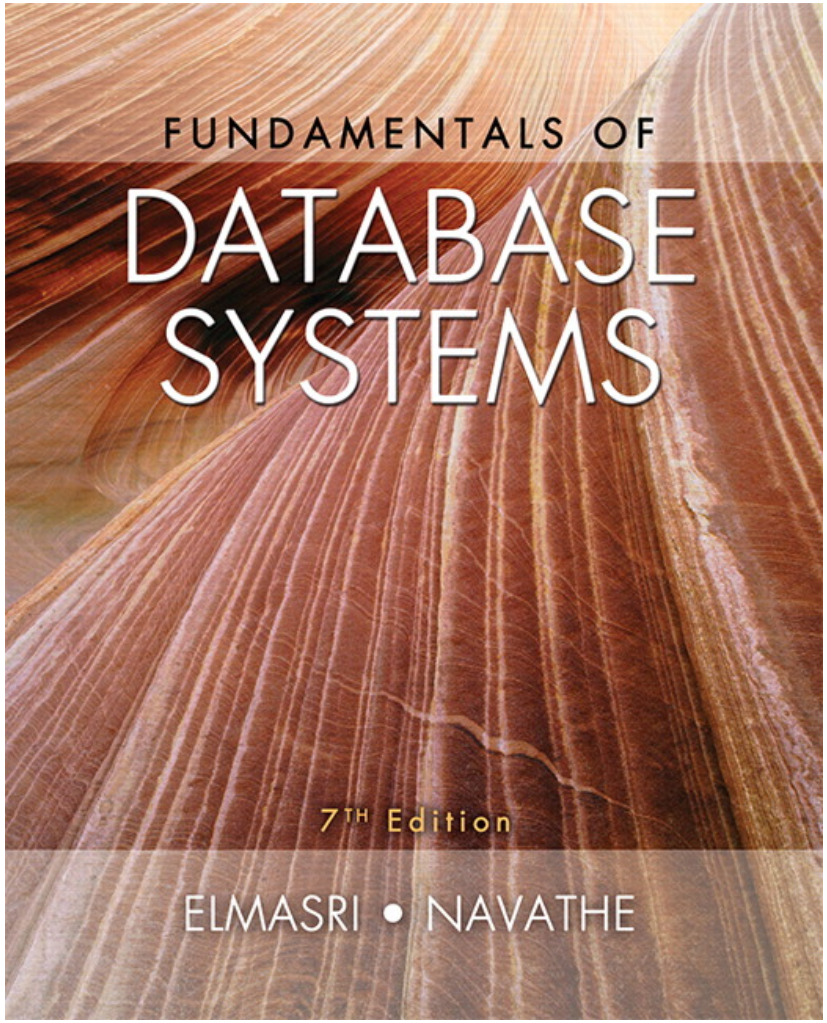


CHAPTER 21

Concurrency Control Techniques

Introduction

- Concurrency control protocols
  - Set of rules to guarantee serializability
- Two-phase locking protocols
  - Lock data items to prevent concurrent access

21.1 Two-Phase Locking Techniques for Concurrency Control

- Lock
  - Variable associated with a data item describing status for operations that can be applied
  - One lock for each item in the database
- Binary locks
  - Two states (values)
    - Locked (1)
      - Item cannot be accessed
    - Unlocked (0)
      - Item can be accessed when requested

Two-Phase Locking Techniques for Concurrency Control (cont'd.)

- Transaction requests access by issuing a lock_item(X) operation

```
lock_item(X):
B:  if LOCK(X) = 0                (*item is unlocked*)
        then LOCK(X) ←1           (*lock the item*)
    else
        begin
        wait (until LOCK(X) = 0
            and the lock manager wakes up the transaction);
        go to B
        end;
unlock_item(X):
    LOCK(X) ← 0;                  (* unlock the item *)
    if any transactions are waiting
        then wakeup one of the waiting transactions;
```

Figure 21.1 Lock and unlock operations for binary locks

Two-Phase Locking Techniques for Concurrency Control (cont'd.)

- Lock table specifies items that have locks
- Lock manager subsystem
  - Keeps track of and controls access to locks
  - Rules enforced by lock manager module
- At most one transaction can hold the lock on an item at a given time
- Binary locking too restrictive for database items

Two-Phase Locking Techniques for Concurrency Control (cont'd.)

- Shared/exclusive or read/write locks
  - Read operations on the same item are not conflicting
  - Must have exclusive lock to write
  - Three locking operations
    - read_lock(X)
    - write_lock(X)
    - unlock(X)

```
read_lock(X):
B:  if LOCK(X) = "unlocked"
        then begin LOCK(X) ← "read-locked";
            no_of_reads(X) ← 1
            end
    else if LOCK(X) = "read-locked"
        then no_of_reads(X) ← no_of_reads(X) + 1
    else begin
            wait (until LOCK(X) = "unlocked"
                and the lock manager wakes up the transaction);
            go to B
            end;
write_lock(X):
B:  if LOCK(X) = "unlocked"
        then LOCK(X) ← "write-locked"
    else begin
            wait (until LOCK(X) = "unlocked"
                and the lock manager wakes up the transaction);
            go to B
            end;
unlock (X):
    if LOCK(X) = "write-locked"
        then begin LOCK(X) ← "unlocked";
            wakeup one of the waiting transactions, if any
            end
    else it LOCK(X) = "read-locked"
        then begin
            no_of_reads(X) ← no_of_reads(X) –1;
            if no_of_reads(X) = 0
                then begin LOCK(X) = "unlocked";
                    wakeup one of the waiting transactions, if any
                    end
            end;
```

Figure 21.2 Locking and unlocking operations for two-mode (read/write, or shared/exclusive) locks

Two-Phase Locking Techniques for Concurrency Control (cont'd.)

- Lock conversion
  - Transaction that already holds a lock allowed to convert the lock from one state to another
- Upgrading
  - Issue a read_lock operation then a write_lock operation
- Downgrading
  - Issue a read_lock operation after a write_lock operation

Locking without 2PL

A=1000, B = 1000

T1	T2
write_lock(A)	
read_item(A)	
	read_lock(A)
A=A-100	
write_item(A)	
unlock(A)	
	R(A)
	unlock(A)
	read_lock(B)
write_lock(B)	
	read_item(B)
	unlock(B)
read(B)	print A + B
B=B+100	
write_item(B)	
unlock(B)	

Guaranteeing Serializability by Two-Phase Locking

- Two-phase locking protocol
  - All locking operations precede the first unlock operation in the transaction
  - Phases
    - Expanding (growing) phase
      - New locks can be acquired but none can be released
      - Lock conversion upgrades must be done during this phase
    - Shrinking phase
      - Existing locks can be released but none can be acquired
      - Downgrades must be done during this phase

2PL Example

A=1000, B = 1000

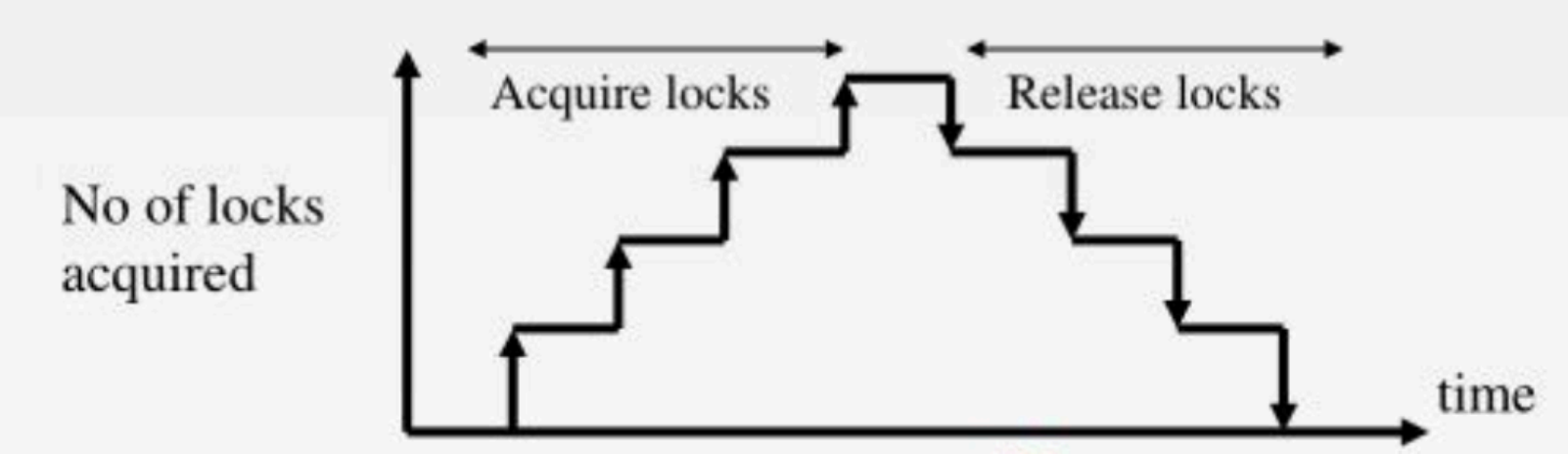
T1	T2
write_lock(A)	
read_item(A)	read_lock(A)
A=A-100	
write_item(A)	
write_lock(B)	
unlock(A)	read_item(A)
	read_lock(B)
read(B)	
B=B+100	
write_item(B)	
unlock(B)	read_item(B)
	unlock(A)
	unlock(B)
	print A + B

Guaranteeing Serializability by Two-Phase Locking

- If every transaction in a schedule follows the two-phase locking protocol, schedule guaranteed to be serializable
- Two-phase locking may limit the amount of concurrency that can occur in a schedule
- Some serializable schedules will be prohibited by two-phase locking protocol

Variations of Two-Phase Locking

- Basic 2PL
  - Technique described on previous slides



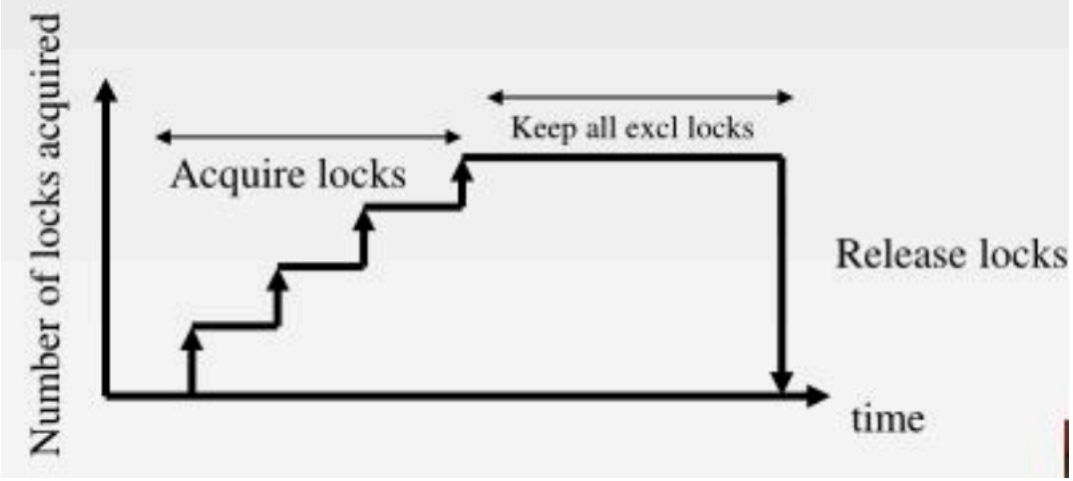
- Problems
  - Deadlocks
  - Cascading aborts.

Variations of Two-Phase Locking

- Conservative (static) 2PL
  - Requires a transaction to lock all the items it accesses before the transaction begins
    - Predeclare read-set and write-set
  - Deadlock-free protocol

Variations of Two-Phase Locking (cont'd.)

- Strict 2PL
  - Transaction does not release exclusive locks until after it commits or aborts



Variations of Two-Phase Locking (cont'd.)

- Rigorous 2PL
  - Transaction does not release any locks until after it commits or aborts
- Concurrency control subsystem responsible for generating read_lock and write_lock requests
- Locking generally considered to have high overhead

Dealing with Deadlock and Starvation

- Deadlock
  - Occurs when each transaction T in a set is waiting for some item locked by some other transaction *T'*
  - Both transactions stuck in a waiting queue

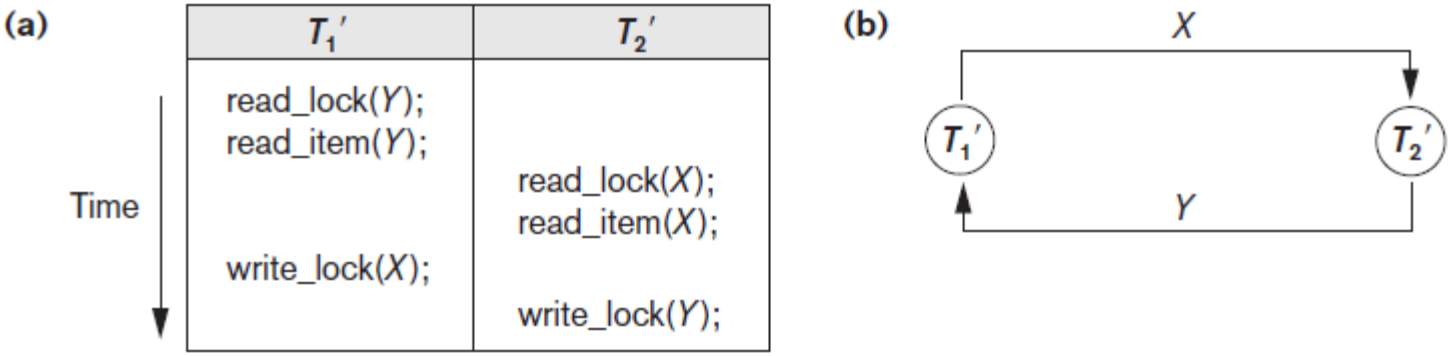


Figure 21.5 Illustrating the deadlock problem (a) A partial schedule of *T*1′ and *T*2′ that is in a state of deadlock (b) A wait-for graph for the partial schedule in (a)

Dealing with Deadlock and Starvation (cont'd.)

- Deadlock prevention protocols
  - Every transaction locks all items it needs in advance
  - Ordering all items in the database
    - Transaction that needs several items will lock them in that order
  - Both approaches impractical

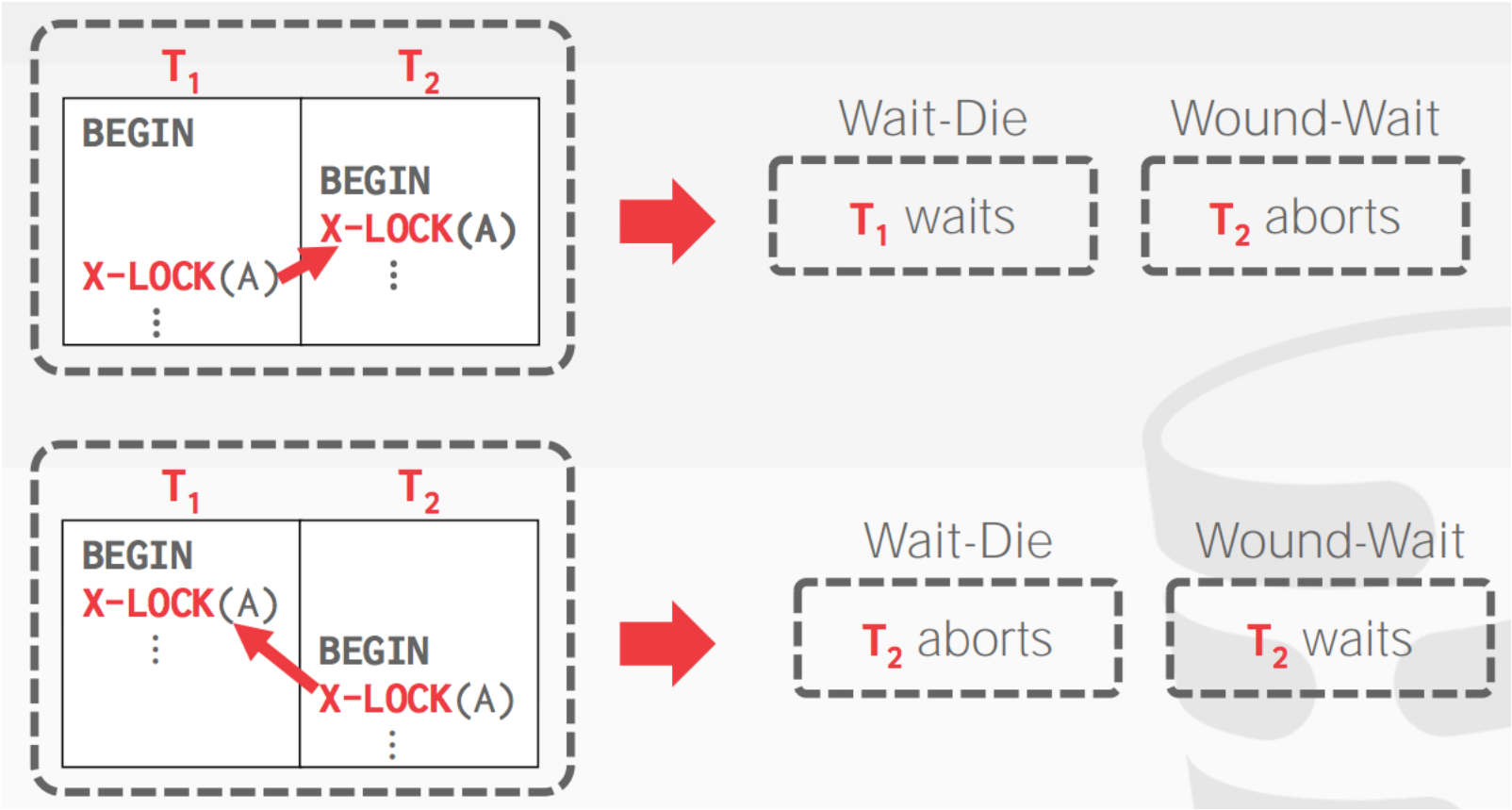
Dealing with Deadlock and Starvation (cont'd.)

- Protocols based on a timestamp:

Older Timestamp = Higher Priority (e.g., T1 > T2)

  - Wait-Die ("Old Waits for Young")
    - If requesting T has higher priority than holding T, then requesting T waits for holding T.
    - Otherwise requesting T aborts.
  - Wound-Wait ("Young Waits for Old")
    - If requesting T has higher priority than holding T, then holding T aborts and releases lock.
    - Otherwise requesting T waits.

Dealing with Deadlock and Starvation (cont'd.)



Dealing with Deadlock and Starvation (cont'd.)

- No waiting algorithm
  - If transaction unable to obtain a lock, immediately aborted and restarted later
- Cautious waiting algorithm
  - Deadlock-free
- Deadlock detection
  - System checks to see if a state of deadlock exists
  - Wait-for graph

Dealing with Deadlock and Starvation (cont'd.)

- Victim selection
  - Deciding which transaction to abort in case of deadlock
- Timeouts
  - If system waits longer than a predefined time, it aborts the transaction
- Starvation
  - Occurs if a transaction cannot proceed for an indefinite period of time while other transactions continue normally
  - Solution: first-come-first-served queue

Exercise

Here, S(·) and X(·) stand for 'Shared Lock' and 'Exclusive Lock', respectively. T1, T2, and T3 represent three transactions. LM stands for 'lock manager' and transactions will never release a granted lock.

Time	t_1	t_2	t_3	t_4	t_5	t_6	t_7
T_1	X(P)						S(R)
T_2			S(Q)	S(R)		S(P)	
T_3		S(R)			X(Q)		
LM							

For the lock requests in above table, decide which lock will be granted or blocked by LM. Write 'GRNT' in the LM row to indicate the lock is granted and 'BLCK' to indicate that the lock is blocked (the lock acquisition fails).
Moreover, provide a brief explanation to justify your answer.

Exercise

- Does the following schedule cause a deadlock?

T1	T2	T3
read_lock(A)		
	write_lock(B)	
		read_lock(C)
read_lock(B)		
	write_lock(C)	
		write_lock(A)