HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

DISTRIBUTES SYSTEMS AND APPLICATIONS

HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

CHAPTER 4: SYNCHRONIZATION

Based on the lectures of Assoc. Prof. Hà Quốc Trung

Outline

- Physical clock synchronization
- Logical clock synchronization
- Mutual exclusion algorithms
- Election algorithms

Introduction

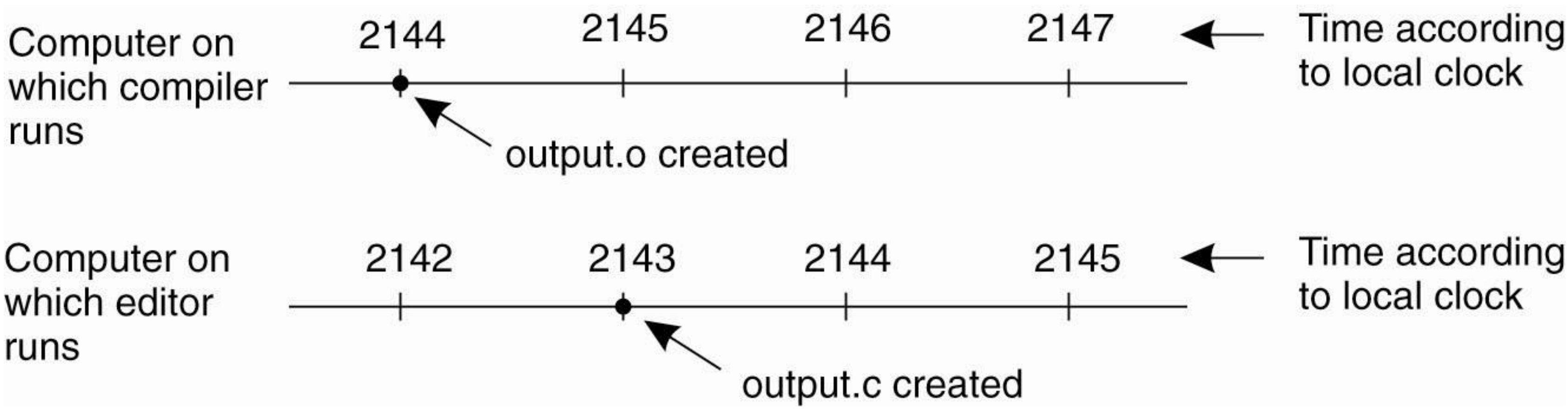
- How process synchronize
  - Multiple process to not simultaneously access to the same resources: printers, files
  - Multiple process are agreed on the ordering of event.
    - Ex: message m1 of P is sent after m2 of Q
- Synchronization based on actual time
- Synchronization by relative ordering

1. Physical clock Synchronization

1. Physical clock Synchronization

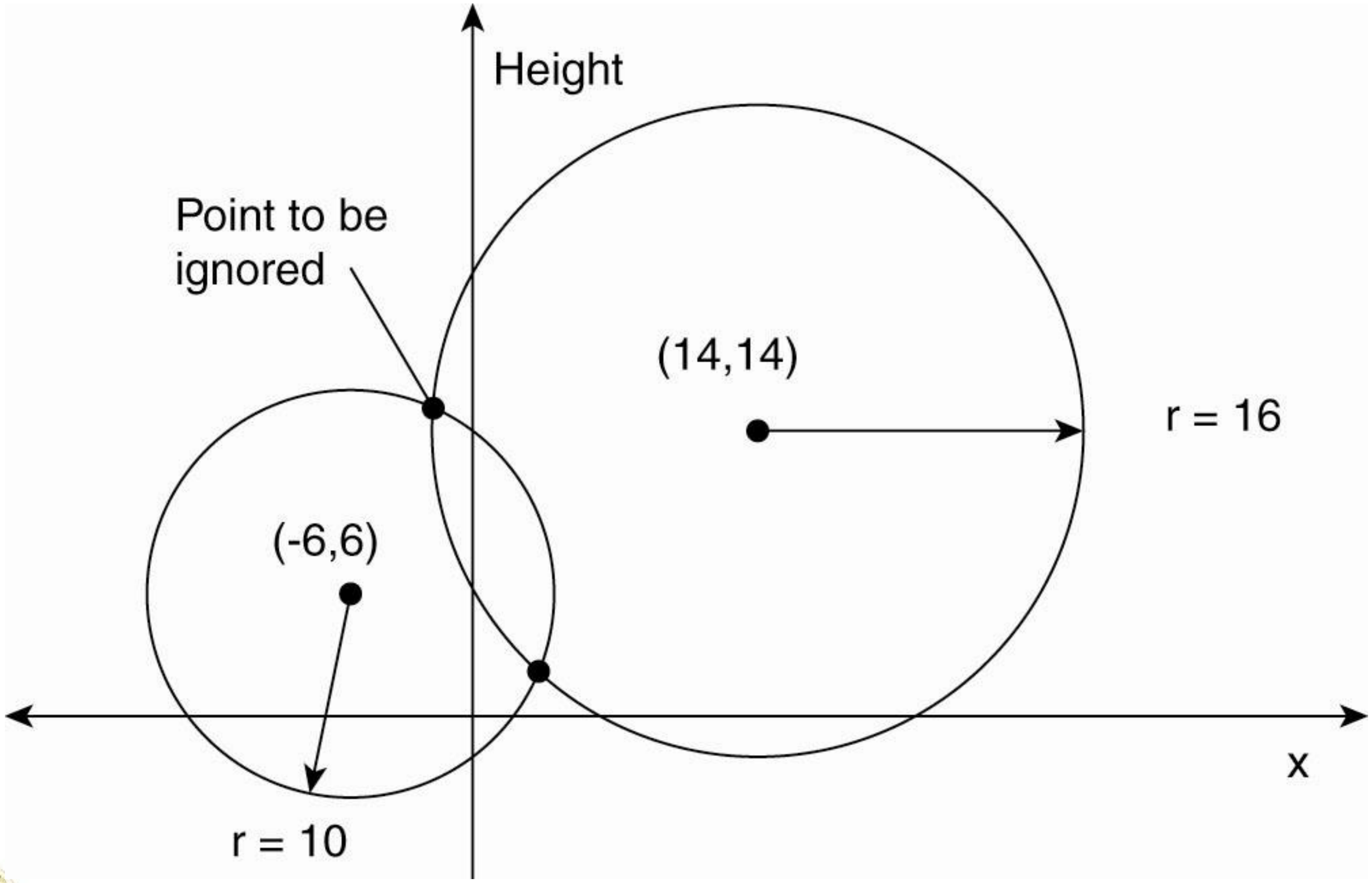
- Notion of synchronization
- Physical Clocks
- Global Positioning System
- Clock Synchronization Algorithms
- Use of Synchronized Clocks

Why do we need it?
Example 1: Programming in DS



- When each machine has its own clock, an event that occurred after another event may nevertheless be assigned an earlier time.

Example 2: Global Positioning System (1)



Computing a position in a two-dimensional space

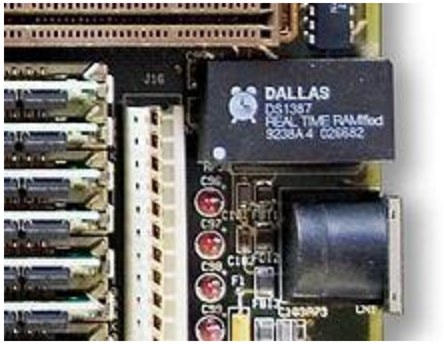
Global Positioning System (2)

- Real world facts that complicate GPS

1. It takes a while before data on a satellite's position reaches the receiver.
2. The receiver's clock is generally not in synch with that of a satellite.

Physical Clocks

- Timer
- Counter & Holding register
- Clock tick
- Problem in distributed systems:
  - How do we synchronize them with real-world?
  - How do we synchronize the clocks with each other?

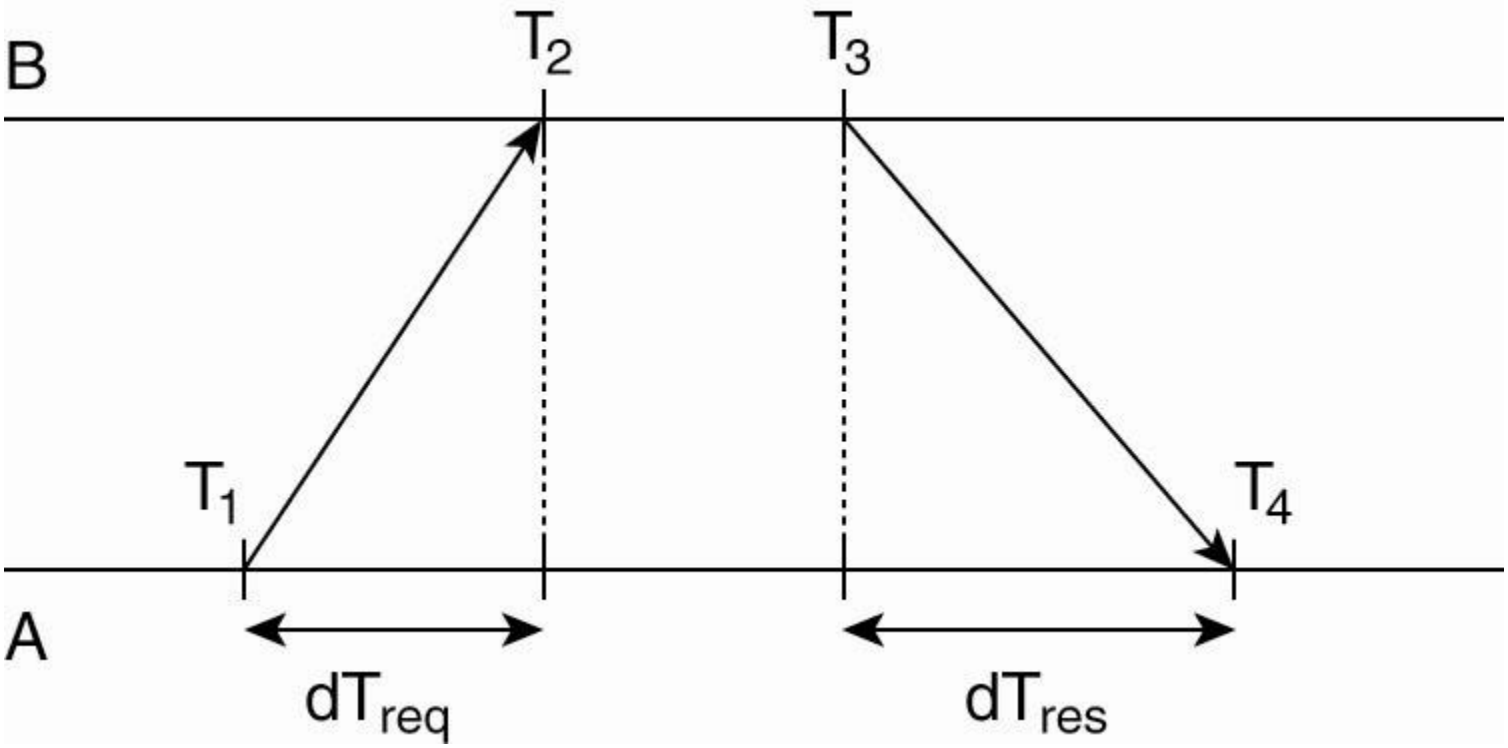


RTC IC
(Real Time Clock)

Physical Clock Synchronization Algorithms

- Network Time Protocol
- Berkeley Algorithm
- Clock Synchronization in Wireless Networks

Network Time Protocol

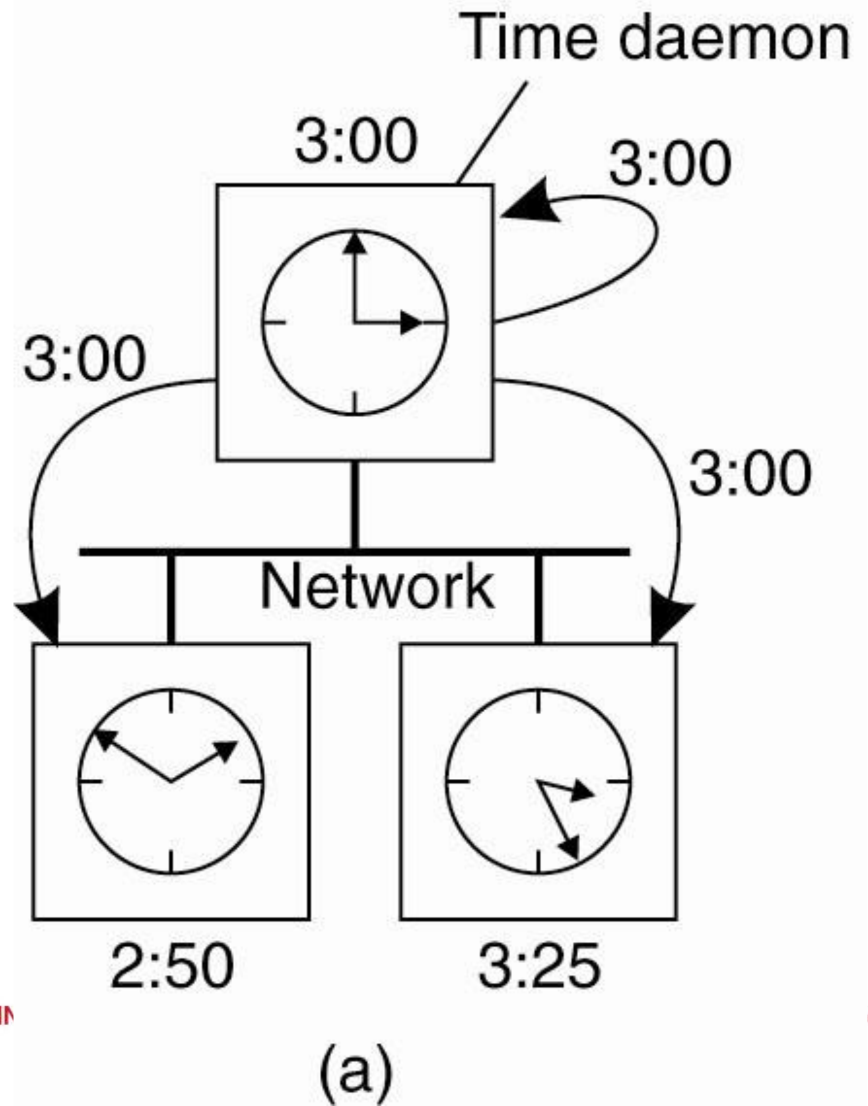


$$\theta = T_3 + \frac{(T_2 - T_1) + (T_4 - T_3)}{2} - T_4 = \frac{(T_2 - T_1) + (T_3 - T_4)}{2}$$

Getting the current time from a time server

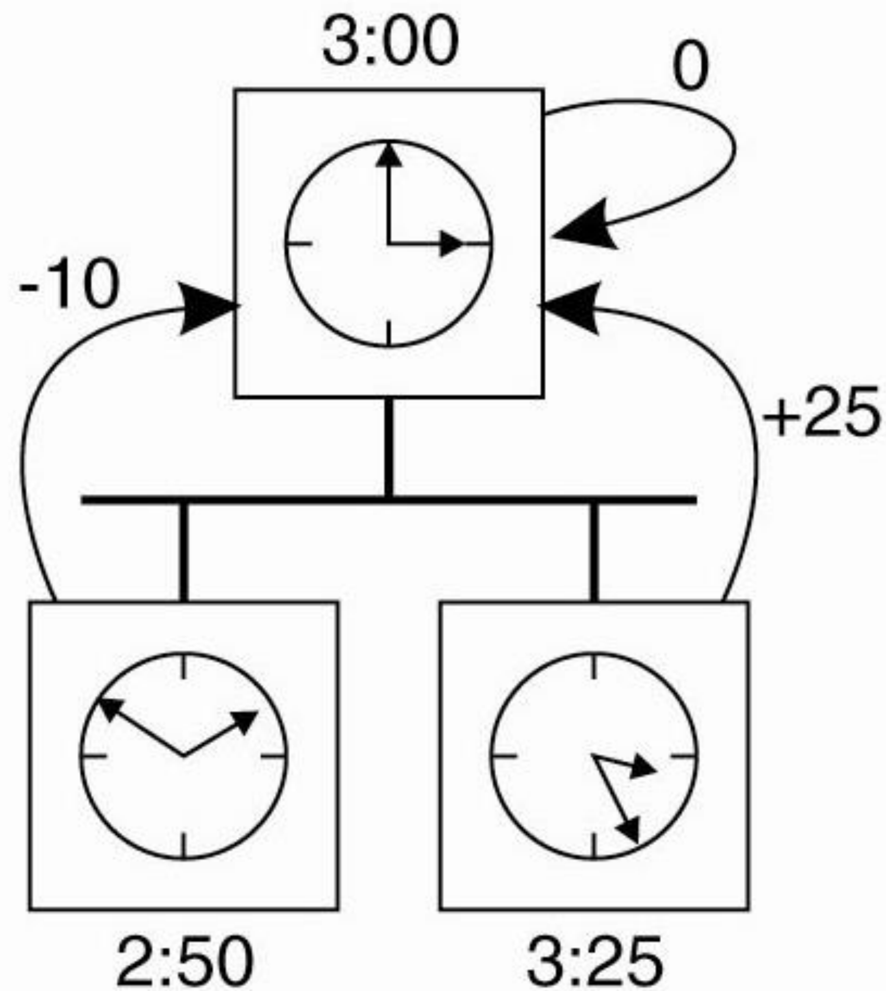
The Berkeley Algorithm (1)

- The time daemon asks all the other machines for their clock values.



The Berkeley Algorithm (2)

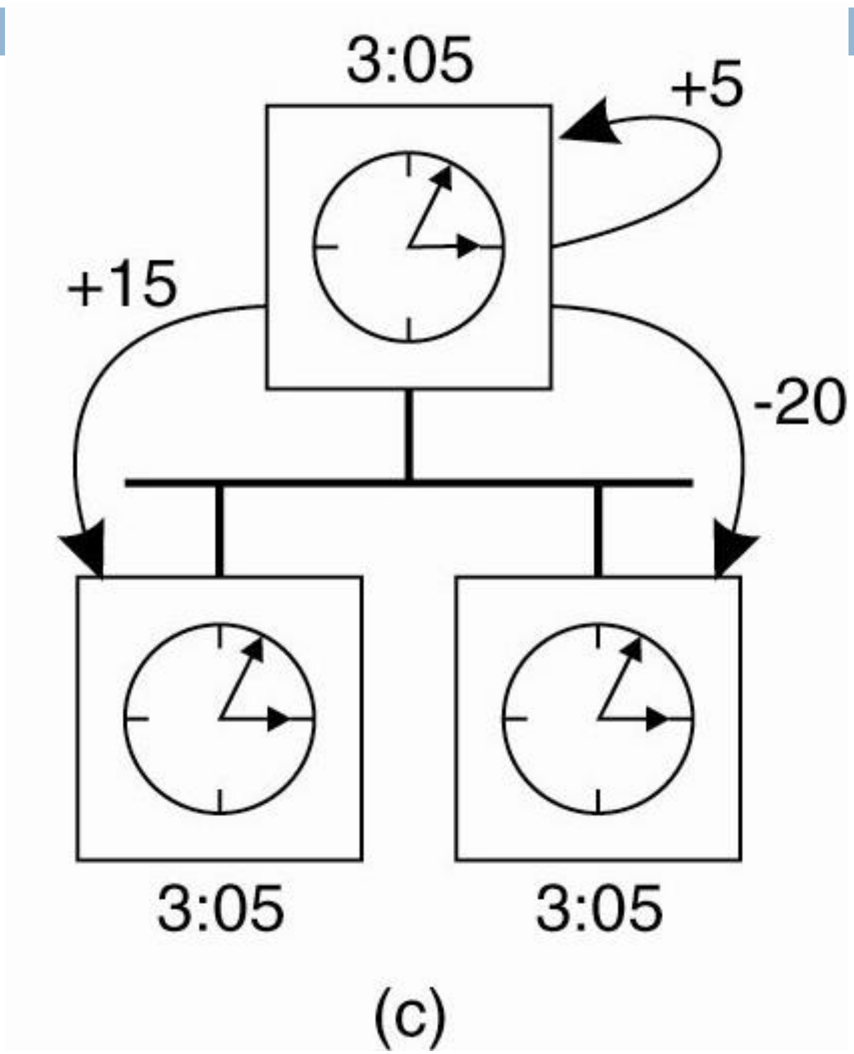
- The machines answer.



(b)

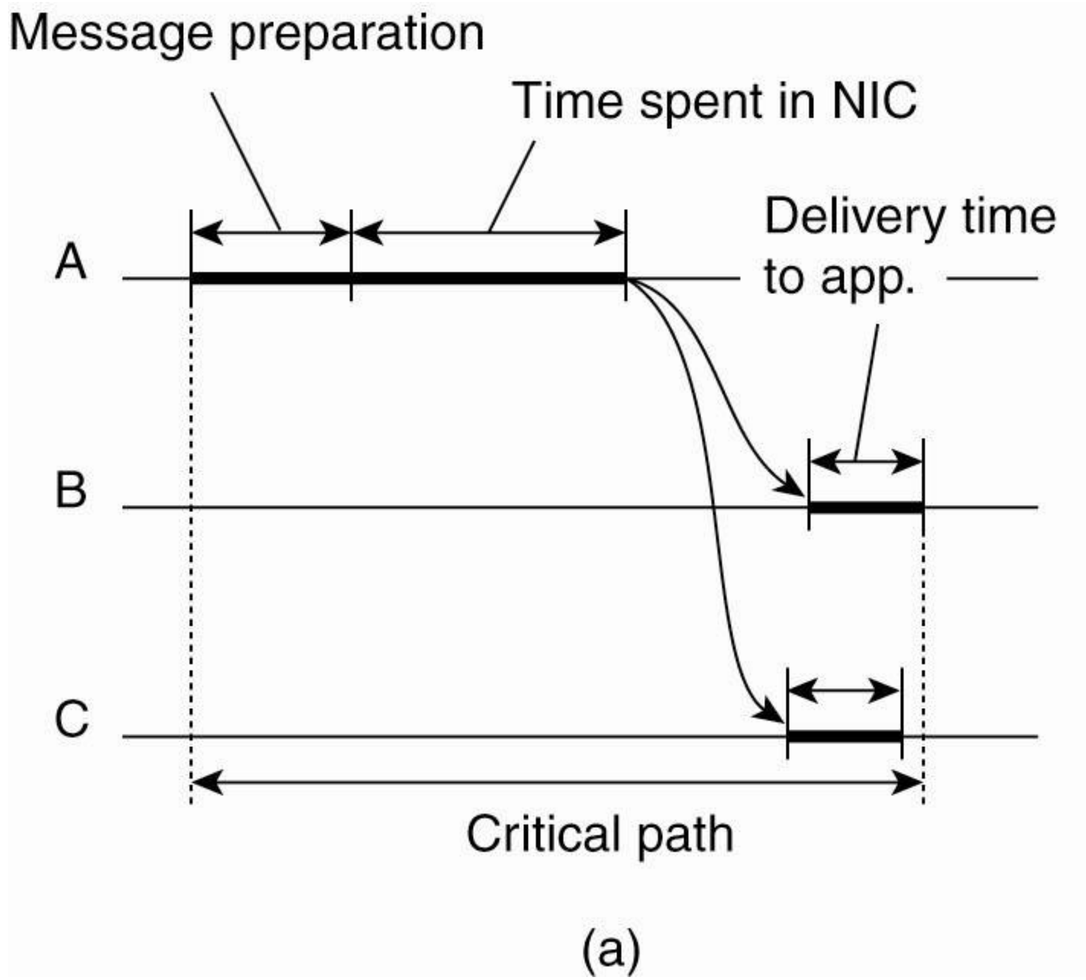
The Berkeley Algorithm (3)

- The time daemon tells everyone how to adjust their clock.



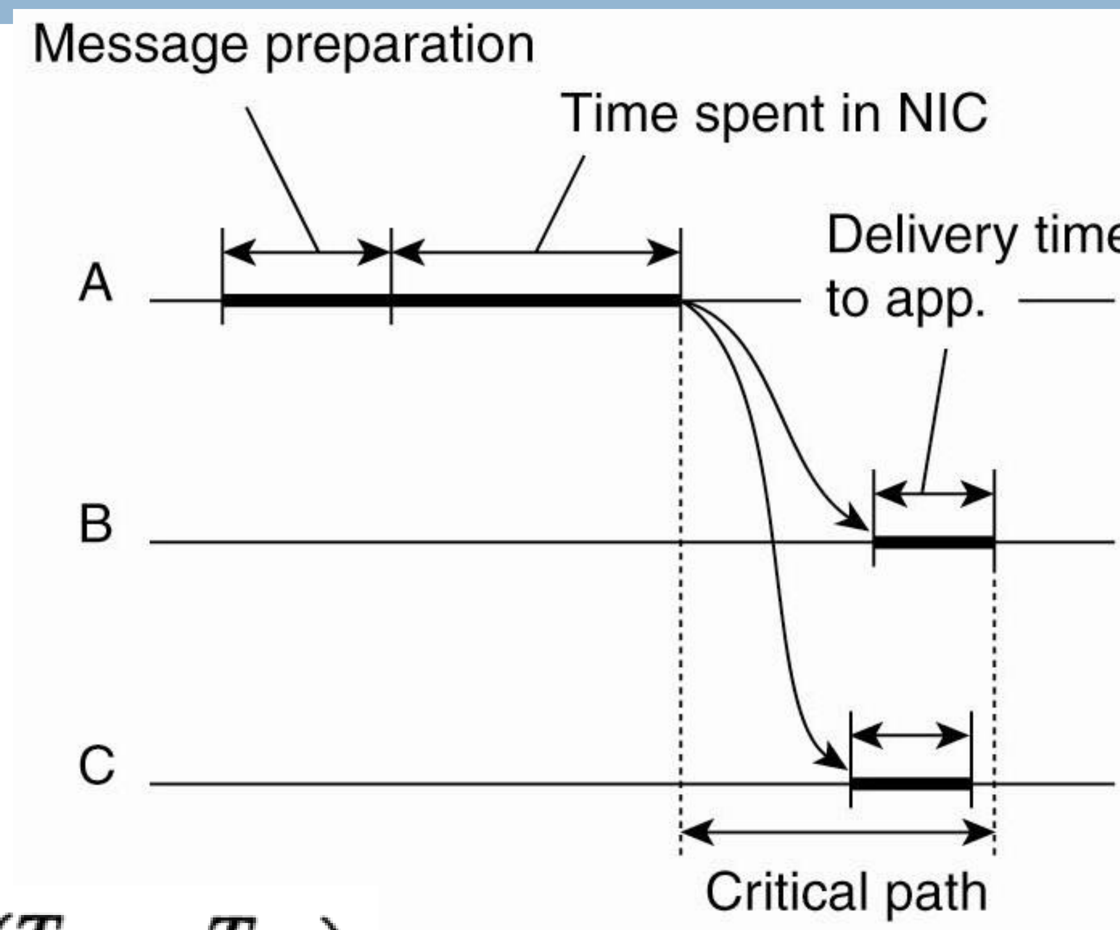
Clock Synchronization in Wireless Networks (1)

- The usual critical path in determining network delays.



Clock Synchronization in Wireless Networks (2)

- The critical path in the case of RBS.



$$Offset[p,q] = \frac{\sum_{k=1}^{M}(T_{p,k} - T_{q,k})}{M}$$

(b)

2. Logical clock synchronization

2. Logical clock synchronization

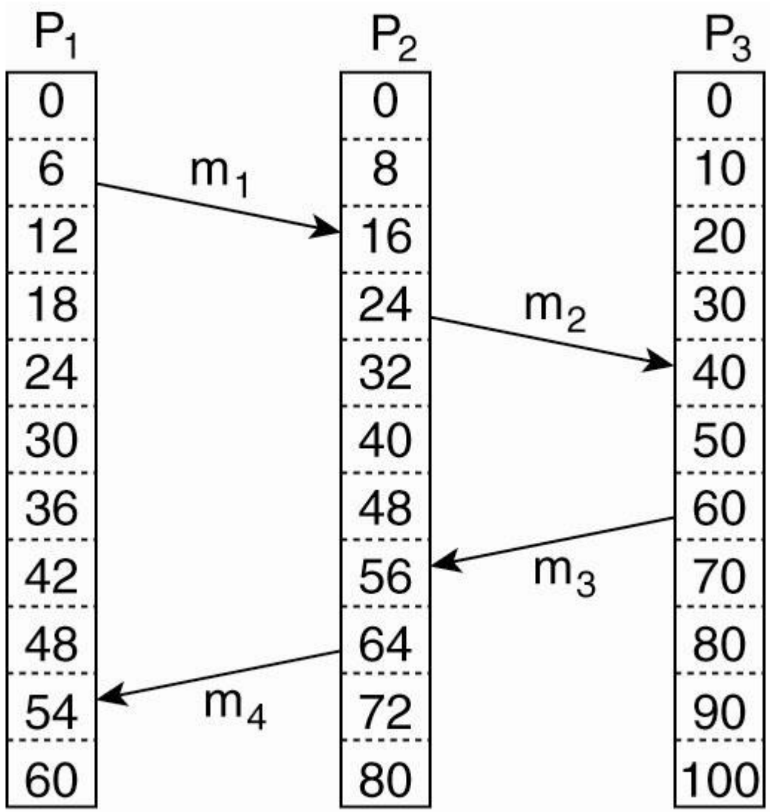
- Lamport logical clocks
- Vector clocks

2.1. Lamport's Logical Clocks (1)

- The "happens-before" relation $\rightarrow$ can be observed directly in two situations:
  1. If a and b are events in the same process, and a occurs before b, then $a \rightarrow b$ is true.
  2. If a is the event of a message being sent by one process, and b is the event of the message being received by another process, then $a \rightarrow b$
- Transitive relation: $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$
- Concurrent

Lamport's Logical Clocks (2)

- Three processes, each with its own clock. The clocks run at different rates.



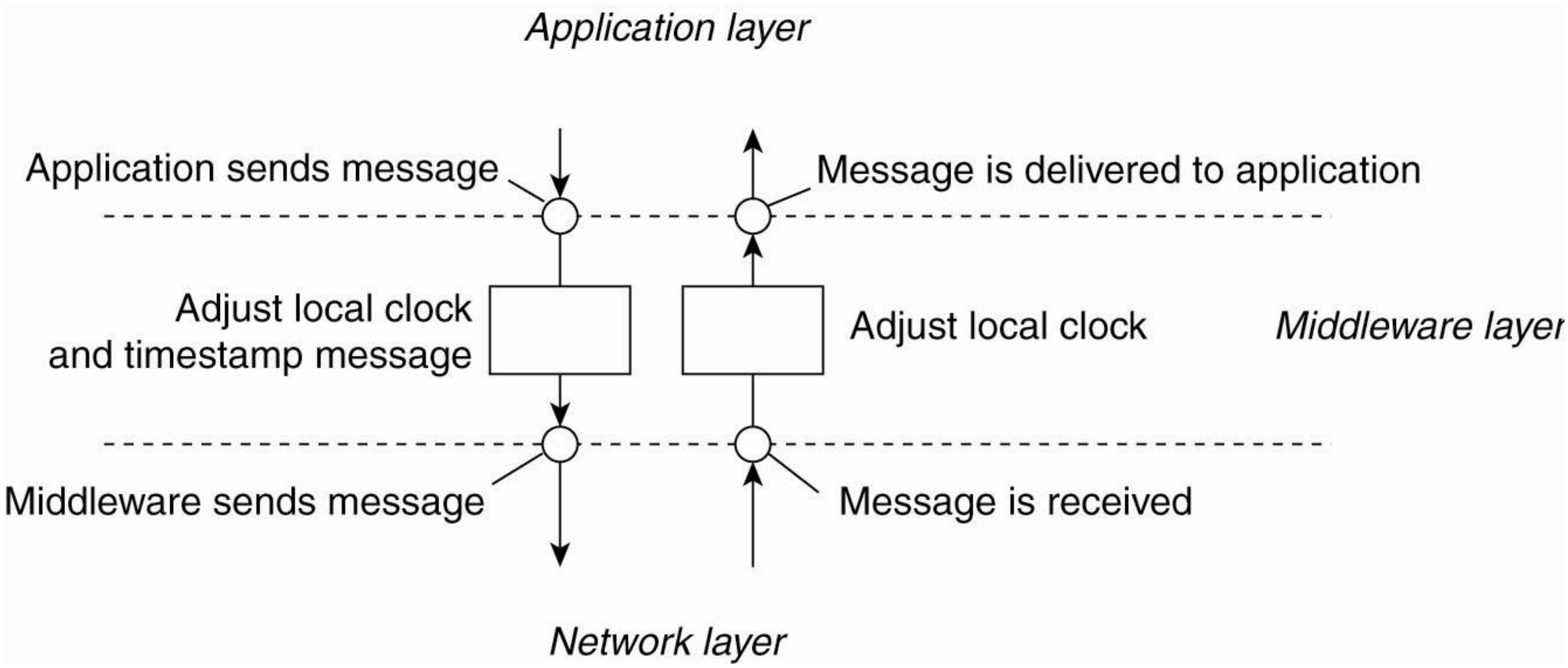
(a)

Lamport's Logical Clocks (3)

- Updating counter C_i for process P_i

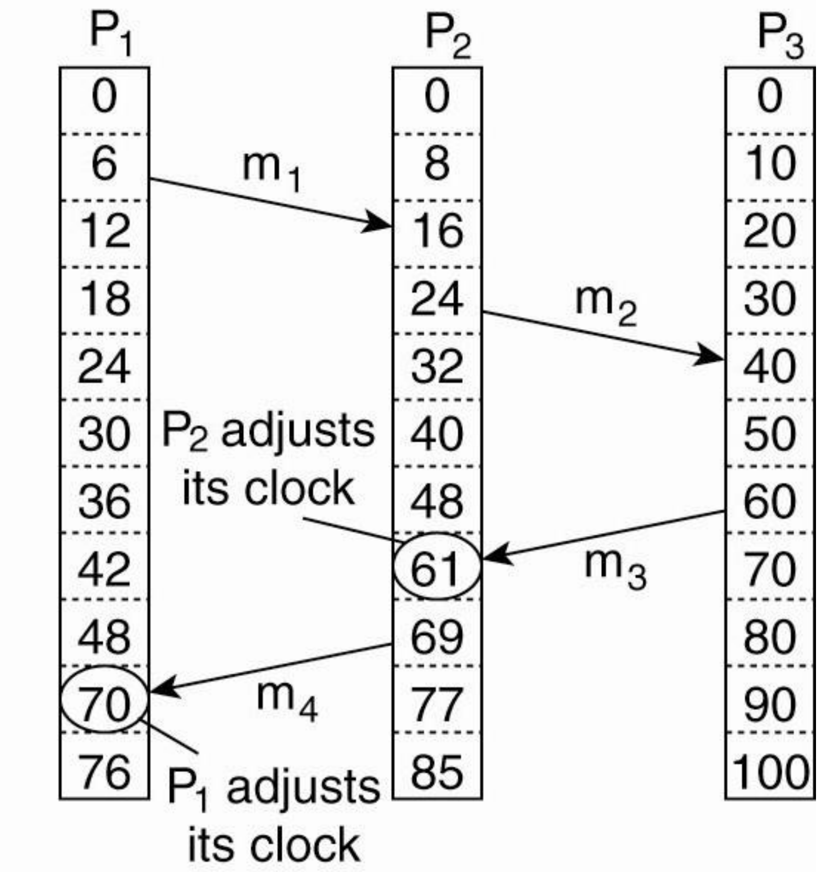
1. Before executing an event P_i executes
   $C_i \leftarrow C_i + 1$.
2. When process P_i sends a message m to P_j, it sets *m*' s timestamp *ts (m)* equal to C_i after having executed the previous step.
3. Upon the receipt of a message *m*, process P_j adjusts its own local counter as
   $C_j \leftarrow \max\{C_j , ts(m)\}$, after which it then executes the first step and delivers the message to the application.

Lamport’s Logical Clocks (4)



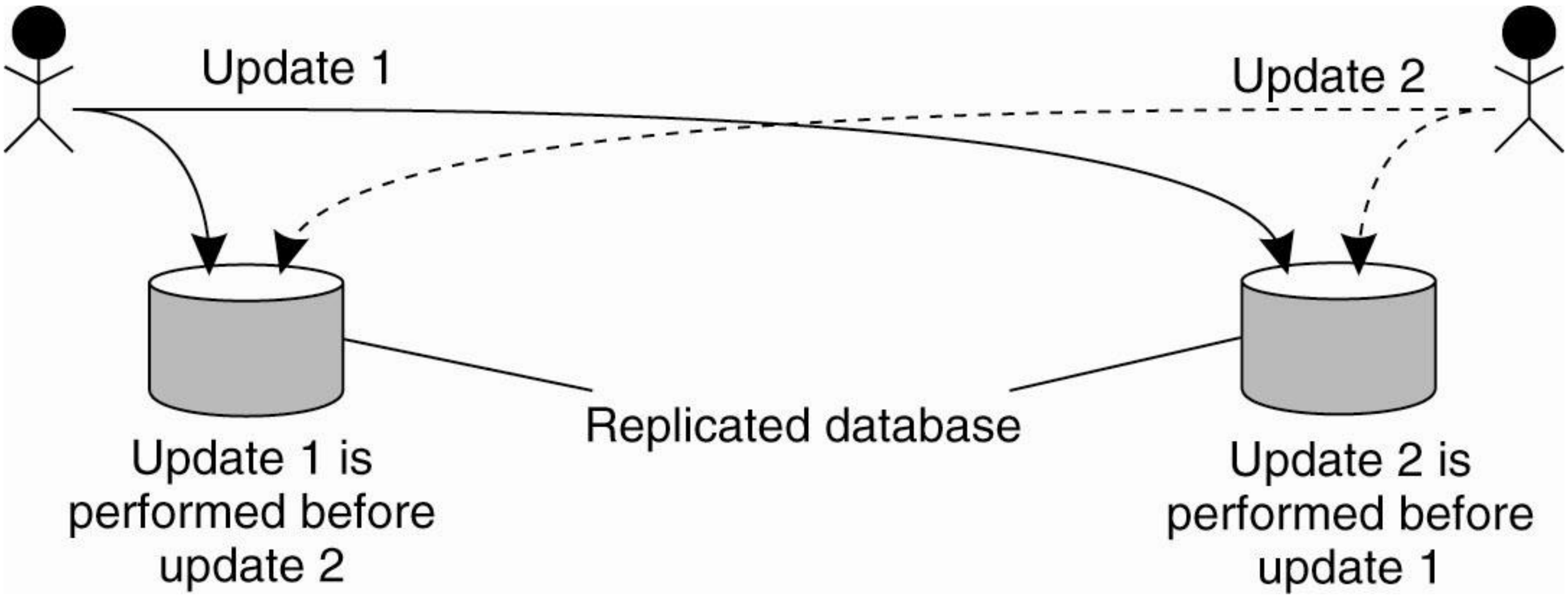
Lamport's Logical Clocks (5)

(b) Lamport's algorithm corrects the clocks.



(b)

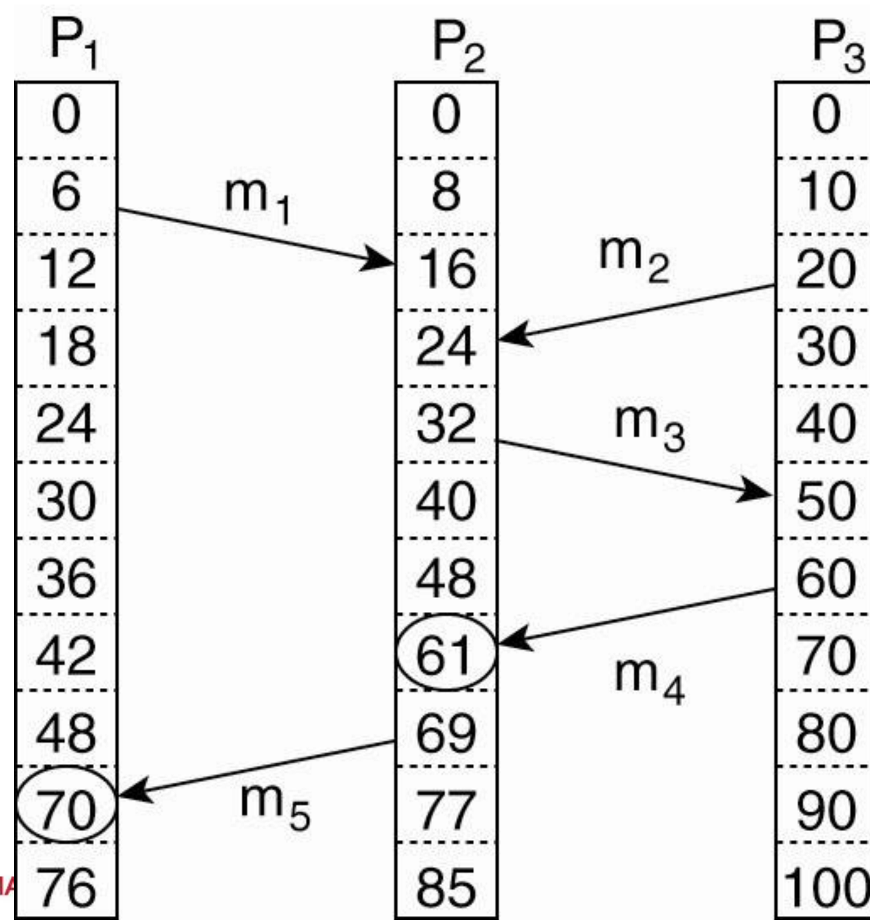
Example: Totally Ordered Multicasting



Updating a replicated database and leaving it in an inconsistent state.

2.2. Vector Clocks (1)

- Concurrent message transmission using logical clocks.



Vector Clocks (2)

- Vector clocks are constructed by letting each process P_i maintain a vector VC_i with the following two properties:

1. VC_i [i] is the number of events that have occurred so far at P_i. In other words, VC_i [i] is the local logical clock at process P_i .
2. If VC_i [j] = k then P_i knows that k events have occurred at P_j. It is thus P_i' s knowledge of the local time at P_j .

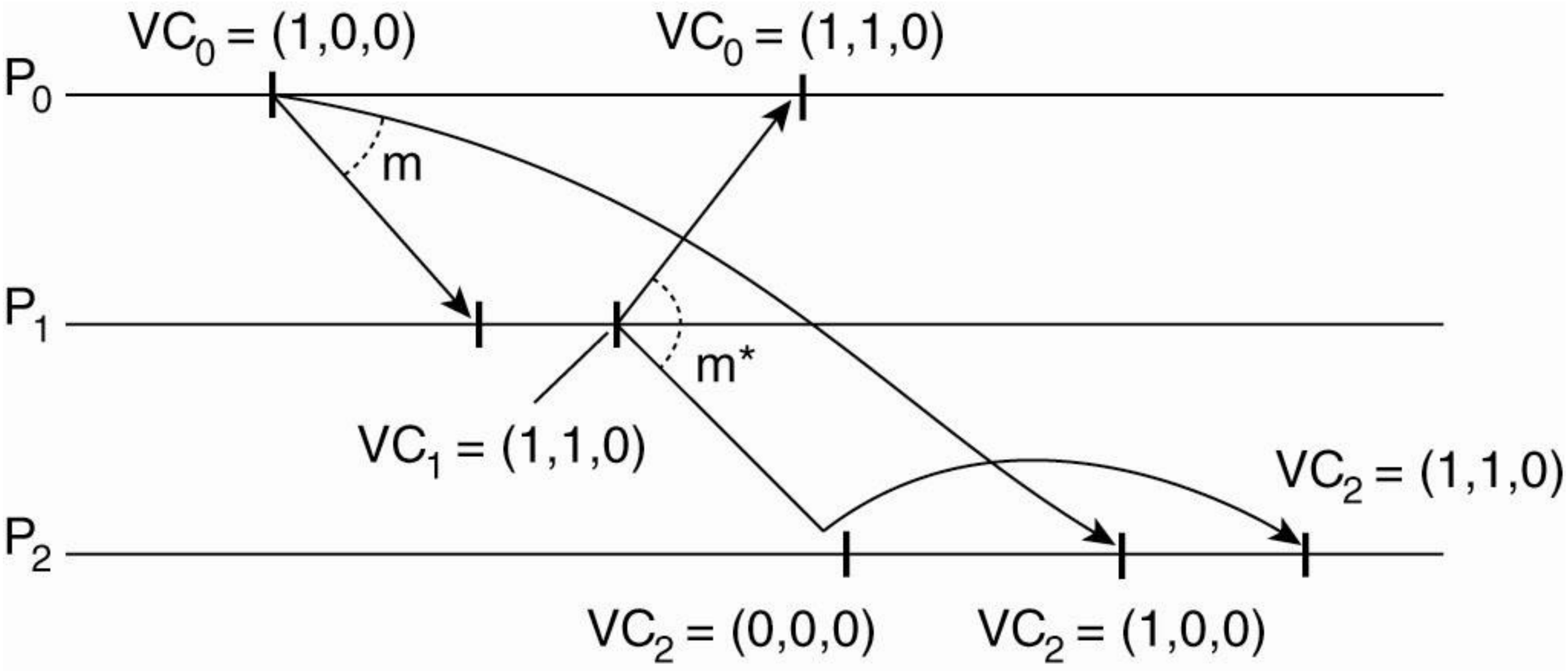
Vector Clocks (3)

- Steps carried out to accomplish property 2 of previous slide:

1. Before executing an event P_i executes $VC_i[\,i\,] \leftarrow VC_i[i\,] + 1$.
2. When process P_i sends a message m to P_j, it sets *m*' s (vector) timestamp *ts (m)* equal to VC_i after having executed the previous step.
3. Upon the receipt of a message m, process P_j adjusts its own vector by setting $VC_j[k\,] \leftarrow \max\{VC_j[k\,], ts(m)[k\,]\}$ for each *k*, after which it executes the first step and delivers the message to the application.

Enforcing Causal Communication

- Enforcing causal communication.



2 conditions:

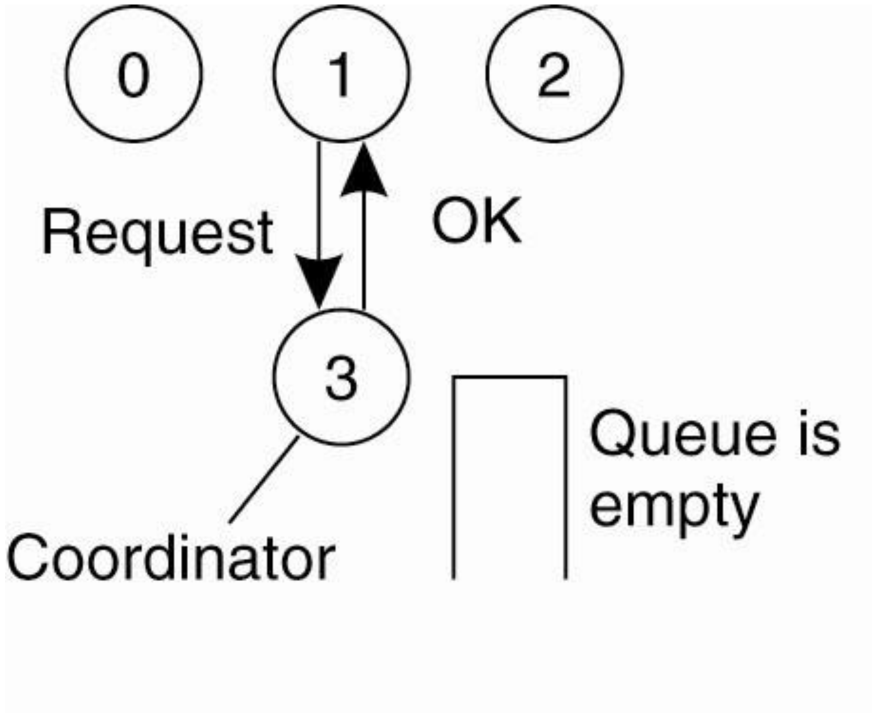
1. $ts(m)[i] = VC_j[i]+1$
2. $ts(m)[k] \leq VC_j[k]$ for all $k \neq i$

3. Mutual exclusion

3. Mutual exclusion algorithms

- Classification of Mutual exclusion algorithms:
  - Permission-based approach
    - A Centralized Algorithm
    - A Distributed Algorithm
  - Token-based solutions
    - Token Ring Algorithm
- Problems:
  - Starvation
  - Deadlocks
  - Token loss (for token-based approach)

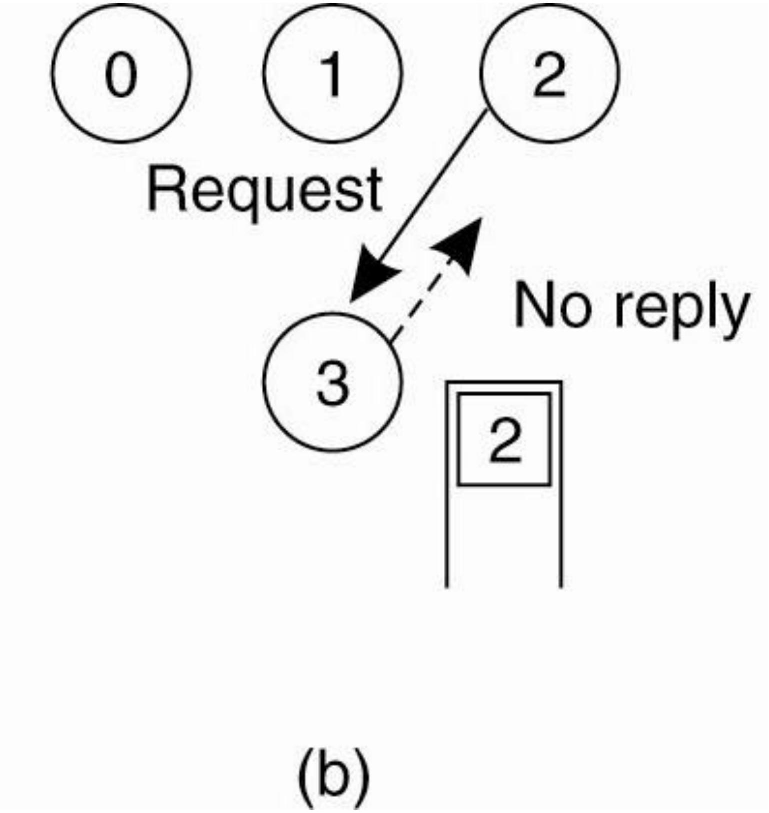
Mutual Exclusion
3.1. Centralized Algorithm (1)



(a)

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

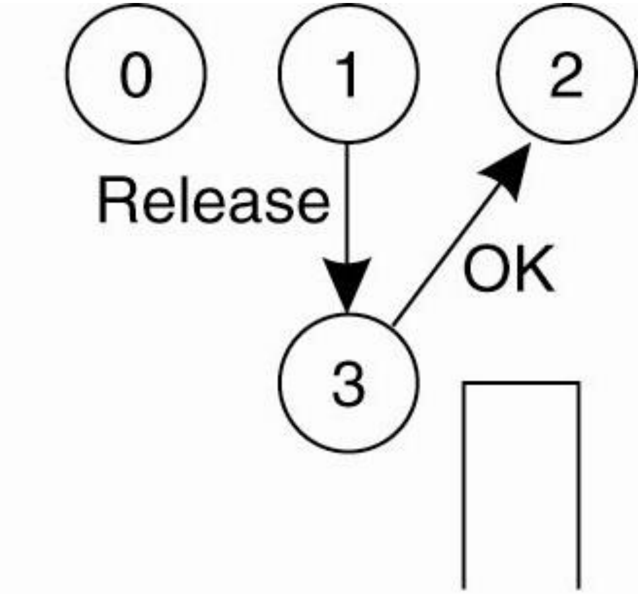
Mutual Exclusion
Centralized Algorithm (2)



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

Mutual Exclusion
Centralized Algorithm (3)

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



(c)

Algorithm analysis

- Advantages
  - Simplicity
  - Resolve both *starvation* and *deadlocks* problems

- Disadvantages
  - Single point of failure
  - Performance bottleneck

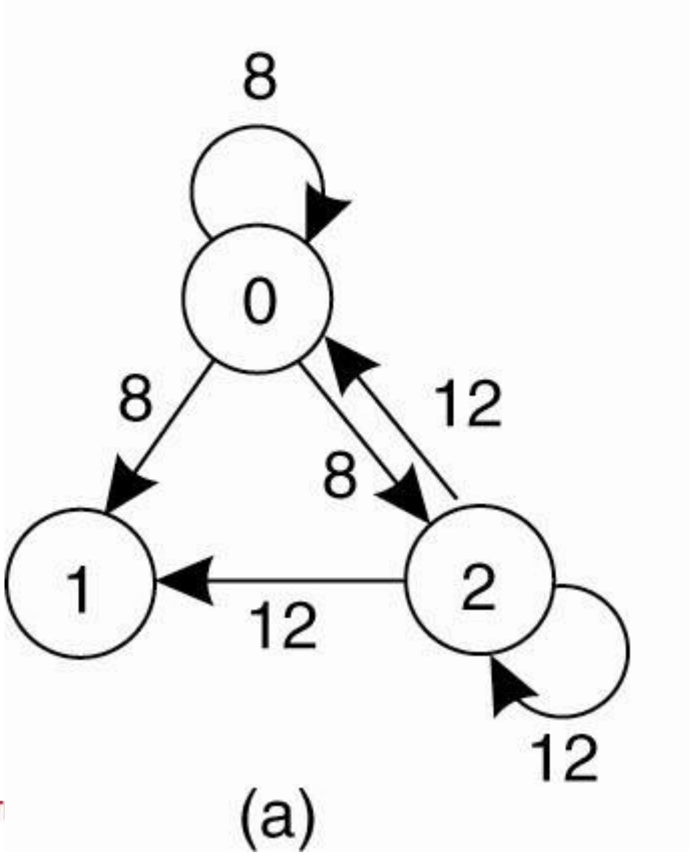
3.2. A Distributed Algorithm (1)

- Three different cases:

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.

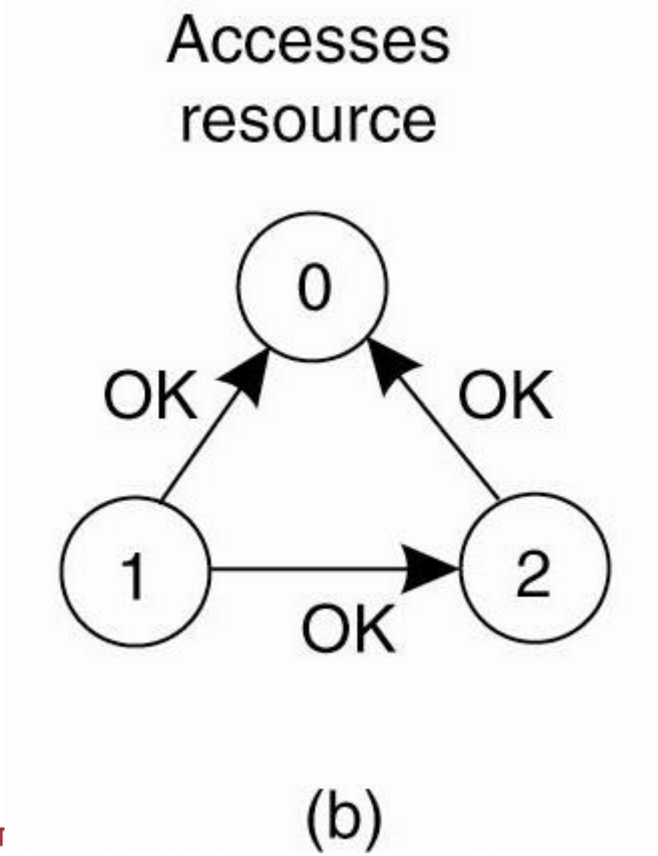
A Distributed Algorithm (2)

- Two processes want to access a shared resource at the same moment.



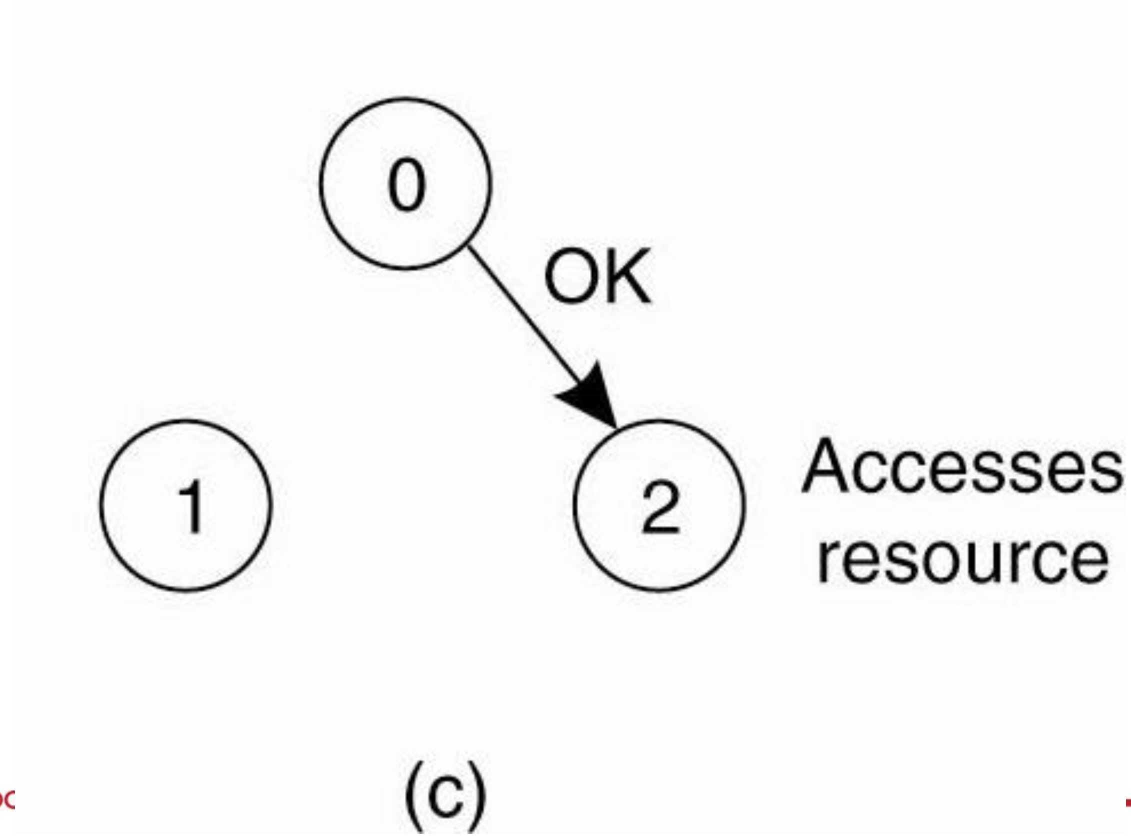
A Distributed Algorithm (3)

- Process 0 has the lowest timestamp, so it wins.



A Distributed Algorithm (4)

- When process 0 is done, it sends an OK also, so 2 can now go ahead.

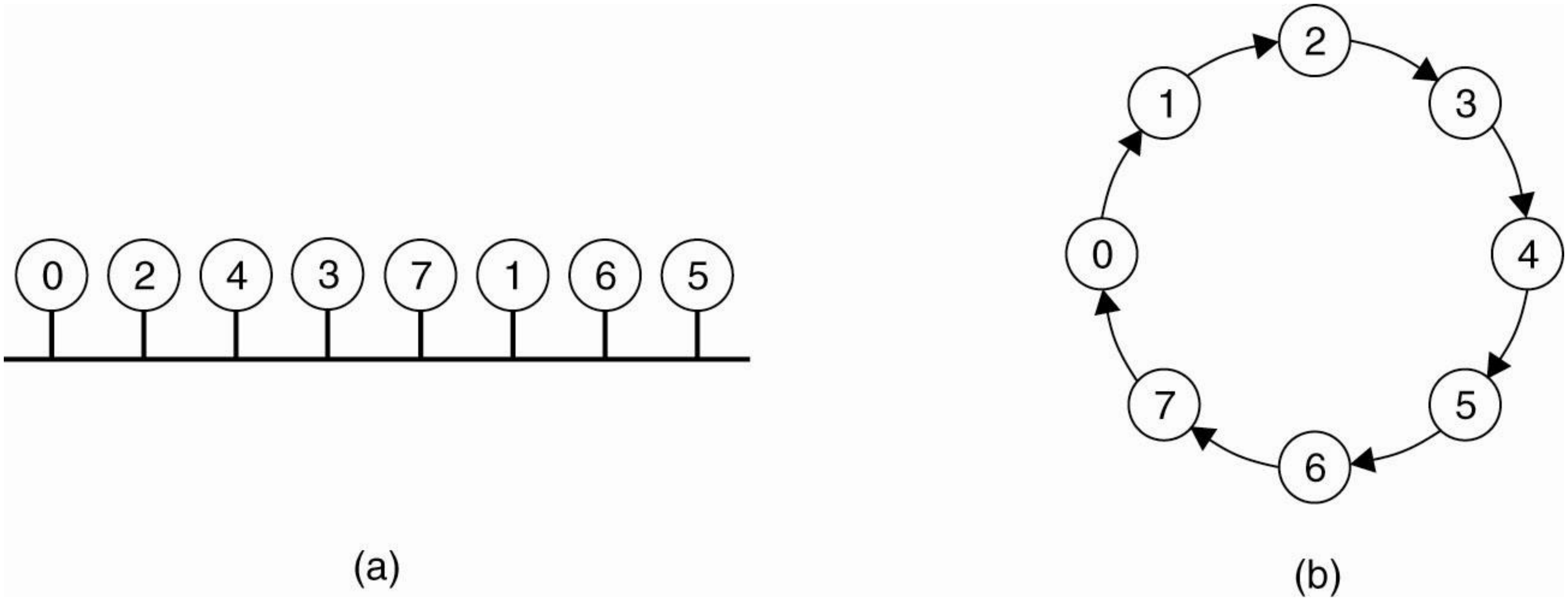


Algorithm analysis

- Advantages
  - Avoidance of deadlock and starvation problems

- Disadvantages:
  - Complicated
  - *n* points of failure
  - Broadcasting mechanism → scalability
  - More expensive, but less robust than the centralized algo

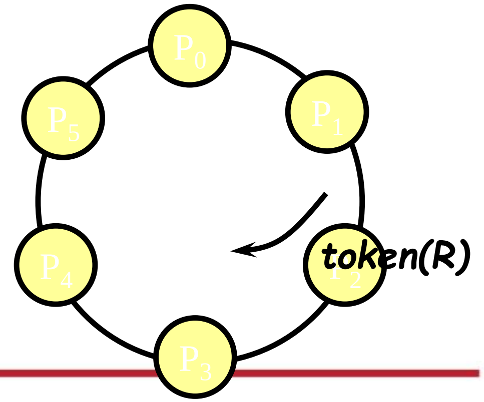
3.3. A Token Ring Algorithm



- (a) An unordered group of processes on a network.
(b) A logical ring constructed in software.

Token Ring algorithm

- Initialization
  - Process 0 gets token for resource R
- Token circulates around ring
  - From P_i to $P_{(i+1)}$mod N
- When process acquires token
  - Checks to see if it needs to enter critical section
  - If no, send token to neighbor
  - If yes, access resource
    - Hold token until done



Algorithm analysis

- Advantages
  - Avoidance of deadlock and starvation problems

- Disadvantages:
  - Token loss

4. Election algorithms

4. Election Algorithms

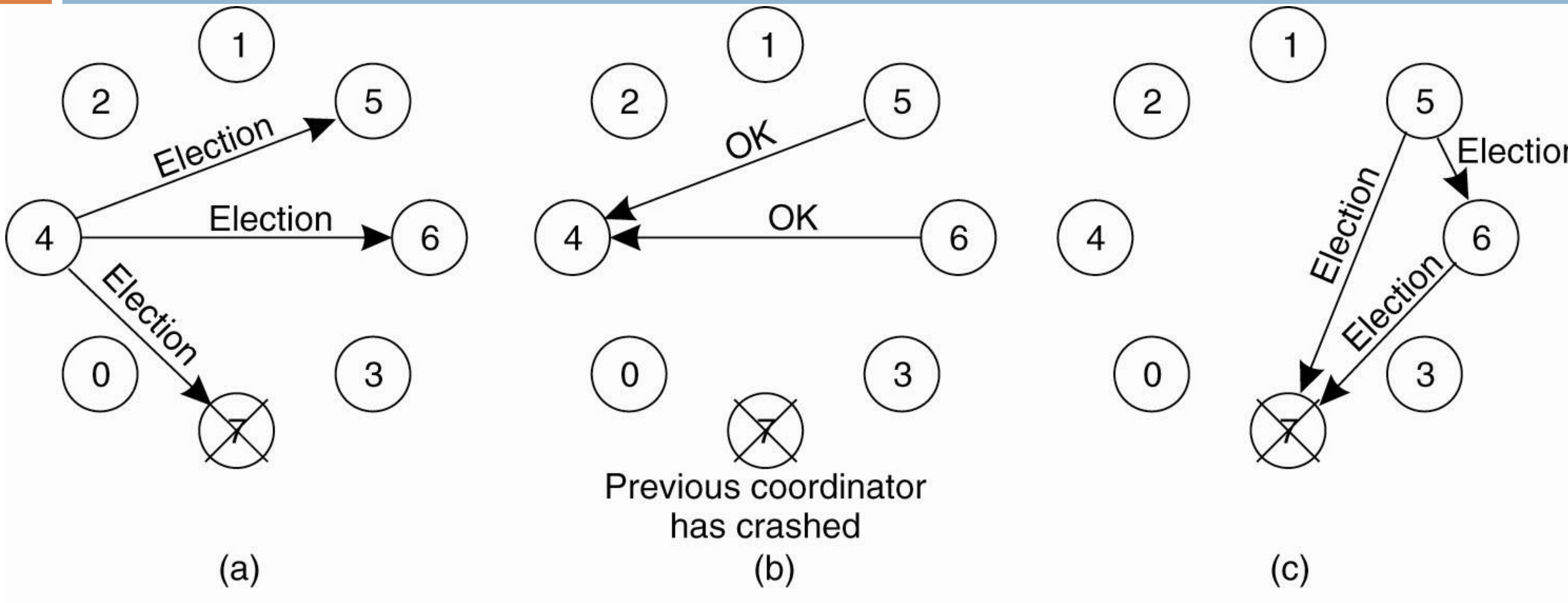
- Traditional Election algorithms
  - The Bully Algorithm
  - A Ring Algorithm
- Election in Wireless Environments
- Election in Large-Scale Systems

Election Algorithms

- The Bully Algorithm

1. *P* sends an *ELECTION* message to all processes with higher numbers.
2. If no one responds, *P* wins the election and becomes coordinator.
3. If one of the higher-ups answers, it takes over. *P*'s job is done.

The Bully Algorithm (1)

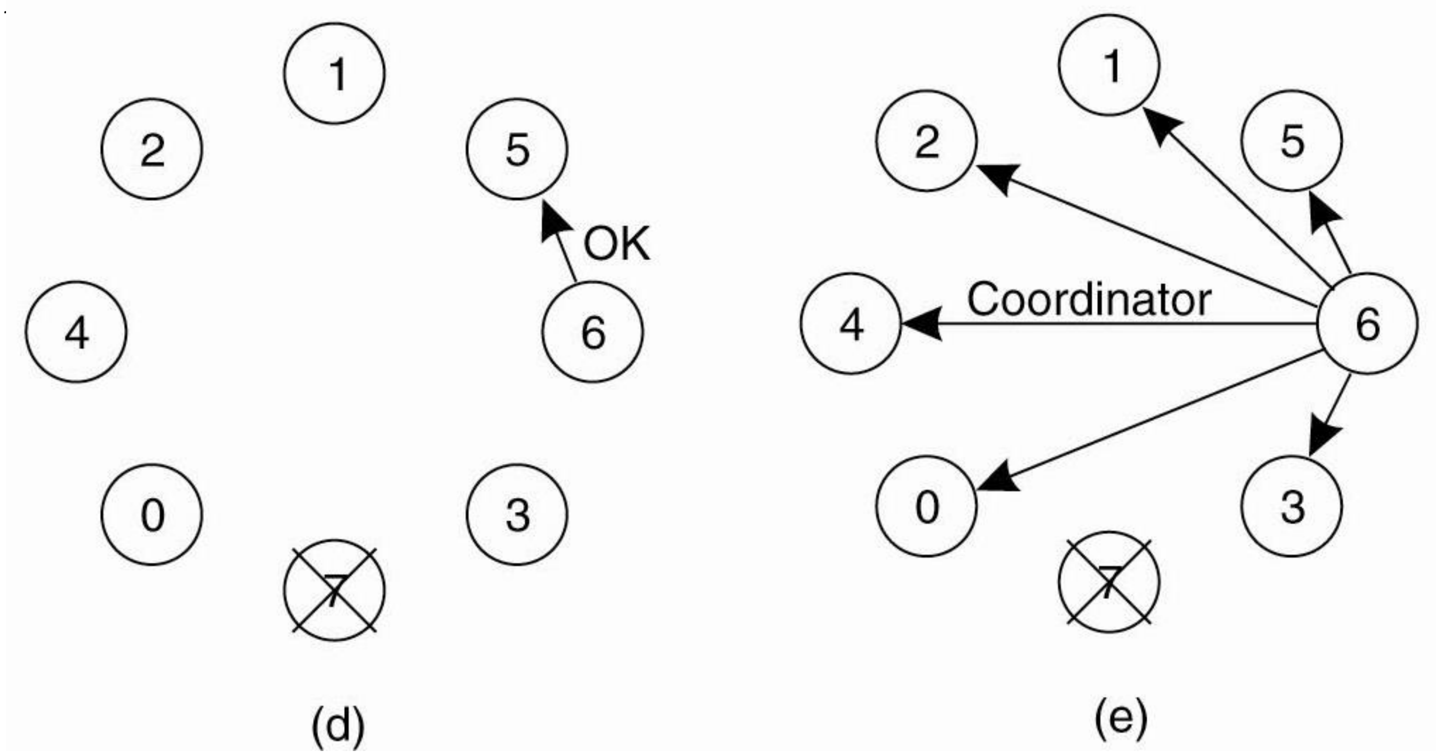


- The bully election algorithm. (a) Process 4 holds an
- election. (b) Processes 5 and 6 respond, telling 4 to stop.

(c) Now 5 and 6 each hold an election.

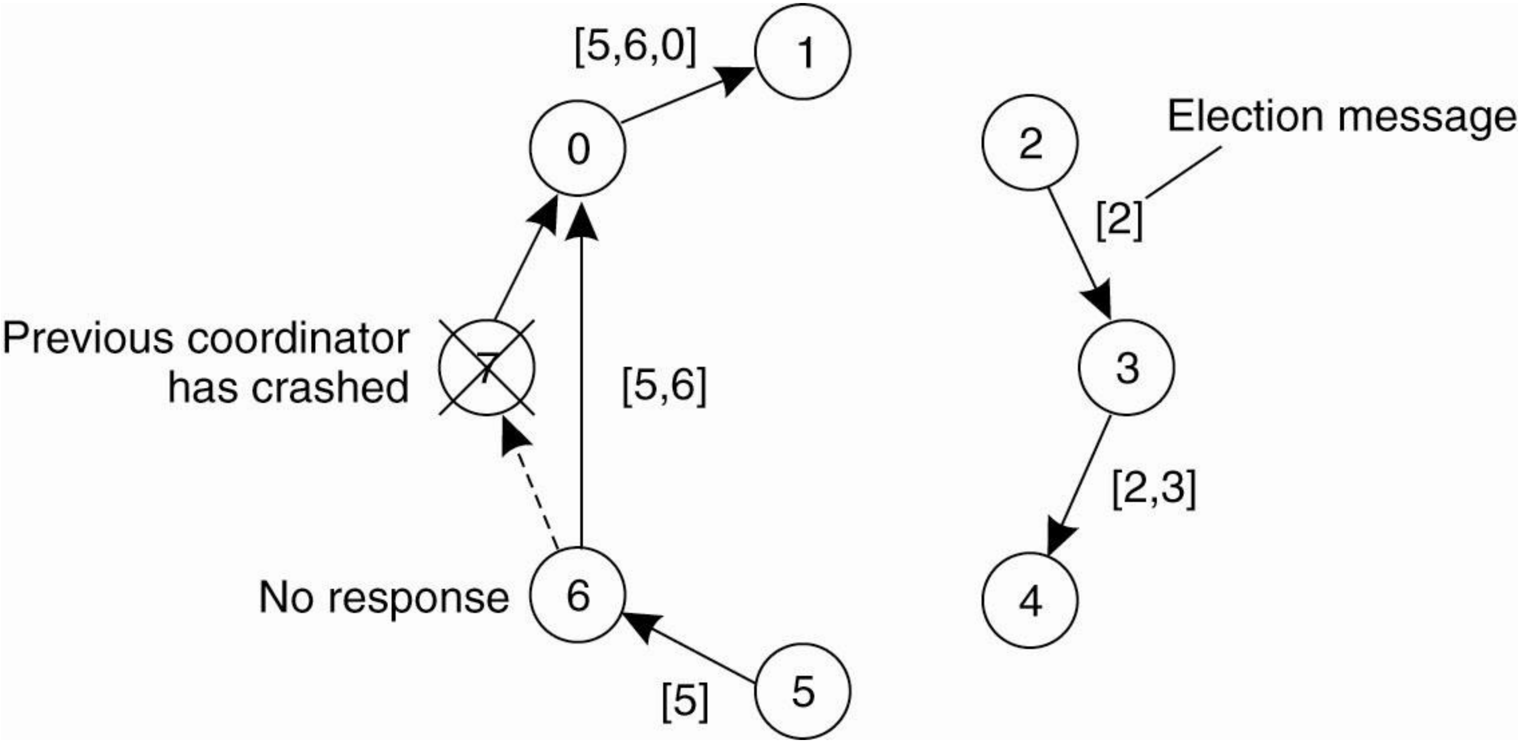
The Bully Algorithm (2)

- The bully election algorithm. (d) Process 6 tells 5

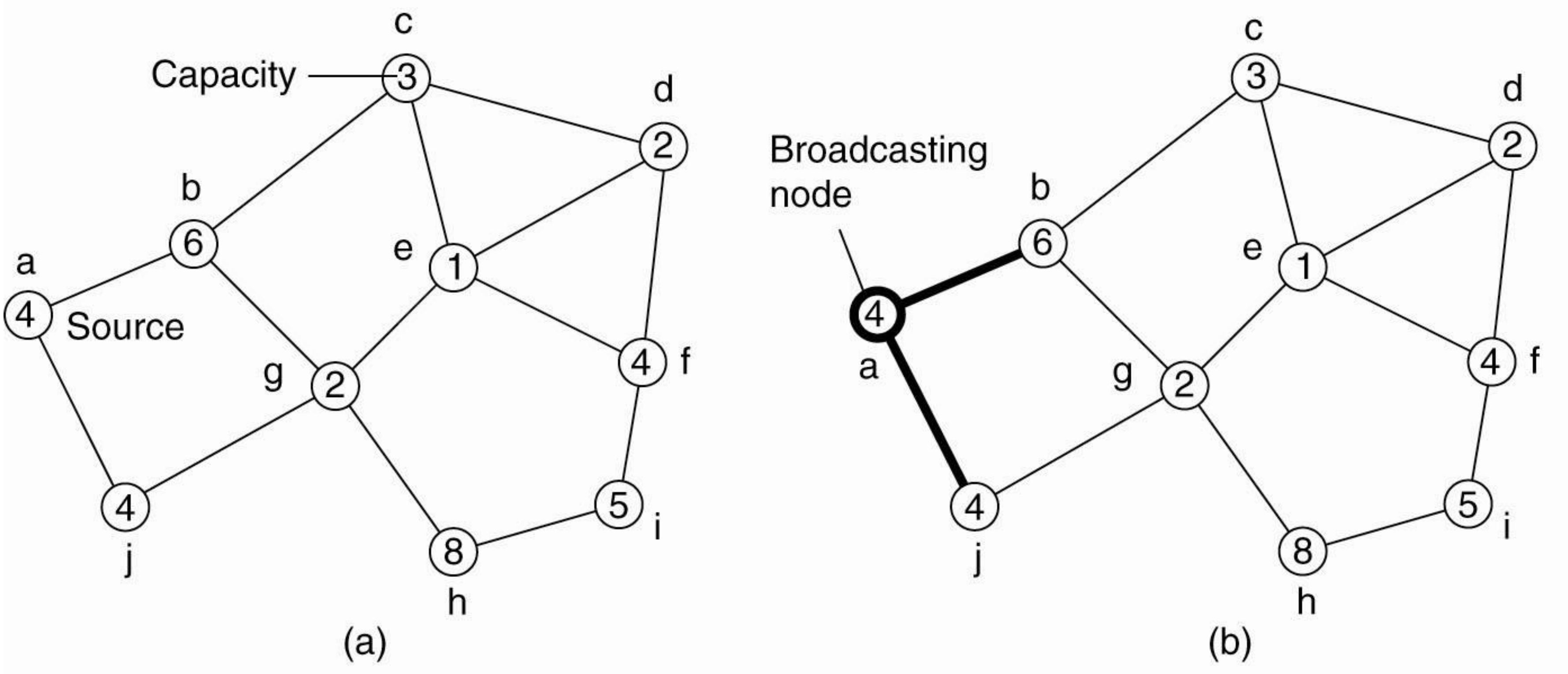


A Ring Algorithm

- Election algorithm using a ring.

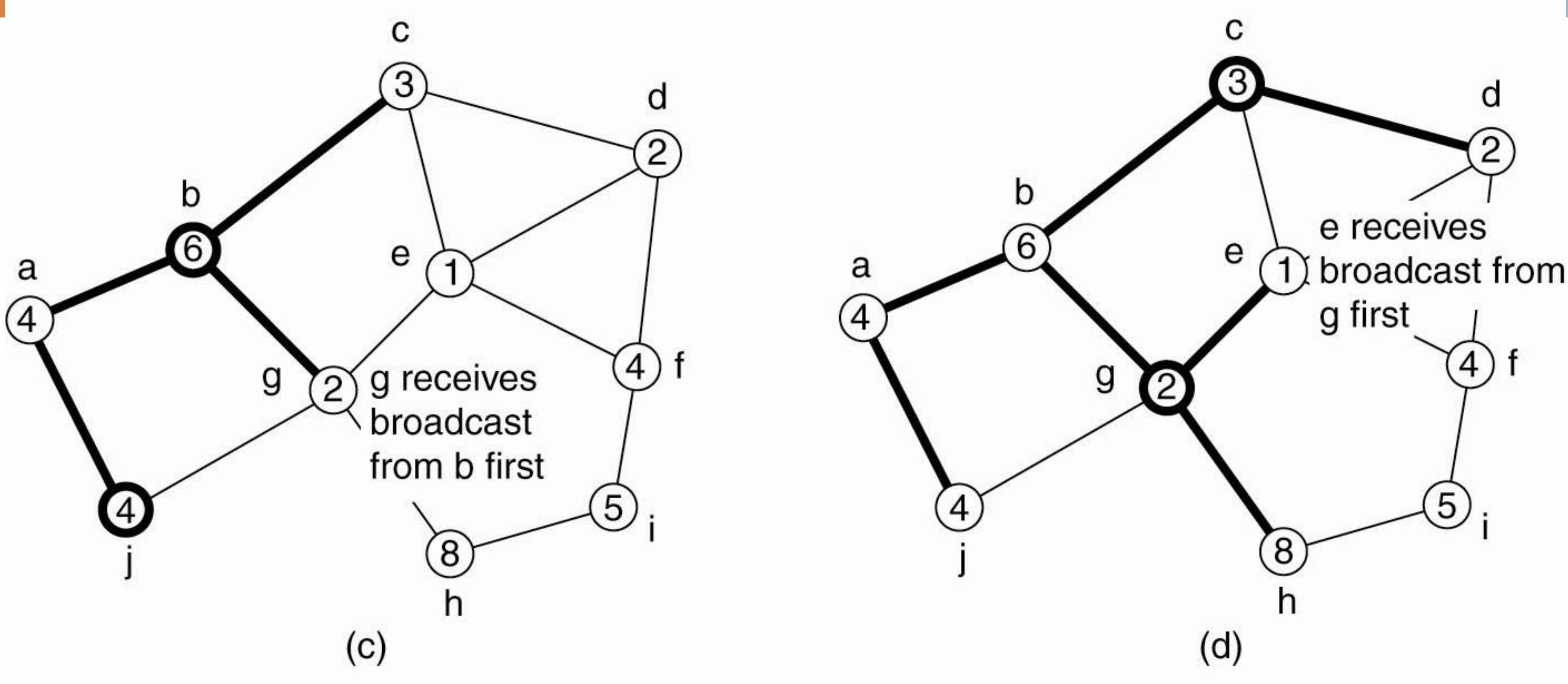


Elections in Wireless Environments (1)



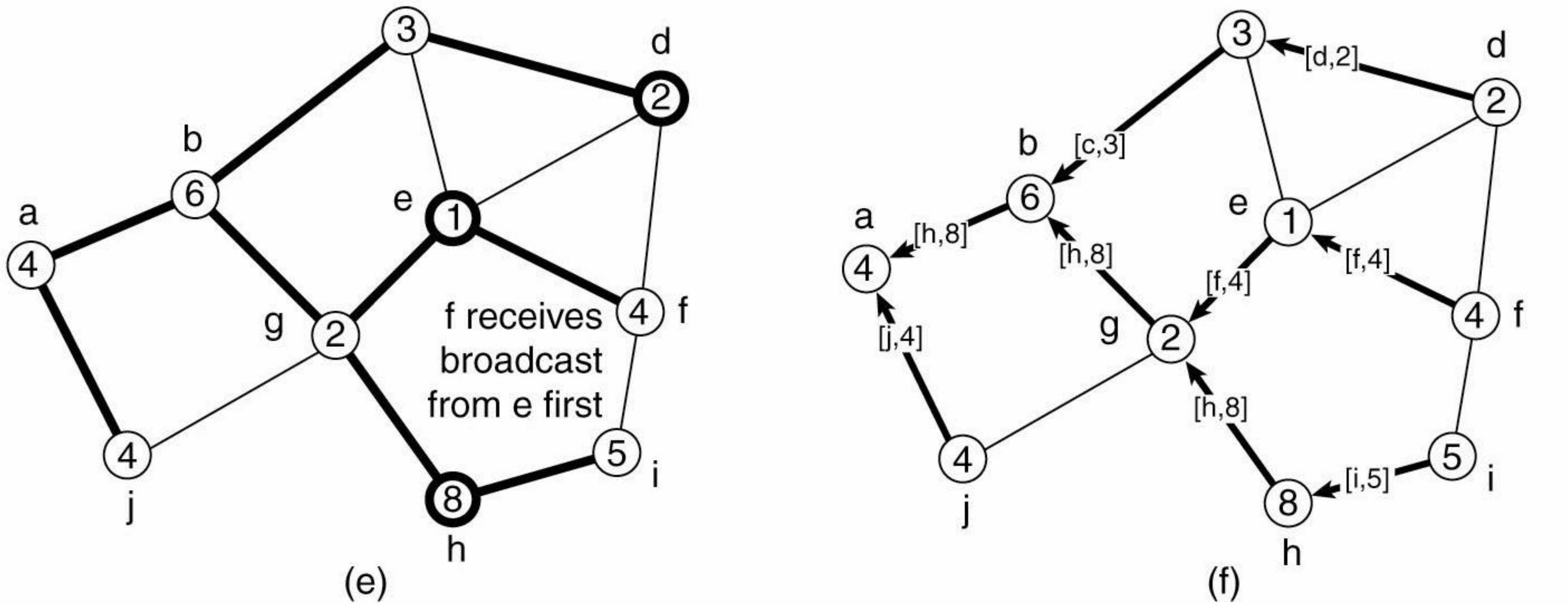
- Election algorithm in a wireless network, with node a as the source. (a) Initial network. (b)–(e) The build-tree phase

Elections in Wireless Environments (2)



Elections in Wireless Environments (3)

- (e) The build-tree phase.
  (f) Reporting of best node to source.

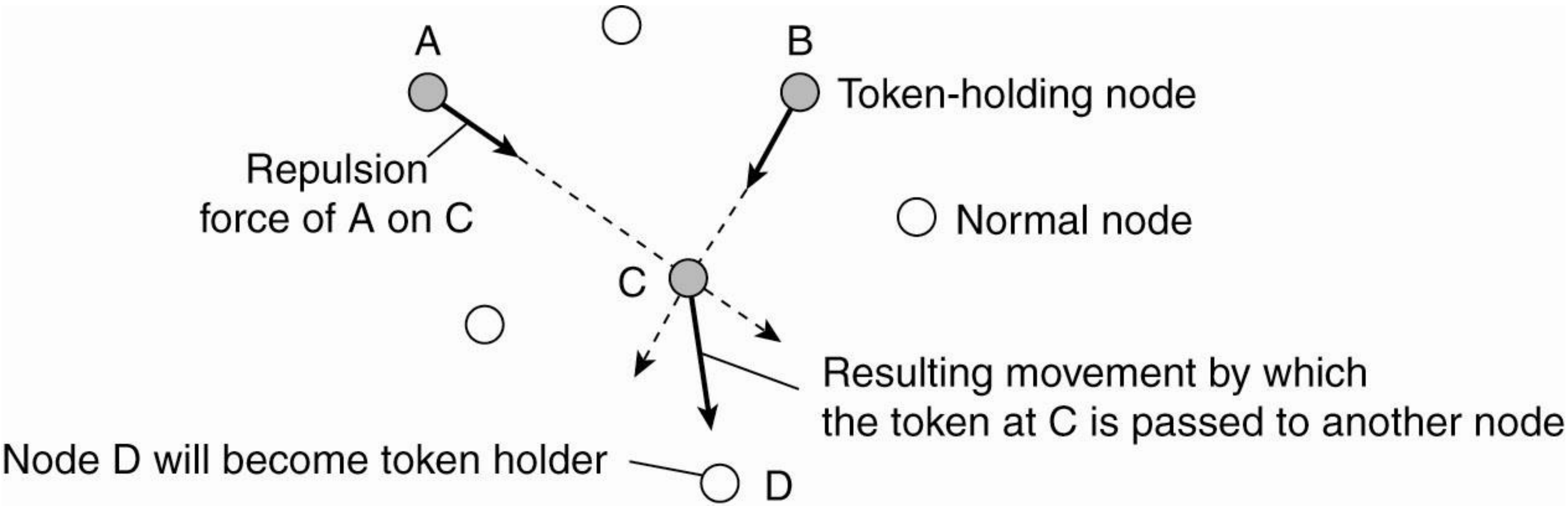


Elections in Large-Scale Systems (1)

- Requirements for superpeer selection:

1. Normal nodes should have low-latency access to superpeers.
2. Superpeers should be evenly distributed across the overlay network.
3. There should be a predefined portion of superpeers relative to the total number of nodes in the overlay network.
4. Each superpeer should not need to serve more than a fixed number of normal nodes.

Elections in Large-Scale Systems (2)



- Moving tokens in a two-dimensional space using repulsion forces.

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Questions ?