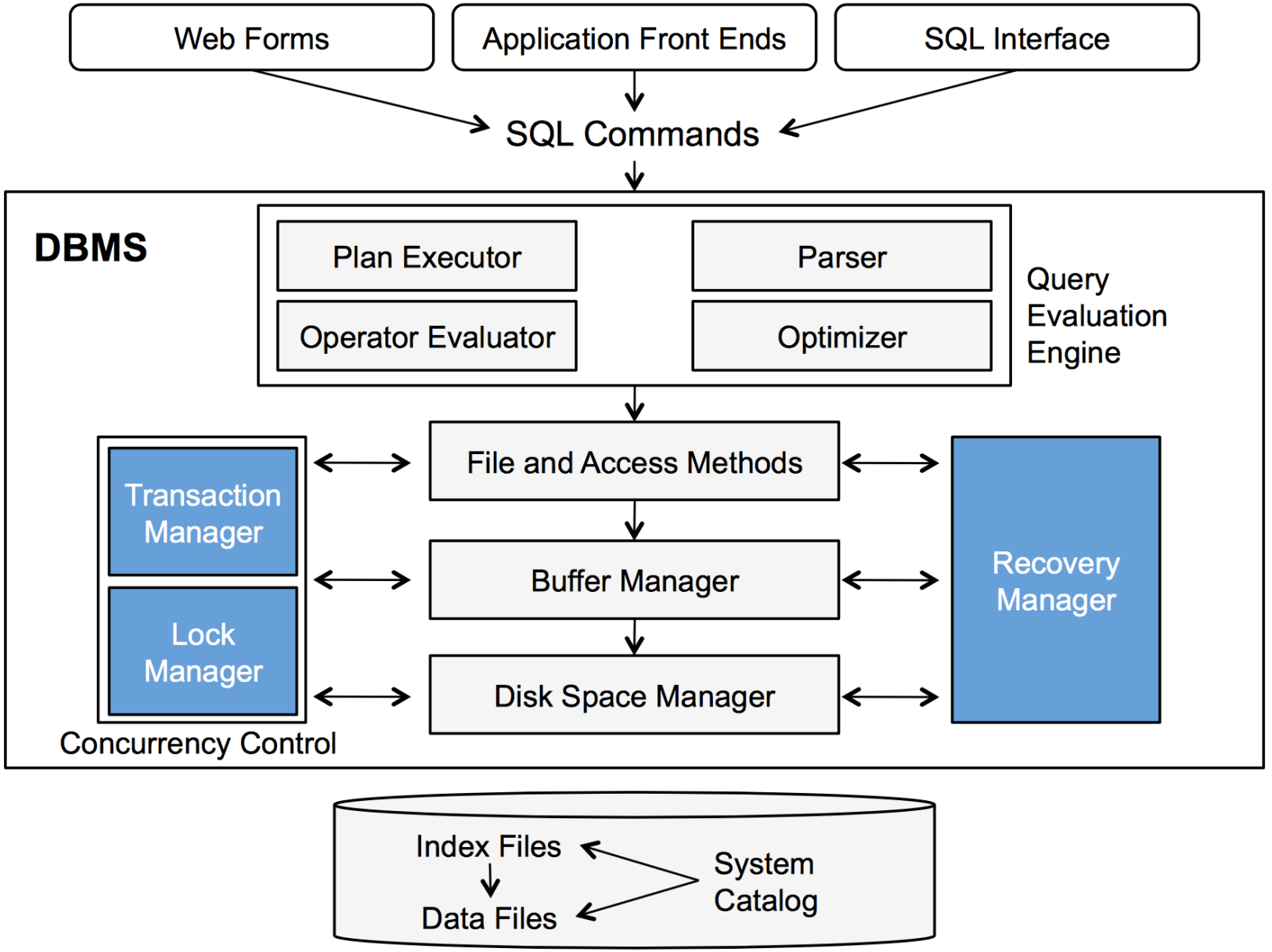
Database Management System

Lecture 10

Recovery

* Some materials adapted from R. Ramakrishnan, J. Gehrke and Shawn Bowers

Basic Database Architecture



Recovery

- Which ACID properties have to do with Recovery?
  - Atomicity (all or no actions happen in a transaction)
  - Durability (if transaction commits, effects are permanent)
- We like to think updates happen "*in place*"
  - Data is overwritten or deleted on disk directly
- But when you insert, delete, or update a record
  - You are modifying data in a buffer (holds a copy of the page)
  - The buffer is later copied back out to disk
  - If the system crashes during or after a transaction ...
  - then zero or more updates may have been persisted
  - This can violate atomicity and durability

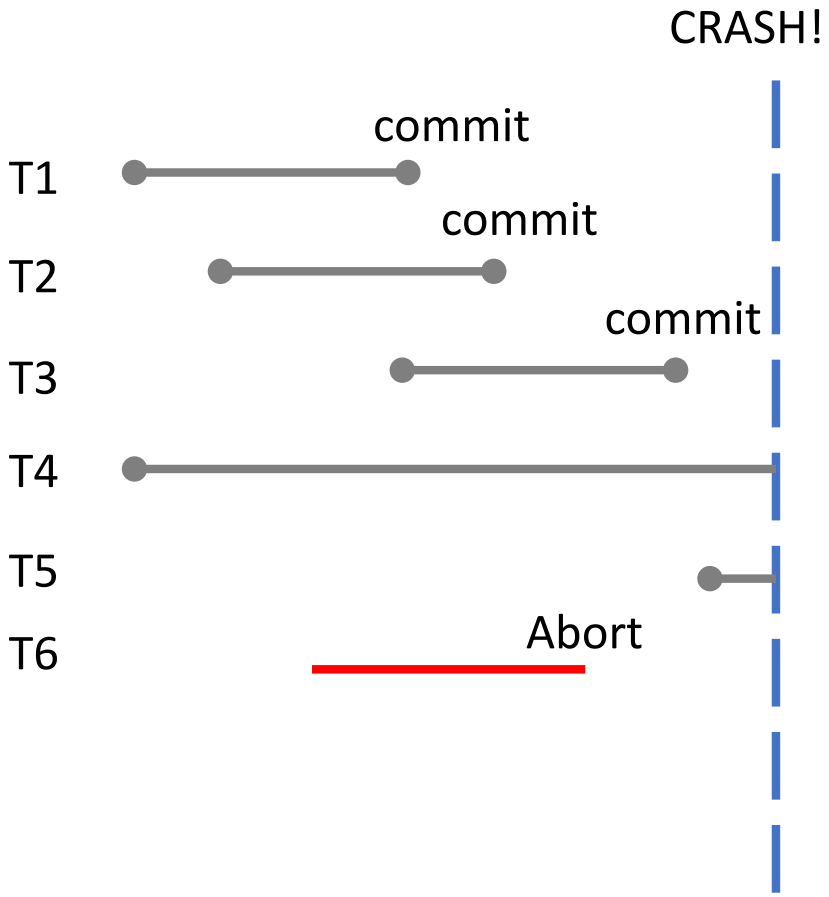
Recovery System

- Recovery in a DBMS is based on a *Write Ahead Log*
  - Also used for aborts …
- The write ahead log
  - is (often) placed on a separate disk from the data
  - begins after each backup (e.g., nightly)
- A log record is written for every insert, update, delete, begin trans., commit, abort, and checkpoint
- A *log record* contains:
  - <transid, rid, action, old data, new data>

Write-Ahead Logging (WAL)

When system restarts after this crash:

- T1, T2, and T3 committed before crash
  - They must be “durable” (persist), WAL should REDO transactions
- T4 and T5 were still running and must be aborted
  - Effects should not be permanent, WAL should UNDO transactions
- T6 was aborted before the crash
  - WAL should UNDO T6 (without overriding redos)



Write Ahead Logs

- *"Write ahead" logs must obey these rules*
- The Atomic Rule:
  - The **log entry** for an insert, update, or delete must be written to (log) disk **before the change is made to the DB**
- The Durability Rule:
  - All **log entries** for a transaction must be written to (log) disk **before the commit record is written to (log) disk**
- *We know what could have changed (atomic rule) ... and we know all operations of committed transactions (durability rule)*

Example Log

transid	rid	action	old	new
T1	A	update	100	200
T2	C	update	1000	500
T2	D	update	500	1000
T2		commit		
CRASH!				

- *What did each transaction do before the crash?*

Example Log

transid	rid	action	old	new
T1	A	update	100	200
T2	C	update	1000	500
T2	D	update	500	1000
T2		commit		
CRASH!				

- *After the crash, how should the recovery manager ensure each transaction is atomic?*

It is More Complex in Practice

- Sources of Complexities:
  - Inserts and deletes
  - Updating B+ Trees (e.g., during page splits)
  - Rolling back aborts at time of crash
  - Crashes during recovery
- Lots of work in the DB community on implementing recovery correctly

Handling Aborts

Write ahead logs enable transaction aborts

- The DBMS uses the Write Ahead Log and does a regular undo
- Note that in UNDO
  - the DBMS applies the “old values” from the Write Ahead
- Log in *reverse order* for the transaction

Example Abort

transid	rid	action	old	new
T1	A	update	ABC	DEF
T2	C	update	1000	500
T2	D	update	500	1000
T1	B	update	300	400
T1	A	update	DEF	GHI
T1		Abort		

- A starts as ABC, is updated to DEF, then updated to GHI
- What should A be after the abort (i.e., UNDO)?

Checkpoints

Periodically the DBMS creates a checkpoint

- A“snapshot”
- Flush all DB pages to disk
  - We know all updates in log prior to checkpoint have been written to disk
  - May or may not need require “stop” the system ... waiting until all transactions finish
  - Writes a “checkpoint” record to the log
- Checkpoint record implies DB is in a consistent state
- Checkpoints minimize time required to recover from a crash ... by telling us how far back to go in the log