CS 3551 DISTRIBUTED COMPUTING

UNIT II LOGICAL TIME AND GLOBAL STATE 10

Logical Time: Physical Clock Synchronization: NTP – A Framework for a System of Logical Clocks – Scalar Time – Vector Time; Message Ordering and Group Communication: Message Ordering Paradigms – Asynchronous Execution with Synchronous Communication – Synchronous Program Order on Asynchronous System – Group Communication – Causal Order – Total Order; Global State and Snapshot Recording Algorithms: Introduction – System Model and Definitions – Snapshot Algorithms for FIFO Channels.

LIKE
COMMENT
SHARE
SUBSCRIBE

Physical clock synchronization: NTP

- **Need of Knowing Time:**
  - The time of the day at which an event happened on a specific machine in the network.
  - The time interval between two events that happened on different machines in the network.
  - The relative ordering of events that happened on different machines in the network.
- **Need for synchronization** J_1, J_2, J_3
  - In database systems, the order in which processes perform updates on a database is important to ensure a consistent, correct view of the database.To ensure the right ordering of events, a common notion of time between co-operating processes becomes imperative.
  - Improves the performance of distributed algorithms by replacing communication with local computation.
  - Distributed protocols use timeouts, and their performance depends on how well physically dispersed processors are time-synchronized. Design of such applications is simplified when clocks are synchronized.
- Clock synchronization is the process of ensuring that physically distributed processors have a common notion of time. It has a significant effect on many problems like secure systems, fault diagnosis and recovery, scheduled operations, database systems, and real-world clock values.

Definitions

6pm
W → 6.15pm

1. **Time** The time of a clock in a machine p is given by the function $C_p(t)$, where $C_p(t) = t$ for a perfect clock.
2. **Frequency** Frequency is the rate at which a clock progresses. The frequency at time t of clock C_a is $C'_a(t)$.
3. **Offset** Clock offset is the difference between the time reported by a clock and the *real time*. The offset of the clock C_a is given by $C_a(t) - t$. The offset of clock C_a relative to C_b at time $t \geq 0$ is given by $C_a(t) - C_b(t)$.
4. **Skew** The skew of a clock is the difference in the frequencies of the clock and the perfect clock. The skew of a clock C_a relative to clock C_b at time t is $C'_a(t) - C'_b(t)$.
   If the skew is bounded by ρ, then as per Eq.(3.1), clock values are allowed to diverge at a rate in the range of $1 - \rho$ to $1 + \rho$.
5. **Drift (rate)** The drift of clock C_a is the second derivative of the clock value with respect to time, namely, $C''_a(t)$. The drift of clock C_a relative to clock C_b at time t is $C''_a(t) - C''_b(t)$.

Different Clocks

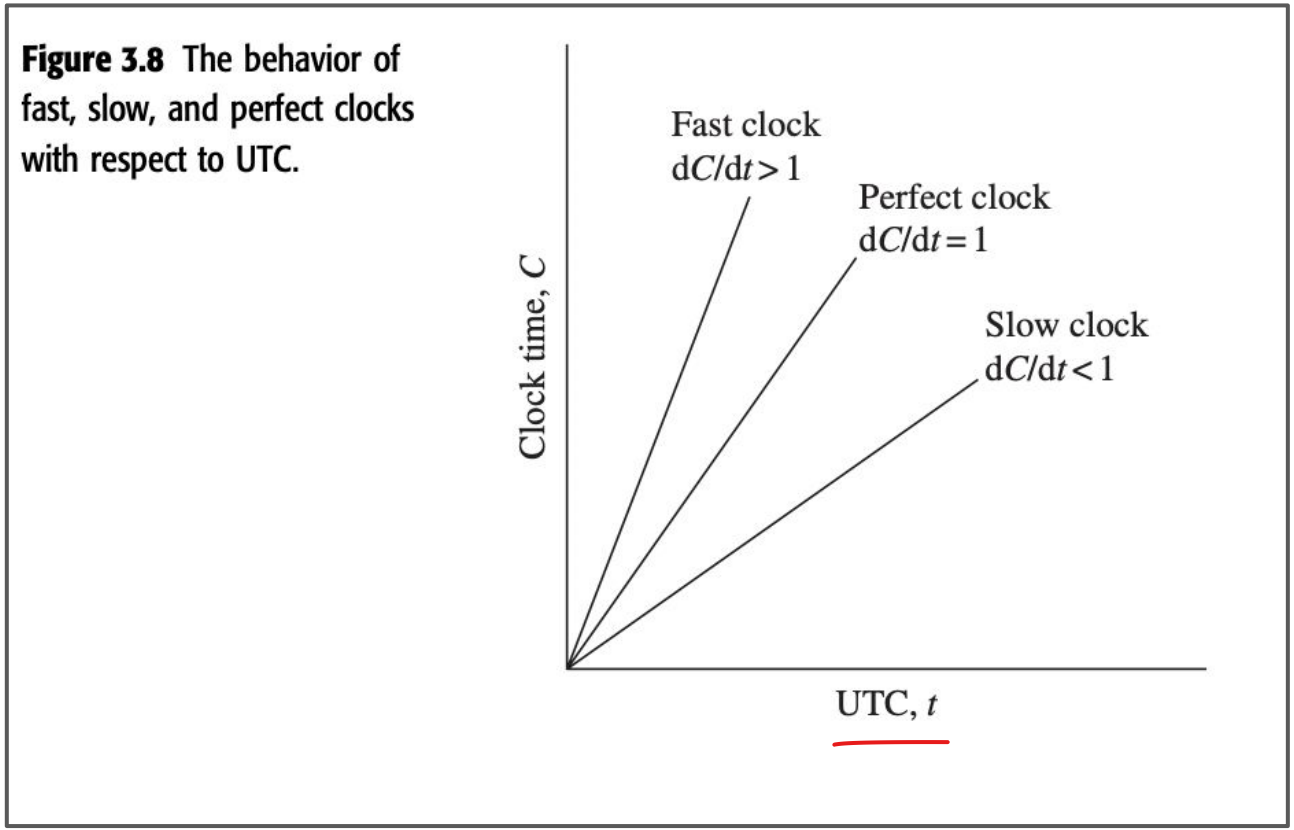


Figure 3.8 The behavior of fast, slow, and perfect clocks with respect to UTC.

Offset Delay Estimation using Network Transmission Protocol

- In practice, a source node cannot accurately estimate the local time on the target node due to varying message or network delays between the nodes.
- This protocol employs a very common practice of **performing several trials and chooses the trial with the minimum delay.**
- The design of NTP involves a hierarchical tree of time servers.
  - The primary server at the root synchronizes with the UTC.
  - The next level contains secondary servers, which act as a backup to the primary server.
  - At the lowest level is the synchronization subnet which has the clients.

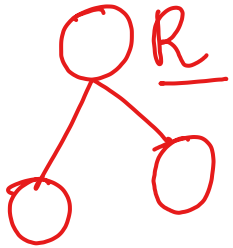
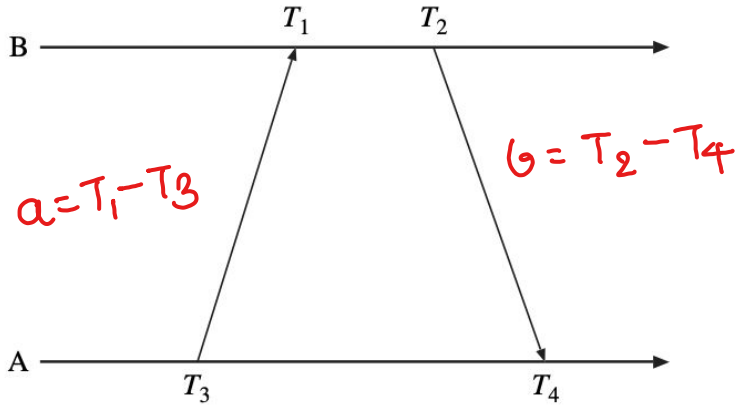


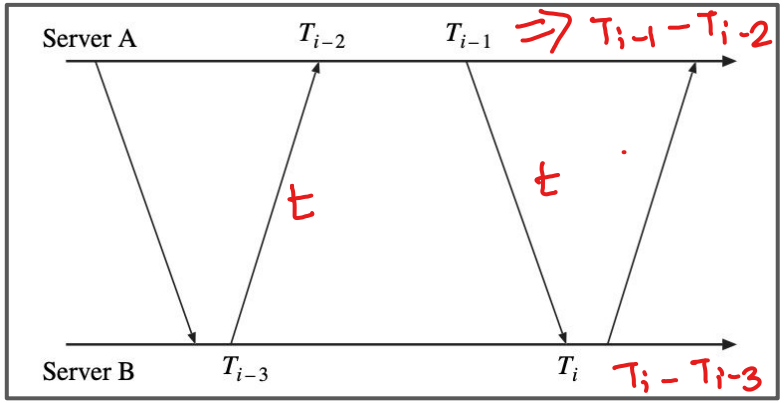
Figure 3.9 Offset and delay estimation [15].



differential delay $\theta = \frac{a+b}{2}$

Round trip delay $\delta = a - b$ //

NTP Synch Protocol



- A pair of servers in symmetric mode exchange pairs of timing messages.
- A store of data is then built up about the relationship between the two servers (pairs of offset and delay).

  Specifically, assume that each peer maintains pairs (O_i, D_i), where:

  O_i – measure of offset (θ)

  D_i – transmission delay of two messages (δ).
- The offset corresponding to the minimum delay is chosen. Specifically, the delay and offset are calculated as follows. Assume that message m takes time t to transfer and m' takes t' to transfer.

A local clock – A(t)
B " " → B(t)

A(t) > B(t)

① $A(t) = B(t) + O \Rightarrow B(t) = A(t) - O$

② $T_{i-2} = T_{i-3} + t + O$

$T_i = T_{i-1} + t - O$

$O = T_{i-2} - T_{i-3} + t$

$O = T_{i-1} + t - T_i$

$2O_i = T_{i-2} - T_{i-3} - t + T_{i-1} + t - T_i$

$$O_i = \frac{T_{i-2} - T_{i-3} - t + T_{i-1} + t - T_i}{2}$$

$$D_i = (T_i - T_{i-3}) - (T_{i-1} - T_{i-2})$$

- The offset between A's clock and B's clock is O. If A's local clock time is $A(t)$ and B's local clock time is $B(t)$, we have

$$A(t) = B(t) + O. \tag{3.3}$$

Then,

$$T_{i-2} = T_{i-3} + t + O, \tag{3.4}$$

$$T_i = T_{i-1} - O + t'. \tag{3.5}$$

Assuming $t = t'$, the offset O_i can be estimated as

$$O_i = (T_{i-2} - T_{i-3} + T_{i-1} - T_i)/2. \tag{3.6}$$

The round-trip delay is estimated as

$$D_i = (T_i - T_{i-3}) - (T_{i-1} - T_{i-2}). \tag{3.7}$$

- The eight most recent pairs of (O_i, D_i) are retained.
- The value of O_i that corresponds to minimum D_i is chosen to estimate O.

LIKE

COMMENT

SHARE

SUBSCRIBE

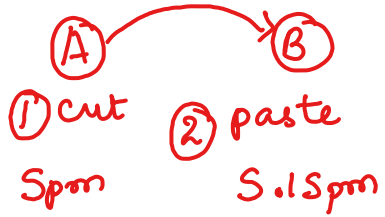
CS 3551 DISTRIBUTED COMPUTING

UNIT II LOGICAL TIME AND GLOBAL STATE 10

Logical Time: Physical Clock Synchronization: NTP – A Framework for a System of Logical Clocks – Scalar Time – Vector Time; Message Ordering and Group Communication: Message Ordering Paradigms – Asynchronous Execution with Synchronous Communication – Synchronous Program Order on Asynchronous System – Group Communication – Causal Order – Total Order; Global State and Snapshot Recording Algorithms: Introduction – System Model and Definitions – Snapshot Algorithms for FIFO Channels.

LIKE
COMMENT
SHARE
SUBSCRIBE

What is physical clock?



- A physical clock in a system refers to a hardware-based timekeeping device that tracks the passage of time based on a stable oscillating signal, typically provided by a quartz crystal.
- It provides a reference for measuring the duration between events, scheduling tasks, and maintaining synchronization within the system.

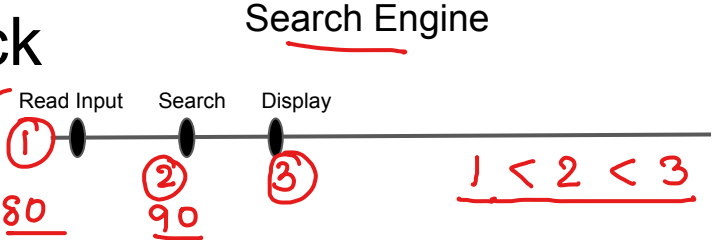
Need of Clock

1. **Synchronization of Operations:** Clocks in a CPU help synchronize the execution of different operations and instructions, ensuring that the various components work in coordination with each other, and enabling the orderly execution of tasks.
2. **Regulation of Timing:** Clock signals regulate the timing of various CPU operations, ensuring that instructions are fetched, decoded, and executed within specific time intervals known as clock cycles. This timing precision is crucial for maintaining the proper functioning of the CPU.
3. **Prevention of Hazards:** Clocks aid in preventing hazards such as data hazards and control hazards by enabling the precise timing of instruction fetching and execution, thus avoiding conflicts and ensuring the smooth flow of data and instructions through the CPU.
4. **Pipelining Efficiency:** Clocks are essential for pipelining, a technique that allows multiple instructions to be processed simultaneously. The regular clock signals ensure that each stage of the pipeline operates in synchronization, maximizing the CPU's efficiency in executing multiple instructions concurrently.
5. **Control of CPU Frequency:** Clocks control the frequency of the CPU, determining how many instructions the CPU can execute in a given unit of time. Higher clock frequencies often result in faster processing speeds, enabling the CPU to handle more complex tasks and calculations.

Why not physical clocks in distributed system?

1. **Clock Drift:** Different machines may have their own local clocks, which can drift apart over time due to differences in hardware, temperature, and other factors. This can lead to significant discrepancies in timekeeping between different nodes, making it difficult to establish a reliable global time.
2. **Network Latency and Delay:** Network communication introduces unpredictable latency and delays. Messages may take varying amounts of time to travel between nodes, making it challenging to precisely synchronize events across the entire distributed system.
3. **Faults and Failures:** Hardware failures, software errors, and network issues can further disrupt the accuracy of physical clocks. In the event of a node failure or network partition, maintaining a consistent global time becomes even more challenging.
4. **Scalability:** In large-scale distributed systems with numerous interconnected nodes, maintaining a single global clock that accurately represents the order of events across the entire system becomes increasingly complex and impractical.

So what is the solution? Logical Clock



Prepare → Write

C → E

1. In distributed computing, logical time refers to a mechanism used to establish an ordering of events that occur across different nodes or processes in a distributed system.
2. It provides a way to reason about the causal relationships between events, even in the absence of a globally consistent physical time.
3. Logical time is crucial for coordinating activities, maintaining consistency, and enabling the implementation of various distributed algorithms and protocols.
4. It also ensures proper order of events.
5. Every event is assigned a timestamp and the causality relation between events can be generally inferred from their timestamps.
6. The timestamps assigned to events obey the fundamental monotonicity property; that is, if an event a causally affects an event b, then the timestamp of a is smaller than the timestamp of b

E_1 E_2

35 10

E_2, E_1

Methods of Representing Logical Time

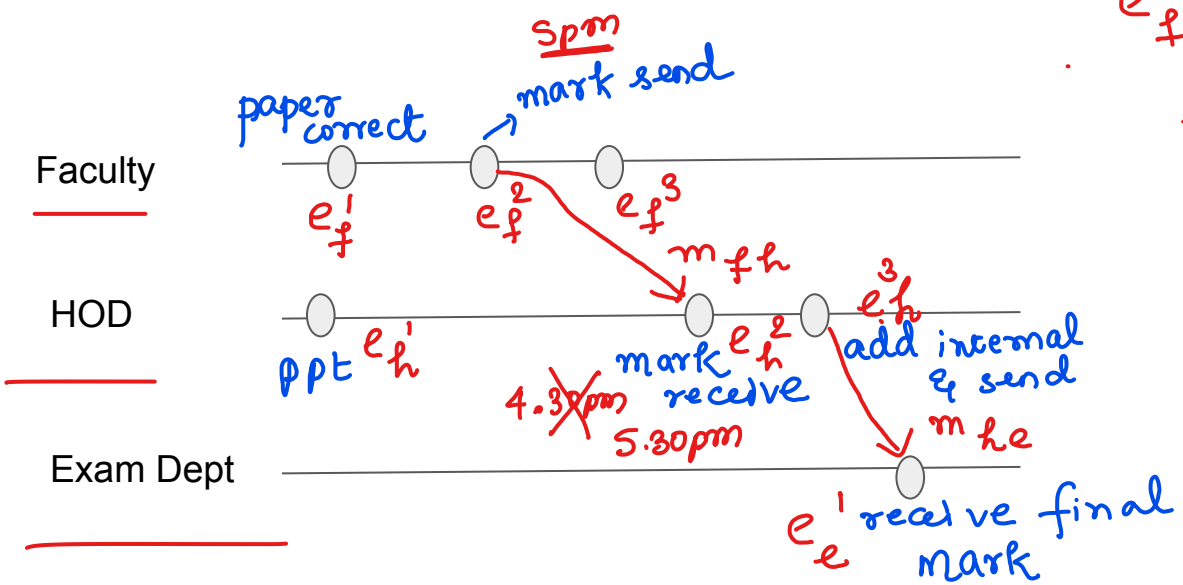
1. Lamport's scalar clocks, the time is represented by non-negative integers
2. The time is represented by a vector of non-negative integers
3. The time is represented as a matrix (array) of non-negative integers

What is Causal Relationship?

1. Brush -> Tea
2. Bike Start -> Bike Drive
3. Cook -> Eat

- A causal relationship refers to a cause-and-effect connection between two or more events or variables, where one event (the cause) brings about another event (the effect)
- It signifies that the occurrence of one event leads to or affects the occurrence of another event.

Example Event Diagram



$e_f^1 \rightarrow e_h^1$ X

$e_h^1 \rightarrow e_f^1$ X

$e_f^2 \rightarrow e_h^2$ ✓✓

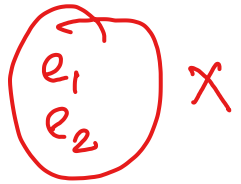
Benefits of Causal Precedence

- **Distributed algorithms design** The knowledge of the causal precedence relation among events helps ensure liveness and fairness in mutual exclusion algorithms, helps maintain consistency in replicated databases, and helps design correct deadlock detection algorithms to avoid phantom and undetected deadlocks.
- **Tracking of dependent events** In distributed debugging, the knowledge of the causal dependency among events helps construct a consistent state for resuming reexecution; in failure recovery, it helps build a checkpoint; in replicated databases, it aids in the detection of file inconsistencies in case of a network partitioning.

- **Knowledge about the progress** The knowledge of the causal dependency among events helps measure the progress of processes in the distributed computation. This is useful in discarding obsolete information, garbage collection, and termination detection.
- **Concurrency measure** The knowledge of how many events are causally dependent is useful in measuring the amount of concurrency in a computation. All events that are not causally related can be executed concurrently. Thus, an analysis of the causality in a computation gives an idea of the concurrency in the program.

$e_4 \not\rightarrow e_5$

$e_5 \not\rightarrow e_4$



$e_1 \rightarrow e_2$

LIKE

COMMENT

SHARE

SUBSCRIBE

CS 3551 DISTRIBUTED COMPUTING

UNIT II LOGICAL TIME AND GLOBAL STATE 10

Logical Time: Physical Clock Synchronization: NTP – A Framework for a System of Logical Clocks – Scalar Time – Vector Time; Message Ordering and Group Communication: Message Ordering Paradigms – Asynchronous Execution with Synchronous Communication – Synchronous Program Order on Asynchronous System – Group Communication – Causal Order – Total Order; Global State and Snapshot Recording Algorithms: Introduction – System Model and Definitions – Snapshot Algorithms for FIFO Channels.

LIKE
COMMENT
SHARE
SUBSCRIBE

Framework for System of Logical Clocks

$e_1 \Rightarrow 4$ $e_3 \Rightarrow 1$
$e_2 \Rightarrow 8$ $e_4 \Rightarrow 8$

$T \rightarrow (1, 2, 3 \ldots)$

- A system of logical clocks consists of a time domain T and a logical clock C.
- The logical clock C is a function that maps an event e in a distributed system to an element in the time domain T, denoted as C(e) and called the timestamp of e, and is defined as

$$C : H \mapsto T,$$

Consistent: for two events e_i and e_j, $e_i \rightarrow e_j \Longrightarrow C(e_i) < C(e_j)$.

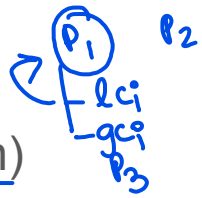
Strongly: for two events e_i and e_j, $e_i \rightarrow e_j \Leftrightarrow C(e_i) < C(e_j)$,

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

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

Implementing Logical Clocks

Every process needs data structure(store logical time)+algorithms(updation)



Each process p_i maintains data structures that allow it the following two capabilities:

1. A *local logical clock*, denoted by lc_i, that helps process p_i measure its own progress.
2. A *logical global clock*, denoted by gc_i, that is a representation of process p_i's local view of the logical global time. It allows this process to assign consistent timestamps to its local events. Typically, lc_i is a part of gc_i.

- **R1** This rule governs how the local logical clock is updated by a process when it executes an event (send, receive, or internal).
- **R2** This rule governs how a process updates its global logical clock to update its view of the global time and global progress. It dictates what information about the logical time is piggybacked in a message and how this information is used by the receiving process to update its view of the global time.

LIKE

COMMENT

SHARE

SUBSCRIBE

Scalar Time - non-negative integer

1, 2, 25, 3, 30 .

(1)

- **R1** Before executing an event (send, receive, or internal), process p_i executes the following:

$$C_i := C_i + d \qquad (d > 0).$$

d=1 4,5,6 ...

(2)

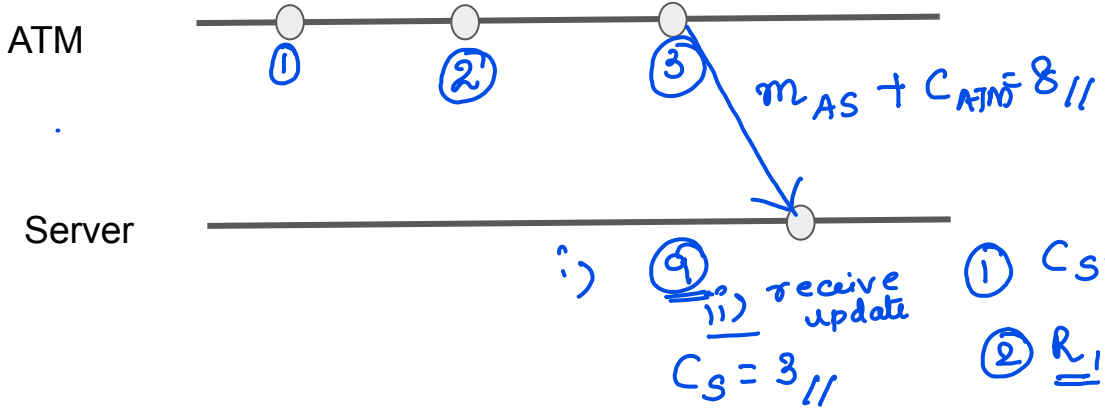
- **R2** Each message piggybacks the clock value of its sender at sending time. When a process p_i receives a message with timestamp C_{msg}, it executes the following actions:
  1. $C_i := max(C_i, C_{msg})$;
  2. execute **R1**;
  3. deliver the message.

d=1

i) 5+1=6 ✓ i) 6+1=7

i) 7+1=8

$C_{ATM}=5$ ii) cash dispense ii) receipt ii) update server



Properties of Scalar Time

1. Consistency → $e_i \rightarrow e_j \Rightarrow C(e_i) < C(e_j)$
2. Total Ordering
   a. Case 1 → all timestamp unique
   b. Case 2 → repeat
3. Event Counting
4. No strong consistency

$e_1 \rightarrow 4 \quad e_2 - 8 \quad e_3 - 12$

e_1, e_2, e_3

no → 1

$e_{id \rightarrow 1} - 4$ ✓

$e_{4}^{1} - 4$

event-id low ⇒ 1st

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

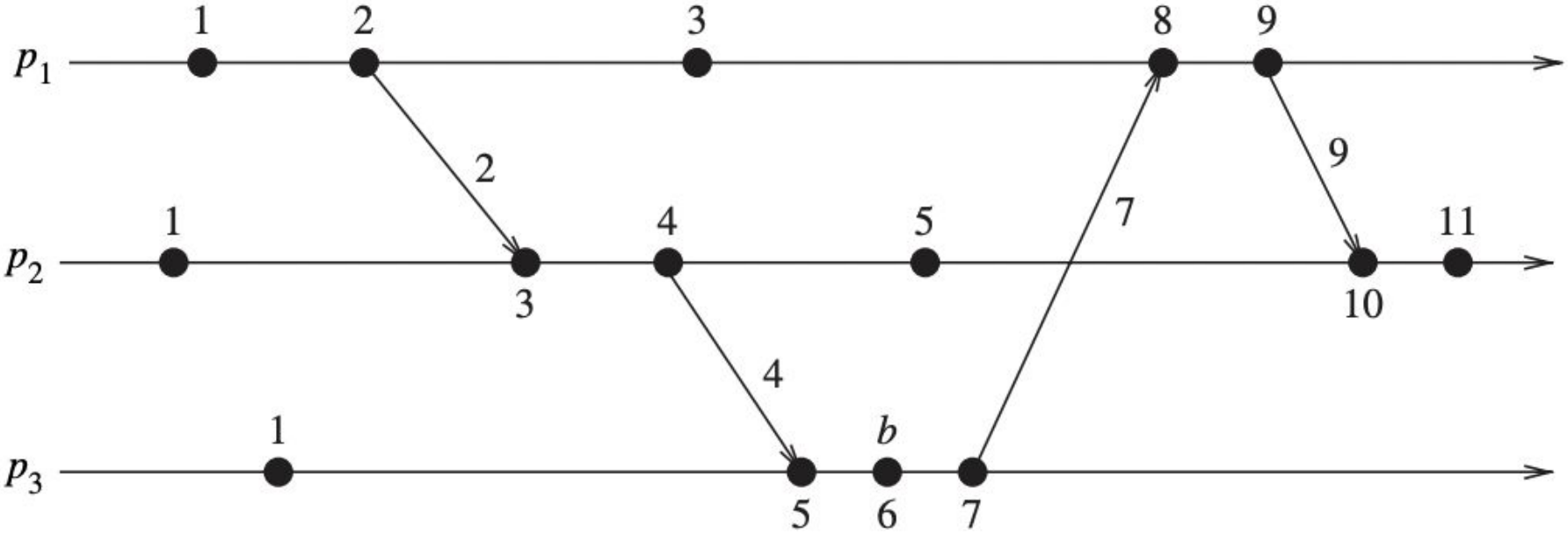
d = 1

h - 1 = 6 //

$e_4^3 - 7$ //

h

Figure 3.1 shows the evolution of scalar time with $d=1$.



LIKE

COMMENT

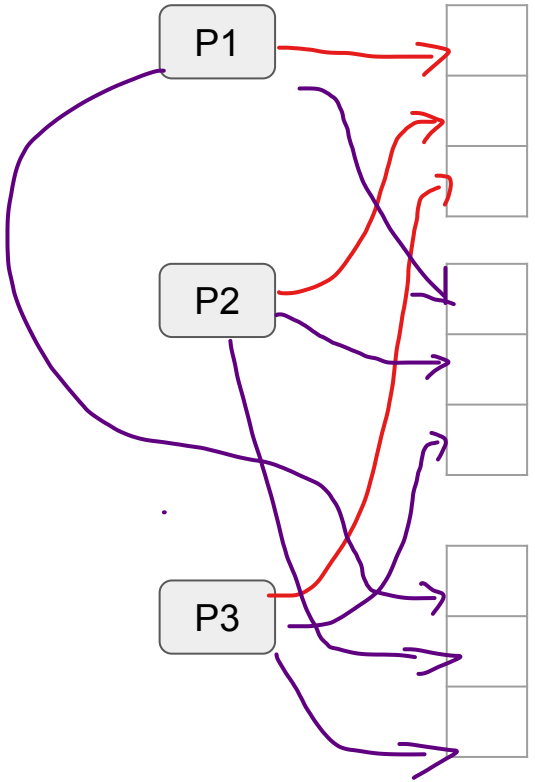
SHARE

SUBSCRIBE

Vector Time

- array of time
- non-negative.
- size = n

$vt_i[i]$ - The process's own logical time
$vt_i[j]$ - Info on j's process local time



$vt_1[1]$ → own time

$vt_1[3]$

Rules

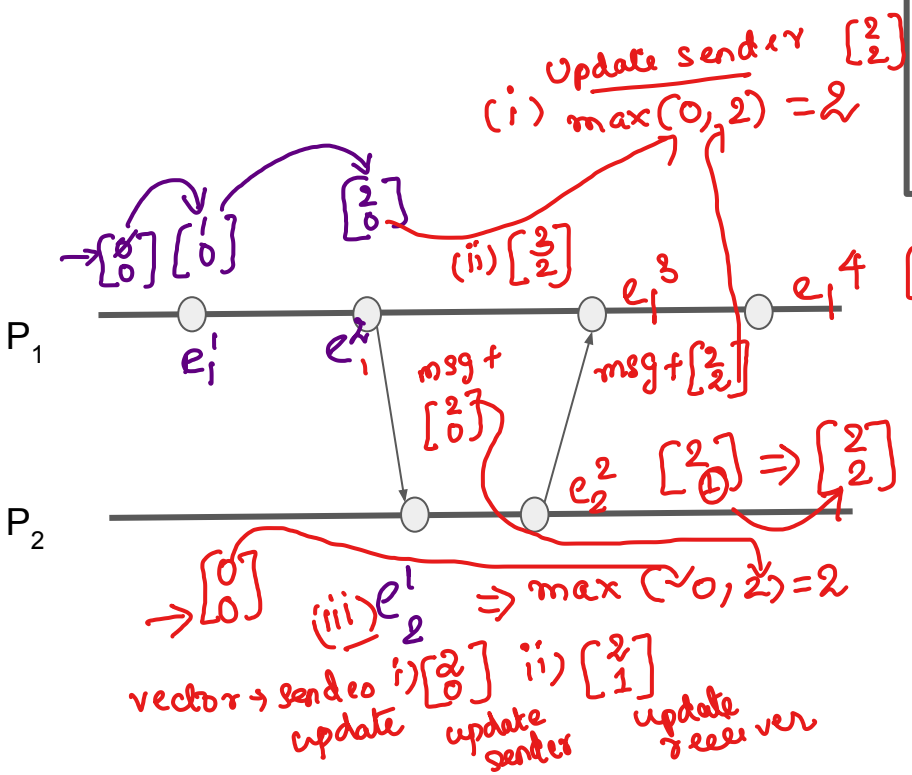
- **R1** Before executing an event, process p_i updates its local logical time as follows:

$$vt_i[i] := vt_i[i] + d \qquad (d > 0).$$

- **R2** Each message m is piggybacked with the vector clock vt of the sender process at sending time. On the receipt of such a message (m, vt), process p_i executes the following sequence of actions:
  1. update its global logical time as follows:

$$1 \le k \le n \ : \ vt_i[k] := max(vt_i[k], vt[k]);$$

  2. execute **R1**;
  3. deliver the message m.



Properties on Comparing two Timestamps

The following relations are defined to compare two vector timestamps, vh and vk:

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

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

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

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

Properties

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

$C(e_j) < C(e_j) \not\Rightarrow$

$e_i \rightarrow e_j$

1. Isomorphism

If two events x and y have timestamps vh and vk, respectively, then

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

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

2. Strong Consistency - The system of vector clocks is strongly consistent; thus, by examining the vector timestamp of two events, we can determine if the events are causally related.
3. Event Counting -

Event counting

If d is always 1 in rule **R1**, then the ith component of vector clock at process p_i, $vt_i[i]$, denotes the number of events that have occurred at p_i until that instant.

Size of Vectors

- A vector clock provides the latest known local time at each other process. If this information in the clock is to be used to explicitly **track the progress at every other process**, then a vector clock of size n is necessary.
- A popular use of vector clocks is to determine the causality between a pair of events.

> It can be shown that a size equal to the dimension of the partial order $(E, \prec)$ is necessary, where the upper bound on this dimension is n.

LIKE

COMMENT

SHARE

SUBSCRIBE