CSCI 5408

Dr. Saurabh Dey

saurabh.dey@dal.ca

Content

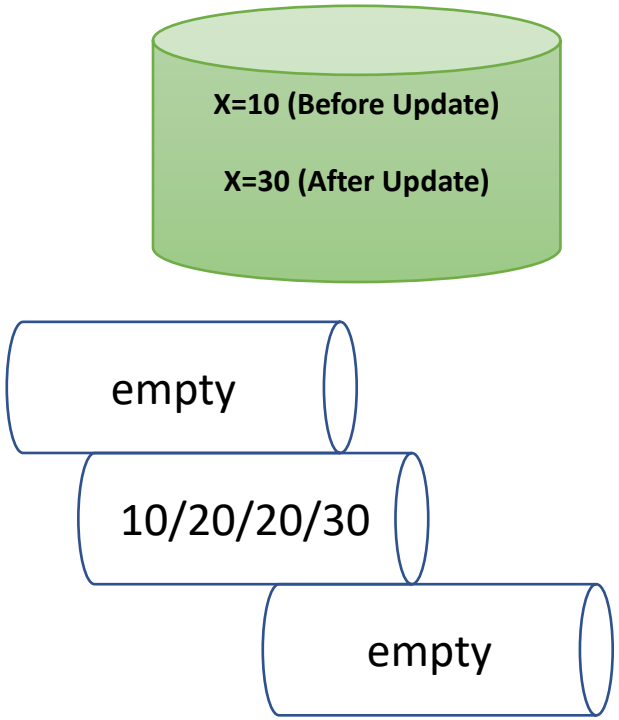
1. Concurrency Control
2. Locking

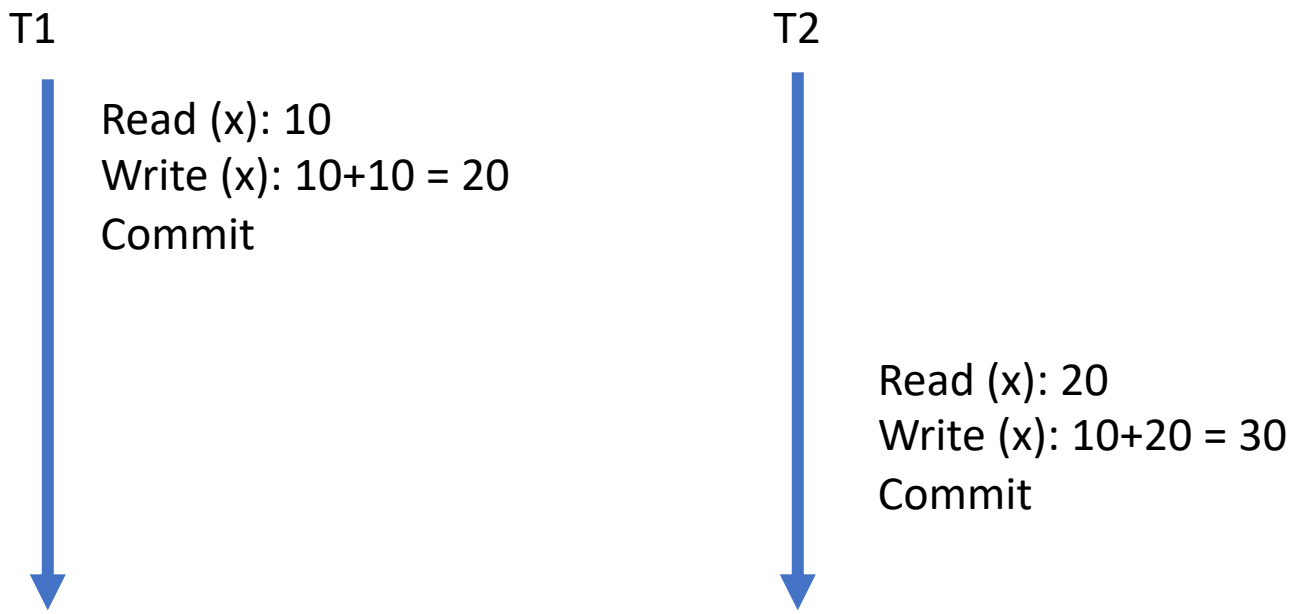
Schedule – Serial

- In multi-transaction environment, the set of operations performed by transactions is known as a **Schedule**.

S = {T1_Read(x), T1_ Write(x), T1_ Commit, T2_ Read(x), T2_ Write(x), T2_ Commit}

- The process of execution is known as **Scheduling**. The execution is managed by **Scheduler**

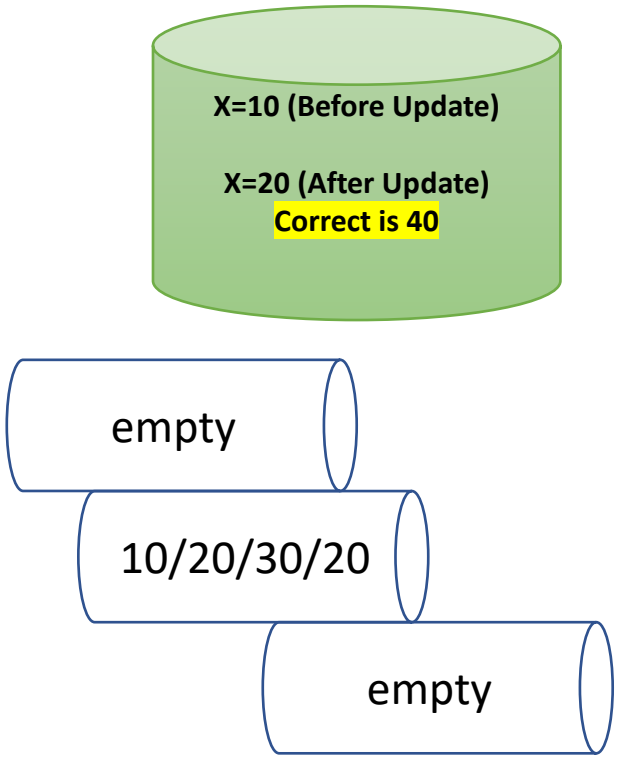


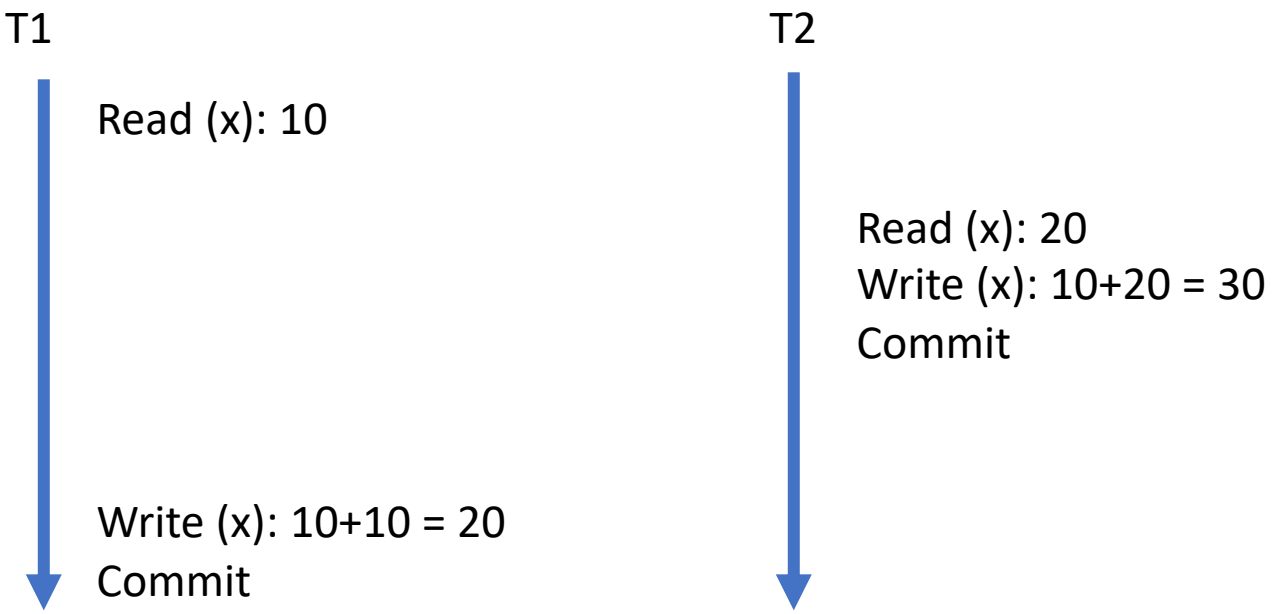


Schedule Non-Serial

- In reality, the schedule S is non-serial.
- How to maintain consistency when transactions are interleaving?

S = {T1_Read(x), T2_ Read(x), T2_ Write(x), T2_ Commit, T1_ Write(x), T1_ Commit}





Why do we need Concurrency Control?

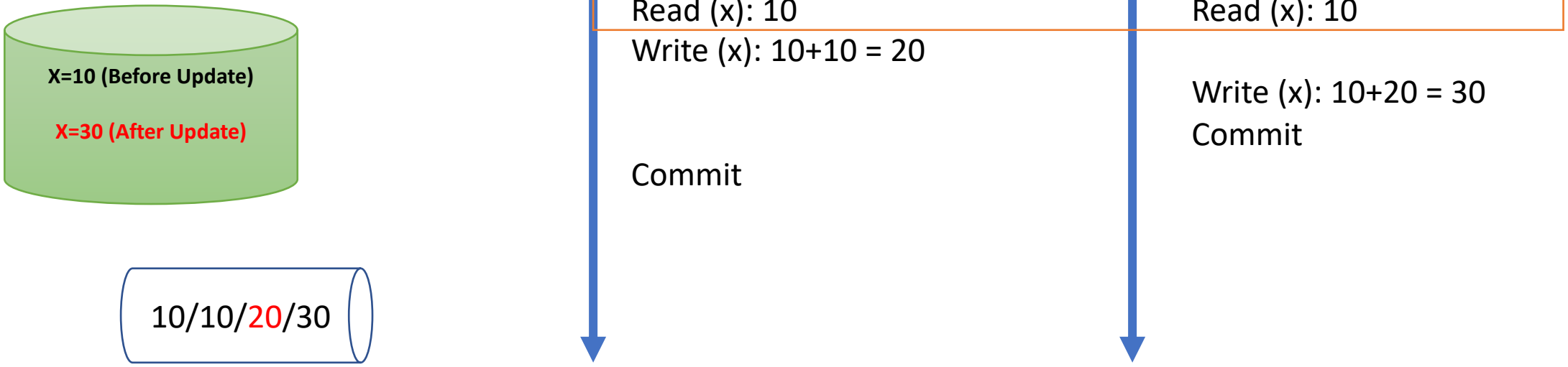
- Multiple simultaneous execution of transactions may create inconsistencies. Serializability is needed

Problems if not done:

- Lost Update
- Uncommitted data or Dirty read
- Inconsistent retrievals
- Phantom read

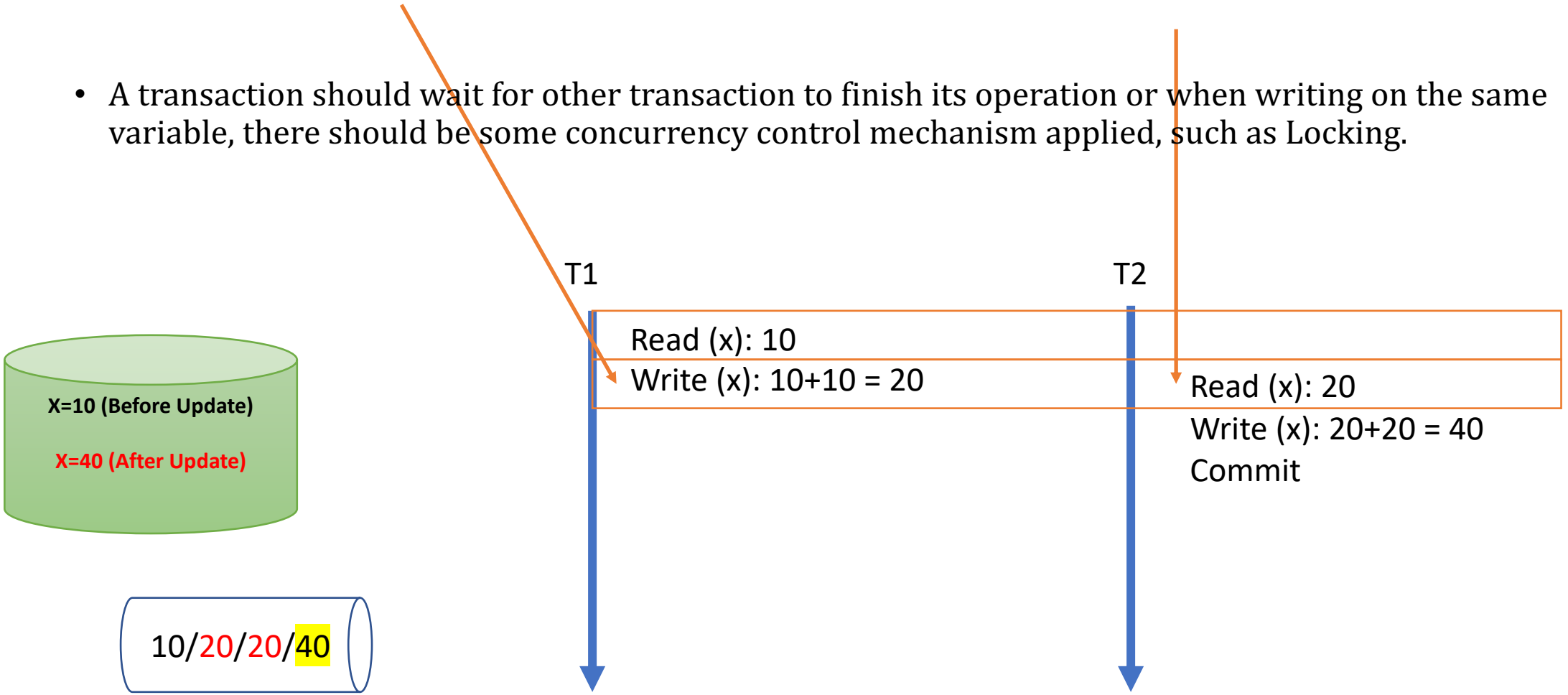
A. Lost Update

- Lost update happens when one transaction's update is overwritten by another transaction.
- If a transaction performs blind write, this may happen.



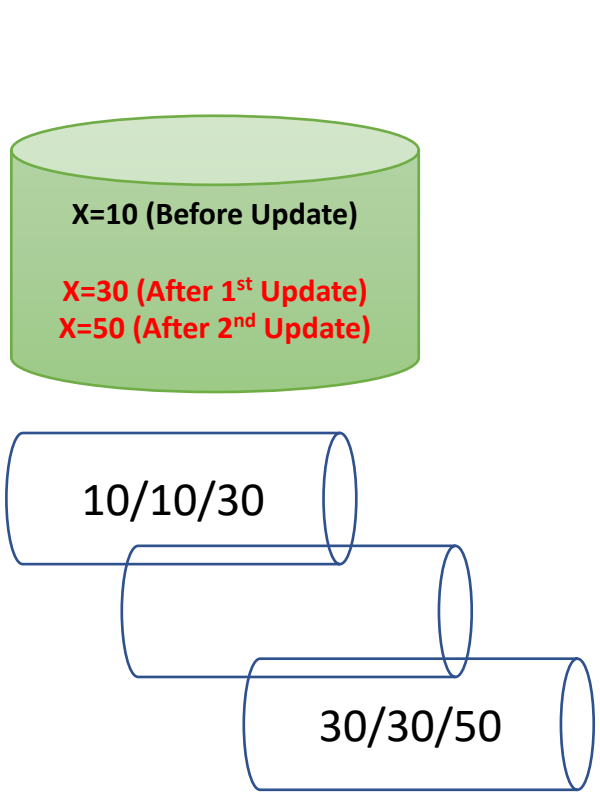
B. Uncommitted Data or Dirty Read

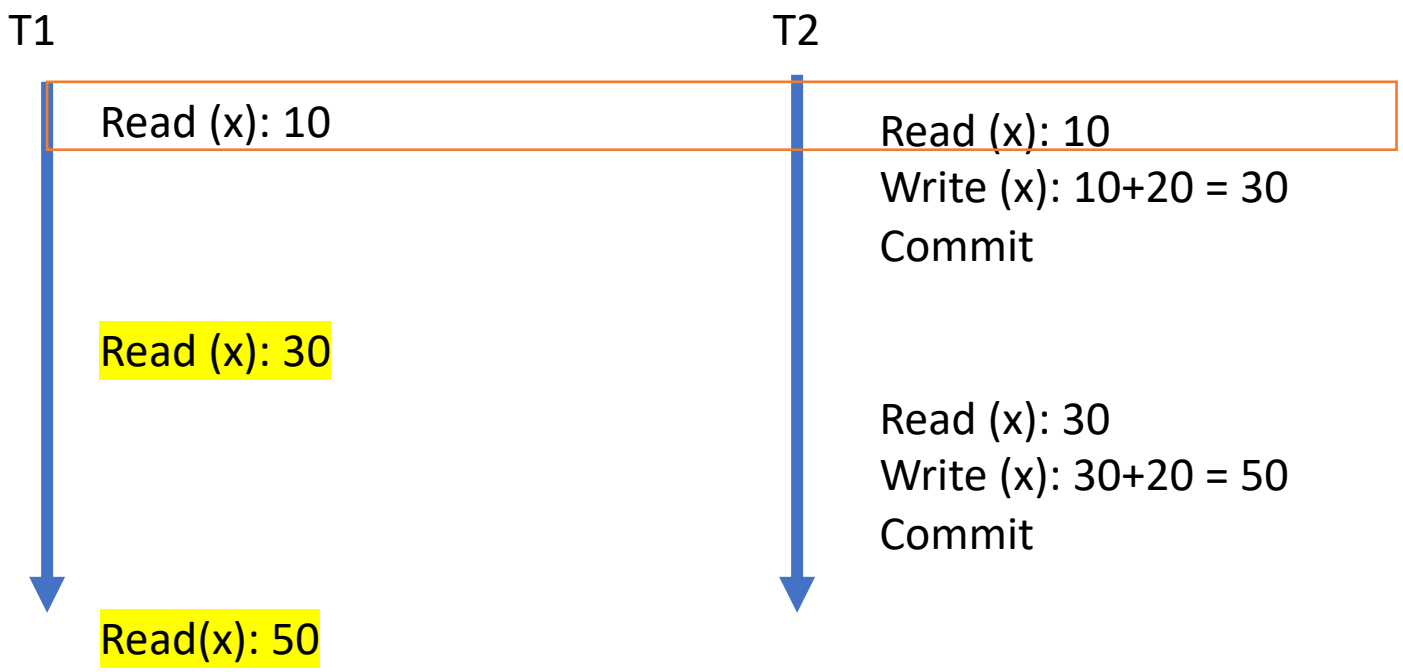
- A transaction should wait for other transaction to finish its operation or when writing on the same variable, there should be some concurrency control mechanism applied, such as Locking.



C. Inconsistent Retrieval

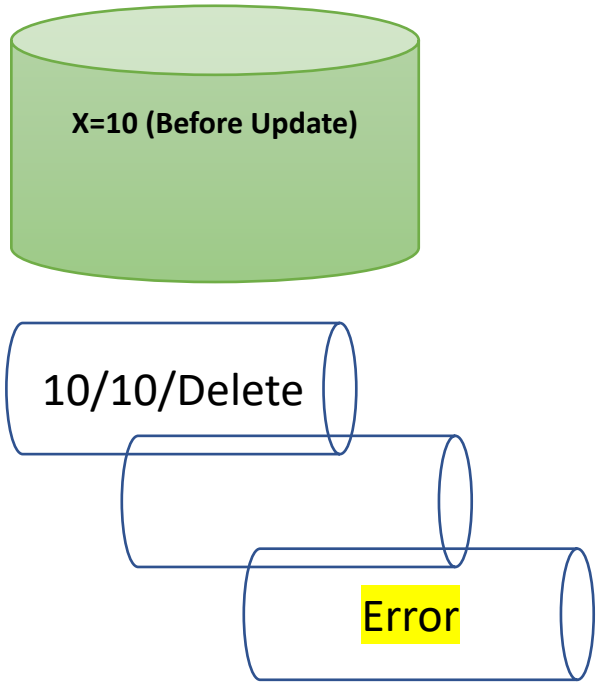
- One transaction reads the same variable multiple times, but due to operations performed on that variable by other transactions, the value changes in the consecutive read operation

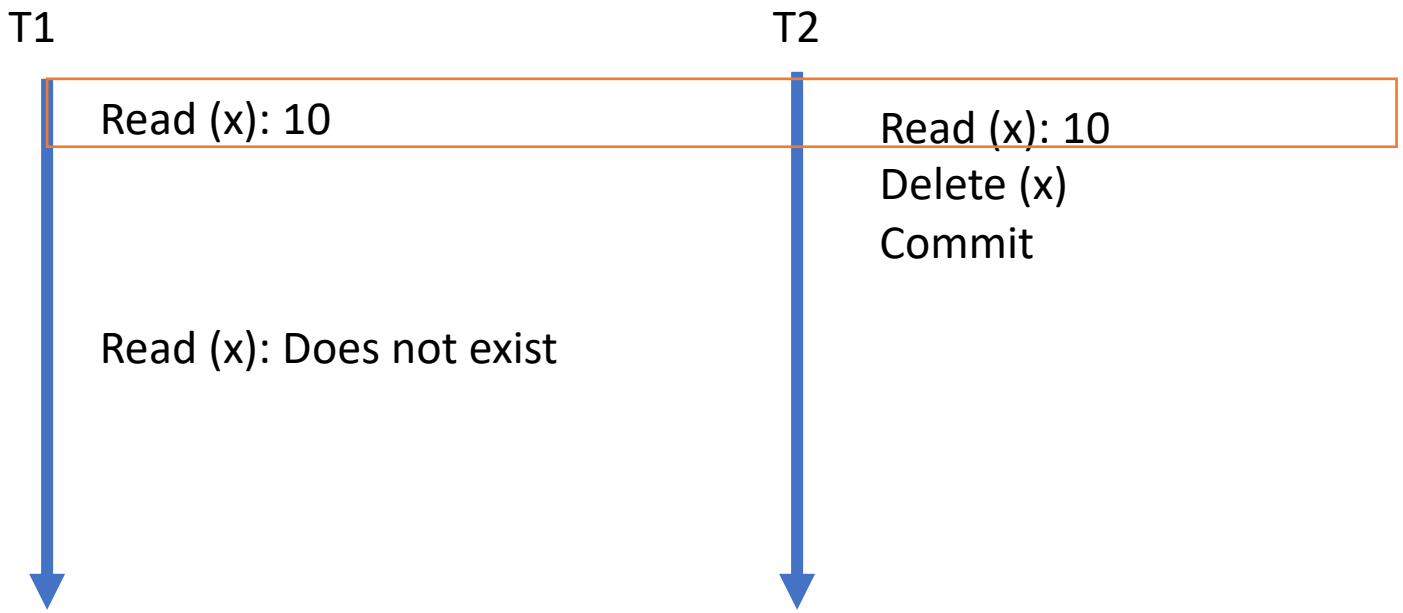




D. Phantom Read

- One transaction reads a database records multiple times, but due to operations performed on that variable or set of records by other transactions, new entries are found or the expected entries are lost





Scheduler

To determine the appropriate order, the scheduler bases its actions on concurrency control algorithms

1. Locking
2. Time-stamping methods
3. Optimistic

Recap

Set of operations in all transactions is **Schedule**
Process of concurrent execution of transactions is **Scheduling**
Serializability is ensured by **Scheduler**

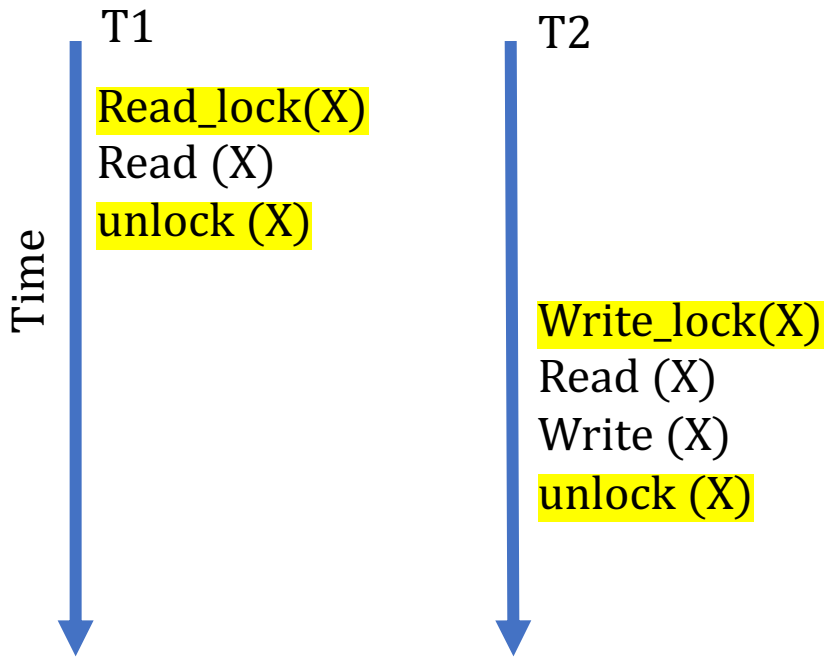
A. Concurrency Control with Locking

- Most common technique for concurrency control
- Locking methods facilitate isolation of data items used in concurrently executing transactions

Lock: guarantees exclusive use of a data item to a current transaction

Optimistic locking: Assumes no transactions conflict will occur.

Pessimistic locking: use of locks based on the assumption that conflict between transactions is likely



Locking Method

Lock manager: responsible for assigning and policing the locks used by the transactions

Lock Granularity: Indicates the level of lock use

- Database-level lock
- Table-level lock
- Page-level lock
- Row-level lock
- Field-level lock

Database-level Lock

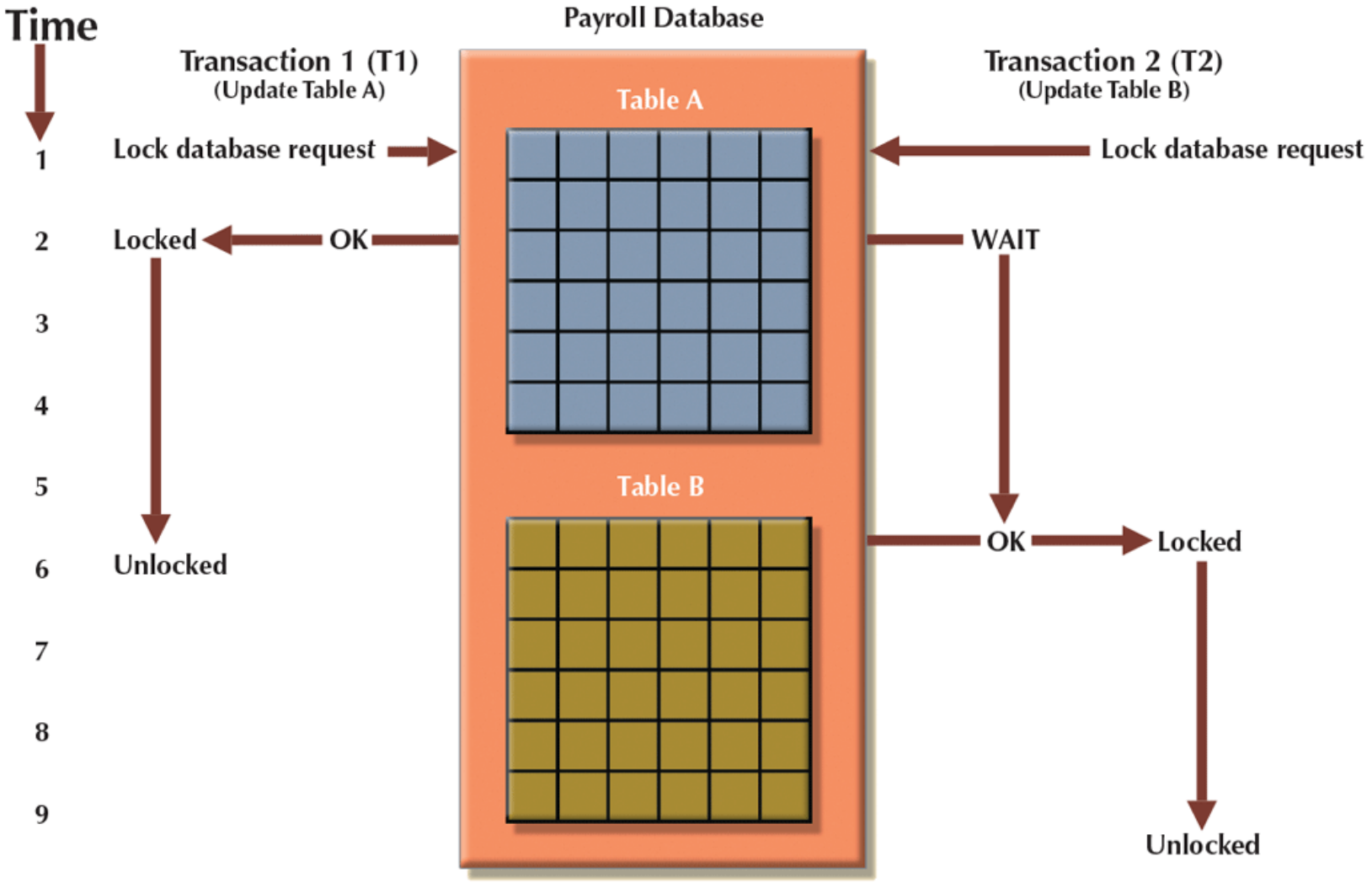
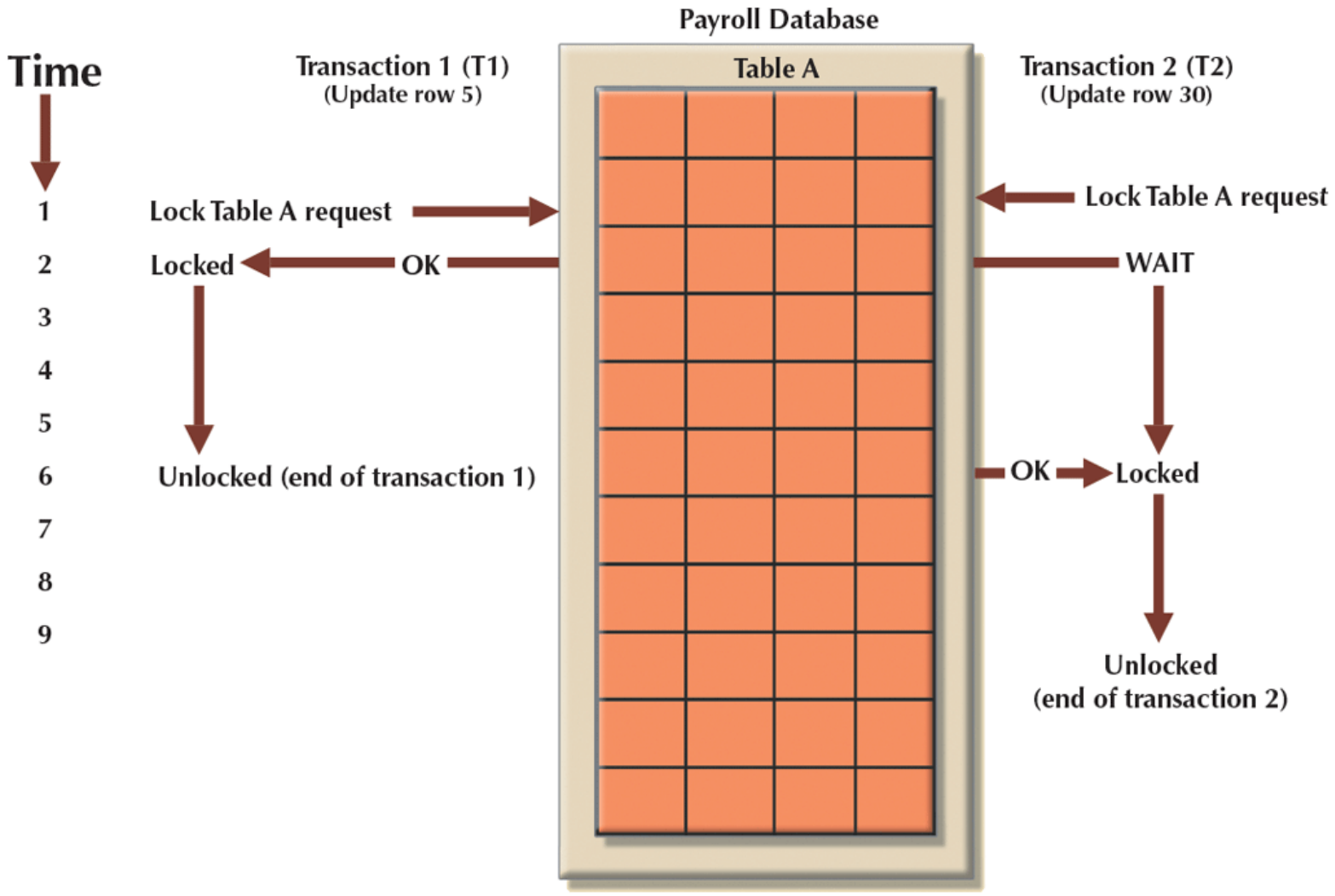
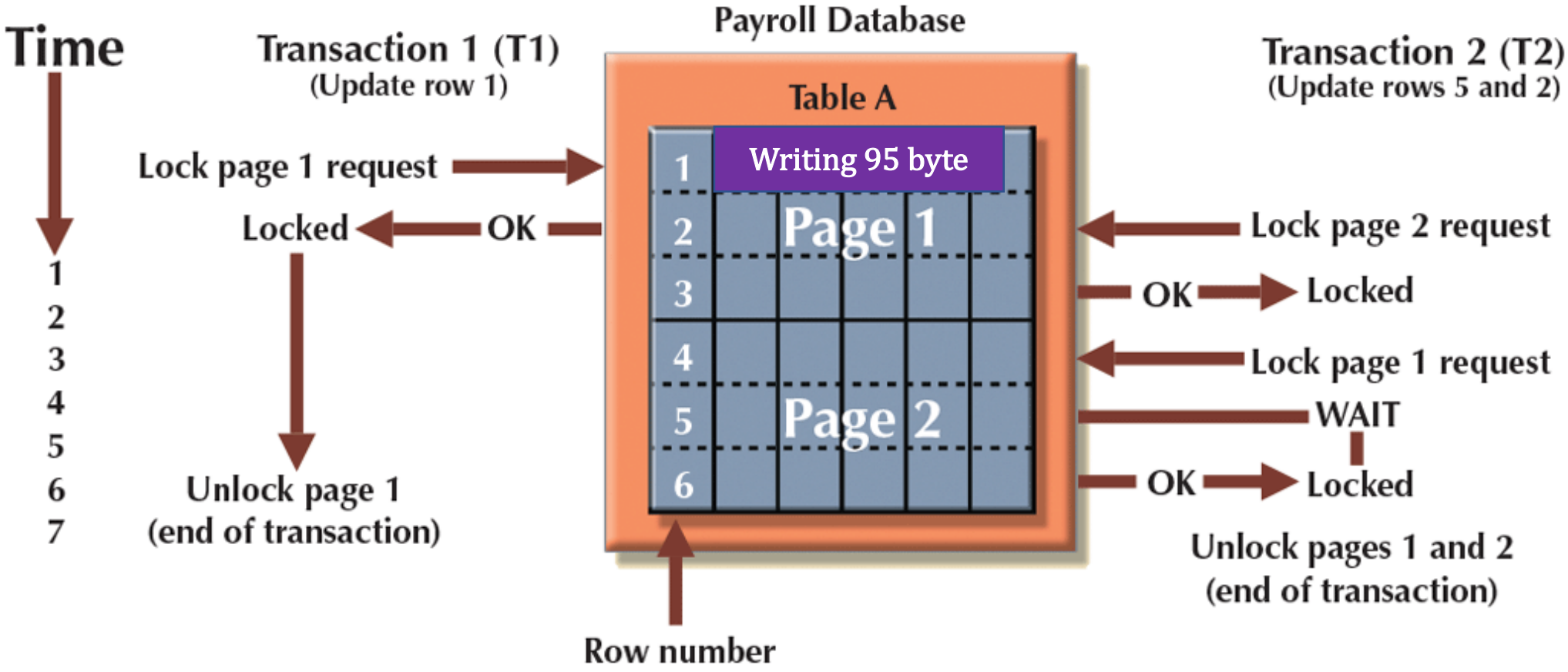


Table-level Lock

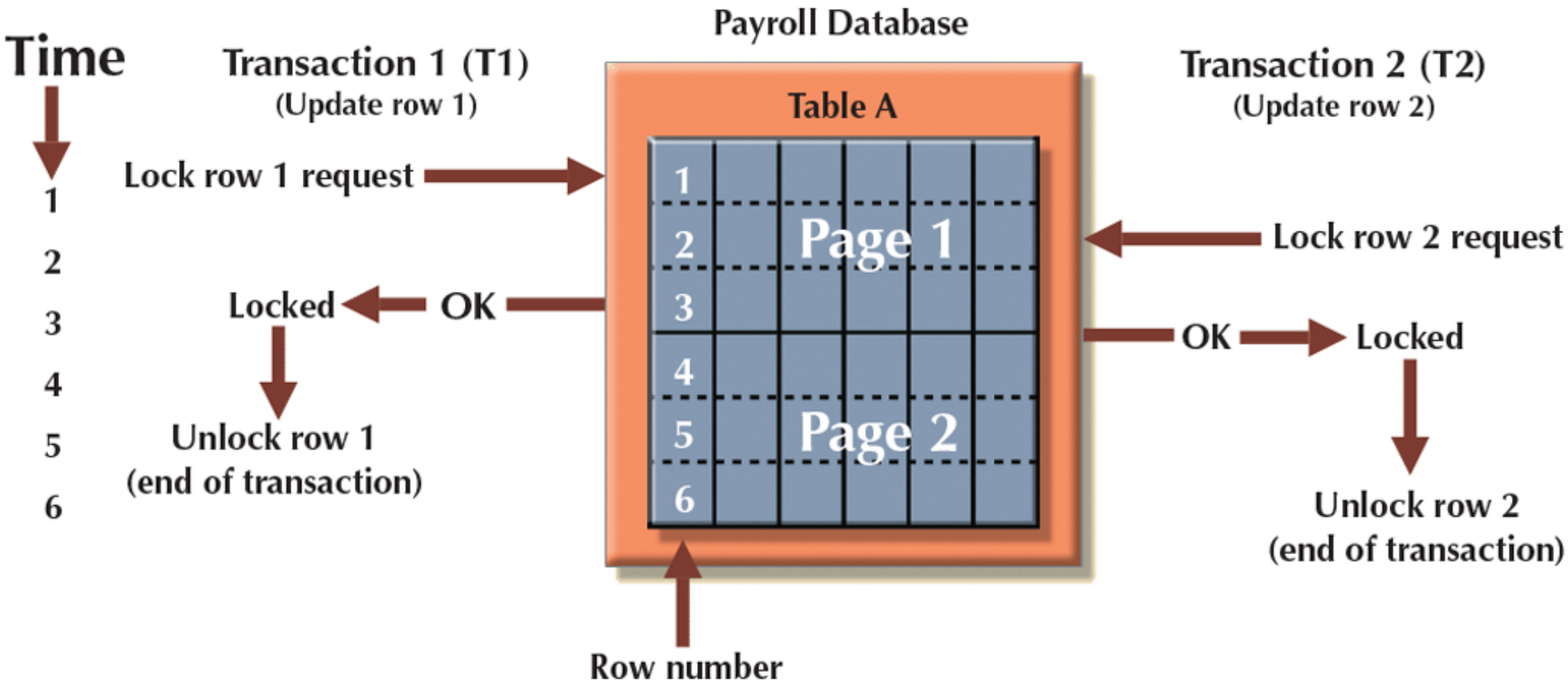


Page-level Lock



Page has fixed size, such as 4K, 8K, 16K

Row-level Lock



Field-level Lock

Allows concurrent transactions to access the same row as long as they require the use of different fields (attributes) within that row.

Requires an extremely high level of computer overhead and because the row-level lock is much more useful in practice.

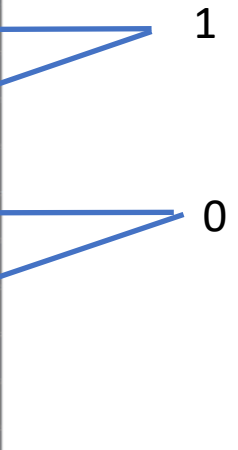
Lock Types (Binary, Exclusive/Shared)

Binary lock

- Two states: locked (1) and unlocked (0)
- If an object is locked by a transaction, no other transaction can use that object
- If an object is unlocked, any transaction can lock the object for its use

AN EXAMPLE OF A BINARY LOCK

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Lock PRODUCT	
2	T1	Read PROD_QOH	15
3	T1	PROD_QOH = 15 + 10	
4	T1	Write PROD_QOH	25
5	T1	Unlock PRODUCT	
6	T2	Lock PRODUCT	
7	T2	Read PROD_QOH	23
8	T2	PROD_QOH = 23 – 10	
9	T2	Write PROD_QOH	13
10	T2	Unlock PRODUCT	



Lock Types

Exclusive lock

- Access is reserved for the transaction that locked the object

Shared lock

- Concurrent transactions are granted read access on the basis of a common lock

READ/WRITE CONFLICT SCENARIOS: CONFLICTING DATABASE OPERATIONS MATRIX

	TRANSACTIONS		
	T1	T2	RESULT
Operations	Read	Read	No conflict
	Read	Write	Conflict
	Write	Read	Conflict
	Write	Write	Conflict

Problems using Locks

Resulting transaction schedule might not be serializable

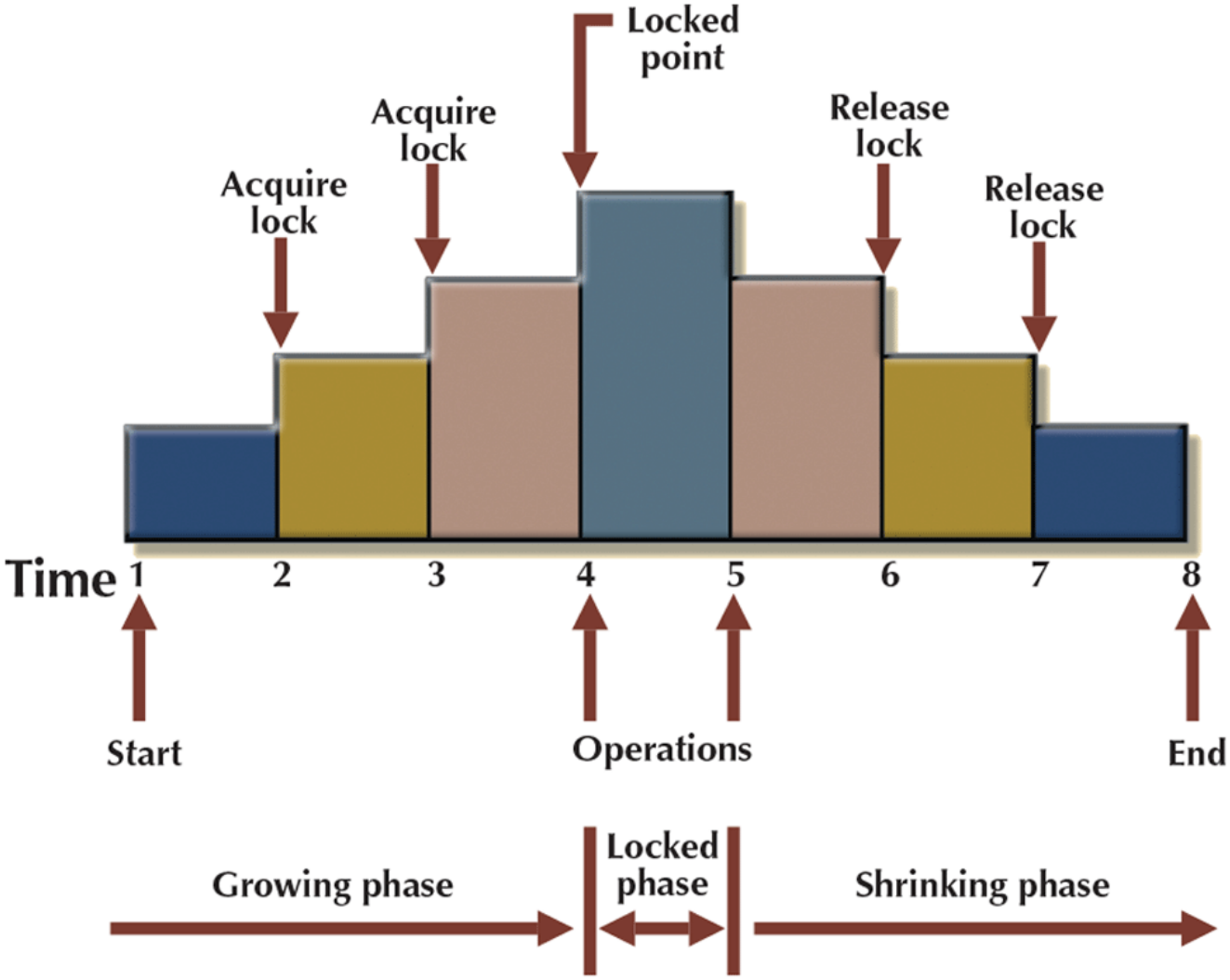
The schedule might create deadlocks

Two-Phase Locking to Ensure Serializability

Defines how transaction acquire and relinquish locks

Growing phase: Transaction acquires all required locks without unlocking any data

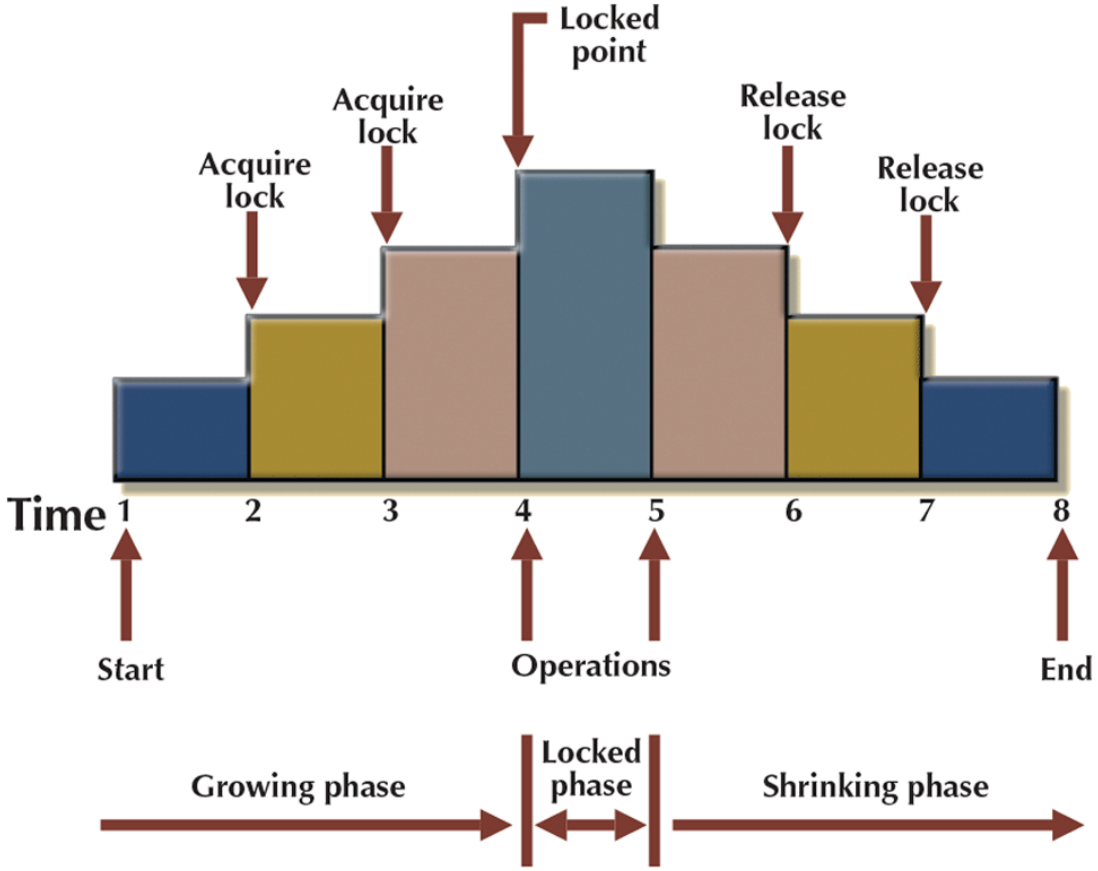
Shrinking phase: Transaction releases all locks and cannot obtain any new lock



Two-Phase Locking to Ensure Serializability

Governing rules

- Two transactions cannot have conflicting locks
- No unlock operation can precede a lock operation in the same transaction
- No data are affected until all locks are obtained



Deadlocks

Occur when two transactions wait indefinitely for each other to unlock data
Also known as deadly embrace

HOW A DEADLOCK CONDITION IS CREATED

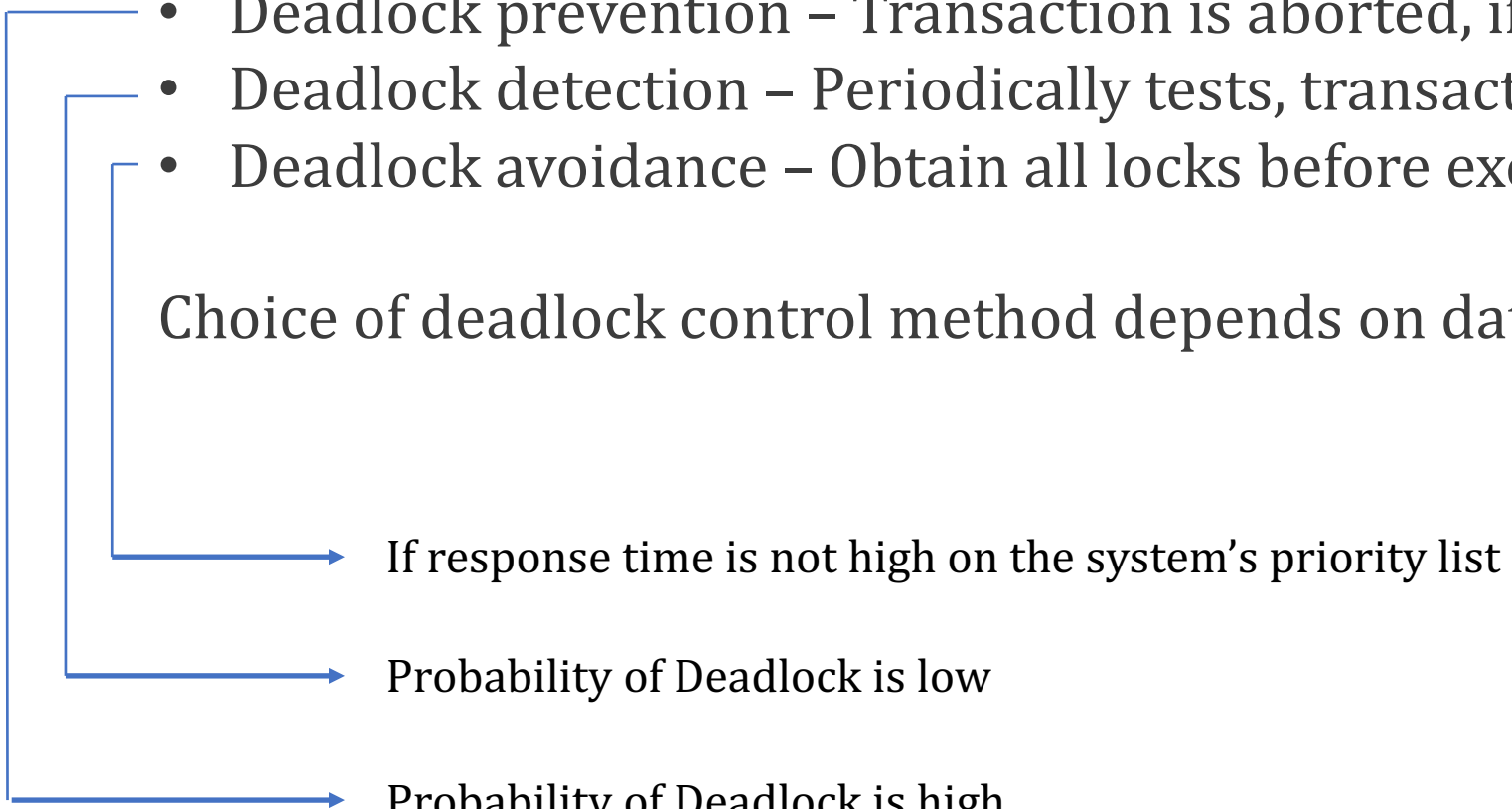
TIME	TRANSACTION	REPLY	LOCK STATUS	
			DATA X	DATA Y
0			Unlocked	Unlocked
1	T1:LOCK(X)	OK	Locked	Unlocked
2	T2:LOCK(Y)	OK	Locked	Locked
3	T1:LOCK(Y)	WAIT	Locked	Locked
4	T2:LOCK(X)	WAIT	Locked	Locked
5	T1:LOCK(Y)	WAIT	Locked	Locked
6	T2:LOCK(X)	WAIT	Locked	Locked
7	T1:LOCK(Y)	WAIT	Locked	Locked
8	T2:LOCK(X)	WAIT	Locked	Locked
9	T1:LOCK(Y)	WAIT	Locked	Locked
...	.....	.....	.....	.....
...	.....	.....	.....	.....
...	.....	.....	.....	.....
...	.....	.....	.....	.....

Deadlocks

Control techniques

- Deadlock prevention – Transaction is aborted, if requesting new lock
- Deadlock detection – Periodically tests, transaction is aborted if found
- Deadlock avoidance – Obtain all locks before execution

Choice of deadlock control method depends on database environment



B. Concurrency Control with Time-Stamping

Time stamping assigns global, unique time stamp to each transaction

- Produces explicit order in which transactions are submitted to DBMS

Properties

- Uniqueness: ensures no equal time stamp values exist
- Monotonicity: ensures time stamp values always increases

Time-Stamping Method Disadvantages

Disadvantages

- Each value stored in the database requires two additional stamp fields
- Increases memory needs
- Increases the database's processing overhead
- Demands a lot of system resources

Wait/Die and Wound/Wait Schemes

Wait/Die

A concurrency control scheme in which an older transaction must wait for the younger transaction to complete and release the locks before requesting the locks itself

- Otherwise, the newer transaction dies and is rescheduled

Wound/Wait

A concurrency control scheme in which an older transaction can request the lock, preempt the younger transaction, and reschedule it

- Otherwise, the newer transaction waits until the older transaction finishes

Wait/Die and Wound/Wait Schemes

WAIT/DIE AND WOUND/WAIT CONCURRENCY CONTROL SCHEMES

TRANSACTION REQUESTING LOCK	TRANSACTION OWNING LOCK	WAIT/DIE SCHEME	WOUND/WAIT SCHEME
T1 (11548789)	T2 (19562545)	• T1 waits until T2 is completed and T2 releases its locks.	• T1 preempts (rolls back) T2. • T2 is rescheduled using the same time stamp.
T2 (19562545)	T1 (11548789)	• T2 dies (rolls back). • T2 is rescheduled using the same time stamp.	• T2 waits until T1 is completed and T1 releases its locks.

C. Concurrency Control with Optimistic Method

Optimistic approach: based on the assumption that the majority of database operations do not conflict

- Does not require locking or time stamping techniques
- Transaction is executed without restrictions until it is committed

Phases of optimistic approach

Read
Validation
Write

C. Concurrency Control with Optimistic Method

Read phase

- Transaction:
  - Reads the database
  - Executes the needed computations
  - Makes the updates to a private copy of the database values

Validation phase

- Transaction is validated to ensure that the changes made will not affect the integrity and consistency of the database

Write phase

- Changes are permanently applied to the database

Database Recovery Management

Database recovery: restores database from a given state to a previously consistent state

Recovery transactions are based on the atomic transaction property

All portions of a transaction must be treated as a single logical unit of work

- If transaction operation cannot be completed:
  - Transaction must be aborted
  - Changes to database must be rolled back

Transaction Recovery

Concepts that affect the recovery process

Buffers

Temporary storage areas in a primary memory used to speed up disk operations

Checkpoints

Allows DBMS to write all its updated buffers in memory to disk

Transaction Recovery

Concepts that affect the recovery process

Write-ahead log protocol

Ensures that transaction logs are always written before the data are updated

Redundant transaction logs

Ensure that a physical disk failure will not impair the DBMS's ability to recover data

Transaction Recovery

Techniques used in transaction recovery procedures

Deferred-write technique or deferred update

Transaction operations do not immediately update the physical database
Only transaction log is updated

Write-through technique or immediate update

Database is immediately updated by transaction operations during transaction's execution

Summary

- A transaction is a sequence of database operations that access the database
- Transactions have four main properties: **a**tomicity, **c**onsistency, **i**solation, and **d**urability
- SQL provides support for transactions through the use of two statements: COMMIT, which saves changes to disk, and ROLLBACK, which restores the previous database state
- Concurrency control coordinates the simultaneous execution of transactions
- A lock guarantees unique access to a data item by a transaction
- **Serializability** of schedules is guaranteed through the use of two-phase locking

Summary

- Concurrency control with **time stamping methods** assigns a unique time stamp to each transaction and schedules the execution of conflicting transactions in time stamp order

- Concurrency control with **optimistic methods** assumes that the majority of database transactions do not conflict and that transactions are executed concurrently, using private, temporary copies of the data

- Database recovery restores the database from a given state to a previous consistent state