Recovery Techniques

Overview

- Recovery Concepts
  - Purpose of Recovery
  - Types of Failure
  - Transaction Log
  - Cascading Rollback
  - Write-Ahead Logging (WAL)
  - Checkpoint
- Deferred Update
- Immediate Update
- Shadow Paging

Introduction

- **Purpose of Recovery**
  - To bring the database into the last consistent state, which existed prior to the failure.
  - To preserve transaction properties (Atomicity, Consistency, Isolation and Durability).

Introduction

- **Types of Failure**
- The database may become unavailable for use due to

Transaction failure: Transactions may fail because of incorrect input, deadlock, incorrect synchronization.

System failure: System may fail because of addressing error, application error, operating system fault, RAM failure, etc.

Media failure: Disk head crash, power disruption, etc.

Database Recovery

- **System (Transaction) Log**
- For recovery from any type of failure data values prior to modification (BFIM - BeFore Image) and the new value after modification (AFIM – AFter Image) are required.
- These values and other information is stored in a sequential file called Transaction log.
- A sample log is given below. **Back P** and **Next P** point to the previous and next log records of the same transaction.

T ID	Back P	Next P	Operation	Data item	BFIM	AFIM
T1	0	1	Begin			
T1	1	4	Write	X	X = 100	X = 200
T2	0	8	Begin			
T1	2	5	W	Y	Y = 50	Y = 100
T1	4	7	R	M	M = 200	M = 200
T3	0	9	R	N	N = 400	N = 400
T1	5	nil	End			

Database Recovery

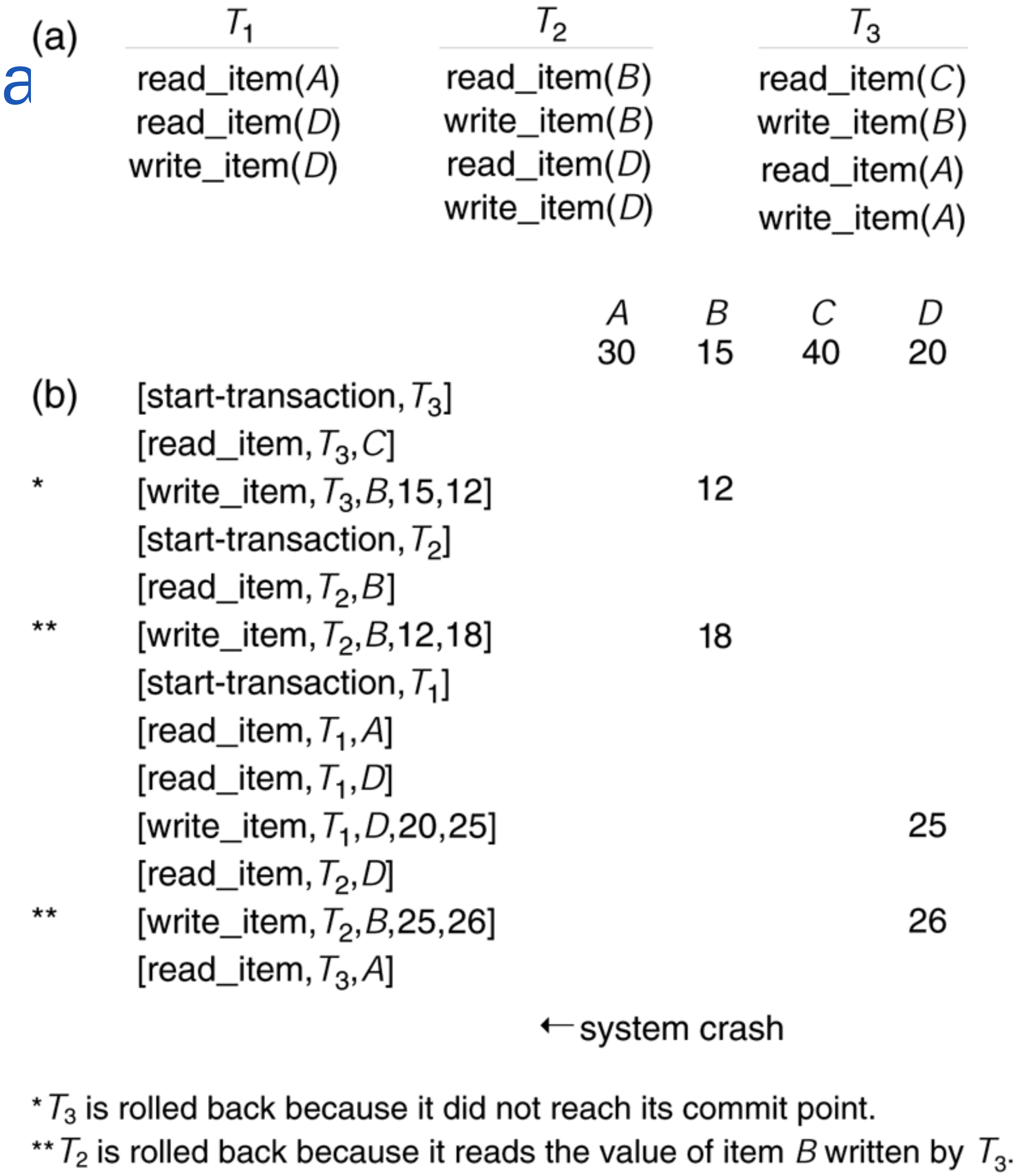
- **Transaction Roll-back (Undo) and Roll-Forward (Redo)**
- To maintain atomicity, a transaction’s operations are **redone** or **undone**.

Undo: Restore all BFIMs on to disk (Remove all AFIMs).

Redo: Restore all AFIMs on to disk.

Database recovery is achieved either by performing only Undos or only Redos or by a combination of the two. These operations are recorded in the log as they happen.

Data



(a)

T_1	T_2	T_3
read_item(A)	read_item(B)	read_item(C)
read_item(D)	write_item(B)	write_item(B)
write_item(D)	read_item(D)	read_item(A)
	write_item(D)	write_item(A)

		A	B	C	D
		30	15	40	20
(b)	[start-transaction, T_3]				
	[read_item, T_3,C]				
*	[write_item, T_3,B,15,12]		12		
	[start-transaction, T_2]				
	[read_item, T_2,B]				
**	[write_item, T_2,B,12,18]		18		
	[start-transaction, T_1]				
	[read_item, T_1,A]				
	[read_item, T_1,D]				
	[write_item, T_1,D,20,25]				25
	[read_item, T_2,D]				
**	[write_item, T_2,B,25,26]				26
	[read_item, T_3,A]				

← system crash

* T_3 is rolled back because it did not reach its commit point.

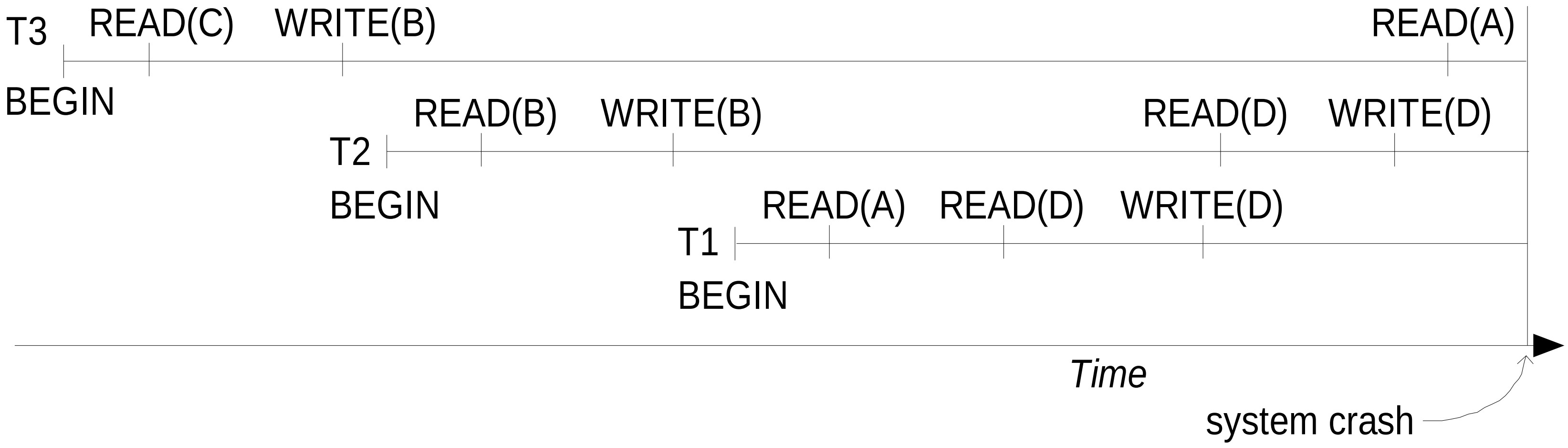
** T_2 is rolled back because it reads the value of item B written by T_3.

Cascading Rollback

(a) The read and write operations of three transactions.

(b) System log at point of crash.

Database Recovery



(c) Operations before the crash

Database Recovery

- **Write-Ahead Logging**
- When **in-place** update (immediate or deferred) is used then log is necessary for recovery and it must be available to recovery manager.
- This is achieved by **Write-Ahead Logging** (WAL) protocol. WAL states:

  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.

  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 store.

Database Recovery

- **Checkpointing**
- Time to time (randomly or under some criteria) the database flushes its buffer to database disk to minimize the task of recovery.
- The following steps defines a checkpoint operation:

  1. Suspend execution of transactions temporarily.
  2. Force write modified buffer data to disk.
  3. Write a [`checkpoint`] record to the log, save the log to disk.
  4. Resume normal transaction execution.

  During recovery **redo** or **undo** is required to transactions appearing after [checkpoint] record.

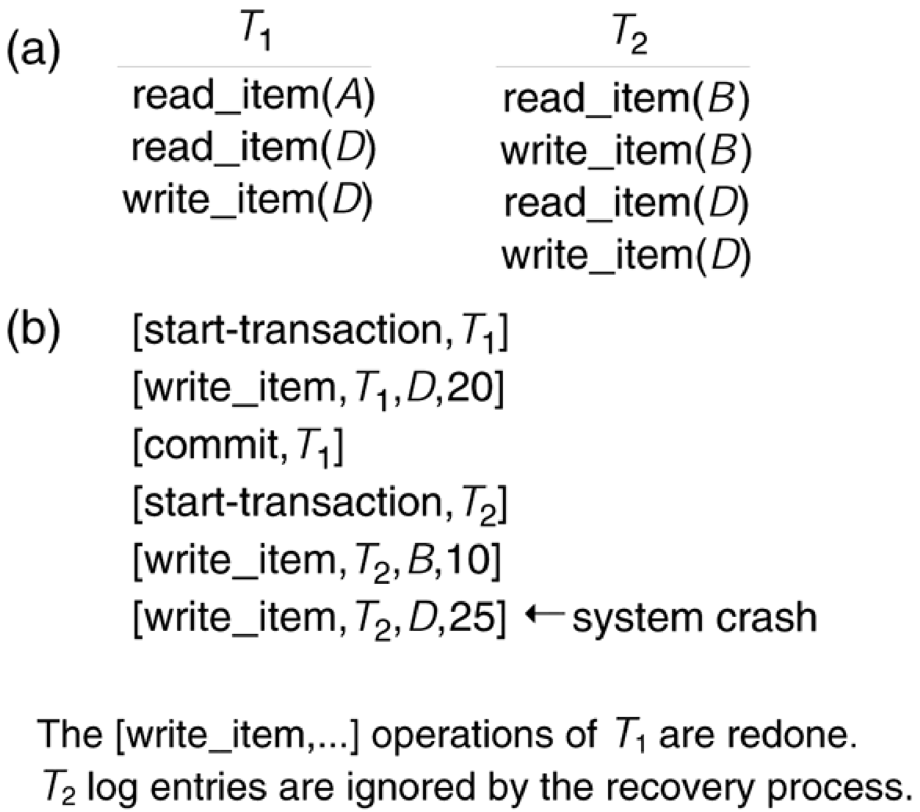
Database Recovery

- Introduction
- **Deferred Update**
  - Recovery Based on Deferred Update
  - Recovery in Single-user Environment
  - Concurrent Execution in Multi-user Environment
- Immediate Update
- Shadow Paging

Deferred Update

- Defer or postpone any updates to the database until the transaction completes its execution successfully or reaches its commit point
- Deferred Update (No UNDO/REDO):
    1. A set of transactions records their updates in the log.
    2. At commit point under WAL scheme these updates are saved on database disk.
- No UNDO is required because no AFIM is flushed to the disk before a transaction commits.
- REDO is required in case the system fails after a transaction commits but before all its changes are recorded in the database on disk.

Deferred Update in Single-User



(a)

T_1	T_2
read_item(A)	read_item(B)
read_item(D)	write_item(B)
write_item(D)	read_item(D)
	write_item(D)

(b)

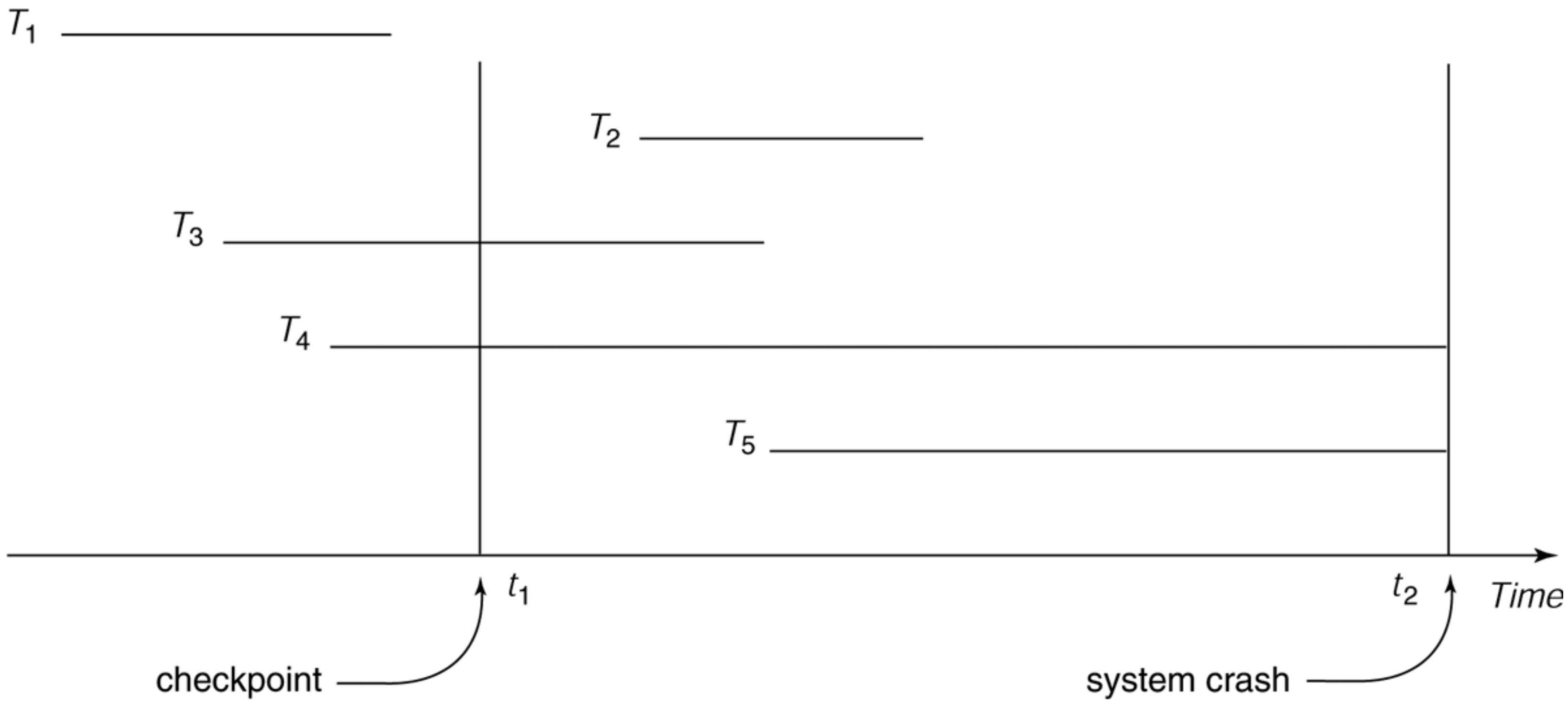
[start-transaction, T_1]
[write_item, T_1, D,20]
[commit, T_1]
[start-transaction, T_2]
[write_item, T_2, B,10]
[write_item, T_2, D,25] ← system crash

The [write_item,...] operations of T_1 are redone.
T_2 log entries are ignored by the recovery process.

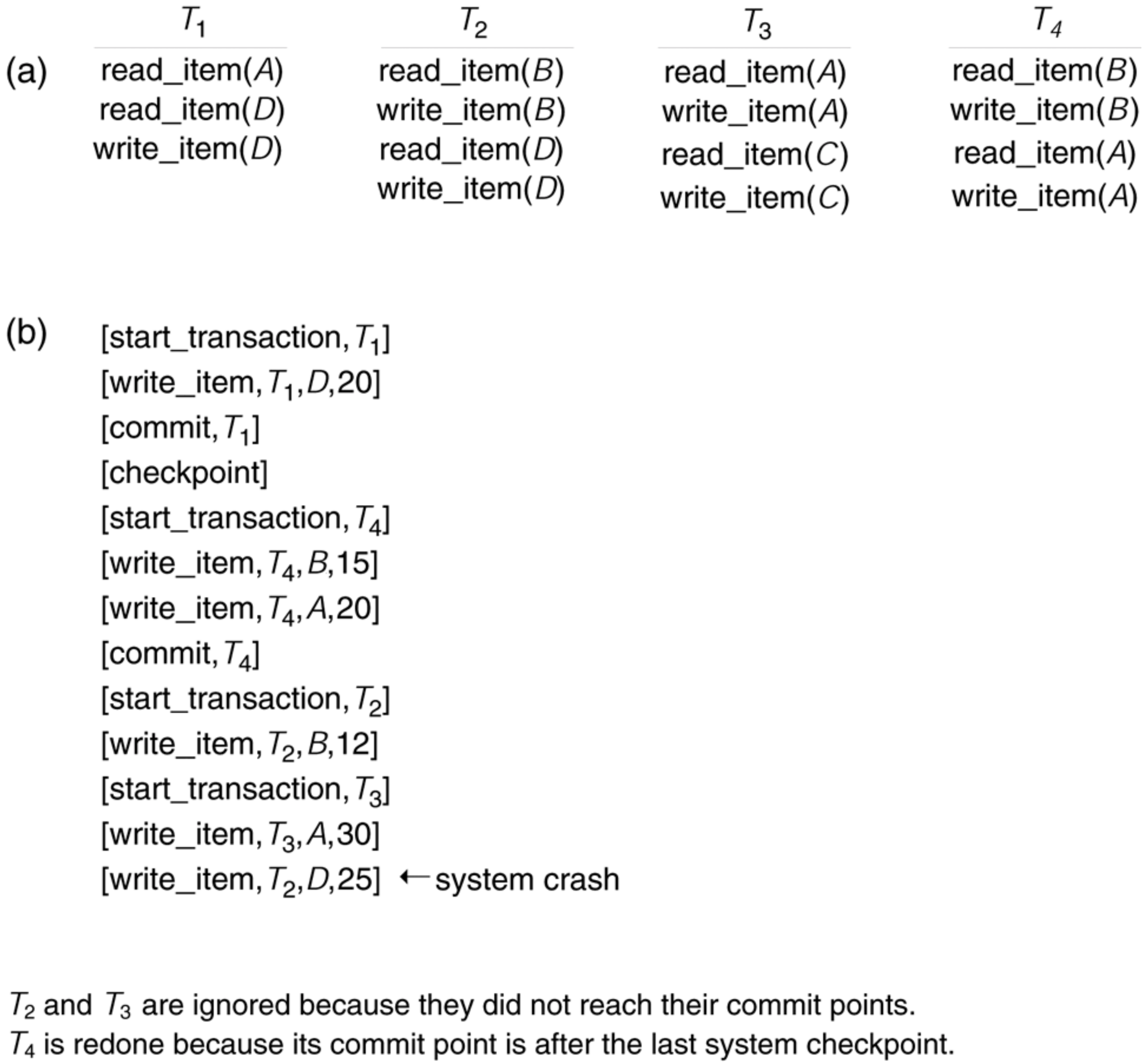
Deferred Update

- Two tables are required for implementing this protocol:
- **Active table**: All active transactions are entered in this table.
- **Commit table**: Committed transactions since the last checkpoint.
- During recovery, all transactions of the **commit** table are redone and all transactions of **active** tables are ignored since none of their AFIMs reached the database.
- WRITE operations of committed transactions are redone *in the order in which they were written to the log*

Deferred Update Concurrent Users



Deferred Update - Concurrent Transactions



(a)

T_1	T_2	T_3	T_4
read_item(A)	read_item(B)	read_item(A)	read_item(B)
read_item(D)	write_item(B)	write_item(A)	write_item(B)
write_item(D)	read_item(D)	read_item(C)	read_item(A)
	write_item(D)	write_item(C)	write_item(A)

(b)

[start_transaction, T_1]
[write_item, T_1, D, 20]
[commit, T_1]
[checkpoint]
[start_transaction, T_4]
[write_item, T_4, B, 15]
[write_item, T_4, A, 20]
[commit, T_4]
[start_transaction, T_2]
[write_item, T_2, B, 12]
[start_transaction, T_3]
[write_item, T_3, A, 30]
[write_item, T_2, D, 25] ← system crash

T_2 and T_3 are ignored because they did not reach their commit points.
T_4 is redone because its commit point is after the last system checkpoint.

Database Recovery

- Introduction
- Deferred Update
- **Immediate Update**
  - Recovery Based on Immediate Update
  - Recovery in Single-user Environment
  - Concurrent Execution in Multi-user Environment
- Shadow Paging

Immediate Update

- When a transaction issues an update command, the database can be updated "immediately", without any need to wait for transaction to reach its commit point
- Immediate Update (UNDO/REDO):
  1. Transactions records their updates in the log *before* it is applied to the database using WAL
  2. Transaction is allowed to commit *before* all its changes are written to the database
- UNDO is required because AFIM is flushed to the disk before a transaction commits.
- REDO is required in case the system fails after a transaction commits but before all its changes are recorded in the database on disk.

Immediate Update

- In a single-user environment no concurrency control is required but a log is maintained under WAL.
- Note that at any time there will be one transaction in the system and it will be either in the commit table or in the active table.
- The recovery manager performs:

1. **Undo** of a transaction if it is in the **active** table.
2. **Redo** of a transaction if it is in the **commit** table.

Immediate Update

- In concurrent execution environment a concurrency control is required and log is maintained under WAL.
- To minimize the work of the recovery manager checkpointing is used.
- Commit table records transactions to be committed and active table records active transactions.
- During recovery, all transactions of the **commit** table are *redone* and all transactions of **active** tables are *undone*.
- UNDO:

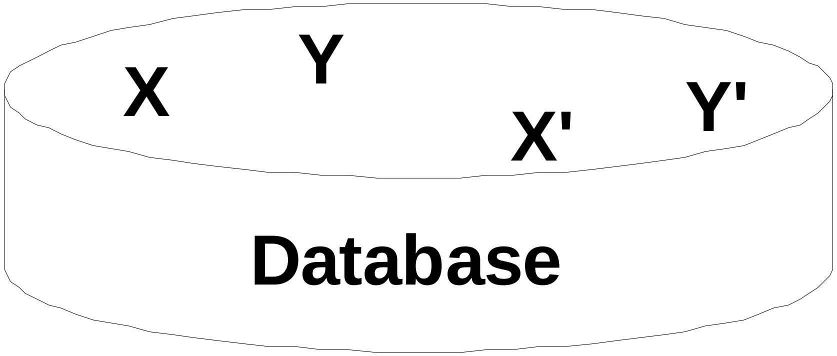
1. Examine the log entry [write_item, T, X, old_value, new_value], and set the value of X in the database to old_value (BFIM)
2. UNDO must proceed in the *reverse order* from the order in which the operations were written in the log

Database Recovery

- Introduction
- Deferred Update
- Immediate Update
- **Shadow Paging**
  - Current directory & Shadow directory
  - No-UNDO/No-REDO Algorithm

Shadow Paging

- The AFIM does not overwrite its BFIM but recorded at another place on the disk.
- Thus, at any time a data item has AFIM and BFIM (Shadow copy of the data item) at two different places on the disk.

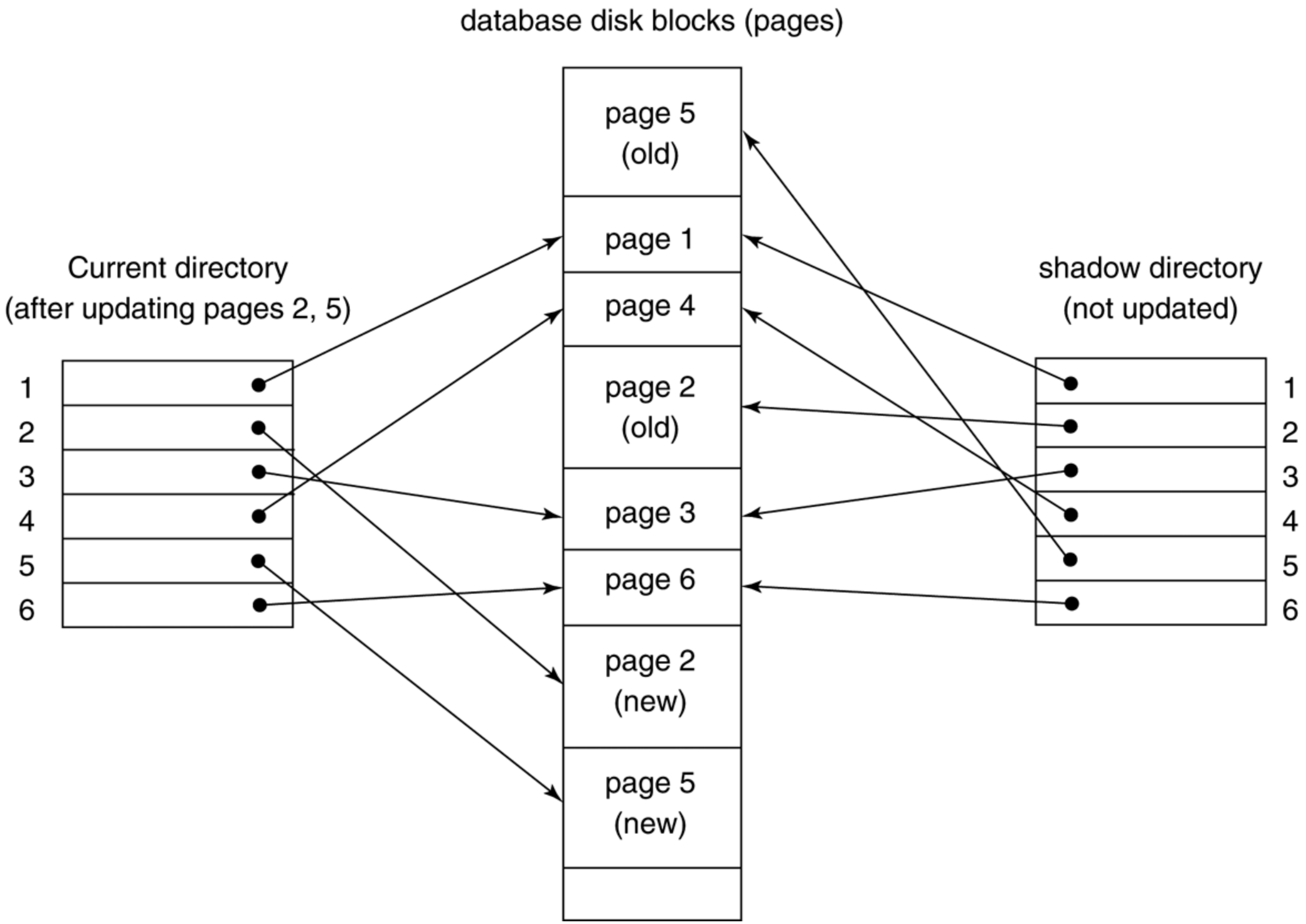


X and Y: Shadow copies of data items
X` and Y`: Current copies of data items

Shadow Paging

- Database is made up of a number of fixed-size disk pages for recovery.
- A directory with n entries – where i^{th} entry points to the i^{th} database page on disk.
- When a transaction begins, the current directory – points to recent pages on disk – is copied into a shadow directory.
- During transaction execution, the shadow directory is *never* modified.
- When write_item operation, a new copy of database page is created and the current directory entry is modified to point to the new disk block.

Shadow Paging



Shadow Paging

- To recover from failure, free the modified database pages and discard the current directory.
- Committing corresponds to discarding the shadow directory.
- Recovery involves neither UNDO nor REDO – no-undo/no-redo technique.
- Disadvantages:
  - Need of complex storage management strategies.
  - Overhead of writing shadow directories to disk, if directory is large.
  - Released pages must be added to a list of free pages for future use – garbage collection.

References

- Fundamentals of Database Systems, Elmasri and Navathe, 3rd Edition

THANK
YOU!