Synchronization
Part 2

REK's adaptation of Claypool's adaptation of Tanenbaum's

Distributed Systems Chapter 5
and

Silberschatz Chapter 17

Outline – Part 2

- *Clock Synchronization*
- *Clock Synchronization Algorithms*
- *Logical Clocks*
- → *Election Algorithms*
- *Mutual Exclusion*
- *Distributed Transactions*
- *Concurrency Control*

Election Algorithms

- *Many distributed algorithms such as mutual exclusion and deadlock detection require a coordinator process.*
- *When the coordinator process fails, the distributed group of processes must execute an election algorithm to determine a new coordinator process.*
- *These algorithms will assume that each active process has a unique priority id.*

The Bully Algorithm

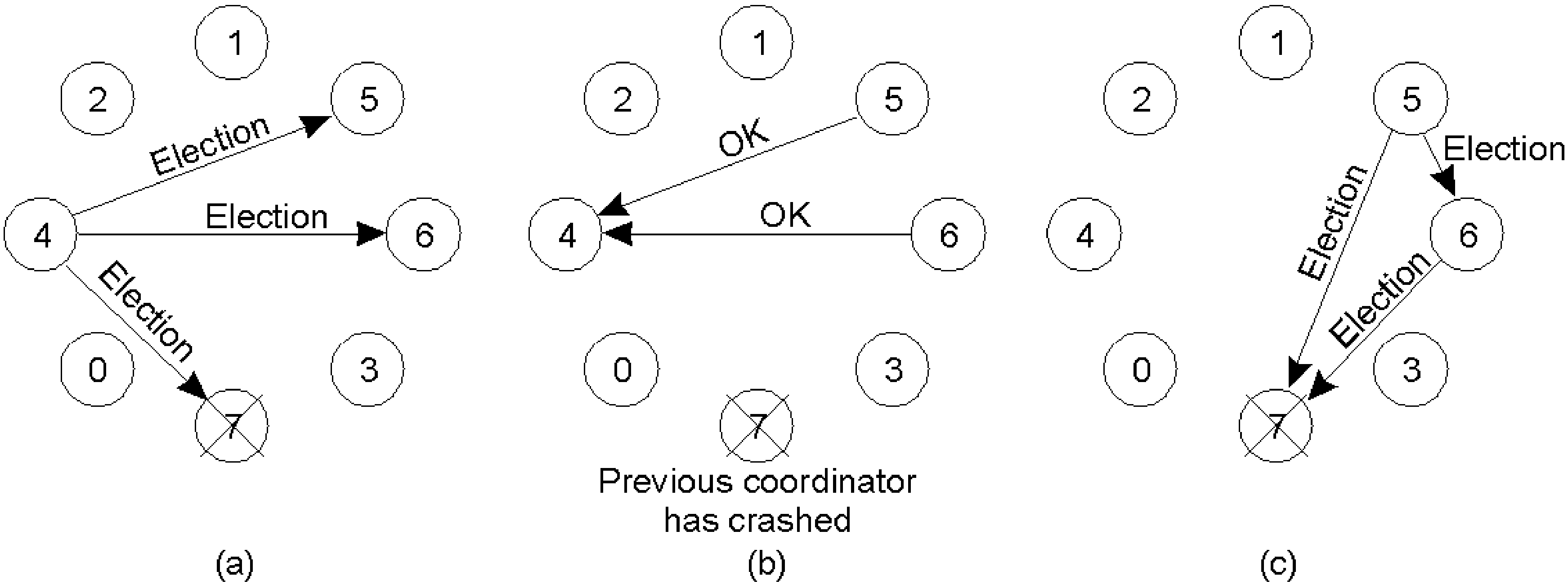
When any process, P, notices that the coordinator is no longer responding it initiates an election:

1. *P sends an election message to all processes with higher id numbers.*
2. *If no one responds, P wins the election and becomes coordinator.*
3. *If a higher process responds, it takes over. Process P's job is done.*

The Bully Algorithm

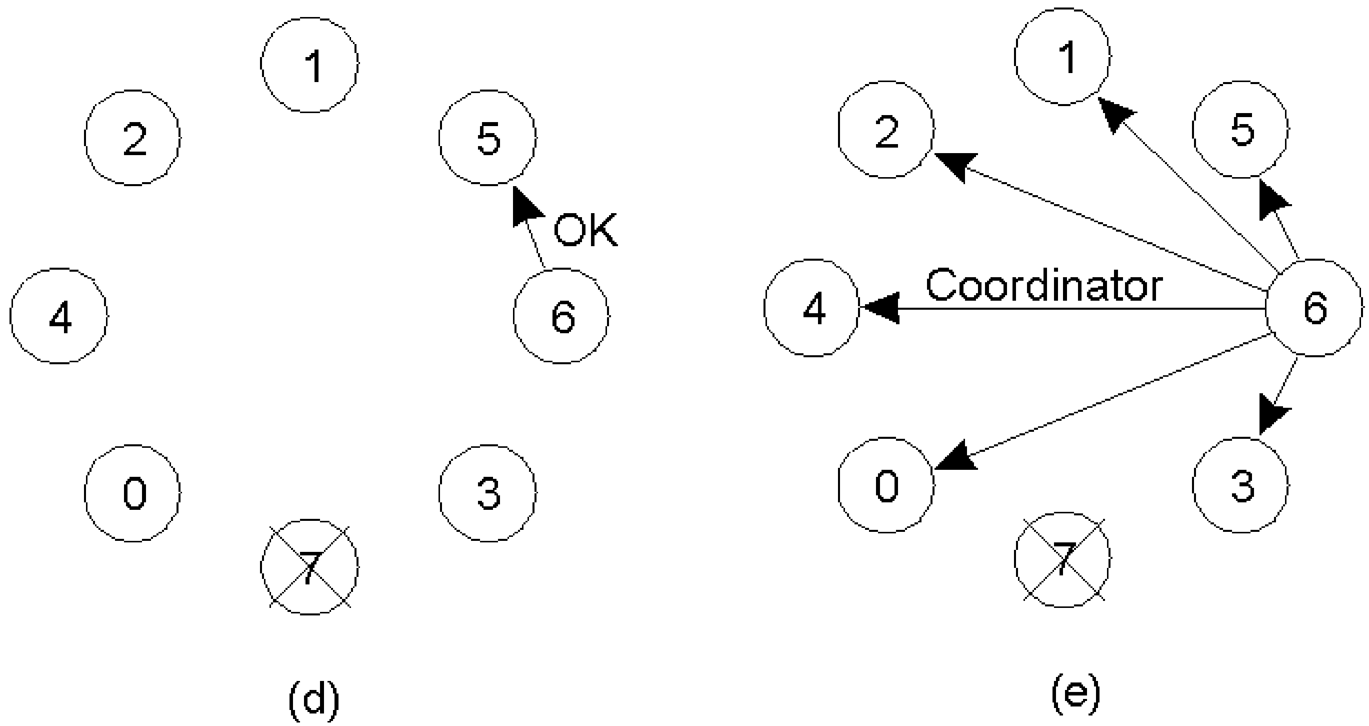
- *At any moment, a process can receive an election message from one of its lower-numbered colleagues.*
- *The receiver sends an OK back to the sender and conducts its own election.*
- *Eventually only the bully process remains. The bully announces victory to all processes in the distributed group.*

Bully Algorithm Example



- *Process 4 notices 7 down.*
- *Process 4 holds an election.*
- *Process 5 and 6 respond, telling 4 to stop.*
- *Now 5 and 6 each hold an election.*

Bully Algorithm Example



- *Process 6 tells process 5 to stop.*
- *Process 6 (the bully) wins and tells everyone.*
- *If processes 7 comes up, starts elections again.*

A Ring Algorithm

- *Assume the processes are logically ordered in a ring {implies a successor pointer and an active process list} that is unidirectional.*

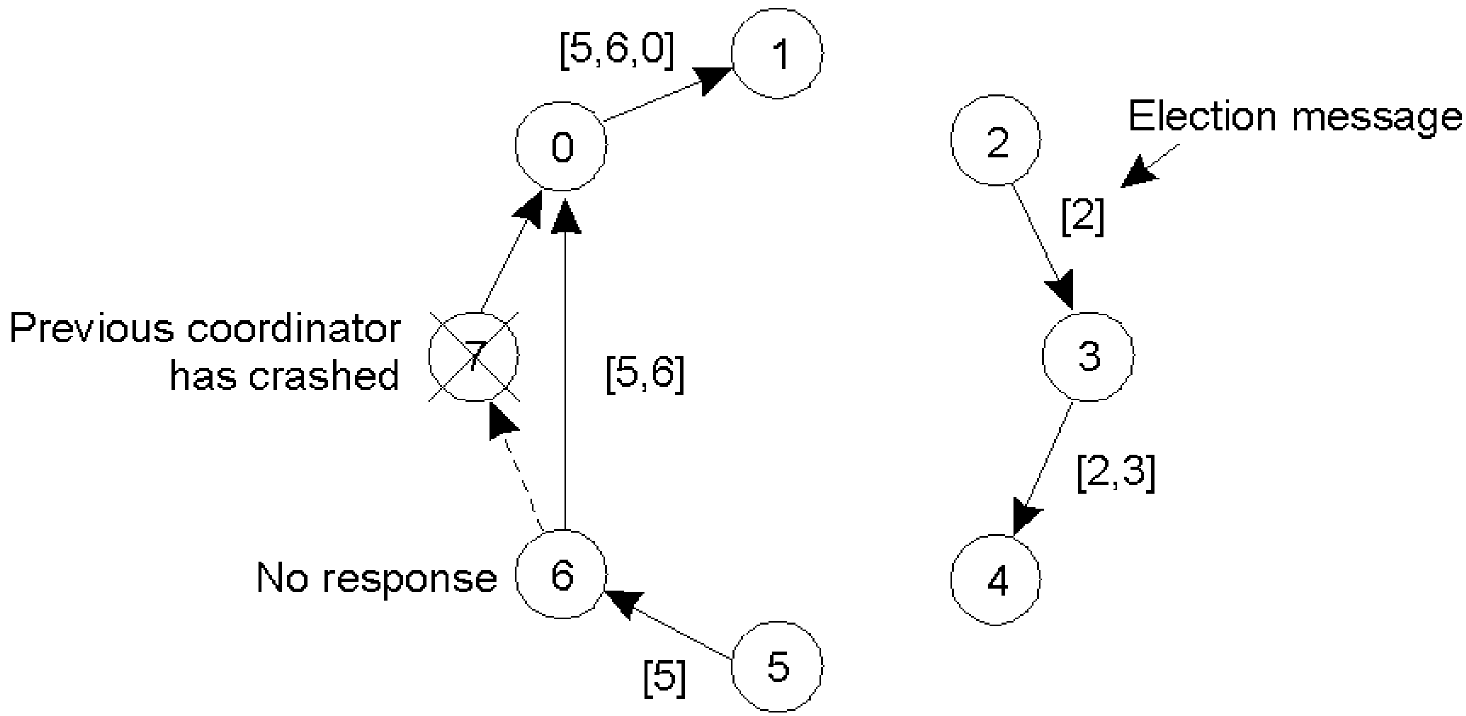
When any process, P, notices that the coordinator is no longer responding it initiates an election:

1. *P sends message containing P's process id to the next available successor.*

A Ring Algorithm

2. *At each active process, the receiving process adds its process number to the list of processes in the message and forwards it to its successor.*
3. *Eventually, the message gets back to the sender.*
4. *The initial sender sends out a second message letting everyone know who the coordinator is {the process with the highest number} and indicating the current members of the active list of processes.*

A Ring Algorithm



- Even if two ELECTIONS start at once, everyone will pick the same leader.

Outline – Part 2

- Clock Synchronization
- Clock Synchronization Algorithms
- Logical Clocks
- Election Algorithms
- → Mutual Exclusion
- Distributed Transactions
- Concurrency Control

Mutual Exclusion

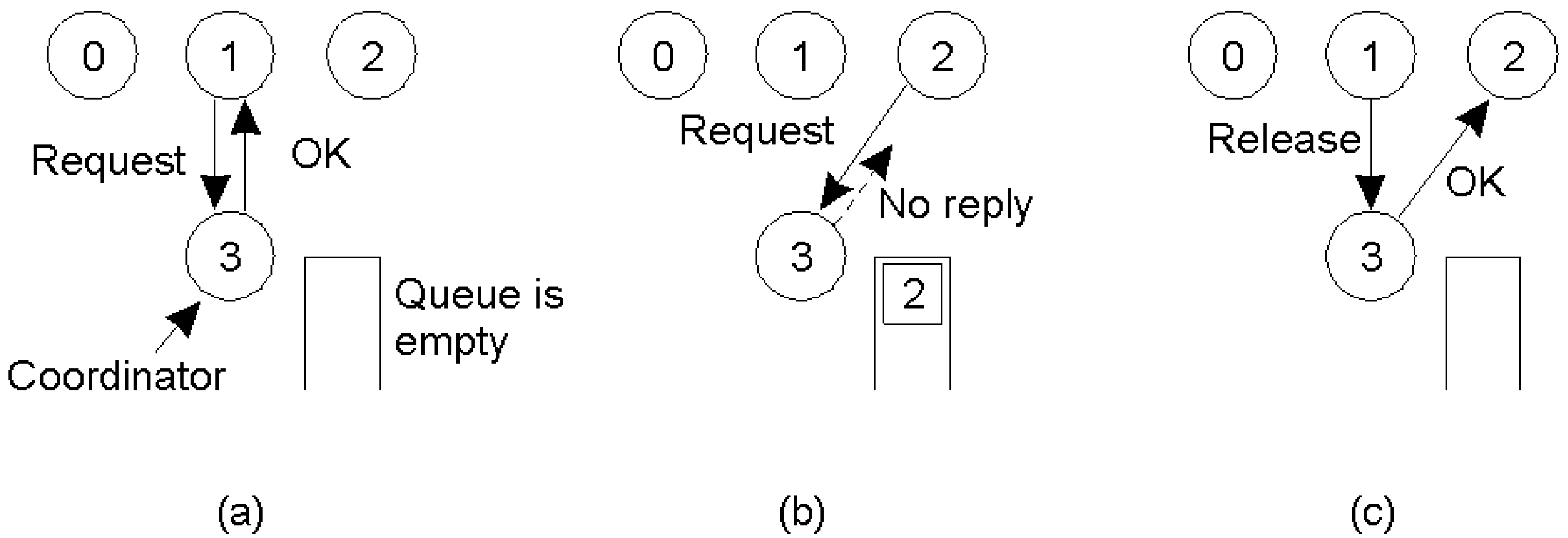
- *To guarantee consistency among distributed processes that are accessing shared memory, it is necessary to provide mutual exclusion when accessing a critical section.*
- *Assume n processes.*

A Centralized Algorithm for Mutual Exclusion

Assume a coordinator has been elected.

- *A process sends a message to the coordinator requesting permission to enter a critical section. If no other process is in the critical section, permission is granted.*
- *If another process then asks permission to enter the same critical region, the coordinator does not reply (Or, it sends "permission denied") and queues the request.*
- *When a process exits the critical section, it sends a message to the coordinator.*
- *The coordinator takes first entry off the queue and sends that process a message granting permission to enter the critical section.*

A Centralized Algorithm for Mutual Exclusion



A Distributed Algorithm for Mutual Exclusion

Ricart and Agrawala algorithm (1981) assumes there is a mechanism for "totally ordering of all events" in the system (e.g. Lamport's algorithm) and a reliable message system.

1. *A process wanting to enter critical sections (cs) sends a message with (cs name, process id, current time) to all processes (including itself).*
2. *When a process receives a cs request from another process, it reacts based on its current state with respect to the cs requested. There are three possible cases:*

A Distributed Algorithm for Mutual Exclusion (cont.)

a) If the receiver is not in the cs and it does not want to enter the cs, it sends an OK message to the sender.

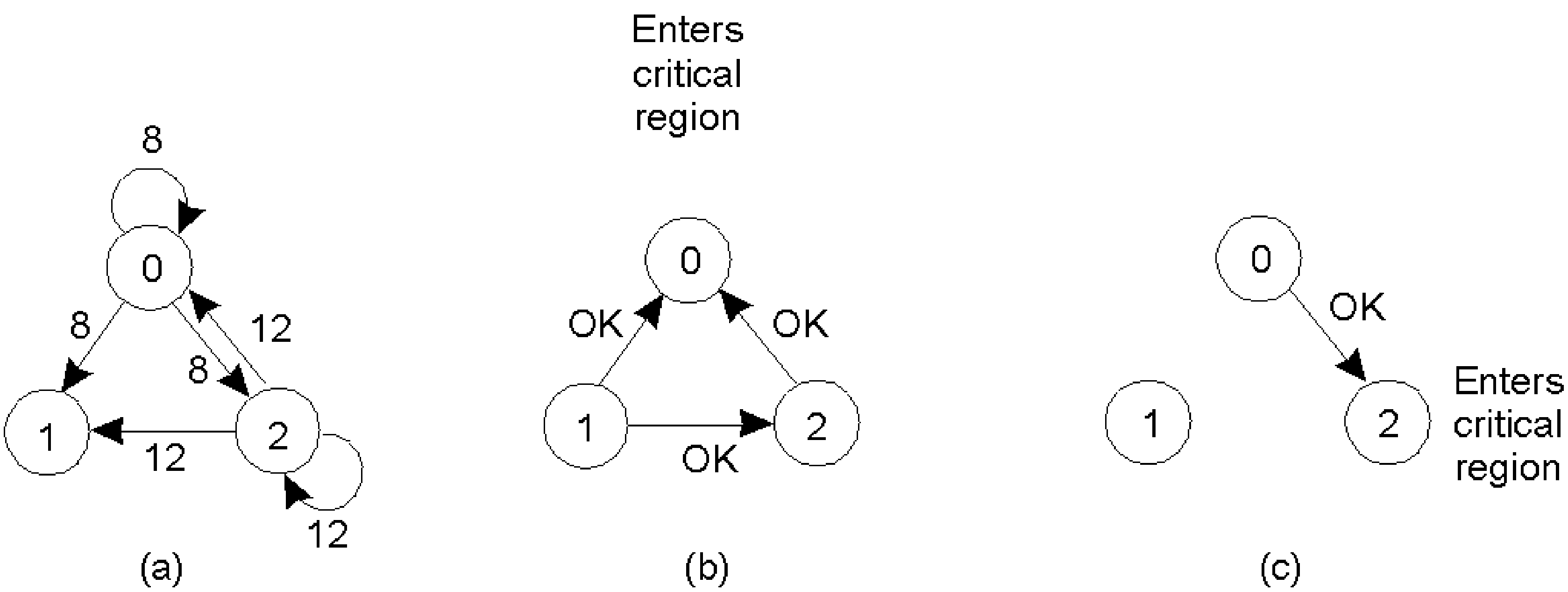
b) If the receiver is in the cs, it does not reply and queues the request.

c) If the receiver wants to enter the cs but has not yet, it compares the timestamp of the incoming message with the timestamp of its message sent to everyone. **{The lowest timestamp wins.}** If the incoming timestamp is lower, the receiver sends an OK message to the sender. If its own timestamp is lower, the receiver queues the request and sends nothing.

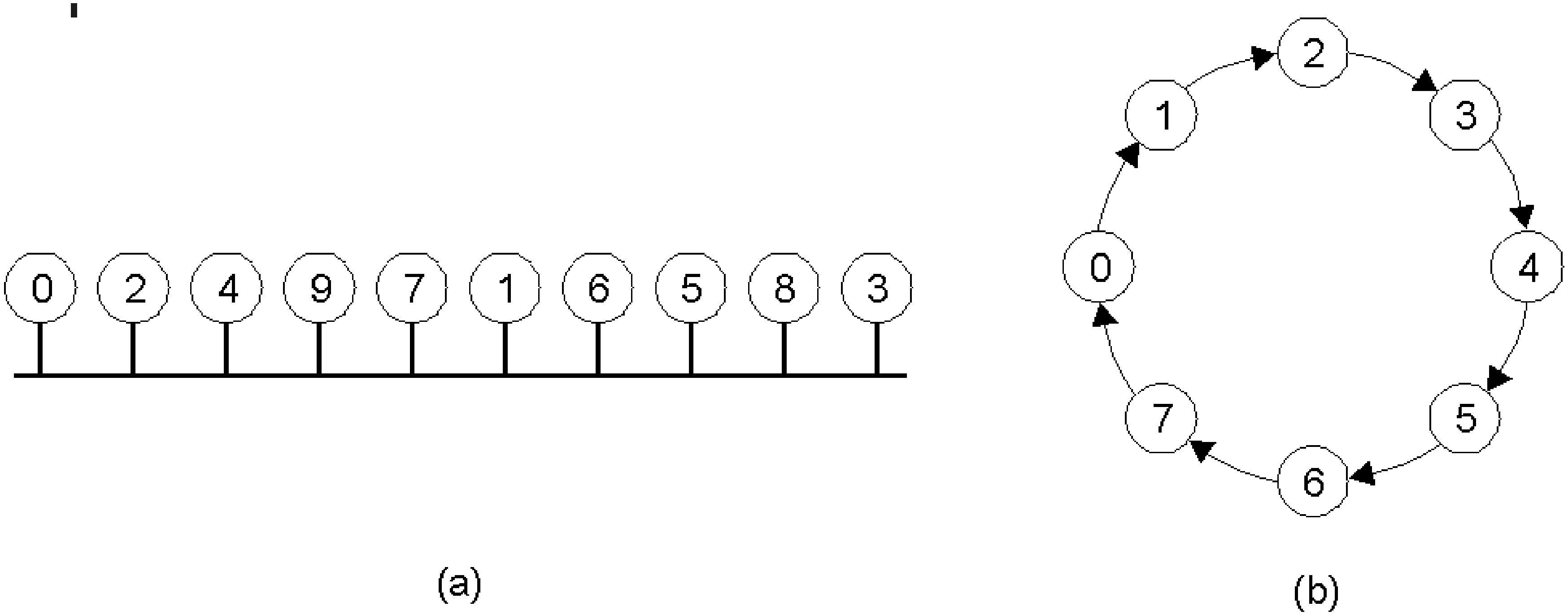
A Distributed Algorithm for Mutual Exclusion (cont.)

- *After a process sends out a request to enter a cs, it waits for an OK from all the other processes. When all are received, it enters the cs.*
- *Upon exiting cs, it sends OK messages to all processes on its queue for that cs and deletes them from the queue.*

A Distributed Algorithm for Mutual Exclusion



A Token Ring Algorithm



a) *An unordered group of processes on a network.*
b) *A logical ring constructed in software.*
- *A process must have token to enter.*

Mutual Exclusion Algorithm 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)	Process crash
Token ring	1 to ∞	0 to n – 1	Lost token, process crash

- *Centralized is the most efficient.*
- *Token ring efficient when many want to use critical region.*

Outline – Part 2

- *Clock Synchronization*
- *Clock Synchronization Algorithms*
- *Logical Clocks*
- *Election Algorithms*
- *Mutual Exclusion*
- → *Distributed Transactions*
- *Concurrency Control*

The Transaction Model

- The transaction model ensures mutual exclusion and supports atomic operations.
- Consider using PC to:
  - Withdraw $100 from account 1
  - Deposit $100 to account 2
- Interruption of the transaction is the problem. In distributed systems, this happens when a connection is broken.

The Transaction Model

- *If a transaction involves multiple actions or operates on multiple resources in a sequence, the transaction by definition is a single, atomic action. Namely,*
  - *It all happens, or none of it happens.*
  - *If process backs out, the state of the resources is as if the transaction never started. {This may require a rollback mechanism.}*

Transaction Primitives

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

The primitives may be system calls, libraries or statements in a language (Sequential Query Language or SQL).

Example: Reserving Flight from White Plains to Nairobi

```
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.*

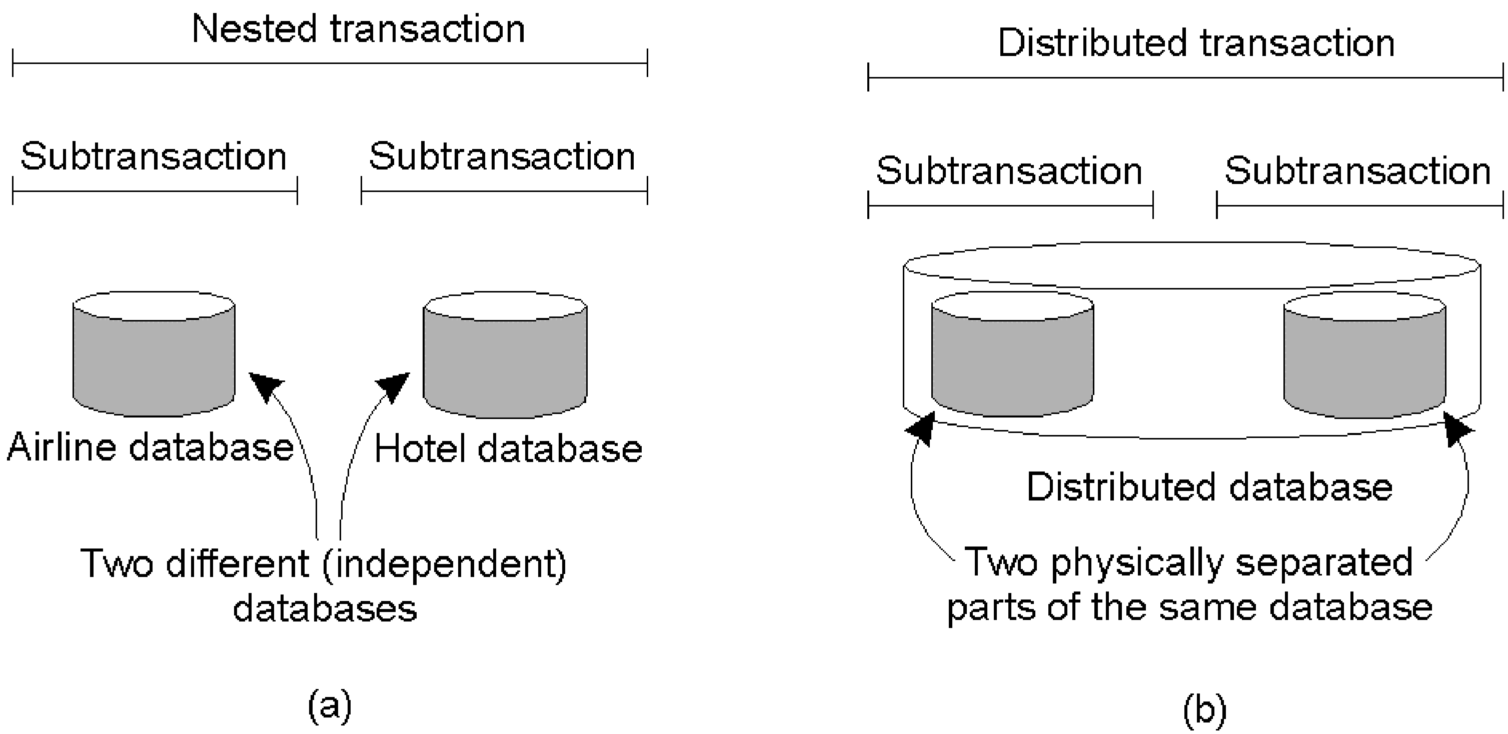
Transaction Properties [ACID]

1) *Atomic: transactions are indivisible to the outside world.*
2) *Consistent: system invariants are not violated.*
3) *Isolated: concurrent transactions do not interfere with each other. {serializable}*
4) *Durability: once a transaction commits, the changes are permanent. {requires a distributed commit mechanism}*

Classification of Transactions

- *Flat Transactions* {satisfy ACID properties}
  - *Limited – partial results cannot be committed.*
  - *Example: what if want to keep first part of flight reservation? If abort and then restart, those might be gone.*
  - *Example: what if want to move a Web page. All links pointing to it would need to be updated. Requiring a flat transaction could lock resources for a long time.*
- *Also Distributed and Nested Transactions*

Nested vs. Distributed Transactions

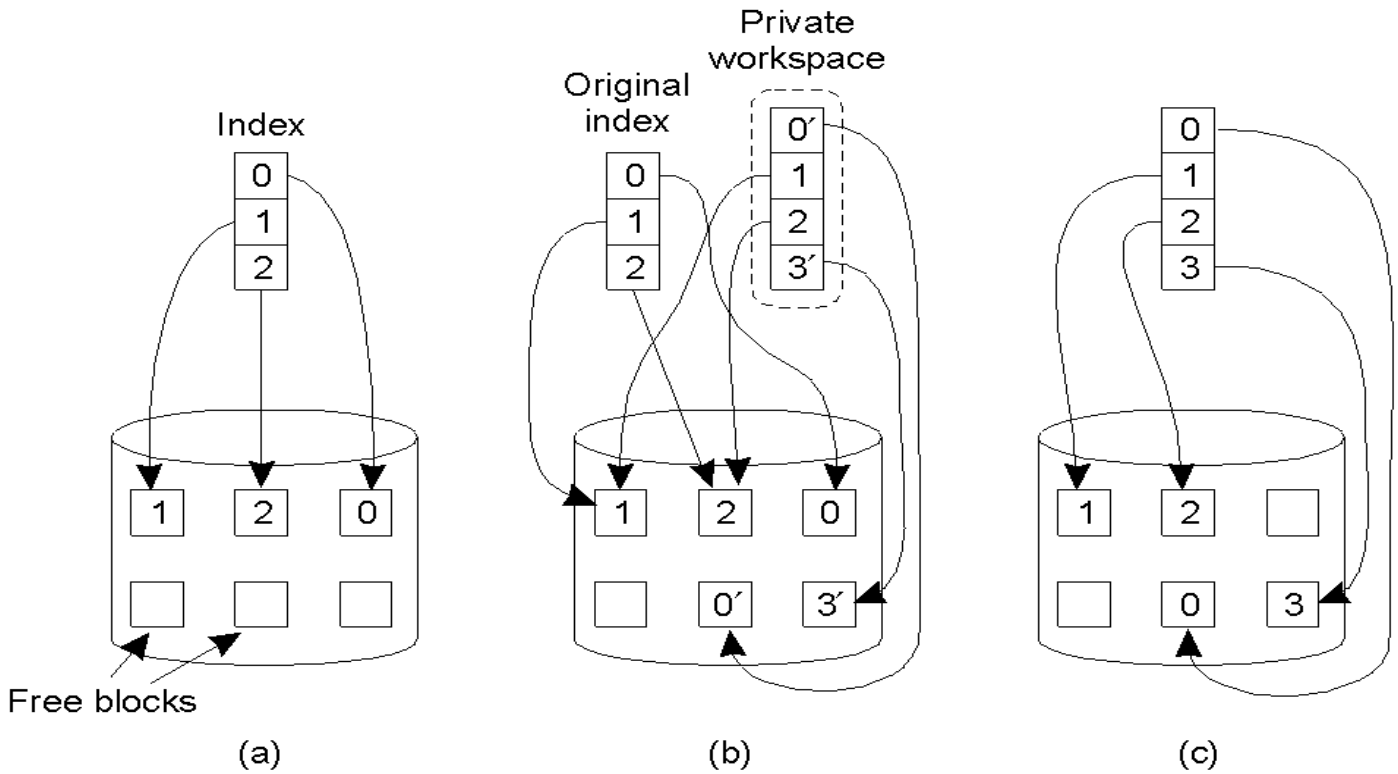


- *Nested transaction gives you a hierarchy*
  - *Commit mechanism is complicated with nesting.*
- *Distributed transaction is "flat" but across distributed data (example: JFK and Nairobi dbase)*

Private Workspace

- File system with transaction across multiple files
  - Normally, updates seen + No way to undo.
- Private Workspace → need to copy files.
- Only update Public Workspace when done.
- If abort transaction, remove private copy.
- But copy can be expensive!

Private Workspace



a) *Original file index (descriptor) and disk blocks*
b) *Copy descriptor only. Copy blocks only when written.*
  - *Modified block 0 and appended block 3 {shadow blocks}*
c) *Replace original file (new blocks plus descriptor) after commit.*

Writeahead Log

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

b) – d) log records old and new values before each statement is executed.

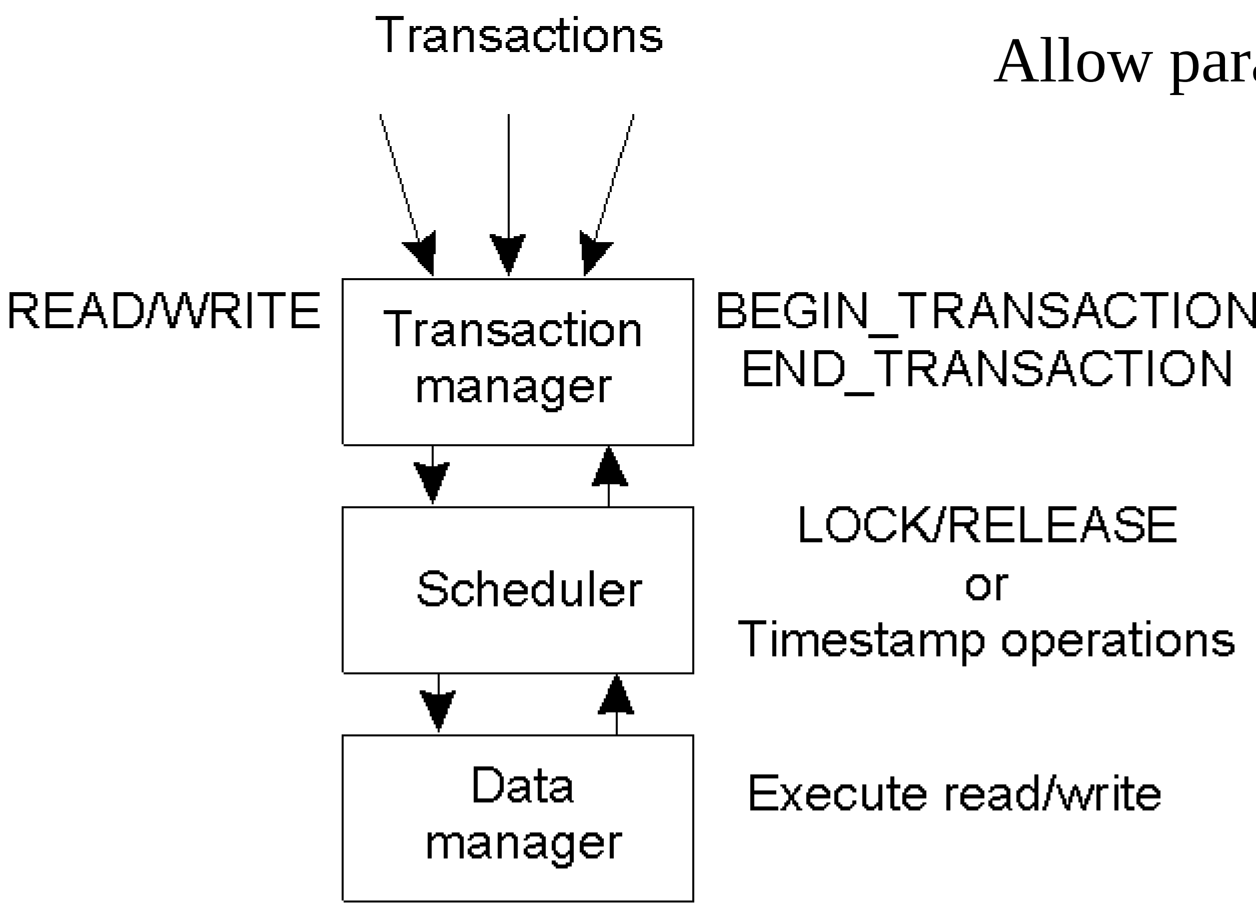
- *If transaction commits, nothing to do.*
- *If transaction is aborted, use log to rollback.*

Outline – Part 2

- *Clock Synchronization*
- *Clock Synchronization Algorithms*
- *Logical Clocks*
- *Election Algorithms*
- *Mutual Exclusion*
- *Distributed Transactions*
- → *Concurrency Control*

Concurrency Control

Allow parallel execution

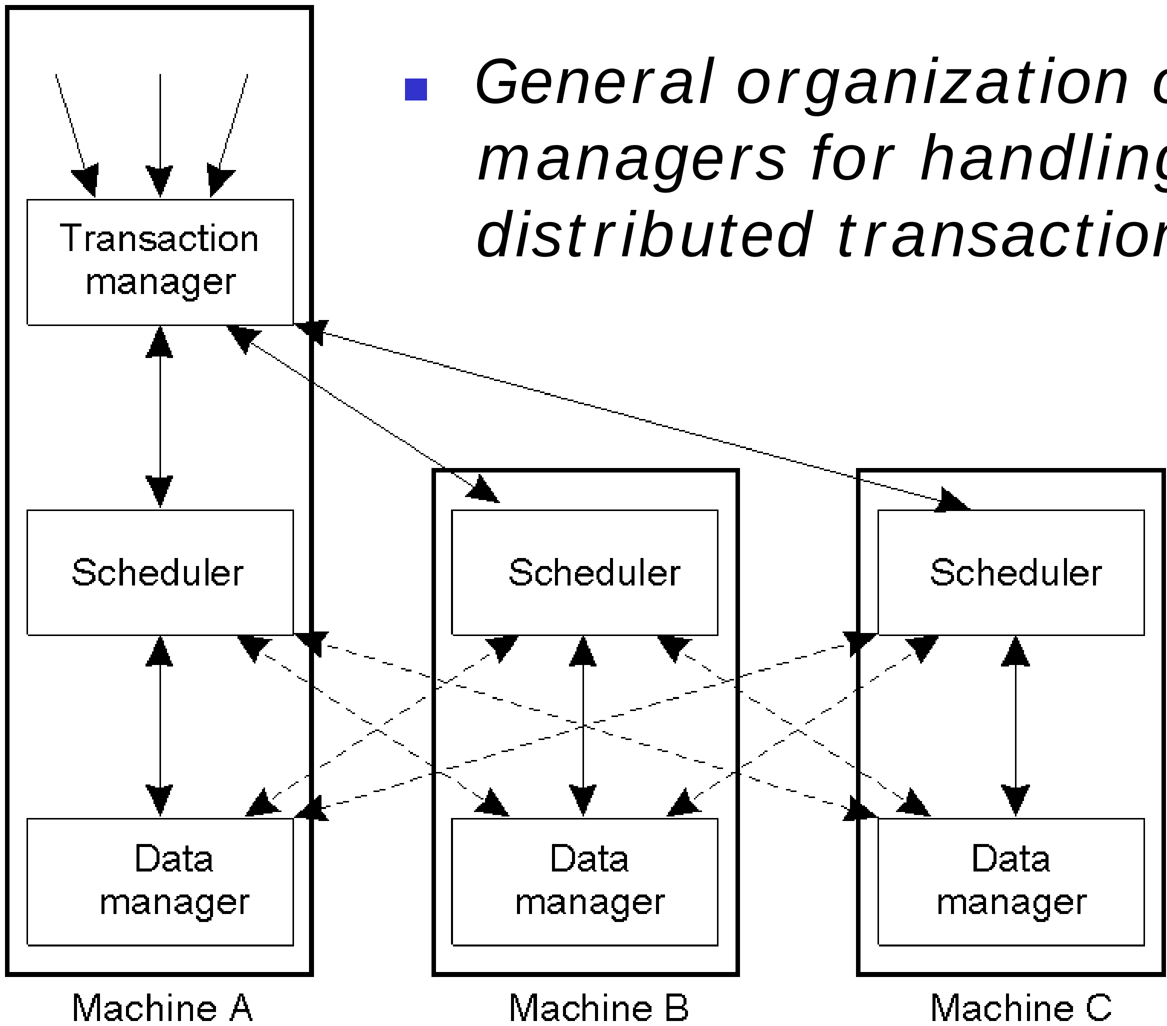


(ensure atomic)

(ensure serial)

General organization of managers for handling transactions.

Concurrency Control



- *General organization of managers for handling distributed transactions.*

Serializability

Allow parallel execution, but end result as if serial

(a)	(b)	(c)

BEGIN_TRANSACTION
x = 0;
x = x + 1;
END_TRANSACTION
	BEGIN_TRANSACTION
	x = 0;
	x = x + 2;
	END_TRANSACTION
		BEGIN_TRANSACTION
		x = 0;
		x = x + 3;
		END_TRANSACTION

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

- Concurrency controller needs to manage

Atomicity

- *Either all the operations associated with a program unit are executed to completion, or none are performed.*
- *Ensuring atomicity in a distributed system requires a local transaction coordinator, which is responsible for the following:*

Silberschatz Slide

Atomicity

- *Starting the execution of the transaction.*
- *Breaking the transaction into a number of subtransactions, and distribution these subtransactions to the appropriate sites for execution.*
- *Coordinating the termination of the transaction, which may result in the transaction being committed at all sites or aborted at all sites.*

- Assume each local site maintains a log for recovery.

Silberschatz Slide

Two-Phase Commit Protocol (2PC)

- *Assumes fail-stop model.*
- *Execution of the protocol is initiated by the coordinator after the last step of the transaction has been reached.*
- *When the protocol is initiated, the transaction may still be executing at some of the local sites.*
- *The protocol involves all the local sites at which the transaction executed.*
- *Example: Let T be a transaction initiated at site S_i and let the transaction coordinator at S_i be C_i.*

Silberschatz Slide

Phase 1: Obtaining a Decision

- C_i adds <prepare T> record to the log.
- C_i sends **<prepare T> message** to all sites.
- When a site receives a ***<prepare T> message***, the transaction manager determines if it can commit the transaction.
  - If no: add <no T> record to the log and respond to C_i with ***<abort T> message***.
  - If yes:
    - add <ready T> record to the log.
    - force all log records for T onto stable storage.
    - transaction manager sends ***<ready T> message*** to C_i.

Silberschatz Slide

Phase 1 (Cont.)

- *Coordinator collects responses*
  - *All respond "ready", decision is commit.*
  - *At least one response is "abort", decision is abort.*
  - *At least one participant fails to respond within time out period, decision is abort.*

Silberschatz Slide

Phase 2: Recording Decision in the Database

- *Coordinator adds a decision record*

  <abort T> or <commit T>

  to its log and forces record onto stable storage.
- *Once that record reaches stable storage it is irrevocable (even if failures occur).*
- *Coordinator sends a message to each participant informing it of the decision (**commit** or **abort message**).*
- *Participants take appropriate action locally.*

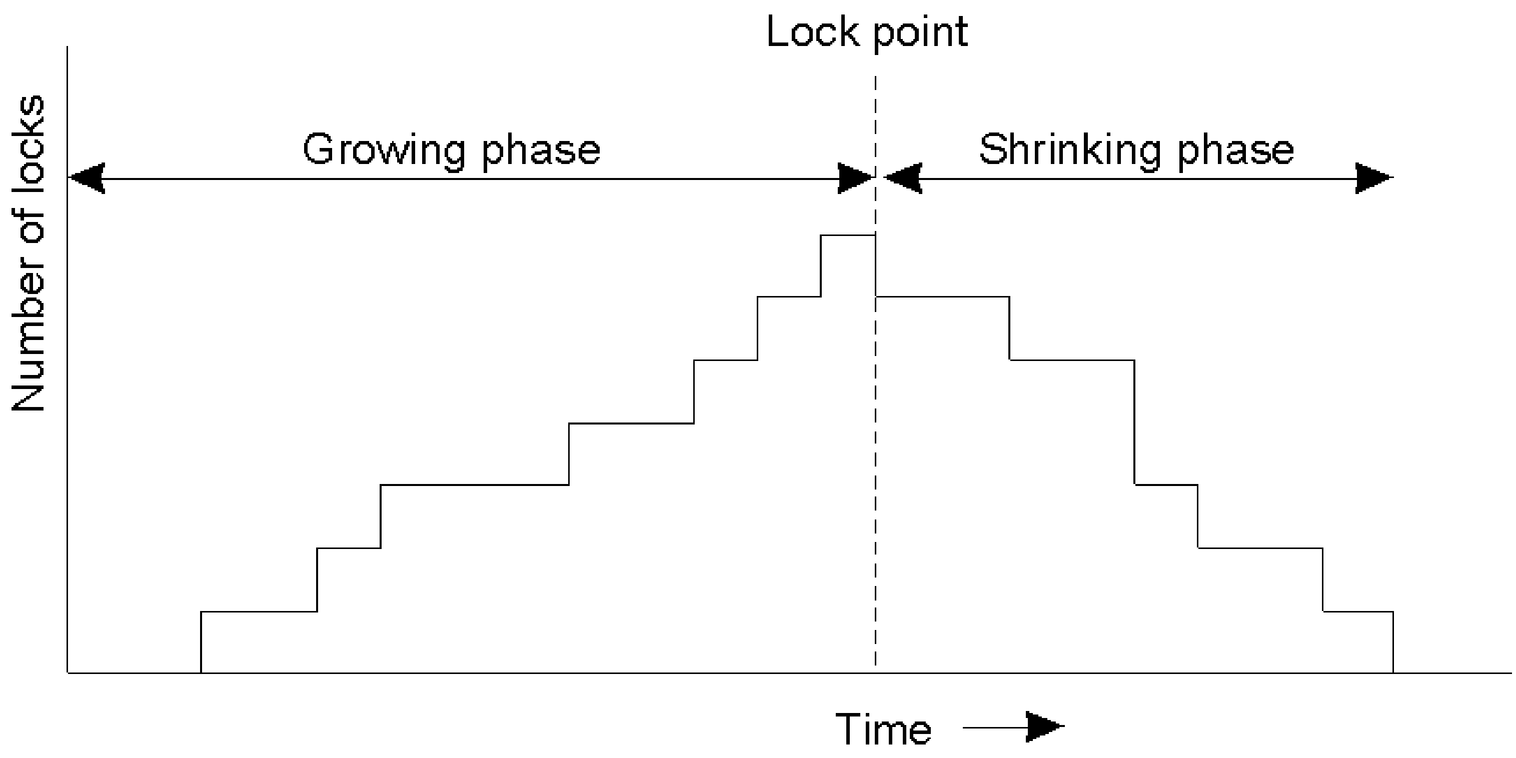
Two-Phase Locking

1. When scheduler receives an operation oper(T,x) from the TM, it tests for operation conflict with any other operation for which it already granted a lock. If conflict, oper(T,x) is delayed. No conflict → lock for x is granted and oper(T,x) is passed to DM.

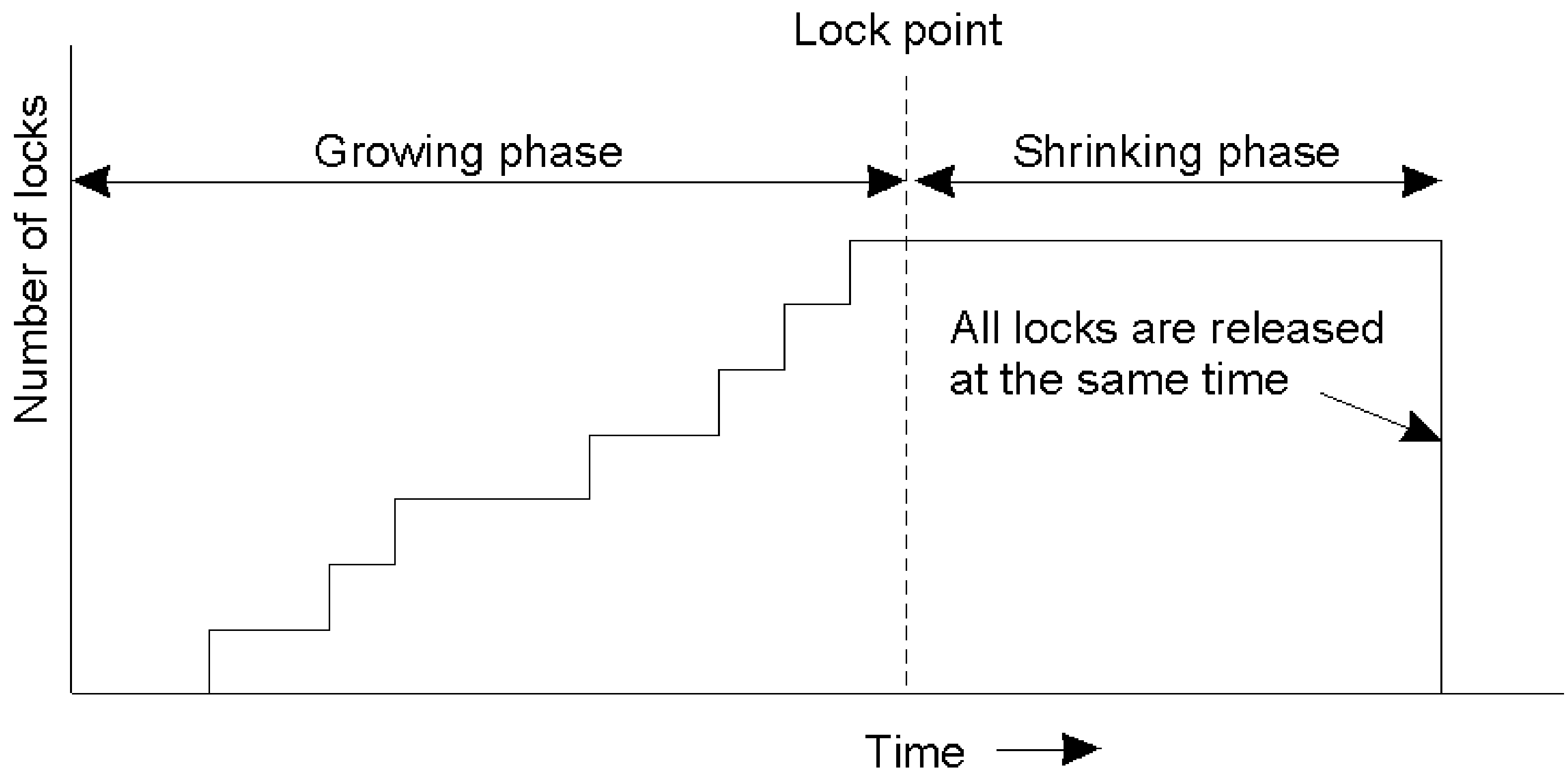
Two-Phase Locking

2. The scheduler will never release a lock for x until DM indicates it has performed the operation for which the lock was set.
3. Once the scheduler has released a lock on behalf of T, T will NOT be permitted to acquire another lock.

Two-Phase Locking



Strict Two-Phase Locking



- *Always reads value written by a committed transaction. → This policy eliminates cascading aborts.*
- *Releasing locks at the end of the transaction means transaction is “unaware” of the release operation.*