

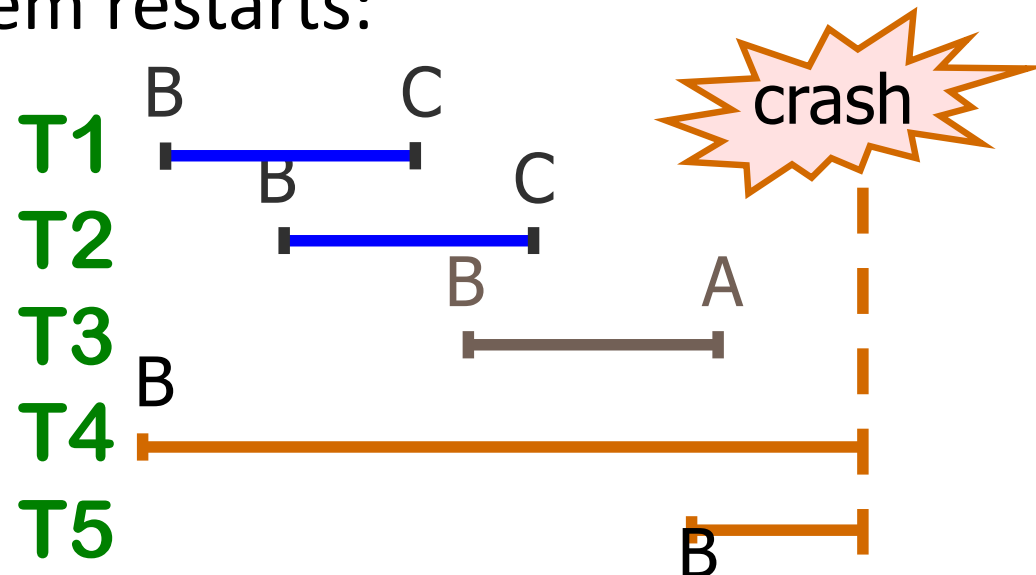
Spring 2017

# **LOGGING AND RECOVERY**

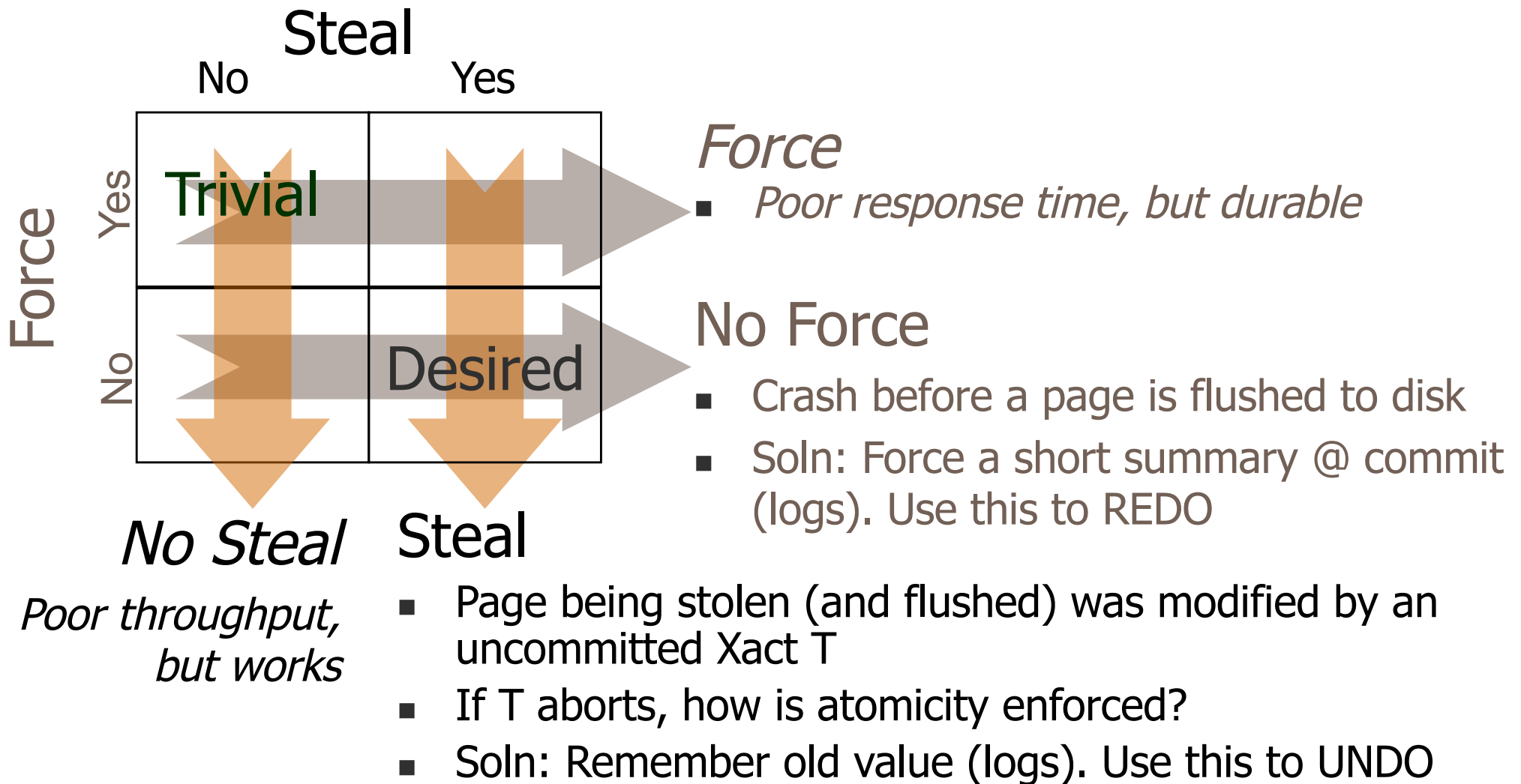
## **[CH 18]**

# Motivation

- Atomicity:
  - Transactions may abort (“Rollback”).
- Durability:
  - What if DBMS stops running? (Causes?)
- Desired Behavior after system restarts:
  - T1, T2 & T3 should be durable.
  - T4 & T5 should be aborted



# Buffer Pool: Sharing & Writing



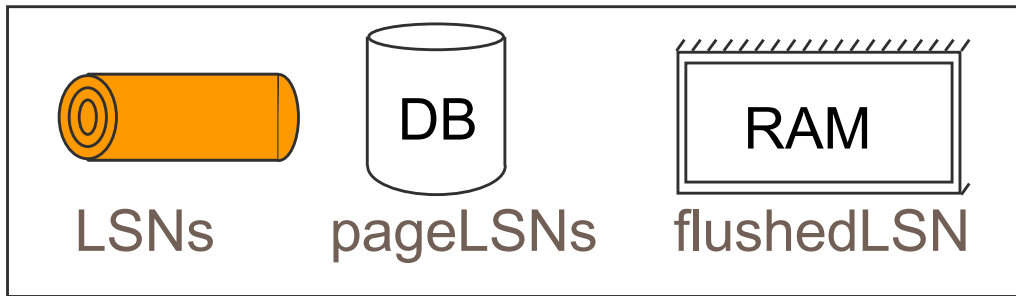
# Basic Idea: Logging

- Record information, for every change, in a *log*.
  - Sequential writes to log (put it on a separate disk).
  - Stored in stable storage to survive system crash
    - disk mirroring
  - Each record has a log sequence number (LSN)
  - Log record contains:
    - <prevLSN, XID, type, ... >
    - and additional control info (which we'll see soon)
    - Note: the log records for a transaction are chained by prevLSN

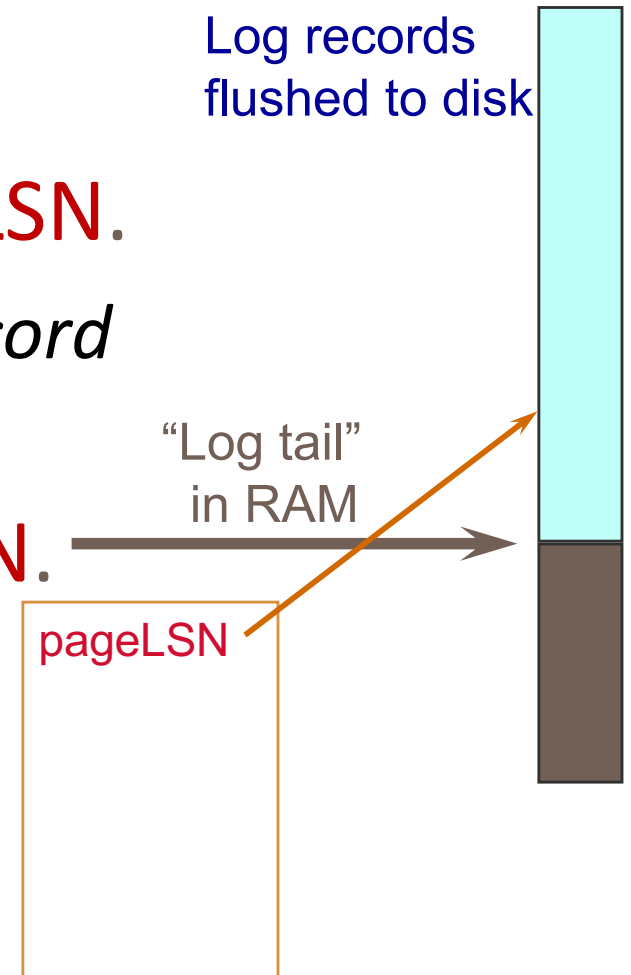
# Write-Ahead Logging (WAL)

- The **Write-Ahead Logging** Protocol:
  1. Must force the log record for an update before the corresponding data page gets to disk.
  2. Must write all log records for a Xact before commit.
- #1 guarantees Atomicity.
- #2 guarantees Durability.
- Exactly how is logging (and recovery!) done?
  - We'll study the ARIES algorithms
    - breakthrough in recovery algorithms!
    - repeating history paradigm
    - fine-granularity locking and logical logging

# WAL & the Log

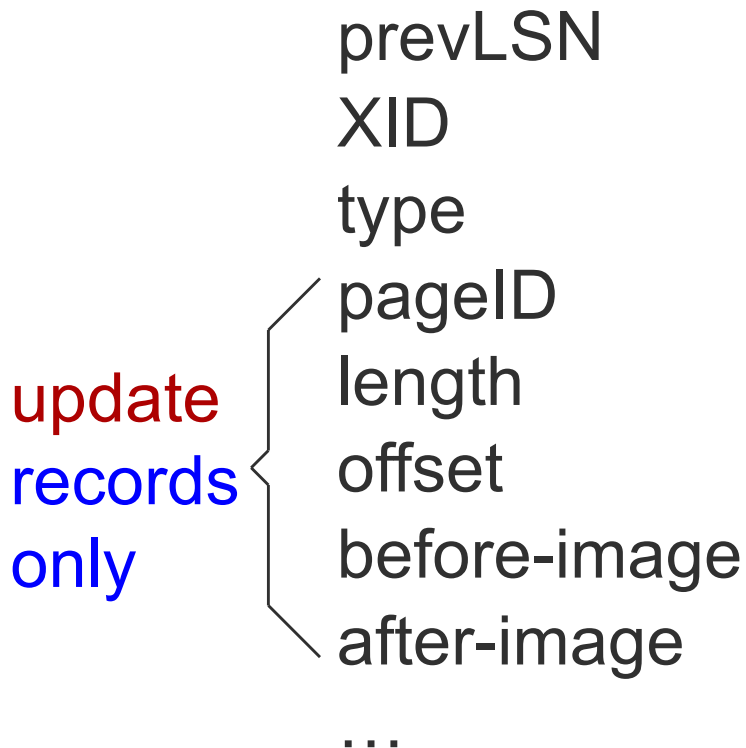


- Log Sequence Number (LSN).
  - Unique and always increasing.
- Each *data page* contains a **pageLSN**.
  - The LSN of the most recent *log record* that updated the page
- System keeps track of **flushedLSN**.
  - The max LSN flushed so far.
- WAL: *Before* a page is written,
  - $\text{pageLSN} \leq \text{flushedLSN}$



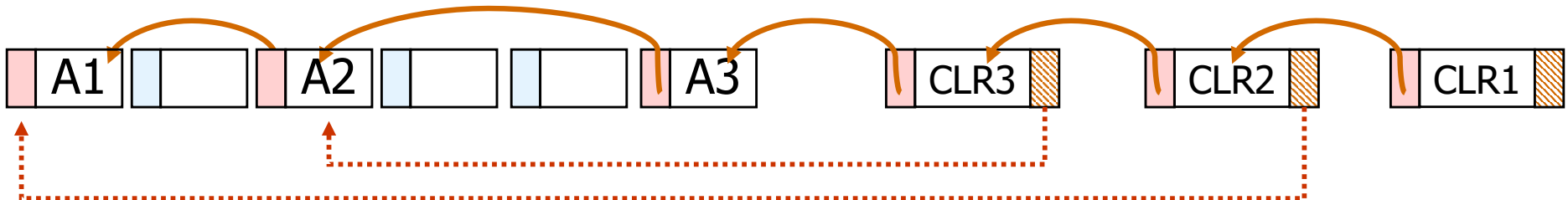
# Log Records

## LogRecord fields:



## Possible log record types:

- **Update**
- **Commit**
- **Abort**
- **End** (end of commit or abort)
- **Compensation Log Rec. (CLRs)**
  - For UNDO actions. When?
  - Contains **undoNextLSN**
    - Reverse chain of update logs
  - Contains before-image



# Other Log-Related State

- **Transaction Table**: One entry per active Xact.
  - Contains
    - **XID**: Transaction identifier
    - **status**: running/committed/aborted
    - **lastLSN**: LSN of the most recent log rec. for this Xact.
- **Dirty Page Table**: One entry per dirty page in BP
  - Contains **recLSN**:- LSN of the log record that ***first*** caused the page to be dirty.
    - Starting point for REDO



# Checkpointing

- **Checkpoint**: Snapshot of the database
  - Minimize recovery time
- Write to log:
  - **begin\_checkpoint** record: Indicates when chkpt began.
  - **end\_checkpoint** record:
    - Record Xact table and D.P.T.
    - Tables accurate only as of the time of the **begin\_checkpoint** record
    - No attempt to force dirty pages to disk
    - This is a **fuzzy checkpoint**
  - Store LSN of chkpt record in a safe place (**master** record).

# Normal Execution of an Xact

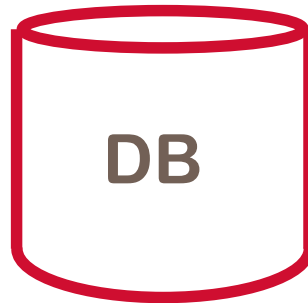
- Series of **reads** & **writes**, followed by **commit** or **abort**.
  - Updates are “in place”: i.e., data on disk is overwritten
  - We will assume that write is atomic on disk.
    - In practice, additional details to deal with non-atomic writes.
- **Strict 2PL**.
- STEAL, NO-FORCE buffer management, with **Write-Ahead Logging**.

# The Big Picture: What's Stored Where



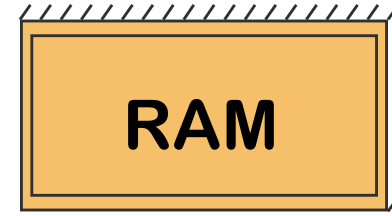
## LogRecords

prevLSN  
XID  
type  
pageID  
length  
offset  
before-image  
after-image



Data pages  
each  
with a  
pageLSN

master record



## Xact Table

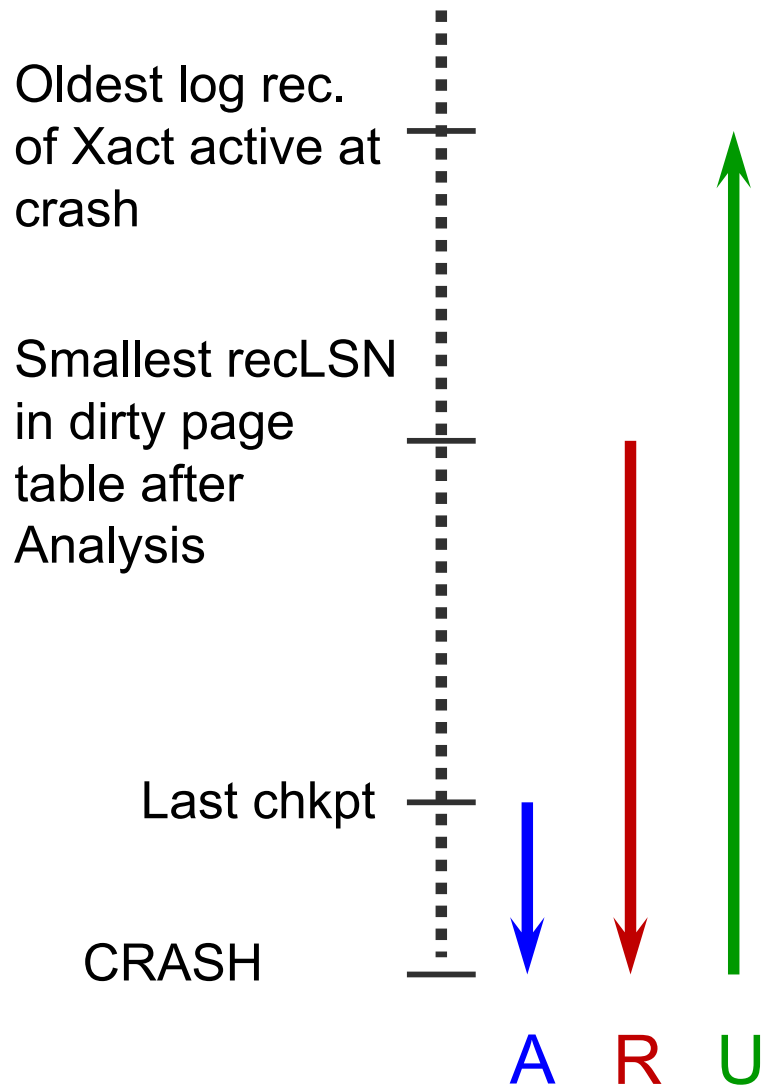
lastLSN  
status

## Dirty Page Table

recLSN

flushedLSN

# Crash Recovery: Big Picture



- Start from a **checkpoint** (found via **master** record).
- Three phases:
  - **Analysis**: Since checkpoint, find
    - Xacts that must be aborted (losers)
    - dirty pages at time of crash (conservative estimate)
  - **REDO** *all* actions.
    - repeat history
  - **UNDO** effects of losers

# Recovery: The Analysis Phase

- Compute
  - Set of dirty pages (conservative)
  - Uncommitted transactions at the crash point
- Scan log forward from checkpoint.
  - End record: Remove Xact from Xact table.
  - Other records: Add Xact to Xact table, set lastLSN=LSN
  - Commit record: change Xact status to commit.
  - Update or CLR record: If P not in Dirty Page Table,
    - Add P to D.P.T., set its recLSN=LSN.

# Recovery: The Analysis Phase

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

45 — T1 End

50 — update: T3 writes P1

60 — update: T2 writes P5

✗ **CRASH, RESTART**

prevLSNs

Transaction Table

XACT	LastLSN
T2	60
T3	50

D. P. T.

Page	recLSN
P1	50
P3	20
P5	10

# Recovery: The REDO Phase

- *Repeat History* to reconstruct state at crash:
  - Reapply *all* updates (even of aborted Xacts!), redo CLR's
  - Bring the database to the same state as @ crash
- Scan forward from log record containing smallest **recLSN** in D.P.T.

For each update log record or CLR, REDO the action unless we can verify that the change has already been written to disk:

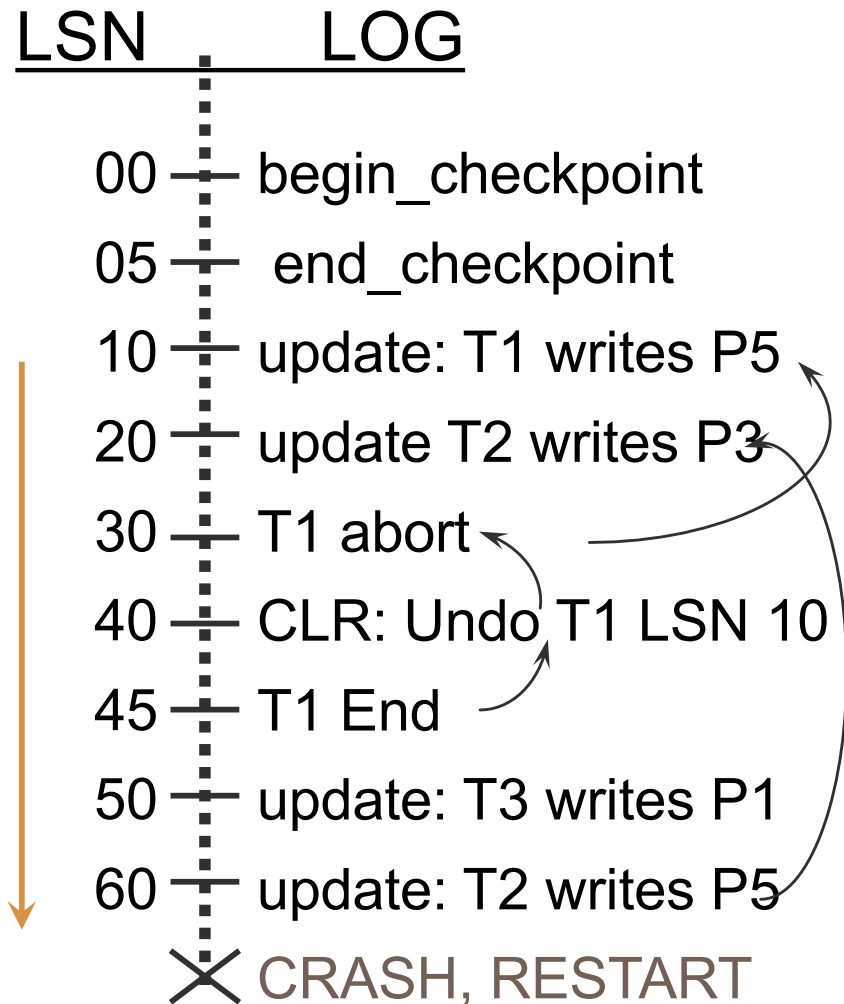
- Affected page is not in the Dirty Page Table, or
- Affected page is in D.P.T., but  $LSN < recLSN$ , or
  - update was propagated to disk
- $LSN \leq pageLSN$  (in DB)
  - requires fetching the page

# To REDO An Action

- Reapply logged action.
- Set pageLSN to LSN. No additional logging!
- Use of CLRs ensures that no change is ever carried out twice on the disk copy of an object.
  - For every “DO” there is one and only one “UNDO”
- At the end of REDO
  - Write END log recs for all committed Xacts.
  - Remove committed Xacts from the Xact table.



# Recovery: The REDO Phase



Transaction Table

XACT	LastLSN
T2	60
T3	50

D. P. T.

Page	recLSN
P1	50
P3	20
P5	10

# Recovery: The UNDO Phase

ToUndo = {lastLSNs of all “loser” Xact}

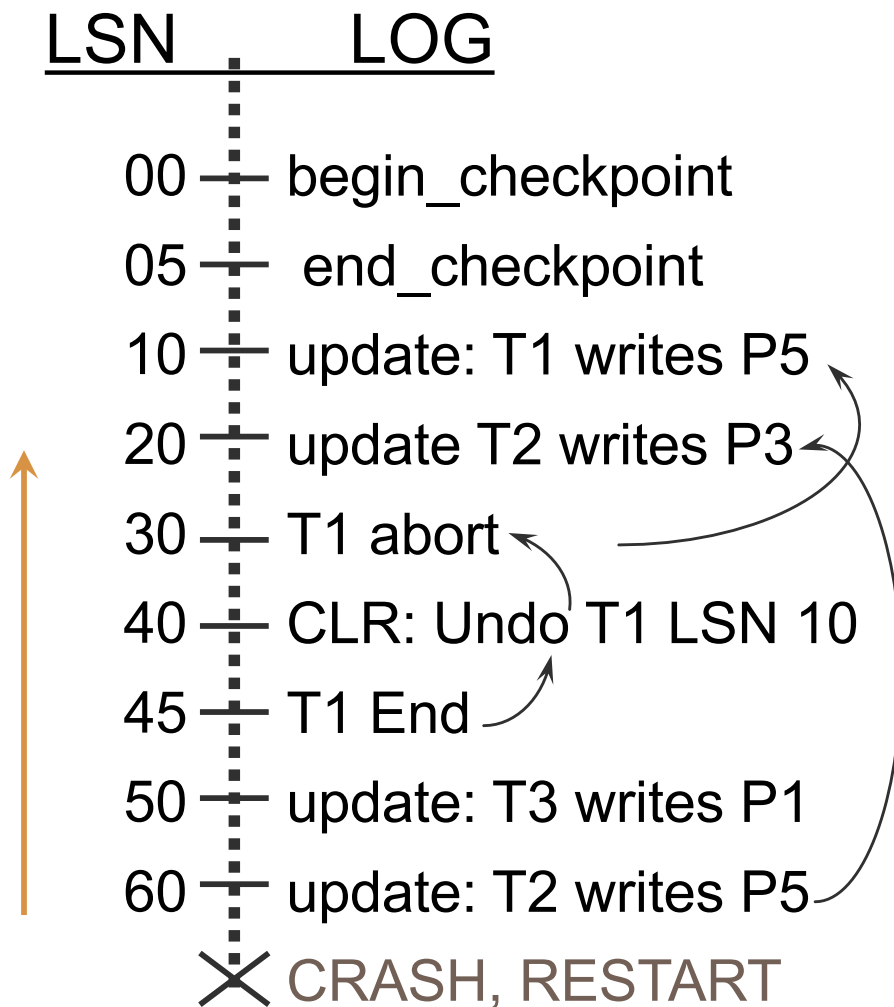
## Repeat:

- Choose largest LSN among ToUndo.
- If this LSN is a CLR and undonextLSN==NULL
  - Write an End record for this Xact.
- If this LSN is a CLR, and undonextLSN != NULL
  - Add undonextLSN to ToUndo
- Else this LSN is an update. Undo the update, write a CLR, add prevLSN to ToUndo.

Abort: special  
case of UNDO

**Until ToUndo is empty.**

# Recovery: The UNDO Phase



XACT	LastLSN
T2	60
T3	50

Page	recLSN
P1	50
P3	20
P5	10

LSN	LOG	(undoNextLSN)
70	CLR: Undo T2, LSN 60,	(20)
80	CLR: Undo T3, LSN 50,	(null)
85	T3 End	
90	CLR: Undo T2, LSN 20,	(null)
95	T2 End	

# Example: Crash During Restart!

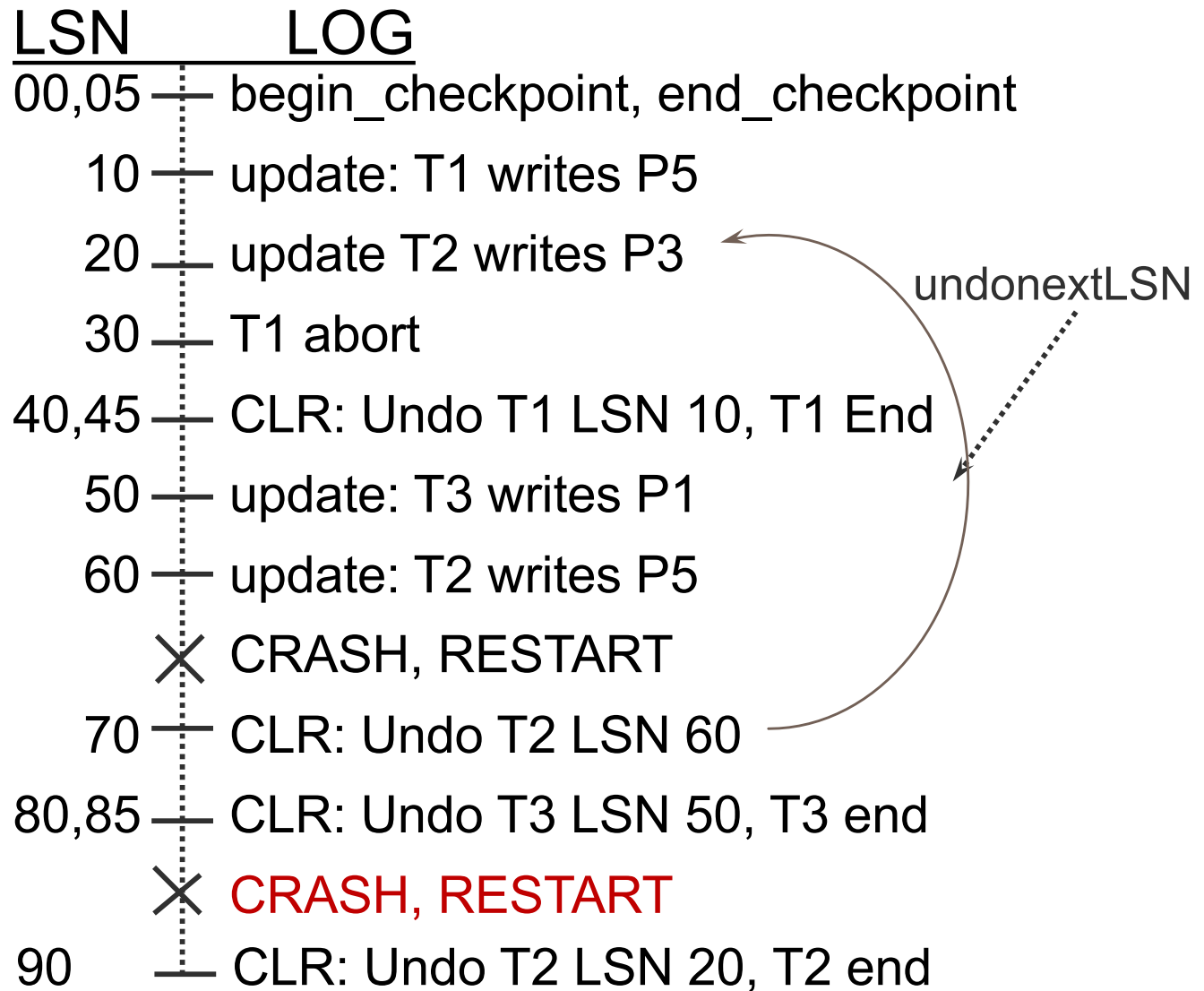
XACT	LastLSN
T2	70

Page	recLSN
P1	50
P3	20
P5	10

REDO: 10 to 85

UNDO:

- Undo 70, CLR
- Undo 20
- Take a ckpt



# Additional Crash Issues

- How do you limit the amount of work in REDO?
  - Flush asynchronously in the background.
  - Watch “hot spots”!
- How do you limit the amount of work in UNDO?
  - Avoid long-running Xacts.

# Media Recovery

- Used for disaster recovery.
- Based on periodically making a copy of the database
  - similar to a fuzzy checkpoint
- Apply logs to the copy of the object in the media to bring it up-to-date

# Summary of Logging/Recovery

- Atomicity & Durability.
- WAL to allow STEAL/NO-FORCE
- Checkpointing: A quick way to limit the amount of log to scan on recovery.
- Recovery works in 3 phases:
  - Analysis: Forward from checkpoint.
  - Redo: Forward from oldest recLSN.
  - Undo: Backward from end to first LSN of oldest Xact alive at crash.
- Upon Undo, write CLR.
- Redo “repeats history”: Simplifies the logic!
- **Interested in the history of ARIES:**
  - <http://www.sigmod.org/publications/ds-collection/discs-new/2000.1/out/slides/vldb/repeatinghistorc/index.pdf>