Mansoura University
Faculty of Computers and Information Sciences
Department of Computer Science
First Semester- 2020-2021

[CS412P] Distributed Systems

Grade : Fourth grade

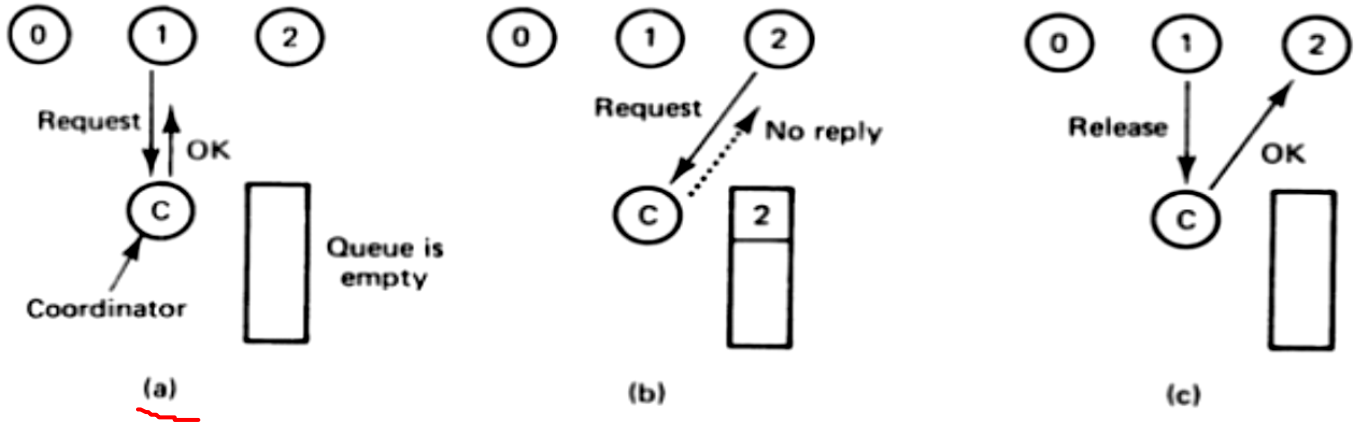
By : Zeinab Awad

REQUIREMENTS FOR DISTRIBUTED MUTUAL EXCLUSION

- Mutual Exclusion: guarantee that only one request access the CR at a time.
- Freedom from deadlock: two or more sites(hosts) should not endlessly wait for msg's that will never arrive.
- Freedom from starvation: a site should not be forced to wait indefinitely to access CR.
- Fairness: requests must be executed in the order they are made. (fairness → freedom of starvation, but not reverse)
- Fault tolerance: in the case of failure, the algorithm can reorganize itself so that it continues to function without any disruption.

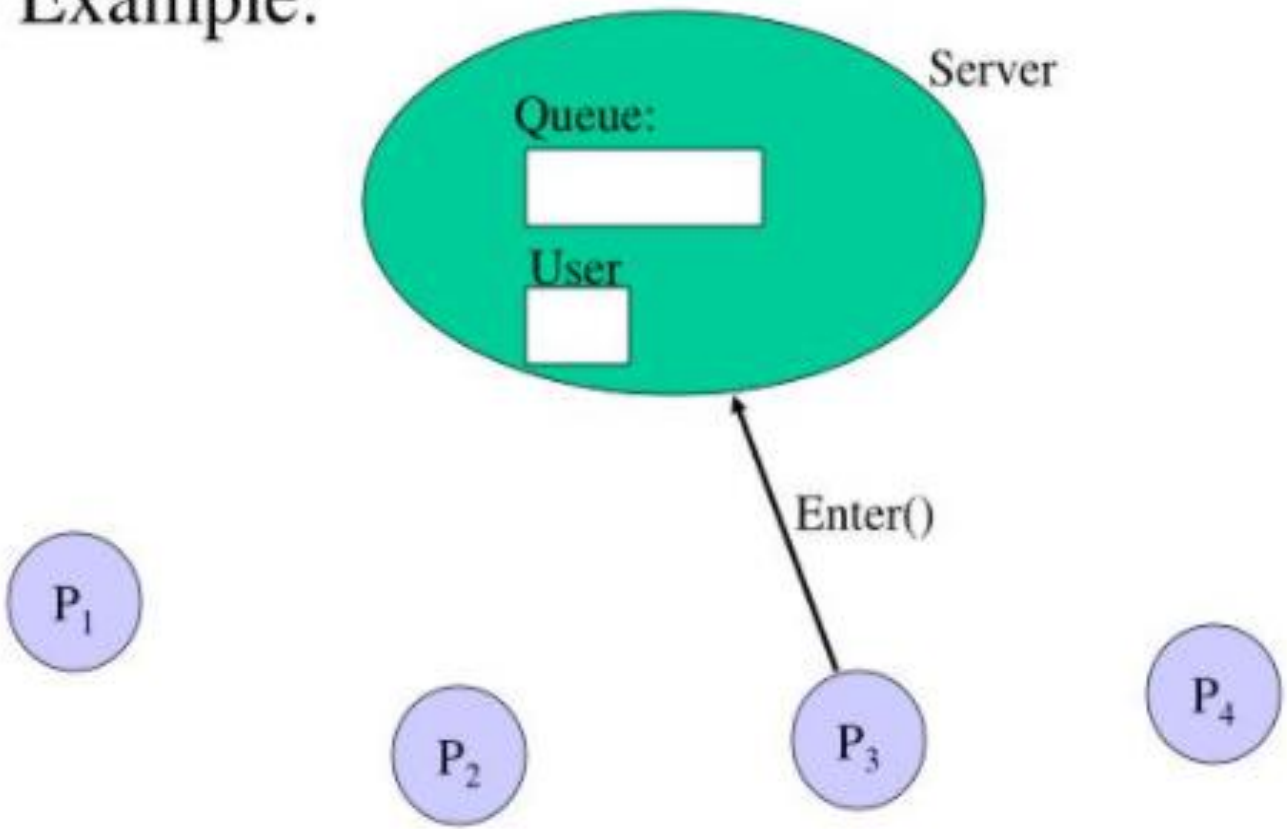
CENTRAL SERVER ALGORITHM

- Use a coordinator which enforces mutual exclusion.
- Two operations: request and release.
  - 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 des not reply.
  - When process 1 exists the critical region, it tells the coordinator, which then replies to 2.

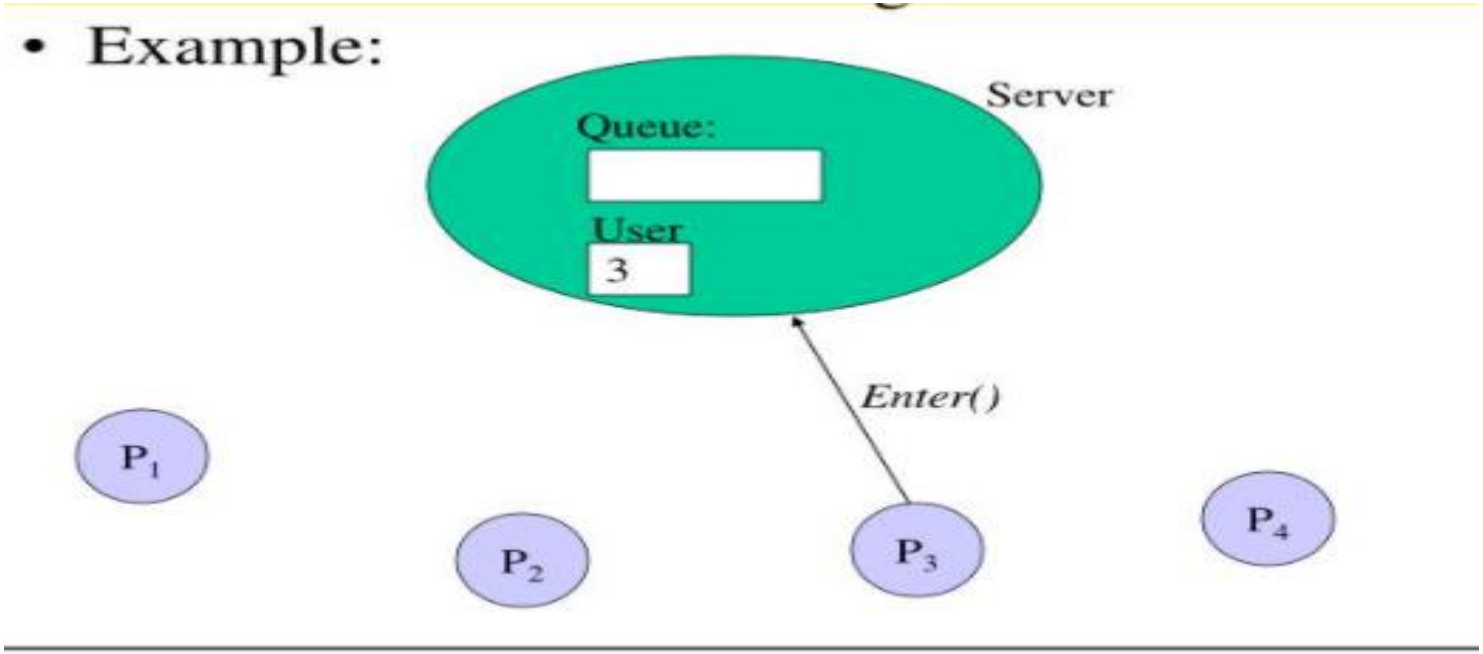


CENTRAL SERVER ALGORITHM

- Example:

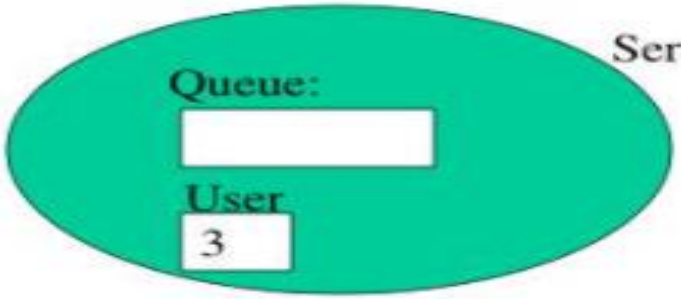


CENTRAL SERVER ALGORITHM



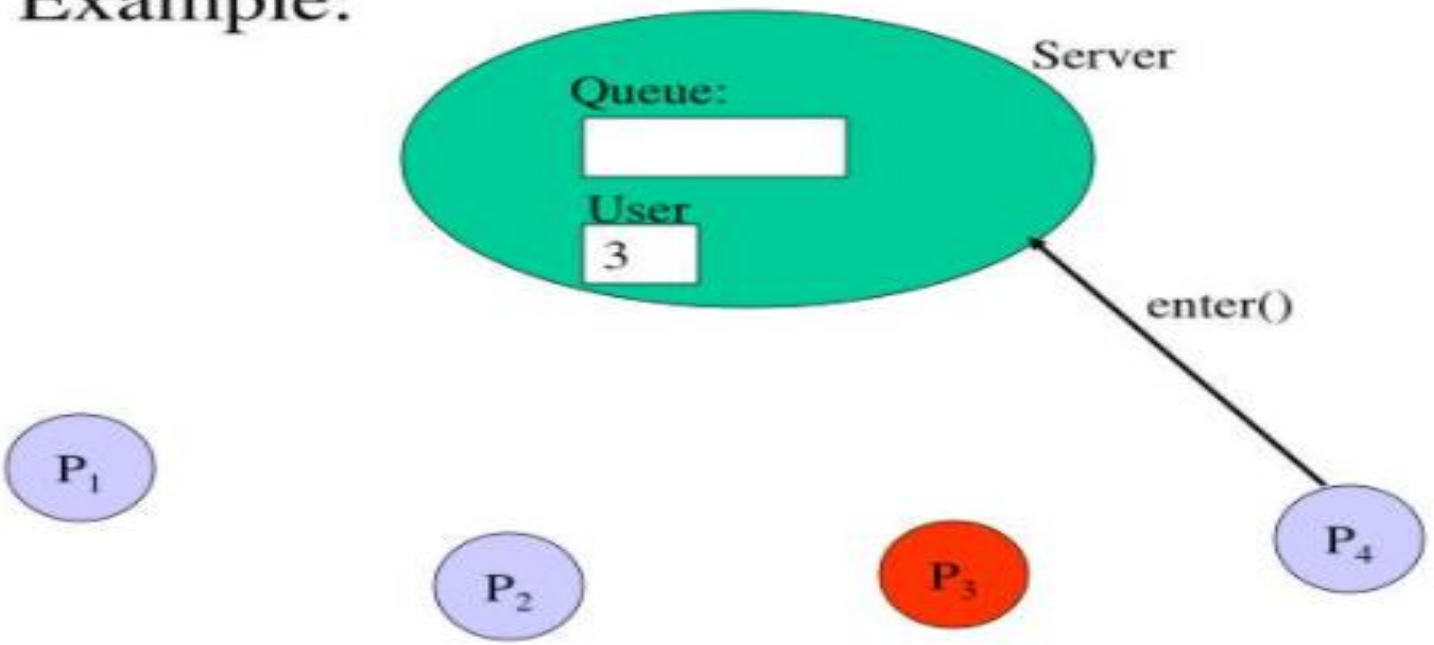
CENTRAL SERVER ALGORITHM

- Example:



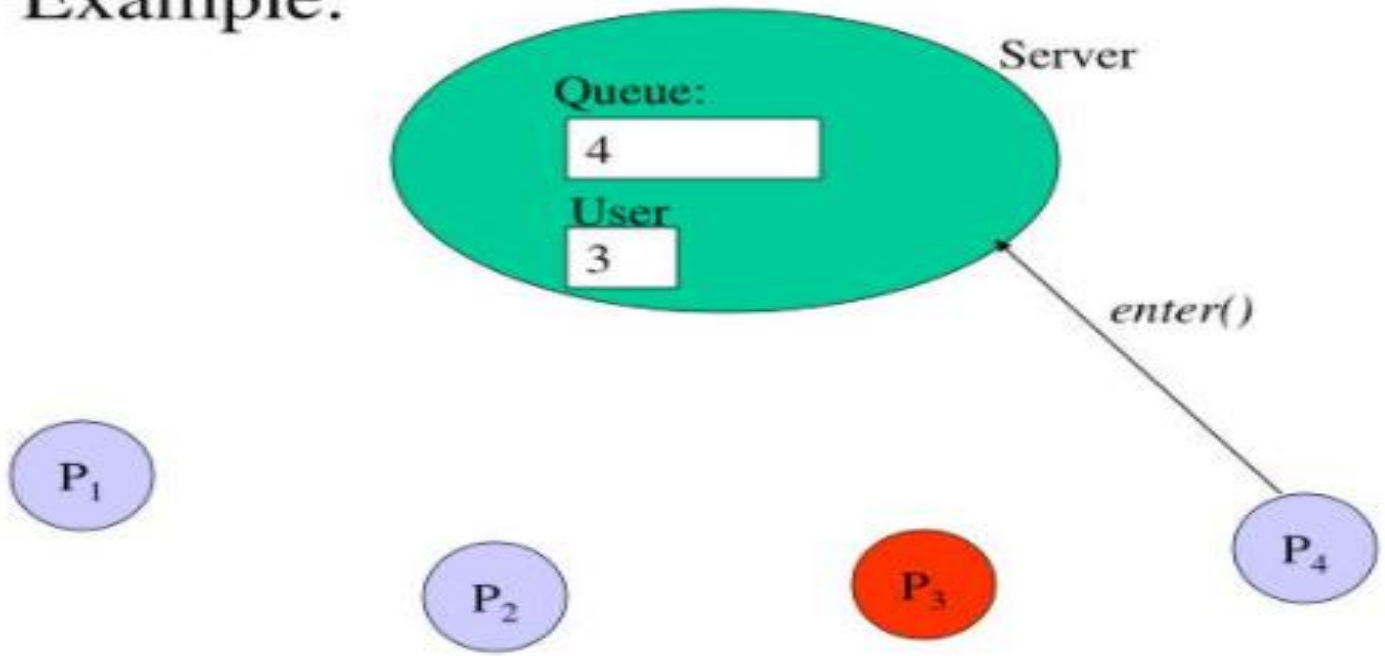
CENTRAL SERVER ALGORITHM

- Example:

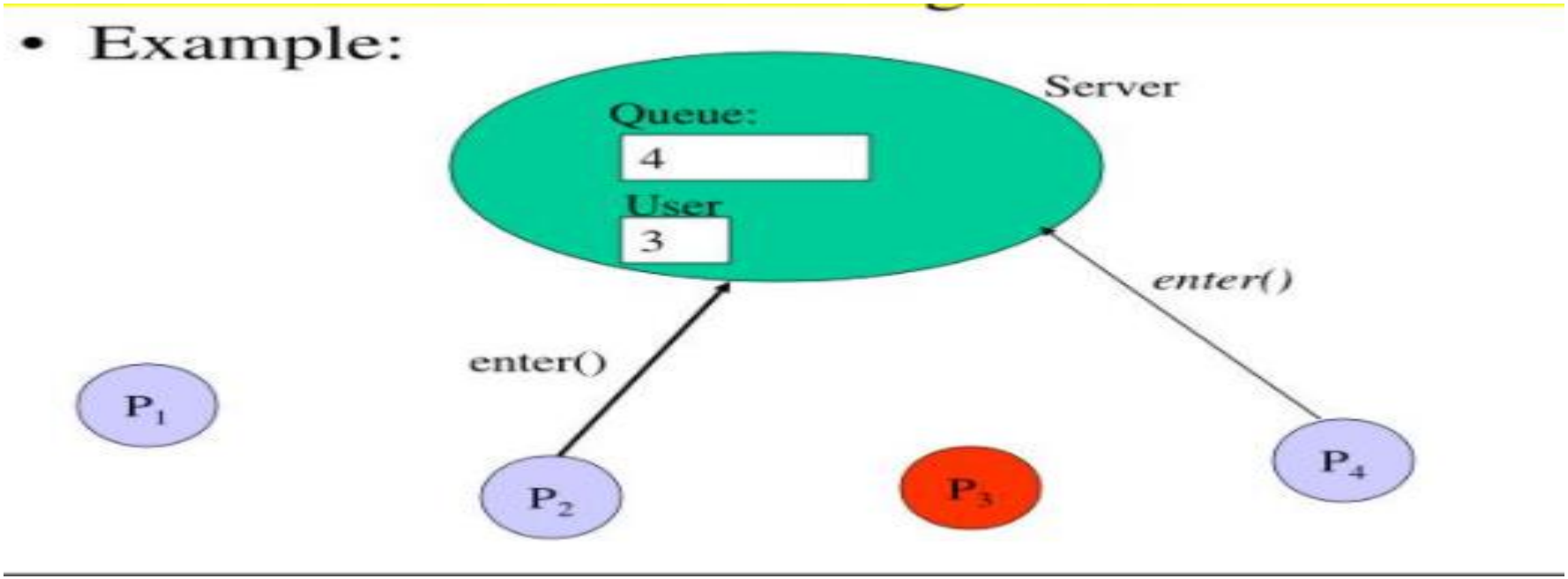


CENTRAL SERVER ALGORITHM

- Example:

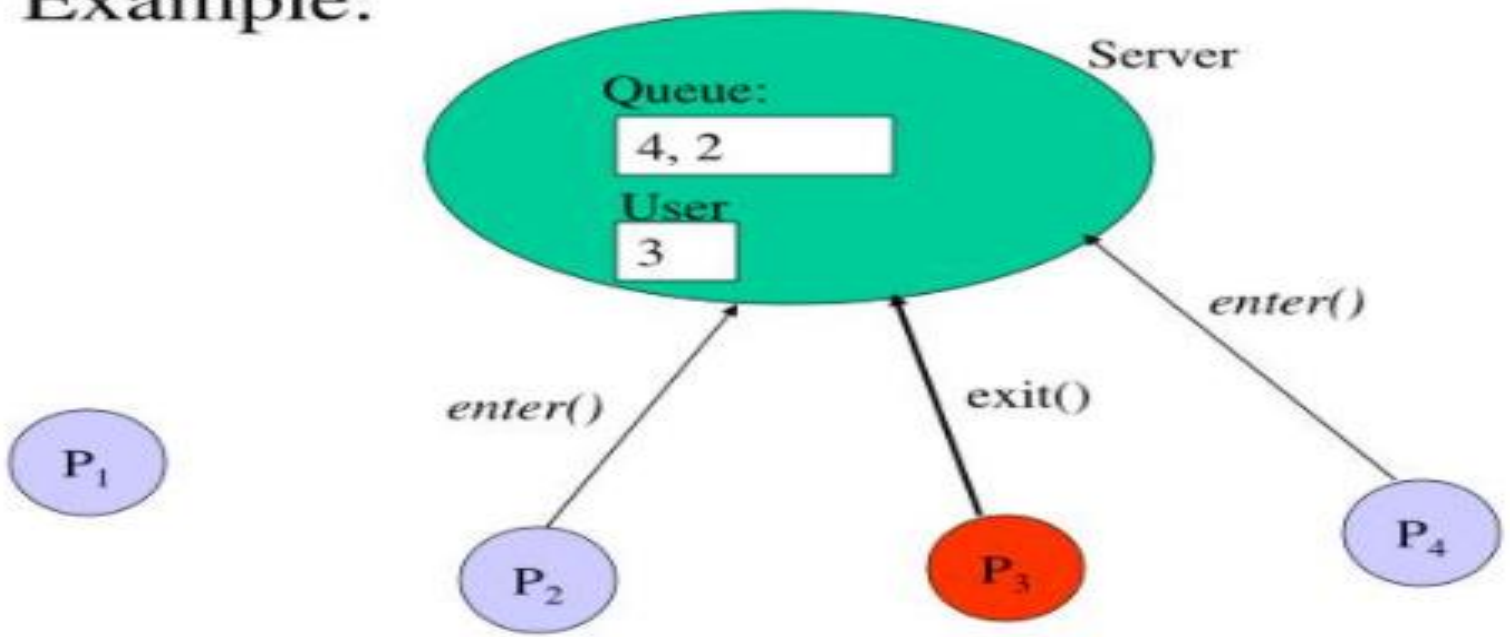


CENTRAL SERVER ALGORITHM



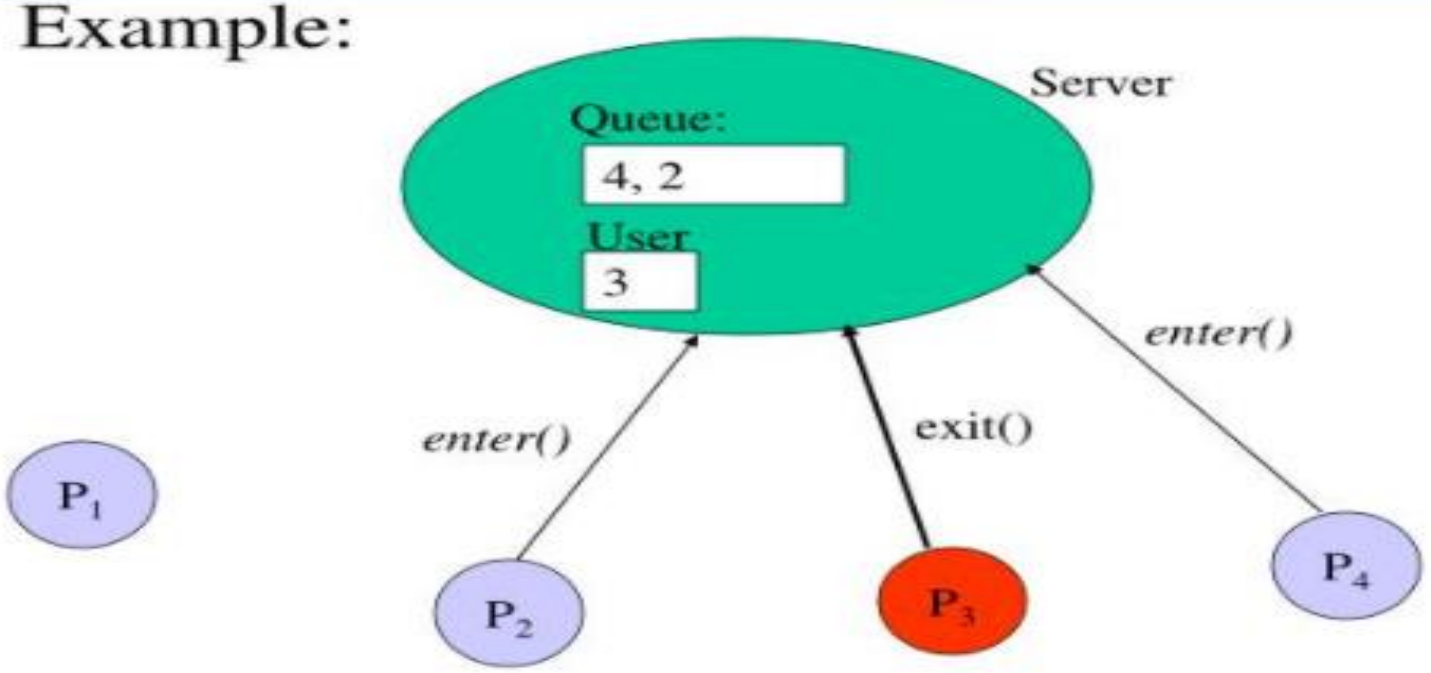
CENTRAL SERVER ALGORITHM

- Example:



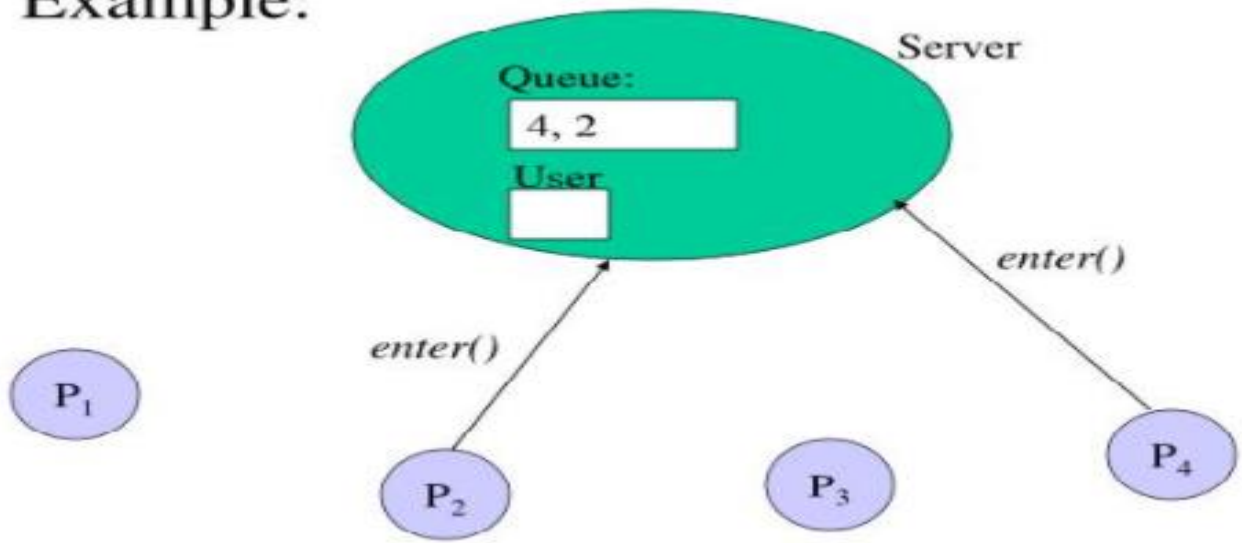
CENTRAL SERVER ALGORITHM

- Example:

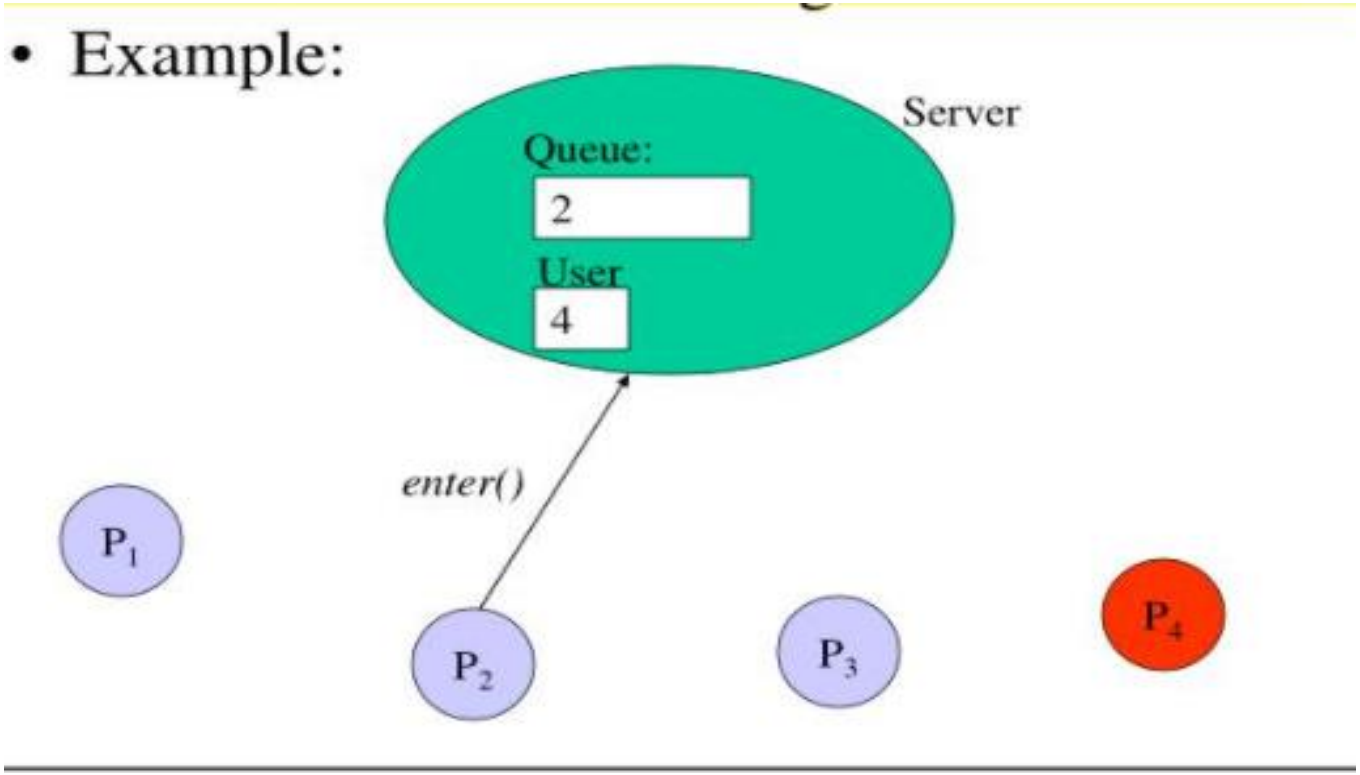


CENTRAL SERVER ALGORITHM

- Example:



CENTRAL SERVER ALGORITHM



ELECTION ALGORITHMS :BULLY ALGORITHIM

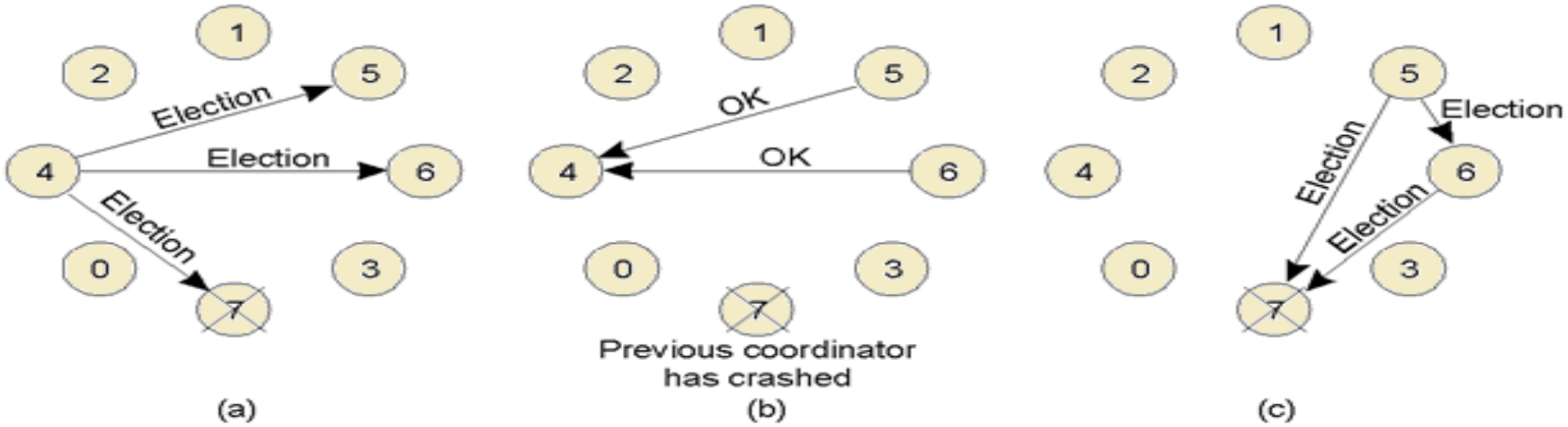
- Each process includes a unique numerical ID.
- Processes understand the Ids as well as address of each and every additional process.
- Communication is actually thought reliable.
- Key Concept: choose process along with highest ID.
- Process initiates election if it just recovered from failure or even if coordinator failed.
- 3 message kinds: election, OK, I won.
- Several processes may start a good selection concurrently and Need consistent out come .
- $O(n^2)$ messages needed along with n processes

BULLY ELECTION ALGORITHM

Bully Algorithm Details:

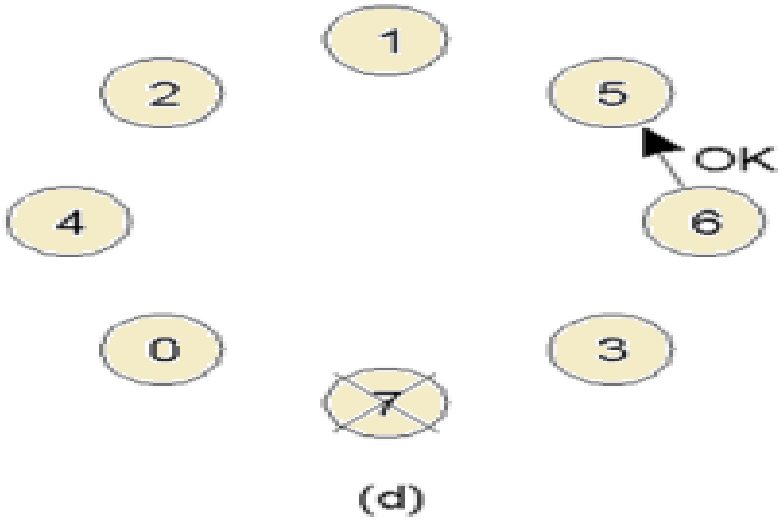
•Any process P may start initiate an election .
•P sends Election messages to any or all process along with higher Ids as well as awaits OK messages.
•If no OK messages, P becomes coordinator as well as sends I won messages to any or all process along with lower Ids.
•If this receives an OK, it drops out as well as waits to have an I won
•If the process receives an Election msg, it returns an OK as well as begins an election
•If the process gets the receives a I won, it treats sender as coordinator

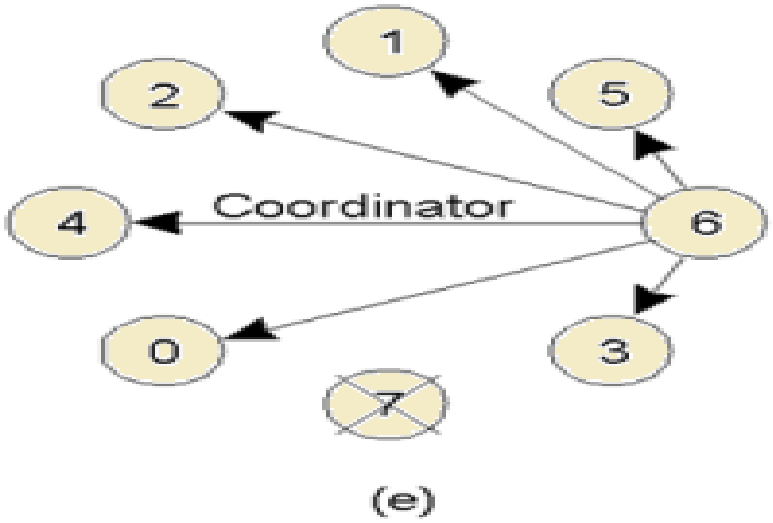
BULLY ELECTION ALGORITHM-EXAMPLE



- The bully election algorithm
- Process 4 holds an election
- Process 5 and 6 respond, telling 4 to stop
- Now 5 and 6 each hold an election

BULLY ELECTION ALGORITHM-EXAMPLE

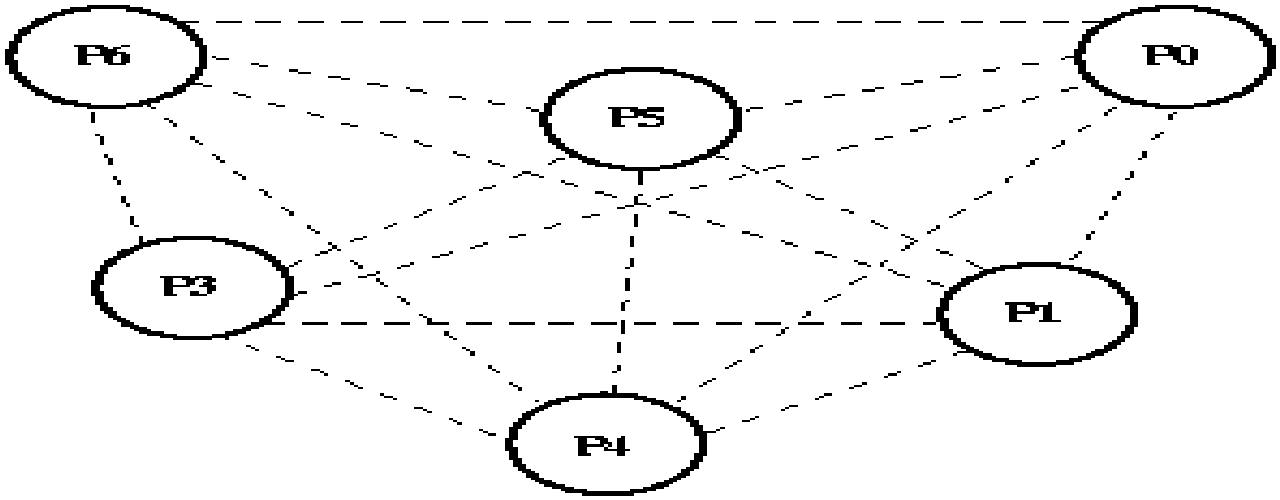




- d) Process 6 tells 5 to stop
- e) Process 6 wins and tells everyone

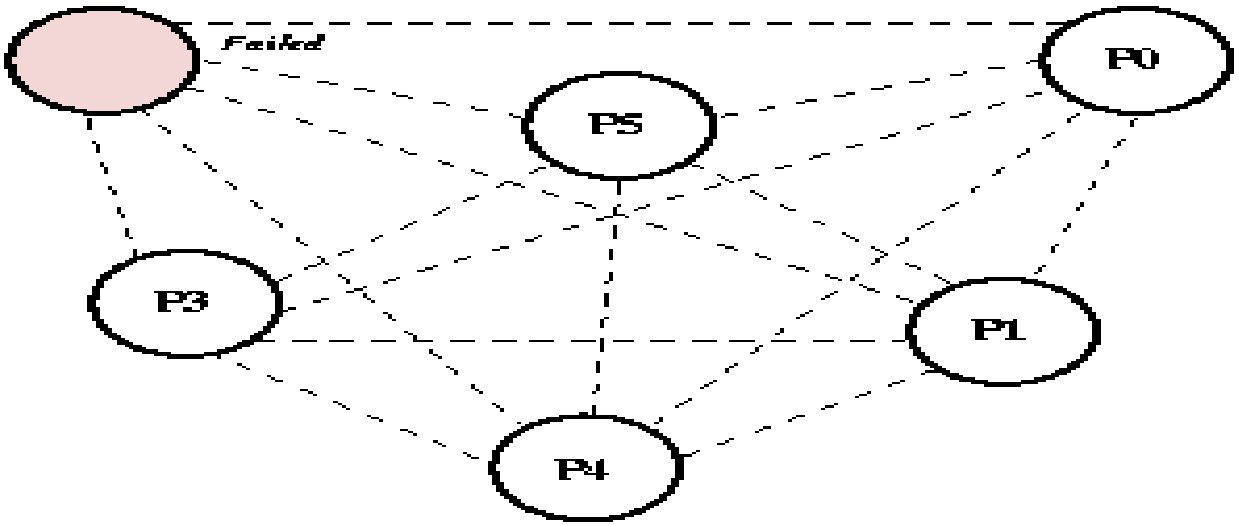
BULLY ELECTION ALGORITHM-EXAMPLE2

- We start with 6 processes , all directly connected to each other. Process 6 is the leader, as it has the highest number.



Bully Algorithm: Step 0

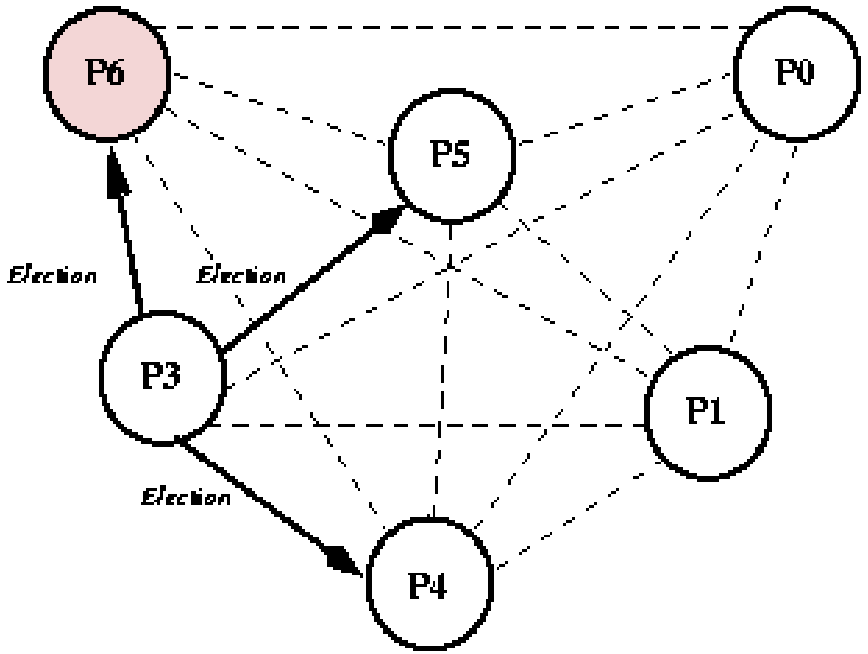
BULLY ELECTION ALGORITHM-EXAMPLE2



Bully Algorithm: Step 1

Process 6 fails.

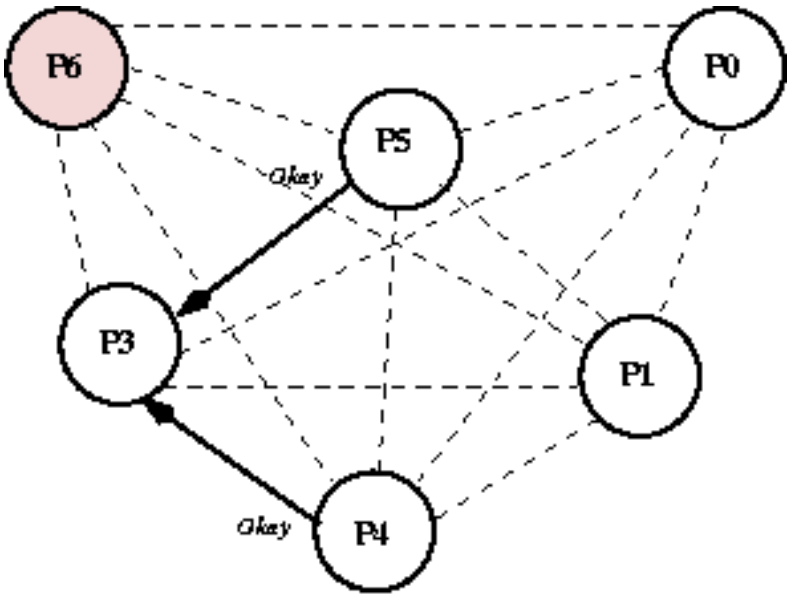
BULLY ELECTION ALGORITHM-EXAMPLE2



Bully Algorithm: Step 2

- Process 3 notices that Process 6 does not respond So it starts an election, notifying those processes with ids greater than 3.

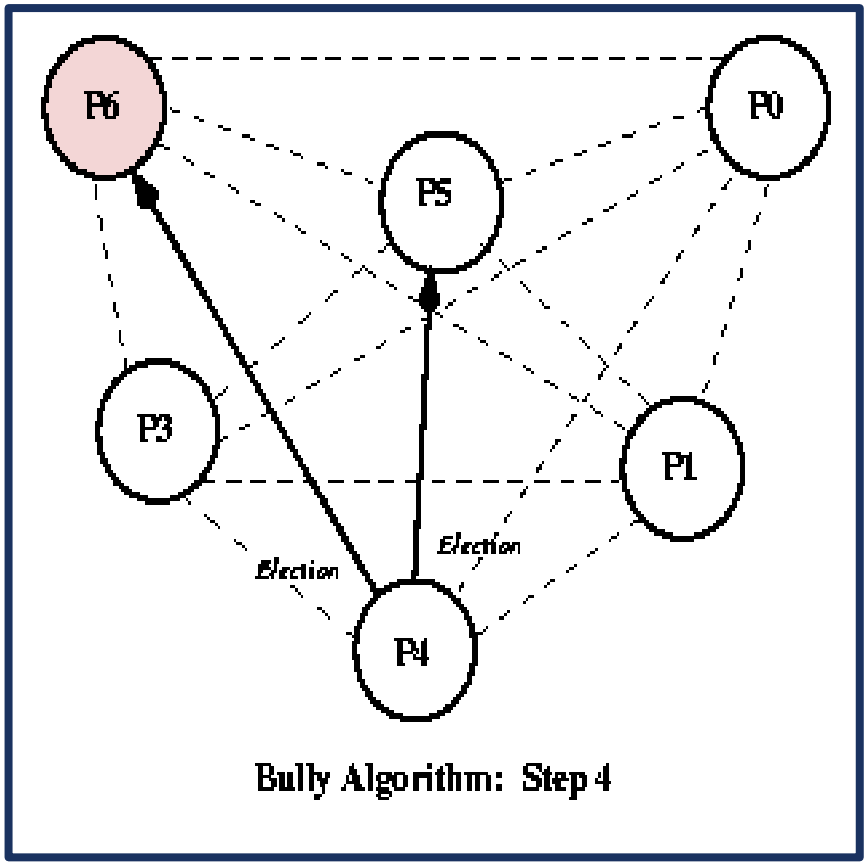
BULLY ELECTION ALGORITHM-EXAMPLE2



Bully Algorithm: Step 3

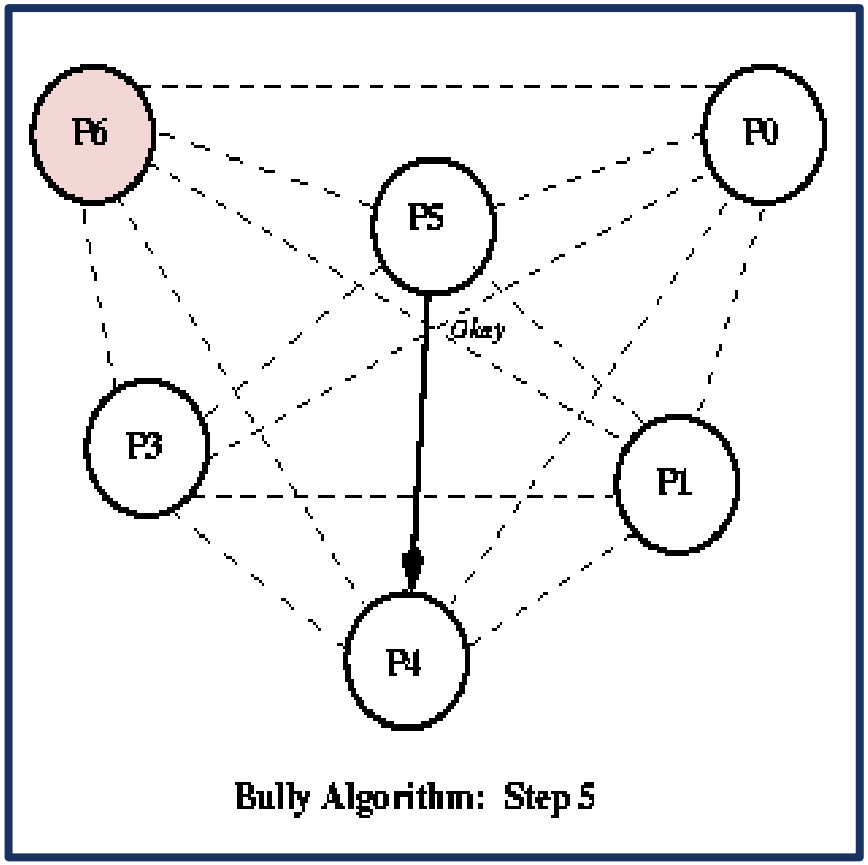
- Both Process 4 and Process 5 respond, telling Process 3 that they'll take over from here.

BULLY ELECTION ALGORITHM-EXAMPLE2



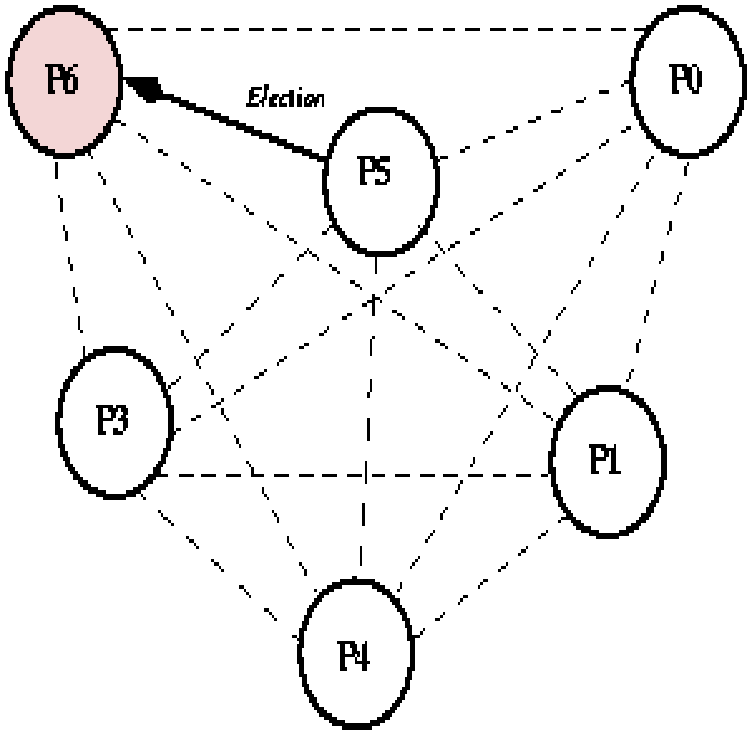
- Process 4 sends election messages to both Process 5 and Process 6.

BULLY ELECTION ALGORITHM-EXAMPLE2



Only Process 5 answers and takes over the election..

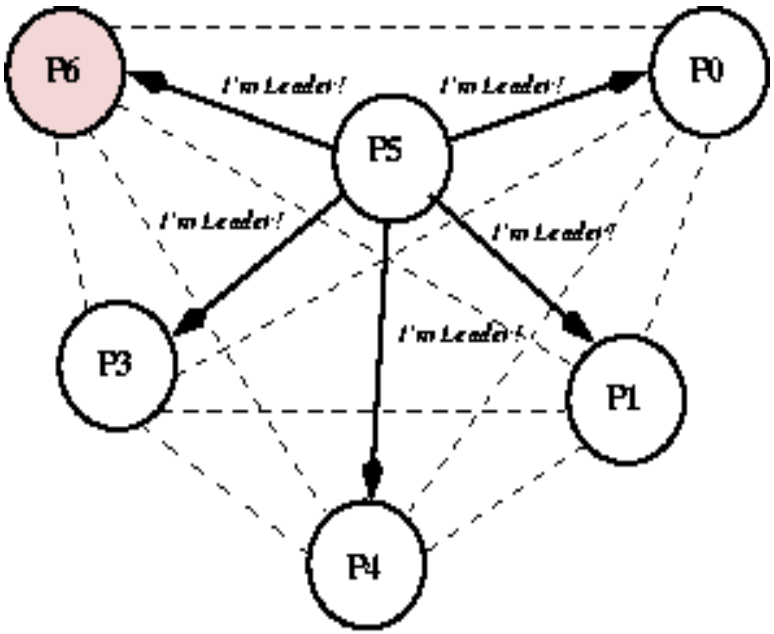
BULLY ELECTION ALGORITHM-EXAMPLE2



Bully Algorithm: Step 6

Process 5 sends out only one election message to Process 6.

BULLY ELECTION ALGORITHM-EXAMPLE2



Bully Algorithm: Step 7

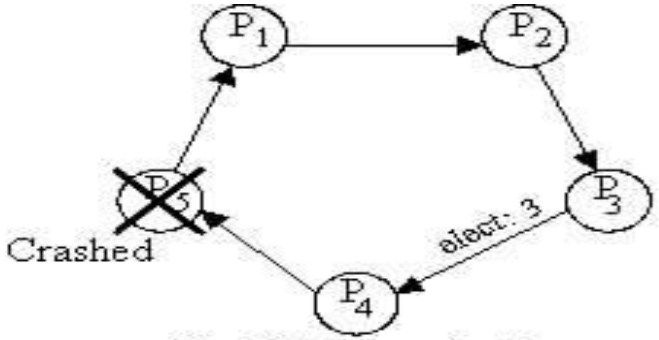
When Process 6 does not respond Process 5 declares itself the winner.

RING ELECTION ALGORITHIM

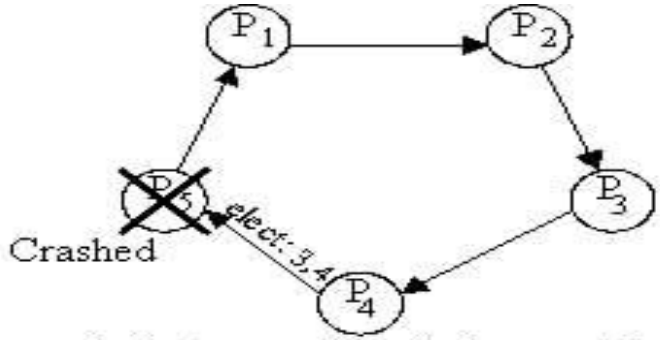
•all processes are arranged in a logical ring (each process knows its successor)

-When a process notices that the coordinator is down, it sends an *elect* message to its successor containing its process number.
-If the successor does not acknowledge, the process by-passes its successor and sends the *elect* message to the next process after its successor.
- On receiving an *elect* message a process appends its process number to the message before forwarding it to the next process in the ring. When the *elect* message reaches the originator, the message type is changed to *coordinator* with the highest process number being the coordinator. As the *coordinator* message traverses the ring, each process is informed of the coordinator and all the active processes in the ring, called the *active list.*

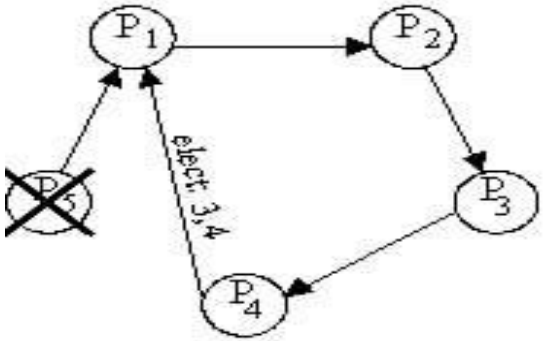
RING ELECTION ALGORITHM EXAMPLE -1



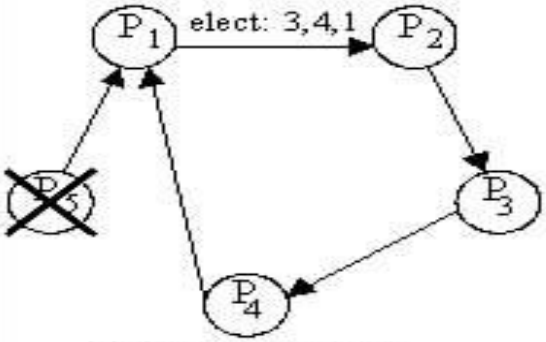
P_3 initiate an election



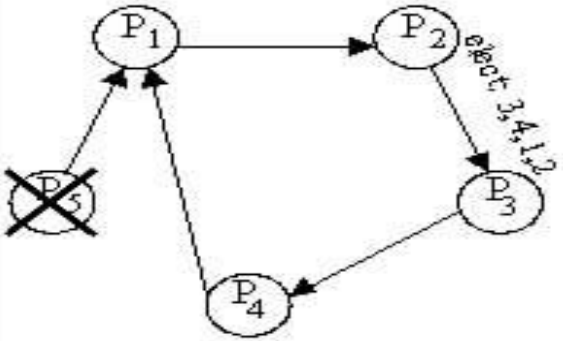
P_4 forwards, but no acknowledgement from P_5



P_4 by-passes P_5



P_1 forwards to P_2



P_3 knows that P_4 is new coordinator

RING ELECTION ALGORITHIM

Principle

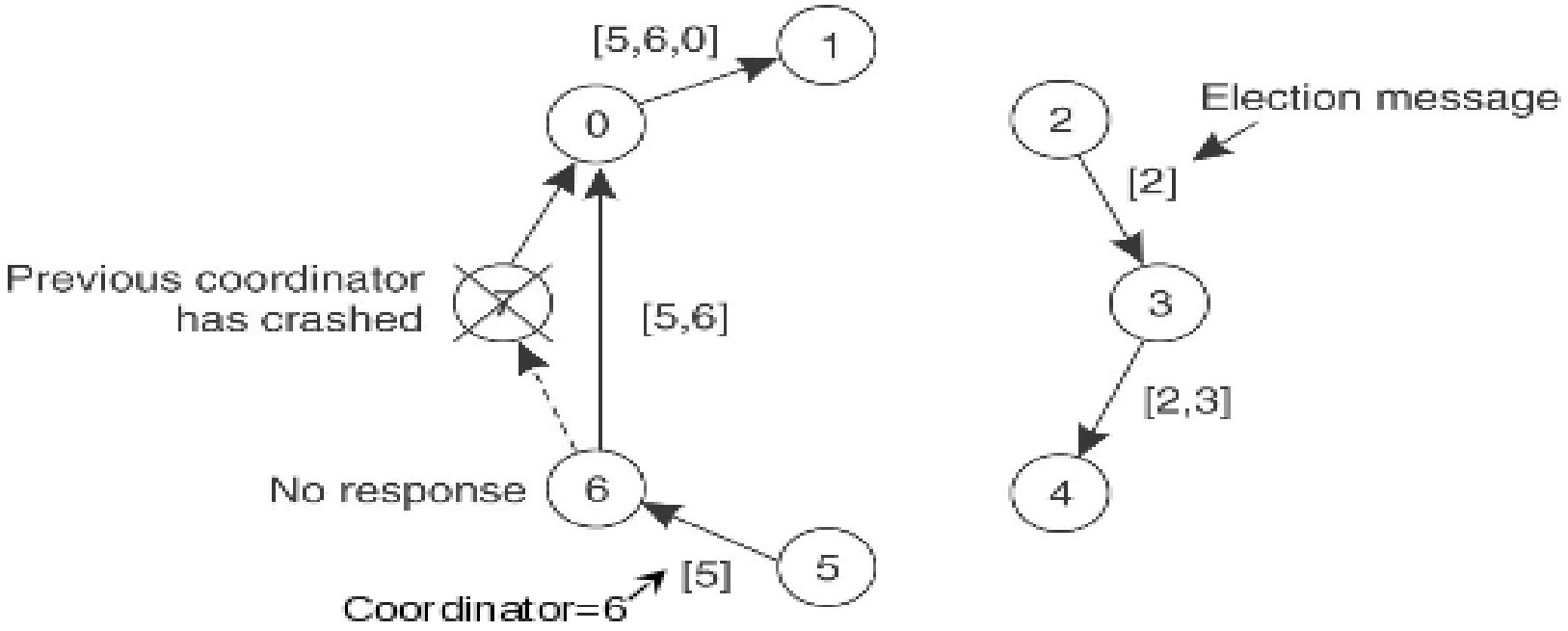
Process priority is obtained by organizing processes into a (logical) ring. Process with the highest priority should be elected as coordinator.

•Any process can start an election by sending an election message to its successor. If a successor is down, the message is passed on to the next successor.

•If a message is passed on, the sender adds itself to the list. When it gets back to the initiator, everyone had a chance to make its presence known.

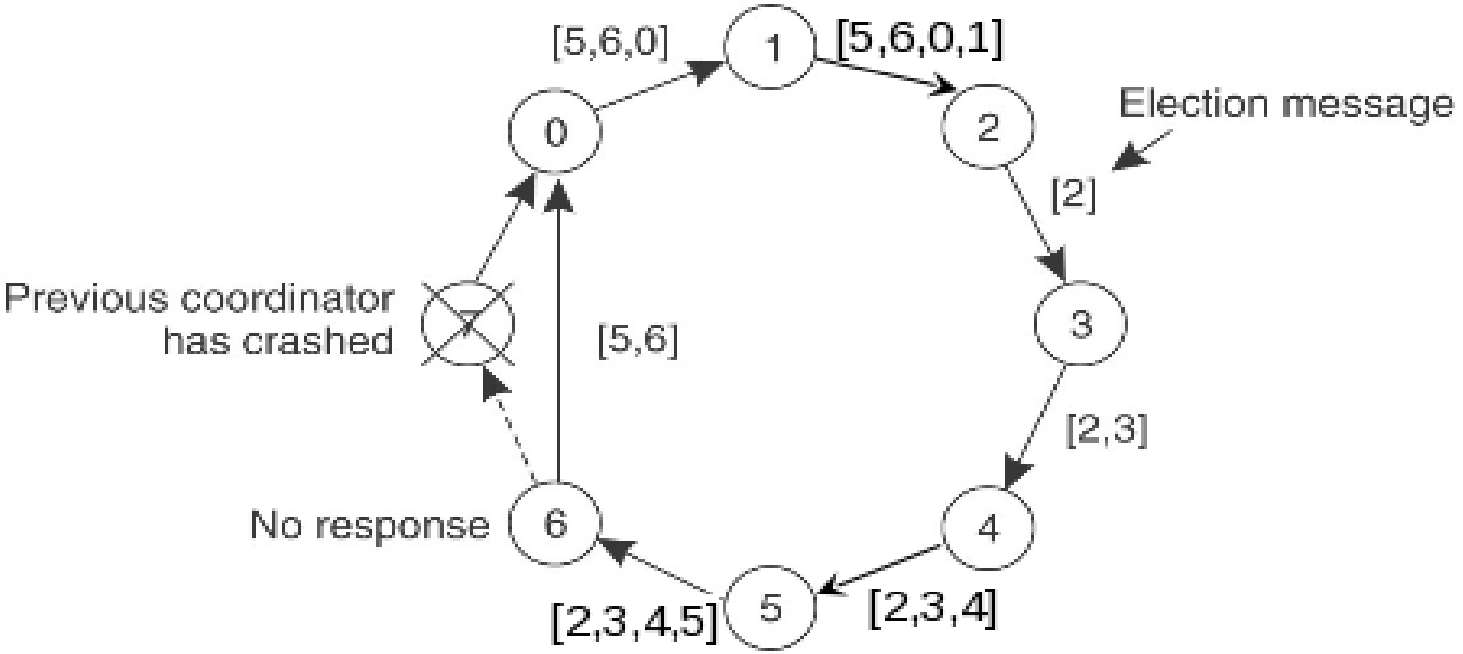
•The initiator sends a coordinator message around the ring containing a list of all living processes. The one with the highest priority is elected as coordinator

RING ELECTION ALGORITHIM



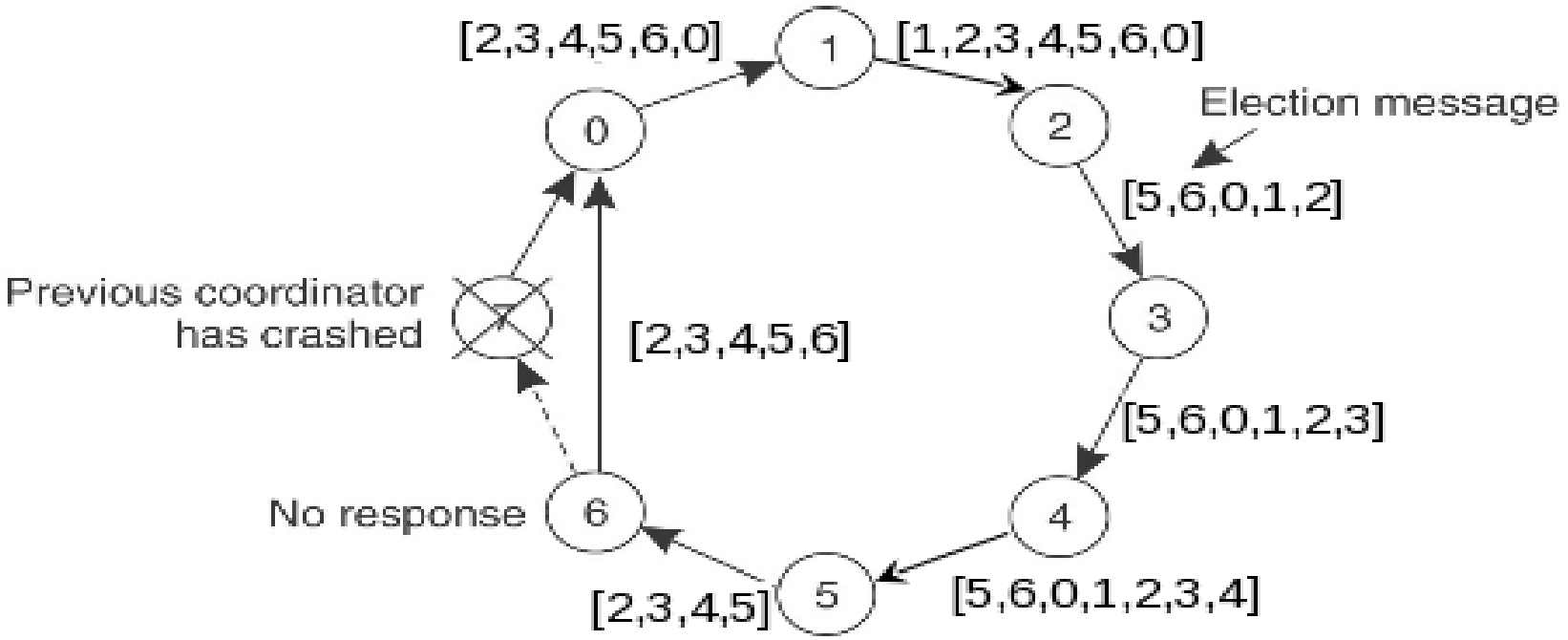
2 and 5 start election message independently.

RING ELECTION ALGORITHIM



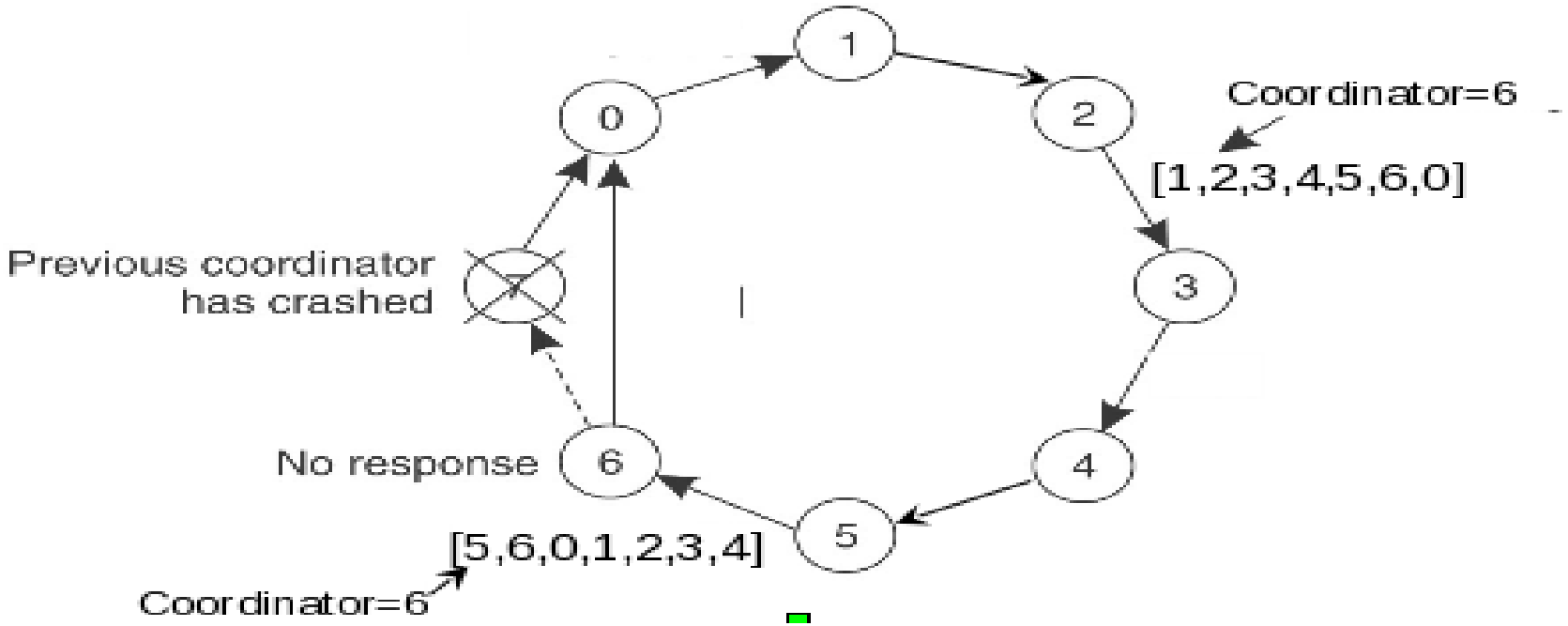
Both messages continue to circulate.

RING ELECTION ALGORITHIM



Eventually, both messages will go all the way around .

RING ELECTION ALGORITHIM



2 and 5 will convert Election messages to COORDINATOR messages.
All processes recognize highest numbered process as new coordinator.

RING ELECTION ALGORITHIM: PROBLEM 2

Problem 2: Modify the basic ring-based leader election algorithm to elect 2 leaders (two processes with the highest IDs).

Answer:

To start an election, a process sends a message < election > with its ID appended. Each node that receives this message appends its ID to it. Once the election message reaches the initiator after going through the circle, the top two nodes (nodes with the highest and second highest IDs) are selected, and a message < choose : highest1, highest2 > is sent with the initiator's ID appended to it. Each node that receives this message appends its ID again. When this message reaches the initiator, if the appended ID list contains the top two nodes, then the election is successful and ends. Otherwise, a new election is initiated.

CHANG-ROBERTS RING ALGORITHIM

1.initially each process in the ring is marked as *non-participant*.
2.A process that notices a lack of leader starts an election. It creates an *election message* containing its UID. It then sends this message clockwise to its neighbour.
3.Every time a process sends or forwards an *election message*, the process also marks itself as a participant.
4.When a process receives an *election message* it compares the UID in the message with its own UID.

1. If the UID in the election message is larger, the process unconditionally forwards the *election message* in a clockwise direction.
2. If the UID in the election message is smaller, and the process is not yet a participant, the process replaces the UID in the message with its own UID, sends the updated *election message* in a clockwise direction.
3. If the UID in the election message is smaller, and the process is already a *participant* (i.e., the process has already sent out an election message with a UID at least as large as its own UID), the process discards the election message.
4. If the UID in the incoming election message is the same as the UID of the process, that process starts acting as the leader.

CHANG-ROBERTS RING ALGORITHM

When a process starts acting as the leader, it begins the second stage of the algorithm.

1.The leader process marks itself as *non-participant* and sends an *elected message* to its neighbour announcing its election and UID.

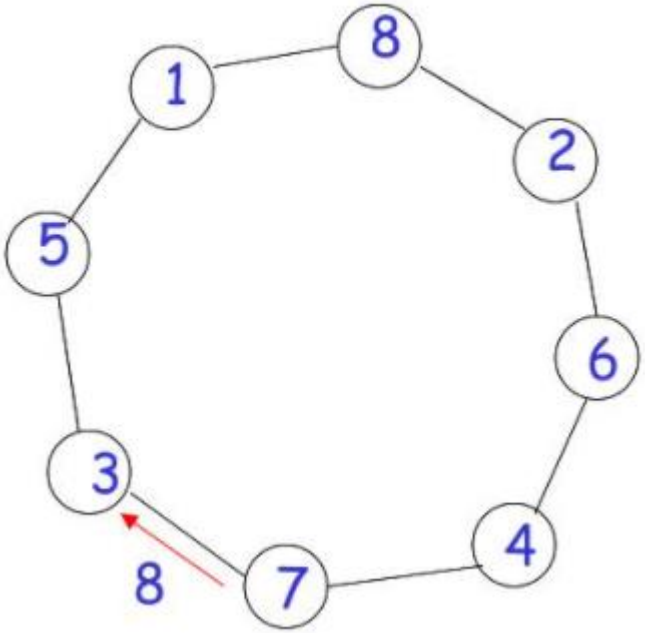
2.When a process receives an *elected message*, it marks itself as *non-participant*, records the elected UID, and forwards the *elected message* unchanged.

3.When the *elected message* reaches the newly elected leader, the leader discards that message, and the election is over.

Assuming there are no failures this algorithm will finish. The algorithm works for any number of processes N, and does not require any process to know how many processes are in the ring.

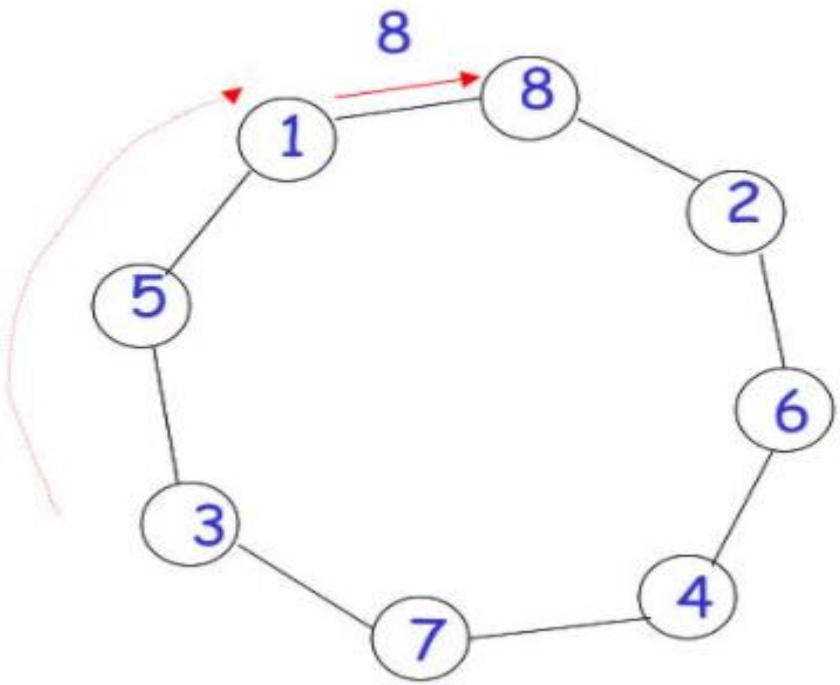
CHANG-ROBERTS RING ALGORITHIM

Each node sends a message with its id to the left neigbour
If message received id greater than current node then forward the message

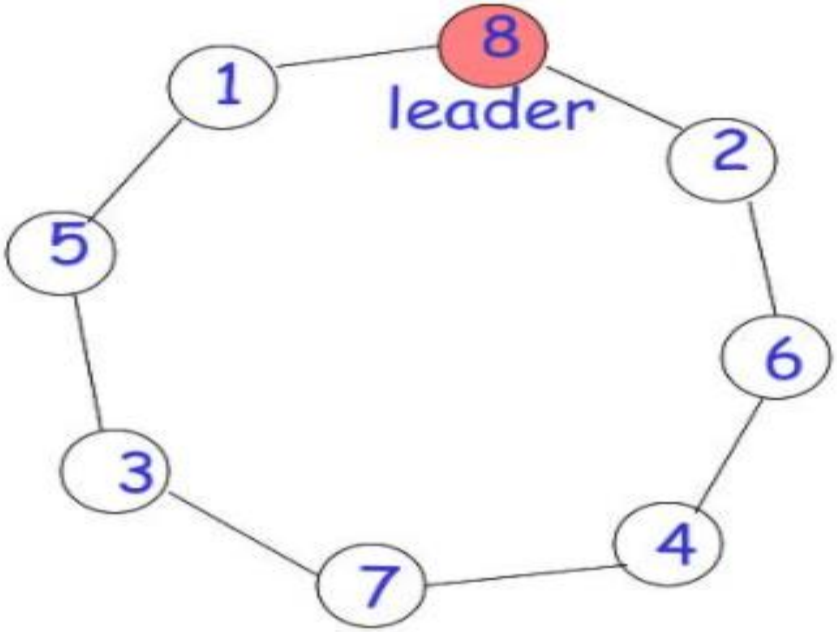


CHANG-ROBERTS RING ALGORITHIM

If a node receives its own message then it elects itself as aleader

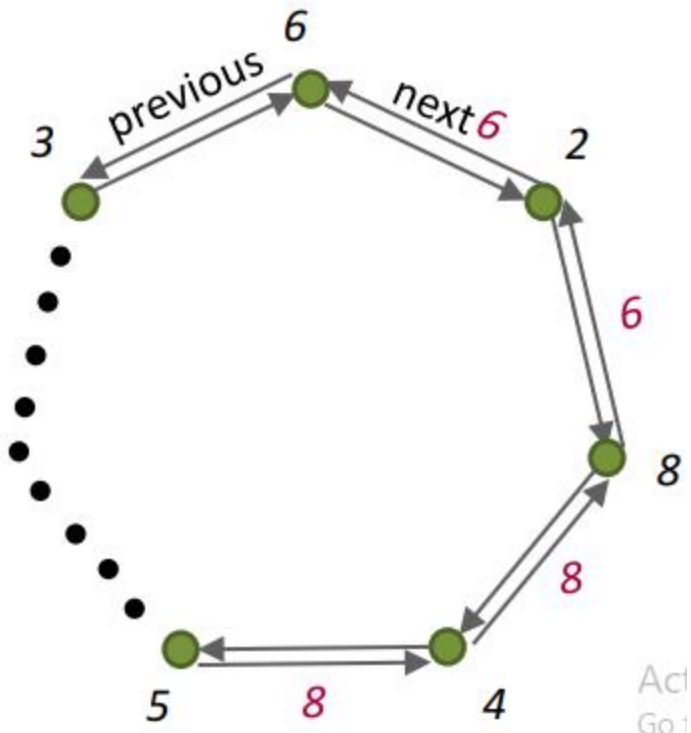


CHANG- ROBERTS RING ALGORITHIM



CHANG-ROBERTS RING ALGORITHIM

- The value "8" goes around the ring and comes back to 8
- Then 8 knows that "8" is the highest ID
  - *Since if there was a higher ID, that would have stopped 8.*
- 8 declares itself the leader: sends a message around the ring.



CHANG-ROBERTS RING ALGORITHM

- If node p receives election message m with m.ID=p.ID
- P declares itself leader
  - Set p.leader=p.ID
  - Send leader message with p.ID to p.NEXT
  - Any other node q receiving the leader message
    - Set q.leader=p.ID
    - Forwards leader message to q.NEXT

PROCESS SYNCHRONIZATION-MUTUAL EXCLUSION

Thanks