Lecture #22: Database Crash Recovery

15-445/645 Database Systems (Fall 2025)
`https://15445.courses.cs.cmu.edu/fall2025/`
Carnegie Mellon University
Andy Pavlo

1 Crash Recovery

The DBMS relies on its recovery algorithms to ensure database consistency, transaction atomicity, and durability despite failures. Each recovery algorithm is comprised of two parts:

- Actions during normal transaction processing to ensure that the DBMS can recover from a failure
- Actions after a failure to recover the database to a state that ensures the atomicity, consistency, and durability of transactions.

The key to database resilience is the management of transaction integrity and durability, particularly in failure scenarios. This foundational concept sets the stage for the introduction of the ARIES recovery algorithm.

Algorithms for **R**ecovery and **I**solation **E**xploiting **S**emantics (ARIES) is a recovery algorithm developed at IBM research in early 1990s for the DB2 system.

There are three key concepts in the ARIES recovery protocol:

- **Write Ahead Logging:** Any change is recorded in log on stable storage before the database change is written to disk (STEAL + NO-FORCE).
- **Repeating History During Redo:** On restart, retrace actions and restore database to exact state before crash.
- **Logging Changes During Undo:** Record undo actions to log to ensure action is not repeated in the event of repeated failures.

2 WAL Records

Write-ahead log records extend the DBMS's log record format to include a globally unique *log sequence number* (LSN). All log records have an LSN. A high level diagram of how log records with LSN's are written is shown in Figure 1.

Each data page contains a `pageLSN`, which is the LSN of the most recent update to that page. The `pageLSN` is updated every time a transaction modifies a record in the page.

The DBMS also keeps track of the max *LSN* flushed so far (`flushedLSN`). The `flushedLSN` in memory is updated every time the DBMS writes out the WAL buffer to disk.

Various components in the system keep track of **LSNs** that pertain to them. Section 2 shows a summary of these LSNs.

3 Normal Execution

Every transaction invokes a sequence of reads and writes, followed by a commit or abort. It is this sequence of events that recovery algorithms must have.

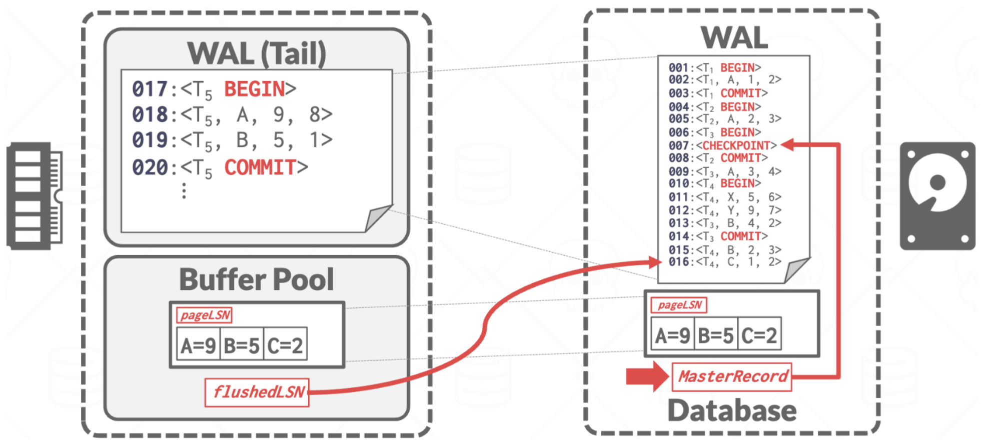


Figure 1: Writing Log Records – Each WAL has a counter of LSNs that is incremented at every step. The page also keeps a pageLSN, which stores the most recent log record that updated this page. The flushedLSN is a pointer to the last LSN that was written out to disk. Before the DBMS can write page i to disk, it must flush log at least to the point where $\texttt{pageLSN}_i \leq \texttt{flushedLSN}$. The MasterRecord points to the last successful checkpoint passed.

Name	Location	Definition
flushedLSN	Memory	Last LSN in log on disk
pageLSN	page$_x$	Newest update to page$_x$
recLSN	Dirty Page Table	Oldest update to pagex since it was last flushed
lastLSN	Active Transaction Table	Latest record of txn T_i (managed by transaction)
MasterRecord	Disk	LSN of latest checkpoint

Table 1: Log Sequence Number Types – A list of the different LSNs that a DBMS maintains in its internal components and data structures.

Transaction Commit

When a transaction goes to commit, the DBMS first writes `COMMIT` record to log buffer in memory. Then the DBMS flushes all log records up to and including the transaction's `COMMIT` record to disk. Note that these log flushes are sequential, synchronous writes to disk. There can be multiple log records per log page. A diagram of a transaction commit is shown in Figure 2.

Once the `COMMIT` record is safely stored on disk, the DBMS returns an acknowledgment back to the application that the transaction has committed. At some later point, the DBMS will write a special `TXN-END` record to log. This indicates that the transaction is completely finished in the system and there will not be anymore log records for it. These `TXN-END` records are used for internal bookkeeping and do not need to be flushed immediately.

Transaction Abort

Aborting a transaction is a special case of the ARIES undo operation applied to only one transaction.

An additional field is added to the log records called the `prevLSN`. This corresponds to the previous LSN for the transaction. The DBMS uses these `prevLSN` values to maintain a linked-list for each transaction that makes it easier to walk through the log to find its records.

A new type of record called the *compensation log record* (CLR) is also introduced. A CLR describes the

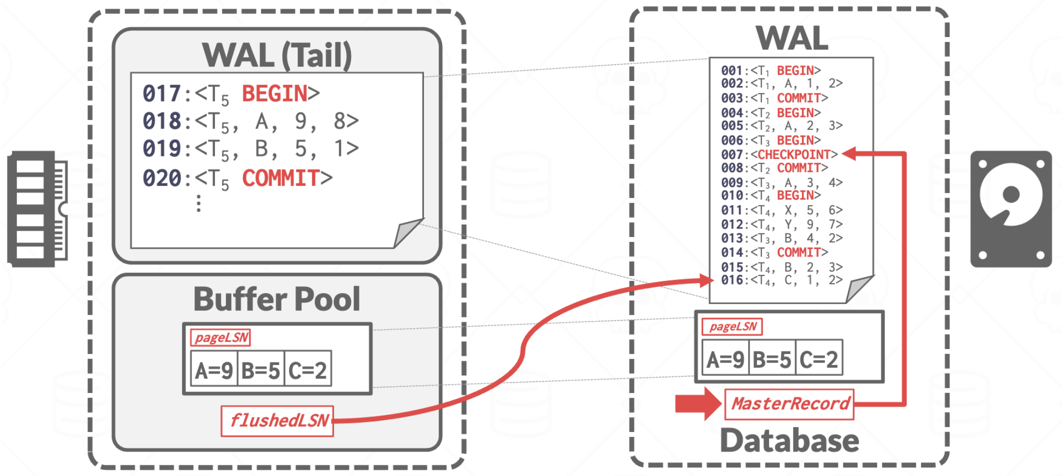


Figure 2: Transaction Commit – After the transaction commits (015), the log is flushed out and the `flushedLSN` is modified to point to the last log record generated. At some later point, a transaction end message is written to signify in the log that this transaction will not appear again. Then we can trim the in-memory log up to flushedLSN.

actions taken to undo the actions of a previous update record. It has all the fields of an update log record plus the *undoNextLSN* pointer (i.e., the next-to-be-undone LSN). The DBMS adds CLRs to the log like any other record but they never need to be undone. Moreover, the DBMS does not wait for CLRs to be flushed to disk before notifying the application that the transaction has been aborted. This approach ensures efficient transaction management, especially in scenarios involving transaction rollbacks.

To abort a transaction, the DBMS first appends a `ABORT` record to the log buffer in memory. It then undoes the transaction's updates in reverse order to remove their effects from the database. For each undone update, the DBMS creates **CLR** entry in the log and restore old value. After all of the aborted transaction's updates are reversed, the DBMS then writes a `TXN-END` log record. A diagram of this is shown in Figure 3.

4 Checkpointing

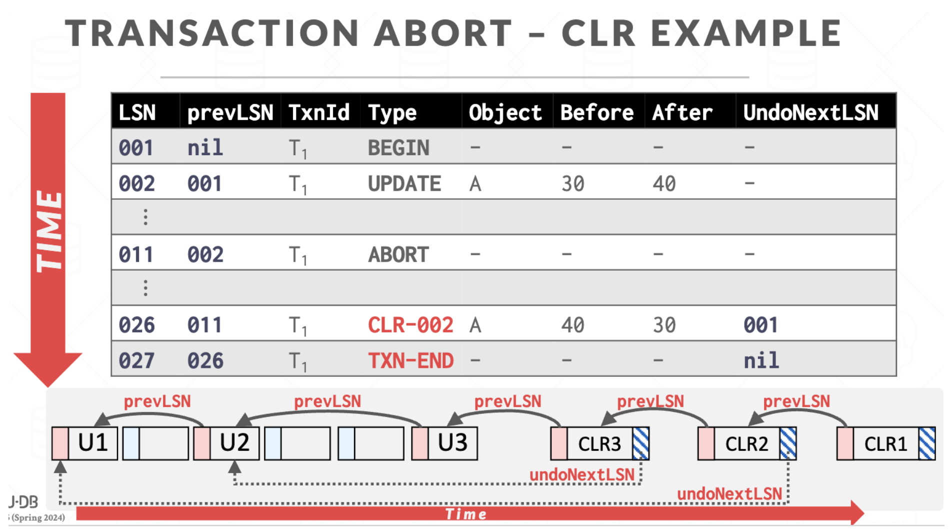
The DBMS periodically takes *checkpoints* where it writes the dirty pages in its buffer pool out to disk. This is used to minimize how much of the log it has to replay upon recovery.

The first two blocking checkpoint methods discussed below pause transactions during the checkpoint process. This pausing is necessary to ensure that the DBMS does not miss updates to pages during the checkpoint. Then, a better approach that allows transactions to continue to execute during the checkpoint but requires the DBMS to record additional information to determine what updates it may have missed is presented.

Non-Fuzzy Checkpoints

The DBMS halts the execution of transactions and queries when it takes a checkpoint to ensure that it writes a consistent snapshot of the database to disk. The is the same approach discussed in previous lecture:

- Halt the start of any new transactions.
- Wait until all active transactions finish executing.
- Flush dirty pages to disk.



TRANSACTION ABORT – CLR EXAMPLE

LSN	prevLSN	TxnId	Type	Object	Before	After	UndoNextLSN
001	nil	T_1	BEGIN	-	-	-	-
002	001	T_1	UPDATE	A	30	40	-
⋮							
011	002	T_1	ABORT	-	-	-	-
⋮							
026	011	T_1	CLR-002	A	40	30	001
027	026	T_1	TXN-END	-	-	-	nil

Figure 3: Transaction Abort – The DBMS maintains an LSN and prevLSN for each log record that the transaction creates. When the transaction aborts, all of the previous changes are reversed. After the log entries of the reversed changes make it to disk, the DBMS appends the TXN-END record to the log for the aborted transaction.

While this process impacts runtime performance, it significantly simplifies recovery.

Slightly Better Blocking Checkpoints

Like previous checkpoint scheme except that you the DBMS does not have to wait for active transactions to finish executing. The DBMS now records the internal system state as of the beginning of the checkpoint.

- Halt the start of any new transactions.
- Pause transactions while the DBMS takes the checkpoint.

The checkpoint process requires recording the internal state at its commencement. This includes two key components: the Active Transaction Table (ATT), which tracks ongoing transactions, and the Dirty Page Table (DPT), which lists all modified pages not yet written to disk.

Active Transaction Table (ATT): The ATT represents the state of transactions that are actively running in the DBMS. A transaction's entry is removed after the DBMS completes the commit/abort process for that transaction. For each transaction entry, the ATT contains the following information:

- `transactionId`: Unique transaction identifier
- `status`: The current "mode" of the transaction (Running, Committing, Undo Candidate).
- `lastLSN`: Most recent LSN written by transaction

Note that the ATT contains every transcation without the `TXN-END` log record. This includes both transactions that are either committed or abort.

Dirty Page Table (DPT): The DPT contains information about the pages in the buffer pool that were modified by uncommitted transactions. There is one entry per dirty page containing the `recLSN` (i.e., the LSN of the log record that first caused the page to be dirty).

The DPT contains all pages that are dirty in the buffer pool. It doesn't matter if the changes were caused by a transaction that is running, committed, or aborted.

Overall, the ATT and DPT are vital in both checkpointing and recovery processes. During checkpointing, they capture the database's current state, with the ATT tracking active transactions and the DPT listing unflushed modified pages. In recovery, such as with the ARIES protocol, these tables aid in restoring the database to its consistent pre-crash state.

Fuzzy Checkpoints

A *fuzzy checkpoint* is where the DBMS allows other transactions to continue to run. This is what ARIES uses in its protocol.

The DBMS uses additional log records to track checkpoint boundaries:

- `<CHECKPOINT-BEGIN>`: Indicates the start of the checkpoint. At this point, the DBMS takes a snapshot of the current ATT and DPT, which are referenced in the `<CHECKPOINT-END>` record. Transactions that start after the checkpoint initiation are not included in the ATT.
- `<CHECKPOINT-END>`: When the checkpoint has completed. It contains the ATT + DPT, captured just as the `<CHECKPOINT-BEGIN>` log record is written.

Upon the completion of the checkpoint, the LSN of the `<CHECKPOINT-BEGIN>` record is recorded in the MasterRecord.

Fuzzy checkpoints strike a balance between performance and recoverability by minimizing transaction disruption while still capturing sufficient recovery information.

5 ARIES Recovery

The ARIES protocol is comprised of three phases. Upon start-up after a crash, the DBMS will execute the following phases as shown in Figure 4:

1. **Analysis**: Read the WAL to identify dirty pages in the buffer pool and active transactions at the time of the crash. At the end of the analysis phase the *ATT* tells the DBMS which transactions were active at the time of the crash. The *DPT* tells the DBMS which dirty pages might not have made it to disk.
2. **Redo**: Repeat all actions starting from an appropriate point in the log (even txns that will abort).
3. **Undo**: Reverse the actions of transactions that did not commit before the crash.

Analysis Phase

Start from last checkpoint found via the database's `MasterRecord` *LSN*.

1. Scan log forward from the checkpoint.
2. If the DBMS finds a `TXN-END` record, remove its transaction from ATT.
3. All other records, add transaction to ATT with status **UNDO**, and on commit, change transaction status to **COMMIT**.
4. For `UPDATE` log records, if page P is not in the DPT, then add P to DPT and set P's `recLSN` to the log record's *LSN*.

Redo Phase

The goal of this phase is for the DBMS to repeat history to reconstruct its state up to the moment of the crash. It will reapply all updates (even aborted transactions) and redo **CLRs**.

The DBMS scans forward from log record containing smallest `recLSN` in the DPT. For each update log record or CLR with a given *LSN*, the DBMS re-applies the update unless:

- Affected page is not in the DPT, or

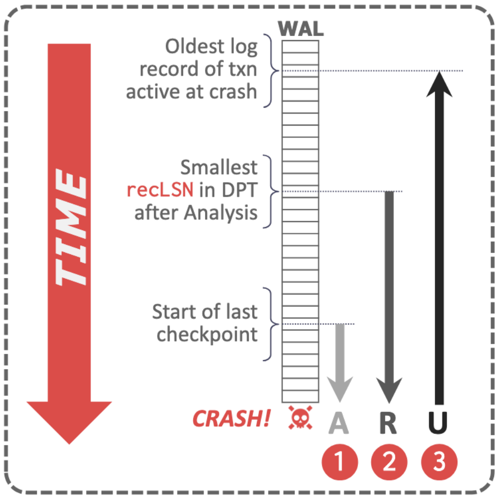

WAL
TIME
Oldest log record of txn active at crash
Smallest recLSN in DPT after Analysis
Start of last checkpoint
CRASH!
A
R
U
1
2
3

Figure 4: ARIES Recovery: The DBMS starts the recovery process by examining the log starting from the last `BEGIN-CHECKPOINT` found via `MasterRecord`. It then begins the Analysis phase by scanning forward through time to build out ATT and DPT. In the Redo phase, the algorithm jumps to the smallest recLSN, which is the oldest log record that may have modified a page not written to disk. The DBMS then applies all changes from the smallest recLSN. The Undo phase starts at the oldest log record of a transaction active at crash and reverses all changes up to that point.

- Affected page is in DPT but that log record's LSN is less than the page's `recLSN`, (the update was propagated to disk), or
- Affected `pageLSN` (on disk) $\geq$ *LSN*. (Note, we must fetch the page from the disk to read the page value.)

To redo an action, the DBMS re-applies the change in the log record and then sets the affected page's *pageLSN* to that log record's *LSN*. Also, there is no additional logging or forced flushes.

At the end of the redo phase, write `TXN-END` log records for all transactions with status COMMIT and remove them from the ATT.

Undo Phase

In the last phase, the DBMS reverses all transactions that were active at the time of crash. These are all transactions with UNDO status in the ATT after the Analysis phase.

The DBMS processes transactions in reverse *LSN* order using the `lastLSN` to speed up traversal. At each step, pick the largest lastLSN across all transactions in the ATT. As it reverses the updates of a transaction, the DBMS writes a CLR entry to the log for each modification.

Once the last transaction has been successfully aborted, the DBMS flushes out the log and then is ready to start processing new transactions.