Lecture 16

- Administration

Logical clocks

Causality is important

Not possible to have a global "physical clock"

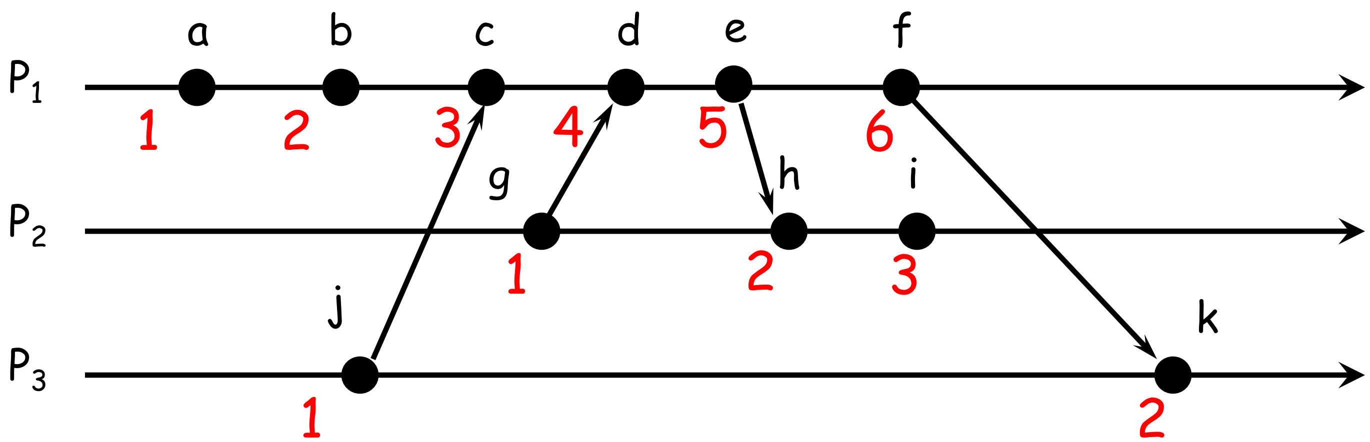
Causality used to reason, analyze and prove properties of concurrent systems

Systems depends on ordering only NOT the speed at which things take place

Event counting example

- Three systems: P_0, P_1, P_2
- Events *a*, *b*, *c*, …
- Local event counter on each system
- Systems occasionally communicate

Event counting example



Bad ordering:

e → h

f → k

Lamport's algorithm

- Each message carries a timestamp of the sender's clock
- When a message arrives:
  - if receiver's clock < message timestamp
    set system clock to (message timestamp + 1)
  - else do nothing
- Clock must be advanced between any two events in the same process

Properties of a clock

A system of clocks and a time domain where for every event e using the clock we can assign a value from the time domain.

$$C : H \rightarrow T$$

- Clocks should be monotonically increasing

- Consistent, $e_i \rightarrow e_j \Rightarrow c(e_i) < c(e_j)$

Strongly consistent $e_i \rightarrow e_j \Leftrightarrow c(e_i) < c(e_j)$

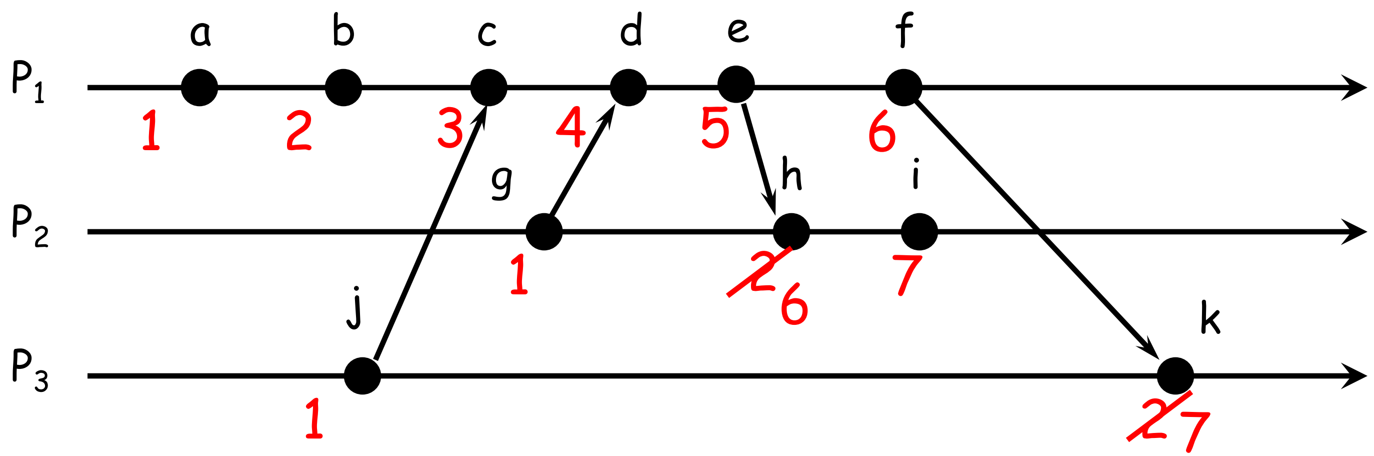
Lamport's Clock

- R1: Internal events $Ci = Ci + d$ ($d>0$, typically $d=1$)

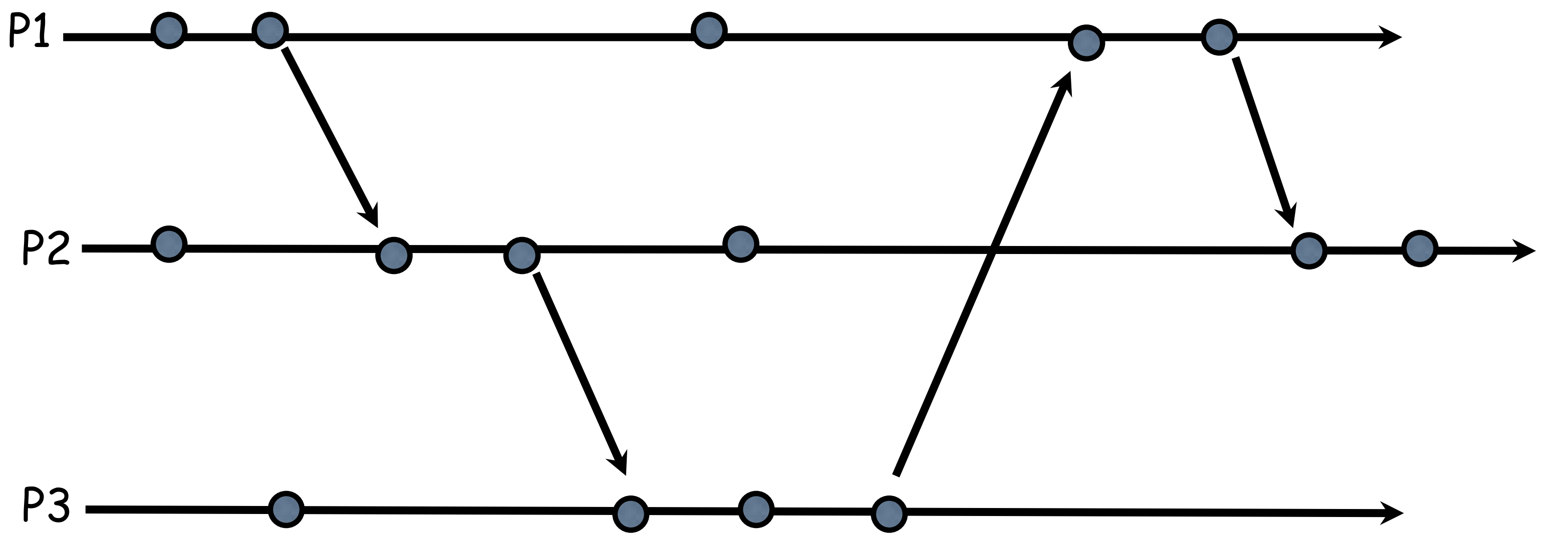
- R2: Piggyback local logical clock on every message, when received do the following:

  - $Ci = \max(Ci, C \text{ value in message})$
  - Execute R1 (updates time)
  - Deliver the message

Event counting example



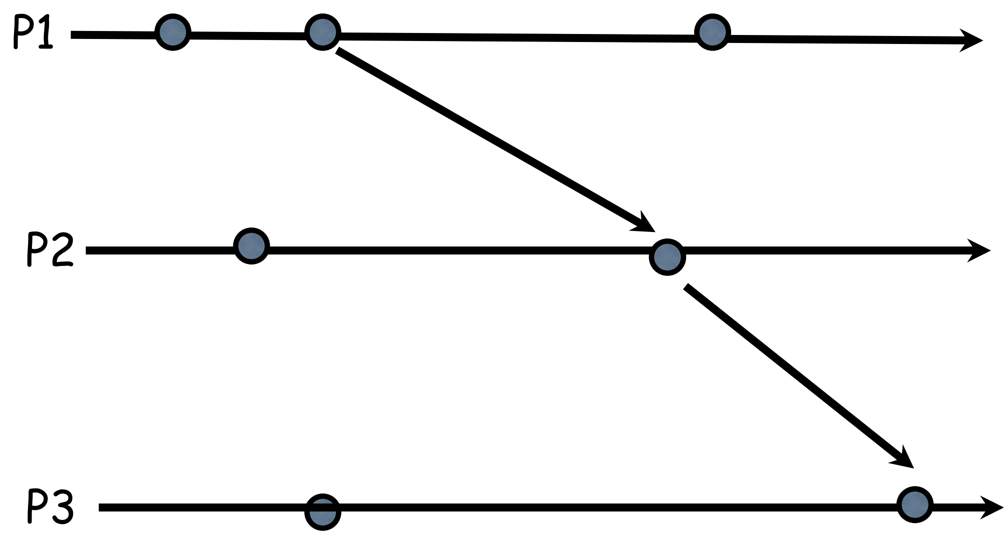
Example from Notes

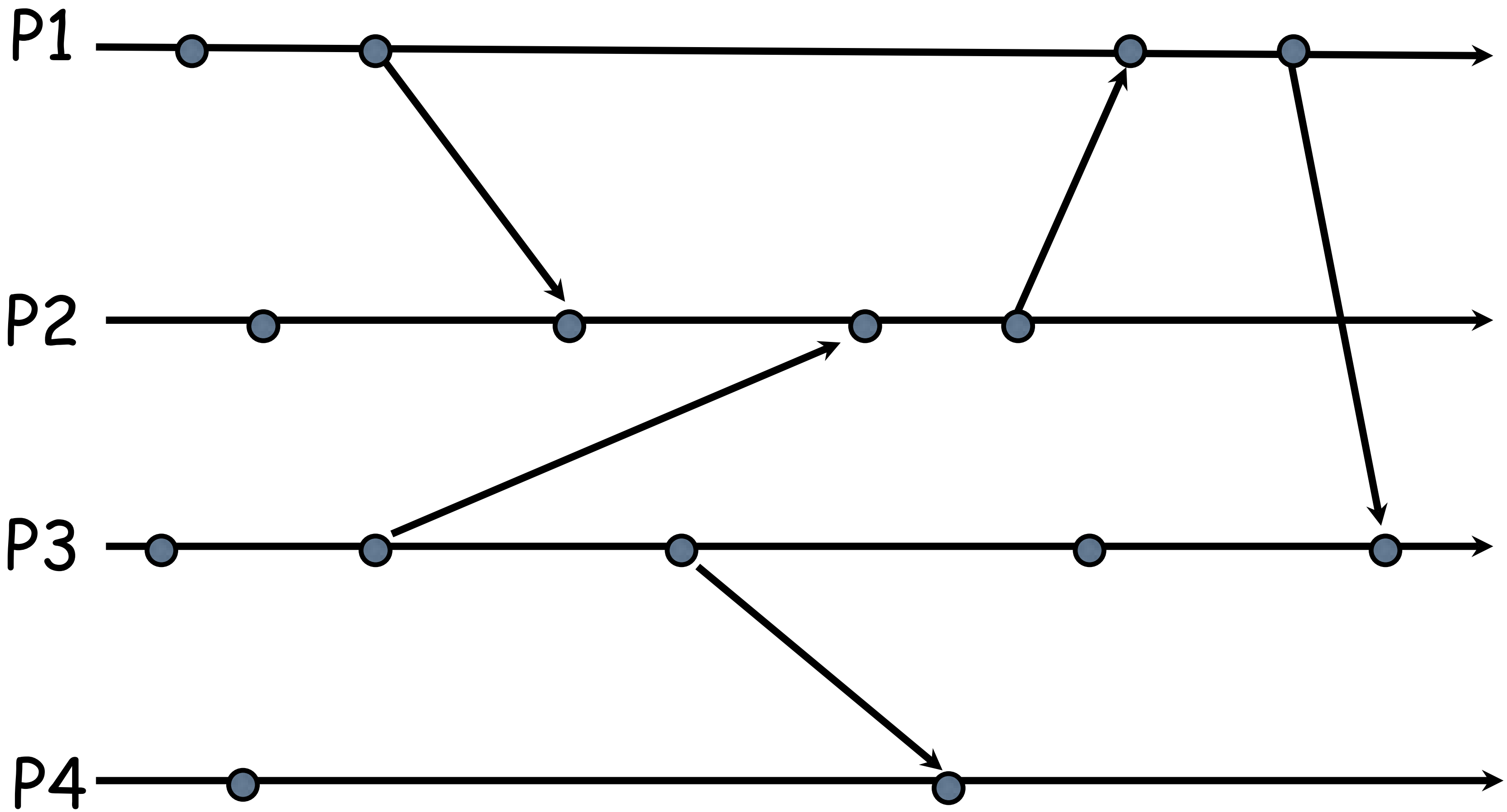


Summary

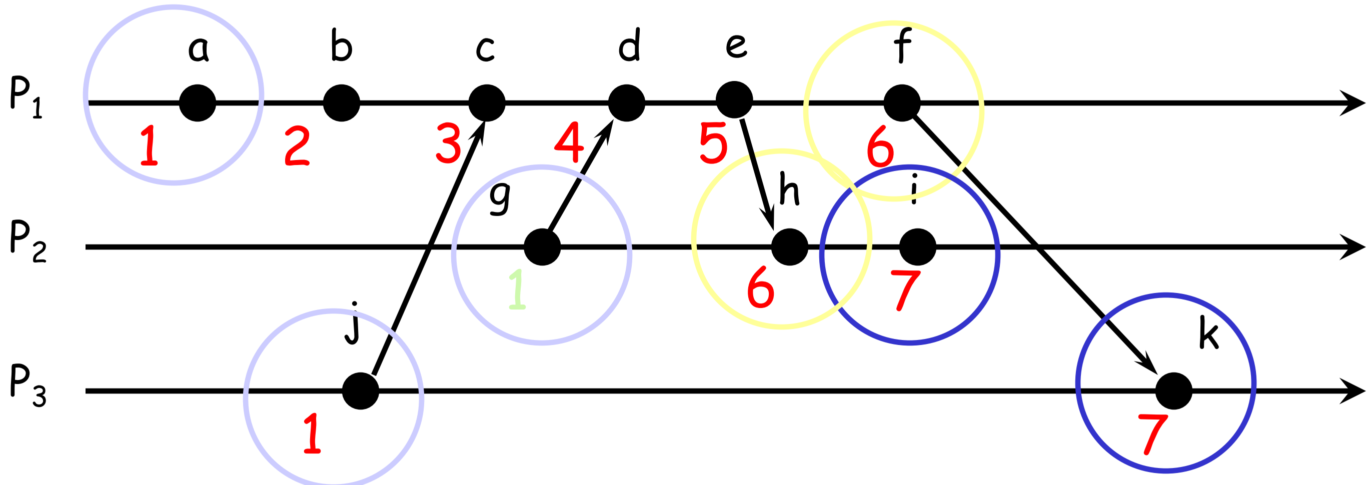
- Algorithm needs monotonically increasing software counter
- Incremented at least when events that need to be timestamped occur
- Each event has a **Lamport timestamp** attached to it
- For any two events, where $a \rightarrow b$:

  $$L(a) < L(b)$$


P1
P2
P3


P1
P2
P3
P4

Problem: Identical timestamps



a→b, b→c, …: local events sequenced

i→c, f→d , d→g, … : Lamport imposes a *send*→*receive* relationship

Concurrent events (e.g., a and i) *may* have the same timestamp … or not

Unique timestamps (total ordering)

We can force each timestamp to be unique

- Define global logical timestamp (T_i, i)
  - T_i represents local Lamport timestamp
  - i represents process number (globally unique)
    - E.g. (host address, process ID)

- Compare timestamps:

  $(T_i, i) < (T_j, j)$

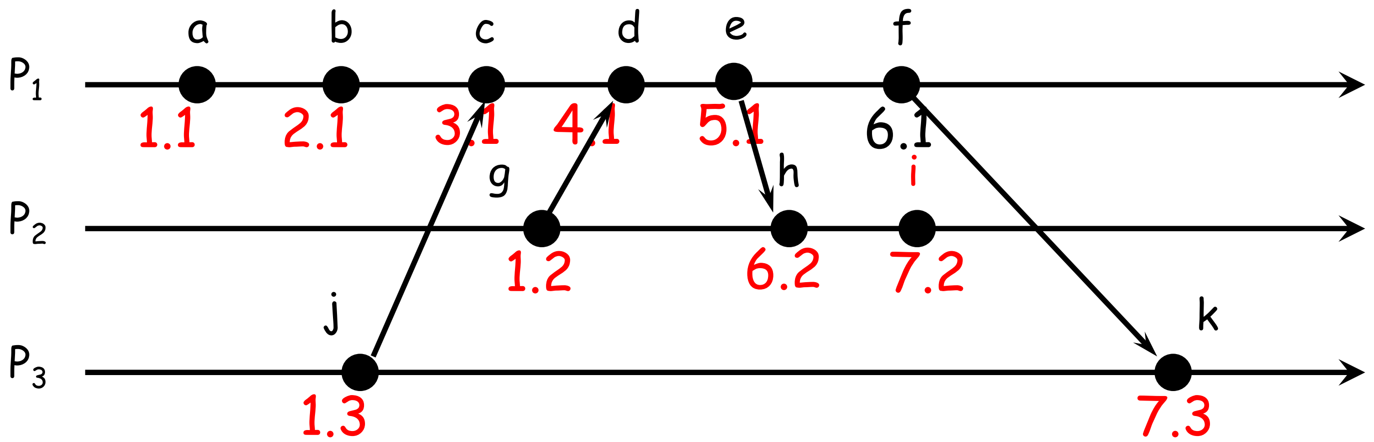
  if and only if

  $T_i < T_j$ or

  $T_i = T_j$ and $i < j$

Does not relate to event ordering

Unique (totally ordered) timestamps



Problem: Detecting causal relations

If $L(e) < L(e')$

- Cannot conclude that $e \rightarrow e'$

Looking at Lamport timestamps

- Cannot conclude which events are causally related

Solution: use a **vector clock**

Vector clocks

Rules:

1. Vector initialized to 0 at each process

   $V_i[j] = 0$ for $i, j = 1, \ldots, N$

2. Process increments its element of the vector in local vector before timestamping event:

   $V_i[i] = V_i[i] + 1$

3. Message is sent from process P_i with V_i attached to it

4. When P_j receives message, compares vectors element by element and sets local vector to higher of two values

   $V_j[i] = \max(V_i[i], V_j[i])$ for $i=1, \ldots, N$

Comparing vector timestamps

Define

$V = V'$ iff $V[i] = V'[i]$ for $i = 1 \dots N$

$V \leq V'$ iff $V[i] \leq V'[i]$ for $i = 1 \dots N$

For any two events e, e′

if $e \rightarrow e'$ then $V(e) < V(e')$

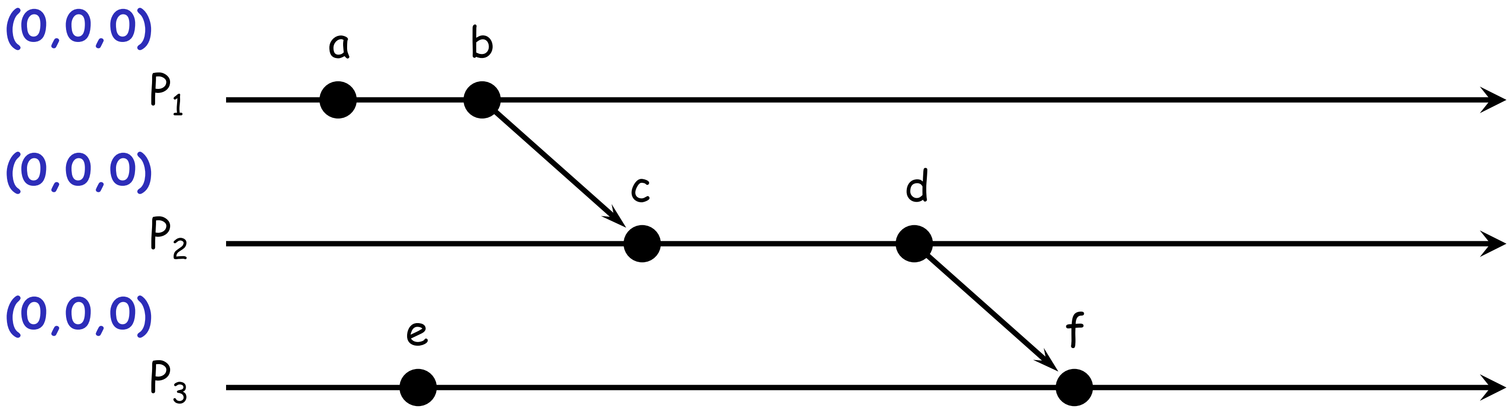
- Just like Lamport's algorithm

if $\mathbf{V(e) < V(e')}$ **then** $\mathbf{e \rightarrow e'}$

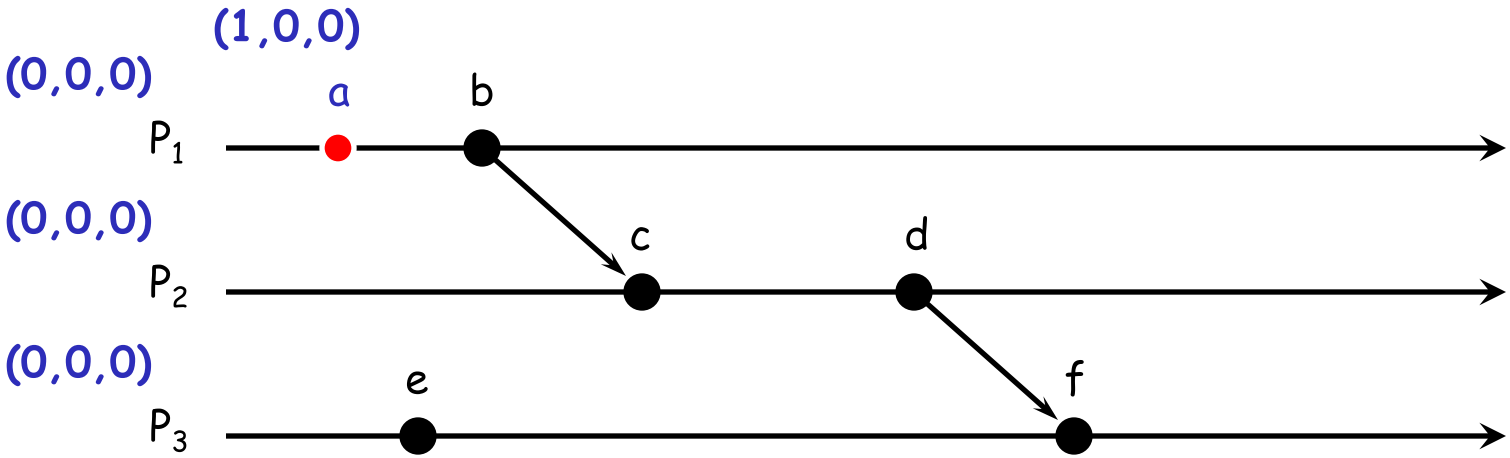
Two events are **concurrent** if **neither**

$V(e) \leq V(e')$ nor $V(e') \leq V(e)$

Vector timestamps

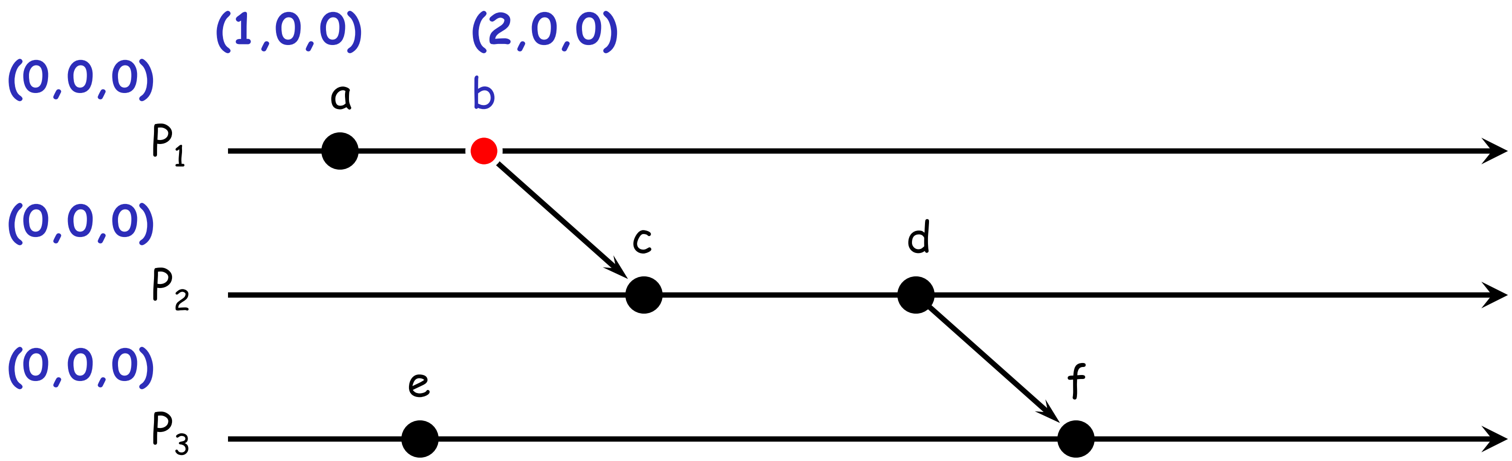


Vector timestamps



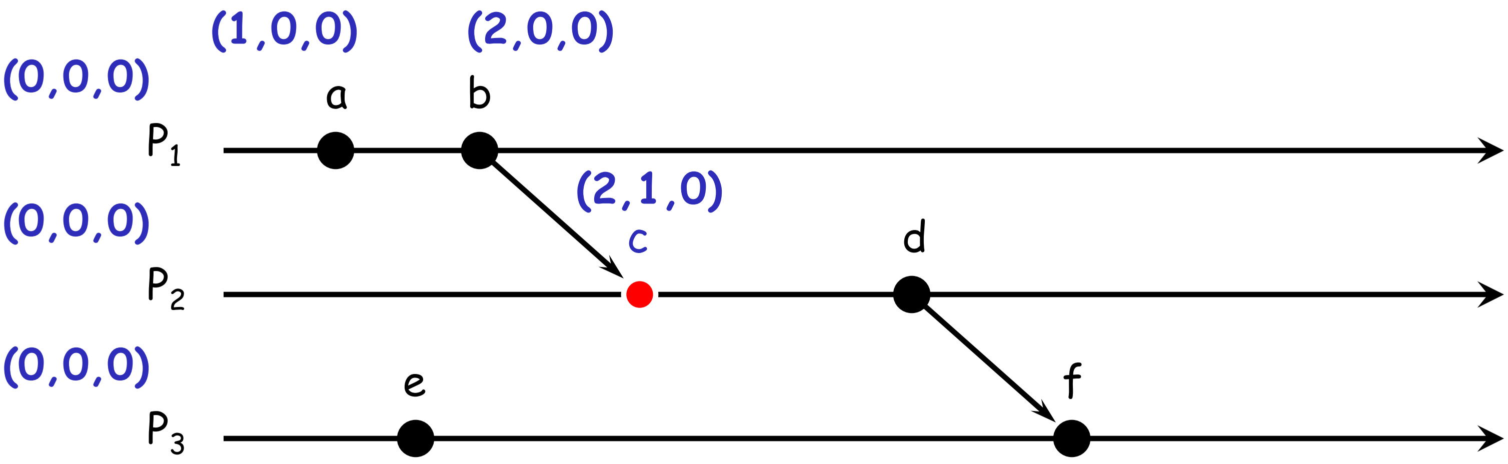
Event	timestamp
a	(1,0,0)

Vector timestamps



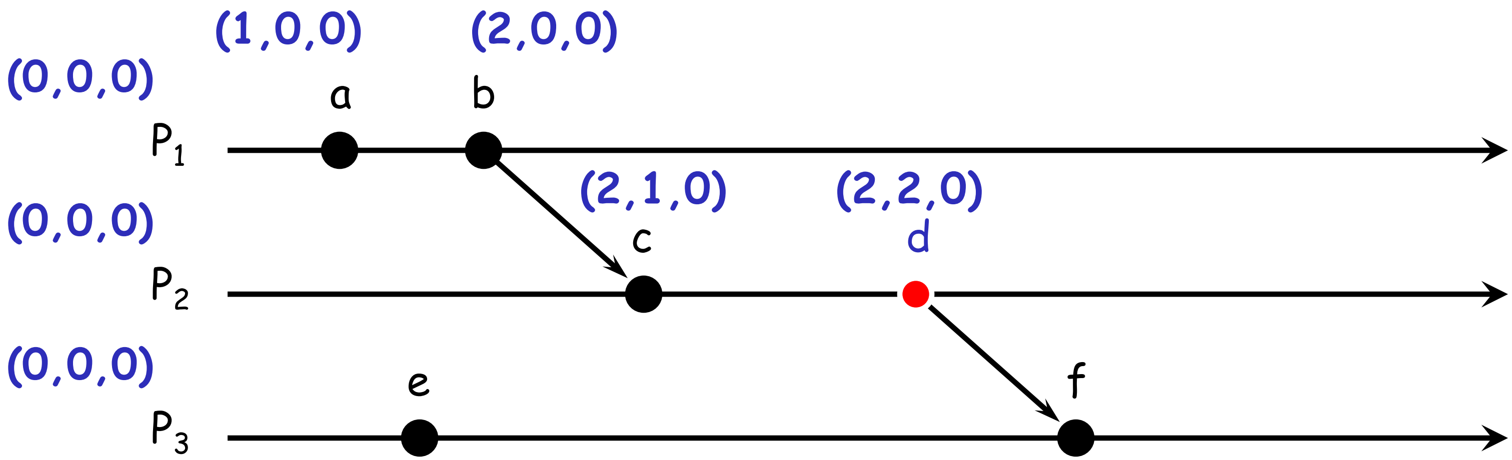
Event	timestamp
a	(1,0,0)
b	(2,0,0)

Vector timestamps



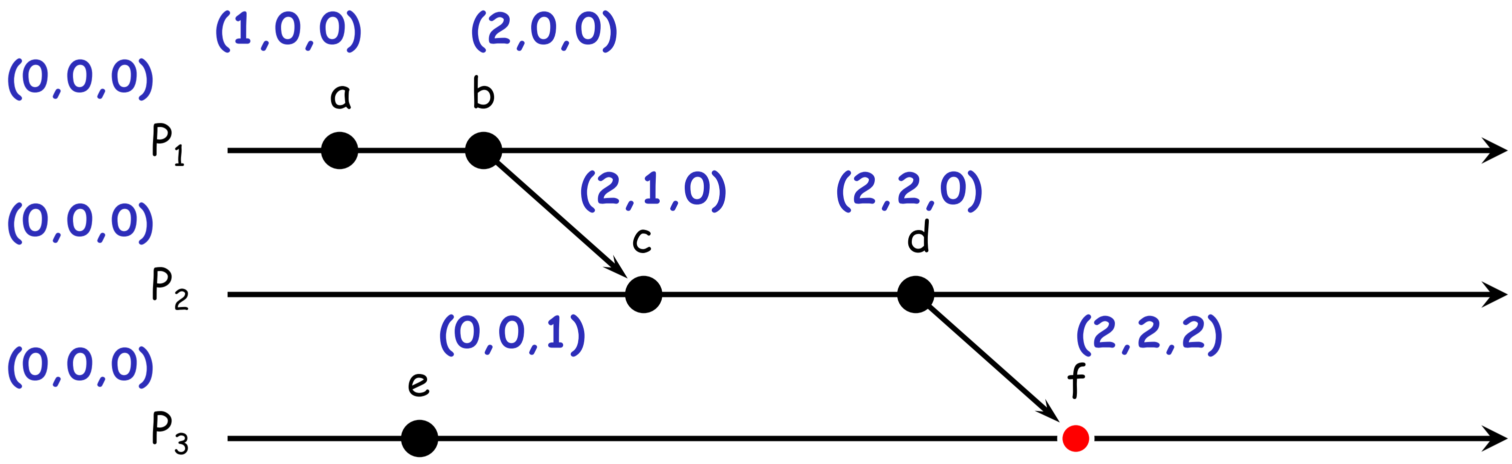
Event	timestamp
a	(1,0,0)
b	(2,0,0)
c	(2,1,0)

Vector timestamps



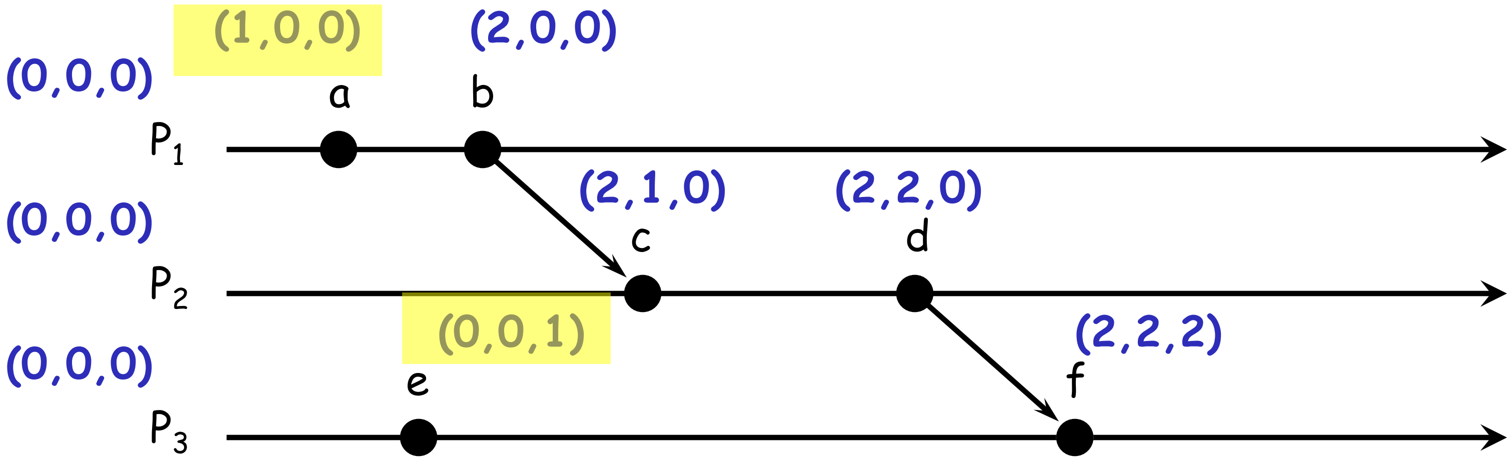
Event	timestamp
a	(1,0,0)
b	(2,0,0)
c	(2,1,0)
d	(2,2,0)

Vector timestamps



Event	timestamp
a	(1,0,0)
b	(2,0,0)
c	(2,1,0)
d	(2,2,0)
e	(0,0,1)
f	(2,2,2)

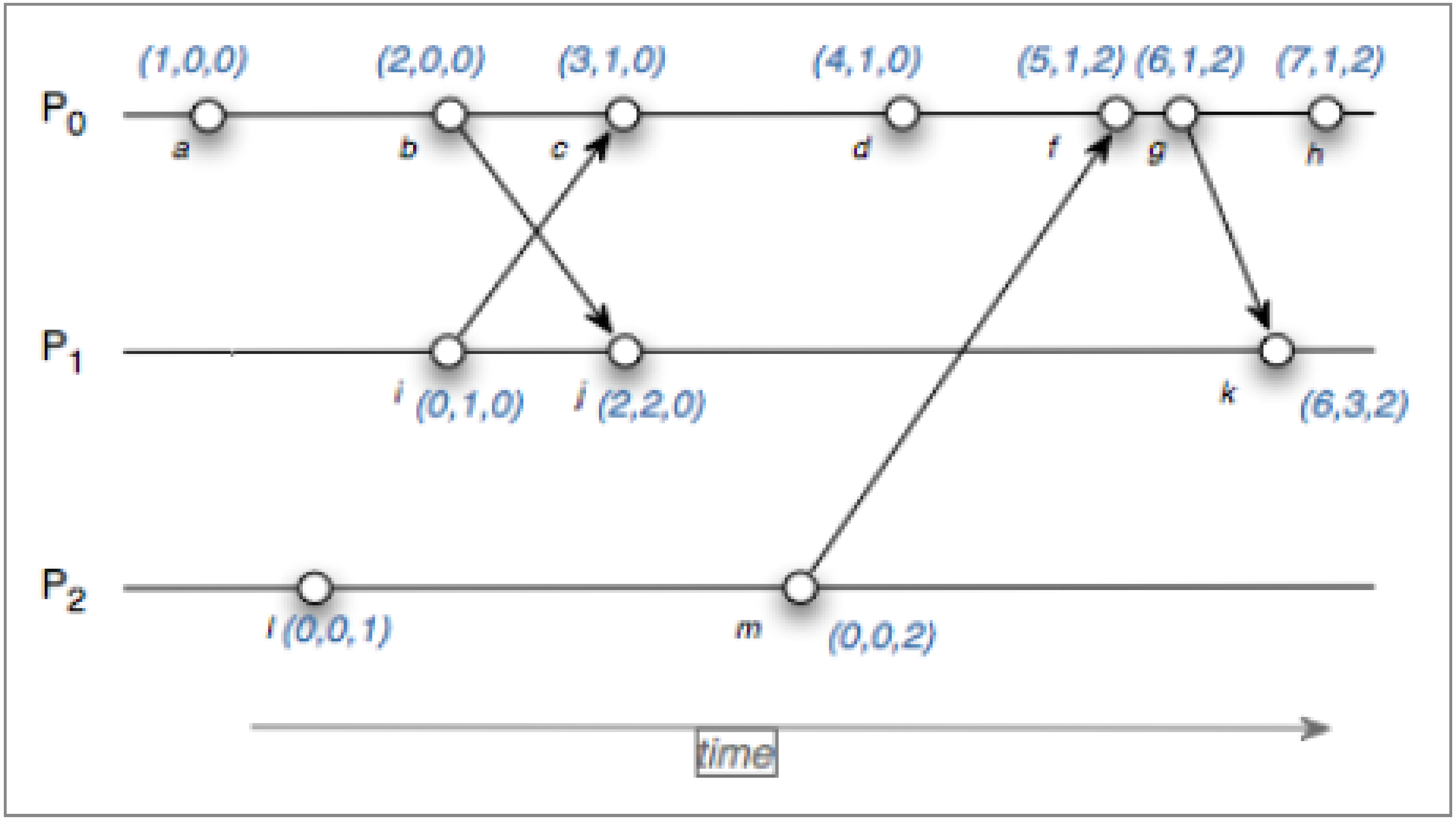
Vector timestamps



Event	timestamp
a	(1,0,0)
b	(2,0,0)
c	(2,1,0)
d	(2,2,0)
e	(0,0,1)
f	(2,2,2)

concurrent events

Another Example



Vector Clock Assignments

Summary: Logical Clocks & Partial Ordering

- Causality
  - If *a->b* then event *a* can affect event *b*
- Concurrency
  - If neither *a->b* nor *b->a* then one event cannot affect the other
- Partial Ordering
  - Causal events are sequenced
- Total Ordering
  - All events are sequenced