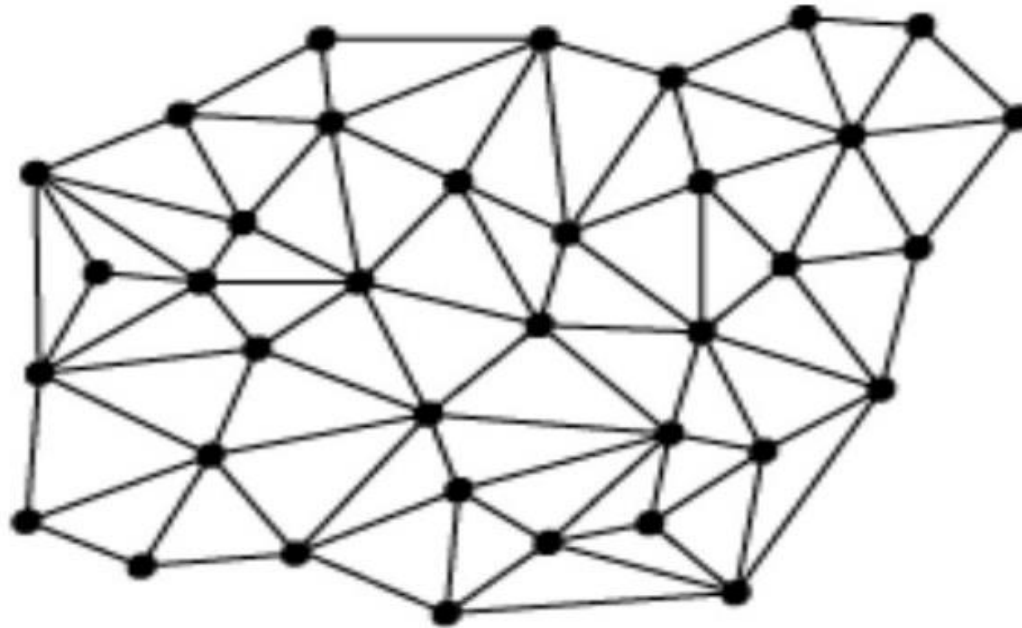


Distributed System Course

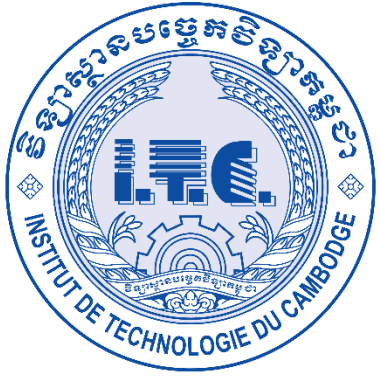
2021-22-GICI41SSD-Distributed System



Academic Year: 2021-2022 Lecturer: SOK Kimheng

Information

Course	Distributed System	48h, 12 Weeks, 4h/week (3 Groups = 96h)
General Distributed System	Week 1	Information, Self-Study Skill, Introduction
	Week 2	Distributed Communication (TCP/IP, Socket, RPC, REST, gRPC, OMQ)
	Week 3	Clock, Timestamp
	Week 4	Fault Tolerance (Two general problem, Byzantine General Problem)
	Week 5	Consensus Algorithm (Paxos, ZooKeeper, Raft)
	Week 6	Quiz
Blockchain	Week 7	Basic Cryptography
	Week 8	Blockchain and Bitcoin (Proof of Work)
	Week 9	Ethereum and Smart Contract (Proof of Stake)
	Week 10	Hyperledger and Self-Sovereign Identity
	Week 11	Security
	Week 12	Final Exam



Distributed System Course

2021-22-GICI41SSD-Distributed System

Week3:

Clock & Timestamp

Academic Year: 2021-2022 Lecturer: SOK Kimheng

Agenda

- 1 General Knowledge
- 2 Atomic Clock & Quartz Clock
- 3 Causality and Happens-before relation
- 4 Physical and Logical clock
- 5 Lamport clock
- 6 Vector clock
- 7 Hashgraph Gossip Protocol

Clock and Timestamp

General Knowledge

- Earth rotate around itself = 1 day = 24h (1h=60min 1min=60s)
- Solar calendar: Earth rotate around the sun = 1 year = 12 months
- Lunar calendar: from new moon to another new moon = 1 month

What you don't realize

- We used to have only 10 months (September=7, December=10)
 - March is the first month of the year, January and February was added later
- Leap year (February 29)
- Leap second (June 30, December 31)

Clock and Timestamp

How 1 second is measure?

- The need for precision
 - 1955 Atomic Clock
 - depend on the vibration/oscillation of electron inside atom by exposing it to the electromagnetic field.
 - Cesium-133 was chosen
 - $1 \text{ year} = 356.1/4 \Rightarrow 1\text{s} = 9,192,631,770 \text{ ticks of Cs Atom}$
 - Quartz Clock
- Problem
 - **Clock drift:** clock error gradually increase
 - **Clock skew:** different between two clocks at a point of time
- Solution
 - Get current time from server using NTP (Network Time Protocol)
 - NTP server is connected to the accurate time source (Atomic clock, GPS receiver)

Clock and Timestamp

What happen when client and server time mismatch?



Did Not Connect: Potential Security Issue

Firefox detected an issue and did not continue to www.google.com. The website is either misconfigured or your computer clock is set to the wrong time.

What can you do about it?

www.google.com has a security policy called HTTP Strict Transport Security (HSTS), which means that Firefox can only connect to it securely. You can't add an exception to visit this site.

Your computer clock is set to 12/9/2021. Make sure your computer is set to the correct date, time, and time zone in your system settings, and then refresh www.google.com.

If your clock is already set to the right time, the website is likely misconfigured, and there is nothing you can do to resolve the issue. You can notify the website's administrator about the problem.

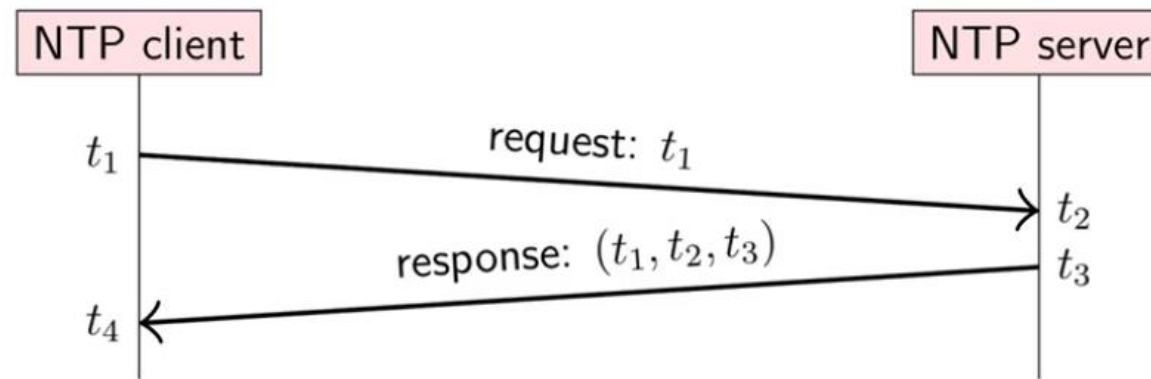
[Learn more...](#)

Go Back

Advanced...

Clock and Timestamp

Estimating time over network



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

Estimated server time when client receives response: $t_3 + \frac{\delta}{2}$

Estimated clock skew: $\theta = t_3 + \frac{\delta}{2} - t_4 = \frac{t_2 - t_1 + t_3 - t_4}{2}$

Clock and Timestamp

Correcting Clock Skew

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

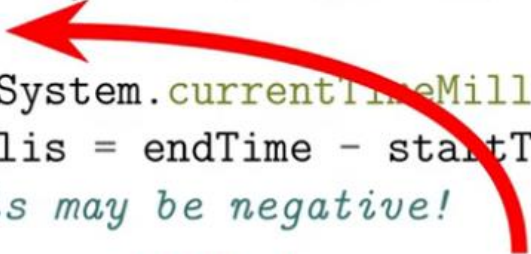
- ▶ If $\theta < 125$ ms, **slew** the clock:
slightly speed it up or slow it down by up to 500 ppm
(brings clocks in sync within ≈ 5 minutes)
- ▶ If $125 \text{ ms} \leq \theta < 1,000$ s, **step** the clock:
suddenly reset client clock to estimated server timestamp
- ▶ If $\theta \geq 1,000$ s, **panic**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

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

Clock and Timestamp

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 >= 0
```

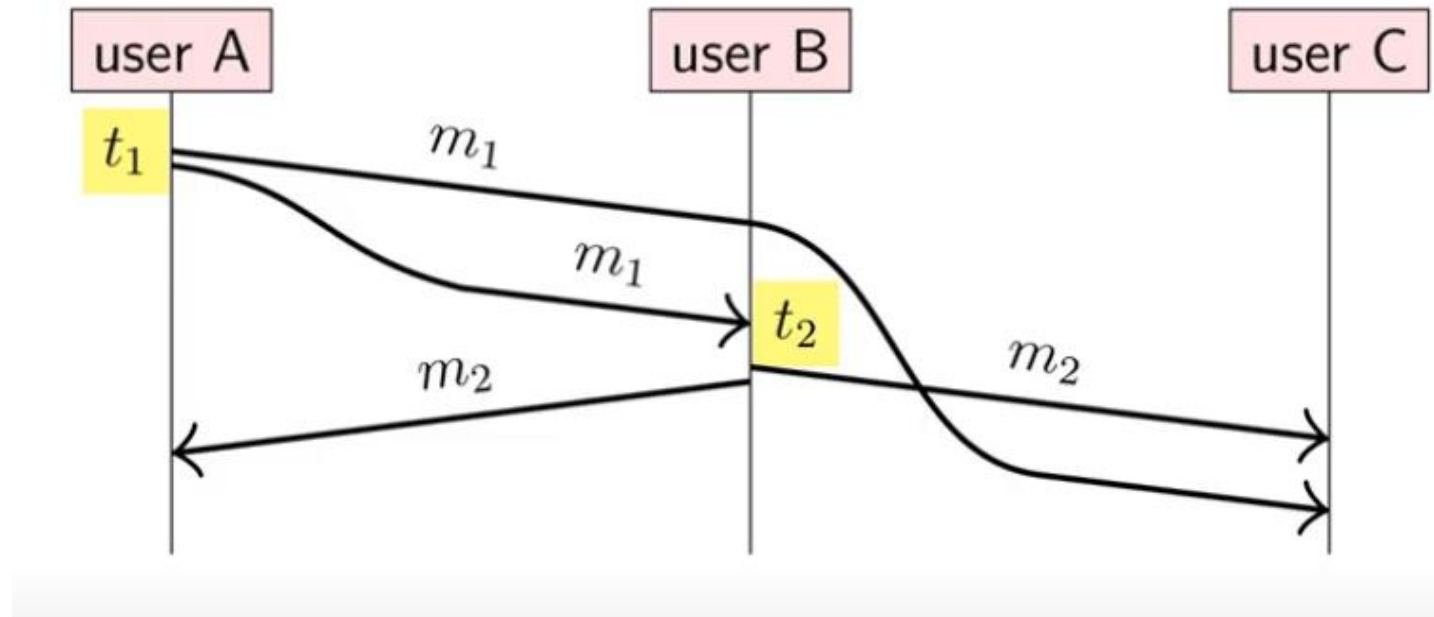
Clock and Timestamp

Monotonic and time-of-day clocks

- Time-of-day clock
 - Time since a fixed date (Ex: 1 January 1970 epoch)
 - May suddenly move forwards or backwards (NTP stepping), subject to leap second adjustment
 - Timestamps can be compared across nodes (if synced)
 - Java: `System.currentTimeMillis()`
 - Linux: `clock_gettime(CLOCK_REALTIME)`
- Monotonic clock
 - Time since arbitrary point (Ex: 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)`

Clock and Timestamp

Message ordering problem



Clock and Timestamp

The happens-before relation

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

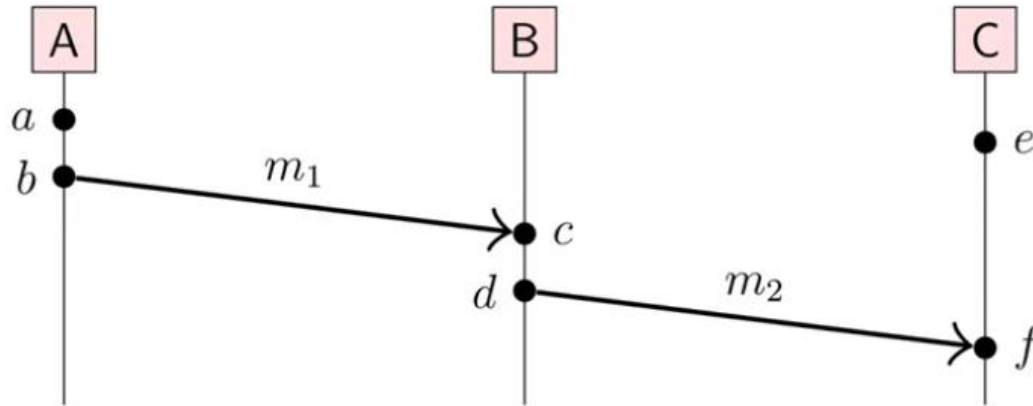
We say event a **happens before** event b (written $a \rightarrow b$) iff:

- ▶ 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 $a \parallel b$).

Clock and Timestamp

The happens-before relation



- ▶ $a \rightarrow b$, $c \rightarrow d$, and $e \rightarrow f$ due to process 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
- ▶ $a \parallel e$, $b \parallel e$, $c \parallel e$, and $d \parallel e$

Clock and Timestamp

Physical clock

- Count number of seconds elapsed

Logical clock

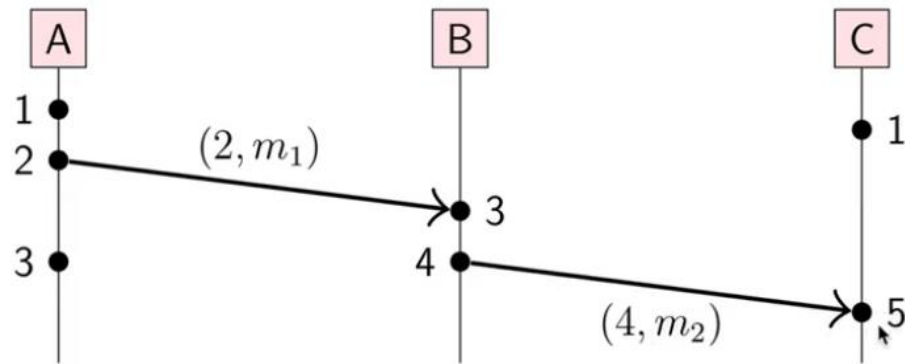
- Count number of events occurred

Causality

- When $a \rightarrow b$, then a might have caused b
- When $a \parallel b$, we know that a cannot have caused b
- Happens-before relation encodes potential causality
- Physical clock useful, but may be inconsistent with causality
- Logical clock designed to capture causal dependencies

Clock and Timestamp

Lamport Clock algorithm



on initialisation **do**

$t := 0$

▷ 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**

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

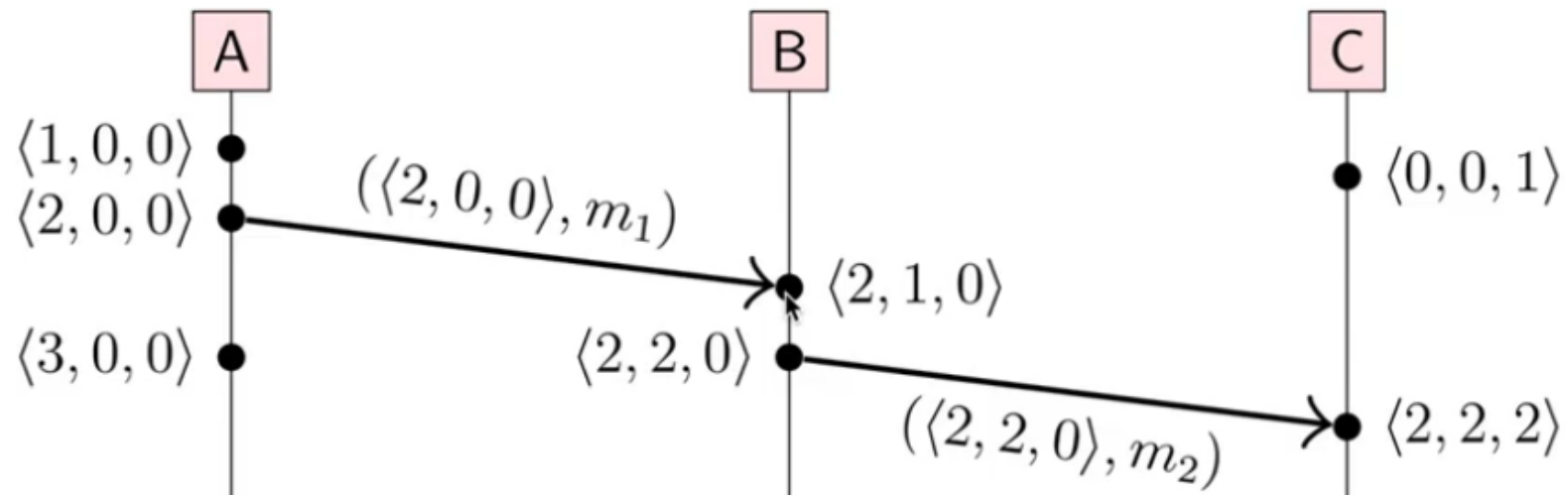
deliver m to the application

end on

Clock and Timestamp

Vector Clock

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



Clock and Timestamp

Hashgraph gossip protocol

