

Lecture 10: Larger-than-Memory Databases

Time

Take-down

Recap

- # of threads
- size of buffer pool
- # of ops

Naïve Compression

- Choice 1: Entropy Encoding
 - ▶ More common sequences use less bits to encode, less common sequences use more bits to encode.
- Choice 2: Dictionary Encoding
 - ▶ Build a data structure that maps data segments to an identifier.
 - ▶ Replace the segment in the original data with a reference to the segment's position in the dictionary data structure.

the → 010
database → 00100011

the : 1
database : 2

Columnar Compression

- Null Suppression
- Run-length Encoding
- Bitmap Encoding
- Delta Encoding
- Incremental Encoding
- Mostly Encoding
- Dictionary Encoding

Today's Agenda

- Background
- Design Decisions
- Case Studies

Background

Observation

- DRAM is expensive (roughly \$10 per GB)
 - ▶ Expensive to buy
 - ▶ Expensive to maintain (e.g., energy associated with refreshing DRAM state).
- SSD is 50 × cheaper than DRAM (roughly \$0.2 per GB)
- It would be nice if an in-memory DBMS could use cheaper storage without having to bring in the entire baggage of a disk-oriented DBMS.



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. Why?
 - ▶ Disk Access = Block-Oriented.

4096B
 128B
 32

64B

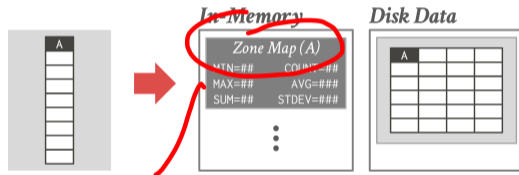
128B

4096B

4KB/HDD
 16KB/SSD

OLAP

- OLAP queries generally access the entire table.
- Thus, an in-memory DBMS may handle OLAP queries in the same a disk-oriented DBMS does.
- All the optimizations in a disk-oriented DBMS apply here (e.g., scan sharing, buffer pool bypass).



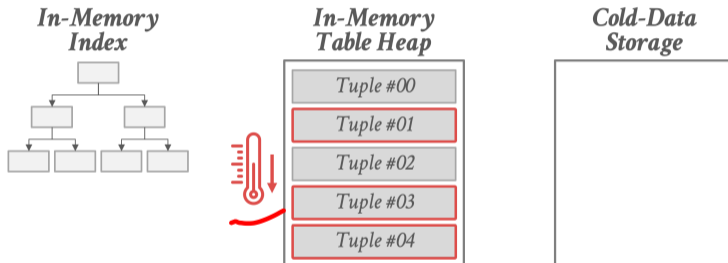
OLTP

1 instagram

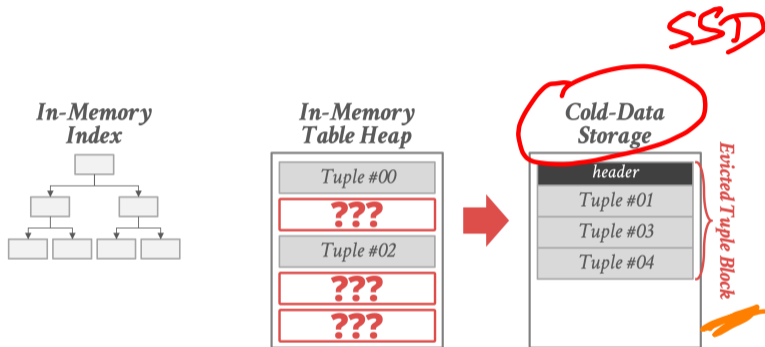
- OLTP workloads almost always have hot and cold portions of the database.
 - ▶ We can assume txns will almost always access hot tuples.
- **Goal:** The DBMS needs a mechanism to move cold data out to disk and then retrieve it if it is ever needed again.

Caching
 → Hot data - DRAM
 → Cold data - SSD
 Anti-Caching

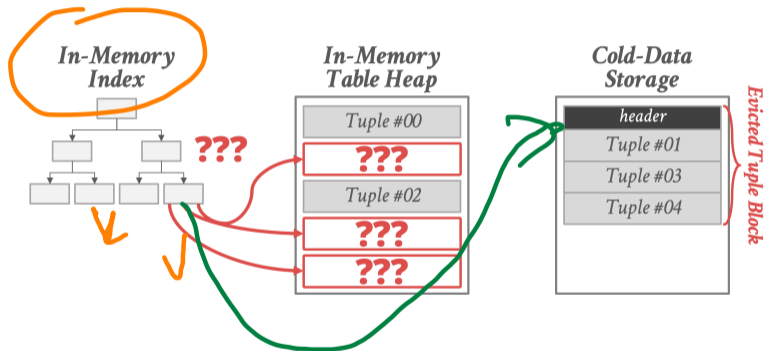
Larger-than-Memory Databases



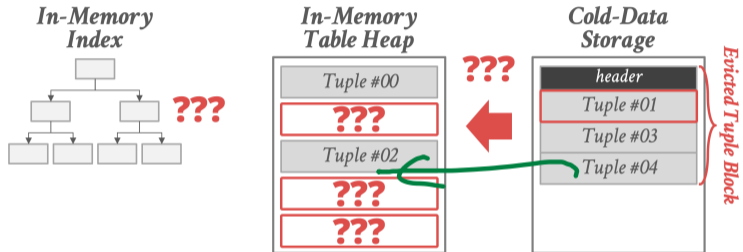
Larger-than-Memory Databases



Larger-than-Memory Databases



Larger-than-Memory Databases



```
SELECT *  
FROM table  
WHERE id = <Tuple 01>
```



Design Decisions

Design Decisions

• Run-time Operation

- ▶ Cold Data Identification: When the DBMS runs out of DRAM space, what data should we evict?

• Eviction Policies

- ▶ Timing: When to evict data?
- ▶ Evicted Tuple Metadata: During eviction, what meta-data should we keep in DRAM to track disk-resident data and avoid false negatives?

• Data Retrieval Policies

- ▶ Granularity: When we need data, how much should we bring in?
- ▶ Merging: Where to put the retrieved data?

Reference

Cold Data Identification

- **Choice 1: On-line**

- ▶ The DBMS monitors txn access patterns and tracks how often tuples/pages are used.
- ▶ Embed the tracking meta-data directly in tuples/pages.

- **Choice 2: Off-line**

- ▶ Maintain a tuple access log during txn execution.
- ▶ Process in background to compute frequencies.

Eviction Timing

- Choice 1: Threshold

- ▶ The DBMS monitors memory usage and begins evicting tuples when it reaches a threshold.
- ▶ The DBMS must manually move data.

- Choice 2: On Demand

- ▶ The DBMS/OS runs a replacement policy to decide when to evict data to free space for new data that is needed.

Evicted Tuple Metadata

• Choice 1: Tuple Tombstones

- ▶ Leave a marker that points to the on-disk tuple.
- ▶ Update indexes to point to the tombstone tuples.

• Choice 2: Bloom Filters

- ▶ Use an in-memory, approximate data structure for each index.
- ▶ Only tells us whether tuple exists or not (with potential false positives)
- ▶ Check on-disk index to find actual location

• Choice 3: DBMS Managed Pages

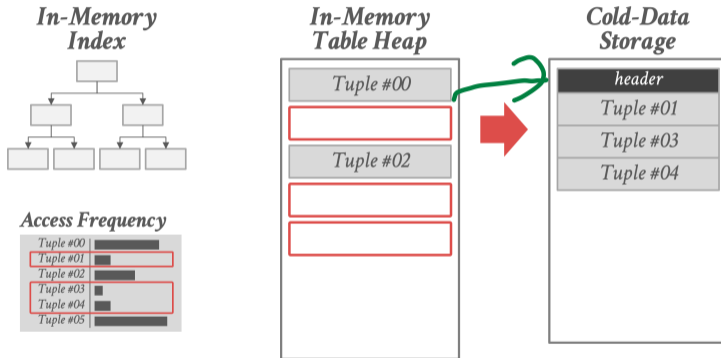
- ▶ DBMS tracks what data is in memory vs. on disk.

• Choice 4: OS Virtual Memory

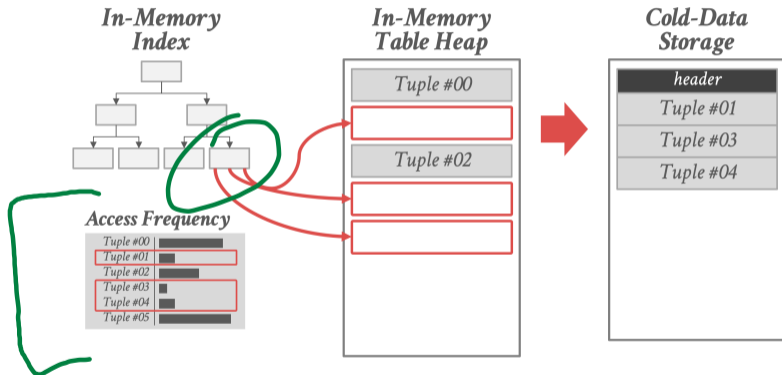
- ▶ OS tracks what data is on in memory vs. on disk.

(BTW, ~)

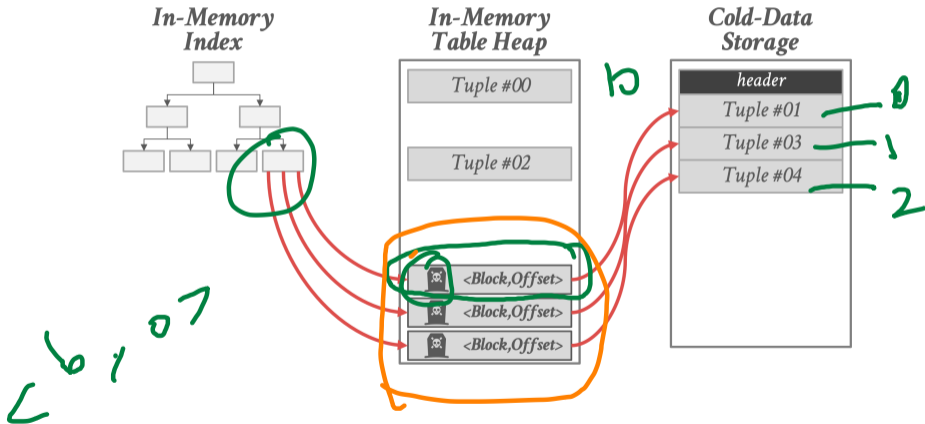
Evicted Tuple Metadata



Evicted Tuple Metadata



Evicted Tuple Metadata



Data Retrieval Granularity

Choice 1: All Tuples in Block

- ▶ Merge all the tuples retrieved from a block regardless of whether they are needed.
- ▶ More CPU overhead to update indexes.
- ▶ Tuples are likely to be evicted again.

Choice 2: Only Tuples Needed

- ▶ Only merge the tuples that were accessed by a query back into the in-memory table heap.
- ▶ Requires additional bookkeeping to track holes.

Merging Threshold = k

$k=1$

- Choice 1: Always Merge

- ▶ Retrieved tuples are always put into table heap.

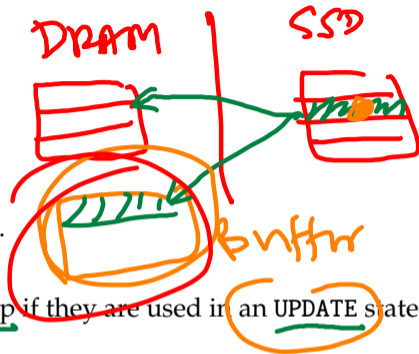
- Choice 2: Merge Only on Update

- ▶ Retrieved tuples are only merged into table heap if they are used in an UPDATE statement.
- ▶ All other tuples are put in a temporary buffer.

- Choice 3: Selective Merge

- ▶ Keep track of how often each block is retrieved.
- ▶ If a block's access frequency is above some threshold, merge it back into the table heap.

$k=10$



Retrieval Mechanism

• Choice 1: Abort-and-Restart

- ▶ Abort the txn that accessed the evicted tuple.
- ▶ Retrieve the data from disk and merge it into memory with a separate background thread.
- ▶ Restart the txn when the data is ready.
- ▶ Requires MVCC to guarantee consistency for large txns that access data that does not fit in memory.

• Choice 2: Synchronous Retrieval

- ▶ Stall the txn when it accesses an evicted tuple while the DBMS fetches the data and merges it back into memory.

100D 2 ops

N VM 100 ops

Case Studies

Case Studies

- **Tuple-Oriented Systems**

- ▶ H-Store – Anti-Caching
- ▶ Hekaton – Project Siberia
- ▶ EPFL's VoltDB Prototype
- ▶ Apache Geode – Overflow Tables

- **Block-Oriented Systems**

- ▶ LeanStore – Hierarchical Buffer Pool
- ▶ Umbra – Variable-length Buffer Pool
- ▶ MemSQL – Columnar Tables

H-Store – Anti-Caching

IMDB

- Cold Tuple Identification: On-line Identification
- Eviction Timing: Administrator-defined Threshold
- Evicted Tuple Metadata: Tombstones
- Retrieval Mechanism: Abort-and-restart Retrieval
- Retrieval Granularity: Block-level Granularity
- Merging Threshold: Always Merge
- Reference

HEKATON – PROJECT SIBERIA

MSR

In-memory
engine

- Cold Tuple Identification: Off-line Identification
- Eviction Timing: Administrator-defined Threshold
- Evicted Tuple Metadata: Bloom Filters
- Retrieval Mechanism: Synchronous Retrieval
- Retrieval Granularity: Tuple-level Granularity
- Merging Threshold: Always Merge
- Reference

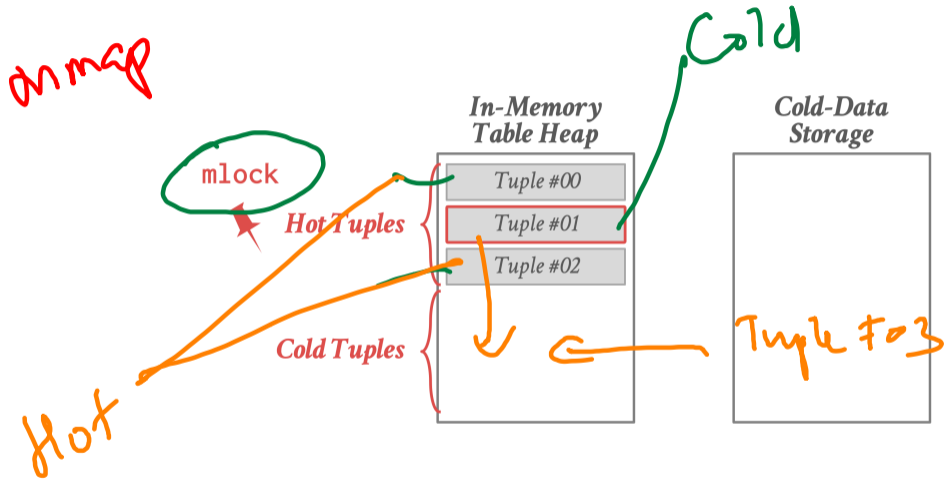
SQL Feeder

EPFL VOLTDB - H-Store

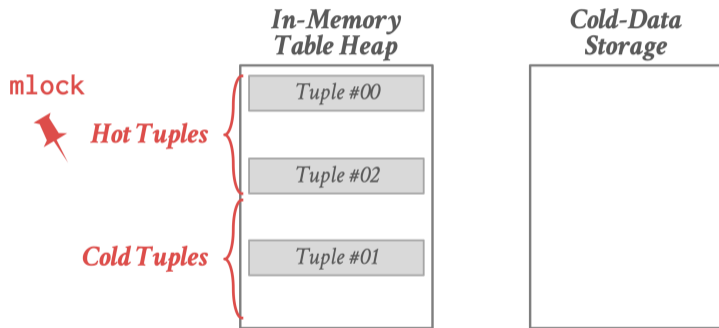
OS

- Cold Tuple Identification: Off-line Identification
- Eviction Timing: OS Virtual Memory
- Evicted Tuple Metadata: N/A
- Retrieval Mechanism: Synchronous Retrieval
- Retrieval Granularity: Page-level Granularity
- Merging Threshold: Always Merge
- Reference

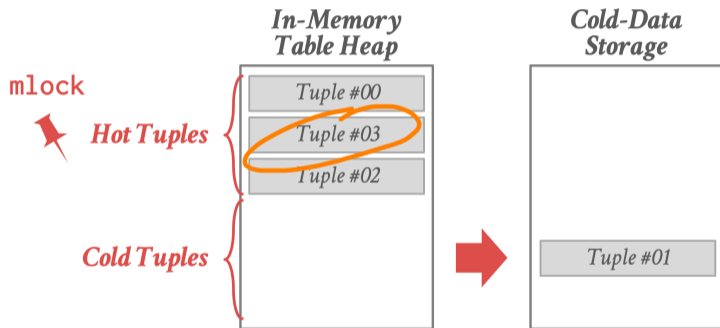
EPFL VOLTDB



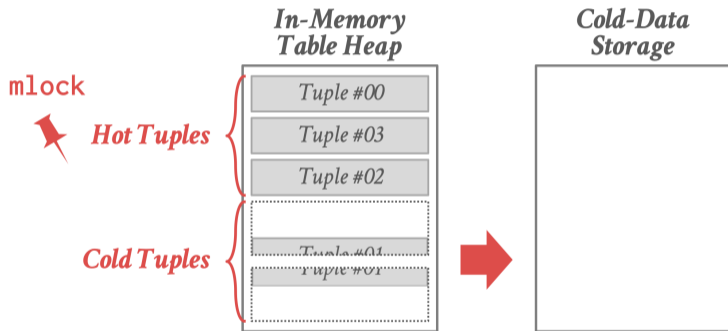
EPFL VOLTDB



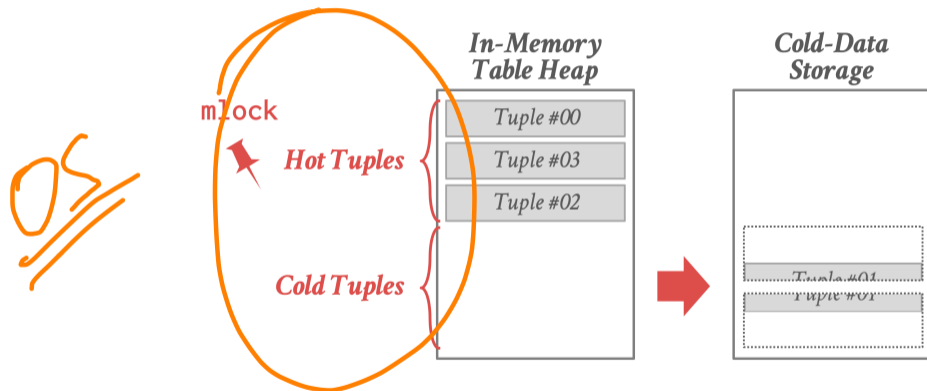
EPFL VOLTDB



EPFL VOLTDB



EPFL VOLTDB



APACHE GEODE – OVERFLOW TABLES

Genfize

- Cold Tuple Identification: On-line Identification
- Eviction Timing: Administrator-defined Threshold
- Evicted Tuple Metadata: Tombstones (?)
- Retrieval Mechanism: Synchronous Retrieval
- Retrieval Granularity: Tuple-level Granularity
- Merging Threshold: Merge Only on Update (?)
- Reference

Observation

memory footprint

- The systems that we have discussed so far are **tuple-oriented**.
 - ▶ The DBMS must track meta-data about individual tuples.
 - ▶ Does not reduce ~~storage~~ overhead of indexes.
 - ▶ Indexes may occupy up to 60% of DRAM in an OLTP database.
- **Goal:** Need an unified way to evict cold data from both tables and indexes with low overhead...

LeanStore

TUM Munich (HyPer)

- In-memory storage manager from TUM that supports larger-than-memory databases.
 - ▶ Handles both tuples + indexes
 - ▶ Not part of the HyPer project.
- Hierarchical + Randomized Block Eviction
 - ▶ Use pointer swizzling to determine whether a block is evicted or not.
 - ▶ Instead of tracking when pages are accessed, randomly evict pages and then track whether they ended up getting used.
 - ▶ If yes, put it back in the hot space.
 - ▶ If not, then evict it.

• Reference

ICDE '2017

2Q policy

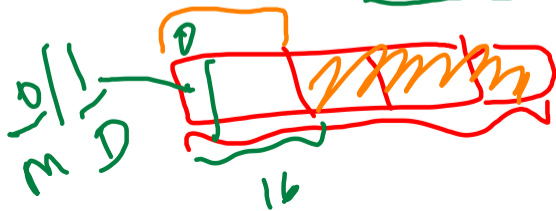
Pointer Swizzling

Centralized: ~~Hash Table~~
~~Bloom filter~~

- Switch the contents of pointers based on whether the target object resides in memory or on disk.

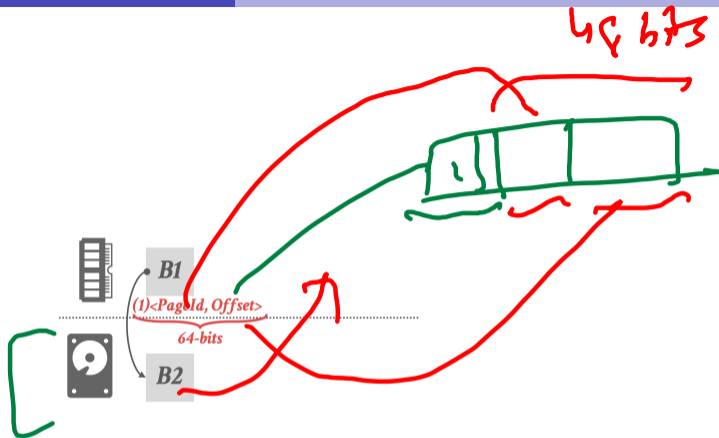
→ Decentralized way to track whether a page is in memory or not.

- We track everything with 64-bit pointers, but currently only use 48-bits.
 - Use first bit in address to tell what kind of address it is.
 - Only works if there is only one pointer to the object.

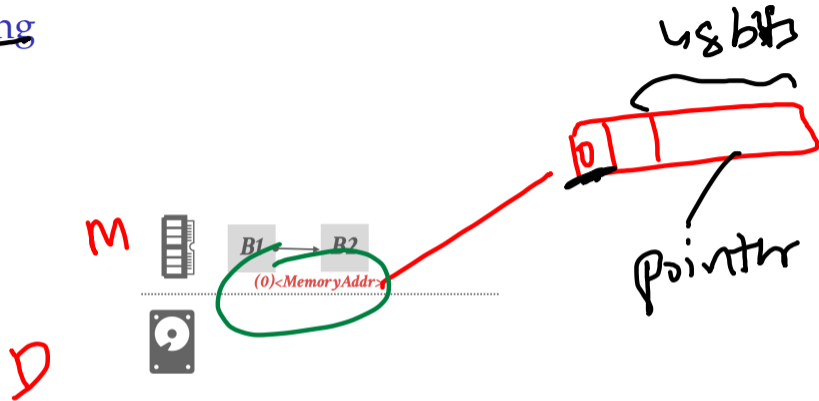


x86

Pointer Swizzling



Pointer Swizzling

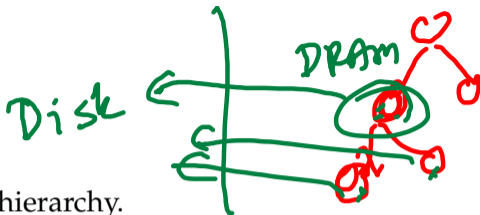


Replacement Strategy

- Randomly select blocks for eviction.
 - ▶ Don't have to maintain meta-data every time a txn accesses a hot block.
 - ▶ Only track accesses for cold data, which should be rare if it is cold.
- Unswizzle their pointer but leave in memory.
 - ▶ Add to a FIFO queue of blocks staged for eviction.
 - ▶ If page is accessed again, remove from queue.
 - ▶ Otherwise, evict pages when reaching front of queue.

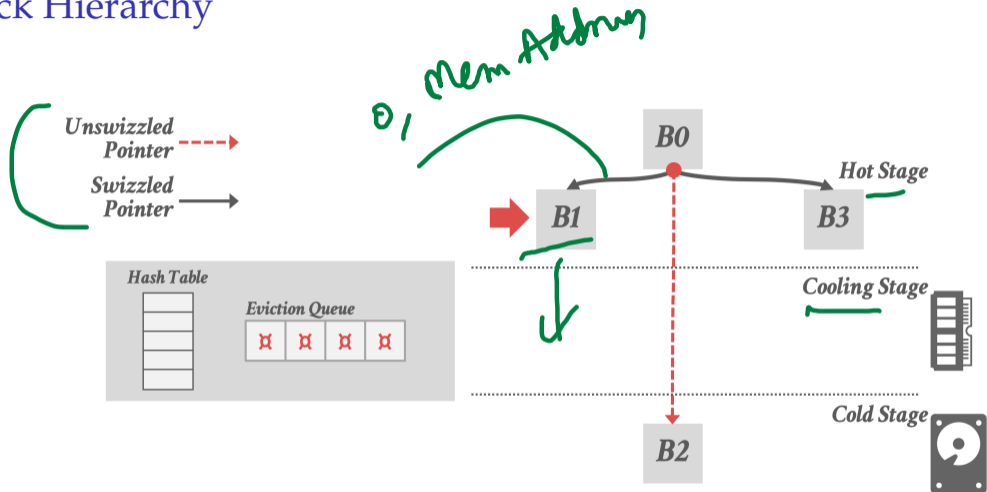


Block Hierarchy

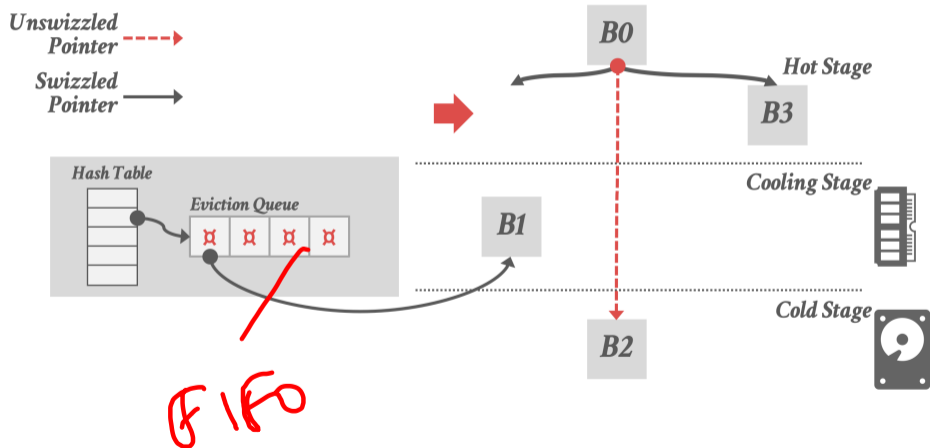


- Blocks are organized in a tree hierarchy.
 - ▶ Each page has only one parent, which means that there is only a single pointer.
 - ▶ No centralized page table (as is the case in a disk-oriented DBMS).
- The DBMS can only evict a block if its children are also evicted.
 - ▶ This avoids the problem of evicting blocks that contain swizzled pointers
 - ▶ Otherwise, these pointers are invalid because they will point to old locations in memory.
 - ▶ If a block is selected but it has in-memory children, then it automatically switches to select one of its children.

Block Hierarchy



Block Hierarchy

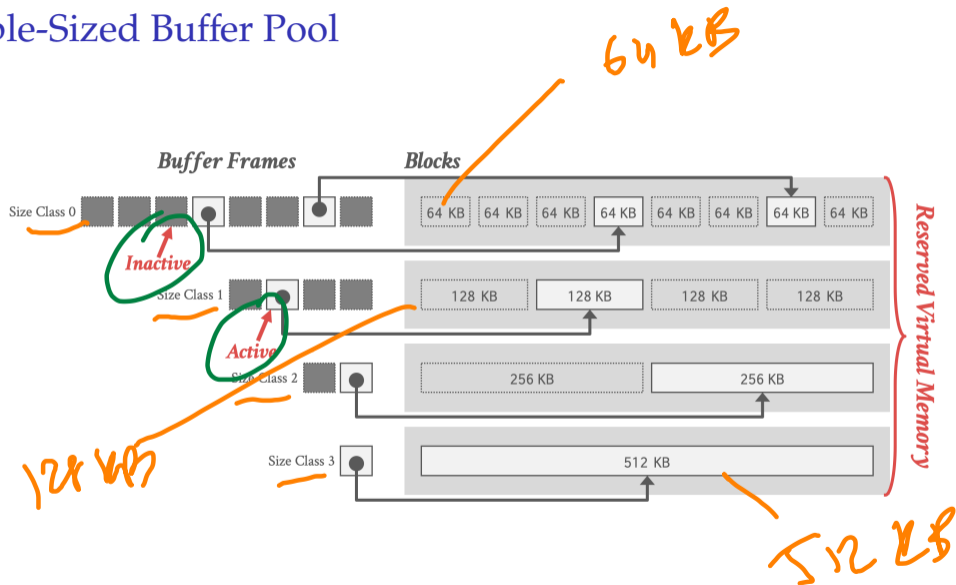


Umbra

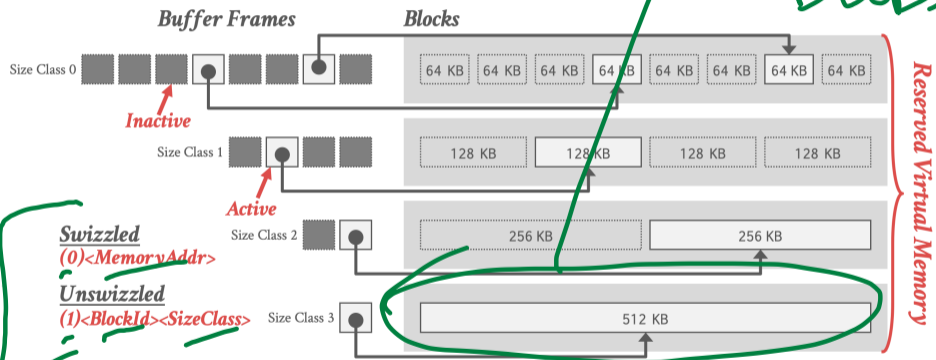
- New DBMS from HyPer team at TUM.
 - ▶ Low overhead buffer pool with variable-sized pages.
 - ▶ Employs the same hierarchical organization and randomized block eviction algorithm from LeanStore.
 - ▶ Uses virtual memory to allocate storage but the DBMS manages block eviction on its own.
- DBMS stores relations as index-organized tables, so there is no separate management needed to handle index blocks.
- **Reference**

CIDR'20

Variable-Sized Buffer Pool



Variable-Sized Buffer Pool



MEMSQL – Columnar Tables

OLAP

- Administrator manually declares a table as a disk-resident columnar table with zone maps.
 - ▶ Pre-2017: Used mmap but this was a bad idea.
 - ▶ Current: Unified single logical table format that combines mutable delta store with immutable column store.
- Evicted Tuple Metadata: None
- Retrieval Mechanism: Synchronous Retrieval
- Merging Threshold: Always Merge
- Reference

Conclusion

SSD
HDD

- Today we focused on working around the block-oriented access granularity and lower bandwidth of secondary storage.
- We will learn about how recently-released byte-addressable, non-volatile memory (2019) changes the hardware landscape.

Optane DIMMs
Optane SSD

64B

NVM

References I