Intro to Database Systems (15-445/645)

20 Database Recovery

Carnegie Mellon University
FALL 2022
Andy Pavlo
Project #3 is due **Wed Nov 16\(^{th}\) @ 11:59pm**

Project #4 is due **Sun Dec 11\(^{th}\) @ 11:59pm**
→ Zoom Q&A Session **Thu Nov 17\(^{th}\) @ 8:00pm**

We are looking for spirited and impressionable TAs for 15-445/645 in Spring 2023.
→ All BusTub projects will remain in C++.
→ I will announce this on Piazza.
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.
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 DBMS 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 log sequence number (LSN).
→ LSNs represent the physical order that txns make changes to the database.

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Location</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>flushedLSN</code></td>
<td>Memory</td>
<td>Last LSN in log on disk</td>
</tr>
<tr>
<td><code>pageLSN</code></td>
<td><code>page_x</code></td>
<td>Newest update to <code>page_x</code></td>
</tr>
<tr>
<td><code>recLSN</code></td>
<td><code>page_x</code></td>
<td>Oldest update to <code>page_x</code> since it was last flushed</td>
</tr>
<tr>
<td><code>lastLSN</code></td>
<td><code>T_i</code></td>
<td>Latest record of txn <code>T_i</code></td>
</tr>
<tr>
<td><code>MasterRecord</code></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 the DBMS can write page x to disk, it must flush the log at least to the point where:
→ pageLSNₓ ≤ flushedLSN
Writing Log Records

Log Sequence Numbers

<table>
<thead>
<tr>
<th>Log Sequence Numbers</th>
<th>Log Sequence Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>017 &lt;T₅ BEGIN&gt;</td>
<td>001 &lt;T₁ BEGIN&gt;</td>
</tr>
<tr>
<td>018 &lt;T₅, A, 9, 8&gt;</td>
<td>002 &lt;T₁, A, 1, 2&gt;</td>
</tr>
<tr>
<td>019 &lt;T₅, B, 5, 1&gt;</td>
<td>003 &lt;T₁ COMMIT&gt;</td>
</tr>
<tr>
<td>020 &lt;T₅ COMMIT&gt;</td>
<td>004 &lt;T₂ BEGIN&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Buffer Pool</th>
<th>Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>pageLSN recLSN A=9 B=5 C=2</td>
<td>pageLSN recLSN A=9 B=5 C=2</td>
</tr>
</tbody>
</table>

flushedLSN

MasterRecord
WRITING LOG RECORDS

WAL (Tail)

017: <T5 BEGIN>
018: <T5, A, 9, 8>
019: <T5, B, 5, 1>
020: <T5 COMMIT>

Buffer Pool

pageLSN recLSN
A=9 B=5 C=2

flushedLSN

WAL

001: <T1 BEGIN>
002: <T1, A, 1, 2>
003: <T1 COMMIT>
004: <T2 BEGIN>
005: <T2, A, 2, 3>
006: <T2 COMMIT>
007: <CHECKPOINT>
008: <T3 BEGIN>
009: <T3, A, 3, 4>
010: <T3 COMMIT>
011: <T4, X, 5, 6>
012: <T4, Y, 9, 7>
013: <T3, B, 4, 2>
014: <T3 COMMIT>
015: <T4, B, 2, 3>
016: <T4, C, 1, 2>

Database

pageLSN recLSN
A=9 B=5 C=2

MasterRecord

flushedLSN
**WRITING LOG RECORDS**

**WAL (Tail)**

- 017: `<T₅ BEGIN>`
- 018: `<T₅, A, 9, 8>`
- 019: `<T₅, B, 5, 1>`
- 020: `<T₅ COMMIT>`

**Buffer Pool**

```
<table>
<thead>
<tr>
<th>pageLSN</th>
<th>recLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>A=9</td>
<td>B=5</td>
</tr>
<tr>
<td>C=2</td>
<td></td>
</tr>
</tbody>
</table>
```

**WAL**

```
001: `<T₁ BEGIN>`
002: `<T₁, A, 1, 2>`
003: `<T₁ COMMIT>`
004: `<T₂ BEGIN>`
005: `<T₂, A, 2, 3>`
006: `<T₁ BEGIN>`
007: `<CHECKPOINT>`
008: `<T₂ COMMIT>`
009: `<T₃, A, 3, 4>`
010: `<T₄ BEGIN>`
011: `<T₄, X, 5, 6>`
012: `<T₄, Y, 9, 7>`
013: `<T₃, B, 4, 2>`
014: `<T₃ COMMIT>`
015: `<T₄, B, 2, 3>`
016: `<T₄, C, 1, 2>`
```

**Database**

```
<table>
<thead>
<tr>
<th>pageLSN</th>
<th>recLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>A=9</td>
<td>B=5</td>
</tr>
<tr>
<td>C=2</td>
<td></td>
</tr>
</tbody>
</table>
```

flushedLSN: `A=9 B=5 C=2`
WRITING LOG RECORDS

WAL (Tail)

017: <T<sub>5</sub> BEGIN>
018: <T<sub>5</sub>, A, 9, 8>
019: <T<sub>5</sub>, B, 5, 1>
020: <T<sub>5</sub> COMMIT>

Buffer Pool

pageLSN  recLSN
A=9  B=5  C=2

flashedLSN

WAL

001: <T<sub>1</sub> BEGIN>
002: <T<sub>1</sub>, A, 1, 2>
003: <T<sub>1</sub> COMMIT>
004: <T<sub>2</sub> BEGIN>
005: <T<sub>2</sub>, A, 2, 3>
006: <T<sub>2</sub> COMMIT>
007: <CHECKPOINT>
008: <T<sub>3</sub> COMMIT>
009: <T<sub>3</sub>, A, 3, 4>
010: <T<sub>4</sub> BEGIN>
011: <T<sub>4</sub>, X, 5, 6>
012: <T<sub>4</sub>, Y, 9, 7>
013: <T<sub>3</sub>, B, 4, 2>
014: <T<sub>3</sub> COMMIT>
015: <T<sub>4</sub>, B, 2, 3>
016: <T<sub>4</sub>, C, 1, 2>

Database

pageLSN  recLSN
A=9  B=5  C=2

MasterRecord

flushedLSN

pageLSN  recLSN
A=9  B=5  C=2

MasterRecord
WRITING LOG RECORDS

WAL (Tail)

017: <T₅ BEGIN>
018: <T₅, A, 9, 8>
019: <T₅, B, 5, 1>
020: <T₅ COMMIT>

Buffer Pool

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

WAL

001: <T₁ BEGIN>
002: <T₁, A, 1, 2>
003: <T₁ COMMIT>
004: <T₂ BEGIN>
005: <T₂, A, 2, 3>
006: <T₂ COMMIT>
007: <CHECKPOINT>
008: <T₂ COMMIT>
009: <T₃, A, 3, 4>
010: <T₄ BEGIN>
011: <T₄, X, 5, 6>
012: <T₄, Y, 9, 7>
013: <T₃, B, 4, 2>
014: <T₃ COMMIT>
015: <T₄, B, 2, 3>
016: <T₄, C, 1, 2>

MasterRecord

pageLSN recLSN
A=9 B=5 C=2
flushedLSN
**WRITING LOG RECORDS**

**WAL (Tail)**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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**

Safe to evict because pageLSN ≤ flushedLSN

**WAL**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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;T₁, BEGIN&gt;</td>
</tr>
<tr>
<td>007:</td>
<td>&lt;CHECKPOINT&gt;</td>
</tr>
<tr>
<td>008:</td>
<td>&lt;T₂, COMMIT&gt;</td>
</tr>
<tr>
<td>009:</td>
<td>&lt;T₃, A, 3, 4&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>

**Database**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>pageLSN</td>
<td>recLSN</td>
</tr>
<tr>
<td>A=9</td>
<td>B=5</td>
</tr>
</tbody>
</table>

**MasterRecord**

flushedLSN
**WRITING LOG RECORDS**

### WAL (Tail)

- 017: \(<T_5\ \text{BEGIN}>\)
- 018: \(<T_5, A, 9, 8>\)
- 019: \(<T_5, B, 5, 1>\)
- 020: \(<T_5\ \text{COMMIT}>\)

### Buffer Pool

- **Not safe to evict because pageLSN > flushedLSN**

### WAL

- 001: \(<T_1\ \text{BEGIN}>\)
- 002: \(<T_1, A, 1, 2>\)
- 003: \(<T_1\ \text{COMMIT}>\)
- 004: \(<T_2\ \text{BEGIN}>\)
- 005: \(<T_2, A, 2, 3>\)
- 006: \(<T_1\ \text{BEGIN}>\)
- 007: \(<\text{CHECKPOINT}>\)
- 008: \(<T_2\ \text{COMMIT}>\)
- 009: \(<T_2, A, 3, 4>\)
- 010: \(<T_4\ \text{BEGIN}>\)
- 011: \(<T_4, X, 5, 6>\)
- 012: \(<T_4, Y, 9, 7>\)
- 013: \(<T_3, B, 4, 2>\)
- 014: \(<T_3\ \text{COMMIT}>\)
- 015: \(<T_4, B, 2, 3>\)
- 016: \(<T_4, C, 1, 2>\)

### Database

- pageLSN
- recLSN
- A=9, B=5, C=2

- **MasterRecord**

- **flushedLSN**

- **A=9, B=5, C=2**
WRITING LOG RECORDS

All log records have an \textit{LSN}.

Update the \textit{pageLSN} every time a txn modifies a record in the page.

Update the \textit{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 Strong Strict 2PL.
→ STEAL + NO-FORCE buffer management with WAL.
TRANSACTION COMMIT

When a txn commits, the DBMS writes a COMMIT record to log and guarantees that all log records up to txn's COMMIT record are flushed to disk.
→ 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.
→ Indicates that no new log record for a txn will appear in the log ever again.
→ This does not need to be flushed immediately.
TRANSACTION COMMIT

WAL (Tail)
012: <T₄, BEGIN>
013: <T₄, A, 9, 8>
014: <T₄, B, 5, 1>
015: <T₄, COMMIT>

Buffer Pool
pageLSN  recLSN
A=9  B=5  C=2
flushedLSN

WAL
001: <T₁, BEGIN>
002: <T₁, A, 1, 2>
003: <T₁, COMMIT>
004: <T₂, BEGIN>
005: <T₂, A, 2, 3>
006: <T₁, BEGIN>
007: <CHECKPOINT>
008: <T₂, COMMIT>
009: <T₃, A, 3, 4>
010: <T₃, B, 4, 2>
011: <T₃, COMMIT>

Database
pageLSN  recLSN
flushedLSN
A=9  B=5  C=2
MasterRecord
 TRANSACTION COMMIT

WAL (Tail)

012:<T4 BEGIN>
013:<T4, A, 9, 8>
014:<T4, B, 5, 1>
015:<T4 COMMIT>

Buffer Pool

flushedLSN

pageLSN recLSN
A=9 B=5 C=2

Database

flushedLSN = 015

WAL

001:<T1 BEGIN>
002:<T1, A, 1, 2>
003:<T1 COMMIT>
004:<T2 BEGIN>
005:<T2, A, 2, 3>
006:<T1 BEGIN>
007:<CHECKPOINT>
008:<T2 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>

MasterRecord

pageLSN recLSN
A=9 B=5 C=2
We can trim the in-memory log up to flushedLSN.
We can trim the in-memory log up to flushedLSN
Aborting a txn is a special case of the ARIES undo operation applied to only one txn.

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.
**TRANSACTION ABORT**

**Buffer Pool**
- Page 012: nil
- Page 013: 012
- Page 014: 013

**WAL (Tail)**
- LSN: 012
  - prevLSN: nil
- LSN: 013
  - prevLSN: 012
- LSN: 014
  - prevLSN: 013

**Database**
- Page LSN: A=9, B=5, C=2
- Rec LSN: A=9, B=5, C=2

**WAL**
- Page LSN: A=9, B=5, C=2
- Rec LSN: A=9, B=5, C=2

**Notes:**
- BEGIN
- T4
- A, 9, 8
- T4, B, 5, 1
- flushedLSN
- pageLSN
- recLSN
**TRANSACTION ABORT**

**WAL (Tail)**

```
012|nil:<T_4 BEGIN>
013|012:<T_4, A, 9, 8>
014|013:<T_4, B, 5, 1>
015|014:<T_4 ABORT>
???
099|098:<T_4 TXN-END>
```
Important: Need to record what steps we took to undo the txn.
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 records but the DBMS does **not** wait for them to be flushed before notifying the application that the txn aborted.
### 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>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>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>
# 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>
<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>T₁</td>
<td>CLR-002</td>
<td>A</td>
<td>40</td>
<td>30</td>
<td>001</td>
</tr>
</tbody>
</table>

TIME

- CLR EXAMPLE

Transaction Abort - CLR-002

- CLR Example

- CLR Example
## 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>Undo</th>
<th>Next</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>
<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>
<td>-</td>
</tr>
<tr>
<td></td>
<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>
<td>-</td>
</tr>
<tr>
<td></td>
<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>T₁</td>
<td>CLR-002</td>
<td>A</td>
<td>40</td>
<td>30</td>
<td>001</td>
<td>-</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></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>
<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>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>
**ABORT ALGORITHM**

First write an **ABORT** record to log for the txn. Then analyze the txn's updates in reverse order. For each update record:
- Write a **CLR** entry to the log.
- Restore old value.

Lastly, write a **TXN-END** record and release locks.

Notice: **CLRs** 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 bad for runtime performance but makes recovery easy.
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.
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.
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.
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.
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.
→ A
→ Dirty Page Table (DPT)
→ (ATT)
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.

Remove entry after the **TXN-END** record.

**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 that have not been flushed to disk.

One entry per dirty page in the buffer pool:
→ recLSN: The LSN of the log record that first caused the page to be dirty.
At the first checkpoint, assuming $P_{11}$ was flushed, $T_2$ is still running and there is only one dirty page ($P_{22}$),

At the second checkpoint, assuming $P_{22}$ was flushed, $T_2$ and $T_3$ are active and the dirty pages are ($P_{11}$, $P_{33}$).

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 writes the log records for checkpoint.

→ No attempt to force dirty pages to disk.

New log records to track checkpoint boundaries:

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

→ **CHECKPOINT-END**: Contains ATT + DPT.
FUZZY CHECKPOINT

Assume the DBMS flushes $P_{11}$ before the first checkpoint starts.

Any txn that begins after the checkpoint starts is excluded from the ATT in the CHECKPOINT-END record.

The LSN of the CHECKPOINT-BEGIN record is written to the MasterRecord when it completes.
FUZZY CHECKPOINT

Assume the DBMS flushes $P_{11}$ before the first checkpoint starts.

Any txn that begins after the checkpoint starts is excluded from the ATT in the CHECKPOINT-END record.

The LSN of the CHECKPOINT-BEGIN record is written to the MasterRecord when it completes.
**ARIES – RECOVERY PHASES**

**Phase #1 – Analysis**
→ Examine the WAL in forward direction starting at **MasterRecord** 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.
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.
ANALYSIS PHASE

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

All other records:
→ If txn not in **ATT**, add it with status **UNDO**.
→ On commit, change txn status to **COMMIT**.

For update log records:
→ If page **P** not in **DPT**, add **P** to **DPT**, set its **recLSN=LSN**.
ANALYSIS PHASE

At end of the Analysis Phase:

→ **ATT** identifies which txns were active at time of crash.
→ **DPT** identifies which dirty pages might not have made it to disk.
**ANALYSIS PHASE EXAMPLE**

**WAL**

010: `<CHECKPOINT-BEGIN>`

...  
020: `<T_{96}, A\rightarrow P_{33}, 10, 15>`  

...  
030: `<CHECKPOINT-END>`

ATT={T_{96}, T_{97}},  
DPT={P_{20}, P_{33}}  

...  
040: `<T_{96} COMMIT>`

...  
050: `<T_{96} TXN-END>`

CRASH!

---

<table>
<thead>
<tr>
<th>LSN</th>
<th>ATT</th>
<th>DPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>020</td>
<td></td>
<td></td>
</tr>
<tr>
<td>030</td>
<td></td>
<td></td>
</tr>
<tr>
<td>040</td>
<td></td>
<td></td>
</tr>
<tr>
<td>050</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ANALYSIS PHASE EXAMPLE

WAL

010: <CHECKPOINT-BEGIN>
    ...
020: <T_{96}, A\rightarrow P_{33}, 10, 15>
    ...
030: <CHECKPOINT-END
    ATT=\{T_{96}, T_{97}\},
    DPT=\{P_{20}, P_{33}\}>
    ...
040: <T_{96} COMMIT>
    ...
050: <T_{96} TXN-END>
    ...
CRASH!

LSN  ATT  DPT
010
020  (T_{96}, U)
030
040
050

(TxnId, Status)
ANALYSIS PHASE EXAMPLE

Modify A in page $P_{33}$

<table>
<thead>
<tr>
<th>ATT</th>
<th>DPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>010</td>
<td></td>
</tr>
<tr>
<td>020</td>
<td>$(T_{96}, U)$ $(P_{33}, 020)$</td>
</tr>
<tr>
<td>030</td>
<td></td>
</tr>
<tr>
<td>040</td>
<td></td>
</tr>
<tr>
<td>050</td>
<td></td>
</tr>
</tbody>
</table>

Modify A in page $P_{33}$

```
010: <CHECKPOINT>

020: <T_{96}, A→P_{33}, 10, 15>

030: <CHECKPOINT-END
ATT={T_{96}, T_{97}},
DPT={P_{20}, P_{33}}>

040: <T_{96} COMMIT>

050: <T_{96} TXN-END>

CRASH!
```
ANALYSIS PHASE EXAMPLE

WAL

010:<CHECKPOINT-BEGIN>

020:<T_{96}, A\rightarrow P_{33}, 10, 15>

030:<CHECKPOINT-END
ATT=\{T_{96}, T_{97}\},
DPT=\{P_{20}, P_{33}\}>

040:<T_{96} COMMIT>

050:<T_{96} TXN-END>

CRASH!

<table>
<thead>
<tr>
<th>LSN</th>
<th>ATT</th>
<th>DPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>020</td>
<td>(T_{96}, U)</td>
<td>(P_{33}, 020)</td>
</tr>
<tr>
<td>030</td>
<td>(T_{96}, U), (T_{97}, U)</td>
<td>(P_{33}, 020), (P_{20}, 008)</td>
</tr>
<tr>
<td>040</td>
<td></td>
<td></td>
</tr>
<tr>
<td>050</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ANALYSIS PHASE EXAMPLE

WAL

\[010: <\text{CHECKPOINT-BEGIN}>\]
\[020: <T_{96}, A \rightarrow P_{33}, 10, 15>\]
\[030: <\text{CHECKPOINT-END}\]
\[\text{ATT} = \{T_{96}, T_{97}\}, \]
\[\text{DPT} = \{P_{20}, P_{33}\}\]
\[040: <T_{96} \text{ COMMIT}>\]
\[050: <T_{96} \text{ TXN-END}>\]
\[\text{CRASH!}\]

<table>
<thead>
<tr>
<th>LSN</th>
<th>ATT</th>
<th>DPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>020</td>
<td>(T_{96}, U)</td>
<td>(P_{33}, 020)</td>
</tr>
<tr>
<td>030</td>
<td>(T_{96}, U), (T_{97}, U)</td>
<td>(P_{33}, 020), (P_{20}, 008)</td>
</tr>
<tr>
<td>040</td>
<td>(T_{96}, C), (T_{97}, U)</td>
<td>(P_{33}, 020), (P_{20}, 008)</td>
</tr>
<tr>
<td>050</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ANALYSIS PHASE EXAMPLE

**WAL**

<table>
<thead>
<tr>
<th>LSN</th>
<th>ATT</th>
<th>DPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>020</td>
<td>(T₉₆, U)</td>
<td>(P₃₃, 020)</td>
</tr>
<tr>
<td>030</td>
<td>(T₉₆, U), (T₉₇, U)</td>
<td>(P₃₃, 020), (P₂₀, 008)</td>
</tr>
<tr>
<td>040</td>
<td>(T₉₆, C), (T₉₇, U)</td>
<td>(P₃₃, 020), (P₂₀, 008)</td>
</tr>
<tr>
<td>050</td>
<td>(T₉₇, U)</td>
<td>(P₃₃, 020), (P₂₀, 008)</td>
</tr>
</tbody>
</table>

010: <CHECKPOINT-BEGIN>

020: <T₉₆, A→P₃₃, 10, 15>

030: <CHECKPOINT-END
ATT={T₉₆, T₉₇},
DPT={P₂₀, P₃₃}>

040: <T₉₆ COMMIT>

050: <T₉₆ TXN-END>

CRASH!
REDO PHASE

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

There are 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.
**REDO PHASE**

To redo an action:
→ Reapply logged update.
→ Set $\text{pageLSN}$ to log record's $\text{LSN}$.
→ No additional logging, no forced flushes!

At the end of Redo Phase, write $\text{TXN-END}$ log records for all txns with status $\text{C}$ and remove them from the $\text{ATT}$. 
**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

<table>
<thead>
<tr>
<th>TIME</th>
<th>LSN</th>
<th>LOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td></td>
<td><strong>&lt;CHECKPOINT-BEGIN&gt;</strong></td>
</tr>
<tr>
<td>05</td>
<td></td>
<td><strong>&lt;CHECKPOINT-END&gt;</strong></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td><strong>&lt;T₁, A→P₅, 1, 2&gt;</strong></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td><strong>&lt;T₂, B→P₃, 2, 3&gt;</strong></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td><strong>&lt;T₁ ABORT&gt;</strong></td>
</tr>
<tr>
<td>40</td>
<td></td>
<td><strong>&lt;CLR: Undo T₁ LSN 10&gt;</strong></td>
</tr>
<tr>
<td>45</td>
<td></td>
<td><strong>&lt;T₁ TXN-END&gt;</strong></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

prevLSNs
# FULL EXAMPLE

<table>
<thead>
<tr>
<th>LSN</th>
<th>LOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>&lt;CHECKPOINT-BEGIN&gt;</td>
</tr>
<tr>
<td>05</td>
<td>&lt;CHECKPOINT-END&gt;</td>
</tr>
<tr>
<td>10</td>
<td>&lt;T₁, A→P₅, 1, 2&gt;</td>
</tr>
<tr>
<td>20</td>
<td>&lt;T₂, B→P₃, 2, 3&gt;</td>
</tr>
<tr>
<td>30</td>
<td>&lt;T₁ ABORT&gt;</td>
</tr>
<tr>
<td>40</td>
<td>&lt;CLR: Undo T₁ LSN 10&gt;</td>
</tr>
<tr>
<td>45</td>
<td>&lt;T₁ TXN-END&gt;</td>
</tr>
<tr>
<td>50</td>
<td>&lt;T₃, C→P₁, 4, 5&gt;</td>
</tr>
<tr>
<td>60</td>
<td>&lt;T₂, D→P₅, 6, 7&gt;</td>
</tr>
</tbody>
</table>

**CRASH!**

---

**TIME**

---

**LSN**

---

**LOG**
### FULL EXAMPLE

#### LSN

<table>
<thead>
<tr>
<th>LSN</th>
<th>LOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>00,05</td>
<td>&lt;CHECKPOINT-BEGIN&gt;, &lt;CHECKPOINT-END&gt;</td>
</tr>
<tr>
<td>10</td>
<td>&lt;T₁, A→P₅, 1, 2&gt;</td>
</tr>
<tr>
<td>20</td>
<td>&lt;T₂, B→P₃, 2, 3&gt;</td>
</tr>
<tr>
<td>30</td>
<td>&lt;T₁ ABORT&gt;</td>
</tr>
<tr>
<td>40,45</td>
<td>&lt;CLR: Undo T₁ LSN 10&gt;, &lt;T₁ TXN-END&gt;</td>
</tr>
<tr>
<td>50</td>
<td>&lt;T₃, C→P₁, 4, 5&gt;</td>
</tr>
<tr>
<td>60</td>
<td>&lt;T₂, D→P₅, 6, 7&gt;</td>
</tr>
</tbody>
</table>

#### CRASH! RESTART!

---

### ATT

<table>
<thead>
<tr>
<th>TxnId</th>
<th>Status</th>
<th>lastLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>T₂</td>
<td>U</td>
<td>60</td>
</tr>
<tr>
<td>T₃</td>
<td>U</td>
<td>50</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</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
**FULL EXAMPLE**

### LSN
- **00,05**: `<CHECKPOINT-BEGIN>, <CHECKPOINT-END>`
- **10**: `<T_1, A\rightarrow P_5, 1, 2>`
- **20**: `<T_2, B\rightarrow P_3, 2, 3>`
- **30**: `<T_1 ABORT>`
- **40,45**: `<CLR: Undo T_1 LSN 10>, <T_1 TXN-END>`
- **50**: `<T_3, C\rightarrow P_1, 4, 5>`
- **60**: `<T_2, D\rightarrow P_5, 6, 7>`

### LOG
- **70**: `<CLR: Undo T_2 LSN 60, UndoNext 20>`

### ATT
- **TxnId**: `T_2, T_3`
- **Status**: `U, U`
- **lastLSN**: `60, 50`

### DPT
- **PageId**: `P_1, P_3, P_5`
- **recLSN**: `50, 08, 10`
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>

Flush dirty pages + WAL to disk!

CRASH! RESTART!

70 | <CLR: Undo T2 LSN 60, UndoNext>
80,85 | <CLR: Undo T3 LSN 50>, <T3 TXN-END>
**FULL EXAMPLE**

<table>
<thead>
<tr>
<th>LSN</th>
<th>LOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>00,05</td>
<td>&lt;CHECKPOINT-BEGIN&gt;, &lt;CHECKPOINT-END&gt;</td>
</tr>
<tr>
<td>10</td>
<td>&lt;T₁, A→P₅, 1, 2&gt;</td>
</tr>
<tr>
<td>20</td>
<td>&lt;T₂, B→P₃, 2, 3&gt;</td>
</tr>
<tr>
<td>30</td>
<td>&lt;T₁ ABORT&gt;</td>
</tr>
<tr>
<td>40,45</td>
<td>&lt;CLR: Undo T₁ LSN 10&gt;, &lt;T₁ TXN-END&gt;</td>
</tr>
<tr>
<td>50</td>
<td>&lt;T₃, C→P₁, 4, 5&gt;</td>
</tr>
<tr>
<td>60</td>
<td>&lt;T₂, D→P₅, 6, 7&gt;</td>
</tr>
</tbody>
</table>

**CRASH! RESTART!**

Flush dirty pages + WAL to disk!

<table>
<thead>
<tr>
<th>LSN</th>
<th>LOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>70</td>
<td>&lt;CLR: Undo T₂ LSN 60, UndoNext</td>
</tr>
<tr>
<td>80,85</td>
<td>&lt;CLR: Undo T₃ LSN 50&gt;, &lt;T₃ TXN-END&gt;</td>
</tr>
</tbody>
</table>

**CRASH! RESTART!**
**FULL EXAMPLE**

**ATT**

<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>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></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>

**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 20>
80,85: <CLR: Undo T3 LSN 50>, <T3 TXN-END>

CRASH! RESTART!

90,95: <CLR: Undo T2 LSN 20>, <T2 TXN-END>
ADDITIONAL CRASH ISSUES (1)

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:
→ **LSNs** identify log records; linked into backwards chains per transaction via **prevLSN**.
→ **pageLSN** allows comparison of data page and log records.
You now know how to build a single-node DBMS.

So now we can talk about distributed databases!