CLOUD COMPUTING CONCEPTS

with Indranil Gupta (Indy)

TIME AND ORDERING

Lecture D

LAMPORT TIMESTAMPS

Ordering Events in a Distributed System

- **To order events across processes, trying to sync clocks is one approach.**
- **What if we instead assigned timestamps to events that were not *absolute* time?**
- **As long as these timestamps obey *causality*, that would work.**
  - If an event A causally happens before another event B, then timestamp(A) < timestamp(B).
  - Humans use causality all the time.
    - E.g., I enter a house only after I unlock it.
    - E.g., you receive a letter only after I send it.

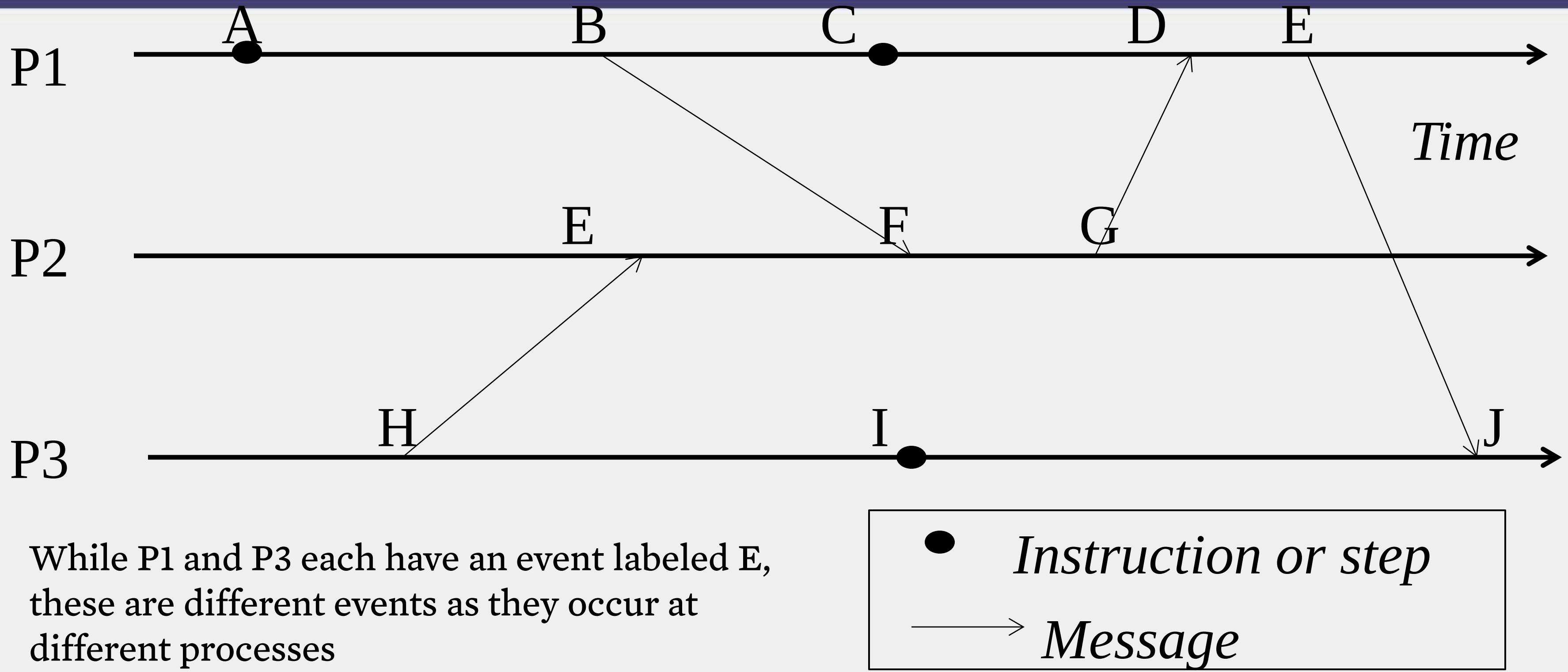
Logical (or Lamport) Ordering

- Proposed by Leslie Lamport in the 1970s
- Used in almost all distributed systems since then
- Almost all cloud computing systems use some form of logical ordering of events

Logical (or Lamport) Ordering(2)

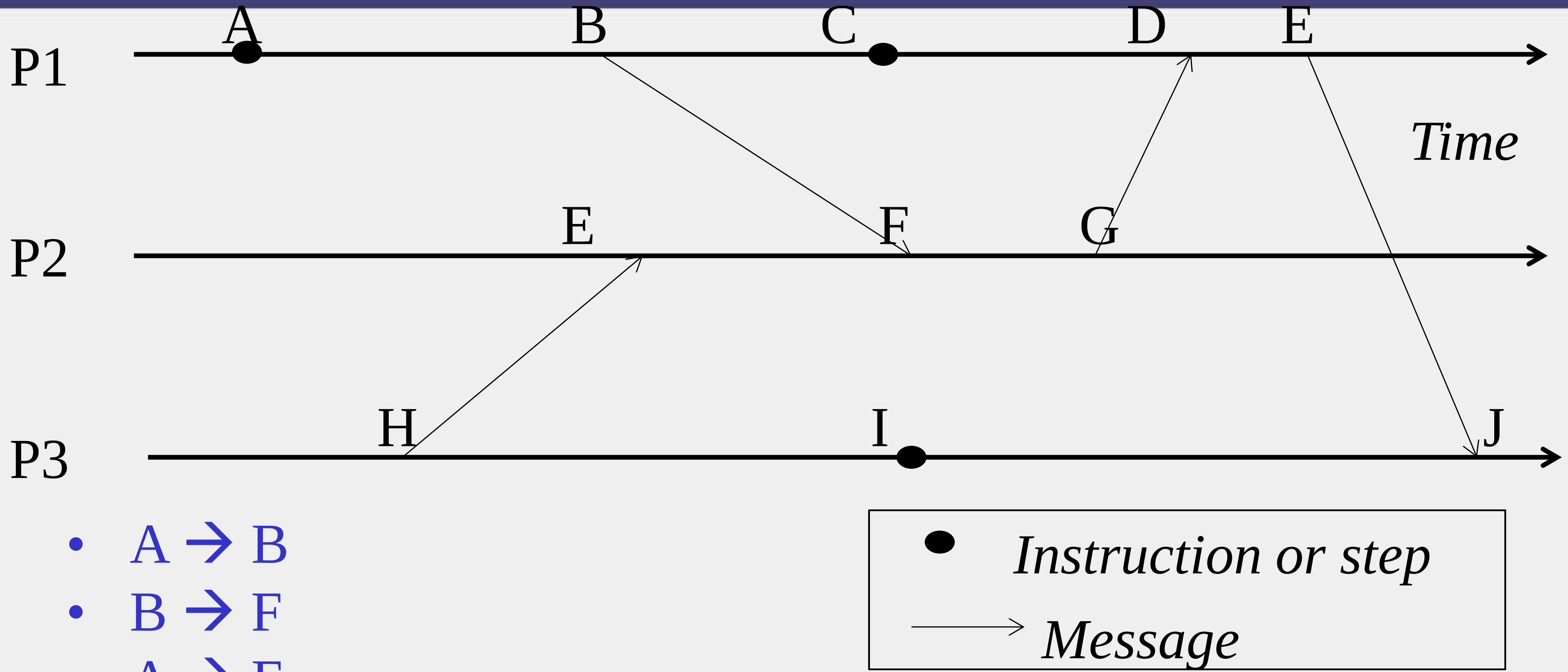
- Define a logical relation *Happens-Before* among pairs of events
- Happens-Before denoted as $\rightarrow$
- Three rules
  1. On the same process: $a \rightarrow b$, if $time(a) < time(b)$ (using the local clock)
  2. If p1 sends *m* to p2: $send(m) \rightarrow receive(m)$
  3. (Transitivity) If $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$
- Creates a *partial order* among events
  - Not all events related to each other via $\rightarrow$

Example



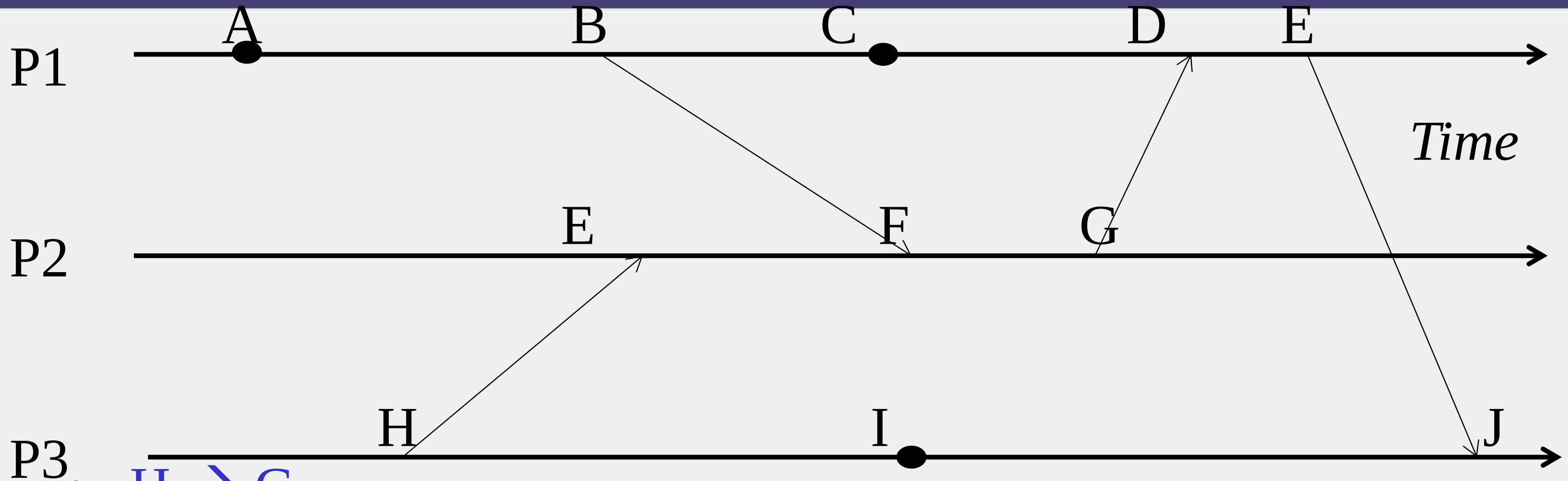
While P1 and P3 each have an event labeled E, these are different events as they occur at different processes

Happens-Before

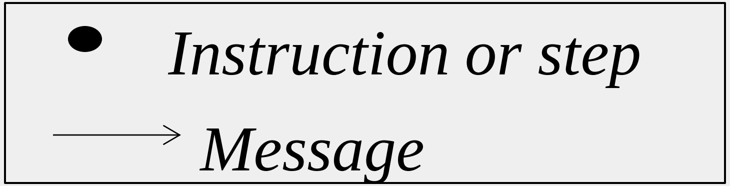


- A → B
- B → F
- A → F

Happens-Before (2)



- H → G
- F → J
- H → J
- C → J

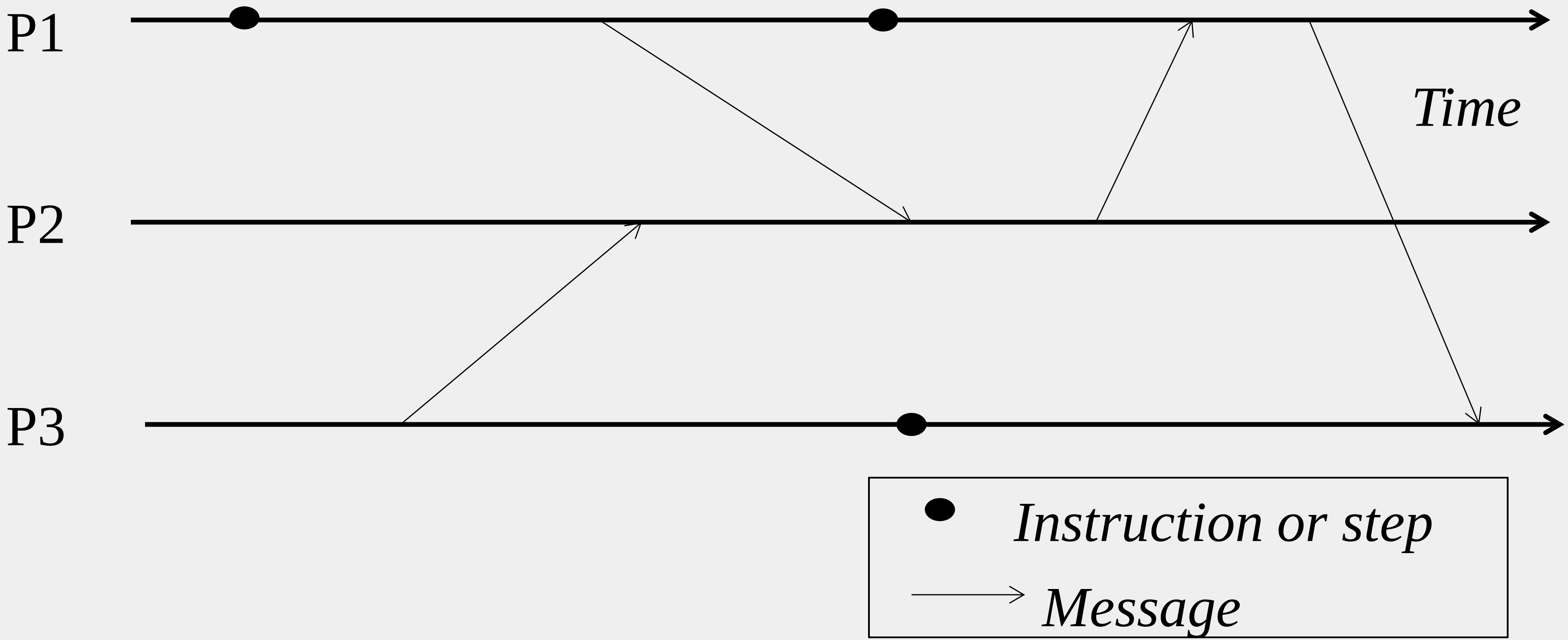


In practice: Lamport timestamps

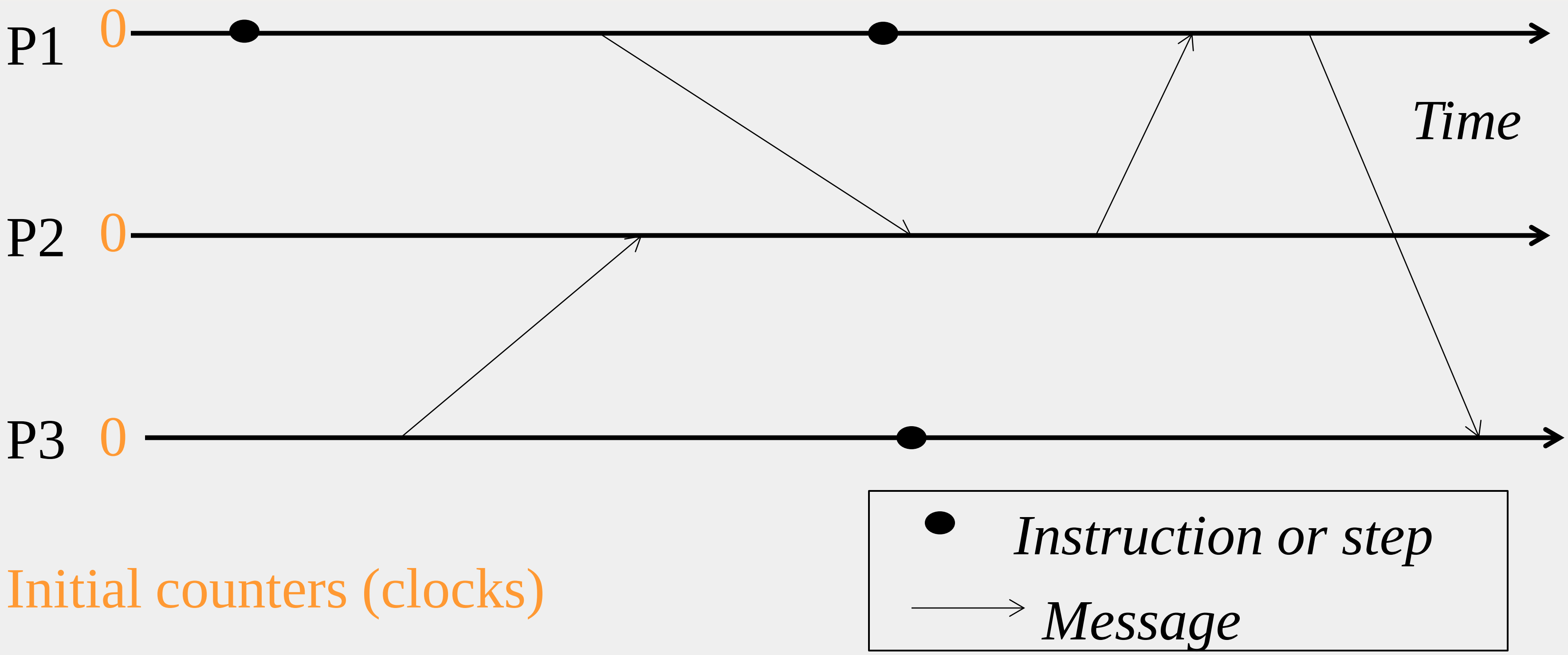
- **Goal: Assign logical (Lamport) timestamp to each event**
- **Timestamps obey causality**
- **Rules**
  - Each process uses a local counter (clock) which is an integer
    - Initial value of counter is zero
  - A process increments its counter when a send or an instruction happens at it. The counter is assigned to the event as its timestamp.
  - A send (message) event carries its timestamp
  - For a receive (message) event the counter is updated by

    max(local clock, message timestamp) + 1

Example

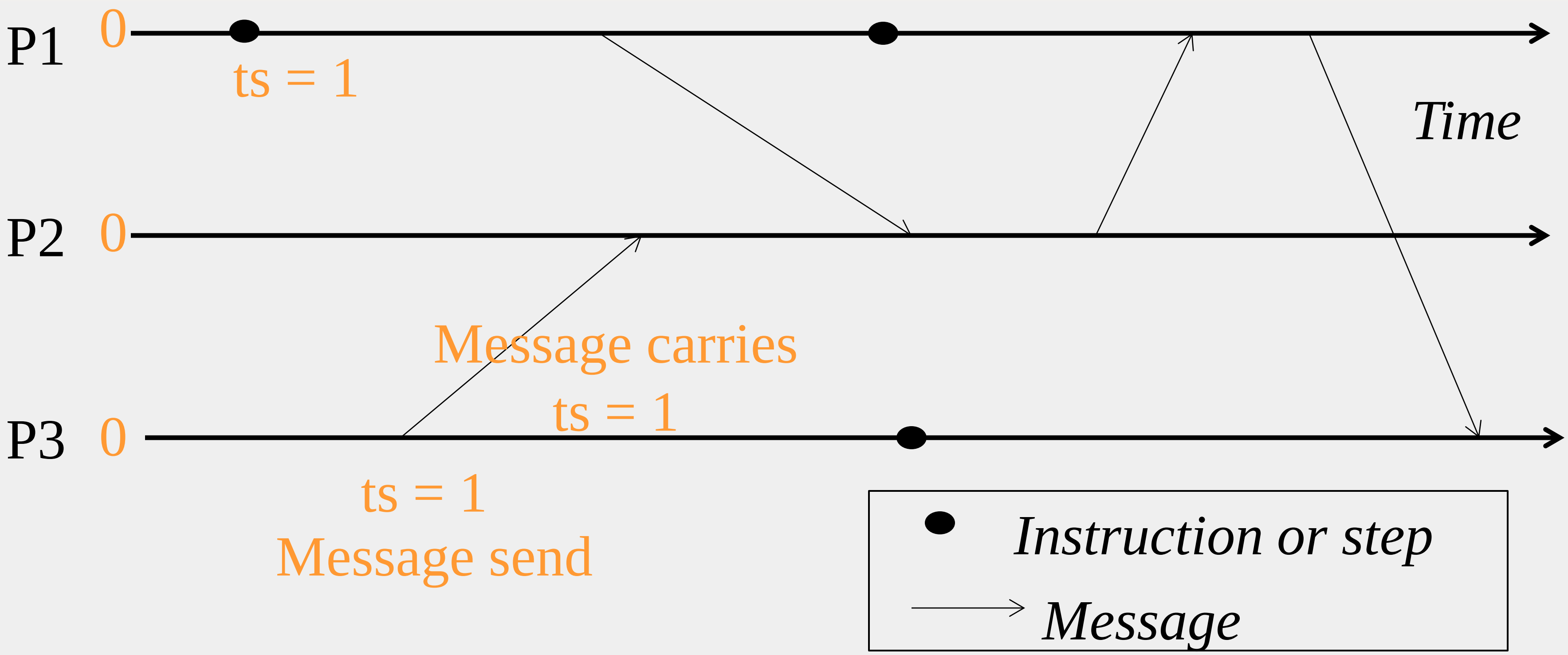


Lamport Timestamps

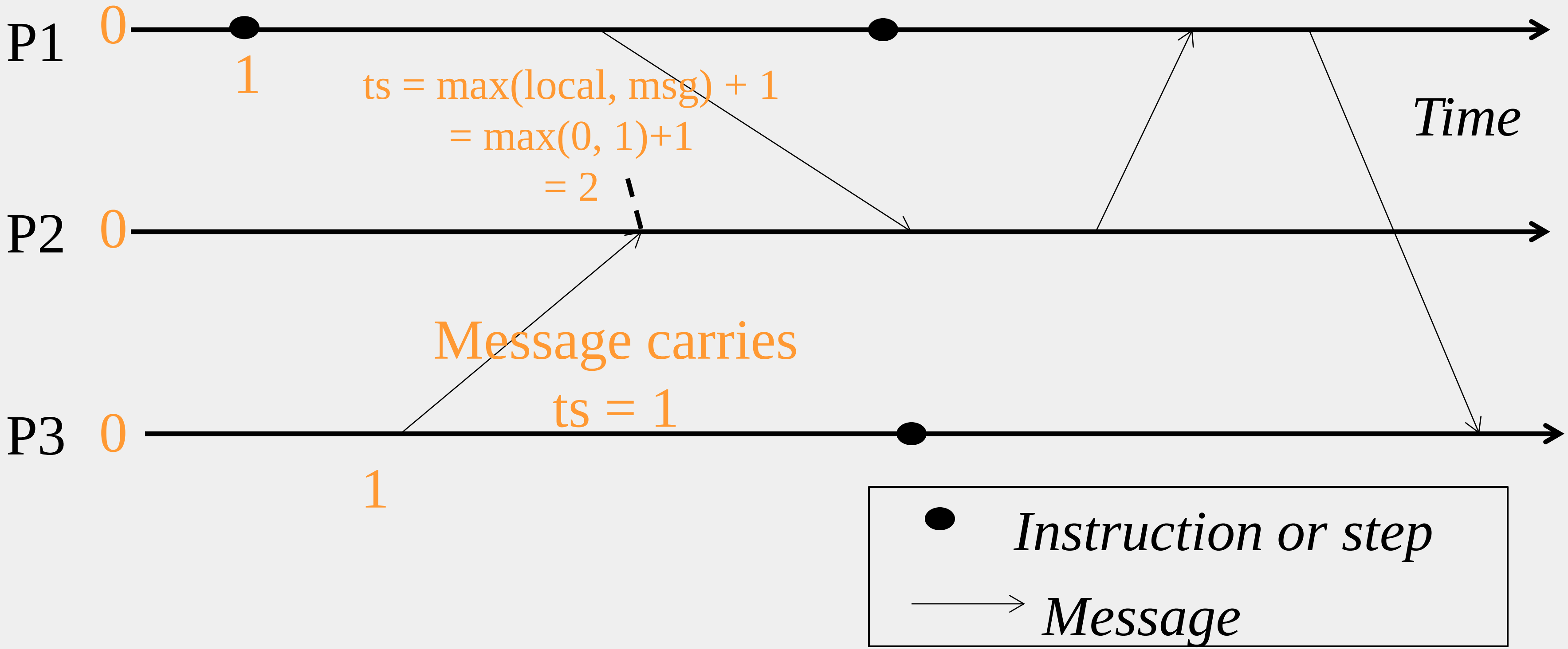


Initial counters (clocks)

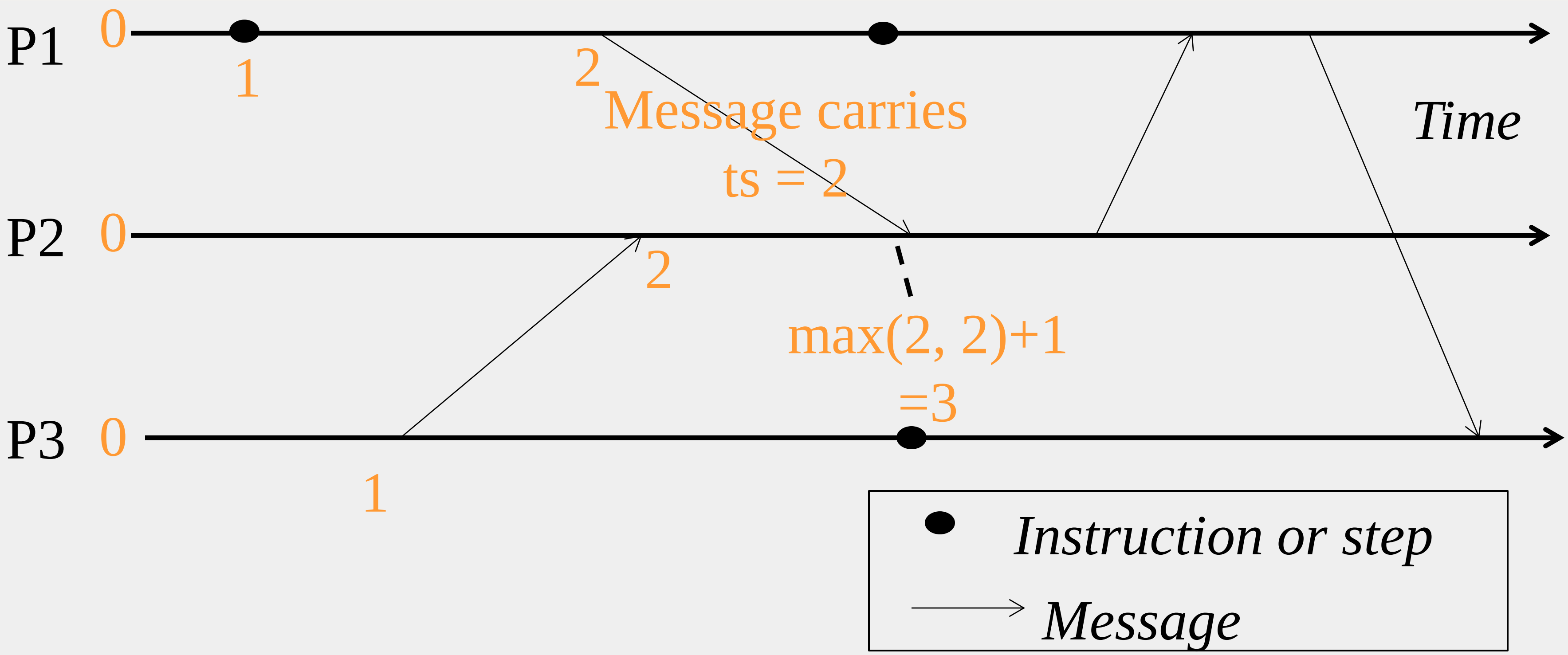
Lamport Timestamps



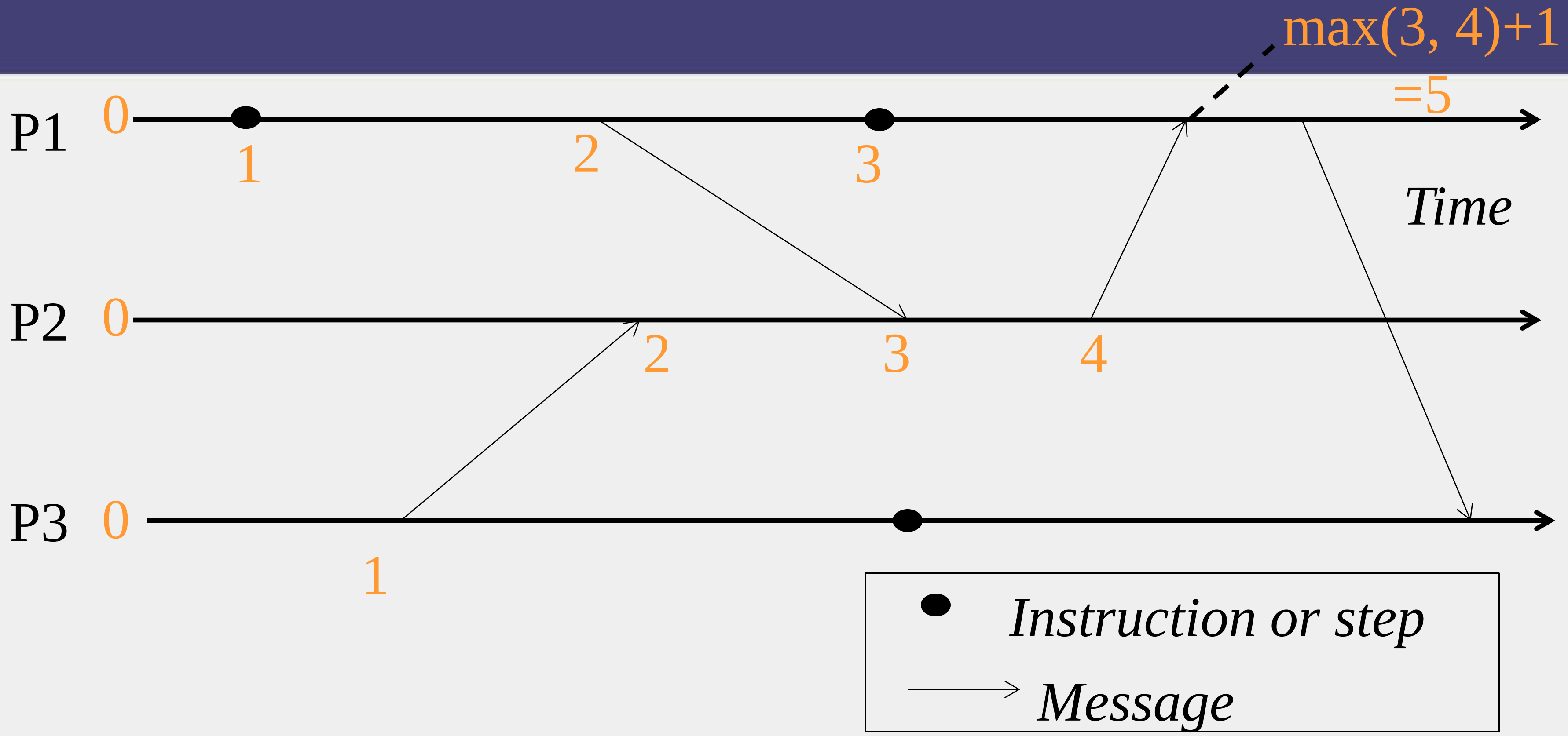
Lamport Timestamps



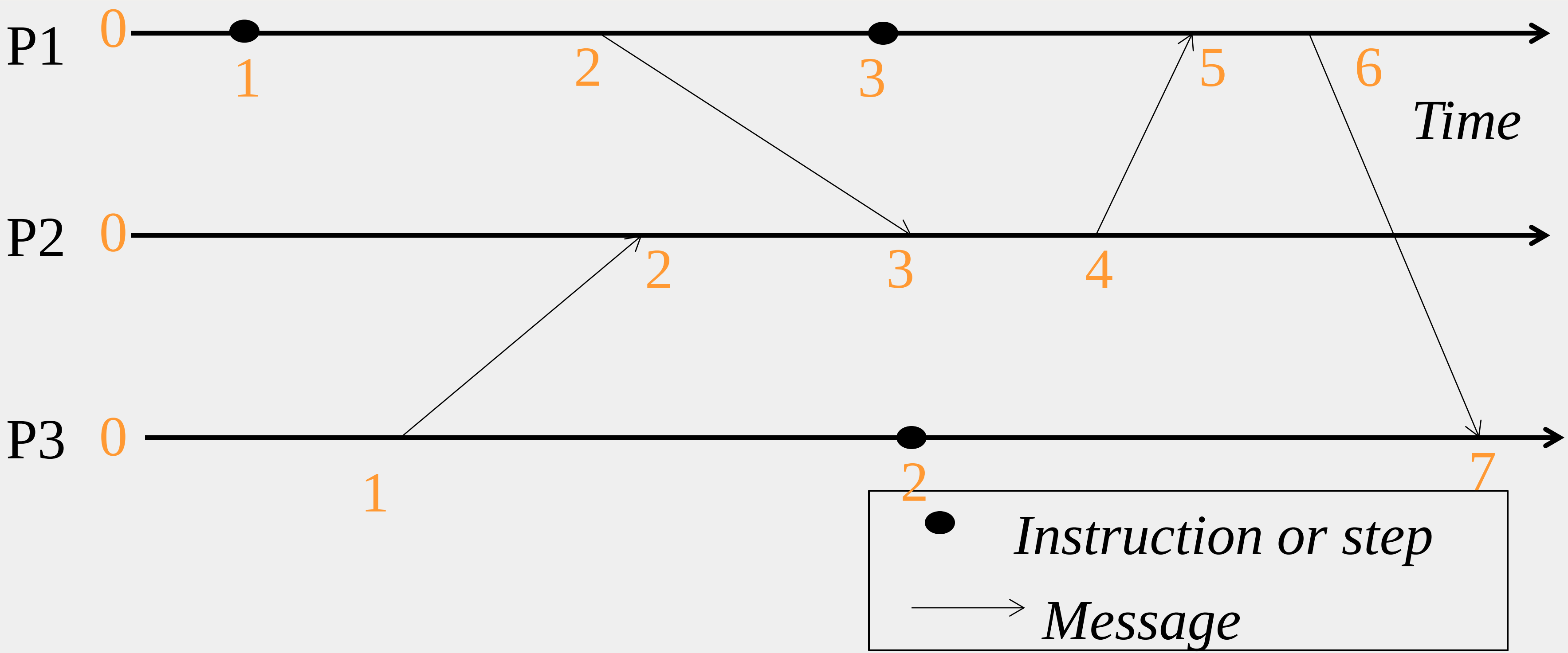
Lamport Timestamps



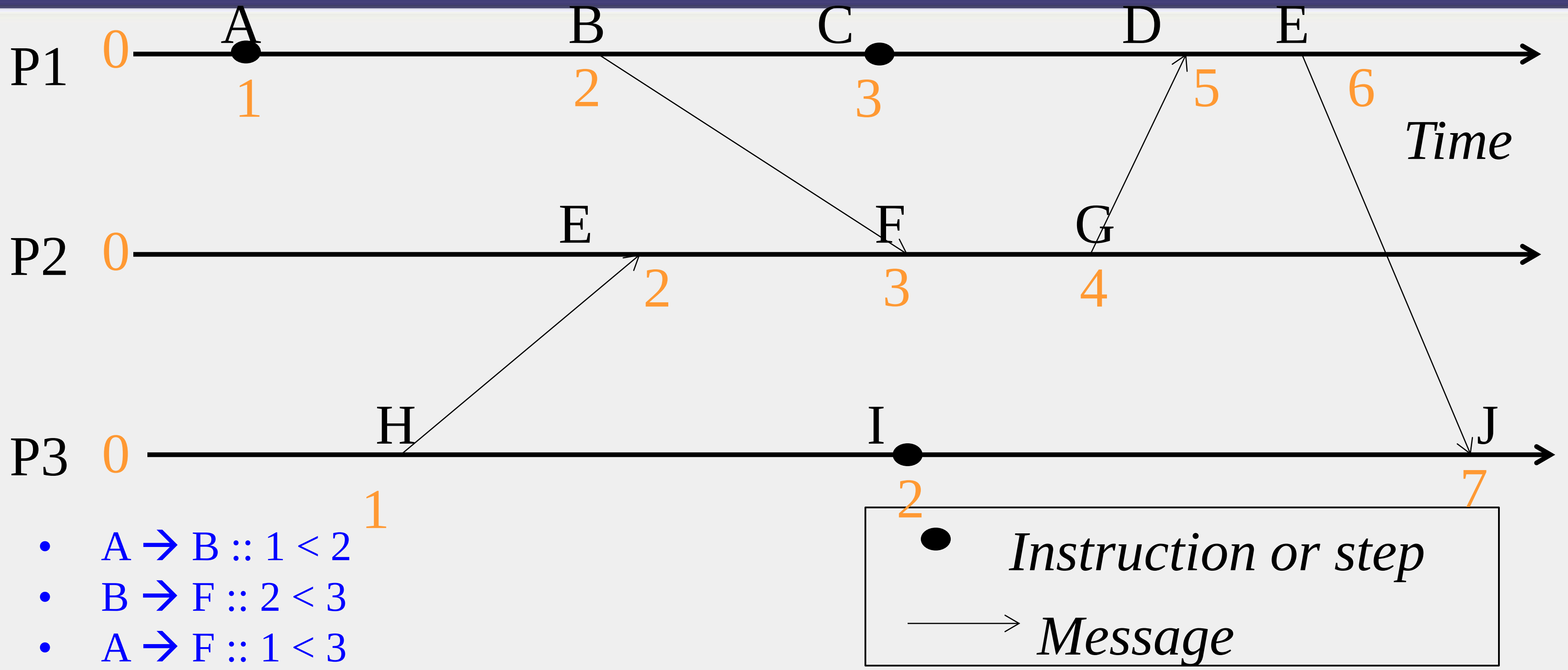
Lamport Timestamps



Lamport Timestamps

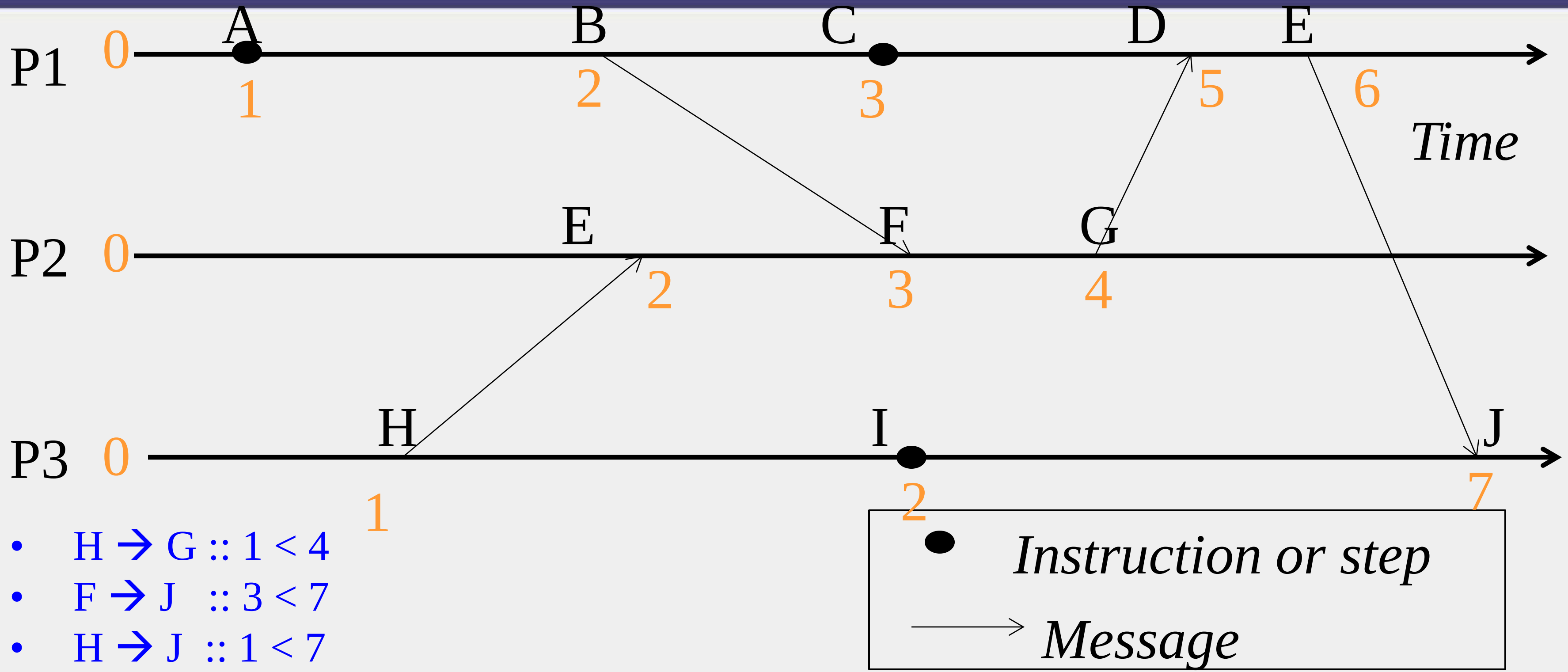


Obeying Causality



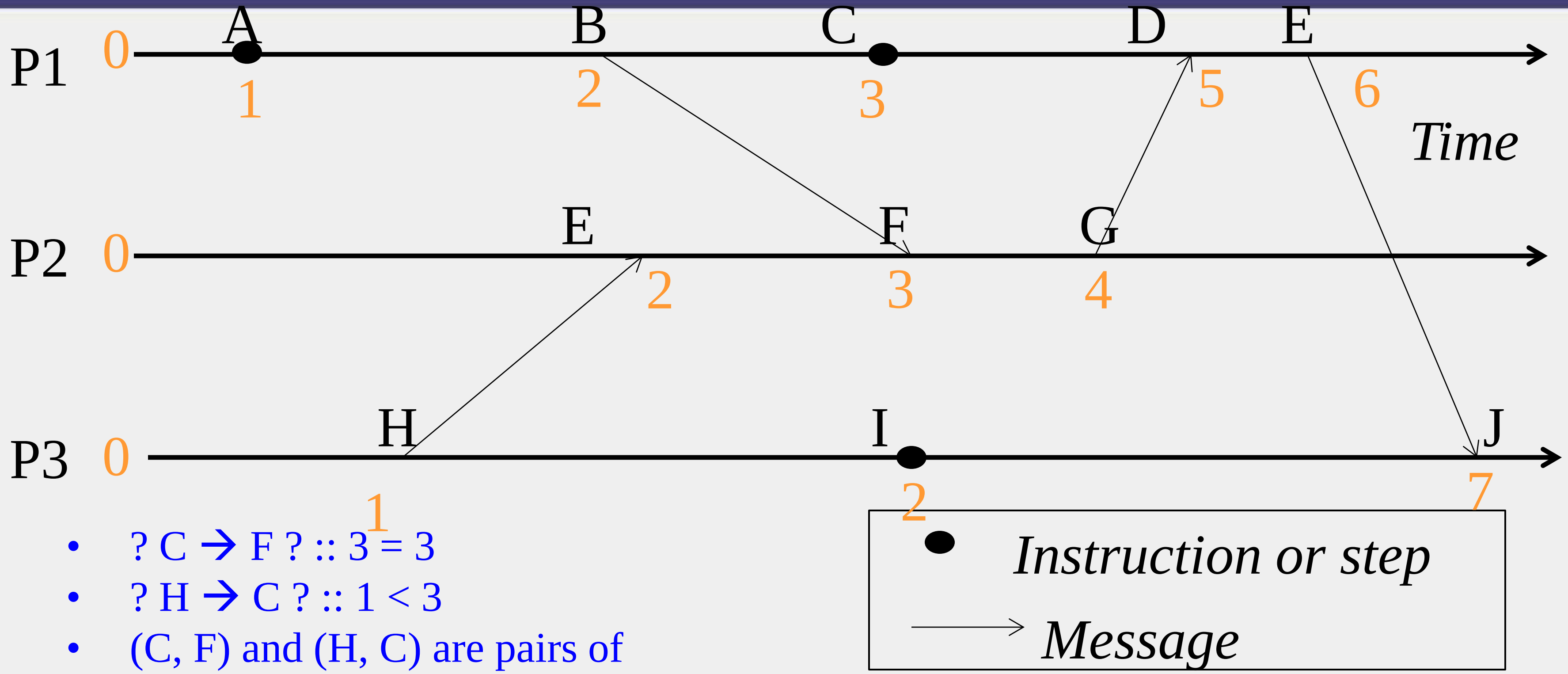
- A → B :: 1 < 2
- B → F :: 2 < 3
- A → F :: 1 < 3

Obeying Causality (2)



- H → G :: 1 < 4
- F → J :: 3 < 7
- H → J :: 1 < 7
- C → J :: 3 < 7

NOT ALWAYS *IMPLYING* CAUSALITY



- ? C → F ? :: 3 = 3
- ? H → C ? :: 1 < 3
- (C, F) and (H, C) are pairs of *concurrent* events

Concurrent Events

- **A pair of concurrent events doesn’t have a causal path from one event to another (either way, in the pair)**
- **Lamport timestamps not guaranteed to be ordered or unequal for concurrent events**
- **Ok, since concurrent events are not causality related!**
- **Remember**

$E1 \rightarrow E2 \Rightarrow$ timestamp(E1) < timestamp (E2), BUT

timestamp(E1) < timestamp (E2) $\Rightarrow$

{E1 → E2} OR {E1 and E2 concurrent}

Next

- Can we have causal or logical timestamps from which we can tell if two events are concurrent or causally related?