Recovery

SWEN 304

Trimester 2, 2017

Lecturer: Dr Hui Ma

Engineering and Computer Science

TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

Slides by: Pavle Morgan & Hui Ma

Plan For The Recovery Topic

- Transaction Log File
- Classification of database recovery procedures
  - Deferred database update
  - Immediate database update
- Checkpoints
  - *Readings:*
    - *Chapter 23 of the textbook*

Purpose of Database Recovery

- To bring the database into the last consistent state, which existed prior to the failure
- To preserve transaction properties (**A**tomicity, **C**onsistency, **I**solation and **D**urability)
- Example:
  - If 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

Transaction Log

- To be able to recover from failures DBMS maintains a log file
- For recovery from any type of failure data values prior to modification (BFIM - BeFore IMage) and the new value after modification (AFIM – AFter IMage) are required
- Typically, a log file contains records with the following contents:

  [start_transaction, *T*] (* *T* is a transaction id*)

  [write_item, *T*, *X*, old_value, new_value]

  [read_item, *T*, *X*] (*optional*)

  [commit, *T*]

  [abort, *T*]

Database Recovery

Transaction Roll-back (Undo) and Roll-Forward (Redo)

- To maintain atomicity, a transaction’s operations are **redone** or **undone**
  - **Undo**: Restore all BFIMs on to disk (Remove all AFIMs)
  - **Redo**: Restore all AFIMs on to disk
- Database recovery is achieved either by performing only Undos or only Redos or by a combination of the two
- These operations are recorded in the log as they happen

Classification of database recovery procedures

- According to the type of a failure, recovery procedures are classified as:
  - Recovery from a catastrophic failure (like disk crash), and
  - Recovery from a noncatastrophic failure

- Recovery from a catastrophic failure is based on restoring a database back_up copy by redoing operations of committed transactions (stored in archived **log** files) up to the time of the failure

Noncatastrophic Failures

- A computer failure (system crash):
  - E.g. a hardware, software or network error occurs in the computer system
- A transaction or system error:
  - E.g. integer overflow, division by zero, logical error, user interruption
- Local errors or exception conditions detected by transactions
  - E.g. Data not found, exception condition
- Concurrency control enforcement:
  - E.g. violate serializability, deadlock

Classification (continued)

- If a database becomes inconsistent due to a noncatastrophic failure, the strategy is to reverse only those changes that made database inconsistent
- It is accomplished by undoing (and sometimes also redoing) some operations, with the use of an in memory log file
- From now on we consider only recovery from non disk crash failures (we suppose data on disk are safe)
- The recovery from noncatastrophic failures can be based on many algorithms, as:
    - Deferred update,
    - Immediate update, and
    - Shadow update (not discussed)

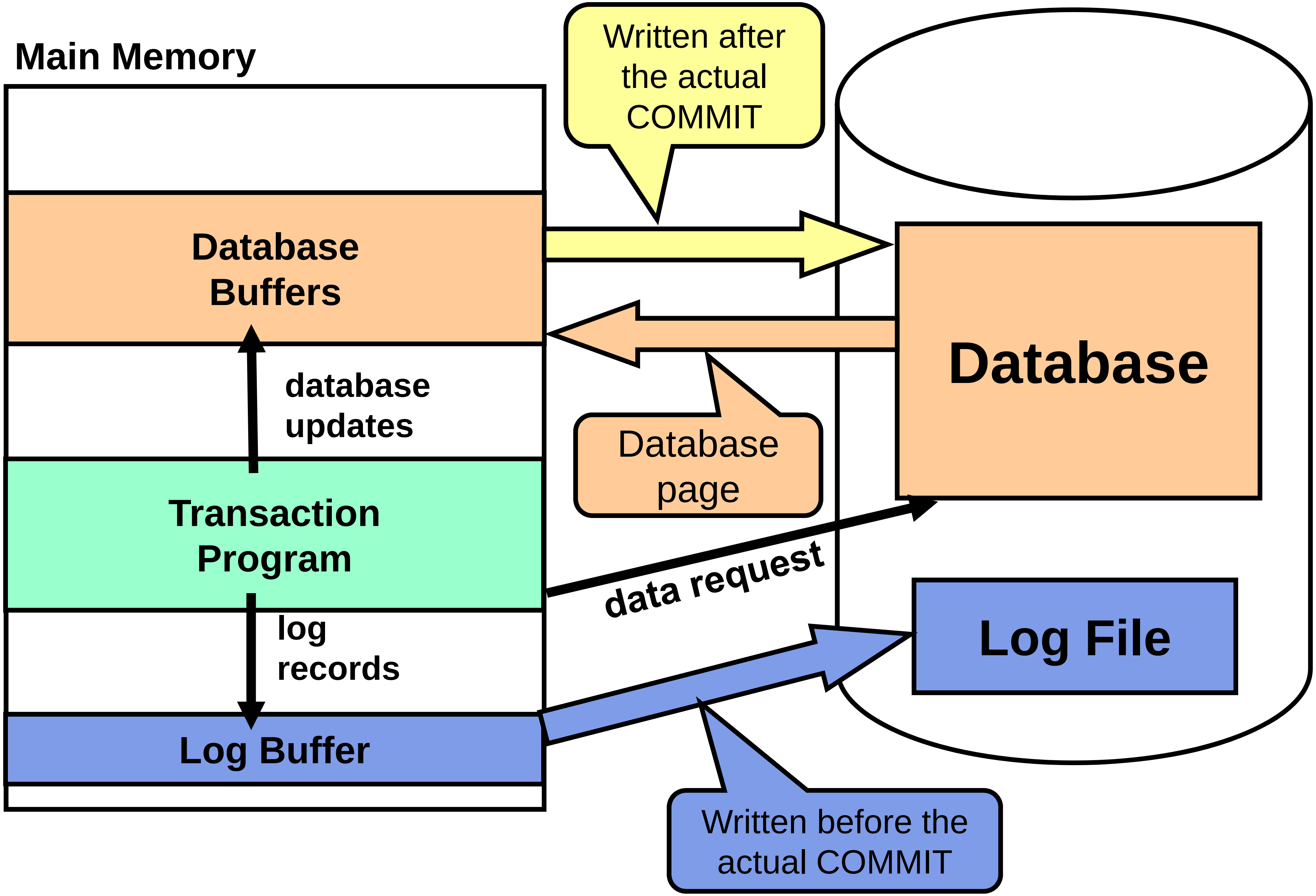
Database Recovery

- **Data Update**
  - **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
  - **Immediate Update**: As soon as a data item is modified in cache, the disk copy is updated
  - **Shadow update**: The modified version of a data item does not overwrite its disk copy but is written at a separate disk location

Deferred Update

- The idea:
  - Postpone updates to the database until a transaction reaches its **commit** command
- Updates are recorded in a log file and in cache buffers with database pages (all in RAM)
- When the COMMIT is reached, before it is executed all log updates are first force written to the **log file** on disk, and then the transaction commits
- After that, corresponding updates are written from buffers to the database

Deferred Update Layout



Deferred Update (continued)

- If a transaction fails before reaching COMMIT, there is no need to make any recovery
- If a system crash occurs after COMMIT, but before all changes are recorded in the database on disk, **Redo** of operations is needed,
  - The operations have to be redone from the log file (that is already on disk) to the database
  - Using after images, AFIMs (item values intended to be written to the database) to perform redo
  - Deferred update recovery log file has to contain only after images - the **new** database item values

Deferred Update (An Example)

Operations of T_1	**Generated log records**
start_T_1	begin,< T_1 >
read_item(A) //A = 4000	
$A = A - 1000$	
write_item(A)	< T_1 ,write_item(A), 3000 >
read_item(B) //B = 0	
$B = B + 1000$	
write_item(B)	< T_1 ,write_item(B), 1000 >
read_item(A) // A = 3000	
$A = A - 1000$	
write_item(A)	< T_1 ,write_item(A), 2000 >
read_item(C) //C = 0	
$C = C + 1000$	
write_item(C)	< T_1 ,write_item(C), 1000 >
commit T_1	
	//force write to the log on disk
	commit, < T_1 > // **actual**

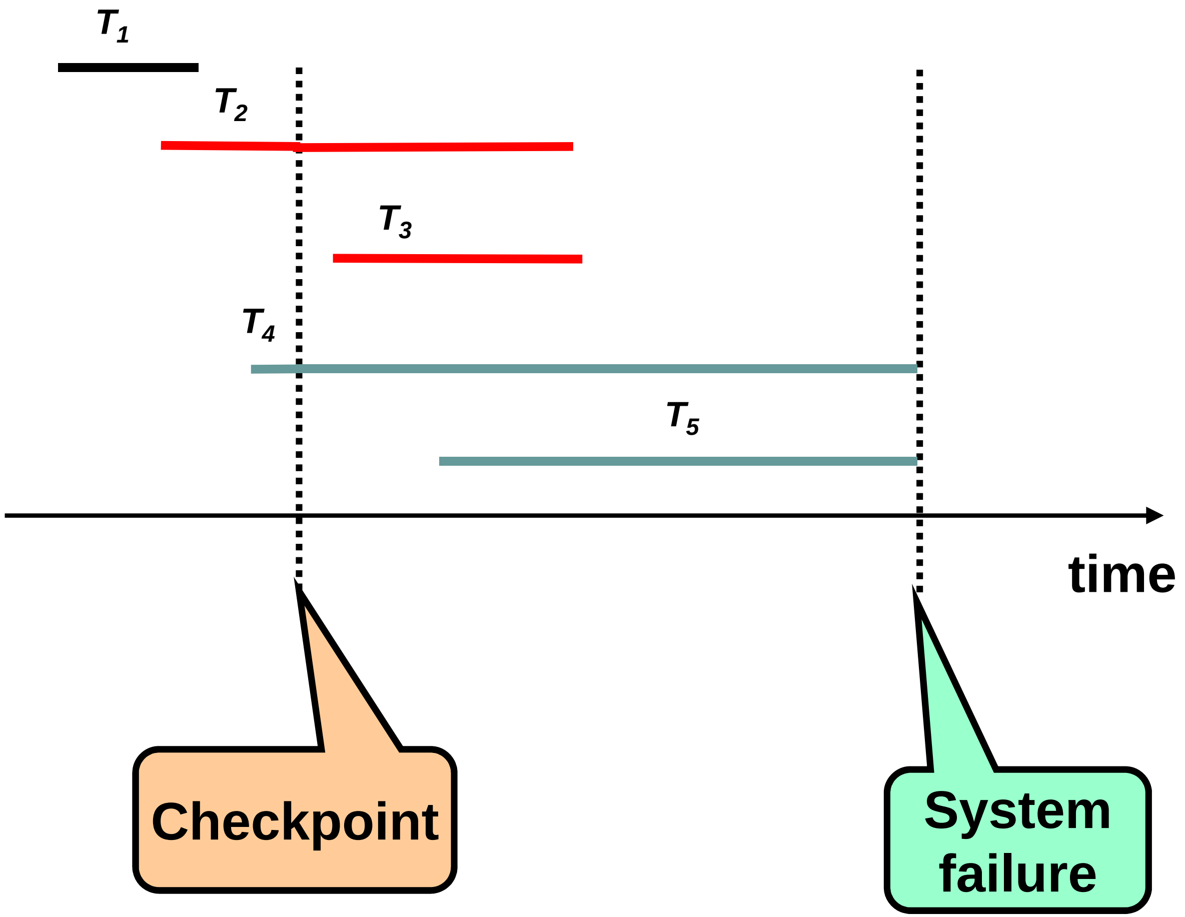
Deferred Update

- To finish a transaction, DBMS has force stored log records on disk
- If the system fails after that point, DBMS will use the log records to REDO changes in the database
- It is sufficient to redo only the **last** written value (after image) of every item changed
- So, redoing starts from the end of the log file and maintains a list of already redone database items

Checkpoints

- Some DBMS use CHECKPOINT records in the log file to prevent unnecessary redo operations
- To take a checkpoint record, DBMS has:
  - To suspend temporarily operations of all transactions,
  - To force write results of all update operations of committed transactions from main memory buffers to disk,
  - Write a checkpoint record into the log file and write log to disk
  - Resume executing transactions
- Only changes made by transactions committed between the last checkpoint and a system failure have to be redone

Redo With Checkpoint



Changes made by T_1 are stored in the database, so there is nothing to do with them

DBMS will REDO transactions T_2 and T_3 (their logs are already on disk)

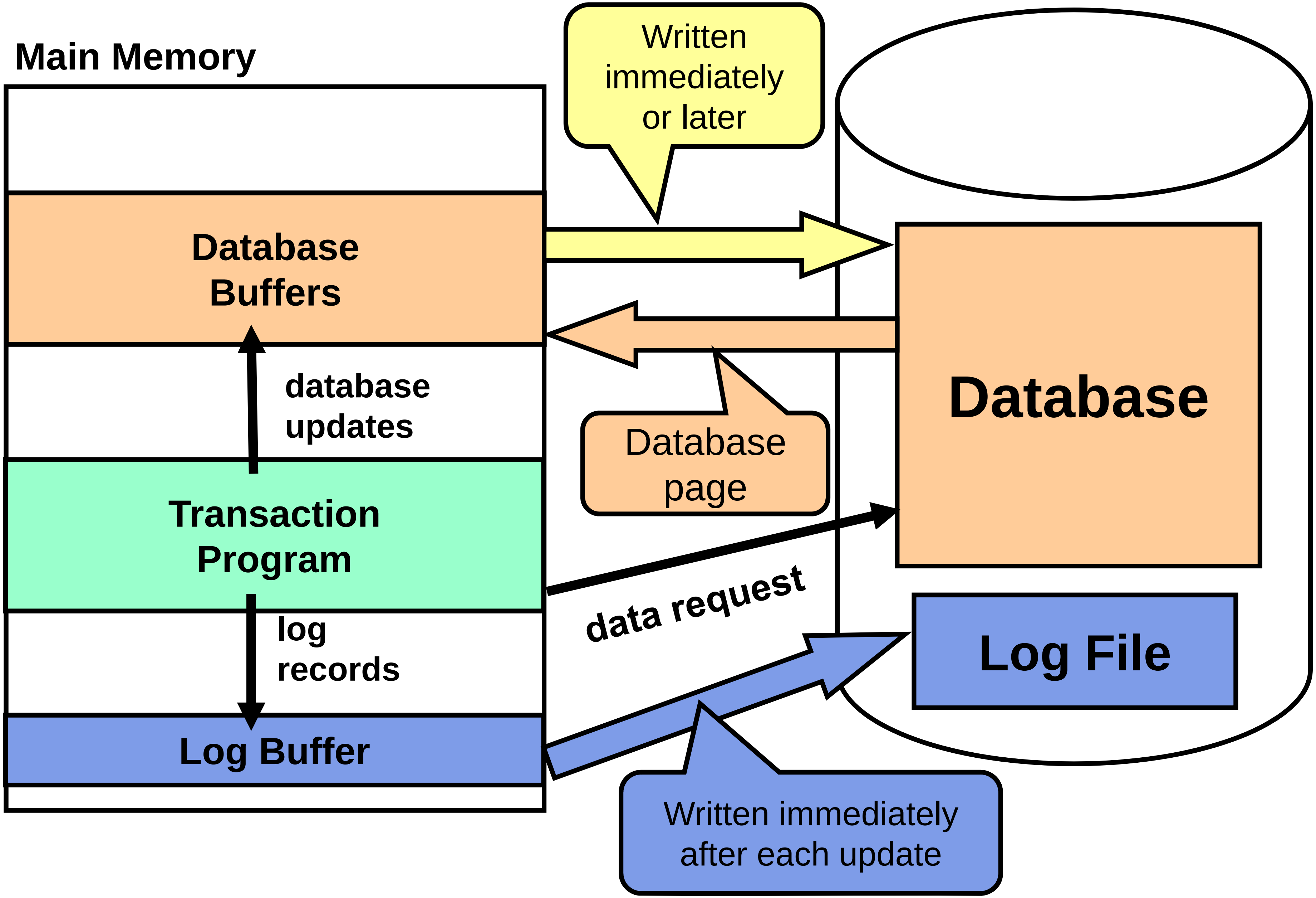
DBMS or the user has to rerun T_4 and T_5

Immediate Update

- The main idea:
  - When a transaction issues an update command, before and after images are recorded into a log file on **disk**, and (thereupon) the database **can** (but do not have to) be **immediately** updated

- There are two versions of immediate update algorithm
  - One writes all updates from buffers into the database before the transaction actually commits
  - The other one allows a transaction to commit before all its updates have been written in the database
  - The second version is the most common one

Immediate Update Layout



Main Memory

Database Buffers

database updates

Transaction Program

log records

Log Buffer

Written immediately or later

Database page

data request

Written immediately after each update

Database

Log File

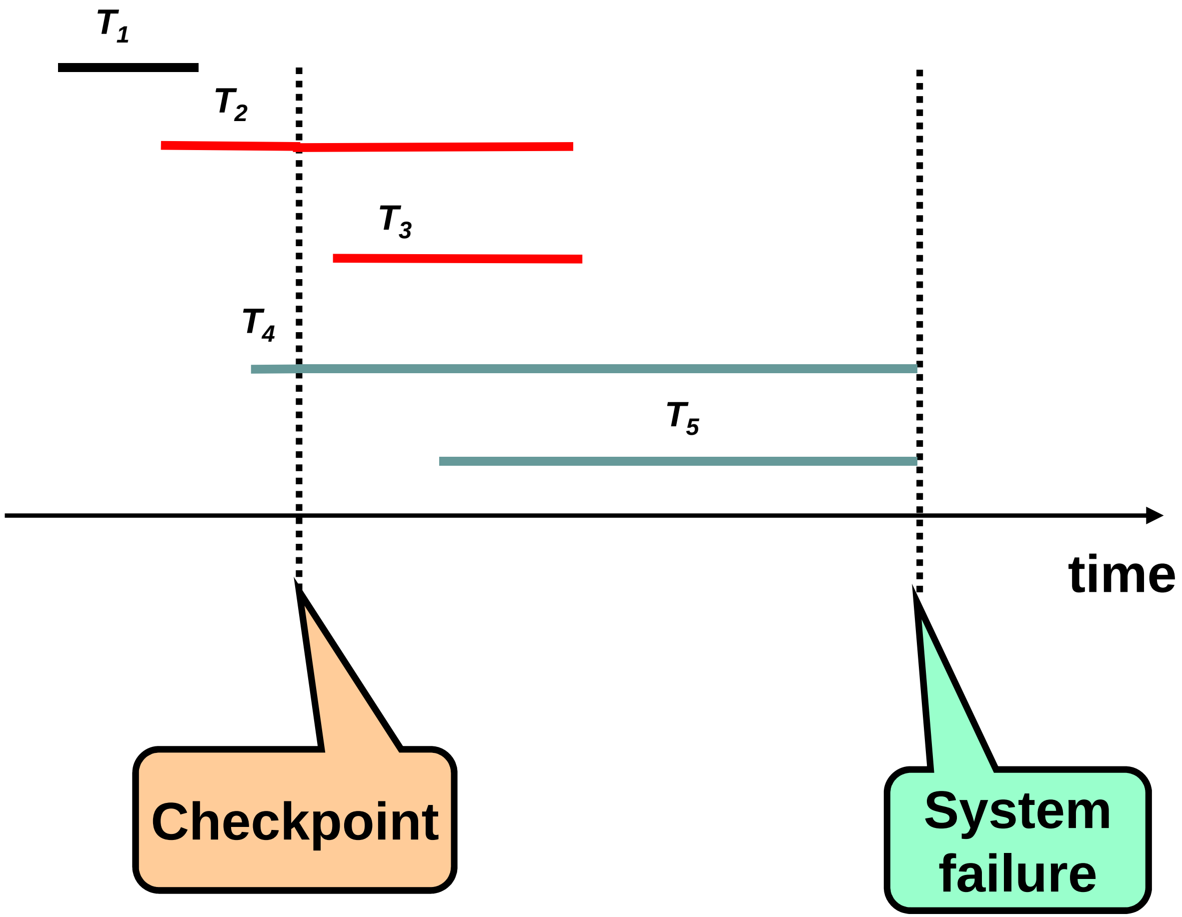
Transaction Roll–Back (Undo)

- If a transaction fails for any reason, its effects are rolled-back
- **Roll-back** of a transaction is a procedure of undoing changes made against the database by a non-committed transaction
- A roll-back is done by restoring the **before images** (old values) of each database item changed by the transaction
- The roll-back is done in the **reverse order** of the order the operations were written into the log file on disk

Redo in Immediate Update

- In the case of a system crash:
  - Because there can be some updates of the committed transactions that are not written into the database, effects of the committed transactions are redone to the database
    - Recall, we allow a transaction to commit before all of its updates have been written into the database
  - Only the last updates of each item have to be redone

Immediate Update



Changes made by T_1 are stored in the database, so there is nothing to do with them

DBMS will REDO transactions T_2 and T_3 (their logs are already on disk)

DBMS has to UNDO T_4 and T_5

A Question for You

- Immediate update with roll-back requires redoing transactions that committed between the last checkpoint and the moment of system crash
- What if the changes are already made against the database?

  a) Redoing will bring the database into an inconsistent state

  b) This question is too complex, and can not be answered without a careful analysis

  c) Redoing will make no harm to the database, because it will bring the database in the same consistent state as it was in

Summary

- *Recovery* from a *catastrophic* failure is made by applying operations of committed transactions from archived log files on an archived database back up copy
- Recovery from a *noncatastrophic* failure can be accomplished through:
  - Deferred update,
  - Immediate update, and
  - Other recovery schemes
- *Deferred update* means writing changes into a database after a transaction commits, but the log is written to disk just before the transaction actually commits
- *Immediate update* means that changes are immediately stored in a log file on disk. So, changes can be written into a database before or after a transaction commits