# Attention Please

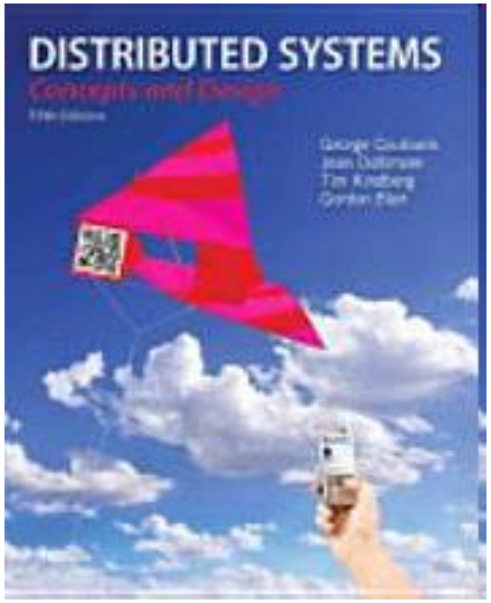
- The Research Project web page is available now. Please check our class web page.

- Grader: Prashant Joshi [prashantjoshi93@gmail.com]

  Office Hours: MW 11am~12pm at GMCS 557

Please send the grader your OS proof by email.

# Chapter 2: System Models



This chapter provides an explanation of important ways in which the design of distributed systems can be described and discussed

## Objectives Today

- **Architectural models**

  Architectural models determine the distribution of data and computational tasks amongst the physical nodes of the system.

- **Fundamental models**

  Each fundamental model represents a set of issues that must be addressed in the design of distributed systems.

## Definitions of Model

- A hypothetical description of a complex entity or process.
- A description of observed behavior, simplified by ignoring certain details.
- Models allow complex systems to be understood and their behavior predicted within the scope of the model, but may give incorrect descriptions and predictions for situations outside the realm of their intended use.
- A model may be used as the basis for simulation.

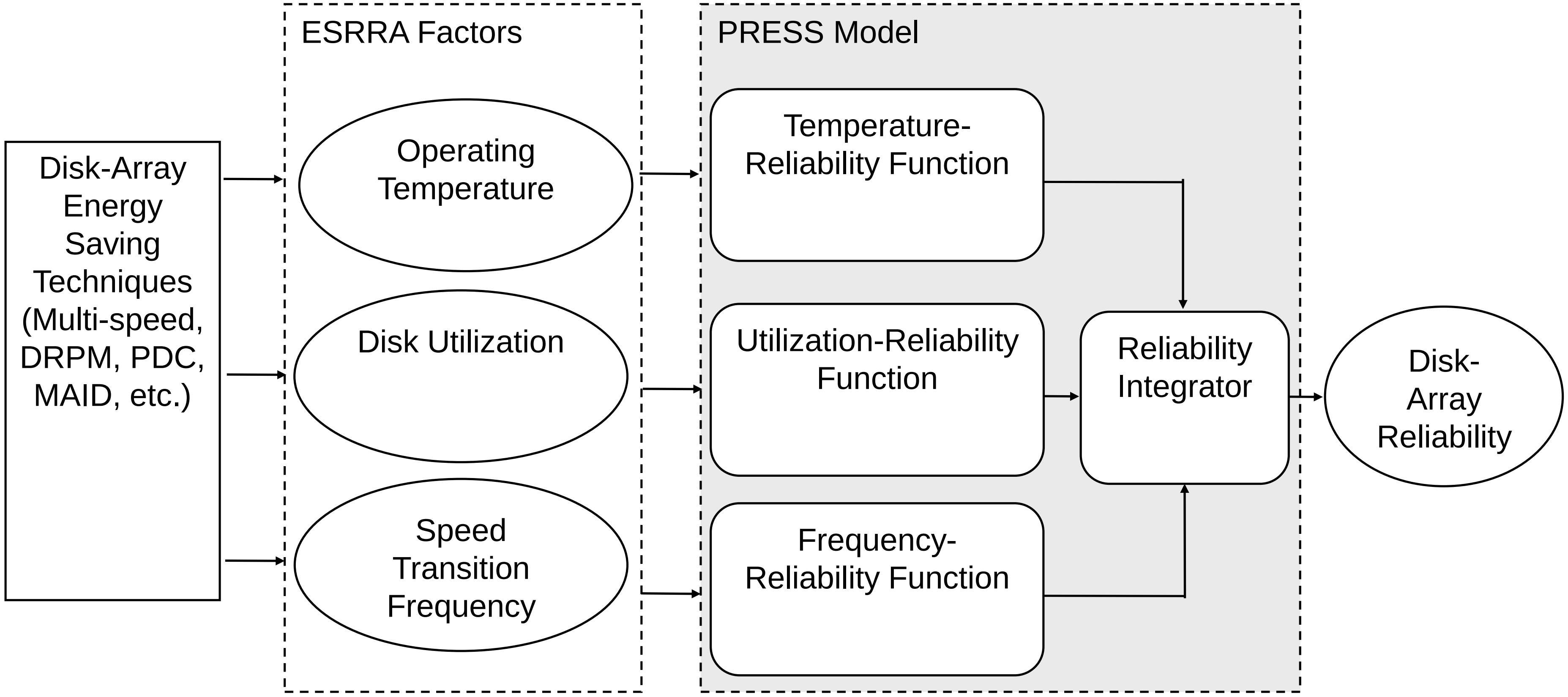
## A model has to address:

- What are the main entities in the system?
- How do they interact
- What are the characteristics that affect their individual and collective behaviour?

## The purpose of a model:

- To make explicit all the relevant assumptions about the system we are modelling.
- To make generalizations concerning what is possible or impossible, given those assumptions.

# An Example of Model



Disk-Array Energy Saving Techniques (Multi-speed, DRPM, PDC, MAID, etc.)
ESRRA Factors
Operating Temperature
Disk Utilization
Speed Transition Frequency
PRESS Model
Temperature-Reliability Function
Utilization-Reliability Function
Frequency-Reliability Function
Reliability Integrator
Disk-Array Reliability

Fig. 1. Overall architecture of the PRESS model.

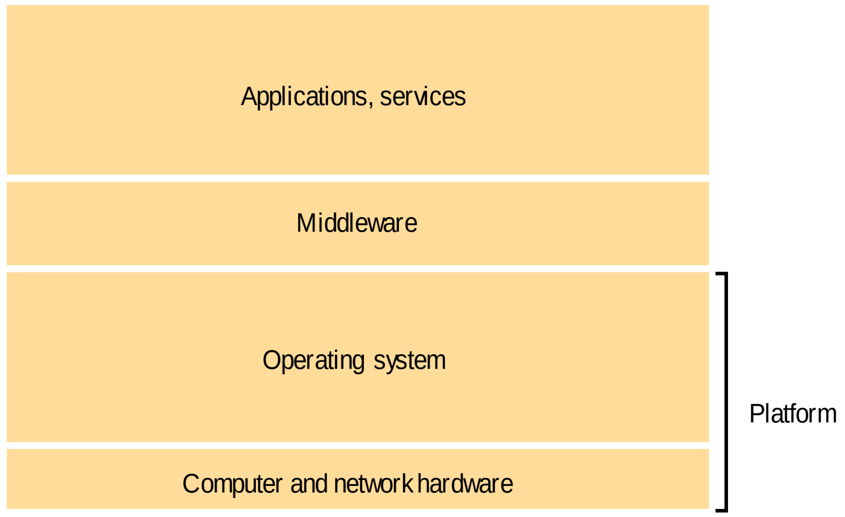
## Making An Architectual Model (3 steps)

- An architectural model of a distributed system simplifies and abstracts the functions of the individual components of a distributed system
- The placement of the components across a network of computers (distribution of data and workload)
- the interrelationships between the components (roles and communication patterns)

## Software 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 resource.

- masking heterogeneity of underlying platform, hence providing a common "platform"
- Examples:
  - Sun RPC, Java RMI
  - CORBA, Microsoft .NET, Java J2EE
- reliable services in the middle layer, still need application-specific reliability

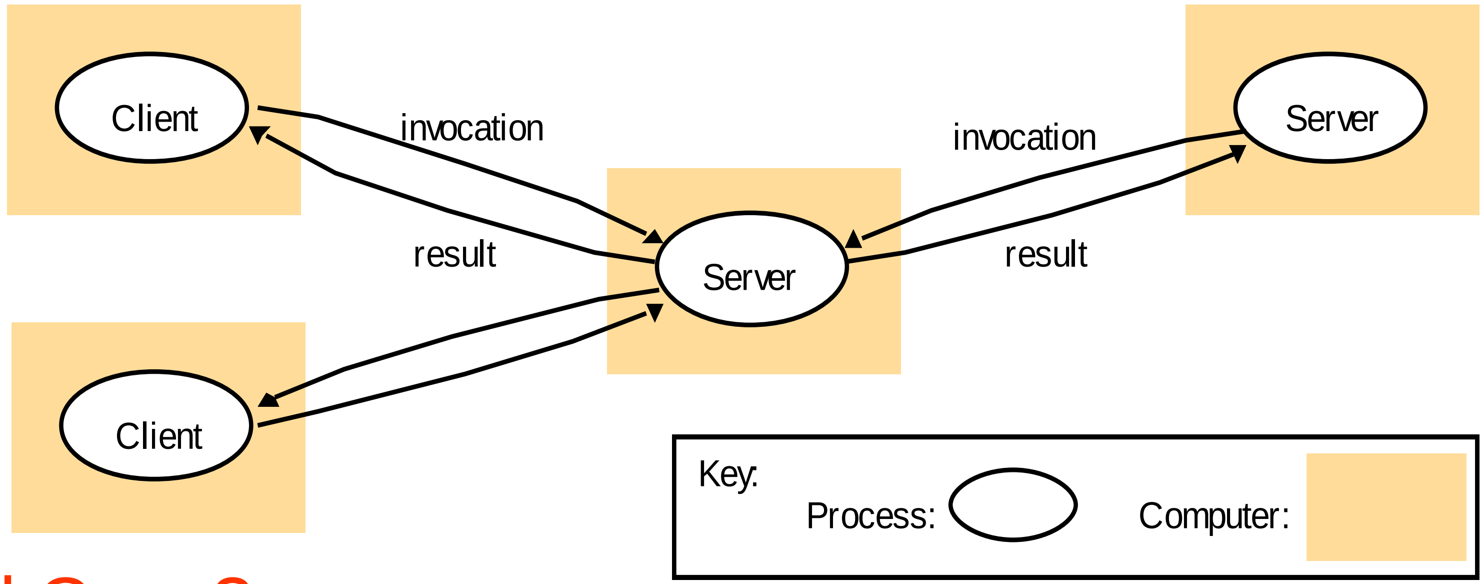


- Applications, services.
  What is a server? What is a distributed service?

A *server* is a process that accepts requests from other processes.

# Client-Server

- Servers provide services
- Clients access services

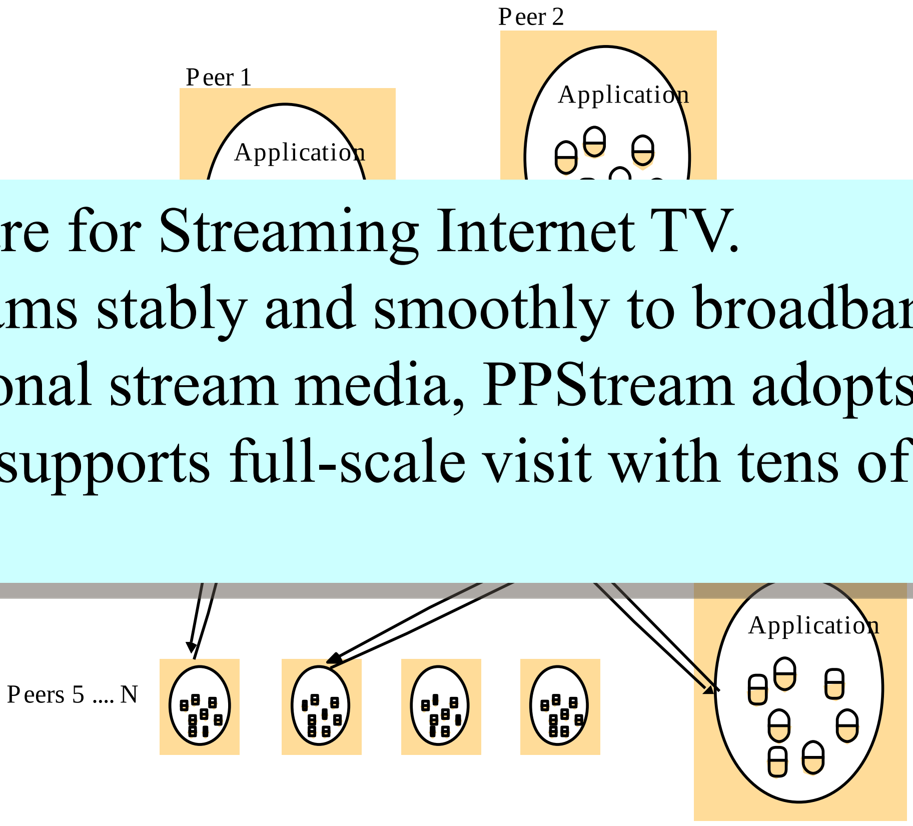


- Pros and Cons?

**Pros:** Simple and widely used; **Cons:** Poor scalability because of the centralization of service provision and management

# Peer to Peer

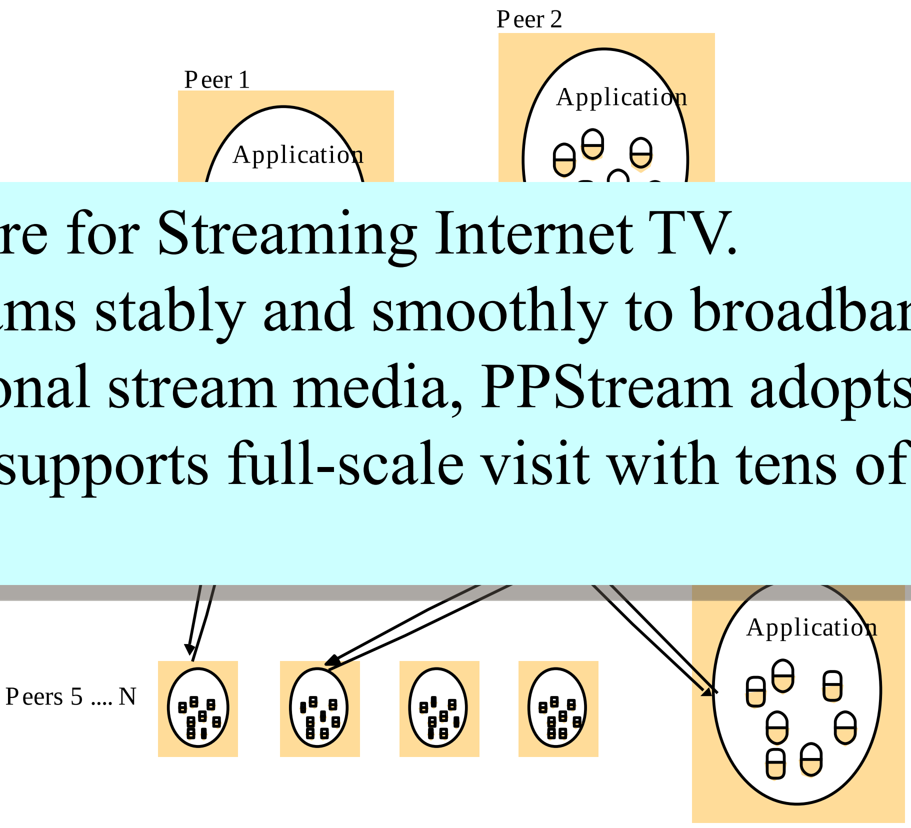
- Peers play similar roles
- No distinction of responsibilities
- Example?



**PPStream** is a P2P software for Streaming Internet TV.
It can broadcast TV programs stably and smoothly to broadband users. Compared to traditional stream media, PPStream adopts P2P - streaming technology and supports full-scale visit with tens of thousands of users online.

# Peer to Peer

- Peers play similar roles
- No distinction of responsibilities
- Example?



**PPStream** is a P2P software for Streaming Internet TV.
It can broadcast TV programs stably and smoothly to broadband users. Compared to traditional stream media, PPStream adopts P2P -streaming technology and supports full-scale visit with tens of thousands of users online.

## Features of Peer to Peer

1. This architecture responses to the need to **distribute shared resources** much more widely in order to share loads.
2. The aim of the P2P architecture is to exploit the resources (both data and hardware) in **a large number of participating computers** for the fulfillment of a given task.
3. The downside is that this architecture is **substantially more complex** than the client server architecture because of the need to place individual objects and retrieve them and to maintain replicas amongst many computers.

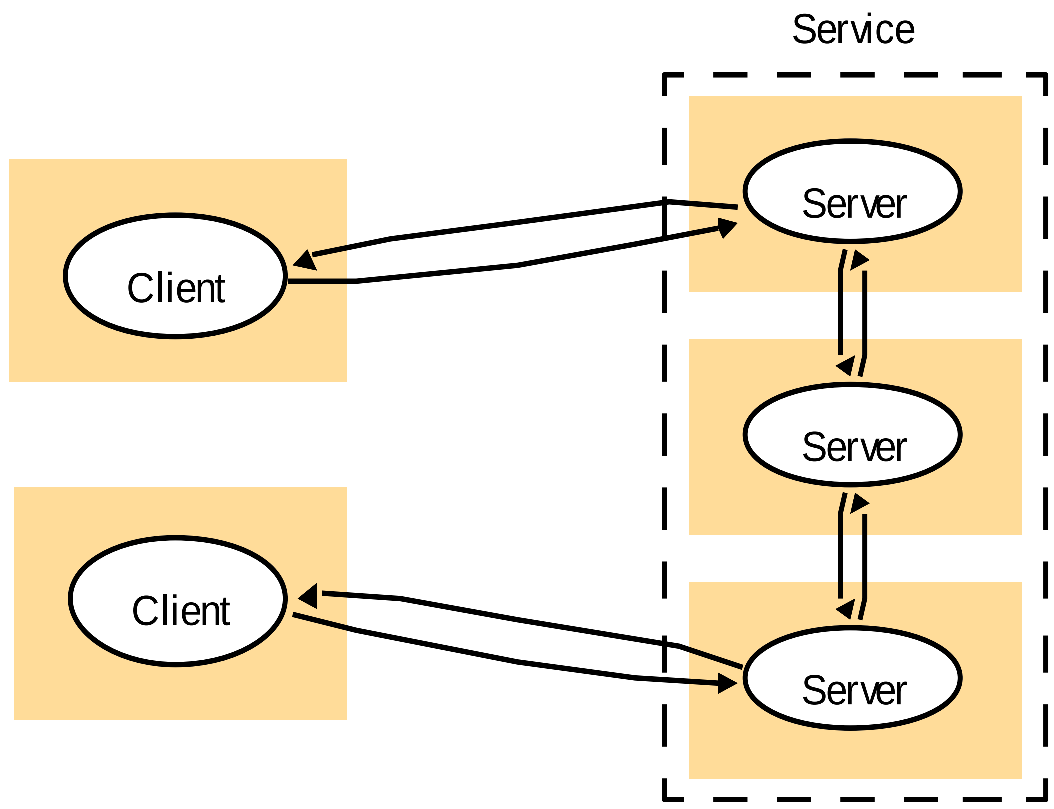
## Variations of Architectural Models

Several variations of the above two models can be derived from the considerations of the following factors:

- 1. The use of multiple servers and caches to increase performance
- 2. The use of mobile code and mobile agents
- 3. User's need for low-cost computers with limited hardware resources
- 4. The requirement to add and remove mobile device in a convenient manner.

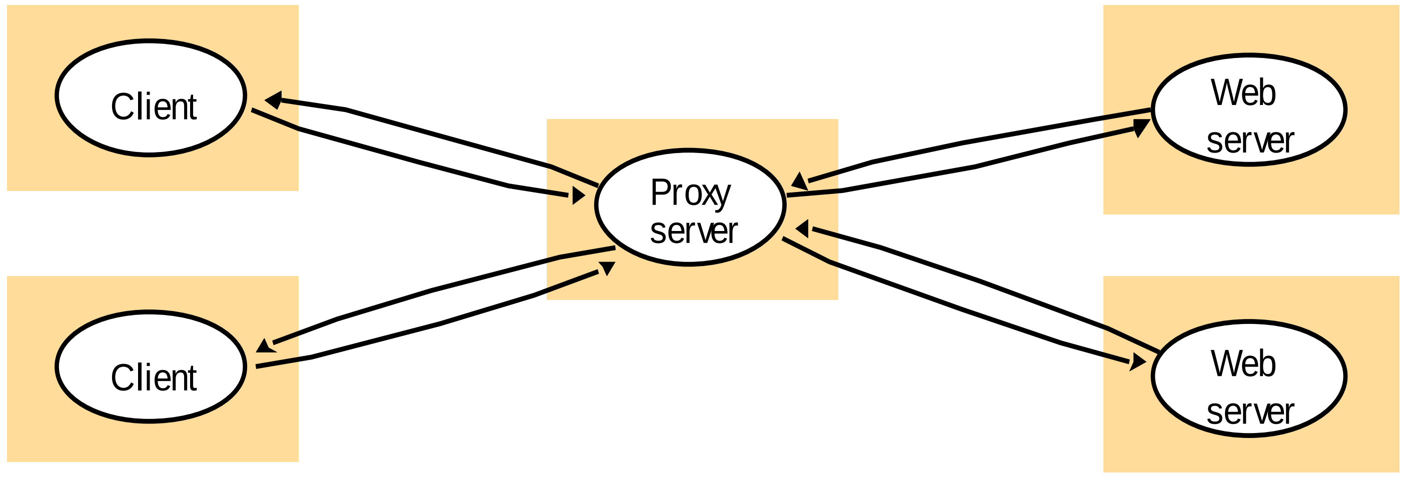
## A service provided by multiple servers

- Objects are partitioned/replicated
  - Web: each server manages its objects
  - NIS: replicated login/password info
  - Cluster: closely coupled, scalable (search engines)



# Web proxy servers and caches

- Cache: local copies of remote objects for faster access
- Browser cache
- Proxy server
  - Additional roles: filtering, firewall

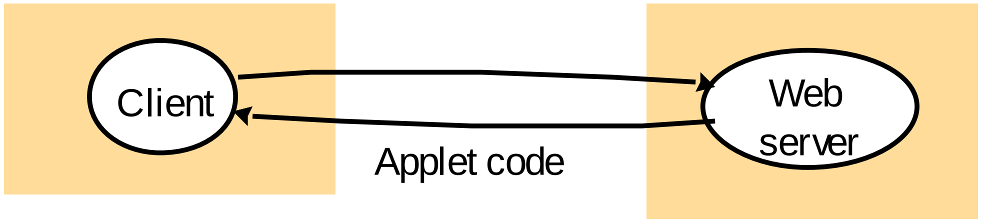


A ***cache*** is a store of recently used data objects that is closer than the objects themselves.
Caches may be located with each client or they may be located in a proxy server that can be shared by several clients

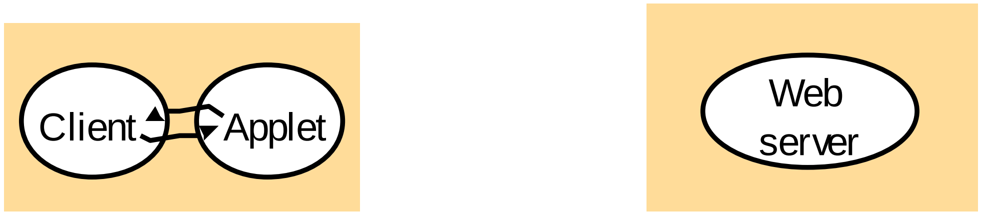
# Mobile code and web applets

- Running code locally vs. remotely
  - Network bandwidth
  - What examples have you seen?
  - Security issues?

a) client request results in the downloading of applet code



b) client interacts with the applet



A stockbroker might provide a customized service to notify customers of changes in the prices of shares; to use the service, each customer would have to download a special applet that receives updates from the broker’s server, displays them to the user and perhaps performs automatic buy and sell operations triggered by conditions set up by the customer and stored locally in the customer’s computer.

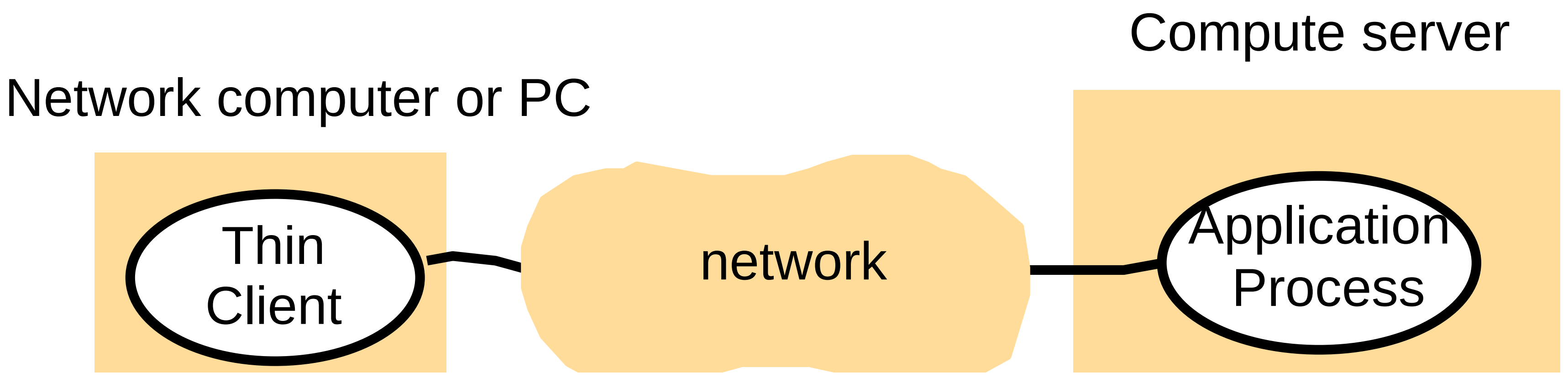
## Mobile agents

- Running program that travels from one computer to another
  - Perform tasks on users' behalf
  - Security issues
- How does it compare to one client interacting with multiple servers?
  - What if the program needs to access a lot of remote data?
- How does it compare with applets?

1. There is a reduction in communication cost and time through the replacement of remote invocations with local ones.
2. Applets can be used to initiate a conversation between server and clients, e.g., to perform an update so that clients have the up-to-date information of server.

# Thin clients and compute servers (network computers)

- Thin client
  - Basically a display with keyboard (“dumb terminal”)
  - Remote computation, storage
- How does it compare with the PC model or distributed workstation model? Pros and Cons?



Pros: low management and hardware costs.
Cons: highly interactive graphical activities such as CAD and image processing incur both network and operating system latencies.

## Mobile devices and spontaneous interoperation

- Laptops, PDA's, cell phones, wearable computers…
- Metropolitan (GSM, CDPD): hundreds of meters, 100s Kb/s
- Local Area (BlueTooth, infra-red): meters, 10 Mb/s
- Infrastructure vs ad-hoc networking
- Accessing services, variable bandwidth/connectivity, power supply, security,
- Spontaneous interoperation--easy connection and location of services
  - Service discovery
  - Context-aware

# Interfaces and objects

- At the programming level
  - Need standard interfaces
    - Access/provide services/objects
    - RPC (Remote Procedure Call)/RMI (Remote Method Invocation)

The set of functions available for invocation in a process (whether it is a server or a peer process) is specified by one or more interface definitions.

Example? In a basic form of client-server architecture, each server process is seen as a single entity with a fixed interface defining the functions that can be invoked in it.

# Design requirements for distributed architectures

- Performance
  - Responsiveness, throughput, load balancing
- Quality of service (QoS)
  - time-critical applications (real-time apps)
  - guarantee certain level of quality delivered by the deadline
  - allocation of computation and communication resources
- Caching and replication
  - web caching: server provides expiration time
- Dependability
  - fault tolerance: redundancy, recovery
  - security

## Fundamental Models

- Fundamental properties in processes and communication, shared among different architectures discussed previously

## Interaction model (1)

- sequential vs. distributed algorithms timing, distributed state
- performance of communication channels
  - latency: transmission, access, os
  - bandwidth
  - jitter: variation among messages
- clocks and timing events
  - clock drift
  - synchronization

distributed algorithm – a definition of the steps to be taken by each of the processes of which the system is composed., including the transmission of messages between them.

## Interaction Model (2)

- synchronous distributed systems
  - lower and upper bounds for execution of a step
  - message transmission in bounded time
  - clock drift rate is bounded
  - failures can be detected when bounds are exceeded
  - accomplished by allocating sufficient resources
- asynchronous distributed systems
  - no bounds on process speed, message delay, clock drift rate
  - failures are harder to detect
  - performance can't be guaranteed

## Exercise

- Consider a simple server that carries out client requests without accessing other servers. Explain why it is generally not possible to set a limit on the time taken by such a server to respond to a client request. What would need to be done to make the server able to execute requests within a bounded time? Is this a practical option?

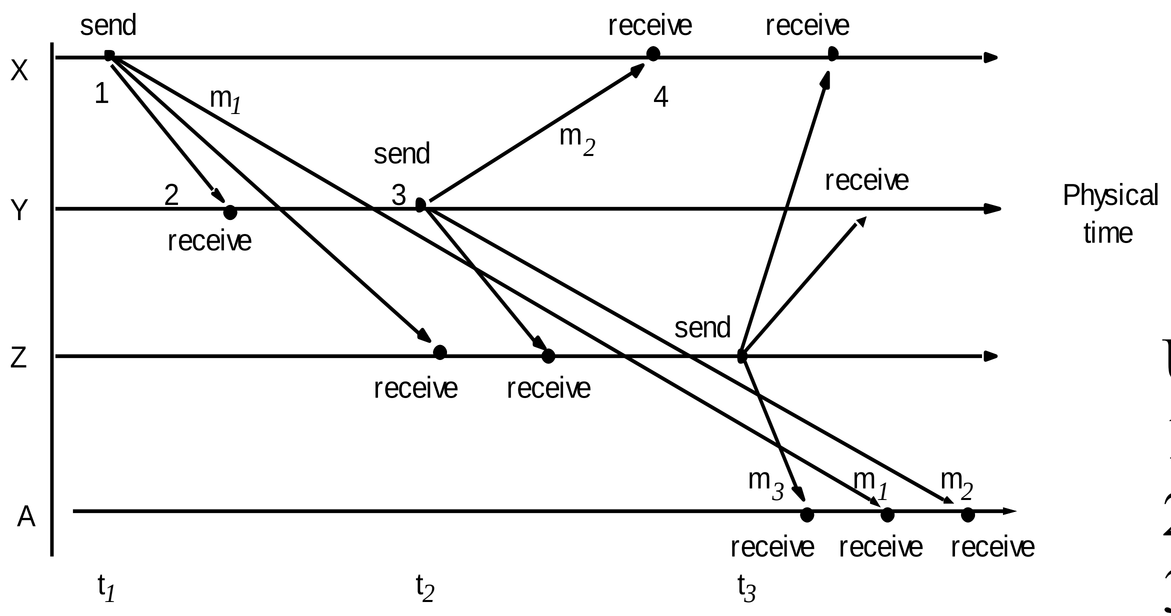
## Answer

- **The rate of arrival of client requests is unpredictable**. If the server uses threads to execute the requests concurrently, it may not be able to allocate sufficient time to a particular request within any given time limit. If the server queues the request and carries them out one at a time, they may wait in the queue for an unlimited amount of time.
- To execute requests within bounded time, **limit the number of clients to suit its capacity**. To deal with more clients, use a server with more processors. After that, (or instead) replicate the service....

- **The solution may be costly** and in some cases keeping the replicas consistent may take up useful processing cycles, reducing those available for executing requests.

# Interaction Model (3)

- event ordering
  - Relative ordering might be more important than exact time
  - logical clocks--ordering events without physical clocks



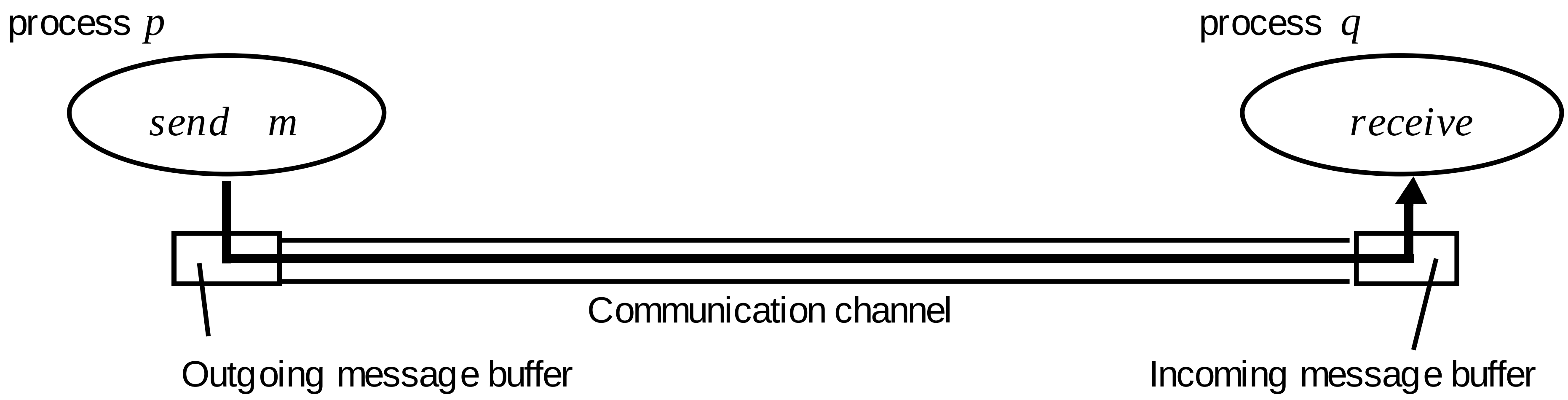
User A:

1. From Z: Re: Meeting
2. From X: Meeting
3. From Y: Re: Meeting

Users X, Y, Z, and A are 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*;

## Failure Model (1)

- ⌘Failure of processes or communication channels



## Failure Model (2):
## 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 |

The faults classified as omission failures refer to cases when a process or communication channel fails to perform actions that it is supposed to do.

A Byzantine failure, in which a component of some system not only behaves **erroneously**, but also fails to behave **consistently** when interacting with multiple other components.

## Failure Model (3): Timing failures

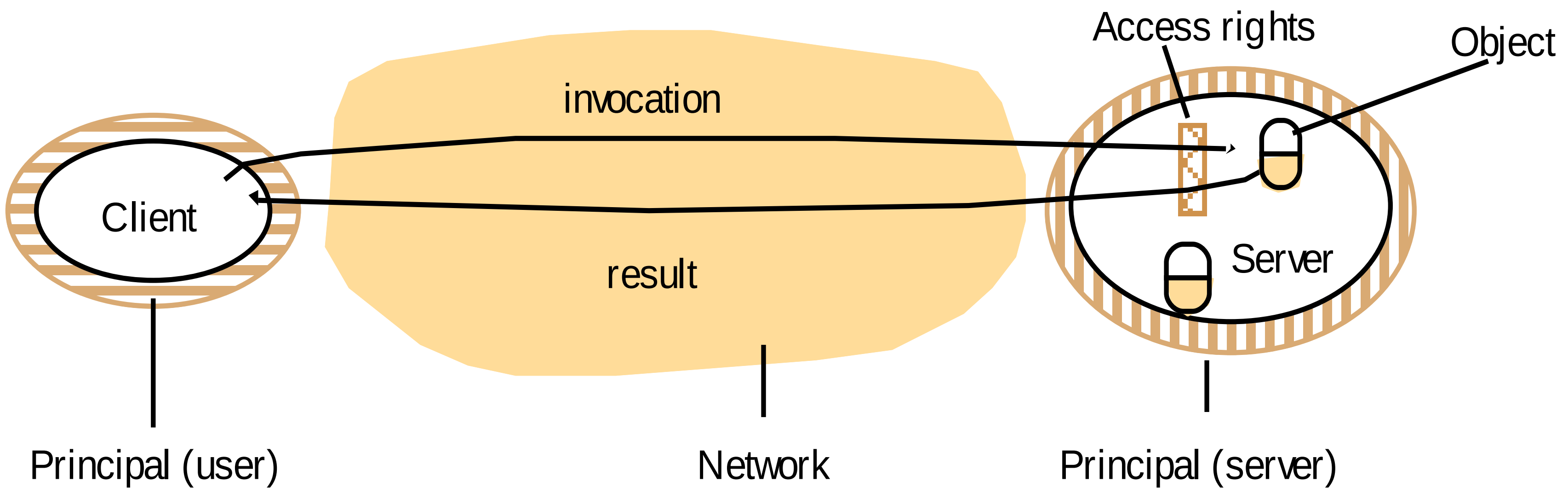
| *Class of Failure* | *Affects* | *Description* |
|---|---|---|
| Clock | Process | Process's local clock exceeds the bounds on its rate of drift from real time. |
| Performance | Process | Process exceeds the bounds on the interval between two steps. |
| Performance | Channel | A message's transmission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

## Failure Model (4)

- masking failures
  - hiding--use another server to respond
  - converting it into more acceptable--drop the packet if it is corrupted
- reliable one-to-one communication
  - validity: eventually delivered
  - integrity: content not corrupted or duplicated

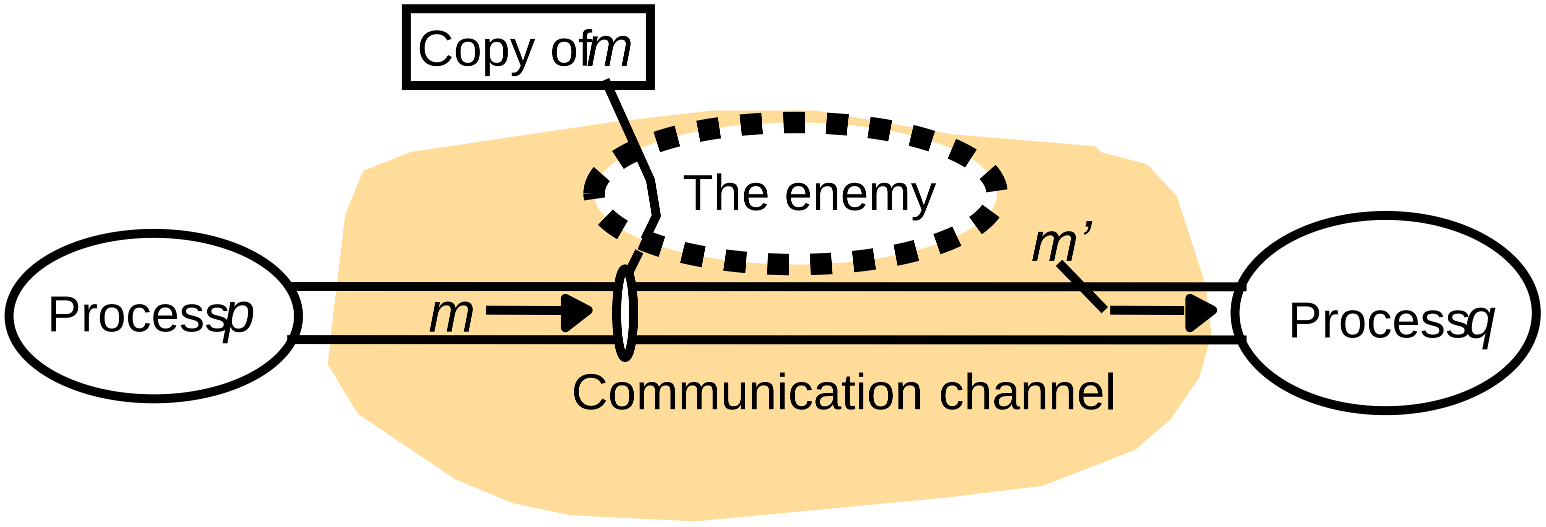
# Security Model (1)

- Protecting objects:
  - authorization (access rights to principals)
  - authentication (identity of parties/principals)



## Security Model (2)

- Threat to processes & communication channels



## Security Model (3)

- Threat to processes & communication channels
- Denial of service
- Mobile code
- Cryptography: science of keeping messages secret
    - encryption: process of scrambling a message to hid its content
    - secret keys--large numbers that are difficult to guess
    - authentication--encrypt the identity, check the decrypted identity
    - secured channels--authentication, privacy/integrity, time stamp to prevent replaying and reordering

## Assignment1 (chapter2)

- Exercise (Page 78)
- 2.11
- 2.14

- Assignment1 will be due one week after we finish Chapter 5.