Distributed Systems

CS425/ECE428

Feb 1 2023

Instructor: Radhika Mittal

Logistics Related

- HW1 released! Due on Feb 15th
  - You should be able to solve the first three questions right-away.
  - You should be able to solve the fourth question by the end of this class (hopefully).
  - You should be able to solve the fifth question by the end of next class.
- Newly registered students:
  - Please make sure you have access to Campuswire and Gradescope
  - If you are in 4 credits, make sure you have been allocated a VM cluster for the MPs.
  - Email Manoj (netid: gmk6) to get the required access.

- *Please say your name before speaking up in class ☺*

Recap: Logical timestamps

- How to reason about ordering of events across processes without synchronized clocks?
- Happened-before Relationship
- Lamport Logical Clock
- Vector Clock

Today's agenda

- **Global State**
  - Chapter 14.5
  - Goal: reason about how to capture the state across all processes of a distributed system without requiring time synchronization.

Process, state, events

- Consider a system with **n** processes: $<p_1, p_2, p_3, \ldots, p_n>$.
- Each process p_i is associated with *state* $\mathbf{s_i}$.
  - State includes values of all local variables, affected files, etc.
- Each channel can also be associated with a state.
  - Which messages are currently *pending* on the channel.
  - Can be computed from process' state:
    - Record when a process sends and receives messages.
    - if p_i sends a message that p_j has not yet received, it is pending on the channel.
- State of a process (or a channel) gets transformed when an *event* occurs. 3 types of events:
  - local computation, sending a message, receiving a message.

Capturing a global snapshot

- Useful to capture a global snapshot of the system:
    - *Checkpointing* the system state.
    - Reasoning about unreferenced objects (for garbage collection).
    - Deadlock detection.
    - Distributed debugging.

Capturing a global snapshot

- Global state or global snapshot is state of each process (and each channel) in the system at a given *instant of time*.
- Difficult to capture a global snapshot of the system.
- Strawman:
  - Each process records its state at 2:05pm.
  - We get the global state of the system at 2:05pm.
  - *But precise clock synchronization is difficult to achieve.*
- **How do we capture global snapshots without precise time synchronization across processes?**

Some more notations and definitions

- State of a process (or a channel) gets transformed when an *event* occurs.

- 3 types of events:
  - local computation, sending a message, receiving a message.

- $\mathbf{e_i^n}$ is the n^{th} event at $\mathbf{p_i}$.

Some more notations and definitions

- For a process p_i , where events $e_i^0, e_i^1, \ldots$ occur:

  history(p_i) = $h_i = \langle e_i^0, e_i^1, \ldots \rangle$

  prefix history(p_i^k) = $h_i^k = \langle e_i^0, e_i^1, \ldots, e_i^k \rangle$

  s_i^k : p_i's state immediately after k^{th} event.
- For a set of processes $\langle p_1, p_2, p_3, \ldots, p_n \rangle$:

  global history: $H = \cup_i (h_i)$

Some more notations and definitions

- For a process p_i , where events $e_i^0, e_i^1, \ldots$ occur:

  history(p_i) = $h_i = \langle e_i^0, e_i^1, \ldots \rangle$

  prefix history(p_i^k) = $h_i^k = \langle e_i^0, e_i^1, \ldots, e_i^k \rangle$

  s_i^k : p_i's state immediately after k^{th} event.
- For a set of processes $\langle p_1, p_2, p_3, \ldots, p_n \rangle$:

  global history: $H = \cup_i (h_i)$

Some more notations and definitions

- For a process p_i , where events $e_i^0, e_i^1, \ldots$ occur:

  history(p_i) = $h_i = \langle e_i^0, e_i^1, \ldots \rangle$

  prefix history(p_i^k) = $h_i^k = \langle e_i^0, e_i^1, \ldots, e_i^k \rangle$

  s_i^k : p_i's state immediately after k^{th} event.

- For a set of processes $\langle p_1, p_2, p_3, \ldots, p_n \rangle$:

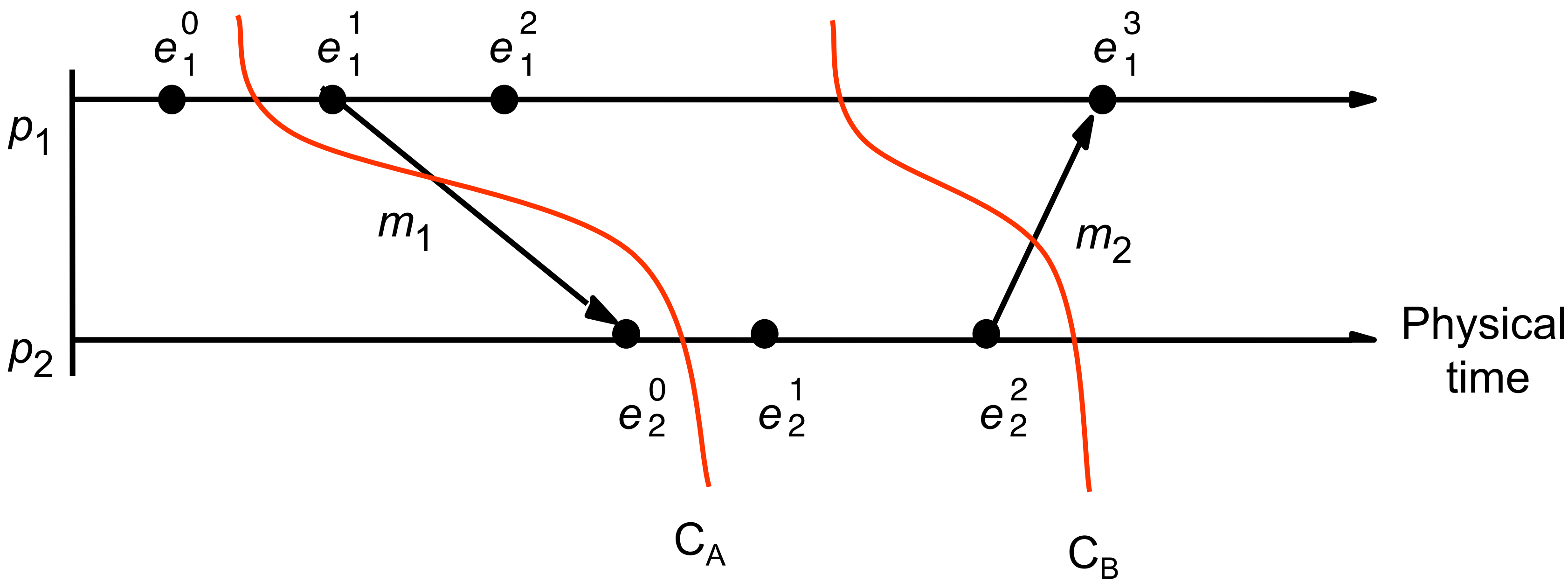
  global history: $H = \cup_i (h_i)$

  a cut $C \subseteq H = h_1^{c_1} \cup h_2^{c_2} \cup \ldots \cup h_n^{c_n}$

  the frontier of $C = \{e_i^{c_i}, i = 1,2, \ldots n\}$

  global state S that corresponds to cut $C = \cup_i (s_i^{c_i})$

Example: Cut



C_A: < e_1^0, e_2^0>

Frontier of C_A:

C_B: < e_1^0, e_1^1, e_1^2, e_2^0, e_2^1 e_2^2 >

Frontier of C_B:

Some more notations and definitions

- For a process p_i , where events $e_i^0, e_i^1, \ldots$ occur:

  history(p_i) = h_i = <$e_i^0, e_i^1, \ldots$ >

  prefix history(p_i^k) = h_i^k = <$e_i^0, e_i^1, \ldots, e_i^k$ >

  s_i^k : p_i's state immediately after k^{th} event.

- For a set of processes <$p_1, p_2, p_3, \ldots, p_n$>:

  global history: $H = \cup_i (h_i)$

  a cut $C \subseteq H = h_1^{c_1} \cup h_2^{c_2} \cup \ldots \cup h_n^{c_n}$

  the frontier of $C = \{e_i^{c_i}, i = 1,2, \ldots n\}$

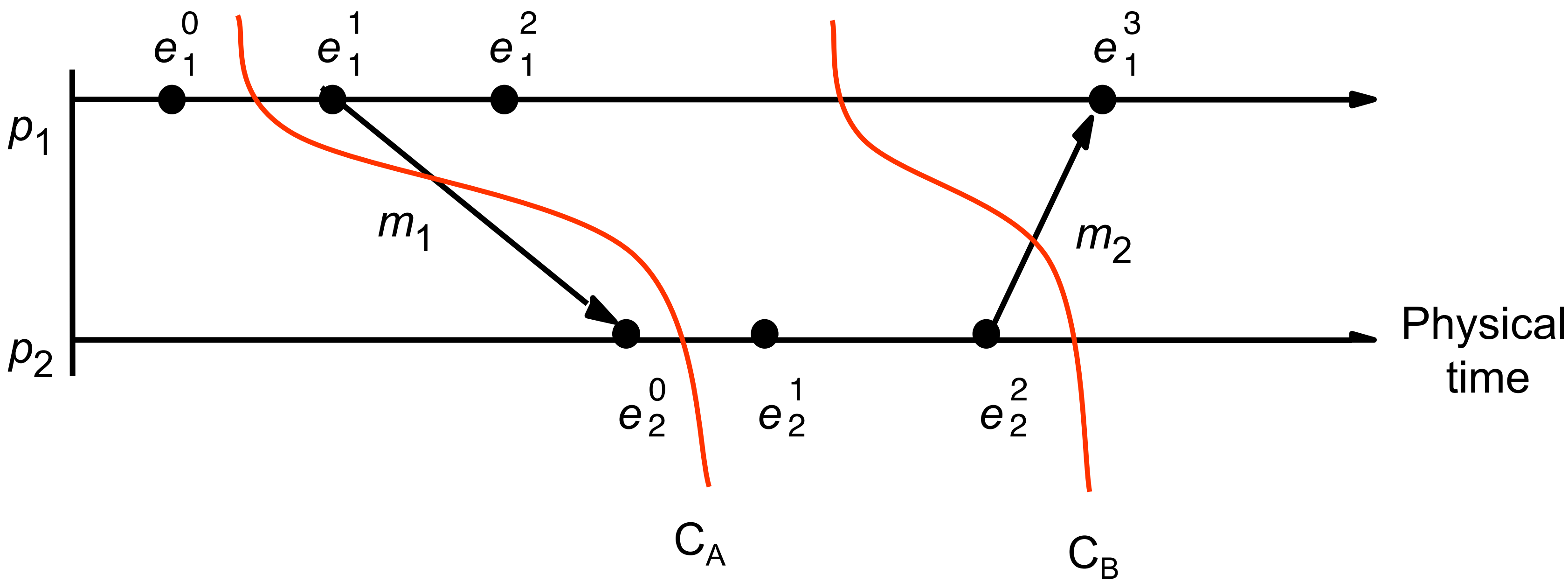
  global state S that corresponds to cut $C = \cup_i (s_i^{c_i})$

Consistent cuts and snapshots

- A cut **C** is **consistent** if and only if

$$\forall \mathbf{e} \in \mathbf{C} \ (\text{if } \mathbf{f} \rightarrow \mathbf{e} \text{ then } \mathbf{f} \in \mathbf{C})$$

Example: Cut



C_A: < e_1^0, e_2^0>
Frontier of C_A: {e_1^0, e_2^0}
Inconsistent cut.

C_B: < e_1^0, e_1^1, e_1^2, e_2^0, e_2^1 e_2^2 >
Frontier of C_B: {e_1^2, e_2^2}
Consistent cut.

Consistent cuts and snapshots

- A cut **C** is **consistent** if and only if

$$\forall \mathbf{e} \in \mathbf{C} \ (\text{if } \mathbf{f} \rightarrow \mathbf{e} \text{ then } \mathbf{f} \in \mathbf{C})$$

- A global state **S** is consistent if and only if it corresponds to a consistent cut.

How to capture global state?

- Ideally: state of each process (and each channel) in the system *at a given instant of time.*
  - Difficult to capture -- requires precisely synchronized time.

- Relax the problem: find a consistent global state.
  - For a system with n processes <p_1, p_2, p_3, …., p_n>, capture the state of the system after the c_i th event at process p_i.
    - State corresponding to the *cut* defined by frontier events {$e_i^{c_i}$, for i = 1,2, … n}.
  - We want the state to be consistent.
    - Must correspond to a consistent cut.

How to find a consistent global state that corresponds to a consistent cut ?

Chandy-Lamport Algorithm

- Goal:
  - Record a global snapshot
    - Process state (and channel state) for a set of processes.
  - The recorded global state is consistent.
- Identifies a consistent cut.
- Records corresponding state locally at each process.

Chandy-Lamport Algorithm

- *System model and assumptions:*
  - System of **n** processes: <p_1, p_2, p_3, …., p_n>.
  - There are two uni-directional communication channels between each ordered process pair : p_j to p_i and p_i to p_j.
  - Communication channels are FIFO-ordered (first in first out).
    - if p_i sends **m** before **m'** to p_j , then p_j receives **m** before **m'**.
  - All messages arrive intact, and are not duplicated.
  - No failures: neither channel nor processes fail.

Chandy-Lamport Algorithm

- *Requirements:*
  - Snapshot should not interfere with normal application actions, and it should not require application to stop sending messages.
  - Any process may initiate algorithm.

Chandy-Lamport Algorithm Intuition

- First, initiator p_i:
  - records its own state.
  - creates a special **marker** message.
  - sends the **marker** to all other process.

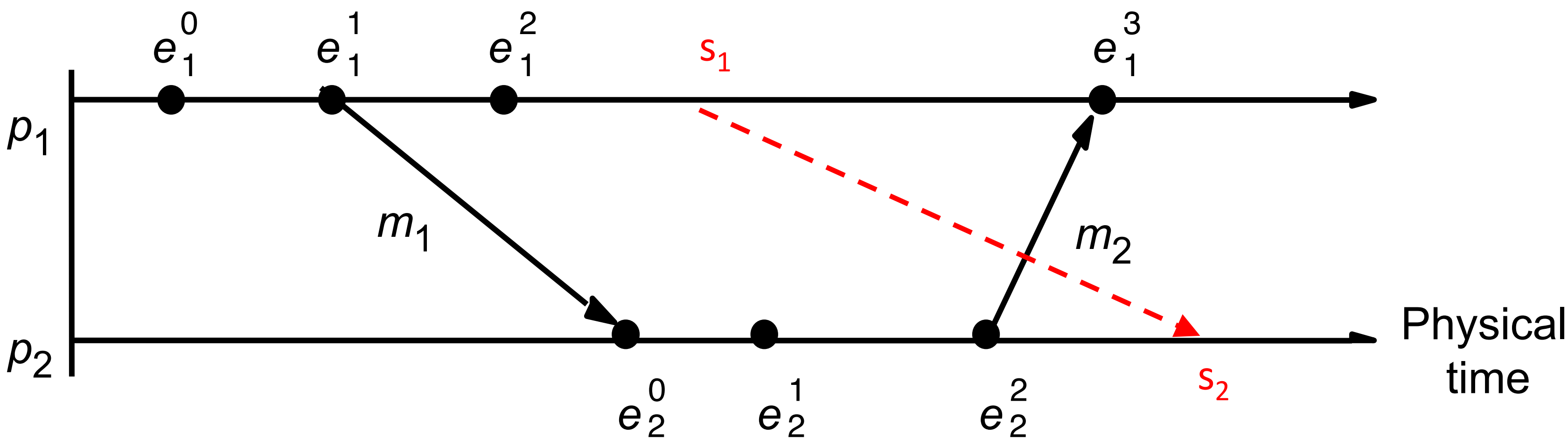
- When a process receives a **marker**.
  - records its own state.

Chandy-Lamport Algorithm Intuition

- First, initiator p_i:
  - records its own state.
  - creates a special **marker** message.
  - sends the **marker** to all other process.

- When a process receives a **marker**.
  - records its own state.

Chandy-Lamport Algorithm Intuition



Cut frontier: $\{e_1{}^2, e_2{}^2\}$

Chandy-Lamport Algorithm Intuition

- First, initiator p_i:
  - records its own state.
  - creates a special **marker** message.
  - sends the **marker** to all other process.

- When a process receives a **marker**.
  - records its own state.

This captures the local state at each process.

How do we ensure the state is consistent?

What about the channel state?

Chandy-Lamport Algorithm Intuition

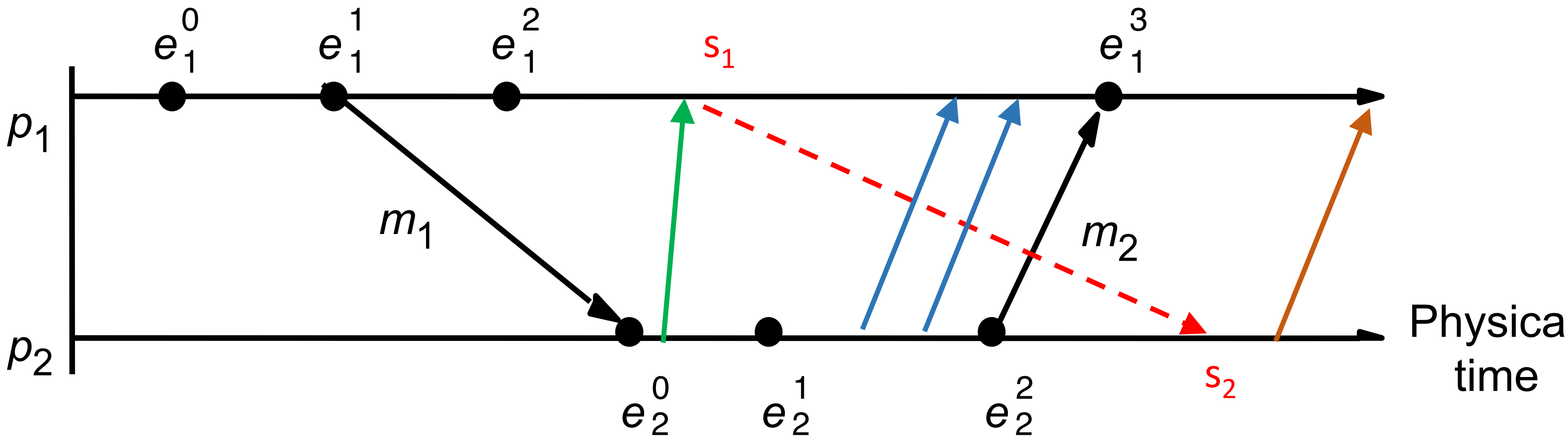
- First, initiator p_i:
  - records its own state.
  - creates a special **marker** message.
  - sends the **marker** to all other process.

- When a process receives a **marker**.
  - If marker is received for the first time.
    - records its own state.
    - sends marker on all other channels.

Leads to a consistent cut (we'll get back to it)

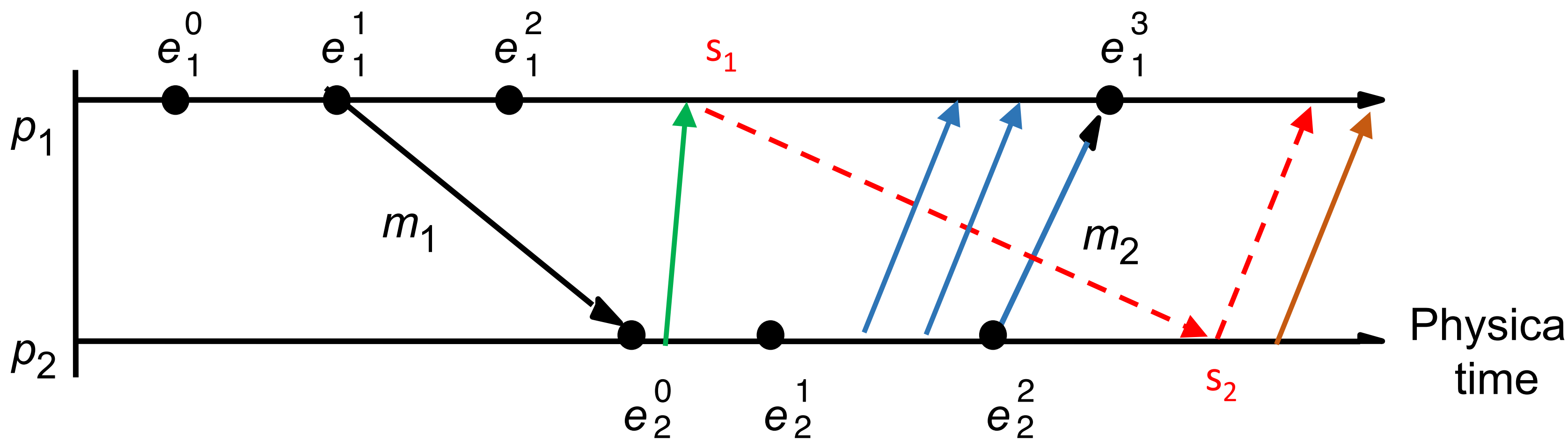
What about the channel state?

Chandy-Lamport Algorithm Intuition



Cut frontier: $\{e_1{}^2, e_2{}^2\}$

Chandy-Lamport Algorithm Intuition



Cut frontier: $\{e_1^2, e_2^2\}$

Chandy-Lamport Algorithm Intuition

- First, initiator p_i:
  - records its own state.
  - creates a special **marker** message.
  - sends the **marker** to all other process.
  - start recording messages received on other channels.
    - until a marker is received on a channel.
- When a process receives a **marker**.
  - If marker is received for the first time.
    - records its own state.
    - sends marker on all other channels.
    - start recording messages received on other channels.
      - until a marker is received on a channel.

Chandy-Lamport Algorithm

- First, initiator **p_i**:
  - records its own state.
  - creates a special **marker** message.
  - for *j=1 to n* except *i*
    - **p_i** sends a **marker** message on outgoing channel **c_{ij}**
    - starts recording the incoming messages on each of the incoming channels at **p_i** : **c_{ji}** (for *j=1 to n* except *i*).

Chandy-Lamport Algorithm

Whenever a process $\mathbf{p_i}$ receives a **marker** message on an incoming channel $\mathbf{c_{ki}}$

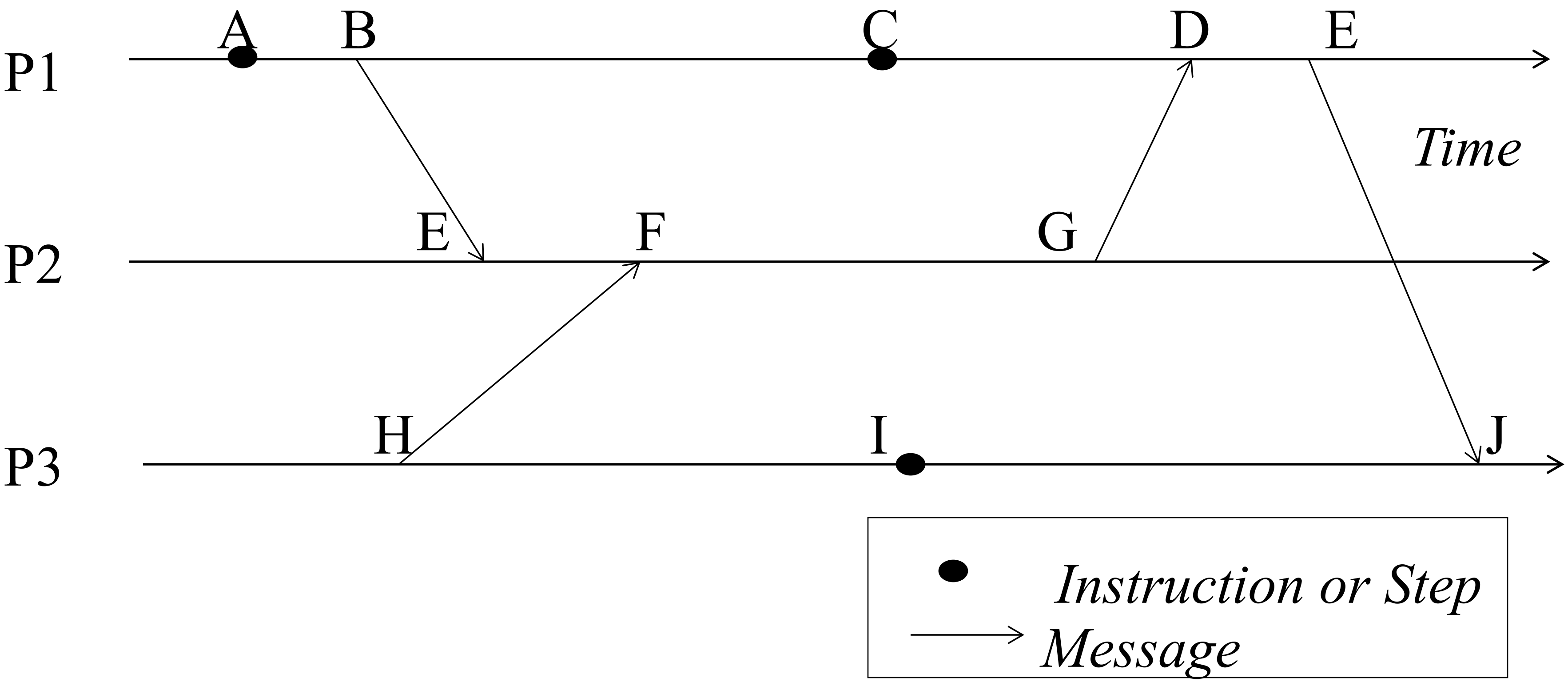
- if (this is the first **marker** $\mathbf{p_i}$ is seeing)
  - $\mathbf{p_i}$ records its own state first
  - marks the state of channel $\mathbf{c_{ki}}$ as "empty"
  - for j=1 to n except i
    - $\mathbf{p_i}$ sends out a **marker** message on outgoing channel $\mathbf{c_{ij}}$
  - starts recording the incoming messages on each of the incoming channels at $\mathbf{p_i}$: $\mathbf{c_{ji}}$ (for *j=1 to n* except *i and k*).
- else // already seen a **marker** message
  - mark the state of channel $\mathbf{c_{ki}}$ as all the messages that have arrived on it since recording was turned on for $\mathbf{c_{ki}}$

Chandy-Lamport Algorithm

The algorithm terminates when

- All processes have received a **marker**
  - To record their own state
- All processes have received a **marker** on all the (n-1) incoming channels
  - To record the state of all channels

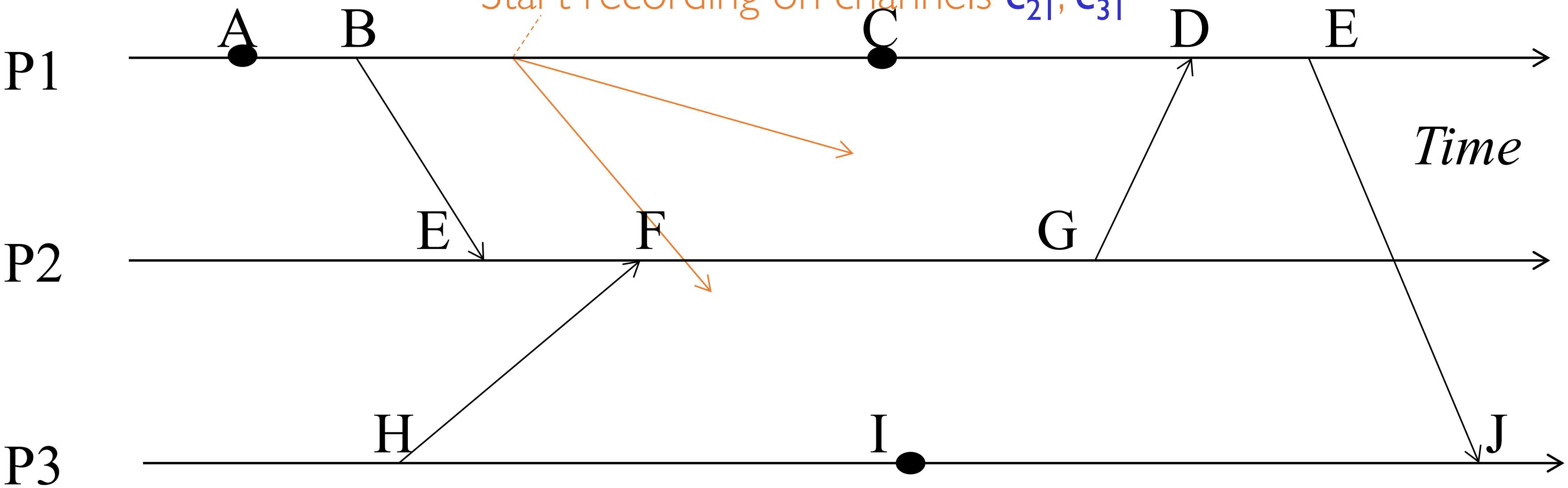
Example



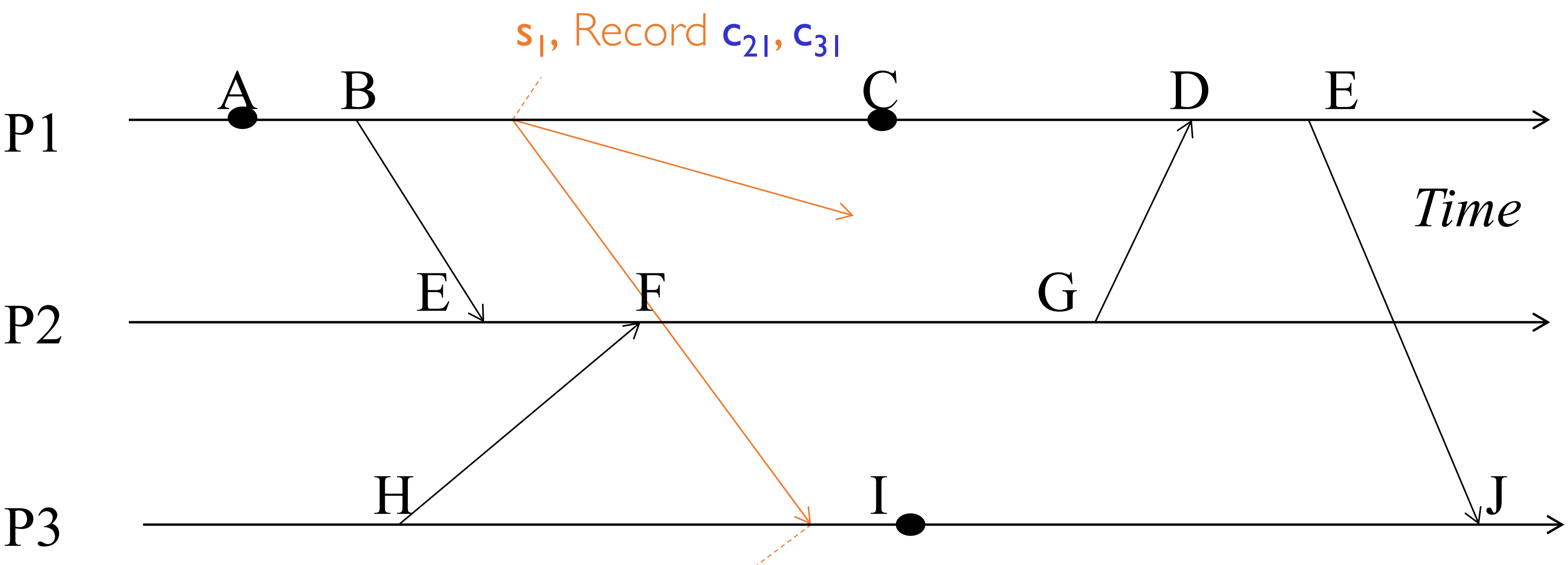
Example

p_1 is initiator:

- Record local state s_1,
- Send out markers
- Start recording on channels c_{21}, c_{31}

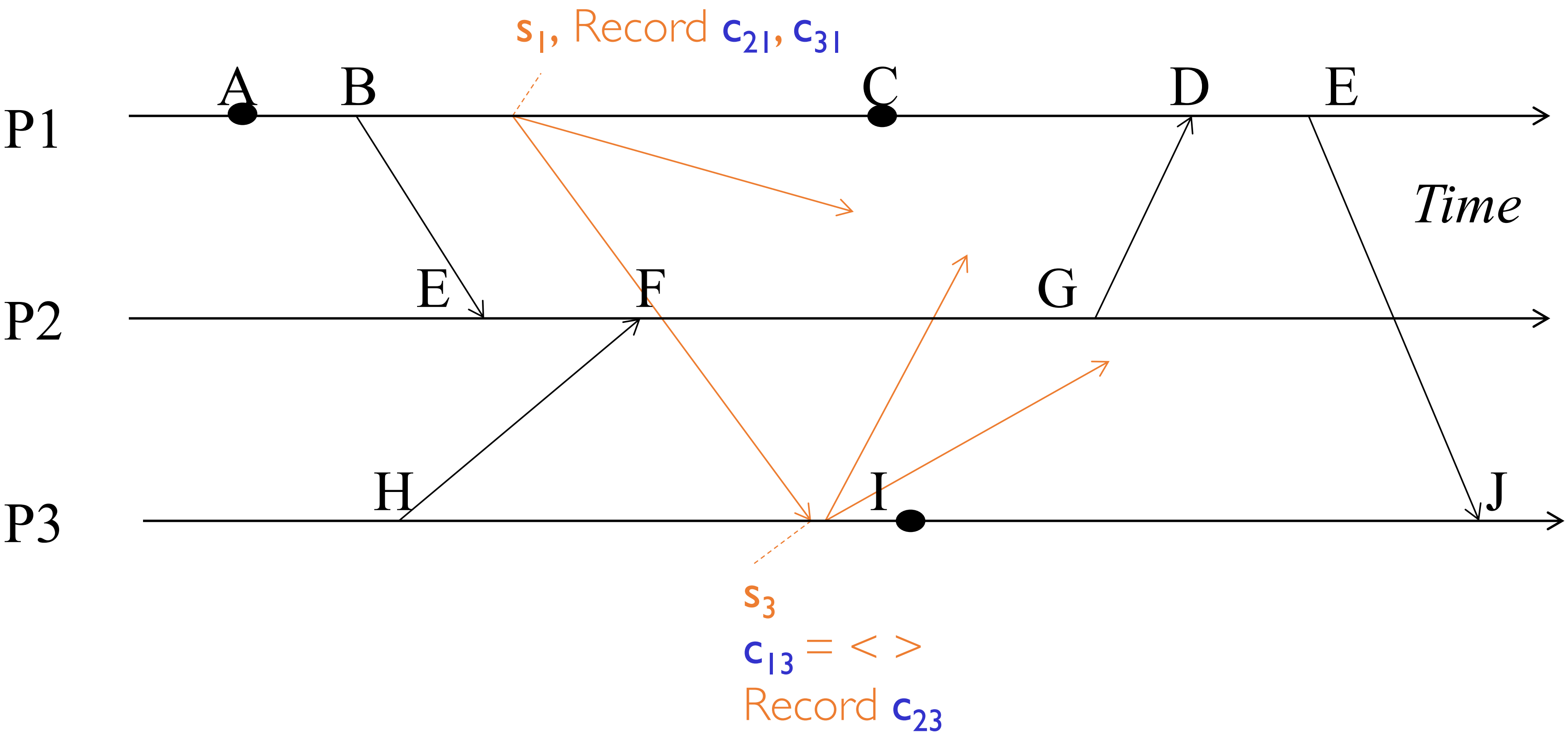


Example

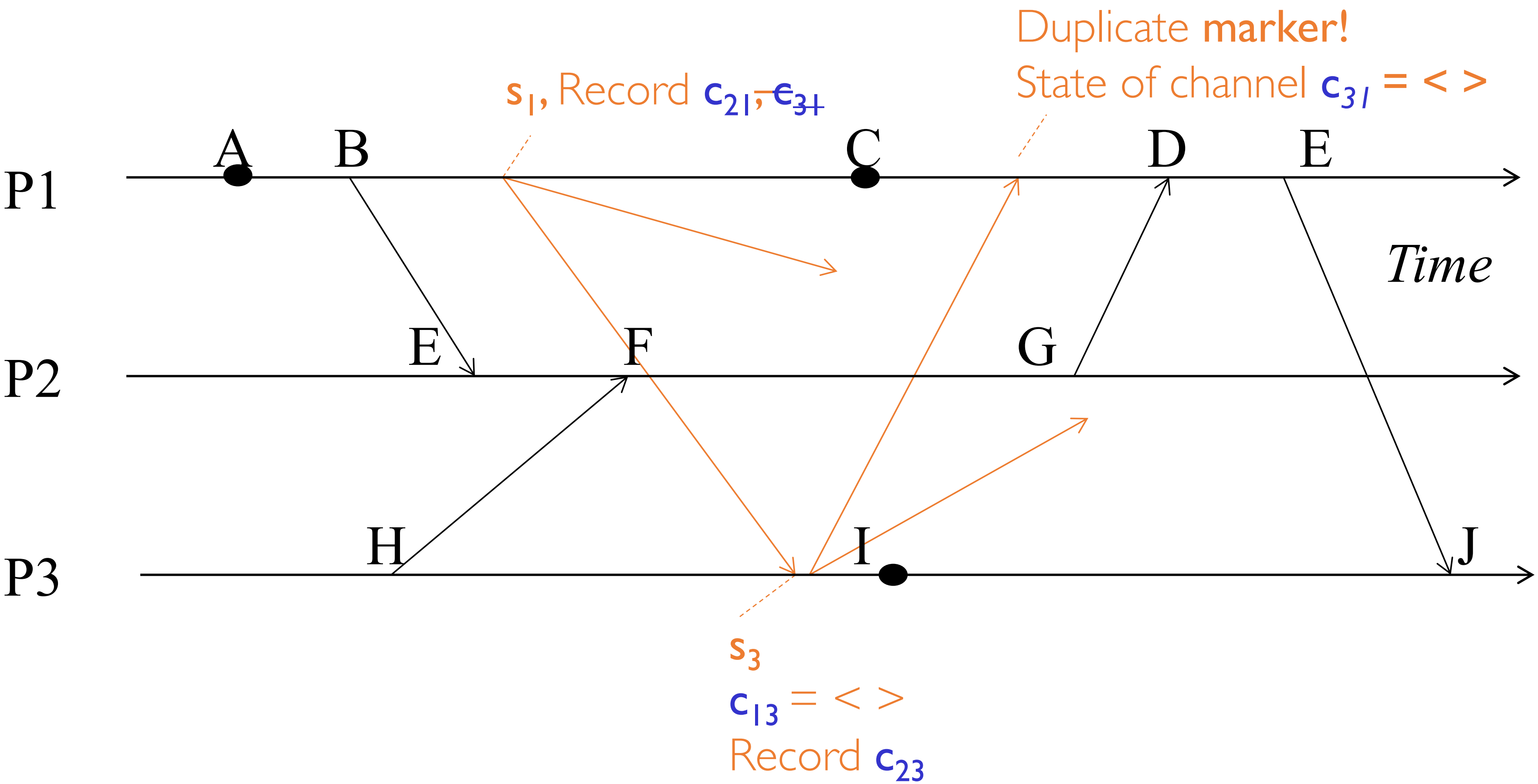


- First **marker**!
- Record own state as s_3
- Mark c_{13} state as empty
- Start recording on other incoming c_{23}
- Send out markers

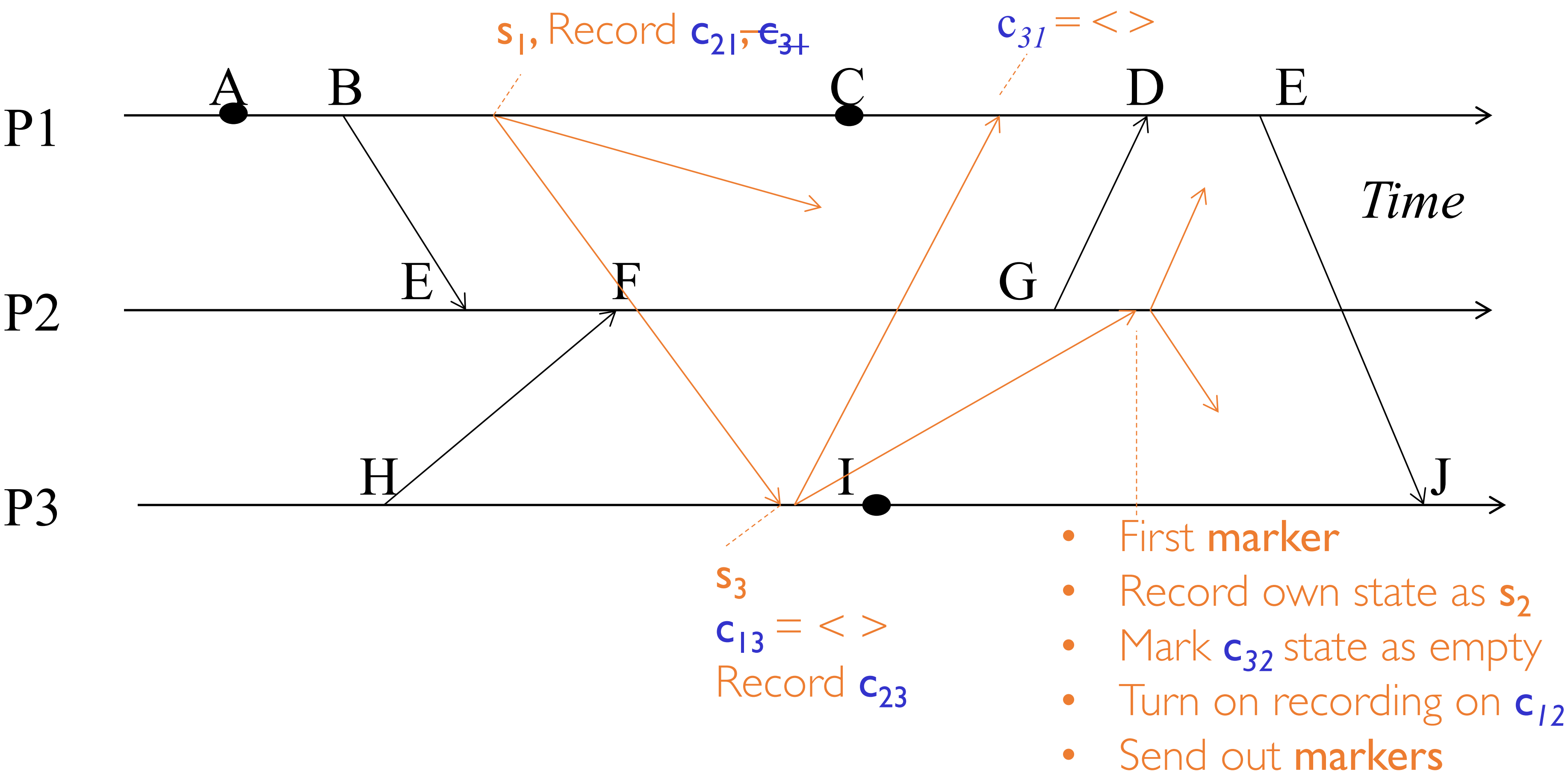
Example



Example

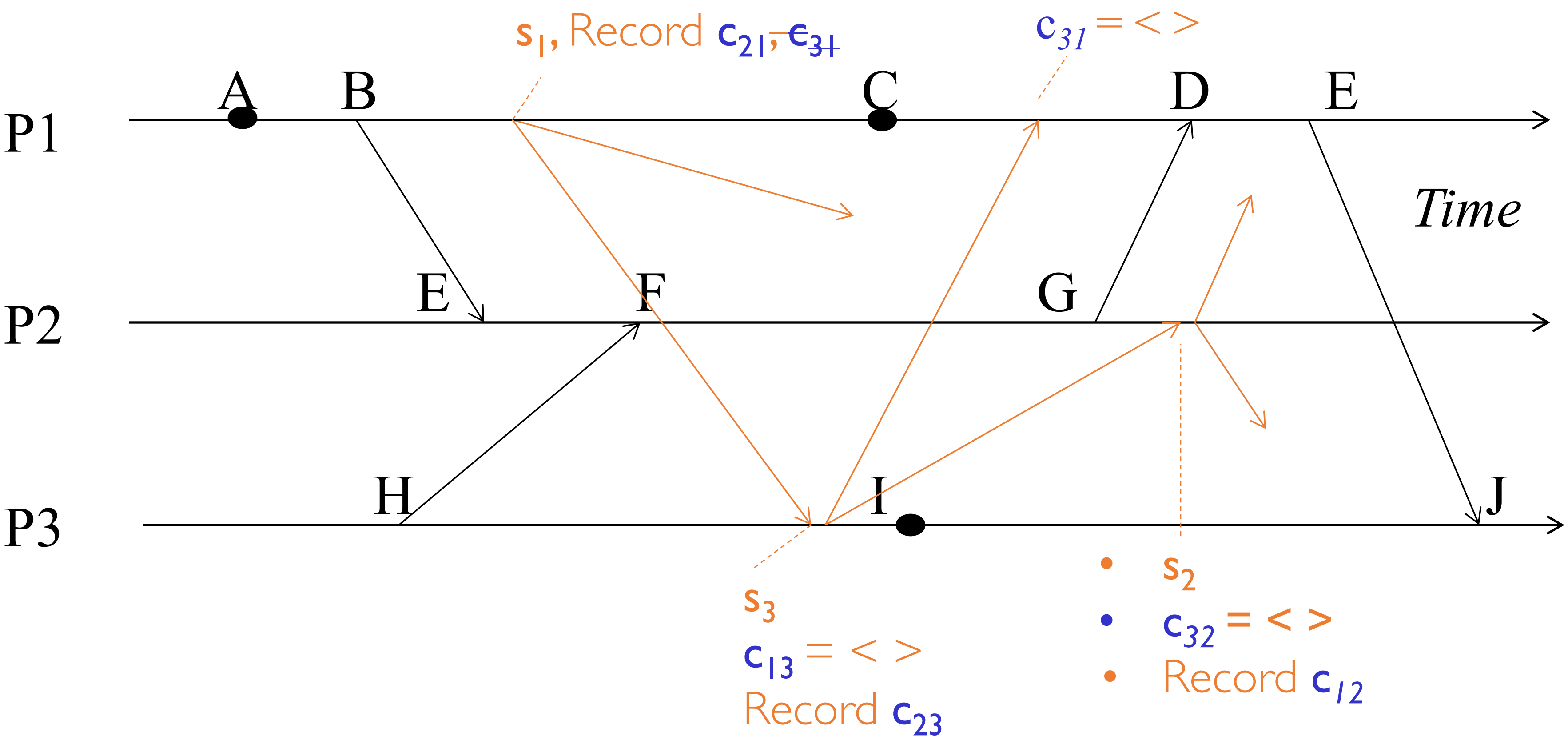

Duplicate marker!
State of channel c31 = < >
s1, Record c21, ~~c31~~
A
B
C
D
E
P1
Time
E
F
G
P2
H
I
J
P3
s3
c13 = < >
Record c23

Example



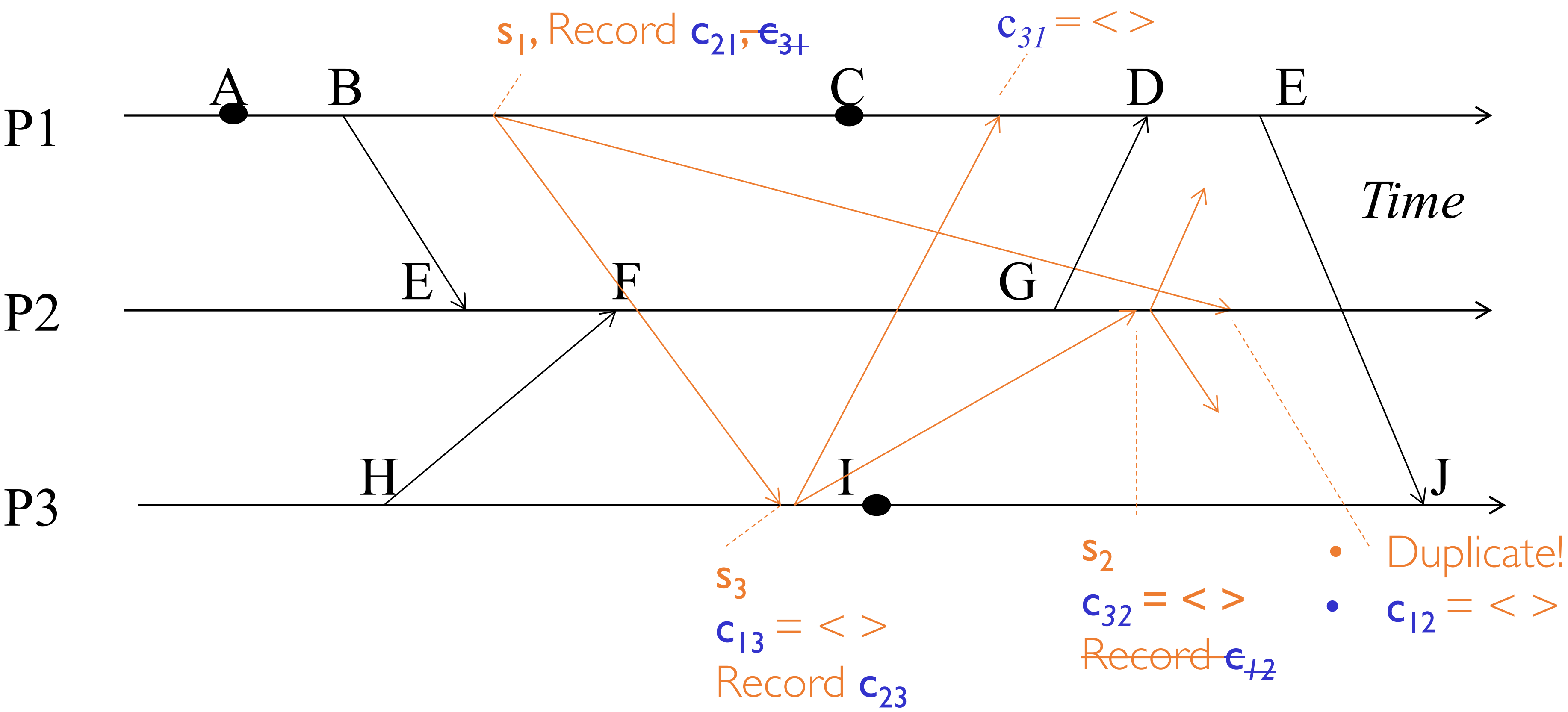
s1, Record c21, c31
c31 = < >
A
B
C
D
E
P1
Time
E
F
G
P2
H
I
J
P3
s3
c13 = < >
Record c23
• First marker
• Record own state as s2
• Mark c32 state as empty
• Turn on recording on c12
• Send out markers

Example



s1, Record c21, c31
c31 = < >
A
B
C
D
E
P1
Time
E
F
G
P2
H
I
J
P3
s3
c13 = < >
Record c23
• s2
• c32 = < >
• Record c12

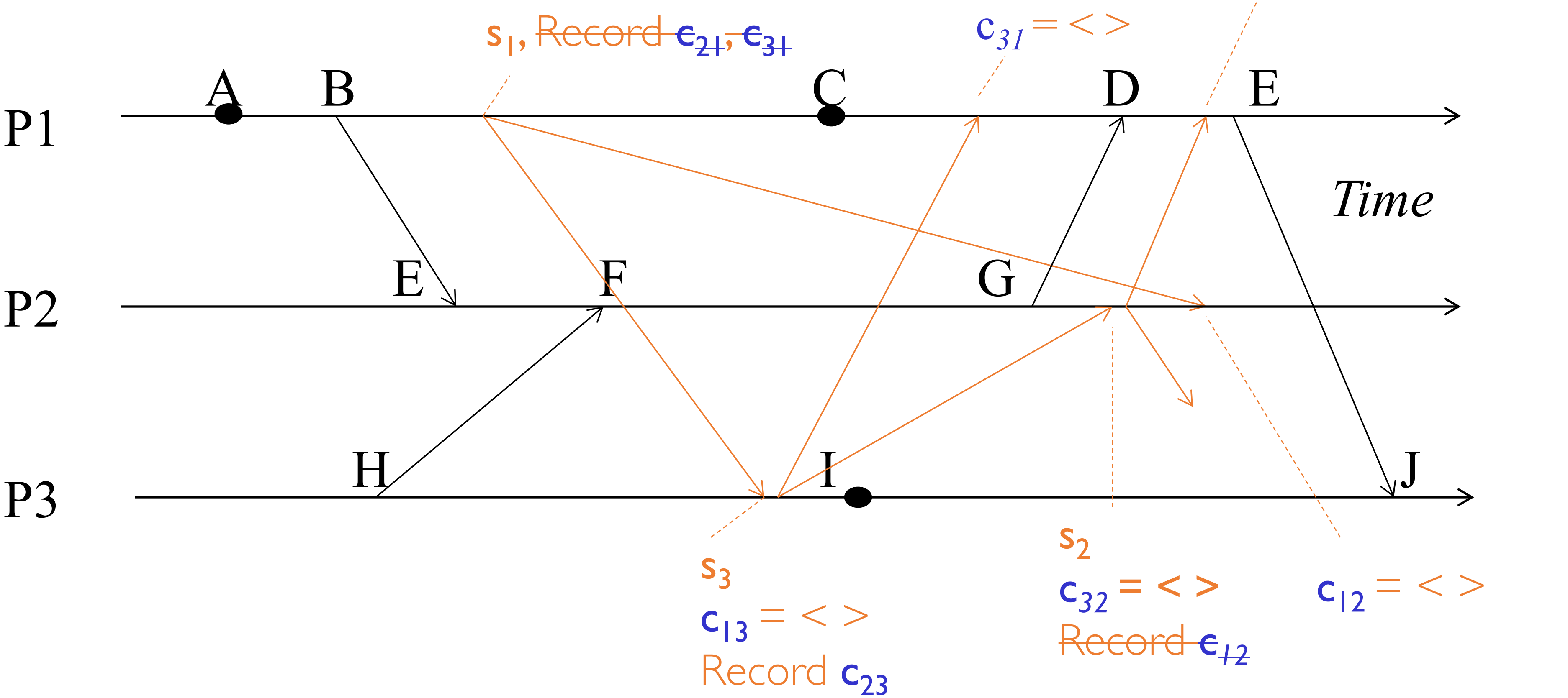
Example



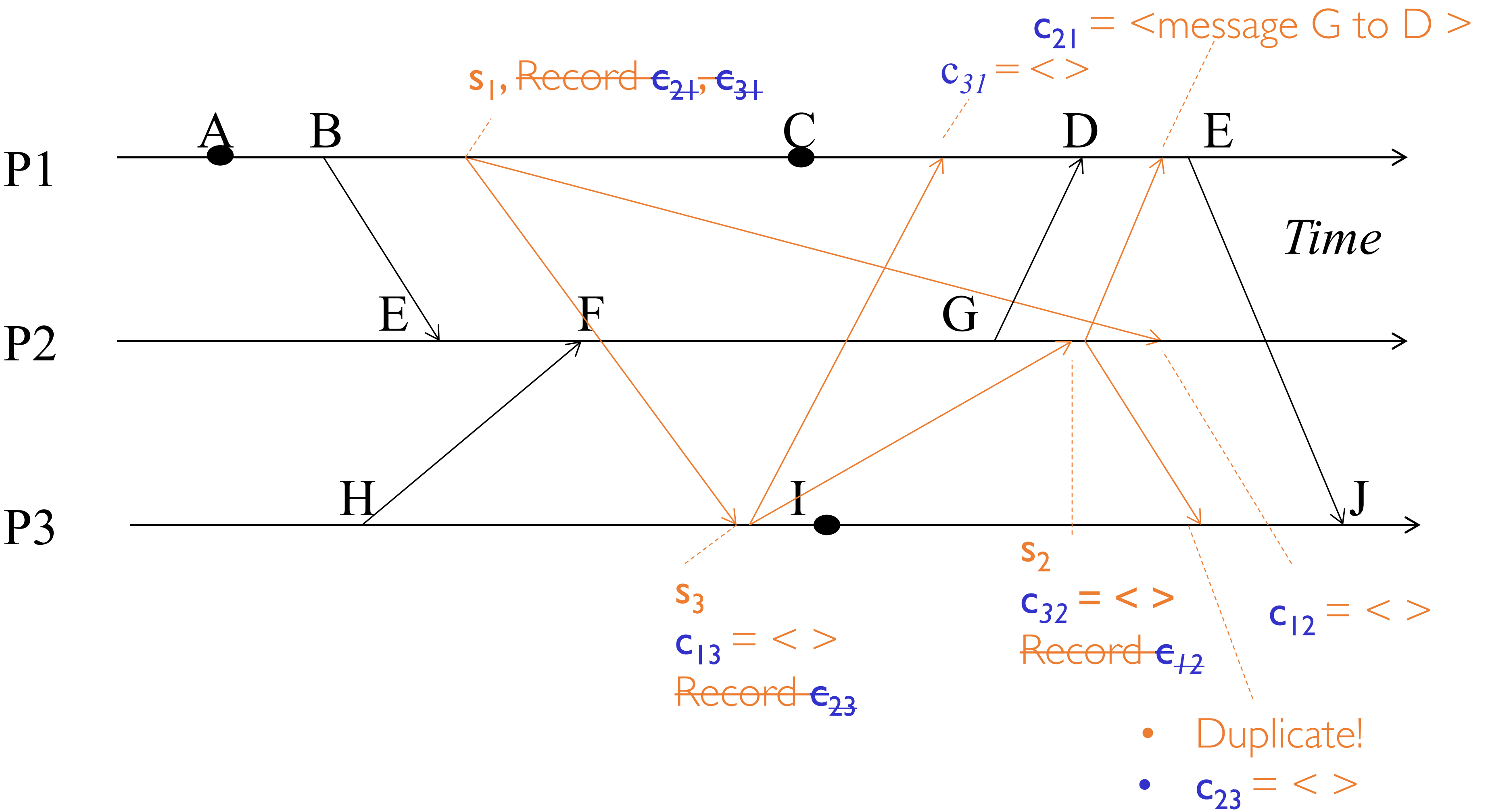
s1, Record c21, c31
c31 = < >
A
B
C
D
E
P1
Time
E
F
G
P2
H
I
J
P3
s3
c13 = < >
Record c23
s2
c32 = < >
Record c12
Duplicate!
c12 = < >

Example

- Duplicate!
- c_{21} = <message G to D >

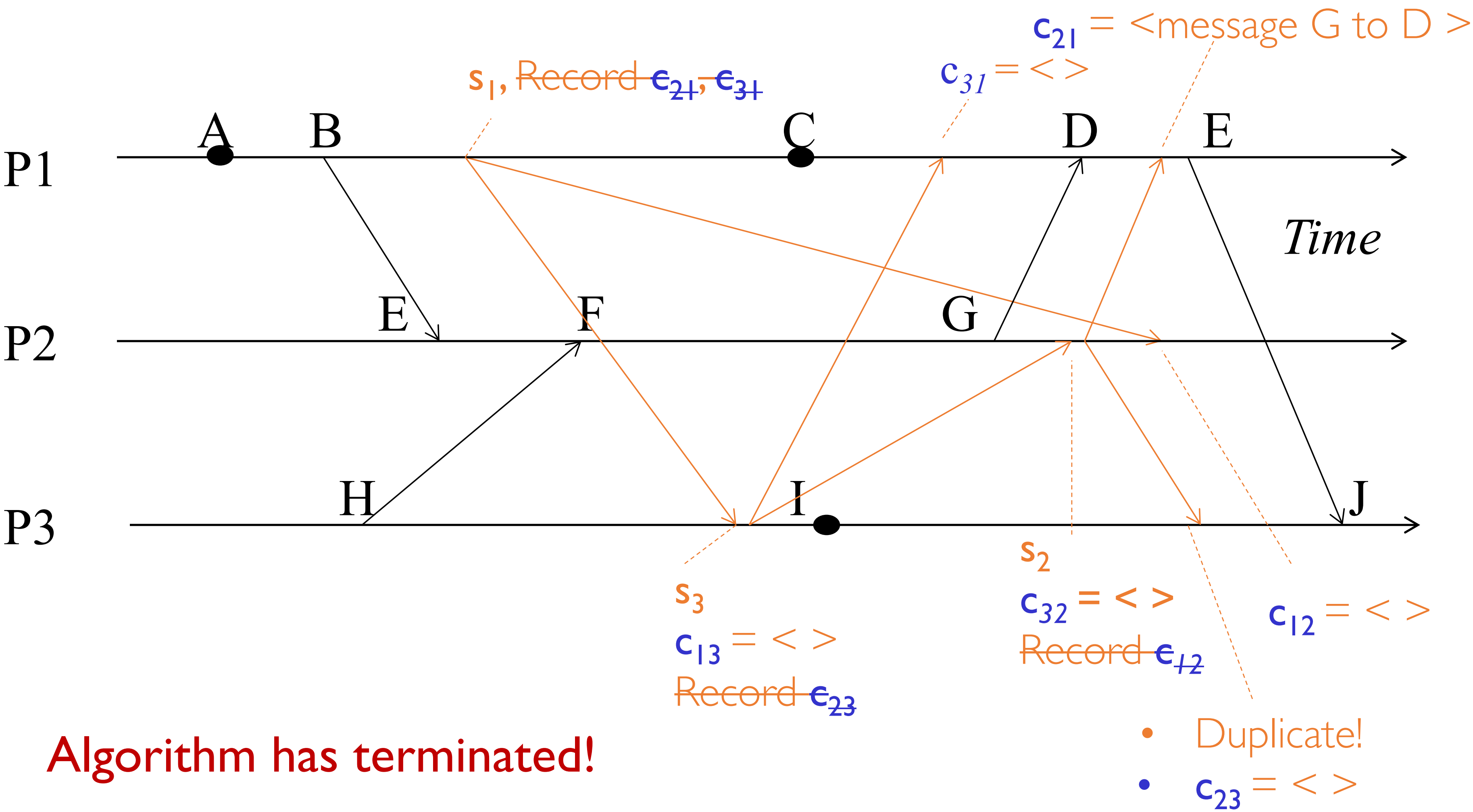


Example



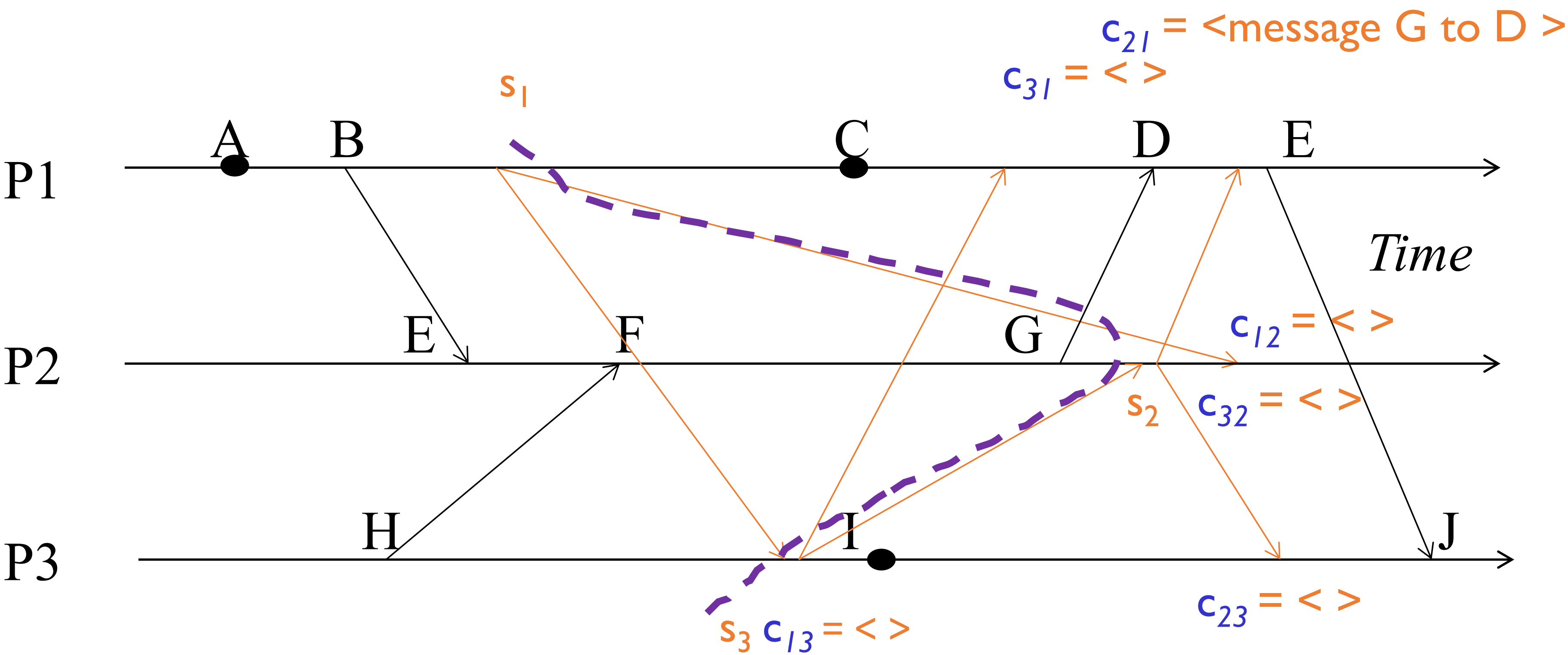
c_{21} = <message G to D >
c_{31} = < >
s_1, ~~Record c_{21}, c_{31}~~
A
B
C
D
E
P1
Time
E
F
G
P2
H
I
J
P3
s_3
c_{13} = < >
~~Record c_{23}~~
s_2
c_{32} = < >
~~Record c_{12}~~
c_{12} = < >
• Duplicate!
• c_{23} = < >

Example



Algorithm has terminated!

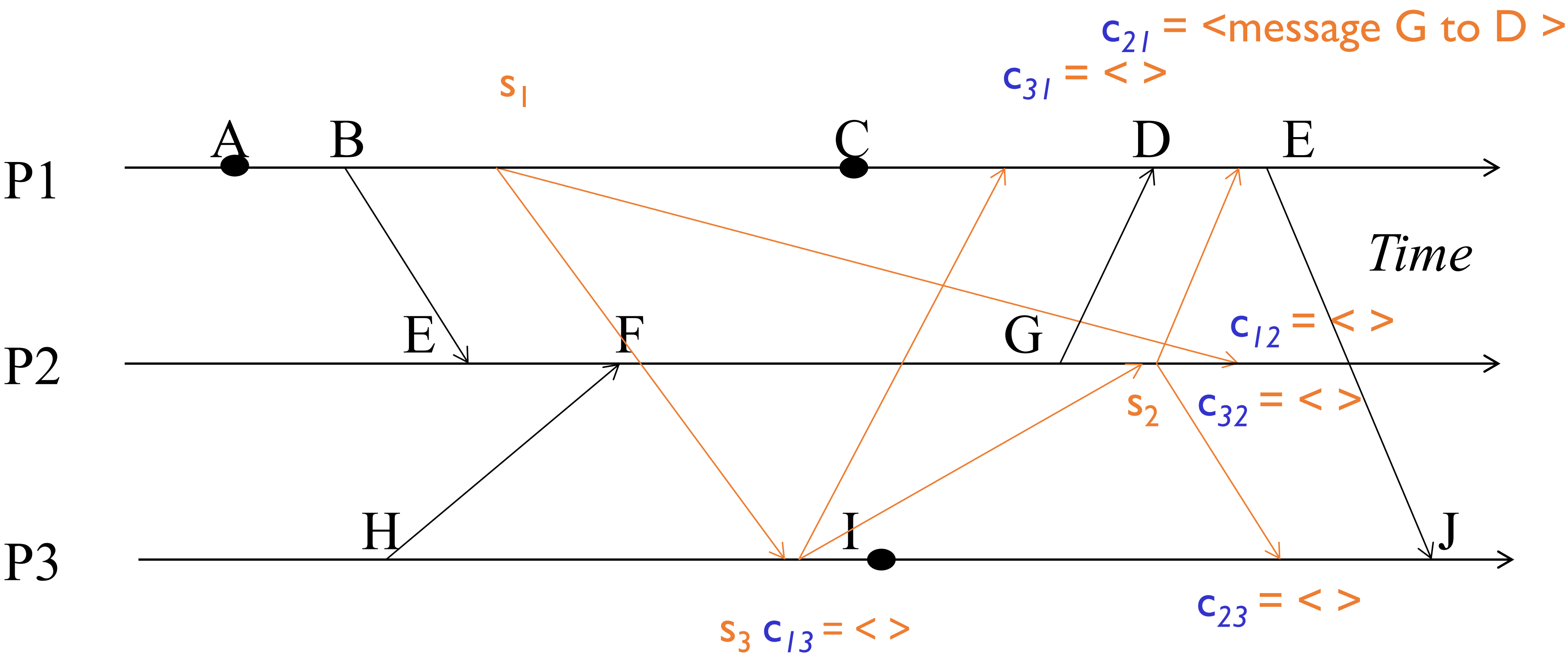
Example



Frontier for the resulting cut:
{B, G, H}

Channel state for the cut:
Only c_{21} has a pending message.

Example



Global snapshots pieces can be collected at a central location.

Chandy-Lamport Algorithm: Properties

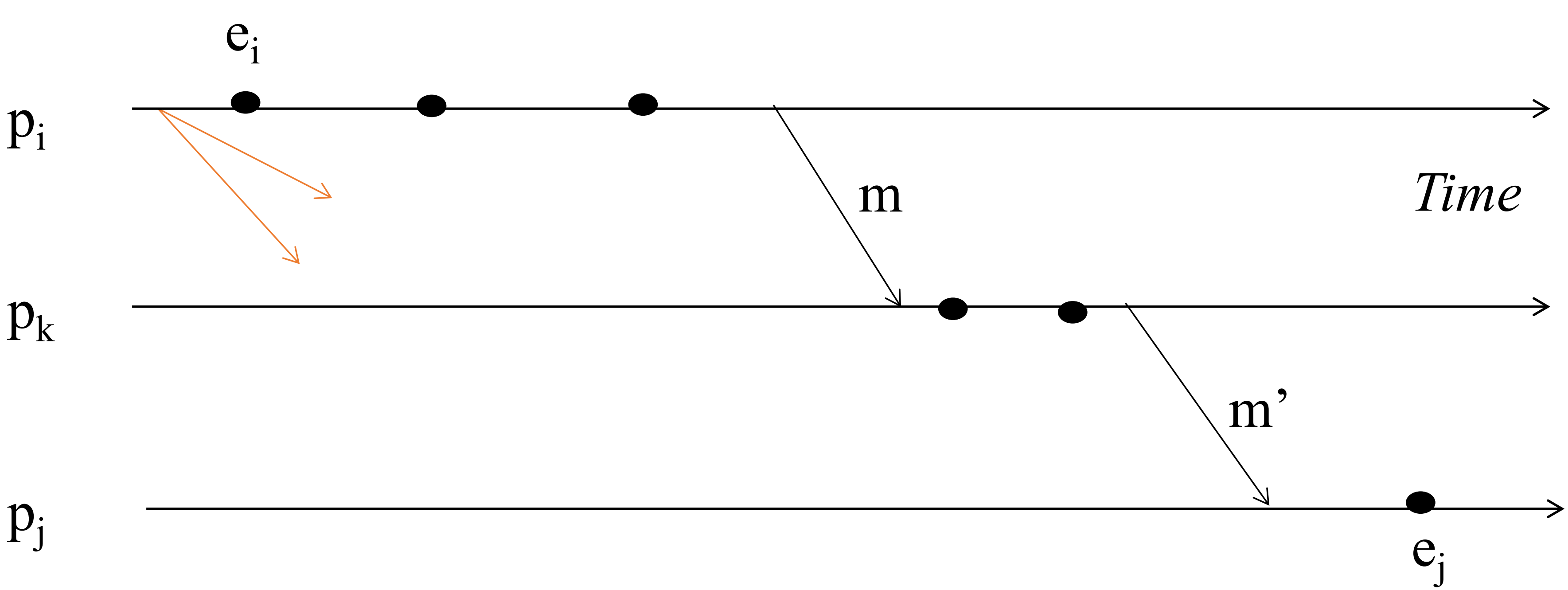
- Any run of the Chandy-Lamport Global Snapshot algorithm creates a consistent cut.
- Let $\mathbf{e}_i$ and $\mathbf{e}_j$ be events occurring at $\mathbf{p}_i$ and $\mathbf{p}_j$, respectively such that
  - $\mathbf{e}_i \rightarrow \mathbf{e}_j$ ($\mathbf{e}_i$ happens before $\mathbf{e}_j$)
- The snapshot algorithm ensures that

  if $\mathbf{e}_j$ is in the cut then $\mathbf{e}_i$ is also in the cut.

- That is: if $\mathbf{e}_j \rightarrow$ < $\mathbf{p}_j$ records its state>, then it must be true that $\mathbf{e}_i \rightarrow$ <$\mathbf{p}_i$ records its state>.

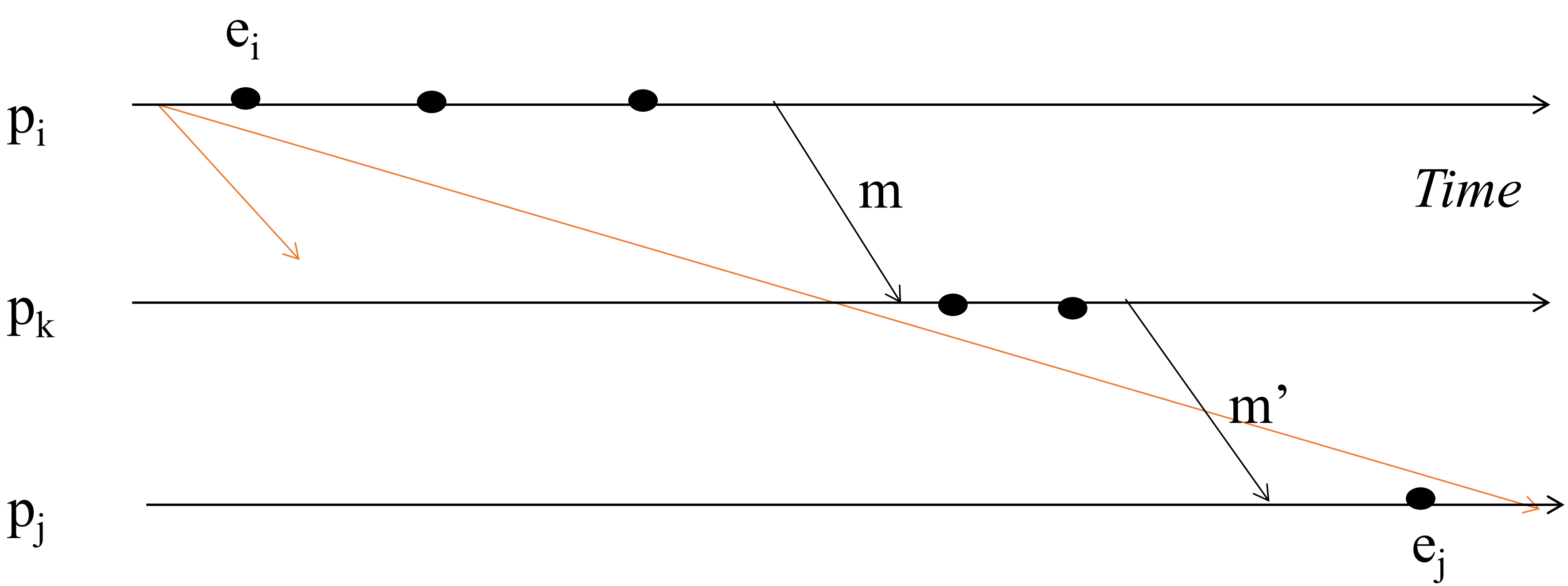
Chandy-Lamport Algorithm: Properties

- If $\mathbf{e}_j$ → < $\mathbf{p}_j$ records its state>, then it must be true that $\mathbf{e}_i$ → <$\mathbf{p}_i$ records its state>.
- By contradiction, *suppose* $\mathbf{e}_j$ → *< p_j records its state>, and <p_i records its state>* → $\mathbf{e}_i$.



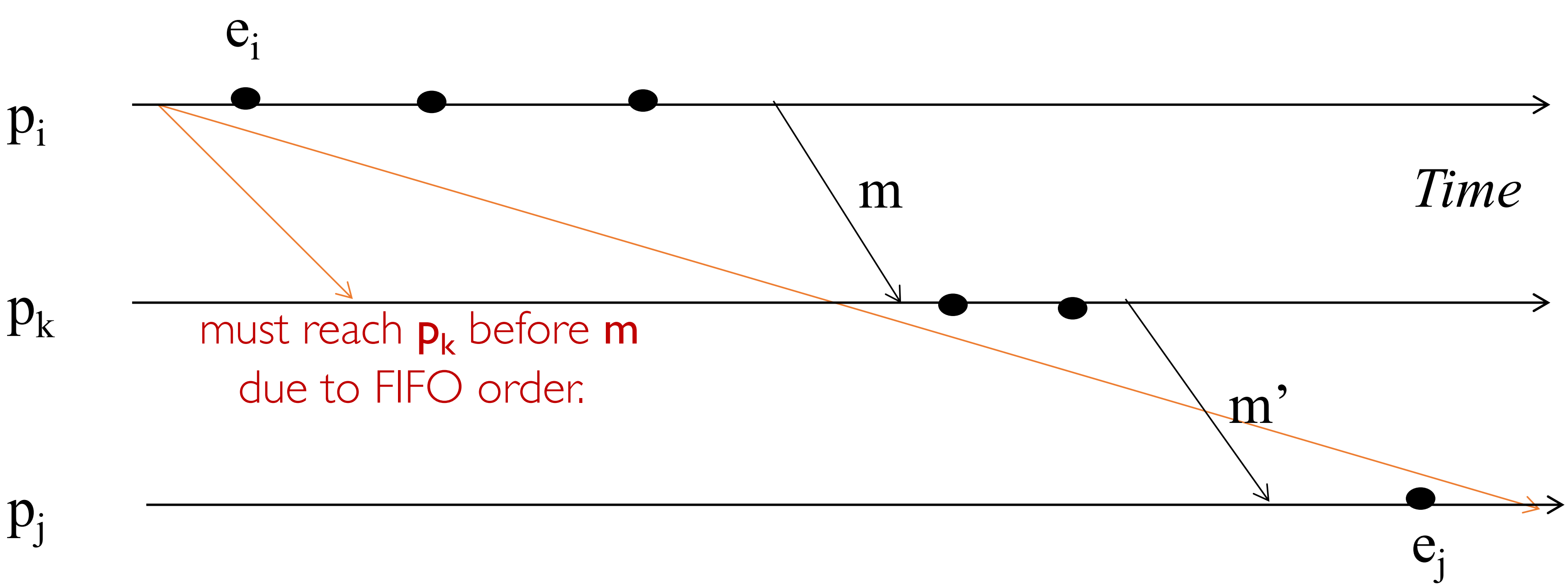
Chandy-Lamport Algorithm: Properties

- If $\mathbf{e_j} \rightarrow$ < $\mathbf{p_j}$ records its state>, then
  it must be true that $\mathbf{e_i} \rightarrow$ <$\mathbf{p_i}$ records its state>.
- By contradiction, *suppose* $\mathbf{e_j} \rightarrow$ *< p_j records its state>, and <p_i records its state>* $\rightarrow \mathbf{e_i}$.



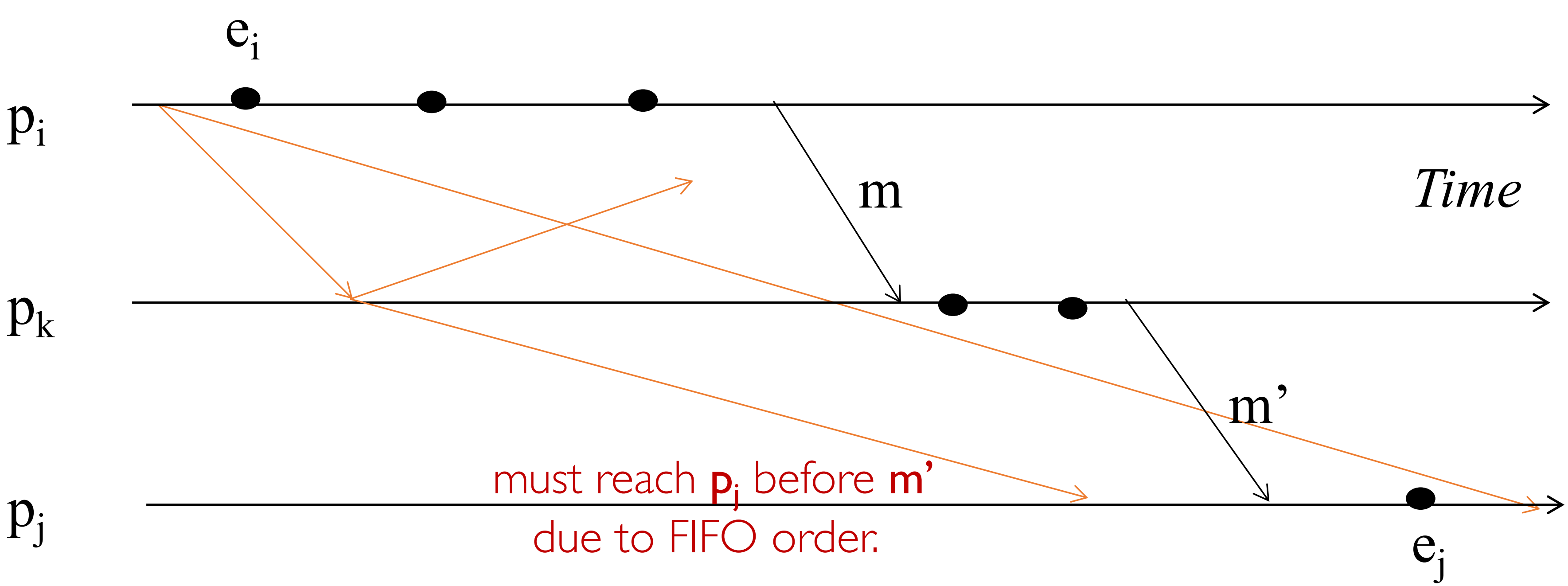
Chandy-Lamport Algorithm: Properties

- If $\mathbf{e_j} \rightarrow$ < $\mathbf{p_j}$ records its state>, then
  it must be true that $\mathbf{e_i} \rightarrow$ <$\mathbf{p_i}$ records its state>.
- By contradiction, *suppose* $\mathbf{e_j} \rightarrow$ *< $\mathbf{p_j}$ records its state>, and <$\mathbf{p_i}$ records its state>* $\rightarrow \mathbf{e_i}$.



Chandy-Lamport Algorithm: Properties

- If e_j → < p_j records its state>, then it must be true that e_i → < p_i records its state>.
- By contradiction, *suppose* e_j → *<* p_j *records its state>, and <*p_i *records its state>* → e_i*.*



Chandy-Lamport Algorithm: Properties

- If $\mathbf{e}_j \rightarrow$ < $\mathbf{p}_j$ records its state>, then
  it must be true that $\mathbf{e}_i \rightarrow$ <$\mathbf{p}_i$ records its state>.
- By contradiction, *suppose* $\mathbf{e}_j \rightarrow$ *< p_j records its state>, and* *<p_i records its state>* $\rightarrow \mathbf{e}_i$.
- Consider the path of app messages (through other processes) that go from $\mathbf{e}_i$ to $\mathbf{e}_j$.
- Due to FIFO ordering, markers on each link in above path will precede regular app messages.
- Thus, since <$\mathbf{p}_i$ records its state> $\rightarrow \mathbf{e}_i$, it must be true that $\mathbf{p}_j$ received a marker before $\mathbf{e}_j$.
- Thus $\mathbf{e}_j$ is not in the cut => contradiction.

Global Snapshot Summary

- The ability to calculate global snapshots in a distributed system is very important.
- But don't want to interrupt running distributed application.
- Chandy-Lamport algorithm calculates global snapshot.
- Obeys causality (creates a consistent cut).
- Can be used to detect global properties.
  - Safety vs. Liveness (next class)