Revision

About Quiz

- Date: 17 Feb 2021 (Wed)
- Time: 9:15am – 10:00am
- Seated by: 9:00am sharp
- Venue: P1 (SR4F), P2 (SR4G)
- To bring:
  - 2B pencils
  - Pen
  - Eraser
  - Calculator (optional)

MCQ & Structured Questions

Course Introduction

Learning Outcomes

1. Demonstrate understanding of the principles and techniques behind the design and implementation of distributed systems.

2. Explain the basic algorithms and their respective assumptions.

3. Identify software and architecture needed to enable the development of distributed applications.

4. Write programs that can interoperate using well-defined protocols.

5. Debug a code that spans multiple programs running on several machines.

6. Design, implement and debug a real distributed system.

Major Topics & Assessment

Part 1:

- Introduction to Distributed Systems
- Distributed System Architectures
- Communication of Distributed Systems
- Distributed Synchronization

Lecture 1: Introduction

- Enablers of Distributed Systems
- Definition of Distributed Systems
- Important Characteristics of Distributed Systems
- Goals of Distributed Systems
- Types of Distributed Systems

Important Characteristics of DS

- Heterogeneity – How align is a DS in supporting heterogeneity?
- Transparency – How well a DS hide the details of internal organization, communication and processing from users?
- Scalability – How well a DS can expand or scale up?
- Partial Failure – How well a DS reacts to failure?

Main Design Goals

There are four main goals for DSs Designs

- **Accessibility**: Making resources easily accessible
- **Transparency**: Hiding the fact that resources are distributed
- **Openness**: Can integrate different hardware/software platforms from different administrative domains
- **Scalability**: Can add more hardware/software and serve more users

Users Perceive DS as a Single System

DS should be designed so that users can perceive it as a single system

This will reduce the complexity of the system

- Users don’t have to care about distributed aspects of the system
- Users can use the system as if it was a local system

This aspect depends heavily on the Transparency of the design of DS

- Access Transparency
- Location Transparency
- Migration Transparency
- Relocation Transparency
- Replication Transparency
- Concurrency Transparency
- Failure Transparency
- Implementation Transparency

Types of Distributed Systems

There are 3 main types of DSs:

- Distributed Computing Systems: High performance computing tasks such as Cluster, Grid and Cloud.
- Distributed Information Systems: Business information systems, e.g. Enterprise Application Integration (EAI).
- Distributed Embedded Systems (Pervasive): Small devices and components communicating over wireless, e.g. sensor networks, home automation, etc.

Lecture 2: Architectures

Software Architecture vs System Architecture

Software Architectural Styles

System Architecture

Architectures

Logic

Software Architectures tell us

- How various software components are organized
- How various software components should interact

The final instantiation of a software architecture is also referred to as a System Architecture

Software Architecture Styles

Using components and connectors, we can come to various configurations, which, in turn are are classified as architectural styles

Most important Architectural Styles are

- Layered architectures
- Object-based architectures
- Data-centered architectures
- Event-based architectures

System Architectures

Deciding on software components, their interaction, and their placement leads to an instance of software architecture, called a system architecture

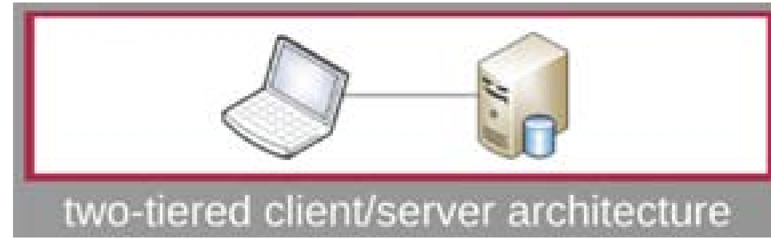
There are two main forms of system architectures: Centralized, and Decentralized architectures

- Centralized Architectures
  - 2-tier architectures
  - 3-tier architectures
  - N-tier architectures
- Decentralized Architectures
  - Structured peer-to-peer
  - Unstructured peer-to-peer

Client/Server Architectures

Three different logical levels suggest a number of possibilities for physically distributing components of a client/server application across several machines such as

- Two-tiered client/server
- Three-tiered client/server
- N-tiered client/server



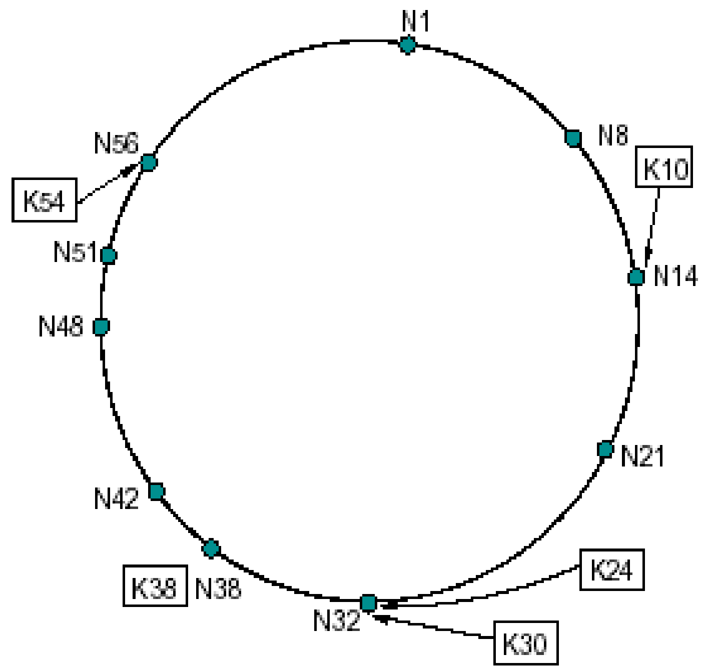
Decentralized Architectures

- Unstructured
  - Centralized: Locating of resource is centralized
  - Fully Decentralized: All peers are responsible to assist
  - Partially Decentralized: Concept of super nodes aggregating a set of peers

- Structured
  - Distributed hash tables (DHTs)
  - Place restrictions on overlay structures and data placement
  - E.g. Chord, Pastry, Tapestry, CAN

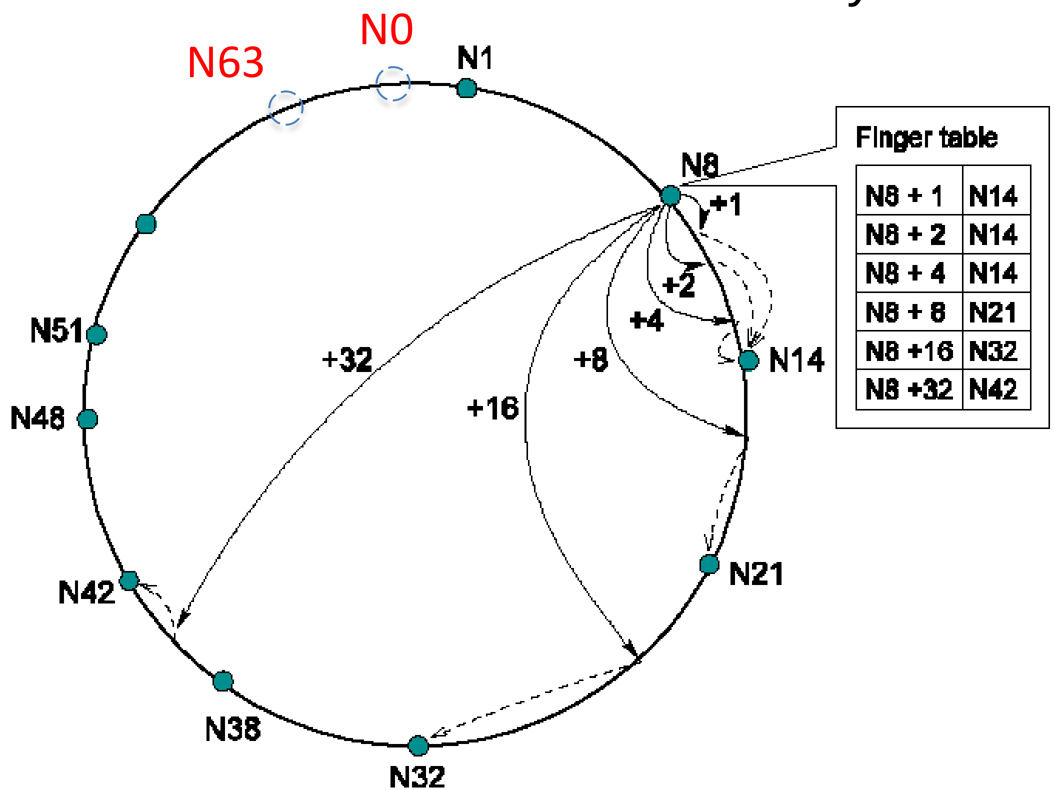
Structured P2P: Chord

- Each node is represented by a unique m-bit ID (hash of IP address) called NodeID
- Each file is represented by a unique m-bit ID (hash of file name) call Key Value
- Distributed Hash Table
  - File name -> Key value
    - E.g. xyz.mp3 -> K10
  - IP address -> Node ID
    - E.g. 123.45.67.89 -> N14
- Keys are assigned to the successor node whose Node ID >= Keys
  - E.g. K10 assigned to N14
- Each node contains info
  - Keys
  - Successor Node ID
- When node joins/departures
  - When a node n joins the network, certain keys previously assigned to n's successor now become assigned to n.
  - When node n leaves the network, all of its assigned keys are reassigned to n's successor.
  - Successor Node ID will also change.



Chord – Finger Table

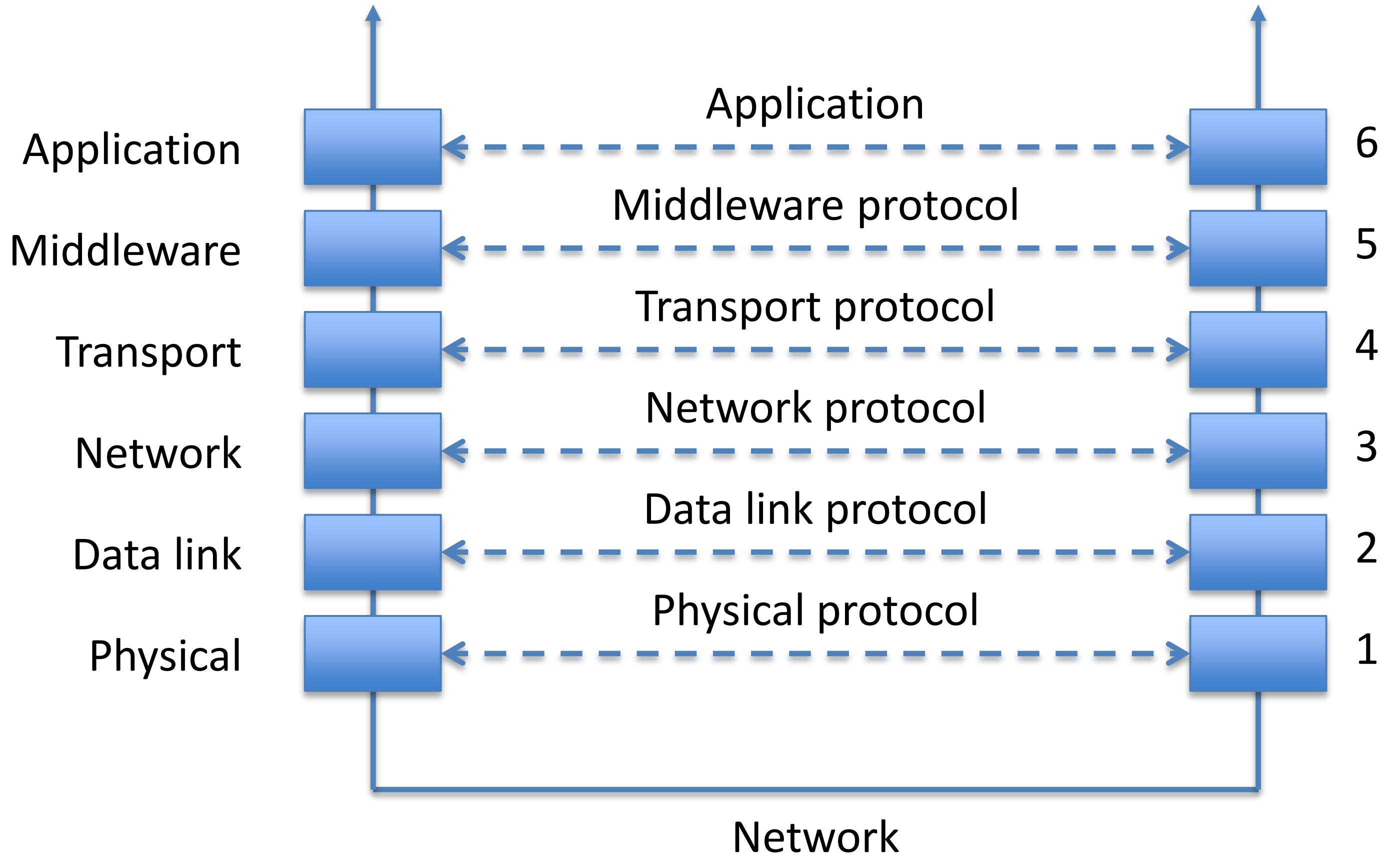
- Extend table of each node to *m* pointers where m is m-bit representation of Node ID
- *i*th finger points to first node that succeeds n by at least 2^{i-1}



Lecture 3: Communications

- OSI Reference Model
- Middleware
- Types of Communications

Middleware Protocols



An adapted reference model for networked communication

Four Types of Communications

1. Persistent Communication
2. Transient Communication
3. Asynchronous Communication
4. Synchronous Communication

Remote Procedure Call (RPC)

RPC should look as much as possible like a local call

- The solution for this is to use client and server stubs

Client stub pack parameters

- Takes its parameters
- Packs parameters into a message (marshaling)
- Sends the message to the server stub

Server stub unpack parameters

- Takes requests coming from the network
- Transforms them into local procedure calls

Message Queuing Systems

Message queuing systems, or message oriented middleware (MOM)

- Provide support for persistent asynchronous communication
- Offer intermediate-term storage capacity for messages
- Sender is not required to be active during message transmission
- Receiver is not required to be active during message transmission

Applications communicate by inserting messages in specific queue

- The message will then be sent to the destination

Message Broker Approach

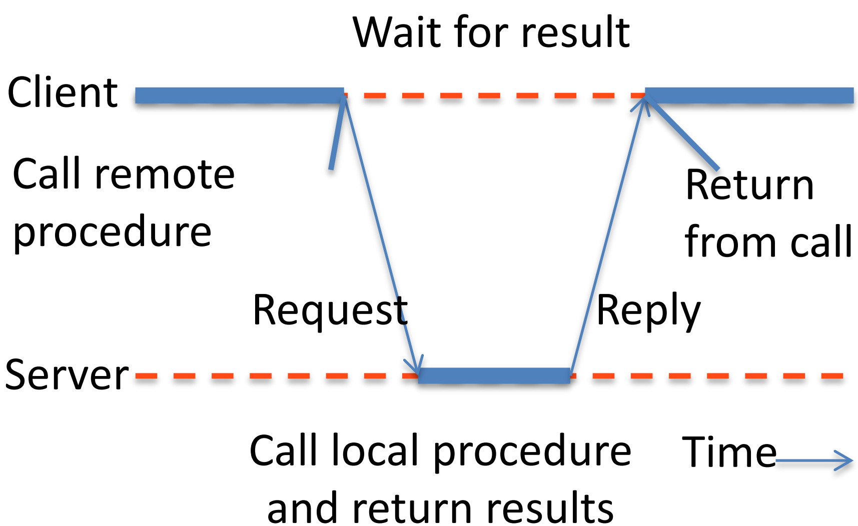
One important task of MQS is to integrate applications into one coherent DS, this requires

- Receiver and sender to agree on the message format

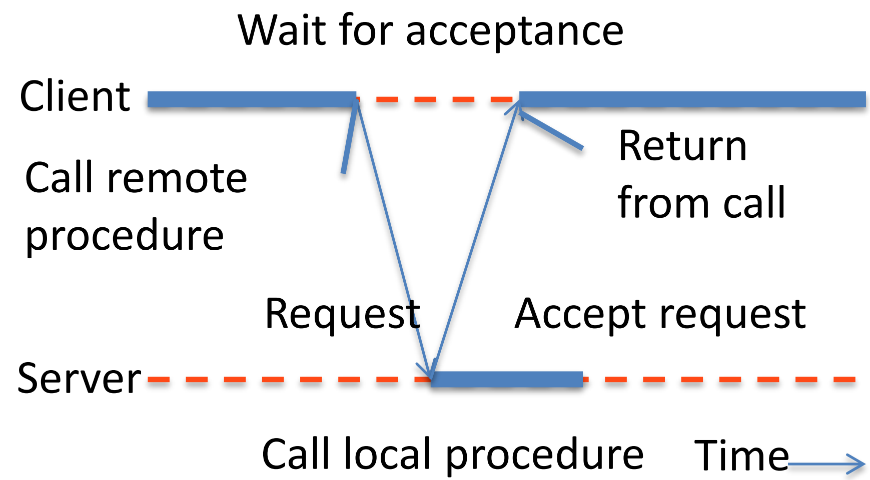
In general, message broker is another application

- Means it is not an integral part of the queuing system
- It acts as an application level gateway
- To convert messages to format that can be read by receiver

CONVENTIONAL VS ASYNCHRONOUS RPC

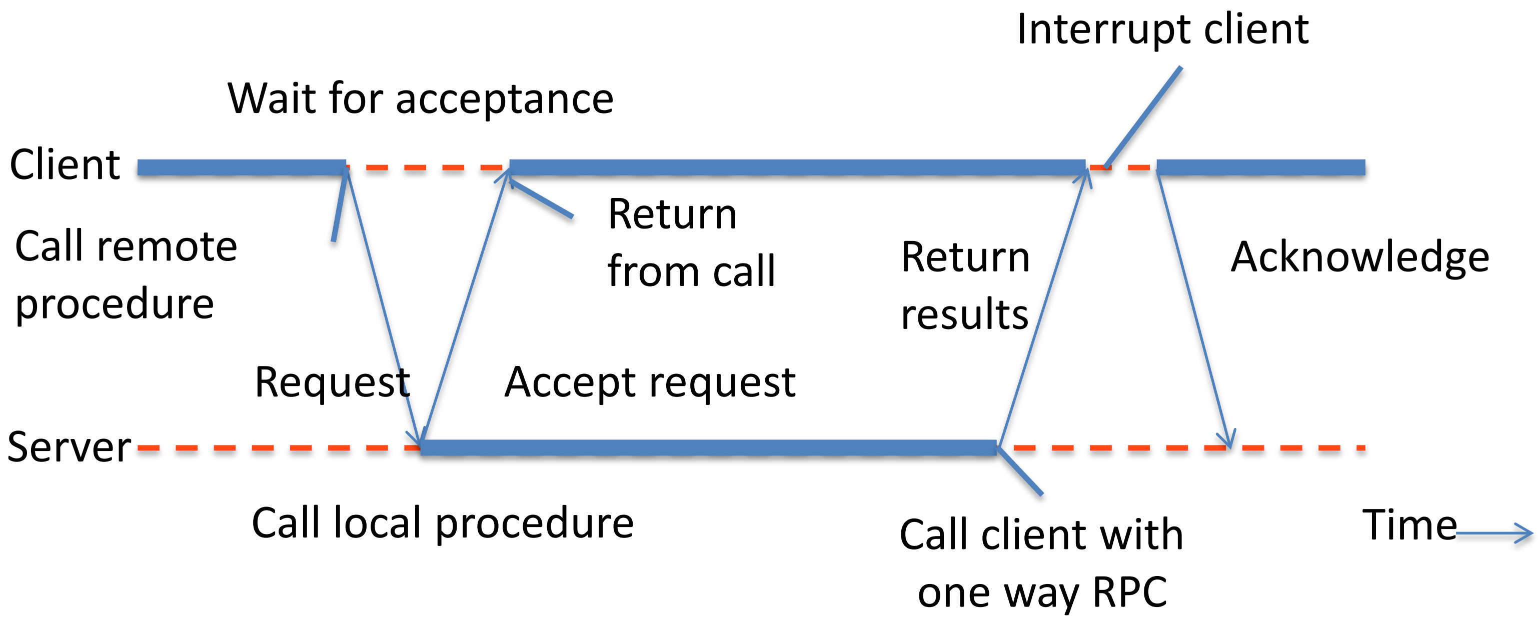


Conventional RPC



Asynchronous RPC

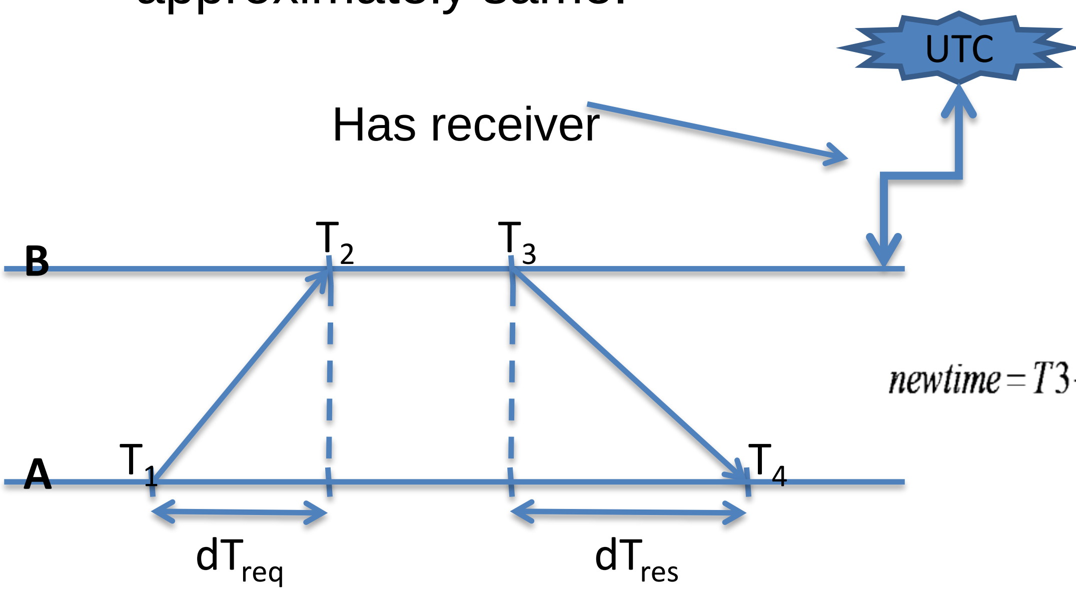
DEFERRED SYNCHRONOUS RPC(s)



Client and server interacting through two asynchronous RPCs

Lecture 4: Synchronization

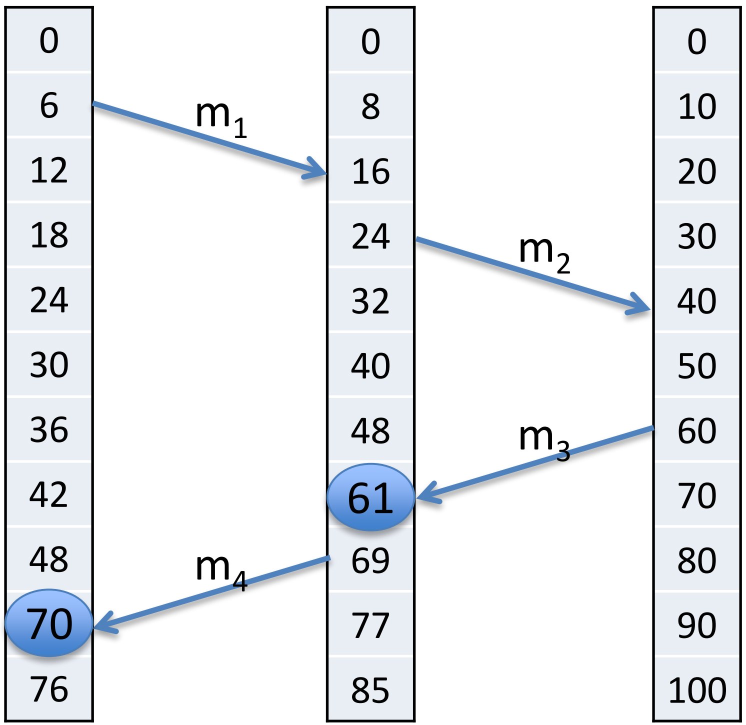
- Clock synchronization
  - Physical clocks
  - Clock synchronization using Cristian algorithm
    - Assumption that the forward and backward delays are approximately same.



$$newtime = T3 + \frac{(T4 - T1) - (T3 - T2)}{2} = T3 + \frac{T_{roundtrip} - T_{processing}}{2}$$

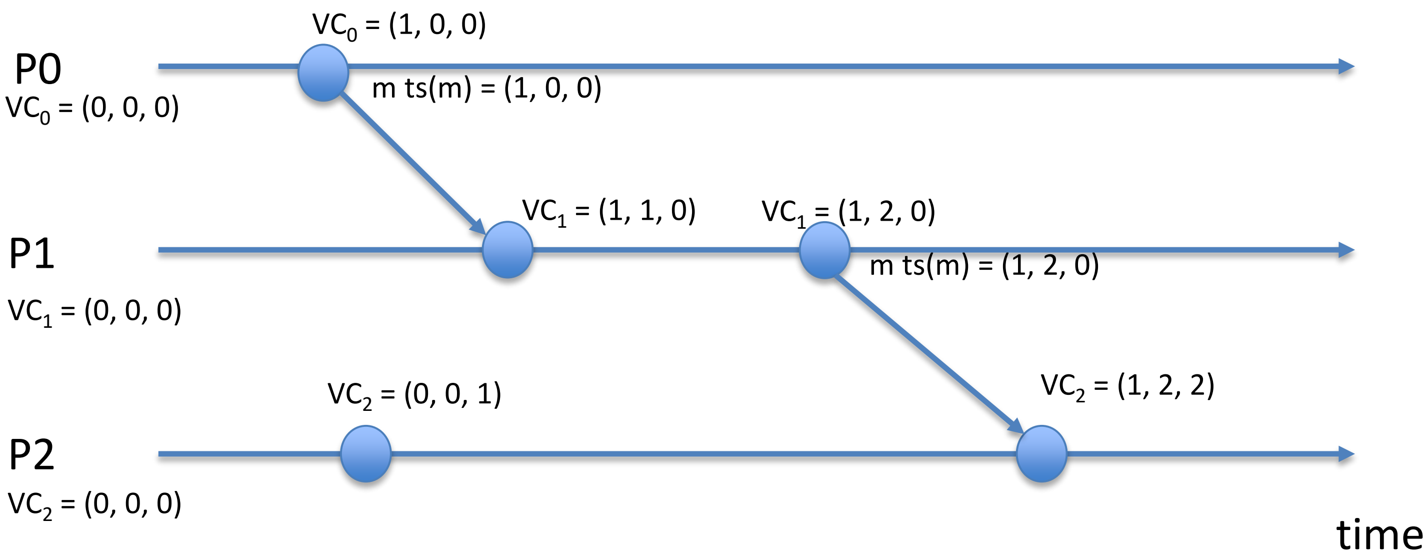
Logical Clocks

– Lamport's Logical Clocks: If a -> b then C(a) < C(b)



- Now consider
  - m3 leaves P3 at 60 and arrives at P2 at 56
  - m4 leaves P2 64 and arrives at P1 at 54
  - These values are clearly impossible
- Lamport's solution with happens-before relation
  - Each message carries the sending time
  - m3 left at 60, it must arrive at 61 or later
  - m4 left at 69, it must arrive at 70 or later

Vector Clock



Mutual Exclusion

- Centralized algorithm: Single Coordinator
- Decentralized algorithm: Multiple Coordinators
- Distributed algorithm:
    - Multicast request to all processes
    - Smaller timestamp wins
- Token ring algorithm
    - Acquire token to gain access

Election Algorithms

- Process with higher ID wins.
  - Bully algorithm
    - Decentralized approach.
    - Any process that detects a coordinator is down can initiate an election
  - Ring algorithm