CSE535 Asynchronous Systems
Logical Time

YoungMin Kwon

Clock

- The elements of a time domain T form a partial order relation <.
- A clock C: H $\rightarrow$ T is a function that maps an event to a time domain element in T such that the clock consistency holds

$$e_i \rightarrow e_j \Longrightarrow \mathrm{C}(e_i) < \mathrm{C}(e_j)$$

- Clocks are called strongly consistent if

$$e_i \rightarrow e_j \Leftrightarrow \mathrm{C}(e_i) < \mathrm{C}(e_j)$$

Scalar Time

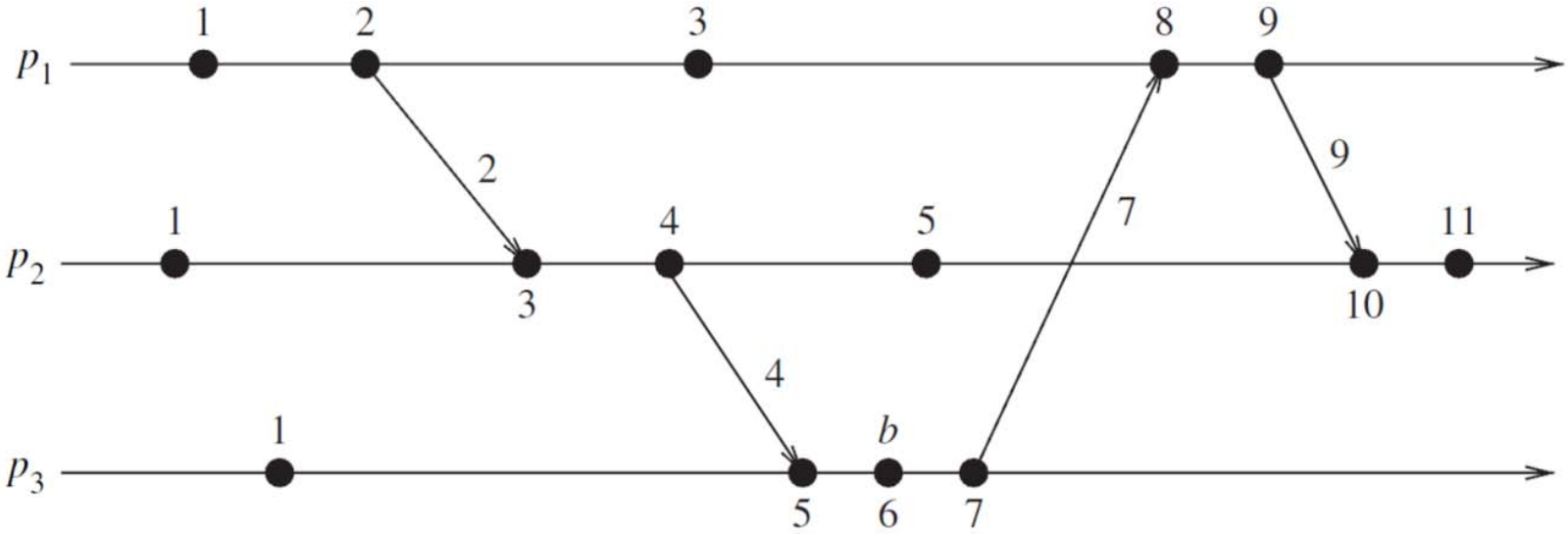
- Time domain:
  - The set of non-negative integers

- The logical clock of a process p_i and its local view of the global time are an integer C_i

Scalar Time

- Clock update rules:
  - R1: before sending a message or executing an internal event
    - $C_i := C_i + d$, where d is typically 1
  - R2: each message piggybacks the clock of its sender. When a process p_i receives a message with C_{msg}:
    - $C_i = \max(C_i, C_{msg})$
    - Execute R1
    - Deliver the message

Scalar Time



Scalar Time

- Consistency

$$e_i \rightarrow e_j \Longrightarrow \mathrm{C}(e_i) < \mathrm{C}(e_j)$$

- Can totally order events with a tie breaking rule

$$x \prec y \Leftrightarrow (h < k \textbf{ or } (h = k \textbf{ and } i < j))$$

- No strong consistency

$$\mathrm{C}(e_i) < \mathrm{C}(e_j) \not\Longrightarrow e_i \rightarrow e_j$$

Vector Time

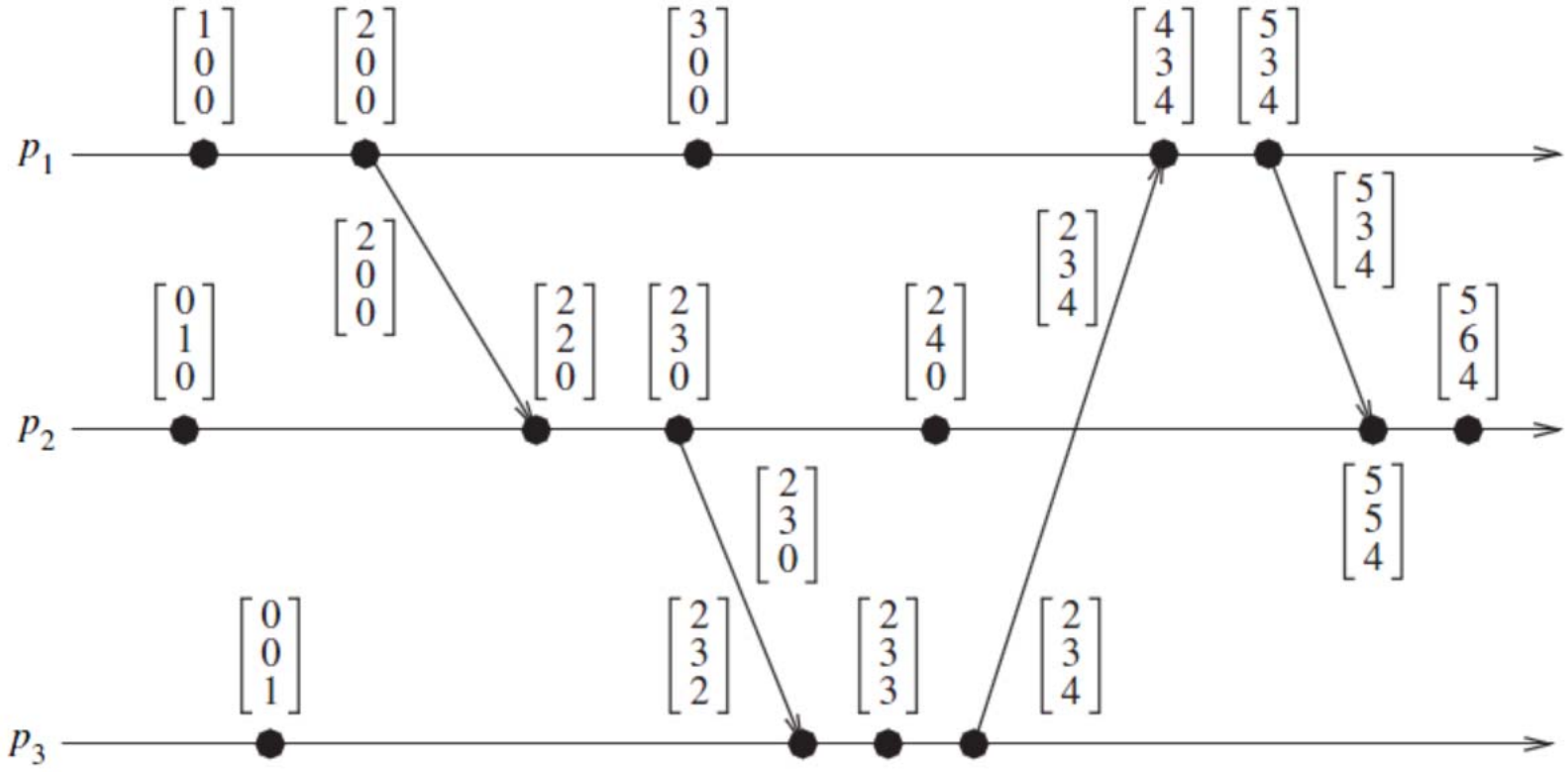
- Time domain:
  - n-dimensional non-negative integer vectors

- Each process p_i maintains a vector $vt_i[1..n]$
  - $vt_i[i]$: the local time of p_i
  - $vt_i[j]$: p_i's knowledge about p_j's local time

Vector Time

- Clock update rules:
  - R1: before sending a message or executing an internal event
    - $vt_i[i] := vt_i[i] + d$
  - R2: each message piggybacks the sender's vector clock vt. On receiving a message (m,vt):
    - $1 \leq k \leq n$: $vt_i[k] := \max(vt_i[k], vt[k])$
    - Execute R1
    - Deliver the message

Vector Time


p_1
p_2
p_3
[1 0 0]
[2 0 0]
[3 0 0]
[4 3 4]
[5 3 4]
[2 0 0]
[5 3 4]
[0 1 0]
[2 2 0]
[2 3 0]
[2 4 0]
[2 3 4]
[5 6 4]
[2 3 0]
[5 5 4]
[0 0 1]
[2 3 2]
[2 3 3]
[2 3 4]

Vector Time

- Comparison between two vector times

$$vh = vk \Leftrightarrow \forall x : vh[x] = vk[x]$$

$$vh \leq vk \Leftrightarrow \forall x : vh[x] \leq vk[x]$$

$$vh < vk \Leftrightarrow vh \leq vk \text{ and } \exists x : vh[x] < vk[x]$$

$$vh \parallel vk \Leftrightarrow \neg(vh < vk) \wedge \neg(vk < vh).$$

Vector Time: Basic Properties

- Isomorphism between structures (P, <) and (Q, $\prec$) is a one-to-one function h: P $\rightarrow$ Q such that
  - For all a, b $\in$ P, a < b iff h(a) $\prec$ h(b)

- C is an isomorphism

$$x \rightarrow y \Leftrightarrow vh < vk$$

$$x \parallel y \Leftrightarrow vh \parallel vk.$$

- If x and y occurred at p_i and p_j respectively

$$x \rightarrow y \Leftrightarrow vh[i] \leq vk[i]$$

$$x \parallel y \Leftrightarrow vh[i] > vk[i] \wedge vh[j] < vk[j]$$

Vector Time: Basic Properties

- Strong Consistency
  - Vector clocks are strongly consistent
  - Causality between two events can be determined by checking their vector times.
  - Applications: distributed debugging, checkpointing …

- Event counting
  - $Vt_i[i]$ denotes the number events occurred at p_i when d is 1.

Singhal-Kshemkalyani's Differential Technique

- Vector clocks can take huge amount when the number of processes are large.
  - Between successive sends to the same process, only few entries are likely to change.

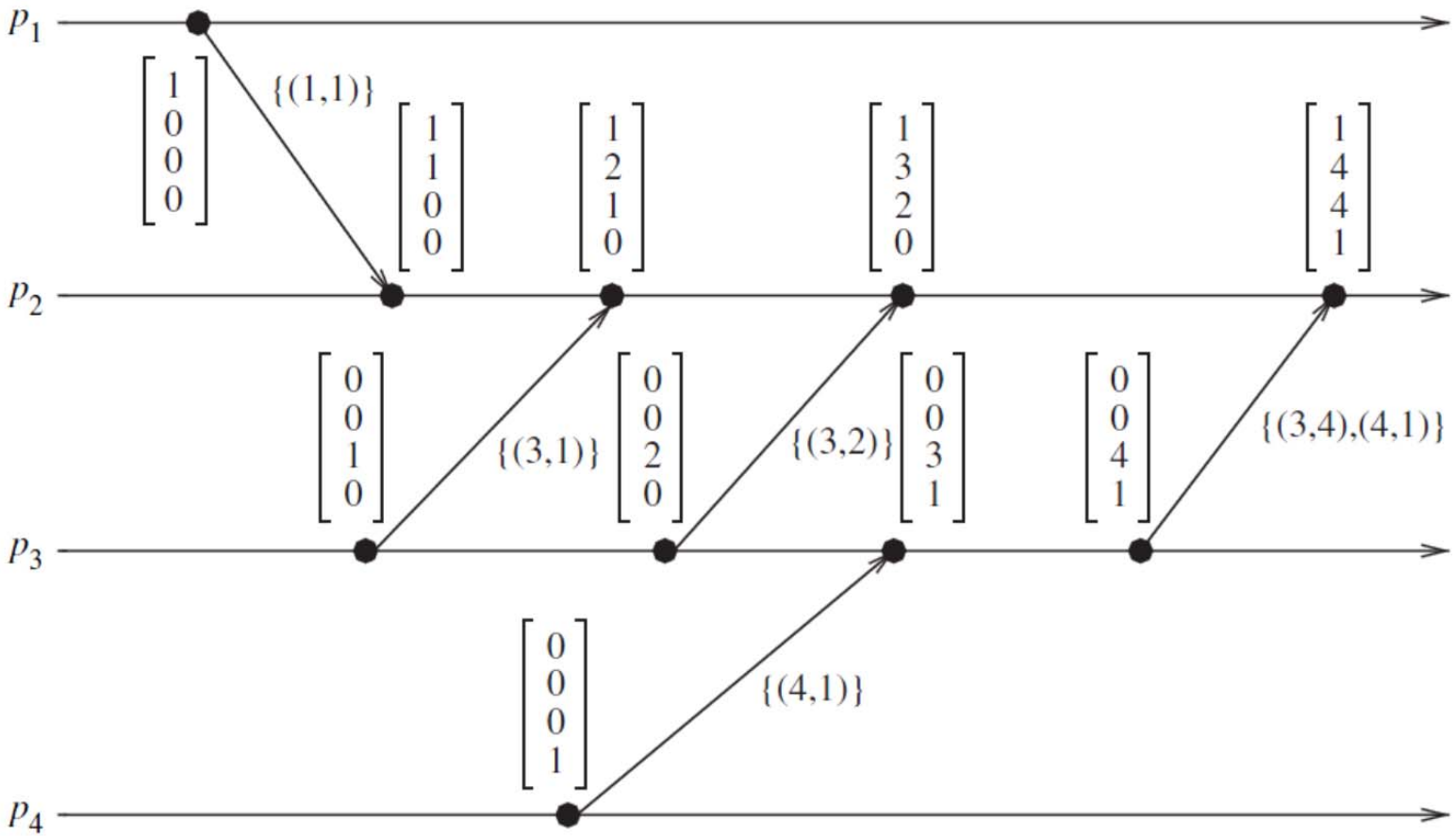
- Clock information to piggyback are a set of index and time pairs.

$$\{(i_1, v_1), (i_2, v_2), \ldots, (i_{n_1}, v_{n_1})\}$$

  - Update rule:

$$vt_i[i_k] = max(vt_i[i_k], v_k) \text{ for } k = 1, 2, \ldots, n_1$$

Singhal-Kshemkalyani's Differential Technique



Singhal-Kshemkalyani's Differential Technique

- p_i maintains
  - $LS_i[1..n]$ (Last Sent): $LS_i[j]$ is $vt_i[i]$ when p_i last sent a message to p_j
  - $LU_i[1..n]$ (Last Update): $LU_i[j]$ is $vt_i[i]$ when p_i last update $vt_i[j]$

- The clock information to piggyback when p_i sends a message to p_j

$$\{(x, vt_i[x]) | LS_i[j] < LU_i[x]\}$$

Fowler-Zwaenepoel's direct dependency technique

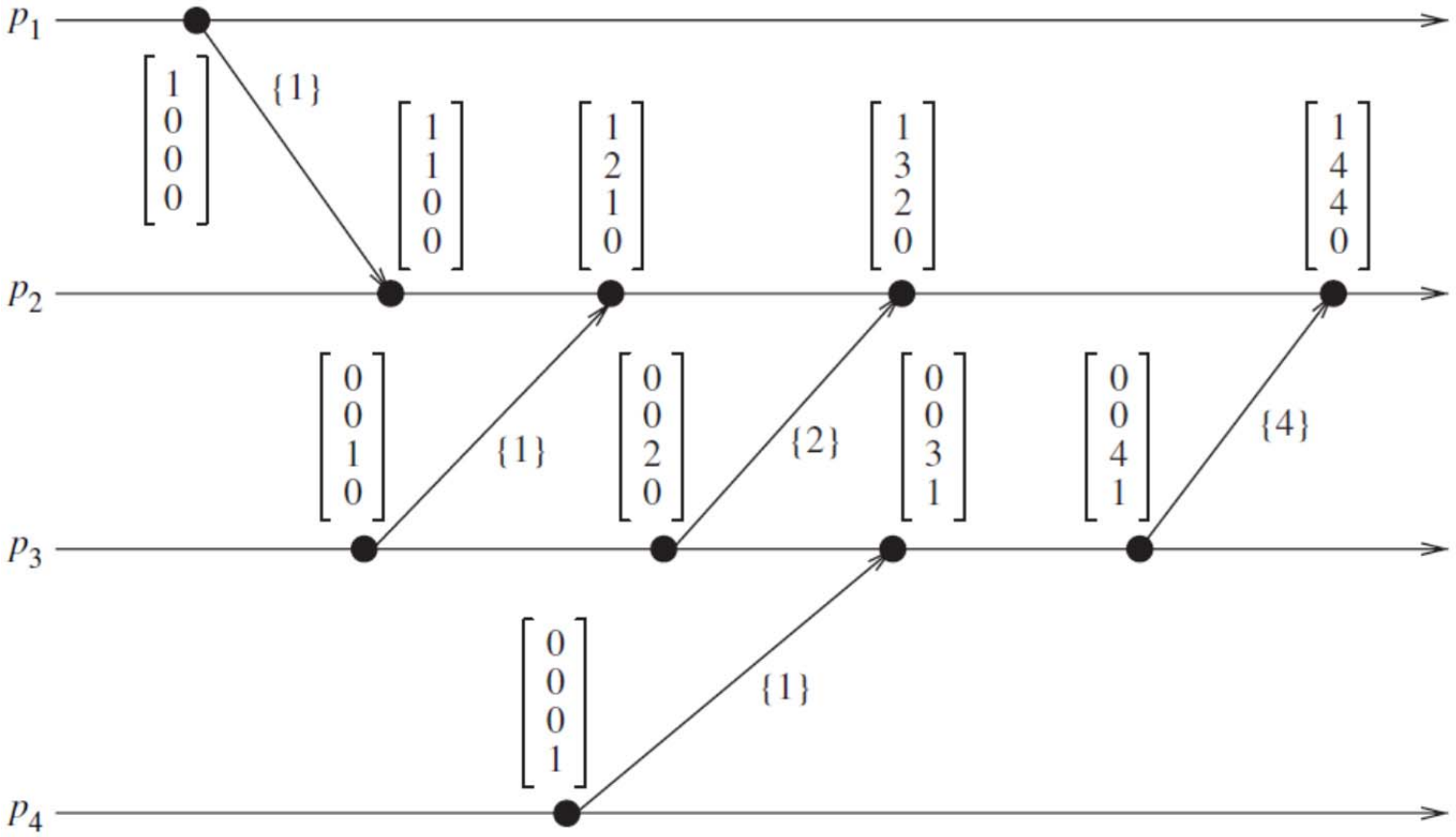
- Piggybacks only a scalar time in the message.
  - Vector clocks are not maintained on-the-fly
  - Transitive dependencies are not maintained
  - Only direct dependencies are maintained

- A vector time can be restored off-line

Fowler-Zwaenepoel's direct dependency technique

- Each process p_i maintains a dependency vector D_i
  - Initially, $D_i[j] := 0$ for $j=1,\ldots,n$
  - On each event at p_i, $D_i[i] := D_i[i] + 1$
  - When p_i sends a message to p_j, it piggybacks $D_i[i]$ in the message
  - When p_i receives a message from p_j with piggybacked value d, $D_i[j] := \max(D_i[j], d)$

Fowler-Zwaenepoel's direct dependency technique



Fowler-Zwaenepoel's direct dependency technique

- To get the vector time of p_i's e^{th} event
  - Call DependencyTrack(i, e) and then call VisitEvent(i, e)

```
DependencyTrack(i: process, e: event index) {
    for all k ≠ i {
        DTV[k] = 0
    }
    DTV[i] = e
}

VisitEvent(i: process, e: event index) {
    for all k ≠ i {
        a = D^e_i[k]
        if a > DTV[k] {
            DTV[k] = a
            VisitEvent(k, a)
        }
     }
}
```

Jard-Jourdan's adaptive technique

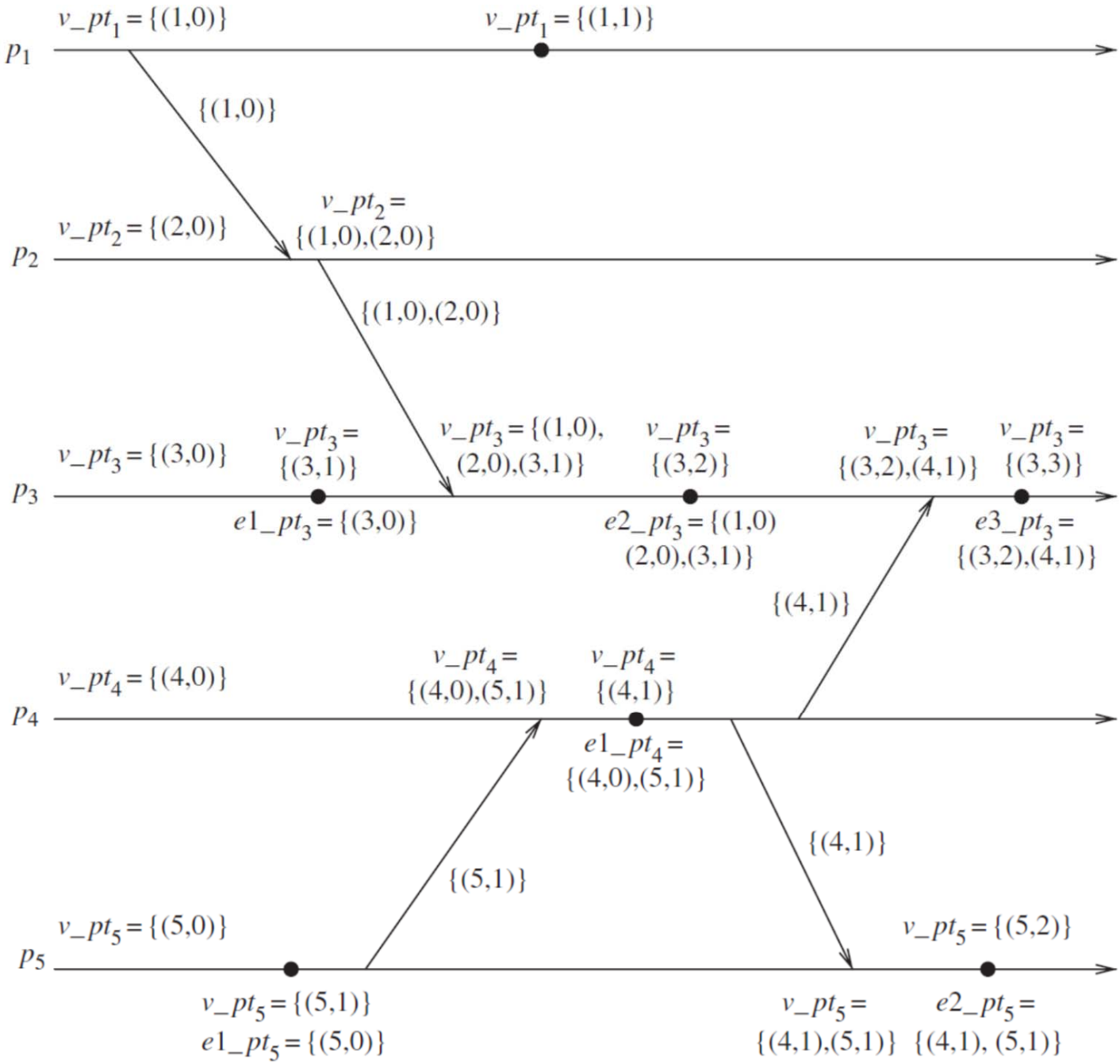
- Observing an event:
  - Recording the information about its dependencies
- Fowler-Zwaenepoel's direct dependent technique
  - Must observe an event after receiving a message but before sending any messages.
  - Require the history of dependency vectors for all events.

Jard-Jourdan's adaptive technique

- Jard-Jourdan's adaptive technique
  - Use a partial vector clock in between two observations
    - p_vt_i: partial vector time of process p_i
  - When an event is observed, the events that follow can determine their transitive dependencies.
    - e_vt_i: variable that holds the timestamp of the observed event

Jard-Jourdan's adaptive technique

- Clock Update Rules
  - When p_i observes an event,
    - p_vt_i is copied to e_vt_i: $e\_vt_i = \{(i_1,v_1),\ldots,(i,v),\ldots,(i_n,v_n)\}$
    - p_vt_i is reset: $p\_vt_i = \{(i,v+1)\}$
  - When p_i sends a message to p_j it piggybacks p_vt_i
  - When p_i receives a message with p_vt
    - Let $p\_vt = \{(i_{m1}, v_{m1}),\ldots,(i_{mk}, v_{mk})\}$, $p\_vt_i = \{(i_1,v_1),\ldots,(i_l, v_l)\}$
    - $p\_vt_i := U\ (i_x, \max(v_x,v_{mx}))$ for all $(i_x,.)$ in p_vt or in p_vt_i


$v\_pt_1 = \{(1,0)\}$
$v\_pt_1 = \{(1,1)\}$
p_1
$\{(1,0)\}$
$v\_pt_2 = \{(2,0)\}$
$v\_pt_2 = \{(1,0),(2,0)\}$
p_2
$\{(1,0),(2,0)\}$
$v\_pt_3 = \{(3,0)\}$
$v\_pt_3 = \{(3,1)\}$
$v\_pt_3 = \{(1,0), (2,0),(3,1)\}$
$v\_pt_3 = \{(3,2)\}$
$v\_pt_3 = \{(3,2),(4,1)\}$
$v\_pt_3 = \{(3,3)\}$
p_3
$e1\_pt_3 = \{(3,0)\}$
$e2\_pt_3 = \{(1,0)$ $(2,0),(3,1)\}$
$e3\_pt_3 = \{(3,2),(4,1)\}$
$\{(4,1)\}$
$v\_pt_4 = \{(4,0)\}$
$v\_pt_4 = \{(4,0),(5,1)\}$
$v\_pt_4 = \{(4,1)\}$
p_4
$e1\_pt_4 = \{(4,0),(5,1)\}$
$\{(4,1)\}$
$\{(5,1)\}$
$v\_pt_5 = \{(5,0)\}$
$v\_pt_5 = \{(5,2)\}$
p_5
$v\_pt_5 = \{(5,1)\}$ $e1\_pt_5 = \{(5,0)\}$
$v\_pt_5 = \{(4,1),(5,1)\}$
$e2\_pt_5 = \{(4,1), (5,1)\}$

Matrix Time

- Each process p_i maintains a matrix $mt_i[1..n, 1..n]$
  - $mt_i[i,i]$ is the local logical clock of p_i
  - $mt_i[i,*]$ is the vector clock: p_i's knowledge about p_j's logical clock $mt_j[j,j]$ for all j in [1,n]
  - $mt_i[j,k]$ is p_i's knowledge about p_j's logical clock for p_k $mt_k[k,k]$.

Matrix Time

- Clock Update Rule
  - R1: Before executing a send or internal event,
    $mt_i[i,i] := mt[i,i] + d$
  - R2: Piggybacks mt to each message. When p_i receives (m,mt) from p_j
    - $mt_i[i,k] := \max(mt_i[i,k], mt[j,k])$ for $1 \le k \le n$
    - $mt_i[k,l] := \max(mt_i[k,l], mt[k,l])$ for $1 \le k, l \le n$
    - Execute R1
    - Deliver message m

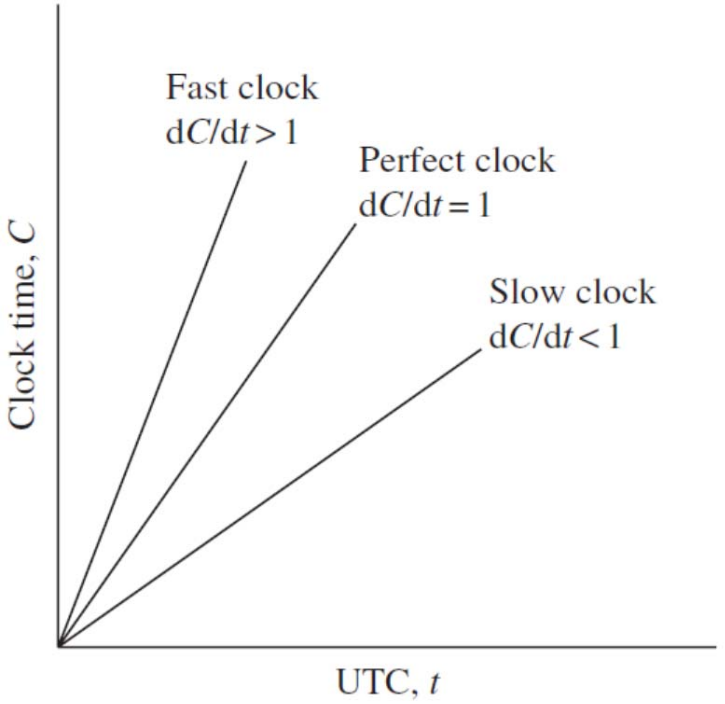
Matrix Time

- Basic properties
  - $\min_k (mt_i[k,l]) \geq t$: p_i knows that every other process knows that p_l's local time has progressed till t
  - A process can discard obsolete information

Physical Clock Synchronization

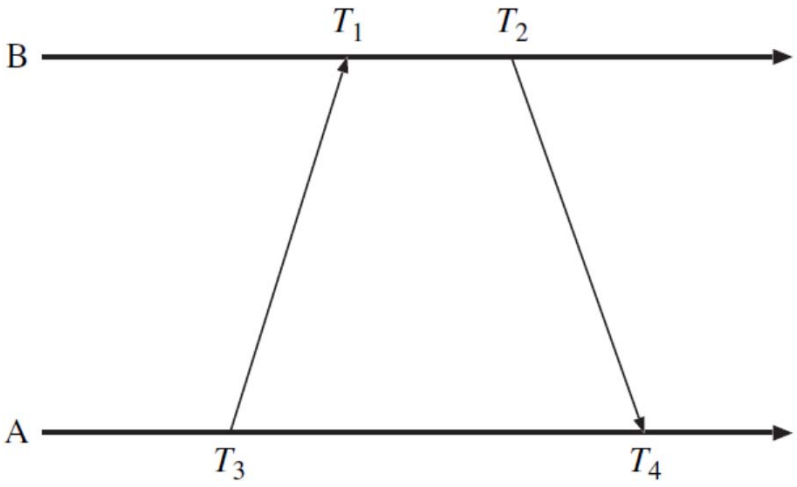
- NTP (Network Time Protocol)
  - Estimates clock offset and network delay

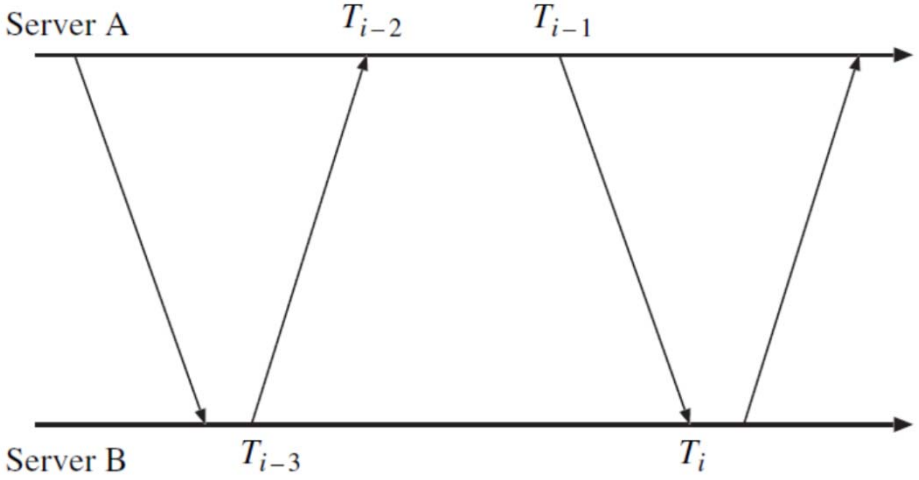
- Clocks in reality



NTP

- NTP estimates clock offset θ and roundtrip delay δ
  - Assuming the delay difference from A to B and B to A is small.
  - $T_1 = T_3 + \delta/2 + \theta$
  - $T_2 = T_4 - \delta/2 + \theta$
  - $\theta = (T_1 + T_2 - T_3 - T_4) / 2$
  - $\delta = (T_1 - T_2 + T_4 - T_3)$





Convergence Lab (C408)

- How to login
- Quick Connection Menu
  - Select **"Remote Desktop"**
- Remote Desktop Connection
  - **Session Name <Quick Connection>**
  - **Computer: IP of your VM**
  - **User Name: cse535**
  - **Password: cse535**
  - **Domain: empty**

- Change your password
  - Open a terminal
  - **Type passwd**
  - **Type your old password (cse535)**
    Type your new password
    Retype your new password

No.	Name	VM
1	Jung, Kyeong Joo	192.168.40.74
2	Kim, Ho Young	192.168.40.98
3	Lee, Sangyup	192.168.40.101
4	Tariq, Shahroz	192.168.40.27
5		
6		
7		
8		
9		
10		

SUNY Korea

Practice

- Download VectorTime.java from
  - http://www3.cs.stonybrook.edu/~youngkwon/cse535/VectorTime.java

- Implement Singhal-Kshemkalyani's Differential Technique using the VectorTime.java