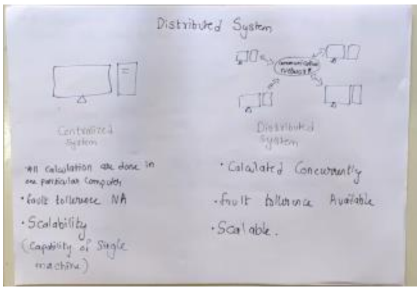
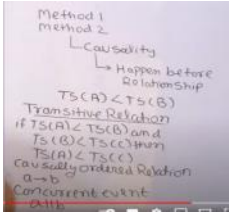
Fundamental of Distributed Computing

14 January 2024 22:14



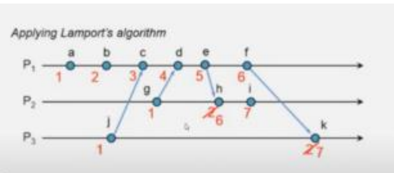


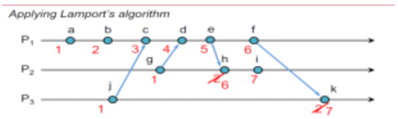
Casual relation yeh kehta hai ki agr A mein kuch change ho raha hai to uska asar B mein bhi padega

Lamport's Algorithm

If *a* and *b* occur on different processes that do not exchange messages, then neither $a \rightarrow b$ nor $b \rightarrow a$ are true

- These events are **concurrent**
- Otherwise, they are **causal**



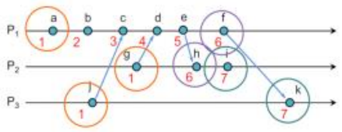


We have good ordering where we used to have bad ordering:

e → h and 5 < 6
f → k and 6 < 7

• When a message arrives:
  if receiver's *clock* < *message_timestamp*
    set system clock to (*message_timestamp* + 1)
  else do nothing

Problem: Identical timestamps



a→b, b→c, … : local events sequenced

i→c, f→d , d→g, … : Lamport imposes a *send→receive* relationship

Concurrent events (e.g., *b* & *g*; *i* & *k*) *may* have the same timestamp … or not

Unique timestamps (total ordering)

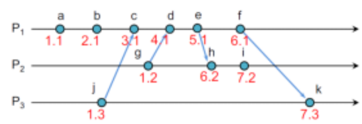
We can force each timestamp to be unique

- Define global logical timestamp (T_i, i)
  - T_i represents local Lamport timestamp
  - i represents process number (globally unique)
    - e.g., (host address, process ID)
- Compare timestamps:
  $(T_i, i) < (T_j, j)$
  if and only if
  $T_i < T_j$ or
  $T_i = T_j$ and $i < j$

Does not necessarily relate to actual event ordering

Solution

Unique (totally ordered) timestamps



Problem: Detecting causal relations

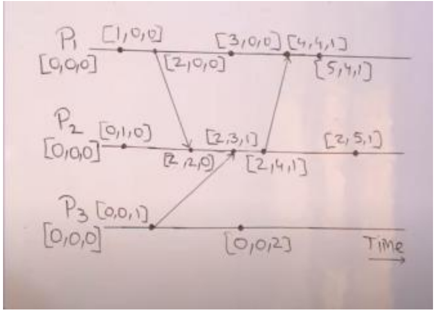
If $L(e) < L(e')$
- We cannot conclude that $e \rightarrow e'$

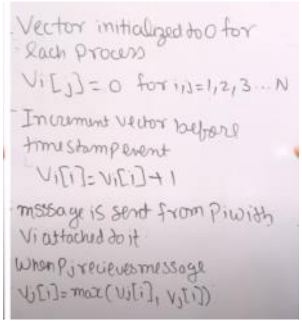
By looking at Lamport timestamps
- We cannot conclude which events are causally related

Solution: use a vector clock

Vector clocks are a way to prove the sequence of events bt keeping version history based on each process that made changes to an object

Vector Clock





Distributed Mutual Exclusion Algorithms

Lamport's Distributed Mutual Exclusion Algorithm
(Non-token)

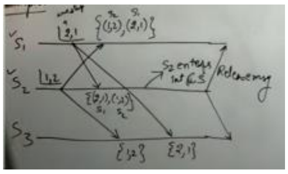
a request with smaller timestamp will be given permission to execute critical section first than a request with larger timestamp.

Three type of messages (**REQUEST**, **REPLY** and **RELEASE**)

- Every site Si, keeps a queue to store critical section requests ordered by their timestamps. **request_queue** denotes the queue of site Si
- A timestamp is given to each critical section request using Lamport's logical clock.
- Timestamp is used to determine priority of critical section requests. Smaller timestamp gets high priority over larger timestamp. The execution of critical section request is always in the order of their timestamp.

Algorithm:

- **To enter Critical section:**
  - When a site S_i wants to enter the critical section, it sends a request message **Request(ts_i, i)** to all other sites and places the request on **request_queue$_i$**. Here, Ts_i denotes the timestamp of Site S_i
  - When a site S_j receives the request message **REQUEST(ts_i, i)** from site S_i, it returns a timestamped REPLY message to site S_i and places the request of site S_i on **request_queue$_j$**
- **To execute the critical section:**
  - A site S_i can enter the critical section if it has received the message with timestamp larger than **(ts_i, i)** from all other sites and its own request is at the top of **request_queue$_i$**
- **To release the critical section:**
  - When a site S_i exits the critical section, it removes its own request from the top of its request queue and sends a timestamped **RELEASE** message to all other sites
  - When a site S_j receives the timestamped **RELEASE** message from site S_i, it removes the request of S_i from its request queue



Message Complexity: Lamport's Algorithm requires invocation of 3(N – 1) messages per critical section execution. These 3(N – 1) messages involves

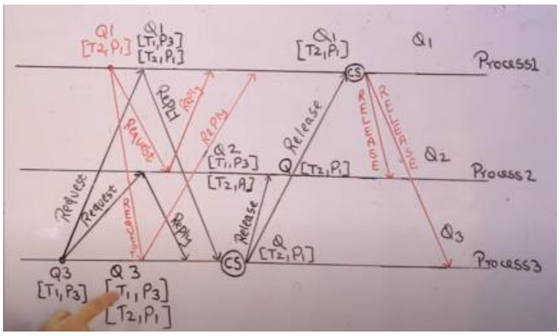
- (N – 1) request messages
- (N – 1) reply messages
- (N – 1) release messages

Drawbacks of Lamport's Algorithm:

- **Unreliable approach:** failure of any one of the processes will halt the progress of entire system.
- **High message complexity:** Algorithm requires 3(N-1) messages per critical section invocation.

Performance:

- Synchronization delay is equal to maximum message transmission time
- It requires 3(N – 1) messages per CS execution.
- Algorithm can be optimized to 2(N – 1) messages by omitting the REPLY message in some situations.



An optimization in performance in 3(N-1)

In Lamport's algorithm, when one site asks for permission (let's call it Si), and another site (Sj) already asked earlier with a higher timestamp, Sj doesn't need to reply to Si. Si can figure out that Sj doesn't have any earlier requests pending.

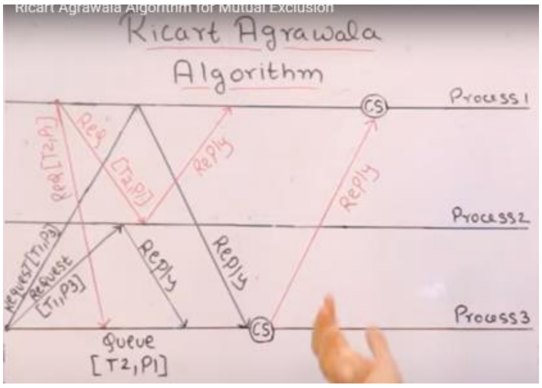
This optimization makes Lamport's algorithm work more efficiently, needing fewer messages (between 3(N-1) and 2(N-1) messages) each time a site wants to do something important, like entering a critical section.

Theorem: Lamport's algorithm achieves mutual exclusion.
Proof:

Certainly! In Lamport's algorithm for achieving mutual exclusion:

- Suppose two sites, Si and Sj, are trying to enter the critical section at the same time.
- The proof shows that if both Si and Sj are in the critical section concurrently, it leads to a contradiction.
- The contradiction arises from the fact that Si's request with a smaller timestamp is at the top of both Si and Sj's request queues simultaneously.
- The contradiction demonstrates that Lamport's algorithm effectively prevents two sites from being in the critical section at the same time, ensuring mutual exclusion in a distributed system.

Ricart Agrawala Algorithm



Algorithm:

- **To enter Critical section:**
  - When a site S_i wants to enter the critical section, it send a timestamped **REQUEST** message to all other sites.
  - When a site S_j receives a **REQUEST** message from site S_i, It sends a **REPLY** message to site S_i if and only if
    - Site S_j is neither requesting nor currently executing the critical section.
    - In case Site S_j is requesting, the timestamp of Site S_i's request is smaller than its own request.
- **To execute the critical section:**
  - Site S_i enters the critical section if it has received the **REPLY** message from all other sites.
- **To release the critical section:**
  - Upon exiting site S_i sends **REPLY** message to all the deferred requests.

Theorem: Ricart-Agrawala algorithm achieves mutual exclusion.
Proof:

- Proof is by contradiction. Suppose two sites S_i and S_j' are executing the CS concurrently and S_i's request has higher priority than the request of S_j. Clearly, S_i received S_j's request after it has made its own request.
- Thus, S_j can concurrently execute the CS with S_i only if S_i returns a REPLY to S_j (in response to S_j's request) before S_i exits the CS.
- However, this is impossible because S_j's request has lower priority. Therefore, Ricart-Agrawala algorithm achieves mutual exclusion.

Message Complexity: Ricart–Agrawala algorithm requires invocation of 2(N – 1) messages per critical section execution. These 2(N – 1) messages involves

Q4. [CO2 2+ 2 marks] a) In a system, each process may use critical section many times before another process requires it. Explain why Ricart-Agarwala ME algorithm is inefficient for this case and describe how its performance be improved.

b) A DS may have multiple independent critical regions (CR). Imagine Process 0 wants to enter CR A and process 1 wants to enter CR B. Can Ricart-Agarwala ME algorithm lead to deadlocks?

Q5 [CO2 2 Mark-s] Raymond's tree based algorithm docs no! order CS requests. based on time. Explain whether the algorithm is fair or not

a) Ricart-Agrawala can be inefficient when a process uses the critical section many times due to frequent message exchanges. To improve:

1. **Batching Requests:**
   - Processes can batch critical section requests to reduce message frequency.

2. Local Permission Mechanism:
• Allow processes to locally grant permission to enter the critical section without having to obtain global agreement from all other processes.
Processes can maintain a counter to keep track of the number of times they have entered the critical section and release the resources accordingly.

b) Yes, the Ricart-Agrawala algorithm can lead to deadlocks when multiple processes are trying to enter different critical regions simultaneously. If two processes request permission for different critical regions concurrently, a circular wait condition may occur, resulting in a deadlock. Consider modifying the algorithm or exploring alternatives for scenarios with multiple independent critical regions.

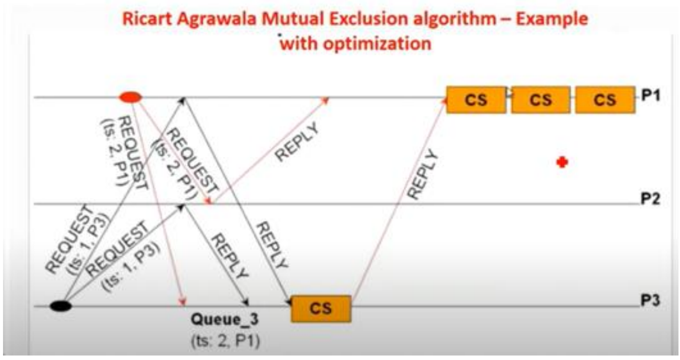
Q5 [CO2 2 Marks] Raymond's tree based algorithm does not order CS requests based on time. Explain whether the algorithm is fair or not

Q5 [CO2 2 Marks] Raymond's tree based algorithm does not order CS requests based on time. Explain whether the algorithm is fair or not

Q1 [CO1, Marks 3] What problem of Lamport\s clocks do vector clocks solve? Give 2 examples of events concurrent with the vector timestamp (2, 8 , 4).

Q2 [CO2, Marks 3+3] a) You want to select a distributed mutual exclusion algorithm for the scenario: You have 20 low-cost, unreliable computers with varying speeds connected to the same Ethernet switch. When a request for processing a job arrives, it should be assigned to one of these computers, which then processes the job. Exactly ONE computer should handle each request. Which distributed mutual exclusion algorithm would you use? Justify.
b) Compare Lamport's distributed mutual exclusion algorithm with Ricart Agrawala algorithm in terms of correctness, fairness, deadlock freedom and message complexity.

Compare Lamport's distributed mutual exclusion algorithm with Ricart Agrawala algorithm in terms of correctness, fairness, deadlock freedom and message complexity.

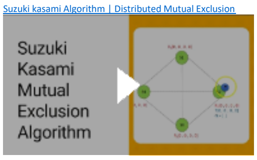


Vector clocks address the lack of causality information and false concurrency in Lamport's clocks. Two events concurrent with the vector timestamp (2, 8, 4) could be (2, 8, 7) and (2, 8, 2), where the first two entries match, indicating events in processes 1 and 2 are in the same state.

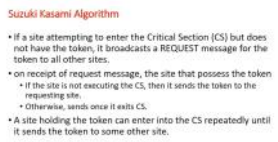
For 20 low-cost, unreliable computers connected to the same Ethernet switch, the **Token Ring Algorithm** is suitable. It provides sequential access to the critical section through a circulating token, ensuring exactly one computer processes each job request. The algorithm is simple, has low overhead, and is well-suited for scenarios with such computer characteristics.

Q4. [CO2 2+ 2 marks] a) In a system, each process may use critical section many times before another process requires it. Explain why Ricart-Agarwala ME algorithm is inefficient for this case and describe how its performance be improved.

[Suzuki kasami Algorithm | Distributed Mutual Exclusion]

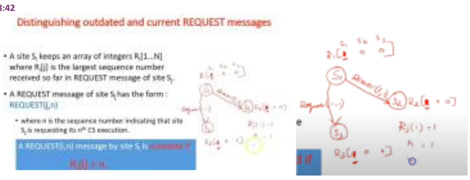


00:57

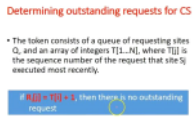


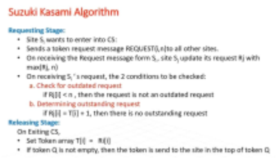
05:09

08:42

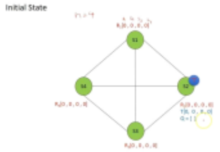


09:45



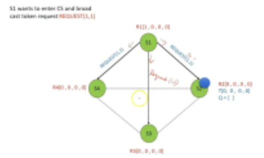


13:04

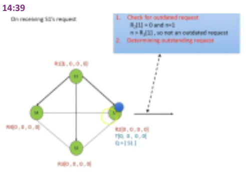


jo request karta hai uska token increase hota hai

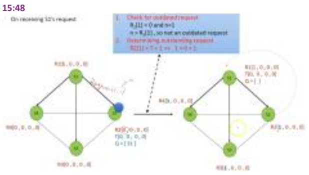
13:54



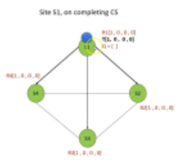
14:39



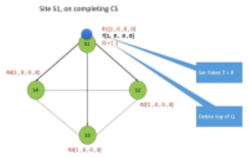
15:48



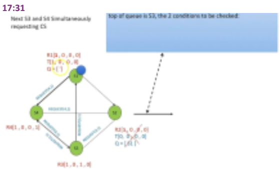
16:35



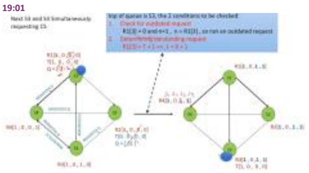
17:10



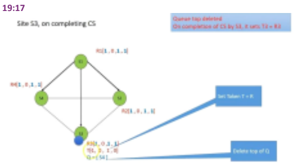
17:31



19:01



19:17



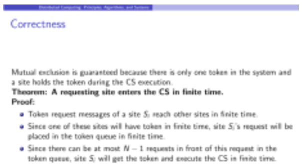
From <https://www.youtube.com/watch?v=Uy8hI-Gqic4>

Q6 [CO2 3 Marks] Show the working of Suzuki-Kazami algorithm in the following scenario: There are 1 processes P1 to P4. Initially the token is with P1 and then P2 and P3 send request to P1. P1 sends token to P2 While P2 is in CS, P1 and P4 broadcast request messages. What will be the contents of various data structures at each step?

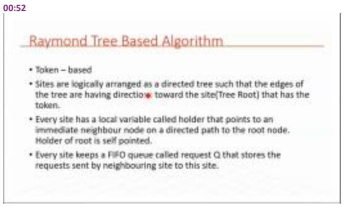
Q6 Show the working of Suzuki-Kazami algorithm in the follwing scenario: r here are I processes PI to PC. Initially the token is with PI and then P2 and P 3 send request to PI. PI sends token to P2 While P2 is in C'S, PI and P4 broadcast request messages. What Will be contents Of various data structures at each step?

b) Identify the Distributed System (DS) characteristic: (i) A DS allows new components to be integrated with existing components (ii)b) A DS can accommodate more users if required

b) Identify the Distributed System (D.S) characteristic: (i) A DS allows new components to be integrated with existing components (ii)b) A DS can accommodate more users if required



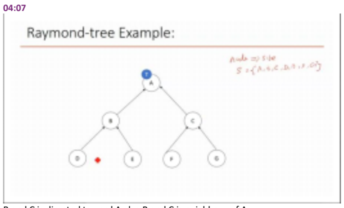
00:52



Tree data structure is used

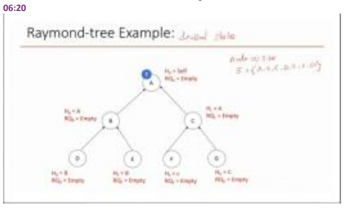
03:25

04:07

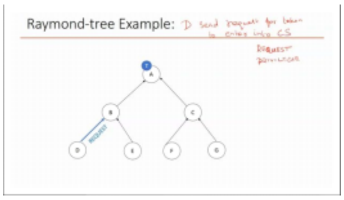


B and C is directed toward A also B and C is neighbour of A

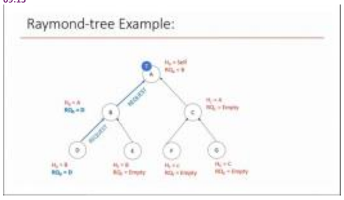
06:20



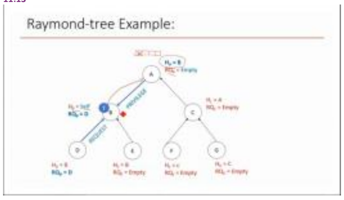
07:52



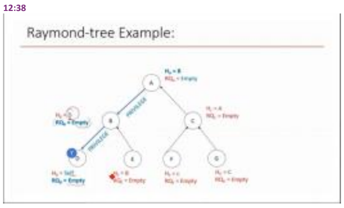
09:15



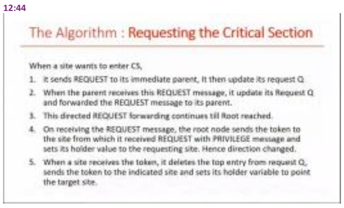
11:13



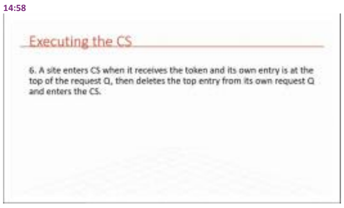
12:38



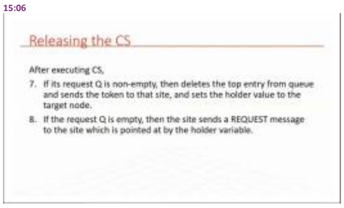
12:44



14:58

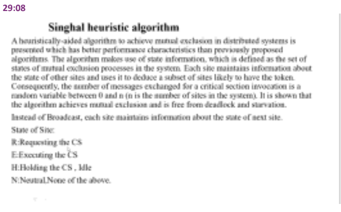


15:06



From <https://www.youtube.com/watch?v=1zPFuEvSFKk>

29:08



Singhal heuristic algorithm

A heuristically-aided algorithm to achieve mutual exclusion in distributed systems is presented which has better performance characteristics than previously proposed algorithms. The algorithm makes use of state information, which is defined as the set of states of mutual exclusion processes in the system. Each site maintains information about the state of other sites and uses it to deduce a subset of sites likely to have the token. Consequently, the number of messages exchanged for a critical section invocation is a random variable between 0 and n (n is the number of sites in the system). It is shown that the algorithm achieves mutual exclusion and is free from deadlock and starvation.

Instead of Broadcast, each site maintains information about the state of next site.

State of Site:

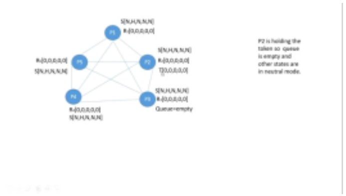
R:Requesting the CS

E:Executing the CS

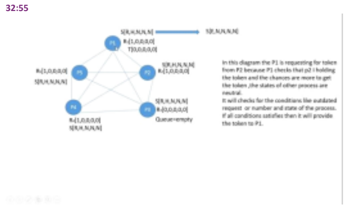
H:Holding the CS , Idle

N:Neutral.None of the above.

29:53



32:55



From <https://www.youtube.com/watch?v=txEK36SF184>

11:11

From <https://www.youtube.com/watch?v=t5O9-nJ5PMM>