Need of Synchronization in DS

- It is important that multiple processes do not simultaneously access a shared resource, such as printer, but instead cooperate in granting each other temporary exclusive access.
- Multiple processes may sometimes need to agree on the ordering of events, such as whether message *ml* from process *P* was sent before or after message *m2* from process Q.

In distributed systems, there is no single global clock.

- A distributed system consists of multiple independent computers (nodes) connected by a network.
- Each machine has its own local clock, but due to hardware differences and network delays, these clocks drift (move at slightly different speeds).
- Unlike in a single system, there is no shared physical clock to keep all nodes in perfect sync.

Problem with Physical Clocks

- Multiple physical clocks are generally considered desirable, which yields two problems:
- (1) How do we synchronize them with real world clocks.
- (2) How do we synchronize the clocks with each other?

Lamport's Logical Clock

Lamport's Logical Clock was created by Leslie Lamport. It is a procedure to determine the order of events occurring. It provides a basis for the more advanced vector clock algorithm Due to the absence of a Global Clock in a Distributed Operating System Lamport Logical Clock is needed.

Lamport’s Algorithm

- **Happened before relation(->):** a ~ b, means 'a' happened before 'b'.
- **Logical Clock:** The criteria for the logical clocks are:
  - [C1]: C_i (a) < C_i(b), [C_i -> Logical Clock, If 'a' happened before 'b', then time of 'a' will be less than 'b' in a particular process.]
  - [C2]: C_i(a) < C_j(b), [Clock value of C_i(a) is less than C_j(b)]

Reference

- **Process:** P_i
- **Event: E_{ij}**, where i is the process in number and j: $\mathbf{j_{th}}$ event in the $\mathbf{i^{th}}$ **process**.
- $\mathbf{t_m}$**:** vector time span for message m.
- $\mathbf{C_i}$ vector clock associated with process $\mathbf{P_i}$, the $\mathbf{j^{th}}$ element is **Ci[j]** and contains $\mathbf{P_i}$**'s** latest value for the current time in process $\mathbf{P_j}$.
- **d:** drift time, generally d is 1.

Lamport's Algorithm

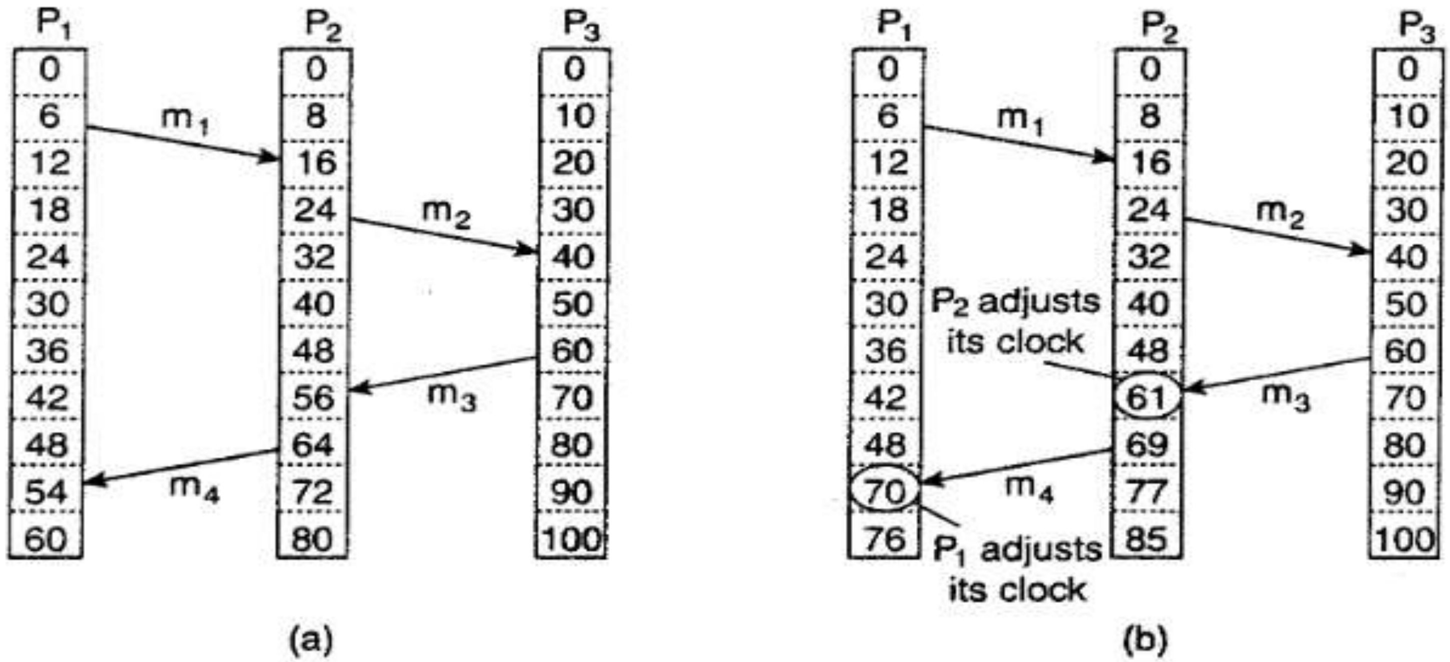


Figure 6-9. (a) Three processes, each with its own clock. The clocks run at different rates. (b) Lamport's algorithm corrects the clocks.

Vector Clock

- **Definition**: An improved logical clock mechanism using a vector of counters.

Working

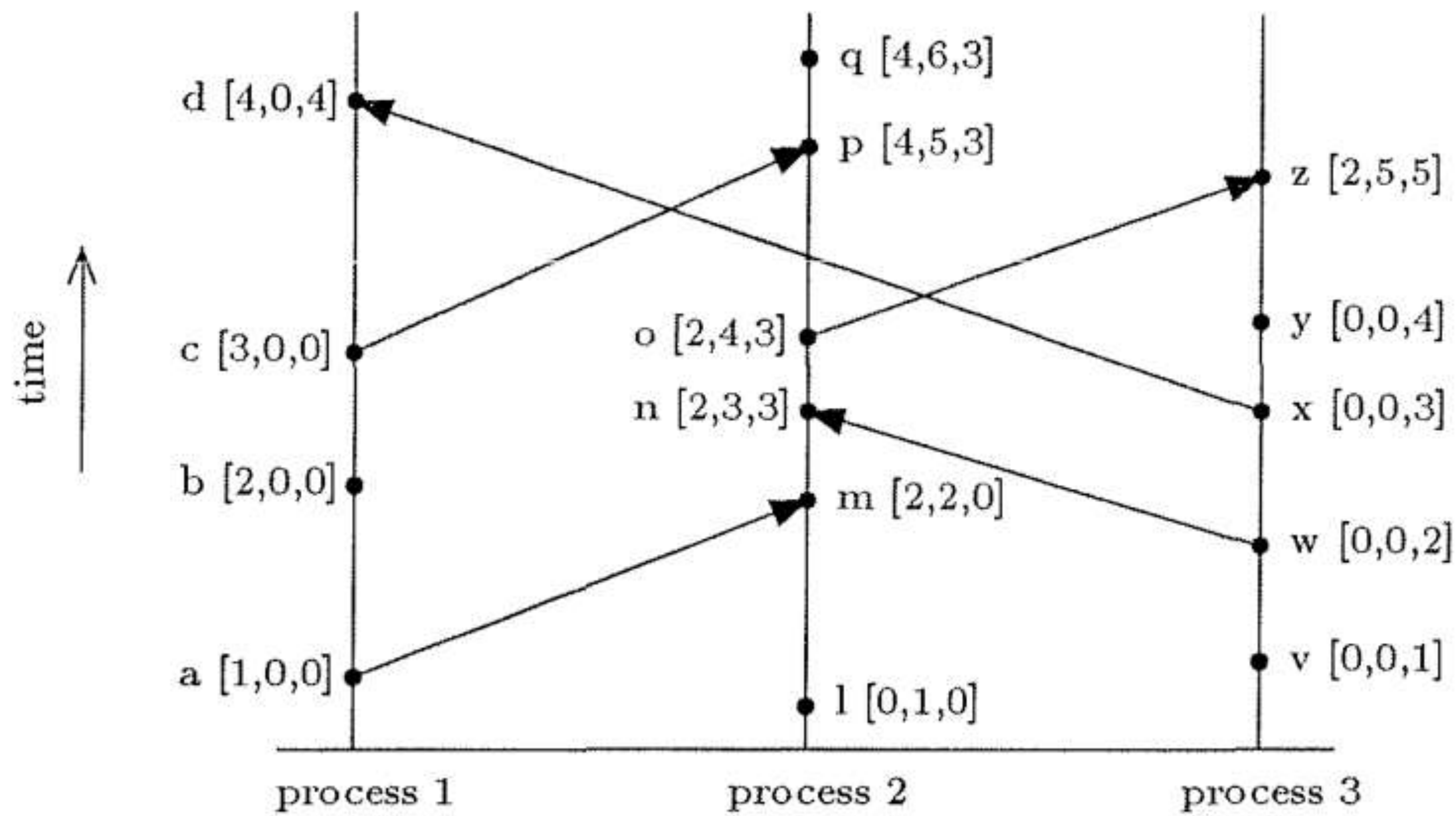
- Each process maintains a vector [P1, P2, ..., Pn] of size = number of processes.
- On a local event : increment own entry.
- On sending message : send the vector.
- On receiving message : take element-wise maximum of local vector and received vector.

Vector Clock

Properties

- If VC(a) < VC(b) (element-wise), then a → b.
- If neither VC(a) ≤ VC(b) nor VC(b) ≤ VC(a), then a || b (concurrent).
- **Advantage**: Can detect concurrent events.
- **Limitation:** Higher overhead (vector size grows with number of processes).

Vector Clock



Vector Clock

Aspect	**Lamport Clock**	**Vector Clock**
Structure	Single integer counter per process	Vector of integers (size = no. of processes)
Ordering	Provides **partial ordering**	Provides **causal ordering**
Concurrency	Cannot detect concurrency	Can detect anomalies in concurrency
Overhead	Very low (just one counter)	High (vector maintained & exchanged)
Use cases	Simple event ordering	Precise causal dependency tracking