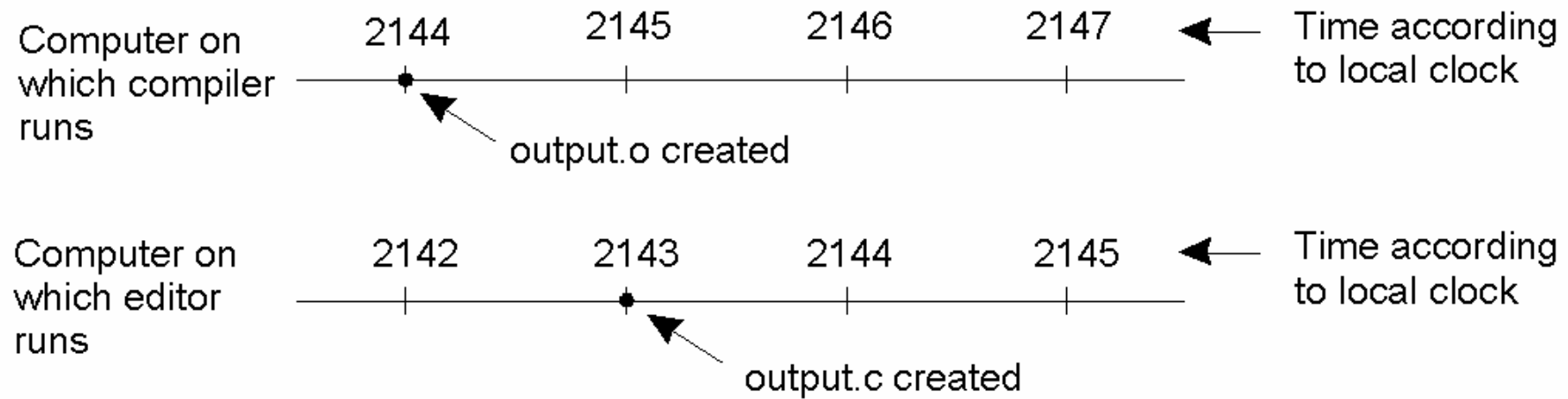


# Synchronization

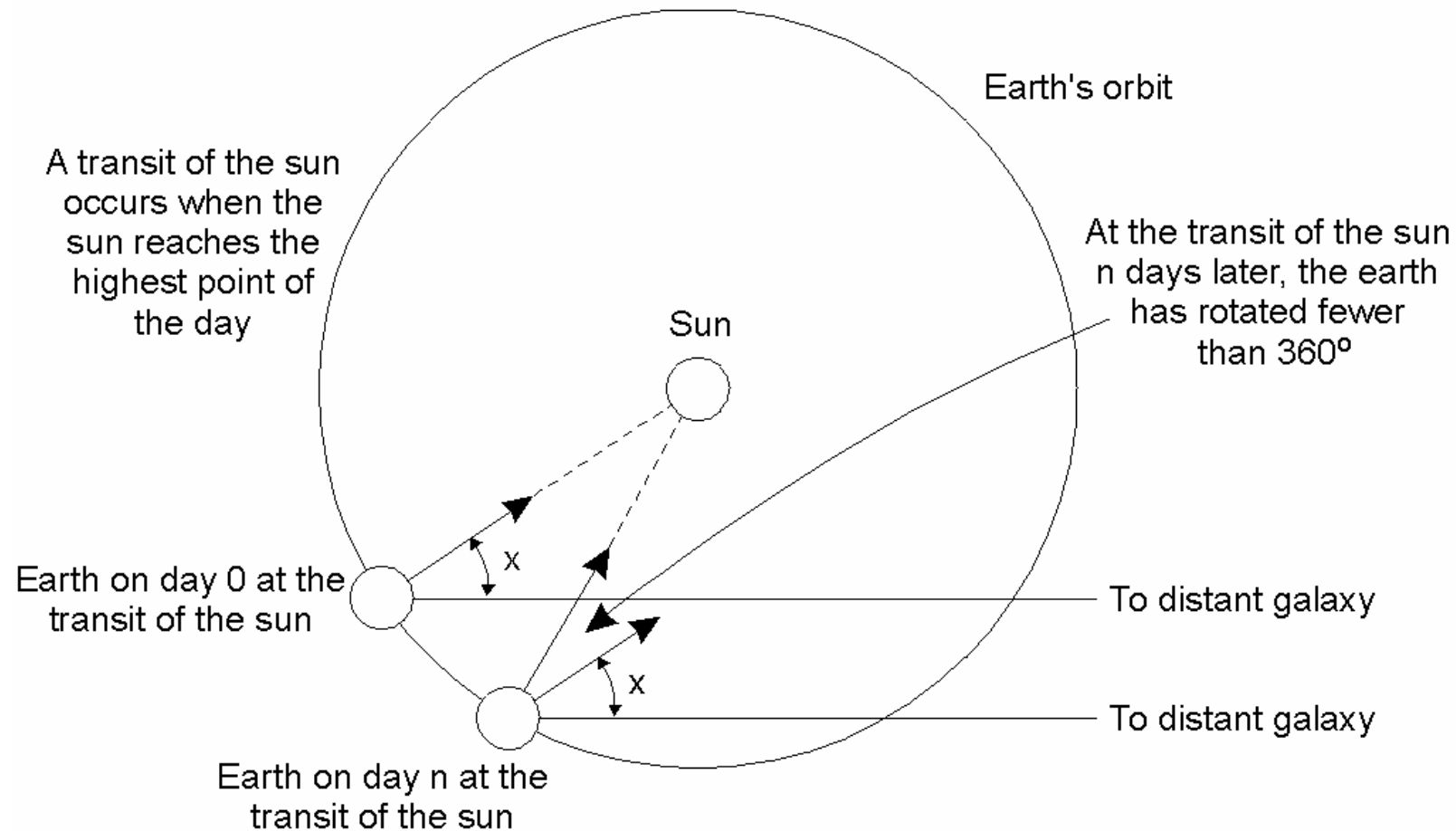
## Chapter 5

# Clock Synchronization



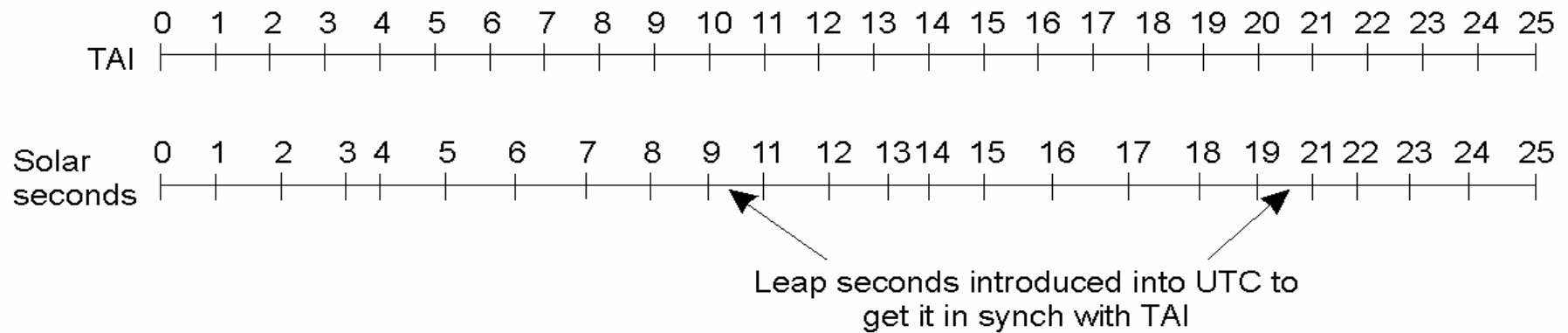
When each machine has its own clock, an event that occurred after another event may nevertheless be assigned an earlier time.

# Physical Clocks (1)



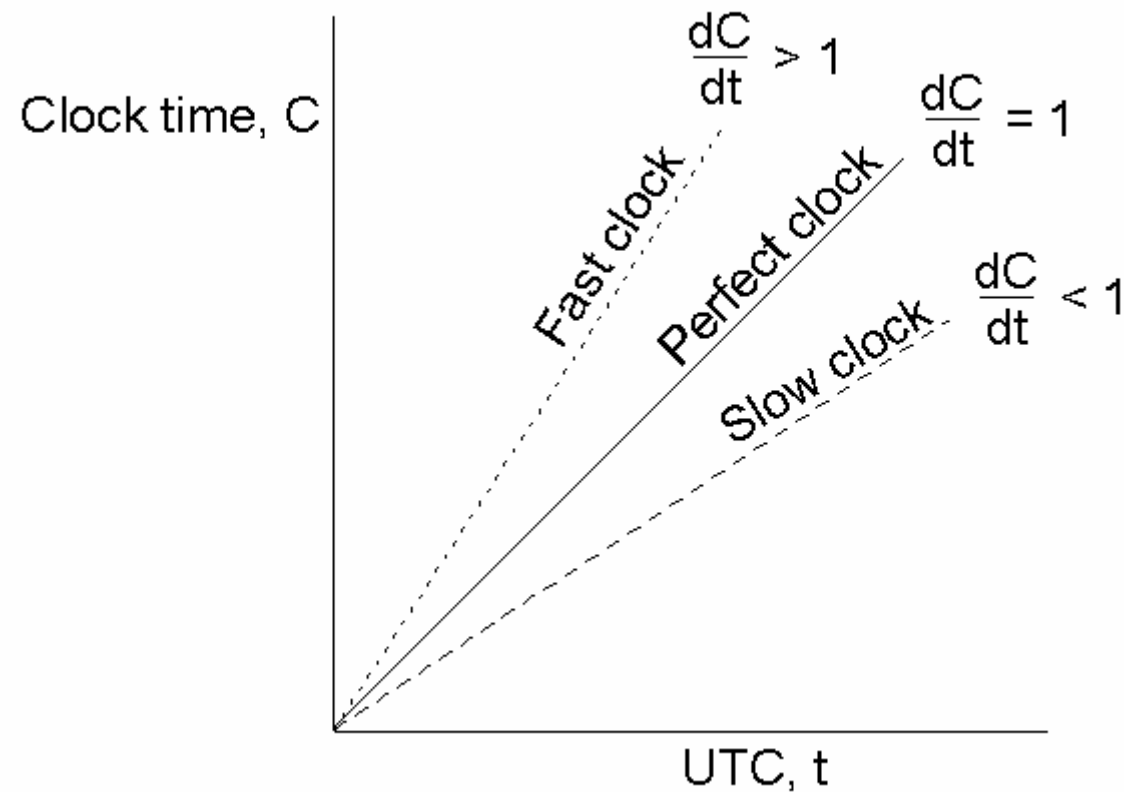
Computation of the mean solar day.

# Physical Clocks (2)



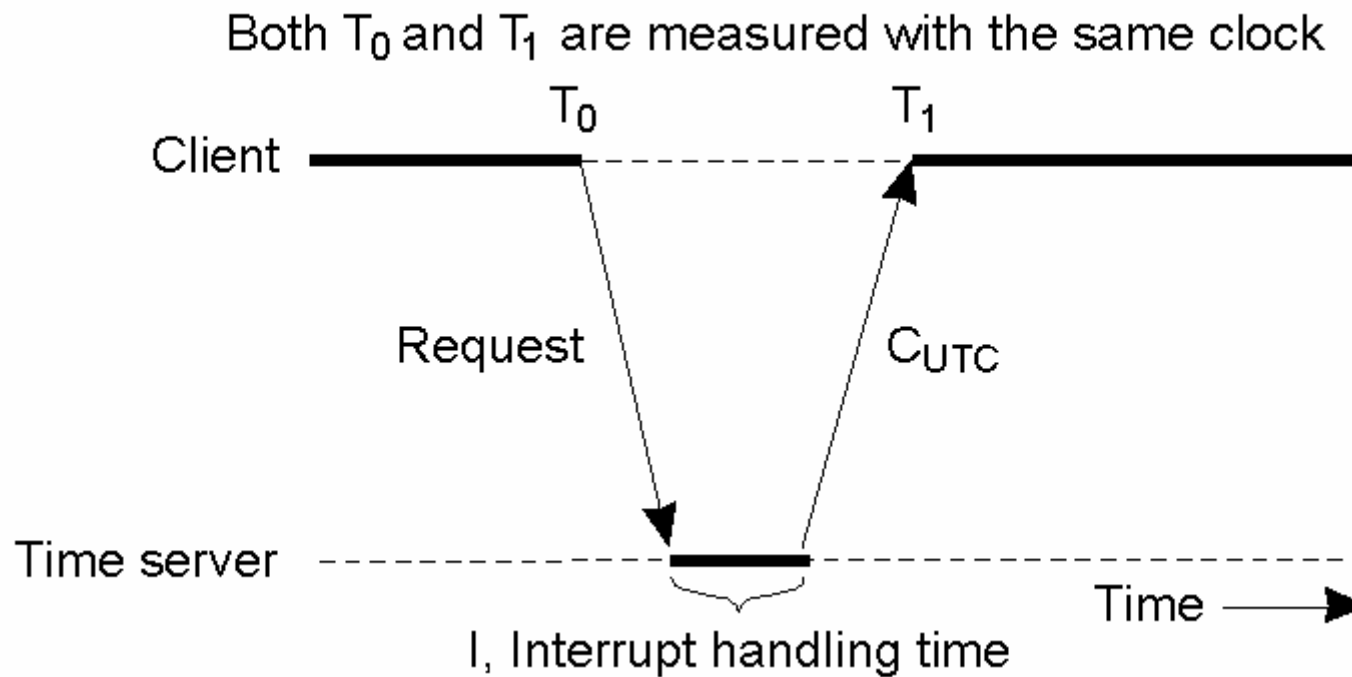
TAI seconds are of constant length, unlike solar seconds. Leap seconds are introduced when necessary to keep in phase with the sun.

# Clock Synchronization Algorithms



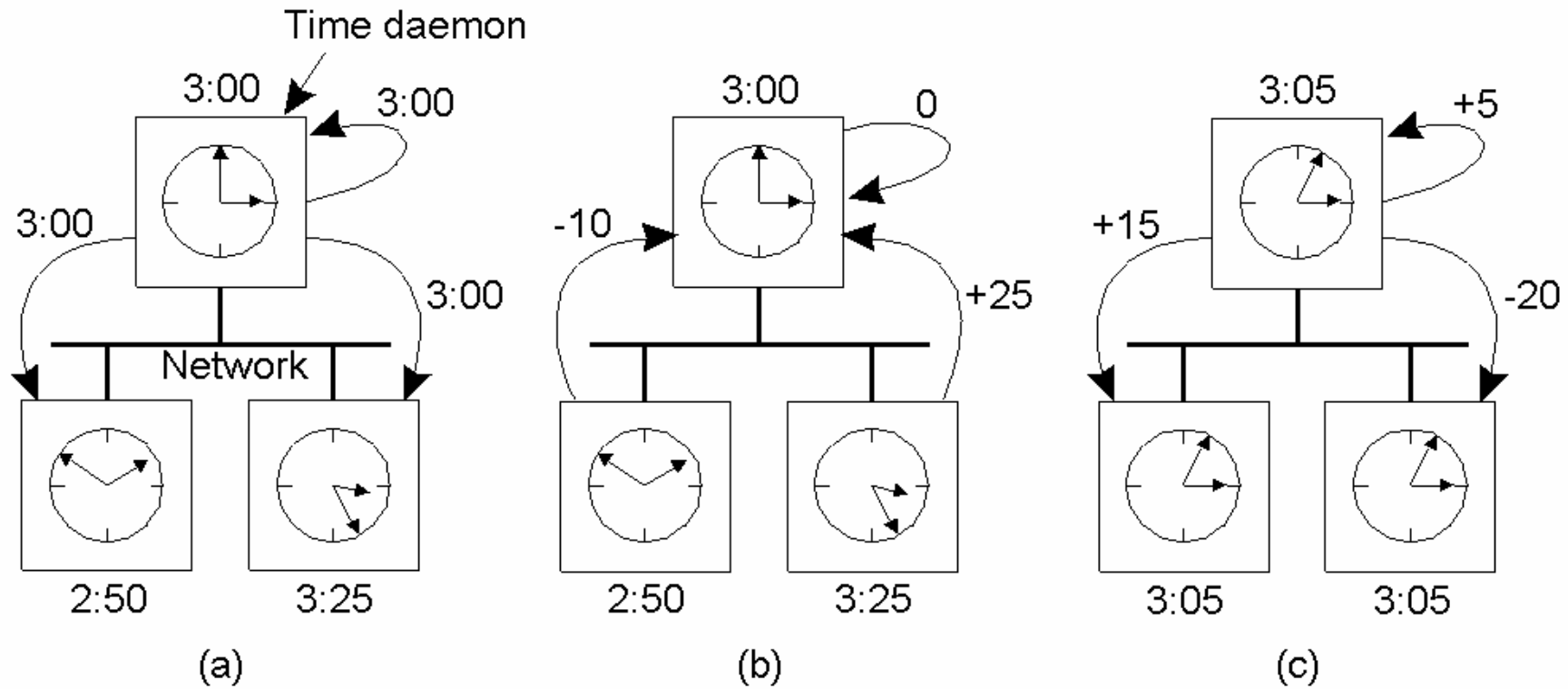
The relation between clock time and UTC when clocks tick at different rates.

# Cristian's Algorithm



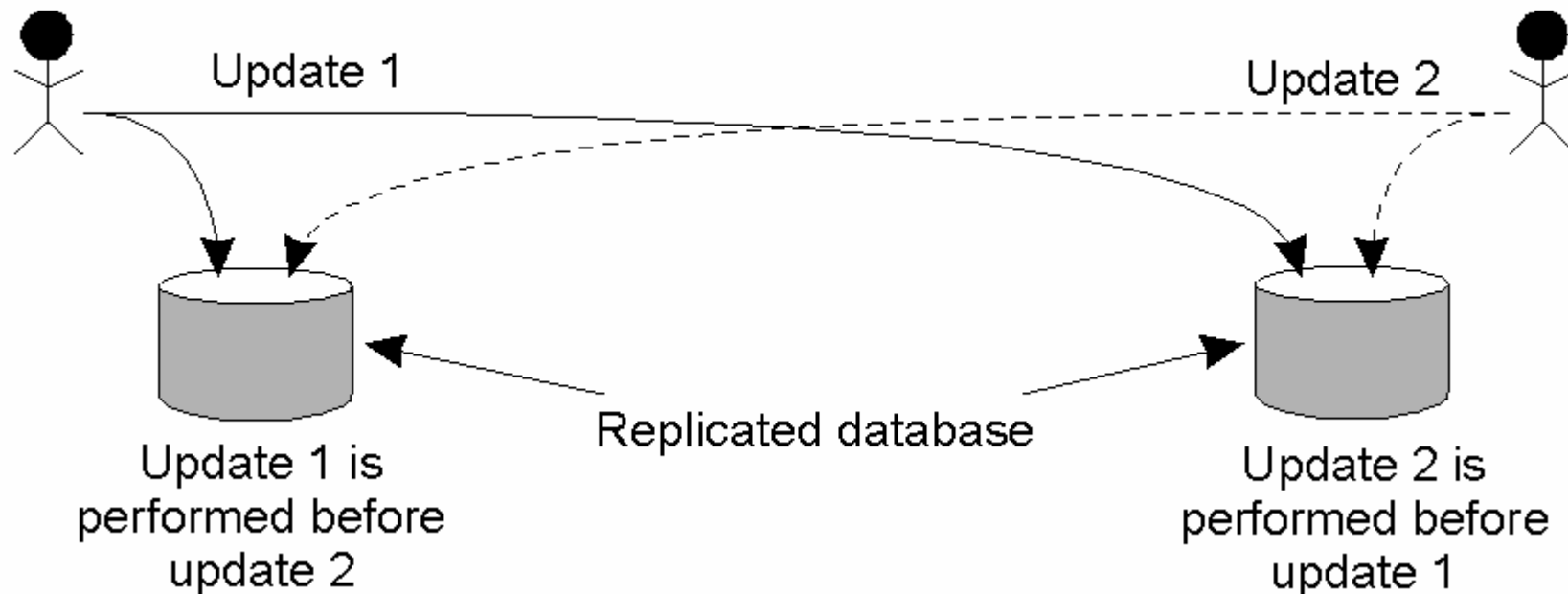
Getting the current time from a time server.

# The Berkeley Algorithm



- a) The time daemon asks all the other machines for their clock values
- b) The machines answer
- c) The time daemon tells everyone how to adjust their clock

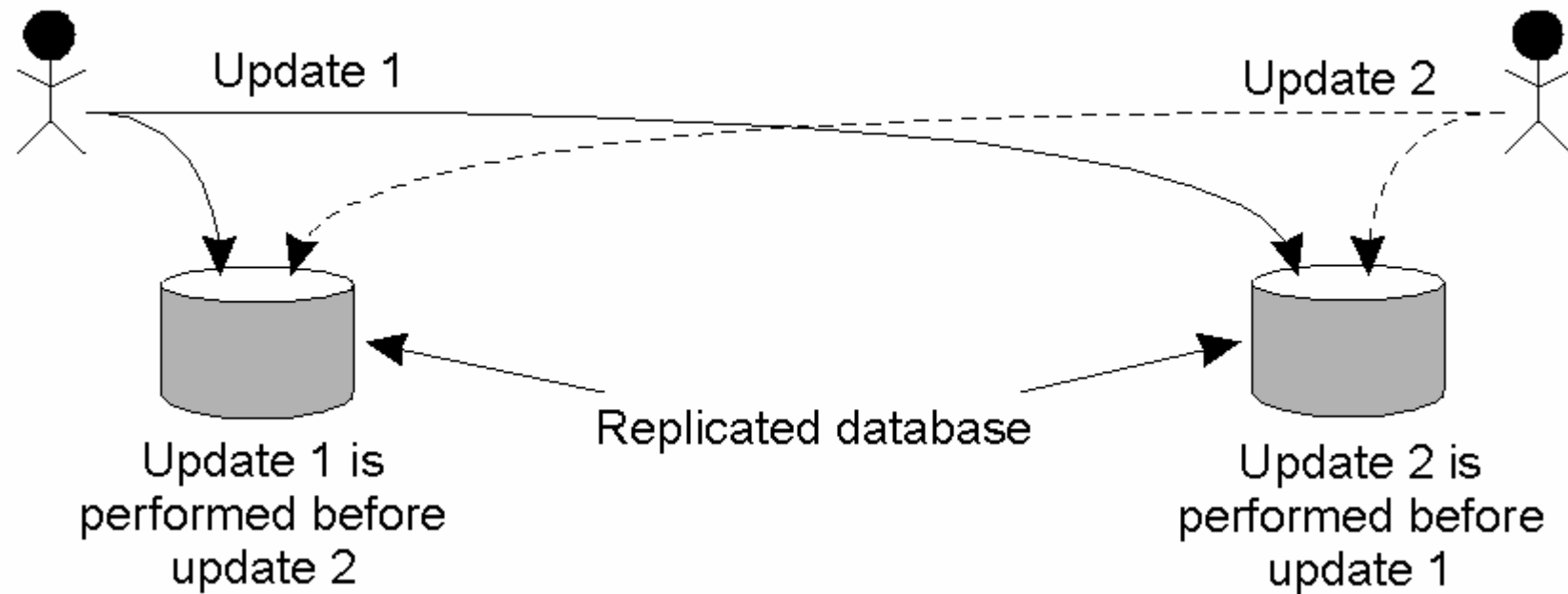
# Lamport Timestamps



- a) Three processes, each with its own clock. The clocks run at different rates.
- b) Lamport's algorithm corrects the clocks.

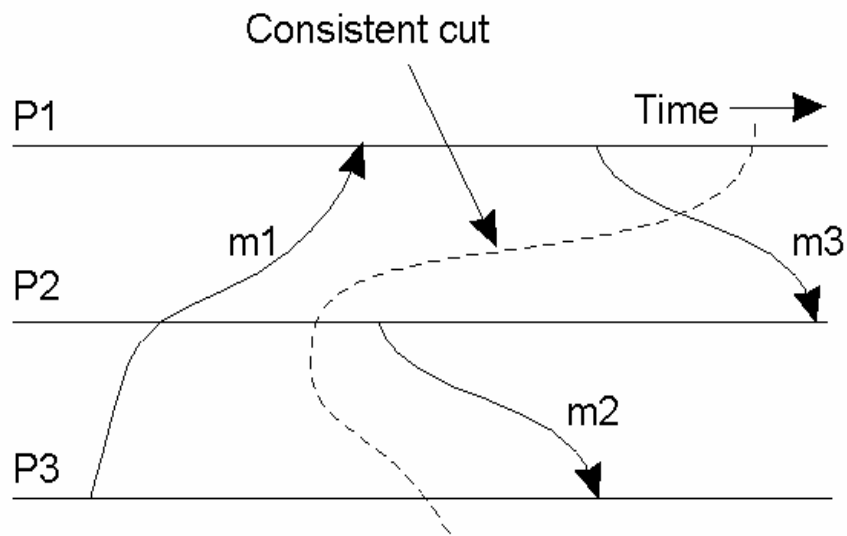


# Example: Totally-Ordered Multicasting

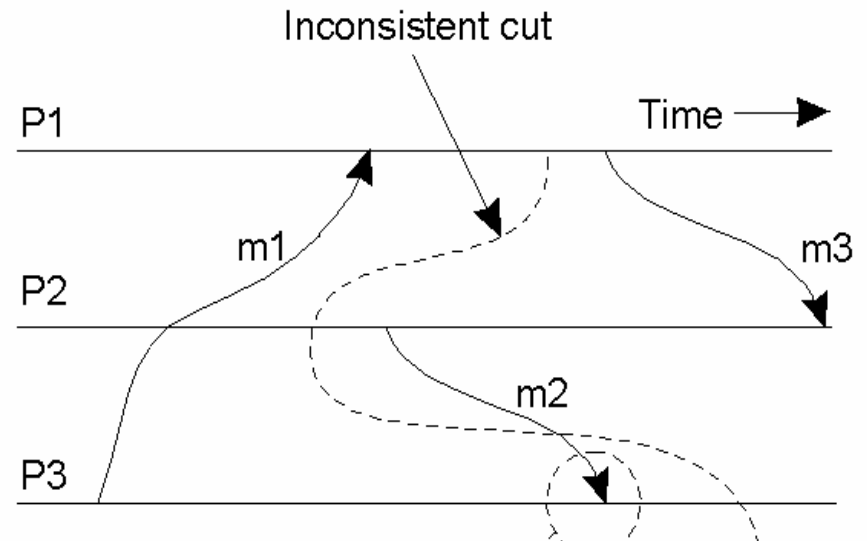


Updating a replicated database and leaving it in an inconsistent state.

# Global State (1)



(a)

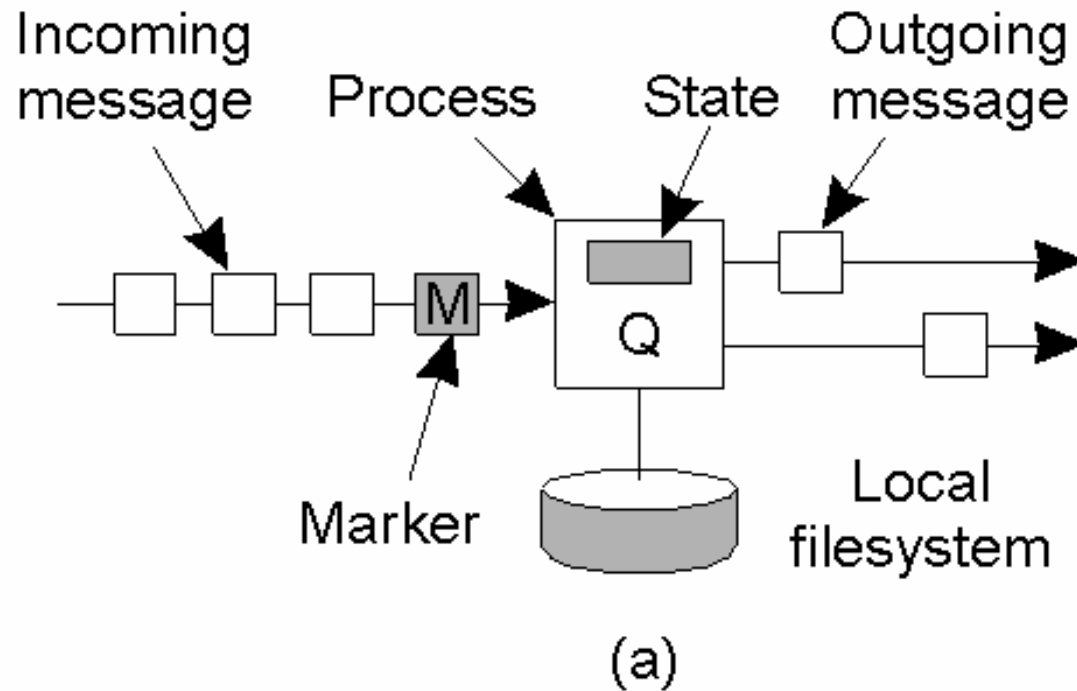


(b)

a) A consistent cut

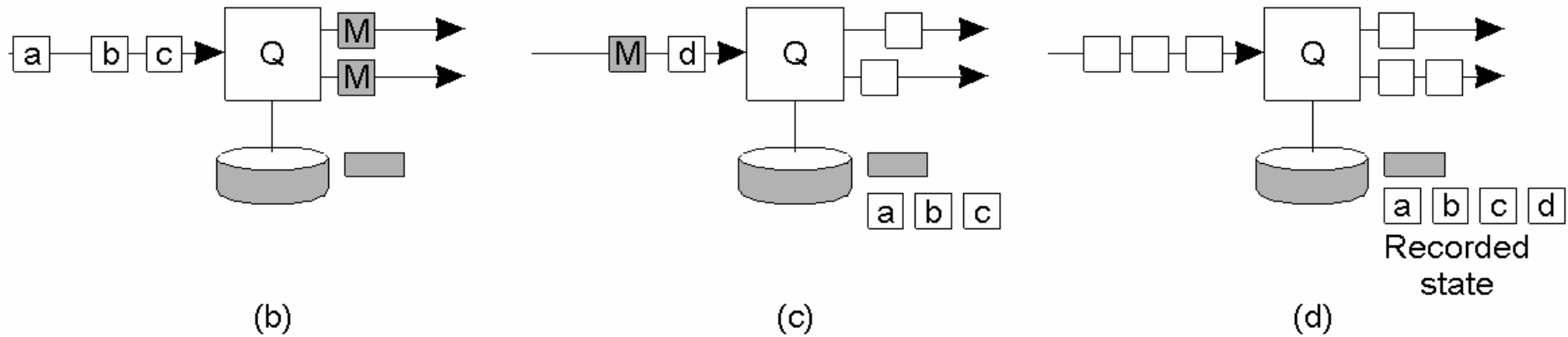
b) An inconsistent cut

## Global State (2)



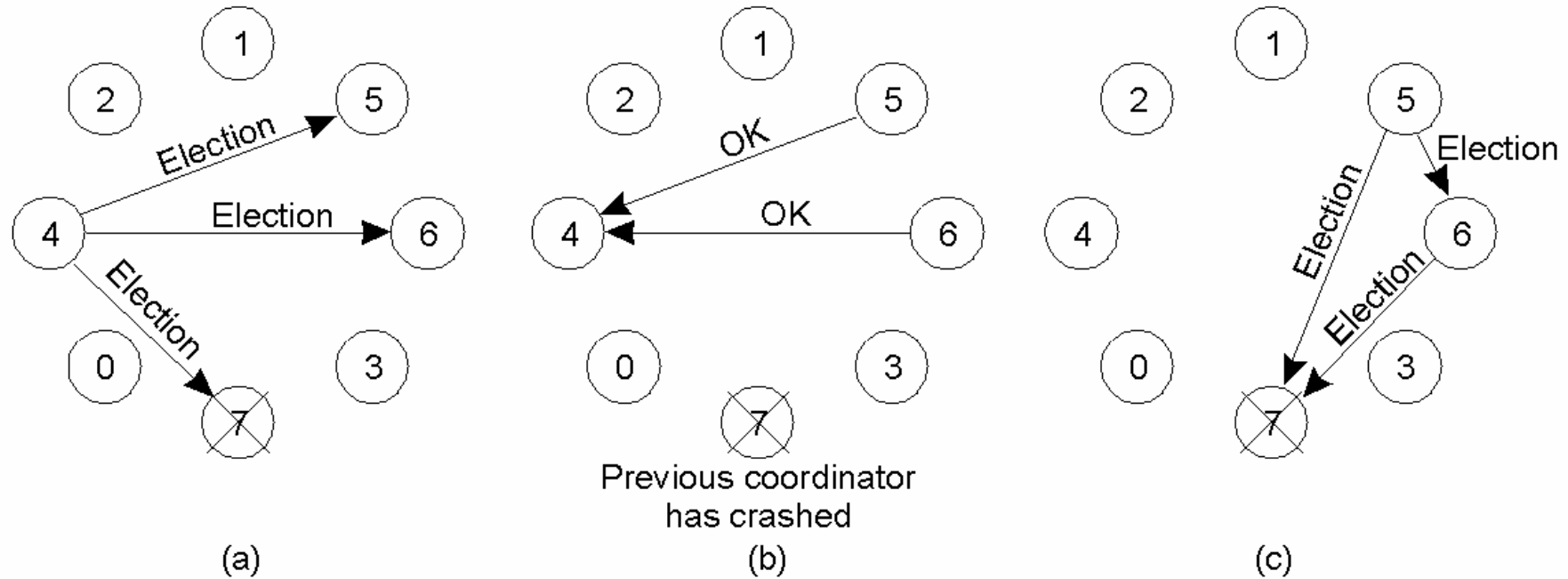
- a) Organization of a process and channels for a distributed snapshot

# Global State (3)



- b)* Process *Q* receives a marker for the first time and records its local state
- c)* *Q* records all incoming message
- d)* *Q* receives a marker for its incoming channel and finishes recording the state of the incoming channel

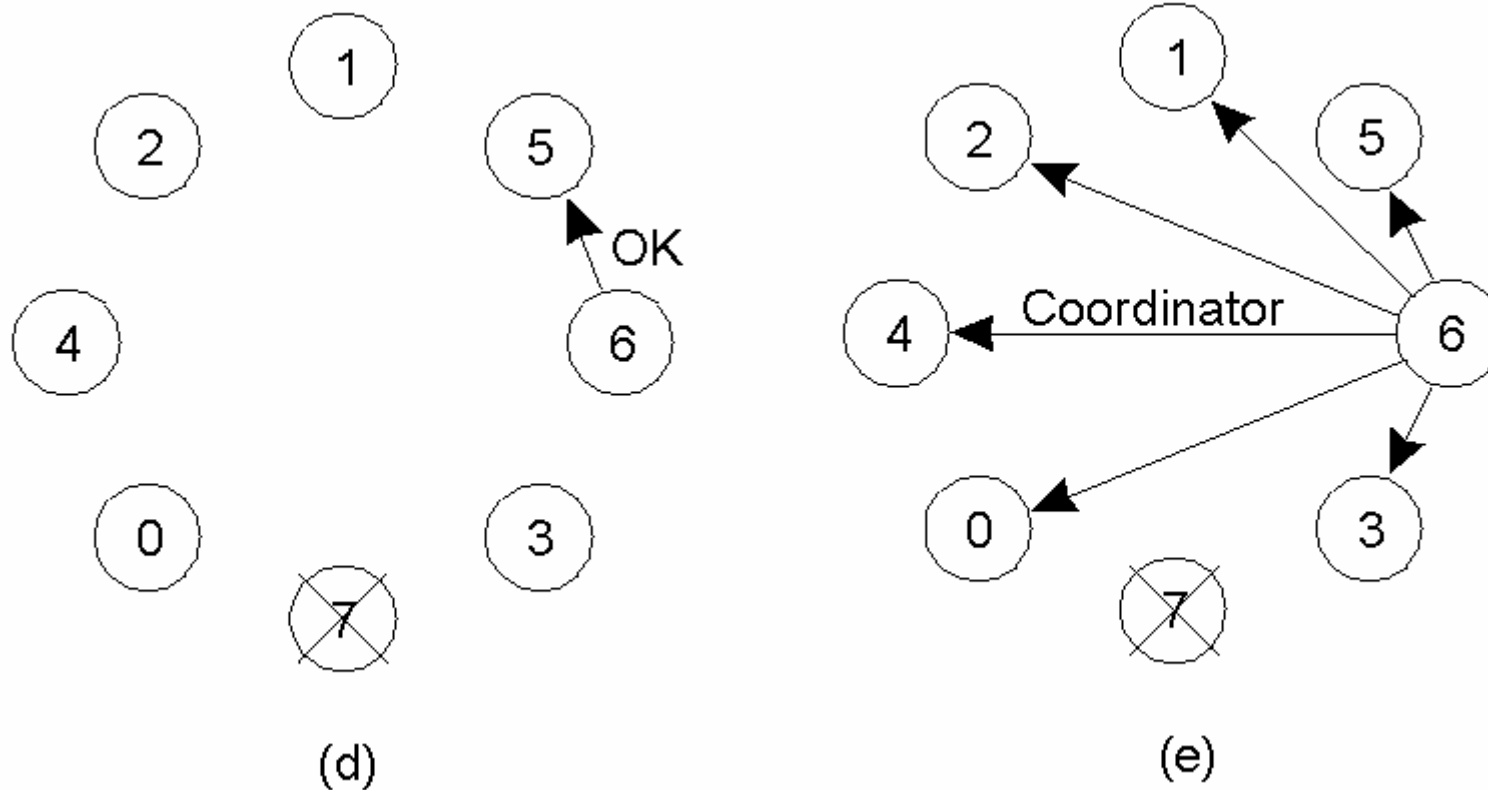
# The Bully Algorithm (1)



The bully election algorithm

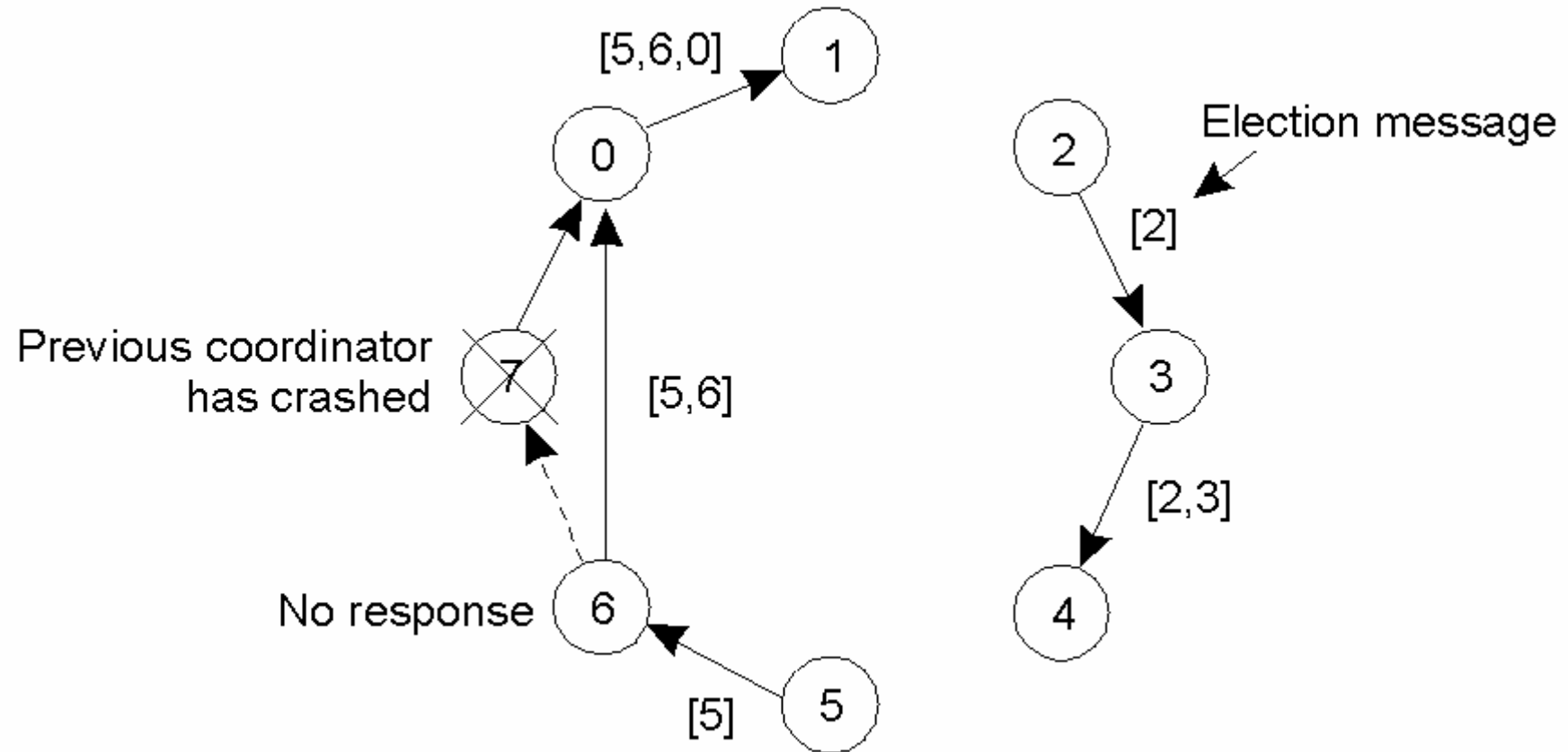
- Process 4 holds an election
- Process 5 and 6 respond, telling 4 to stop
- Now 5 and 6 each hold an election

## Global State (3)



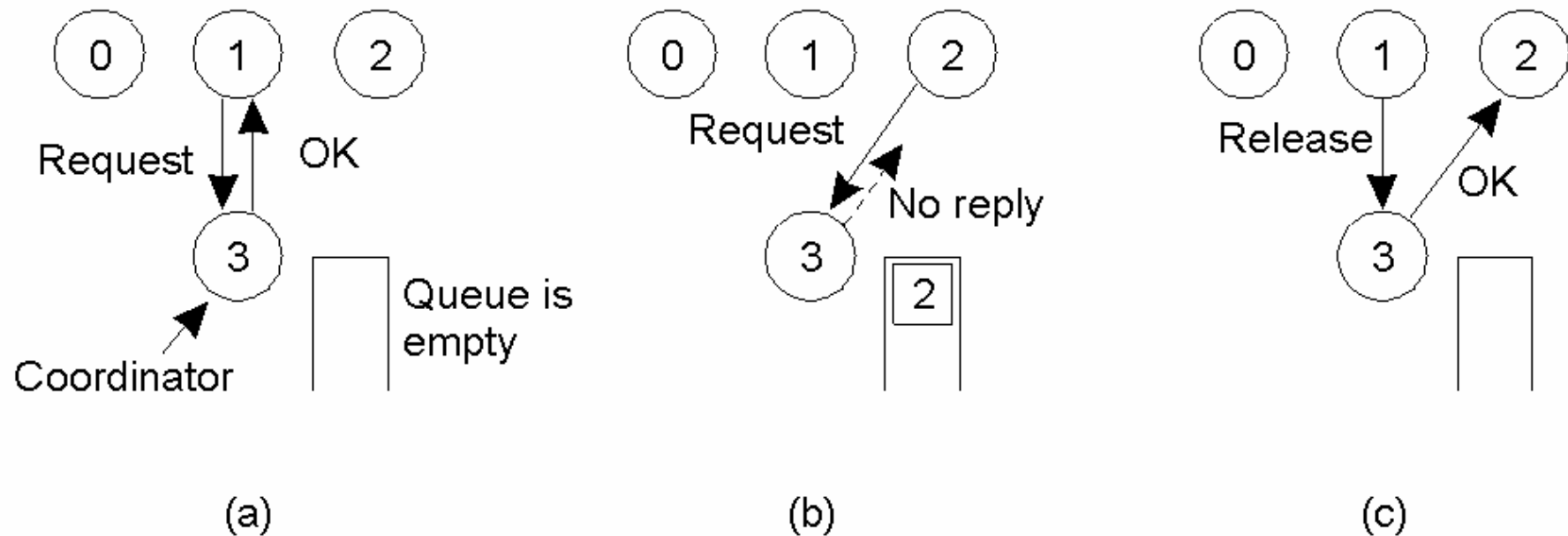
- d) Process 6 tells 5 to stop
- e) Process 6 wins and tells everyone

# A Ring Algorithm



## Election algorithm using a ring.

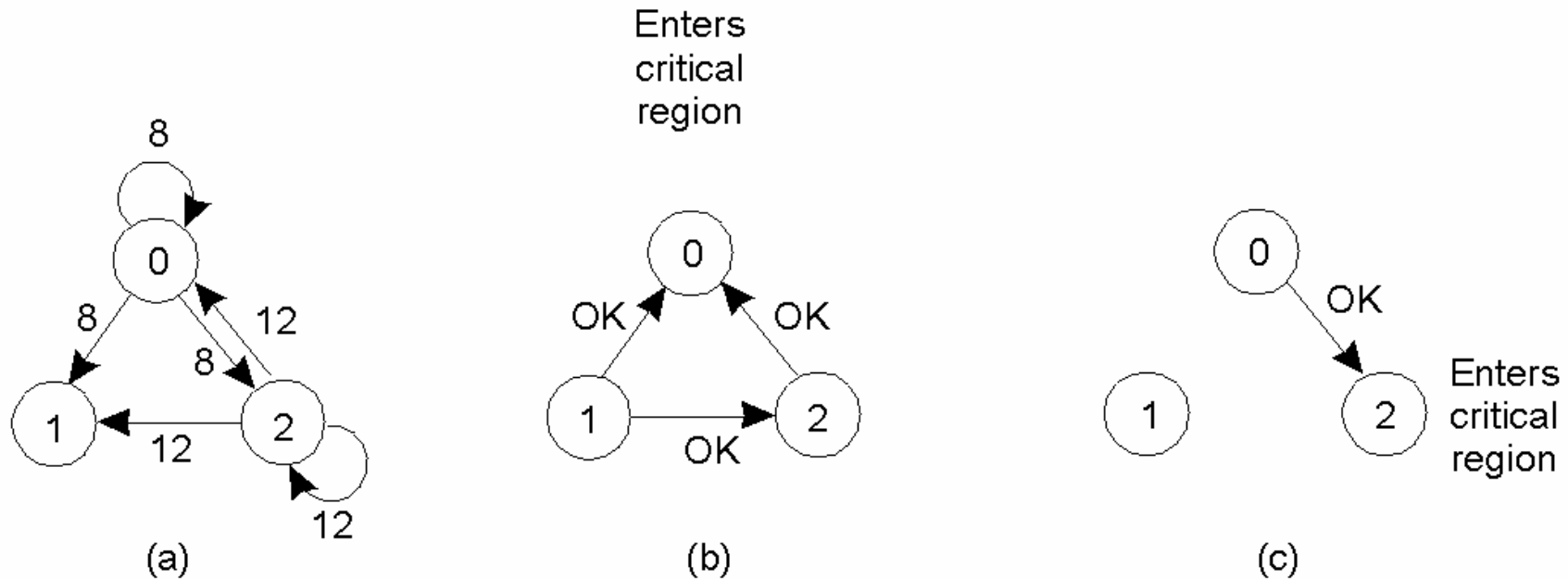
# Mutual Exclusion: A Centralized Algorithm



- a) Process 1 asks the coordinator for permission to enter a critical region. Permission is granted
- b) Process 2 then asks permission to enter the same critical region. The coordinator does not reply.
- c) When process 1 exits the critical region, it tells the coordinator, when then replies to 2

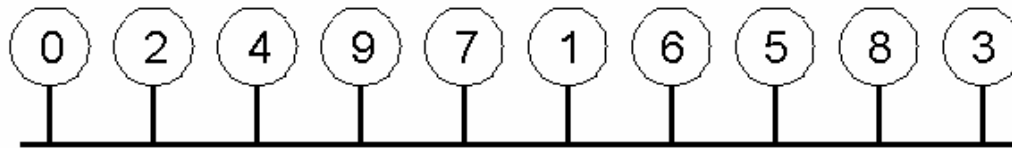


# A Distributed Algorithm

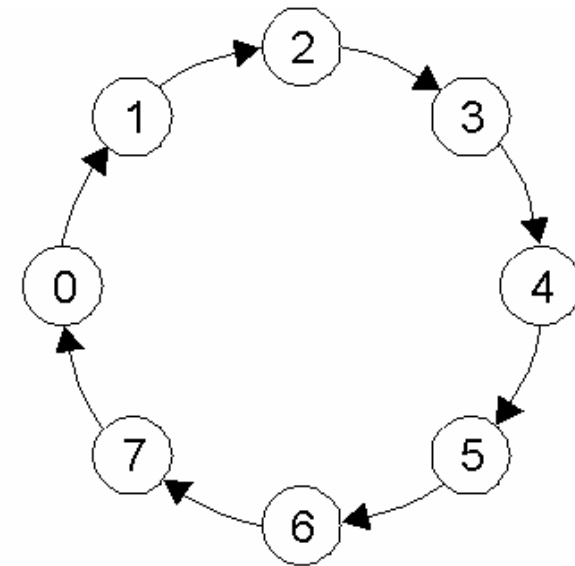


- a) Two processes want to enter the same critical region at the same moment.
- b) Process 0 has the lowest timestamp, so it wins.
- c) When process 0 is done, it sends an OK also, so 2 can now enter the critical region.

# A Token Ring Algorithm



(a)



(b)

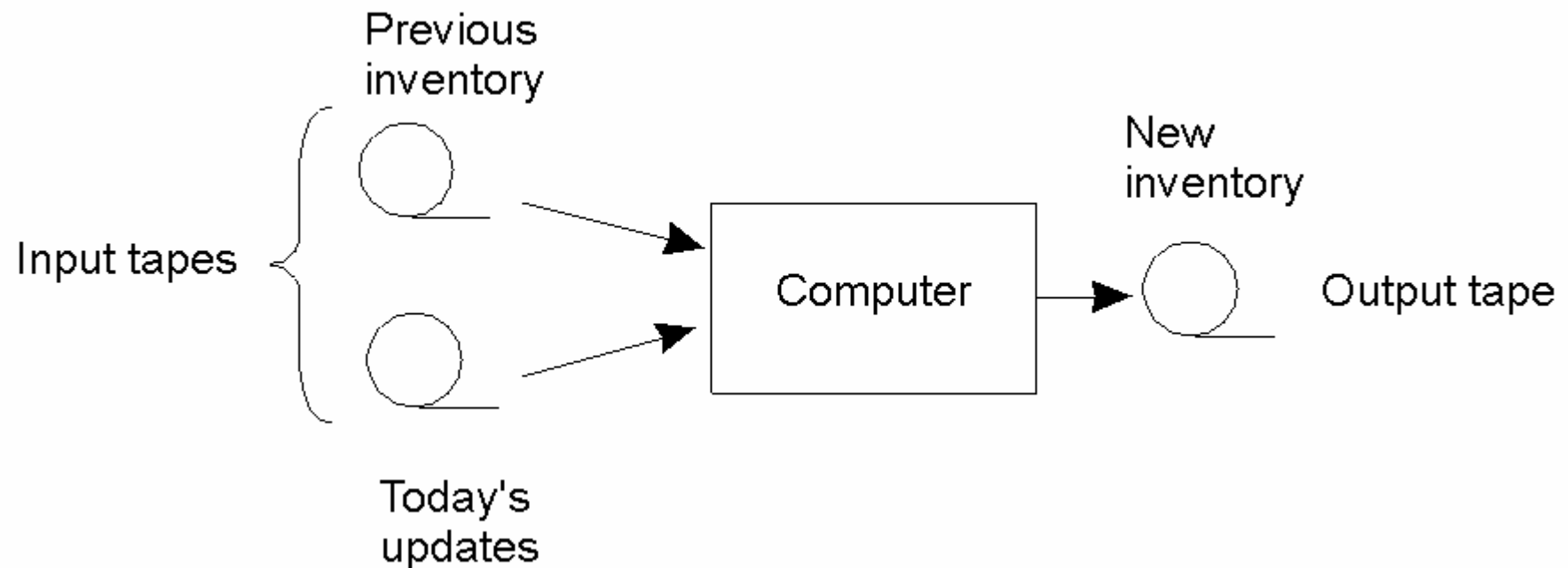
- a) An unordered group of processes on a network.
- b) A logical ring constructed in software.

# Comparison

Algorithm	Messages per entry/exit	Delay before entry (in message times)	Problems
Centralized	3	2	Coordinator crash
Distributed	$2 ( n - 1 )$	$2 ( n - 1 )$	Crash of any process
Token ring	1 to $\infty$	0 to $n - 1$	Lost token, process crash

A comparison of three mutual exclusion algorithms.

# The Transaction Model (1)



Updating a master tape is fault tolerant.

# The Transaction Model (2)

Primitive	Description
BEGIN_TRANSACTION	Make the start of a transaction
END_TRANSACTION	Terminate the transaction and try to commit
ABORT_TRANSACTION	Kill the transaction and restore the old values
READ	Read data from a file, a table, or otherwise
WRITE	Write data to a file, a table, or otherwise

Examples of primitives for transactions.

# The Transaction Model (3)

```
BEGIN_TRANSACTION  
  reserve WP -> JFK;  
  reserve JFK -> Nairobi;  
  reserve Nairobi -> Malindi;  
END_TRANSACTION
```

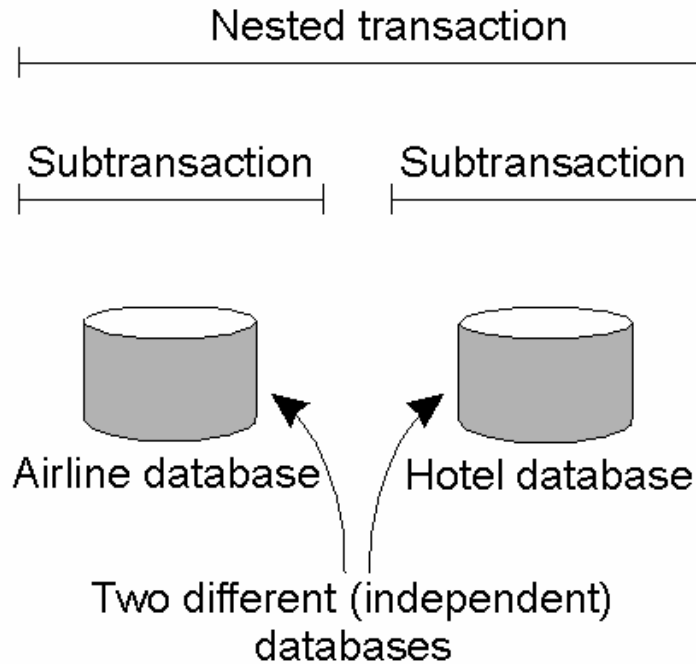
(a)

```
BEGIN_TRANSACTION  
  reserve WP -> JFK;  
  reserve JFK -> Nairobi;  
  reserve Nairobi -> Malindi full =>  
ABORT_TRANSACTION
```

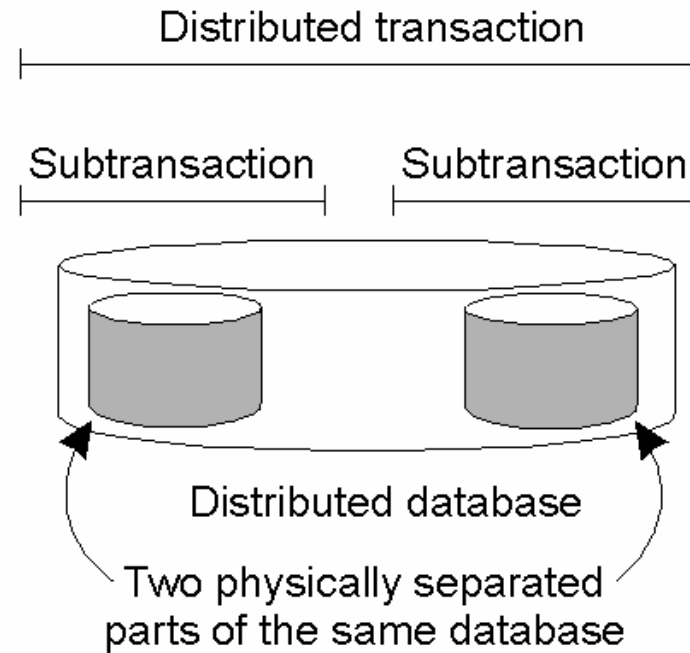
(b)

- a) Transaction to reserve three flights commits
- b) Transaction aborts when third flight is unavailable

# Distributed Transactions



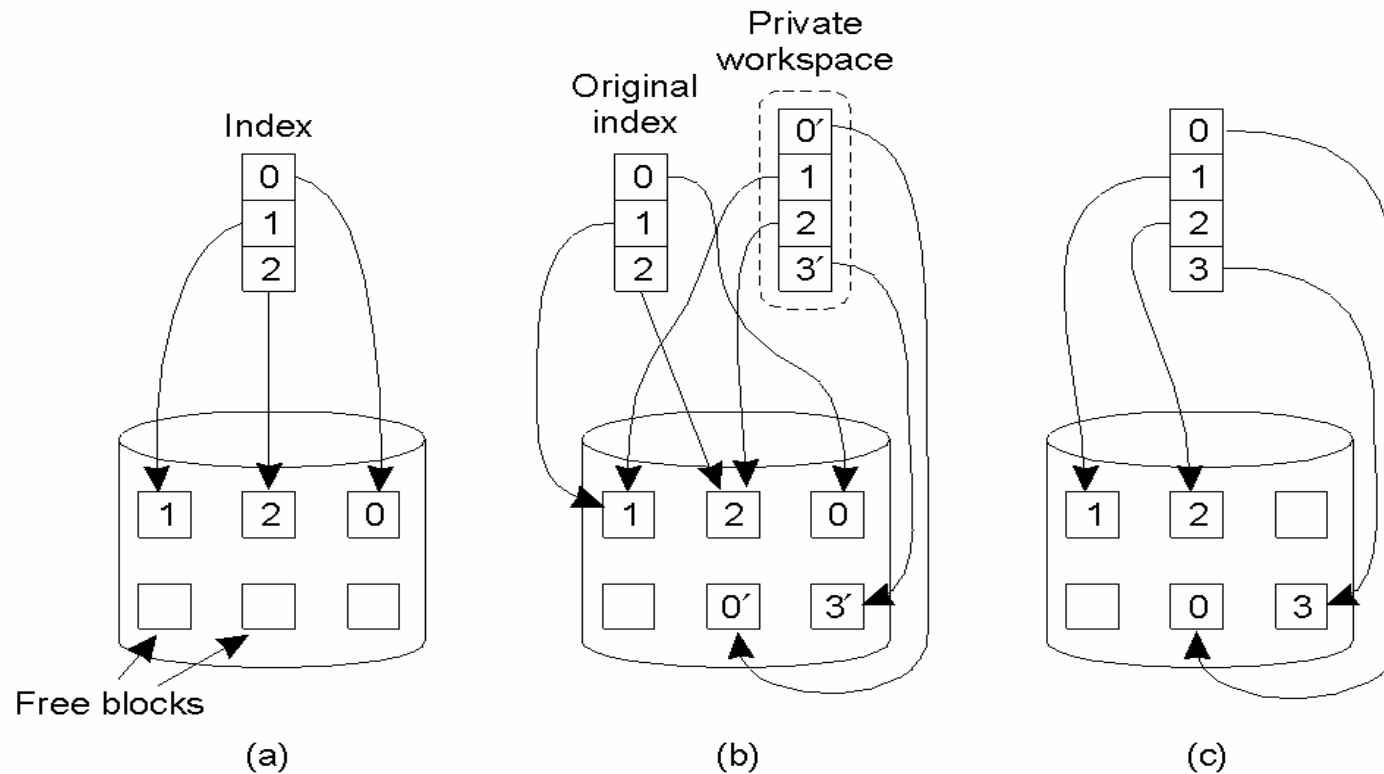
(a)



(b)

- a) A nested transaction
- b) A distributed transaction

# Private Workspace



- a) The file index and disk blocks for a three-block file
- b) The situation after a transaction has modified block 0 and appended block 3
- c) After committing



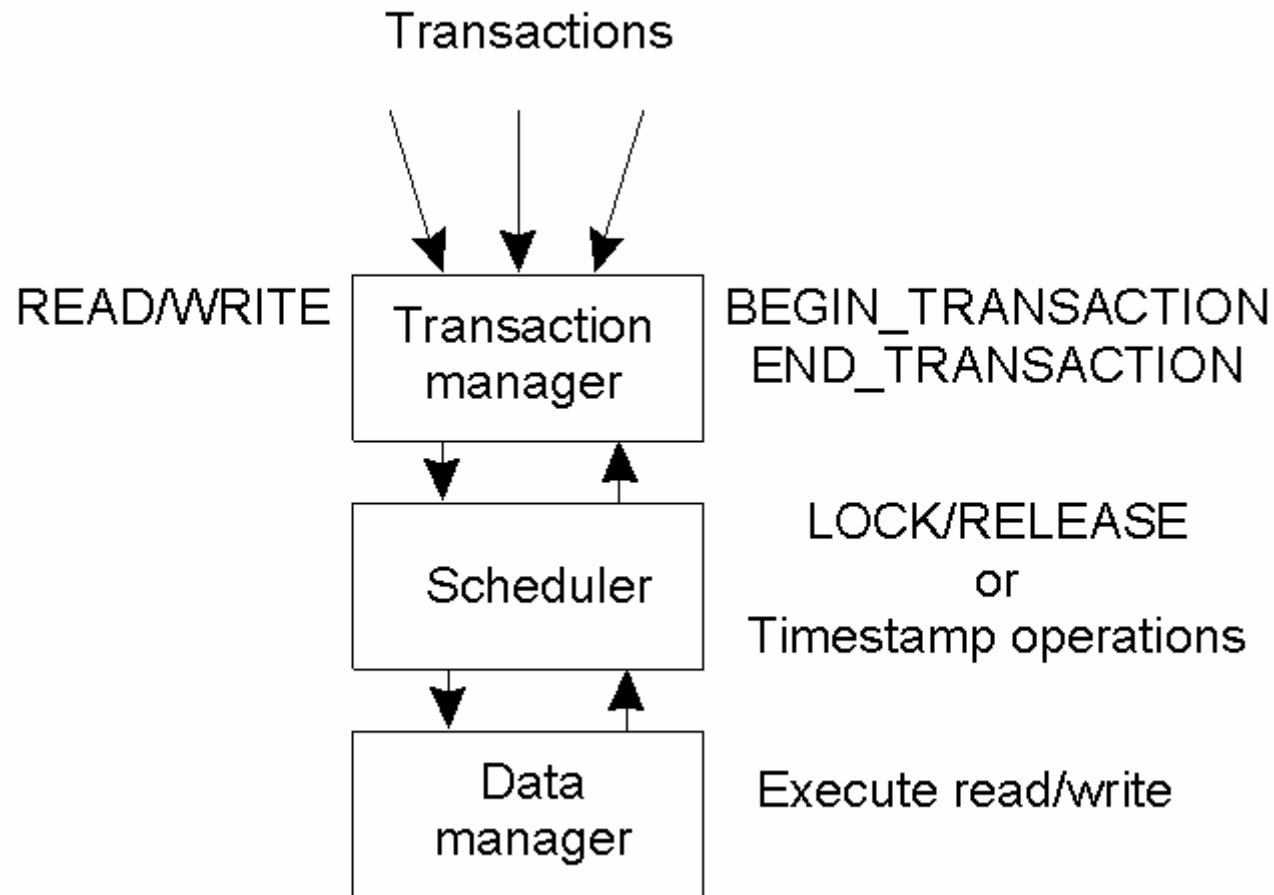
# Writeahead Log

x = 0; y = 0; BEGIN_TRANSACTION; x = x + 1; y = y + 2 x = y * y; END_TRANSACTION;	Log   [x = 0 / 1]	Log   [x = 0 / 1] [y = 0/2]	Log   [x = 0 / 1] [y = 0/2] [x = 1/4]
(a)	(b)	(c)	(d)

a) A transaction

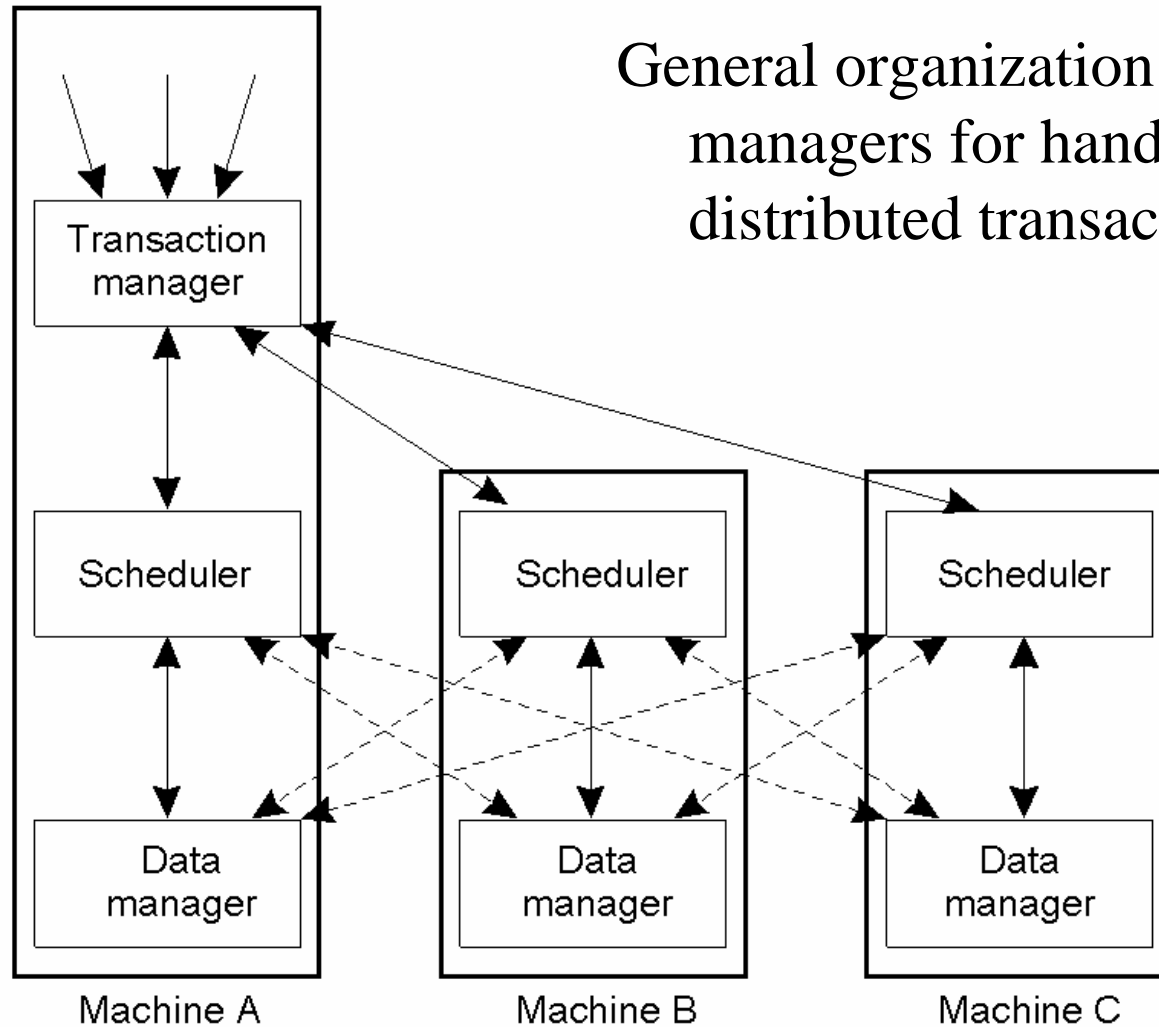
b) – d) The log before each statement is executed

# Concurrency Control (1)



General organization of managers for handling transactions.

# Concurrency Control (2)



# Serializability

BEGIN\_TRANSACTION  
x = 0;  
x = x + 1;  
END\_TRANSACTION

(a)

BEGIN\_TRANSACTION  
x = 0;  
x = x + 2;  
END\_TRANSACTION

(b)

BEGIN\_TRANSACTION  
x = 0;  
x = x + 3;  
END\_TRANSACTION

(c)

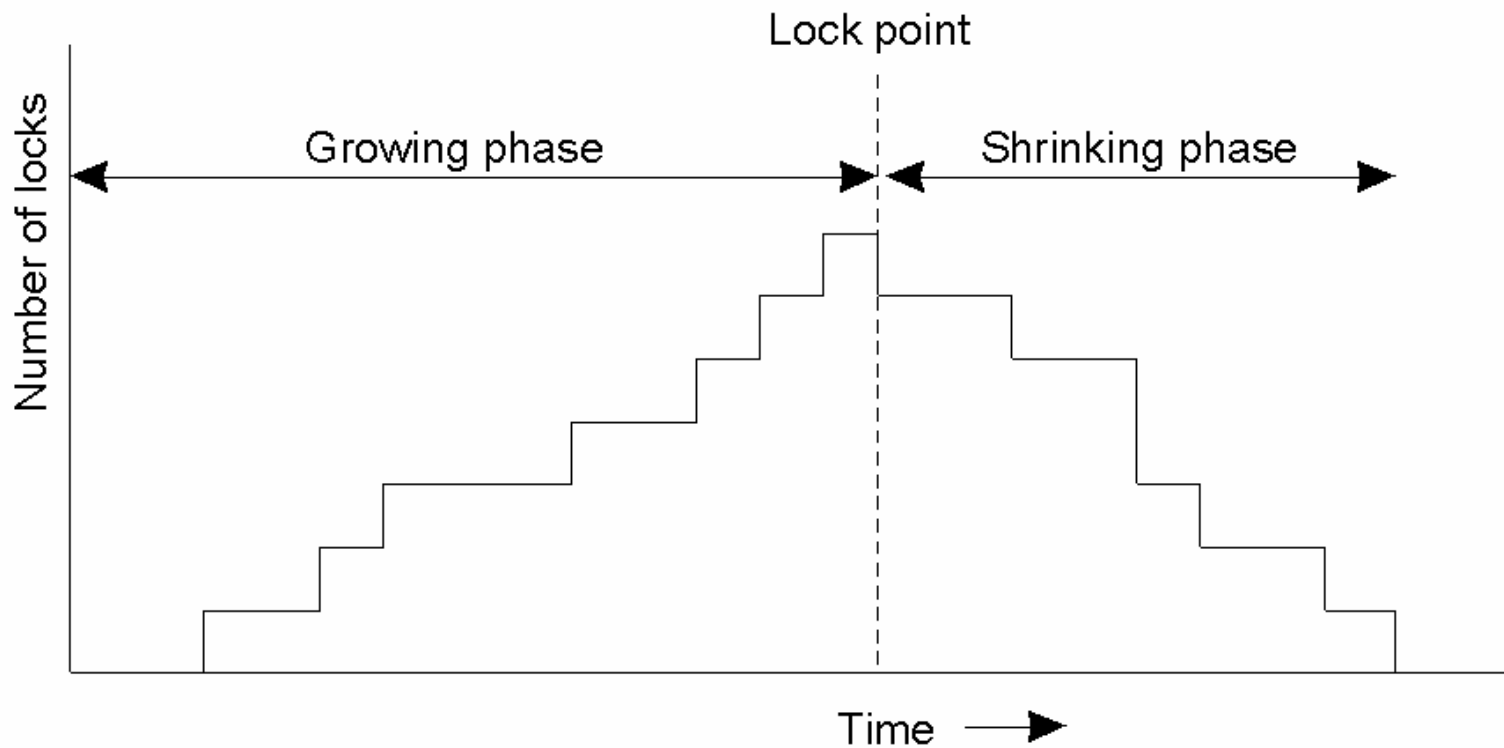
Schedule 1	x = 0; x = x + 1; x = 0; x = x + 2; x = 0; x = x + 3	Legal
Schedule 2	x = 0; x = 0; x = x + 1; x = x + 2; x = 0; x = x + 3;	Legal
Schedule 3	x = 0; x = 0; x = x + 1; x = 0; x = x + 2; x = x + 3;	Illegal

(d)

a) – c) Three transactions  $T_1$ ,  $T_2$ , and  $T_3$

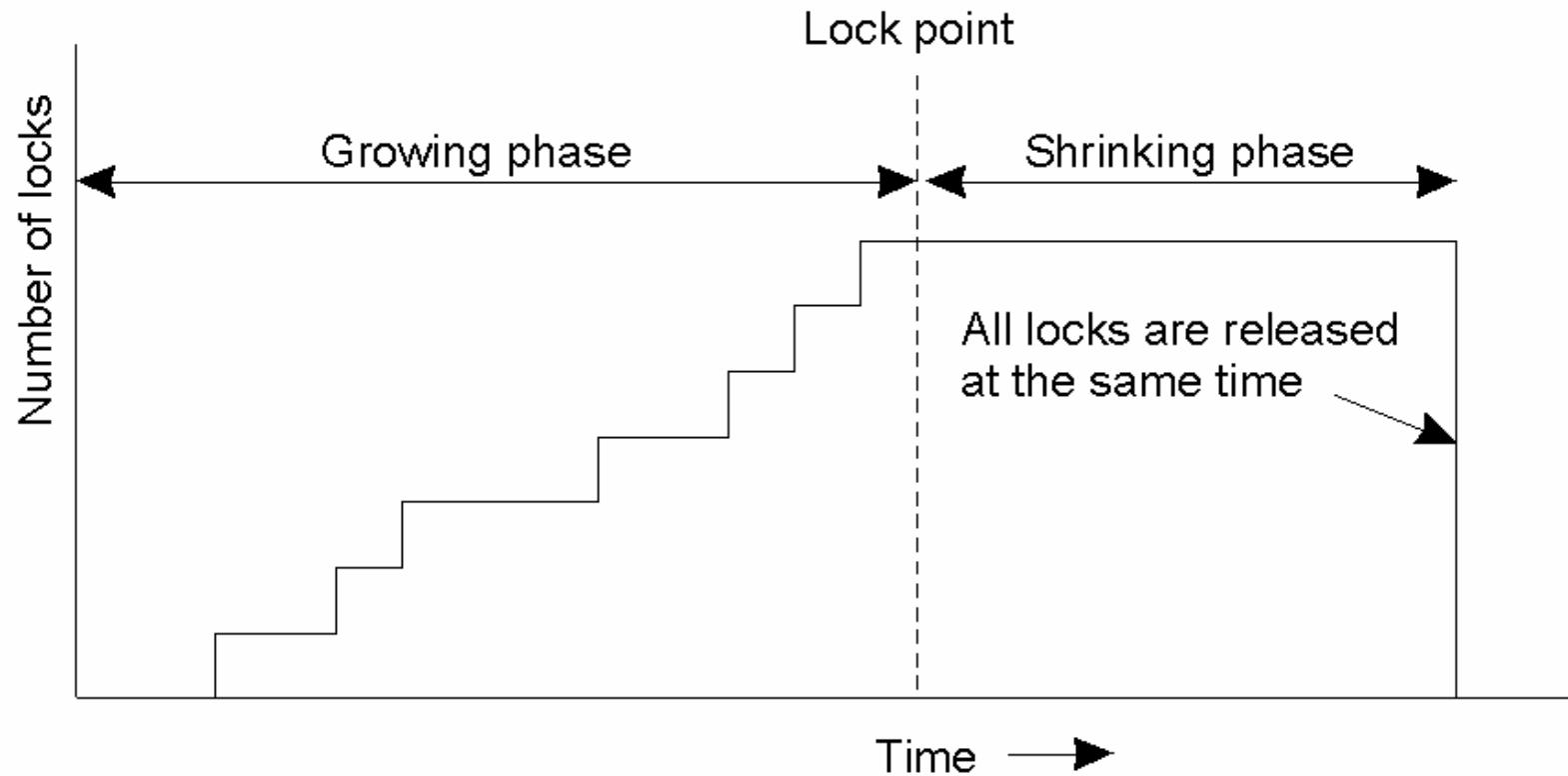
d) Possible schedules

# Two-Phase Locking (1)



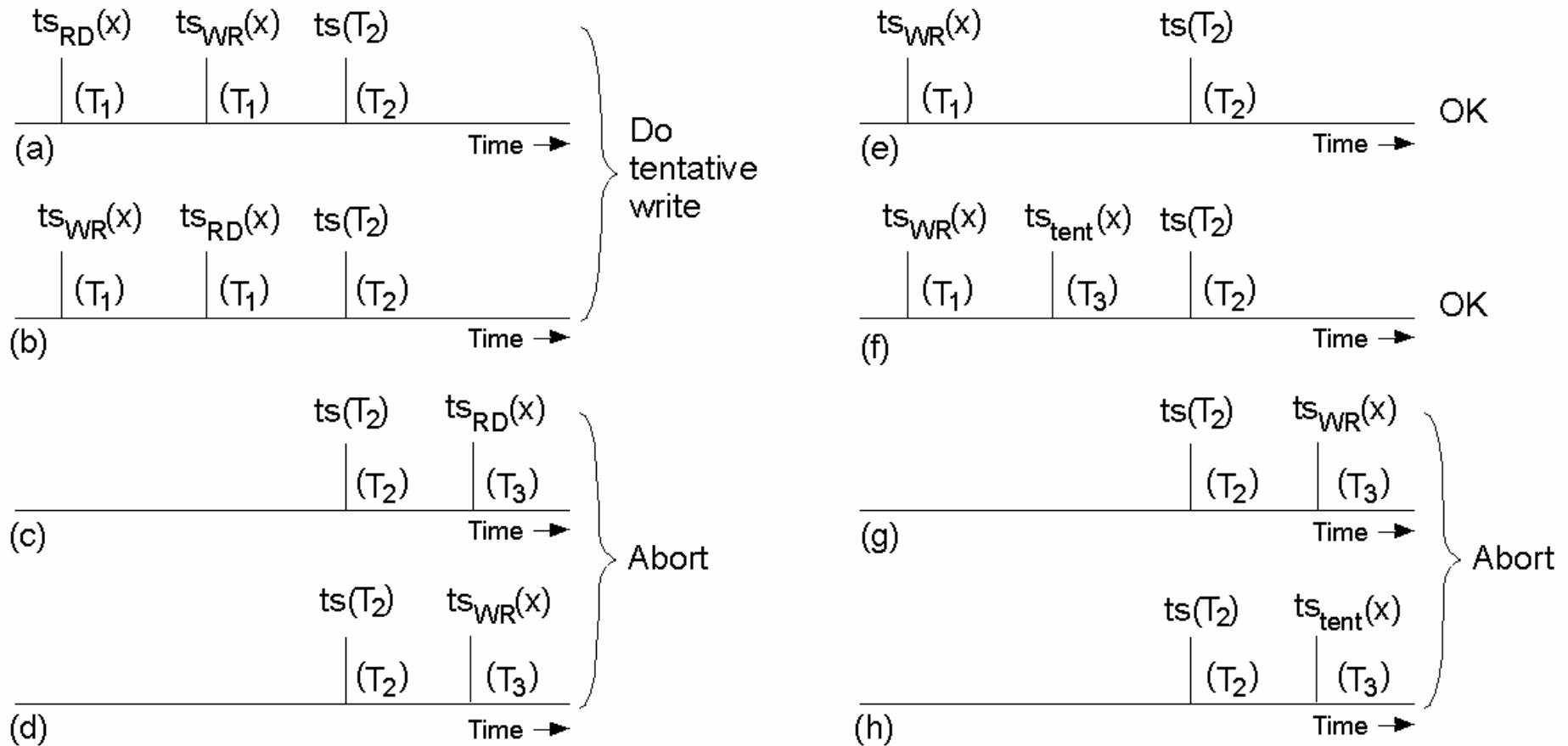
Two-phase locking.

# Two-Phase Locking (2)



Strict two-phase locking.

# Pessimistic Timestamp Ordering



Concurrency control using timestamps.