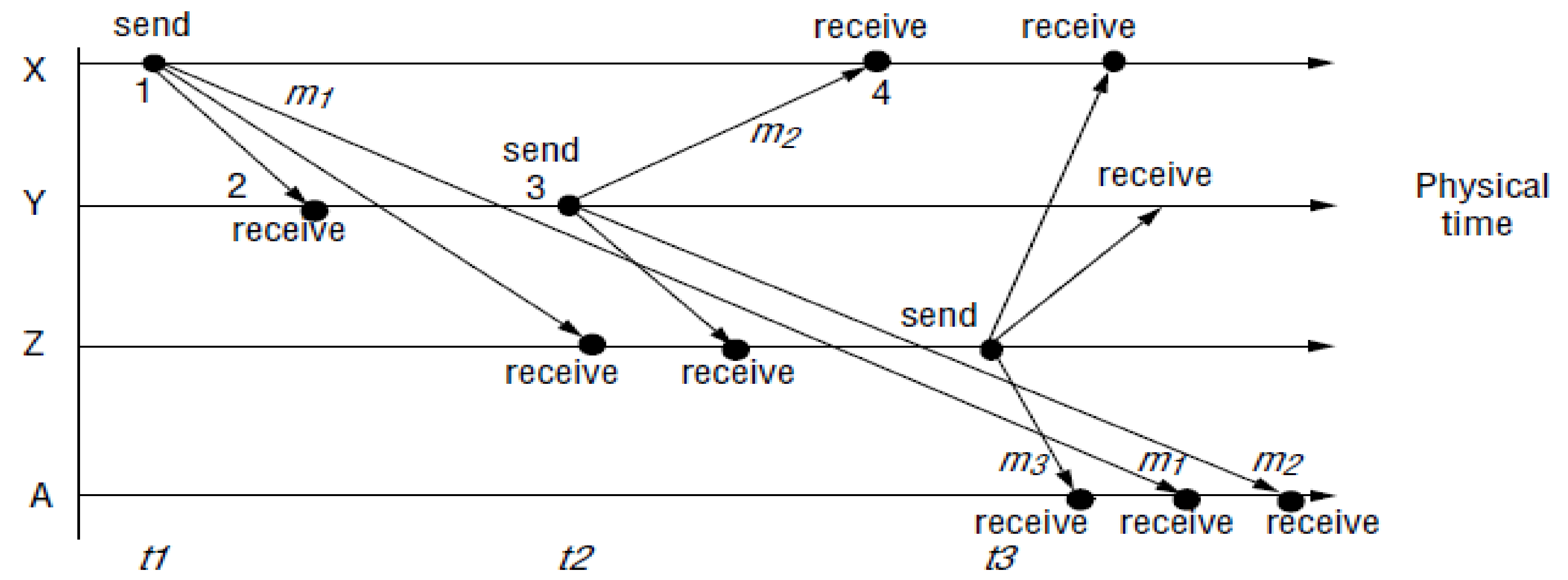
- Actual distributed systems are very often asynchronous because of the need for processes to share the processors and for communication channels to share the network.
- For example, if too many processes of unknown character are sharing a processor, then the resulting performance of any one of them cannot be guaranteed.
- But there are many design problems that cannot be solved for an asynchronous system that can be solved when some aspects of time are used. The need for each element of a multimedia data stream to be delivered before a deadline is such a problem. For problems such as these, a synchronous model is required.

Event Ordering

- In many cases, we are interested in knowing whether an event (sending or receiving a message) at one process occurred before, after or concurrently with another event at another process. The execution of a system can be described in terms of events and their ordering despite the lack of accurate clocks.
- For example, consider the following set of exchanges between a group of email users, X, Y, Z and A, on a mailing list:
  - User X sends a message with the subject Meeting.
  - Users Y and Z reply by sending a message with the subject Re: Meeting.

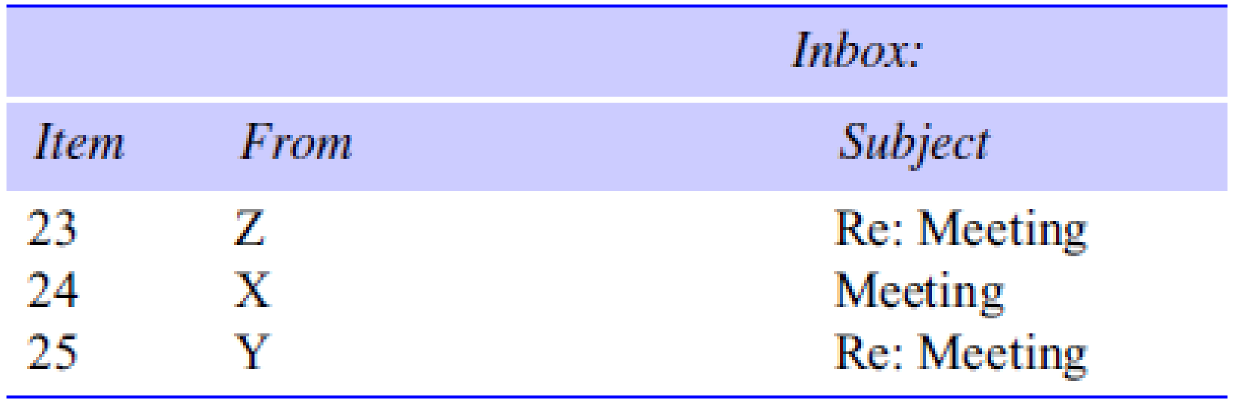
Event Ordering (cont.)

Real-time ordering of events



Event Ordering (cont.)

- 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. Some users may view these two messages in the wrong order. For example, user A might see:



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

Event Ordering (cont.)

- If the clocks on X’s, Y’s and Z’s computers could be synchronized, then each message could carry the time on the local computer’s clock when it was sent.
- For example, messages *m1*, *m2* and *m3* would carry times *t1*, *t2* and *t3* where $t1<t2<t3$. The messages received will be displayed to users according to their time ordering. If the clocks are roughly synchronized, then these timestamps will often be in the correct order.

Event Ordering (cont.)

- Since clocks cannot be synchronized perfectly across a distributed system, Lamport [1978] 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.

- Logical time allows the order in which the messages are presented to be inferred without recourse to clocks.

- Let us understand how some aspects of logical ordering can be applied to our email ordering problem.

Event Ordering (cont.)

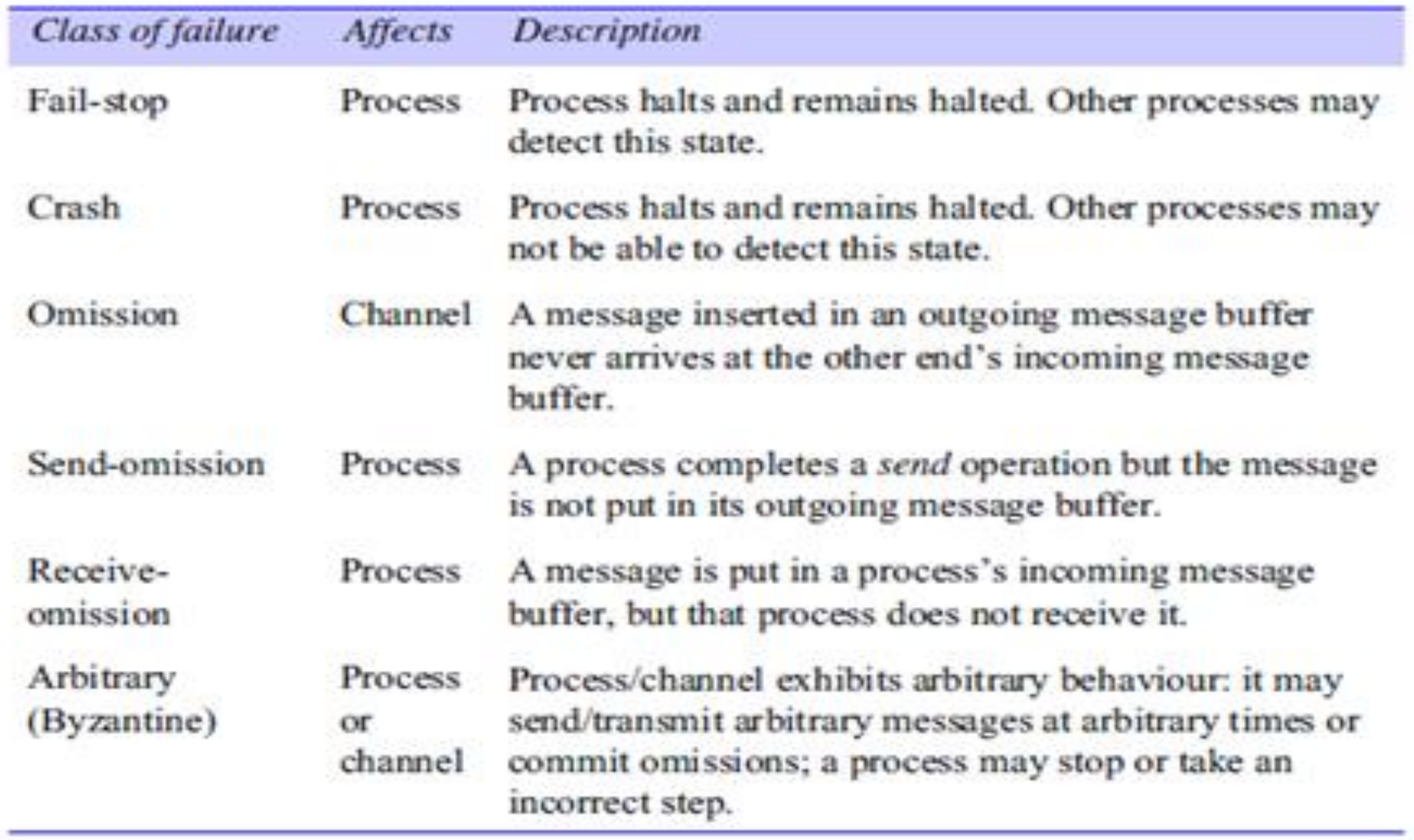
- Logically, we know that a message is received after it was sent. Therefore we can state a logical ordering for pairs of events shown in Figure, for example, considering only the events concerning X and Y:
  - X sends *m1* before Y receives *m1*; Y sends *m2* before X receives *m2*.

- We also know that replies are sent after receiving messages, so we have the following logical ordering for Y:
  - Y receives *m1* before sending *m2*.

- 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.

Failure model

Figure 2.15 Omission and arbitrary failures



Class of failure	*Affects*	*Description*
Fail-stop	Process	Process halts and remains halted. Other processes may detect this state.
Crash	Process	Process halts and remains halted. Other processes may not be able to detect this state.
Omission	Channel	A message inserted in an outgoing message buffer never arrives at the other end's incoming message buffer.
Send-omission	Process	A process completes a *send* operation but the message is not put in its outgoing message buffer.
Receive-omission	Process	A message is put in a process's incoming message buffer, but that process does not receive it.
Arbitrary (Byzantine)	Process or channel	Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times or commit omissions; a process may stop or take an incorrect step.

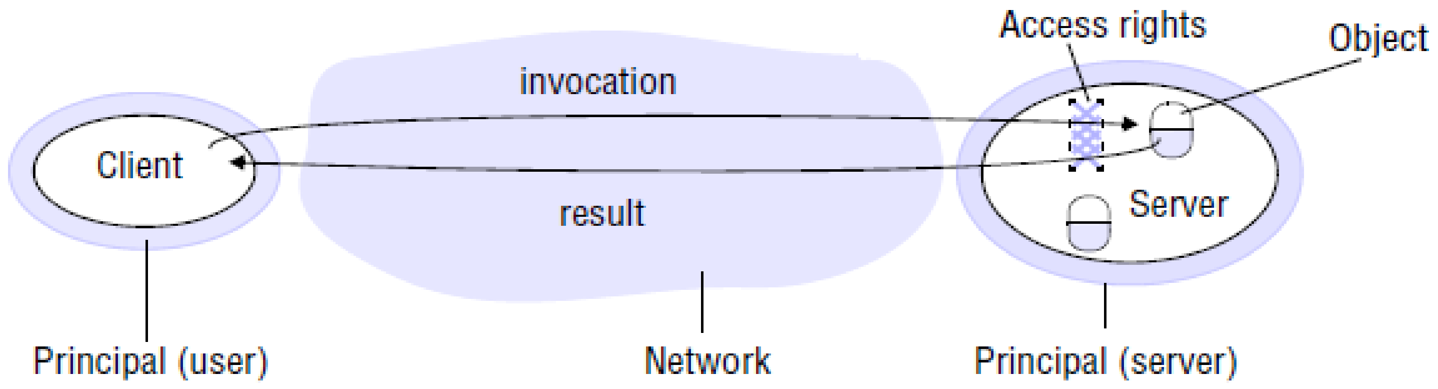
Figure 2.16 Timing failures

Class of failure	*Affects*	*Description*
Clock	Process	Process's local clock exceeds the bounds on its rate of drift from real time.
Performance	Process	Process exceeds the bounds on the interval between two steps.
Performance	Channel	A message's transmission takes longer than the stated bound.

Security Model

Basics of (3) Security Model

- The architectural model provides the basis for security model.
- The security of a distributed system can be achieved by:
  - Protecting the objects against unauthorized access
  - Securing the processes and the channels



Basics of (3) Security Model (cont.)

1. **Protecting objects**

- Figure shows a server that manages a collection of objects on behalf of some users. The users can run client programs that send invocations to the server to perform operations on the objects. The server carries out the operation specified in each invocation and sends the result to the client.
- Objects are intended to be used in different ways by different users. For example, some objects may hold a user’s private data, such as their mailbox, and other objects may hold shared data such as web pages.
- To support this, access rights specify who is allowed to perform the operations of an object – for example, who is allowed to read or to write its state.

Basics of (3) Security Model (cont.)

- We must include users in our model as the beneficiaries of access rights. We do so by associating with each invocation and each result the authority on which it is issued. Such an authority is called a *principal*. A principal may be a user or a process.
- In our illustration, the invocation comes from a user and the result from a server.
- The server is responsible for verifying the identity of the principal behind each invocation and checking that they have sufficient access rights to perform the requested operation on the particular object invoked, rejecting those that do not.
- The client may check the identity of the principal behind the server to ensure that the result comes from the required server.

Basics of (3) Security Model (cont.)

2. Securing processes and their interaction

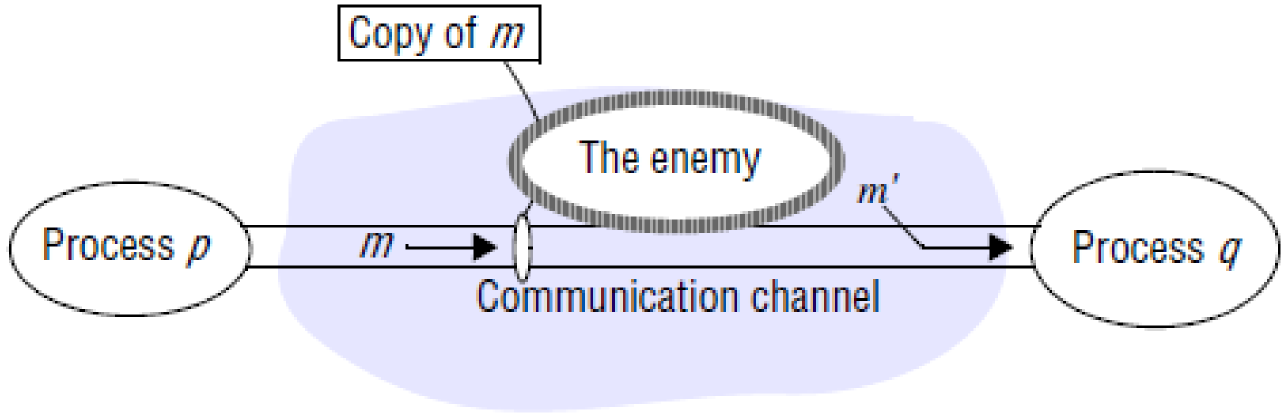
- Processes interact by sending messages.
- The messages are exposed to attack because the network and the communication service that they use are open, to enable any pair of processes to interact.
- Servers and peer processes expose their interfaces, enabling invocations to be sent to them by any other process.
- Distributed systems are often deployed and used in tasks that are likely to be subject to external attacks by hostile users. This is especially true for applications that handle financial transactions, confidential or classified information or any other information whose secrecy or integrity is crucial.

Basics of (3) Security Model (cont.)

- Integrity is threatened by security violations as well as communication failures. So we know that there are likely to be threats to the processes of which such applications are composed and to the messages travelling between the processes.
- But how can we analyze these threats in order to identify and defeat them? The further discussion introduces a model for the analysis of security threats.

Security Model- The Enemy

- To model security threats, we postulate an enemy (sometimes also known as the adversary) that is capable of sending any message to any process and reading or copying any message sent between a pair of processes, as shown in Figure.



Security Model- The Enemy (cont.)

- Such attacks can be made simply by using a computer connected to a network to run a program that reads network messages addressed to other computers on the network, or a program that generates messages that make false requests to services, purporting to come from authorized users.
- The attack may come from a computer that is legitimately connected to the network or from one that is connected in an unauthorized manner.
- 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.
- Communication protocols such as IP do include the address of the source computer in each message, but it is not difficult for an enemy to generate a message with a forged source address.
- This lack of reliable knowledge of the source of a message is a threat to the correct functioning of both servers and clients, as explained further:

Threats to Processes (cont.)

Servers:

- Since a server can receive invocations from many different clients, it cannot necessarily determine the identity of the principal behind any particular invocation.
- Even if a server requires the inclusion of the principal's identity in each invocation, an enemy might generate an invocation with a false identity. Without reliable knowledge of the sender's identity, a server cannot tell whether to perform the operation or to reject it.
- For example, a mail server would not know whether the user behind an invocation that requests a mail item from a particular mailbox is allowed to do so or whether it was a request from an enemy.

Threats to Processes (cont.)

Clients:

- When a client receives the result of an invocation from a server, it cannot necessarily tell whether the source of the result message is from the intended server or from an enemy, perhaps ‘spoofing’ the mail server.
- Thus, the client could receive a result that was unrelated to the original invocation, such as a false mail item (one that is not in the user’s mailbox).

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. For example, a result message containing a user’s mail item might be revealed to another user or it might be altered to say something quite different.
- Another form of attack is the attempt to save copies of messages and to replay them at a later time, making it possible to reuse the same message over and over again.
- For example, someone could benefit by resending an invocation message requesting a transfer of a sum of money from one bank account to another.
- All these threats can be defeated by the use of ***secure channels***, which are described below and are based on cryptography and authentication.

Defeating Security Threats

Here we will introduce the main techniques on which secure systems are based.

1. **Cryptography and shared secrets:**

- Suppose that a pair of processes (for example, a particular client and a particular server) share a secret; that is, they both know the secret but no other process in the distributed system knows it.
- Then if a message exchanged by that pair of processes includes information that proves the sender’s knowledge of the shared secret, the recipient knows for sure that the sender was the other process in the pair.
- Of course, care must be taken to ensure that the shared secret is not revealed to an enemy.

1. Cryptography and shared secrets (cont.)

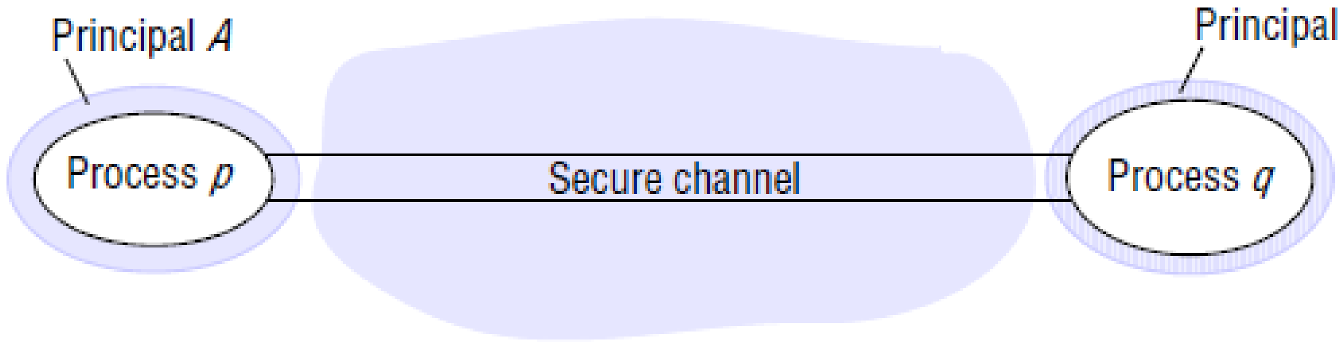
- *Cryptography* is the science of keeping messages secure, and *encryption* is the process of scrambling a message in such a way as to hide its contents.
- Modern cryptography is based on encryption algorithms that use secret keys – large numbers that are difficult to guess – to transform data in a manner that can only be reversed with knowledge of the corresponding decryption key.

2. Authentication

- The use of shared secrets and encryption provides the basis for the *authentication* of messages – proving the identities supplied by their senders.
- The basic authentication technique is to include in a message an encrypted portion that contains enough of the contents of the message to guarantee its authenticity.
- The authentication portion of a request to a file server to read part of a file, for example, might include a representation of the requesting principal’s identity, the identity of the file and the date and time of the request, all encrypted with a secret key shared between the file server and the requesting process.
- The server would decrypt this and check that it corresponds to the unencrypted details specified in the request.

3. Secure Channels

- Encryption and authentication are used to build secure channels as service layers on the top of the existing communication services.
- A secure channel is a communication channel connecting a pair of processes, each of which acts on behalf of a principal, as shown in Figure.



3. Secure Channels (cont.)

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, the server knows the identity of the principal behind the invocations and can check their access rights before performing an operation. This enables the server to protect its objects correctly and allows the client to be sure that it is receiving results from a *bona fide* server.
- 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.

4. Other Possible Threats

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 (network bandwidth, server processing capacity).
- Such attacks are usually made with the intention of delaying or preventing actions by other users.
- For example, the operation of electronic door locks in a building might be disabled by an attack that saturates the computer controlling the electronic locks with invalid requests.

4. Other Possible Threats (cont.)

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 but in fact including code that accesses or modifies resources that are legitimately available to the host process but not to the originator of the code.
- The methods by which such attacks might be carried out are many and varied, and the host environment must be very carefully constructed in order to avoid them. Many of these issues have been addressed in Java and other mobile code systems.

The Uses of Security Models

- It might be thought that the achievement of security in DS would be a straightforward matter involving the control of access to objects according to predefined access rights and the use of secure channels for communication. Unfortunately, this is not generally the case.
- The use of security techniques such as encryption and access control incurs substantial processing and management costs.
- The security model outlined above provides the basis for the analysis and design of secure systems in which these costs are kept to a minimum, but threats to a distributed system arise at many points, and a careful analysis of the threats that might arise from all possible sources in the system's network environment, physical environment and human environment is needed. This analysis involves the construction of a *threat model* listing all the forms of attack to which the system is exposed and an evaluation of the risks and consequences of each.
- The effectiveness and the cost of the security techniques needed can then be balanced against the threats.