EECS 482
Introduction to Operating Systems

Winter 2018

Baris Kasikci

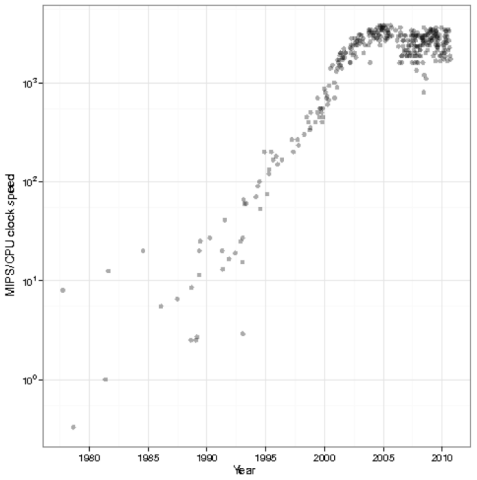
Thanks Manos Kapritsos for the slides

Making a distributed system look like a local system

- RPC: make request/response look like function call/return
- Distributed Shared Memory: make multiple memories look like a single memory
- Distributed File System: make disks on multiple computers look like a single file system
- Parallelizing compilers: make multiple CPUs look like one CPU
- Process migration (and RPC): allow users to easily use remote processors

Building distributed systems

- Why build distributed systems?
- Performance
  - Aggregate performance of many computers can be faster than that of (even a fast) single computer

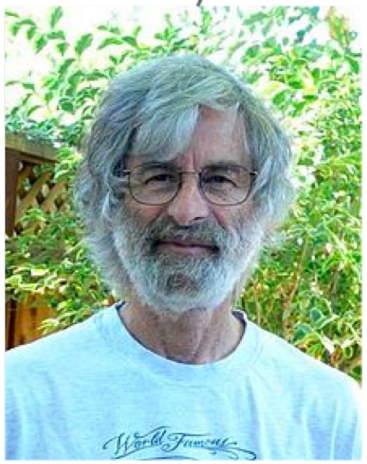


- Reliability
  - Try to provide continuous service, even if some computers fail
  - Try to preserve data, even if some storage fails

What is a distributed system?

"A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable."

Leslie Lamport



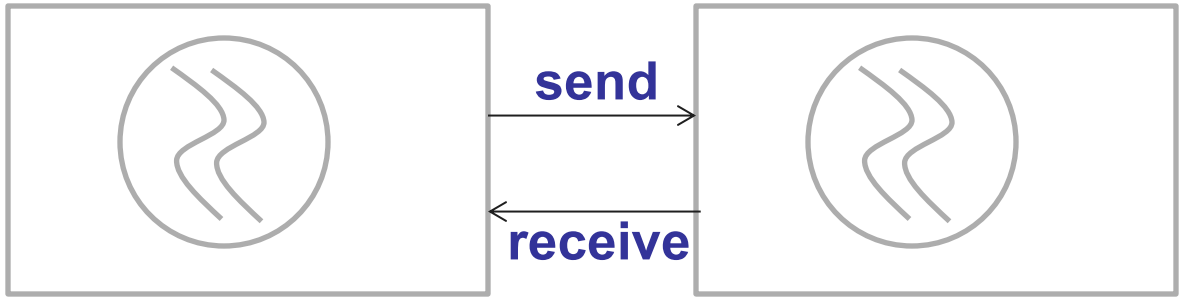
What is a distributed system?

- A collection of distinct processes that:
  - are spatially separated
  - communicate with each other by exchanging messages
  - have non-negligible communication delay
  - do not share fate

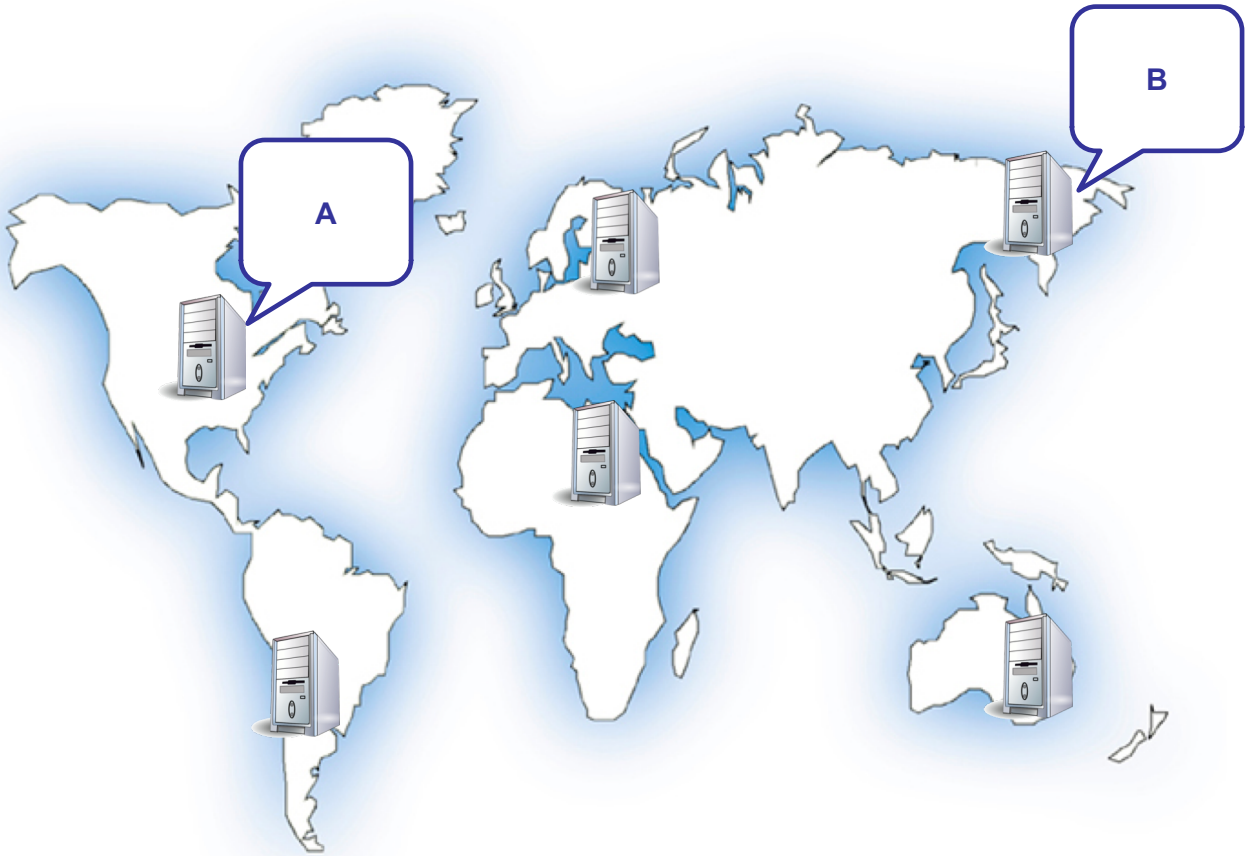
A distributed system is a concurrent system

- One multi-threaded process on one computer

- Several multi-threaded processes on several computers



What is a distributed system?



Ordering events in a distributed system

What does it mean for an event to "happen before" another event?

What is a distributed system?

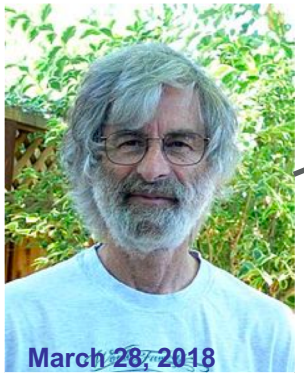
- A collection of distinct processes that:
  - are spatially separated
  - communicate with each other by exchanging messages
  - have non-negligible communication delay
  - do not share fate

What is a distributed system?

- A collection of distinct processes that:
  - are spatially separated
  - communicate with each other by exchanging messages
  - have non-negligible communication delay
  - do not share fate
  - have separate physical clocks

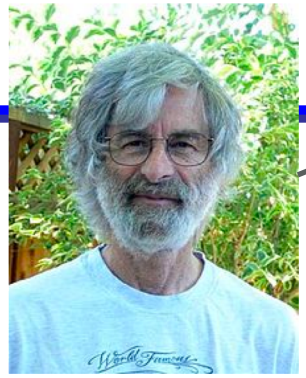
Single machine	Distributed system
• A single clock	• Each process has its own clock
• Each event has a timestamp	• Each clock runs at a different speed
• Compare timestamps to order events	• Cannot directly compare clocks

an absolute temporal ordering is not what you want in a distributed system anyway



Leslie Lamport

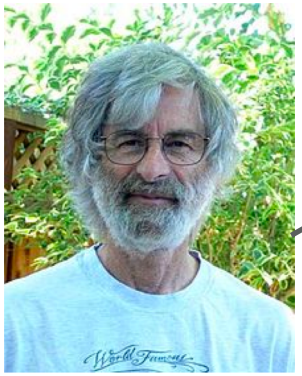
an absolute temporal ordering is not what you want in a distributed system anyway



Leslie Lamport

Why not?

Because temporal ordering is not observable. You cannot read two separate clocks simultaneously!

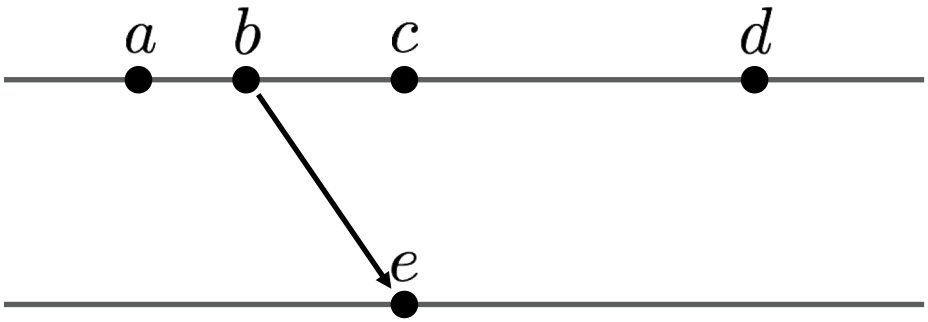


High-level point:
if a system is to meet a specification correctly, then that specification must be given in terms of events observable within the system

Ordering events without physical clocks

Modeling a process:

- A set of instantaneous events with an a priori total ordering
- Events can be local, sends, or receives.



Ordering events without physical clocks

"Happened-before" relation, denoted: $\longrightarrow$

Part 1

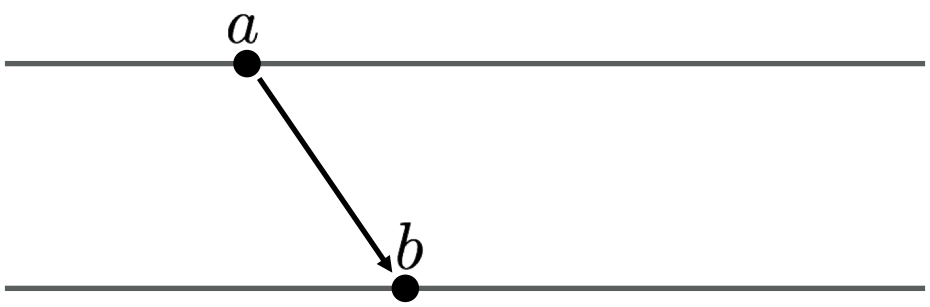
- If a and b are events on the same process and a comes before b, then $a \rightarrow b$

Ordering events without physical clocks

"Happened-before" relation, denoted: $\rightarrow$

Part 2

- If a is the sending of a message by one process and b is the receipt of the same message by another process, then $a \rightarrow b$

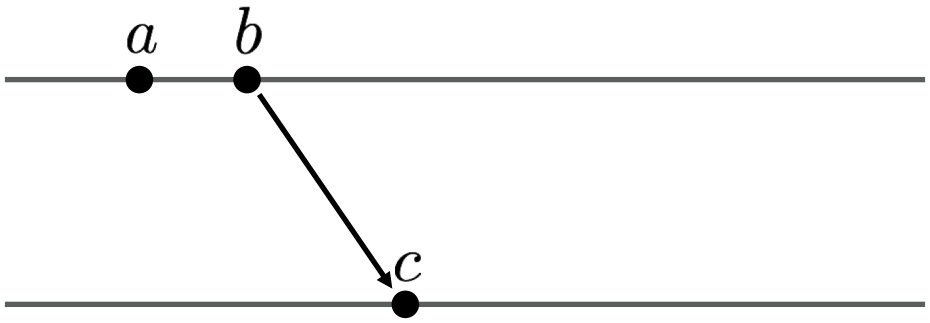


Ordering events without physical clocks

"Happened-before" relation, denoted: $\longrightarrow$

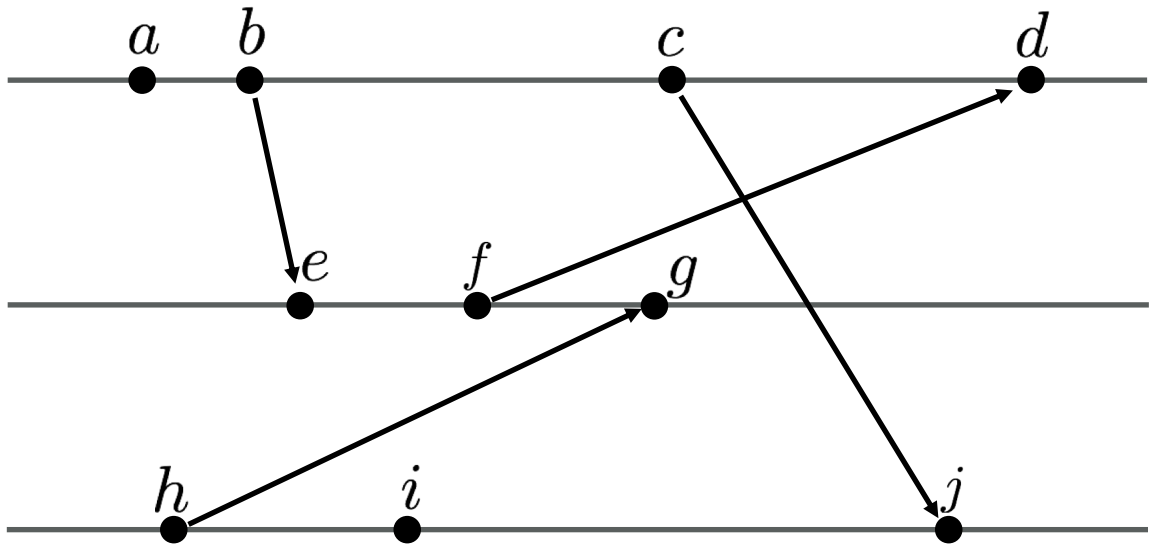
Part 3

- If $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$



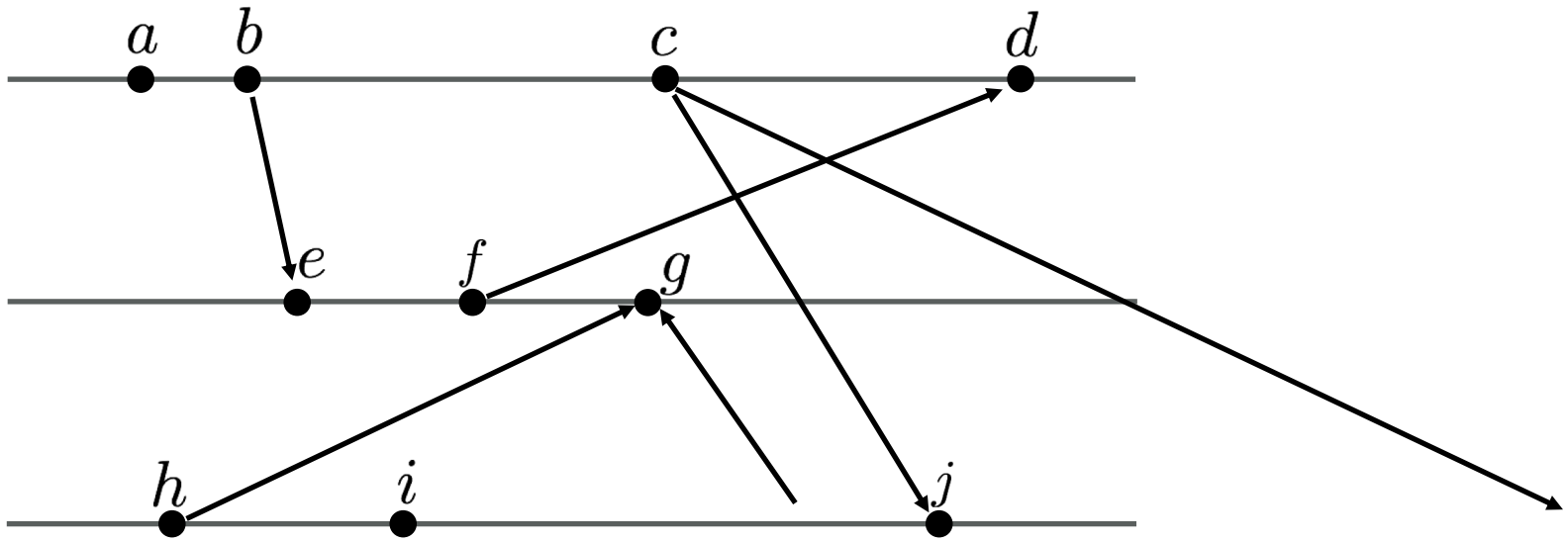
Ordering events without physical clocks

Putting it all together



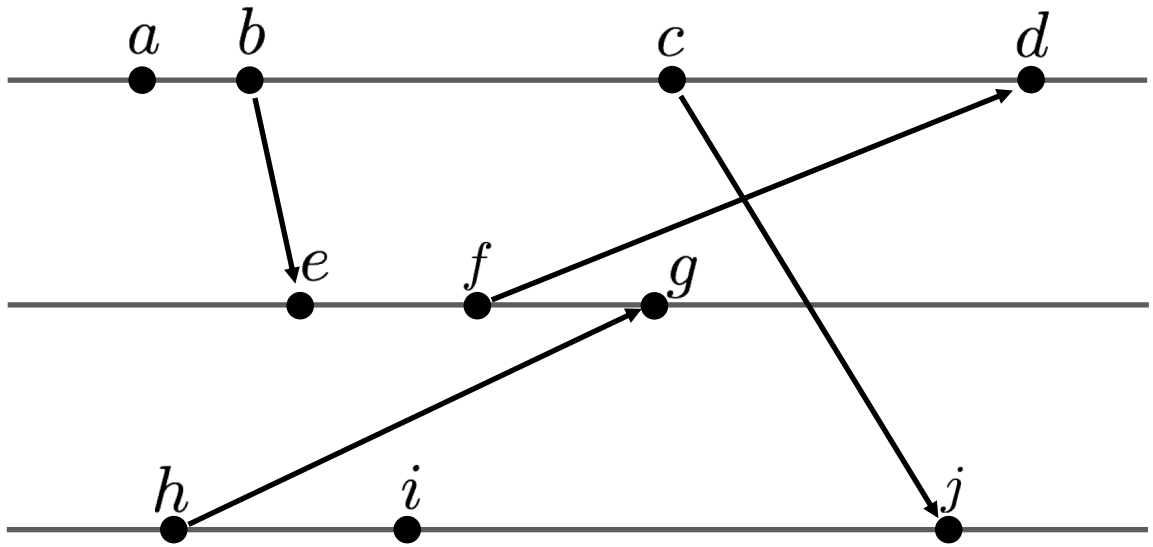
Ordering events without physical clocks

Can arrows go backwards?



Ordering events without physical clocks

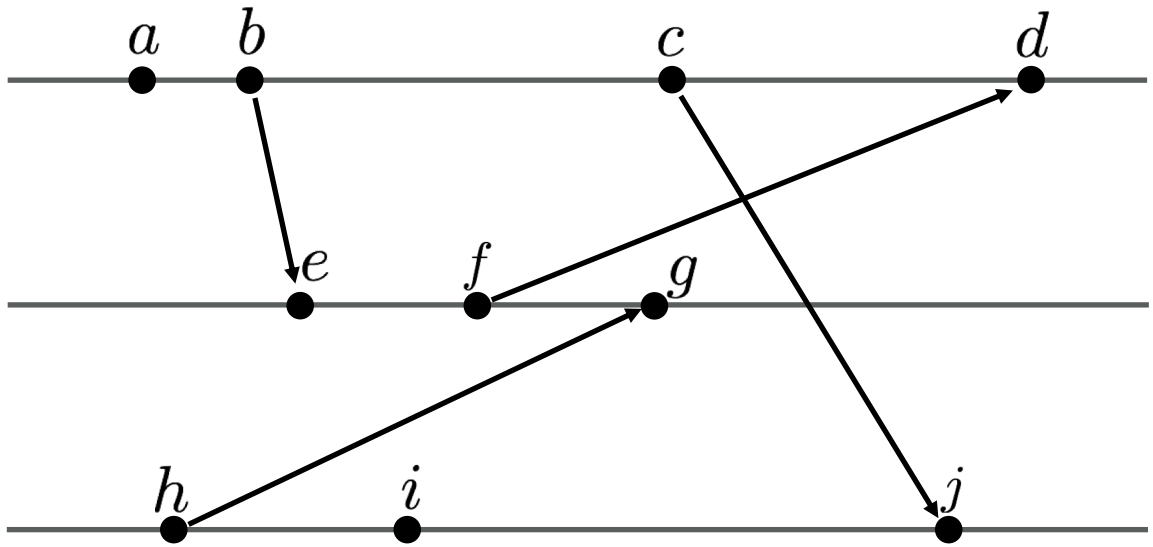
Can cycles be formed?



No, because the same event would happen at two different times

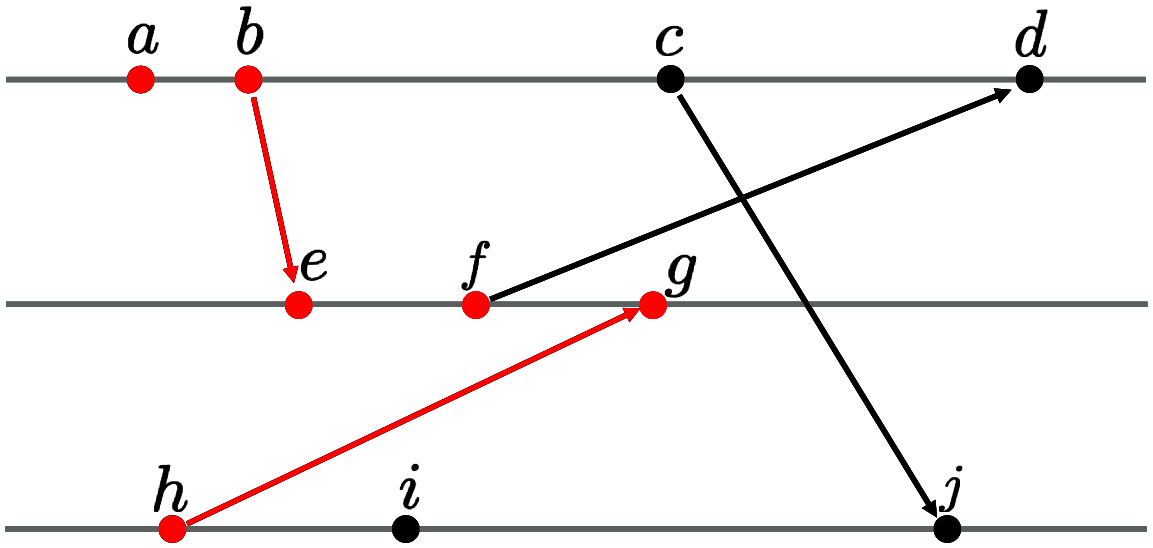
Ordering events without physical clocks

Are all events related by $\rightarrow$?



A partial order

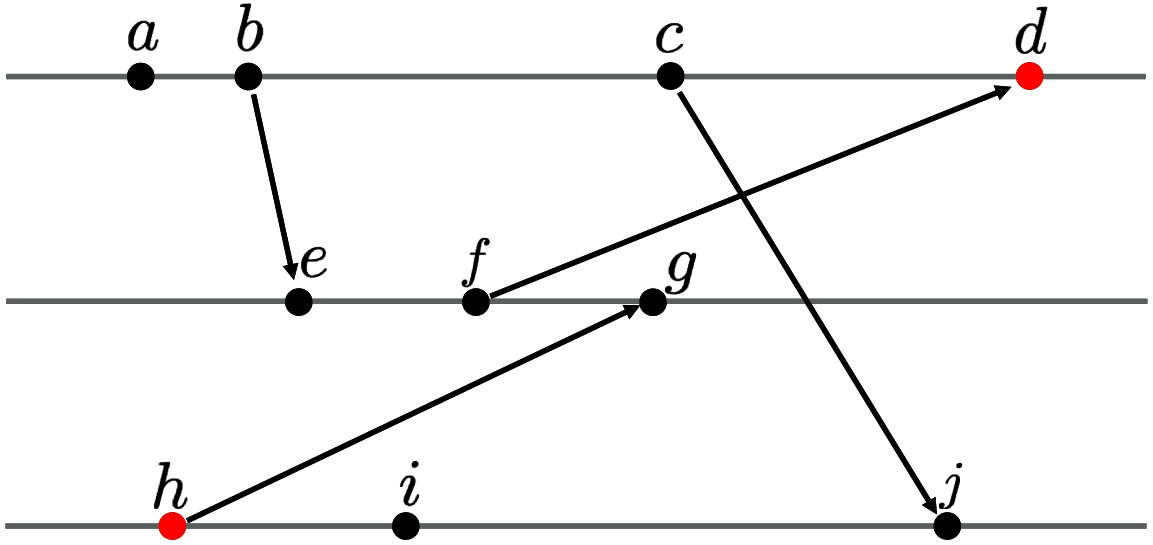
The set of events q such that $q \rightarrow p$ are the events that could have influenced p in some way



$$\{a, b, e, f, h\}$$

A ~~partial~~ CAUSAL order

If two events could not have influenced each other, it doesn't matter when they happened relatively to each other



h **and** d **are concurrent:** $h \not\mapsto d \quad d \not\mapsto h$

Goal

- Generate a total order that is consistent with the happened-before partial order
  - E.g. $a \quad b \quad c \quad d \ldots$

Lamport clocks

Define a function LC such that:

$$p \rightarrow q \Rightarrow LC(p) < LC(q)$$

(the Clock condition)

Lamport clocks

Define a function LC such that:

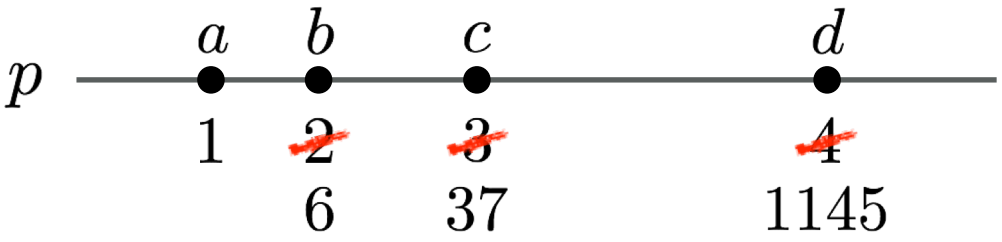
$$p \rightarrow q \Rightarrow LC(p) < LC(q)$$

(the Clock condition)

Implement LC by keeping a local LC_i at each process i

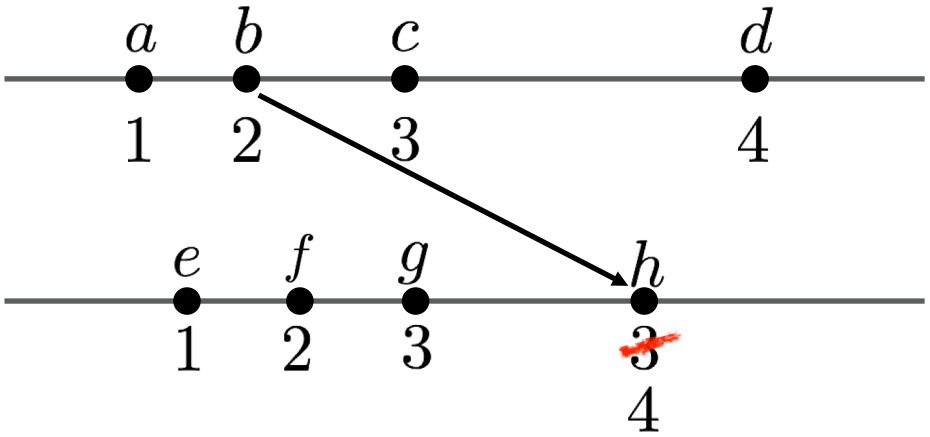
Lamport clocks

Single process



Lamport clocks

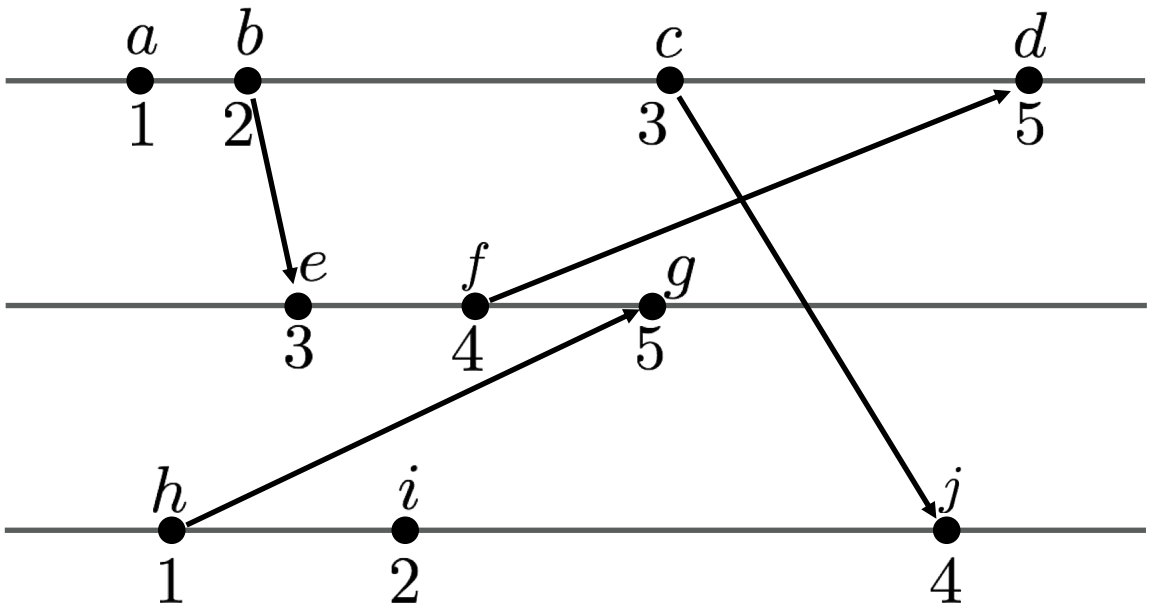
Across processes



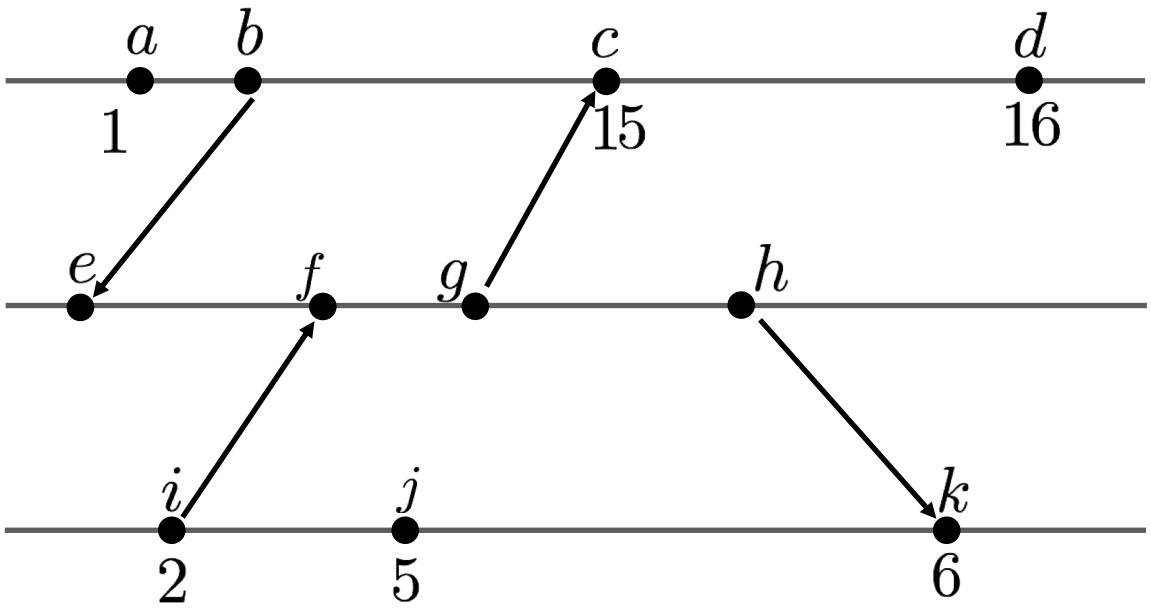
$$b \rightarrow h \Rightarrow LC(b) < LC(h)$$

$$g \rightarrow h \Rightarrow LC(g) < LC(h)$$

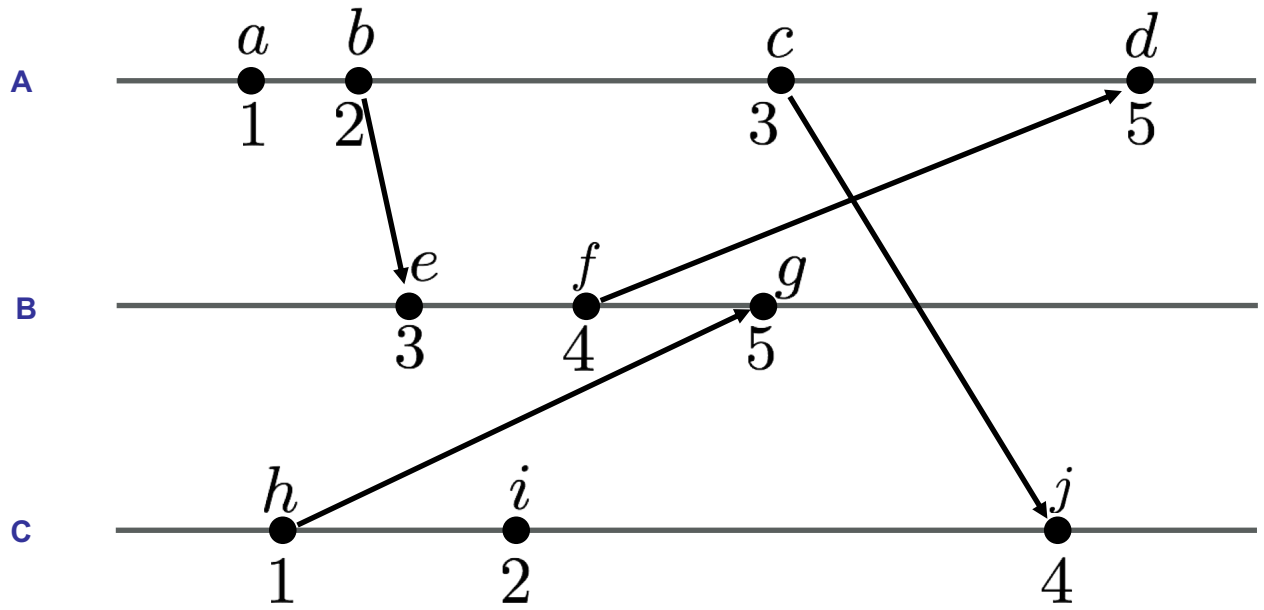
Putting it all together



Is this correct?

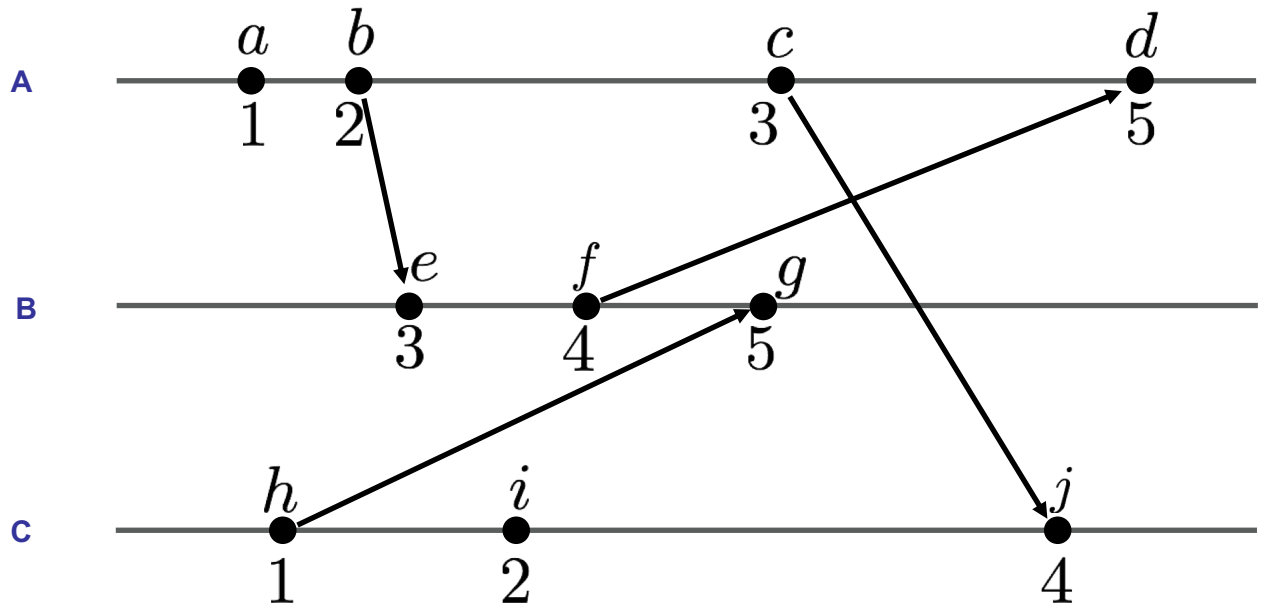


Generating a total order



- Order messages by LC
- Ties are broken by unique process ID

Generating a total order



- **Total order:** $a\ \ h\ \ b\ \ i\ \ c\ \ e\ \ f\ \ j\ \ d\ \ g$