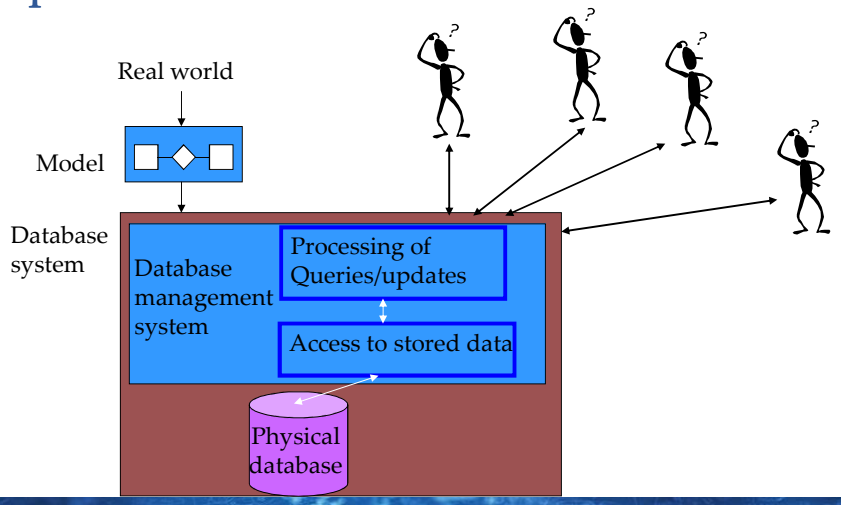
Lecture 10: Database recovery

Jose M. Peña
jose.m.pena@liu.se

How can several users access and update the database at the same time ?



Concurrent processing

- Single user system: At most one user can use the system at each point in time.
- Multiple user system: Several users can use the system at the same time.
  - Multiple CPU: Parallel processing.
  - One CPU: Concurrent processing, interleaving.
- Hereinafter, we focus on *multiple user systems with just one CPU*.

Transactions: Definition

- A *transaction* is a logical unit of database processing and consists of one or several operations.
- Simplified database operations in a transaction:
  - Read-item(X)
  - Write-item(X)

Properties for transactions

ACID: *Atomicity*, Consistency preservation, Isolation, *Durability*

- A: A transaction is an *atomic* unit: It is either executed completely or not at all.
- C: A database that is in a *consistent* state before the execution of a transaction (i.e. it fulfills the conditions in the relational model and any other condition declared for the database), is also in a *consistent* state after the execution of the transaction.
- I: A transaction should act as if it is executed *isolated* from the other transactions.
- D: Changes in the database made by a *committed* transaction are *permanent*.

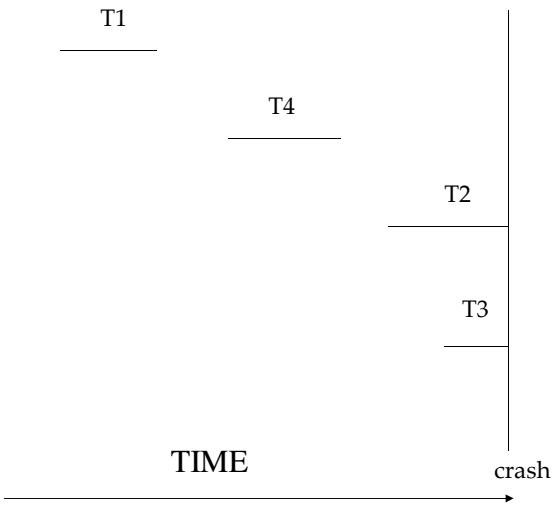
Properties for transactions

- Who ensures that the ACID property are satisfied ?
  - *Atomicity: Recovery system.*
  - Consistency preservation: Programmer + DBMS.
  - Isolation: Concurrency contol.
  - *Durability: Recovery system.*

Recovery: Example

System log

start-transaction T1
write-item T1, D, 10, 20
commit T1
start-transaction T4
write-item T4, B, 10, 20
write-item T4, A, 5, 10
commit T4
start-transaction T2
write-item T2, B, 20, 15
start-transaction T3
write-item T3, A, 10, 30
write-item T2, D, 20, 25
CRASH



Reasons for a crash

1. System crash.
2. Transaction or system error.
3. Local error or exception.
4. Concurrency control.
5. Disk failure.
6. Catastrophy.

- Reasons 1-4: Focus of the rest of the lecture.
- Reasons 5 and 6:
  - Use the backup of the database and *system log*, and
  - redo all the operations for the *committed* transactions.

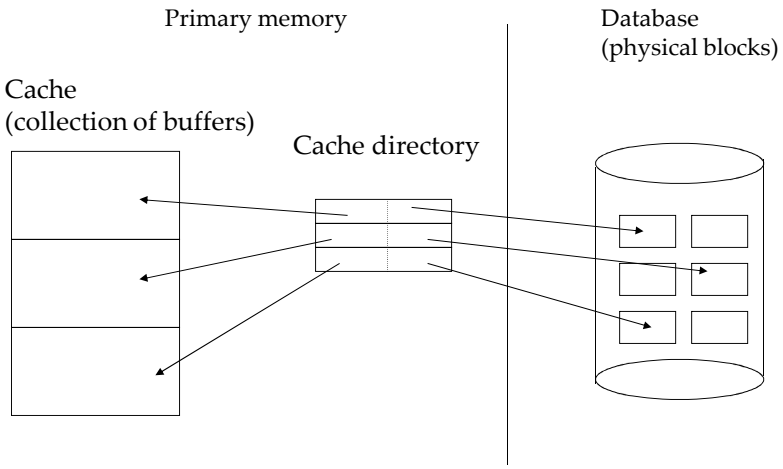
System log

- File with log records.
- Saved on disk + periodically on tape.
- Types of log records:
  - start-transaction T
  - write-item T, X, oldvalue, newvalue
  - read-item T, X
  - commit T
  - abort T
  - checkpoint

Commit

- It indicates that the transaction has been executed successfully in all respects (including serializability) and, thus, the changes made by the transaction can be stored permanently on disk.

Storage hierarchy



How to execute Read- and Write-item(X)

- Read-item(X)
  1. Locate the block on *disk* that contains X.
  2. Copy the block to *primary memory* (a buffer).
  3. Copy X from the buffer to the program variable X.
- Write-item(X)
  1. Locate the block on *disk* that contains X.
  2. Copy the block to *primary memory* (a buffer).
  3. Copy the value of the program variable X to the right place in the buffer.
  4. Store the modified block on *disk*.
- Mind that
  - the block cointaining X may already be in primary memory, or
  - there may not be any empty buffer: Flush the cache.

Flush the cache

- There is no empty buffer for the incoming block. So, some buffer must be freed. The block in this buffer may need to be written to disk.
- How do we know that if the buffer has been modified since it was read from disk ? Dirty bit.
- How do we know that the buffer can be written to disk ? Pin-unpin bit.

Checkpoint

- The system writes on disk
  - all the buffers that have been modified (dirty bit) and can be written to the disk (pin-unpin bit),
  - writes``checkpoint'' in the system log, and
  - writes the system log to disk.
- Advantage: Operations belonging to transactions that have committed before a checkpoint do not need to be redone in case of a crash.
- How often does the system runs a checkpoint ? According to time, number of committed transactions, etc.

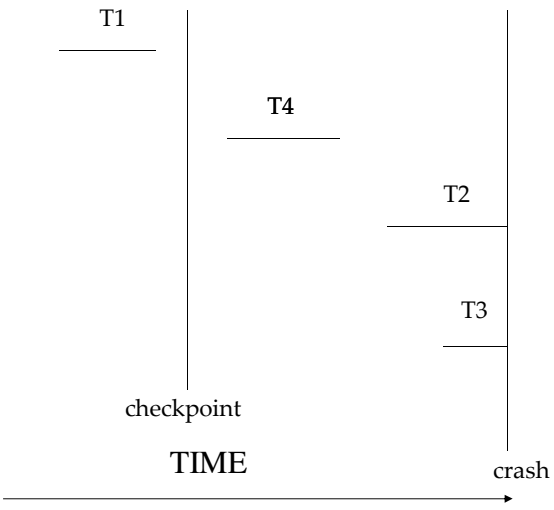
Update methods

- Updating the database in disk after each change is inefficient.
- Deferred update:
  - The database is updated in disk *after (but not neccessarily immediately after)* the transaction has committed. Pin-unpin bit !
  - Before the commit, the transaction has a local environment.
  - At some point after the commit, the system log and buffers are written to the disk.
- Deferred update may run out of buffers.
- Immediate update
  - The database *can* be updated in disk *before* the transaction commits.
  - The system log is written first and, then, the buffers.
- When does the database get updated in disk ? At checkpoint and when flushing the cache.

Recovery: Example

System log

start-transaction T1
write-item T1, D, 10, 20
commit T1
checkpoint
start-transaction T4
write-item T4, B, 10, 20
write-item T4, A, 5, 10
commit T4
start-transaction T2
write-item T2, B, 20, 15
start-transaction T3
write-item T3, A, 10, 30
write-item T2, D, 20, 25
CRASH



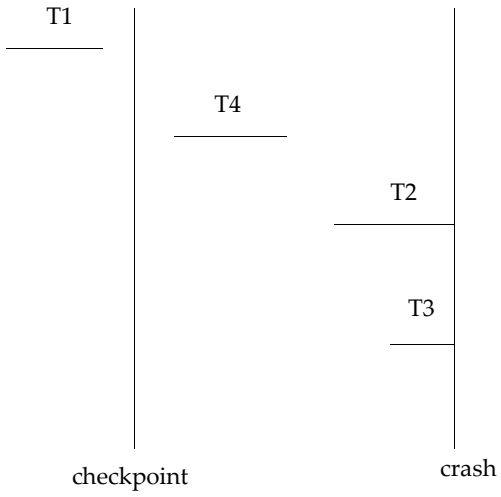
Recovery with deferred update

- The database is updated in disk *after (but not neccessarily immediately after)* the transaction has committed. Then:
  - No need to undo the changes of non-committed transactions.
  - Need to redo the changes of committed transactions.
- NO-UNDO/REDO
- Algorithm:
  - Create a list with active (i.e. non-committed) transactions and a list with committed transactions since the last checkpoint.
  - REDO all the write-item operations of all the transactions in the second list in the order in which they appear in the system log.

NO-UNDO

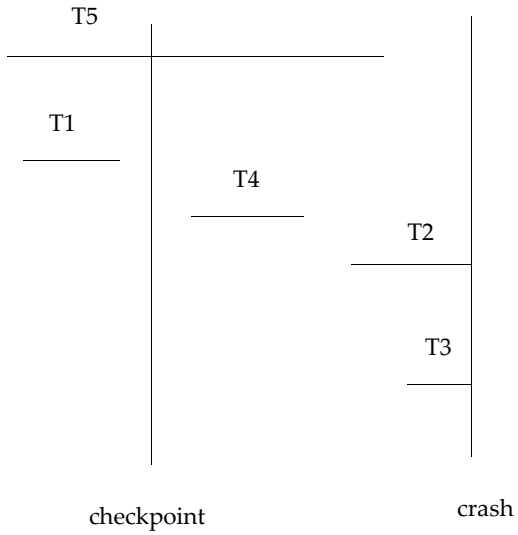
REDO: T4

start-transaction T1
write-item T1, D, ~~10~~, 20
commit T1
checkpoint
start-transaction T4
write-item T4, B, ~~10~~, 20
write-item T4, A, ~~5~~, 10
commit T4
start-transaction T2
write-item T2, B, ~~20~~, 15
start-transaction T3
write-item T3, A, ~~10~~, 30
write-item T2, D, ~~20~~, 25
CRASH



Start-transaction T5
write-item T5, E, 10, 15

start-transaction T1
write-item T1, D, 10, 20
commit T1
checkpoint
start-transaction T4
write-item T4, B, 10, 20
write-item T4, A, 5, 10
commit T4
start-transaction T2
commit T5
write-item T2, B, 20, 15
start-transaction T3
write-item T3, A, 10, 30
write-item T2, D, 20, 25
CRASH

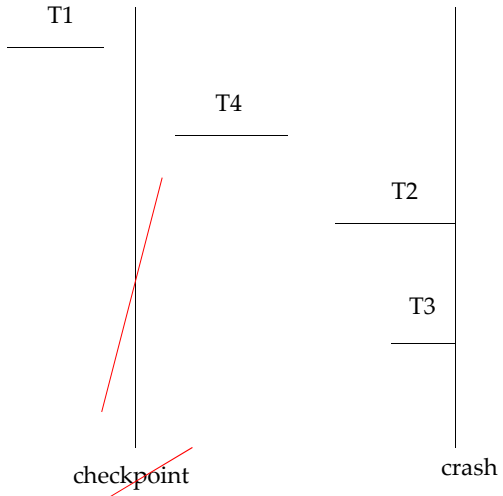


Recovery with immediate update - 1

- The database *can* be updated in disk *before* the transaction commits.
- Additional requirement: The database *must* be updated in disk *before* the transaction commits. Then:
  - No need to redo the changes of committed transactions.
  - Need to undo the changes of non-committed transactions.
- UNDO/NO-REDO
- Algorithm:
  - Create a list with active (i.e. non-committed) transactions and a list with committed transactions since the last checkpoint.
  - UNDO all the write-item operations of all the transactions in the first list in the *reverse* order in which they appear in the system log.

start-transaction T1
write-item T1, D, 10, 20
commit T1
checkpoint
start-transaction T4
write-item T4, B, 10, 20
write-item T4, A, 5, 10
commit T4
start-transaction T2
write-item T2, B, 20, 15
start-transaction T3
write-item T3, A, 10, 30
write-item T2, D, 20, 25
CRASH

NO-REDO
UNDO: T2, T3

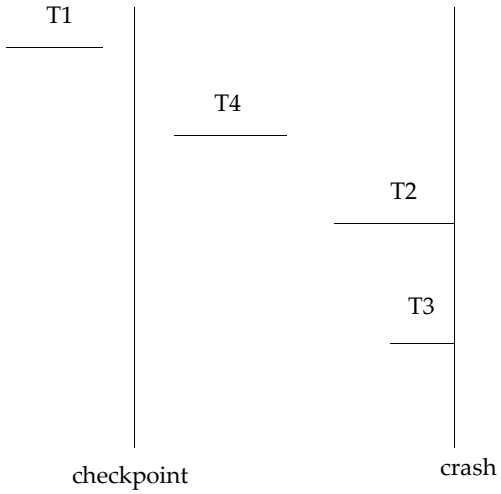


Recovery with immediate update - 2

- The database *can* be updated in disk *before* the transaction commits.
- No additional requirement. Then:
  - Need to redo the changes of committed transactions.
  - Need to undo the changes of non-committed transactions.
- UNDO/REDO
- Algorithm:
  - Create a list with active (i.e. non-committed) transactions and a list with committed transactions since the last checkpoint.
  - UNDO all the write-item operations of all the transactions in the first list in the *reverse* order in which they appear in the system log.
  - REDO all the write-item operations of all the transactions in the second list in the order in which they appear in the system log.

UNDO: T2, T3
REDO: T4

start-transaction T1
write-item T1, D, 10, 20
commit T1
checkpoint
start-transaction T4
write-item T4, B, 10, 20
write-item T4, A, 5, 10
commit T4
start-transaction T2
write-item T2, B, 20, 15
start-transaction T3
write-item T3, A, 10, 30
write-item T2, D, 20, 25
CRASH



Strict schedules

- Problem ?
  - Dirty read.
  - Immediate update 1.

start-transaction T1
write-item T1, D, 10, 20
read-item T2, D
write-item T2, D, 20, 15
commit T2
CRASH

- Recoverable schedule: A transaction T commits only if the last transaction that modified each item read by T has committed.
- Cascadeless schedule: A transaction T can read an item only if the last transaction that modified it has committed.
- Strict schedule: A transaction T can read or write an item only if the last transaction that modified it has committed.
- How to obtain strict schedules ?
  - Strict 2PL: Do not release write locks until after commit.
  - Rigorous 2PL: Do not release any lock until after commit.
- With strict schedules, no need to store read-item T, X in the system log.