Introduction to Distributed Computing

Prof. Sanjeev Setia

Distributed Software Systems

CS 707

About this Class

- Distributed systems are ubiquitous
- Focus:
  - Fundamental concepts underlying distributed computing
  - designing and writing moderate-sized distributed applications
- Prerequisites:
  - CS 571 (Operating Systems)
  - CS 656 (Computer Networks)
  - CS 706 (Concurrent Software)

What you will learn

"I hear and I forget, I see and I remember, I do and I understand" – Chinese proverb

- Issues that arise in the development of distributed software
- Middleware technology
  - Threads, sockets
  - RPC, Java RMI/CORBA
  - Javaspaces (JINI), SOAP/Web Services/.NET, Enterprise Javabeans
    - Not discussed in class, but you can become more familiar with these technologies by

Logistics

- Grade: 60% projects, 40% exams
- Slides, assignments, reading material on class web page http://www.cs.gmu.edu/~setia/cs707/
- Two small (2-3 week) programming assignments + one larger project (3-4 weeks)
  - To be done individually
- Use any platform; all the necessary software will be available on IT&E lab computers

Readings

- Textbook:
  - "Distributed Systems: Principles and Paradigms" - Tannenbaum & van Steen
  - Some lectures based on Coulouris et al "Distributed Systems: Concepts & Design"
- Research literature
  - Each lecture/chapter will be supplemented with articles from the research literature
  - Links on class web site

Schedule

- Introduction (today)
- Client-server application design
- Application-level protocols
  - Sockets
- Communication
  - RPC/RMI/CORBA
- Naming
- Synchronization
- Consistency & Replication
- Fault Tolerance

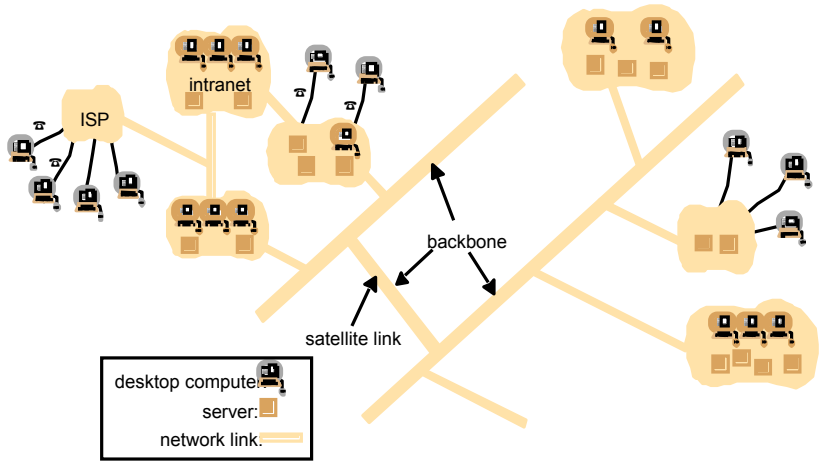
Example Distributed systems

- Internet
- ATM (bank) machines
- Intranets/Workgroups
- Computing landscape will soon consist of ubiquitous network-connected devices
  - "The network is the computer"

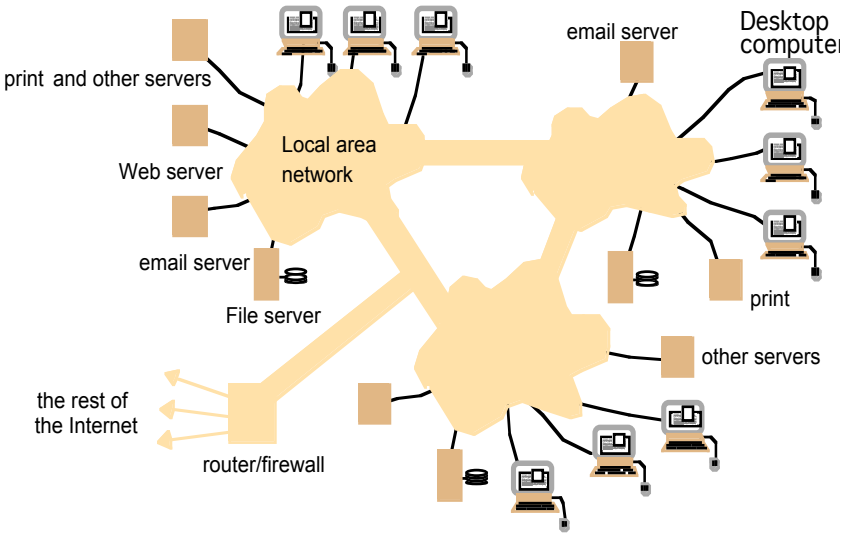
Characteristics of Distributed Systems

- Concurrency
- No global clock
- Independent failures

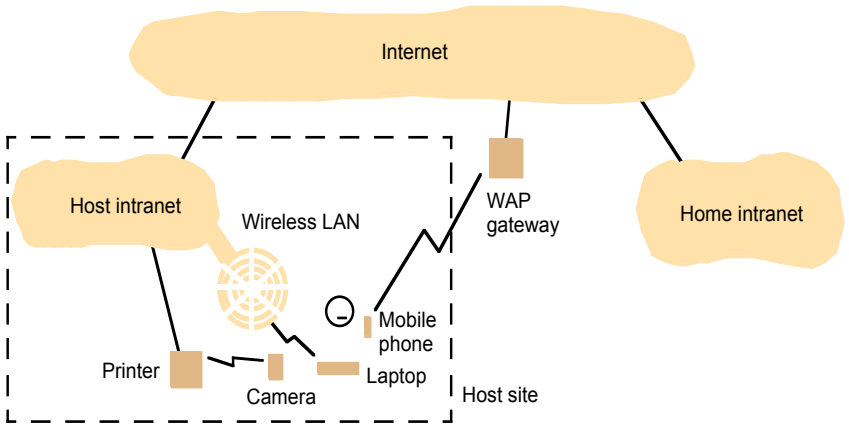
A typical portion of the Internet



A typical intranet



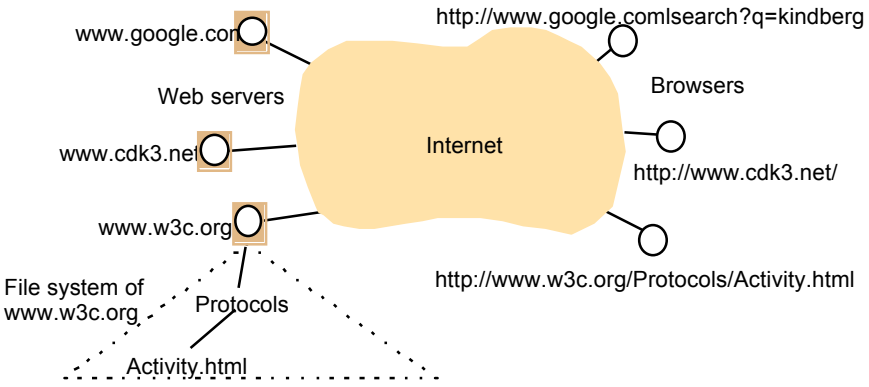
Portable and handheld devices in a distributed system



Distributed applications

- Applications that consist of a set of processes that are distributed across a network of machines and work together as an ensemble to solve a common problem
- In the past, mostly "client-server"
  - Resource management centralized at the server
- "Peer to Peer" computing represents a movement towards more "truly" distributed applications

Web servers and web browsers



Goals/Benefits

- Resource sharing
- Scalability
- Fault tolerance and availability
- Performance
  - Parallel computing can be considered a subset of distributed computing

Challenges(Differences from Local Computing)

- Heterogeneity
- Latency
- Remote Memory vs Local Memory
- Synchronization
  - Concurrent interactions the norm
- Partial failure
  - Applications need to adapt gracefully in the face of partial failure
  - Lamport once defined a distributed system as "One on which I cannot get any work done because some machine I have never heard of has crashed"

Challenges cont'd

- Need for "openness"
  - Open standards: key interfaces in software and communication protocols need to be standardized
- Security
  - Denial of service attacks
  - Mobile code
- Scalability
- Transparency

Scalability

- Becoming increasingly important because of the changing computing landscape
- Key to scalability: decentralized algorithms and data structures
  - No machine has complete information about the state of the system
  - Machines make decisions based on locally available information
  - Failure of one machine does not ruin the algorithm
  - There is no implicit assumption that a global clock exists

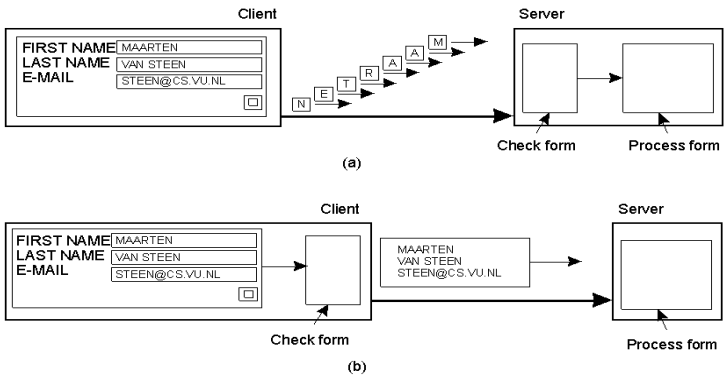
Computers in the Internet

Date	*Computers*	*Web servers*
1979, Dec.	188	0
1989, July	130,000	0
1999, July	56,218,000	5,560,866
2003, Jan.	171,638,297	35,424,956

Computers vs. Web servers in the Internet

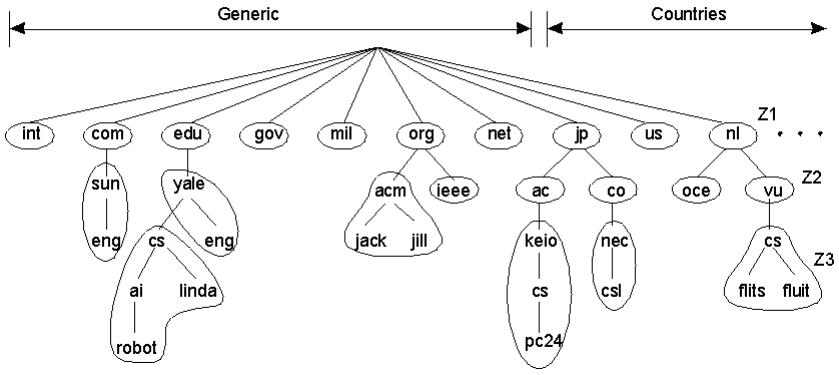
Date	*Computers*	*Web servers*	*Percentage*
1993, July	1,776,000	130	0.008
1995, July	6,642,000	23,500	0.4
1997, July	19,540,000	1,203,096	6
1999, July	56,218,000	6,598,697	12
2001, July	125,888,197	31,299,592	25
2003, July		42,298,371	

Scaling Techniques (1)



The difference between letting: (a) a server or (b)a client check forms as they are being filled

Scaling Techniques (2)



An example of dividing the DNS name space into zones.

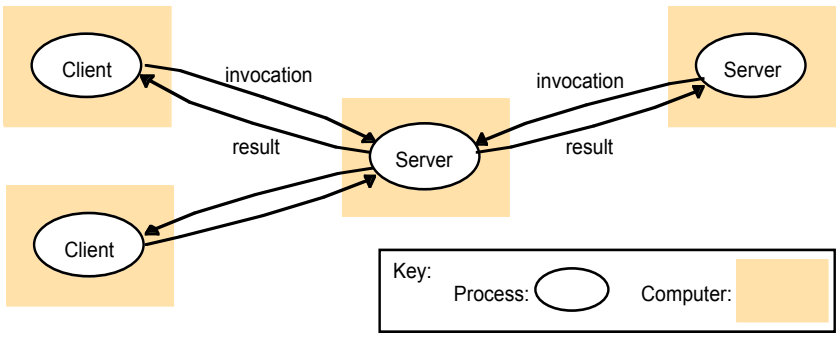
Transparency in Distributed Systems

Access transparency: enables local and remote resources to be accessed using identical operations.
Location transparency: enables resources to be accessed without knowledge of their physical or network location (for example, which building or IP address).
Concurrency transparency: enables several processes to operate concurrently using shared resources without interference between them.
Replication transparency: enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers.
Failure transparency: enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of hardware or software components.
Mobility transparency: allows the movement of resources and clients within a system without affecting the operation of users or programs.
Performance transparency: allows the system to be reconfigured to improve performance as loads vary.
Scaling transparency: allows the system and applications to expand in scale without change to the system structure or the application algorithms.

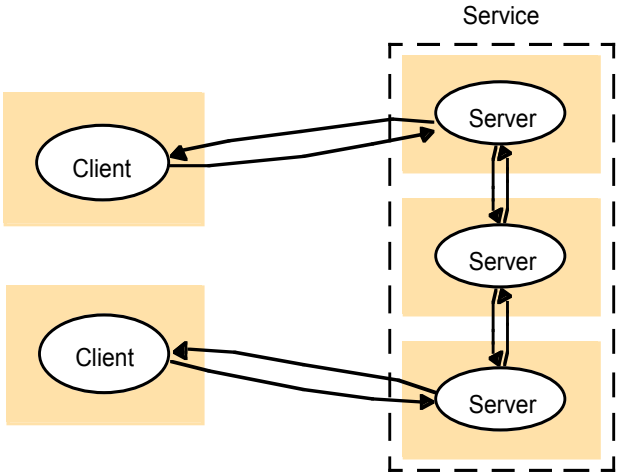
Communication Patterns

- Client-server
- Group-oriented/Peer-to-Peer
  - Applications that require reliability, scalability
- Function-shipping/Mobile Code/Agents
  - Postscript, Java

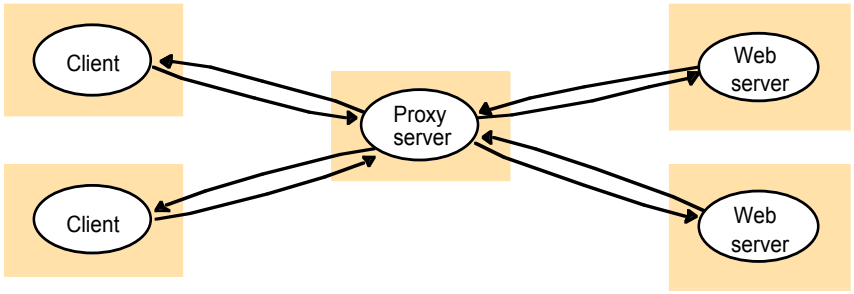
Clients invoke individual servers



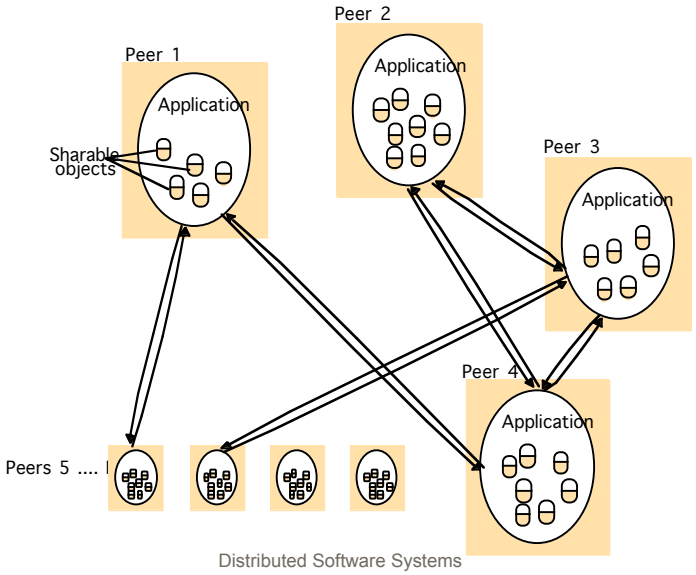
A service provided by multiple servers



Web proxy server

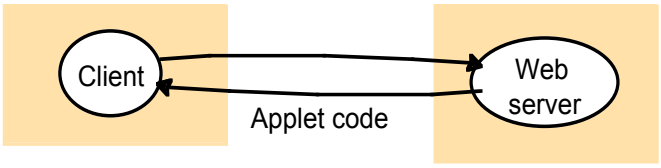


A distributed application based on peer processes



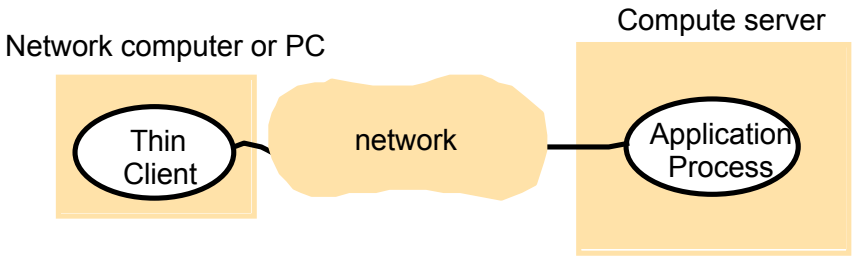
Web applets

a) client request results in the downloading of applet code

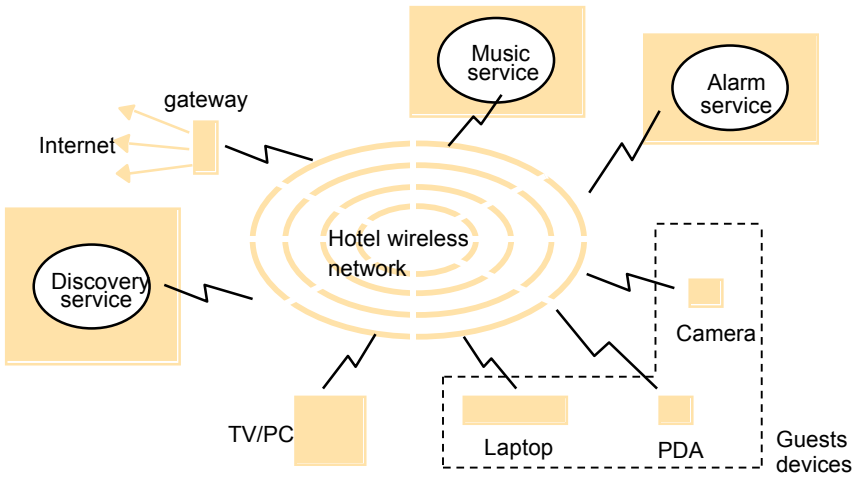


b) client interacts with the applet

Thin clients and compute servers



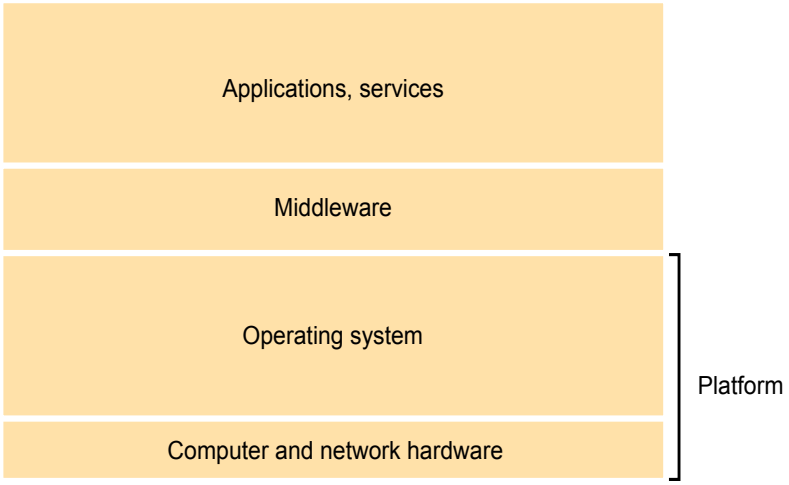
Spontaneous networking in a hotel



Distributed Software: Goals

- Middleware handles heterogeneity
- Higher-level support
  - Make distributed nature of application **transparent** to the user/programmer
    - Remote Procedure Calls
    - RPC + Object orientation = CORBA
- Higher-level support BUT **expose** remote objects, partial failure, etc. to the programmer
  - JINI, Javaspaces
- Scalability

Software and hardware service layers in distributed systems



Fundamental/Abstract Models

- A fundamental model captures the essential ingredients that we need to consider to understand and reason about a system's behavior
- Addresses the following questions
  - What are the main entities in the system?
  - How do they interact?
  - What are the characteristics that affect their collective and individual behavior?

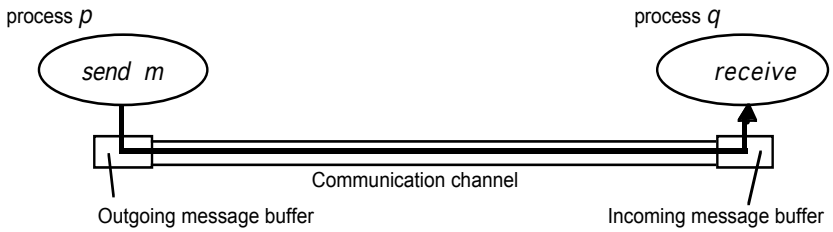
Fundamental/Abstract Models

- Three models
  - Interaction model
    - Reflects the assumptions about the processes and the communication channels in the distributed system
  - Failure model
    - Distinguish between the types of failures of the processes and the communication channels
  - Security Model
    - Assumptions about the principals and the adversary

Interaction Models

- Synchronous Distributed Systems: a system in which the following bounds are defined
  - The time to execute each step of a process has an upper and lower bound
  - Each message transmitted over a channel is received within a known bounded delay
  - Each process has a local clock whose drift rate from real time has a known bound
- Asynchronous distributed system
  - Each step of a process can take an arbitrary time
  - Message delivery time is arbitrary
  - Clock drift rates are arbitrary
- Some implications
  - In a synchronous system, timeouts can be used to detect failures
  - Impossible to detect failures or "reach agreement" in an asynchronous system

Processes and channels



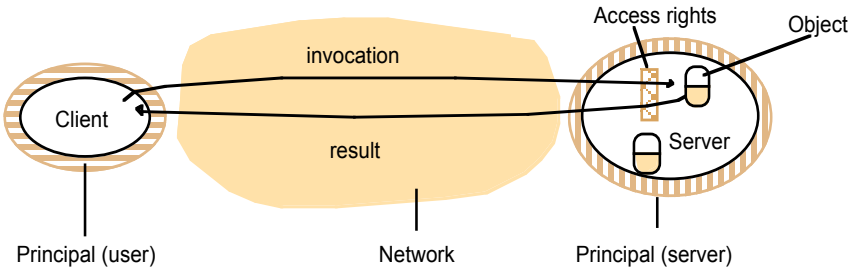
Omission and arbitrary failures

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.

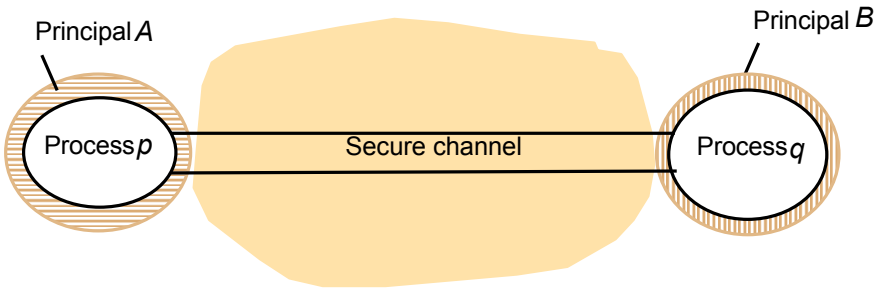
Timing failures

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.

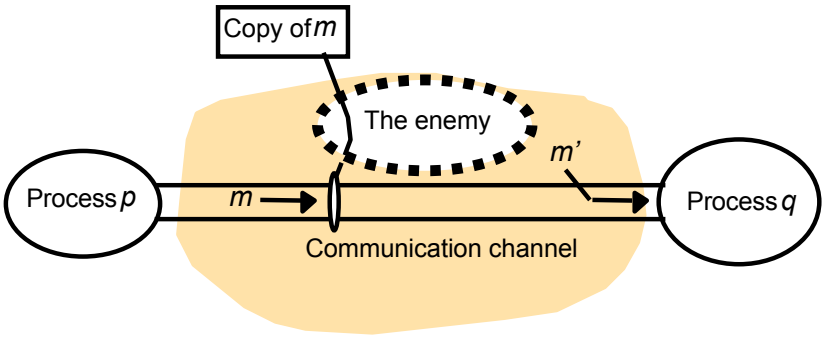
Objects and principals



Secure channels



The enemy/adversary



Readings

- Chapter 1 of textbook (Tannenbaum)
- Chapters 1, 2 of Coulouris, Kindberg, Dollimore (on reserve in library)
- "A Note on Distributed Computing" – Waldo, Wyant, Wollrath, Kendall
  - Link on class web page