Distributed Systems

Monsoon 2024

Lecture 4

International Institute of Information Technology

Hyderabad, India

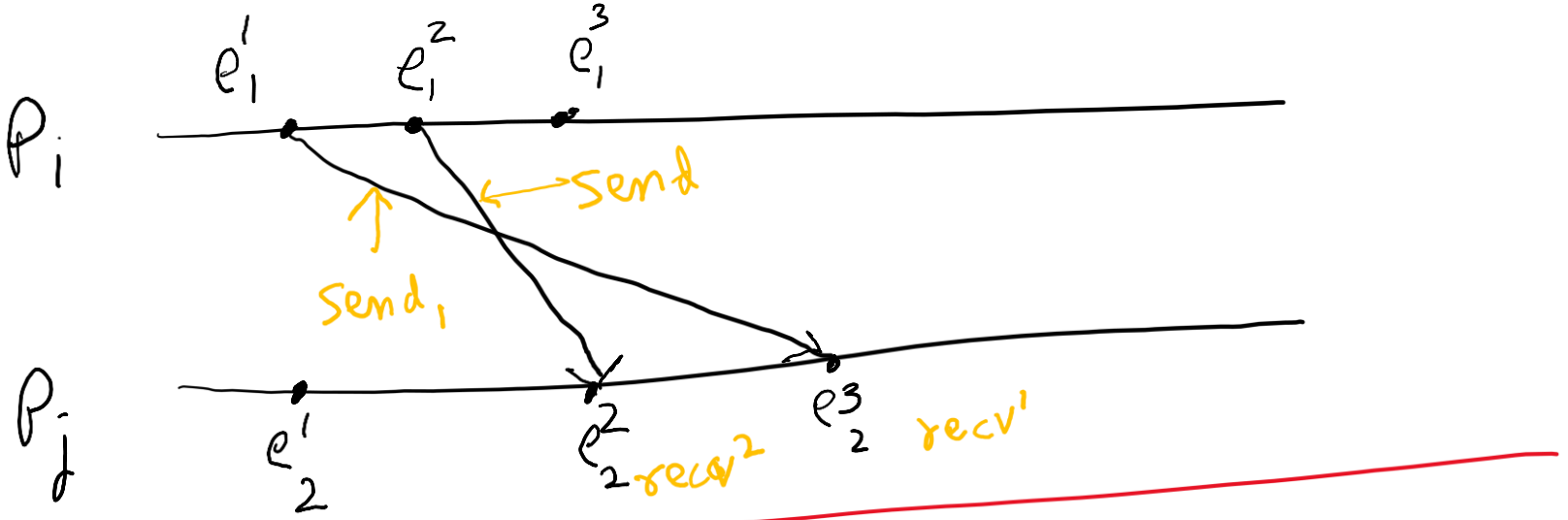
A Distributed Program in Execution

- Let us now model message delivery.
- Several ways are possible
  - Arbitrary : Message can be reordered in transit in the channel
  - FIFO : Messages are NOT reordered in transit in the channel
  - Causal : Slightly stronger than FIFO. Messages destined for the same destination are not reordered. In symbols,

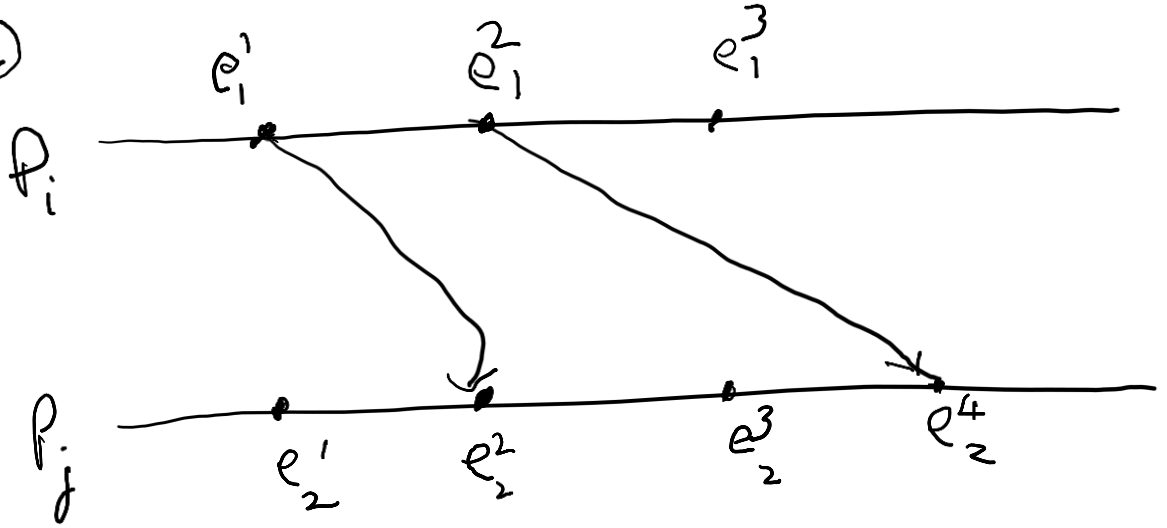
    for any two messages m_{ij} and m_{kj},
    if send(m_{ij}) $\rightarrow$ send(m_{kj}) then recv(m_{ij}) $\rightarrow$ recv(m_{kj}).

  - Causal $\subset$ FIFO $\subset$ Arbitrary
  - Draw pictures to understand the differences.

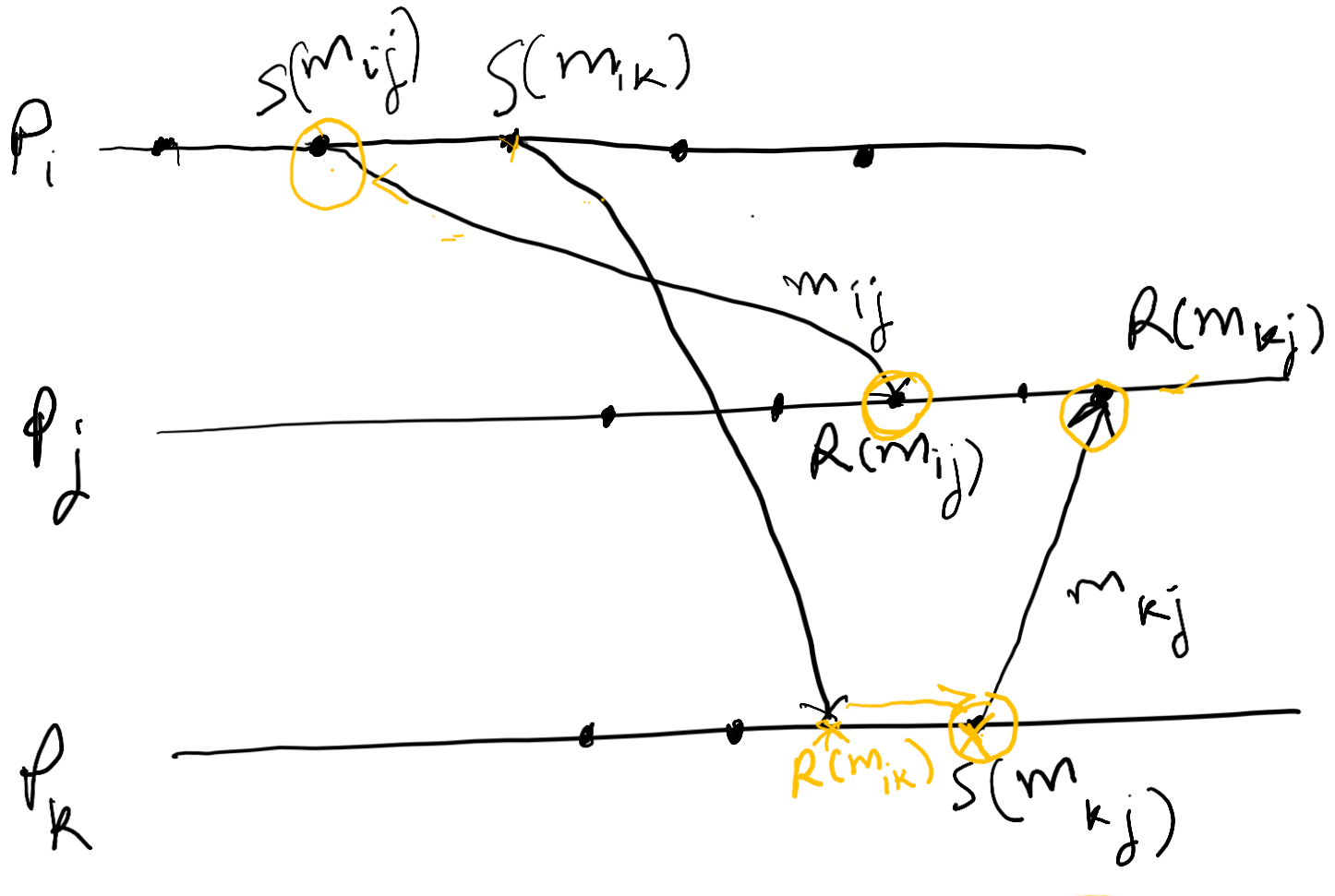
- Arbitrary



- FIFO



Causal



S send
R Recv.

$S(m_{ij}) \leq S(m_{kj})$

$\Downarrow$

$R(m_{ij}) \leq R(m_{kj})$

A Distributed Program in Execution

- Several ways are possible
  - Non-FIFO : Message can be reordered in transit
  - FIFO : Messages are NOT reordered in transit.
  - Causal : Slightly stronger than FIFO. Messages destined for the same destination are not reordered. In symbols,

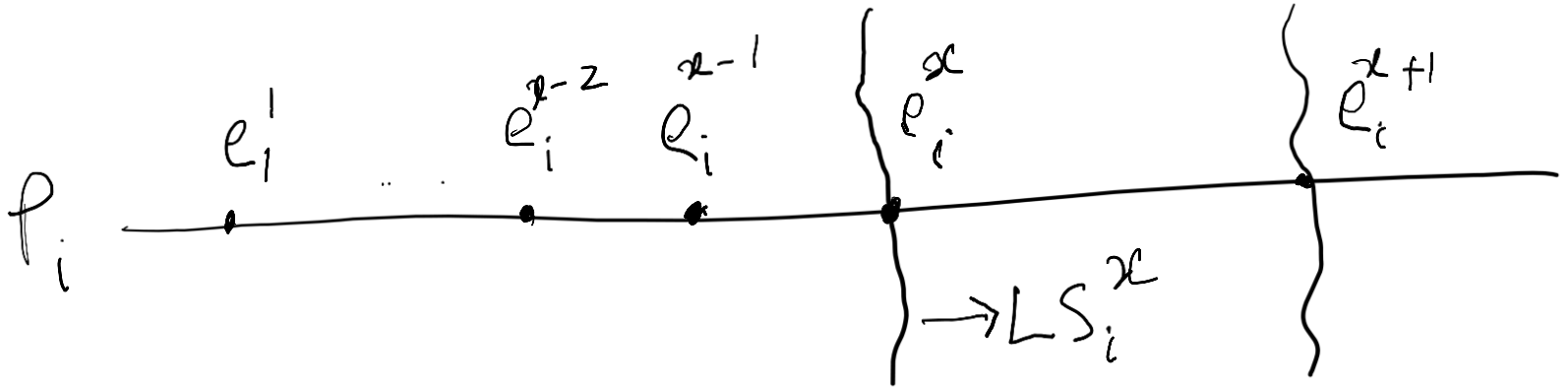
    for any two messages m_{ij} and m_{kj},
    if send(m_{ij}) $\rightarrow$ send(m_{kj}) then recv(m_{ij}) $\rightarrow$ recv(m_{kj}).

  - Causal $\subset$ FIFO $\subset$ Non-FIFO.
  - How can causal order among messages be guaranteed? Read about it from An Efficient Causal Order Algorithm for Message Delivery in Distributed System, Jangt, Park, Cho, and Yoon

The Global State of a Distributed System

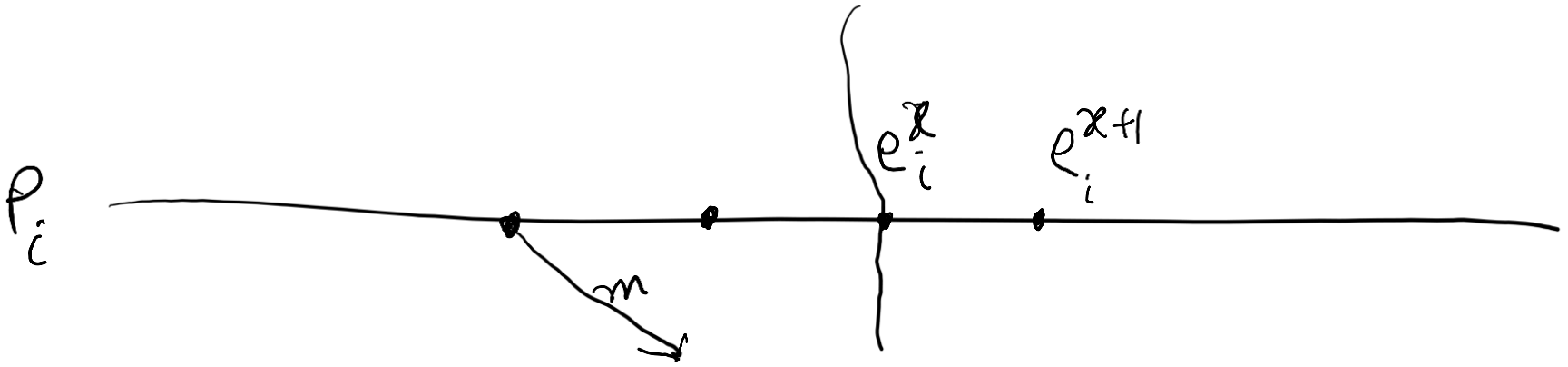
- One can think of the collection of local states as the global state.
- Recall that the local state of a process, P_i, is the contents of the processor registers, stack, local memory, etc.
- The state of a channel is the set of messages in transit in the channel.
- Events lead to changes in the state of local process(es).

The Global State of a Distributed System



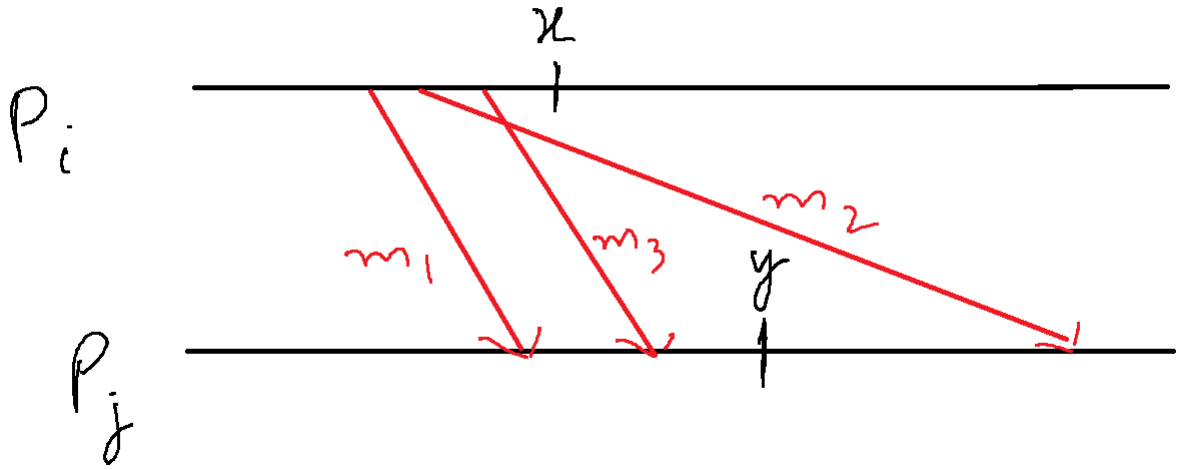
- To formalize, let LS^x_i denote the local state of process P_i after the occurrence of the event e^x_i and before the occurrence of the event e^{x+1}_i.
  - LS^0_i is the initial state of P_i
- By definition of LS^x_i, the effect of all the events before the event e^x_i are accounted for.

The Global State of a Distributed System



- For a message m and the event send(m), let send(m) $\leq$ LS^x_i denote that the message m is sent by process P_i on or before the event e^x_i.
  - There exists y s.t. $1 \leq y \leq x$, e^y_i= send(m).
- Likewise, can define recv(m) $\leq$ LS^x_i as
  - There exists y s.t. $1 \leq y \leq x$, e^y_i= recv(m).
- Can negate the above statements to indicate that send(m) not$\leq$ LS^x_i .

Global State of a Distributed System



- We use the above to capture the state of a channel C_{ij}, denoted $SC^{x,y}_{i,j}$ as follows.

  $SC^{x,y}_{i,j} = \{m_{ij} \mid send(m_{ij}) \leq LS^{x}_{i}$ and $recv(m_{ij})$ not$\leq LS^{y}_{j}\}$

- In words, the state of $SC^{x,y}_{i,j}$ denotes all messages that have been sent by P_i up to event e^{x}_{i} and not received by P_j up to event e^{y}_{j}.

Global State of a Distributed System

- Finally, the global state of a distributed system, denoted GS, is defined as

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

- A global state is said to be consistent iff it satisfies the condition that a message m that is recorded as received is also recorded as sent in the state.

- Question: Write in symbols the above condition for a consistent global state.

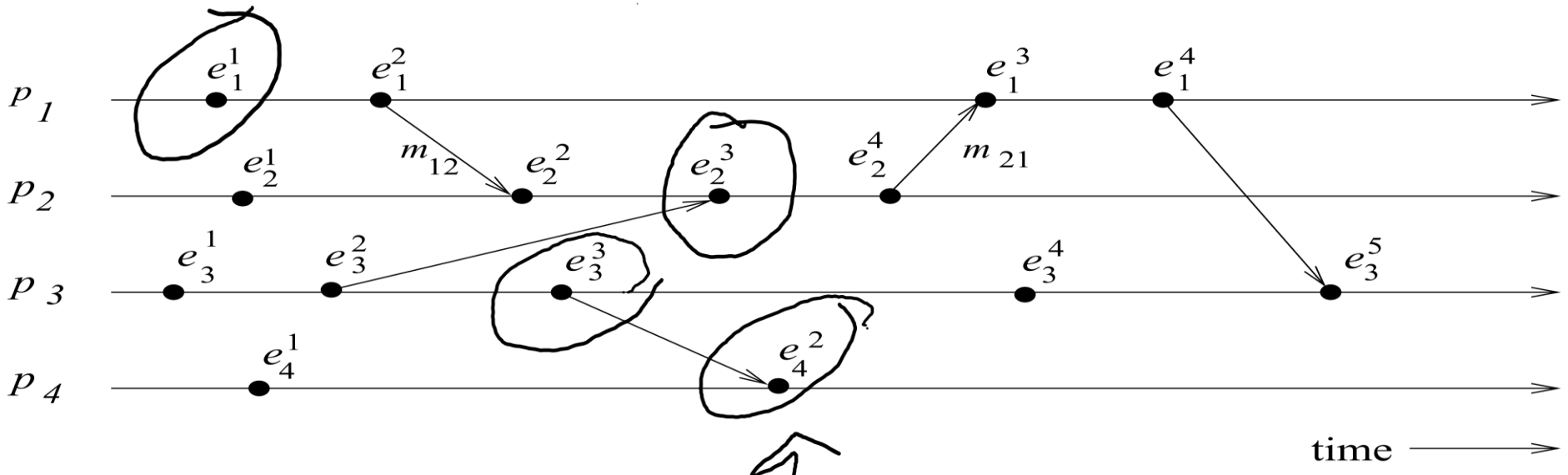
Global State of a Distributed System

- Finally, the global state of a distributed system, denoted GS, is defined as

$$GS = \{ \bigcup_{i} LS^{x_i}_{i}, \bigcup_{j,k} SC^{y_j z_k}_{jk} \}$$

- A global state is said to be consistent iff it satisfies the condition that a message m that is recorded as received is also recorded as sent in the state.

Example



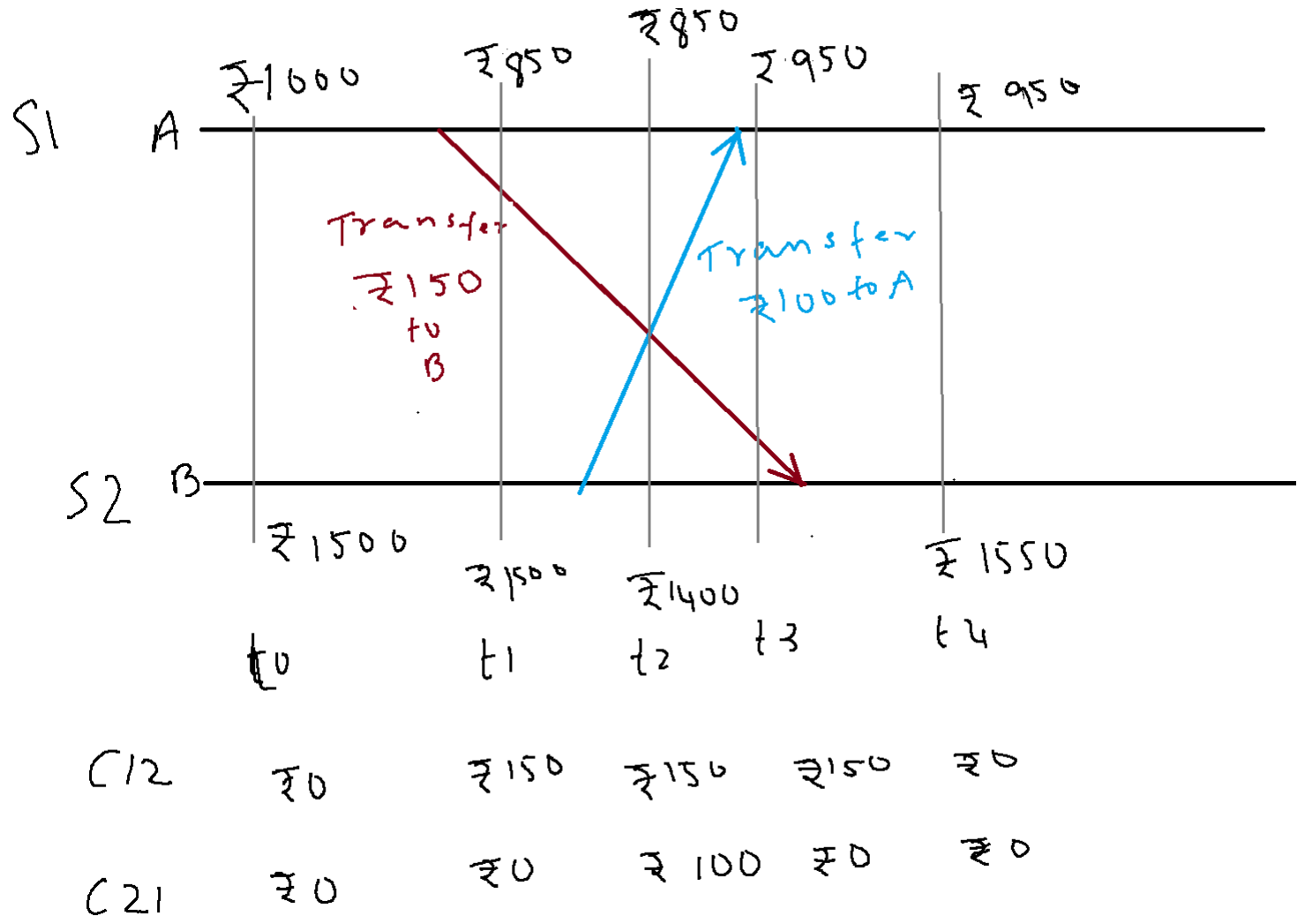
- Which global states are consistent?

1) $\{LS^1_1, LS^3_2, LS^3_3, LS^2_4\}$
2) $\{LS^2_1, LS^4_2, LS^4_3, LS^2_4\}$
3) $\{LS^2_1, LS^1_2, SC^{12}_{12}, LS^3_3, LS^2_4\}$

Global States and Snapshots

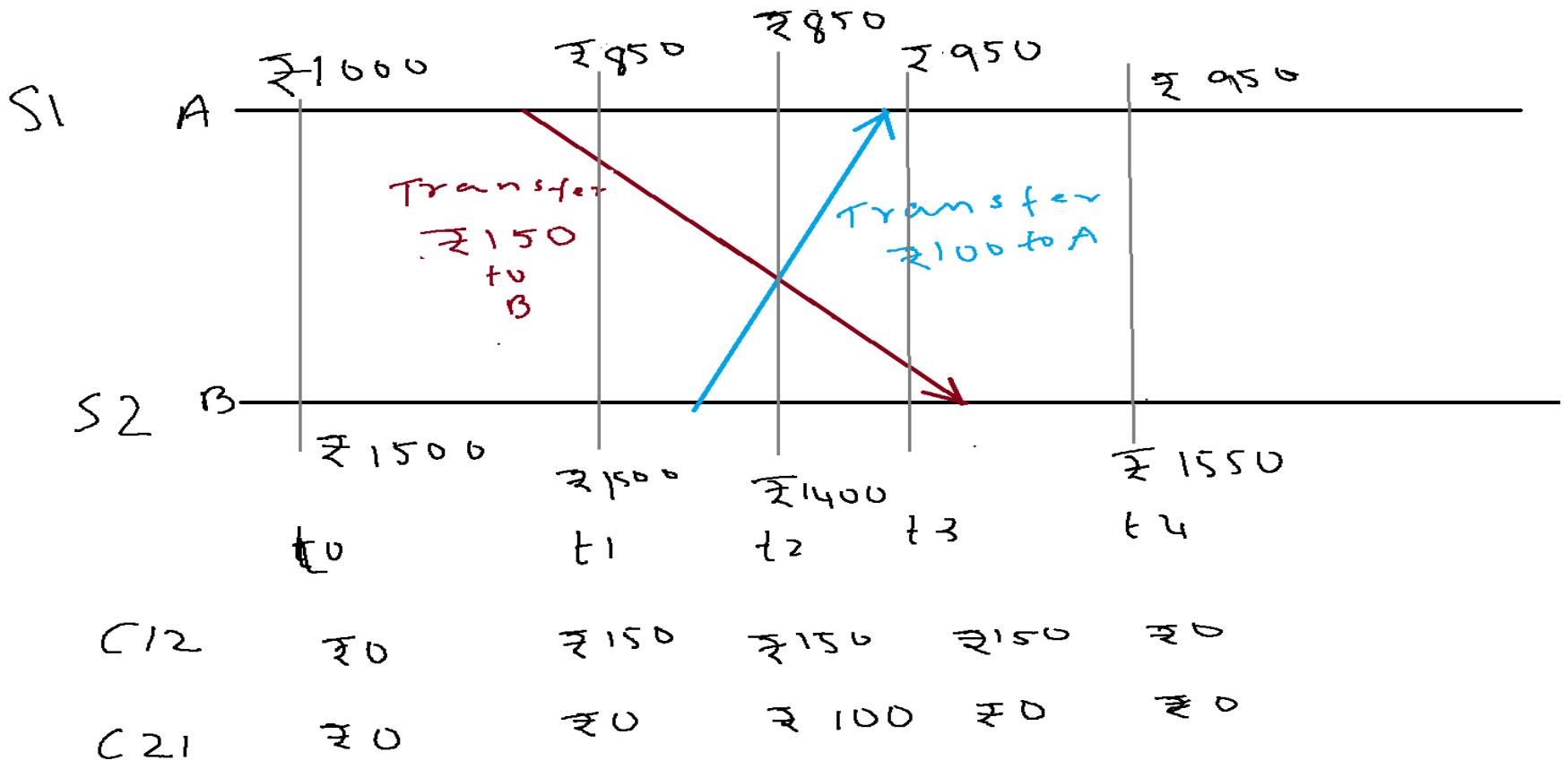
- Recording the global state of a distributed system on-the-fly is an important paradigm.
- The lack of globally shared memory, global clock and unpredictable message delays in a distributed system make this problem non-trivial.
- Building on the definition consistent global states we discuss issues to be addressed to obtain consistent distributed snapshots.
- Then several algorithms to determine on-the-fly such snapshots are presented for different message delivery models.

Global States and Snapshots



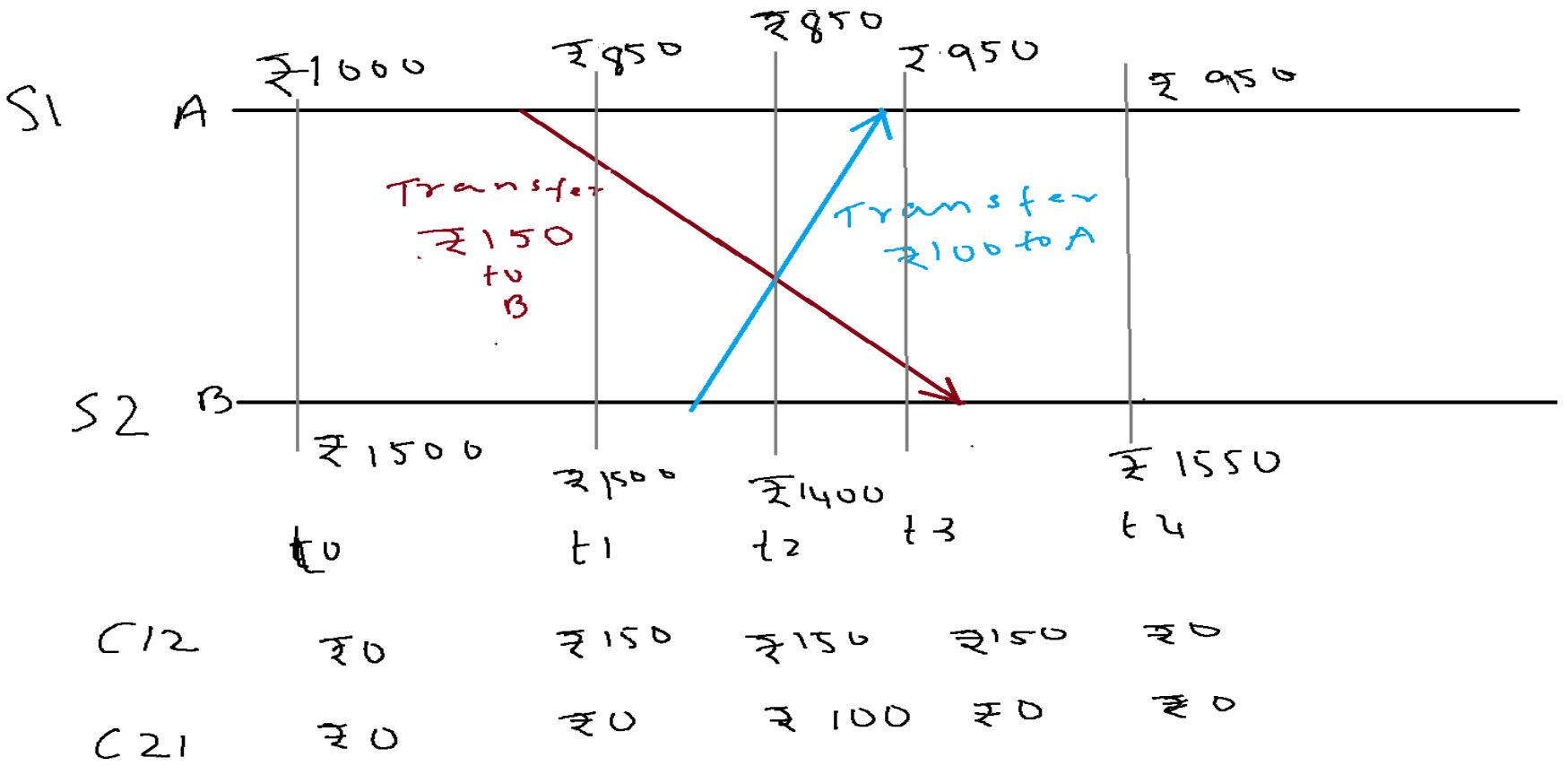
- Consider the distributed execution shown above.

Global States and Snapshots



- Consider the distributed execution shown above.
- If the state of the account A is recorded at time t_0 and the state of account B and that of the channels C_{12} and C_{21} are recorded at t_2, what is the total money in the system?

Global States and Snapshots

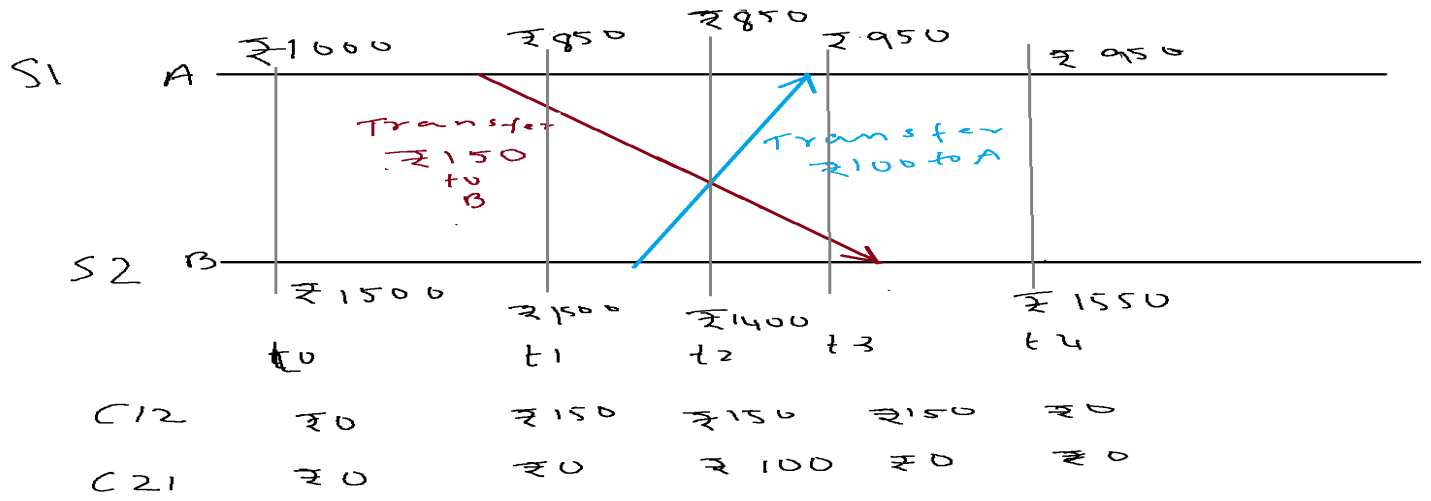


- Consider the distributed execution shown above.

An extra 150 Rs.

- If the state of the account A is recorded at time t_0 and the state of account B and that of the channels C_{12} and C_{21} are recorded at t_2, what is the total money in the system?

Global States and Snapshots



- Consider the distributed execution shown above.
- If the state of the account A is recorded at time t_0 and the state of account B and that of the channels C_{12} and C_{21} are recorded at t_2, what is the total money in the system?

  An extra 150 Rs.
- If this state is used for checkpointing, then there is trouble!

Messages in Transit

- For a channel C_{ij} , the following set of messages can be defined based on the local states of the processes p_i and p_j

  Transit: $transit(LS_i, LS_j) = \{m_{ij} \mid send(m_{ij}) \in LS_i \wedge rec(m_{ij}) \notin LS_j \}$

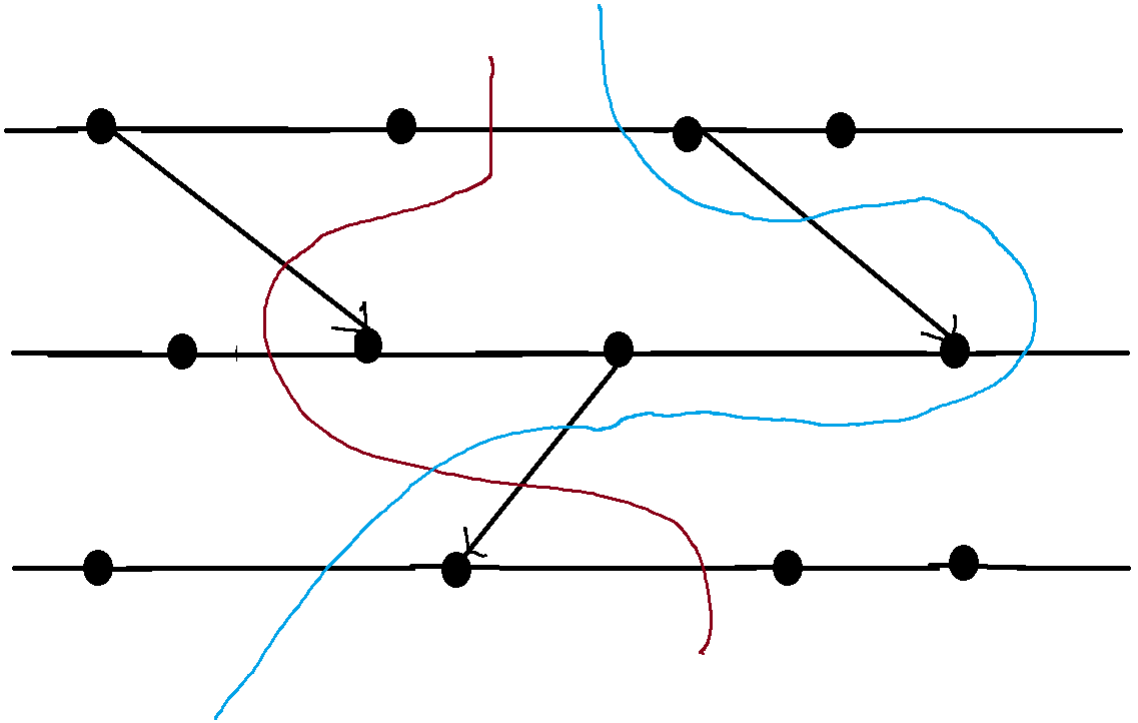
Conditions for Consistent Global States

- The global state of a distributed system is a collection of the local states of the processes and the channels.
- Notation wise, a global state GS is defined as,
  - $GS = \{ \cup_i LS_i , \cup_{i,j} SC_{ij} \}$
- A global state GS is a consistent global state iff it satisfies the following two conditions :
- C1: Law of Conservation of Messages
- C2: For every effect, its cause must be present.

Conditions for Consistent Global States

- A global state GS is a consistent global state iff it satisfies the following two conditions :
- C1 states the law of conservation of messages.
  - Every message that is recorded as sent in the local state of some process is either captured in the state of the channel or is captured in the local state of the receiver.
  - $send(m_{ij}) \in LS_i \Rightarrow m_{ij} \in SC_{ij} \oplus rec(m_{ij}) \in LS_j$. ($\oplus$ is the Exclusive-OR operator)
- C2 states that for every effect, its cause must be present.
  - If a message is not recorded as sent in the local state of a process P_i, then the message cannot be included in the state of the channel C_{ij} or be captured as received by P_j.
  - $send(m_{ij}) \notin LS_i \Rightarrow m_{ij} \notin SC_{ij} \wedge rec(m_{ij}) \notin LS_j$.

Interpretation in terms of cuts



- A cut in a space-time diagram is a line joining an arbitrary point on each process line that slices the space-time diagram into a PAST and a FUTURE.
- A consistent global state corresponds to a cut in which every message received in the PAST of the cut was also sent in the PAST of that cut.
- Such a cut is known as a consistent cut.

Interpretation in terms of cuts – Example

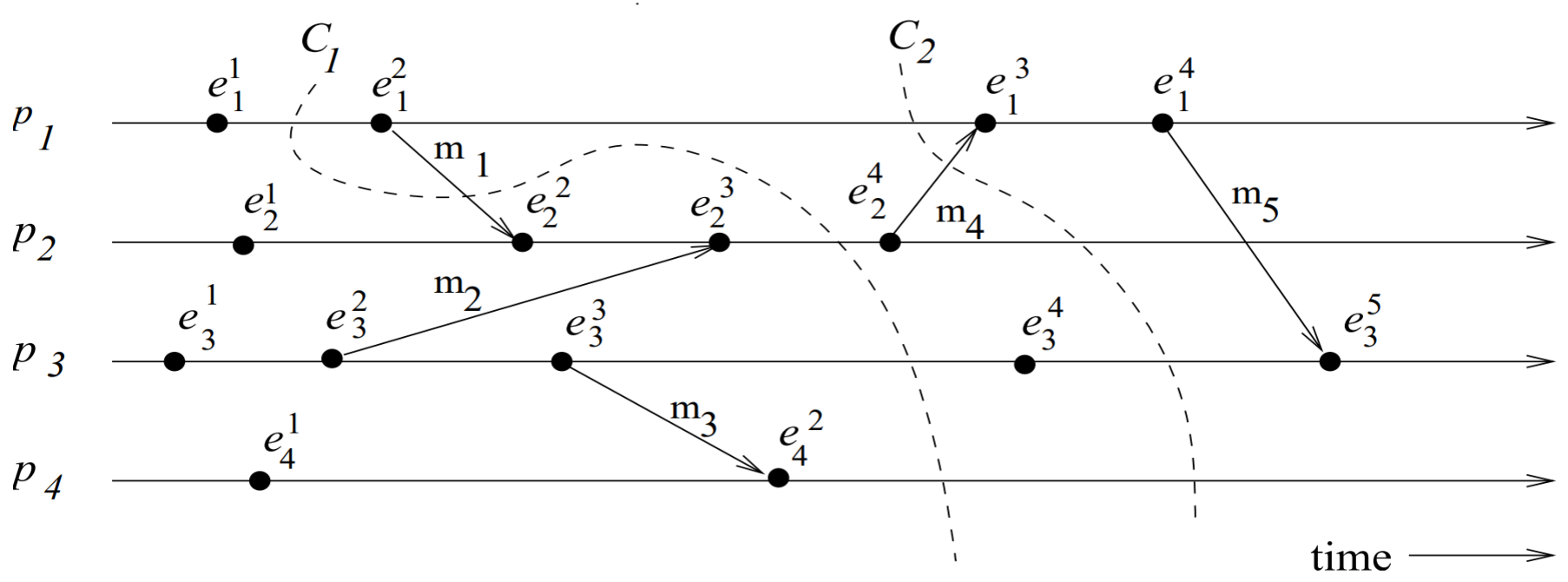


Figure 4.1: An Interpretation in Terms of a Cut.

- Is the cut C_1 consistent?
- Is the cut C_2 consistent?

Interpretation in terms of cuts – Example

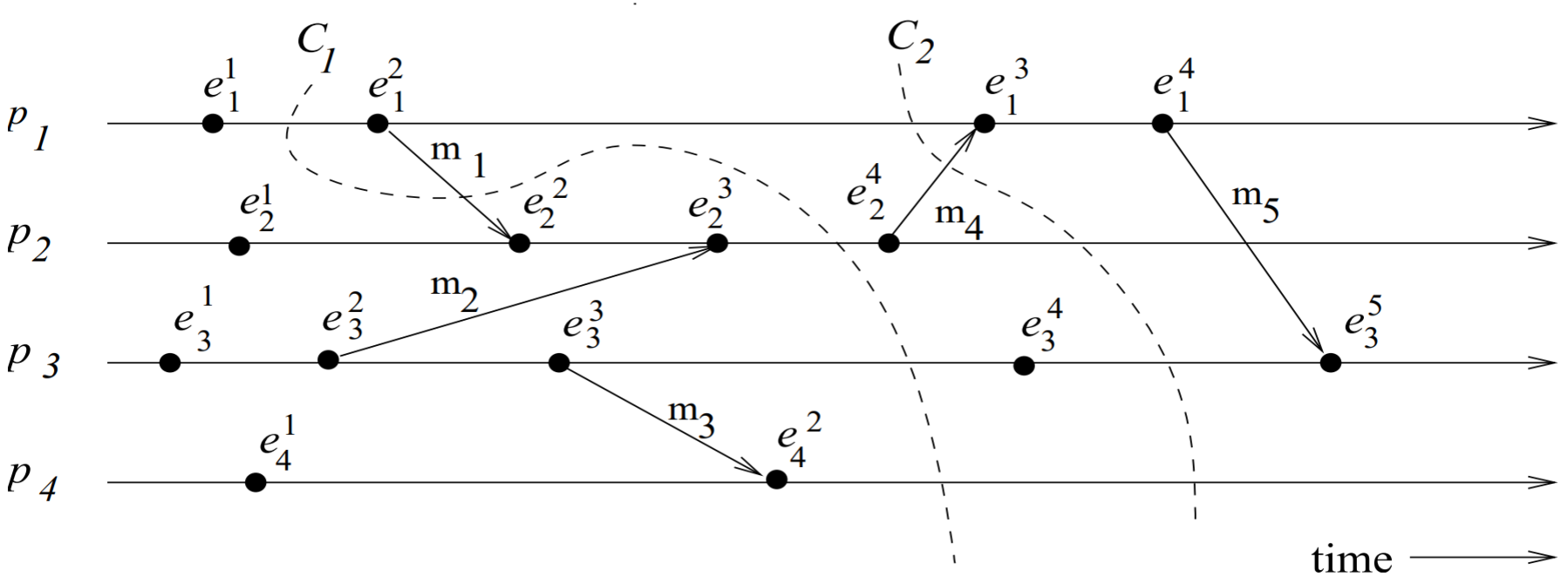


Figure 4.1: An Interpretation in Terms of a Cut.

- Cut C_1 is inconsistent because message m_1 is flowing from the FUTURE to the PAST.
- Cut C_2 is consistent and message m_4 must be captured in the state of channel C_{21}.

Difficulty of Taking a Snapshot

- Issue 1: How to distinguish between the messages to be recorded in the snapshot from those not to be recorded.
  - Any message that is sent by a process before recording its snapshot, must be recorded in the global snapshot (from C1).
  - Any message that is sent by a process after recording its snapshot, must not be recorded in the global snapshot (from C2).
- Issue 2: How to determine the instant when a process takes its snapshot.
  - A process p_j must record its snapshot before processing a message m_{ij} that was sent by process p_i after recording its (p_i) snapshot.

Difficulty of Taking a Snapshot

- Issue 1: How to distinguish between the messages to be recorded in the snapshot from those not to be recorded.
  - Any message that is sent by a process before recording its snapshot, must be recorded in the global snapshot (from C1).
  - Any message that is sent by a process after recording its snapshot, must not be recorded in the global snapshot (from C2).
- Issue 2: How to determine the instant when a process takes its snapshot.
  - A process p_j must record its snapshot before processing a message m_{ij} that was sent by process p_i after recording its snapshot.

Algorithms for Capturing a Snapshot

- Armed with our knowledge, we will now study algorithms for capturing a snapshot of a distributed system.
- We will see how the guarantees on message delivery helps simplify algorithms.
- We now consider systems where it is not necessary that every pair of processors share a direct channel between them.
- We will start with algorithms under the assumption that message delivery follows causal rules, then study algorithms for FIFO channels, and then for arbitrary channels.

Snapshots in a Causal Channels

- The causal message delivery property provides a built-in message synchronization to control and computation messages.
- Two global snapshot recording algorithms, namely, Acharya-Badrinath and Alagar-Venkatesan, exist that assume that the underlying system supports causal message delivery.

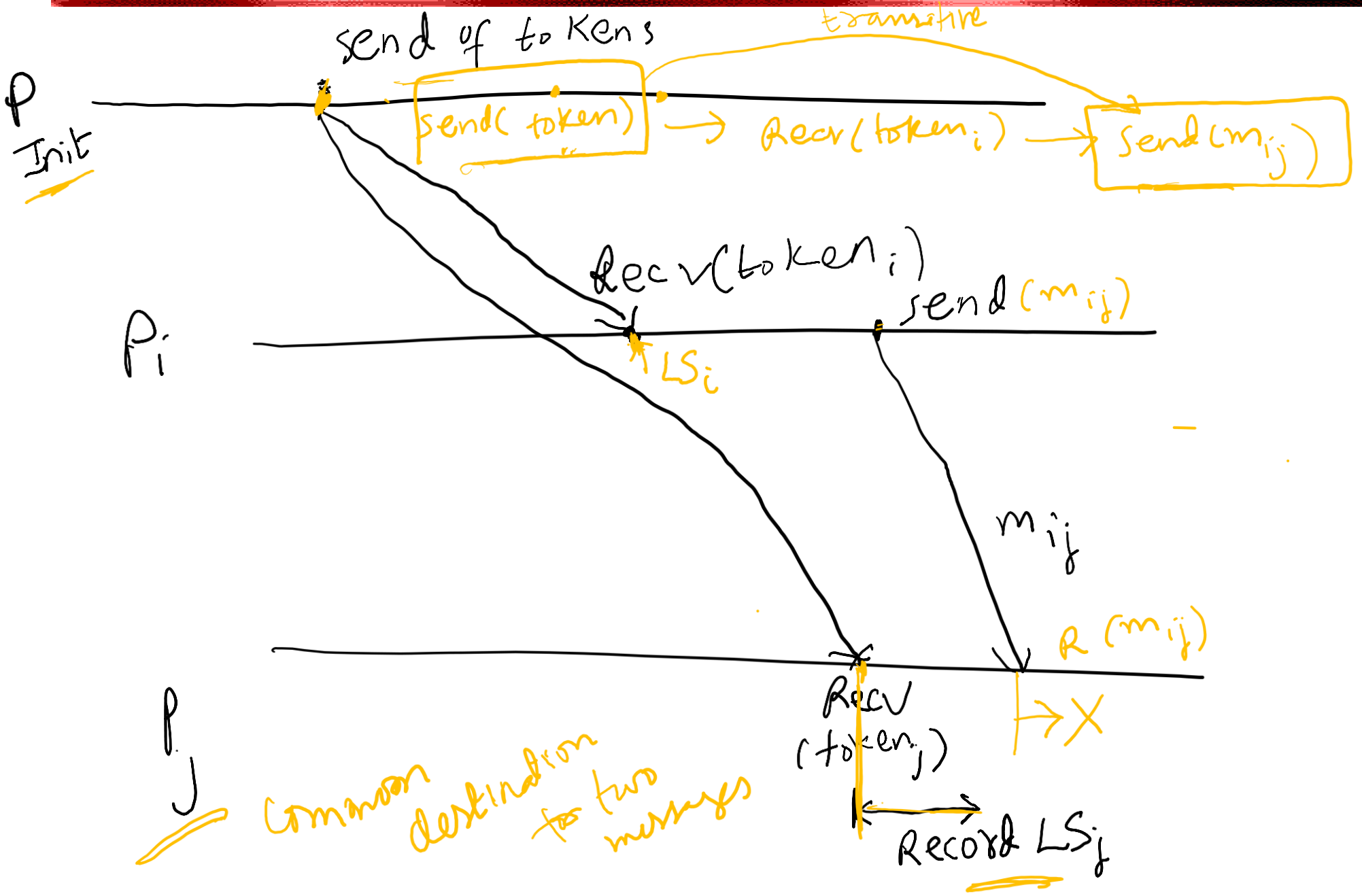
Snapshots in a Causal Channels

- In both these algorithms the recording of process state is identical and proceeds as follows :
- An initiator process broadcasts a token, denoted as `token`, to every process including itself.
- Let the copy of the token received by process p_i be denoted $token_i$.
- A process p_i records its local snapshot LS_i on receiving $token_i$ and sends the recorded snapshot to the initiator.
- The algorithm terminates when the initiator receives the snapshot recorded by each process.

Snapshots in a Causal Channels

- The correctness of the approach depends on the following observation.
- For any two processes p_i and p_j, the following property is satisfied:
  - $send(m_{ij}) \notin LS_i \Rightarrow rec(m_{ij}) \notin LS_j$.
- This is due to the causal ordering property of the underlying system as explained next.

Snapshots in a Causal Channels



Snapshots in a Causal Channels

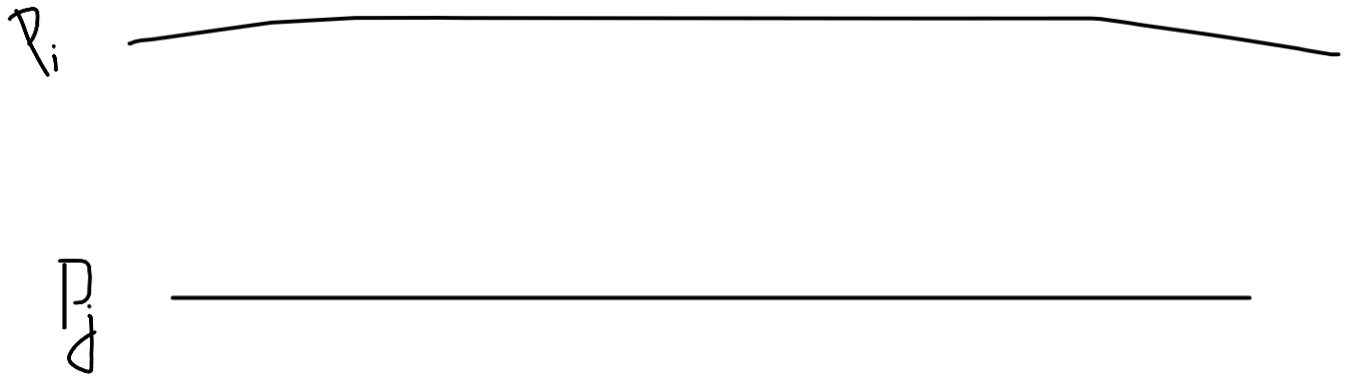
- Let a message m_{ij} be such that $rec(token_i) \rightarrow send(m_{ij})$
- Then $send(token_j) \rightarrow send(m_{ij})$ and the underlying causal ordering property ensures that $rec(token_j)$, at which instant process p_j records LS_j, happens before $rec(m_{ij})$.

- Thus, m_{ij}, whose send is not recorded in LS_i, is not recorded as received in LS_j.

Acharya – Badrinath Algorithm

- Each process p_i maintains arrays $SENT_i[1, ...N]$ and $RECD_i[1, ..., N]$.
- $SENT_i[j]$ is the number of messages sent by process p_i to process p_j.
- $RECD_i[j]$ is the number of messages received by process p_i from process p_j.
- Channel states are recorded as follows:
  - When a process p_i records its local snapshot LS_i on the receipt of token, it includes arrays $RECD_i$ and $SENT_i$ in its local state before sending the snapshot to the initiator.

Acharya – Badrinath Algorithm



- When the algorithm terminates, the initiator determines the state of channels as follows:
- The state of each channel from the initiator to each process is empty.
- The state of channel from process p_i to process p_j is the set of messages whose sequence numbers are given by $\{RECD_j[i] + 1, \ldots, SENT_i[j]\}$.

Acharya – Badrinath Algorithm

Complexity:

- This algorithm requires 2n messages and 2 time units for recording and assembling the snapshot, where one time unit is required for the delivery of a message.
- If the contents of messages in channels state are required, the algorithm requires 2n messages and 2 time units additionally.