RAMAIAH
Institute of Technology

M.S. Ramaiah Institute of Technology
(Autonomous Institute, Affiliated to VTU)
Department of Computer Science and Engineering

Course Name: Distributed Systems
Course Code: CSE751
Credits: 3:0:0

Term: Oct-Feb 2022

Faculty:
Sini Anna Alex

Global State of a Distributed System

"A collection of the local states of its components, namely, the processes and the communication channels."

- The state of a process is defined by the contents of processor registers, stacks, local memory, etc. and depends on the local context of the distributed application.
- The state of channel is given by the set of messages in transit in the channel.
- The occurrence of events changes the states of respective processes and channels.
- An internal event changes the state of the process at which it occurs.
- A send event changes the state of the process that sends the message and the state of the channel on which the message is sent.
- A receive event changes the state of the process that or receives the message and the state of the channel on which the message is received.

Global State of a Distributed System

Notations

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

Global State of a Distributed System

A Channel State

- The state of a channel depends upon the states of the processes it connects.
- Let $SC_{ij}^{x,y}$ denote the state of a channel C_{ij}.

The state of a channel is defined as follows:

$$SC_{ij}^{x,y} = \{m_{ij} | \ send(m_{ij}) \leq e_i^x \bigwedge rec(m_{ij}) \not\leq e_j^y\}$$

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

Global State of a Distributed System

Global State

- The global state of a distributed system is a collection of the local states of the processes and the channels.
- Notationally, 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.
- This will be possible if the local clocks at processes were perfectly synchronized or if there were a global system clock that can be instantaneously read by the processes. (However, both are impossible.)

Cuts of a Distributed Computation

"In the space-time diagram of a distributed computation, a *cut* is a zigzag line joining one arbitrary point on each process line."

- A cut slices the space-time diagram, and thus the set of events in the distributed computation, into a PAST and a FUTURE.
- The PAST contains all the events to the left of the cut and the FUTURE contains all the events to the right of the cut.
- For a cut C, let PAST(C) and FUTURE(C) denote the set of events in the PAST and FUTURE of C, respectively.
- Every cut corresponds to a global state and every global state can be graphically represented as a cut in the computation's space-time diagram.
- Cuts in a space-time diagram provide a powerful graphical aid in representing and reasoning about global states of a computation.

Cuts of a Distributed Computation

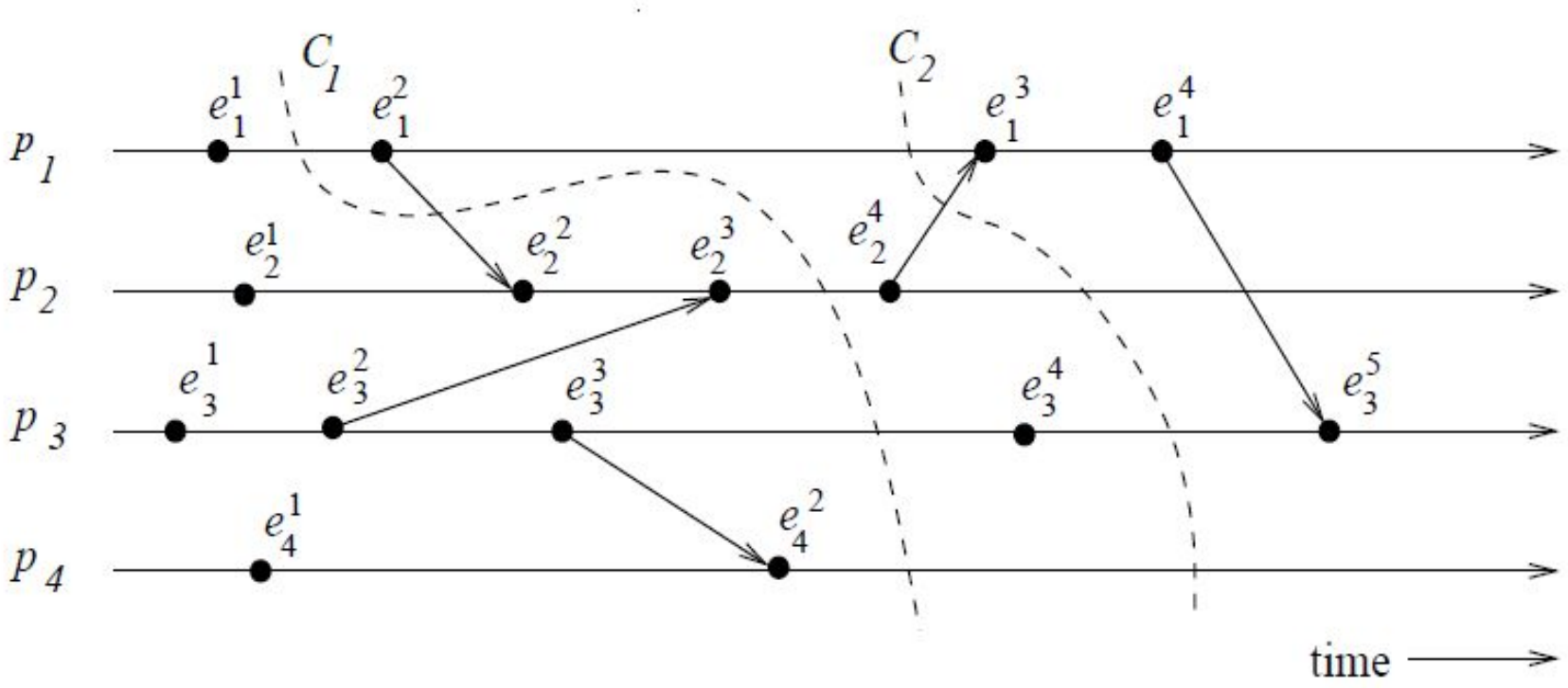
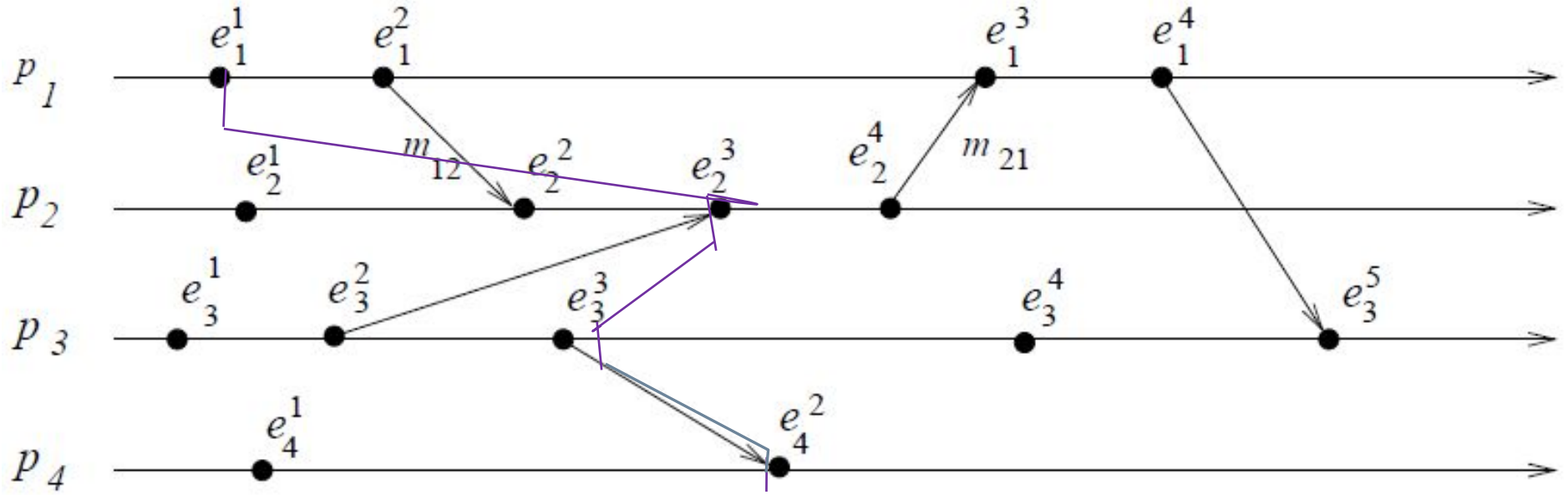


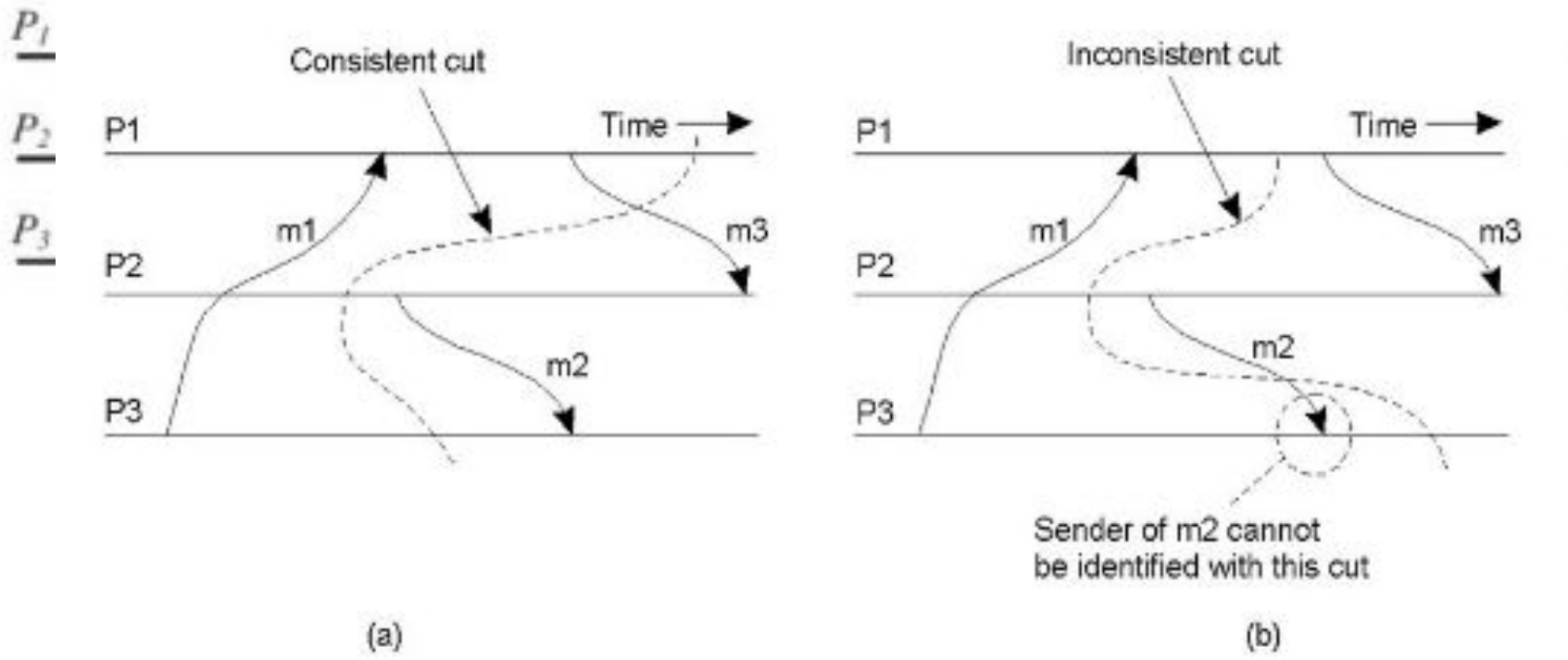
Illustration of cuts in a distributed execution

Global State of a Distributed System



- A global state $GS_1 = \{LS_1^1, LS_2^3, LS_3^3, LS_4^2\}$ 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.
- A global state GS_2 consisting of local 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}.

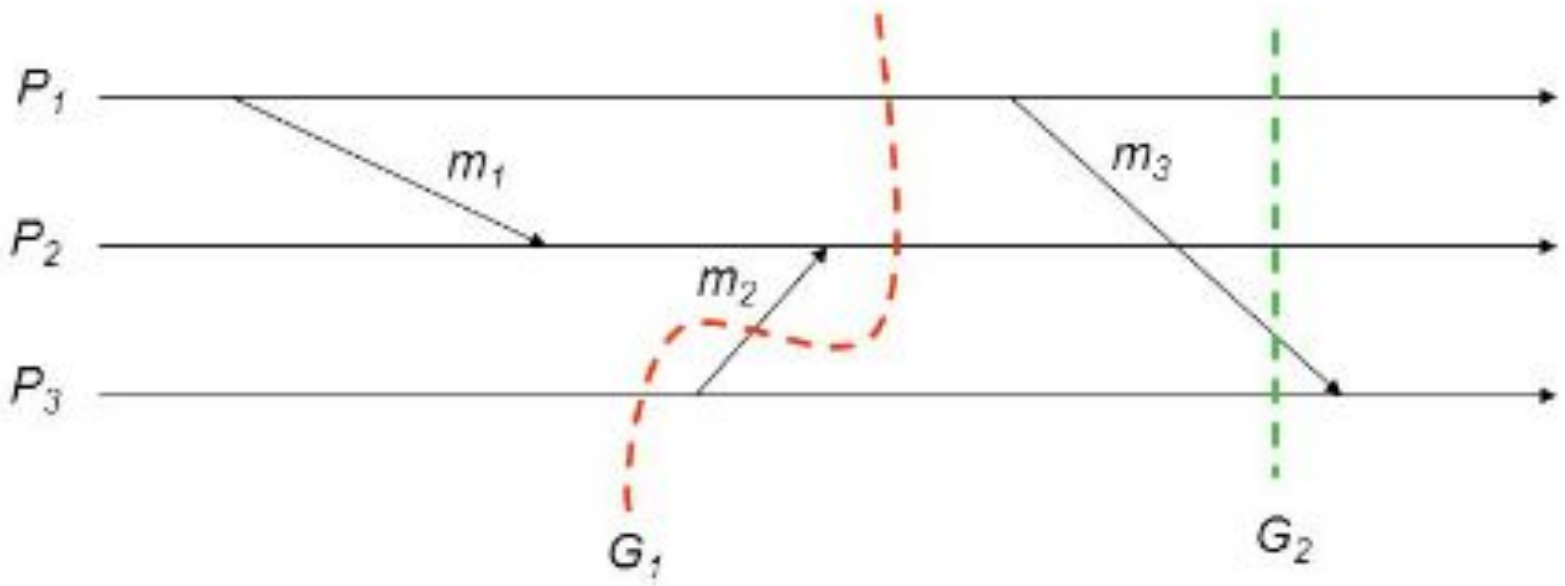
Analyze the cuts in distributed computation are consistent or inconsistent



a) A consistent cut

b) An inconsistent cut

Analyze the cuts in distributed computation are consistent or inconsistent



- G_1 is not consistent
- G_2 is consistent but m_3 must be recorded

Chapter 3: Logical Time

Implementing Logical Clocks

- Implementation of logical clocks requires addressing two issues: data structures local to every process to represent logical time and a protocol to update the data structures to ensure the consistency condition.
- Each process p_i maintains data structures that allow it the following two capabilities:
  - A *local logical clock*, denoted by lc_i, that helps process p_i measure its own progress.
  - A *logical global clock*, denoted by gc_i, that is a representation of process p_i's local view of the logical global time. Typically, lc_i is a part of gc_i.

Implementing Logical Clocks

The protocol ensures that a process's logical clock, and thus its view of the global time, is managed consistently. The protocol consists of the following two rules:

- *R1*: This rule governs how the local logical clock is updated by a process when it executes an event.
- *R2*: This rule governs how a process updates its global logical clock to update its view of the global time and global progress.
- Systems of logical clocks differ in their representation of logical time and also in the protocol to update the logical clocks.

Scalar Time

- Proposed by Lamport in 1978 as an attempt to totally order events in a distributed system.
- Time domain is the set of non-negative integers.
- The logical local clock of a process p_i and its local view of the global time are squashed into one integer variable C_i.
- Rules *R1* and *R2* to update the clocks are as follows:
- *R1*: Before executing an event (send, receive, or internal), process p_i executes the following:

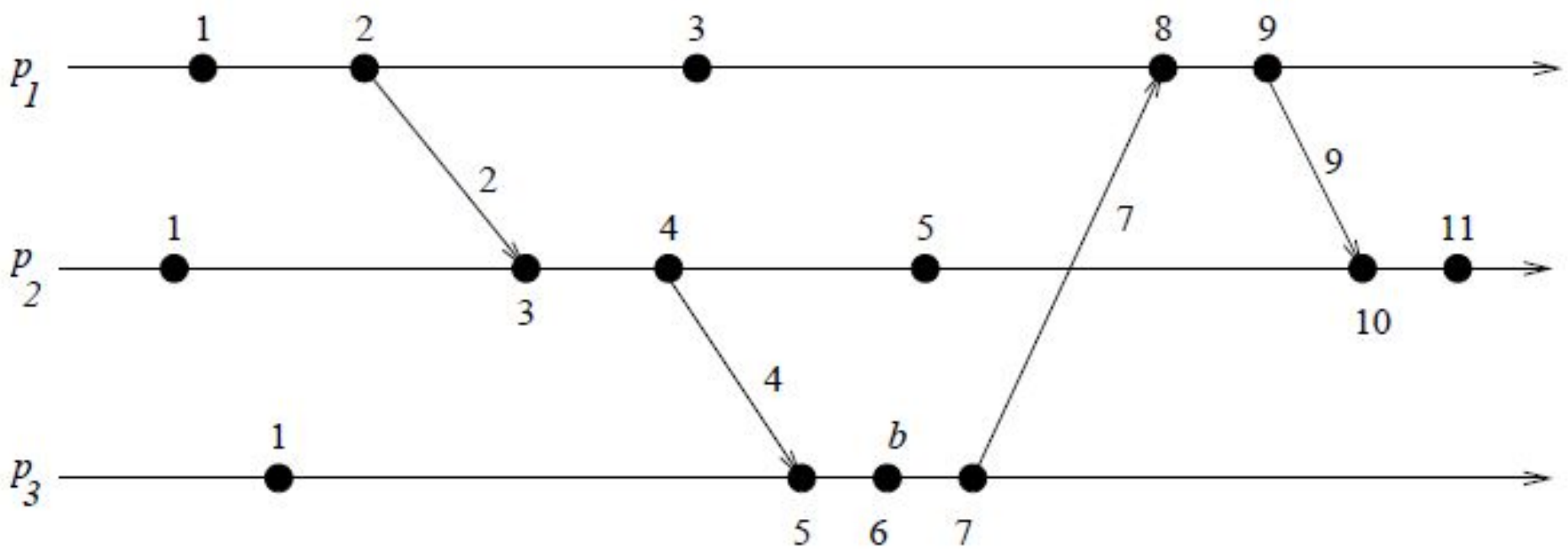
$$C_i := C_i + d \qquad (d > 0)$$

In general, every time *R1* is executed, d can have a different value; however, typically d is kept at 1.

Scalar Time

- *R2*: Each message piggybacks the clock value of its sender at sending time. When a process p_i receives a message with timestamp C_{msg}, it executes the following actions:
  - $C_i := max(C_i, C_{msg})$
  - Execute *R1*.
  - Deliver the message.

Evolution of scalar time



The space-time diagram of a distributed execution

Vector Time

- The system of vector clocks was developed independently by Fidge, Mattern and Schmuck.
- In the system of vector clocks, the time domain is represented by a set of n-dimensional non-negative integer vectors.
- Each process p_i maintains a vector $vt_i[1..n]$, where $vt_i[i]$ is the local logical clock of p_i and describes the logical time progress at process p_i.
- $vt_i[j]$ represents process p_i's latest knowledge of process p_j local time.
- If $vt_i[j]=x$, then process p_i knows that local time at process p_j has progressed till x.
- The entire vector vt_i constitutes p_i's view of the global logical time and is used to timestamp events.

Vector Time

Process p_i uses the following two rules *R1* and *R2* to update its clock:

- *R1*: Before executing an event, process p_i updates its local logical time as follows:

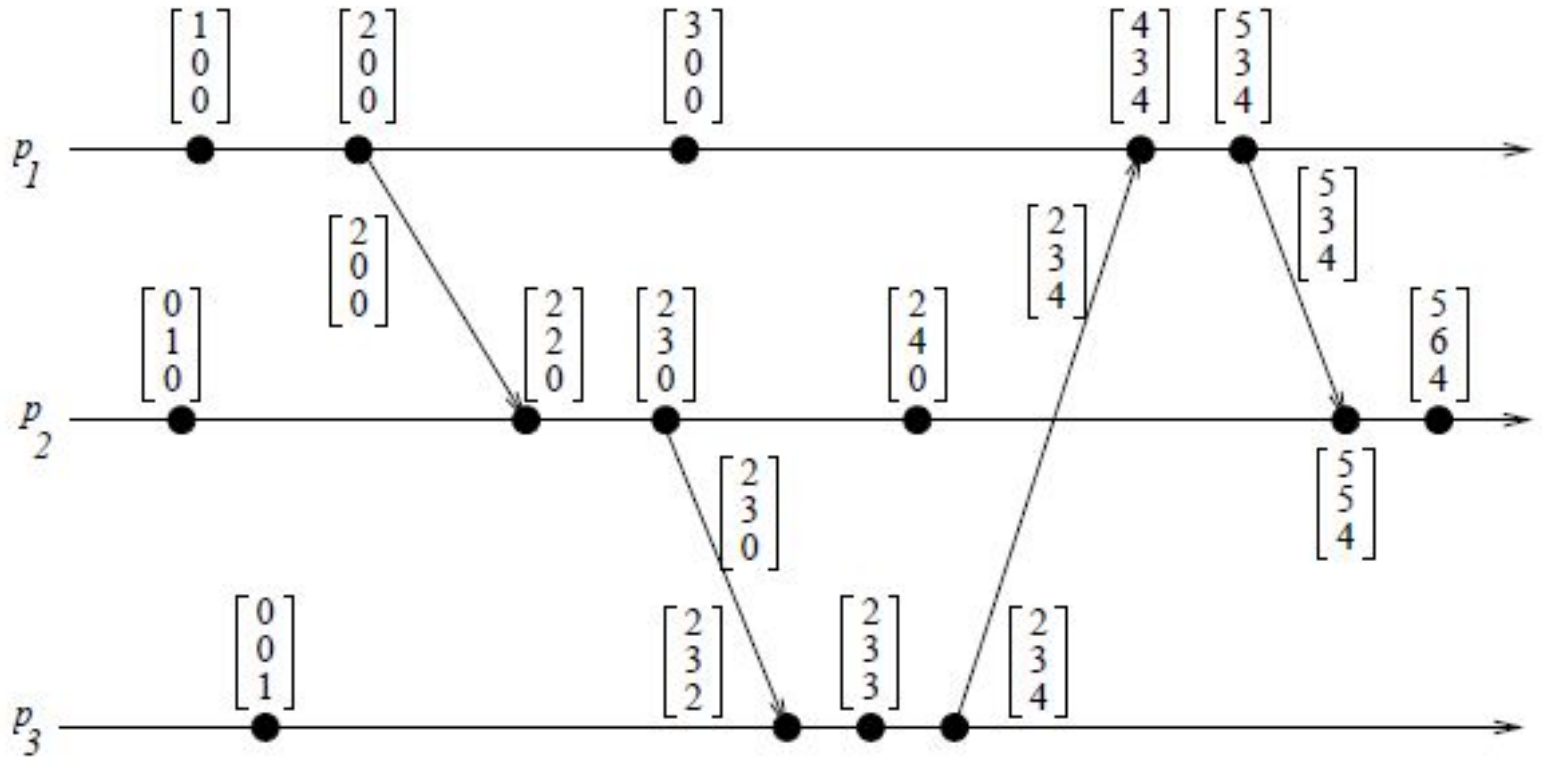
$$vt_i[i] := vt_i[i] + d \qquad (d > 0)$$

- *R2*: Each message m is piggybacked with the vector clock vt of the sender process at sending time. On the receipt of such a message (m,vt), process p_i executes the following sequence of actions:
  - Update its global logical time as follows:

$$1 \leq k \leq n \ : \ vt_i[k] := max(vt_i[k], vt[k])$$

  - Execute *R1*.
  - Deliver the message m.

Evolution of vector time



The timestamp of an event is the value of the vector clock of its process when the event is executed.

Initially, a vector clock is [0, 0, 0, ..., 0].

vector clocks progress with the increment value d=1.

Efficient Implementations of Vector Clocks

- If the number of processes in a distributed computation is large, then vector clocks will require piggybacking of huge amount of information in messages.
- The message overhead grows linearly with the number of processors in the system and when there are thousands of processors in the system, the message size becomes huge even if there are only a few events occurring in few processors.

Singhal–Kshemkalyani's differential technique

- *Singhal–Kshemkalyani's differential technique* is based on the observation that between successive message sends to the same process, only a few entries of the vector clock at the sender process are likely to change. This is more likely when the number of processes is large because only a few of them will interact frequently by passing messages.

- In this technique, when a process pi sends a message to a process pj , it piggybacks only those entries of its vector clock that differ since the last message sent to pj .

Singhal–Kshemkalyani's differential technique

When p_j receives this message, it updates its vector clock as follows:

$$vt_i[i_k] = max(vt_i[i_k], v_k) \text{ for } k = 1, 2, \ldots, n_1.$$

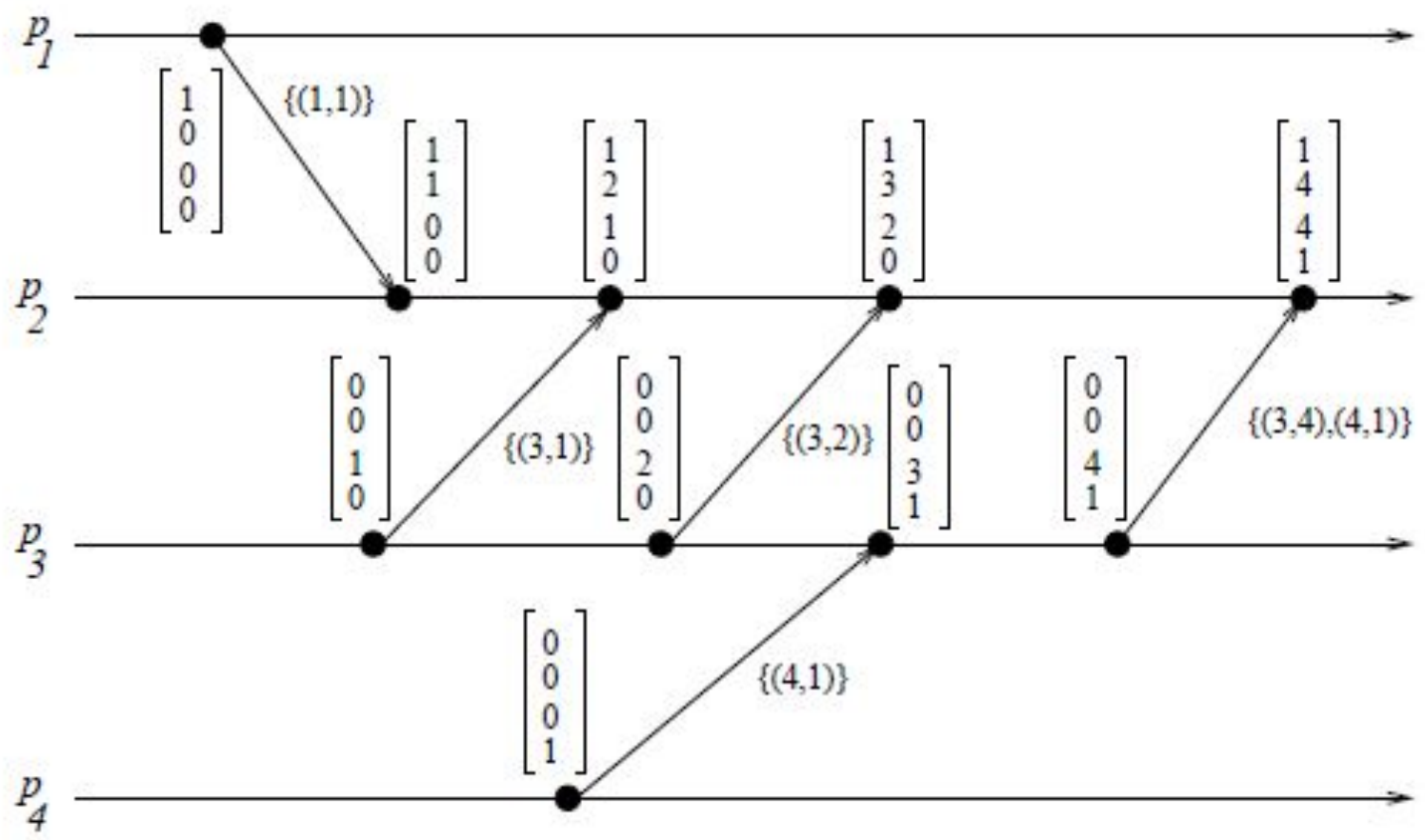
- Thus this technique cuts down the message size, communication bandwidth and buffer (to store messages) requirements.
- In the worst of case, every element of the vector clock has been updated at p_i since the last message to process p_j, and the next message from p_i to p_j will need to carry the entire vector timestamp of size n.
- However, on the average the size of the timestamp on a message will be less than n.

Singhal–Kshemkalyani's differential technique

- Implementation of this technique requires each process to remember the vector timestamp in the message last sent to every other process.
- Direct implementation of this will result in $O(n^2)$ storage overhead at each process.
- Singhal and Kshemkalyani developed a clever technique that cuts down this storage overhead at each process to $O(n)$. The technique works in the following manner:
- Process p_i maintains the following two additional vectors:
  - $LS_i[1..n]$ ('Last Sent'):
    $LS_i[j]$ indicates the value of $vt_i[i]$ when process p_i last sent a message to process p_j.
  - $LU_i[1..n]$ ('Last Update'):
    $LU_i[j]$ indicates the value of $vt_i[i]$ when process p_i last updated the entry $vt_i[j]$.

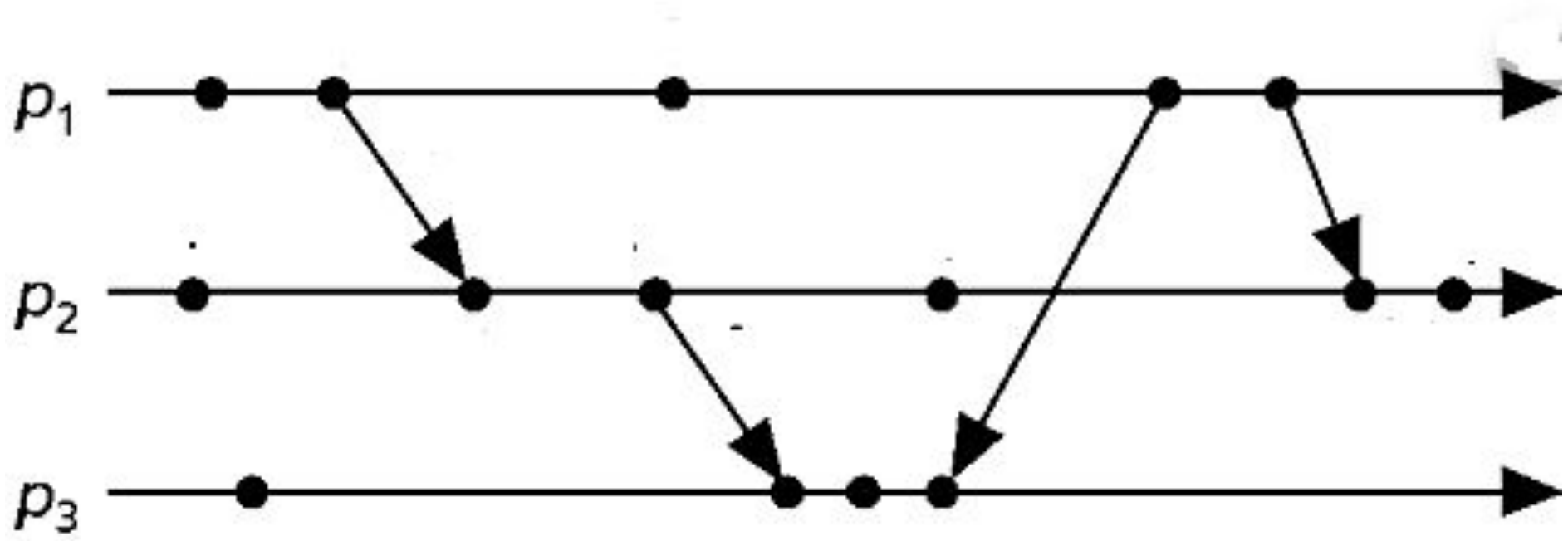
RAMAIAH Institute of Technology

Vector clocks progress in Singhal-Kshemkalyani technique



RAMAIAH Institute of Technology

Implement Singhal–Kshemkalyani's differential technique



Thank you