12/5(화) A형독감 유고결석

평균

교수2명충원안되면...

Crash Recovery

Review: The ACID properties

- **A tomicity**: All actions in the transaction happen, or none happen.
- **C onsistency**: If each transaction is consistent, and the DB starts consistent, it ends up consistent.
- **I solation**: Execution of one transaction is isolated from that of other transactions.
- **D urability**: If a transaction commits, its effects persist.

- The **Recovery Manager** guarantees Atomicity & Durability.

Isolation level

- Isolation level of a transaction
  - Can be set in SET TRANSACTION statement

  - *Serializable*: default in SQL standard.
  - *Repeatable Read*: prevents non-repeatable read.
  - *Read Committed*: default in Oracle DBMS.
    - See changes only committed by another transactions.
    - Prevents dirty-read anomaly.
  - *Read Uncommitted*:
    - See changes incurred by any (including uncommitted) transactions.

Isolation level Cont'd

- Serializable

commit;

T1

```
SET TRANSACTION
    ISOLATION LEVEL
        SERIALIZABLE;
select pid from prod;

select count(*) from prod;
commit;
```

T2

```
commit;
insert into prod (2,'Galaxy_Y',100, 'Samsung');
commit;
```

- If DBMS supports the serializability, then the result should be the same as either T1 and T2, or T2 and T1.
- Lock the proj table in S mode so that any write operation to the proj table is not allowed. => prevent *phantom*.

Isolation level Cont'd

- Serializable cont'd

commit;

T1

```
SET TRANSACTION
    ISOLATION LEVEL
    SERIALIZABLE;
select count(*) from prod;

select count(*) from prod;
commit;
```

T2

```
commit;
insert into prod (2,'Item_Y', 300, 'Samsung');
delete from prod where pid=3 or 2;
commit;
```

- If DBMS supports the serializability, then the result should be the same as either T1 and T2, or T2 and T1.
- Lock the proj table in S mode so that any write operation to the proj table is not allowed. => prevent *phantom*.

Isolation level Cont'd

- Repeatable Read

commit;

T1

select price from prod where pid=2;

select price from prod where pid=2;

commit;

commit;

T2

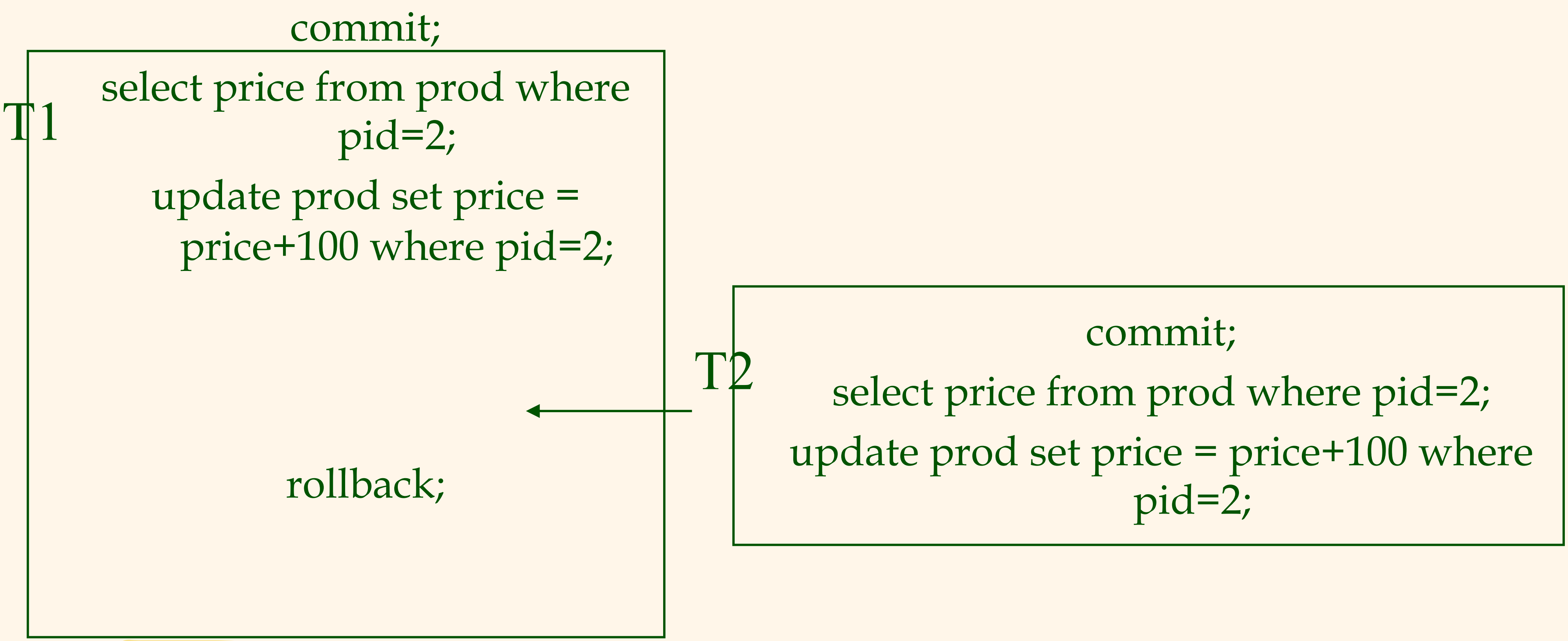
update prod set price = 200 where pid=2;

commit;

  - Non repeatable read : the budget value of the first select is different to the budget value of the second select.
- If the isolation level is set to REPEATABLE READ, then two budget values are the same. That's why it is called *repeatable read*.

Isolation level Cont'd

- Read Committed

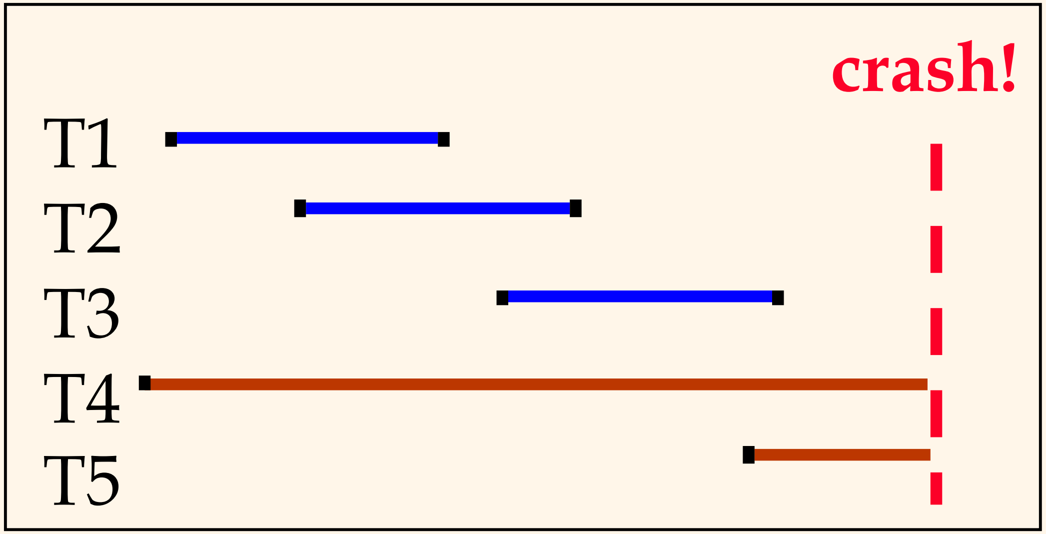


- Dirty read : T2 read the budget written by T1 which has not committed. => Not desired, No practical at all.

Motivation

- Atomicity:
  - Transactions may abort ("Rollback").
- Durability:
  - What if DBMS stops running? (Causes?)

- Desired Behavior after system restarts:
  - T1, T2 & T3 should be durable.
  - T4 & T5 should be aborted (effects not seen).



Recovery Strategy

디스크 업데이트는 커밋되면

- Deferred update
  - Do not physically update the database in disk until the transaction commit.
  - During the commit, the updates are first recorded persistently in the log and then written to the database.
  - No-UNDO/REDO

DB 업데이트는 W하면

- Immediate update
  - the database may be "updated" immediately after the write operations although the transaction does not reach commit.
  - What's happening in "updating" ? See the following slide.

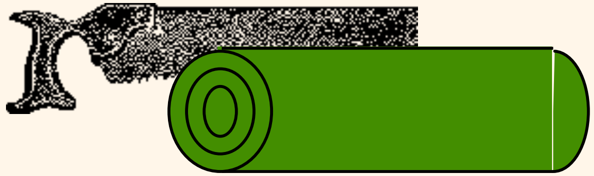
Handling the Buffer Pool

→ force stealing 정책!

- Force committed all updates to disk when the transaction commits.
  - Poor response time.
  - Providing durability is not hard.
- Steal buffer frames from uncommitted transactions.
  - If not, poor throughput.
  - Providing atomicity is not easy.

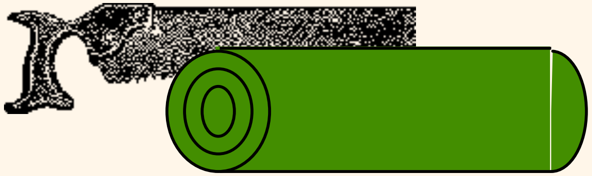
	No Steal	Steal
Force	Trivial	
No Force		Desired/ in Practice

Basic Idea: *Logging*



- REDO : It may be necessary to “redo” the operations of a committed transaction
  - Need new values.
- UNDO : We need “undo” the operations of a failed transaction or of another uncommitted transaction which reads the dirty data.
  - Need old values.

Basic Idea: Logging



- Record REDO and UNDO information, for every update, in a *log*.
  - Sequential writes to log (put it on a separate disk).
  - Minimal info (diff) written to log, so multiple updates fit in a single log page.
- Log: An ordered list of REDO/UNDO actions
  - Log record contains:

    <transactionID, data_item, old value, new value>

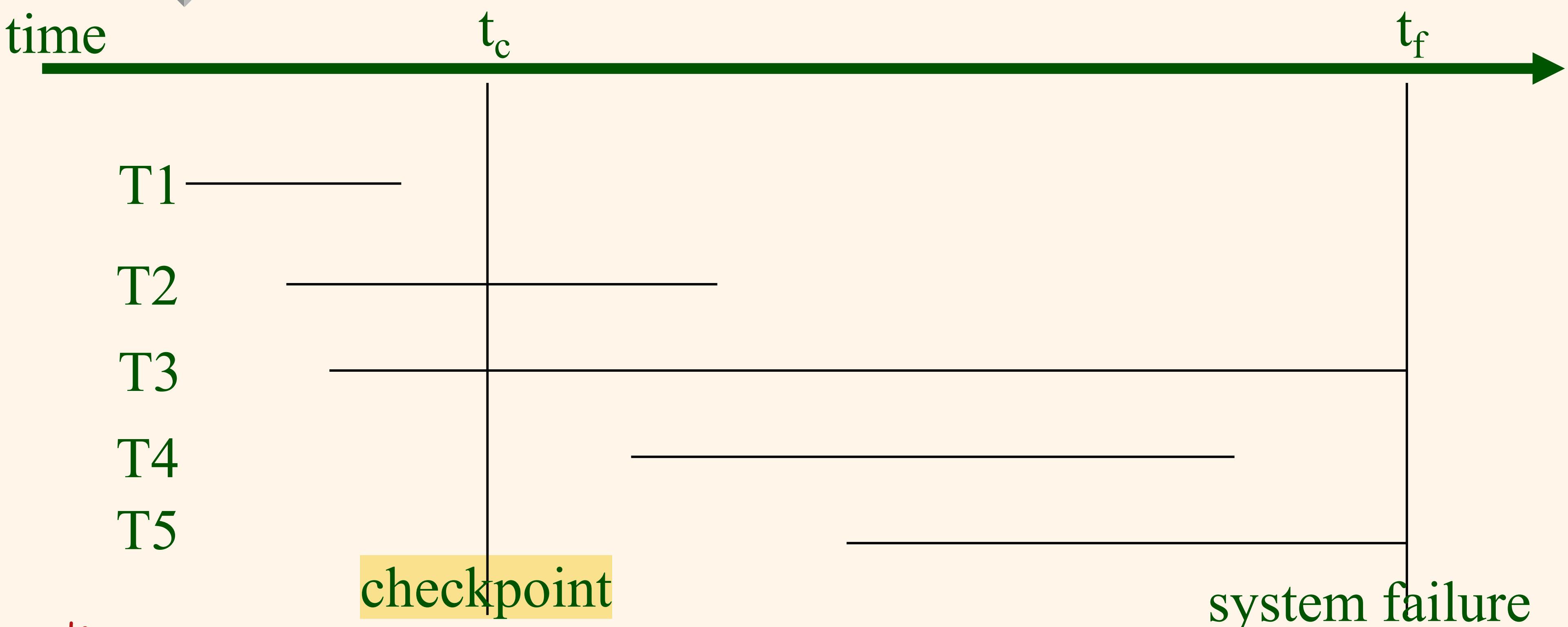
    and other info such as begin, and commit/rollback.
  - Can vary according to the recovery scheme.

ㅏ 12/7

Write-Ahead Logging (WAL)

- The Write-Ahead Logging Protocol:
  - ① Must force the log record for an update *before* the corresponding data page gets to disk.
  - ② Must write all log records for a transaction *before* *commit*.
- #1 guarantees Atomicity.
- #2 guarantees Durability.
- Exactly how is logging (and recovery!) done?
  - Like **ARIES**(Algorithm for Recovery and Isolation Exploiting Semantics) by IBM Almaden Research.
  - IBM DB2, Informix, MS SQL Server, Oracle 8, Sybase uses ARIES or its variant.

Big Picture



다음페이지!

- T3 and T5 must be undone. Why? → commit 되지 않음 system failure (atomicity를위해)
- T2 and T4 must be redone. Why? → checkpoint 이후에 commit, 제대로했다는 보장이없어서 redo함
- How about T1? → redo 필요x, checkpoint 이전에 commit

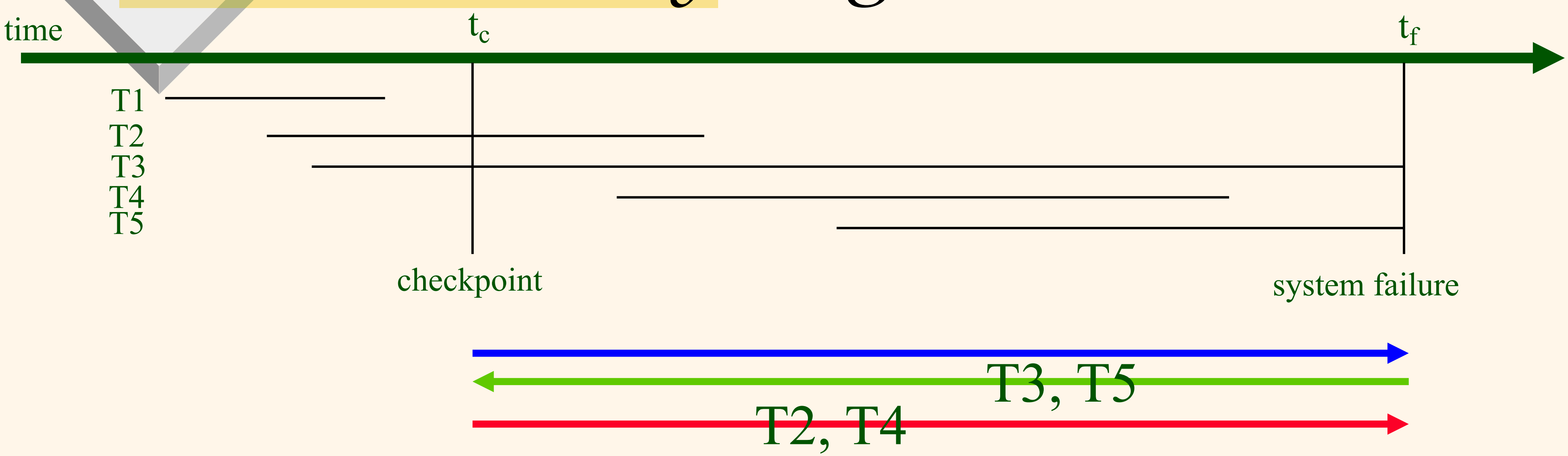
Big Picture

- T3 and T5 must be undone. Why?
  - Any change that was made by uncommitted transaction must be undone.
- T2 and T4 must be redone. Why?
  - No-force. In other words, there is no guarantee that their updates were actually written to the database.
- How about T1?
  - Updates were forced out to the database at t_c when the checkpoint was taken.

Checkpointing

- Periodically, the DBMS creates a checkpoint, in order to minimize the time taken to recover in the event of a system crash. The checkpoint may involve in general :
  - Forcing a “checkpoint record” out to the log storage.
  - Forcing the content of database buffers out to the database.
  - Writing the the address of the checkpoint record within the log into a “master record”. ex) OS에서 첫부팅때 boot record
- No work done prior to the checkpoint need ever be redone.

Crash Recovery: Big Picture

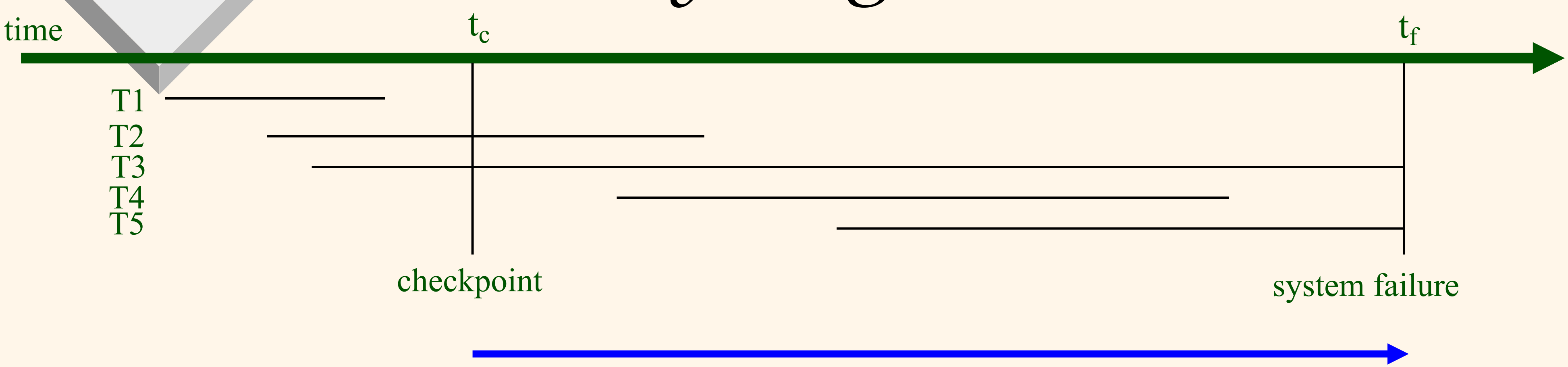


- Start from a checkpoint (found via master record).
- Three phases. Need to:
  - ① Figure out which Xacts committed since checkpoint, which failed (**Analysis**). 순방향으로 가면서 undo/redo 대상분류
  - ② **UNDO** effects of failed Xacts 역방향으로 가며 undo
  - ③ **REDO** *all* actions 순방향으로 가면서 redo

※ 반드시 undo 먼저 하지 않아도 결과는 같음
sequencial access : → 효율적
random access가 된다면 시간낭비가 없음

recursive하게 여러번 해도 같은결과를 내야함.

Crash Recovery: Big Picture

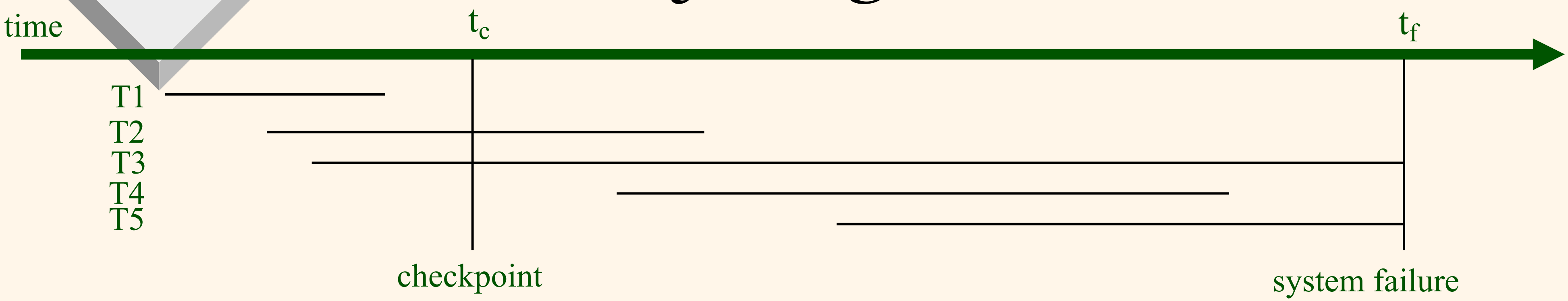


UNDO-list : {T2,T3} {T2,T3,T4} {T3,T4,T5} {T3,T5}

REDO-list : {} {T2} {T2,T4}

- Analysis Phase
  - Initializes UNDO-list to have all transactions listed in the checkpoint record. Initializes REDO-list empty.
  - If finding a BEGIN TRANSACTION record for a transaction, it adds that transaction to the UNDO-list.
  - If finding a COMMIT TRANSACTION record for a transaction, it moves that transaction from UNDO-list to the UNDO-list.

Crash Recovery: Big Picture



UNDO-list : {T3,T5}

REDO-list : {T2,T4}

- **Undo Phase**
  - Undoing the transactions in the undo-list
- **Redo Phase**
  - Redoing the transactions in the redo-list

Exactly how is logging (and recovery!) done?

- Like **ARIES**(Algorithm for Recovery and Isolation Exploiting Semantics) by IBM Almaden Research.
- IBM DB2, Informix, MS SQL Server, Oracle 8, Sybase uses ARIES or similar to its variant.
- Should I know ARIES Algorithm??????

기록에제대로 남겨야!

Summary of Logging/Recovery

- Recovery Manager guarantees Atomicity & Durability.
- Use WAL to allow STEAL/NO-FORCE w/o sacrificing correctness.
- Checkpointing: A quick way to limit the amount of log to scan on recovery.
- Recovery works in 3 phases:
  - Analysis:
  - Redo:
  - Undo:

concerency control & recovery

scheduling → locking

atomicity & durability → logging

기말

12/7