Database Design and Implementation
CS 645
Recovery
Review: the ACID properties

- **Atomicity**
  - All actions of a Xact happen, or none happen

- **Consistency**
  - If each Xact is consistent, and the DB starts consistent, it ends up consistent

- **Isolation**
  - Execution of one Xact is isolated from others

- **Durability**
  - If a Xact commits, its effects persist

Which ones does the Recovery Manager help with?

*(also consistency related rollbacks)*
Primitive Operations of Transactions

- **READ**(X,t)
  - copy element X to transaction local variable t

- **WRITE**(X,t)
  - copy transaction local variable t to element X

- **INPUT**(X)
  - read element X to memory buffer

- **OUTPUT**(X)
  - write element X to disk
Example

START TRANSACTION
READ(A,t);
t := t*2;
WRITE(A,t);
READ(B,t);
t := t*2;
WRITE(B,t);
COMMIT;

Atomicity:
BOTH A and B are multiplied by 2
START TRANSACTION
READ(A,t);
t := t*2;
WRITE(A,t);
READ(B,t);
t := t*2;
WRITE(B,t);
COMMIT;
START TRANSACTION
READ(A,t);
t := t*2;
WRITE(A,t);
READ(B,t);
t := t*2;
WRITE(B,t);
COMMIT;

A=8
B=8

buffer

A=8
B=8

t=8
START TRANSACTION
READ(A,t);
t := t*2;
WRITE(A,t);
READ(B,t);
t := t*2;
WRITE(B,t);
COMMIT;

A=8
B=8

buffer

A=16


t=16
A=16
B=8

buffer

A=8
B=8

START TRANSACTION
READ(A,t);
t := t*2;
WRITE(A,t);
READ(B,t);
t := t*2;
WRITE(B,t);
COMMIT;

t=16
START TRANSACTION
READ(A,t);
t := t*2;
WRITE(A,t);
READ(B,t);
t := t*2;
WRITE(B,t);
COMMIT;

buffer

A=16  B=8

A=8  B=8

t=8
START TRANSACTION
READ(A,t);
t := t*2;
WRITE(A,t);
READ(B,t);
t := t*2;
WRITE(B,t);
COMMIT;

A=16  B=16

A=8   B=8

buffer

t=16
START TRANSACTION
READ(A,t);
t := t*2;
WRITE(A,t);
READ(B,t);
t := t*2;
WRITE(B,t);
COMMIT;

A=8   B=8
A=16   B=16

buffer

t=16
Solution: Use a Log

- Log = append-only file containing log records
- Note: multiple transactions run concurrently, log records are interleaved
- After a system crash, use log to:
  - Redo some transactions that did commit
  - Undo other transactions that did not commit

- Three kinds of logs: undo, redo, undo/redo

- WAL: Write Ahead Logging
  - All modification are written to a log before they are applied
Buffer Manager

Page requests from higher-level code

READ/WRITE

Buffer pool

Disk page

Free frame

Disk = collection of blocks

1 page corresponds to 1 disk block

Files and access methods

Buffer pool manager

Main memory

choice of frame dictated by replacement policy

INPUT/OUTPUT

Disk space manager

• Data must be in RAM for DBMS to operate on it!
• Buffer pool = table of <frame#, pageid> pairs
Buffer Manager Policies

**STEAL or NO-STEAL**
- Can an update made by an uncommitted transaction overwrite the most recent committed value of a data item on disk?

**FORCE or NO-FORCE**
- Should all updates of a transaction be forced to disk before the transaction commits?
ARIES Recovery Algorithm Overview

Three phases:

1. **Analysis**
   - Figure out what was going on at time of crash
   - List of dirty pages and active transactions

2. **Redo**
   - Redo all operations, even for transactions that will not commit
   - Get back to state at the moment of the crash

3. **Undo**
   - Remove effects of all uncommitted transactions
   - Log changes during undo in case of another crash during undo

Algorithms for Recovery and Isolation Exploiting Semantics
ARIES Recovery Algorithm Overview

Three principles:

1. **Write-Ahead Logging (WAL)**
   - Any change to a DB object is first recorded to the log
   - A log record must be written to disk before the corresponding object

2. **Repeating history**
   - Reinstate the exact state of the system before the crash

3. **Logging changes during UNDO**
   - Log UNDOs so we don’t repeat in a subsequent crash
Write-Ahead Log

1. **Must force the log record** of an update before the corresponding data page gets to disk

    ![Diagram](Diagram1.png)

2. **Must force all log records for a Xact** before commit
   - Xact is considered committed when its commit log record makes it to stable storage.

    ![Diagram](Diagram2.png)

#1 (with **UNDO** info) helps guarantee atomicity
#2 (with **REDO** info) helps guarantee durability
The Log

- Each log record has a unique Log Sequence Number (LSN)
  - Always increasing
- Each data page contains a pageLSN
  - The LSN of the most recent log record that updated that page
- System keeps track of flushedLSN
  - Max LSN flushed to stable storage
Types of Log Records

- **Update**
  - Whenever a page is modified, and update record is appended to the log tail.

- **Commit**
  - When a Xact commits it force-writes a commit log record (i.e. flushes the log tail, up to and including this record). The Xact is considered committed the moment this record is on stable storage.

- **Abort**
  - When a transaction is aborted (initiates rollback).

- **End**
  - When a Xact aborts or commits additional actions are initiated (e.g. rollback). Once those finish, an end record is appended.

- **CLR**
  - Compensation Log Record: Logs the UNDOs.

- **Checkpoint**
Log Records

<table>
<thead>
<tr>
<th>LSN</th>
<th>prevLSN</th>
<th>transID</th>
<th>type</th>
<th>pageID</th>
<th>length</th>
<th>offset</th>
<th>before-image</th>
<th>after-image</th>
</tr>
</thead>
</table>

- **Fields common to all log records**
- **Additional fields for update log records**

**CLR records**
- **REDO only**: they do not get undone
  - Only contain after-image
  - Additional **undoNextLSN** field
  - Points to the next log record of the Xact that should be undone
### Other Recovery-Related Structures

#### Transaction Table

<table>
<thead>
<tr>
<th>transID</th>
<th>status</th>
<th>lastLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Dirty Page Table

<table>
<thead>
<tr>
<th>pageID</th>
<th>recLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

First log entry that dirtied the page

The most recent log record for the Xact

running/committing/aborting
### Example of Recovery Structures

#### Transaction Table

<table>
<thead>
<tr>
<th>transID</th>
<th>status</th>
<th>lastLSN</th>
</tr>
</thead>
</table>

#### Dirty Page Table

<table>
<thead>
<tr>
<th>pageID</th>
<th>recLSN</th>
</tr>
</thead>
</table>

#### Buffer Pool

### Table Data

<table>
<thead>
<tr>
<th>LSN</th>
<th>prevLSN</th>
<th>transID</th>
<th>type</th>
<th>pageID</th>
<th>length</th>
<th>offset</th>
<th>before-image</th>
<th>after-image</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>null</td>
<td>T1</td>
<td>update</td>
<td>P5</td>
<td>3</td>
<td>21</td>
<td>ABC</td>
<td>DEF</td>
</tr>
</tbody>
</table>
Example of Recovery Structures

### Transaction Table

<table>
<thead>
<tr>
<th>transID</th>
<th>status</th>
<th>lastLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>running</td>
<td>10</td>
</tr>
</tbody>
</table>

### Dirty Page Table

<table>
<thead>
<tr>
<th>pageID</th>
<th>recLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>P5</td>
<td>10</td>
</tr>
</tbody>
</table>

### Buffer Pool

- P5
  - pageLSN=10

### Log Buffer Table

<table>
<thead>
<tr>
<th>LSN</th>
<th>prevLSN</th>
<th>transID</th>
<th>type</th>
<th>pageID</th>
<th>length</th>
<th>offset</th>
<th>before-image</th>
<th>after-image</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>null</td>
<td>T1</td>
<td>update</td>
<td>P5</td>
<td>3</td>
<td>21</td>
<td>ABC</td>
<td>DEF</td>
</tr>
</tbody>
</table>
Example of Recovery Structures

### Transaction Table

<table>
<thead>
<tr>
<th>transID</th>
<th>status</th>
<th>lastLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>running</td>
<td>10</td>
</tr>
</tbody>
</table>

### Dirty Page Table

<table>
<thead>
<tr>
<th>pageID</th>
<th>recLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>P5</td>
<td>10</td>
</tr>
</tbody>
</table>

### Buffer Pool

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P5</td>
<td>pageLSN=10</td>
</tr>
</tbody>
</table>

### Log Table

<table>
<thead>
<tr>
<th>LSN</th>
<th>prevLSN</th>
<th>transID</th>
<th>type</th>
<th>pageID</th>
<th>length</th>
<th>offset</th>
<th>before-image</th>
<th>after-image</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>null</td>
<td>T1</td>
<td>update</td>
<td>P5</td>
<td>3</td>
<td>21</td>
<td>ABC</td>
<td>DEF</td>
</tr>
<tr>
<td>20</td>
<td>null</td>
<td>T2</td>
<td>update</td>
<td>P6</td>
<td>3</td>
<td>41</td>
<td>HIJ</td>
<td>KLM</td>
</tr>
</tbody>
</table>
Example of Recovery Structures

<table>
<thead>
<tr>
<th>Transaction Table</th>
<th>Dirty Page Table</th>
<th>Buffer Pool</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>transID</strong></td>
<td><strong>pageID</strong></td>
<td></td>
</tr>
<tr>
<td>T1 running</td>
<td>P5 10</td>
<td></td>
</tr>
<tr>
<td>T2 running</td>
<td>P6 20</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>LSN</strong></th>
<th><strong>prevLSN</strong></th>
<th><strong>transID</strong></th>
<th><strong>type</strong></th>
<th><strong>pageID</strong></th>
<th><strong>length</strong></th>
<th><strong>offset</strong></th>
<th><strong>before-image</strong></th>
<th><strong>after-image</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>null</td>
<td>T1</td>
<td>update</td>
<td>P5</td>
<td>3</td>
<td>21</td>
<td>ABC</td>
<td>DEF</td>
</tr>
<tr>
<td>20</td>
<td>null</td>
<td>T2</td>
<td>update</td>
<td>P6</td>
<td>3</td>
<td>41</td>
<td>HIJ</td>
<td>KLM</td>
</tr>
</tbody>
</table>

- LSN and prevLSN represent the transaction log sequence numbers.
- The **transID** column lists the transaction IDs.
- The **type** column indicates whether the change is an update (T1) or another type.
- The **pageID** column identifies the affected page.
- The **length** column specifies the length of the update.
- The **offset** column indicates the byte offset for the update.
- The **before-image** and **after-image** columns show the data before and after the update.
### Example of Recovery Structures

<table>
<thead>
<tr>
<th>Transaction Table</th>
<th>Dirty Page Table</th>
<th>Buffer Pool</th>
</tr>
</thead>
<tbody>
<tr>
<td>transID</td>
<td>status</td>
<td>lastLSN</td>
</tr>
<tr>
<td>T1</td>
<td>running</td>
<td>10</td>
</tr>
<tr>
<td>T2</td>
<td>running</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LSN</th>
<th>prevLSN</th>
<th>transID</th>
<th>type</th>
<th>pageID</th>
<th>length</th>
<th>offset</th>
<th>before-image</th>
<th>after-image</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>null</td>
<td>T1</td>
<td>update</td>
<td>P5</td>
<td>3</td>
<td>21</td>
<td>ABC</td>
<td>DEF</td>
</tr>
<tr>
<td>20</td>
<td>null</td>
<td>T2</td>
<td>update</td>
<td>P6</td>
<td>3</td>
<td>41</td>
<td>HIJ</td>
<td>KLM</td>
</tr>
<tr>
<td>30</td>
<td>20</td>
<td>T2</td>
<td>update</td>
<td>P5</td>
<td>3</td>
<td>20</td>
<td>GDE</td>
<td>QRS</td>
</tr>
</tbody>
</table>

Buffer Pool:
- P5
  - pageLSN=10
- P6
  - pageLSN=20
### Example of Recovery Structures

#### Transaction Table

<table>
<thead>
<tr>
<th>transID</th>
<th>status</th>
<th>lastLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>running</td>
<td>10</td>
</tr>
<tr>
<td>T2</td>
<td>running</td>
<td>30</td>
</tr>
</tbody>
</table>

#### Dirty Page Table

<table>
<thead>
<tr>
<th>pageID</th>
<th>recLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>P5</td>
<td>10</td>
</tr>
<tr>
<td>P6</td>
<td>20</td>
</tr>
</tbody>
</table>

#### Buffer Pool

- P5: pageLSN=30
- P6: pageLSN=20

#### Log Table

<table>
<thead>
<tr>
<th>LSN</th>
<th>prevLSN</th>
<th>transID</th>
<th>type</th>
<th>pageID</th>
<th>length</th>
<th>offset</th>
<th>before-image</th>
<th>after-image</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>null</td>
<td>T1</td>
<td>update</td>
<td>P5</td>
<td>3</td>
<td>21</td>
<td>ABC</td>
<td>DEF</td>
</tr>
<tr>
<td>20</td>
<td>null</td>
<td>T2</td>
<td>update</td>
<td>P6</td>
<td>3</td>
<td>41</td>
<td>HIJ</td>
<td>KLM</td>
</tr>
<tr>
<td>30</td>
<td>20</td>
<td>T2</td>
<td>update</td>
<td>P5</td>
<td>3</td>
<td>20</td>
<td>GDE</td>
<td>QRS</td>
</tr>
</tbody>
</table>
Example of Recovery Structures

**Transaction Table**

<table>
<thead>
<tr>
<th>transID</th>
<th>status</th>
<th>lastLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>running</td>
<td>10</td>
</tr>
<tr>
<td>T2</td>
<td>running</td>
<td>30</td>
</tr>
</tbody>
</table>

**Dirty Page Table**

<table>
<thead>
<tr>
<th>pageID</th>
<th>recLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>P5</td>
<td>10</td>
</tr>
<tr>
<td>P6</td>
<td>20</td>
</tr>
</tbody>
</table>

**Buffer Pool**

- P5: pageLSN=30
- P6: pageLSN=20

**LSN Table**

<table>
<thead>
<tr>
<th>LSN</th>
<th>prevLSN</th>
<th>transID</th>
<th>type</th>
<th>pageID</th>
<th>length</th>
<th>offset</th>
<th>before-image</th>
<th>after-image</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>null</td>
<td>T1</td>
<td>update</td>
<td>P5</td>
<td>3</td>
<td>21</td>
<td>ABC</td>
<td>DEF</td>
</tr>
<tr>
<td>20</td>
<td>null</td>
<td>T2</td>
<td>update</td>
<td>P6</td>
<td>3</td>
<td>41</td>
<td>HIJ</td>
<td>KLM</td>
</tr>
<tr>
<td>30</td>
<td>20</td>
<td>T2</td>
<td>update</td>
<td>P5</td>
<td>3</td>
<td>20</td>
<td>GDE</td>
<td>QRS</td>
</tr>
<tr>
<td>40</td>
<td>10</td>
<td>T1</td>
<td>update</td>
<td>P7</td>
<td>3</td>
<td>21</td>
<td>TUV</td>
<td>WXY</td>
</tr>
</tbody>
</table>
Example of Recovery Structures

### Transaction Table

<table>
<thead>
<tr>
<th>transID</th>
<th>status</th>
<th>lastLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>running</td>
<td>40</td>
</tr>
<tr>
<td>T2</td>
<td>running</td>
<td>30</td>
</tr>
</tbody>
</table>

### Dirty Page Table

<table>
<thead>
<tr>
<th>pageID</th>
<th>recLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>P5</td>
<td>10</td>
</tr>
<tr>
<td>P6</td>
<td>20</td>
</tr>
<tr>
<td>P7</td>
<td>40</td>
</tr>
</tbody>
</table>

### Buffer Pool

- P5: pageLSN=30
- P6: pageLSN=20
- P7: pageLSN=40

---

<table>
<thead>
<tr>
<th>LSN</th>
<th>prevLSN</th>
<th>transID</th>
<th>type</th>
<th>pageID</th>
<th>length</th>
<th>offset</th>
<th>before-image</th>
<th>after-image</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>null</td>
<td>T1</td>
<td>update</td>
<td>P5</td>
<td>3</td>
<td>21</td>
<td>ABC</td>
<td>DEF</td>
</tr>
<tr>
<td>20</td>
<td>null</td>
<td>T2</td>
<td>update</td>
<td>P6</td>
<td>3</td>
<td>41</td>
<td>HIJ</td>
<td>KLM</td>
</tr>
<tr>
<td>30</td>
<td>20</td>
<td>T2</td>
<td>update</td>
<td>P5</td>
<td>3</td>
<td>20</td>
<td>GDE</td>
<td>QRS</td>
</tr>
<tr>
<td>40</td>
<td>10</td>
<td>T1</td>
<td>update</td>
<td>P7</td>
<td>3</td>
<td>21</td>
<td>TUV</td>
<td>WXY</td>
</tr>
</tbody>
</table>
Normal Execution

- Update transaction table on Xact start/end

- For each update:
  - Create log record with LSN $l = ++\text{MaxLSN}$ and prevLSN=TransTable(transID).lastLSN
  - Update TransTable(transID).lastLSN=$l$
  - If modified page not in dirty table, add it with recLSN=$l$

- If the buffer manager steals a dirty page, remove its entry from the DPT
Transaction Commit

- Write commit record to log

- Flush the log tail up to Xact’s commit to disk
  - WAL rule #2: flushedLSN ≥ lastLSN
  - Note that log flushes are sequential, synchronous writes, so cheaper than forcing updated data

- Remove entry from the TransTable

- Write end record to log
Transaction Abort (no crash)

- Write **abort** log record before starting rollback

- “Play back” undoing all updates
  - Get **lastLSN** of Xact from the TransTable
  - Follow chain of log records via **prevLSN**
  - For each update encountered
    - Write a **CLR** for each undone operation with **undoNextLSN = prevLSN** of record being undone
    - Undo the operation (using the before-image of the log record)

- Remove entry from the TransTable

- Write **end** record to log
Checkpoints

- **begin_checkpoint**
  - Indicates where checkpoint began

- **end_checkpoint**
  - Contains the Transaction Table and the Dirty Page Table as they were at begin_checkpoint

- Store the LSN of the most recent checkpoint at a master record on disk
The Big Picture: What’s Where

Log Records
- LSN
- prevLSN
- transID
- type
- ...

Data pages
- Each with a pageLSN

Master record
- LSN of most recent checkpoint

Transaction Table
- lastLSN
- status

Dirty Page Table
- recLSN

flushedLSN
Crash Recovery: Big Picture

- Start from a **checkpoint** (found from master record)

**Three phases:**

1. **Analysis** – update structures
   - TransTable: active Xacts at crash
   - DBT: pages that *might* be dirty at crash
2. **REDO everything** (repeat history)
   - Start at the smallest recLSN in DPT
3. **UNDO** failed Xacts
   - Stop at the oldest LSN of active Xact
Phase 1: Analysis

Goal
- Determine point in log where to start REDO
- Determine set of dirty pages when crashed
  - Conservative estimate
- Identify active transactions when crashed (loser transactions)

Approach
- Rebuild active transactions table and dirty pages table
- Compute: firstLSN = smallest of all recLSN in DPT
Phase 1: Analysis

Load the Transaction Table and Dirty Page Table stored at the checkpoint

Scan log forward from checkpoint
- end record: remove Xact from TransTable
- All other records:
  - add Xact to TransTable (if not there)
  - Set lastLSN=LSN
  - Change status accordingly
- update record: if P not in DPT, add it with recLSN=LSN
Phase 1: Analysis

Transaction Table

<table>
<thead>
<tr>
<th>transID</th>
<th>status</th>
<th>lastLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Dirty Page Table

<table>
<thead>
<tr>
<th>pageID</th>
<th>recLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Replay history

log

First LSN

Smallest recLSN

Last chkpt

CRASH
Phase 2: REDO

Principles:
- Scan the log forward from firstLSN
- Read all records sequentially, and reapply all updates
- Do not record REDO actions in the log
- Needs the DPT

Why start here?
Phase 2: REDO

Details:

For each updateable record (update or CLR) REDO the action, unless:

- Affected page not in DPT
- Affected page in DPT but recLSN > LSN
- pageLSN (in DB) ≥ LSN (requires I/O)

To REDO:

- Reapply logged action
- Set pageLSN to LSN
Phase 3: UNDO

**Principles:**
- Start from the end of the log, move backwards
- Read only affected log entries (loser Xacts)
- Undo actions logged as special entries: CLR (Compensation Log Records)
- CLRs are redone, but never undone
Phase 3: UNDO

Details:
- **Loser Xacts**: all Xacts in the Transaction Table
- **ToUndo**: \{lastLSN of all Loser Xacts\}

While ToUndo is not empty:
- Choose the most recent (largest) LSN in ToUndo
- If LSN is a **CLR** and undoNextLSN=null
  - Write end record for Xact
- If LSN is a **CLR** and undoNextLSN ≠ null
  - Add undoNextLSN to ToUndo
- If LSN is an update
  - Undo the action
  - Write a CLR
  - Add prevLSN to ToUndo
Example of Recovery – (up to crash)

Xact Table
- lastLSN
- status

Dirty Page Table
- recLSN
- flushedLSN

RAM

<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, UndoNxt=Null</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>-----</td>
</tr>
</tbody>
</table>

CRASH, RESTART
## Trans Table

<table>
<thead>
<tr>
<th>Trans</th>
<th>lastLSN</th>
<th>Stat</th>
</tr>
</thead>
<tbody>
<tr>
<td>T2</td>
<td>60</td>
<td>r</td>
</tr>
<tr>
<td>T3</td>
<td>50</td>
<td>r</td>
</tr>
</tbody>
</table>

## Dirty Page Table

<table>
<thead>
<tr>
<th>PageId</th>
<th>recLSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>P5</td>
<td>10</td>
</tr>
<tr>
<td>P3</td>
<td>20</td>
</tr>
<tr>
<td>P1</td>
<td>50</td>
</tr>
</tbody>
</table>

## LSN LOG

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

**CRASH, RESTART**

**Redo** starts at LSN 10; in this case, reads P5, P3, and P1 from disk, redoes ops if pageLSN < LSN

**ToUndo** set initializes to \{50, 60\}
After Analysis & REDO:
ToUndo:  \{50, 60\}

<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, UndoNxt=Null</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>70</td>
<td>CRASH, RESTART</td>
</tr>
<tr>
<td></td>
<td>CLR: Undo T2 LSN 60; UndoNxtLSN=20</td>
</tr>
</tbody>
</table>
After Analysis & REDO:

ToUndo: {50, 60}

ToUndo: {50, 20}

begin_checkpoint
end_checkpoint
update: T1 writes P5
update: T2 writes P3
T1 abort
CLR: Undo T1 LSN 10, UndoNxt=Null
T1 End
update: T3 writes P1
update: T2 writes P5
CRASH, RESTART
CLR: Undo T2 LSN 60; UndoNxtLSN=20
CLR: Undo T3 LSN 50; UndoNxtLSN=null
After Analysis & REDO:

**ToUndo:** {50, 60}
- update: T1 writes P5

**ToUndo:** {50, 20}
- update: T2 writes P3
- T1 abort
- CLR: Undo T1 LSN 10, UndoNxt=Null

**ToUndo:** {20}
- update: T3 writes P1
- update: T2 writes P5
- CRASH, RESTART
- CLR: Undo T2 LSN 60; UndoNxtLSN=20
- CLR: Undo T3 LSN 50; UndoNxtLSN=null
- T3 end
After Analysis & REDO:

**ToUndo:** {50, 60}
- **LSN 10:** update: T1 writes P5
- **LSN 20:** update T2 writes P3
- **LSN 30:** T1 abort
- **LSN 40:** CLR: Undo T1 LSN 10, UndoNxt=Null
- **LSN 45:** T1 End
- **LSN 50:** update: T3 writes P1
- **LSN 60:** update: T2 writes P5

**ToUndo:** {50, 20}

**ToUndo:** {20}
- **LSN 70:** CRASH, RESTART
- **LSN 75:** CLR: Undo T2 LSN 60; UndoNxtLSN=20
- **LSN 80:** CLR: Undo T3 LSN 50; UndoNxtLSN=null
- **LSN 85:** T3 end

**ToUndo:** {50, 60}
- **LSN 40:** CLR: Undo T1 LSN 10, UndoNxt=Null
- **LSN 45:** T1 End
- **LSN 50:** update: T3 writes P1
- **LSN 60:** update: T2 writes P5

**ToUndo:** {50, 20}
- **LSN 70:** CRASH, RESTART
- **LSN 75:** CLR: Undo T2 LSN 60; UndoNxtLSN=20
- **LSN 80:** CLR: Undo T3 LSN 50; UndoNxtLSN=null
- **LSN 85:** T3 end

**ToUndo:** {20}
### After Analysis & REDO:

<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, UndoNxt=Null</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>
</tbody>
</table>

**ToUndo:** {50, 60}

**ToUndo:** {50, 20}

**ToUndo:** {20}

---

**After Analysis & REDO:**

<table>
<thead>
<tr>
<th>LSN</th>
<th>LOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>70</td>
<td>CRASH, RESTART</td>
</tr>
<tr>
<td>70</td>
<td>CLR: Undo T2 LSN 60; UndoNxtLSN=20</td>
</tr>
<tr>
<td>80</td>
<td>CLR: Undo T3 LSN 50; UndoNxtLSN=null</td>
</tr>
<tr>
<td>85</td>
<td>T3 end</td>
</tr>
<tr>
<td>90</td>
<td>CRASH, RESTART</td>
</tr>
<tr>
<td>90</td>
<td>CLR: Undo T2 LSN 20; UndoNxtLSN=null</td>
</tr>
</tbody>
</table>

**ToUndo:** {70}

**ToUndo:** {20}
After Analysis & REDO:

ToUndo: \{50, 60\}

After Analysis & REDO:

ToUndo: \{50, 20\}

After Analysis & REDO:

ToUndo: \{20\}

After Analysis & REDO:

ToUndo: \{70\}

After Analysis & REDO:

ToUndo: \{20\}

CRASH, RESTART
CLR: Undo T2 LSN 60; UndoNxtLSN=20
CLR: Undo T3 LSN 50; UndoNxtLSN=null
T3 end

CRASH, RESTART
CLR: Undo T2 LSN 20; UndoNxtLSN=null
T2 end
Discussion

- What if we crash during Analysis? During REDO?

- How can we reduce the amount of work in Analysis?

- How do we reduce the amount of work in REDO?

- What affects the amount of work in UNDO?