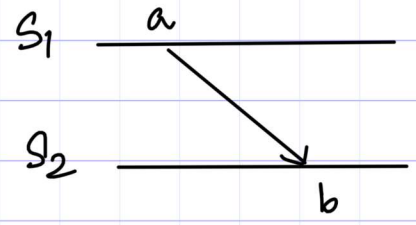
Logical Clocks (causality: as per chronology)

By Leslie Lamport

"happened before" relationship → causally affects.

- If 'a' happened before 'b', then
  $Clock_a < Clock_b$.
  $a \longrightarrow b$

- If two processes interact with 'a' in one to 'b' in another, then
  $send(m) \longrightarrow recv(m)$
  $a \longrightarrow b$.
  $C_a < C_b$

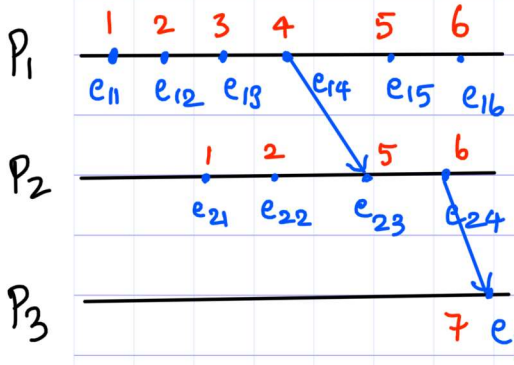


- Rules for incrementing:

$C_i = C_i + d$ for processing events ($d = 1$)

$C_{j+1} = \max(C_j, t_m) + 1$ for communication events.
↓
(message timestamp)

Partial Orders



- Transitive $e_{11} \longrightarrow e_{22}$
- Antisymmetric $e_{11} \not\longrightarrow e_{21}$
- Irreflexive $e_{11} \longrightarrow e_{11}$

Cons: if $a \rightarrow b$; $C_a \leq C_b$.

but,

(not strongly consistent) if $C_a \leq C_b$; $a \rightarrow b$? FALSE.

i.e. cannot determine chronological order given any two clock values.

Vector Clocks

- Rules for incrementing

$C_i[i] = C_i[i] + 1$ for processing events.

$\forall_k \; C_j[k] = \max(E_m[k], C_j[k] + 1)$

↓

Increment self clock on receive.

For some clock values: t_a, t_b

$t_a == t_b \Rightarrow \forall_k \; t_a[k] == t_b[k]$

$t_a != t_b \Rightarrow \exists_k \; t_a[k] != t_b[k]$.

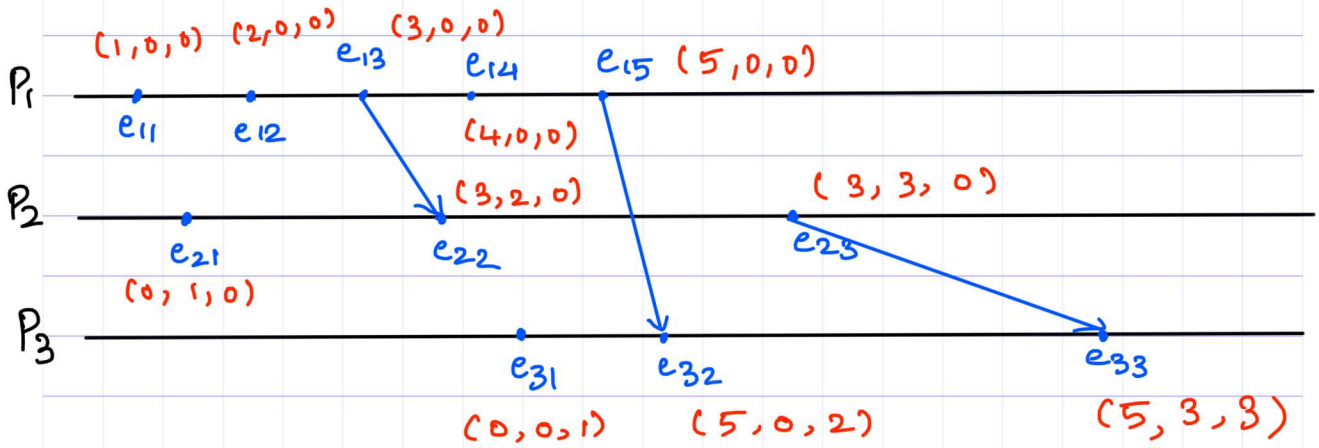
Cons:

Overlooks concurrent events.

Cannot determine which event occurred in which process, in what order.

Need more memory.

No. of processes may not be known in advance.



Applications :

- distributed debugging
- causal distributed shared memory
- global breakpoints
- consistency of checkpoints in optimistic recovery.

Vector clocks are strongly consistent.

i.e. if $t_a < t_b \Rightarrow a \rightarrow b$.

↓

solves the Lamport clock problem.