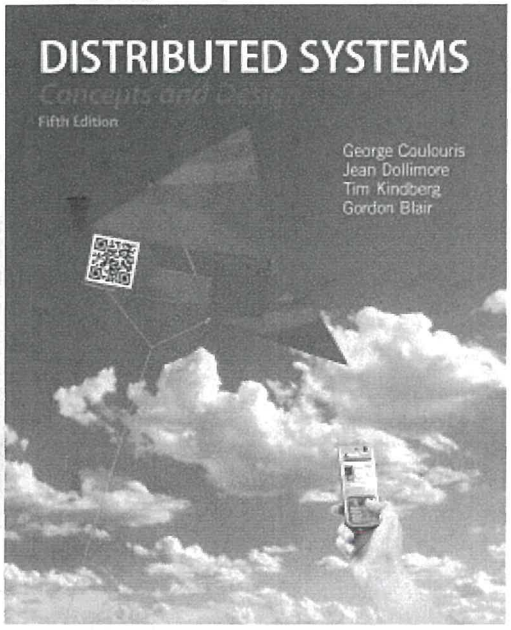
Distributed Systems
COMP90015 2019 SM1
Introduction
Lectures by Aaron Harwood
© University of Melbourne 2019

Textbook

From Coulouris, Dollimore and Kindberg, *Distributed Systems: Concepts and Design*, Edition 5, © Addison-Wesley 2012.



Materials used in all lecture slides with permission from Coulouris.

Search for and download an electronic copy (PDF) of the textbook online.

All content presented in lecture slides is contained in these notes. Additional information (such as this) is also provided in these notes.

The subject content draws strongly from the text book, but you are recommended to read broadly as there are many useful distributed systems textbooks.

Staff

- Aaron Harwood, Lecturer, aharwood@unimelb.edu.au
- Irum Bukhari, Head Tutor, irum.bukhari@unimelb.edu.au
- Gayashan Amarasinghe, Tutor, gayashan.amarasinghe@unimelb.edu.au
- Oyemike Ebinum, Tutor, oyemike.ebinum@unimelb.edu.au

- Haowen Tang, Tutor, haowen.tang@unimelb.edu.au
- Qizhou Wang, Tutor, q.wang39@student.unimelb.edu.au
- Jiayun He, Tutor, jiayun.he@student.unimelb.edu.au

Assessment

- Project 1, 20%, software and written report (potentially multiple parts)
- Project 2, 20%, software and written report
- Final Exam, 60%, closed book, no calculator

All projects will be group work with target group size of 4. The language of choice is Java. Please start considering your project members now. Project details will be provided probably around Week 3 or 4. Computer labs around the university can be used to undertake programming work. We may schedule additional Java programming tutorials if there is enough demand; there is a forum thread for you to indicate your interest.

- Online Quiz Questions, not assessed, via LMS on a somewhat weekly basis

Academic Integrity

All University of Melbourne students are expected to uphold academic integrity in all aspects of every piece of work that they submit for assessment.

- Melbourne School of Engineering Academic Integrity
- Academic Integrity Module, Join the Community
- Academic Integrity Module, Re-visit the Community

Course Overview

- Introduction
- Models
- Interprocess Communication
- Remote Invocation
- Indirect Communication
- Operating System Support
- Security
- File Systems
- Naming

The order of the material may change, as well as the content. For this reason, lecture slides are not available all at the beginning, but rather weekly.

Introduction Overview

- What is a distributed system?
- Why distributed systems?
- Examples of distributed systems
- Challenges

What is a Distributed System?

Distributed System definitions - many and varying:

- A system in which hardware or software components located at networked computers communicate and coordinate their actions only by passing message [Coulouris]
- A collection of independent computers that appears to its users as a single coherent system [Tanenbaum]

Key aspects:

- a number of components
- communication between the components
- synergy: achieve more than the simple sum of individual components

Computer Networks vs Distributed Systems

A Computer Network: Is a collection of spatially separated, interconnected computers that exchange messages based on specific protocols. Computers are addressed by IP addresses.

A Distributed System: Multiple computers on the network working together as a system. The spatial separation of computers and communication aspects are hidden from users.

Why Distributed Systems?

- Resource Sharing:
  - Hardware Resources (Disks, printers, scanners etc)
  - Software Resources (Files, databases etc)
  - Other (Processing power, memory, bandwidth)
- Benefits of resource sharing:
  - Economy
  - Reliability
  - Availability
  - Scalability

Consequences of Distributed Systems

- Concurrency

In a Distributed System computers perform their tasks autonomously and communicate with other computers via message passing when needed. Services provided by distributed systems will be accessed by multiple users simultaneously. Distributed system design should take this into consideration and implement appropriate techniques for handling concurrency.

- No global clock

Clocks on individual computers operate independently. Since computers communicate by message passing there are limits to the accuracy with which the computers in a network can synchronize their clocks. Therefore, distributed systems have to deal with the fact of not having a global clock and implement time synchronization techniques when needed.

- Independent failures

Some components of the system may fail while the others are still running. Failures of participating computers of the system is not known to others immediately.

Computer Networks

Common distributed systems are based on widely used computer networks, e.g.:

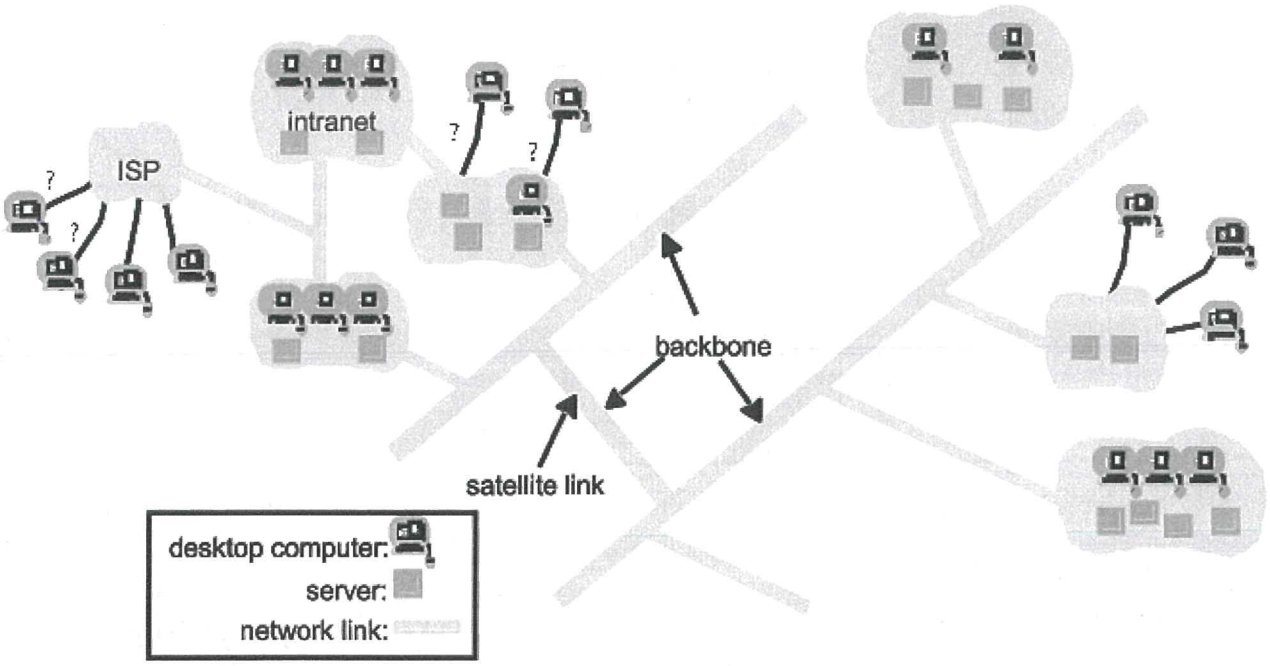
- The Internet
- Intranets
- Wireless networks

The Internet

Consists of a large number of interconnected collection of computer networks of different types. Features include:

- Computers interacting by message passing using a common means of communication (Internet protocol)
- Many different services (applications) (World Wide Web (www), email, file transfer)
- A number of Intranets linked by backbones
- Internet Service Providers (ISPs), that provide access to the services on the Internet while providing local-service such as email and web hosting
- A backbone network link with high transmission capacity
- Communication via Satellite, fiber optic cables and other high-bandwidth circuits

A typical portion of the Internet



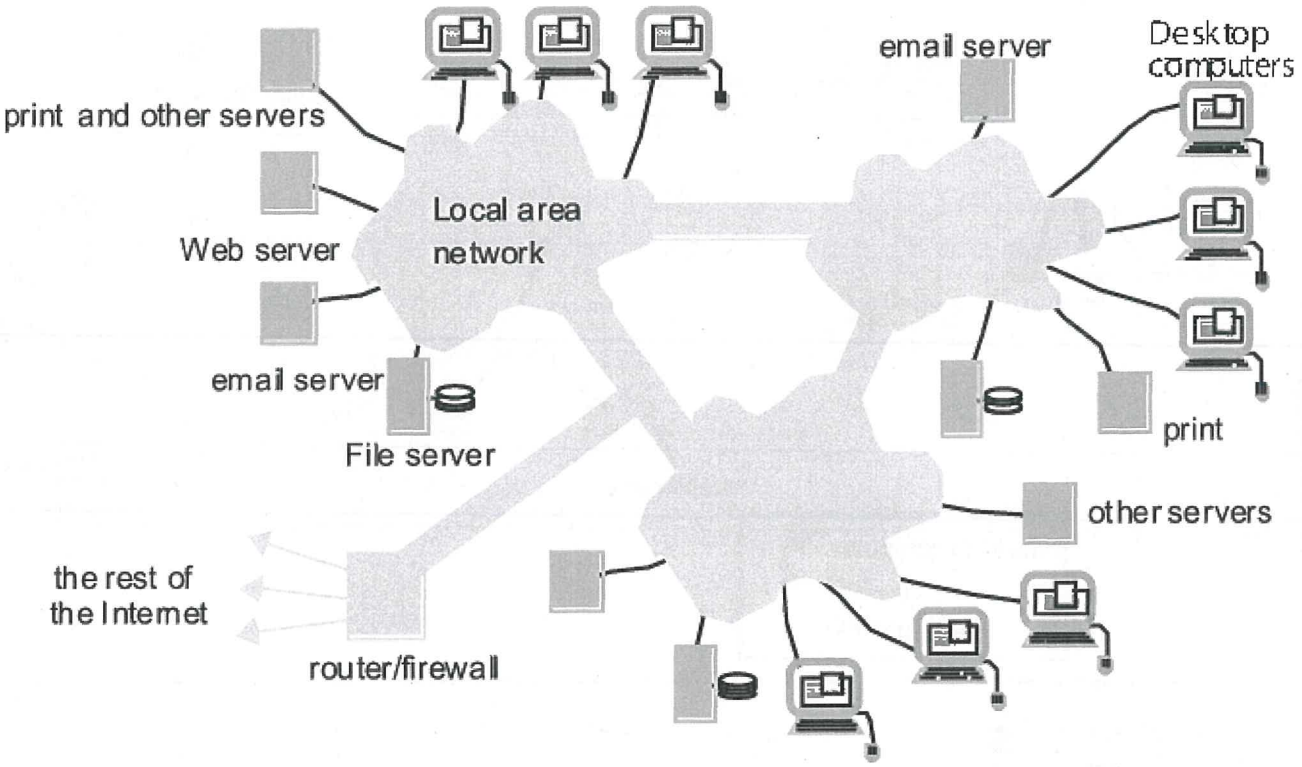
Intranets

Is a portion of the Internet that is separately administered by organizations. Features include:

- A boundary that can be configured to enforce local security policies
- Several local area connection (LANs) linked by backbone connections
- A connection to the Internet via a router allowing users within the intranet to access services on the Internet
- Firewalls to protect an intranet by preventing unauthorized messages leaving or entering by filtering incoming and outgoing messages e.g. by source or destination

A typical intranet

preventing unauthorized message.



Wireless Networks

Wireless networking allows the integration of small and portable computing devices (e.g. laptop computers), hand-held devices (PDAs and mobile phones, video and digital cameras), wearable devices, device embedded applications (washing machines, refrigerators) into distributed systems.

Three popular (or not so popular) paradigms that may make heavy use of wireless networks are:

1. Mobile Computing (Nomadic Computing)

The mobile computing paradigm allows users to perform computing tasks on the move, while being a part of a distributed system. User is normally connected to other resources via a wireless network.

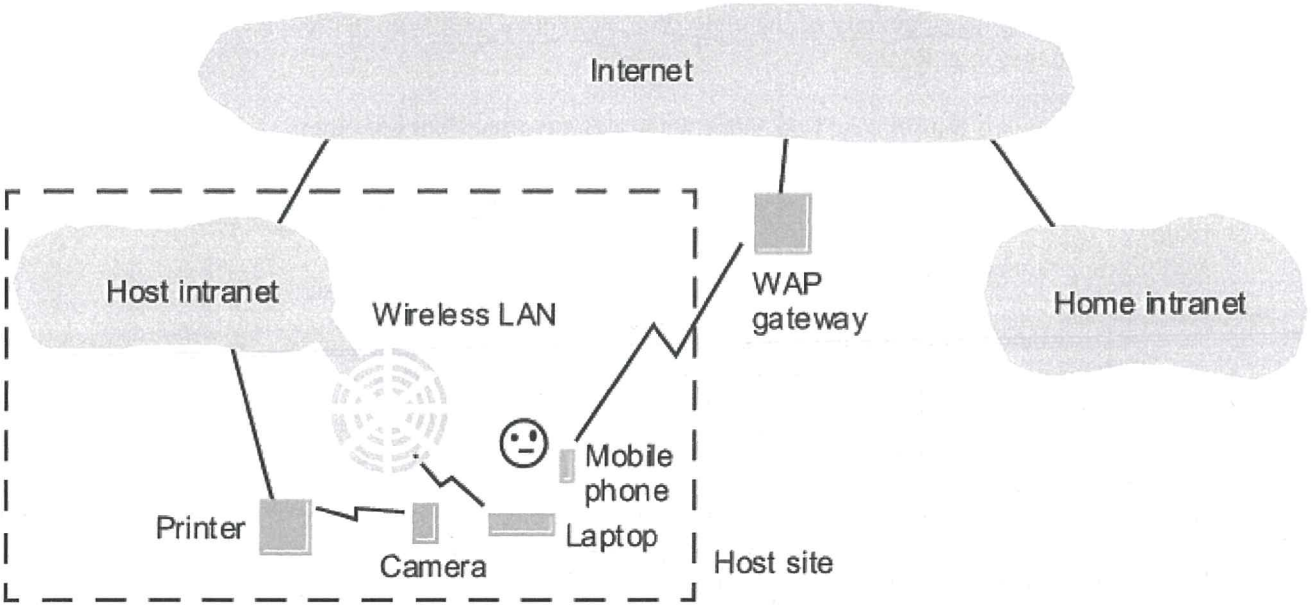
2. Ubiquitous Computing

Refers to the paradigm where small, cheap, computing devices that are embedded in devices are used as a part of a part of a distributed system, using wireless networks.

3. Internet of Things

Refers to the paradigm where everyday objects are addressable and connected to the Internet.

Portable and hand-held devices in a distributed systems



Distributed system challenges

- Heterogeneity
- Openness
- Security
- Scalability
- Failure Handling
- Concurrency
- Transparency

Heterogeneity

Distributed systems use hardware and software resources of varying characteristics (heterogeneous resources):

- Networks
- Computer Hardware
- Operating Systems
- Programming Languages
- Implementation by different developers

Some approaches for handling heterogeneity issues are:

- Using standard protocols
- Using agreed upon message formats and data types
- Adhering to an agreed upon Application Program Interfaces (APIs)
- Using Middleware
- Portable code

Middleware is a software layer between the distributed application and the operating systems that:

- provides a programming abstraction and

- masks the heterogeneity of the underlying networks, hardware, operating systems and programming languages, e.g. RMI.

Certain middleware solutions address some heterogeneity issues but not others.

Example - Java Remote Method Invocation (RMI) does not address programming language heterogeneity issue
Middleware Models:

- Distributed File Systems
- Remote Procedure Call (RPC) (procedural languages)
- Remote Method Invocation (RMI) (object-oriented languages)
- Distributed Documents
- Distributed Data Bases

Heterogeneity and mobile code

Mobile code is sent from one computer to the anther to run at the destination (e.g. Java applets):

- Code that is compiled to run in one OS does not run in another:
  - different hardware
  - different OS versions
- Virtual Machine approach provides a way of making code executable on any hardware - compiler produces code that is interpreted by the virtual machine
- Cross-platform compilation and code portability is another way, that compiles source code to multiple targets.

Openness

Openness refers to the ability of extend the system in different ways by adding hardware or software resources. Following are some approaches to address openness:

- Publishing key interfaces
- Allowing a uniform communication mechanism to communicate over the published interfaces
- Ensuring all implementations adhere to the published standards

---

Security

- There has three aspects of security:
  - Confidentiality (protection against disclosure to unauthorized individuals)
  - Integrity (protection against alteration and corruption)
  - Availability (protection against interference with the means of access)
- Security Mechanisms:
  - Encryption (e.g. Blowfish, RSA)
  - Authentication (e.g. passwords, public key authentication)
  - Authorization (e.g. access control lists)

Security

Types of security challenges have not yet been resolved completely:

- Denial of service attacks
- Security against mobile code (executables as attachments)

Scalability

A system is considered to be scalable if it can handle the growth of the number of users. Scalability Challenges:

- Cost of physical resources: For a system with n users to be scalable, the quantity of physical resources required to support the system should be O(n) - if one file server can support 20 users, then two servers should be able to support 40 users.
- Controlling the performance loss: Complexity of algorithms that are used for searches, lookup etc should be scalable - for example a search algorithm for n data items requiring O(log(n)) search steps is scalable but one that requires n^2 is not.
- Resources should not run out: e.g. 32-bit Internet IP addresses running out requiring a new version of 128-bit address.
- Avoiding Performance bottlenecks: Decentralized architectures and algorithms can be used to avoid performance bottlenecks.

Failure Handling

Following are approaches to handing failures:

- **Detecting:** some types of failures can be detected, e.g. checksums can be used to detect corrupted data in a message, and some kinds of failure are hard to be certain about, e.g. the failure of a remote server.
- **Masking:** some failures that have been detected can be hidden or made less severe, e.g. timeout and message retransmission.
- **Tolerating:** it is sometimes impractical to try and handle every failure that occurs, sometimes it is better to tolerate them, e.g. failure is reported back to the user, e.g. web server not being available, try again later, or video is rendered with errors
- **Recovery:** failure sometimes leads to corrupted data and software can be designed so that it can recover the original state after failure, e.g. implementing a roll back mechanism.
- **Redundancy:** services can be made to tolerate failures using redundant components, e.g. multiple servers that provide the same service, as so called fail over.

Concurrency

- Multiple clients can access the same resource at the same time, in some cases for updates
- One approach to handling concurrency is making access sequential - slows down the system
- Semaphores supported by the operating system is a well accepted mechanism to handle concurrency

Transparency

This is the aspect of hiding the components of a distributed system from the user and the application programmer. Types of transparencies:

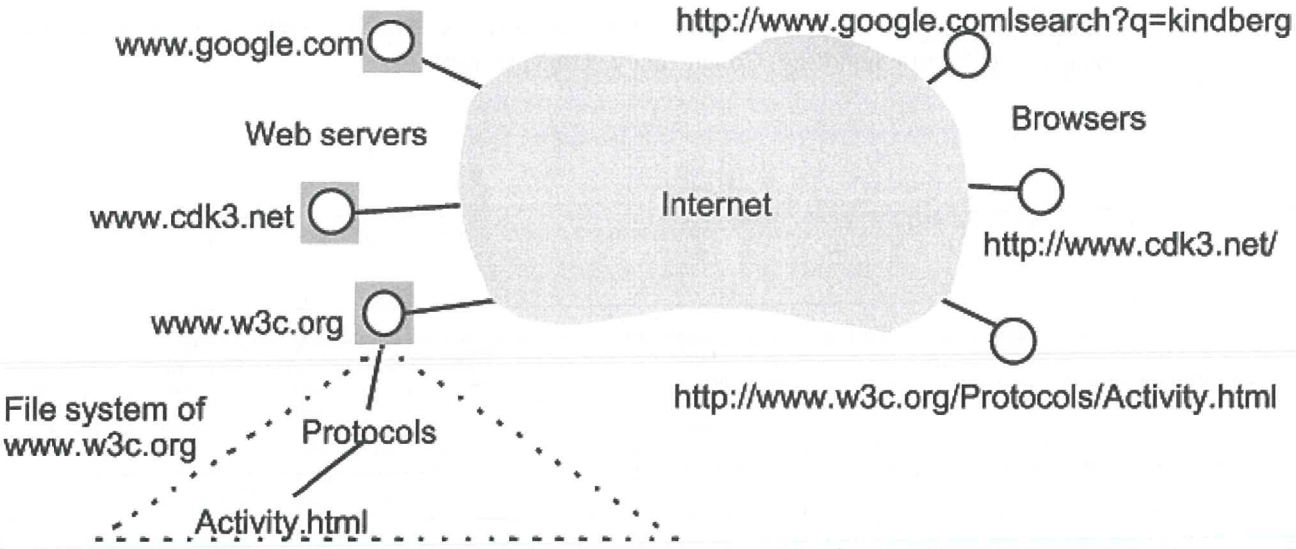
- Access transparency
- Location transparency
- Concurrency transparency
- Replication transparency
- Failure transparency
- Mobility transparency
- Performance transparency
- Scaling transparency

The World Wide Web (example)

Began as a simple system for exchanging documents at the European centre for nuclear research (CERN), Switzerland, in 1989. Documents are logically organized using a *hypertext* structure and are connected through *links*. The three main standard technological components used in the web are:

- HypterText Markup Language (HTML) - This is a language for specifying the contents and layout of pages to be displayed by browsers.
- Uniform Resource Locators (URLs) - These identify the resources stored on the web. A URL has two components: *scheme* and *scheme-specific-identifier*. Scheme - declares the type of resource. Scheme-specific-identifier - identifies the resource. E.g., *mailto:xxx@yahoo.com*, *ftp://ftp.downloadIt.com/aProg.exe*
- HyperText Transfer Protocol (HTTP) - This is the protocol used for transferring resources between web servers and clients (e.g. browsers). The main features are:
  - Uses request-reply interactions
  - Supports different content types
  - Requires one request per resource
  - Supports simple access control

Web servers and web browsers



Dynamic pages

- Static web pages allow data to be made available for retrieval
- Dynamic web pages allow users to interact with resources by taking user input, executing programs and returning results
- Programs executed on the request can take different forms:
  - A Common Gateway Program (CGI)
  - Servlet
  - Downloaded code - Javascript, applet

Web Services & APIs

- Web services paradigm allows programs (not just browsers) to be clients for web programs
- Data between the server and the client is passed using Extensible Markup Language (XML).
- SOAP protocol allows client to invoke web services.

Growth of APIs available to provide various services:

- Google (searching and maps)
- Facebook (social networking)
- Twitter (information)
- Amazon (marketplace)
- YouTube (video sharing)
- Flickr (photo sharing)
- Dropbox (file/document sharing)

Summary

- Distributed systems are common
- Communication networks that enable distributed systems are the Internet, intranets and wireless networks
- Resource sharing is the main motivation for distributed systems

- There are many challenges associated with these systems - Heterogeneity, Openness, Security, Scalability, Failure handling, Concurrency, Transparency

Distributed Systems
COMP90015 2019 SM1
Models
Lectures by Aaron Harwood
© University of Melbourne 2019

Models Overview

- Introduction
- Architectural models
- Fundamental models
- Summary

Introduction

A *pyhsical model* considers: - underlying hardware elements

An *architectural model* considers:

- *Architectural elements* - components of the system that interact with one another
- *Architectural patterns* - the way components are mapped to the underlying system
- *Associated middleware solutions* - existing solutions to common problems

Several well accepted architectural models are available for distributed systems (e.g client-server, peer-to-peer)

Fundamental models define:

- The non-functional aspects of the distributed system such as
  - reliability
  - security
  - performance

Communicating Entities

From a system perspective:

- processes
- threads
- nodes, e.g. sensors

From a programming perspective:

- Objects, distributed object system
- Components, like objects but with dependencies made more explicit

- Objects and components are usually used within an organization, while web services are usually seen as providing public interfaces

Interfaces

Distributed processes can not directly access each others internal variables or procedures. Passing parameters needs to be reconsidered, in particular, call by reference is not supported as address spaces are not the same between distributed processes. This leads to the notion of an *interface*. The set of functions that can be invoked by external processes is specified by one or more *interface definitions*. Defining interfaces is an important part of designing objects, components and web services.

- Programmers are only concerned with the abstraction offered by the interface, they are not aware of the implementation details.
- Programmers also need not know the programming language or underlying platform used to implement the service.
- So long as the interface does not change (or that changes are backwards compatible), the service implementation can change transparently.

Communication paradigms

From low level to high level:

- Interprocess communication are the underlying primitives -- relatively low-level (perhaps the lowest level) of support for communication between processes in a distributed system, e.g. shared memory, sockets, multicast communication
- Remote invocation -- based on a two-way exchange between communicating entities in a distributed system and resulting in the calling of a remote operation, procedure or method, e.g. request-reply protocols, remote procedure calls, remote method invocation
- Indirect communication
  - *space uncoupling* -- senders do not need to know who they are sending to
  - *time uncoupling* -- senders and receivers do not need to exist at the same time
  - for example: group communication, publish-subscribe systems, message queues, tuple spaces, distributed shared memory

Roles and responsibilities

- Client, a process that initiates connections to some other process
- Server, a process that can receive connections from some other process
- Peer, can be seen as taking both the role of client and server, connecting to and receiving connections from other peers

E.g. we say that a process is acting as a client to another process which is acting as a server, to mean that the first process will be the one responsible for initiating the connection.

Placement

- mapping services to multiple servers

- a single service may not make use of a single process and multiple processes may be distributed across multiple machines
- caching
  - storing data at places that are typically closer to the client or whereby subsequent accesses to the same data will take less time
- mobile code
  - transferring the code to the location that is most efficient, e.g. running a complex query on the same machine that stores the data, rather than pulling all data to the machine that initiated the query
- mobile agents
  - code and data together
  - e.g. used to install and maintain software on a users computer, the agent continues to check for updates in the background

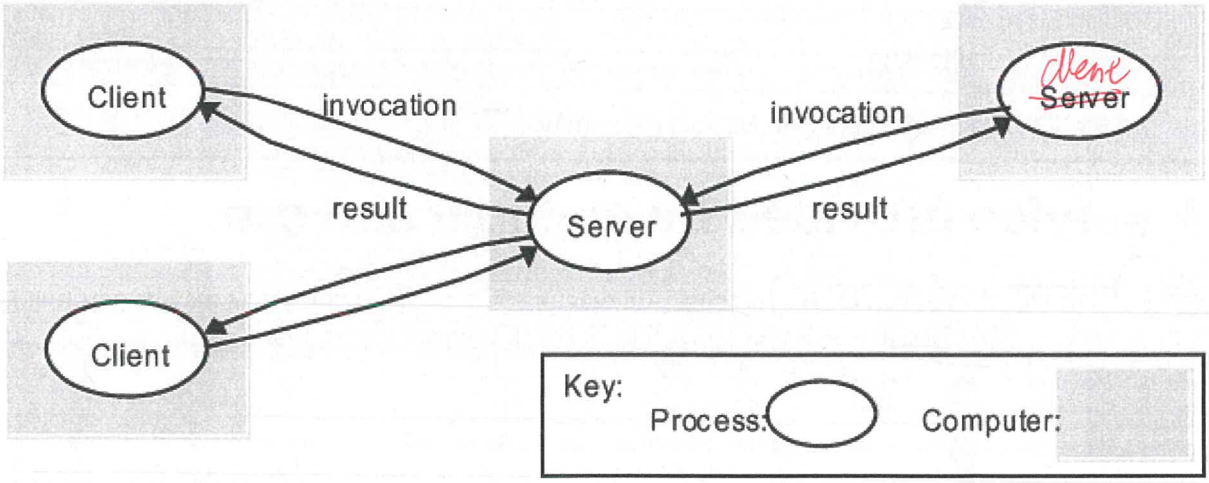
Architectural Patterns

The two widely used distributed architectures are:

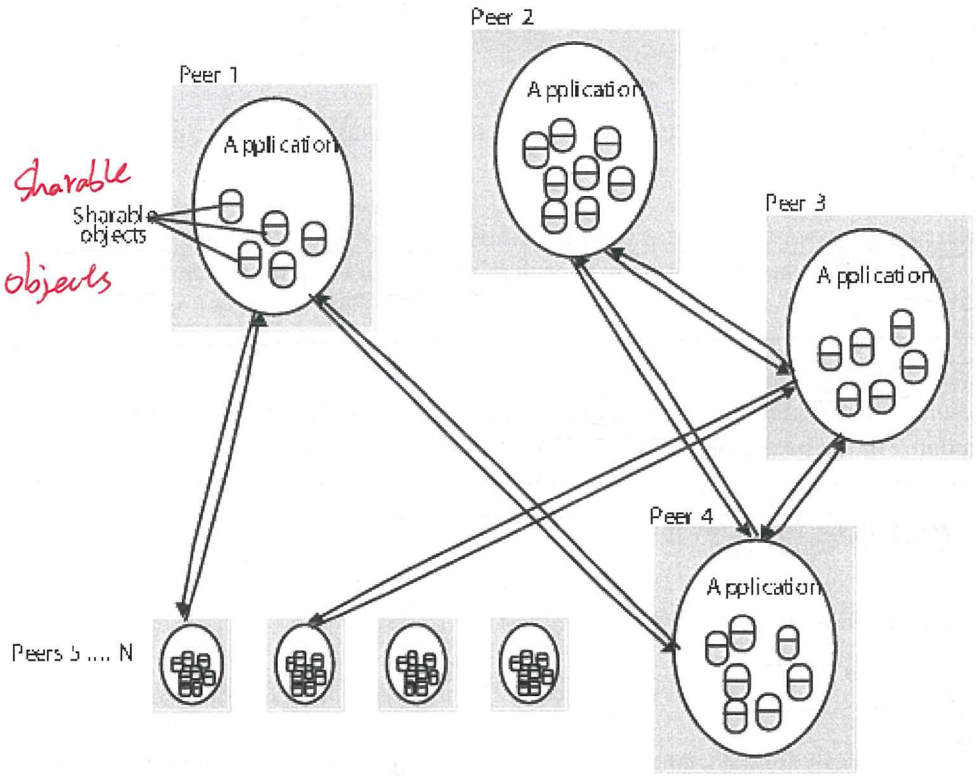
- **Client-server**: Clients invoke services in servers and results are returned. Servers in turn can become clients to other services.
- **Peer-to-peer**: Each process in the systems plays a similar role interacting cooperatively as peers (playing the roles of client and server simultaneously).

Many variations of these architectures are present in today's distributed systems.

Client-server



Peer-to-Peer



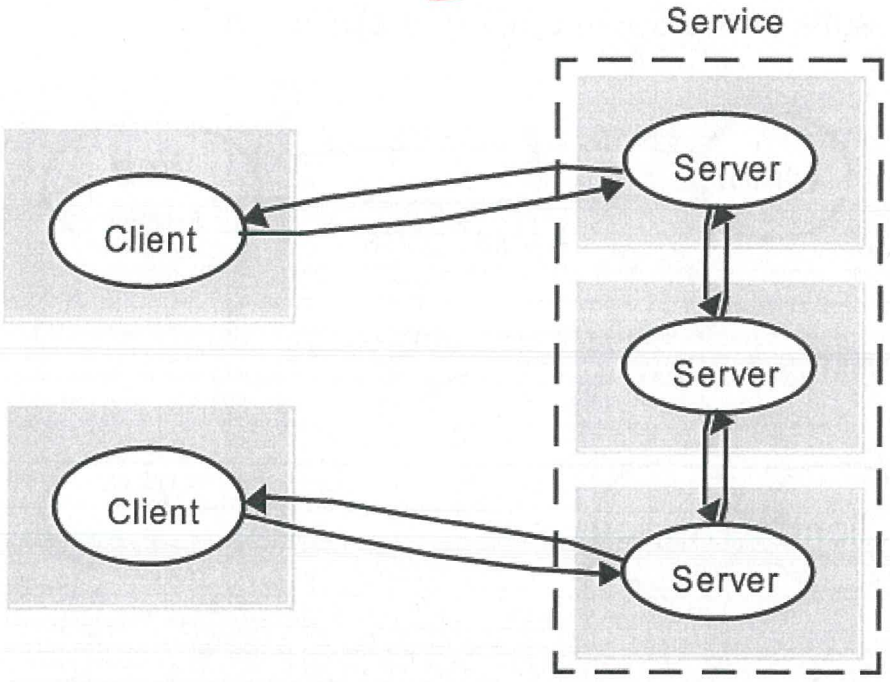
Distributed System Architecture Variations

- Services provided by multiple servers
- Proxy servers and caches
- Mobile code
- Mobile agents
- Network computers
- Thin clients
- Mobile devices and spontaneous inter-operation

A service provided by multiple servers

Service is provided by several server processes interacting with each other. Objects may be partitioned (e.g web servers) or replicated across servers (e.g. Sun Network Information Service (NIS)).

mapping services to multiple server.

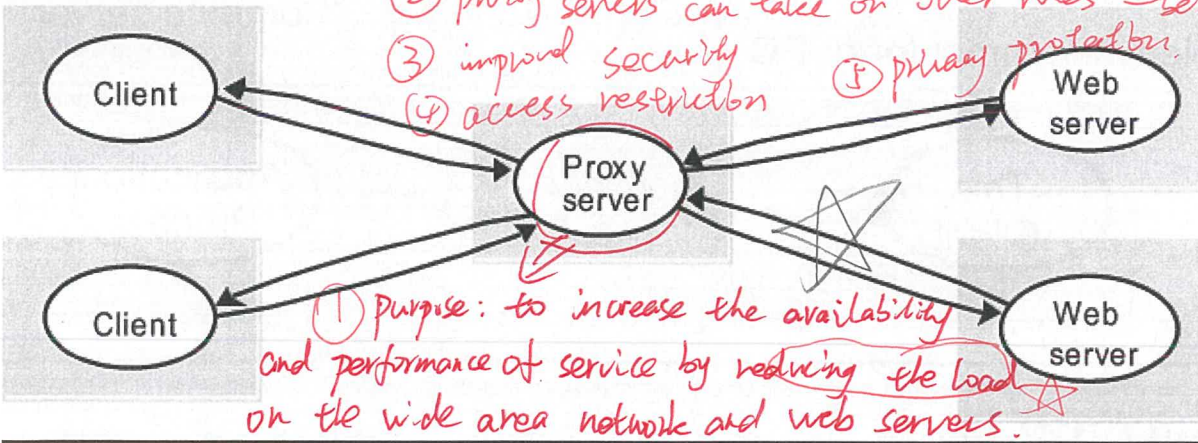


Proxy servers and caches

- Cache is a store of recently used objects that is closer to client
- New objects are added to the cache replacing existing objects
- When an object is requested, the caching service is checked to see if an up-to-date copy is available (fetched in not available)

first.

② proxy servers can take on other roles — better reliability
③ improved security
④ access restriction
⑤ privacy protection



① purpose: to increase the availability and performance of service by reducing the load on the wide area network and web servers.

Mobile Code and Agents

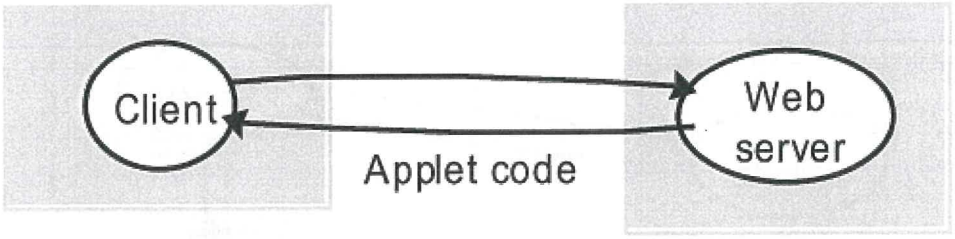
⇒ They may be used to access remote web servers through a firewall

Mobile Code is down loaded to the client and is executed on the client (e.g. applet).

Mobile agents are running programs that includes both code and data that travels from one computer to another.

E.g. Web Applets:

a) client request results in the downloading of applet code

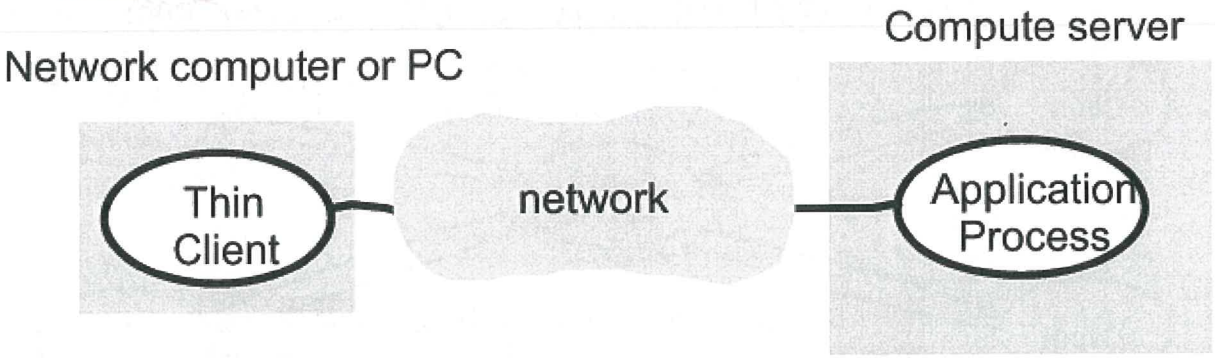


b) client interacts with the applet

Network Computers and Thin clients

- **Network Computers**: download their operating system and application software from a remote file system. Applications are run locally.
- **Thin Clients**: application software is not downloaded but runs on the computer server - e.g. UNIX.

This paradigm is usually not suitable for highly interactive graphical activities.



Layering

A software architecture abstracts software into layers or modules in a single computer. Each layer of software provides a service to the next layer. The layers can are referred to as **service layers**.

Two important layers for a distributed system are:

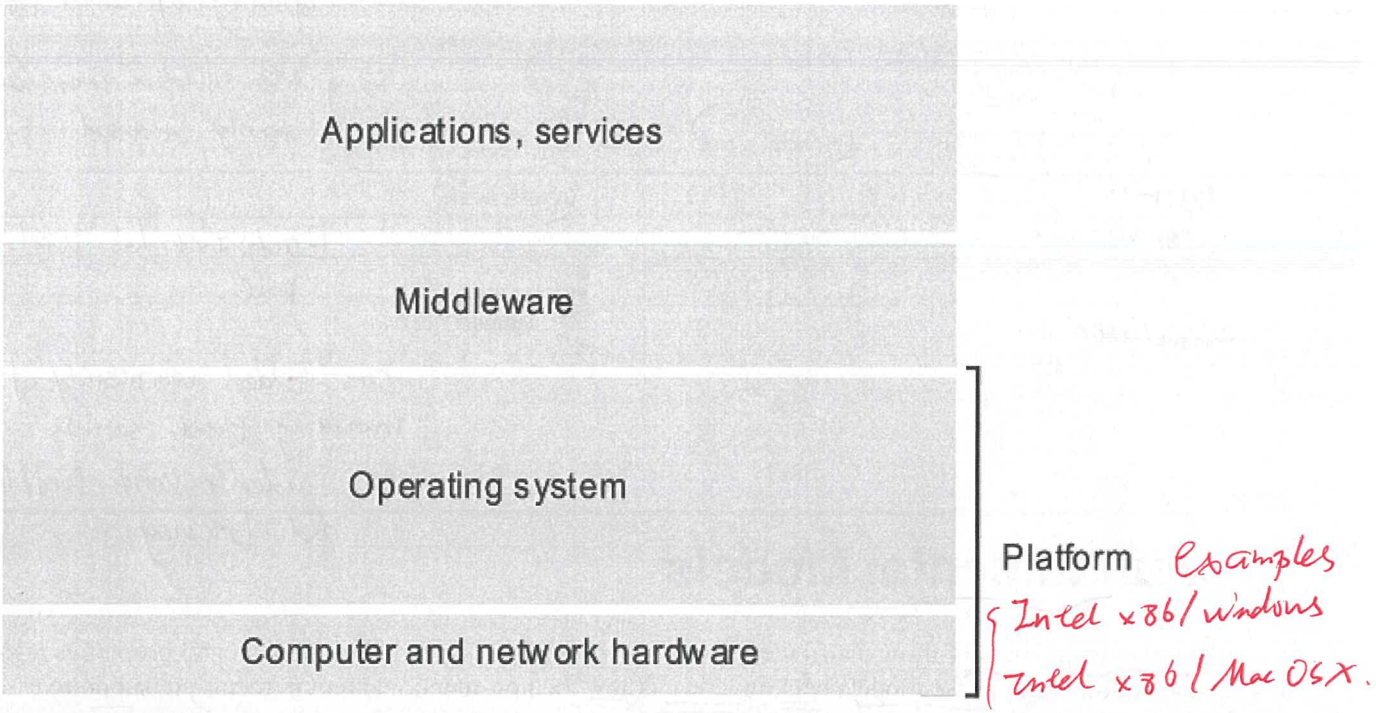
- **Platform**

This are the lowest-level hardware and software layers which include the computer and network hardware and the operating system. The platform provide services to higher levels through system programming interfaces.

- **Middleware**

Middleware masks the network heterogeneity from application developers by providing software components that allow interaction between components in a distributed system.

Abstract software layers



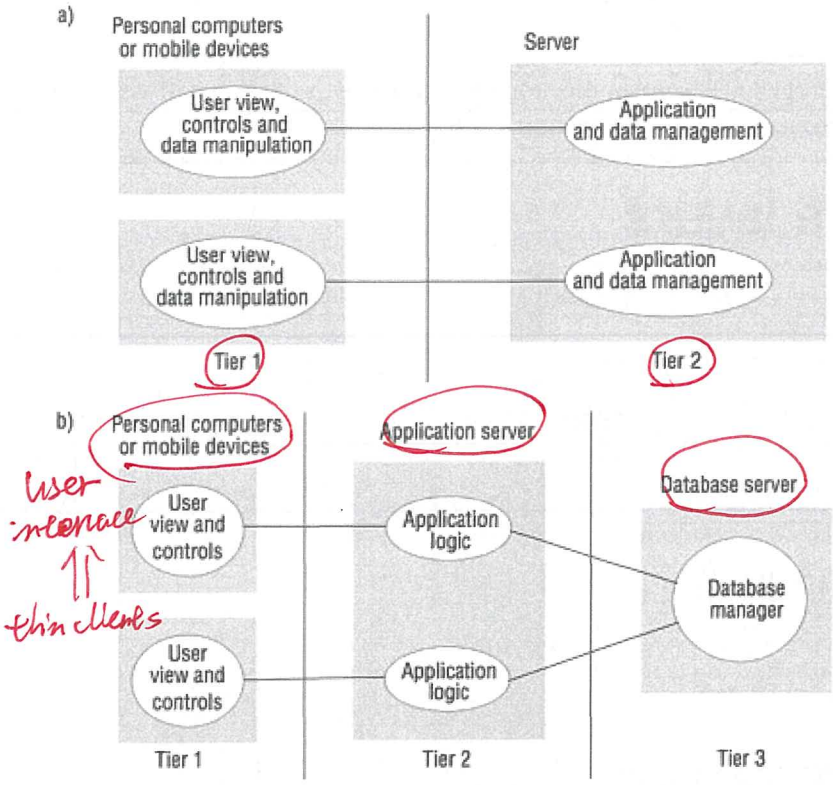
Middleware

- Provides value added services - e.g.
  - Naming
  - security
  - transactions
  - persistent storage and
  - event service
- Adds overhead due to the additional level of abstraction
- Communication cannot be completely hidden from applications since appropriate error handling is important

Tiered architecture

Tiered architectures are complementary to layering. Layering deals with vertical organization of services.

presentation logic : 用户交互 user interaction
application logic : business logic.
data logic : persistent storage of app.



(1) 2 two processes: client and server
plus: low latency.
cons: splitting application logic.
⇒ parts of logic cannot be directly invoked.

user interface
↑
thin clients

Each tier has well-defined role.

cons: (1) added complexity of managing three servers.
(2) added network traffic and latency.

Fundamental Models

Fundamental models allow distributed systems to be analyzed in terms of fundamental properties regardless of the architecture. These models help understand how the non-functional requirements are supported.

The aspects of distributed systems that will be captured in the fundamental models are:

- Interaction — communication delay.

In a distributed system processes interact with each other by passing messages. Message passing results in communication delays, which impact the accuracy with which the processes can be coordinated. Communication delays have to be accounted for when designing distributed systems. *Interaction Model* facilitates this task.

- Failure

Faults in processes or communication channels can impact the behaviour of distributed systems. *Failure Model* helps understand the different types of faults possible in a distributed system.

- Security

Openness of distributed systems make them prone to various types of attacks. *Security Model* defines and classifies the forms of attacks possible in a distributed system.

Interaction Model

- Models the interaction between processes of a distributed system - e.g. interaction between clients and servers or peers.
- Distributed algorithms specify:
  - Steps taken by each process in the distributed system
  - The transmission of messages between processes
- Two important aspects of interaction modeling are:
  - Performance of communication channels
  - Computer clocks and timing events

Performance of communication channels

Three important performance characteristics of communication channels:

- **Latency**

This is the delay between the start of a message transmission from one process to the start of its receipt by the receiving process. Includes network access delay, network transmission delay, OS delays both at the sending and receiving processes.

- **Bandwidth**

This is the total amount of information that can be transmitted over a given time.

- **Jitter**

This is the variation in the time taken to deliver a series of messages.

Event timing

Each computer in a distributed system has its own internal clock. The timestamps between two processes can vary due to:

- Initial time setting being different
- Differences in clock drift rates

GPS can be used synchronize clocks (accuracy of 1 micro second) but they do not operate inside buildings and are too costly therefore special clock synchronization techniques have to be used.

Variations of Interaction Models

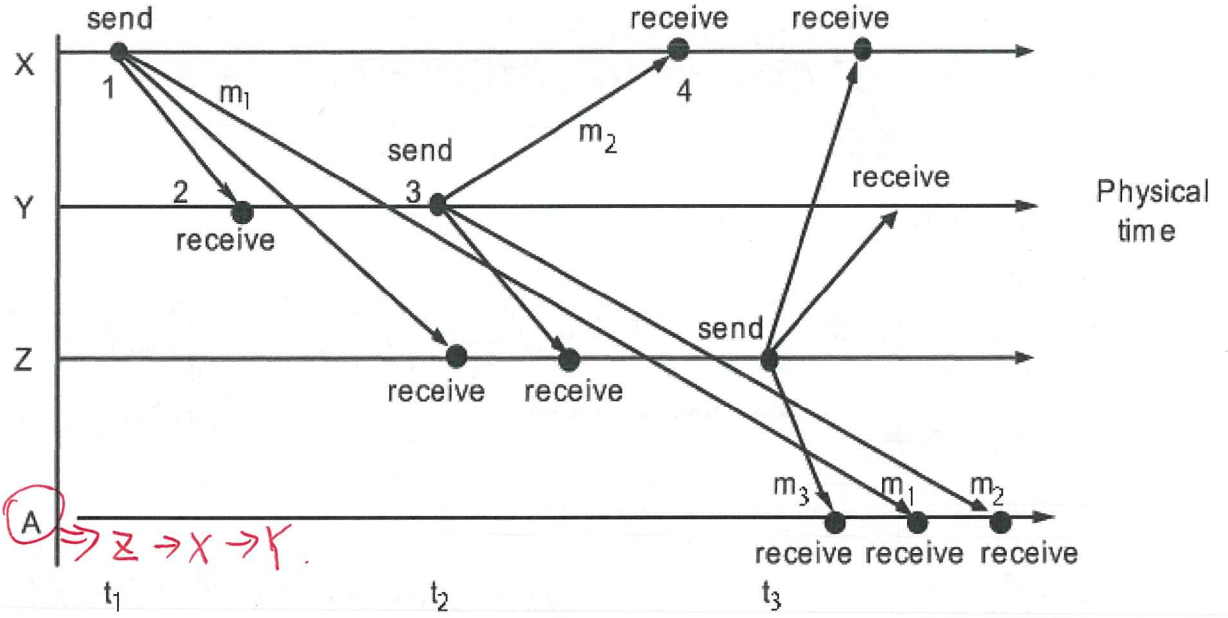
Two simple models of distributed system interaction are:

- Synchronous system model - assumes known bounds on:
  - the time to execute each step of a process
  - message transmission delay
  - local clock drift rate

- Asynchronous system model - assumes no bound on:
  - process execution speed
  - message transmission delays
  - clock drift rates

Event Ordering

Event ordering is important in some scenarios. However, this is challenging without a global clock for all processes. Shows the email example where the messages are not received in the logical order.



The concept of *logical time* can be used for ordering events. For example an event number can be generated for a sequence of events, and the events can be interpreted based on this sequence number as opposed to the time of receipt.

Failure Model

The failures in processes and channels are presented using the following taxonomy:

- Omission failures
- Arbitrary failures
- Timing failures

Omissions Failures

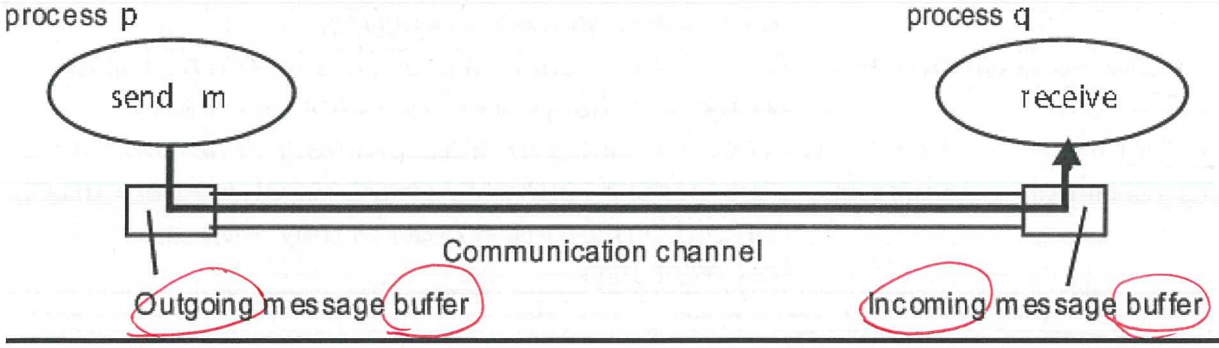
Omission failures refers to cases where a process or a communication channel fails to perform what is expected to do.

- Process omission failures:
  - Normally caused by a process crash
  - Repeated failures during invocation is an indication

- Timeouts can be used to detect this type of crash
- A crash is referred to as a *fail-stop* if other processes can detect certainly that the process crashed

For communication omission failures consider communication as involving the following steps:

- Process **p** inserting the message **m** to its outgoing buffer (*send*)
- The communication channel transporting **m** from **p**'s outgoing buffer to **q**'s incoming buffer
- Process **q** taking the message from its incoming buffer (*receive*)



Communication omission failure could be due to:

- *Send omission failure*: A message not being transported from sending process to its outgoing buffer
- *Receive omission failure*: A message not being transported from the receiving process's incoming message buffer and the receiving process
- *Channel omission failures*: A message not being transported from **p**'s outgoing message buffer to **q**'s incoming message buffer

Arbitrary Failures (Byzantine failure)

Refers to any type of failure that can occur in a system. Could be due to:

- Intended steps omitted in processing
- Message contents corrupted
- Non-existent messages delivered
- Real messages delivered more than once

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 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

These failures occur when time limits set on process execution time, message delivery time and clock rate drift. They are particularly relevant to synchronous systems and less relevant to asynchronous systems since the later usually places no or less strict bounds on timing.

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.

Reliability of one-to-one communication

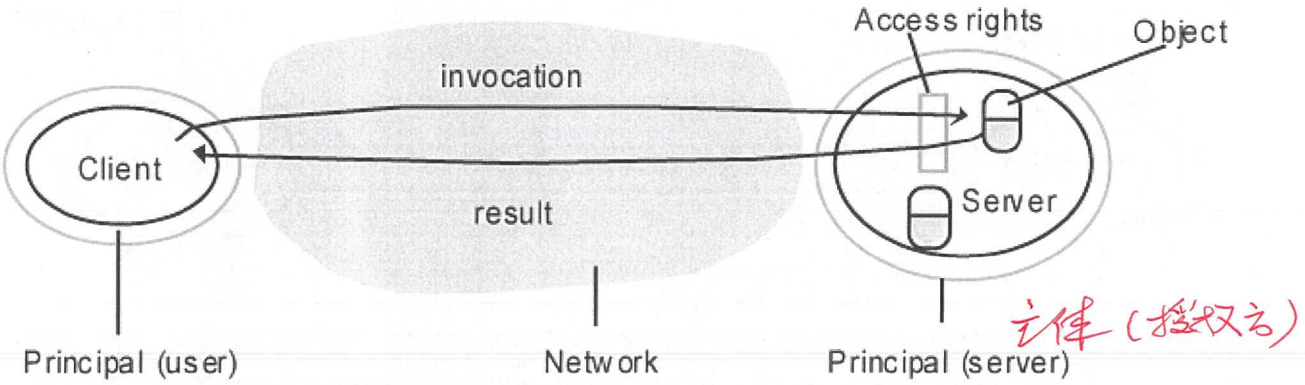
Reliable communication can be defined in terms of two properties:

- **validity**: any message in the outgoing buffer is eventually delivered to the incoming message buffer.
- **integrity**: the message is identical to the one sent, and no messages are delivered twice.

Security Model

Security of a distributed systems is achieved by securing processes, communication channels and protecting objects they encapsulate against unauthorized access.

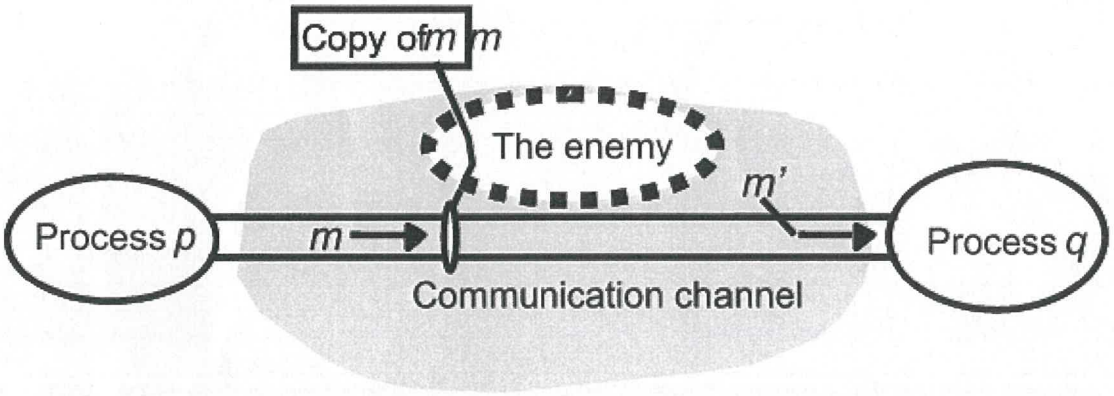
Protecting Objects:



- *Access rights* specify who is allowed to perform operations on an object
- Each invocation and result is associated with a *principal*

Securing Processes and Interactions:

Enemy (adversary) is one capable of sending any message to any process or reading/copying any message between a pair of processes.



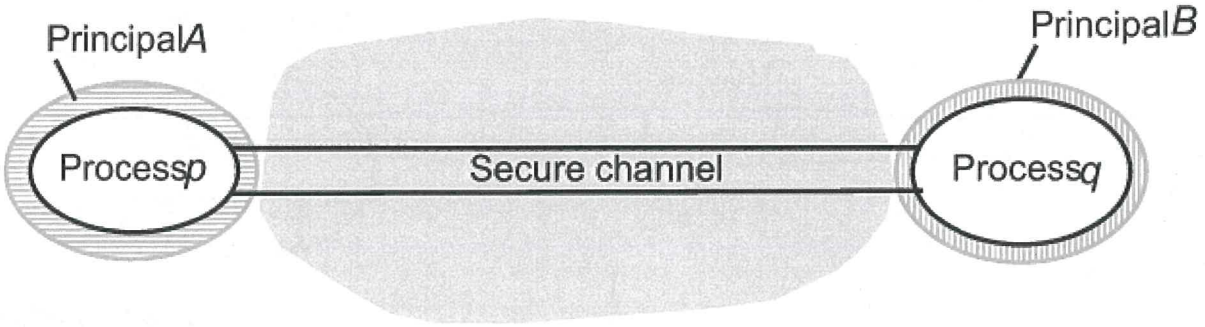
Possible threats from an enemy

- Threats to processes: Servers and clients cannot be sure about the source of the message. Source address can be spoofed.
- Threats to communication channels: Enemy can copy, alter or inject messages
- Denial of service attacks: overloading the server or otherwise triggering excessive delays to the service
- Mobile code: performs operations that corrupt the server or service in an arbitrary way

---

Addressing security threats

- Cryptography and shared secrets: encryption is the process of scrambling messages
- Authentication: providing identities of users
- Secure Channel: Encryption and authentication are used to build secure channels as a service layer on top of an existing communication channel. A secure channel is a communication channel connecting a pair of processes on behalf of its principles


PrincipalA
PrincipalB
Processp
Secure channel
Processq