



TRANSACTION MANAGEMENT

CONCURRENCY CONTROL AND RECOVERY MECHANISMS

Presented to -

Mohammad Jahangir Alam

Assistant Professor,

Department of Computer Science and Engineering

Faculty of Science and Information Technology

Daffodil International University



Our Team

[In order of appearance]

Nur Sayda [232-15-437]

Samira Haque Vabna [232-15-764]

Supan Roy [232-15-716]

Abdullah Al Noman [232-15-797]

Shakira Rahman Simi [232-15-723]

Table Of Contents

What is a Transaction?

ACID Properties

Concurrent Transactions

Concurrency Control

Recovery Mechanisms

Transaction States

Best Practices in DBMS

What is a Transaction?

- A transaction is a logical unit of work in a database.
- It consists of one or more operations (like reading, writing, updating data).
- A transaction must be completed fully or not at all.
- Ensures data accuracy and integrity.



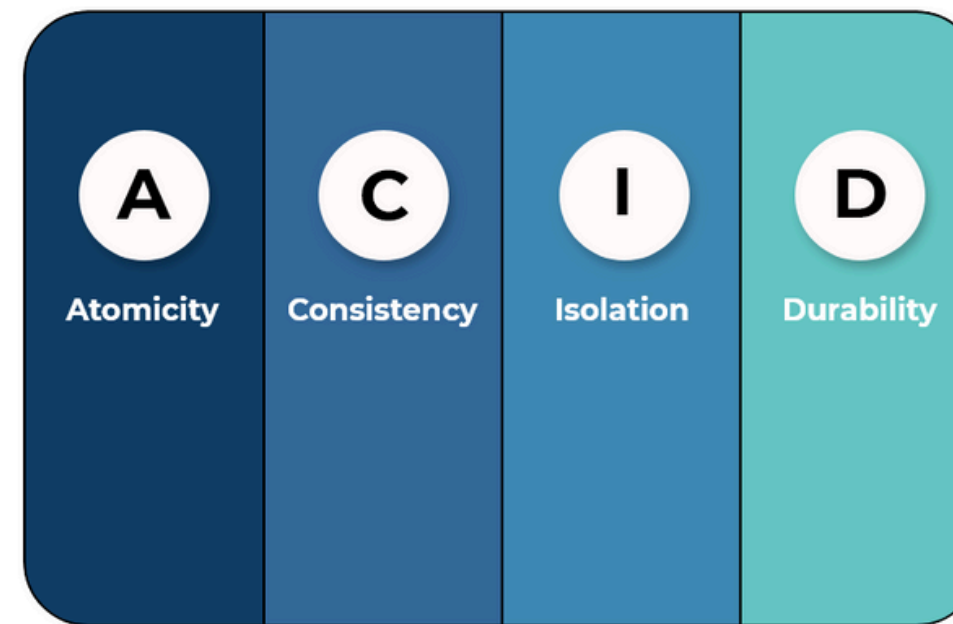
Example: Transferring ₺1000 from Account A to B:

1. Deduct ₺1000 from A

2. Add ₺1000 to B

(Both steps must succeed together)

ACID Properties of Transactions

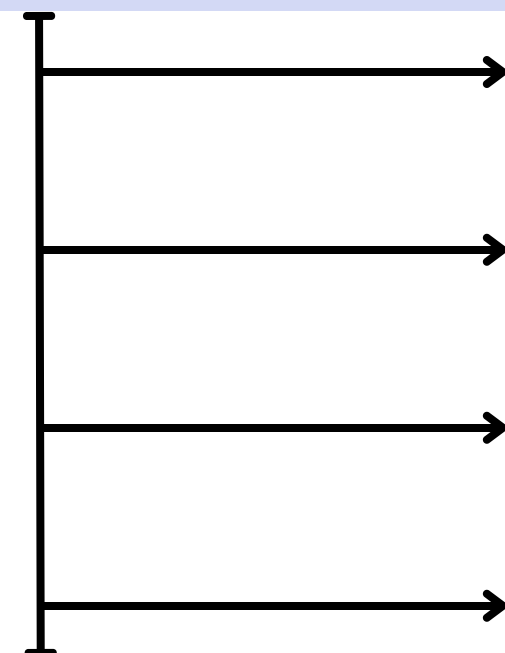


1. **Atomicity:** All operations of a transaction are completed, or none at all.
2. **Consistency:** Ensures the database moves from one valid state to another.
3. **Isolation:** Transactions are independent and do not interfere with each other.
4. **Durability:** Once a transaction is committed, changes are permanent—even after a crash.

The Problem – Concurrent Transactions

In real systems, multiple transactions run at the same time. This can lead to conflicts and inconsistencies in the database. Concurrency control is needed to manage this.

Common Problems from Concurrency

- 
- Dirty Read Problem
 - Unrepeatable Read Problem
 - Phantom Read Problem
 - Lost Update Problem

Concurrency Control – What & Why?

- ✚ Concurrency control manages multiple transactions executing at the same time.
- ✚ It ensures correctness, consistency, and isolation.
- ✚ Techniques Used:
 1. Lock-Based Protocols
 2. Timestamp Ordering
 3. Optimistic Concurrency Control



1. Lock-Based Concurrency Control

- Uses locks to control access to data items.
- Prevents multiple transactions from accessing the same data at the same time.

Types of Locks:

1. **Shared Lock (S)**: Allows multiple transactions to read.
2. **Exclusive Lock (X)**: Only one transaction can write (no one else can read or write).

	S	X
S	✓	X
X	X	X

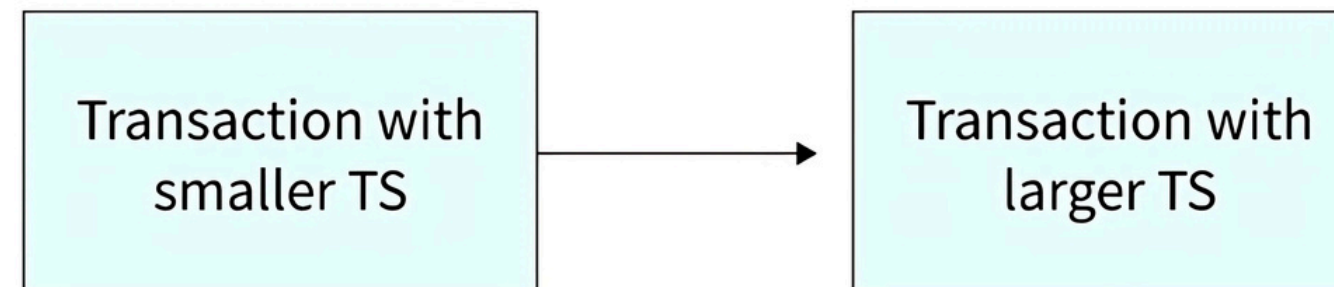
2. *Timestamp Ordering Protocol*

- Every transaction is given a unique timestamp when it starts.
- Prevents conflicts without using locks.

➤ For example:

If Transaction T1 enters the system first, it gets a timestamp $TS(T1) = 007$ (assumption).

If Transaction T2 enters after T1, it gets a timestamp $TS(T2) = 009$ (assumption).

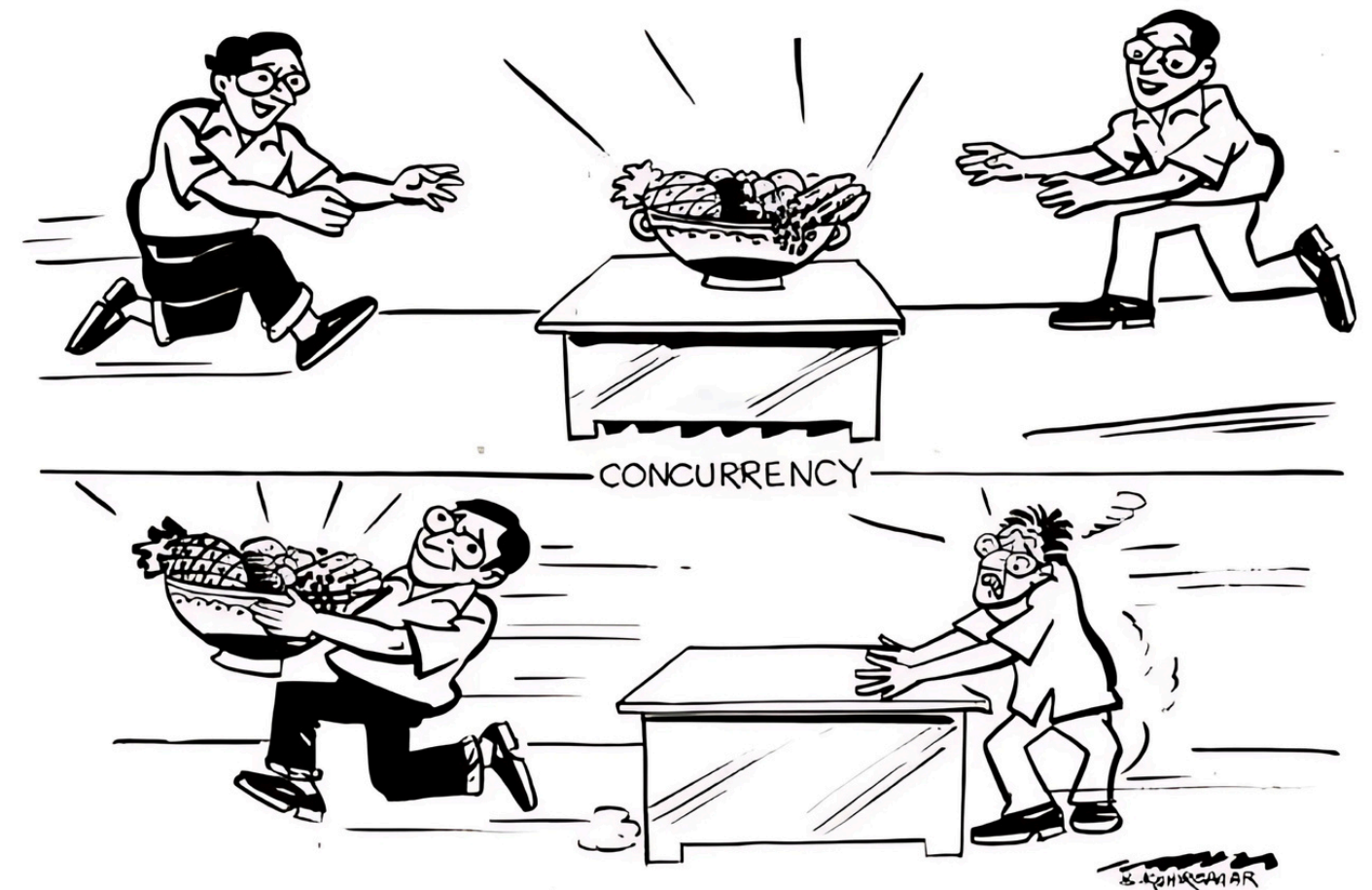


3. Optimistic Concurrency Control

- Assumes conflicts are rare — so no locks used initially.
- Transactions proceed without restrictions.

At commit time, system checks for conflicts:

? If no conflict → transaction is **committed**.
If conflict detected → transaction is **rolled back**.



What Happens if Something Fails?

Transactions may fail due to:

- System crashes
- Power failures
- Application or logic errors

These failures can leave the database in an inconsistent state.

We need recovery mechanisms to:

1. Undo uncommitted changes
2. Redo committed changes
3. Restore database consistency

Recovery Mechanisms: Log-Based Recovery

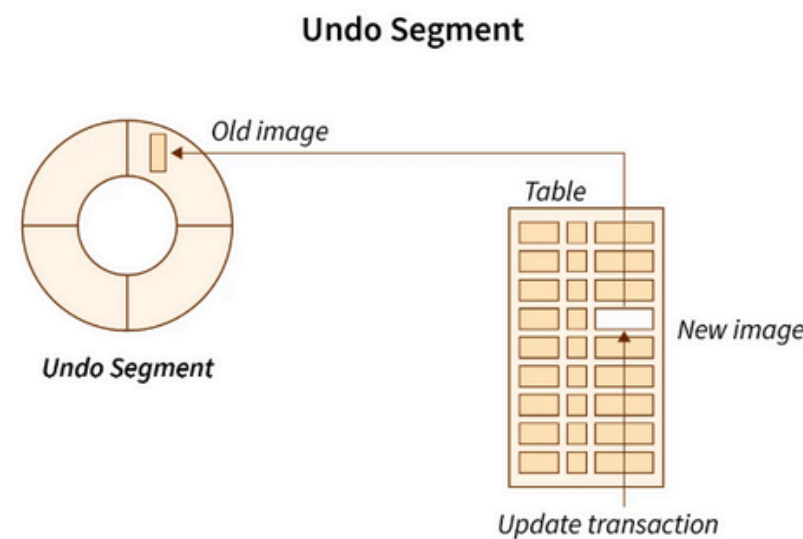
The **log** is a file that records every database operation performed by a transaction. Stored on disk, not in memory → survives crashes.

Every log entry includes:

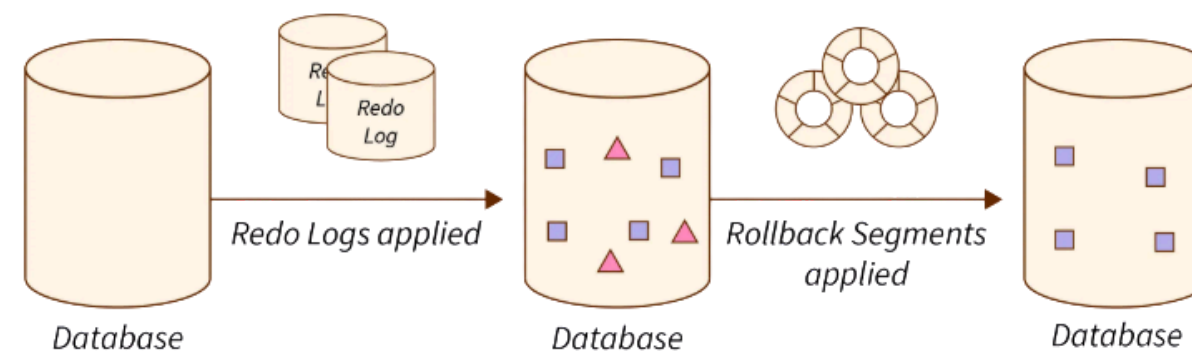
- Transaction ID
- Operation (e.g., write)
- Data item affected
- Old and new values

Recovery Mechanisms: Log-Based Recovery

1. Undo(T_i): Restores the old values of each of the items in T_i that have been updated by the transaction

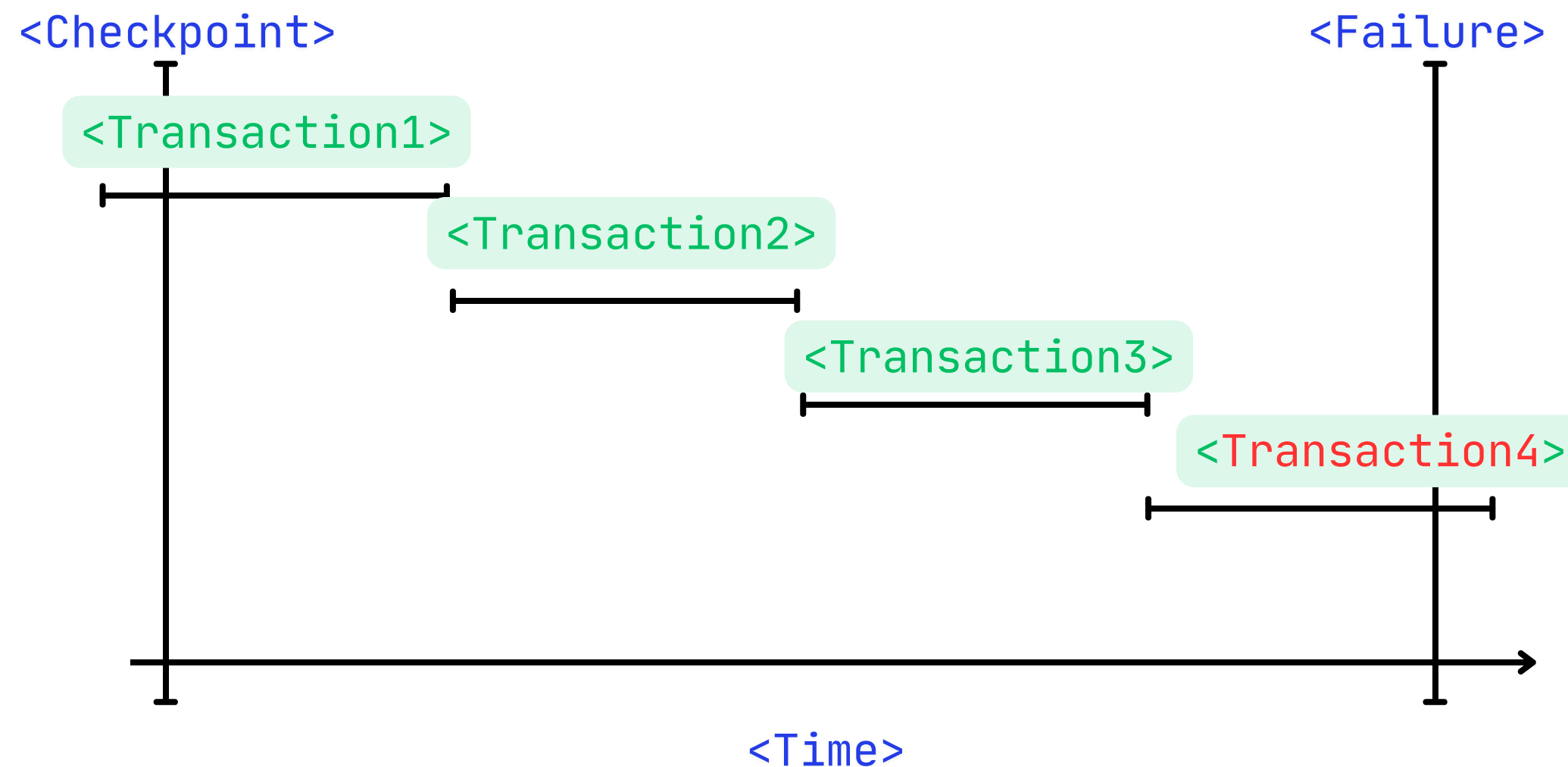


2. Redo(T_i): Updates all data items updated by transaction T_i with their new values. Undoing a T requires log to contain both the record <start> and the record <commit>.



Recovery Mechanisms: Checkpoints

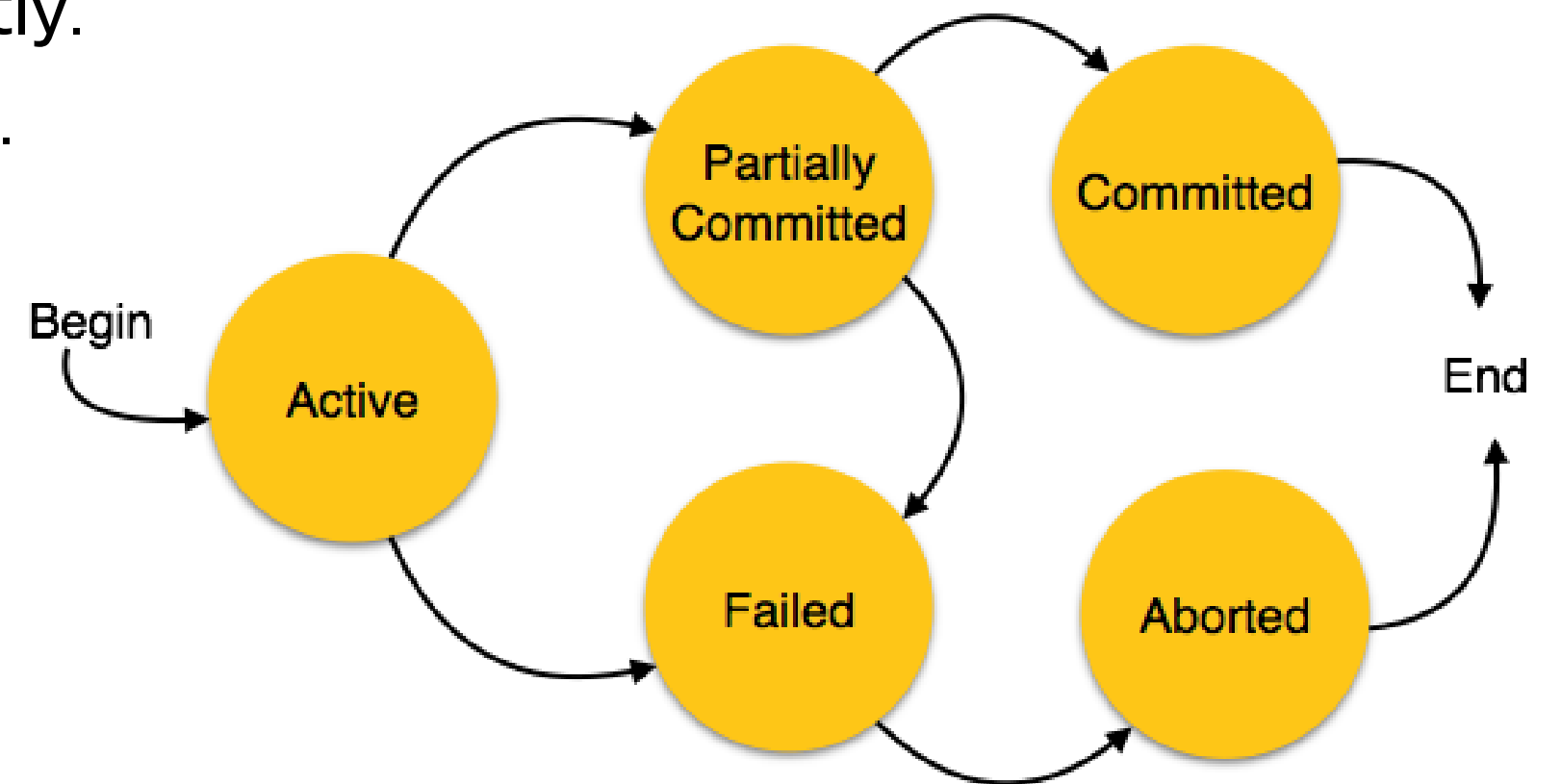
- A checkpoint is a snapshot of the database state at a particular time.
- It's written to disk periodically by the DBMS.



Transaction States

A transaction goes through the following states:

1. **Active** – Transaction is running.
2. **Partially Committed** – Final operation executed, but not yet saved to disk.
3. **Committed** – All changes are saved permanently.
4. **Failed** – Some error occurred during execution.
5. **Aborted** – Changes are rolled back (undone).



Best Practices in Transaction Management

- Always design transactions to be short and atomic.
- Use proper isolation levels to balance concurrency and consistency.
- Ensure logs and checkpoints are regularly maintained for recovery.
- Understand when to use locking vs optimistic methods.
- Monitor transaction failures to improve system reliability.



THANK YOU

If you have any questions, feel
free to ask now

