



Lecture #23

Carnegie Mellon University

# ADVANCED DATABASE SYSTEMS

Larger-than-Memory Databases

@Andy\_Pavlo // 15-721 // Spring 2018

# ADMINISTRIVIA

---

Snowflake Guest Speaker: May 2<sup>nd</sup>

Final Exam Handout: May 2<sup>nd</sup>

Code Review #2: May 2<sup>nd</sup>

Project #3 Final Presentation: May 14<sup>th</sup> @ 8:30am

# FoundationDB is Open Source

Published April 19, 2018

## The next chapter

Starting today, FoundationDB starts its next chapter as an open source project!

FoundationDB is a distributed datastore, designed from the ground up to be deployed on clusters of commodity hardware. These clusters scale well as you add machines, automatically heal from hardware failures, and have a simple API. The key-value store supports fully global, cross-row ACID transactions. That's the highest level of data consistency possible. What does this mean for you? Strong consistency makes your application code simpler, your data models more efficient, and your failure modes less surprising.

The great thing is that FoundationDB is already well-established — it's actively developed and has years of production use. We intend to drive FoundationDB forward as a community project and we welcome your participation.



# FoundationDB is

Published April 19, 2018

## The next chapter

Starting today, FoundationDB starts its

FoundationDB is a distributed datastore. These clusters scale well as you add more nodes. The value store supports fully global, cross-replica reads. Does this mean for you? Strong consistency, failure modes less surprising.

The great thing is that FoundationDB is open source. We intend to drive FoundationDB forward.

Documentation [Blog](#) [Download](#)

ORACLE

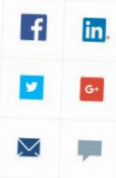
The Oracle MySQL Blog

Try Oracle Cloud for Free

Q MENU



Share



MYSQL, THE ORACLE MYSQL | Thursday, April 19, 2018

## Announcing General Availability of MySQL 8.0

By: [Mike Frank](#) | Product Management Director

MySQL adds NoSQL and many new enhancements to the world's most popular open source database:

1. **NoSQL** Document Store gives developers the flexibility of developing traditional SQL relational applications and NoSQL, schema-free document database applications. This eliminates the need for a separate NoSQL document database.
2. **SQL** Window functions, Common Table Expressions, NOWAIT and SKIP LOCKED, Descending Indexes, Grouping, Regular Expressions, Character Sets, Cost Model, and Histograms.
3. **JSON** Extended syntax, new functions, improved sorting, and partial updates. With JSON table functions you can use the SQL machinery for JSON data.
4. **GIS** Geography support. Spatial Reference Systems (SRS), as well as SRS aware spatial datatypes, spatial indexes, and spatial functions.
5. **Reliability** DDL statements have become atomic and crash safe, meta-data is stored in a single, transactional data dictionary
6. **Observability** Performance Schema, Information Schema, Invisible Indexes, Error Logging.

ORACLE

The Oracle MySQL Blog

Try Oracle Cloud for Free


  MENU


## MySQL 8.0

source database:

ational applications and  
separate NoSQL

ending Indexes,

table functions you

datatypes, spatial

single, transactional

ation Schema, Invisible Indexes, Error Logging,

# TODAY'S AGENDA

---

Background

Implementation Issues

Real-world Examples

Evaluation



# MOTIVATION

---

DRAM is expensive, son.

It would be nice if our in-memory DBMS could use cheaper storage.



# 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.

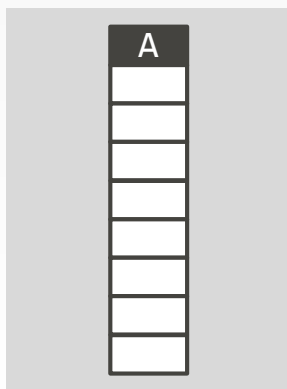
Need to be aware of hardware access methods

- In-memory Storage = Tuple-Oriented
- Disk Storage = Block-Oriented

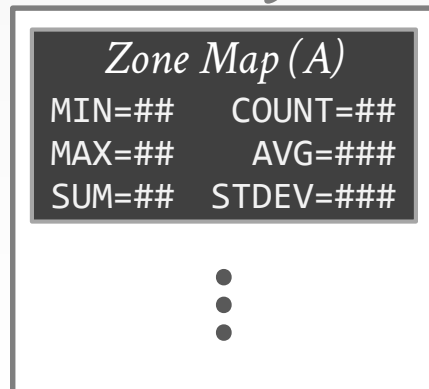


# OLAP

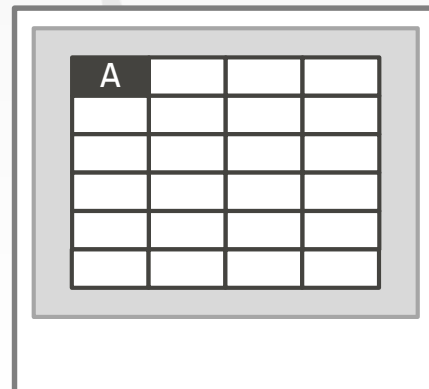
OLAP queries generally access the entire table.  
Thus, there isn't anything about the workload for the DBMS to exploit that a disk-oriented buffer pool can't handle.



## *In-Memory*



## *Disk Data*



# OLTP

---

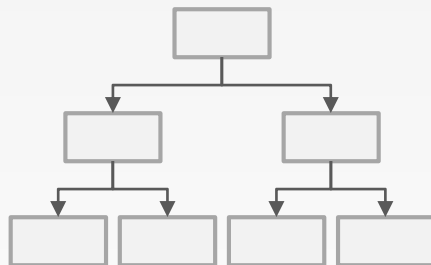
OLTP workloads almost always have hot and cold portions of the database.

→ We can assume that txns will almost always access hot tuples.

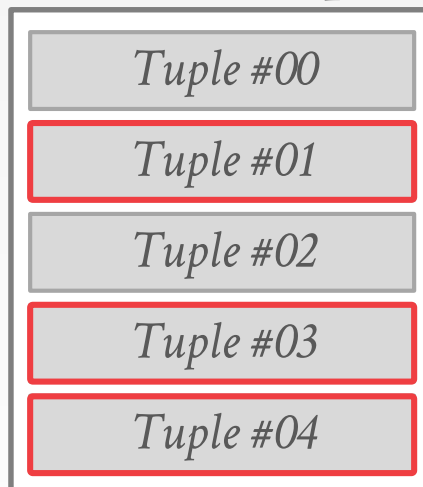
The DBMS needs a mechanism to move cold data out to disk and then retrieve it if it is ever needed again.

# LARGER-THAN-MEMORY DATABASES

*In-Memory  
Index*



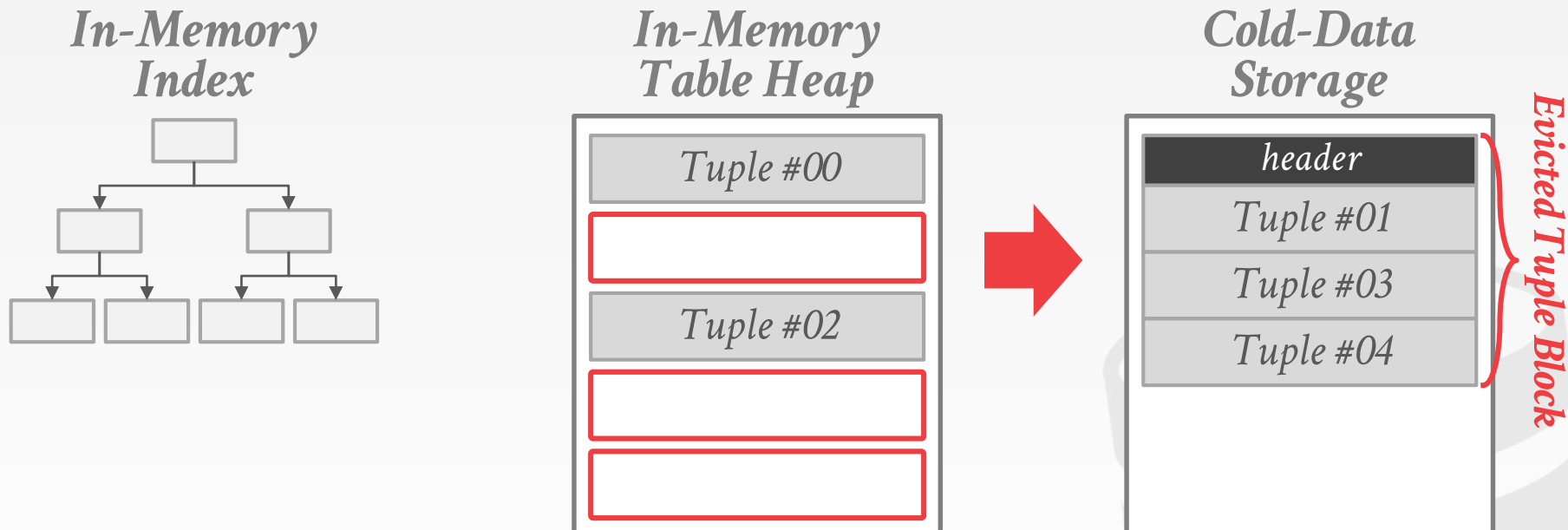
*In-Memory  
Table Heap*



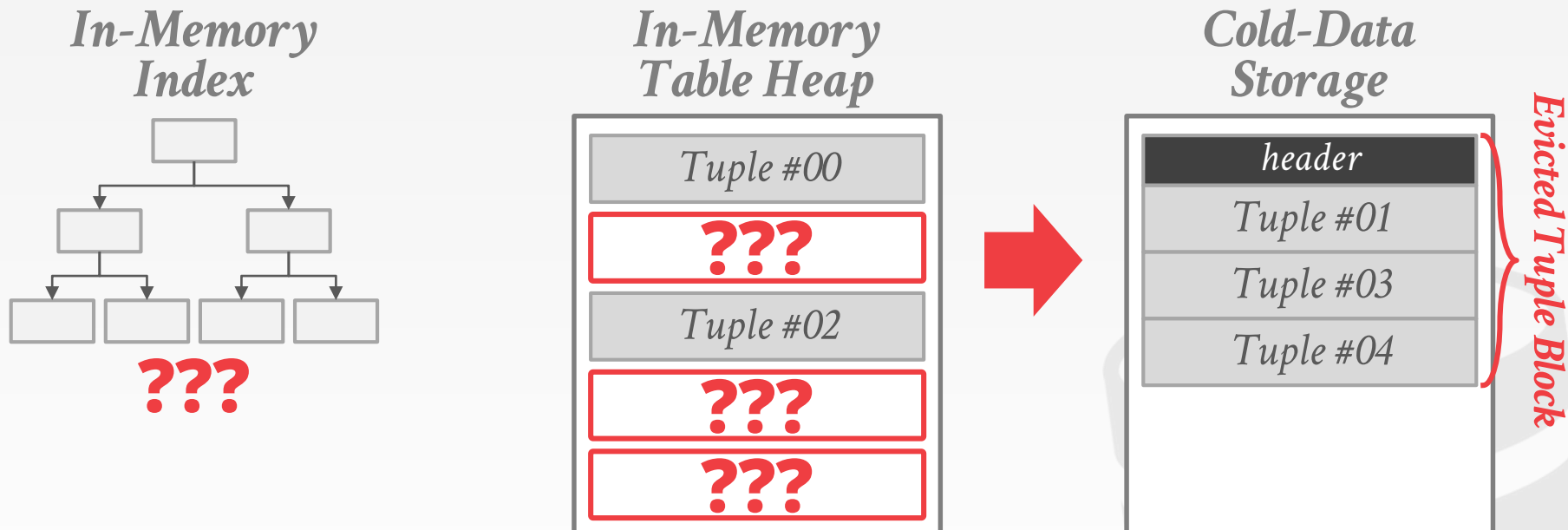
*Cold-Data  
Storage*



# LARGER-THAN-MEMORY DATABASES

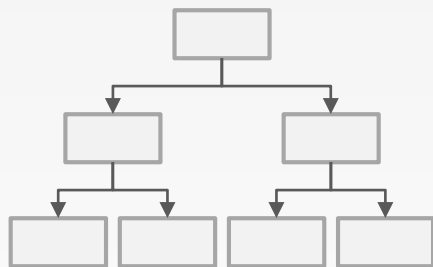


# LARGER-THAN-MEMORY DATABASES



# LARGER-THAN-MEMORY DATABASES

## In-Memory Index

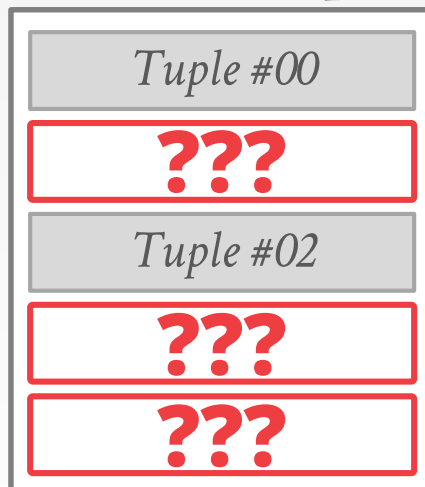


???

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

???

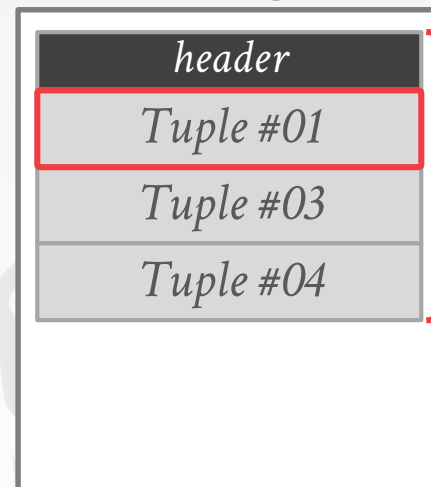
## In-Memory Table Heap



???



## Cold-Data Storage



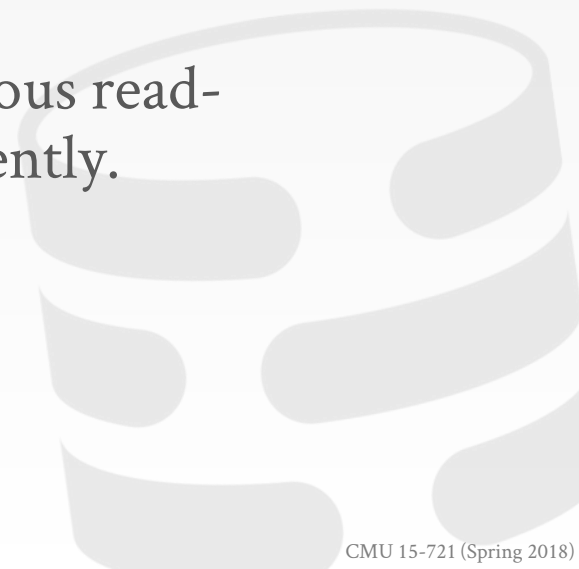
Evicted Tuple Block

## AGAIN, WHY NOT MMAP?

---

Write-ahead logging requires that a modified page cannot be written to disk before the log records that made those changes is written.

There are no mechanisms for asynchronous read-ahead or writing multiple pages concurrently.



IN-MEMORY PERFORMANCE FOR BIG DATA  
VLDB 2014

# OLTP ISSUES

---

## **Run-time Operations**

→ Cold Tuple Identification

## **Eviction Policies**

→ Timing

→ Evicted Tuple Metadata

## **Data Retrieval Policies**

→ Granularity

→ Retrieval Mechanism

→ Merging back to memory





# COLD TUPLE IDENTIFICATION

---

## Choice #1: On-line

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

## 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 has to manually move data.

## Choice #2: OS Virtual Memory

- The OS decides when it wants to move data out to disk. This is done in the background.

# EVICTED TUPLE METADATA

---

## Choice #1: Tombstones

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

## Choice #2: Bloom Filters

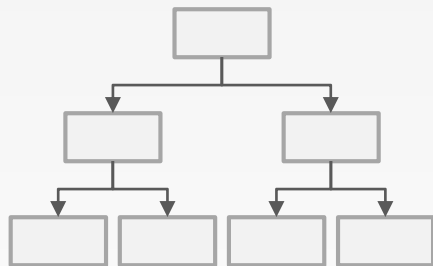
- Use approximate data structure for each index.
- Check both index + filter for each query.

## Choice #3: OS Virtual Memory

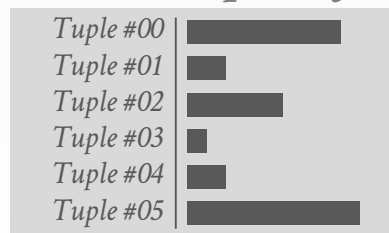
- The OS tracks what data is on disk. The DBMS does not need to maintain any additional metadata.

# EVICTED TUPLE METADATA

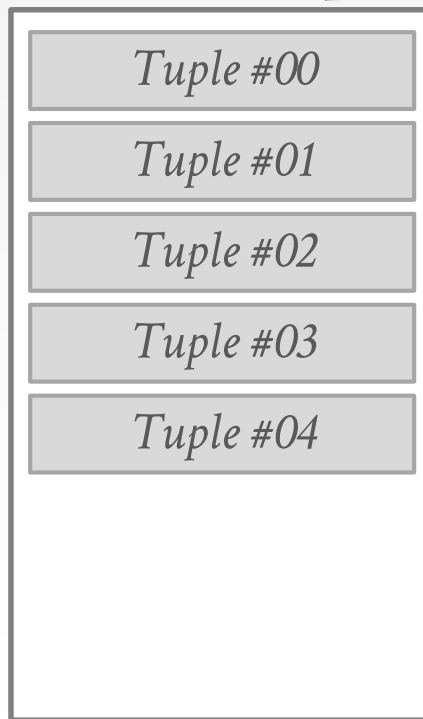
## *In-Memory Index*



## *Access Frequency*



## *In-Memory Table Heap*

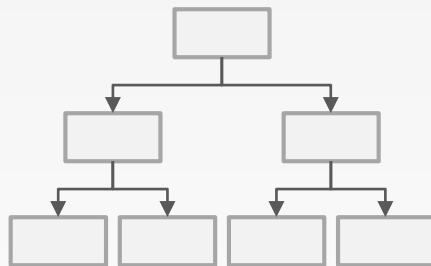


## *Cold-Data Storage*

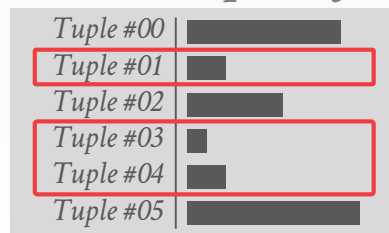


# EVICTED TUPLE METADATA

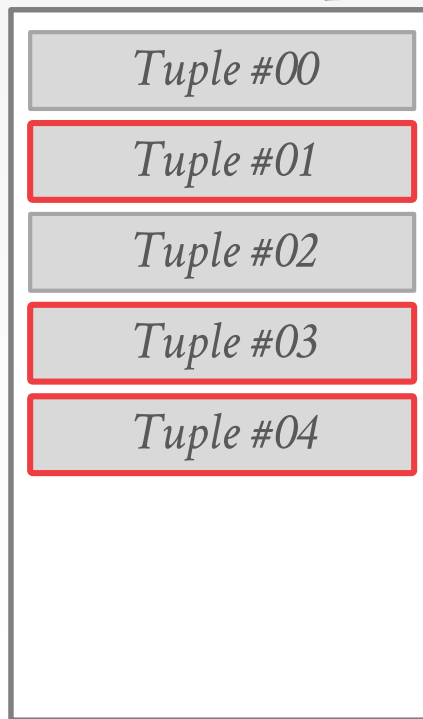
## *In-Memory Index*



## *Access Frequency*



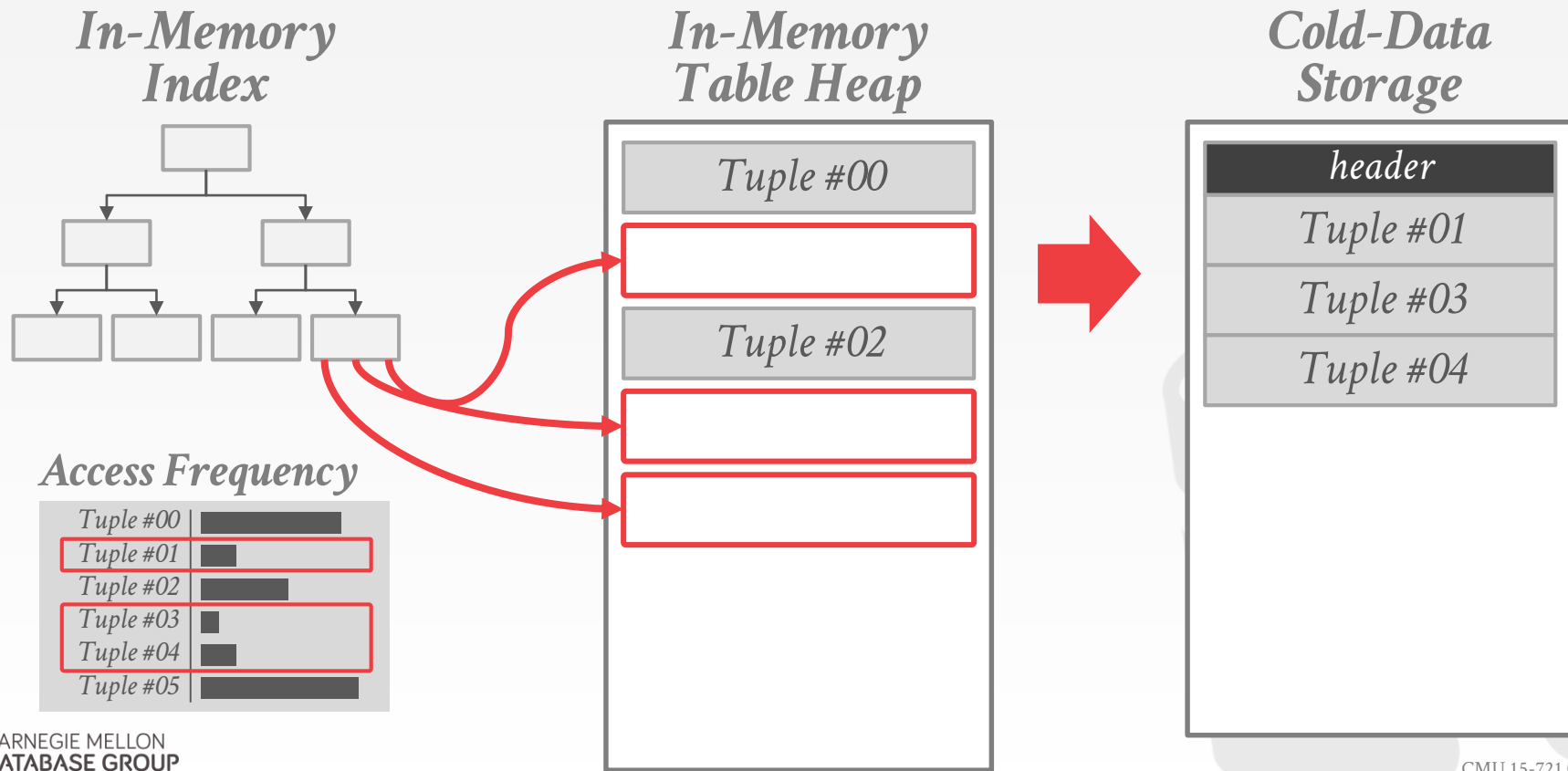
## *In-Memory Table Heap*



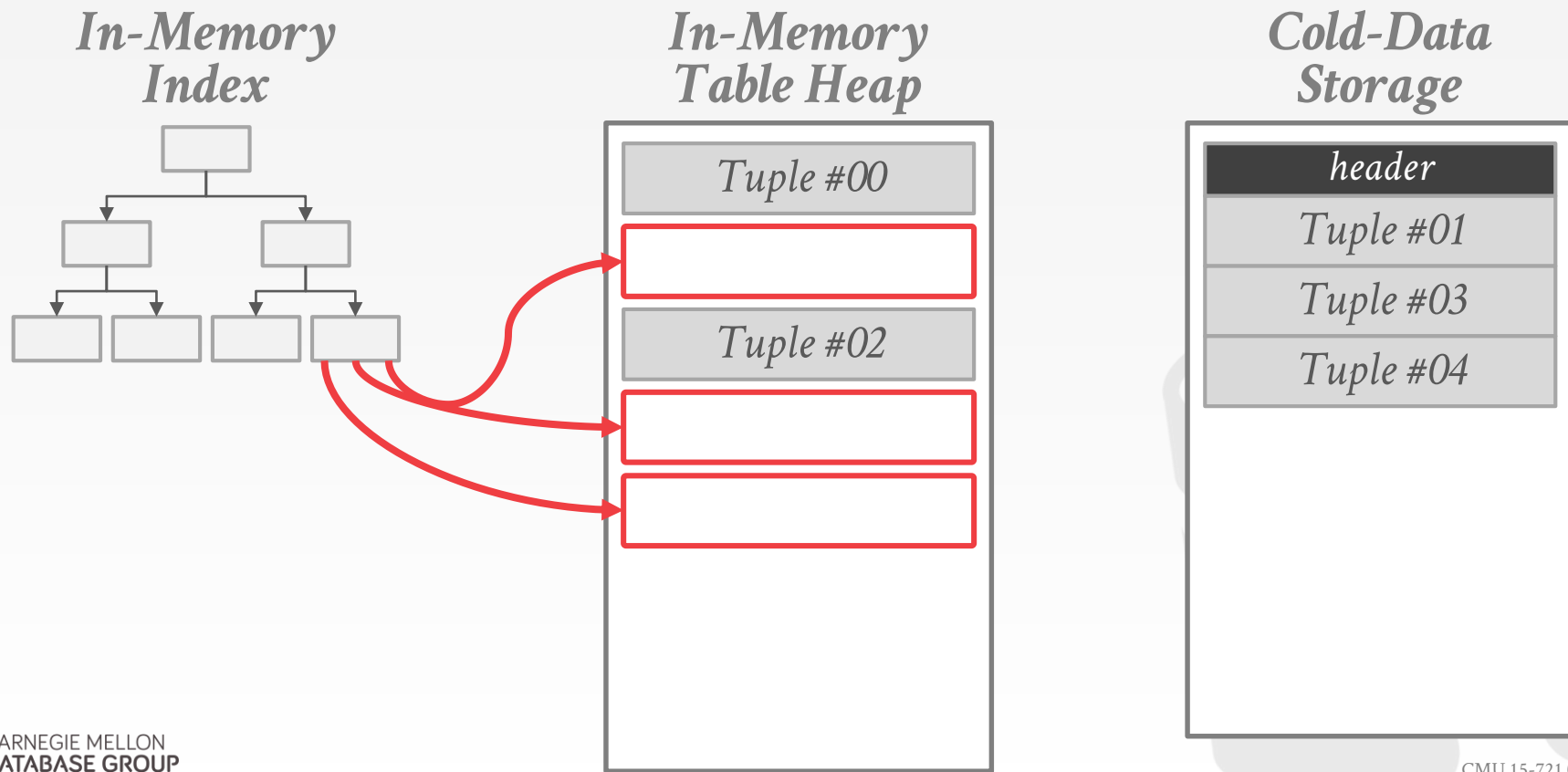
## *Cold-Data Storage*



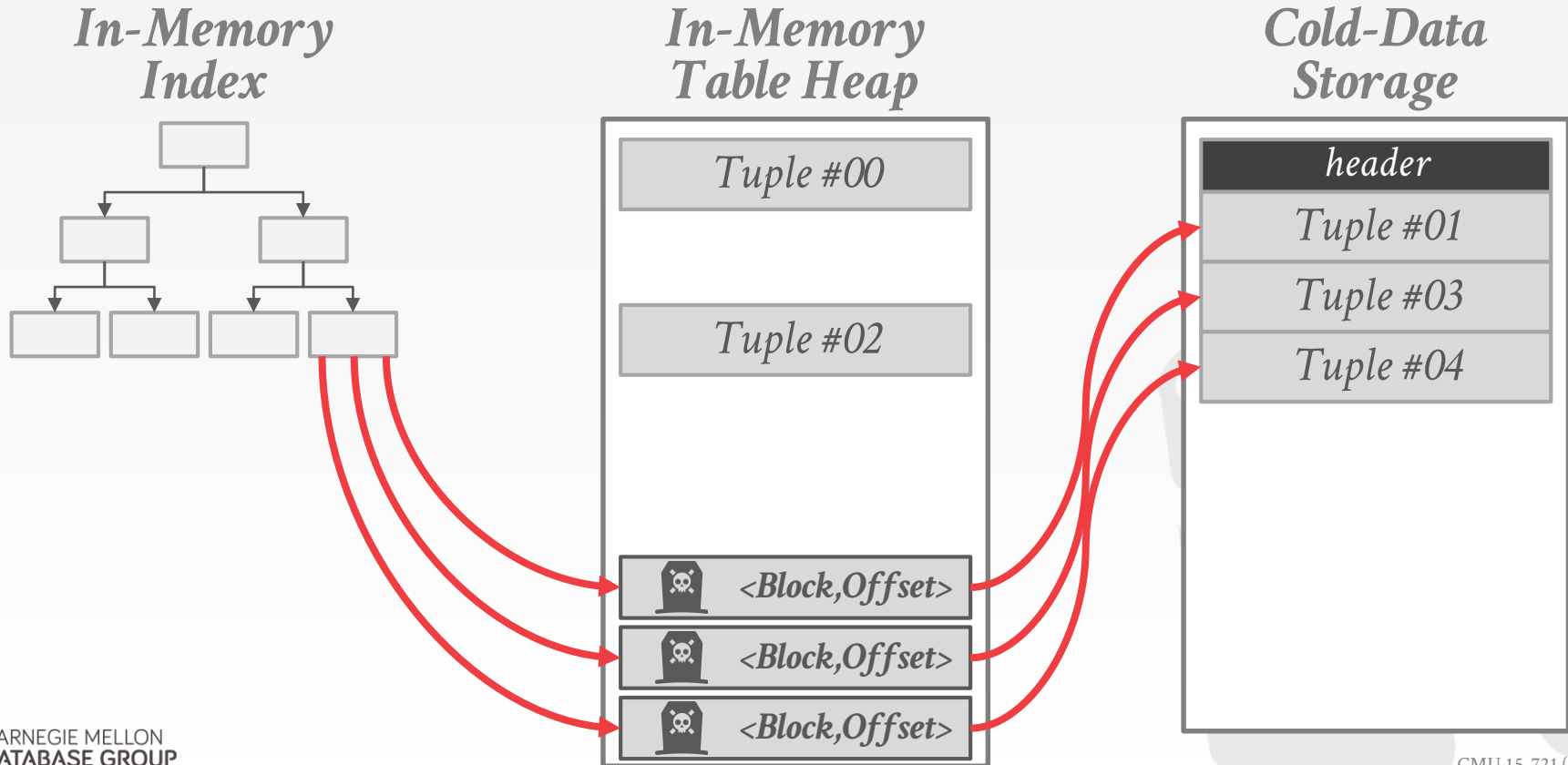
# EVICTED TUPLE METADATA



# EVICTED TUPLE METADATA



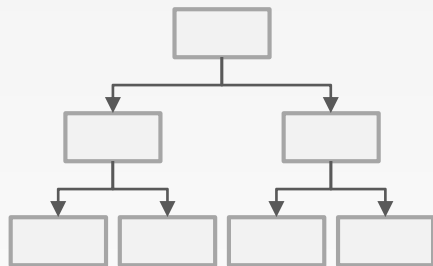
# EVICTED TUPLE METADATA



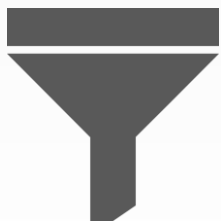


# EVICTED TUPLE METADATA

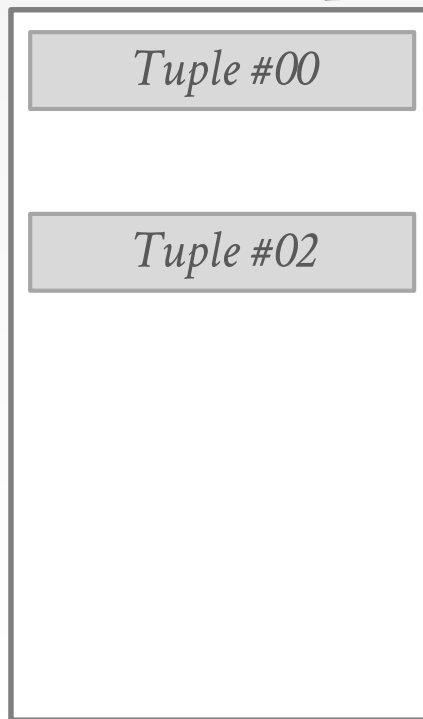
*In-Memory  
Index*



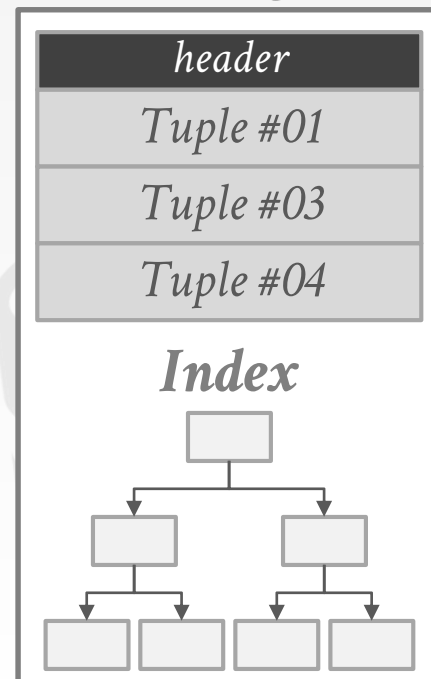
*Bloom Filter*



*In-Memory  
Table Heap*



*Cold-Data  
Storage*



# DATA RETRIEVAL GRANULARITY

---

## **Choice #1: 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.

## **Choice #2: 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.

# 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.
- Cannot guarantee consistency for large queries.

## **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.

# MERGING THRESHOLD

---

## 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** query.

→ 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.

# REAL-WORLD IMPLEMENTATIONS

---

H-Store – Anti-Caching

Hekaton – Project Siberia

EPFL's VoltDB Prototype

Apache Geode – Overflow Tables

MemSQL – Columnar Tables



# H-STORE – ANTI-CACHING

---

On-line Identification

Administrator-defined Threshold

Tombstones

Abort-and-restart Retrieval

Block-level Granularity

Always Merge



ANTI-CACHING: A NEW APPROACH TO DATABASE  
MANAGEMENT SYSTEM ARCHITECTURE  
VLDB 2013

# HEKATON – PROJECT SIBERIA

---

Off-line Identification

Administrator-defined Threshold

Bloom Filters

Synchronous Retrieval

Tuple-level Granularity

Always Merge



# EPFL VOLTDB

---

Off-line Identification

OS Virtual Memory

Synchronous Retrieval

Page-level Granularity

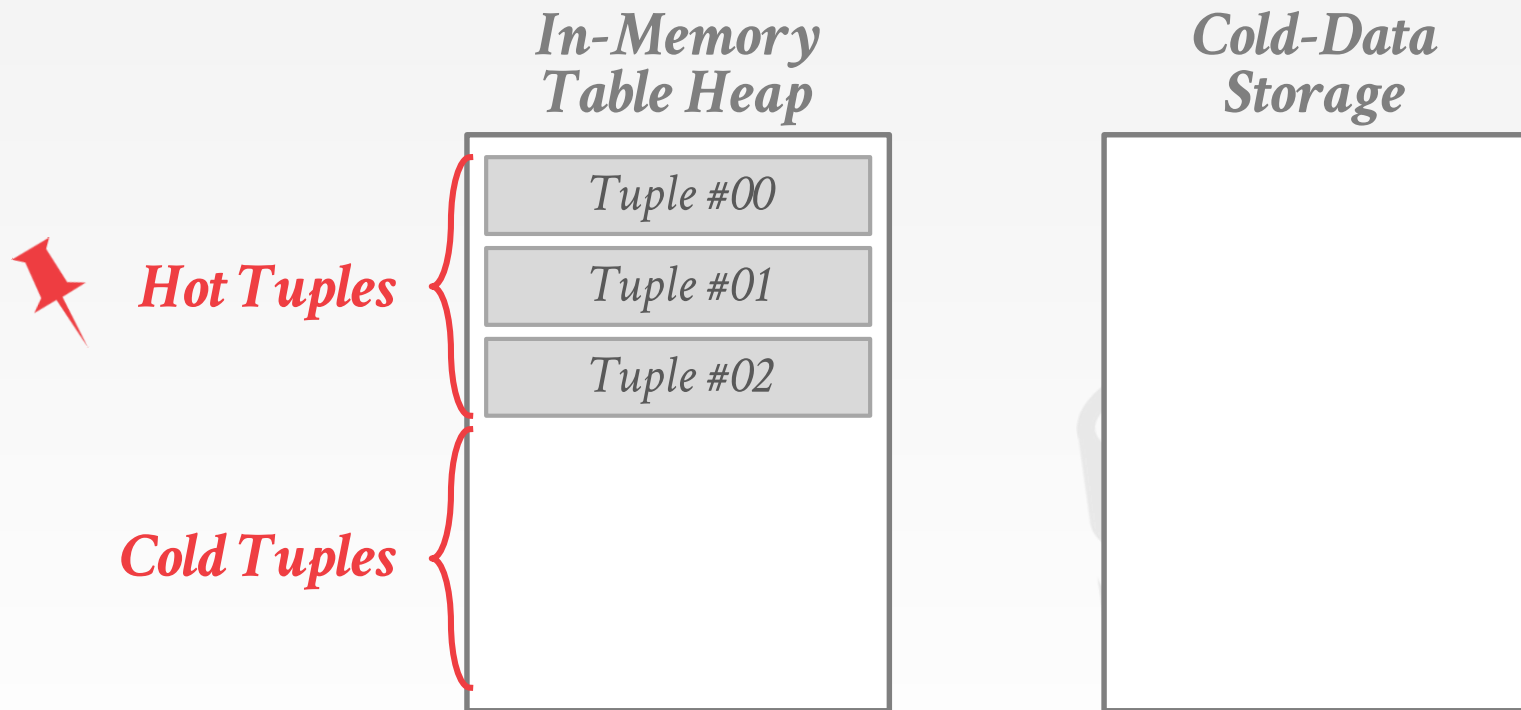
Always Merge



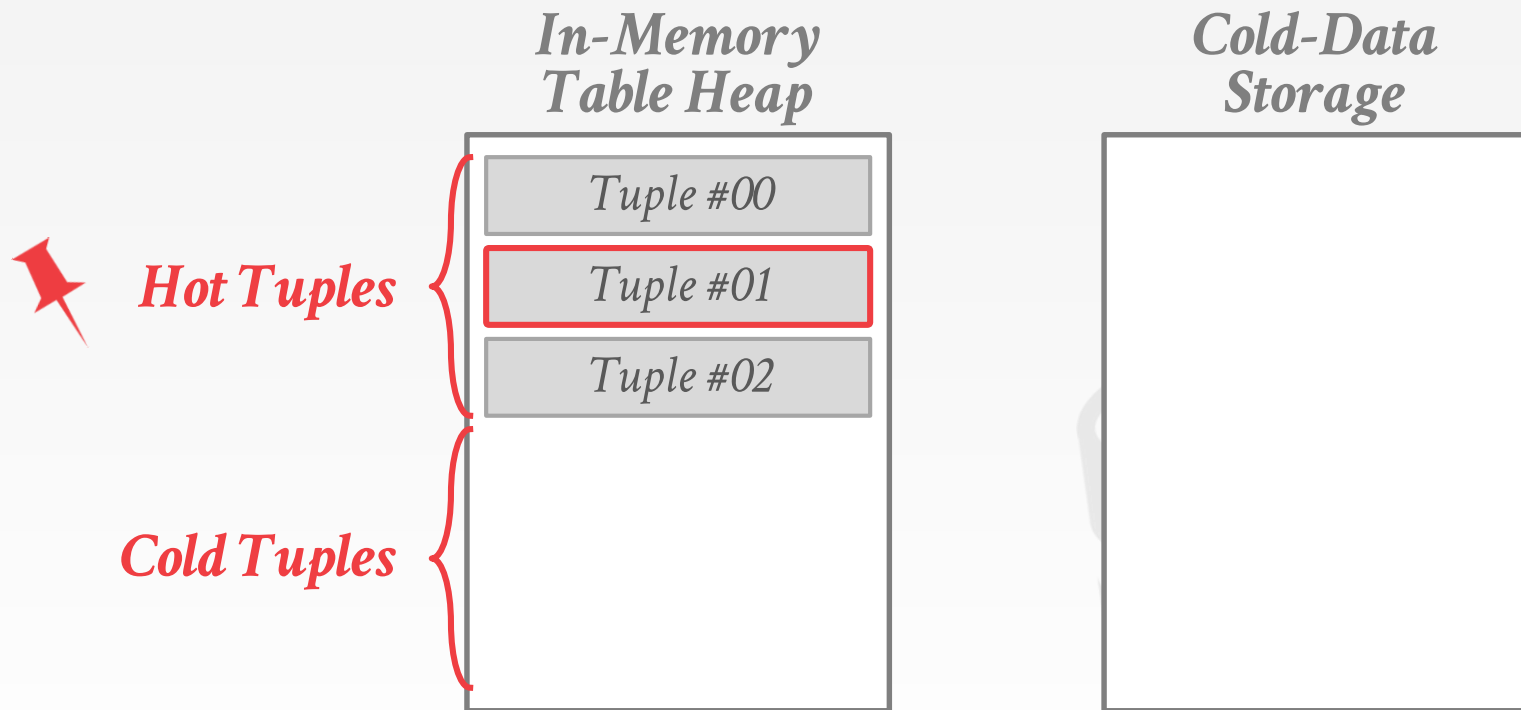
ENABLING EFFICIENT OS PAGING FOR MAIN-MEMORY OLTP DATABASES  
DAMON 2013



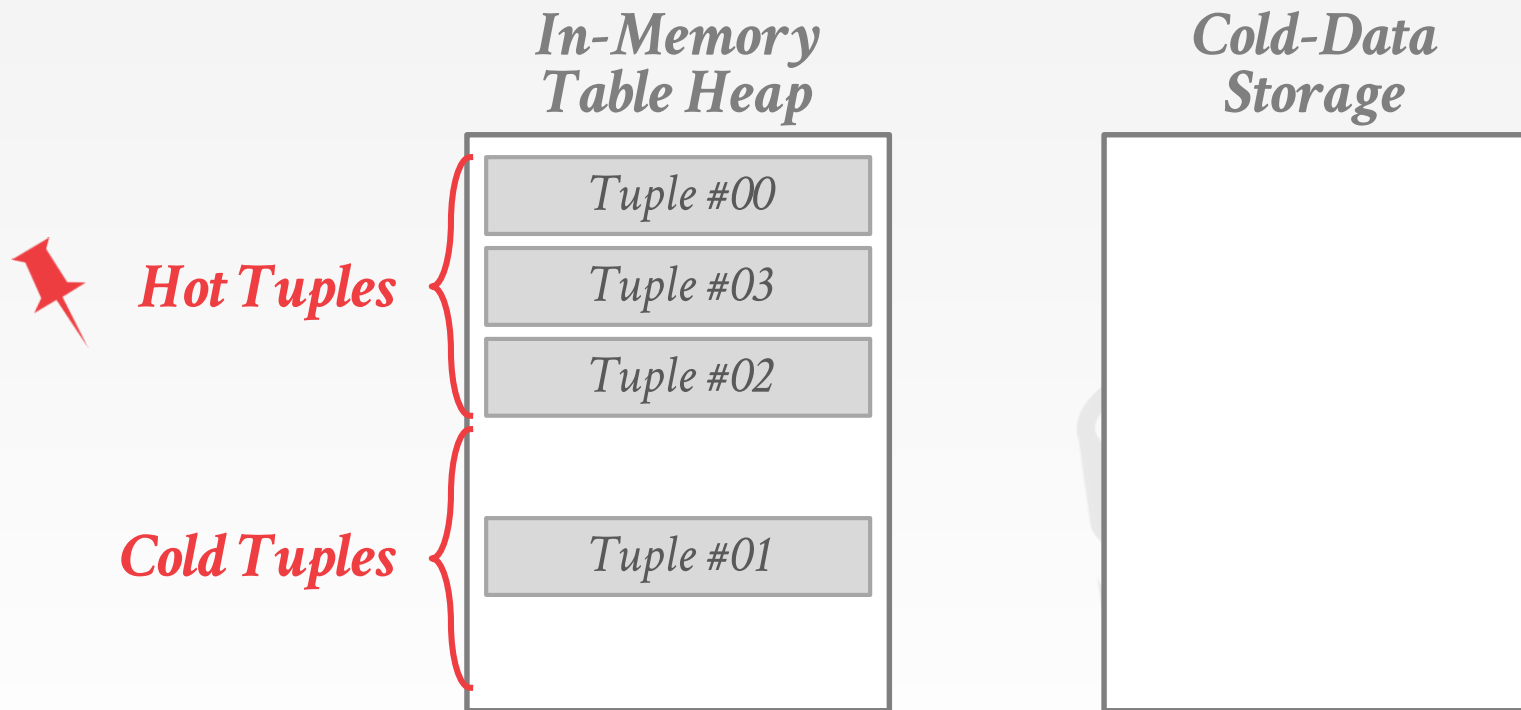
# EPFL VOLTDB



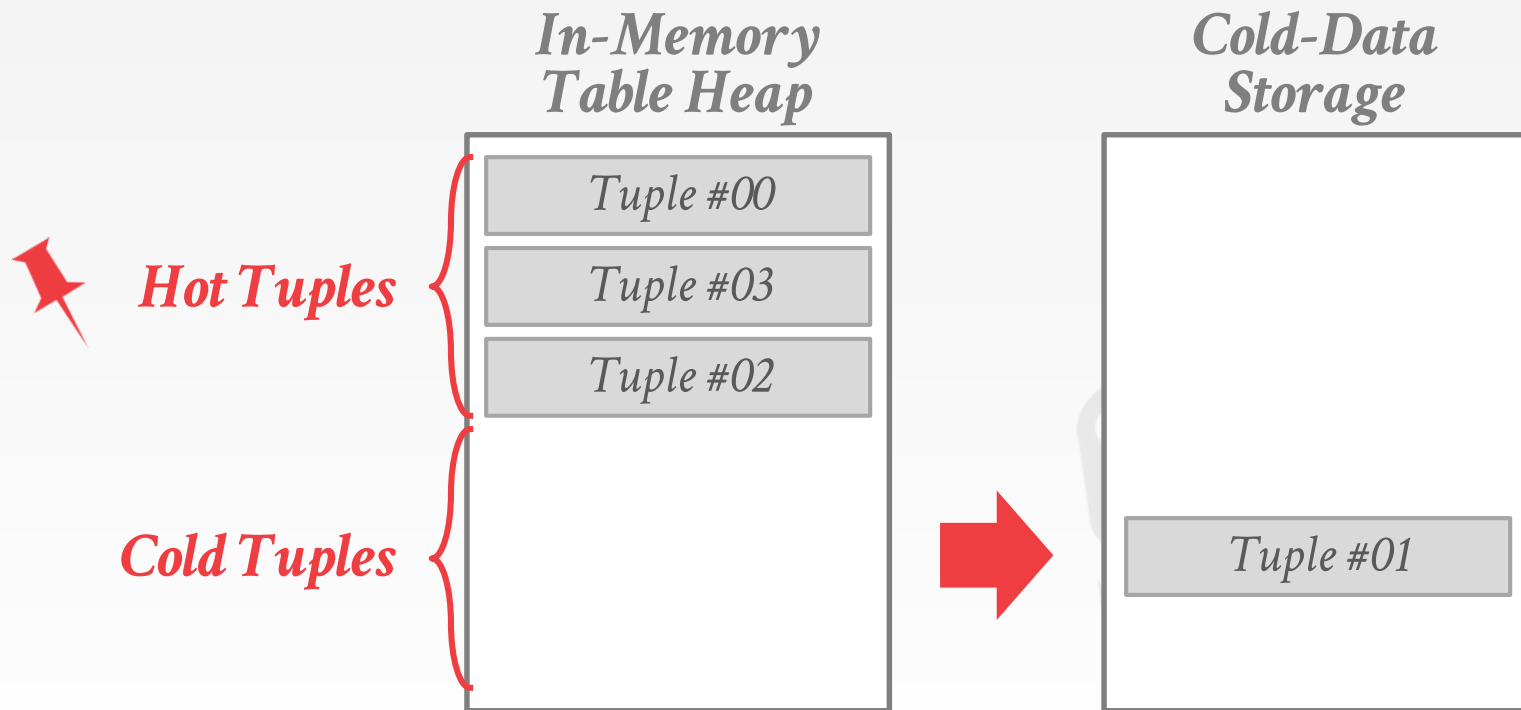
# EPFL VOLTDB



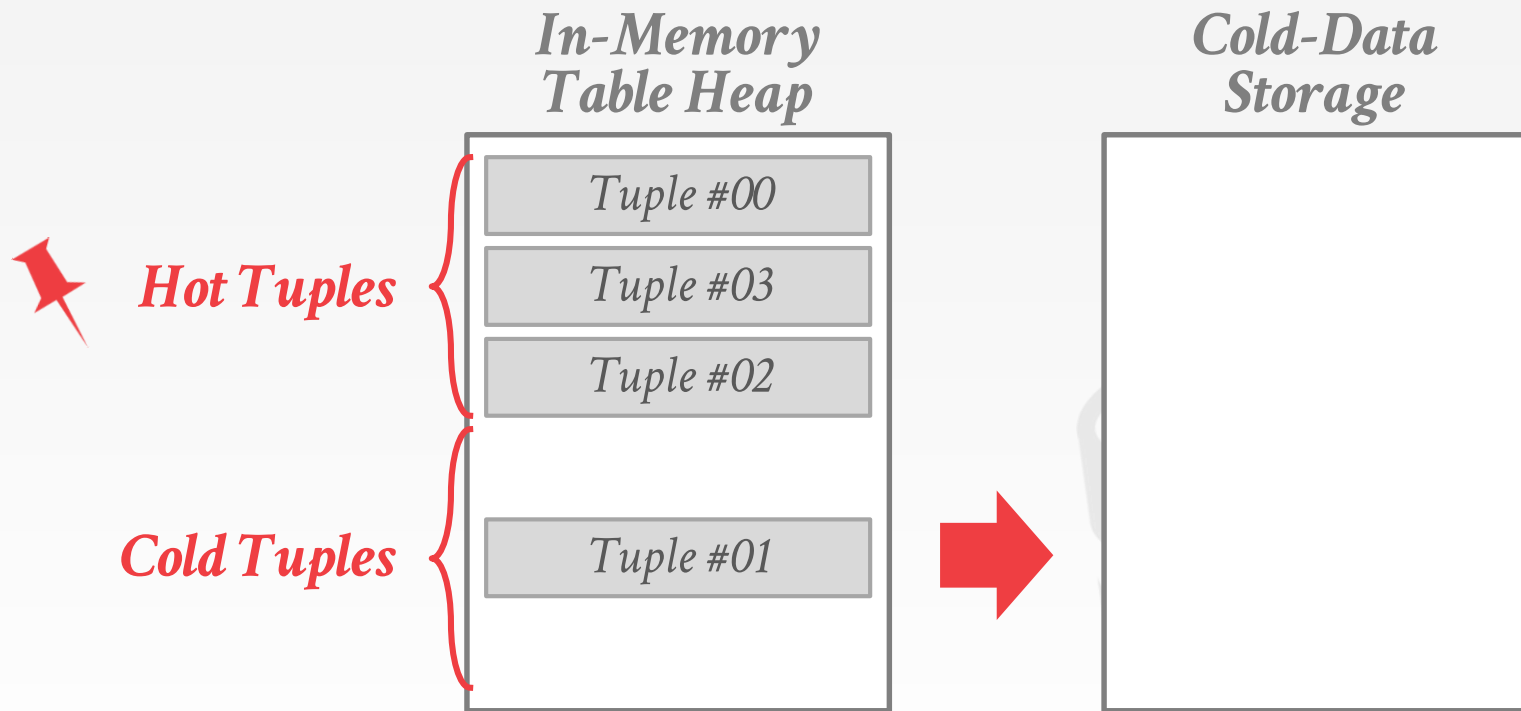
# EPFL VOLTDB



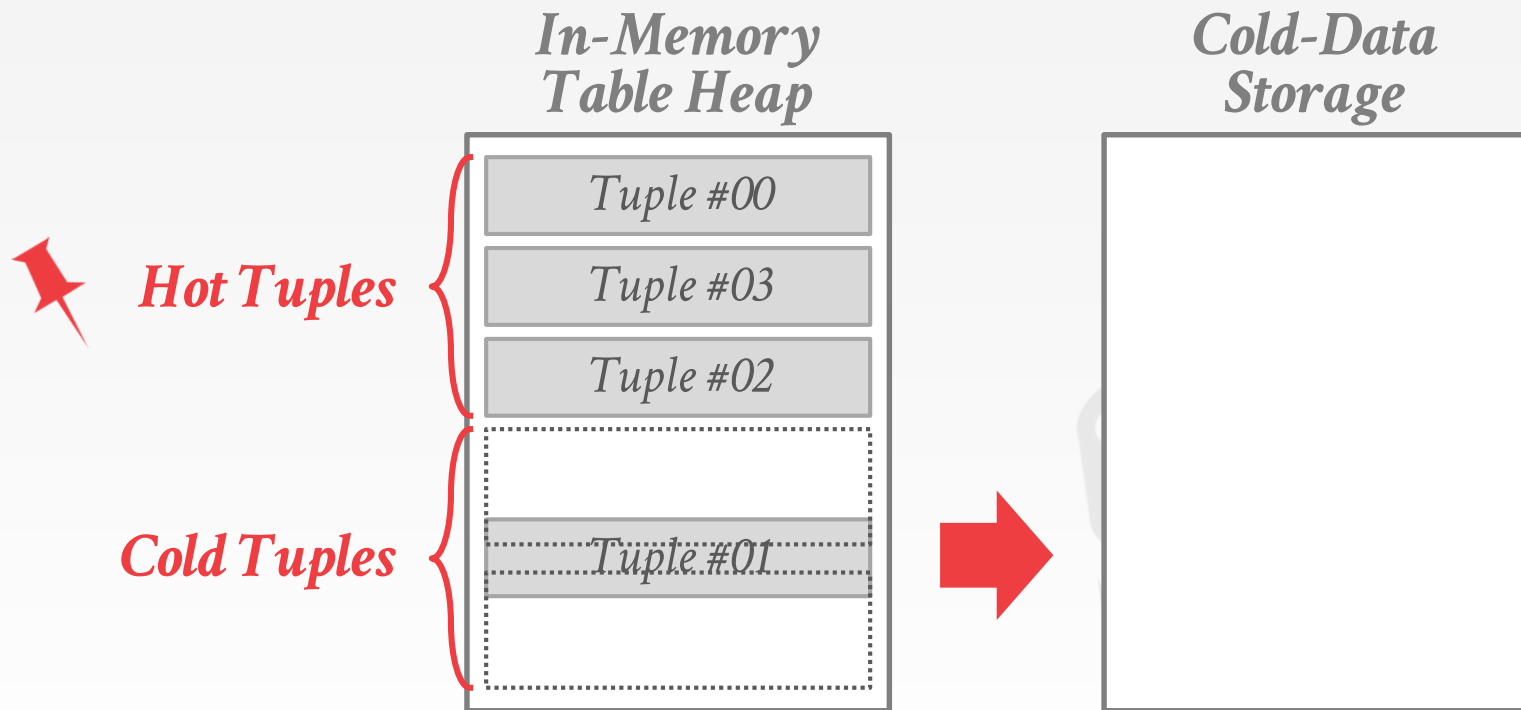
# EPFL VOLTDB



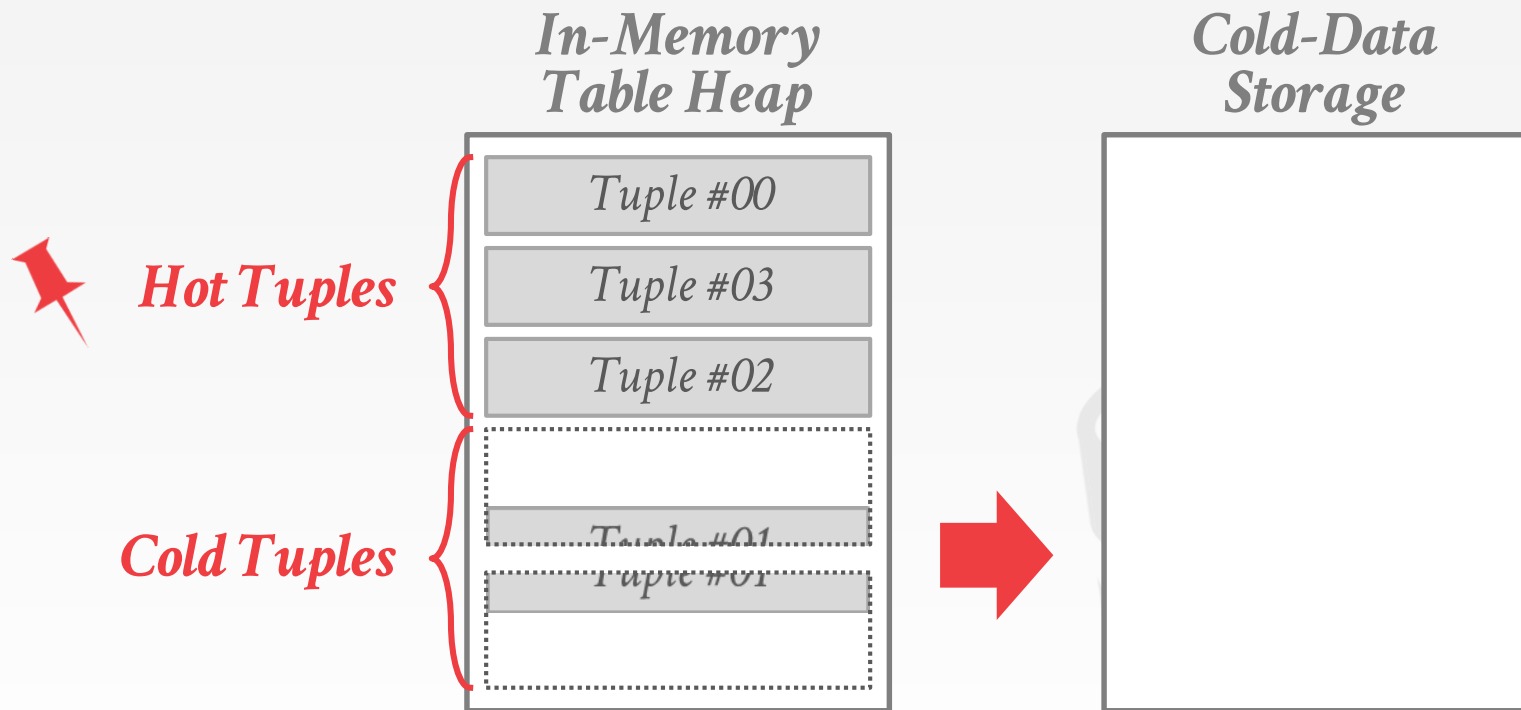
# EPFL VOLTDB



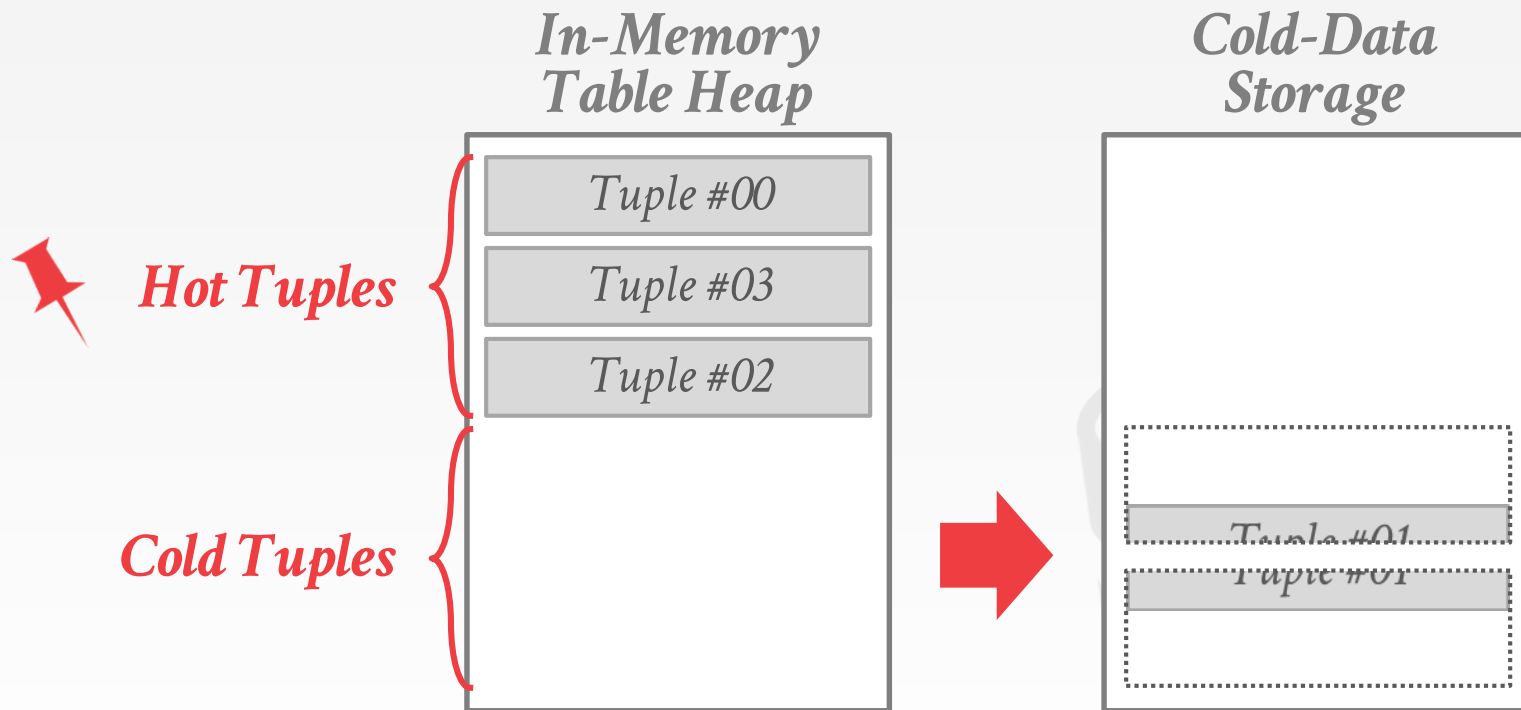
# EPFL VOLTDB



# EPFL VOLTDB



# EPFL VOLTDB





# APACHE GEODE – OVERFLOW TABLES

---

On-line Identification

Administrator-defined Threshold

Tombstones (?)

Synchronous Retrieval

Tuple-level Granularity

Merge Only on Update (?)



# MEMSQL – COLUMNAR TABLES

---

Administrator manually declares a table as a distinct disk-resident columnar table.

- Appears as a separate logical table to the application.
- Uses **mmap** to manage buffer pool.
- Pre-computed aggregates per block always in memory.

Manual Identification

No Evicted Metadata is needed.

Synchronous Retrieval

Always Merge

# EVALUATION

---

Compare different design decisions in H-Store with anti-caching.

## Storage Devices:

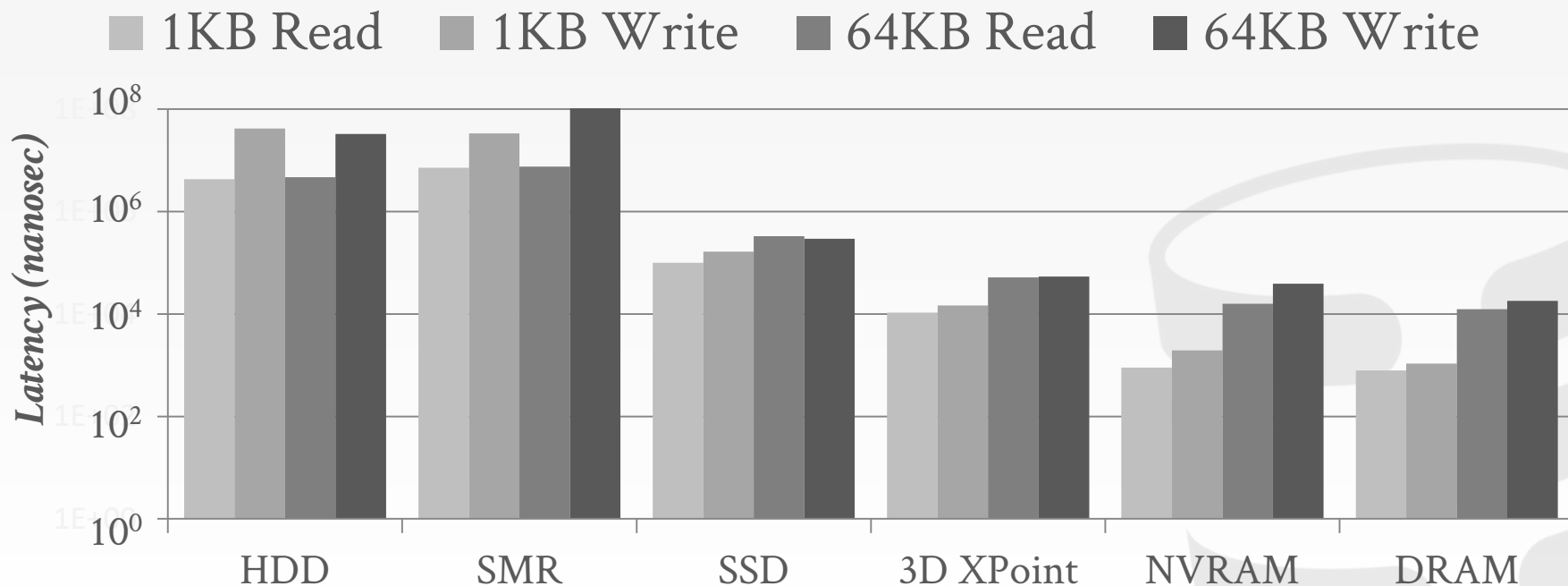
- Hard-Disk Drive (HDD)
- Shingled Magnetic Recording Drive (SMR)
- Solid-State Drive (SSD)
- 3D XPoint (3DX)
- Non-volatile Memory (NVRAM)



LARGER-THAN-MEMORY DATA MANAGEMENT ON MODERN  
STORAGE HARDWARE FOR IN-MEMORY OLTP DATABASE SYSTEMS  
DAMON 2016

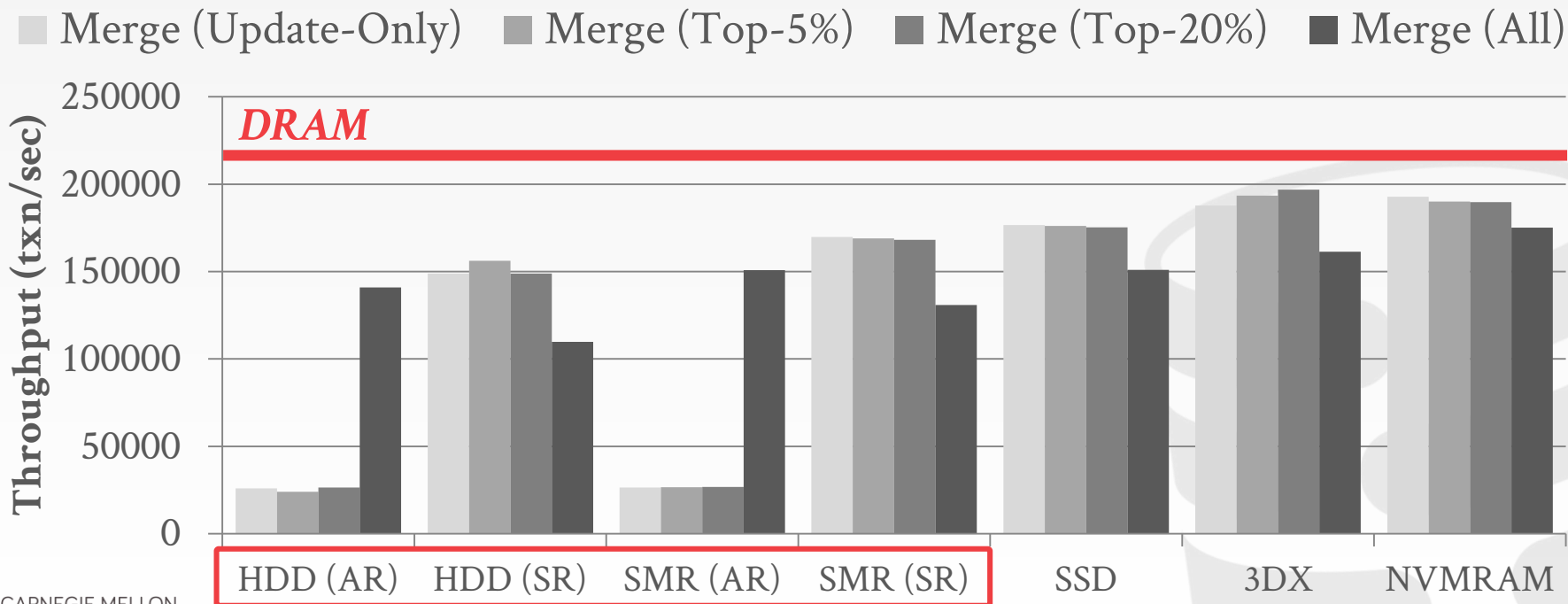
# MICROBENCHMARK

*10m tuples – 1KB each*  
*50% Reads / 50% Writes – Synchronization Enabled*



# MERGING THRESHOLD

*YCSB Workload – 90% Reads / 10% Writes*  
*10GB Database using 1.25GB Memory*



# CONFIGURATION COMPARISON

---

## Generic Configuration

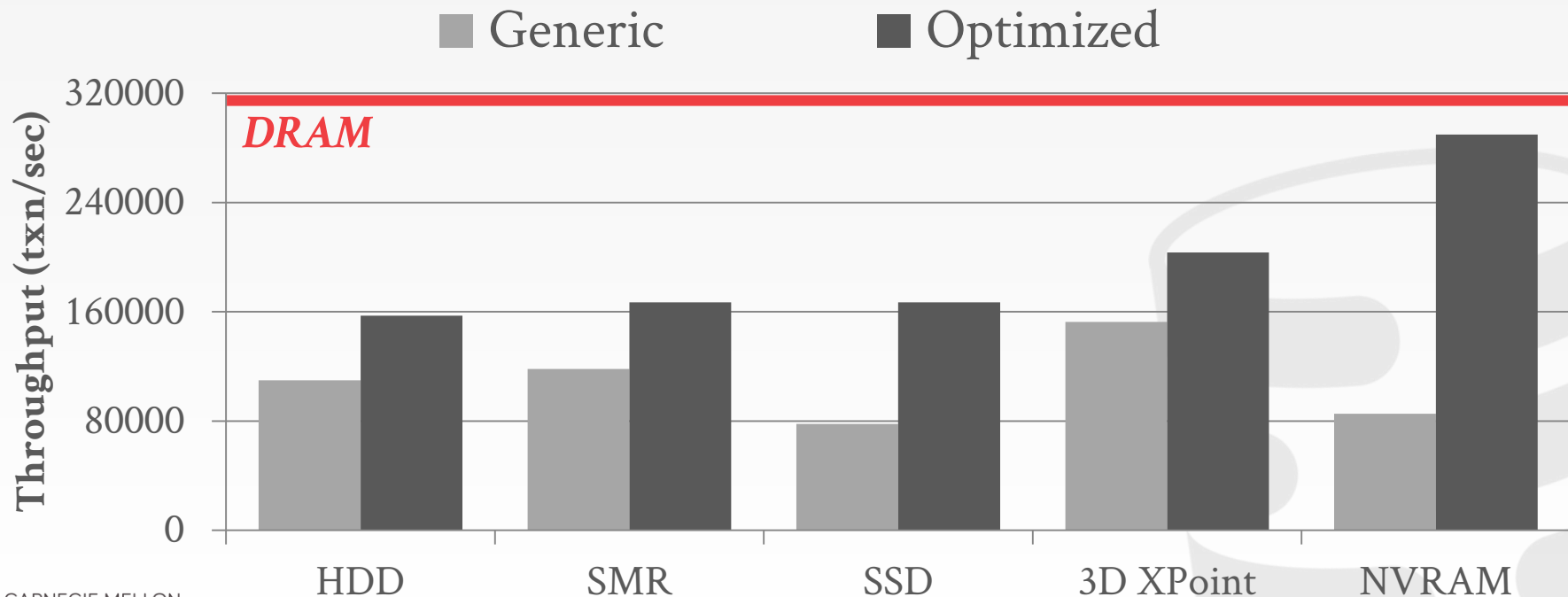
- Abort-and-Restart Retrieval
- Merge (All) Threshold
- 1024 KB Block Size

## Optimized Configuration

- Synchronous Retrieval
- Top-5% Merge Threshold
- Block Sizes (HDD/SMR - 1024 KB) (SSD/3DX - 16 KB)

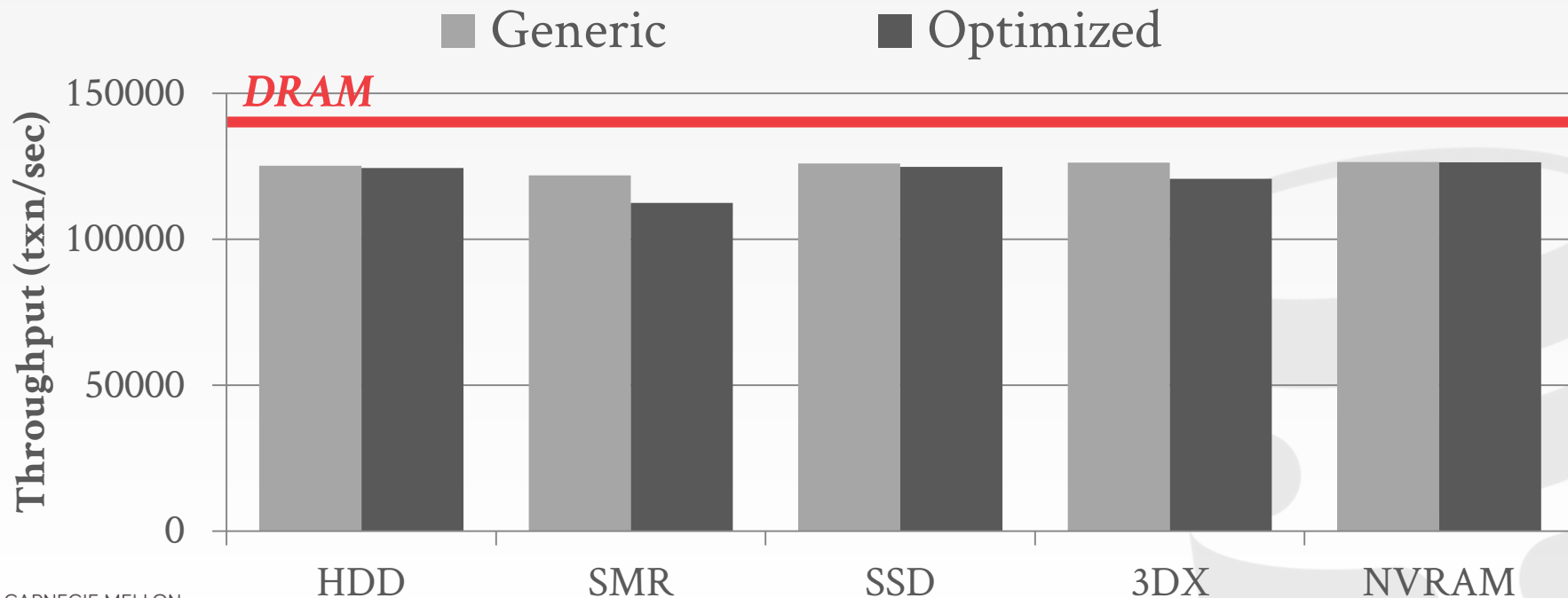
# TATP BENCHMARK

*Optimal Configuration per Storage Device*  
*1.25GB Memory*



# VOTER BENCHMARK

*Optimal Configuration per Storage Device*  
*1.25GB Memory*





# PARTING THOUGHTS

---

Today was about working around the block-oriented access and slowness of secondary storage.

None of these techniques handle index memory.

Fast & cheap byte-addressable NVM will make this lecture unnecessary.

# NEXT CLASS

---

Hardware! NVM! GPUs! HTM!