Database Recovery
ADMINISTRIVIA

Homework #6: Monday November 27th @ 11:59pm

Project #4: Wednesday December 6th @ 11:59am

No class on Wednesday November 22nd
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.
Algorithms for Recovery and Isolation Exploiting Semantics

Developed at IBM Research in early 1990s.

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

Write-Ahead Logging:
→ Any change is recorded in log on stable storage before the database change is written to disk.
→ Has to be **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.
TODAY’S AGENDA

Log Sequence Numbers
Normal Commit & Abort Operations
Fuzzy Checkpointing
Recovery Algorithm
WAL RECORDS

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).

Various components in the system keep track of 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>RAM</td>
<td>Last $LSN$ in log on disk</td>
</tr>
<tr>
<td>pageLSN</td>
<td>@page$_i$</td>
<td>Newest update to page$_i$</td>
</tr>
<tr>
<td>recLSN</td>
<td>@page$_i$</td>
<td>Oldest update to page$_i$</td>
</tr>
<tr>
<td>lastLSN</td>
<td>$T_j$</td>
<td>Latest action of txn $T_j$</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 `i` can be written to disk, we must flush log at least to the point where:
→ `pageLSN_i ≤ 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>
<th>A=9</th>
<th>B=5</th>
<th>C=2</th>
</tr>
</thead>
<tbody>
<tr>
<td>flushedLSN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**Database**

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

**WAL**

001: `<T₁ BEGIN>`
002: `<T₁, A, 1, 2>`
003: `<T₁ COMMIT>`
004: `<T₂ BEGIN>`
005: `<T₂, A, 2, 3>`
006: `<T₂ BEGIN>`
007: `<CHECKPOINT>`
008: `<T₂ COMMIT>`
009: `<T₂, A, 3, 4>`
010: `<T₃ BEGIN>`
011: `<T₄, X, 5, 6>`
012: `<T₄, Y, 9, 7>`
013: `<T₅, B, 4, 2>`
014: `<T₅ COMMIT>`
015: `<T₅, B, 2, 3>`
016: `<T₅, C, 1, 2>`
WRITING LOG RECORDS

Log Sequence Numbers

Buffer Pool

Database

Log Sequence Numbers

MasterRecord

Page dimensions: 720.0x405.0
**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>
</tbody>
</table>
```

**WAL**

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

**Database**

MasterRecord

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

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

Database

pageLSN recLSN
A=9 B=5 C=2

MasterRecord

CMU 15-445/645 (Fall 2017)
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

Database

WAL

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

pageLSN  recLSN
A=9  B=5  C=2

MasterRecord
WAL (Tail)

017: \texttt{<T\_5 BEGIN>}
018: \texttt{<T\_5, A, 9, 8>}
019: \texttt{<T\_5, B, 5, 1>}
020: \texttt{<T\_5 COMMIT>}

Safe to unpin because pageLSN ≤ flushedLSN

Buffer Pool

\texttt{pageLSN recLSN}

\texttt{A=9 B=5 C=2}

MasterRecord

\texttt{pageLSN recLSN}

\texttt{A=9 B=5 C=2}

Database

\texttt{WAL}

001: \texttt{<T\_1 BEGIN>}
002: \texttt{<T\_1, A, 1, 2>}
003: \texttt{<T\_1 COMMIT>}
004: \texttt{<T\_2 BEGIN>}
005: \texttt{<T\_2, A, 2, 3>}
006: \texttt{<T\_2 BEGIN>}
007: \texttt{<CHECKPOINT>}
008: \texttt{<T\_2, COMMIT>}
009: \texttt{<T\_3, A, 3, 4>}
010: \texttt{<T\_3 BEGIN>}
011: \texttt{<T\_4, X, 5, 6>}
012: \texttt{<T\_4, Y, 9, 7>}
013: \texttt{<T\_4, B, 4, 2>}
014: \texttt{<T\_4, COMMIT>}
015: \texttt{<T\_4, B, 2, 3>}
016: \texttt{<T\_4, C, 1, 2>}

\textit{MasterRecord}

\textit{flushedLSN}

\textit{pageLSN recLSN}
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.

Record the $flushedLSN$ in memory.
NORMAL EXECUTION

Series of reads & writes, followed by commit or abort.

Assumptions:
→ Disk writes are atomic.
→ Strict 2PL.
→ STEAL + NO-FORCE buffer management, with Write-Ahead Logging.
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.
 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: <T2 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>

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

flushedLSN = 015

flashedLSN

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

pageLSN  recLSN
A=9  B=5  C=2

MasterRecord

flushedLSN = 015
We can trim the in-memory log up to flushedLSN.
We can trim the in-memory log up to flushedLSN.
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.
**TRANSACTION COMMIT**

```
012|nil <T4 BEGIN>
013|012 <T4, A, 9, 8>
014|013 <T4, B, 5, 1>
015|014 <T4 ABORT>
    ...
016|015:<T4 TXN-END>
```

**Buffer Pool**

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

**WAL (Tail)**

```
LSN | prevLSN
012|nil
013|012
014|013
015|014
```

**WAL**

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

**Database**

```
MasterRecord
```
TRANSACTION COMMIT

WAL (Tail)

012|nil:<T_4 \text{ BEGIN}> 
013|012:<T_4, A, 9, 8> 
014|013:<T_4, B, 5, 1> 
015|014:<T_4 \text{ ABORT}> 
016|015:<T_4 \text{ TXN-END}>

Buffer Pool

\begin{array}{c|c|c}
\hline
\text{Page LSN} & \text{Rec LSN} & \text{Value} \\
\hline
A=9 & B=5 & C=2 \\
\hline
\end{array}

\text{Flushed LSN}

WAL

\begin{array}{c|c|c|c}
\hline
\text{Page LSN} & \text{Rec LSN} & \text{Value} \\
\hline
A=9 & B=5 & C=2 \\
\hline
\end{array}

\text{Master Record}

Database
Important: We 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 (i.e., 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>
<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</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>$T_1$</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</td>
<td>A</td>
<td>40</td>
<td>30</td>
<td>001</td>
</tr>
</tbody>
</table>

**TIME**
## 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</td>
<td>A</td>
<td>40</td>
<td>30</td>
<td>001</td>
</tr>
</tbody>
</table>

**TIME**
**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</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</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.
Then play back updates in reverse order.
For each update:
→ Write a **CLR** entry.
→ Restore old value.
At end, write a **TXN-END** log record.

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 obviously bad...
Pause txns while the DBMS takes the checkpoint.

→ We don't have to wait until all txns finish before taking the checkpoint.
Pause txns while the DBMS takes the checkpoint.
→ We don't have to wait until all txns finish before taking the checkpoint.
Better Checkpoints

Pause txns while the DBMS takes the checkpoint.
→ We don't have to wait until all txns finish before taking the checkpoint.

We have to now record internal system 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** written by txn.

Entry removed when txn commits or aborts.

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:

→ recLSN: The LSN of the log record that first caused the page to be dirty.
At the first checkpoint, $T_2$ is still running and there are two dirty pages (i.e., $P_{10}$, $P_{12}$).
At the first checkpoint, $T_2$ is still running and there are two dirty pages (i.e., $P_{10}$, $P_{12}$).
At the first checkpoint, $T_2$ is still running and there are two dirty pages (i.e., $P_{10}$, $P_{12}$).

At the second checkpoint, $T_3$ is active and there are two dirty pages (i.e., $P_{10}$, $P_{33}$).
At the first checkpoint, T_2 is still running and there are two dirty pages (i.e., P_{10}, P_{12}).

At the second checkpoint, T_3 is active and there are two dirty pages (i.e., P_{10}, P_{33}).

This still isn't great because we have to stall all txns during checkpoint...
FUZZY CHECKPOINTS

A **fuzzy checkpoint** is where the DBMS allows other txns to continue the run.

New log records to track checkpoint boundaries:

→ **CHECKPOINT-BEGIN**: Indicates start of checkpoint
→ **CHECKPOINT-END**: Contains **ATT** + **DPT**.
FUZZY CHECKPOINTS

The LSN of the CHECKPOINT-BEGIN record is written to the MasterRecord entry.

Any txn that starts after the checkpoint is excluded from the txn table listing.
The **LSN** of the **CHECKPOINT-BEGIN** record is written to the **MasterRecord** entry.

Any txn that starts after the checkpoint is excluded from the txn table listing.
FUZZY CHECKPOINTS

The LSN of the CHECKPOINT-BEGIN record is written to the MasterRecord entry.

Any txn that starts after the checkpoint is excluded from the txn table listing.
ARIES – RECOVERY PHASES

**Analysis:** Read the WAL to identify dirty pages in the buffer pool and active txns at the time of the crash.

**Redo:** Repeat all actions starting from an appropriate point in the log.

**Undo:** Reverse the actions of txns that did not commit before the crash.
ARIES – OVERVIEW

Start from last checkpoint found via MasterRecord.

Three phases.
→ Analysis: Figure out which txns committed or failed since checkpoint.
→ Redo: Repeat all actions.
→ Undo: Reverse effects of failed txns.

Oldest log record of txn active at crash
Smallest recLSN in dirty page table after Analysis

Last checkpoint
CRASH!
ANALYSIS PHASE

Re-establish knowledge of state at checkpoint.
→ Examine ATT and DPT stored in the checkpoint.
ANALYSIS PHASE

Scan log forward from checkpoint.
If you find a TXN-END record, remove its 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.
ANALYSIS PHASE

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

WAL

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!

<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→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></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: <T₉₆, A→P₃₃, 10, 15>

030: <CHECKPOINT-END

ATT={T₉₆, T₉₇},

DPT={P₂₀, P₃₃}>

040: <T₉₆ COMMIT>

050: <T₉₆ TXN-END>

CRASH!

**LSN**

<table>
<thead>
<tr>
<th>LSN</th>
<th>ATT</th>
<th>(TxnId, Status)</th>
</tr>
</thead>
<tbody>
<tr>
<td>010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>020</td>
<td>(T₉₆, U)</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

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!

<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})</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>

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})</td>
</tr>
<tr>
<td>030</td>
<td>(T_{96}, U), (T_{97}, U)</td>
<td>(P_{33}), (P_{20})</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!

<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})</td>
</tr>
<tr>
<td>030</td>
<td>(T_{96}, U),  (T_{97}, U)</td>
<td>(P_{33}), (P_{20})</td>
</tr>
<tr>
<td>040</td>
<td>(T_{96}, C),  (T_{97}, U)</td>
<td>(P_{33}), (P_{20})</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_{97} 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})</td>
</tr>
<tr>
<td>030</td>
<td>(T_{96}, U), (T_{97}, U)</td>
<td>(P_{33}), (P_{20})</td>
</tr>
<tr>
<td>040</td>
<td>(T_{96}, C), (T_{97}, U)</td>
<td>(P_{33}), (P_{20})</td>
</tr>
<tr>
<td>050</td>
<td>(T_{97}, U)</td>
<td>(P_{33}), (P_{20})</td>
</tr>
</tbody>
</table>
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.

We can 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 the DPT, or
→ Affected page is in DPT but that record's LSN is greater than smallest recLSN, or
→ Affected pageLSN (on disk) ≥ LSN
**REDO PHASE**

To redo an action:
→ Reapply logged action.
→ Set `pageLSN` to log record's `LSN`.
→ No additional logging, no forcing!

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 ("loser" txns).

→ These are all 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

ATT

<table>
<thead>
<tr>
<th>TxnId</th>
<th>Status</th>
<th>lastLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_2</td>
<td>U</td>
<td>60</td>
</tr>
<tr>
<td>T_3</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>
<th>flushedLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_1</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>P_3</td>
<td>08</td>
<td></td>
</tr>
<tr>
<td>P_5</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

LSN

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

CRASH! RESTART!
**FULL EXAMPLE**

<table>
<thead>
<tr>
<th>ATT</th>
<th>TxnId</th>
<th>Status</th>
<th>lastLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_2</td>
<td>U</td>
<td></td>
<td>60</td>
</tr>
<tr>
<td>T_3</td>
<td>U</td>
<td></td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DPT</th>
<th>PageId</th>
<th>recLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_1</td>
<td></td>
<td>50</td>
</tr>
<tr>
<td>P_3</td>
<td></td>
<td>08</td>
</tr>
<tr>
<td>P_5</td>
<td></td>
<td>10</td>
</tr>
</tbody>
</table>

<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;T_1, A→P_5, 1, 2&gt;</td>
</tr>
<tr>
<td>20</td>
<td>&lt;T_2, B→P_3, 2, 3&gt;</td>
</tr>
<tr>
<td>30</td>
<td>&lt;T_1 ABORT&gt;</td>
</tr>
<tr>
<td>40,45</td>
<td>&lt;CLR: Undo T_1 LSN 10&gt;, &lt;T_1 TXN-END&gt;</td>
</tr>
<tr>
<td>50</td>
<td>&lt;T_3, C→P_1, 4, 5&gt;</td>
</tr>
<tr>
<td>60</td>
<td>&lt;T_2, D→P_5, 6, 7&gt;</td>
</tr>
<tr>
<td>70</td>
<td>&lt;CLR: Undo T_2 LSN 60, UndoNext 20&gt;</td>
</tr>
</tbody>
</table>

**CRASH! RESTART!**
### Full Example

#### LSN | LOG
---|---
00,05 | `<CHECKPOINT-BEGIN>, <CHECKPOINT-END>`
10 | `<T_2, A→P_5, 1, 2>`
20 | `<T_2, B→P_3, 2, 3>`
30 | `<T_1 ABORT>`
40,45 | `<CLR: Undo T_1 LSN 10>, <T_1 TXN-END>`
50 | `<T_3, C→P_1, 4, 5>`
60 | `<T_2, D→P_5, 6, 7>`
70 | `<CLR: Undo T_2 LSN 60, UndoNext 20>`
80,85 | `<CLR: Undo T_3 LSN 50>, <T_3 TXN-END>`

### ATT

<table>
<thead>
<tr>
<th>TxnId</th>
<th>Status</th>
<th>lastLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_2</td>
<td>U</td>
<td>60</td>
</tr>
<tr>
<td>T_3</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>P_1</td>
<td>50</td>
</tr>
<tr>
<td>P_3</td>
<td>08</td>
</tr>
<tr>
<td>P_5</td>
<td>10</td>
</tr>
</tbody>
</table>

**Flush WAL to disk!**
**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>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>

**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>`

**Flush WAL to disk!**

CRASH! RESTART!
<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>
<tr>
<td>70</td>
<td>&lt;CLR: Undo T₂ LSN 60, UndoNext 20&gt;</td>
</tr>
<tr>
<td>80,85</td>
<td>&lt;CLR: Undo T₃ LSN 50&gt;, &lt;T₃ TXN-END&gt;</td>
</tr>
</tbody>
</table>

Flush WAL to disk!
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>
</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>

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>
70 | CRASH! RESTART!
80,85 | <CLR: Undo T₂ LSN 60, UndoNext 20>

flushedLSN

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>

**CLR**: Undo T1 LSN 10, <T1 TXN-END>

**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>

**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>

**CRASH! RESTART!**

70  <CLR: Undo T2 LSN 60, UndoNext 20>

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

**CRASH! RESTART!**
FULL EXAMPLE

ATT

<table>
<thead>
<tr>
<th>TxnId</th>
<th>Status</th>
<th>lastLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_2</td>
<td>U</td>
<td>70</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>P_1</td>
<td>50</td>
</tr>
<tr>
<td>P_3</td>
<td>08</td>
</tr>
<tr>
<td>P_5</td>
<td>10</td>
</tr>
</tbody>
</table>

LSN | LOG
---|----------------------------------
00,05 | <CHECKPOINT-BEGIN>, <CHECKPOINT-END>
10   | <T_1, A→P_5, 1, 2>
20   | <T_2, B→P_3, 2, 3>
30   | <T_1, ABORT>
40,45| <CLR: Undo T_1 LSN 10>, <T_1 TXN-END>
50   | <T_3, C→P_1, 4, 5>
60   | <T_2, D→P_5, 6, 7>
70   | CRASH! RESTART!
80,85| <CLR: Undo T_2 LSN 60, UndoNext 20>
90,95| <CLR: Undo T_2 LSN 20>, <T_2 TXN-END>

CRASH! RESTART!
ADDITIONAL CRASH ISSUES

What happens if system crashes during the Analysis Phase? During the Redo Phase?

How do you limit the amount of work in the Redo Phase?
→ Flush asynchronously in the background.

How do you limit the amount of work in the Undo Phase?
→ Avoid long-running txns.
CONCLUSION

Mains ideas of ARIES:
→ WAL (write ahead log), **STEAL/NO-FORCE**
→ Fuzzy Checkpoints (snapshot of dirty page ids)
→ Redo everything since the earliest dirty page; undo "loser" transactions
→ 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.
NEXT CLASS

You now know how to build an entire single-node ACID DBMS.

So now we can talk about distributed databases!