Synchronization
Part 2

Outline – Part 2

- Clock Synchronization
- Clock Synchronization Algorithms
- Logical Clocks
- **Election Algorithms**
- Mutual Exclusion
- Distributed Transactions
- Concurrency Control

Election Algorithms

- Many distributed algorithms such as mutual exclusion and deadlock detection require a **coordinator process.**
- When the coordinator process fails, the distributed group of processes must execute an **election algorithm** to determine a new coordinator process.
- These algorithms will assume that each active process has a unique **priority id.**

The Bully Algorithm

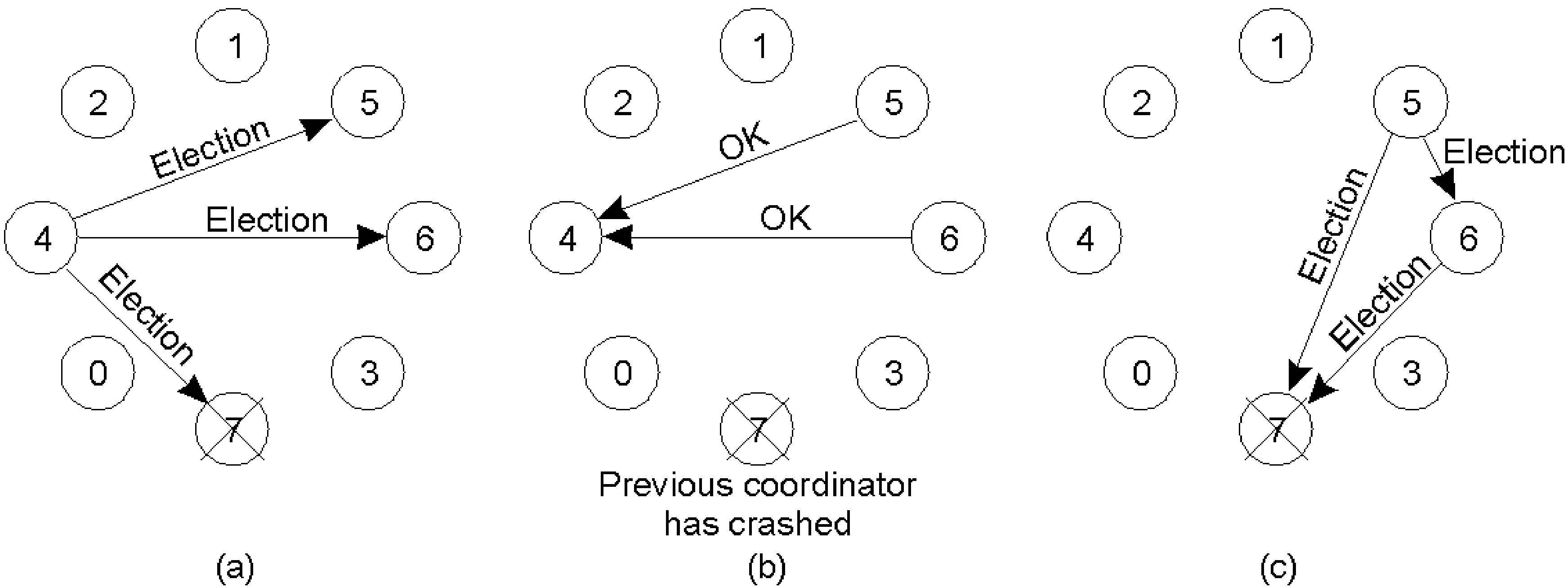
When any process, P, notices that the coordinator is no longer responding it initiates an election:

1. P sends an *election* message to all processes with higher id numbers.
2. If no one responds, P wins the election and becomes coordinator.
3. If a higher process responds, it takes over. Process P's job is done.

The Bully Algorithm

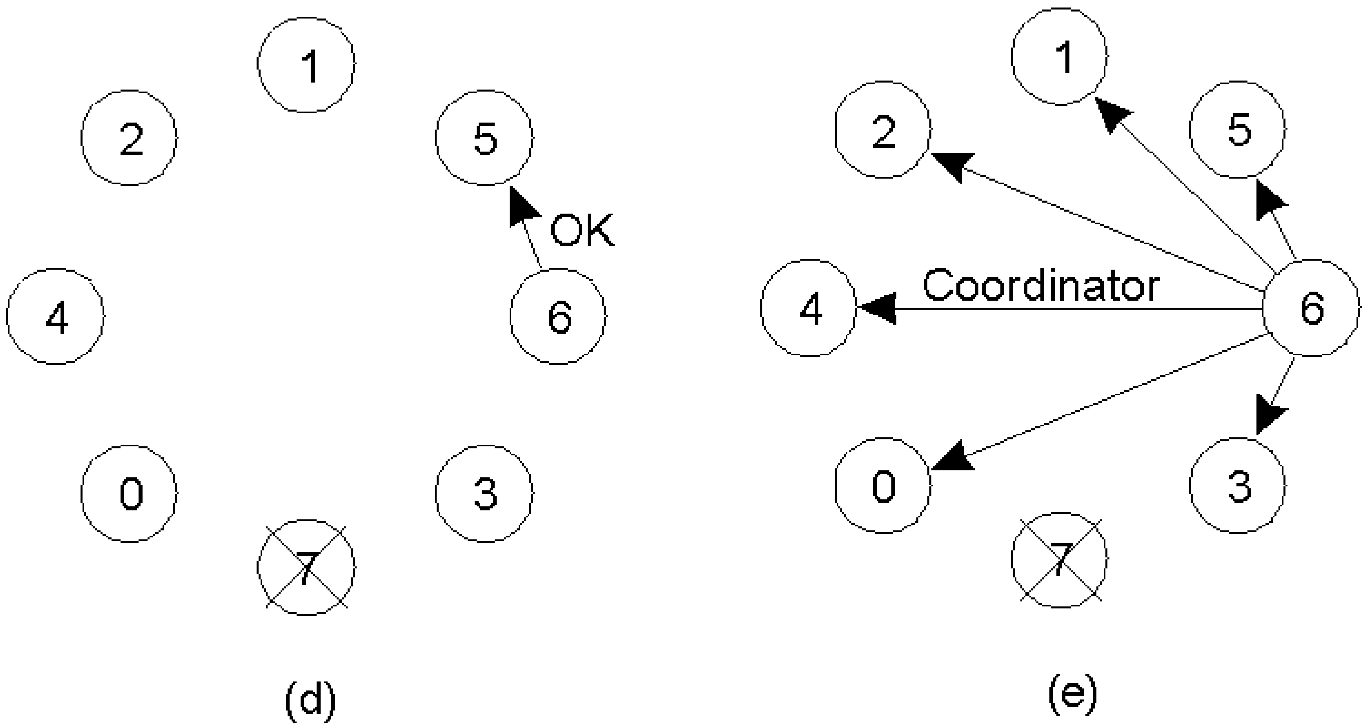
- At any moment, a process can receive an **election** message from one of its lower-numbered colleagues.
- The receiver sends an OK back to the sender and conducts its own election.
- Eventually only the bully process remains. The bully announces victory to all processes in the distributed group.

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.

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.

A Ring Algorithm

- Assume the processes are logically ordered in a ring **{implies a successor pointer and an active process list}** that is unidirectional.

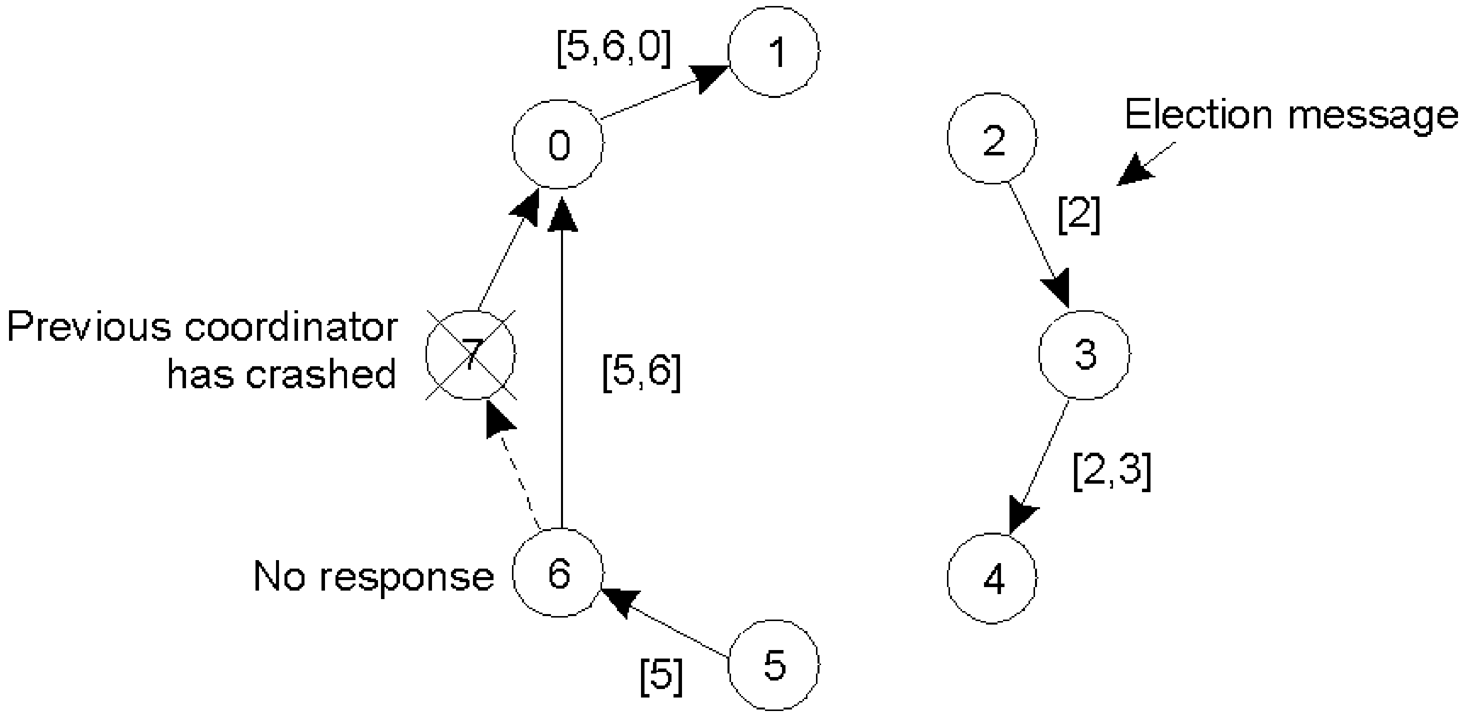
When any process, P, notices that the coordinator is no longer responding it initiates an election:

1. P sends message containing P's process id to the next available successor.

A Ring Algorithm

2. At each active process, the receiving process adds its process number to the list of processes in the message and forwards it to its successor.
3. Eventually, the message gets back to the sender.
4. The initial sender sends out a second message letting everyone know who the coordinator is {the process with the highest number} and indicating the current members of the active list of processes.

A Ring Algorithm



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

Outline – Part 2

- Clock Synchronization
- Clock Synchronization Algorithms
- Logical Clocks
- Election Algorithms
- **Mutual Exclusion**
- Distributed Transactions
- Concurrency Control

Mutual Exclusion

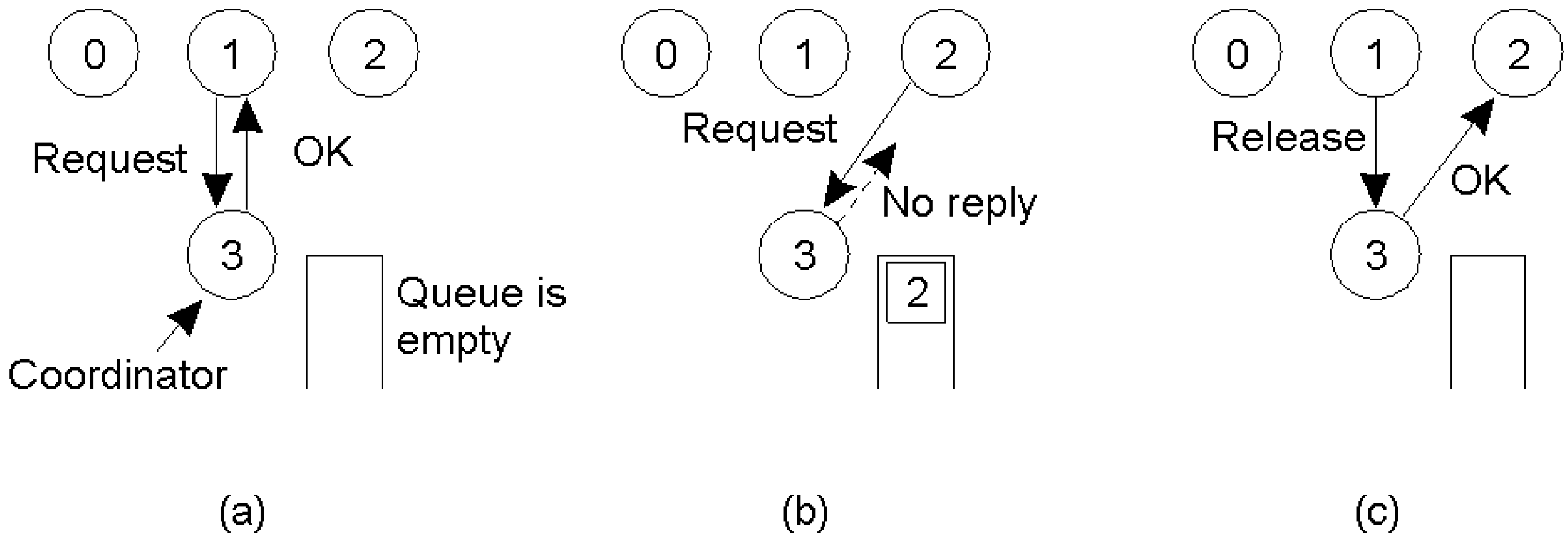
- To guarantee consistency among distributed processes that are accessing shared memory, it is necessary to provide **mutual exclusion** when accessing a critical section.
- Assume n processes.

A Centralized Algorithm for Mutual Exclusion

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.

A Centralized Algorithm for Mutual Exclusion



A Distributed Algorithm for Mutual Exclusion

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:

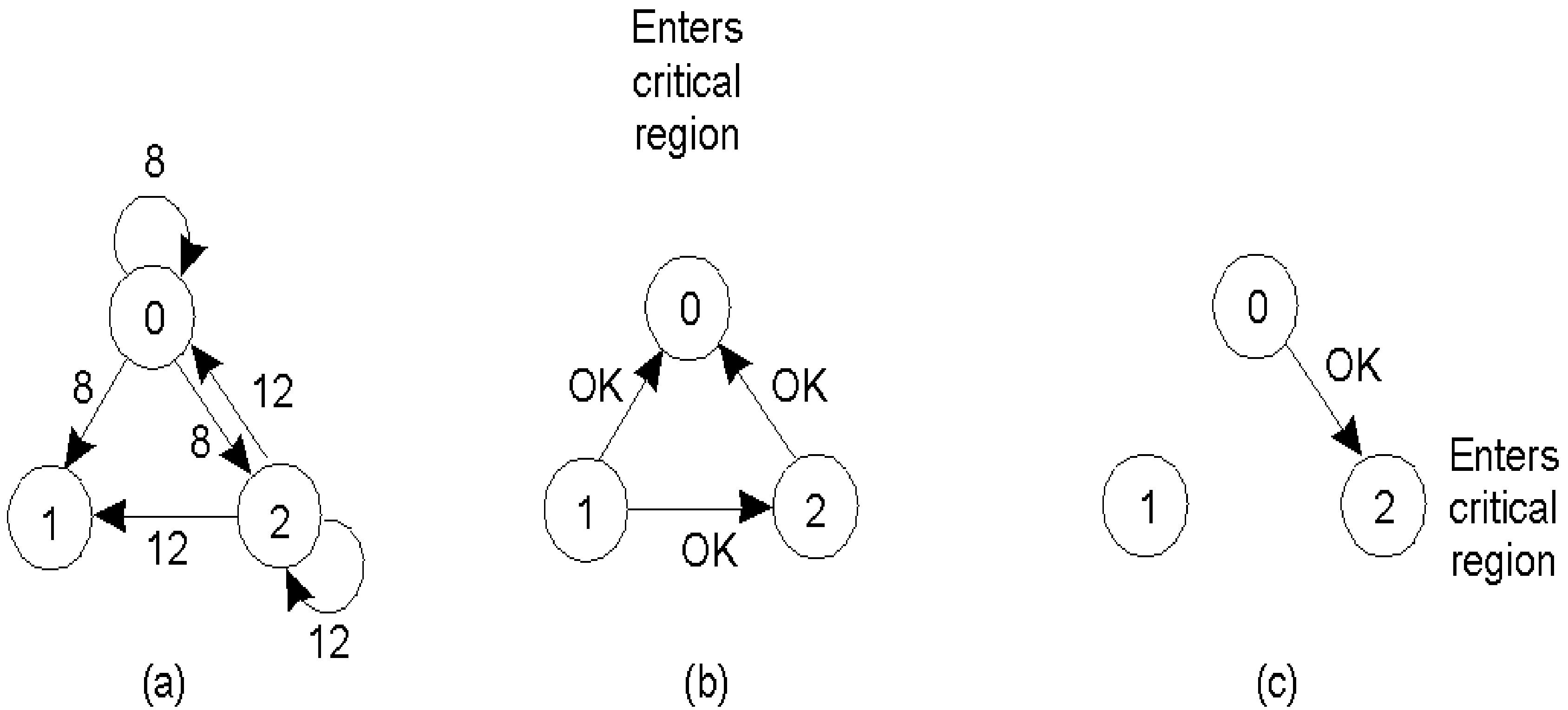
A Distributed Algorithm for Mutual Exclusion (cont.)

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.

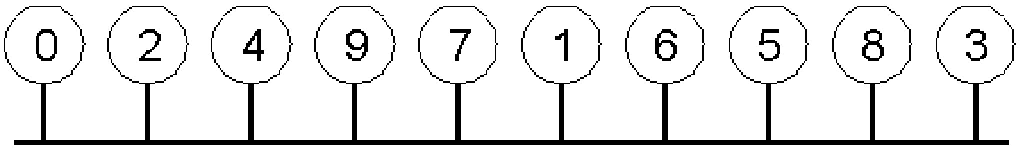
A Distributed Algorithm for Mutual Exclusion (cont.)

- 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.

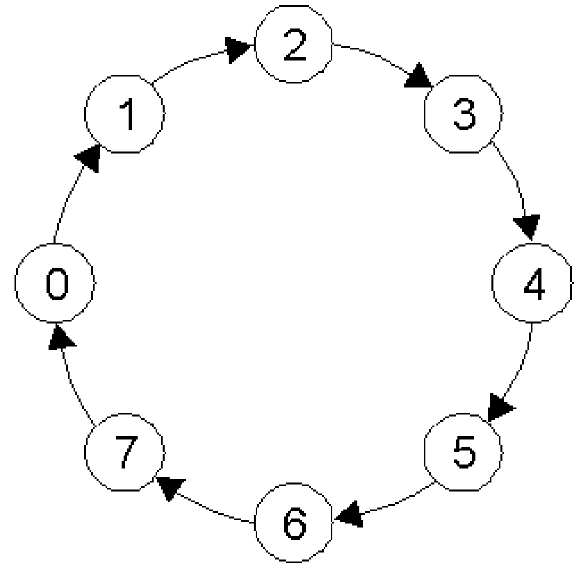
A Distributed Algorithm for Mutual Exclusion



A Token Ring Algorithm



(a)



(b)

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 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.

Outline – Part 2

- Clock Synchronization
- Clock Synchronization Algorithms
- Logical Clocks
- Election Algorithms
- Mutual Exclusion
- **Distributed Transactions**
- Concurrency Control