

Lecture 5: Logging (Part 1)

CREATING THE NEXT®

Today's Agenda

Logging (Part 1)

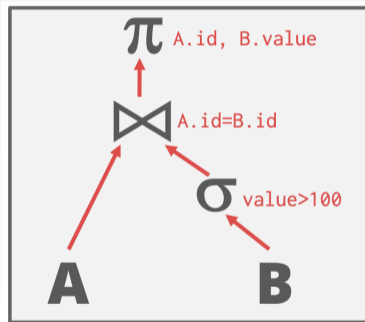
- 1.1 Recap
- 1.2 Motivation
- 1.3 Failure Classification
- 1.4 Buffer Pool Policies
- 1.5 Shadow Paging
- 1.6 Conclusion

Recap

Query Plan

- The operators are arranged in a tree.
- Data flows from the leaves of the tree up towards the root.
- The output of the root node is the result of the query.

```
SELECT A.id, B.value  
FROM A, B  
WHERE A.id = B.id AND B.value > 100
```



Query Processing

- Access methods are the alternative ways for retrieving specific tuples
- Hashing is almost always better than sorting for operator execution.
- Caveats:
 - ▶ Sorting is better on non-uniform data.
 - ▶ Sorting is better when result needs to be sorted.
- Good DBMSs use either or both.

Process Models

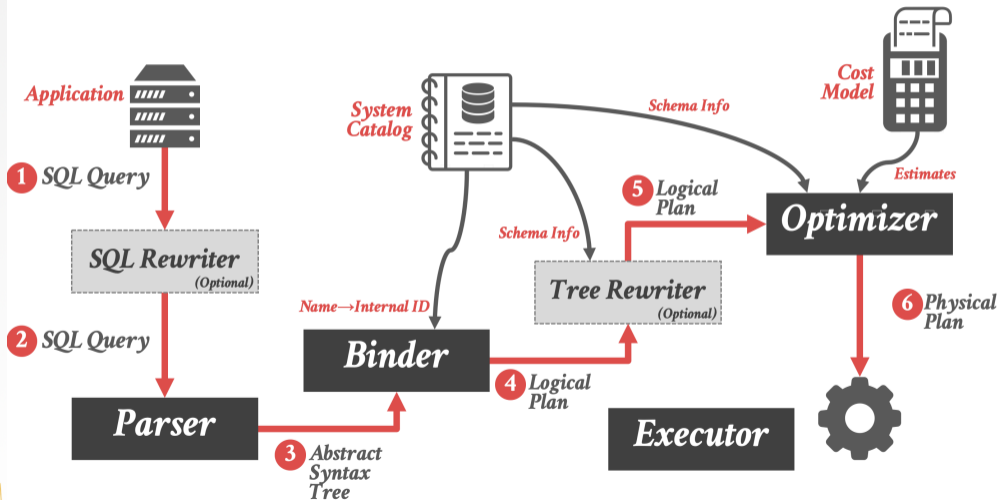
- The same query plan be executed in multiple ways.
- A DBMS's processing model defines how the system executes a query plan.
- (Most) DBMSs will want to use an index scan as much as possible.
- Parallel execution is important.
- (Almost) every DBMS supports this.
- This is really hard to get right.

Today's Agenda

- Motivation
- Failure Classification
- Buffer Pool Policies
- Shadow Paging

Motivation

Anatomy of a Database System [Monologue]



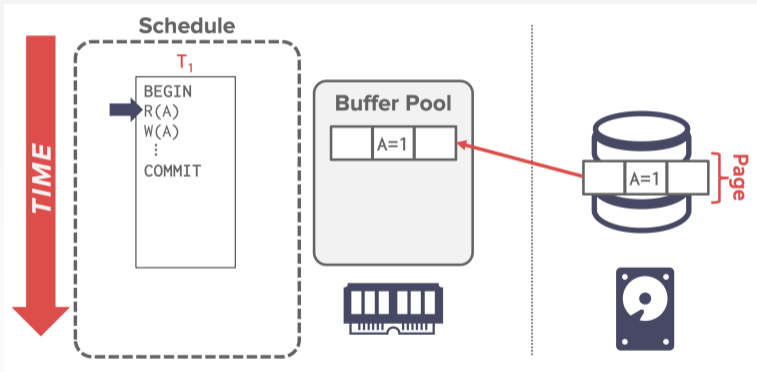
Anatomy of a Database System [Monologue]

- Process Manager
 - ▶ Manages client connections
- Query Processor
 - ▶ Parse, plan and execute queries on top of storage manager
- Transactional Storage Manager
 - ▶ Knits together buffer management, concurrency control, logging and recovery
- Shared Utilities
 - ▶ Manage hardware resources across threads

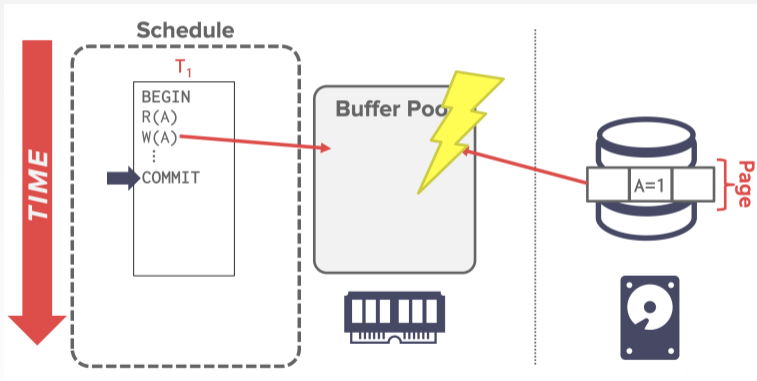
Anatomy of a Database System [Monologue]

- Process Manager
 - ▶ Connection Manager + Admission Control
- Query Processor
 - ▶ Query Parser
 - ▶ Query Optimizer (*a.k.a.*, Query Planner)
 - ▶ Query Executor
- Transactional Storage Manager
 - ▶ Lock Manager
 - ▶ Access Methods (*a.k.a.*, Indexes)
 - ▶ Buffer Pool Manager
 - ▶ Log Manager
- Shared Utilities
 - ▶ Memory, Disk, and Networking Manager

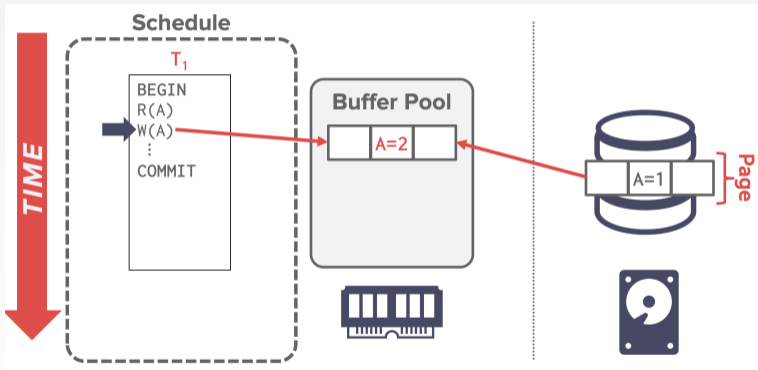
Motivation



Motivation



Motivation



Crash Recovery

- Recovery algorithms are techniques to ensure database consistency, transaction atomicity, and durability despite failures.
- Recovery algorithms have two parts:
 - ▶ Actions during normal txn processing to ensure that the DBMS can recover from a failure.
 - ▶ Actions after a failure to recover the database to a state that ensures atomicity, consistency, and durability.

Failure Classification

Crash Recovery

- DBMS is divided into different components based on the underlying storage device.
- We must also classify the different types of failures that the DBMS needs to handle.

Storage Types

- Volatile Storage

- ▶ Data does not persist after power loss or program exit.
- ▶ Examples: DRAM, SRAM

- Non-volatile Storage

- ▶ Data persists after power loss and program exit.
- ▶ Examples: HDD, SDD

- Stable Storage

- ▶ A non-existent form of non-volatile storage that survives all possible failures scenarios.
- ▶ Approximated using a collection of storage devices.

Failure Classification

- Type 1 – Transaction Failures
- Type 2 – System Failures
- Type 3 – Storage Media Failures

Transaction Failures

- **Logical Errors:**
 - ▶ Transaction cannot complete due to some internal error condition (*e.g.*, integrity constraint violation).
- **Internal State Errors:**
 - ▶ DBMS must terminate an active transaction due to an error condition (*e.g.*, deadlock).

System Failures

- **Software Failure:**
 - ▶ Problem with the DBMS implementation (*e.g.*, uncaught divide-by-zero exception).
- **Hardware Failure:**
 - ▶ The computer hosting the DBMS crashes (*e.g.*, power plug gets pulled).
 - ▶ **Fail-stop Assumption:** Non-volatile storage contents are assumed to not be corrupted by system crash.

Storage Media Failure

- **Non-Repairable Hardware Failure:**
 - ▶ A head crash or similar disk failure destroys all or part of non-volatile storage.
 - ▶ Destruction is assumed to be detectable (*e.g.*, disk controller use checksums to detect failures).
- No DBMS can recover from this! Database must be restored from archived version.

Observation

- The primary storage location of the database is on non-volatile storage, but this is much slower than volatile storage.
- Use volatile memory for faster access:
 - ▶ First copy target record into memory.
 - ▶ Perform the writes in memory.
 - ▶ Write dirty records back to disk.

Observation

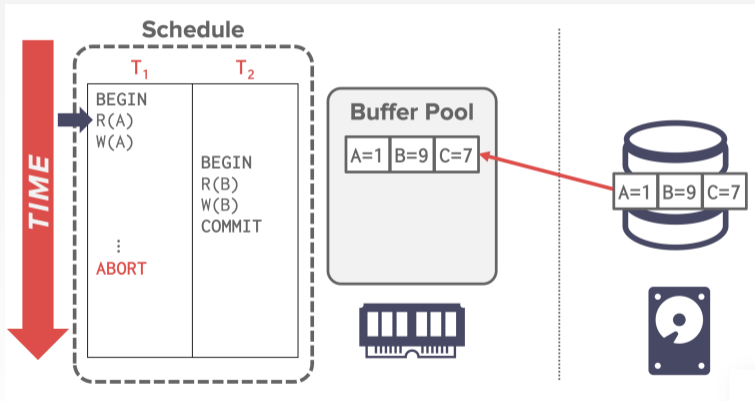
- The DBMS needs to ensure the following guarantees:
 - ▶ The changes for any txn are durable once the DBMS has told somebody that it committed.
 - ▶ No partial changes are durable if the txn aborted.

Buffer Pool Policies

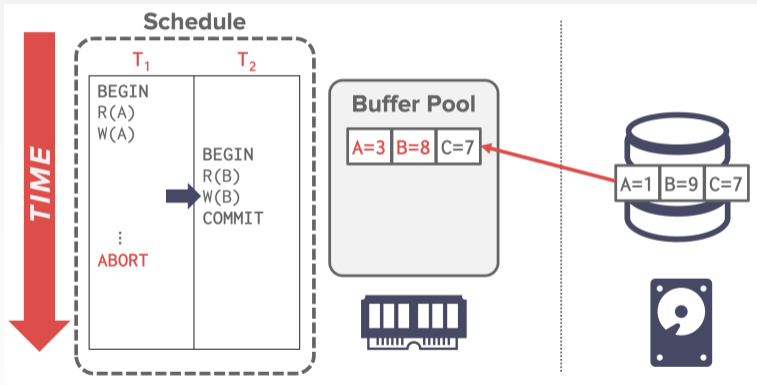
Undo vs. Redo

- **Undo**: The process of removing the effects of an incomplete or aborted txn.
- **Redo**: The process of re-instating the effects of a committed txn for durability.
- How the DBMS supports this functionality depends on how it manages the buffer pool. . .

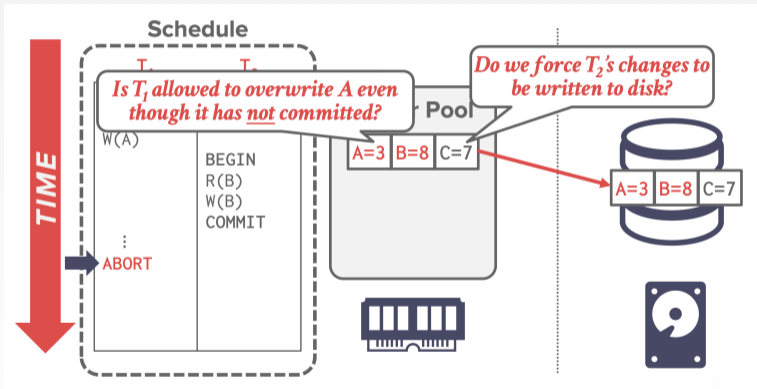
Buffer Pool



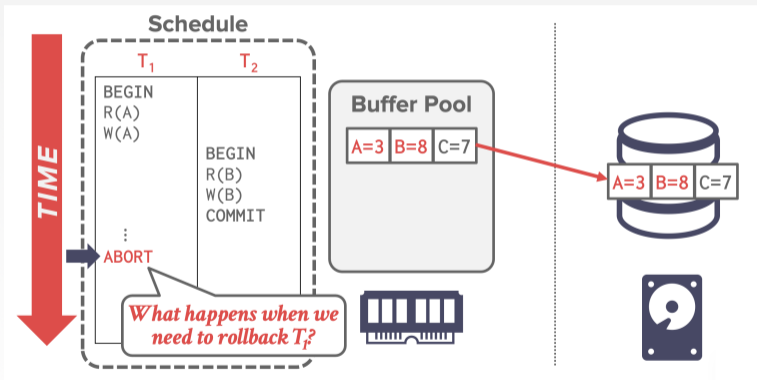
Buffer Pool



Buffer Pool



Buffer Pool



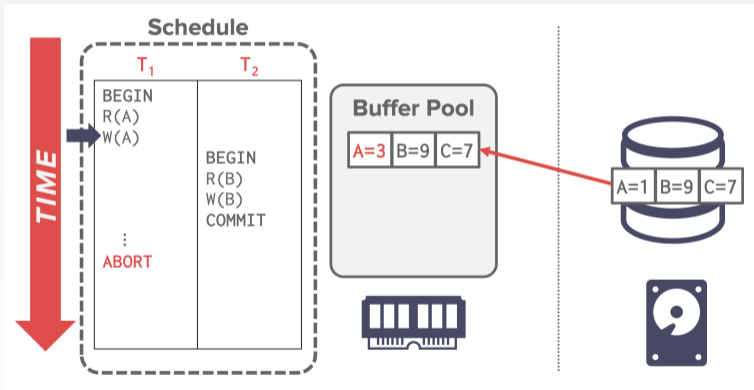
Steal Policy

- Whether the DBMS allows an uncommitted txn to overwrite the most recent committed value of an object in non-volatile storage.
- **STEAL**: Is allowed.
- **NO-STEAL**: Is **not** allowed.

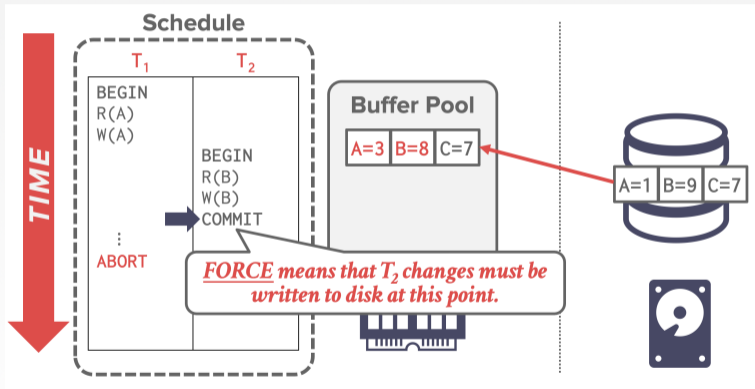
Force Policy

- Whether the DBMS requires that all updates made by a txn are reflected on non-volatile storage before the txn is allowed to commit.
- FORCE: Is required.
- NO-FORCE: Is not required.

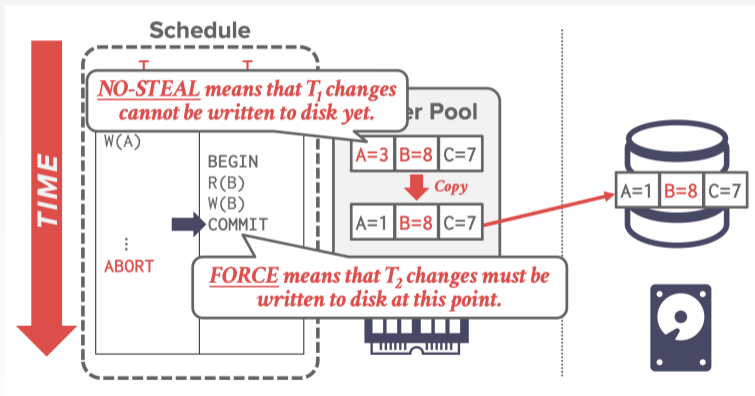
NO-STEAL + FORCE



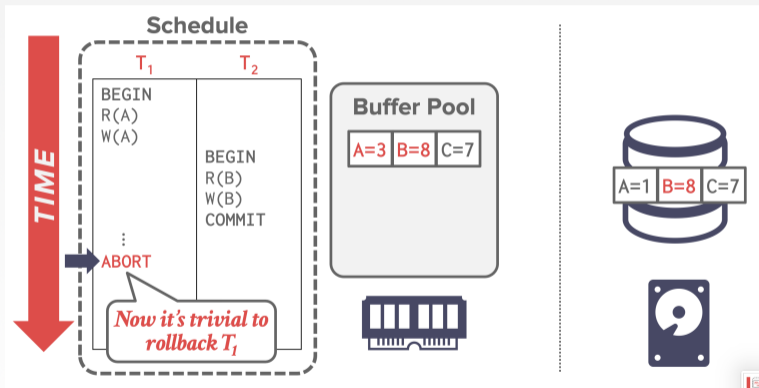
NO-STEAL + FORCE



NO-STEAL + FORCE



NO-STEAL + FORCE



NO-STEAL + FORCE

- This approach is the easiest to implement:
 - ▶ Never have to undo changes of an aborted txn because the changes were not written to disk.
 - ▶ Never have to redo changes of a committed txn because all the changes are guaranteed to be written to disk at commit time (assuming atomic hardware writes).
- Previous example cannot support **write sets** that exceed the amount of physical memory available.

Shadow Paging

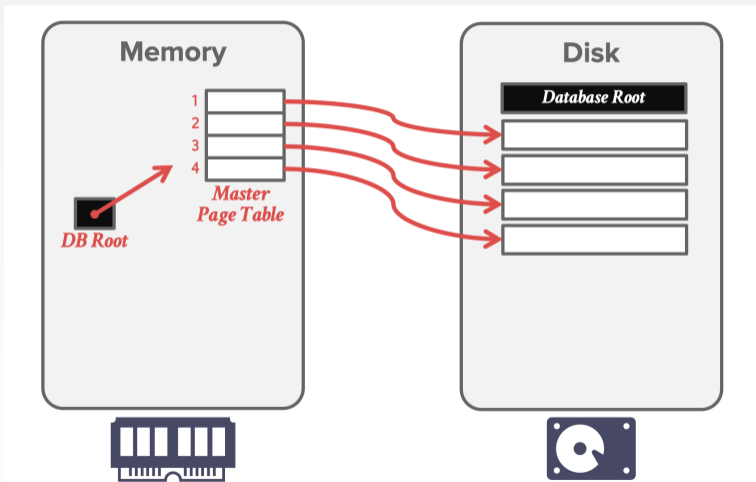
Shadow Paging

- Maintain two separate copies of the database:
 - ▶ Master: Contains only changes from committed txns.
 - ▶ Shadow: Temporary database with changes made from uncommitted txns.
- Txns only make updates in the shadow copy.
- When a txn commits, atomically switch the shadow to become the new master.
- Buffer Pool Policy: NO-STEAL + FORCE

Shadow Paging

- Instead of copying the entire database, the DBMS copies pages on write.
- Organize the database pages in a tree structure where the root is a single disk page.
- There are two copies of the tree, the master and shadow
 - ▶ The root points to the master copy.
 - ▶ Updates are applied to the shadow copy.

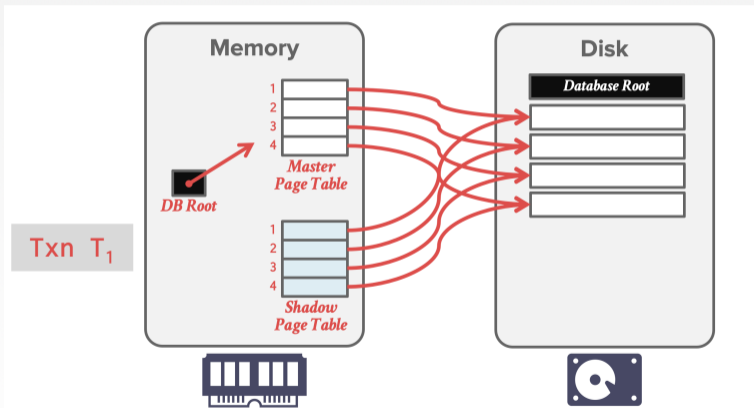
Shadow Paging – Example



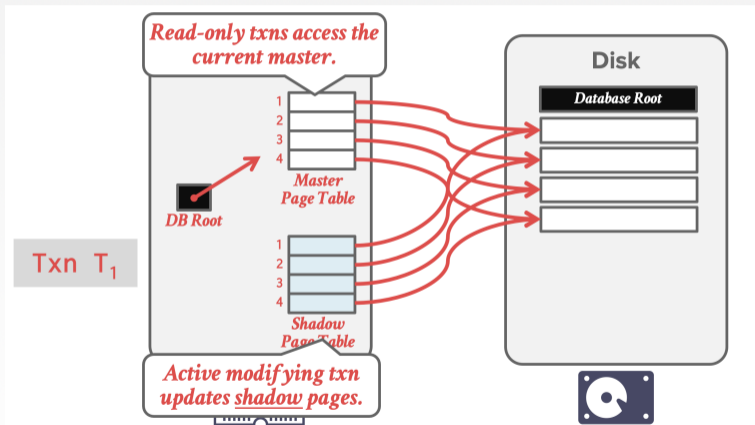
Shadow Paging

- To install the updates, overwrite the root so it points to the shadow, thereby swapping the master and shadow:
 - ▶ Before overwriting the root, none of the txn's updates are part of the disk-resident database
 - ▶ After overwriting the root, all the txn's updates are part of the disk-resident database.

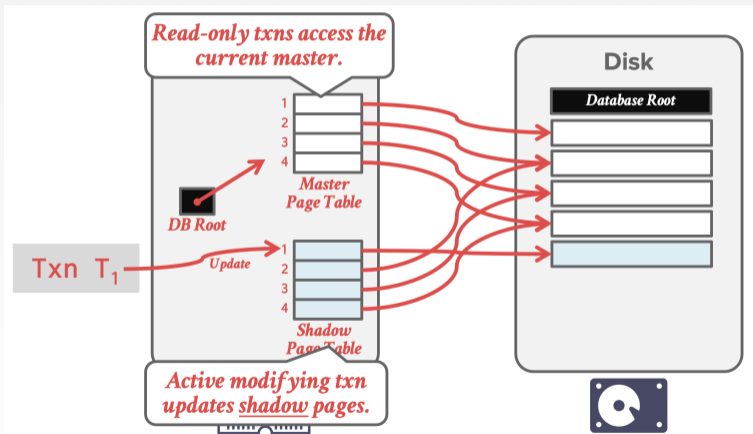
Shadow Paging – Example



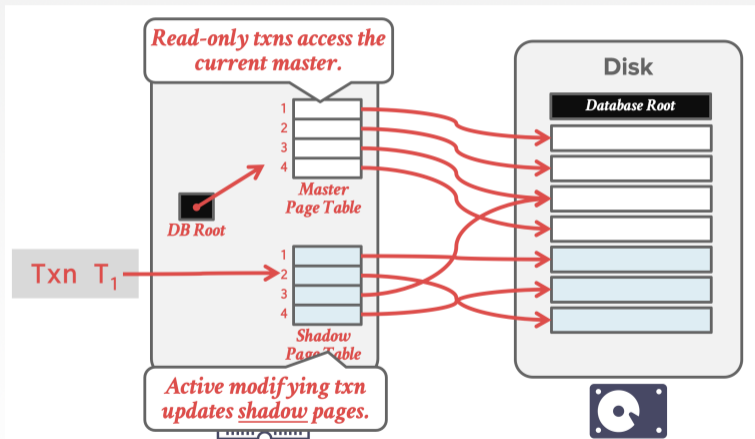
Shadow Paging – Example



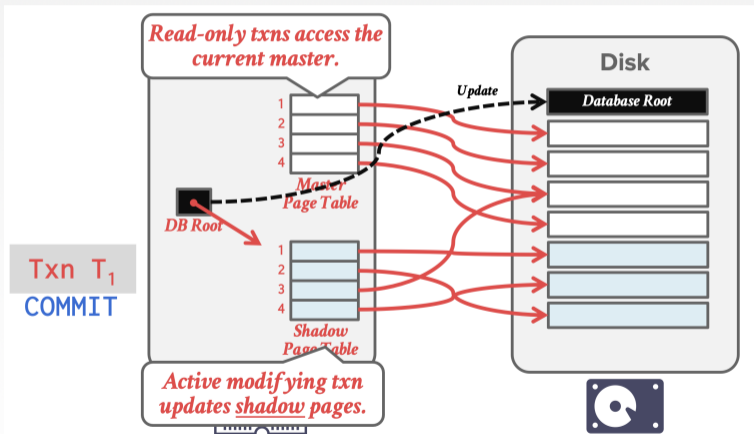
Shadow Paging – Example



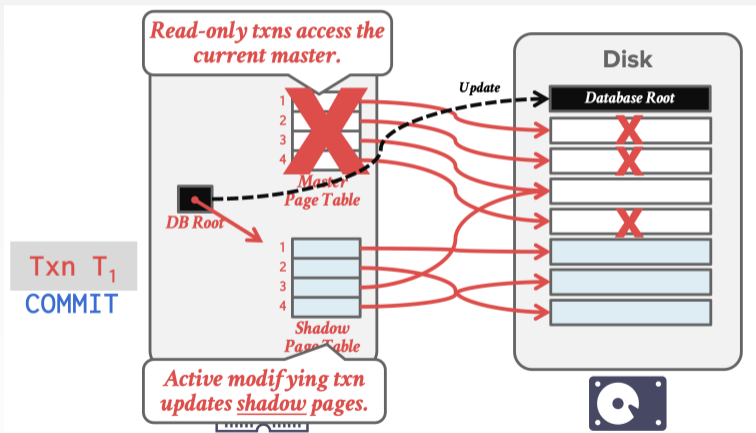
Shadow Paging – Example



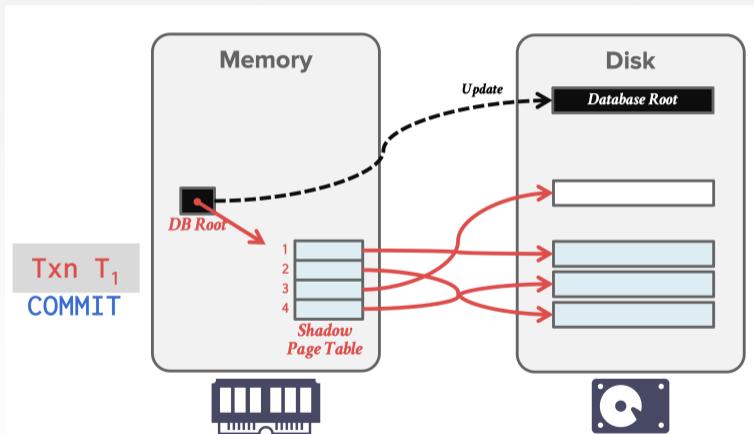
Shadow Paging – Example



Shadow Paging – Example



Shadow Paging – Example



Shadow Paging – Undo/Redo

- Supporting rollbacks and recovery is easy.
- **Undo**: Remove the shadow pages. Leave the master and the DB root pointer alone.
- **Redo**: Not needed at all.

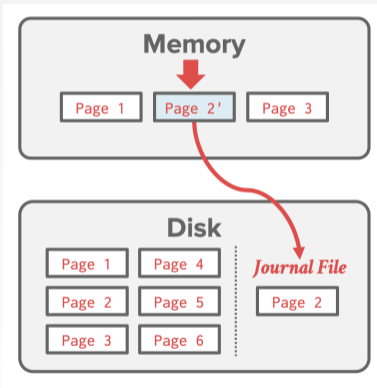
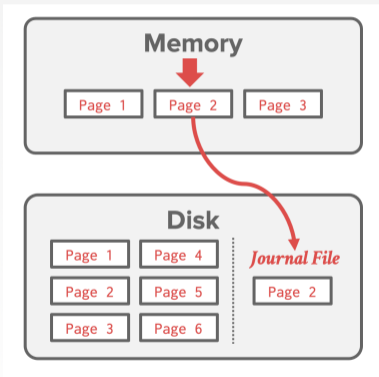
Shadow Paging – Disadvantages

- Copying the entire page table is expensive:
 - ▶ Use a page table structured like a B+tree.
 - ▶ No need to copy entire tree, only need to copy paths in the tree that lead to updated leaf nodes.
- Commit overhead is high:
 - ▶ Flush every updated page, page table, and root.
 - ▶ Data gets fragmented.
 - ▶ Need garbage collection.
 - ▶ Only supports one writer txn at a time or txns in a batch.

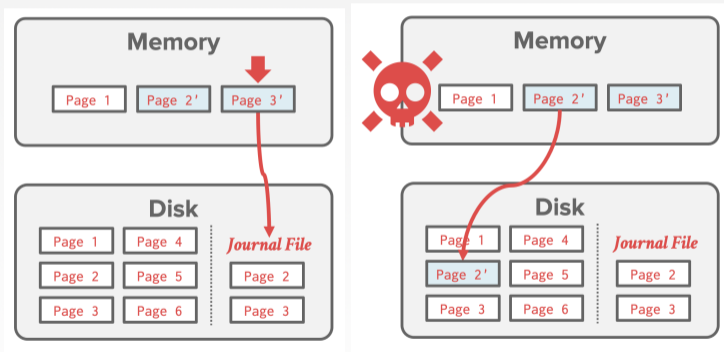
SQLITE (PRE-2010)

- When a txn modifies a page, the DBMS copies the original page to a separate **journal file** before overwriting master version.
- After restarting, if a journal file exists, then the DBMS restores it to undo changes from uncommitted txns.

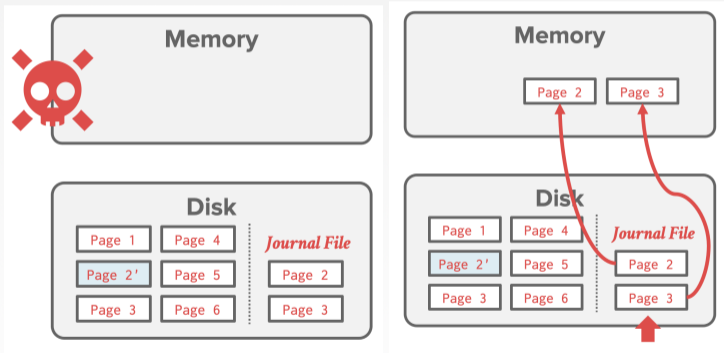
SQLITE (PRE-2010)



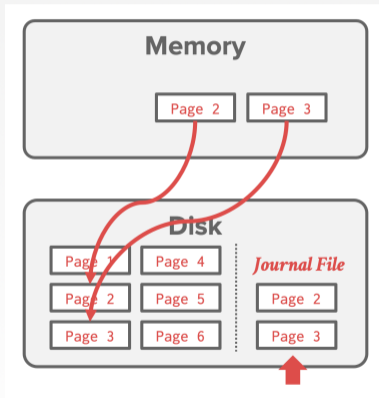
SQLITE (PRE-2010)



SQLITE (PRE-2010)



SQLITE (PRE-2010)



Observation

- Shadowing page requires the DBMS to perform writes to random non-contiguous pages on disk.
- We need a way for the DBMS convert random writes into sequential writes.

Conclusion

Parting Thoughts

- Recovery algorithms are techniques to ensure database consistency, transaction atomicity, and durability despite failures.
- Recovery algorithms have two parts:
 - ▶ Actions during normal txn processing to ensure that the DBMS can recover from a failure.
 - ▶ Actions after a failure to recover the database to a state that ensures atomicity, consistency, and durability.
- Three types of failures: transaction, system, and hardware failures
- Buffer policies: NO-STEAL + FORCE

Next Class

- Write Ahead Logging