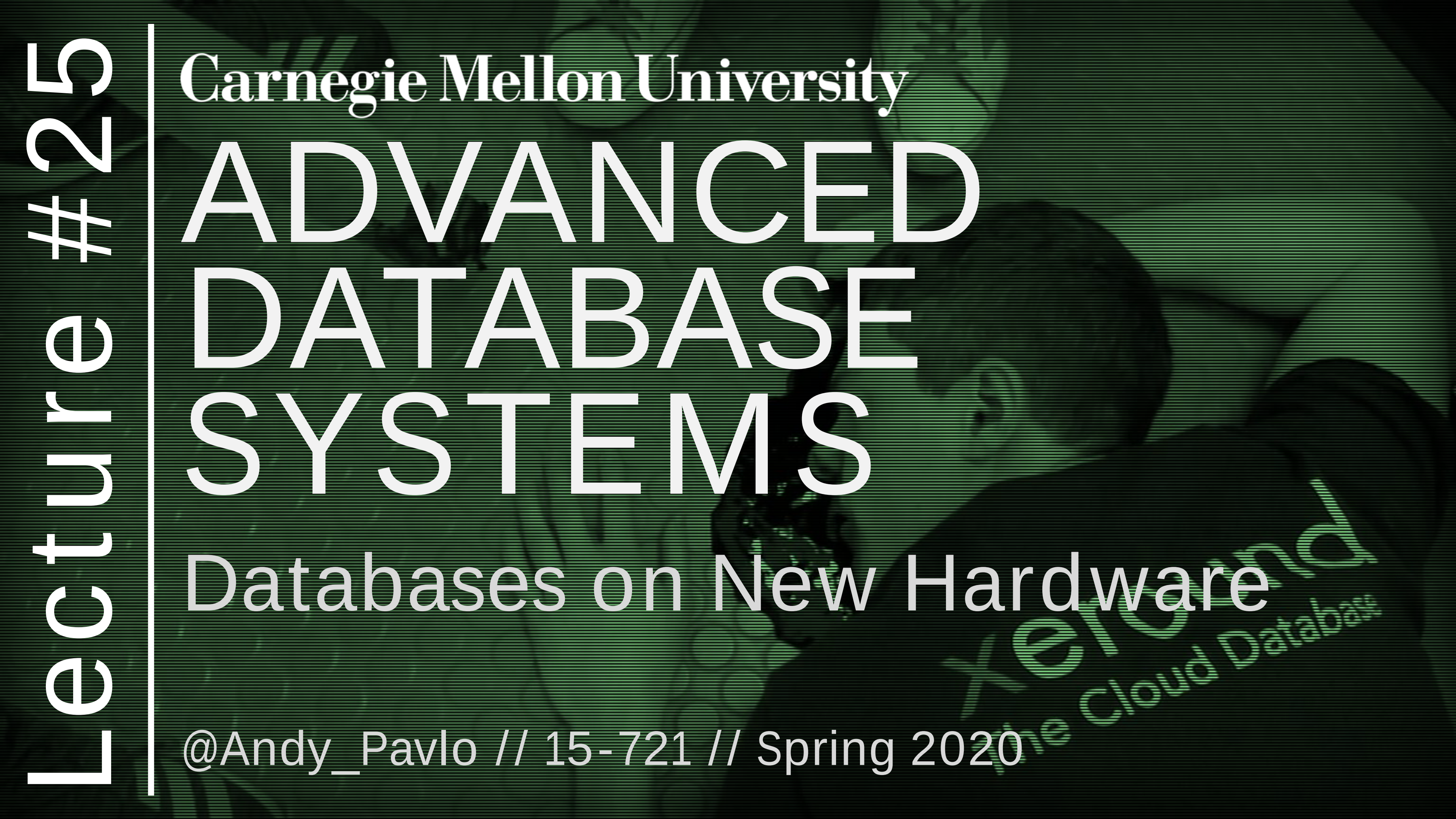


Lecture #25

Carnegie Mellon University

ADVANCED DATABASE SYSTEMS

Databases on New Hardware

@Andy_Pavlo // 15-721 // Spring 2020

ADMINISTRIVIA

April 29: Guest Speaker (Live)

May 4: Code Review #2 Submission

May 5: Final Presentations (Live)

May 13: Final Exam Due Date

May 16: Hack-a-Thon (Extra Credit, Optional)

ADMINISTRIVIA

Course Evaluation

→ Please tell me what you really think of me.
→ I take your feedback in consideration.
→ Take revenge on next year's students.

https://cmu.smartevals.com/

DATABASE 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!

DATABASE MACHINES: AN IDEA WHOSE TIME HAS PASSED? A CRITIQUE OF THE FUTURE OF DATABASE MACHINES
UNIVERSITY OF WISCONSIN 1983

TODAY'S AGENDA

Persistent Memory

GPU Acceleration

Hardware Transactional Memory

PERSISTENT MEMORY

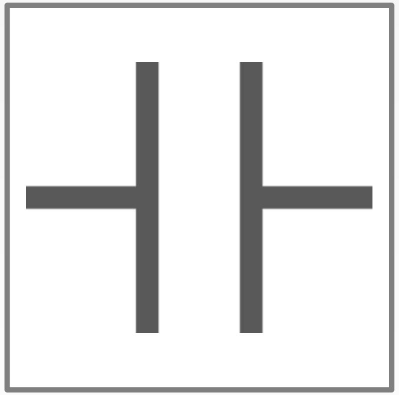
Emerging storage technology that provide low latency read/writes like DRAM, but with persistent writes and large capacities like SSDs.
→ aka Storage-class Memory, Non-Volatile Memory

First devices are block-addressable (NVMe)

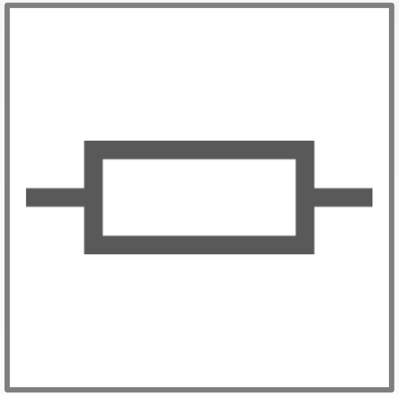
Later devices are byte-addressable.

FUNDAMENTAL ELEMENTS OF CIRCUITS

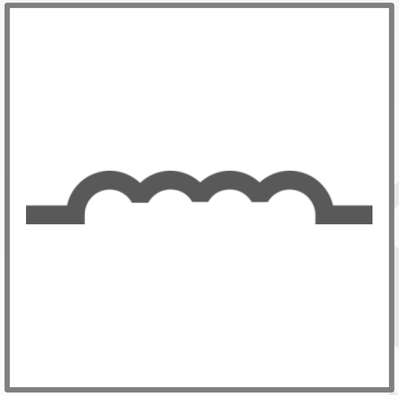
Capacitor (1745)



Resistor (1827)



Inductor (1831)



FUNDAMENTAL ELEMENTS OF CIRCUITS

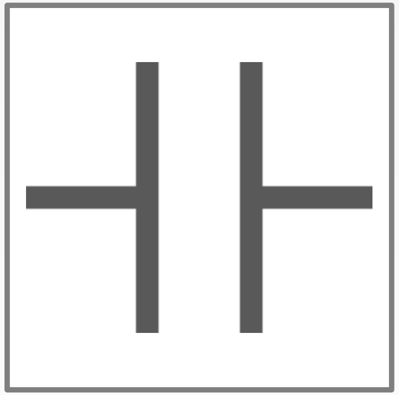
In 1971, Leon Chua at Berkeley predicted the existence of a fourth fundamental element.

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.

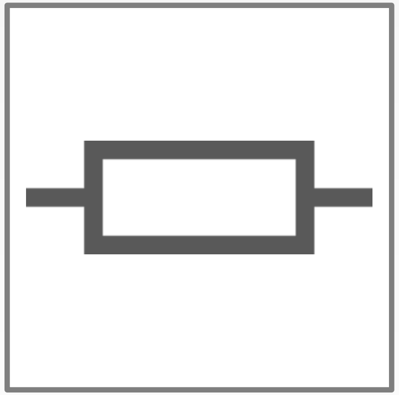
TWO CENTURIES OF MEMRISTORS
NATURE MATERIALS 2012

FUNDAMENTAL ELEMENTS OF CIRCUITS

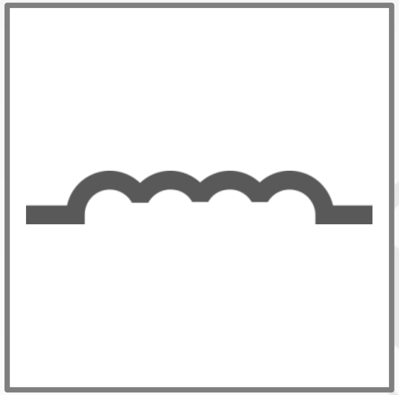
Capacitor (1745)



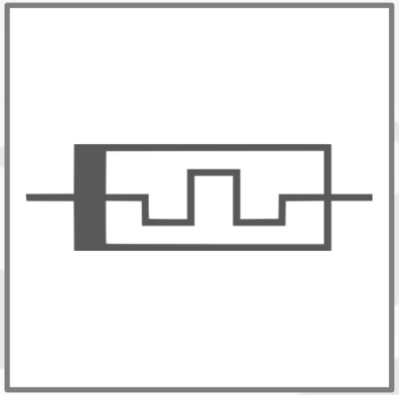
Resistor (1827)



Inductor (1831)



Memristor (1971)

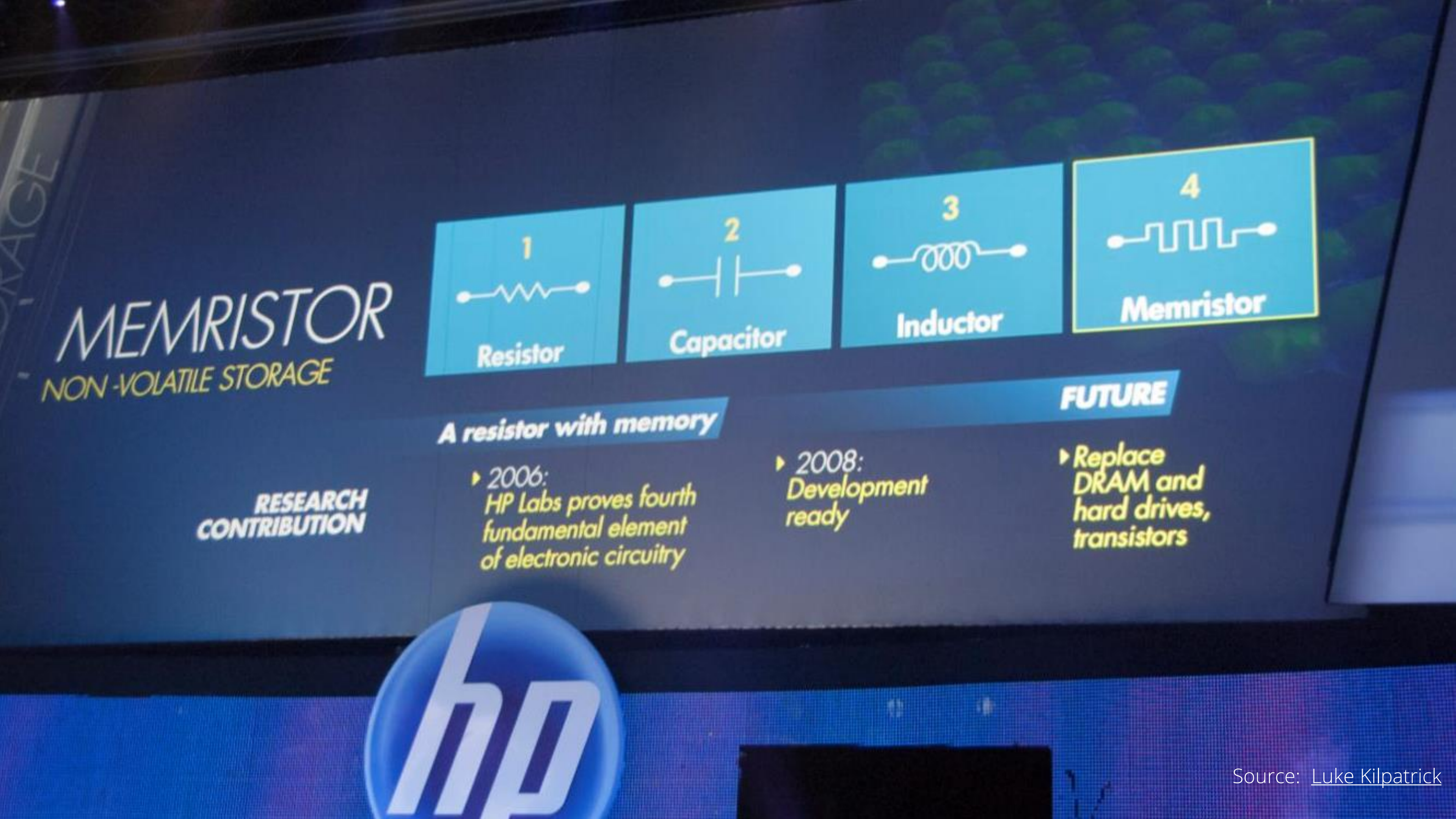


MERISTORS

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.

HOW WE FOUND THE MISSING MEMRISTOR
IEEE SPECTRUM 2008



MEMRISTOR

NON-VOLATILE STORAGE

1 Resistor

2 Capacitor

3 Inductor

4 Memristor

A resistor with memory

FUTURE

RESEARCH CONTRIBUTION

- 2006: HP Labs proves fourth fundamental element of electronic circuitry
- 2008: Development ready
- Replace DRAM and hard drives, transistors

Source: Luke Kilpatrick

TECHNOLOGIES

Phase-Change Memory (PRAM)

Resistive RAM (ReRAM)

Magnetoresistive RAM (MRAM)

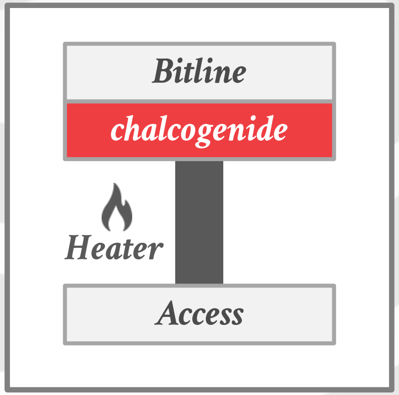
PHASE-CHANGE MEMORY

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'.

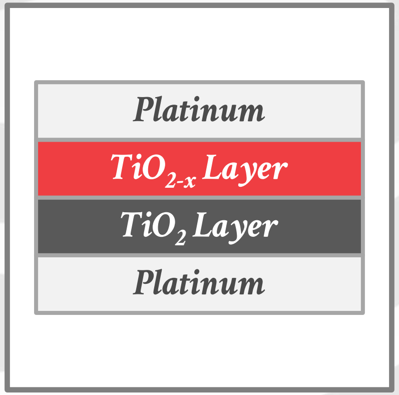


PHASE CHANGE MEMORY ARCHITECTURE AND THE QUEST FOR SCALABILITY
COMMUNICATIONS OF THE ACM 2010

RESISTIVE RAM

Two metal layers with two TiO_2 layers in between. Running a current one direction moves electrons from the top TiO_2 layer to the bottom, thereby changing the resistance.

Potential programmable storage fabric…
→ Bertrand Russell's Material Implication Logic

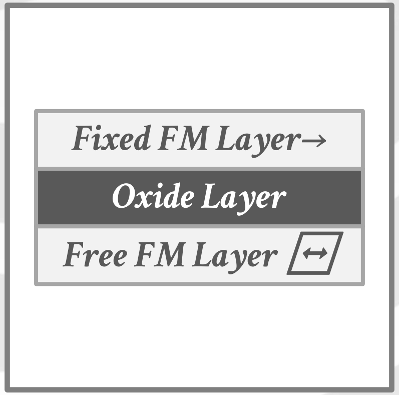


HOW WE FOUND THE MISSING MEMRISTOR
IEEE SPECTRUM 2008

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 small sizes (10nm) and have SRAM latencies.



SPIN MEMORY SHOWS ITS MIGHT
IEEE SPECTRUM 2014

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).

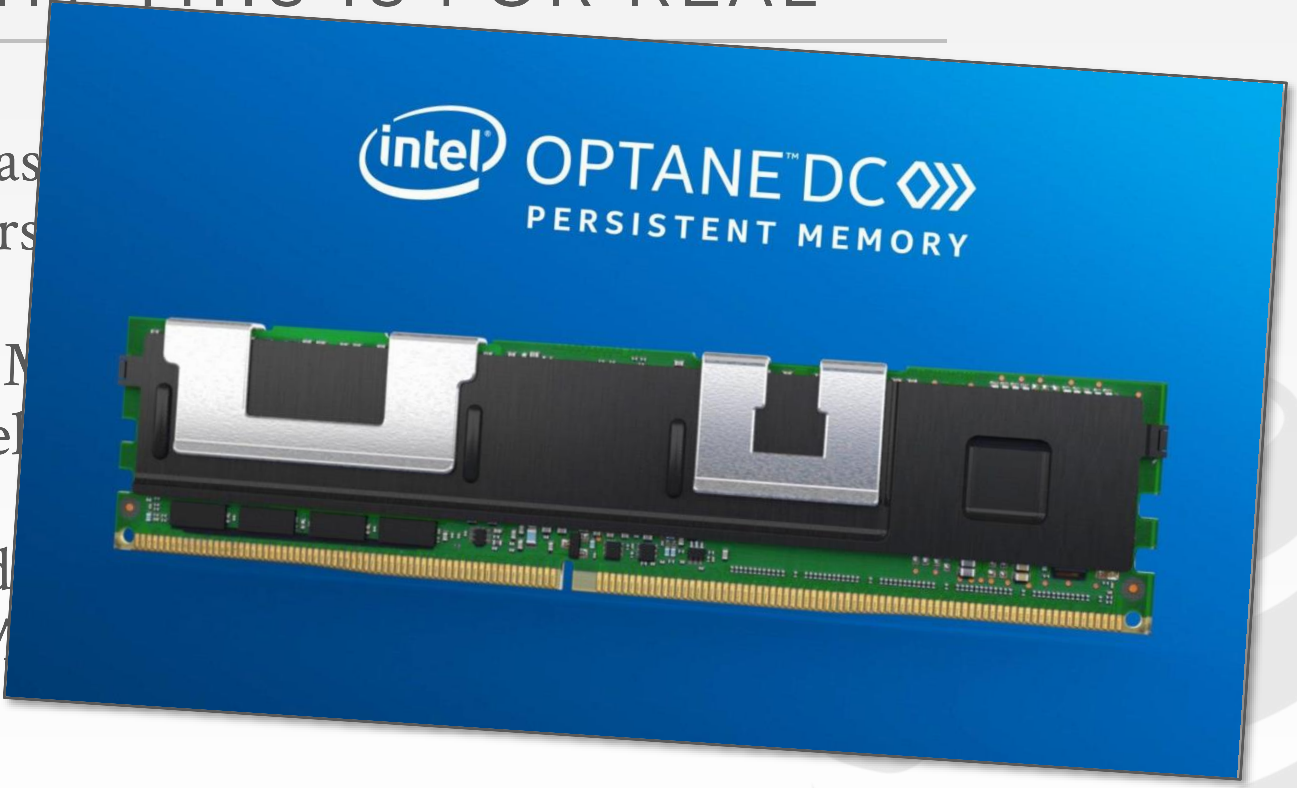
Intel added new instructions for flushing cache lines to PM (**`CLFLUSH`**, **`CLWB`**).

WHY THIS IS FOR REAL

Industry has
form factors

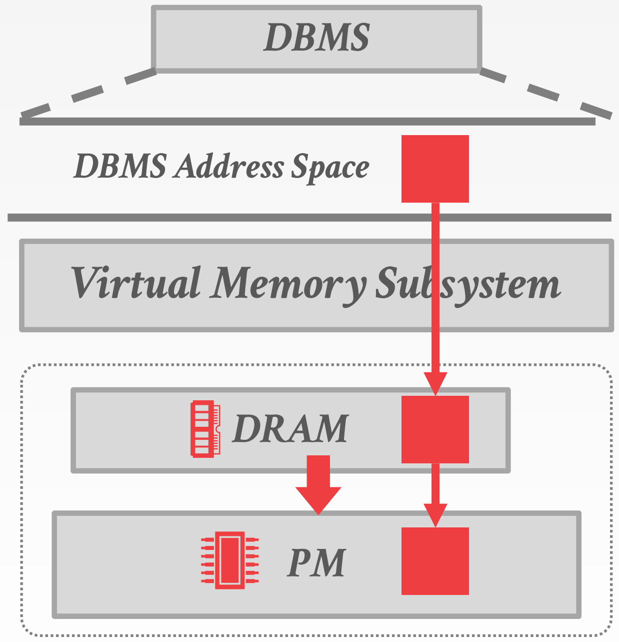
Linux and M
their kernel

Intel added
lines to PM

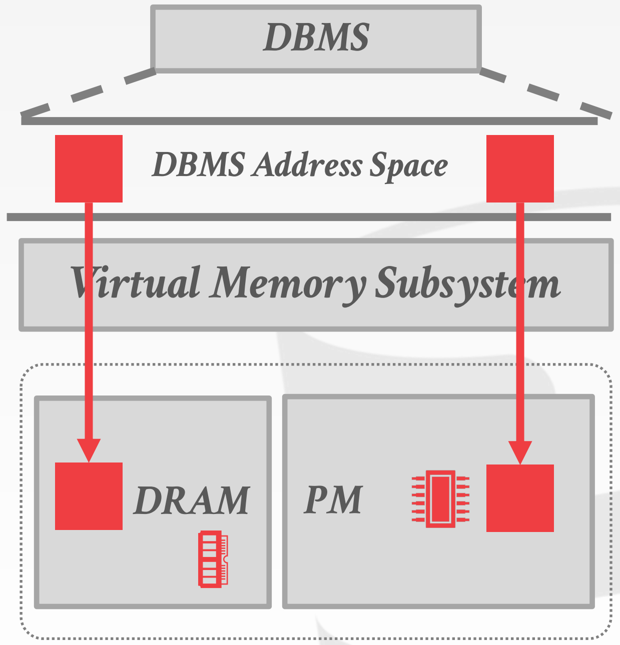


PM CONFIGURATIONS

DRAM as Hardware-Managed Cache



PM Next to DRAM



Source: Ismail Oukid

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.

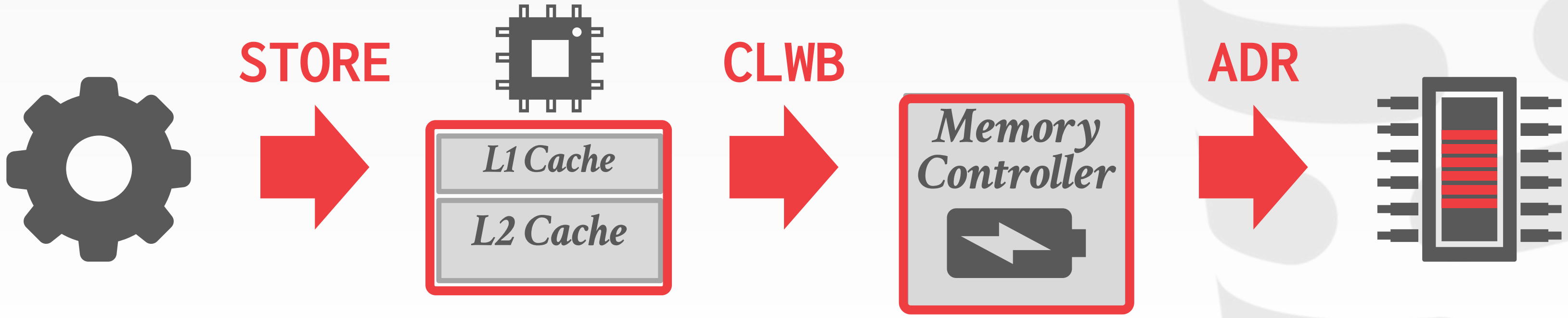
Based on the N-Store prototype DBMS.

LET'S TALK ABOUT STORAGE & RECOVERY METHODS FOR NON-VOLATILE MEMORY DATABASE SYSTEMS
SIGMOD 2015

SYNCHRONIZATION

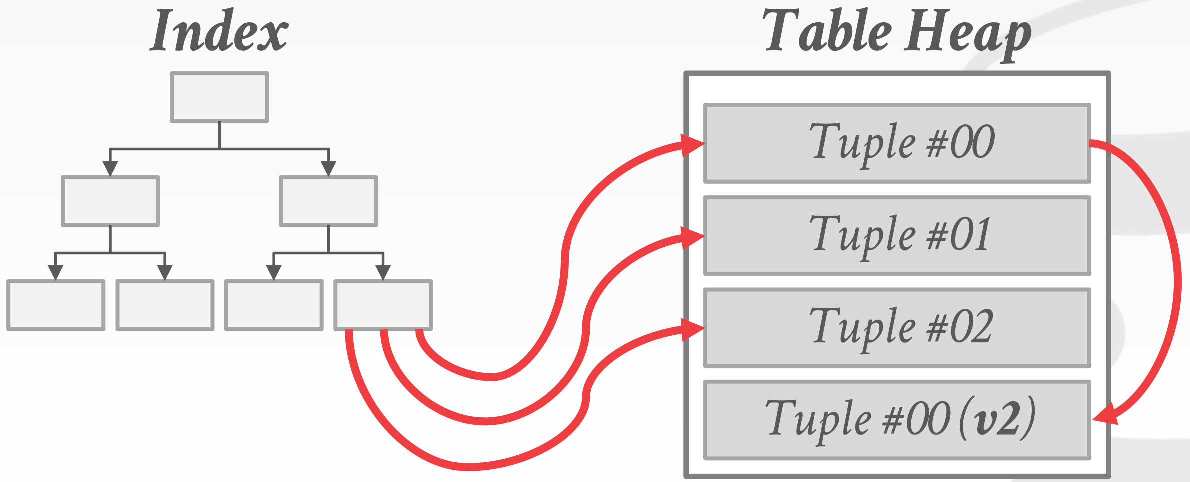
Existing programming models assume that any write to memory is non-volatile.
→ CPU decides when to move data from caches to DRAM.

The DBMS needs a way to ensure that data is flushed from caches to PM.



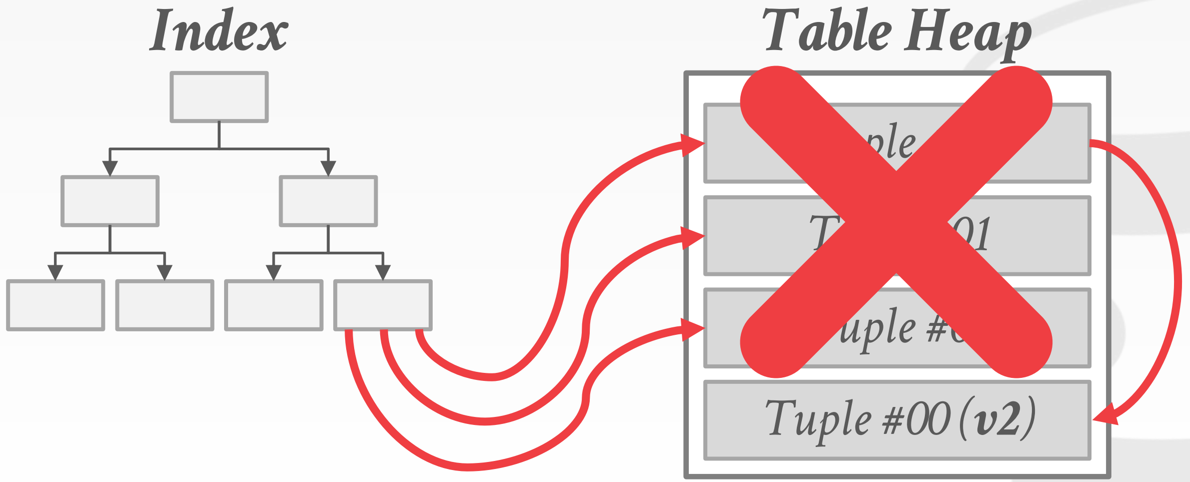
NAMING

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.



NAMING

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

Index

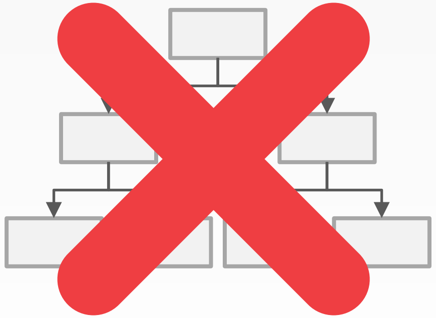
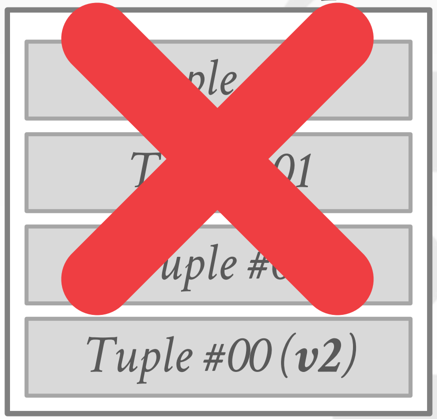


Table Heap



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.

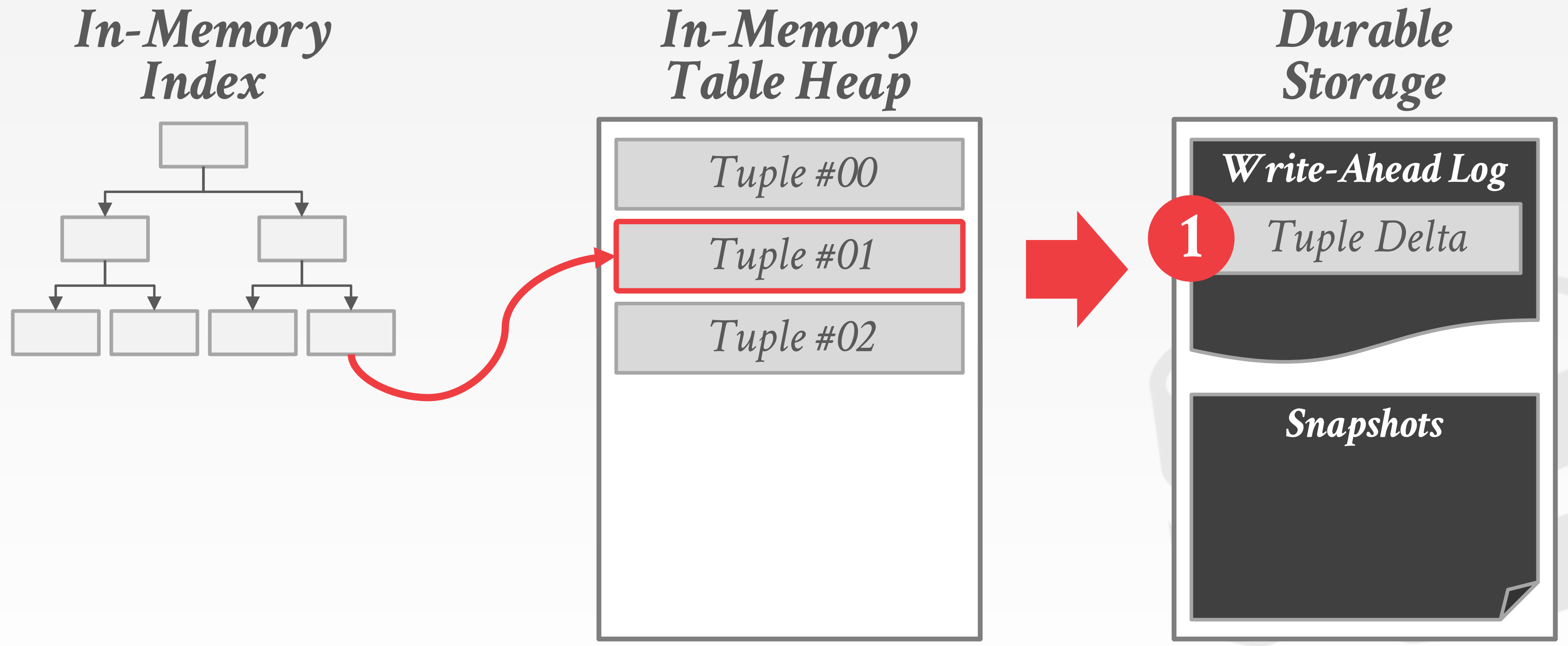
DBMS ENGINE ARCHITECTURES

Choice #1: In-place Updates
→ Table heap with a write-ahead log + snapshots.
→ Example: VoltDB

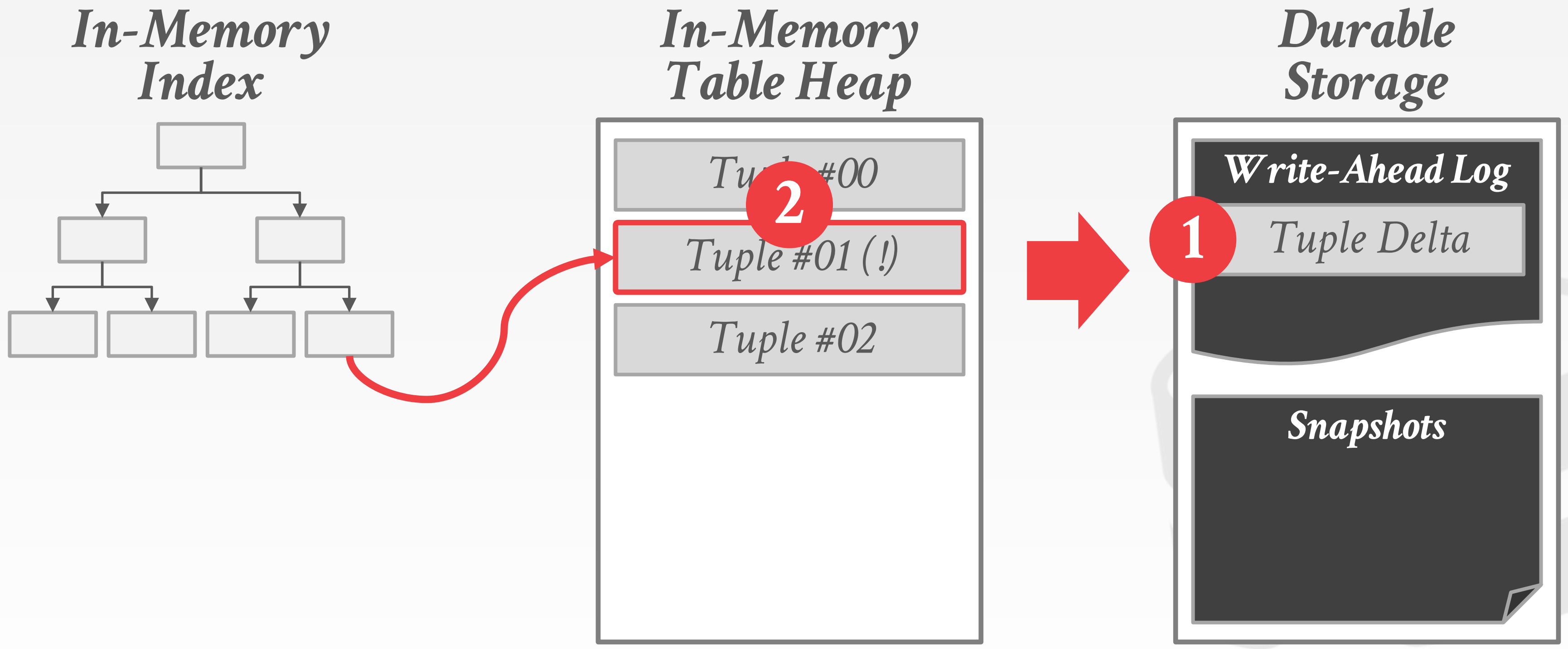
Choice #2: Copy-on-Write
→ Create a shadow copy of the table when updated.
→ No write-ahead log.
→ Example: LMDB

Choice #3: Log-structured
→ All writes are appended to log. No table heap.
→ Example: RocksDB

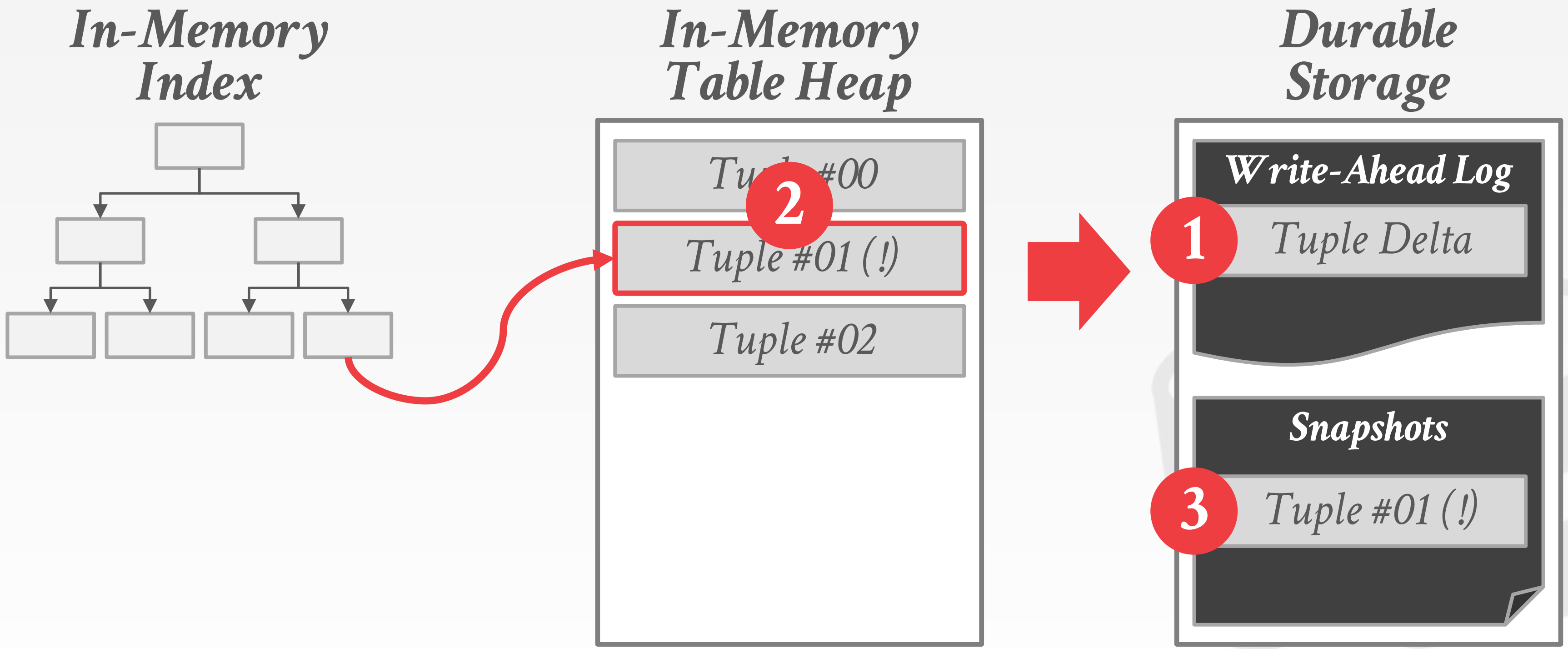
IN-PLACE UPDATES ENGINE



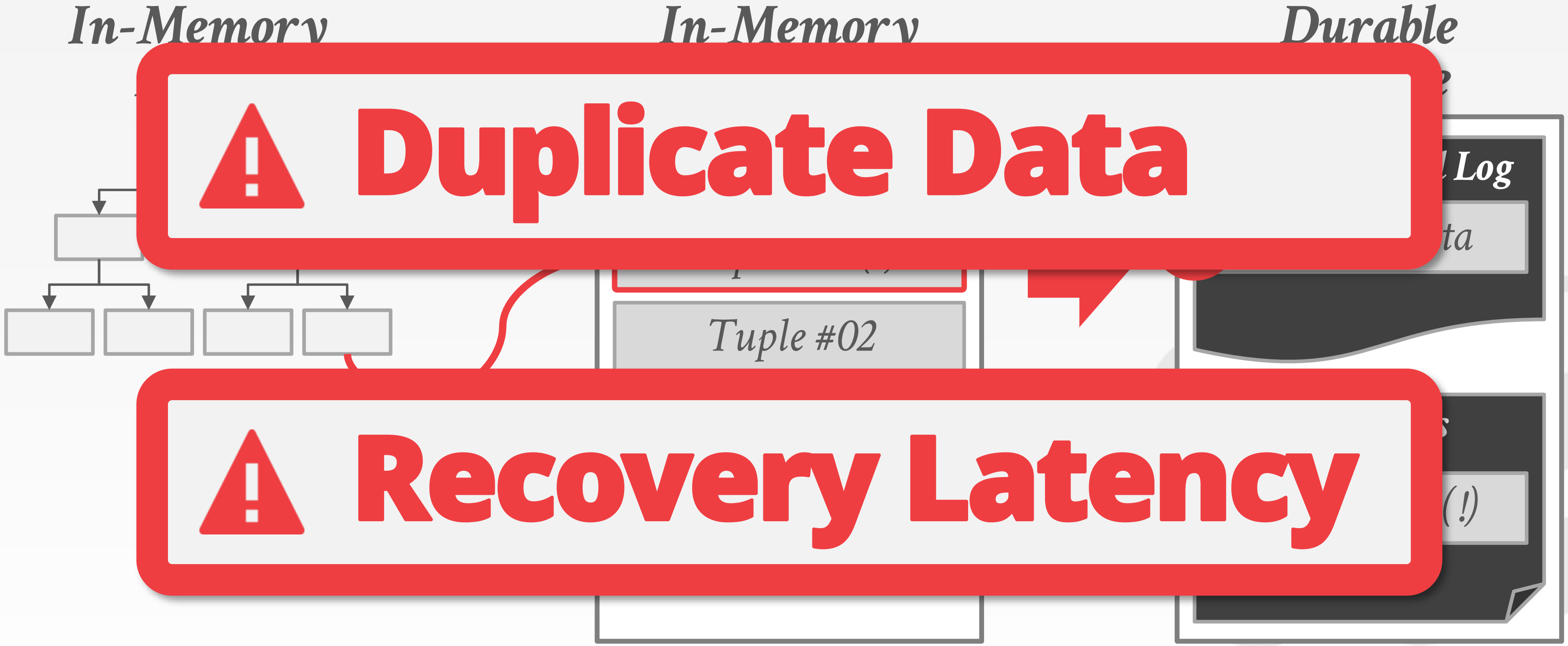
IN-PLACE UPDATES ENGINE



IN-PLACE UPDATES ENGINE



IN-PLACE UPDATES ENGINE



⚠ Duplicate Data

⚠ Recovery Latency

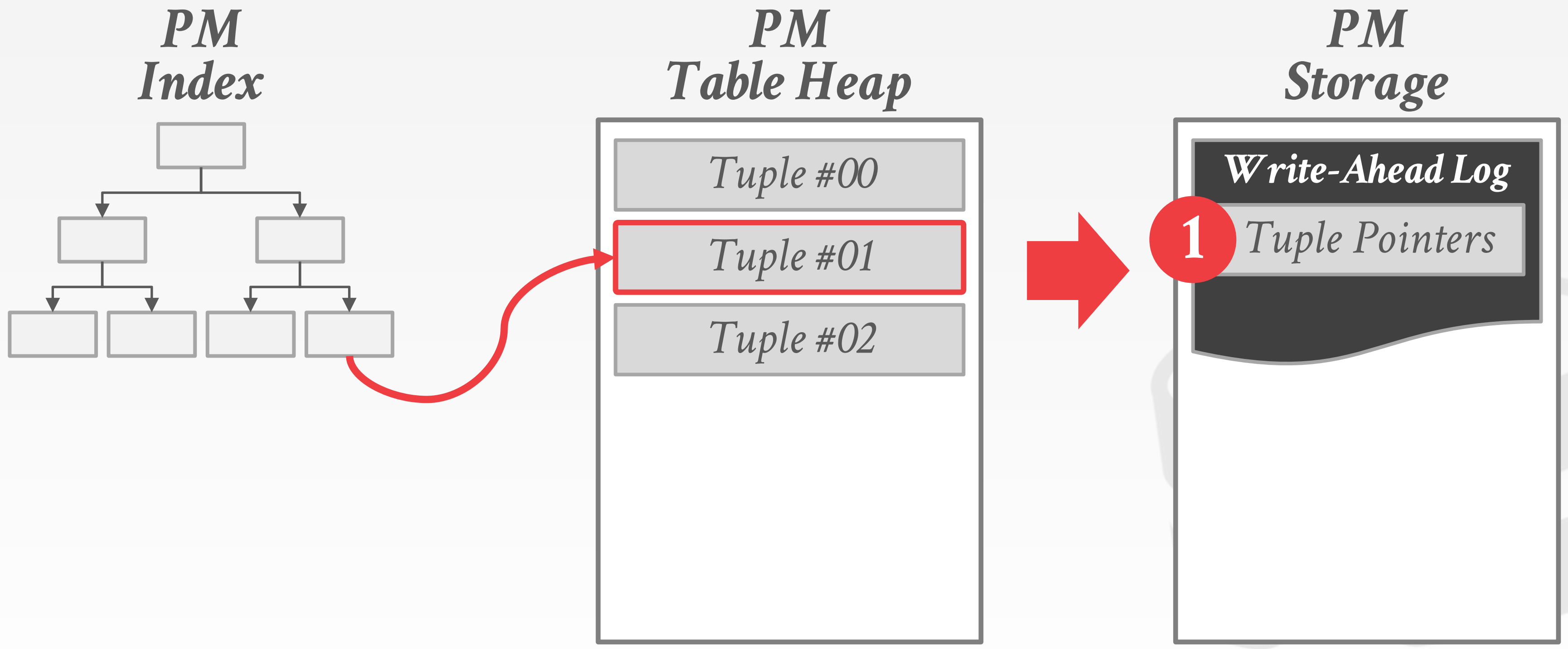
PM-OPTIMIZED ARCHITECTURES

Leverage the allocator's non-volatile pointers to only record what changed rather than how it changed.

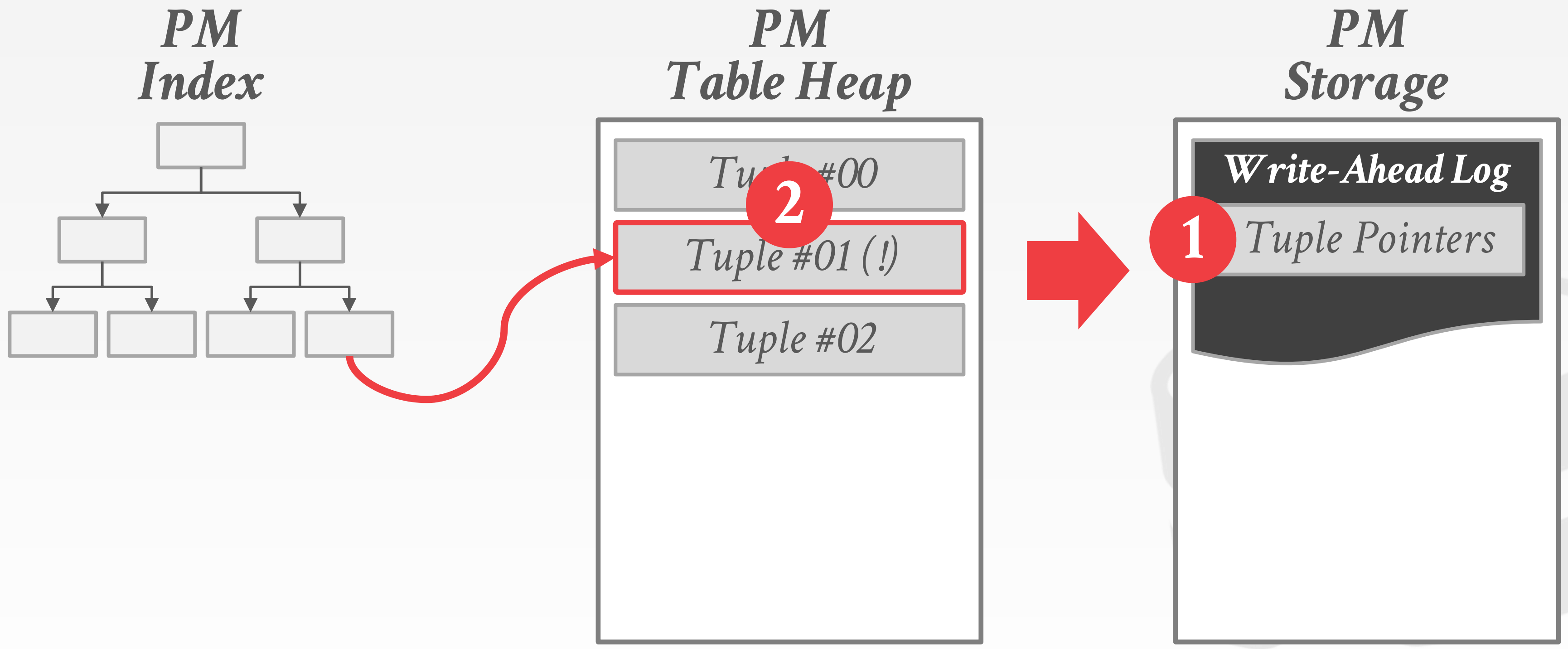
The DBMS only must maintain a transient UNDO log for a txn until it commits.

→ 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.

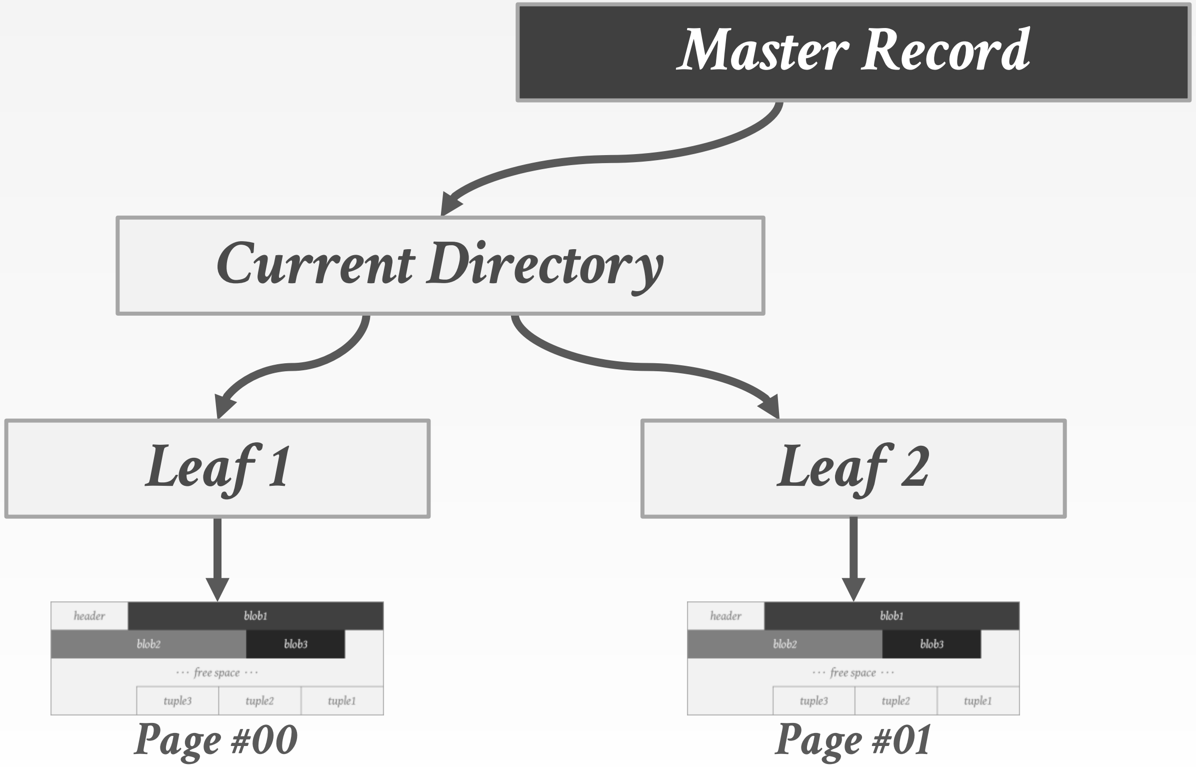
PM IN-PLACE UPDATES ENGINE



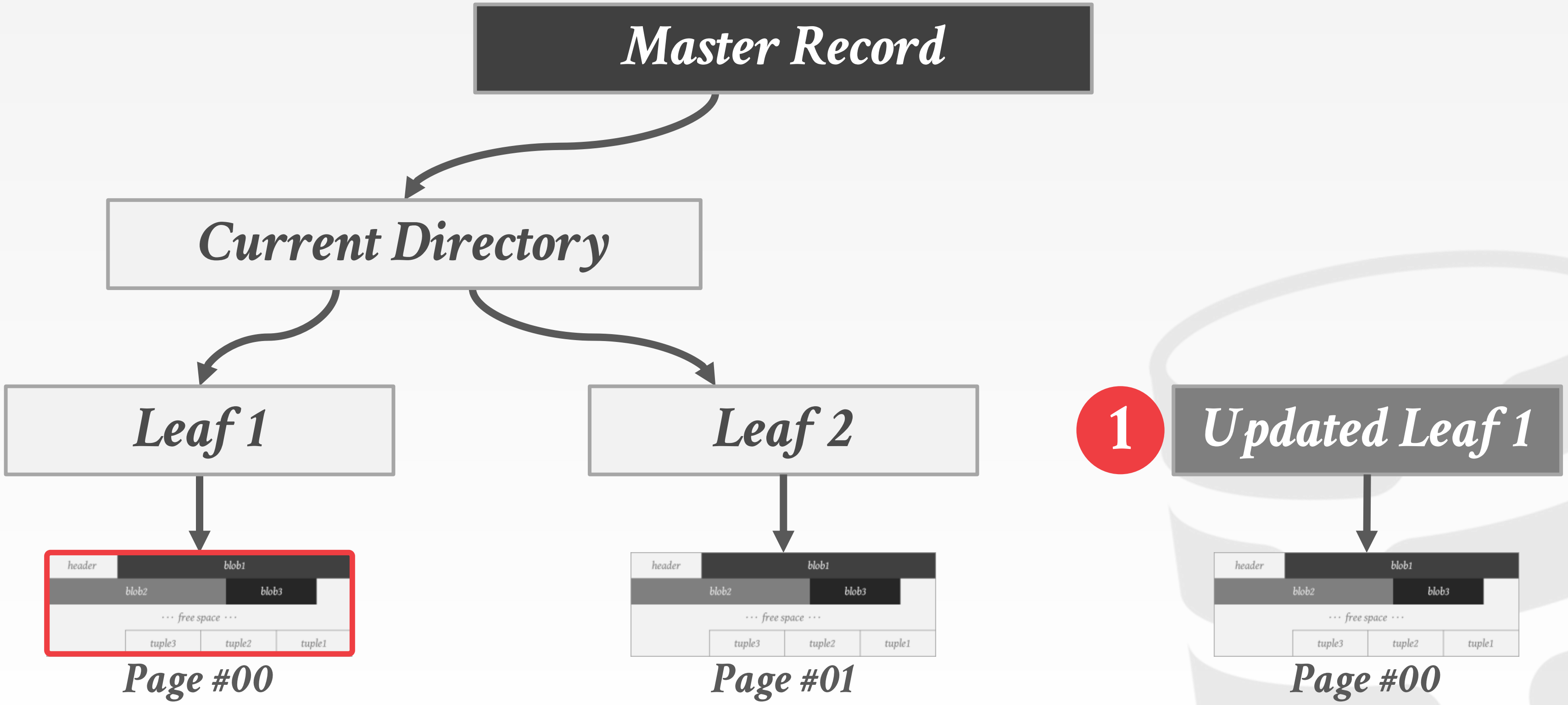
PM IN-PLACE UPDATES ENGINE



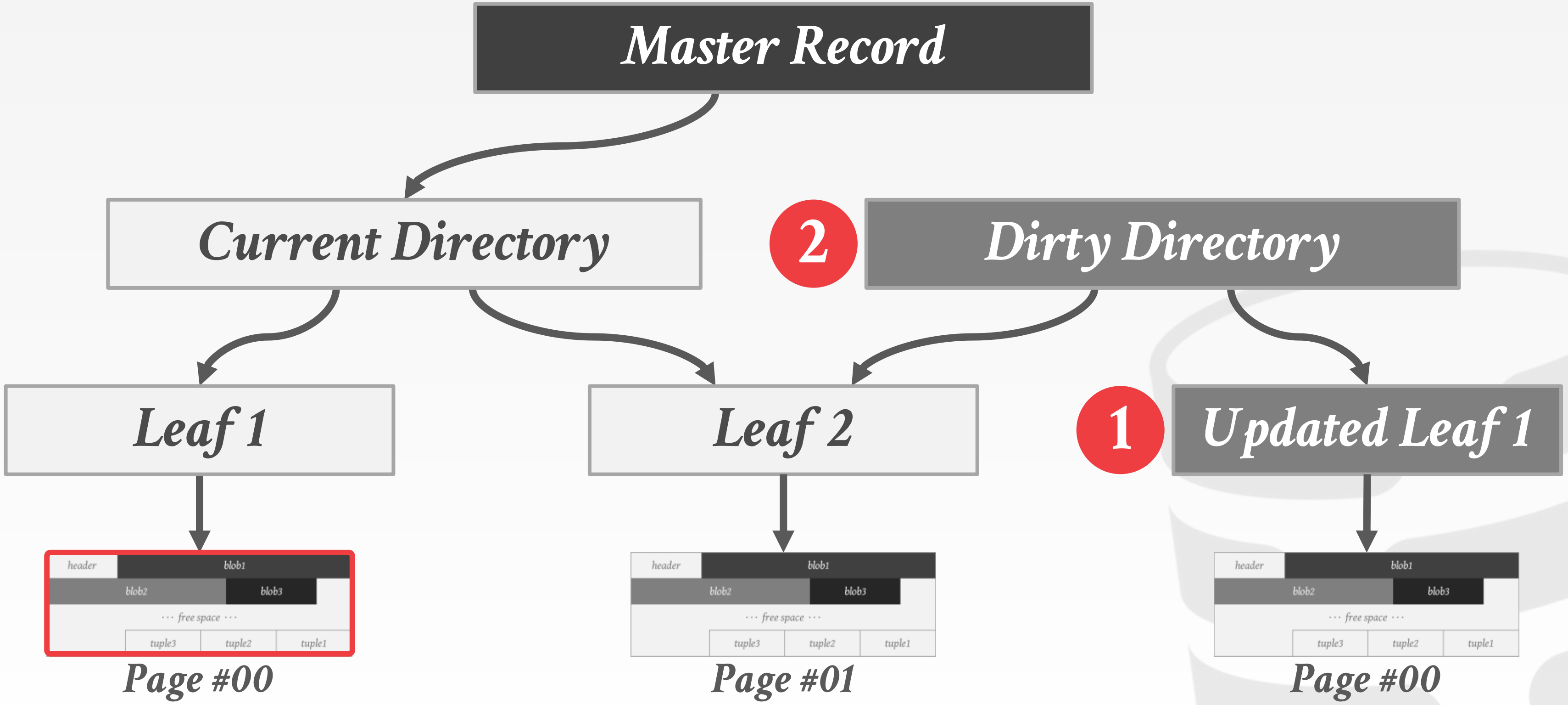
COPY-ON-WRITE ENGINE



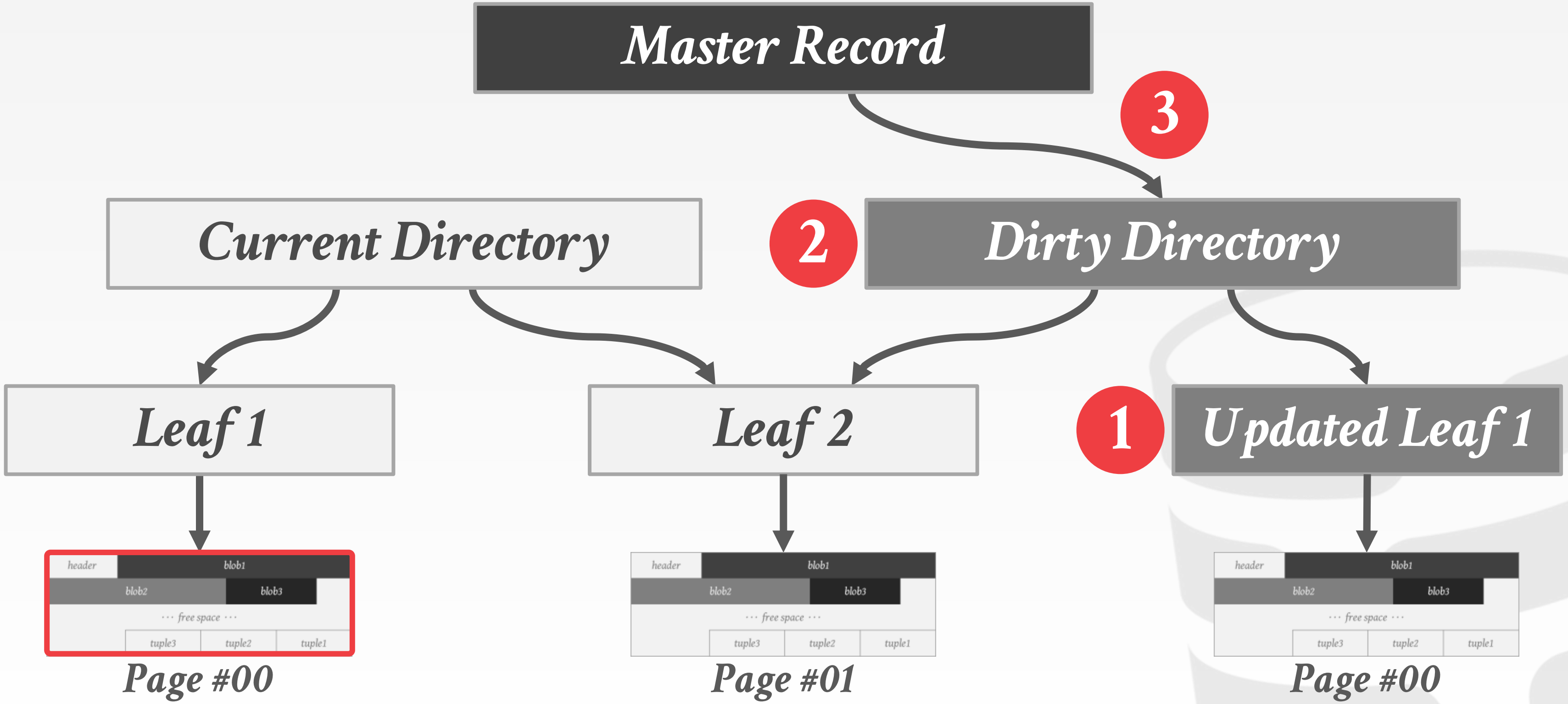
COPY-ON-WRITE ENGINE



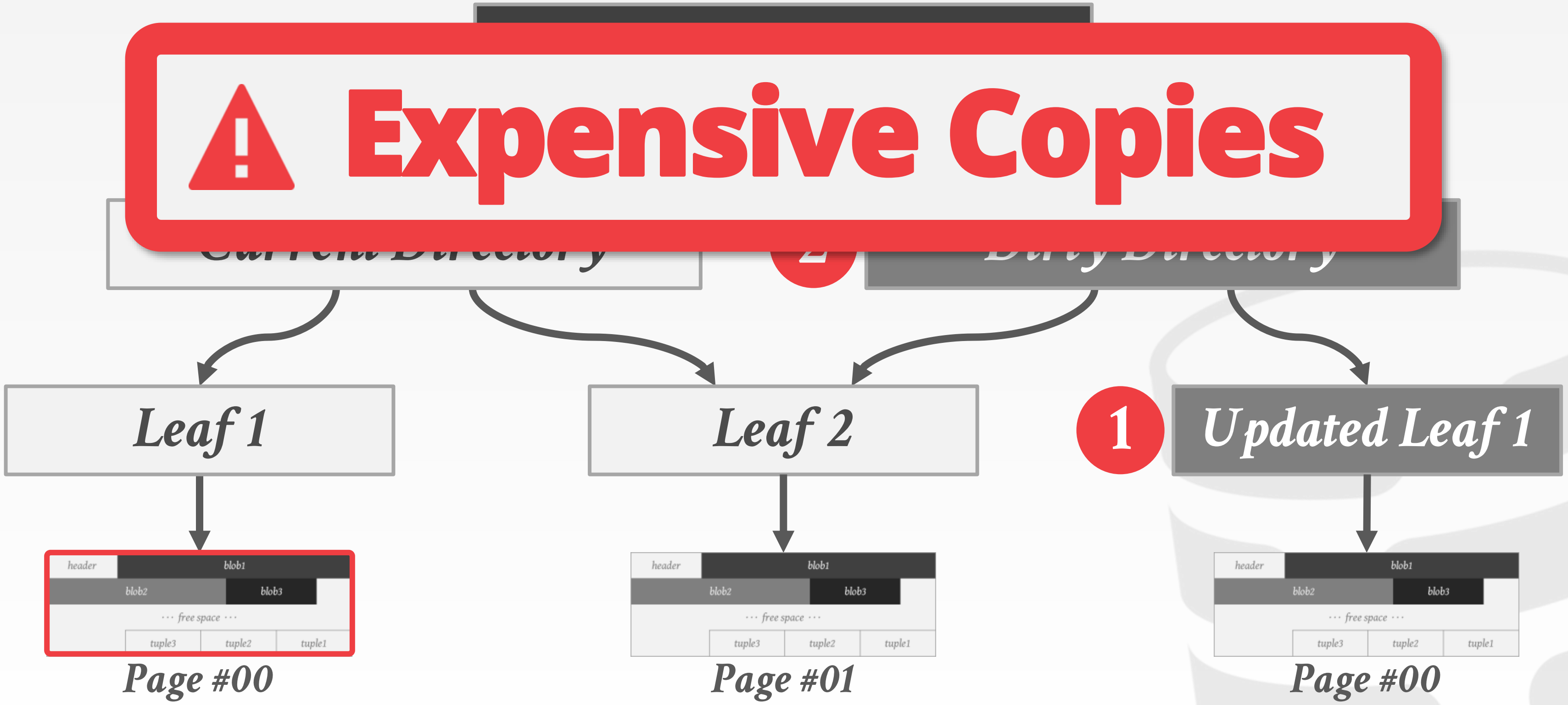
COPY-ON-WRITE ENGINE



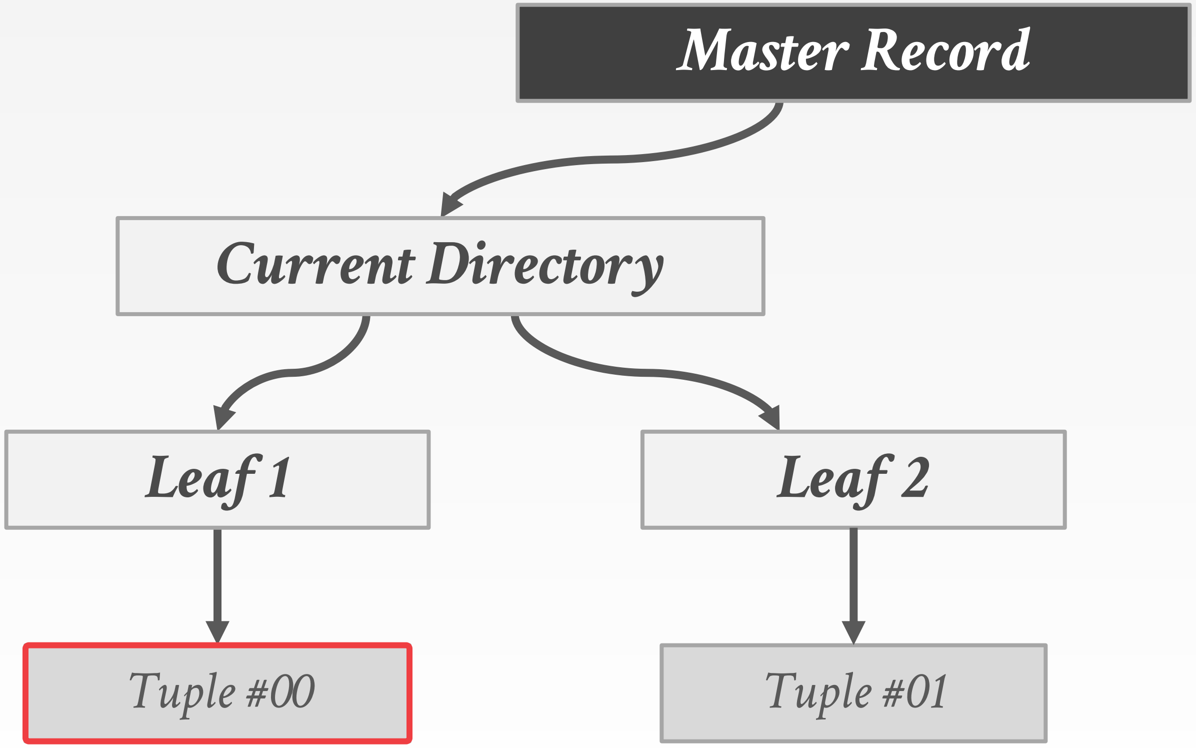
COPY-ON-WRITE ENGINE



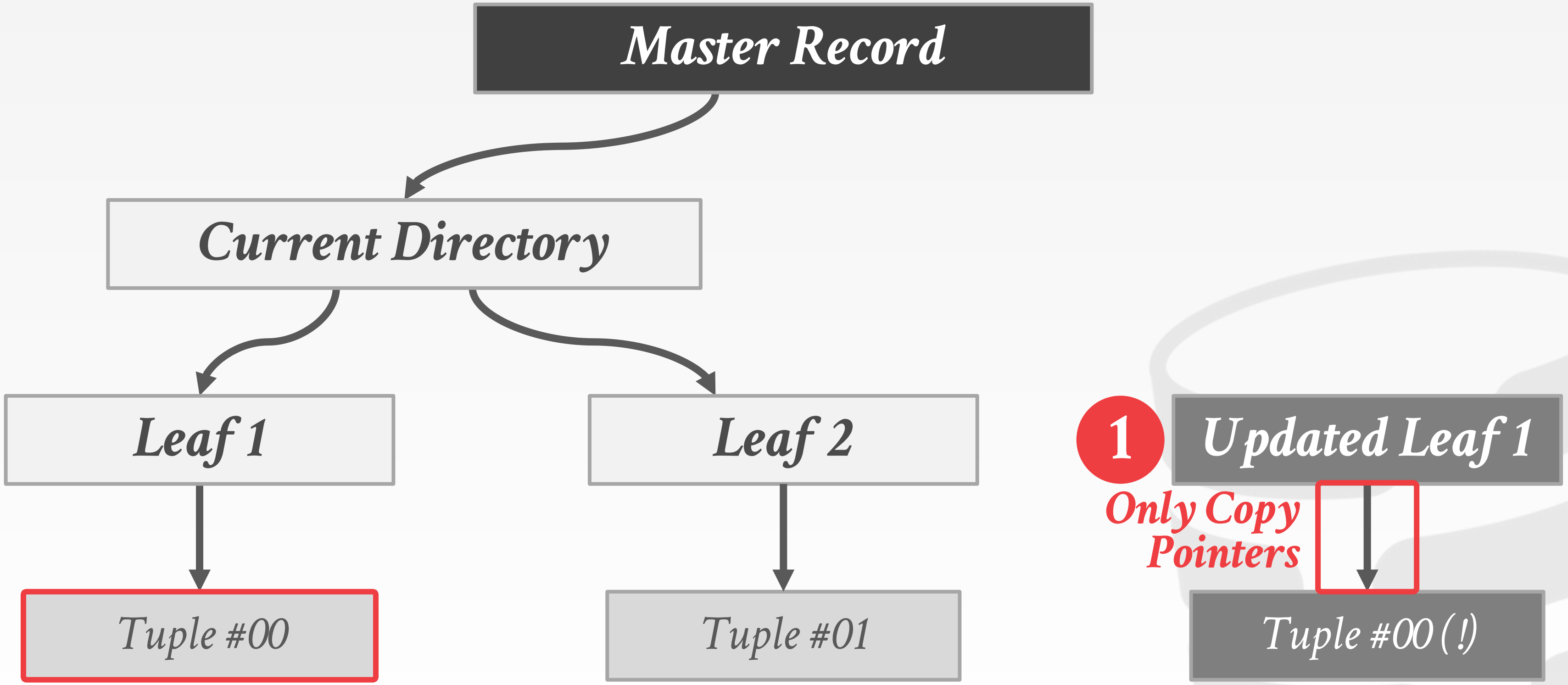
COPY-ON-WRITE ENGINE



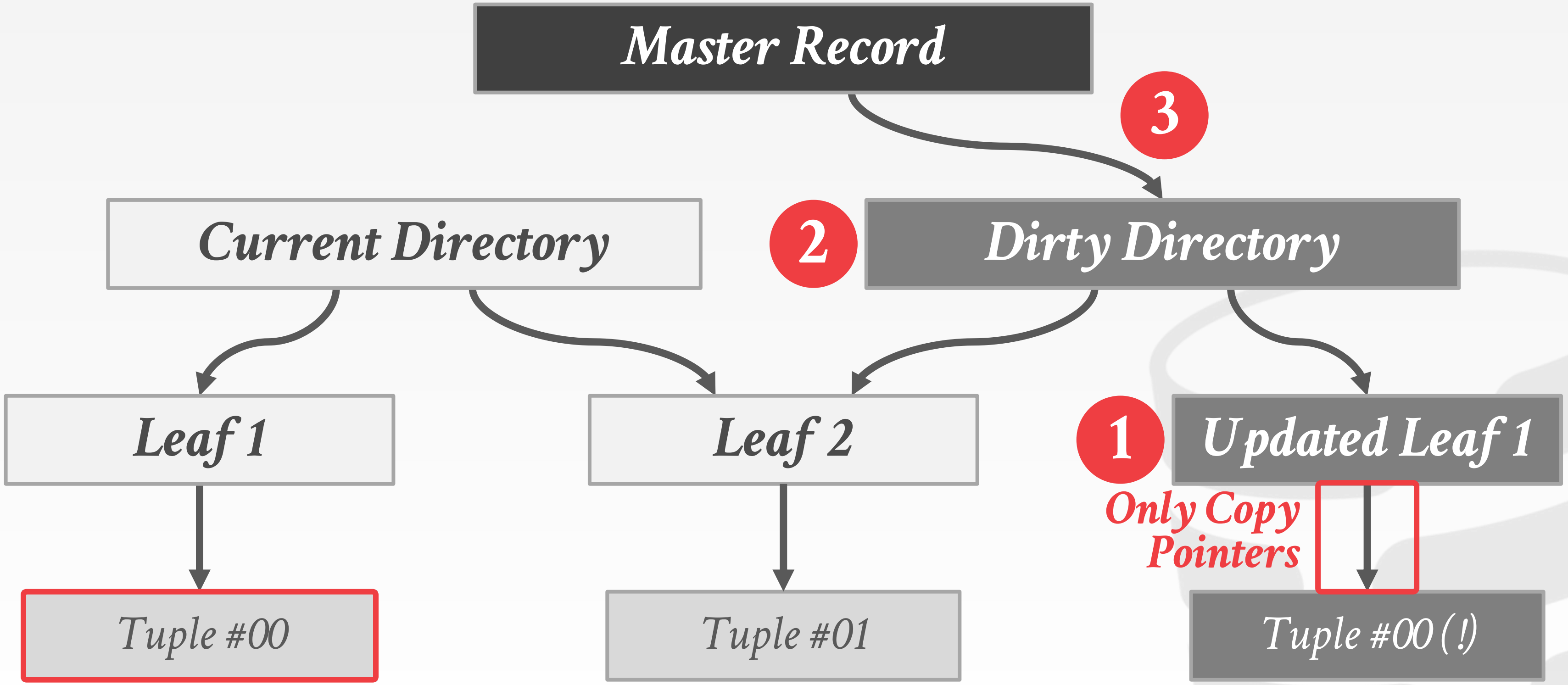
PM COPY-ON-WRITE ENGINE



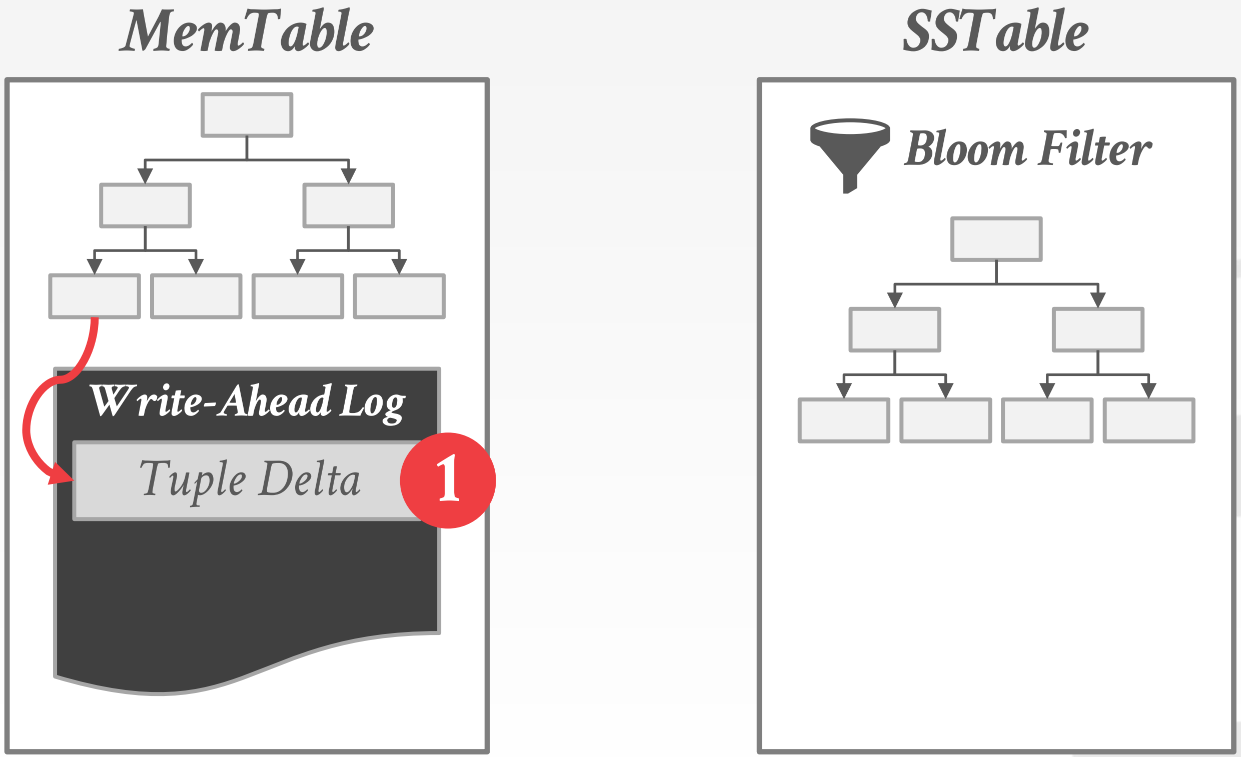
PM COPY-ON-WRITE ENGINE



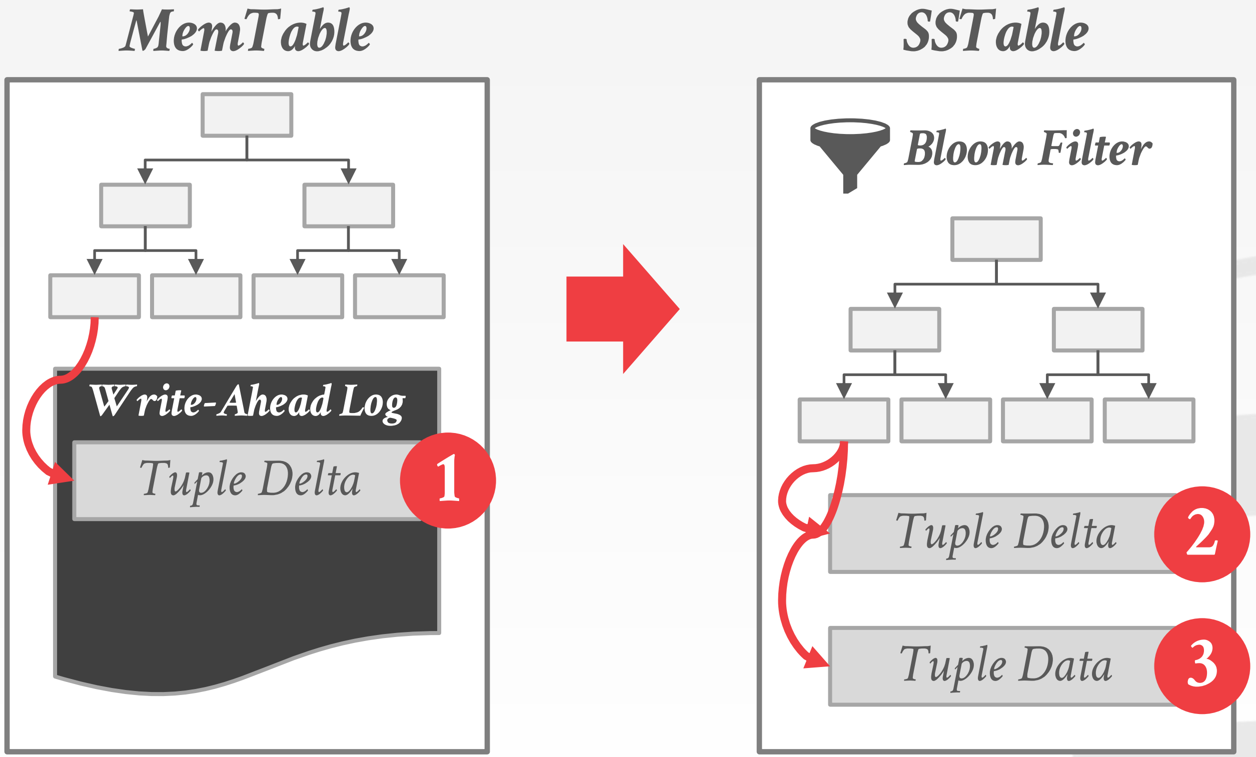
PM COPY-ON-WRITE ENGINE



LOG-STRUCTURED ENGINE



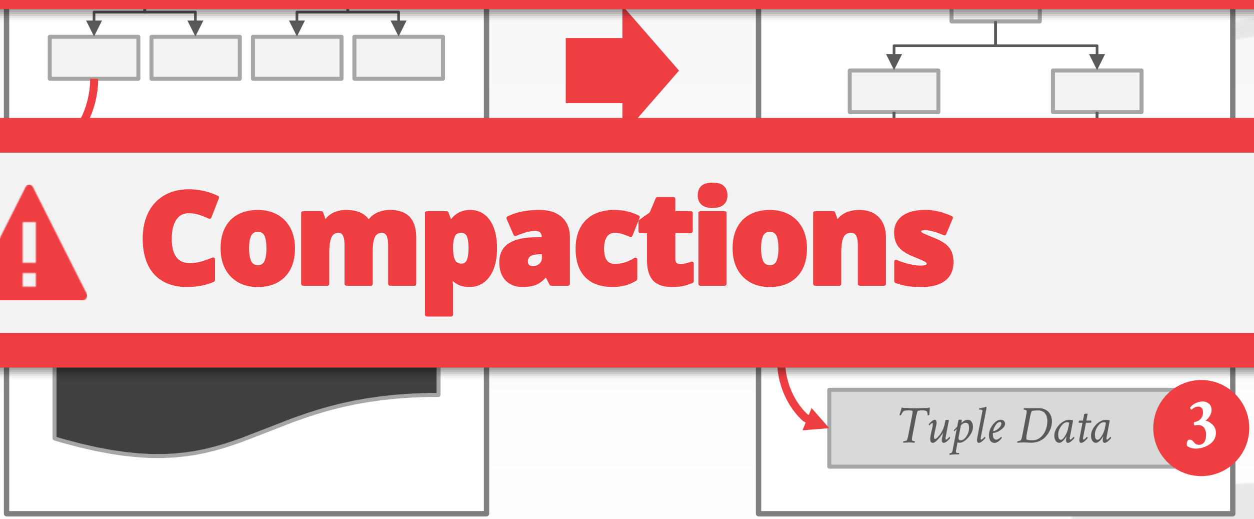
LOG-STRUCTURED ENGINE



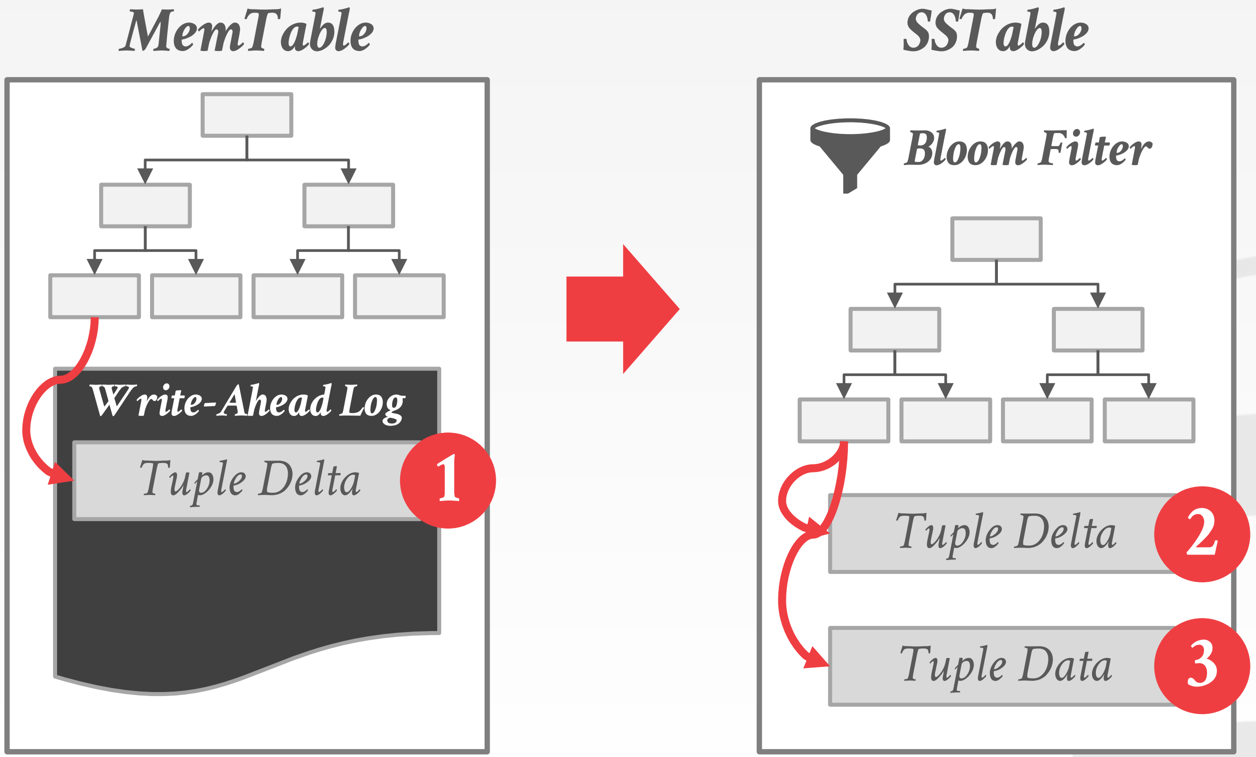
LOG-STRUCTURED ENGINE

⚠ **Duplicate Data**

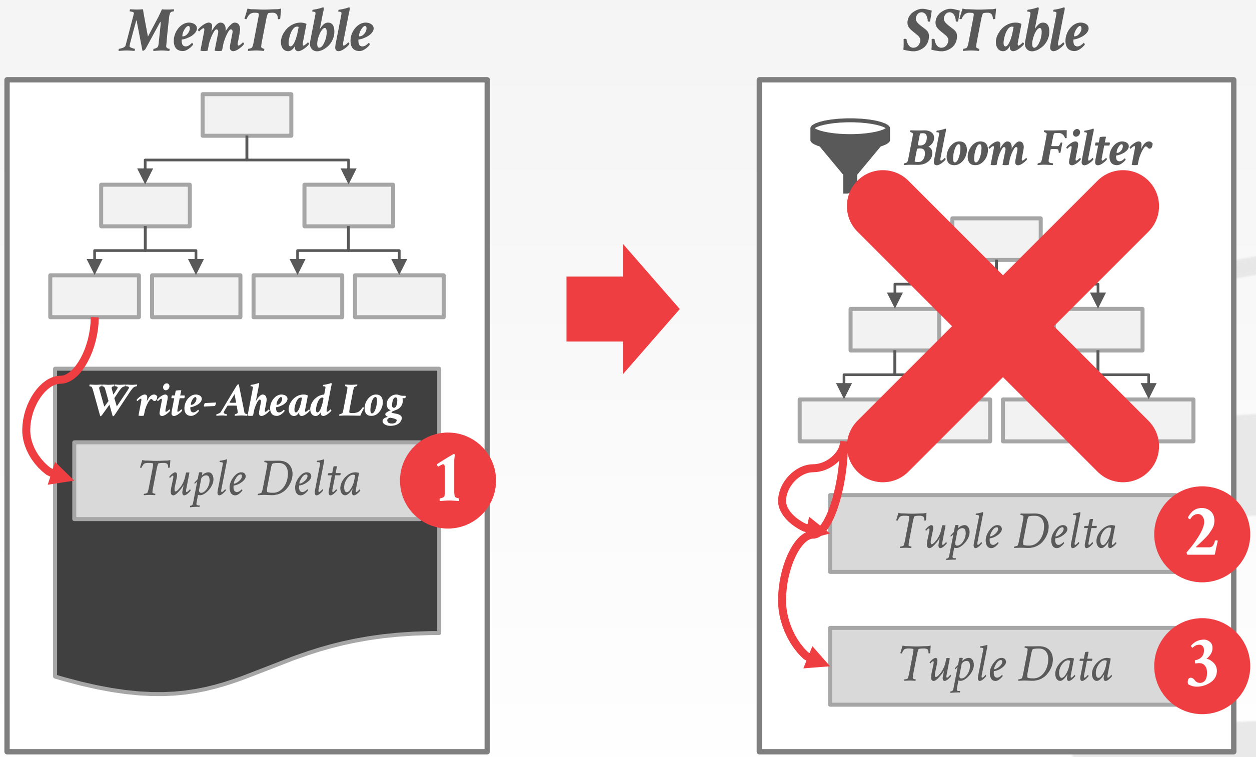
⚠ **Compactions**



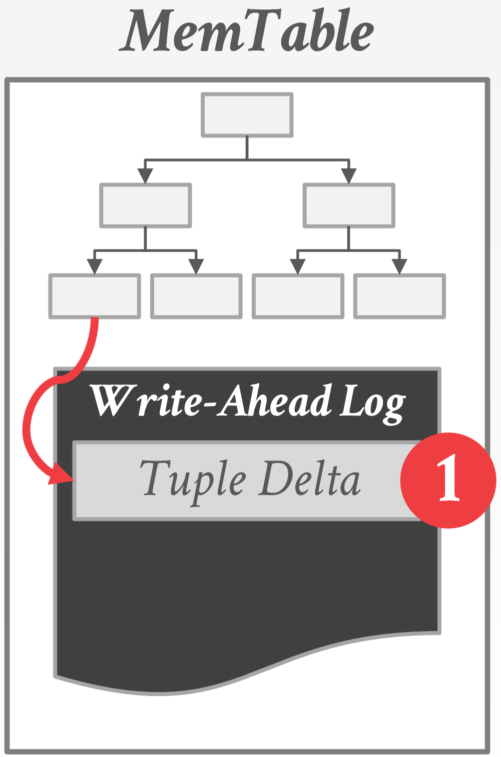
PM LOG-STRUCTURED ENGINE



PM LOG-STRUCTURED ENGINE



PM LOG-STRUCTURED ENGINE



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.

Recover the database almost instantaneously.

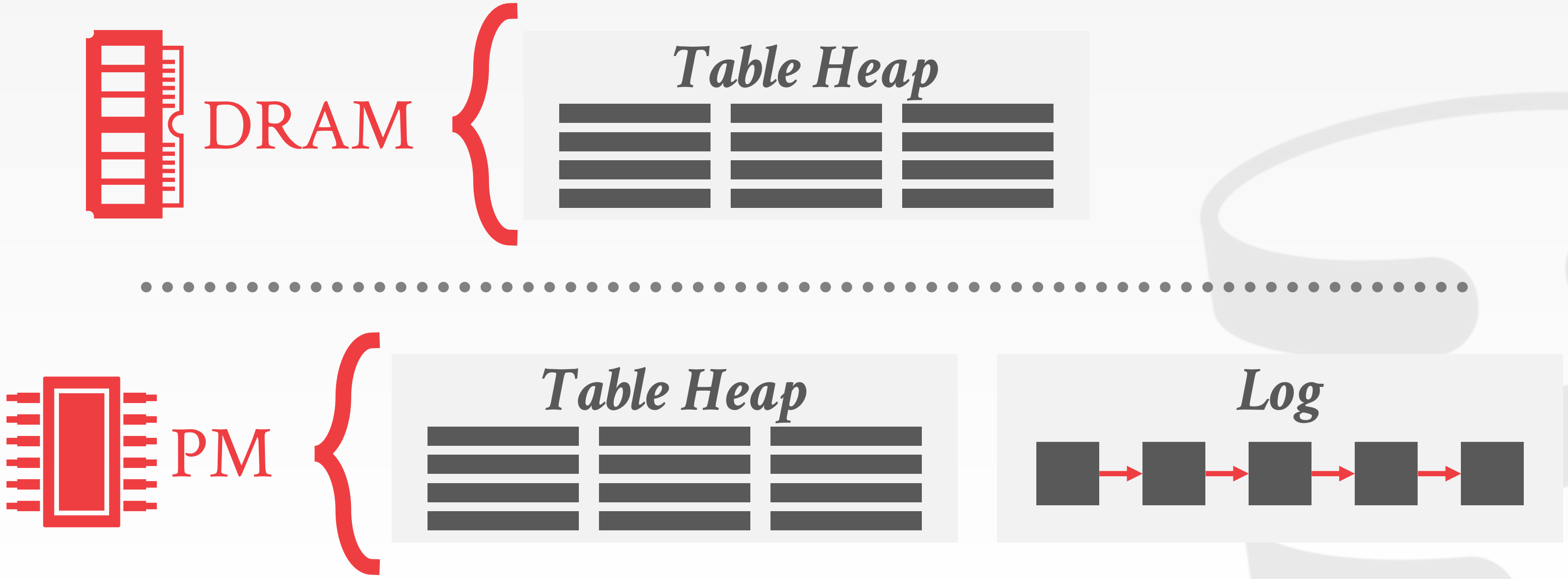
→ Need to record meta-data about in-flight transactions.

→ In case of failure, ignore their effects.

WRITE-BEHIND LOGGING
VLDB 2017

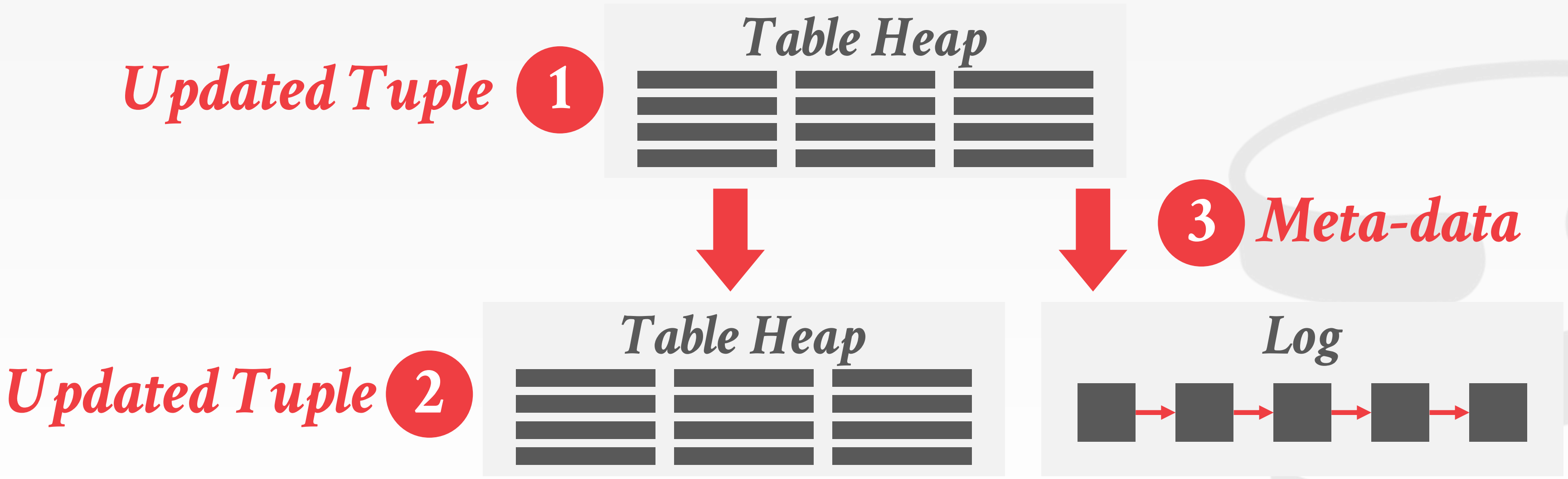
WRITE-BEHIND LOGGING

```
UPDATE table SET val=ABC
 WHERE id=123
```



WRITE-BEHIND LOGGING

```
UPDATE table SET val=ABC
 WHERE id=123
```



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.

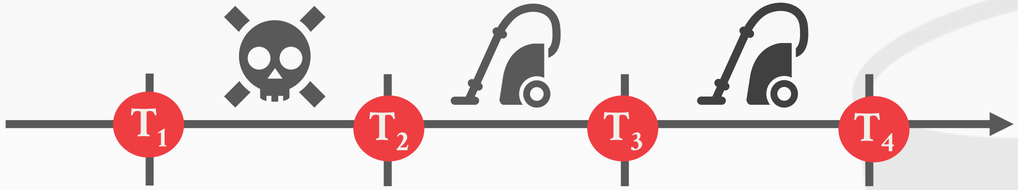
WRITE-BEHIND LOGGING

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

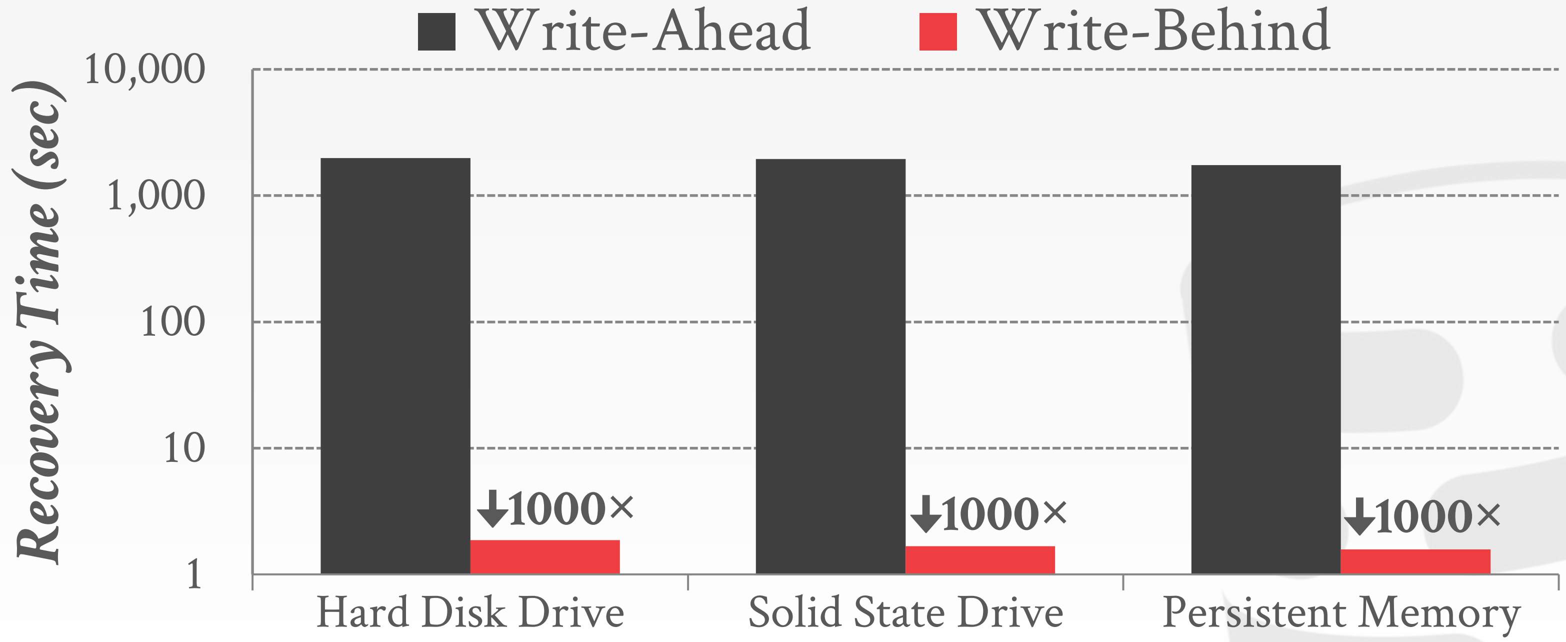
Use group commit timestamp range to ignore effects of transactions in failed group commit.
→ Maintain list of failed timestamp ranges.



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)	

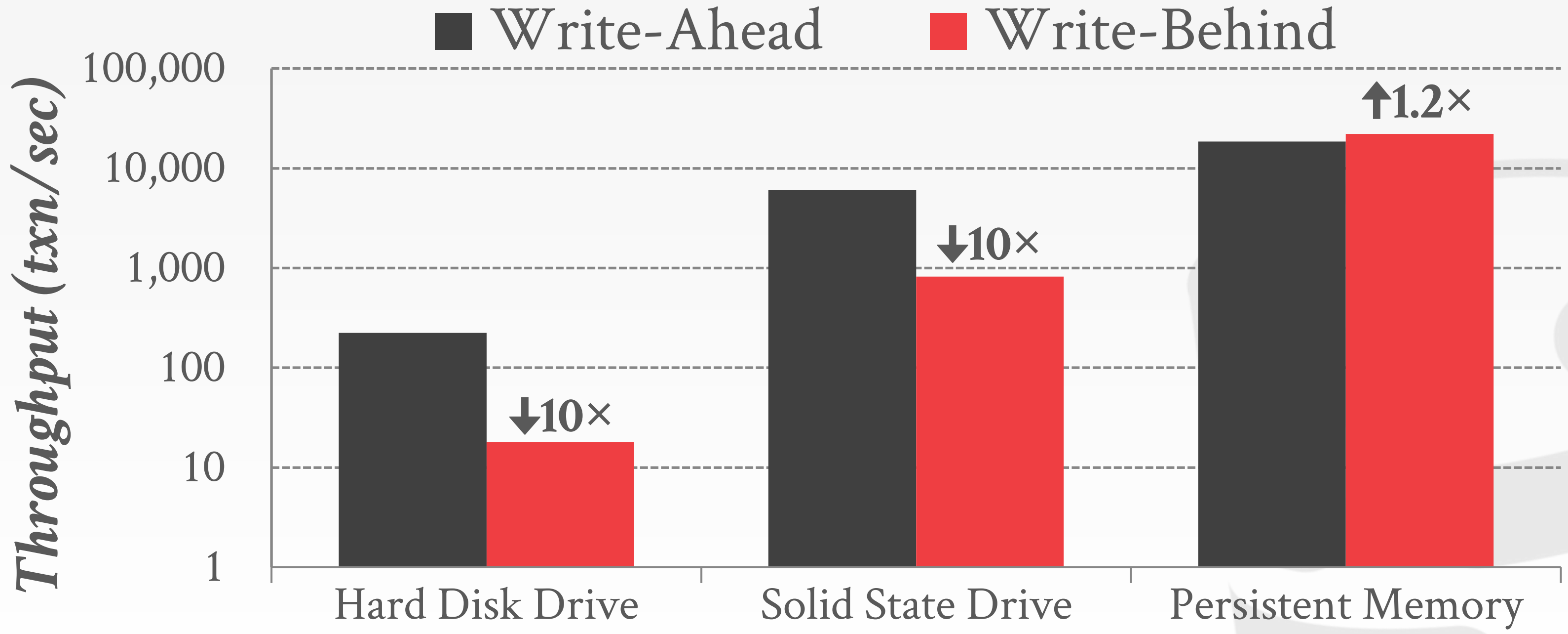
WRITE-BEHIND LOGGING – RECOVERY

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



PM SUMMARY

Storage Optimizations

→ Leverage byte-addressability to avoid unnecessary data duplication.

Recovery Optimizations

→ PM-optimized recovery protocols avoid the overhead of processing a log.

→ Non-volatile data structures ensure consistency.

GPU ACCELERATION

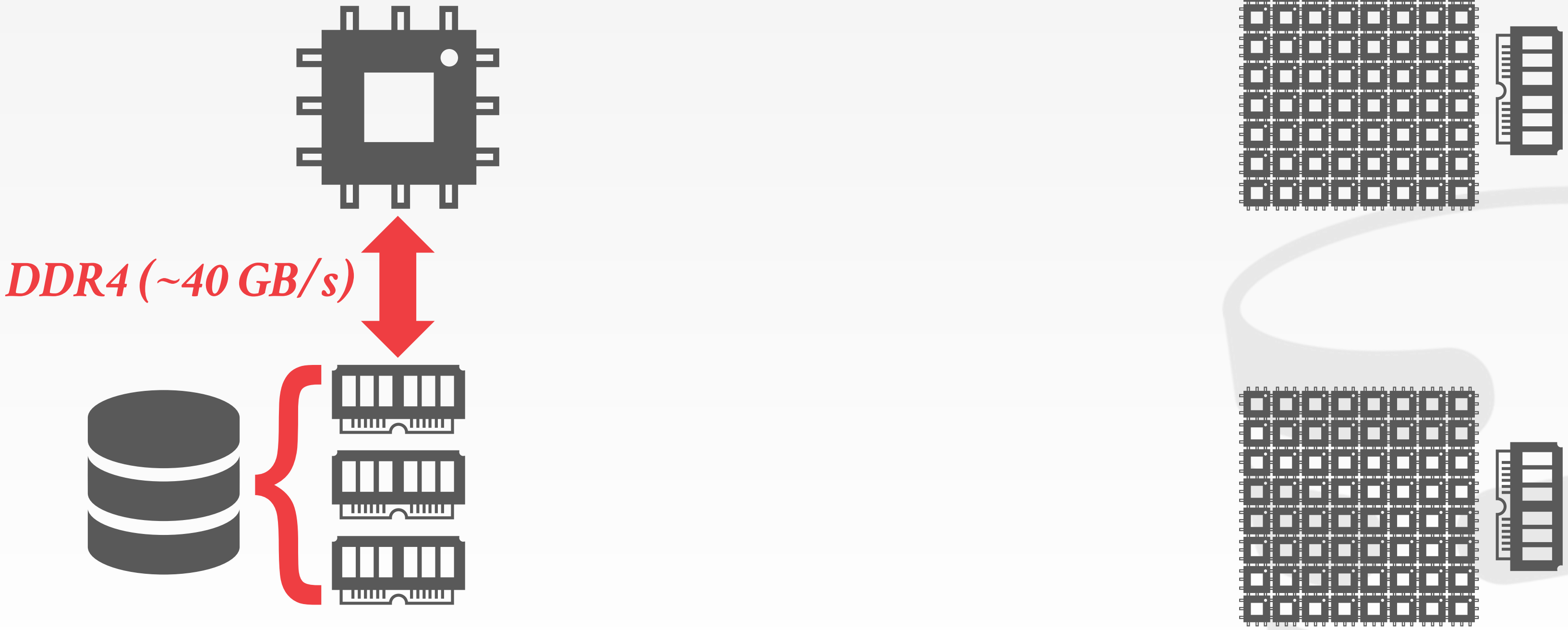
GPUs excel at performing (relatively simple) repetitive operations on large amounts of data over multiple streams of data.

Target operations that do not require blocking for input or branches:
→ Good: Sequential scans with predicates
→ Bad: B+Tree index probes

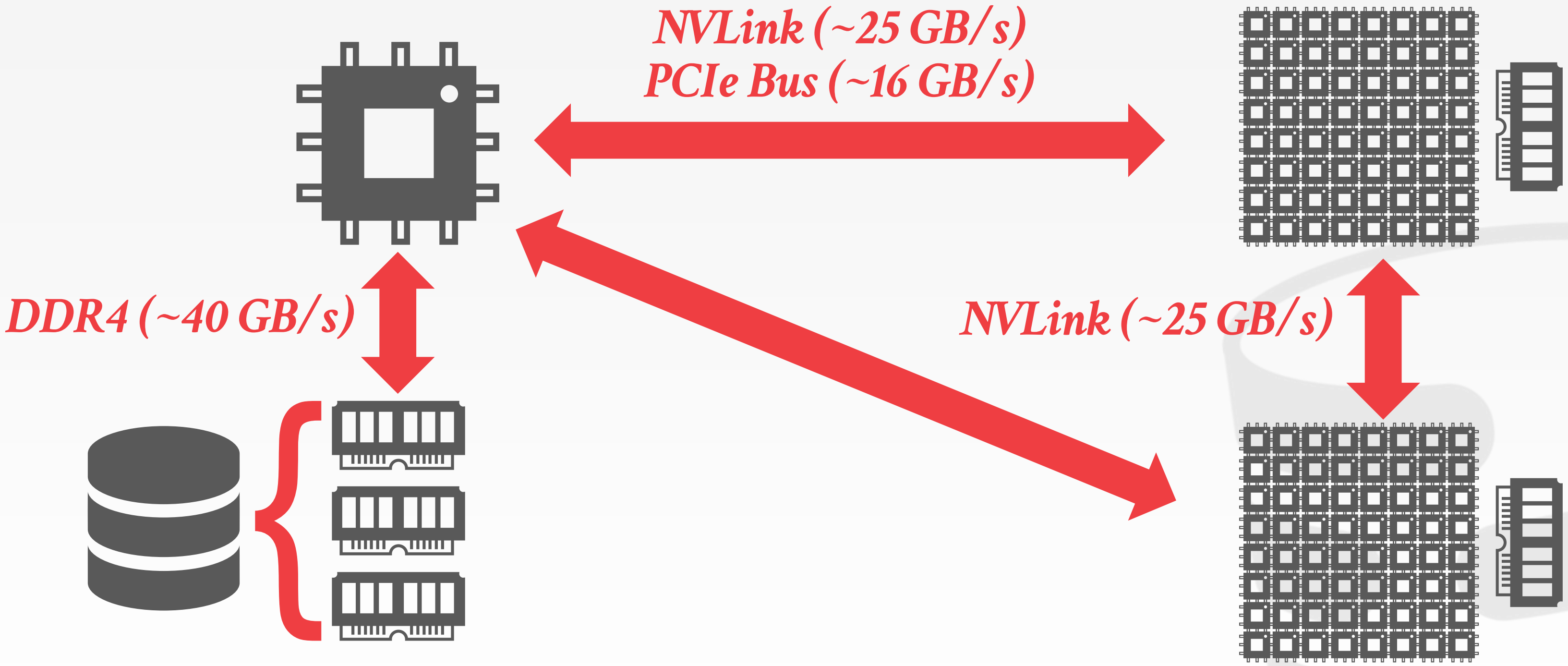
AFAIK, GPU memory is **not** cache coherent with CPU memory.

GPU ACCELERATION

GPU ACCELERATION



GPU ACCELERATION



GPU ACCELERATION

Choice #1: Entire Database

→ Store the database in the GPU(s) VRAM.

→ All queries perform massively parallel seq scans.

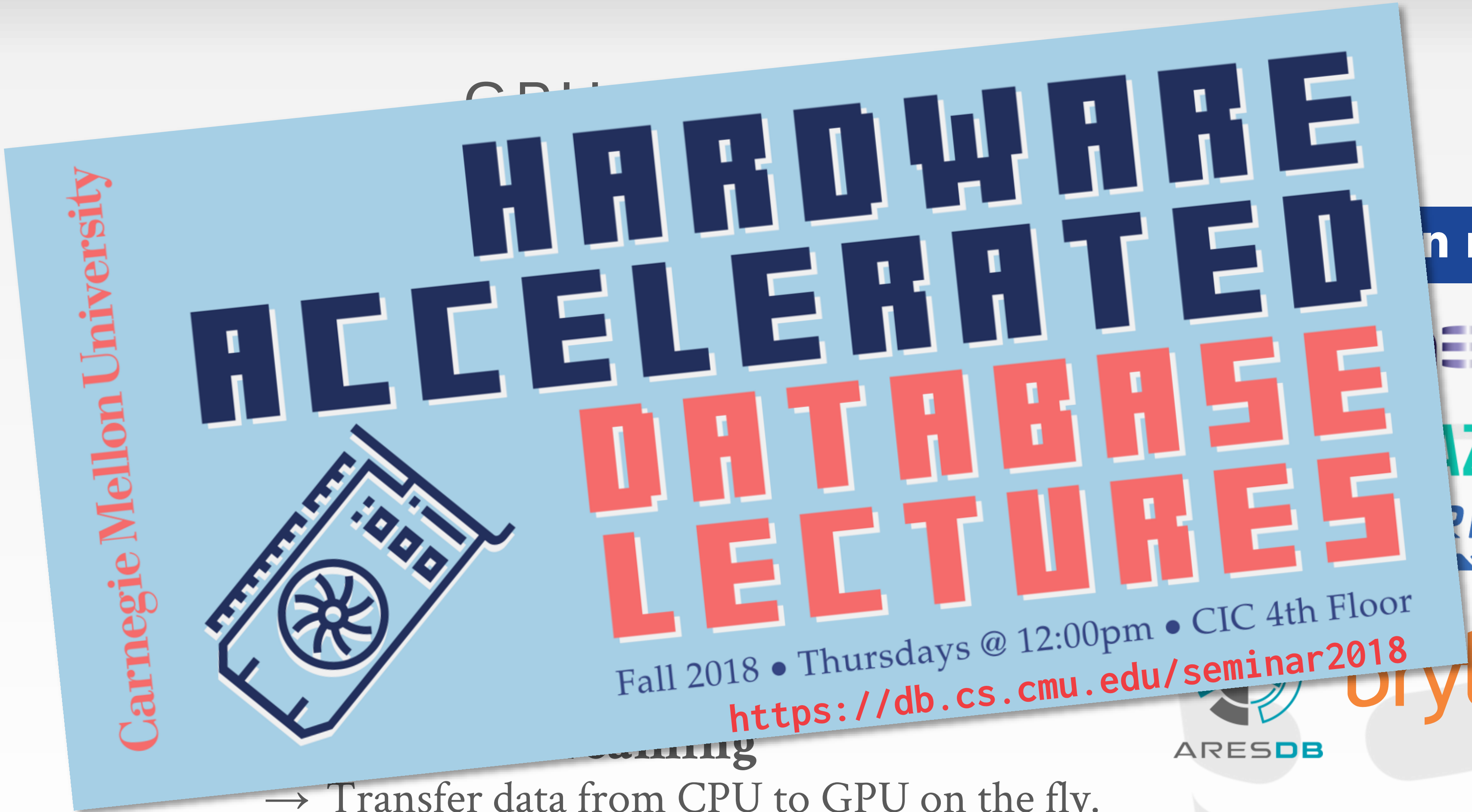
Choice #2: Important Columns

→ Return the offsets of records that match the portion of the query that accesses GPU-resident columns.

→ Must materialize full results in CPU.

Choice #3: Streaming

→ Transfer data from CPU to GPU on the fly.



Carnegie Mellon University

HARDWARE ACCELERATED DATABASE LECTURES

Fall 2018 • Thursdays @ 12:00pm • CIC 4th Floor

https://db.cs.cmu.edu/seminar2018

→ Transfer data from CPU to GPU on the fly.

HARDWARE TRANSACTIONAL MEMORY

Create critical sections in software that are managed by hardware.

→ Leverages same cache coherency protocol to detect transaction conflicts.

→ Intel x86: Transactional Synchronization Extensions

Read/write set of transactions must fit in L1 cache.

→ This means that it is not useful for general purpose txns.

→ It can be used to create latch-free indexes.

TO LOCK, SWAP OR ELIDE: ON THE INTERPLAY OF HARDWARE TRANSACTIONAL MEMORY AND LOCK-FREE INDEXING
VLDB 2015

HTM PROGRAMMING MODEL

Hardware Lock Elision (HLE)

→ Optimistically execute critical section by ***eliding*** the write to a lock so that it appears to be free to other threads.

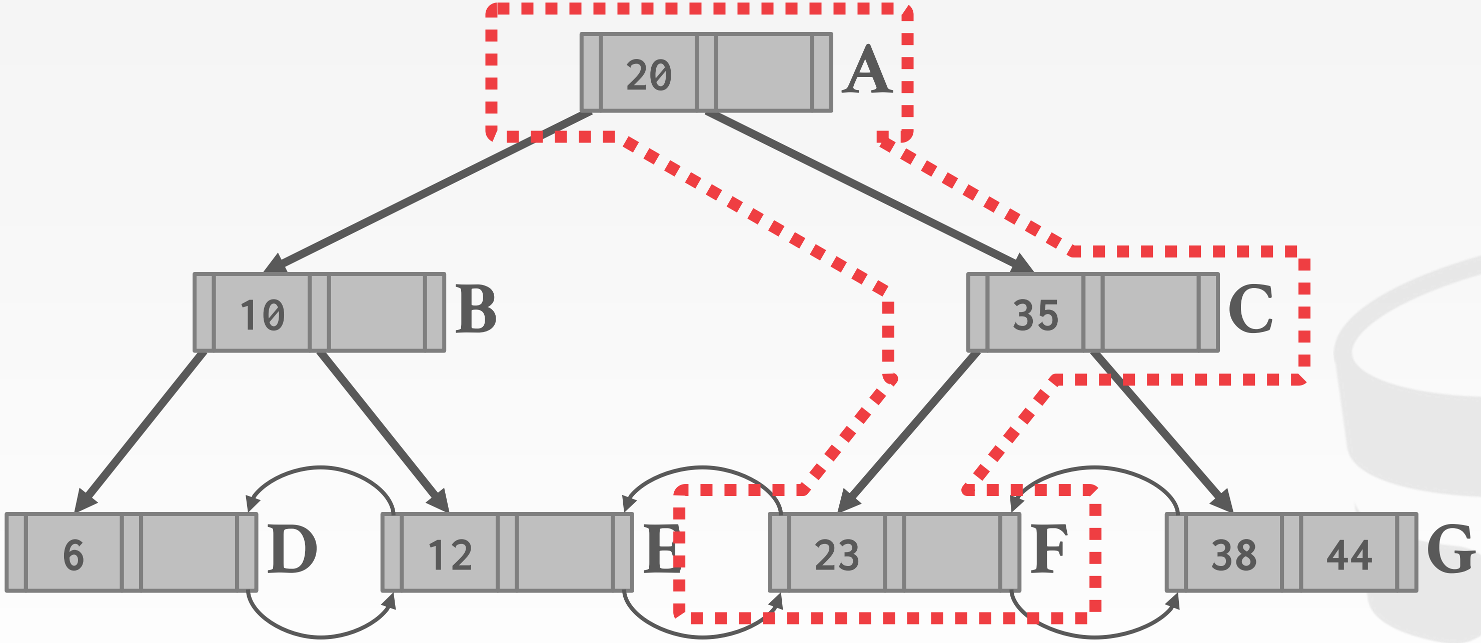
→ If there is a conflict, re-execute the code but take locks the second time.

Restricted Transactional Memory (RTM)

→ Like HLE but with an optional fallback codepath that the CPU jumps to if the txn aborts.

HTM LATCH ELISION

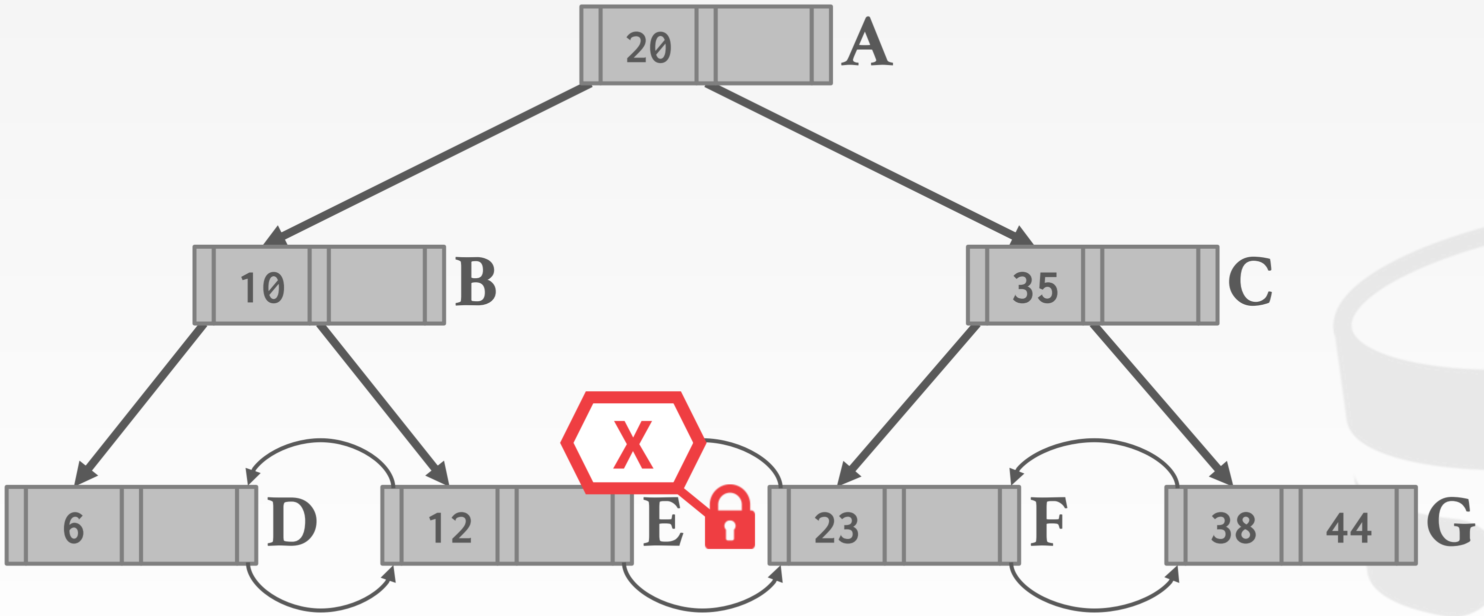
Insert Key 25



```
TSX-START {
  LATCH A
  Read A
  LATCH C
  UNLATCH A
  Read C
  LATCH F
  UNLATCH C
}
TSX-COMMIT
Insert 25
UNLATCH F
```

HTM LATCH ELISION

Insert Key 25



```
TSX-START {
  LATCH A
  Read A
  LATCH C
  UNLATCH A
  Read C
  LATCH F
  UNLATCH C
}
TSX-COMMIT
Insert 25
UNLATCH F
```

PARTING THOUGHTS

Byte-addressable PM is going to be a game changer when it comes out.

We are likely to see many new computational components that DBMSs can use in the next decade.

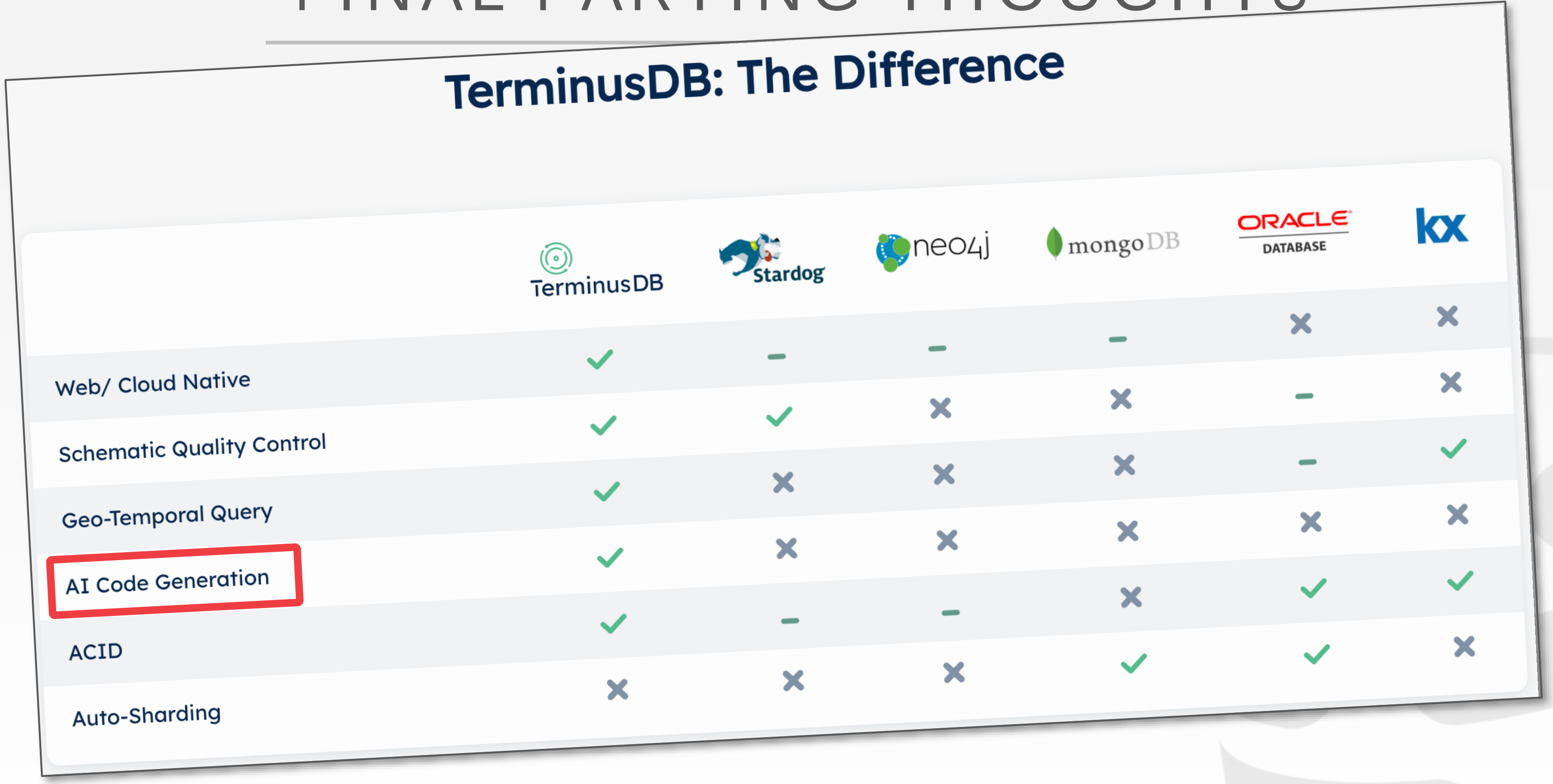
→ The core ideas / algorithms will still be the same.

FINAL PARTING THOUGHTS

You now are aware of the major topics involved in building a modern, single-node DBMS.

You have a foundation for reasoning about systems in order to discern whether claims are legitimate or marketing hype.

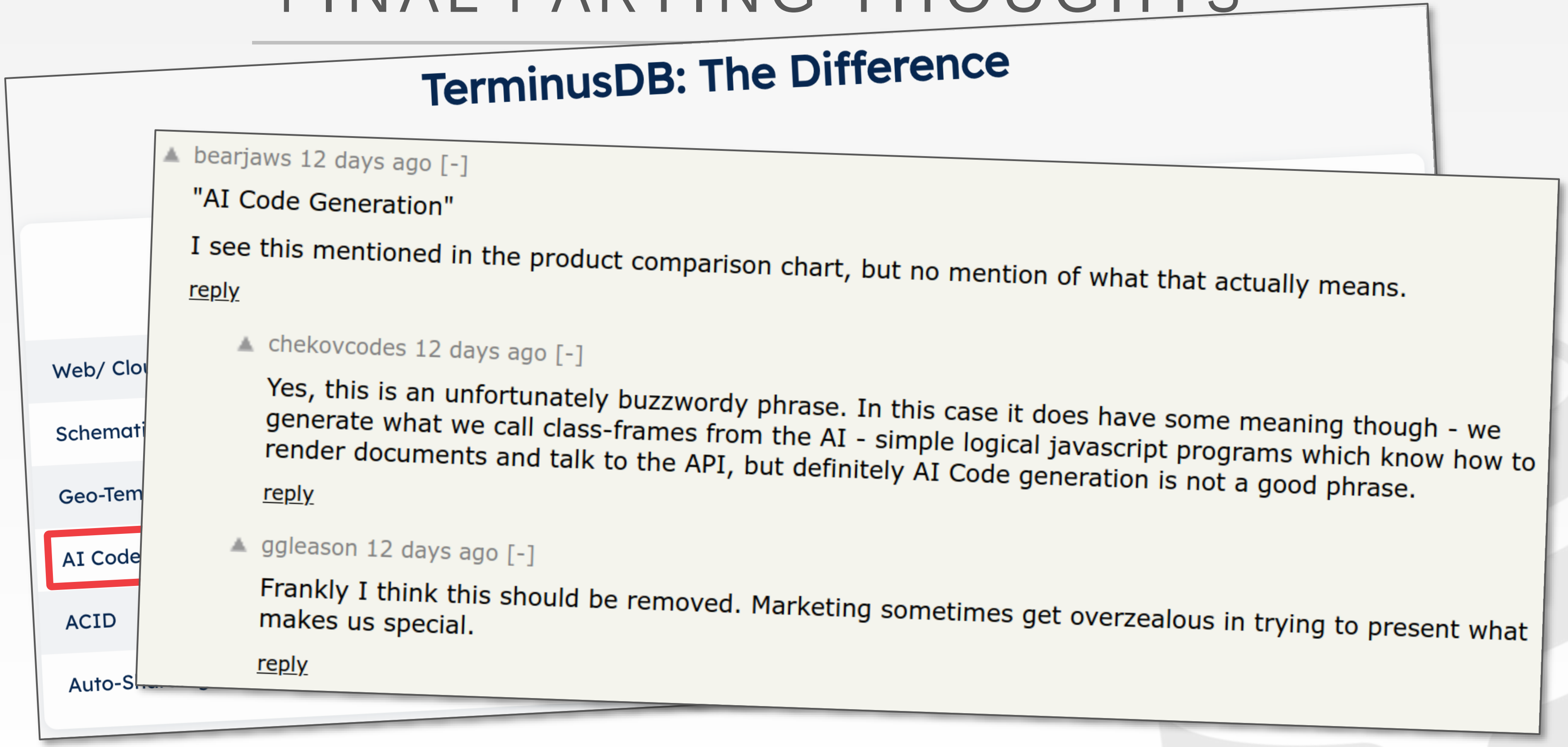
FINAL PARTING THOUGHTS



TerminusDB: The Difference

	TerminusDB	Stardog	neo4j	mongoDB	ORACLE DATABASE	kx
Web/ Cloud Native	✓	–	–	–	✗	✗
Schematic Quality Control	✓	✓	✗	✗	–	✗
Geo-Temporal Query	✓	✗	✗	✗	–	✓
AI Code Generation	✓	✗	✗	✗	✗	✗
ACID	✓	–	–	✗	✓	✓
Auto-Sharding	✗	✗	✗	✓	✓	✗

FINAL PARTING THOUGHTS



TerminusDB: The Difference

Web/ Clou
Schemati
Geo-Tem
AI Code
ACID
Auto-Sh

▲ bearjaws 12 days ago [-]

"AI Code Generation"

I see this mentioned in the product comparison chart, but no mention of what that actually means.

reply

▲ chekovcodes 12 days ago [-]

Yes, this is an unfortunately buzzwordy phrase. In this case it does have some meaning though - we generate what we call class-frames from the AI - simple logical javascript programs which know how to render documents and talk to the API, but definitely AI Code generation is not a good phrase.

reply

▲ ggleason 12 days ago [-]

Frankly I think this should be removed. Marketing sometimes get overzealous in trying to present what makes us special.

reply

NEXT CLASS

amazon