Chapter 19:
Database Recovery Concepts

Prof. Kirtankumar Rathod

Dept. of Computer Science

Indus University

Purpose of database 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)

- Example: Suppose, the system crashes before a fund transfer transaction completes its execution, then either one or both accounts may have incorrect value. Thus, the database must be restored to the state before the transaction modified any of the accounts..

Purpose of Backup and Recovery

- As a backup administrator, your principal duty is to devise, implement, and manage a backup and recovery strategy.
- In general, the purpose of a backup and recovery strategy is to protect the database against data loss and reconstruct the database after data loss.

What is Data Recovery?

- Data recovery is the process of salvaging and handling the data through the data from damaged, failed, corrupted, or inaccessible secondary storage media when it cannot be accessed normally.

- Recovery may be required due to physical damage to the storage device or logical damage to the file system that prevents it from being mounted by the host operating system (OS).

- Recovery should protect the database and associated users from unnecessary problems and avoid or reduce the possibility of having to duplicate work manually.

Two main techniques:

- We can distinguish two main techniques for recovery from noncatastrophic transaction failures:

(1) deferred update and (2) immediate update.

1. Deferred update

- These techniques do not physically update the database on disk until after a transaction reaches its commit point; then the updates are recorded in the database.
- If a transaction fails before reaching its commit point, it will not have changed the database in any way, so UNDO is not needed.
- REDO could be needed During transaction execution, the updates are recorded only in the log and in the cache buffers.

1. Deferred update

i. A transaction cannot change the database on disk until it reaches its commit point.

ii. A transaction does not reach its commit point until all its REDO-type log entries are recorded in the log and the log buffer is force-written to disk.

2. Immediate update

- When a transaction issues an update command, the database on disk can be updated immediately, without any need to wait for the transaction to reach its commit point.
- Provisions must be made for undoing the effect of update operations that have been applied to the database by a failed transaction.
- If the recovery technique ensures that all updates of a transaction are recorded in the database on disk before the transaction commits, there is never a need to REDO any operations of committed transactions.

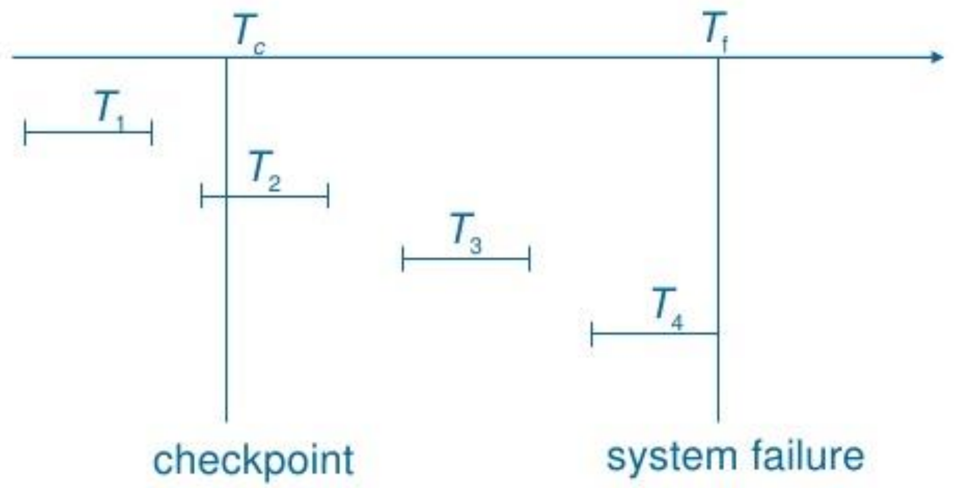
Checkpoints in the System log

- A [checkpoint] record is written into the log periodically at that point when the system writes out to the database on disk all DBMS buffers that have been modified.

- All transactions that have their [commit, T] entries in the log before a [checkpoint] entry do not need to have their WRITE operations *redone* in case of a system crash, since all their updates will be recorded in the database on disk during check-pointing.

Checkpoints in the System log

Example of Checkpoints

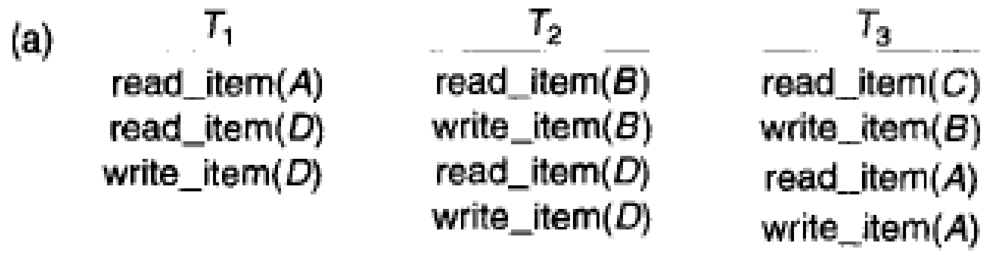


- T_1 can be ignored (updates already output to disk due to checkpoint)
- T_2 and T_3 redone.
- T_4 undone

Transaction Rollback

- If a transaction fails for whatever reason after updating the database, it may be necessary to roll back the transaction.
- If any data item values have been changed by the transaction and written to the database, they must be restored to their previous values.
- The undo-type log entries are used to restore the old values of data items that must be rolled back.

Consider following example:

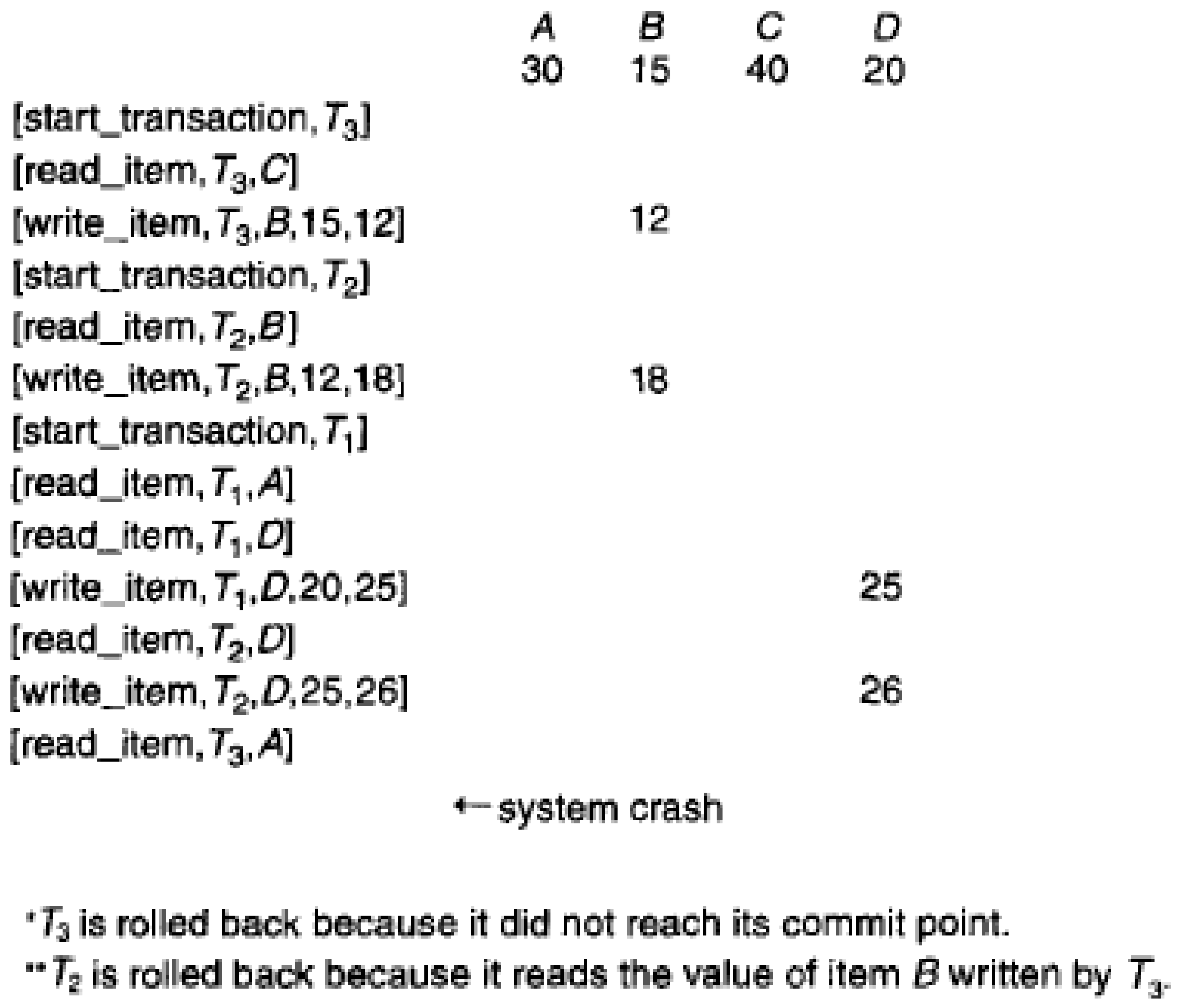


(a)

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*)

- There are 3 transactions T1, T2 and T3.
- Values of A,B,C and D are 30,15,40 and 20 respectively.
- Also, consider the given read and write operations of all transactions which is given in next slide….

Schedule is given below…



	A	B	C	D
	30	15	40	20
[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, D, 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.

Log-Based Recovery

- Logs are the sequence of records, that maintain the records of actions performed by a transaction.
- In Log – Based Recovery, log of each transaction is maintained in some stable storage. If any failure occurs, it can be recovered from there to recover the database.
- The log contains the information about the transaction being executed, values that have been modified and transaction state.
- All these information will be stored in the order of execution.

Example..

- Assume, a transaction to modify the address of an employee. The following logs are written for this transaction,

  Log 1: Transaction is initiated, writes 'START' log.
  Log: <T_n START>

  Log 2: Transaction modifies the address from 'Pune' to 'Mumbai'.
  Log: <T_n Address, 'Pune', 'Mumbai'>

  Log 3: Transaction is completed. The log indicates the end of the transaction.
  Log: <T_n COMMIT>

- **There are two methods of creating the log files and updating the database,**

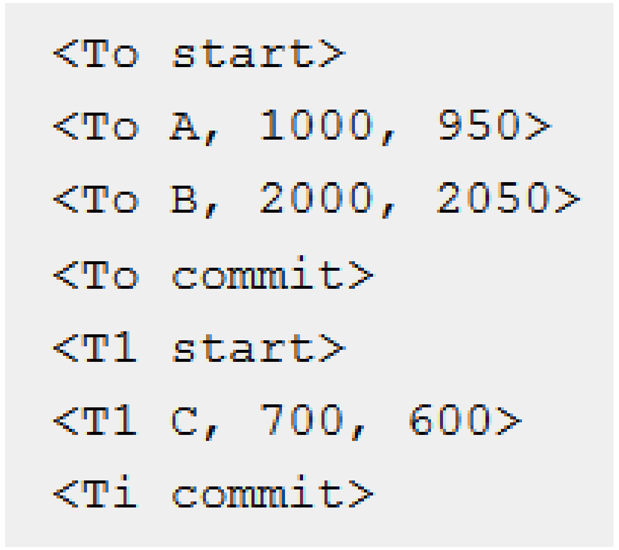
1. Deferred Database Modification

2. Immediate Database Modification

1. **In Deferred Database Modification,** all the logs for the transaction are created and stored into stable storage system. In the above example, three log records are created and stored it in some storage system, the database will be updated with those steps.

2. **In Immediate Database Modification,** after creating each log record, the database is modified for each step of log entry immediately. In the above example, the database is modified at each step of log entry that means after first log entry, transaction will hit the database to fetch the record, then the second log will be entered followed by updating the employee's address, then the third log followed by committing the database changes.

Example



```
<To start>
<To A, 1000, 950>
<To B, 2000, 2050>
<To commit>
<T1 start>
<T1 C, 700, 600>
<Ti commit>
```

Here, we consider an example of banking system taken earlier for transaction To and T1 such that To is followed by T1.

If the system crash occurs just after the log record and during recovery we do redo (To) and undo (T1) as we have both < To start > and <To commit> in the log record.

But we do not have <T1 commit> with <T1 start> in log record. Undo(T1) should be done first then redo (To) should be done.

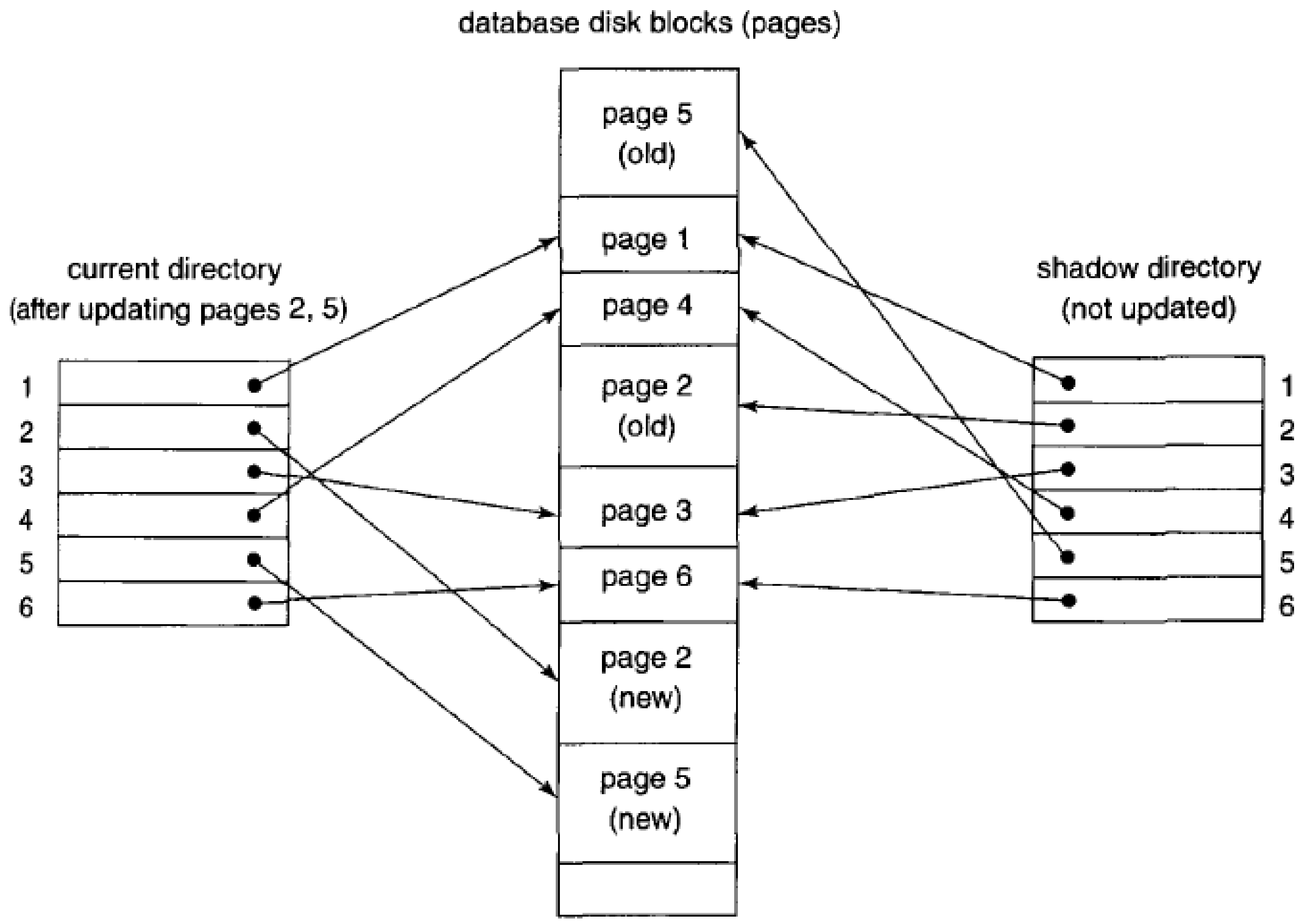
SHADOW PAGING

- This recovery scheme does not require the use of a log in a single-user environment. In a multiuser environment, a log may be needed for the concurrency control method.
- Shadow paging considers the database to be made up of a number of fixed-size disk pages (or disk blocks)-say, n-for recovery purposes.
- When a transaction begins executing, the current directory-whose entries point to the most recent or current database pages on disk-is copied into a shadow directory. The shadow directory is then saved on disk while the current directory is used by the transaction.

SHADOW PAGING

- During transaction execution, the shadow directory is *never* modified.
- When a write_item operation is performed, a new copy of the modified database page is created, but the old copy of that page is *not overwritten.*
- Instead, the new page is written elsewhere-on some previously unused disk block.
- The current directory entry is modified to point to the new disk block, whereas the shadow directory is not modified and continues to point to the old unmodified disk block.

Example of shadow paging…


database disk blocks (pages)
page 5 (old)
page 1
page 4
page 2 (old)
page 3
page 6
page 2 (new)
page 5 (new)
current directory (after updating pages 2, 5)
1
2
3
4
5
6
shadow directory (not updated)
1
2
3
4
5
6