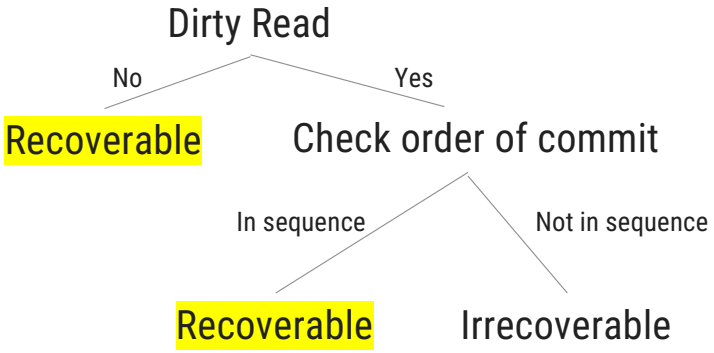
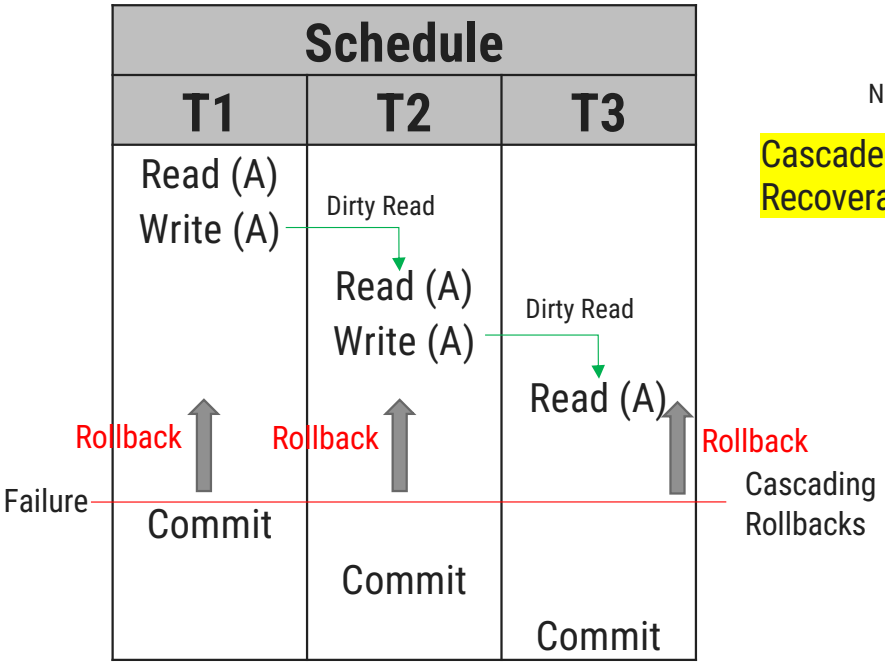
Recoverable Schedule

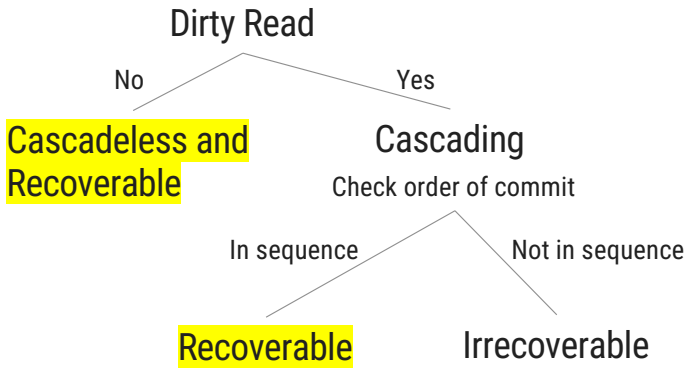
If a Schedule is conflict serializable or view serializable, there could be a possibility of inconsistency due to the failure during a transaction.

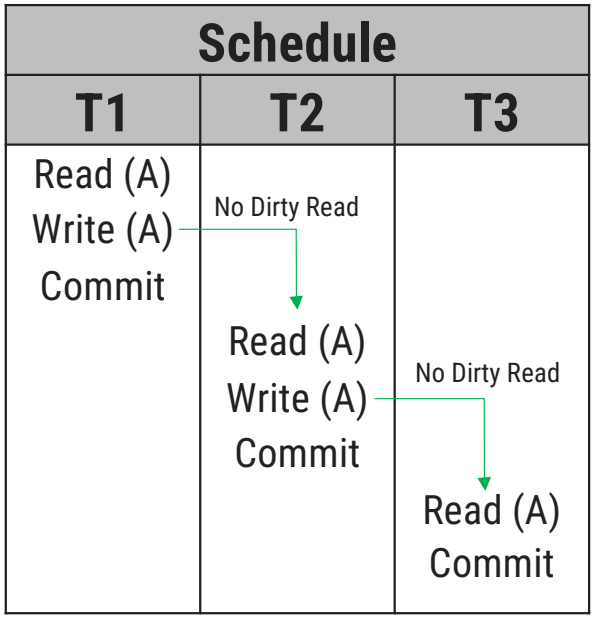
Schedule	
T1	**T2**
Read (A)	
A=A+10	
Write (A)	
	Read (A)
	A=A-5
	Write (A)
	Commit
Commit	



Cascading and Cascadeless Schedule







- It is a Recoverable Schedule, i.e., it can maintain consistency.
- But it leads to cascading rollbacks.
- Cascading rollbacks happen due to Dirty Read.

- This schedule does not have Dirty Read.
- It is free from cascading rollbacks.
- Such schedules are called Cascadeless Schedule

Database recovery

Database recovery

- There are many situations in which a transaction may not reach a commit or abort point.
  - Operating system crash
  - DBMS crash
  - System might lose power (power failure)
  - Disk may fail or other hardware may fail (disk/hardware failure)
  - Human error
- In any of above situations, data in the database may become inconsistent or lost.
- For example, if a transaction has completed 30 out of 40 write instructions to the database when the DBMS crashes, then the database may be in an inconsistent state as only part of the transaction's work was completed.
- Database recovery is the **process of restoring the database and the data to a consistent state**.
- This may include **restoring lost data up to the point of the event** (e.g. system crash).

Log based recovery method

- The log is a **sequence of log records, which maintains information about update activities on the database**.
- A log is **kept on stable storage (i.e HDD)**.
- Log contains
  - Start of transaction
  - Transaction-id
  - Record-id
  - Type of operation (insert, update, delete)
  - Old value, new value
  - End of transaction that is committed or aborted.

Log based recovery method

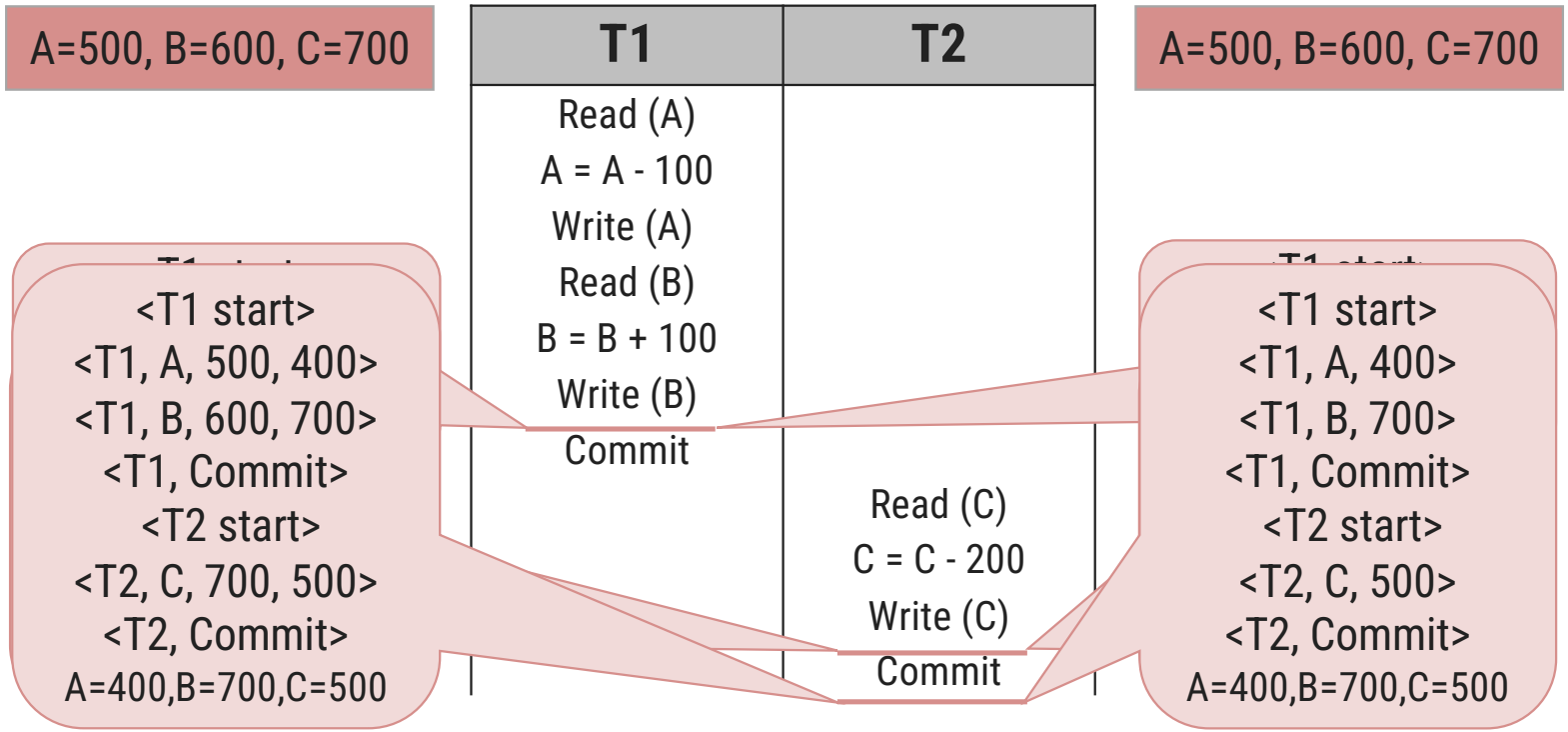
- When transaction **Ti starts**, it registers itself by writing a record **<Ti start>** to the log.
- Before **Ti executes write(X)**, a log record **<Ti, X, V1, V2>** is written, where V1 is the value of X before the write (the old value), and V2 is the value to be written to X (the new value).
- When **Ti finishes it last statement**, the log record **<Ti commit>** is written.
- **Undo** of a log record **<Ti, X, V1, V2>** writes the old value V1 to X
- **Redo** of a log record **<Ti, X, V1, V2>** writes the new value V2 to X
- Types of log based recovery method
  - Immediate database modification
  - Deferred database modification

Immediate v/s Deferred database modification

Immediate database modification	Deferred database modification
Updates (changes) to the database are **applied immediately** as they occur without waiting to reach to the commit point.	**Updates (changes)** to the database are **deferred (postponed)** until the transaction commits.

Immediate v/s Deferred database modification

Immediate database modification	Deferred database modification



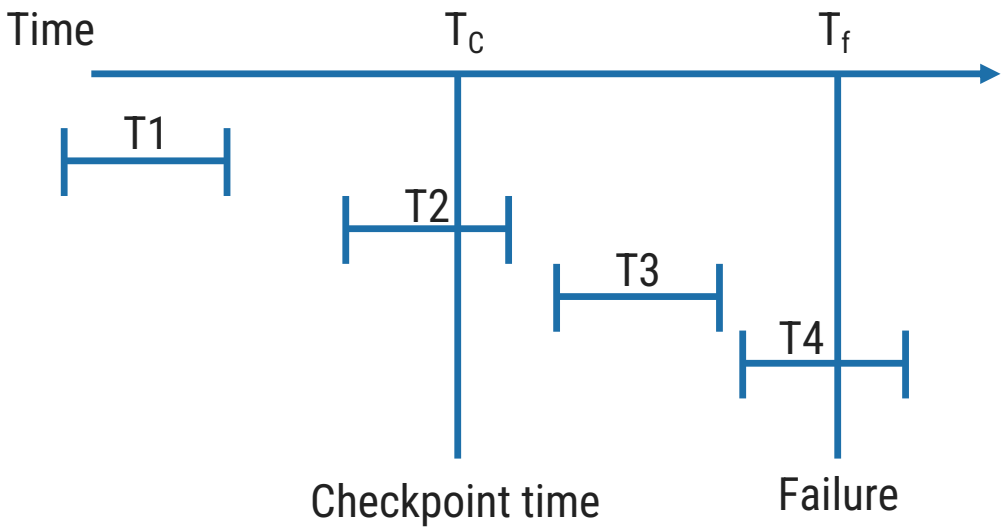
Immediate v/s Deferred database modification

Immediate database modification	Deferred database modification
Updates (changes) to the database are **applied immediately** as they occur without waiting to reach to the commit point.	**Updates (changes)** to the database are **deferred (postponed)** until the transaction commits.
If **transaction is not committed**, then we **need to do undo** operation and **restart the transaction** again.	If **transaction is not committed**, then **no need to do any undo** operations. Just **restart the transaction**.
If **transaction is committed**, then **no need to do redo** the updates of the transaction.	If **transaction is committed**, then we **need to do redo** the updates of the transaction.
Undo and Redo both operations are performed.	**Only Redo operation is performed**.

Problems with Deferred & Immediate Updates (Checkpoint)

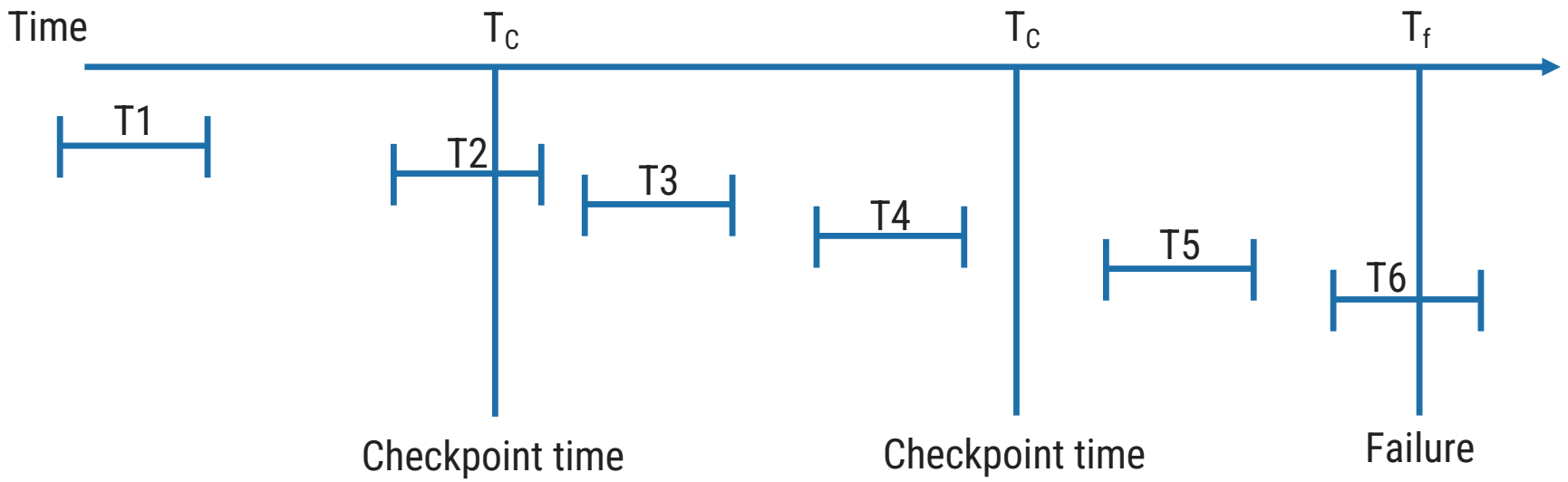
- Searching the entire log is time consuming.
  - Immediate database modification
    - When transaction fail log file is used to undo the updates of transaction.
  - Deferred database modification
    - When transaction commits log file is used to redo the updates of transaction.
- **To reduce the searching time** of entire log we can use **check point**.
- It is a **point** which specifies that **any operations executed before it are done correctly and stored safely** (updated safely in database).
- At this point, all the **buffers are force-fully written to the secondary storage** (database).
- Checkpoints are scheduled at predetermined time intervals.
- It is used to limit:
  - Size of transaction log file
  - Amount of searching

How the checkpoint works when failure occurs



- At failure time:
  - **Ignore the transaction T1** as it has already been committed before checkpoint.
  - **Redo transaction T2 and T3** as they are active after checkpoint and are committed before failure.
  - **Undo transaction T4** as it is active after checkpoint and has not committed.

How the checkpoint works when failure occurs



Exercise Give the name of transactions which has already been committed before checkpoint.

Exercise Give the name of transactions which will perform Redo operation.

Exercise Give the name of transactions which will perform Undo operation.