

Physical & logical time, clocks, ordering of events

Course: Real-Time Backend

Lecturer: Gleb Lobanov

June, 2024

Contents

01 Physical clocks. Leap seconds

02 Problems with Time

03 Clock synchronisation. NTP

04 Causality. Happens-before relation

05 Logical time. Lamport and vector clocks

05 Why do i need this?

01

Physical clocks. Leap seconds

Clocks and time in distributed systems

Distributed systems often need to measure time, e.g.:

- Schedulers, timeouts, failure detectors, retry timers
- Performance measurements, statistics, profiling
- Log files & databases: record when an event occurred
- Data with time-limited validity (e.g. cache entries)
- Determining order of events across several nodes

We distinguish two types of clock:

- **Physical clocks:** count number of seconds elapsed
- **Logical clocks:** count events, e.g. messages sent

Note: Clock in digital electronics (oscillator) \neq
 \neq clock in distributed systems (source of **timestamps**)

Time measurement

In 1583, [Galileo Galilei](#) (1564–1642) discovered that a [pendulum's harmonic motion](#) has a constant period, which he learned by timing the motion of a swaying lamp in [harmonic motion](#) at [mass](#) at the cathedral of [Pisa](#), with his [pulse](#).^[18]

Time measurement

Galileo's experimental setup to measure the literal *flow of time*, in order to describe the motion of a ball, preceded **Isaac Newton**'s statement in his *Principia*, "I do not define *time*, *space*, *place* and *motion*, as being well known to all."^[20]

The **Galilean transformations** assume that time is the same for all **reference frames**.

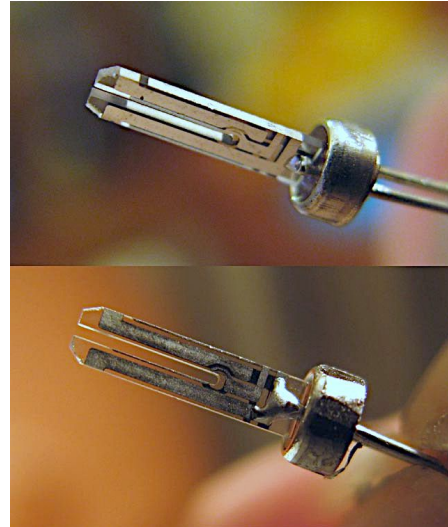
Not true (but we ignore this)

Quartz clocks

- Quartz crystal laser-trimmed to mechanically resonate at a specific frequency
- Piezoelectric effect: mechanical force \leftrightarrow electric field
- Oscillator circuit produces signal at resonant frequency
- Count number of cycles to measure elapsed time



Quartz crystal cluster

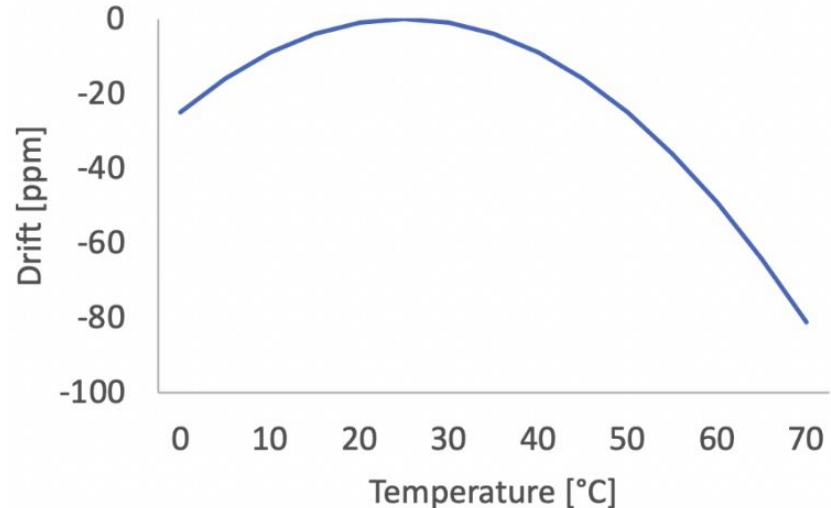


Picture of a quartz crystal resonator, used as the timekeeping component in quartz watches and clocks, with the case removed. It is formed in the shape of a tuning fork. Most such quartz clock crystals vibrate at a frequency of 32768 Hz.

Quartz clock error: drift

- One clock runs slightly fast, another slightly slow
- Drift measured in parts per million (ppm)
- $1 \text{ ppm} = 1 \text{ microsecond/second} = 86 \text{ ms/day} = 32 \text{ s/year}$
- Most computer clocks correct within $\approx 50 \text{ ppm}$

Temperature significantly
affects drift



Mechanic clocks



**Rolex Oyster Perpetual Chronometer 14K
Yellow Gold Ref. # 1002 1974 "Mint Shape"**
\$8,850



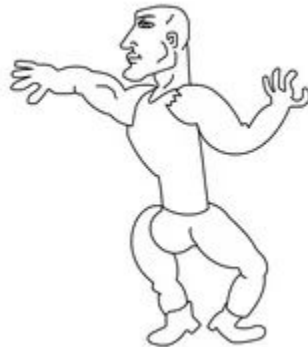
- Contrôle Officiel Suisse des Chronomètres: Swiss certification. To receive it, mechanical watches must show a deviation of no more than $-4/+6$ seconds per day.
- Some modern models of mechanical watches (Omega Co-Axial, Rolex Superlative Chronometer), can show deviations of around ± 2 seconds.
- Marine chronometers used for navigation were among the most accurate mechanical watches. They could achieve accuracy of around ± 1 .

Atomic clocks

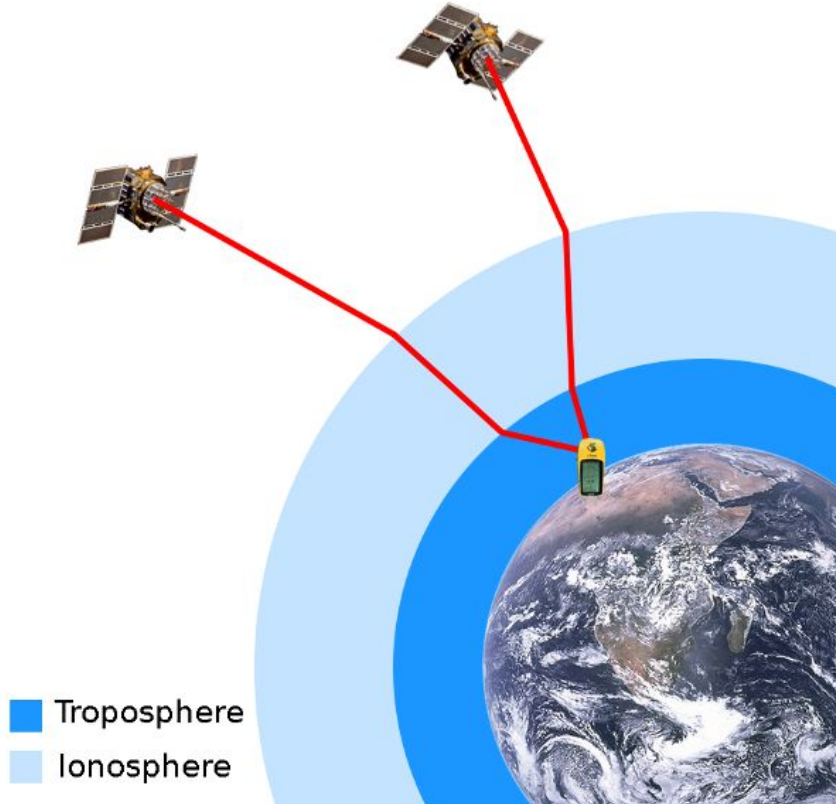


- Caesium-133 has a resonance (“hyperfine transition”) at ≈ 9 GHz
- Tune an electronic oscillator to that resonant frequency
- 1 second = 9,192,631,770 periods of that signal
- Accuracy ≈ 1 in 10^{14} (1 second in 3 million years)
- Price $\approx \$25,000$ (?) (can get cheaper rubidium clocks for $\approx \$1,000$)

Comparison



GPS as time source



- 31 satellites, each carrying an atomic clock
- satellite broadcasts current time and location
- calculate position from speed-of-light delay between satellite and receiver
- corrections for atmospheric effects, relativity, etc.
- in datacenters, need antenna on the roof



EPIC

Institute of Technology
Powered by ePam

Coordinated Universal Time (UTC)



- Greenwich Mean Time (GMT, solar time): it's noon when the sun is in the south, as seen from the Greenwich meridian
- International Atomic Time (TAI): 1 day is $24 \times 60 \times 60 \times 9,192,631,770$ periods of caesium-133's resonant frequency
- Problem: speed of Earth's rotation is not constant
- Compromise: UTC is TAI with corrections to account for Earth rotation
- Time zones and daylight savings time are offsets to UTC

Leap seconds

Every year, on 30 June and 31 December at 23:59:59 UTC, one of three things happens:

- The clock immediately jumps forward to 00:00:00, skipping one second (**negative leap second**)
- The clock moves to 00:00:00 after one second, as usual
- The clock moves to 23:59:60 after one second, and then moves to 00:00:00 after one further second (**positive leap second**)

This is announced several months beforehand.



The first leap second was inserted into the UTC time scale on June 30, 1972. Leap seconds are used to keep the difference between UT1 and UTC to within ± 0.9 s. The table below lists all leap seconds that have already occurred, or are scheduled to occur.

All leap seconds listed in the table are positive leap seconds, which means an extra second is inserted into the UTC time scale. The sequence of events is:

23h 59m 59s - 23h 59m 60s - 00h 00m 00s

Leap Seconds Inserted into the UTC Time Scale

Date	MJD	Date	MJD	Date	MJD	Date	MJD
2016-12-31	57753	1998-12-31	51178	1989-12-31	47891	1979-12-31	44238
2015-06-30	57203	1997-06-30	50629	1987-12-31	47160	1978-12-31	43873
2012-06-30	56108	1995-12-31	50082	1985-06-30	46246	1977-12-31	43508
2008-12-31	54831	1994-06-30	49533	1983-06-30	45515	1976-12-31	43143
2005-12-31	53735	1993-06-30	49168	1982-06-30	45150	1975-12-31	42777
		1992-06-30	48803	1981-06-30	44785	1974-12-31	42412
		1990-12-31	48256			1973-12-31	42047
						1972-12-31	41682
						1972-06-30	41498

Who said?

The decision on when to add a an extra second falls to the Earth Orientation Centre, a Paris-based division of the International Earth Rotation and Reference System Service, and must be announced "at least eight weeks in advance," ITU spokesman Sanjay Acharya told AFP.

It is no split-second decision. In fact, countries are at such odds over whether to do away with the "leap second"—an extra second periodically added to compensate for irregularities in the earth's rotation around the sun—that they have put off deciding the matter until 2023, the United Nations announced Thursday.

How computers represent timestamps

Two most common representations:

- **Unix time:** number of seconds since 1 January 1970 00:00:00 UTC (the “epoch”), *not counting leap seconds*
- **ISO 8601:** year, month, day, hour, minute, second, and timezone offset relative to UTC
example: 2021-11-09T09:50:17+00:00

Conversion between the two requires:

- Gregorian calendar: 365 days in a year, except leap years
($\text{year} \% 4 == 0 \ \&\& \ (\text{year} \% 100 != 0 \ || \ \text{year} \% 400 == 0)$)
- Knowledge of past and future leap seconds. . . ?!

How most software deals with leap seconds

By ignoring them!

However, OS and DistSys often need timings with sub-second accuracy.

30 June 2012: bug in Linux kernel caused livelock on leap second, causing many Internet services to go down

Pragmatic solution: “smear” (spread out) the leap second over the course of a day



02

Problems with Time

Y2K

Most computers in the 1900s formatted the year as 2 digits, like 1999 coded as 99. Computer programmers and world leaders were worried that on January 1, 2000, computers wouldn't read 00 as 2000 and instead shift back to 1900, 0000, or 1000.

This was speculated to cause computer malfunctions and lead to global damage related to banking, transportation, and power plant operations.



EPIC

Institute of Technology
Powered by <epam>

Y2K



2038's problem

The problem is related to 32-bit Unix systems that use time representation as the number of seconds elapsed since January 1, 1970 (Unix epoch).

At 03:14:07 UTC on January 19, 2038, this value will exceed 2,147,483,647 seconds, causing an overflow and interpreting the time as a negative number.

A detective story

In the night from 30 June to 1 July 2012 (UK time), many online services and systems around the world crashed simultaneously.

Servers locked up and stopped responding.

Some airlines could not process any reservations or check-ins for several hours.

What happened?

Leap second

On the night of June 30 to July 1, 2012, a leap second was added at 23:59:60 UTC. This insertion caused various systems and services to crash or behave unexpectedly. Some of the most notable issues included:

- 1) Several popular websites and services, including Reddit, LinkedIn, and Yelp, experienced outages.
- 2) Linux-based systems, in particular, were affected. The Linux kernel and certain Java applications did not handle the leap second properly, leading to high CPU usage and system instability.

Daylight Saving Time

The transition to and from daylight saving time can lead to problems in IT systems, especially those that depend on precise timekeeping. Issues can arise due to incorrect handling of time changes.

Daylight Saving Time

Daylight Savings Time (DST) Changes for 2007 and their impact on SQL Server

By  [mssql-support](#)

Published Jan 15 2019 10:44 AM

 377 Views



First published on MSDN on Feb 21, 2007

As many of you know, Daylight Savings Time (DST) in the United States is changing this year to start on March 11th, 2007 (3 weeks earlier than normal) and end on November 4th, 2007 (1 week later than normal). I thought I would write a quick post on the impact of these changes on SQL Server.

The impact can be summarized in 3 key points:

- Notification Services is the ONLY SQL product that has a DST 2007 bug and requires a fix to be applied. You can find this update at the following KB article <http://support.microsoft.com/kb/931815>
- SQL Server Engine, Analysis Services, etc. uses/relies on Windows for many date/time functionalities, so if you have the proper updates for Windows there should be no noticeable issues. You can find the proper updates for Windows at this KB article <http://support.microsoft.com/kb/931836>
- There are other perceived DST issues for SQL Server that have been around that are not specific to the 2007 change but you may perceive them to be related. You can read about these issues at this KB article <http://support.microsoft.com/kb/931975>

Leap year problem

Incorrect handling of February 29 in leap years can lead to software and system failures.

Leap year problem

PRIYA GANAPATI

GEAR DEC 31, 2008 5:47 PM

Zune Freeze Result of Leap Year: Microsoft

It was the Z2k problem after all, a glitch related to the inability of the device clock to handle the extra day in a leap year that froze thousands of Zune media players Wednesday morning. "A bug in the internal clock driver related to the v



It was the Z2k problem after all, a glitch related to the inability of the device clock to handle the extra day in a leap year that froze thousands of Zune media players Wednesday morning.

"A bug in the internal clock driver related to the way the device handles a leap year affected Zune users," said the company in a statement. "That being the case, the issue should be resolved over the next 24 hours as the time change moves to January 1, 2009."

Time Zone

Colombo, Sri Lanka is 3:30 hours ahead of Belgrade, Serbia.

Colombo, Sri Lanka is **3:30 hours ahead** of Belgrade, Serbia. · [Current time](#) · [Map from Belgrade, Serbia to Colombo, Sri Lanka](#) · [More trip calculations](#) · [Meeting ...](#)

UTC+12:45



New Zealand: Chatham Islands

**EPIC**Institute of Technology
Powered by <epam>

GPS Problem

Massive GPS Jamming Attack by North Korea

May 8, 2012 - By **GPS World Staff**

Est. reading time: 2:30 

Large coordinated cyber attacks from North Korea near its border with South Korea produced electronic jamming signals that affected GPS navigation for passenger aircraft, ships, and in-car navigation for roughly a week in late April and early May. To date, no accidents, casualties, or fatalities have been attributed to jammed navigation signals aboard 337 commercial flights in and out of South Korean international airports, on 122 ships, including a passenger liner carrying 287 people and a petroleum tanker. One South Korean driver tweeted "It also affects the car navigation GPS units. I am getting a lot of errors while driving in Seoul."

03

Clock synchronisation. NTP

Clock synchronisation

Computers track physical time with a quartz clock (with battery, continues running when power is off)

Due to **clock drift**, clock error gradually increases

Clock skew: difference between two clocks at a point in time

Solution: Periodically get the current time from a server that has a more accurate time source (atomic clock or GPS receiver)

Protocols: Network Time Protocol (**NTP**), Precision Time Protocol (**PTP**)

Network Time Protocol (NTP)

Many operating system vendors run NTP servers, configure OS to use them by default.

Hierarchy of clock servers arranged into strata:

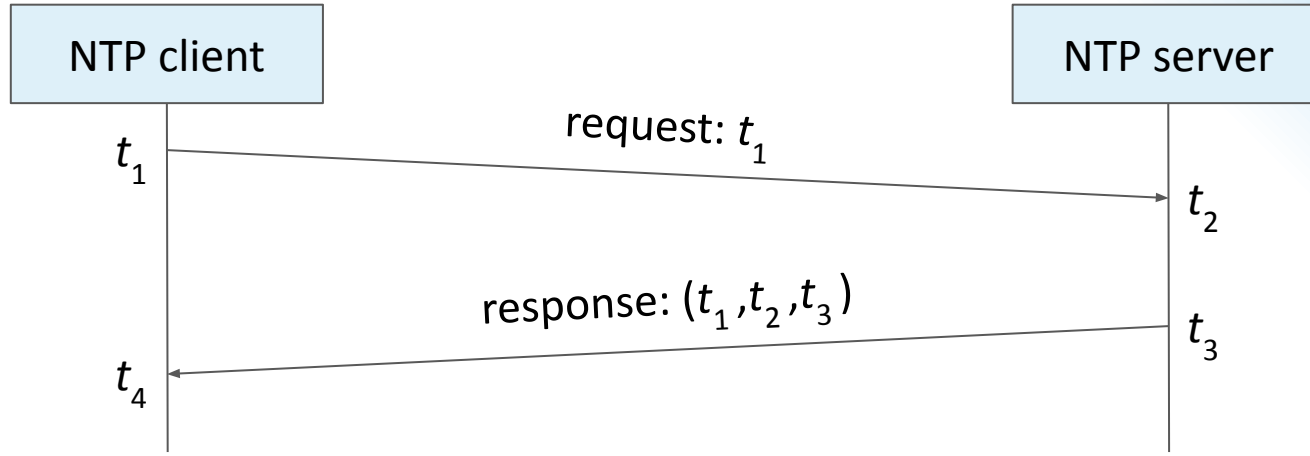
- Stratum 0: atomic clock or GPS receiver
- Stratum 1: synced directly with stratum 0 device
- Stratum 2: servers that sync with stratum 1, etc.

May contact multiple servers, discard outliers, average rest.

Makes multiple requests to the same server, use statistics to reduce random error due to variations in network latency.

Reduces clock skew to a few milliseconds in good network conditions, but can be much worse!

Estimating time over a network



Round-trip network delay: $\delta = (t_4 - t_1) - (t_3 - t_2)$

Estimated server time when client receives response: $t_3 + \delta/2$

Estimated clock skew: $\theta = t_3 + \delta/2 - t_4 = (t_2 - t_1 + t_3 - t_4)/2$

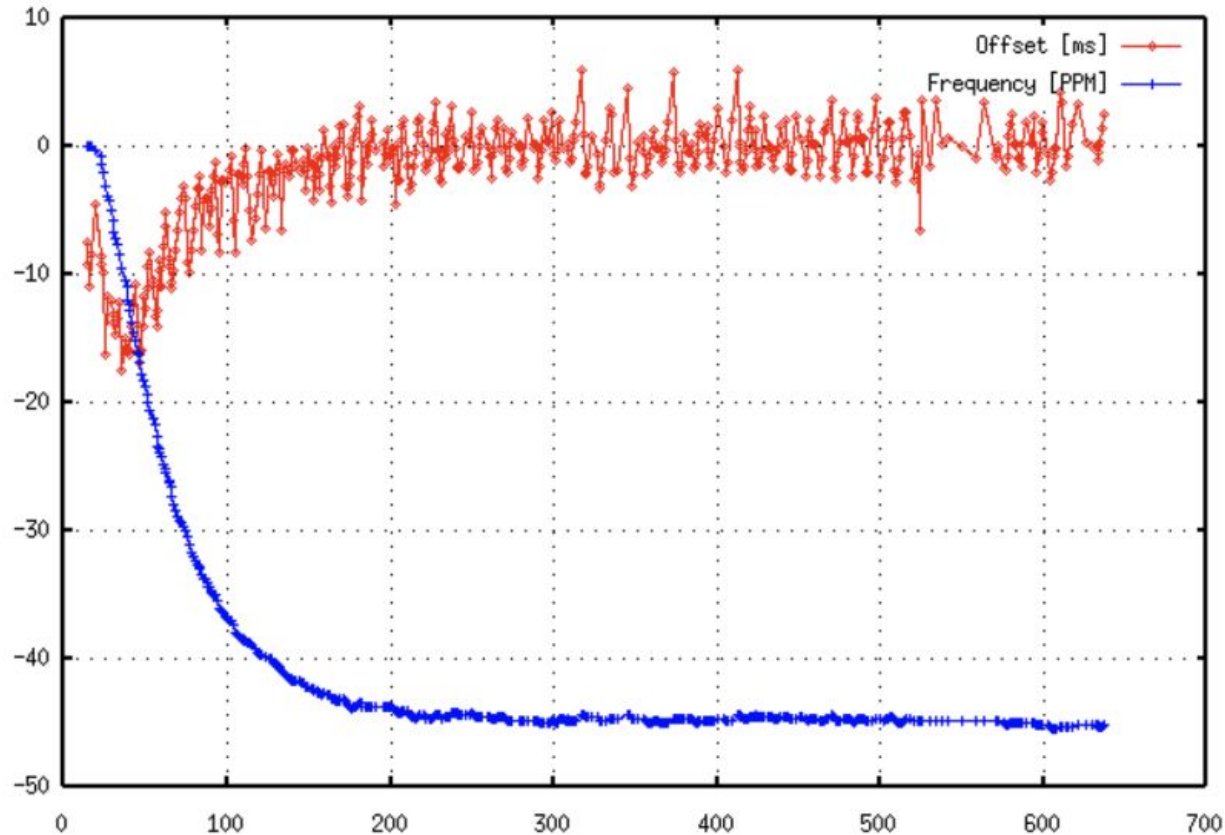
Correcting clock skew

Once the client has estimated the clock skew θ , it needs to apply that correction to its clock.

- If $|\theta| < 125 \text{ ms}$, slew the clock:
slightly speed it up or slow it down by up to 500 ppm
(brings clocks in sync within ≈ 5 minutes)
- If $125 \text{ ms} \leq |\theta| < 1,000 \text{ s}$, step the clock:
suddenly reset client clock to estimated server timestamp
- If $|\theta| \geq 1,000 \text{ s}$, panic and do nothing:
leave the problem for a human operator to resolve

Systems that rely on clock sync need to monitor clock skew!

Initial run of NTP 3.5f on HP L2000-44/2



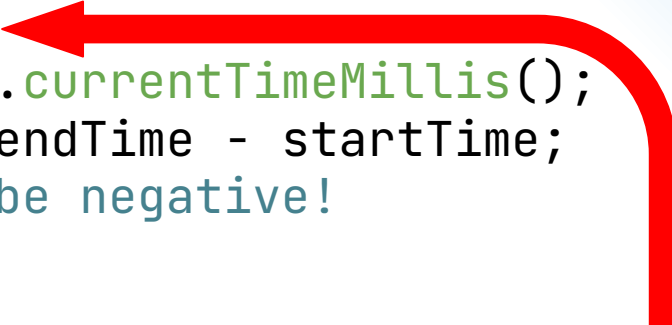
[Source](#)

**EPIC**Institute of Technology
Powered by epan

Monotonic and time-of-day clocks

// BAD:

```
long startTime = System.currentTimeMillis();  
doSomething();  
long endTime = System.currentTimeMillis();  
long elapsedMillis = endTime - startTime;  
// elapsedMillis may be negative!
```



NTP client steps the clock during this

// GOOD:

```
long startTime = System.nanoTime();  
doSomething();  
long endTime = System.nanoTime();  
long elapsedNanos = endTime - startTime;  
// elapsedNanos is always  $\geq 0$ 
```

Monotonic and time-of-day clocks

```
public static long nanoTime()
```

Returns the current value of the most precise available system timer, in nanoseconds.

This method can only be used to measure elapsed time and is not related to any other notion of system or wall-clock time. The value returned represents nanoseconds since some fixed but arbitrary *origin* time (perhaps in the future, so values may be negative).

This method provides nanosecond precision, but not necessarily nanosecond accuracy. No guarantees are made about how frequently values change. Differences in successive calls that span greater than approximately 292 years (2^{63} nanoseconds) will not accurately compute elapsed time due to numerical overflow.

Monotonic and time-of-day clocks

Time-of-day clock:

- Time since a fixed date (e.g. 1 January 1970 epoch)
- May suddenly move forwards or backwards (NTP stepping), subject to leap second adjustments
- Timestamps can be compared across nodes (if synced)
- Java:
`System.currentTimeMillis()`
- Linux:
`clock_gettime(CLOCK_REALTIME)`

Monotonic clock:

- Time since arbitrary point (e.g. when machine booted up)
- Always moves forwards at near-constant rate
- Good for measuring elapsed time on a single node
- Java:
`System.nanoTime()`
- Linux:
`clock_gettime(CLOCK_MONOTONIC)`

04

Causality. Happens-before relation

The happens-before relation

An **event** is something happening at one node (sending or receiving a message, or a local execution step).

We say event a **happens before** event b (written $a \rightarrow b$) if:

- a and b occurred at the same node, and a occurred before b in that node's local execution order; or
- event a is the sending of some message m , and event b is the receipt of that same message m (assuming sent messages are unique); or
- there exists an event c such that $a \rightarrow c$ and $c \rightarrow b$.

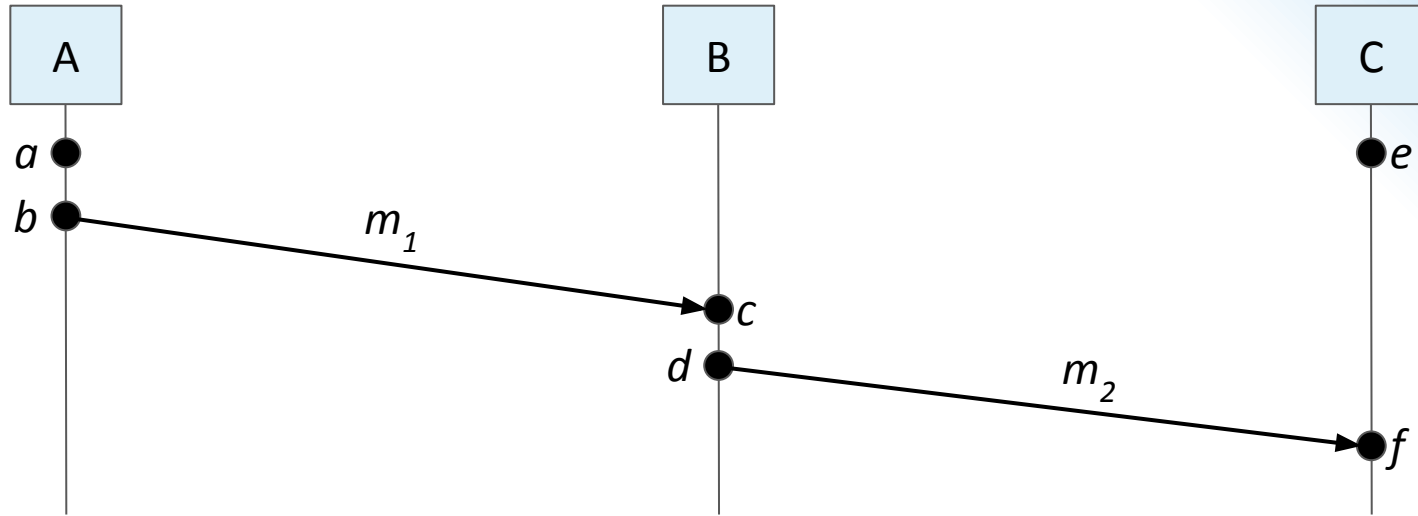
The happens-before relation is a partial order: it is possible that neither $a \rightarrow b$ nor $b \rightarrow a$. In that case, a and b are concurrent (written $a \parallel b$).



EPIC

Institute of Technology
Powered by epan

Happens-before relation example



- $a \rightarrow b$, $c \rightarrow d$, and $e \rightarrow f$ due to node execution order
- $b \rightarrow c$ and $d \rightarrow f$ due to messages m_1 and m_2
- $a \rightarrow c$, $a \rightarrow d$, $a \rightarrow f$, $b \rightarrow d$, $b \rightarrow f$, and $c \rightarrow f$ due to transitivity

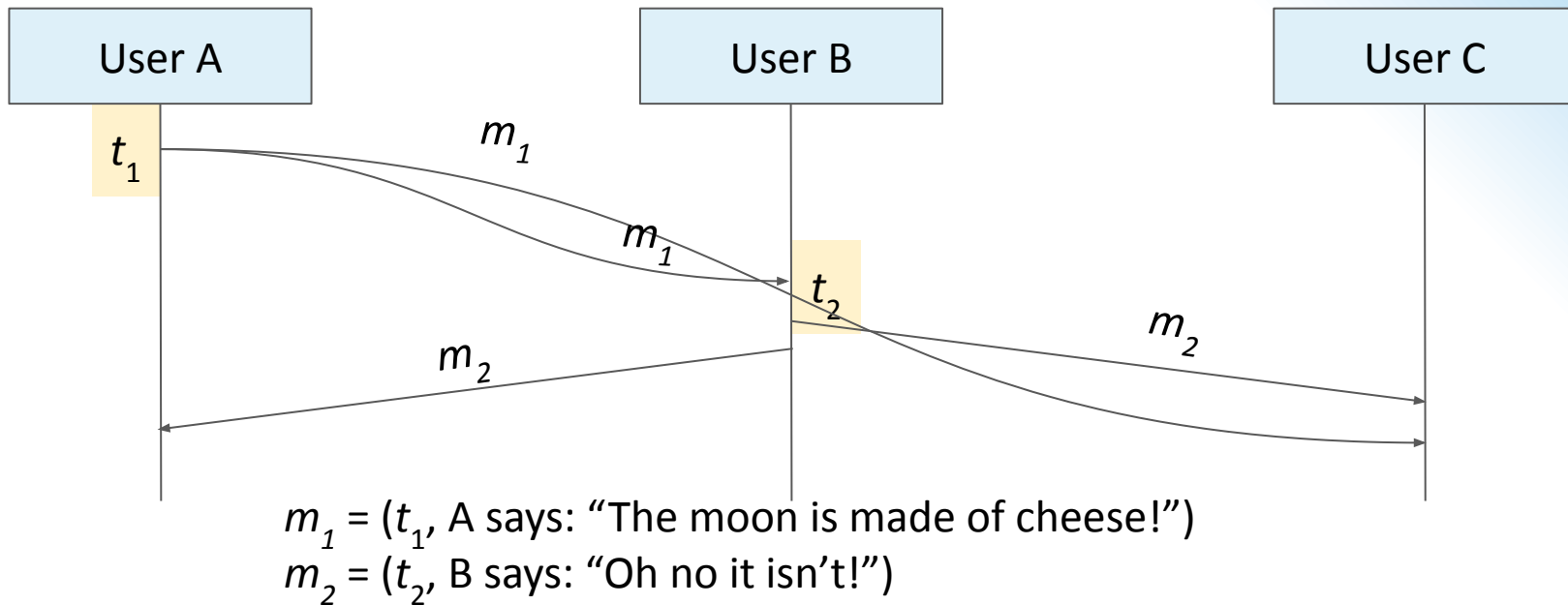
Causality

Taken from physics (relativity).

- When $a \rightarrow b$, then a might have caused b .
- When $a \perp\!\!\!\perp b$, we know that a cannot have caused b .

Happens-before relation encodes potential causality.

Physical timestamps inconsistent with causality



Problem: even with synced clocks, $t_2 < t_1$ is possible.
Timestamp order is inconsistent with expected order!

05

Logical time. Lamport and vector clocks

Logical vs. physical clocks

- Physical clock: count number of **seconds elapsed**
- Logical clock: count number of **events occurred**

Physical timestamps: useful for many things, but may be **inconsistent with causality**.

Logical clocks: designed to **capture causal dependencies**.

$$(e1 \rightarrow e2) \Rightarrow (T(e1) < T(e2))$$

We will look at two types of logical clocks:

- Lamport clocks
- Vector clocks

**EPIC**Institute of Technology
Powered by ePam

Lamport clocks algorithm

on initialisation **do**

$t := 0$ \square each node has its own local variable t

end on

on any event occurring at the local node **do**

$t := t + 1$

end on

on request to send message m **do**

$t := t + 1$; send (t, m) via the underlying network link

end on

on receiving (t', m) via the underlying network link **do**

$t := \max(t, t') + 1$

deliver m to the application

end on

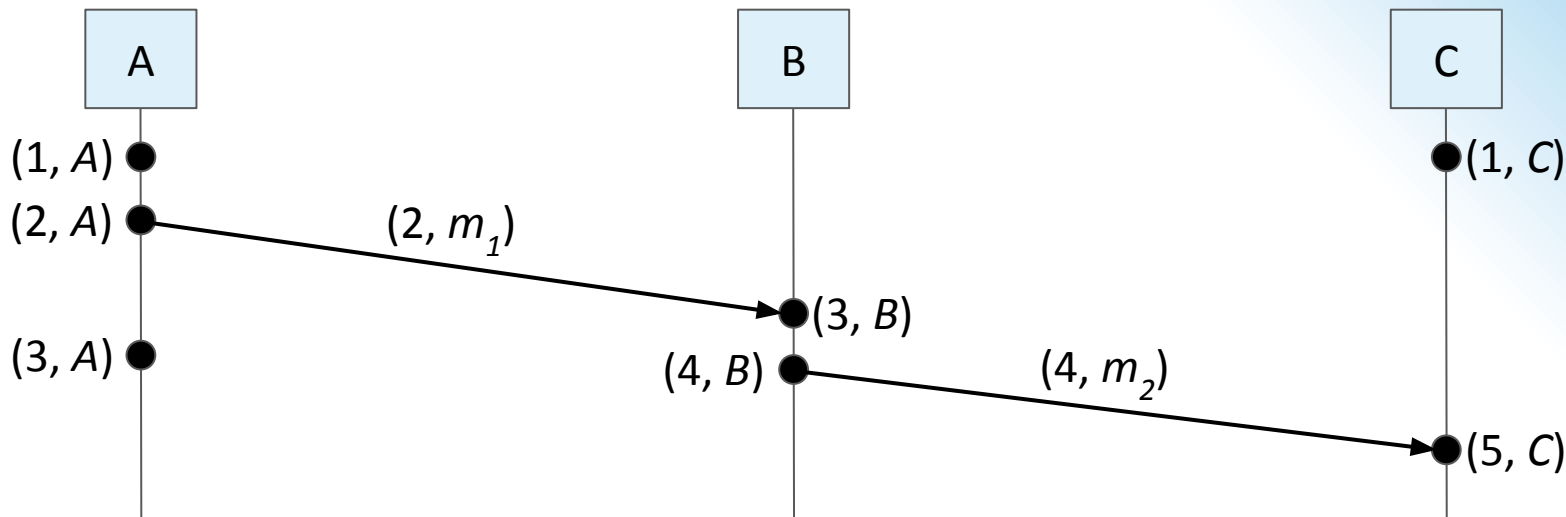
Lamport clocks in words

- Each node maintains a counter t , incremented on every local event e
- Let $L(e)$ be the value of t after that increment
- Attach current t to messages sent over network
- Recipient moves its clock forward to timestamp in the message (if greater than local counter), then increments

Properties of this scheme:

- If $a \rightarrow b$ then $L(a) < L(b)$
- However, $L(a) < L(b)$ does not imply $a \rightarrow b$
- Possible that $L(a) = L(b)$ for $a \neq b$

Lamport clocks example



Let $N(e)$ be the node at which event e occurred.

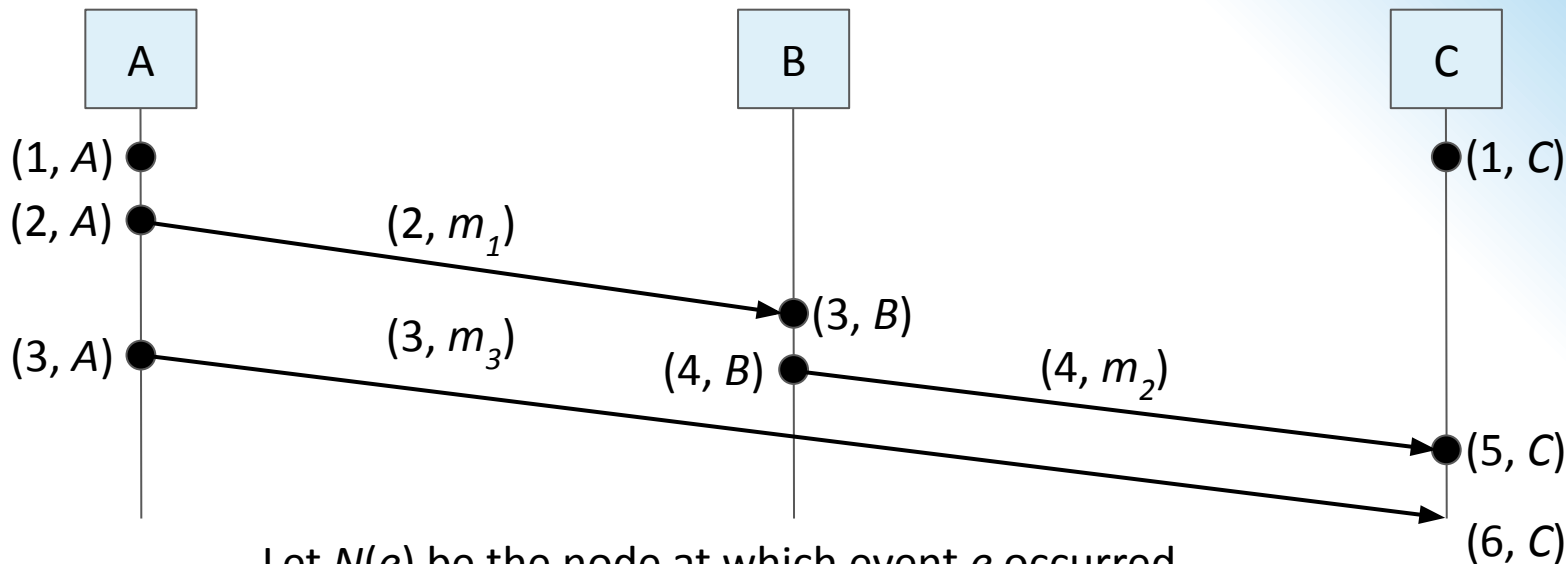
Then the pair $(L(e), N(e))$ **uniquely identifies** event e .

Define a **total order** $<$ using Lamport timestamps:

$$(a < b) \Leftrightarrow (L(a) < L(b) \vee (L(a) = L(b) \wedge N(a) < N(b)))$$

This order is causal: $(a \rightarrow b) \Rightarrow (a < b)$

Lamport clocks example



Let $N(e)$ be the node at which event e occurred.

Then the pair $(L(e), N(e))$ **uniquely identifies** event e .

Define a **total order** $<$ using Lamport timestamps:

$$(a < b) \Leftrightarrow (L(a) < L(b) \vee (L(a) = L(b) \wedge N(a) < N(b)))$$

This order is causal: $(a \rightarrow b) \Rightarrow (a < b)$

Vector clocks

Given Lamport timestamps $L(a)$ and $L(b)$ with $L(a) < L(b)$ we can't tell whether $a \rightarrow b$ or $a \parallel b$.

If we want to detect which events are concurrent, we need **vector clocks**:

- Assume n nodes in the system, $N = \langle N_1, N_2, \dots, N_n \rangle$
- Vector timestamp of event a is $V(a) = \langle t_1, t_2, \dots, t_n \rangle$
- t_i is number of events observed by node N_i
- Each node has a current vector timestamp T
- On event at node N_i , increment vector element $T[i]$
- Attach current vector timestamp to each message
- Recipient merges message vector into its local vector

Vector clocks algorithm

on initialisation at node N_i **do**

$T := \langle 0, 0, \dots, 0 \rangle$ \square local variable at node N_i

end on

on any event occurring at node N_i **do**

$T[i] := T[i] + 1$

end on

on request to send message m at node N_i **do**

$T[i] := T[i] + 1$; send (T, m) via network

end on

on receiving (T', m) at node N_i via the network **do**

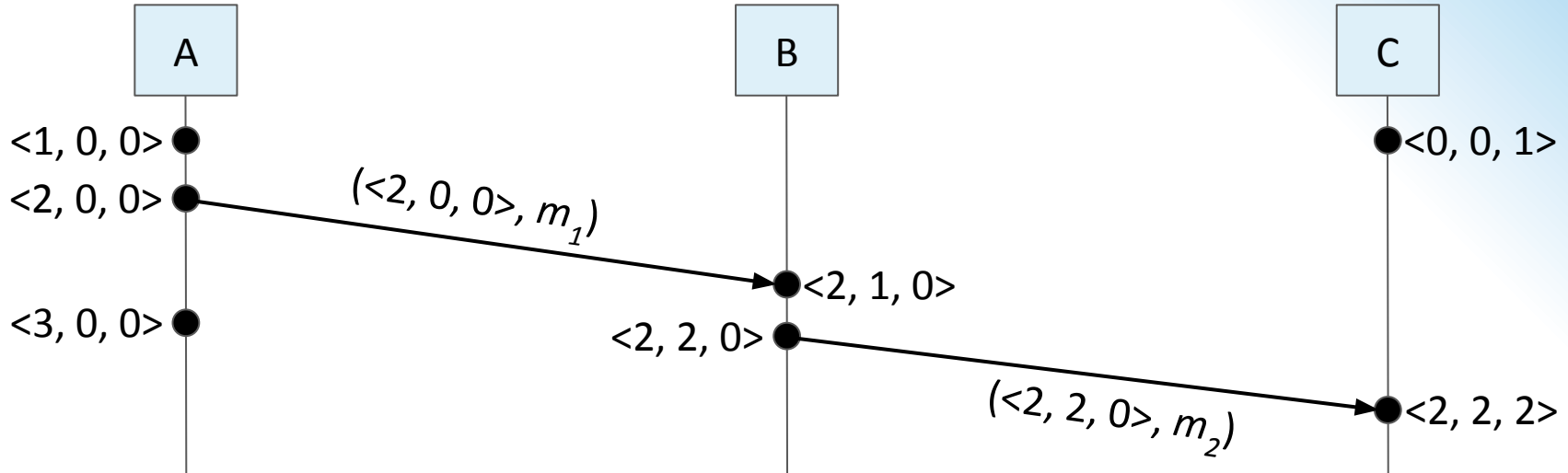
$T[j] := \max(T[j], T'[j])$ for every $j \in \{1, \dots, n\}$

$T[i] := T[i] + 1$; deliver m to the application

end on

Vector clocks example

Assuming the vector of nodes is $N = \langle A, B, C \rangle$:



The vector timestamp of an event e represents a set of events, e and its causal dependencies: $\{e\} \cup \{a \mid a \rightarrow e\}$

For example, $\langle 2, 2, 0 \rangle$ represents the first two events from A, the first two events from B, and no events from C.

Vector clocks ordering

Define the following order on vector timestamps (in a system with n nodes):

- $T = T'$ iff $T[i] = T'[i]$ for all $i \in \{1, \dots, n\}$
- $T \leq T'$ iff $T[i] \leq T'[i]$ for all $i \in \{1, \dots, n\}$
- $T < T'$ iff $T \leq T'$ and $T \neq T'$
- $T \parallel T'$ iff $T \not\leq T'$ and $T' \not\leq T$

$$V(a) \leq V(b) \text{ iff } (\{a\} \cup \{e \mid e \rightarrow a\}) \subseteq (\{b\} \cup \{e \mid e \rightarrow b\})$$

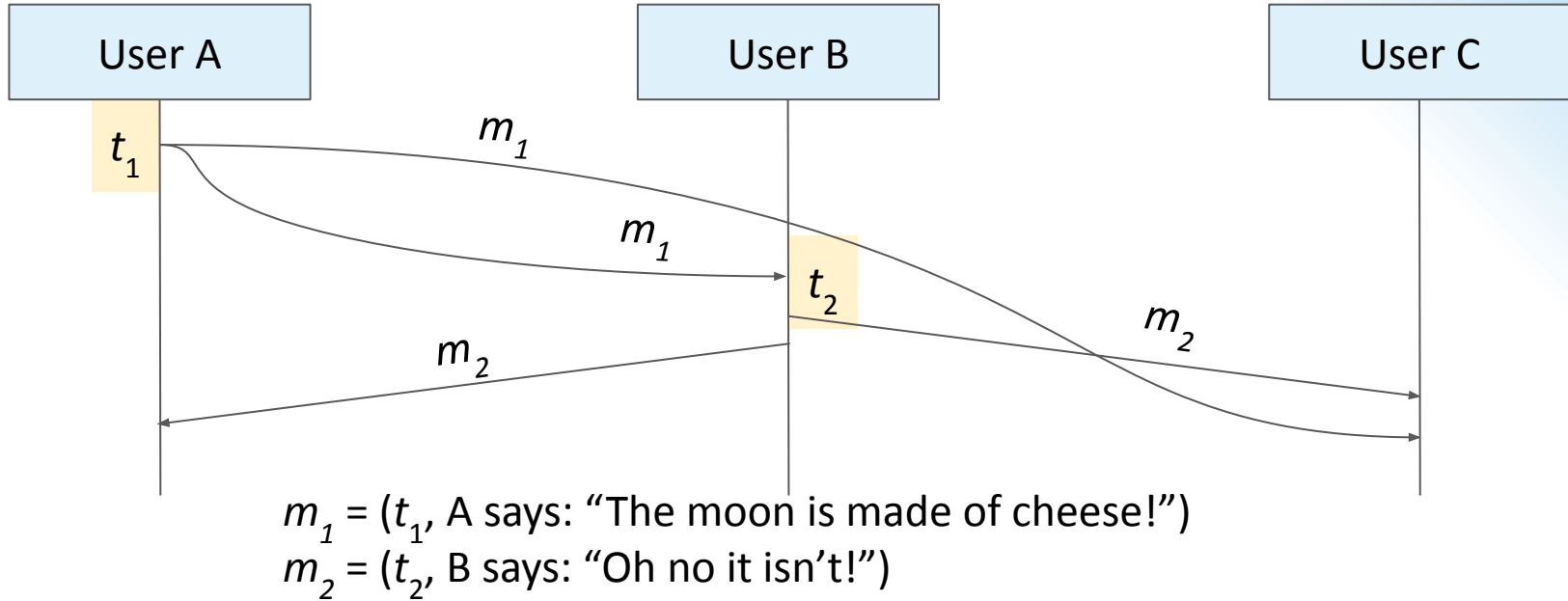
Properties of this order:

- $(V(a) < V(b)) \Leftrightarrow (a \rightarrow b)$
- $(V(a) = V(b)) \Leftrightarrow (a = b)$
- $(V(a) \parallel V(b)) \Leftrightarrow (a \parallel b)$

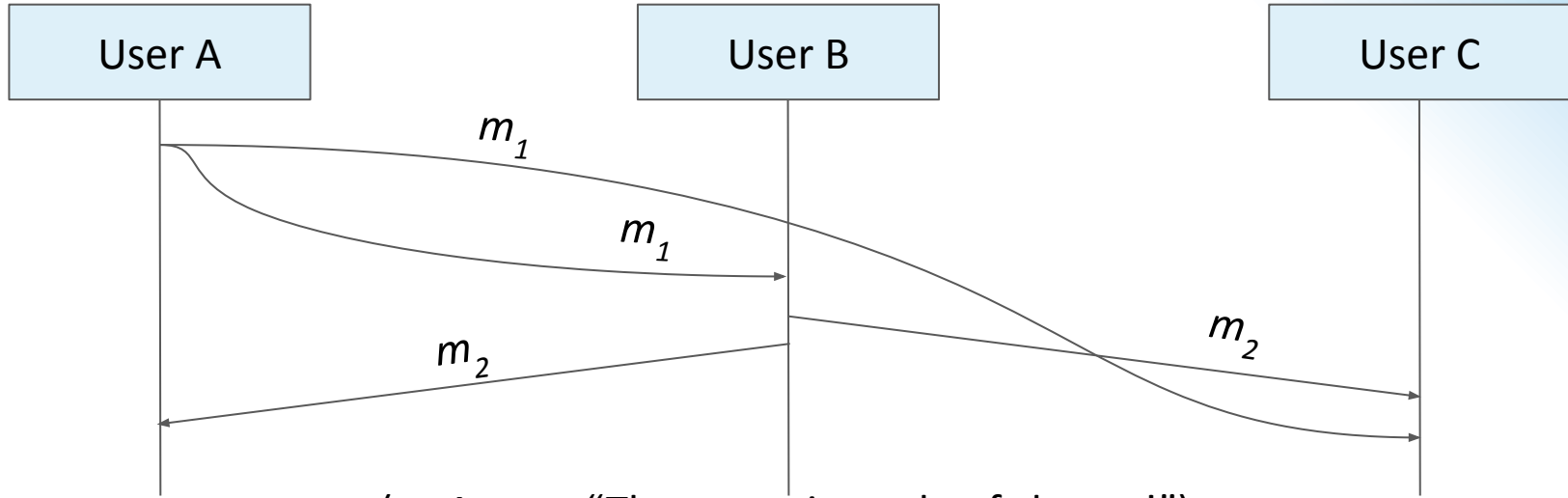
06

Why do i need this?

Ordering of messages using timestamps?



Ordering of messages

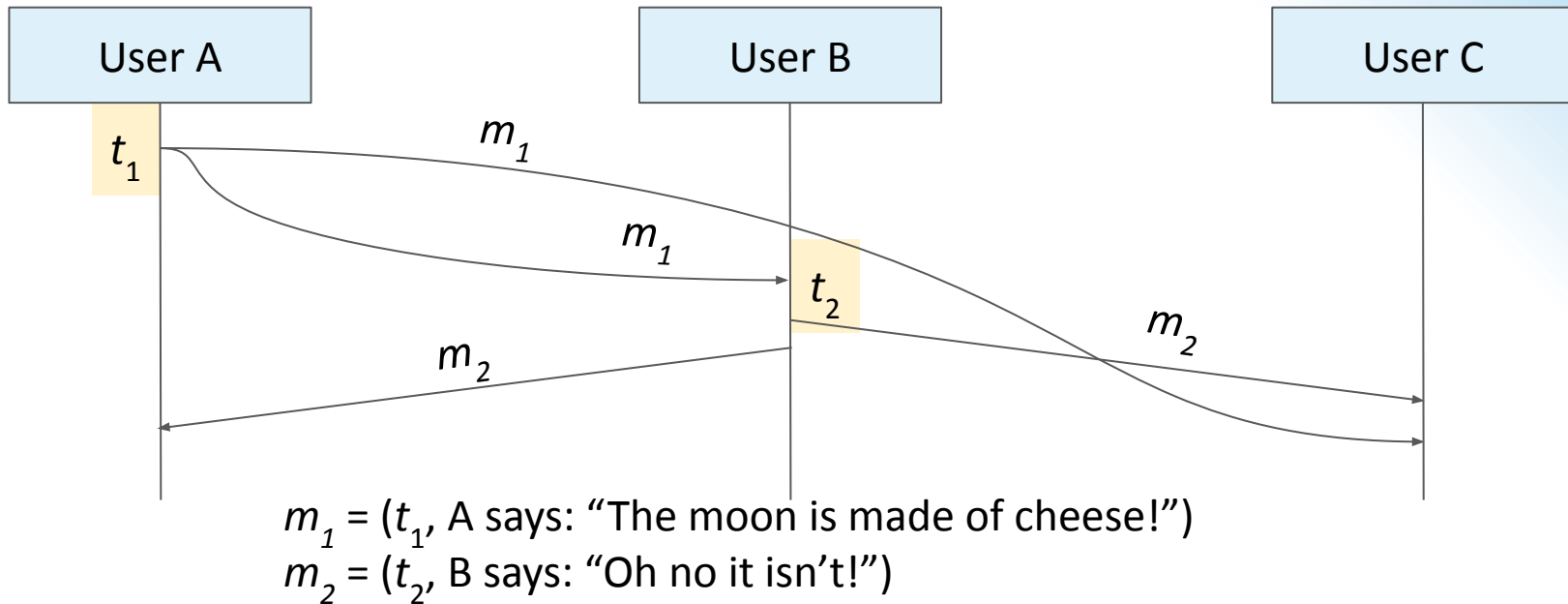


$m_1 = (t_1, \text{A says: "The moon is made of cheese!"})$

$m_2 = (t_2, \text{B says: "Oh no it isn't!"})$

User C sees m_2 first, m_1 second,
even though logically m_1 **happened before** m_2 .

Ordering of messages using timestamps?



Problem: even with synced clocks, $t_2 < t_1$ is possible.
Timestamp order is inconsistent with expected order!

Math

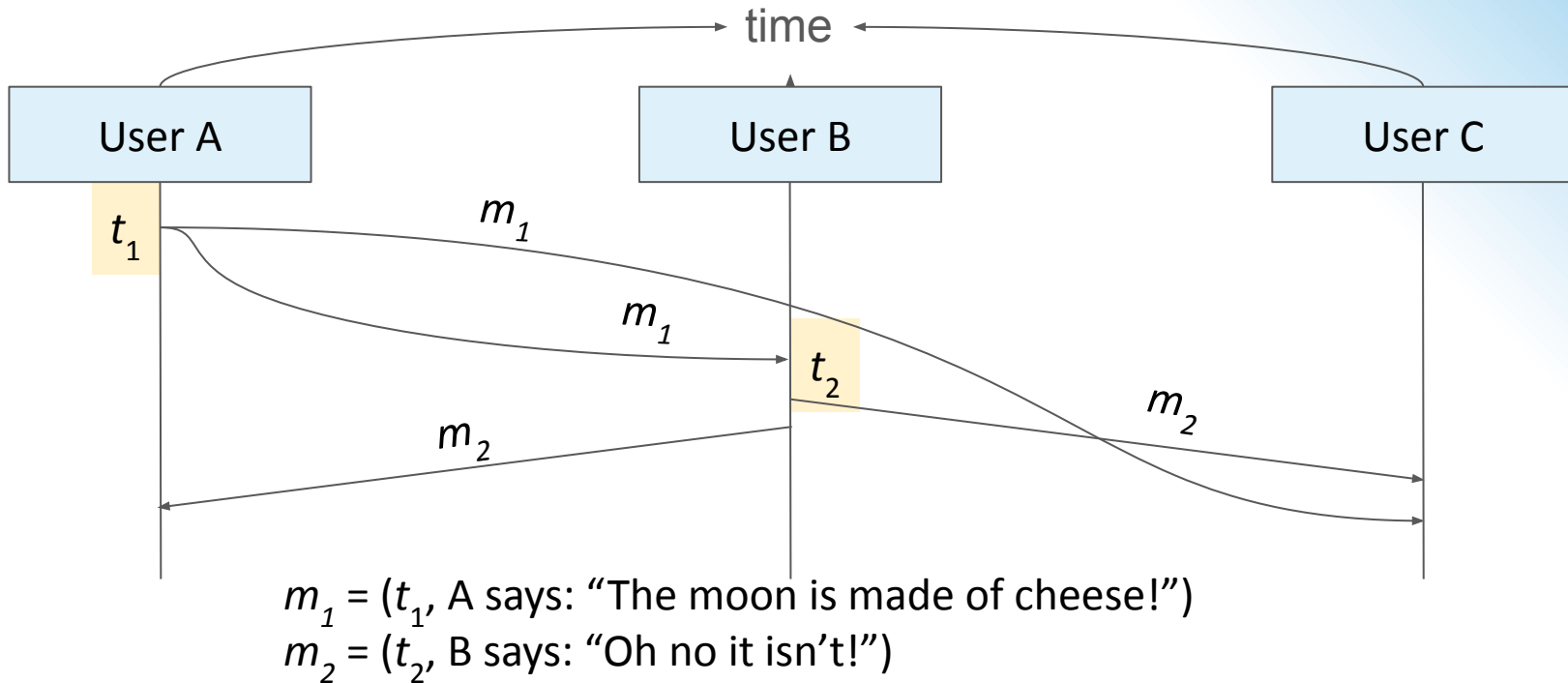
The PAXOS algorithm is used to implement Atomic Broadcast, an important communication primitive useful for building fault-tolerant distributed systems. Transforming a formal description into an efficient, scalable and reliable implementation is a difficult process that requires addressing a number of practical issues and making careful design choices. In this document we share our experience in building, verifying and benchmarking different Paxos-based implementations of Atomic Broadcast.

Practice

Does etcd use Paxos?

Both are written in Go, but a big difference is that Doozer uses Paxos, while **etcd's consensus protocol is Raft**, which gives it the ability to keep the same logs of state changing commands on all nodes in a cluster. 16 июл. 2014 г.

Ordering of messages using timestamps?



Problem: even with synced clocks, $t_2 < t_1$ is possible.
Timestamp order is inconsistent with expected order!

That's All Folks!