

Time



COS 418/518: Distributed Systems
Lecture 4

Mike Freedman, Jialin Ding

1

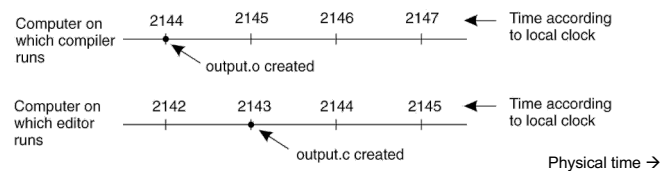
Today

1. The need for time synchronization
2. “Wall clock time” synchronization
3. Logical Time: Lamport Clocks

2

2

A distributed edit-compile workflow



- $2143 < 2144 \rightarrow$ make **doesn't call compiler**

Lack of time synchronization result
– a **possible object file mismatch**

3

3

What makes time synchronization hard?

1. When using local clock, your quartz oscillator is sensitive to temperature, age, vibration, radiation
 - Accuracy ~one part per million $\rightarrow$ ~1s clock drift over 12 days
2. When synchronizing from remote clock, the internet is:
 - Asynchronous: arbitrary message delays
 - Best-effort: messages don't always arrive

4

4

Today

1. The need for time synchronization
2. “Wall clock time” synchronization
 - Cristian’s algorithm, NTP
3. Logical Time: Lamport clocks

5

5

Just use Coordinated Universal Time?

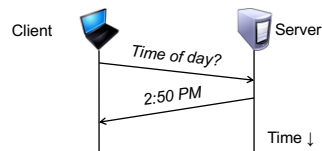
- UTC is broadcast from radio stations on land and GPS satellites
 - Computers with receivers can synchronize their clocks with these timing signals
- Signals from land-based stations are accurate to about 0.1–10 milliseconds
- Signals from GPS are accurate to about one microsecond
 - (Why don't we just put GPS receivers on all our computers?)

6

6

Synchronization to a time server

- Suppose a server with an accurate clock (e.g., GPS-receiver)
 - Could simply issue an RPC to obtain the time:



- But this doesn't account for network latency
 - Message delays will have **outdated** server's answer

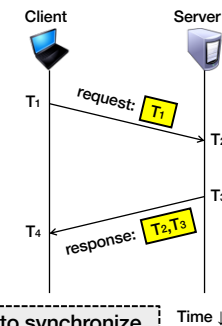
7

7

Cristian's algorithm: Outline

1. Client sends a **request** packet, timestamped with its local clock T_1
2. Server timestamps its receipt of the request T_2 with its local clock
3. Server sends a **response** packet with its local clock T_3 and T_2
4. Client locally timestamps its receipt of the server's response T_4

How can the client use these timestamps to synchronize its local clock to the server's local clock?



8

8

Cristian's algorithm: Offset sample calculation

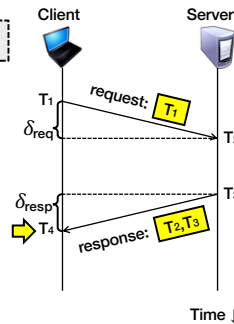
Goal: Client sets clock $\leftarrow T_3 + \delta_{\text{resp}}$

- Client samples round trip time (δ)
 $\delta = \delta_{\text{req}} + \delta_{\text{resp}} = (T_4 - T_1) - (T_3 - T_2)$

- But client knows δ , not δ_{resp}

Assume: $\delta_{\text{req}} \approx \delta_{\text{resp}}$

Client sets clock $\leftarrow T_3 + \frac{1}{2}\delta$



9

Clock synchronization: Take-away points

- Clocks on different systems will always behave differently
 - Disagreement between machines can result in undesirable behavior
- NTP clock synchronization
 - Rely on timestamps to estimate network delays
 - 100s μs –ms accuracy
 - Clocks never exactly synchronized
- Often **inadequate** for distributed systems
 - Often need to reason about the order of events

10

Today

- The need for time synchronization
- "Wall clock time" synchronization
 - Cristian's algorithm, NTP
- Logical Time: Lamport clocks

11

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



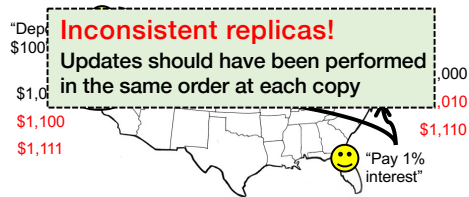
12

12

11

The consequences of concurrent updates

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



13

RFC 677 “The Maintenance of Duplicate Databases” (1975)

“To the extent that the communication paths can be made **reliable**, and the clocks used by the processes kept close to **synchrony**, the probability of seemingly strange behavior can be made very small.

However, **the distributed nature of the system dictates that this probability can never be zero.**”

14

Idea: Logical clocks

- Landmark 1978 paper by Leslie Lamport
- Insight: only the **events themselves** matter

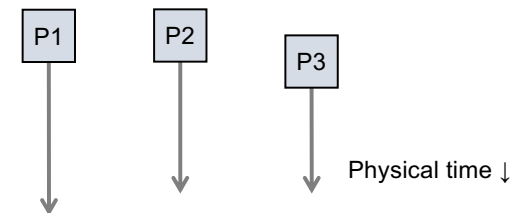


Idea: Disregard the precise clock time
Instead, capture **just** a “happens before” relationship between a pair of events

15

Defining “happens-before” ($\rightarrow$)

- Consider three processes: P1, P2, and P3
- Notation: Event **a** happens before event **b** ($a \rightarrow b$)

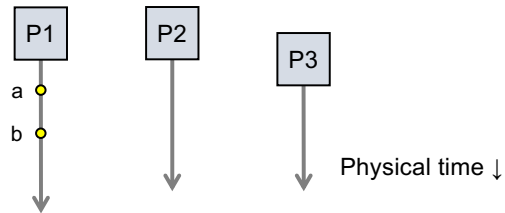


16

16

Defining “happens-before” ($\rightarrow$)

- Can observe event order at a single process

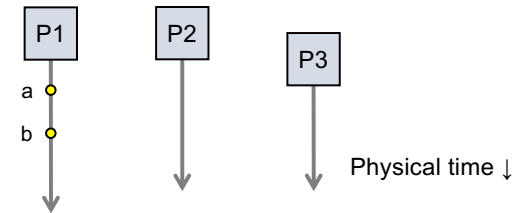


17

17

Defining “happens-before” ($\rightarrow$)

1. If **same process** and **a** occurs before **b**, then $a \rightarrow b$

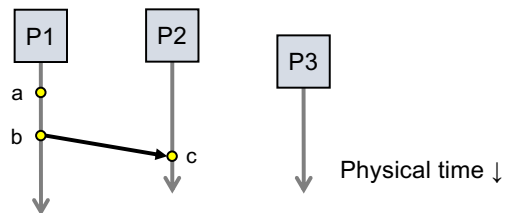


18

18

Defining “happens-before” ($\rightarrow$)

1. If **same process** and **a** occurs before **b**, then $a \rightarrow b$
2. Can observe ordering when processes communicate

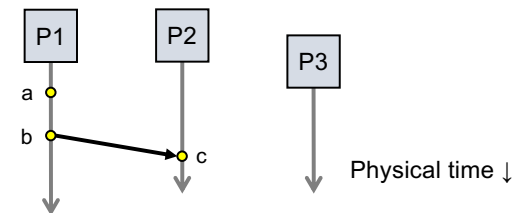


19

19

Defining “happens-before” ($\rightarrow$)

1. If **same process** and **a** occurs before **b**, then $a \rightarrow b$
2. If **c** is a message receipt of **b**, then $b \rightarrow c$

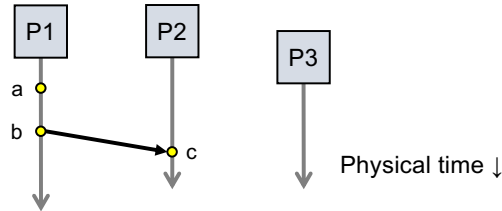


20

20

Defining “happens-before” ($\rightarrow$)

1. If **same process** and **a** occurs before **b**, then $a \rightarrow b$
2. If **c** is a message receipt of **b**, then $b \rightarrow c$
3. Can observe ordering transitively

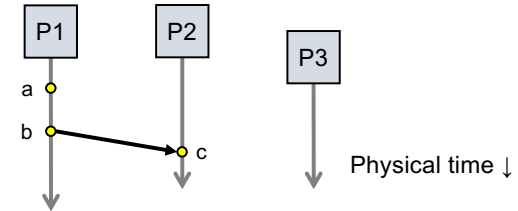


21

21

Defining “happens-before” ($\rightarrow$)

1. If **same process** and **a** occurs before **b**, then $a \rightarrow b$
2. If **c** is a message receipt of **b**, then $b \rightarrow c$
3. If $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$

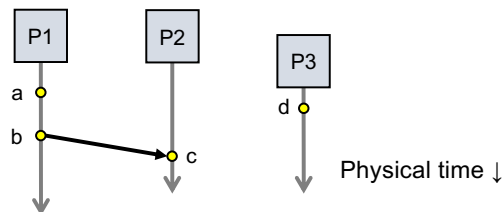


22

22

Concurrent events

- Not all events are related by $\rightarrow$
- a, d not related by $\rightarrow$ so **concurrent**, written as $a \parallel d$



23

23

Lamport clocks: Objective

- We seek a **clock time** $C(a)$ for every event **a**

Plan: Tag events with clock times; use clock times to make distributed system correct

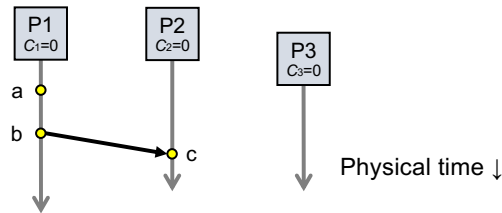
- Clock condition: If $a \rightarrow b$, then $C(a) < C(b)$

24

24

The Lamport Clock algorithm

- Each process P_i maintains a local clock C_i
- 1. Before executing an event, $C_i \leftarrow C_i + 1$

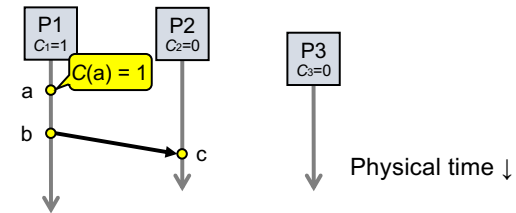


25

25

The Lamport Clock algorithm

- Each process P_i maintains a local clock C_i
- 1. Before executing an event, $C_i \leftarrow C_i + 1$
 - Set event time $C(a) \leftarrow C_i$

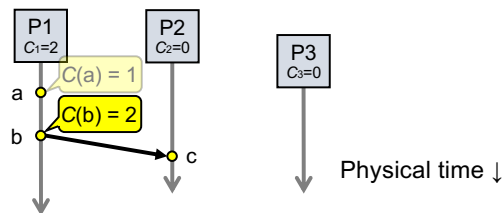


26

26

The Lamport Clock algorithm

- Each process P_i maintains a local clock C_i
- 1. Before executing an event, $C_i \leftarrow C_i + 1$
 - Set event time $C(b) \leftarrow C_i$

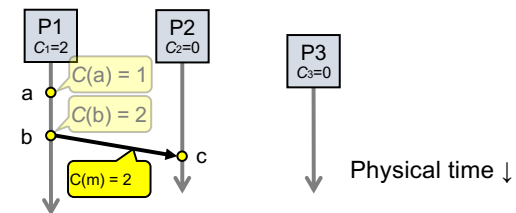


27

27

The Lamport Clock algorithm

- Each process P_i maintains a local clock C_i
- 1. Before executing an event, $C_i \leftarrow C_i + 1$
- 2. Send the local clock in the message m



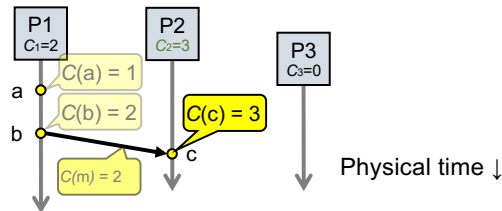
28

28

The Lamport Clock algorithm

3. On process P_j receiving a message m :

- Set C_j and receive event time $C(c) \leftarrow 1 + \max\{C_j, C(m)\}$



29

29

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 $i < j$

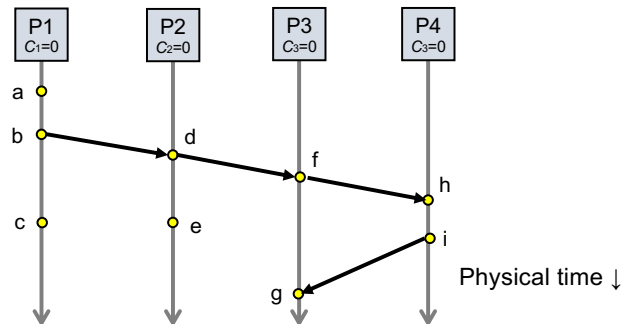
- Now, for any two events a and b , $C(a) < C(b)$ or $C(b) < C(a)$

- This is called a total ordering of events

30

30

Order all these events



31

Take-away points: Lamport clocks

- Can totally-order events in a distributed system: that's useful!

- We will see an application of Lamport clocks for totally-ordered multicast next time

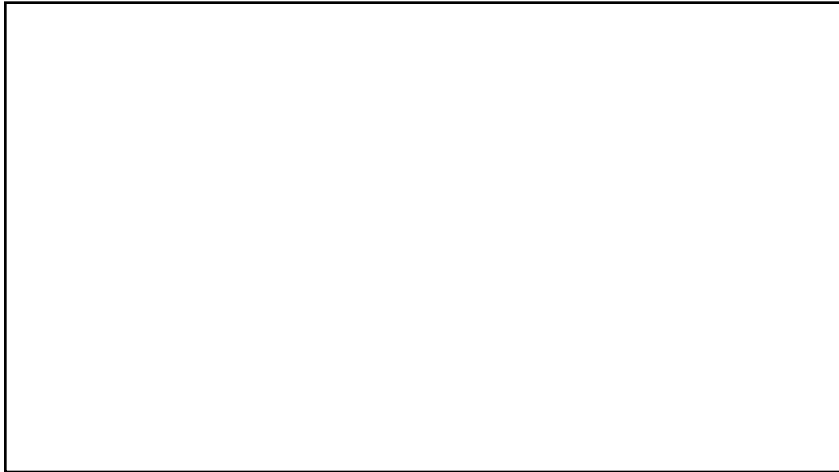
- But: while by construction, $a \rightarrow b$ implies $C(a) < C(b)$,

- The converse is not necessarily true: $C(a) < C(b)$ does not imply $a \rightarrow b$
 - $C(a) < C(b)$ can also occur when $a \parallel b$, provided that $a.i < b.j$
 - (But, if $C(a) < C(b)$, then $b \rightarrow a$ cannot be true)

Can't use Lamport timestamps to infer causal relationships between events

32

32



33