Database 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.
ARIES

Algorithms for Recovery and Isolation Exploiting Semantics

Developed at IBM Research in early 1990s for the DB2 DBMS.

Not all systems implement ARIES exactly as defined in this paper but they're close enough.
ARIES – MAIN IDEAS

**Write-Ahead Logging:**
→ Any change is recorded in log on stable storage before the database change is written to disk.
→ Must use **STEAL + NO-FORCE** buffer pool policies.

**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.
TODAY’S AGENDA

Log Sequence Numbers
Normal Commit & Abort Operations
Fuzzy Checkpointing
Recovery Algorithm
We need to extend our log record format from last class to include additional info.

Every log record now includes a globally unique \textit{log sequence number} (LSN).

Various components in the system keep track of \textit{LSNs} that pertain to them...
## LOG SEQUENCE NUMBERS

<table>
<thead>
<tr>
<th>Name</th>
<th>Where</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>flushedLSN</td>
<td>Memory</td>
<td>Last LSN in log on disk</td>
</tr>
<tr>
<td>pageLSN</td>
<td>page_x</td>
<td>Newest update to page_x</td>
</tr>
<tr>
<td>recLSN</td>
<td>page_x</td>
<td>Oldest update to page_x since it was last flushed</td>
</tr>
<tr>
<td>lastLSN</td>
<td>T_i</td>
<td>Latest record of txn T_i</td>
</tr>
<tr>
<td>MasterRecord</td>
<td>Disk</td>
<td>LSN of latest checkpoint</td>
</tr>
</tbody>
</table>
WRITING LOG RECORDS

Each data page contains a pageLSN.
→ The LSN of the most recent update to that page.

System keeps track of flushedLSN.
→ The max LSN flushed so far.

Before page $x$ can be written to disk, we must flush log at least to the point where:
→ $\text{pageLSN}_x \leq \text{flushedLSN}$
WRITING LOG RECORDS

Log Sequence Numbers

<table>
<thead>
<tr>
<th>LSN</th>
<th>Log Record</th>
</tr>
</thead>
<tbody>
<tr>
<td>017</td>
<td>&lt;T₅, BEGIN&gt;</td>
</tr>
<tr>
<td>018</td>
<td>&lt;T₅, A, 9, 8&gt;</td>
</tr>
<tr>
<td>019</td>
<td>&lt;T₅, B, 5, 1&gt;</td>
</tr>
<tr>
<td>020</td>
<td>&lt;T₅, COMMIT&gt;</td>
</tr>
</tbody>
</table>

Buffer Pool

MasterRecord flushedLSN

<table>
<thead>
<tr>
<th>pageLSN</th>
<th>recLSN</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>&lt;T₁, BEGIN&gt;</td>
<td>9</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>002</td>
<td>&lt;T₁, A, 1, 2&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>003</td>
<td>&lt;T₁, COMMIT&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>004</td>
<td>&lt;T₂, BEGIN&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>005</td>
<td>&lt;T₂, A, 2, 3&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>006</td>
<td>&lt;T₂, COMMIT&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>007</td>
<td>&lt;CHECKPOINT&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>008</td>
<td>&lt;T₃, A, 3, 4&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>009</td>
<td>&lt;T₃, COMMIT&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>&lt;T₄, BEGIN&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>011</td>
<td>&lt;T₄, X, 5, 6&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>012</td>
<td>&lt;T₄, Y, 9, 7&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>013</td>
<td>&lt;T₃, B, 4, 2&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>014</td>
<td>&lt;T₃, COMMIT&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>015</td>
<td>&lt;T₄, B, 2, 3&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>016</td>
<td>&lt;T₄, C, 1, 2&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Log Sequence Numbers

<table>
<thead>
<tr>
<th>LSN</th>
<th>Log Record</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>&lt;T₁, BEGIN&gt;</td>
</tr>
<tr>
<td>002</td>
<td>&lt;T₁, A, 1, 2&gt;</td>
</tr>
<tr>
<td>003</td>
<td>&lt;T₁, COMMIT&gt;</td>
</tr>
<tr>
<td>004</td>
<td>&lt;T₂, BEGIN&gt;</td>
</tr>
<tr>
<td>005</td>
<td>&lt;T₂, A, 2, 3&gt;</td>
</tr>
<tr>
<td>006</td>
<td>&lt;T₂, COMMIT&gt;</td>
</tr>
<tr>
<td>007</td>
<td>&lt;CHECKPOINT&gt;</td>
</tr>
<tr>
<td>008</td>
<td>&lt;T₃, A, 3, 4&gt;</td>
</tr>
<tr>
<td>009</td>
<td>&lt;T₃, COMMIT&gt;</td>
</tr>
<tr>
<td>010</td>
<td>&lt;T₄, BEGIN&gt;</td>
</tr>
<tr>
<td>011</td>
<td>&lt;T₄, X, 5, 6&gt;</td>
</tr>
<tr>
<td>012</td>
<td>&lt;T₄, Y, 9, 7&gt;</td>
</tr>
<tr>
<td>013</td>
<td>&lt;T₃, B, 4, 2&gt;</td>
</tr>
<tr>
<td>014</td>
<td>&lt;T₃, COMMIT&gt;</td>
</tr>
<tr>
<td>015</td>
<td>&lt;T₄, B, 2, 3&gt;</td>
</tr>
<tr>
<td>016</td>
<td>&lt;T₄, C, 1, 2&gt;</td>
</tr>
</tbody>
</table>

Not safe to unpin because pageLSN > flushedLSN
Writing Log Records

All log records have an LSN.

Update the pageLSN every time a txn modifies a record in the page.

Update the flushedLSN in memory every time the DBMS writes out the WAL buffer to disk.
NORMAL EXECUTION

Each txn invokes a sequence of reads and writes, followed by commit or abort.

Assumptions in this lecture:
→ All log records fit within a single page.
→ Disk writes are atomic.
→ Single-versioned tuples with Strict 2PL.
→ **STEAL + NO-FORCE** buffer management with WAL.
TRANSACTION COMMIT

Write **COMMIT** record to log.

All log records up to txn’s **COMMIT** record are flushed to disk.
→ Note that log flushes are sequential, synchronous writes to disk.
→ Many log records per log page.

When the commit succeeds, write a special **TXN-END** record to log.
→ This does **not** need to be flushed immediately.
We can trim the in-memory log up to flushedLSN

Buffer Pool

flushedLSN = 015

pageLSN | recLSN
A=9 | B=5 | C=2

WAL

flushedLSN = 015

pageLSN | recLSN
A=9 | B=5 | C=2

MasterRecord

001: <T1, BEGIN>
002: <T1, A, 9, 8>
003: <T2, COMMIT>
004: <T2, BEGIN>
005: <T2, A, 2, 3>
006: <T3, BEGIN>
007: <CHECKPOINT>
008: <T3, COMMIT>
009: <T3, A, 3, 4>
010: <T3, B, 4, 2>
011: <T3, COMMIT>
012: <T4, BEGIN>
013: <T4, A, 9, 8>
014: <T4, B, 5, 1>
015: <T4, COMMIT>

...
TRANSACTION ABORT

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

We need to add another field to our log records:

→ **prevLSN**: The previous *LSN* for the txn.
→ This maintains a linked-list for each txn that makes it easy to walk through its records.
Important: Need to record what steps we took to undo the txn.
COMPENSATION LOG RECORDS

A CLR describes the actions taken to undo the actions of a previous update record.

It has all the fields of an update log record plus the undoNext pointer (the next-to-be-undone LSN).

CLRs are added to log like any other record.
# TRANSACTION ABORT – CLR EXAMPLE

<table>
<thead>
<tr>
<th>LSN</th>
<th>prevLSN</th>
<th>TxnId</th>
<th>Type</th>
<th>Object</th>
<th>Before</th>
<th>After</th>
<th>UndoNext</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>nil</td>
<td>T₁</td>
<td>BEGIN</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>002</td>
<td>001</td>
<td>T₁</td>
<td>UPDATE</td>
<td>A</td>
<td>30</td>
<td>40</td>
<td>-</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>011</td>
<td>002</td>
<td>T₁</td>
<td>ABORT</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
### TRANSACTION ABORT – CLR EXAMPLE

<table>
<thead>
<tr>
<th>LSN</th>
<th>prevLSN</th>
<th>TxnId</th>
<th>Type</th>
<th>Object</th>
<th>Before</th>
<th>After</th>
<th>UndoNext</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>nil</td>
<td>T_1</td>
<td>BEGIN</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>002</td>
<td>001</td>
<td>T_1</td>
<td>UPDATE</td>
<td>A</td>
<td>30</td>
<td>40</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>011</td>
<td>002</td>
<td>T_1</td>
<td>ABORT</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>026</td>
<td>011</td>
<td>T_1</td>
<td>CLR-002</td>
<td>A</td>
<td>40</td>
<td>30</td>
<td>001</td>
</tr>
</tbody>
</table>

The LSN of the next log record to be undone.
## TRANSACTION ABORT – CLR EXAMPLE

<table>
<thead>
<tr>
<th>LSN</th>
<th>prevLSN</th>
<th>TxnId</th>
<th>Type</th>
<th>Object</th>
<th>Before</th>
<th>After</th>
<th>UndoNext</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>nil</td>
<td>T₁</td>
<td>BEGIN</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>002</td>
<td>001</td>
<td>T₁</td>
<td>UPDATE</td>
<td>A</td>
<td>30</td>
<td>40</td>
<td>-</td>
</tr>
<tr>
<td>011</td>
<td>002</td>
<td>T₁</td>
<td>ABORT</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>026</td>
<td>011</td>
<td>T₁</td>
<td>CLR-002</td>
<td>A</td>
<td>40</td>
<td>30</td>
<td>001</td>
</tr>
<tr>
<td>027</td>
<td>026</td>
<td>T₁</td>
<td>TXN-END</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>nil</td>
</tr>
</tbody>
</table>

**TIME**

**TRANSACTION ABORT**

**CLR EXAMPLE**
ABORT ALGORITHM

First write an **ABORT** record to log for the txn.
Then play back the txn's updates in reverse order.
For each update record:
→ Write a **CLR** entry to the log.
→ Restore old value.
At end, write a **TXN-END** log record.

Notice: **CLR**s never need to be undone.
TODAY’S AGENDA

Log-Sequence Numbers
Normal Commit & Abort Operations
Fuzzy Checkpointing
Recovery Algorithm
NON-FUZZY CHECKPOINTS

The DBMS halts everything when it takes a checkpoint to ensure a consistent snapshot:
→ Halt the start of any new txns.
→ Wait until all active txns finish executing.
→ Flushes dirty pages on disk.

This is obviously bad...
SLIGHTLY BETTER CHECKPOINTS

Pause modifying txns while the DBMS takes the checkpoint.
→ Prevent queries from acquiring write latch on table/index pages.
→ Don't have to wait until all txns finish before taking the checkpoint.

We must record internal state as of the beginning of the checkpoint.
→ Active Transaction Table (ATT)
→ Dirty Page Table (DPT)
ACTIVE TRANSACTION TABLE

One entry per currently active txn.

→ **txnId**: Unique txn identifier.
→ **status**: The current "mode" of the txn.
→ **lastLSN**: Most recent *LSN* created by txn.

Entry removed when txn commits or aborts.

Txn Status Codes:

→ **R** → Running
→ **C** → Committing
→ **U** → Candidate for Undo
DIRTY PAGE TABLE

Keep track of which pages in the buffer pool contain changes from uncommitted transactions.

One entry per dirty page in the buffer pool:
→ recLSN: The LSN of the log record that first caused the page to be dirty.
SLIGHTLY BETTER CHECKPOINTS

At the first checkpoint, $T_2$ is still running and there are two dirty pages ($P_{11}, P_{22}$).

At the second checkpoint, $T_3$ is active and there are two dirty pages ($P_{11}, P_{33}$).

This still is not ideal because the DBMS must stall txns during checkpoint...
A fuzzy checkpoint is where the DBMS allows active txns to continue the run while the system flushes dirty pages to disk.

New log records to track checkpoint boundaries:

→ **CHECKPOINT-BEGIN**: Indicates start of checkpoint

→ **CHECKPOINT-END**: Contains ATT + DPT.
The LSN of the **CHECKPOINT-BEGIN** record is written to the database's **MasterRecord** entry on disk when the checkpoint successfully completes.

Anytxn that starts after the checkpoint is excluded from the ATT in the **CHECKPOINT-END** record.
ARIES – RECOVERY PHASES

Phase #1 – Analysis
→ Read WAL from last checkpoint to identify dirty pages in the buffer pool and active txns at the time of the crash.

Phase #2 – Redo
→ Repeat all actions starting from an appropriate point in the log (even txns that will abort).

Phase #3 – Undo
→ Reverse the actions of txns that did not commit before the crash.
ARIES – OVERVIEW

Start from last **BEGIN-CHECKPOINT** found via **MasterRecord**.

**Analysis**: Figure out which txns committed or failed since checkpoint.

**Redo**: Repeat **all** actions.

**Undo**: Reverse effects of failed txns.

**CRASH!**
ANALYSIS PHASE

Scan log forward from last successful checkpoint.
If you find a **TXN-END** record, remove its corresponding txn from **ATT**.

All other records:
→ Add txn to **ATT** with status **UNDO**.
→ On commit, change txn status to **COMMIT**.

For **UPDATE** records:
→ If page **P** not in **DPT**, add **P** to **DPT**, set its **recLSN=LSN**.
At end of the Analysis Phase:
→ **ATT** tells the DBMS which txns were active at time of crash.
→ **DPT** tells the DBMS which dirty pages might not have made it to disk.
ANALYSIS PHASE EXAMPLE

Modify A in page P_{33}

\begin{itemize}
  \item 010: \texttt{<CHECKPOINT-BEGIN>}
  \item 020: \texttt{<T_{96}, A\rightarrow P_{33}, 10, 15>}
  \item 030: \texttt{<CHECKPOINT-END>}
  \begin{itemize}
    \item ATT={T_{96}, T_{97}},
    \item DPT={P_{20}, P_{33}}
  \end{itemize}
  \item 040: \texttt{<T_{96} COMMIT>}
  \item 050: \texttt{<T_{96} TXN-END>}
\end{itemize}

\begin{itemize}
  \item CRASH!
\end{itemize}

\begin{itemize}
  \item 010
  \item 020
  \item 030
  \item 040
  \item 050
\end{itemize}

\begin{itemize}
  \item (TxnId, Status)
  \item (PageId, RecLSN)
\end{itemize}
REDO PHASE

The goal is to repeat history to reconstruct state at the moment of the crash:
→ Reapply all updates (even aborted txns!) and redo CLRs.

There techniques that allow the DBMS to avoid unnecessary reads/writes, but we will ignore that in this lecture...
REDO PHASE

Scan forward from the log record containing smallest recLSN in DPT.

For each update log record or CLR with a given LSN, redo the action unless:
→ Affected page is not in DPT, or
→ Affected page is in DPT but that record's LSN is less than the page's recLSN.
To redo an action:
→ Reapply logged action.
→ Set `pageLSN` to log record's `LSN`.
→ No additional logging, no forced flushes!

At the end of Redo Phase, write `TXN-END` log records for all txns with status `C` and remove them from the `ATT`.
**UNDO PHASE**

Undo all txns that were active at the time of crash and therefore will never commit.
→ These are all the txns with **U** status in the **ATT** after the Analysis Phase.

Process them in reverse **LSN** order using the **lastLSN** to speed up traversal.
Write a **CLR** for every modification.
### FULL EXAMPLE

- **LOG**
  - LSN 00: `<CHECKPOINT-BEGIN>`
  - LSN 05: `<CHECKPOINT-END>`
  - LSN 10: `<T₁, A→P₅, 1, 2>`
  - LSN 20: `<T₂, B→P₃, 2, 3>`
  - LSN 30: `<T₁ ABORT>`
  - LSN 40: `<CLR: Undo T₁ LSN 10>`
  - LSN 45: `<T₁ TXN-†END>`
  - LSN 50: `<T₃, C→P₁, 4, 5>`
  - LSN 60: `<T₂, D→P₅, 6, 7>`

- **LSN**
  - LSN 00: 00
  - LSN 05: 05
  - LSN 10: 10
  - LSN 20: 20
  - LSN 30: 30
  - LSN 40: 40
  - LSN 45: 45
  - LSN 50: 50
  - LSN 60: 60

- **TIME**
  - PrevLSNs: Crash!
FULL EXAMPLE

LSN  LOG
00,05  <CHECKPOINT-BEGIN>, <CHECKPOINT-END>
10  <T₁, A→P₅, 1, 2>
20  <T₂, B→P₃, 2, 3>
30  <T₁ ABORT>
40,45  <CLR: Undo T₁ LSN 10>, <T₁ TXN-END>
50  <T₃, C→P₁, 4, 5>
60  <T₂, D→P₅, 6, 7>
  CRASH! RESTART!
70  <CLR: Undo T₂ LSN 60, UndoNext
80,85  <CLR: Undo T₃ LSN 50>, <T₃ TXN-END>
  CRASH! RESTART!

Flush dirty pages + WAL to disk!
FULL EXAMPLE

LSN

00,05  <CHECKPOINT-BEGIN>, <CHECKPOINT-END>
10  <T₁, A→P₅, 1, 2>
20  <T₂, B→P₃, 2, 3>
30  <T₁ ABORT>
40,45  <CLR: Undo T₁ LSN 10>, <T₁ TXN-END>
50  <T₃, C→P₁, 4, 5>
60  <T₂, D→P₅, 6, 7>
70  <CLR: Undo T₂ LSN 60, UndoNext 20>
80,85  <CLR: Undo T₃ LSN 50>, <T₃ TXN-END>
90,95  <CLR: Undo T₂ LSN 20>, <T₂ TXN-END>

LOG

ATN

<table>
<thead>
<tr>
<th>TxnId</th>
<th>Status</th>
<th>LastLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>T₂</td>
<td>U</td>
<td>70</td>
</tr>
</tbody>
</table>

DPT

<table>
<thead>
<tr>
<th>PageId</th>
<th>recLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>50</td>
</tr>
<tr>
<td>P₃</td>
<td>08</td>
</tr>
<tr>
<td>P₅</td>
<td>10</td>
</tr>
</tbody>
</table>

FlushedLSN

CRASH! RESTART!

CRASH! RESTART!
What does the DBMS do if it crashes during recovery in the Analysis Phase?
→ Nothing. Just run recovery again.

What does the DBMS do if it crashes during recovery in the Redo Phase?
→ Again nothing. Redo everything again.
How can the DBMS improve performance during recovery in the Redo Phase?
→ Assume that it is not going to crash again and flush all changes to disk asynchronously in the background.

How can the DBMS improve performance during recovery in the Undo Phase?
→ Lazily rollback changes before new txns access pages.
→ Rewrite the application to avoid long-running txns.
CONCLUSION

Mains ideas of ARIES:
→ WAL with **STEAL/NO-FORCE**
→ Fuzzy Checkpoints (snapshot of dirty page ids)
→ Redo everything since the earliest dirty page
→ Undo txns that never commit
→ Write **CLR**s when undoing, to survive failures during restarts

Log Sequence Numbers:
→ **LSN**s identify log records; linked into backwards chains per transaction via \(\text{prevLSN}\).
→ **pageLSN** allows comparison of data page and log records.
NEXT CLASS

You now know how to build a single-node DBMS.

So now we can talk about distributed databases!