

CLOUD  
COMPUTING  
CONCEPTS

Snapshots

LECTURE A

WHAT IS A GLOBAL SNAPSHOT?

Indranil Gupta (Indy)  
University of Illinois

# HERE'S A SNAPSHOT



# DISTRIBUTED SNAPSHOT

- More often, each country's representative is sitting in their respective capital, and sending messages to each other (say emails).
- How do you calculate a “global snapshot” in that distributed system?
- What does a “global snapshot” even mean?

# IN THE CLOUD

- **In a cloud: each application or service is running on multiple servers**
- **Servers handling concurrent events and interacting with each other**
- **The ability to obtain a “global photograph” of the system is important**
- **Some uses of having a global picture of the system**
  - *Checkpointing*: can restart distributed application on failure
  - *Garbage collection* of objects: objects at servers that don't have any other objects (at any servers) with pointers to them
  - Deadlock detection: Useful in database transaction systems
  - Termination of computation: Useful in batch computing systems like Folding@Home, SETI@Home

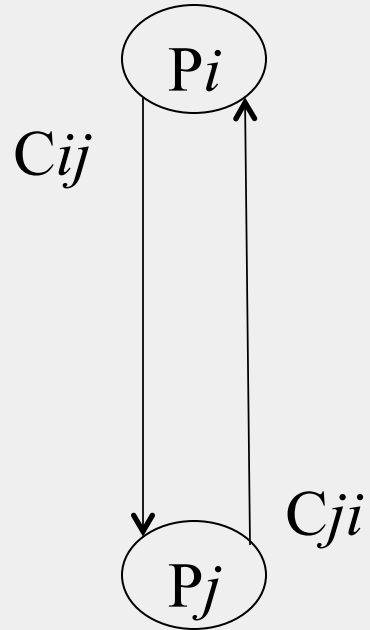
# WHAT'S A GLOBAL SNAPSHOT?

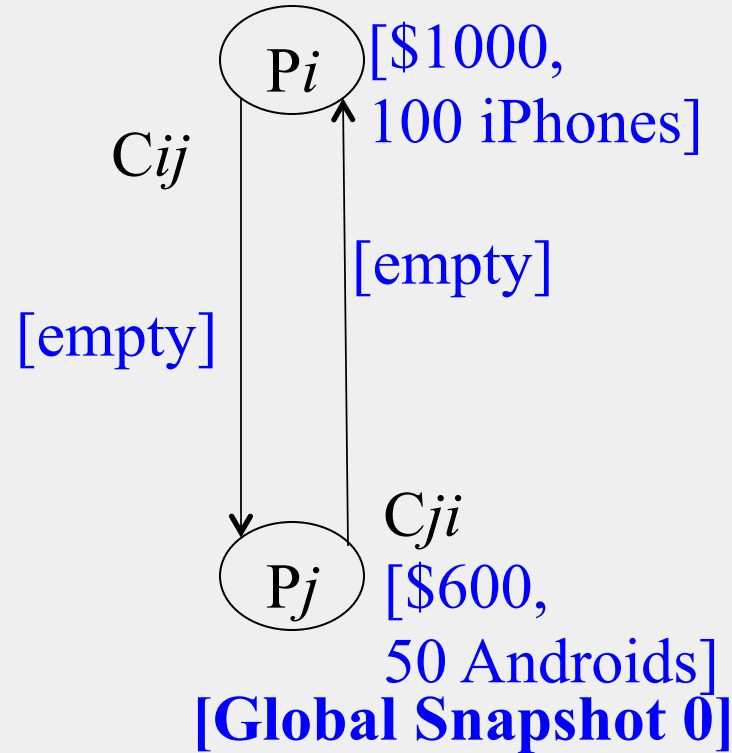
- **Global Snapshot = Global State =**  
Individual state of each process in the distributed system  
+  
Individual state of each communication channel in the distributed system
- Capture the *instantaneous state* of each process
- And the *instantaneous state* of each communication channel, i.e., *messages* in transit on the channels

# OBVIOUS FIRST SOLUTION

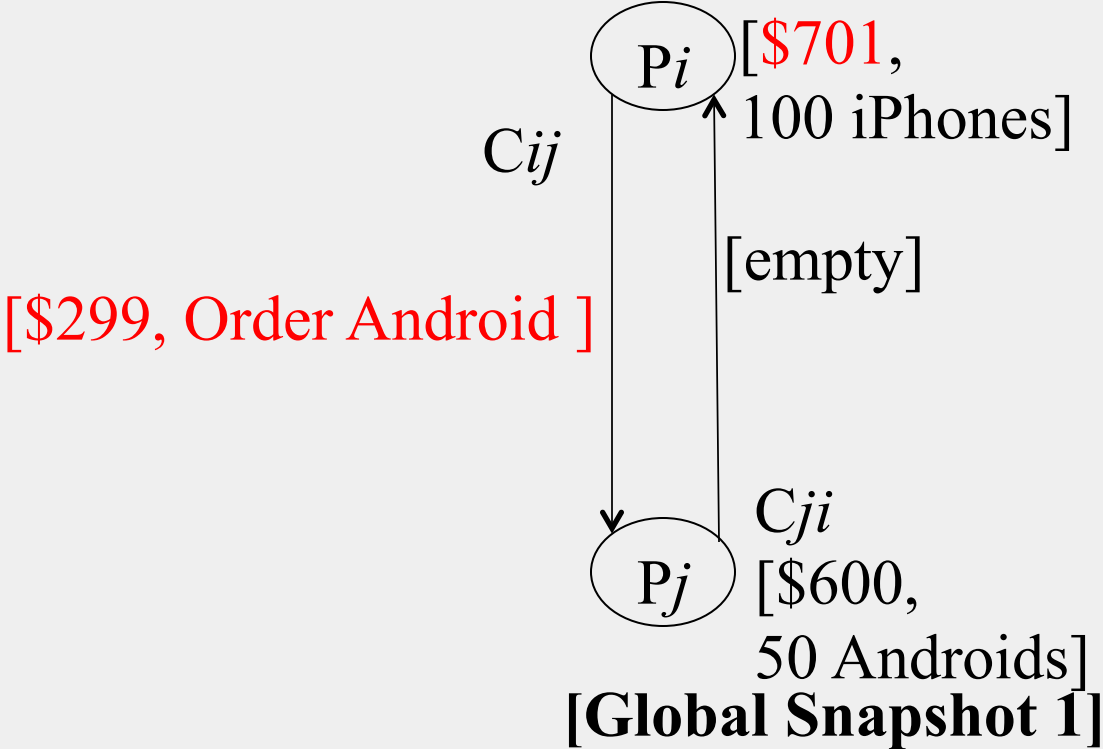
- Synchronize clocks of all processes
- Ask all processes to record their states at known time  $t$
- Problems?
  - Time synchronization always has error
    - Your bank might inform you, “We lost the state of our distributed cluster due to a 1 ms clock skew in our snapshot algorithm.”
  - Also, does not record the state of messages in the channels
- Again: synchronization not required – causality is enough!

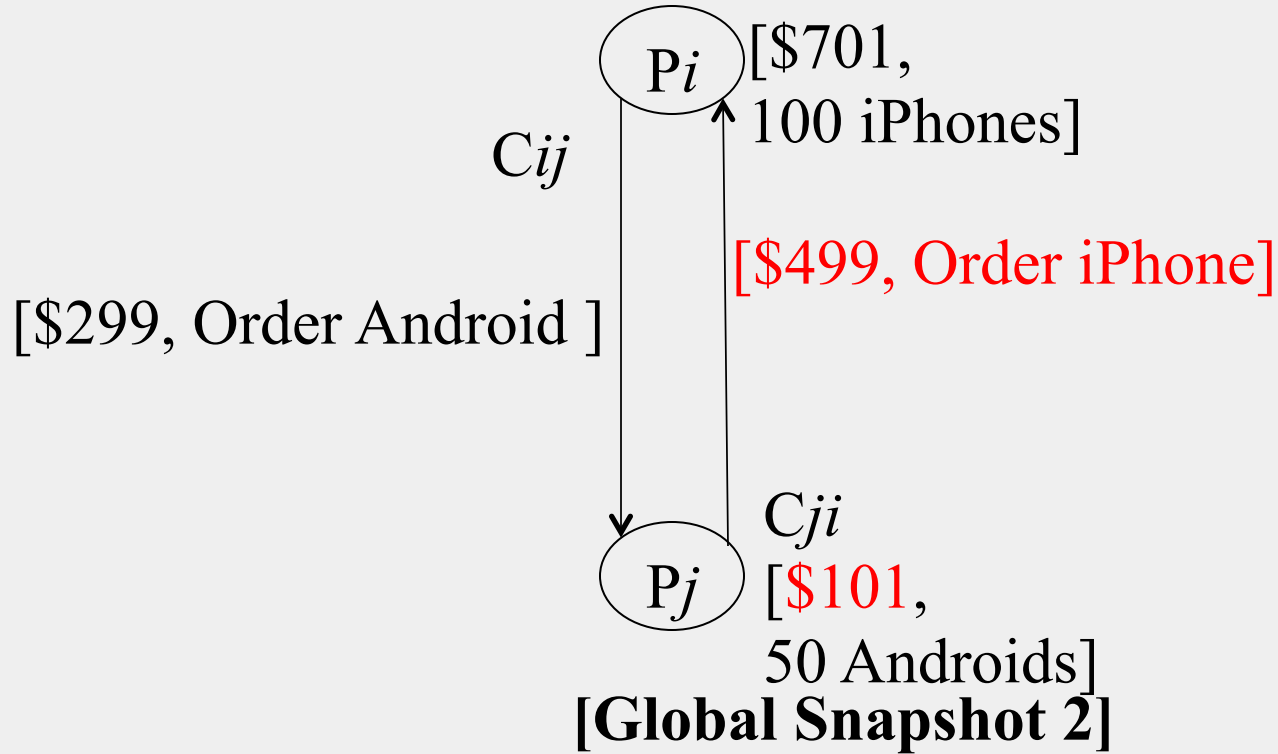
# EXAMPLE

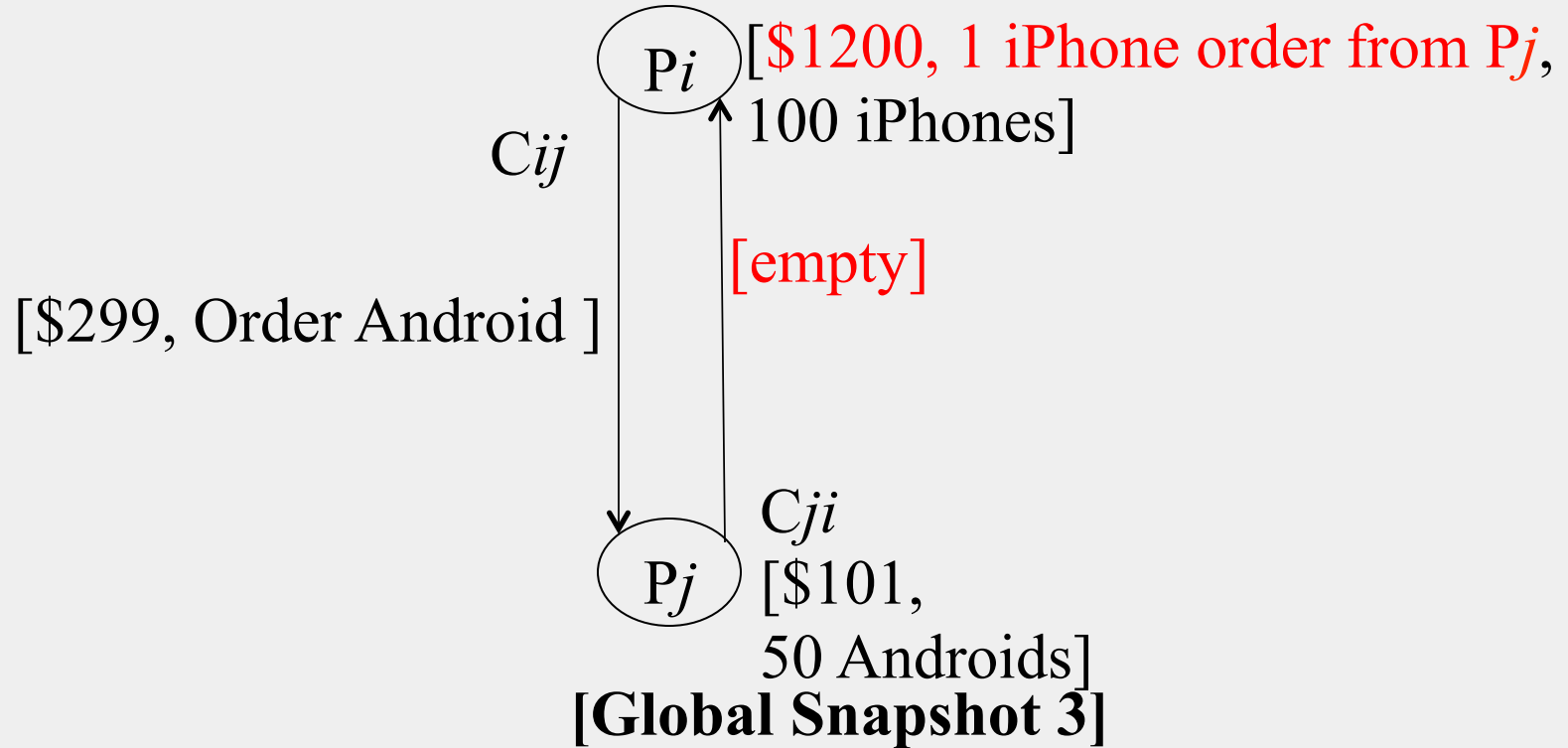




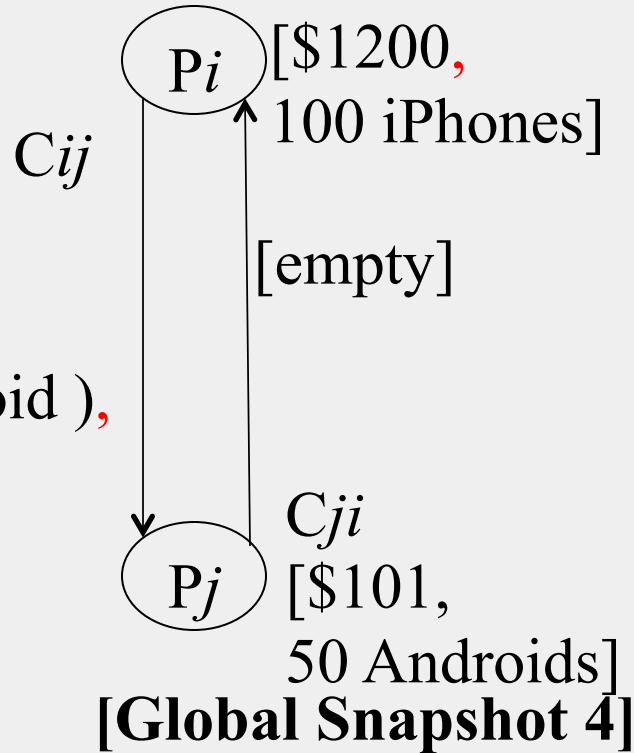




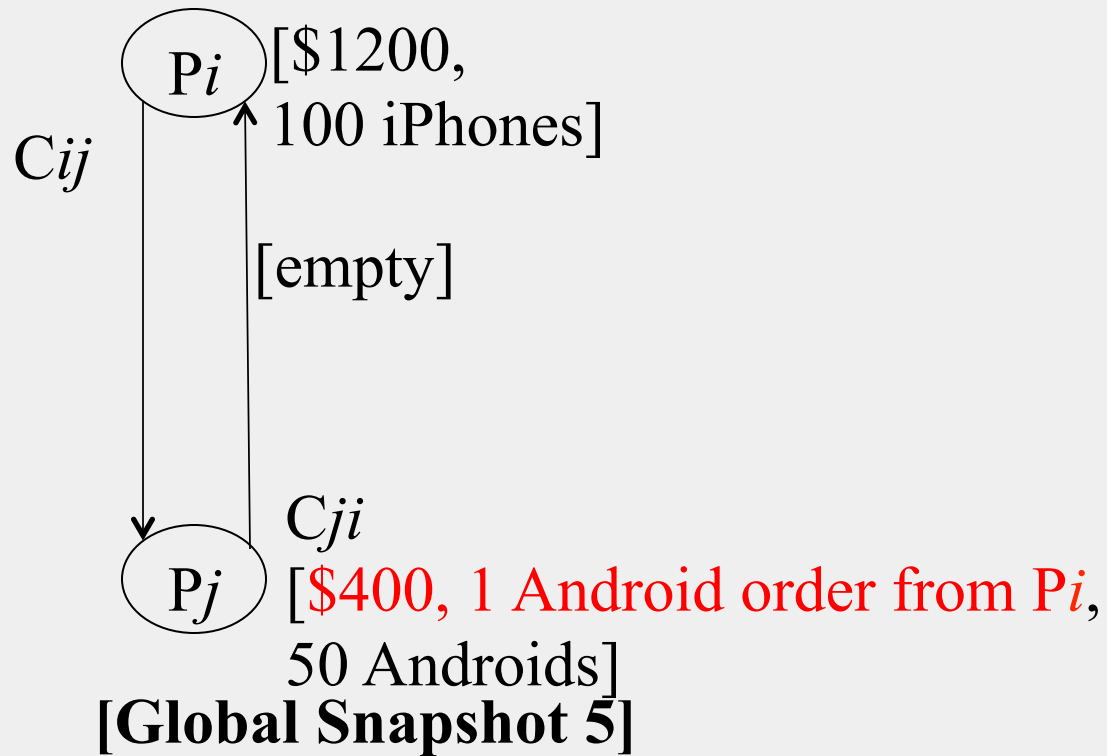


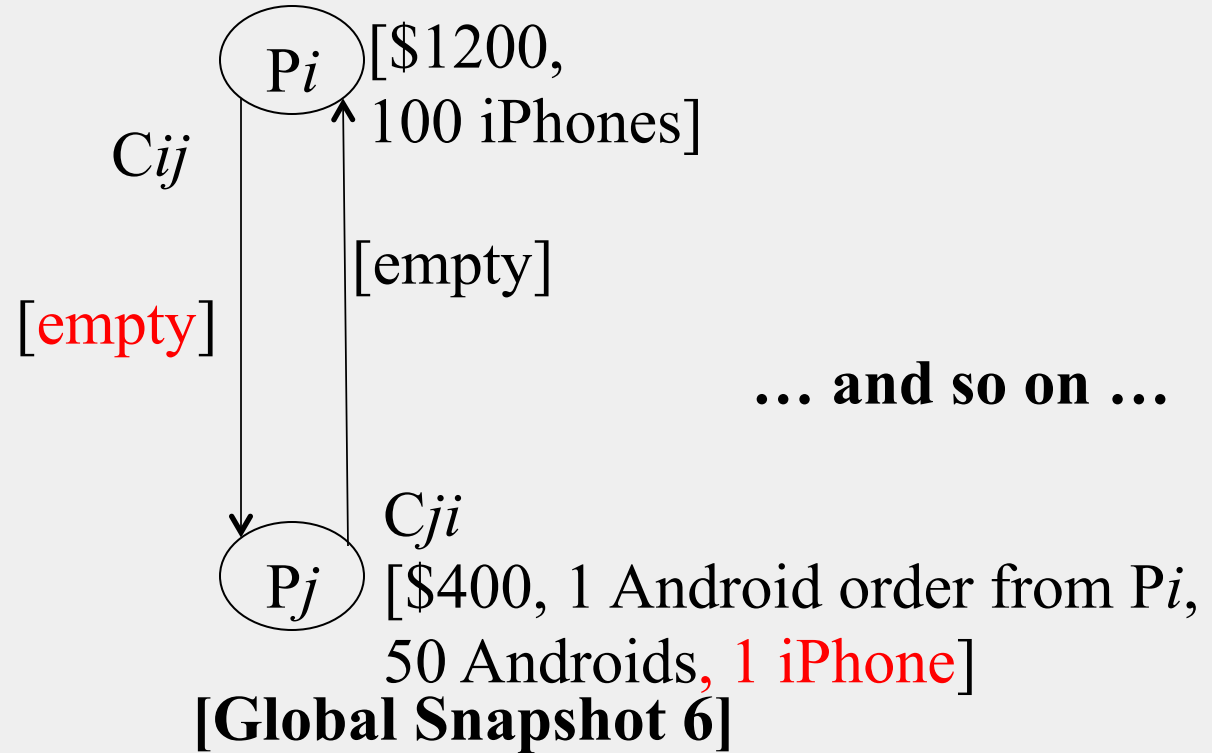


[  
(\$299, Order Android ),  
(1 iPhone)  
]



[  
(1 iPhone)  
]





# MOVING FROM STATE TO STATE

- **Whenever an event happens anywhere in the system, the global state changes**
  - Process receives message
  - Process sends message
  - Process takes a step
- **State to state movement obeys causality**
  - Next: Causal algorithm for Global Snapshot calculation

CLOUD  
COMPUTING  
CONCEPTS

Snapshots

LECTURE B

GLOBAL SNAPSHOT ALGORITHM

Indranil Gupta (Indy)  
University of Illinois



# SYSTEM MODEL

- **Problem:** Record a global snapshot (state for each process, and state for each channel)
- *System Model:*
  - $N$  processes in the system
  - There are two uni-directional communication channels between each ordered process pair :  $P_j \rightarrow P_i$  and  $P_i \rightarrow P_j$
  - Communication channels are FIFO-ordered
    - First in First out
  - No failure
  - All messages arrive intact, and are not duplicated
    - Other papers later relaxed some of these assumptions

# REQUIREMENTS

- **Snapshot should not interfere with normal application actions, and 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 heap, registers, program counter, code, etc. (essentially the coredump)
- **Global state is collected in a distributed manner**
- **Any process may initiate the snapshot**
  - We'll assume just one snapshot run for now

# CHANDY-LAMPORT GLOBAL SNAPSHOT ALGORITHM

- First, Initiator  $P_i$  **records** its own state
- Initiator process creates special messages called “**Marker**” messages
  - Not an application message, does not interfere with application messages
- **for**  $j=1$  **to**  $N$  **except**  $i$ 
  - $P_i$  **sends** out a Marker message on outgoing channel  $C_{ij}$ 
    - $(N-1)$  channels
- **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 GLOBAL SNAPSHOT ALGORITHM (2)

Whenever a process  $P_i$  receives a Marker message on an incoming channel  $C_{ji}$

- **if** (this is the first Marker  $P_i$  is seeing)
  - $P_i$  **records** its own state first
  - **Marks the state of channel  $C_{ji}$  as “empty”**
  - for  $j=1$  to  $N$  except  $i$ 
    - $P_i$  **sends** out 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  $i$ )
- **else // already seen a Marker message**
  - **Mark** the state of channel  $C_{ji}$  as all the messages that have arrived on it **since recording was turned on for  $C_{ji}$**

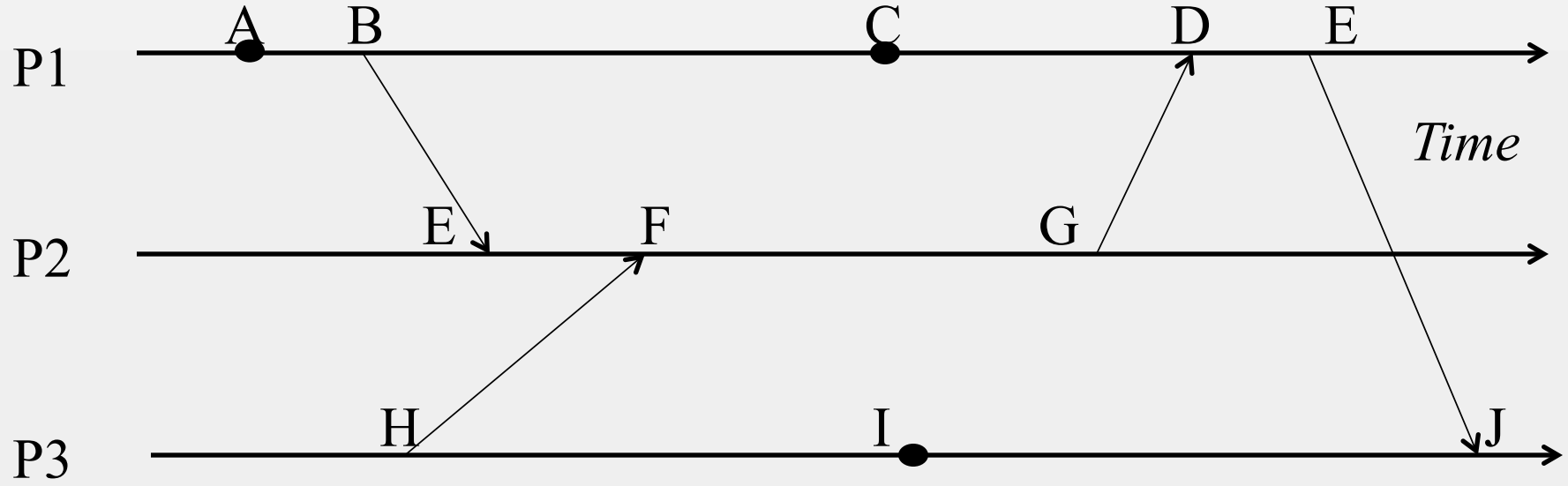
# CHANDY-LAMPORT GLOBAL SNAPSHOT ALGORITHM (3)

**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 at each
  - To record the state of all channels

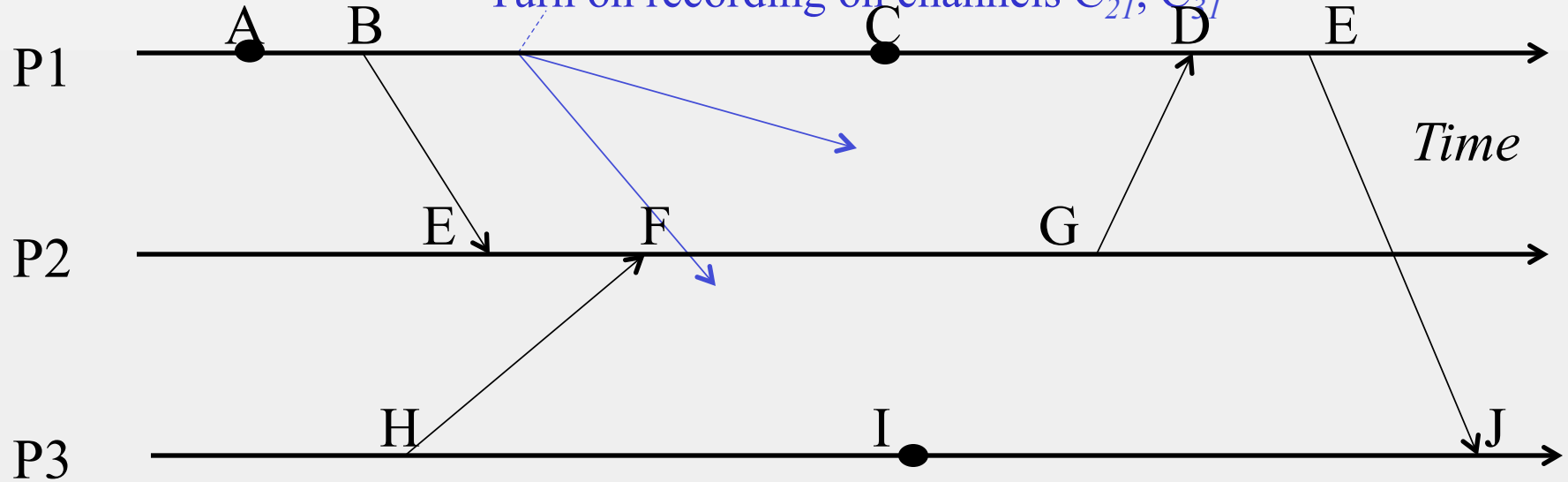
**Then, (if needed), a central server collects all these partial state pieces to obtain the full global snapshot**

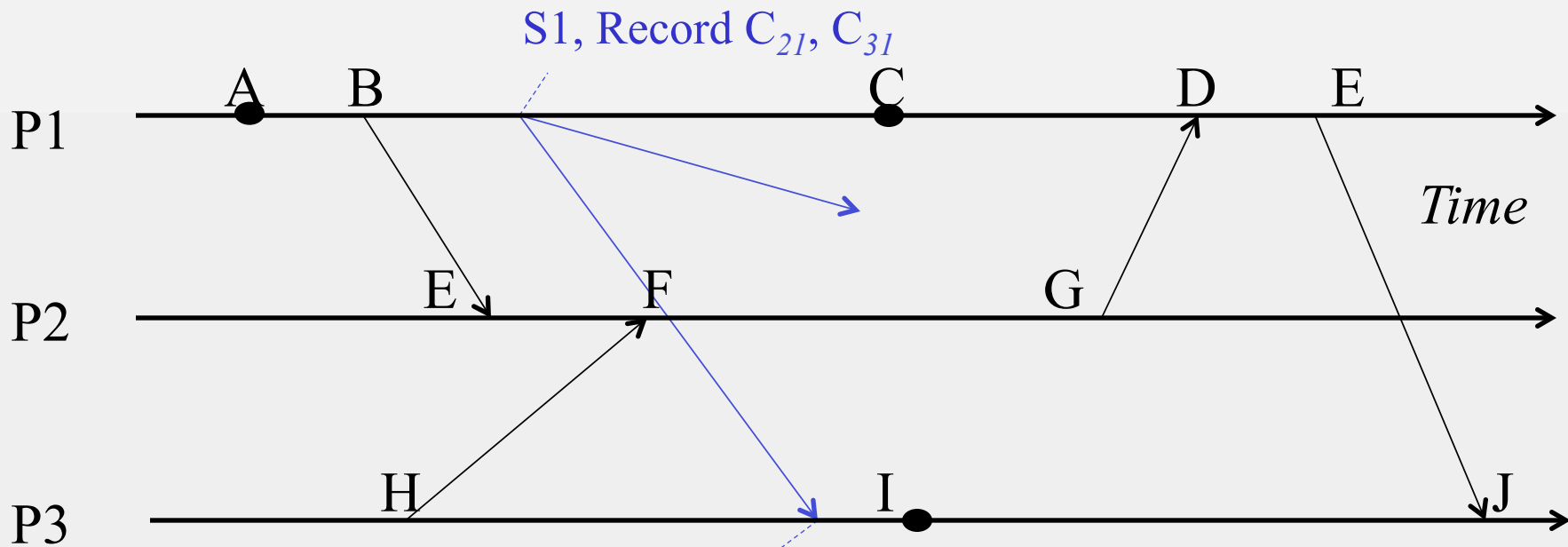
# EXAMPLE



P1 is Initiator:

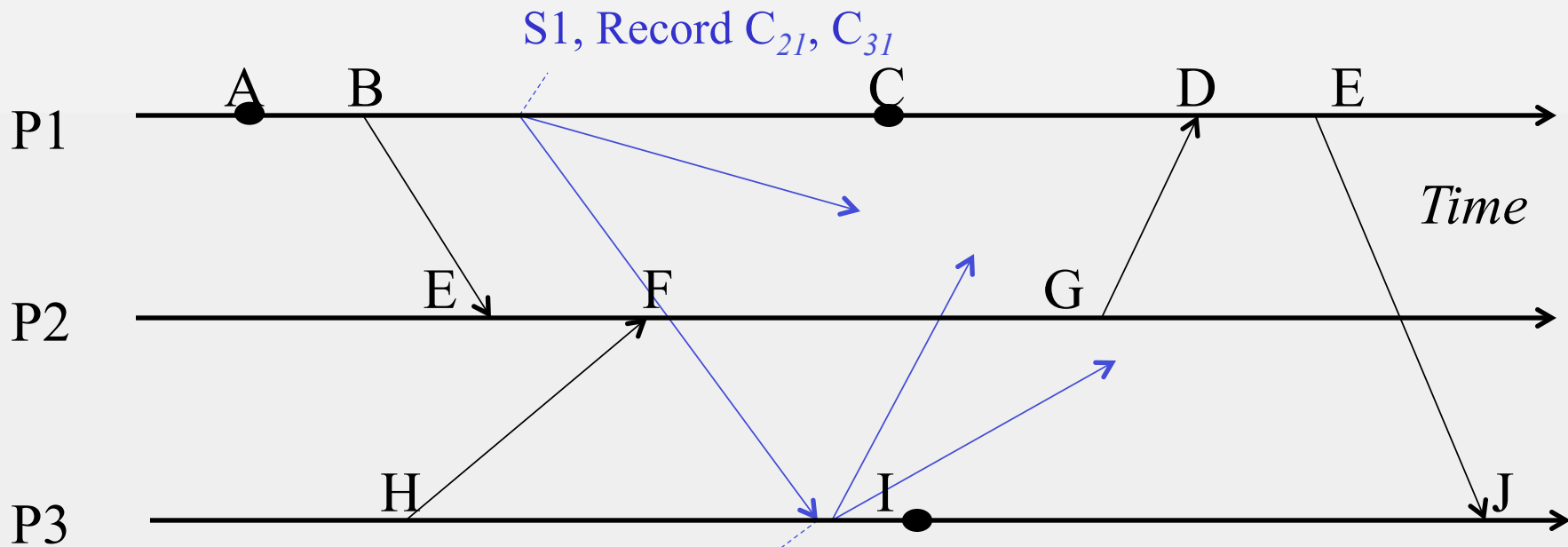
- Record local state S1,
- Send out markers
- Turn on recording on channels  $C_{21}, C_{31}$



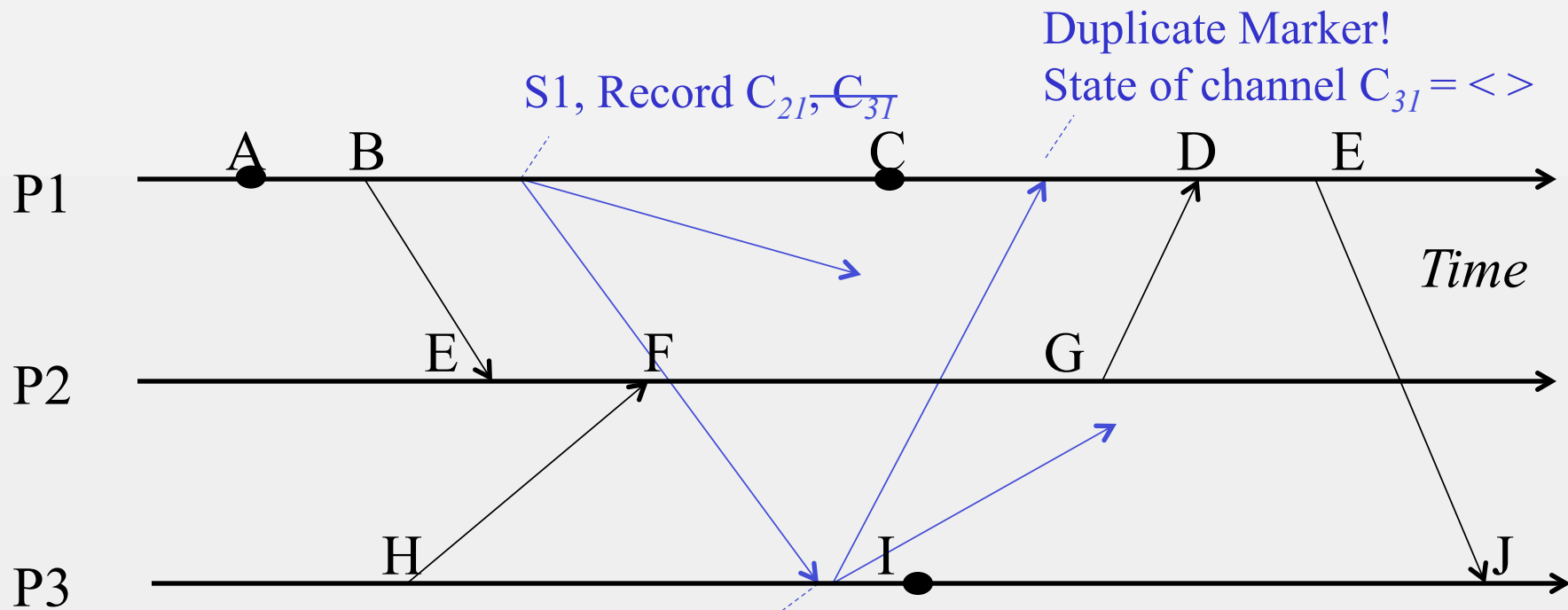


- First Marker!
- Record own state as S3
- Mark  $C_{13}$  state as empty
- Turn on recording on other incoming  $C_{23}$
- Send out Markers

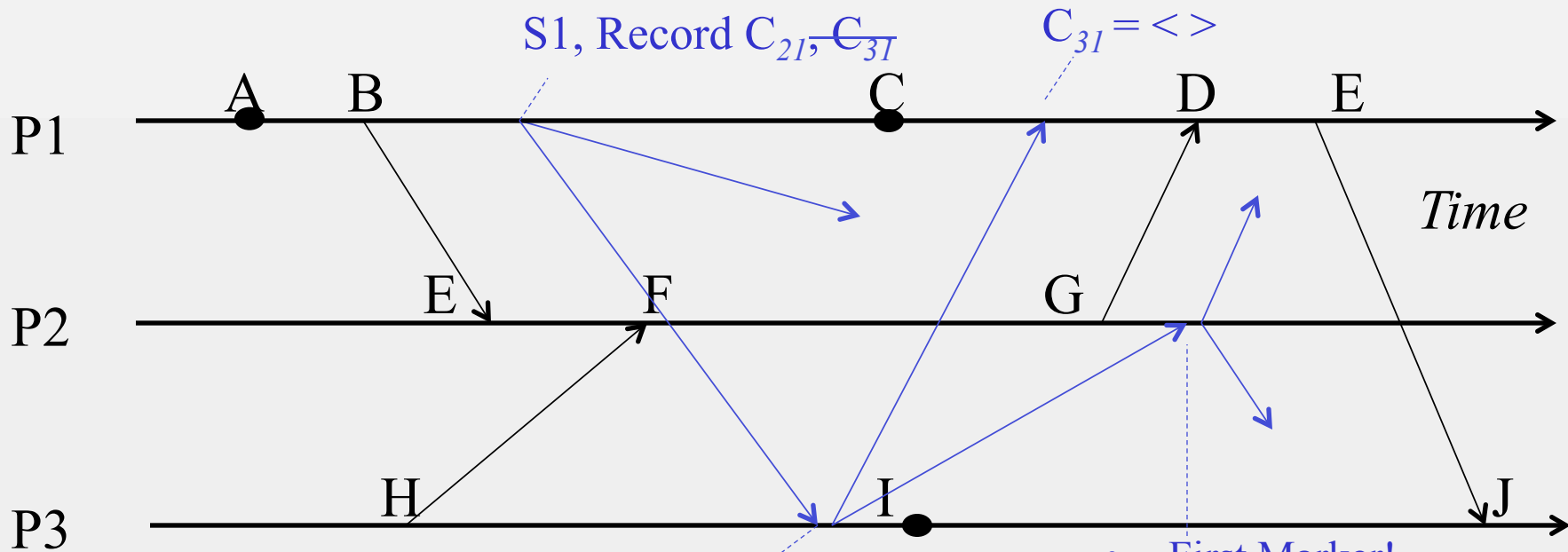




- S3
- $C_{13} = \langle \rangle$
- Record  $C_{23}$

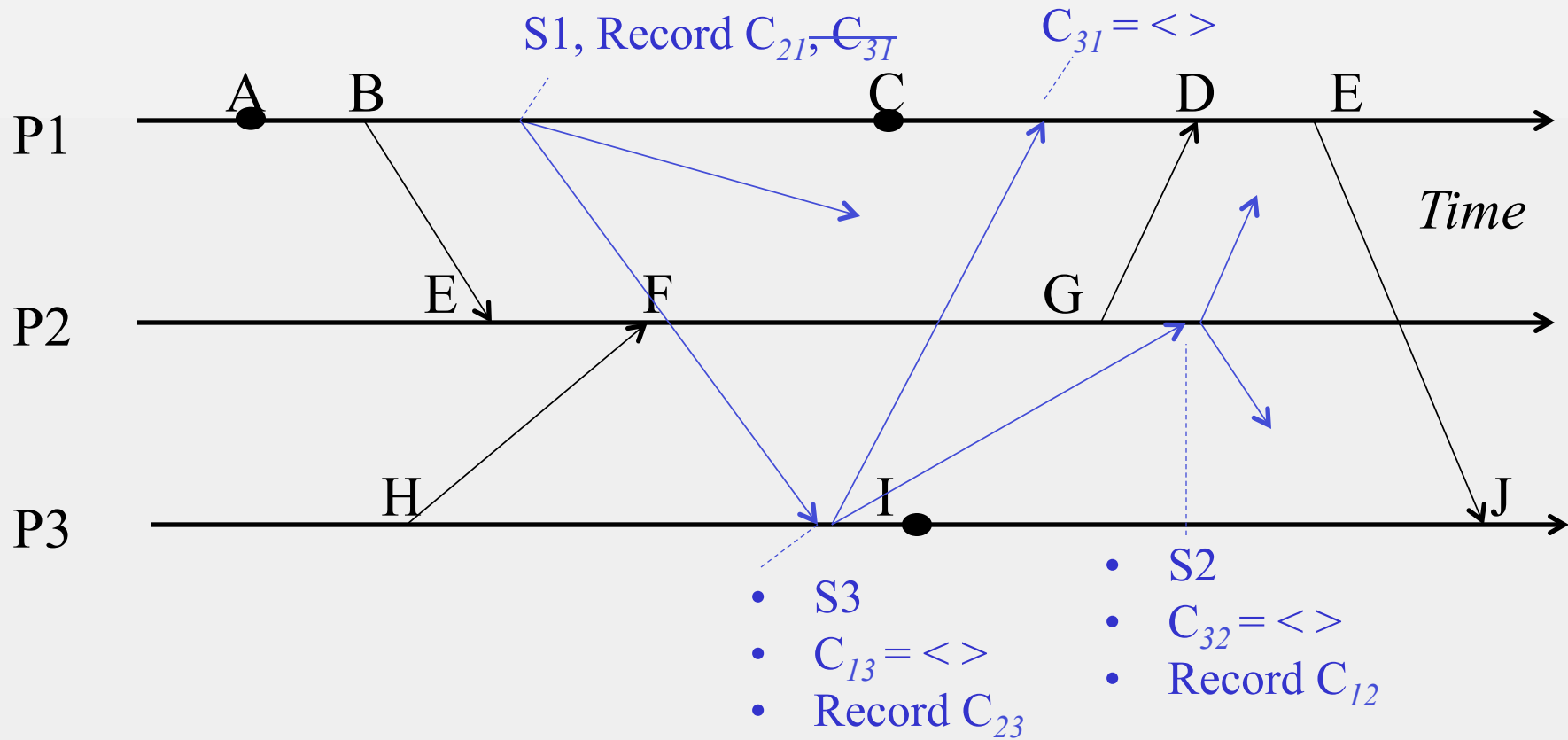


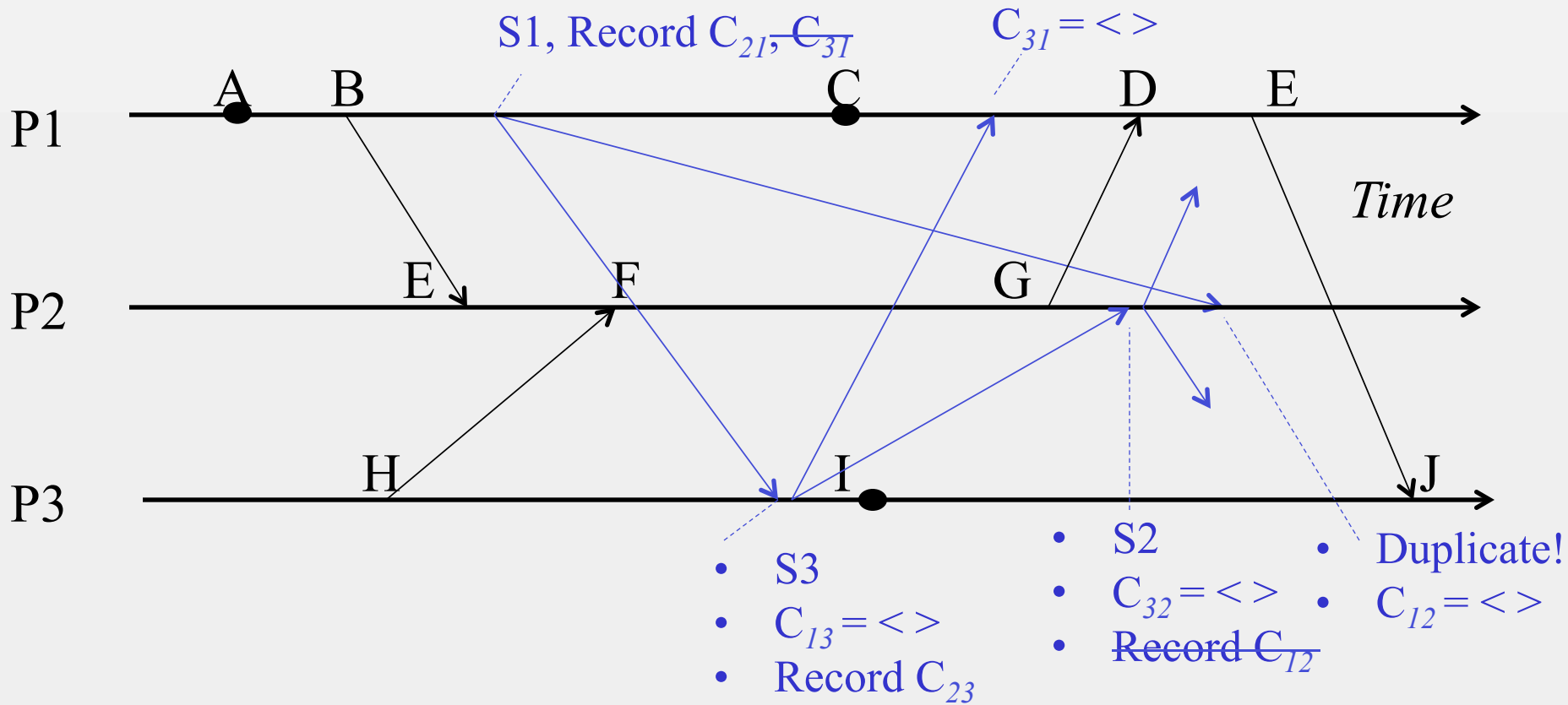
- S3
- $C_{13} = \langle \rangle$
- Record  $C_{23}$

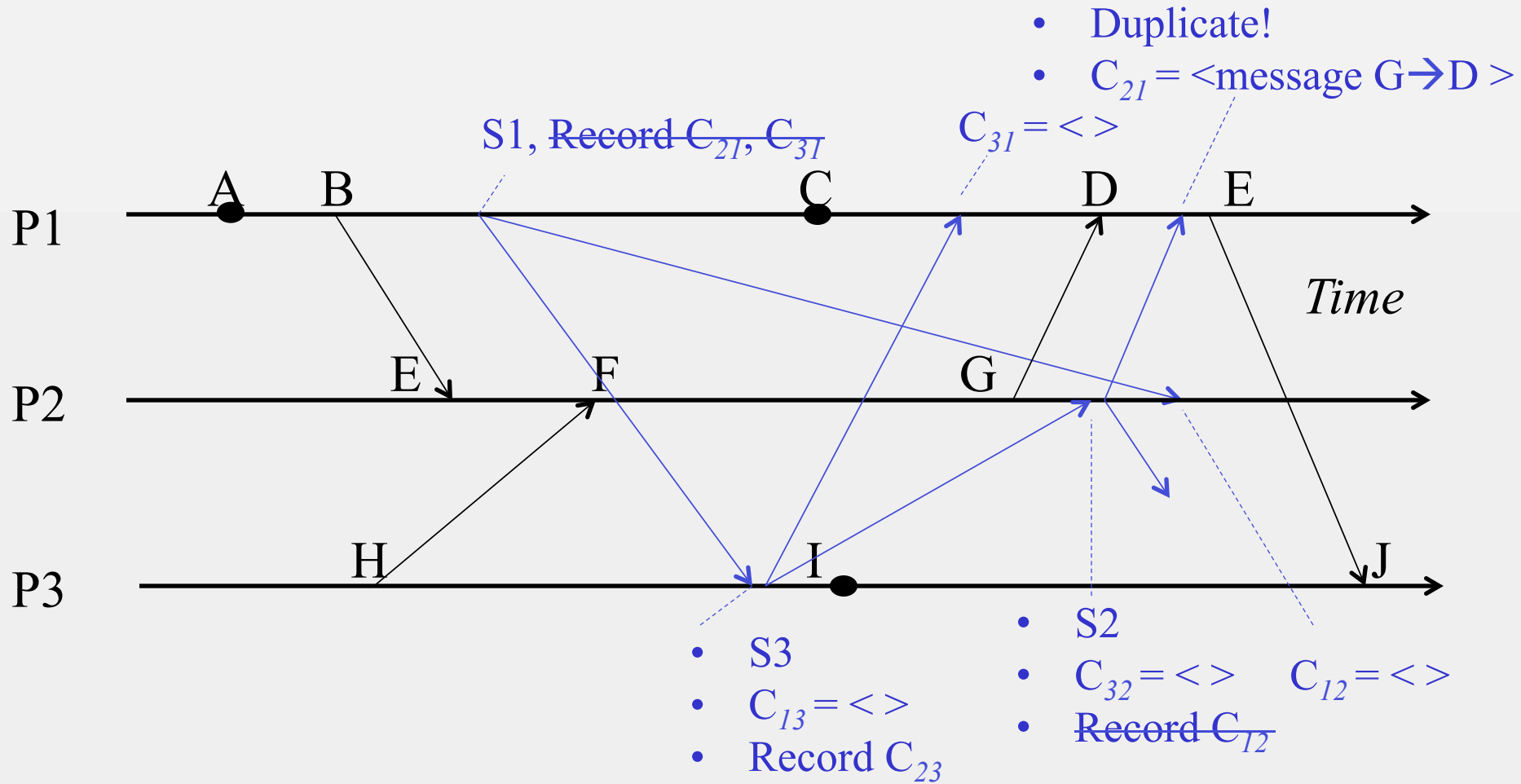


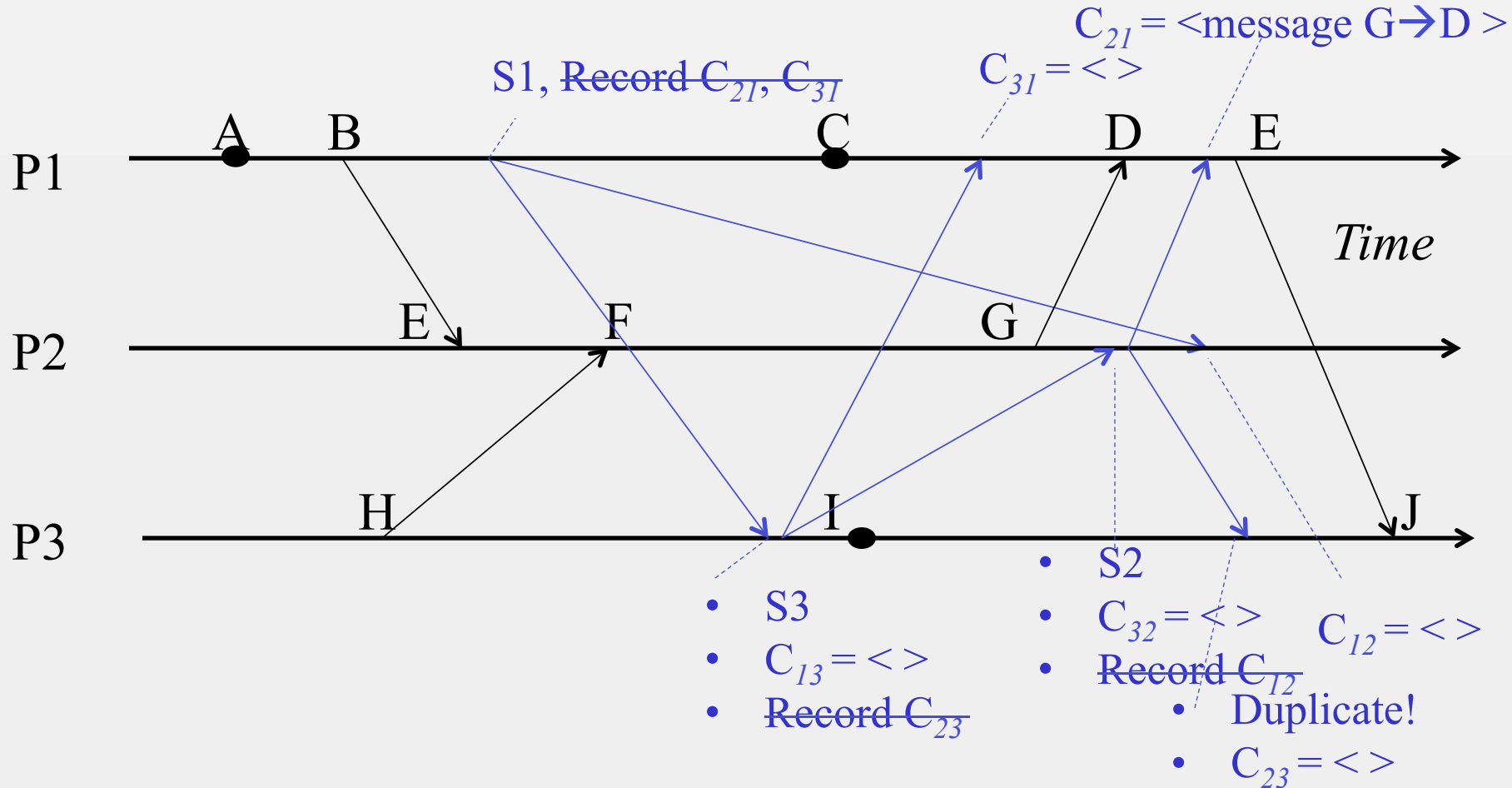
- S3
- $C_{13} = \langle \rangle$
- Record  $C_{23}$

- First Marker!
- Record own state as S2
- Mark  $C_{32}$  state as empty
- Turn on recording on  $C_{12}$
- Send out Markers

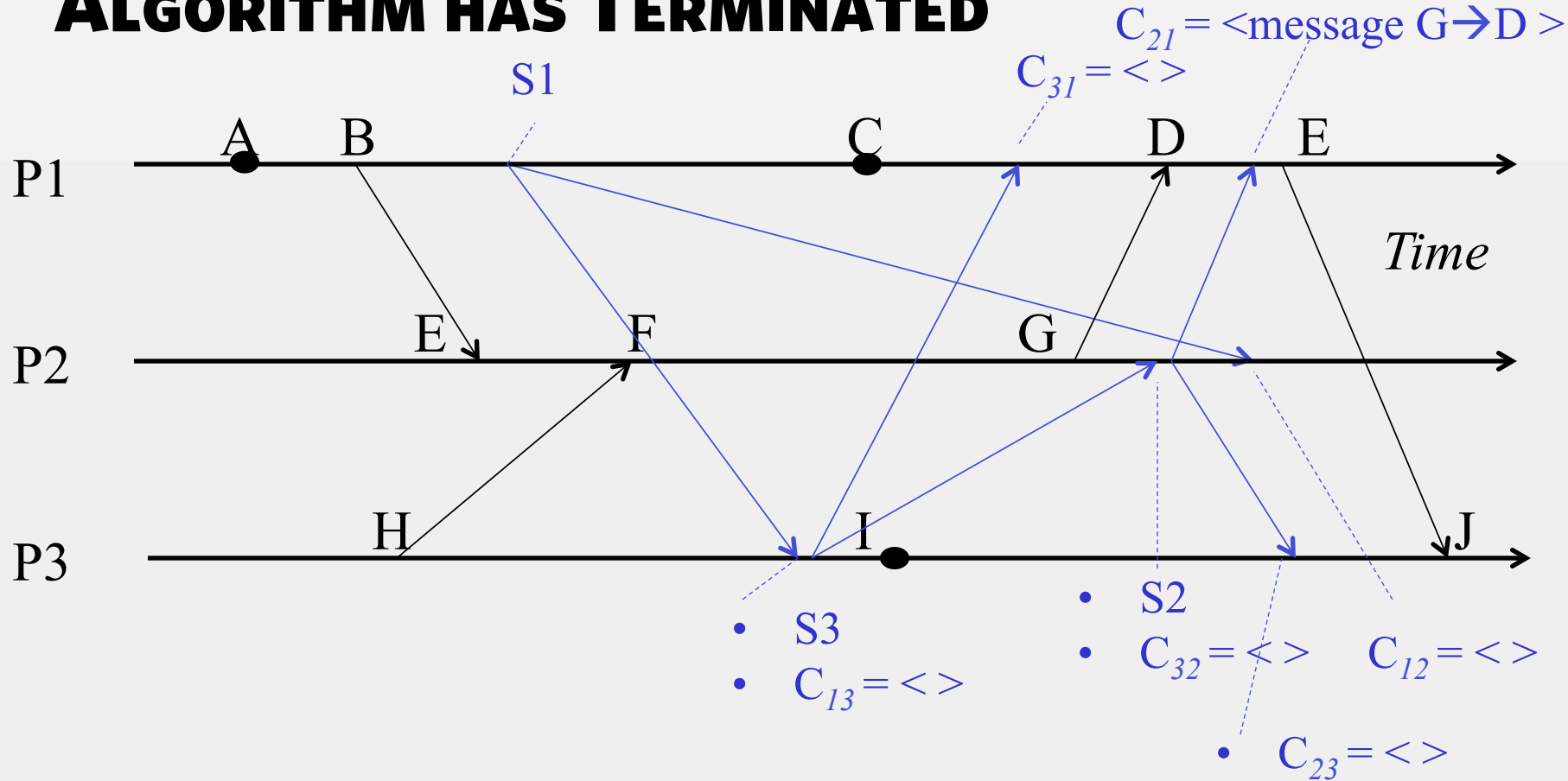






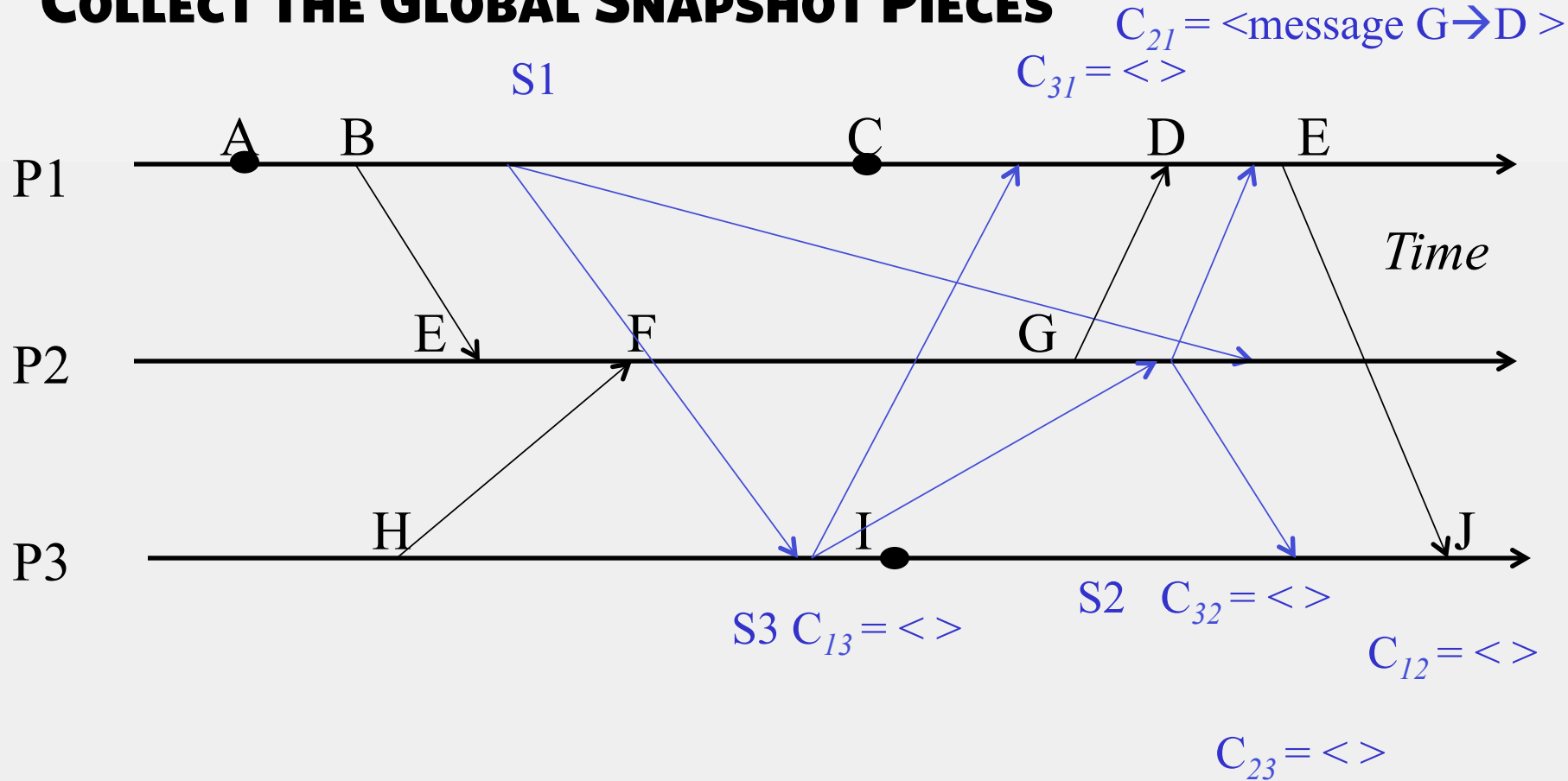


# ALGORITHM HAS TERMINATED





# COLLECT THE GLOBAL SNAPSHOT PIECES



# NEXT

- **Global Snapshot calculated by Chandy-Lamport algorithm is causally correct**
  - What?

CLOUD  
COMPUTING  
CONCEPTS

Snapshots

LECTURE C

CONSISTENT CUTS

Indranil Gupta (Indy)  
University of Illinois

# CUTS

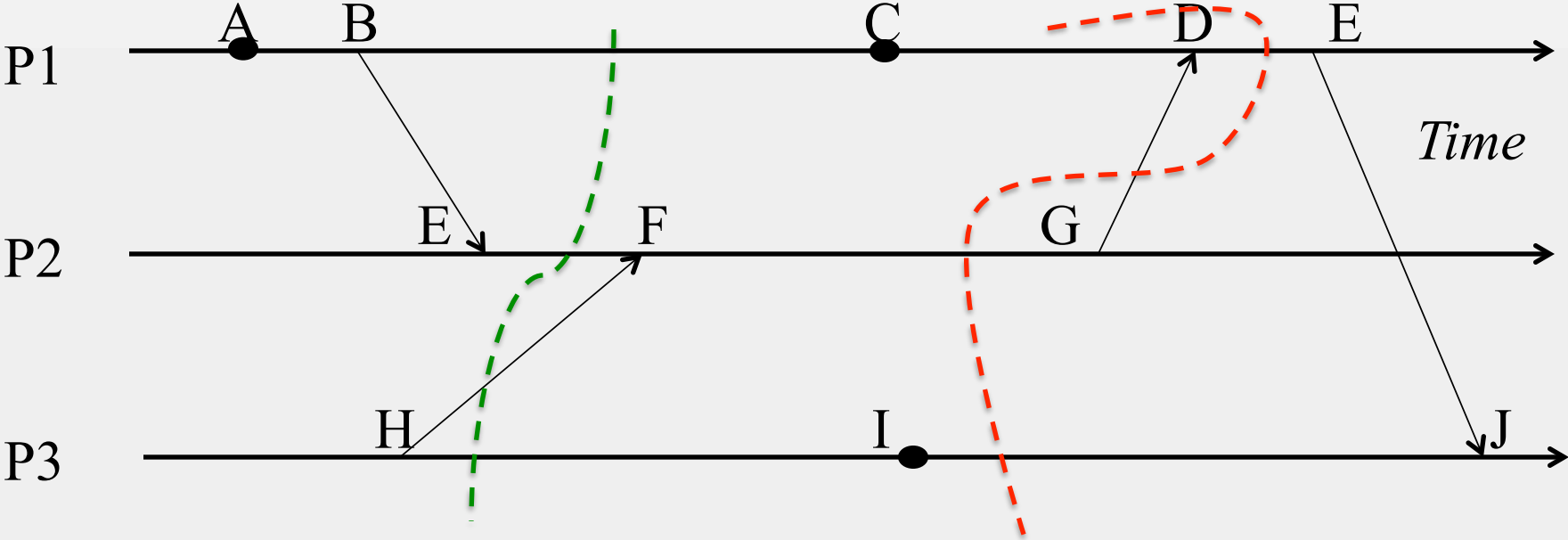
- **Cut** = time frontier at each process and at each channel
- **Events at the process/channel that happen before the cut are “in the cut”**
  - And happening after the cut are “out of the cut”

# CONSISTENT CUTS

**Consistent Cut:** a cut that obeys causality

- A cut  $C$  is a consistent cut if and only if:
  - for (each pair of events  $e, f$  in the system)
    - Such that event  $e$  is in the cut  $C$ , and if  $f \rightarrow e$  ( $f$  happens-before  $e$ )
      - Then: Event  $f$  is also in the cut  $C$

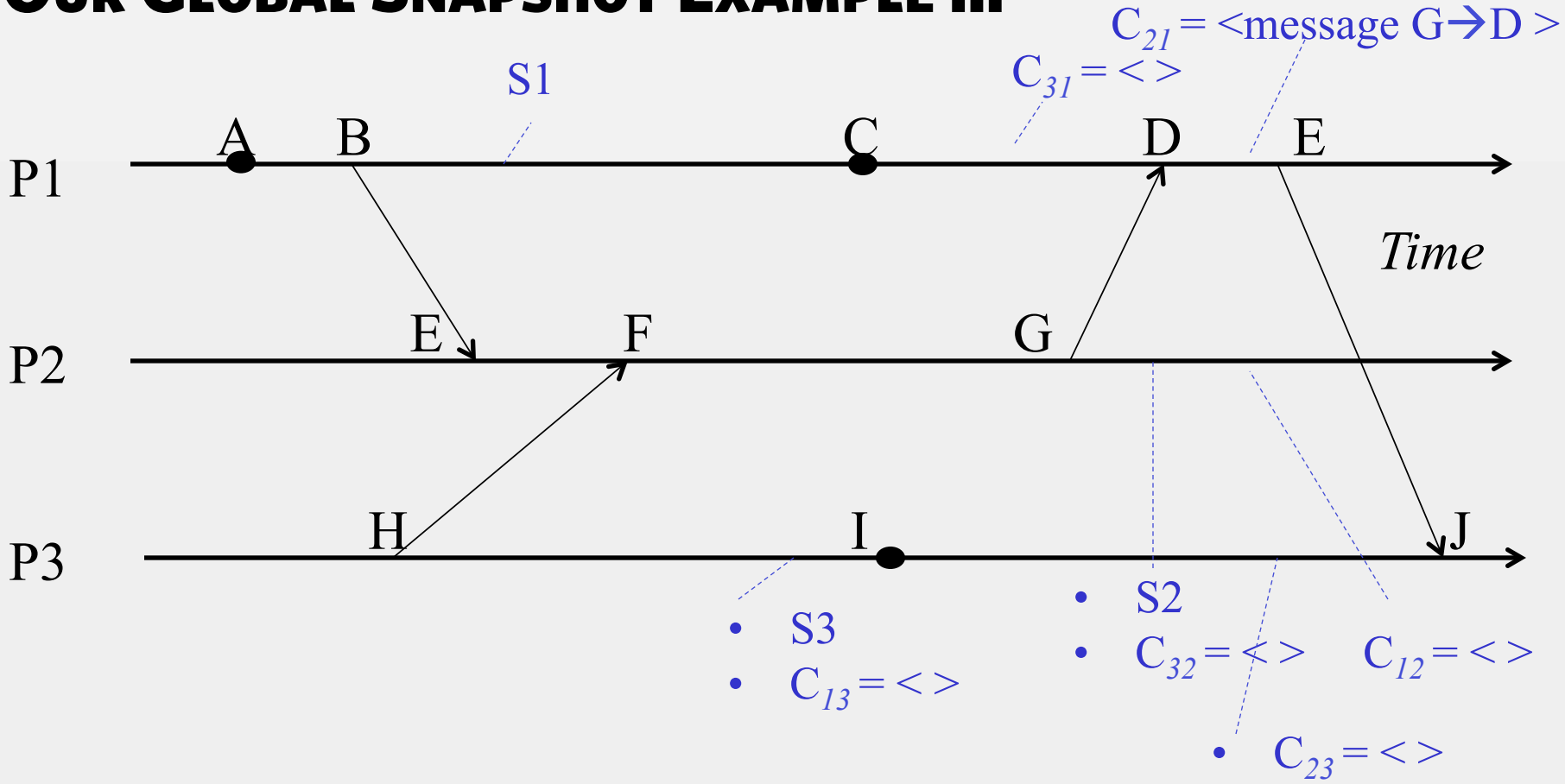
# EXAMPLE



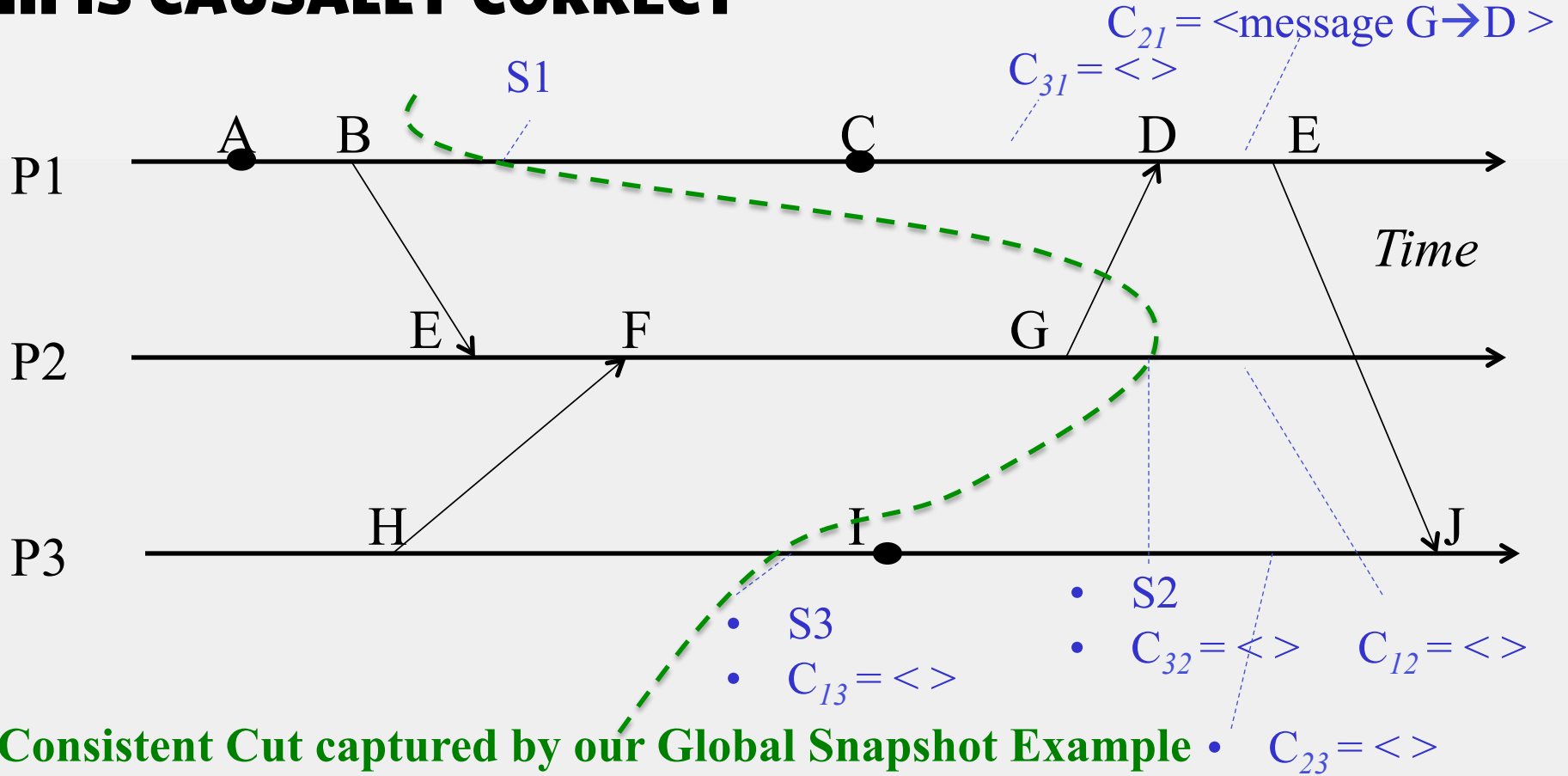
Consistent Cut

Inconsistent Cut  
 $G \rightarrow D$ , but only D is in cut

# OUR GLOBAL SNAPSHOT EXAMPLE ...



# ... IS CAUSALLY CORRECT





# IN FACT...

- Any run of the Chandy-Lamport Global Snapshot algorithm creates a consistent cut

# CHANDY-LAMPORT GLOBAL SNAPSHOT ALGORITHM CREATES A CONSISTENT CUT

Let's quickly look at the proof

- Let  $e_i$  and  $e_j$  be events occurring at  $P_i$  and  $P_j$ , respectively such that

- $e_i \rightarrow e_j$  ( $e_i$  happens before  $e_j$ )

- The snapshot algorithm ensures that

- if  $e_j$  is in the cut then  $e_i$  is also in the cut.

- That is: if  $e_j \rightarrow \langle P_j \text{ records its state} \rangle$ , then

- it must be true that  $e_i \rightarrow \langle P_i \text{ records its state} \rangle$ .

# CHANDY-LAMPORT GLOBAL SNAPSHOT ALGORITHM CREATES A CONSISTENT CUT

- if  $e_j \rightarrow \langle P_j \text{ records its state} \rangle$ , then it must be true that  $e_i \rightarrow \langle P_i \text{ records its state} \rangle$ .
  - By contradiction, suppose  $e_j \rightarrow \langle P_j \text{ records its state} \rangle$  and  $\langle P_i \text{ records its state} \rangle \rightarrow e_i$
  - Consider the path of app messages (through other processes) that go from  $e_i \rightarrow e_j$
  - Due to FIFO ordering, markers on each link in above path will precede regular app messages
  - Thus, since  $\langle P_i \text{ records its state} \rangle \rightarrow e_i$ , it must be true that  $P_j$  received a marker before  $e_j$
  - Thus  $e_j$  is not in the cut  $\Rightarrow$  contradiction

# NEXT

- What is the Chandy-Lamport algorithm used for?

CLOUD  
COMPUTING  
CONCEPTS

Snapshots

LECTURE D

SAFETY AND LIVENESS

Indranil Gupta (Indy)  
University of Illinois

# "CORRECTNESS" IN DISTRIBUTED SYSTEMS

- Can be seen in two ways
- Liveness and Safety
- Often confused – it's important to distinguish from each other

# LIVENESS

- **Liveness** = guarantee that something **good** will happen, **eventually**
  - Eventually == does not imply a time bound, but if you let the system run long enough, then ...

# LIVENESS: EXAMPLES

- **Liveness** = guarantee that something **good** will happen, **eventually**
  - Eventually == does not imply a time bound, but if you let the system run long enough, then ...
- **Examples in Real World**
  - Guarantee that “at least one of the athletes in the 100m final will win gold” is liveness
  - A criminal will eventually be jailed
- **Examples in a Distributed System**
  - Distributed computation: Guarantee that it will terminate
  - “Completeness” in failure detectors: every failure is eventually detected by some non-faulty process
  - In Consensus: All processes eventually decide on a value



# SAFETY

- Safety = guarantee that something bad will never happen

# SAFETY: EXAMPLES

- **Safety** = guarantee that something **bad** will **never** happen
- **Examples in Real World**
  - A peace treaty between two nations provides safety
    - War will never happen
  - An innocent person will never be jailed
- **Examples in a Distributed System**
  - There is no deadlock in a distributed transaction system
  - No object is orphaned in a distributed object system
  - “Accuracy” in failure detectors
  - In Consensus: No two processes decide on different values

# CAN'T WE GUARANTEE BOTH?

- **Can be difficult to satisfy both liveness and safety in an asynchronous distributed system!**
  - Failure Detector: Completeness (Liveness) and Accuracy (Safety) cannot both be guaranteed by a failure detector in an asynchronous distributed system
  - Consensus: Decisions (Liveness) and correct decisions (Safety) cannot both be guaranteed by any consensus protocol in an asynchronous distributed system
  - Very difficult for legal systems (anywhere in the world) to guarantee that all criminals are jailed (Liveness) and no innocents are jailed (Safety)

# IN THE LANGUAGE OF GLOBAL STATES

- **Recall that a distributed system moves from one global state to another global state, via causal steps**
- **Liveness w.r.t. a property  $Pr$  in a given state  $S$  means**
  - $S$  satisfies  $Pr$ , or there is **some** causal path of global states from  $S$  to  $S'$  where  $S'$  satisfies  $Pr$
- **Safety w.r.t. a property  $Pr$  in a given state  $S$  means**
  - $S$  satisfies  $Pr$ , and **all** global states  $S'$  reachable from  $S$  also satisfy  $Pr$

# USING GLOBAL SNAPSHOT ALGORITHM

- **Chandy-Lamport algorithm can be used to detect global properties that are **stable****
  - Stable = once true, stays true forever afterwards
- **Stable Liveness examples**
  - Computation has terminated
- **Stable Non-Safety examples**
  - There is a deadlock
  - An object is orphaned (no pointers point to it)
- **All stable global properties can be detected using the Chandy-Lamport algorithm**
  - **Due to its causal correctness**

# 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 stable global properties
- Safety vs. Liveness