Monsoon 2022

Distributed Computing
- Global Snapshot Algorithm

Dr. Rajendra Prasath

Indian Institute of Information Technology Sri City, Chittoor

> Distributed Computing?

➤ How will you design a **Distributed Algorithm?**

➤ **Learn to Solve** using Distributed Algorithms

Overview

Recap: Distributed Systems

A Distributed System:

➔ A collection of independent systems that appears to its users as a single coherent system

➔ A system in which hardware and software components of networked computers communicate and coordinate their activity only by passing messages

➔ A computing platform built with many computers that:

   ➔ Operate concurrently
   ➔ Are physically distributed (have their own failure modes)
   ➔ Are linked by a network
   ➔ Have independent clocks

Recap: Characteristics

➔ Concurrent execution of processes:
  - ➔ Non-determinism, Race Conditions, Synchronization, Deadlocks, and so on
➔ No global clock
  - ➔ Coordination is done by message exchange
  - ➔ No Single Global notion of the correct time
➔ No global state
  - ➔ No Process has a knowledge of the current global state of the system
➔ Units may fail independently
  - ➔ Network Faults may isolate computers that are still running
  - ➔ System Failures may not be immediately known
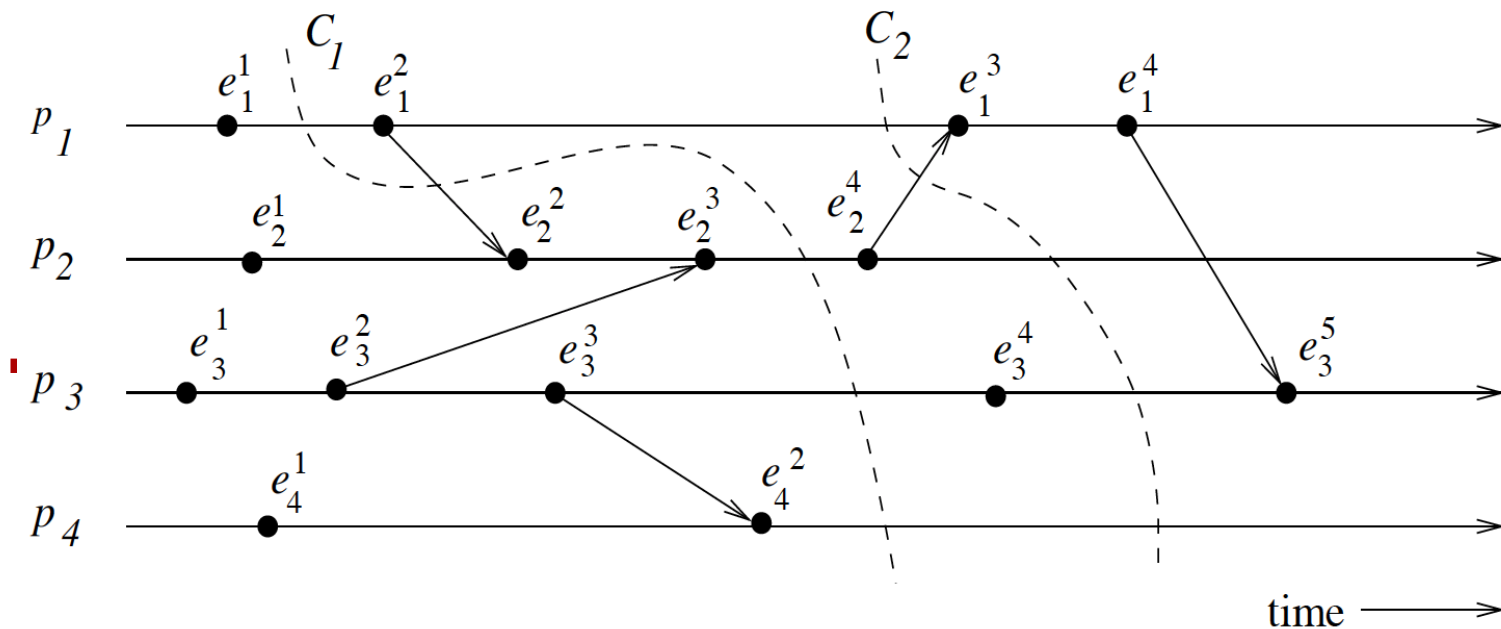
What did you learn so far?

➔ **Goals / Challenges in Message Passing systems**

➔ **Distributed Sorting**
➔ **Space-Time diagram**
➔ **Partial Ordering / Total Ordering**

➔ **Causal Precedence Relation**
    ➔ **Happens Before**

➔ **Concurrent Events**
    ➔ **How to define Concurrent Events**
    ➔ **Logical vs Physical Concurrency**

➔ **Causal Ordering**
➔ **Logical Clocks vs Physical Clocks**

Recap

> About this Lecture

What do we learn today?

➤ This covers a model of distributed computations that every algorithm designer needs to know

   ➤ **Global States**
   ➤ **Global Snapshot algorithm**

Let us explore these topics ➔ ➔ ➔

Today's Focus

Causal Ordering

➔ The "causal ordering" model is based on Lamport's "happens before" relation

➔ A system that supports the causal ordering model satisfies the following property:

CO: For any two messages m_{ij} and m_{kj}
   if $send(m_{ij}) \rightarrow send(m_{kj})$,
   then $receive(m_{ij}) \rightarrow receive(m_{kj})$

➔ 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. (Note that CO $\subset$ FIFO $\subset$ Non-FIFO.)

➔ Causal ordering model considerably simplifies the design of distributed algorithms because it provides a built-in synchronization.
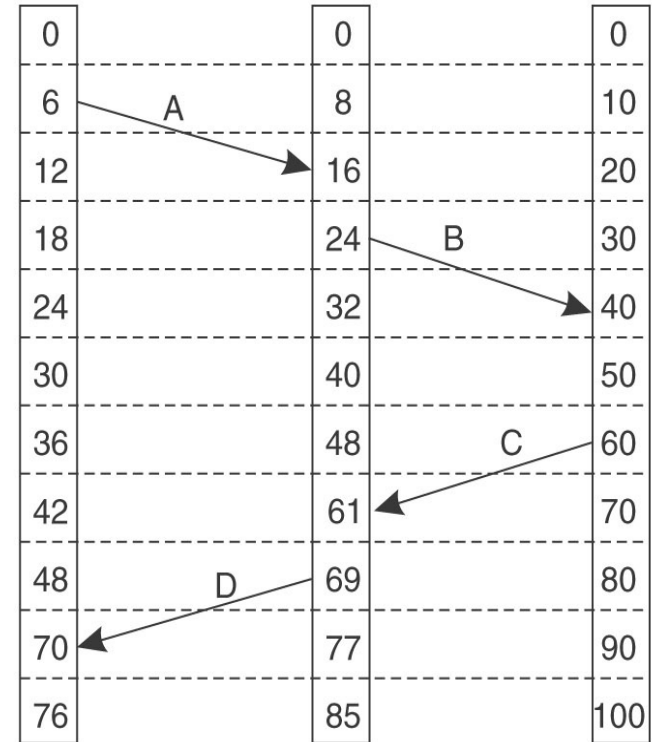
A Close Look
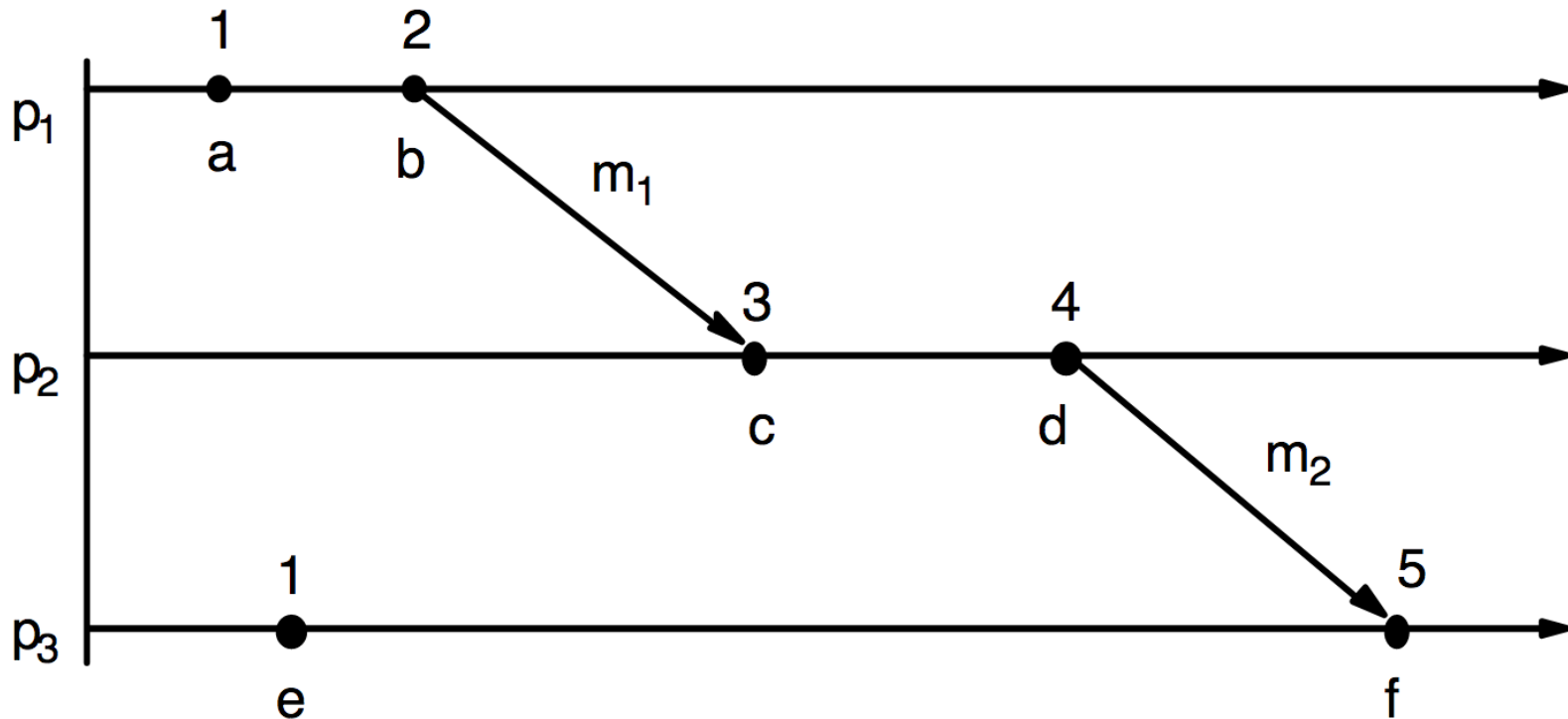
➜ e → e' ⇒ L(e) < L(e') but not vice versa
➜ Example: event b and event e

Correction of Clocks
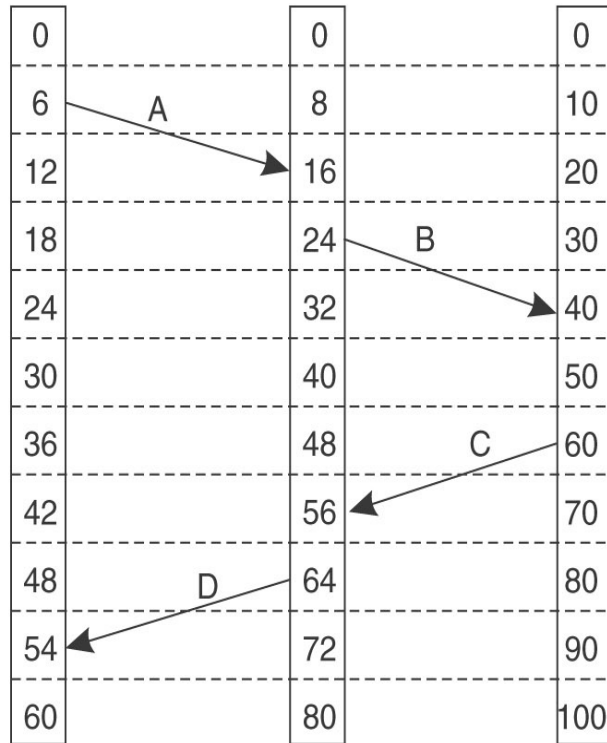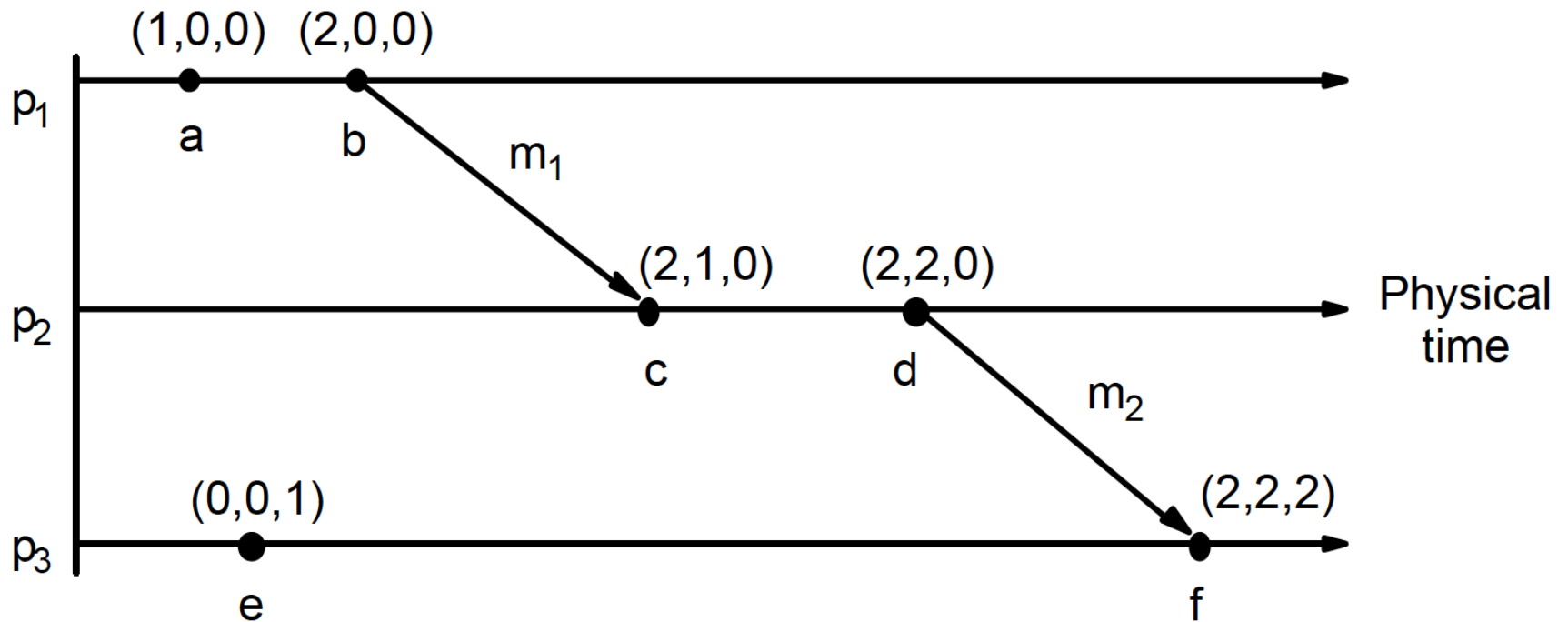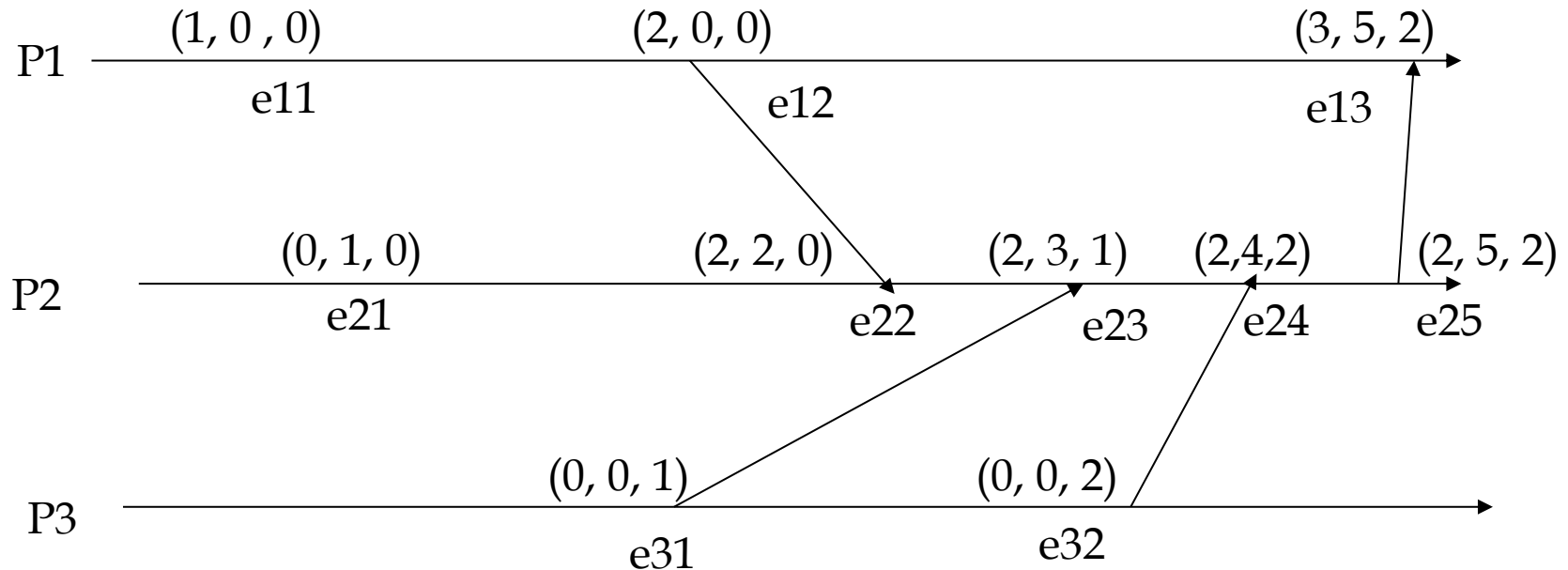

(a)   (b)

Vector Clocks

➜ **Vector clocks allow causality to be captured**

➜ **Rules of Vector Clocks:**

  ➜ A vector clock $VC(a)$ is assigned to an event a

  ➜ If $VC(a)<VC(b)$ for events a and b, then event a is known to causally precede b

➜ **Each Process P_i maintains a vector VC_i with the following properties:**

  ➜ $VC_i[i]$ is the number of events that have occurred so far at P_i that is, $VC_i[i]$ is the **local logical clock** at process P_i

  ➜ If $VC_i[j]=k$ then P_i knows that k events have occurred at P_j. It is thus P_i's knowledge of the local time at P_j

An Example

An illustrative example

P1 ——— (1, 0 , 0) ——————— (2, 0, 0) ——————————————— (3, 5, 2) →
        e11   e12   e13

P2 ——— (0, 1, 0) ——————— (2, 2, 0) ——— (2, 3, 1) ——— (2,4,2) (2, 5, 2) →
        e21   e22   e23   e24   e25

P3 ——————————————— (0, 0, 1) ——————— (0, 0, 2) ——————————————— →
                        e31   e32

Less than or equal:

➡ $ts(a) \leq ts(b)$ **if** $ts(a)[i] \leq ts(b)[i]$ **for all** i

(3,3,5) ≤ (3,4,5)

➡ $ts(e11) = (1, 0, 0)$ **and** $ts(e22) = (2, 2, 0)$

This implies $e11$ $e22$

Global States in a Distributed System

Partial / Total Ordering

A relation $\leq$ is a total order on a set S (" $\leq$ totally orders S") if the following properties hold:

1. Reflexivity: $a \leq a$ for all a in S

2. Anti-Symmetry: $a \leq b$ and $b \leq a$ implies $a = b$

3. Transitivity: $a \leq b$ and $b \leq c$ implies $a \leq c$

4. Comparability (Trichotomy law):
   For any a,b in S, either $a \leq b$ or $b \leq a$.

First 3 properties → the axioms of a partial ordering
Adding Trichotomy law defines a total ordering $H=(H, \rightarrow)$

Global State

➔ A collection of the local states of its components:

    ➔ The processes and the communication channels

➔ The state of a process is defined by the local contents of processor: registers, stacks, local memory

➔ The state of channel depends the set of messages in transit in the channel

➔ An internal event changes only state of the process

➔ A send event changes

    ➔ state of the process that sends the message and

    ➔ the state of the channel on which the message is sent.

➔ Similarly a receive event changes

    ➔ the state of the process that receives the message and

    ➔ the state of the channel on which the message is received

DC - Monsoon 2022 @ IIIT Sri City
    20/09/2022
    16

Global State (contd)

Notations

➔ LS_i^x **denotes the state of** p_i **after 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

➔ LS_i^x **is a result of the execution of all the events executed by process** p_i **till** e_i^x

➔ **Let** $send(m) \leq LS_i^x$ **denote the fact:**

$$\exists y,\ 1{\leq}y{\leq}x\ s.t\ e_i^y = send(m)$$

➔ **Let** $rec(m)\ (not \leq)\ LS_i^x$ **denote the fact:**

$$\forall y,\ 1{\leq}y{\leq}x\ s.t\ e_i^y\ (not\ equal\ to)\ rec(m)$$

Global State (contd)

➔ **The global state of a distributed system is a collection of the local states of the processes and the channels.**

A global state GS is defined as,

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

➔ **For a global state to be meaningful, the states of all the components of the distributed system must be recorded at the same instant**

➔ **Two important situations (Impossible !!):**
  ➔ **local clocks at processes were perfectly synchronized**
  ➔ **there were a global system clock that can be instantaneously read by the processes**

A Consistent Global State

Basic idea:

➔ A state should not violate causality – an effect should not be present without its cause

➔ A message cannot be received if it was not sent.

➔ Such states are called consistent global states and are meaningful global states.

➔ Inconsistent global states are not meaningful in the sense that a distributed system can never be in an inconsistent state

A Consistent Global State

Definition:

➔ **A global state is a consistent global state iff**

$$\forall m_{ij} : \ send(m_{ij}) \not\preceq LS_i^{x_i} \ \Leftrightarrow \ m_{ij} \notin SC_{ij}^{x_i,y_j} \bigwedge rec(m_{ij}) \not\preceq LS_j^{y_j}$$
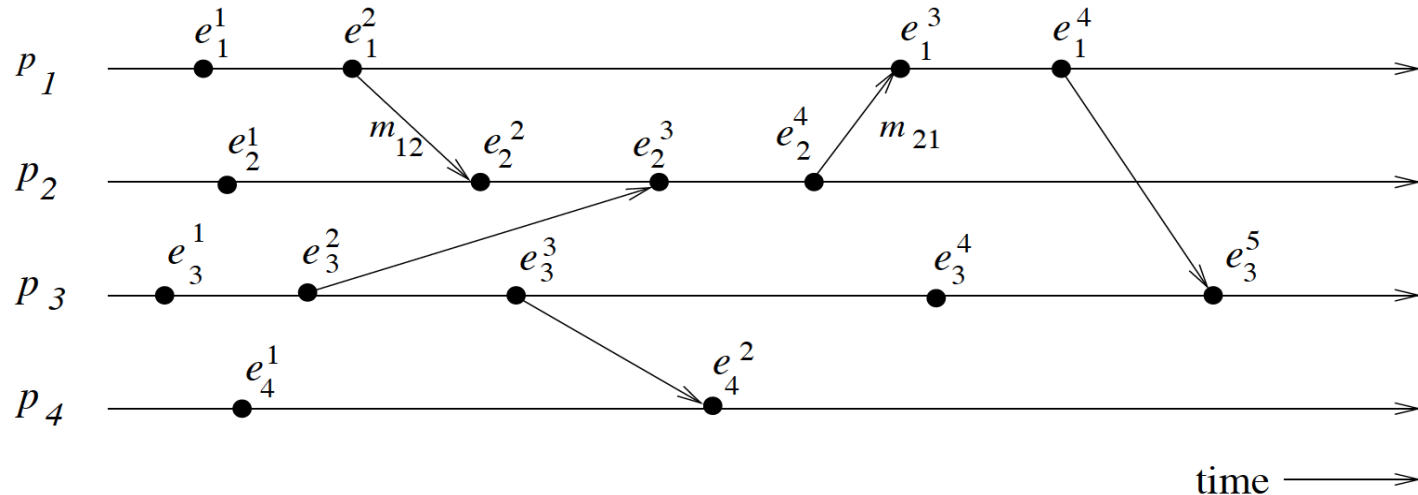
Where the global state is given by

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

➔ **This implies that the channel state and process state must not include any message that process P$_i$ sent after executing event**

Consistent Global State – An Example

→ **Space-Time Diagram**

→ $GS_1 = \{LS_1^1, LS_2^3, LS_3^3, LS_4^2\}$ is **inconsistent**
  → The state of p_2 has recorded the receipt of m_{12} however, the state of p_1 has not recorded its send
→ $GS_2 = \{LS_1^2, LS_2^4, LS_3^4, LS_4^2\}$ is **consistent**
  → All channels are empty except C_{21} that contains m_{21}

Consistent Global State – Details

➔ A global state GS1 = {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}

  ➔ The state of p_1 has not recorded its send

➔ A global state GS2 consisting of local states GS2 = {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}.

Changes in Global State?

➔ **The global state changes whenever an event happens**

- ➔ Process sends message
- ➔ Process receives message
- ➔ Process performs a local event

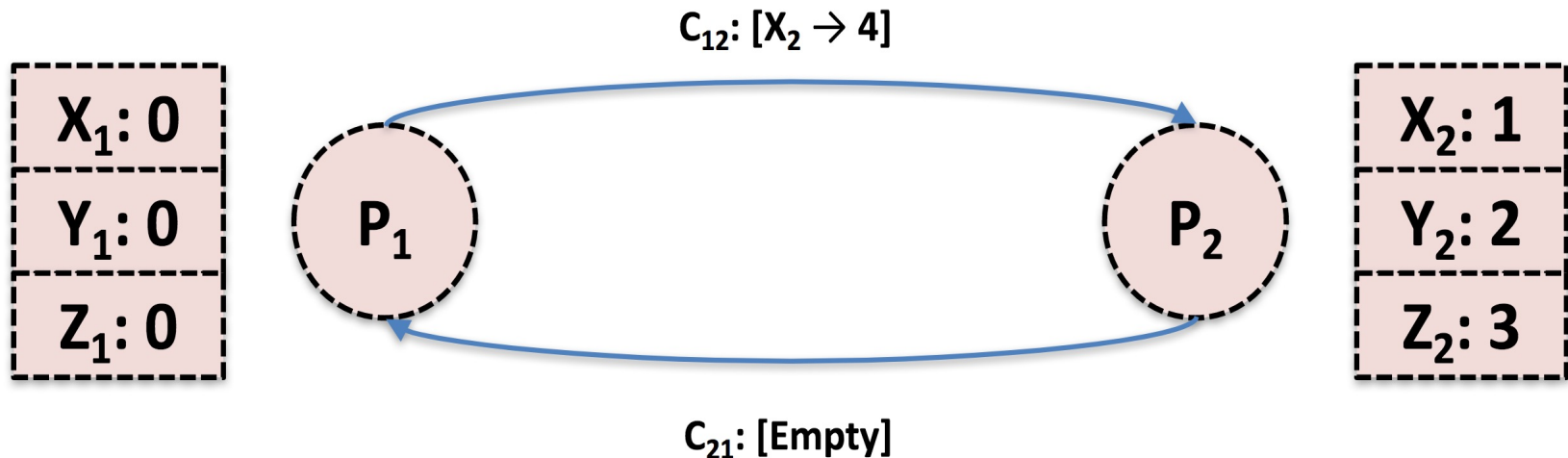➔ **Moving from state to state obeys causality**

An Example Scenario

➔ **There are two processes: P_1 and P_2**

   ➔ **Channel C_{12} from P_1 to P_2**

   ➔ **Channel C_{21} from P_2 to P_1**

C_{12}

P_1 P_2

C_{21}
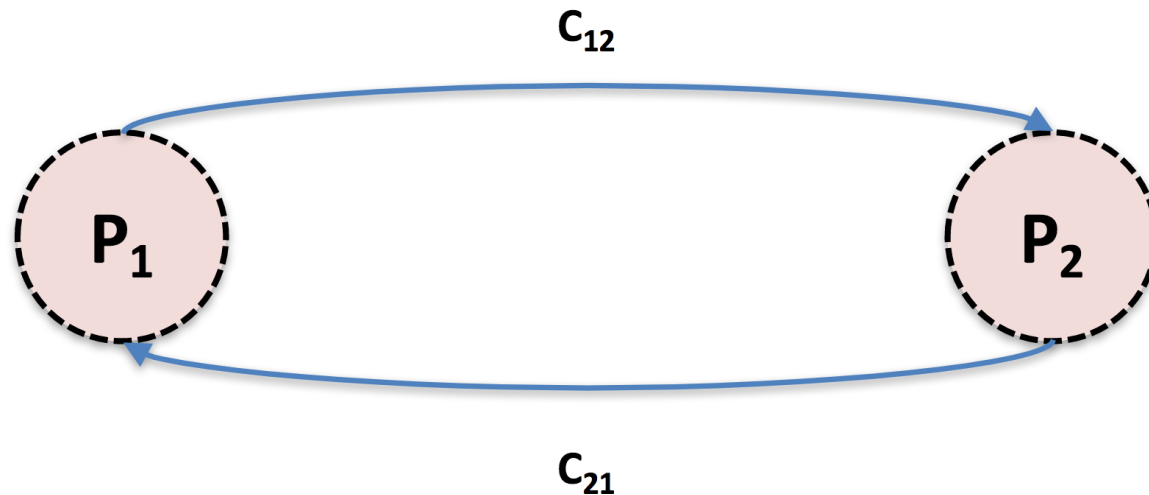
An Example (contd)

➔ P$_1$ tells P$_2$ to change its state variable, X$_2$, from 1 to 4

C_{12}: [X$_2$ → 4]

| X$_1$: 0 |
| Y$_1$: 0 |
| Z$_1$: 0 |

P$_1$ P$_2$

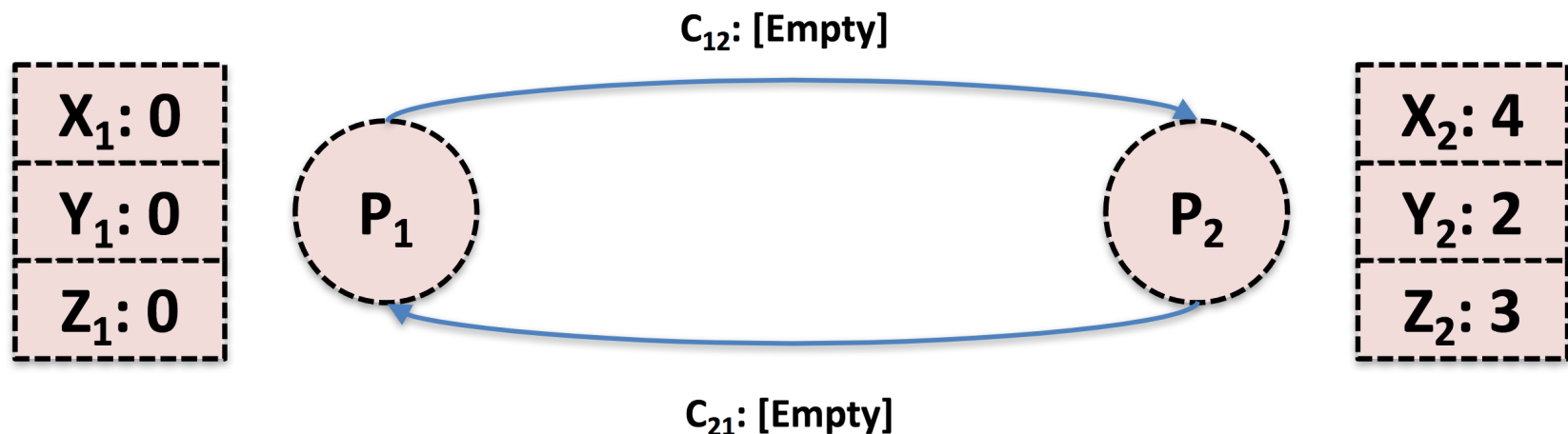| X$_2$: 1 |
| Y$_2$: 2 |
| Z$_2$: 3 |

C_{21}: [Empty]

An Example (contd)

➔ **P_2 receives message from P_1 to change its state variable, X_2 from 1 to 4**
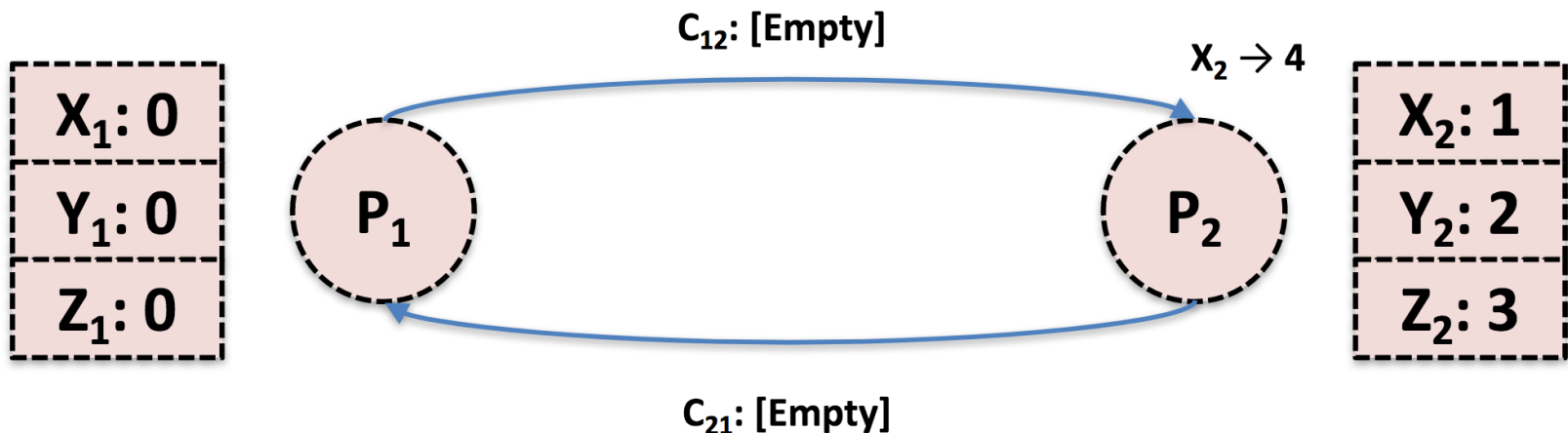
An Example (contd)

➔ **P_2 changes its state variable, X_2 from 1 to 4**

Why do we need Global Shapshot?

Checkpointing:

➔ restart if the application fails

Collecting garbage:

➔ remove objects that do not have any references

Detecting deadlocks:

➔ can examine the current application state

Other debugging:

➔ a little easier to work with than printf

Chandy-Lamport's Global Snapshot Recording Alorithm

System Model (Lamport and Chandy)

Problem: How to record a global shapshot (state for each process and each channel)?

System Model (assumptions):

➜ **N Processes in the system**

➜ **Communication channels are bi-directional between each ordered pair** (p_i, p_j)**:** $p_i \rightarrow p_j$ **and** $p_j \rightarrow p_i$

➜ **Channels are FIFO: First In First Out**

➜ **No Failures**

➜ **All messages arrive intact and not duplicated**

Requirements

➔ **Snapshot should not interfere with normal application actions**

➔ **It should not require application to stop sending messages**

➔ Each Process is able to record its own state:

    ➔ Process State: Application-defined state or in the worst case

    ➔ Its heaps, registers, program counters, code and other related application interfaces

➔ Global state is collected in a distributed manner (no centralized approach)

➔ Any Process may initiate the recording of a snapshot (there can be multiple shapshots, we focus on only one)

Initiating a snapshot

➔ Let's say process P_i initiates the snapshot

➔ P_i records its own state and prepares a special marker message (distinct from application messages)

➔ Send the marker message to all other processes (using $N-1$ outbound channels)

➔ Start recording all incoming messages from channels C_{ji} for j not equal to i
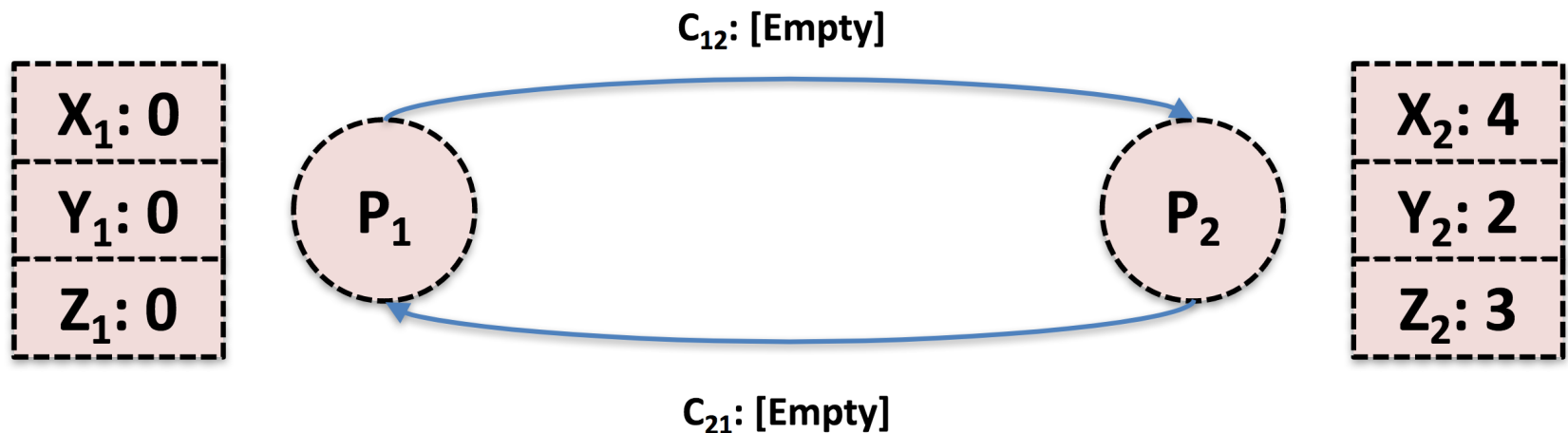
Propagating a snapshot

→ **For all processes P_i (including the initiator) consider a message on channel C_{kj}**

→ **If P_j sees the marker message for the first time**

  → P_j **records own state and marks C_{kj} as empty**
  → **Send the marker message to all other processes (using $N\text{-}1$ outbound channels)**
  → **Start recording all incoming messages from channels C_{lj} for l not equal to j or k**

→ **Otherwise**

  → **add all messages from inbound channels since we began recording their states**

Terminating a snapshot

➔ **All processes have received a marker** (and recorded their own state)

➔ **All processes have received a marker on all $N-1$ incoming channels** (and recorded their states)

➔ Later, a central server can gather the partial state to build a global snapshot (Do we require this? – Not Really!!)
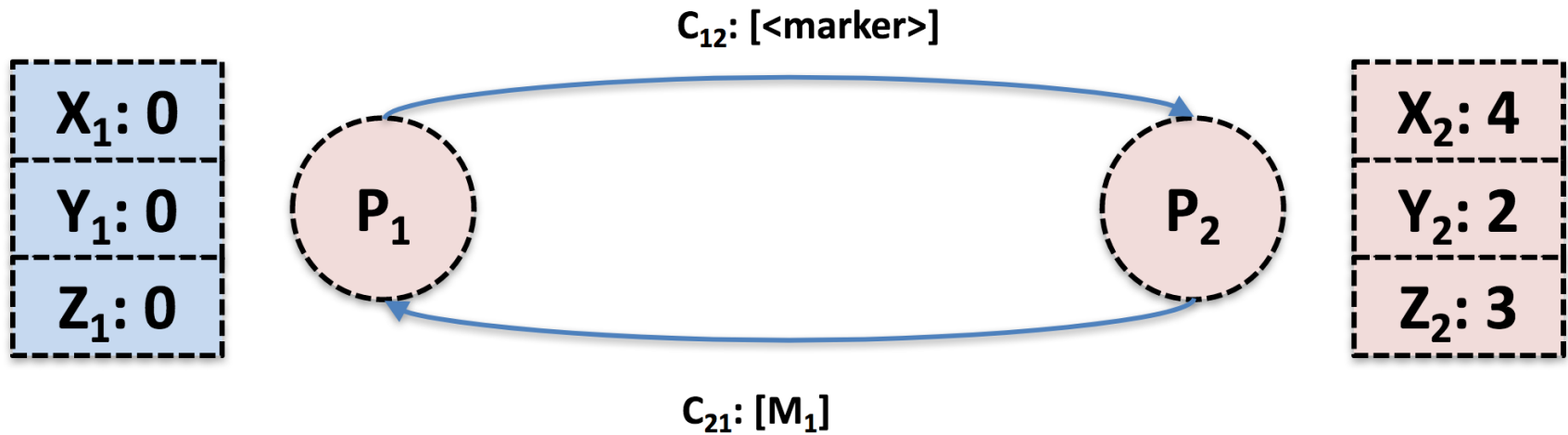
# A Close Look of an Example

➔ **Initially all channels are empty!**

C_{12}: [Empty]

X$_1$: 0 → $X_1: 0$

P$_1$

$X_1: 0$
$Y_1: 0$
$Z_1: 0$

P$_2$

$X_2: 4$
$Y_2: 2$
$Z_2: 3$

C_{21}: [Empty]

An Example (contd)

➔ Then, P_1 sends a marker message to P_2 and begins recording all messages on inbound channels

➔ Meanwhile, P_2 sent a message to P_1

C_{12}: [<marker>]

X_1: 0
Y_1: 0
Z_1: 0

P_1

P_2

X_2: 4
Y_2: 2
Z_2: 3

C_{21}: [M_1]

An Example (contd)

➔ **P_2 receives a marker message for the first time, so records its state**

➔ **P_2 then sends a marker message to P_1**

C_{12}: [Empty]

<marker>

X_1: 0
Y_1: 0
Z_1: 0

P_1

P_2

X_2: 4
Y_2: 2
Z_2: 3

M_1

C_{21}: [<marker>]
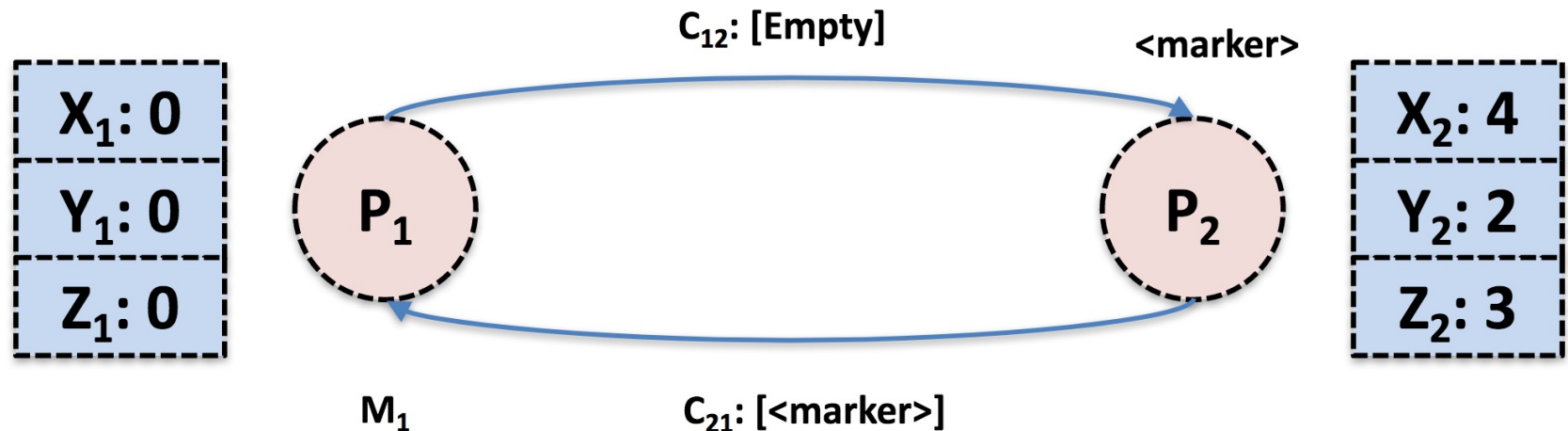
An Example (contd)

➔ **P_1 has already sent a marker message, so it records all messages it received on inbound channels to the appropriate channel's state**

C_{12}: [Empty]

X_1: 0	
Y_1: 0	
Z_1: 0	

P_1

P_2

X_2: 4	
Y_2: 2	
Z_2: 3	

M_1

C_{21}: [Empty]
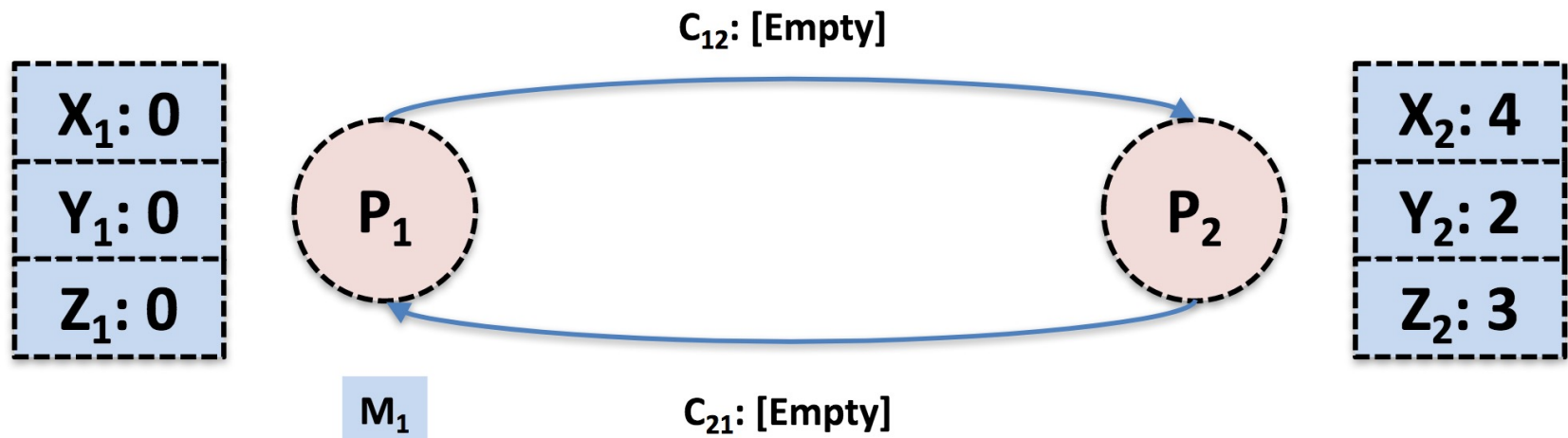
Causal Consistency

➜ Related to Lamport's clock partial ordering

➜ An event is pre-snapshot if it occurs before the local snapshot on a process

➜ Post-snapshot if afterwards

➜ If an event A happens causally before an event B, and B is a pre-snapshot, then A is too

Summary

➜ **Global Snapshots**
- ➜ Causal Precedence Relations
- ➜ Global State of a DS

➜ **Chandy – Lamport's Algorithm**
- ➜ Global Snapshot Recording Algorithm
- ➜ Initiating Snapshot
- ➜ Propagating Snapshots
- ➜ Terminating the Snapshot Algorithm

➜ **Causal Consistency**

- ➜ Many more to come up … stay tuned in !!

Penalties

➤ Every Student is expected to strictly follow a fair Academic Code of Conduct to avoid penalties

➤ Penalties is heavy for those who involve in:
  ➤ Copy and Pasting the code
  ➤ Plagiarism (copied from your neighbor or friend – in this case, both will get "0" marks for that specific take home assignments)
  ➤ If the candidate is unable to explain his own solution, it would be considered as a "copied case"!!
  ➤ Any other unfair means of completing the assignments

Help among Yourselves?

- **Perspective Students** (having CGPA above 8.5 and above)

- **Promising Students** (having CGPA above 6.5 and less than 8.5)

- **Needy Students** (having CGPA less than 6.5)
    - Can the above group help these students? (Your work will also be rewarded)

- You may grow a culture of **collaborative learning** by helping the needy students

How to reach me?

➔ **Please leave me an email:**
rajendra [DOT] prasath [AT] iiits [DOT] in

➔ **Visit my homepage @**

➔ https://www.iiits.ac.in/people/regular-faculty/dr-rajendra-prasath/

(OR)

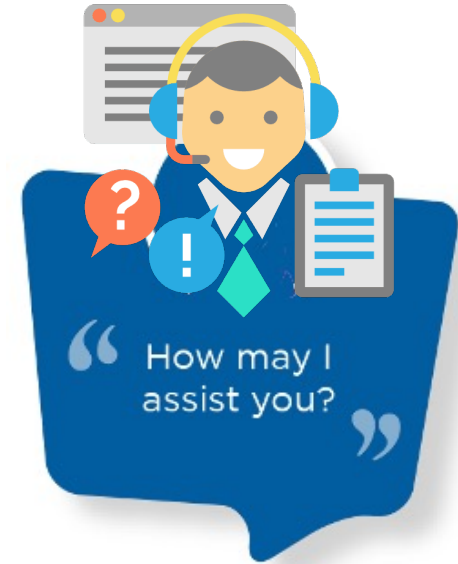➔ http://rajendra.2power3.com

Assistance

➤ **You may post your questions to me at any time**

➤ **You may meet me in person on available time or with an appointment**

➤ **You may ask for one-to-one meeting**

Best Approach

➤ **You may leave me an email any time (email is the best way to reach me faster)**

Questions
It's Your Time

How may I
assist you?

Contact Information

Dr. Rajendra Prasath
IIIT Sri City, Chittoor