Synchronization

Election Algorithms

Election Algorithms

- Many distributed algorithms require one process to act as coordinator, initiator, or otherwise perform a special role.
- It does not matter who does it but one of them has to be a coordinator.
- These algorithms will assume that each active process has a unique priority id.
- In general, election algorithms attempt to locate the process with the highest process number and designate it as the coordinator. (could use network addresses)
  - For simplicity we assume one process per machine

Election Algorithms: Election principles

- Any process can call for an election.
- A process can call for at most one election at a time.
- There could be concurrent calls for the same election.
- The result of an election does not depend on which process calls for it.
- Processes involved in an election are called the "participants" of the election.
- The non-failed process with the best election attribute value (e.g., highest id or address, fastest cpu, etc.) is elected.
- A run of the election algorithm must always guarantee at the end:
  - Safety: Non-failed process selected has best attribute value
  - Liveness: All processes participate and set one as coordinator

Bully Algorithm

- Each process has a unique numerical ID
- Processes know the Ids and address of every other process
- Communication is assumed reliable
- *Key Idea*: select process with highest ID
- Process initiates election if it just recovered from failure or if coordinator failed
- 3 message types: *election*, *OK*, *I won*
- Several processes can initiate an election simultaneously
  - Need consistent result

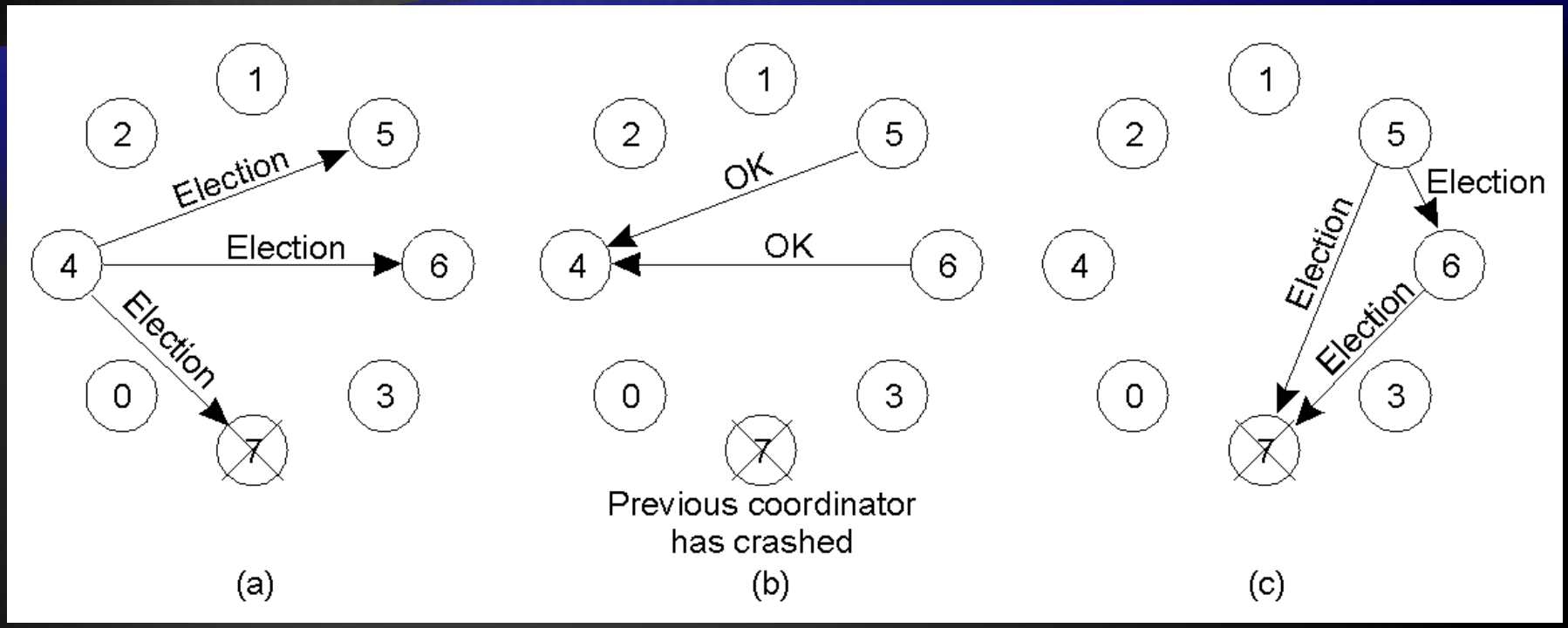
Election Algorithms:
Bully Algorithm

- When any process, P, notices that the coordinator is no longer responding it initiates an election:
  - P sends an *election* message to all processes with higher id numbers
  - If no one responds, P wins the election and becomes coordinator
  - If a higher process responds, it takes over. Process P's job is done

Election Algorithms:
Bully Algorithm

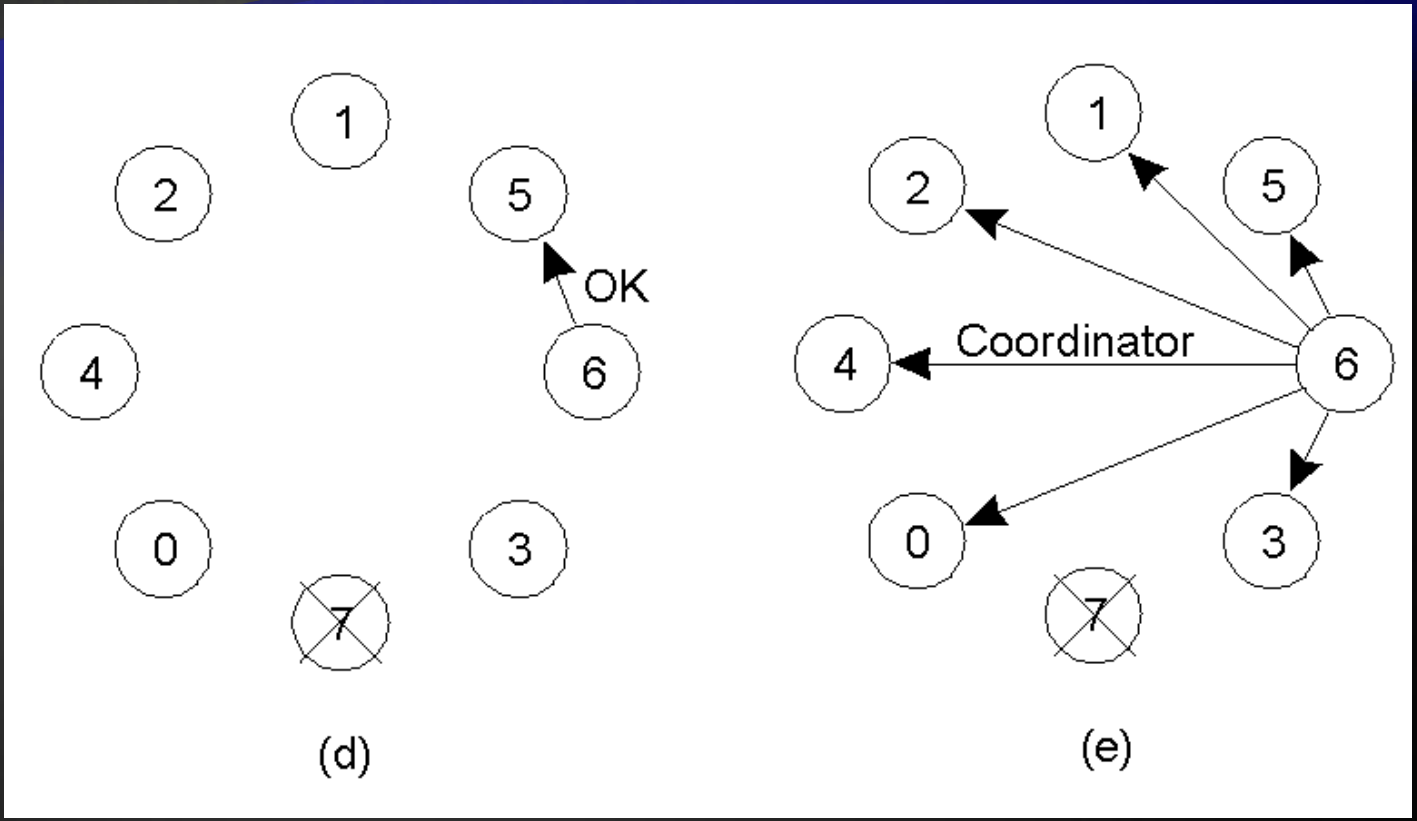
- At any moment, a process can receive an election message from one of its lower-numbered colleagues.
- The receiver sends an OK back to the sender and conducts its own election.
- Eventually only the bully process remains. The bully announces victory to all processes in the distributed group by sending a COORDINATOR message

Election Algorithms: Bully Algorithm, Example



- Process 4 notices 7 down.
- Process 4 holds an election.
- Process 5 and 6 respond, telling 4 to stop.
- Now 5 and 6 each hold an election.

Election Algorithms: Bully Algorithm, Example



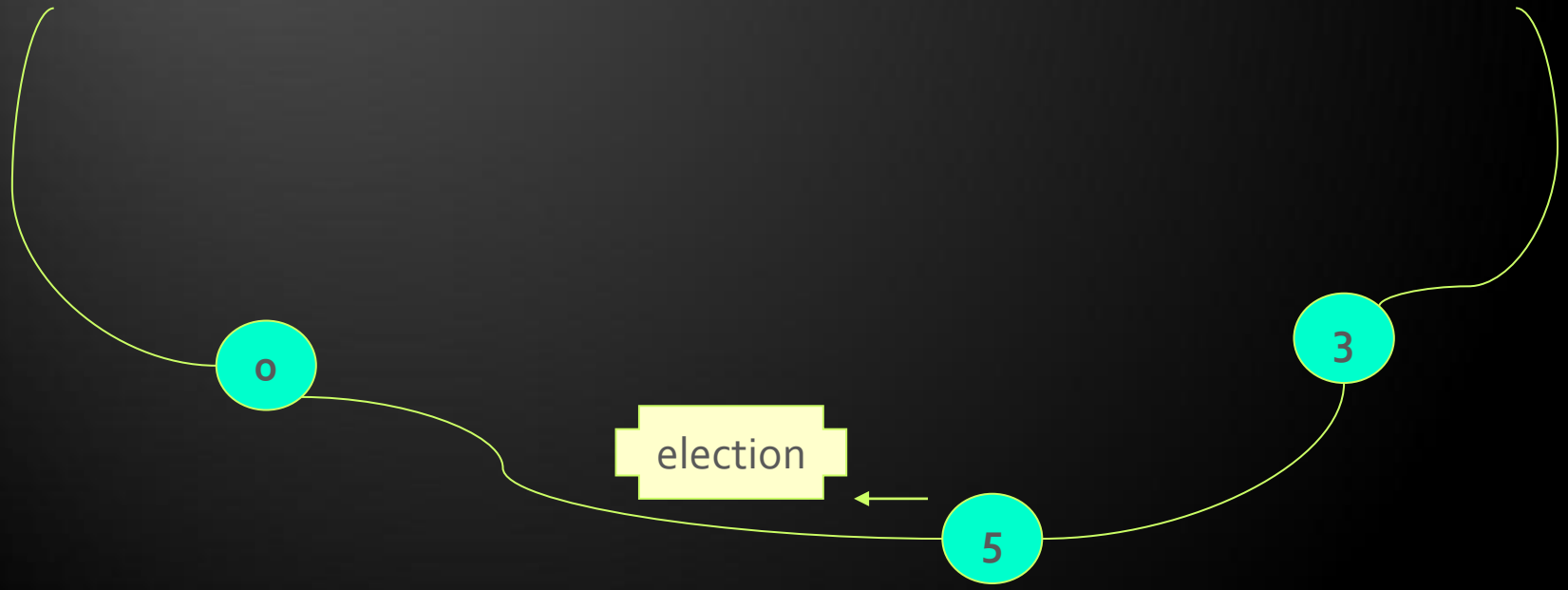
- Process 6 tells process 5 to stop.
- Process 6 (the bully) wins and tells everyone.
- If processes 7 comes up, starts elections again.

Election Algorithms:
Ring Algorithm

- Nodes are physically or logically organized in a ring.
- Nodes might not know the number of nodes in the ring and what all of the IDs are; they only know who their successor is.
- The ring can be unidirectional, in which case, they only need to be able to send messages to their clockwise neighbor.

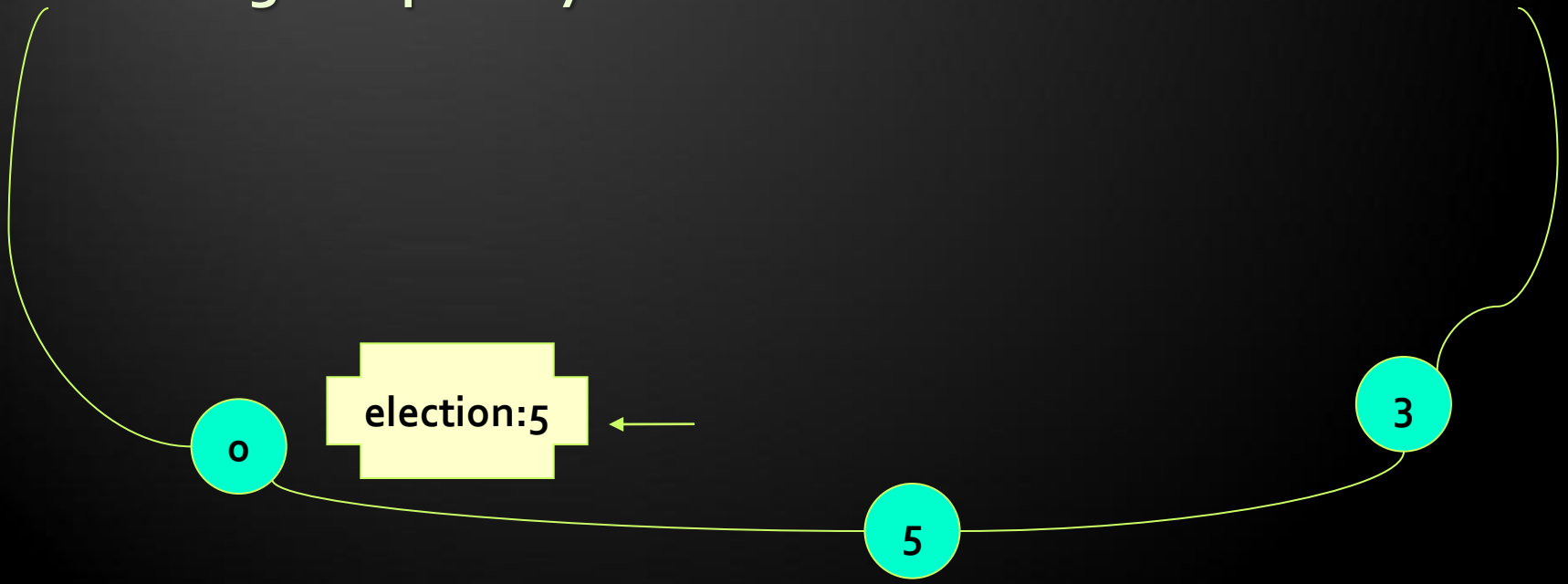
Election Algorithms: Ring Algorithm

- Any node that notices that the leader is not functioning, changes his state to Election, starts an election message containing his ID and sends it to his clockwise neighbor
- If a successor is down, the message is passed on to the next successor.



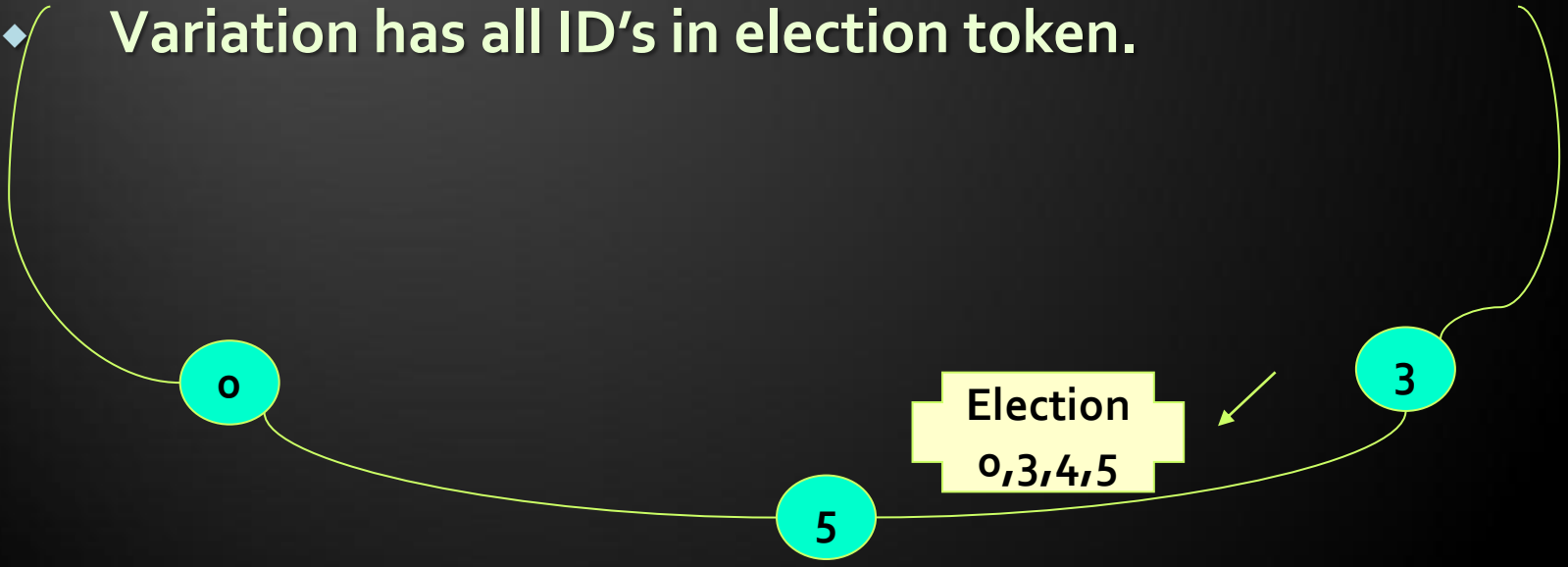
Election Algorithms: Ring Algorithm

- As 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

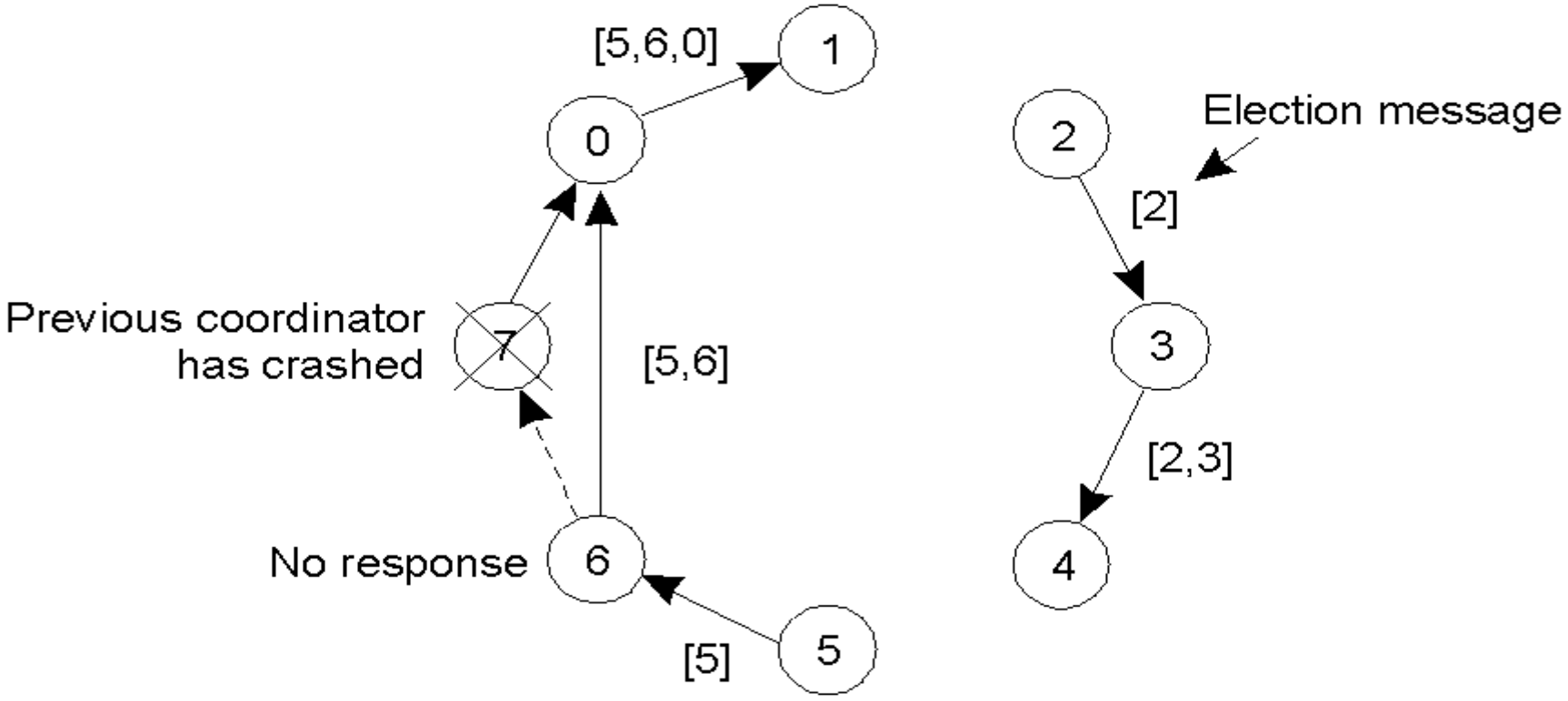


Election Algorithms: Ring Algorithm

- When a node receives a "coordinator message", he records the ID of the new leader and changes state to Normal, and forwards the message (unless he is the leader).
- Variation has all ID's in election token.

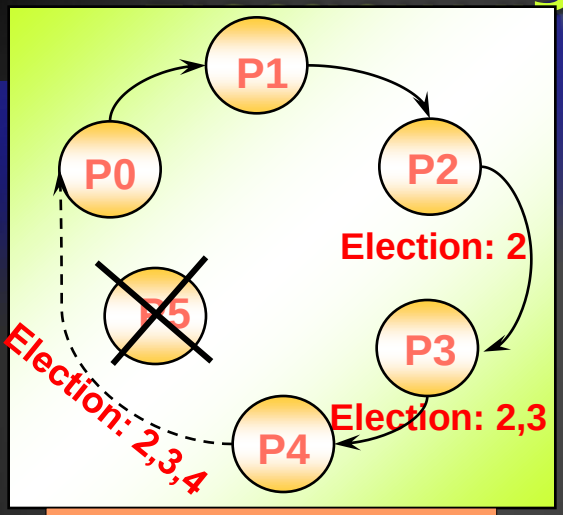


Election Algorithms: Ring Algorithm, Example

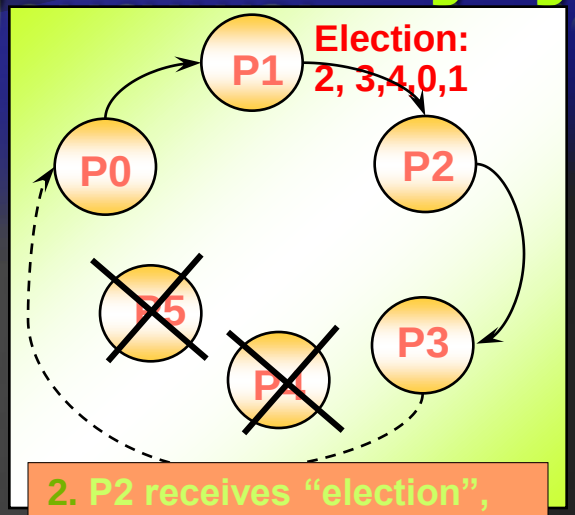


- Even if two ELECTIONS start at once, everyone will pick the same leader.

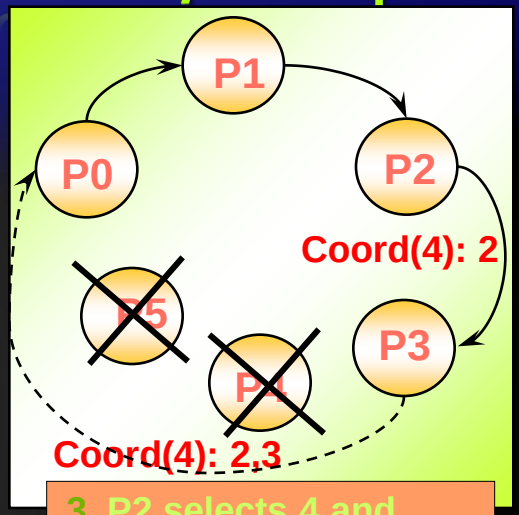
Election Algorithms: Ring Algorithm, Example



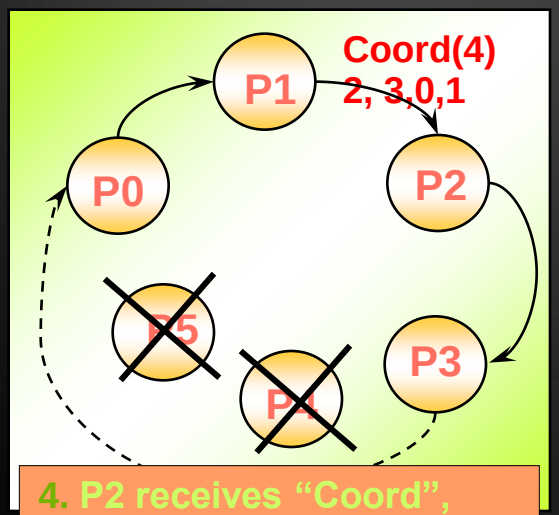
1. P2 initiates election



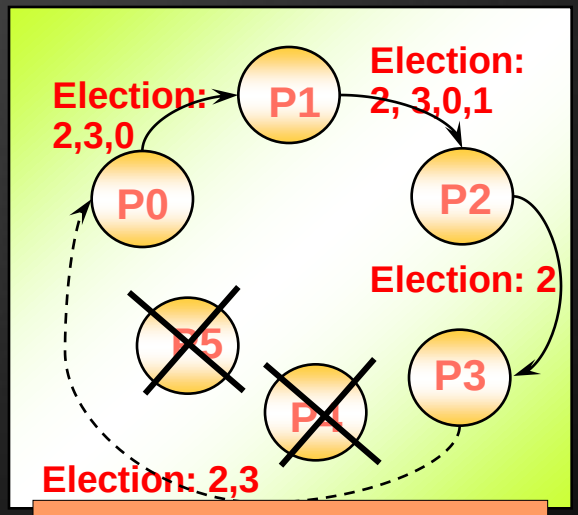
2. P2 receives "election", P4 dies



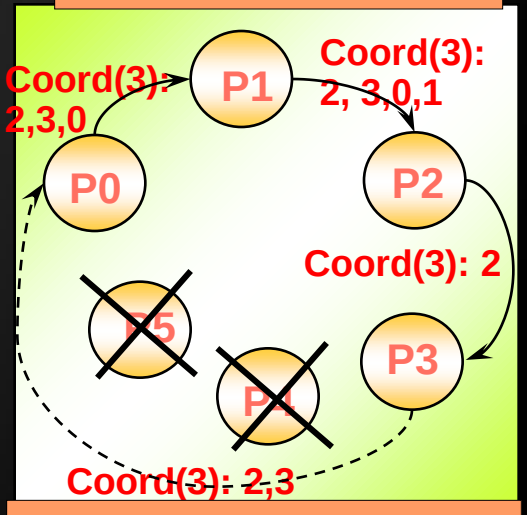
3. P2 selects 4 and announces the result



4. P2 receives "Coord", but P4 is not included



5. P2 re-initiates election



6. P3 is finally elected

Elections in Wireless Environments

- Traditional election algorithms are generally based on assumptions that are not realistic in wireless environments.
- For example, they assume that message passing is reliable and that the topology of the network does not change.
- These assumptions are false in most wireless environments, especially those for mobile adhoc networks.

Elections in Wireless Environments

- Only few protocols for elections have been developed that work in adhoc networks. Vasudevan et al. (2004) propose a solution that can handle failing nodes and partitioning networks.
- An important property of their solution is that the best leader can be elected rather than just a random as was more or less the case in the previously discussed solutions.

Elections in Wireless Environments

- Consider a wireless ad hoc network. To elect a leader, any node in the network, called the source, can initiate an election by sending an ELECTION message to its immediate neighbors (i.e., the nodes in its range).
- When a node receives an ELECTION for the first time, it designates the sender as its parent and subsequently sends out an ELECTION message to all its immediate neighbors, except for the parent.

Elections in Wireless Environments

- When a node receives an ELECTION message from a node other than its parent, it merely acknowledges the receipt.
- When node R has designated node Q as its parent, it forwards the ELECTION message to its immediate neighbors (excluding Q) and waits for acknowledgments to come in before acknowledging the ELECTION message from Q. This waiting has an important consequence.

Elections in Wireless Environments

- First, note that neighbors that have already selected a parent will immediately respond to R.
- If all neighbors already have a parent, R is a leaf node and will be able to report back to Q quickly. In doing so, it will also report information such as its battery lifetime and other resource capacities.

Elections in Wireless Environments

- This information will later allow Q to compare R's capacities to that of other downstream nodes, and select the best eligible node for leadership.
- Note: Q had sent an ELECTION message only because its own parent P had done so as well. In turn, when Q eventually acknowledges the ELECTION message previously sent by P, it will pass the most eligible node to P as well.

Elections in Wireless Environments

- In this way, the source will eventually get to know which node is best to be selected as leader after which it will broadcast this information to all other nodes.
- When multiple elections are initiated, each node will decide to join only one election. To this end, each source tags its ELECTION message with a unique identifier

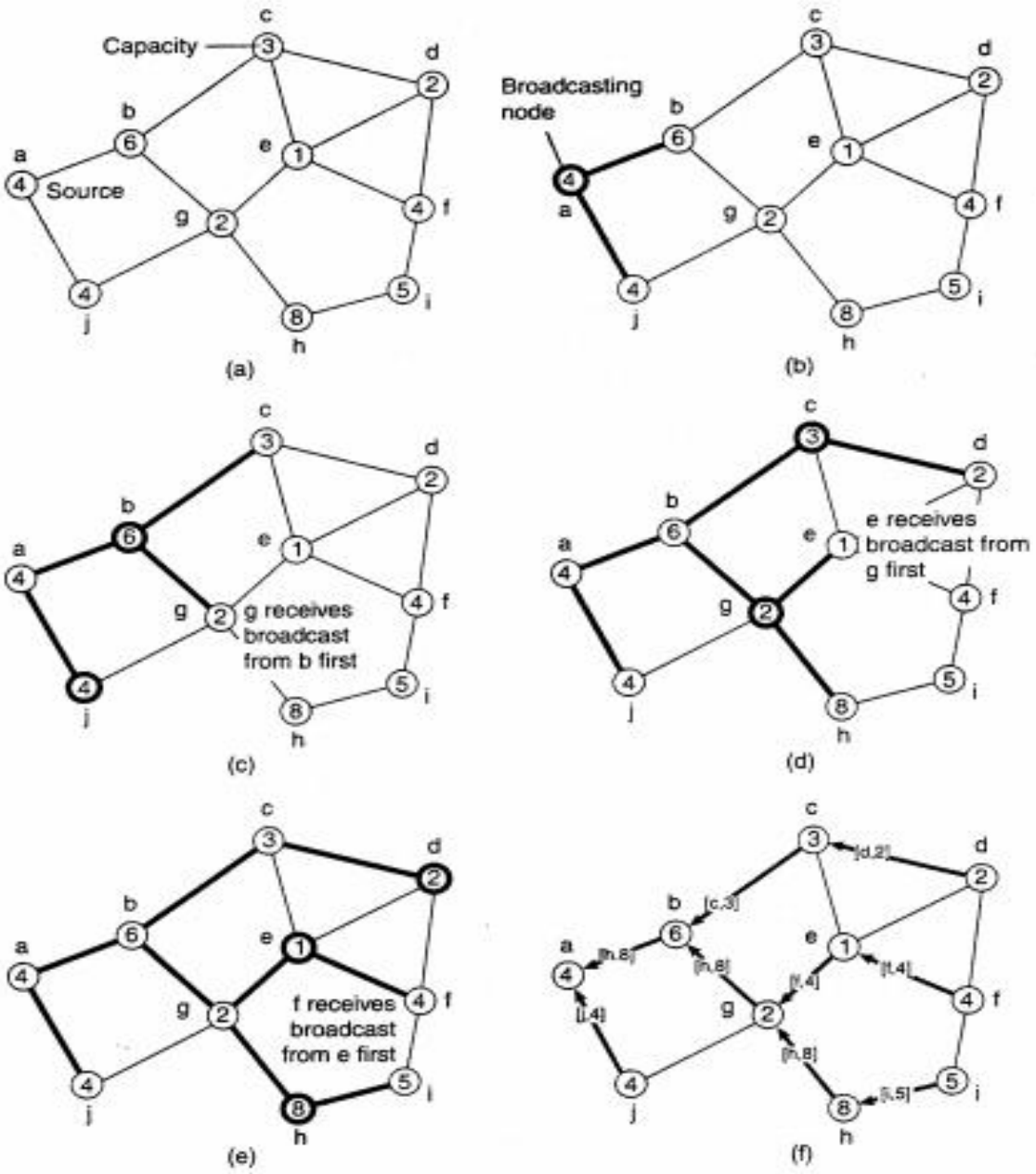


Figure 6-22. Election algorithm in a wireless network, with node a as the source. (a) Initial network., (b)-(e) The build-tree phase (last broadcast step by nodes f and i not shown). (f) Reporting of best node to source.

Borrowed from Distributed systems and paradigms 2nd Edition by A.Tanenbaum

Mutual Exclusion

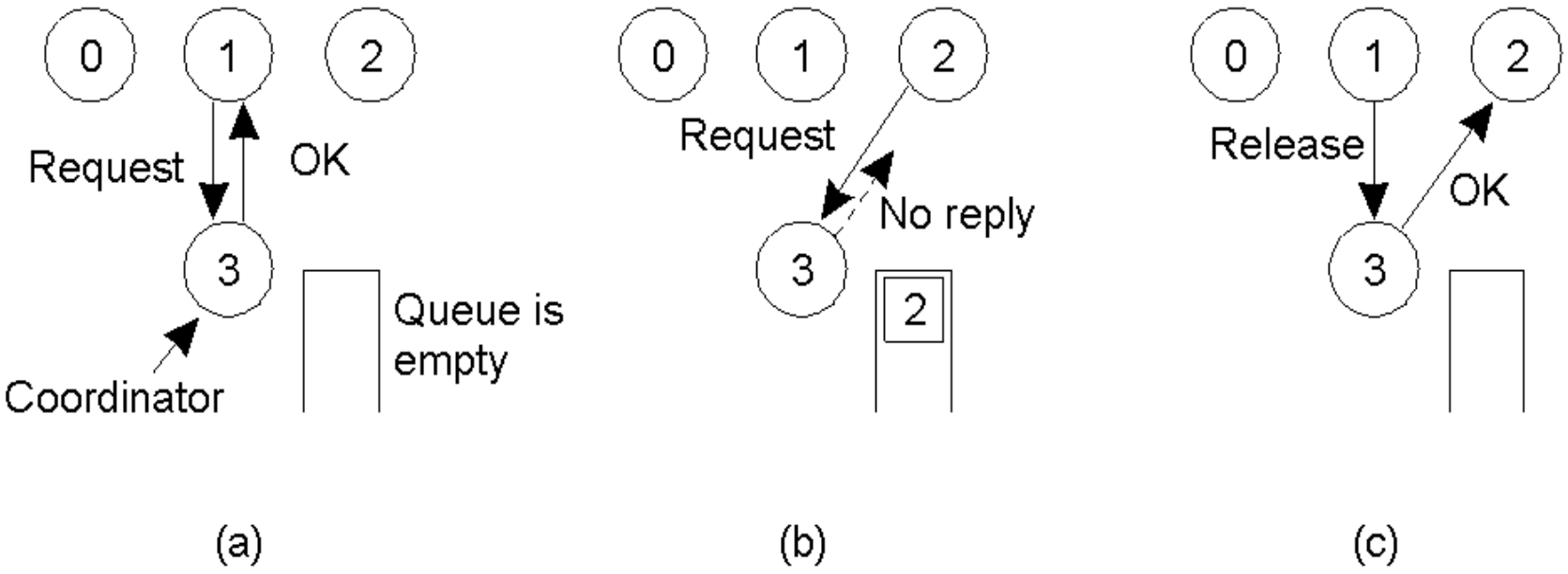
Mutual Exclusion: Introduction

- Mutual exclusion is needed for restricting access to a shared resource.
- We use semaphores, monitors and similar constructs to enforce mutual exclusion on a centralized system.
- We need the same capabilities on DS.
- As in the one processor case, we are interested in safety (mutual exclusion), progress, and bounded waiting (fairness).
- Assume n processes.

Mutual Exclusion: Centralized Algorithm

- Assume a coordinator has been elected.
  - A process sends a message to the coordinator requesting permission to enter a critical section. If no other process is in the critical section, permission is granted.
  - If another process then asks permission to enter the same critical region, the coordinator does not reply (Or, it sends "permission denied") and queues the request
  - When a process exits the critical section, it sends a message to the coordinator.
  - The coordinator takes first entry off the queue and sends that process a message granting permission to enter the critical section.

Mutual Exclusion: Centralized Algorithm



a) Process 1 asks the coordinator for permission to enter a critical region. Permission is granted

b) Process 2 then asks permission to enter the same critical region. The coordinator does not reply.

c) When process 1 exits the critical region, it tells the coordinator, which then replies to 2

Centralized Algorithm, Properties

- Simulates centralized lock using blocking calls
- Fair: requests are granted the lock in the order they were received
- Simple: three messages per use of a critical section (request, grant, release)
- Shortcomings:
  - Single point of failure
  - How do you detect a dead coordinator?
    - A process can not distinguish between "lock in use" from a dead coordinator
      - No response from coordinator in either case
  - Performance bottleneck in large distributed systems

Mutual Exclusion:Distributed Algorithm

- *Ricart and Agrawala algorithm (1981) assumes there is a mechanism for "totally ordering of all events" in the system (e.g. Lamport's algorithm) and a reliable message system.*

1. A process wanting to enter critical sections (cs) sends a message with *(cs name, process id, current time)* to all processes (including itself).
2. When a process receives a cs request from another process, it reacts based on its current state with respect to the cs requested. There are three possible cases:

Ricart and Agrawala algorithm

a) If the receiver is not in the cs and it does not want to enter the cs, it sends an OK message to the sender.

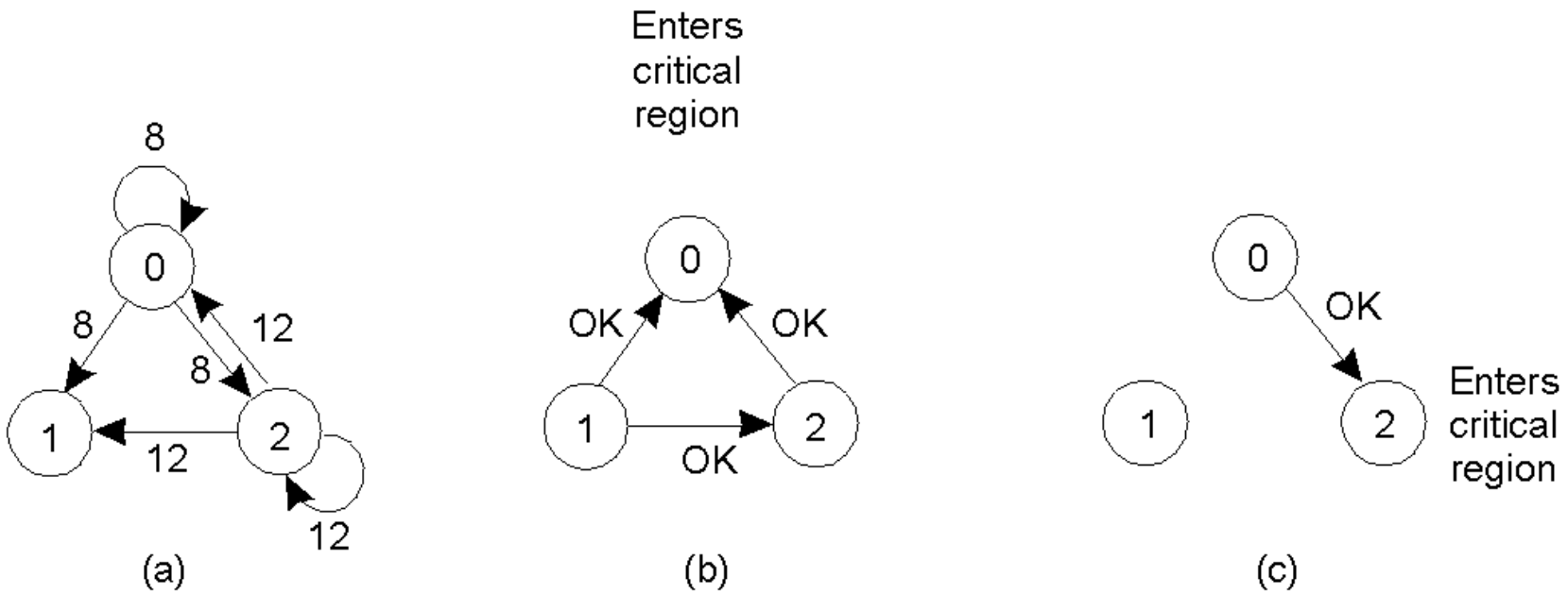
b) If the receiver is in the cs, it does not reply and queues the request.

c) If the receiver wants to enter the cs but has not yet, it compares the timestamp of the incoming message with the timestamp of its message sent to everyone. *{The lowest timestamp wins.}* If the incoming timestamp is lower, the receiver sends an OK message to the sender. If its own timestamp is lower, the receiver queues the request and sends nothing.

Mutual Exclusion: Distributed Algorithm

- After a process sends out a request to enter a cs, it waits for an OK from all the other processes. When all are received, it enters the cs.
- Upon exiting cs, it sends OK messages to all processes on its queue for that cs and deletes them from the queue.

Mutual Exclusion: Distributed Algorithm



a) Two processes want to enter the same critical region at the same moment.

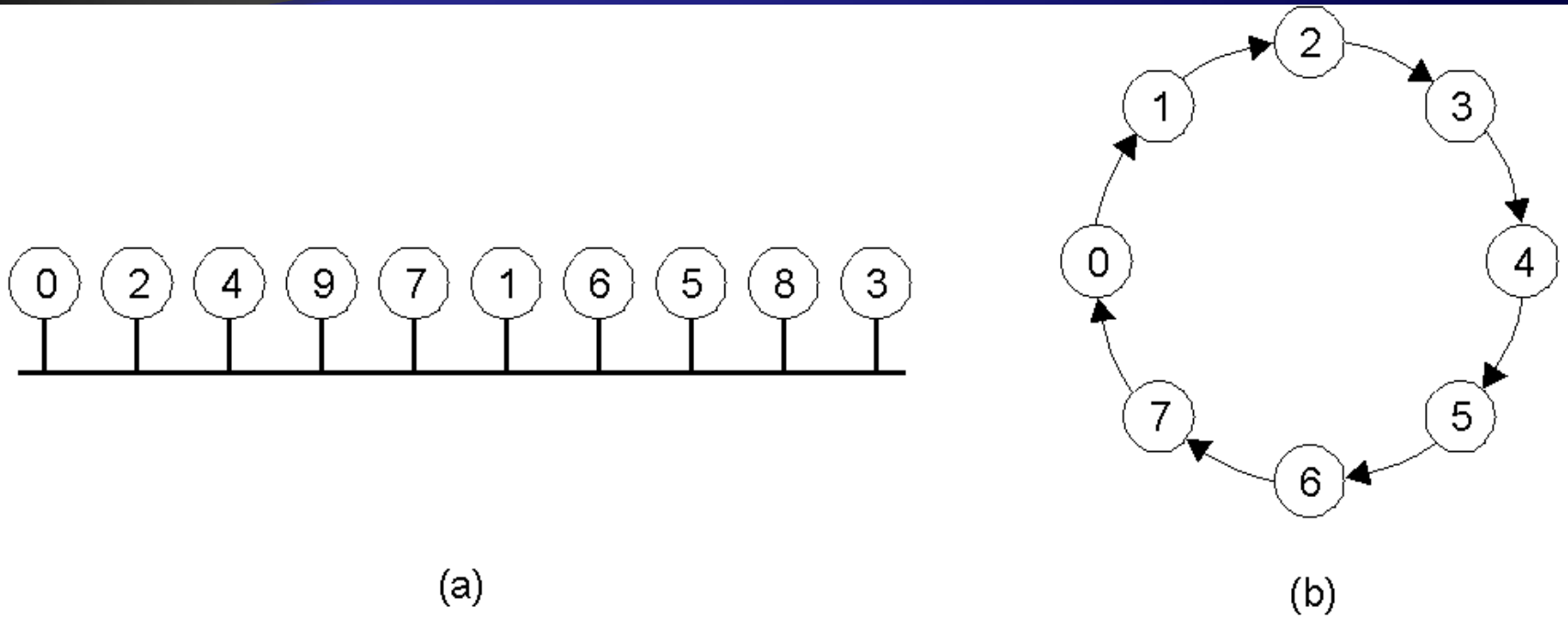
b) Process 0 has the lowest timestamp, so it wins.

c) When process 0 is done, it sends an OK also, so 2 can now enter the critical region.

Mutual Exclusion: Distributed Algorithm, Properties

- Fully decentralized
- *N* points of failure!
- All processes are involved in all decisions
  - Any overloaded process can become a bottleneck

Mutual Exclusion: Token Ring Algorithm



a) An unordered group of processes on a network.

b) A logical ring constructed in software.

- A process must have token to enter.

Mutual Exclusion: Token Ring Algorithm

- Assumptions:
- Processes are ordered in a ring.
- Communications are reliable and can be limited to one direction.
- Size of ring can be unknown and each process is only required to know his immediate neighbor.
- A single token circulates around the ring (in one direction only).

Mutual Exclusion: Token Ring Algorithm

- When a process has the token, it can enter the CR at most once.
  - Then it must pass the token on.
- Only the process with the token can enter the CR, thus Mutual Exclusion is ensured.
- Bounded waiting since the token circulates.
- Problems occur: If a token is lost, it must be regenerated
  - A process crashes – easier recovery using acks after receiving token

Mutual Exclusion: Algorithm Comparison

Algorithm	Messages per entry/exit	Delay before entry (in message times)	Problems
Centralized	3	2	Coordinator crash
Distributed	2 (n – 1)	2 (n – 1)	Process crash
Token ring	1 to ∞	0 to n – 1	Lost token, process crash

- Centralized is the most efficient.
- Token ring efficient when many want to use critical region.