

CPSC 416 – Distributed Systems

Lecture 03 Time, ordering, and causality

UBC Vancouver

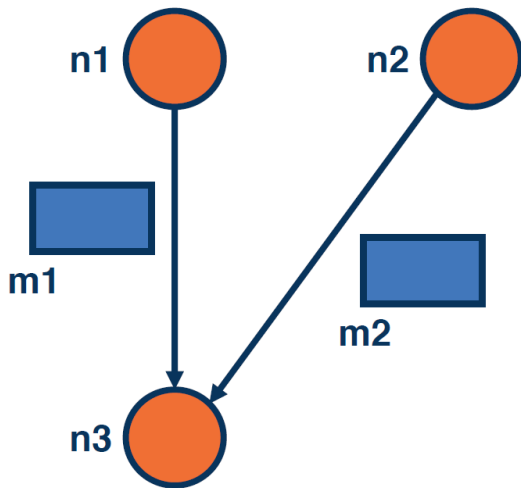
2026-01-13

Failure-first question

If two events are concurrent, what does it even mean to say one happened "before" the other?

Idea 1: Use the receiver's clock

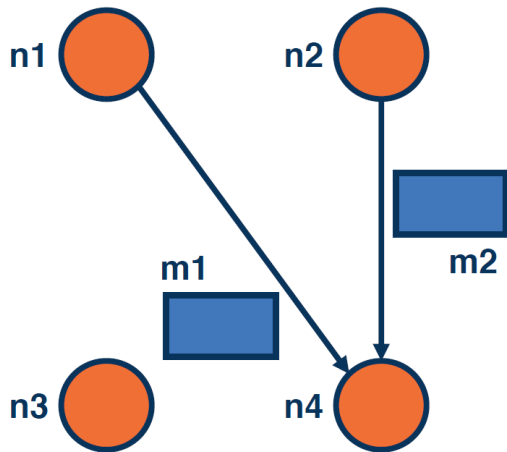
- N_1 sends message m_1
- N_2 sends message m_2
- N_3 receives both
- Order defined by when *receiver* gets each message



Problem: Network delays are not constant

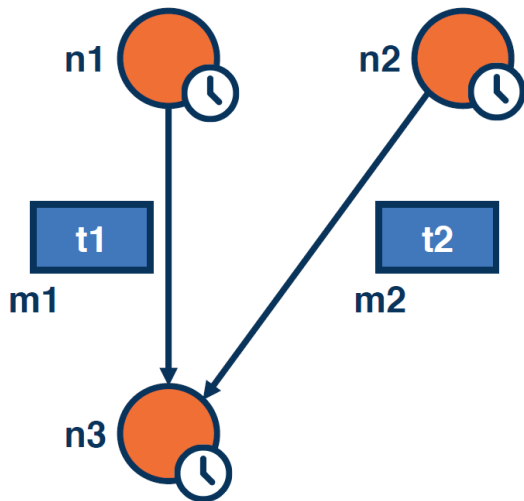
- m_1 sent *before* m_2
- But m_2 arrives *first*
- Receiver sees wrong order
- Lost/delayed messages make this worse

Verdict: Receiver's clock $\neq$ causal order



Idea 2: Use the sender's clock

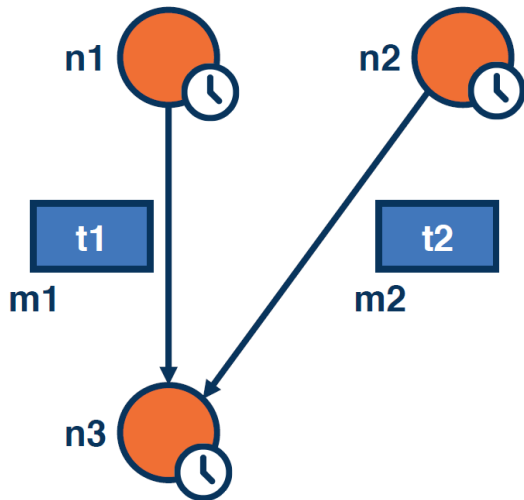
- Each node timestamps with its own clock
- Message m_i carries timestamp t_i
- Receiver compares timestamps to order events



Problem: Clocks are not synchronized

- N_1 's clock says 10:00:05
- N_2 's clock says 10:00:02
- N_2 sent *after* but has *earlier* timestamp
- Clock drift: oscillators are imperfect
- Even NTP can't fix this completely

Verdict: Sender timestamps $\neq$ causal order



Why physical time fails: Summary

- Clocks **drift** — oscillators are not perfect
- Synchronization protocols (NTP) help but aren't exact
- Network delays are **variable and unpredictable**
- No universal global clock (though Google Spanner tries...)

Key insight: Treat physical time as an *assumption*, not a primitive.

What if we could capture ordering *without* synchronized clocks?

Questions to consider

- Do nodes *need* to agree on global time?
- Is message delivery time fixed? Bounded?
- Can the network experience failures?
- Could there be malicious nodes?

Your answers to these questions determine what ordering guarantees are *possible*.

Alternative: Logical time

- Abandon physical clocks entirely
- **Logical time**: captures *causality*, not wall-clock time
- Properties:
 - ▶ Does not require synchronized clocks
 - ▶ Strictly increasing (monotonic)
 - ▶ Useful for reasoning about event ordering

The happens-before relation

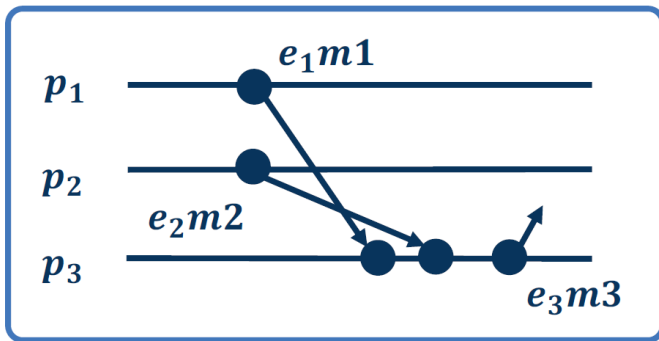
Lamport (1978) defined **happens-before** ($\rightarrow$):

- 1 If a and b are events in the same process, and a comes before b , then $a \rightarrow b$
- 2 If a is a send and b is the corresponding receive, then $a \rightarrow b$
- 3 Transitivity: if $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$

Concurrent ($\parallel$): $a \parallel b$ iff neither $a \rightarrow b$ nor $b \rightarrow a$

Concurrent events have *no causal relationship* — neither could have influenced the other.

Concurrent events: Example



- e_1 and e_2 are concurrent — neither caused the other
- $e_2 \rightarrow e_3$ (message delivery)
- $e_1 \rightarrow e_3$ (transitivity through the receives at p_3)

Lamport clocks: The rules

Each process p_i maintains a counter C_i :

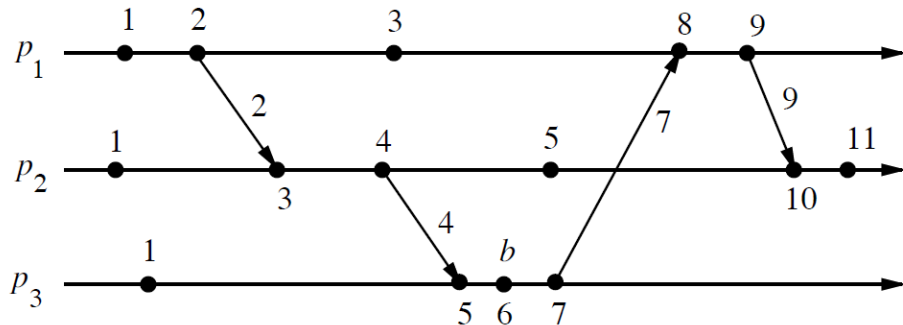
- 1 **Local event:** $C_i := C_i + 1$
- 2 **Send message:** increment C_i , attach C_i to message
- 3 **Receive message** with timestamp t : $C_i := \max(C_i, t) + 1$

Key property: If $a \rightarrow b$, then $C(a) < C(b)$

Limitation: The converse is *not* true!

$C(a) < C(b)$ does *not* imply $a \rightarrow b$

Lamport clocks: Worked example



Walk through: How does p_2 's clock jump from 1 to 3? (received timestamp 2, so $\max(1, 2) + 1 = 3$)

Lamport clocks: What they give you

Strengths:

- Simple — just one integer per process
- Provides a *total order* consistent with causality
- Good for: ordering events in logs, tie-breaking

Weakness:

- Cannot distinguish causality from concurrency
- If $C(a) < C(b)$, you don't know if $a \rightarrow b$ or $a \parallel b$

What if we need to detect concurrent events?

Vector clocks: The idea

Instead of one counter, each process keeps a **vector** of counters — one per process in the system.

Process p_i maintains $V_i[1..n]$ where:

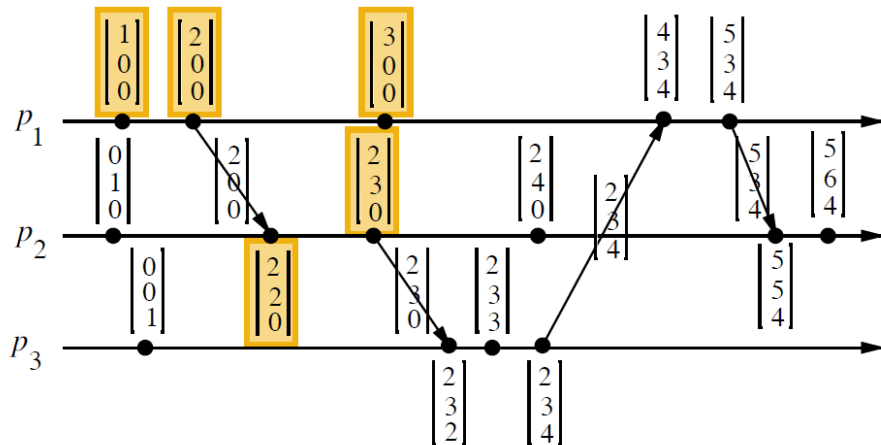
- $V_i[i] =$ number of events p_i has executed
- $V_i[j] =$ latest known event count from p_j

The vector captures “what I know about everyone’s progress.”

Vector clocks: The rules

- ① **Local event:** $V_i[i] := V_i[i] + 1$
- ② **Send message:** increment $V_i[i]$, attach entire vector V_i
- ③ **Receive message** with vector V_{msg} :
 - ▶ $V_i[j] := \max(V_i[j], V_{msg}[j])$ for all j
 - ▶ $V_i[i] := V_i[i] + 1$

Vector clocks: Worked example



Vector clocks: Comparing timestamps

Given vectors V_a and V_b :

- $V_a \leq V_b$ iff $V_a[i] \leq V_b[i]$ for all i
- $V_a < V_b$ iff $V_a \leq V_b$ and $V_a \neq V_b$

The key insight:

- $a \rightarrow b$ iff $V_a < V_b$
- $a \parallel b$ iff neither $V_a < V_b$ nor $V_b < V_a$

Example: $[2, 3, 0]$ vs $[1, 4, 1]$?

Neither dominates $\Rightarrow$ concurrent!

Vector clocks: Trade-offs

Strengths:

- Can detect concurrency (not just ordering)
- $V_a < V_b \Leftrightarrow a \rightarrow b$ (bidirectional!)

Weaknesses:

- Space: $O(n)$ integers per event (grows with system size)
- Every message carries the full vector
- Dynamic membership is tricky

For $n = 3$, no problem. For $n = 10,000$? Need alternatives.

Studio: Reconstruct causality (30 min)

You are debugging a distributed key-value store with 3 nodes (*A*, *B*, *C*). Each node logged events with **only physical timestamps** (wall clock).

The log:

Node	Time	Event	Details
A	10:00:01	PUT	key=x, val=1
B	10:00:02	RECV	from A, key=x, val=1
C	10:00:02	PUT	key=x, val=2
A	10:00:03	RECV	from C, key=x, val=2
B	10:00:04	RECV	from C, key=x, val=2
B	10:00:05	GET	key=x → returns ???

Studio: Questions to answer

Work in groups of 2–3. Answer these questions:

- ① What value should B 's GET return? **Justify your answer.**
- ② Are A 's PUT and C 's PUT concurrent, or is one causally before the other?
- ③ Can you tell from this log alone? What's missing?
- ④ **Redesign the log format:** What should each node record so you can always reconstruct causality?

Deliverable: Be ready to share your proposed log format with the class.

Studio: Discussion prompts

As you work, consider:

- If A 's clock is 1 second fast, does that change your answer?
- What if the network delayed $C \rightarrow B$ by 3 seconds?
- How would Lamport timestamps help here?
- How would vector clocks help here?
- What's the *minimum* metadata you need to attach to each message?

Logistics

- Readings + quiz (Week 02/03 on Canvas Readings page)
- Self-assessment opens at 13:00, due next class 11:30 (Pacific)
- DP1 design submission is now live in Canvas (Modules → Design Project 1)