ICS 421 Spring 2010

Transactions & Recovery

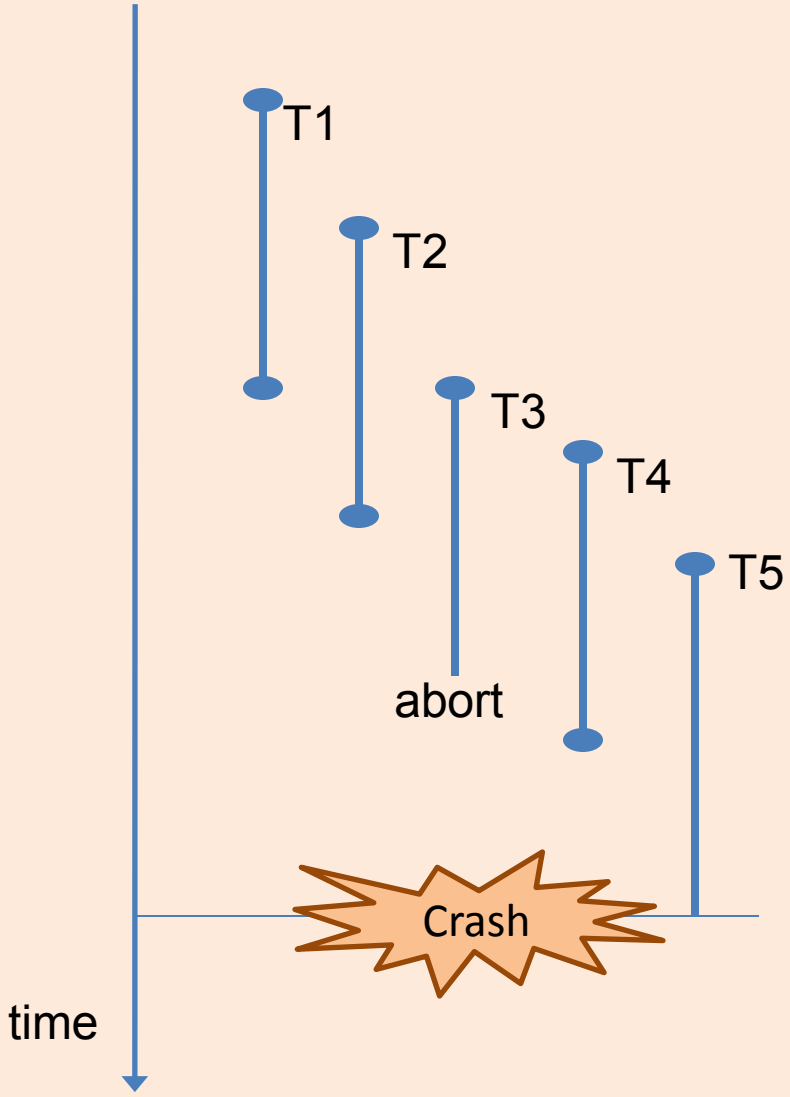
Asst. Prof. Lipyeow Lim

Information & Computer Science Department

University of Hawaii at Manoa

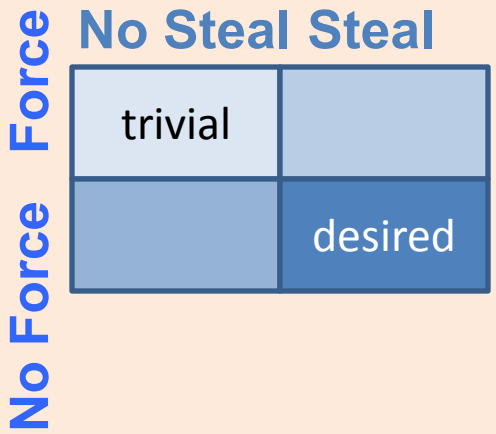
The Problem

- Atomicity
  - What happens when transactions abort ("rollback") ?
- Durability
  - What if DBMS crashes ?
- Desired behavior
  - What is the state before crash ?
  - What is the state after restart ?



Stealing Frames & Forcing Pages

- **Steal**: steal bufferpool frames from uncommited transactions
  - T1 updates row r
  - T2 needs to fetch a page
  - bufferpool is full and page containing r is chosen for eviction
  - Write page containing r back to disk (optimistic)
  - What happens if T1 aborts ?
- **Force**: force modified pages back to disk when a transaction commits.
  - If no-force is used, what happens after a crash ?



Write-Ahead Logging

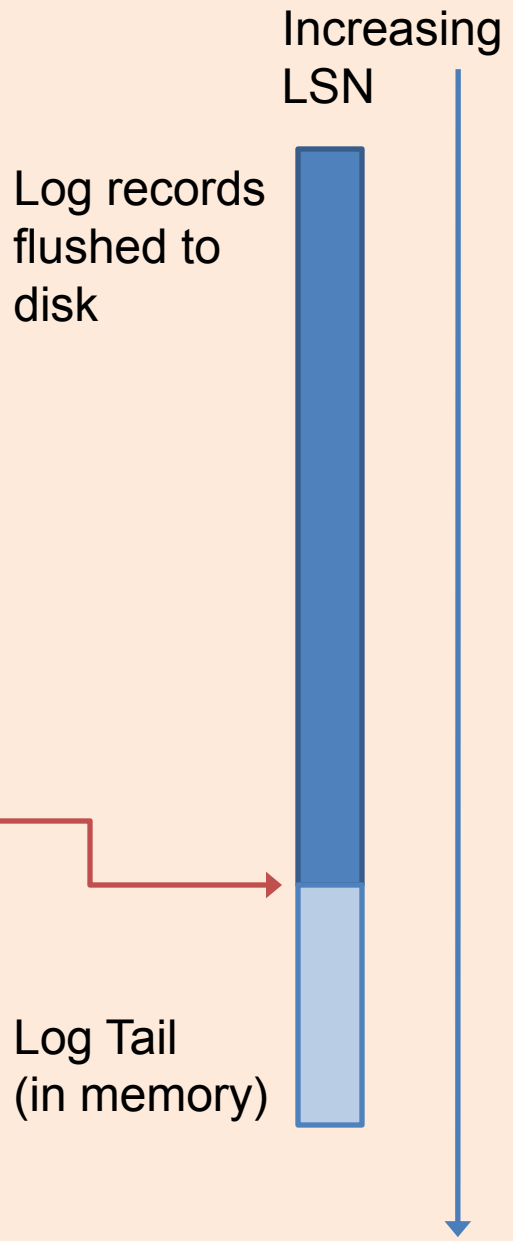
- Keep a log (aka trail, journal) of updates executed by DBMS on disk.
- The Write-Ahead Logging Protocol:
  - ①Must force the log record for an update *before* the corresponding data page gets to disk.
  - ② Must write all log records for a Xact *before commit*.
- Recover using ARIES algorithm

Atomicity

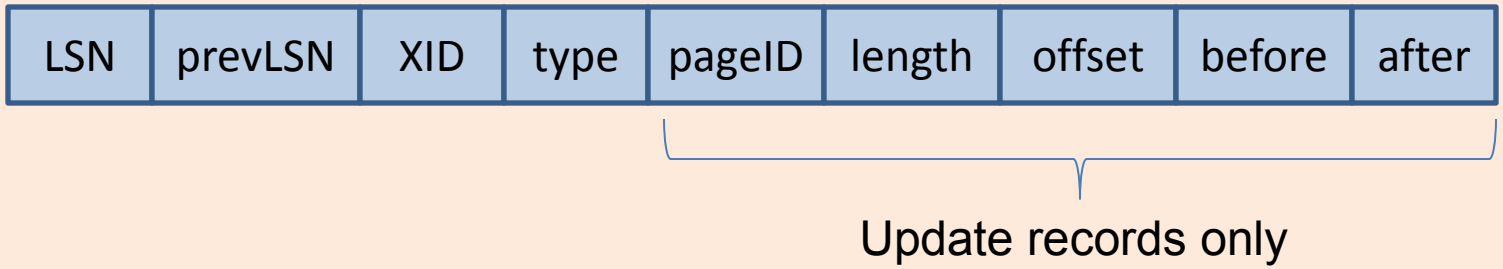
Durability

The Log

- Each log record has a unique Log Sequence Number (LSN).
  - LSNs always increasing.
- Each *data page* contains a pageLSN.
  - The LSN of the most recent *log record* for an update to that page.
- System keeps track of flushedLSN.
  - The max LSN flushed so far.
- WAL: *Before* a page is written,
  - pageLSN ≤ flushedLSN



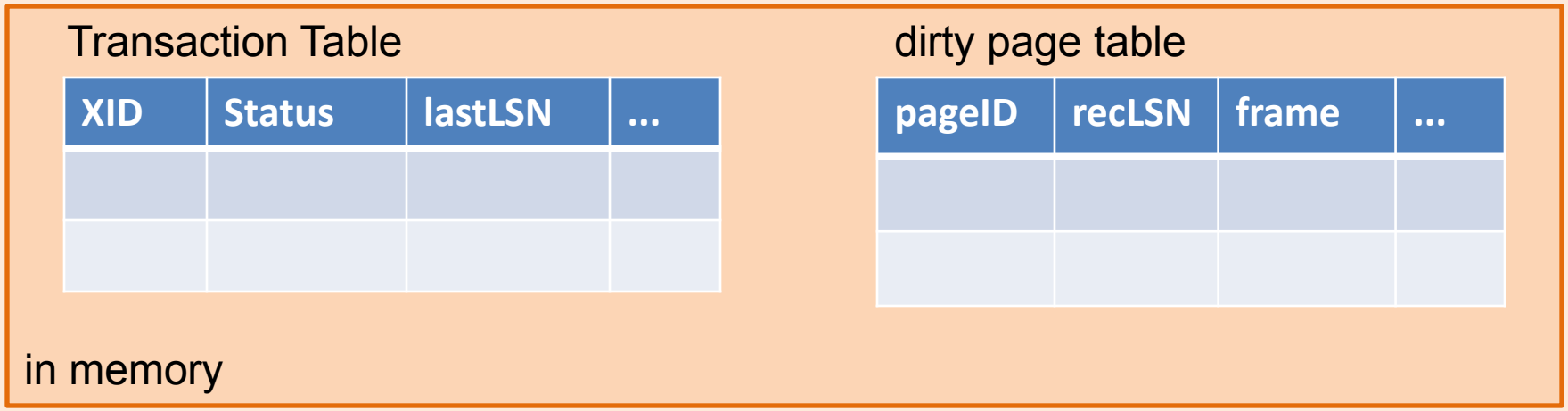
Log Records



Possible log record types:

- **Update**
- **Commit**
- **Abort**
- **End** (signifies end of commit or abort)
- Compensation Log Records (CLRs)
  - for UNDO actions

Other Log-Related State



- Transaction Table:
  - transaction manager
  - One entry per active Xact.
  - Contains XID, status (running/commited/aborted), and lastLSN.
- Dirty Page Table:
  - buffer manager
  - One entry per dirty page in buffer pool.
  - Contains recLSN -- the LSN of the log record which ***first*** caused the page to be dirty

Transaction Abort/Rollback

- No crash. Transaction aborted explicitly.
- UNDO updates using Log.
  - Get lastLSN of Xact from Xact table.
  - Can follow chain of log records backward via the prevLSN field.
  - Before starting UNDO, write an *Abort* log record.
    - For recovering from crash during UNDO!

T1
R(P1)
W(P1)
W(P2)
W(P1)
Abort

Transaction Table

XID	Status	lastLSN
T1	prog	~~110~~ ~~120~~ 130

dirty page table

pageID	recLSN
P1	110
P2	120

LSN	prevLSN	XID	type	pageID	...
110	0	T1	upd	P1	...
120	110	T1	upd	P2	...
130	120	T1	upd	P1	...

Abort : Nitty Gritty

- To perform UNDO, must have a lock on data!
  - No problem!
- Before restoring old value of a page, write a CLR:
  - You continue logging while you UNDO!!
  - CLR has one extra field: undonextLSN
    - Points to the next LSN to undo (i.e. the prevLSN of the record we're currently undoing).
  - CLRs *never* Undone (but they might be Redone when repeating history: guarantees Atomicity!)
- At end of UNDO, write an "end" log record.

Checkpointing

- Periodically, the DBMS creates a checkpoint, in order to minimize the time taken to recover in the event of a system crash. Write to log:
  - begin_checkpoint record: Indicates when chkpt began.
  - end_checkpoint record: Contains current *Xact table* and *dirty page table*. This is a `fuzzy checkpoint':
    - Other Xacts 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; effectiveness of checkpoint limited by oldest unwritten change to a dirty page. (So it's a good idea to periodically flush dirty pages to disk!)
  - Store LSN of chkpt record in a safe place (*master* record).

What's Stored Where

RAM

flushedLSN, **Xact Table** (XID, lastLSN, status)

Dirty Page Table (pageID, recLSN)

Log

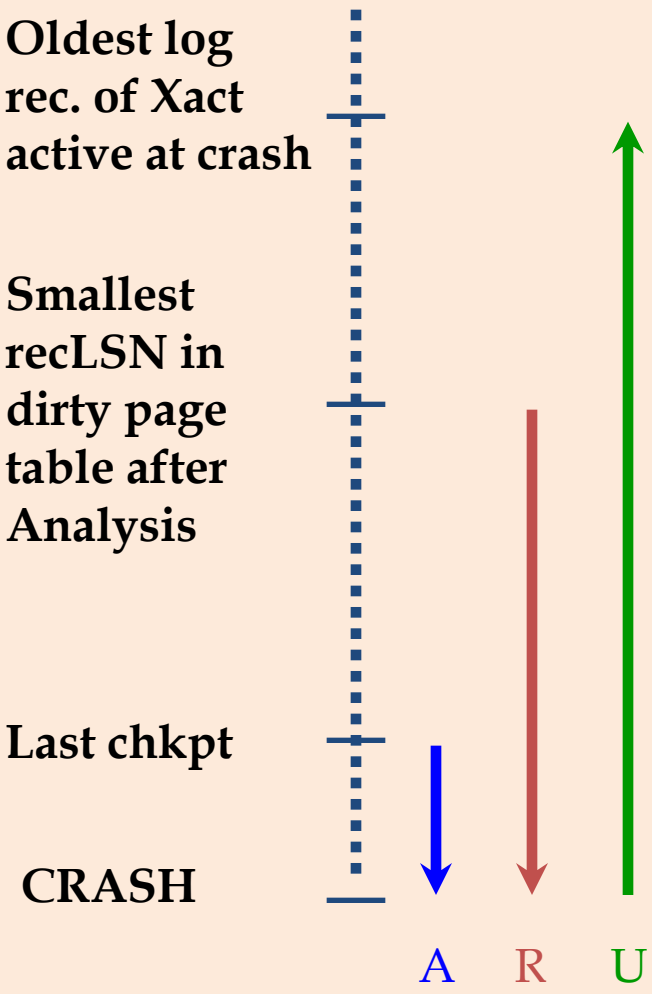
LogRec(LSN, prevLSN,XID,type,pageID len, offset, before, after)

DB on Disk

Data Pages with pageLSN

master record

ARIES Recovery Algorithm



- Start from a checkpoint (found via master record).
- Three phases. Need to:
  - Analyze : Figure out which Xacts committed since checkpoint, which failed.
  - REDO ***all*** actions.
    - (repeat history)
  - UNDO effects of failed Xacts

Analysis Phase

- Reconstruct state at checkpoint.
  - via end_checkpoint record.
- Scan log forward from checkpoint.
  - End record: Remove Xact from Xact table.
  - Other records: Add Xact to Xact table, set lastLSN=LSN, change Xact status on commit.
  - Update record: If P not in Dirty Page Table,
    - Add P to D.P.T., set its recLSN=LSN.

REDO Phase

- We *repeat History* to reconstruct state at crash:
  - Reapply *all* updates (even of aborted Xacts!), redo CLRs.
- Scan forward from log rec containing smallest recLSN in D.P.T. For each CLR or update log rec LSN, REDO the action unless:
  - Affected page is not in the Dirty Page Table, or
  - Affected page is in D.P.T., but has recLSN > LSN, or
  - pageLSN (in DB) ≥ LSN.
- To REDO an action:
  - Reapply logged action.
  - Set pageLSN to LSN. No additional logging!

UNDO Phase

ToUndo={ / | / a lastLSN of a "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.

Until ToUndo is empty.

Example: Crash Recovery

- One transaction.
- Checkpoint has empty Xact & dirty page tables.
- Analysis Phase:
  - rebuilds Xact table & dirty page
- REDO
  - sync on disk data pages up to crash
- UNDO
  - rollback all uncommitted transactions at time of crash

T1
R(P1)
W(P1)
W(P2)
W(P1)
CRASH!

Transaction Table

XID	Status	lastLSN

dirty page table

pageID	recLSN

flushedLSN

LSN	prevLSN	XID	type	pageID	...
90	0	0	begin checkpoint		
100	90	0	end checkpoint		
110	0	T1	upd	P1	...
120	110	T1	upd	P2	...
130	120	T1	upd	P1	...