# Distributed Systems

Logical Clocks

Paul Krzyzanowski pxk@cs.rutgers.edu

# Logical clocks

#### Assign sequence numbers to messages

- All cooperating processes can agree on order of events
- vs. physical clocks: time of day

#### Assume no central time source

- Each system maintains its own local clock
- No total ordering of events
  - · No concept of happened-when

# Happened-before

### Lamport's "happened-before" notation

```
a \rightarrow b event a happened before event b
e.g.: a: message being sent, b: message receipt
```

Transitive:

if  $a \rightarrow b$  and  $b \rightarrow c$  then  $a \rightarrow c$ 

# Logical clocks & concurrency

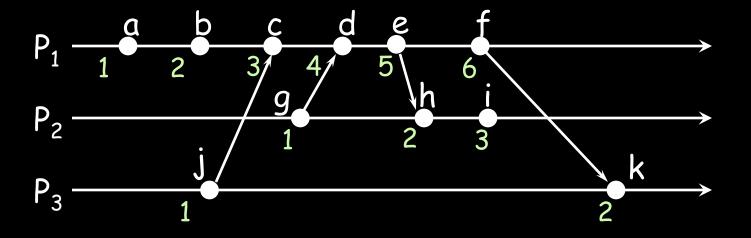
Assign "clock" value to each event

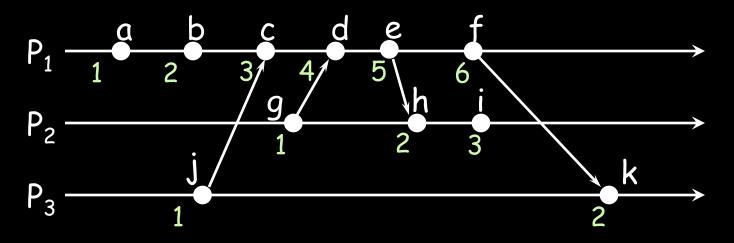
- if  $a \rightarrow b$  then clock(a) < clock(b)
- since time cannot run backwards

If a and b occur on different processes that do not exchange messages, then neither  $a \rightarrow b$  nor  $b \rightarrow a$  are true

- These events are concurrent

- · Three systems: P<sub>0</sub>, P<sub>1</sub>, P<sub>2</sub>
- Events a, b, c, ...
- Local event counter on each system
- · Systems occasionally communicate





#### Bad ordering:

$$e \rightarrow h$$

$$f \rightarrow k$$

# Lamport's algorithm

 Each message carries a timestamp of the sender's clock

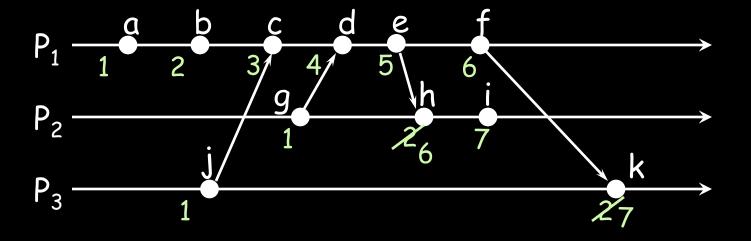
- When a message arrives:
  - if receiver's clock < message timestamp set system clock to (message timestamp + 1)
  - else do nothing

 Clock must be advanced between any two events in the same process

# Lamport's algorithm

Algorithm allows us to maintain time ordering among related events

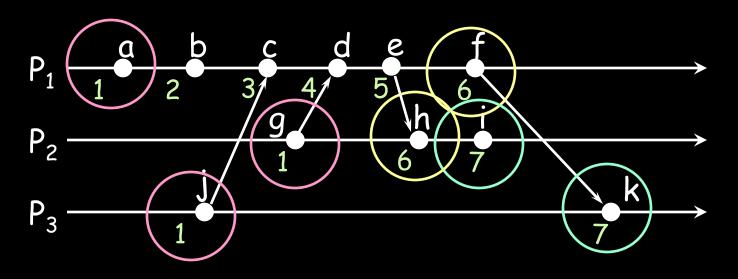
- Partial ordering



# 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)

# Problem: Identical timestamps



 $a \rightarrow b$ ,  $b \rightarrow c$ , ...: local events sequenced  $i \rightarrow c$ ,  $f \rightarrow d$ ,  $d \rightarrow g$ , ...: Lamport imposes a  $send \rightarrow receive$  relationship

Concurrent events (e.g., a & i) <u>may</u> have the same timestamp ... or not

# Unique timestamps (total ordering)

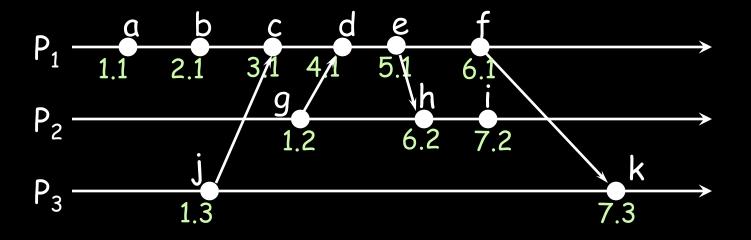
#### We can force each timestamp to be unique

- Define global logical timestamp (Ti, i)
  - · Ti 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_i 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, ..., N
- Process increments its element of the vector in local vector before timestamping event:
   V<sub>i</sub>[i] = V<sub>i</sub>[i] +1
- 3. Message is sent from process P<sub>i</sub> with V<sub>i</sub> attached to it
- 4. When P<sub>j</sub> 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, ...,  $N$ 

### Comparing vector timestamps

#### <u>Define</u>

```
 V = V' \text{ iff } V[i] = V'[i] \text{ for } i = 1 \dots N   V \leq V' \text{ iff } V[i] \leq V'[i] \text{ for } i = 1 \dots N  For any two events e, e'
```

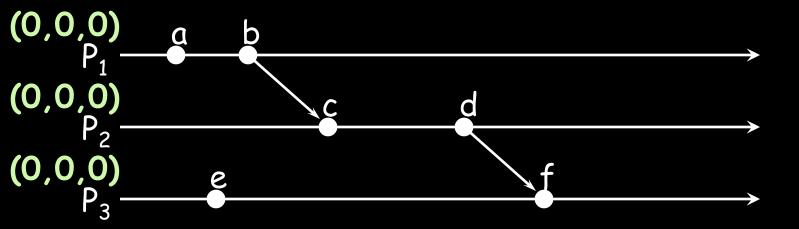
if  $e \rightarrow e'$  then V(e) < V(e')

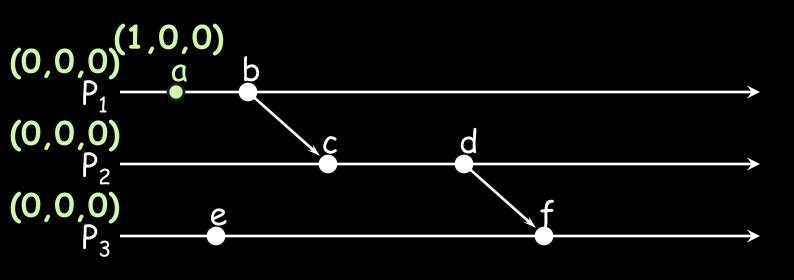
Just like Lamport's algorithm

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

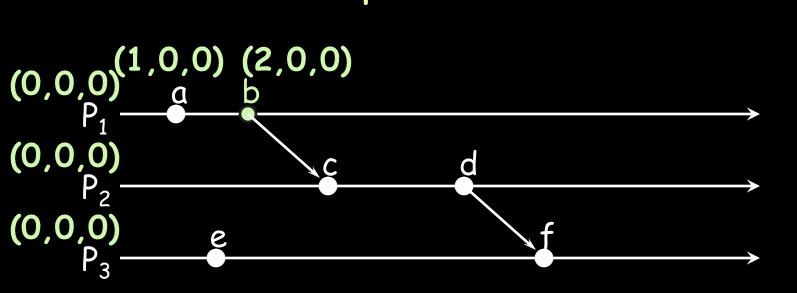
Two events are concurrent if neither

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

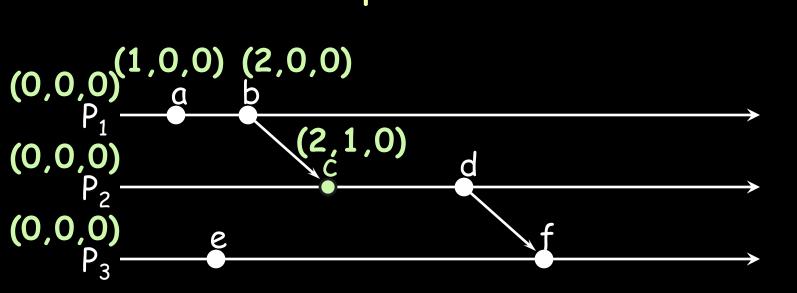




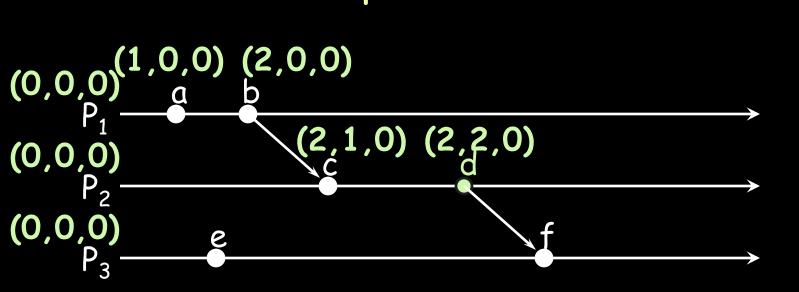
Event	timestamp
α	(1,0,0)



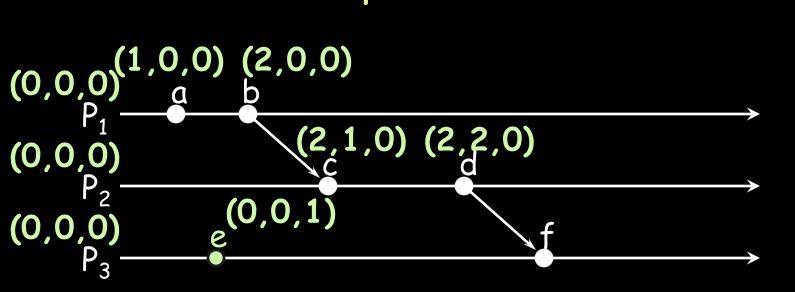
Event	timestamp	
a	(1,0,0)	
b	(2,0,0)	



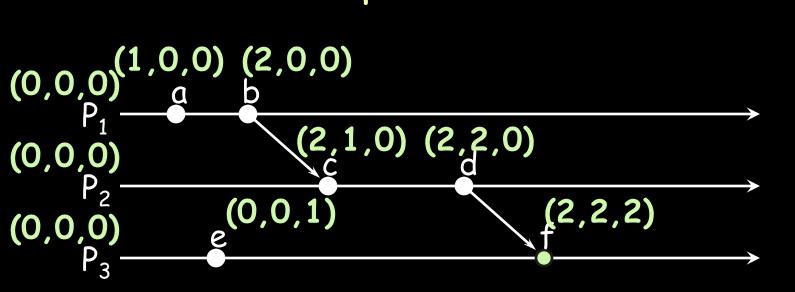
Event	timestamp
a	(1,0,0)
b	(2,0,0)
C	(2,1,0)



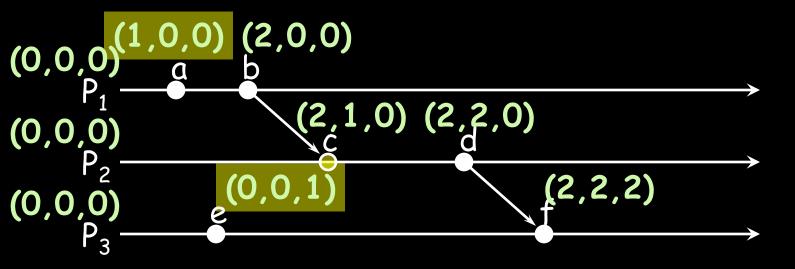
Event	timestamp
a	(1,0,0)
b	(2,0,0)
C	(2,1,0)
d	(2,2,0)

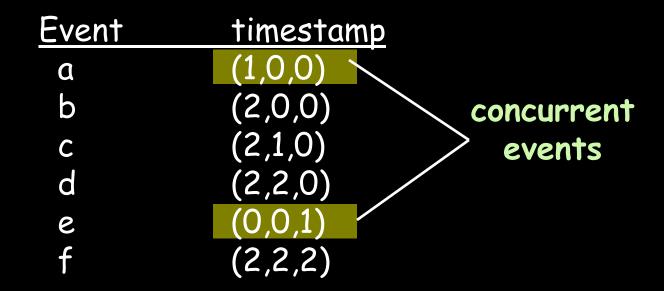


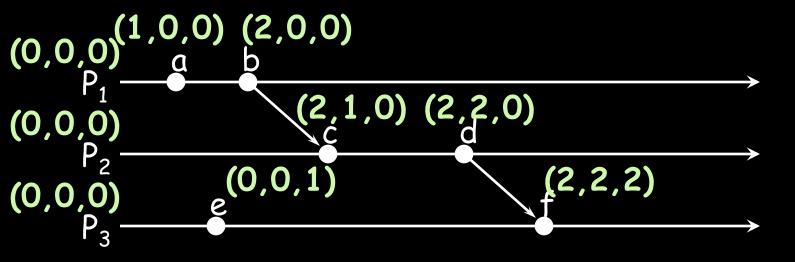
Event	timestamp
a	(1,0,0)
b	(2,0,0)
С	(2,1,0)
d	(2,2,0)
e	(0,0,1)

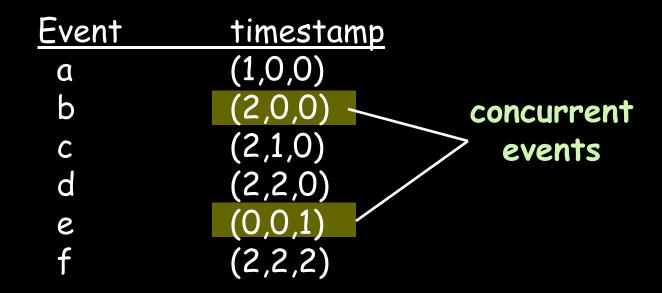


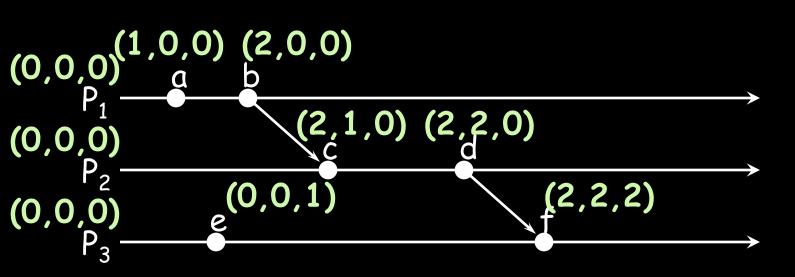
Event	timestamp
α	(1,0,0)
b	(2,0,0)
C	(2,1,0)
d	(2,2,0)
e	(0,0,1)
f	(2,2,2)



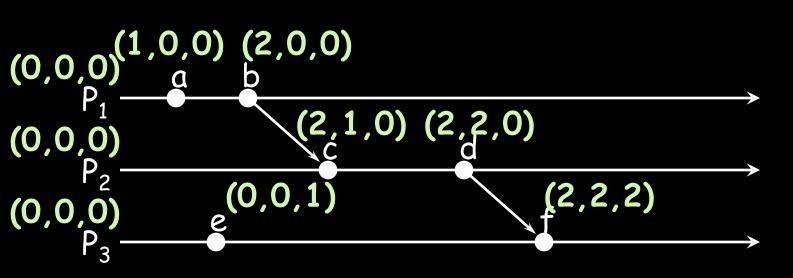








Event	timestamp	
a	(1,0,0)	
b	(2,0,0)	
C	(2,1,0) concurre	nt
d	(2,2,0) events	
e	(0,0,1)	
f	(2,2,2)	



Event	timestamp	
a	(1,0,0)	
b	(2,0,0)	
С	(2,1,0)	
d	(2,2,0)	concurrent
e	(0,0,1)	events
f	(2,2,2)	

# 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

# The end.