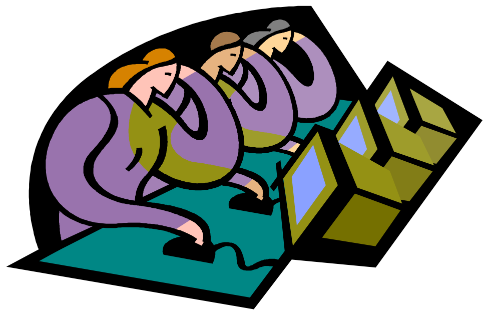
Concurrency Control

DR. MOHAMMED A. MOHAMMED

UNIVERSITY OF SULAIMANI | COLLEGE OF SCIENCE | COMPUTER DEPT.

Concurrency Control



Concurrent execution of user programs is essential for good DBMS performance.

- Since disk accesses are frequent and relatively slow, it is important to keep the **CPU** humming by working on several user programs concurrently.

Interleaving actions of different user programs can lead to inconsistency: e.g., check is cleared while account balance is being computed.

DBMS ensures such problems don’t arise: users can pretend they are using a **single-user** system.

Concurrency Control

The basic idea is :

1. Partial transaction should not be allowed.

2. A completed transaction should affect the result.

An example of concurrent use

- Consider a database that holds information about airline reservations.

- At any given instant, it is possible that several travel agents are looking up information about available seats on various flights and making new seat reservations.

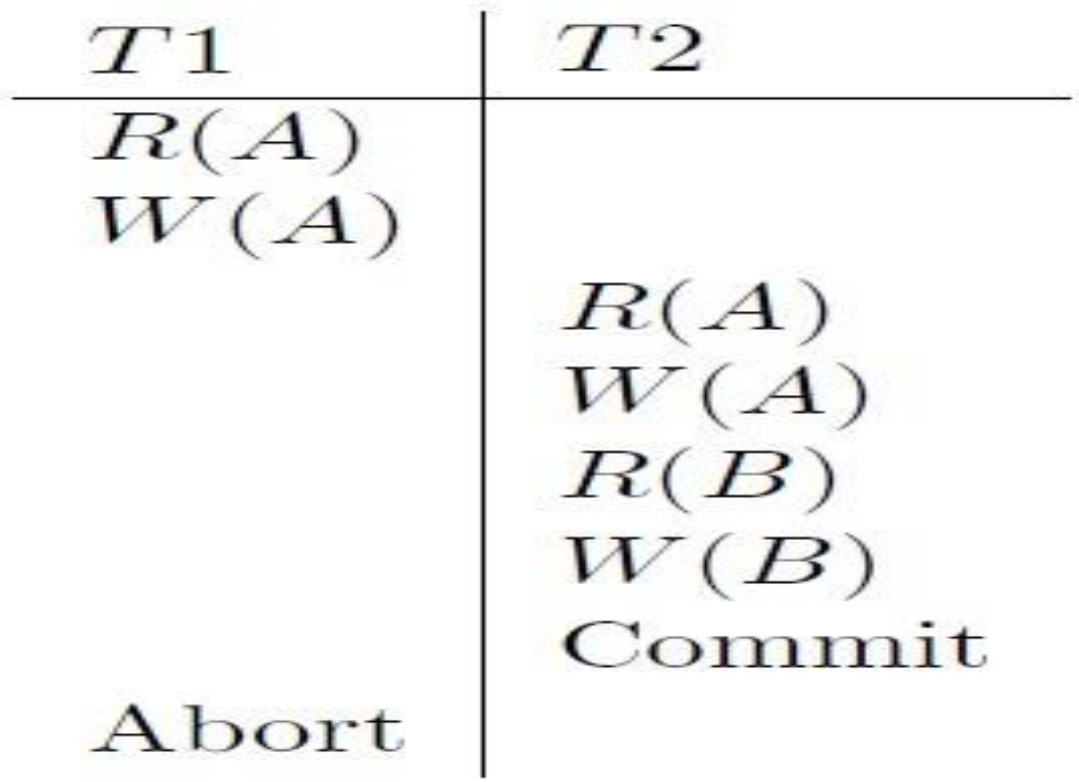
the DBMS must order their requests carefully to avoid conflicts.

Transaction: An Execution of a DB Program

Key concept is **transaction**, which is an **atomic** sequence of database actions (reads/writes).

Each transaction, executed completely, must leave the DB in a **consistent state** if DB is consistent when the transaction begins.

- Users can specify some simple **integrity constraints** on the data, and the DBMS will enforce these constraints.
- Beyond this, the DBMS does not really understand the semantics of the data. (e.g., it does not understand how the interest on a bank account is computed).
- Thus, ensuring that a transaction (run alone) preserves consistency is ultimately the user's responsibility!



$T1$	$T2$
$R(A)$	
$W(A)$	
	$R(A)$
	$W(A)$
	$R(B)$
	$W(B)$
	Commit
Abort	

Unrecoverable schedule (T1 not committed)

Locking Protocol

Locking Protocol: Is a set of rules to be followed by each **transaction** (and enforced by the **DBMS**), in order to ensure that even though actions of several transactions might be interleaved, the effect is like executing all transactions in some serial order.

Lock: Is a mechanism used to control access to database objects.

There are two locks which are supported commonly by DBMS:

1. Shared Lock.
2. Exclusive Lock.

Lock

Shared Lock: An object can be held by two different transaction at the same time.

Shared Lock allows transaction to **only read** the object.

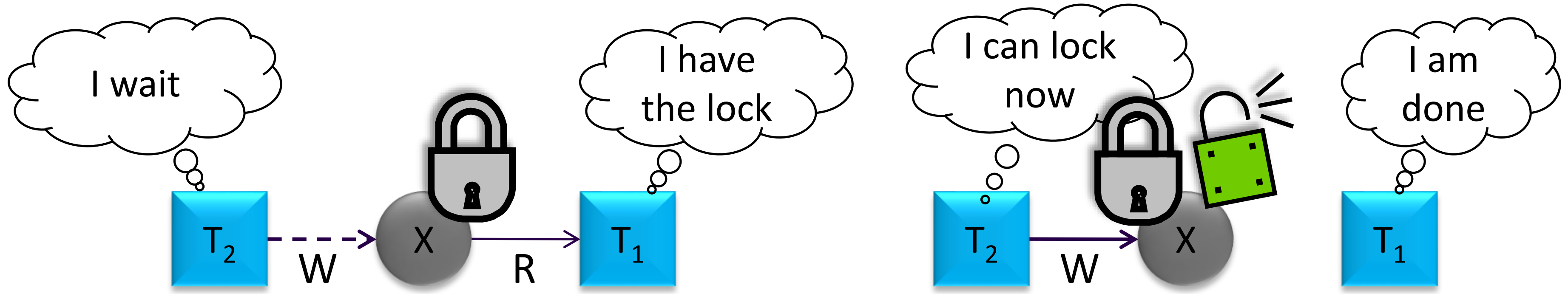
Exclusive Lock: An object ensures that no other transactions hold any lock on this object.

Exclusive Lock allows a transaction to **modify** the object.

Scheduling Concurrent Transactions

DBMS ensures that execution of {T1, ... , Tn} is equivalent to some *serial* execution T1, ... Tn.

- Before reading/writing an object, a transaction requests a lock on the object, and waits till the DBMS gives it the lock.



Two Phase Locking Protocol

2PL Locking Protocol: It defines the rules of how to acquire the locks on a data item and how to release the locks.

It assumes that a transaction can only be in one of two phases.

1. **Growing Phase**
2. **Shrinking Phase**

2PL Locking Protocol

Growing Phase:

- Can only acquire locks, but cannot release any lock.

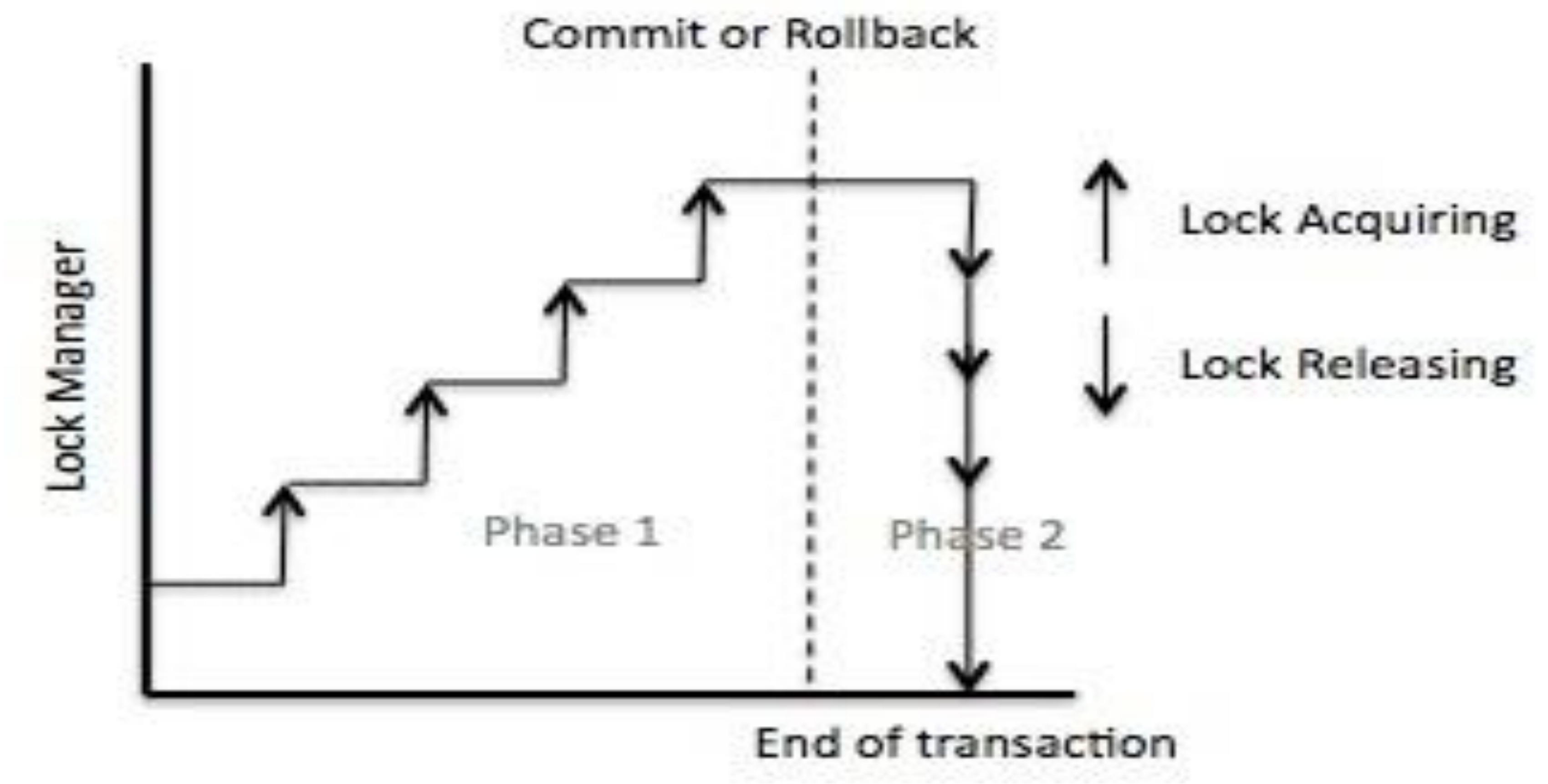
- Ultimately the transaction reaches a point where all the lock it may need has been acquired. This point is called **Lock Point**.

Shrinking Phase:

- Can only release locks, but cannot acquire any new lock.

- The transaction enters the shrinking phase as soon as it releases the first lock after crossing the **Lock Point**.

2PL Locking Protocol



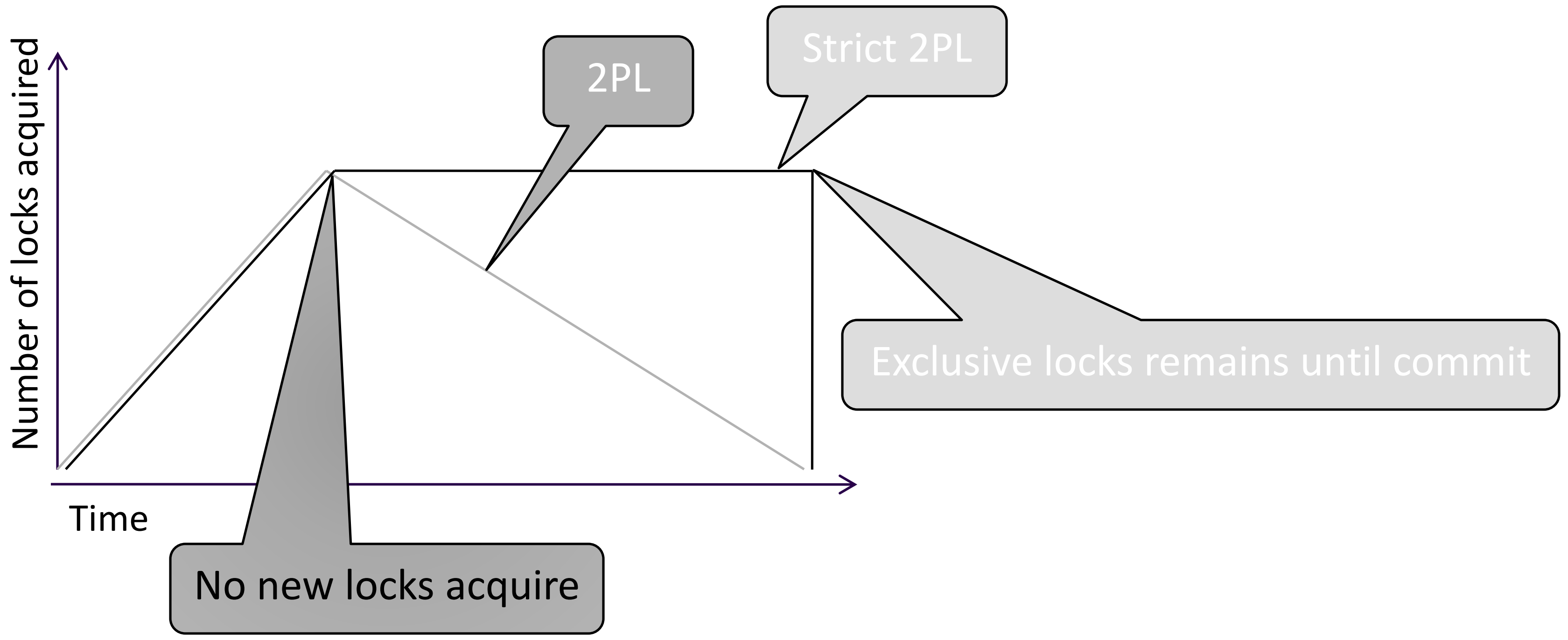
Strict 2PL Locking Protocol

It is a version of 2PL locking protocol.

In this protocol a transaction may release all the shared locks after the Lock Point has been reached, but it **cannot** release any of the **exclusive** locks until the transaction **commits**.

2PL Locking Protocol

2PL Locking Protocol is sufficient and more efficient



T1 acquires S lock on A

T1 acquires X lock on A

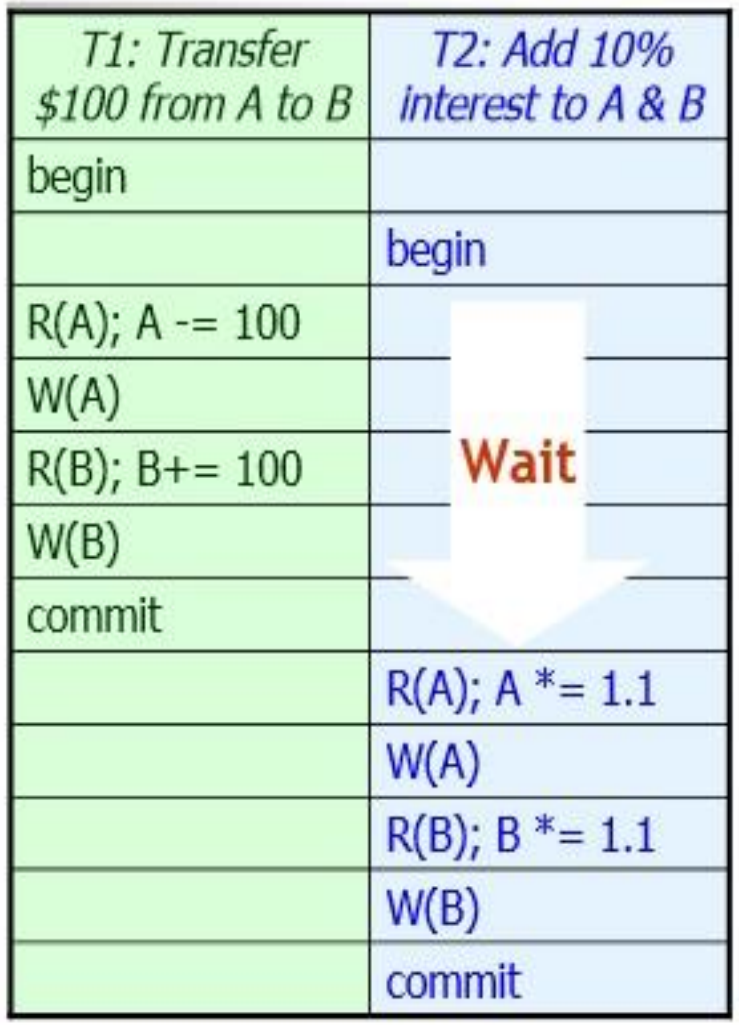
T1 acquires S lock on B

T1 acquires X lock on B

T1 releases all locks

T2 acquires S lock on A

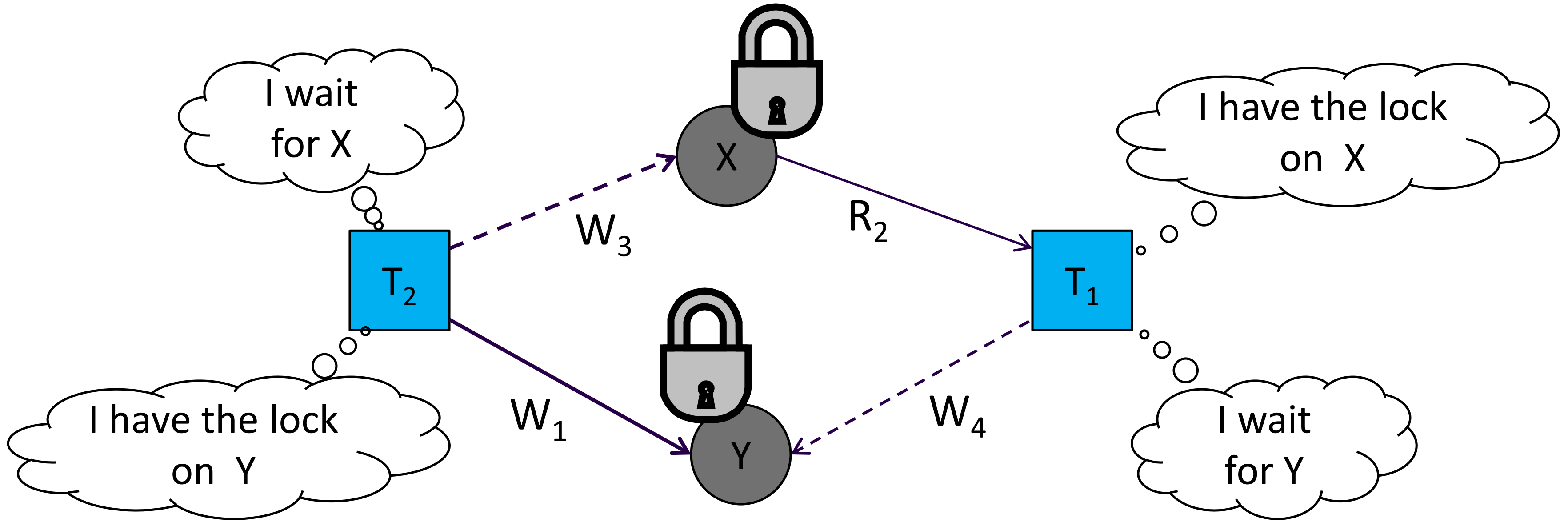
...



T1: Transfer $100 from A to B	*T2: Add 10% interest to A & B*
begin	
	begin
R(A); A -= 100	
W(A)	
R(B); B+= 100	Wait
W(B)	
commit	
	R(A); A *= 1.1
	W(A)
	R(B); B *= 1.1
	W(B)
	commit

Committed actions

Deadlock



A solution: T$_1$ or T$_2$ is aborted and restarted

ACID Properties

There are four important properties of transactions that a DBMS must ensure to maintain data in the face of concurrent access and system failures:

1. Atomicity
2. Consistency
3. Isolation
4. Durability

ACID Properties

A transaction is a collection of actions with the following **ACID** properties:

Atomicity: A transaction's changes to the state are atomic – either all happen or non happen.

Consistency: A transaction is a correct transformation of the state.

Isolation: Even though transaction execute concurrently, it appears to each transaction, T, that other executed either before or after T, but not both.

Durability: Once a transaction completes successfully, its effect should be persist even if the system crashes before all its changes are reflected on the disk.

Ensuring Atomicity

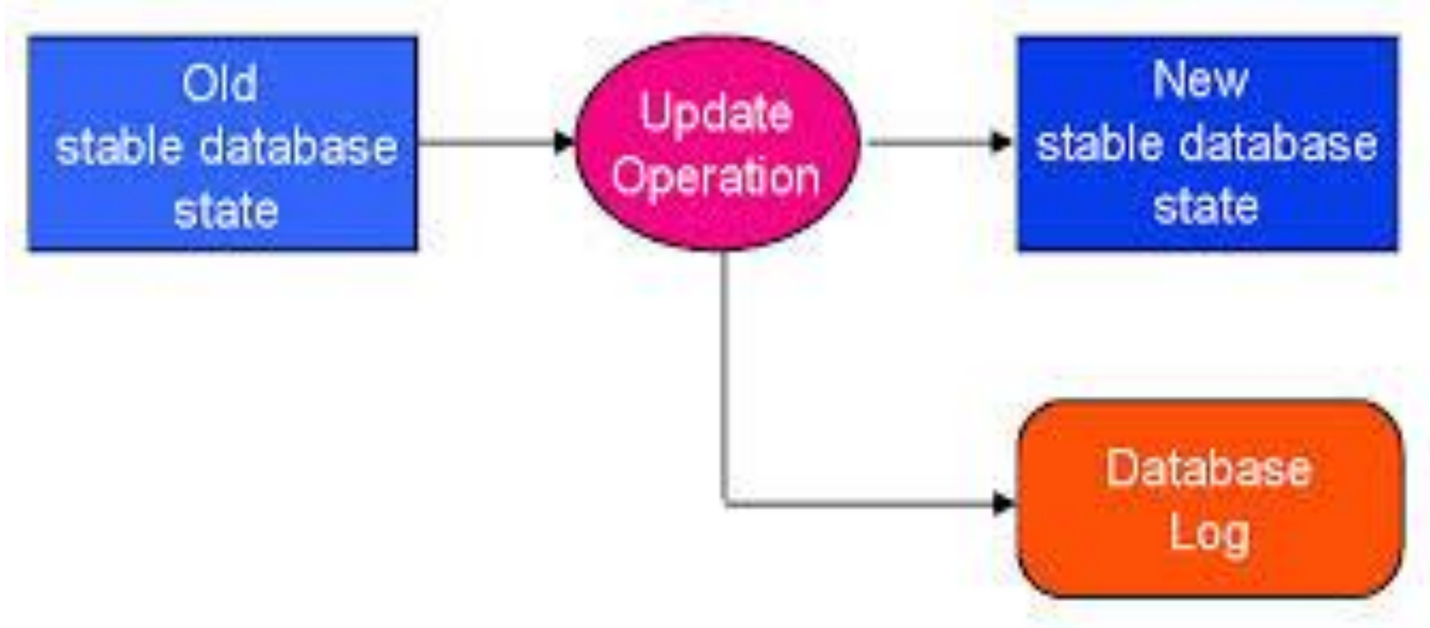
DBMS ensures *atomicity* (all-or-nothing property) even if system crashes in the middle of a Xact.

Idea: Keep a *log* (history) of all actions carried out by the DBMS while executing a set of Xacts:

- Before a change is made to the database, the corresponding log entry is forced to a safe location. (*WAL protocol*; OS support for this is often inadequate.)
- After a crash, the effects of partially executed transactions are *undone* using the log.

Ensuring Atomicity

Write-Ahead Logging (WAL): Is a set of techniques for providing atomicity and durability in database system.



The Log

The following actions are recorded in the log:

- *T_i writes an object*: the old value and the new value.
  - Log record must go to disk *before* the changed page!
- *T_i commits/aborts*: a log record indicating this action.

Log records chained together by Xact id, so it's easy to undo a specific Xact (e.g., to resolve a deadlock).

Log is often *duplexed* and *archived* on "stable" storage.

All log related activities (and in fact, all CC related activities such as lock/unlock, dealing with deadlocks etc.) are handled transparently by the DBMS.

Databases make these people happy

End users and DBMS vendors

DB application programmers

- e.g., smart webmasters

Database administrator (DBA)

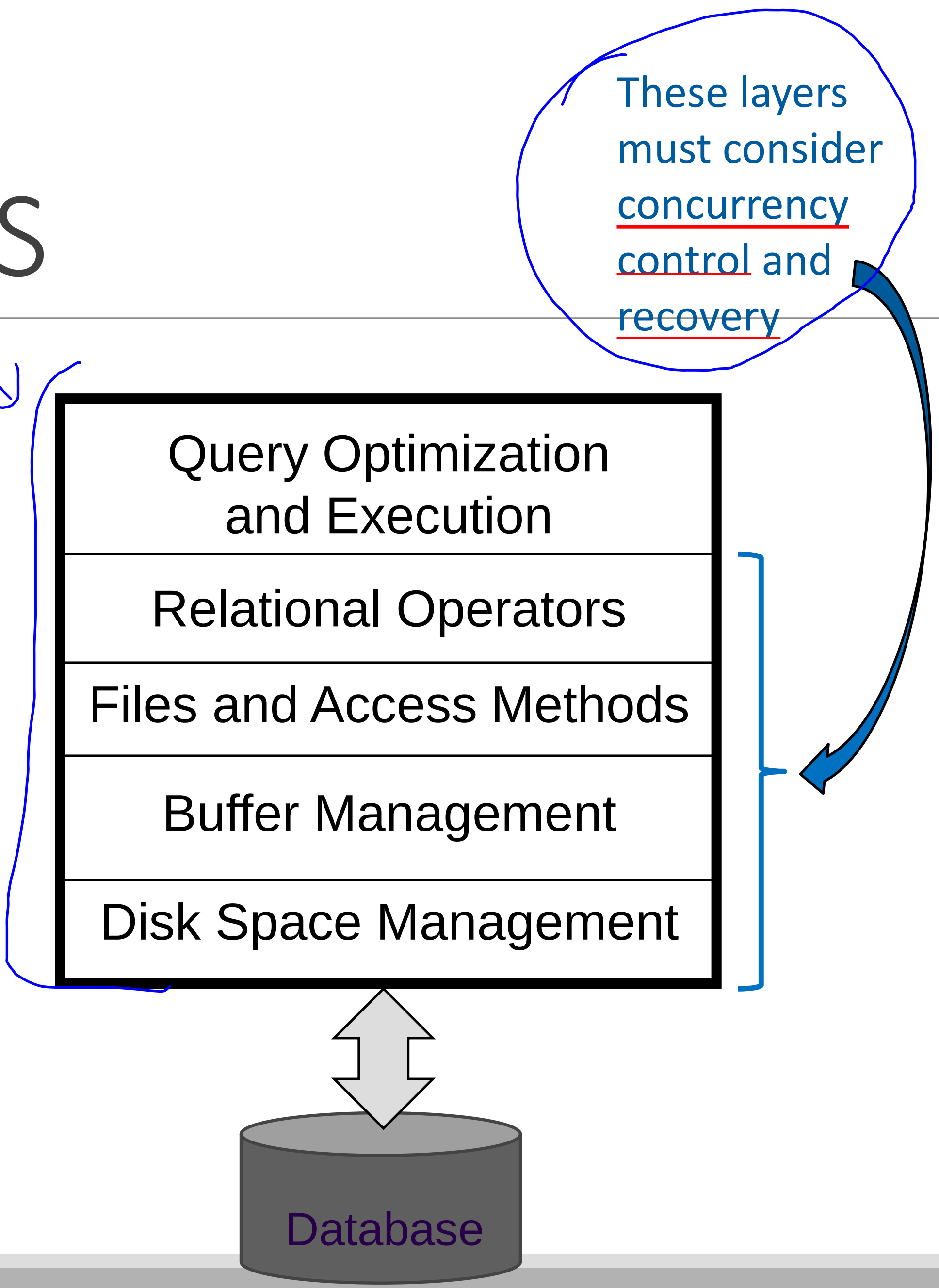
- Designs logical /physical schemas
- Handles security and authorization
- Data availability, crash recovery
- Database tuning as needs evolve

Structure of a DBMS

A typical DBMS has a layered architecture.

The figure does not show the concurrency control and recovery components.

This is one of several possible architectures; each system has its own variations.



These layers must consider concurrency control and recovery
Query Optimization and Execution
Relational Operators
Files and Access Methods
Buffer Management
Disk Space Management
Database

Summary

- How to manage several transaction at the same time.
- Locking protocol:
- Shared Lock.
- Exclusive Lock.
- Dead Lock.
- ACID Properties. (Atomicity, Consistency, Isolation and Durability)
- The Log and WAL protocol.