Distributed Systems

04r. Assignment 3 review

Paul Krzyzanowski

Rutgers University

Fall 2016

Question 1

How does Lamport define concurrent events?

Two events are concurrent if neither can causally affect the other.

Question 2

From the Why Vector Clocks are Easy paper, how can you tell if one vector clock is a descendent of another vector clock?

"In order for vector clock *B* to be considered a descendant of vector clock *A*, each marker in clock *A* must have a corresponding marker in clock *B* that has a revision number greater than or equal to the marker in *A*."

Marker = process ID; Revision # = sequence #
Vector clock = set of <process_id, sequence> tuples

{ <alice, 4>, <bob, 5>, <alic

Question 2 – examples

From the Why Vector Clocks are Easy paper, how can you tell if one vector clock is a descendent of another vector clock?

B = { <alice, 4>, <bob, 5>}

A = { <alice, 2> }

B is a descendent of A (A→B; A happened before B)

> because no element of A is greater the corresponding element of B
> A is missing ”bob”, so it is implicitly <bob, 0>

B = { <alice, 3>, <bob, 5>, <cindy, 2> }

A = { <alice, 2>, <bob, 4>, <cindy, 3> }

A & B are concurrent events (hence, a conflict). “alice” and “bob” have greater values in B but “cindy” has a smaller value.

Question 3

You have the following timestamps:

Client request sent:	7:12:10.100	$\leftarrow T_1$
Client receives response:	7:12.10.150	$\leftarrow T_4$
Server receives request:	7:11:59.900	$\leftarrow T_2$
Server sends response:	7:11:59.920	$\leftarrow T_3$

Time is expressed as hours:minutes:seconds.decimal_seconds
In the case of a client synchronizing with the server, A refers to the client and B refers to the server in the NTP RFC. Using NTP, what is the new time (add the offset, theta, to the *client receives response* time)?

Read section 8 of the NTP RFC (RFC 5905):

Offset, $\theta = ½ * ((T_2 – T_1) + (T_3 – T_4)) =$

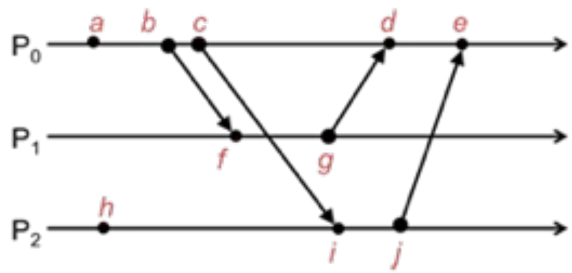
$= ½ * (11{:}59.900 – 12{:}10.100) + (11{:}59.920 – 12.10.150) =$

$= ½ * (-10.200) + (-10.230) = ½ * (-20.430) = -10.215$

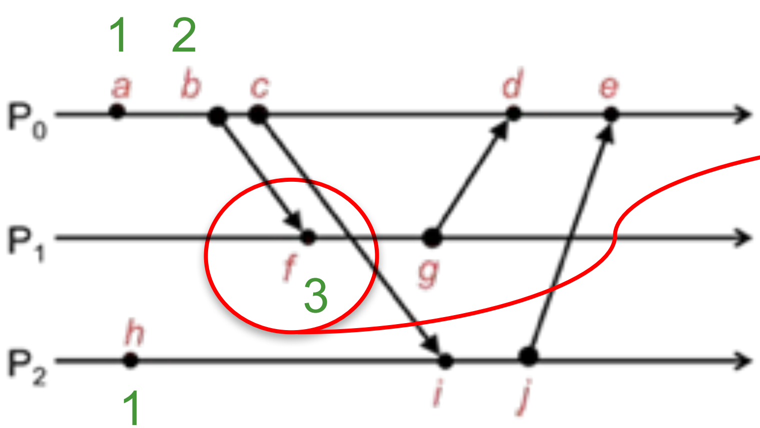
Time = 7:12:10.150 + -10.215 = **7:11:59.935**

Question 4

The table shows ten events *(a, b, ..., j)* taking place among three processes. Assign **Lamport timestamps** to each event.
The event clock on each process is initialized to zero at the beginning and incremented prior to timestamping each event. For instance, the clock on P_0 starts at 0 and event *a* gets assigned a Lamport timestamp of 1.

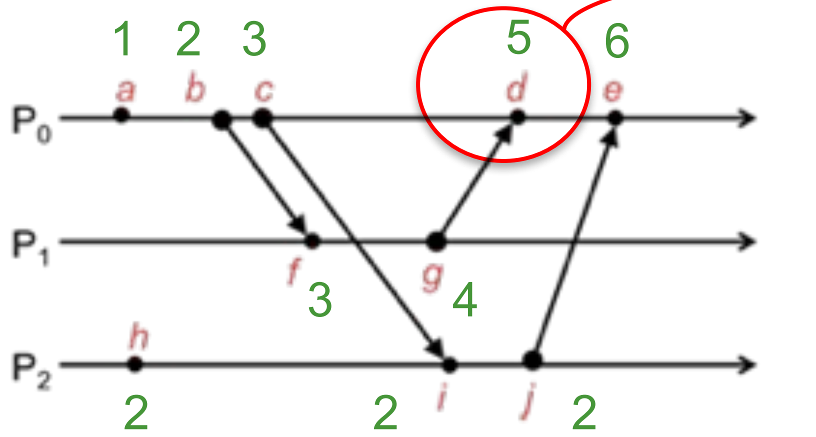


Lamport's "happens before" relationship:
a→*b* means "event *a* happens before b"



If *f* was an isolated event on P_1, it would get a Lamport timestamp = *1*
Since the event is the receipt of a message sent from P_0 with timestamp = 2, *f* has to be set to *max(2+1, 1)* = 3 to enforce the *b*→*f* relationship.

Question 4



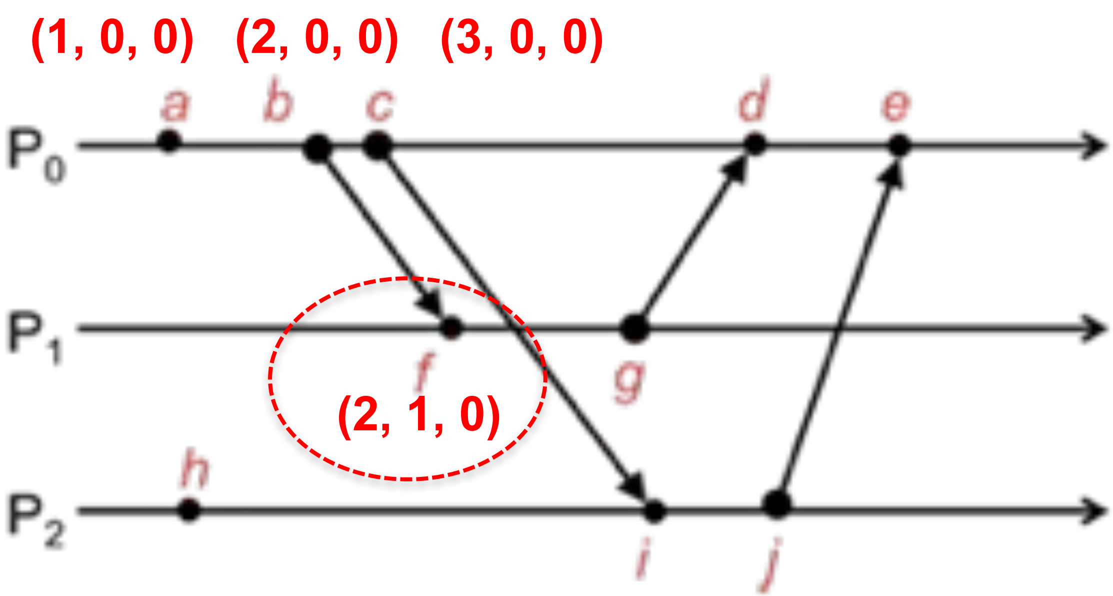
If *f* was an isolated event on P_0, it would get a Lamport timestamp = *4* (timestamp of event *c* + 1).

Since the event is the receipt of a message sent from P_1 with timestamp = 4, *d* has to be set to *max(4+1, 4)* = 5 to enforce the $g \rightarrow d$ relationship.

a. 1	b. 2	c. 3	d. 5	e. 6
f. 3	g. 4	h. 1	i. 4	j. 5

Question 4

Using the same set of events as in the previous question, assign vector timestamps to each event. The event clock vector at each process is initialized to all zeros at the beginning and a process increments its position in the vector prior to timestamping each event. Process positions in the vector are (P_0, P_1, P_2).



Increment per-process counter in the vector prior to each event.

For received messages with received vector r, new vector =

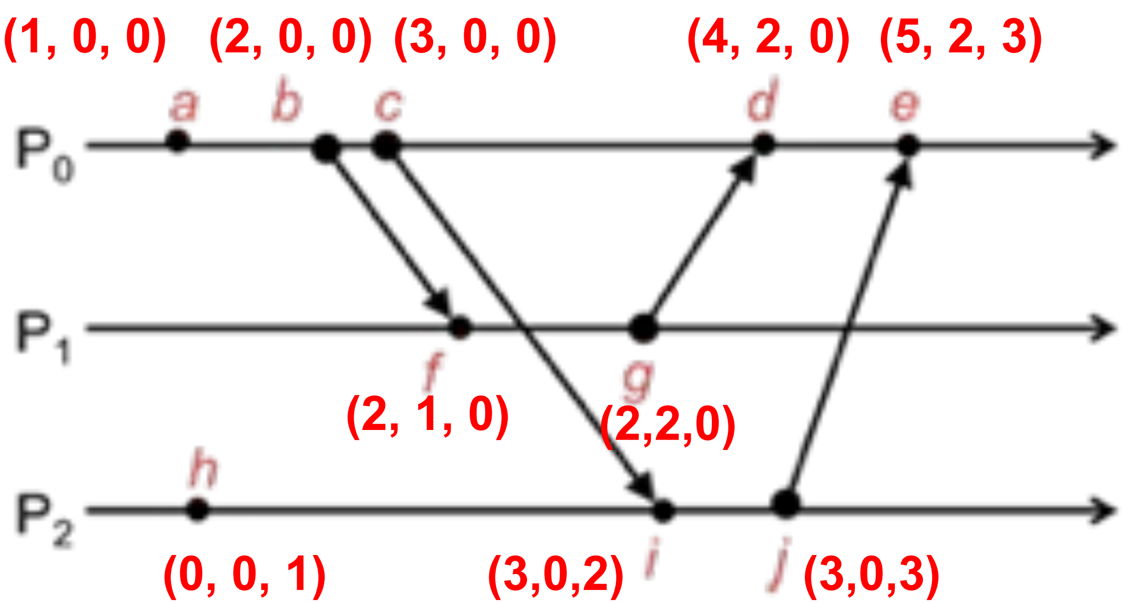
```
v[process_id]++;
for (i = 0; i < #elements; ++i)
    v[ i ] = max(v[ i ], r[ i ] )
```

Vector is (P_0, P_1, P_2)

Event *f* would have been (0,1,0) if it was isolated. Since it's the receipt of a message, We set the vector to (*max*(0,2), *max*(1,0), *max*(0,0)) = (2, 1, 0)

Question 4

Using the same set of events as in the previous question, assign vector timestamps to each event. The event clock vector at each process is initialized to all zeros at the beginning and a process increments its position in the vector prior to timestamping each event. Process positions in the vector are (P_0, P_1, P_2).



Increment per-process counter in the vector prior to each event.

For received messages with received vector r, new vector =

v[process_id]++;
for (i = 0; i < #elements; ++i)
  v[i] = *max*(v[i], r[i])

a. (1,0,0)	b. (2,0,0)	c. (3,0,0)	d. (4,2,0)	e. (5,2,3)
f. (2,1,0)	g. (2,2,0)	h. (0,0,1)	i. (3,0,2)	j. (3,0,3)

Question 5

Based on the vector timestamps, which events are causally dependent on event c (that is, which events follow c and are causally related)?

For two events to be causally dependent on each other, every element of one vector have to be ≥ the corresponding element of the other vector:

```
for (i=0; i<#elements; ++i)
    if ( a[i] < b[i]) smaller = 1
    if (a[i] > b[i]) larger = 1
if ((smaller == 1) && (larger == 1))
    concurrent
else
    causal
```

We need to find events that are > event *c*.

a. (1,0,0)	b. (2,0,0)	c. (3,0,0)	d. (4,2,0)	e. (5,2,3)
f. (2,1,0)	g. (2,2,0)	h. (0,0,1)	i. (3,0,2)	j. (3,0,3)

Question 6

Based on the vector timestamps, which events are causally dependent on event c (that is, which events follow c and are causally related)?

We need to find events that are > event *c*.
c = (3, 0, 0)

a. (1, 0, 0) < (3, 0, 0) – causal but *a* < *c*
b. (2, 0, 0) < (3, 0, 0) – causal but *b* < *c*
d. (4, 2, 0) > (3, 0, 0) – causal and > *c*
e. (5, 2, 3) > (3, 0, 0) – causal and *ed* > *c*
f. (2, 1, 0) $\nless$ (3, 0, 0) and (2, 1, 0) $\ngtr$ (3, 0, 0) – concurrent
g. (2, 2, 0) $\nless$ (3, 0, 0) and (2, 2, 0) $\ngtr$ (3, 0, 0) – concurrent
h. (0, 0, 1) $\nless$ (3, 0, 0) and (0, 0, 1) $\ngtr$ (3, 0, 0) – concurrent
i. (3, 0, 2) > (3, 0, 0) – causal and *i* > *c*
j. (3, 0, 3) > (3, 0, 0) – causal and *i* > *c*

The set of events that causally follow c are:
d, e, i, j

a. (1,0,0)	b. (2,0,0)	c. (3,0,0)	d. (4,2,0)	e. (5,2,3)
f. (2,1,0)	g. (2,2,0)	h. (0,0,1)	i. (3,0,2)	j. (3,0,3)

The End