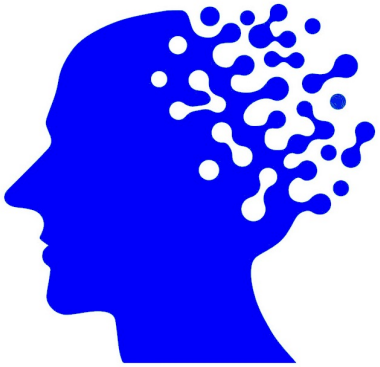


Spring 2024

Distributed Computing

- Logical and Vector Clocks

Dr. Rajendra Prasath

Indian Institute of Information Technology Sri City, Chittoor

14th February 2024 (http://rajendra.2power3.com)

> Distributed Computing?

- How will you design a Distributed Algorithm?

- Learn to Solve using Distributed Algorithms

> About this Lecture

What do we learn today?

- This covers a model of distributed computations that every algorithm designer needs to know
  - **Leslie Lamport's Logical Clocks**
  - **Vector Clocks**
  - **Some Examples using Logical clocks and Vector Clocks**

Let us explore these topics ➔ ➔ ➔

Today's Focus

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
- Local State vs. Global State

Recap

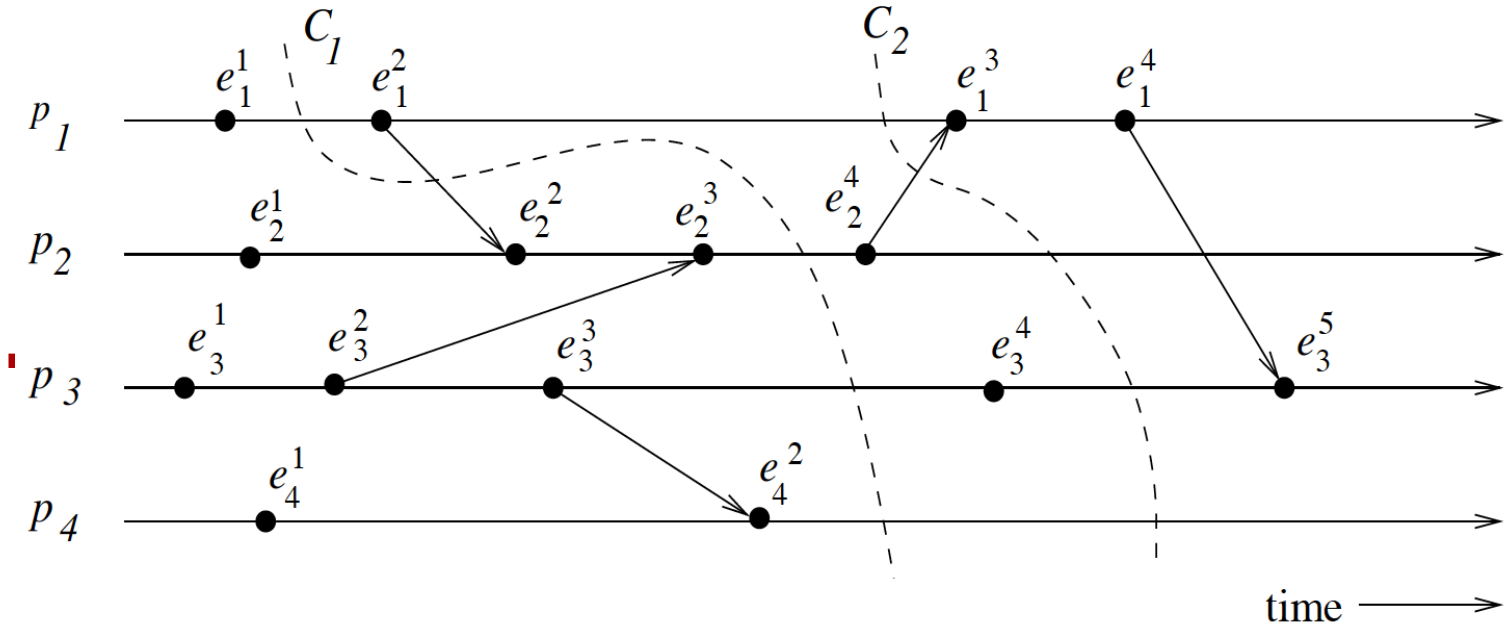
Causal Ordering

- 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.

Cuts of a Distributed Computation



Physical vs Logical clocks?

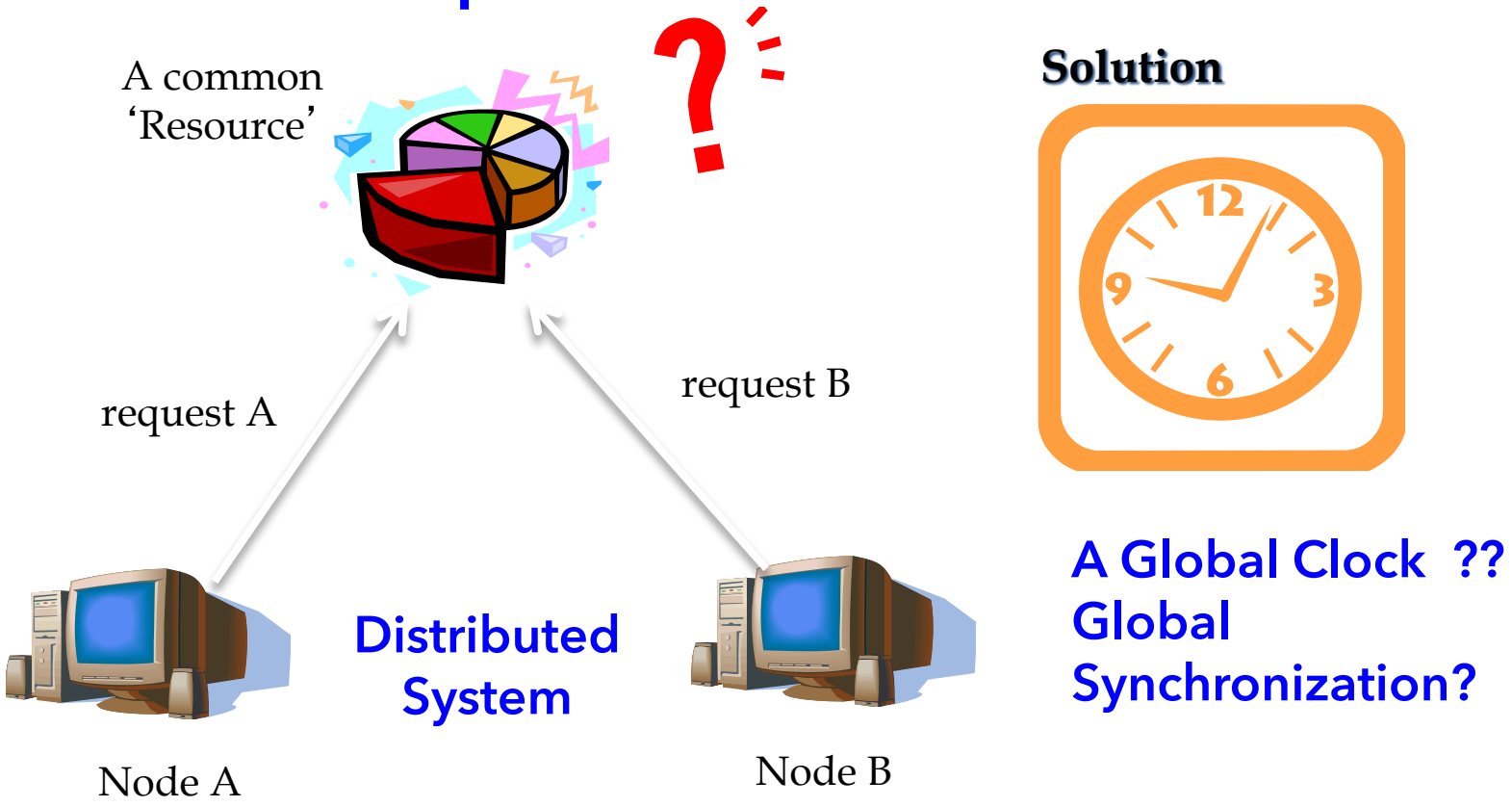
- Logical Clocks
  - Design and Implementation
- Three Different Ways
  - Scalar Time
  - Vector Time
  - Matrix Time
- Virtual Clocks
  - Time Wrap Mechanism
- Clock Synchronization
  - NTP Synchronization Protocol

Logical Clocks

- Logical Clocks (Lamport 1978)
  - Based on "Happens Before" concept
- Knowing the ordering of events is important (?!)
- not enough with physical time
- Two simple points [Lamport 1978]
  - the order of two events in the same process
  - the event of sending message always happens before the event of receiving the message

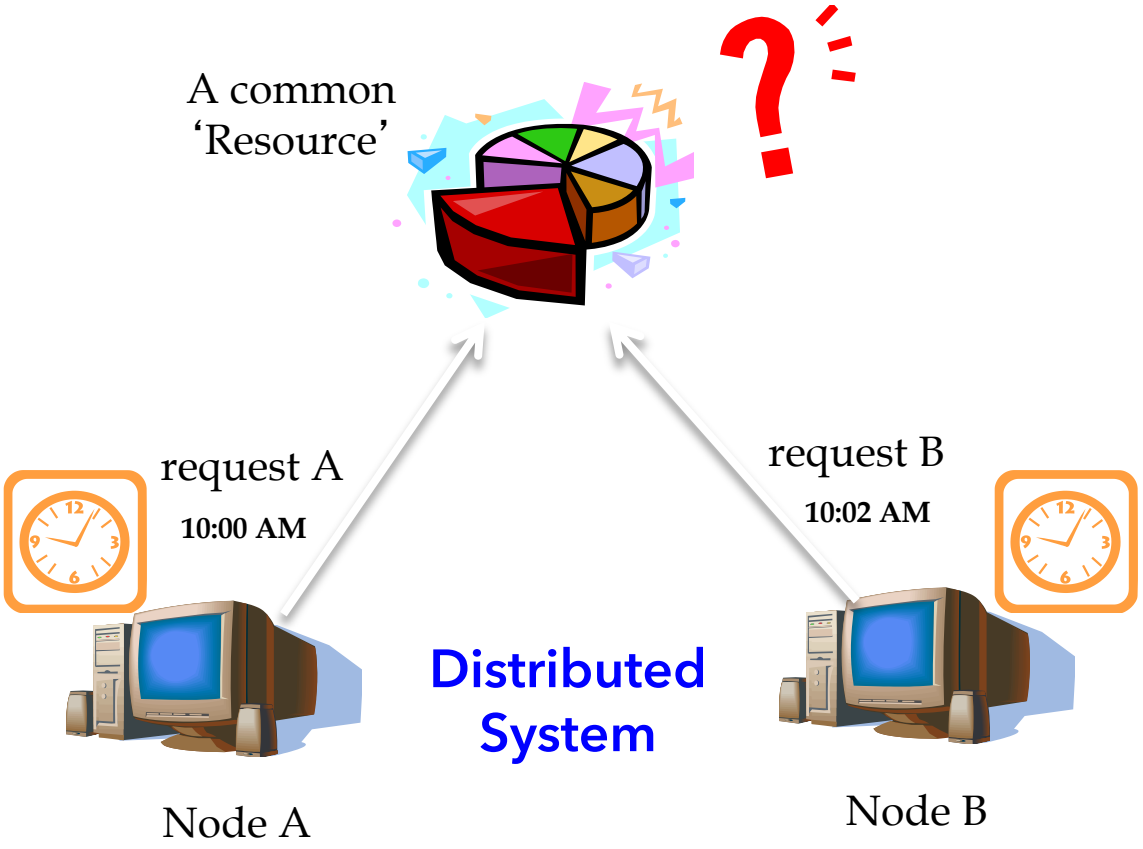
Events Ordering – An Example

➔ Which request was made first?



Events Ordering – An Example (contd)

➔ Which request was made first?



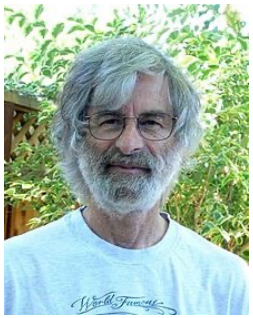
Solution

Individual Clocks?

Are individual clocks accurate, precise?

One clock might run faster/slower?

Logical Clocks (Lamport 1978)



Synchronization in a Distributed System:

- Event Ordering:
  - Which event occurred first?
- How to sync the clocks across the nodes?

- Can we define the notion of happened-before without using physical clocks?

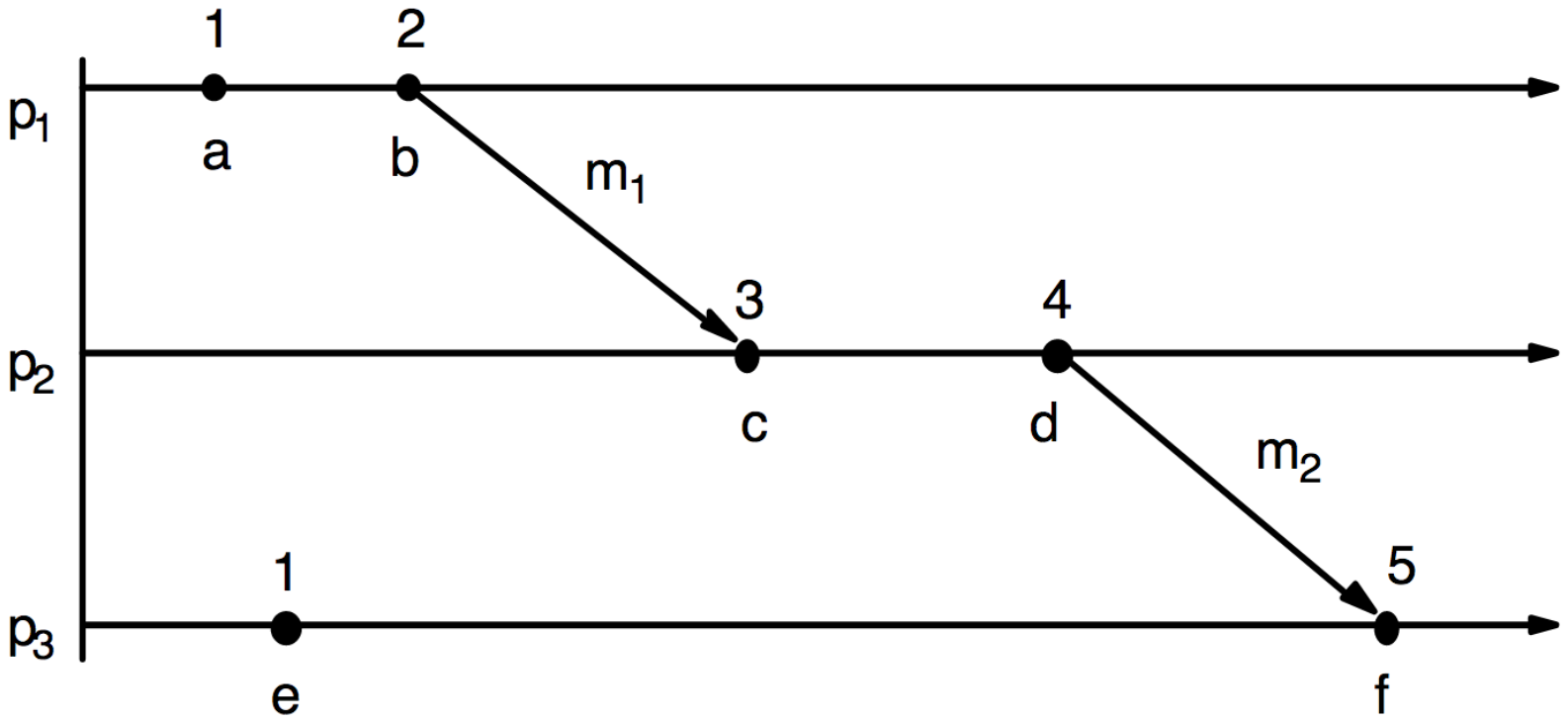
Lamport's Logical clocks

- A monotonically increasing software counter
- It does (need) not relate to a physical clock
- Each process p_i has a logical clock L_i
- LC_1: L_i is incremented by 1 before each event at process p_i
- LC_2:

  A) when process p_i sends message m, it piggybacks $t = L_i$

  B) when p_j receives (m,t), it sets $L_j = max(L_j, t)$ and applies LC_1 before timestamping the event $receive(m)$

A Close Look

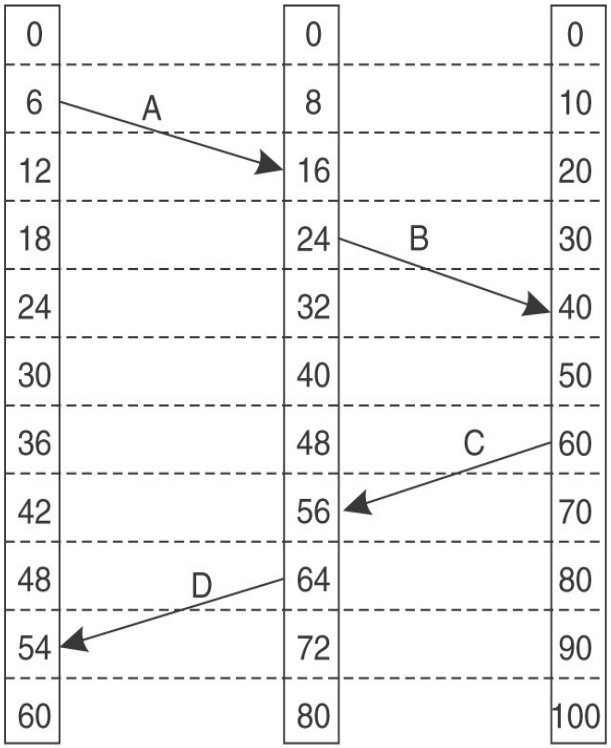


- e → e′ ⇒ L(e) < L(e′) but not vice versa
- Example: event b and event e

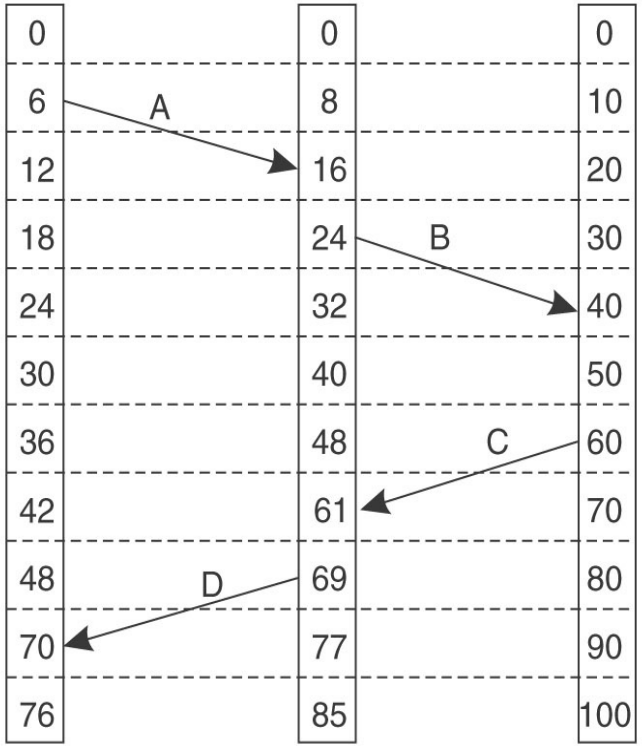
How to implement Lamport's clocks?

- When a message is transmitted from P1 to P2, P1 will encode the send time into the message.
- When P2 receives the message, it will record the time of receipt
- If P2 discovers that the time of receipt is before the send time, P2 will update its software clock to be one greater than the send time (1 milli second at least)
- If the time at P2 is already greater than the send time, then no action is required for P2
- With these actions the "happens-before" relationship of the message being sent and received is preserved

Correction of Clocks



(a)



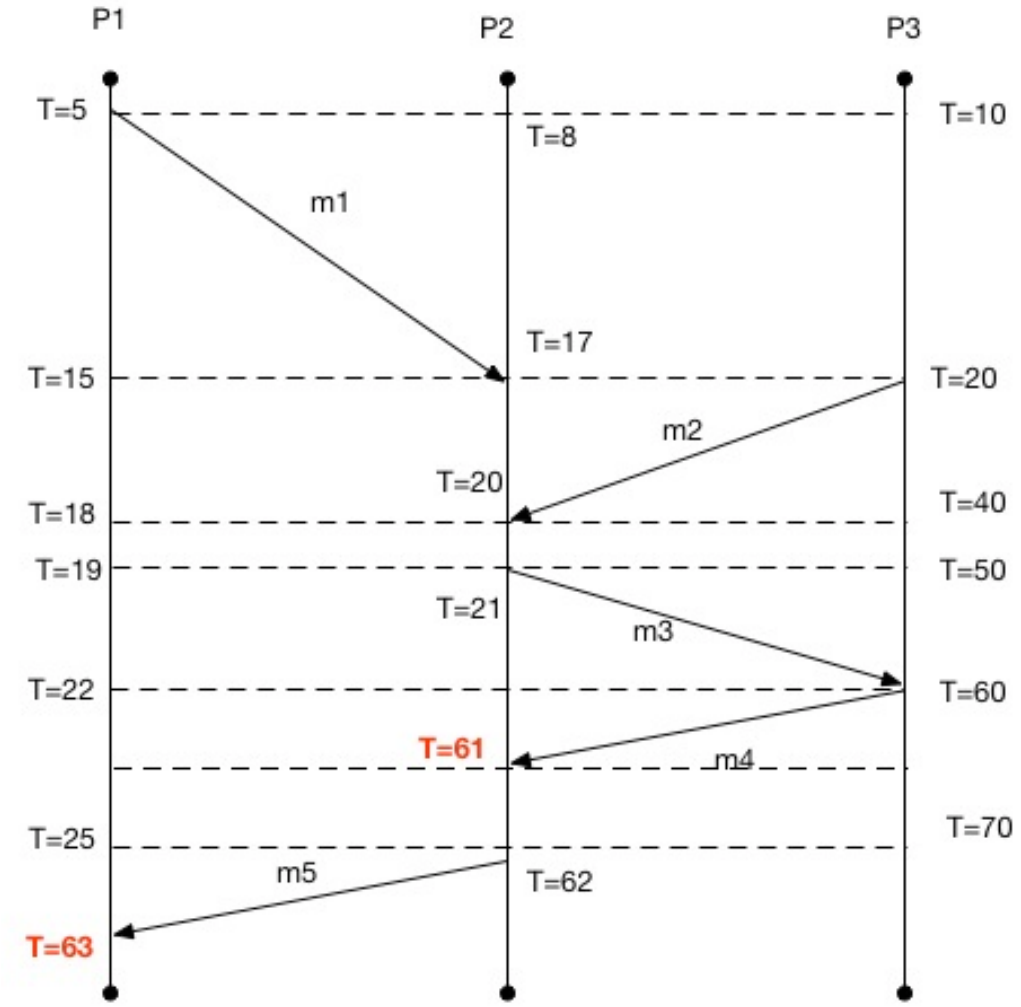
(b)

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?



Limitations

- Lamport's logical clocks lead to a situation where all events in a distributed system are totally ordered

  If a → b, then we can say C(a) < C(b)

- Unfortunately, with Lamport's clocks, nothing can be said about the actual time of a and b

  If the logical clock says a → b, that does not mean in reality that a actually happened before b in terms of real time

Issues with Lamport Clocks

- The problem with Lamport clocks is that they do not capture causality
- If we know that a → c and b → c we cannot say which action initiated c
- This kind of information can be important when trying to reply events in a distributed system (such as when trying to recover after a crash)
- If one node goes down, if we know the causal relationships between messages, then we can replay those messages and respect the causal relationship to get that node back up to the state it needs to be in

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 preceed 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

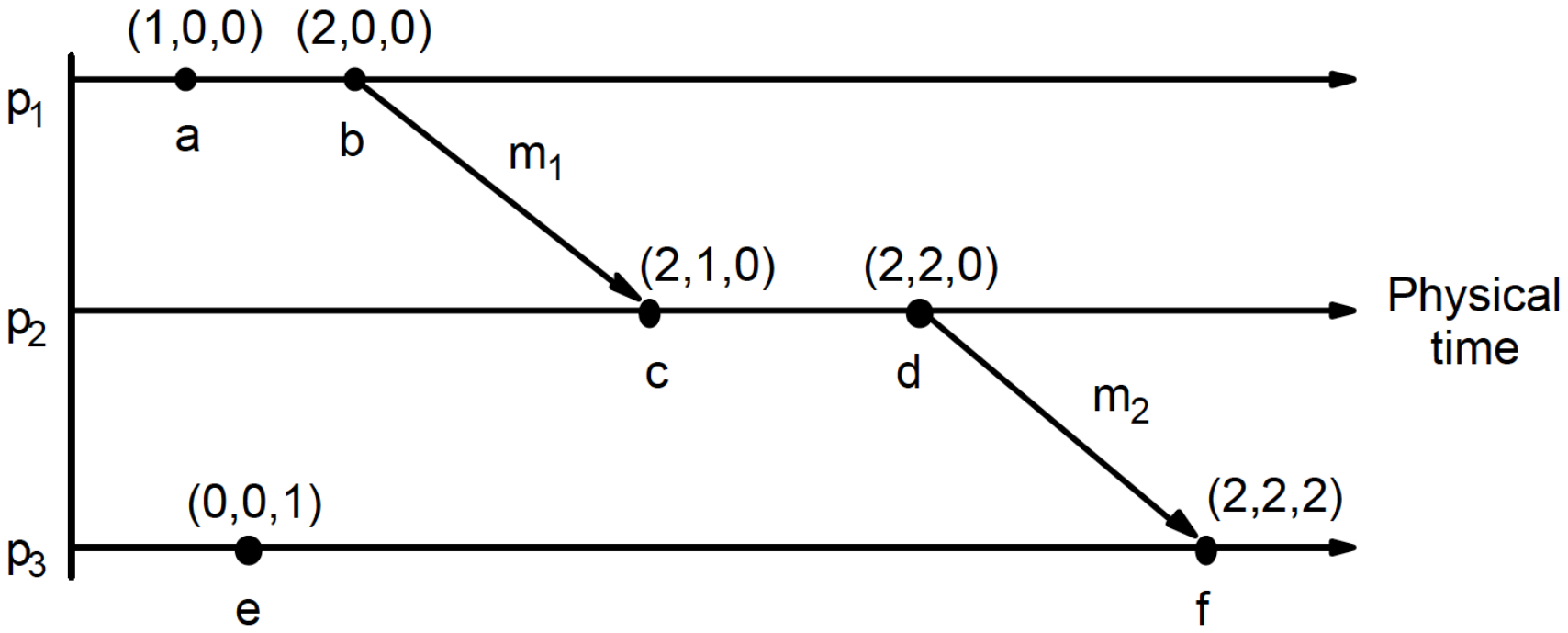
Implementing Vector Clocks

- Increment $VC_i[i]$ at each new event at P_i
- Updating Clocks:
  - Before executing any event (sending a message or an internal event):

    P_i executes $VC_i[i] \leftarrow VC_i[i]+1$

  - When process P_i sends a message m to P_j, it sets m's (vector) timestamp $ts(m)=Vc_i$
  - Upon receiving a message m, process P_j adjusts its own vector by setting $VC_j[k] \leftarrow max(VC_j[k], ts(m)[k])$ for each k

An Example



23

Understanding Vector Clocks

Meaning of =, <=, < for vector timestamps

(1) VC = VC′ iff VC[j] = VC′[j] for j = 1, 2, …, N

(2) VC ≤ VC′ iff VC[j] ≤ VC′[j] for j = 1, 2, …, N

(3) VC < VC′ iff VC ≤ VC′ and VC ≠ VC′

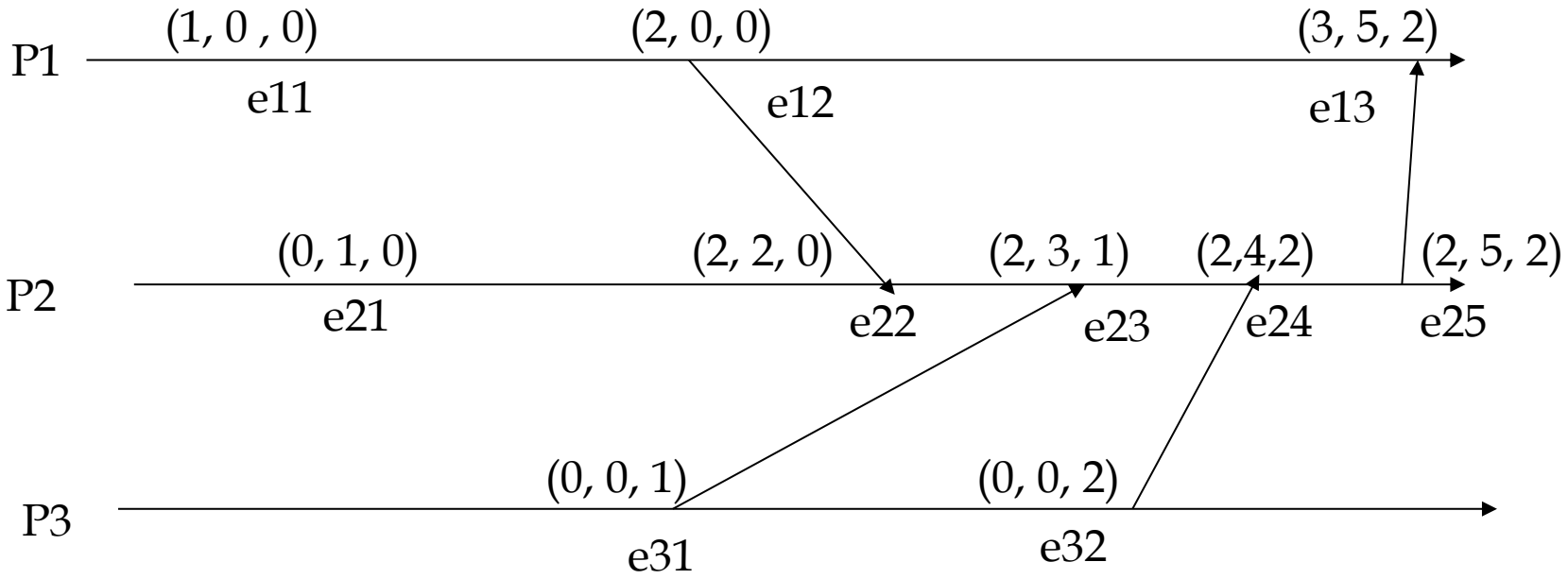
Examples:

(1, 3, 2) < (1, 3, 3)

(1, 3, 2) || (2, 3, 1)

➔ Note: e → e′ implies VC(e) < VC(e′) (The converse is also true)

An illustrative example



Less than or equal:

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

  (2,4,2) ≤ (3,5,2)

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

  This implies $e11 \rightarrow e22$

25

Summary

- A model of Distributed Computations
  - 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 ?
    - Vector Clocks

  - 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

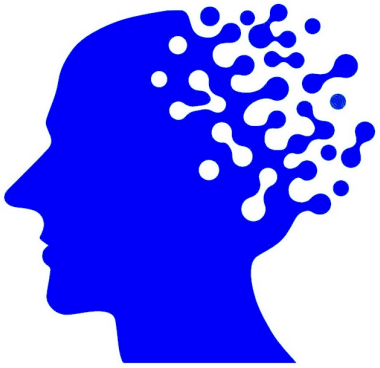
Best Approach

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

Questions
It's Your Time

Contact Information

Dr. Rajendra Prasath
IIIT Sri City, Chittoor



Spring 2024

Distributed Computing

- Global Snapshot Algorithm

Dr. Rajendra Prasath

Indian Institute of Information Technology Sri City, Chittoor

15-16th February 2024 (http://rajendra.2power3.com)

> Distributed Computing?

- How will you design a Distributed Algorithm?

- Learn to Solve using Distributed Algorithms

Recap: Distributed Systems

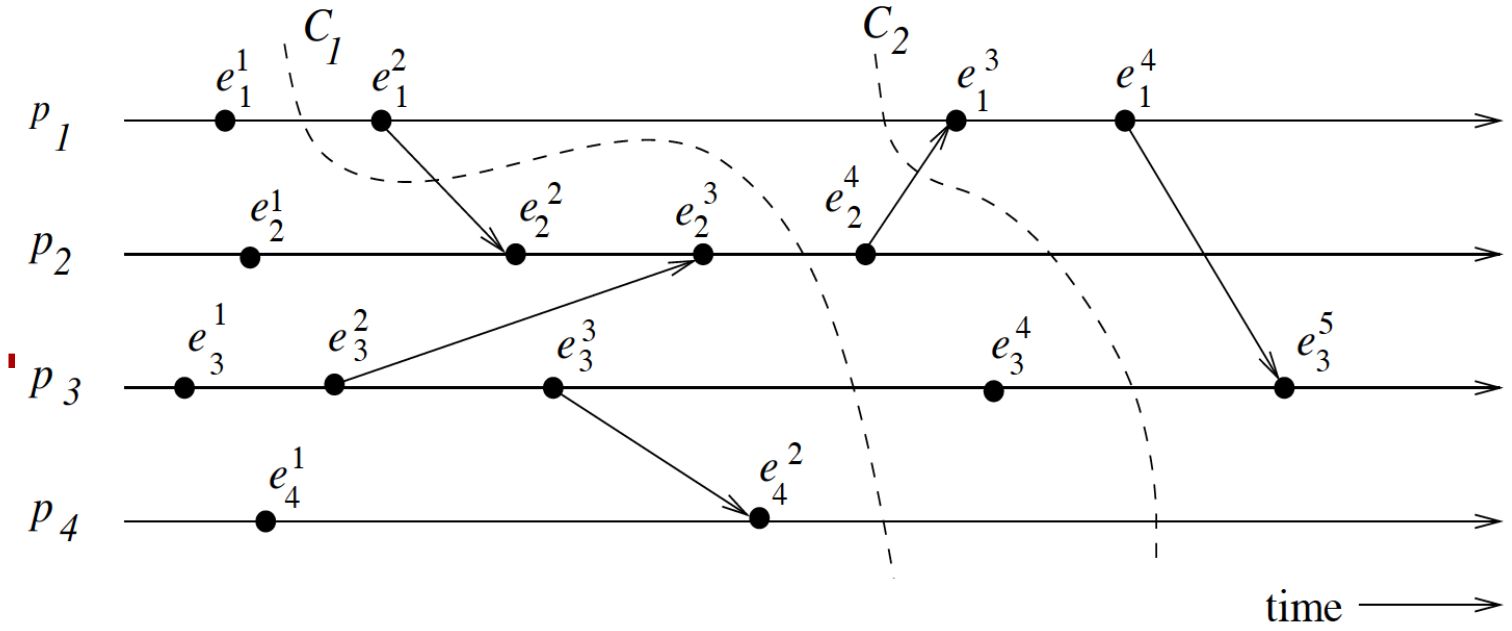
A Distributed System:

- 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

Recap: Cuts - Distributed Computation



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

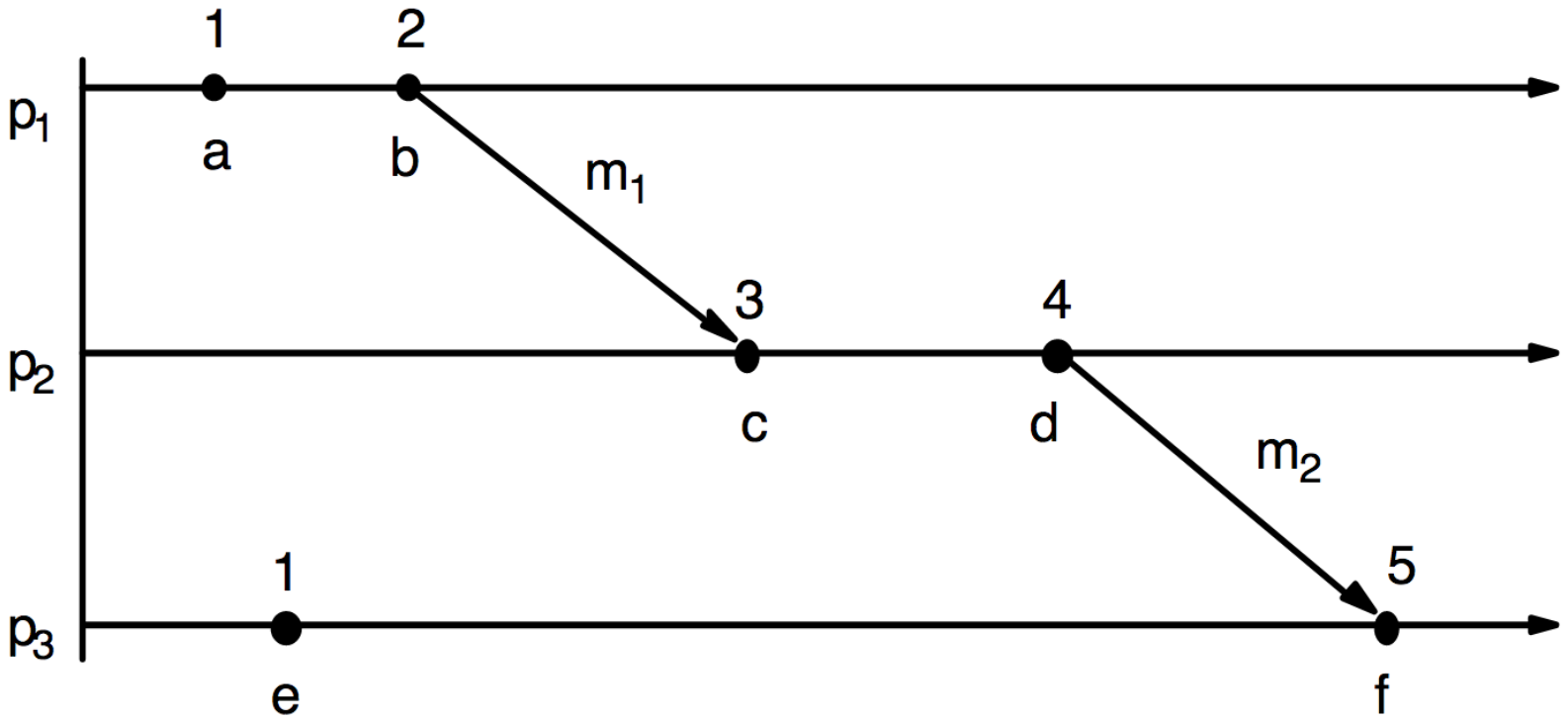
Causal Ordering

- 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 $receive(m_{ij}) \rightarrow receive(m_{kj})$

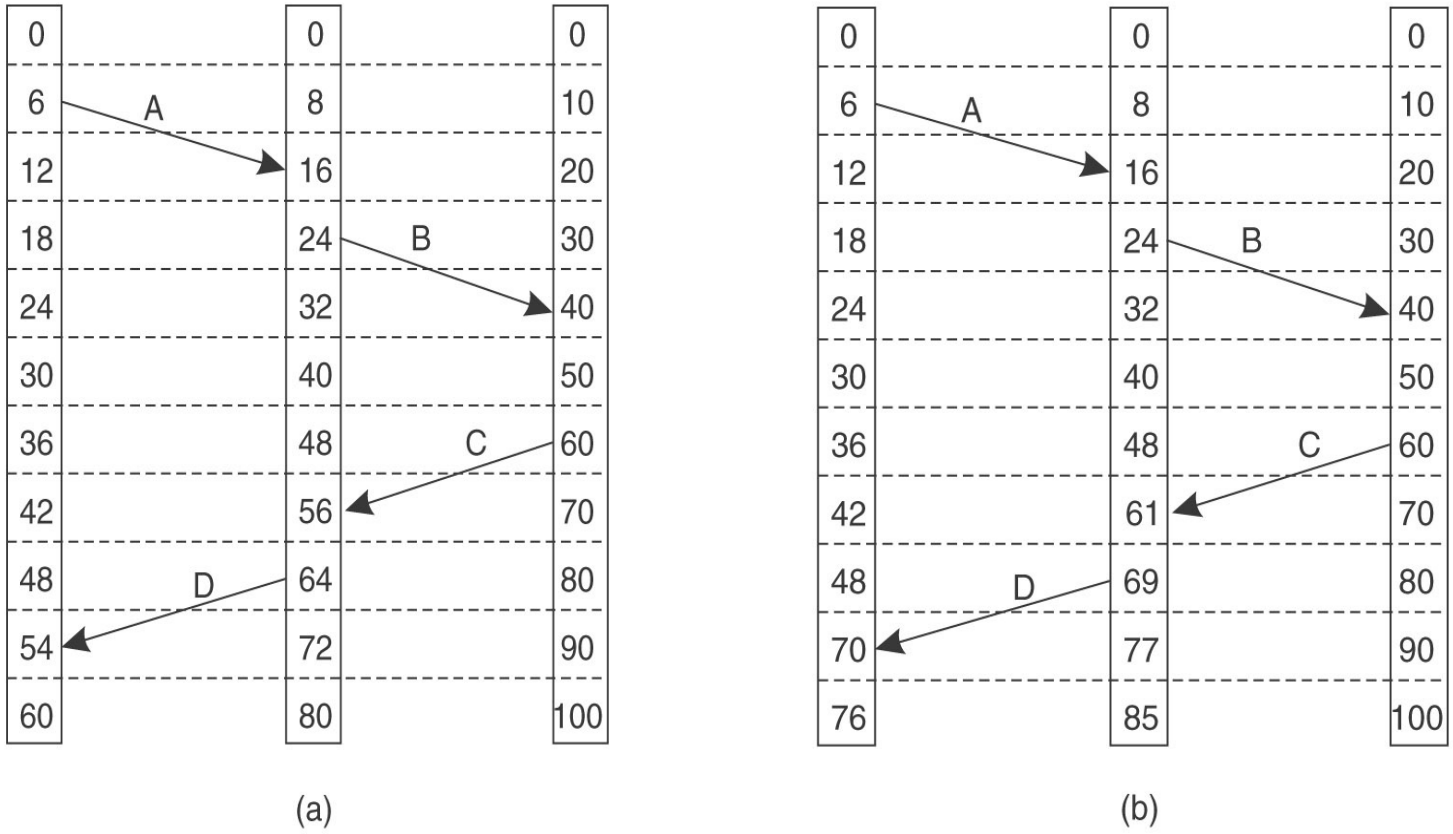
- This property ensures that causally related messages destined to the same destination are delivered in an order that is consistent with their causality relation.
- Causally ordered delivery of messages implies FIFO message delivery. (Note that CO $\subset$ FIFO $\subset$ Non-FIFO.)
- Causal ordering 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

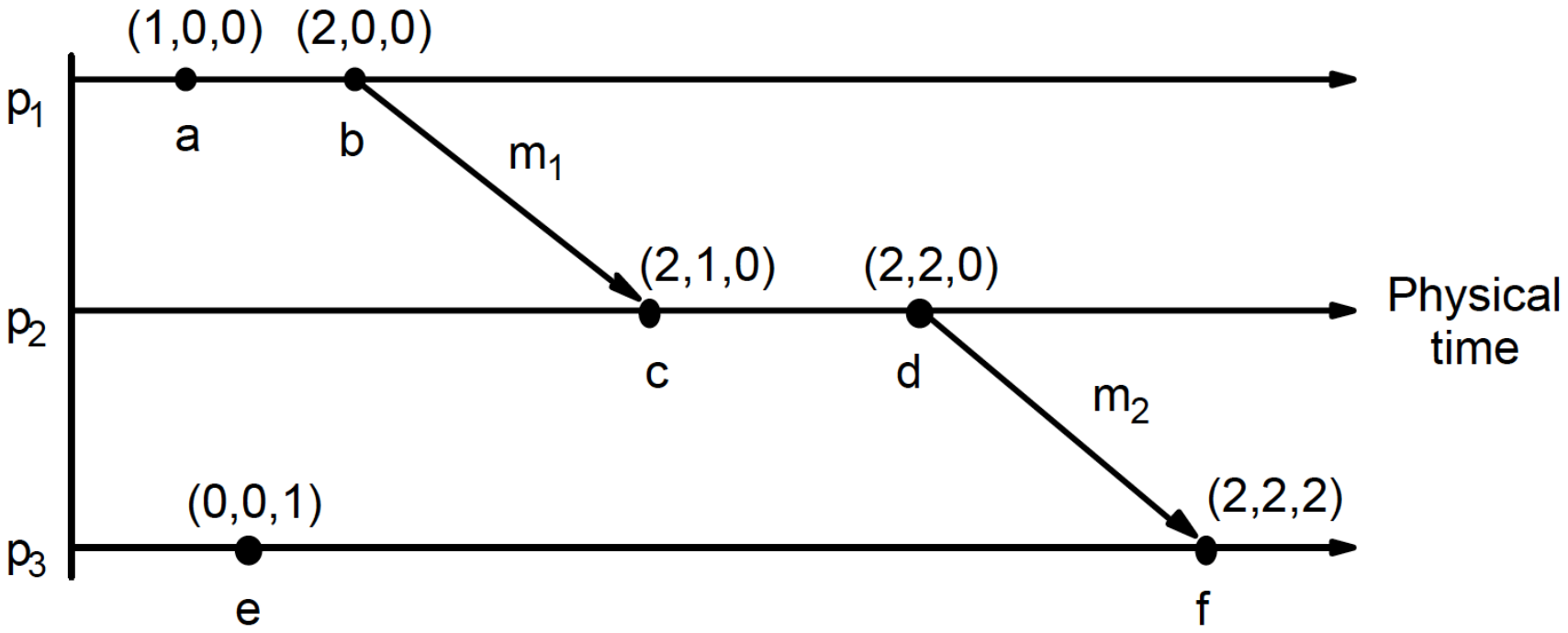
Correction of Clocks



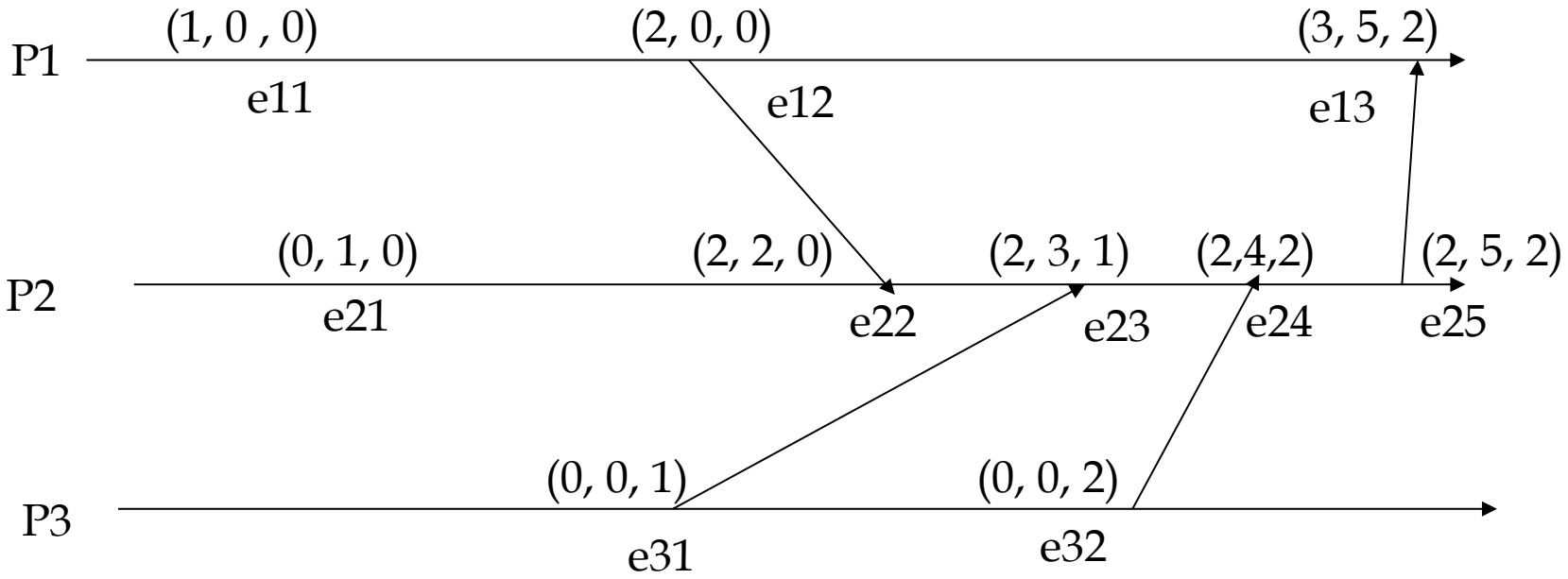
Vector Clocks

- 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



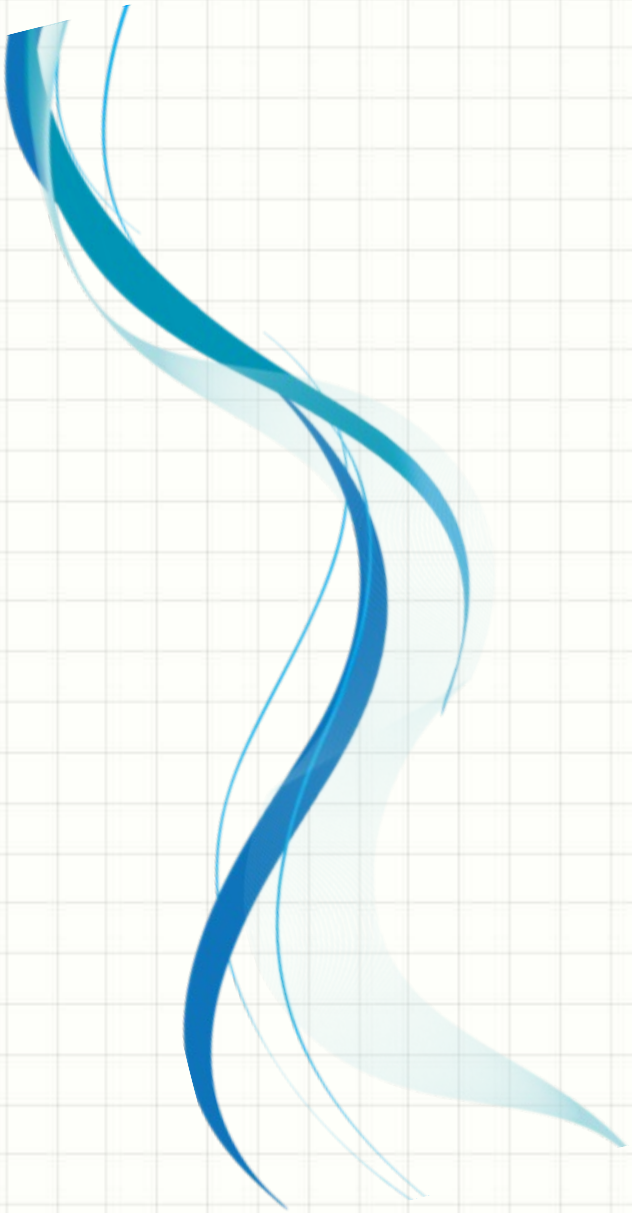
Less than or equal:

- $ts(a) \leq ts(b)$ **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 $e11 \rightarrow e22$



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:

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

- 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 !!):
  - 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

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

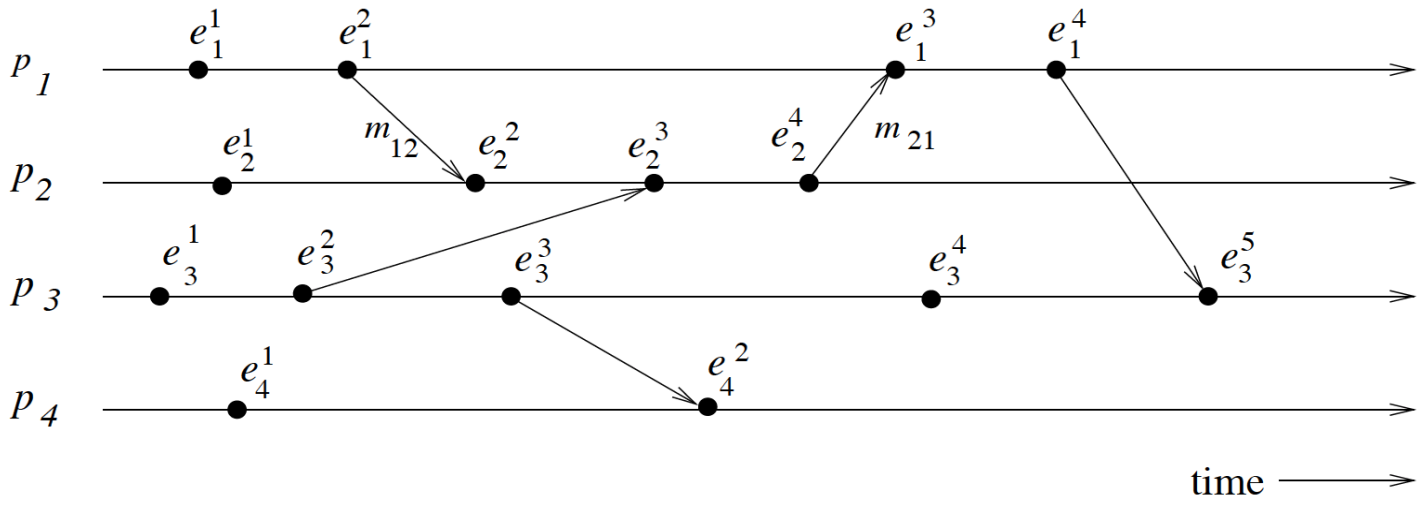
Where the global state is given by

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

➔ 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}.

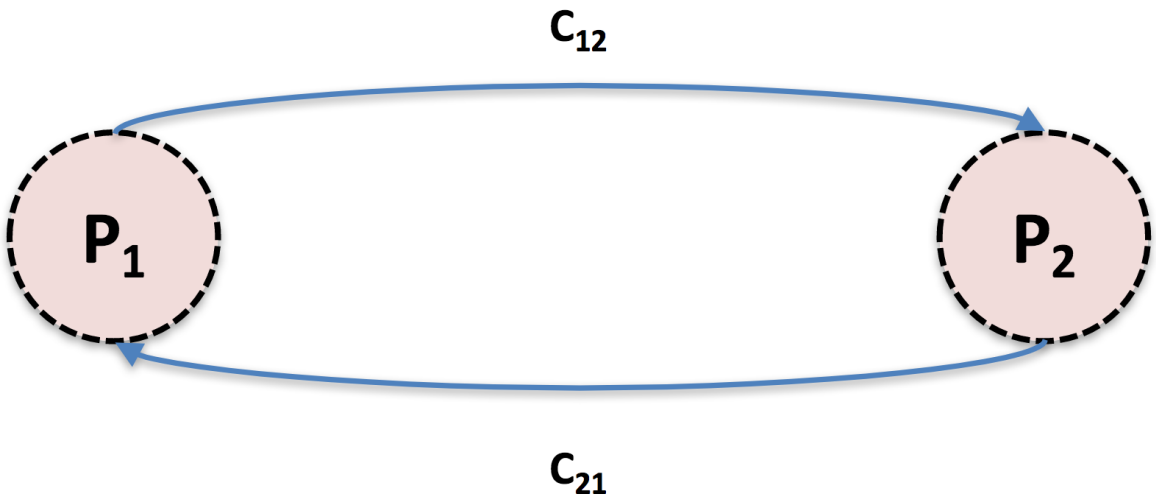
Changes in Global State?

- 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

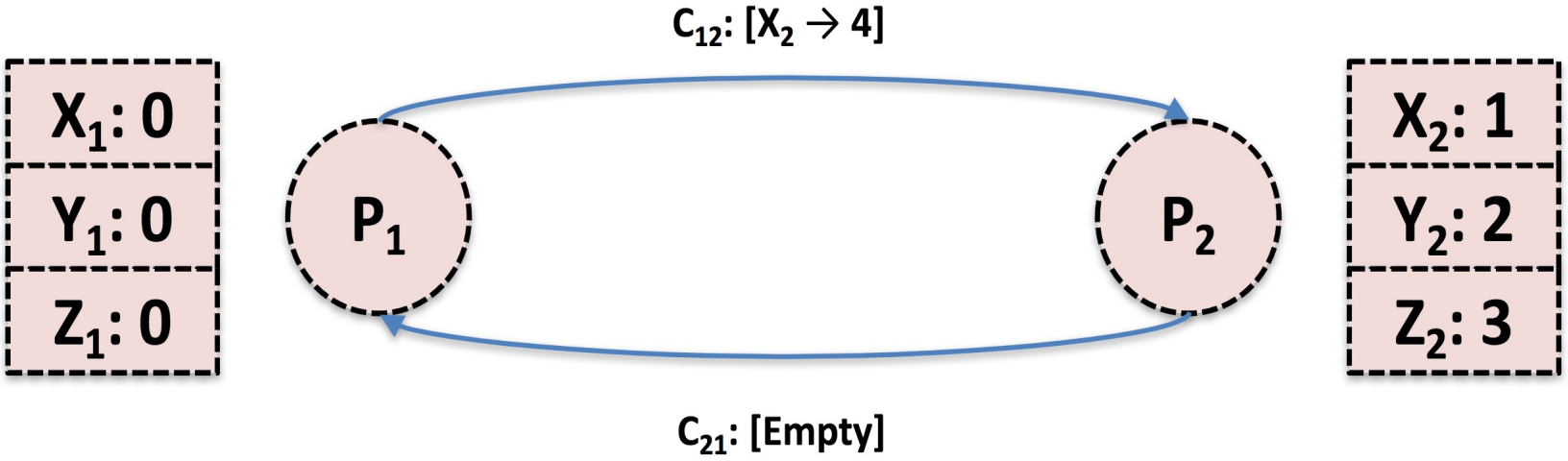
An Example Scenario

- 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



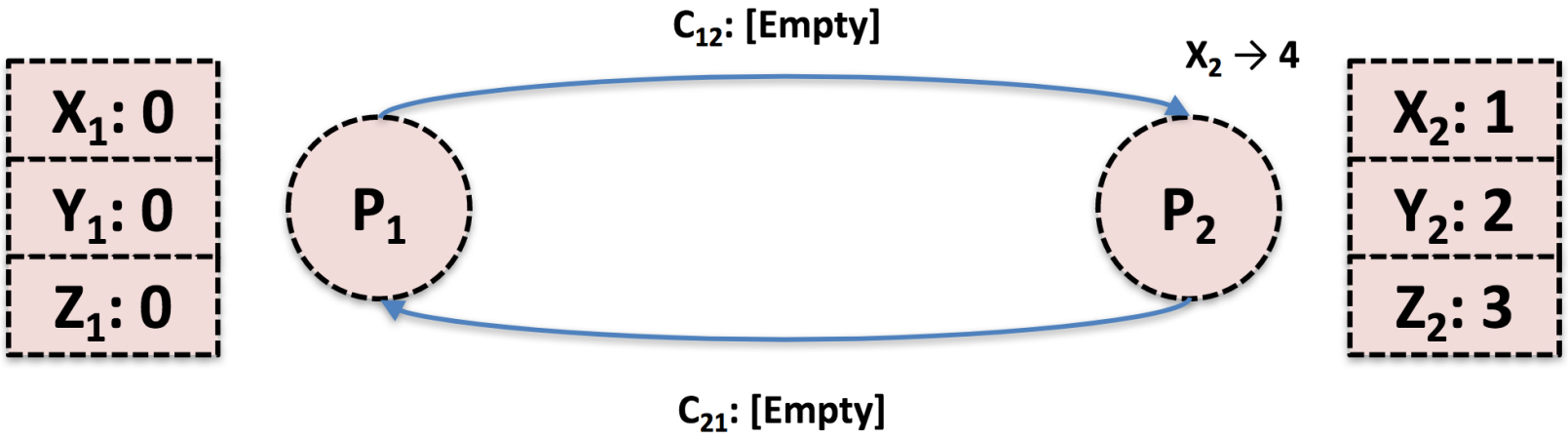
An Example (contd)

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



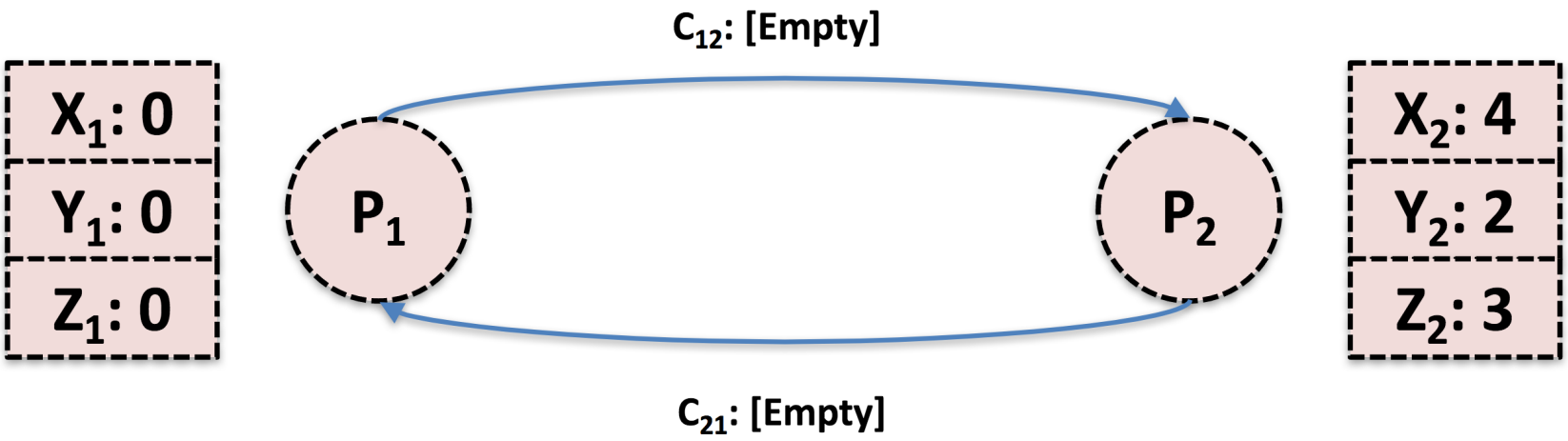
An Example (contd)

➔ 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:

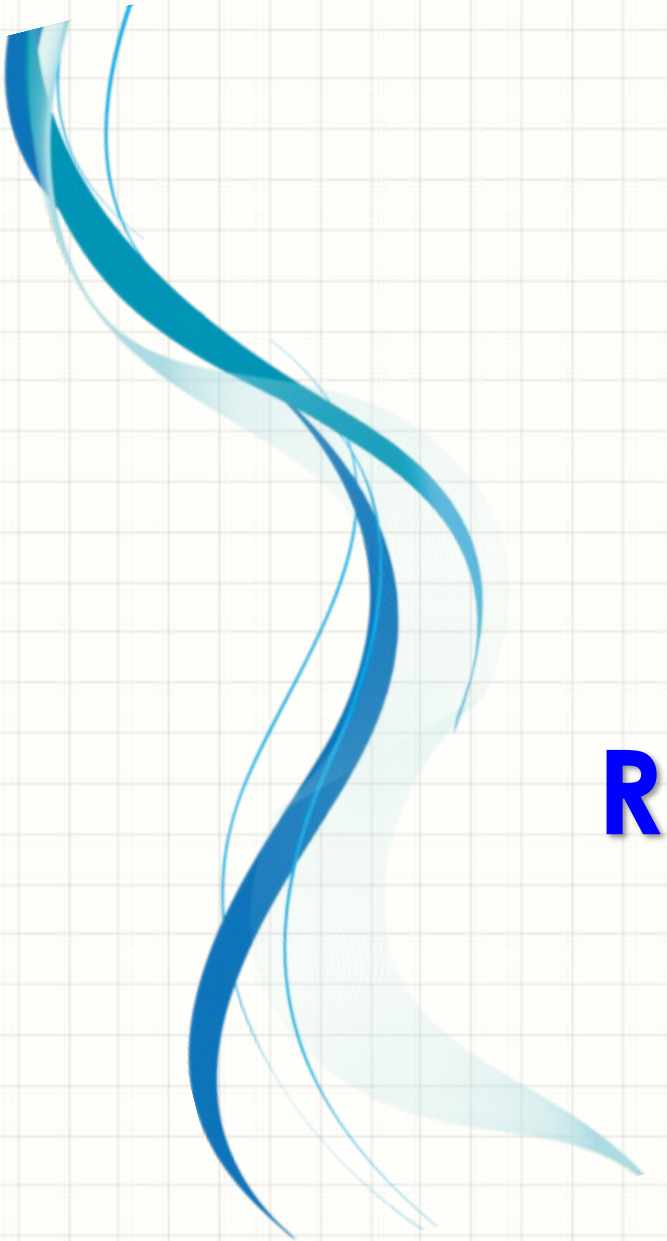
➔ remove objects that do not have any references

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-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-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

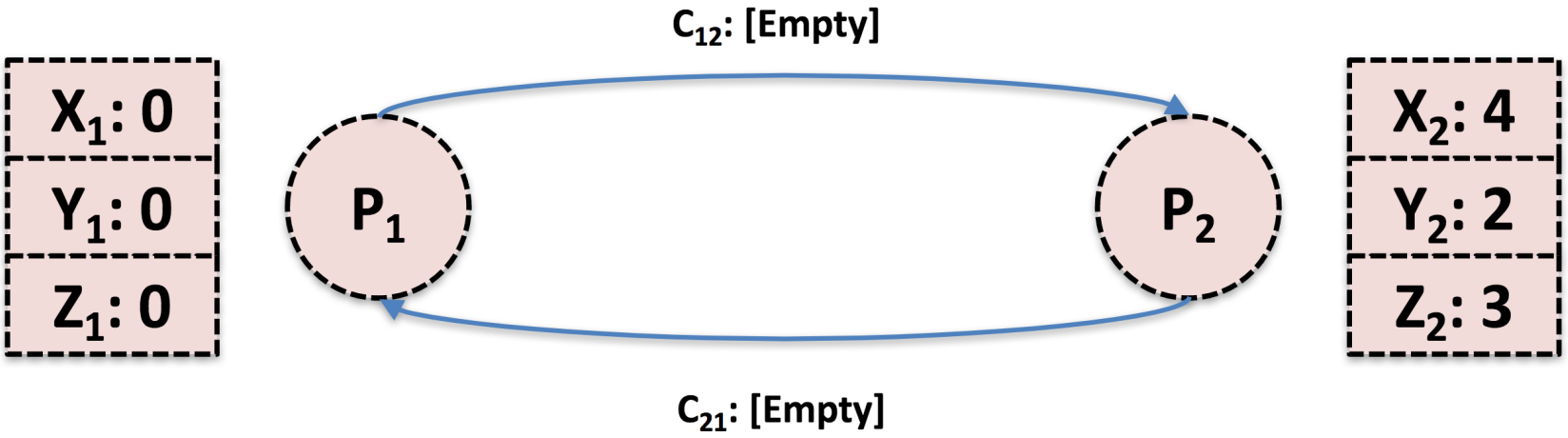
Terminating a snapshot

- 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!!)

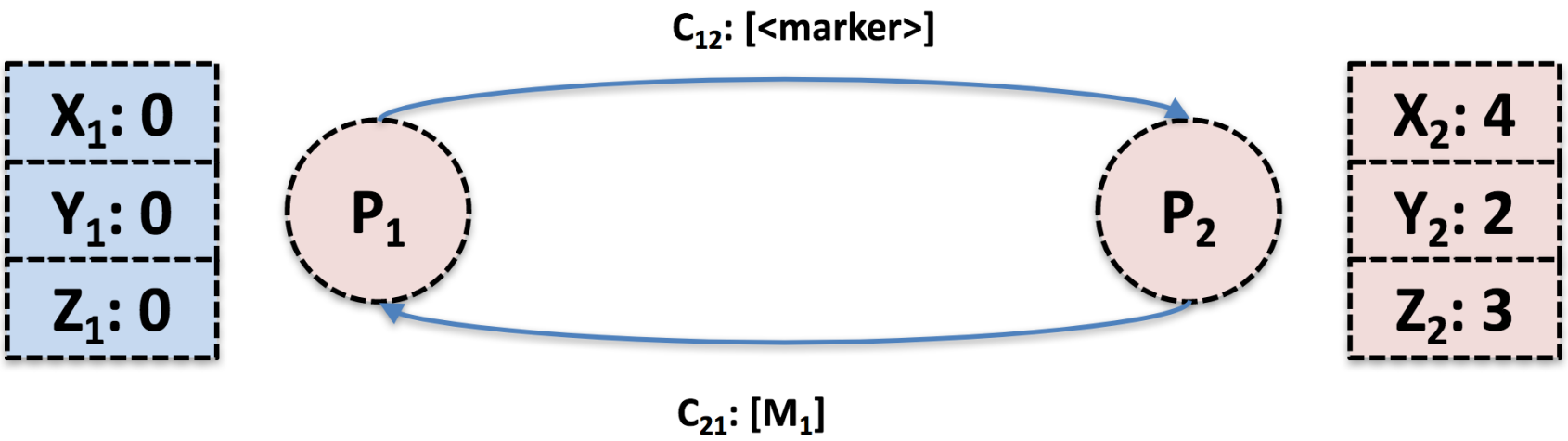
A Close Look of an Example

➔ Initially all channels are empty!



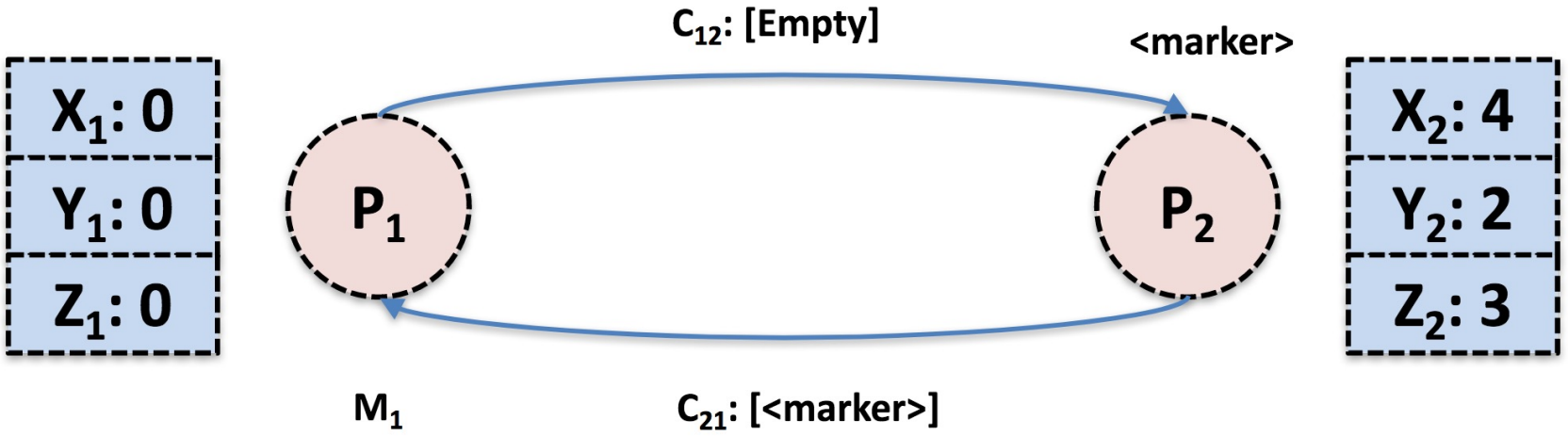
An Example (contd)

- 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



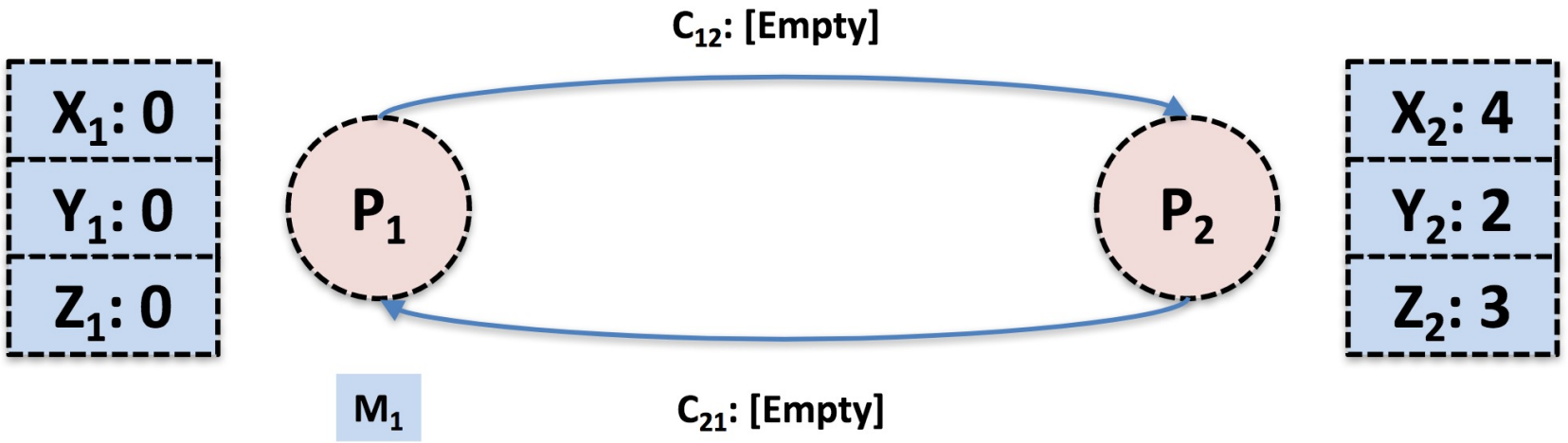
An Example (contd)

- 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 !!

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

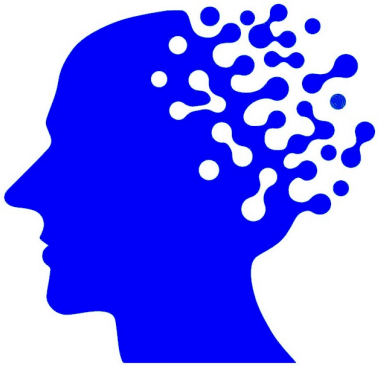
Best Approach

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

Questions
It's Your Time

Contact Information

Dr. Rajendra Prasath
IIIT Sri City, Chittoor



Spring 2024

Distributed Computing

- Termination Detection Algorithm

Dr. Rajendra Prasath

Indian Institute of Information Technology Sri City, Chittoor

21st February 2024 (http://rajendra.2power3.com)

> Distributed Computing?

- How will you design a Distributed Algorithm?

- Learn to Solve using Distributed Algorithms

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

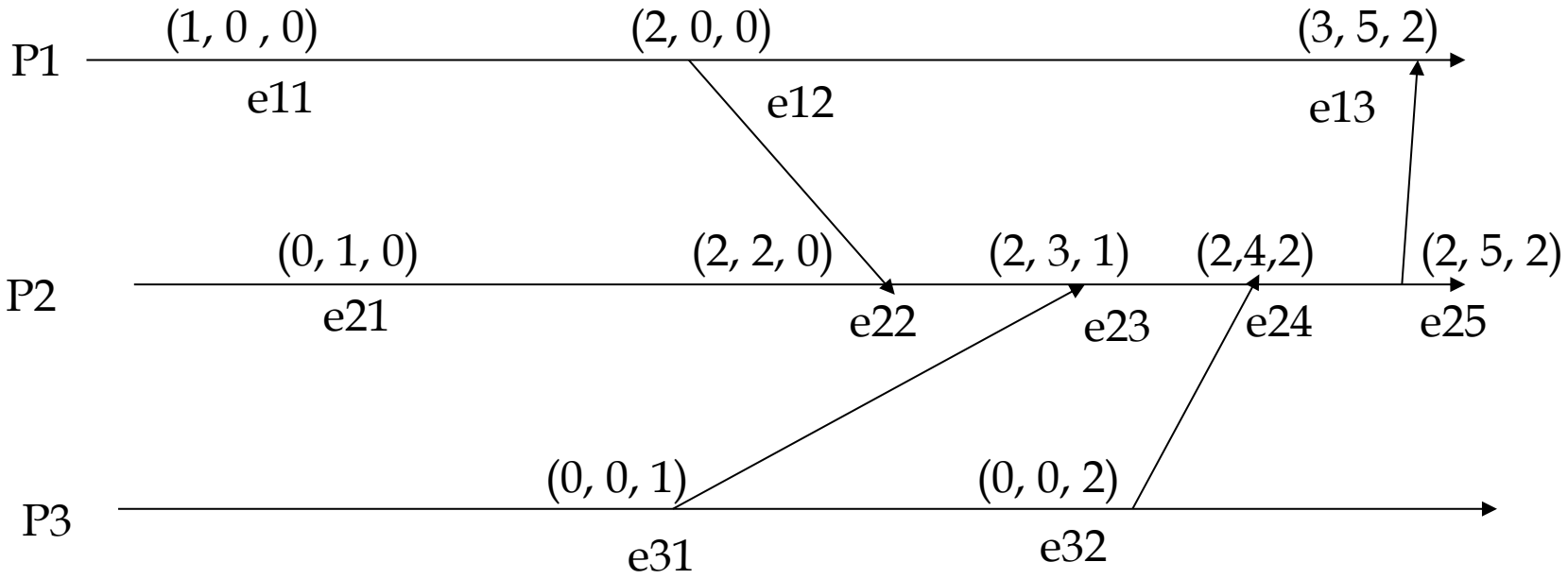
- Concurrent execution of processes:
  - Non-determinism, Race Conditions, Synchronization, Deadlocks, and so on
- No global clock
  - Coordination is done by message exchange
  - No Single Global notion of the correct time
- No global state
  - No Process has a knowledge of the current global state 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

Recap: 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 $e11 \square e22$

> 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**
  - **Termination Detection algorithm**
    - **Using Distributed Snapshots**

Today's Focus

Let us explore these topics ➔ ➔ ➔

A Consistent Global State

Definition:

➔ A global state is a consistent global state iff

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

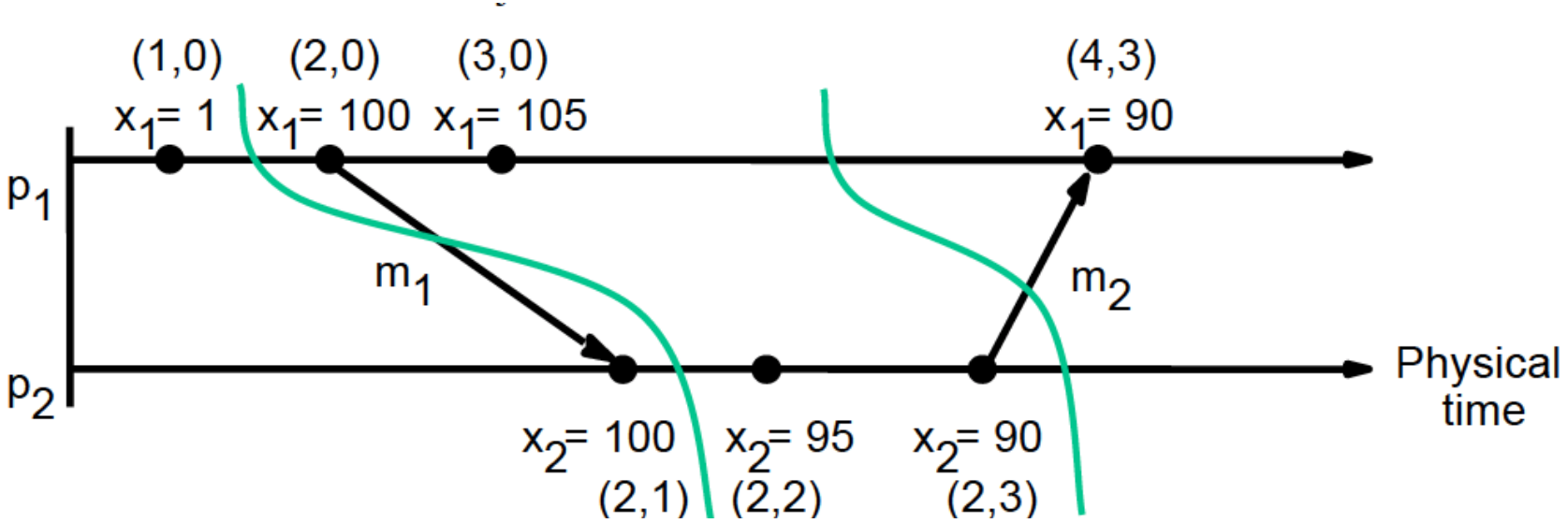
Where the global state is given by

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

➔ 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

- Let $V(s_i)$ be the vector timestamp of state s_i received from P_i.
- S is a consistent global state if and only if: $V(s_i)[i] >= V(s_j)[i]$ for all i,j in $[1, N]$



Reachable

When is a state said to be reachable?

- A state S′ is reachable from a state S if there exists a consistent run (Ordering of events satisfies all happened-before relations) from S to S′.

- May exist more than one consistent run, since the ordering from happened-before relation is a partial order

Data Sensitive Applications

Banking Example

➔ A Few Banking Operations

deposit(amount)

deposit amount in the account

withdraw(amount)

withdraw amount from the account

getBalance() → amount

return the balance of the account

setBalance(amount)

set the balance of the account to amount

Termination Detection

- A Fundamental Problem: Determine the termination status of a distributed computation
- A non-trivial task: NO process has complete knowledge of the global state, and global time does not exist
- A distributed computation is globally terminated if every process is locally terminated and there is no message in transit between any processes
- "Locally terminated" state is a state in which a process has finished its computation and will not restart any action unless it receives a message
- In the termination detection problem, a particular process (or all of the processes) must infer when the underlying computation has terminated

Important aspects

- Messages used in the underlying computation are called basic messages, and messages used for the purpose of termination detection are called control messages

- A termination detection (TD) algorithm must ensure the following:
  - Execution of a TD algorithm cannot indefinitely delay the underlying computation
  - The termination detection algorithm must not require addition of new communication channels between processes

System Model

- At any given time, a process can be in only one of the two states: active where it is doing local computation and idle otherwise and will be reactivated only on the receipt of a message from another process.
- An active process can become idle at any time but an idle process can become active only on the receipt of a message from another process
- Only active processes can send messages
- A message can be received by a process when it is in one of two states: active or idle. On receipt of a message, an idle process becomes active
- The sending of a message and the receipt of a message occur as atomic actions

Termination Detection - Definition

- Let $P_i\ (t)$ denote the state (active or idle) of process P_i at instant t
- Let $c_{i,\ j}\ (t)$ denote the number of messages in transit in the channel at instant t from process P_i to process P_j
- A distributed computation is said to be terminated at time instant t_k iff:

$$for\ all\ i\ ,$$

$$(P_i\ (t_k) = idle)\ \wedge\ (for\ all\ i, j\ such\ that\ \ c_{i,j}\ (t_k)\ ) = 0$$

- Thus, a distributed computation has terminated iff all processes have become idle and there is no message in transit in any channel

TD using Distributed Snapshots

- **Assumption:** There is a logical bidirectional communication channel between every pair of processes
- Communication channels are reliable
- Message delay is arbitrary but finite

TD using Distributed Snapshots

Main idea:

- When a process goes from active to idle state, it issues a request to all other processes to take a local snapshot, and also requests itself to take a local snapshot
- When a process receives the request, if it agrees that the requester became idle before itself, it grants the request by taking a local snapshot for the request
- A request is successful if all processes have taken a local snapshot
- The requester or any external agent may collect all the local snapshots of a request
- If a request is successful, a global snapshot of the request can thus be obtained and the recorded state will indicate termination of the computation

A Formal Description

- Each P_i has a logical clock x initialized to zero at t_0
- A process increments its x by one each time it becomes idle
- A basic message sent by a process at its logical time x is of the form $B(x)$
- A control message that requests processes to take local snapshot issued by P_i at its logical time x is of the form $R(x, i)$
- Each process synchronizes its logical clock x loosely with the logical clocks x's on other processes in such a way that it is the maximum of clock values ever received or sent in messages
- A process also maintains a variable k such that when the process is idle, (x,k) is the maximum of the values (x, k) on all messages $R(x, k)$ ever received or sent by the process
- Logical time is compared as follows: $(x, k) > (x', k')$ *iff* $(x > x')$ *or* $((x=x')$ *and* $(k>k'))$ i.e., a tie between x and x' is broken by the process identification numbers k and k'

Algorithm

➔ Four Rules:

(R1): When process i is active, it may send a basic message to process j at any time by doing
send a $B(x)$ *to* j.

(R2): Upon receiving a $B(x')$, process i does
let $x{:=}x'{+}1$;
if (i is idle) $\rightarrow$ go active.

(R3): When process i goes idle, it does
let $x{:=}x{+}1$;
let $k{:=}i$;
send message $R(x, k)$ to all other processes;
take a local snapshot for the request by $R(x, k)$.

(R4): Upon receiving message $R(x', k')$, process i does
$[((x', k') > (x,k)) \wedge$ (i is idle)$\rightarrow$ let $(x,k){:=}(x', k')$;
take a local snapshot for the request by $R(x', k')$;
$\square$
$((x', k') \leq (x,k)) \wedge (i$ *is idle*$)\rightarrow$ do nothing;
$\square$
(i is active)$\rightarrow$ let $x{:=}\max(x', x)$].

➔ The last process to terminate will have the largest clock value.

Summary

- Global Snapshots
  - Global State of a DS
  - Chandy – Lamport's GS Recording Algorithm
    - Initiating / Propagating / Terminating the Snapshot Algorithm
- Termination Detection
  - Definition
  - A Formal Description
  - TD using Global Snapshots

- 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

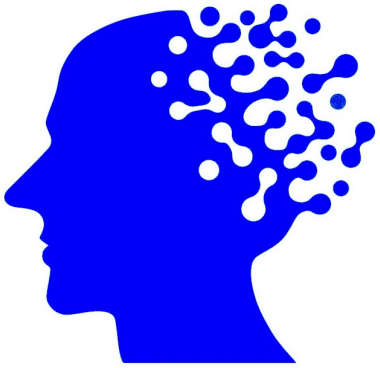
Best Approach

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

Questions
It's Your Time

Contact Information

Dr. Rajendra Prasath
IIIT Sri City, Chittoor



Spring 2024

Distributed Computing

- Leader Election Algorithm

Dr. Rajendra Prasath

Indian Institute of Information Technology Sri City, Chittoor

22-23rd February 2024 (http://rajendra.2power3.com)

> Distributed Computing?

- How will you design a Distributed Algorithm?

- Learn to Solve using Distributed Algorithms

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
- Global Snapshot Detection
- Termination Detection Algorithm
- [Now] Leader Election in Rings …

Recap

> About this Lecture

What do we learn today?

- This covers a model of distributed computations that every algorithm designer needs to know
  - **Termination Detection algorithm**
    - **Using Distributed Snapshots**
  - **Leader Election in Rings**

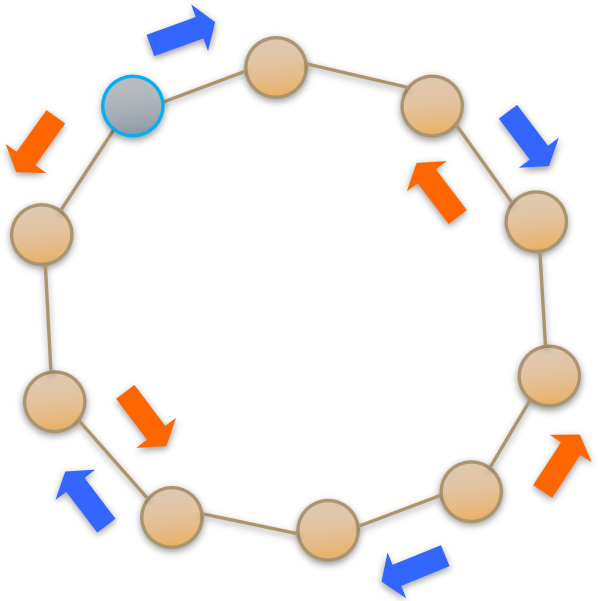
Let us explore these topics ➔ ➔ ➔

Today's Focus

Leader Election in Distributed Systems

Ring Networks

- In an oriented ring, processes have a consistent notion of left and right



Clockwise (right)

Anti-clockwise (left)

- For example, if messages are forwarded on right channel, they will cycle clockwise around the ring

Why Study Rings?

- Simple starting point, easy to analyze
- Abstraction of a token ring
- Lower bounds and impossibility results for ring topology also apply to arbitrary topologies

Leader Election - Definition

- Each processor has a set of **elected** (won) and **not-elected** (lost) states.
- Once an elected state is entered, processor is always in an elected state (and similarly for not-elected): i.e., irreversible decision

- In every admissible execution:
  - every processor eventually enters either an elected or a not-elected state
  - exactly one processor (the **leader**) enters an elected state

Uses of Leader Election

- A leader can be used to coordinate activities of the system:
  - find a spanning tree using the leader as the root
  - reconstruct a lost token in a token-ring network
- We will study leader election in rings.

Anonymous Rings

- How to model situation when processes do not have unique identifiers?
- First attempt: Does each process require to be in the same state machine?
- Subtle point: Does the algorithm rely on knowing the ring size (number of processes)?

Uniform (Anonymous) Algorithms

- A **uniform** algorithm does not use the ring size (same algorithm for each size ring)
  - Formally, every processor in every size ring is modeled with the same state machine
- A **non-uniform** algorithm uses the ring size (different algorithm for each size ring)
  - Formally, for each value of n, every processor in a ring of size n is modeled with the same state machine A_n .
- Note the lack of unique ids.

Leader Election in Anonymous Rings

- **Theorem:** There is *no* leader election algorithm for anonymous rings, even if
  - algorithm knows the ring size (non-uniform)
  - synchronous model

- **Proof Sketch:**
  - Every processor begins in same state with same outgoing messages (since anonymous)
  - Every processor receives same messages, does same state transition, and sends same messages in round 1
  - Do the same for rounds 2, 3, …
  - Eventually some processor is supposed to enter an elected state.

Leader Election in Anonymous Rings

- Proof sketch shows that either safety (never elect more than one leader) or liveness (eventually elect at least one leader) is violated

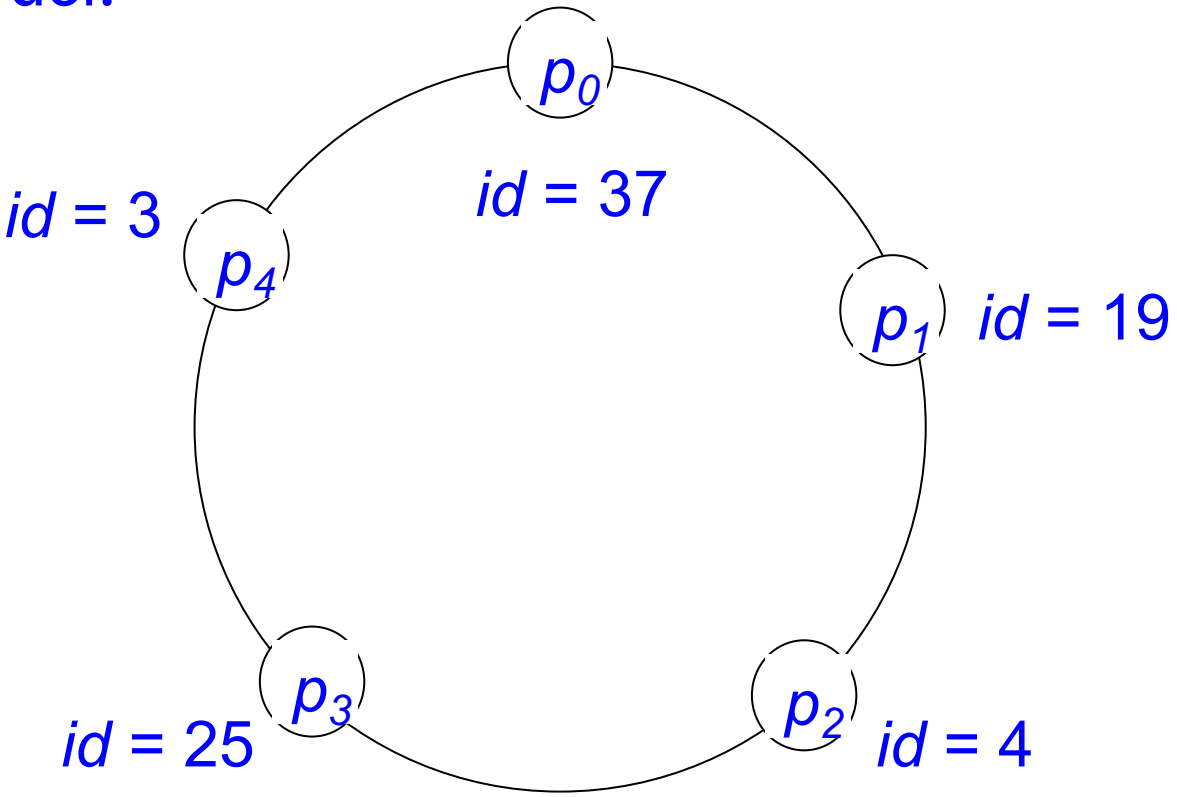
- Since the theorem was proved for non-uniform and synchronous rings, the same result holds for weaker (less well-behaved) models:
  - Uniform
  - Asynchronous

Rings with Identifiers

- Assume each processor has a unique ID.
- Don't confuse indices and IDs:
  - **indices** are 0 to *n - 1*; used only for analysis, not available to the processors
  - **IDs** are arbitrary nonnegative integers; are available to the processors through local variable *ID*

Specifying a Ring

- Start with the smallest ID and list IDs in clockwise order.



- Example: 3, 37, 19, 4, 25

Uniform (Non-anonymous) Algorithms

- **Uniform** algorithm: there is one state machine for every id, no matter what size ring
- **Non-uniform** algorithm: there is one state machine for every id and every different ring size
- These definitions are tailored for leader election in a ring.

Overview of LE in Rings with IDs

- There exist algorithms when nodes have unique IDs.
- We will evaluate them according to their *message complexity*.
- asynchronous ring:
  - $\Theta(n \log n)$ messages
- synchronous ring:
  - $\Theta(n)$ messages under certain conditions
  - otherwise $\Theta(n \log n)$ messages
- All bounds are asymptotically tight.

The LCR algorithm

- Lelann-Chang-Robert (LCR) Algorithm, 1979

- The network graph is a directed ring (uni-directed or bi-directed) consisting of n nodes (n may be unknown to the processes … Does this condition really require for rings?)

- Processes run the same deterministic algorithm
- The only piece of information supplied to the processes is a unique identifier (ID).
- IDs may be used

- In comparisons only (comparison-based algorithms)
- In comparisons and other calculations (non-comparison-based)

LCR algorithm - Description

- Each process sends its ID around the ring.
- When a process receives a ID, it compares this one to its own
- If the incoming ID is greater, then it passes this ID to the next process.
- If the incoming ID is smaller, then it discards it.
- If it is equal, then the process declares itself the leader.

LCR Algorithm for Leader Election

Alternative Algorithm:

- send value of own id to the left
- when receive an ID j (from the right):
- if j > id then
  - forward j to the left (this processor has lost)
  - if j < id then
    - do nothing
  - if j = id then
    - elect self as leader
      (this processor has won)

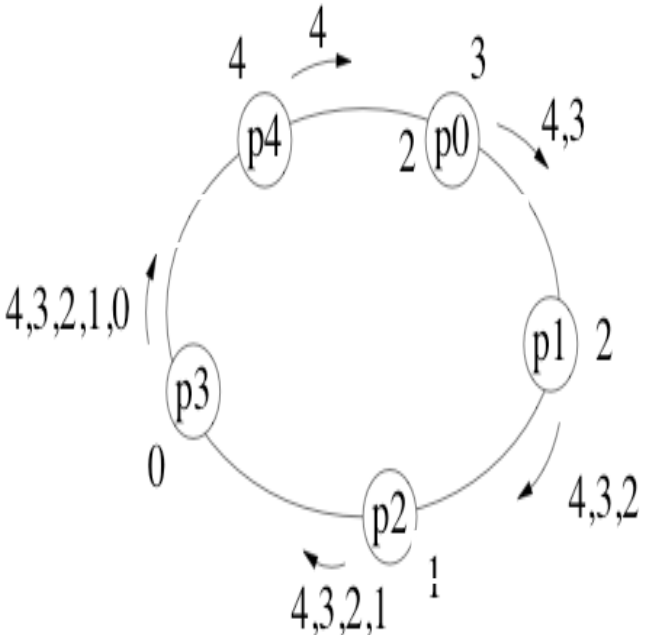
Analysis of $O(n^2)$ Algorithm

- **Correctness:** Elects processor with largest id.
  - message containing largest id passes through every processor
- **Time:** *O(n)*

- **Message complexity:** Depends how the ids are arranged.
  - largest id travels all around the ring (n messages)
  - 2nd largest id travels until reaching largest
  - 3rd largest id travels until reaching largest or second largest
  - And so on

Analysis of $O(n^2)$ Algorithm

- Worst way to arrange the ids is in decreasing order:
  - 2nd largest causes *n - 1* messages
  - 3rd largest causes *n - 2* messages and so on
- Total number of messages is $n + (n-1) + (n-2) + \ldots + 1 = \Theta(n^2)$.



Can We Use Fewer Messages?

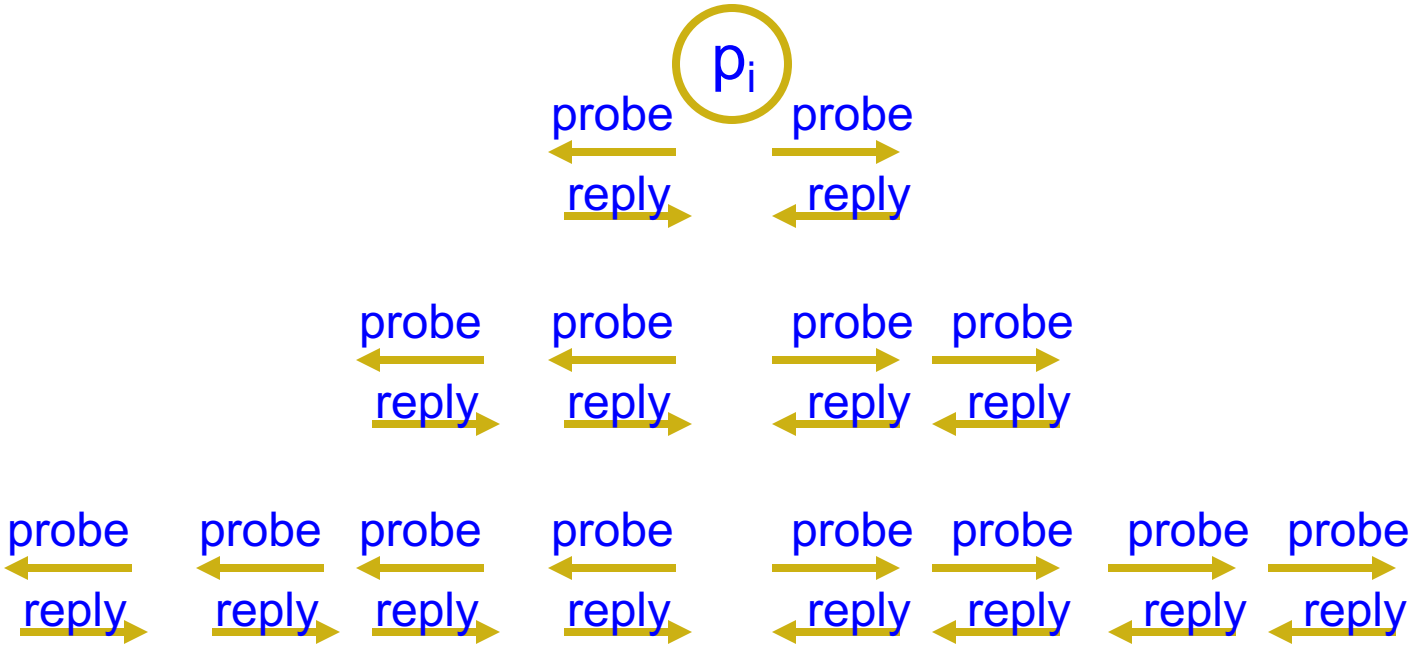
- The $O(n^2)$ algorithm is simple and works in both synchronous and asynchronous model
- But can we solve the problem with fewer messages?
- Idea:
  - Try to have messages containing smaller ids travel smaller distance in the ring

O(n log n) Leader Election

- Each process tries to probe successively larger neighborhoods in both directions
  - size of neighborhood *doubles* in each phase

- If probe reaches a node with a larger id, the probe stops
- If probe reaches end of its neighborhood, then a reply is sent back to initiator
- If initiator gets back replies from both directions, then go to next phase
- If process receives a probe with its own id, it elects itself

O(n log n) Leader Election



Analysis of O(nlogn) Algorithm

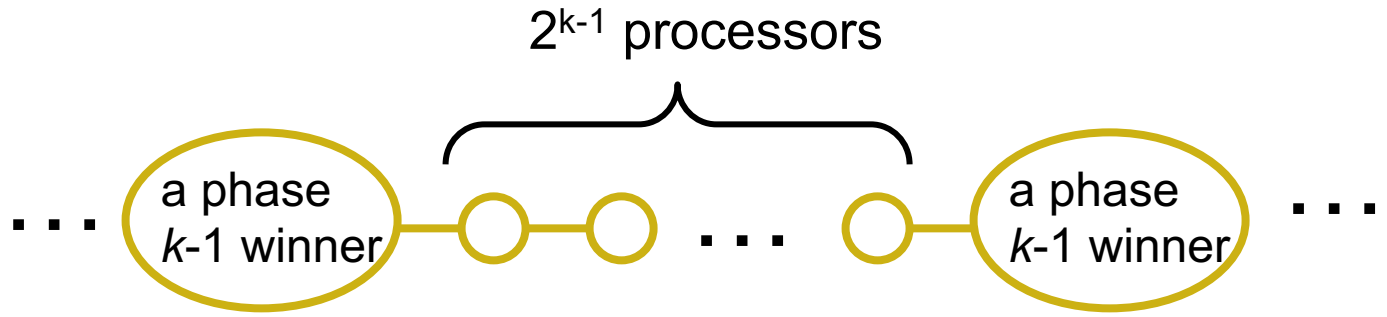
- **Correctness:** Similar to $O(n^2)$ algorithm.
- **Message Complexity:**
  - Each message belongs to a particular phase and is initiated by a particular proc.
  - Probe distance in phase k is 2^k
  - Number of messages initiated by a proc. in phase k is at most $4*2^k$ (probes and replies in both directions)

Analysis of O(n log n) Algo

- How many processes initiate probes in phase *k*?
  - For *k = 0*, every process does
  - For *k > 0*, every process that is a "winner" in phase *k - 1* does
  - "winner" means has largest id in its 2^{k-1} neighborhood

Analysis of O(n log n) Algo

- Maximum number of phase *k - 1* winners occurs when they are packed as densely as possible:



- The total number of phase *k - 1* winners is at most

$$n/(2^{k-1} + 1)$$

Analysis of O(n log n) Algo

- How many phases are there?
- At each phase the number of (phase) winners is cut approx. in half
  - from $n//(2^{k-1} + 1)$ to $n//(2^k + 1)$
- So after approx. $\log_2 n$ phases, only one winner is left.
  - more precisely, max phase is $\lceil \log(n-1) \rceil + 1$

Analysis of O(n log n) Algo

- Total number of messages is sum, over all phases, of number of winners at that phase times number of messages originated by that winner:

$$\leq \underbrace{4n}_{\text{phase 0 msgs}} + \underbrace{n}_{\text{termination msgs}} + \underbrace{\sum_{k=1}^{\lceil \log(n-1) \rceil + 1} 4 \bullet 2^k \bullet n/(2^{k-1}+1)}_{\text{msgs for phases 1 to } \lceil \log(n-1) \rceil + 1}$$

$$< 8n(\log n + 2) + 5n$$

$$= O(n \log n)$$

Can We Do Better?

- The $O(n \log n)$ algorithm is more complicated than the $O(n^2)$ algorithm but uses fewer messages in the worst case.
- Works in both synchronous and asynchronous case.
- Can we reduce the number of messages even more?
- Not in the asynchronous model … !!

Summary

- Termination Detection
  - Definition
  - TD using Global Snapshots
- Leader Election in Rings
  - Leader Election Problem
  - Formulation of the problem
  - LCR algorithm
  - O(nlogn) algorithm using probes
  - Complexity Analysis
- 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

Contact Information

Dr. Rajendra Prasath
IIIT Sri City, Chittoor

Spring 2024

Distributed Computing

- Topology Abstraction and Overlays

Dr. Rajendra Prasath

Indian Institute of Information Technology Sri City, Chittoor

28th February 2024 (http://rajendra.2power3.com)

> Distributed Computing?

- How will you design a Distributed Algorithm?

- Learn to Solve using Distributed Algorithms

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
- Concurrent Events
- Causal Ordering
- Logical Clocks vs Physical Clocks
- Global Snapshot Detection
- Termination Detection Algorithm
- Leader Election in Rings

- [Now] Topology Abstraction and Overlays …

Recap

> About this Lecture

What do we learn today?

- Recap: Leader Election in Rings
  - LCR algorithm
- Topology Abstraction and Overlays
  - Interconnection Topologies
  - Abstraction - Basic Concepts
  - Interconnection Patterns suitable for message propagation
  - Types of Algorithms and their executions
  - Measures and Metrics

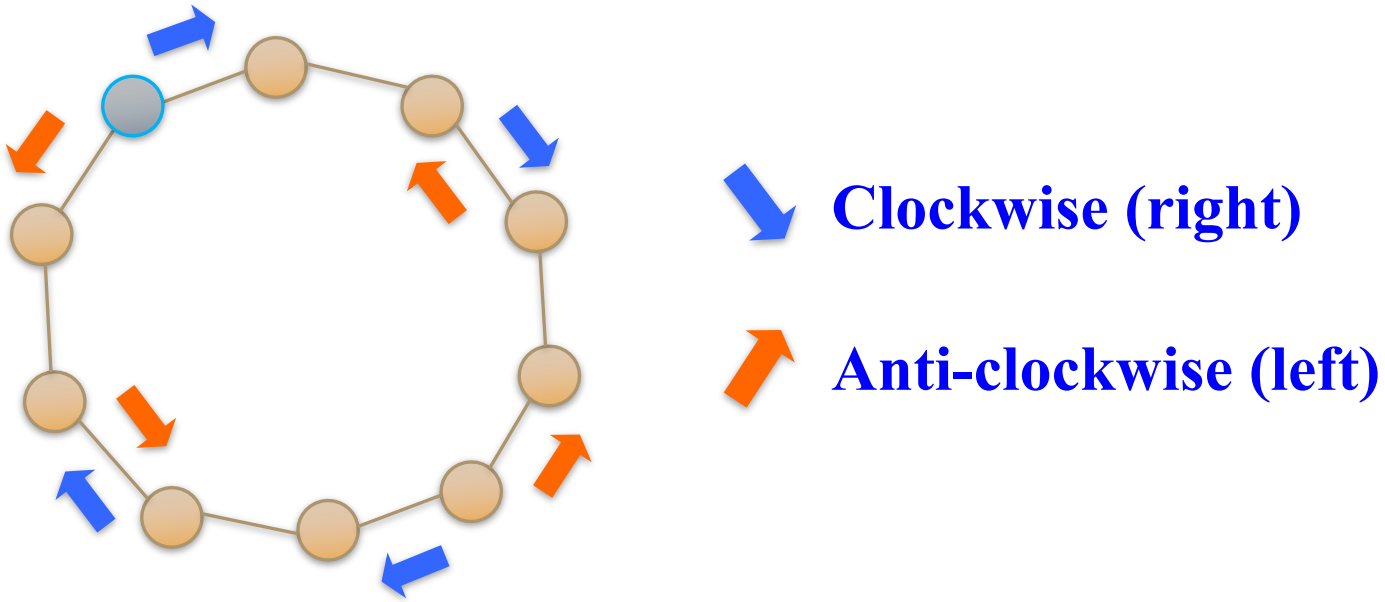
Let us explore these topics ➔ ➔ ➔

Today's Focus

Topology Abstraction and Overlays in Distributed Systems

Ring Networks

- In an oriented ring, processes have a consistent notion of left and right



- For example, if messages are forwarded on right channel, they will cycle clockwise around the ring

Why Study Rings?

- Simple starting point, easy to analyze
- Abstraction of a token ring
- Lower bounds and impossibility results for ring topology also apply to arbitrary topologies

Interconnection Topologies

- Various Interconnection Networks
  - Abstraction of the overall networks
  - Message Propagation
  - Distributed Processing
  - Computational Complexity
- Overlays
  - Sampling the underlying network topology

Basic Terminologies

- System Model: Undirected (weighted) graph G = (V, E), where n = |V|
  - Model the underlying topology in such a way that the pattern of message passing / communication could be efficiently handled
  - Easy to maintain and apply logics on the abstraction of the underlying topology

Physical Topology

- Physical topology
  - Nodes: network nodes, routers, all end hosts (whether participating or not)
  - Edges: all LAN, WAN links, direct edges between end hosts

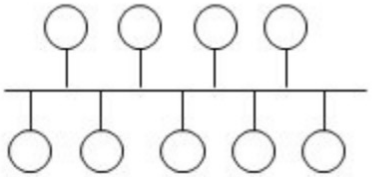
Logical Topology

- Logical topology (application context)
  - Nodes: end hosts where application executes
  - Edges: logical channels among these nodes
  - Fully connected or any subgraph – partial system view, needs multi-hop paths

Superimposed Topology

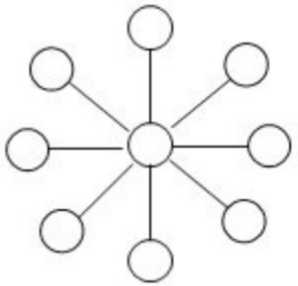
- Superimposed topology (also called as "topology overlay"):
  - superimposed on logical topology
  - Goal: efficient information gathering, distribution, or search (as in P2P overlays)
  - Examples: ring, tree, mesh, hypercube

Interconnection Topologies

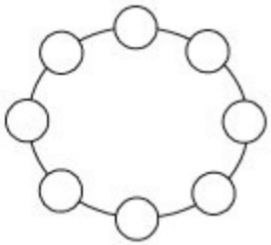


a) Bus

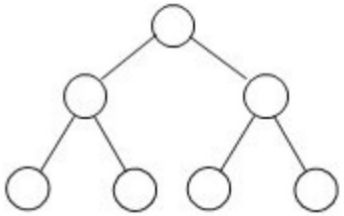
b) Linear array



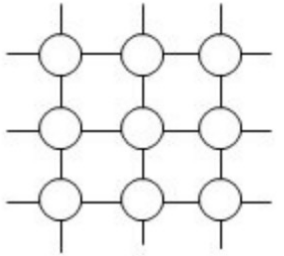
c) Star



d) Ring

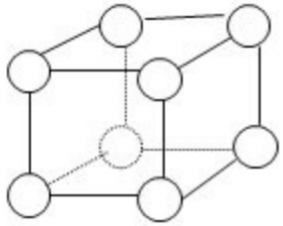


e) Tree



f) Near-neighbor mesh

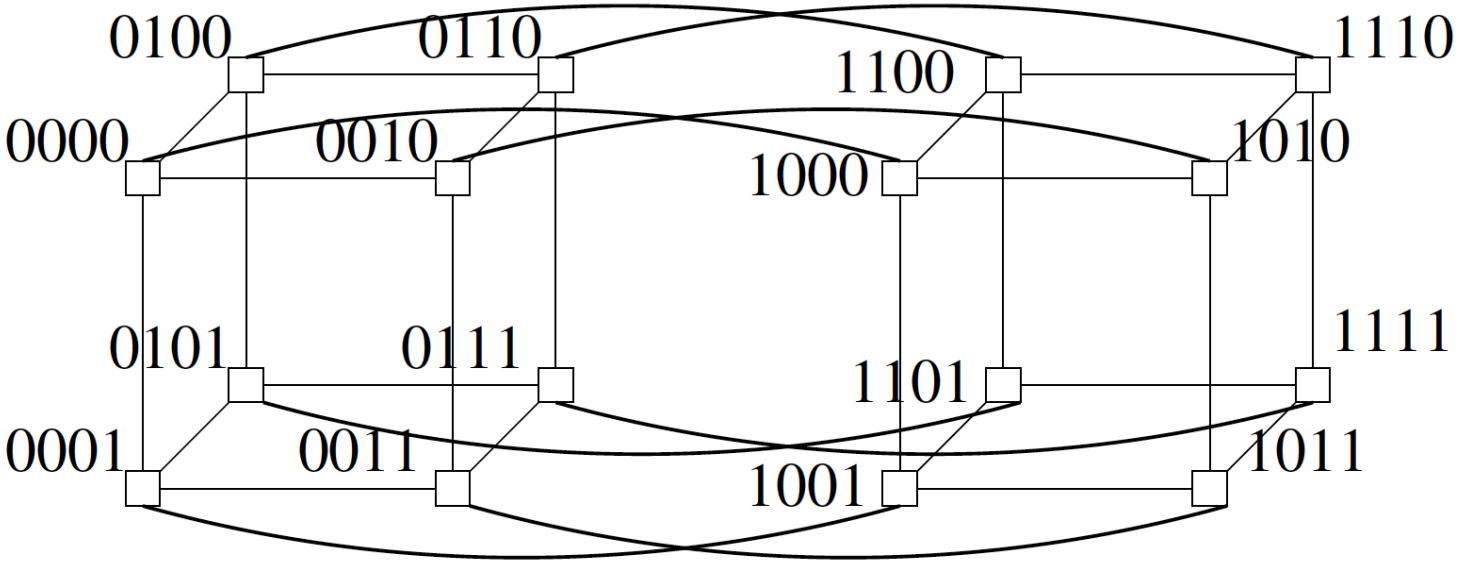
g) Completely connected



h) 3–cube (hypercube)

Interconnection Topologies (contd)

➔ Hypercube of Dimension 4



➔ k-ary d-cudes (generalized version)

Basic Concepts

- Application execution vs. control algorithm execution, each with own events
- Control algorithm:
  - for monitoring and auxiliary functions, e.g., reaching consensus, global state detection (deadlock, termination etc.), check pointing and
  - superimposed on application execution, but does not interfere
  - its send, receive, internal events are transparent to application execution

Classifications

- Centralized and distributed algorithms
  - Centralized: asymmetric roles; client-server configuration; processing and bandwidth bottleneck; point of failure
  - Distributed: more balanced roles of nodes, difficult to design perfectly distributed algorithms (e.g., snapshot algorithms, tree-based algorithms)
- Symmetric and asymmetric algorithms

Important Concepts

- **Anonymous algorithm:**
  - Process ids are not used to make any execution (run-time) decisions
  - Structurally elegant but hard to design, or impossible, (anonymous leader election is impossible)
- **Uniform algorithm:**
  - Cannot use n, the number of processes, as a parameter
  - Allows scalability; process leave/join is easy and only neighbors need to be aware of logical topology changes
- **Adaptive algorithm:**
  - Let k ($\leq n$) be the number of processes participating in the context of a problem X when X is being executed.
    Complexity should be expressible as a function of k, not n
  - For example, Mutual Exclusion is a good example

Deterministic vs Nondeterministic

Deterministic vs. nondeterministic executions

- ➔ Nondeterministic exec: contains at least 1 nondeterministic receive; deterministic execution has no nondeterministic receive
  - ➔ Nondeterministic receive: can receive a message from any source
  - ➔ Deterministic receive: source is specified

Difficult to reason with

- ➔ Asynchronous system: re-execution of deterministic program will produce same partial order on events ((used in debugging, unstable predicate detection etc.)
- ➔ Asynchronous system: re-execution of nondeterministic program may produce different partial order (unbounded delivery times and unpredictable congestion, variable local CPU scheduling delays)

Synchronous vs Asynchronous

Synchronous:

- ➔ Upper bound on message delay
- ➔ Known bounded drift rate of clock with respect to the real time
- ➔ Known upper bound for process to execute a logical step

Asynchronous: above criteria not satisfied

- ➔ Spectrum of models in which some combo of criteria satisfied
- ➔ Algorithm to solve a problem depends greatly on this model

Distributed systems are inherently asynchronous

Algorithms / Channels

- Wait-free algorithms (for synchronization operations)
  - resilient to n - 1 process failures, i.e., operations of any process must complete in bounded number of steps, irrespective of other processes
  - very robust, but expensive
  - possible to design for mutual exclusion
  - may not always be possible to design, for example, the producer-consumer problem
- Communication channels
  - point-to-point: FIFO, non-FIFO
  - At application layer, FIFO usually provided by network stack

Process failures (Sync + Async syst.)

- Fail-stop: Properly functioning process stops execution. Other processes learn about the failed process (thru some mechanism)
- Crash: Properly functioning process stops execution. Other processes do not learn about the failed process
- Receive omission: Properly functioning process fails by receiving only some of the messages that have been sent to it, or by crashing.
- Send omission: Properly functioning process fails by sending only some of the messages it is supposed to send, or by crashing. Incomparable with receive omission model.
- General omission: Send omission + receive omission
- Byzantine (or malicious) failure: Process may (mis) behave anyhow, including sending fake messages. Authentication facility => If a faulty process claims to have received a message from a correct process, that is verifiable.

Process Failures (contd.)

- Process failures → Timing failures (sync systems):
  - General omission failures, or process violating bounds on time to execute a step
  - More severe than general omission failures

Failure models influence design of algorithms

- Link failures
  - Crash failure: Properly functioning link stops carrying messages
  - Omission failure: Link carries only some of the messages sent on it, not others
  - Byzantine failure: Link exhibits arbitrary behavior, including creating fake and altering messages sent on it
- Link failures → Timing failures (sync systems):
  - messages delivered faster/slower than specified behavior

Complexity Measures

- Each metric specified using
  - lower bound (Omega)
  - upper bound (big O)
  - exact bound(Theta)

Metrics

- Space complexity per node
- System-wide space complexity (= n space complexity per node). E.g., worst case may never occur at all nodes simultaneously!
- Time complexity per node
- System-wide time complexity. Do nodes execute fully concurrently?

Metrics

- Message complexity
  - Number of messages (affects space complexity of message overhead)
  - Size of messages (affects space complexity of message overhead + time component via increased transmission time)
  - Message time complexity: depends on number of messages, size of messages, concurrency in sending and receiving messages
- Other metrics: # send and # receive events; # multicasts, and implementation related metrics?
- (Shared memory systems): size of shared memory; # synchronization operations

Summary

- Racap: Leader Election in Rings
  - Leader Election Problem
  - LCR algorithm / O(nlogn) algorithm using probes
- Topology Abstraction and Overlays
  - Various Interconnection Topologies
  - Abstraction - Basic Concepts
  - Interconnection Patterns suitable for message propagation
  - Types of Algorithms and their executions
  - Measures and Metrics
    - 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

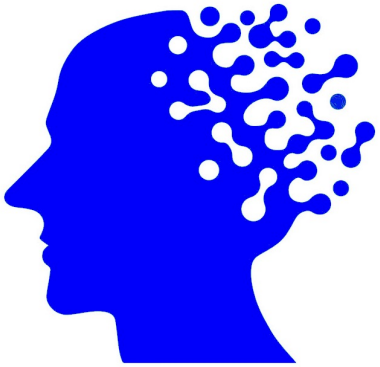
- You may leave me an email any time
  (email is the best way to reach me faster)

Questions
It's Your Time

Contact Information

Dr. Rajendra Prasath
IIIT Sri City, Chittoor

Spring 2024



Distributed Computing

- Message Ordering and Group Communication

Dr. Rajendra Prasath

Indian Institute of Information Technology Sri City, Chittoor

06-07th March 2024 (http://rajendra.2power3.com)

> Distributed Computing?

- How will you design a Distributed Algorithm?

- Learn to Solve using Distributed Algorithms

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
- Concurrent Events
- Causal Ordering
- Logical Clocks vs Physical Clocks
- Global Snapshot Detection
- Termination Detection Algorithm
- Leader Election in Rings

- [Now] Topology Abstraction and Overlays …

Recap

> About this Lecture

What do we learn today?

- **Recap: Topology Abstraction and Overlays**
  - Interconnection Topologies
  - Interconnection Patterns suitable for message propagation

- **Message Ordering and Group Communication**
  - Models of Communication
  - Design Issues / Failures
  - Multiple Unicasts
  - Good / Bad Ordering
  - Causal Ordering of Messages

Let us explore these topics ➔ ➔ ➔

Today's Focus

Message Ordering / Group Communication

Models of Communication

- One – to – One
  - Unicast
    - 1 – 1
    - Point – to – point
  - Anycast
    - 1 – nearest 1 of several identical nodes
- One – to – Many
  - Multicast
    - 1 – many
    - Group Communication
  - Broadcast
    - 1 – All

Groups

- Why groups?
  - Groups allow us to deal with a collection of processes as one abstraction
- Send message to one entity
  - Deliver to entire group
- Groups are dynamic
  - Created and destroyed
  - Processes can join or leave
    - May belong to 0 or more groups
- Primitives
  - join_group, leave_group, send_to_group, query_membership

Design Issues

Closed vs. Open

- ➔ Closed: only group members can sent messages

Peer vs. Hierarchical

- ➔ Peer: each member communicates with group
- ➔ Hierarchical: go through dedicated coordinator(s)
- ➔ Diffusion: send to other servers & clients

Managing membership & group creation/deletion

- ➔ Distributed vs. centralized

Leaving & joining must be synchronous

Fault tolerance

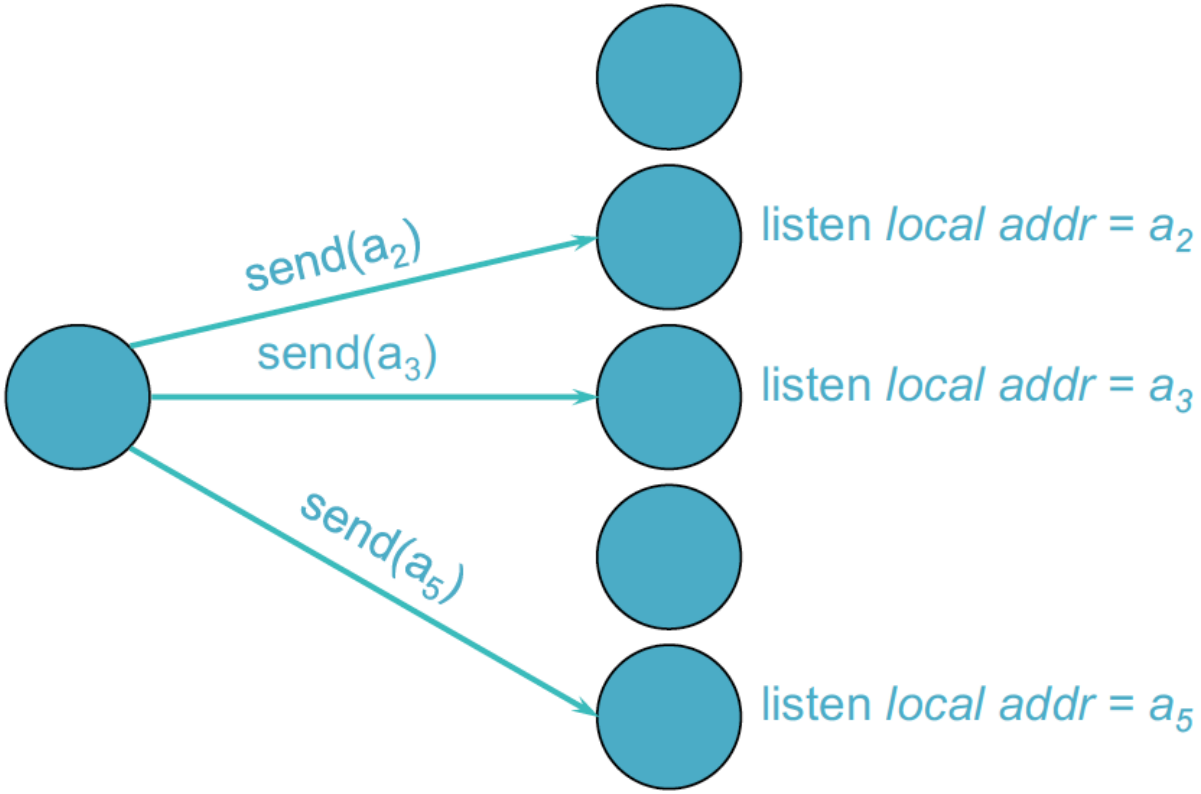
- ➔ Reliable message delivery? What about missing members?

Failures

- Crash failure
  - Process stops communicating
- Omission failure (typically due to network)
  - Send omission: A process fails to send messages
  - Receive omission: A process fails to receive messages
- Byzantine Failure
  - Some messages are faulty, including sending fake messages
- Partition Failure
  - The network may get segmented, dividing the group into two or more unreachable sub-groups

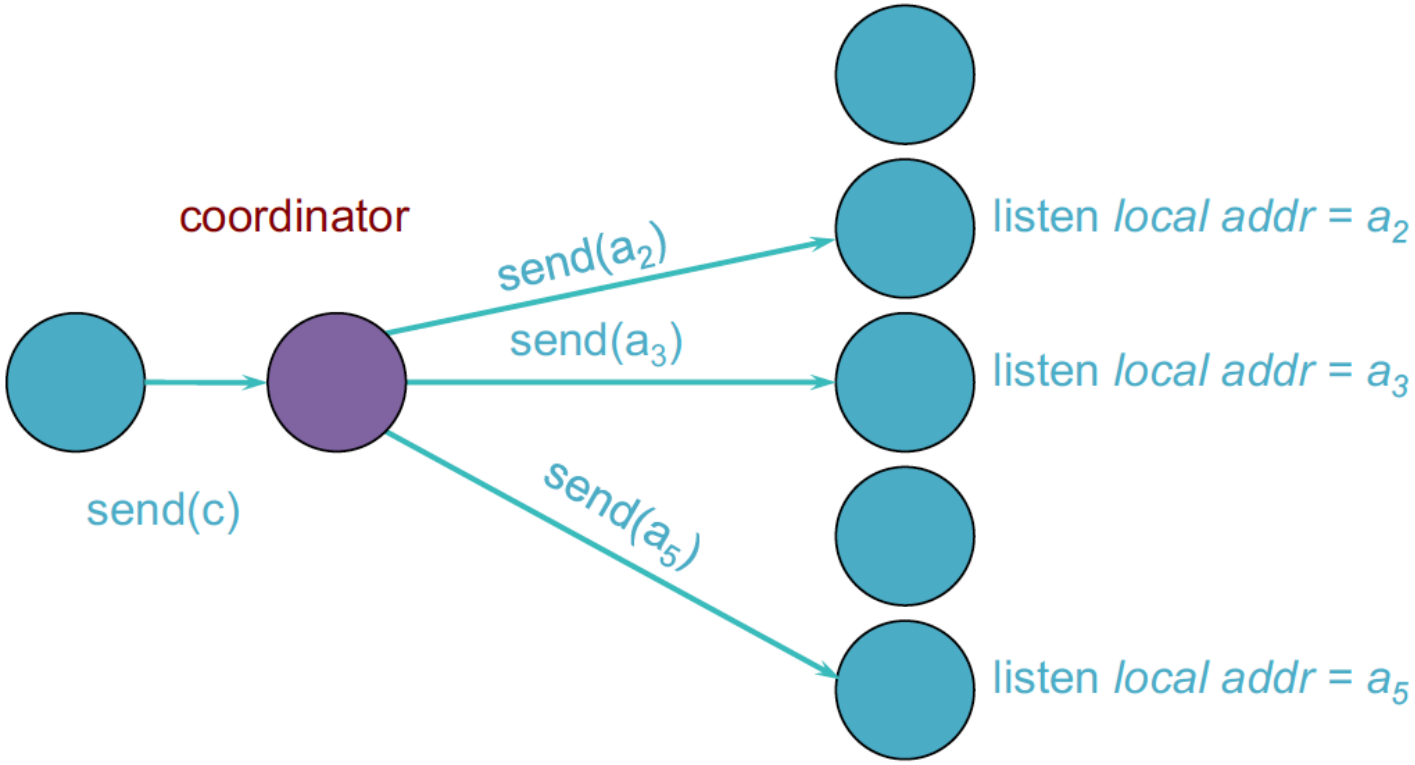
Multiple Unicasts

➔ Sender knows Group members



Hierarchical

- Multiple unicasts via group coordinator
  - coordinator knows group members



Atomic Multicast

- Atomicity
  - Message sent to a group arrives at all group members
    - If it fails to arrive at any member, no member will process it
- Problems
  - Unreliable network
    - Each message should be acknowledged
    - Acknowledgements can be lost
  - Message sender might die

How to achieve Atomicity?

- General Idea
  - Ensure that every recipient acknowledges receipt of the message
  - Only then allow the application to process the message
  - If we give up on a recipient then no recipient can process the received message
- Easier said than done!
  - What if a recipient dies after acknowledging the message?
    - Is it obligated to restart?
    - If it restarts, will it know to process the message?
  - What if the sender (or coordinator) dies partway through the protocol?

Achieving Atomicity – An Example

Retry through network failures & system downtime

- ➔ Sender & receivers maintain a persistent log
- ➔ Each message has a unique ID so we can discard duplicates
- ➔ Sender
  - sends the message to all group members
  - Writes the message to log
  - Waits for acknowledgement from each group member
  - Writes the acknowledgement to log
  - If timeout on waiting for an acknowledgement, retransmit to group member
- ➔ Receiver logs received non-duplicate message to persistent log and sends an acknowledgement

NEVER GIVE UP! – Assume that dead senders or receivers will be rebooted and will restart where they left off

Reliable multicast

All non-faulty group members will receive a message

- Assume sender & recipients will remain alive
- Network may have glitches
  - Retransmit undelivered messages

Acknowledgements

- Send message to each group member
- Wait for acknowledgement from each group member
- Retransmit to non-responding members
- Subject to feedback implosion

Negative acknowledgements

- Use a sequence number on each message
- Receiver requests retransmission of a missed message
- More efficient but requires sender to buffer messages indefinitely

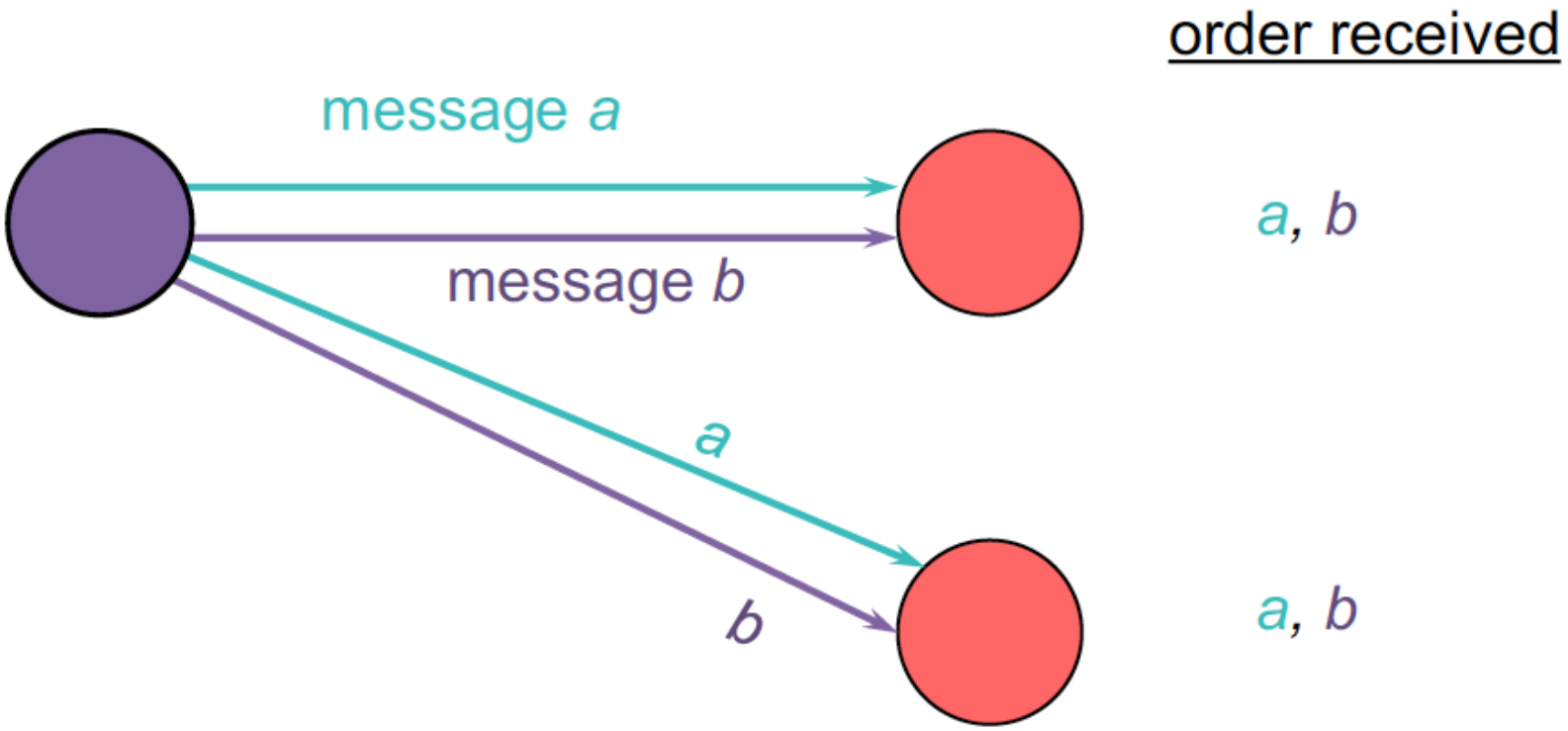
Acknowledgements

- Easiest thing is to wait for an ACK before sending the next message
  - But that incurs a round-trip delay
- Optimizing
  - Pipelining
    - Send multiple messages – receive ACKs asynchronously
    - Set timeout – retransmit message for missing ACKs
  - Cumulative ACKs
    - Wait a little while before sending an ACK
    - If you receive others, then send one ACK for everything
  - Piggybacked ACKs
    - Send an ACK along with a return message
- TCP does all of these … but now we have to do this on each recipient

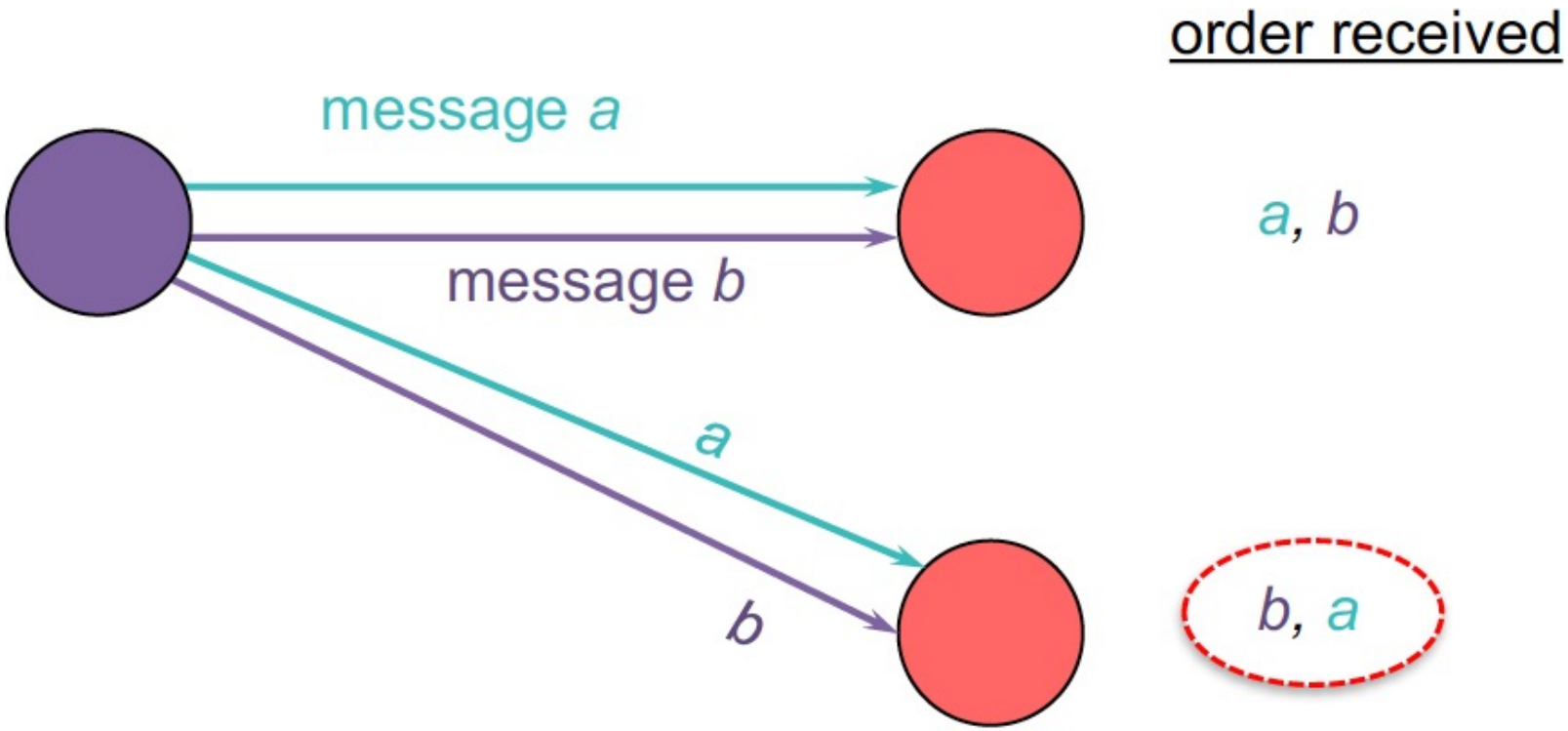
Message Ordering

- How to order messages?
  - Send vs Delivery
  - Global Time Ordering
  - Total Ordering
  - Causal Ordering
  - Sync Ordering
  - FIFO Ordering
  - Unordered multicast
- Good / Bad Ordering

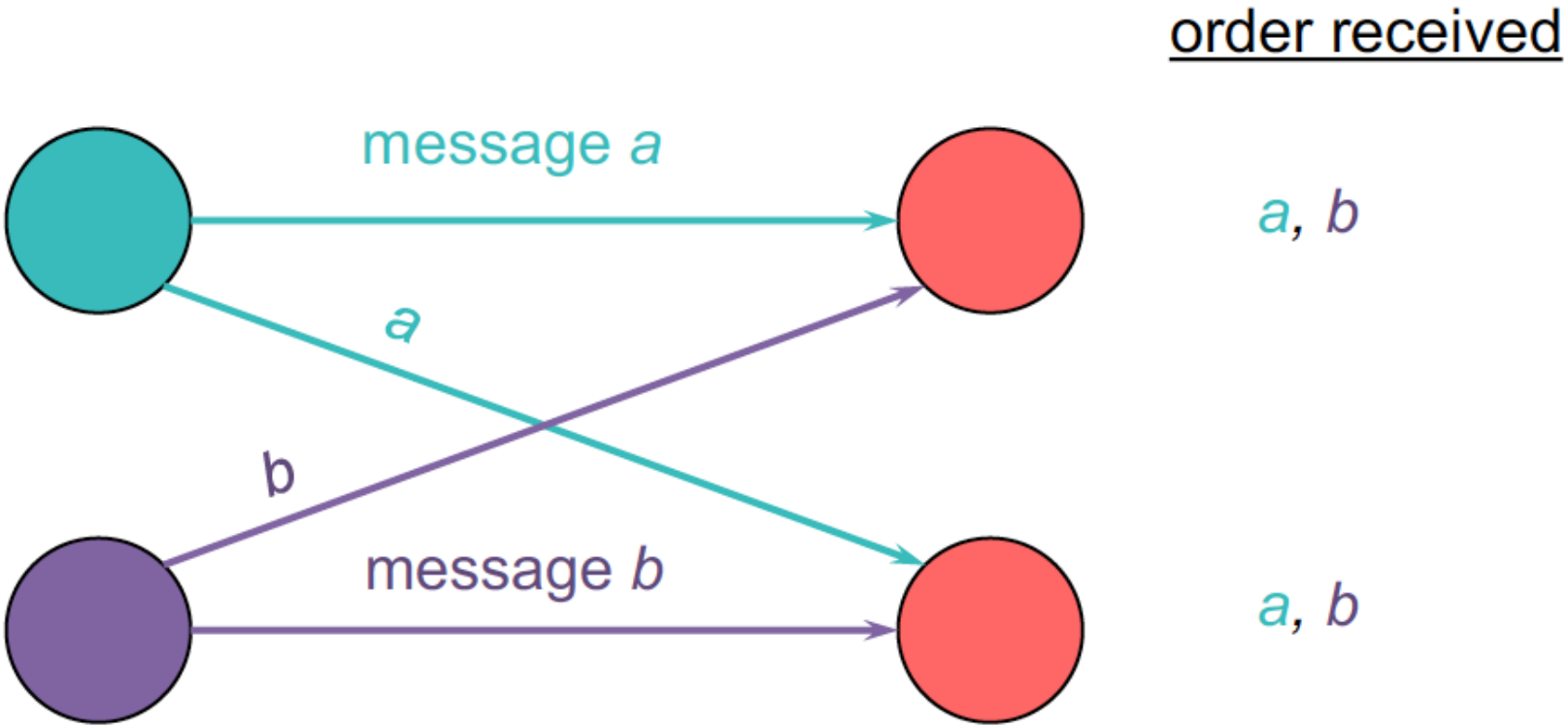
Good Ordering



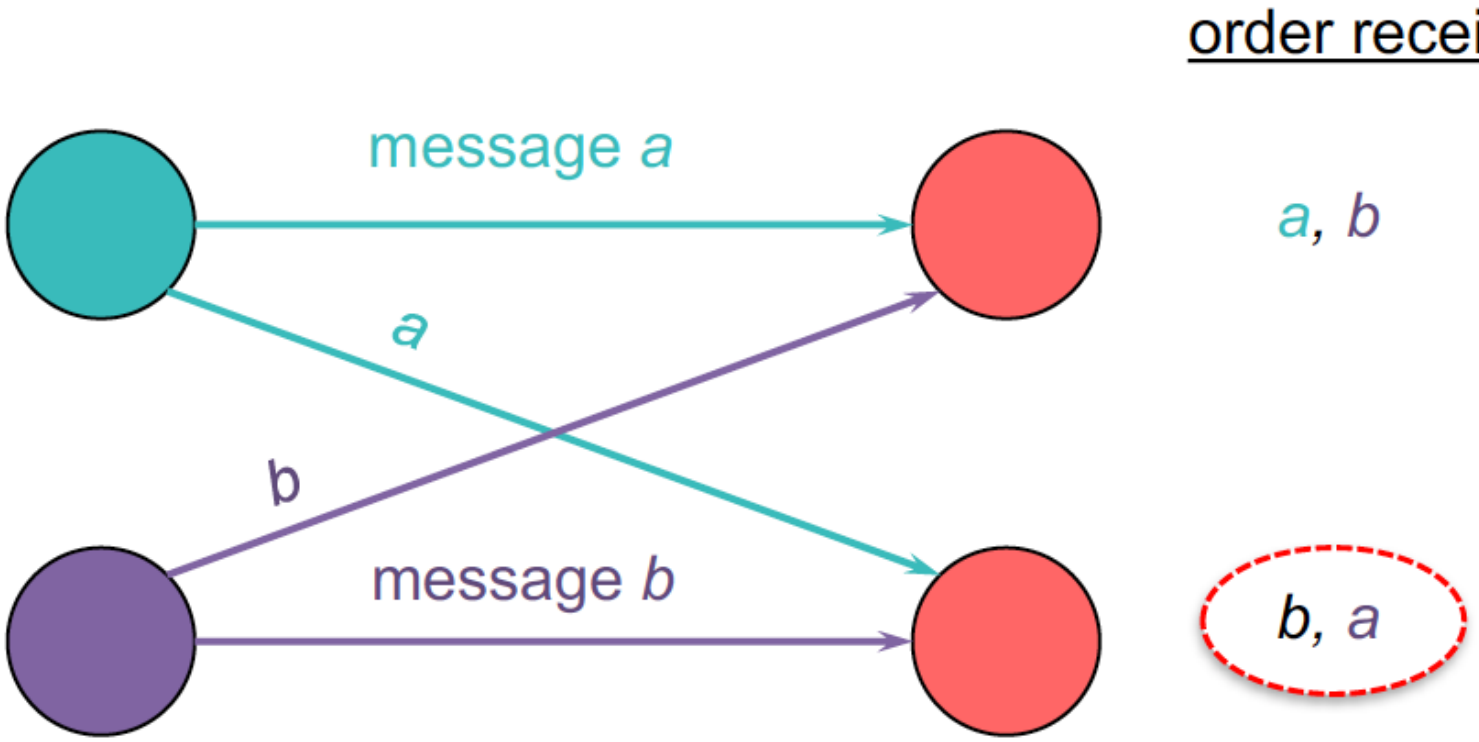
Bad Ordering



Good Ordering



Bad Ordering



Send vs. Delivery of Messages

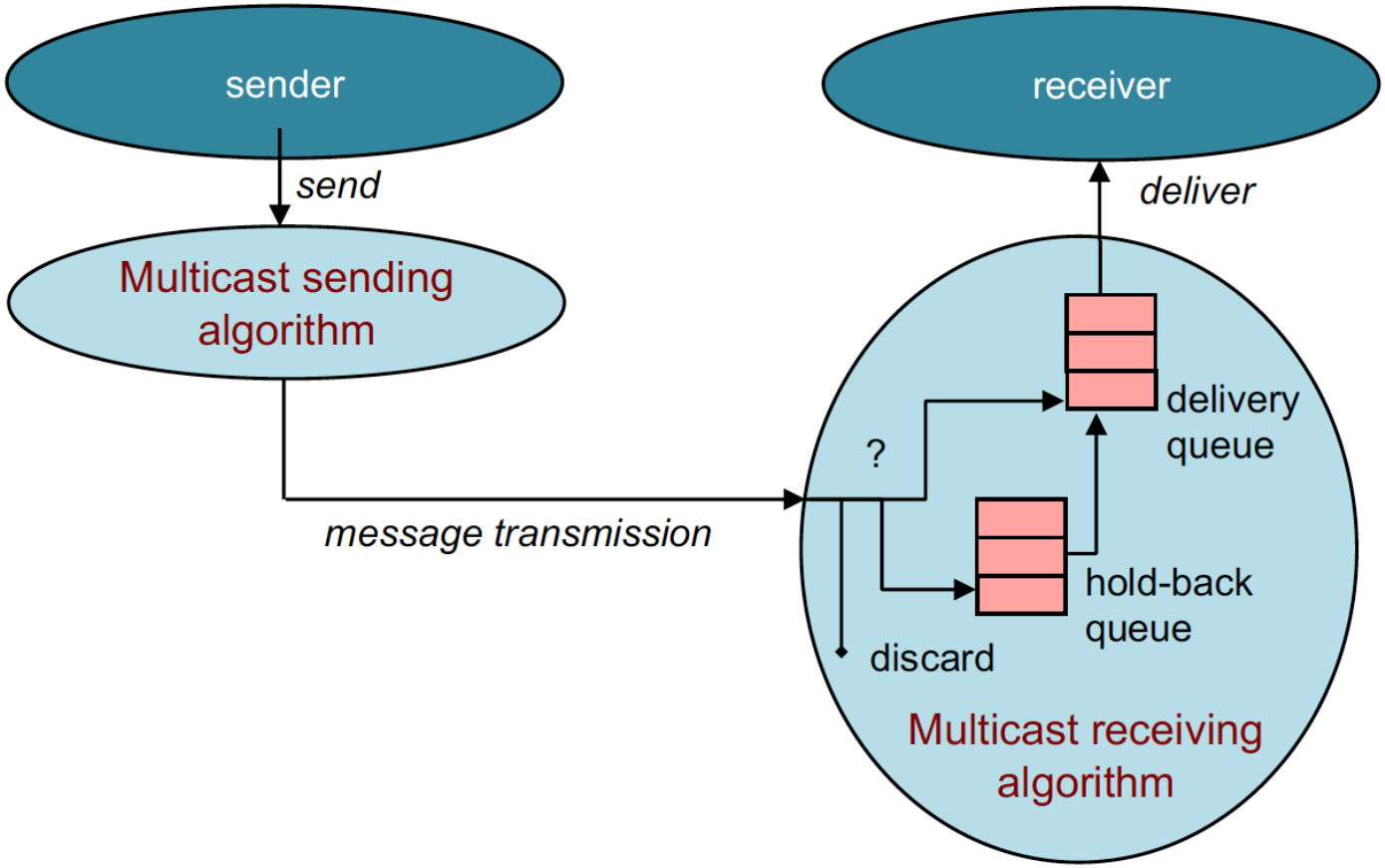
- Multicast receiver algorithm decides when to deliver a message to a process

A received message may be:

- delivered immediately (put on a delivery queue that the process reads)
- placed on a hold-back queue (because we need to wait for an earlier message)
- rejected/discarded (duplicate or earlier message that we no longer want)

An Illustration

- Sending, delivering and holding back



Global Time Ordering

- All messages arrive in exact order sent
- Assumes that two events never happen at exactly the same time!
  - Why Not? No global clocks … right?
- Difficult (impossible) to achieve

Total Ordering

- Consistent ordering everywhere
- All messages arrive at all group members in the same order
  - They are sorted in the same order in the delivery queue

Two Conditions:

- If a process sends m before m′ then any other process that delivers m′ will have delivered m
- If a process delivers m′ before m″ then every other process will have delivered m′ before m″

Total Ordering - Implementation

- How to implement this?
  - Attach unique totally sequenced message ID
  - Receiver delivers a message to the application only if it has received all messages with a smaller ID

Causal Ordering

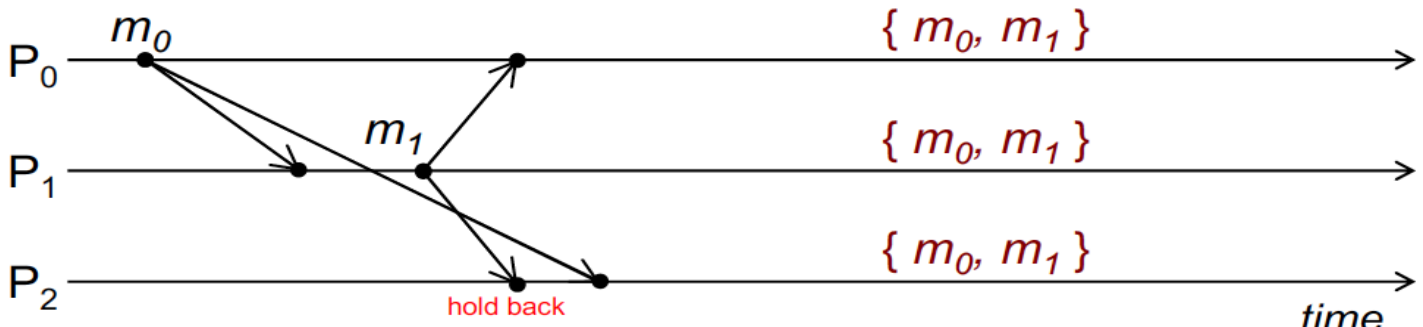
- Partial ordering
  - Messages sequenced by Lamport or Vector timestamps

Condition:

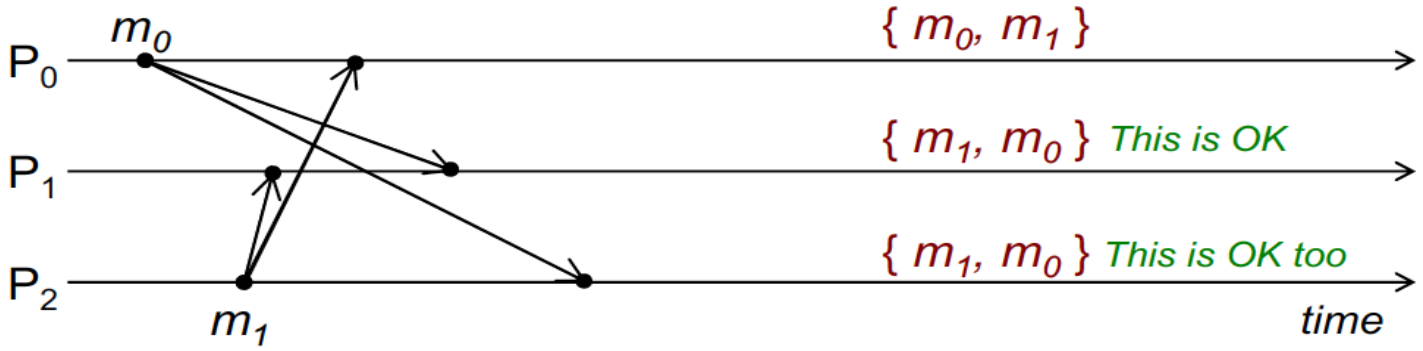
- If multicast(G, m) → multicast(G, m′)
  - then every process that delivers m′ will have m delivered already

- If message m′ is causally dependent on the message m, then all processes must deliver m before m′

Causal vs Concurrent



m_1 is causally dependent on the receipt of m_0.
Hence, m_1 must be delivered after m_0 has been delivered.



m_0 and m_1 have no causal relationship (they are concurrent).
Any process can deliver them in any order.

Causal Ordering - Implementation

How to implement CO?

- P_i receives a message from P_j

- Each process keeps a precedence vector (similar to vector timestamp)
- Vector is updated on multicast send and receive events
  - Each entry = number of the latest message from the corresponding group member that causally precedes the event

Causal Ordering - Algorithm

- When P_j sends a message, it increments its own entry and sends the vector
  - $V_j[j] = V_j[j] + 1$
  - Send V_j with the message
- When P_i receives a message from P_j
  - Check that the message arrived in FIFO order from Pj : $V_j[j] == V_i[j] + 1$?
  - Check that the message does not causally depend on something P_i has not seen

    $$\forall k, k \neq j: V_j[k] \leq V_i[k] \ ?$$

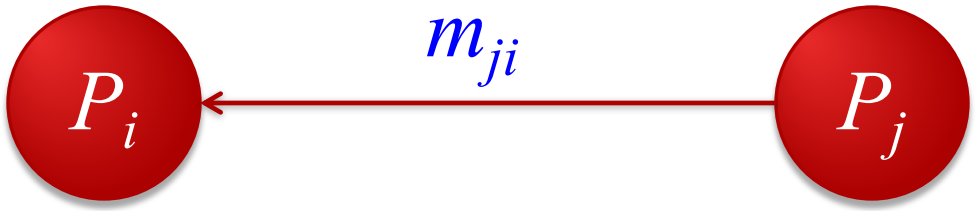
  - If both conditions are satisfied, P_i will deliver the message
  - Otherwise, hold the message until the conditions are satisfied

Causal Ordering – Work out

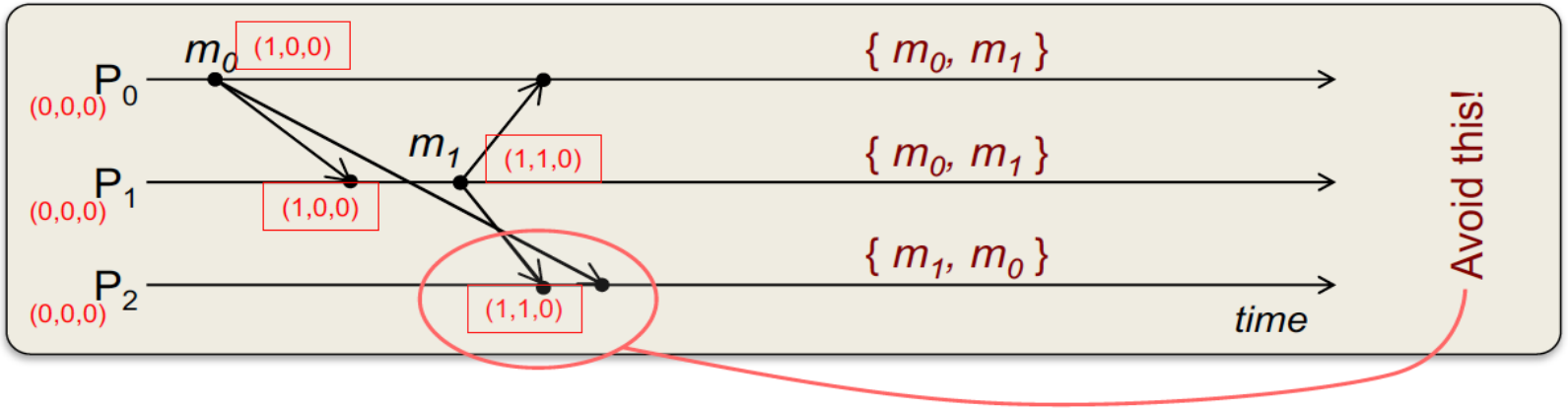
- Implementation:

  P_i receives a message from P_j

- Each process keeps a precedence vector (similar to vector timestamp)
- Vector is updated on multicast send and receive events
  - Each entry = Number of the latest message from the corresponding group member that causally precedes the event message

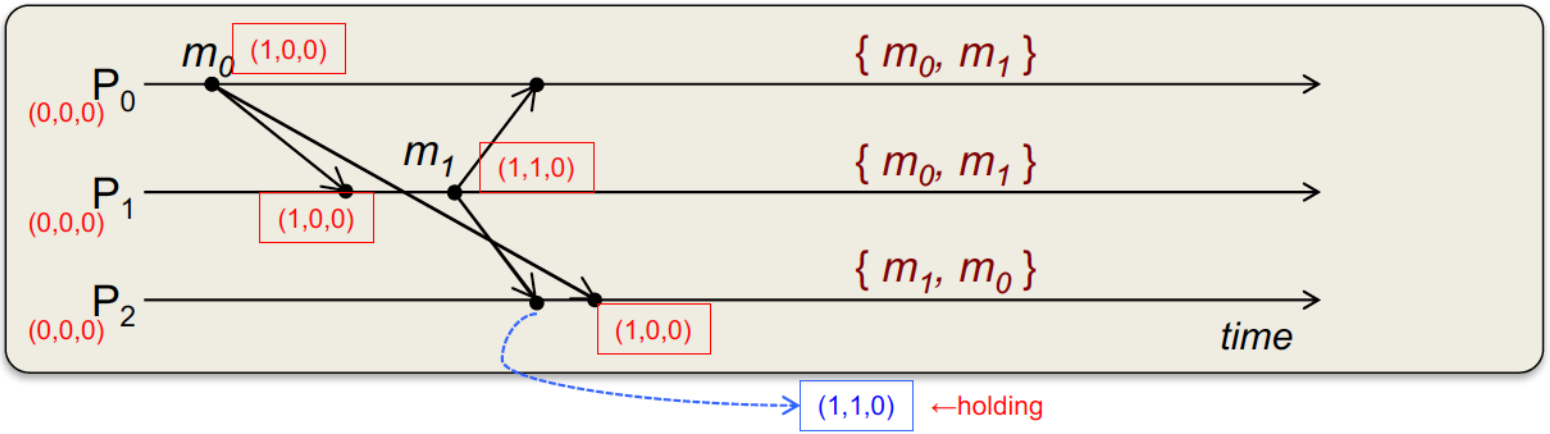


Causal Ordering – Example



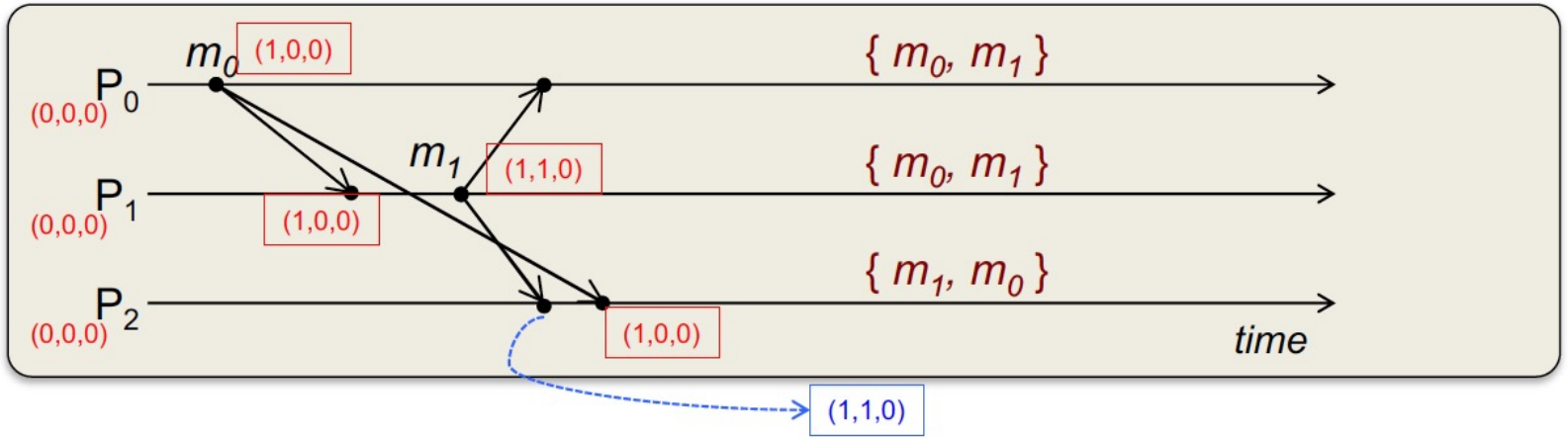
- P_2 receives message m1 from P_1 with V_1=(1,1,0)
- Is this in FIFO order from P_1?
  - Compare current V on P_2: V2=(0,0,0) with received V from P_1, V_1=(1,1,0)
  - Yes: $V_2[1] = 0$, received $V_1[1] = 1 \Rightarrow$ sequential order
- Is $V_1[i] \leq V_2[i]$ for all other i?
  - Compare the same vectors: V_2=(0,0,0) vs. V_1=(1,1,0)
  - No. $V_1[0] > V_2[0]$ (1 > 0)
  - Therefore: hold back m_1 at P_2

Causal Ordering – Example (contd)



- P_2 receives message m_0 from P_0 with V=(1,0,0)
- (1) Is this in FIFO order from P0?
  - Compare current V on P_2: V_2=(0,0,0) with received V from P_2, V_2=(1,0,0)
  - Yes: $V_2[0] = 0$, received $V_1[0] = 1 \Rightarrow$ sequential
- (2) Is $V_0[i] \leq V_2[i]$ for all other i?
  - Yes
- Deliver m_0
  - Now check hold-back queue. Can we deliver m_1?

Causal Ordering – Example (contd)



- Is the held-back message m_1 in FIFO order from P_0?
  - Compare current V on P_2: V_2=(1,0,0) with held-back V from P_0, V_1=(1,1,0)
  - Yes: $V_2[1] = 0$, received $V_1[1] = 1 \Rightarrow$ sequential
- Is V0[i] ≤ V2[i] for all other i?
  - Now yes. Element 0: (1 ≤ 1), element 2: (0 ≤ 0); Deliver m_1
- More efficient than total ordering:
  - No need for a global sequencer.
  - No need to send acknowledgements.

Sync Ordering

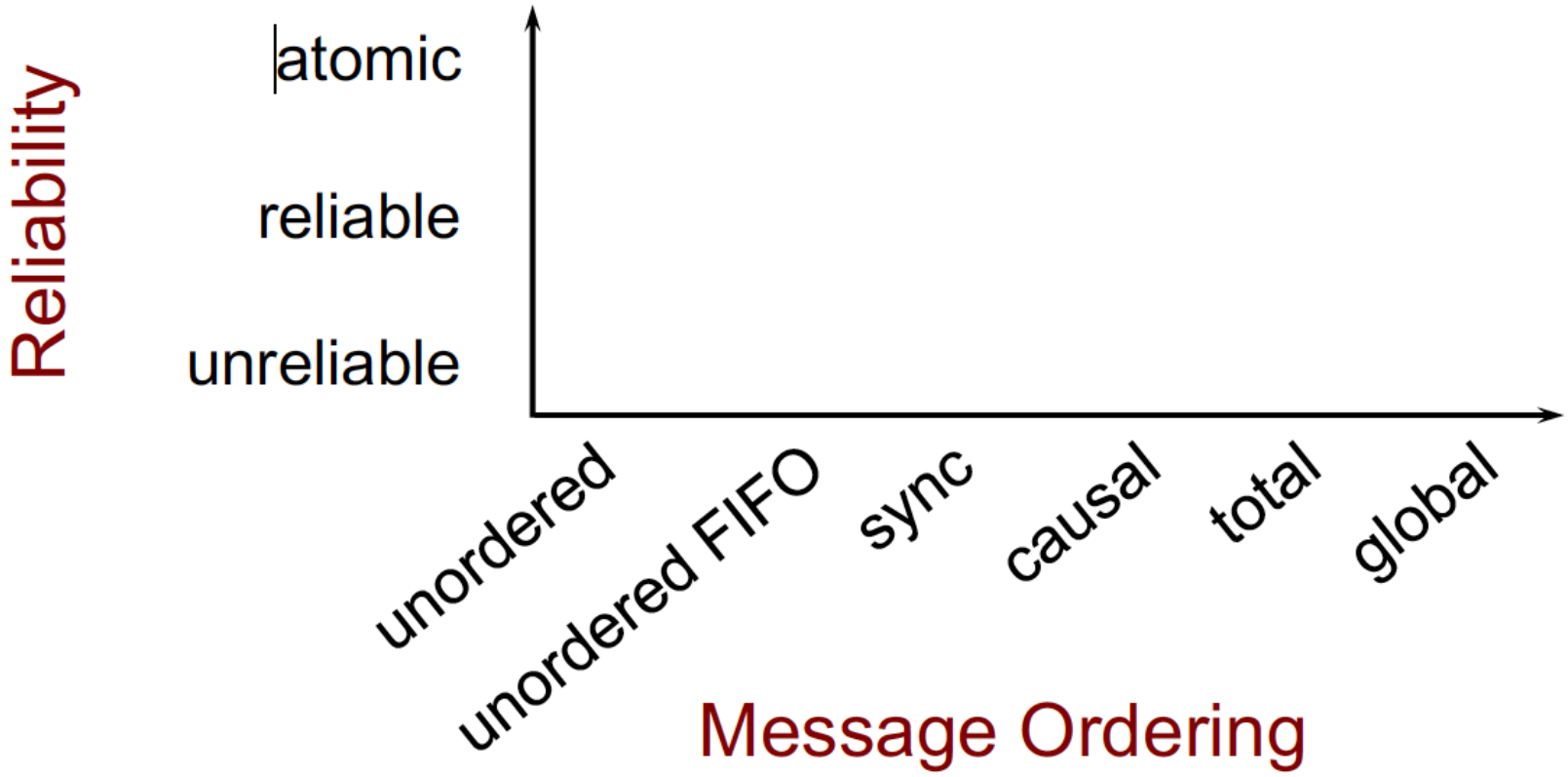
- Messages can arrive in any order
- Special message type
  - Synchronization primitive
  - Ensure all pending messages are delivered before any additional (post-sync) messages are accepted

Unordered multicast

- Messages can be delivered in different order to different members
- Order per-source does not matter

Multicast Considerations

- Follow this order !!



Summary

- Racap: Topology Abstraction and Overlays
  - Various Interconnection Topologies
  - Interconnection Patterns suitable for message propagation
- Communication Models
  - Message Ordering & Group Communications
  - Design Issues
    - Process Failures
  - Message Ordering
    - Good / Bad ordering
    - Various Types of Ordering of messages
  - Group Communication
    - Causal ordering based approach

- 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

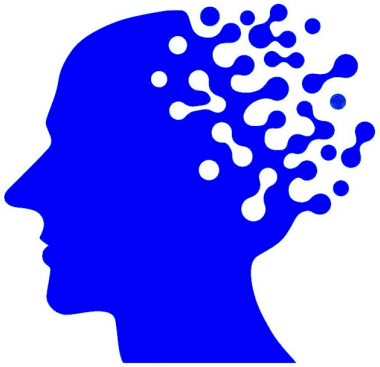
Best Approach

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

Questions
It's Your Time

Contact Information

Dr. Rajendra Prasath
IIIT Sri City, Chittoor



Spring 2024

Distributed Computing

- Basics of Mutual Exclusion algorithms

Dr. Rajendra Prasath

Indian Institute of Information Technology Sri City, Chittoor

11th March 2024 (http://rajendra.2power3.com)

> Distributed Computing?

- How will you design a Distributed Algorithm?

- Learn to Solve using Distributed Algorithms

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
- Concurrent Events / Causal Ordering
- Logical Clocks vs Physical Clocks
- Global Snapshot Detection
- Termination Detection Algorithm
- Leader Election in Rings
- Topology Abstraction and Overlays
- Message Ordering and Group Communication

- Mutual Exclusion Algorithm

[Now] → → →

Recap

> About this Lecture

What do we learn today?

- Recap: Message Ordering and GC
  - Models of Communication
  - Design Issues / Good / Bad Ordering

- Mutual Exclusion Algorithms
  - Centralized Algorithm
  - Token-Based / Permission-Based Algorithms
  - Quorum-Based Algorithm
  - Tree-Based Algorithm

Let us explore these topics ➔ ➔ ➔

Today's Focus

Distributed Mutual Exclusion Algorithms

Why do we need MutEx?

- Mutual Exclusion
  - Operating systems: Semaphores
    - In a single machine, you could use semaphores to implement mutual exclusion
    - How to implement semaphores?
      - Inhibit interrupts
      - Use clever instructions (e.g. test-and-set)
    - On a multiprocessor shared memory machine, only the latter works

Characteristics

- Processes communicate only through messages – no shared memory or no global clocks
- Processes must expect unpredictable message delays
- Processes coordinate access to shared resources (printer, file, etc.) that should only be used in a mutually exclusive manner.

Race Conditions

- Consider Online systems – For example, Airline reservation systems maintain records of available seats
- Suppose two people buy the same seat, because each checks and finds the seat available, then each buys the seat
- Overlapped accesses generate different results than serial accesses – race condition

Distributed Mutual Exclusion

- Needs
  - Only one process should be in critical section at any point of time
  - What about resources?

Distributed Mutual Exclusion

- No Deadlocks – no set of sites should be permanently blocked, waiting for messages from other sites in that set
- No starvation – no site should have to wait indefinitely to enter its critical section, while other sites are executing the CS more than once
- Fairness - requests honored in the order they are made. This means processes have to be able to agree on the order of events. (Fairness prevents starvation.)
- Fault Tolerance – the algorithm is able to survive a failure at one or more sites

Distributed MutEx – An overview

Token-based solution: Processes share a special message known as a token

- ➔ Token holder has right to access shared resource
- ➔ Wait for/ask for (depending on algorithm) token; enter Critical Section (CS) when it is obtained, pass to another process on exit or hold until requested (depending on algorithm)
- ➔ If a process receives the token and doesn't need it, just pass it on

Distributed MutEx – A Few Issues

- Who can access the resource?
- When does a process to be privileged to access the resource?
- How long does a process access the resource? Any finite duration?
- How long can a process wait to be privileged?
- Computation complexity of the solution

Types of Distributed MutEx

- Token-based distributed mutual exclusion algorithms
  - Suzuki – Kasami's Algorithm
- Non-token based distributed mutual exclusion algorithms
  - Lamport's Algorithm
  - Ricart-Agartala's Algorithm

Token Based Methods

Advantages:

- Starvation can be avoided by efficient organization of the processes
- Deadlock is also avoidable

Disadvantage: Token Loss

- Must initiate a cooperative procedure to recreate the token
- Must ensure that only one token is created!

Permission-Based Methods

- **Non-Token Based solutions:** a process that wishes to access a shared resource must first get permission from one or more other processes.

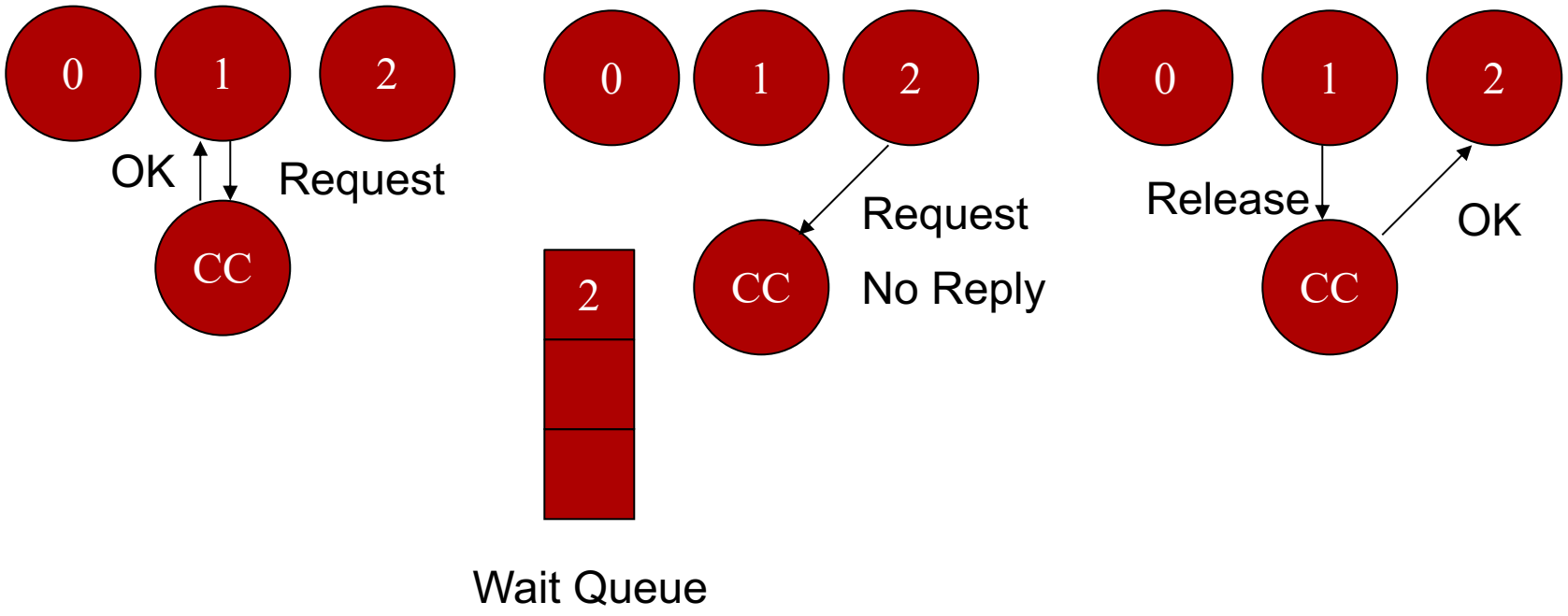
- Avoids the problems of token-based solutions, but is more complicated to implement

Basic Algorithms

- Centralized
- Decentralized
- Distributed
  - Distributed with "voting" – for increased fault tolerance
- Token Ring

Centralized Mutual Exclusion

➔ Central coordinator manages the FIFO queue of requests to guarantee "no starvation"



Centralized Control of MutEx

- A central coordinator (master or leader)
- Is elected (which algorithm?)
- Grants permission to enter CS & keeps a queue of requests to enter the CS.
- Ensures only one process at a time can access the CS
- Has a special token message, which it can give to any process to access CS

Centralized Control - Operations

- To enter a CS, send a request to the coordinator & wait for token.
- On exiting the CS, send a message to the coordinator to release the token.
- Upon receipt of a request, if no other process has the token, the coordinator replies with the token; otherwise, the coordinator queues the request
- Upon receipt of a release message, the coordinator removes the oldest entry in the queue (if any) and replies with a token

Centralized Control - Features

- Safety, Liveness are guaranteed
- Ordering also guaranteed (what kind?)
- Requires 2 messages for entry + 1 messages for exit operation.
- Client delay: one round trip time (request + grant)
- Synchronization delay: 2 message latencies (release + grant)
- The coordinator becomes performance bottleneck and single point of failure

Decentralized MutEx

- More fault-tolerant than centralized approach
- Uses the Distributed Hash Table (DHT) approach to locate objects/replicas
  - Object names are hashed to find the node where they are stored (succ function)
- n replicas of each object are placed on n successive nodes
  - Hash object name to get addresses
- Now every replica has a coordinator that controls access

The Decentralized Algorithm

- Coordinators respond to requests at once:

  Yes OR No

- Majority: To use the resource, a process must receive permission from m > n/2 coordinators
  - If the requester gets fewer than m votes, it will wait for a random time and then ask again
- If a request is denied, or when the CS is completed, notify the coordinators who have sent OK messages, so they can respond again to another request. (Why is this important?)

The Decentralized Algo - Analysis

- More robust than the central coordinator approach. If one coordinator goes down others are available.
  - If a coordinator fails and resets then it will not remember having granted access to one requestor, and may then give access to another.
  - It is highly unlikely that this will lead to a violation of mutual exclusion

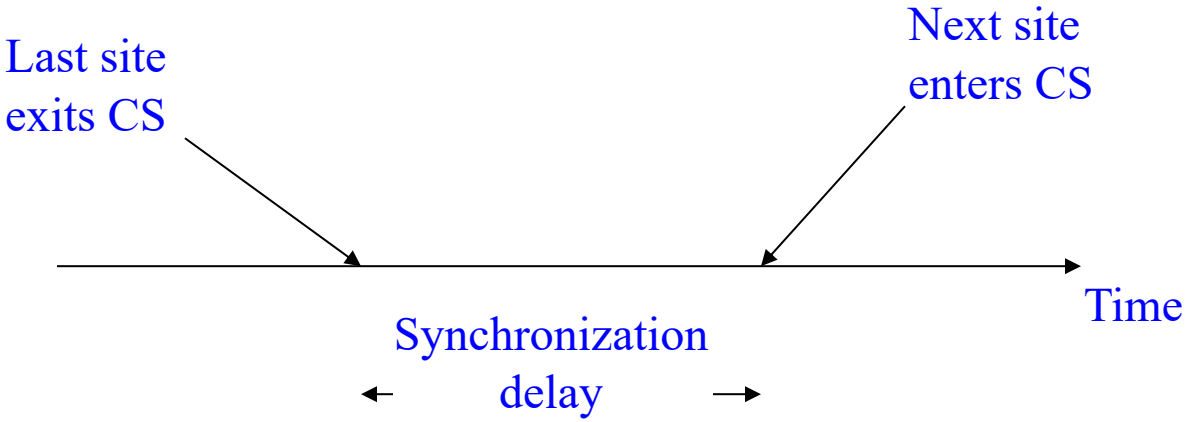
The Decentralized Algorithm - Issues

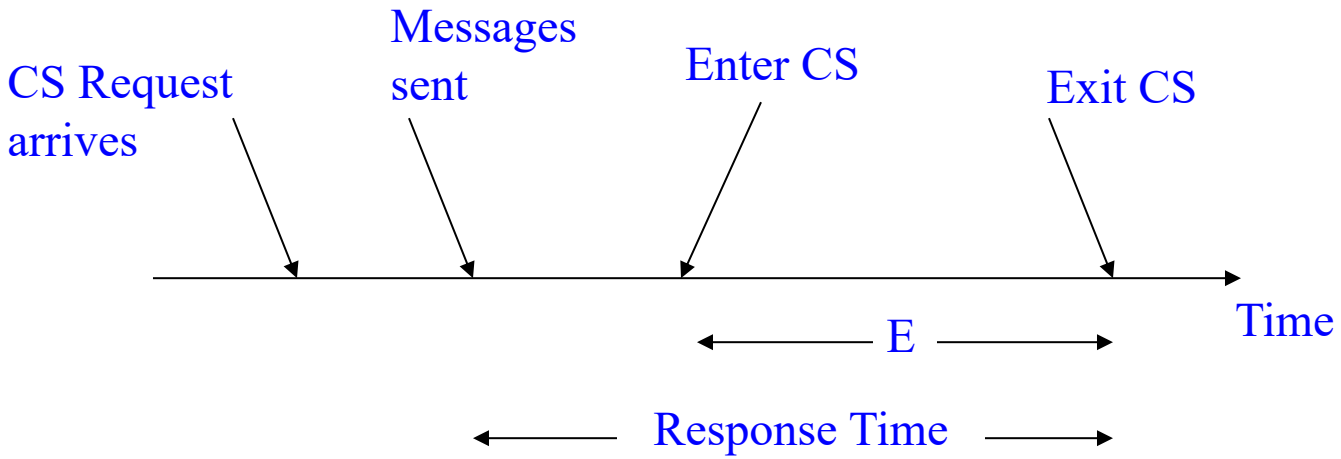
- If a resource is in high demand, multiple requests will be generated by different processes.
- High level of contention
- Processes may wait a long time to get permission - Possibility of starvation exists
- Resource usage drops.

Performance Analysis

- Guarantees mutual exclusion
- No starvation: Only if requests served in order
- No deadlock
- Fault tolerant?
  - Single point of failure
  - Blocking requests mean client processes have difficulty distinguishing crashed coordinator from long wait
  - Bottlenecks
- The solution is simple and ease

Performance Metrics





Performance - Analysis

- Number of messages per CS invocation: should be minimized
- Synchronization delay, i.e., time between the leaving of CS by a site and the entry of CS by the next one: should be minimized
- Response time: time interval between request messages transmissions and the exit of CS
- System throughput, i.e., rate at which system executes the requests for CS: should be maximized
- If d is the synchronization delay, e the average CS execution time:

$$\text{System Throughput} = 1 / (d + e)$$

Performance – Analysis (contd)

- Low and High Load:
  - Low load: No more than one request at a given point in time
  - High load: Always a pending mutual exclusion request at a site
- Best and Worst Case:
  - Best Case (low loads): Round-trip message delay + Execution time - 2T + E
  - Worst case (high loads)
- Message traffic: low at low loads, high at high loads
- Average performance: when load conditions fluctuate widely

Token Ring Approach

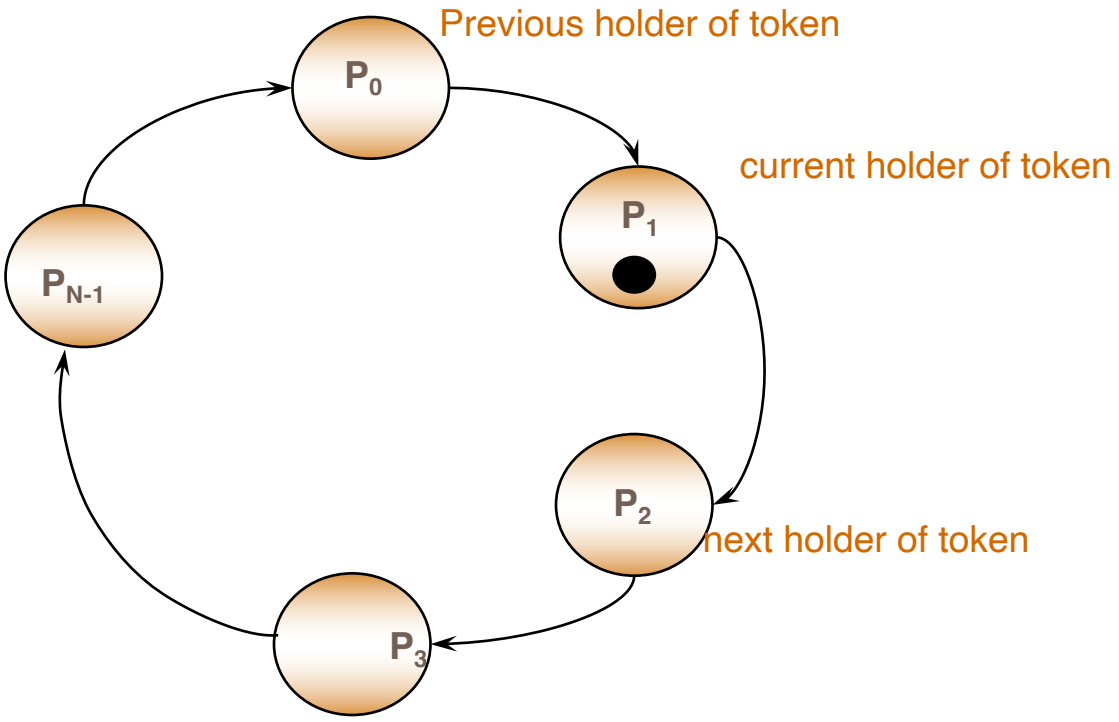
- Processes are organized in a logical ring: P_i has a communication channel to $P_{(i+1) mod\ N}$

Operations:

- Only the process holding the token T can enter the CS
- To enter the critical section, wait passively for T When in CS, hold on to T and don't release it
- To exit the CS, send T onto your neighbor
- If a process does not want to enter the CS when it receives T, it simply forwards T to the next neighbor

Token Rings – Illustration

➔ **Request movements in an unidirectional ring network**



Token Rings - Features

- Safety & Liveness are guaranteed
- Ordering is not guaranteed
- Bandwidth: 1 message per exit
- Client delay: 0 to N message transmissions
- Synchronization delay between one process's exit from the CS and the next process's entry is between 1 and N-1 message transmissions

Summary

- Racap: Message Ordering & Group Communications
  - Good / Bad Ordering
  - Causal Ordering

- Distributed Mutual Exclusion Algorithms
  - Mutual Exclusion Problem
    - Basics of MutEx algorithms
    - Types of MutEx algorithms
    - Centralized Approach
    - Token-based / Permission-based algorithms
    - Token RingsPerformance Metrics

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

Contact Information

Dr. Rajendra Prasath
IIIT Sri City, Chittoor