# CS 3551 DISTRIBUTED COMPUTING

**UNIT I** **INTRODUCTION** **8**

Introduction: Definition-Relation to Computer System Components – Motivation – Message - Passing Systems versus Shared Memory Systems – Primitives for Distributed Communication – Synchronous versus Asynchronous Executions – Design Issues and Challenges; A Model of Distributed Computations: A Distributed Program – A Model of Distributed Executions – Models of Communication Networks – Global State of a Distributed System.

# Distributed Program

$P_1$ $P_2$ $P_3$ $P_4$ idle

$P_1$ ? 1 only

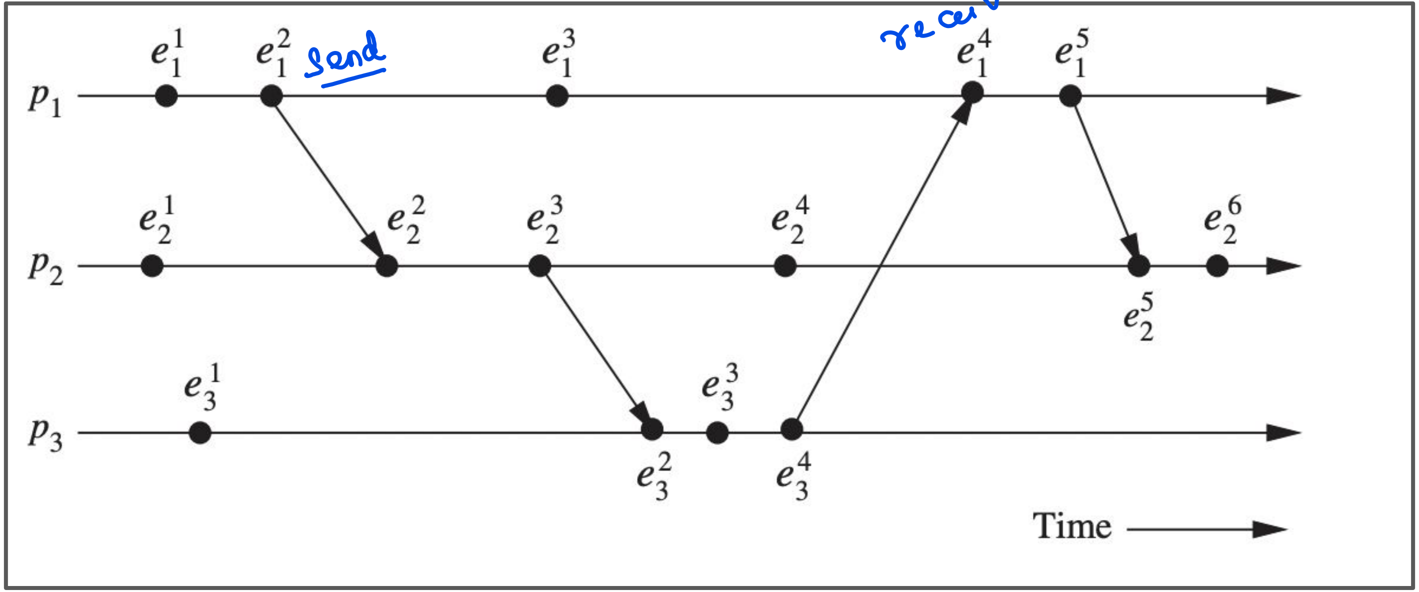
- A distributed program is composed of a set of n asynchronous processes $p_1$, $p_2$, ………, $p_i$, , $p_n$ that communicate by message passing over the communication network.
- We assume that each process is running on a different processor.
- The processes do not share a global memory and communicate solely by passing messages.
- **$C_{ij}$ - Channel from $p_i$ to process $p_j$**
- **$m_{ij}$- a message sent by $p_i$ to $p_j$.**
- Don't share a global clock.
- Process execution and message transfer are asynchronous
- The global state of a distributed computation is composed of the states of the processes and the communication channels
- The state of a process is characterized by the state of its local memory and depends upon the context. The state of a channel is characterized by the set of messages in transit in the channel

$$P_1 \xrightarrow[C_{12}]{m_{12}} P_2$$

# Model for Distributed Execution

• - internal

$e^{\text{no.}}_{\text{process id}}$




binary relation

$H_i = (h_i, \rightarrow_i)$

events

(x) atomic

1) internal -> process

2) send -> sender + channel

3) receive -> receiver + channel

# Model for Distributed Execution


1. Execution of a process consists of a sequential execution of its actions.
2. The actions are atomic and the actions of a process are modeled as three types of events, internal events, message send events(send (m)), and message receive events(rec(m)).
3. The occurrence of events changes the states of respective processes and channels, thus causing transitions in the global system state.
4. An internal event changes the state of the process at which it occurs.
5. A send event (or a receive event) changes the state of the process that sends (or receives) the message and the state of the channel on which the message is sent (or received)

$p_1$

$e_1^1 \quad e_1^2 \quad e_1^3 \ldots$

The events at a process are linearly ordered by their order of occurrence. The execution of process $p_i$ produces a sequence of events $e_i^1$, $e_i^2$, . . . , $e_i^x$, $e_i^{x+1}$, . . . and is denoted by $\mathcal{H}_i$:

$e_1^1 \rightarrow e_1^2$

$$\mathcal{H}_i = (h_i, \rightarrow_i),$$

where $h_i$ is the set of events produced by $p_i$ and binary relation $\rightarrow_i$ defines a linear order on these events. Relation $\rightarrow_i$ expresses causal dependencies among the events of $p_i$.

The send and the receive events signify the flow of information between processes and establish causal dependency from the sender process to the receiver process. A relation $\rightarrow_{msg}$ that captures the causal dependency due to message exchange, is defined as follows. For every message $m$ that is exchanged between two processes, we have

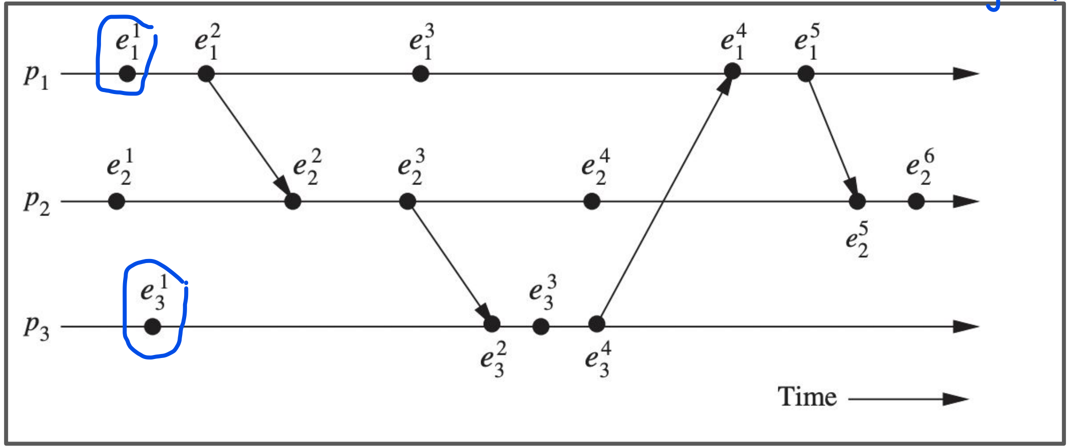
$$send(m) \rightarrow_{msg} rec(m).$$

Relation $\rightarrow_{msg}$ defines causal dependencies between the pairs of corresponding send and receive events.

# Causal Precedence Relation

① $e_i \to e_j$
$e_j \to e_i$
$e_j \| e_i$

$$e_i \to e_j$$




x, y

$e_1^1 \to e_3^1$

vv

① LHS → occur first

② LHS info ⇒ also available for RHS

$e_i \not\to e_j$

Rules

① • for any two events $e_i$ and $e_j$, $e_i \not\to e_j \not\Rightarrow e_j \not\to e_i$

② • for any two events $e_i$ and $e_j$, $e_i \to e_j \Rightarrow e_j \not\to e_i$.

For any two events $e_i$ and $e_j$, if $e_i \not\to e_j$ and $e_j \not\to e_i$, then events $e_i$ and $e_j$ are said to be concurrent and the relation is denoted as $e_i \| e_j$.

## Causal precedence relation

The execution of a distributed application results in a set of distributed events produced by the processes. Let $H = \cup_i h_i$ denote the set of events executed in a distributed computation. Next, we define a binary relation on the set $H$, denoted as $\rightarrow$, that expresses causal dependencies between events in the distributed execution.

$$\forall e_i^x,\ \forall e_j^y \in H,\quad e_i^x \ \rightarrow\ e_j^y \quad \Leftrightarrow \quad \begin{cases} e_i^x \rightarrow_i e_j^y \text{ i.e., } (i = j) \wedge (x < y) \\ \text{or} \\ e_i^x \rightarrow_{msg} e_j^y \\ \text{or} \\ \exists e_k^z \in H : e_i^x \rightarrow e_k^z \ \wedge \ e_k^z \rightarrow e_j^y \end{cases}$$

The causal precedence relation induces an irreflexive partial order on the events of a distributed computation [6] that is denoted as $\mathcal{H} = (H, \rightarrow)$.

# Logical vs Physical Concurrency

$e_1^1$

$e_3$

- In a distributed computation, two events are logically concurrent if and only if they do not causally affect each other.
- Physical concurrency, on the other hand, has a connotation that the events occur at the same instant in physical time.
- Note that two or more events may be logically concurrent even though they do not occur at the same instant in physical time.

$$\Rightarrow \{e_1^3, e_2^4, e_3^3\}$$


thank you


LIKE
COMMENT
SHARE
SUBSCRIBE

# Models of Communication Networks (FIFO, Non-FIFO, Causal Ordering)

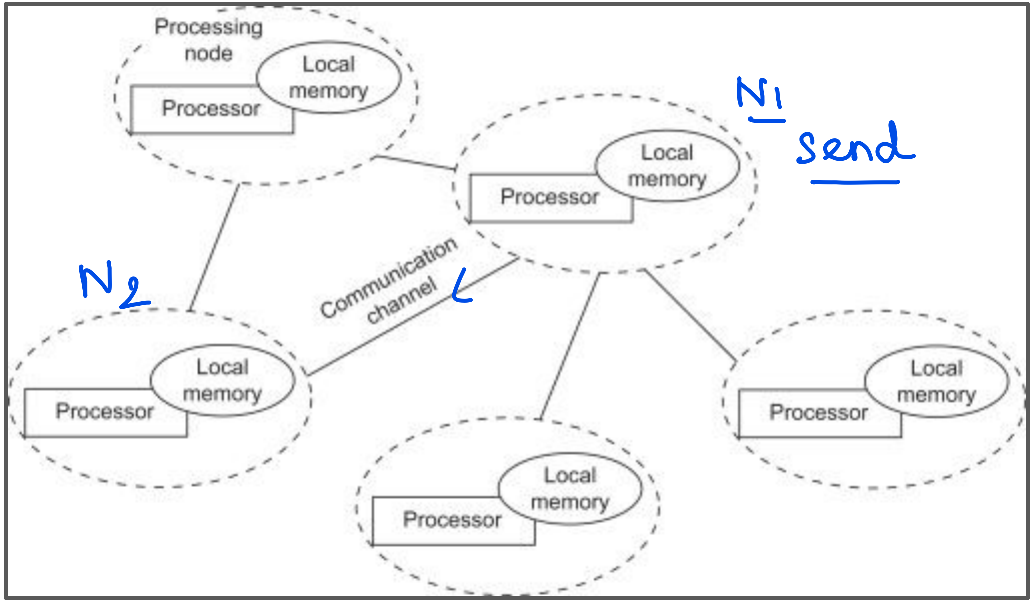
$P_1$ $P_2$

1. **FIFO -** each channel acts as a first-in first-out message queue and thus, message ordering is preserved by a channel. $m_1$ $m_2$ $m_3$ $m_3$ $m_1$ $m_2$
2. **non-FIFO model,** a channel acts like a set in which the sender process adds messages and the receiver process removes messages from it in a random order.
3. **Causal Ordering (built-in synch)-** based on Lamport's "happens before" relation. A system that supports the causal ordering model satisfies the following property: $m_{12}$ $m_{32}$

> **CO**: For any two messages $m_{ij}$ and $m_{kj}$, if $send(m_{ij}) \longrightarrow send(m_{kj})$, then $rec(m_{ij}) \longrightarrow rec(m_{kj})$.
>
> That is, this property ensures that causally related messages destined to the same destination are delivered in an order that is consistent with their causality relation. Causally ordered delivery of messages implies FIFO message delivery. Furthermore, note that CO $\subset$ FIFO $\subset$ Non-FIFO.

# Global State of Distributed System

process
channels




1. **The state of a process at any time is defined by the contents of processor registers, stacks, local memory, etc. and depends on the local context of the distributed application.**
2. **The state of a channel is given by the set of messages in transit in the channel.**
3. **The occurrence of events changes the states of respective processes and channels, thus causing transitions in global system state.**
    a. **For example, an internal event changes the state of the process at which it occurs. A send event (or a receive event) changes the state of the process that sends (or receives) the message and the state of the channel on which the message is sent (or received).**

# State of Process

$p_1 \Rightarrow LS_1^x \Rightarrow LS_1^{1}$ →event complete

process

$p_1 \rightarrow e_1^1, e_1^2, e_1^3$

Let $LS_i^x$ denote the state of process $p_i$ after the occurrence of event $e_i^x$ and before the event $e_i^{x+1}$. $LS_i^0$ denotes the initial state of process $p_i$.

1 $send(m) \leq LS_i^x$ denote the fact that $\exists y{:}1 \leq y \leq x :: e_i^y = send(m)$

$rec(m) \not\leq LS_i^x$ denote the fact that $\forall y{:}1 \leq y \leq x :: e_i^y \neq rec(m)$.

$e_1^1$ $e_1^3$ $LS_1^3$

# State of Channel

Let $SC_{ij}^{x,y}$ denote the state of a channel $C_{ij}$ defined as follows:

$$SC_{ij}^{x,y} = \{m_{ij} \mid send(m_{ij}) \leq LS_i^x \wedge rec(m_{ij}) \not\leq LS_j^y\}.$$

Thus, channel state $SC_{ij}^{x,y}$ denotes all messages that $p_i$ sent up to event $e_i^x$ and which process $p_j$ had not received until event $e_j^y$.

# How to find global state?

$$GS = \{\bigcup_i LS_i^{x_i}, \bigcup_{j,k} SC_{jk}^{y_j, z_k}\}$$

- For a global snapshot to be meaningful, the states of all the components of the distributed system must be recorded at the same instant.
- This will be possible if the local clocks at processes were perfectly synchronized or there was a global system clock that could be instantaneously read by the processes. However, both are impossible.
- Solution: ✓
- Recording at different time will be meaningful provided every message that is recorded as received is also recorded as sent.
- **Basic idea is that an effect should not be present without its cause.**
- **States that don't violate causality are called consistent global states and are meaningful global states**

$P_1$ $P_2$ $P_3$

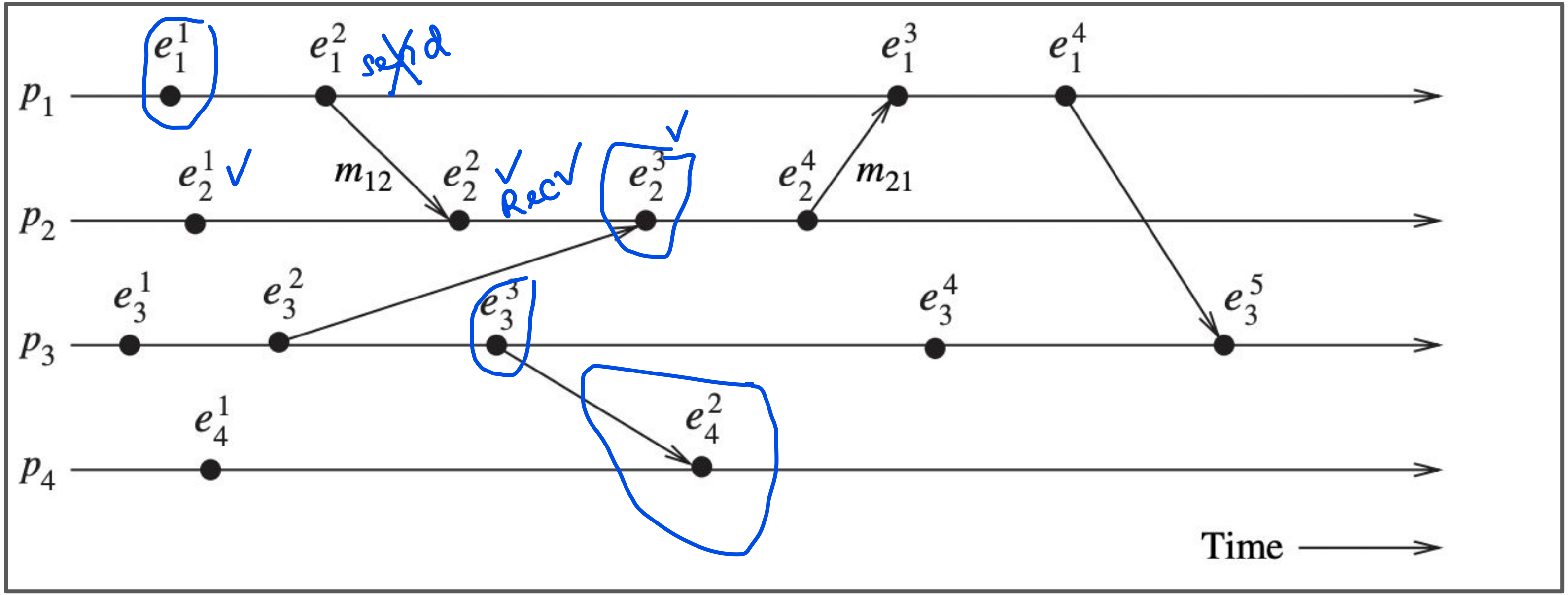
A global state $GS = \{\bigcup_i LS_i^{x_i}, \bigcup_{j,k} SC_{jk}^{y_j, z_k}\}$ is a *consistent global state* iff it satisfies the following condition:

$$\forall m_{ij} : send(m_{ij}) \not\preceq LS_i^{x_i} \Rightarrow m_{ij} \notin SC_{ij}^{x_i, y_j} \wedge rec(m_{ij}) \not\preceq LS_j^{y_j})$$

That is, channel state $SC_{ik}^{y_i, z_k}$ and process state $LS_k^{z_k}$ must not include any message that process $p_i$ sent after executing event $e_i^{x_i}$.

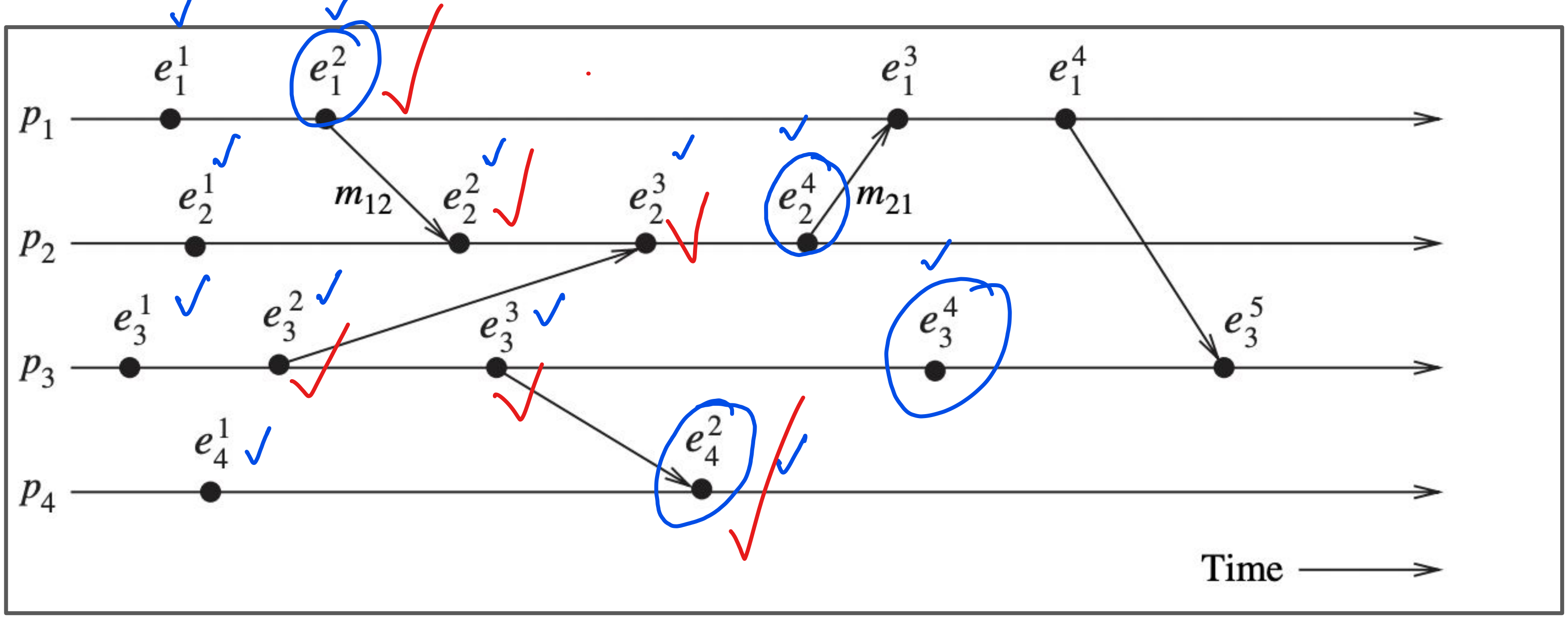
$A \longrightarrow B$

send ✓ rec ✓




Inconsistent state

of local states $\{LS_1^1, LS_2^3, LS_3^3, LS_4^2\}$ is inconsistent because the state of $p_2$ has recorded the receipt of message $m_{12}$, however, the state of $p_1$ has not recorded its send.




Consistent state

states $\{LS_1^2, LS_2^4, LS_3^4, LS_4^2\}$ is consistent; all the channels are empty except $C_{21}$ that contains message $m_{21}$.

# Strongly Consistent

A global state $GS = \{\bigcup_i LS_i^{x_i}, \bigcup_{j,k} SC_{jk}^{y_j,z_k}\}$ is *transitless* iff

$$\forall i, \forall j \ : 1 \leq i, j \ \leq n \ :: SC_{ij}^{y_i,z_j} = \phi.$$

Thus, all channels are recorded as empty in a transitless global state. A global state is *strongly consistent* iff it is transitless as well as consistent. Note that in Figure 2.2, the global state consisting of local states $\{LS_1^2, LS_2^3, LS_3^4, LS_4^2\}$ is strongly consistent.


LIKE

COMMENT

SHARE

SUBSCRIBE