Distributed Systems Lecture 7

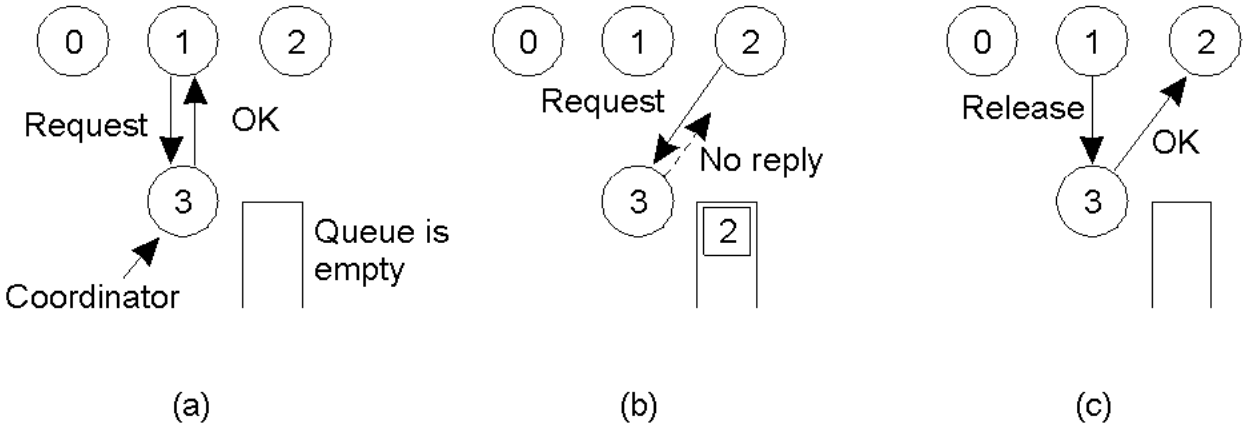
Professor Louise E. Moser

Spring 2014

Mutual Exclusion

- **Problem:** Multiple processes in a distributed system want exclusive access to a shared resource
- **Four approaches**
  - **Centralized algorithm**
  - **Decentralized algorithm**
  - **Distributed algorithm, with no imposed topology**
  - **Distributed algorithm, using a logical ring topology**

Centralized Mutual Exclusion Algorithm



(a) Process 1 asks the **coordinator** (process 3) for permission to enter a critical region where the shared resource may be accessed. The coordinator grants permission to process 1

(b) Process 2 then asks permission to enter the same critical region. The coordinator does not reply, and puts process 2 in its queue

(c) When process 1 exits the critical region, it tells the coordinator. The coordinator grants permission to process 2, and removes process 2 from its queue

Problem: The coordinator is a single point of failure

Decentralized Mutual Exclusion Algorithm

- **Idea:** **Replicate** the shared resource **n times**; **each replica has its own coordinator** for controlling access by concurrent processes
- To access the shared resource, a process needs to get permission from a **majority m > n/2** of the coordinators
  - **If a coordinator does not give permission, it tells the process requesting access**
- If the process gets less than n/2 votes, it backs off a random amount of time and tries again later
- The decentralized algorithm is less vulnerable to failure of a single coordinator, but is only probabilistically correct
  - **When a coordinator crashes, it recovers quickly and resets itself. Doing so can result in incorrectly granting access to the shared resource**
- Moreover, if many processes want to access the shared resource, no one will get enough votes and the resource will be un-utilized

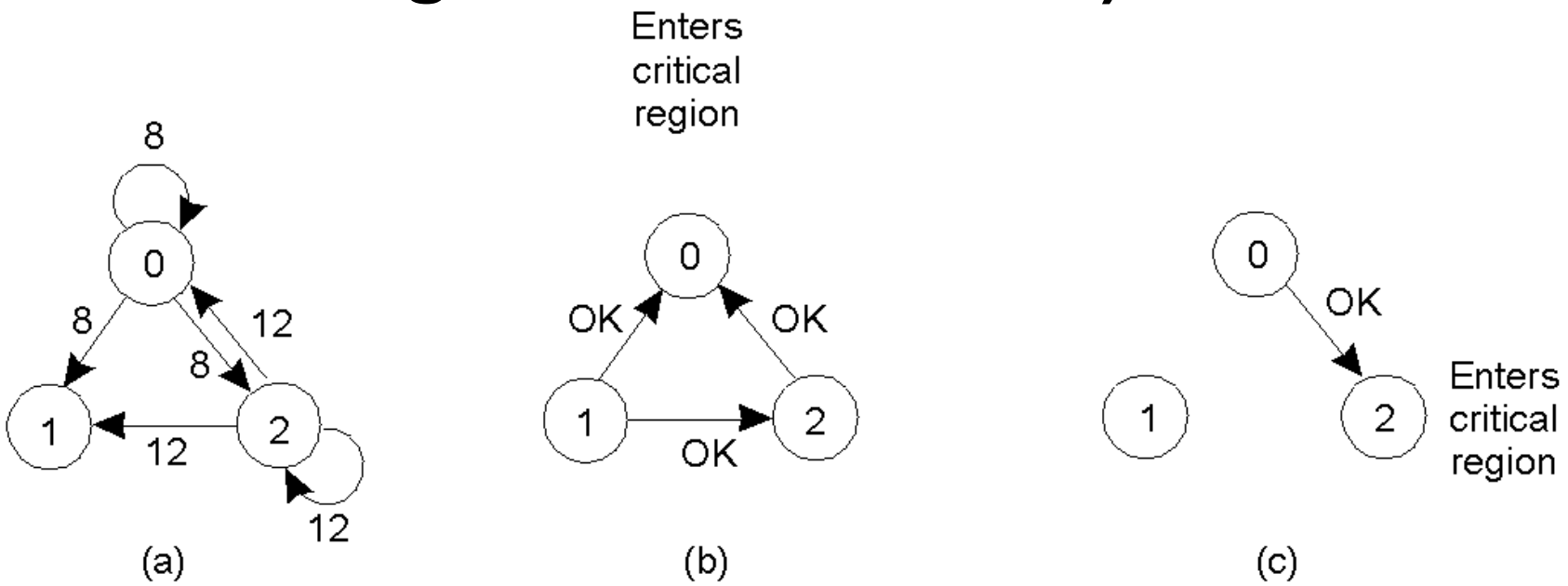
Distributed Mutual Exclusion Algorithm

- **Ricart and Agrawala Mutual Exclusion Algorithm**
  - **Requires total ordering of events in the distributed system**
  - **Assumes reliable delivery of messages**
- To access a shared resource, a process **sends** a **request message** to **all** processes (including itself), containing:
  - **The resource name**
  - **Its process id**
  - **Its current timestamp**
- After sending the request message, the process **waits** until it has received permission from **all** processes and then proceeds to access the resource
- When it has finished accessing the resource, the process sends **OK messages** to all processes in its queue and then removes them from its queue (see queue explanation on next slide)

Distributed Mutual Exclusion Algorithm (Ricart and Agrawala continued)

- When a process **receives** a request message from another process, its action depends on its current state
  - **If it is not accessing the resource and does not want to access the resource, the process sends an OK message to the sending process**
  - **If it currently has access to the resource, the process sends nothing. Instead, it queues the request message**
  - **If it wants to access the resource but has not yet done so, the process compares the timestamp of the request message it received with the timestamp of the request message it sent**
    - If the **request message it received** has a **lower timestamp**, it sends an **OK message** to the sending process
    - If the **request message it sent** has a **lower timestamp**, it queues the request message it received and **sends nothing**

Distributed Mutual Exclusion Algorithm (Ricart and Agrawala continued)



(a) Two processes, process 0 and process 2, want to enter the critical region at the same time

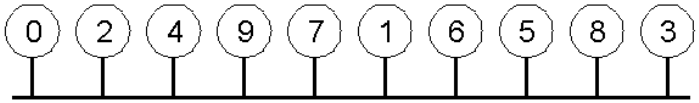
(b) Process 0 has the lower timestamp, so it wins and enters the critical region

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

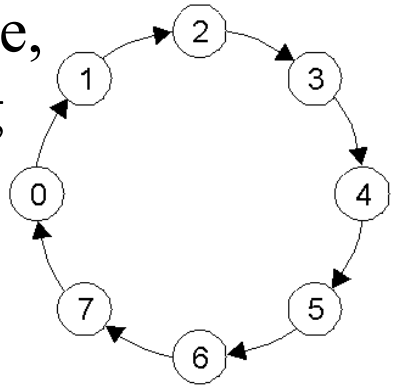
Token Ring Mutual Exclusion Algorithm

Idea: Organize the processes in a **logical ring**, with a **token** that is passed around the ring. A process may enter the critical region (access the shared resource) only if it holds the token

- To enter the critical region, a process must wait to receive the token
- When a process has finished accessing the resource, it releases the token to the next process on the ring



(a)



(b)

(a) An unordered group of processes on a network

(b) A logical ring of processes, constructed in software

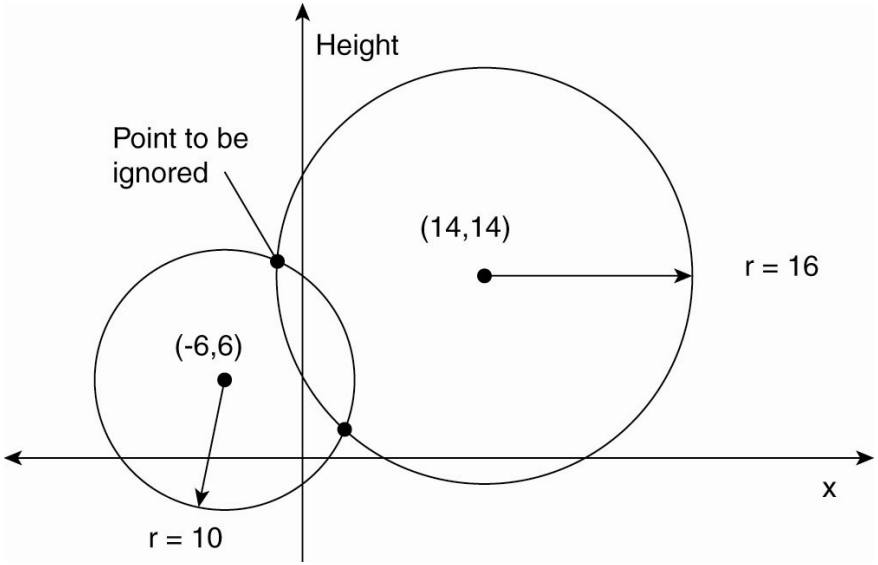
Comparison of the Mutual Exclusion Algorithms

Algorithm	Messages per access/release	Delay before access (in message times)	Problems
Centralized	3	2	Coordinator crash
Decentralized	2mk + m	2mk	Starvation; inefficiency
Distributed	2(n–1)	2(n–1)	Process crash
Token ring	1 to ∞	0 to n–1	Token loss; process crash

n is the number of processes
m is the number of coordinators
k is the number of attempts to get m > n/2 votes

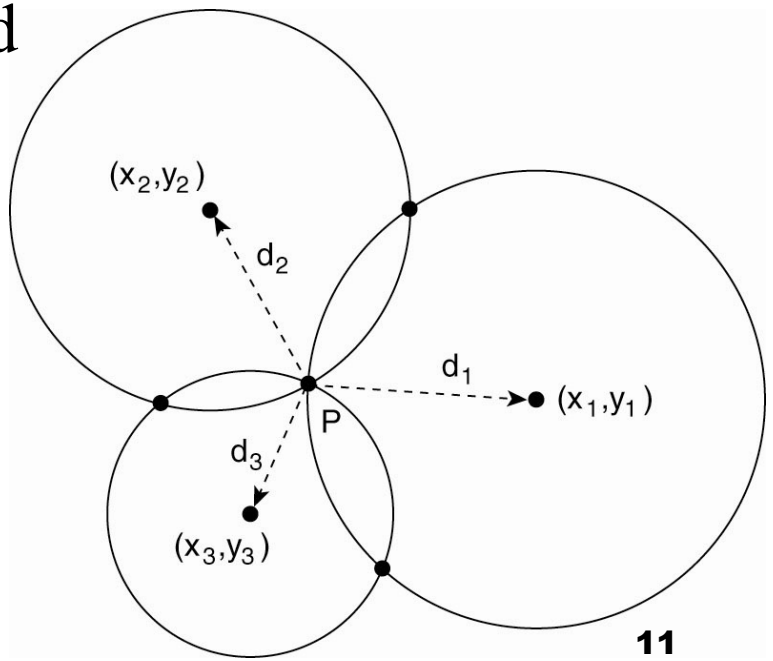
Global Positioning System

- Global Positioning System (GPS) satellites broadcast an accurate clock, but the time at which you receive that clock depends on the distance to the satellite
- If you know the distance you are from 2 GPS satellites, you are somewhere on a circle (in the figure, the 2 points represent the circle and the circles represent spheres)



Global Positioning System

- If you know the distance you are from 3 GPS satellites, you are approximately at a point (in the figure, the intersection point represents the intersection of 2 circles and the circles represent spheres)
- Actually, 5 GPS satellites are used to get an accurate spatial position, because of variations in the signal propagation delay due to atmospheric conditions



Election Algorithms

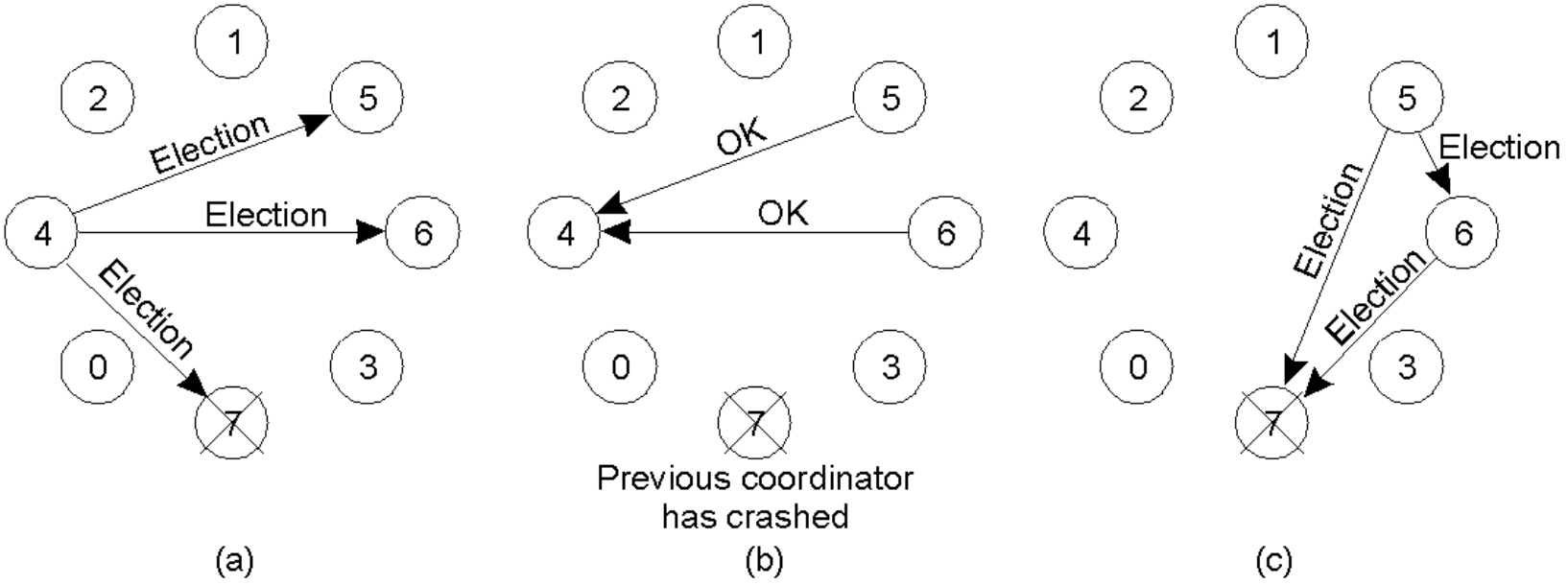
- **Principle:** An algorithm in a distributed system requires that some process acts as the **coordinator (leader)**
- **Question:** How do we select the coordinator **dynamically**?
  - **Note: In many systems, the coordinator is chosen manually (e.g., file server); doing so leads to a centralized solution with a single point of failure**
- **Question:** If the coordinator is chosen dynamically, to what extent can we talk about a centralized or distributed solution?
- **Question:** Is a fully distributed algorithm, i.e., one without a coordinator, more robust than a centralized/coordinated algorithm?
  - **Think again about the mutual exclusion algorithms**

Election by Bullying

- **Idea:** Each process has an associated **weight (priority)**
  - **Elect the process with the *greatest* weight as the coordinator**
- **Problem:** How do we find the process with the greatest weight?

1. Any process P can start an election (e.g., when it detects that the coordinator is no longer responding) by sending an **Election message** to *all* other processes, or to all processes with *greater* weights if it knows their weights
2. If a *heavier* weight process Q receives an **Election message** from a *lighter* weight process P, process Q sends an **OK message** to *P*, indicating that Q is alive and can take over, and that *P* is out of the race
3. If process P doesn't receive an **OK message** from any other process, it wins the race and sends a **Victory (Coordinator) message** to all other processes

The Bully Election Algorithm

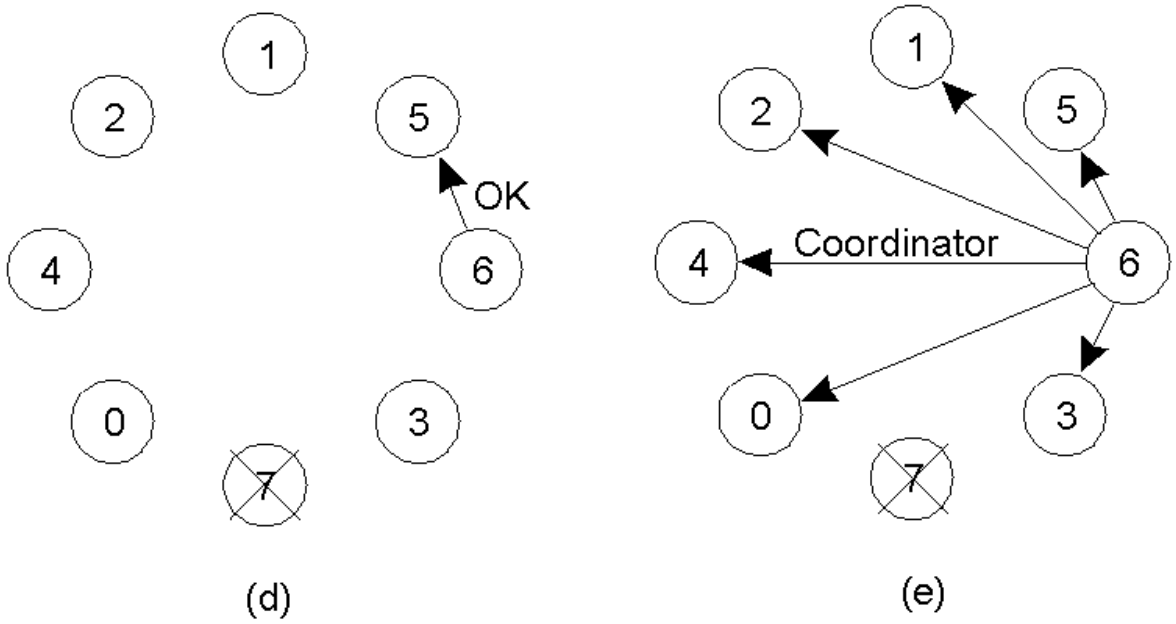


(a) Process 4 detects that process 7 has crashed, and initiates an election by sending Election messages to processes 5, 6, 7

(b) Processes 5 and 6 respond with OK messages, indicating that they are alive and can take over, and process 4 drops out of the race

(c) Process 5 holds an election by sending Election messages to processes 6 and 7, and process 6 holds an election by sending an Election message to process 7

The Bully Election Algorithm (continued)



(d) Process 6 responds with an OK message to process 5, indicating that it is alive and can take over, and process 5 drops out of the race

(e) Process 6 wins the race and sends a Victory (Coordinator) message to all other processes

Election in a Ring

- **Idea:** The **priority (weight)** of a process is determined by its **position on a logical ring**
  - **Elect the process with the *highest* priority as the coordinator**

1. Any process can start an election by sending an **Election message** to its **successor** on the ring. If its successor is down, it sends the message to the next successor on the ring

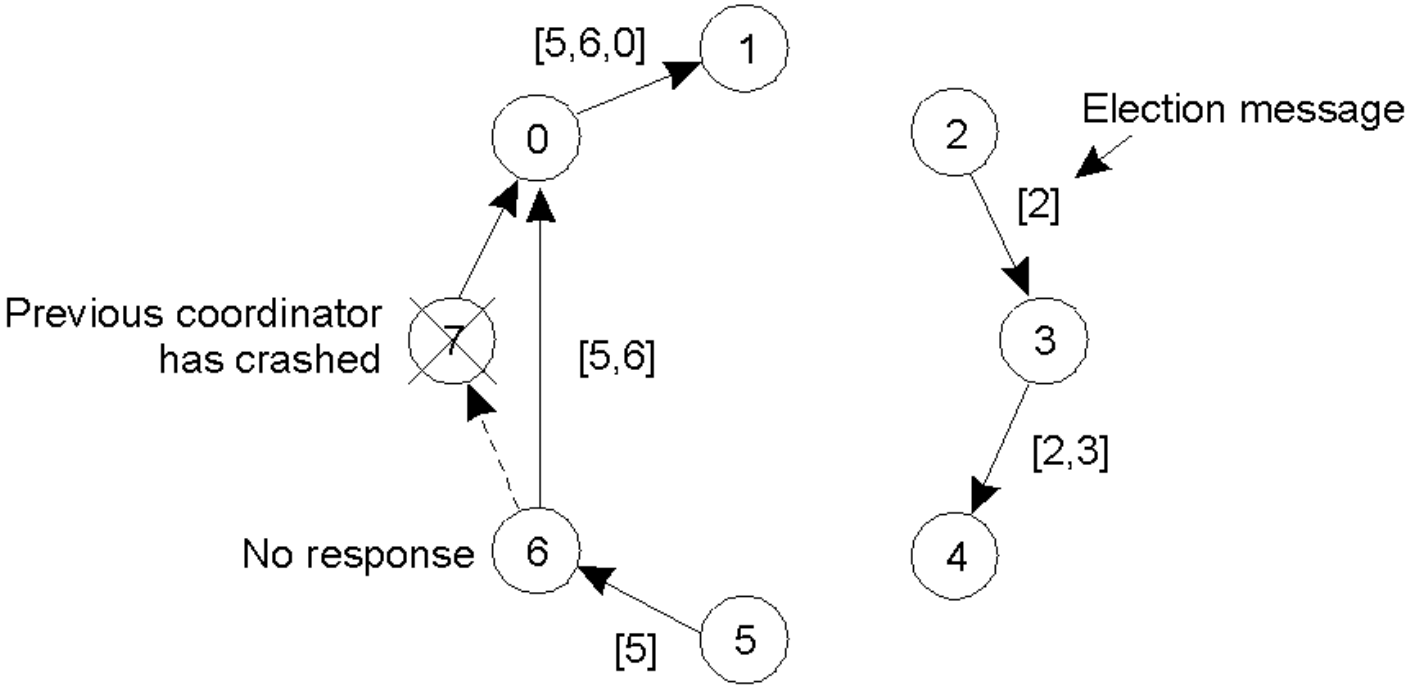
2. When a sender sends an **Election message**, the sender adds itself to an **alive list**. When the message gets back to the initiator, each process has had the chance to let the initiator know that it is alive

3. The initiator sends a **Coordinator message** around the ring containing the list of all processes that are alive. The process with the *highest* priority becomes the coordinator

- **Does it matter if two different processes initiate an election?**
- **What happens if a process crashes *during* the election?**

A Ring Election Algorithm

- Processes 2 and 5 each detect that process 7 (the coordinator) has crashed, and each initiate an election



A Ring Election Algorithm (continued)

- Process 5 sends an Election message to process 6, containing [5]
- Process 6 sends an Election message to process 7, but gets no response, so process 6 sends an Election message to process 0, containing [5, 6]
- Process 0 sends an Election message to process 1, containing [5,6,0]
- Process 1 sends an Election message to process 2, containing [5,6,0,1]
- Process 2 sends an Election message to process 3, containing [5,6,0,1,2]
- Process 3 sends an Election message to process 4, containing [5,6,0,1,2,3]
- Process 4 sends an Election message to process 5, containing [5,6,0,1,2,3,4]
- Process 5, the initiator, sends the message containing [5,6,0,1,2,3,4] around the ring again
  - **When a process receives the message, it knows which other processes are alive and which process is the coordinator, in this case process 6**
  - **The same is true in the case that process 2 initiates the election**

Elections in Wireless Environments

- The previous election algorithms assume that
  - **Message communication is reliable**
  - **Network topology does not change**

  which is not the case in wireless environments
- Moreover, the previous election algorithms do not try to choose **the "best" leader**, which is more important in wireless environments
- Elections in wireless environments often use a tree-like algorithm, as described on Slides 20-23

Elections in Wireless Environments

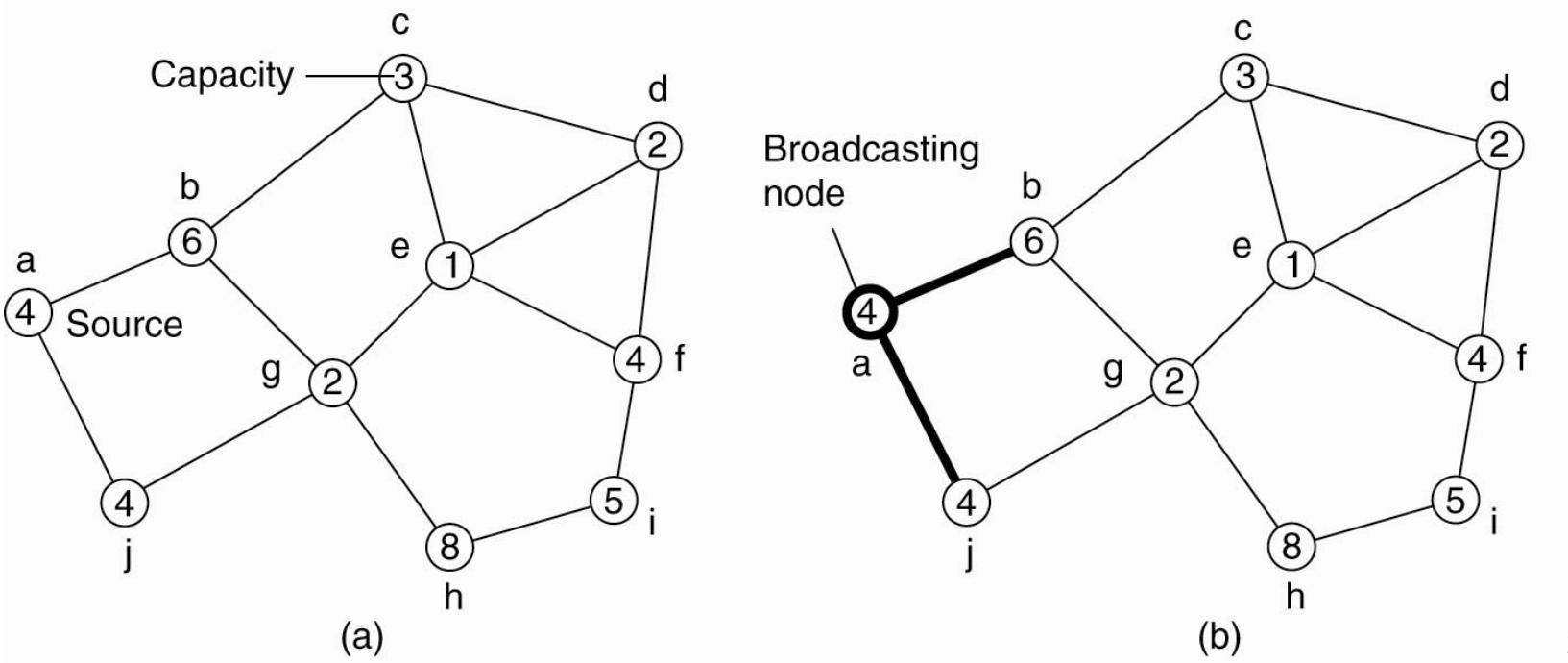
- **Idea:** The **capacity** of a node is a measure of the node's ability to be the leader, based on *how quickly it responds*

1. Any node S in the network (called the **source**) can start an election by sending an **Election message** to *all* its neighbors
2. The first time a node R (a **receiver**) receives an **Election message**, it designates the sender P as its **parent** and sends an **Election message** to each neighbor N, except for P
   - **R waits for acks from its neighbors, before it sends an ack to its parent P (to assess the capacities of those nodes and ultimately to select the best leader)**
3. When a node N receives an **Election message** from a node R other than its parent, it sends an ack to R
4. After the **Election messages** are propagated to all nodes, each node reports to its parent which node has the *best* capacity to be the leader
5. Finally, the source node S determines which node is the *best* leader and broadcasts that information to all other nodes

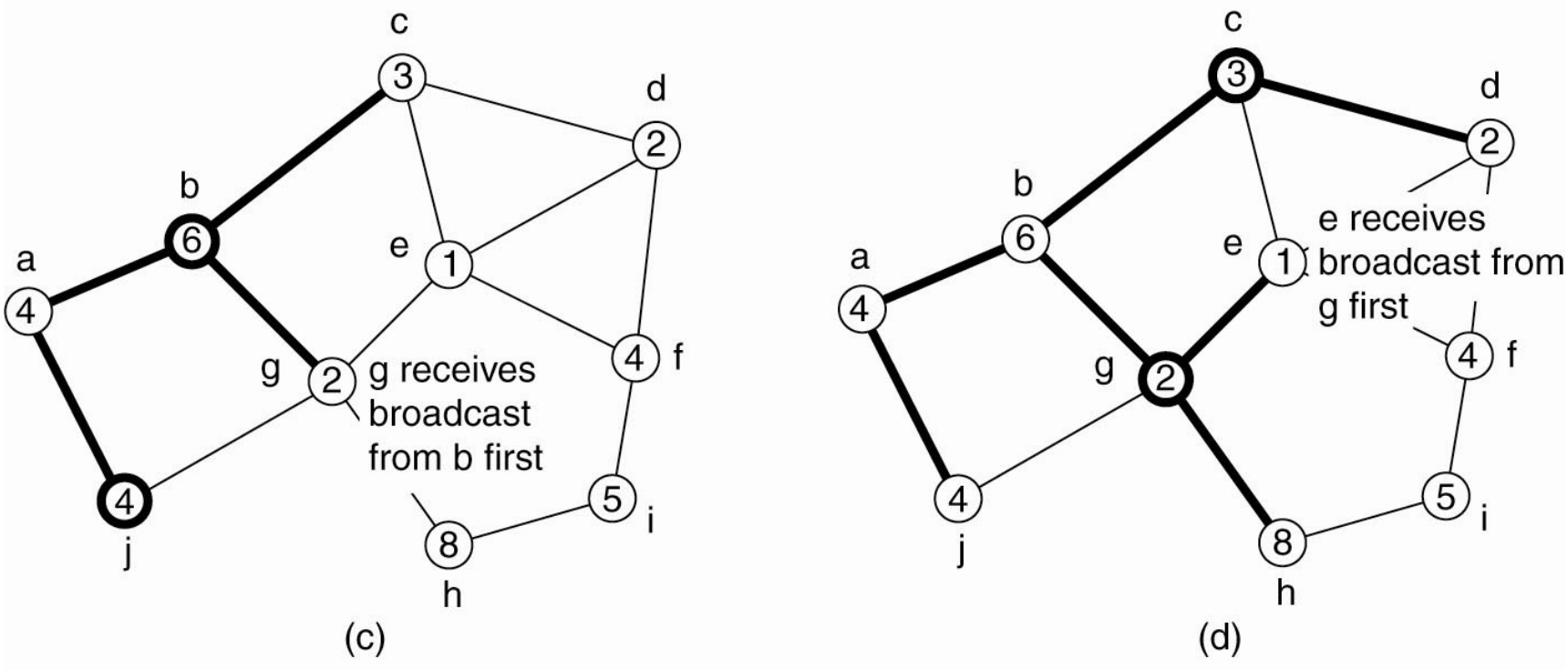
Elections in Wireless Environments

- Election algorithm in a wireless network with the source node *a*

(a) Initial network **(b)–(e) The build-tree phase**



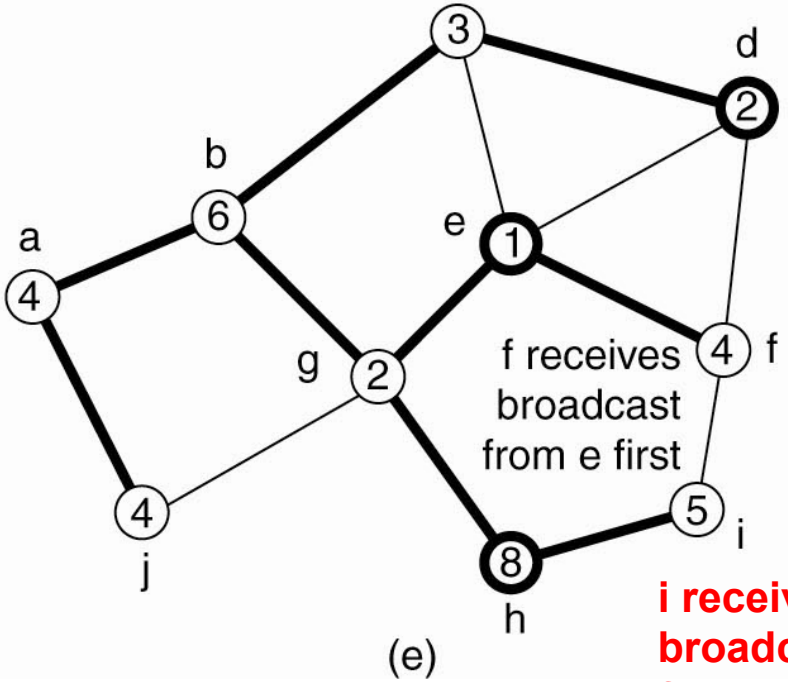
Elections in Wireless Environments (continued)

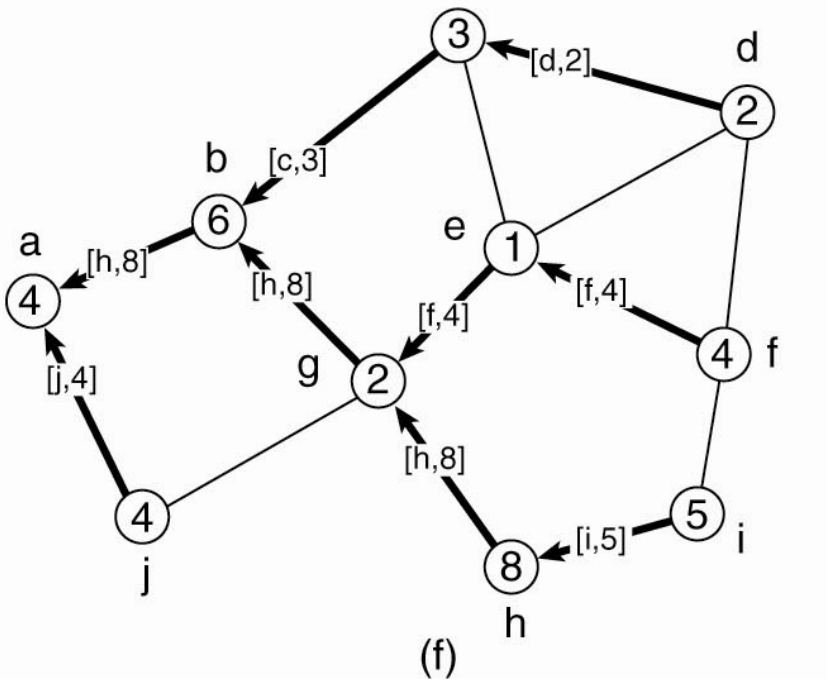


Elections in Wireless Environments (continued)

(f) Reporting the node with the best capacity to the parents, up the tree back to the source node a

For example, node h has capacity 8 and node g reports to its parent node b that node h has the best capacity 8





i receives broadcast from h first

Elections in Large-Scale Systems

- The election problem in large-scale systems is different, because the objective is *not* to elect a *single* leader, but to select *several* nodes, called **super-peers**
- Requirements for super-peer selection in P2P networks with an overlay network
  - **Ordinary nodes should have low-latency access to super-peers**
  - **Super-peers should be evenly distributed across the overlay network**
  - **There should be a pre-defined number of super-peers relative to the total number of nodes in the overlay network**
  - **Each super-peer should serve a fixed number of ordinary peer nodes**