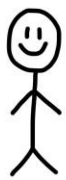
A Model for Distributed Computing

Venkatesh Vinayakarao
venkateshv@cmi.ac.in
http://vvtesh.co.in

Chennai Mathematical Institute

Data is the new oil. - Clive Humby, 2006.

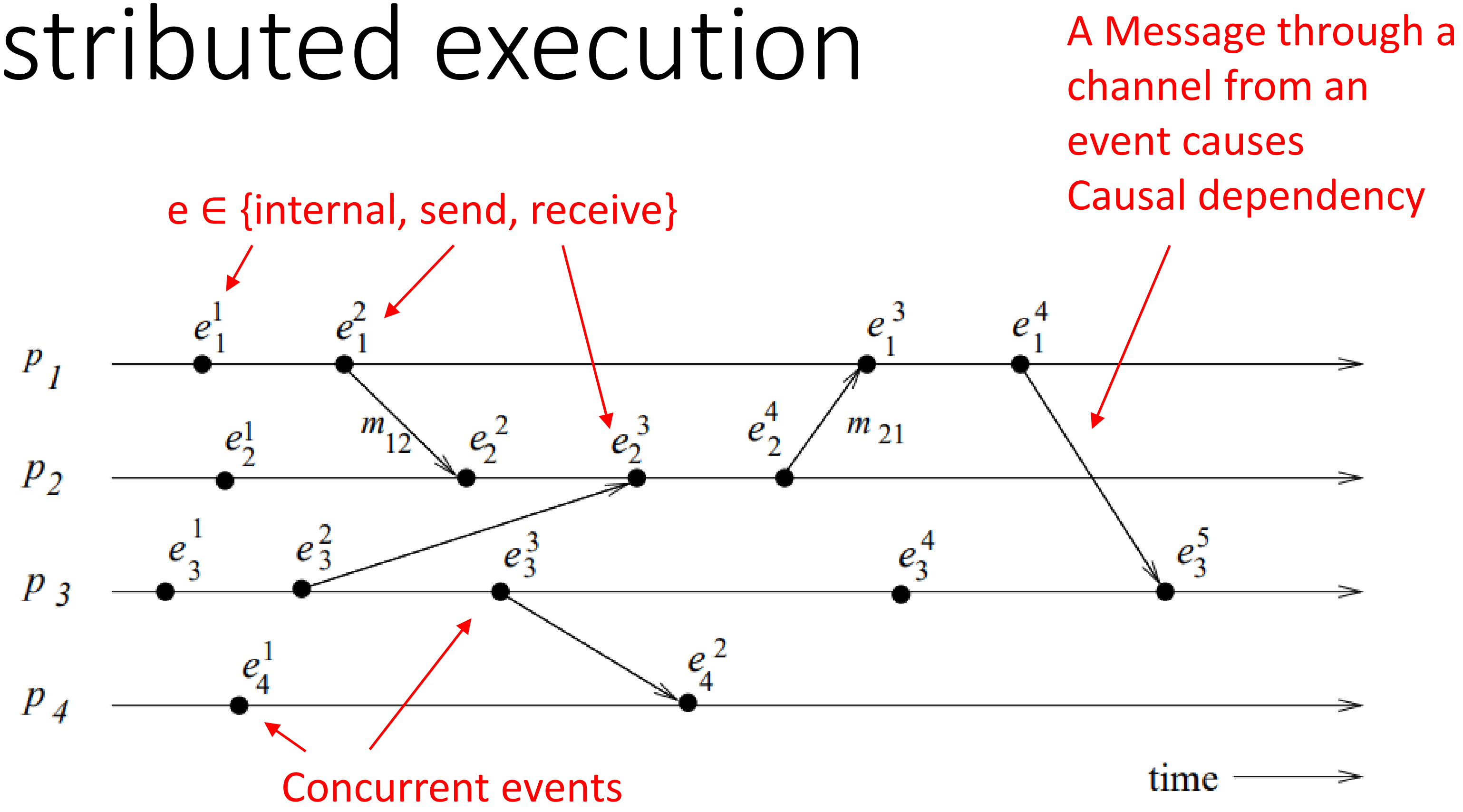
A Distributed Computing Model



No shared memory.
No notion of global clock.

How to co-ordinate to get a job done?

Space-time diagram of a distributed execution



A Distributed Execution

For a more complete and formal handling, see https://www.cs.uic.edu/~ajayk/Chapter2.pdf

Causal dependencies between events

$$\forall e_i^x,\ \forall e_j^y \in H,\quad e_i^x \ \rightarrow\ e_j^y \quad \Leftrightarrow \quad \begin{cases} e_i^x \rightarrow_i e_j^y \quad i.e., (i = j) \wedge (x < y) \\ or \\ e_i^x \rightarrow_{msg} e_j^y \\ or \\ \exists e_k^z \in H : e_i^x \rightarrow e_k^z \ \wedge\ e_k^z \rightarrow e_j^y \end{cases}$$

$H = \cup_i h_i$ denotes set of all events of process p_i

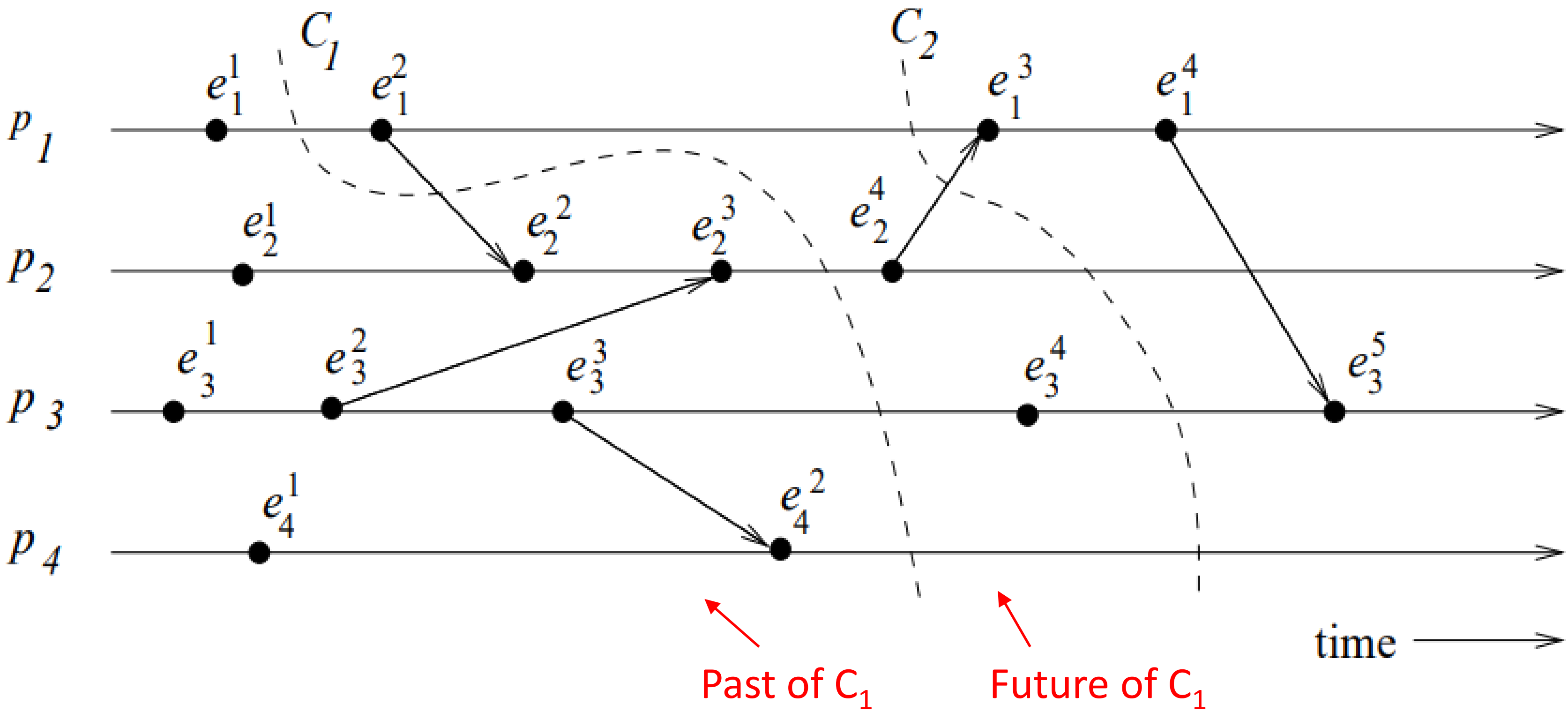
$e_1^1 \rightarrow e_3^3$ and $e_3^3 \rightarrow e_2^6$ denotes a causal dependency from the figure.

$\mathcal{H} = (H, \rightarrow)$ denotes causal precedence relation

Cuts of an execution

$$Past(e_j) = \{e_i | \forall e_i \in H,\ e_i \rightarrow e_j\}$$

$$Future(e_j) = \{e_i | \forall e_i \in H,\ e_j \rightarrow e_i\}$$



Note that C_1 is **inconsistent**. C_2 is **consistent**. Can you see why?

Local and Global States

$$GS = \{\bigcup_i LS_i^{x_i}, \bigcup_{j,k} SC_{jk}^{y_j, z_k}\}$$

Local State Channel State

$$SC_{ij}^{x,y} = \{m_{ij} \mid send(m_{ij}) \leq e_i^x \bigwedge rec(m_{ij}) \not\leq e_j^y\}$$

Thus, channel state $SC_{ij}^{x,y}$ denotes all messages that p_i sent upto event e_i^x and which process p_j had not received until event e_j^y.

Synchrony

- Synchronous communication model
  - On message *send*, the sender process blocks until the message has been *received.*
- Asynchronous
  - Non-blocking type.

How to track causality without the notion of global time?

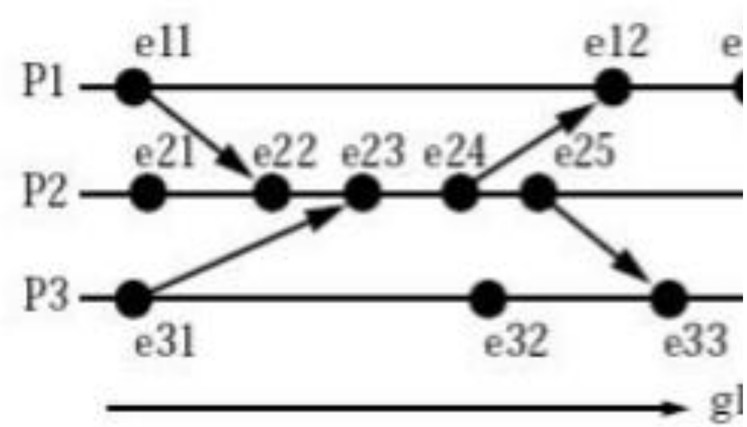
Let us define a local logical clock

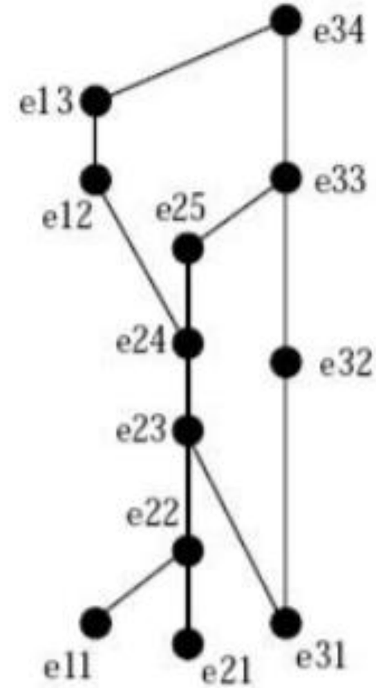
$$C : H \mapsto T$$

T is a timestamp

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

Processes with Local Clocks

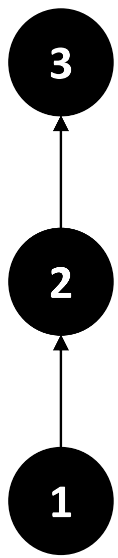




Virtual Time and Global States of Distributed Systems, Friedemann Mattern.

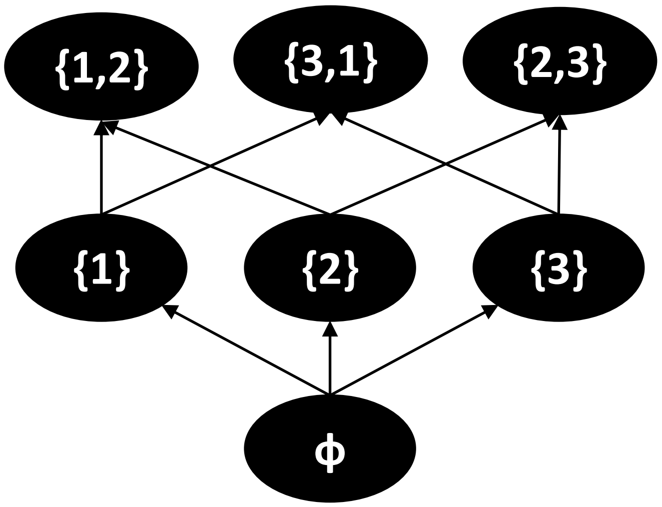
Total Vs. Partial Order

The Pair ({1,2,3}, <)



A strict total order.

The Pair ({{},{1},{2},{3},{1,2},{1,3},{2,3}}, ⊆)



Partially ordered under
the ⊆ operation!

Reflexive, Transitive and Anti-symmetric

a<=a

a<=b and b<=c implies a <= c

a<=b and b<=a implies a = b

Hasse Diagram

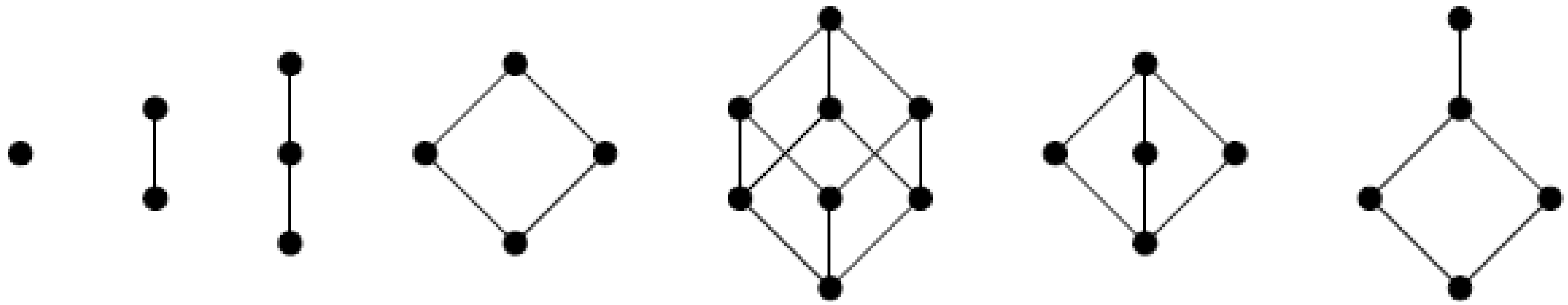
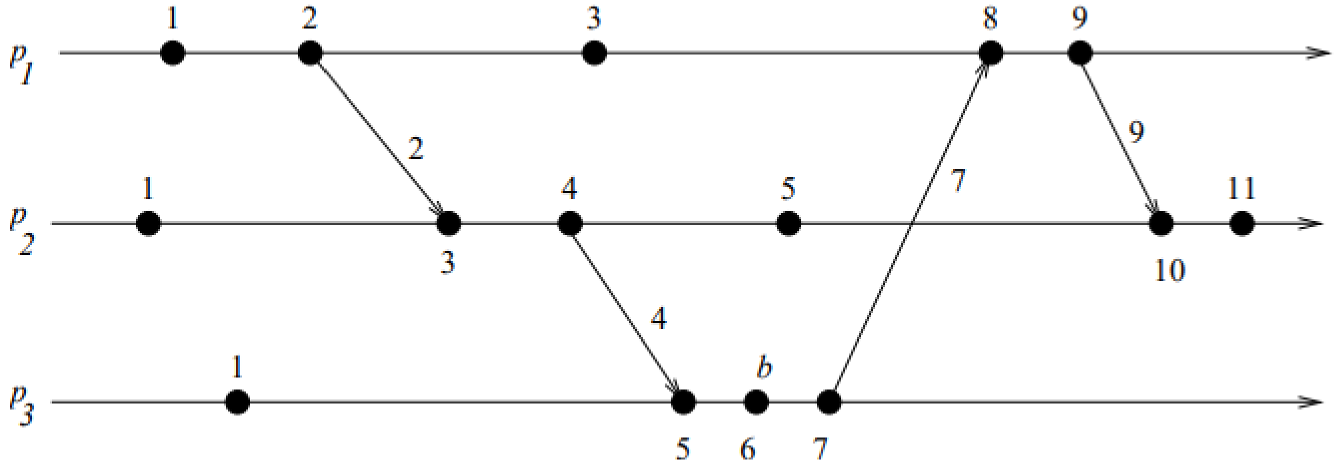
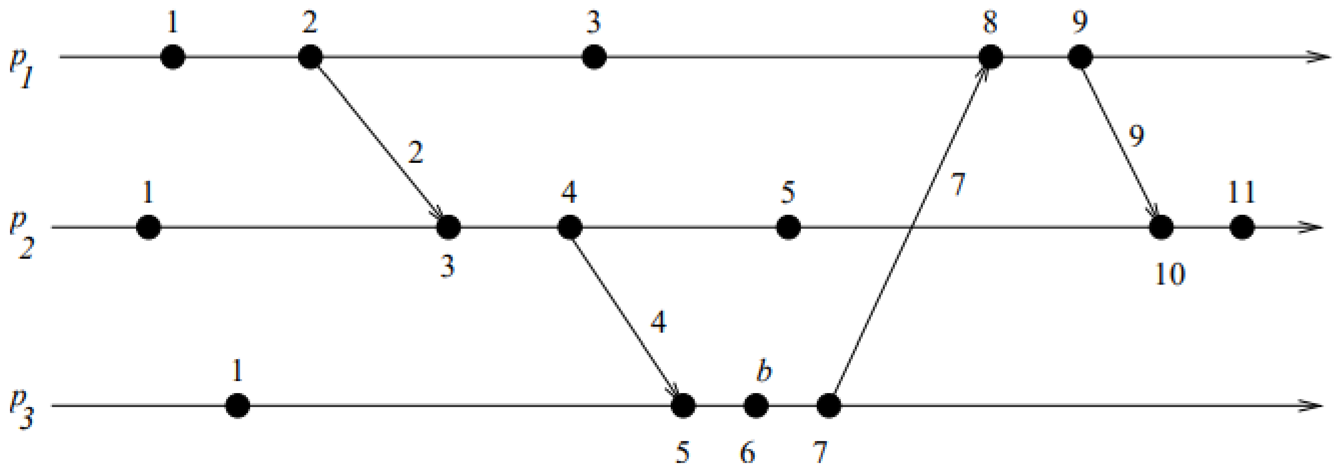


Image source: Static Program Analysis, Moller and Schwartzbach

Scalar Time



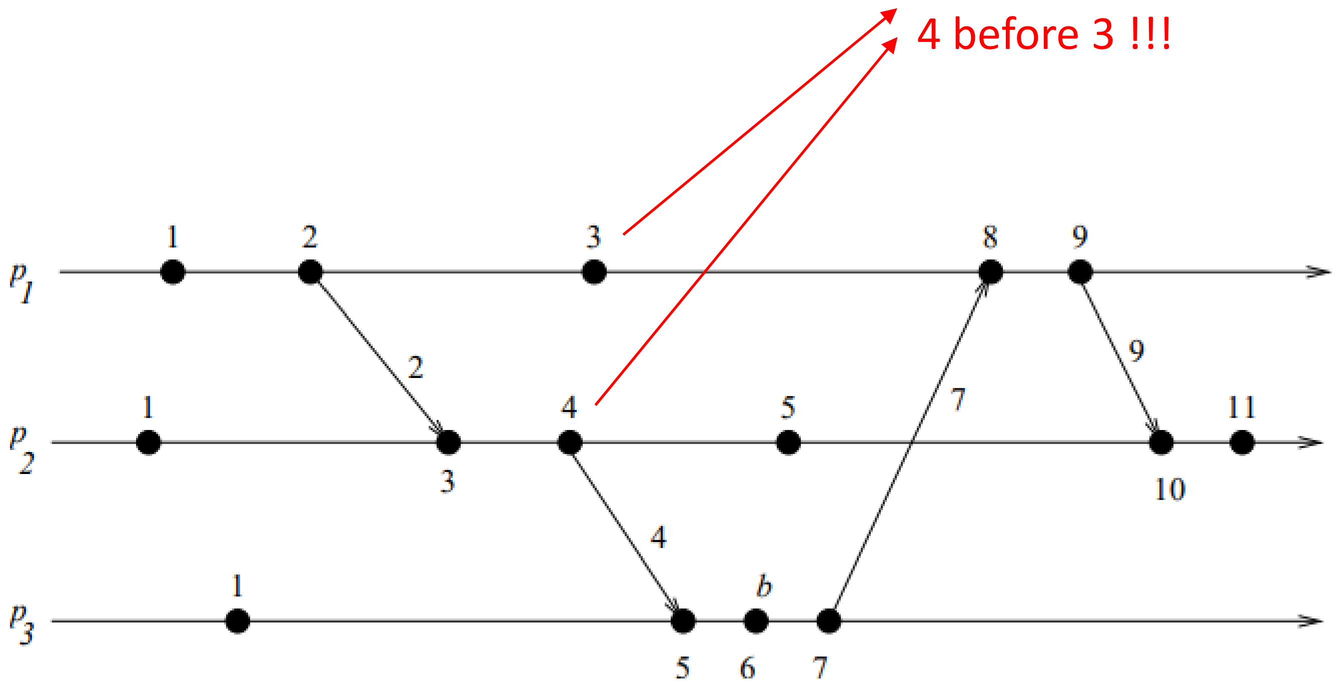
Scalar Time



$$C(e_i) < C(e_j) \not\Longrightarrow e_i \rightarrow e_j$$

Not strongly consistent

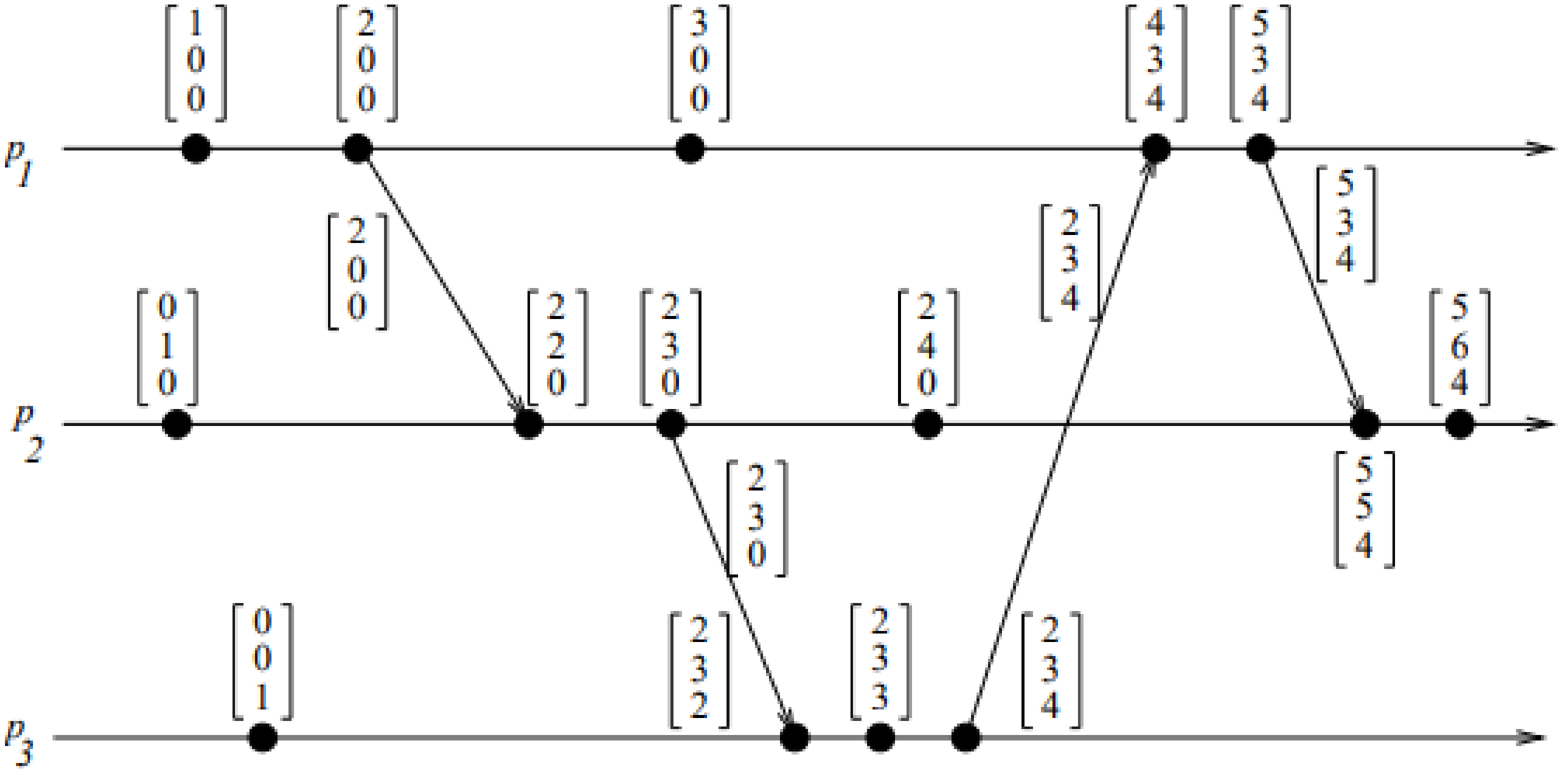
Scalar Time



$$C(e_i) < C(e_j) \not\Longrightarrow e_i \rightarrow e_j$$

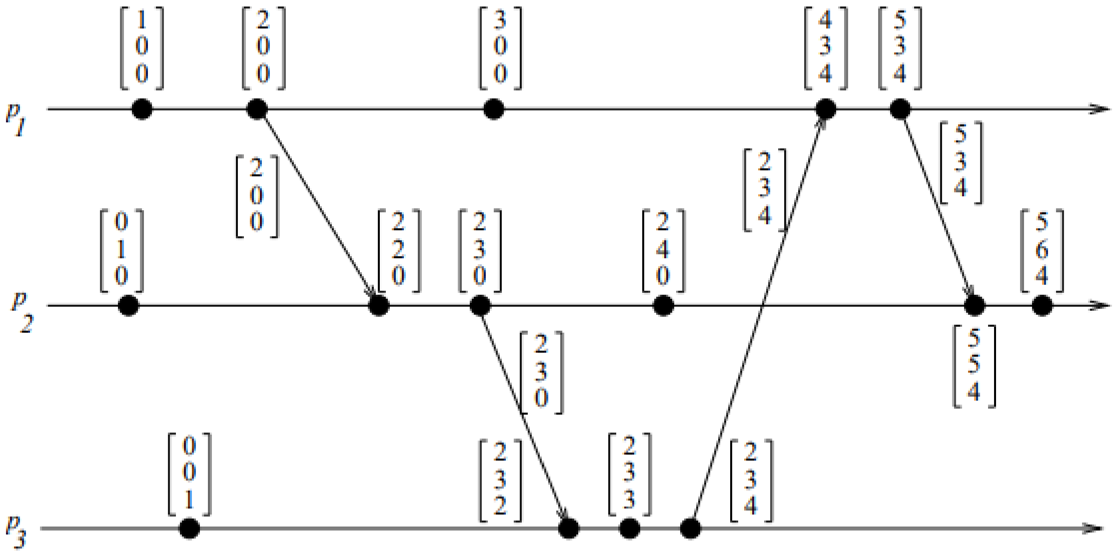
Not strongly consistent

Vector Time



p1
p2
p3
[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]
[5 5 4]
[2 3 0]
[0 0 1]
[2 3 2]
[2 3 3]
[2 3 4]

Vector Time

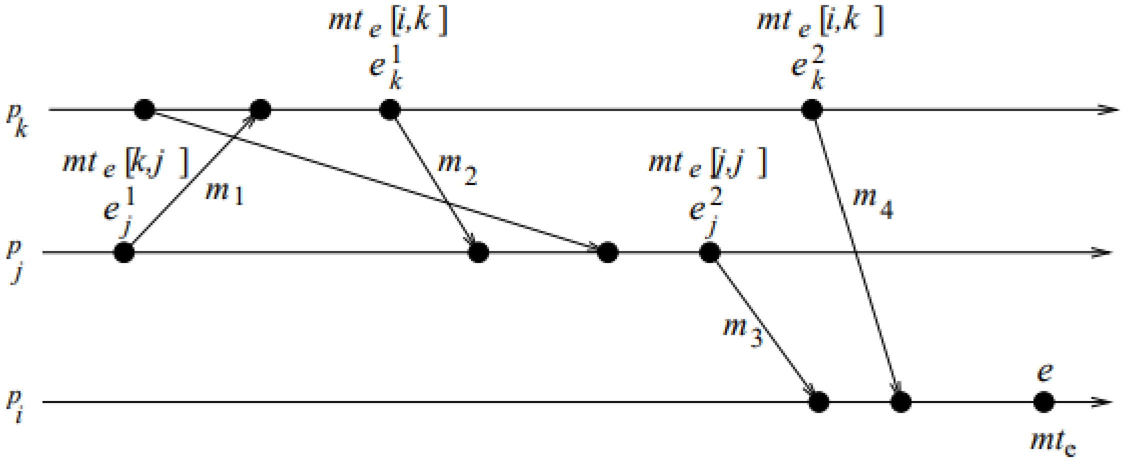


Implementing Vector Time? Notice that “*between successive message sends to the same process, only a few entries of the vector clock at the sender process are likely to change*”.

Singhal-Kshemkalyani’s differential technique uses this observation.

Matrix Time

The entire matrix mt_i denotes p_i's local view of the global logical time.



In addition, matrix clocks have the following property:
$\min_k(mt_i[k,l]) \geq t \Rightarrow$ process p_i knows that every other process p_k knows that p_l's local time has progressed till t.

A Distributed Computing Algorithm

- Consensus Problem
  - All processes have an initial value
  - All non-faulty processes must agree on the same (single) value.

A Distributed Computing Algorithm

- Consensus Problem
  - All processes have an initial value.
  - All non-faulty processes must agree on the same (single) value.
  - Setting: Message-Passing, Synchronous.

```
(global constants)
integer: f;                          // maximum number of crash failures tolerated
(local variables)
integer: x ⟵ local value;

(1) Process P_i (1 ≤ i ≤ n) executes the Consensus algorithm for up to f crash failures:
(1a) for round from 1 to f + 1 do
(1b)     if the current value of x has not been broadcast then
(1c)         broadcast(x);
(1d)     y_j ⟵ value (if any) received from process j in this round;
(1e)     x ⟵ min(x, y_j);
(1f) output x as the consensus value.
```