Parallel and Distributed Computing

Dr. Ali Sayyed
Department of Computer Science
National University of Computer & Emerging Sciences

Synchronization and Coordination

Mutual exclusion

- Fundamental to distributed systems is the concurrency and collaboration among multiple processes.

- In many cases, this also means that processes will need to simultaneously access the same resources.

- **To prevent that such concurrent accesses, corrupt the resource, or make it inconsistent, solutions are needed to grant mutual exclusive access by processes.**

Mutual Exclusion in Parallel & Distributed Systems

- **Parallel Computing**
  - Shared memory model.
  - Uses synchronization primitives like **locks**, **semaphores**, and **monitors**.

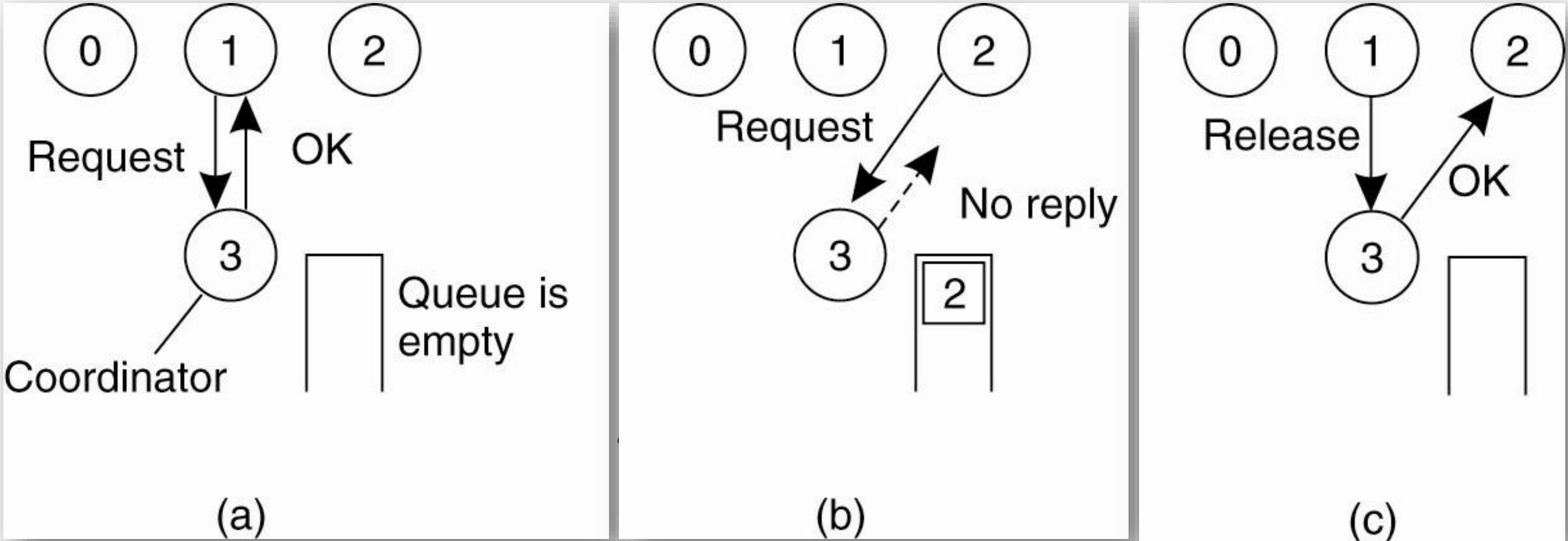
- **Distributed Computing**
  - No shared memory; processes communicate via messages.
  - Requires **distributed algorithms** for coordination.

Mutual Exclusion - A Centralized Algorithm

(a) Process 1 asks the coordinator for permission to access a hared resource. Permission is granted.

(b) Process 2 then asks permission to access the same resource. The coordinator does not reply.

(c) When process 1 releases the resource, it tells the coordinator, which then replies to 2.



Mutual Exclusion - A Distributed Algorithm

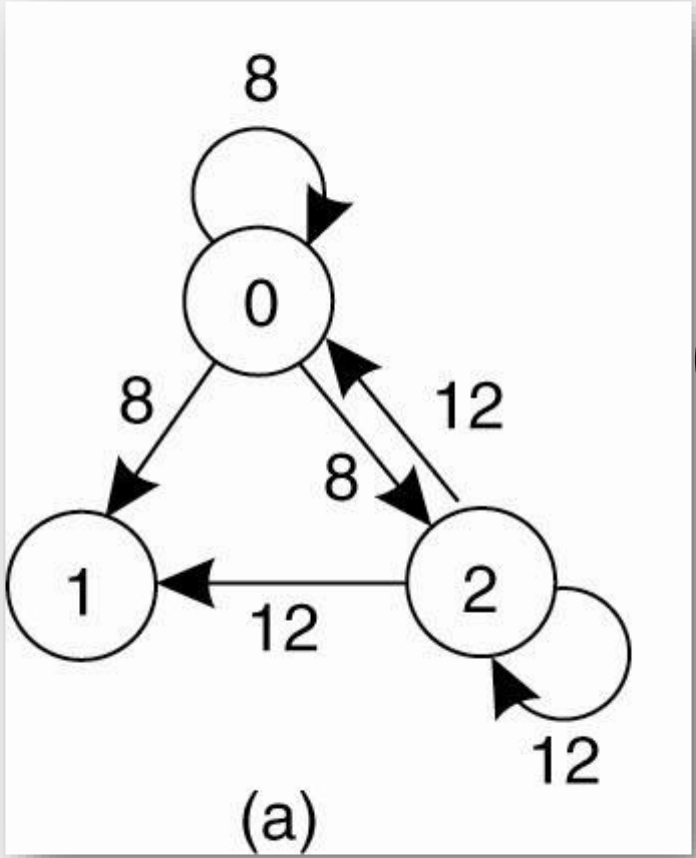
- When a process wants to access a shared resource, it **builds a message** containing the **name** of the resource, its **process number**, and the current **(logical) time**. It then sends the message to all other processes, conceptually including itself.
- When a process **receives a request message** from another process, **three different cases** must be clearly distinguished:
  1. If the receiver is not accessing the resource and does not want to access it, it sends back an **OK** message to the sender.

  2. If the receiver already has access to the resource, it simply **does not reply**. Instead, it queues the request.

  3. If the receiver wants to access the resource as well but has not yet done so, it compares the timestamp of the incoming message with the one contained in the message that it has sent everyone. The lowest one wins.

Mutual Exclusion – A Distributed Algorithm

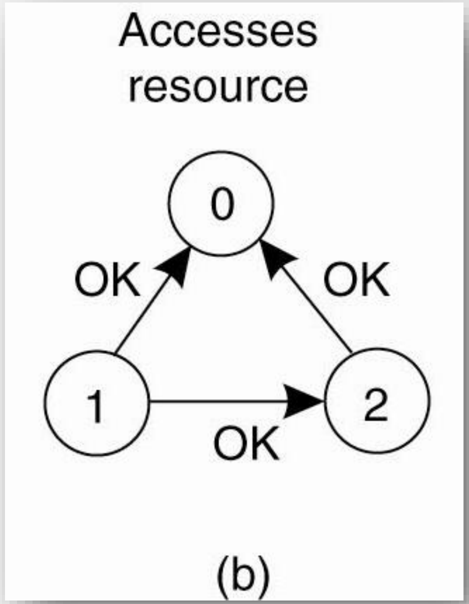
- **(a) Two processes want to access a shared resource at the same moment.**
- Process P0 sends everyone a request with timestamp 8, while at the same time, process P2 sends everyone a request with timestamp 12.
- P1 is not interested in the resource, so it sends OK to both senders.
- Processes P0 and P2 both see the conflict and compare timestamps.
- P2 sees that it has lost, so it grants permission to P0 by sending OK.

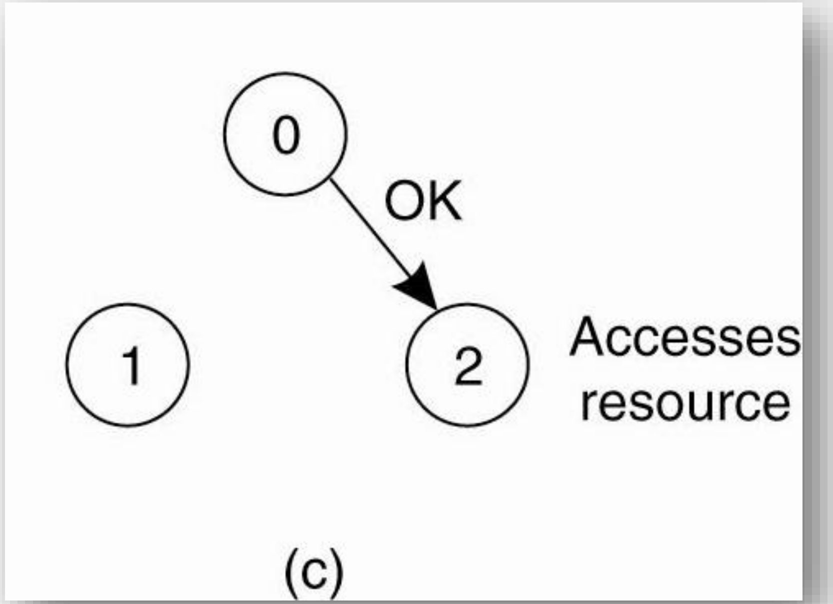


Mutual Exclusion – A Distributed Algorithm

(b) Process P0 now queues the request from P2 for later processing and accesses the resource
(c) When P0 is finished, it removes the request from P2 from its queue and sends an OK message to P2, allowing it to go ahead.

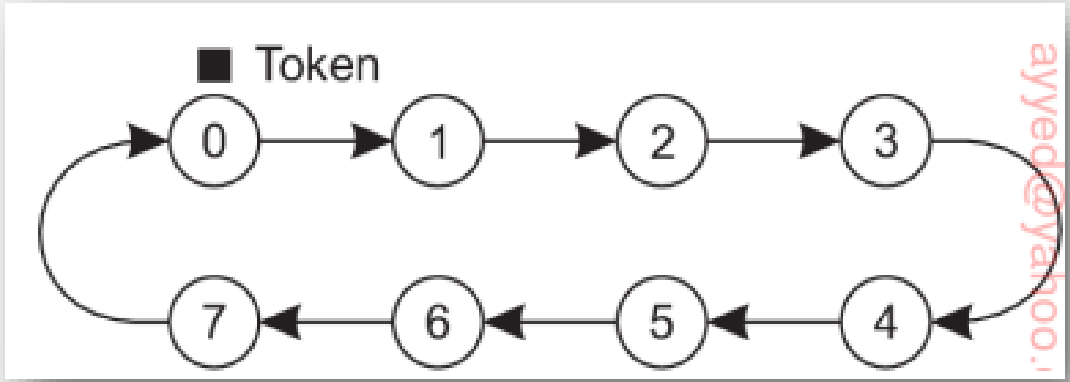
With this algorithm, mutual exclusion is guaranteed without deadlock or starvation.





Mutual Exclusion - A Token Ring Algorithm

- A completely different approach to **deterministically achieving mutual exclusion** in a distributed system is the Token Ring Algorithm.
- In software, we construct an **overlay network** in the form of a **logical ring** in which each process is assigned a position in the ring. All that matters is that each process knows who is next in line after itself.
- When the ring is initialized, process P0 is given a token. The token circulates around the ring.
- If a process is not interested in the resource, it just passes the token along. As a consequence, when no processes need the resource, the token just circulates around the ring.



Election algorithms

- Many distributed algorithms require one process to act as **coordinator**, **initiator**, or otherwise perform **some special role**.

- In general, it does not matter which process takes on this special responsibility, but one of them has to do it.

- The goal of an election algorithm is to ensure that when an election starts, it concludes with all processes agreeing on who the new coordinator is to be.

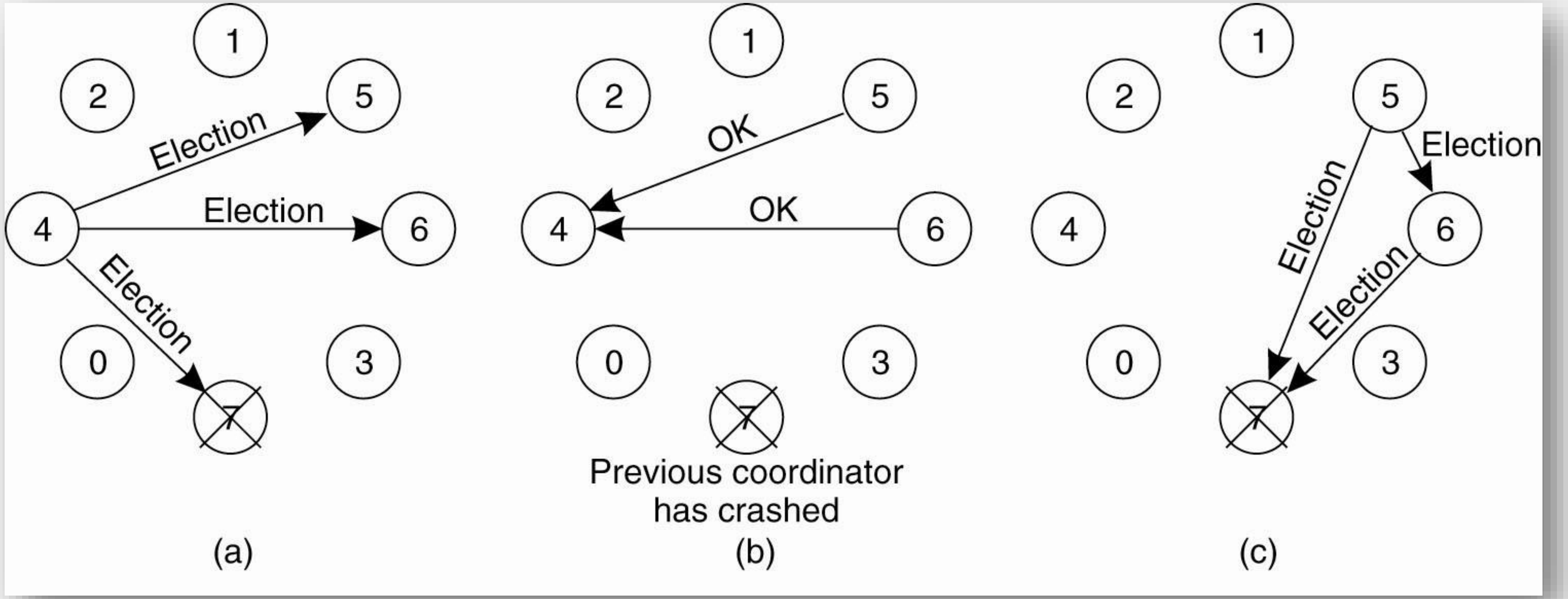
Election Algorithms - The bully algorithm

- A well-known solution for electing a coordinator is the bully algorithm devised by Garcia in 1982.
- Consider N processes $\{P_0, \ldots, P_{N-1}\}$ having an identifier $id(P_k) = k$. When any process notices that the coordinator is no longer responding to requests, it initiates an election. A process P_k, holds an election as follows:
    1. P_k sends an *ELECTION* message to all processes with higher identifier: $P_{k+1}, P_{k+2}, \ldots, P_{N-1}$.
    2. If no one responds, P_k wins the election and becomes coordinator.
    3. If one of the higher-ups answers, it takes over. P_k's job is done.

Thus the biggest guy in town always wins, hence the name “bully algorithm.”

Election Algorithms - The bully algorithm

a) Process 4 holds an election.
b) Processes 5 and 6 respond, telling 4 to stop.
c) Now 5 and 6 each hold an election.

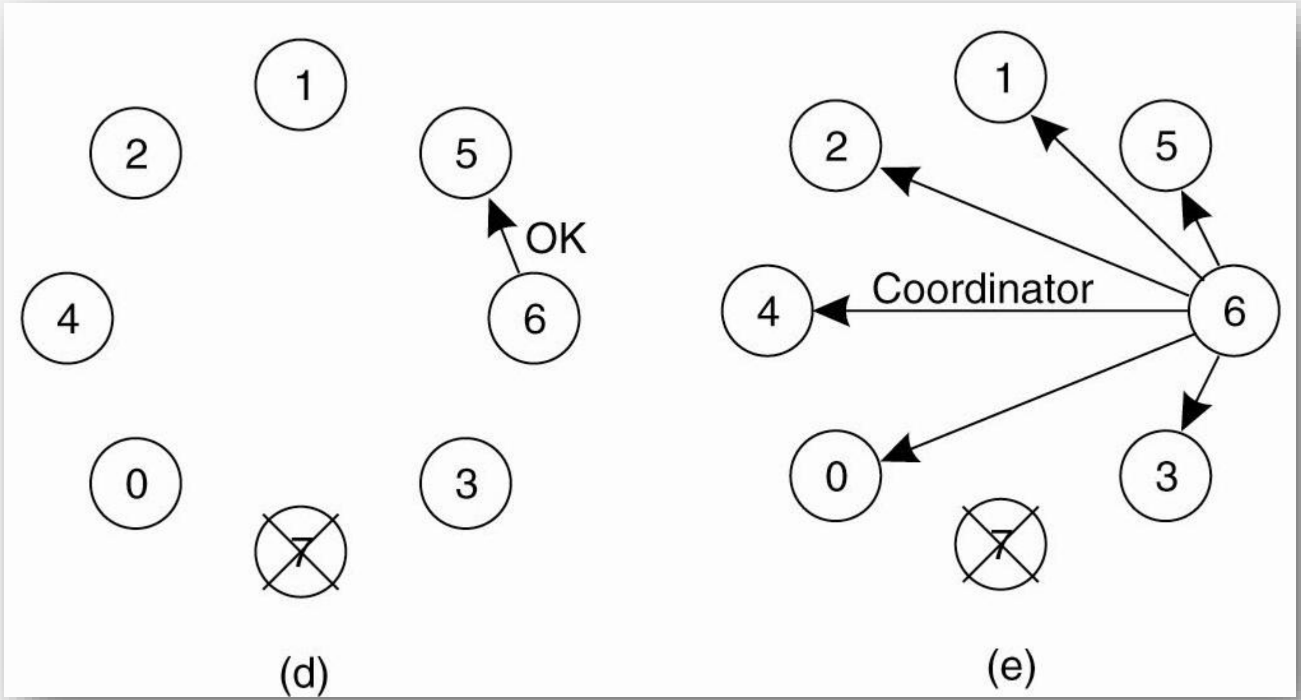


Election Algorithms - The bully algorithm

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

When P4 gets this message, it can now continue with the operation it was trying to do when it discovered that P7 was dead but using P6 as the coordinator this time.

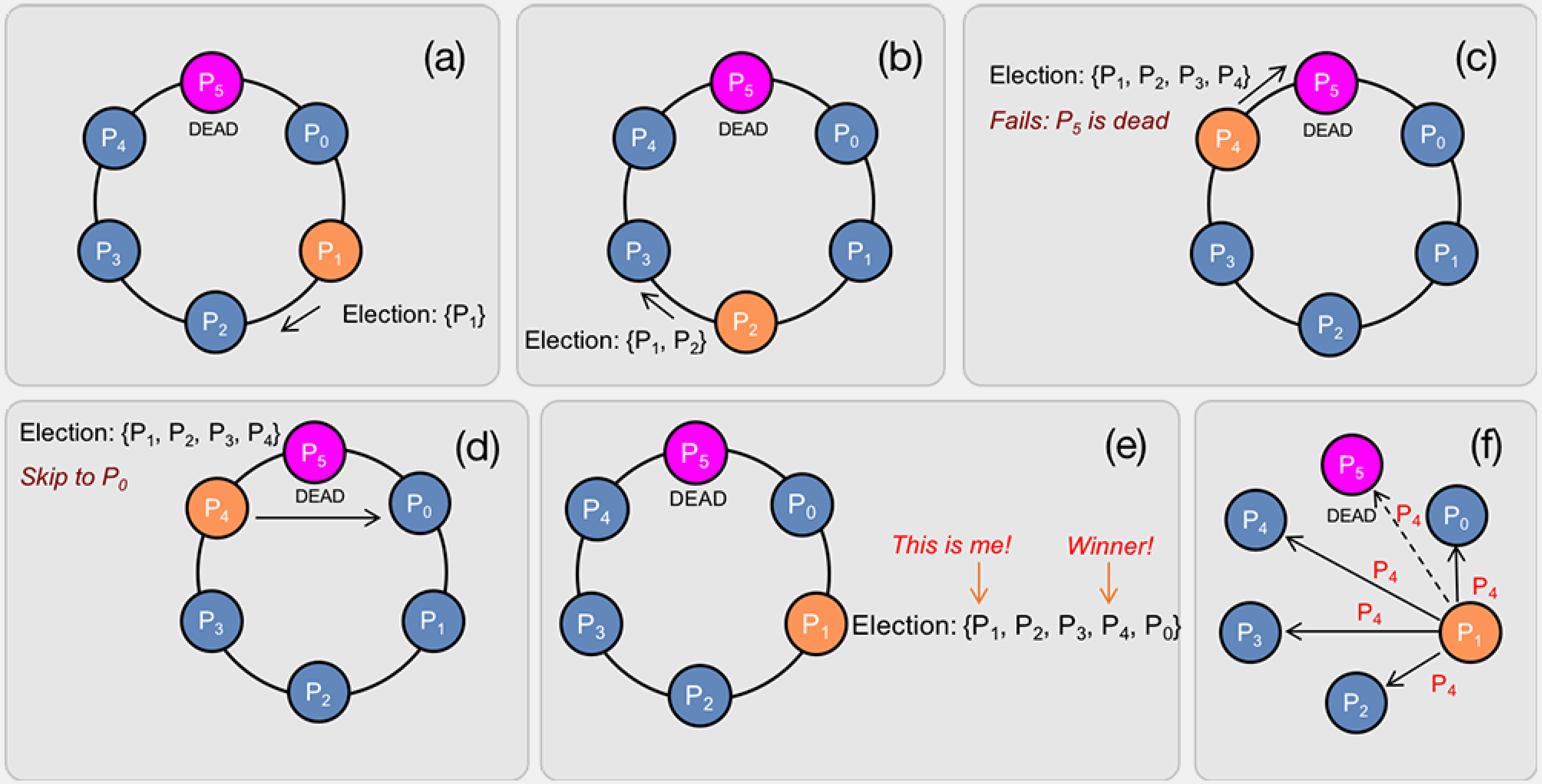
In this way the failure of P7 is handled and the work can continue.



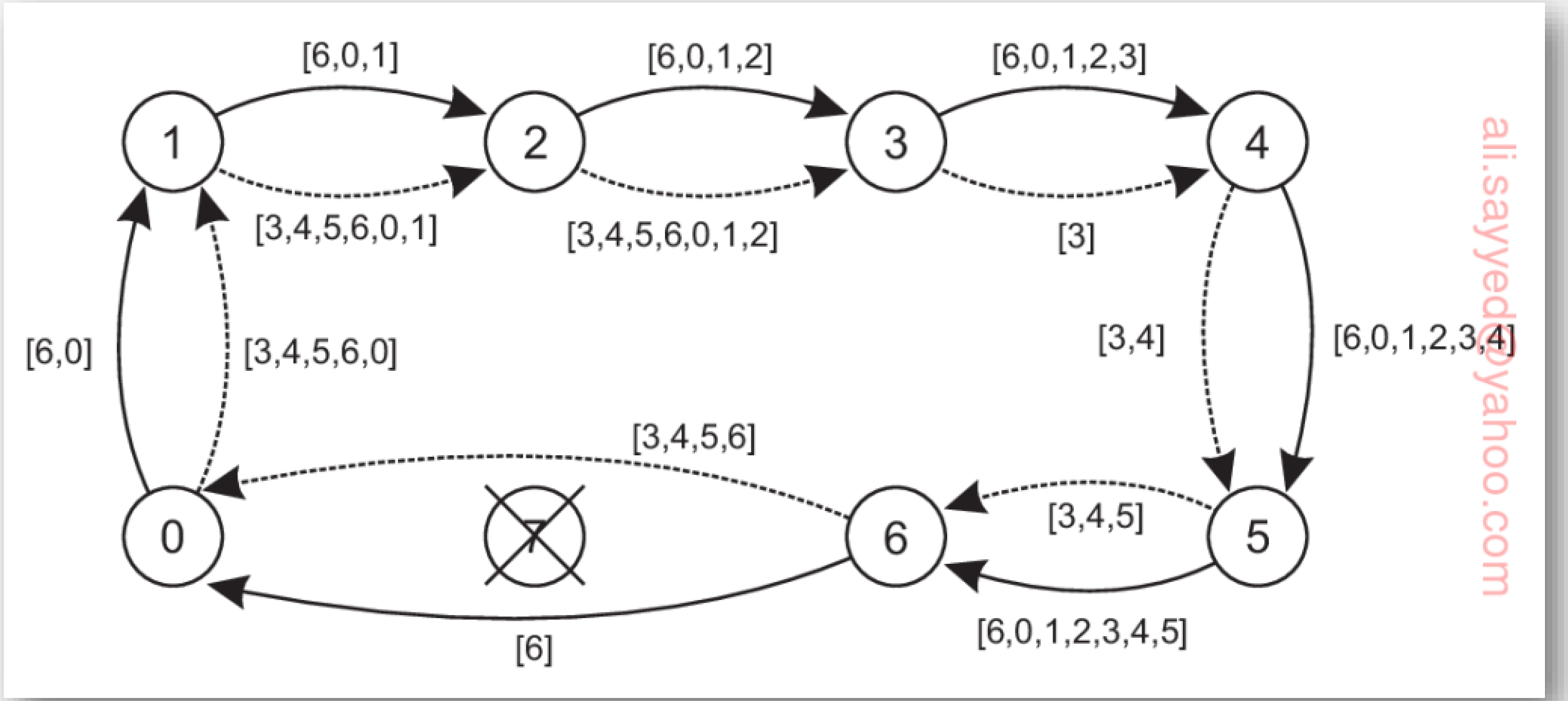
Election Algorithms – A Ring Algorithm

- This algorithm is based on the use of a (logical) ring. Unlike some ring algorithms, this one does not use a token.
- We assume that each process knows who its successor is.
- When any process notices that the coordinator is down, it builds an ELECTION message, adds its own process identifier 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, or the one after that, until a running process is located.
- Along the way, each sender adds its own identifier to the message.
- Eventually, the message gets back to the process that started it all.
- At that point, the message type is changed to COORDINATOR and circulated once again, this time to inform everyone else who the coordinator is (the list member with the highest identifier) and who the members of the new ring are.
- When this message has circulated once, it is removed, and everyone goes back to work.

Election Algorithms – A Ring Algorithm



Election Algorithms – A Ring Algorithm



[6,0,1]
[6,0,1,2]
[6,0,1,2,3]
1
2
3
4
[3,4,5,6,0,1]
[3,4,5,6,0,1,2]
[3]
[6,0]
[3,4,5,6,0]
[3,4]
[6,0,1,2,3,4]
[3,4,5,6]
0
7
6
5
[3,4,5]
[6]
[6,0,1,2,3,4,5]