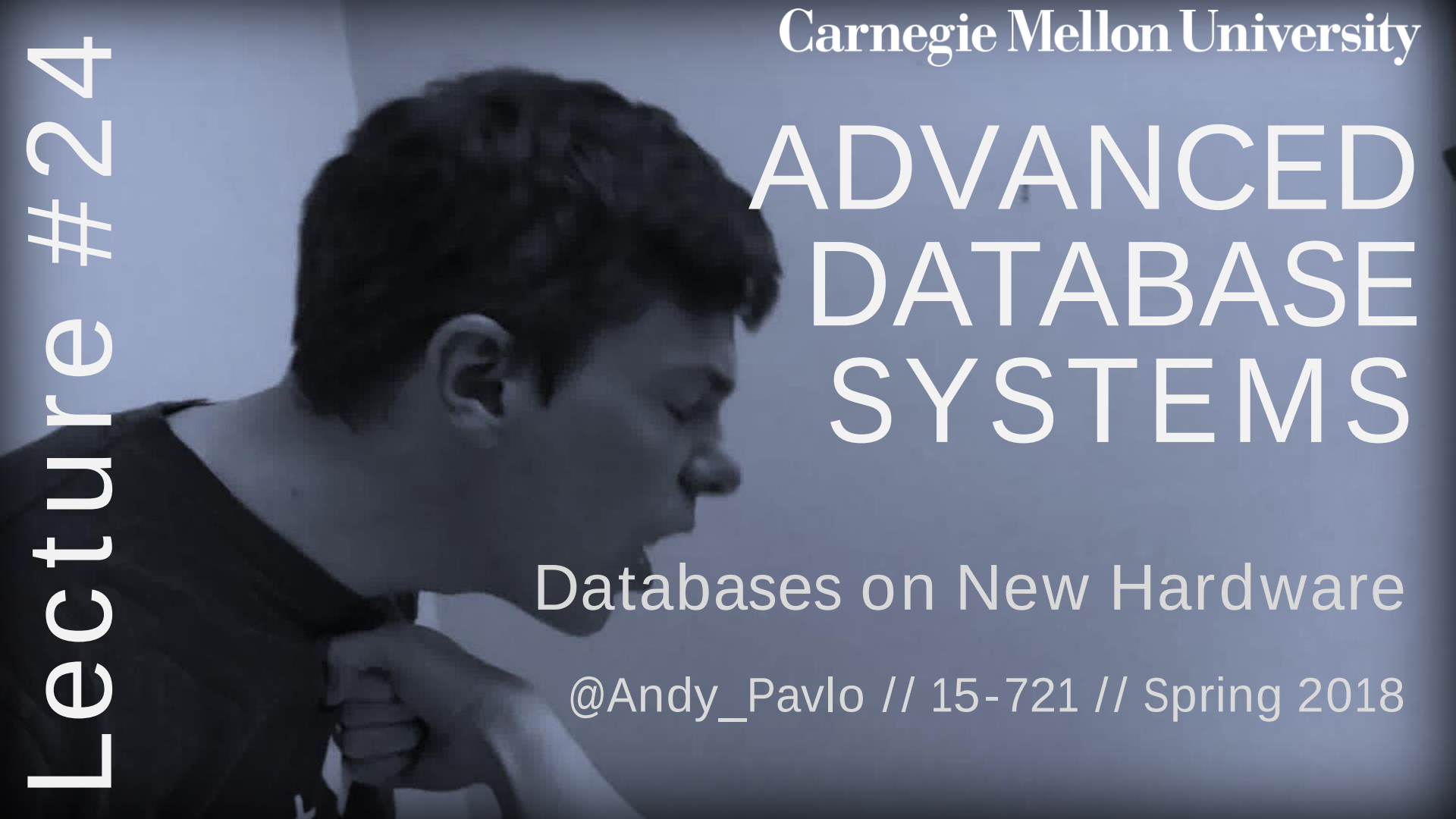

Carnegie Mellon University
Lecture #24
ADVANCED DATABASE SYSTEMS
Databases on New Hardware
@Andy_Pavlo // 15-721 // Spring 2018

ADMINISTRIVIA

Snowflake Guest: May 2th @ 3:00pm

Final Exam Handout: May 2nd

Code Review #2: May 2nd @ 11:59pm

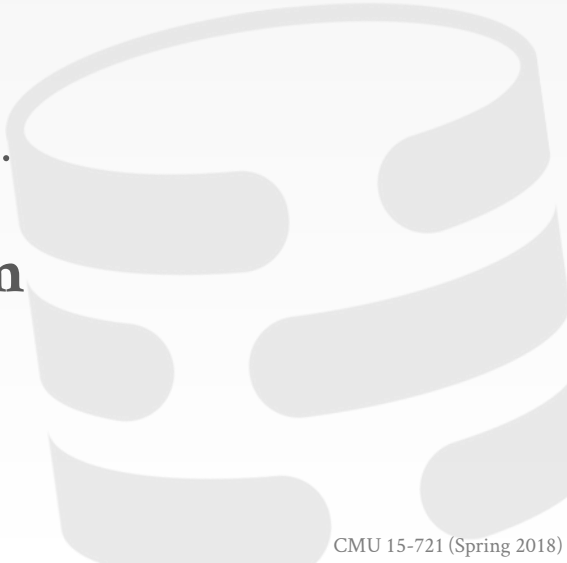
→ We will use the same group pairings as before.

Final Presentations: May 14th @ 8:30am

→ **GHC 4303 (ignore schedule!)**

→ 12 minutes per group

→ Food and prizes for everyone!



ADMINISTRIVIA

Course Evaluation

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

https://cmu.smartevals.com/

Cours

→ Plea

→ I act

→ Tak

https:/

MENU

ANDREW PAVLO

Carnegie Mellon University

Customize columns

Instructor	Questions	Text Responses
PAVLO	Question: Comments	Andy smells terrible. He has this weird odor like a he rolled around in trash can and then let feral cats piss all over him. I don't think he washes those shitty database shirts that he wears for the videos. In addition to filling out this course evaluation, I also filled a complaint with the Alleghany County Health Department. I don't think a grown man should smell as bad as Andy does. I guess the database part of the course was okay but man that odor was tough to deal with each class.
PAVLO	Question: Comments	[redacted]
PAVLO	Question: Comments	[redacted]

DATABASE HARDWARE

People have been thinking about using hardware to accelerate DBMSs for decades.

1980s: Database Machines

2000s: FPGAs + Appliances

2010s: FPGAs + GPUs

DATABASE MACHINES: AN IDEA WHOSE TIME HAS PASSED?
A CRITIQUE OF THE FUTURE OF DATABASE MACHINES
University of Wisconsin 1983

TODAY'S AGENDA

Non-Volatile Memory

GPU Acceleration

Hardware Transactional Memory

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

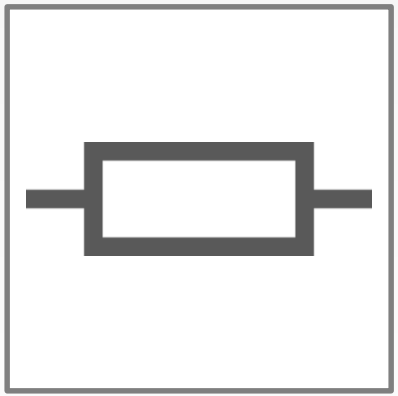
First devices will be block-addressable (NVMe)

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.

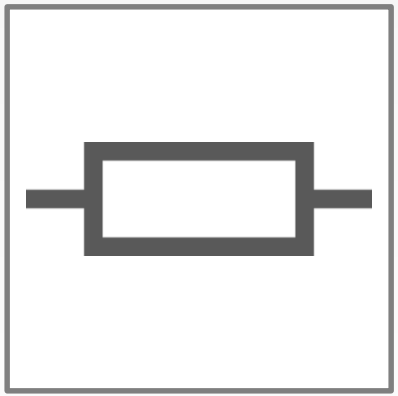
A two-terminal device whose resistance depends on the voltage applied to it, but when that voltage is turned off it permanently **remembers** its last resistive state.

TWO CENTURIES OF MEMRISTORS
Nature Materials 2012

FUNDAMENTAL ELEMENTS OF CIRCUITS

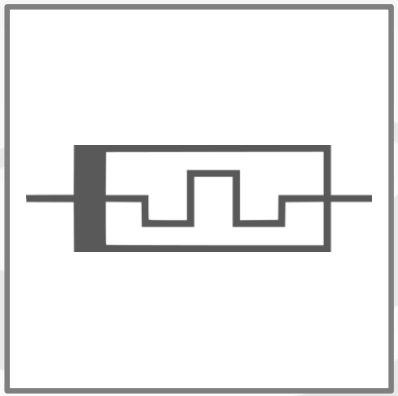
Capacitor (ca. 1745)

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

TECHNOLOGIES

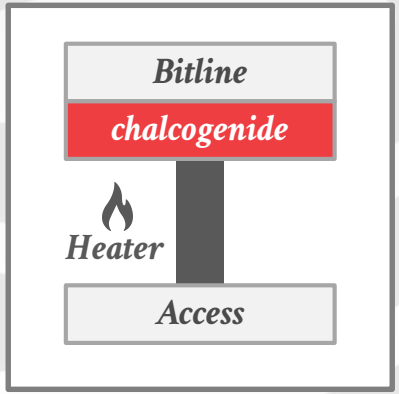
Phase-Change Memory (PRAM)
Resistive RAM (ReRAM)
Magnetoresistive RAM (MRAM)

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'.

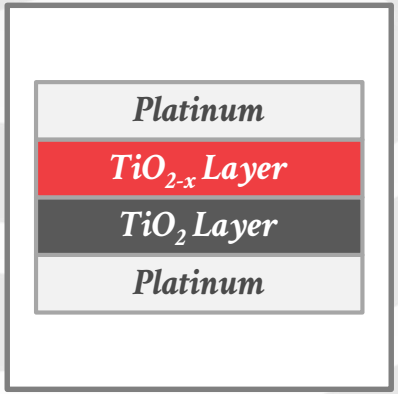


PHASE CHANGE MEMORY ARCHITECTURE AND THE QUEST FOR SCALABILITY
Communications of the ACM 2010

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



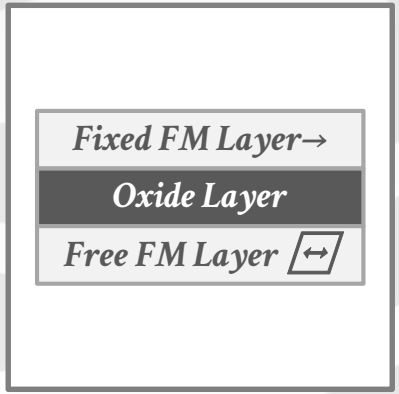
HOW WE FOUND THE MISSING MEMRISTOR
IEEE Spectrum 2008

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

WHY THIS IS FOR REAL THIS TIME

Industry has agreed to standard technologies and form factors.

Linux and Microsoft have added support for NVM in their kernels (DAX).

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

NVM DIMM FORM FACTORS

NVDIMM-F (2015)

→ Flash only. Has to be paired with DRAM DIMM.

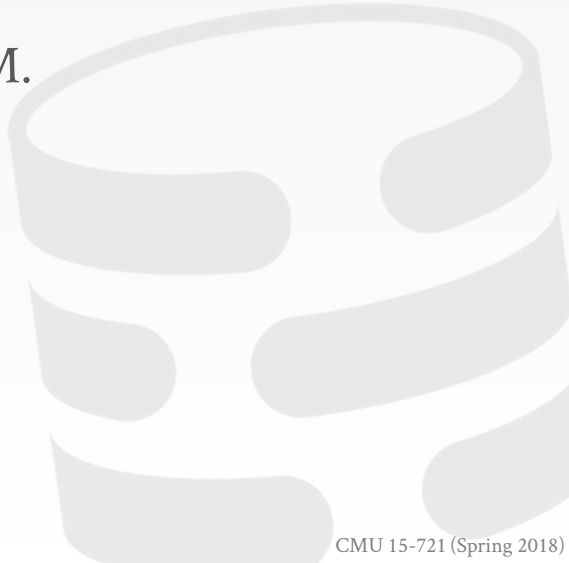
NVDIMM-N (2015)

→ Flash and DRAM together on the same DIMM.

→ Appears as volatile memory to the OS.

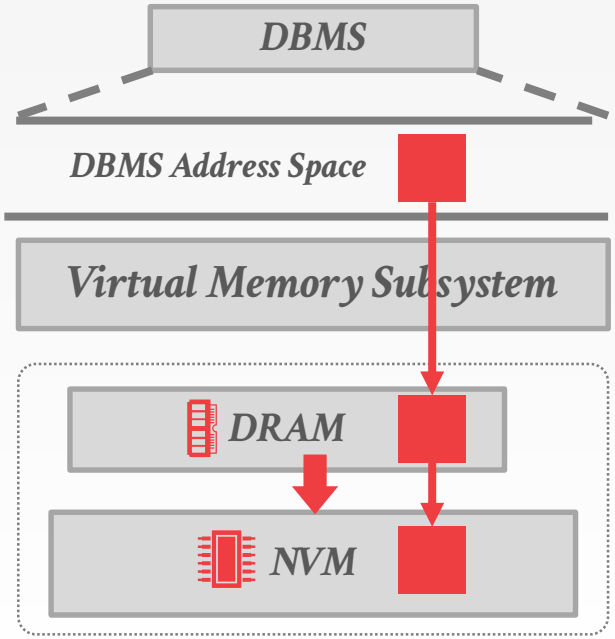
NVDIMM-P (2018)

→ True persistent memory. No DRAM or flash.

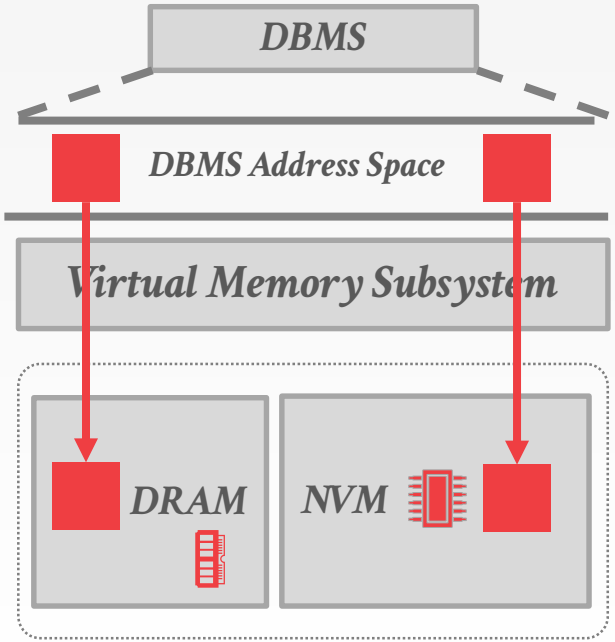


NVM CONFIGURATIONS

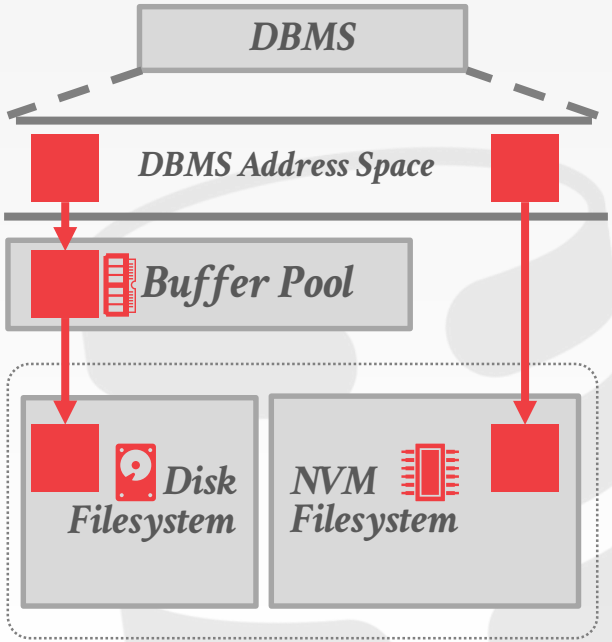
DRAM as Hardware-Managed Cache



NVM Next to DRAM



NVM as Persistent Memory



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.

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.

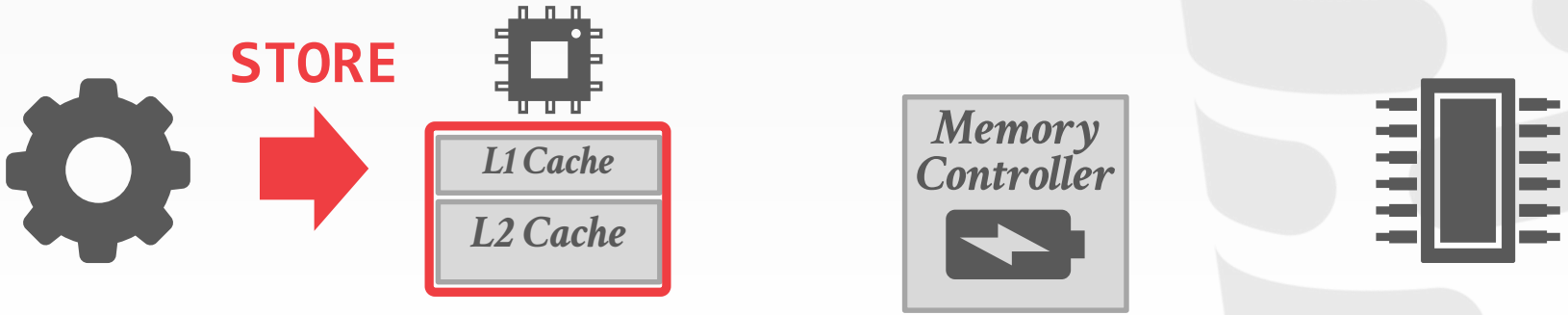
LET'S TALK ABOUT STORAGE & RECOVERY METHODS FOR NON-VOLATILE MEMORY DATABASE SYSTEMS
SIGMOD 2015

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.

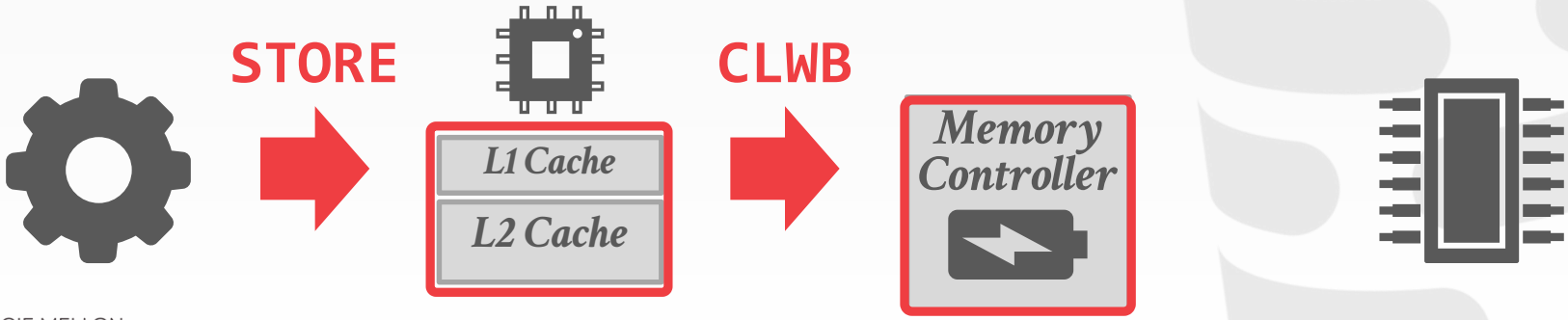


SYNCHRONIZATION

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.

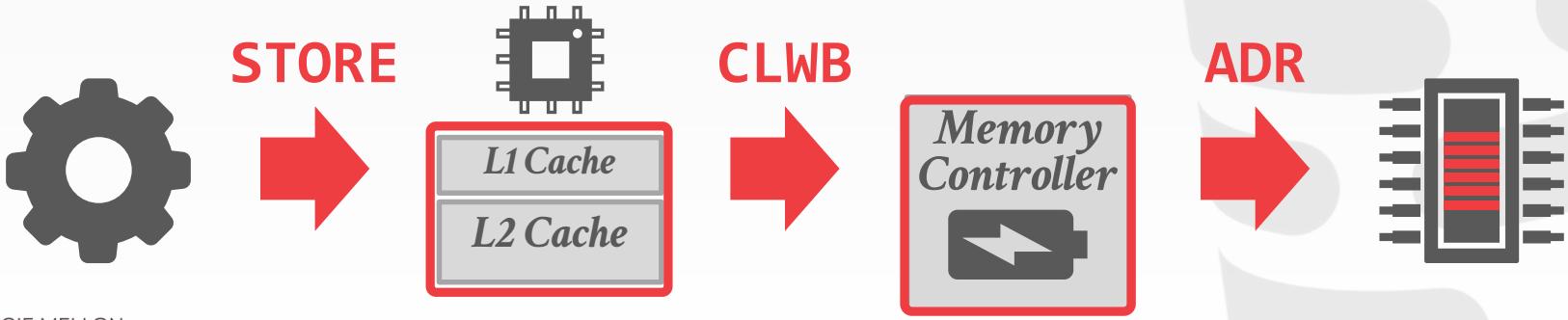


SYNCHRONIZATION

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

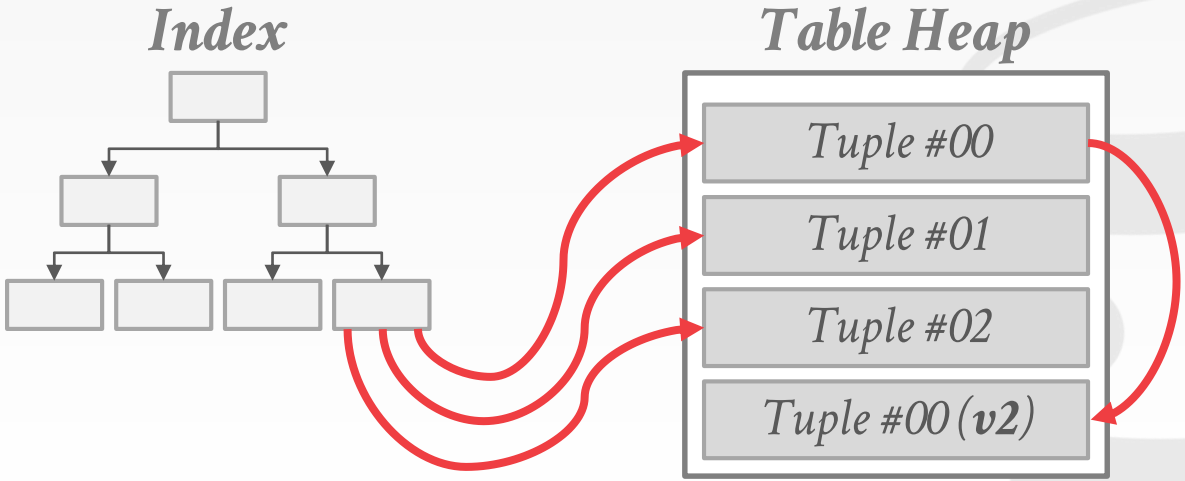
→ CPU decides when to move data from caches to DRAM.

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



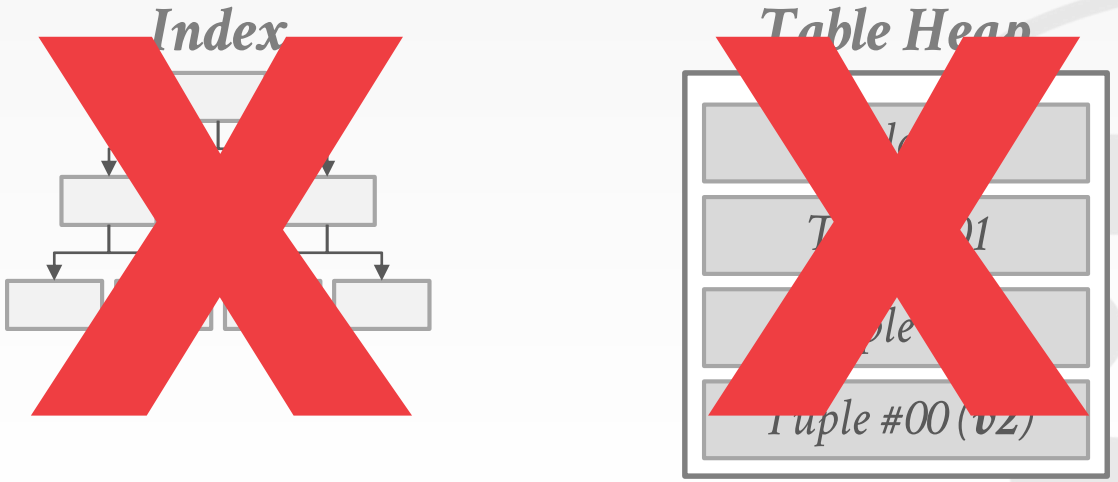
NAMING

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



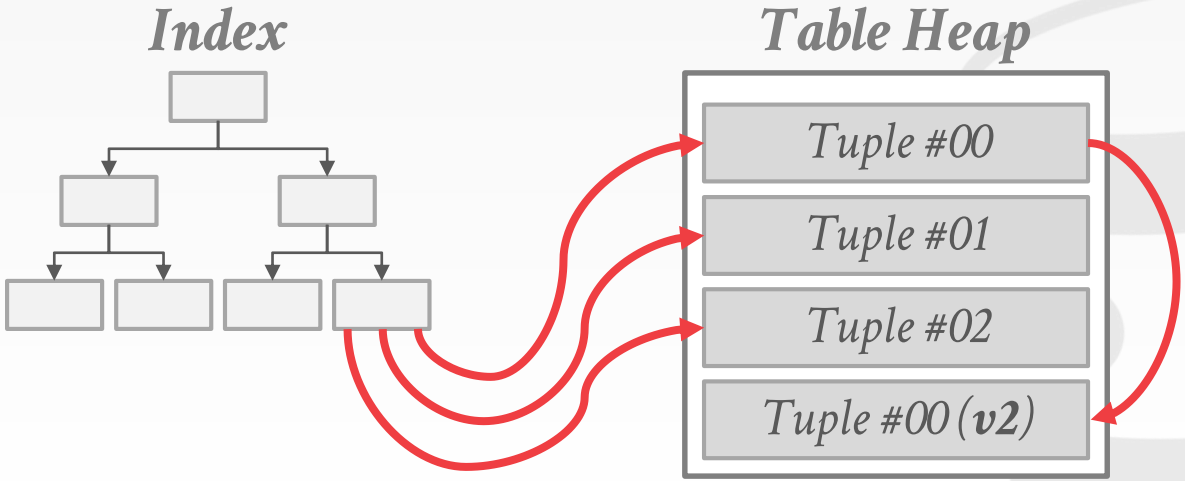
NAMING

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

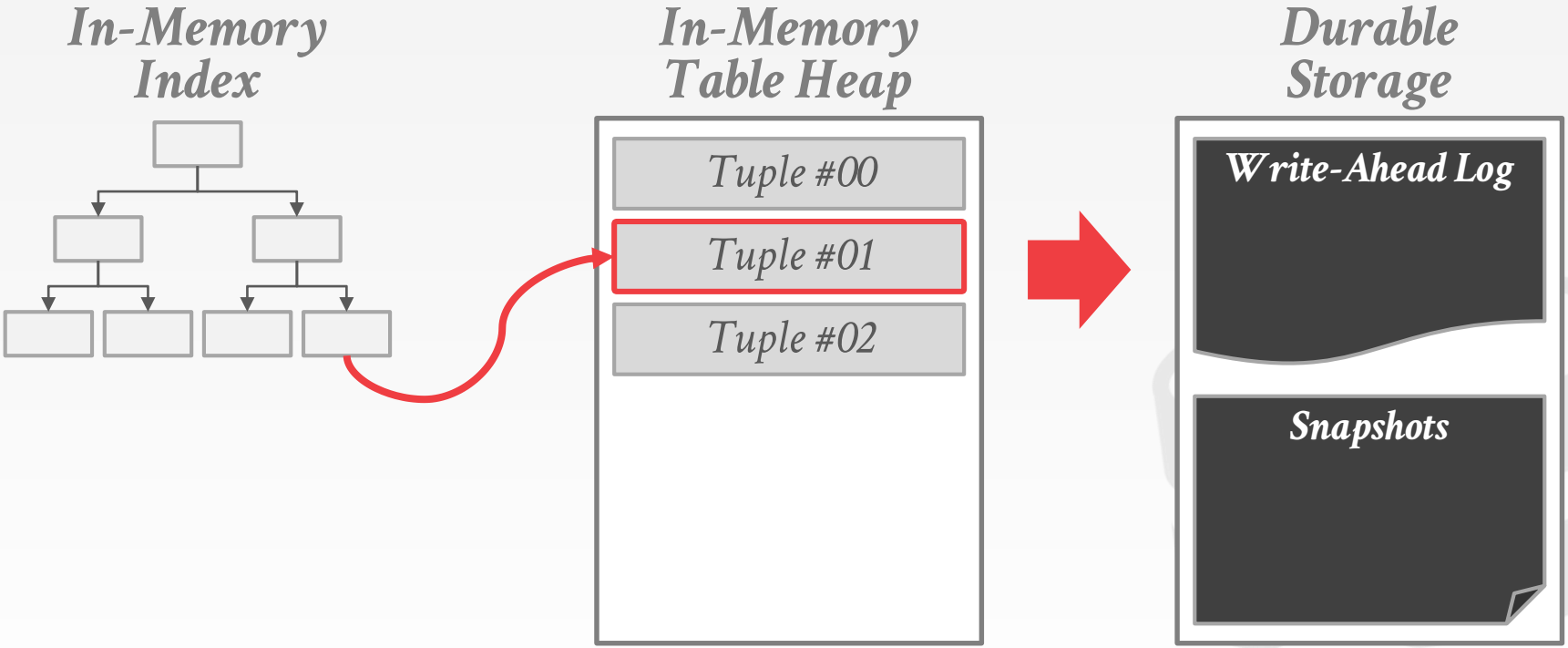
Choice #2: Copy-on-Write

→ Create a shadow copy of the table when updated.
→ No write-ahead log.
→ Example: LMDB

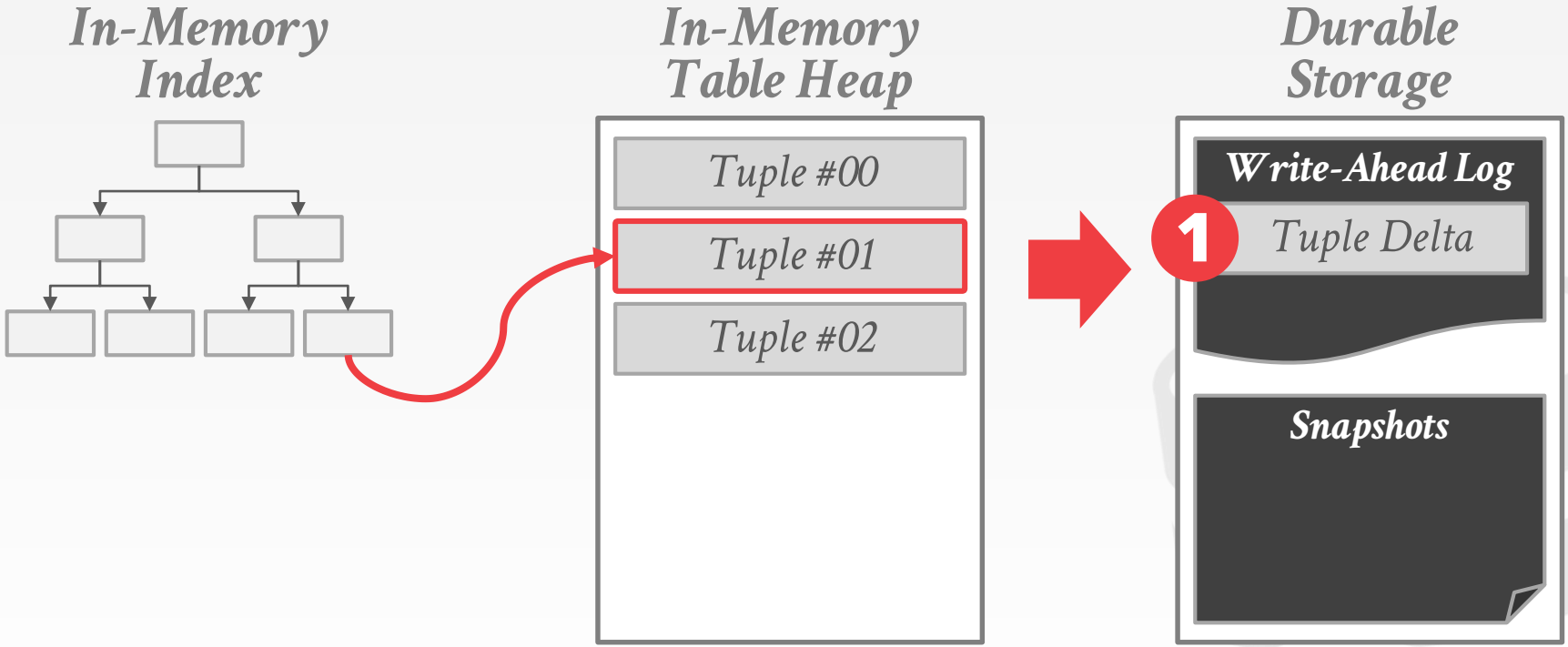
Choice #3: Log-structured

→ All writes are appended to log. No table heap.
→ Example: RocksDB

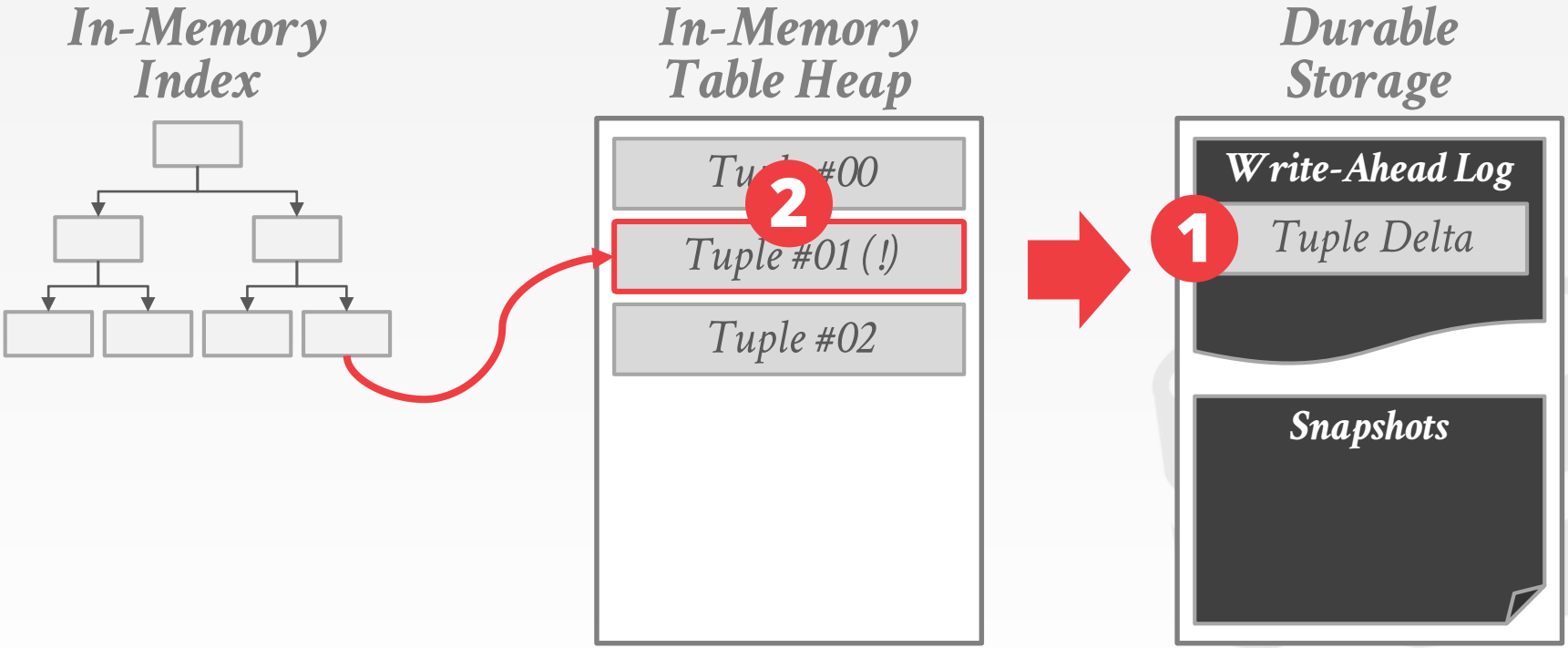
IN-PLACE UPDATES ENGINE



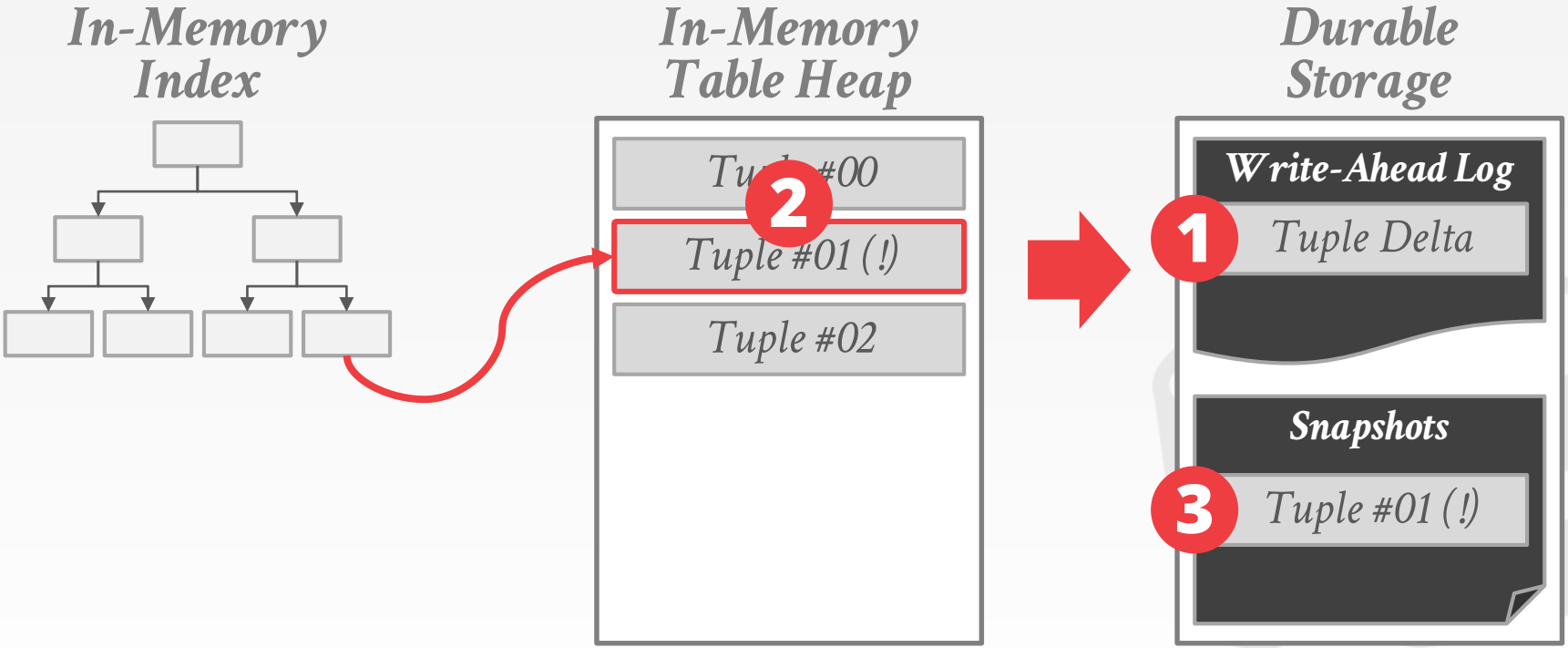
IN-PLACE UPDATES ENGINE



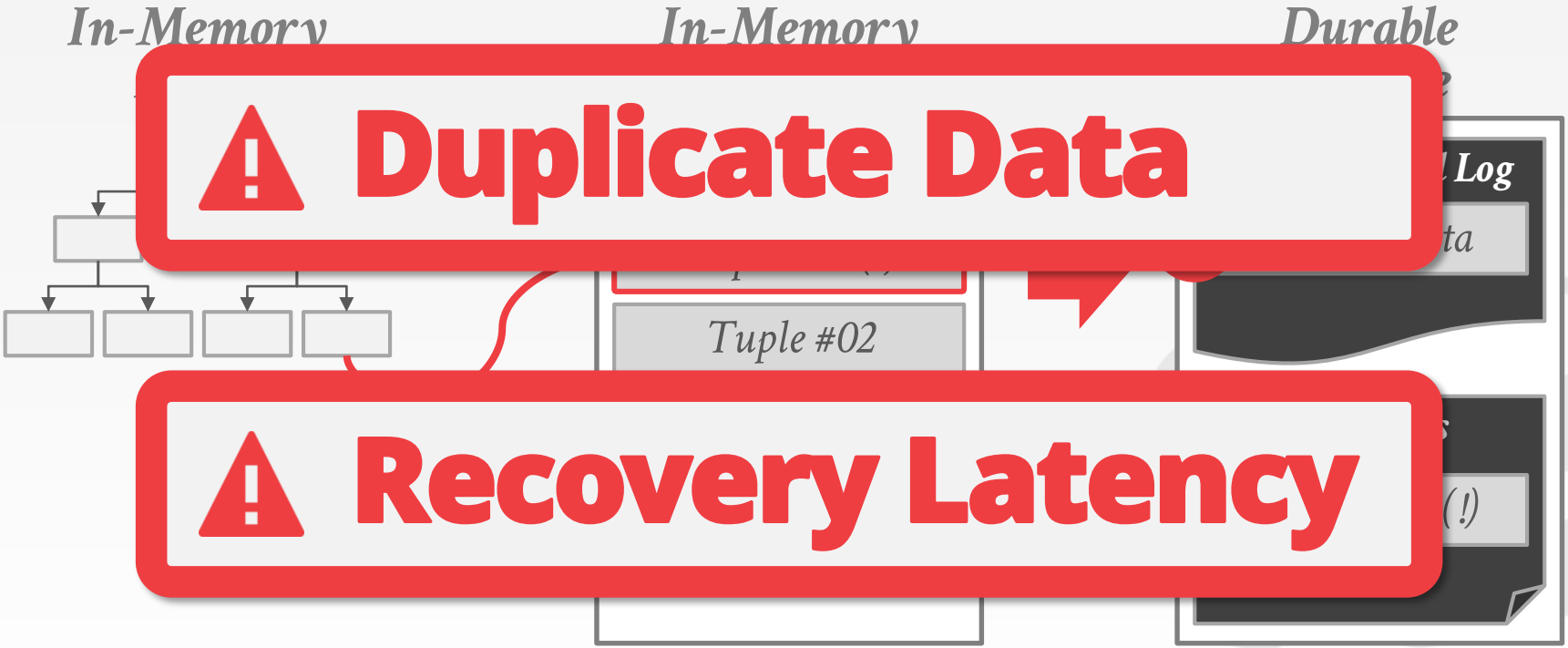
IN-PLACE UPDATES ENGINE



IN-PLACE UPDATES ENGINE



IN-PLACE UPDATES ENGINE



NVM-OPTIMIZED ARCHITECTURES

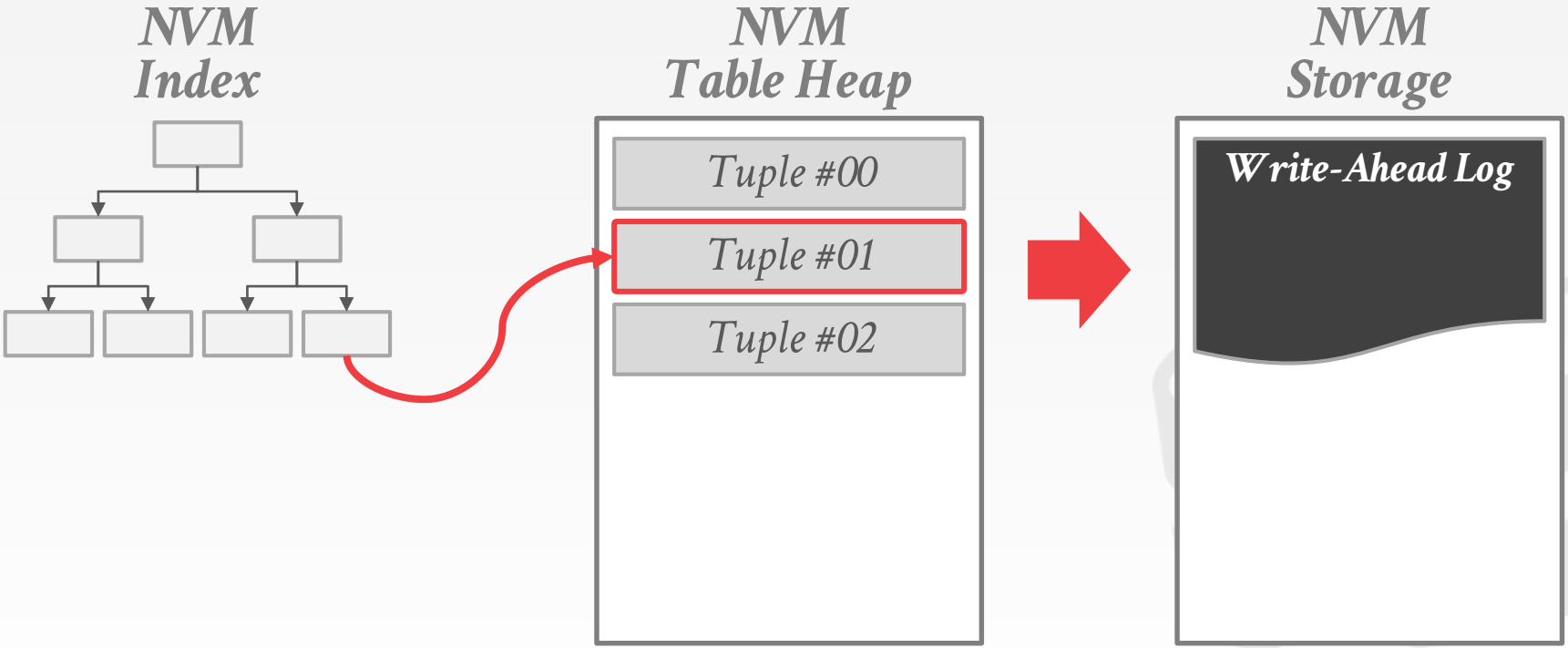
Leverage the allocator's non-volatile pointers to only record what changed rather than how it changed.

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

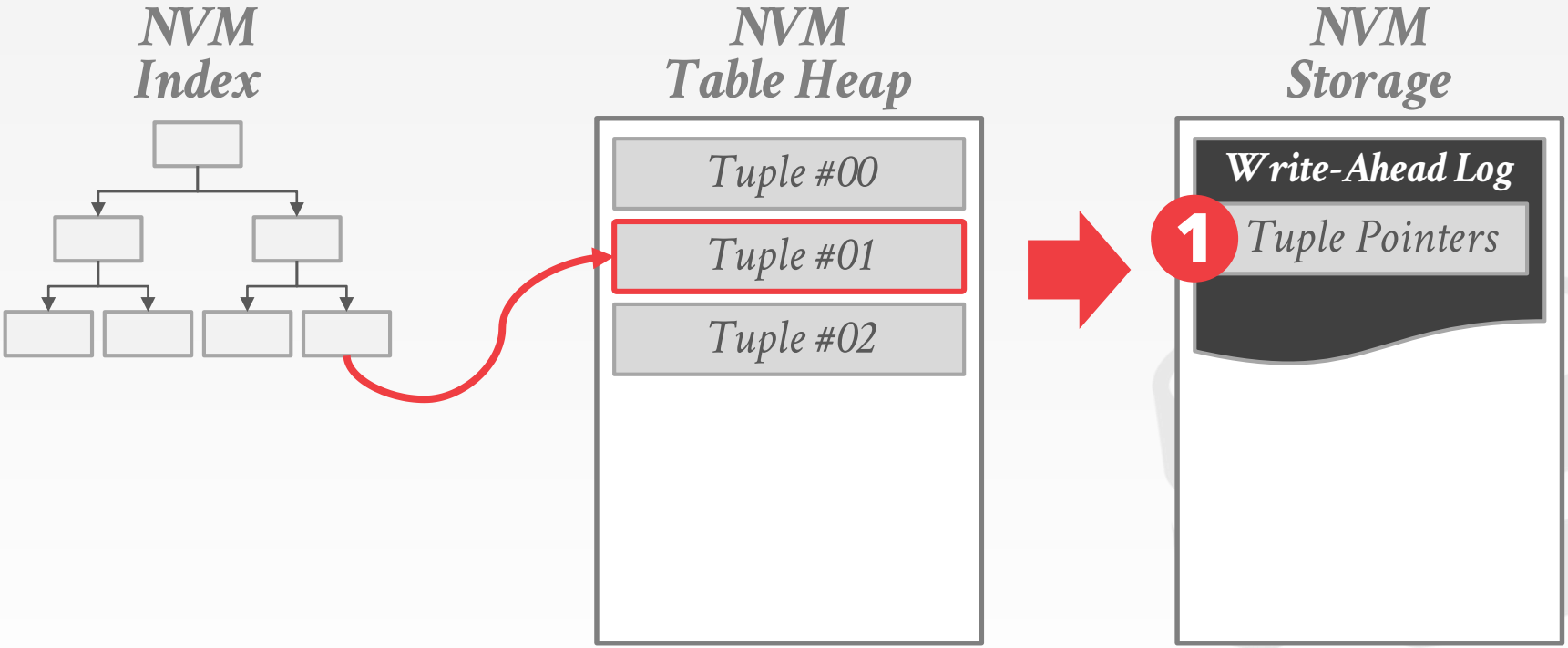
→ Dirty cache lines from an uncommitted txn can be flushed by hardware to the memory controller.

→ No REDO log because we flush all the changes to NVM at the time of commit.

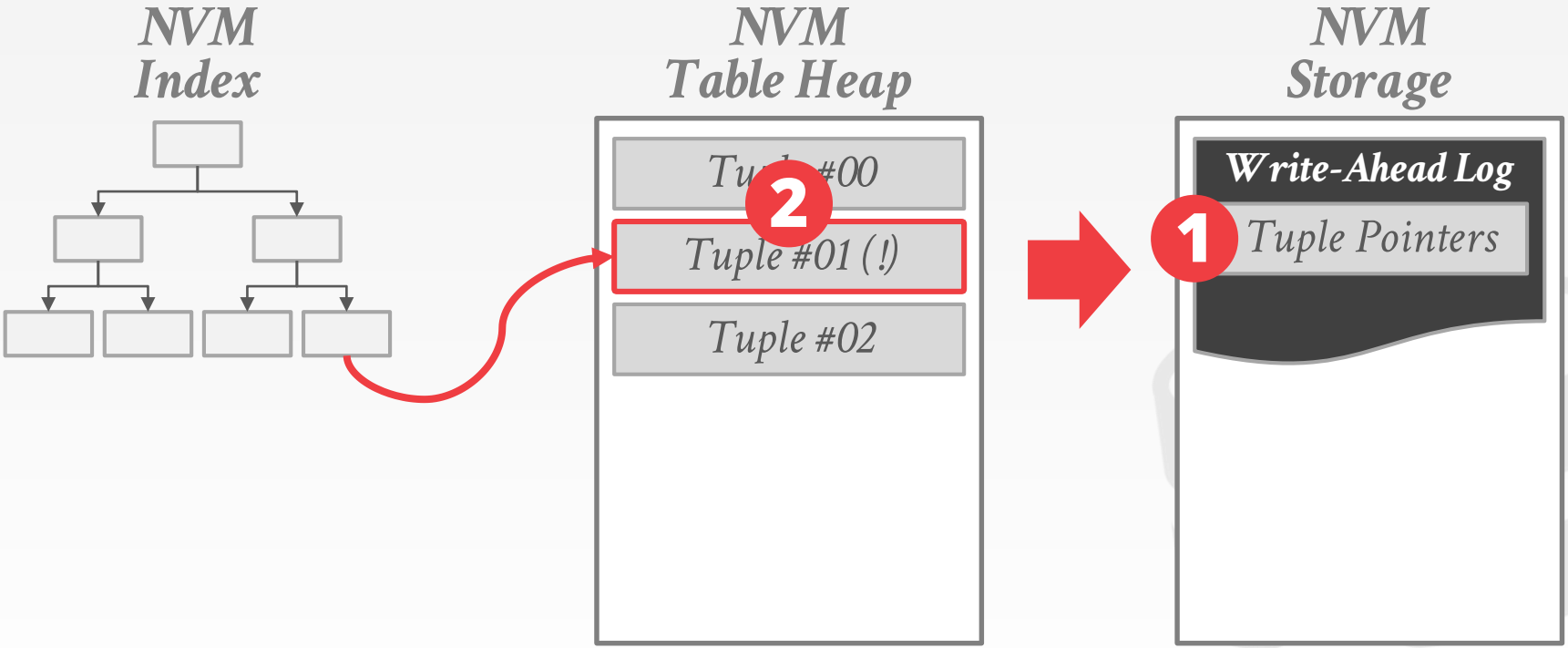
NVM IN-PLACE UPDATES ENGINE



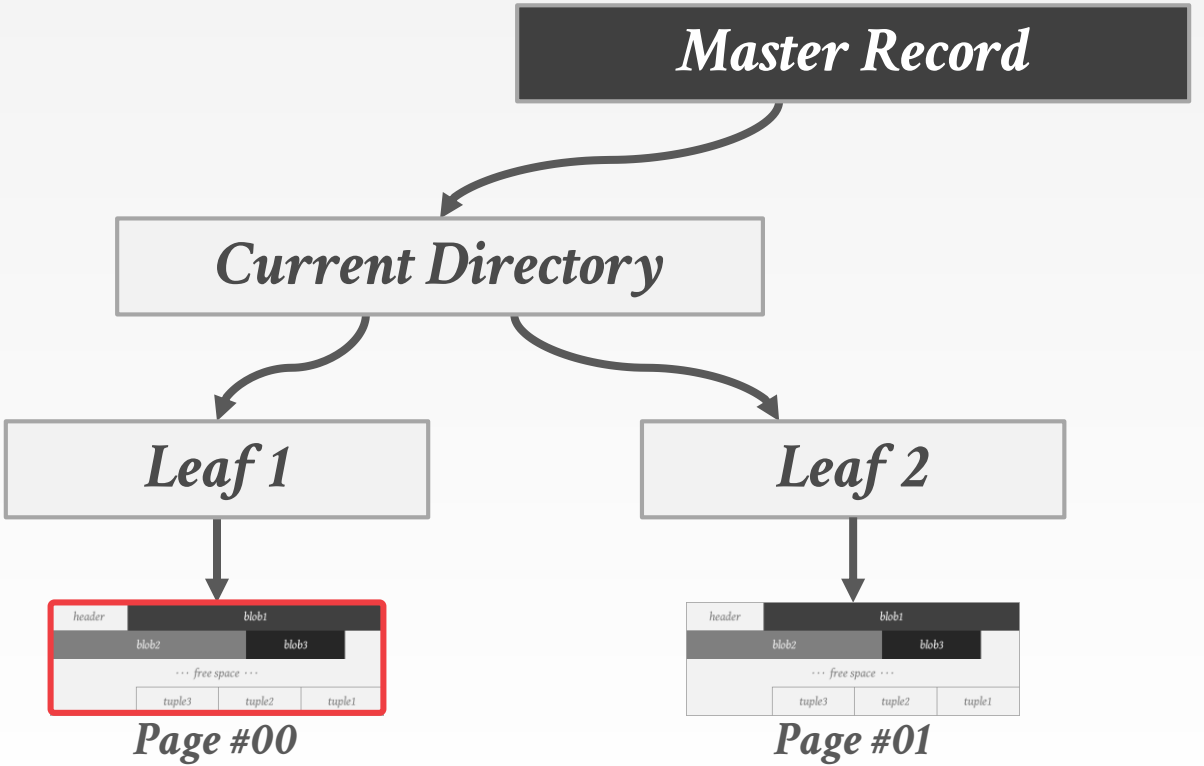
NVM IN-PLACE UPDATES ENGINE



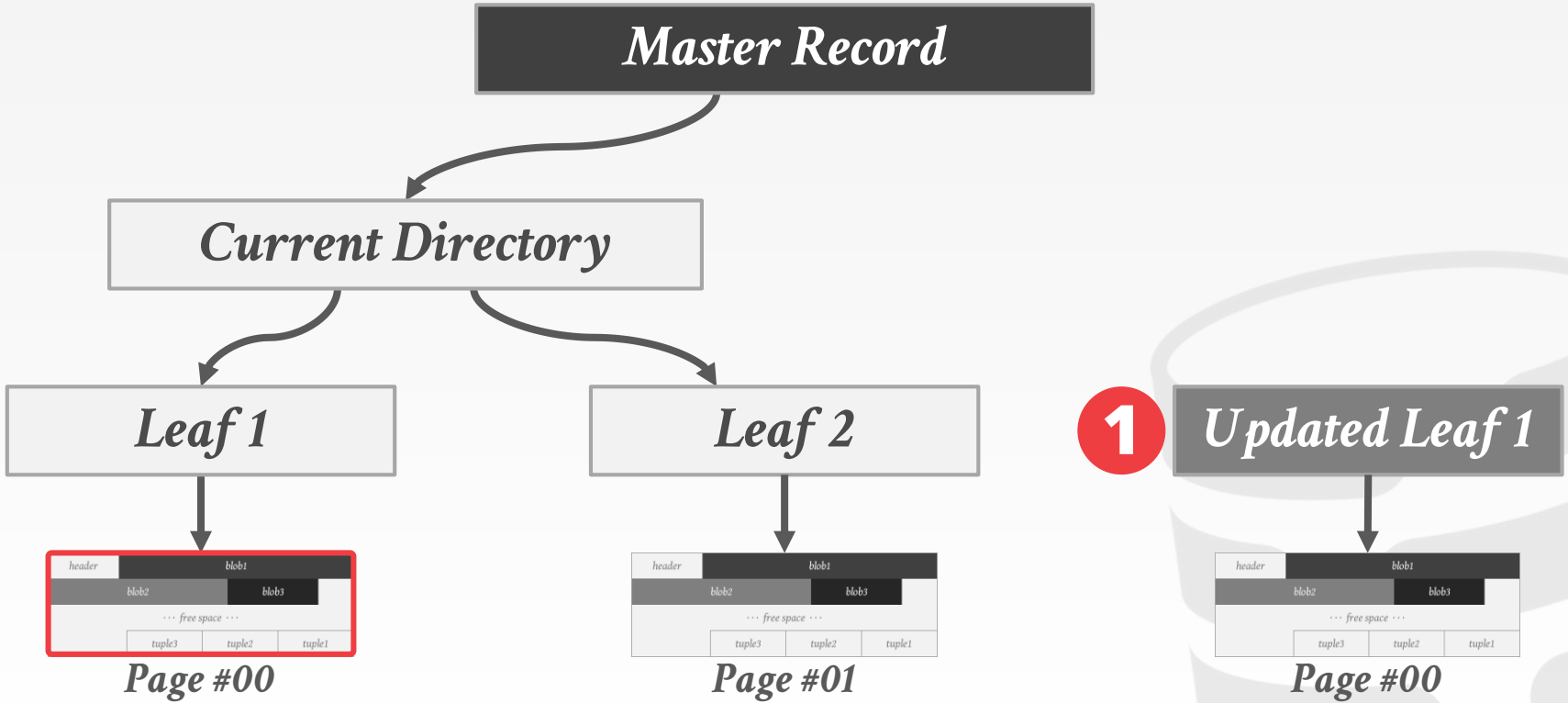
NVM IN-PLACE UPDATES ENGINE



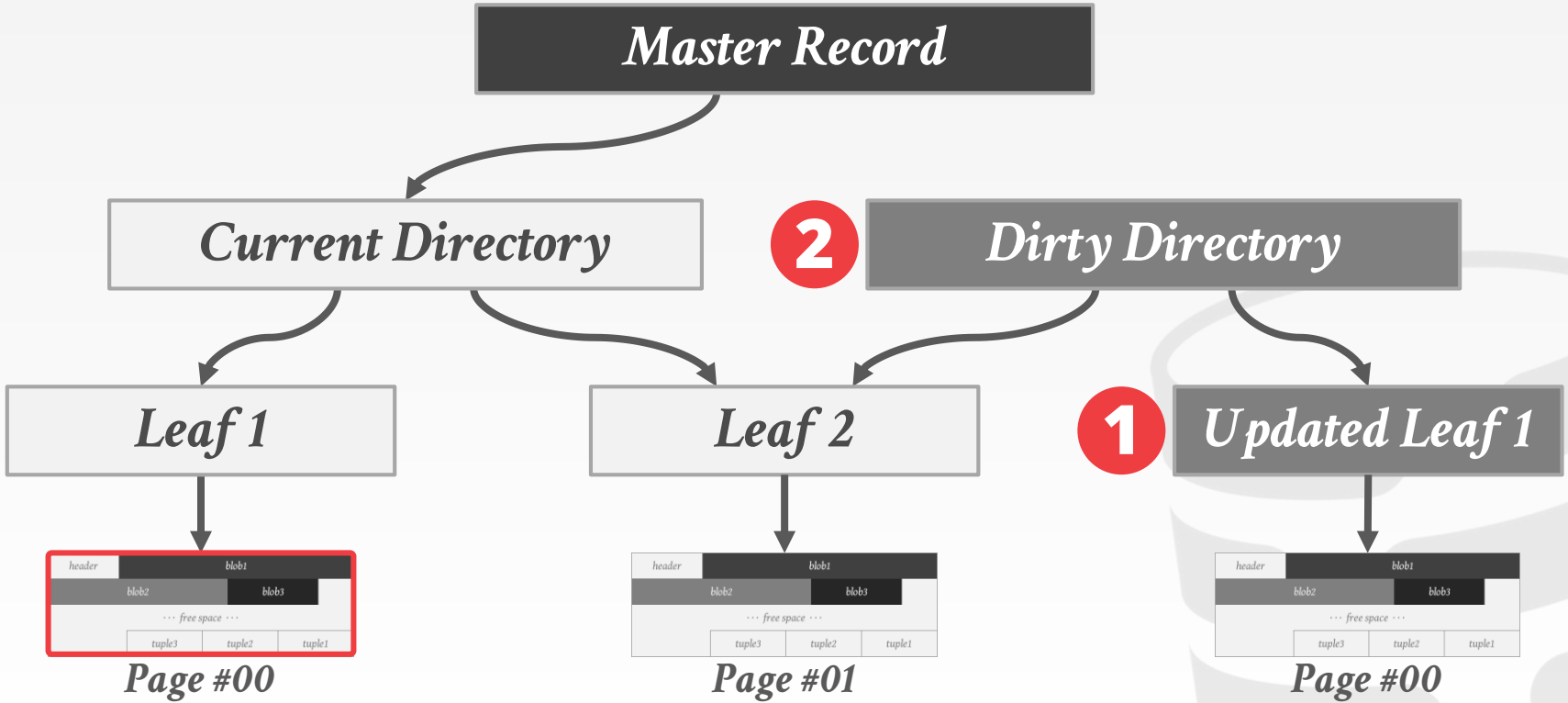
COPY-ON-WRITE ENGINE



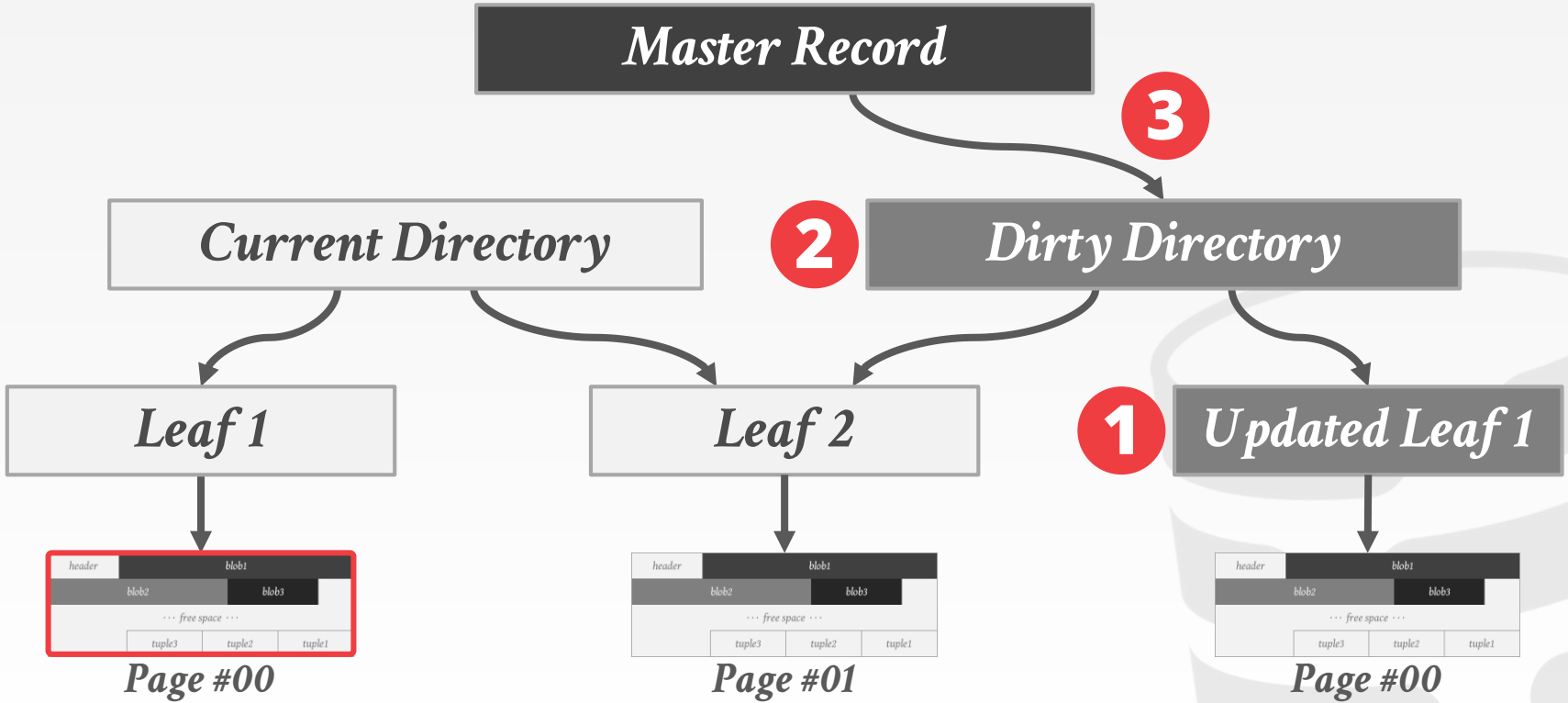
COPY-ON-WRITE ENGINE



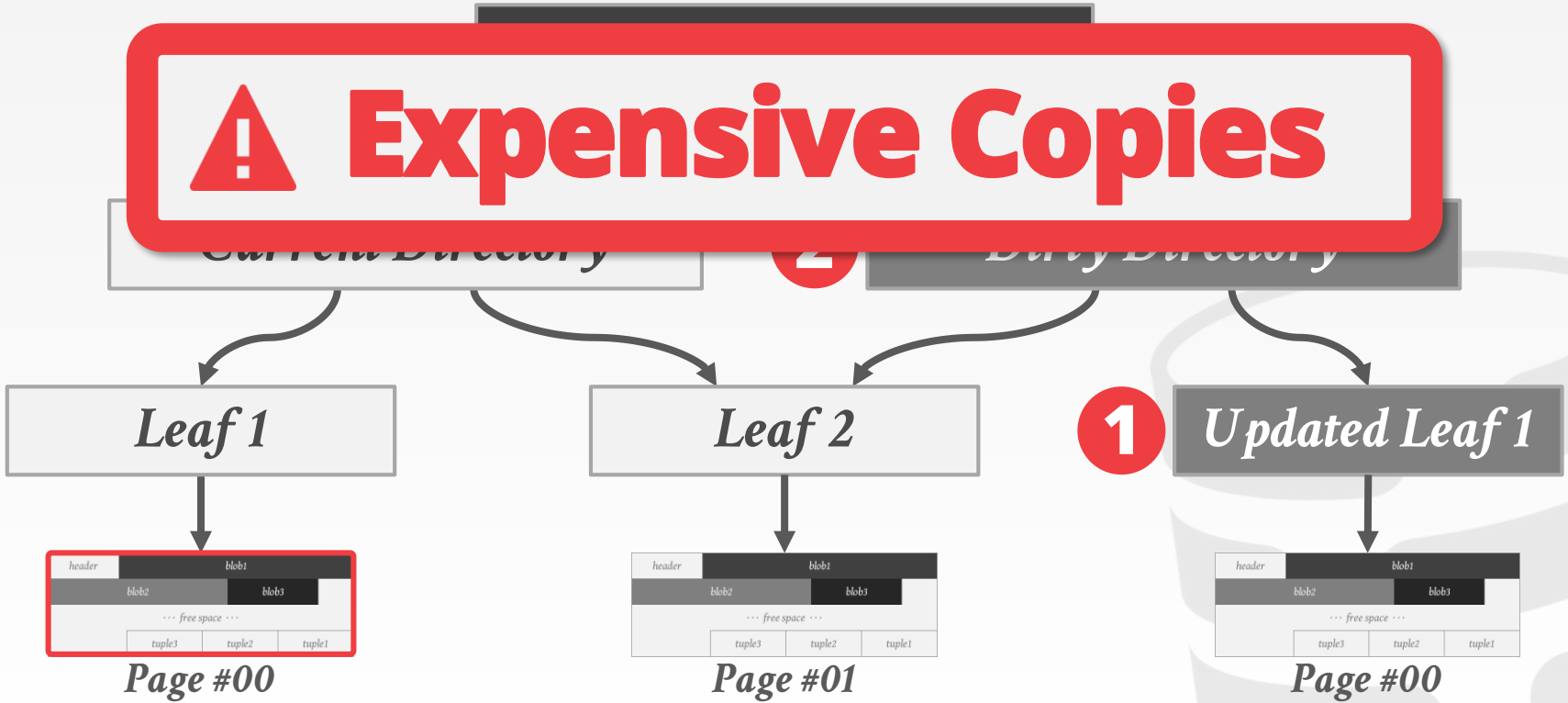
COPY-ON-WRITE ENGINE



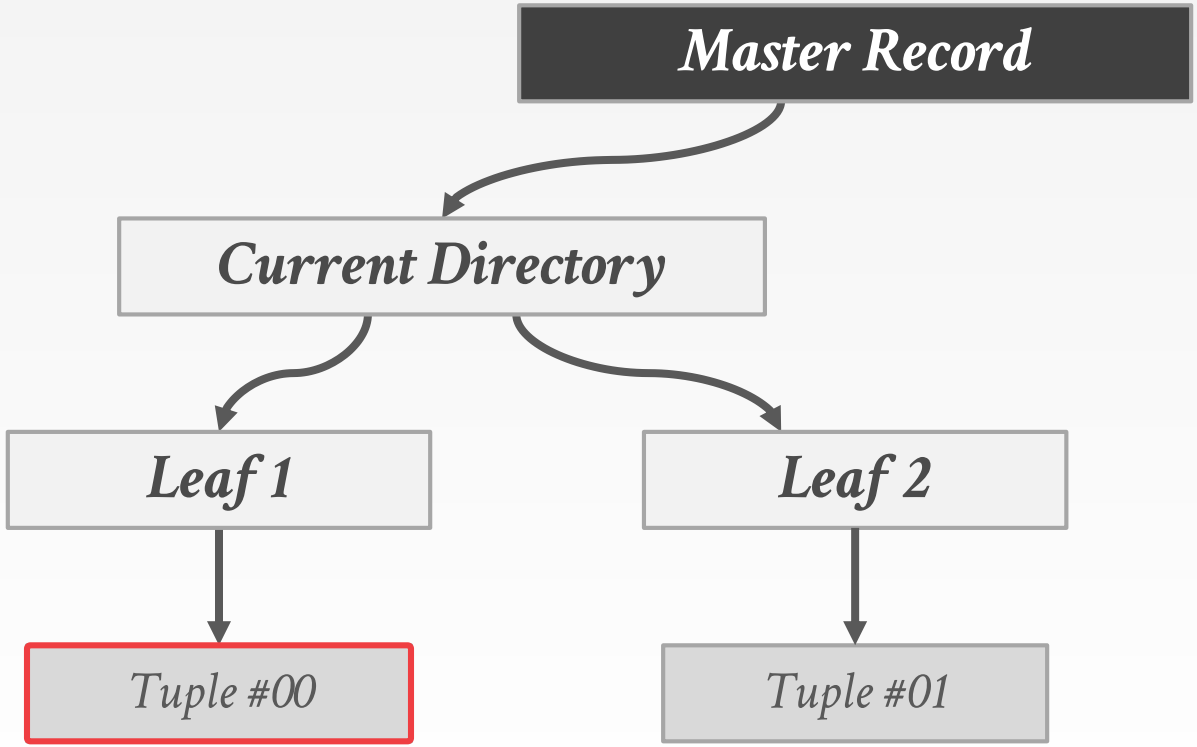
COPY-ON-WRITE ENGINE



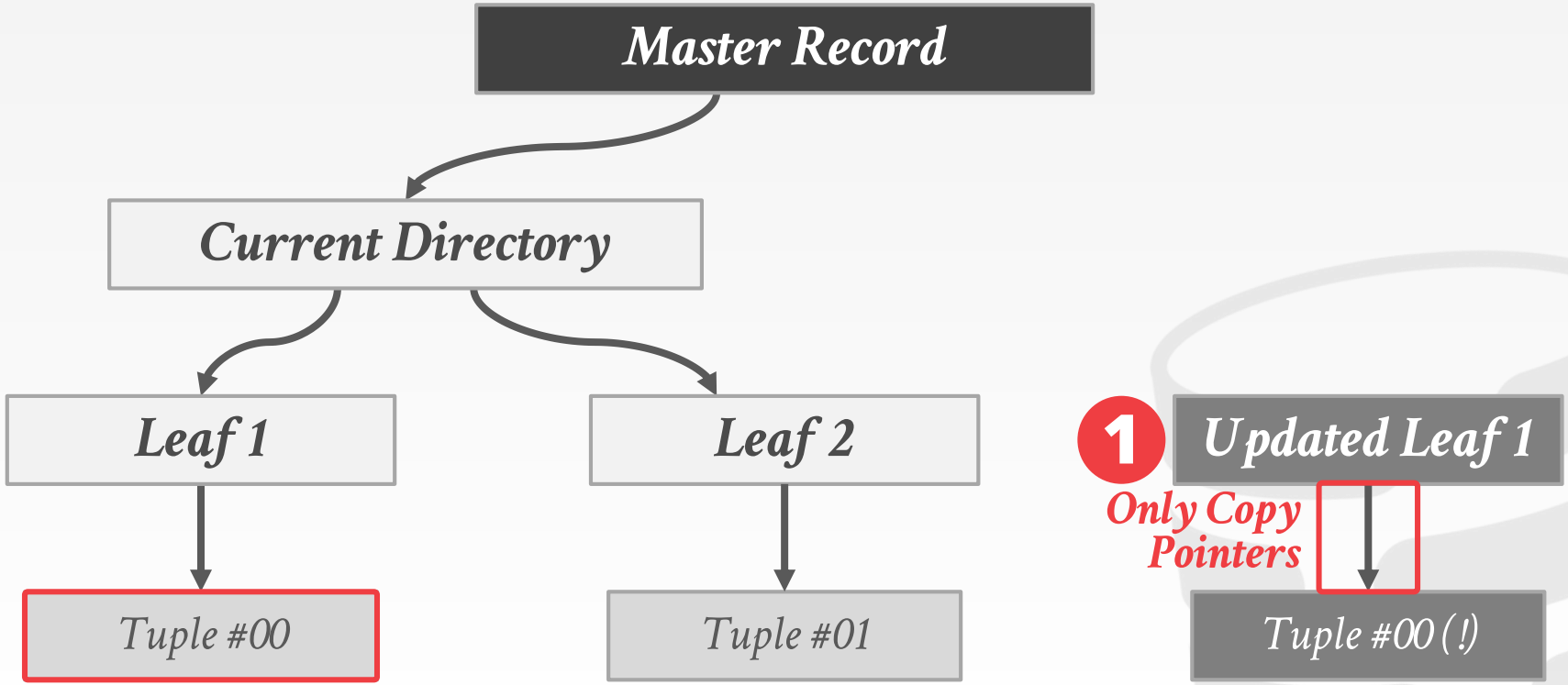
COPY-ON-WRITE ENGINE



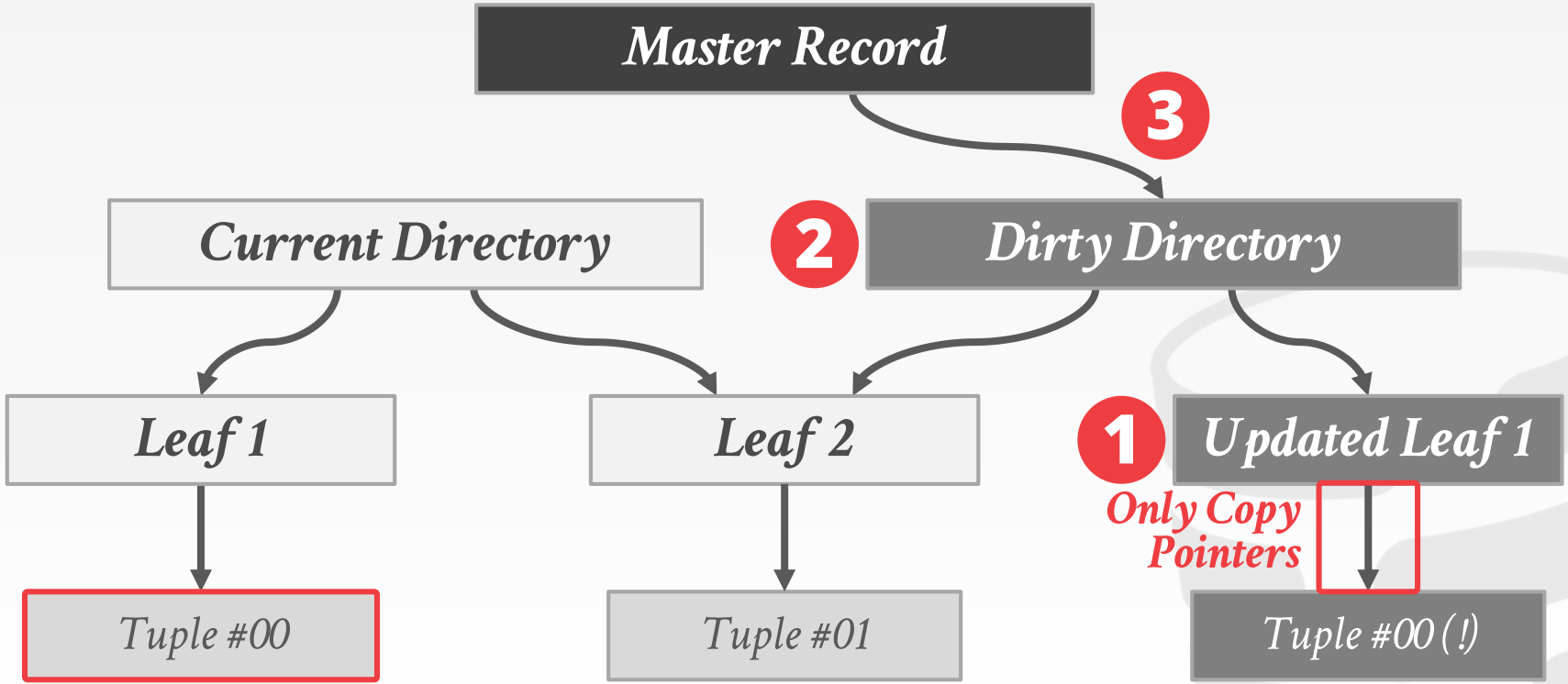
NVM COPY-ON-WRITE ENGINE



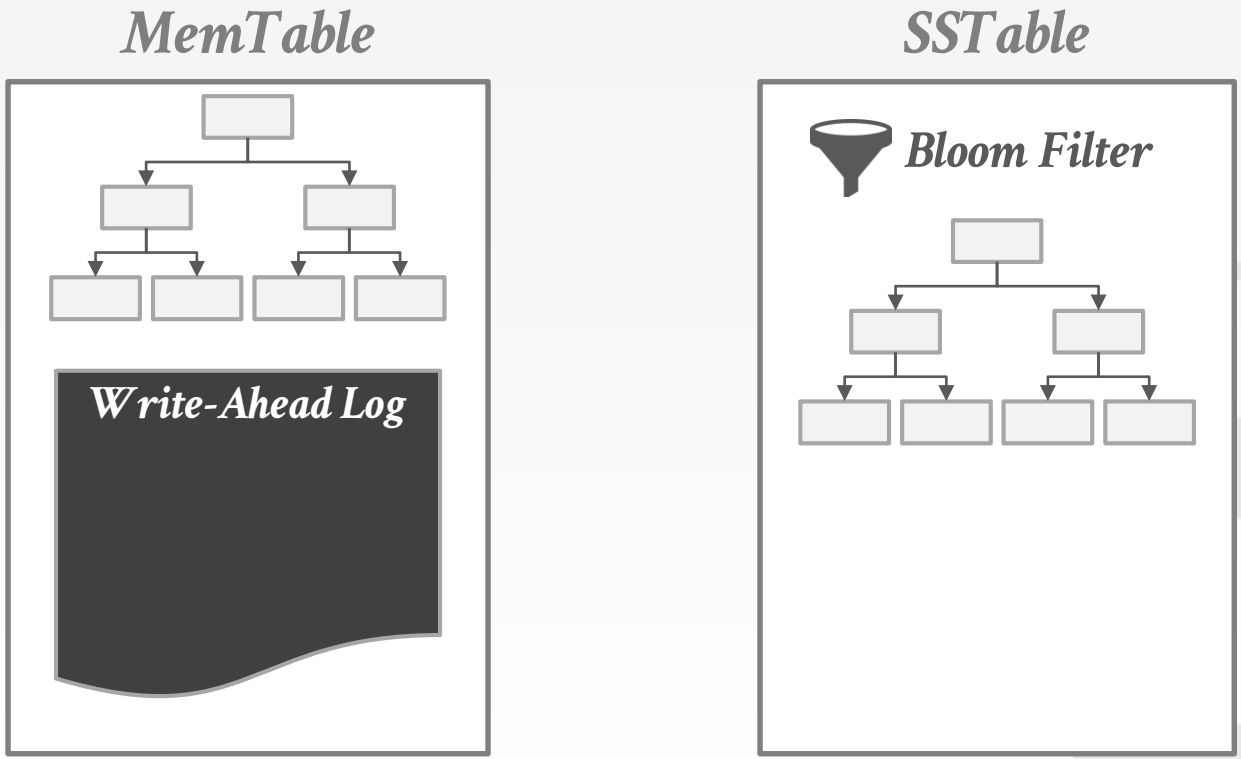
NVM COPY-ON-WRITE ENGINE



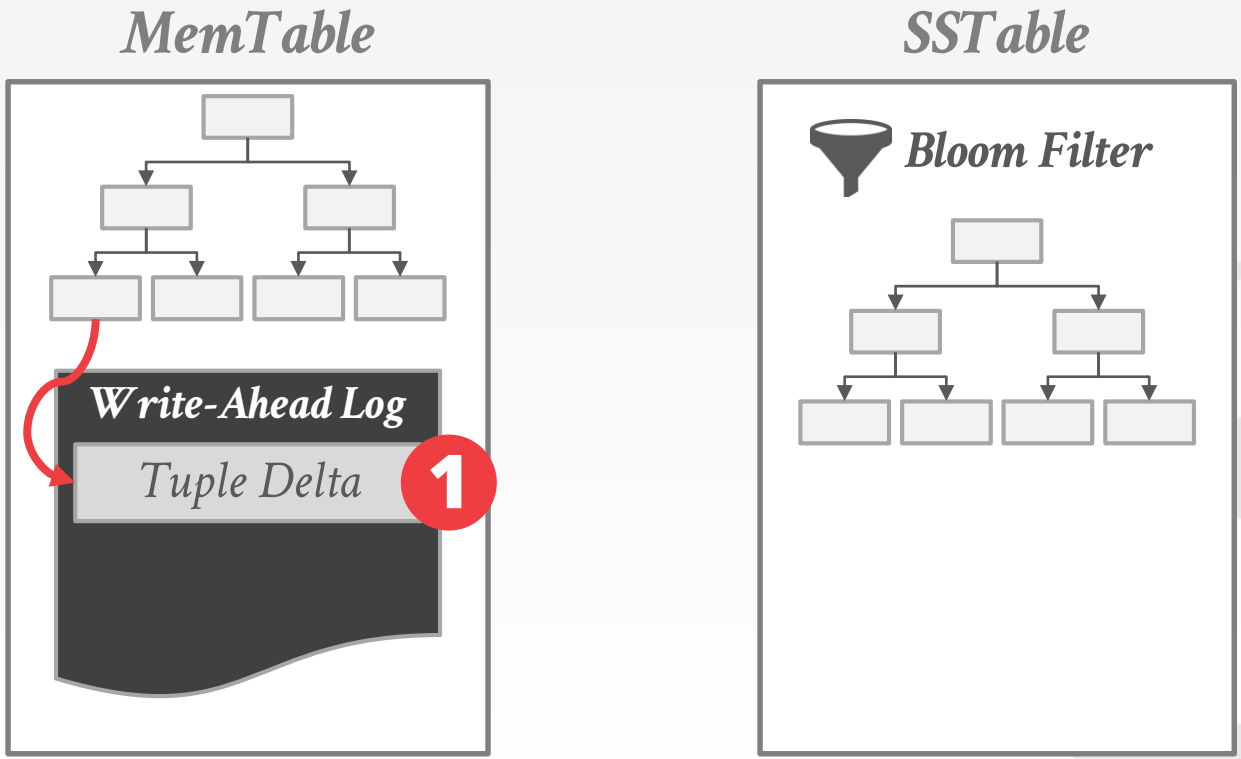
NVM COPY-ON-WRITE ENGINE



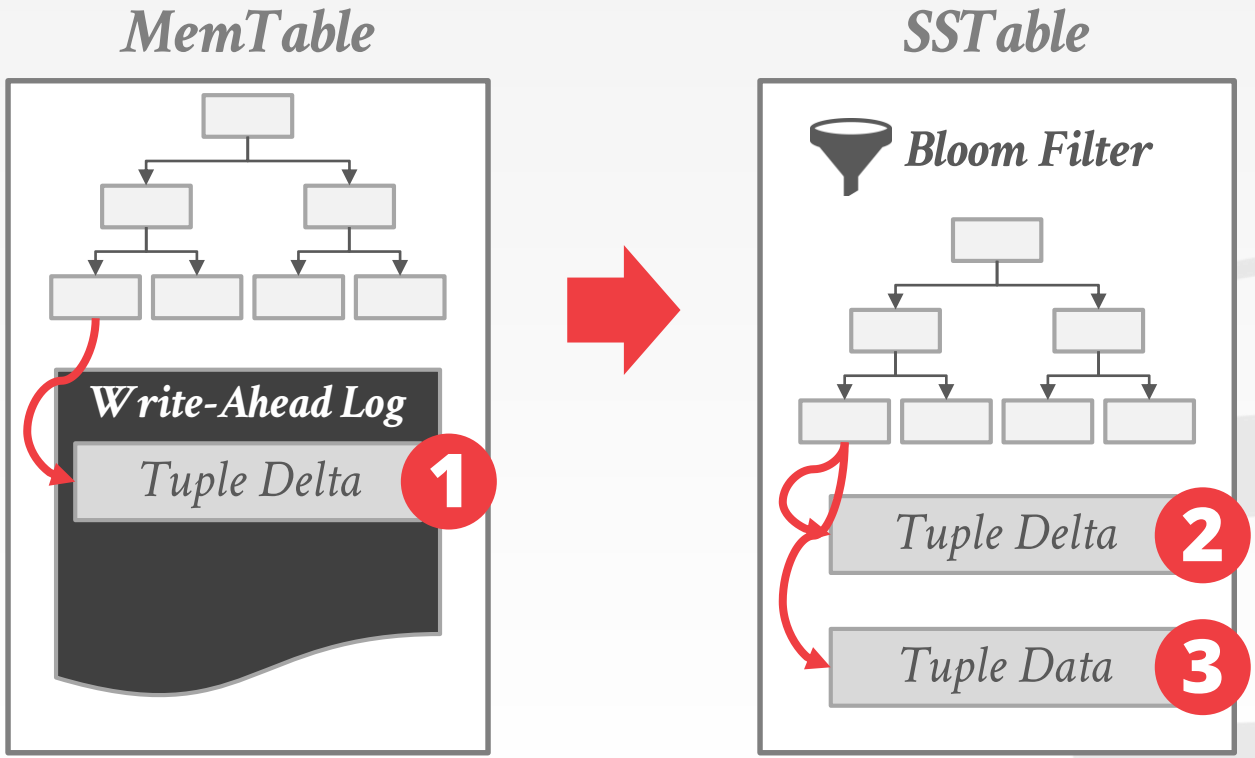
LOG-STRUCTURED ENGINE



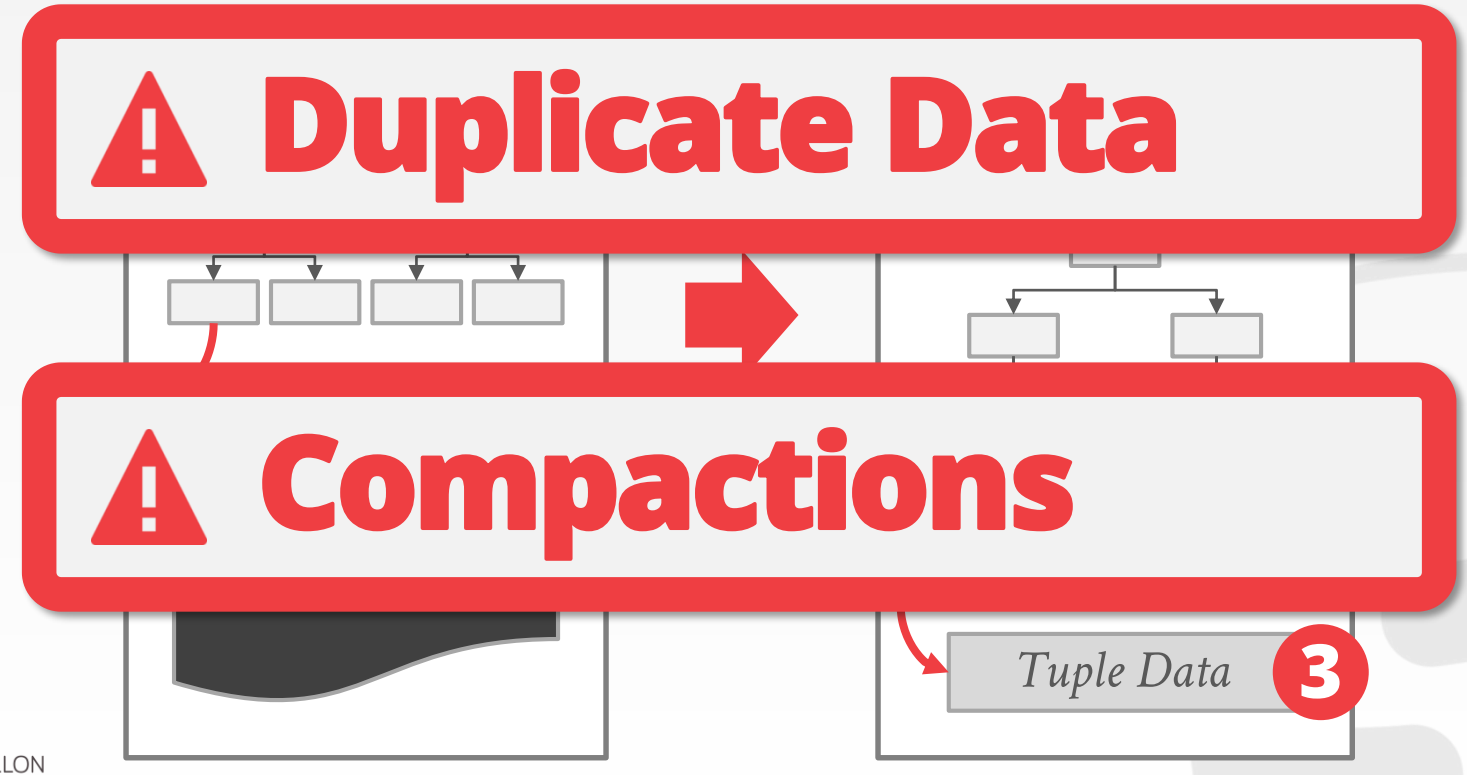
LOG-STRUCTURED ENGINE



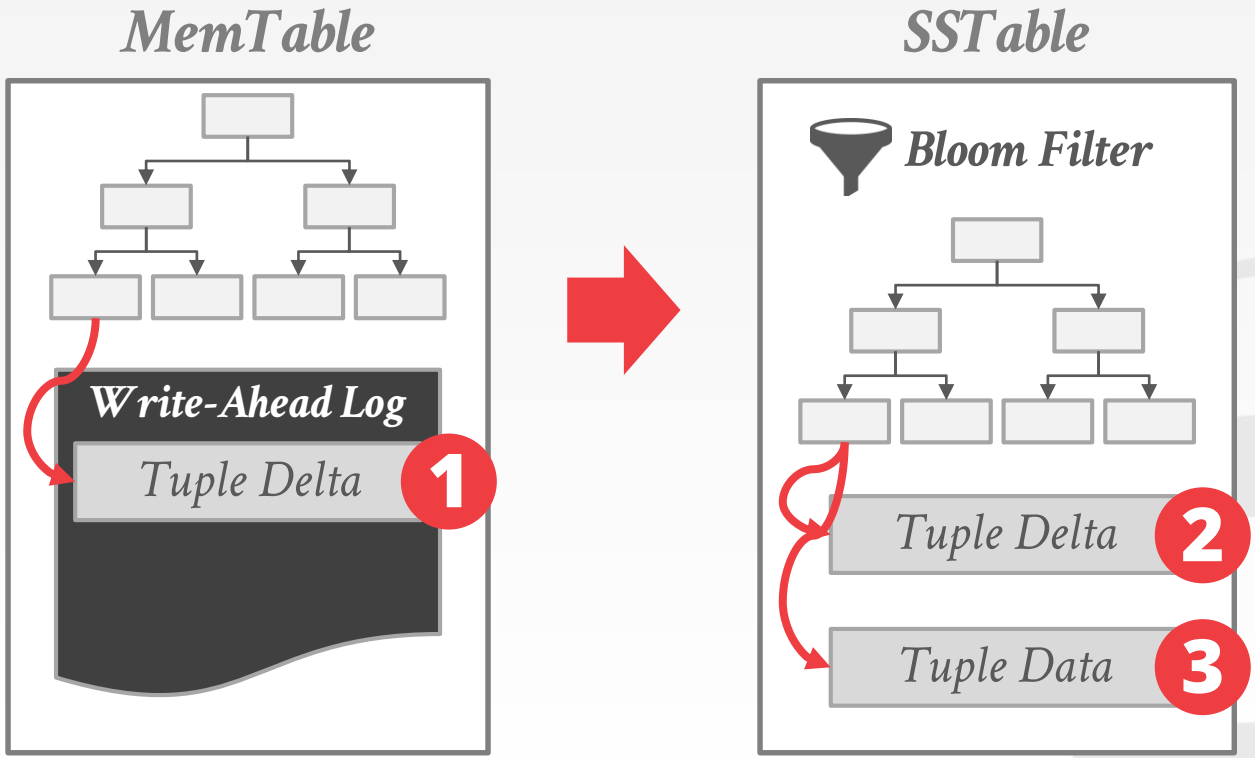
LOG-STRUCTURED ENGINE



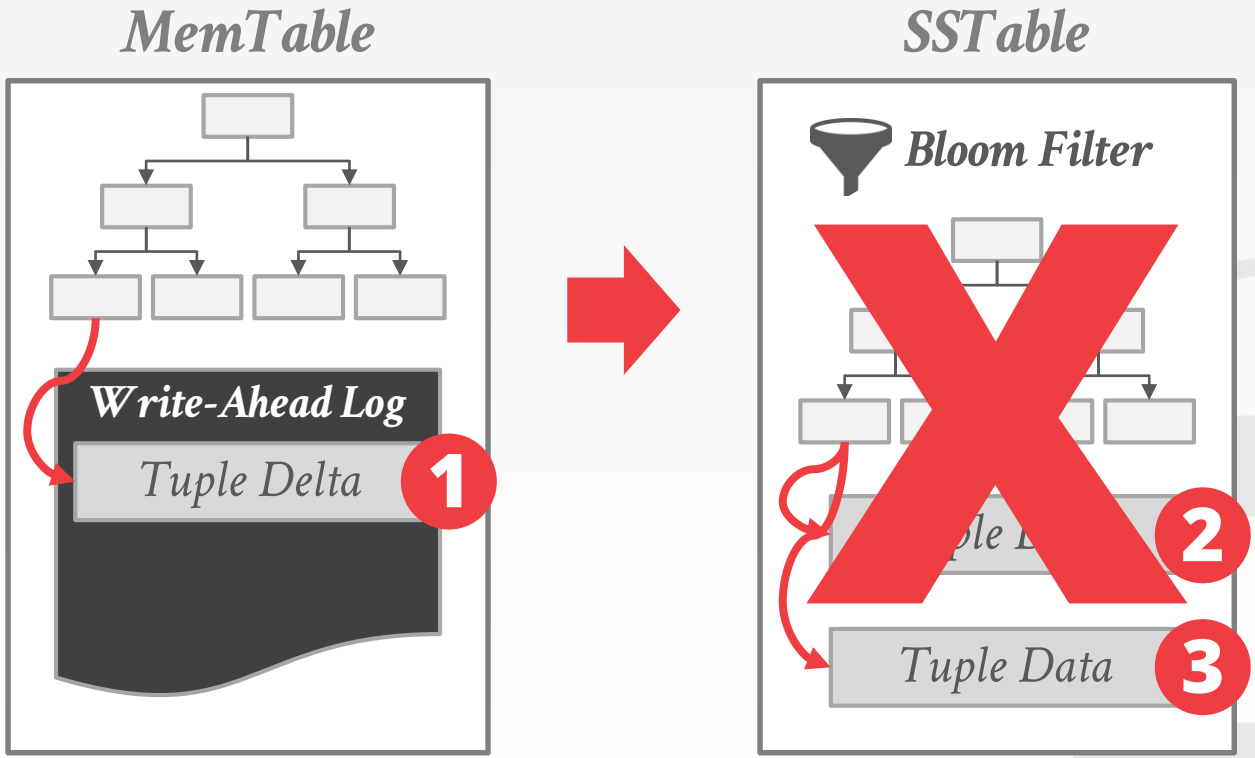
LOG-STRUCTURED ENGINE



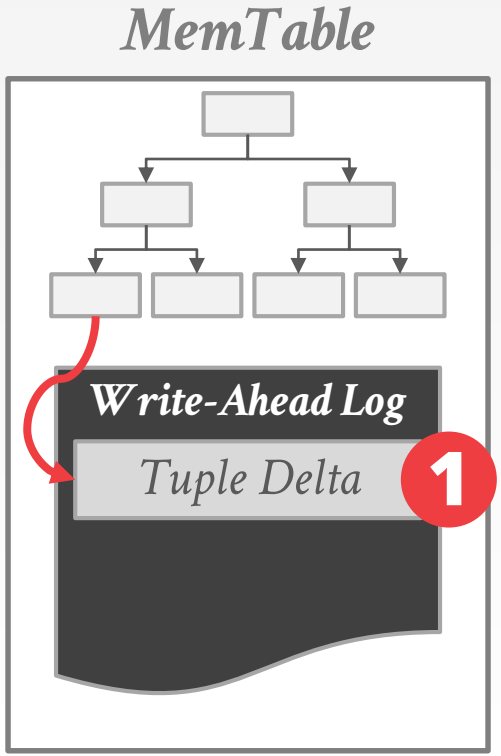
NVM LOG-STRUCTURED ENGINE



NVM LOG-STRUCTURED ENGINE



NVM LOG-STRUCTURED ENGINE



NVM SUMMARY

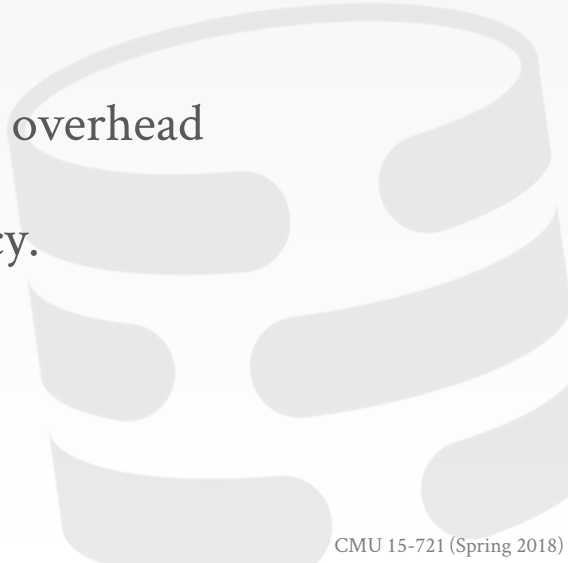
Storage Optimizations

→ 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.



GPU ACCELERATION

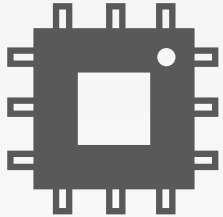
GPUs excel at performing (relatively simple) repetitive operations on large amounts of data over multiple streams of data.

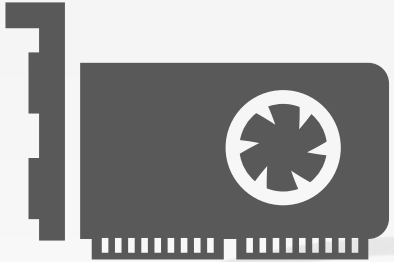
Target operations that do not require blocking for input or branches:

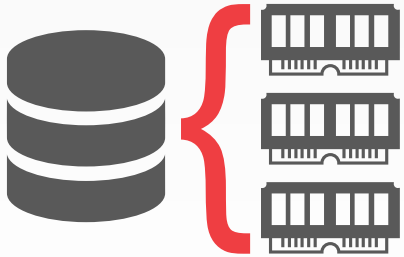
→ Good: Sequential scans with predicates
→ Bad: B+Tree index probes

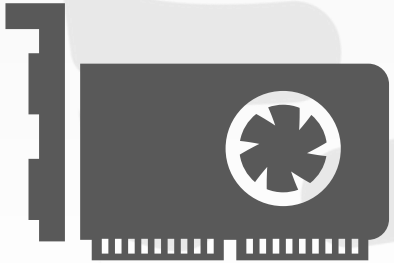
GPU memory is (usually) not cache coherent with CPU memory.

GPU ACCELERATION

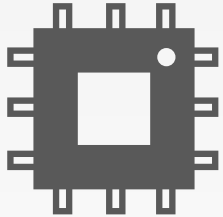


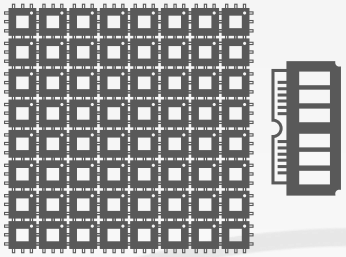


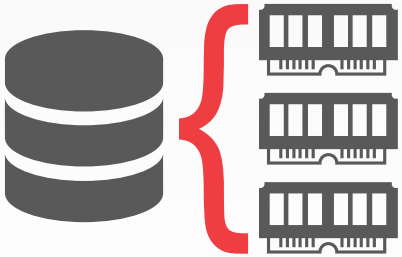


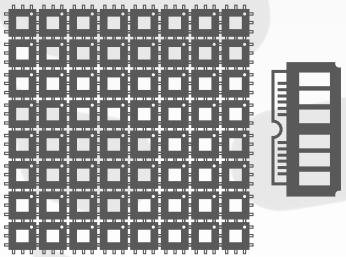


GPU ACCELERATION

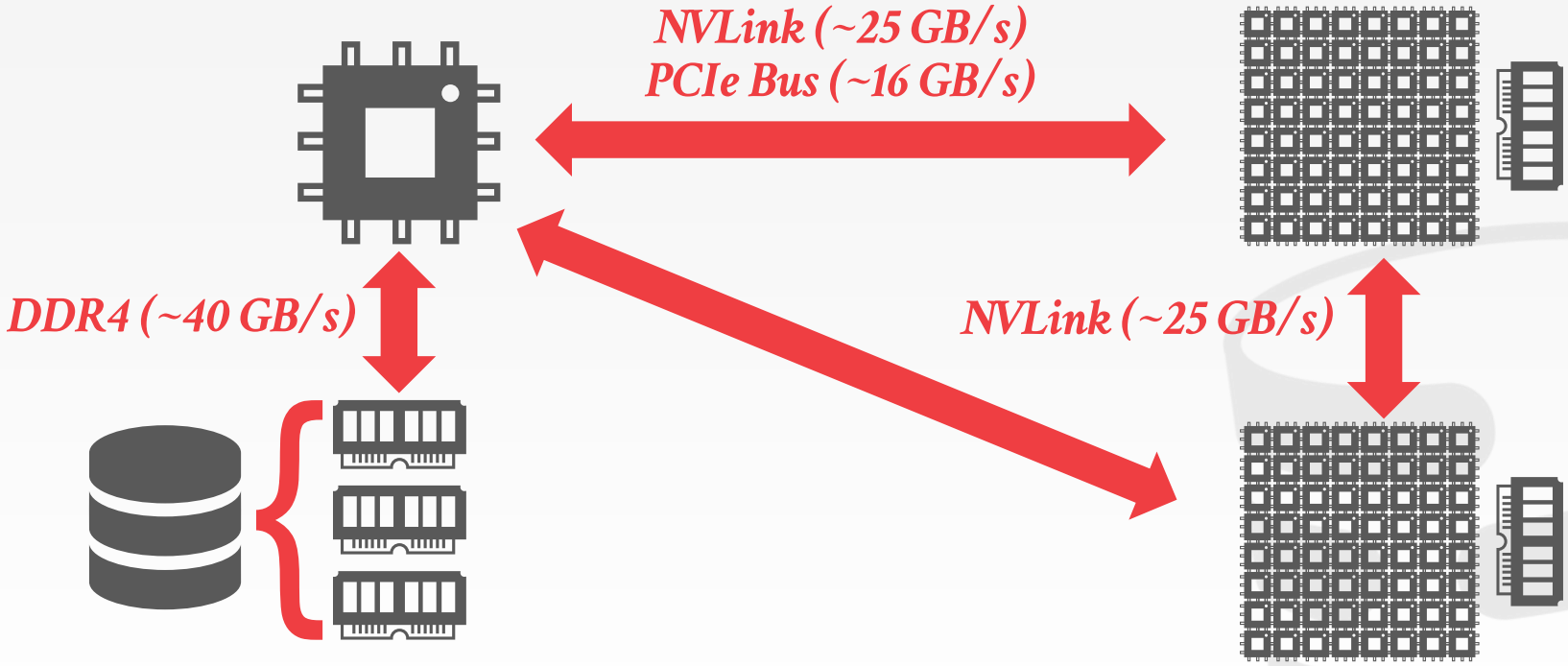








GPU ACCELERATION



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.
→ Have to materialize full results in CPU.

Choice #3: Streaming

→ Transfer data from CPU to GPU on the fly.

MAPD
kinetica
BLAZINGDB
SQREAM
brytlyt
blazegraph™

Carnegie Mellon University

HARDWARE ACCELERATED DATABASE LECTURES

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

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

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.

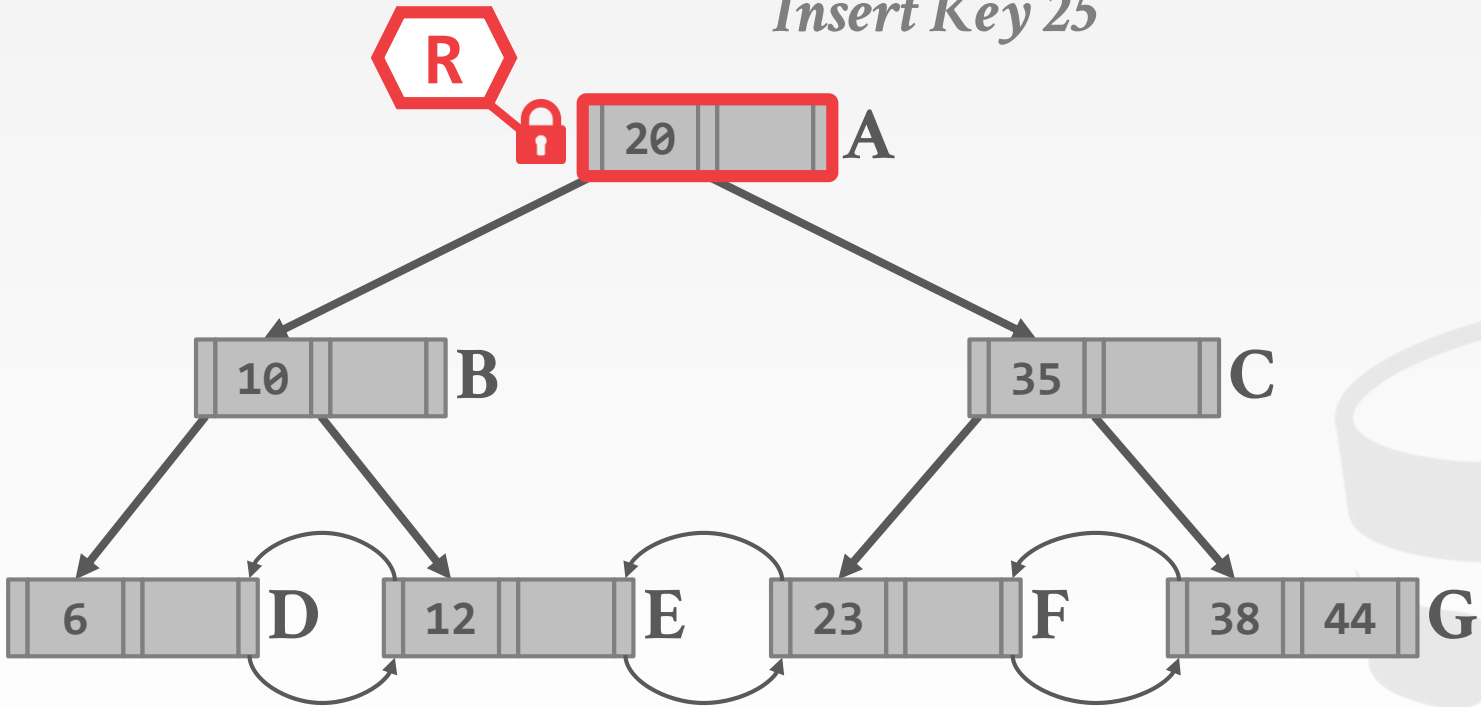
→ If there is a conflict, re-execute the code but actually take locks the second time.

Restricted Transactional Memory (RTM)

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

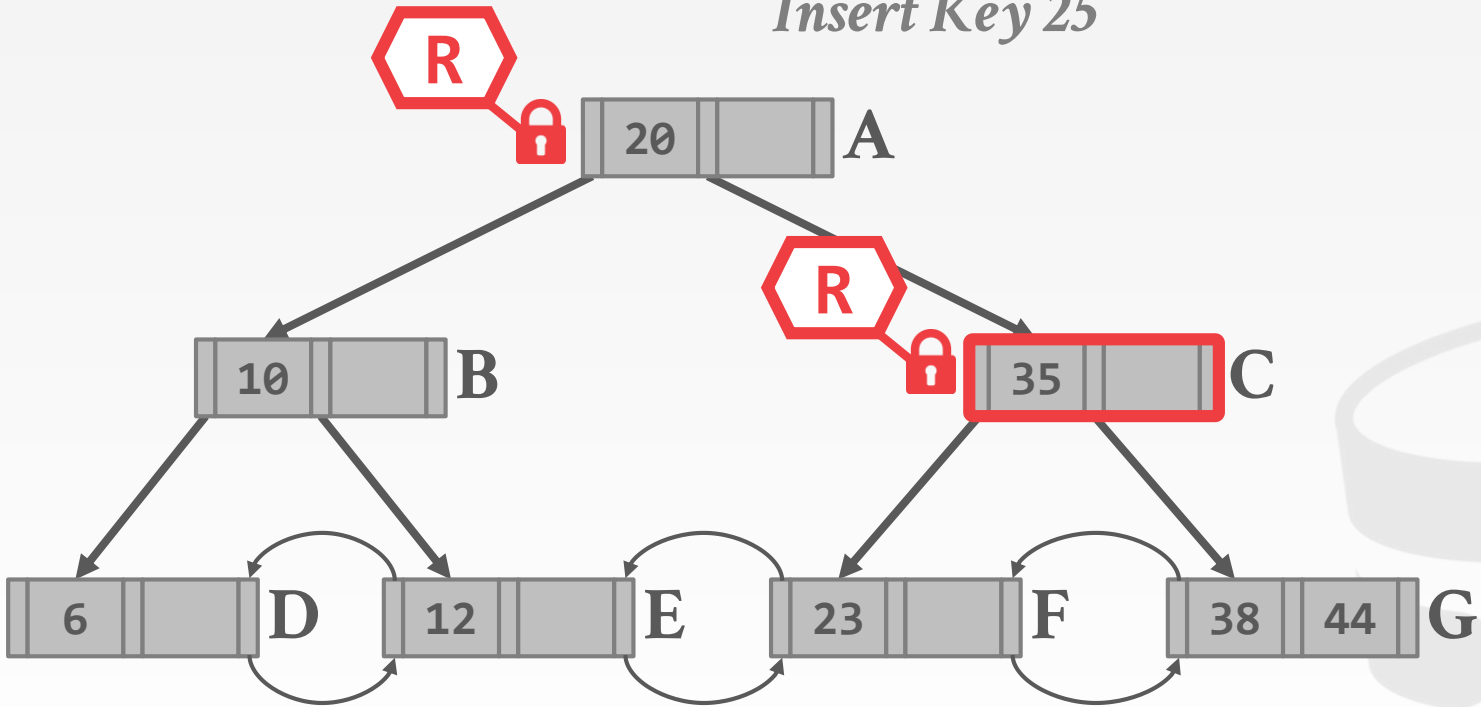
HTM LATCH ELISION

Insert Key 25



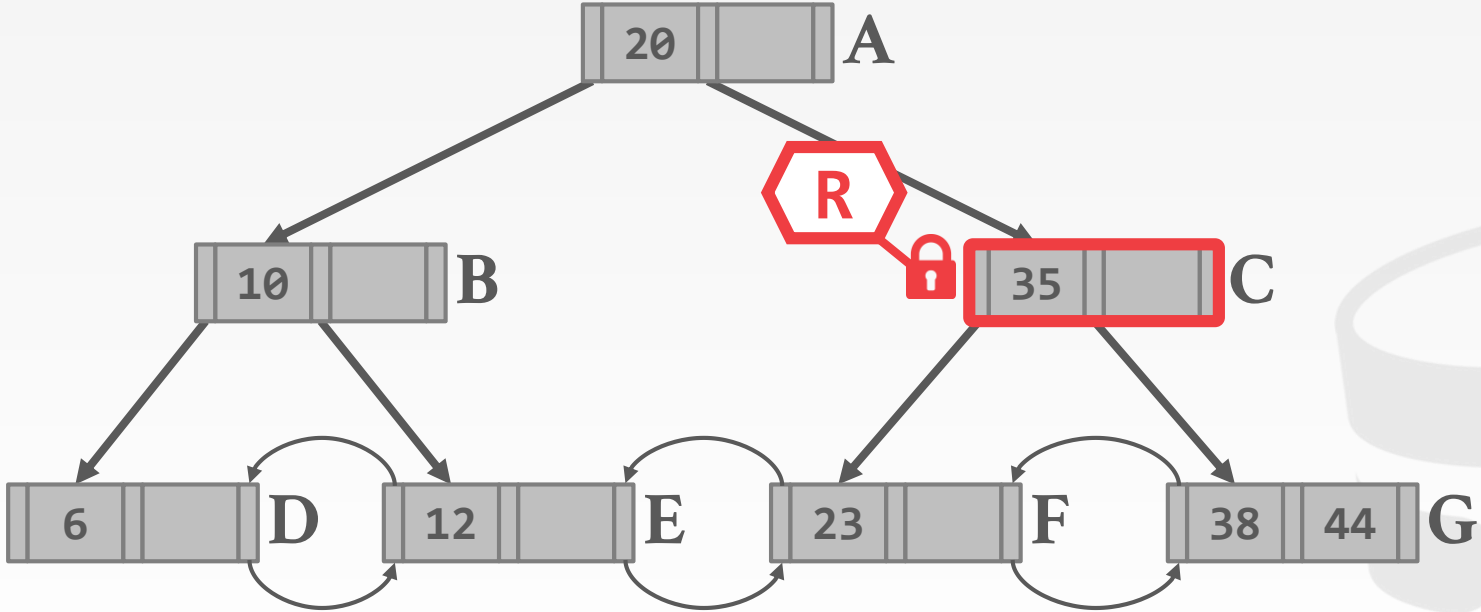
HTM LATCH ELISION

Insert Key 25



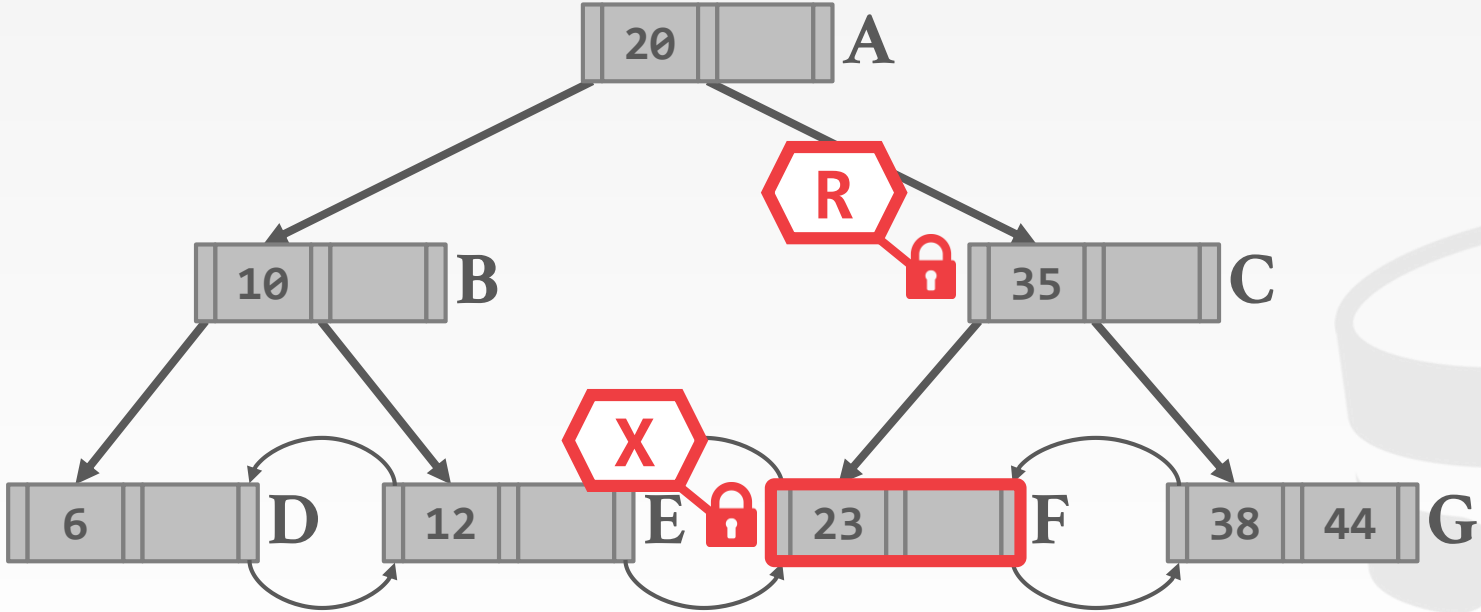
HTM LATCH ELISION

Insert Key 25



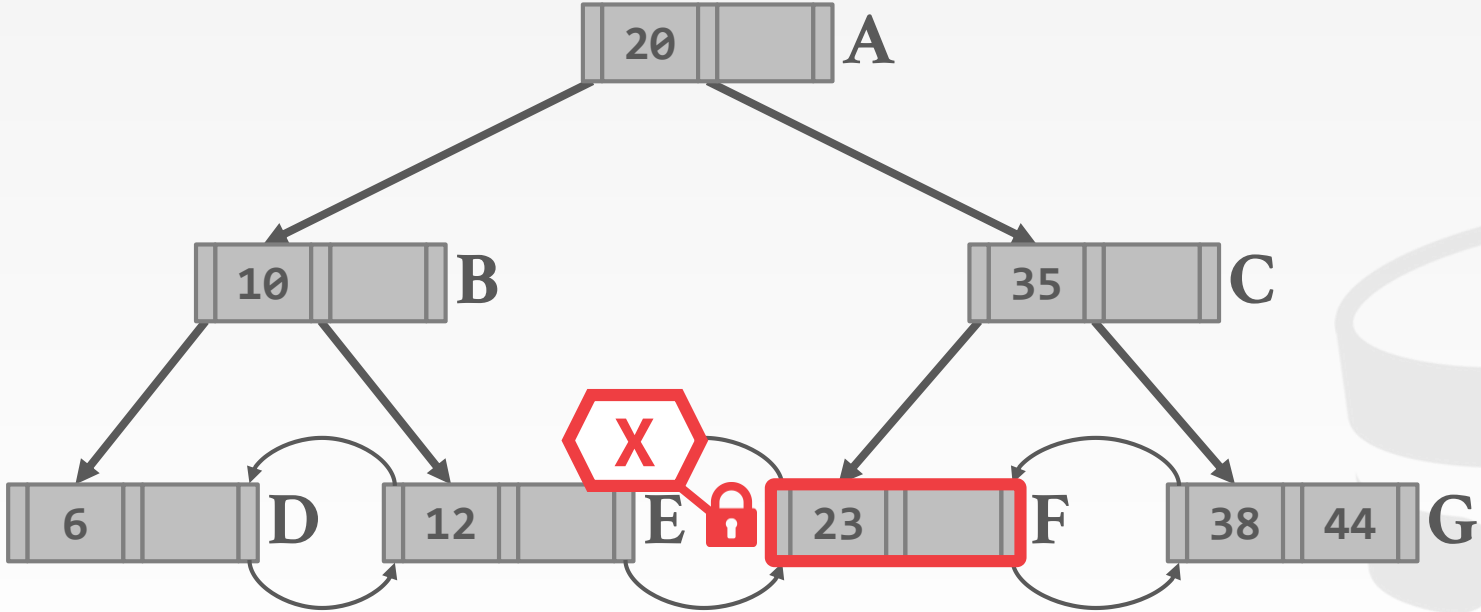
HTM LATCH ELISION

Insert Key 25



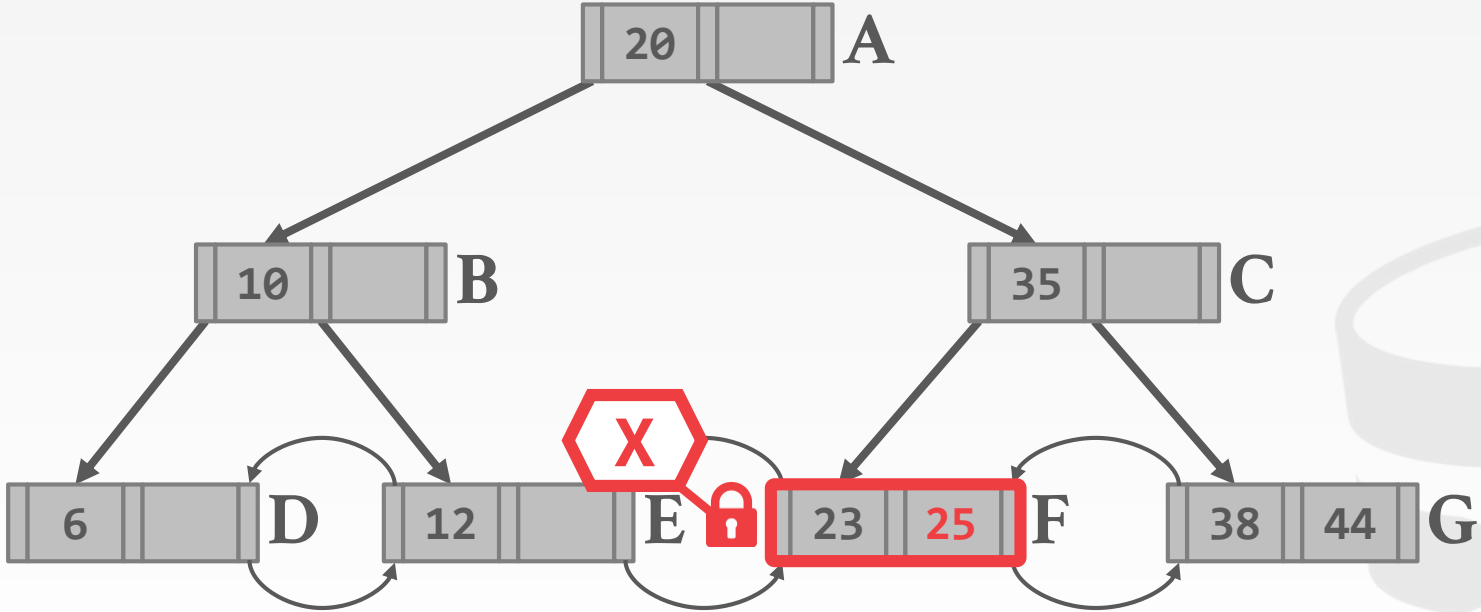
HTM LATCH ELISION

Insert Key 25



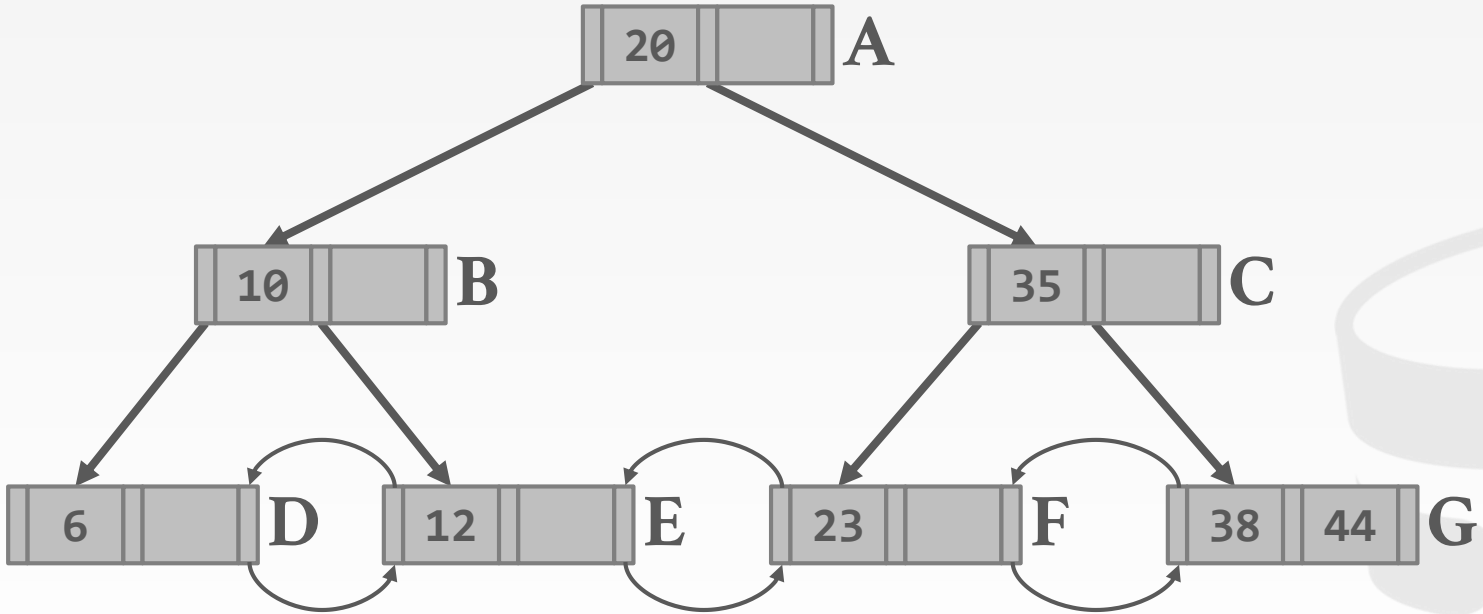
HTM LATCH ELISION

Insert Key 25



HTM LATCH ELISION

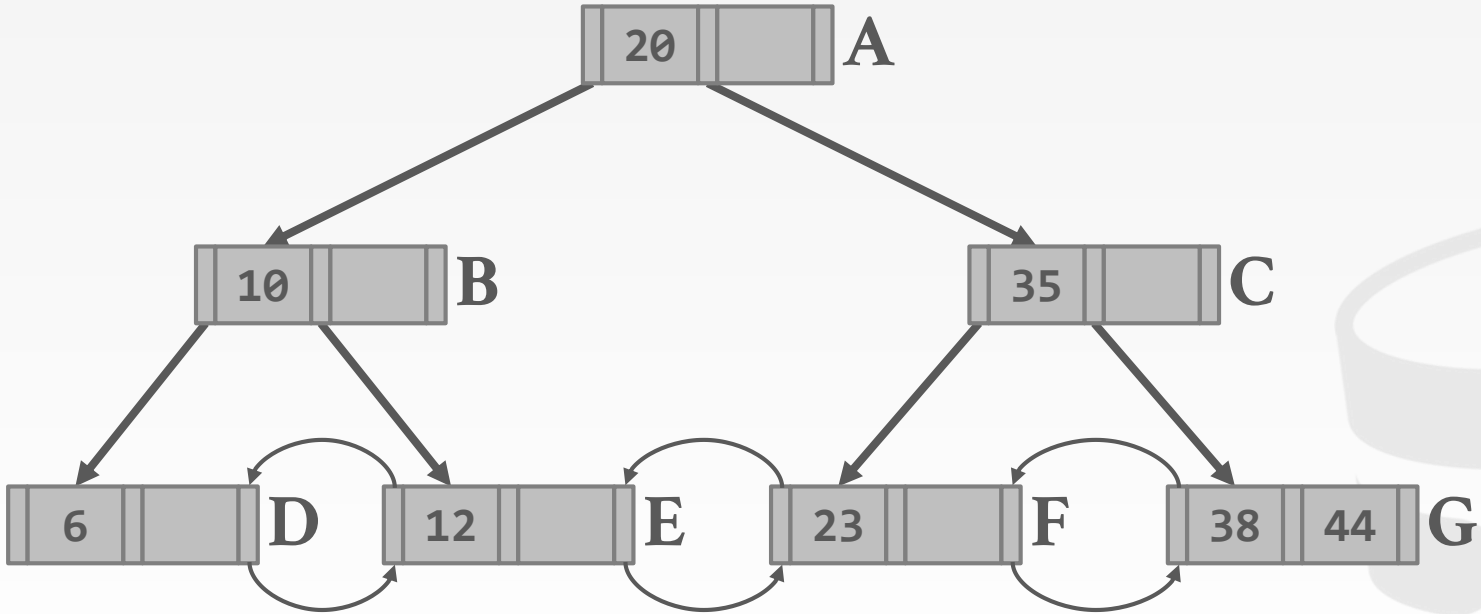
Insert Key 25




```
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

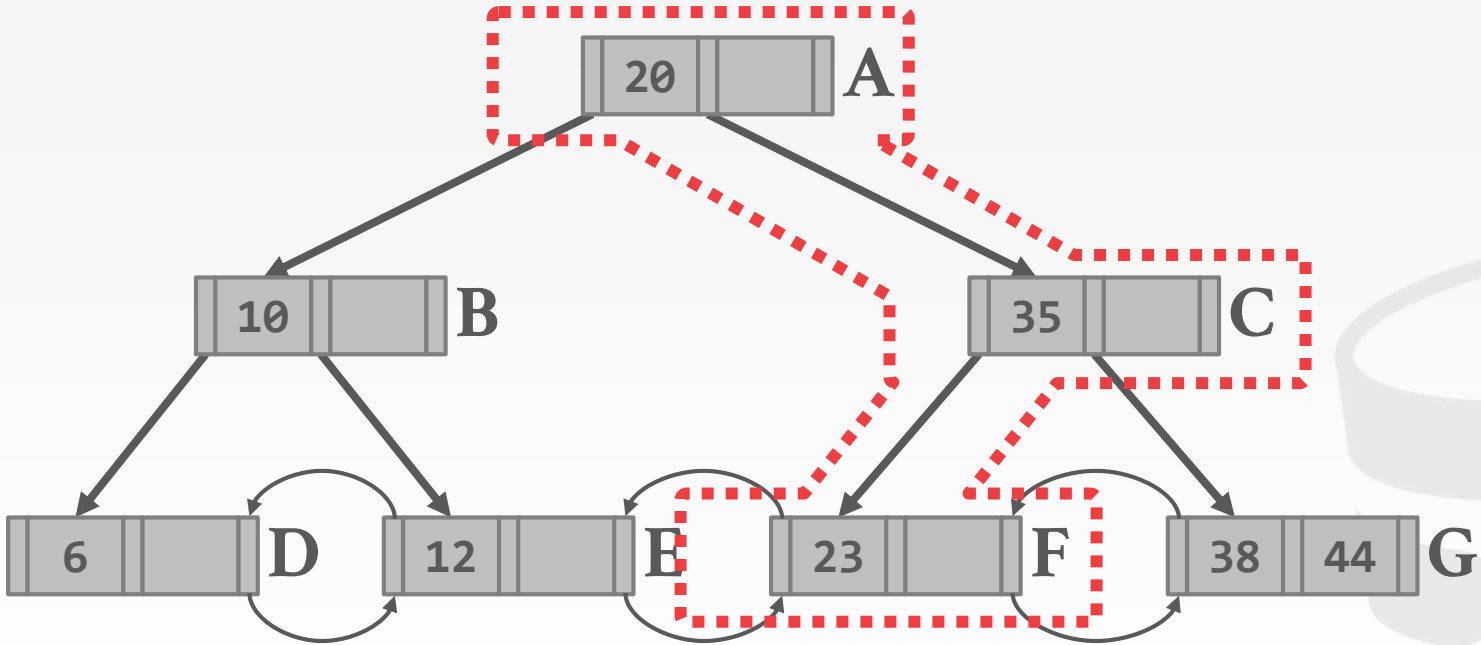
Insert Key 25




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

HTM LATCH ELISION

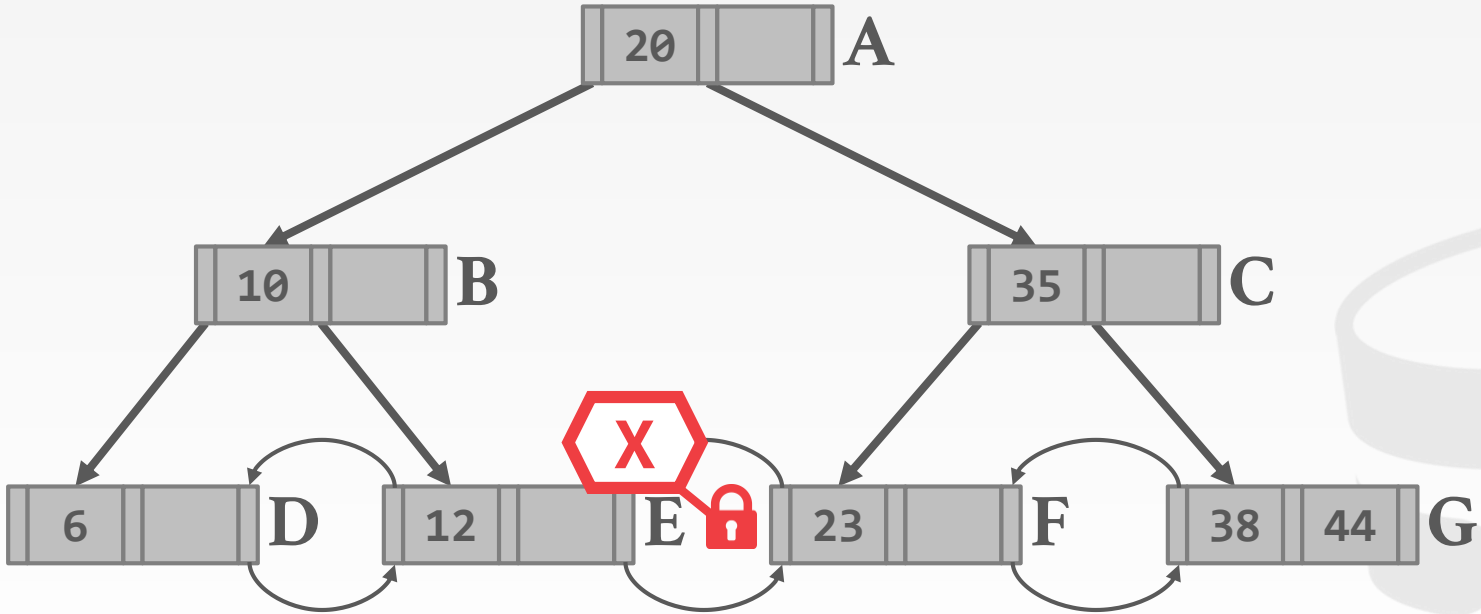
Insert Key 25




```
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

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.

NEXT CLASS

Final Exam Handout