Logical Clock (Lamport Clock)

- Li initialized to 0 at process Pi
- Compute event at process Pi:
  - Increment Li by 1
  - New value of Li is the timestamp of the compute event
- Send event at process Pi: Consider e = send(m)
  - Increment Li by 1
  - New value of Li is the timestamp of send event e
  - Piggyback the timestamp of e with message m
- Receive event at process Pi: Suppose (m,t) where m is a message, and t is the piggybacked timestamp, is received at event e at Pi
  - Update Li as Li := max(Li, t)+1

Causal Ordering using vector timestamps

Vector Logical Clocks

- With Lamport Logical Timestamp
  - e \rightarrow f \Rightarrow \text{timestamp}(e) < \text{timestamp}(f)$, but \text{timestamp}(e) < \text{timestamp}(f) \Rightarrow (e \rightarrow f)$ OR (e and f concurrent)
- Vector Logical time addresses this issue:
  - Each process maintains a vector clock, length = number of processes
  - At each event, process i increments i-th element of vector $V_i$
    - The new $V_i$ is the timestamp of the event
  - A message carries the Send event's vector timestamp
  - For a receive(message) event at process $k$ ... let $V_{message}$ denote vector timestamp received with the message

$$V_i(j) = \begin{cases} \max(V_i(j), V_{message}(j)), & \text{if } j \neq k \\ V_i(j) + 1, & j = k \end{cases}$$

Comparing Vector Timestamps

- $VT_1 = VT_2$; iff $VT_1[i] = VT_2[i]$, for all $i = 1, \ldots, n$
- $VT_1 \leq VT_2$; iff $VT_1[i] \leq VT_2[i]$, for all $i = 1, \ldots, n$
- $VT_1 < VT_2$; iff $VT_1[i] < VT_2[i]$ & \exists j (1 \leq j \leq n & VT_1[j] < VT_2[j])
- $VT_1$ is concurrent with $VT_2$; iff (not $VT_1 < VT_2$ AND not $VT_2 < VT_1$)

Theoretical Base for NTP

- $T_{oA} = T_{oB} + t + \alpha$
- $T_{oB} = T_{oA} - T_{rB} + T_{rA}$

Linearizability

- An execution is linearizable if there exists a permutation that is valid,
  - per-process order-preserving, and
  - real-time order-preserving