Causal Ordering in Distributed Computing

Course: Distributed Computing

Faculty: Dr. Rajendra Prasath

Spring 2019

About this Course

This course covers essential aspects that every serious programmer needs to know about **Causal Ordering in Distributed Systems and the related concepts**

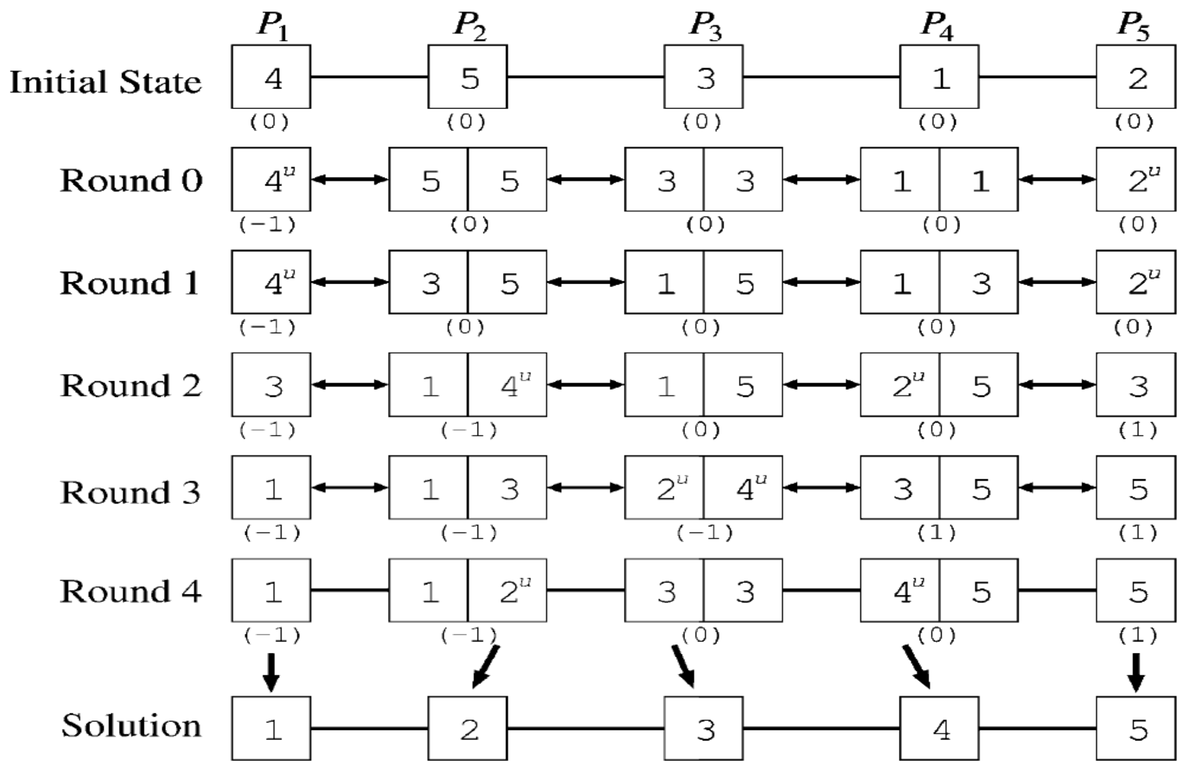
RECAP

What did you learn so far?

- Goals / Challenges with Distributed Systems
- Message Passing systems
  - Basic Primitive operations
  - 3 types of EVENTs: Internal, send and receive
  - States of a process and Channel
- Distributed Sorting
  - Odd – Even Transposition Sort
  - Sasaki's (n-1) rounds algorithms
  - Did you try (n-2) rounds algorithm for distributed sorting on line network?
  - Implementing Discrete Events Simulation

Implementation - How is it going?

[Sasaki, 2002]



No knowledge about the Global position;
Make copies of elements at intermediate nodes;
Rule to select the final Solution; Computing n at runtime

Causal Ordering

A Model of Distributed Computations

Focused Topics:

- Causal Precedence Relation
- Models of Communication Networks
- Causal Ordering
- Global State and
- Cuts of a DS
  - PAST and FUTURE events

A Distributed Program

- A distributed program is composed of a set of n asynchronous processes, $p_1, p_2, \ldots, p_i, \ldots, p_n$
- The processes do not share a global memory and communicate solely by passing messages
- The processes do not share a global clock that is instantaneously accessible to these processes
- Process execution and message transfer are asynchronous
- Without loss of generality, we assume that each process is running on a different processor
- Let c_{ij} denote the channel from process p_i to p_j and let m_{ij} denote a message sent by p_i to p_j
- The message transmission delay is finite and unpredictable

A Model of Distributed Executions

- The execution of a process consists of a sequential execution of its actions.
- The actions are atomic and modeled as three types of events: internal events, message send events, and message receive events
- Let e_i^x denote the x^{th} event at process p_i .
- For a message m, let send(m) and receive(m) denote send and receive events, respectively.
- The occurrence of events changes the states of respective processes and channels.
- Internal event → changes state of the process
- Send and Receive events change the state of the process that sends / receives the message & the state of the channel on which the message is sent / received respectively

A Model of Distributed Executions

- 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 H_i where

$$H_i = (h_i , \rightarrow i)$$

h_i is the set of events produced by p_i and binary relation $\rightarrow i$ defines a linear order on these events

- **Linear Relation:** Mathematically, the independent variable is multiplied by the slope coefficient, added by a constant, which determines the dependent variable
- Relation $\rightarrow i$ expresses causal dependencies among the events of p_i

A Model of Distributed Executions (contd.)

- 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
- Define a relation $\rightarrow_{msg}$ that captures the causal dependency due to message exchanges as follows:

  For every message m that is exchanged between two processes, we have

$$send(m) \rightarrow_{msg} receive(m)$$

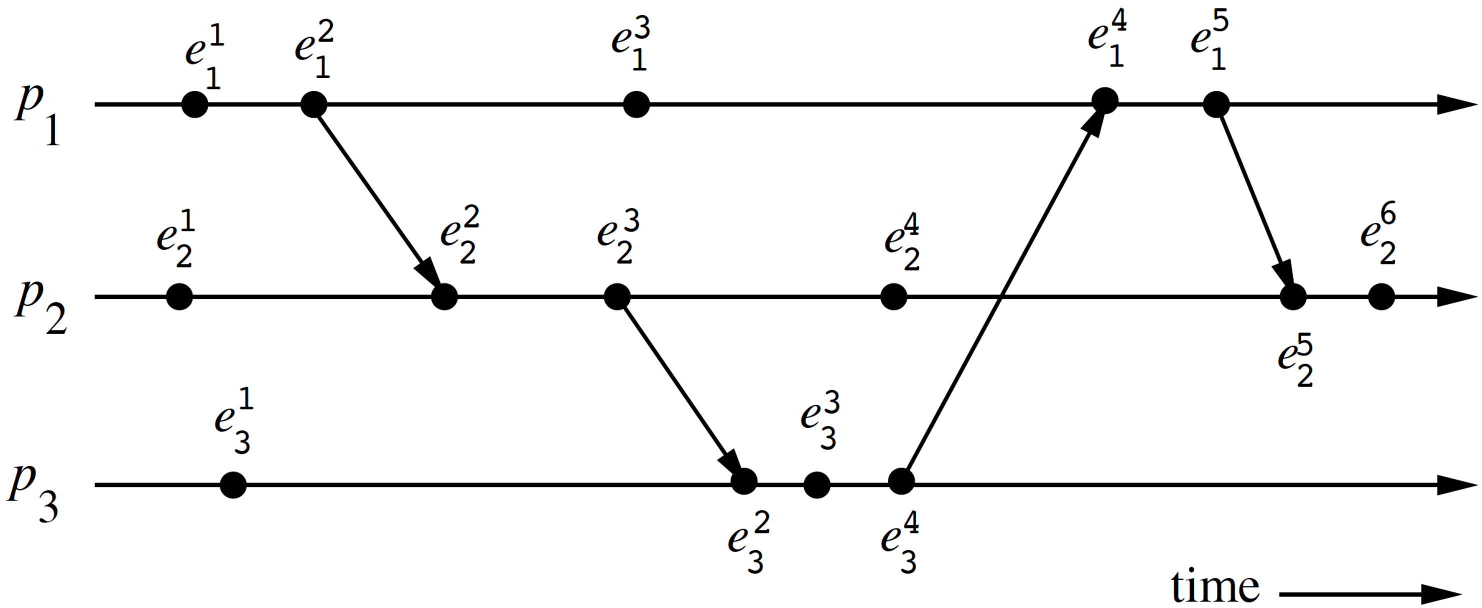
- Relation $\rightarrow_{msg}$ defines causal dependencies between the pairs of corresponding send and receive events

A State-Time diagram

- The evolution of a distributed execution is depicted by a space-time diagram
- A horizontal line represents the progress of a specific process
- A dot indicates an event
- A slant arrow indicates a message transfer

- Since an event execution is atomic (indivisible and instantaneous), it is justified to denote it as a dot on a process line

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

Partial Order / 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. Antisymmetry: $a \leq b$ 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 order
Addition of trichotomy law defines a total order $H=(H, \rightarrow)$

Causal Precedence Relation

- The execution of a distributed app. results in a set of distributed events
- Let $H = \bigcup_i h_i$ denote the set of events executed in a distributed computation.
- Define a binary relation $\rightarrow$ on the set H that expresses causal dependencies between events in the distributed execution.

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

- The causal precedence relation induces an irreflexive partial order on the events of a distributed computation that is *denoted* as $H=(H, \rightarrow)$

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

  $$e_i \not\rightarrow e_j \textit{ does not imply } e_j \not\rightarrow e_i$$
- 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 $e_i \parallel e_j$)

Example:

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

- The relation $\parallel$ is not transitive;

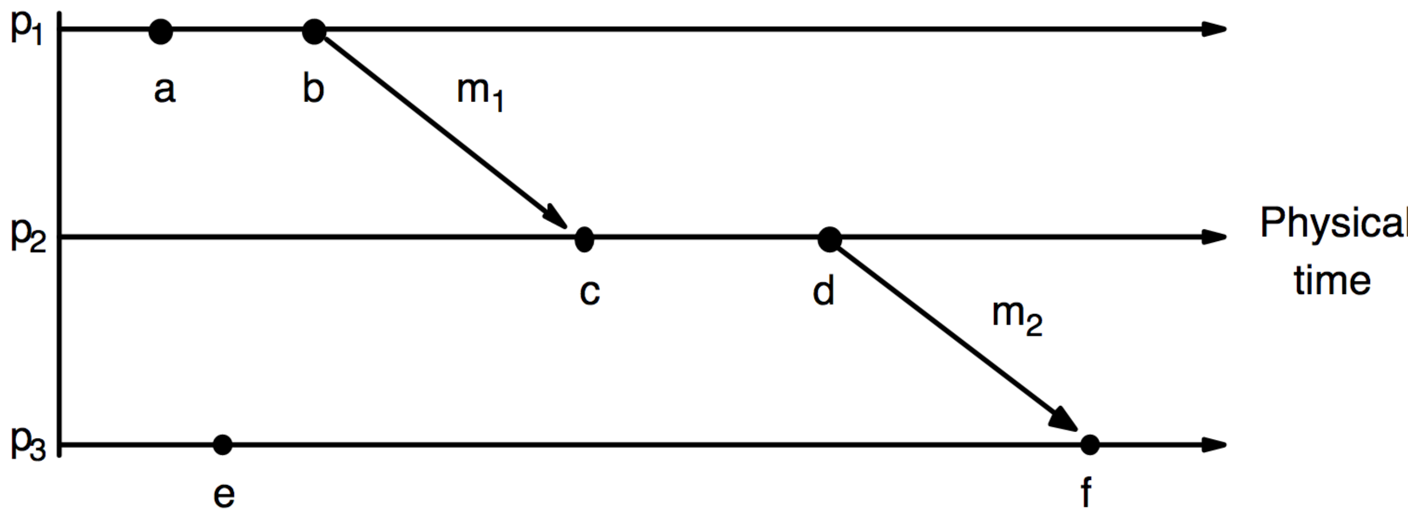
  that is,

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

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

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

Concurrency – An Example



$a \rightarrow b$ (at $p1$) $c \rightarrow d$ (at $p2$)

$b \rightarrow c$ ($m1$)

also $d \rightarrow f$ ($m2$)

Not all events are related by $\rightarrow$, e.g., a $\not\rightarrow$ e and e $\not\rightarrow$ a
they are said to be concurrent; write as $a \parallel e$

Logical vs. Physical concurrency

- Two events are logically concurrent if and only if they do not causally affect each other.
- In physical concurrency: events occur at the same instant in physical time.
- Two+ events may be logically concurrent even though they do not occur at same instant in physical time.
- If processor speed and message delays would have been different, the execution of these events could have very well coincided in physical time.
- Whether a set of logically concurrent events coincide in the physical time or not, does not change the outcome of the computation.
- A set of logically concurrent events may not have occurred at the same instant in physical time, we can assume that these events occurred at the same instant in physical time.

Models of Communication networks

- There are several models of the service provided by communication networks:

  FIFO, Non-FIFO, and causal ordering
- In the FIFO model, each channel acts as a first-in first-out message queue and thus, message ordering is preserved by a channel.
- In the 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.

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.

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, etc
- 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 pi 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,

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

- 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 !!):
  - the 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

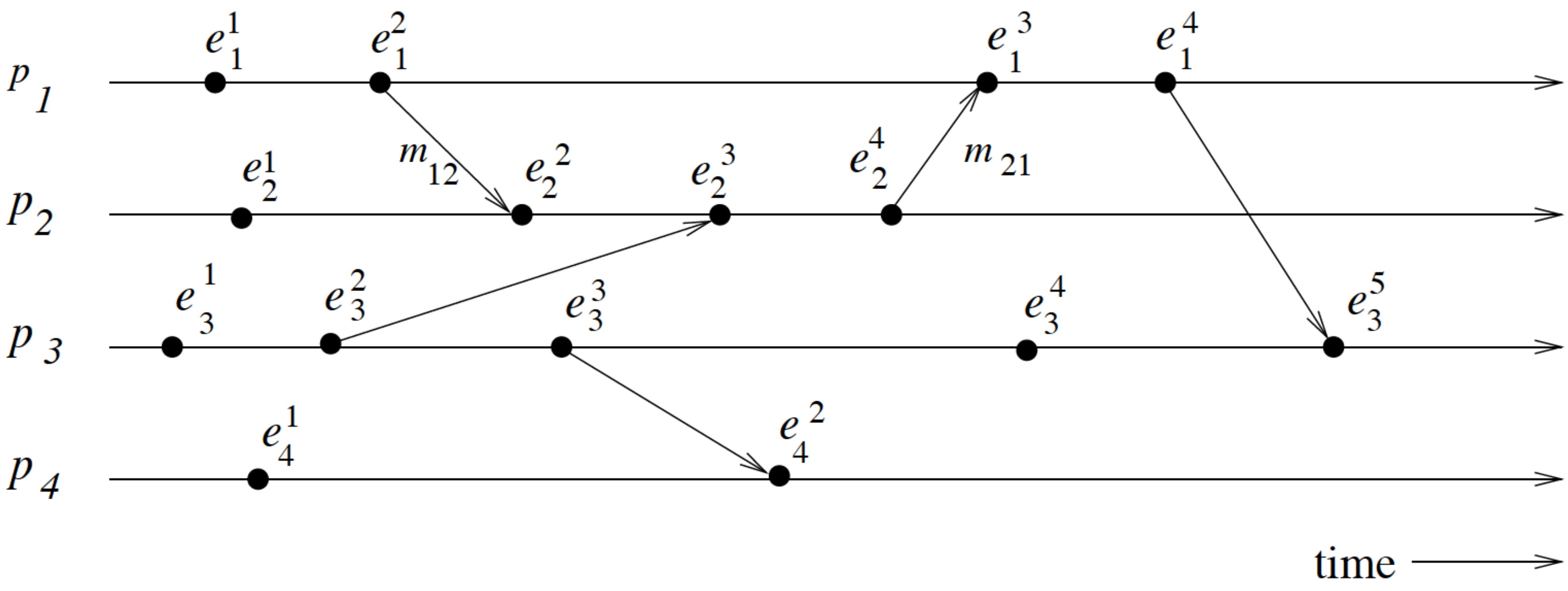
$$\forall m_{ij} : \; send(m_{ij}) \not\preceq LS_i^{x_i} \;\Leftrightarrow\; m_{ij} \notin SC_{ij}^{x_i, y_j} \bigwedge rec(m_{ij}) \not\preceq LS_j^{y_j}$$

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 pi sent after executing event

Consistent Global State – An Example

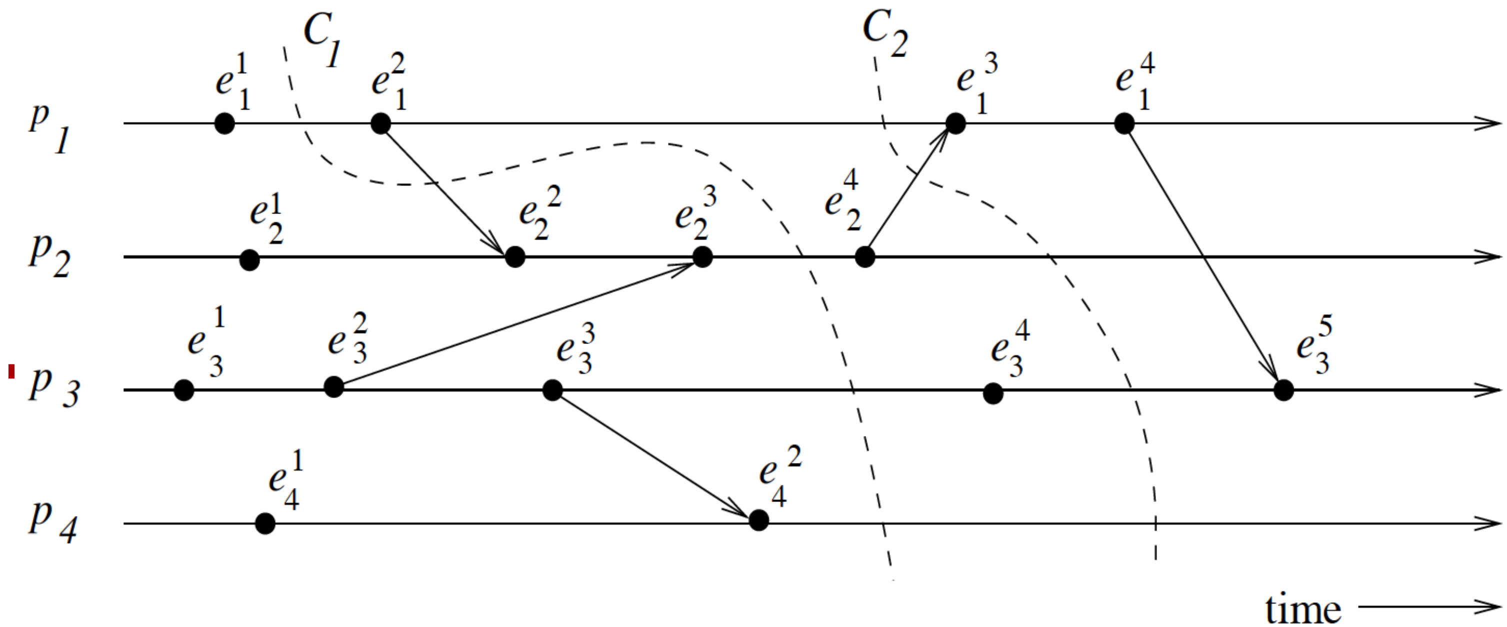


Consistent Global State – Details

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

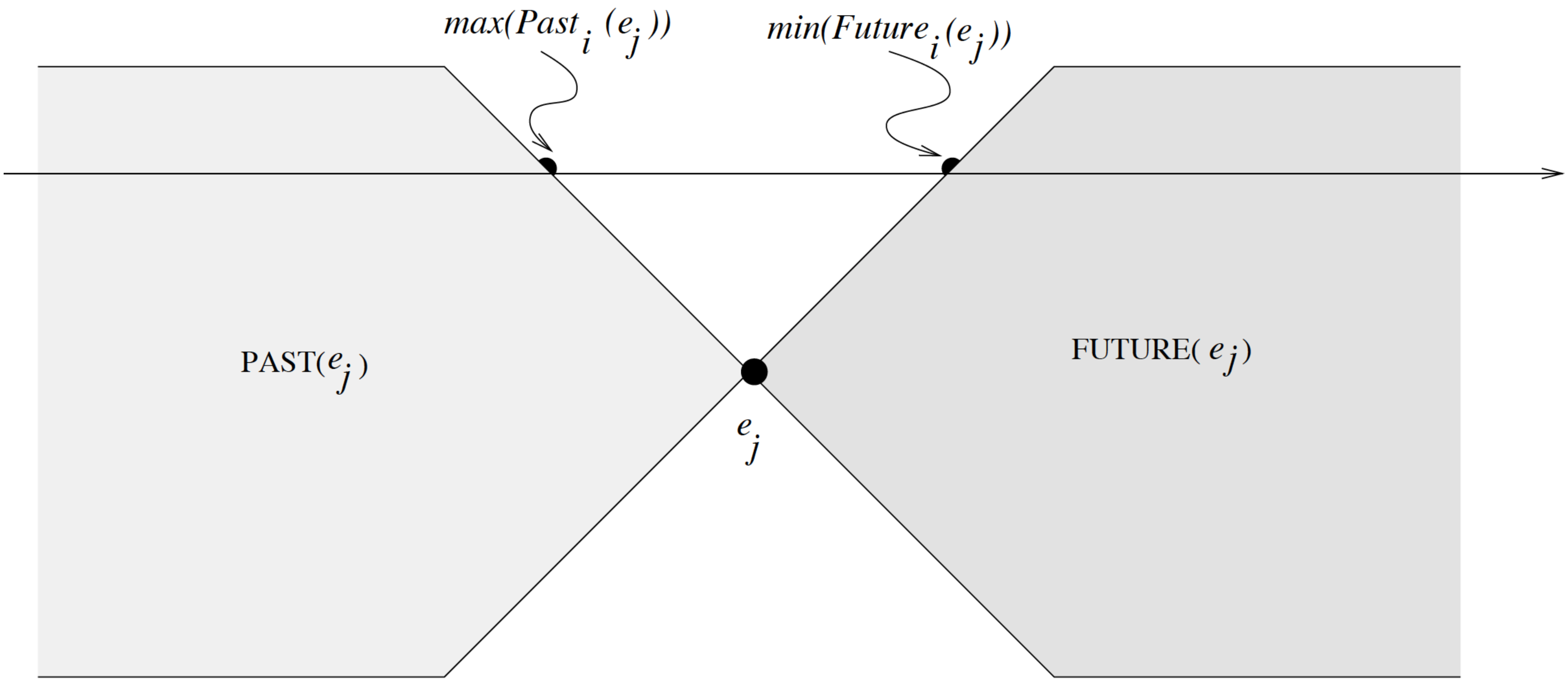
Cuts of a Distributed Computation



Cuts of a Distributed Computation

- In a consistent cut, every message received in the PAST of the cut was sent in the PAST of that cut
  - In previous figure, cut C2 is a consistent cut
- All messages that cross the cut from the PAST to the FUTURE are in transit in the corresponding consistent global state.
- A cut is inconsistent if a message crosses the cut from the FUTURE to the PAST
  - In previous figure cut C1 is an inconsistent cut

Past and Future Cones of an event



Physical vs Logical clocks?

- Logical Clocks
  - Design and Implementation
- Three Different Ways
  - Scalar Time
  - Vector Time
  - Matrix Time
- Virtual Clocks
  - Time Wrap Mechanism
- Clock Synchronization
  - NTP Synchronization Protocol

Summary

- A Model of Distributed Computations
  - Distributed Sorting
    - Design and Implementation Issues
  - Causal Precedence Relations
  - Global State and Cuts of a DS
  - PAST and FUTURE events
  - What about the ordering of events?
    - How do we efficiently handle the ordering of events (discrete events)?
    - Lamport's Logical Clocks ?
    - 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

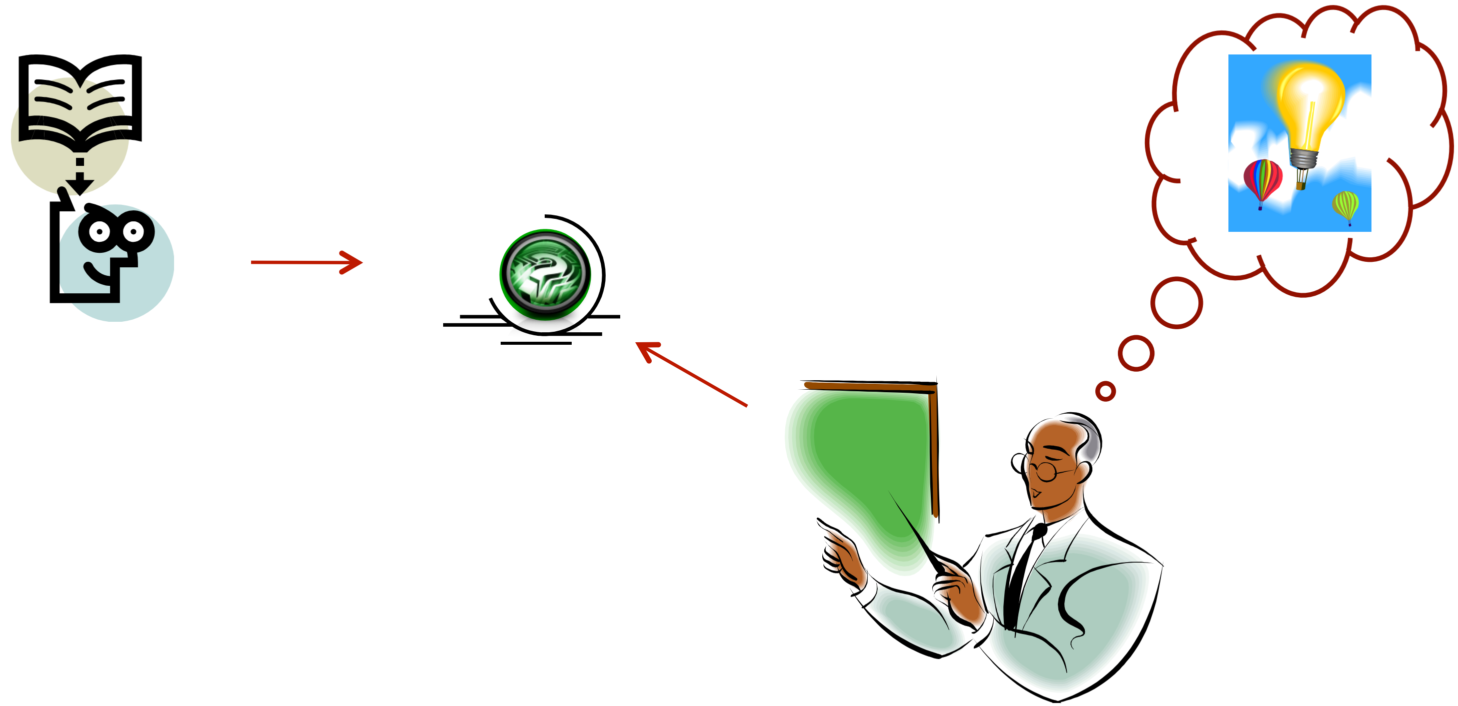
Help among Yourselves?

- **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 ???