

Logical Time 2



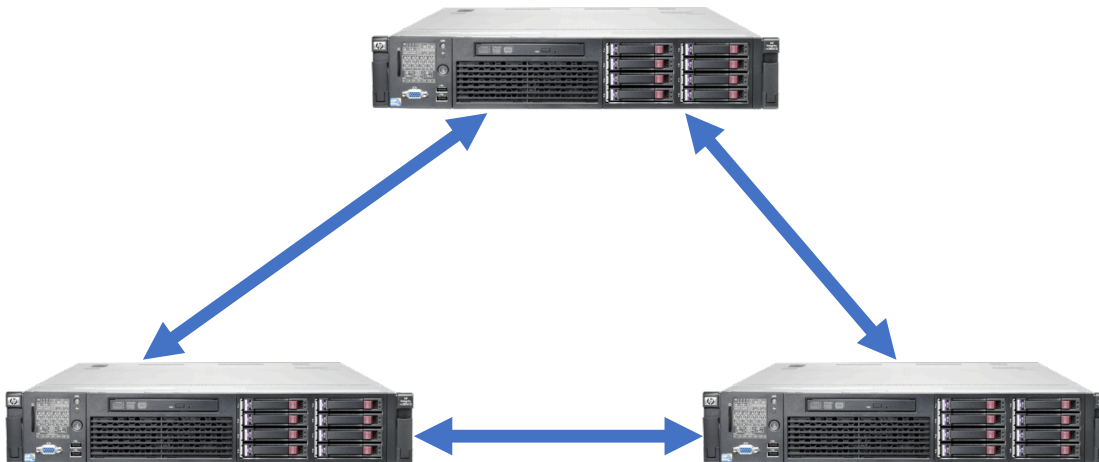
COS 316: Principles of Computer System Design
Lecture 15

Amit Levy & Wyatt Lloyd

Concurrency

- Multiple things happening at the same time
- Primary benefit is better performance
 - Do more work in the same amount of time
 - Complete fixed amount work in less time
 - Better utilize resources
- Primary cost is complexity
 - Hard to reason about
 - Hard to get right
 - (Systems deal with it, not applications, ... to some extent)

Distributed Systems, What?

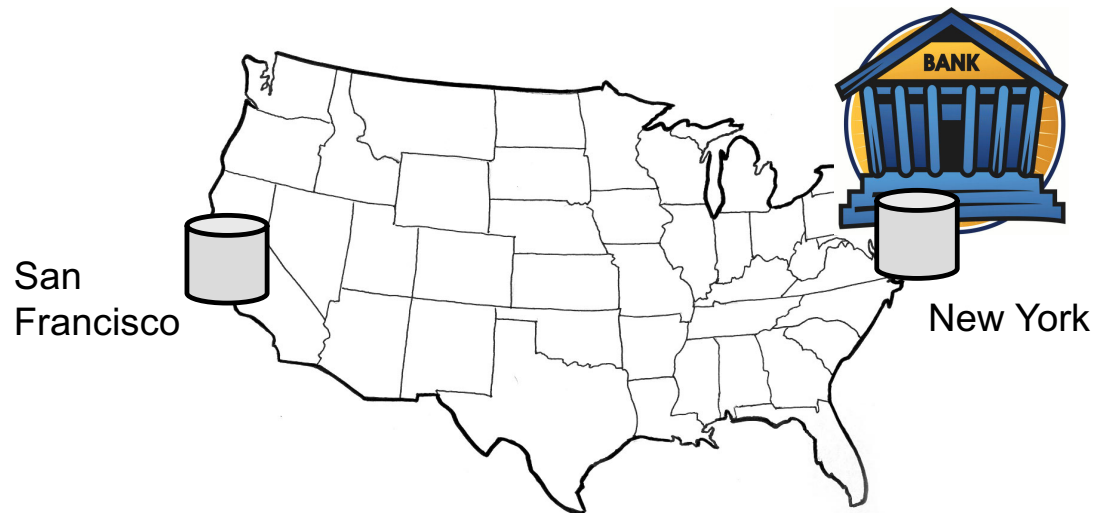


- 1) Multiple computers
- 2) Connected by a network
- 3) Doing something together

Concurrency is Inevitable!

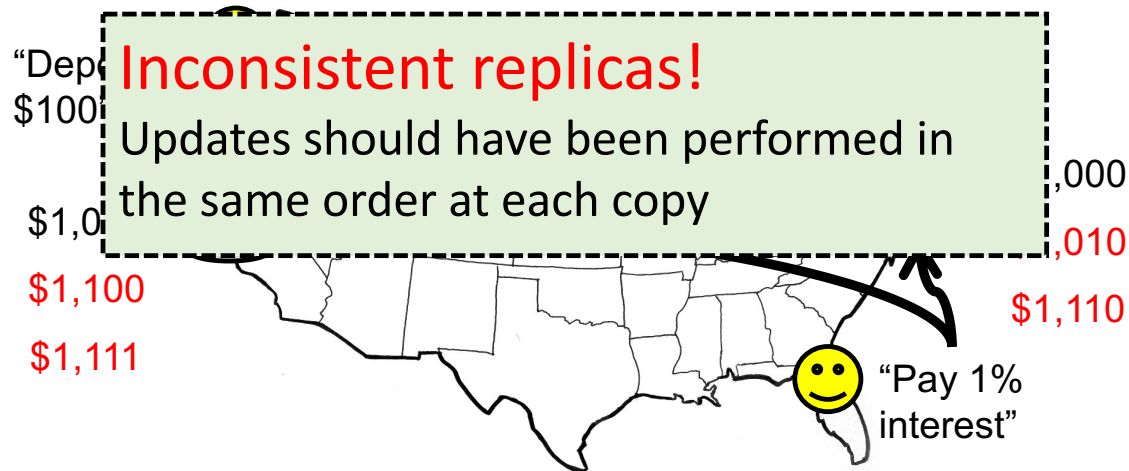
Motivation: Multi-site database replication

- A New York-based bank wants to make its transaction ledger database resilient to whole-site failures
- **Replicate** the database, keep one copy in sf, one in nyc



The consequences of concurrent updates

- **Replicate** the database, keep one copy in sf, one in nyc
 - Client sends query to the nearest copy
 - Client sends update to both copies



Lamport Timestamps: Ordering all events

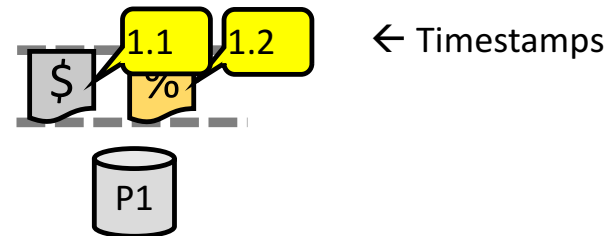
- **Break ties** by appending the process number to each event:
 1. Process P_i timestamps event e with $C_i(e).i$
 2. $C(a).i < C(b).j$ when:
 - $C(a) < C(b)$, **or** $C(a) = C(b)$ and $i < j$
- Now, for any two events a and b , $C(a) < C(b)$ or $C(b) < C(a)$
 - This is called a total ordering of events

Totally-Ordered Multicast

Goal: All sites apply updates in (same) Lamport clock order

- Client sends update to one replica site j
 - Replica assigns it Lamport timestamp $C_j . j$
- Key idea: Place events into a sorted **local queue**
 - **Sorted** by increasing Lamport timestamps

Example: P1's
local queue:



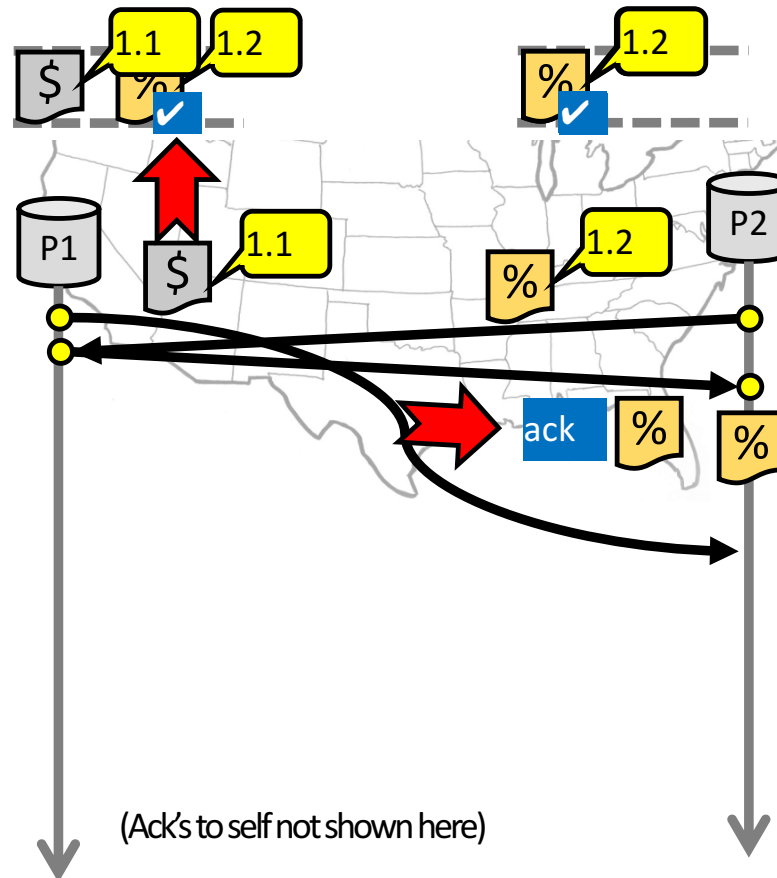
Totally-Ordered Multicast (Almost correct)

1. On receiving an update from client, broadcast to others (including yourself)
2. On receiving an update from replica:
 - a) Add it to your local queue
 - b) Broadcast an **acknowledgement message** to every replica (including yourself)
3. On receiving an acknowledgement:
 - Mark corresponding update **acknowledged** in your queue
4. **Remove and process** updates everyone has ack'ed from head of queue

Totally-Ordered Multicast (Almost correct)

- P1 queues \$, P2 queues %
- P1 queues and ack's %
 - P1 marks % fully ack'ed
- P2 marks % fully ack'ed

✗ P2 processes %



Totally-Ordered Multicast (Correct version)

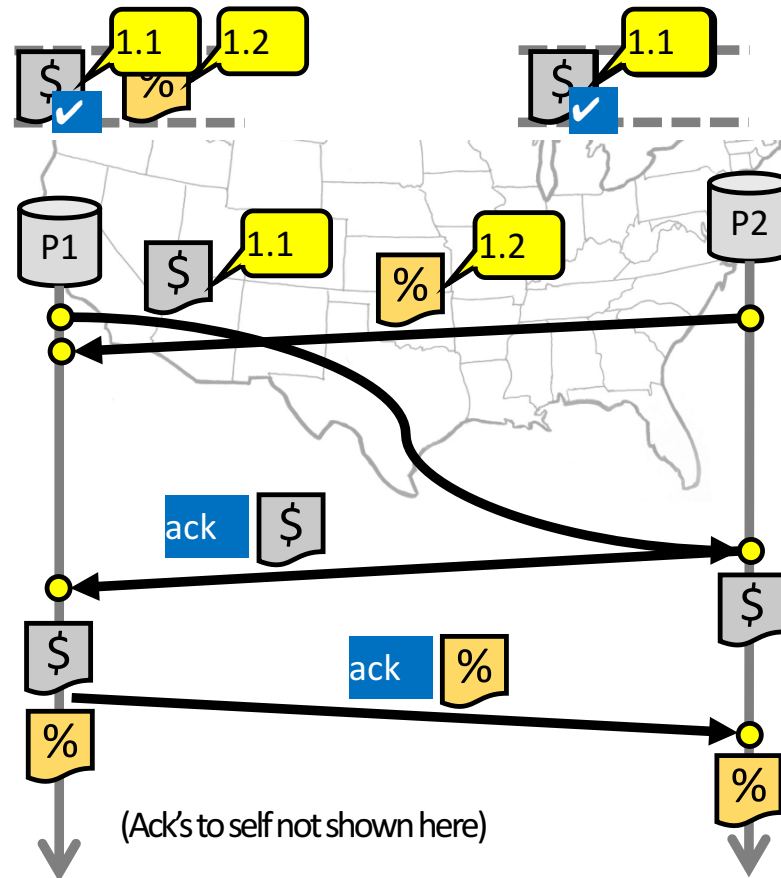
1. On receiving an update from client, broadcast to others (including yourself)

2. On receiving or processing an update:
 - a) Add it to your local queue, if received update
 - b) Broadcast an **acknowledgement message** to every replica (including yourself) **only from head of queue**

3. On receiving an acknowledgement:
 - Mark corresponding update **acknowledged** in your queue

4. **Remove and process** updates everyone has ack'ed from head of queue

Totally-Ordered Multicast (Correct version)



So, are we done?

- *Does totally-ordered multicast solve the problem of multi-site replication in general?*
 - Not by a long shot!
1. Our protocol **assumed:**
 - No node failures
 - No message loss
 - No message corruption
 2. All to all communication **does not scale**
 3. **Waits forever** for message delays (performance?)

Lamport Clocks Review

Q: $a \rightarrow b$ \Rightarrow $LC(a) < LC(b)$

Q: $LC(a) < LC(b)$ \Rightarrow $b \not\rightarrow a$ ($a \rightarrow b$ or $a \parallel b$)

Q: $a \parallel b$ \Rightarrow nothing

Lamport Clocks and causality

- Lamport clock timestamps do not capture causality
- Given two timestamps $C(a)$ and $C(z)$, want to know whether there's a chain of events linking them:

$$a \rightarrow b \rightarrow \dots \rightarrow y \rightarrow z$$

Vector clock: Introduction

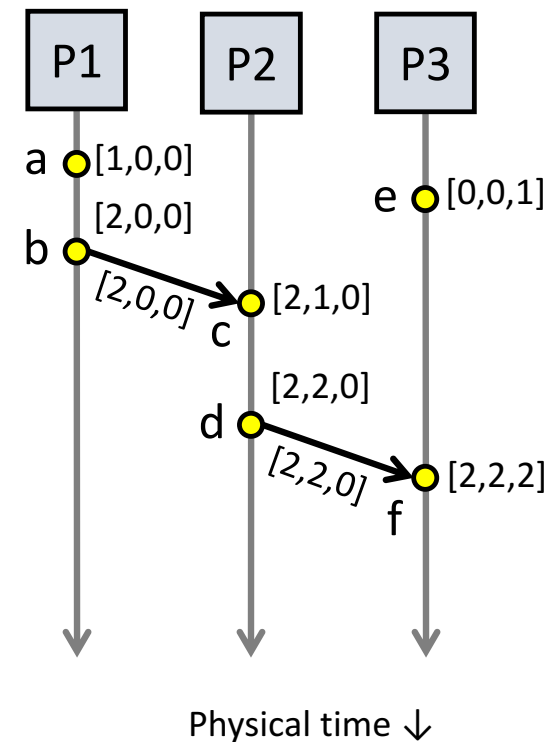
- One integer can't precisely order events in more than one process
- So, a **Vector Clock (VC)** is a vector of integers, one entry for each process in the entire distributed system
 - Label event e with $VC(e) = [c_1, c_2, \dots, c_n]$
 - Each entry c_k is a count of events in process k that causally precede e

Vector clock: Update rules

- Initially, all vectors are $[0, 0, \dots, 0]$
- Two update rules:
 1. For each local event on process i , increment local entry c_i
 2. If process j receives message with vector $[d_1, d_2, \dots, d_n]$:
 - Set each local entry $c_k = \max\{c_k, d_k\}$
 - Increment local entry c_j

Vector clock: Example

- All processes' VCs start at $[0, 0, 0]$
- Applying local update rule
- Applying message rule
 - Local vector clock piggybacks on inter-process messages

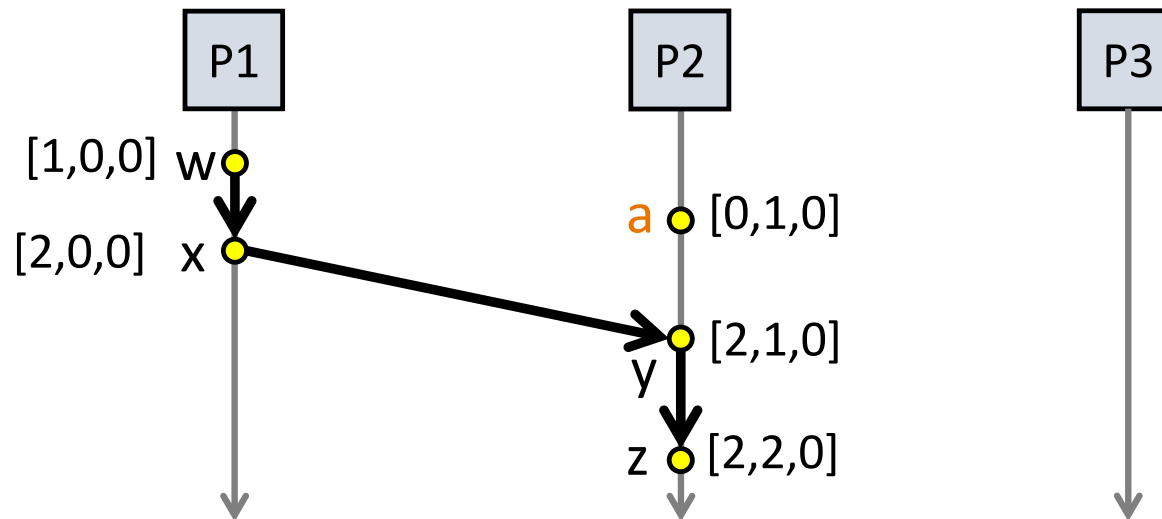


Comparing vector timestamps

- Rule for comparing vector timestamps:
 - $V(a) = V(b)$ when $a_k = b_k$ for all k
 - $V(a) < V(b)$ when $a_k \leq b_k$ for all k and $V(a) \neq V(b)$
- Concurrency:
 - $a \parallel b$ if $a_i < b_i$ and $a_j > b_j$, some i, j

Vector clocks capture causality

- $V(w) < V(z)$ then there is a chain of events linked by Happens-Before (\rightarrow) between w and z
- $V(a) \parallel V(w)$ then there is **no** such chain of events between a and w



Two events a, z

Lamport clocks: $C(a) < C(z)$

Conclusion: $z \not\rightarrow a$, i.e., either $a \rightarrow z$ or $a \parallel z$

Vector clocks: $V(a) < V(z)$

Conclusion: $a \rightarrow z$

Vector clock timestamps precisely capture happens-before relation (potential causality)

Motivation: Distributed discussion board

A screenshot of a discussion board interface. It lists four items, each with a question mark icon, a title, a category, an author, and a timestamp. Yellow arrows with the text 'OK' point to the right of each item, indicating a confirmation or approval action.

- Updating item in FIFO cache
Assignment anonymous 10d 2 1
- Precepts posted
Precepts Teague Tomesh 5d 1
- Office hours on Oct
Problem Sets Anonymous 11d 1
- A3 Hit and Miss
Assignments - A3 Anonymous 12d 1

Primary key auto incrementing

A screenshot of a discussion thread titled "Primary key auto incrementing". The thread shows a question by Oliver Schwartz and two answers. Yellow arrows with the text 'HB' point to the right of each comment, indicating a 'Helpful' or 'Bookmark' action.

Oliver Schwartz 2 days ago in Assignments - A4 58
PIN STAR WATCH VIEWS

Could a TA please elaborate on what this means:

```
Optionally, at most one of the fields of the provided 'model'  
// might be annotated with the tag `gorm:"primary_key"``
```

By annotated, it means one of the fields of the provided struct will be `PrimaryKey`. If this is not the case, the 'annotation' and how can we access this?

Comment 1

1 Answer

Yue Tan TA a day ago

The annotation means using golang's [field tag](#).

Comment 1

Add comment

Oliver Schwartz a day ago

Thanks. Just a clarifying question: for a struct with a `primary_key` annotation, should we have a value associated with `primary_key` in the tag to be just some empty string?

Reply Edit Delete

Jeff Helt TA 11 hours ago

The [README](#) shows an example Post struct and corresponding database schema. In the schema, the `id` is the `primary_key` and its associated type.

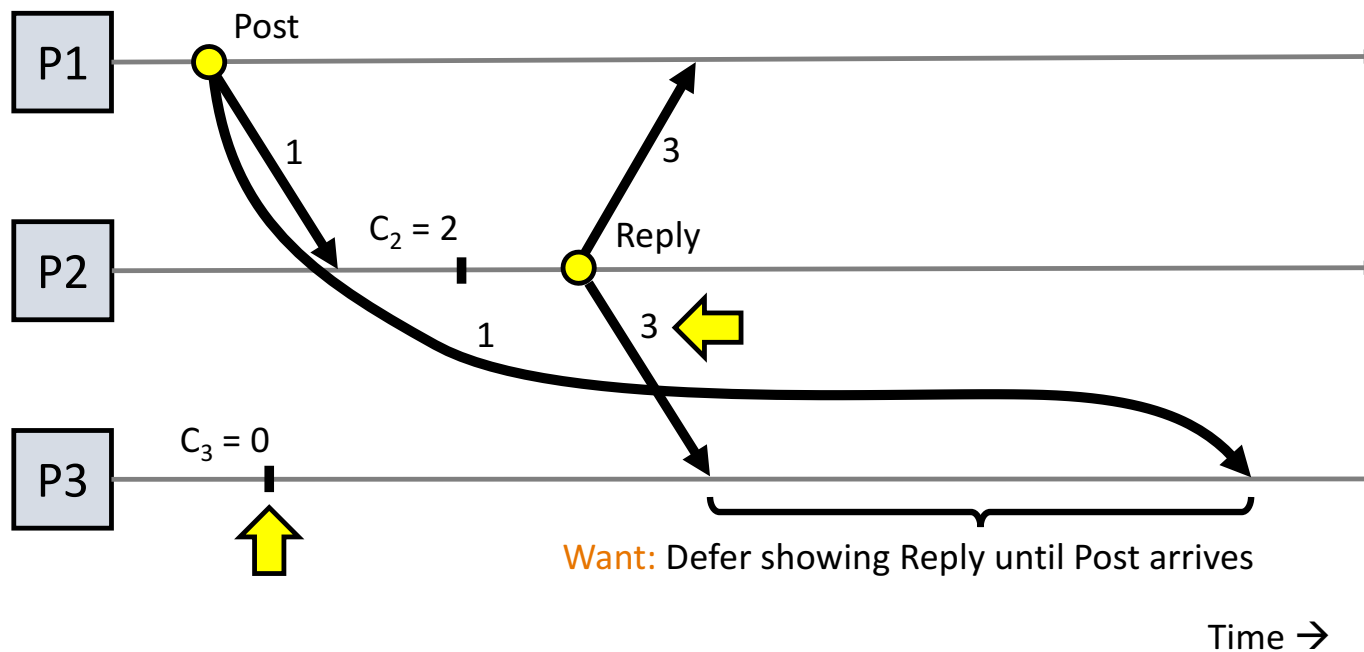
Reply Edit Delete

Distributed discussion board

- Users join specific discussion groups
 - Each user runs a process on a different machine
 - Messages (posts or replies) sent to all users in group
- Goal: Ensure replies follow posts
- Non-goal: Sort posts and replies chronologically
- Q: Can Lamport Clocks help here?

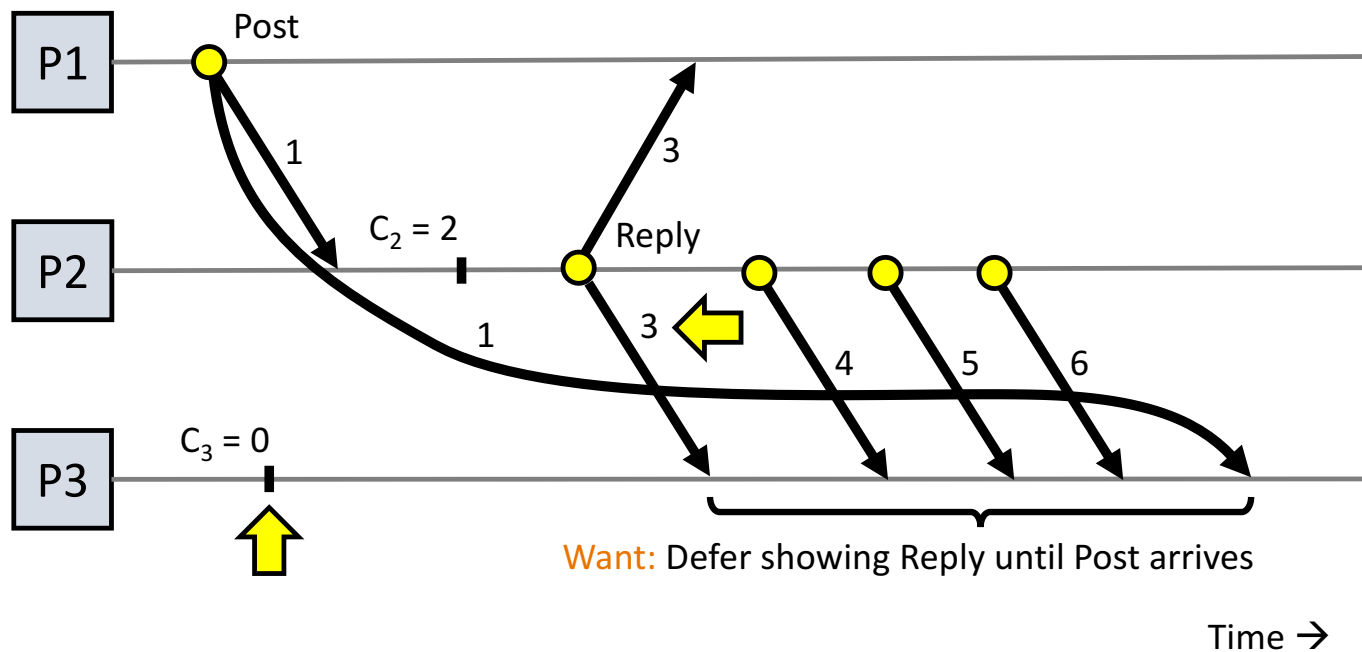


Lamport Clock-based discussion board



Proposal 1 : Defer showing message if $C(\text{message}) > \text{local clock} + 1$?

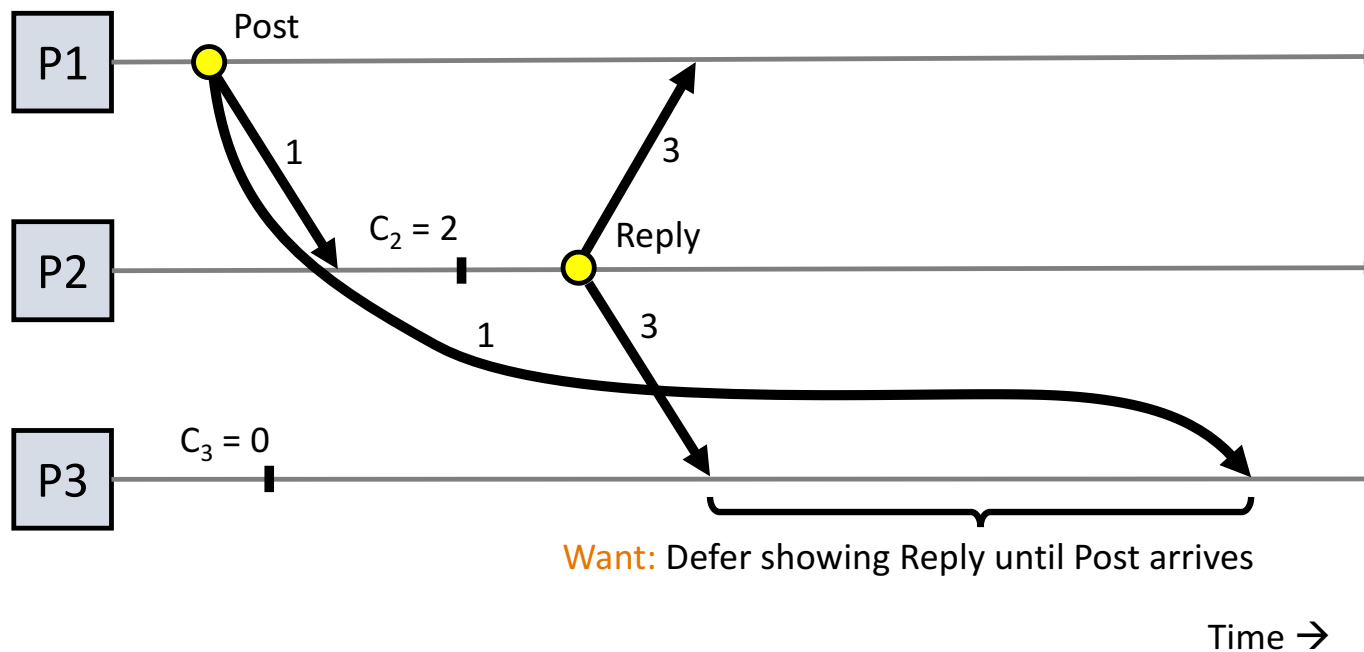
Lamport Clock-based discussion board



Proposal 1 : Defer showing message if $C(\text{message}) > \text{local clock} + 1$?

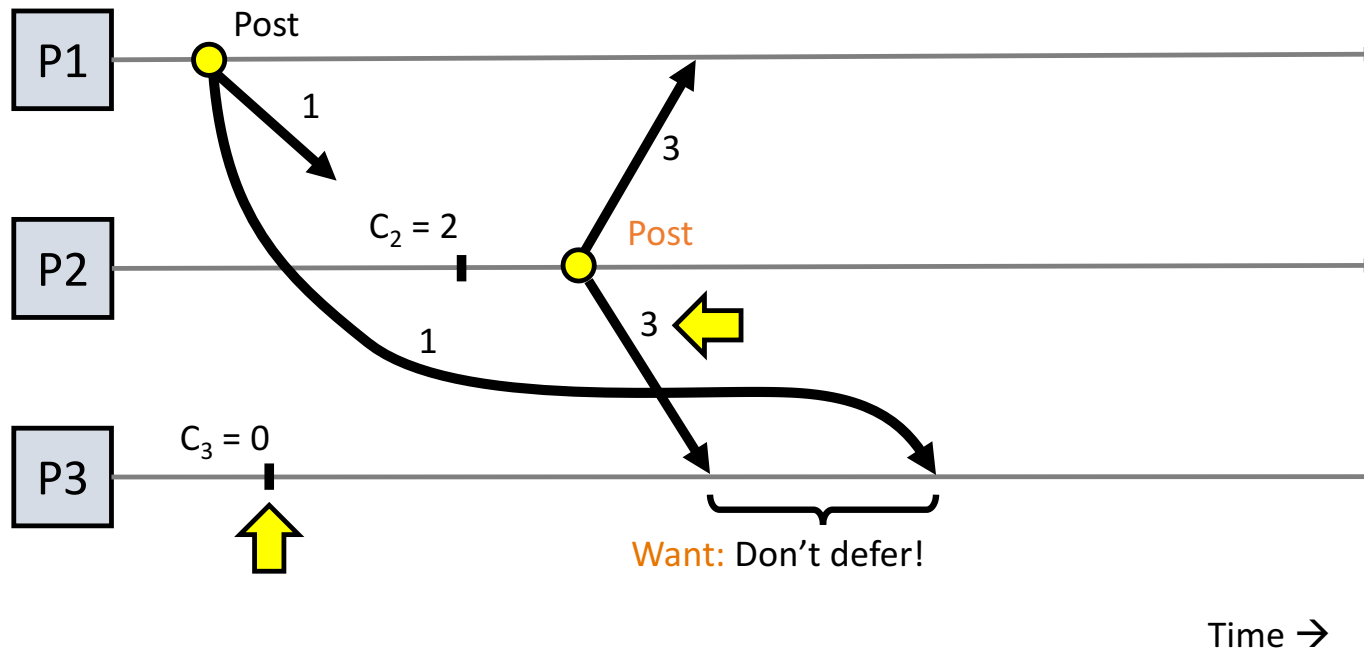
No! Local clock can be advanced by independent messages

Lamport Clock-based discussion board



Proposal 2: Use totally ordered multicast?

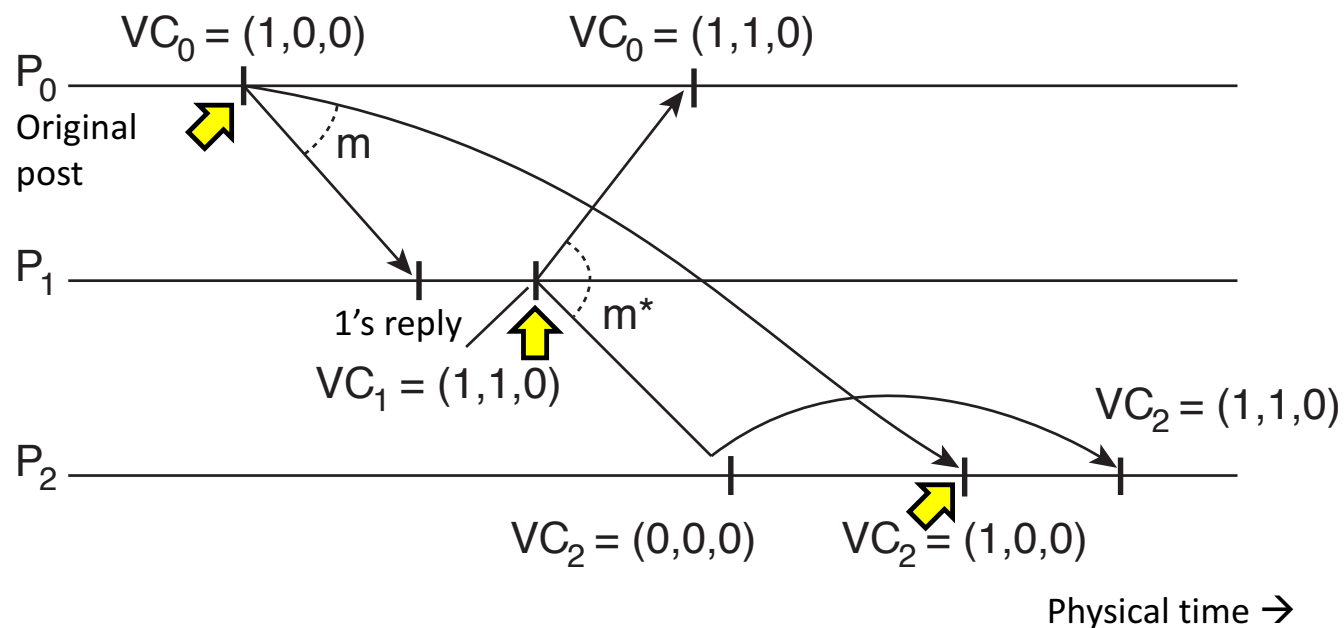
Lamport Clock-based discussion board



Proposal 2: Use totally ordered multicast?

No! It's quite slow & gap could be due to other independent posts

VC application: Causally-ordered discussion board

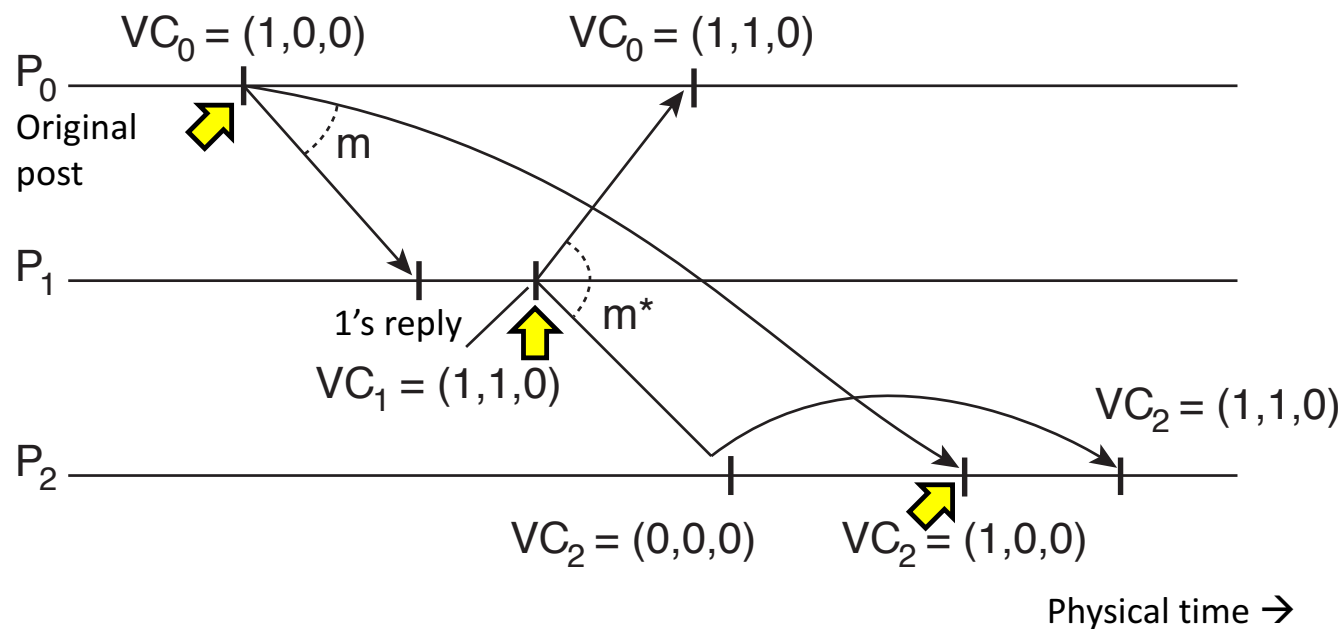


Proposal 3: Defer showing message if $C(\text{message}) > \text{local clock} + 1$?

Making VC-based discussion board work?

- Delay exposing updates until you've applied all causally previous updates
- 1) Use a TCP connection between each process

VC application: Causally-ordered discussion board



User 0 posts, user 1 replies to 0's post; user 2 observes

Logical Time Day 2 Conclusion

- Lamport clocks agree with happens-before
 - Easily extended to a total order
- Totally ordered multicast used lamport clocks!
 - Lamport clocks + careful protocol = correct replication
- Vector clocks capture happens-before (causality)
- Causally ordered discussion board
 - Totally ordered multicast correct ... but loses performance (concurrency)
 - Vector clocks for precise causal ordering with more concurrency

