Election Algorithms

Module 3

Need for a Coordinator

- Many distributed algorithms need one process to act as coordinator – Doesn't matter which process does the job, just need to pick one.
- For example,
  - see the centralized mutual exclusion algorithm.
  - Clock synchronization algorithms
- In general, all processes in the distributed system are equally suitable for the role
- Election algorithms are designed to choose a coordinator.

Election Algorithms

- If we are using one process as a coordinator for a shared resource …
- …how do we select that one process?

Solution – an *Election*

- All processes currently involved get together to *choose* a coordinator
- If the coordinator crashes or becomes isolated, elect a new coordinator
- If a previously crashed or isolated process, comes on line, a new election *may* have to be held

Election Algorithms

- Any process can serve as coordinator
- Any process can “call an election”
- (initiate the algorithm to choose a new coordinator).
- There is no harm (other than extra message traffic) in having multiple concurrent elections.
- Elections may be needed when the system is initialized, or if the coordinator crashes or retires.

Assumptions

- Every process/site has a unique ID; e.g.
  - **the network address**
  - **a process number**
- Every process in the system should know the values in the set of ID numbers, although not which processors are up or down.
- The process with the highest ID number will be the new coordinator.

Requirements

- When the election algorithm terminates a single process has been selected and every process knows its identity.
- Formalize: every process p_i has a variable e_i to hold the coordinator's process number.
  - $\forall i$, e_i = undefined or e_i = P, where P is the non-crashed process with highest id
  - All processes (that have not crashed) eventually set e_i = P.

Election Algorithms

- Wired systems
  - Bully algorithm
  - Ring algorithm

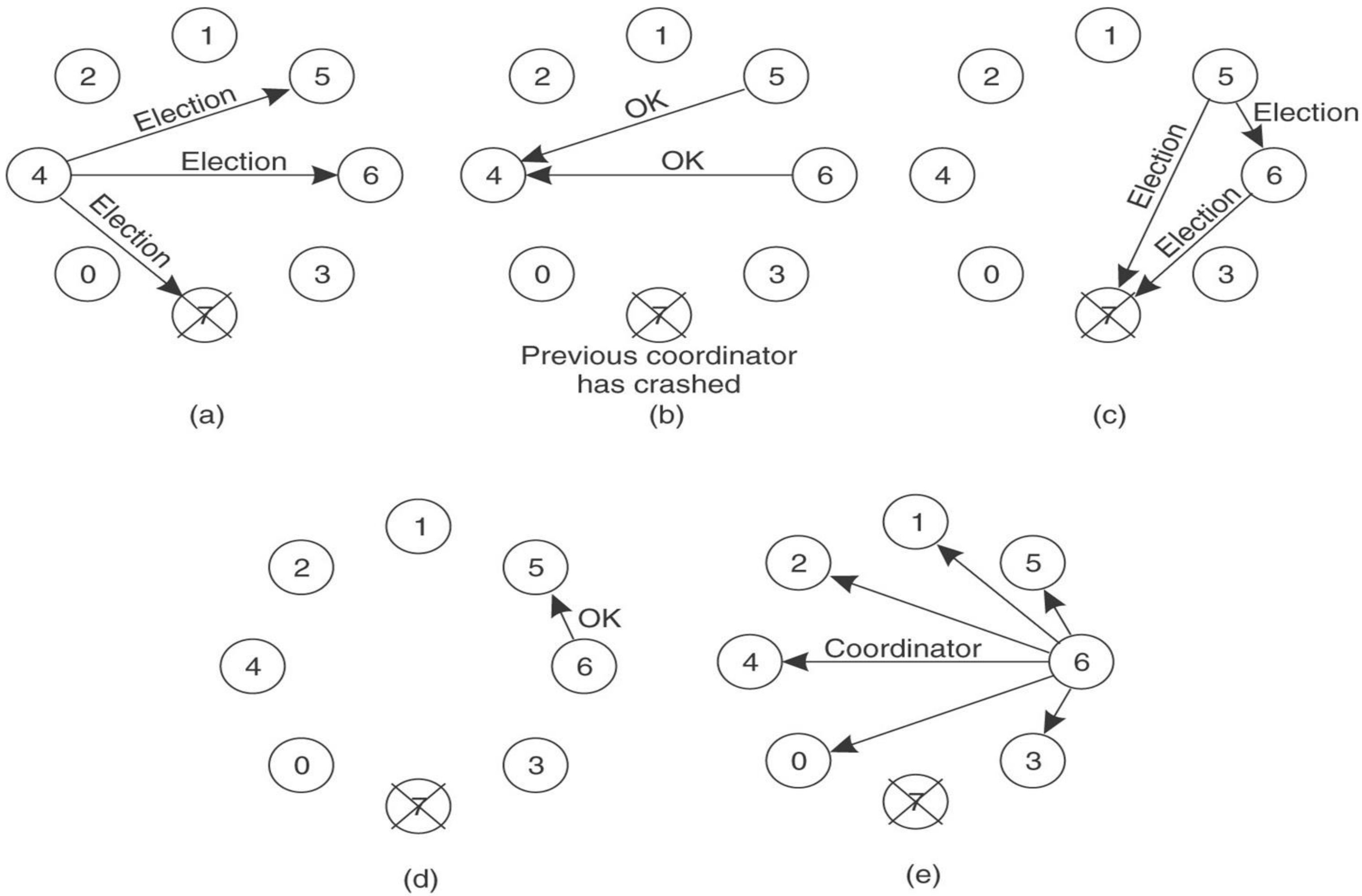
Bully Algorithm

- **Key Idea: select process with highest ID**
- Assume
  - All processes know about each other
  - Processes numbered uniquely
  - They do not know each other's state
- Suppose *P* notices no coordinator
  - Sends *election message* to all higher numbered processes
  - If no response, *P* takes over as coordinator
  - If any responds, *P* yields

Bully Algorithm (continued)

- Suppose *Q* receives *election message*
  - Replies *OK* to sender, saying it will take over
  - Sends a new *election message* to higher numbered processes
- Repeat until only one process left standing
  - Announces victory by sending message saying that it is the coordinator
- Three types of messages
  - Election.
  - OK
  - Coordinator

Bully Algorithm (continued)



Bully Algorithm (continued)

- Suppose *R* comes back on line
  - Sends a new *election message* to higher numbered processes
- Repeat until only one process left standing
  - Announces victory by sending message saying that it is the coordinator (if not already the coordinator)
- Existing (lower numbered) coordinator yields
  - Hence the term “bully”

Analysis

- Works best if communication in the system has bounded latency so processes can determine that a process has failed by knowing the upper bound (UB) on message transmission time (T) and message processing time (M).
  - UB = 2 * T + M

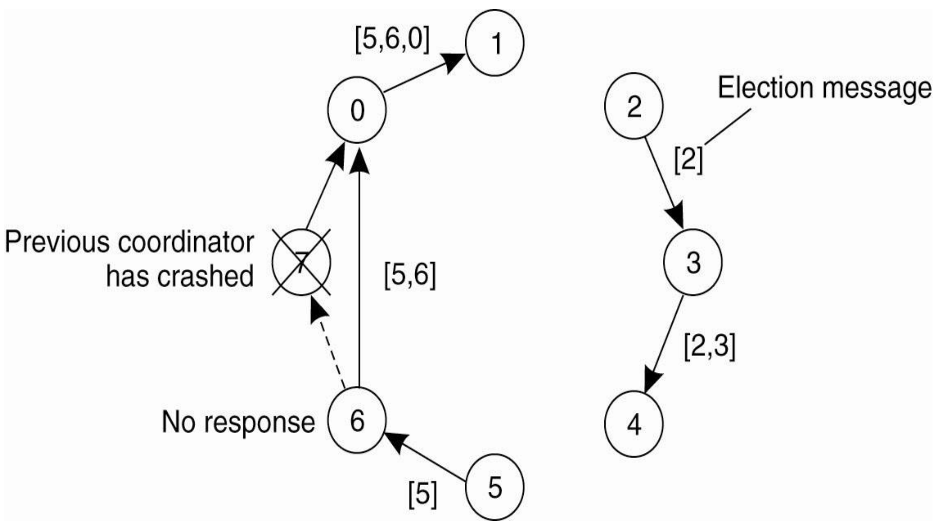
- However, if a process calls an election when the coordinator is still active, the coordinator will win the election.

A Ring Algorithm - Overview

- The ring algorithm assumes that the processes are arranged in a logical ring and each process is knows the order of the ring of processes.
- Processes are able to “skip” faulty systems: instead of sending to process j, send to j + 1.
- Faulty systems are those that don’t respond in a fixed amount of time.

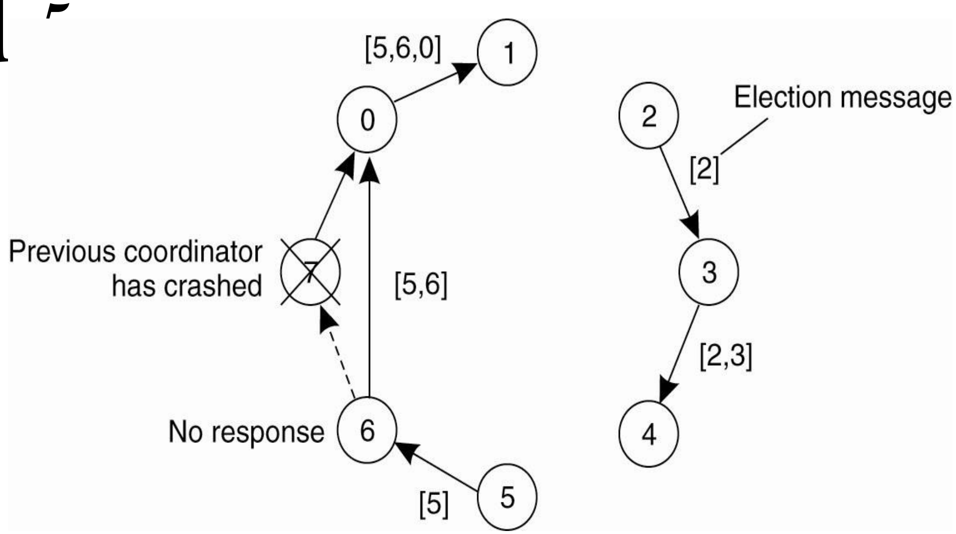
A Ring Algorithm

- P thinks the coordinator has crashed; builds an ELECTION message which contains its own ID number.
- Sends to first live successor
- Each process adds its own number and forwards to next.
- OK to have two elections at once.

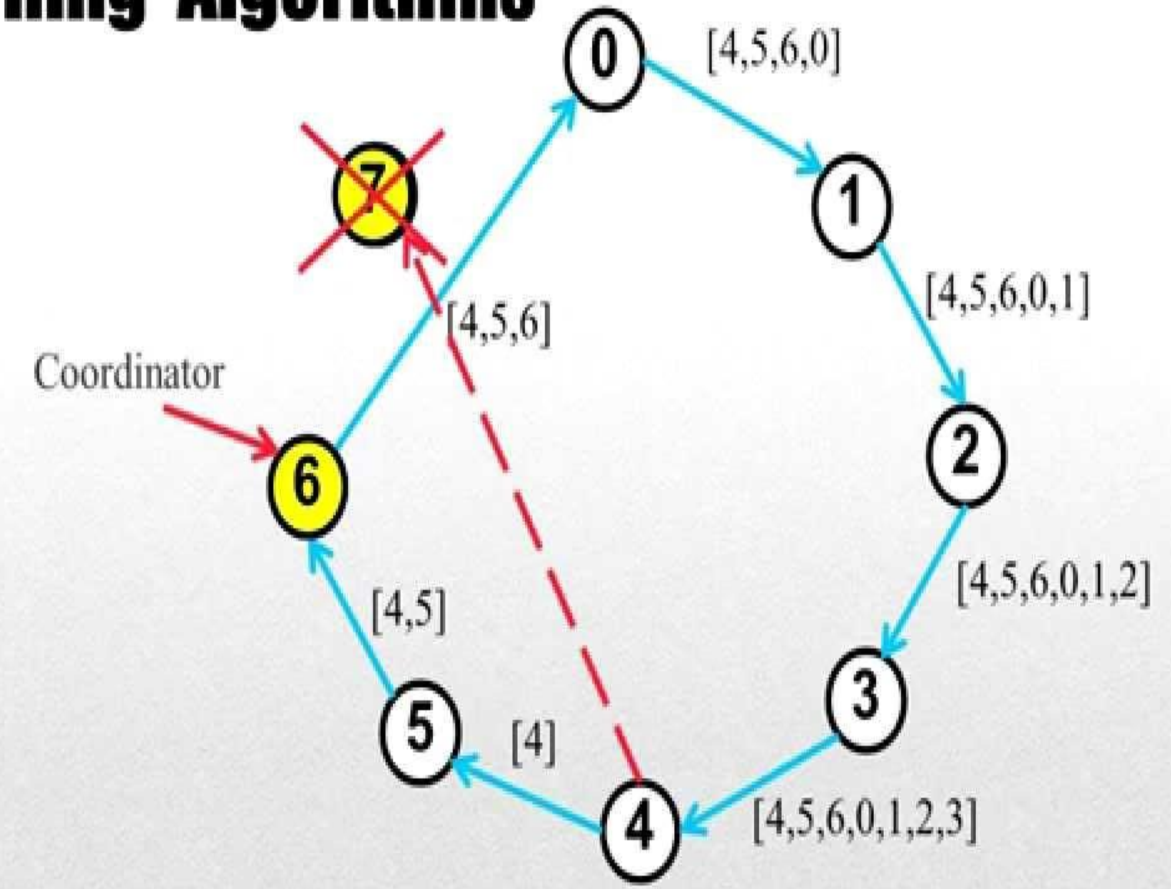


Ring Algorithm - Details

- When the message returns to *p*, it sees its own process ID in the list and knows that the circuit is complete.
- P circulates a COORDINATOR message with the new high number.
- Here, both 2 and 5 elect 6:
  [5,6,0,1,2,3,4]
  [2,3,4,5,6,0,1]



Ring Algorithms



Comparison

- Assume n processes and one election in progress
- **Bully Algorithm**
  - **Worst case:** initiator is node with lowest ID

    Triggers n-2 elections at higher ranked nodes: O(n2) msgs
  - **Best case:** immediate election: n-2 messages
  - • **Ring Algorithm**
  - 2 (n-1) messages always