Mansoura University
Faculty of Computers and Information
Department of Information System
First Semester- 2021-2022

[IS313P] Database System II

Grade: 3 rd. IS & IT

Dr. Amira Rezk

DATABASE RECOVERY TECHNIQUES

AGENDA

Recovery Concepts

✓ NO-UNDO/REDO Recovery Based on Deferred Update

Recovery Techniques Based on Immediate Update

RECOVERY CONCEPTS

- Recovery from transaction failures 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. (**system log)**
- **Recover from:**
  - **Physical damages (backup)**
  - **Non-physical damages (recovery techniques)**
    - **in deferred update (NO-UNDO/REDO algorithm)**
    - **in immediate update (UNDO/REDO algorithm/ UNDO/NO-REDO algorithm.**)

CACHING (BUFFERING) OF DISK BLOCKS

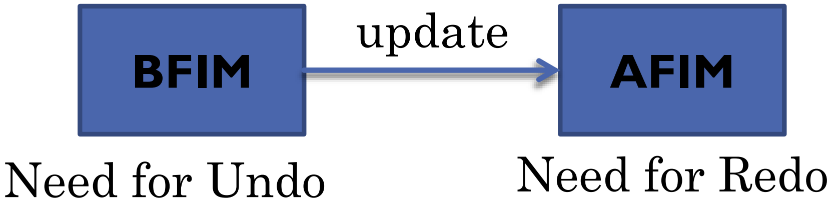
- Data items to be modified are first stored into database cache by the Cache Manager (CM) and after modification they are flushed (written) to the disk.
- The flushing is controlled by Modified and Pin-Unpin bits.
  - Modified (dirty bit): to indicate whether or not the buffer has been modified.
  - Pin-Unpin (pin-unpin bit): Instructs the operating system not to flush the data item.
    - (bit value I (one)) if it cannot be written back to disk as yet.
    - For example, the recovery protocol may restrict certain buffer pages from being written back to the disk until the transactions that changed this buffer have committed

TYPES OF DATA UPDATE

- Based on time of update
  - Immediate Update: As soon as a data item is modified in cache, the disk copy is updated.
  - Deferred Update: All modified data items in the cache is written either after a transaction ends its execution or after a fixed number of transactions have completed their execution.
- Based on updated version
  - Shadow update: The modified version of a data item does not overwrite its disk copy but is written at a separate disk location.
  - In-place update:The disk version of the data item is overwritten by the cache version

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 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 store (log 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 store.



STEAL/NO-STEAL AND FORCE/NO-FORCE

- Possible ways for flushing database cache to database disk:
  - Steal: Cache can be flushed before transaction commits.
  - No-Steal: Cache cannot be flushed before transaction commit.
  - Force: Cache is immediately flushed (forced) to disk.
  - No-Force: Cache is deferred until transaction commits.
- These give rise to four different ways for handling recovery:
  - Steal/No-Force (Undo/Redo), Steal/Force (Undo/No-redo), No-Steal/No-Force (Redo/No-undo) and No-Steal/Force (No-undo/No-redo).

STEAL/NO-STEAL AND FORCE/NO-FORCE

	Before Commit **Steal**	After commit **No-Steal**
Immediately **Force**	Undo/No-redo **(BFIM)**	~~No-undo/No-redo~~
Deferred **No-Force**	Undo/Redo **(BFIM)&(AFIM)**	No-undo/Redo **(AFIM)**

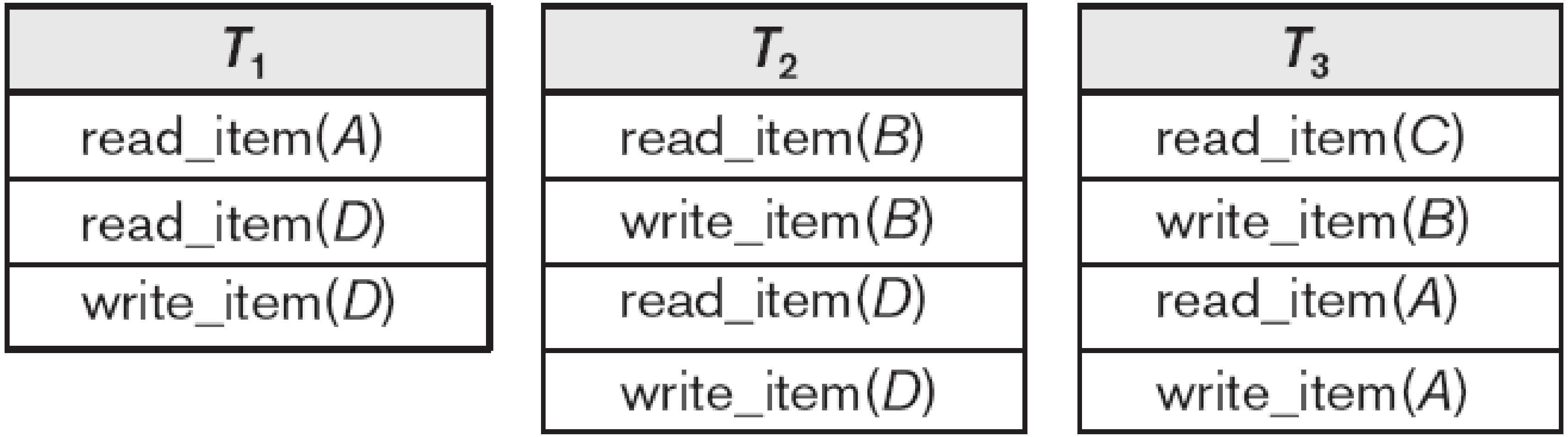
CHECKPOINTS IN THE SYSTEM LOG

- 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:
  - Suspend execution of transactions temporarily.
  - Force write modified buffer data to disk.
  - Write a [checkpoint] record to the log, save the log to disk.
  - Resume normal transaction execution.
- During recovery redo or undo is required to transactions appearing after [checkpoint] record.

TRANSACTION ROLLBACK AND CASCADING ROLLBACK

- If a transaction fails after updating the database, but before the transaction commits, 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 (BFIMs).
- The undo-type log entries are used to restore the old values of data items that must be rolled back.
- If a transaction T is rolled back, any transaction S that has 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 is called cascading rollback,

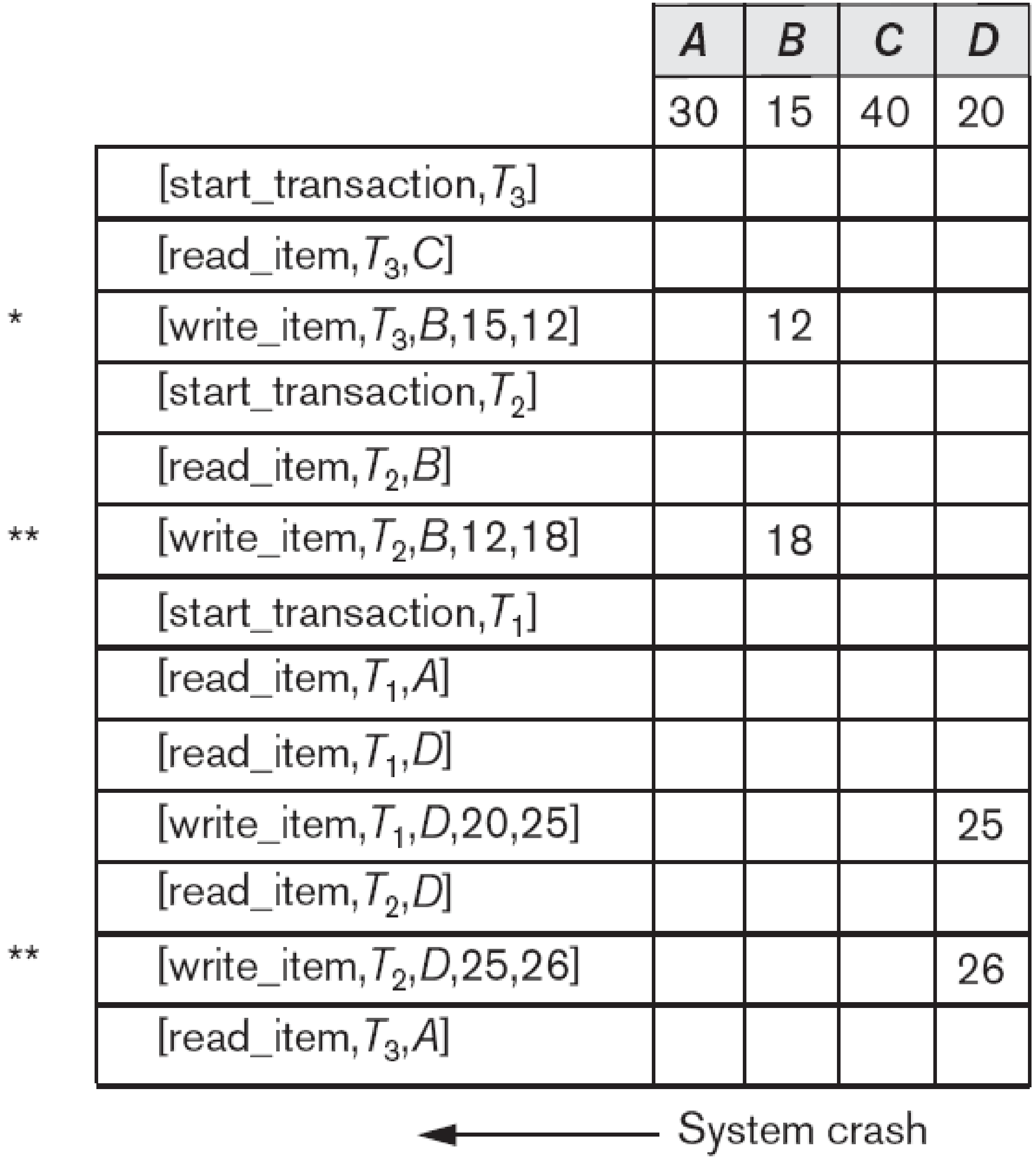
EXAMPLE



T_1
read_item(A)
read_item(D)
write_item(D)

T_2
read_item(B)
write_item(B)
read_item(D)
write_item(D)

T_3
read_item(C)
write_item(B)
read_item(A)
write_item(A)

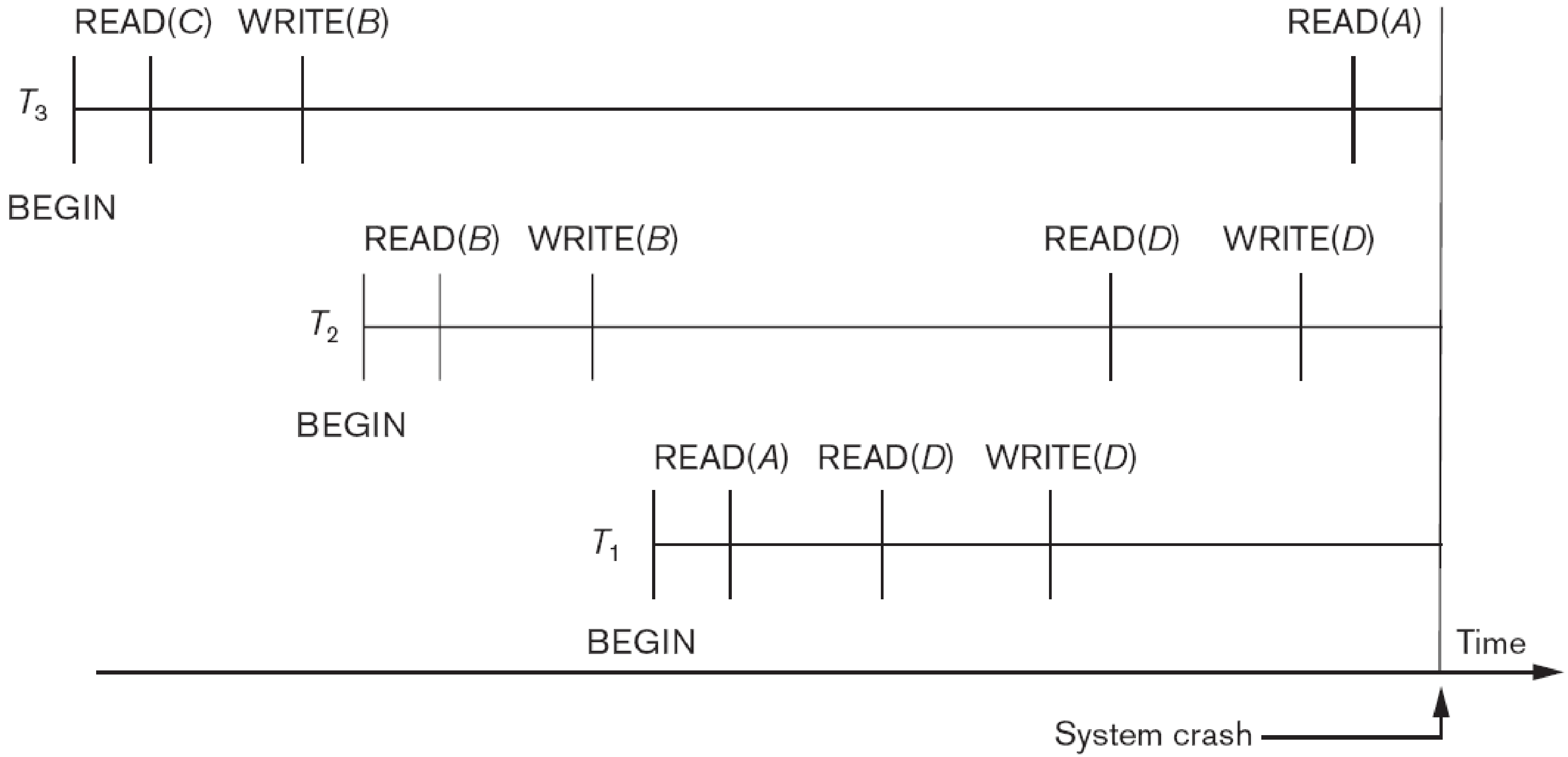


		A	B	C	D
		30	15	40	20
	[start_transaction,T_3]				
	[read_item,T_3,C]				
*	[write_item,T_3,B,15,12]		12		
	[start_transaction,T_2]				
	[read_item,T_2,B]				
**	[write_item,T_2,B,12,18]		18		
	[start_transaction,T_1]				
	[read_item,T_1,A]				
	[read_item,T_1,D]				
	[write_item,T_1,D,20,25]				25
	[read_item,T_2,D]				
**	[write_item,T_2,D,25,26]				26
	[read_item,T_3,A]				

← System crash

* $T3$ 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)
T_3
BEGIN
READ(B)
WRITE(B)
READ(D)
WRITE(D)
T_2
BEGIN
READ(A)
READ(D)
WRITE(D)
T_1
BEGIN
Time
System crash

NO-UNDO/REDO RECOVERY BASED ON DEFERRED UPDATE

- The data update goes as follows:
  - 1. A transaction cannot change the database on disk until it reaches its commit point.
  - 2. 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.
- After reboot from a failure the log is used to redo all the transactions affected by this failure. No undo is required because no AFIM is flushed to the disk before a transaction commits.

DEFERRED UPDATE IN A SINGLE-USER SYSTEM - EXAMPLE

- (a)

T1	T2
read_item (A)	read_item (B)
read_item (D)	write_item (B)
write_item (D)	read_item (D)
	write_item (A)

- (b)

  [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

The [write_item, …] operations of T1 are redone. T2 log entries are ignored by the recovery manager.

RECOVERY USING DEFERRED UPDATE IN A MULTIUSER ENVIRONMENT

- **Procedure RDU_M (NO-UNDO/REDO with checkpoints).**
  - Use two lists of transactions maintained by the system:
    - the committed transactions *T* since the last checkpoint (**commit list**), and
    - the active transactions *T* (**active 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 that are active and did not commit are effectively canceled and must be resubmitted.
- The REDO procedure is defined as follows:
- **Procedure REDO (WRITE_OP).** Redoing a write_item operation WRITE_OP consists of examining its log entry [write_item, *T*, *X*, new_value] and setting the value of item *X* in the database to new_value, which is the after image (AFIM).

RECOVERY USING DEFERRED UPDATE IN A MULTIUSER ENVIRONMENT - EXAMPLE

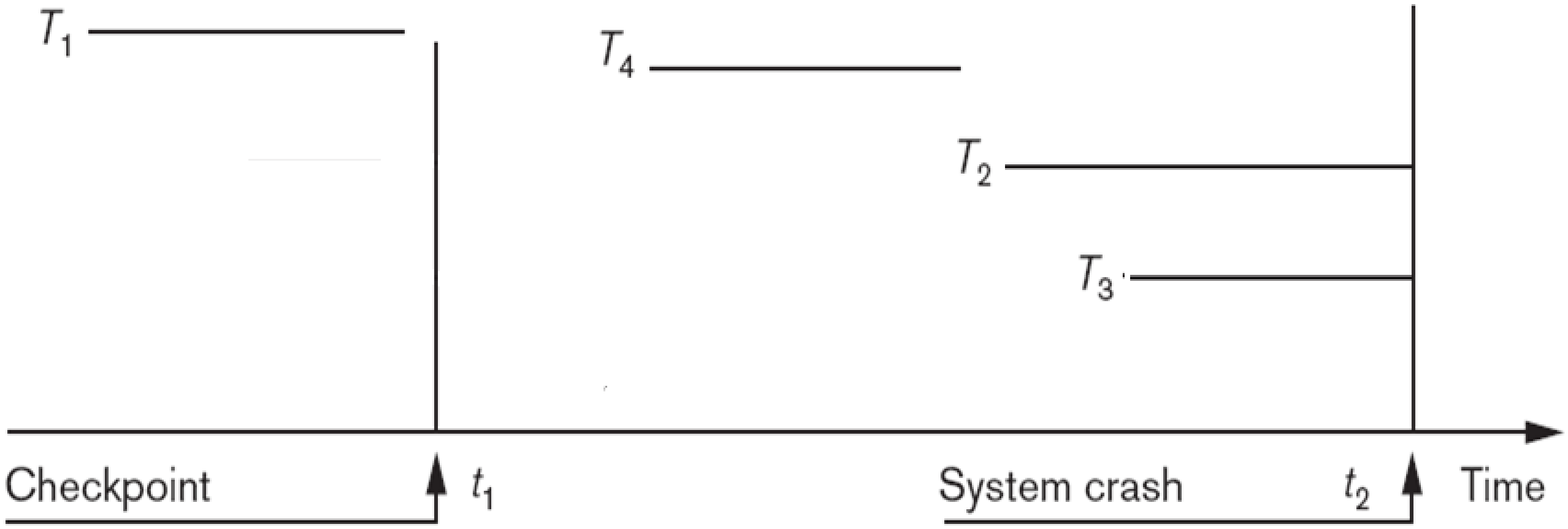
(a)

T1	T2	T3	T4
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, T1]
[write_item, T1, D, 20]
[commit, T1]
[checkpoint]
[start_transaction, T4]
[write_item, T4, B, 15]
[write_item, T4, A, 20]
[commit, T4]
[start_transaction T2]
[write_item, T2, B, 12]
[start_transaction, T3]
[write_item, T3, A, 30]
[write_item, T2, D, 25] ← system crash

T2 and T3 are ignored because they did not reach their commit points.
T4 is redone because its commit point is after the last checkpoint.

RECOVERY USING DEFERRED UPDATE IN A MULTIUSER ENVIRONMENT - EXAMPLE

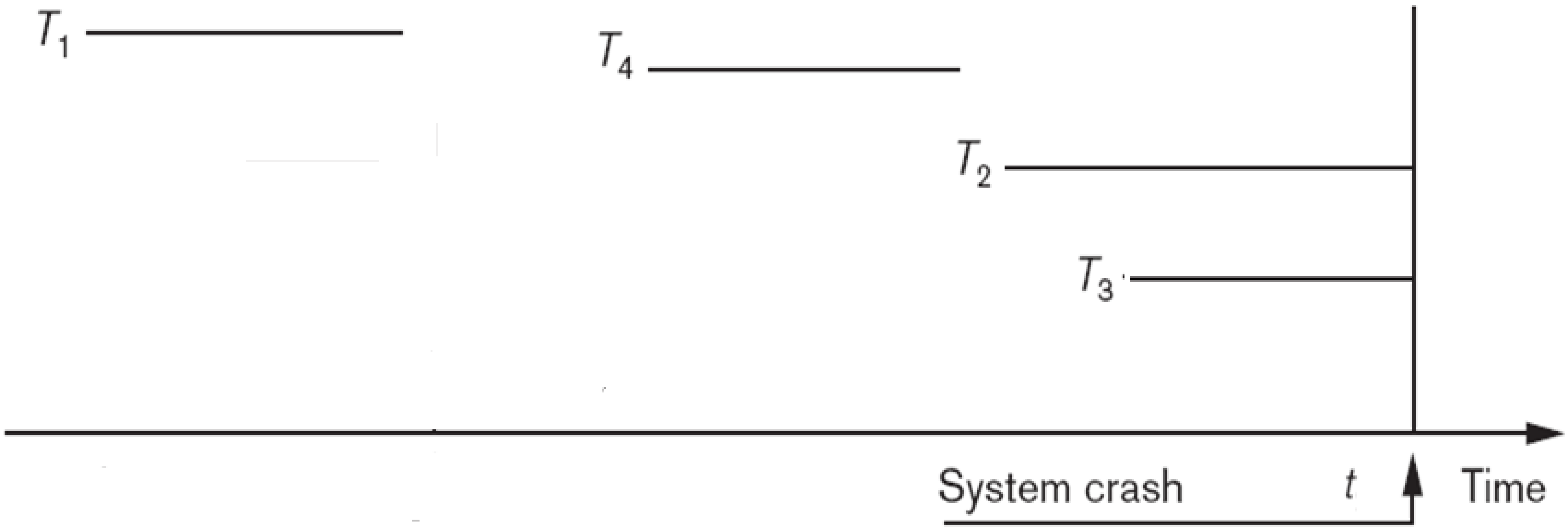


T2 and T3 are ignored because they did not reach their commit points.
T4 is redone because its commit point is after the last checkpoint.

RECOVERY TECHNIQUES BASED ON IMMEDIATE UPDATE

- **Undo/No-redo Algorithm**
- In this algorithm AFIMs of a transaction are flushed to the database disk under WAL before it commits. For this reason the recovery manager undoes all transactions during recovery. No transaction is redone.
- It is possible that a transaction might have completed execution and ready to commit but this transaction is also undone.
  - all updates by a transaction must be recorded on disk before the transaction commits, so that REDO is never needed.

RECOVERY TECHNIQUES BASED ON IMMEDIATE UPDATE



T1 and T4 are ignored because they reach their commit points.

T2 and T3 is undone because they did not reach their commit points.

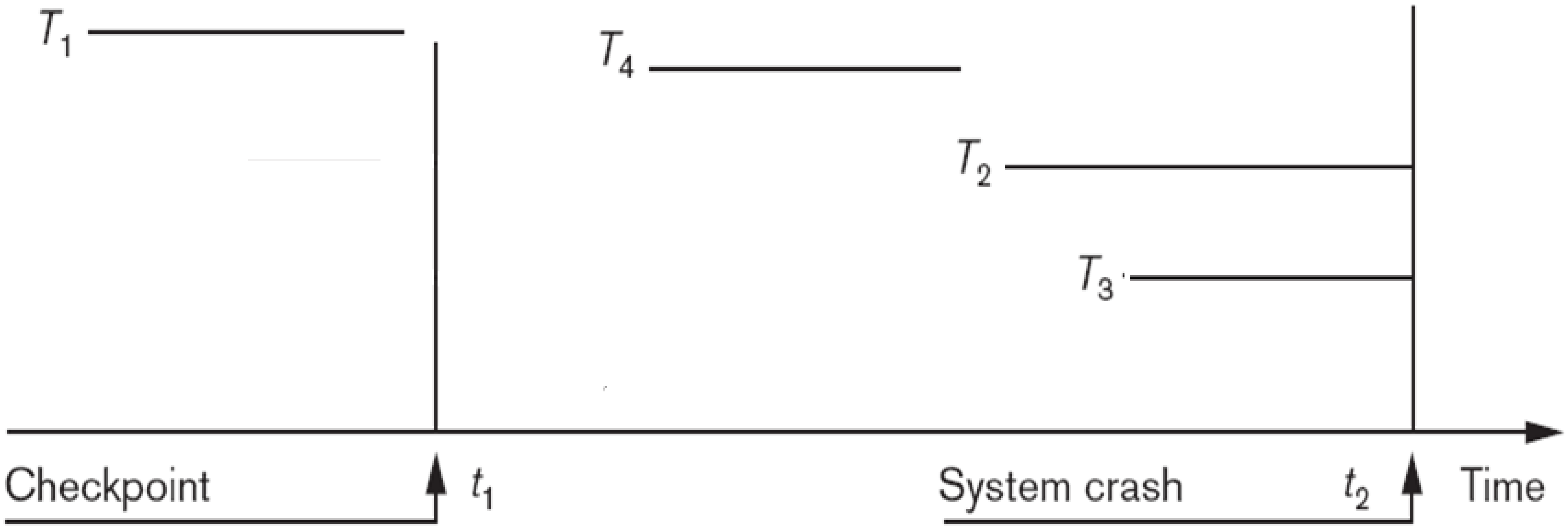
UNDO/REDO ALGORITHM (SINGLE-USER ENVIRONMENT)

- Recovery schemes of this category apply undo and also redo for recovery.
- 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:
- Undo of a transaction if it is in the active table.
- Redo of a transaction if it is in the commit table.

UNDO/REDO ALGORITHM (CONCURRENT EXECUTION)

- Recovery schemes of this category applies undo and also redo to recover the database from failure.
- In concurrent execution environment a concurrency control is required and log is maintained under WAL. Commit table records transactions to be committed and active table records active transactions.
- To minimize the work of the recovery manager check pointing is used. The recovery performs:
  - Undo of a transaction if it is in the active table.
  - Redo of a transaction if it is in the commit table.

UNDO/REDO ALGORITHM - EXAMPLE



T2 and T3 are UNDO because they did not reach their commit points.

T4 is redone because its commit point is after the last checkpoint.

Questions?