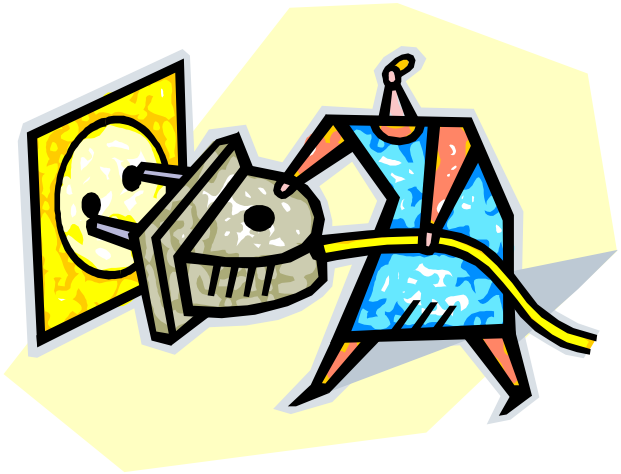
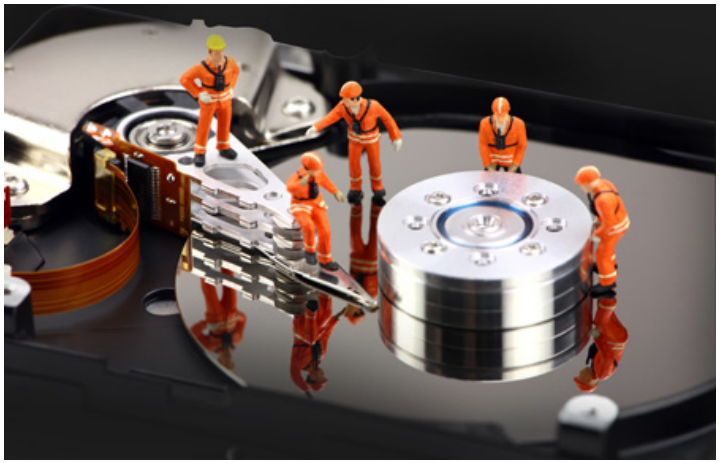
COSC344
Database Theory and Applications



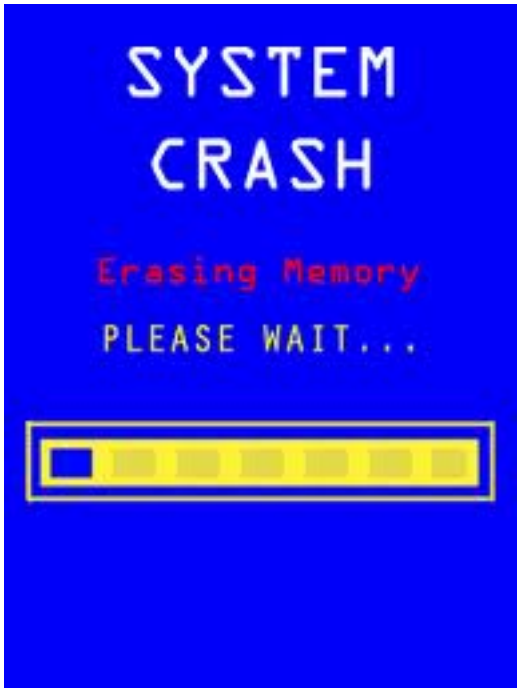
Lecture 23
Database Recovery

Overview

- Previous Lectures
  - Transactions
  - Concurrency control
- This Lecture
  - Recovery
  - Source: Chapter 22



Overview



Questions to Ponder

- ACID properties of transactions

Atomicity

Consistency

Isolation

Durability

How to?

ACID Properties

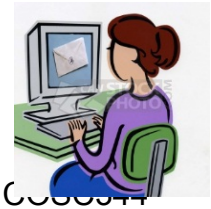
- ***Question 1**: What happens if it fails after step 3 before 6.*

ROLLBACK

errors

Transaction
BEGIN TRANSACTION
1. **read**(*A*)
2. A := *A* − 1000
3. **write**(*A*)
4. **read**(*B*)
5. *B* := *B* + 1000
6. **write**(*B*)
COMMIT TRANSACTION

- *Property 1:* ***Atomicity***
- A transaction is atomic
- Either all operations in the transaction have to be performed or none should be performed.
- Logical unit of work/recovery
- Failure recovery

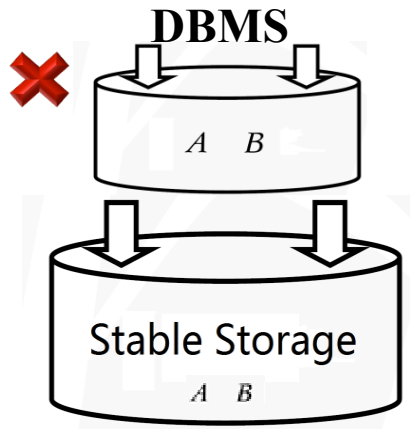


"Atomicity" ensures "no money is taken from A without being given to B"

ACID Properties (Durability)

- *Question 4: What happens if database server crashes before the changed data is written to stable storage?*

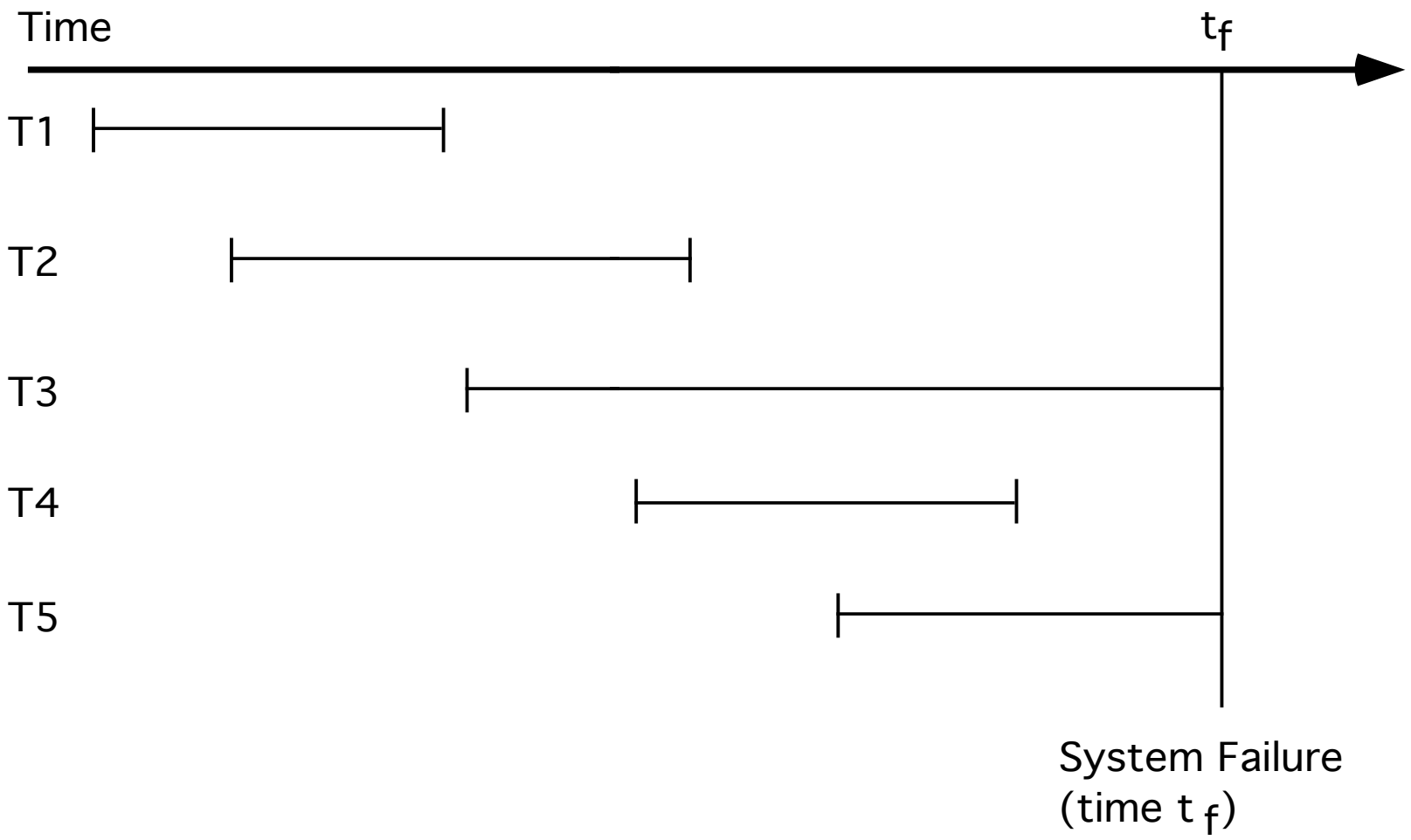
Transaction
BEGIN TRANSACTION …. COMMIT TRANSACTION



- *Property 4:* ***Durability***
  - once the user is notified that the transaction has completed, updates to the database by the transaction must be persistent despite failures.
  - Committed changes will not be reversed
  - Failure Recovery

"Durability" guarantees "Once completed, money does not go back to A"

Why Recovery is Needed



Why Recovery is Needed

- disks do not reflect the true state of the committed database
- What if the contents of main memory are lost?
- Types of Failures
  - System crash
  - Transaction failures or errors
  - Disk failure
  - Physical problems and catastrophes
- If there is a failure
  - Some transactions must be __redone__ or __undone__.

System Log File

- File maintained on disk of all __changes__ to the database.
- sequential, append-only file, not affected by any failure except for disk or catastrophic failures
- Entries in the log file
  - [start, Ti]
  - [write_item, Ti, X, old_value, new_value]
  - [commit, Ti]
  - [abort, Ti]

(a)

T_1	T_2
read_item(A)	read_item(B)
read_item(D)	write_item(B)
write_item(D)	read_item(D)
	write_item(D)

(b)

[start_transaction, T_1]
[write_item, T_1, D, 20]
[commit, T_1]
[start_transaction, T_2]
[write_item, T_2, B, 10]

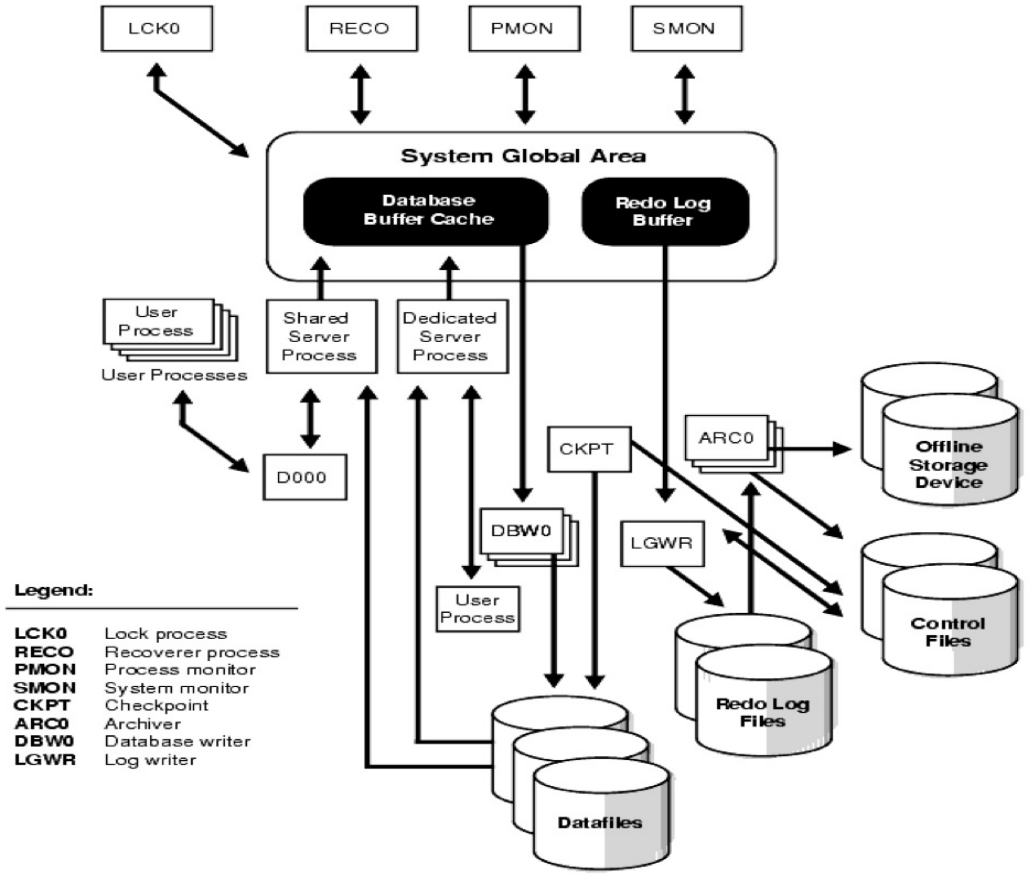
Commit Point of a Transaction

- Commit Point
  - All operations executed successfully + recorded in the log file
- Force write a commit record [commit, Ti] into the log file.
- Beyond the commit point, the transaction is said to be **committed** and its effect is assumed to be permanently recorded in the database.
- System failure occurs
  - For transactions that have written [start, Ti], but not [commit, Ti]: undo (rollback)

  - For transactions that have written both: redo the effects

Caching of Disk Blocks

- Disk blocks to be updated are cached into main memory buffers, and then updated in memory before being written back to disk.



Caching of Disk Blocks

- Disk blocks to be updated are cached into main memory buffers, and then updated in memory before being written back to disk.
- In-place updating: write the buffer to the _same_ original disk location.

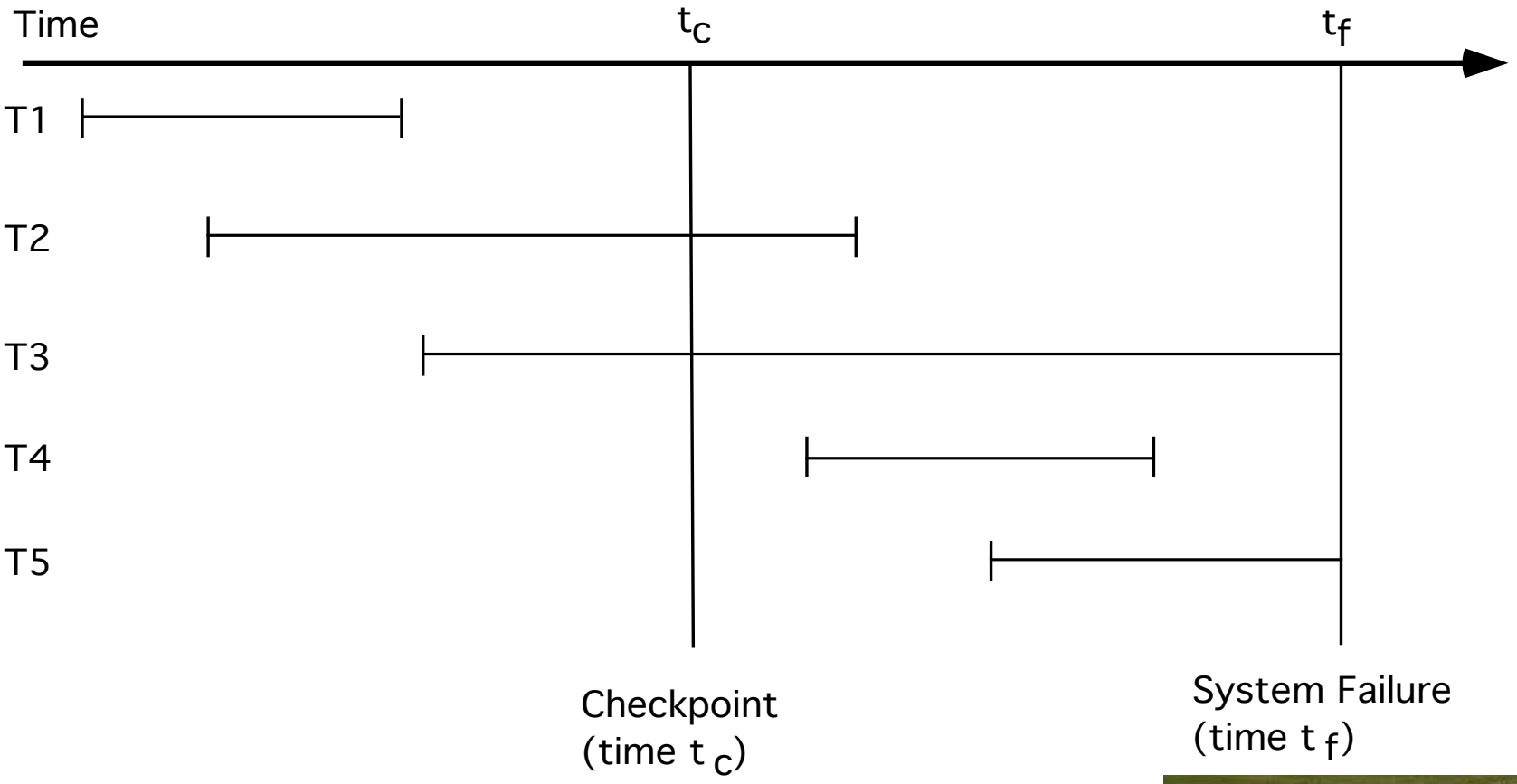
Write-Ahead Logging Protocol

- **Before Image (BFIM):** the _old_ value before updating
- **After Image (AFIM):** the _new_ value after updating
- Write-Ahead Logging
  - The BFIM of the data item is recorded in the log entry and is flushed to disk _before_ the BFIM is overwritten with the AFIM on disk.

Checkpoints in System Log

- Checkpoint: the point at which a record is written into the log when the system writes __all__ modified buffers to the database on disk.
- Taking a checkpoint involves:
  - Temporarily suspend execution of transactions.
  - 'Force-write' the contents of the database buffers to disk.
    - Note that we must log the 'force-write'
  - Write a [checkpoint] record to the log and 'force-write' the log to disk.
  - Resume executing transactions.

Transaction Recovery



Transaction Recovery (continued)

- How to fix on startup?
- Transactions of type T1 were forced to disk as part of the checkpoint.
  - Not part of the recovery process
- Transactions of type T3 and T5 must be undone.
- Transactions of type T2 and T4 must be redone.
- Transactions that complete unsuccessfully (rollback) before time t_f do not enter into the recovery process.

Recovery Process in General

- On startup
  - Users are locked out.
  - The log file is checked.
- If there was a failure, users remain locked out.
- Execute the recovery process.
- Activate users.

Steal/No-Steal, Force/No-Force

- Steal/No-Steal
  - Steal: allows writing an updated buffer to disk before the transaction commits.
  - No-Steal means that UNDO will never be needed during recovery
- Force/No-Force
  - Force: All pages updated by a transaction are immediately written to disk when the transaction commits.
  - Force means that REDO will never be needed for committed transactions during recovery
- Which to use?
  - Most DBMS use steal/no-force strategy.
  - Steal does not need more buffer
  - No-force eliminates the I/O cost

Deferred Update

- Defer or postpone any actual updates to the database until the transaction completes successfully and reaches its commit point.
- During the transaction, updates are recorded only in the log file and the cache buffers. After the transaction reaches its commit point and the log is force-written, the updates are recorded in the database.
- Failures will require a REDO. No UNDO is needed. (NO-UNDO/REDO algorithm)

No-Steal

Recovery - Deferred Update, Single User

- Create two lists
  - Commit list: the committed transactions since the last checkpoint.
  - Active list: the active transactions.
- Search forward through the log from the most recent [checkpoint] record
- If a [start_transaction] record is found for a transaction T, add T to the active list.
- Add any [write_item] records to the active list.
- If a [commit] record is found for T, move T and its [write_item] records from the active list to the committed list.
- When the end of the log is reached, work forward through the committed list, redoing the [write_item] records.

Recovery - Deferred Update, Single User (cont.)

(a)

T_1	T_2
read_item(A)	read_item(B)
read_item(D)	write_item(B)
write_item(D)	read_item(D)
	write_item(D)

(b)

[start_transaction, T_1]
[write_item, T_1, D, 20]
[commit, T_1]
[start_transaction, T_2]
[write_item, T_2, B, 10]
[write_item, T_2, D, 25] ← system crash

REDO

The [write_item,...] operations of T_1 are redone.
T_2 log entries are ignored by the recovery process.

Figure 21.2 An example of recovery using deferred update in a single-user environment. (a) The `read` and `write` operations of two transactions. (b) The system log at the point of crash.

Immediate Update

Steal

- When a transaction issues an update command, the database can be updated without any need to wait for the transaction to reach its commit point. However the update operation must still be recorded in the log before it is applied to the database.
- Updates written to the database before commit completes
  - No need to redo any operations of the committed transactions. UNDO/NO-REDO

Steal/Force

- Commit completes before updates are written to the database
  - (lazy update) Some changes will have to be undone and some redone. UNDO/REDO.
  - This is the more general case. Popular

Steal/No-Force

Recovery - Immediate Update

- Create two lists of transactions
  - Commit list: the committed transactions since the last checkpoint.
  - Active list: the active transactions.
- Search forward through the log from the most recent checkpoint record.
- If a [start_transaction] record is found for a transaction T, add T to the active list.
- Add any [write_item] records for T to the active list .
- If a [commit] record is found for T, move T and its [write_item] records from the active list to the committed list
- When the end of the log is reached
  - Work backwards through the active list, undoing all [write_item] entries
  - Work forwards through the committed list, redoing all [write_item] entries

Catastrophic Failures

- So far all the recovery techniques have assumed the system log, maintained on disk, has not been lost.
- If there is a disk failure, we're in the area of catastrophic failure

- Recovery techniques include
  - Database backup
    - periodically the whole database is copied to a cheap storage medium
    - follow with frequent backups of the system log. New log started after every database backup

Summary

- System log
- Commit point
- Steal/no-steal, Force/no-force
- Write-ahead logging protocol
- Check point
- Deferred update
- Immediate update