IS473 Distributed Systems

CHAPTER 2

System Models

OUTLINE

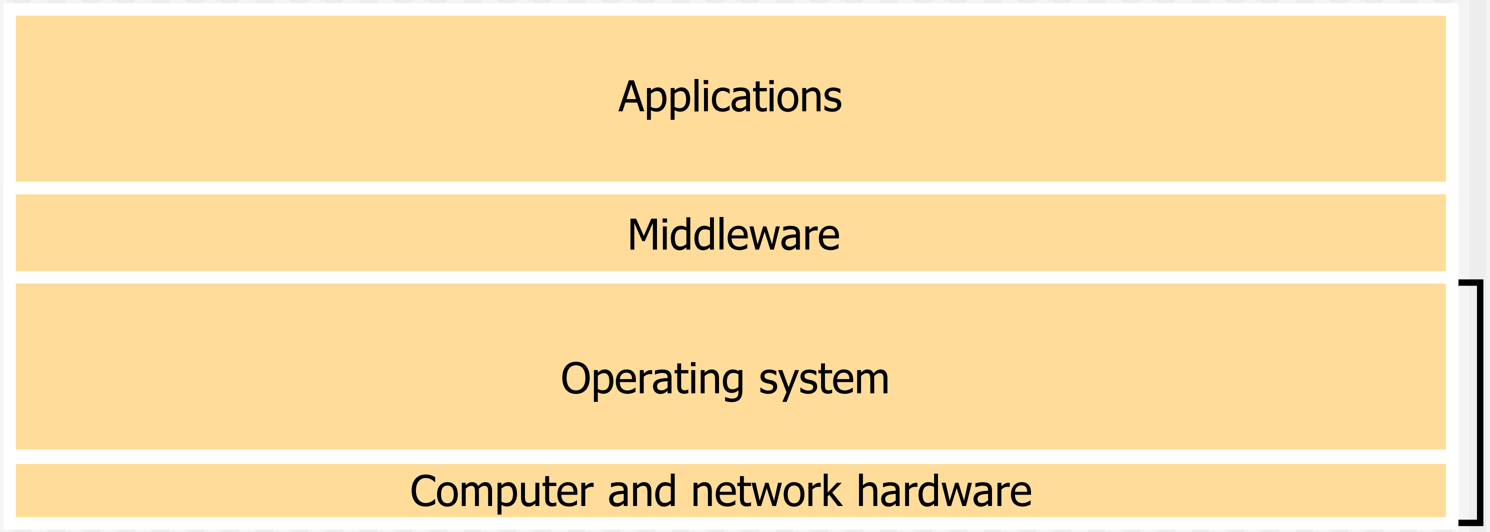
- Distributed System Models.
- Distributed System Software Layers
- Distributed System Architectures.
- Client-Server Model Variations.
- Distributed Processes Interfaces and Objects.
- Distributed Architectures Design Requirements.
- Fundamental Models:
  - Interaction Model.
  - Failure Model.
  - Security Model.

Distributed System Models

- Architectural models: (*as client-server and peer process models*)
  - Define the way in which the components of systems are:
    - Interact with one another, and
    - Mapped onto an underlying network of computers.
  - Describe the layered structure of distributed system software.
- Fundamental models:
  - Concerned with properties that are common in all of the architectural models.
  - Addressed by three models:
    - The interaction model: deals with the difficulty of setting time limits.
    - The failure model: attempts to give a specification of the exhibited faults by processes and communication channels.
    - The security model: discusses possible threats to processes and communication channels.

Software Layers

- In the layered view of a system each layer offers its services to the level above and builds its own service on the services of the layer below.
- Software architecture is the structuring of software in terms of layers (modules) or services that can be requested locally or remotely.



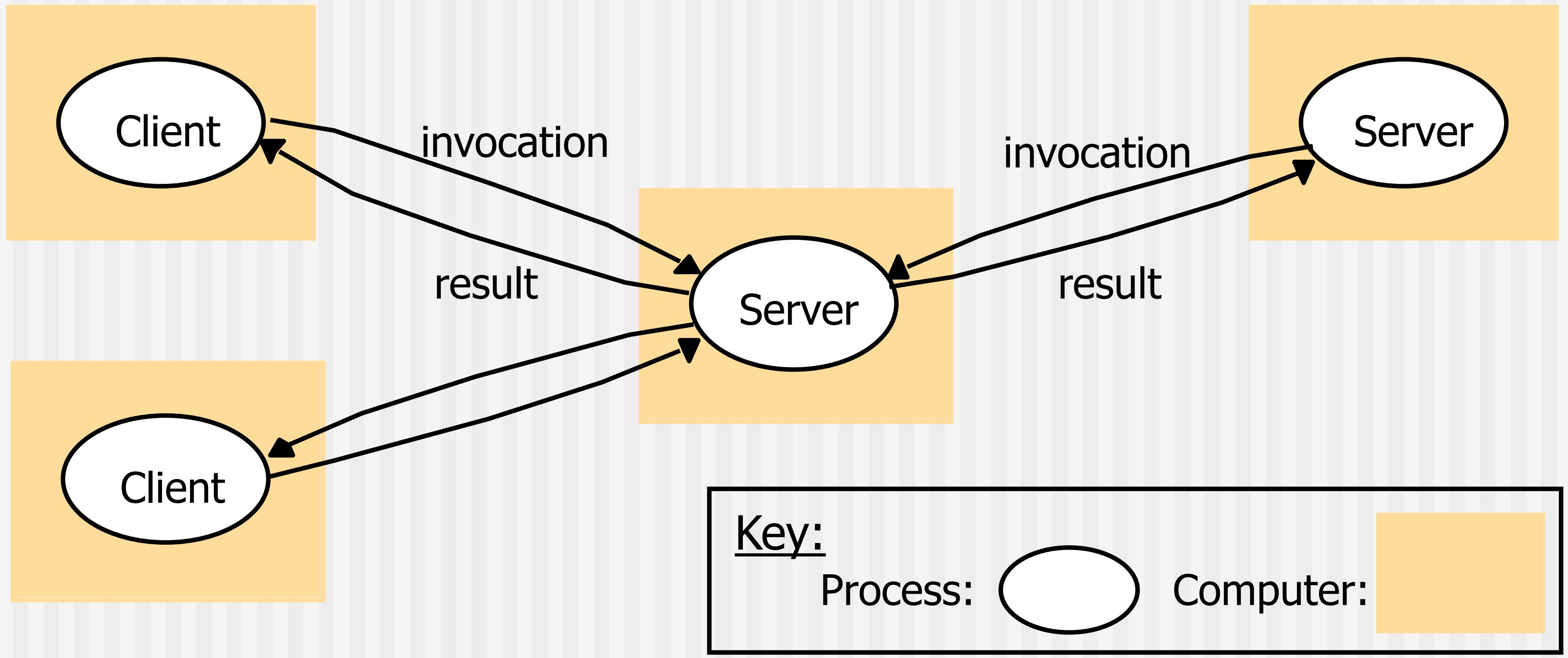
Software Layers

- Platform:
  - Lowest-level layers that provide services to other higher layers.
  - bring a system's programming interface for communication and coordination between processes .
  - Examples:
    - Pentium processor / Windows NT
    - SPARC processor / Solaris
- Middleware:
  - Layer of software to mask heterogeneity and provide a unified distributed programming interface to application programmers.
  - Provide services, *infrastructure services*, for use by application programs.
  - Examples:
    - Object Management Group's Common Object Request Broker Architecture (CORBA).
    - Java Remote Object Invocation (RMI).
    - Microsoft's Distributed Common Object Model (DCOM).
  - Limitation: require application level involvement in some tasks.

System Architectures

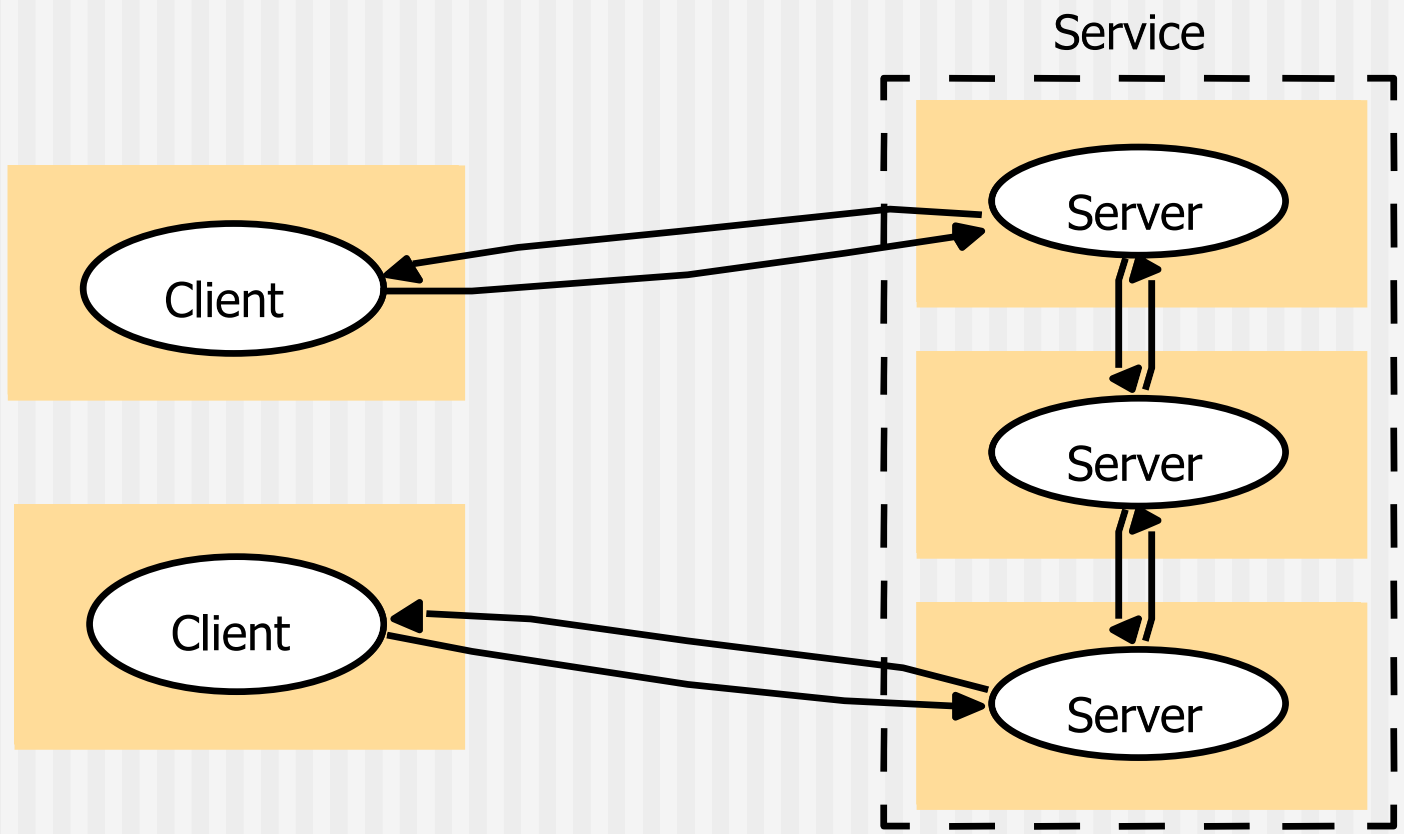
- The architecture include:
  - The division of responsibilities between system components.
  - The placement of the components on computers in the network.
- **Client-server model:**
  - Most important and most widely distributed system architecture.
  - Client and server roles are assigned and changeable.
    - Servers may in turn be clients of other servers.
  - Services may be implemented as several interacting processes in different host computers to provide a service to client processes:
    - Servers partition the set of objects on which the service is based and distribute them among themselves (e.g. Web data and web servers)
    - Servers maintain replicated copies of the service objects on several hosts for reliability and performance (e.g. AltaVista)

System Architectures



Clients invoke individual servers

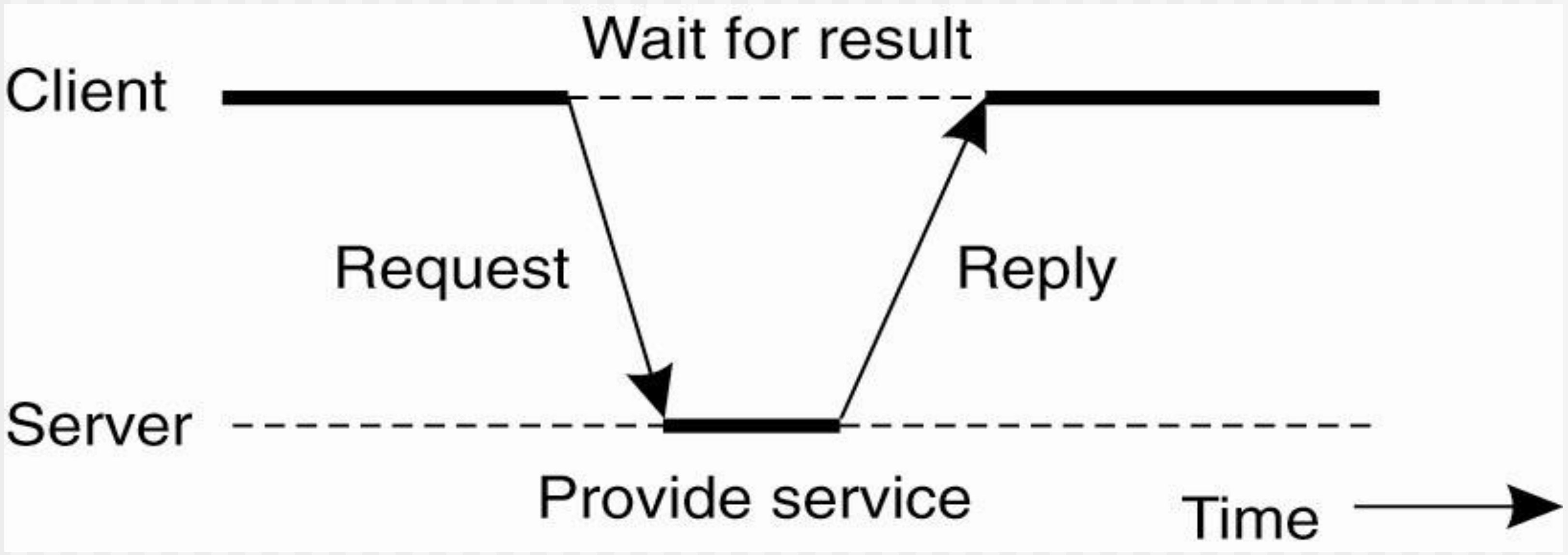
System Architectures



A service provided by multiple servers

Centralized Architectures

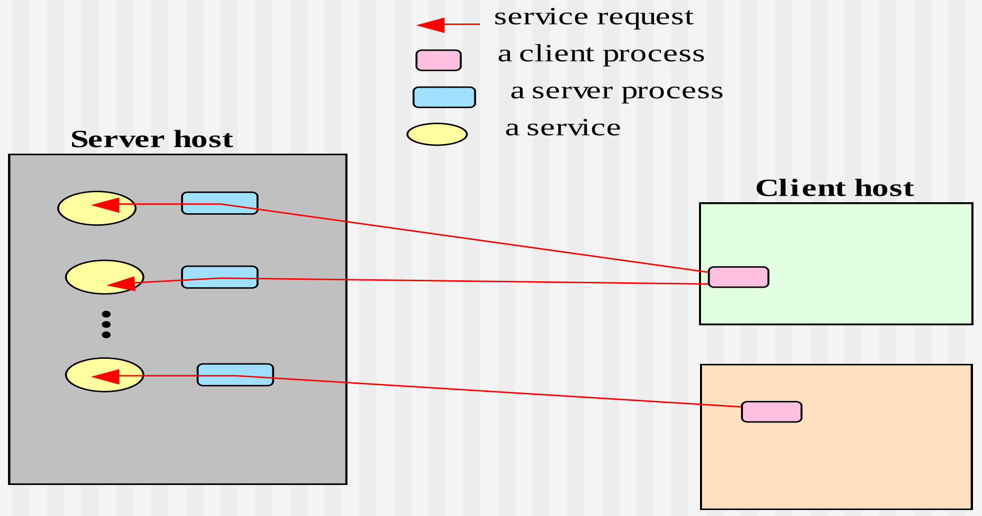
- Figure 2-3. General interaction between a client and a server.



The Client-Server Paradigm(نموذج)

Perhaps the best known paradigm for network applications, the client-server[2] model assigns asymmetric roles to two collaborating processes.

One process, the server, plays the role of a service provider which waits passively for the arrival of requests. The other, the client, issues specific requests to the server and awaits its response.



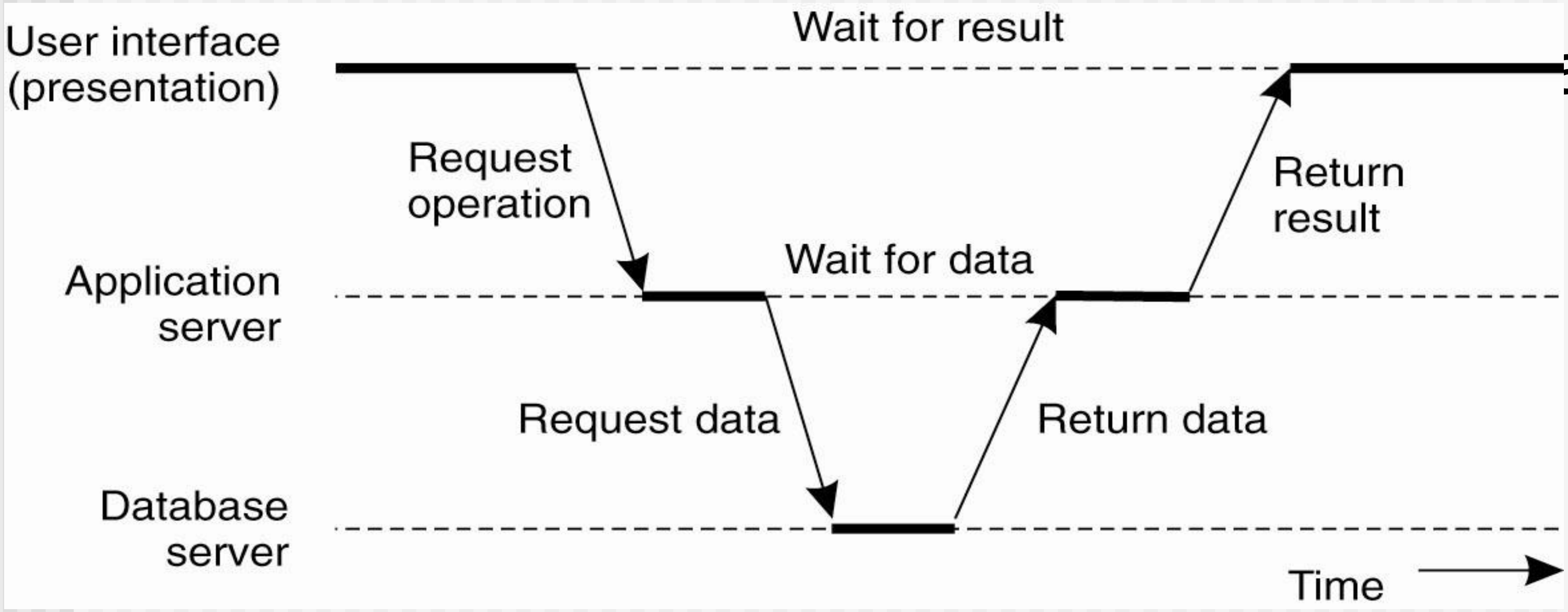
The Client-Server Paradigm, conceptual

The Client-Server Paradigm - 2

- Simple in concept, the client-server model provides an efficient abstraction for the delivery of network services.
- Operations required include those for a server process to listen and to accept requests, and for a client process to issue requests and accept responses.
- By assigning asymmetric roles to the two sides, event synchronization is simplified: the server process waits for requests, and the client in turn waits for responses.
- Many Internet services are client-server applications. These services are often known by the protocol that the application implements. Well known Internet services include HTTP, FTP, DNS, finger, gopher, etc.

Multitiered Architectures (3)

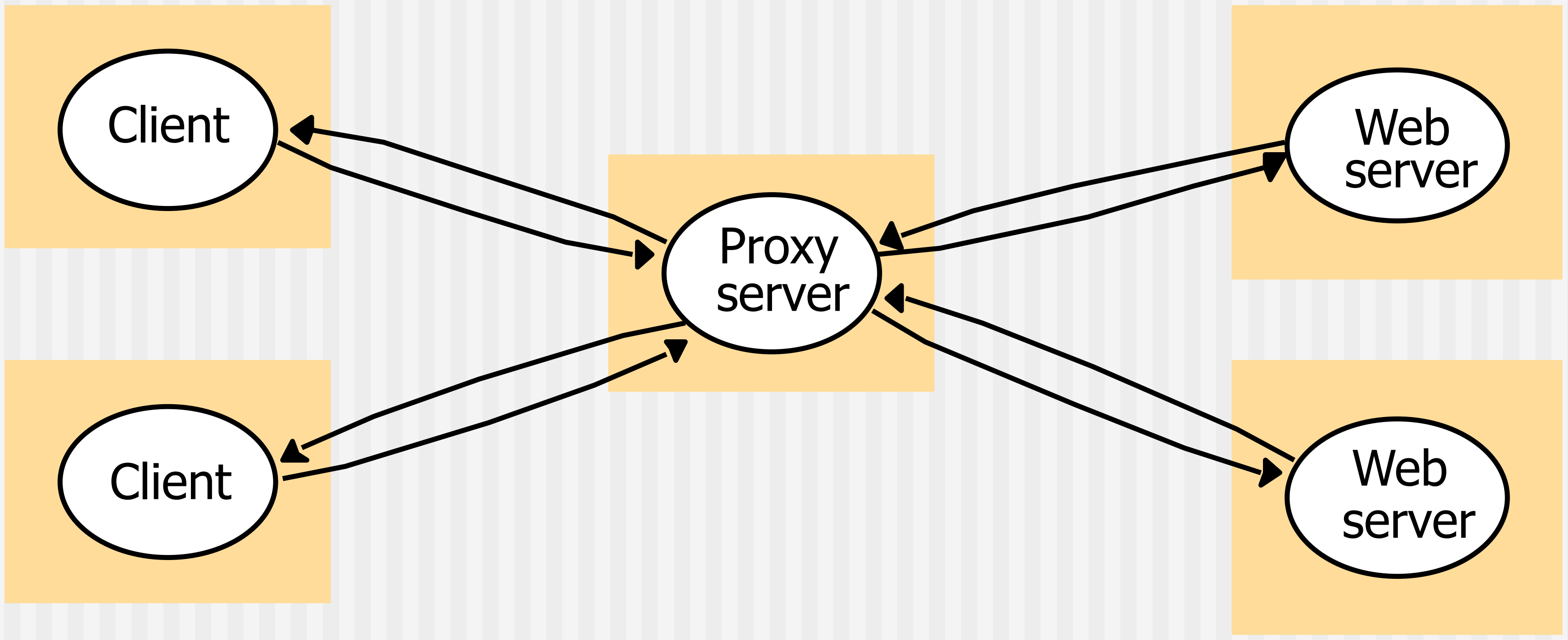
as



System Architectures

- **Caches and proxy servers:**
  - Cache:
    - A store of recently used data objects that is closer to the client process than those remote objects.
    - When an object is needed by a client process the caching service checks the cache and supplies the object from there in case of an up-to-date copy is available.
  - Proxy server:
    - Provides a shared cache of web resources for client machines at a site or across several sites.
    - Increase availability and performance of a service by reducing load on the WAN and web servers.
    - May be used to access remote web servers through a firewall.

System Architectures

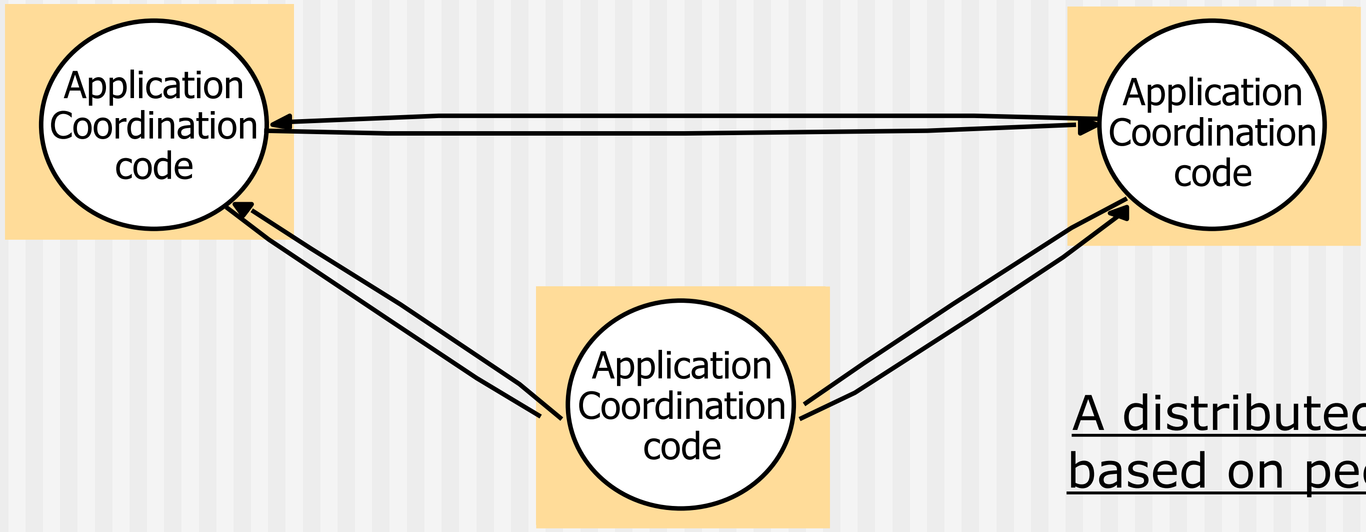


Web proxy server

System Architectures

⌘ **Peer processes:**

- All processes play similar roles without destination as a client or a server.
- Interacting cooperatively to perform a distributed activity.
- Communications pattern will depend on application requirements.


Application Coordination code
Application Coordination code
Application Coordination code

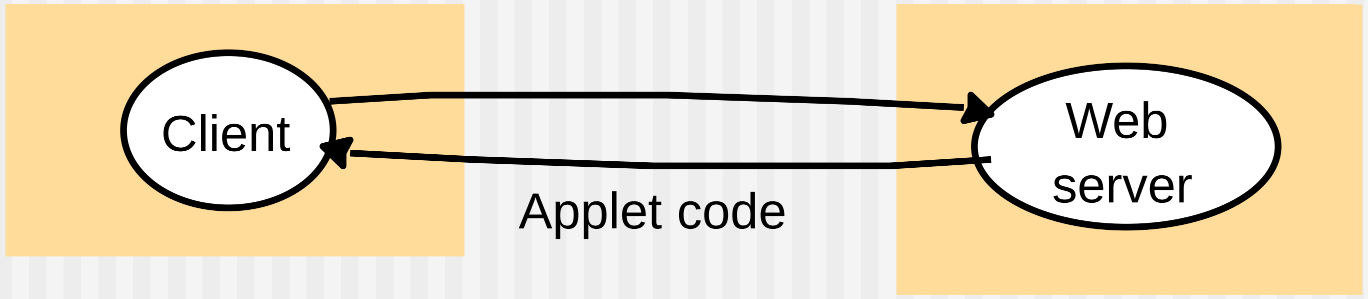
A distributed application based on peer processes

Client-server Model Variations (Mobile Code)

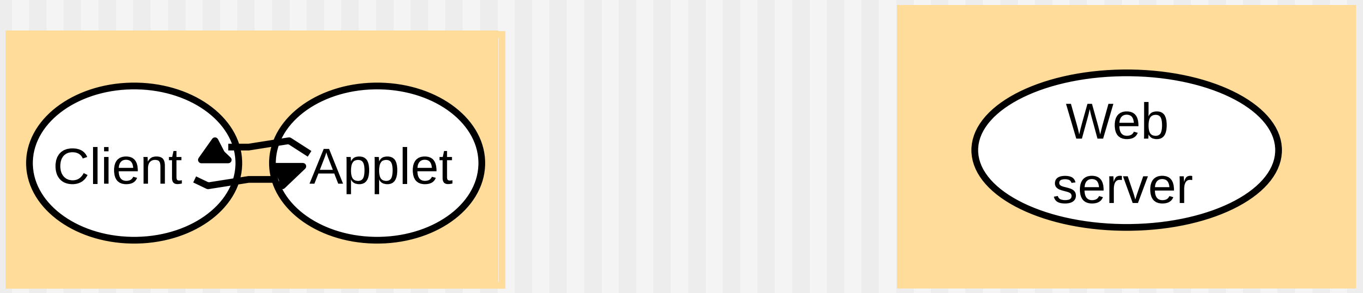
- Example: Java applets
  - The user running a browser selects a link to an applets whose code is stored on a web server.
  - The code is downloaded to the browser and runs there.
- Advantage:
  - Good interactive response since.
  - Does not suffer from the delays or variability of bandwidth associated with network communication.
- Disadvantage:
  - Security threat to the local resources in the destination computer.

Client-server Model Variations (Mobile Code)

a) client request results in the downloading of applet code



b) client interacts with the applet



<u>Web applets</u>

Client-server Model Variations (Mobile Agents)

- 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.
- Can make many invocations to local resources at each visited site.
- Visited sites must decide which local resources are allowed to use based on the identity of the user owning the agent.
- Advantage: Reduce communication cost and time by replacing remote invocation with local ones.
- Disadvantages:
  - Limited applicability.
  - Security threat of the visited sites resources.

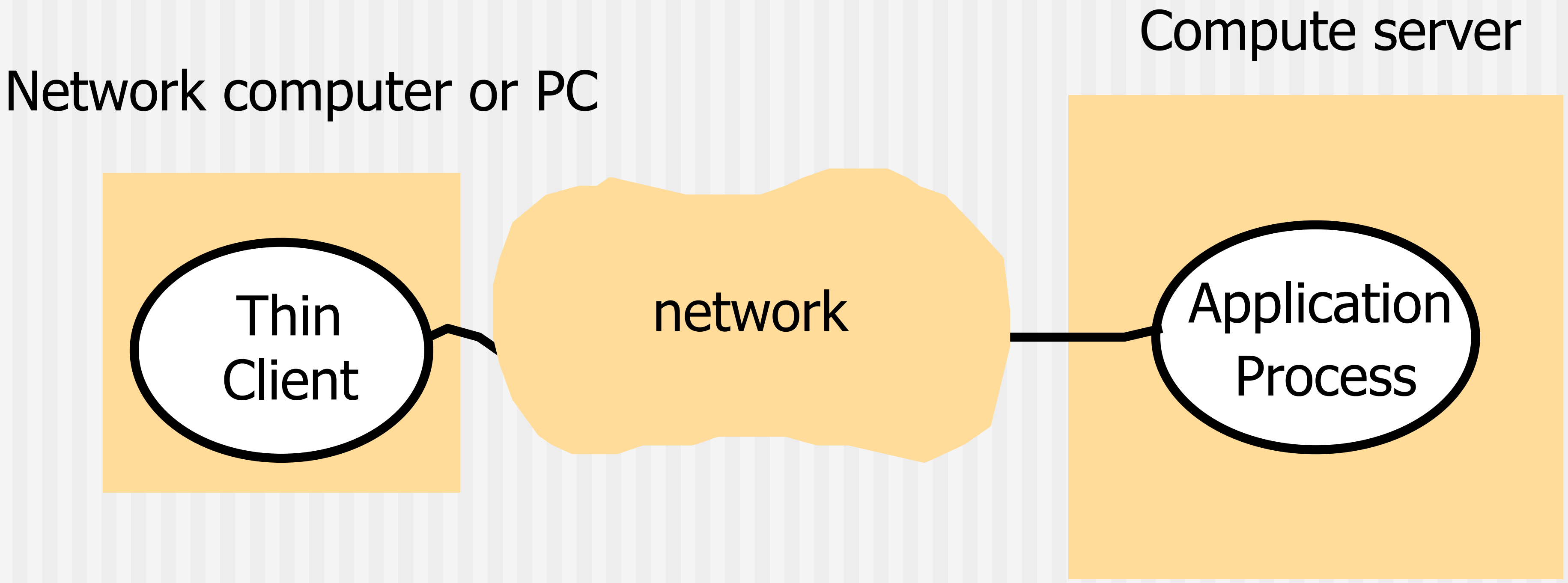
Client-server Model Variations (Network Computers)

- Downloads its operating system and any applications needed by the user from a remote file server.
- Applications run locally but files are managed by a remote file server.
- Users can migrate from one network computer to another.
- Its processor and memory capacities can be restricted to reduce its cost.
- Its disk (if included) holds only a minimum of software and use the reminder space as cache storage to hold copies of the most recently software and data files loaded from servers.

Client-server Model Variations (Thin Clients)

- Software layer that supports a window-based user interface on a local computer while executing application programs on a remote computer.
- Same as the network computer scheme but instead of downloading the applications code into the user's computer, it runs them on a server machine, *compute server*.
- Compute server is a powerful computer that has the capacity to run large numbers of applications simultaneously.
- Disadvantage: Increasing of the delays in highly interactive graphical applications

Client-server Model Variations (Thin Clients)



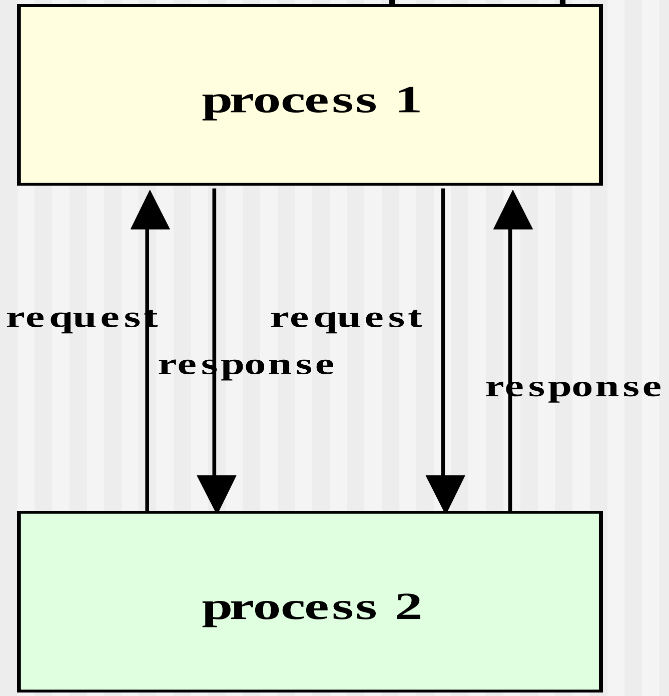
Thin clients and compute servers

The Peer-to-Peer System Architecture http://www.peer-to-peerwg.org/whatis/index.html

- In system architecture and networks, peer-to-peer is an architecture where computer resources and services are direct exchanged between computer systems.
- These resources and services include the exchange of information, processing cycles, cache storage, and disk storage for files..
- In such an architecture, computers that have traditionally been used solely as clients communicate directly among themselves and can act as both clients and servers, assuming whatever role is most efficient for the network.

The Peer-to-Peer Distributed Computing Paradigm

In the peer-to-peer paradigm, the participating processes play equal roles, with equivalent capabilities and responsibilities (hence the term “peer”). Each participant may issue a request to another participant and receive a response.



Peer-to-Peer distributed computing

Whereas the client-server paradigm is an ideal model for a centralized network service, the peer-to-peer paradigm is more appropriate for applications such as instant messaging, peer-to-peer file transfers, video conferencing, and collaborative work. It is also possible for an application to be based on both the client-server model and the peer-to-peer model.

A well-known example of a peer-to-peer file transfer service is *Napster.com* or similar sites which allow files (primarily audio files) to be transmitted among computers on the Internet. It makes use of a server for directory in addition to the peer-to-peer computing.

The future of peer-to-peer

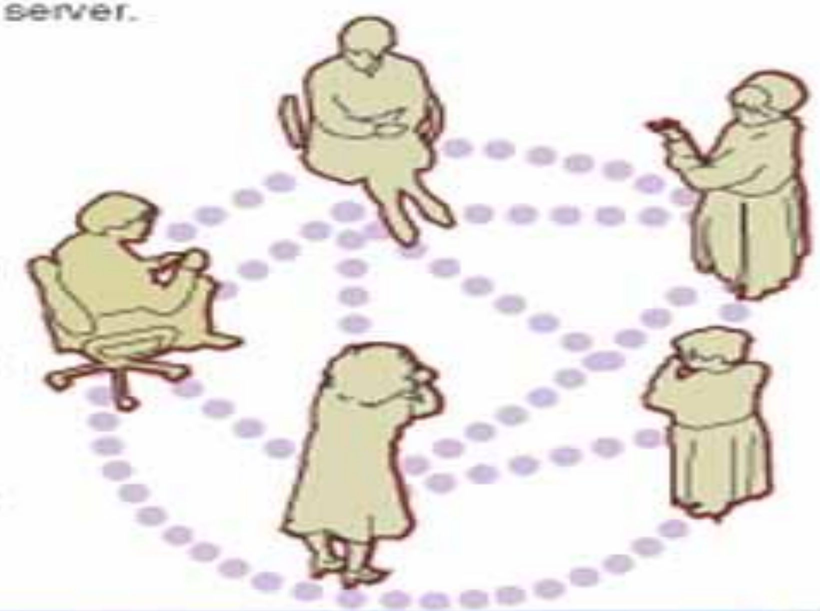
The file-swapping technology popularized by Napster, known as peer-to-peer networking, is about to change how people and corporations use the Internet. Instead of relying on central servers to process and relay information, new applications being developed will allow users to turn any computing device into a server.

A virtual meeting room

Users logon to the Internet using a program that looks like an online chat room.

A file is placed into a "shared space" within the virtual meeting room, which allows users to work on data files at the same time.

Users work in real time and can instant message each other. In the future, this might be done through devices such handhelds and cell phones.

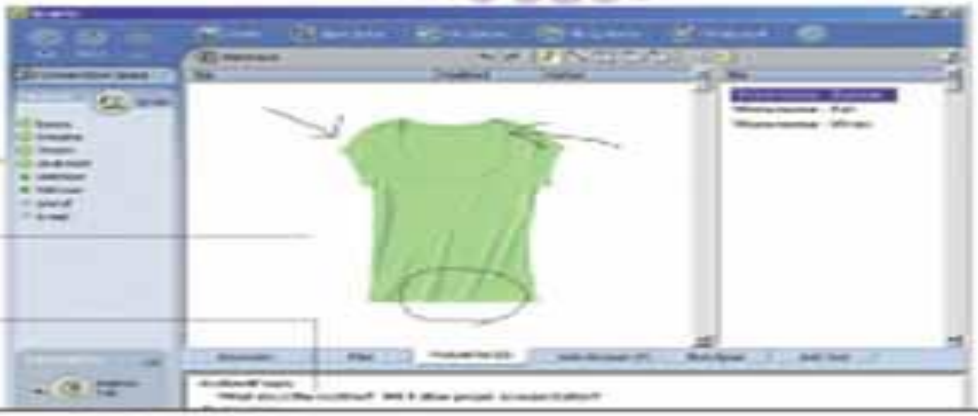


What is on the screen

Names of online users

File that is being worked on

Instant messaging



SOURCES: Groove Networks; compiled from AP wire reports AP

Peer-to-Peer distributed computing

The peer-to-peer paradigm can be implemented with facilities using any tool that provide message-passing, or with a higher-level tool such as one that supports the point-to-point model of the Message System paradigm.

For web applications, the ***web agent*** is a protocol promoted by the XNSORG (the XNS Public Trust Organization) for peer-to-peer interprocess communication

"Project JXTA is a set of open, generalized peer-to-peer protocols that allow any connected device (cell phone, to PDA, PC to server) on the network to communicate and collaborate. JXTA is short for Juxtapose, as in side by side. It is a recognition that peer to peer is juxtapose to client server or Web based computing -- what is considered today's traditional computing model. "

Interfaces and Objects

- Interface definitions specify the set of functions available for invocation in a server (or peer) processes.
- In OO paradigm, distributed processes can be constructed as objects whose methods can be accessed by remote invocation (COBRA approach).
- Many objects can be encapsulated in server or peer processes.
- Number, types, and locations (in some implementation) of objects may change dynamically as system activates require.
- Therefore, new services can be instantiated and immediately be made available for invocation.

The Message Passing Paradigm

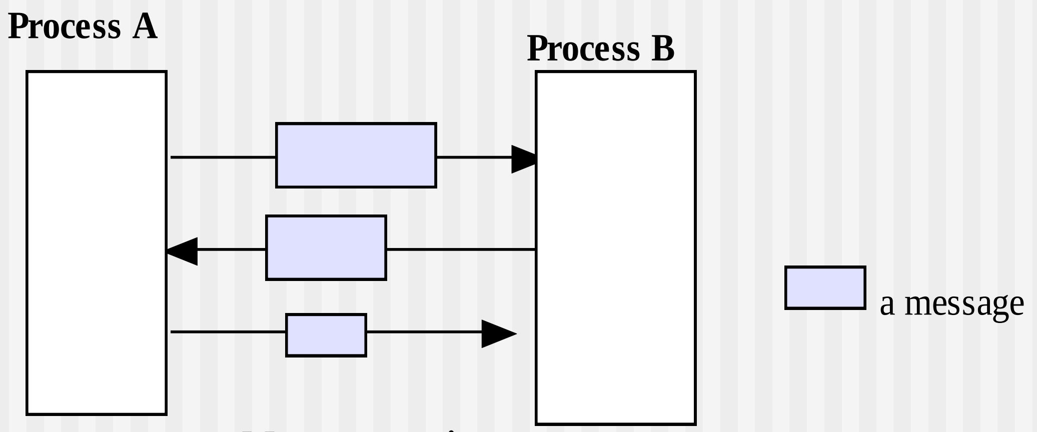
Message passing is the most fundamental paradigm for distributed applications.

- **A process sends a message representing a request.**
- **The message is delivered to a receiver, which processes the request, and sends a message in response.**
- **In turn, the reply may trigger a further request, which leads to a subsequent reply, and so forth. The message-**

The Message Passing Paradigm - 2

Message passing is the most fundamental paradigm for distributed applications.

- **A process sends a message representing a request.**
- **The message is delivered to a receiver, which processes the request, and sends a message in response.**
- **In turn, the reply may trigger a further request, which leads to a subsequent reply, and so forth. -**



Message passing

The Message Passing Paradigm - 3

- **The basic operations required to support the basic message passing paradigm are *send*, and *receive*.**
- **For connection-oriented communication, the operations *connect* and *disconnect* are also required.**
- **With the abstraction provided by this model, the interconnected processes perform input and output to each other, in a manner similar to file I/O. The I/O operations encapsulate the detail of network communication at the operating-system level.**
- **The socket application programming interface is based on this paradigm.**
  - **http://java.sun.com/products/jdk/1.2/docs/api/index.html**
  - **http://www.sockets.com/**

Architectures Design Requirements

- **Performance Issues:**
  - Considered under the following factors:
    - Responsiveness:
      - Fast and consistent response time is important for the users of interactive applications.
      - Response speed is determined by the load and performance of the server and the network and the delay in all the involved software components.
      - System must be composed of relatively few software layers and small quantities of transferred data to achieve good response times.
    - Throughput:
      - The rate at which work is done for all users in a distributed system.
    - Load balancing:
      - Enable applications and service processes to proceed concurrently without competing for the same resources.
      - Exploit (مأثرة)available processing resources.

Architectures Design Requirements

- **Quality of Service:**
  - Main system properties that affect the service quality are:
    - Reliability: related to failure fundamental model (discussed later).
    - Performance: ability to meet timeliness guarantees.
    - Security: related to security fundamental model (discussed later).
    - Adaptability: ability to meet changing resource availability and system configurations.
- **Dependability issues:**
  - A requirement in most application domains.
  - Achieved by:
    - Fault tolerance: continuing to function in the presence of failures.
    - Security: locate sensitive data only in secure computers.
    - Correctness of distributed concurrent programs: research topic.

Fundamental Models (Interaction Model)

- Distributed systems consists of multiple interacting processes with private set of data that can access.
- Distributed processes behavior is described by *distributed algorithms*.
- Distributed algorithms define the steps to be taken by each process in the system including the transmission of messages between them.
- Transmitted messages transfer information between these processes and coordinate their ordering and synchronization activities.

Fundamental Models (Interaction Model)

⌘ Interacting processes in a distributed system are affected by two significant factors:

1. Performance of communication channels: is characterized by:
   - **Latency**: delay between sending and receipt of a message including
     - Network access time.
     - Time for first bit transmitted through a network to reach its destination.
     - Processing time within the sending and receiving processes.
   - **Throughput**: number of units (e.g., packets) delivered per time unit.
   - **Bandwidth**: total amount of information transmitted per time unit.
   - **Jitter**: variation in the time taken to deliver multiple messages of the same type (relevant to multimedia data).

Fundamental Models (Interaction Model)

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.
- Several approaches to correcting the times on computer clocks are proposed.
- Clock corrections can be made by sending messages, from a computer has an accurate time to other computers, which will still be affected by network delays.

Fundamental Models (Interaction Model)

- Setting time limits for process execution, as message delivery, in a distributed system is hard.
- Two opposing extreme positions provide a pair of simple interaction models:
  - *Synchronous* distributed systems:
    - A system in which the following bounds are defined:
      - 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 perfect time has a known bound.
    - Easier to handle, but determining realistic bounds can be hard or impossible.

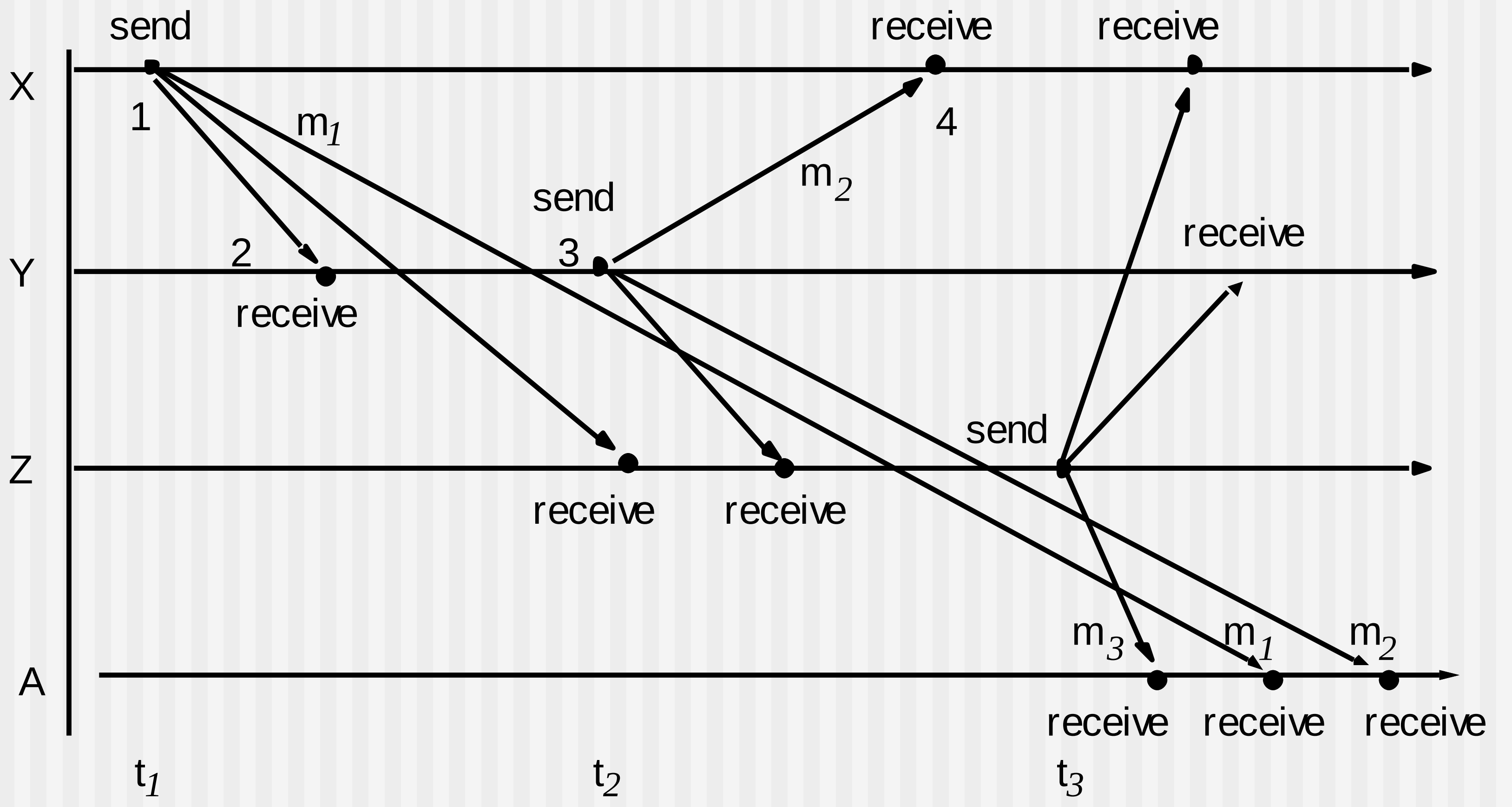
Fundamental Models
(Interaction Model)

- *Asynchronous* distributed systems:
  - A system in which there are no bounds on:
    - process execution times.
    - message delivery times.
    - clock drift rate.
  - Allows no assumptions about the time intervals involved in any execution.
  - Exactly models the Internet.
    - Browsers are designed to allow users to do other things while they are waiting.
  - More abstract and general:
    - A distributed algorithm executing on one system is likely to also work on another one.

Fundamental Models
(Interaction Model)

- **<u>Event ordering</u>**: when need to know if an event at one process (sending or receiving a message) occurred before, after, or concurrently with another event at another process.
- It is impossible for any process in a distributed system to have a view on the current global state of the system.
- The execution of a system can be described in terms of events and their ordering despite the lack of accurate clocks.
- Logical clocks define some event order based on causality.
- Logical time can be used to provide ordering among events in different computers in a distributed system (since real clocks cannot be synchronized).

Fundamental Models
(Interaction Model)

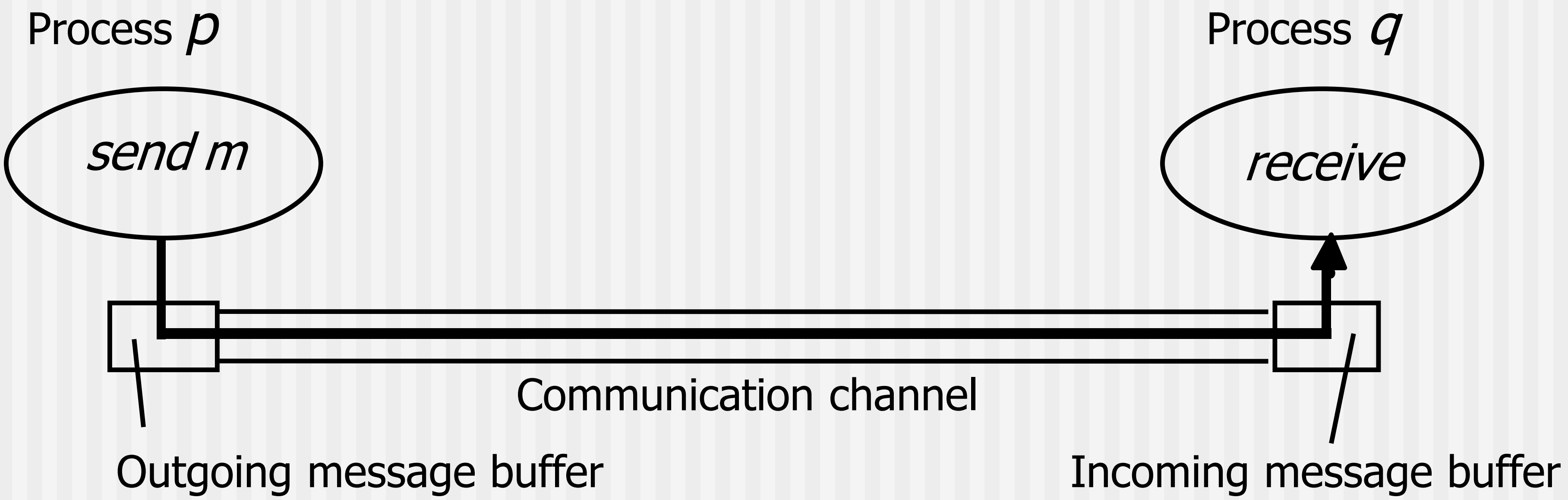


<u>Real-time ordering of events</u>

Fundamental Models (Failure Model)

- Defines the ways in which failure may occur in order to provide an understanding of its effects.
- A taxonomy of failures which distinguish between the failures of processes and communication channels is provided:
  - *Omission* failures
    - Process or channel failed to do something.
  - *Arbitrary* failures
    - Any type of error can occur in processes or channels (worst).
  - *Timing* failures
    - Applicable only to synchronous distributed systems where time limits may not be met.

Fundamental Models
(Failure Model)



<u>Processes and channels</u>

Fundamental Models (Failure Model)

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,* 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, commit omissions; a process may stop or take an incorrect step.

Fundamental Models
(Failure Model)

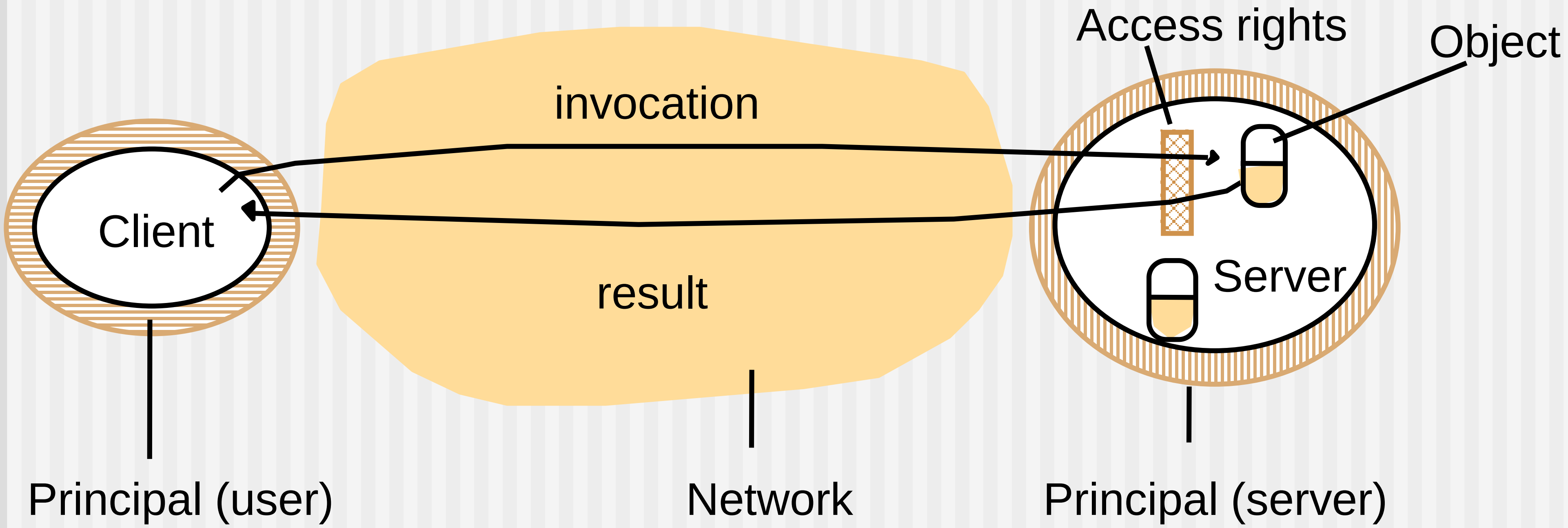
Timing failures

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.

Fundamental Models (Security Model)

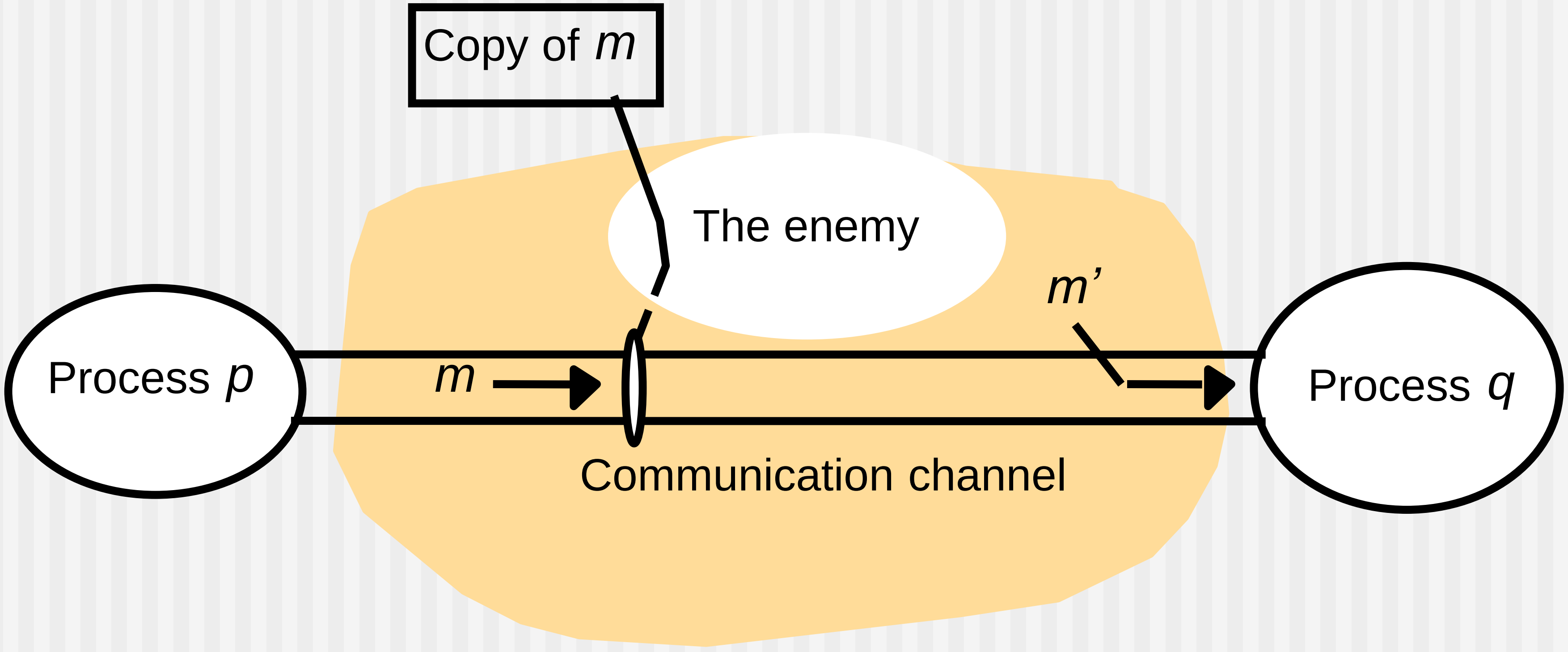
- Secure processes and channels and protect objects encapsulated against unauthorized access.
- **Protecting access to objects**
  - Access rights
  - In client server systems: involves authentication of clients.
- **Protecting processes and interactions**
  - Threats to processes: problem of unauthenticated requests / replies.
    - e.g., "man in the middle"
  - Threats to communication channels: enemy may copy, alter or inject messages as they travel across network.
    - Use of “secure” channels, based on cryptographic methods.

Fundamental Models
(Security Model)



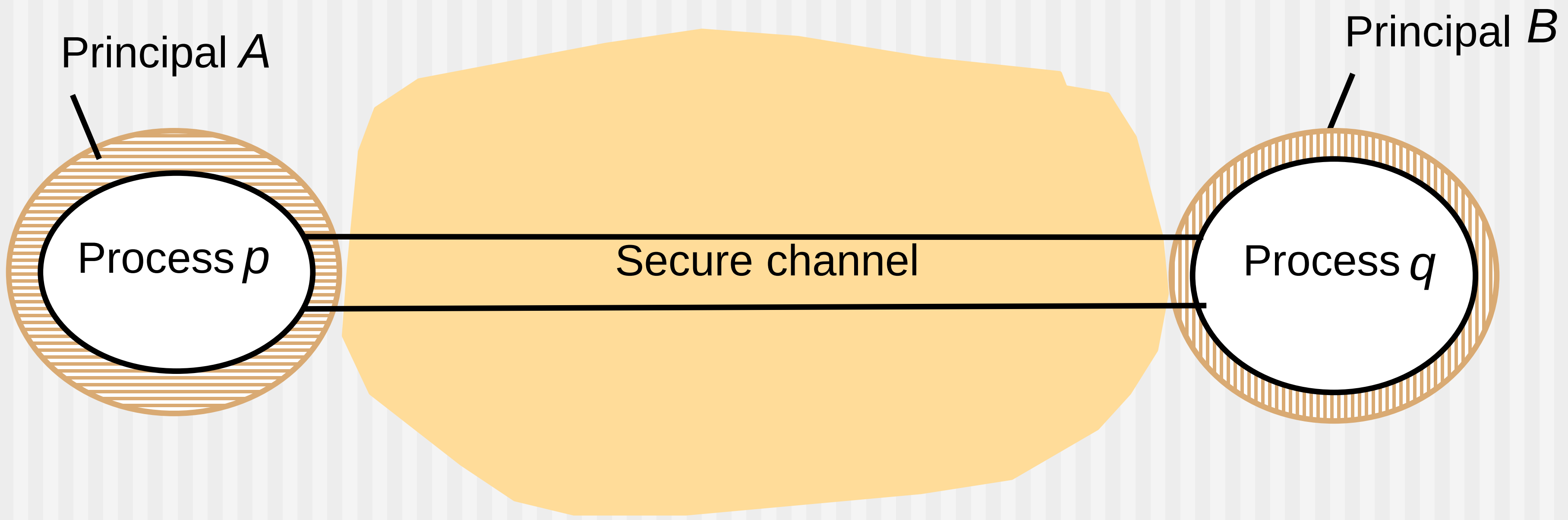
<u>Objects and principals</u>

Fundamental Models
(Security Model)



<u>The enemy</u>

Fundamental Models
(Security Model)



Secure channels

Fundamental Models (Security Model)

- **Denial of service**
  - e.g., “pings” to selected web sites
  - Generating debilitating network or server load so that network services become de facto unavailable
- **Mobile code:**
  - Requires executability privileges on target machine
  - Code may be malicious (e.g., mail worms)