CSE138 Lecture 3

- happens-before recap
- <u>partial orders</u> / total orders
- Lamport clocks
- Vector clocks

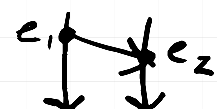
if time

happens-before relation ($\longrightarrow$)

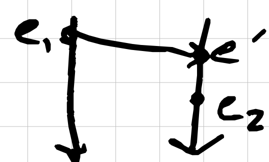
Given events e_1 and e_2 in an execution, we say $e_1 \longrightarrow e_2$ if:

- e_1 and e_2 on same process with e_1 before e_2

- e_1 is a send and e_2 is a receive



- ★ there is some event e' such that $e_1 \rightarrow e'$ and $e' \rightarrow e_2$



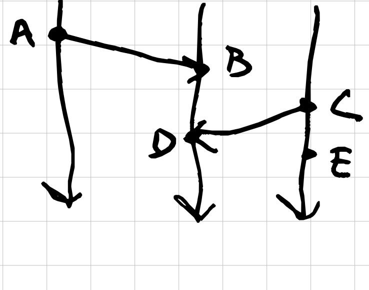
"partial order" is short for partially ordered set:

- A set S
- A binary relation $\leq$ that lets you compare elements of S and has the following properties:
  - Reflexivity: For all $a \in S$, $a \leq a$.
  - Antisymmetry: For all $a, b \in S$, if $a \leq b$ and $b \leq a$ then $a = b$.
  - Transitivity: For all $a, b, c \in S$, if $a \leq b$ and $b \leq c$, $a \leq c$.

not true for → !

vacuously true for → .

true in a meaningful way for → .



$\{A, B, C, D, E\}$

OK, what about a *real* *partially* ordered set?

set of all subsets of:

$\{\square, \triangle, \circ\}$

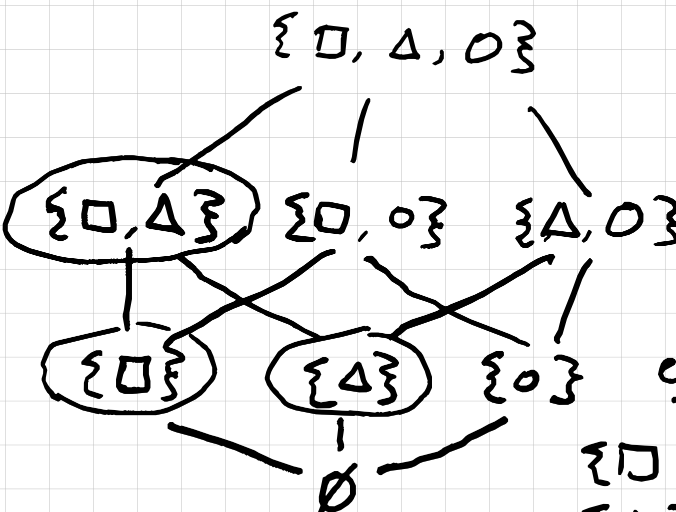
$$S = \{\ \emptyset,\ \{\square\},\ \{\triangle\},\ \{\circ\},\ \{\square, \triangle\},\ \{\square, \circ\},\ \{\triangle, \circ\},\ \{\square, \triangle, \circ\}\ \}$$

ordering relation: $\subseteq$

examples: $\{\square\} \subseteq \{\square, \circ\}$

Hasse diagram:

$\subseteq$ is a partial order on the set S.



example of partiality:

$\{\square\} \not\subseteq \{\triangle\}$

$\{\triangle\} \not\subseteq \{\square\}$

what about a totally ordered set?

example: $\mathbb{N}$, ordered by the usual $\le$ relation.

Hasse diagram:

any two elements of $\mathbb{N}$ you care to pick are ordered by $\le$.

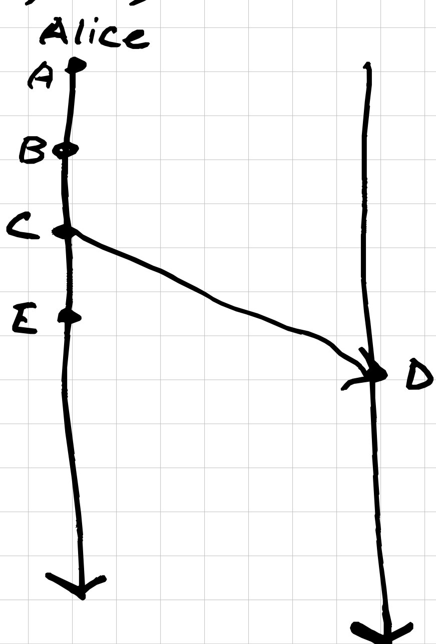
2 and 3

$2 \le 3$

or

$3 \le 2$.

$(\mathbb{N}, \le)$



Logical clocks, and in particular Lamport clocks.

A way of assigning natural numbers to events.

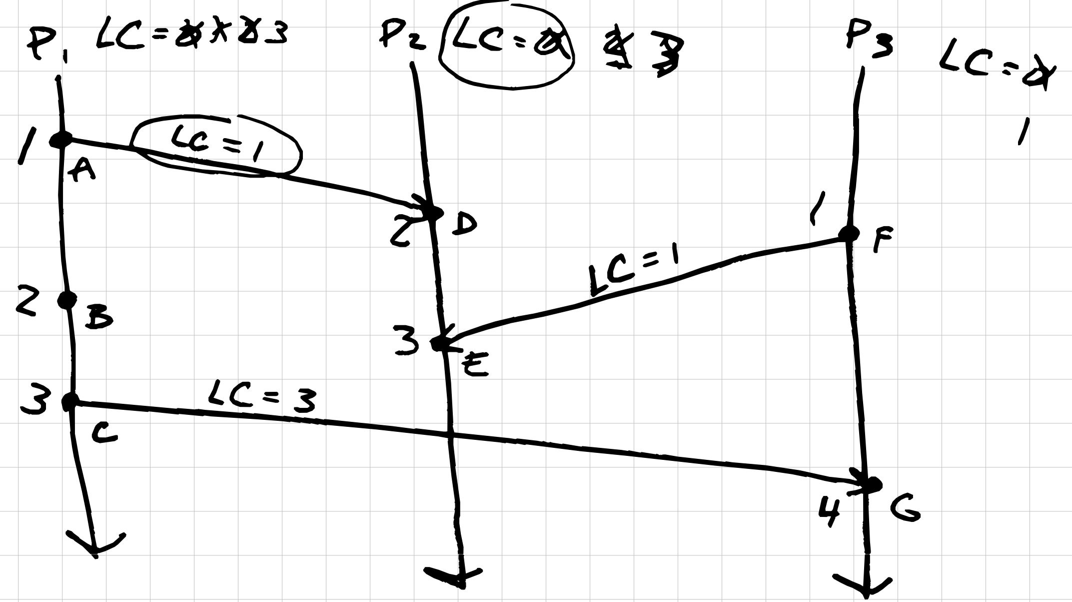
$LC(A) = 3$

Clock condition:

if $e_1 \to e_2$, then $LC(e_1) < LC(e_2)$.

we say that Lamport clocks are <u>consistent with</u> the happens-before relation.

(fancier way to say it: "consistent with causality")



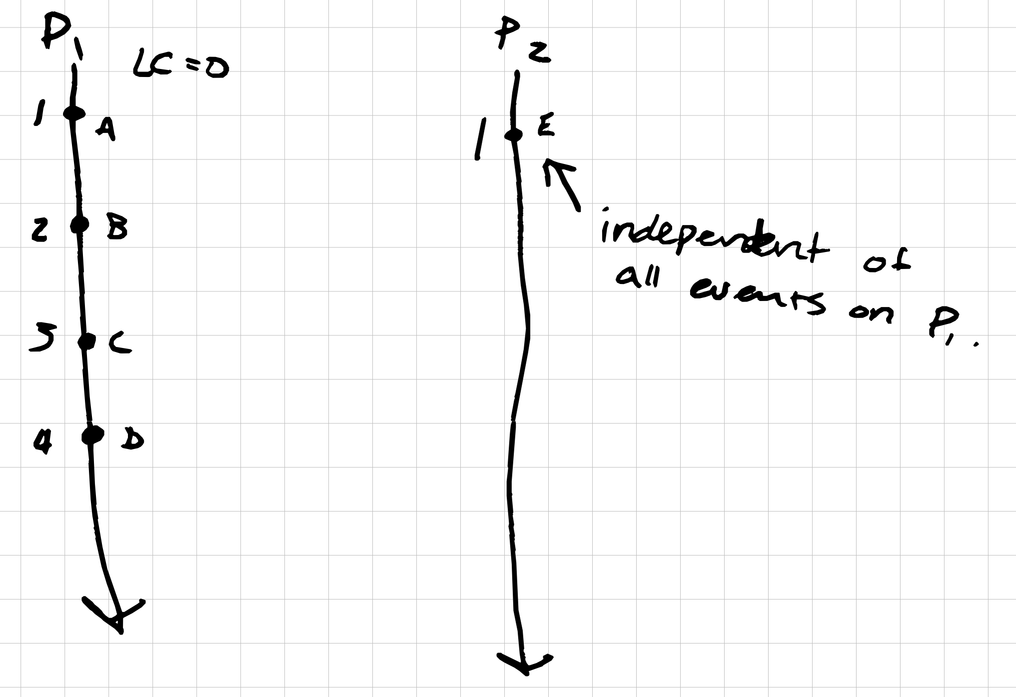
LC algorithm:

1) Every process will keep a counter, initially 0.
2) On every event, a process increments its counter by 1.
3) When sending a message, a process includes its counter with the message.
4) When receiving a message, set your clock to max(local clock, received clock) + 1.

What about the other way around?

if $LC(e_1) < LC(e_2)$, do we know that $e_1 \to e_2$?

No!



we don't know that $E \to D$.
what's actually true is that E and D are concurrent.

we know only the clock condition:
if $e_1 \to e_2$ then $LC(e_1) < LC(e_2)$.

Clock condition:

$$\boxed{\underbrace{e_1 \to e_2}_{P} \Rightarrow \underbrace{LC(e_1) < LC(e_2)}_{Q}}$$

$P \Rightarrow Q$

Contrapositive:

$\neg Q \Rightarrow \neg P$.

contrapositive of the clock condition:

$$\neg(LC(e_1) < LC(e_2)) \Rightarrow \neg(e_1 \to e_2)$$

either $LC(e_1) = LC(e_2)$ or $LC(e_1) > LC(e_2)$	$\Rightarrow$	either e_1 and e_2 are concurrent, or $e_2 \to e_1$.

$LC(A) < LC(B) \overset{?}{\Rightarrow} A \to B$

No, but either:

$A \to B$ or A and B are concurrent.

At least you know that B didn't happen before A.

But this would be nice:

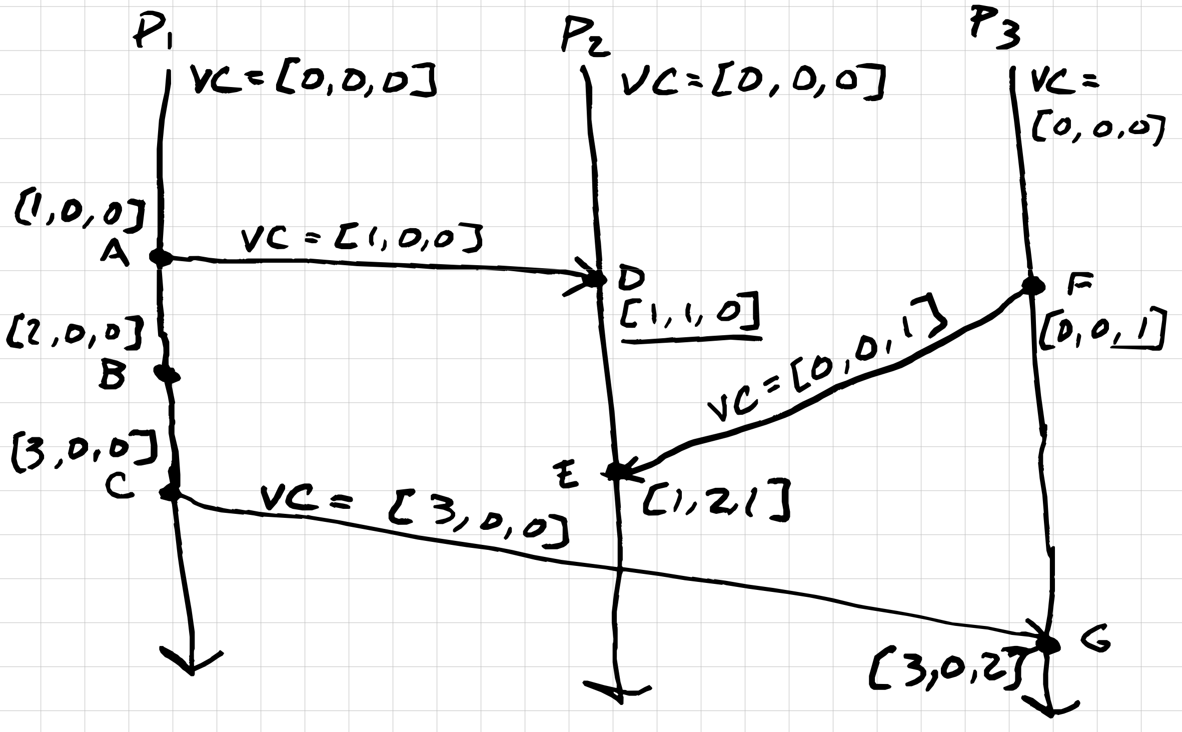
$A \to B \Rightarrow LC(A) < LC(B)$ (clock condition)

$LC(A) < LC(B) \Rightarrow A \to B$ (inverse clock condition)

LCs don't give you this, but vector clocks do!

vector clocks -

An event is associated with a vector of natural numbers.



vector clock algorithm

1) Every process maintains a vector of natural numbers, initialized to 0. The length of the vector is the number of processes.

2) On every event, a process, a process increments its own position in its clock.

3) when sending a message, a process attaches its VC to the message.

4) when receiving a message, you take the maximum of the received VC and your own local VC. (and increment your own position by 1, because a receive is an event)

Take the pointwise maximum

$\max([1, 12, 4], [7, 0, 2])$

$= [7, 12, 4]$