Crash Recovery on DBMS: A Survey

Driven by Hardware Revolution

Ethan Zhu

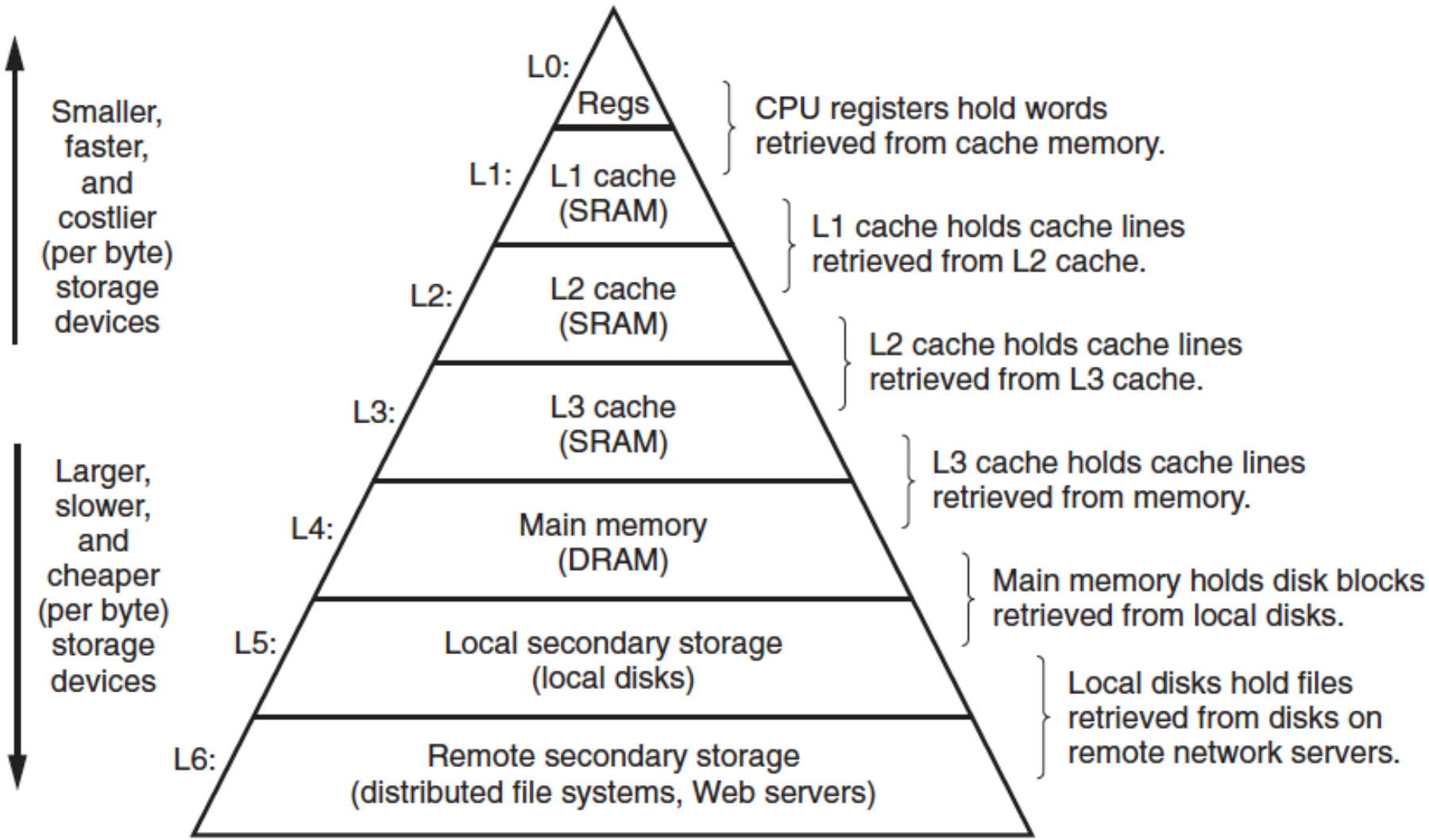
Department of Computer Science and Technology
Nanjing University

Jan, 16, 2021

Table of Contents

Table of Contents

Storage hierarchy



Storage Technology

- DRAM
  - ✓ Byte addressable access granularity.
  - ✓ No difference between random and sequential access.
  - ✓ In-palace update.
  - ✓ No wear leveling.
- HDD
  - ✓ Block addressable access granularity.
  - ✓ Significant difference between random and sequential access.
  - ✓ In-place update.
  - ✓ No wear leveling.
- Flash SSD
  - ✓ Block addressable access granularity.
  - ✓ Minor difference between random and sequential access.
  - ✓ Illusion of in-place updates provided by FTL, which requires a separate erase operation to clear data before a write.
  - ✓ Garbage Collection, address mapping and wear leveling.

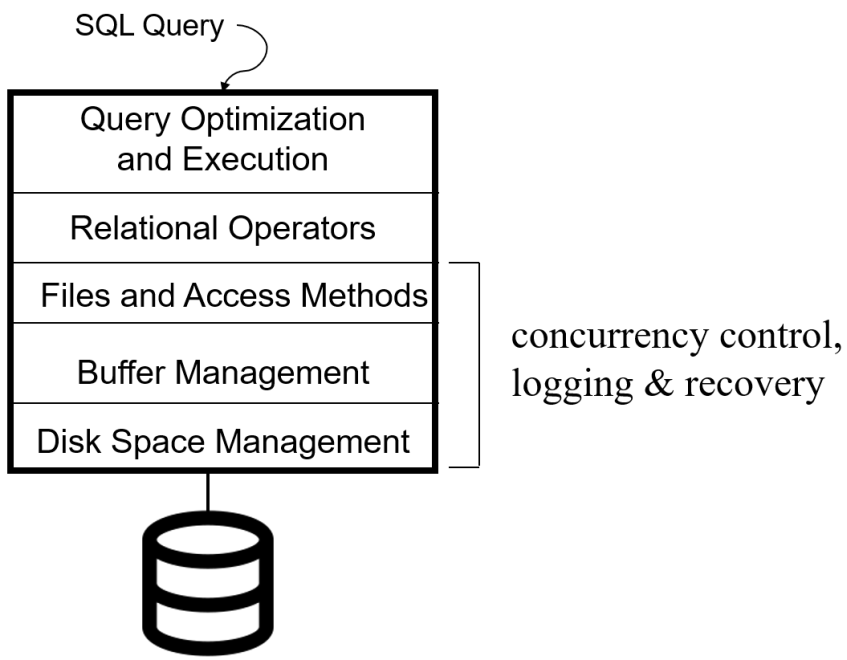
Storage Technology I

- NVRAM
  - ✓ Byte-addressable access granularity.
  - ✓ No difference between random and sequential access in theory, but minor difference in practise.
  - ✓ 8-byte atomic In-place update.
  - ✓ Cosuming very low power (energy-efficient) and providing very high density (cost-effective).
  - ✓ Wear leveling.
    - ∗ Write endurance is several orders of magnitude lower than DRAM.

	Access Granularity	Diff. of R/S Performance	In-place	Wear Leveling	Garbage Collection
DRAM	Byte	No	Yes	No	No
DISK	Block	Significant	Yes	No	No
FLASH SSD	Block	Minor	Illusion	Yes	Yes
NVMM	Byte	No (but minor in practice)	Yes	Yes	No

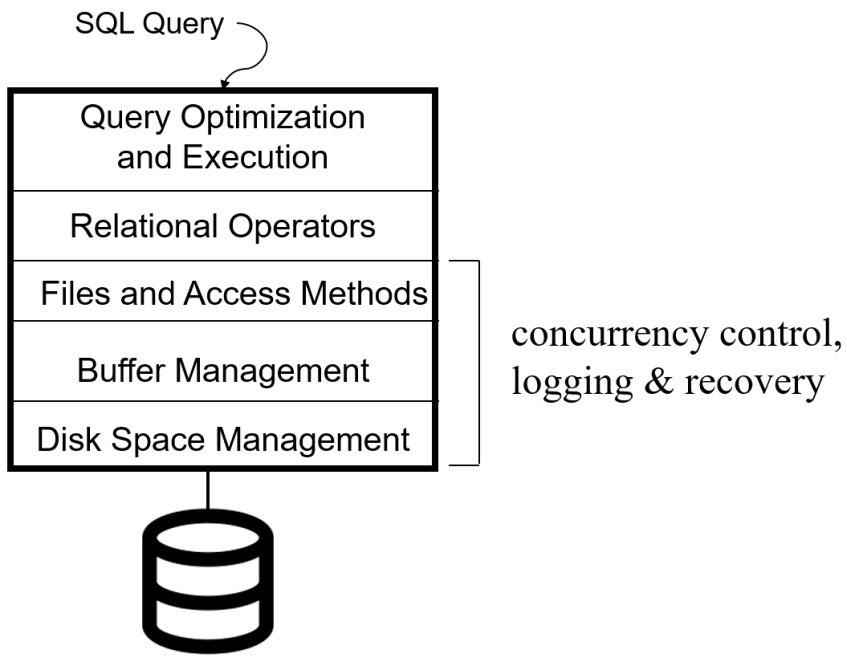
Architecture of RDBMS

- Query optimization and execution
- Relational operators
- Files and access methods
- Buffer pool management
- Disk space management



Architecture of RDBMS

- Query optimization and execution
- Relational operators
- Files and access methods
- Buffer pool management
- Disk space management



- Crash recovery is awfully difficult!
  - The recovery system depends on behavior of many other components of DBMS, such as concurrency control, buffer management, disk management, and query processing.

Buffer Pool Management

- Force policy —make sure that every update is on the DB disk before commit.
  - ✓ Provides durability without REDO logging.
  - ✓ But, can cause poor performance due to a large random write operations.

Buffer Pool Management

- Force policy —make sure that every update is on the DB disk before commit.
  - ✓ Provides durability without REDO logging.
  - ✓ But, can cause poor performance due to a large random write operations.
- No Steal policy —don't allow buffer pool frames with uncommited updates to overwrite committed data on DB disk.
  - ✓ Useful for ensuring atomicity without UNDO logging.
  - ✓ But can cause poor performance due to (1)A larger buffer is required; or (2)writing that data to temporary location on non-volatile storage (e.g., swap area).

Buffer Pool Management

- Force policy —make sure that every update is on the DB disk before commit.
  - ✓ Provides durability without REDO logging.
  - ✓ But, can cause poor performance due to a large random write operations.
- No Steal policy —don't allow buffer pool frames with uncommited updates to overwrite committed data on DB disk.
  - ✓ Useful for ensuring atomicity without UNDO logging.
  - ✓ But can cause poor performance due to (1)A larger buffer is required; or (2)writing that data to temporary location on non-volatile storage (e.g., swap area).

In practice, even to get Force/No-Steal to work requires some nasty details for handling unexpected failures...

Buffer Pool Replace Policy

- No Force
  - What if system crashes before a modified page written by a committed transaction makes it to DB disk?
    - ✓ Write as little as possible, in a convenient place, at commit time, to support REDOing modifications. → WAL Logging.

Buffer Pool Replace Policy

- No Force
  - What if system crashes before a modified page written by a committed transaction makes it to DB disk?
    - ✓ Write as little as possible, in a convenient place, at commit time, to support REDOing modifications. → WAL Logging.
- Steal
  - What if a transaction that performed updates aborts? → WAL Logging
  - What if system crashes before transaction is finished? → WAL Logging
    - ✓ Must remember the old value of P (to support UNDOing the write to page P).

Write Ahead Log Protocol

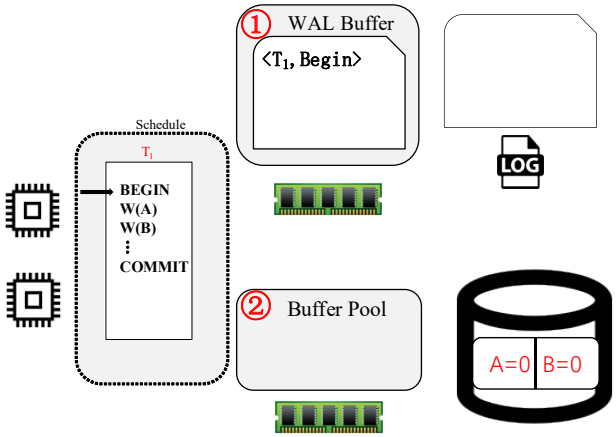
- The Write-Ahead Logging Protocol
  1. All log records pertaining to an updated page are written to non-volatile storage before the page itself is allowed to be over-written in non-volatile storage.
     - ✓ With UNDO info helps guarantee atomicity.
  2. A transaction is not considerd to be commited until all of its log records (including its commited record) have been written to stable storage.
     - ✓ With REDO info helps guarantee durability.
- This allows us to implement Steal/No-Force.

Write Ahead Log Protocol

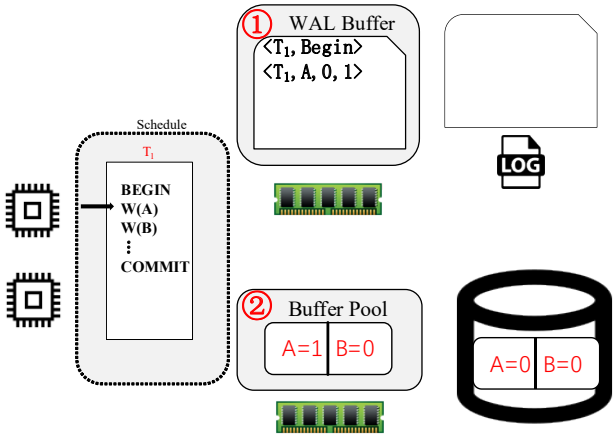
- The Write-Ahead Logging Protocol
  1. All log records pertaining to an updated page are written to non-volatile storage before the page itself is allowed to be over-written in non-volatile storage.
     - ✓ With UNDO info helps guarantee atomicity.
  2. A transaction is not considerd to be commited until all of its log records (including its commited record) have been written to stable storage.
     - ✓ With REDO info helps guarantee durability.
- This allows us to implement Steal/No-Force.

How do the systems implement logging?

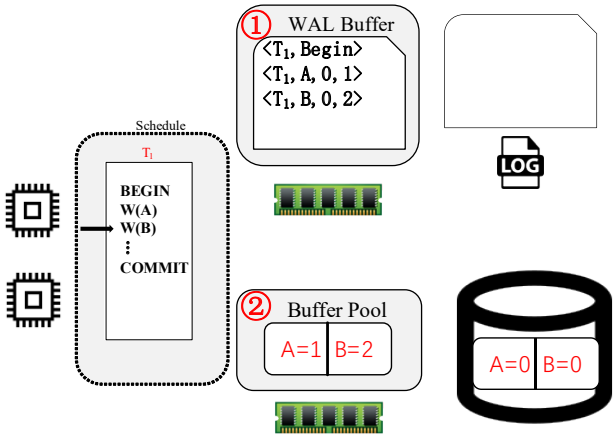
Illustration of WAL Protocol



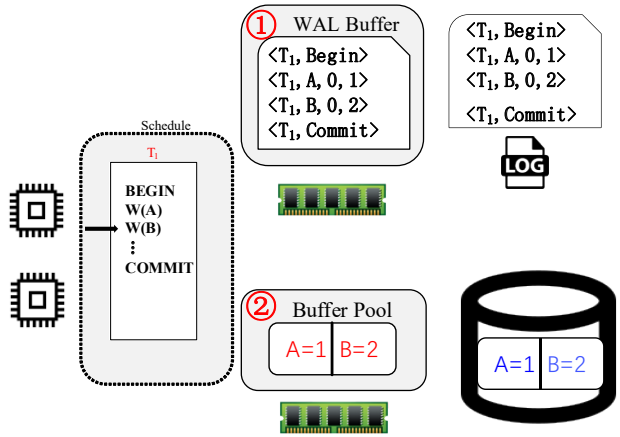
(1)



(2)

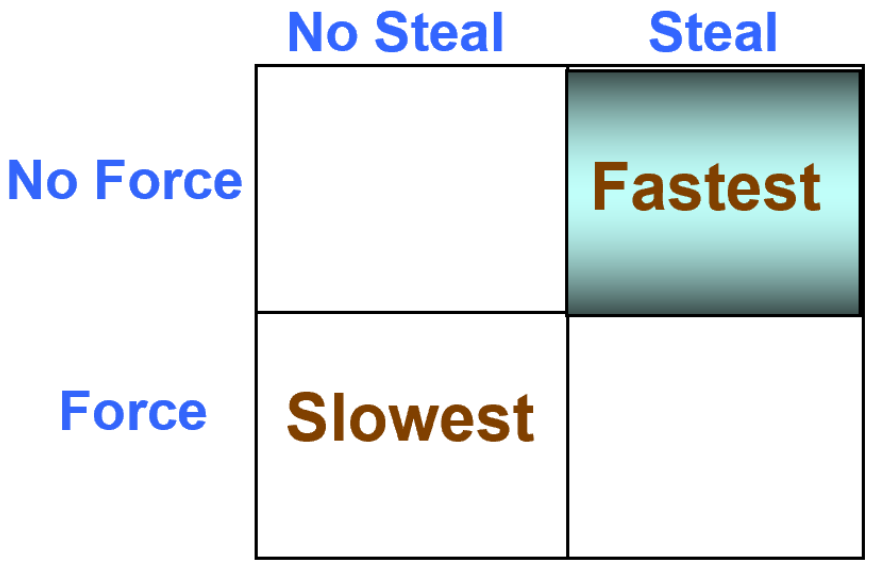


(3)

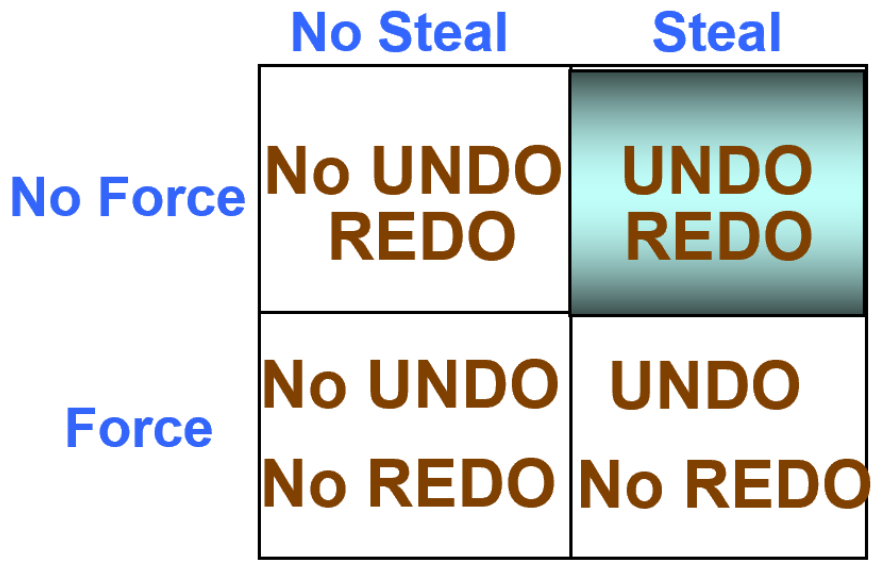


(4)

Buffer Pool Management



Performance Implications



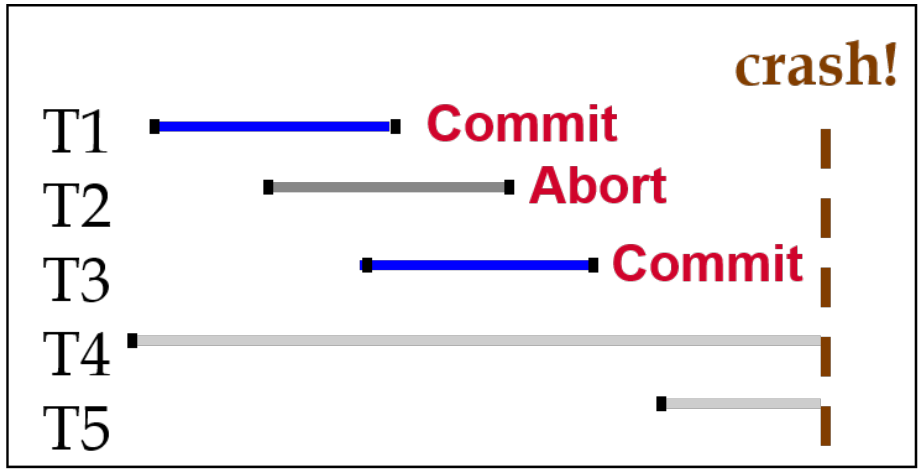
Log/Recovery Implications

Assumptions

- Concurrency control is in effect.
  - ✓ Strict 2PL.
  - ✓ Rigorous 2PL, in particular.
  - ✓ MVTO
  - ✓ MVOCC
  - ✓ MV2PL
- Updates are happening in-place. (a.k.a It's not multi-version!)
  - ✓ We will extend the recovery algorithm after ARIES.

Motivation of Crash Recovery

- T_1, T_3 should be durable.
- T_2, T_4, T_5 should be aborted (effects not seen).



Let's Try It Out!

- Can you think of a simple scheme (requiring no logging) to guarantee atomicity and durability?

Let's Try It Out!

- Can you think of a simple scheme (requiring no logging) to guarantee atomicity and durability?
  - ✓ Swapping File.

Let's Try It Out!

- Can you think of a simple scheme (requiring no logging) to guarantee atomicity and durability?
  - ✓ Swapping File.
    - Like vim editor save the swapping file.

Let's Try It Out!

- Can you think of a simple scheme (requiring no logging) to guarantee atomicity and durability?
  - ✓ Swapping File.
    - Like vim editor save the swapping file.
- There are 3 questions are considered:
  - What happens during normal execution?
  - What happens when a transaction commits?
  - What happens when a transaction aborts?

Let's Try It Out!

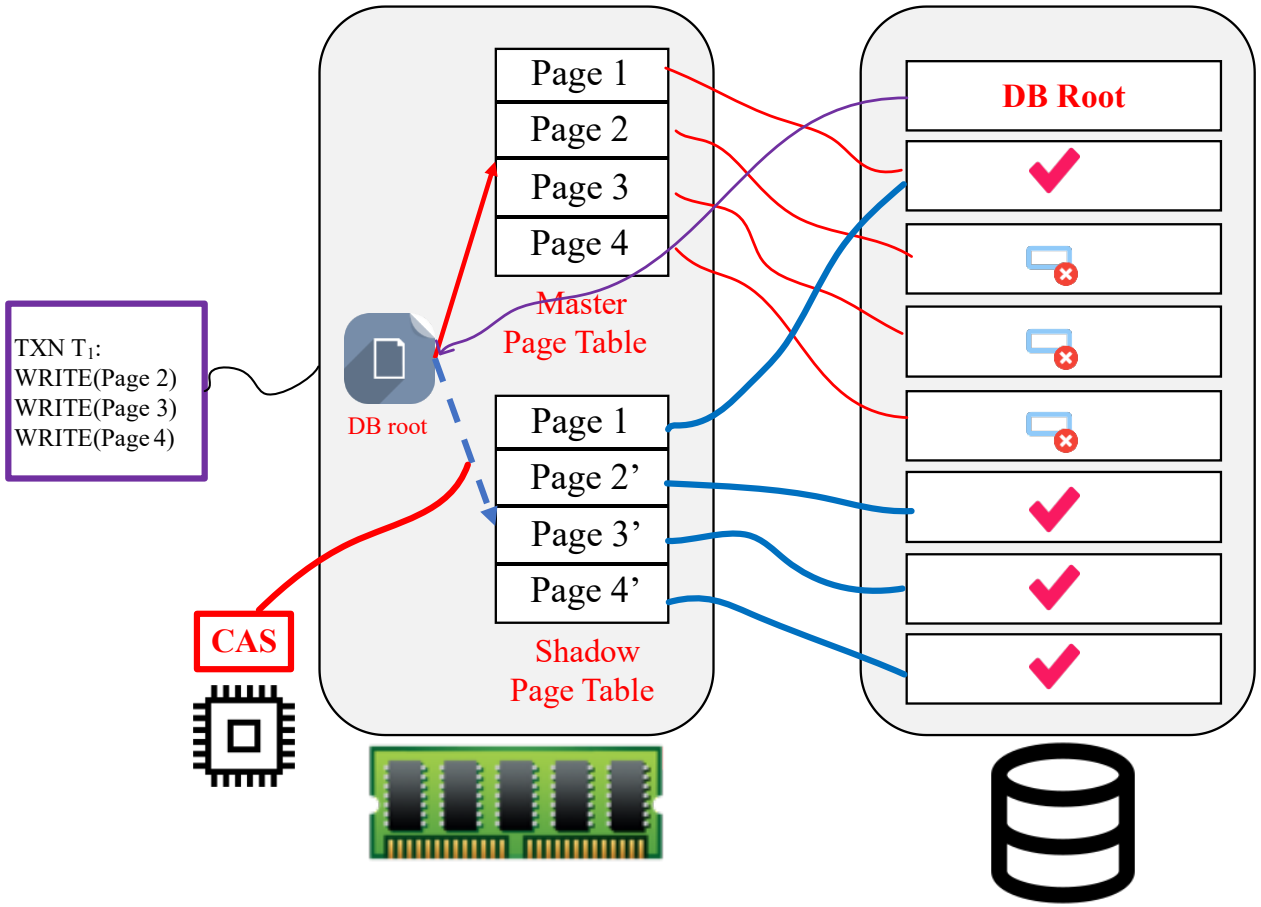
- Can you think of a simple scheme (requiring no logging) to guarantee atomicity and durability?
  - ✓ Swapping File.
    - Like vim editor save the swapping file.
- There are 3 questions are considered:
  - What happens during normal execution?
  - What happens when a transaction commits?
  - What happens when a transaction aborts?

Next...

Table of Contents

Table of Contents

Shadow paging



- Master: Contains only changes from committed txns.
- Shadow: Temporary db with changes made from uncommitted txns.

Advantages and Disadvantages of Shadow Paging

- Advantages
  - ✓ No undo, No redo.
- Disadvamtages
  - ∗ Copying the entire page table is expensive.
  - ∗ Commit overhead is high.
    - ✓ Flush every updated page, page table, and root.
    - ✓ Data gets fragmented so that it needs GC.
    - ✓ Only supports one writer txn at a time.

Advantages and Disadvantages of Shadow Paging

- Advantages
  - ✓ No undo, No redo.
- Disadvamtages
  - ∗ Copying the entire page table is expensive.
  - ∗ Commit overhead is high.
    - ✓ Flush every updated page, page table, and root.
    - ✓ Data gets fragmented so that it needs GC.
    - ✓ Only supports one writer txn at a time.
- How do we generalize Shadowing pages to support concurrent transactions?
  - ∗ If several transactions concurrently alter a file, file save or restore is inappropriate because it commits or aborts the updates of all transactions to the file.
  - ∗ It is desirable to be able to commit or undo updates on a per transaction basis.

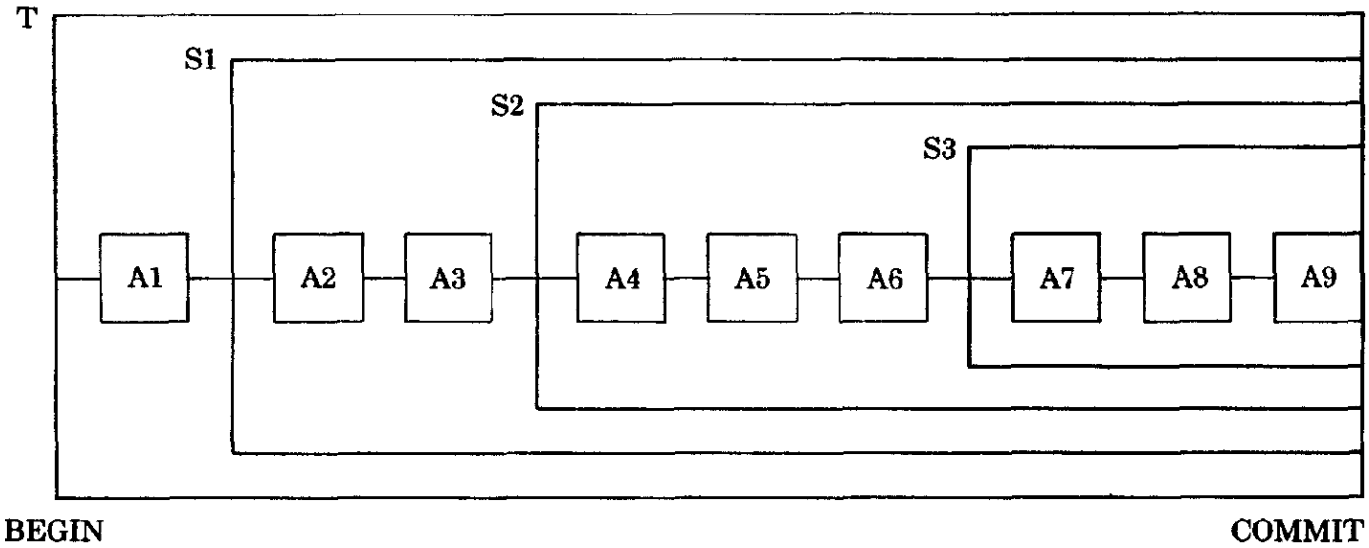
Table of Contents

Basic Ideas

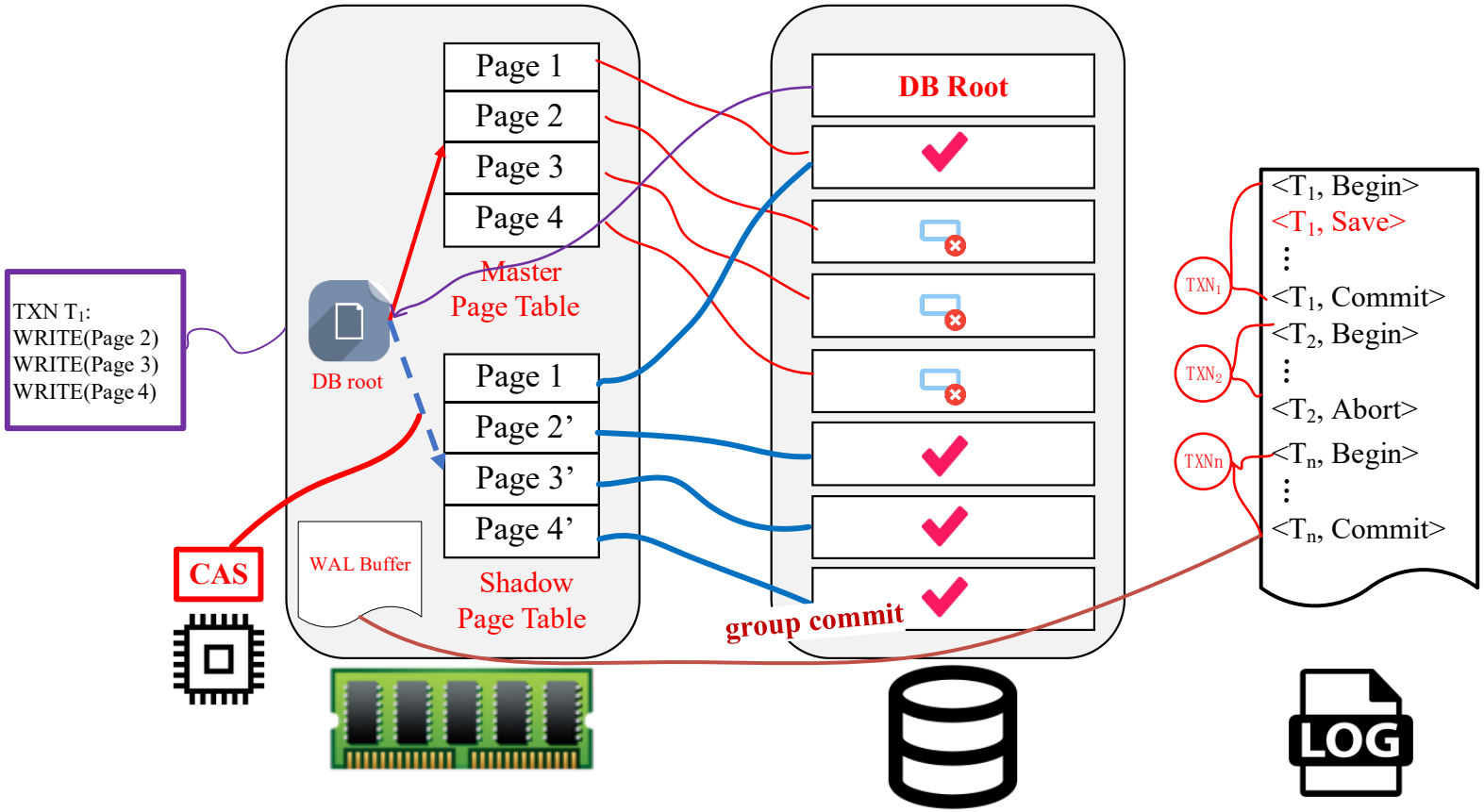
- Save point - Transactins includes BEGIN, SAVE, COMMIT, ABORT, and UODO commands. (why do it need save point? Operation error was the most common source of failure in 1980s!)
- WAL - Logs are organized by TXN id.
- Shadow paging (a.k.a CoW)
- Checkpoint

Basic Ideas

- Save point - Transactins includes BEGIN, SAVE, COMMIT, ABORT, and UODO commands. (why do it need save point? Operation error was the most common source of failure in 1980s!)
- WAL - Logs are organized by TXN id.
- Shadow paging (a.k.a CoW)
- Checkpoint



System R Models in 1981: WAL + CoW



Commit Processing

- The transaction log is written to disk before the current database is replaced by the shadowing database state. (WAL)
  1. The transaction commit action writes a commit log record in the log buffer
  2. Then force all the transaction's log records to disk(with the commit record being the last such record).
- Log records are aggregated by transaction and collected in a common system log file.

Checkpoint

- How do System R implement checkpoint?
  - A checkpoint saves all logged shadow files so that no work prior to the checkpoint will have to be redone at restart.
    - ✓ The simplist form of checkpoint is to record a transaction-consistent state by quiescing the system, which is not a cheap way.
    - ✓ Output onto stable storage all log records currently residing in main memory.
    - ✓ Output to the disk all modified pages.
- When do System R to take checkpoint?
  - Checkpoints are taken after a specified amount of log activity.
  - Checkpoints are taken after at system operator request.

Recovery Algorithms of System R

- Assumption: Given a checkpoint of the state at time T along with a log of all changes to the state made by transactions prior to time T + E.

✓ UNDOing all update prior to T+E transactions which were uncommited and abort.

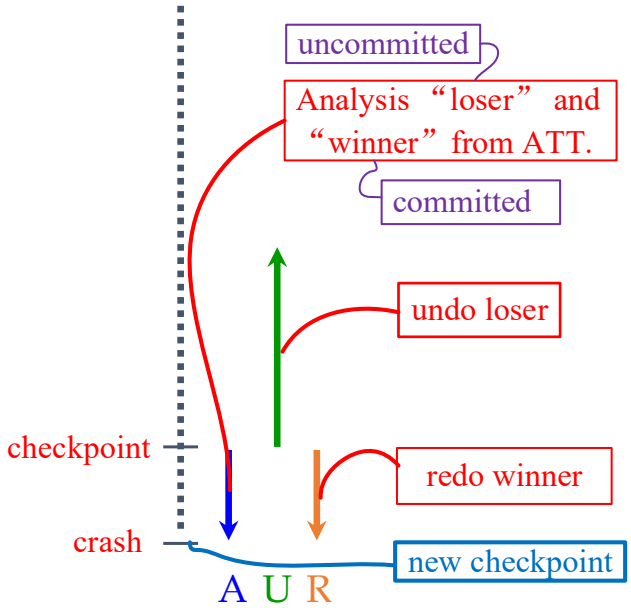
✓ REDOing all update between T and T+E transactions which are committed.

Recovery Algorithms of System R

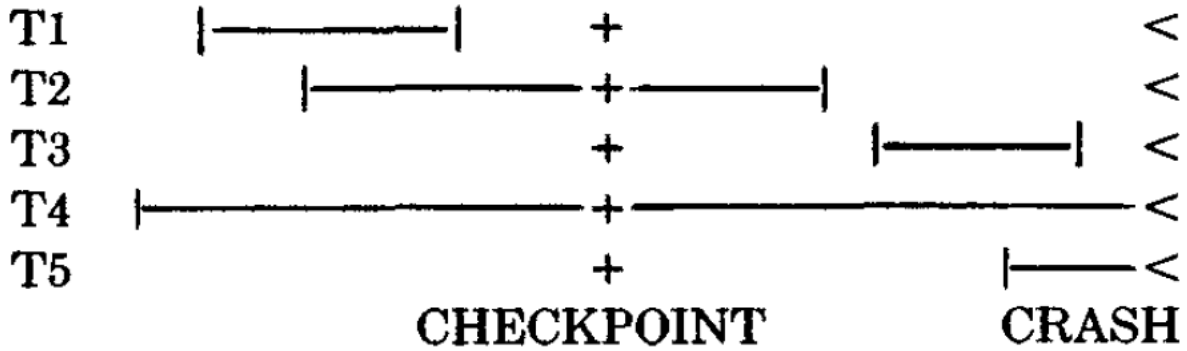
- Assumption: Given a checkpoint of the state at time T along with a log of all changes to the state made by transactions prior to time T + E.

✓ UNDOing all update prior to T+E transactions which were uncommited and abort.

✓ REDOing all update between T and T+E transactions which are committed.



An Recovery Example of Checkpoint in System R



- T_1, T_2, and T_3 committed, T_4 and T_5 uncommited.
  - ✓ No need undo, and no need redo for T_1.
  - ✓ No need undo, and need redo for T_2 and T_3.
  - ✓ UNDOing, and no need redo for T_4.
  - ✓ No need undo, and no need redo for T_5.

Issues for System R Recovery

- Log makes shadows redundant, and the shadow mechanism is quite expensive for large files.
- Recovery needs locking.
  - ∗ Objective!→ In order to blinder undo the actions of one transaction without ereasing subsequent updates by others transactions.
  - ✓ Method!→ All transactions lock all updates in exclusive mode and hold all such locks until the transaction is committed or undone.
  - ∗ Issue!→ Dead lock dection and prevention.

Issues for System R Recovery

- Log makes shadows redundant, and the shadow mechanism is quite expensive for large files.
- Recovery needs locking.
  - ∗ Objective!→ In order to blinder undo the actions of one transaction without ereasing subsequent updates by others transactions.
  - ✓ Method!→ All transactions lock all updates in exclusive mode and hold all such locks until the transaction is committed or undone.
  - ∗ Issue!→ Dead lock dection and prevention.

These issues are addressed in ARIES!

Table of Contents

ARIES Main Ideas

- Write-Ahead Logging - Extension
  - ✓ Any change is recorded in log on stable storage before the database change is written to disk.
  - ✓ Must use STEAL + NO-FORCE buffer pool policies.
- Repeating History During Redo - Idempotence
  - ✓ On restart, retrace actions and restore database to exact state before crash, which including the CLRs.
  - ✓ Gingerliness (小心翼翼) REDOing.
- Logging Changes During Undo - Non-idempotence
  - ✓ Record undo actions to log to ensure action is not repeated in the event of repeated failures.
  - ✓ Drastic (大刀阔斧) UNDOing.

The First Extension of WAL: LSNs

	Where	Definition
flushedLSN	Memory	**Last LSN in log on the disk**
recLSN	DPT	**Oldest LSN to a page since it was last flushed**
pageLSN	Page	**Newest LSN to a page**
lastLSN	ATT	**Last record of TXN**
MasterRecord	Disk	**LSN of last checkpoint**
preLSN	**Log**	**LSN of the previous log record written by this TXN using ATT**
LSN	**Log**	**LSN log record written by this TXN**

- ✓ Before page i is written to DB log must satisfy: pageLSN <= flushedLSN.
- ✓ LSNs are often implemented using the physical address of the log record in the log to enable the efficient location of a log record given its LSN.

The Big Picture: What's and Where

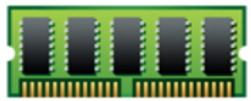
LogRecords

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

Data pages

each
with a
pageLSN

Master record



ATT lastLSN + status

DPT recLSN

flushedLSN

Buffer pool

Log tail

Data Structures of ARIES

- Active Transactions Table - ATT
  - ✓ One entry per currently active transaction, and entry removed when transaction commits or aborts
  - ✓ Containing TID, status (running/committing/aborting), and lastLSN (most recent LSN written by TID).
- Dirty Page Table - DPT
  - ✓ One entry per dirty page currently in buffer pool.
  - ✓ Containing recLSN (recoveryLSN) – the LSN of the log record which first caused the page to be dirty.

The Second Extension of WAL: CLRs

- Simple Transactions Abort
  - ✓ No crash involved.
  - ✓ We want to "play back" the log in reverse order, UNDOing updates.
  - ✓ Write a "CLR" (compensation log record) for each undone operation.
- Crash
  - ✓ You continue logging while you UNDO!!
  - ✓ At end of all UNDOs, write an "end" log record.

A Fuzzy Checkpoint

- When do ARIES to take checkpoint?
  - During normal operation, checkpoints are taken periodically.
- How do ARIES implement checkpoint?
  - BEGINCHECKPOINT record: Indicates when checkpoint began.
  - ENDCHECKPOIINT record: Contains current ATT and DPT. A 'fuzzy checkpoint':
    - ✓ Other transactions continue to run so that these tables (ATT and DPT) accurate only as of the time of the BEGINCHECKPOINT record.
    - ✓ No attempt to force dirty pages to disk.

A Fuzzy Checkpoint

- When do ARIES to take checkpoint?
  - During normal operation, checkpoints are taken periodically.
- How do ARIES implement checkpoint?
  - BEGINCHECKPOINT record: Indicates when checkpoint began.
  - ENDCHECKPOIINT record: Contains current ATT and DPT. A 'fuzzy checkpoint':
    - ✓ Other transactions continue to run so that these tables (ATT and DPT) accurate only as of the time of the BEGINCHECKPOINT record.
    - ✓ No attempt to force dirty pages to disk.
    - ✓ It just checkpoints main memory! why?

A Fuzzy Checkpoint

- When do ARIES to take checkpoint?
  - During normal operation, checkpoints are taken periodically.
- How do ARIES implement checkpoint?
  - BEGINCHECKPOINT record: Indicates when checkpoint began.
  - ENDCHECKPOIINT record: Contains current ATT and DPT. A 'fuzzy checkpoint':
    - ✓ Other transactions continue to run so that these tables (ATT and DPT) accurate only as of the time of the BEGINCHECKPOINT record.
    - ✓ No attempt to force dirty pages to disk.
    - ✓ It just checkpoints main memory! why? Because of log.
  - It is helpful to have a background process that periodically writes dirty pages to non-volatile storage.

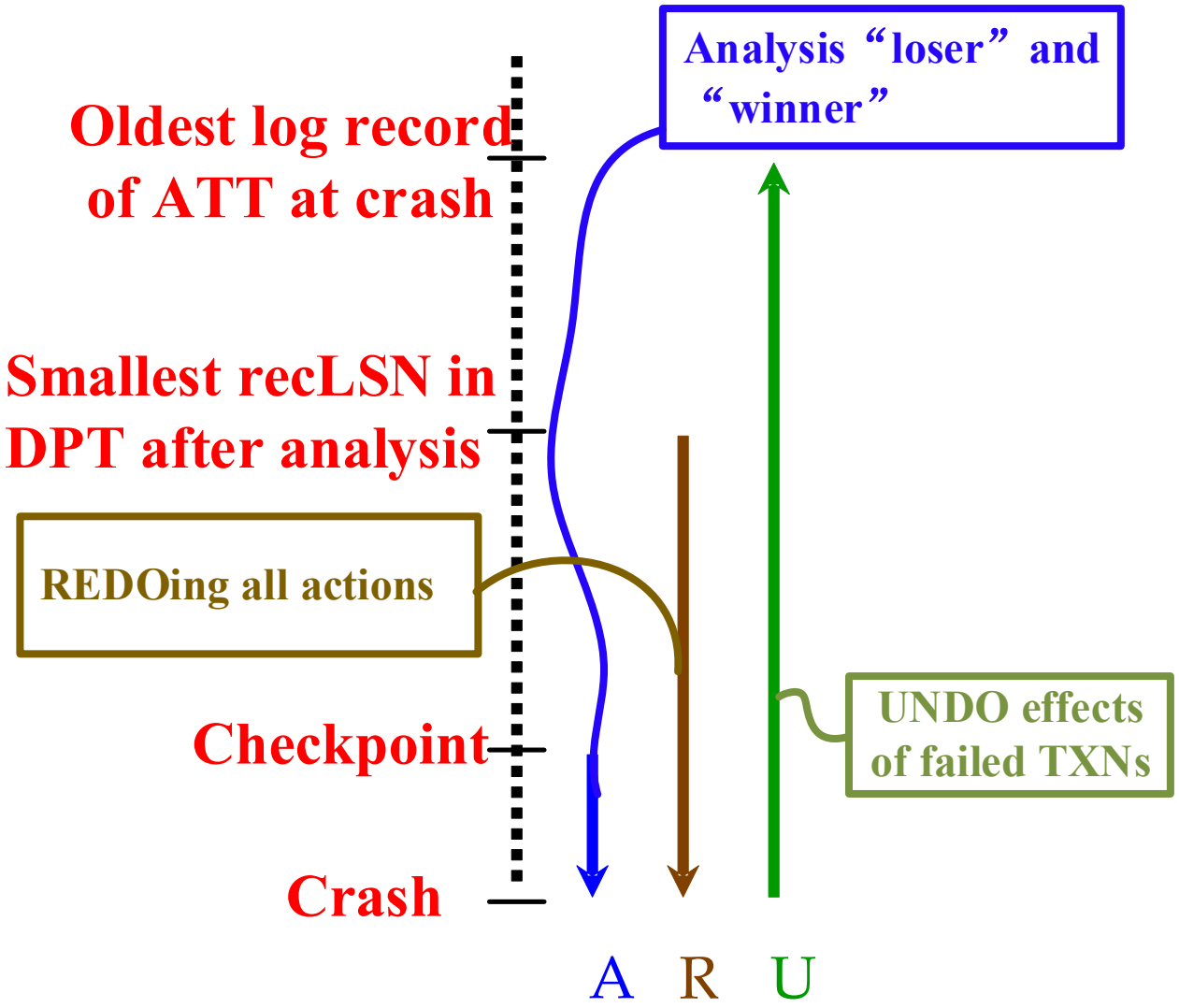
A Fuzzy Checkpoint

- When do ARIES to take checkpoint?
  - During normal operation, checkpoints are taken periodically.
- How do ARIES implement checkpoint?
  - BEGINCHECKPOINT record: Indicates when checkpoint began.
  - ENDCHECKPOIINT record: Contains current ATT and DPT. A 'fuzzy checkpoint':
    - ✓ Other transactions continue to run so that these tables (ATT and DPT) accurate only as of the time of the BEGINCHECKPOINT record.
    - ✓ No attempt to force dirty pages to disk.
    - ✓ It just checkpoints main memory! why? Because of log.
  - It is helpful to have a background process that periodically writes dirty pages to non-volatile storage. Why?

A Fuzzy Checkpoint

- When do ARIES to take checkpoint?
  - During normal operation, checkpoints are taken periodically.
- How do ARIES implement checkpoint?
  - BEGINCHECKPOINT record: Indicates when checkpoint began.
  - ENDCHECKPOIINT record: Contains current ATT and DPT. A 'fuzzy checkpoint':
    - ✓ Other transactions continue to run so that these tables (ATT and DPT) accurate only as of the time of the BEGINCHECKPOINT record.
    - ✓ No attempt to force dirty pages to disk.
    - ✓ It just checkpoints main memory! why? Because of log.
  - It is helpful to have a background process that periodically writes dirty pages to non-volatile storage. Why? Recovery effectiveness!

ARIES Algorithm



Notes of ARIES Algorithm

- Much of the power and relative simplicity of the ARIES algorithm is due to its REDO paradigm of repeating history, in which it redoes updates for all transactions - including those that will eventually be undone.
- Repeating history enables ARIES to employ a variant of the phyiological logging technique described earlier: it uses page-oriented REDO and a form of logical UNDO.
  - ✓ Page-oriented REDO means that REDO operations involve only a single page and that the affected page is specified in the log record.
  - ✓ Logical UNDO means that the operations performed to undo an update do not need to be the exact inverses of the operations of the original update.
    - ∗ Logical UNDO is used to support fine-grained (i.e., tuple-level) locking and high-currency index management.

MVCC DBMS's ARIES

- ♡ How do crash recovery change if the transactions uses the multi-version concurrency control in hybrid storage DBMS or in-memory DBMS?
  - ✓ It only contains analysis phase and redo phase for recovery algorithm. (Why do it reduce the undo phase?

MVCC DBMS's ARIES

- ♡ How do crash recovery change if the transactions uses the multi-version concurrency control in hybrid storage DBMS or in-memory DBMS?
  - ✓ It only contains analysis phase and redo phase for recovery algorithm. (Why do it reduce the undo phase? The log is multi-version database, and database is a current version cache at the log.)
  - ✓ It only contains ATT, which constructs transactional-consistent checkpoint.

Any Questions so far?

- Why do it use the WAL?

Any Questions so far?

- Why do it use the WAL?
  - ✓ Disk-based optimizations to minimize the cost of synchronous writes.
  - ✓ Leverage the sequential bandwidth of disk.
  - ✓ WAL-based systems write data to a log before updating the data in-place, sssbut they typically delay the in-place updates until they can be batched into larger sequential writes.

Any Questions so far?

- Why do it use the WAL?
  - ✓ Disk-based optimizations to minimize the cost of synchronous writes.
  - ✓ Leverage the sequential bandwidth of disk.
  - ✓ WAL-based systems write data to a log before updating the data in-place, sssbut they typically delay the in-place updates until they can be batched into larger sequential writes.
- Does disk-centric WAL design decisions fit into the NVRAM database systems?

Any Questions so far?

- Why do it use the WAL?
  - ✓ Disk-based optimizations to minimize the cost of synchronous writes.
  - ✓ Leverage the sequential bandwidth of disk.
  - ✓ WAL-based systems write data to a log before updating the data in-place, sssbut they typically delay the in-place updates until they can be batched into larger sequential writes.
- Does disk-centric WAL design decisions fit into the NVRAM database systems?
  - ✓ No!

Any Questions so far?

- Why do it use the WAL?
  - ✓ Disk-based optimizations to minimize the cost of synchronous writes.
  - ✓ Leverage the sequential bandwidth of disk.
  - ✓ WAL-based systems write data to a log before updating the data in-place, sssbut they typically delay the in-place updates until they can be batched into larger sequential writes.
- Does disk-centric WAL design decisions fit into the NVRAM database systems?
  - ✓ No!
- Why is not inappropriate?

Any Questions so far?

- Why do it use the WAL?
  - ✓ Disk-based optimizations to minimize the cost of synchronous writes.
  - ✓ Leverage the sequential bandwidth of disk.
  - ✓ WAL-based systems write data to a log before updating the data in-place, sssbut they typically delay the in-place updates until they can be batched into larger sequential writes.
- Does disk-centric WAL design decisions fit into the NVRAM database systems?
  - ✓ No!
- Why is not inappropriate?
  - ✓ Byte addressable access granularity!
  - ✓ Better access performance.

Any Questions so far?

- Write amplification!

Any Questions so far?

- Write amplification!
  - ✓ System R? Writing one logical byte to the database requires writing three bytes to storage.
  - ✓ ARIES? Writing one logical byte to the database requires writing three bytes to storage.

Any Questions so far?

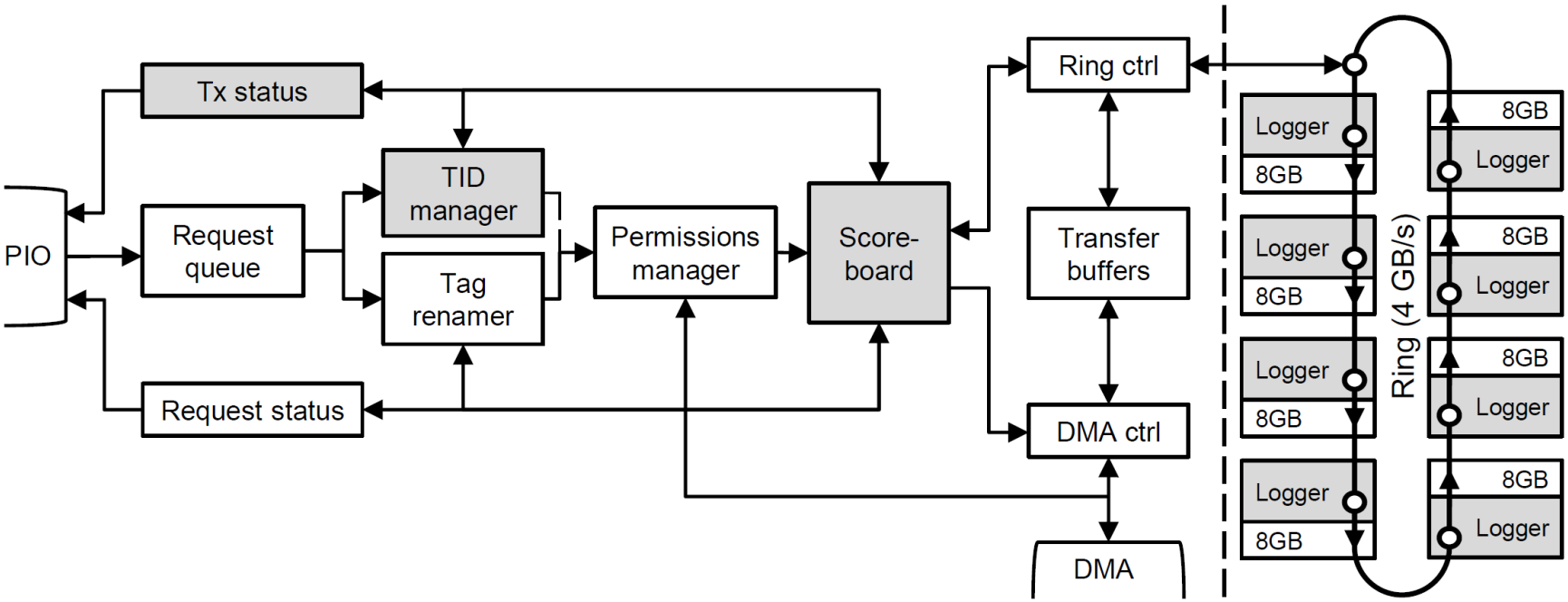
- Write amplification!
  - ✓ System R? Writing one logical byte to the database requires writing three bytes to storage.
  - ✓ ARIES? Writing one logical byte to the database requires writing three bytes to storage.
- LSN is not friendly for the large object!
  - ✓ Large object may exists across multiple pages.

Any Questions so far?

- Write amplification!
  - ✓ System R? Writing one logical byte to the database requires writing three bytes to storage.
  - ✓ ARIES? Writing one logical byte to the database requires writing three bytes to storage.
- LSN is not friendly for the large object!
  - ✓ Large object may exists across multiple pages.
- Centralized Logging is bottleneck!

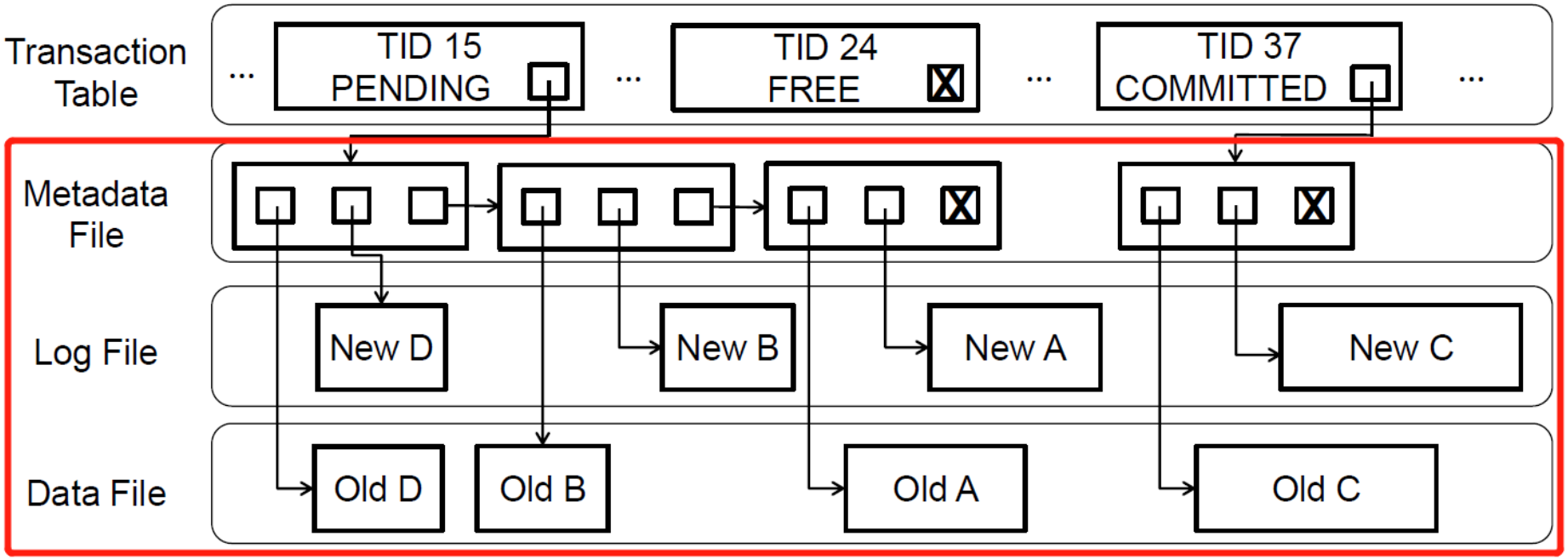
Table of Contents

Architecture of MARS



- The operations of a transaction on different objects are splitted over different loggers, which makes full use of internal bandwidth of hardware.
- The order of transaction operations are given by commit sequences number.

Architecture of MARS



- The design idea is close to System R, except for the elimination of shadow paging and undo logs.
- EAW primitive.
- No checkpoint mechanism.

EAW and Application Interface

Command	Description
`LogWrite`(TID, file, offset, data, len, logfile, logoffset)	Record a write to the log at the specified log offset. After commit, copy the data to the offset in the file.
`Commit`(TID)	Commit a transaction.
`AtomicWrite`(TID, file, offset, data, len, logfile, logoffset)	Create and commit a transaction containing a single write.
`NestedTopAction`(TID, logfile, logoffset)	Commit a nested top action by applying the log from a specified starting point to the end.
`Abort`(TID) `PartialAbort`(TID, logfile, logoffset)	Cancel the transaction entirely, or perform a partial rollback to a specified point in the log.

- Creating transactions.
- Commiting transactions.
  - ✓ Logical commit is the first phase.
  - ✓ Physical commit is the second phase.
- Aborting, etc.

Recovery Algorithm

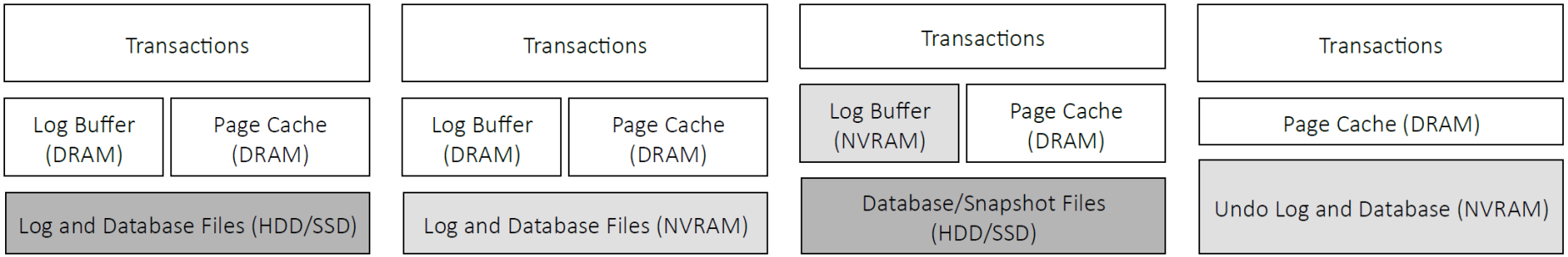
- Recovery is simple!
  - ✓ On boot, the driver scans the transaction tables of each logger to assemble a complete picture of transaction state across all the controllers.
  - ✓ A transaction must abort when some loggers may have marked a transaction as COMMITTED while others have not.
    - Deleting the metata simply.
  - ✓ Otherwise it commits and is reapply.
    - The system must apply the transactions in the order of their commit sequence numbers.

Advantage and Disadvantage

- Advantages
  - ✓ Software and hardware co-design, which makes full use of hardware bandwidth.
  - ✓ Reducing the write amplification.
  - ✓ Object-oriented log protocol which make WAL simple due to eliminating the LSN.
- Disadvantages
  - ∗ Concurrency control protocol becomes complicated, especially OCC.

Table of Contents

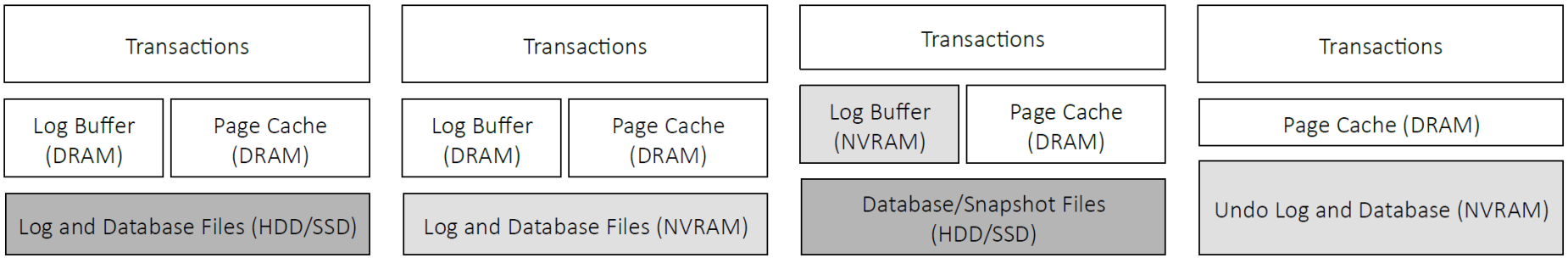
NVRAM Database Reconstructing



(a) Traditional Design (b) NV-Disk (c) NV-Logging (d) NV-WSP

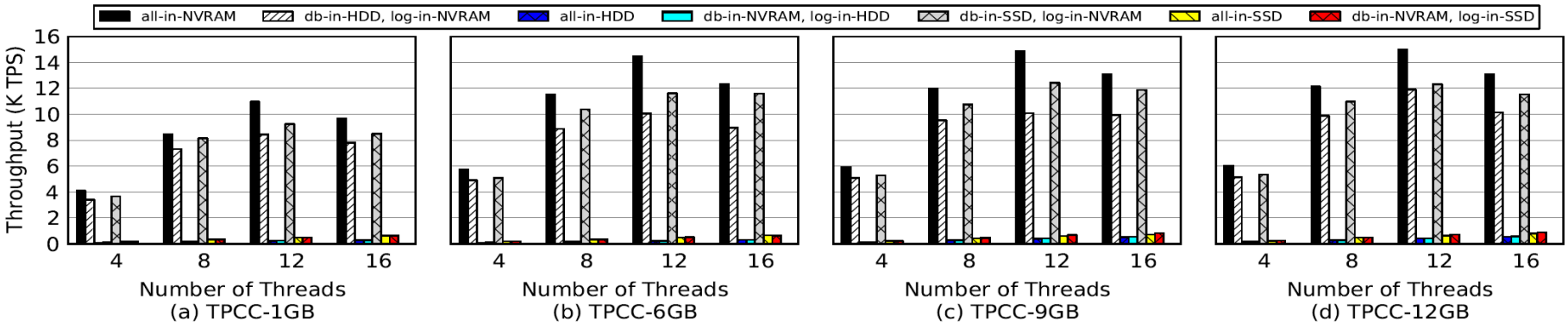
- a As faster NVRAM devices become available, with their access latencies close to that of DRAM, it becomes important to rethink this traditional disk-centric design.
- b This approach replaces the entire disk with NVRAM, and its interactions with NVRAM via file system APIs suffer from software overhead.

NVMRAM Database Reconstructing



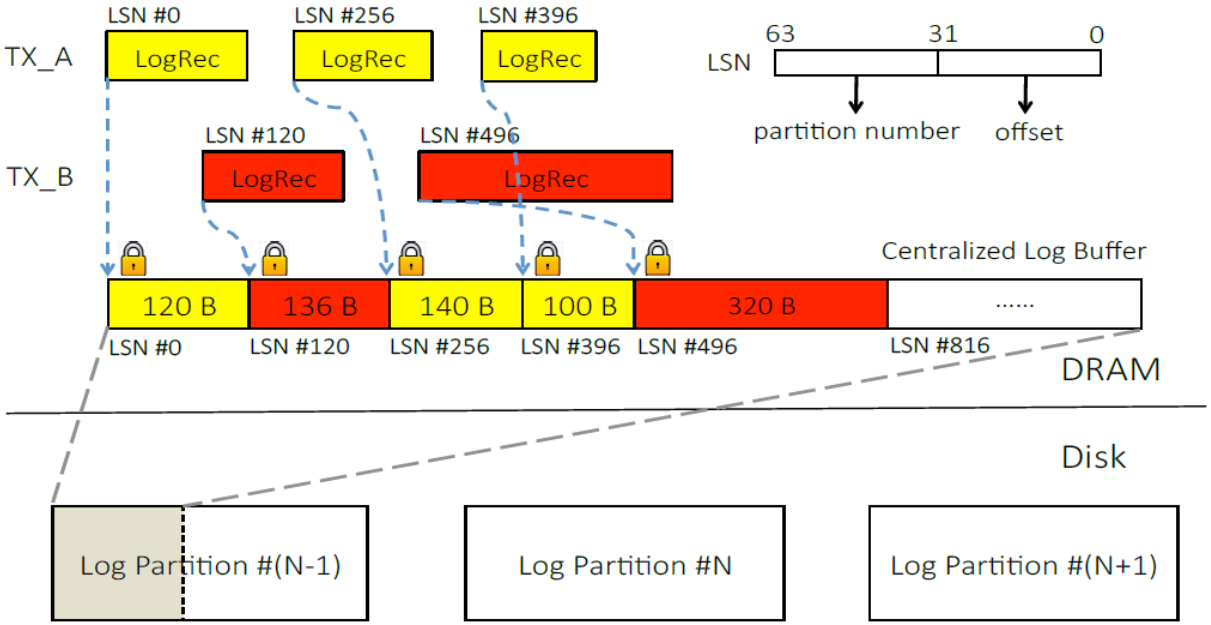
- c NV-Logging solution continues to use disks to store database tables, but uses NVRAM only to maintain logs. But snapshot of logging are stored on high capacity, low cost disks for recovery and long term use.
- d All database pages are hosted in NVRAM for NV-WSP.
  - Due to the slower speed of the state-of-the-art NVRAM technologies compared to DRAM, however, bridging this performance gap still requires caching pages in DRAM.
  - Its high cost suggests the need for alternative, more cost-effective approaches like the NV-Logging outlined above.

Performance Gap Analysis



- More interestingly, the throughput gained with option db-in-HDD, log-in-NVRAM comes close to the throughput of the much more expensive all-in-NVRAM option, reaching an average 74% of the performance of all-in-NVRAM.
- The performance gap is further reduced with db-in-SSD, log-in-NVRAM, reaching an average 82.5% of all-in-NVRAM's performance.

Disk-based Logging System

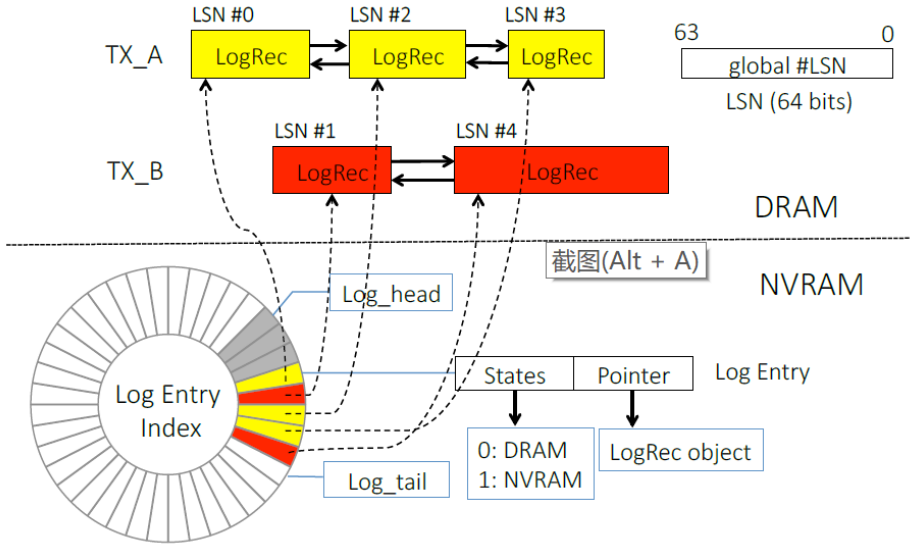


- e.g., The LSN of *TX_A*'s first log record is 0, the LSN of *TX_B*'s first log record is 120 as one log record whose size is 120 bytes from *TX_A* has been inserted into the log buffer. The procedure repeats for subsequent log records.

Decentralized Logging

- To avoid bottlenecks in the ceteralized log buffer, NV-logging leverages per-transaction logging for decentralized operation.
  - ✓ Each transaction maintains its own private log buffer for storing log records.
  - ✓ The NV-logging continues to use global LSN to track the order of logs, but it does not need log partitions and no offset is required, assigning a LSN to a log and updating its value are straighforward, efficient operations.

Decentralized Logging



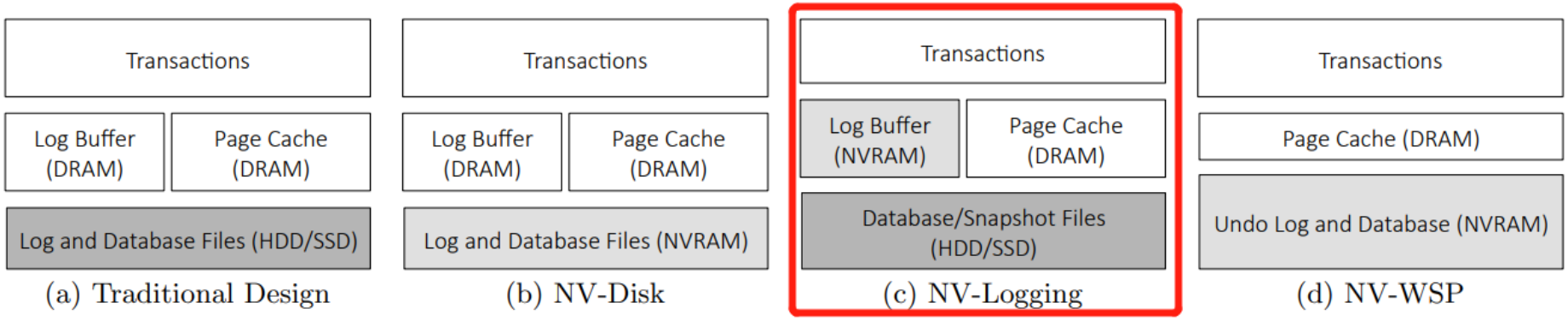
- The per-transaction log buffer is structured as a set of log entries organized as a circular buffer in NVRAM.
- A log record is first created in DRAM due to the the difference between DRAM and NVRAM write operation.
- NV-logging provides two persistence policies:
  - ✓ flush-on-insert uses clflush.
  - ✓ flush-on-commit combines group commit and clflush.

NV-logging's Checkpoint, Abort and Recovery

- Checkpoint is simple.
  - ✓ The checkpoint thread scans the log entry index from the log_head.
  - ✓ If the state bit is set 1, the checkpoint is created and the log_head moves to the checkpoint.
  - ✓ The log is trucated from old log_head to current checkpoint, but the log object's space is not deallocated.
- Transaction Abort
  - ✓ It is the same as the ARIES.
  - ✓ But, NV-logging uses the back pointers to obtain previous log object due to the byte addressable characteristic of NVRAM. → Disk-Logging uses preLSN.
- Recovery
  - ✓ NV-logging follows ARIES rules for database recovery.

Table of Contents

Design Decisions of WBL



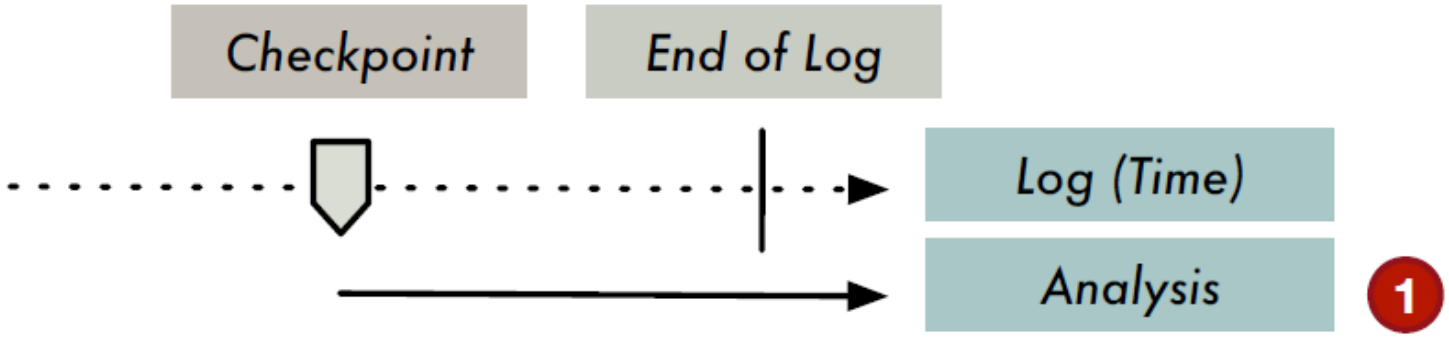
- WBL uses the NV-Logging scheme.
- Force + Steal buffer pool management policy.
  - ✓ Writing logging after page written to HDD firstly.
  - ✓ WBL makes full use of byte-addressable granularity and low-latency of random/sequencial read/write.
- Reconstructing the log entry.

Log Record of WBL

Checksum	LSN	Log Record Type	Persisted Commit Timestamp(C_p)	Dirty Commit Timestamp(C_d)

- Persisted commit timestamp (c_p).
- Dirty commit timestamp (c_d).
- This is a gap, c_d - c_p, which is equivalent to UNDOing the effects of any transaction that was active at the time of failure.
- Construct a checkpoint containing only the active commit identifier gaps.

Recovery Algorithm of WBL



- Recovery algorithm only contains an anslysis phase.
  - ✓ no redo. (Why?)
  - ✓ no undo in mvcc, need undo in single-version DBMS.
- Adding the checkpoint which reducing the analysis work.

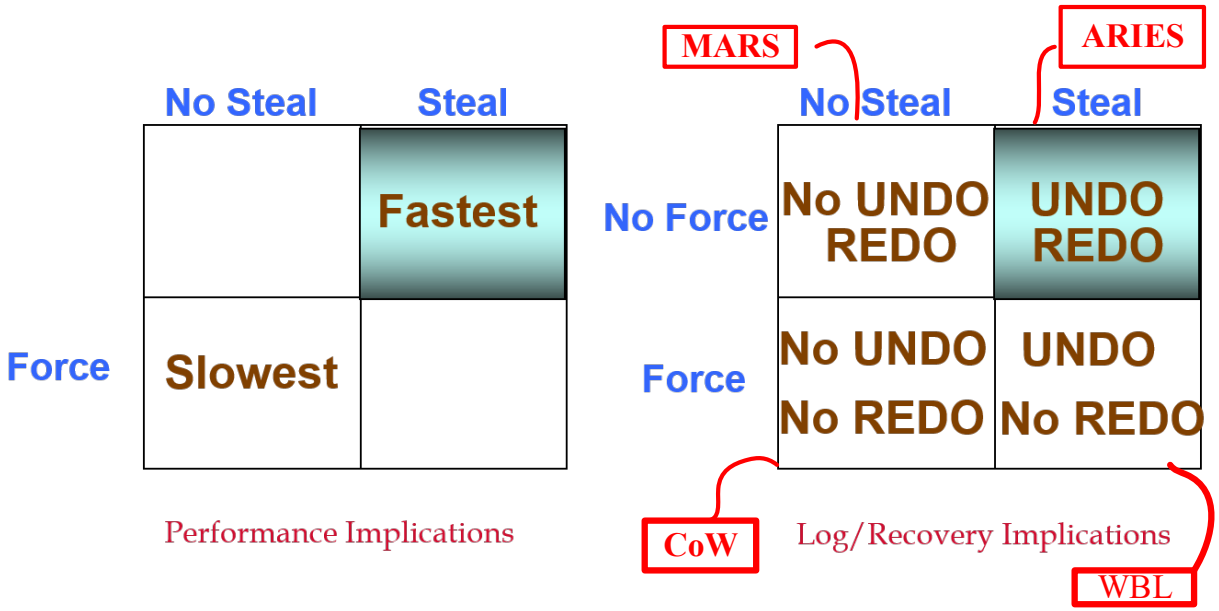
Table of Contents

Summary

- Recovery manager guarantees atomicity and durability.
  - ✓ Durablity of update.
  - ✓ Failure atomicity.
- Architecture researcher and database researcher work on the research from different angles.
- Software and hardware co-design is of great significance to it.

Summary

- Recovery manager guarantees atomicity and durability.
  - ✓ Durablity of update.
  - ✓ Failure atomicity.
- Architecture researcher and database researcher work on the research from different angles.
- Software and hardware co-design is of great significance to it.



Futures

- ♡ How do crash recovery change if the environment is new hardware architecture?
  - ✓ multi-core (众核 1000+)
  - ✓ NVRAM

Table of Contents

Thank you!
Welcome for any questions!

Ethan Zhu (朱君鹏)
Department of Computer Science and Technology
Nanjing University