Global States of A Distributed System

Course: Distributed Computing

Faculty: Dr. Rajendra Prasath

Spring 2019

About this topic

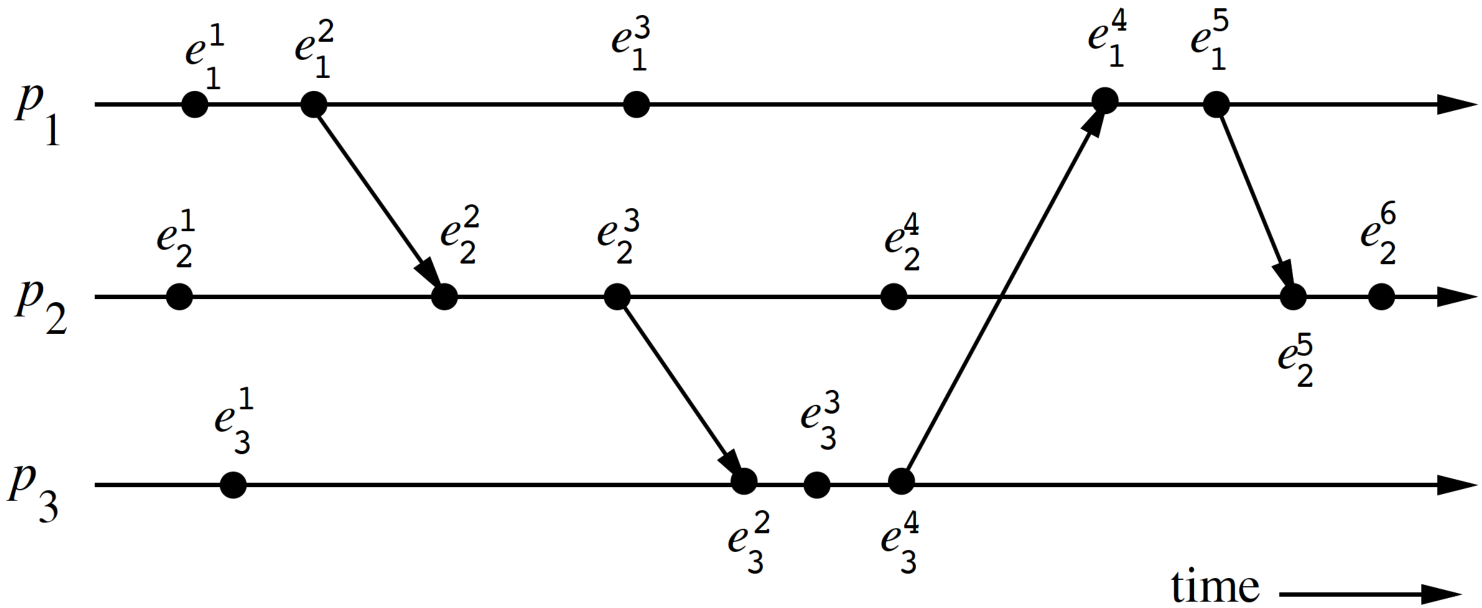
This course covers essential aspects of **Global states of a Distributed System and its related concepts**

RECAP

What did you learn so far?

- Goals / Challenges in Message Passing systems
- Distributed Sorting
- Space-Time Diagram
- Partial Ordering / Total Ordering
- Causal Ordering
- Causal Precedence Relation
    - Happens Before
- Concurrent Events
    - How to define Concurrent Events
    - Logical vs Physical Concurrency
- Local Clocks and Vector Clocks

A State-Time diagram – An Example



- **For Process P'_1:**

  Second event is a message send event

  First and Third events are internal events

  Fourth event is a message receive event

Causal Precedence Relation (contd)

- The relation $\rightarrow$ is as defined by Lamport

  "happens before"

An event e_1 happens before the event e_2 and denoted by $e_1 \rightarrow e_2$ if the following holds true:

- e_1 occurs before e_2 on the same process OR
- e_1 is the send message and e_2 is the corresponding receive message OR
- There exists another event e' such that e_1 happens before e' and e' happens before e_2

Causal Precedence Relation (contd)

- For any two events e_i and e_j , $e_i \not\rightarrow e_j$ denotes the fact that event e_j does not directly or transitively dependent on event e_i

  That is, event e_i does not causally affect event e_j
- In this case, event e_j is not aware of the execution of e_i or any event executed after e_i on the same process.

Note the following two rules:

- For any two events e_i and e_j

  $$e_i \not\rightarrow e_j \text{ does not imply } e_j \not\rightarrow e_i$$
- For any two events e_i and e_j

  $$e_i \rightarrow e_j \Rightarrow e_j \not\rightarrow ei \ .$$

Concurrent Events

- For any two events e_i and e_j:

  if $e_i \not\rightarrow e_j$ and $e_j \not\rightarrow e_i$,

  then events e_i and e_j are said to be concurrent (denoted as $e_i \,||\, e_j$)

Example:

$$e_3^3 \,||\, e_2^4 \;\; and \; e_2^4 \,||\, e_1^5 \;\; but \; e_3^3 \; not \,||\, e_1^5$$

- The relation || is not transitive;

  that is,

$$(e_i \,||\, e_j) \wedge (e_j \,||\, e_k) \; does \; not \; imply \; e_i \,||\, e_k$$

- For any two events e_i and e_j in a distributed execution,

$$e_i \rightarrow e_j \; OR \; e_j \rightarrow e_i, \; OR \; e_i \,||\, e_j$$

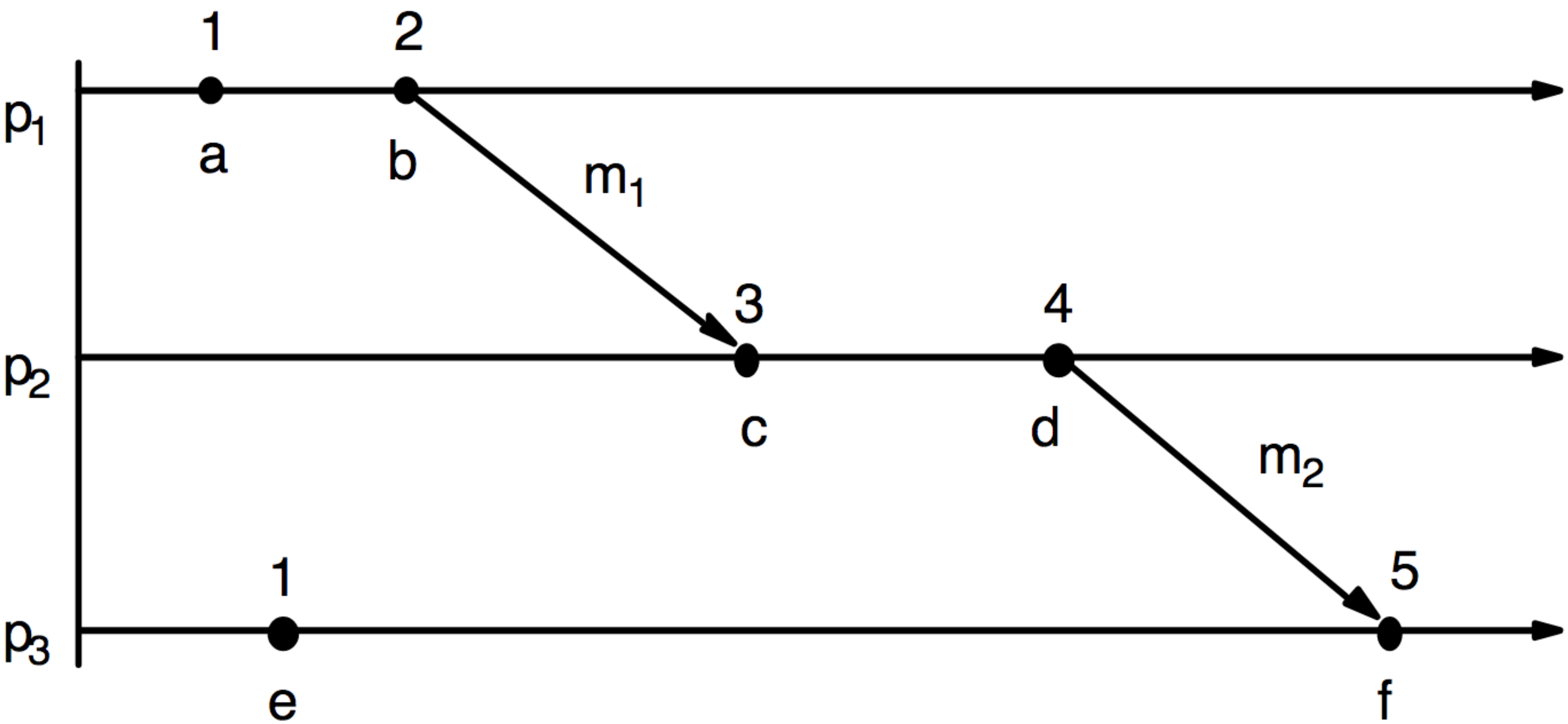
Causal Ordering

- 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 $send(m_{ij}) \longrightarrow send(m_{kj})$,
then $receive(m_{ij}) \longrightarrow receive(m_{kj})$

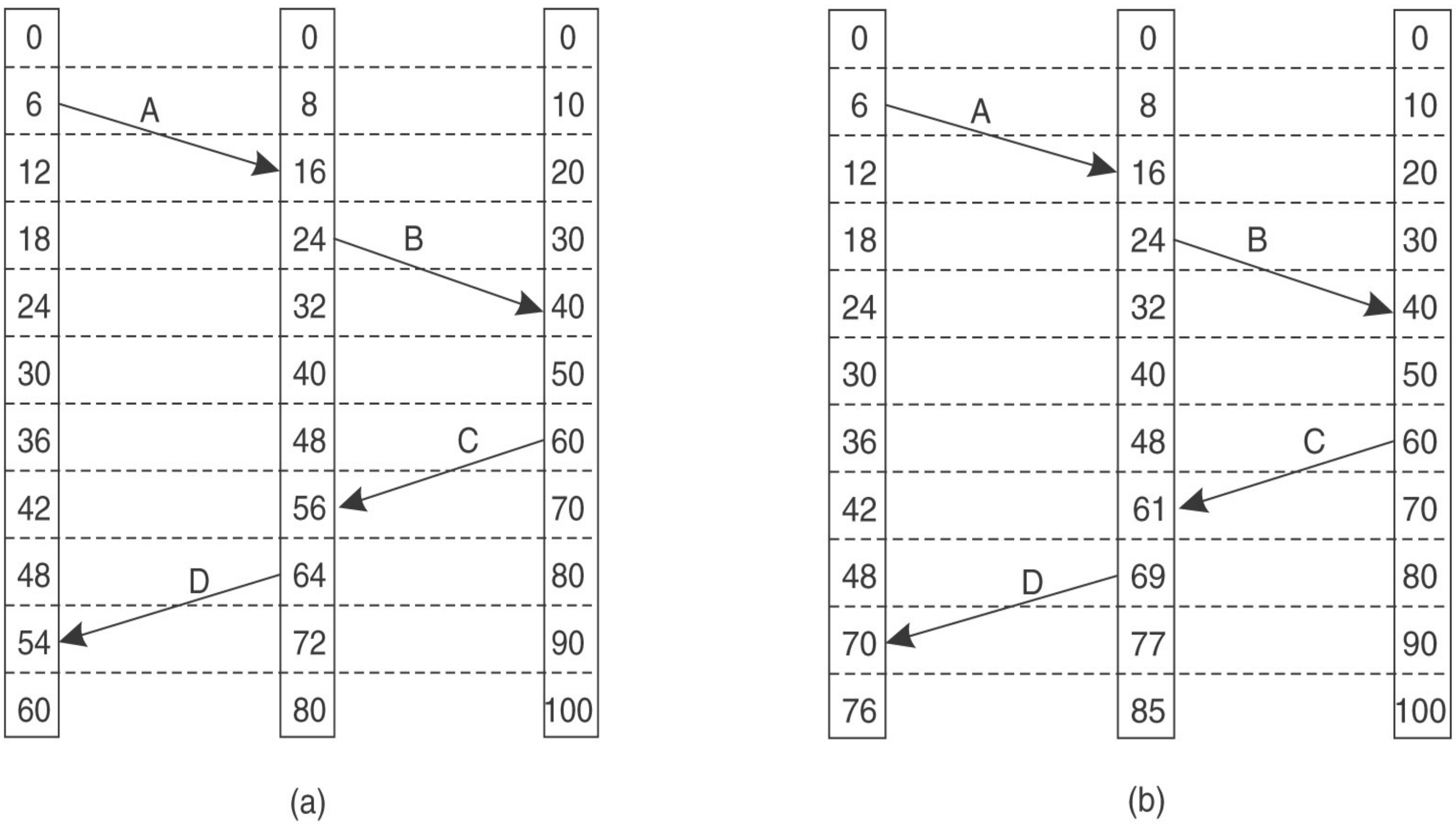
- This property ensures that causally related messages destined to the same destination are delivered in an order that is consistent with their causality relation.
- Causally ordered delivery of messages implies FIFO message delivery. (Note that CO $\subset$ FIFO $\subset$ Non-FIFO.)
- Causal ordering model considerably simplifies the design of distributed algorithms because it provides a built-in synchronization.

A Close Look



- ➔ $e \rightarrow e' \Rightarrow L(e) < L(e')$ but not vice versa
- ➔ Example: event b and event e

Correction of Clocks

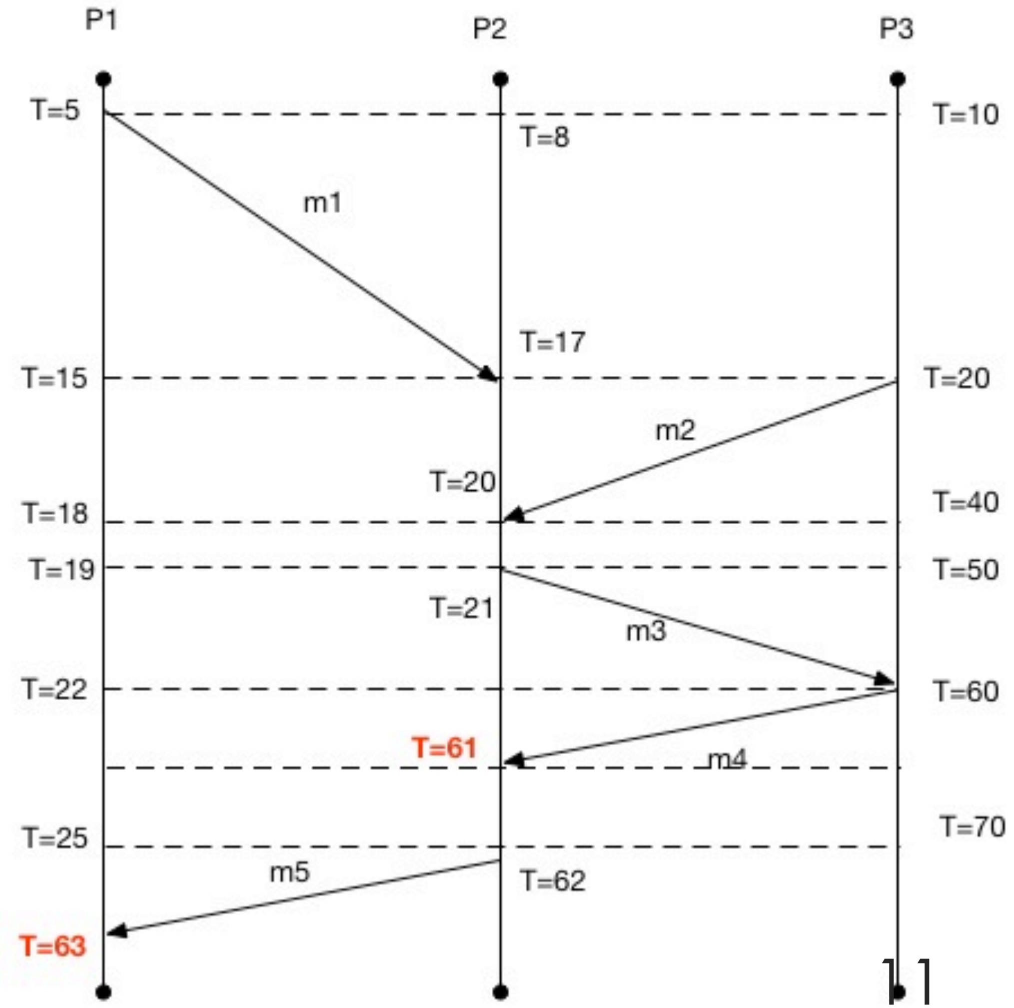


An Illustration

m1 –> m3
=> C(m1) < C(m3)

m2 –> m3
=> C(m2) < C(m3)

Here which event,
either m1 or m2,
caused
m3 to be sent?

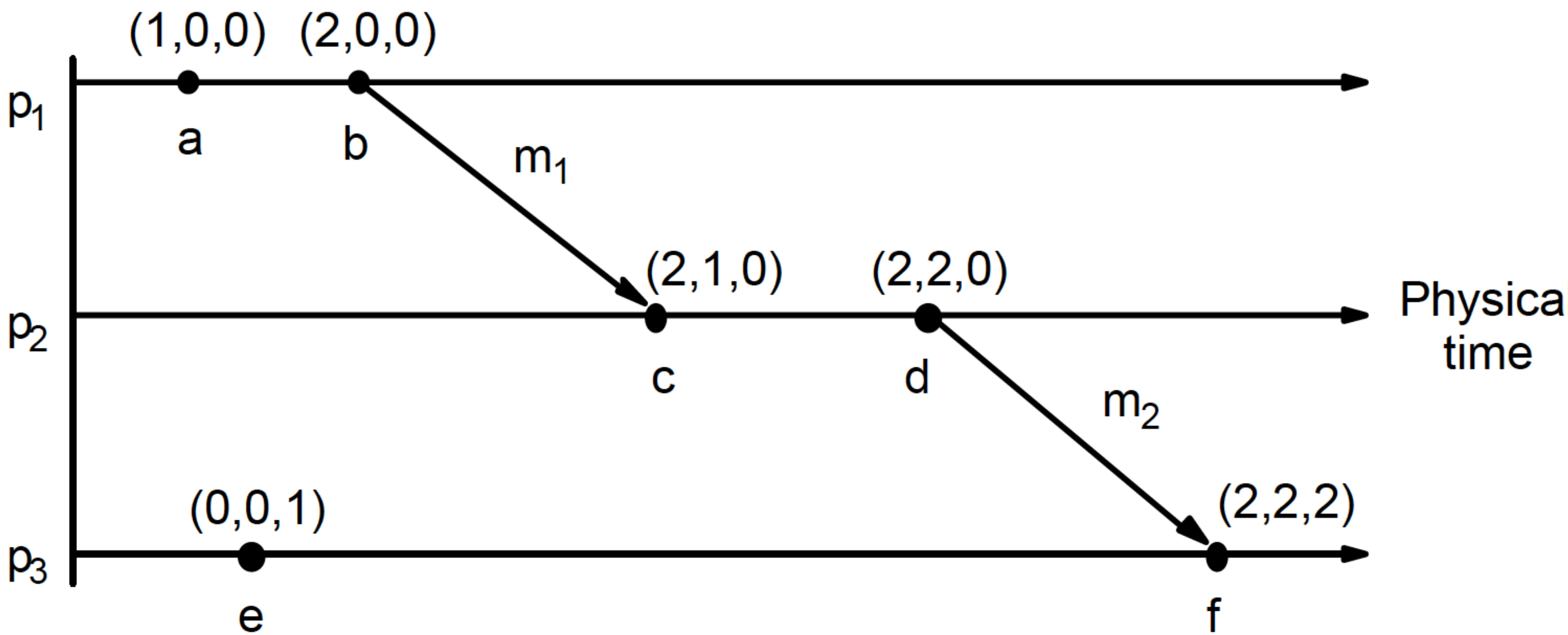


Rajendra, IIIT Sri City

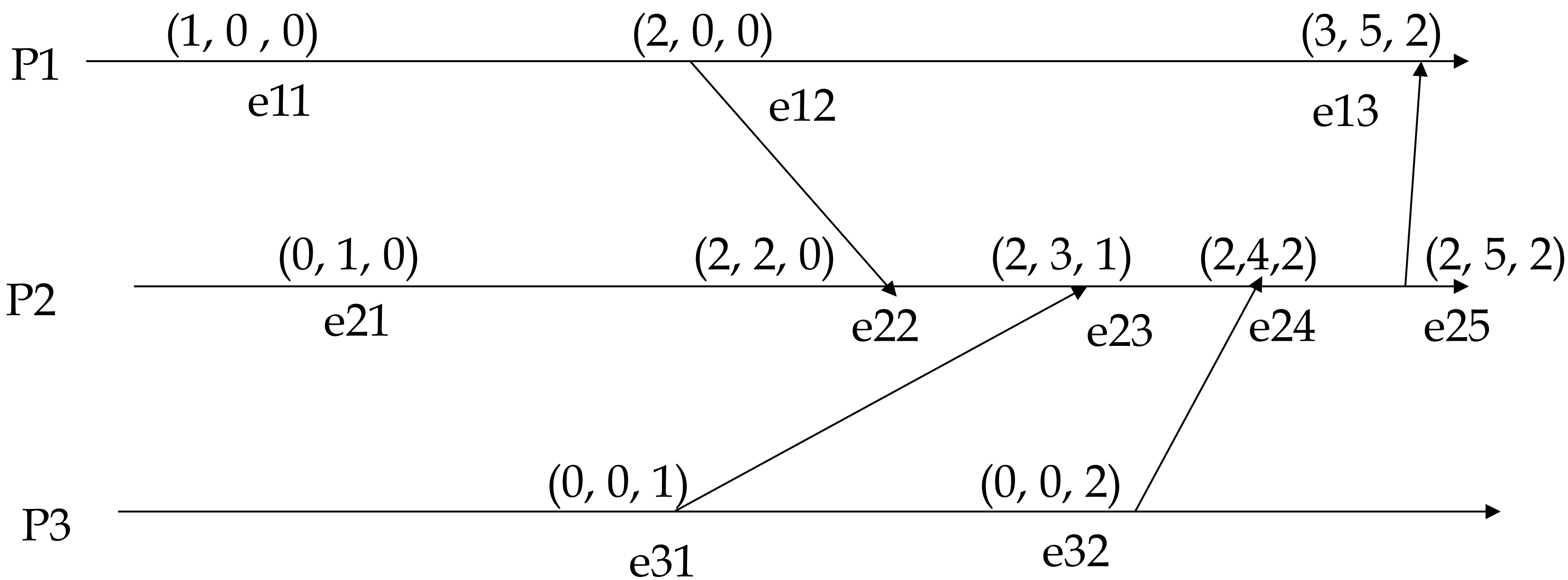
Vector Clocks

- 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



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 $e11 \rightarrow e22$

14

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

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:

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

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

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

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,

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

- 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

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

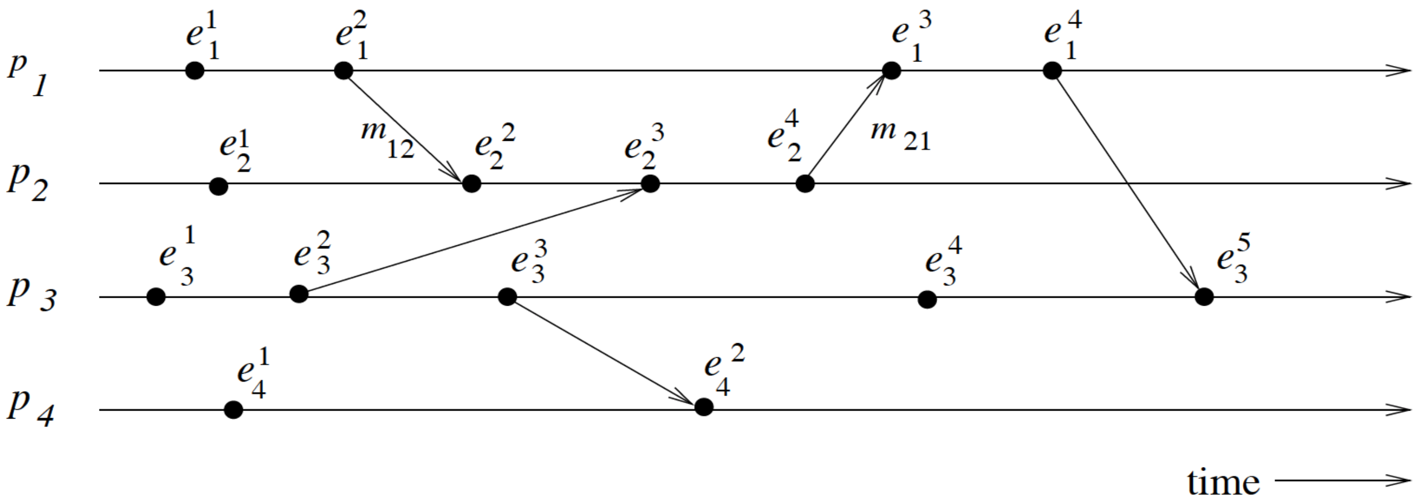
Where the global state is given by

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

- 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}.

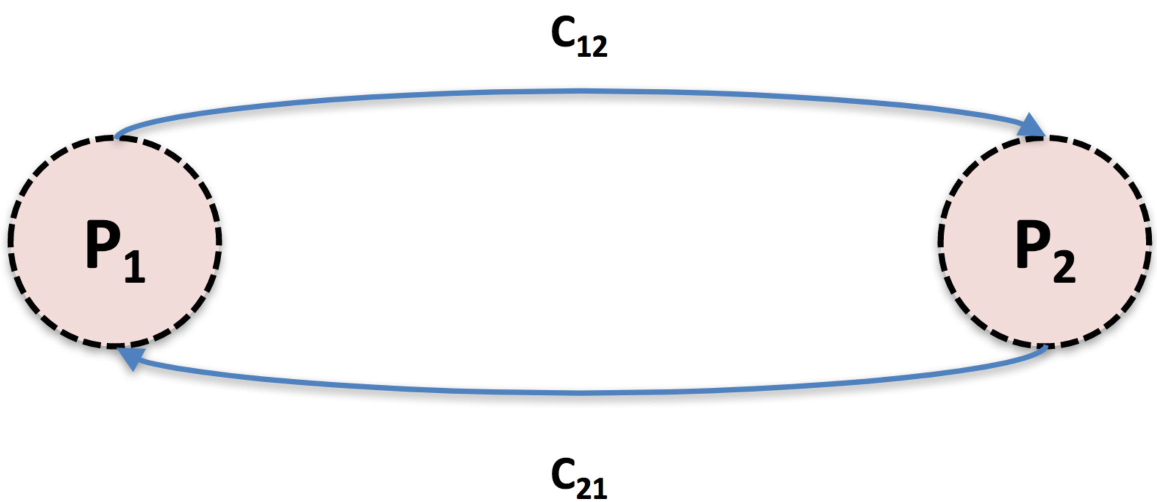
Changes in Global State?

- 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

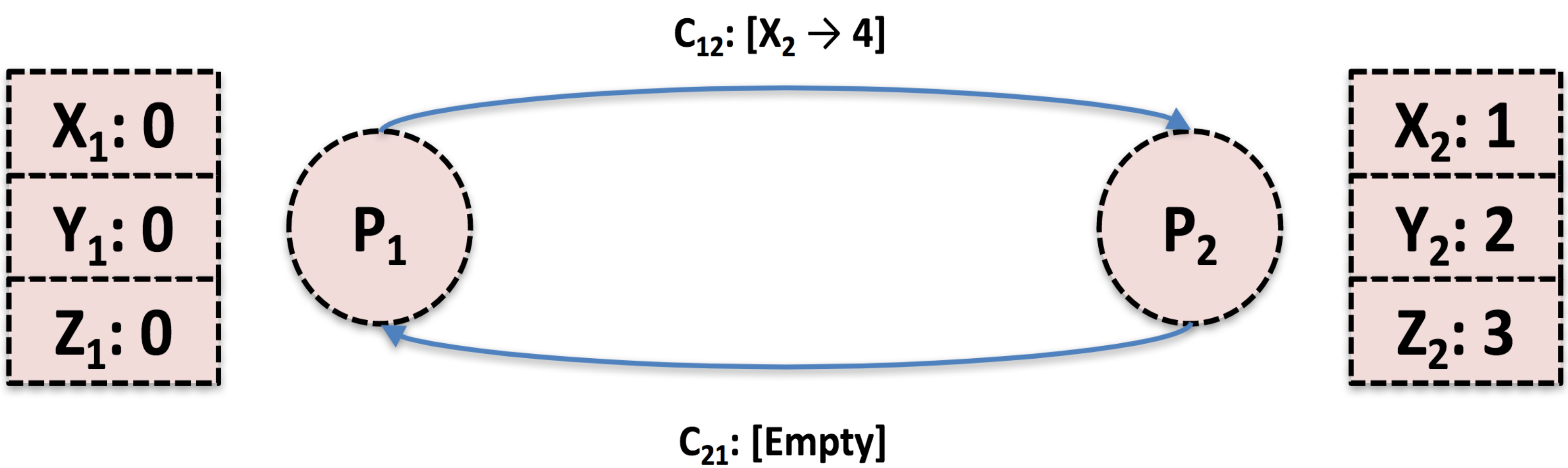
An Example Scenario

- 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



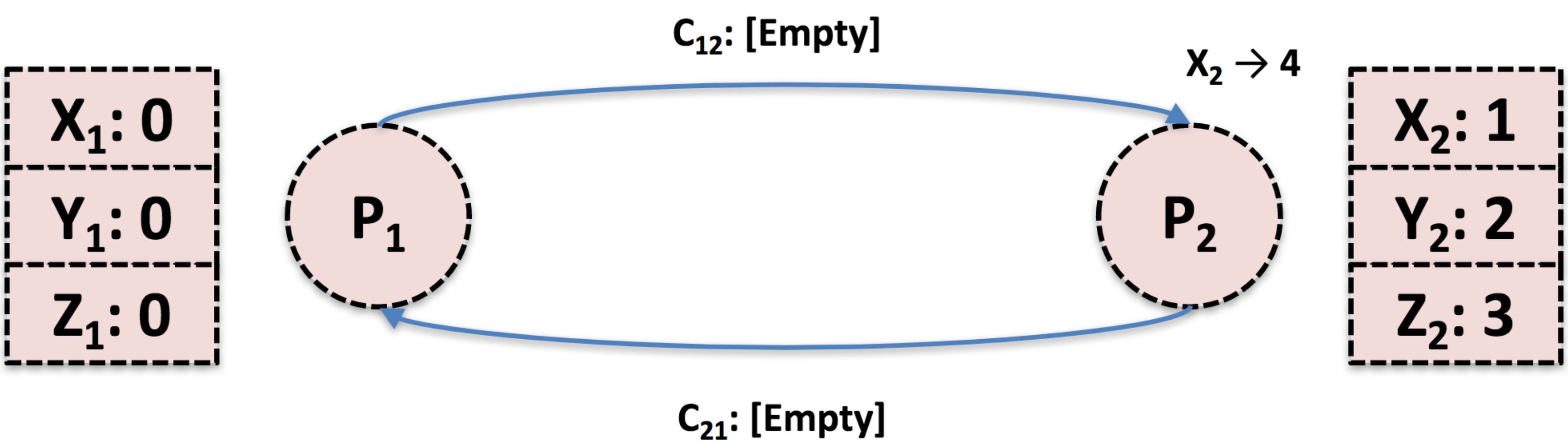
An Example (contd)

- P_1 tells P_2 to change its state variable, X_2, from 1 to 4



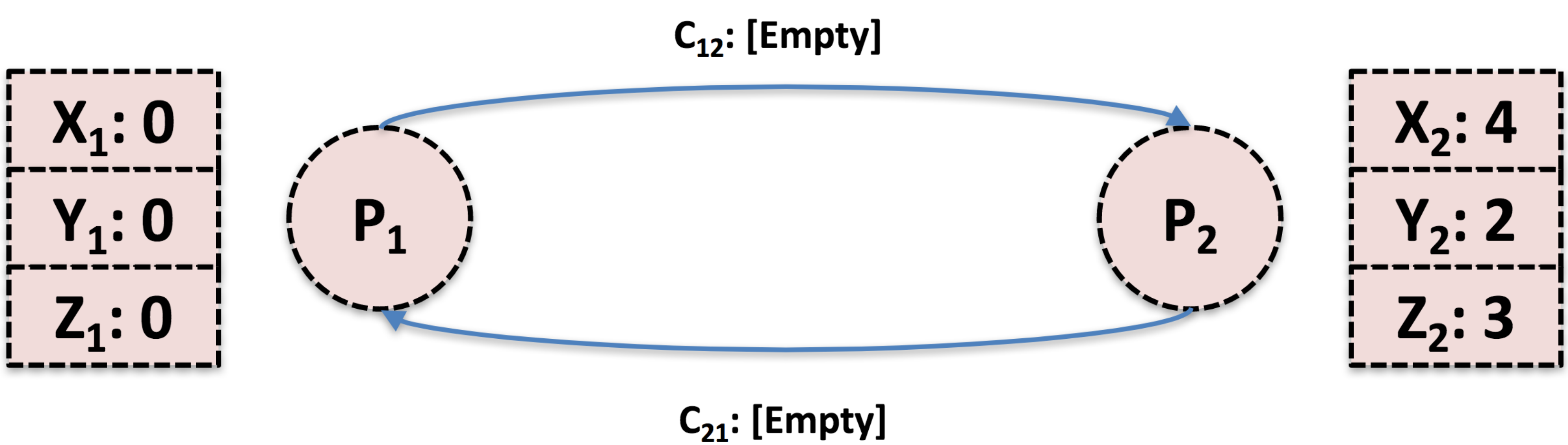
An Example (contd)

- 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 $N\text{-}1$ 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)

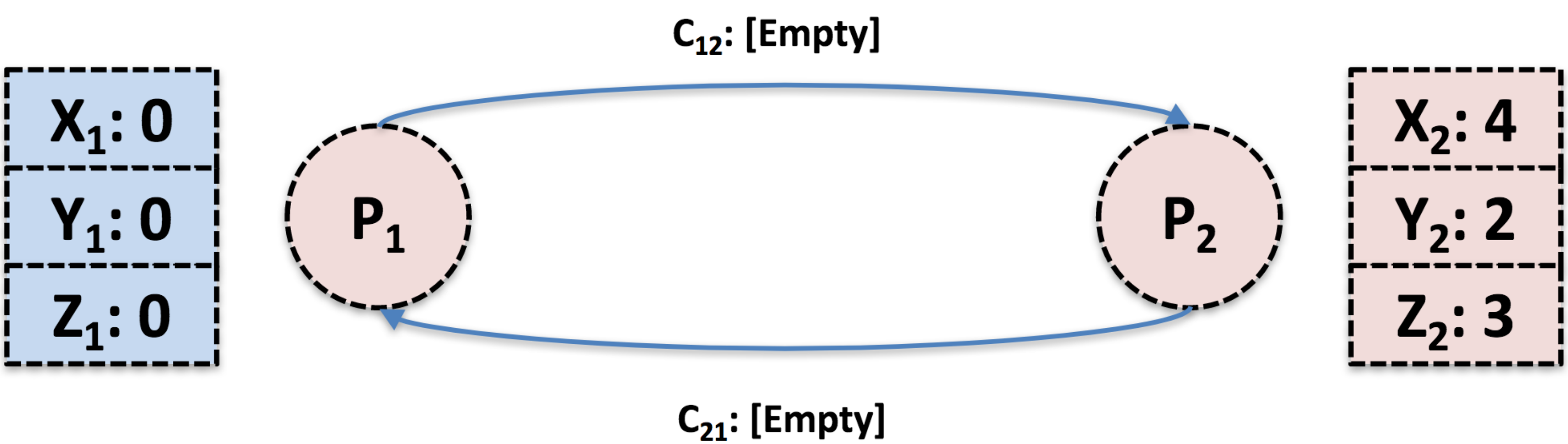
- All processes have received a marker on all N-1 incoming channels (and recorded their states)

- Later, a central server can gather the partial state to build a global snapshot

(Do we require this? – Not Really!!)

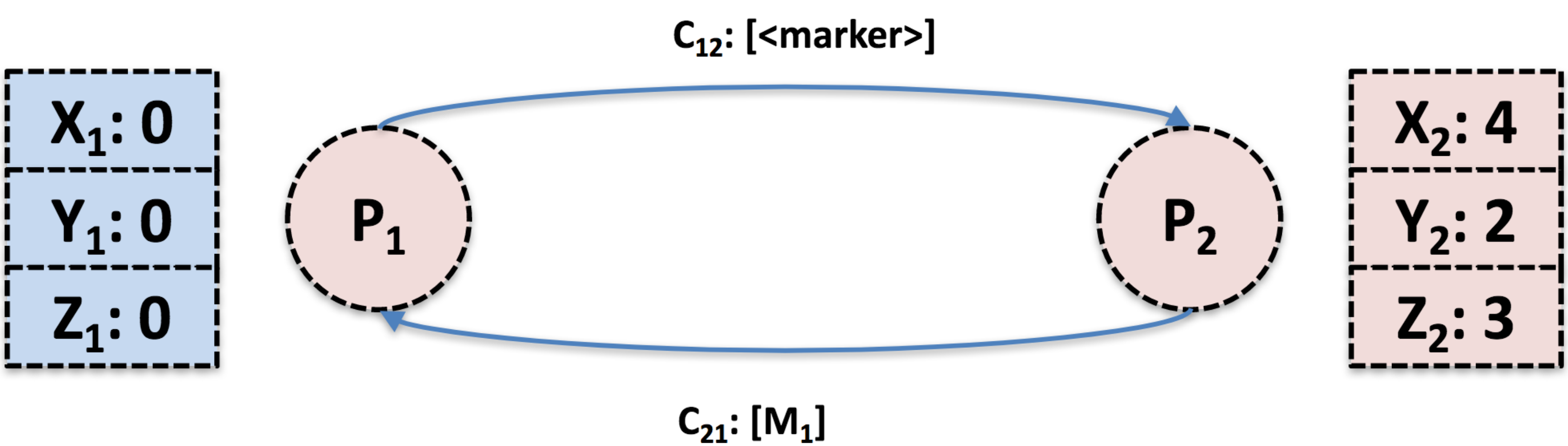
A Close Look of an Example

➔ Initially all channels are empty!



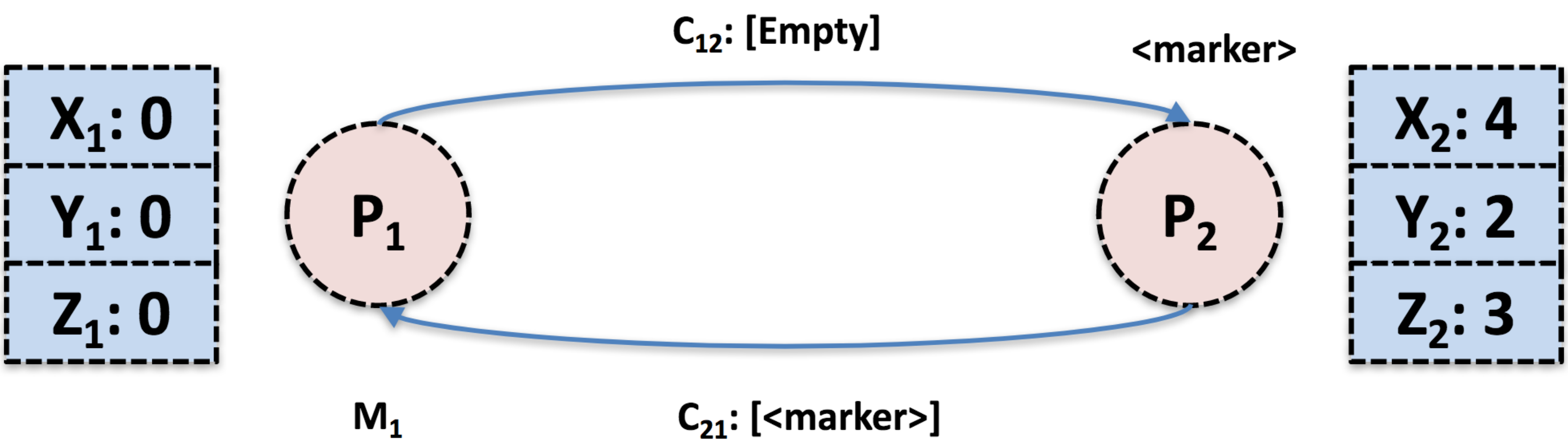
An Example (contd)

- 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



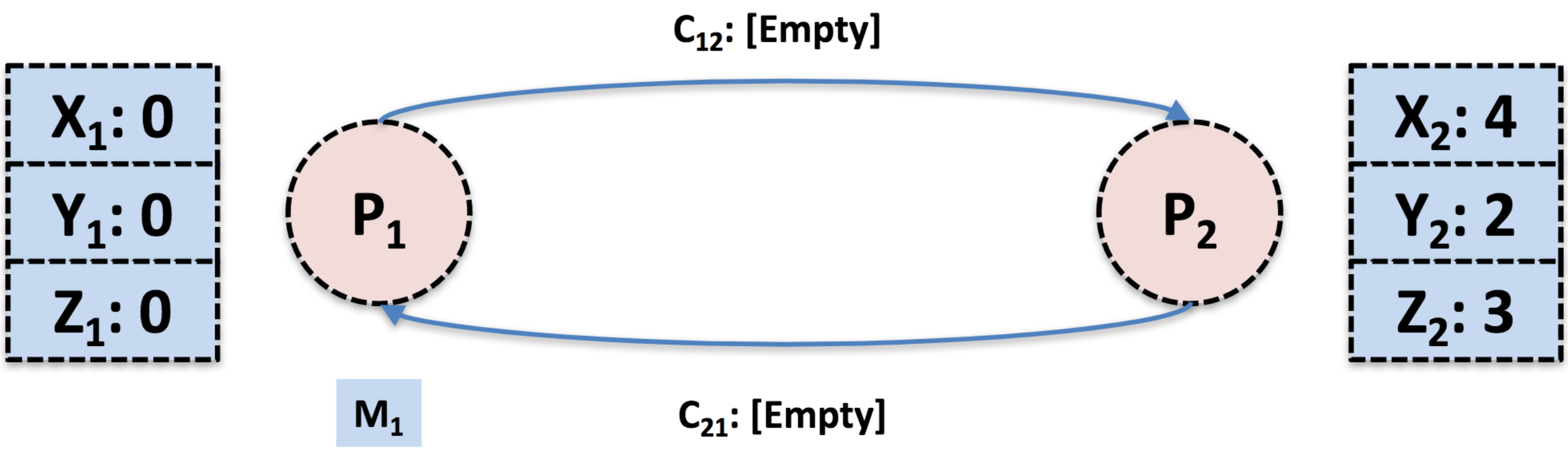
An Example (contd)

- P_2 receives a marker message for the first time, so records its state
- P_2 then sends a marker message to P_1



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



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 !!

How to reach me?

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

- Visit my homepage @

  - http://www.iiits.ac.in/FacPages/index-rajendra.html

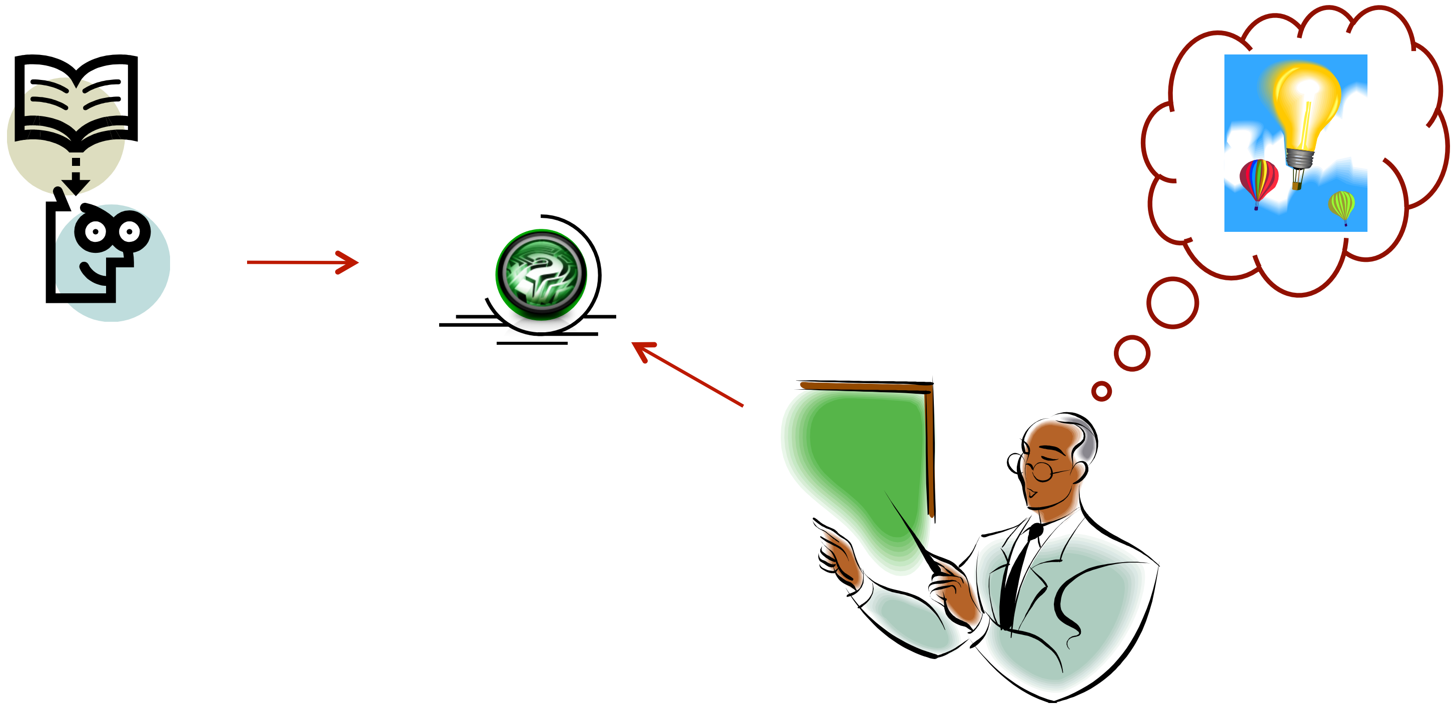
  OR

  - http://rajendra.2power3.com

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

Thanks …



… Questions ???