
DBMS Database
Management
System
ctrl
alt

RECOVERY CONCEPTS

- Recovery Outline and Categorisation of Recovery Algorithm
- Caching (Buffering) of Disk Blocks
- Write - Ahead Logging, Steal/No-Steal, and, Force /No -Force
- Checkpoints in the System Log and Fuzzy Checkpointing
- Transaction Rollback and Cascading Rollback
- Transactions Actions that Do not Affect the Database

Recovery outline and categorisation of recovery algorithms:

- Recovery from transaction failures usually means that the database is restored to the most recent consistent state just before the time of failure

- To do this, the system must keep information about the changes that were applied to data items by the various transactions. This information is typically kept in the **system log.**

- To Distinguish two main techniques for recovery from non-catastrophic transaction failures

    1. ***Deferred update Techniques***
    2. ***Immediate update Techniques***

Deferred Update Technique:

- They do not physically update the database on disk until after a transaction reaches its commit point; then the updates are recorded in the database
- Before reaching commit, all transaction updates are recorded in the local transaction workspace or in the main memory buffers that the DBMS maintains
- Before commit, the updates are recorded persistently in the log, and then after commit, the updates are written to the database on disk

- If a transaction fails before reaching its commit point, it will not have changed the database in any way, so UNDO is not needed
- It may be necessary to REDO the effect of the operations of a committed transaction from the log, because their effect may not yet have been recorded in the database on disk
- Hence, deferred update is also known as the **NO-UNDO/REDO algorithm**

Immediate update Technique:

- The database may be updated by some operations of a transaction before the transaction reaches its commit point.
- However, these operations must also be recorded in the log on disk by force-writing before they are applied to the database on disk, making recovery still possible
- If a transaction fails after recording some changes in the database on disk but before reaching its commit point, the effect of its operations on the database must be undone; that is, the transaction must be rolled back
- This technique, known as the **UNDO/REDO algorithm**, requires both operations during recovery

Caching (Buffering) of Disk Blocks:

- Typically a collection of in-memory buffers, called the **DBMS cache**, is kept under the control of the DBMS for the purpose of holding these buffers.
- A **directory** for the cache is used to keep track of which database items are in the buffers
- This can be a table of <Disk_page_address, Buffer_location, ... > entries.
- Associated with each buffer in the cache is a **dirty bit**, which can be included in the directory entry, to indicate whether or not the buffer has been modified.
- When a page is first read from the database disk into a cache buffer, a new entry is inserted in the cache directory with the new disk page address, and the dirty bit is set to 0 (zero).
- As soon as the buffer is modified, the dirty bit for the corresponding directory entry is set to 1 (one)

- Two main strategies can be employed when flushing a modified buffer back to disk.
- - The first strategy, known as **In-place updating,** writes the buffer to the same original disk location, thus overwriting the old value of any changed data items on
disk. Hence, a single copy of each database disk block is maintained.
- - The second strategy, known as **shadowing**, writes an updated buffer at a different disk location, so multiple versions of data items can be maintained, but this approach is not typically used in practice.

Write - Ahead Logging, Steal/No-Steal, and, Force /No -Force:

- The recovery mechanism must ensure that the BFIM(Before image) of the data item is recorded in the appropriate log entry and that the log entry is flushed to disk before the BFIM is overwritten with the AFIM(After image) in the database on disk.This process is generally known as **write-ahead logging**
- **A REDO-type log entry** includes the new value (AFIM) of the item written by the operation
- **The UNDO-type log entries** include the old value (BFIM) of the item since this is needed to undo the effect of the operation from the log

Steal/ No-Steal, Force/No-Force:

- If a cache buffer page updated by a transaction cannot be written to disk before the transaction commits, the recovery method is called a **no-steal approach.**
- If the recovery protocol allows writing an updated buffer before the transaction commits, it is called **steal.**
- If all pages updated by a transaction are immediately written to disk before the transaction commits, it is called a **force approach**. Otherwise, it is called **no- force.**
- For example, consider the following **write-ahead logging(WAL)** protocol for a recovery algorithm that requires both UNDO and REDO:
- The before image of an item cannot be overwritten by its after image in the database on disk until all UNDO-type log records for the updating transaction up to this point have been force-written to disk.
- The commit operation of a transaction cannot be completed until all the REDO-type and UNDO-type log records for that transaction have been force written to disk.

Checkpoints in the System Log and Fuzzy Checkpointing:

- A [check point, list of active transactions] 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

- Taking a checkpoint consists of the following actions:
  1. Suspend execution of transactions temporarily.
  2. Force-write all main memory buffers that have been modified to disk.
  3. Write a [checkpoint] record to the log, and force-write the log to disk.
  4. Resume executing transactions.

- In **fuzzy checkpointing** technique, the system can resume transaction processing after a [begin_checkpoint] record is written to the log without having to wait for step 2 to finish.

- When step 2 is completed, an [end_checkpoint, ...] record is written in the log with the relevant information collected during checkpointing.

Transaction Rollback and Cascading Rollback:

- If a transaction fails for whatever reason after updating the database, but before the transaction commits, it may be necessary to **roll back** the transaction
- If a transaction T is rolled back, any transaction S that has, in the interim, read the value of some data item X written by T must also be rolled back
- Similarly, once S is rolled back, any transaction R that has read the value of some data item Y written by S must also be rolled back; and so on. This phenomena is called as **cascading rollback.**

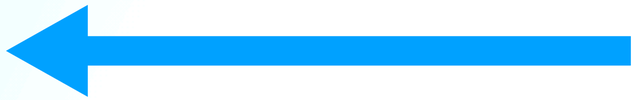
.

T1
read_item(A)
read_item(D)
write_item(D)

T2
read_item(B)
write_item(B)
read_item(D)
write_item(D)

T3
read_item(C)
write_item(B)
read_item(A)
write_item(A)

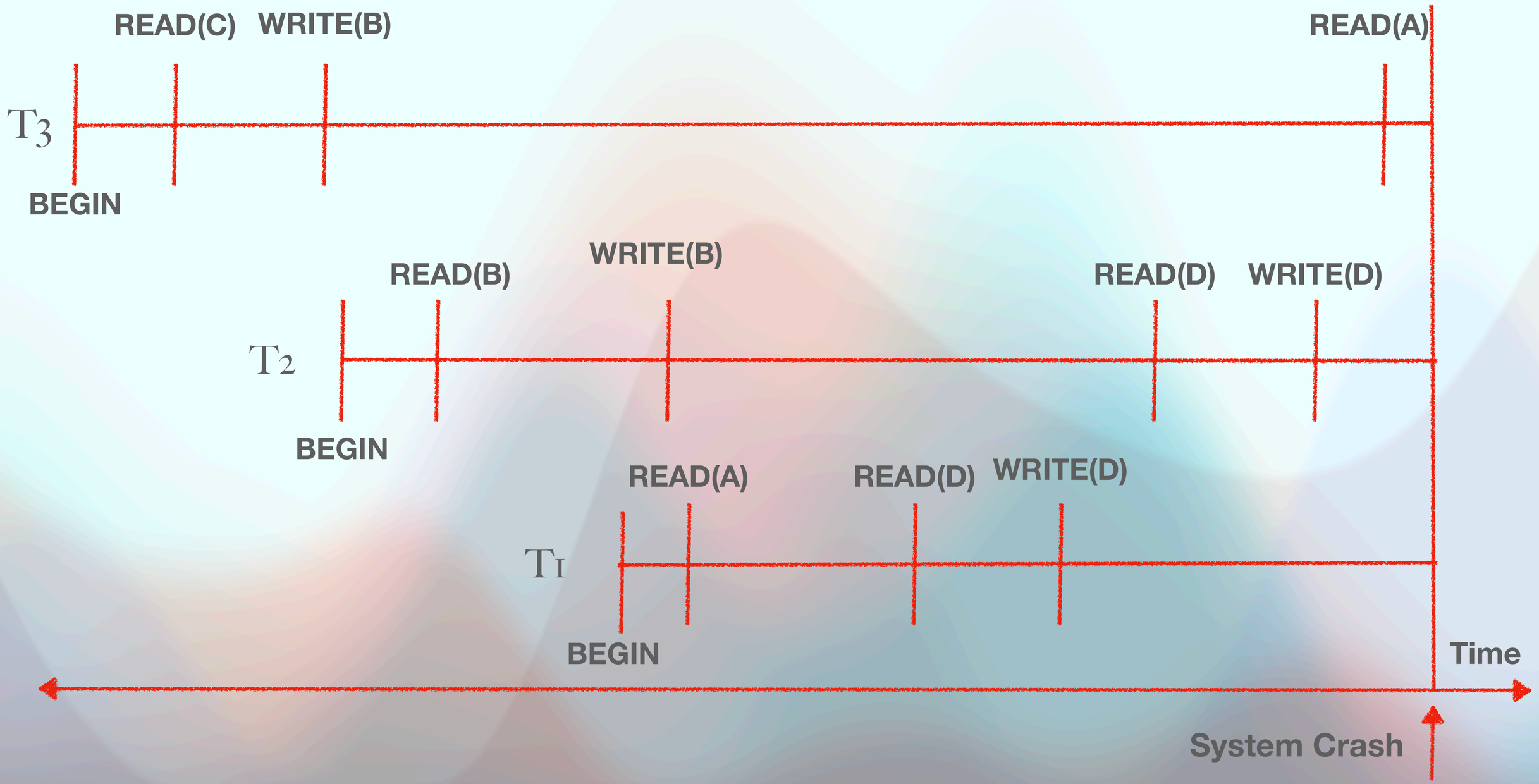
	A	B	C	D
	30	15	40	20
[start_transaction, T3]				
[read_item, T3,C]				
* [write_item, T3, B,15,12]		12		
[start_transaction, T2]				
[read_item, T2,B]				
** [write_item, T2,B,12, 18]		18		
[start_transaction,T1]				
[read_item, T1, A]				
[read_item,T1,D]				
[write_item,T1,D,20,25]				25
[read_item,T2,D]				
** [write_item,T2,D, 25,26]				26
[read_item, T3,A]				



System crash

* T3 is rolled back because it Did not reach its commit point

** T2 is rolled back because it reads the value of item B written by T3


READ(C)
WRITE(B)
READ(A)
T3
BEGIN
READ(B)
WRITE(B)
READ(D)
WRITE(D)
T2
BEGIN
READ(A)
READ(D)
WRITE(D)
T1
BEGIN
Time
System Crash

Transaction Actions that do not Affect the Database:

- In general, a transaction will have actions that do not affect the database, such as generating and printing messages or reports from information retrieved from the database
- If such erroneous reports are produced, part of the recovery process would have to inform the user that these reports are wrong, since the user may take an action based on these reports that affects the database
- Hence, such reports should be generated only after the transaction reaches its commit point.

No UNDO/REDO RECOVERY BASED ON DEFERRED UPDATE

- The idea behind deferred update is to defer or postpone any actual updates to the database itself until the transaction completes its execution successfully and reaches its commit point.
- During transaction execution, the updates are recorded only in the log and in the transaction workspace. After the transaction reaches its commit point and the log is force-written to disk, the updates are recorded in the database itself.
- If a transaction fails before reaching its commit point, there is no need to undo any operations, because the transaction has not affected the database in any way.

THE STEPS INVOLVED IN DEFERRED UPDATE PROTOCOL ARE AS FOLLOWS:

- When a transaction starts, write an entry start_transaction(T) to the log.
- When any operation is performed that will change values in the database, write a log entry write_item(T, x, old_value, new_value).
- When a transaction is about to commit, write a log record of the form commit(T); write all log records to disk.
- Commit the transaction, using the log to write the updates to the database; the writing of data to disk need not occur immediately.
- If the transaction aborts, ignore the log records and do not write the changes to disk.

- The database is never updated until after the transaction commits, And there is never need to UNDO any operations hence, this technique is known as NO-UNDO /REDO algorithm

Log entry	Log written to disk	Changes written to database buffer	Changes written on disk
start_transaction(T)	No	N/A	N/A
read_item(T, x)	No	N/A	N/A
write_item(T, x)	No	No	No
commit(T)	Yes	Yes	*Yes
checkpoint	Yes	Undefined	Yes(of committed Ts)

*Yes: writing back to disk may occur not immediately.

DEFERRED UPDATE IN SINGLE USER ENVIRONMENT:

- Use two lists to maintain the transactions: the committed transactions list, which contains all the committed transactions since the last checkpoint, and the active transactions list (at most one transaction falls in this category, because the system is a single-user one).
- Apply the REDO operation to all the write_item operations of the committed transactions from the log in the order in which they were written to the log. Restart the active transactions.

The REDO procedure is defined as follows:

- Redoing a write_item operation consists of examining its log entry write_item(T, x, old_value, new_value) and setting the value of item x in the database to new_value.

- This method's main benefit is that any transaction operation need never be undone, as a transaction does not record its changes in the database until it reaches its commit point.

Action	Entry in log	
	start_transaction(T)	commit(T)
Re-submit	Yes	No
Redo	Yes	Yes

Example of Recovery in single User environment:

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)

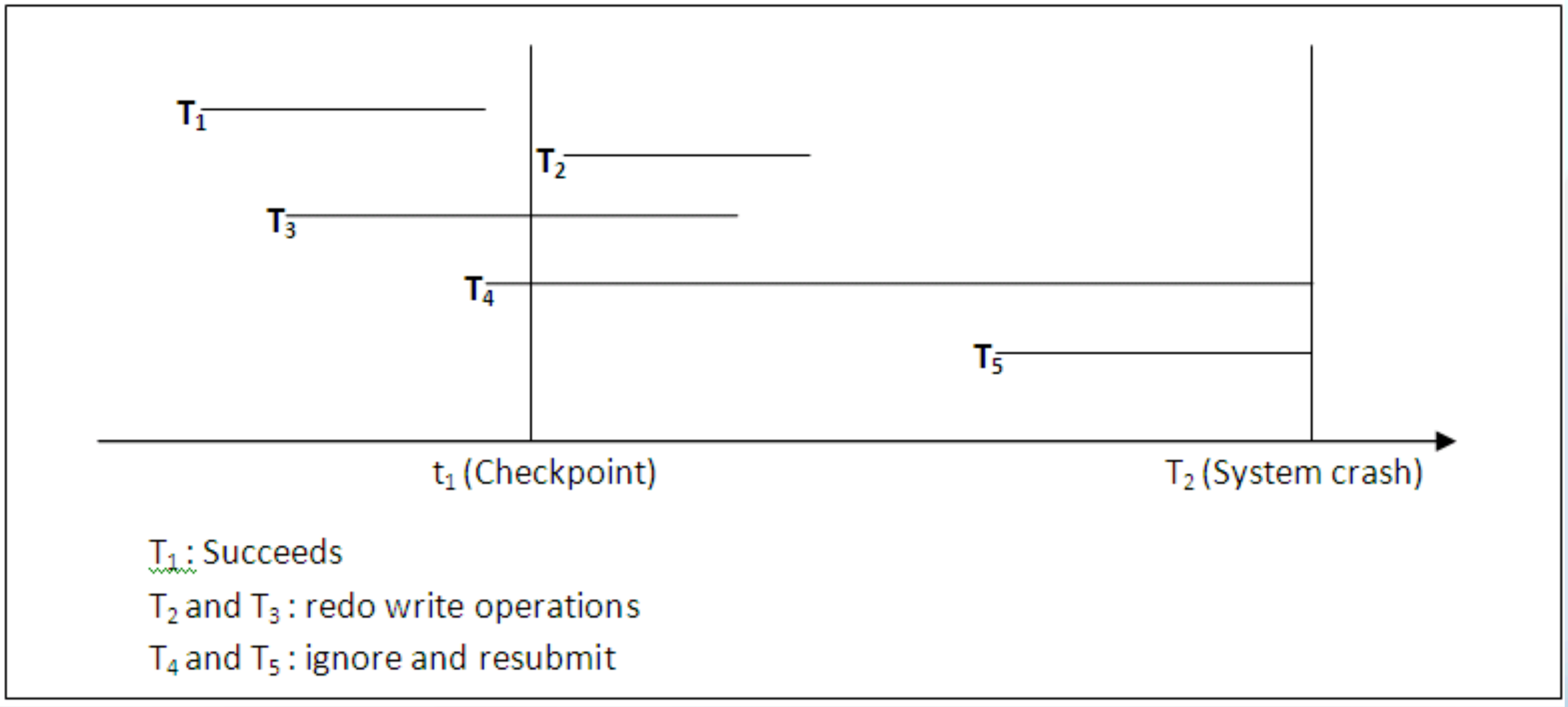
start_transaction(T1)	
write_item(T1, D, 20)	
commit(T1)	
start_transaction(T2)	
write_item(T2, B, 10)	
write_item(T2, D 25)	← System crash

In given figure, the first failure occurs during execution of transaction T2. The recovery process will redo the write_item(T1, D, 20) entry in the log by resetting the value of item D to 20 (its new value). The write(T2, â€¦) entries in the log are ignored by the recovery process because T2 is not committed. If a second failure occurs during recovery from the first failure, the same recovery process is repeated from start to finish, with identical results.

DEFERRED UPDATE IN MULTI-USER ENVIRONMENT:

- Use two lists of transactions maintained by the system: the committed transactions list which contains all committed transactions since the last checkpoint, and the active transactions list.
- REDO all the write operations of the committed transactions from the log, in the order in which they were written into the log.
- The transactions in the active list that are active and did not commit are effectively cancelled and must be resubmitted.

Example of Recovery in multi-user environment:



T_1 : Succeeds

T_2 and T_3 : redo write operations

T_4 and T_5 : ignore and resubmit

- The diagram above shows an example schedule of executing transactions. When the checkpoint was taken at time t1, transaction T1 had committed, whereas transaction T3 and T4 had not.
- Before the system crash at time t2, T3 and T2 were committed but not T4 and T5.
- According to the deferred update method, there is no need to redo the write operations of transaction T1 or any transactions committed before the last checkpoint time t1.
- Write operations of T2 and T3 must be redone, however, because both transactions reached their commit points after the last checkpoint.
- Recall that the log is force-written before committing a transaction. Transaction T4 and T5 are ignored: they are effectively cancelled and rolled back because none of their write operations were recorded in the database under the deferred update protocol.

Deferred update protocol:

- Given the operations of the four concurrent transactions in (1) below and the system log at the point of system crash in (2), discuss how each transaction recovers from the failure using deferred update.

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)

1) The read and write operations of four transactions

start_transaction(T_1)	
write_item(T_1, D, 20)	
commit(T_1)	
checkpoint	
start_transaction(T_4)	
write_item(T_4, B, 15)	
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

2)System log at point of crash

T2 And T3 are ignored because they did not reach their commit point.
T4 is redone it's commit point is after last system check point

- The method’s main benefit is that transaction operations never need to be undone, for two reasons:
- A transaction does not record any changes in the database on disk until after reaches its commit point — that is, until it completes its execution successfully. Hence, Transaction is never rolled back because of failure during transaction execution.
- A transaction will never read the value of an item that is written by an uncommitted transaction, because items remain locked until a transaction reaches its commit point. Hence, no cascading rollback will occur.


BY AMAAN&
REVAN
THANK YOU....
DON'T WISH FOR IT, WORK FOR IT..!!