LECTURE 2

Friday, September 8, 2017 8:37 PM

RECAP

HAPPENS BEFORE RELATION

$e \to f$.

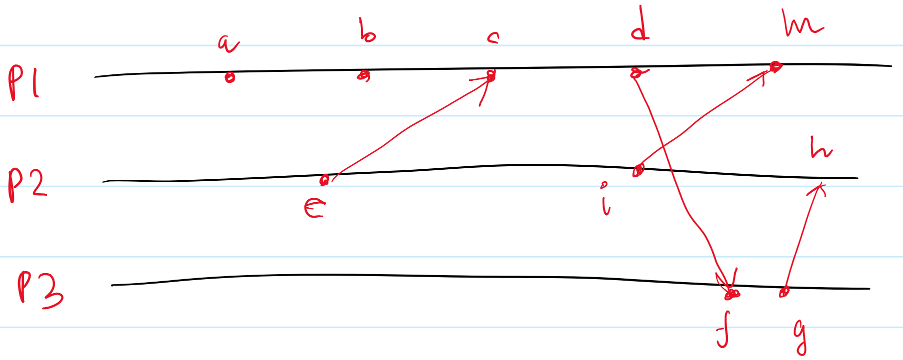
(I) If a and b takes place on same process & a happens first, $a \to b$

(II) If a is send(m) and b is receive(m), $a \to b$

(III) If $a \to b$ and $b \to c$, by transitivity $a \to c$

THIS IS NOT A TOTAL ORDERING

$a \| b$ iff $\neg(a \to b)$ and $\neg(b \to a)$



All events happening before f: a, b, c, e, d

All events concurrent to i: a, b, c, d, f, g

LOGICAL CLOCK

Each process P_i has own logical clock C_i

- assign a number to each event that occurs at p_i
- $c_i(e) \Rightarrow$ time of event e

System of clocks = global function c. : assigns time to every event.

$C(e) = C_i(e)$ if e takes place on p_i

if $a \to b$ then $c(a) < c(b) \Rightarrow$ LAMPORTS' CLOCK CONDITION.

Clocks need to satisfy 2 conditions :

(i) if a and b takes place on same process p_i:

and $a \to b$, then $c(a) < c(b)$.

(ii) If $a = send(m)$, $b = receive(m)$ then,

$$c(a) < c(b)$$

① and ② are gonna implement Lamport's clock condition. because both $\rightarrow$ and $<$ are transitive relations.

$\boxed{\text{CLOCK IMPLEMENTATION}}$ (from perspective of a single process)

Initially $C_i = 0$

Increment C_i between any 2 successive events (tick) by 1

If event is send of message :

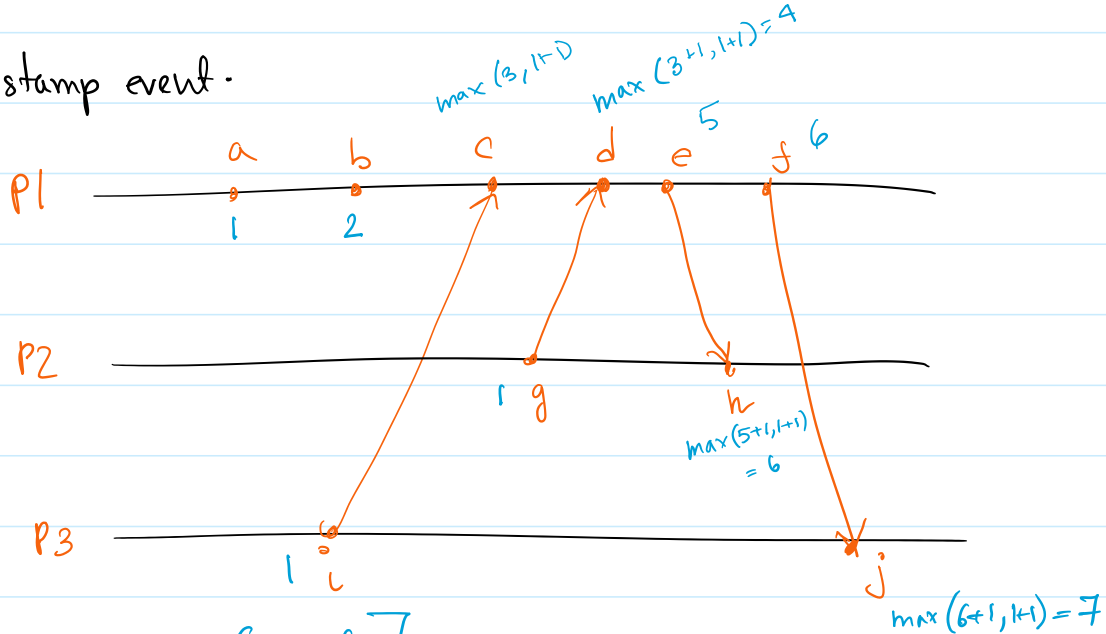
timestamp message with $T_m = C_i$

timestamp message with $T_m = C_i$

If event is receive of message with timestamp T_m:

$$C_i = \max(C_i, T_{m+1})$$

Timestamp event.



now $C(h) = 6$, $C(f) = 6$ — that means, we can have collisions

$\therefore$ C is a partial order

$\therefore$ If we timestamp only using Lamport's algorithm, we can get $\boxed{\text{ties}}$

⇒ TOTALLY ORDERED LAMPORT TIMESTAMPS

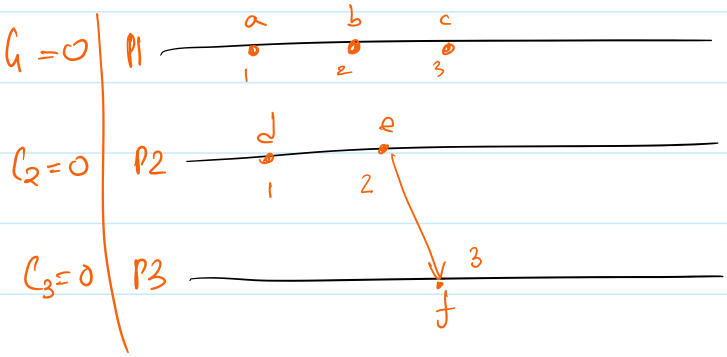
$C(a) = (i, C_i(a))$ event takes place at p_i

↓ process id

$C(a) < C(b)$ if either —
- (i) $C_i(a) < C_j(b)$
- (ii) $C_i(a) = C_j(b)$ and $i < j$.

In previous figure, now, $c(h) = (2, 6)$ $\therefore$ $c(h) < c(f)$

$c(f) = (3, 6)$ $\Rightarrow h \rightarrow f$



If $a \rightarrow b$, $c(a) < c(b)$

in this case,

$\neg (a \rightarrow e)$

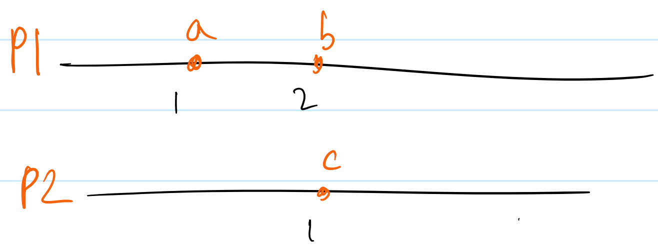
they are concurrent

WEAK CLOCK CONDITION

So it happens that I can't just look at the timestamps and determine a causal relationship. So this condition is not strong.

⇒ STRONG CLOCK CONDITION (maybe what we want?)

$$c(e) < c(f) \text{ iff } e \rightarrow f$$



$c(a) < c(b)$

but $\neg (c \rightarrow b)$

$a \rightarrow b \Rightarrow c(a) < c(b)$

$a \| c \Rightarrow c(a) = c(b)$

$b \| c \Rightarrow c(b) = c(c)$

Not possible using Integer Lamport Timestamps

Does totally ordered Lamport's Algorithm solve this?

No, Process id doesnot break causality, its just for breaking ties.

STRONG CLOCK CONDITIONS

$$c(e) < c(f) \text{ iff } e \to f$$

VECTOR CLOCKS: (Mattern 1989, Fidge 1991)

So now, we have N processes in the system,

- ↳ Each process has a vector clock : array of N integers
- ↳ timestamp all events.
- ↳ piggyback on all messages.

ALGORITHM (at P_i) (they all run the same thing)

INITIALLY: $my\text{-}VT = (0, 0, 0 \cdots 0) \in \mathbb{R}^N$

On event e :

$my\text{-}VT(i) = my\text{-}VT(i) + 1$ (tick)

If e is send of message (m) :

$m.VT = my\text{-}VT$ (piggyback)

If e is receive of message (m) :

$my\text{-}VT(j) = \max(m.VT(j), my\text{-}VT(j))$

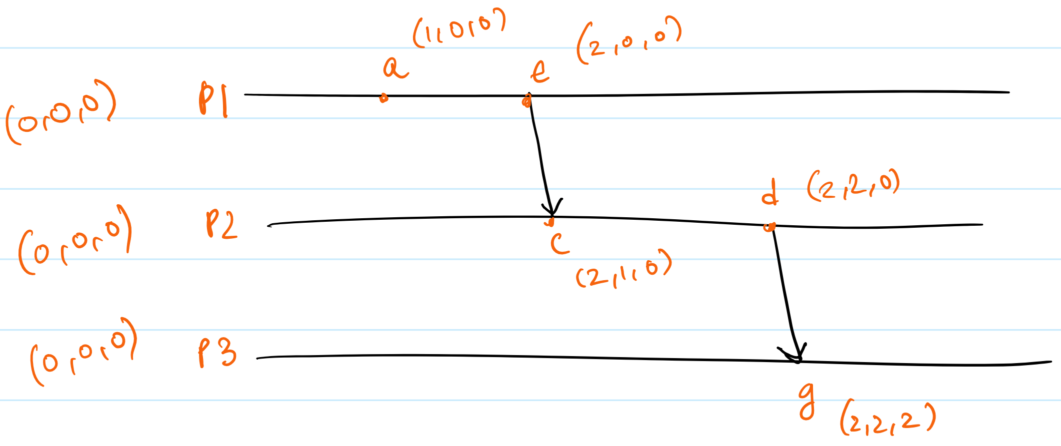
for $j = 1, 2, 3 \cdots N$

(pairwise max of elements)

for eg. $\max[(0,1,2), (2,2,0)] \equiv (2,2,2)$

for eg. $\max\lfloor(0,1,2), (2,2,0)\rfloor \equiv (2,2,2)$

* timestamp e



at p_i, my-VT(i) = # of events that p_i, have time stamped

my-VT(j) = # of events that occurred at p_j that causally precede e

$j \neq i$

How do you compare these VT now? e.VT for event e

f.VT for event f

we will say,

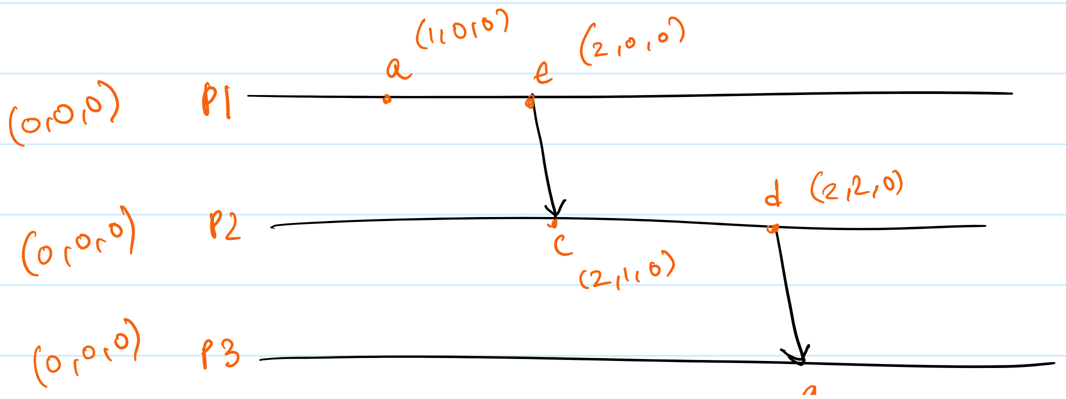
(i) $e.VT = f.VT$ iff $e.VT(i) = f.VT(i)\ \forall i \in \{1,N\}$

(ii) $e.VT \leqslant f.VT$ iff $e.VT(i) \leqslant f.VT(i)\ \forall i \in \{1,N\}$

(iii) $e.VT < f.VT$ iff $e.VT \leqslant f.VT$ & $e.VT \neq f.VT$

Partial ordering

Strong clock condition



condition

$(0,0,0)$ p3 — g $(2,2,2)$

① $VT(e) = (2,0,0)$ $\therefore VT(e) < VT(g)$

$VT(g) = (2,2,2)$ $\Rightarrow e \to g$

② $VT(f) = (0,0,1)$ $\therefore VT(e) < VT(f)$? No

$VT(e) = (2,0,0)$ $VT(f) < VT(e)$? No

Concurrent $e \| f$

$\boxed{\text{PROOF}}$ $\boxed{\text{THEOREM}}$ Vector clock satisfies the strong clock condition $e \to f$ iff $VT(e) < VT(f)$

SKETCH

(a) $e \to f \Rightarrow VT(e) < VT(f)$

(i) If they are events on the same process, e & f and $e \to f$, we P_i increments P_i's component in the VT strictly from e to f. P_i's gonna tick i^{th} component before each event in my-VT & no other component in my-VT whatsoever. [(So its own component in the $\mathbb{R}^N$ vector is strictly greater in f in compared to e)]

(ii) Send & Receive: If e is send by P_i and S is received by P_j because it added 1 to its own component & the rest is pairwise max, therefore,

P_j ticks component j of its my-VT and then does pairwise max with m-VT. So because we ticked we have atleast 1 element which is greater.

(i) and (ii) $\Rightarrow$ other cases by transitivity

(b) if $VT(e) < VT(f) \Rightarrow e \rightarrow f$

we are going to prove by contrapositive ↓

$$\neg(e \rightarrow f) \Rightarrow \neg(e.VT < f.VT)$$

CASE I : assume $f \rightarrow e$, then $f.VT < e.VT$ } weak clock condition

CASE II : assume $f \parallel e$:

CLAIM : e & f must take place on different processes

say e is on p_i and f is on p_j

when e occurs $e.VT(i) > p_j.VT(i)$

↓

j has no idea e is happening

$e.VT(i) > f.VT(i)$

↳ how many events having occurred at p_i that process f knows about

Same argument

$e.VT(j) < f.VT(j)$. Only because $e \parallel f$

P_i knows about e, but p_j doesnot

P_j knows about f, but p_i doesnot

p_j knows about f, but p_i doesnot

$\neg(e.VT < f.VT)$
and $\neg(f.VT < e.VT)$

Scalability??

Not good for
$N > 1000$
you don't want to
send a $\mathcal{X} \in \mathbb{R}^N$
with each message.