Election Algorithms

1. Introduction to Election Algorithms

- **Purpose**: In distributed systems, many algorithms need a specific process to take on a special role such as **coordinator**, **initiator**, or **sequencer**.
- Examples include the **coordinator in centralized mutual exclusion algorithms**.
- **Goal**: To select one process to take on the special role when all processes are running.

2. Problem Overview

- **Unique Identifiers**: Each process has a **unique number**, such as a **network address**. This makes it possible to distinguish between processes.
- The aim of election algorithms is to select the process with the **highest process number** to become the **coordinator**.

3. Assumptions

- **Each process knows the process numbers** of all other processes.
- However, processes do not know the current **status** (up or down) of other processes.

4. Goal of Election Algorithms

- To ensure that all processes **agree** on who the new coordinator is.
- When an election begins, it should end with a **consensus** on the new coordinator, regardless of any process failures.

The Bully Algorithm

1. Introduction

- The **Bully Algorithm** was created by **Garcia-Molina** in 1982.
- It is used in distributed systems when a coordinator (a special process) stops responding, and a new one needs to be elected.

2. Steps of the Bully Algorithm

When a process, say **P**, notices the coordinator is down, it follows these steps:

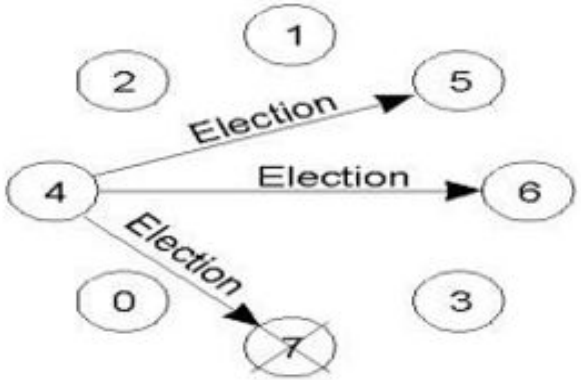
1. **P sends an ELECTION message** to all processes with **higher process numbers.**
2. If **no one responds**, **P wins the election** and becomes the new coordinator.
3. If a **higher-numbered process** responds, **P's job is done**; the higher process will take over the election.

3. Receiving an Election Message

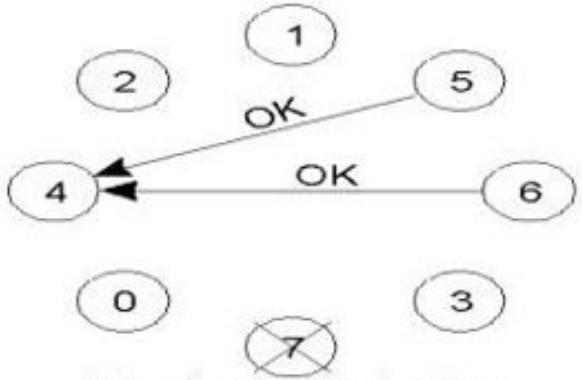
- If a process receives an **ELECTION message** from a lower-numbered process, it sends an **OK message** back, signaling that it is alive and will take over.
- The process that responded then **holds its own election** to find the highest-numbered process.

4. The Election Process

- Eventually, all lower-numbered processes give up, leaving one higher-numbered process as the **new coordinator**.
- This new coordinator sends a **COORDINATOR message** to all other processes, announcing that it is now in charge.

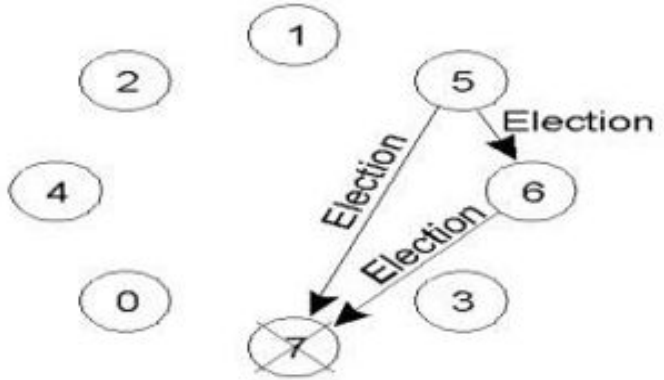


(a)

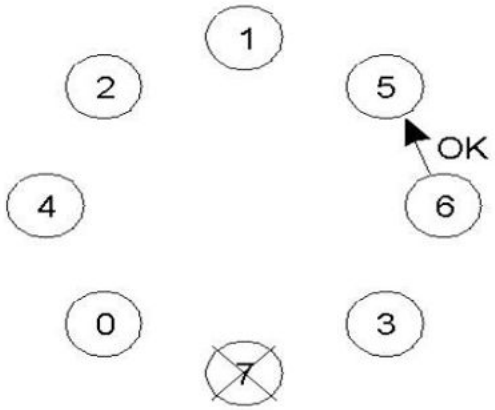


Previous coordinator has crashed

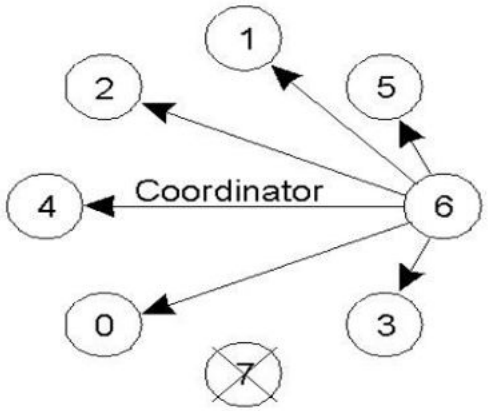
(b)



(c)



(d)



(e)

Imagine a system with eight processes numbered **0 to 7**:

- **Process 7** was the coordinator, but it has crashed.
- **Process 4** notices this first and sends an **ELECTION message** to **processes 5, 6, and 7.**
- **Processes 5 and 6 respond** with **OK messages**, so process 4 knows one of them will take over and its role is complete.
- Then, **processes 5 and 6 hold their own elections.** Eventually, **process 6 wins** and becomes the new coordinator.
- **Process 6 sends a COORDINATOR message** to all processes, announcing that it is the new coordinator.

6. Rejoining After a Crash

- If **process 7** (which crashed) restarts, it will send a **COORDINATOR message** to all other processes, using its higher process number to take back the coordinator role. This is why it's called the **Bully Algorithm**—the highest-numbered process always takes over.

In summary, the Bully Algorithm ensures that the highest-numbered process becomes the coordinator and allows smooth recovery when the coordinator crashes.

The Ring Algorithm

1. Introduction

- The **Ring Algorithm** is another method for electing a coordinator in a distributed system.
- Unlike the Bully Algorithm, this one uses a **ring structure** to organize the processes.

2. Assumptions

- The processes are arranged in a **logical or physical ring**, meaning each process knows who its **successor** is.
- There is **no token** involved; instead, the election is based on message passing.

3. Steps of the Ring Algorithm

When a process notices the coordinator is down:

1. It creates an **ELECTION message** containing its **own process number**.
2. It sends the message to its **successor**.
    - If the successor is down, the process skips to the **next available running process** in the ring.
3. Each process that receives the ELECTION message **adds its process number** to the message and passes it along.

4. Detecting the New Coordinator

- The ELECTION message travels around the ring and eventually returns to the **originating process**.
- When the original process sees its own number in the message, it recognizes that the election is complete.
- At this point, it converts the ELECTION message into a **COORDINATOR message**, identifying the process with the **highest number** as the new coordinator.

5. Informing All Processes

- The **COORDINATOR message** is circulated around the ring, informing all processes about the new coordinator and the current members of the ring.
- Once the message has completed one full cycle, it is **removed** and normal operation resumes.

6. Simultaneous Elections

- If two processes (like **processes 2 and 5**) detect that the coordinator has crashed at the same time, they may each start their own ELECTION messages.
- Both messages will travel around the ring, gathering process numbers and converting into **COORDINATOR messages**.
- Although both messages contain the same information and follow the same order, there is no harm in this duplication. The extra messages will eventually be removed, and normal operation will resume.

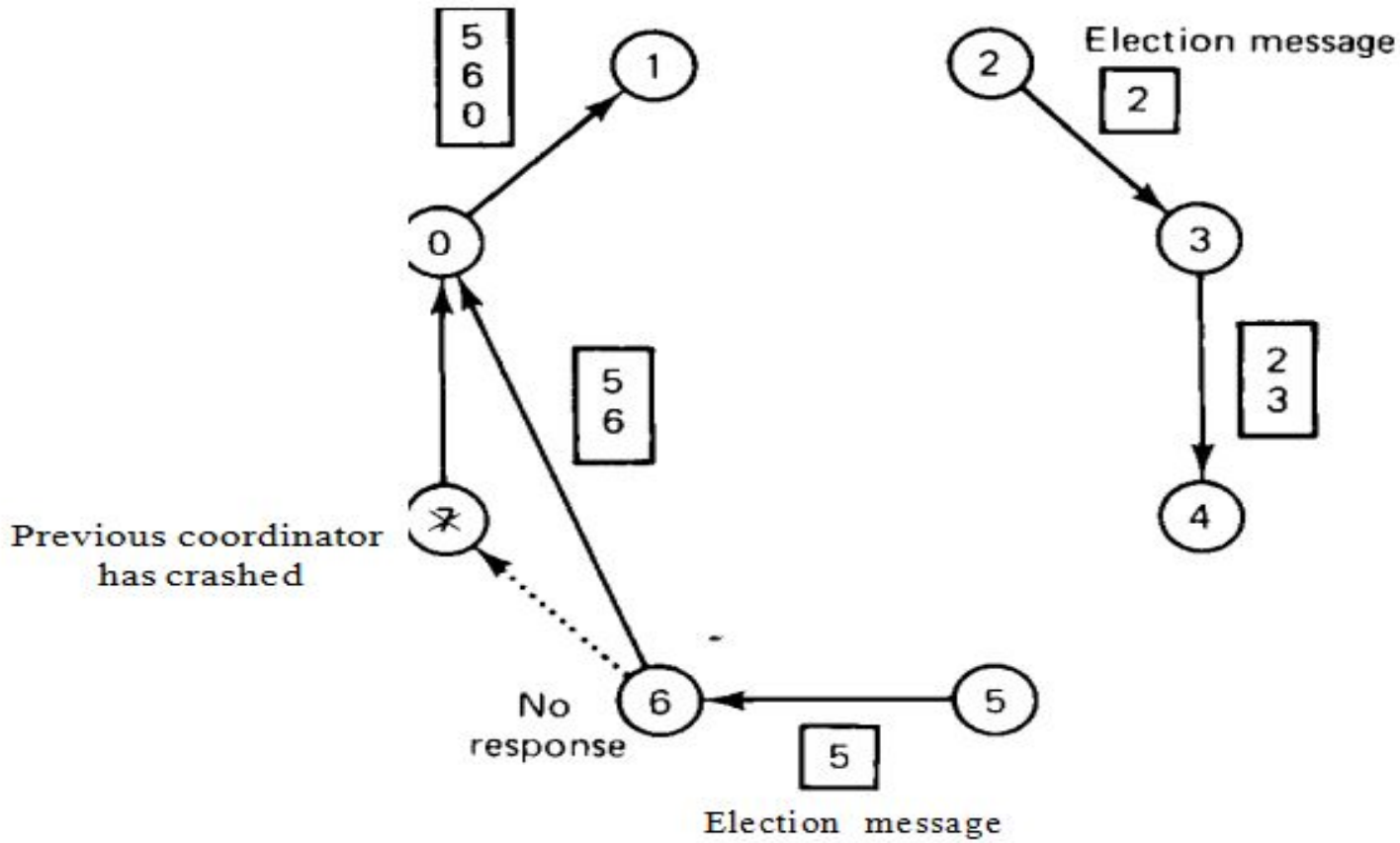


Fig. 3-13. Election algorithm using a ring.

In Fig. 3-13 we see what happens if two processes, 2 and 5, discover simultaneously that the previous coordinator, process 7, has crashed. Each of these builds an *ELECTION* message and starts circulating it. Eventually, both messages will go all the way around, and both 2 and 5 will convert them into *COORDINATOR* messages, with exactly the same members and in the same order. When both have gone around again, both will be removed. It does no harm to have extra messages circulating; at most it wastes a little bandwidth.