BRINGING ORDER TO THE GALAXY

THE EMPIRE
BRINGING ORDER TO THE GALAXY
SINCE: 1977
Synchrony vs Asynchrony

Synchronous systems
- Known bound on message delivery
- Known bound on processing speed
- Considered a strong assumption

Asynchronous systems
- No bound on message delivery
- No bound on processing speed
- Weak assumption = less vulnerable
- Asynchronous ≠ slow

This lecture: asynchronous + no process failures
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 one another by exchanging messages
- have non-negligible communication delay
- do not share fate
- have separate physical clocks

*(imperfect, unsynchronized)*
Non-distributed system

- A single clock
- Each event has a timestamp
- Compare timestamps to order events

Distributed system

- Each process has its own clock
- Each clock runs at a different speed
- Cannot directly compare clocks
Modeling a process:

- A set of instantaneous events with an a priori total ordering
- Events can be local, sends, or receives.
“Happened-before” relation, denoted: $\rightarrow$

**Part I**

- 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: →

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: $\rightarrow$

Part 3

- If $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$
ORDERING EVENTS WITHOUT PHYSICAL CLOCKS

Putting it all together
Can arrows go backwards?  Yes.
**Ordering events without physical clocks**

Can cycles be formed?

No, because an event would happen before itself.
Are all events related by →? **No.**
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\}
If two events could not have influenced each other, it doesn’t matter when they happened relatively to each other.

A causal partial order:

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

\[ h \leftrightarrow d, \quad d \leftrightarrow h \]
Goal: generate a **total** order that is consistent with the happened-before partial order
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

$p \quad a \quad b \quad c \quad d$

$\quad \quad \quad 1 \quad 2 \quad 3 \quad 4$

$\quad \quad \quad 6 \quad 37 \quad 1145$
Across processes

\[
\begin{align*}
&b \rightarrow h \Rightarrow LC(b) < LC(h) \\
&g \rightarrow h \Rightarrow LC(g) < LC(h)
\end{align*}
\]
PUTTING IN ALL TOGETHER
IS THIS CORRECT?

No.
Generating a total order

- Order messages by LC
- Ties are broken by unique process ID
Lamport clocks implement the Clock condition

\[ p \rightarrow q \Rightarrow LC(p) < LC(q) \]

But is that all we need?
ADMINISTRIVIA

• Send me a picture of yourself by next class
  • Subject: [EECS591] Picture of <first name> (preferred) <last name>

• Make sure to subscribe to our Piazza forum
  • Announcements, discussion, etc.

• At capacity
FIFO DELIVERY

FIFO delivery

$$send_i(m) \rightarrow send_i(m') \Rightarrow deliver_j(m) \rightarrow deliver_j(m')$$
Causal delivery

When more processes are involved, causal delivery is needed:
\[ \text{send}_i(m) \rightarrow \text{send}_j(m') \Rightarrow \text{deliver}_k(m) \rightarrow \text{deliver}_k(m') \]
Gap detection: Given two events $e$ and $e'$, where $LC(e) < LC(e')$, determine whether some other event $e''$ exists such that

$$LC(e) < LC(e'') < LC(e')$$
Gap detection: Given two events $e$ and $e'$, where $LC(e) < LC(e')$, determine whether some other event $e''$ exists such that 

$$LC(e) < LC(e'') < LC(e')$$

Lamport clocks don’t provide gap detection!
How to implement causal delivery?

(in other words, when is it safe to deliver \( m' \)?)

a) Wait to receive a message with higher LC from each channel

b) Implement better clocks!
FROM CLOCKS TO STRONG CLOCKS

\[ p \rightarrow q \implies LC(p) < LC(q) \]

Clock condition

\[ p \rightarrow q \iff LC(p) < LC(q) \]

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

$$\theta(g) = \{a, b, e, f, h\}g$$
IMPLEMENTING STRONG CLOCKS
(the hard way)

- Initialize $\theta := \emptyset$
- For send and local events $e$, $\theta(e) := \theta \cup \{e\}$
- For receive events $e = \text{recv}(m)$, $\theta(e) := \theta \cup \{e\} \cup \theta(m)$
IMPLEMENTING STRONG CLOCKS
(the hard way)

Strong clock condition: $p \rightarrow q \iff \theta(p) \subset \theta(q)$
IMPLEMENTING STRONG CLOCKS
(the hard way)

Strong clock condition: $p \rightarrow q \iff \theta(p) \subset \theta(q)$
Vector clocks

Each process keeps a vector of natural numbers $\mathbf{VC}$, one for each process.

**Update rules**

If $e_i$ is a local or send event at process $i$:

$$VC(e_i)[i] := VC[i] + 1$$

If $e_i$ is a receive event of message $m$:

$$VC(e_i) := \max\{VC, VC(m)\}$$
$$VC(e_i)[i] := VC[i] + 1$$