Carnegie Mellon Univ.  
Dept. of Computer Science  
15-415/615 - DB Applications  

C. Faloutsos – A. Pavlo  
Lecture#23: Crash Recovery – Part 2  
(R&G ch. 18)  

Administrivia  

• HW8 is due Thurs April 24th  

Last Class  

• Shadow Paging  
• Write-Ahead Log  
• Checkpoints  
• Logging Schemes
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.

fsync(2)

- Kernel maintains a buffer cache between applications & disks.
  - If you just call `write()`, there is no guarantee that the data is durable on disk.
- Use `fsync()` to force the OS to flush all modified in-core data to disk.
  - This blocks the thread until it completes.
  - Data may still live in on-disk cache but we cannot control that.

Buffer Pool – Steal Policy

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

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

### Write-Ahead Logging

- **WAL (Tail)**:
  - `<T5 begin>`
  - `<T5, A, 99, B>`
  - `<T5, B, 5, 10>`
  - `<T5 commit>`

- **WAL**

<table>
<thead>
<tr>
<th>Buffer Pool</th>
<th>WAL (Tail)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A=99</td>
<td>A=99</td>
</tr>
<tr>
<td>B=5</td>
<td>B=5</td>
</tr>
</tbody>
</table>

- **Volatile Storage**
- **Non-Volatile Storage**

### Writing Log Records

- We don’t want to write one record at a time
- How should we buffer them?
  - Batch log updates (group commit).
- Page $i$ can be written out only after the corresponding log record has been flushed.
Memory Pinning

- The DBMS needs to be able restrict when pages are flushed to disk.
- “Pinning” a page means that the buffer pool manager is not allowed to flush that page.
  - Think of it like a lock.
- **NOTE:** Block == Page
  - I use these terms interchangeably.
  - They mean the same thing.

Memory Pinning

- The DBMS un-pins a data page ONLY if all the corresponding log records that modified that page have been flushed to the log.

Checkpoints

- Any txn that committed before the checkpoint is ignored (T1).
- T2 + T3 did not commit before the last checkpoint.
  - Need to redo T2 because it committed after checkpoint.
  - Need to undo T3 because it did not commit before the crash.
Summary

• Write-Ahead Log to handle loss of volatile storage.
• Use incremental updates (i.e., STEAL, NO-FORCE) with checkpoints.

• On recovery, make sure that:
  – Committed txns are atomic + durable.
  – Uncommitted txns are removed.

Today's Class – ARIES

• Algorithms for Recovery and Isolation
  Exploiting Semantics
  – Write-ahead Logging
  – Repeating History during Redo
  – Logging Changes during Undo

ARIES

• Developed at IBM during the early 1990s.
• Considered the “gold standard” in database crash recovery.
  – Implemented in DB2.
  – Everybody else more or less implements a variant of it.

C. Mohan
IBM Fellow
ARIES – Main Ideas

• **Write-Ahead Logging:**
  – Any change is recorded in log on stable storage before the database change is written to disk.

• **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.

ARIES – Main Ideas

• **Write Ahead Logging**
  – Fast, during normal operation
  – Least interference with OS (i.e., STEAL, NO FORCE)

• Fast (fuzzy) checkpoints

• On Recovery:
  – Redo everything.
  – Undo uncommitted txns.

Today’s Class

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

- We’re going to extend our log record format from last class to include additional info.
- Every log record has a globally unique log sequence number (LSN).
- Q: Why do we need it?

Log Sequence Number

<table>
<thead>
<tr>
<th>Name</th>
<th>Where</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSN</td>
<td>–</td>
<td>Log sequence number</td>
</tr>
<tr>
<td>flushedLSN</td>
<td>RAM</td>
<td>Last LSN on log (disk).</td>
</tr>
<tr>
<td>pageLSN</td>
<td>@page</td>
<td>Latest update to page</td>
</tr>
<tr>
<td>recLSN</td>
<td>@page</td>
<td>Earliest update to page</td>
</tr>
<tr>
<td>lastLSN</td>
<td>Tj</td>
<td>Latest action of Tj</td>
</tr>
<tr>
<td>Master Record</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.
- For a page i to be written, must flush log at least to the point where:
  - pageLSN ≤ flushedLSN
Writing Log Records

- **LSNs**: Written for each log record.
- **pageLSN**: Stored in each page in database.
- **flushedLSN**: In-Memory only.

```
Faloutsos/Pavlo  CMU SCS 15-415/615
```
Today’s Class

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

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.

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 an TXN-END record to log.
Transaction Commit

• **Q:** Why not flush the dirty pages too?
• **A:** Speed! This is why we use NO-FORCE
  – Example: One txn changes 100 tuples…

Transaction Abort

• Aborting a txn is actually a special case of the ARIES **undo** operation applied to only one transaction.
• 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.
### 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>TIME</th>
<th>LSN</th>
<th>prevLSN</th>
<th>TxnId</th>
<th>Type</th>
<th>Object</th>
<th>Before</th>
<th>After</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>
</tbody>
</table>

**Important:** We need to record what steps we took to undo the txn.
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>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>nil</td>
<td>T1</td>
<td>BEGIN</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>
</tr>
<tr>
<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>
</tr>
<tr>
<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>
</tr>
</tbody>
</table>

The *LSN* of the next log record to be undone.

Abort Algorithm

- First, write an **ABORT** record on log
- Play back updates, in reverse order: for each update
  - Write a **CLR** log record
  - Restore old value
- At end, write an **END** log record
- Notice: CLRs never need to be undone
Today’s Class

- Log Sequence Numbers
- Normal Execution & Abort Operations
- Fuzzy Checkpointing
- Recovery Algorithm

(Non-Fuzzy) Checkpoints

- The DBMS halts everything when it takes a checkpoint to ensure a consistent snapshot:
  - Stop all transactions.
  - Flushes dirty pages on disk.
- This is bad…

Better Checkpoints

- Allow txns to keep on running.
- Record internal system state as of the beginning of the checkpoint.
  - Active Transaction Table (ATT)
  - Dirty Page Table (DPT)
Active Transaction Table (ATT)

- 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 (DPT)

- One entry per dirty page currently in buffer pool.
  - recLSN: The LSN of the log record that first caused the page to be dirty.

Better Checkpoints

- At the first checkpoint, T2 is still running and there are two dirty pages (i.e., P10, P12).
- At the second checkpoint, T3 is active and there are two dirty pages (i.e., P10, P33).
Fuzzy Checkpoints

• Specifically, write to log:
  – BEGIN-CHECKPOINT: Indicates start of checkpoint
  – END-CHECKPOINT: Contains ATT + DPT.

• The “fuzzy” part is because:
  – Other txns continue to run; so these tables accurate only as of the time of the BEGIN-CHECKPOINT record.
  – No attempt to force dirty pages to disk;

• The LSN of the BEGIN-CHECKPOINT record is written to the Master Record entry.

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

Q: Why do we need store the LSN of most recent checkpoint record on disk in the Master Record?

A: So that we know where to start from on recovery.
Today’s Class

- Log Sequence Numbers
- Normal Execution & Abort Operations
- Fuzzy Checkpointing
- Recovery Algorithm

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 Master Record.
- Three phases.
  - Analysis - Figure out which txns committed or failed since checkpoint.
  - Redo all actions (repeat history)
  - Undo effects of failed txns.

Recovery – Analysis Phase

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

Recovery – Analysis Phase

- Scan log forward from checkpoint.
- **END** record: Remove txn from ATT.
- All other records:
  - Add txn to ATT with status ‘UNDO’
  - Set lastLSN=LSN
  - On commit, change txn status to ‘COMMIT’.
- For **UPDATE** records:
  - If page P not in DPT, add P to DPT, set its recLSN=LSN.
Recovery – 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

<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)</td>
</tr>
<tr>
<td>030</td>
<td>(T96, U), (T97, U)</td>
<td>(P33), (P20)</td>
</tr>
<tr>
<td>040</td>
<td>(T96, C), (T97, U)</td>
<td>(P33), (P20)</td>
</tr>
<tr>
<td>050</td>
<td>(T97, U)</td>
<td>(P33), (P20)</td>
</tr>
</tbody>
</table>

CRASH!

Recovery – 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 try to avoid unnecessary reads/writes.
Recovery – Redo Phase

Why start here?
All else has been flushed.

• 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 has recLSN>LSN, or
  – pageLSN (in DB) ≥ LSN

Recovery – Redo Phase

• To redo an action:
  – Reapply logged action.
  – Set pageLSN to LSN.
  – No additional logging, no forcing!

• At the end of Redo Phase, write END log records for all txns with status ‘C’ and remove them from the ATT.

Recovery – Undo Phase

• Goal: Undo all txns that were active at the time of crash (‘loser txns’)
• That is, all txns with ‘U’ status in the ATT after the Analysis phase
• Process them in reverse LSN order using the lastLSN’s to speed up traversal.
• Write a CLR for every modification.
Recovery – Undo Phase

• ToUndo = \{ lastLSNs of ‘loser’ txns \}
• Repeat until ToUndo is empty:
  – Pop largest LSN from ToUndo.
  – If this LSN is a CLR and undoNext == nil, then write an END record for this txn.
  – If this LSN is a CLR, and undoNext != nil, then add undoNext to ToUndo.
  – Else this LSN is an update. Undo the update, write a CLR, add prevLSN to ToUndo.

Undo Phase Example

Suppose that after end of analysis phase we have the following ATT:

<table>
<thead>
<tr>
<th>TxnId</th>
<th>Status</th>
<th>lastLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>T32</td>
<td>U</td>
<td></td>
</tr>
<tr>
<td>T41</td>
<td>U</td>
<td></td>
</tr>
</tbody>
</table>

undo in reverse LSN order
Full Example

<table>
<thead>
<tr>
<th>LSN</th>
<th>LOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>begin_checkpoint</td>
</tr>
<tr>
<td>05</td>
<td>end_checkpoint</td>
</tr>
<tr>
<td>10</td>
<td>update: T1 writes P5</td>
</tr>
<tr>
<td>20</td>
<td>update: T2 writes P3</td>
</tr>
<tr>
<td>30</td>
<td>T1 abort</td>
</tr>
<tr>
<td>40</td>
<td>CLR: Undo T1 LSN 10</td>
</tr>
<tr>
<td>45</td>
<td>T1 End</td>
</tr>
<tr>
<td>50</td>
<td>update: T3 writes P1</td>
</tr>
<tr>
<td>60</td>
<td>update: T2 writes P5</td>
</tr>
<tr>
<td></td>
<td>CRASH</td>
</tr>
</tbody>
</table>

Volatile Storage

<table>
<thead>
<tr>
<th>PageId</th>
<th>recLSN</th>
<th>flushedLSN</th>
<th>ToUndo</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Full Example

<table>
<thead>
<tr>
<th>LSN</th>
<th>LOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>00,05</td>
<td>begin_checkpoint</td>
</tr>
<tr>
<td>10</td>
<td>update: T1 writes P5</td>
</tr>
<tr>
<td>20</td>
<td>update: T2 writes P3</td>
</tr>
<tr>
<td>30</td>
<td>T1 abort</td>
</tr>
<tr>
<td>40,45</td>
<td>CLR: Undo T1 LSN 10, T1 End</td>
</tr>
<tr>
<td>50</td>
<td>update: T3 writes P1</td>
</tr>
<tr>
<td>60</td>
<td>update: T2 writes P5</td>
</tr>
<tr>
<td></td>
<td>CRASH, RESTART</td>
</tr>
</tbody>
</table>

Volatile Storage

<table>
<thead>
<tr>
<th>PageId</th>
<th>recLSN</th>
<th>flushedLSN</th>
<th>ToUndo</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Full Example

<table>
<thead>
<tr>
<th>LSN</th>
<th>LOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>00,05</td>
<td>begin_checkpoint</td>
</tr>
<tr>
<td>10</td>
<td>update: T1 writes P5</td>
</tr>
<tr>
<td>20</td>
<td>update: T2 writes P3</td>
</tr>
<tr>
<td>30</td>
<td>T1 abort</td>
</tr>
<tr>
<td>40,45</td>
<td>CLR: Undo T1 LSN 10, T1 End</td>
</tr>
<tr>
<td>50</td>
<td>update: T3 writes P1</td>
</tr>
<tr>
<td>60</td>
<td>update: T2 writes P5</td>
</tr>
<tr>
<td></td>
<td>CRASH, RESTART</td>
</tr>
</tbody>
</table>

Volatile Storage

<table>
<thead>
<tr>
<th>PageId</th>
<th>recLSN</th>
<th>flushedLSN</th>
<th>ToUndo</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Full Example**

<table>
<thead>
<tr>
<th>LSN</th>
<th>LOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>00,05</td>
<td>begin_checkpoint, end_checkpoint</td>
</tr>
<tr>
<td>10</td>
<td>update: T1 writes P5</td>
</tr>
<tr>
<td>20</td>
<td>update T2 writes P3</td>
</tr>
<tr>
<td>30</td>
<td>T1 abort</td>
</tr>
<tr>
<td>40,45</td>
<td>CLR: Undo T1 LSN 10, T1 End</td>
</tr>
<tr>
<td>50</td>
<td>update: T3 writes P1</td>
</tr>
<tr>
<td>60</td>
<td>update: T2 writes P5</td>
</tr>
<tr>
<td>70</td>
<td>CRASH, RESTART</td>
</tr>
<tr>
<td>80,85</td>
<td>CLR: Undo T2 LSN 60</td>
</tr>
</tbody>
</table>

Flush WAL to disk!

**Volatile Storage**

Hexahedra

Flush WAL to disk!

**Crash During Restart!**

<table>
<thead>
<tr>
<th>LSN</th>
<th>LOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>00,05</td>
<td>begin_checkpoint, end_checkpoint</td>
</tr>
<tr>
<td>10</td>
<td>update: T1 writes P5</td>
</tr>
<tr>
<td>20</td>
<td>update T2 writes P3</td>
</tr>
<tr>
<td>30</td>
<td>T1 abort</td>
</tr>
<tr>
<td>40,45</td>
<td>CLR: Undo T1 LSN 10, T1 End</td>
</tr>
<tr>
<td>50</td>
<td>update: T3 writes P1</td>
</tr>
<tr>
<td>60</td>
<td>update: T2 writes P5</td>
</tr>
<tr>
<td>70</td>
<td>CRASH, RESTART</td>
</tr>
<tr>
<td>80,85</td>
<td>CLR: Undo T2 LSN 60</td>
</tr>
</tbody>
</table>

**Crash During Restart!**

<table>
<thead>
<tr>
<th>LSN</th>
<th>LOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>00,05</td>
<td>begin_checkpoint, end_checkpoint</td>
</tr>
<tr>
<td>10</td>
<td>update: T1 writes P5</td>
</tr>
<tr>
<td>20</td>
<td>update T2 writes P3</td>
</tr>
<tr>
<td>30</td>
<td>T1 abort</td>
</tr>
<tr>
<td>40,45</td>
<td>CLR: Undo T1 LSN 10, T1 End</td>
</tr>
<tr>
<td>50</td>
<td>update: T3 writes P1</td>
</tr>
<tr>
<td>60</td>
<td>update: T2 writes P5</td>
</tr>
<tr>
<td>70</td>
<td>CRASH, RESTART</td>
</tr>
<tr>
<td>80,85</td>
<td>CLR: Undo T2 LSN 60</td>
</tr>
</tbody>
</table>

Flush WAL to disk!
Crash During Restart!

<table>
<thead>
<tr>
<th>LSN</th>
<th>LOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>00,05</td>
<td>begin_checkpoint, end_checkpoint</td>
</tr>
<tr>
<td>10</td>
<td>update: T1 writes P5</td>
</tr>
<tr>
<td>20</td>
<td>update T2 writes P3</td>
</tr>
<tr>
<td>30</td>
<td>T1 abort</td>
</tr>
<tr>
<td>40,45</td>
<td>CLR: Undo T1 LSN 10, T1 End</td>
</tr>
<tr>
<td>50</td>
<td>update: T3 writes P1</td>
</tr>
<tr>
<td>60</td>
<td>update: T2 writes P5</td>
</tr>
</tbody>
</table>

**CLR: Undo T1 LSN 10, T1 End**

<table>
<thead>
<tr>
<th>LSN</th>
<th>LOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>70</td>
<td>CRASH, RESTART</td>
</tr>
<tr>
<td>80,85</td>
<td>CLR: Undo T2 LSN 60</td>
</tr>
<tr>
<td>90, 95</td>
<td>CLR: Undo T3 LSN 50, T3 end</td>
</tr>
</tbody>
</table>

**CLR: Undo T2 LSN 60**

**CLR: Undo T3 LSN 50, T3 end**

**CRASH, RESTART**

**CLR: Undo T2 LSN 20, T2 end**

**CRASH, RESTART**

**CLR: Undo T2 LSN 20, T2 end**

Volatile Storage

- **ATT**
  - LSN: 00,05
  - LOG: begin_checkpoint, end_checkpoint
  - 10: update: T1 writes P5
  - 20: update T2 writes P3
  - 30: T1 abort
  - 40,45: CLR: Undo T1 LSN 10, T1 End
  - 50: update: T3 writes P1
  - 60: update: T2 writes P5

- **DPT**
  - LSN: 70: CRASH, RESTART
  - 80,85: CLR: Undo T2 LSN 60
  - 90, 95: CLR: Undo T3 LSN 50, T3 end

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.

Summary

- **ARIES** - main ideas:
  - 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 CLRs when undoing, to survive failures during restarts
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.

Summary

- Additional concepts:
  - LSNs: identify log records; linked into backwards chains per transaction (via `prevLSN`).
  - `pageLSN`: allows comparison of data page and log records.
  - And several other subtle concepts: `undoNext`, `recLSN`, etc.

Conclusion

- Recovery is really hard.
- Be thankful that you don’t have to write it yourself.