Distributed Operating System (2160710)

Unit: 3

Synchronization in Distributed Systems

Dr. K. Jairam Naik

NIT Raipur

Unit Outline & Weightage %

- **Synchronization in Distributed Systems** **10%**
  - Clock synchronization
  - Clock Synchronization Algorithms
    - Centralized Algorithms
    - Distributed Algorithms
  - Mutual Exclusion
  - Deadlock
  - Election Algorithms

Synchronization

- Synchronization is coordination with respect to time, and refers to the **ordering of events** and **execution of instructions in time.**

“Ordering of all actions”

- Synchronizing processes in a distributed environment is difficult because:
  - Relevant information can be anywhere on the system
  - Processes can make decisions based only on local information
  - Want to avoid having a centralized coordinator
  - No shared system clock

Clock Synchronization

- It is often important to know **when events occurred** and in **what order they occurred.**
- In a non-distributed system dealing with time is trivial as there is a **single shared clock**, where all processes see the same time.
- In a distributed system, on the other hand, each computer has its **own clock**.
- Because no clock is perfect each of these clocks has its own skew which causes clocks on different computers to drift and eventually become out of sync.

Drifting of Clock

- Due to differences in crystals ,two clock rates different from each other.
- Difference is small but observable, no matter how accurately initialized.
- Therefore, Passage of time, a computer clock drifts from the real-time clock. (After 11.6 days of 1 second)
- Hence, Computer clock must be periodically synchronized with the real-time clock called non-faulty.
- Even non-faulty clocks do not always maintain perfect time.
- A clock is called non-faulty if there is bound on the amount of drift.
- **Clock drift:** The computer clock differs from the real time.
- **Clock Skew:** Difference between two clocks at one point in time.

Drifting of Clock

Sept 18, 2016
8:00:00

Drifting of Clock

8:01:24

Skew = +84 seconds

+84 seconds/35 days

Drift = +2.4 sec/day

Oct 23, 2016

8:00:00

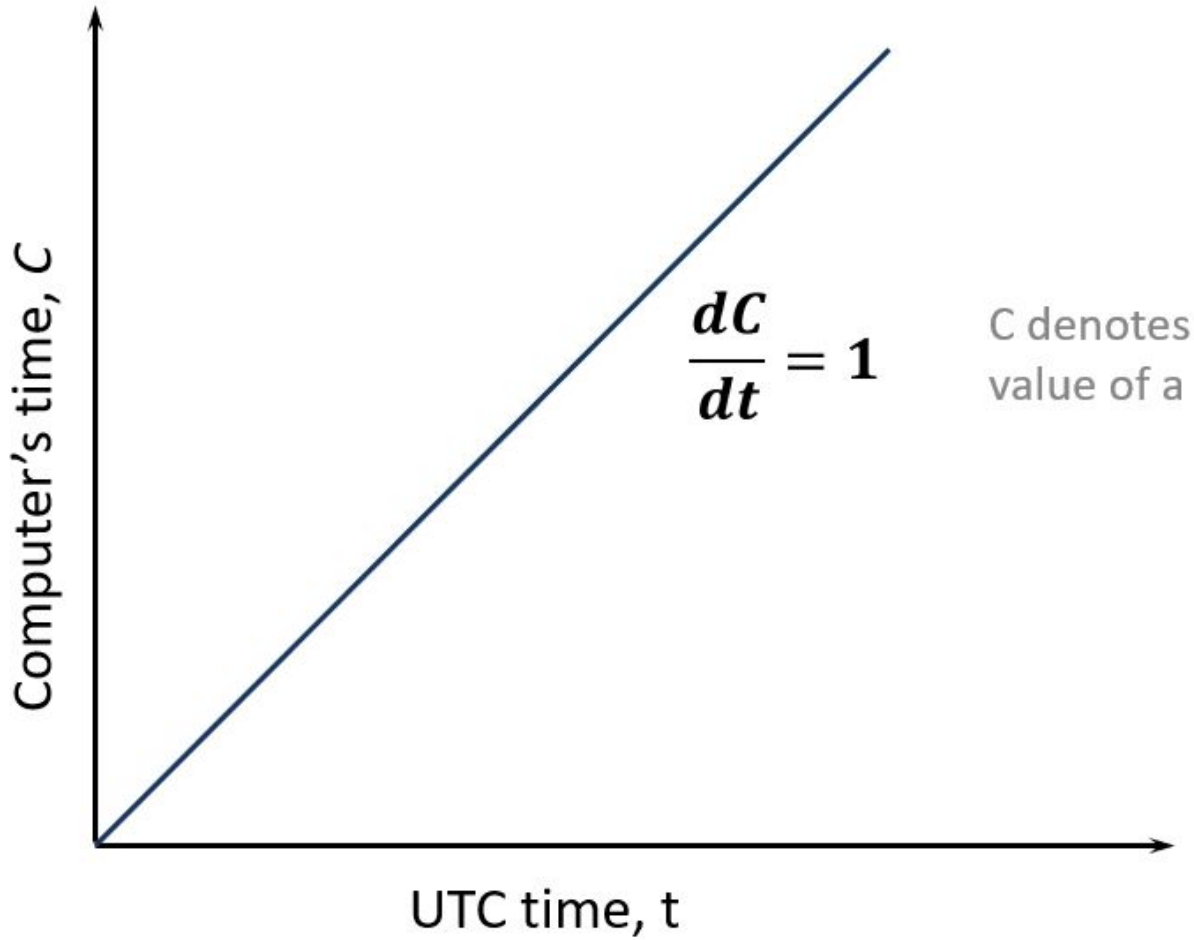
8:01:48

Skew = +108 seconds

+108 seconds/35 days

Drift = +3.1 sec/day

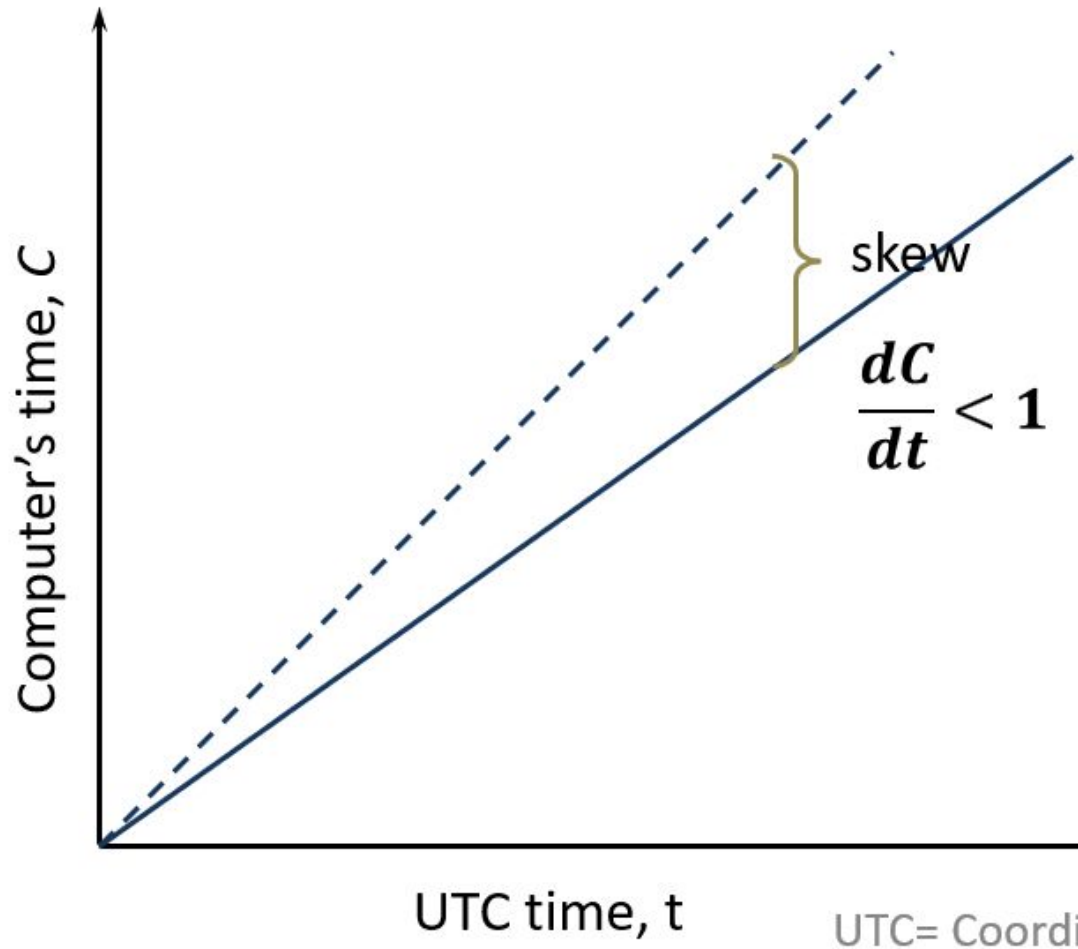
Drifting – Perfect Clock



C denotes the time value of a clock

UTC= Coordinated Universal Time

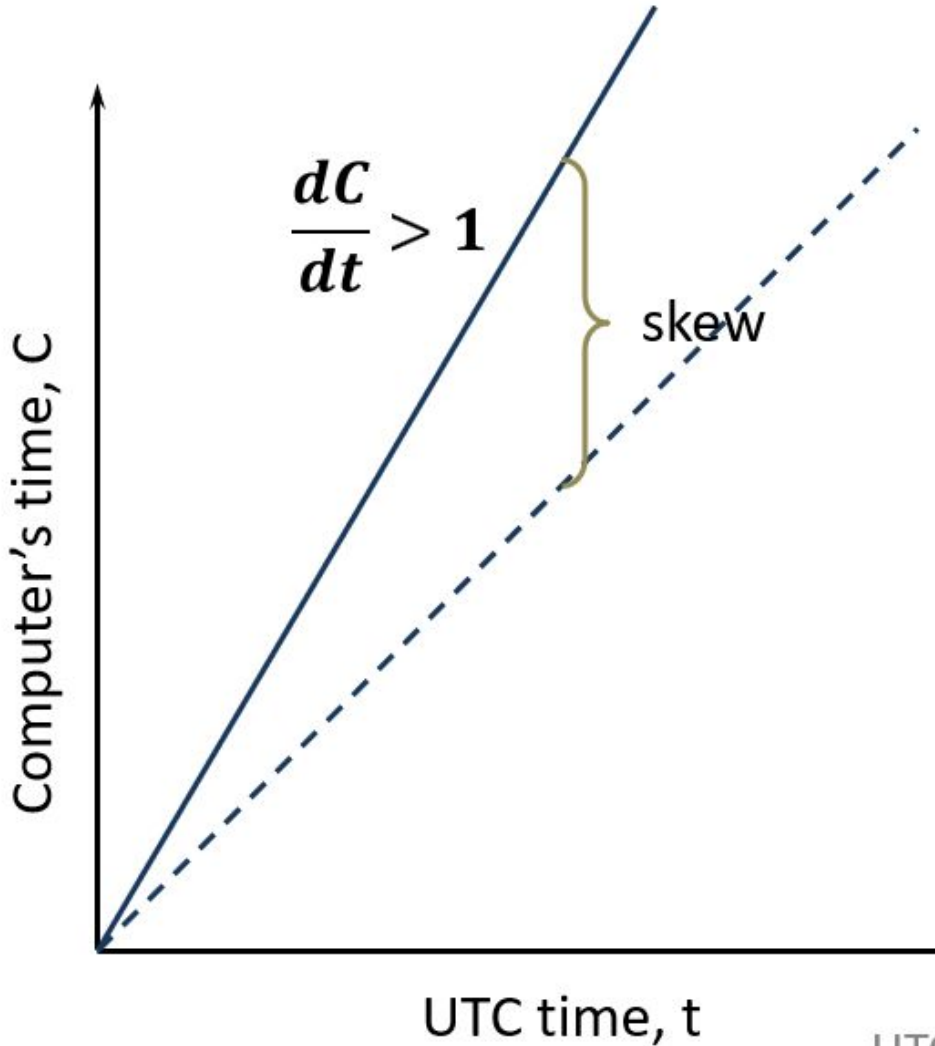
Drifting - Drift with slow clock



C denotes the time value of a clock

UTC= Coordinated Universal Time

Drifting - Drift with Fast clock



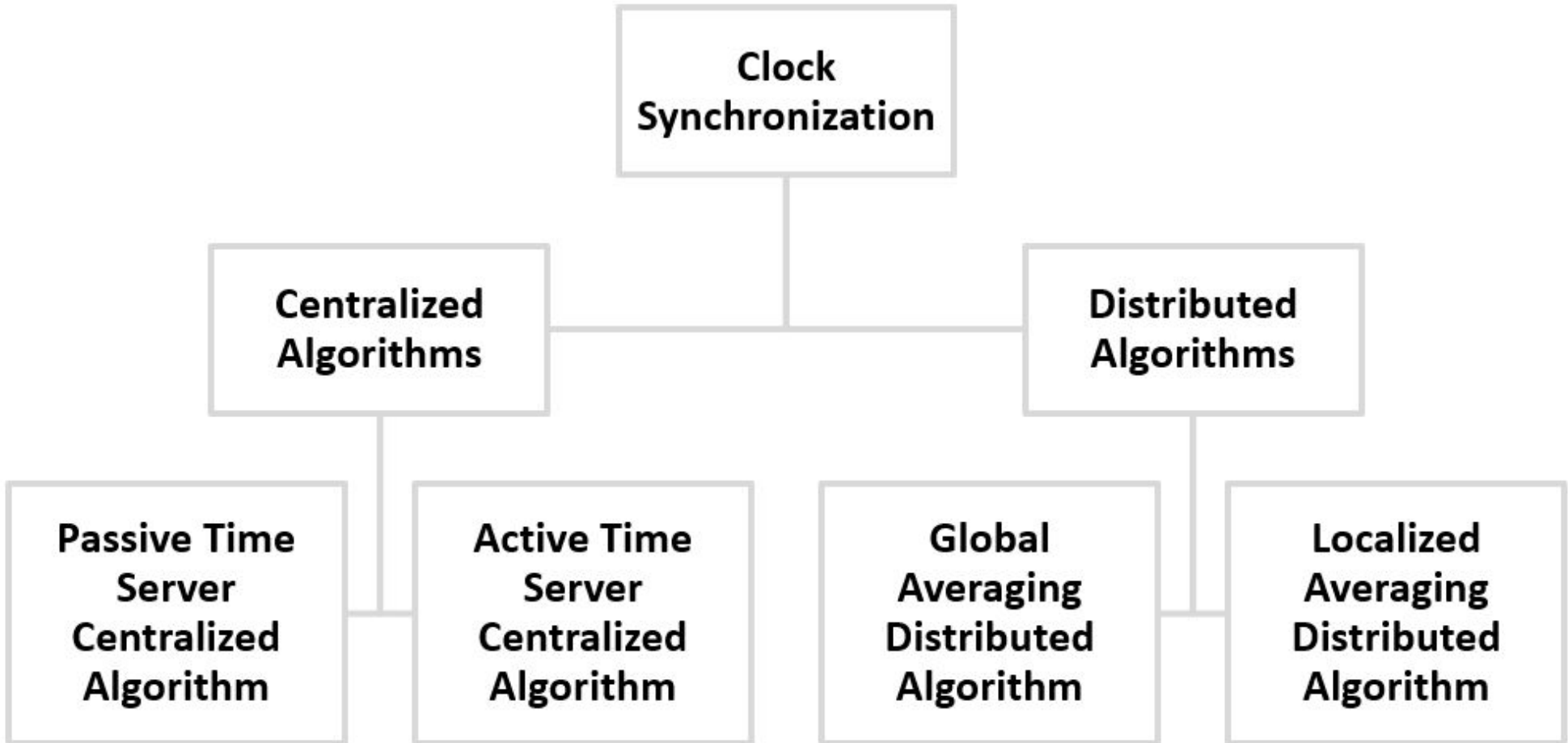
C denotes the time value of a clock

UTC= Coordinated Universal Time

Dealing with Drift

- Assume we set computer to true time. (It is not good idea to set clock back.)
  - Illusion of time moving backwards can confuse message ordering and software development environments.
- There should be go for gradual clock correction.
  - **If fast:** Make clock run slower until it synchronizes.
  - **If slow:** Make clock run faster until it synchronizes.
- Operating System can change rate at which it requests interrupts.
  - if system requests interrupts every 14 msec but clock is too slow: request interrupts at 12 msec.
- **Software correction:** Redefine the interval.

Clock Synchronization Algorithms



Centralized Algorithms

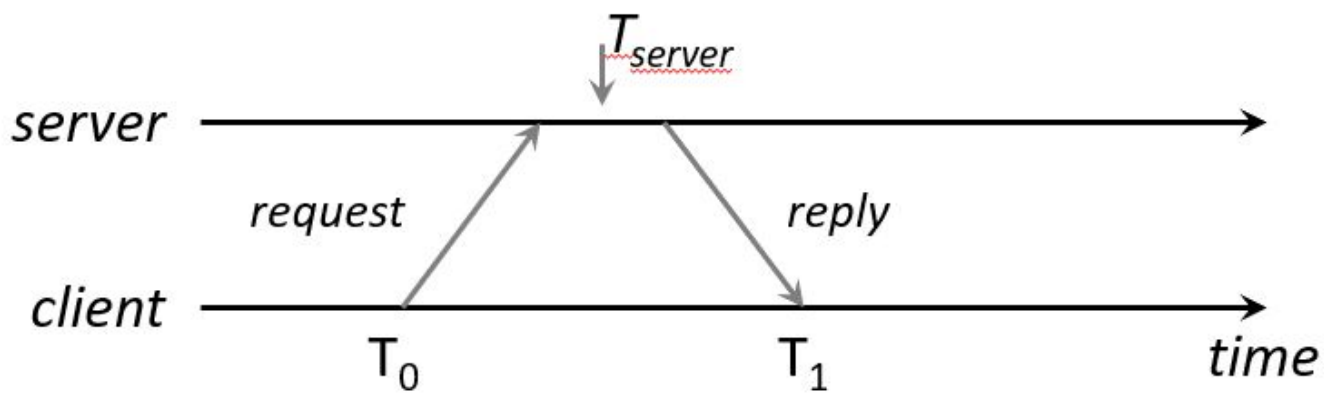
- One node has a real-time receiver called time server node.
- Clock time this node used to correct the time of all other nodes.
- The goal of this algorithm is to keep the clock of all other nodes synchronized with the clock time of the time server node.
- Depending the role of the server node, centralized algorithms are again of two types,
  - Passive time server
  - Active time server

Passive Time Server Algorithm

- Each node periodically sends a message (“ Time = ? “) to the time server.
- Server quickly responds with a message (“Server Time = T_{server}”).
- When client node sends the request of time, its clock time is T_0.
- When client node receives the time from server, its clock time is T_1.
- Propagation time of the message from server to client node is estimated to be $(T_1\text{-}T_0)/2$.
- Therefore, Clock of client node is readjusted to $T_{server}+(T_1\text{-}T_0)/2$.
- Due to unpredictable propagation time between two nodes , is not very good estimation.

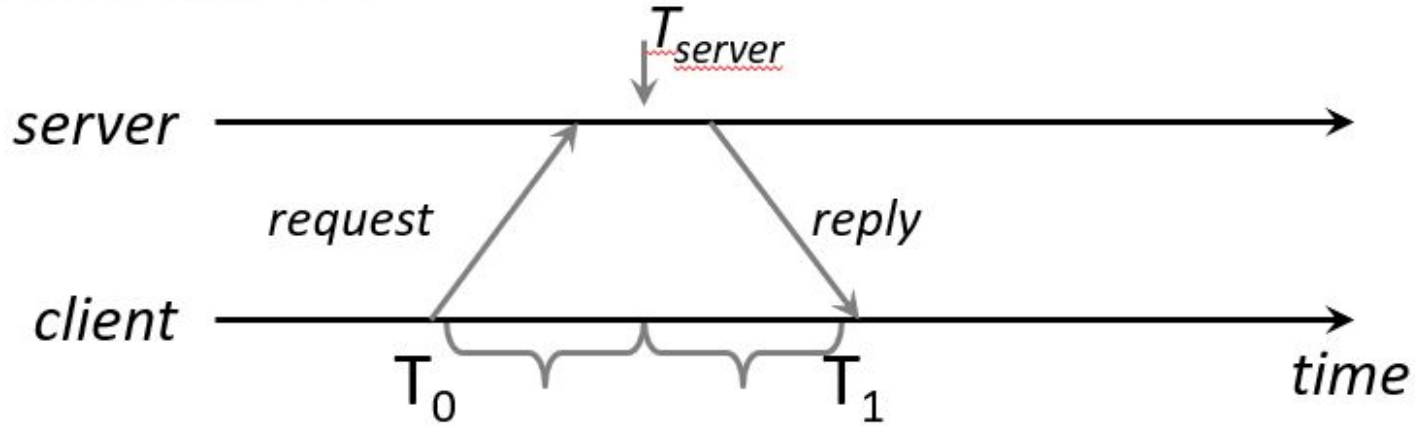
Cristian's algorithm

- Note times:
  - Request sent: T_0
  - Reply received: T_1
- Assume network delays are symmetric.



Cristian's algorithm

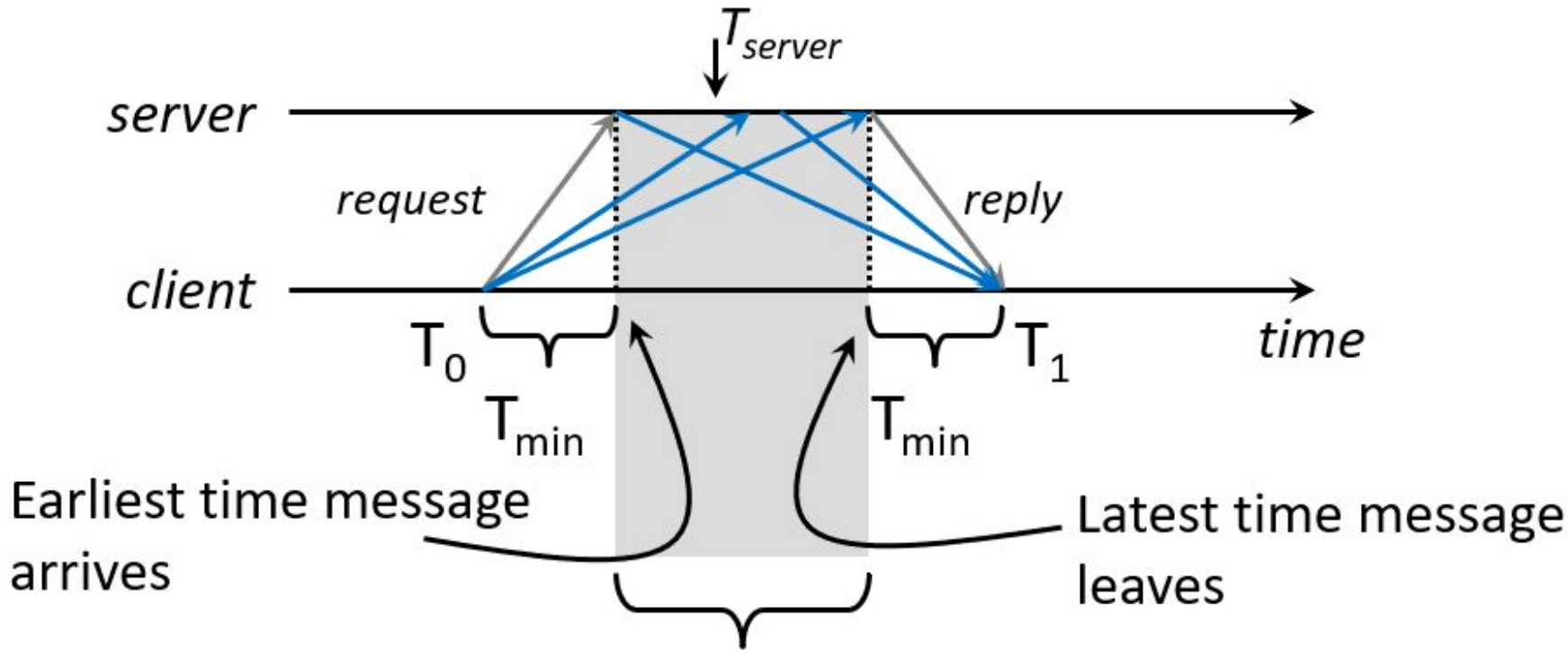
Client sets time to:



$\frac{T_1 - T_0}{2}$ = estimated overhead in each direction

$$T_{new} = T_{server} + \frac{T_1 - T_0}{2}$$

Error bounds in Cristian's algorithm



range = T_1-T_0-2T_{min}

Accuracy of Result = $\pm \frac{T_1 - T_0}{2} - T_{min}$

Active Time Server Algorithm

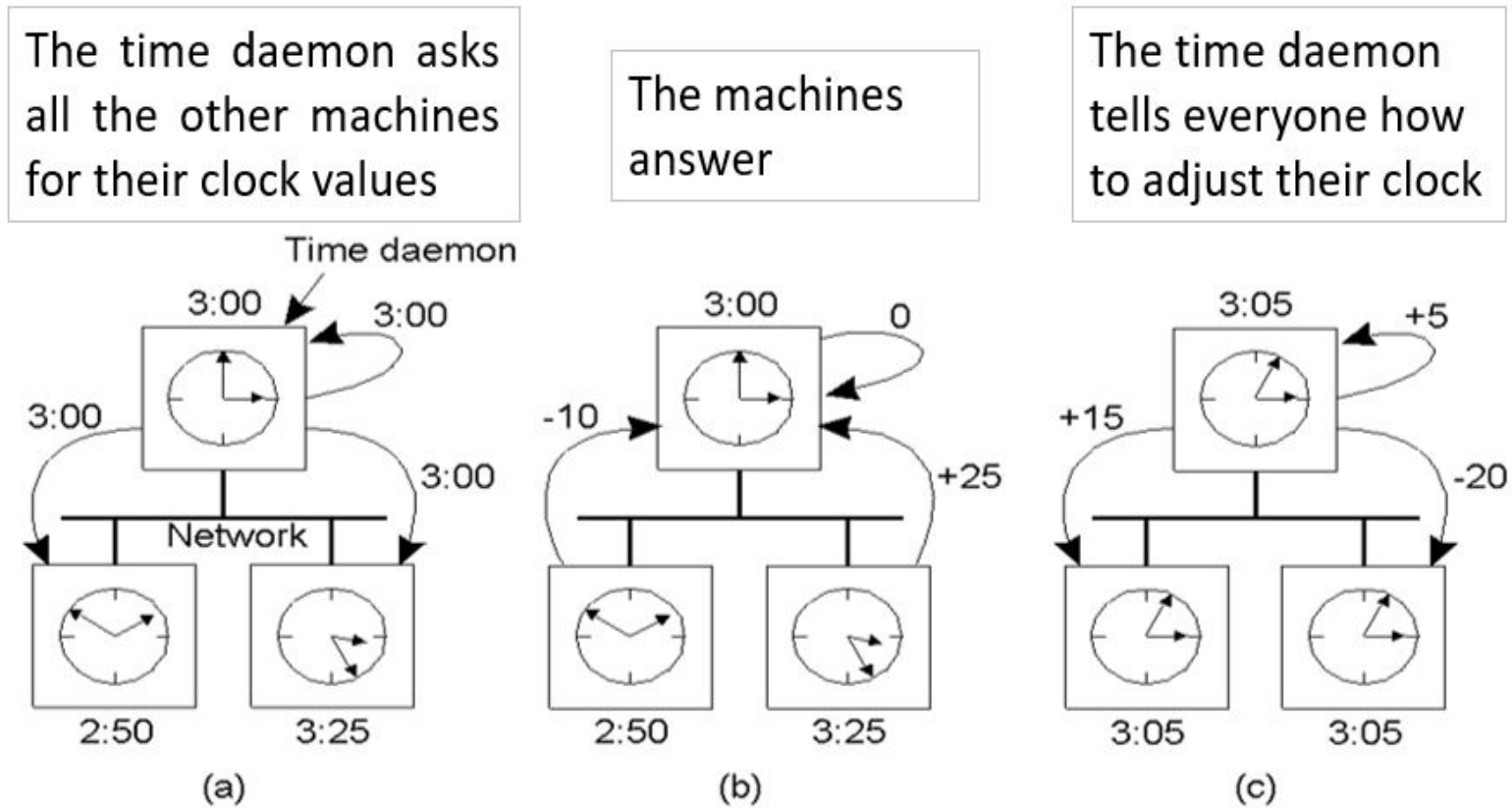
- In this approach, Time server periodically broadcast its clock time (“ Time = T”) .
- Other nodes receive broadcast message and use the clock time in the message for correcting their own clocks.
- Each node has a priori knowledge of the approximate time (Ta).
- Ta is the time from the propagation server to client node.
- Client node is readjusted to T + Ta.
- **Major Faults:**
  - Broadcast message reaches too late at the client node due to some communication fault, Clock of this node readjusted to incorrect value.
  - Broadcast facility should be supported by network.

Active Time Server Algorithm- Solution

- **Berkeley Algorithm**
  - An algorithm for internal synchronization of a group of computers.
  - A master polls to collect clock values from the others (slaves).
  - The master uses round trip times to estimate the slaves' clock values.
  - Obtains average from participating computers.
  - It sends the required adjustment to the slaves.
  - If master fails, can elect a new master to take over.
  - Assumes no machine has an accurate time source.
  - Synchronizes all clocks to average.

Berkeley Algorithm: Example

- A time deamon periodically polls every machine to ask the time.
- Each machine replies.
- Based on the answers, computes an average. Informs every machine to advance or slow down its clock.



Distributed Algorithms

- Externally synchronized clocks are also internally synchronized
- Each node's clock is independently synchronized with real time.
- All clocks of system remain mutually synchronized.
- Each node is equipped with real time receiver so that each node can be independently synchronized.
- Theoretically internal synchronization of clock is not required.
- In practice, Due to inherent inaccuracy of real time clocks, different real time clocks produce different time.
- Internal synchronization is performed for better accuracy.
- Types of internal Synchronization:
  - Global averaging
  - Localized Averaging

Global Averaging Algorithm

- Each node broadcasts its local clock time in the form of a special “resync” message.
- After broadcasting the clock value , the clock process of a node waits for time T (T is calculated by algorithm).
- During this period, clock process collects the resync messages broadcast by other nodes.
- Clock process estimates the skew of its clock with respect to other nodes (when message received) .
- It then computes a fault-tolerant average of the estimated skews and use it to correct the local clock.

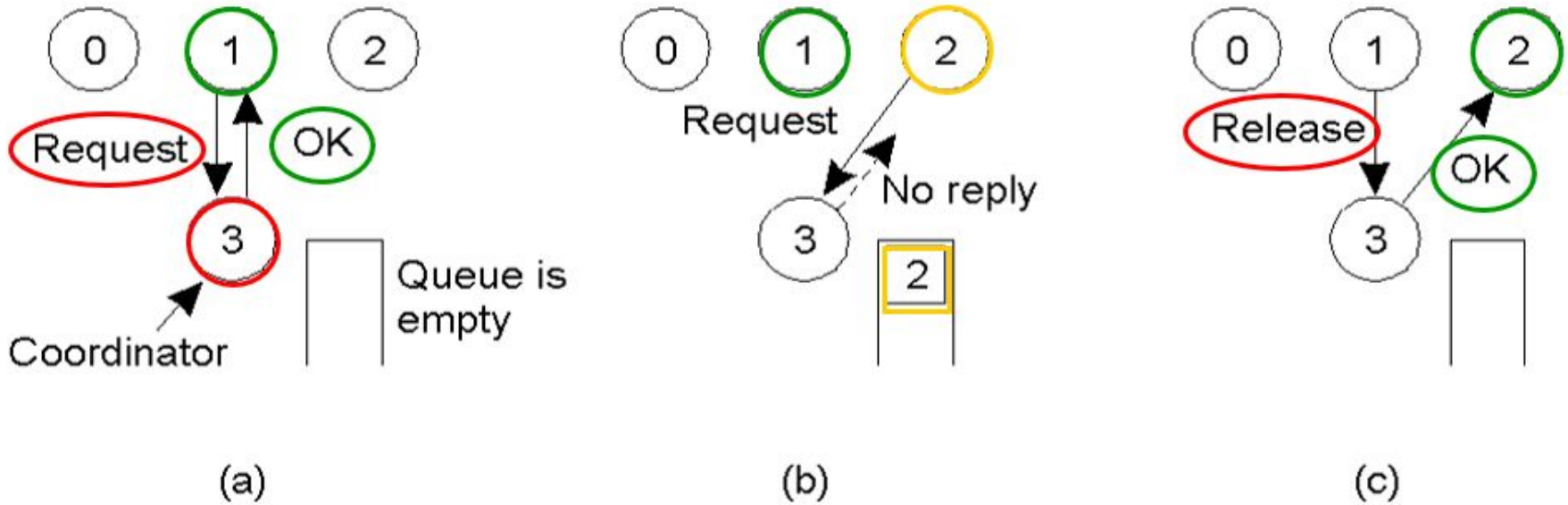
Local Averaging Algorithm

- This algorithm attempt to overcome the drawbacks of the global averaging algorithm.
- Nodes of a distributed system are logically arranged in some kind of pattern such as Ring.
- Periodically each node exchange its clock time with is neighbors in the ring.
- And then sets its clock time by taking Average.
- Average is calculated by , its own clock time and clock times of its neighbors.

Mutual Exclusion

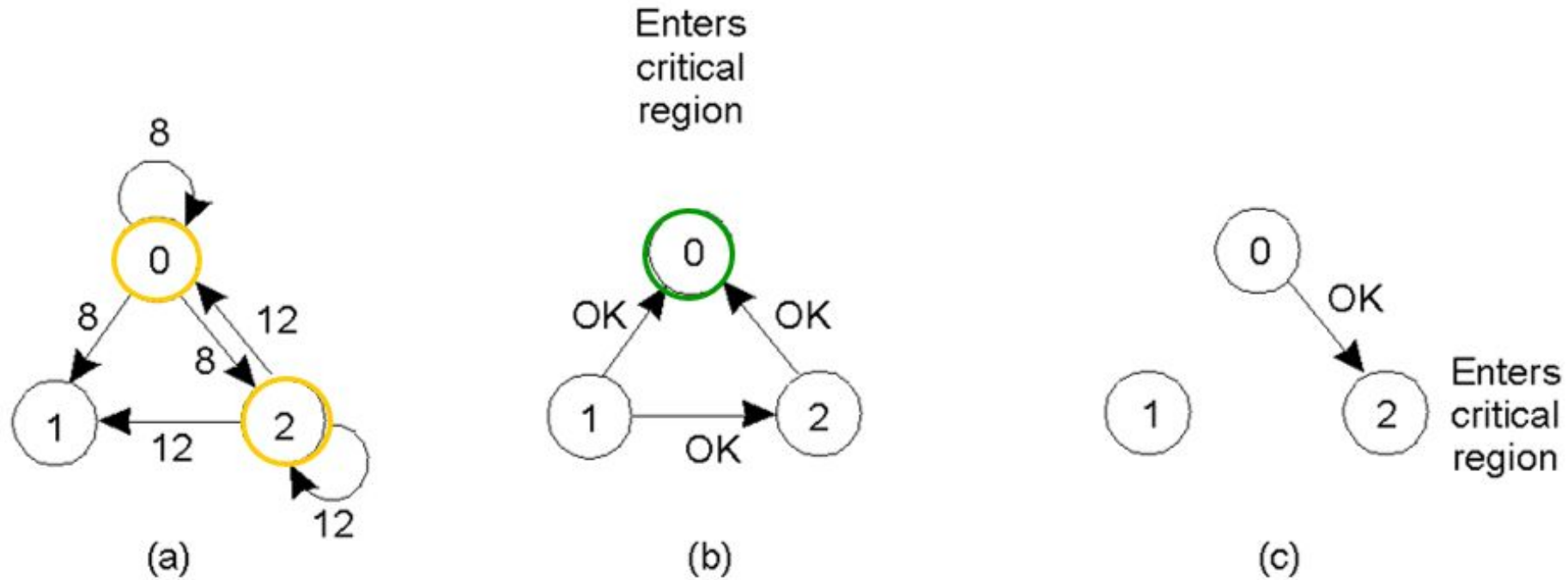
- Mutual Exclusion is a process that prevents multiple threads or processes from accessing shared resources at the same time.
- Concurrent access of processes to a shared resource or data is executed in mutually exclusive manner.
- Only one process is allowed to execute the critical section (CS) at any given time.
- A critical section is a section in a program that accesses shared resources.
- In a distributed system, shared variables or a local kernel cannot be used to implement mutual exclusion.
- Message passing is the sole means for implementing distributed mutual exclusion.

Centralized Mutual Exclusion algorithm



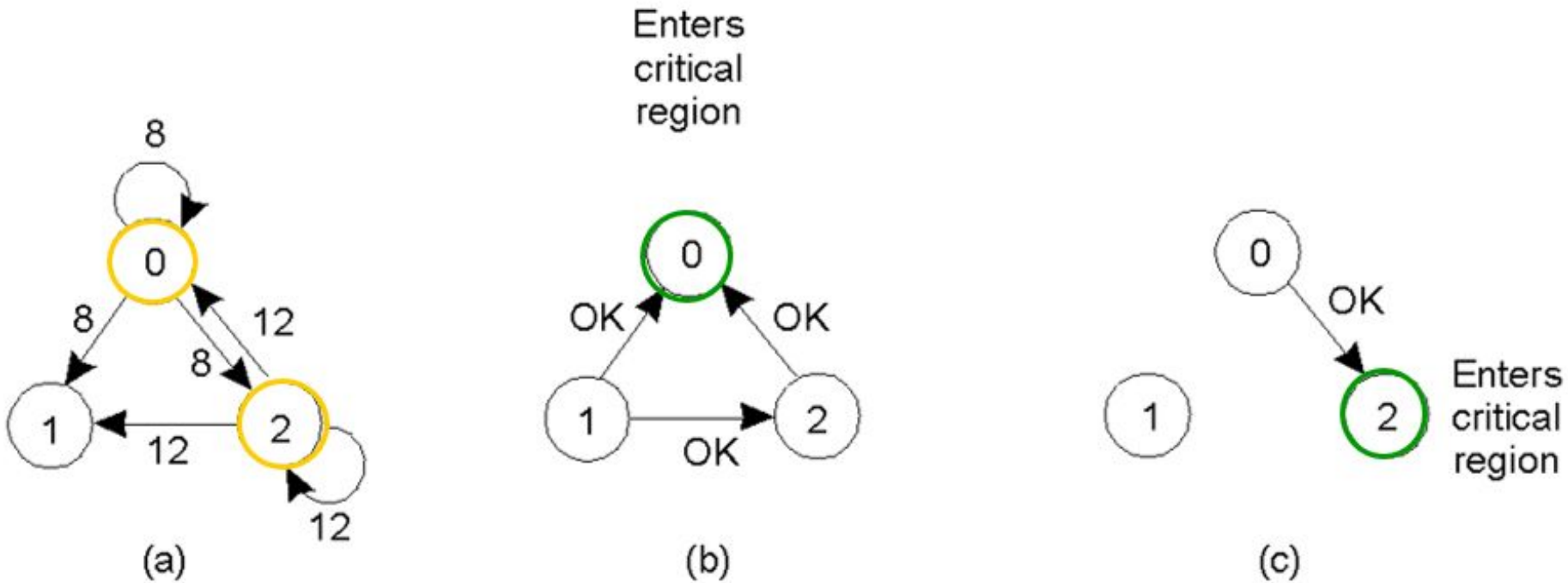
- Process 1 asks the coordinator for permission to enter a critical region. Permission is granted
- Process 2 then asks permission to enter the same critical region. The coordinator does not reply.
- When process 1 exits the critical region, it tells the coordinator, which then replies to 2.

Distributed Mutual Exclusion algorithm



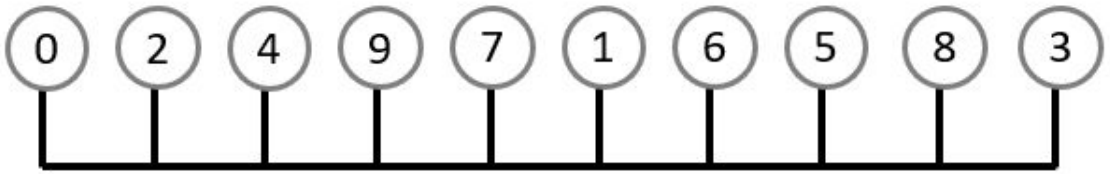
- Process 0 sends everyone a request with timestamp 8.
- while at the same time, process 2 sends everyone a request with timestamp 12.
- Process 1 is not interested in entering the critical region, so it sends OK to both senders.
- Processes 0 and 2 both see the conflict and compare timestamps.

Distributed Mutual Exclusion algorithm(cntd.)

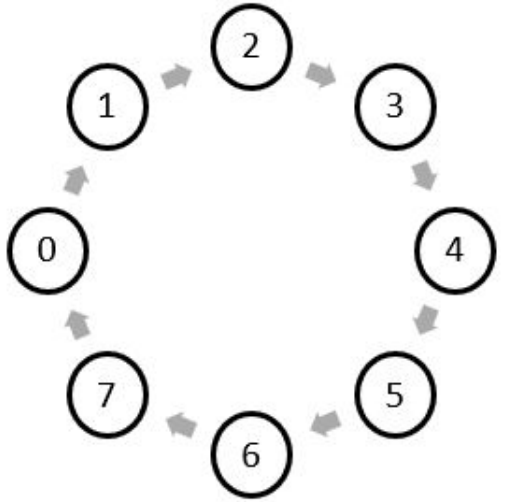


- Process 2 sees that it has lost, so it grants permission to 0 by sending OK.
- Process 0 now queues the request from 2 for later processing and enters the critical region.
- When it is finished, it removes the request from 2 from its queue and sends an OK message to process 2, allowing the latter to enter its critical region.

Token Ring Algorithm



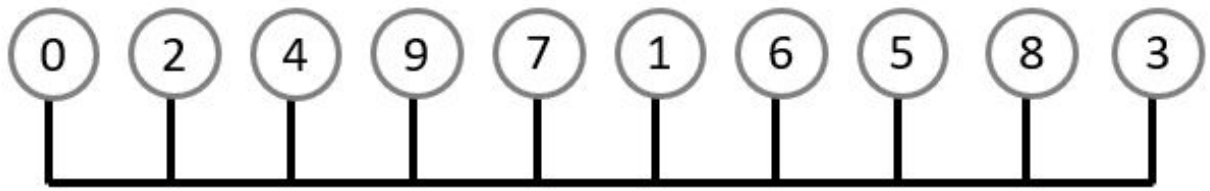
An unordered group of processes on a network.



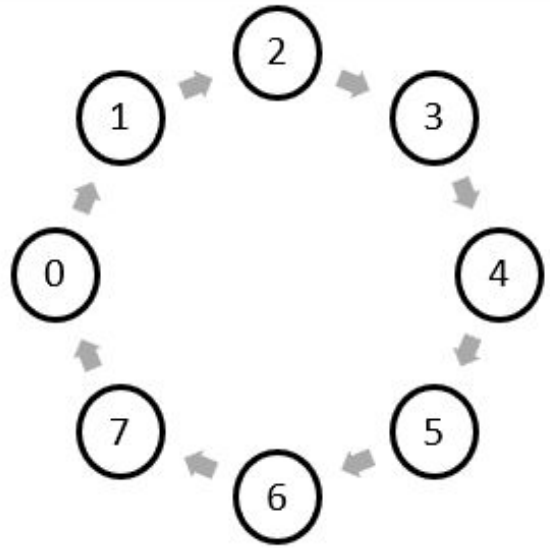
A logical ring constructed in software.

- When the ring is initialized, process 0 is given a token.
- The token circulates around the ring.
- It is passed from process k to process k +1 (modulo the ring size) in point-to-point messages.
- When a process acquires the token from its neighbor, 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.

Token Ring Algorithm (cntd.)



An unordered group of processes on a network.

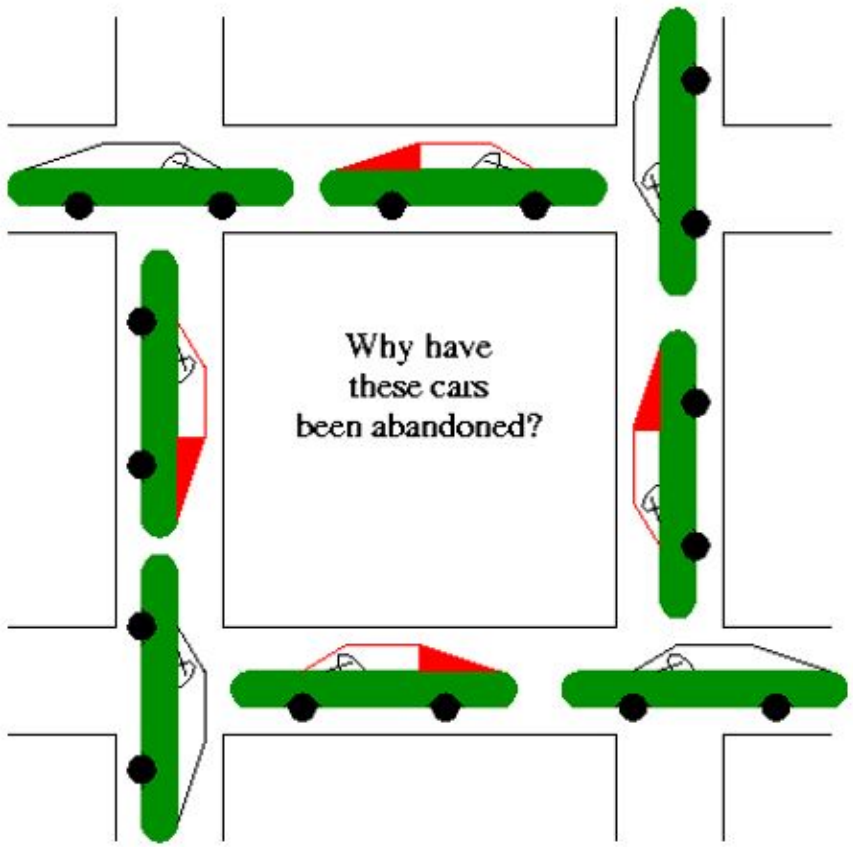


A logical ring constructed in software.

- After it has exited, it passes the token along the ring. It is not permitted to enter a second critical region using the same token.
- If a process is handed the token by its neighbor and 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.

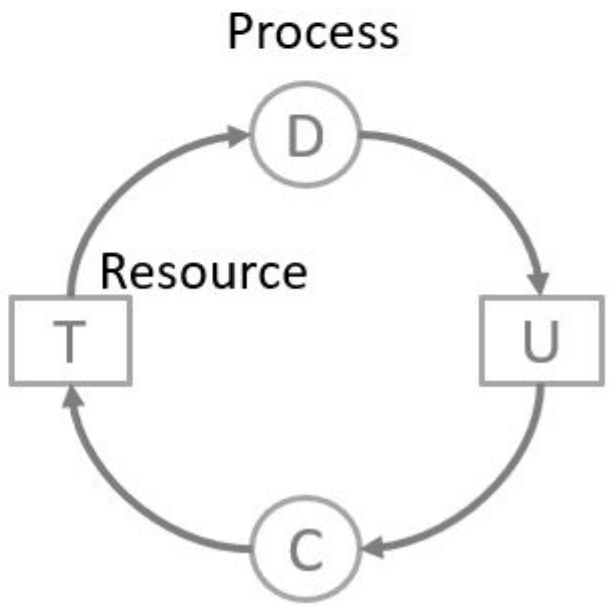
Deadlock

- A deadlock occurs when a every member of a set of processes is waiting for an event that can only be caused by a member of the set.
- Often the event waited for is the release of a resource.
- In the automotive world deadlocks are called gridlocks.
- The processes are the cars.
- The resources are the spaces occupied by the cars



Deadlock

- A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause.
- Suppose process D holds resource T and process C holds resource U.
- Now process D requests resource U and process C requests resource T but none of process will get this resource because these resources are already hold by other process so both can be blocked, with neither one be able to proceed, this situation is called deadlock.
- As shown in figure, resource T assigned to process D and resource U is assigned to process C.
- Process D is requesting / waiting for resource U and process C is requesting / waiting for resource T.
- Processes C and D are in deadlock over resources T and U.



Necessary Conditions for Deadlock

- **Mutual exclusion condition:** If a resource is held by a process, any other process requesting for that resource must wait until the resource has been released.
- **Hold and wait condition:** Processes are allowed to request for new resources without releasing the resources that they are currently holding.
- **No preemption condition:** A resource that has been allocated to a process becomes available for allocation to another process only after it has been voluntarily released by the process holding it.
- **Circular wait condition:** Two or more processes must form a circular chain in which each process is waiting for a resource that is held by the next member of the chain.
- **All four of these conditions must be present for a deadlock to occur.**

Handling Deadlocks In DOS

- Handling of deadlocks in distributed systems is more complex than in centralized systems.
- Because the resources, the processes, and other relevant information are scattered on different nodes of the system.
- Three commonly used strategies to handle deadlocks are as follows:
  - **Avoidance:** Resources are carefully allocated to avoid deadlocks.
  - **Prevention:** Constraints are imposed on the ways in which processes request resources in order to prevent deadlocks.
  - **Detection and recovery:** Deadlocks are allowed to occur and a detection algorithm is used to detect them. After a deadlock is detected, it is resolved by certain means.

Deadlock Recovery

- **Recovery through preemption**
  - In some cases, it may be possible to temporarily take a resource away from its current owner and give it to another process.
  - Recovering this way is frequently difficult or impossible.
  - Choosing the process to suspend depends largely on which ones have resources that can easily be taken back.
- **Recovery through killing processes**
  - The simplest way to break a deadlock is to kill one or more processes.
  - In this approach, the process to be killed is carefully chosen because it is holding resources that some process in the cycle needs.

Deadlock Recovery (cntd.)

- **Recovery through rollback**
  - Checkpoint a process periodically.
  - Checkpointing a process means that its state is written to a file so that it can be restarted later.
  - To do the recovery, a process that owns a needed resource is rolled back to a point in time before it acquired some other resource by starting one of its earlier checkpoints.

Deadlock Prevention

- Deadlock can be prevented by attacking the one of the four conditions that leads to deadlock.
  - Attacking the Mutual Exclusion Condition
  - Attacking the Hold and Wait Condition
  - Attacking the No Preemption Condition
  - Attacking the Circular Wait Condition

Attacking the Mutual Exclusion Condition

- No deadlock if no resource is ever assigned exclusively to a single process.
- Some devices can be spooled such as printer, by spooling printer output; several processes can generate output at the same time.
- Only the printer daemon process uses physical printer.
- Thus deadlock for printer can be eliminated.
- Not all devices can be spooled.
- Principle:
  - Avoid assigning a resource when that is not absolutely necessary.
  - Try to make sure that as few processes as possible actually claim the resource.

Attacking the Hold and Wait Condition

- Require processes to request all their resources before starting execution.
- A process is allowed to run if all resources it needed is available. Otherwise nothing will be allocated and it will just wait.
- Problem with this strategy is that a process may not know required resources at start of run.
- Resource will not be used optimally.
- It also ties up resources other processes could be using.
- Variation: A process must give up all resources before making a new request. Process is then granted all prior resources as well as the new ones only if all required resources are available.
- Problem: what if someone grabs the resources in the meantime how can the processes save its state?

Attacking the No Preemption Condition

- This is not a possible option.
- When a process P0 request some resource R which is held by another process P1 then resource R is forcibly taken away from the process P1 and allocated to P0.
- Consider a process holds the printer, halfway through its job; taking the printer away from this process without having any ill effect is not possible.

Attacking the Circular Wait Condition

- To provide a global numbering of all the resources.
- Processes can request resources whenever they want to, but all requests must be made in numerical order.
- A process need not acquire them all at once.
- Circular wait is prevented if a process holding resource n cannot wait for resource m, if m > n.
- No way to complete a cycle.
- A process may request 1st a CD ROM drive, then tape drive. But it may not request 1st a plotter, then a Tape drive.
- Resource graph can never have cycle.

Election Algorithm

- Several distributed algorithms require a coordinator process in the entire System.
- It performs coordination activity needed for the smooth running of processes.
- Two examples of such coordinator processes encountered here are:
  - The coordinator in the centralized algorithm for mutual exclusion.
  - The central coordinator in the centralized deadlock detection algorithm.
- If the coordinator process fails, a new coordinator process must be elected to take up the job of the failed coordinator.
- Election algorithms are meant for electing a coordinator process from the currently running processes.

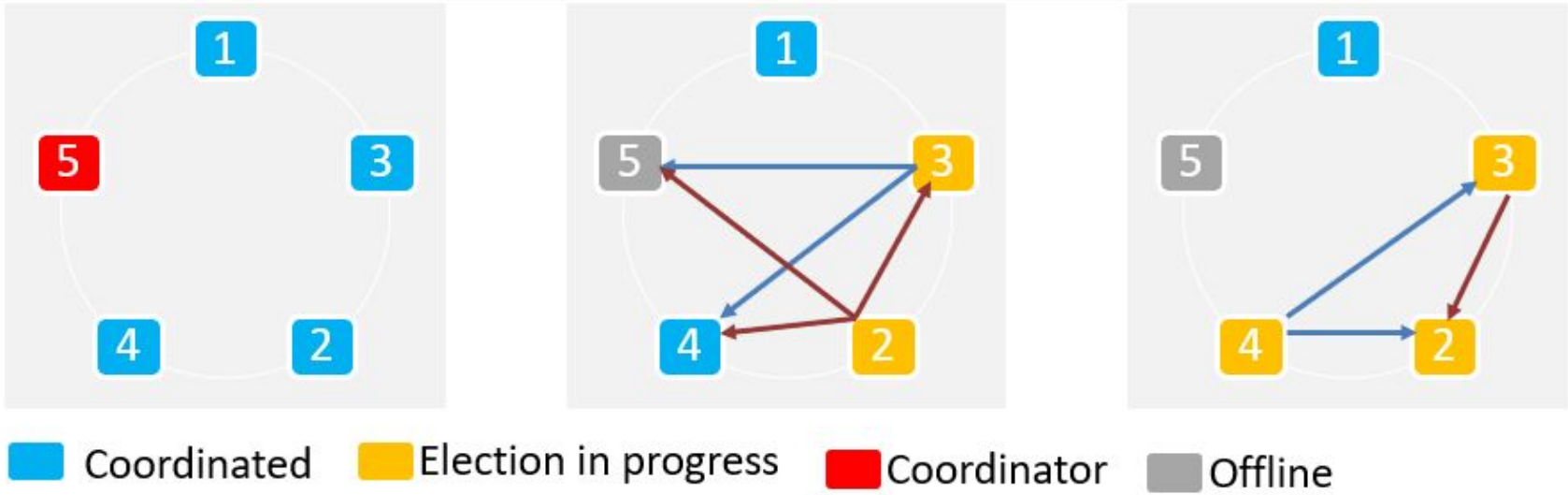
Assumptions: Election Algorithm

- Election algorithms are based on the following assumptions:
  - Each process in the system has a unique priority number.
  - Whenever an election is held, the process having the highest priority number among the currently active processes is elected as the coordinator.
  - On recovery, a failed process can take appropriate actions to rejoin the set of active processes.
- Election Algorithms are:
  - Bully Election Algorithm
  - Ring Election Algorithm

Bully Algorithm

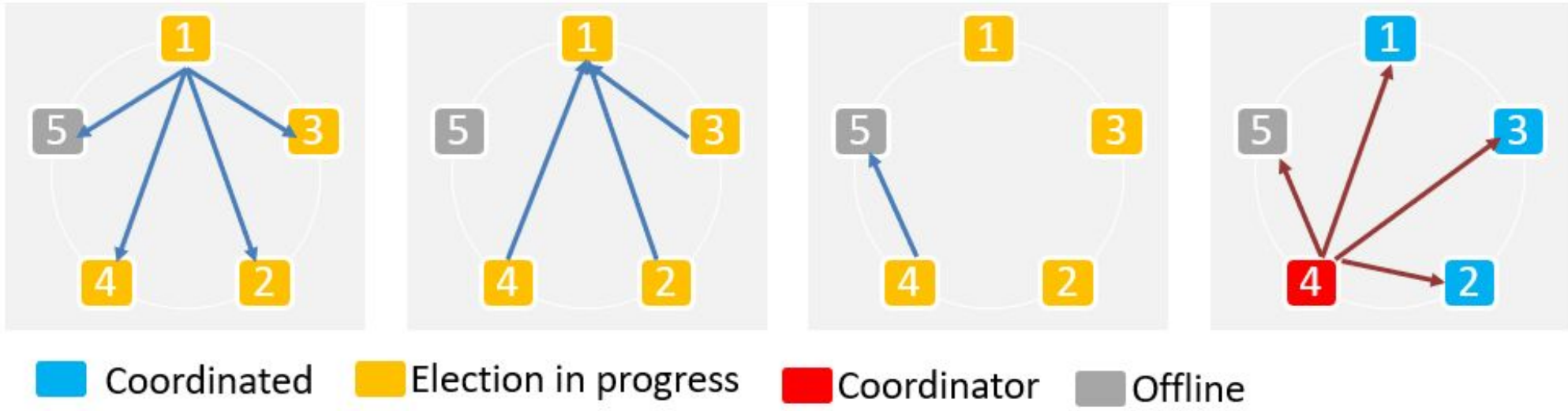
- Bully algorithm specifies the process with the highest identifier will be the coordinator of the group. It works as follows:
- When a process *p* detects that the coordinator is not responding to requests, it initiates an election:
  - *p* sends an election message to all processes with higher numbers.
  - If nobody responds, then p wins and takes over.
  - If one of the processes answers, then p's job is done.
- If a process receives an election message from a lower-numbered process at any time, it:
  - sends an OK message back.
  - holds an election (unless its already holding one).
- A process announces its victory by sending all processes a message telling them that it is the new coordinator.
- If a process that has been down recovers, it holds an election.

Example of Bully Algorithm



- Initially five nodes are in the cluster and node 5 is a globally accepted coordinator.
- Let us assume that node 5 goes down and nodes 3 and 2 detect this simultaneously.
- Both nodes start election procedure and send election messages to the nodes with greater IDs.
- Node 4 kicks out nodes 2 and 3 from the competition by sending OK. Node 3 kicks out node 2.

Example of Bully Algorithm (cntd.)

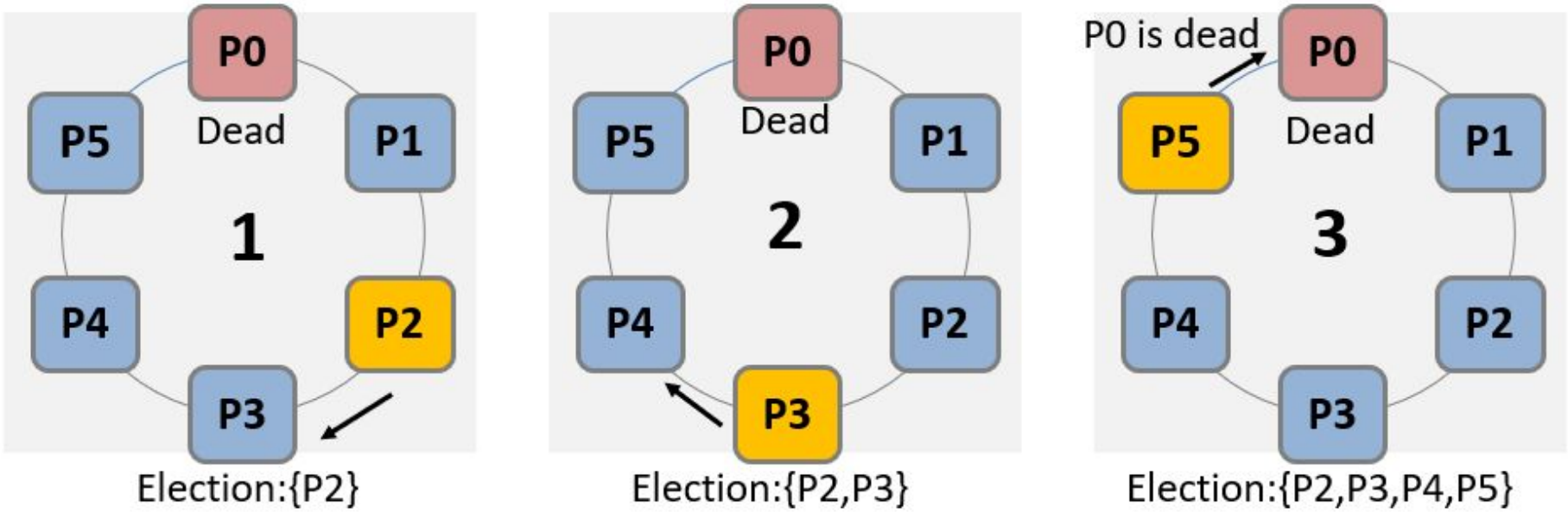


- Imagine that node 1 detects failure of 5 now and an election message to the all nodes with greater IDs.
- Nodes 2, 3, and 4 kick out node 1.
- Node 4 sends an election message to node 5.
- Node 5 does not respond, so node 4 declares itself as a coordinator and announce this fact to all other peers.

Ring Election Algorithm

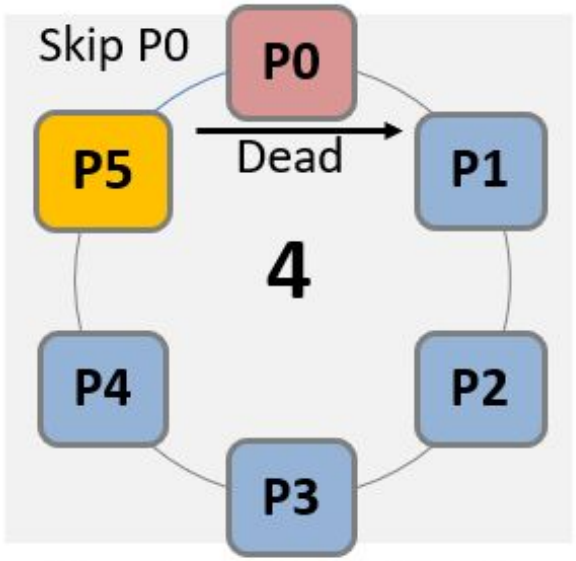
- The ring algorithm assumes that the processes are arranged in a logical ring and each process is knowing the order of the ring of processes.
- If any process detects failure, it constructs an election message with its process ID (e.g., its network address and local process ID) and sends it to its neighbor.
- If the neighbor is down, the process skips over it and sends the message to the next process in the ring.
- This process is repeated until a running process is located.
- At each step, the process adds its own process ID to the list in the message and sends the message to its living neighbor.
- Eventually, the *election* message comes back to the process that started it.
- The process then picks either the highest or lowest process ID in the list and sends out a message to the group informing them of the new coordinator.

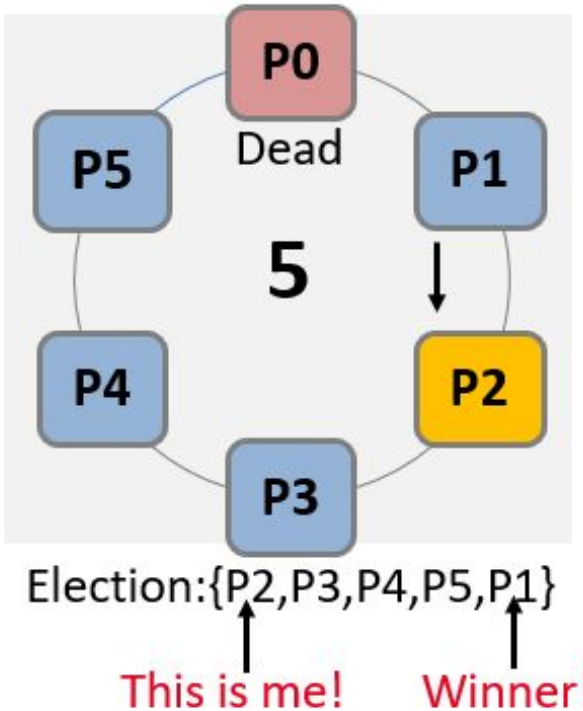
Example of Ring Election Algorithm

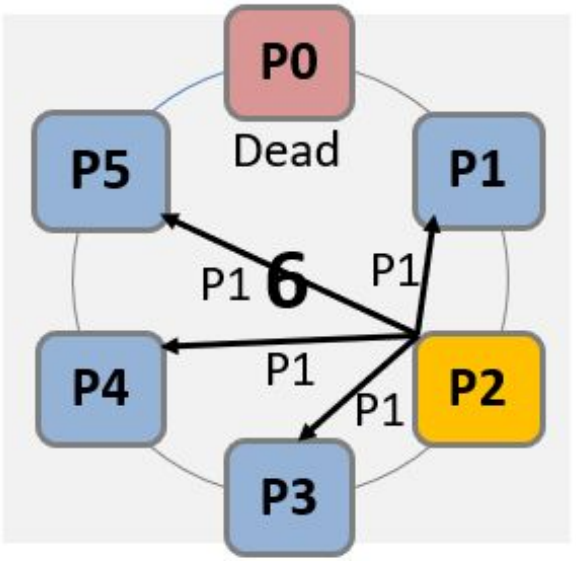


- Fig.1 shows a ring of six processes. P2 detects that the coordinator P0 is dead.
- It starts an election by sending an *election* message containing its process ID to its neighbor P3. (Fig.1)
- P3 receives an *election* message and sends an *election* message to its neighbor with the ID of P3 suffixed to the list. (Fig.2)
- The same sequence of events occurs with P4, which adds its ID to the list it received from P3 and sends an *election* message to P5.
- P5 then tries to send an *election* message to P0. (Fig. 3)

Example of Ring Election Algorithm







- P0 is dead and the message is not delivered, P5 tries again to its neighbor P1. (Fig. 4)
- The message is received by P2, the originator of the election. (Fig. 5)
- P2 recognizes that it is the initiator of the election because its ID is the first in the list of processes.
- It then picks a leader. it chooses the lowest-numbered process ID, which is that of P1.
- It then informs the rest of the group of the new coordinator (Fig. 6).