Database Systems 2

Lecture 16

Recovery

Overview

- Recovery
- Why is it necessary
- Poor Recovery Procedures
- States of a Transaction
- Operations in a Transaction
- System Log
- Transaction Recovery
  - Deferred Update
  - Immediate Update
  - Checkpoints
- Advanced Transaction Models

Transactions – Again

What are they?

"A collection of operations that performs a single logical function in a database application."

(Database System Concepts by Korth et al)

Transactions Outcomes – Again

Two outcomes – what are they?

Commit and Rollback

Recovery

"the act of **restoring a broken database** to a **usable state following an event** that has rendered it **inoperative** or **inconsistent**."

(Database Design and Management by Stamper & Price)

A DBMS must ensure either:

- All operations in a **transaction execute successfully**,

OR

- The transaction **has no effect** on the database.

Why do databases need to recover?

Computer systems fail because of?

- Software errors
- Hardware errors
- Communication errors
- Conflicting programs
- Malicious damage

A DBMS must survive a system failure, because

- It must not *lose* data
- It must be left in a *consistent state*

And how do we recover?

To perform recovery the DBMS must:

1. Recognise that the system is in an **inconsistent state,**
2. Determine the **most useful state**,
3. Transform the database **into the most useful state**.

The objective of the recovery procedure **is to**:

- **Minimise the loss of data**.
- **Return** the database as close as possible **to the state it was in before** the failure.

Operations in a Transaction

The DBMS must keep track of:

- **When** a transaction **begins.**
- The **read and write operations** which may have **to be undone, or redone.**
- **When** the transaction **ends**, **and whether** it was **successful** (committed) or **failed** (and will have to be rolled back).

System Log

The DBMS keeps a log of all changes to the database.

Oracle calls this a Redo Log.

Each transaction has a **unique identifier** (T) **which is recorded** in the log **alongside each operation**. For example:

Transaction	Timestamp	Operation	DataItem	OldValue	NewValue
T_{7641}	10:12.43	Start			
T_{7641}	10:13.23	Update	Staff SL12	5	10
T_{7641}	10:14.56	Delete	Staff SL09	3	
T_{7642}	10:15.11	Start			
T_{7641}	10:16.12	Commit			
T_{7642}	10:17.33	Insert	Staff SL22		22

Transaction Recovery

The system log is **used to <u>undo</u> the uncommitted changes**, and **<u>redo</u> the partially committed changes** that have been lost.

If a transaction fails, the system log can be **rolled back in reverse order**.

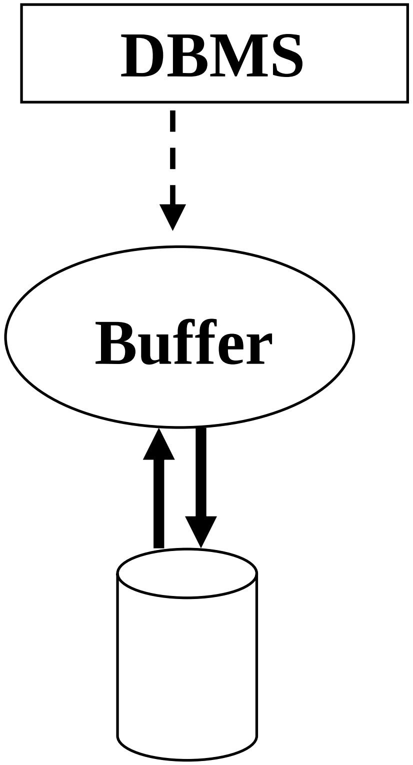
To give you an idea of the amount of data - **daily logging rates of 10 GB** are not uncommon.

As well as being **written to disk, part of the log fi[illegible] kept in memory for quick recovery** from minor fail[illegible]

Why?

There will be **multiple copies of the log file.**

Why is recovery necessary?



First we have to know how the DBMS saves changes.

When the user commits a transaction, the necessary changes are made to the data items.

In order for this to happen, the DBMS must:

- Read disk block into memory buffer
- Change contents of buffer.
- Write the memory buffer to the disk.
- BUT

What if the DBMS fails before the buffer is written to the disk?

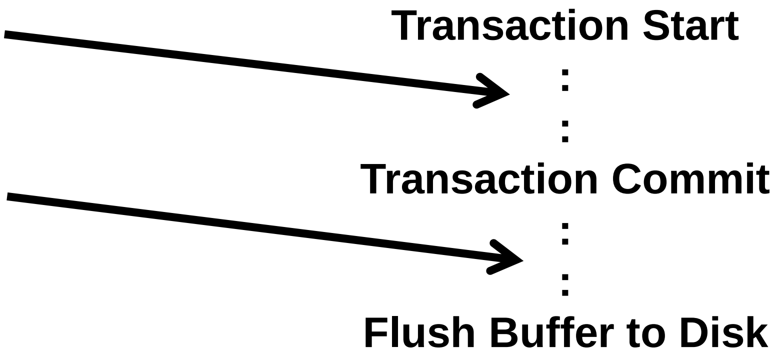
Recovery Procedures

If there is a system failure here:

- **Undo** the transaction.

If there is a system failure here:

- **Redo** the transaction



Transaction Start

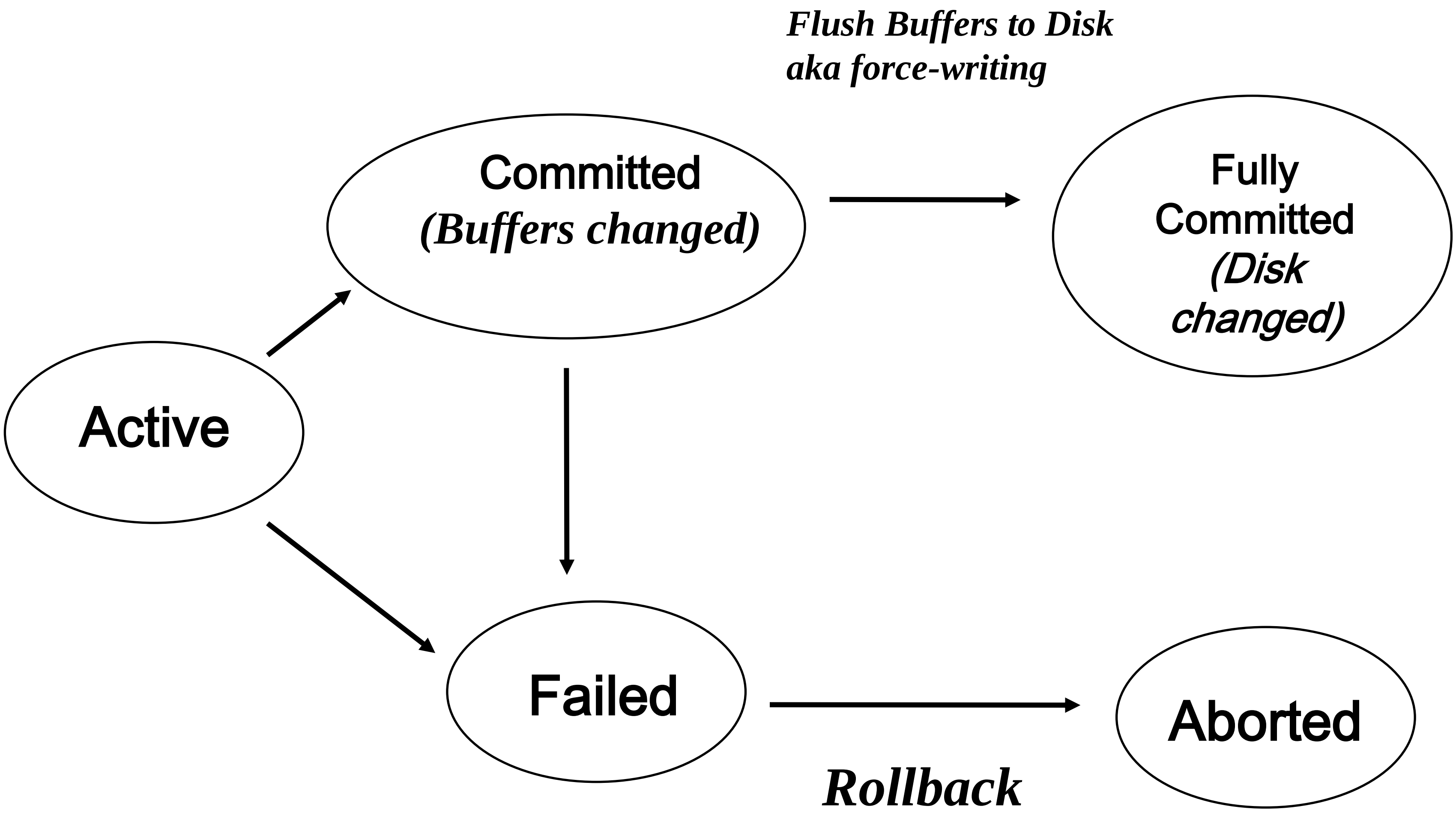
Transaction Commit

Flush Buffer to Disk

To do be able to do this, the database must record information about:

- which transactions had reached the commit point and which hadn't.
- the changes each transaction made to the data.

States of a Transaction



When should the buffer be written to disk?

The obvious time to flush the buffer to disk is:

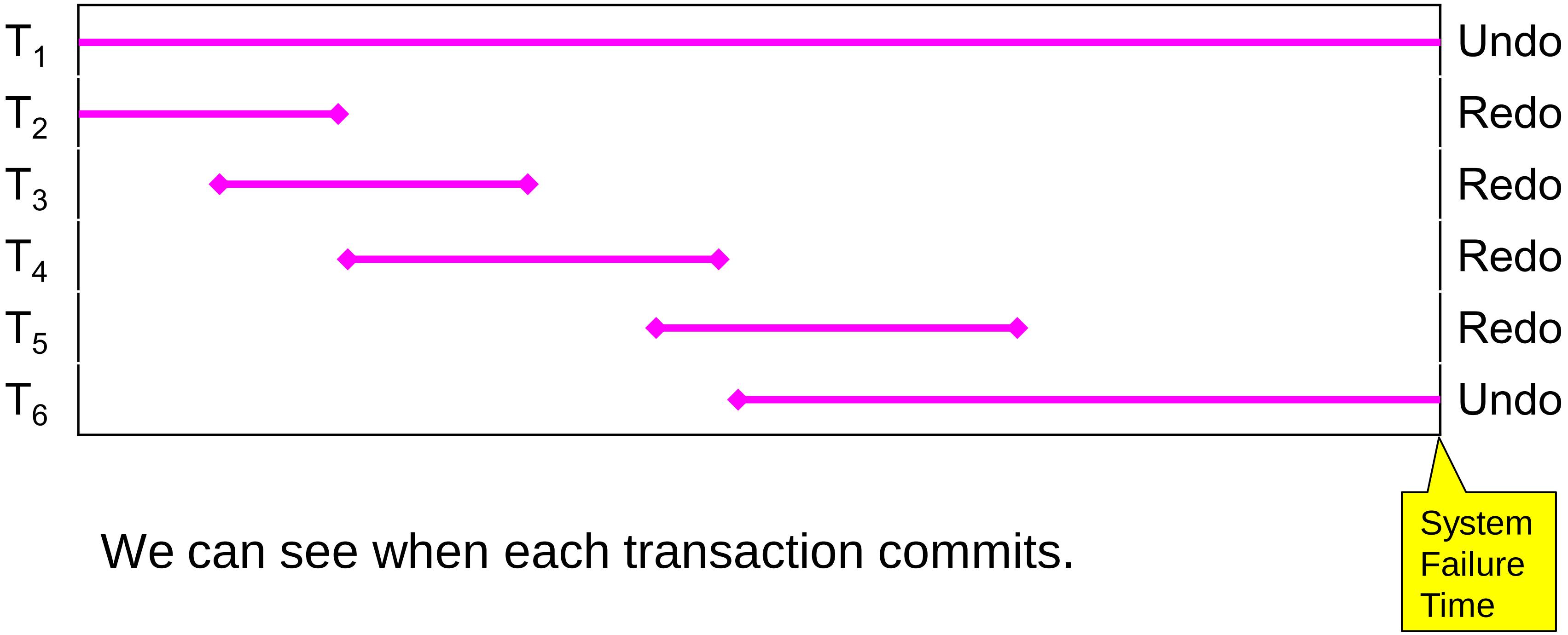
- When each transaction commits.

Why might the administrator configure the DBMS to do it at a different time? e.g. when the buffers are full.

- Because disk access is relatively slow. If the DBMS is handling a lot of transactions at once, writing to disk after each commit would slow the system down.

Assume that a DBMS would not flush the buffer before the transaction has committed.

Simple Example



We can see when each transaction commits.

We cannot be sure when the buffer was flushed.

So we cannot be sure that the changes were written to disk before the system failed.

Checkpoints

When a **transaction finished, all the changes may not be written to disk** immediately.

The **DBMS may only update** the disk **at fixed periods**, *eg when the buffers are filled.*

This is OK **provided the system log has been written to disk first**, as it can be **used to recover any lost data**. **However, the system log may become very large.**

Also it **may be time consuming to work out how far back** in the log **to search**, and **how many transactions have been safely written** to disk.

So to save time we create regular checkpoints.

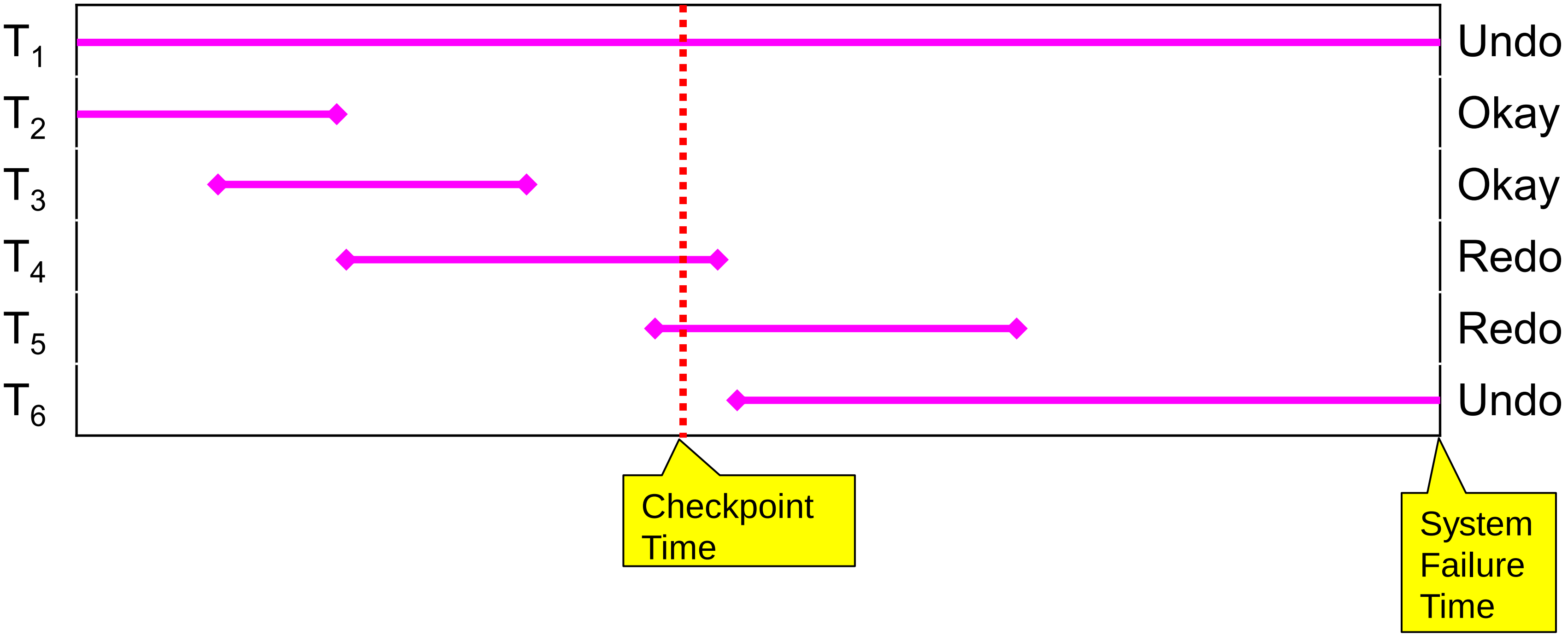
A checkpoint is a point of synchronisation between the database and the transaction log file. All buffers are force written to disk.

Recording a Checkpoint

Checkpoints are scheduled at **predetermined intervals**, and **involve the following**:

- **Write all system log entries to disk**
- **Force write the modified blocks** in the database buffers **to disk.**
- **Write a checkpoint entry in the log file.** This **contains identifiers of all transactions that are active** at the time of the checkpoint.

Example with Checkpoint



We now know that T2 and T3 have been written to disk, and so there is no need to redo these transactions during system recovery.

Recovery Techniques

In the event of a major system failure:

- Restore the last full backup copy of the database.
- Reapply to the disk the write operations of all transactions that have a 'transaction commit' entry in the log.

If the database has only become inconsistent:

- Undo the operations that caused the inconsistency.
- It may also be necessary to redo some transactions to ensure that their updates were written to disk.

Advanced Transaction Models

These methods are suitable for traditional business transactions, which are characterised by:

- Simple data types - integers, text strings, dates
- Short duration of transactions - minutes or seconds.

Databases are now being used for things like **CAD** and Games design:

- **Very large complex data items,** consisting of millions of interdependent parts.
- **Data which evolves** through time over long periods, requiring propagation of changes through rest of dependent parts.
- **Many hundreds of people may be working in parallel** on multiple versions of the data.

Advanced Transaction Models

There are other methods of handling transactions which have been developed as a result of this:

- Nested Transaction Model
- Sagas
- Multi-level Transaction Model
- Dynamic Restructuring
- Workflow Models

Database Systems: A Practical Approach to Design, Implementation and Management
Connolly and Begg Addison-Wesley, 1998

Database Transaction Models for Advanced Applications
Ahmed K. Elmagarmid Morgan Kaufmann, 1992
Available on Google books: