CS 4604: Introduction to Database Management Systems

Logging and Recovery 2: ARIES

Virginia Tech CS 4604 Sprint 2021

Instructor: Yinlin Chen

VIRGINIA TECH

Today's Topics

- ARIES
  - Log Sequence Number (LSN)
  - Fuzzy checkpoints
  - Recovery algorithm

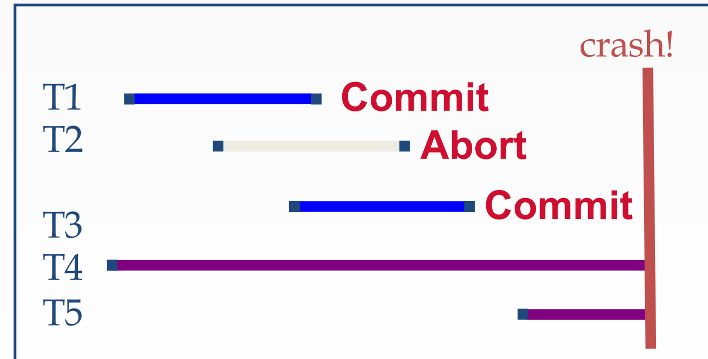
Recap

- Undo / Redo Logging
- Write-Ahead Log, for loss of volatile storage, with incremental updates (STEAL, NO FORCE) and checkpoints
- On recovery: **undo** uncommitted; **redo** committed transactions

Motivation

- Atomicity:
  - Transactions may abort ("Rollback").
- Durability:
  - What if DBMS stops running? (Causes?)

- Desired state after system restarts:
- – T1 & T3 should be durable.
- – T2, T4 & T5 should be aborted (effects not seen).



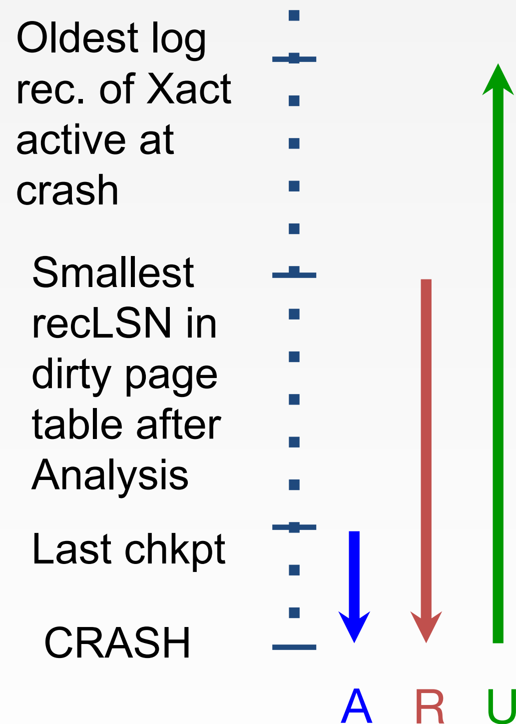
Today: ARIES

- Algorithms for Recovery and Isolation Exploiting Semantics
- With full details on
  - fuzzy checkpoints
  - recovery algorithm

C. Mohan (IBM)

Overview of ARIES

- A recovery algorithm is designed to implement a steal, no-force approach
- Start from a **checkpoint**
  - found via master record.
- Three phases:
  - **Analysis** - Figure out which Xacts committed since checkpoint, which failed.
  - **REDO** all actions.
    - (repeat history)
  - **UNDO** effects of failed Xacts.



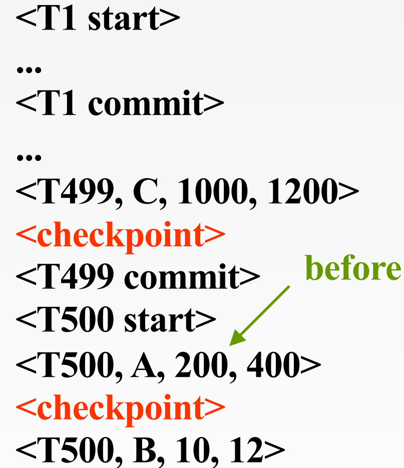
Checkpoint

- Idea: save the state the database periodically so that we don't need to always process the entire log
- During a checkpoint:
  - Stop accepting new transactions
  - Wait until **all** current transactions complete (i.e., commit / abort)
  - Flush log to disk
  - Flush all dirty pages to disk
  - Write a <CKPT> log record, flush log again
  - At this point, changes by committed txns have persisted to disk, and aborted txns have rolled back
  - Resume transactions

checkpoints

Write on the log:

- the id-s of active **transactions** and
- the id-s (ONLY!) of **dirty pages** (rest: obviously made it to the disk!)



Undo Recovery with Checkpointing

During recovery,
Stop at first <CKPT>

...
...
<T9,X9,v9>
...
...
(all completed)
<CKPT>
<START T2>
<START T3>
<START T5>
<START T4>
<T1,X1,v1>
<T5,X5,v5>
<T4,X4,v4>
<COMMIT T5>
<T3,X3,v3>
<T2,X2,v2>

All txns here are completed
No need to recover
Can truncate this part of the log

Txns T2,T3,T4,T5 need to be recovered

Fuzzy Checkpointing

- Problem with checkpointing: database freezes during checkpoint
  - Not accepting any new transactions!
- Would like to checkpoint while database still processes incoming txns
- Idea: *fuzzy* checkpointing
  - Save state of all txns and page statuses
    - Some txns can be running and dirty pages not flushed yet!
    - Need new **data structures** to store such info

Fuzzy Checkpointing: Idea

- Keep track of:
  1. txn states (running, committing, etc)
  2. dirty pages and which txn’s action first caused page to become dirty
- Save 1 and 2 to disk at checkpoint
- At recovery:
  - Re-create 1 and 2 from the log
  - Re-create running txns and dirty pages in memory
  - Replay rest of the log

Fuzzy Checkpointing: idea

Specifically, write to log:

- begin_checkpoint record: indicates start of ckpt
- 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.

solved both problems of non-fuzzy ckpts!!

Fuzzy Checkpointing: idea

And:

- Store LSN of most recent chkpt record on disk (master record)

- Data Structures
  - LSN and Page LSN
  - Dirty page table
  - Transaction table

VIRGINIA TECH™

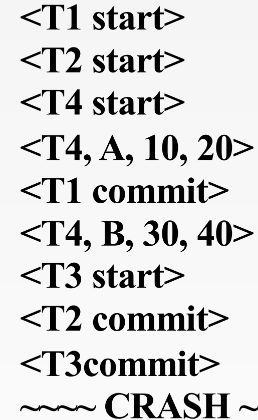
Fuzzy Checkpointing: Data Structures

- Each log record has a **Log Sequence Number** (LSN)
  - A unique integer that's increasing (e.g., line number)
- Each data page has a **Page LSN**
  - The LSN of the most recent log record that updated that page

Log Sequence Number (LSN)

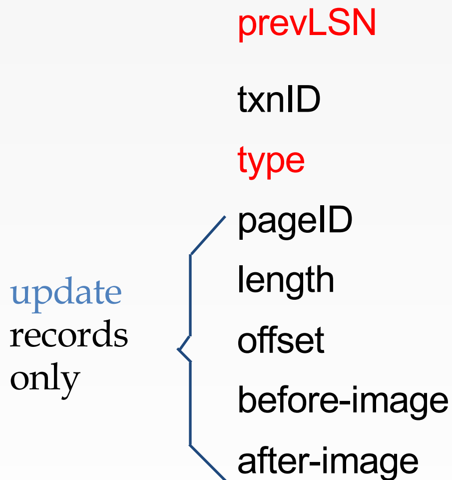
Log (WAL)

LSN	prevLSN	txnID	pageID	Log Payload
101	-	T100	-	START
102	-	T200	-	START
103	102	T200	P6	<old val, new val>
104	101	T100	P5	<old val, new val>



E.g., undo T4 - it is faster, if we have a linked list of the T4 log records

Log Records



Possible log record types:

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

Fuzzy Checkpointing: Data Structures

#1) Transaction Table

- In-memory table
- Lists all txn's and their statuses
- Contains
  - txnID
  - Status: running/committing/aborting
  - lastLSN: most recent update LSN written by txn (if active)

Transactions

txnID	lastLSN	Status
T100	104	commit
T200	103	abort

Fuzzy Checkpointing: Data Structures

#2) **Dirty Page Table**:

- One entry per dirty page currently in buffer pool
- Contains recoveryLSN (recLSN) -- the LSN of the log record which ***first*** caused the page to become dirty

Dirty pages table

pageID	recLSN
P5	102
P6	103
P7	101

Writing log records

Log (WAL)

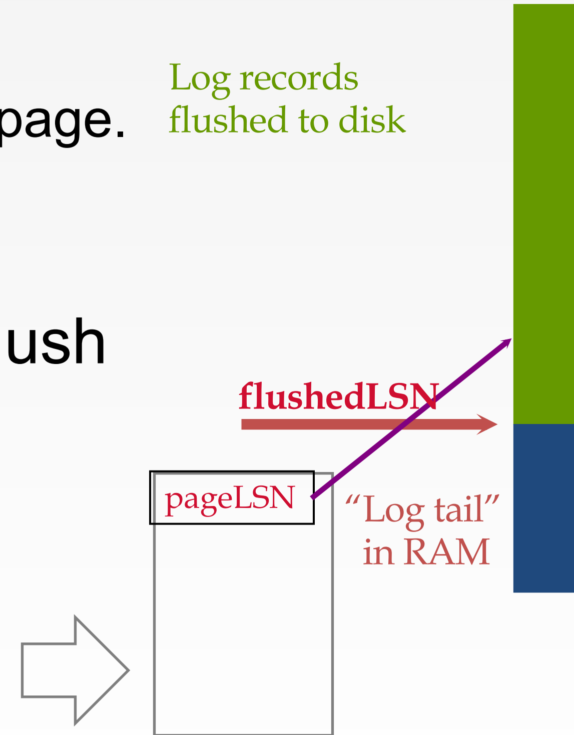
LSN	prevLSN	txnID	pageID	Log Payload
101	-	T100	-	START
102	-	T200	-	START
103	102	T200	P6	<old val, new val>
104	101	T100	P5	<old val, new val>

- Store both old and new values in update records
- New field **prevLSN** = LSN of the previous log record written by this txnID
- Actions of a transaction form a linked list backwards in time

WAL & the Log

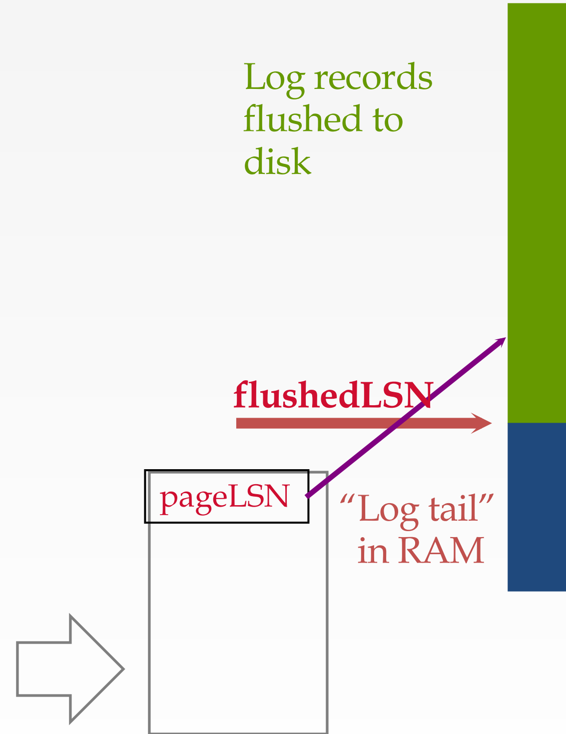
- Each data page contains a pageLSN.
  - The LSN of the **most recent** update to that page.
- System keeps track of flushedLSN.
  - The max LSN flushed so far.
- WAL: For a page *i* to be written must flush log at least to the point where:

$$pageLSN_i \leq flushedLSN$$



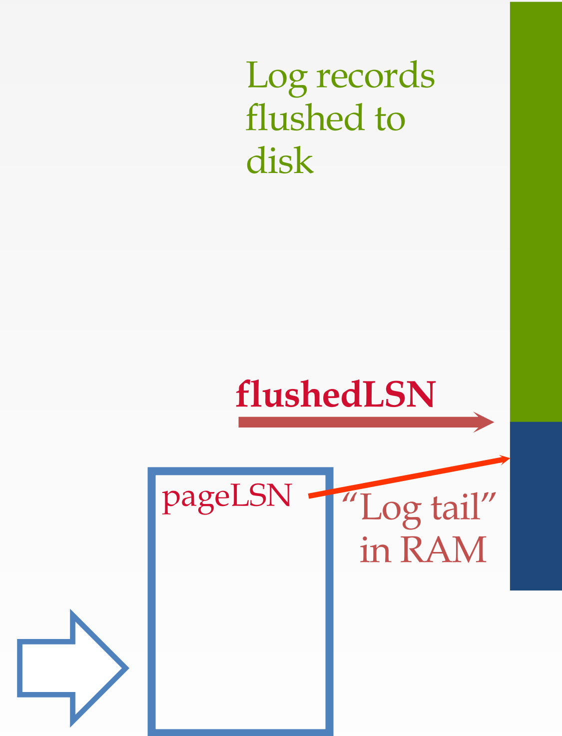
WAL & the Log

- Can we un-pin the gray page?
- A: yes



WAL & the Log

- Can we un-pin the blue page?
- A: no



Normal Execution of an Xact

- Series of reads & **writes**, followed by **commit** or **abort**
  - The recovery manager sees page-level reads/writes
  - We will assume that disk write is atomic.
    - In practice, additional details to deal with non-atomic writes.
- STEAL, NO-FORCE buffer management, with **Write-Ahead Logging**
  - Update, Commit, Abort log records written to log tail as we go
  - Transaction Table and Dirty Page Table being kept current
  - PageLSNs updated in buffer pool
  - Log tail flushed to disk periodically in background
    - And flushedLSN changed as needed
  - Buffer manager stealing pages subject to WAL

Fuzzy Checkpointing: Protocol

- Write a <BEGINCKPT> to log
- Flush log to disk
- Continue normal operation
- When DPT and Transactions tables are written to the disk, write <END CKPT> to log
- Flush log to disk

Dirty pages table (DPT)

pageID	recLSN
P5	102
P6	103
P7	101

Transactions

txnID	lastLSN	Status
T100	104	commit
T200	103	abort

Transaction Commit

- Write commit record to log.
- All log records up to Xact's commit record are flushed to disk.
  - Guarantees that flushedLSN ≥ lastLSN.
  - Note that log flushes are sequential, synchronous writes to disk.
  - Many log records per log page.
- Commit() returns.
- Write end record to log.

Example

LSN	*prevLSN*	*tid*	*type*	*item*	*old*	*new*
10	NULL	T1	update	X	30	40
….						
50	10	T1	update	Y	22	25
…						
63	50	T1	commit			
…						
68	63	T1	end			

dbms flushes
log records
+ some
record-keeping

Simple Transaction Abort

- For now, consider an explicit abort of a Xact.
  - No crash involved.
- We want to “play back” the log in reverse order, UNDOing updates.
  - Get **lastLSN** of Xact from Xact table.
  - Write an **Abort** log record before starting to rollback operations
  - Can follow chain of log records backward via the prevLSN field.
  - Write a “**CLR**” (compensation log record) for each undone operation.

Note: CLRs are a different type of log record

Abort - Example

LSN	*prevLSN*	*tid*	*type*	*item*	*old*	*new*
10	NULL	T2	update	Y	30	40
...						
63	10	T2	abort			

Abort - Example

LSN	*prevLSN*	*tid*	*type*	*item*	*old*	*new*
10	NULL	T2	update	Y	30	40
…						
63	10	T2	abort			
…						
72	63	T2	CLR	(LSN 10)		
…						
78	72	T2	end			

compensating log record (LSN 72)

VIRGINIA TECH™

Abort - Example

LSN	*prevLSN*	*tid*	*type*	*item*	*old*	*new*	*undoNextLSN*
10	NULL	T2	update	Y	30	40	
…							
63	10	T2	abort				
…							
72	63	T2	CLR	Y	40	30	NULL
…							
78	72	T2	end				

Compensation Log Record (CLR)

- A CLR record has all the fields of an ‘update’ record
- CLR has one extra field: undonextLSN
  - points to the next LSN to undo
- You continue logging while you UNDO!!
- CLR contains REDO info
- CLRs never Undone
  - Undo needn’t be idempotent (>1 UNDO won’t happen)
  - But they might be Redone when repeating history
    - (=1 UNDO guaranteed)
- At end of all UNDOs, write an “end” log record

VIRGINIA TECH™

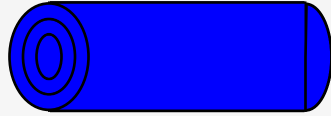
Abort - algorithm:

- First, write an ‘abort’ record on log and
- Play back updates, in **reverse order**: for each update
  - write a CLR log record
  - restore old value
- at end, write an ‘end’ log record

Note: CLR records never need to be undone

VIRGINIA TECH™

The Big Picture: What's Stored Where

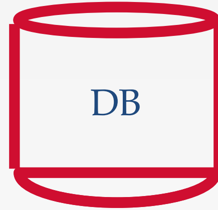


LogRecords

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

update CLR: (pageID, length, offset, before-image, after-image)

CLR: (*undoNextLSN*)



Data pages
each with a pageLSN

master record
LSN of most recent checkpoint

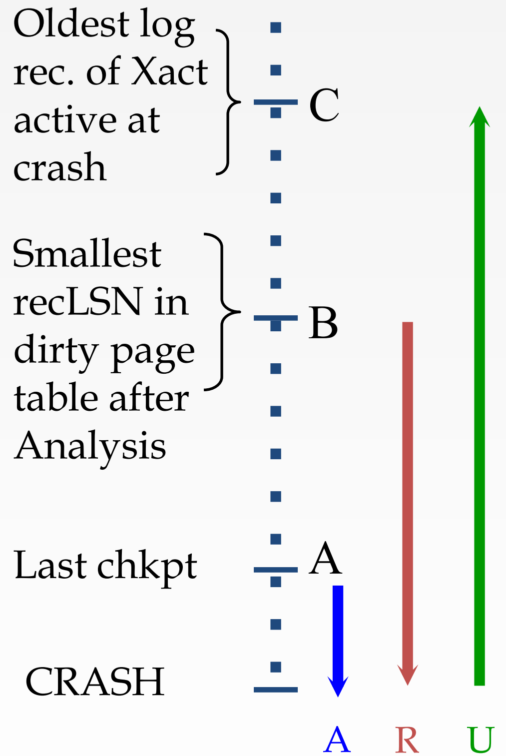
Xact Table
lastLSN
status

Dirty Page Table
recLSN

flushedLSN

VIRGINIA TECH™

Crash Recovery: Big Picture



- Start from a checkpoint (found via master record)
- Three phases
  - Analysis - Figure out which Xacts committed since checkpoint, which failed.
  - REDO all actions (repeat history) before crash and bring DBMS to the exact state right when it crashed
  - UNDO effects of failed Xacts when crash occurred. Log all undo changes to ensure changes are not undone
- Notice: relative ordering of A, B, C may vary!

ARIES Recovery

Recovery from a system crash is done in 3 passes:

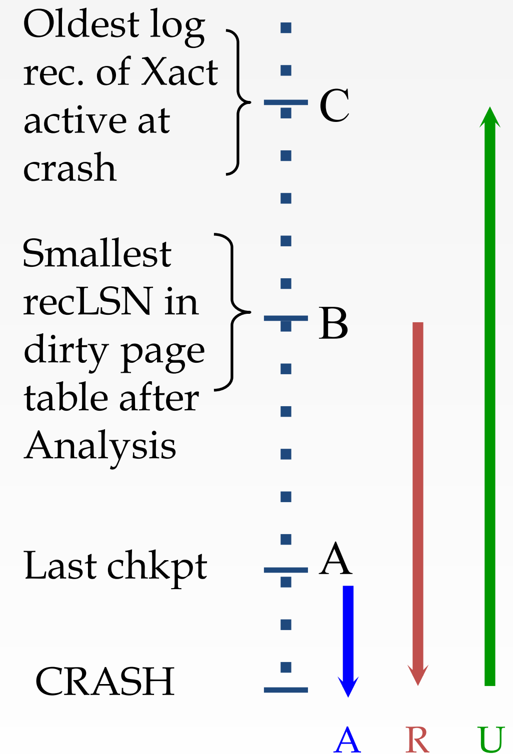
1. **Analysis pass**
   - Recreate list of dirty pages and active transactions
2. **Redo pass**
   - Redo all operations, even for those that were incomplete before crash
   - Goal is to replay DB to the state at the moment of the crash
3. **Undo pass**
   - Unroll effects of all incomplete transactions at time of crash
   - Log changes during undo in case of another crash during undo

Analysis Phase

- Goal
  - Determine point in log (**firstLSN**) where to start REDO
  - Determine set of dirty pages when crashed
  - Identify active transactions when crashed
- Approach
  - Rebuild transactions table and dirty pages table
  - Recover these from the last checkpoint in the log
  - Compute: **firstLSN** = smallest of all pages’ **recoveryLSN**
    - This is the earliest point that a write was made to buffer pool that hasn’t persisted yet

Recovery: The Analysis Phase

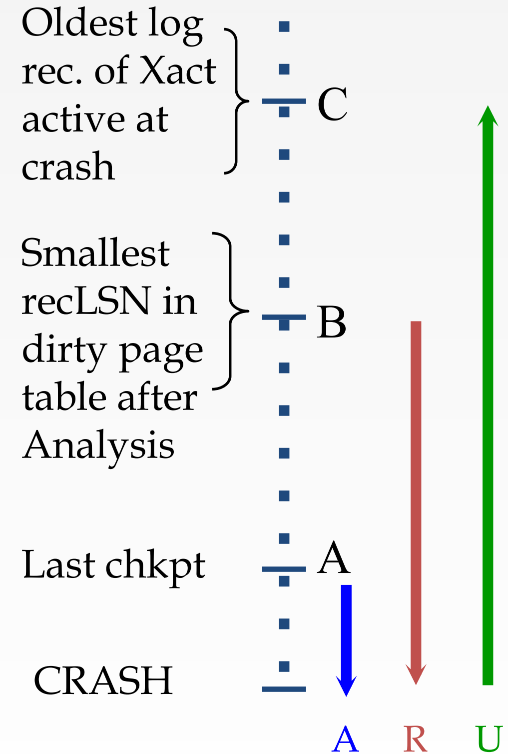
- Re-establish knowledge of state at checkpoint.
  - via transaction table and dirty page table stored in the checkpoint



Recovery: The Analysis Phase

- Scan log forward from checkpoint.
  - End record: Remove Xact from Xact table.
  - All Other records:
    - Add Xact to Xact table
    - set lastLSN=LSN,
    - on commit, change Xact status to 'C'
    - otherwise, with status 'U' (=candidate for undo)
  - also, for Update records: If page P not in Dirty Page Table (DPT),
    - add P to DPT, set its recLSN=LSN.

firstLSN = smallest of all pages'
recoveryLSN = oldest change



VIRGINIA TECH

Recovery: The Analysis Phase

- At end of Analysis:
  - transaction table says which xacts were active at time of crash.
  - DPT says which dirty pages ***might not*** have made it to disk

VIRGINIA TECH™

Example: Analysis Phase

LSN	LOG
00	begin_checkpoint
10	end_checkpoint
20	update: T1 writes P5
30	update: T2 writes P3
40	T2 commit
50	T2 end
60	update: T3 writes P3
70	T1 abort
	CRASH, RESTART

Dirty Page Table

PageID	recLSN

Txn Table

TxID	LastLSN	Status

Example: Analysis Phase

LSN	LOG
00	begin_checkpoint
10	end_checkpoint
20	update: T1 writes P5
30	update: T2 writes P3
40	T2 commit
50	T2 end
60	update: T3 writes P3
70	T1 abort
X	CRASH, RESTART

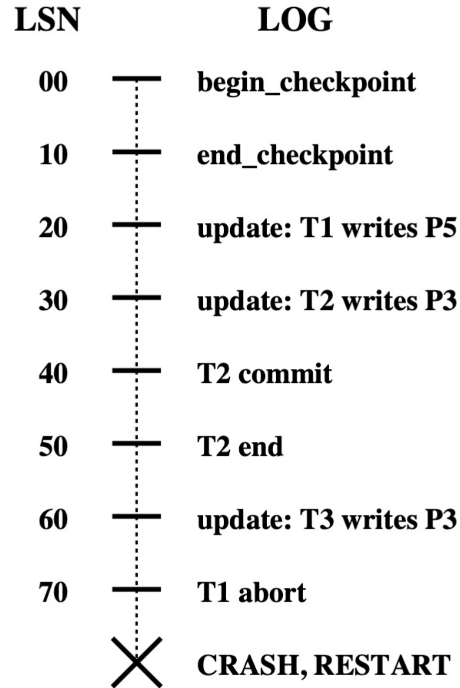
Dirty Page Table

PageID	recLSN
P5	20

Txn Table

TxID	LastLSN	Status
T1	20	U

Example: Analysis Phase



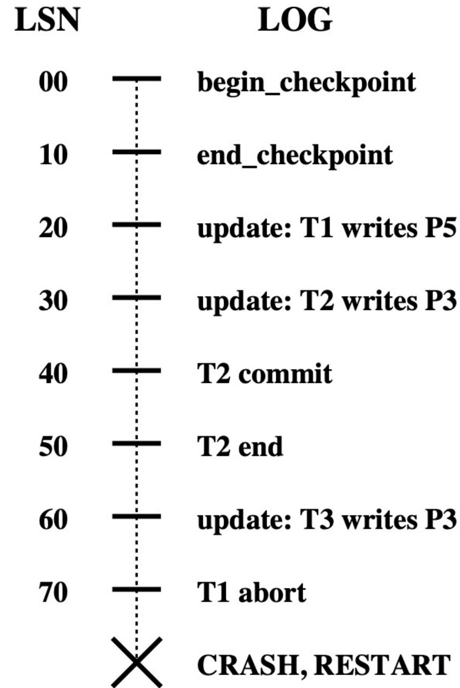
Dirty Page Table

PageID	recLSN
P5	20
P3	30

Txn Table

TxID	LastLSN	Status
T1	20	U
T2	30	U

Example: Analysis Phase



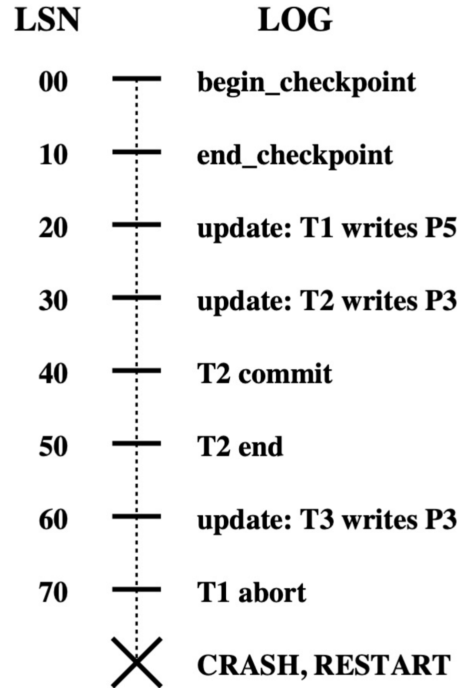
Dirty Page Table

PageID	recLSN
P5	20
P3	30

Txn Table

TxID	LastLSN	Status
T1	20	U
T2	30	C

Example: Analysis Phase



Dirty Page Table

PageID	recLSN
P5	20
P3	30

Txn Table

TxID	LastLSN	Status
T1	20	U
~~T2~~	~~30~~	~~C~~

Example: Analysis Phase

LSN	LOG
00	begin_checkpoint
10	end_checkpoint
20	update: T1 writes P5
30	update: T2 writes P3
40	T2 commit
50	T2 end
60	update: T3 writes P3
70	T1 abort
X	CRASH, RESTART

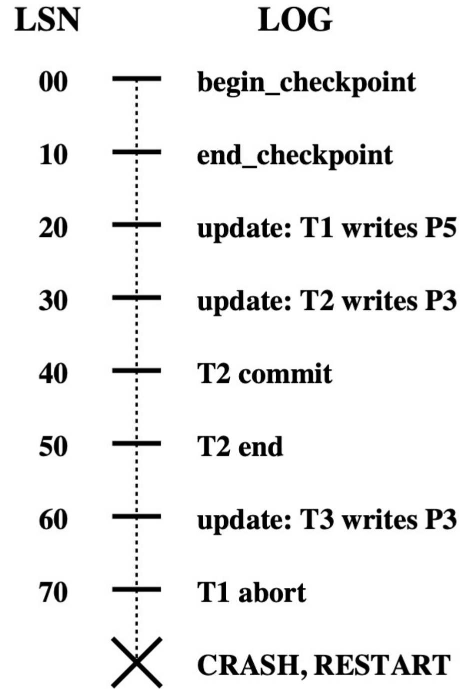
Dirty Page Table

PageID	recLSN
P5	20
P3	30

Txn Table

TxID	LastLSN	Status
T1	20	U
T3	60	U

Example: Analysis Phase



Dirty Page Table

PageID	recLSN
P5	20
P3	30

Txn Table

TxID	LastLSN	Status
T1	70	U
T3	60	U

Phase 2: REDO

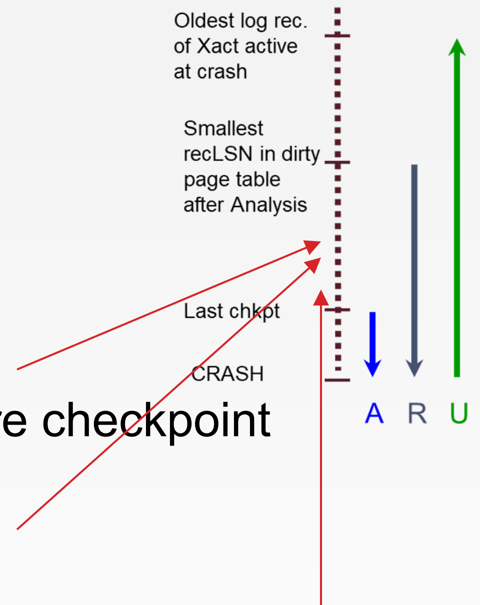
- Goal: *repeat History* to reconstruct state at crash:
  - Reapply *all* updates (even of aborted Xacts!), redo CLRs.
  - (and try to avoid unnecessary reads and writes!)
- Scan forward from log rec containing **smallest recLSN** in DPT
- For each update log record or CLR with a given LSN, **REDO** the action **unless**:
  - Affected page is not in the Dirty Page Table, or
  - Affected page is in D.P.T., but has recLSN > LSN, or
  - pageLSN (in DB) ≥ LSN. (this last case requires I/O)

Phase 2: REDO (cont'd)

- 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 xacts with status 'C',
- and remove them from transaction table

- What happens if system crashes during REDO ?
  - We REDO again! Each REDO operation is idempotent: doing it twice is the as doing it once

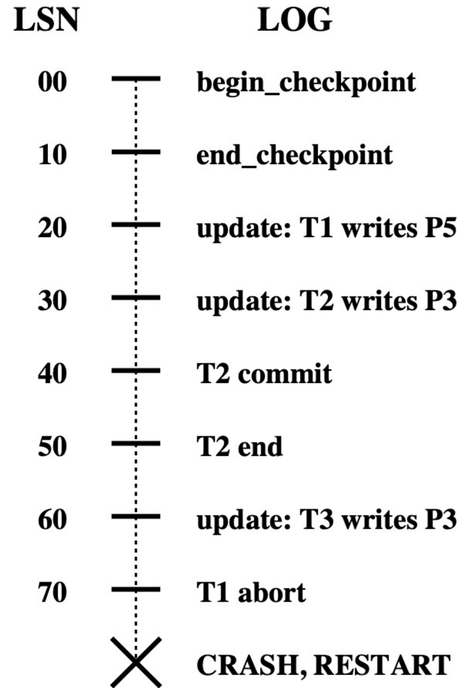
Scenarios When We Do Not REDO



Given an update log record…

- Affected page is not in the Dirty Page Table
  - This page was flushed to DB, removed from DPT before checkpoint
  - *Then* DPT flushed to checkpoint
- Affected page is in DPT, but has DPT recLSN > LSN
  - This page was flushed to DB, removed from DPT before checkpoint
  - *Then* this page was referenced *again* and reinserted in DPT with larger recLSN
- pageLSN (in DB) >= LSN. (this last case requires DB I/O)
  - This page was updated again and flushed to DB after this log record

Example: Redo Phase



Dirty Page Table

PageID	recLSN
P5	20 ⟸ start
P3	30

Txn Table

TxID	LastLSN	Status
T1	70	U
T3	60	U

Phase 3: UNDO

Goal: Undo all transactions that were active at the time of crash (‘loser xacts’)

- That is, all xacts with ‘U’ status on the xact table of the Analysis phase
- Process them in reverse LSN order
  - using the lastLSN’s to speed up traversal
  - and issuing CLRs

Phase 3: UNDO

ToUndo={lastLSNs of ‘loser’ Xacts}

Repeat:

- Choose (and remove) 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.

Phase 3: UNDO

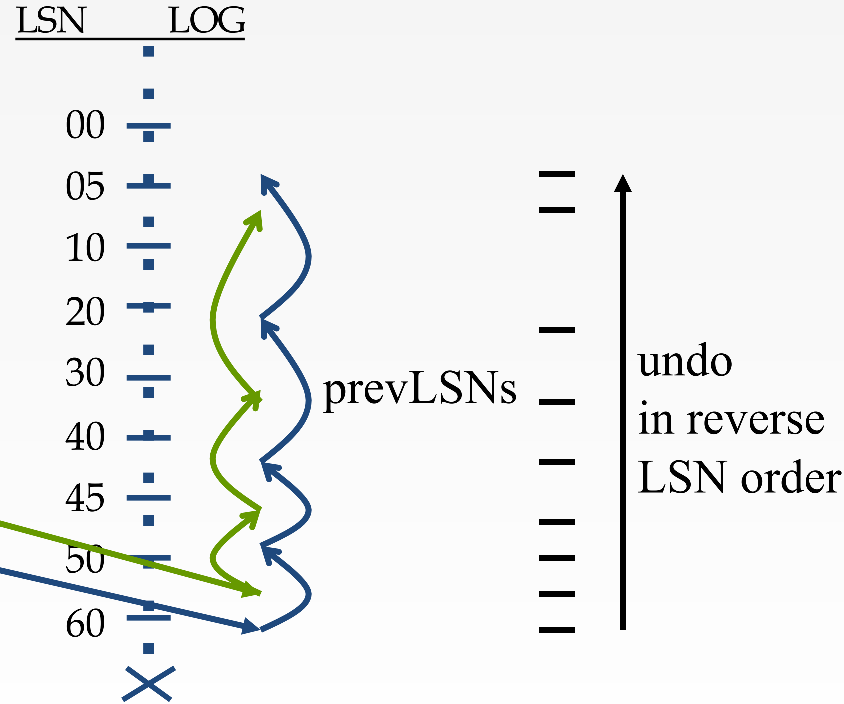
Q: What happens if system crashes during UNDO?

A: We do not UNDO again! Instead, each CLR is a REDO record: we simply redo the undo

Phase 3: UNDO - illustration

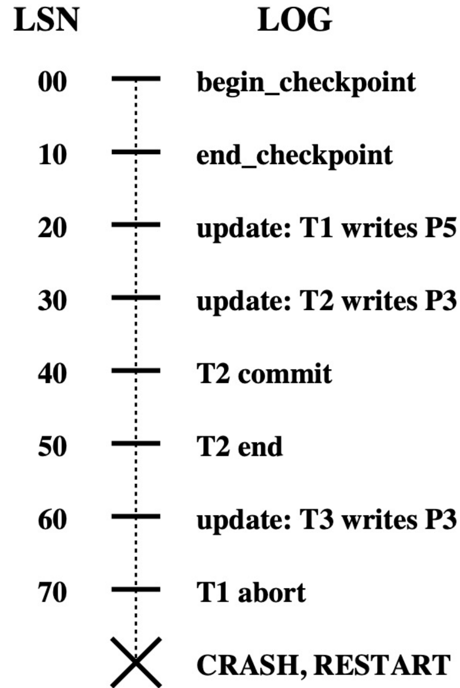
suppose that after end of analysis phase we have:
xact table

xid	status	lastLSN
T32	U	
T41	U	



VIRGINIA TECH

Example: Undo Phase



Dirty Page Table

PageID	recLSN
P5	20
P3	30

Txn Table

TxID	LastLSN	Status
T1	70	U
T3	60	U

← start

Example: Undo Phase

LSN	LOG
00	begin_checkpoint
10	end_checkpoint
20	update: T1 writes P5
30	update: T2 writes P3
40	T2 commit
50	T2 end
60	update: T3 writes P3
70	T1 abort
	CRASH, RESTART

Txn Table

TxID	LastLSN	Status
T1	70	U
T3	60	U

← start

ToUndo={70, 60}
LSN 70, ToUndo={60, 20}
LSN 60, undo change on P3 and adds a CLR, ToUndo={20}
LSN 20, undo change on P3 and adds a CLR

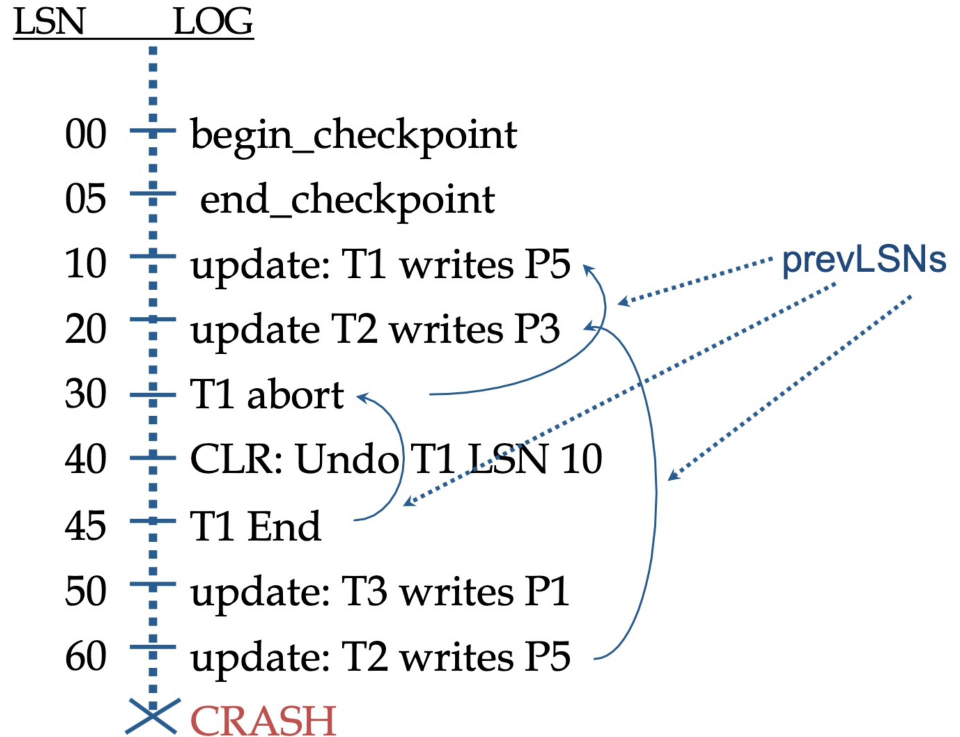
Example of Recovery

Xact Table
- lastLSN
- status

Dirty Page Table
- recLSN

flushedLSN

ToUndo



Questions

- Q1: After the Analysis phase, which are the ‘loser’ transactions?
- Q2: UNDO phase - what will it do?

Questions

- Q1: After the Analysis phase, which are the ‘loser’ transactions?
- A1: T2 and T3
- Q2: UNDO phase - what will it do?
- A2: undo ops of LSN 60, 50, 20

Example: Crash During Restart!

Xact Table
- lastLSN
- status

Dirty Page Table
- recLSN

flushedLSN

ToUndo

LSN	LOG
00,05	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
	CRASH, RESTART

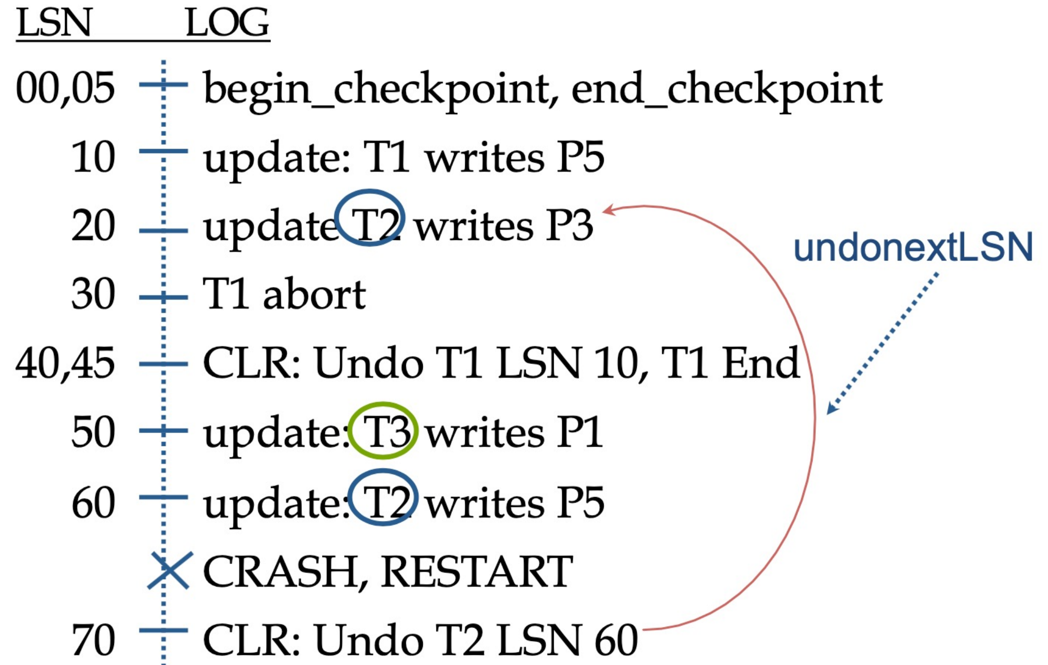
Example: Crash During Restart!

Xact Table
- lastLSN
- status

Dirty Page Table
- recLSN

flushedLSN

ToUndo



Example: Crash During Restart!

Xact Table
lastLSN
status
Dirty Page Table
recLSN
flushedLSN

ToUndo

LSN	LOG
00,05	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
	CRASH, RESTART
70	CLR: Undo T2 LSN 60
80,85	CLR: Undo T3 LSN 50, T3 end

undonextLSN

Example: Crash During Restart!

LSN	LOG
00,05	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
	CRASH, RESTART
70	CLR: Undo T2 LSN 60
80,85	CLR: Undo T3 LSN 50, T3 end
	CRASH, RESTART

undonextLSN

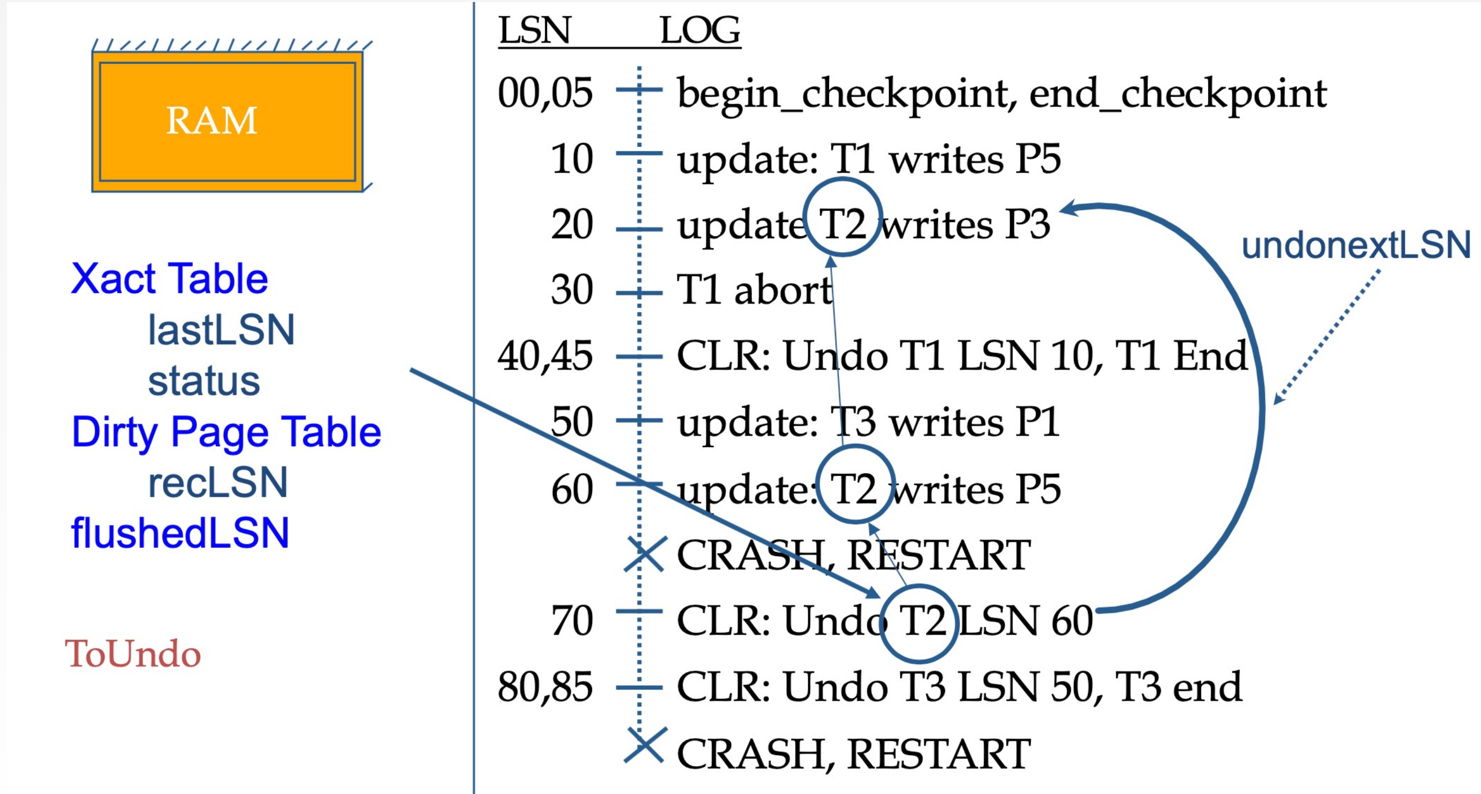
Questions

- Q3: After the Analysis phase, which are the ‘loser’ transactions?
- Q4: UNDO phase - what will it do?

Questions

- Q3: After the Analysis phase, which are the ‘loser’ transactions?
- A3: T2 only
- Q4: UNDO phase - what will it do?
- A4: follow the string of *prevLSN* of T2, exploiting *undoNextLSN*

Example: Crash During Restart!



Questions

- Q5: show the log, after the recovery is finished:

Example: Crash During Restart!

Xact Table
- lastLSN
- status

Dirty Page Table
- recLSN

flushedLSN

ToUndo

LSN	LOG
00,05	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
	CRASH, RESTART
70	CLR: Undo T2 LSN 60
80,85	CLR: Undo T3 LSN 50, T3 end
	CRASH, RESTART
90, 95	CLR: Undo T2 LSN 20, T2 end

undonextLSN

Additional Crash FAQs to Understand

Q: What happens if system crashes during Analysis?

A: Nothing serious. RAM state lost, need to start over next time.

Q: What happens if the system crashes during REDO?

A: Nothing bad. Some REDOs done, and we'll detect that next time.

Q: How do you limit the amount of work in REDO?

A: Flush asynchronously in the background. Even "hot" pages!

Q: How do you limit the amount of work in UNDO?

A: Avoid long-running Xacts.

Summary of Logging/Recovery

- Recovery Manager guarantees Atomicity & Durability
- Use WAL to allow STEAL/NO-FORCE w/o sacrificing correctness
- 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: undoNextLSN, recLSN, etc.

Summary of Logging/Recovery

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

(first two points) let OS do its best

(last two points) idempotency

VIRGINIA TECH™

Summary of Logging/Recovery

- **Checkpointing**: 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 (running, aborting) after Redo.
- Upon Undo, write CLRs.
- Redo “repeats history”: Simplifies the logic!

Reading and Next Class

- ARIES: Ch18
- Next: Data Mining and Warehousing