Vector Clocks

Lec8

Problems with Lamport Clocks

- Lamport timestamps do not capture **causality**.
- With Lamport's clocks, one cannot directly compare the timestamps of two events to determine their precedence relationship.
- The main problem is that a simple integer clock cannot order both events within a process and events in different processes.
    - Knowing that C(a) < C(b) is true does not allow us to conclude that $a \rightarrow b$ is true.

Vector Clocks

- Happened-before relation is a partial order
  - But the domain of Lamport clocks (natural numbers) is a total order (wrt <)
  - Do not provide complete information about the happened-before relation
- Key idea: assign vector timestamps to events
  - s.v refers to the vector timestamp assigned to state s
- For all s,t: s-->t IFF s.v < t.v
- Want the timestamps also to be partial order
- For two vectors x,y of dimension N:
  - x<y =

Properties of Vector Timestamps

- v_i [i] is the number of events that have occurred so far at P_i
- If v_i [j] = k then P_i knows that k events have occurred at P_j

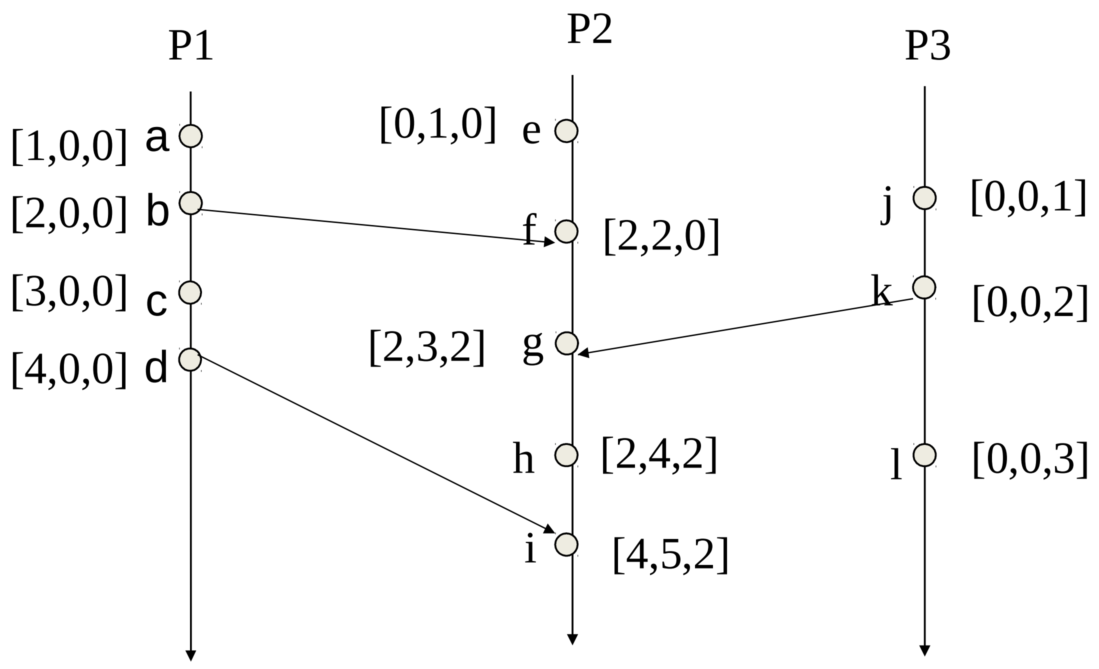
Vector Clock Update Rules

1. Initially all clock values are set to the smallest value (e.g., 0).
2. The local clock value is incremented at least once before each primitive event in a process i.e., $v_i[i] = v_i[i] + 1$
3. The current value of the entire logical clock vector is delivered to the receiver for every outgoing message.
4. Values in the timestamp vectors are never decremented.
5. Upon receiving a message, the receiver sets the value of each entry in its local timestamp vector to the maximum of the two corresponding values in the local vector and in the remote vector received.

- Let v_q be piggybacked on the message sent by process *q* to process p*;* We then have:

```
For i = 1 to num-processes {
    $v_p[i]$ = max($v_p[i]$, $v_q$ [i] )
}
$v_p[p] = v_p[p] + 1$;
```

Vector Clock Example



Vector Clock Proof

- s-->t IFF s.v < t.v

Lemma 1. *Let $s \neq t$, and $s \not\rightarrow t$. Then, $s.v[s.p] > t.v[s.p]$*

Proof. The more interesting case is when s and t occur on two different processes ($s.p \neq t.p$).

$s \not\rightarrow t$ means that there is no message path from s to t.
Process s.p's (say, Process-0) component

A local timestamp (s.v[s.p]) essentially counts local events. On receiving a message from s, t takes component wise max, and increments only its own local index.

Thus, for all other processes t: $s.v[s.p] \geq t.v[s.p]$.

Now, the only way for equality $s.v[s.p] = t.v[s.p]$, is if there is a message path from s to t. But we know that $s \not\rightarrow t$ and thus no message path exists.

Thus, we get strict inequality: $s.v[s.p] > t.v[s.p]$ □

Theorem 2. *$s \rightarrow t$ if and only if $s.v < t.v$*

Proof. Let's start with the forward direction:
If $\boldsymbol{s \rightarrow t}$, then $\boldsymbol{s.v < t.v}$

$s \rightarrow t$ means that there is some message path from s to t. If we apply the rules of vector clock updates, we get:

$\forall k$: $s.v[k] \leq t.v[k]$, because again rcpt involves pairwise maximums.
We thus get that $s.v \leq t.v$. But we want strict inequality.

But, we can apply Lemma 1.
$s \rightarrow t$ means that $t \not\rightarrow s$, which means: $t.v[t.p] > s.v[t.p]$
Thus there is one index for which strict inequality occurs, and thus we get $s.v < t.v$

Now the other direction: **If $\boldsymbol{s.v < t.v}$, then $\boldsymbol{s \rightarrow t}$**

We can prove this by contradiction and Lemma 1.
Suppose $s \not\rightarrow t$. Then we get $s.v[s.p] > t.v[s.p]$
But this contradicts $s.v < t.v$

$\square$

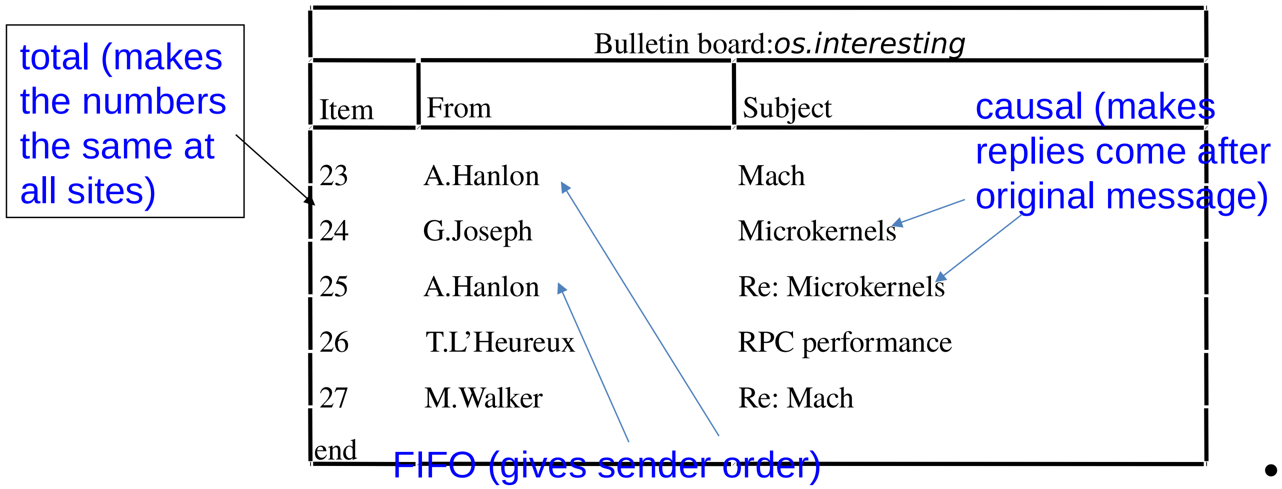
Causal Order Broadcast

Vector Timestamps and Causality

- We have looked at *total order* of messages where all messages are processed in the same order at each process.
- It is possible to have *any order* where all you care about is that a message reaches all processes, but you don’t care about the order of execution.
- *Causal order* is used when a message received by a process can potentially affect any subsequent message sent by that process. Those messages should be received in that order at all processes. Unrelated messages may be delivered in any order.

Display from a Bulletin Board Program

- Users run bulletin board applications which multicast messages
- One multicast group per topic (e.g. *os.interesting*)
- Require reliable multicast - so that all members receive messages
- Ordering:



Bulletin board:*os.interesting*

Item	From	Subject
23	A.Hanlon	Mach
24	G.Joseph	Microkernels
25	A.Hanlon	Re: Microkernels
26	T.L'Heureux	RPC performance
27	M.Walker	Re: Mach
end		

total (makes the numbers the same at all sites)

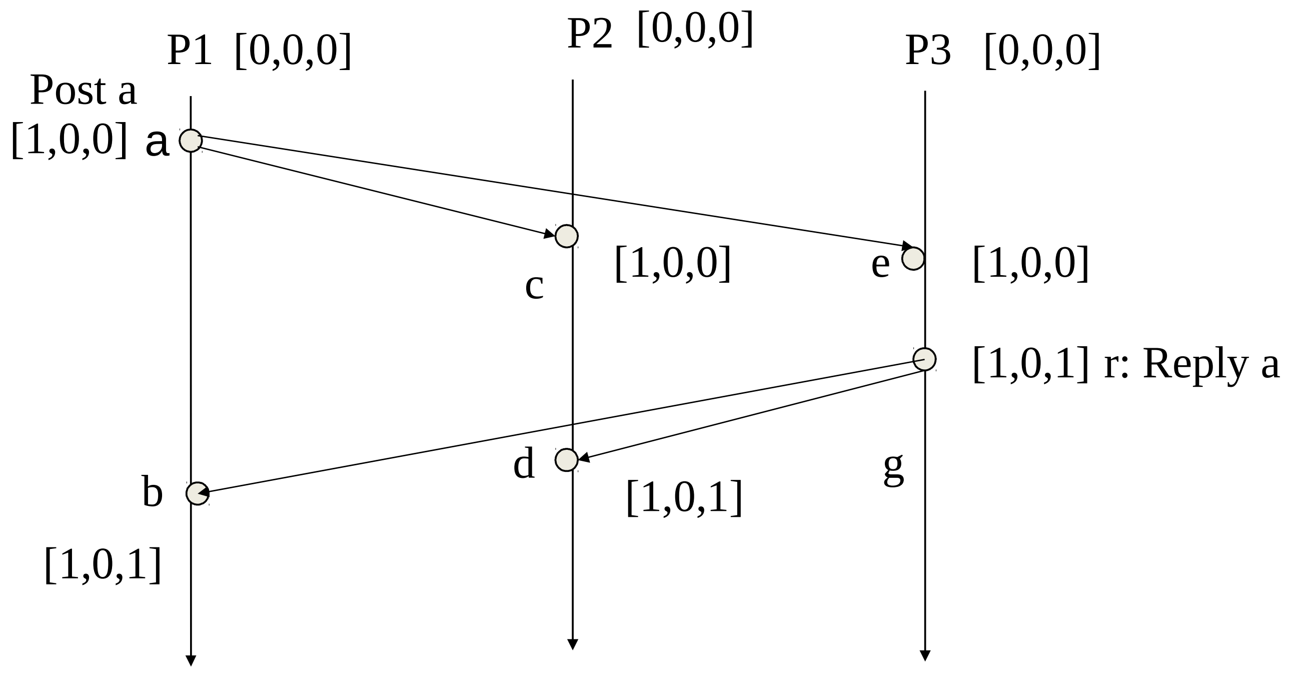
causal (makes replies come after original message)

FIFO (gives sender order)

Example Application: Bulletin Board

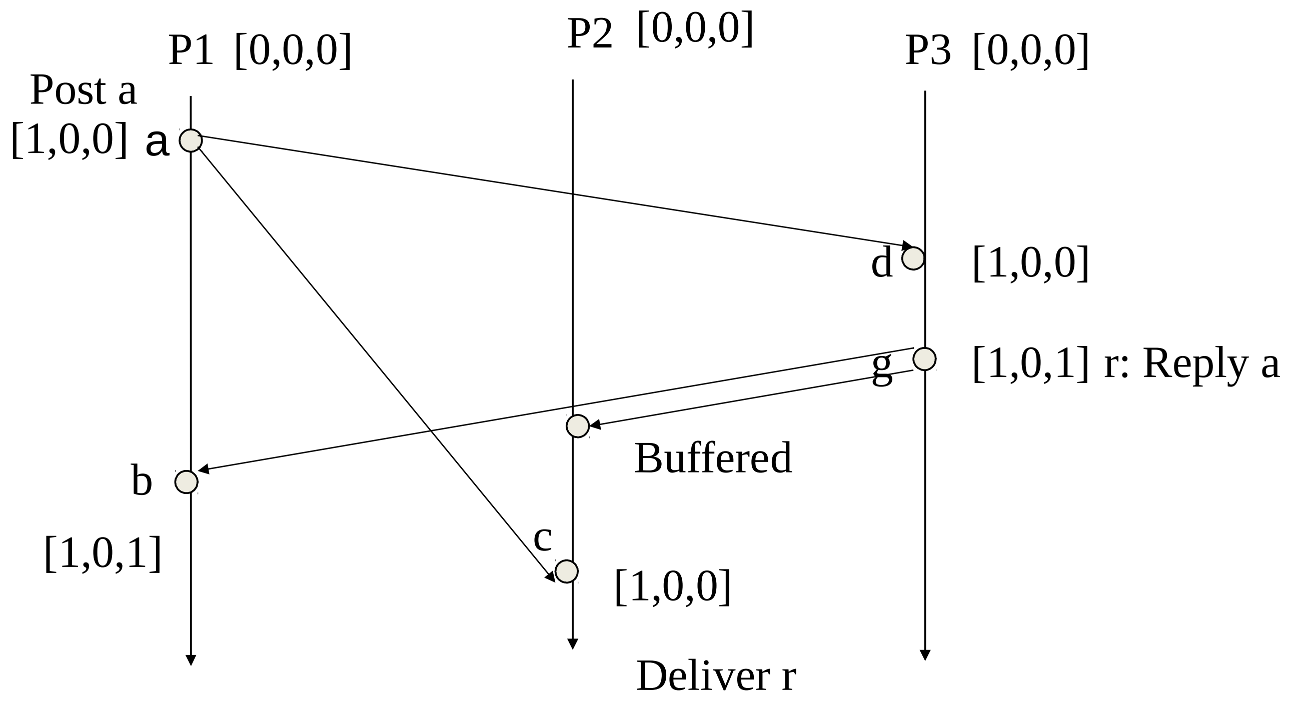
- A totally-ordered multicasting scheme does not imply that if message B is delivered after message A, that B is a reaction to A.
- Totally-ordered multicasting is too strong in this case.
- The receipt of an article causally precedes the posting of a reaction. The receipt of the reaction to an article should always follow the receipt of the article.

Example Application: Bulletin Board



Message *a* arrives at P2 before the reply *r* from P3 does

Example Application: Bulletin Board



The message *a* arrives at P2 after the reply from P3; The reply is not delivered right away.

END