Lecture 11: Persistent Memory Databases

CREATING THE NEXT®

Today's Agenda

Recap

Larger-than-Memory Databases

- Allow an in-memory DBMS to store/access data on disk **without** bringing back all the slow parts of a disk-oriented DBMS.
  - Minimize the changes that we make to the DBMS that are required to deal with disk-resident data.
  - It is better to have only the **buffer manager** deal with moving data around
  - Rest of the DBMS can assume that data is in DRAM.
- Need to be aware of hardware access methods
  - In-memory Access = **Tuple**-Oriented.
  - Disk Access = **Block**-Oriented.

Today's Agenda

- Disk-oriented vs In-Memory DBMSs
- Persistent Memory DBMSs
- Storage Engine Architectures
- Write-Behind Logging

Disk-oriented vs In-Memory DBMSs

Background

- Much of the development history of DBMSs is about dealing with the limitations of hardware.
- Hardware was much different when the original DBMSs were designed in 1970s:
  - Uniprocessor (single-core CPU)
  - DRAM capacity was very limited.
  - The database had to be stored on disk.
  - Disks were even slower than they are now.

Background

- But now DRAM capacities are large enough that most databases can fit in memory.
  - Structured data sets are smaller.
- We need to understand why we can't always use a "traditional" disk-oriented DBMS with a large cache to get the best performance.

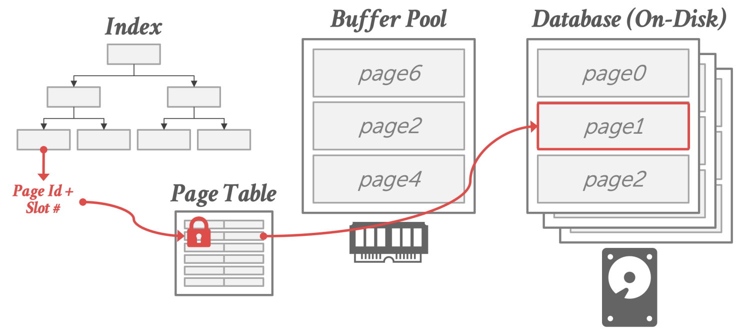
Disk-Oriented DBMS

- The primary storage location of the database is on non-volatile storage (*e.g.*, HDD, SSD).
- The database is organized as a set of fixed-length **pages** (aka blocks).
- The system uses an in-memory **buffer pool** to cache pages fetched from disk.
  - Its job is to manage the movement of those pages back and forth between disk and memory.

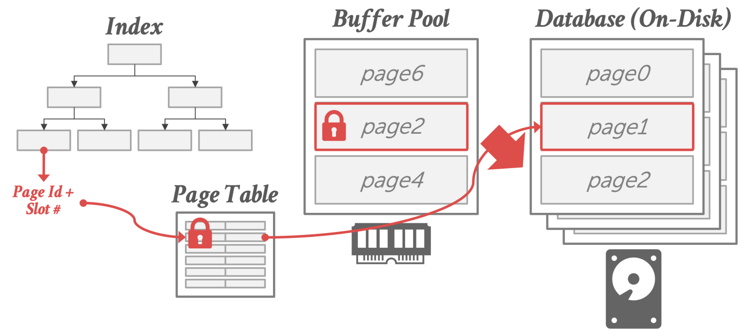
Buffer Pool

- When a query accesses a page, the DBMS checks to see if that page is already in memory:
    - If it's not, then the DBMS must retrieve it from disk and copy it into a **frame** in its buffer pool.
    - If there are no free frames, then find a page to evict.
    - If the page being evicted is dirty, then the DBMS must write it back to disk.
- Once the page is in memory, the DBMS translates any **on-disk addresses** to their **in-memory addresses**.

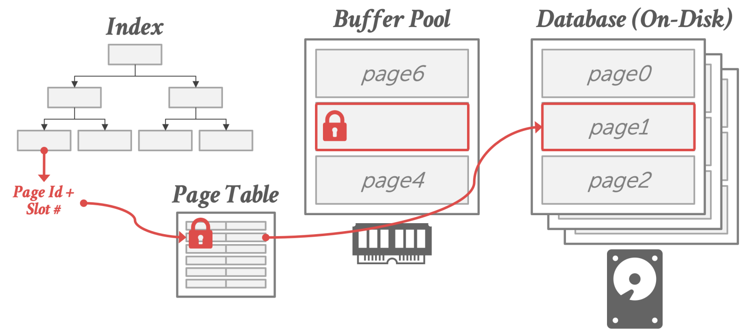
Disk-oriented DBMS: Data Organization



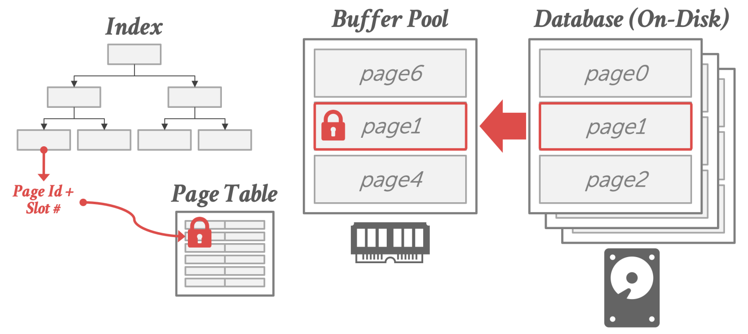
Disk-oriented DBMS: Data Organization



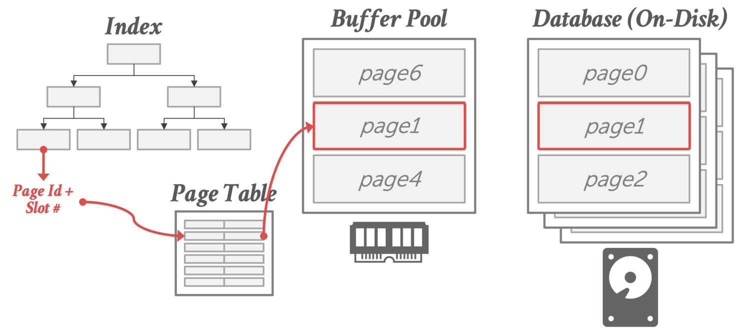
Disk-oriented DBMS: Data Organization



Disk-oriented DBMS: Data Organization



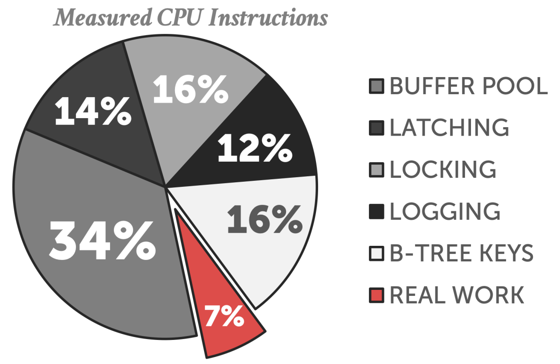
Disk-oriented DBMS: Data Organization



Buffer Pool

- Every tuple access goes through the buffer pool manager regardless of whether that data will always be in memory.
    - Always translate a tuple's record id to its memory location.
    - Worker thread must **pin** pages that it needs to make sure that they are not **swapped to disk**.

Disk-Oriented DBMS Overhead



Reference

In-memory DBMS

- Assume that the primary storage location of the database is **permanently** in memory.
- Early ideas proposed in the 1980s but it is now feasible because DRAM prices are low and capacities are high.
- First commercial in-memory DBMSs were released in the 1990s.
  - **Examples:** TimesTen, DataBlitz, Altibase

Storage Access Latencies

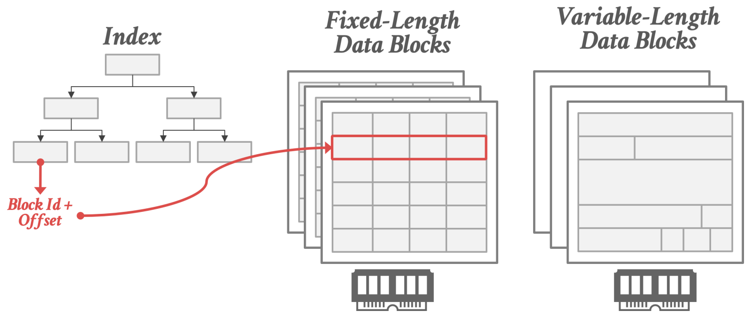
	L3	DRAM	SSD	HDD
Read Latency	20 ns	60 ns	25,000 ns	10,000,000 ns
Write Latency	20 ns	60 ns	300,000 ns	10,000,000 ns

Reference

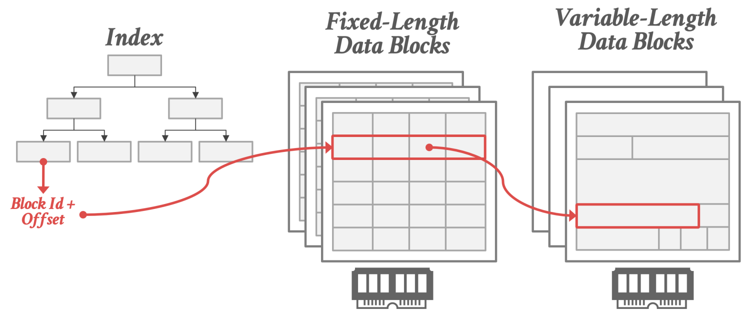
In-Memory DBMS: Data Organization

- An in-memory DBMS does **not** need to store the database in slotted pages but it will still organize tuples in pages:
    - **Direct memory pointers** vs. record ids
    - Fixed-length vs. variable-length data **memory pools**
    - Use checksums to detect software errors from trashing the database.
- The OS organizes memory in pages too. We already covered this.

In-Memory DBMS: Data Organization



In-Memory DBMS: Data Organization



Persistent Memory DBMSs

Importance of Hardware

- People have been thinking about using hardware to accelerate DBMSs for decades.
- 1980s: Database Machines
- 2000s: FPGAs + Appliances
- 2010s: FPGAs + GPUs
- 2020s: PM + FPGAs + GPUs + CSAs + More! Reference

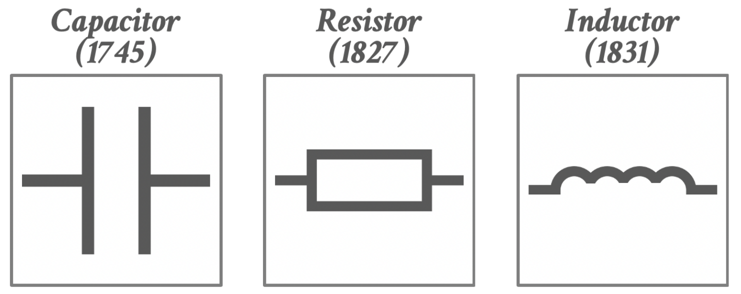
Persistent Memory

- Emerging storage technology that provide low latency read/writes like DRAM, but with persistent writes and large capacities like SSDs.
    - *a.k.a.*, Non-Volatile Memory, Storage-class Memory
- First-generation devices were block-addressable
- Second-generation devices are byte-addressable

Persistent Memory

- Block-addressable Optane SSD
  - NVM Express works with PCI Express to transfer data to and from Optane SSDs
  - NVMe enables rapid storage in SSDs and is an improvement over older HDD-related interfaces (*e.g.*, Serial Attached SCSI (**SAS**) and Serial ATA (**SATA**))
- Byte-addressable Optane DIMMs
  - New assembly instructions and hardware support

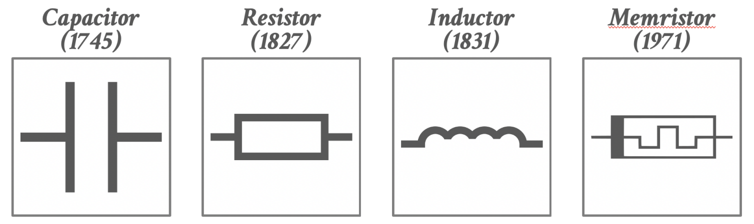
Fundamental Elements of Circuits



Fundamental Elements of Circuits

- In 1971, Leon Chua at Berkeley predicted the existence of a fourth fundamental element.
- A two-terminal device whose resistance depends on the voltage applied to it, but when that voltage is turned off it **permanently remembers** its last resistive state.
- Reference

Fundamental Elements of Circuits



Memristors

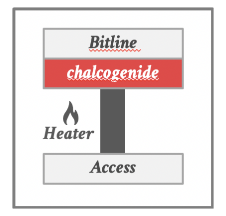
- A team at HP Labs led by Stanley Williams stumbled upon a nano-device that had weird properties that they could not understand.
- It wasn't until they found Chua's 1971 paper that they realized what they had invented.
- Reference
- Video

NVM Technologies

- Phase-Change Memory (PRAM)
- Resistive RAM (ReRAM)
- Magnetoresistive RAM (MRAM)

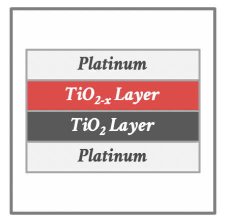
Phase-Change Memory

- Storage cell is comprised of two metal electrodes separated by a resistive heater and the phase change material (**chalcogenide**).
- The value of the cell is changed based on how the material is heated.
  - A short pulse changes the cell to a '0'.
  - A long, gradual pulse changes the cell to a '1'.
- Reference



Resistive RAM

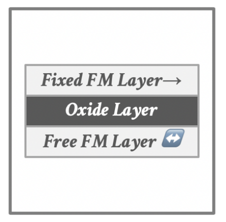
- Two metal layers with two TiO2 layers in between.
- Running a current one direction moves electrons from the top TiO2 layer to the bottom, thereby changing the resistance.
- Potential programmable storage fabric. . .
  - Bertrand Russell's Material Implication Logic
- Reference



Magnetoresistive RAM

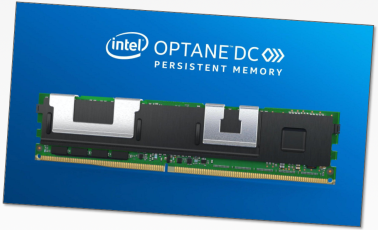
- Stores data using magnetic storage elements instead of electric charge or current flows.
- Spin-Transfer Torque (STT-MRAM) is the leading technology for this type of PM.
  - Supposedly able to scale to very smallsizes (10nm) and have **SRAM**-like latencies. What is SRAM used for?

Reference

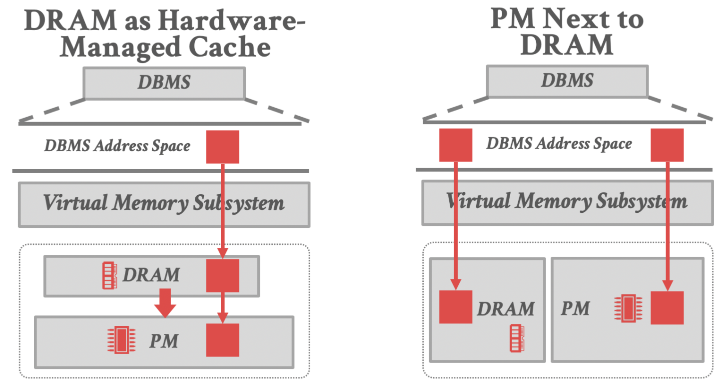


Why This is for Real

- Industry has agreed to standard technologies and form factors (JDEC).
- Linux and Microsoft added support for PM in their kernels (DAX).
- Intel added new instructions for flushing cache lines to PM (CLFLUSH, CLWB).



PM Configurations



Reference

PM for Database Systems

- Block-addressable PM is not that interesting.
- Byte-addressable PM will be a game changer but will require some work to use correctly.
    - In-memory DBMSs will be better positioned to use byte-addressable PM.
    - Disk-oriented DBMSs will initially treat PM as just a faster SSD.

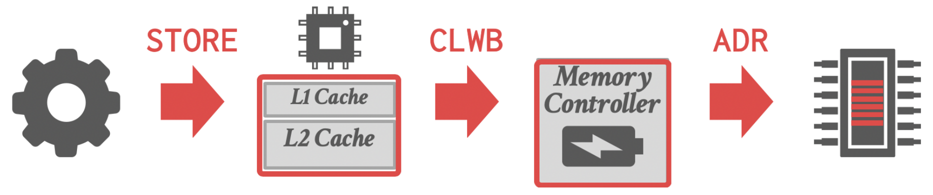
Storage & Recovery Methods

- Understand how a DBMS will behave on a system that only has byte-addressable PM.
- Develop PM-optimized implementations of standard DBMS architectures.
- Based on the N-Store prototype DBMS.
- Reference

Synchronization

- Existing programming models assume that any write to memory is non-volatile.
  - CPU decides when to move data from caches to DRAM.
- The DBMS needs a way to ensure that data is flushed from caches to PM.

Synchronization



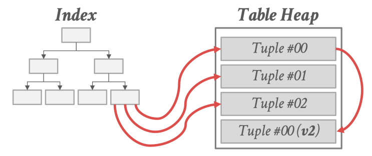
Synchronization

- Cache-line Flush (CLFLUSH)
  - This instruction allows the DBMS to flush a cache-line out to memory.
  - If that cache line contains modified data at any level of the cache hierarchy, that data is written back to memory.
- Cache-line Write Back (CLWB)
  - Writes back the cache line (if modified) to memory
  - The cache line may be retained in the cache hierarchy in non-modified state
  - Improves performance by reducing cache misses
  - CLWB instruction is ordered only by store-fencing (SFENCE) operation.
- Asynchronous DRAM Refresh (ADR)
  - In case of a power loss, there is sufficient reserve power to flush the stores pending in the memory controller back to Optane DIMM.
  - Stores are posted to the Write Pending Queue (WPQ) in the memory controller

Reference

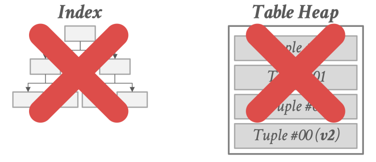
Naming

- If the DBMS process restarts, we need to make sure that all the pointers for in-memory data point to the same data.



Naming

- If the DBMS process restarts, we need to make sure that all the pointers for in-memory data point to the same data.



PM-Aware Memory Allocator

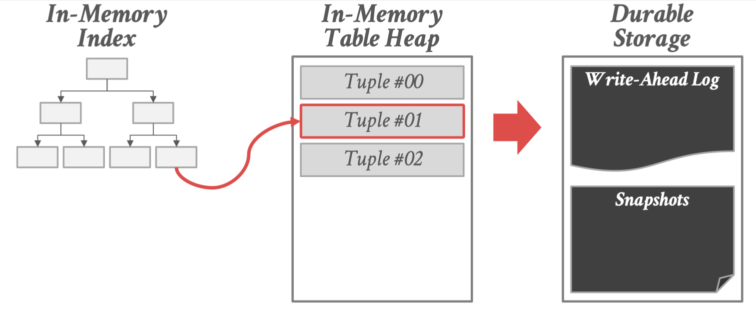
- **Feature 1: Synchronization**
  - The allocator writes back CPU cache lines to PM using the `CLFLUSH` instruction.
  - It then issues a `SFENCE` instruction to wait for the data to become durable on PM.
- **Feature 2: Naming**
  - The allocator ensures that virtual memory addresses assigned to a memory-mapped region never change even after the OS or DBMS restarts.

Storage Engine Architectures

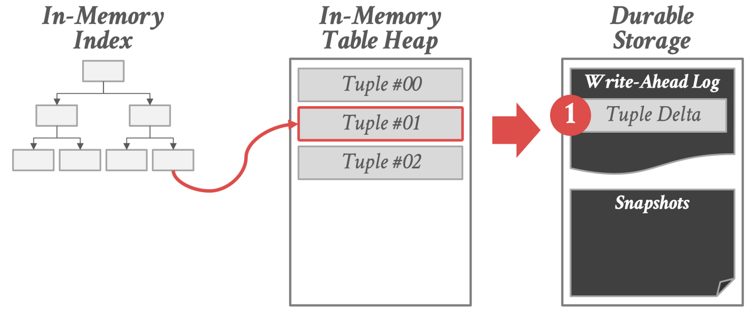
Storage Engine Architectures

- **Choice 1: In-place Updates**
  - Table heap with a write-ahead log + snapshots.
  - Example: VoltDB
- **Choice 2: Copy-on-Write**
  - Create a shadow copy of the table when updated.
  - No write-ahead log.
  - Example: LMDB
- **Choice 3: Log-structured**
  - All writes are appended to log. No table heap.
  - Example: RocksDB

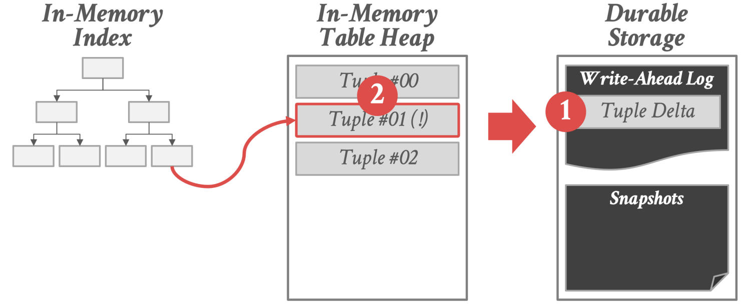
In-place Updates Engine



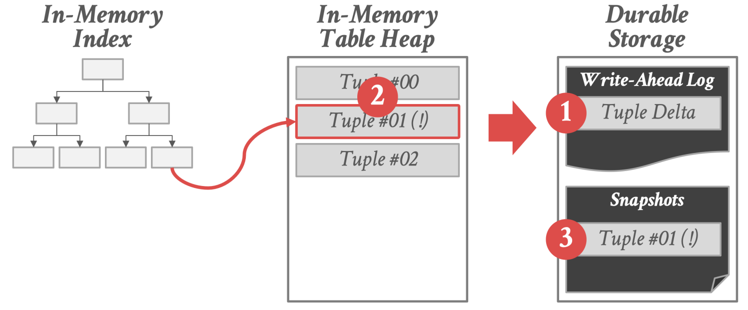
In-place Updates Engine



In-place Updates Engine



In-place Updates Engine



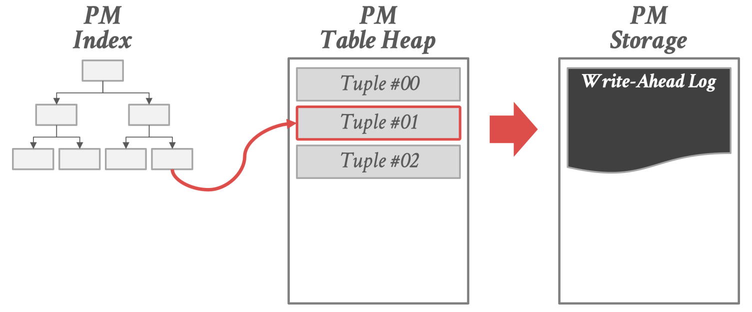
In-place Updates Engine

- Limitations
  - Duplicate Data
  - Recovery Latency

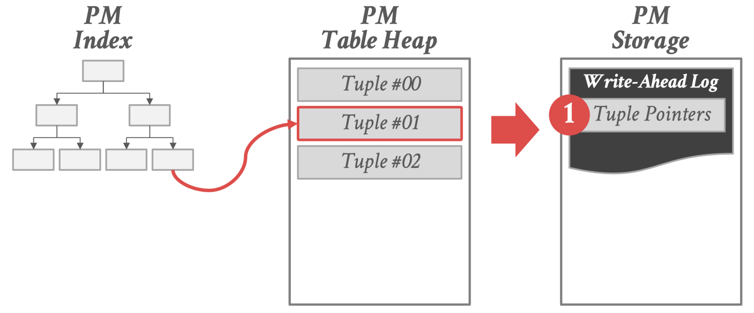
PM-Aware Architectures

- Leverage the allocator's **non-volatile pointers** to only record **what** changed rather than **how** it changed.
- The DBMS only must maintain a transient **UNDO log** for a txn until it commits.
  - Dirty cache lines from an uncommitted txn can be flushed by hardware to the memory controller.
  - **No REDO log** because we flush all the changes to PM at the time of commit.

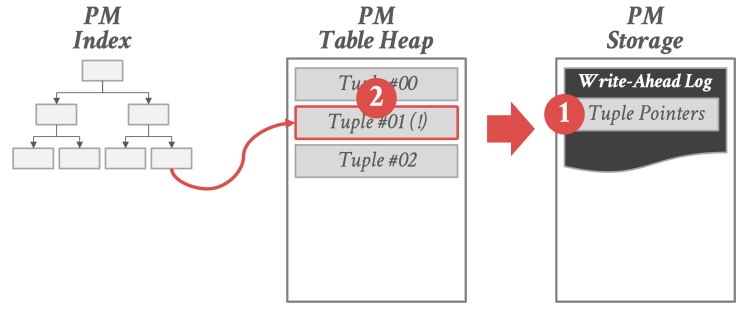
PM-Aware In-place Updates Engine



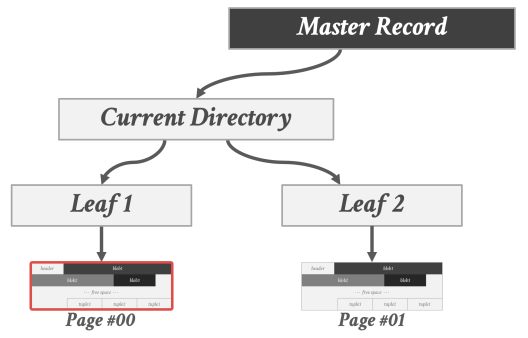
PM-Aware In-place Updates Engine



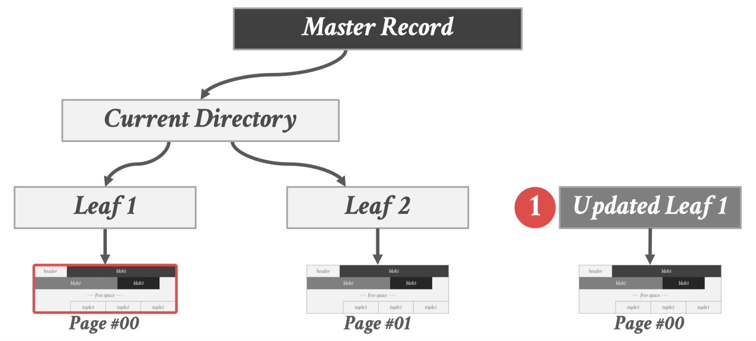
PM-Aware In-place Updates Engine



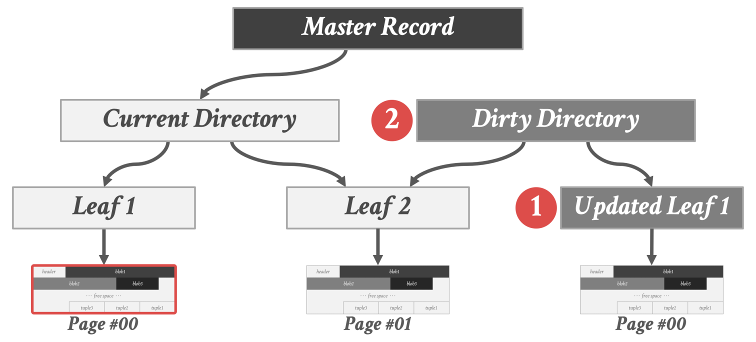
Copy-On-Write Engine



Copy-On-Write Engine



Copy-On-Write Engine



Master Record

Current Directory

2 Dirty Directory

Leaf 1

Leaf 2

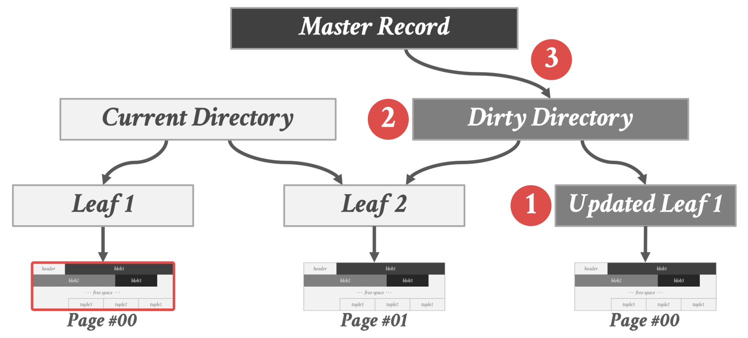
1 Updated Leaf 1

Page #00

Page #01

Page #00

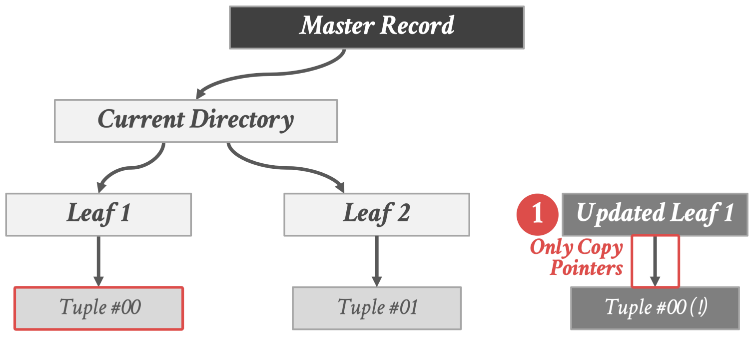
Copy-On-Write Engine



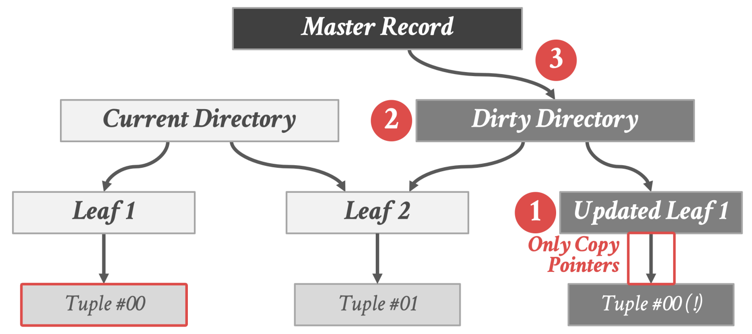
Copy-On-Write Engine

- Limitations
    - Expensive Copies

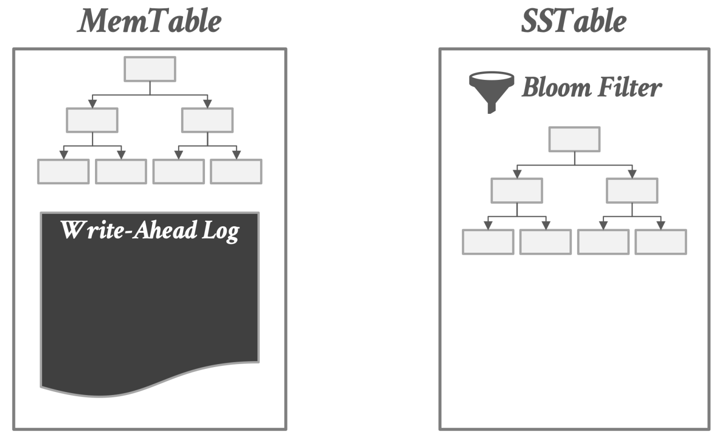
PM-Aware Copy-On-Write Engine



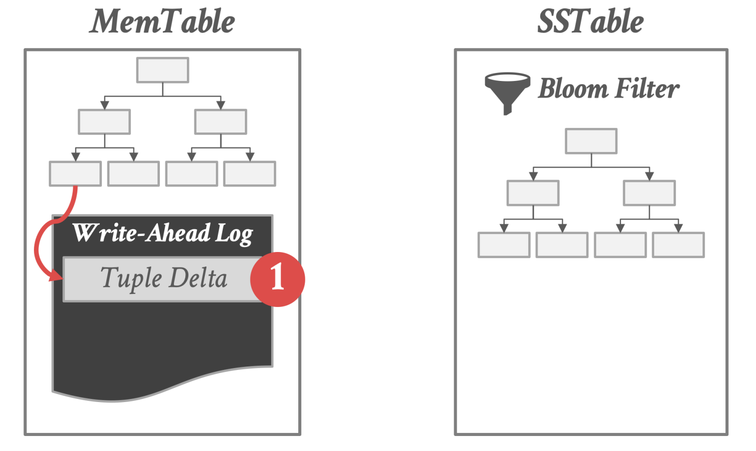
PM-Aware Copy-On-Write Engine



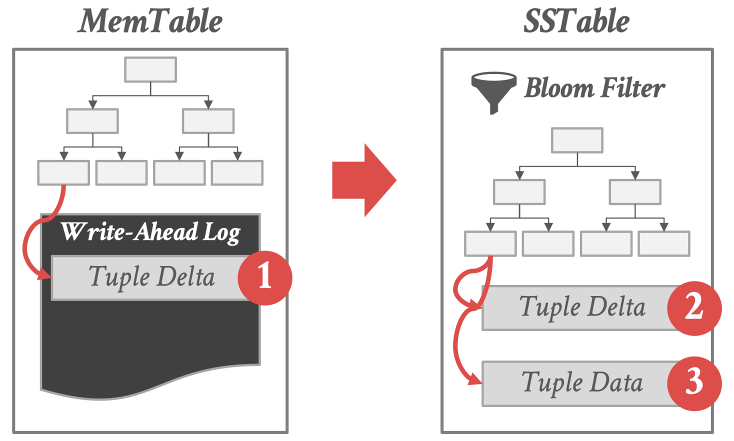
Log-Structured Engine



Log-Structured Engine



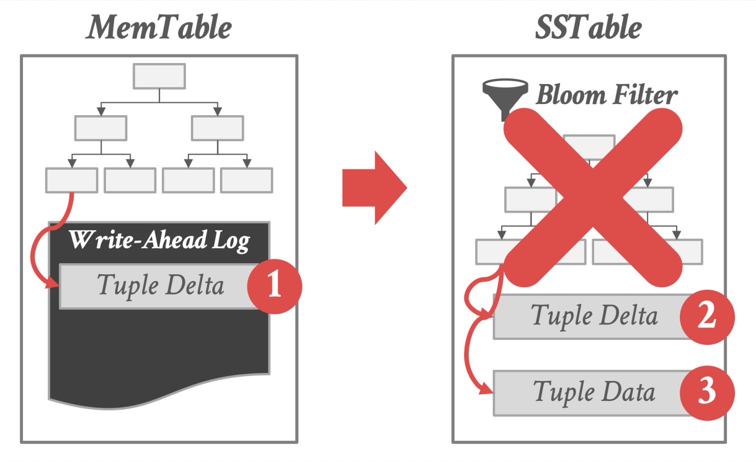
Log-Structured Engine



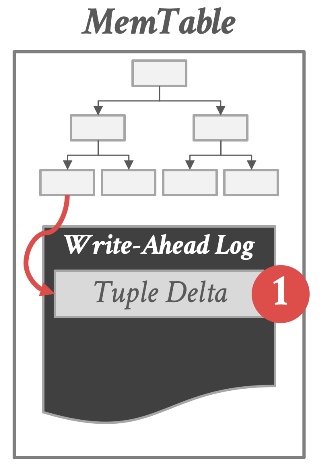
Log-Structured Engine

- Limitations
    - Duplicate Data
    - Compactions

PM-Aware Log-Structured Engine



PM-Aware Log-Structured Engine



Write-Behind Logging

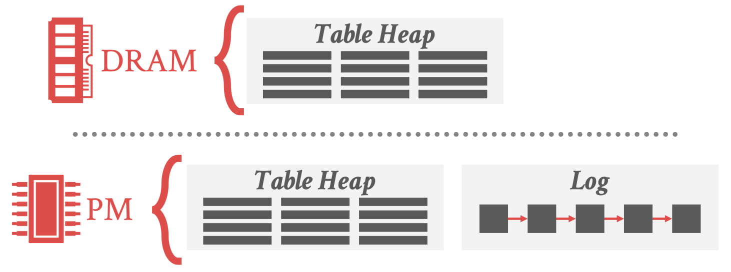
Observation

- WAL serves two purposes
  - Transform random writes into sequential log writes.
  - Support transaction rollback.
  - Design makes sense for disks with slow random writes.
- But PM supports fast random writes
  - Directly write data to the multi-versioned database.
  - Only record **meta-data** about committed txns in log.

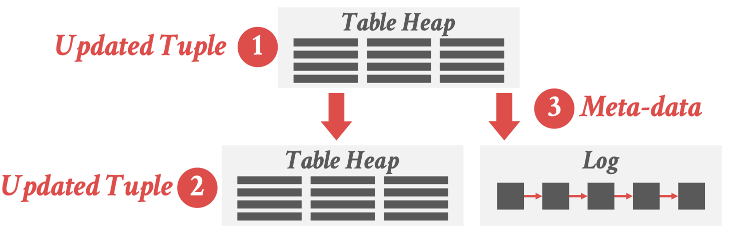
Write-Behind Logging

- PM-centric logging protocol that provides instant recovery and minimal duplication overhead.
  - Directly propagate changes to the database.
  - Only record meta-data in log.
  - Reference
- Recover the database almost instantaneously.
  - Need to record meta-data about in-flight transactions.
  - In case of failure, ignore their effects.

Write-Behind Logging



Write-Behind Logging



Write-Behind Logging

- DBMS assigns timestamps to transactions
  - Get timestamps within same group commit timestamp range to identify and ignore effects of in-flight txns.
- Use failed group commit timestamp range:
  - DBMS uses range during tuple visibility checks.
  - Ignores tuples created or updated within this range.
  - UNDO is implicitly done via visibility checks.

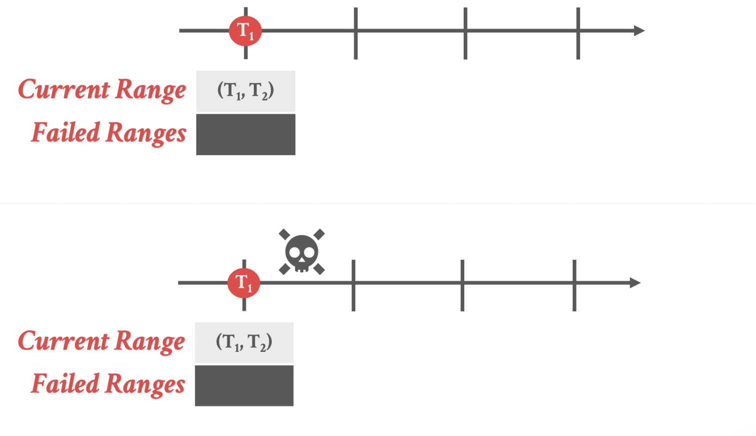
Write-Behind Logging

- Recovery consists of only analysis phase
  - The DBMS can immediately start processing transactions after restart with explicit UNDO/REDO phases.
- Garbage collection eventually kicks in to remove the physical versions of uncommitted transactions.
  - Using timestamp range information in write-behind log.
  - After this finishes, no need to do extra visibility checks.

Metadata for Instant Recovery

- Use group commit timestamp range to ignore effects of transactions in failed group commit.
    - Maintain list of failed timestamp ranges.

Metadata for Instant Recovery



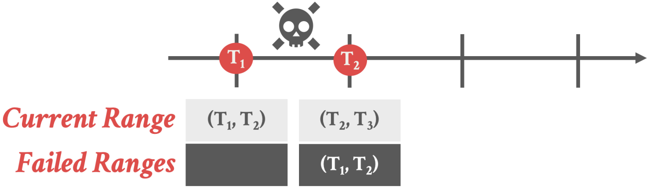
Current Range (T_1, T_2)

Failed Ranges

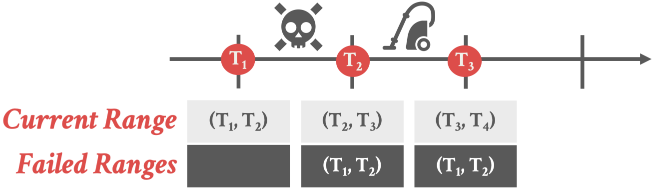
Current Range (T_1, T_2)

Failed Ranges

Metadata for Instant Recovery

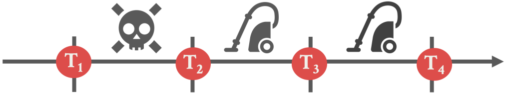


Current Range	(T_1, T_2)	(T_2, T_3)
Failed Ranges		(T_1, T_2)



Current Range	(T_1, T_2)	(T_2, T_3)	(T_3, T_4)
Failed Ranges		(T_1, T_2)	(T_1, T_2)

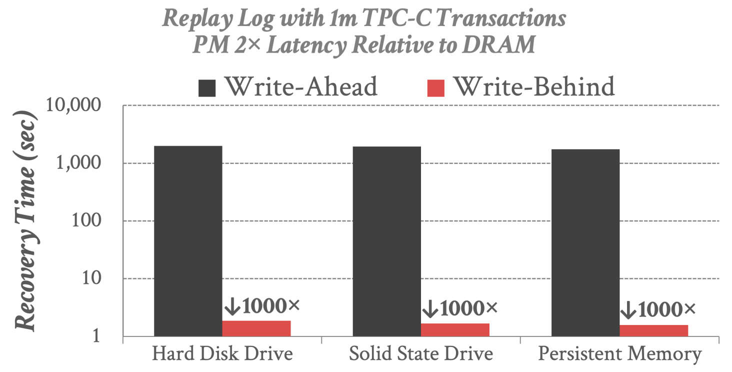
Metadata for Instant Recovery



Current Range	(T_1, T_2)	(T_2, T_3)	(T_3, T_4)	(T_4, T_5)
Failed Ranges		(T_1, T_2)	(T_1, T_2)	

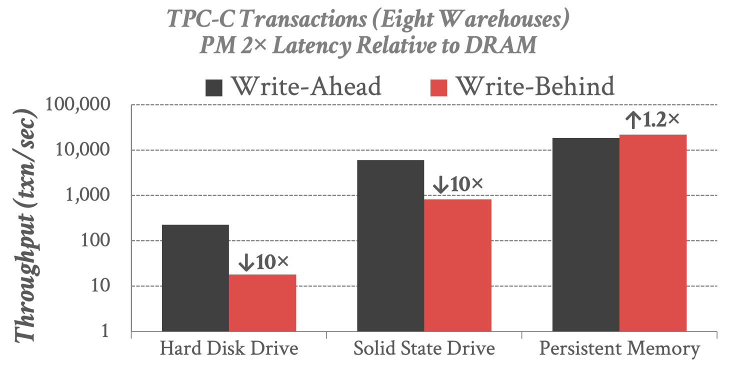
Write-Behind Logging – Recovery

- Replay Log with 1m TPC-C Transactions
- PM 2× Latency Relative to DRAM



Write-Behind Logging – Runtime

- TPC-C Transactions (Eight Warehouses)
- PM 2× Latency Relative to DRAM



Conclusion

PM Summary

- Optimization of **Storage Engine Architectures**
  - Leverage byte-addressability to **avoid unnecessary data duplication**.
- Optimization of **Logging and Recovery Protocol**
  - PM-optimized recovery protocols avoid the overhead of processing a log.
  - Non-volatile data structures ensure consistency.

Parting Thoughts

- The design of a in-memory DBMS is significantly different than a disk-oriented system.
- The world has finally become comfortable with in-memory data storage and processing.
- Byte-addressable PM is going to be a game changer.
- We are likely to see many new computational components that DBMSs can use in the next decade.
  - The core ideas / algorithms will still be the same.

Next Class

- Concurrency Control