Distributed Operating Systems

Synchronization

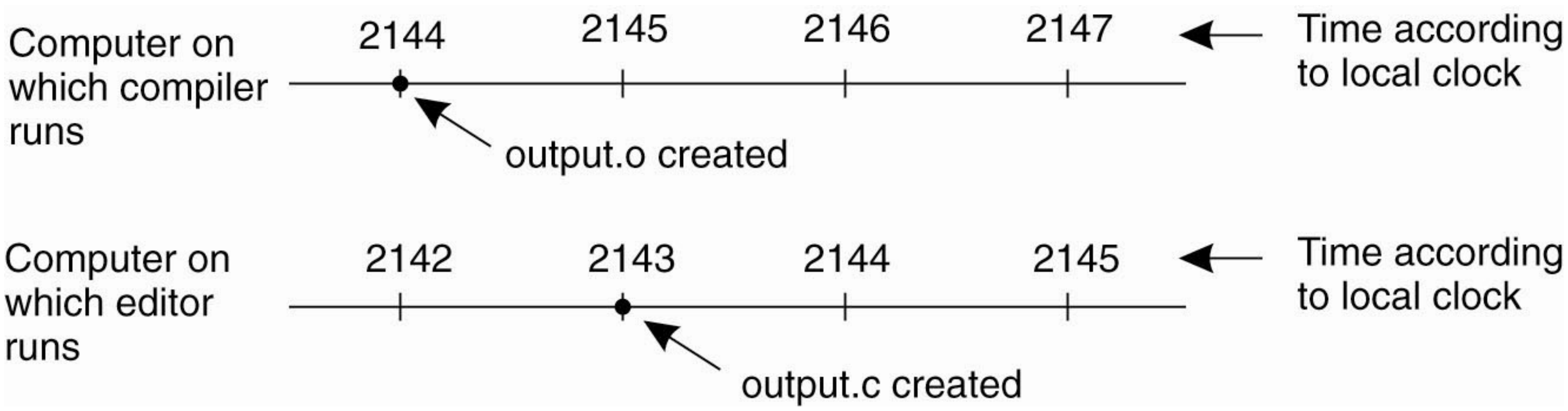
Topics

- Solar Time vis-à-vis Physical Time.
- Clock Synchronization Algorithms
  - Network Time Protocol
  - The Berkeley Algorithm
- Logical Clocks
  - Lamport's Logical Clocks
  - Vector Clocks
- Mutual Exclusion: A Centralized Algorithm
  - Distributed Algorithm
  - Token Ring Algorithm
- Election Algorithms

Synchronization in Distributed Systems

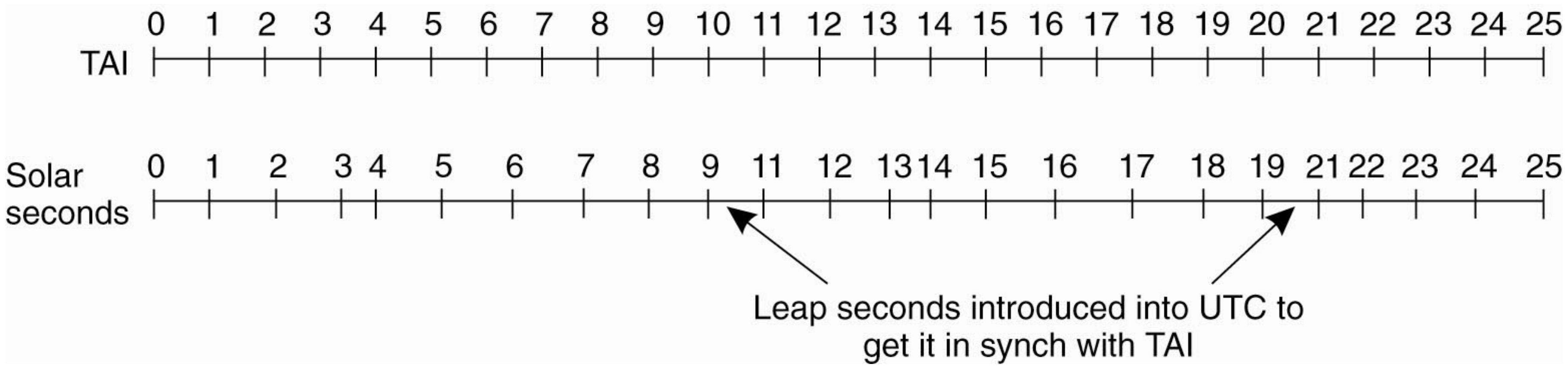
- It is important that multiple processes do not simultaneously access a shared resource, such as printer, but instead cooperate in granting each other temporary exclusive access.
- Besides, multiple processes may sometimes need to agree on the ordering of events, such as whether message m1 from process P was sent before or after message m2 from process Q

Clock Synchronization


Computer on which compiler runs
2144
2145
2146
2147
Time according to local clock
output.o created
Computer on which editor runs
2142
2143
2144
2145
Time according to local clock
output.c created

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

Physical Clocks



TAI (International Atomic Time) seconds are of constant length, unlike solar seconds.

Leap seconds are introduced when necessary.

Physical and Solar Clocks

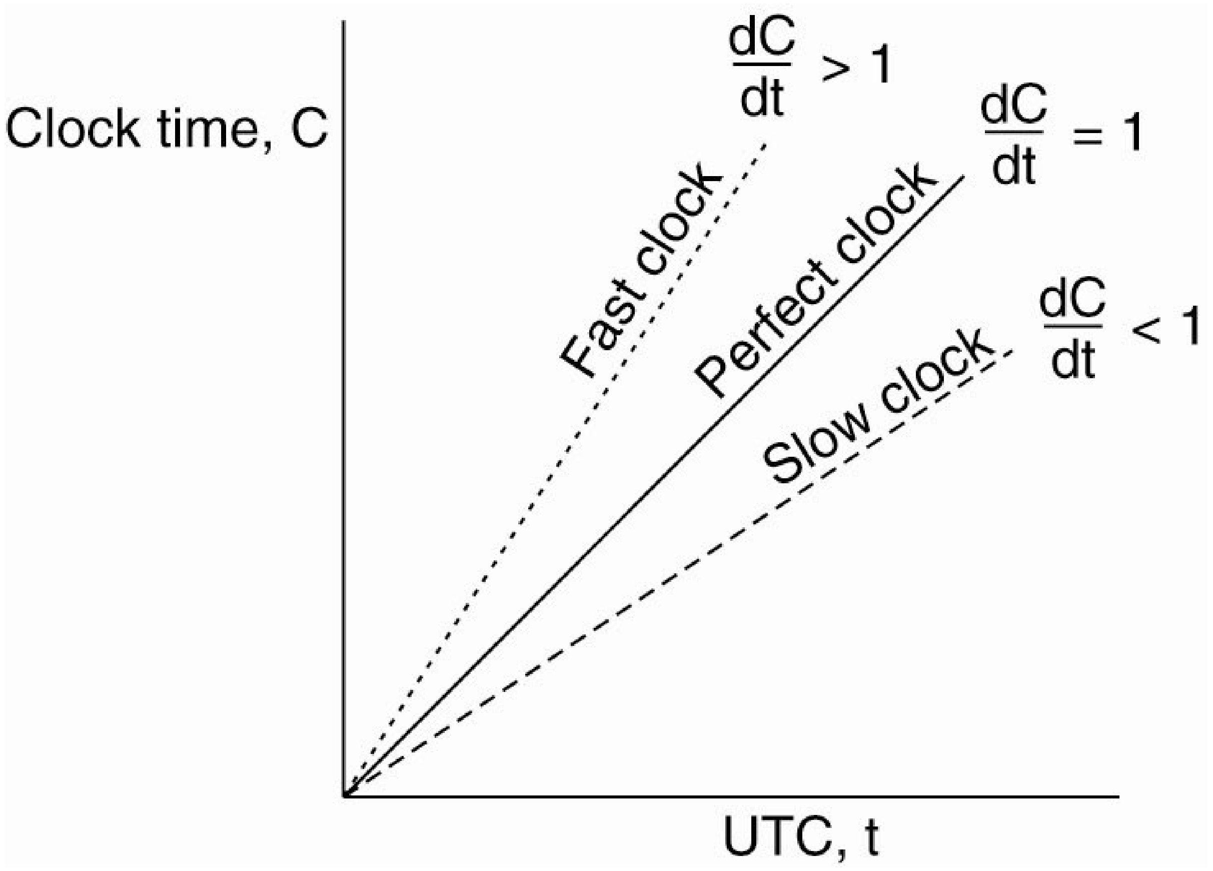
- The difference between physical second and solar second is corrected by inserting a leap second.
- This correction gives rise to a time system based on constant International Atomic Time (which is abbreviated TAl) seconds but which stays in phase with the apparent motion of the sun.
- The universal time is called Universal Coordinated Time, but is abbreviated as UTC.
- Currently, several laboratories around the world have cesium 133 clocks.
- Periodically, each laboratory tells the Bureau International de l'Heure (BIR) in Paris how many times its clock has ticked.
- The BIR averages these to produce TAl.
- Thus TAI is just the mean number of ticks of the cesium 133 clocks since midnight on January

Broadcasting Time Signal

- To provide UTC to people which need precise time, the National Institute of Standard Time (NIST) operates a shortwave radio station with call letters WWV from Fort Collins, Colorado.
- WWV broadcasts a short pulse at the start of each UTC second.
- The accuracy of WWV is ±1 msec.
- Due to random atmospheric fluctuations, in practice the accuracy is no better than ±10 msec.

Clock Synchronization Algorithms

The relation between clock time and UTC when clocks tick at different rates.



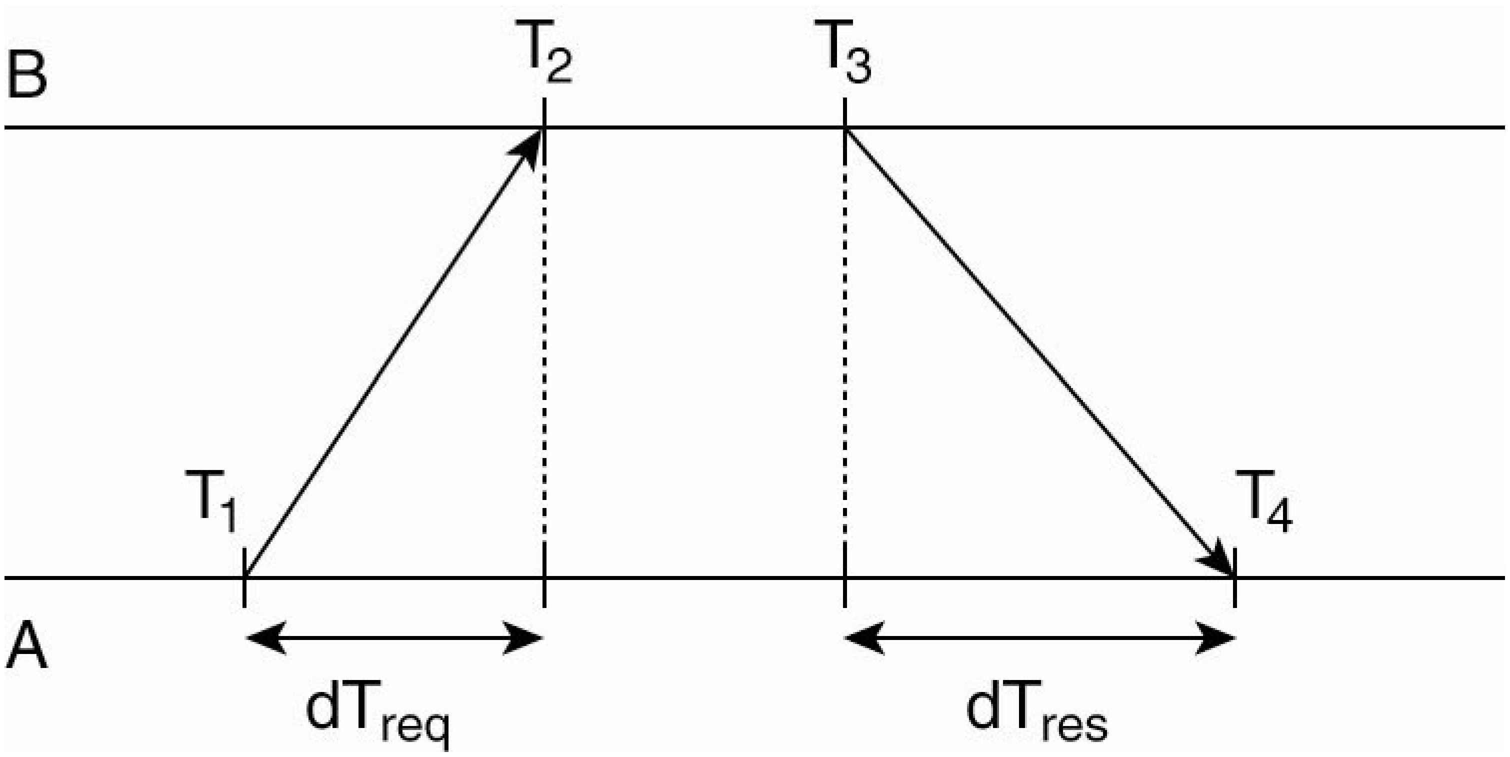
Clock Synchronization Algorithms

- If one machine has a WWV receiver, the goal becomes keeping all the other machines synchronized to it.
- If no machines have WWV receivers, each machine keeps track of its own time, and the goal is to keep all the machines together as well as possible.
- Many clock synchronization algorithms have been proposed for this purpose.

Network Time Protocol (1)

- A common approach in many protocols and originally proposed by Cristian (1989) is to let clients contact a time server.
- The server can accurately provide the current time, for example, because it is equipped with a WWV receiver or an accurate clock.
- The problem is that when contacting the server, message delays will have outdated the reported time.
- The trick is to find a good estimation for these delays.

Network Time Protocol (2)

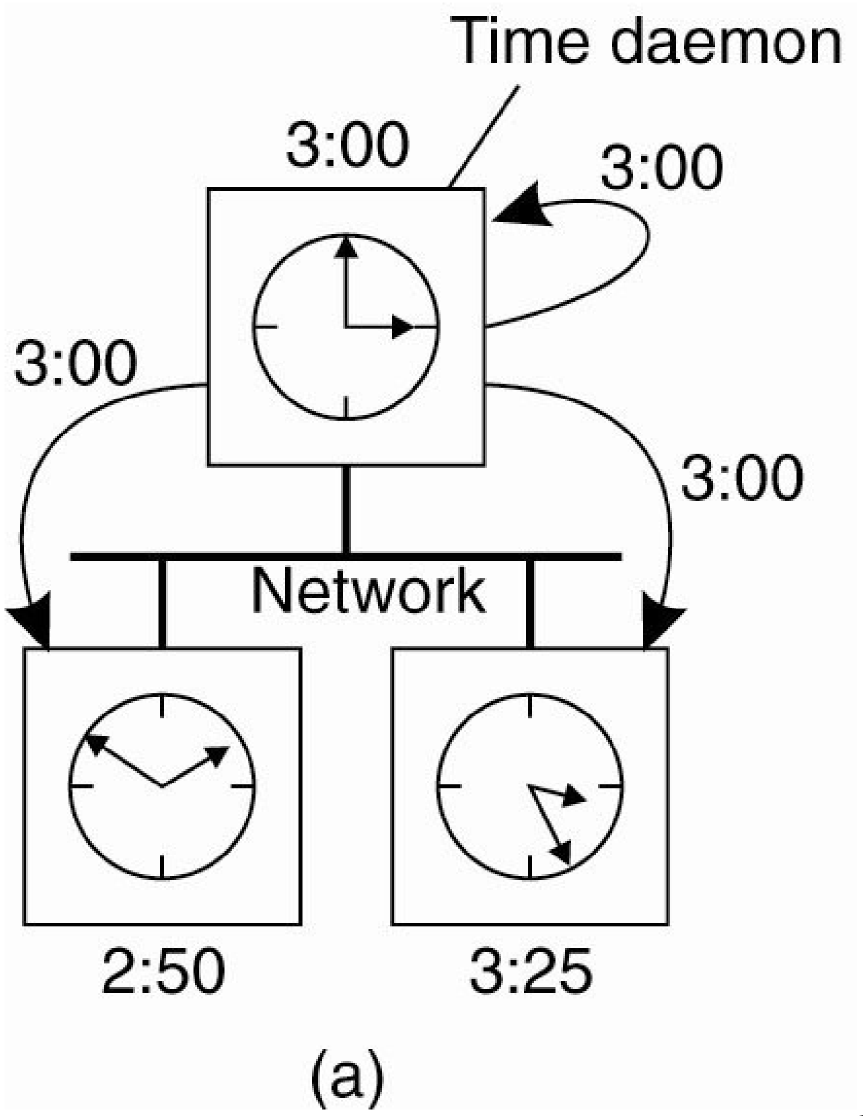


The Berkeley Algorithm (1)

- In many algorithms the time server is passive. Other machines periodically ask it for the time. All it does is respond to their queries.
- In Berkeley UNIX, the time server (actually, a time daemon) is active, polling every machine from time to time to ask what time it is there.
- Based on the answers, it computes an average time and tells all the other machines to advance their clocks to the new time or slow their clocks down until some specified reduction has been achieved.

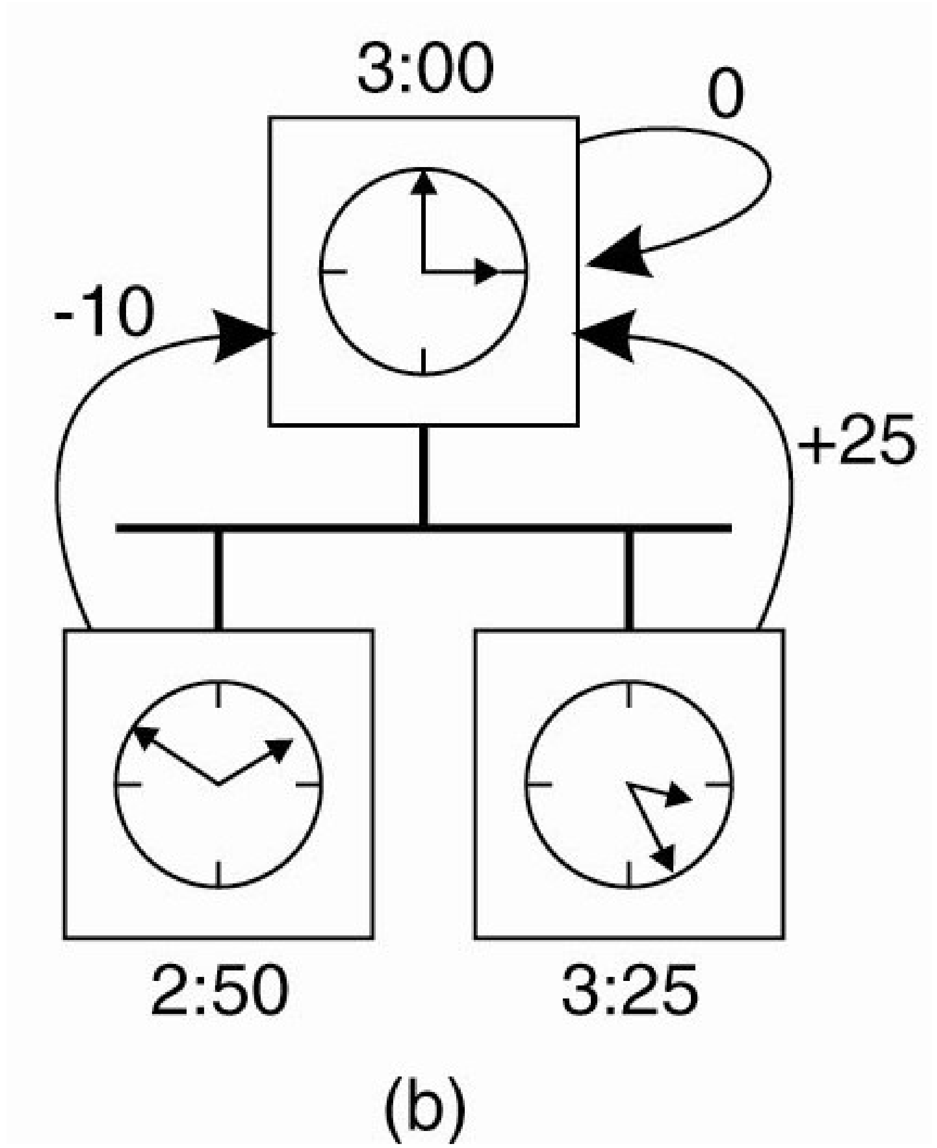
The Berkeley Algorithm (1)

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



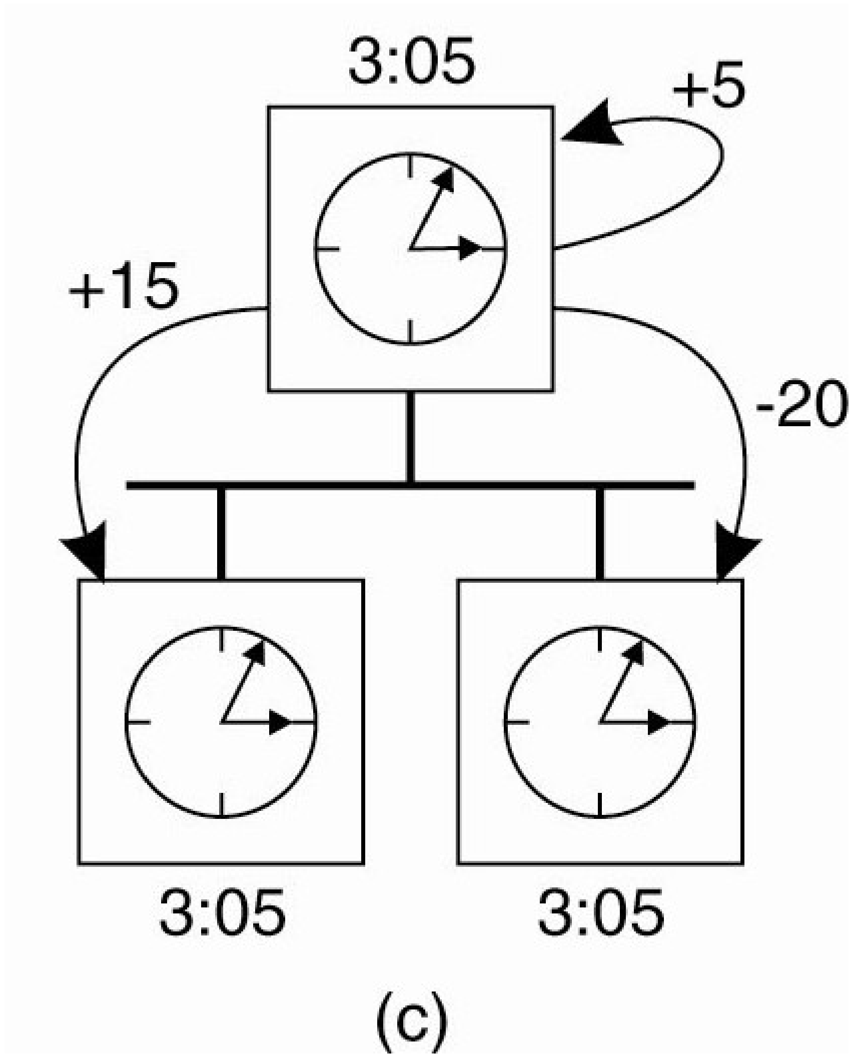
The Berkeley Algorithm (2)

The machines answer.



The Berkeley Algorithm (3)

The time daemon tells everyone how to adjust their clock.



Logical Clocks

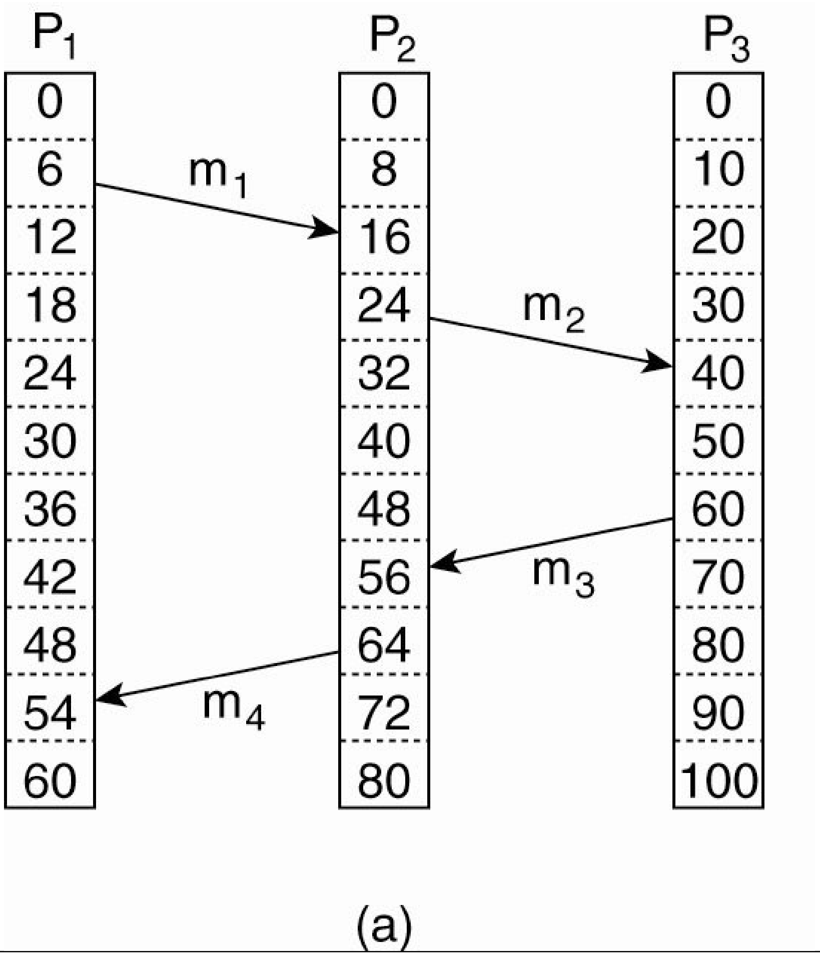
- In some case it is sufficient that every node agrees on a current time, without that time necessarily being the same as the real time.
- We can also claim that if two processes do not interact, it is not necessary that their clocks be synchronized because the lack of synchronization would not be observable and thus could not cause problems.
- Furthermore, what usually matters is not that all processes agree on exactly what time it is, but rather that they agree on the order in which events occur.

Lamport's Logical Clocks (1)

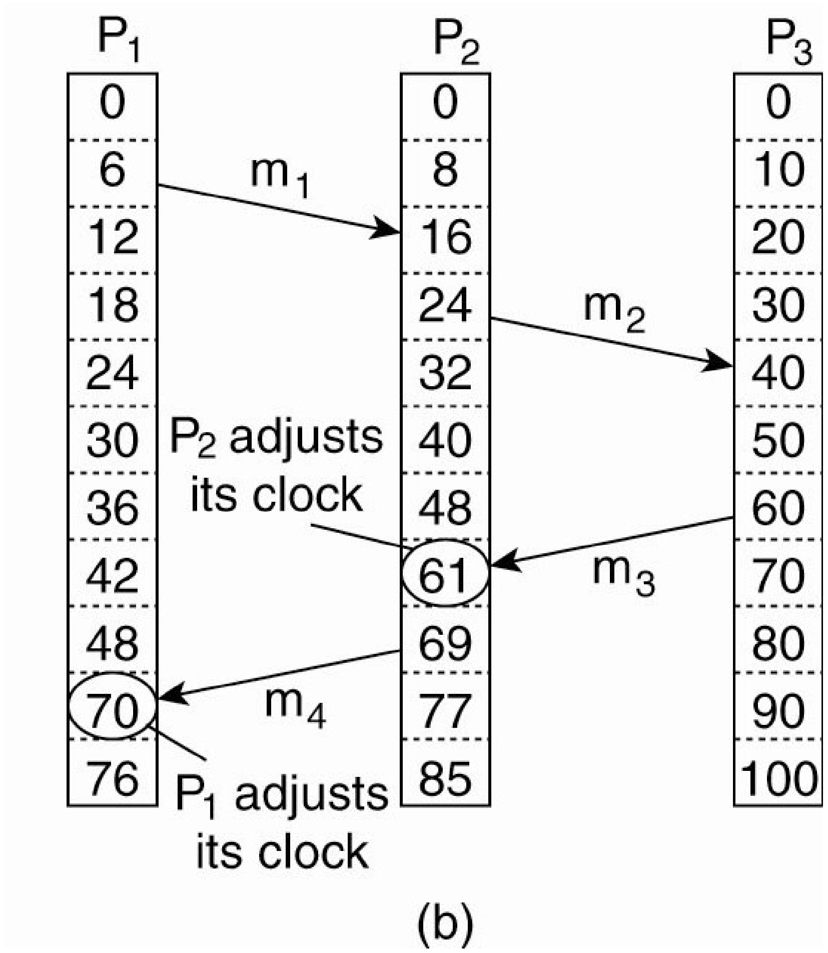
- The "happens-before" relation $\rightarrow$ can be observed directly in two situations:
- If *a* and *b* are events in the same process, and *a* occurs before *b*, then $a \rightarrow b$ is true.
- If a is the event of a message being sent by one process, and *b* is the event of the message being received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

Lamport's Logical Clocks (2)

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

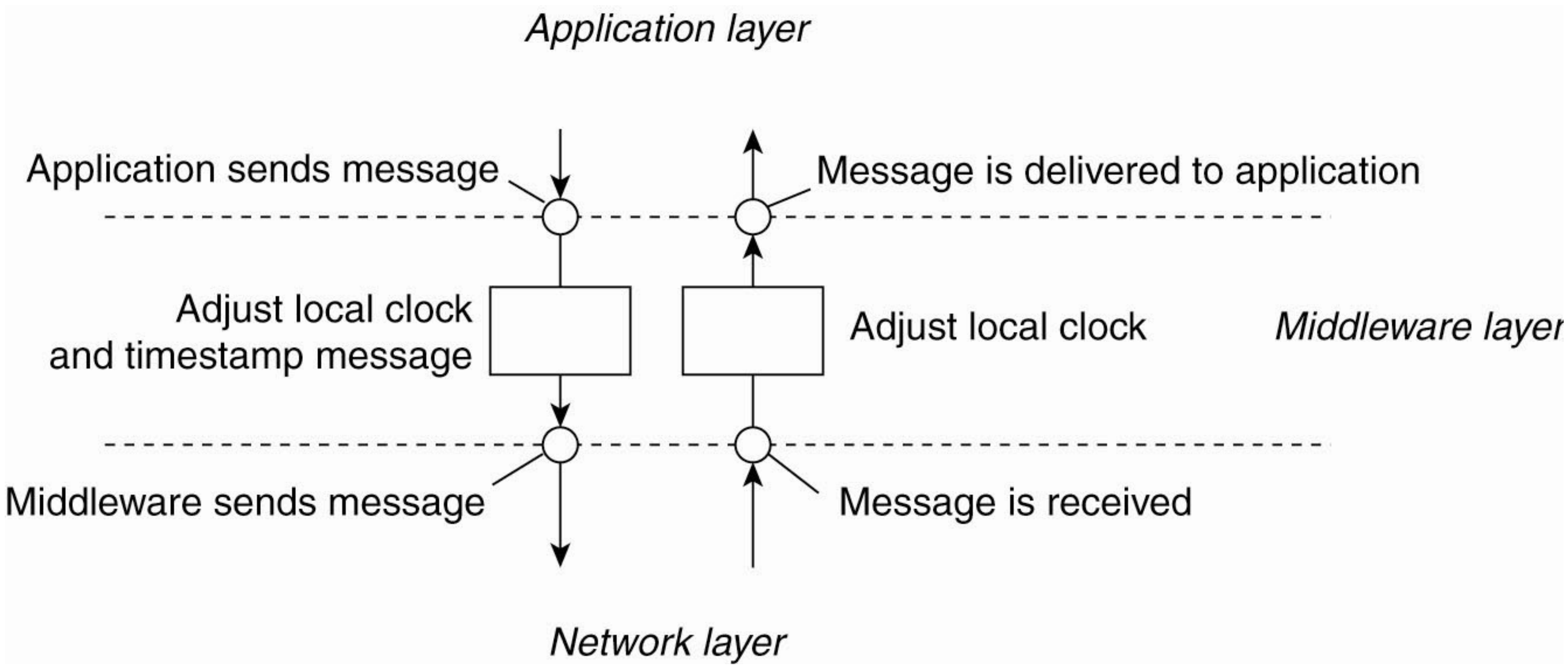


Lamport's Logical Clocks (3)



Lamport's algorithm corrects the clocks.

Lamport’s Logical Clocks (4)

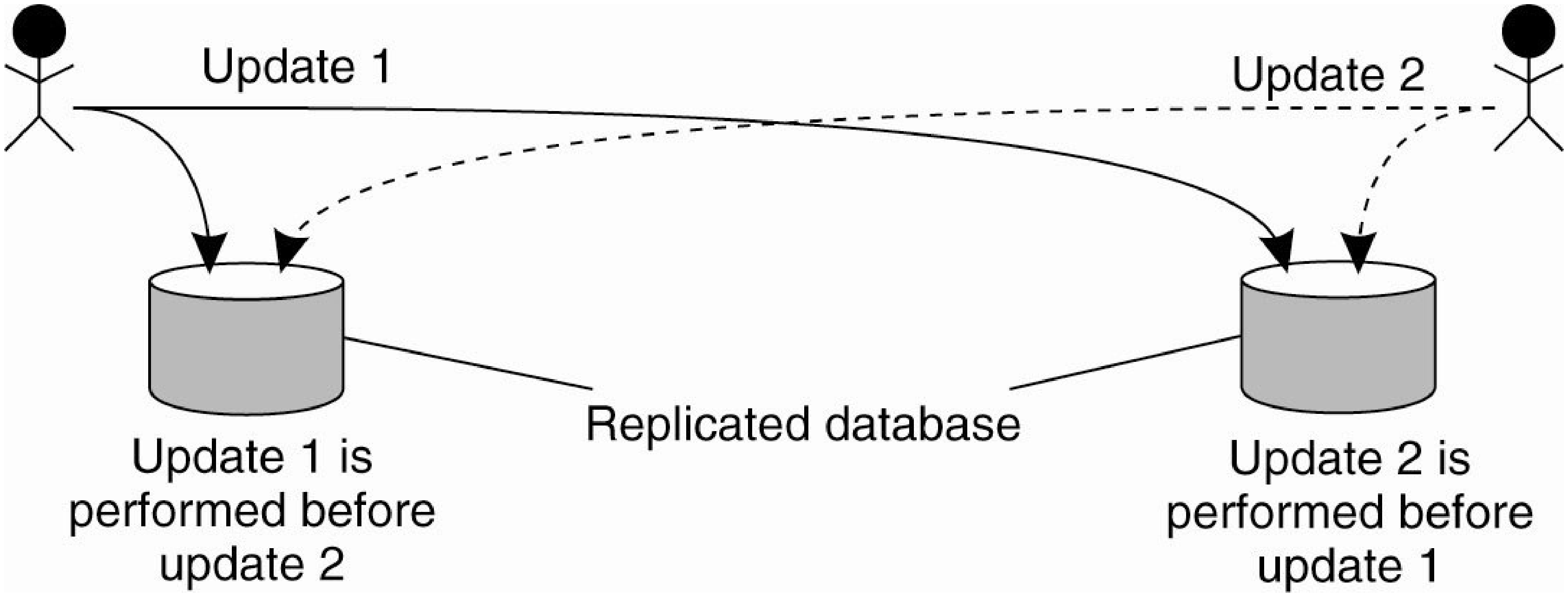


Lamport's Logical Clocks (5)

- 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.

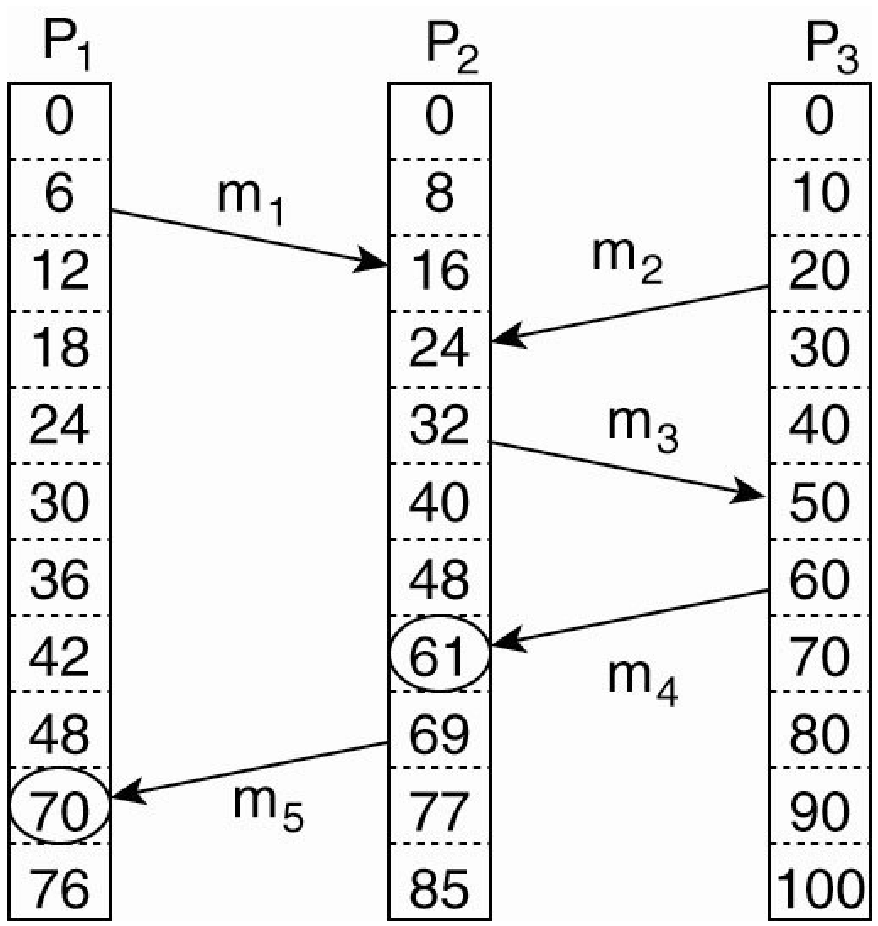
Example: Totally Ordered Multicasting


Update 1
Update 2
Replicated database
Update 1 is performed before update 2
Update 2 is performed before update 1

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

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 .

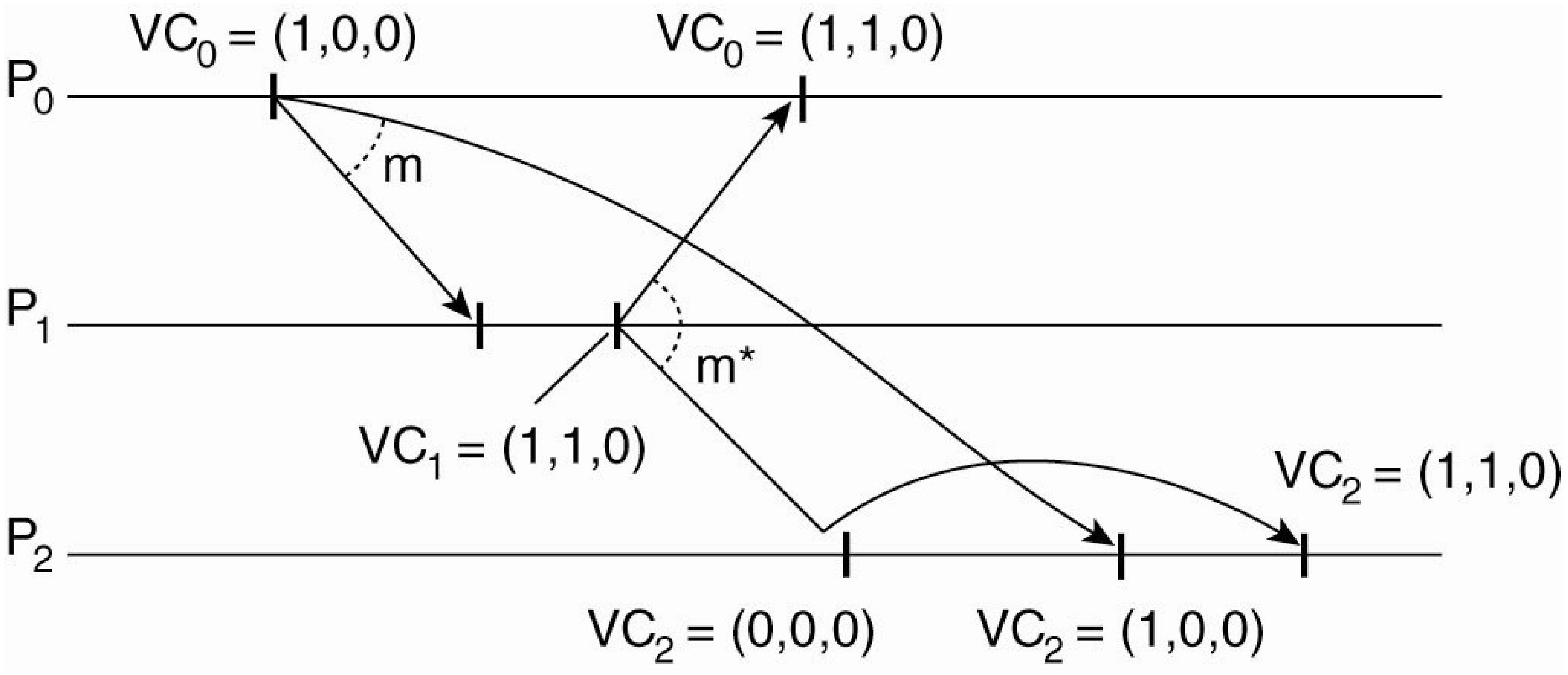
Vector Clocks (3)

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

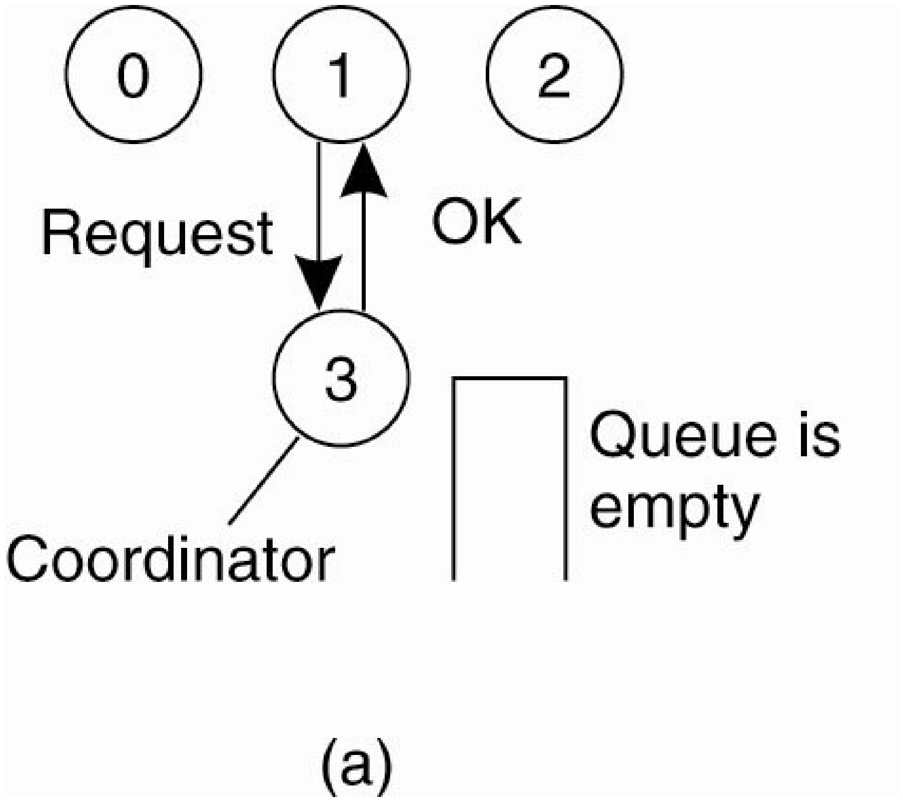
1. Before executing an event P_i executes
   $VC_i[\,i\,] \leftarrow VC_i[i\,] + 1$.
2. When process P_i sends a message m to P_j, it sets *m*'s (vector) timestamp *ts (m)* equal to VC_i after having executed the previous step.
3. Upon the receipt of a message m, process P_j adjusts its own vector by 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.

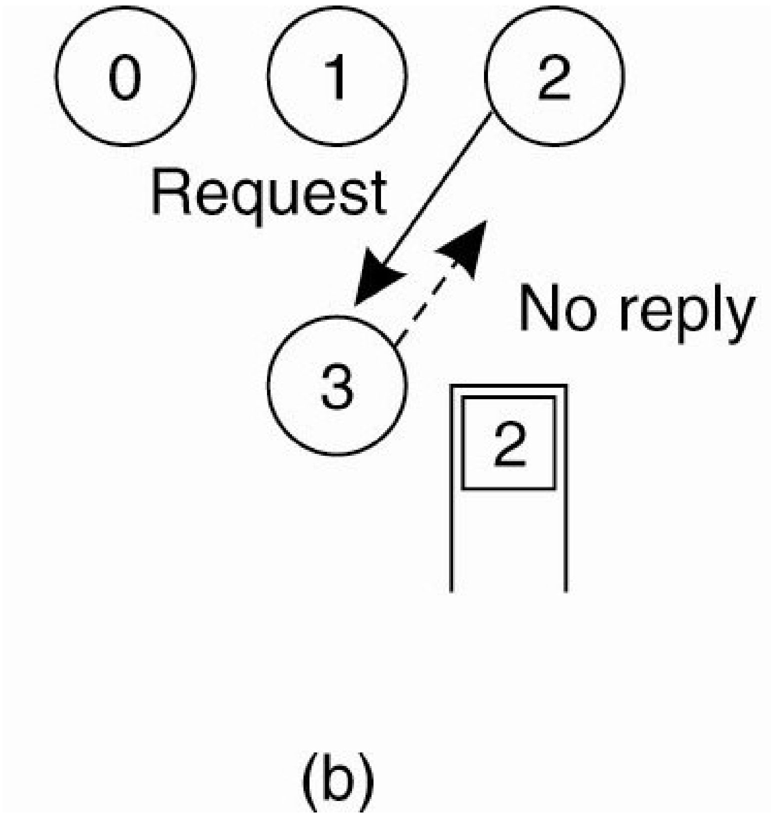


Mutual Exclusion: A Centralized Algorithm (1)



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

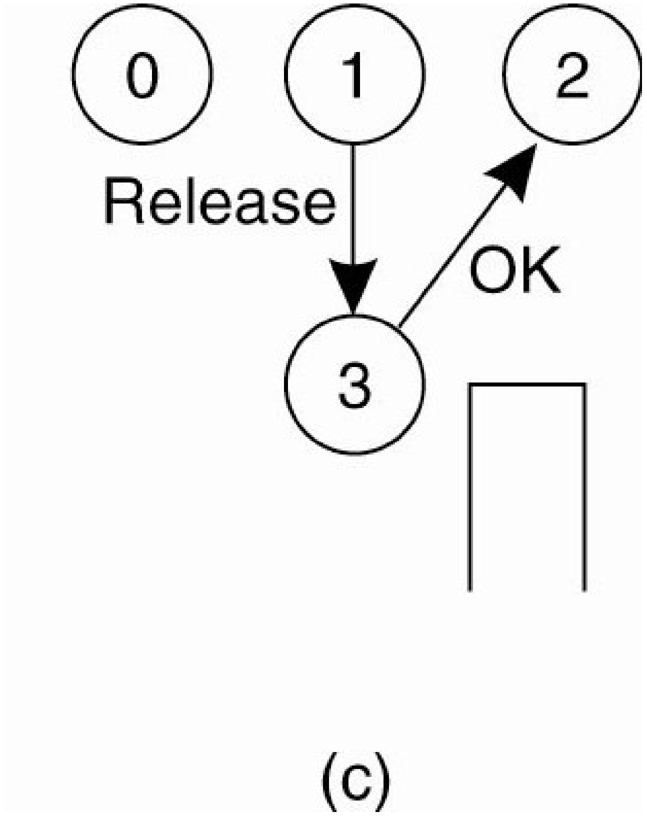
Mutual Exclusion: A Centralized Algorithm (2)



Process 2 then asks permission to access the same resource.

The coordinator does not reply.

Mutual Exclusion: A Centralized Algorithm (3)



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

Mutual Exclusion: A Centralized Algorithm (4)

- Disadvantages
  - Single point of failure
  - Dead coordinator and "Permission Denied" messages cannot be distinguished
  - Performance bottleneck in large systems

A Distributed Algorithm (1)

- The decision to enter a critical region is made by all processes in the group.
- The process which wants to enter the critical region builds a message containing critical region ID, its process number, and the current time.
- The process sends the message to all processes including itself.

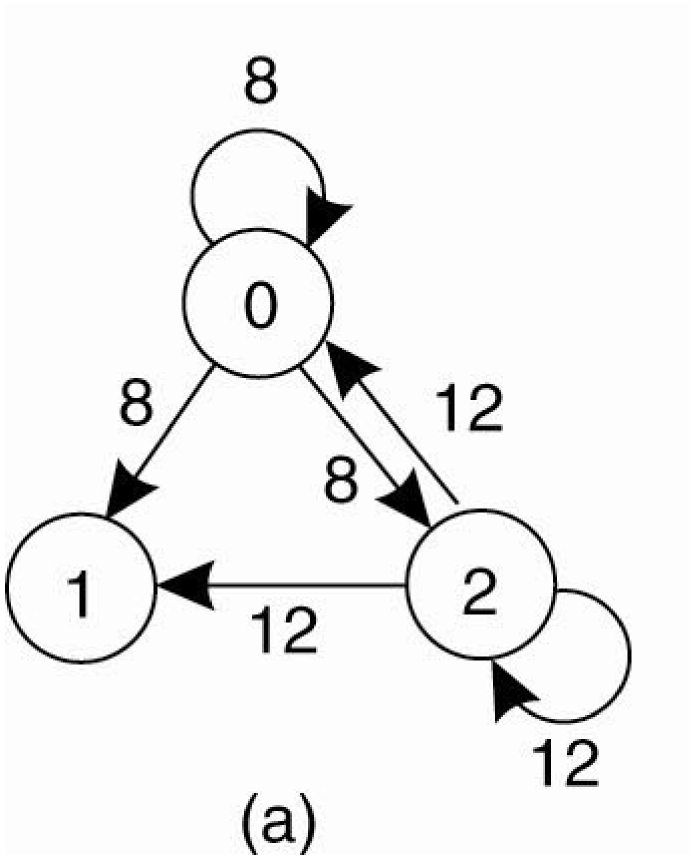
A Distributed Algorithm (2)

- 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 (3)

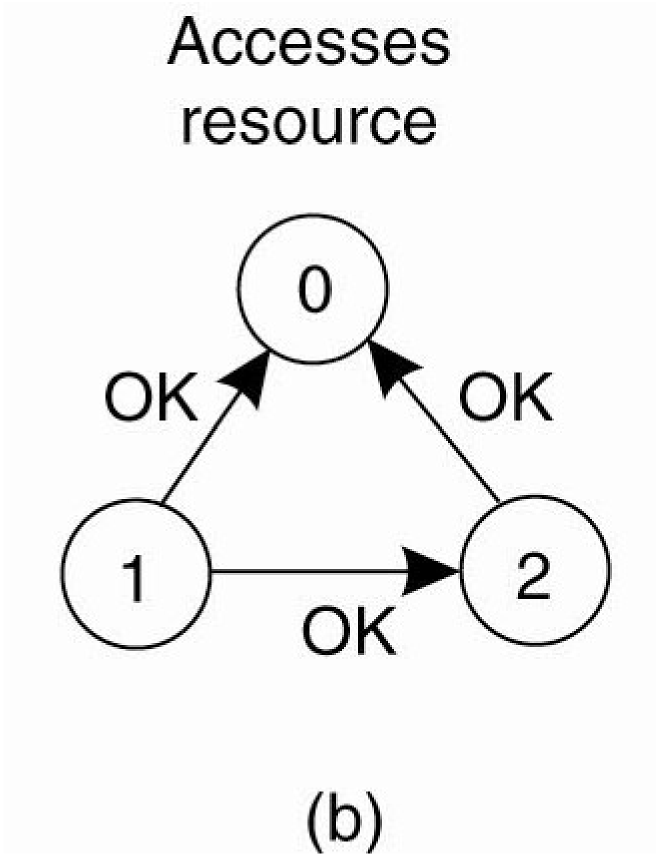
Two processes want to access a shared resource at the same moment.



(a)

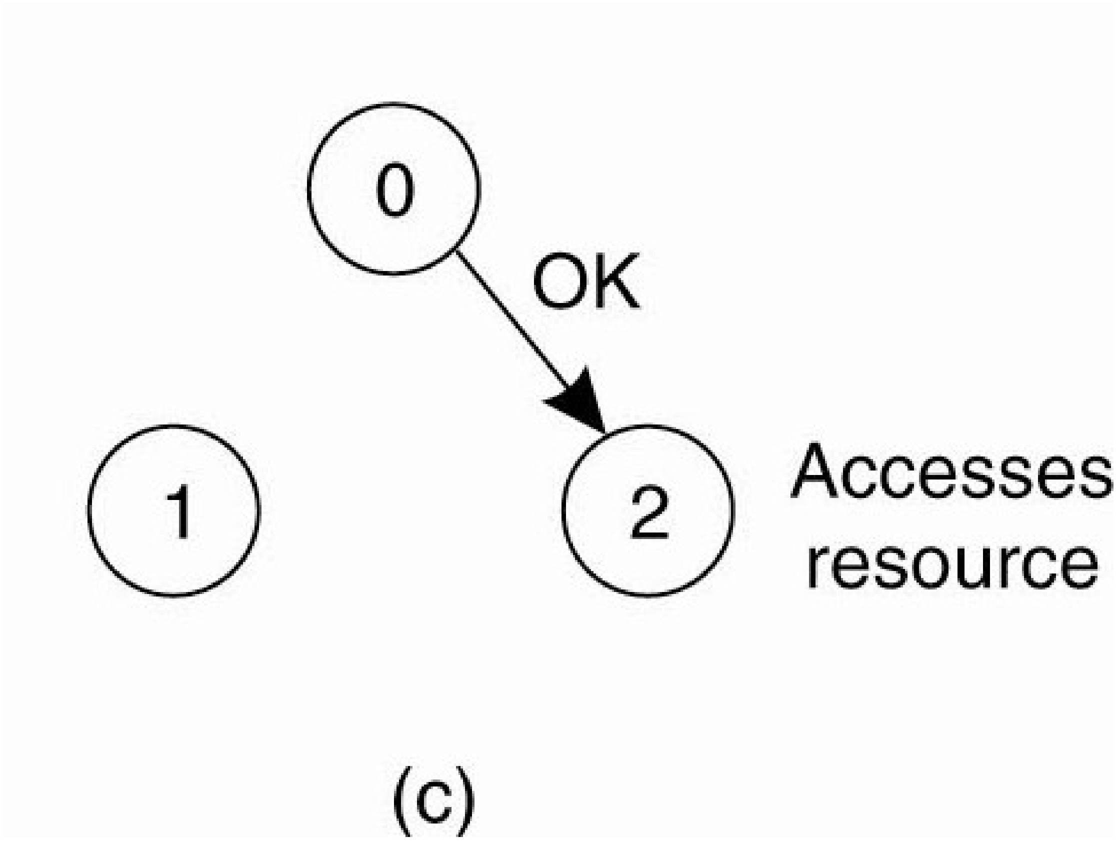
A Distributed Algorithm (4)

Process 0 has the lowest timestamp, so it wins.



A Distributed Algorithm (5)

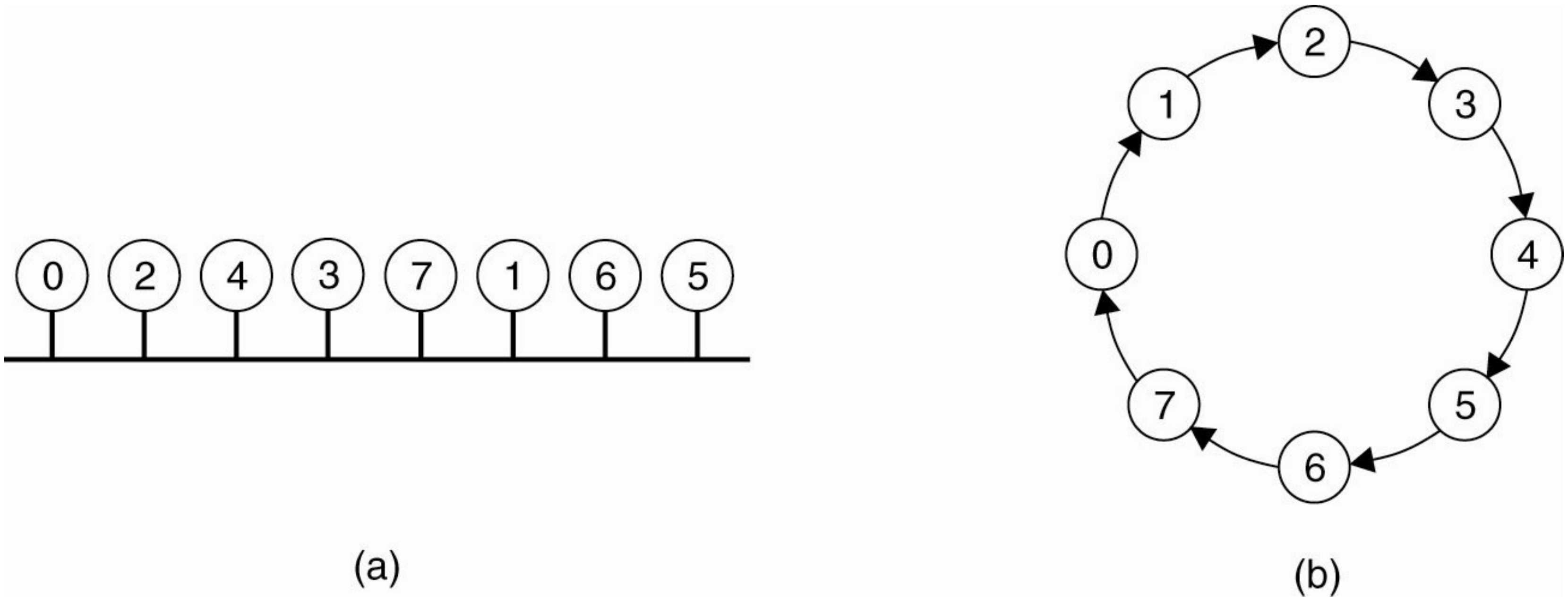
When process 0 is done, it sends an OK also, so 2 can now go ahead.



A Distributed Algorithm

- If no reply is sent back when the critical region is in use, one point of failure will be replaced with n points of failure.
- The number of message sent is n(n-1)
- Each process should maintain a list of group members, including processes entering or leaving, or crashing.
- The bottleneck problem is not solved. It can be replaced with a majority of votes instead of all votes.
- The algorithm is more complicated, slower, and more expensive than the centralized algorithm.

A Token Ring Algorithm



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

Problems with Token Ring Algorithm

- If the token is lost it must be regenerated
- The time between the successive appearance of a token is unbounded.
- The algorithm runs into problem if a process crashes.
- All members of the group should maintain the current ring configuration.

Election Algorithms

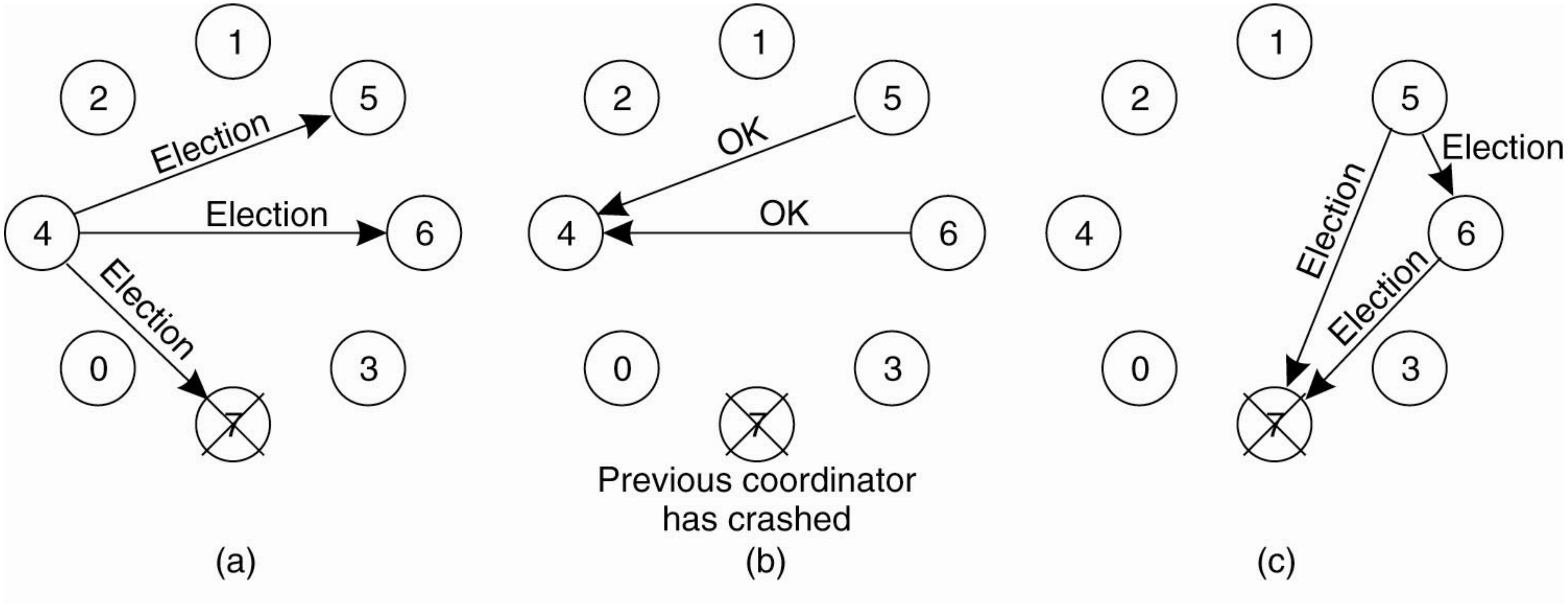
- Many distributed algorithms require one process to act as coordinator, initiator, or otherwise perform some special role.
- In general, it does not matter which process takes on this special responsibility, but one of them has to do it. In this
- If all processes are exactly the same, with no distinguishing characteristics, an election algorithm is used to choose one of them.

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.

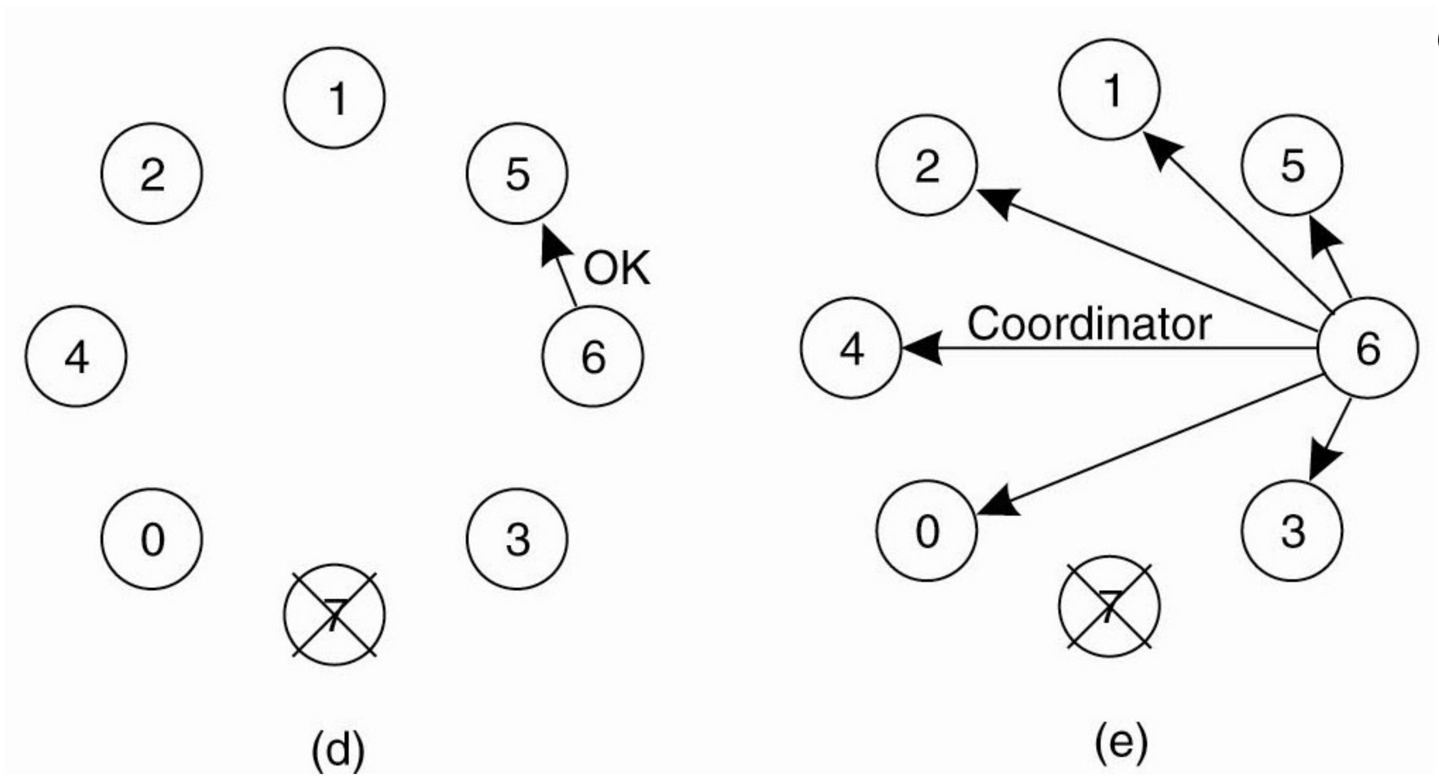
The Bully Algorithm (1)



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)

6 tells



Questions?