Session 5
Fundamental Models

Course Leader: Jishmi Jos Choondal

Objectives

At the end of the lecture, the student will be able to

- Discuss the fundamental models employed to design, develop and analyze Distributed Systems

Contents

- Fundamental models

Fundamental Models

- In general, a model contains only the essential ingredients in order to understand and reason about some aspects of a system's behaviour.

- The purpose of a fundamental model is
  - 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.

Fundamental Models

Important aspects of distributed systems are

- Interaction: computation occurs within processes and between processes
- Failure: The correct operation of a distributed system is threatened whenever a fault occurs in any of the computers on which it runs (including software faults) or in the network that connects them.
- Security: The modular nature of distributed systems and their openness exposes them to attack by both external and internal agents.

➢ Fundamental models are: Interaction model, Failure Model and Security Model

Interaction Model

- Computation occurs within processes, the processes interact by passing messages, resulting in communication and coordination between processes.
- The interaction model must reflect the facts that
  - Communication takes place with delays that are often of considerable duration
  - The accuracy with which independent processes can be coordinated is limited by these delays and by the difficulty of maintaining the same notion of time across all the computers in a DS.

Interaction Model

- Two significant factors affecting interacting processes in DS
  - Communication performance is often a limiting characteristic
  - It is impossible to maintain a single global notion of time

Characteristic Performance of Communication Channels

- **Latency** is the delay between the sending of a message by one process and its receipt by another**.** It includes
  - Time taken for transmission through a network to reach its destination.
  - The delay in acessing the network, which increases significantly when the network is heavily loaded.
  - The time taken by OS communication services at both sending and receiving processes.
- **Bandwidth** of a computer network is the total amount of information that can be trasmitted over it in a given time.
- **Jitter** is the variation in the time taken to deliver a series of messages. (lagging in multimedia data)

Computer Clock and Timing Events

- Computer clocks drift from perfect time and their drift rates differ from one another.
- Clock drift rate refers to the relative amount that a computer clock differs from a perfect reference clock.
- Approaches to correcting the times on compuer clocks
  - Computers use radio receivers to get readings from GPS with accuracy of 1microsecond
  - Computer has an accurate time source (GPS) can send timing messages to other computers in network

Two Variants of the Interaction Model

- Synchronous distributed system
  - It has a strong assumption of time
  - It is used for the delivery of multimedia data

- Asynchronous distributed system
  - It makes no assumptions about time.
  - Actual DS are often of this type because of the need for process to share the processors and for communication channels to share the network.

Two Variants of the Interaction Model

- Synchronous distributed system
  - The time to execute each step of a process has known lower and uper bounds.
  - Each message transmitted over a channel is received within a known bounded time
  - Each process has a local clock whose drift rate from real time has a known bound.
- Asynchronous distributed system
  - No bound on process execution speeds
  - No bound on message transmisson delays
  - No bound on clock drift rates.

Event Ordering

- It says whether an event (sending or receiving a message) at one process occurred before, after or concurrently with another event at another process.
- The execution of a system can be described in terms of events and their ordering despite the lack of accurate clocks.

Event Ordering

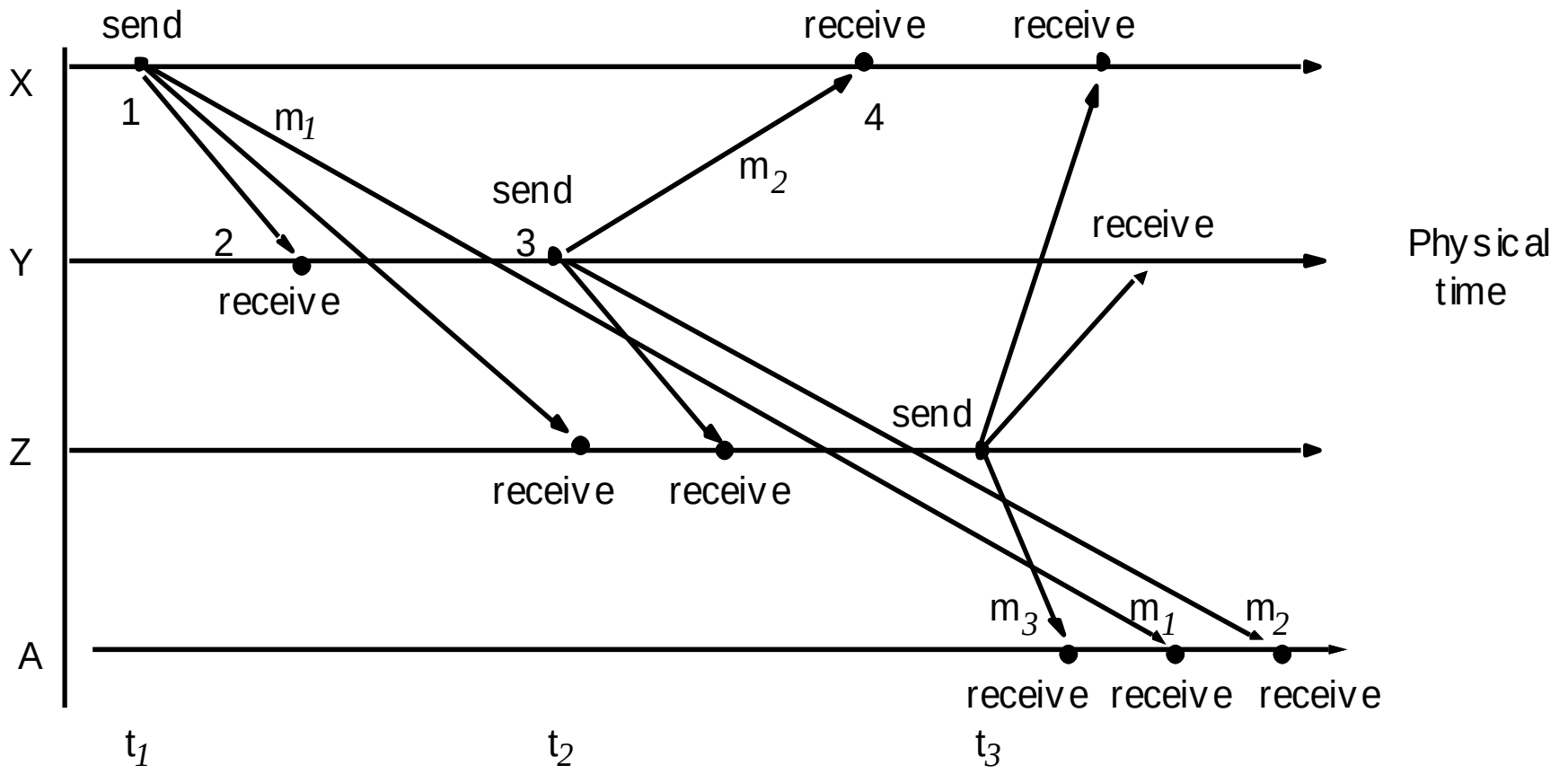
For example, consider the following set of exchanges between a group of email users, X, Y, Z and A, on a mailing list:

1. User X sends a message with the subject *Meeting*.

2. Users Y and Z reply by sending a message with the subject *Re: Meeting*.

In real time, X's message is sent first, and Y reads it and replies; Z then reads both X's message and Y's reply and sends another reply, which references both X's and Y's messages.

Real-time Ordering of Events



Event Ordering

- If the clocks on X's ,Y's and Z's computers could be synchronized, then each message could carry the time on the local computer's clock when it was sent.
- Messages m1,m2 and m3 would carry times t1, t2 and t3 where

  $t1 < t2 < t3$.
- The messages received will be displayed to users according to their time ordering.
- If the clocks are roughly synchronized then these timestamps, will often be in the correct order.

Event Ordering

- Since clocks cannot be synchronised perfectly, Lamport proposed a model of logical time that can be used to provide an ordering among the events at processes running in different computers in a DS.

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

Event Ordering

- First of all we can state a logical ordering for pairs of events.
- Consider only the events concerning X and Y:
  - X sends m1 before Y receives m1
  - Y sends m2 before X receives m2
  - Y receives m1 before sending m2.
- Logical ordering can be done by assigning a number to each event corresponding to its logical ordering, so that later events have higher numbers than earlier ones.
- Here the numbers 1 to 4 on the events X and Y

Failure Model

- It defines the ways in which failure may occur in order to provide an understanding of the effects of failures.
- According to the faults in Failure model, it is classified as
  - Omission failures
  - Arbitrary failures
  - Timing failures
  - Masking failures

Omission Failures

- It refers to cases when a process or communication channel fails to perform actions that it is supposed to do.
- Process Omission failure: The chief omission failure of a process is to ***crash*** (it has halted and will not execute any further steps of its program ever).
  - The method of crash detection relies on the use of ***timeouts*** – a method in which one process allows a fixed period of time for something to occur.
  - A process crash is called *fail-stop* if other processes can detect certainly that the process has crashed.
- Communication omission failure: the loss of messages between the sending process and the outgoing message buffer as *send omission failures*, to loss of messages between the incoming message buffer and the receiving process as *receive-omission failures*, and to loss of messages in between as *channel omission failures*

Arbitrary Failures

- The term *arbitrary* or *Byzantine* failure is used to describe the worst possible failure semantics, in which any type of error may occur.
- For example, a process may set wrong values in its data items, or it may return a wrong value in response to an invocation.
- An arbitrary failure of a process is one in which it arbitrarily omits intended processing steps or takes unintended processing steps.
- Arbitrary failures in processes cannot be detected by seeing whether the process responds to invocations, because it might arbitrarily omit to reply.
- Example: Communication channels can suffer from arbitrary failures

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 not put in its outgoing message buffer.
Receive-omission	Process	A message is put in a process's incoming message buffer, but omission 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.

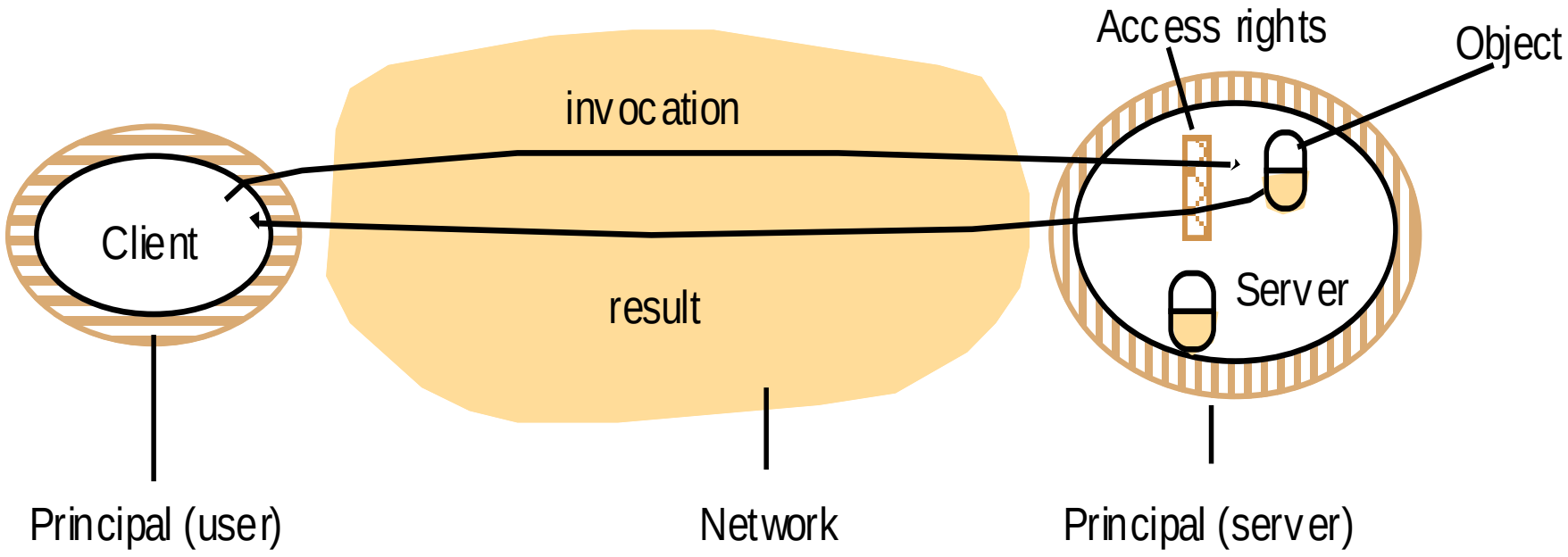
Masking Failures

- A service masks a failure, either by hiding it altogether or by converting it into a more acceptable type of failure.
- Ex. converting an arbitrary failure to Omission failure.
- Omission failure can be hidden by using a protocol that retransmits messages.
- Even process crashes may be masked – by replacing the process and restoring its memory from information stored on disk by its predecessor.

Security Model

- The security of a DS can be achieved by securing the processes and the channels used for their interactions and by protecting the objects that they encapsulate against unauthorised access.
- Protection is described in terms of objects
  - Access rights specify who is allowed to perform the operations of an object
    - Do so by associating with each invocation and each result the authority (principal) on which it is issued.
  - A principal may be a user or a process.
  - The client as well as server responsible for verifying identity each other.

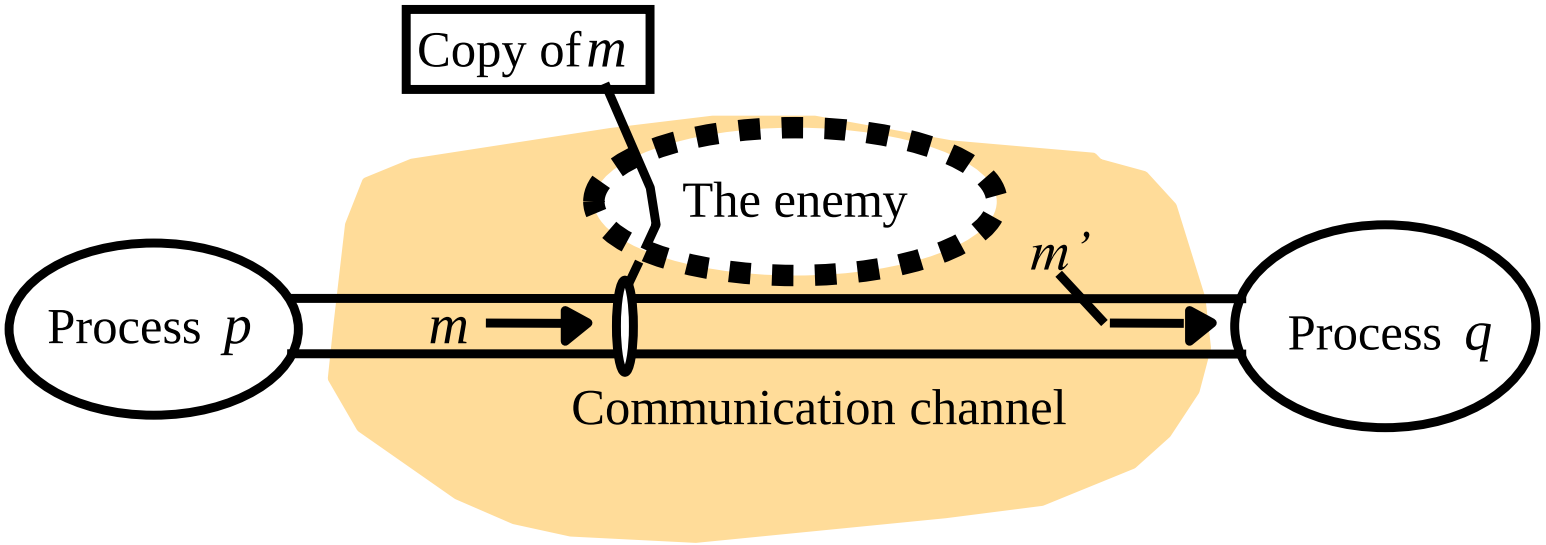
Objects and Principals



The Enemy

- To model security threats we postulate an enemy.
- It is capable of sending any message to any process and reading or copying any message in between a pair of processes.
- The attack may come from a computer that is legitimately connected to the network or from one that is connected in an unauthorised manner.
- The threats from a potential enemy –threats to processes, threats to communication channels and denial of service

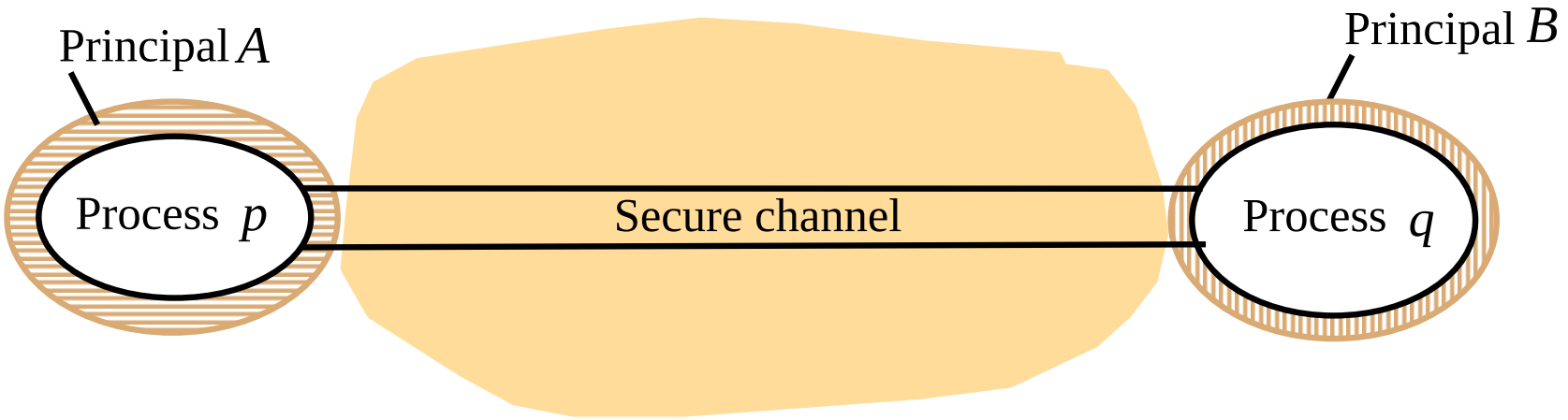
The Enemy



Defeating Security Threats

- Cryptography and shared secrets
- Encryption
- Authentication
- Secure Channels
  - Each of the processes knows reliably the identity of the principal on whose behalf the other process is executing.
  - It ensures the privacy and integrity of the data transmitted across it.
  - Each message includes a physical or logical time stamp to prevent messages from being replayed or recorded.

Secure Channels



Other Possible Threats from an Enemy

- Denial of Service
  - The enemy interferes with the activities of authorized users by making excessive and pointless invocations on services or message transmissions in a network, resulting in overloading of physical resources.
  - These are made with the intention of delaying or preventing actions by other users.
- Mobile Code
  - These code may easily play a Trojan Horse role, purporting to fulfill an innocent purpose but in fact including code that accesses or modifies resources that are legitimately available to the host process but not to the originator of the code.

Summary

- The Fundamental Models of Distributed Systems are Interaction Model, Failure model and Security Model

Questions

Thank you