# Introduction to Database Systems CSE 444

Lectures 11-12
Transactions: Recovery (ARIES)

# Readings

- Material in today's lecture NOT in the book

- Instead, read Sections 1, 2.2, and 3.2 of: Michael J. Franklin. Concurrency Control and Recovery. The Handbook of Computer Science and Engineering, A. Tucker, ed., CRC Press, Boca Raton, 1997.

- (Also, chapter 18 of the "cow book")

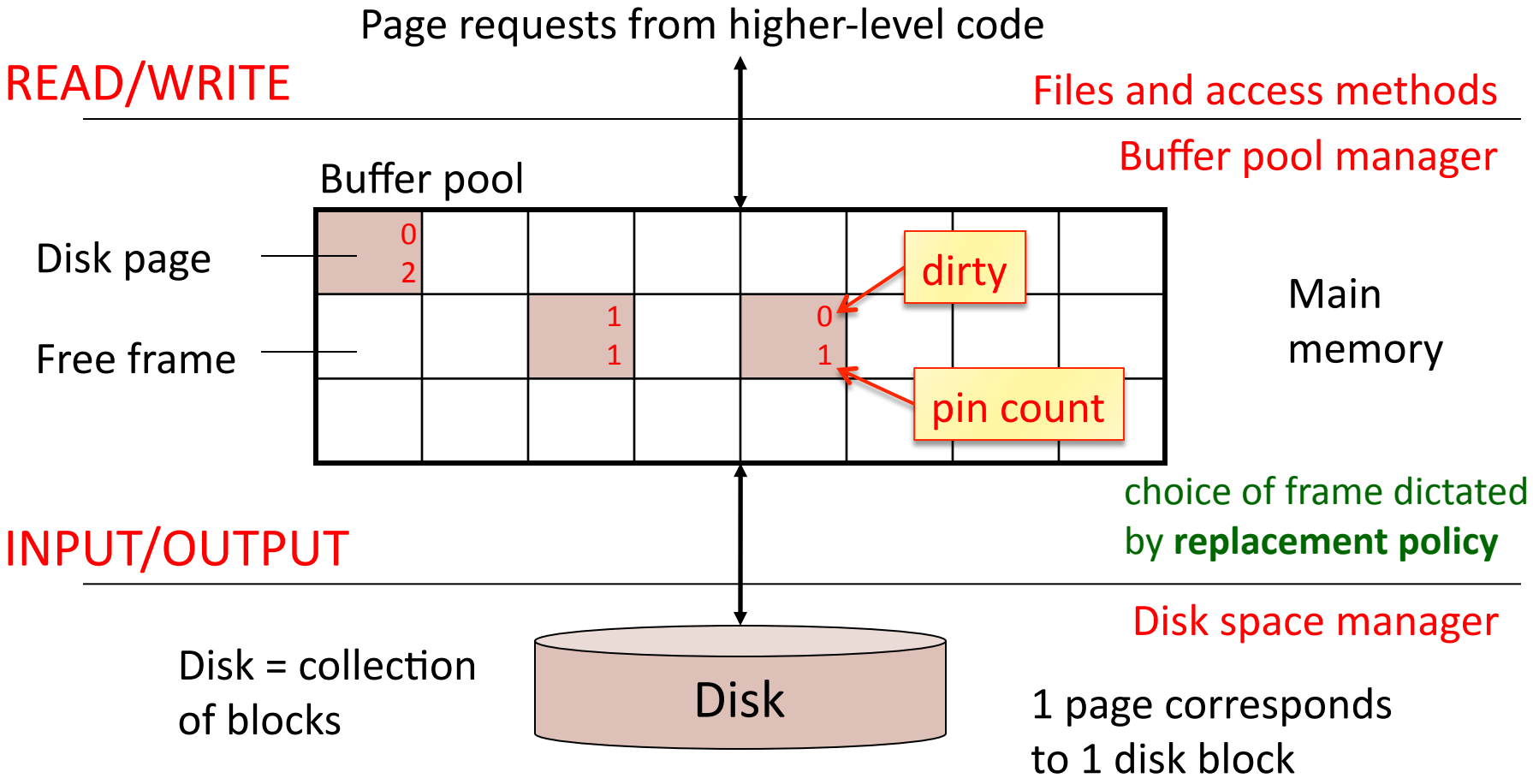
# Review: the ACID properties

- 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)

# Buffer 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?

| | No Steal | Steal |
|---|---|---|
| No Force | | Fastest |
| Force | Slowest | |

| | No Steal | Steal |
|---|---|---|
| No Force | No UNDO<br>REDO | UNDO<br>REDO |
| Force | No UNDO<br>No REDO | UNDO<br>No REDO |

# 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

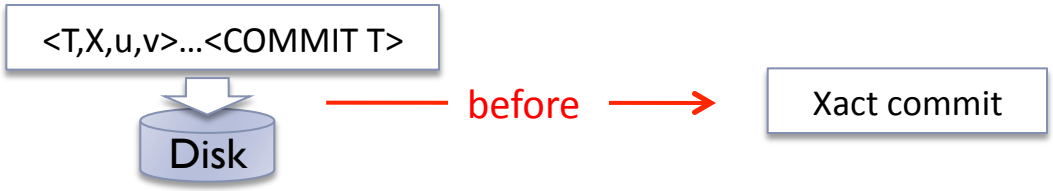
**Three principles:**

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

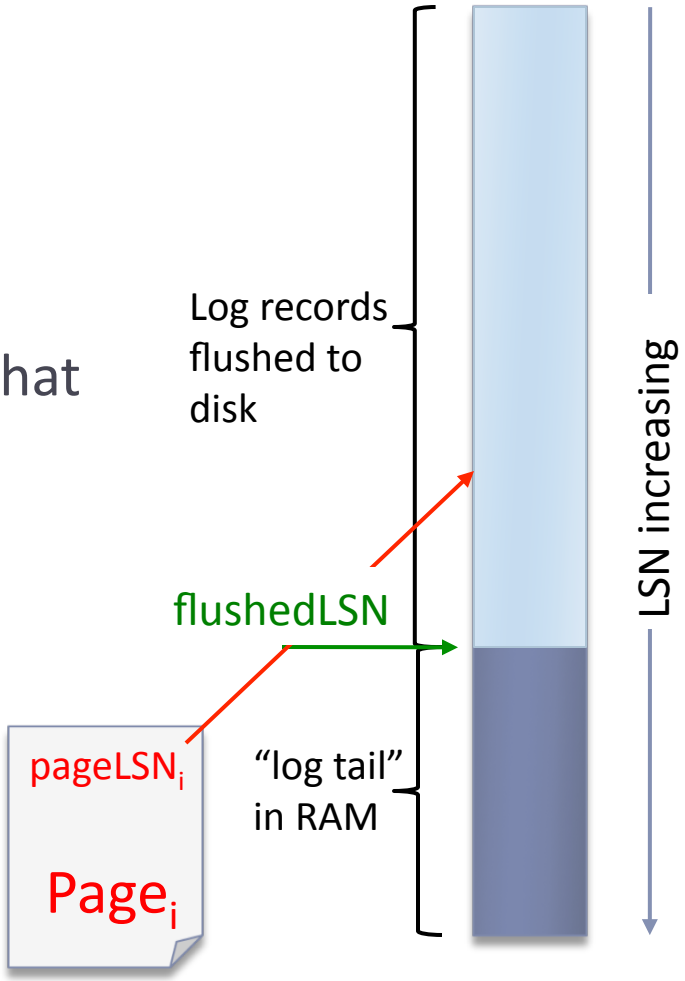

2. Must force all log records for a Xact before commit
   - Xact is considered committed when its commit log record makes it to stable storage.




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

# The Log

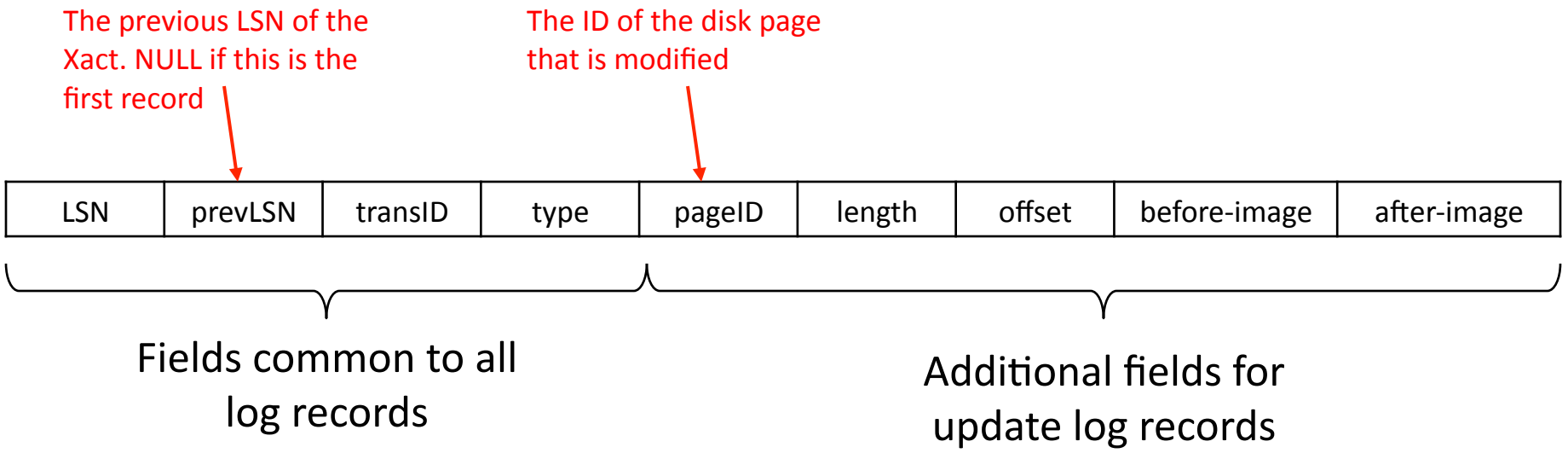
- Each log record has a unique
  Log Sequence Number (LSN)
  - Always increasing
- Each data page contains a pageLSN
  - The LSN of the most recent log record that updated that page
- System keeps track of flushedLSN
  - Max LSN flushed to stable storage



# Types of Log Records

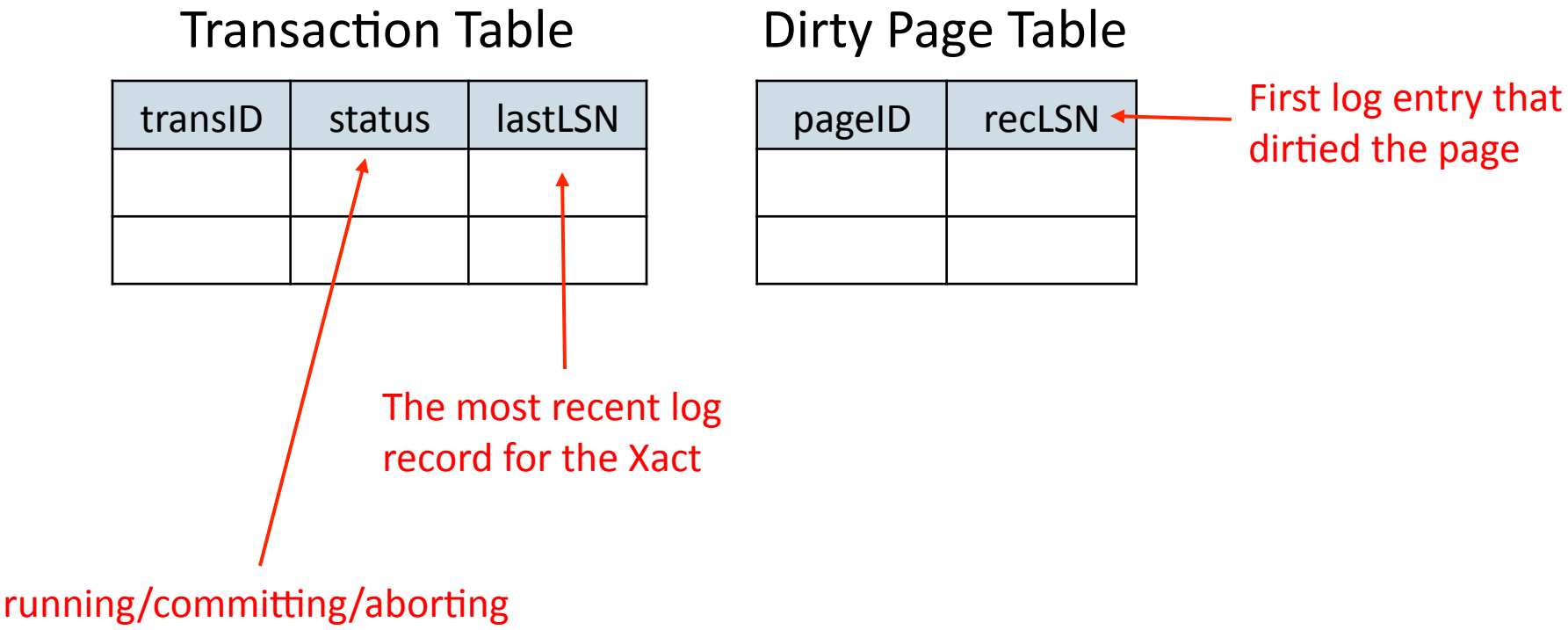
- Update
  - Whenever a page is modified, and update record is appended to the log tail
- Commit
  - When a Xact commits it force-writes a commit log record (i.e. flushes the log tail, up to and including this record). The Xact is considered committed the moment this record is on stable storage
- Abort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




- 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



# Example of Recovery Structures

Transaction Table

| transID | status | lastLSN |
|---|---|---|

Dirty Page Table

| pageID | recLSN |
|---|---|

Buffer Pool

| | |
|---|---|
| | |
| | |

| LSN | prevLSN | transID | type | pageID | length | offset | before-image | after-image |
|---|---|---|---|---|---|---|---|---|
| 10 | null | T1 | update | P5 | 3 | 21 | ABC | DEF |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |

# Example of Recovery Structures

Transaction Table

| transID | status | lastLSN |
|---|---|---|
| T1 | running | 10 |

Dirty Page Table

| pageID | recLSN |
|---|---|
| P5 | 10 |

Buffer Pool

| | |
|---|---|
| P5<br>pageLSN=10 | |
| | |

| LSN | prevLSN | transID | type | pageID | length | offset | before-image | after-image |
|---|---|---|---|---|---|---|---|---|
| 10 | null | T1 | update | P5 | 3 | 21 | ABC | DEF |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |

# Example of Recovery Structures

Transaction Table

| transID | status | lastLSN |
|---|---|---|
| T1 | running | 10 |

Dirty Page Table

| pageID | recLSN |
|---|---|
| P5 | 10 |

Buffer Pool

| | |
|---|---|
| P5<br>pageLSN=10 | |
| | |

| LSN | prevLSN | transID | type | pageID | length | offset | before-image | after-image |
|---|---|---|---|---|---|---|---|---|
| 10 | null | T1 | update | P5 | 3 | 21 | ABC | DEF |
| 20 | null | T2 | update | P6 | 3 | 41 | HIJ | KLM |
| | | | | | | | | |
| | | | | | | | | |

# Example of Recovery Structures

Transaction Table

| transID | status | lastLSN |
|---|---|---|
| T1 | running | 10 |
| T2 | running | 20 |

Dirty Page Table

| pageID | recLSN |
|---|---|
| P5 | 10 |
| P6 | 20 |

Buffer Pool

| | |
|---|---|
| P5 pageLSN=10 | P6 pageLSN=20 |
| | |

| LSN | prevLSN | transID | type | pageID | length | offset | before-image | after-image |
|---|---|---|---|---|---|---|---|---|
| 10 | null | T1 | update | P5 | 3 | 21 | ABC | DEF |
| 20 | null | T2 | update | P6 | 3 | 41 | HIJ | KLM |
| | | | | | | | | |
| | | | | | | | | |

# Example of Recovery Structures

Transaction Table

| transID | status | lastLSN |
|---|---|---|
| T1 | running | 10 |
| T2 | running | 20 |

Dirty Page Table

| pageID | recLSN |
|---|---|
| P5 | 10 |
| P6 | 20 |

Buffer Pool

| | |
|---|---|
| P5 pageLSN=10 | P6 pageLSN=20 |
| | |

| LSN | prevLSN | transID | type | pageID | length | offset | before-image | after-image |
|---|---|---|---|---|---|---|---|---|
| 10 | null | T1 | update | P5 | 3 | 21 | ABC | DEF |
| 20 | null | T2 | update | P6 | 3 | 41 | HIJ | KLM |
| 30 | 20 | T2 | update | P5 | 3 | 20 | GDE | QRS |
| | | | | | | | | |

# Example of Recovery Structures

### Transaction Table

| transID | status | lastLSN |
|---|---|---|
| T1 | running | 10 |
| T2 | running | 30 |

### Dirty Page Table

| pageID | recLSN |
|---|---|
| P5 | 10 |
| P6 | 20 |

### Buffer Pool

| | |
|---|---|
| P5 pageLSN=30 | P6 pageLSN=20 |
| | |

| LSN | prevLSN | transID | type | pageID | length | offset | before-image | after-image |
|---|---|---|---|---|---|---|---|---|
| 10 | null | T1 | update | P5 | 3 | 21 | ABC | DEF |
| 20 | null | T2 | update | P6 | 3 | 41 | HIJ | KLM |
| 30 | 20 | T2 | update | P5 | 3 | 20 | GDE | QRS |
| | | | | | | | | |

# Example of Recovery Structures

## Transaction Table

| transID | status | lastLSN |
|---|---|---|
| T1 | running | 10 |
| T2 | running | 30 |

## Dirty Page Table

| pageID | recLSN |
|---|---|
| P5 | 10 |
| P6 | 20 |

## Buffer Pool

| | |
|---|---|
| P5<br>pageLSN=30 | P6<br>pageLSN=20 |
| | |

| LSN | prevLSN | transID | type | pageID | length | offset | before-image | after-image |
|---|---|---|---|---|---|---|---|---|
| 10 | null | T1 | update | P5 | 3 | 21 | ABC | DEF |
| 20 | null | T2 | update | P6 | 3 | 41 | HIJ | KLM |
| 30 | 20 | T2 | update | P5 | 3 | 20 | GDE | QRS |
| 40 | 10 | T1 | update | P7 | 3 | 21 | TUV | WXY |

# Example of Recovery Structures

## Transaction Table

| transID | status | lastLSN |
|---|---|---|
| T1 | running | 40 |
| T2 | running | 30 |

## Dirty Page Table

| pageID | recLSN |
|---|---|
| P5 | 10 |
| P6 | 20 |
| P7 | 40 |

## Buffer Pool

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

| LSN | prevLSN | transID | type | pageID | length | offset | before-image | after-image |
|---|---|---|---|---|---|---|---|---|
| 10 | null | T1 | update | P5 | 3 | 21 | ABC | DEF |
| 20 | null | T2 | update | P6 | 3 | 41 | HIJ | KLM |
| 30 | 20 | T2 | update | P5 | 3 | 20 | GDE | QRS |
| 40 | 10 | T1 | update | P7 | 3 | 21 | TUV | WXY |

# Normal Execution

- Update transaction table on Xact start/end

- For each update:
  - Create log record with LSN $\ell$=++MaxLSN and prevLSN=TransTable[transID].lastLSN
  - Update TransTable[transID].lastLSN=$\ell$
  - If modified page not in dirty table, add it with recLSN=$\ell$

- 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

DB

RAM

## Log Records

LSN
prevLSN
transID
type
…

## Data pages

Each with a
pageLSN

## Master record

LSN of most
recent checkpoint

## Transaction Table

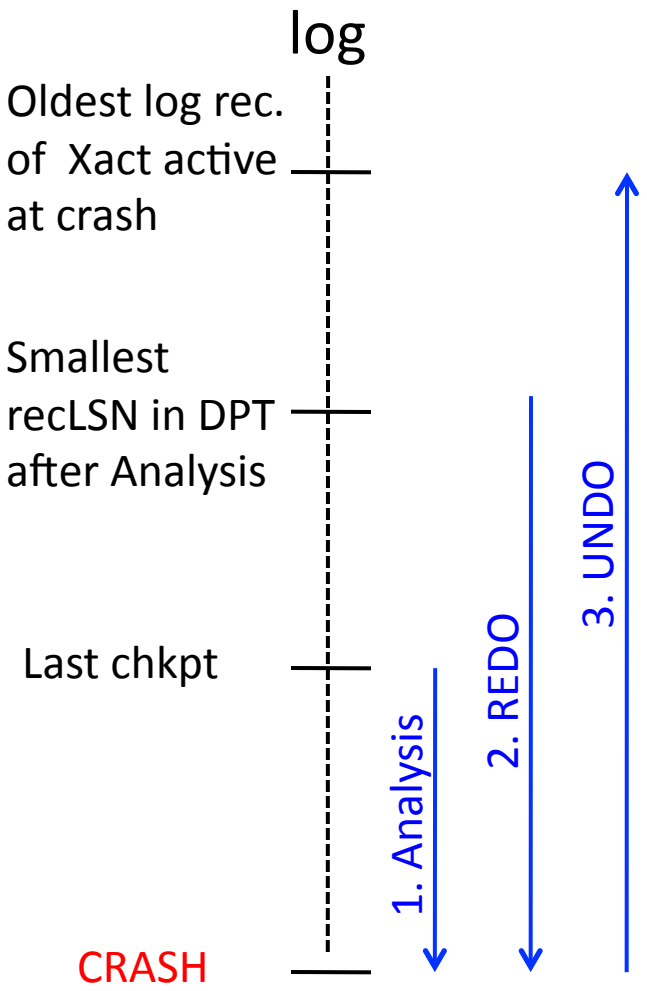
lastLSN
status

## 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)

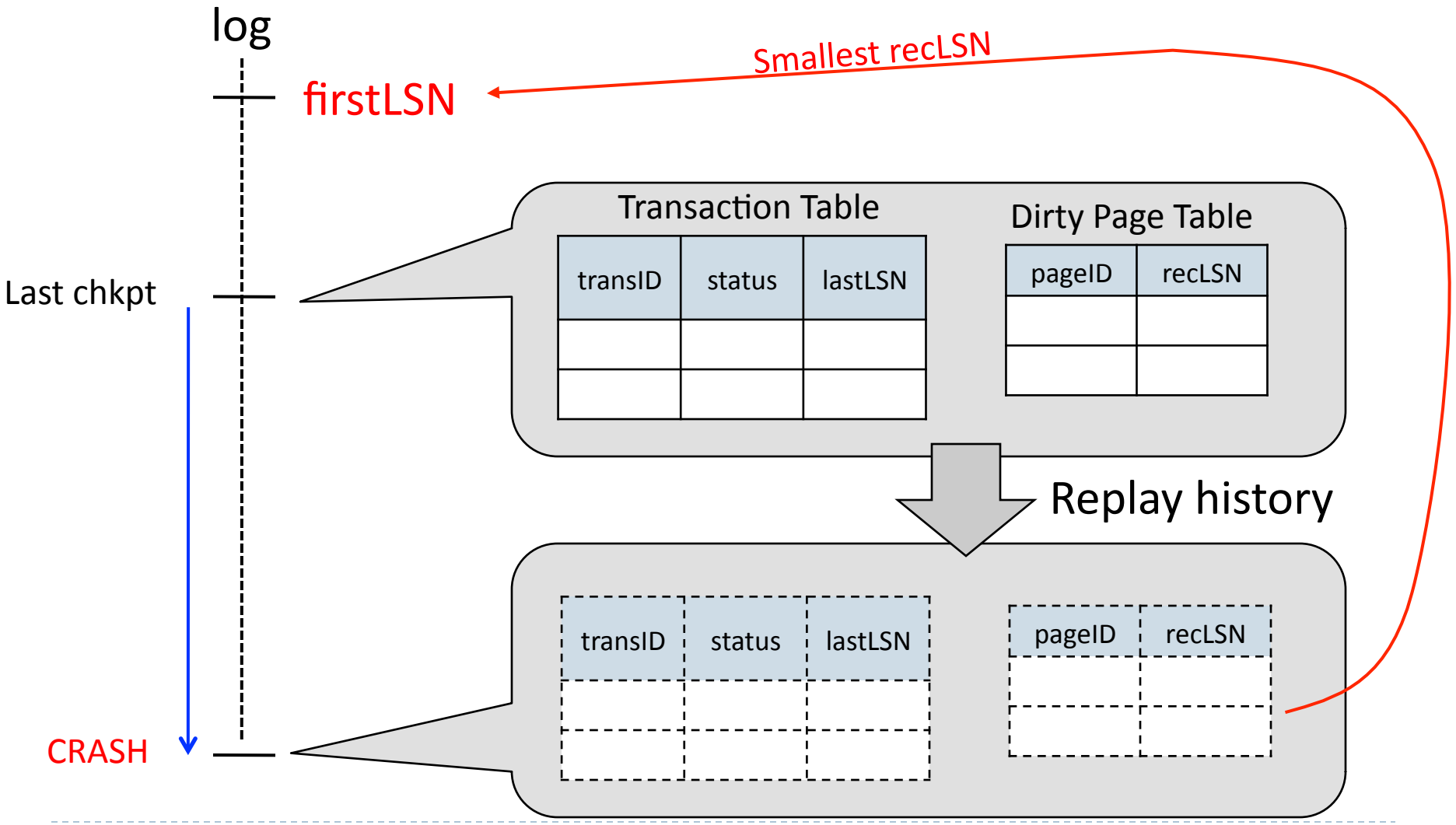
- Approach
  - Rebuild active transactions table and dirty pages table
  - Compute: firstLSN = smallest of all recLSN in DPT

# 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



# Phase 2: REDO

**Principles:**

- Scan the log forward from firstLSN ← Why start here?
- Read all records sequentially, and reapply all updates
- Do not record REDO actions in the log
- Needs the DPT

# 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:**

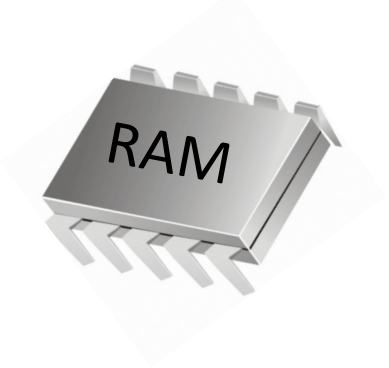
- Start from the end of the log, move backwards
- Read only affected log entries (loser Xacts)
- Undo actions logged as special entries: CLR (Compensation Log Records)
- CLRs are redone, but never undone

# Phase3: 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

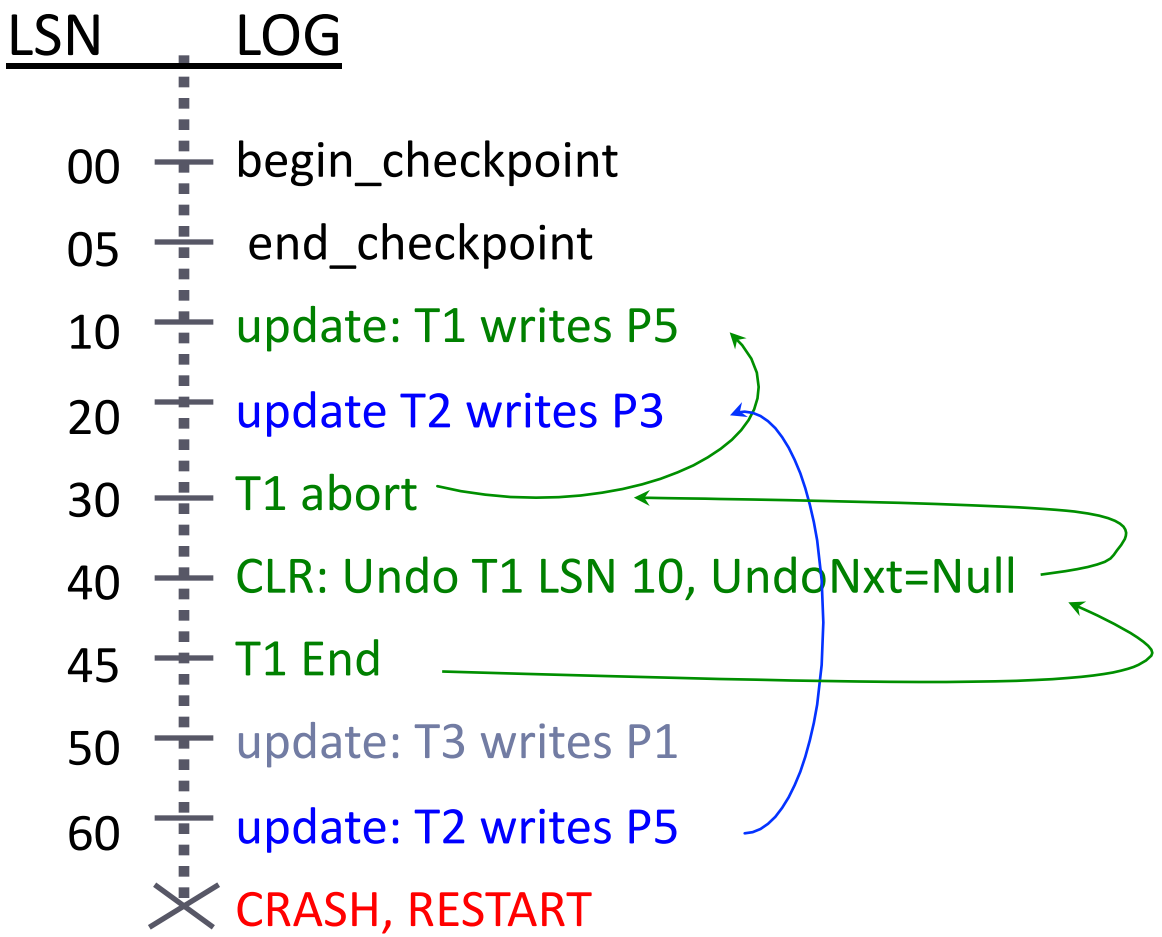
# Example of Recovery – (up to crash)




Xact Table
- lastLSN
- status

Dirty Page Table
- recLSN

flushedLSN

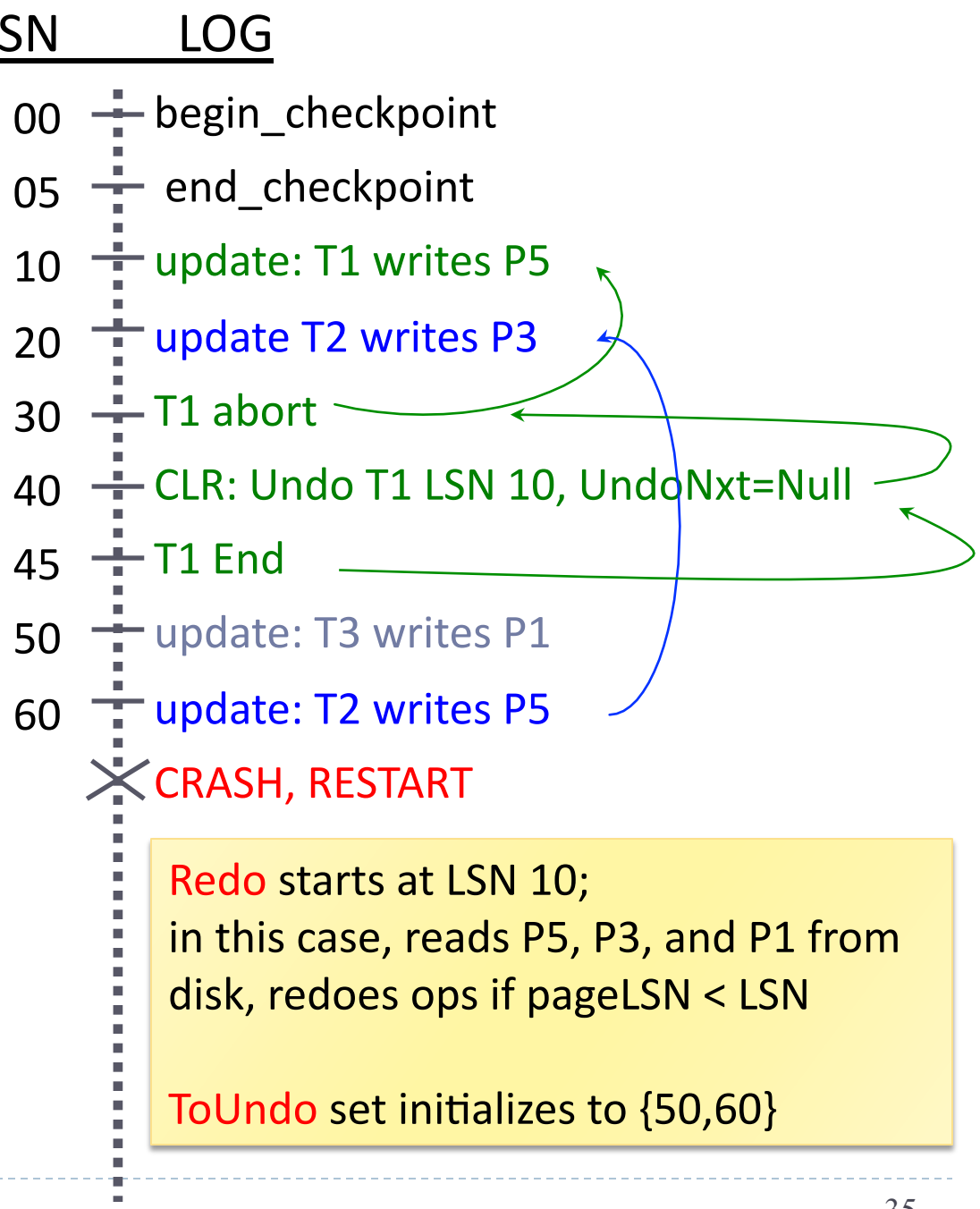


Trans Table

| Trans | lastLSN | Stat |
|---|---|---|
| T2 | 60 | r |
| T3 | 50 | r |

Dirty Page Table

| PageId | recLSN |
|---|---|
| P5 | 10 |
| P3 | 20 |
| P1 | 50 |
| | |



After Analysis & REDO:

ToUndo: {50, 60}

ToUndo: {50, 20}

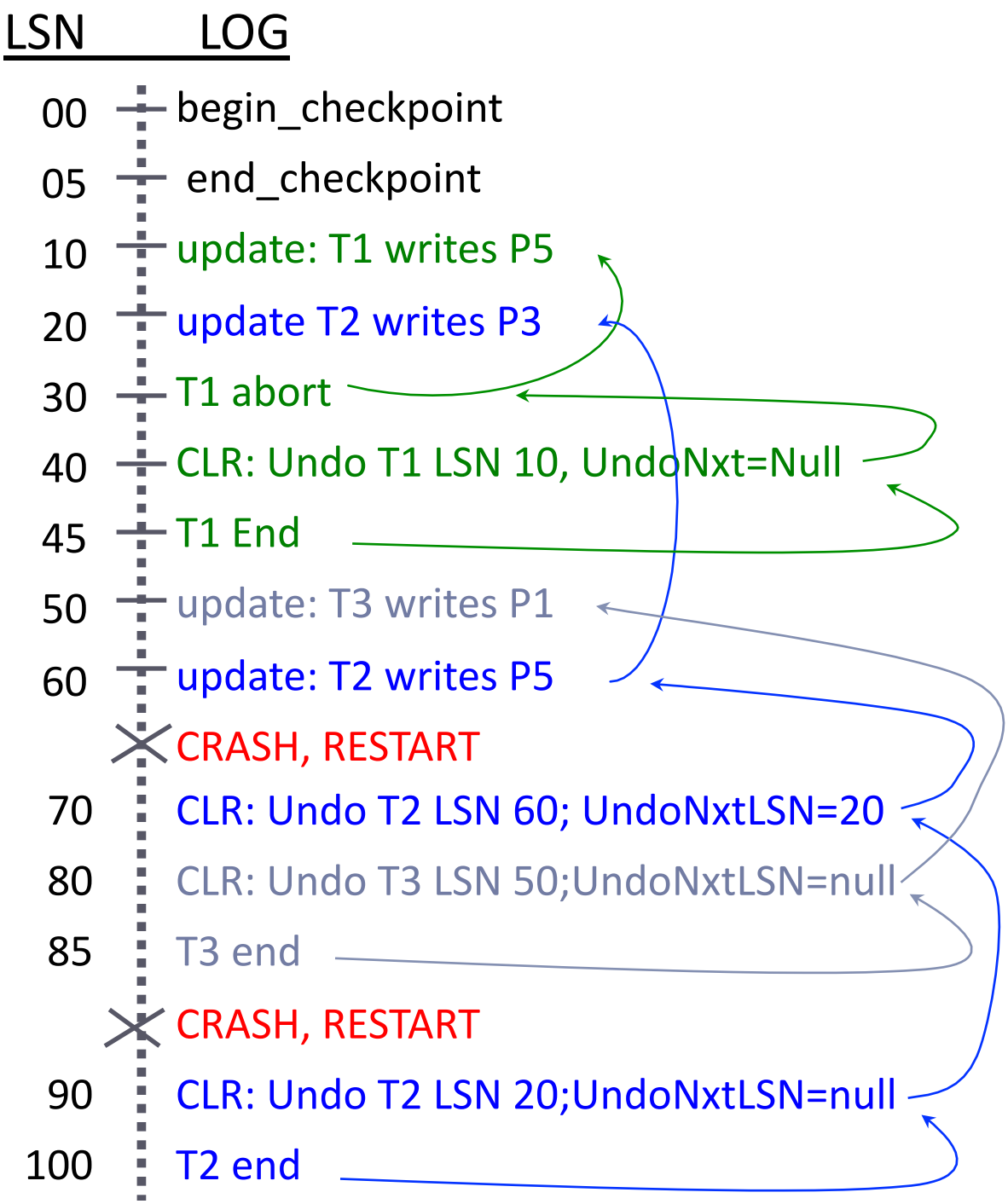
ToUndo: {20}

After Analysis & REDO:

ToUndo: {70}

ToUndo: {20}

ToUndo: {}



# 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?