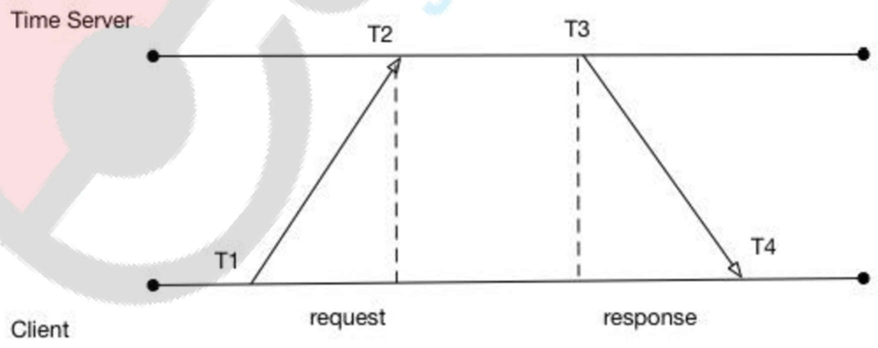
# Unit 2

***Clock synchronization***

Introduction of clock synchronization, global state mutual Exclusion algorithms, election algorithms.

## 2.1) INTRODUCTION OF CLOCK SYNCHRONIZATION

- Clock synchronization deals with understanding the temporal ordering of events produced by concurrent processes.
- It is useful for synchronization between senders and receivers of messages, control of joint activity, and the serialization of concurrent access to shared objects.
- For these kinds of events, we introduce the concept of a logical clock, where one clock need not have any bearing on the time of day but should be useful for generating message sequence numbers. It is also useful for certain applications to have access to an accurate physical clock, one that attempts to provide an accurate measure of the current time.
- The common approach to time synchronization has been to have many computers make use of a time server.
- Typically, the time server is equipped with special hardware that provides a more accurate time than does a cheaper computer timer
- The challenge with this approach is that there is a delay in the transmission from the time server to the client receiving the time update.
- This delay is not constant for all requests. Some request may be faster and others slower.
- So how do we solve this problem?



- The relative time correction C can be calculated as:
  C=(T2−T1)+(T3−T4)2C=(T2−T1)+(T3−T4)2

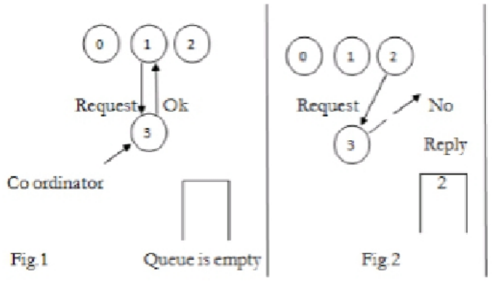
- The way this works is that the client sends a packet with T1T1 recorded to the time server. The time server will record the receipt time of the packet T2T2. When the response is sent, the time server will write its current time T3T3 to the response. When the client receives the response packet, it will record T4T4 from its local clock.
- When the value of C is worked out, the client can correct its local clock
- The client must be careful. If the value of C is positive, then C can be added to the software clock
- If the value of C is negative, then the client must artificially decrease the amount of milliseconds added to its software clock each tick until the offset is cleared.
- It is always inadvisable to cause the clock to go backwards. Most software that relies on time will not react well to this.

## 2.2) GLOBAL STATE MUTUAL EXCLUSION ALGORITMS

**Mutual Exclusion in Distributed System:**

- Mutual Exclusion ensures that no other process will use shared resources at same time.
- Centralized Algorithm
- Distributed Algorithm
- Token Ring Algorithm.
- One process is elected as coordinator.
- Whenever process wants to enter a critical region, it sends request messageai to coordinator asking for permission.
- If no other process is currently in that critical region, the coordinator sends back a reply granting permission.
- When reply arrives, the requesting process enters the critical region.
- If the coordinator knows that a different process is already in critical regions, so it cannot be granted permission.

## Centralized Algorithm:



**Advantages:**

- Guarantees mutual exclusion.
- Fair Approach (Request Granted in FCFS).
- No Starvation.
    - Easy to Implement.
    - Only 3 Messages per use of Critical Section (request, grant, release).

**Drawbacks:**

- Single point of failure.
- Dead co-ordinate & permission denied cannot distinguish.
- In large systems, single coordinators can create performance bottleneck.

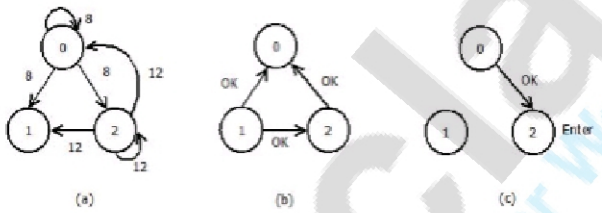
## Distributed Algorithm:

- Timestamps are used for distributed mutual exclusion.
- Kieart & Agarwala's Algorithm:
    - When process wants to enter critical region, it builds message containing name of critical region its process number and current time
    - It sends message to all including itself.
    - If receiver is not in critical region and doesn't want to enter it sends back Ok message to sender.
    - If the receiver is already in critical region, it doesn't reply, instead it queues request.
    - If the receiver wants to enter critical region but has not yet done, so it compares the timestamp in the incoming message the lowest one wins.
    - If its own message has lower timestamp, the receiver queues the incoming request and sends nothing.
- **Drawbacks:**

- If any process fails (Crashes), then it does not respond to request.
- It can be misinterpreted as denial of permission thus may cause blocking of all processes.

- **RPCART & AGARWALA'S SOLUTION:**
  - Same algorithm with reply always given by receiver either by granting or denying permission.
  - Other problem is either group communication must be used, or each process must maintain group information (who enters, who left, etc).

**Enters**



- Other problems is bottleneck
  - Solution can be given with permission from simple majority of processes rather than from all with conditions that a process granted permission to one process cant grant other permission until first has released.
  - Other problems
    - Slower
    - Complicated
    - More expensive.
- **TOKEN RING ALGORITHM**
  - **Mutual Exclusion – Token Ring Algorithm**
    - In software, a logical ring is constructed in which each process is assigned a position in the ring.
    - The ring positions may be allocated in numerical order of network address or some other means.
    - It does not matter what the ordering is all that matters is that each process knows who is next in line after itself.

- **Working of Token Ring Algorithm:**
  - When the ring is initialized, process 0 is given a token.
  - The token circulates around the ring.

- It passes from process K to process Kt (module the ring size) in point to point messages.
- When a process acquires the token from its neighbour, it checks to see if it is attempting to enter a critical region.
- If so, the process enters the region, does all the work it needs to, and leaves the region.
- After it excited, it passes the token along the ring.
- It is not permitted to enter the second critical region with using the same token.
- If a process is handed the token by its neighbour & is not interested in entering a critical region, it just passes it along.
- As a consequence, when no processes want to enter any critical regions the token just circulates at high speed around the ring.
- Only one process has the token at any instant, so only one process can actually be in the critical region.

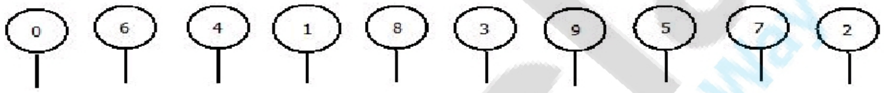


Fig. An Unordered group of process in network

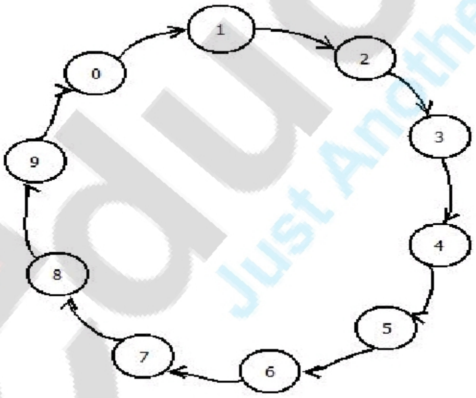


Figure: A Logical ring Constructed in S/W

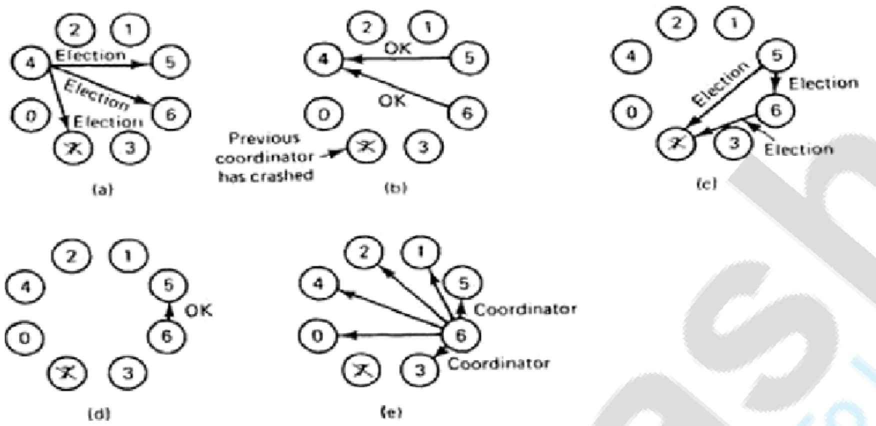
## 2.3) ELECTION ALGORITHM

- Distributed algorithms require one process to act as a coordinator or initiator. To decide which process becomes the coordinator algorithms are used.
- Election algorithms are meant for electing a coordinator process from among the currently running processes in such a manner that at any instance of time there is a single coordinator for all processes in the system.
- The goal of an election algorithm is to ensure that when an election starts it concludes with all the processes agreeing on who the coordinator should be.
- Therefore, whenever initiated, an election algorithm basically finds out which of the currently active processes has the highest priority number and then informs this to all other active processes.

### i.Bully Algorithm

- This algorithm was proposed by Garcia-Molina.
- When the process notices that the coordinator is no longer responding to requests, it initiates an election. A process, P, holds an election as follows:

1. P sends an ELECTION message to all processes with higher numbers.
2. If no one responds, P wins the election and becomes the coordinator.
3. If one of the higher-ups answers, it takes over. P's job is done.
    - A process can get an ELECTION message at any time from one of its lower numbered colleagues.
    - When such a message arrives, the receiver sends an OK message back to the sender to indicate that he is alive and will take over. The receiver then holds an election, unless it is already holding one.
    - All processes give up except one that is the new coordinator. It announces its victory by sending all processes a message telling them that starting immediately it is the new coordinator.
    - If a process that was previously down comes back up, it holds an election. If it happens to the highest numbered process currently running, it will win the election and take over the coordinator's job. Thus the biggest guy in town always wins, hence the name "bully algorithm".
    - Example:



- In fig(a) a group of eight processes taken is numbered from 0 to 7. Assume that previously process 7 was the coordinator, but it has just crashed. Process 4 notices if first and sends ELECTION messages to all the processes higher than it that is 5, 6 and 7.
- In fig (b) processes 5 and 6 both respond with OK. Upon getting the first of these responses, process4job is over. It knows that one of these will become the coordinator. It just sits back and waits for the winner.
- In fig(c), both 5 and 6 hold elections by each sending messages to those processes higher than itself.
- In fig(d), process 6 tells 5 that it will take over with an OK message. At this point 6knows that 7 is dead and that (6) it is the winner. It there is state information to be collected from disk or elsewhere to pick up where the old coordinator left off, 6 must now do what is needed. When it is ready to take over, 6 announce this by sending a COORDINATOR message to all running processes. When 4 gets this message, it can now continue with the operation it was trying to do when it discovered that 7 was dead, but using 6 as the coordinator this time. In this way the failure of is handled and the work can continue.
- If process 7 is ever restarted, it will just send all the others a COORDINATOR message and bully them into submission.

## ii. Ring Algorithm

- This algorithm uses a ring for its election but does not use any token. In this algorithm it is assumed that the processes are physically or logically ordered so each processor knows its successor.
- When any process notices that a coordinator is not functioning, it builds an ELECTION message containing its own process number and sends the message to its successor. If the successor is down the sender skips over

the successor and goes to the next member along the ring until a process is located.

- At each step the sender adds its own process number to the list in the message making itself a candidate to elected as coordinator
- The message gets back to the process that started it and recognizes this event as the message consists its own process number.
- At that point the message type is changed to COORDINATOR and circulated once again to inform everyone who the coordinator is and who are the new members. The coordinator is selected with the process having highest number.
- When this message is circulated once it is removed and normal work is preceded.