

Database Management System (DBMS)

Applications

Web
Forms

Embedded
SQL

Interactive
SQL

SQL Commands

DBMS

Query
Evaluation Engine

Concurrency
Control

Files and Access Methods

Buffer Manager

Disk Space Manager

Recovery
Manager

Database

Data

Indexes

Catalog

ACID Properties

Property	Dealt with by
A, D	Recovery Techniques
I	Concurrency Control Techniques
C	Integrity Constraints

Types of Failures

- Transaction failure:

- Incorrect input, deadlock, etc.

- System failure:

- Addressing error, application error, operating system fault, RAM failure, etc.

- Media failure:

- Disk head crash, power disruption, etc.

Purpose of Database Recovery

- ❑ To bring the database into the **last consistent state**, which existed prior to the failure
- ❑ To preserve transaction properties:

- **Atomicity & Durability**



- ❑ Example: System crashes before a fund transfer transaction completes its execution
 - The database must be restored to the state before the transaction modified any of the accounts.

Purpose of Database Recovery

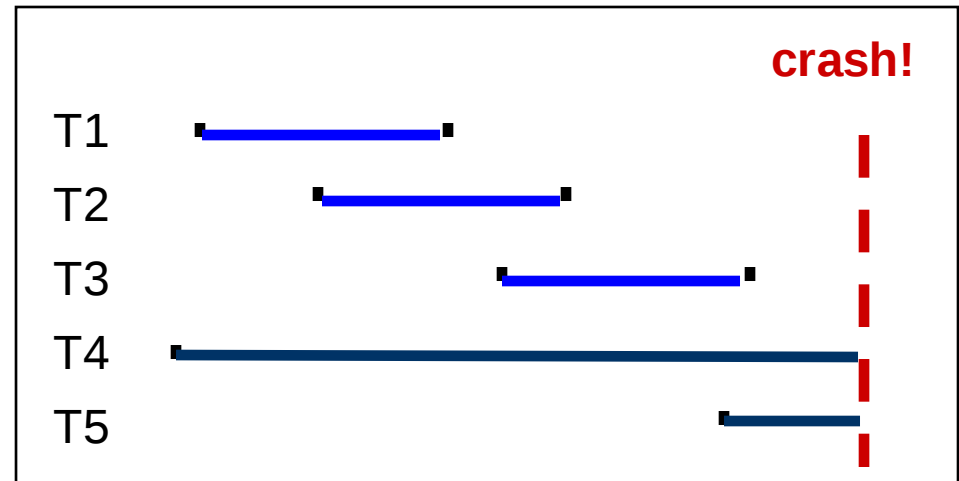
□ Atomicity:

- Transactions may abort (“Rollback”)

□ Durability:

- What if DBMS stops running?

- Desired Behavior after system restarts:
 - T1, T2 & T3 should be **durable**
 - T4 & T5 should be **aborted** (effects not seen)



The Goals of Recovery

1. When a transaction T **commits**

- Make the updates **permanent** in the database so that they can survive subsequent failures

2. When a transaction T **aborts**

- Obliterate any updates on data items by **aborted transaction** in the database
- Obliterate the effects of T **on other transactions**; i.e., transactions that read data items updated by T

3. When the system **crashes** after a failure

- Bring the database to its most recent consistent state

Recovery Actions

- Recovery protocols implement **two actions**:
 - **Undo** action:
 - Required for **atomicity**
 - Undoes all updates on the stable storage by an uncommitted transaction
 - **Redo** action:
 - Required for **durability**
 - Redoes the update on the stable storage of committed transaction

Logging

- A **log** or *journal* is a sequence of records which represent all modifications to the database in the order in which they actually occurred
- Log records may describe either *physical* changes or *logical* database operations
 - A **logical log** represents higher level operations; e.g., insert a record
 - A **physical log** contains information about the actual values of data items written by transactions.
 - State before change, **before image**
 - State after change, **after image**
 - Transition causing the change

Transaction Log

Types of log record:

1. **[start_transaction,T]**: Records that transaction T has started execution
2. **[write_item,T,X,old_value,new_value]**: Records that transaction T has changed the value of database item X from old_value to new_value.
3. **[read_item,T,X]**: Records that transaction T has read the value of database item X
4. **[commit,T]**: Records that transaction T has completed successfully
5. **[abort,T]**: Records that transaction T has been aborted

Transaction Log

TID	Back P	Next P	Operation	Data Item	BFIM	AFIM
T1	0	2	Begin			
T1	1	4	W	X	X = 100	X = 200
T2	0	7	Begin			
T1	2	5	W	Y	Y = 50	Y = 100
T1	4	8	R	M	M = 200	M = 200
T3	0	9	Begin			
T2	3	10	W	N	N = 20	N = 10
T1	5	nil	End			
T3	6	11	W	N	N = 10	N = 30

- ❑ **BFIM**: old values before modification (BeFore IMage)
- ❑ **AFIM**: new value after modification (AFter IMage)
- ❑ **Back P** and **Next P** point to the previous and next log records of the same transaction

What & How to Recover?

☐ What & how to **undo**?

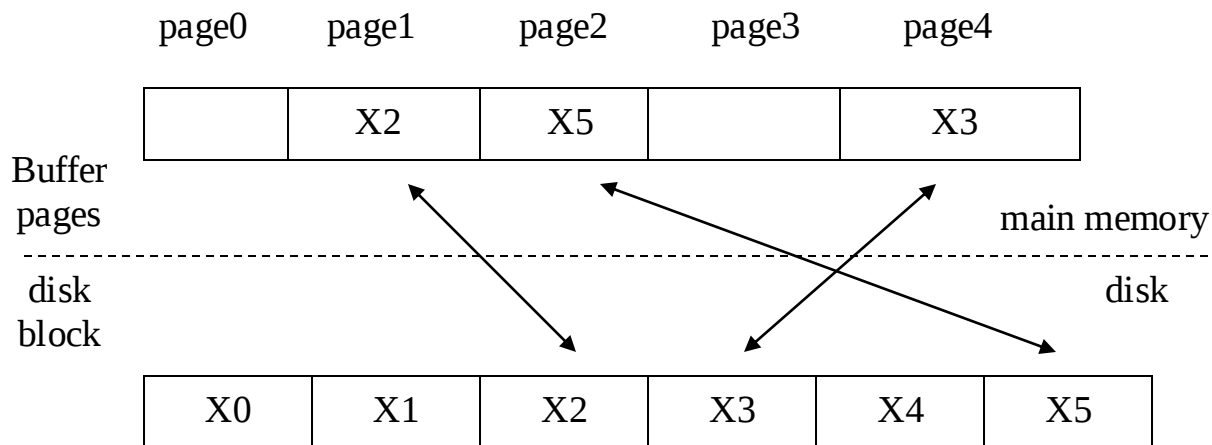
- For all uncommitted transactions
- Scan the log from end to beginning
- Use the BFIM

☐ What & how to **redo**?

- For all committed transactions
- Scan the log from beginning to end
- Use the AFIM

Buffer Manager

- ❑ The goal of Cache or Buffer Manager is to **maximize the likelihood** that a block of data needed by a transaction is in main memory
- ❑ The main memory is **partitioned** into *buffer blocks* or *buffer pages* (or simply buffers, blocks or pages)
- ❑ The size of a page is equal to the *disk block* size



Cache Flushing

- **Steal:** A cache page can be flushed before transaction commits – when?
- **No-Steal:** A cache cannot be flushed before transaction commits
- **Force:** Cache is flushed to disk when transaction commits
- **No-Force:** Otherwise! So when is it flushed?

Cache Flushing

- Leads to four different ways for handling recovery:
 - Steal/No-Force (Undo/Redo),
 - Steal/Force (Undo/No-redo),
 - No-Steal/No-Force (No-undo/Redo) and
 - No-Steal/Force (No-undo/No-redo).

Simplest Recovery Manager

□ Force

- No redo of committed transactions

□ No-Steal

- No undo of incomplete transactions

	No Steal	Steal
Force	Simple	
No Force		

Simplest Recovery Manager - Problems

☐ Force

- High I/O costs!

☐ No-Steal

- Large cache!

☐ Practical Recovery

☐ No-Force

- Need to Redo

☐ Steal

- Need to Undo

	No Steal	Steal
Force	Simple	
No Force		Practical

Write-Ahead Logging (WAL)

- Log is necessary for recovery and it must be available to recovery manager
- The **Write-Ahead Logging** (WAL) protocol states that:
- **For Undo:**
 - Before a data item's AFIM is **flushed** to the database disk (overwriting the BFIM) its **BFIM must be written** to the log and the log must be saved on a stable storage (disk)
- **For Redo:**
 - Before a transaction executes its **commit** operation, all its **AFIMs must be written** to the log and the log must be saved on a stable storage

Commit Point of a Transaction

- ❑ A transaction T reaches its **commit point** when:
 1. All its operations have been executed successfully
 2. The effect of all the transaction operations on the database has been **recorded in the log**.
- ❑ Beyond the commit point, its effect is assumed to be *permanently recorded* in the database.
- ❑ The transaction then writes an entry [commit,T] into the log.
- ❑ **Rollback of transactions:** Needed for transactions that have a [start_transaction,T] entry into the log but **no commit entry** [commit,T]

Recall: What & How to Recover?

□ What & how to **undo**?

- For all uncommitted transactions
- Scan the log from end to beginning
- Use the BFIM

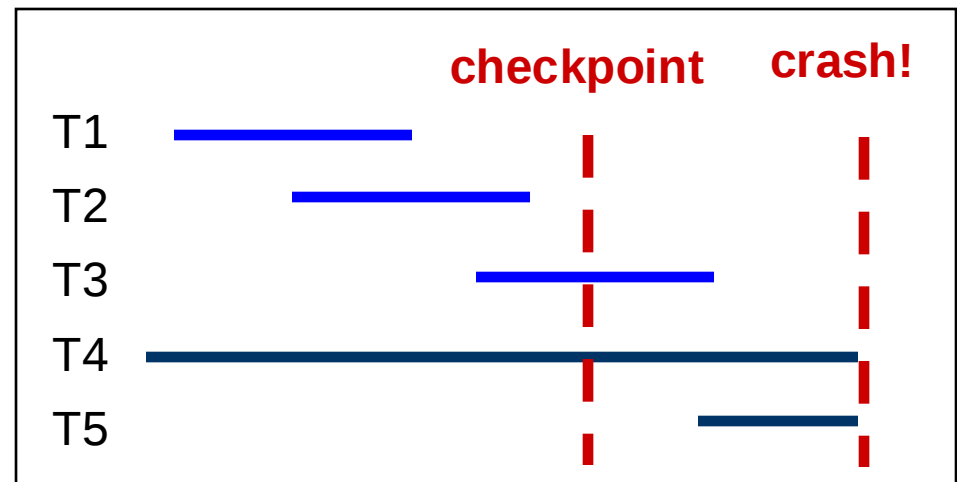
□ What & how to **redo**?

- For all committed transactions
- Scan the log from beginning to end
- Use the AFIM

Checkpoint

- ❑ Scanning the whole log is time-consuming!
- ❑ **Checkpoint**
 - Periodically written to the log
 - All cache pages are flushed to disk

- ❑ What transactions should be undo?
- ❑ What transactions should be redo?



Let's Try

□ Is the schedule conflict serializable?

S1: W2(X);W1(X);R3(X);R1(X);W2(Y);R3(Y);R3(Z);R2(X)

S2: R3(Z);W2(X);W2(Y);R1(X);R3(X);R2(Z);R3(Y);W1(X)

□ Which recovery operations are needed?

■ Checkpoint + steal/force

T1: started and committed before checkpoint

T2: started before checkpoint; not committed before crash

T3: started after checkpoint; committed before crash