Lecture #23 – Non-Volatile Memory

Andy Pavlo // Carnegie Mellon University // Spring 2016

TODAY'S AGENDA

Background
Storage & Recovery Methods for NVM
Project #3 Code Review Guidelines

NON-VOLATILE 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, Persistent Memory

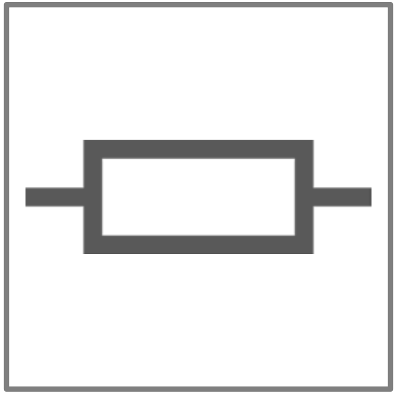
First devices will be block-addressable (NVMe)

Later devices will be byte-addressable.

FUNDAMENTAL ELEMENTS OF CIRCUITS

Capacitor* *(ca. 1745)

Resistor* *(ca. 1827)



Inductor* *(ca. 1831)

FUNDAMENTAL ELEMENTS OF CIRCUITS

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

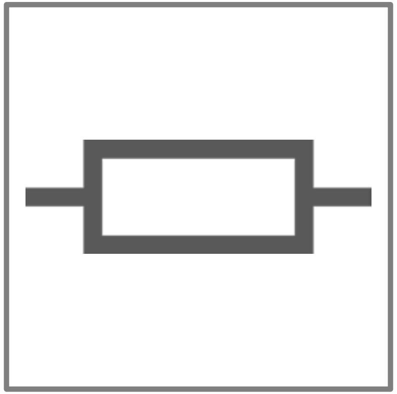
A two-terminal device whose resistance depends on the voltage applied to it, but when that voltage is turned off it permanently **remembers** its last resistive state.

TWO CENTURIES OF MEMRISTORS
Nature Materials 2012

FUNDAMENTAL ELEMENTS OF CIRCUITS

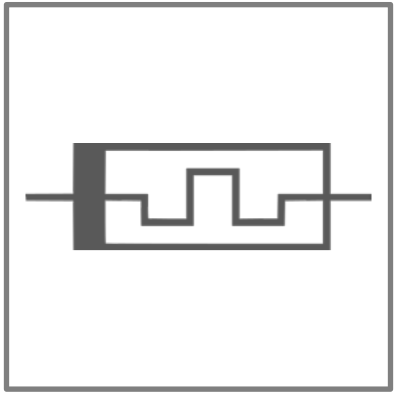
***Capacitor* (ca. 1745)**

***Resistor* (ca. 1827)**



***Inductor* (ca. 1831)**

***Memristor* (ca. 1971)**



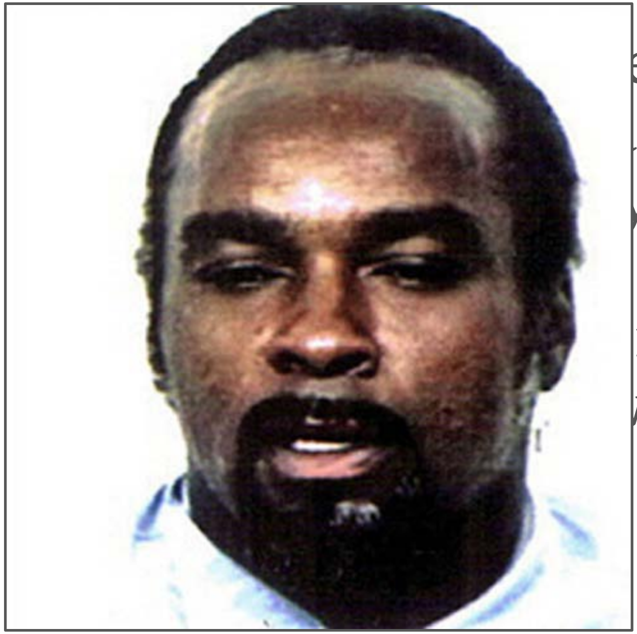
MERISTORS

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

MERISTORS



ed by Stanley Williams
no-device that had weird
y could not understand.

found Chua's 1971 paper
what they had invented.

HOW WE FOUND THE MISSING MEMRISTOR
IEEE Spectrum 2008

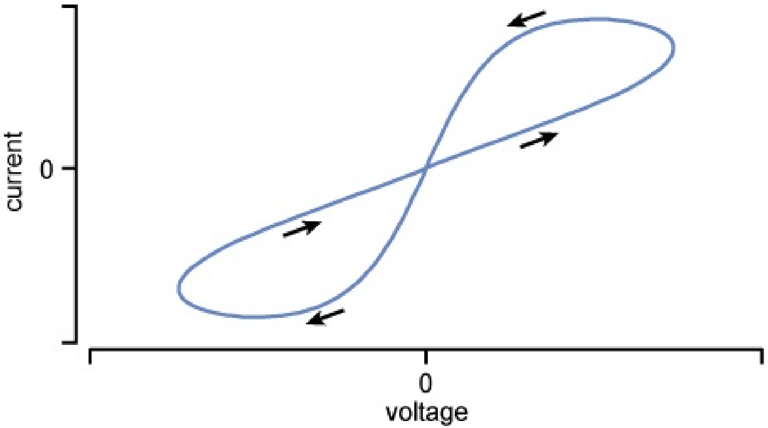
MERISTORS

ed by Sta
no-devic
could n

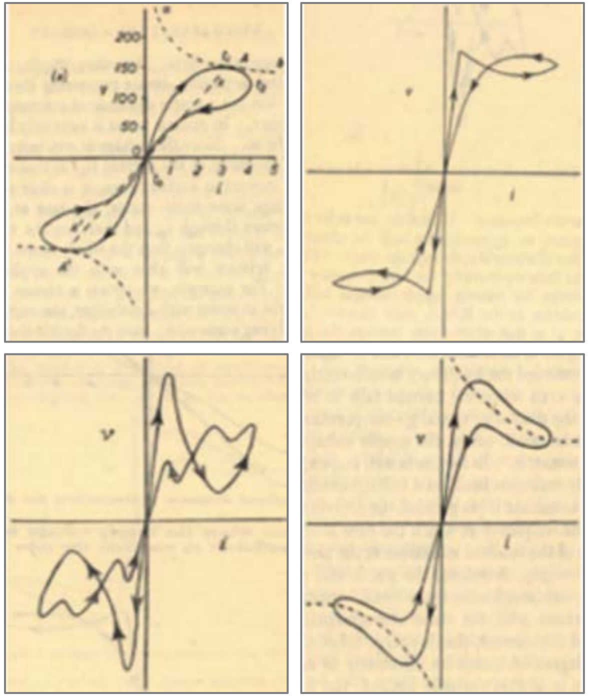
found Ch
hat they

HOW WE FOUND THE MISSING MEMRISTOR
IEEE Spectrum 2008

MEMRISTOR – HYSTERESIS LOOP



Vacuum Circuits (ca. 1948)



TWO CENTURIES OF MEMRISTORS
Nature Materials 2012

TECHNOLOGIES

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

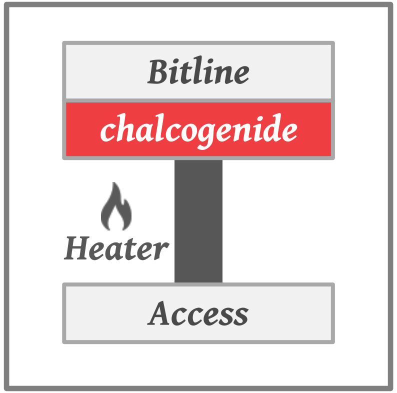
PHASE-CHANGE MEMORY

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’.

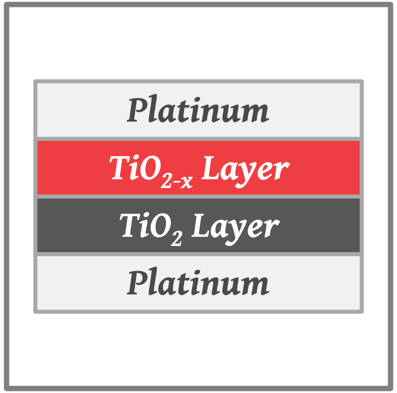


PHASE CHANGE MEMORY ARCHITECTURE AND THE QUEST FOR SCALABILITY
Communications of the ACM 2010

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.

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



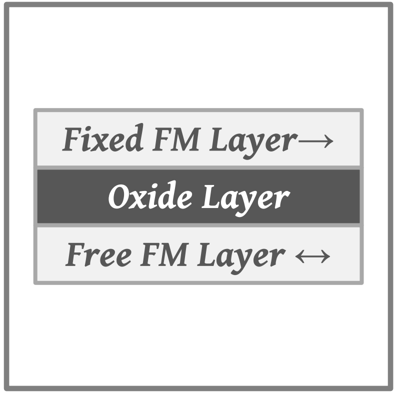
HOW WE FOUND THE MISSING MEMRISTOR
IEEE Spectrum 2008

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 NVM.

→ Supposedly able to scale to very small sizes (10nm) and have SRAM latencies.



SPIN MEMORY SHOWS ITS MIGHT
IEEE Spectrum 2014

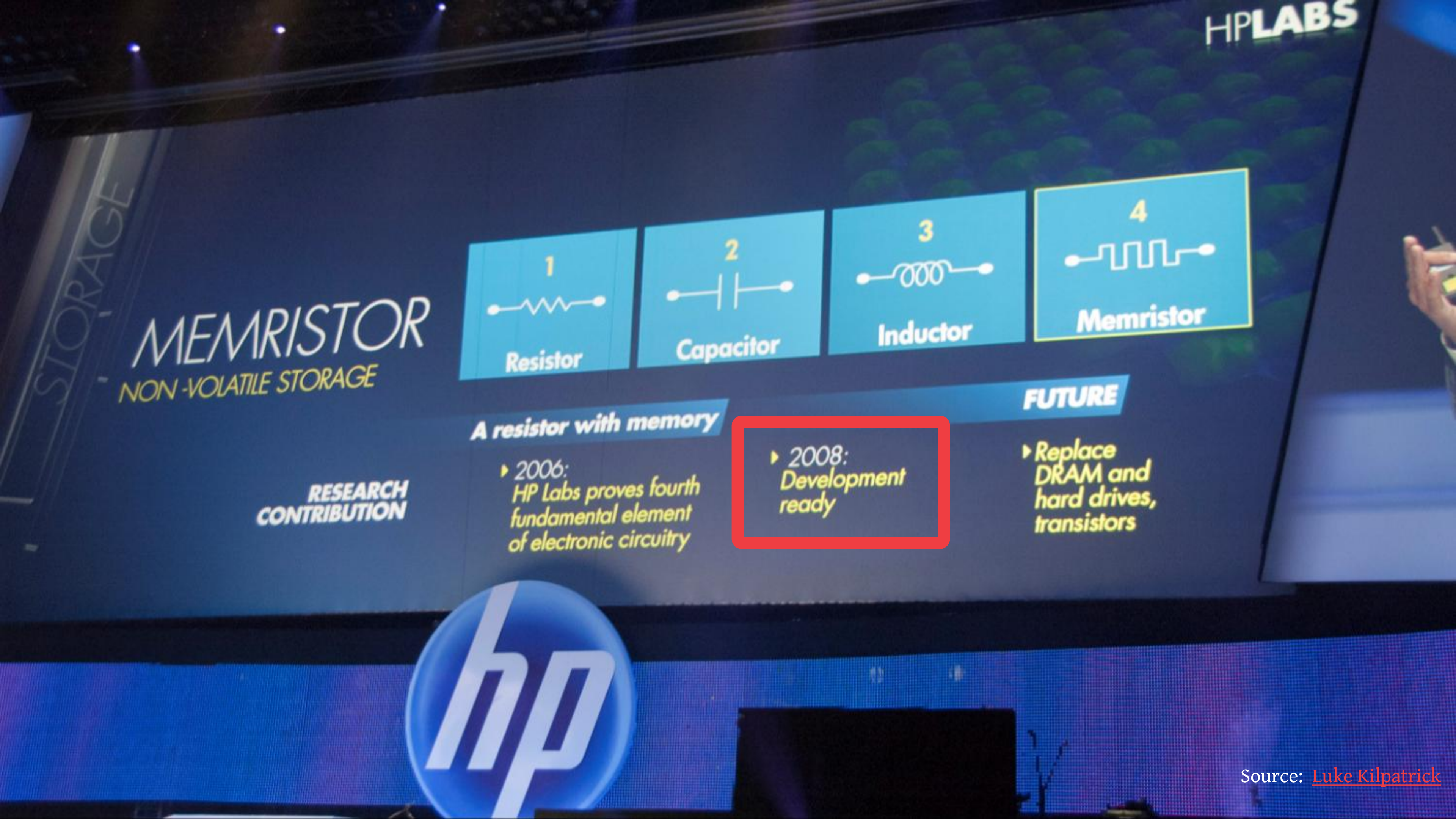
TIMELINE

Intel announced that their 3D XPoint drives will be available in 2016.

→ Rumors are that the 2017 Xeon ISA will include instructions for NVM DIMMs.

Samsung has recently partnered to develop their NVDIMM-P storage.

HP's ReRam is always two years away...



Source: [Luke Kilpatrick]()

NVM FOR DATABASE SYSTEMS

Block-addressable NVM is not that interesting.

Byte-addressable NVM will be a game changer but will require some work to use correctly.

→ In-memory DBMSs will be better positioned to use byte-addressable NVM.
→ Disk-oriented DBMSs will initially treat NVM as just a faster SSD.

More significant for OLTP workloads.

STORAGE & RECOVERY METHODS

Understand how a DBMS will behave on a system that only has byte-addressable NVM.

Develop NVM-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.

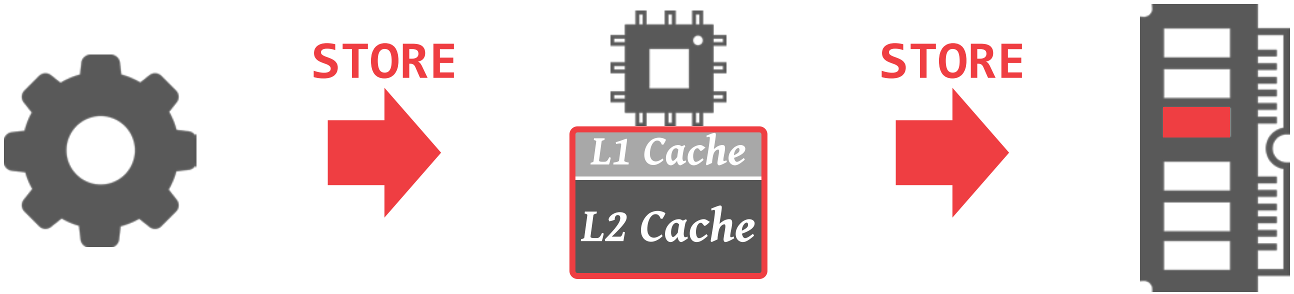
The DBMS needs a way to ensure that data is flushed from caches to NVM.

SYNCHRONIZATION

Existing programming models assume that any write to memory is non-volatile.

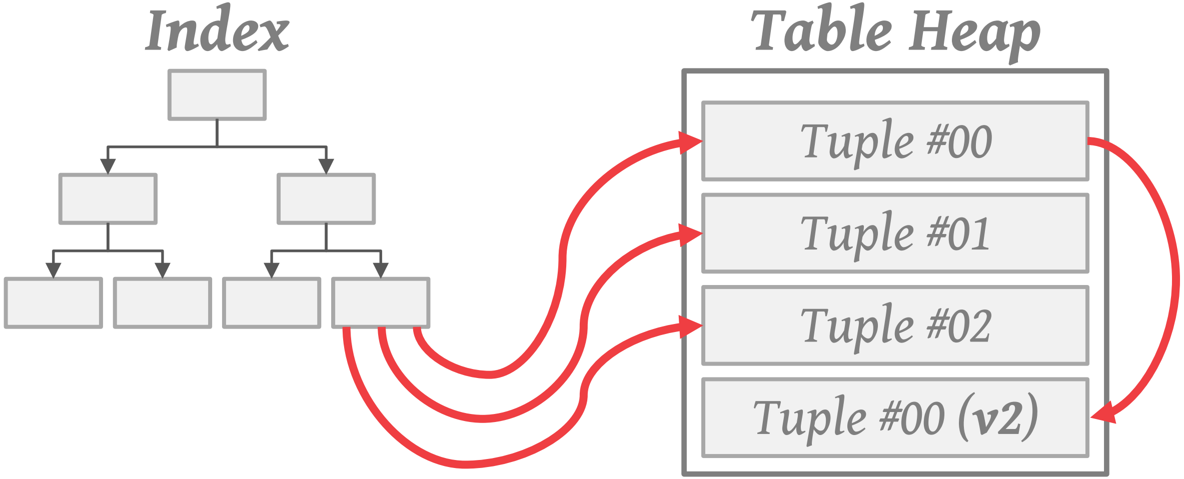
→ CPU decides when to move data from caches to DRAM.

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



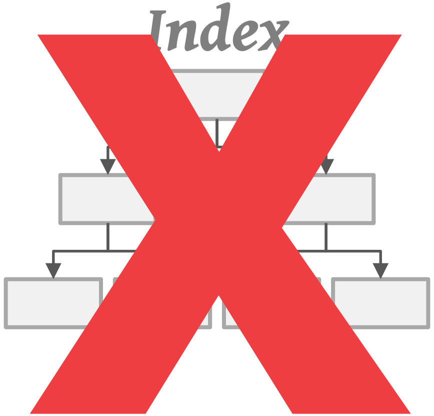
NAMING

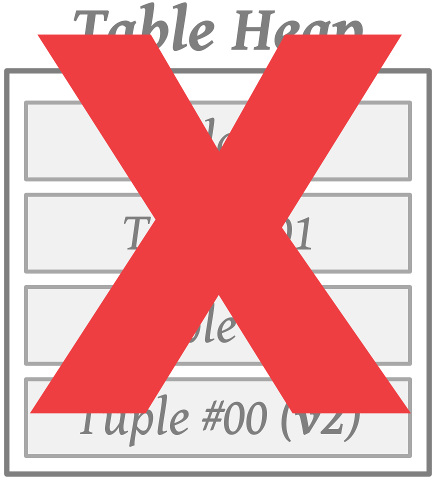
If the DBMS process restarts, we need to make sure that all of the pointers for in-memory data point to the same data.



NAMING

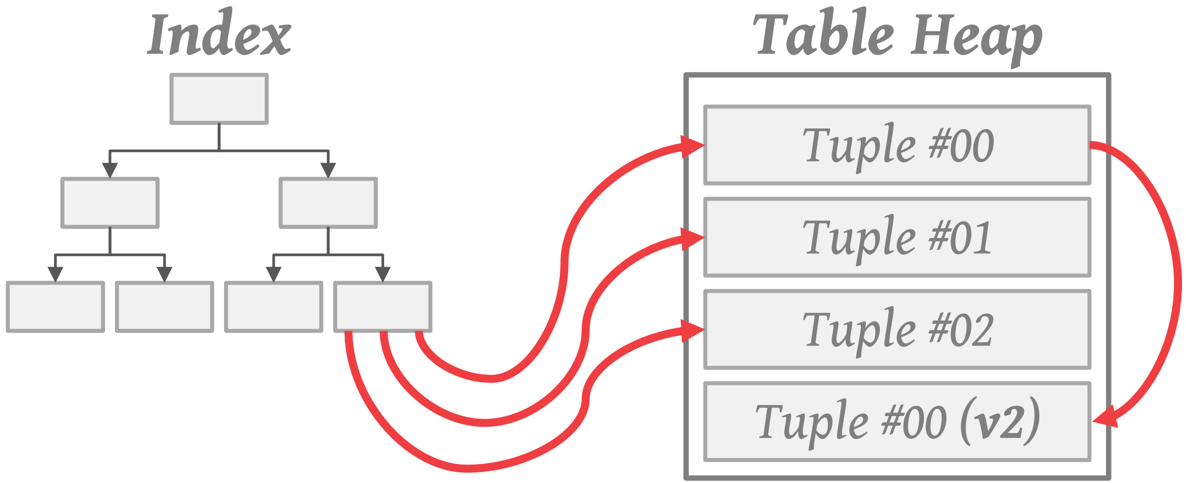
If the DBMS process restarts, we need to make sure that all of the pointers for in-memory data point to the same data.





NAMING

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



NVM-AWARE MEMORY ALLOCATOR

Feature #1: Synchronization

→ The allocator writes back CPU cache lines to NVM using the `CLFLUSH` instruction.

→ It then issues a `SFENCE` instruction to wait for the data to become durable on NVM.

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.

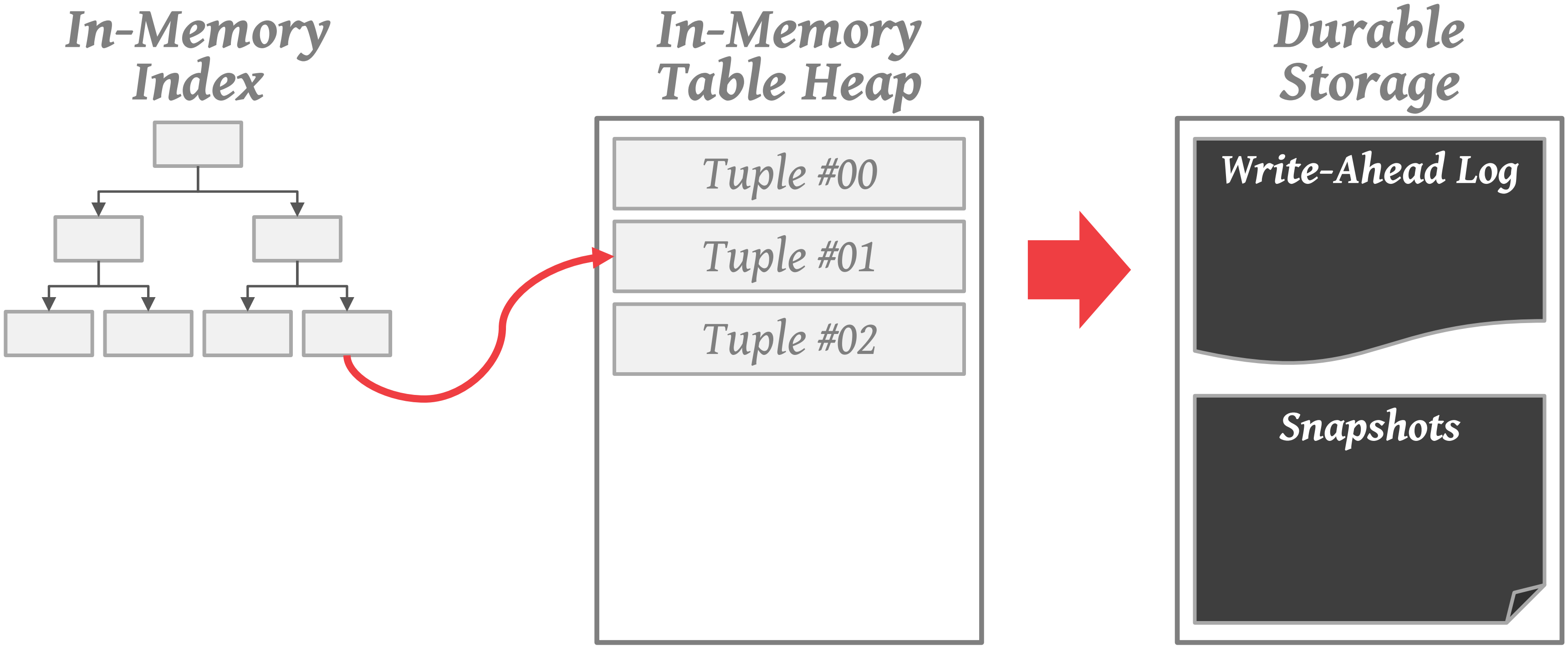
→ Example: LMDB

Choice #3: Log-structured

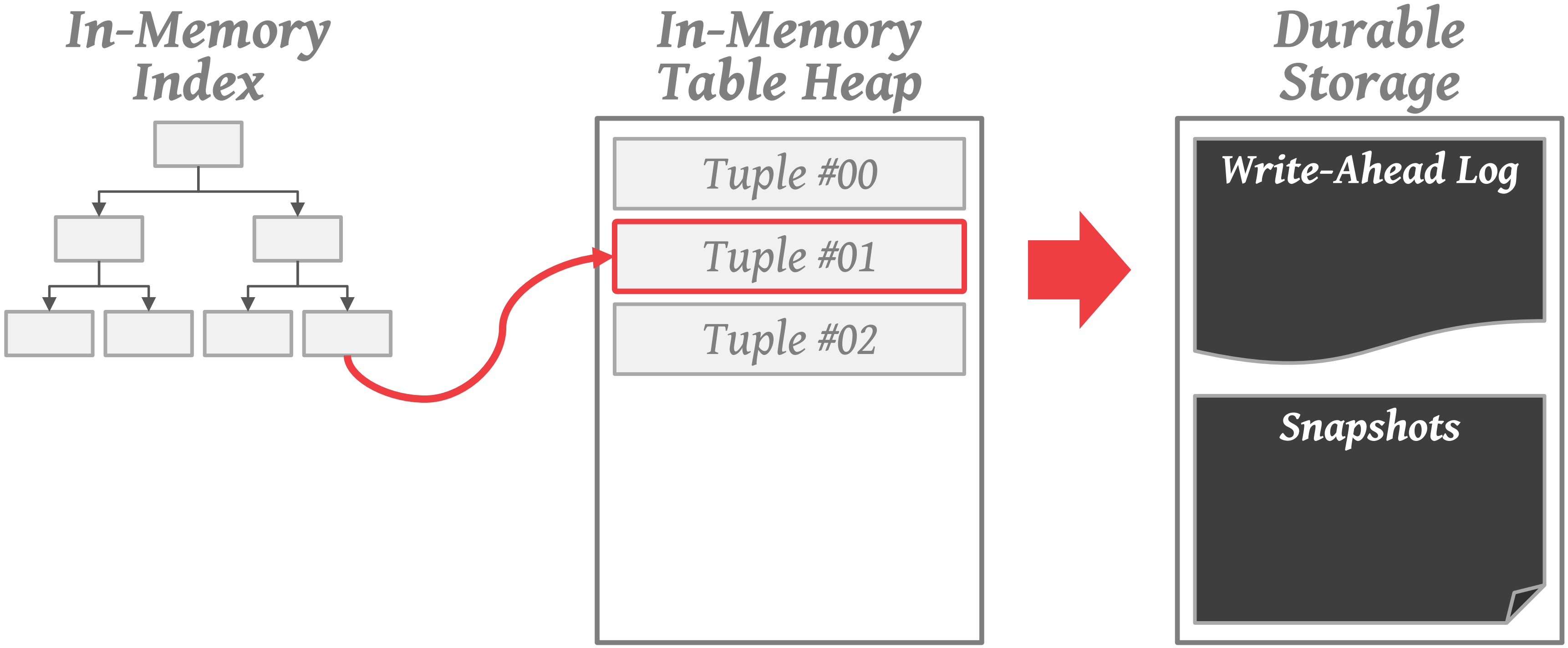
→ All writes are appended to log. No table heap.

→ Example: RocksDB

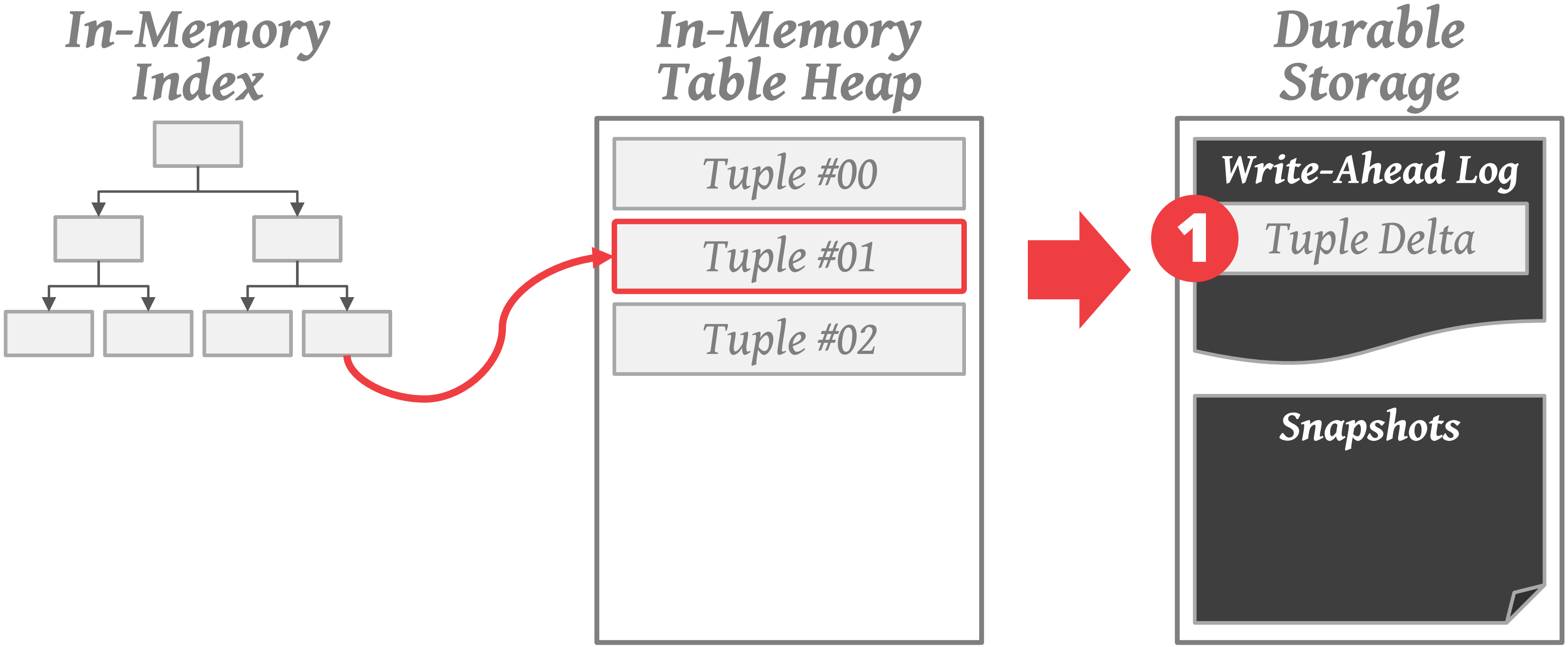
IN-PLACE UPDATES ENGINE



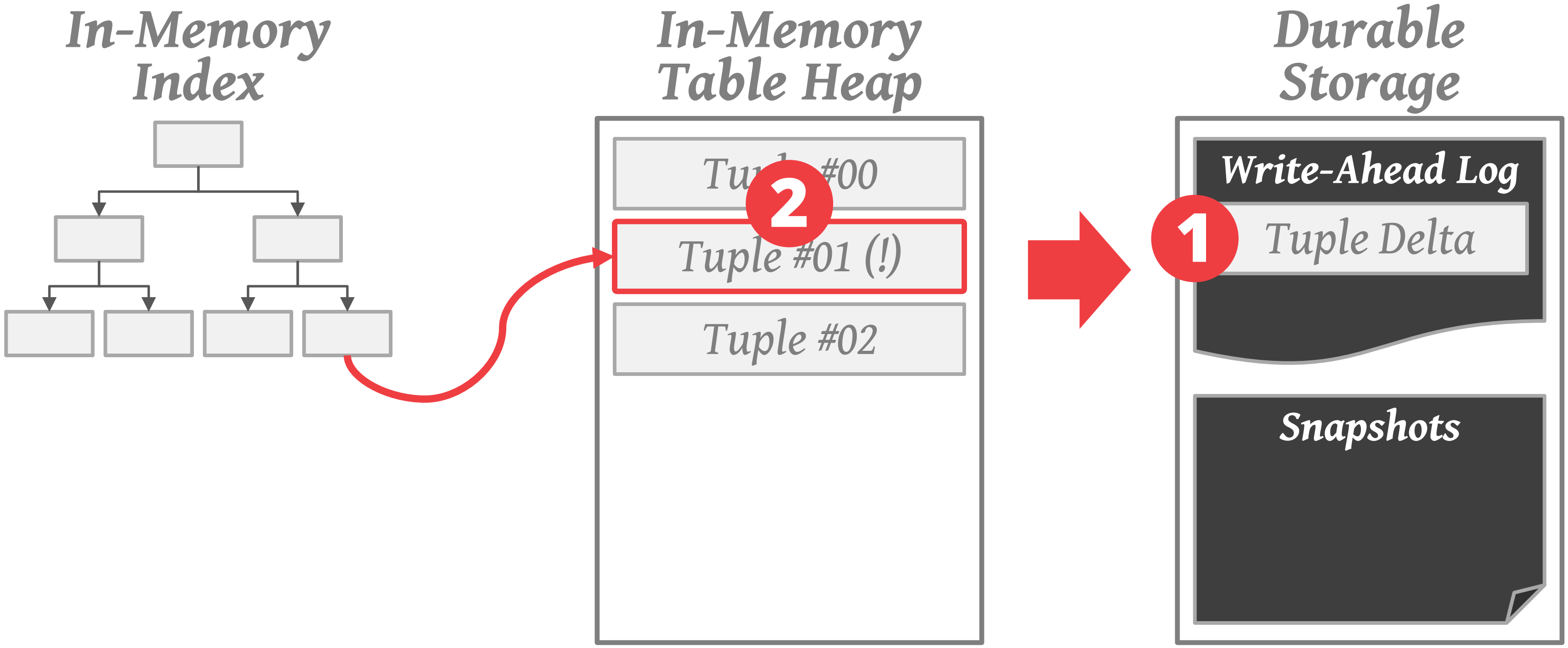
IN-PLACE UPDATES ENGINE



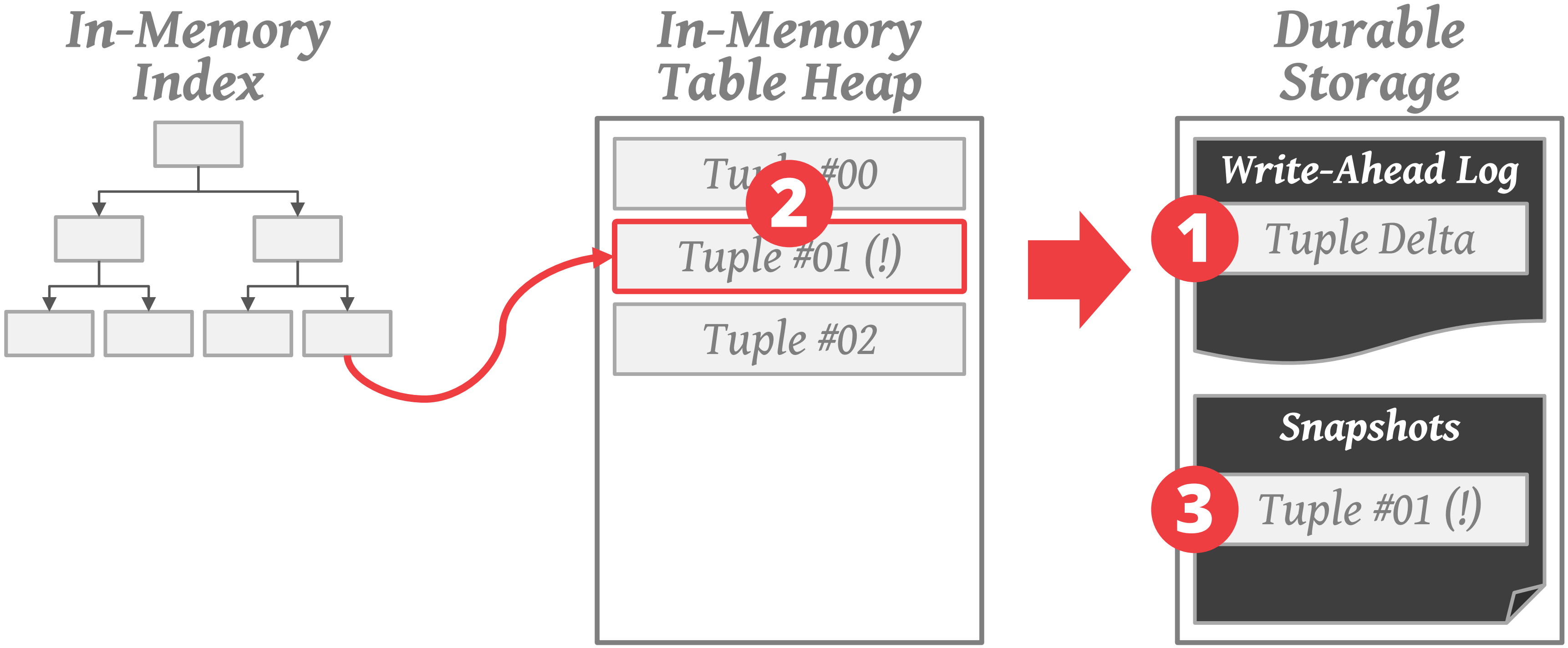
IN-PLACE UPDATES ENGINE



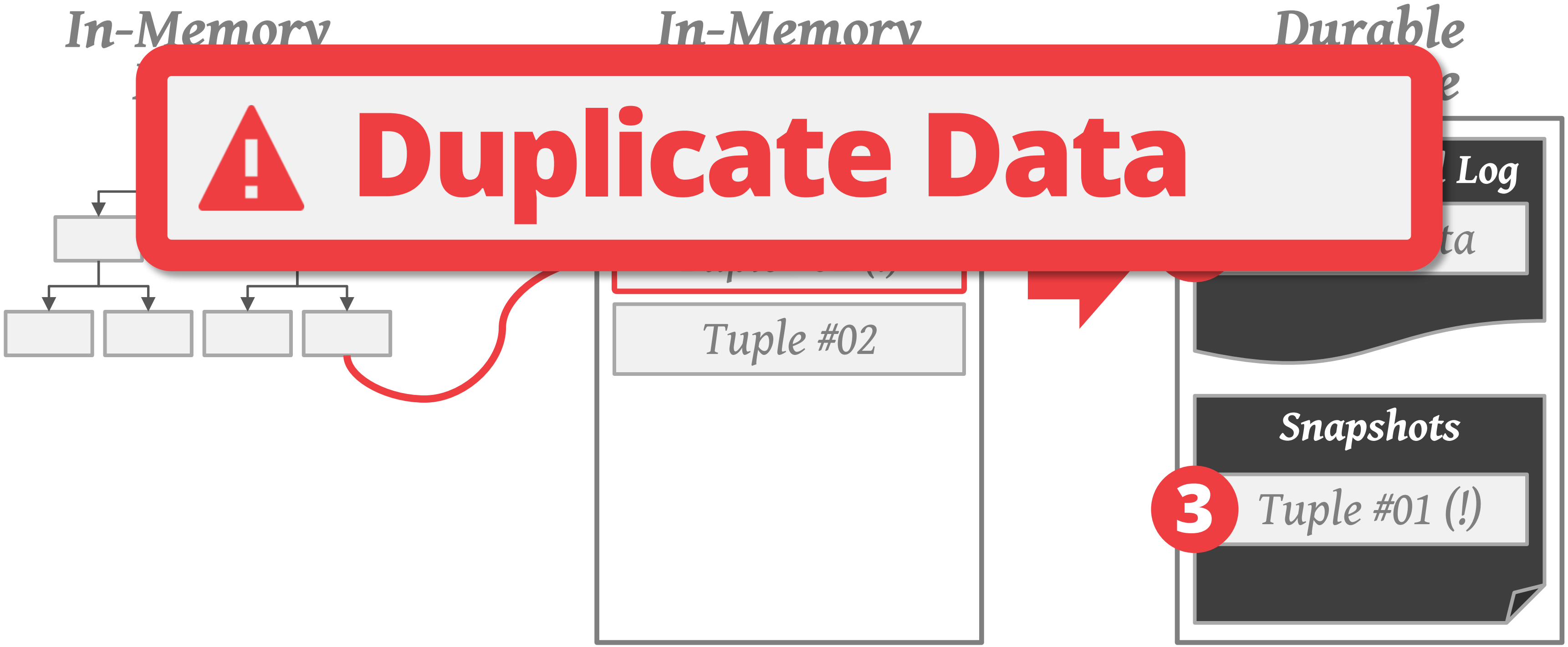
IN-PLACE UPDATES ENGINE



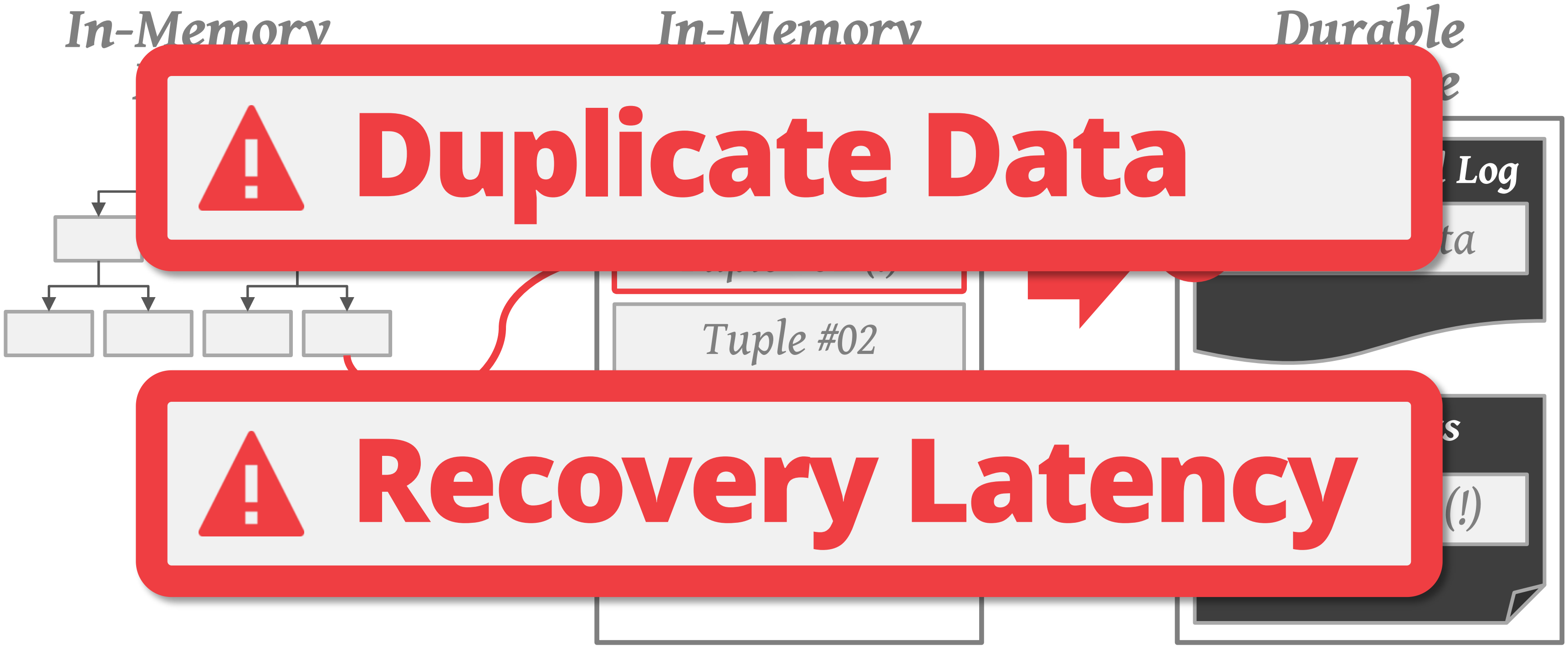
IN-PLACE UPDATES ENGINE



IN-PLACE UPDATES ENGINE



IN-PLACE UPDATES ENGINE



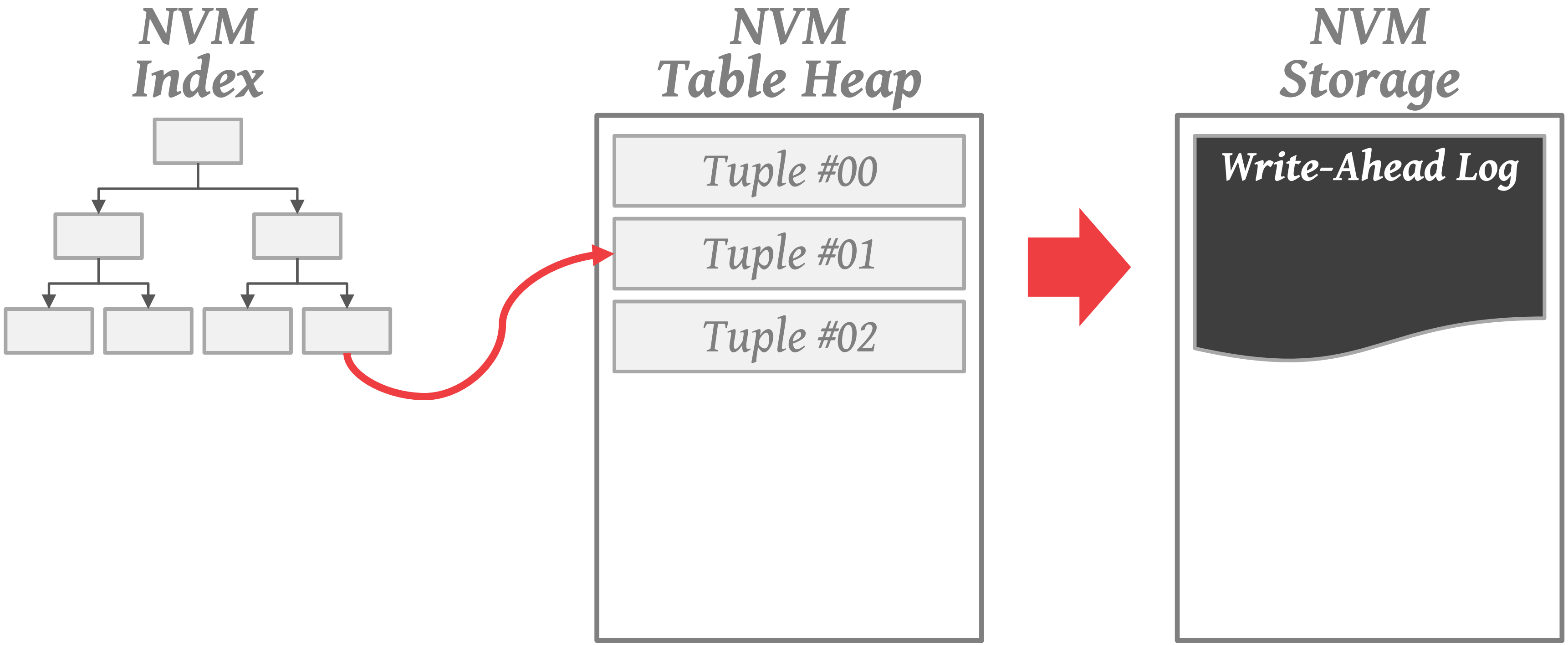
NVM-OPTIMIZED ARCHITECTURES

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

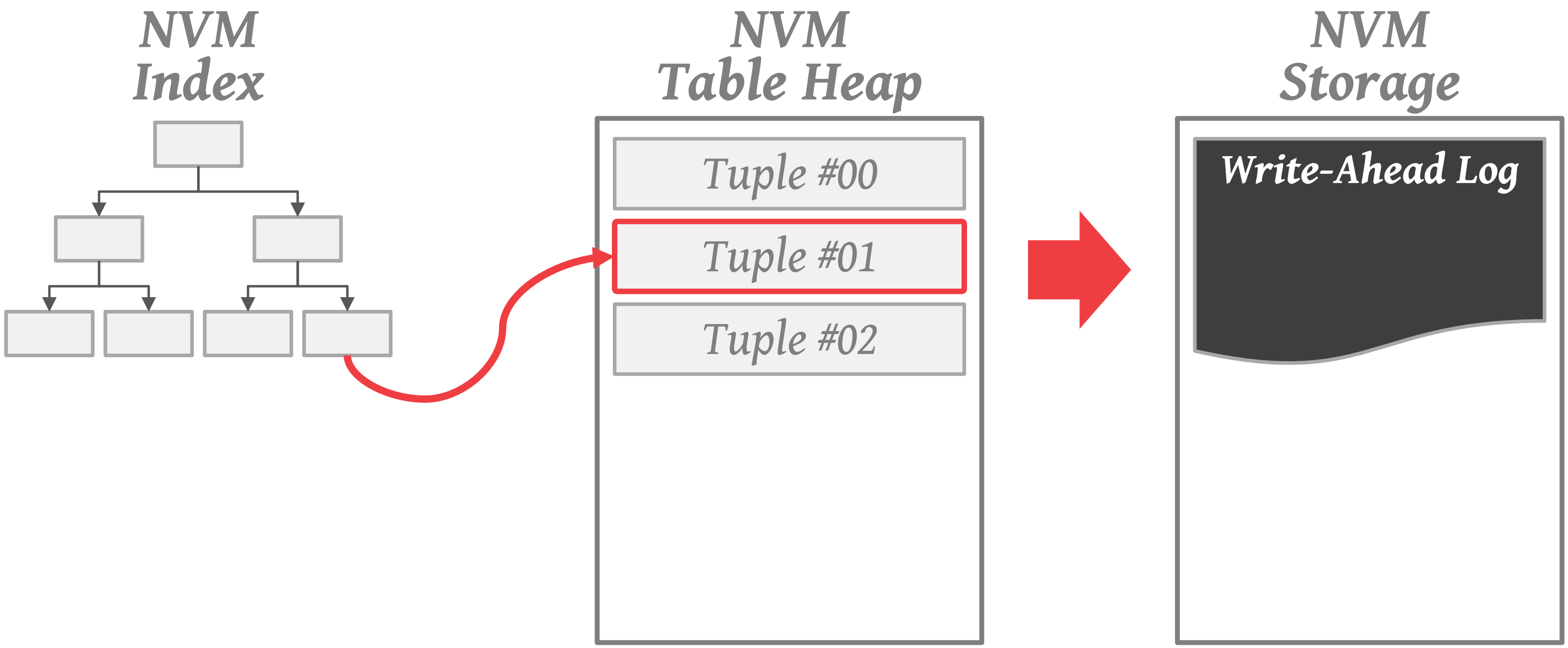
The DBMS only has to maintain a transient UNDO log for a txn until it commits.

→ 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 NVM at the time of commit.

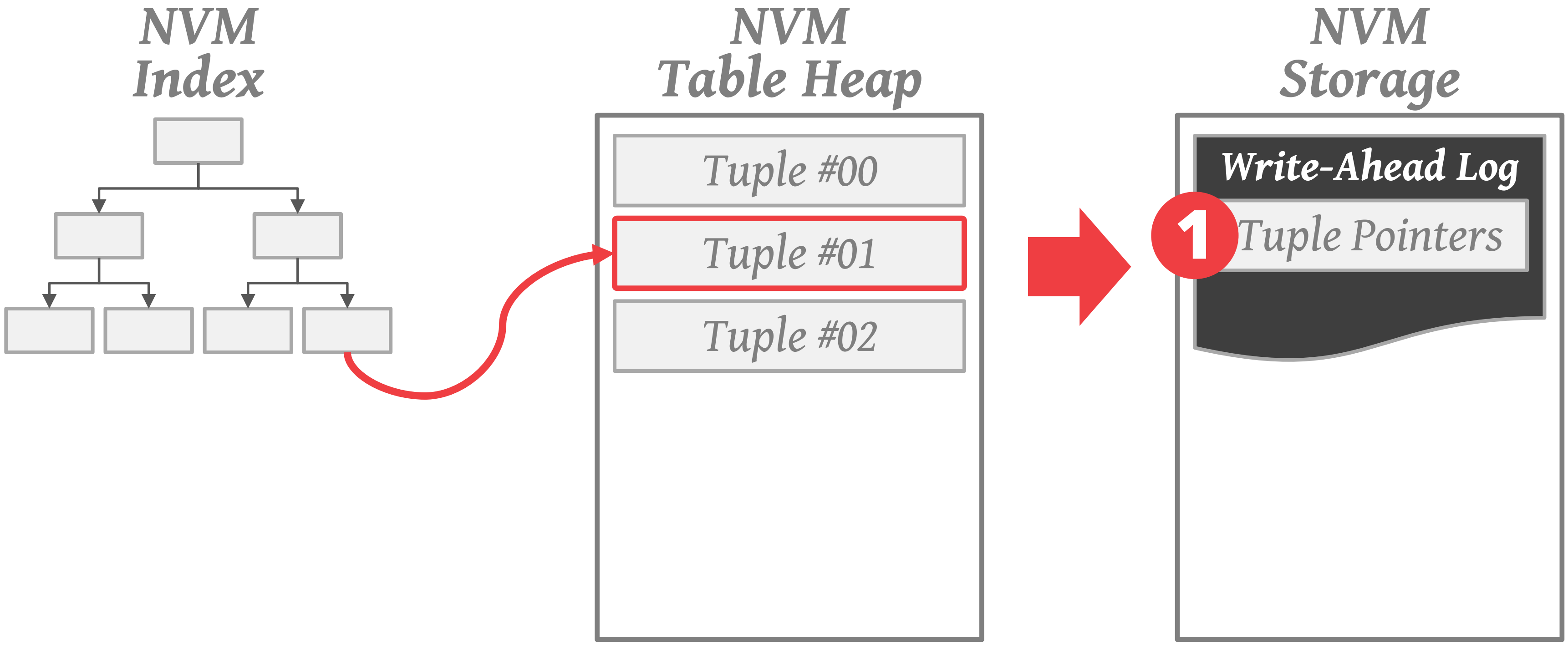
NVM IN-PLACE UPDATES ENGINE



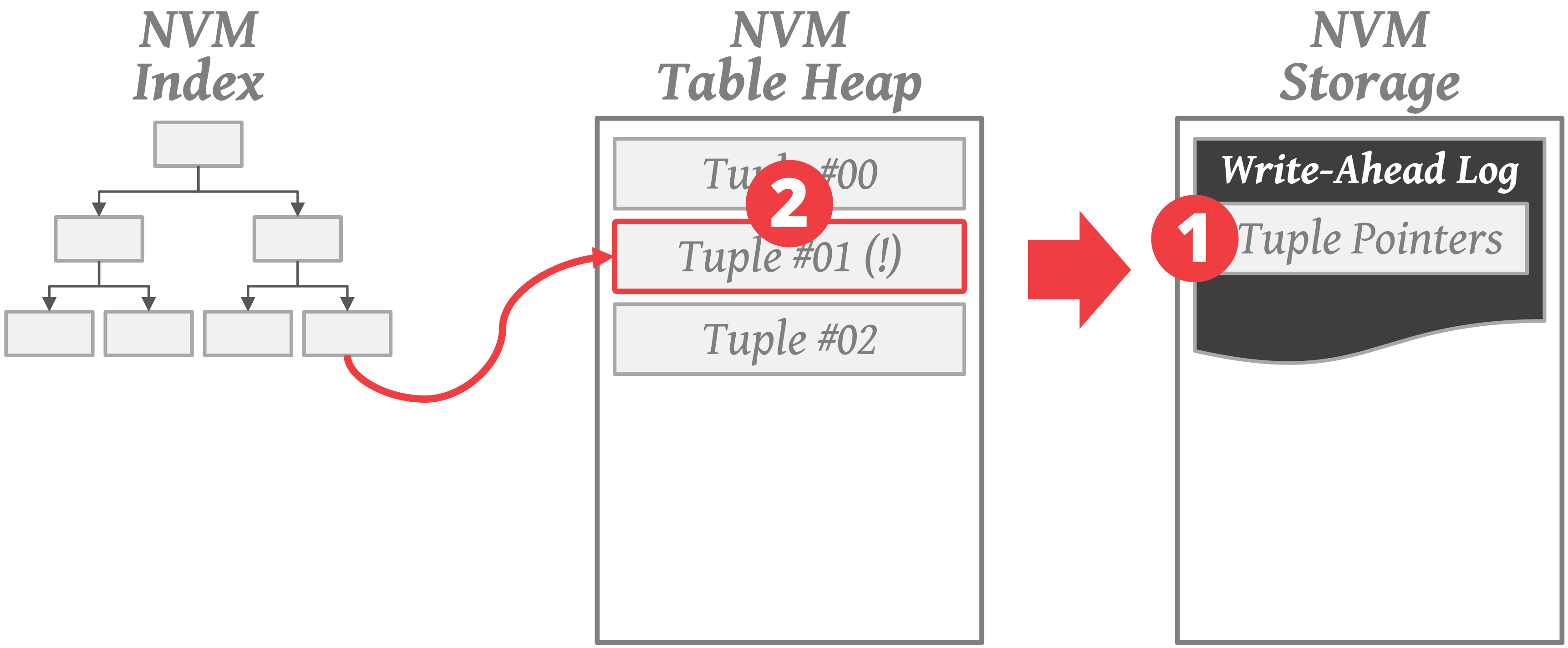
NVM IN-PLACE UPDATES ENGINE



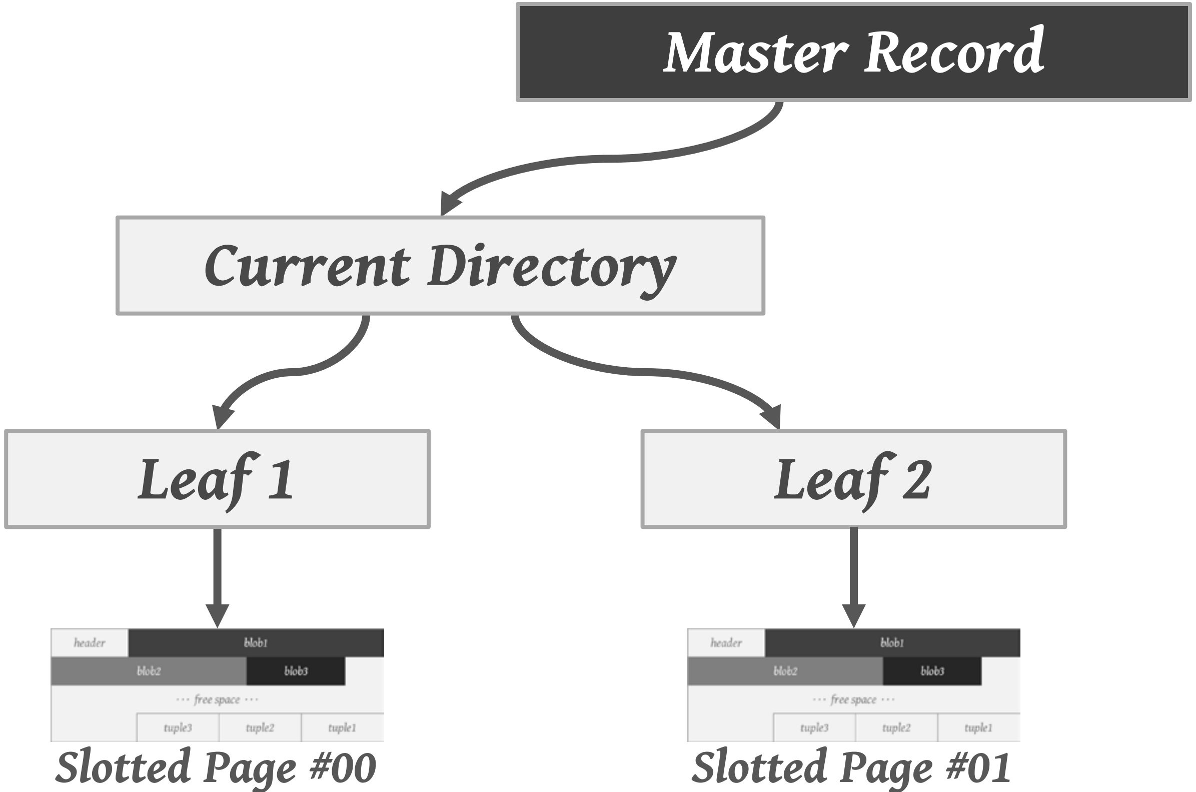
NVM IN-PLACE UPDATES ENGINE



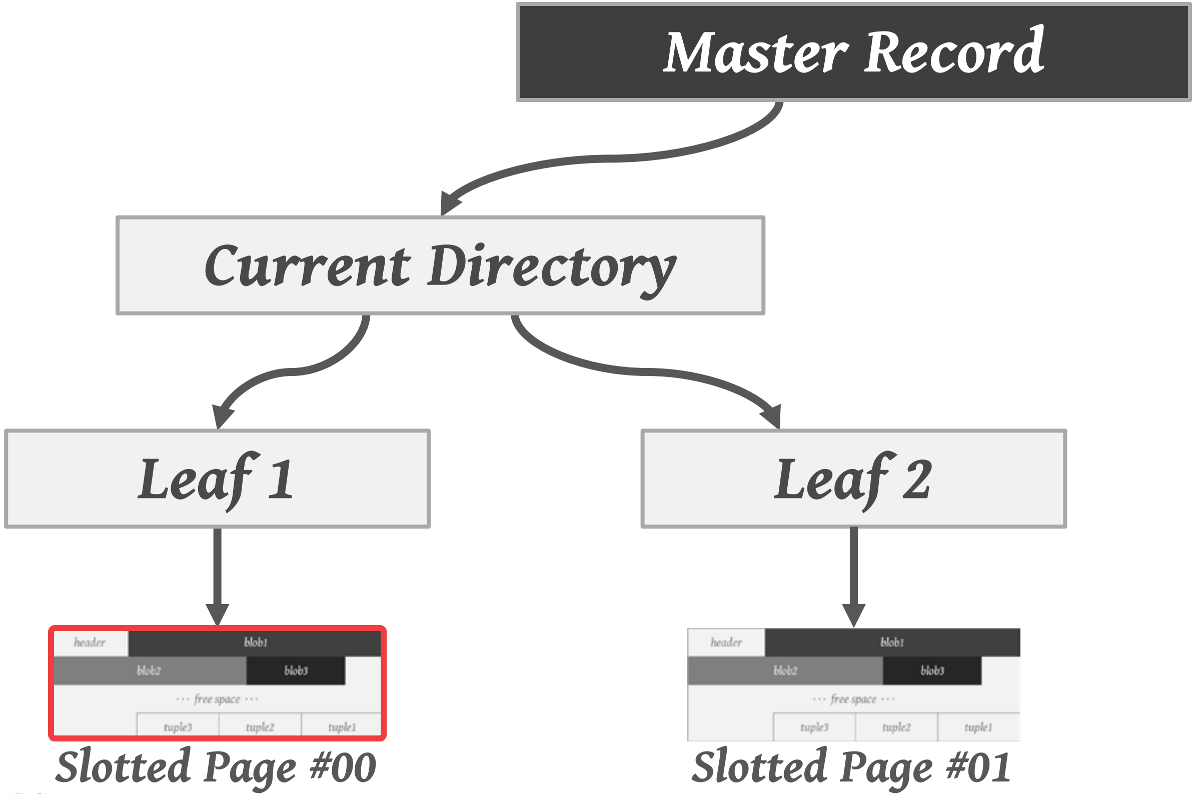
NVM IN-PLACE UPDATES ENGINE



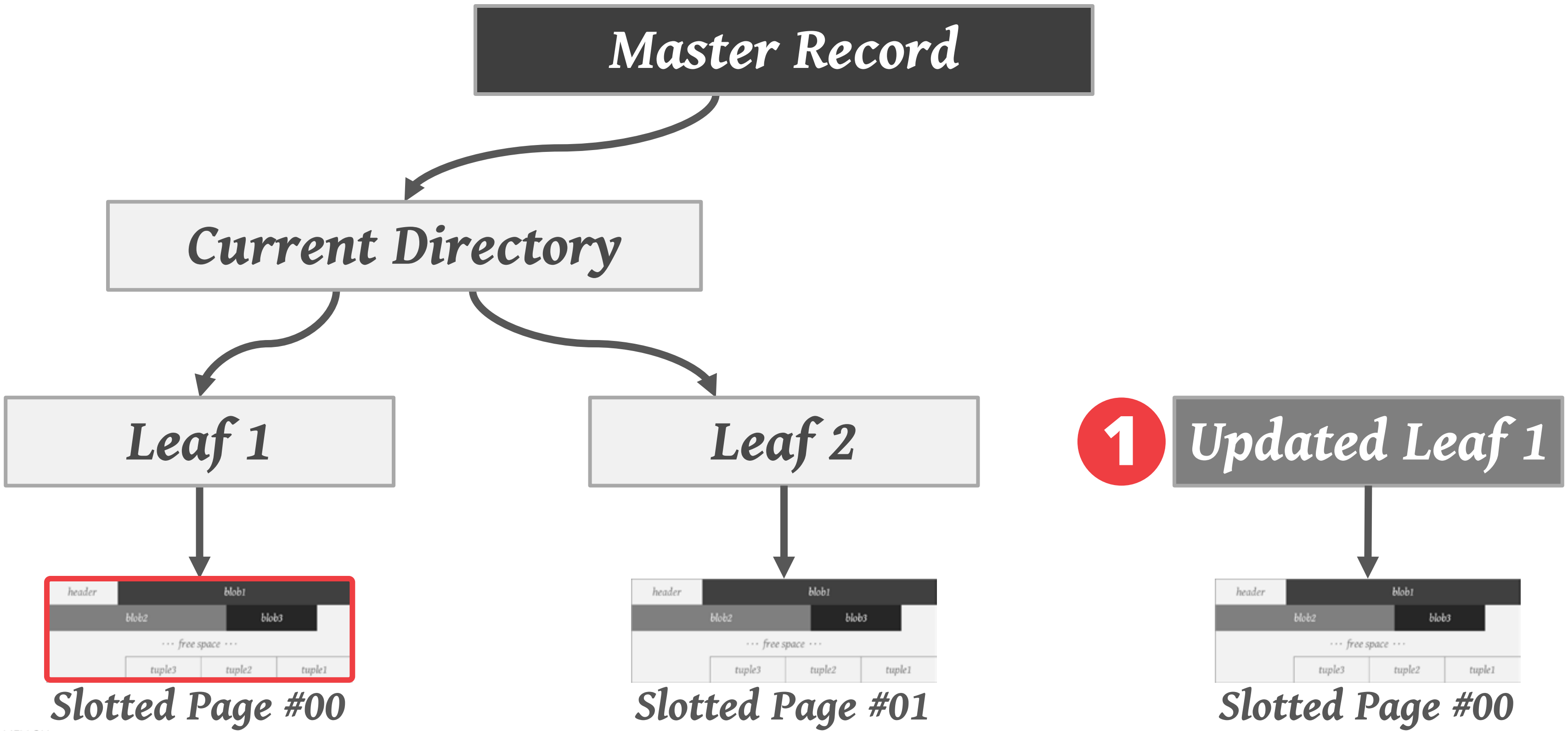
COPY-ON-WRITE ENGINE



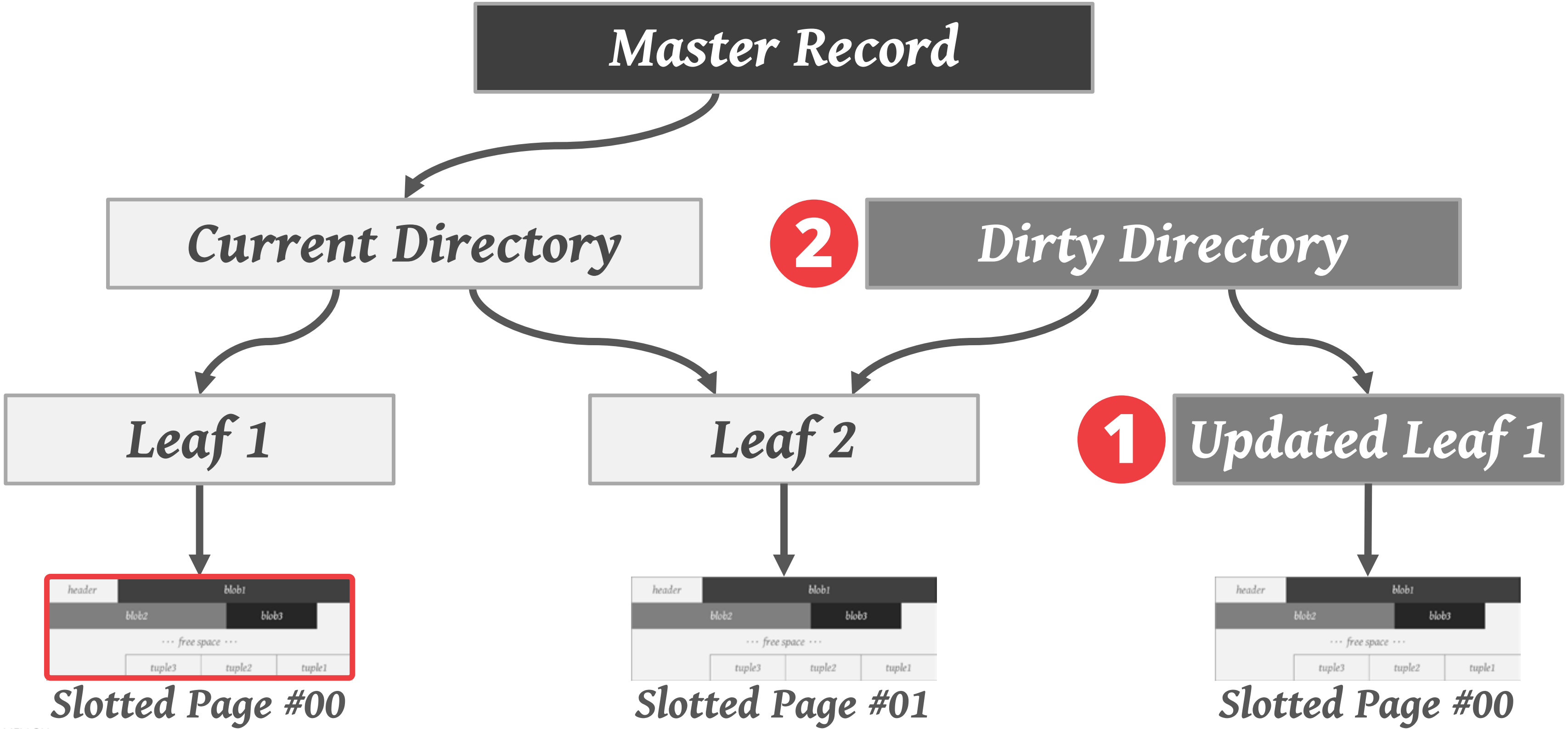
COPY-ON-WRITE ENGINE



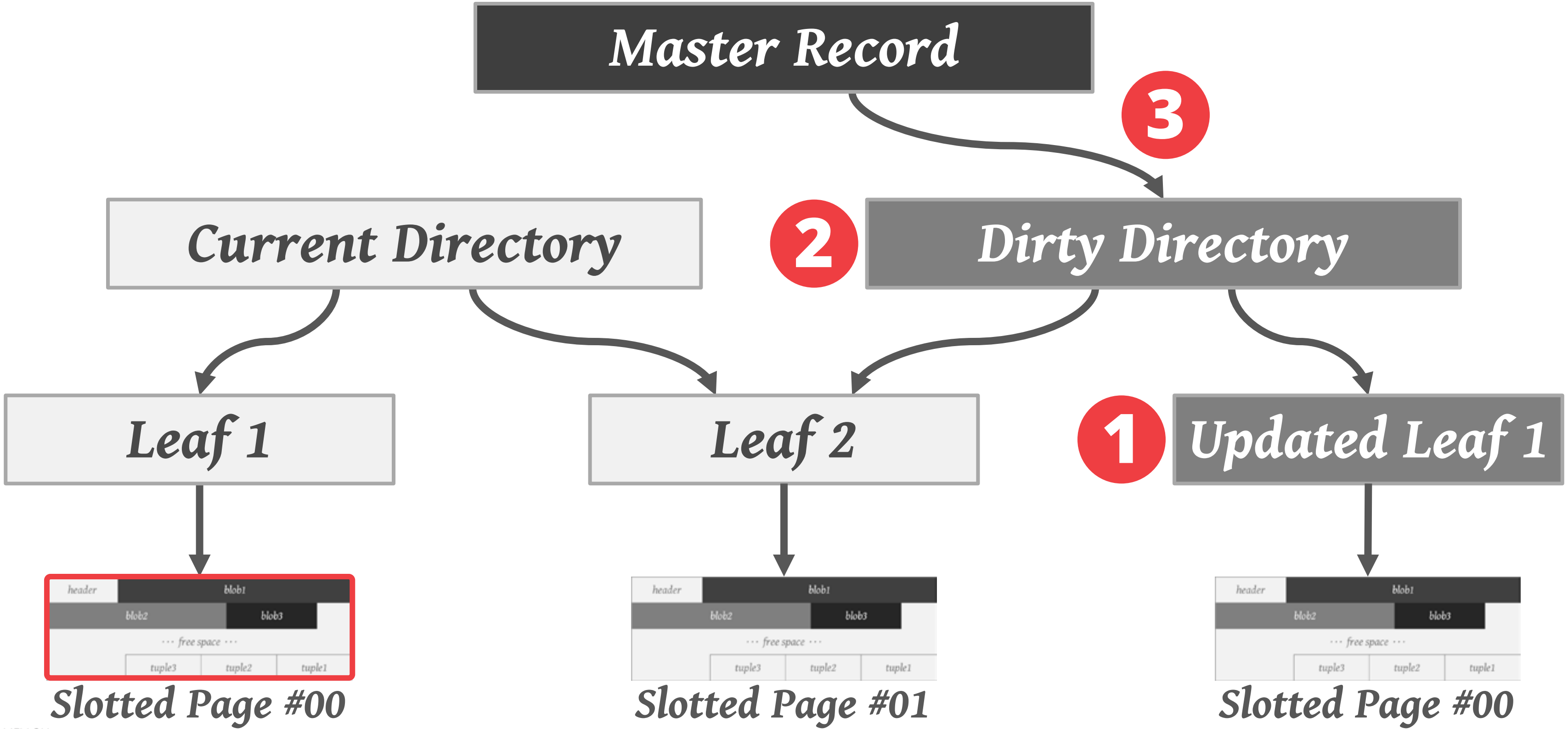
COPY-ON-WRITE ENGINE



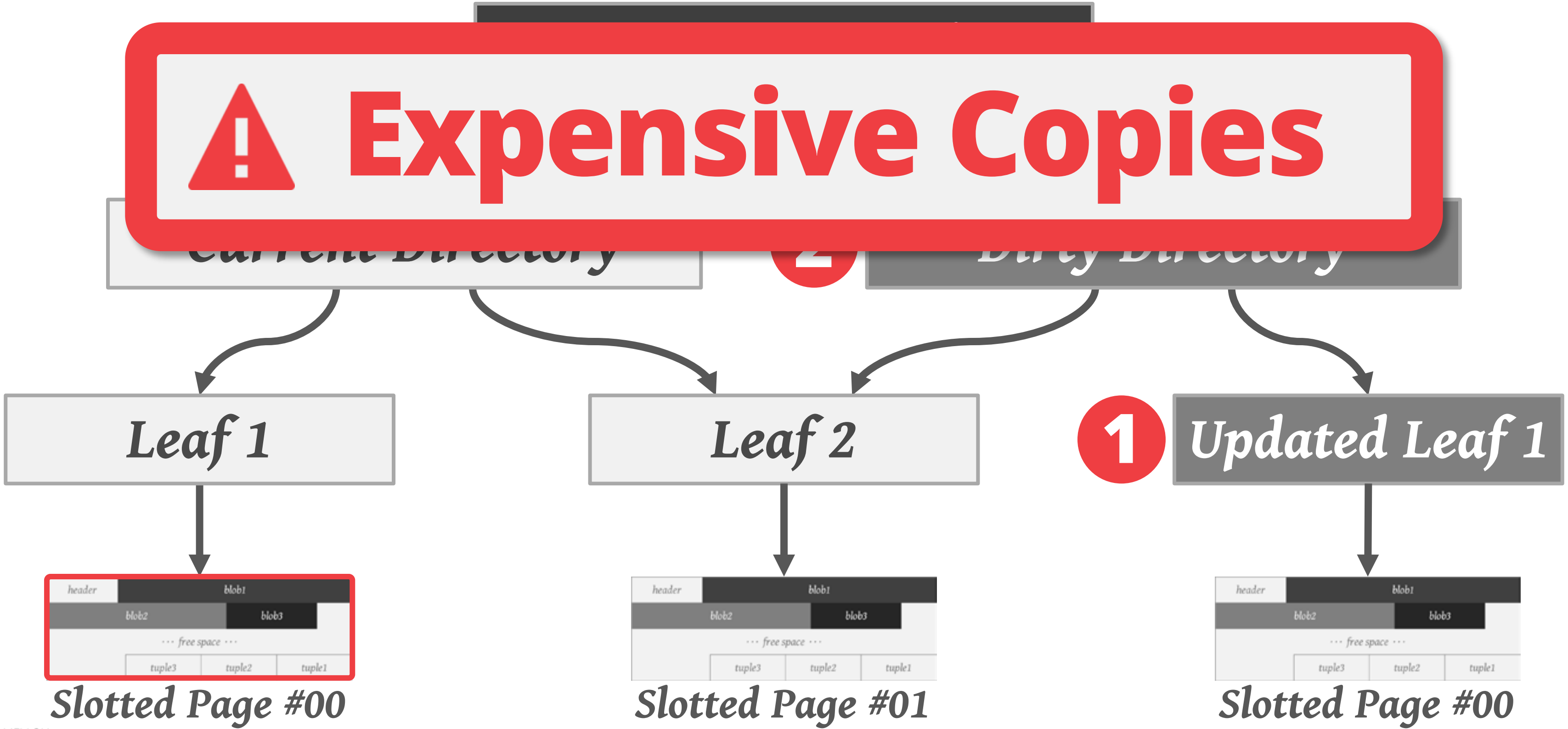
COPY-ON-WRITE ENGINE



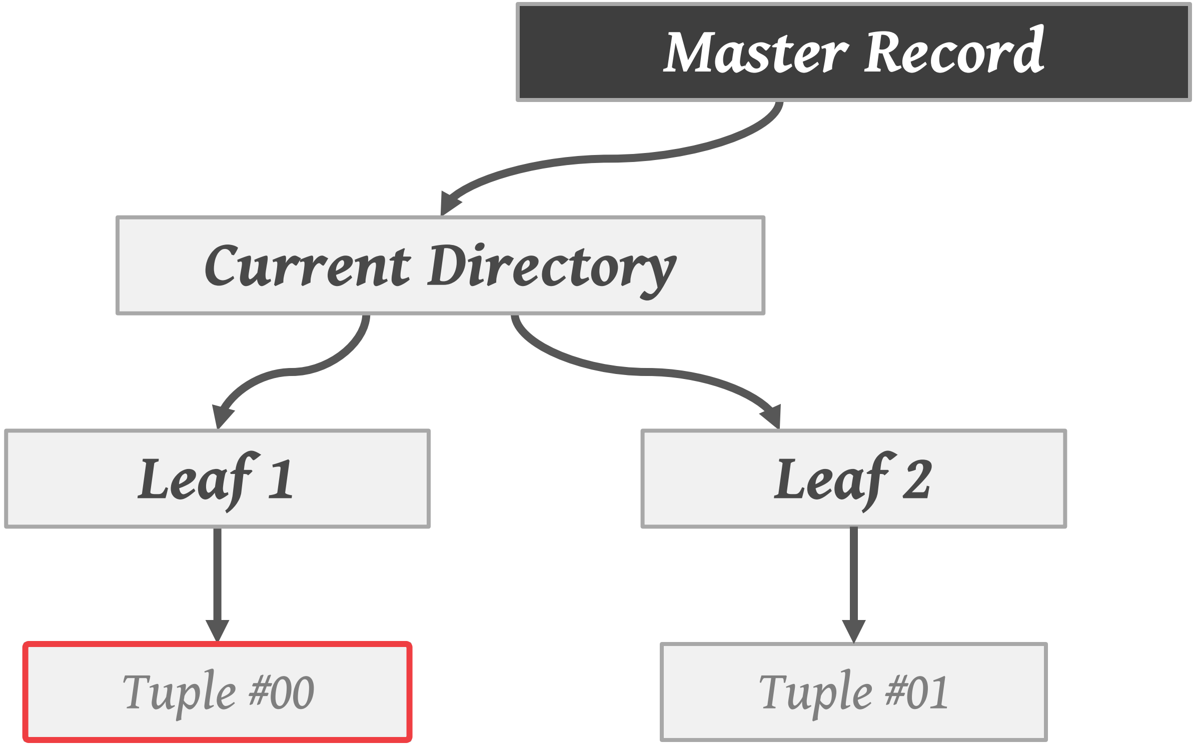
COPY-ON-WRITE ENGINE



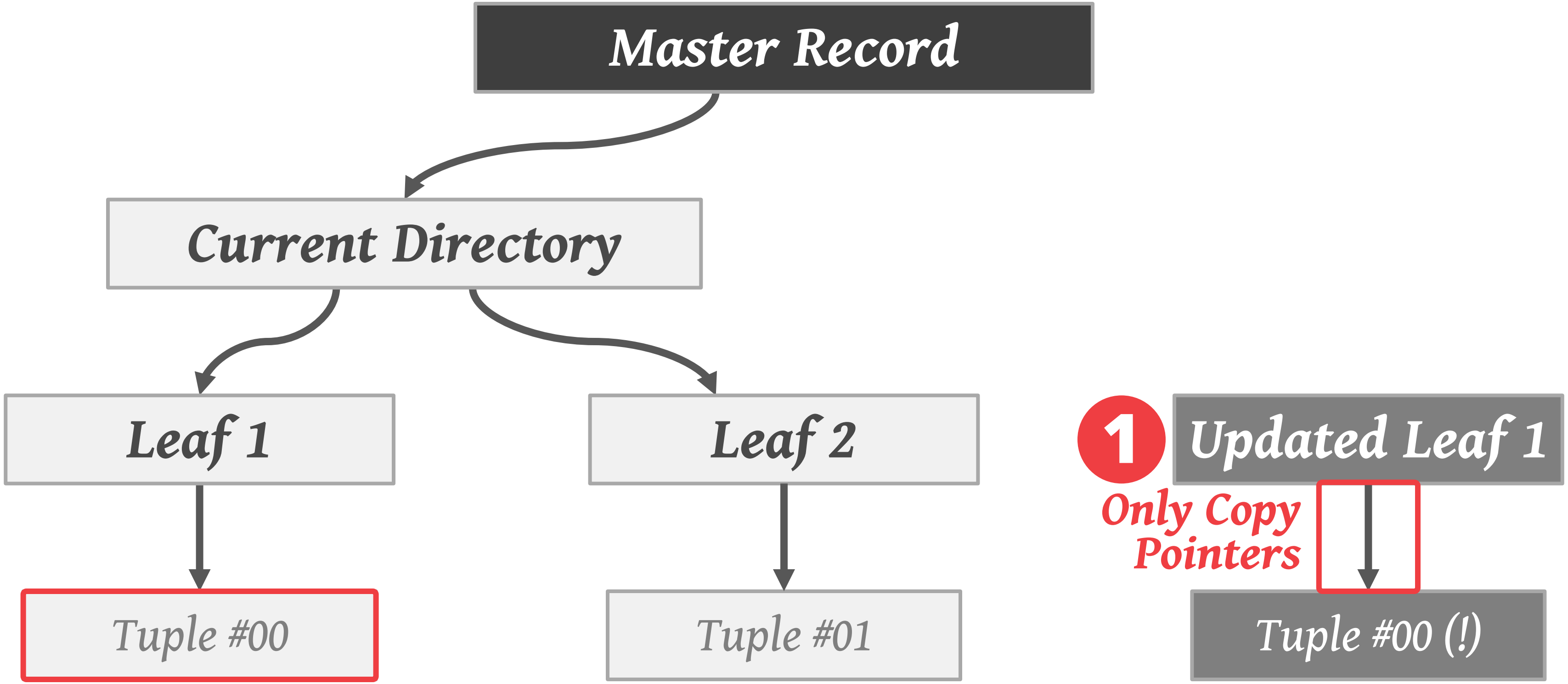
COPY-ON-WRITE ENGINE



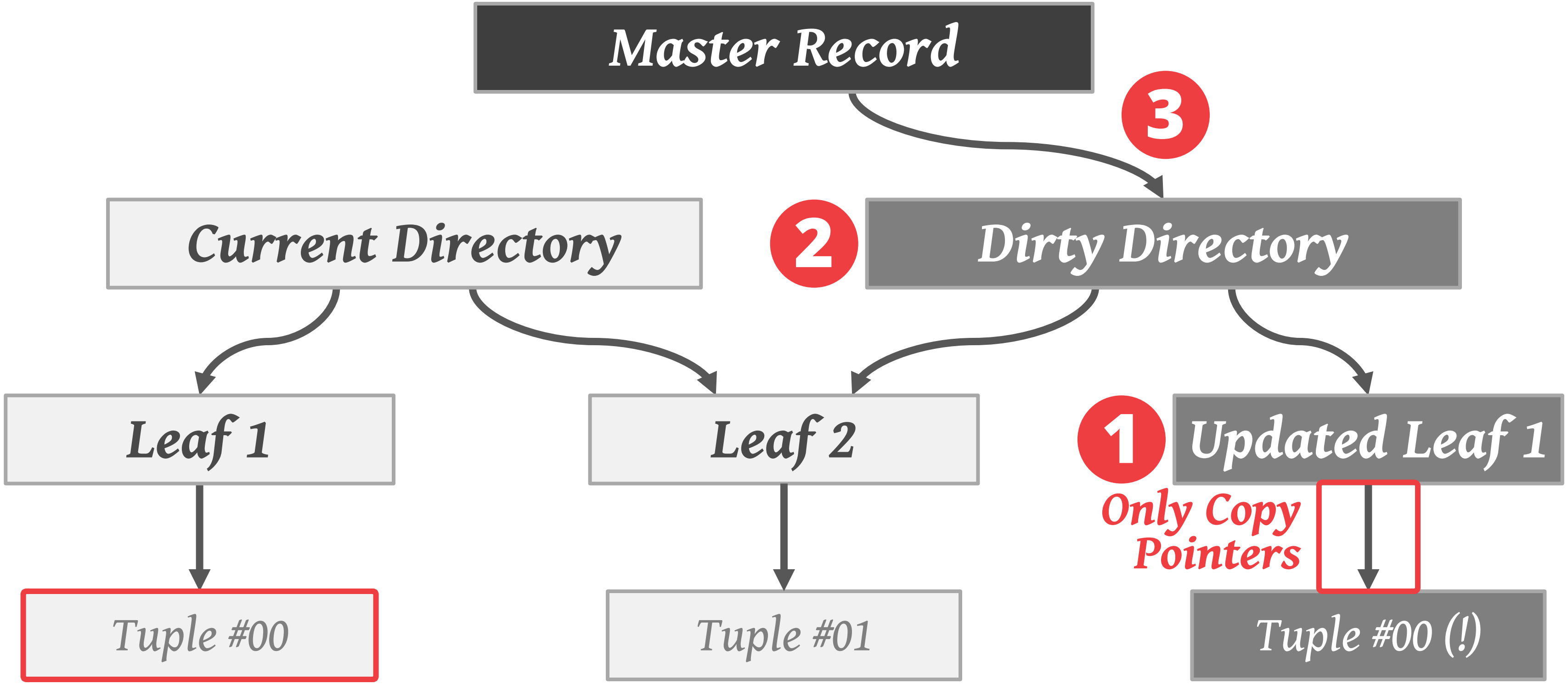
NVM COPY-ON-WRITE ENGINE



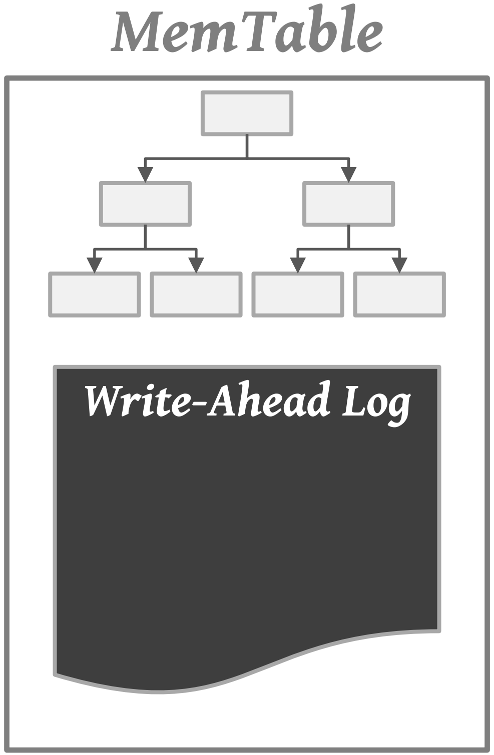
NVM COPY-ON-WRITE ENGINE

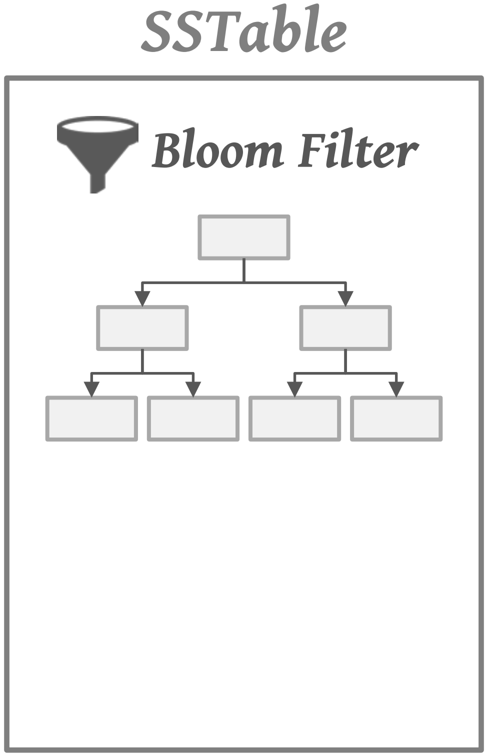


NVM COPY-ON-WRITE ENGINE

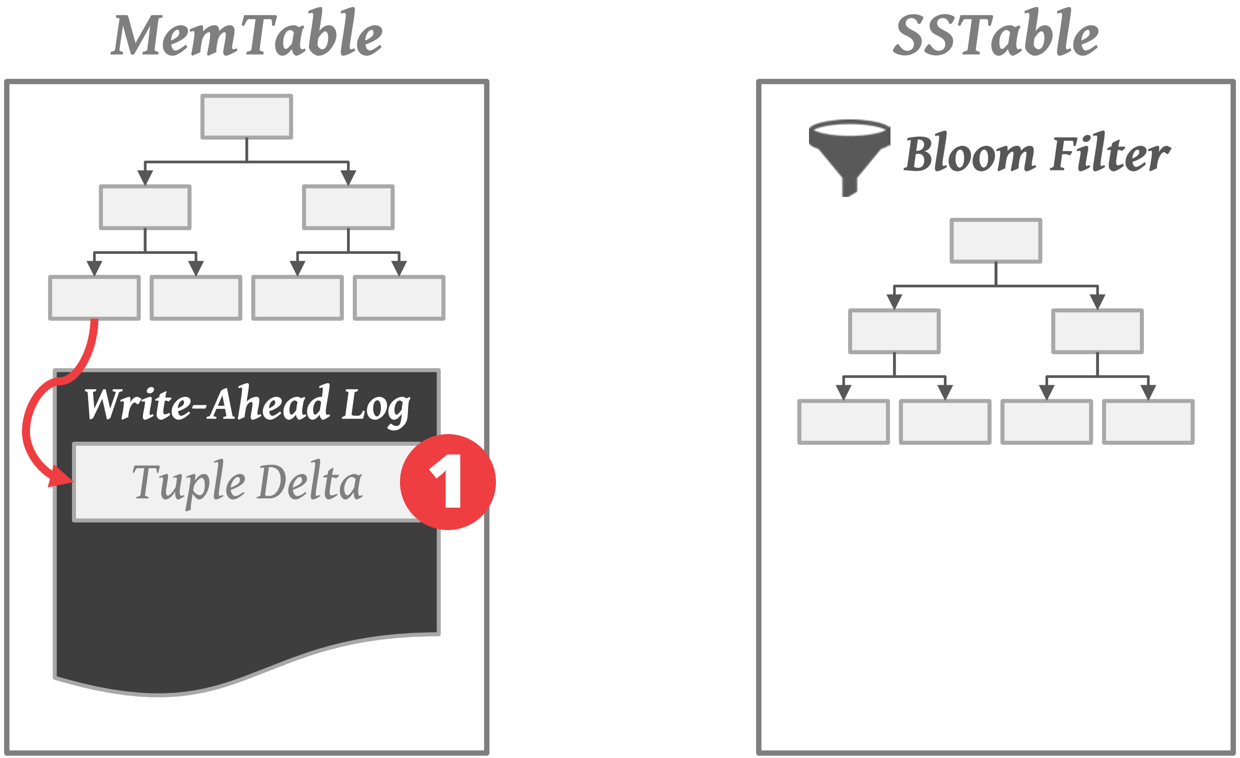


LOG-STRUCTURED ENGINE

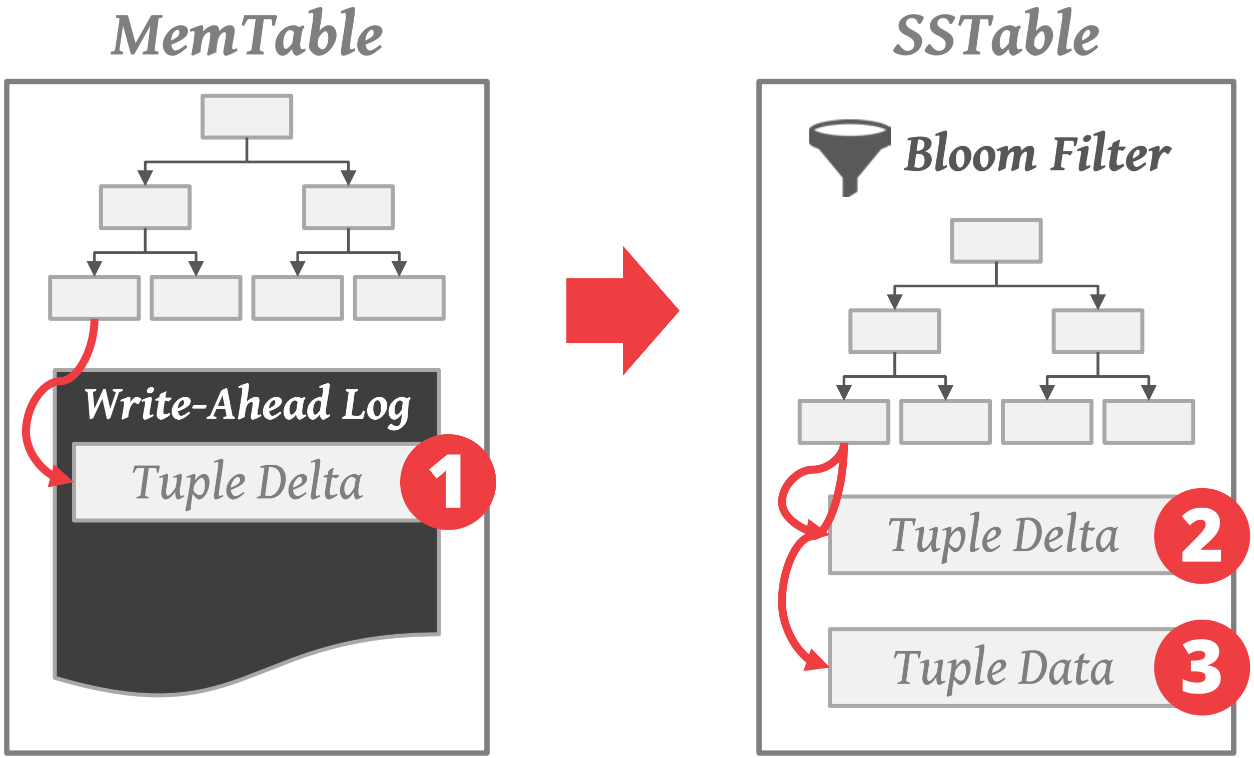




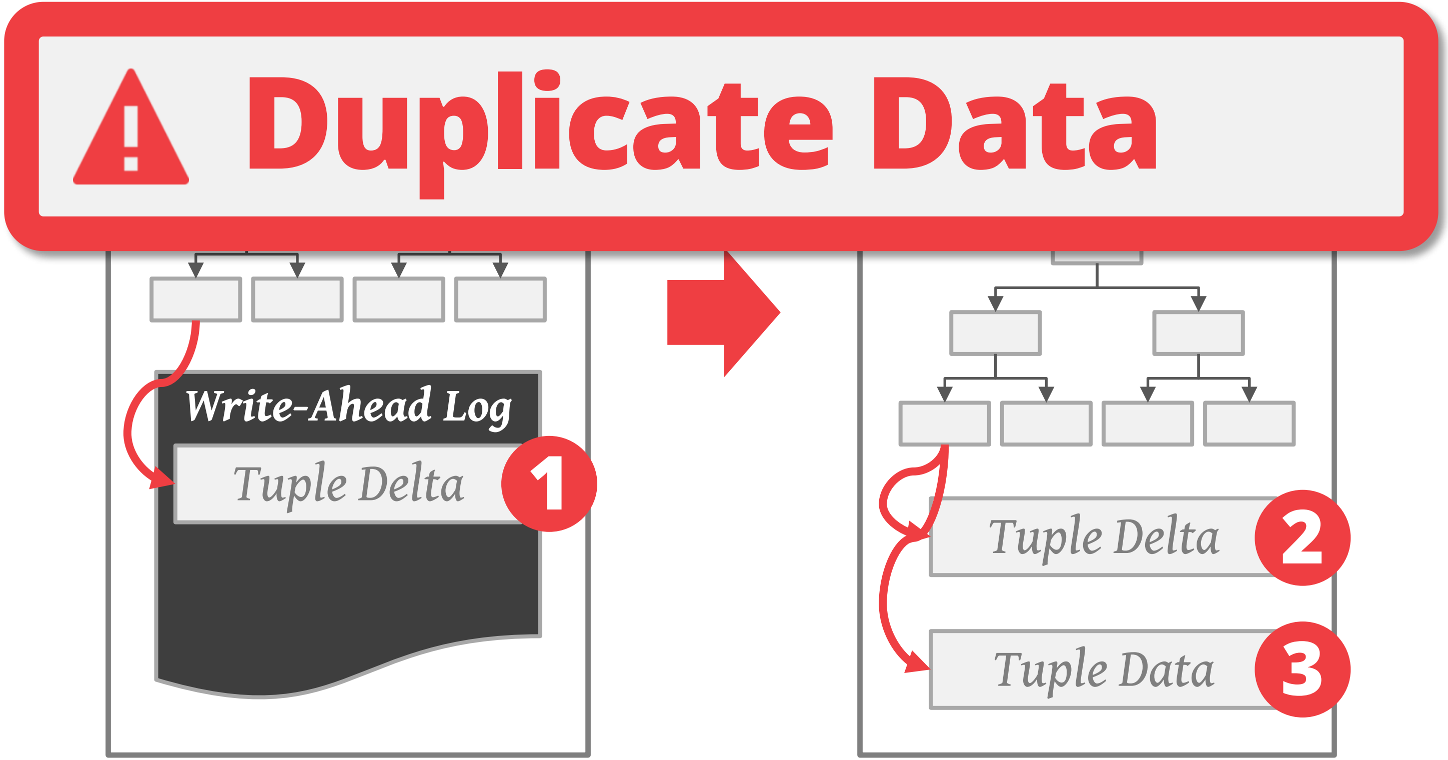
LOG-STRUCTURED ENGINE



LOG-STRUCTURED ENGINE



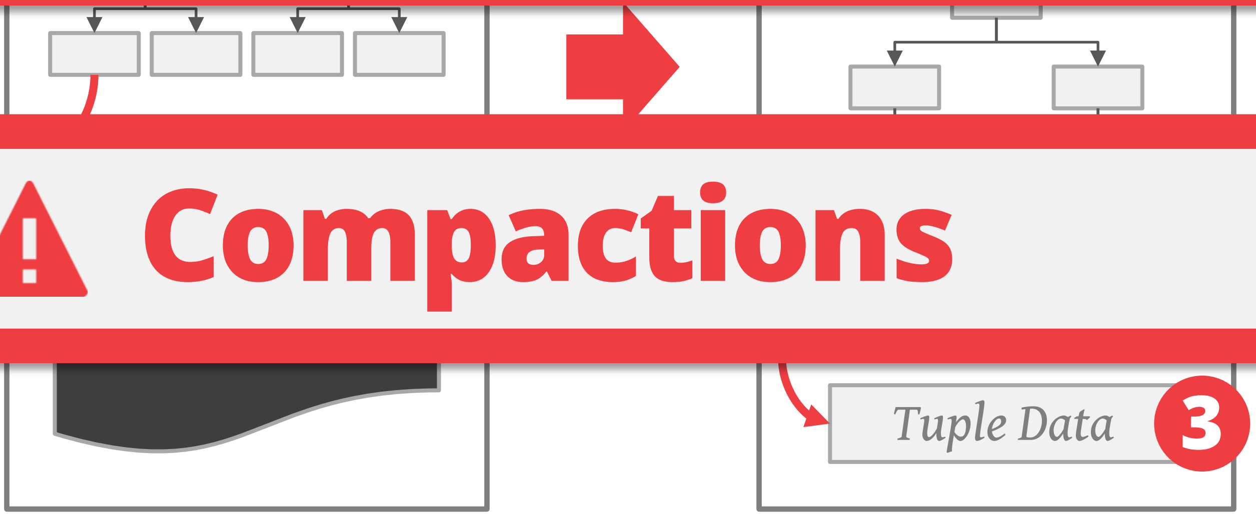
LOG-STRUCTURED ENGINE



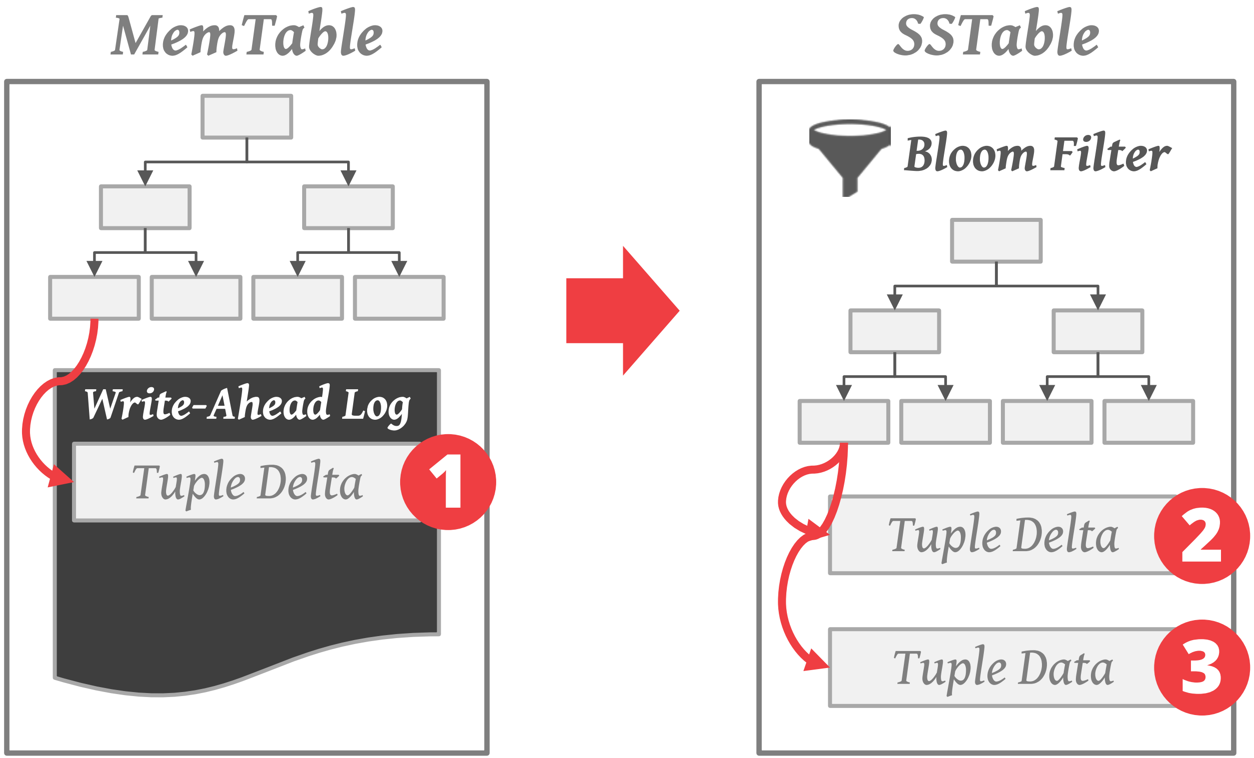
LOG-STRUCTURED ENGINE

⚠ Duplicate Data

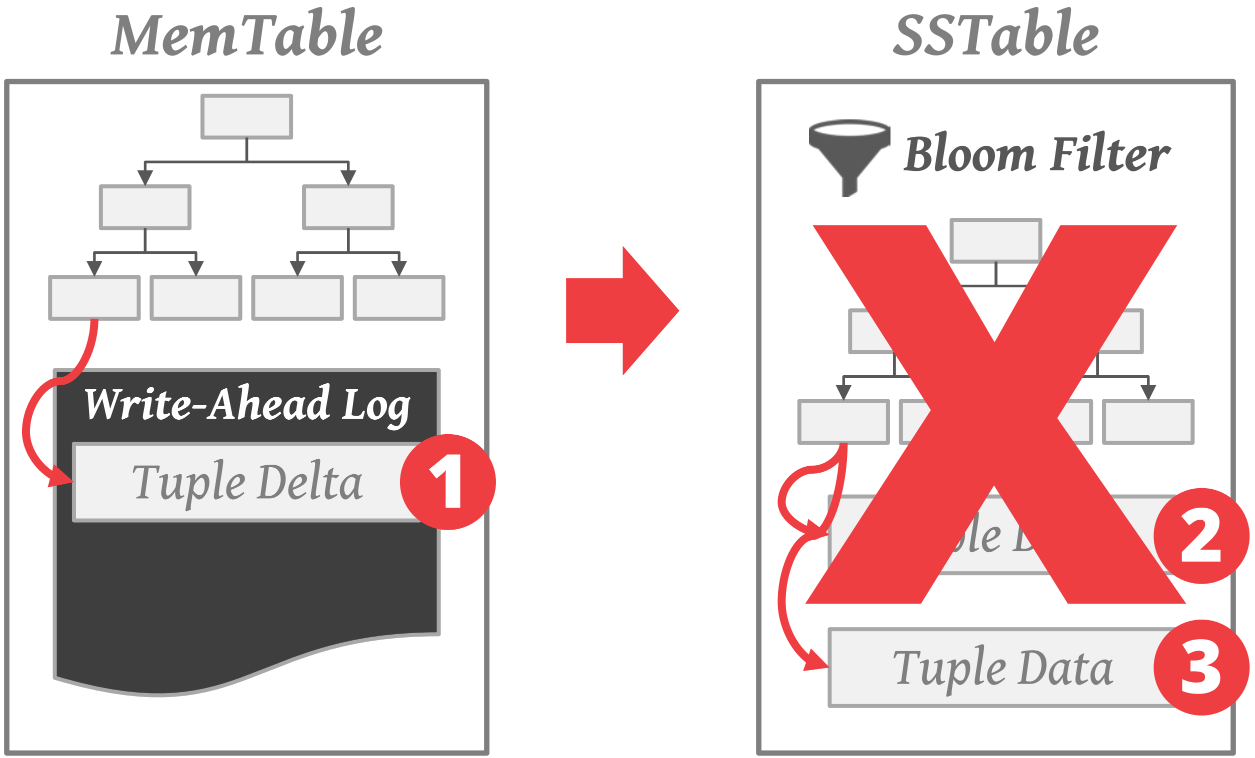
⚠ Compactions



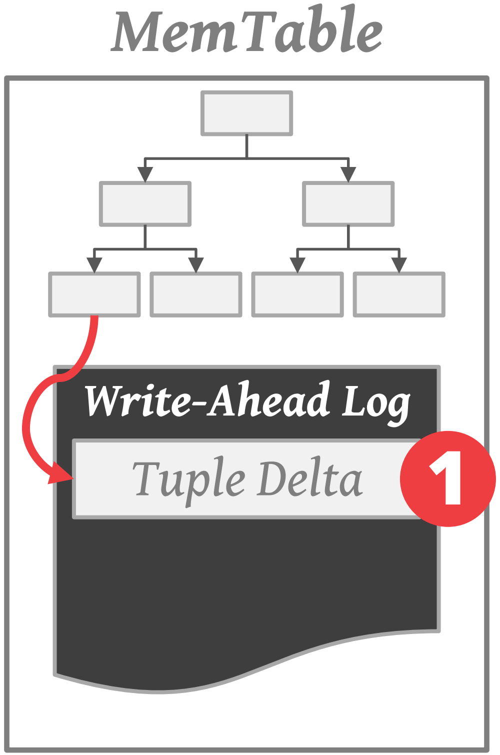
NVM LOG-STRUCTURED ENGINE



NVM LOG-STRUCTURED ENGINE



NVM LOG-STRUCTURED ENGINE



SUMMARY

Storage Optimizations

→ Leverage byte-addressability to avoid unnecessary data duplication.

Recovery Optimizations

→ NVM-optimized recovery protocols avoid the overhead of processing a log.
→ Non-volatile data structures ensure consistency.

EVALUATION

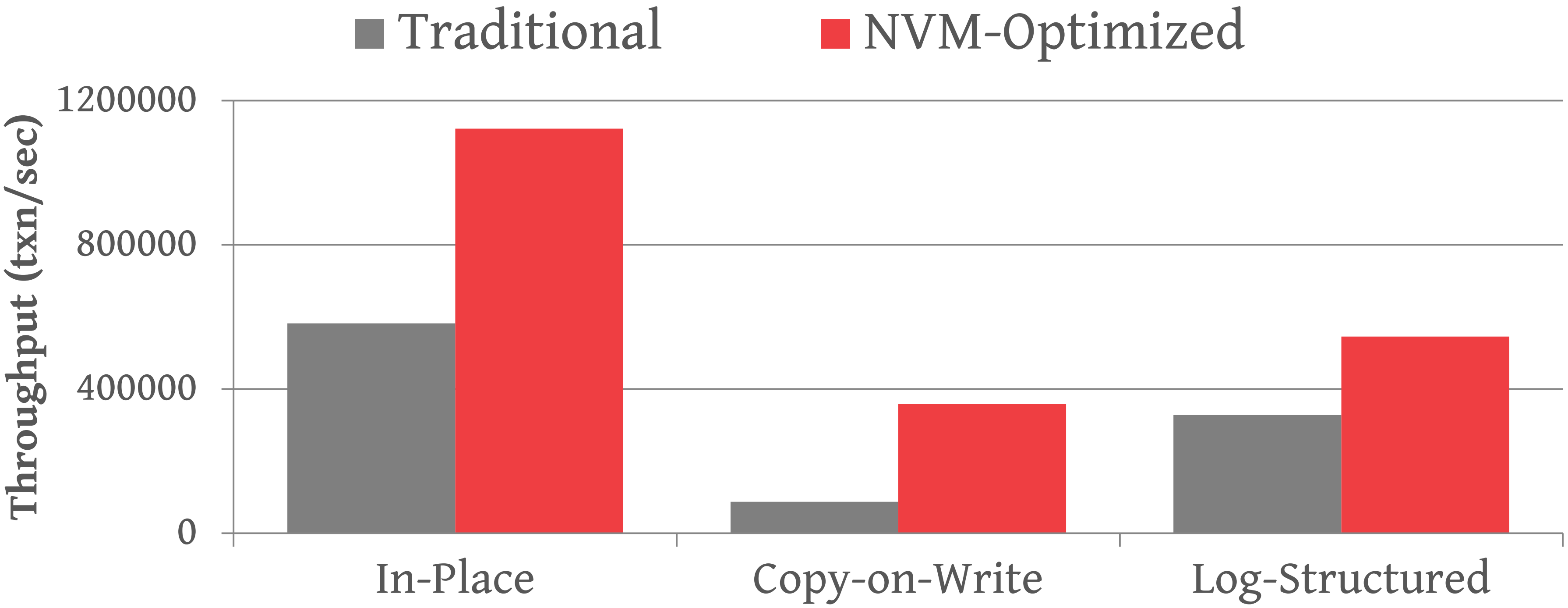
N-Store DBMS testbed with pluggable storage manager architecture.
→ H-Store-style concurrency control

Intel Labs NVM Hardware Emulator
→ NVM latency = 2x DRAM latency

Yahoo! Cloud Serving Benchmark
→ 2 million records + 1 million transactions
→ 10% Reads / 90% Writes
→ High-skew setting

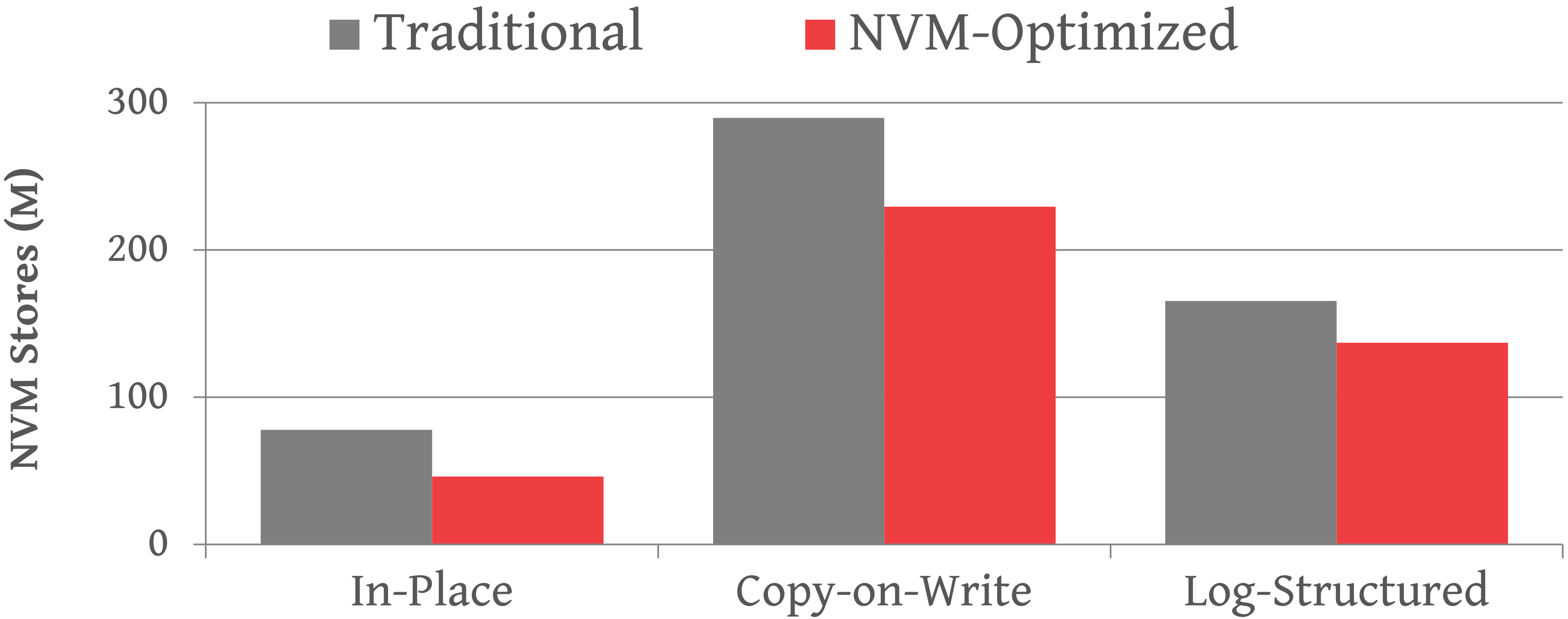
RUNTIME PERFORMANCE

YCSB Workload – 10% Reads / 90% Writes
NVRAM – 2x DRAM Latency



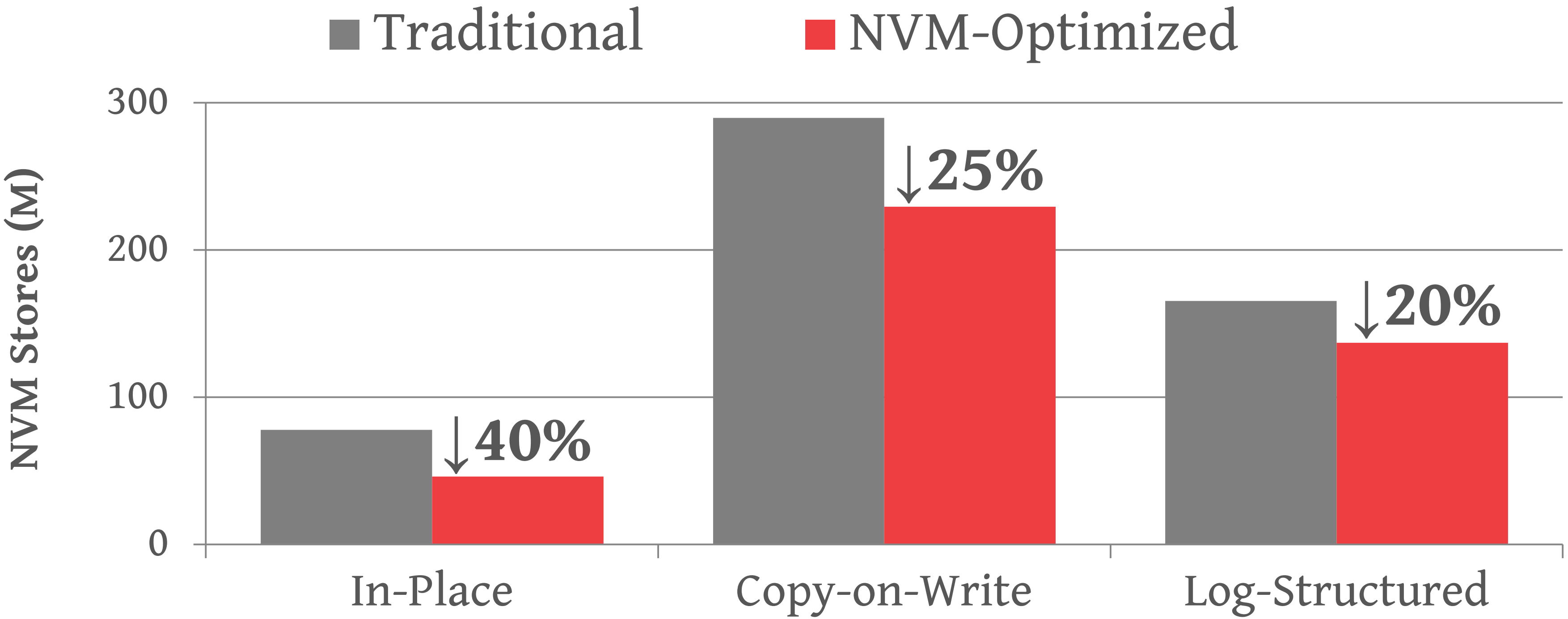
WRITE ENDURANCE

YCSB Workload – 10% Reads / 90% Writes
NVRAM – 2x DRAM Latency



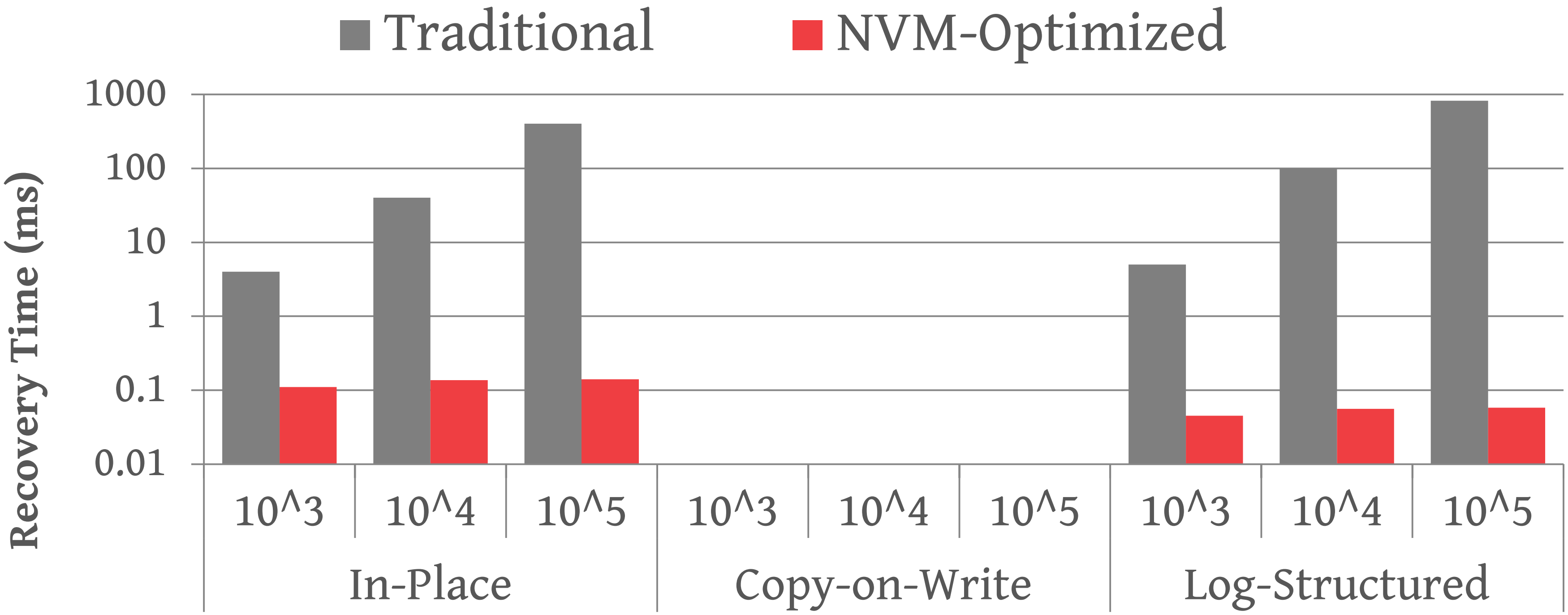
WRITE ENDURANCE

YCSB Workload – 10% Reads / 90% Writes
NVRAM – 2x DRAM Latency



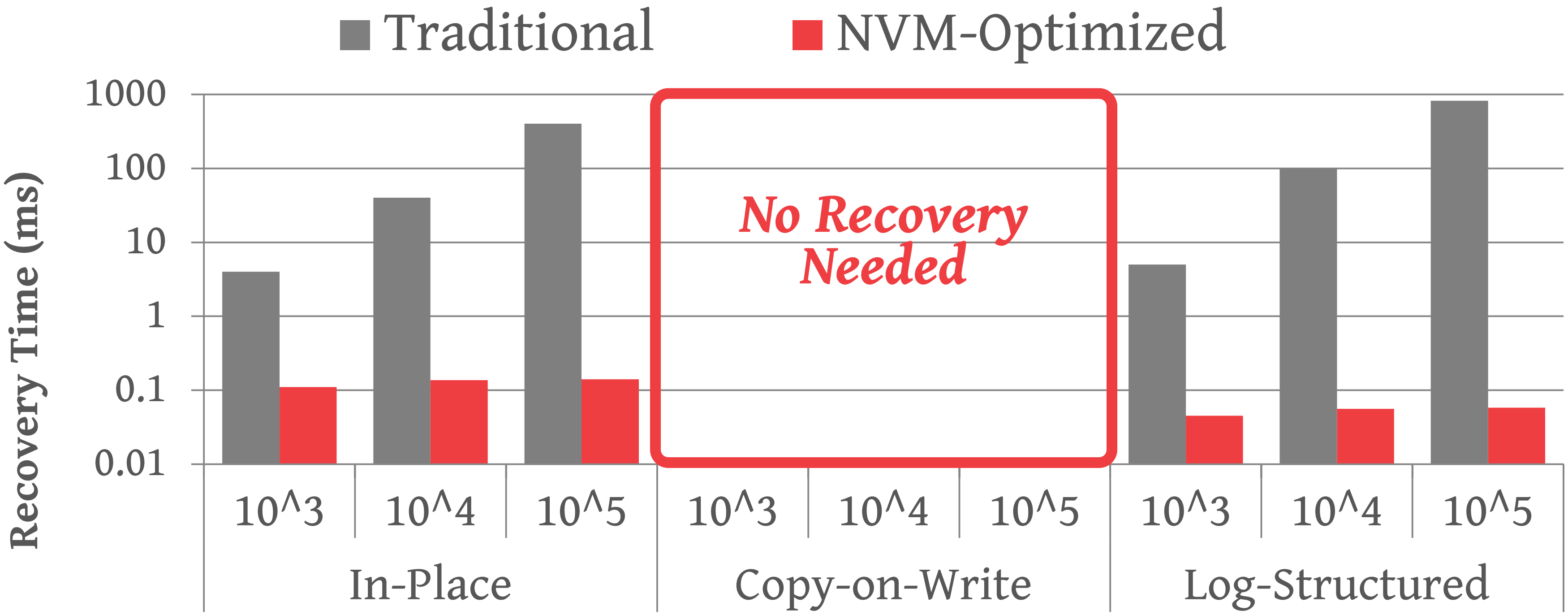
RECOVERY LATENCY

Elapsed time to replay log on recovery
NVRAM – 2x DRAM Latency



RECOVERY LATENCY

Elapsed time to replay log on recovery
NVRAM – 2x DRAM Latency



PARTING THOUGHTS

Designing for NVM is important
→ Non-volatile data structures provide higher throughput and faster recovery

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

CODE REVIEWS

Every group will perform a code review of another group.

→ Dev group will send a pull request on Github.
→ Review group will write comments on that request.
→ You will need to send me your pull request URL

We will provide a write-up later this week.

Due Date: May 8th @ 11:59pm

Please be helpful and courteous.

GENERAL TIPS

The dev team should provide you with a summary of what files/functions the reviewing team should look at.

Review fewer than 400 lines of code at a time and only for at most 60 minutes.

Use a **checklist** to outline what kind of problems you are looking for.

Source: SmartBear

CHECKLIST – GENERAL

Does the code work?

Is all the code easily understood?

Is there any redundant or duplicate code?

Is the code as modular as possible?

Can any global variables be replaced?

Is there any commented out code?

Is it using proper debug log functions?

Source: Gareth Wilson

CHECKLIST – DOCUMENTATION

Do comments describe the intent of the code?

Are all functions commented?

Is any unusual behavior described?

Is the use of 3rd-party libraries documented?

Is there any incomplete code?

Source: Gareth Wilson

CHECKLIST – TESTING

Do tests exist and are they comprehensive?

Are the tests actually testing the feature?

Are they relying on hardcoded answers?

What is the code coverage?

Source: Gareth Wilson

LCOV - code coverage report

			Hit	Total	Coverage
Current view:	top level				
Test:	Peloton-0.0.2 Code Coverage	**Lines:**	14275	32590	43.8 %
Date:	2016-04-15	**Functions:**	2789	6929	40.3 %
Legend:	Rating: low: < 75 % medium: >= 75 % high: >= 90 %				

Directory	Line Coverage			Functions	
/var/lib/jenkins/jobs/Peloton/workspace/src		0.0 %	0 / 347	0.0 %	0 / 33
/var/lib/jenkins/jobs/Peloton/workspace/src/backend/brain		100.0 %	121 / 121	88.9 %	24 / 27
/var/lib/jenkins/jobs/Peloton/workspace/src/backend/bridge/ddl		1.7 %	14 / 845	21.9 %	28 / 128
/var/lib/jenkins/jobs/Peloton/workspace/src/backend/bridge/ddl/tests		2.3 %	6 / 257	34.3 %	12 / 35
/var/lib/jenkins/jobs/Peloton/workspace/src/backend/bridge/dml/executor		1.0 %	1 / 102	28.6 %	2 / 7
/var/lib/jenkins/jobs/Peloton/workspace/src/backend/bridge/dml/expr		2.0 %	6 / 304	16.7 %	4 / 24
/var/lib/jenkins/jobs/Peloton/workspace/src/backend/bridge/dml/mapper		1.7 %	21 / 1249	31.7 %	33 / 104
/var/lib/jenkins/jobs/Peloton/workspace/src/backend/bridge/dml/tuple		0.4 %	1 / 223	33.3 %	2 / 6
/var/lib/jenkins/jobs/Peloton/workspace/src/backend/catalog		73.2 %	232 / 317	74.0 %	71 / 96
/var/lib/jenkins/jobs/Peloton/workspace/src/backend/common		43.8 %	1524 / 3482	57.4 %	316 / 551
/var/lib/jenkins/jobs/Peloton/workspace/src/backend/concurrency		84.5 %	1939 / 2295	88.4 %	205 / 232
/var/lib/jenkins/jobs/Peloton/workspace/src/backend/executor		88.6 %	1711 / 1931	85.9 %	274 / 319
/var/lib/jenkins/jobs/Peloton/workspace/src/backend/expression		11.9 %	321 / 2689	10.3 %	129 / 1253
/var/lib/jenkins/jobs/Peloton/workspace/src/backend/index		35.5 %	200 / 563	8.1 %	67 / 827
/var/lib/jenkins/jobs/Peloton/workspace/src/backend/logging		33.3 %	91 / 273	44.7 %	34 / 76
/var/lib/jenkins/jobs/Peloton/workspace/src/backend/logging/checkpoint		19.9 %	43 / 216	47.1 %	8 / 17
/var/lib/jenkins/jobs/Peloton/workspace/src/backend/logging/loggers		10.2 %	81 / 791	27.8 %	20 / 72
/var/lib/jenkins/jobs/Peloton/workspace/src/backend/logging/records		33.8 %	48 / 142	50.0 %	18 / 36
/var/lib/jenkins/jobs/Peloton/workspace/src/backend/main		11.1 %	1 / 9	50.0 %	2 / 4
/var/lib/jenkins/jobs/Peloton/workspace/src/backend/networking		13.7 %	1182 / 8618	18.7 %	304 / 1626
/var/lib/jenkins/jobs/Peloton/workspace/src/backend/planner		46.0 %	190 / 413	52.3 %	104 / 199
/var/lib/jenkins/jobs/Peloton/workspace/src/backend/storage		55.2 %	853 / 1544	71.7 %	172 / 240
/var/lib/jenkins/jobs/Peloton/workspace/tests		81.8 %	27 / 33	90.5 %	19 / 21
/var/lib/jenkins/jobs/Peloton/workspace/tests/brain		100.0 %	40 / 40	88.9 %	8 / 9
/var/lib/jenkins/jobs/Peloton/workspace/tests/catalog		99.2 %	240 / 242	93.5 %	58 / 62
/var/lib/jenkins/jobs/Peloton/workspace/tests/common		95.8 %	1176 / 1227	96.9 %	186 / 192
/var/lib/jenkins/jobs/Peloton/workspace/tests/concurrency		91.3 %	701 / 768	91.5 %	108 / 118
/var/lib/jenkins/jobs/Peloton/workspace/tests/executor		98.7 %	2049 / 2077	94.8 %	289 / 305
/var/lib/jenkins/jobs/Peloton/workspace/tests/expression		99.1 %	343 / 346	97.3 %	72 / 74
/var/lib/jenkins/jobs/Peloton/workspace/tests/index		98.9 %	277 / 280	97.2 %	35 / 36
/var/lib/jenkins/jobs/Peloton/workspace/tests/language		97.1 %	135 / 139	96.3 %	52 / 54
/var/lib/jenkins/jobs/Peloton/workspace/tests/logging		98.4 %	243 / 247	93.0 %	40 / 43
/var/lib/jenkins/jobs/Peloton/workspace/tests/networking		100.0 %	34 / 34	88.9 %	16 / 18
/var/lib/jenkins/jobs/Peloton/workspace/tests/planner		100.0 %	3 / 3	88.9 %	8 / 9
/var/lib/jenkins/jobs/Peloton/workspace/tests/storage		99.5 %	421 / 423	90.8 %	69 / 76

GROUP ASSIGNMENTS

Logging	Multi-Threaded Queries
Constraints	Garbage Collection
UDFs	Memcache
Query Planning	Concurrency Control
Statistics	Query Compilation

NEXT CLASS

Final Exam Review

Ankur Goyal (CMU'15 / MemSQL)