



Lecture 7: Recovery (Part 1)

Crash Recovery

"downs" "logging"

- Recovery algorithms are techniques to ensure database consistency, transaction atomicity, and durability despite failures
- Recovery algorithms have two parts:
 - ➔ Actions during normal txn processing to ensure that the DBMS can recover from a failure.
 - ➔ Actions after a failure to recover the database to a state that ensures atomicity, consistency, and durability.

error path

recovery

Availability

Logging Protocol

- Write-Ahead Logging is (almost) always the best approach to handle loss of volatile storage.
 - ▶ Use incremental updates (STEAL + NO-FORCE) with checkpoints.
 - ▶ On recovery: undo uncommitted txns + redo committed txns.

persistent memory

write-behind logging

Commit with uncommitted

*~~NO~~ WDO ?
NO RE-DO*

ARIES

System R

- Algorithms for Recovery and Isolation Exploiting Semantics
- Developed at IBM Research in early 1990s for the DB2 DBMS.
- Not all systems implement ARIES exactly as defined in this paper but they're close enough.

70 pages

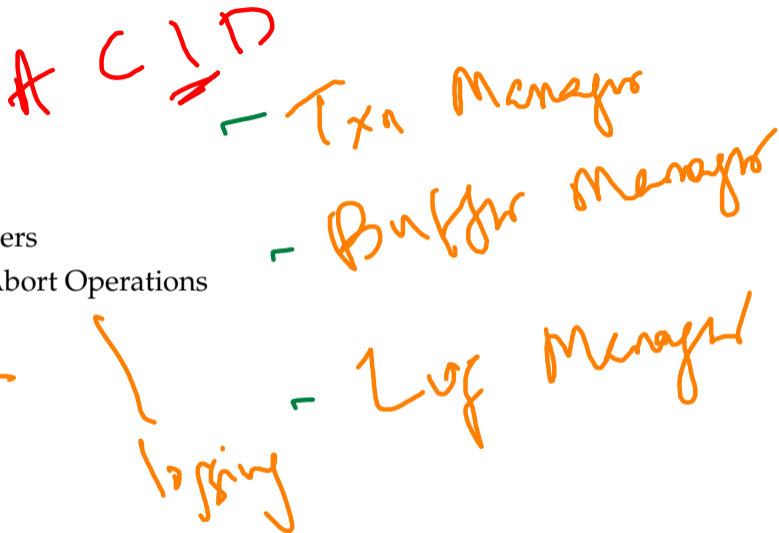
ARIES – Main Ideas

log → database

- Write-Ahead Logging:
 - ▶ Any change is recorded in log on stable storage before the change is written to database on disk.
 - ▶ Must use STEAL + NO-FORCE buffer pool policies.
- Repeating History During Redo:
 - ▶ On restart, retrace actions and restore database to exact state before crash.
- Logging Changes During Undo:
 - ▶ Record undo actions to log to ensure action is not repeated in the event of repeated failures.

A | R | U

Today's Agenda



- Log Sequence Numbers
- Normal Commit & Abort Operations
- Checkpointing
- Recovery Algorithm

Log Sequence Numbers

Log Records



- We need to extend our log record format from last class to include additional info.
- The log is a single ever-growing sequential file (append-only).
- Every log record now includes a globally unique log sequence number (LSN).
- Various components in the system keep track of LSNs that pertain to them...

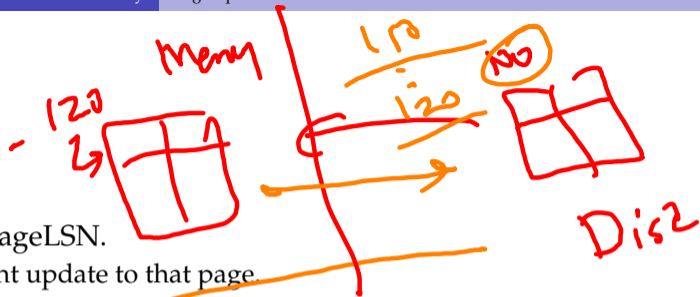
UNDO
REDO

(txn id, object
id,
~~before value~~
after value)

Log Sequence Numbers

<u>LSN Type</u>	<u>Where</u>	<u>Definition</u>
flushedLSN	Memory	Last LSN in log on disk
pageLSN	page _x	Newest update to page _x
recLSN	page _x	Oldest update to page _x since it was last flushed
lastLSN	T _i	Latest record of txn T _i
MasterRecord	Disk	LSN of latest checkpoint

Writing Log Records



- 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.
- Before page x can be written to disk, we must flush log at least to the point where:

$$\text{pageLSN}_x \leq \text{flushedLSN}$$

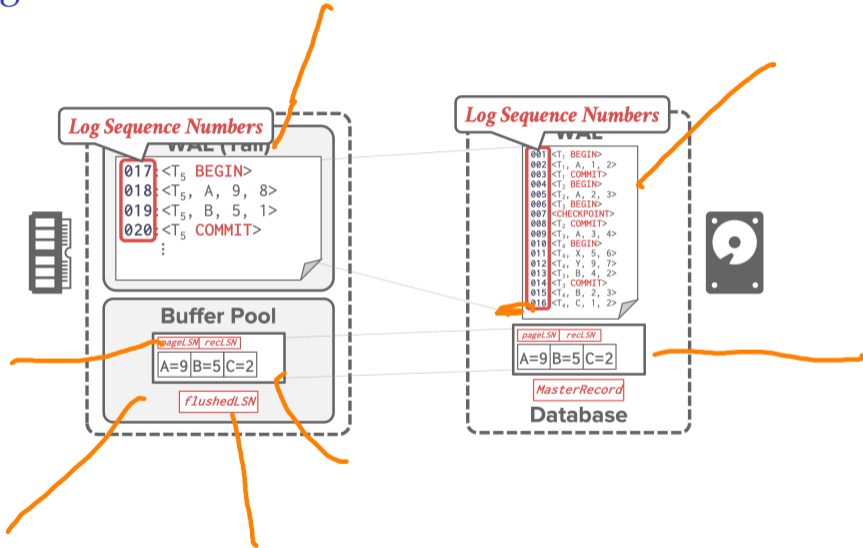
(100)

80

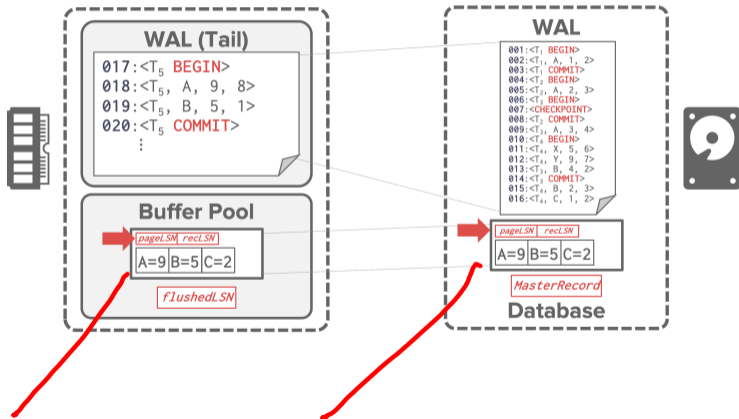
⋮

100

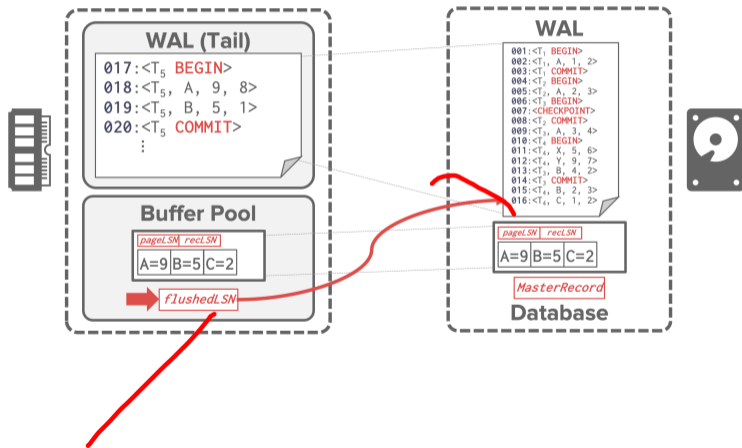
Writing Log Records



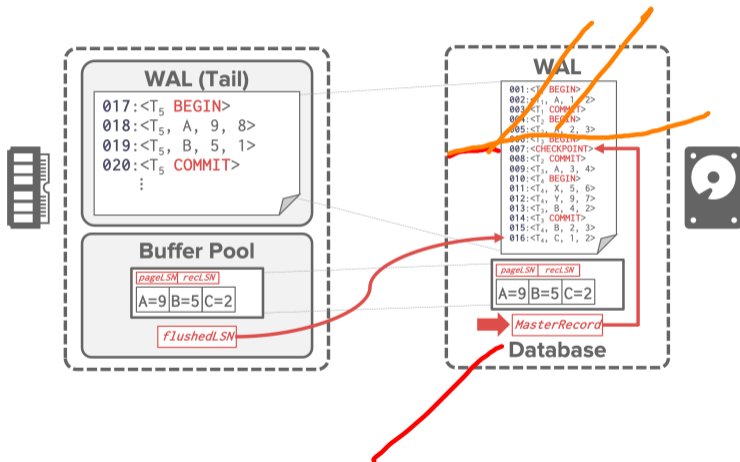
Writing Log Records



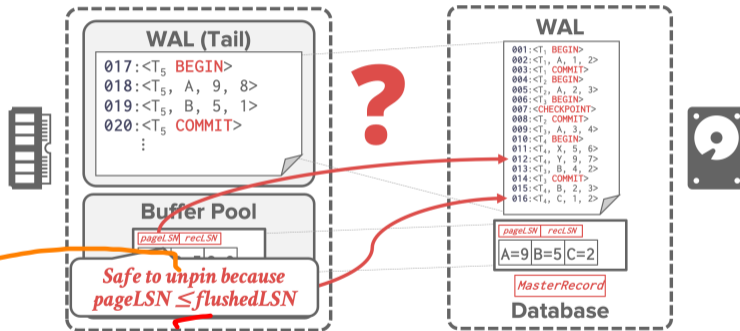
Writing Log Records



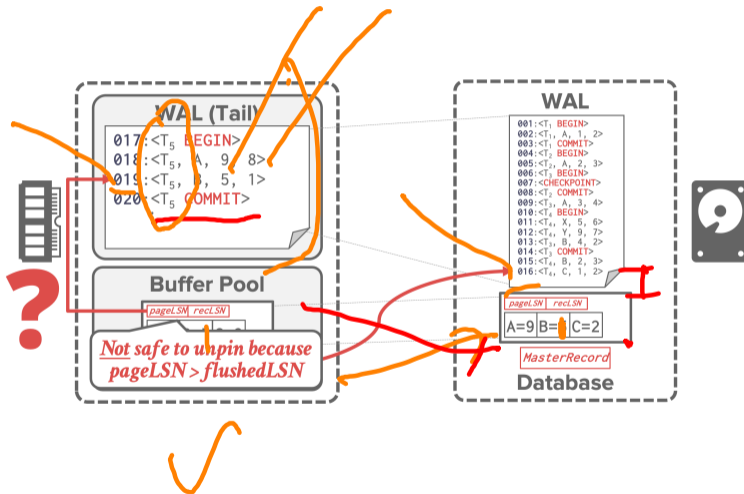
Writing Log Records



Writing Log Records



Writing Log Records



Writing Log Records

- All log records have an LSN.
- Update the pageLSN every time a txn modifies a record in the page.
- Update the flushedLSN in memory every time the DBMS writes out the WAL buffer to disk.
- Must generate the log record first before modifying the page

Normal Commit & Abort Operations

Normal Execution

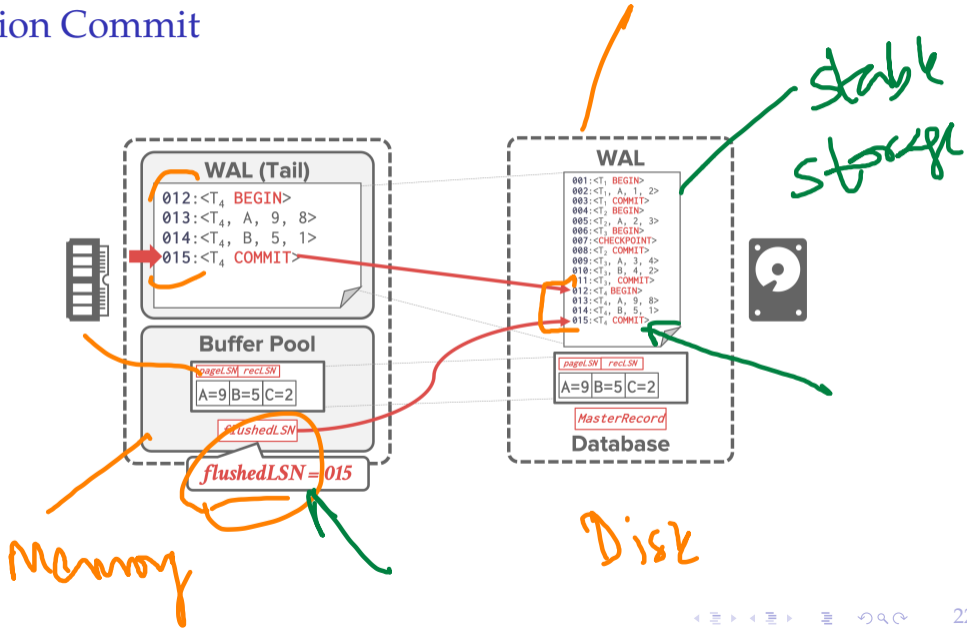
- Each txn invokes a sequence of reads and writes, followed by commit or abort.
- Assumptions in this lecture:
 - ▶ All log records fit within a single page.
 - ▶ Disk writes are atomic.
 - ▶ Single-versioned tuples with **Strict Two Phase Locking**.
 - ▶ ~~STEAL + NO-FORCE~~ buffer management with WAL.

CC
protocols

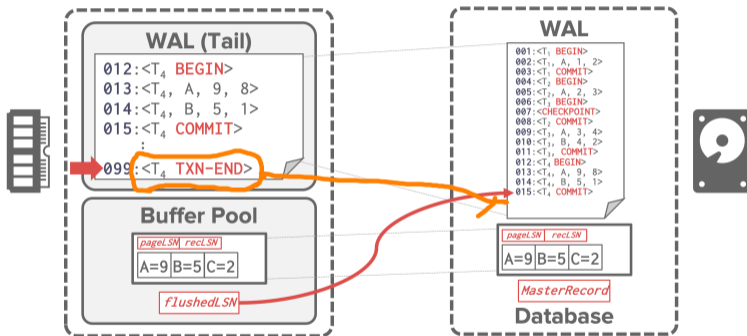
Transaction Commit

- Write <COMMIT> record to log.
- All log records up to txn's <COMMIT> record are flushed to disk.
 - ▶ Note that log flushes are sequential, synchronous writes to disk.
 - ▶ Many log records per log page.
- When the commit succeeds, write a special <TXN-END> record to log.
 - ▶ Now remove transaction from the Active Transaction Table
 - ▶ This does not need to be flushed immediately.

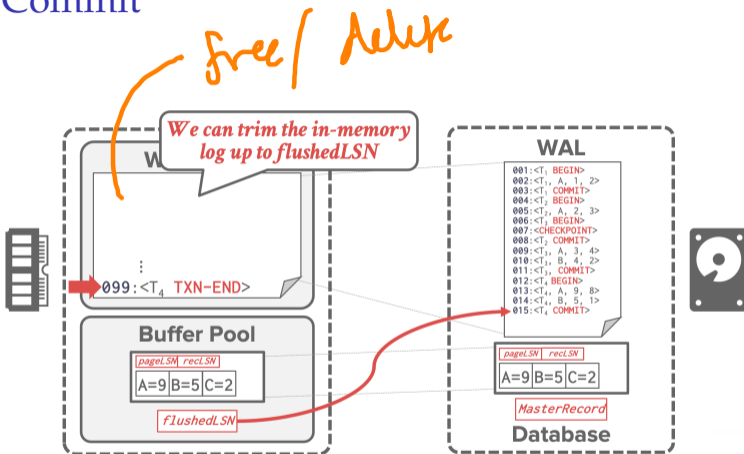
Transaction Commit



Transaction Commit



Transaction Commit

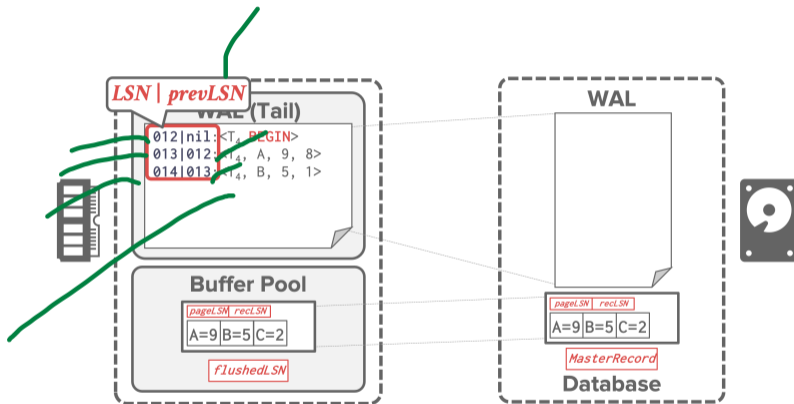


Transaction Abort

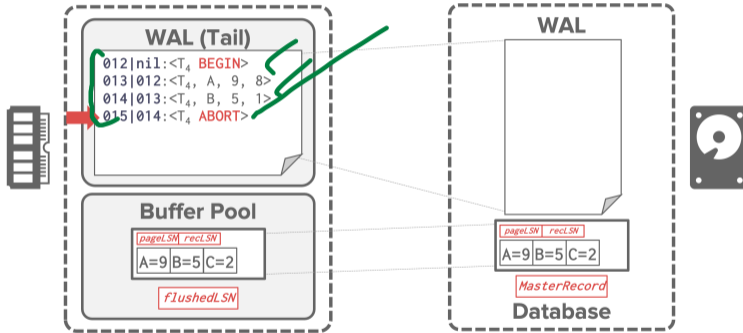
- Aborting a txn is actually a special case of the ARIES undo operation applied to only one transaction.
- We need to add another field to our log records:
 - prevLSN: The previous LSN for the txn.
 - ▶ This maintains a linked-list for each txn that makes it easy to walk through its records.



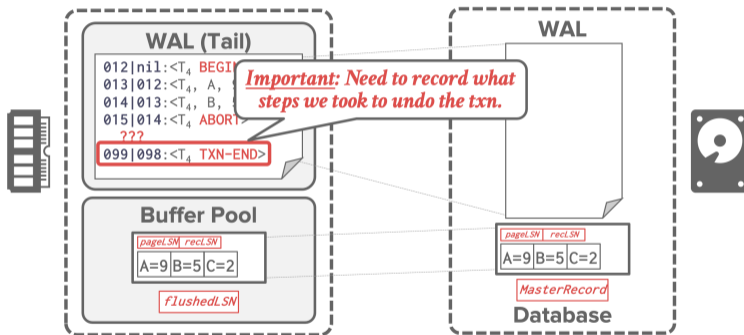
Transaction Abort



Transaction Abort



Transaction Abort



Compensation Log Records



Compensation Log Records



- A Compensation Log Record (CLR) describes the actions taken to undo the actions of a previous update record.
- It has all the fields of an update log record plus the undoNext pointer (the next-to-be-undone LSN).
- CLR's are added to log like any other record.
- Goal: CLR's are necessary to recover the database if there is a crash during recovery.

CLR Example

TIME

LSN	prevLSN	TxnId	Type	Object	Before	After	UndoNext
001	nil	T ₁	BEGIN	-	-	-	-
002	001	T ₁	UPDATE	A	30	40	-
011	002	T ₁	ABORT	-	-	-	-

CLR Example

LSN	prevLSN	TxnId	Type	Object	Before	After	UndoNext
001	nil	T ₁	BEGIN	-	-	-	-
002	001	T ₁	UPDATE	A	30	40	-
⋮							
011	002	T ₁	ABORT	-	-	-	-
⋮							
026	011	T ₁	CLR-002	A	40	30	001

CLR Example

LSN	prevLSN	TxnId	Type	Object	Before	After	UndoNext
001	nil	T ₁	BEGIN	-	-	-	-
002	001	T ₁	UPDATE	A	30	40	-
⋮							
011	002	T ₁	ABORT	-	-	-	-
⋮							
026	011	T ₁	CLR-002	A	40	30	001

CLR Example

LSN	prevLSN	TxnId	Type	Object	Before	After	UndoNext
001	nil	T ₁	BEGIN	-	-	-	-
002	001	T ₁	UPDATE	A	30	40	-
:							
011	002	T ₁	ABORT	-	-	-	-
:							
026	011	T ₁	CLR-002	A	40	30	001

The LSN of the next log record to be undone.

CLR Example

LSN	prevLSN	TxnId	Type	Object	Before	After	UndoNext
001	nil	T ₁	BEGIN	-	-	-	-
002	001	T ₁	UPDATE	A	30	40	-
:							
011	002	T ₁	ABORT	-	-	-	-
:							
026	011	T ₁	CLR-002	A	40	30	001
027	026	T ₁	TXN-END	-	-	-	nil

Abort Algorithm

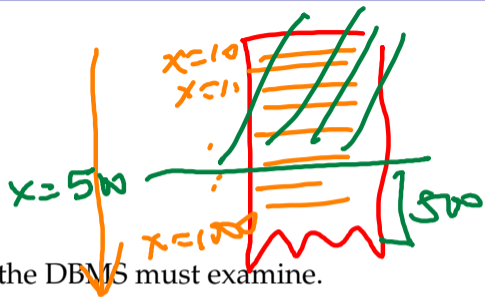
- First write an <ABORT> record to log for the txn.
- Then play back the txn's updates in reverse order. For each update record:
 - ▶ Write a CLR entry to the log.
 - ▶ Restore old value.
- When a txn aborts, we immediately tell the application that it is aborted.
- We don't need to wait to flush the CLR's
- At end, write a <TXN-END> log record.
- Notice: CLR's never need to be undone.

Checkpointing

Checkpointing

Snapshotting

- Log grows forever.
- Use checkpoints to limit the size of the log that the DBMS must examine.
- Checkpoint algorithms
 - Non-Fuzzy Checkpointing
 - Slight Better Checkpointing
 - Fuzzy Checkpointing



$x=10 \mid n \mid n \mid \dots \mid 1000$

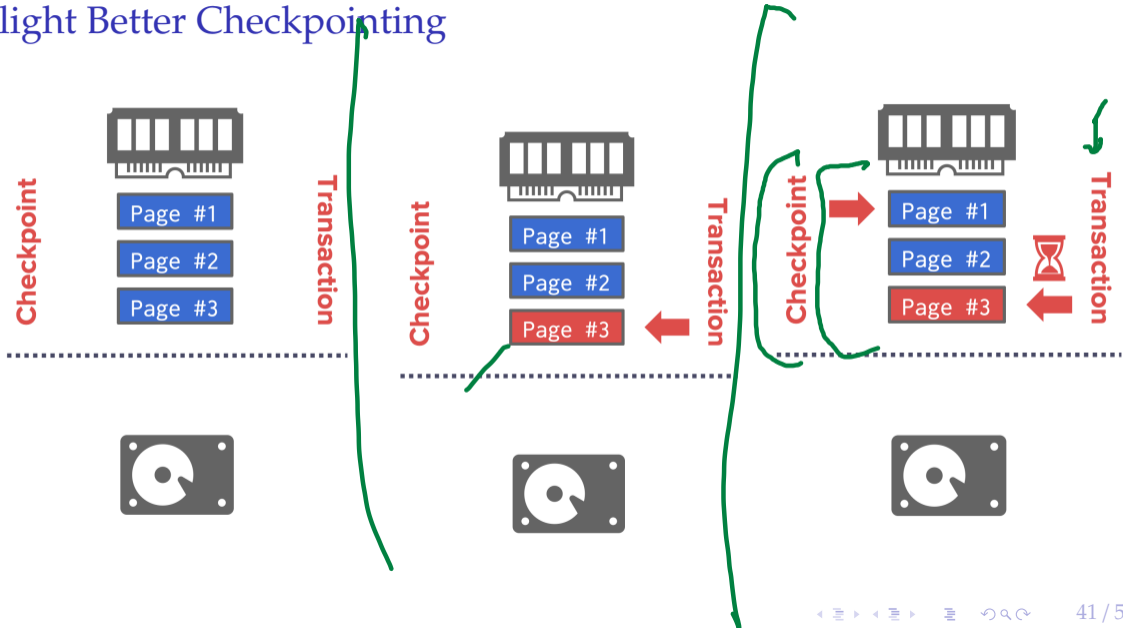
Non-Fuzzy Checkpointing

- The DBMS halts everything when it takes a checkpoint to ensure a consistent snapshot:
 - ▶ Halt the start of any new txns.
 - ▶ Wait until all active txns finish executing.
 - ▶ Flushes dirty pages on disk.
- This is obviously bad. . .

Slight Better Checkpointing

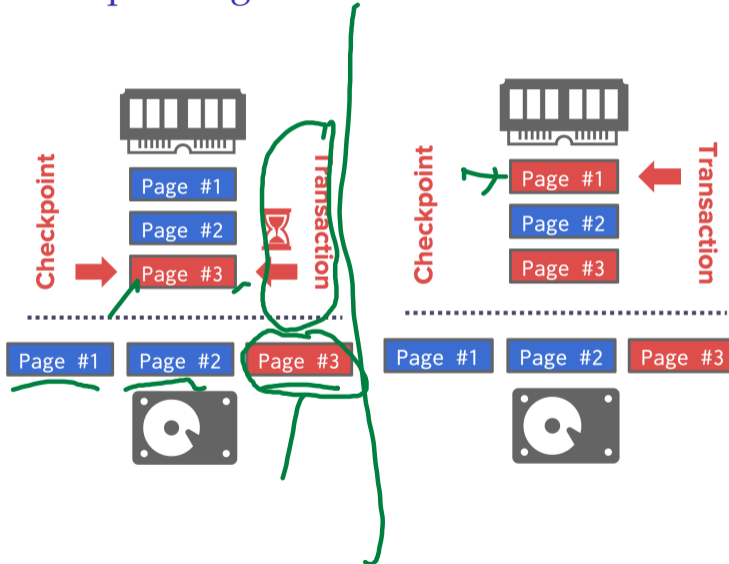
- Pause modifying txns while the DBMS takes the checkpoint.
 - ▶ Prevent queries from acquiring write latch on table/index pages.
 - ▶ Don't have to wait until all txns finish before taking the checkpoint.
- We must record internal state as of the beginning of the checkpoint.
 - ▶ Active Transaction Table (ATT)
 - ▶ Dirty Page Table (DPT)

Slight Better Checkpointing



Slight Better Checkpointing

Loss



Active Transaction Table *ATT*

- Managed by the Transaction Manager in memory
- One entry per currently active txn.
 - ▶ txnId: Unique txn identifier.
 - ▶ status: The current "mode" of the txn.
 - ▶ lastLSN: Most recent LSN created by txn.
- Entry removed when txn commits or aborts.
- Txn Status Codes:
 - ▶ R → Running
 - ▶ C → Committing
 - ▶ U → Candidate for Undo

// TXN-END

Dirty Page Table

DP

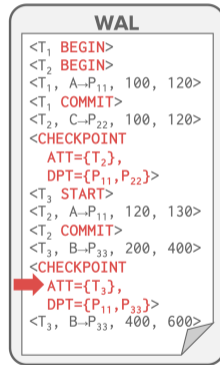
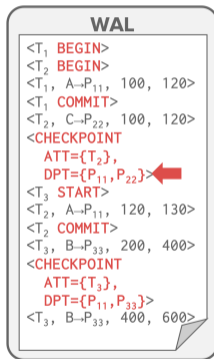
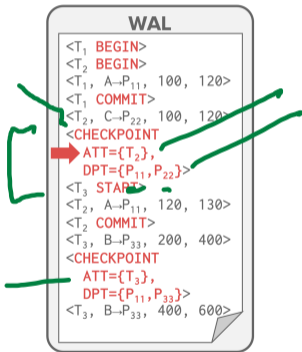
- Keep track of which pages in the buffer pool contain changes from uncommitted transactions.
- One entry per dirty page in the buffer pool:
 - ▶ recLSN: The LSN of the log record that first caused the page to be dirty.



Slight Better Checkpointing

- At the first checkpoint, T2 is still running and there are two dirty pages (P11, P22).
- At the second checkpoint, T3 is active and there are two dirty pages (P11, P33).
- This still is not ideal because the DBMS must stall txns during checkpoint...

Slight Better Checkpointing



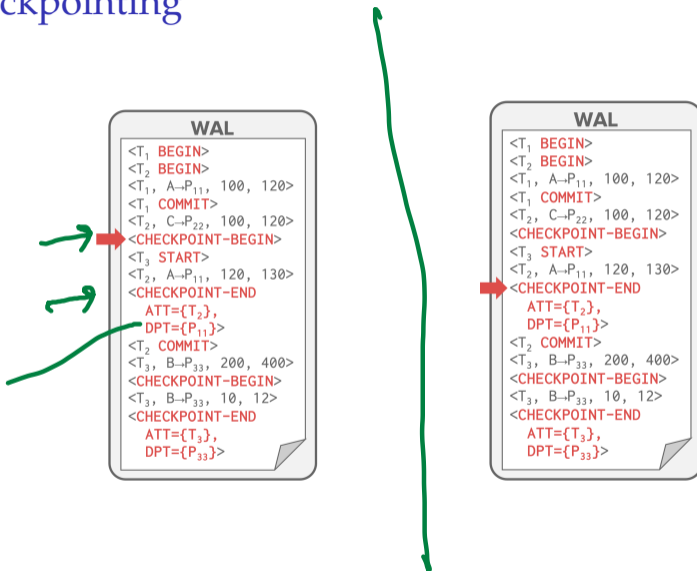
Fuzzy Checkpointing

- A fuzzy checkpoint is where the DBMS allows active txns to continue running while the system flushes dirty pages to disk.
- New types of log records to track checkpoint boundaries:
 - CHECKPOINT – BEGIN: Indicates start of checkpoint
 - CHECKPOINT – END: Contains ATT + DPT.

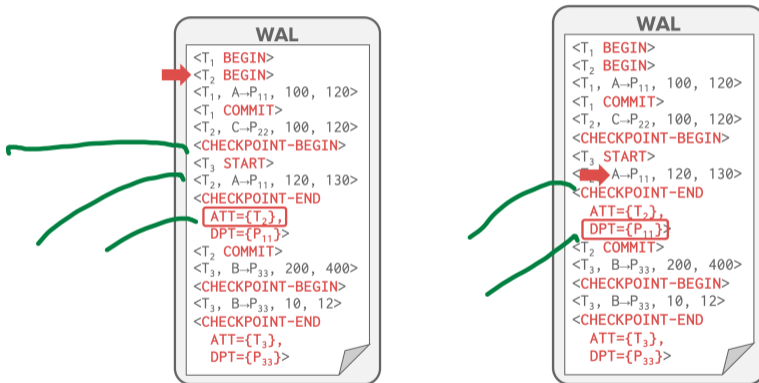
Fuzzy Checkpointing

- The LSN of the <CHECKPOINT-BEGIN> record is written to the database's MasterRecord entry on disk when the checkpoint successfully completes.
- Any txn that starts after the checkpoint is excluded from the ATT in the <CHECKPOINT-END> record.

Fuzzy Checkpointing



Fuzzy Checkpointing



Conclusion

Parting Thoughts

- Log Sequence Numbers:
 - ▶ LSNs identify log records; linked into backwards chains per transaction via prevLSN.
 - ▶ pageLSN allows comparison of data page and log records.
- Main ideas of ARIES:
 - ▶ WAL with STEAL/NO-FORCE
 - ▶ Fuzzy Checkpoints (snapshot of dirty page ids)
 - ▶ Write CLRs when undoing, to survive failures during restarts

Next Class

- Continue the ARIES protocol