EECS 482
Introduction to Operating Systems

Winter 2023

Baris Kasikci

Slides by: Harsha V. Madhyastha

OS abstraction of network

Hardware reality	Abstraction
Multiple computers connected via a network	Single computer
Machine-to-machine communication	Process-to-process communication
Unreliable and unordered delivery of finite messages	Reliable and ordered delivery of byte stream

Client-server

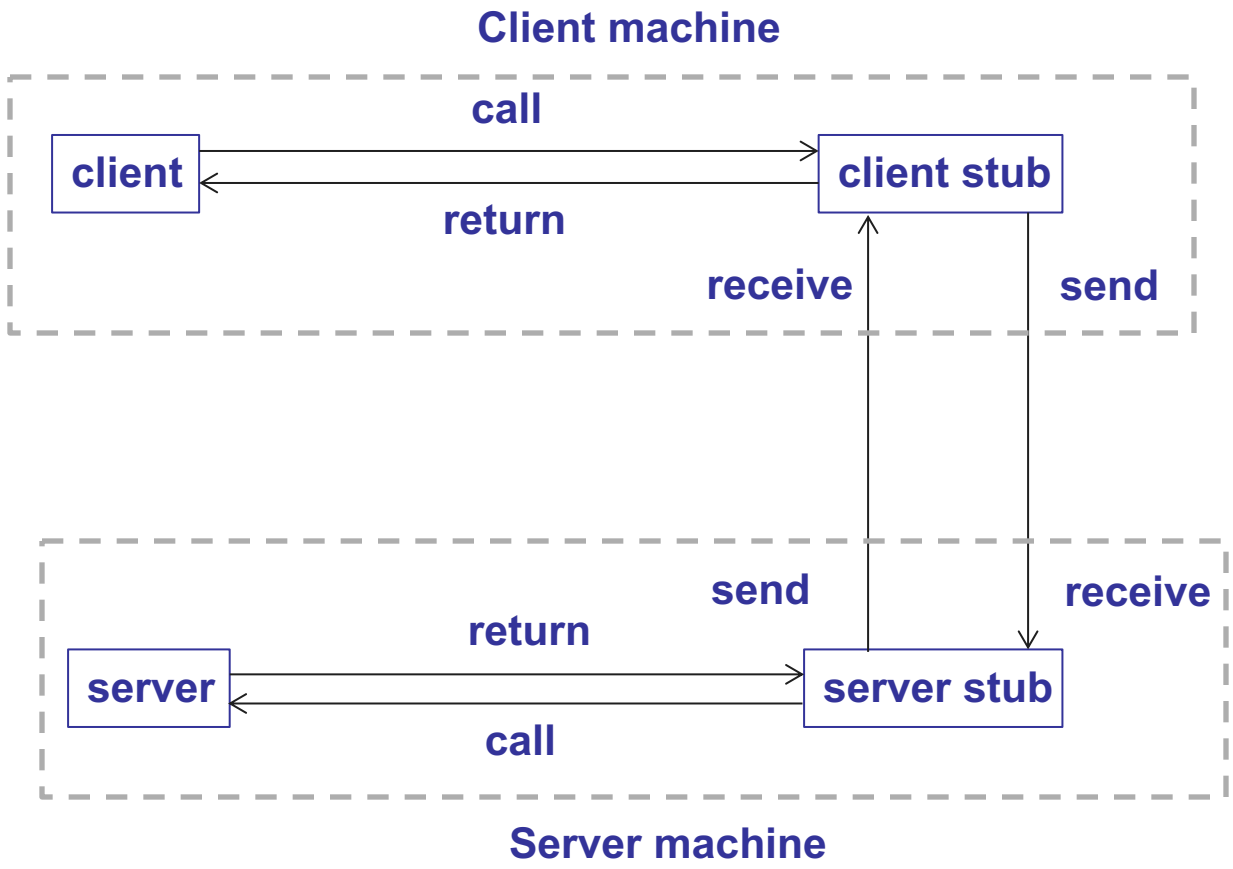
- Common way to structure a distributed application:
  - Server provides some centralized service
  - Client makes request to server, then waits for response
- Example: Web server
  - Server stores and returns web pages
  - Clients run web browsers, which make GET/POST requests
- Example: Producer-consumer
  - Server manages state associated with coke machine
  - Clients call `client_produce()` or `client_consume()`, which send request to the server and return when done
  - Client requests block at the server until they are satisfied

Remote Procedure Call

- Hide complexity of message-based communication from developers
- Procedure calls more natural for inter-process communication

- Goals of RPC:
  - Client sending request → function call
  - Client receiving response → returning from function
  - Server receiving request → function invocation
  - Server sending response → returning to caller

RPC abstraction via stub functions on client and server

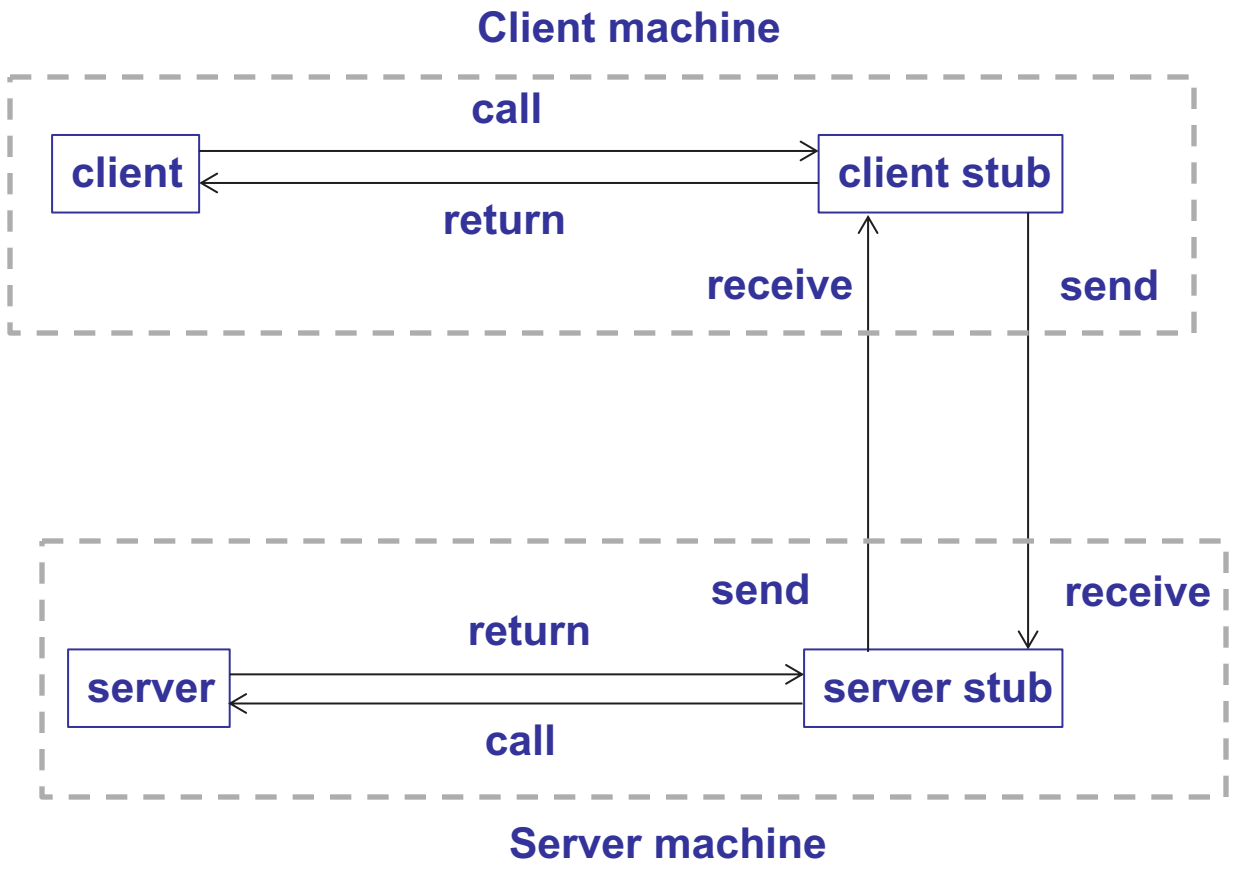


RPC stubs

- Client stub:

- Server stub:

RPC abstraction via stub functions on client and server



Producer-consumer using RPC

- Client stub
- Server stub

Generation of stubs

- Stubs can be generated automatically
- What do we need to know to do this?

- Interface description:
  - Types of arguments and return value

- e.g. rpcgen on Linux

RPC Transparency

- RPC makes remote communication look like local procedure calls
  - Basis of CORBA, Thrift, SOAP, Java RMI, …
  - Examples in this class?

- What factors break illusion?
  - Failures – remote nodes/networks can fail
    - » Independently of the client machine
  - Performance – remote communication is inherently slower
  - Service discovery – client stub needs to bind to server stub on appropriate machine

RPC Arguments

- Can I have pointers as arguments?
- How to pass a pointer as argument?
  - Client stub transfers data at the pointer
  - Server stub stores received data and passes pointer
- Challenge:
  - Data representation should be same on either end
  - Example: I want to send a 4-byte integer:
    - » 0xDE AD BE EF
    - » Send byte 0, then byte 1, byte 2, byte 3
    - » What is byte 0?

Endianness

- int x = 0xDE AD BE EF
- Little endian (Intel):
  - Byte 0 is 0xEF
- Big endian (Power PC):
  - Byte 0 is 0xDE

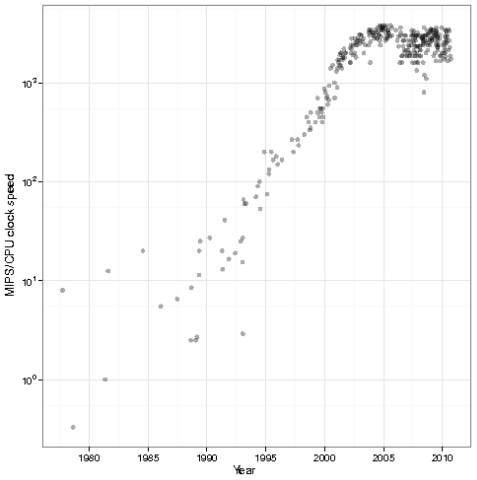
- If a little endian machine sends to a big endian:
  - 0xDE AD BE EF will become 0xEF BE AD DE

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

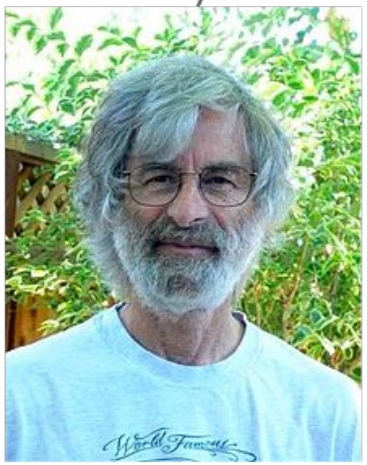


- Reliability
  - Try to provide continuous service, even if some computers fail
  - Try to preserve data, even if some storage fails

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

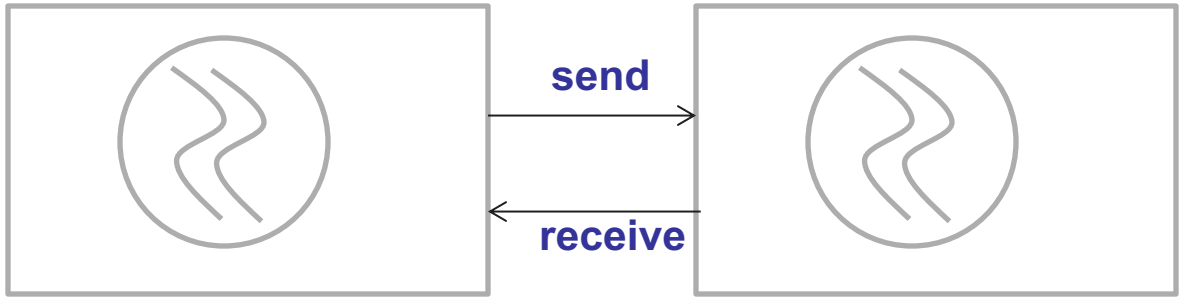
Concurrency and distribution

- Distributed programs are multi-threaded, since each computer has at least one thread
- Need two mechanisms to write multi-threaded programs
  - A way to share data between threads
  - Atomic primitives to synchronize threads
- How do we address this in a distributed system?
  - Sending/receiving of messages
- Can there be race conditions without shared data?

A distributed system is a concurrent system

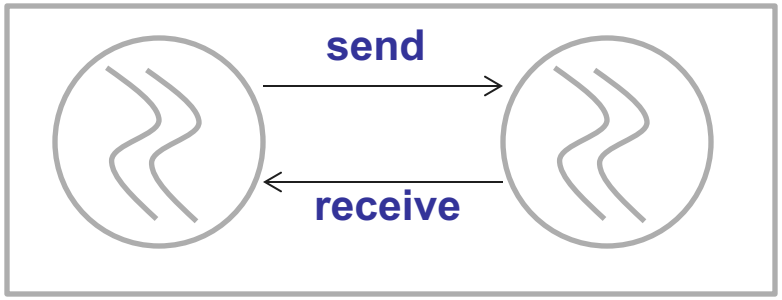
- One multi-threaded process on one computer

- Several multi-threaded processes on several computers



Structuring a concurrent system

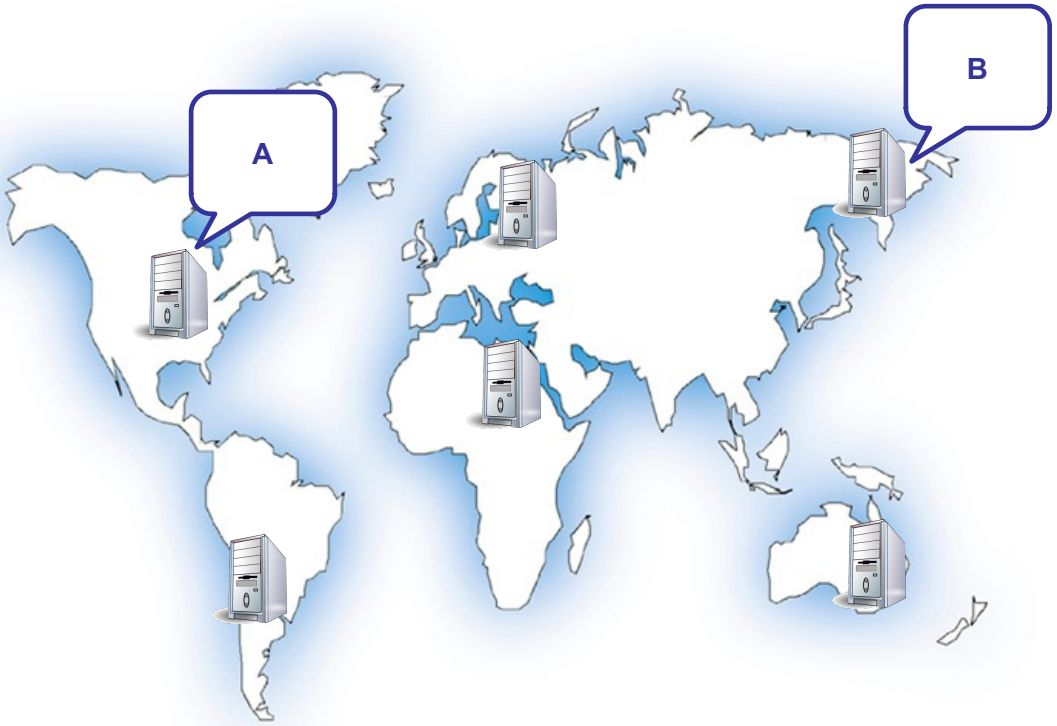
- Several multi-threaded processes on each of several computers



- Why separate threads on one computer into separate address spaces, then use send/receive to communicate and synchronize?
  - Protects modules from each other
- Microkernels
  - OS structure that separates OS functionality into several server processes, each in its own address space

Ordering events in a distributed system

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

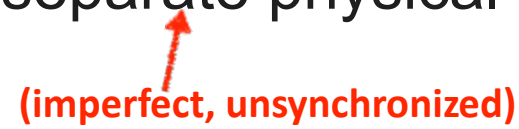


What is a distributed system?

- 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

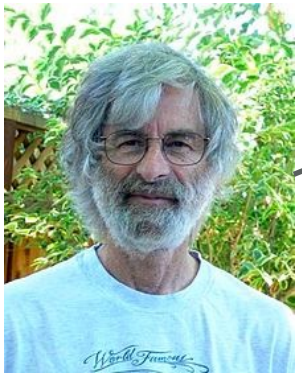
What is a distributed system?

- 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



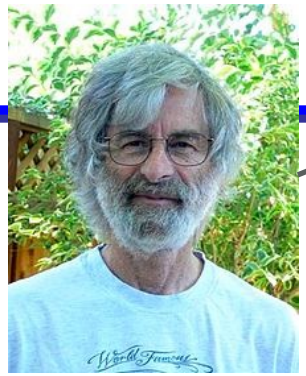
Single machine	Distributed system
• A single clock • Each event has a timestamp • Compare timestamps to order events	• Each process has its own clock • Each clock runs at a different speed • Cannot directly compare clocks

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



Leslie Lamport

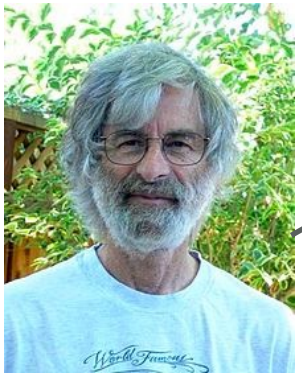
an absolute temporal ordering is not what you want in a distributed system anyway



Leslie Lamport

Why not?

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

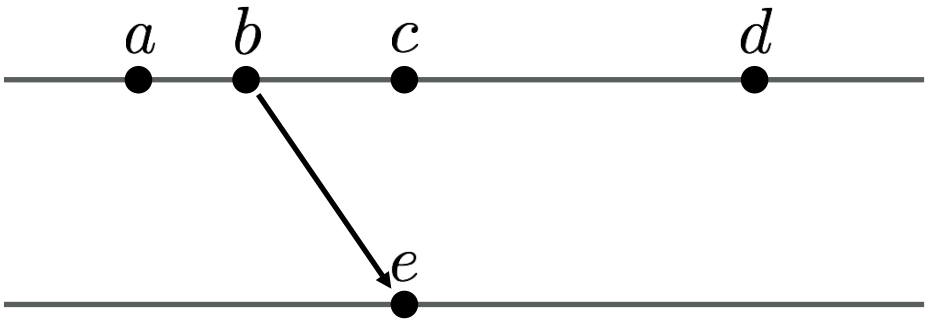


High-level point:
if a system is to meet a specification correctly, then that specification must be given in terms of events observable within the system

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

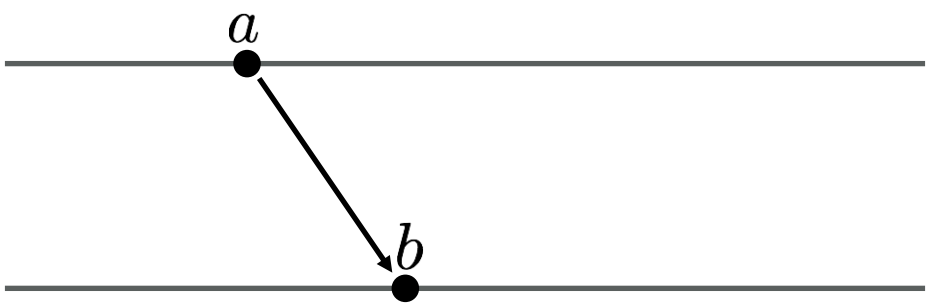
- If a and b are events on the same process and a comes before b, then $a \rightarrow b$

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 $a \rightarrow b$

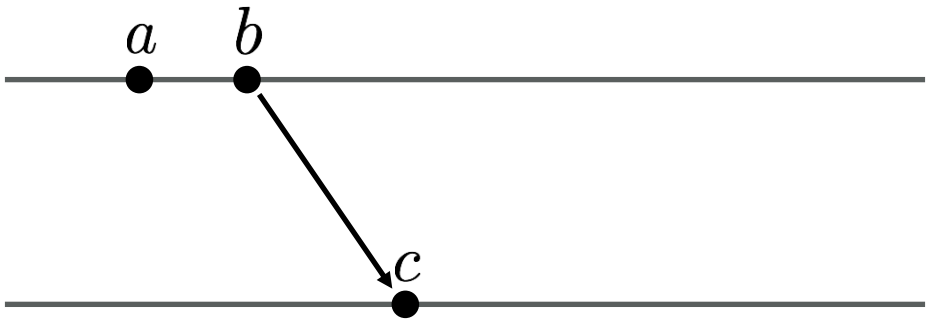


Ordering events without physical clocks

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

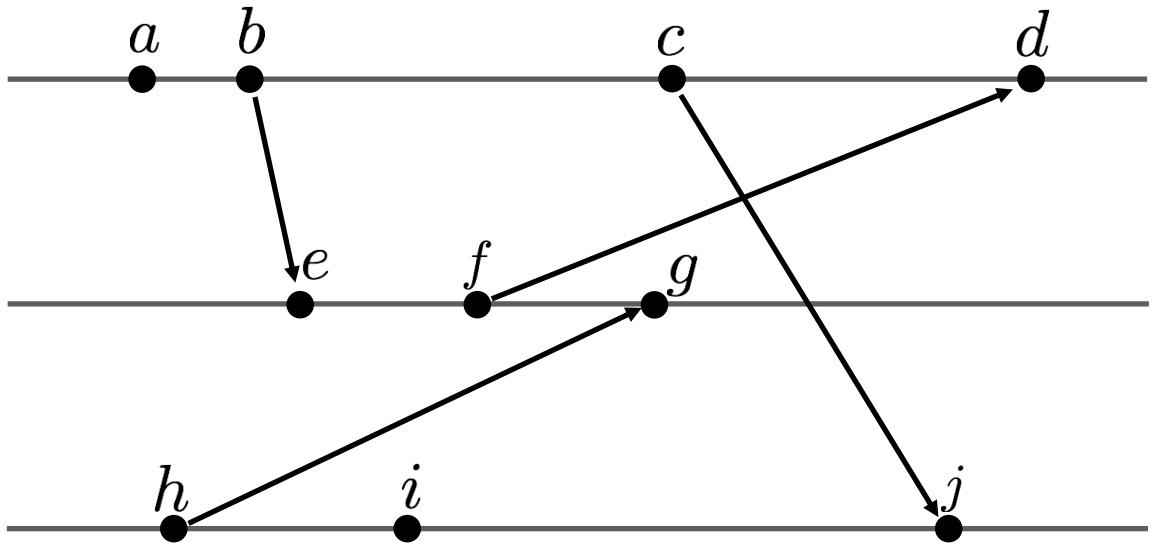
Part 3

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



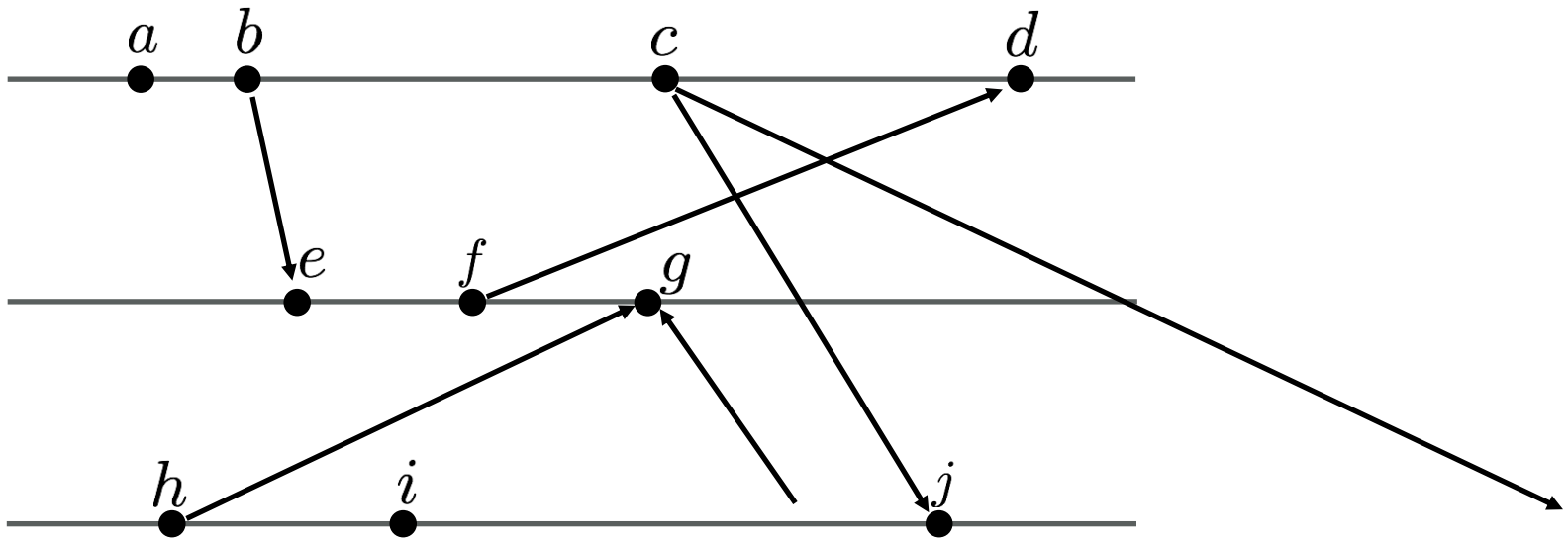
Ordering events without physical clocks

Putting it all together



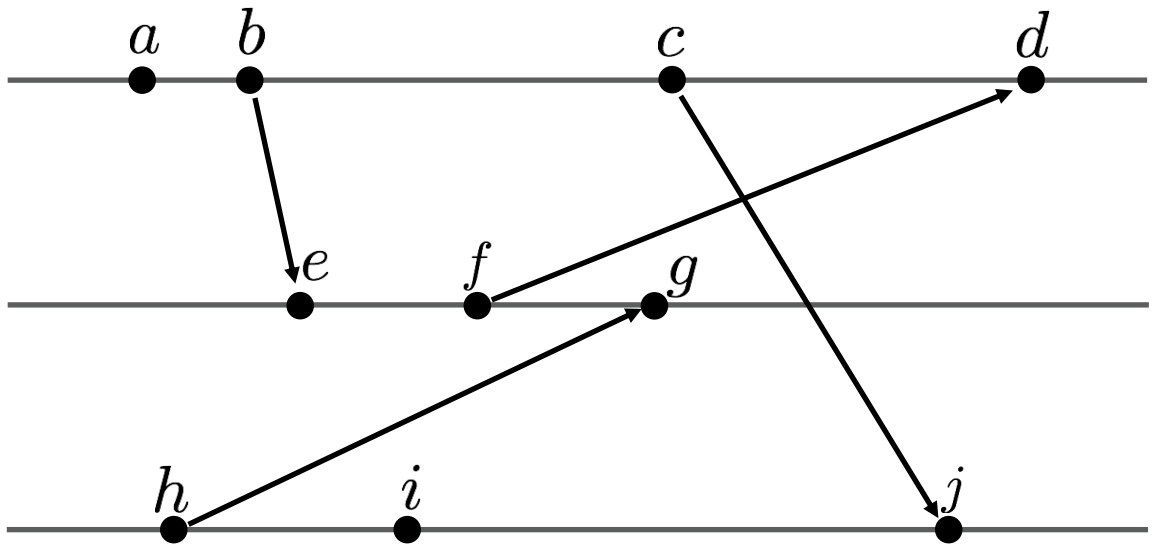
Ordering events without physical clocks

Can arrows go backwards?



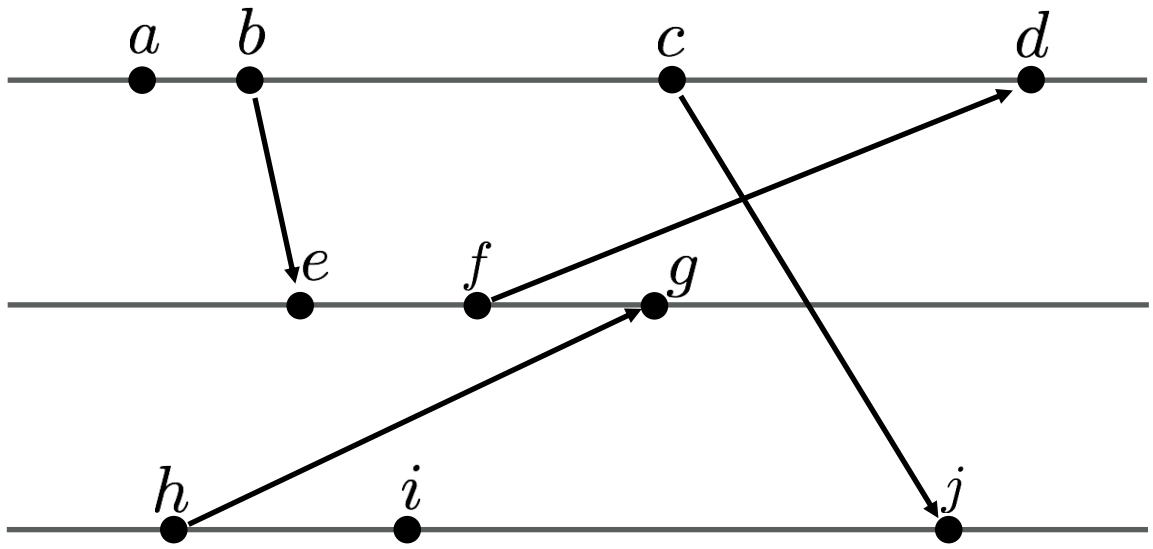
Ordering events without physical clocks

Can cycles be formed?



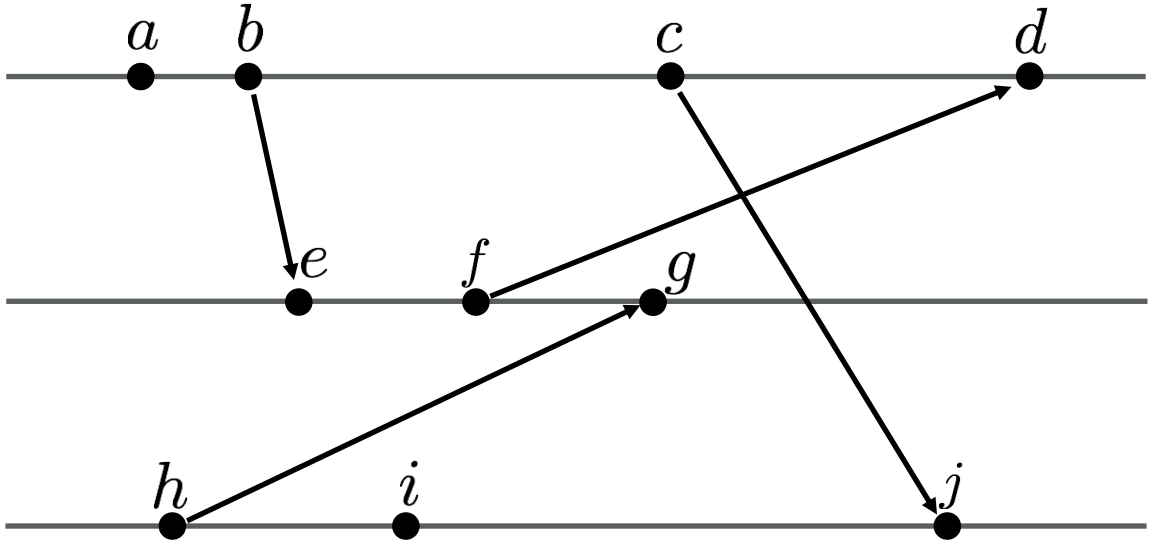
Ordering events without physical clocks

Are all events related by $\rightarrow$?



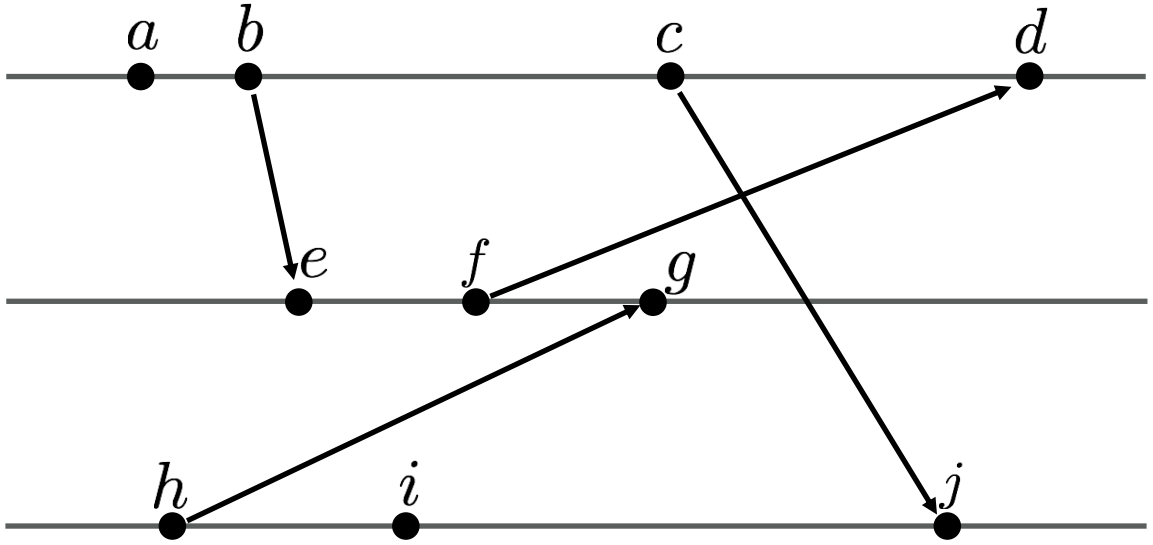
A partial order

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



A ~~partial~~ CAUSAL order

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



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:

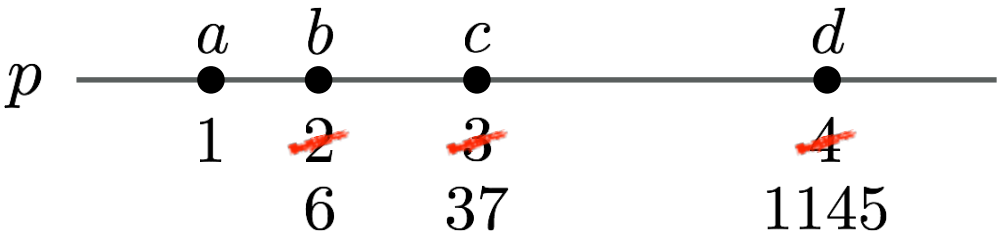
$$p \rightarrow q \Rightarrow LC(p) < LC(q)$$

(the Clock condition)

Implement LC by keeping a local LC_i at each process i

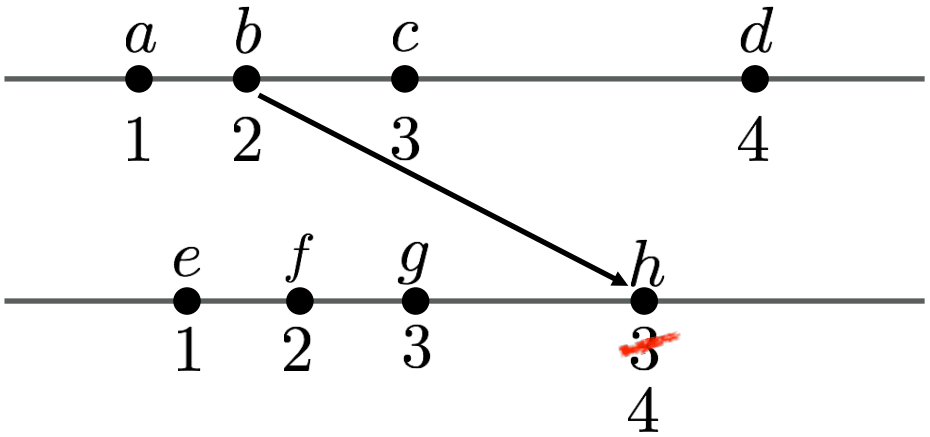
Lamport clocks

Single process



Lamport clocks

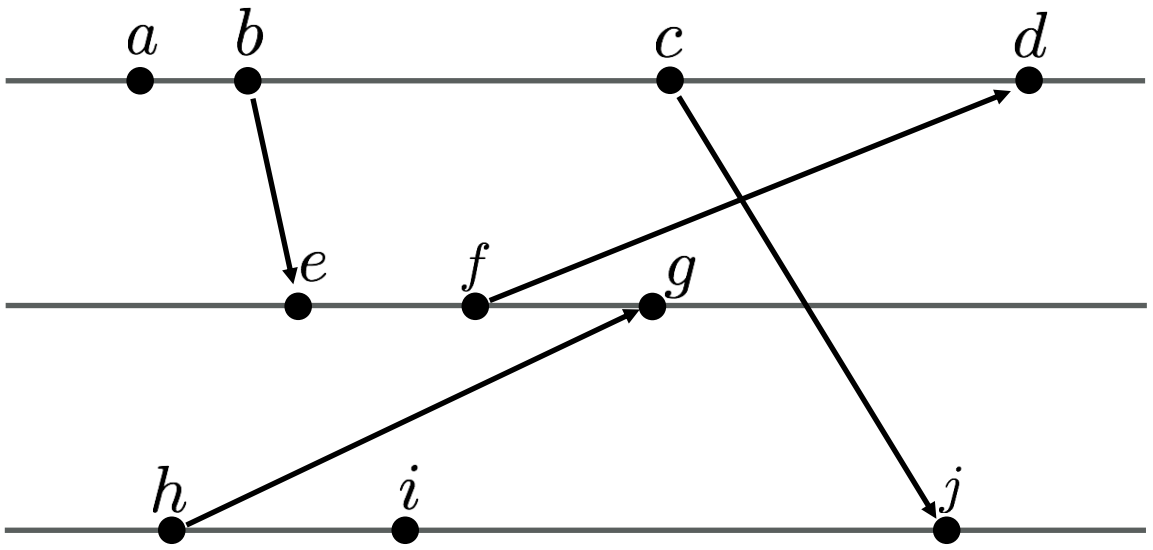
Across processes



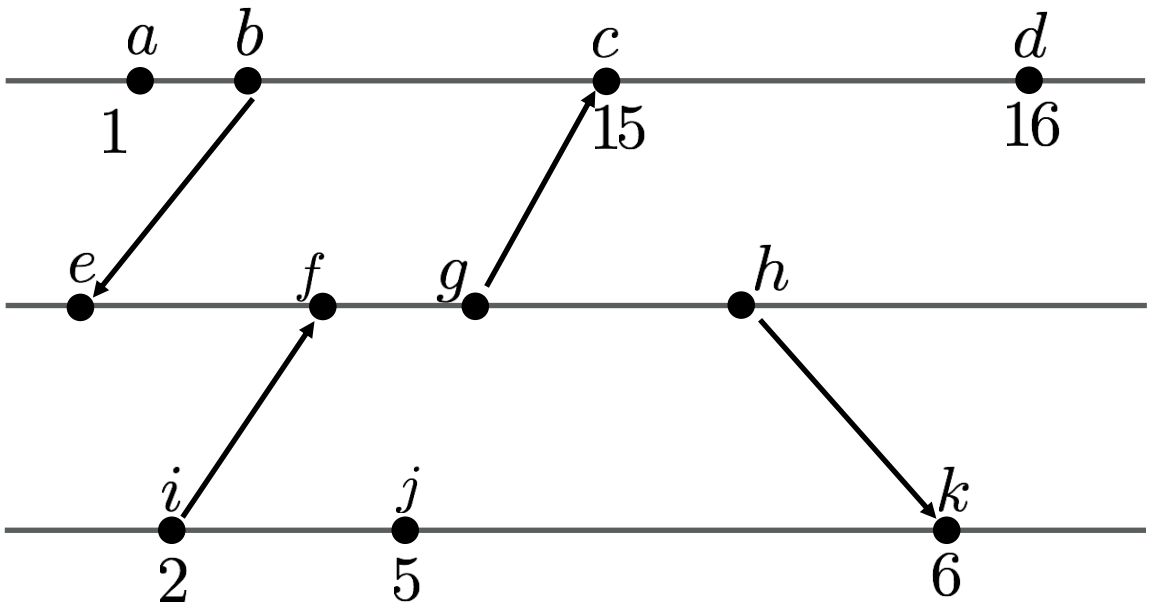
$$b \rightarrow h \Rightarrow LC(b) < LC(h)$$

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

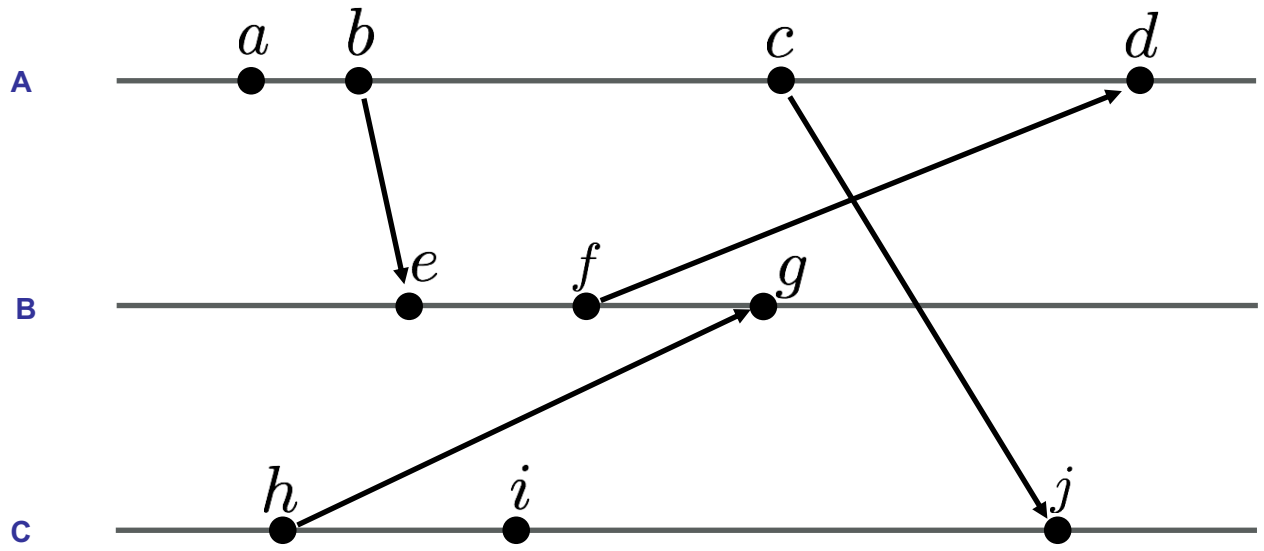
Putting it all together



Is this correct?

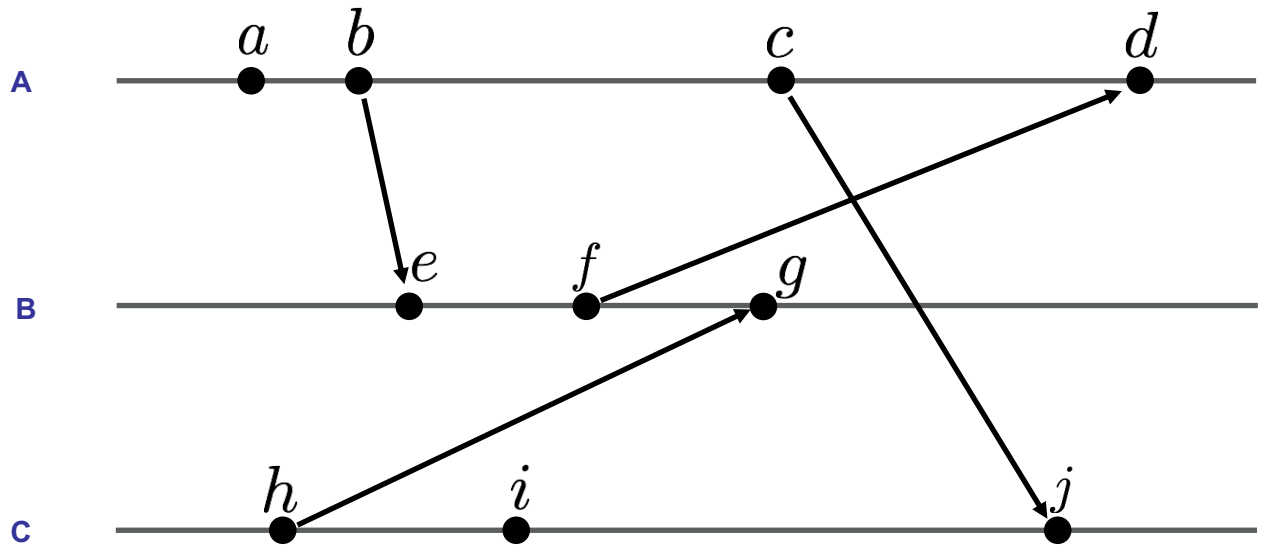


Generating a total order



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

Generating a total order



- **Total order:**