3 LOGICAL TIME

- scalar time
- vector time
- matrix time

logical time used to capture causality of events in distributed systems, as physical time is not accurate enough.

necessity for knowledge of causality

- distributed algorithm design
  - liveness & fairness in mutual exclusion algorithms
  - maintain consistency in replicated databases
  - help design correct deadlock detection algorithms
  - avoid phantom and undetected deadlocks
- tracking of dependent events
  - resuming reexecution, in failure recovery
  - help build a checkpoint in replicated databases
  - aid in detection of file inconsistencies in case of network partitioning.
- knowledge about progress
  - discarding obsolete information
  - garbage collection
  - termination detection
- concurrency measure

logical clock $C : H \rightarrow T$ — time domain

clock consistency condition

$e_i \rightarrow e_j \Rightarrow C(e_i) < C(e_j)$

strongly consistent

$e_i \rightarrow e_j \Leftrightarrow C(e_i) < C(e_j)$

datastructures

(to represent logical time)

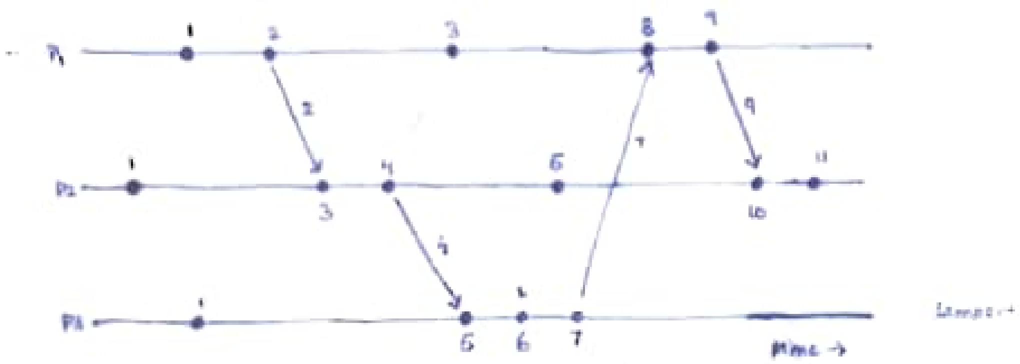
- local clock - own progress
- global clock - local view of others' progress

protocol

(update data to ensure consistency condn.)

- R1 updating local clock
- R2 updating global clock

SCALAR TIME



C : local clock & local view of global time (integer)

$\boxed{R1}$ $C_i = C_i + d$

{on internal, send}

$\boxed{R2}$ 1. $C_i = \max(C_i, C_{msg})$

2. execute R1

3. deliver msg

{on recieve}

- totally ordering events possible

timestamps : $x = (t, i)$ $y = (u, j)$

$x < y \Leftrightarrow t < u$ or $t = u$ & $i < j$

$\Rightarrow$ $x \rightarrow y$ v $x \| y$

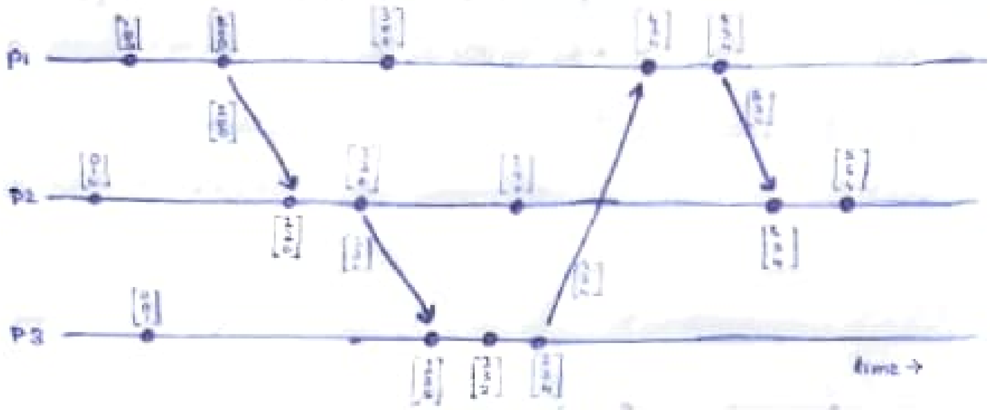
events with same scalar time are independent

tie-breaking mechanism used if same ↑.

- event counting , among processes
- no strong consistency

$e_i^1 < e_2^1 \not\Rightarrow e_i^1 \rightarrow e_2^1$

VECTOR TIME



Fidge, Mattern & Schmuck

C_i: $\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}$ local time

local view of global time

$\boxed{R1}$ $C_i[i] = C_i[i] + d$

{on internal, send}

$\boxed{R2}$ 1. $C_i = \max(C_i, C_{msg})$

2. execute R1

3. deliver msg

{on recieve}

$C_i = C_j \iff \forall n \quad C_i[n] = C_j[n]$

$C_i \leq C_j \iff \forall n \quad C_i[n] \leq C_j[n]$

$C_i < C_j \iff C_i \leq C_j$ and $\exists n \quad C_i[n] < C_j[n]$

$C_i \parallel C_j \iff \neg(C_i < C_j) \wedge \neg(C_j < C_i)$

- isomorphism between events & timestamps

  $x \to y \iff C_x < C_y$

  $x \parallel y \iff C_x \parallel C_y$

- strong consistency we are aware of missing related
- event counting

applications: distributed debugging, causal communication, causal shared memory, establish global breakpoints, consistency of checkpoints.

EFFICIENT IMPLEMENTATIONS OF VECTOR CLOCKS

SINGHAL KSHEMKALYANI'S DIFFERENTIAL TECHNIQUE

observation: between successive send to ~~same pr~~ process, only few entries of vector clock at sender process change (likely).

LS: last sent

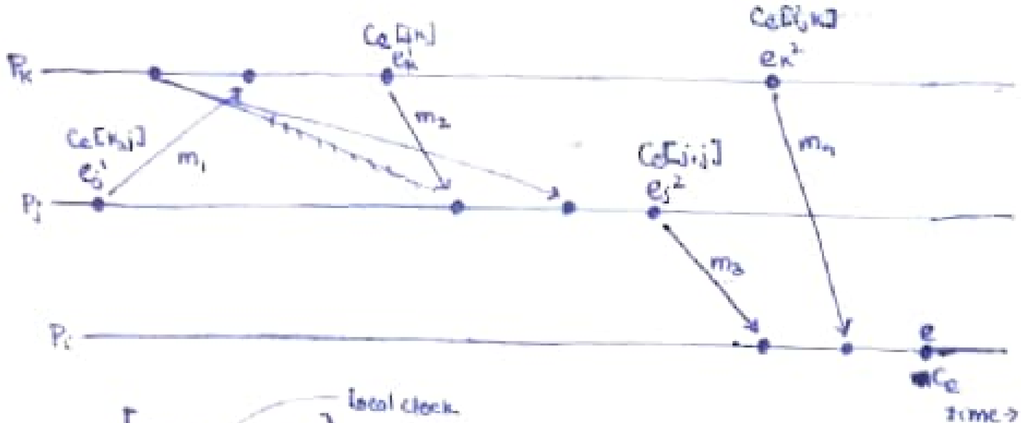
value of local clock when P_i last sent message to P_j

LU: last update

value of localclock when P_i updated respective entry of its clock.

$\{(P_x, c_x)\}$ data can be sent in form of tuples instead of full vector.

MATRIX TIME



$$C_2 : \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{j1} & c_{ii} & & c_{in} \\ c_{n1} & c_{n2} & & c_{nn} \end{bmatrix} \begin{matrix} \} \text{ - vector clock of } c_i \\ \} \text{ - vector clock of } c_n \end{matrix}$$

local clock

$\boxed{R1}$ $C_i[i,i] = C_i[i,i] + d$

{on internal, send}

$\boxed{R2}$ 1. $C_i[i,*] = \max(C_i[i,*], C_{msg}[j,*])$

$C_i = \max(C_i, C_{msg})$

2. execute R1

3. deliver msg

- discard obsolete information $\min(C_i[*,l]) \geq t \Rightarrow$ every process knows that P_l time has progressed till t.

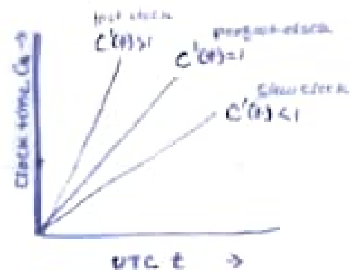
process will no longer require from P_l certain information

can use this fact to discard obsolete information

PHYSICAL CLOCK SYNCHRONIZATION: NTP

process of ensuring that physically distributed processes have a common notion of time, interval, and relative ordering of events. due to different clock rates, periodic clock synchronization is necessary. (with UTC)

- time : $C_a(t)$
  time of clock on machine
- offset : $C_a(t) - t$
  time diff. wrt real time
- drift : $C_a''(t)$
  acceleration of clock progress
- frequency : $C_a'(t)$
  rate at which clock progresses
- skew : $C_a'(t) - 1$
  rate diff. wrt perfect clock



$a = T_1 - T_3$ $b = T_2 - T_4$

clock offset θ (wrt B)

$= \frac{a+b}{2} = \frac{T_1 + T_2 - (T_3 + T_4)}{2}$

roundtrip delay δ

$= a - b$

$= (T_4 - T_3) - (T_2 - T_1)$

Network time protocol (NTP) on internet uses the offset delay estimation method. NTP involves a hierarchical tree of time servers. Here, pairs A, B can independently calculate delay & offset using a single bidirectional message stream.

8 most recent pairs of θ, δs taken. the pair with min. δ is chosen.