Computer Science 425
Distributed Systems

Lecture 3

Logical Clock and Global States/ Snapshots

Reading: Chapter 11.4&11.5

Klara Nahrstedt

Acknowledgement

- The slides during this semester are based on ideas and material from the following sources:
  - Slides prepared by Professors M. Harandi, J. Hou, I. Gupta, N. Vaidya, Y-Ch. Hu, S. Mitra.
  - Slides from Professor S. Gosh's course at University o Iowa.

Administrative

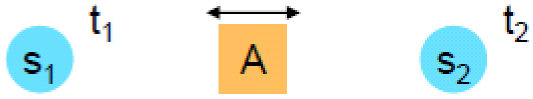
- **Form Groups for MP projects**
  - **Up to 3 members per group**
- **Group Formation Deadline: September 1**
  - **Email names to TA today**
- **Coming Next Homework 1 (Thursday)**
- **Introductory material to get introduced to the Eclipse/Android Programming Environment is posted - (Optional MP0)**
- **Ying Huang (TA) – changed office hours**
  - **Monday 2-3pm**
  - **Thursday 3:15-4:15pm**

Plan for today

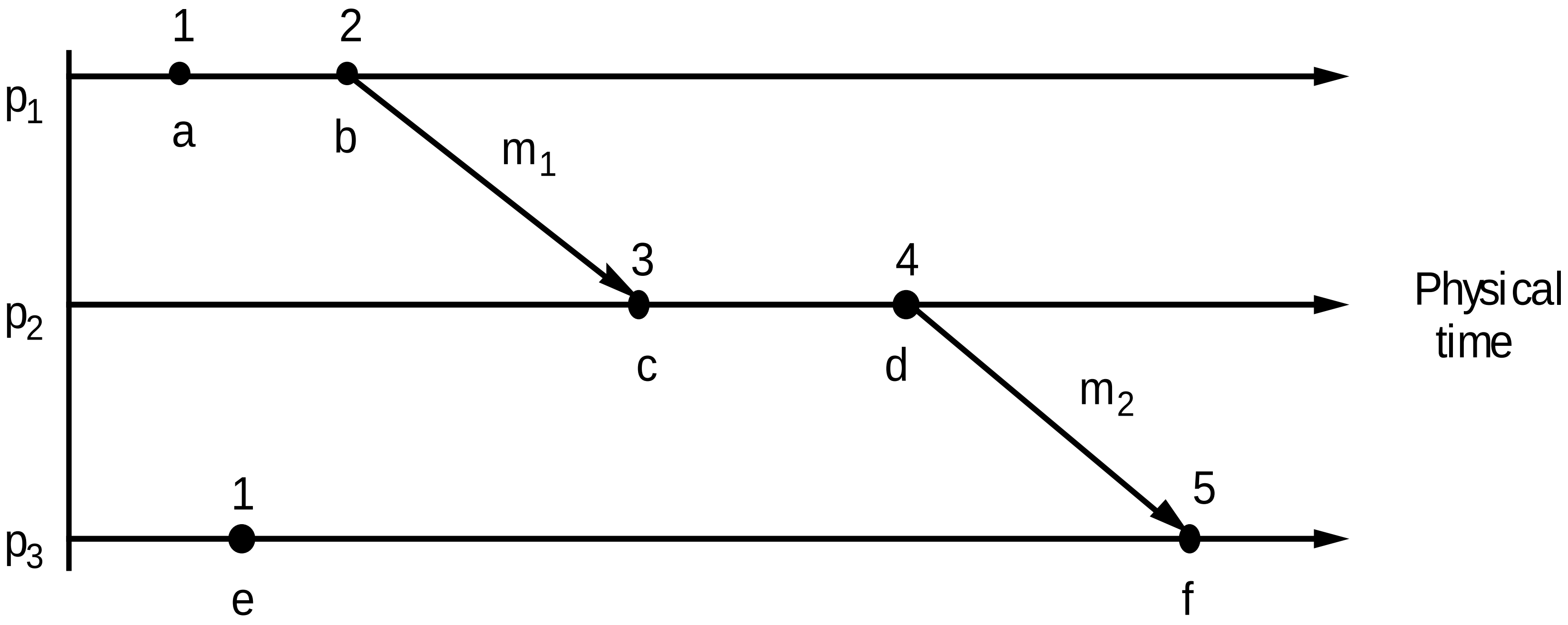
- Can we define sequence of events without using physical clocks?
  - Lamport logical clock
  - Vector clock
- Computing logical sequence of events

Why synchronize clocks?

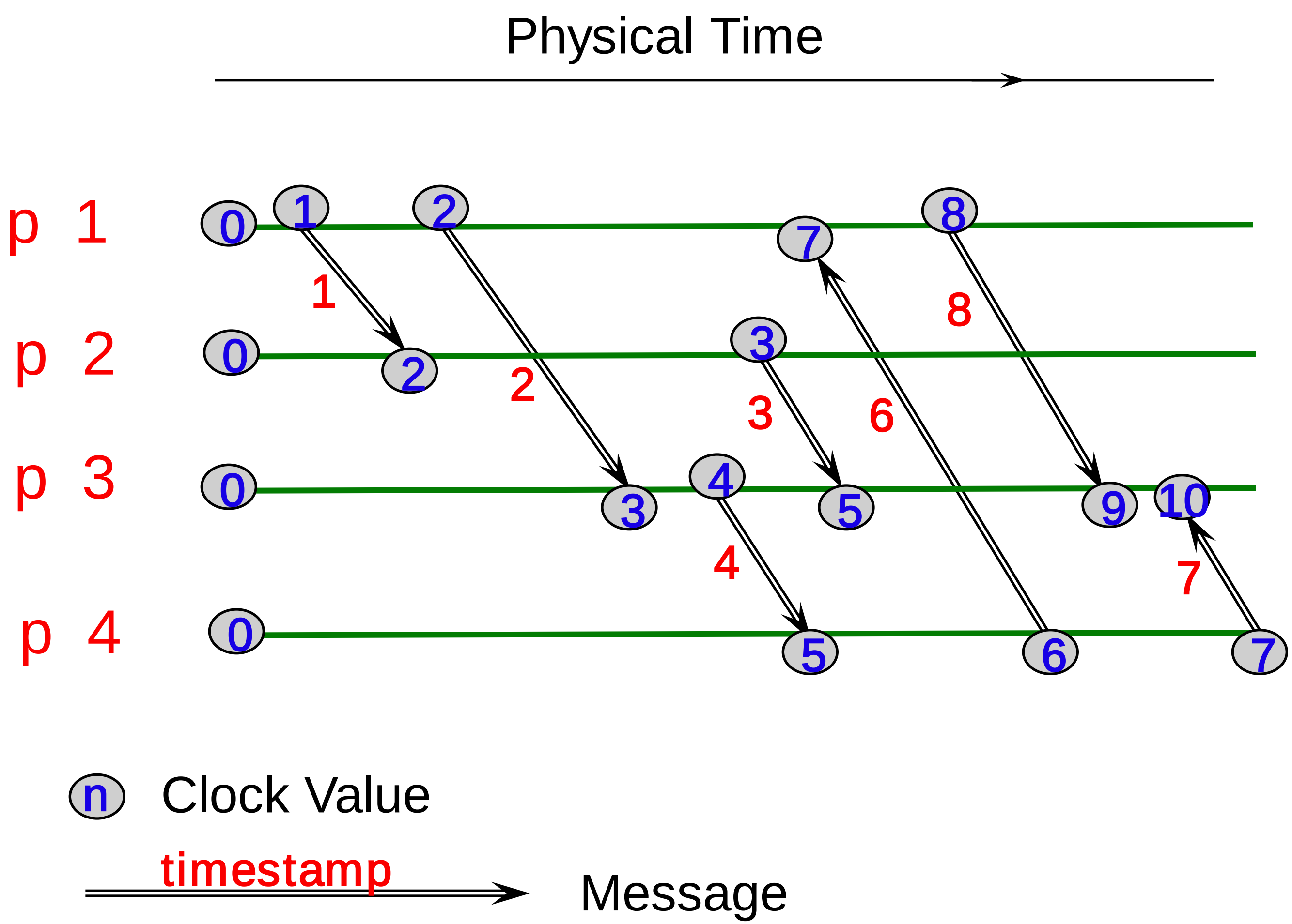
- **Two sharpshooters in a multiplayer online game kill the same target. Who gets the point?**
- **Object A is observed by S1 and S2 at local times t1 and t2. Which way is A moving? How fast?**
- **Synchronizing clocks helps us**
  - Time-stamping events (provide 'Fairness')
  - **Ordering events (provide 'Correctness')**



Review of Lamport Timestamps



Lamport Logical Time Example



Problem with Lamport Logical Clock

- **Let** *timestamp(a)* **be the Lamport logical clock timestamp**

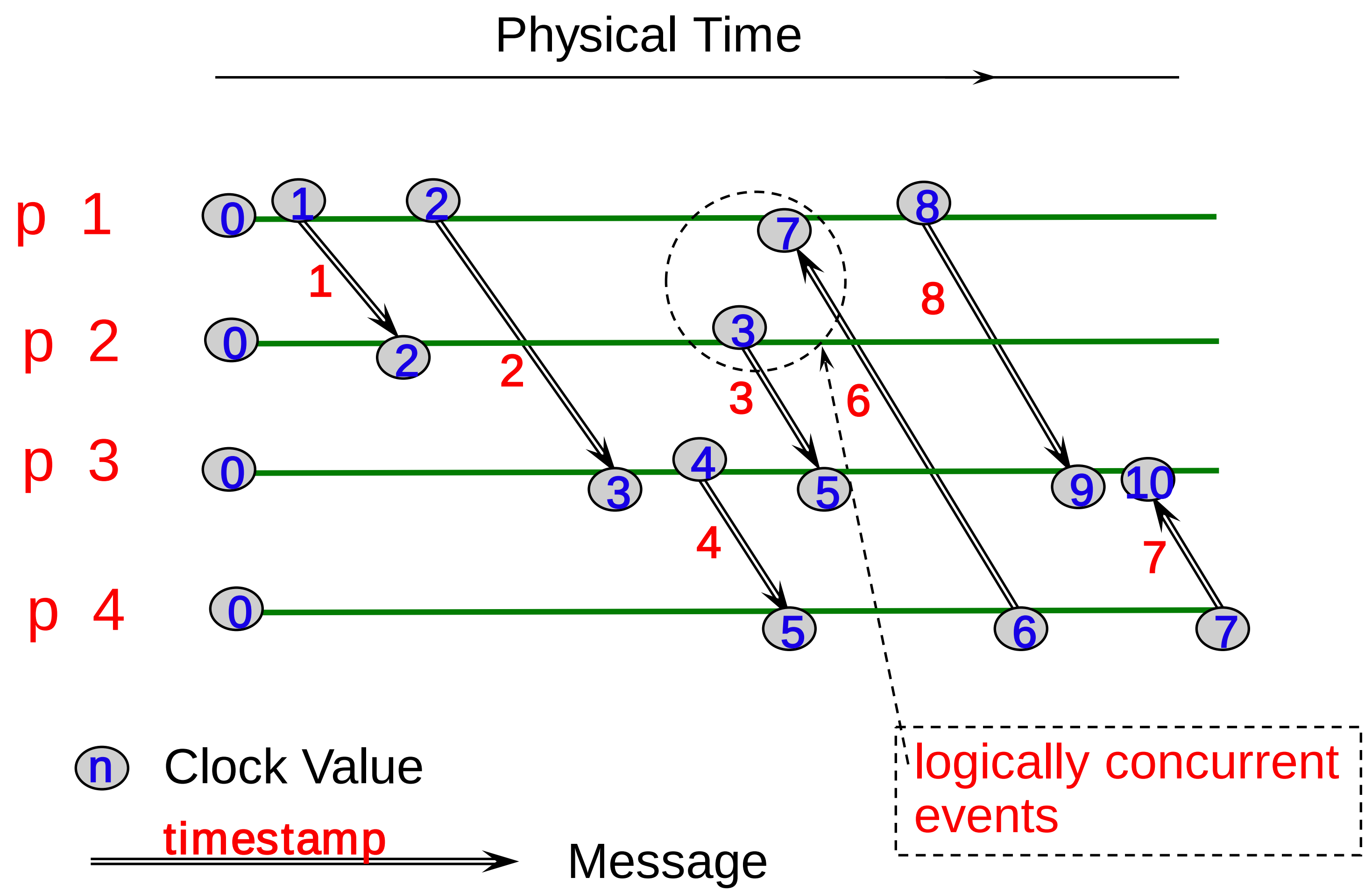
a → *b* **=>** *timestamp(a) < timestamp(b)*

(if *a* **happens before** *b***, then** *Lamport_timestamp(a) < Lamport_timestamp(b)***)**

timestamp(a) < timestamp(b) $\cancel{\Rightarrow}$ *a* → *b*

*(***If** *Lamport_timestamp(a) < Lamport_timestamp(b),* **it does NOT imply that** *a* **happens before** *b*

Example



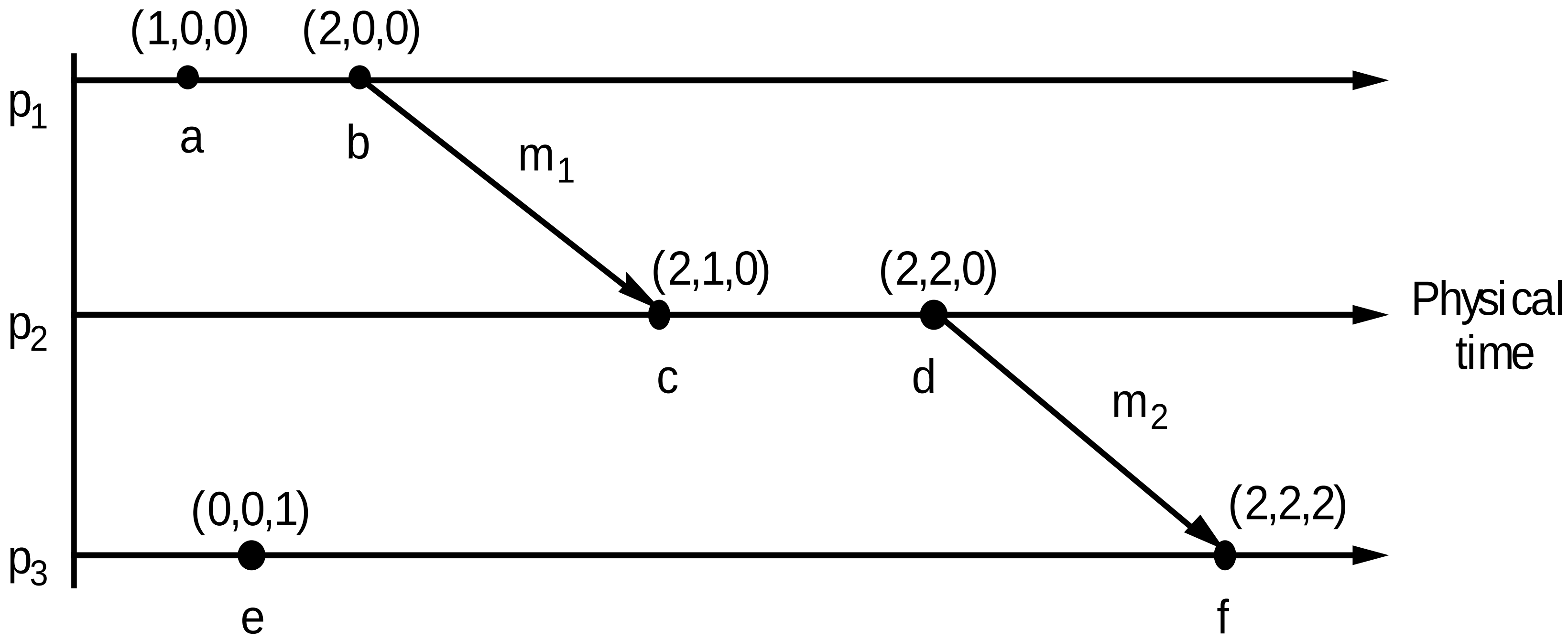
Note: Lamport Timestamps: 3 < 7, but event with timestamp 3 is concurrent to event with timestamp 7, i.e., events are not in ‘happen-before’ relation.

Vector Logical Clocks

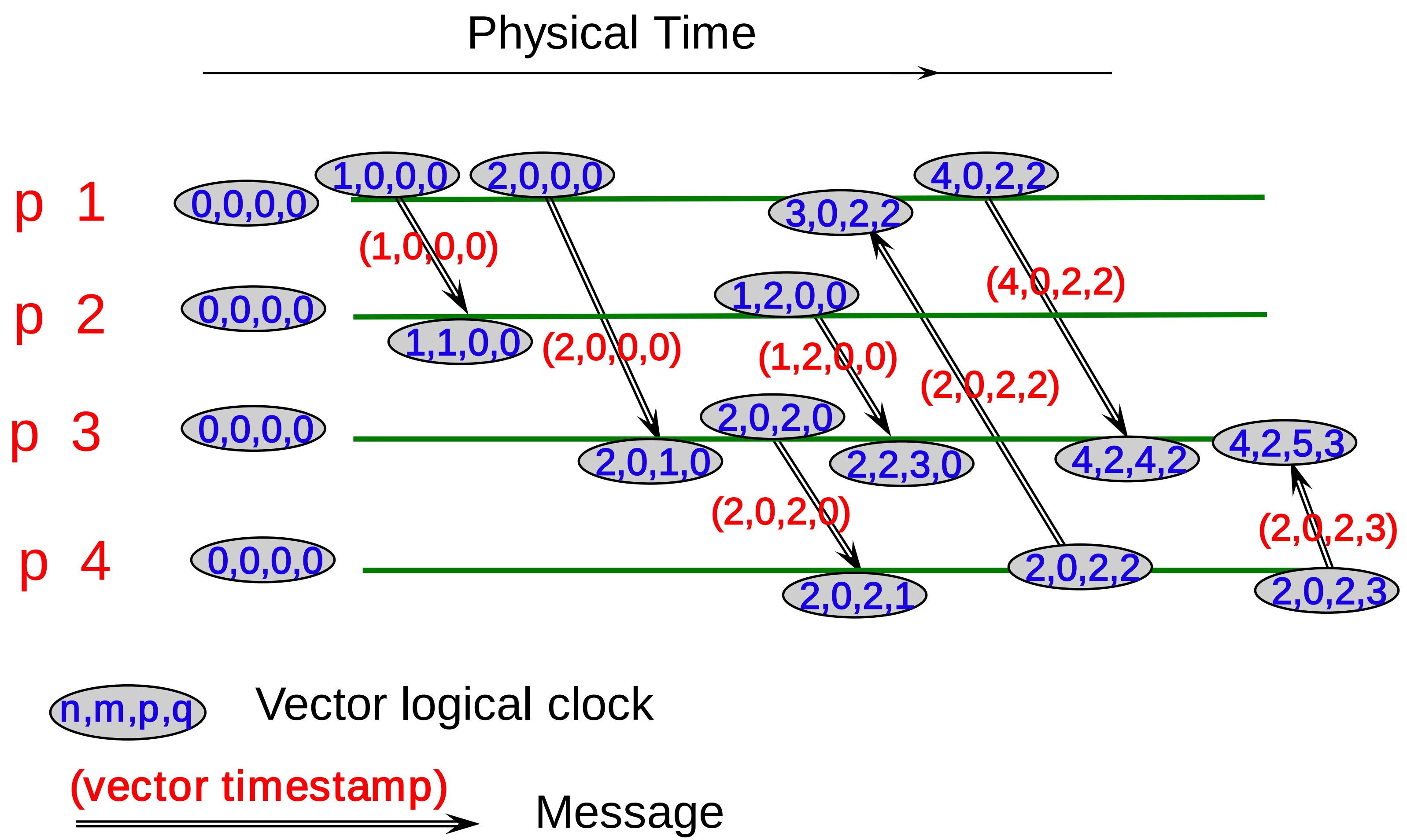
- **Vector Logical time is better:**
  - **All processes use a vector of counters (logical clocks), *i*th element is the clock value for process *i*, initially all zero.**
  - **Each process *i* increments the *i*th element of its vector upon an instruction or send event. Vector value is timestamp of the event.**
  - **A send(message) event carries its vector timestamp (counter vector)**
  - **For a receive(message) event,**

$$V_{receiver}[j] = \begin{cases} \text{Max}(V_{receiver}[j]\ ,\ V_{message}[j]), & \text{if } j \text{ is not self} \\ V_{receiver}[j] + 1 & \text{otherwise} \end{cases}$$

Vector Timestamps



Example: Vector Logical Time



Comparing Vector Timestamps

- $VT_1 = VT_2$,

  iff $VT_1[i] = VT_2[i]$, for all $i = 1, \ldots, n$
- $VT_1 \leqslant VT_2$,

  iff $VT_1[i] \leqslant VT_2[i]$, for all $i = 1, \ldots, n$
- $VT_1 < VT_2$,

  iff $VT_1 \leqslant VT_2$ &

  $\exists\, j\ (1 \leqslant j \leqslant n\ \&\ VT_1[j] < VT_2[j])$
- VT_1 is concurrent with VT_2

  iff (not $VT_1 \leqslant VT_2$ AND not $VT_2 \leqslant VT_1$)

Exercise

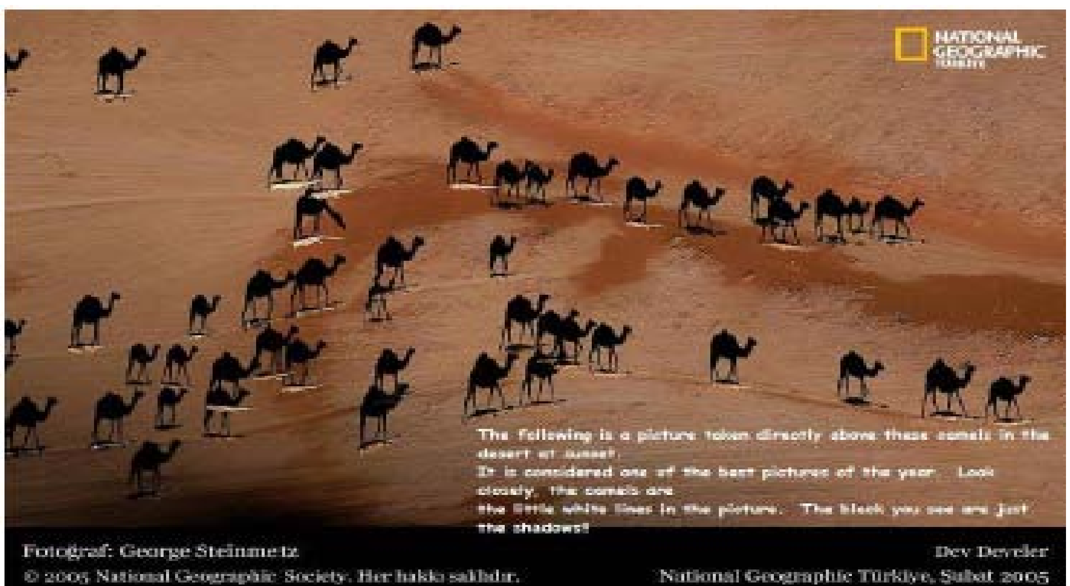
Show:

$a \rightarrow b$ **if and only if** $vectorTS(a) < vectorTS(b)$

Global State and Global Snapshot

- **Counting Camels**
  - **Satellites cover the entire area of interest**
  - **Each satellite can count the number of camels in its zone**
  - **How to compute the total number of camels?**

This is the Global State Problem



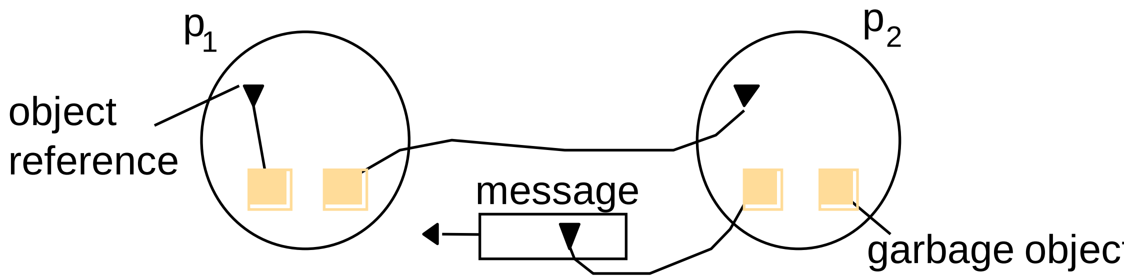
Example of a Global State

[United Nations photo by Paul Skipworth for Eastman Kodak Company ©1995]

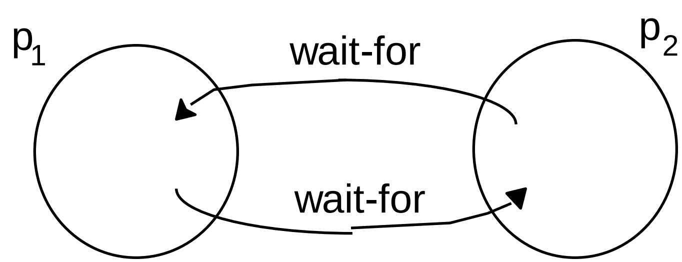
How would you take this photograph if each country's premier were sitting in their respective capital?
-- that's the challenge of distributed global snapshots!

Why take a global snapshot?

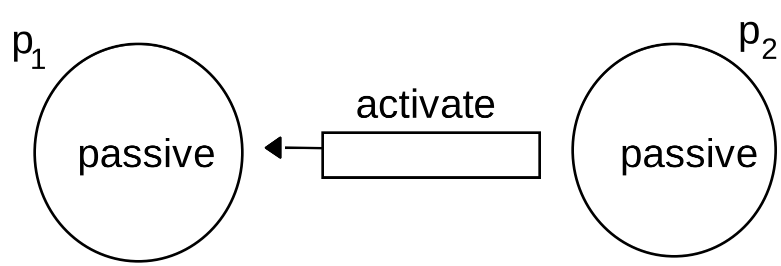
a. (Distributed)
Garbage collection



b. (Distributed)
Deadlock Detection

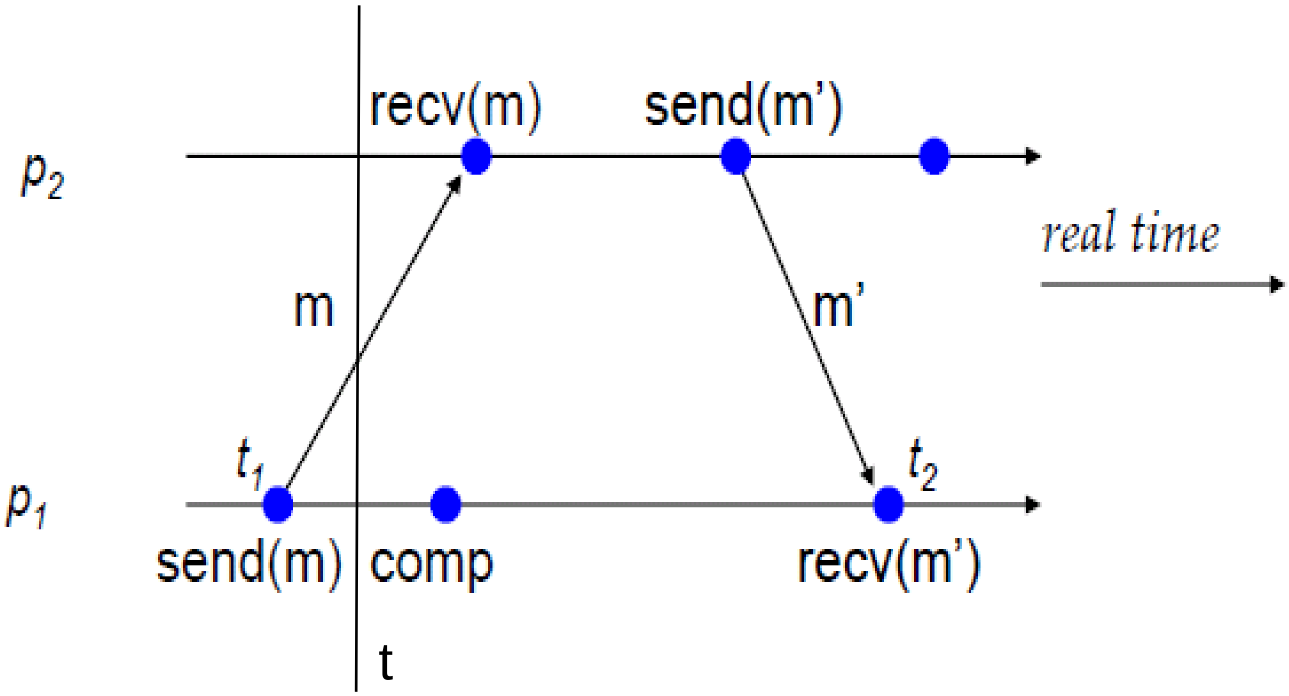


c. Termination Detection



What is Global state of a Distributed System?

- **At real time *t*, the global state is**
  - **Instantaneous states of all processes**
  - **Messages in transit on channels**

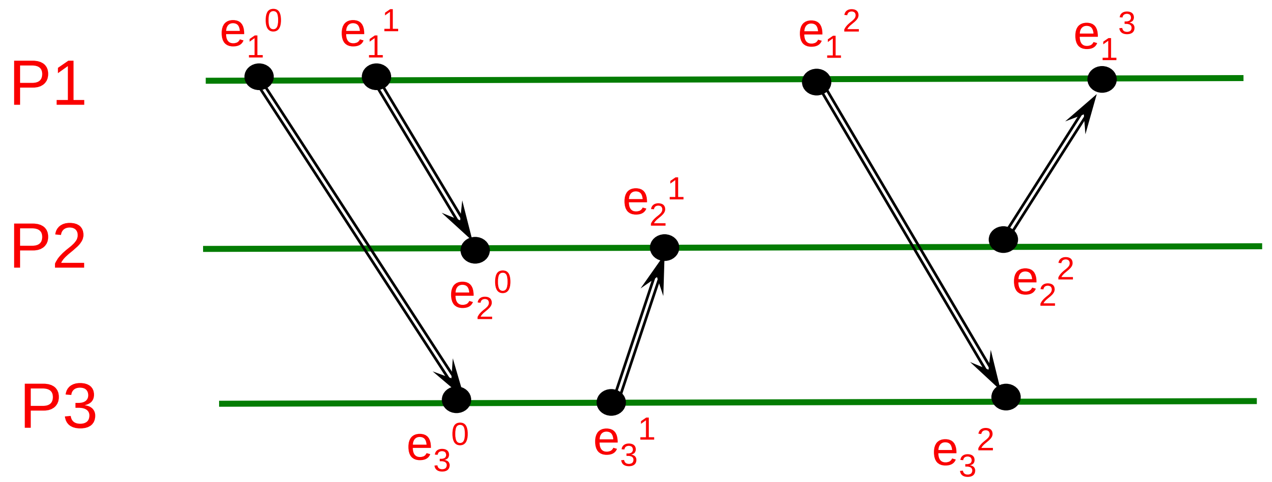


Obvious First Attempt

- If we have perfectly synchronized clocks
  - Each process p_i records its x_i state at some future time t
  - Sends $x_i(t)$ to all other processes
- Problem
  - This approach does not record state of the channels
  - Perfect synchronization is not possible (message delay un-certainity is nonzero)
- Relax the Problem
  - Instead of actual global state of the system we will try to capture a consistent global state

Process History

- For a process P_i , where events e_i^0, e_i^1, … *occur:*
  history(P_i) = h_i = <e_i^0, e_i^1, … >
  prefix history(P_i^k) = h_i^k = <e_i^0, e_i^1, …,e_i^k >
  S_i^k : P_i's state immediately **before *k*^th event**



Global History and Cut

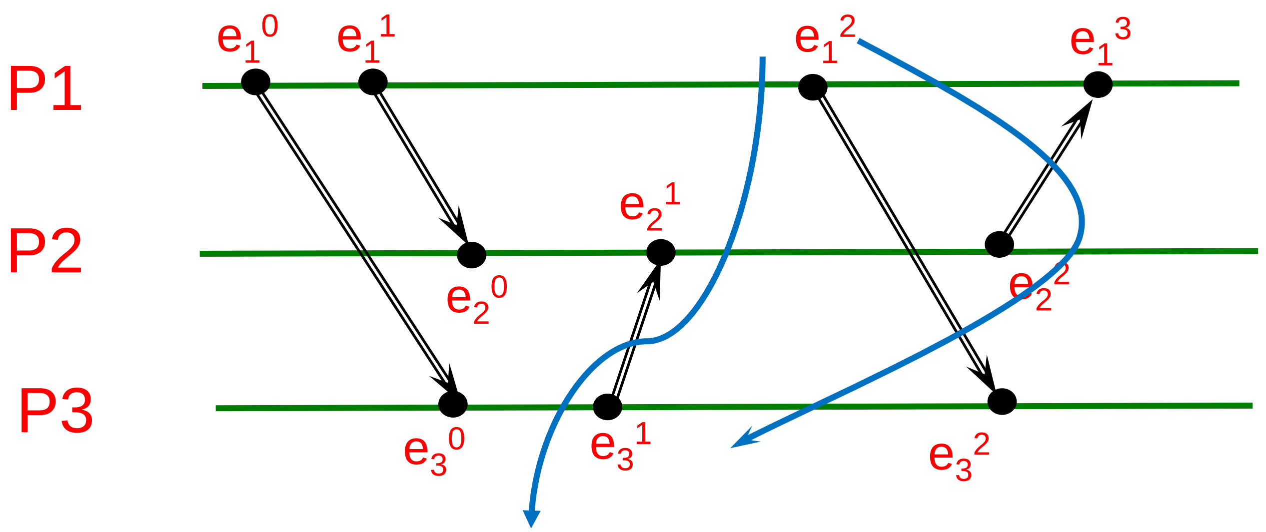
❖ **For a set of processes P_1, …, P_i, …. :**

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

global state: $S = \cup_i (S_i^{k_i})$

a cut $C \subseteq H = h_1^{c1} \cup h_2^{c2} \cup \ldots \cup h_n^{cn}$

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



Note: Global state does not record the state of the channels separately

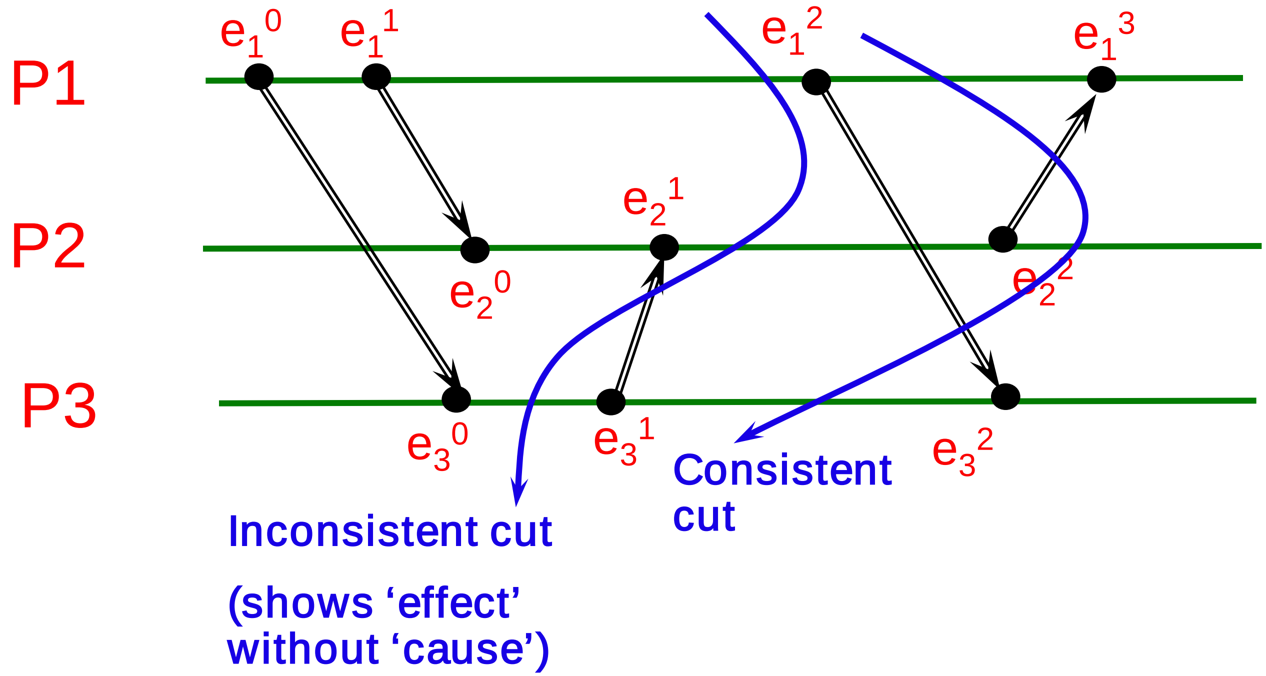
Consistent States

Lamport's "happens-before"

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

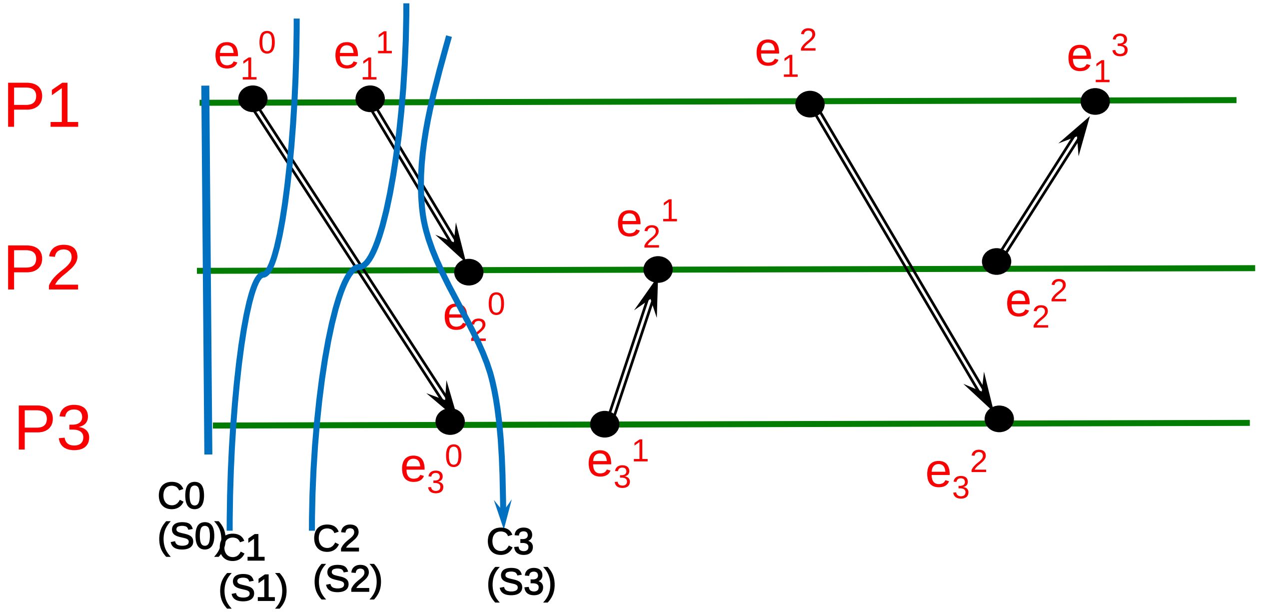
$$\forall e \in C \; (\text{if } f \rightarrow e \text{ then } f \in C)$$

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



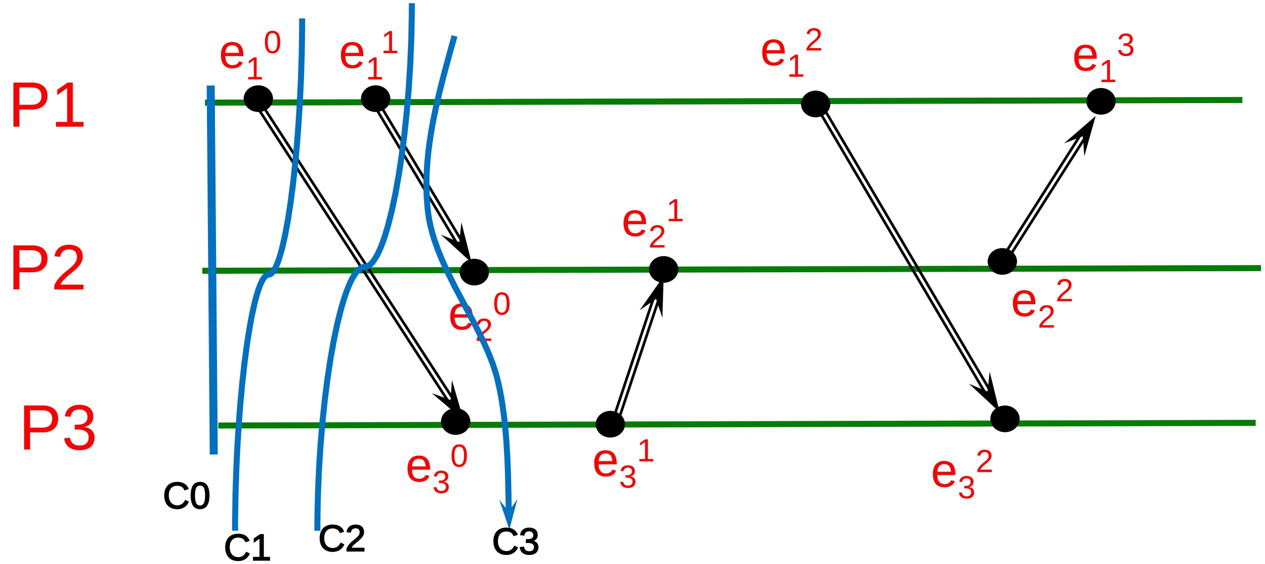
System Execution

- An **execution** of a distributed system is a sequence of transitions between consistent global states of the system S_0 -> S_1 -> S_2 -> S_3
  - *In each transition one event occurs at some single process*



Runs and Linearization

- A **run** is a **total ordering** of events in H that is consistent with each h_i's ordering
  - E.g., $\langle e_1^0, e_1^1, e_1^2, e_1^3, e_2^0, e_2^1, e_2^2, e_3^0\ e_3^1, e_3^2 \rangle$
- A **linearization** is a run consistent with happens-before ($\rightarrow$) relation in H
  - E.g., $\langle e_1^0, e_1^1, e_3^0, e_2^0, \ldots \rangle$, $\langle e_1^0, e_3^0, e_1^1, e_2^0, \ldots \rangle$
  - *Concurrent events are ordered arbitrarily*
  - *Linearizations pass through consistent global states*



State Reachability

- **A global state S_k is reachable from global state S_i, if there is a linearization, L, that passes through S_i and then through S_k.**
- **A DS evolves as a series of transitions between global states S_0, S_1, ….**
- **We denote set of all global states by *S***

Global State Predicates

- **Predicate** *P* **on a set** *S* **is a function** *P: S ----> {true, false}*
- **Global state predicate** *P* **is a function** *P: S ----> {true, false}, where S is a global state set;* ***e.g., deadlock, termination***
- **Given a global state predicate** *P*
  - ***Stable(P) := if P(S) = true and S goes to S', then P(S')=true***
    - » ***Once P becomes true, it remains true in subsequent global states***
    - » ***E.g., deadlocked, terminated, an object O is orphaned***
  - ***Safety(P) := stable(P) and for every initial state*** S_0, ***P***(S_0) ***= true***
    - » ***All reachable states satisfy P***
    - » ***If P is a desirable property, this means that nothing bad ever happens***
    - » ***E.g., P = 'not deadlocked'***
  - ***Liveness(P) := eventually P holds***
    - » ***There exists L є linearization from*** S_0, S_L: ***L passes through*** S_L ***&*** ***P***(S_L) ***= true***
- **We need a way to record global state**

Quick Note – Liveness versus Safety

Can be confusing, but terms are relevant outside CS too:

- **Liveness=guarantee that something good will happen eventually**
  - **"Guarantee of termination" is a liveness property**
  - **Guarantee that "at least one of the athelets in the 100m final will win gold" is liveness**
  - **A criminal will eventually be jailed**
- **Safety=guarantee that something bad will never happen**
  - **Deadlock avoidance algorithms provide safety**
  - **A peace treaty between two nations provides safety**
  - **An innocent person will never be jailed**
- **Can be difficult to satisfy both liveness and safety!**

Summary

- **This class:**
  - importance of logical clocks (Lamport and Vector Clocks),
  - global snapshots

- **Reading for next class: Section 11.5.3 & 12.4**
  - Discussion on Chandy-Lamport Algorithm
  - Multicast Communication