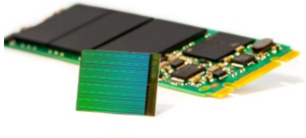
HOW TO BUILD A NON-VOLATILE MEMORY DATABASE SYSTEM

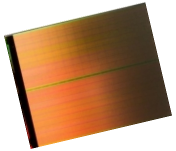
JOY ARULRAJ & ANDY PAVLO
CARNEGIE MELLON UNIVERSITY

SIGMOD 2017, CHICAGO

NON-VOLATILE MEMORY (NVM)



DRAM



NVM

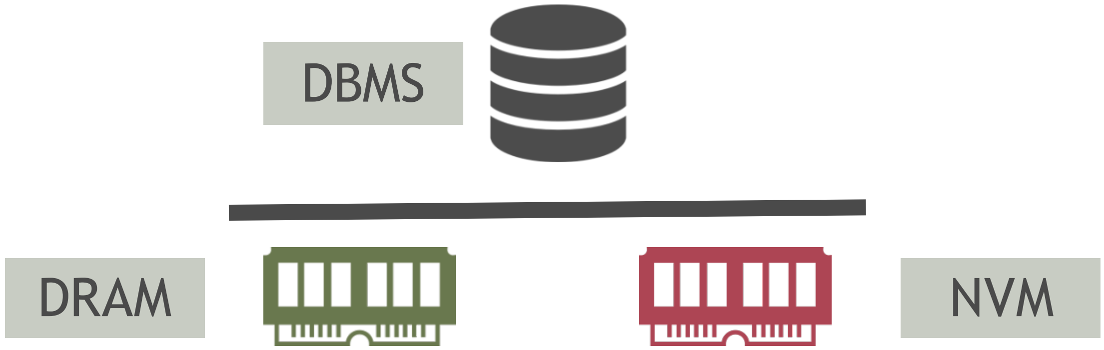
SSD

Like DRAM, low latency loads and stores

Like SSD, persistent writes and high density

TUTORIAL OVERVIEW

- Blueprint of an NVM DBMS
  - Overview of major design decisions impacted by NVM



TUTORIAL OVERVIEW

- Target audience
  - Developers, researchers, and practitioners
- Assume knowledge of DBMS internals
  - No need for any in-depth experience with NVM

TUTORIAL OVERVIEW

- Highlight recent research findings
  - Identify a set of open problems

TECHNISCHE UNIVERSITÄT DRESDEN

SAP HANA

UC San Diego

UNIVERSITY OF TORONTO

THE UNIVERSITY OF EDINBURGH

TECHNISCHE UNIVERSITÄT MÜNCHEN

THE UNIVERSITY OF MICHIGAN 1817

intel

CARNEGIE MELLON DATABASE GROUP

OUTLINE

- Introduction
  - Recent Developments
  - NVM Overview
  - Motivation
- Blueprint of an NVM DBMS
  - Access Interfaces
  - Storage Manager
  - Execution Engine
- Conclusion
  - Outlook

RECENT DEVELOPMENTS

#1: INDUSTRY STANDARDS

- Form factors (e.g., JEDEC classification)
  - **NVDIMM-F:** Flash only. Has to be paired with DRAM DIMM.
  - **NVDIMM-N:** Flash and DRAM together on the same DIMM.
  - **NVDIMM-P:** True persistent memory. No DRAM or flash.
- Interface specifications (e.g., NVM Express over Fabrics)

JUNE 2016

#2: OPERATING-SYSTEM SUPPORT

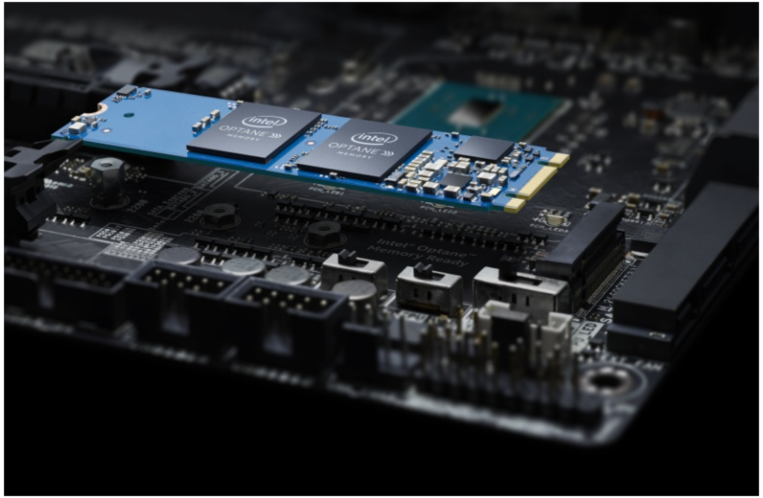
- Growing OS support for NVM
  - **Linux 4.8** (e.g. NVM Express over Fabrics library)
  - **Windows 10** (e.g. Direct access to files on NVM)

OCTOBER 2016

#3: PROCESSOR SUPPORT

- ISA updates in Kaby Lake Processor for NVM management
  - Instructions for flushing cache-lines to NVM
  - Removed PCOMMIT instruction

MARCH 2017



NVM OVERVIEW

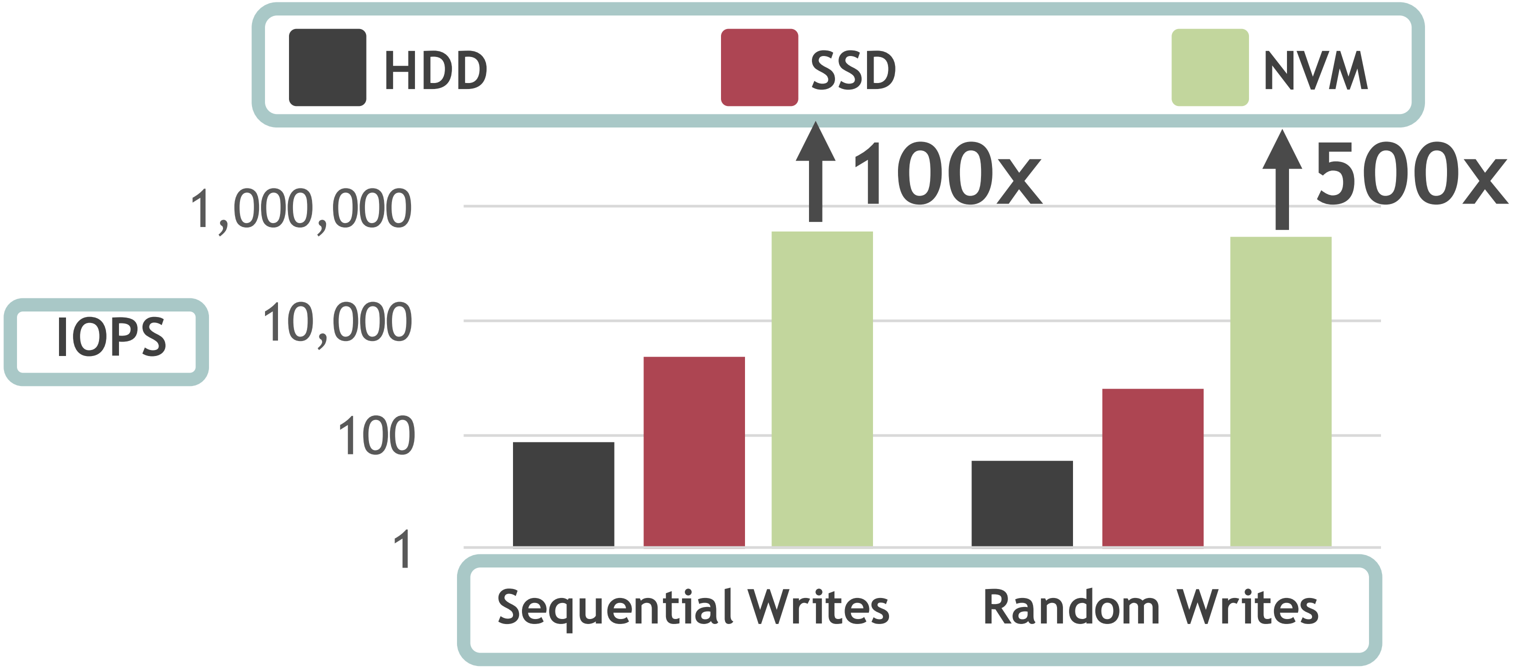
NVM PROPERTIES

1. Byte addressable
   - Loads and stores unlike SSD/HDD
2. High random write throughput
   - Orders of magnitude higher than SSD/HDD
   - Smaller gap between sequential & random write throughput
3. Read-write asymmetry & wear-leveling
   - Writes might take longer to complete compared to reads
   - Excessive writes to a single NVM cell can destroy it

EVALUATION SETUP

- Benchmark storage devices on NVM emulator
  - Write throughput of a single thread with fio
  - Synchronous writes to a large file
- Devices
  - Hard-disk drive (HDD) [Seagate Barracuda]
  - Solid-state disk (SSD) [Intel DC S3700]
  - Emulated NVM

PERFORMANCE



MOTIVATION

EXISTING DBMSs ON NVM

- How do existing systems perform on NVM?
  - Treat NVM like a faster SSD
- Evaluate two types of database systems
  - Disk-oriented DBMS
  - Memory-oriented DBMS
- TPC-C benchmark
  - 1/8th of database fits in DRAM
  - Rest on NVM

A PROLEGOMENON ON OLTP DATABASE SYSTEMS
FOR NON-VOLATILE MEMORY
ADMS 2014

EXISTING DBMSs

- Compare representative DBMSs of each architecture

MySQL™

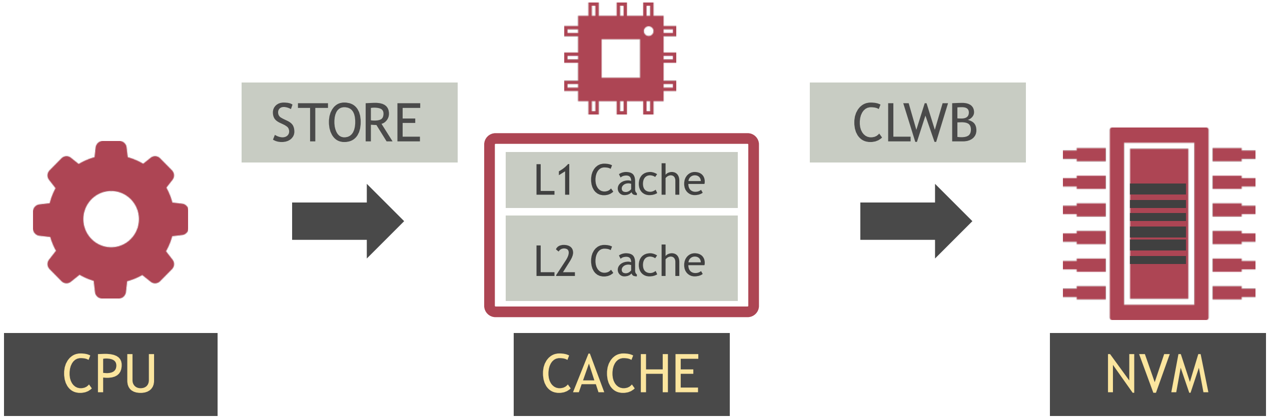
H-Store

DISK-ORIENTED DBMS

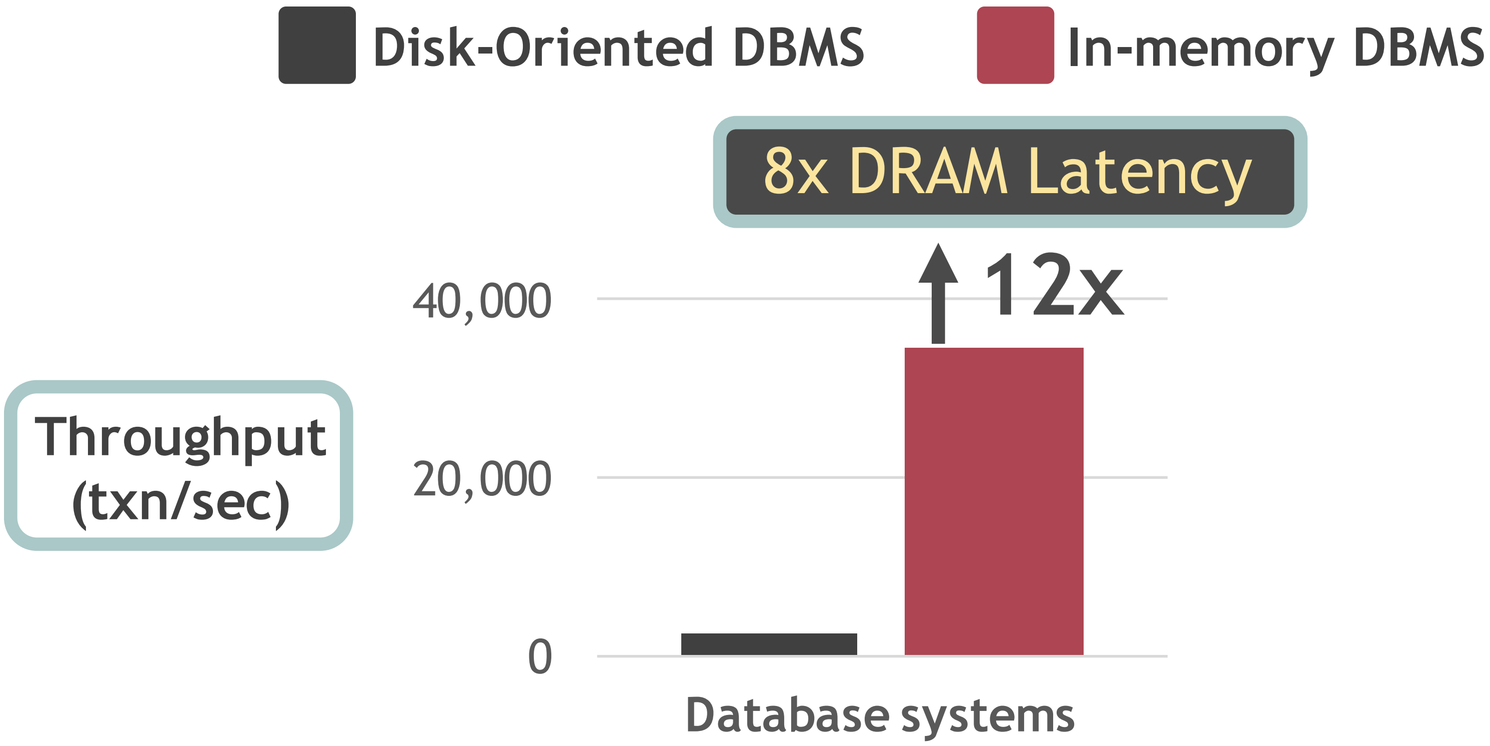
MEMORY-ORIENTED DBMS

NVM HARDWARE EMULATOR

- Special CPU microcode to add stalls on cache misses
  - Tune DRAM latency to emulate NVM
- New instructions for managing NVM
  - Cache-line write-back (CLWB) instruction



PERFORMANCE

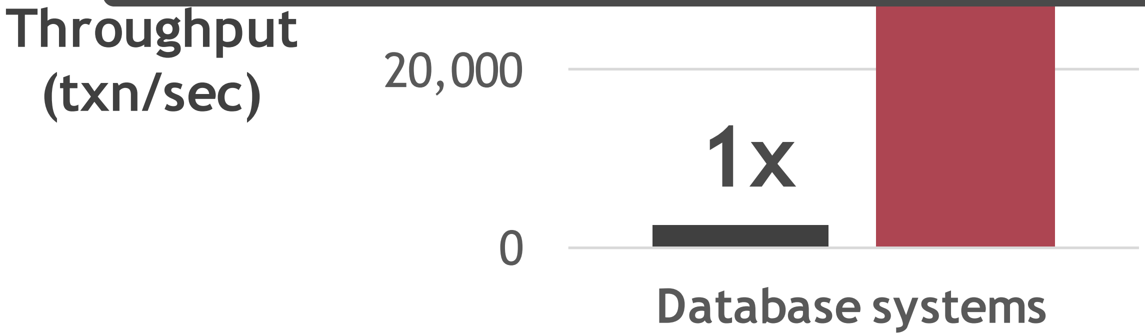


PERFORMANCE

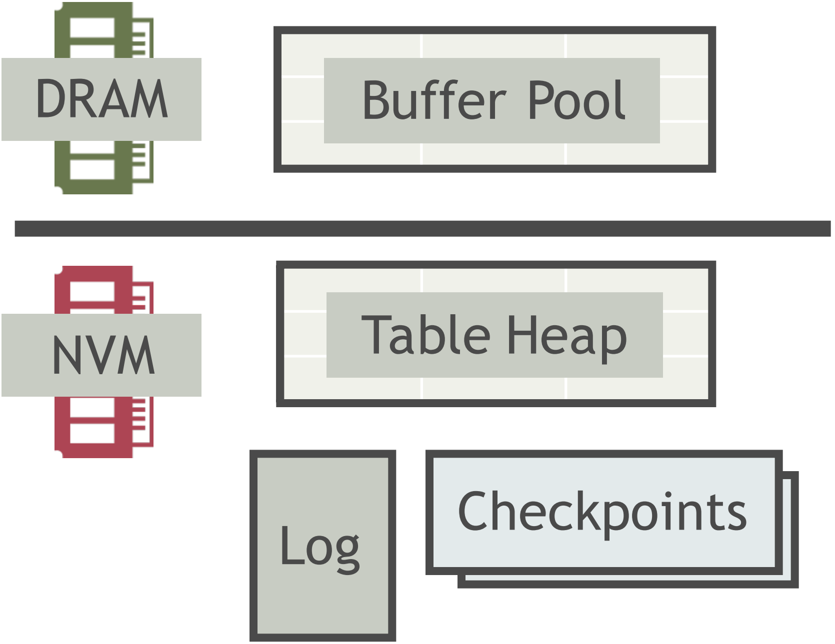
Disk-Oriented DBMS   In-memory DBMS

2x DRAM Latency ↓4x

Legacy database systems are not prepared for NVM



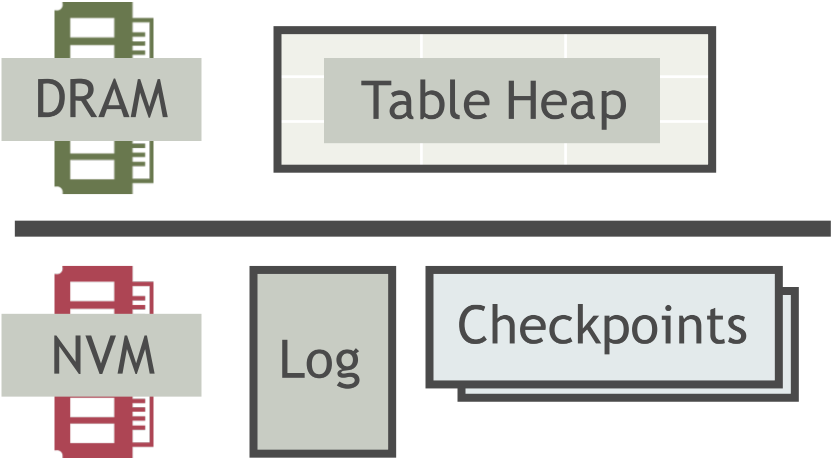
#1: DISK-ORIENTED DBMSs



Designed to minimize random writes to NVM

But, NVM supports fast random writes

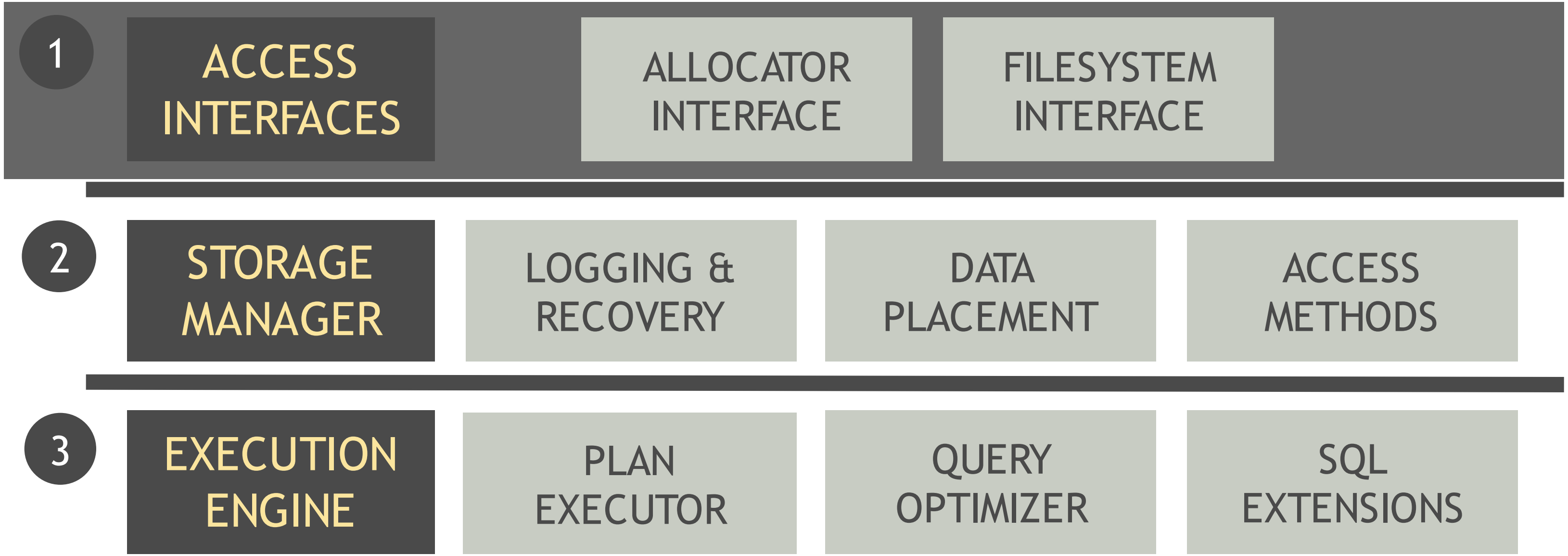
#2: MEMORY-ORIENTED DBMSs



Designed to overcome the volatility of memory

But, writes to NVM are persistent

BLUEPRINT OF AN NVM DBMS



HOW TO BUILD A NON-VOLATILE MEMORY DBMS
SIGMOD 2017 (TUTORIAL)

ACCESS INTERFACES

ACCESS INTERFACES

- Two interfaces used by the DBMS to access NVM
  - Allocator interface (byte-level memory allocation)
  - Filesystem interface (POSIX-compliant filesystem API)
- Support in latest versions of major operating systems
  - Windows Server 2016
  - Linux 4.7

#1: ALLOCATOR INTERFACE

- Similar to regular DRAM allocator
  - Dynamic memory allocation
  - Meta-data management
- Additional features with respect to DRAM allocator
  - Durability mechanism
  - Naming mechanism
  - Recovery mechanism

DURABILITY MECHANISM

- Ensure that data modifications are persisted
  - Necessary because they may reside in volatile processor caches
  - Lost if a power failure happens before changes reach NVM

```
Persist(Address, Length)
```

- Two-step implementation
  - Allocator first writes out dirty cache-lines (CLWB)
  - Issues a memory fence (SFENCE) to ensure changes are visible

NAMING MECHANISM

- Pointers should be valid even after the system restarts
  - NVM region might be mapped to a different base address

Absolute pointer = Base address + Relative pointer

- Allocator maps NVM to a well-defined base address
  - Pointers, therefore, remain valid even after system restart
  - Foundation for building crash-consistent data structures

RECOVERY MECHANISM

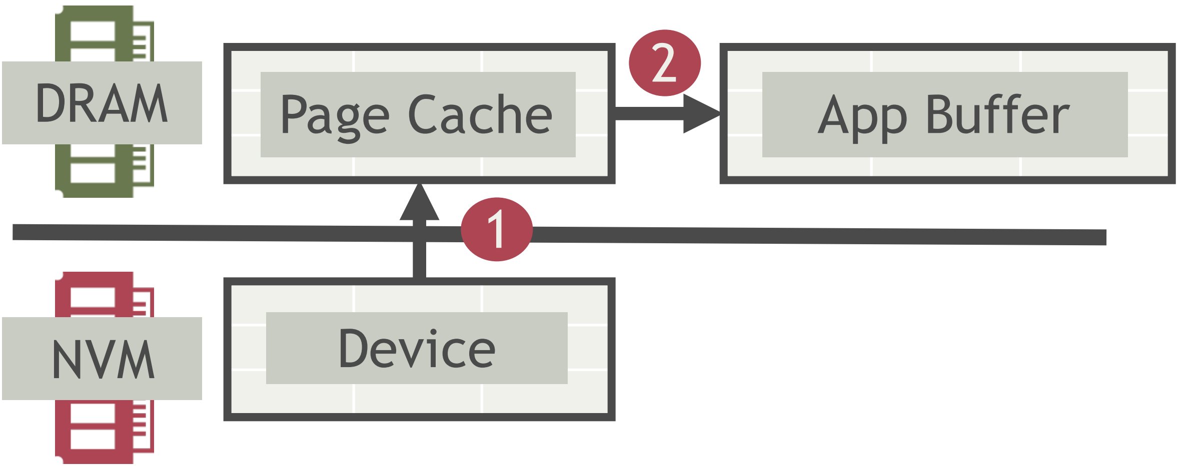
- Unlike DRAM, persistent memory leaks with NVM
  - Let's say an application allocates a memory chunk
  - But crashes before linking the chunk to its data structure
  - Neither allocator nor application can reclaim the space
- Recovery ensures all chunks are either allocated or free
  - Interesting problem, will be covered in next tutorial

Data Structures Engineering For NVM

Ismail Oukid and Wolfgang Lehner, TU Dresden

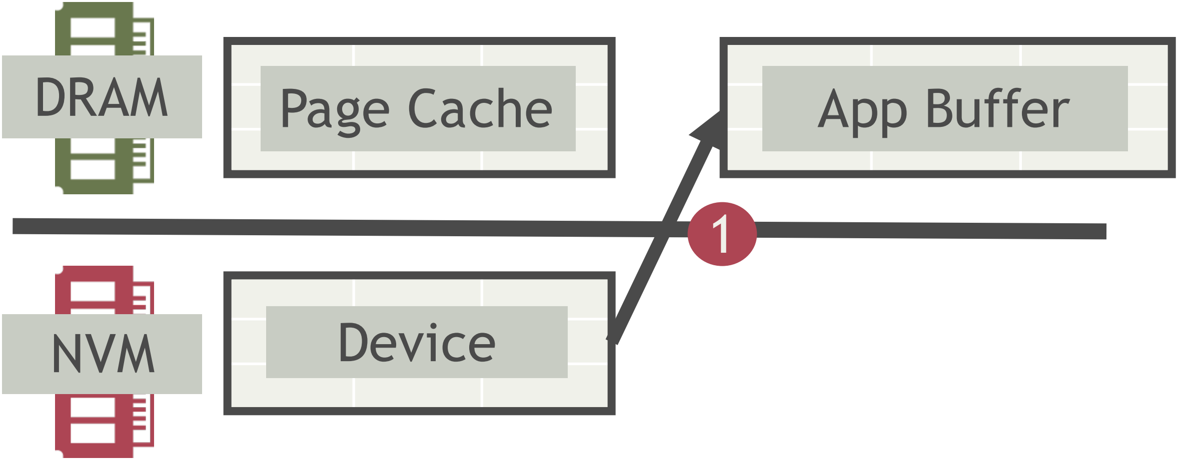
#2: FILESYSTEM INTERFACE

- Traditional block-based filesystem like EXT4
  - File I/O: 2 copies (Device ⟶ Page Cache ⟶ App Buffer)
  - Efficiency of I/O stack not critical when hidden by disk latency
  - However, NVM is byte-addressable and supports very fast I/O



NON-VOLATILE MEMORY FILESYSTEM

- Direct access storage (DAX) to avoid data duplication
  - DBMS can directly manage database by skipping page cache
  - File I/O: 1 copy (Device ⟶ App Buffer)



SYSTEM SOFTWARE FOR PERSISTENT MEMORY
EUROSYS 2014

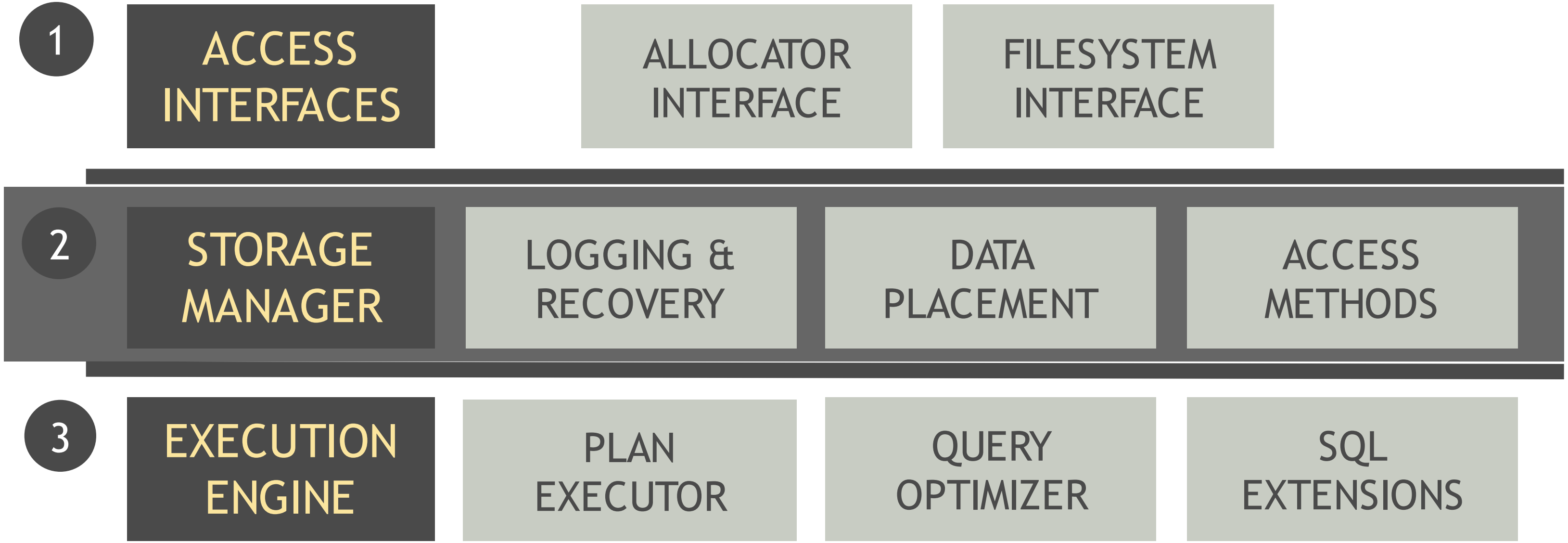
NON-VOLATILE MEMORY FILESYSTEM

- To ensure durability, uses a hybrid recovery protocol
  - NVM only supports 64-byte (cacheline) atomic updates
  - DATA CHANGES: Copy-on-write mechanism at page granularity
  - METADATA CHANGES: In-place updates & write-ahead logging
- NVM filesystem
  - Reduces data duplication
  - Uses lightweight recovery protocol
  - 10x more IOPS compared to EXT4

RECAP: ACCESS INTERFACES

- Allocator interface
  - Non-volatile data structures
  - Table heap, Indexes
- Filesystem interface
  - Log files, Checkpoints

BLUEPRINT OF AN NVM DBMS



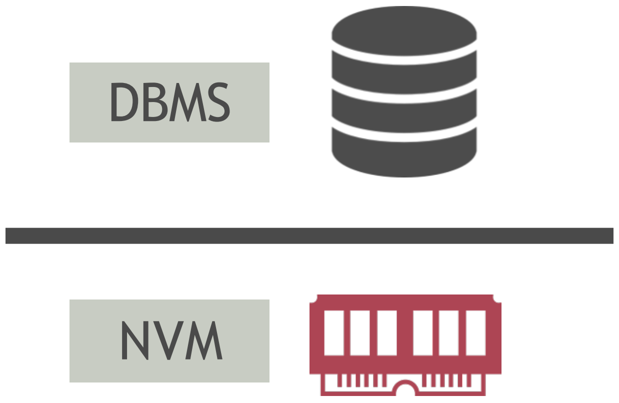
STORAGE MANAGER

MULTI-VERSIONED DBMS

TUPLE ID	BEGIN TIMESTAMP	END TIMESTAMP	PREVIOUS VERSION	TUPLE DATA
1	10	∞	—	X
2	10	20	—	Y

THOUGHT EXPERIMENT

- To keep things simple, NVM-only storage hierarchy
  - No volatile DRAM

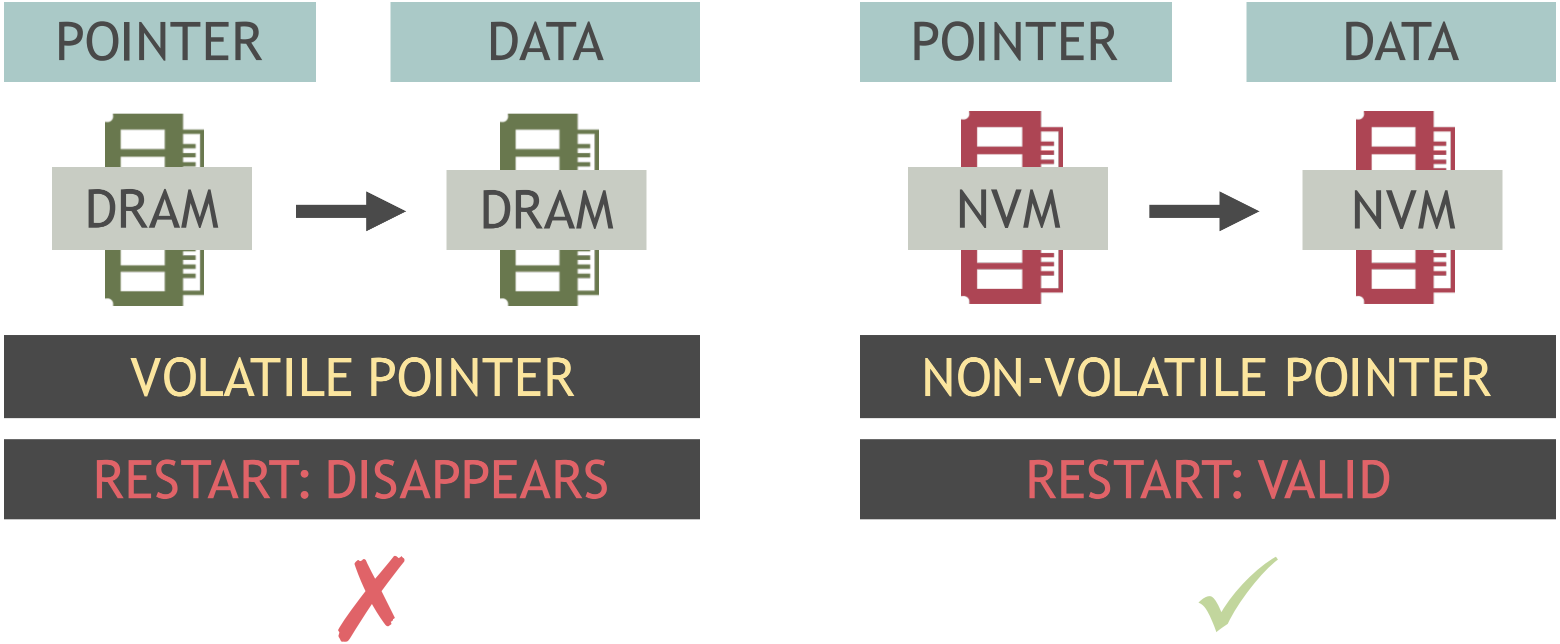


LOGGING AND RECOVERY

- Traditional write-ahead logging in off-the-shelf DBMS

Can we avoid duplicating data in the log as well as the checkpoints?

NON-VOLATILE POINTER



AVOIDING DATA DUPLICATION

- Only store non-volatile tuple pointers in log records

Table Heap

TUPLE ID	TUPLE DATA
100	XYZ
101	X'Y'Z'

Write-Ahead Log

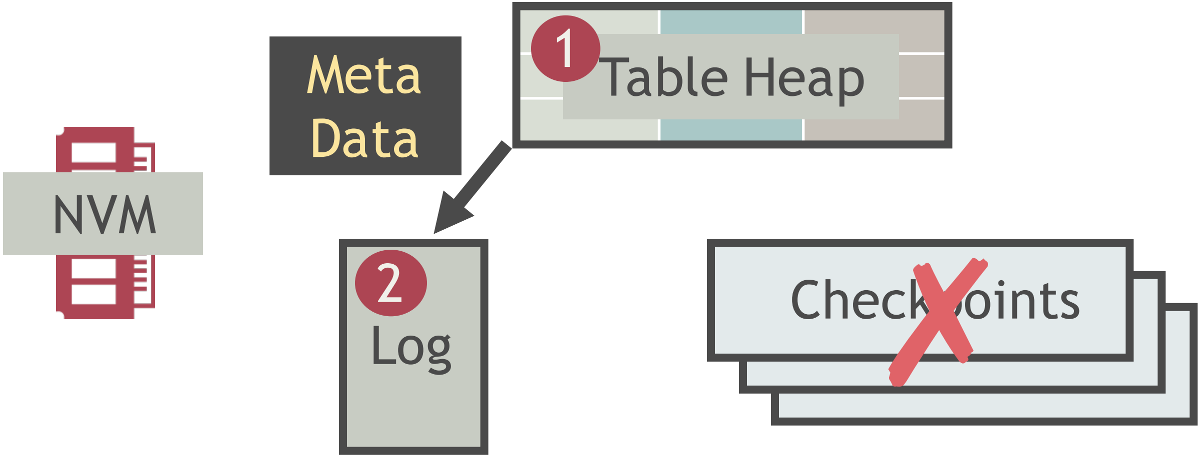
TRADITIONAL MANAGER
INSERT TUPLE XYZ
UPDATE TUPLE XYZ → X'Y'Z'

NVM-AWARE MANAGER
INSERT TUPLE 100
UPDATE TUPLE 100 → 101

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

NVM-AWARE STORAGE MANAGER

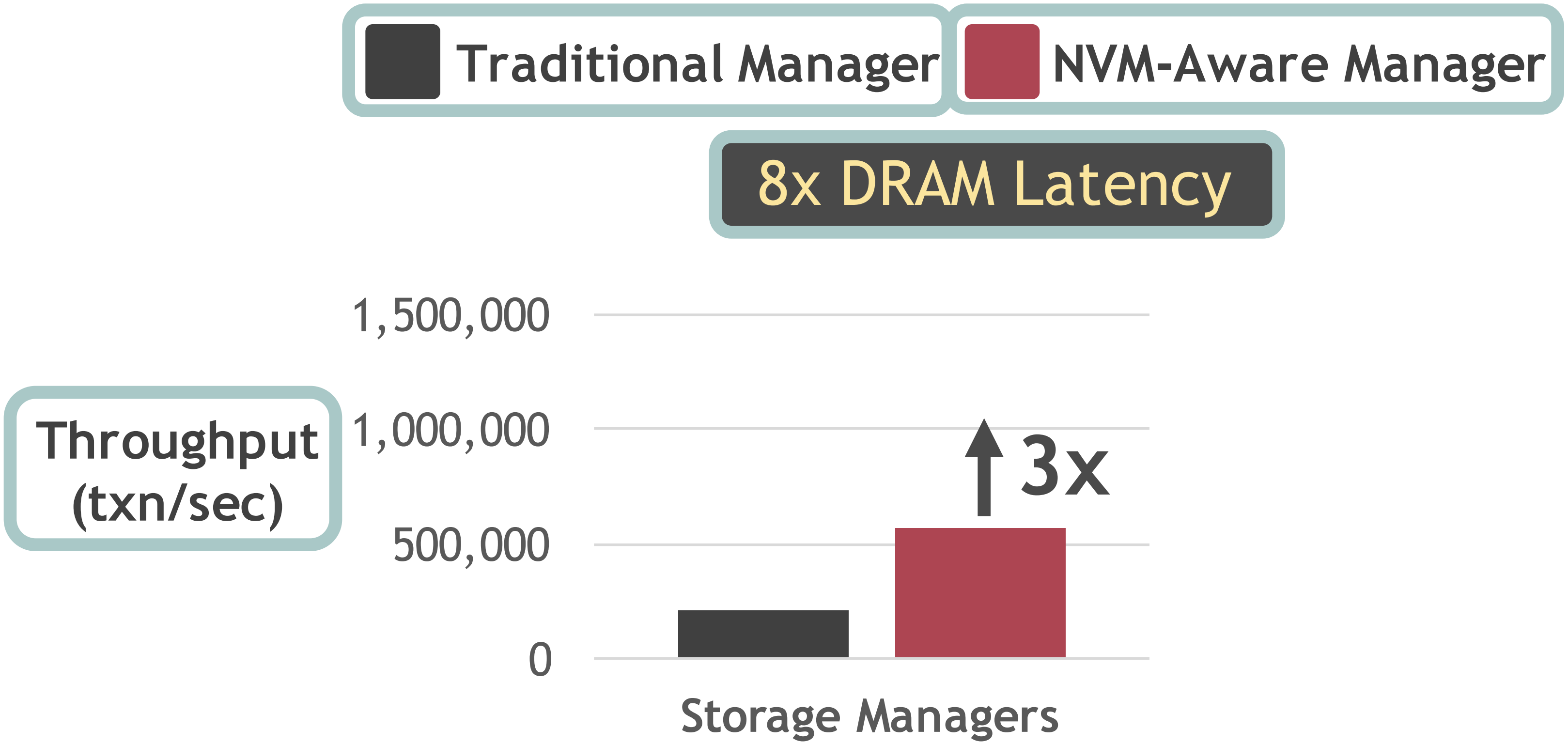
- Write-ahead meta-data logging



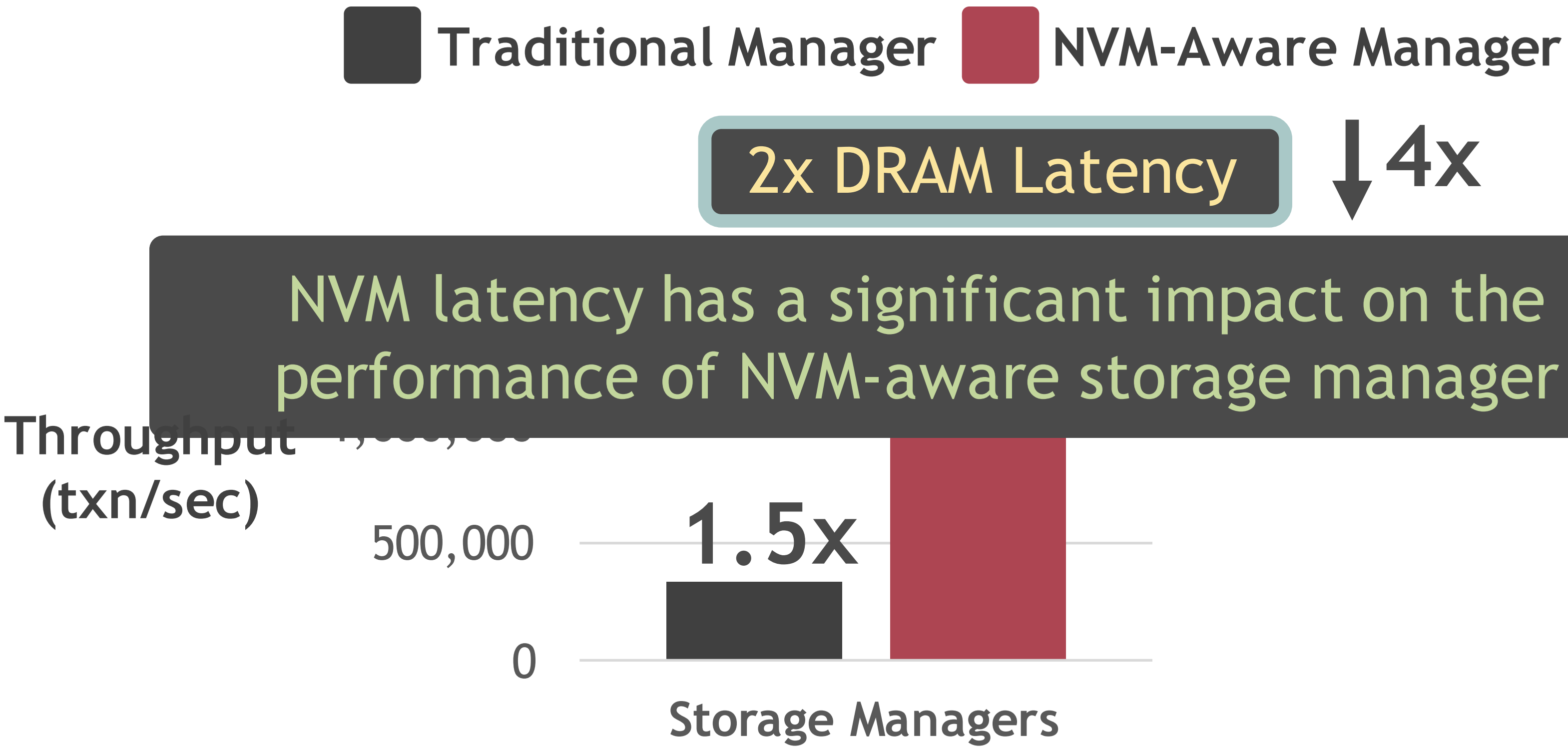
EVALUATION

- Compare storage managers on NVM emulator
  - Traditional storage manager
  - Write-ahead logging + Filesystem interface
  - NVM-aware storage manager
  - Write-ahead meta-data logging + Allocator interface
- Yahoo! Cloud Serving Benchmark
  - Database fits on NVM

RUNTIME PERFORMANCE



RUNTIME PERFORMANCE



NVM latency has a significant impact on the performance of NVM-aware storage manager

DEVICE LIFETIME

Redesigning the storage manager for NVM not only improves runtime performance but also extends device lifetime

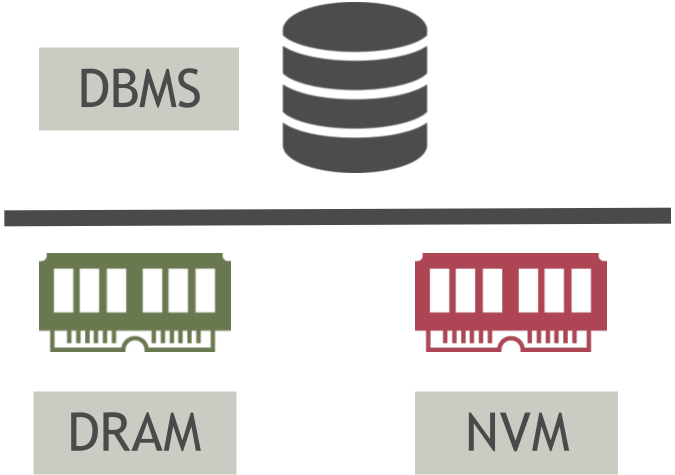
RECAP: WRITE-AHEAD METADATA LOGGING

- Targets an NVM-only storage hierarchy
  - Leverages the durability of memory
  - Skips duplicating data in the log and checkpoints
  - Improves runtime performance
  - Extends lifetime of the device

WRITE-BEHIND LOGGING

TWO-TIER STORAGE HIERARCHY

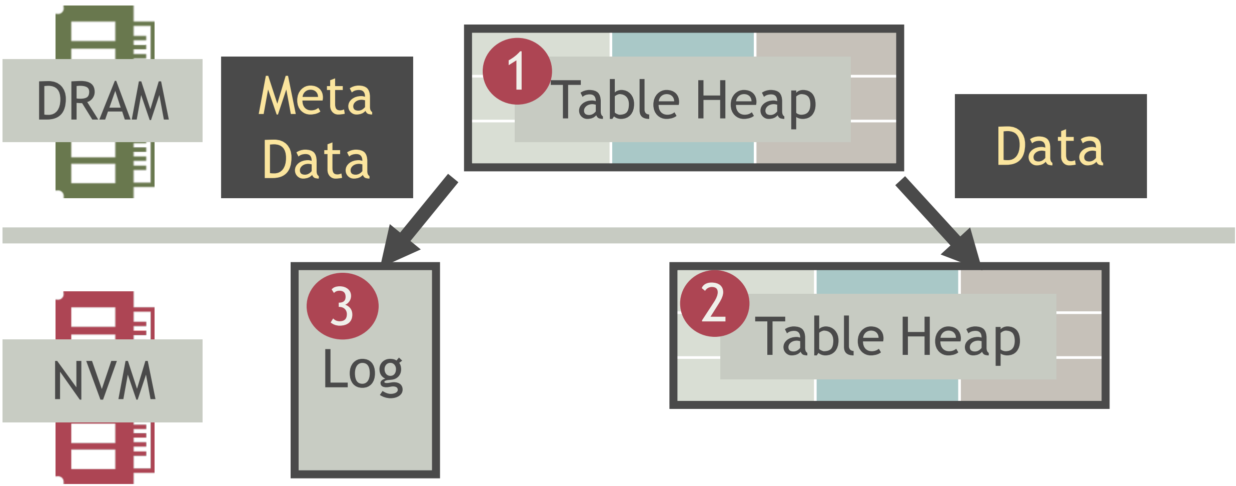
- Generalize the logging and recovery algorithms



WRITE-AHEAD LOGGING

- Write-ahead log serves two purposes
  - Transform random database writes into sequential log writes
  - Support transaction rollback
  - Design makes sense for disks with slow random writes
- But, NVM supports fast random writes
  - Directly write data to the multi-versioned database
  - LATER, only record meta-data about committed txns in log
  - Core idea behind write-behind logging

WRITE-BEHIND LOGGING



WRITE-BEHIND LOGGING

- Recovery algorithm is simple
  - No need to REDO the log, unlike write-ahead logging
  - Since all changes are already persisted in database at commit
  - Can recover the database almost instantaneously from failure
- Supporting transaction rollback
  - Need to record meta-data about in-flight transactions
  - In case of failure, ignore their effects

WRITE-BEHIND LOGGING

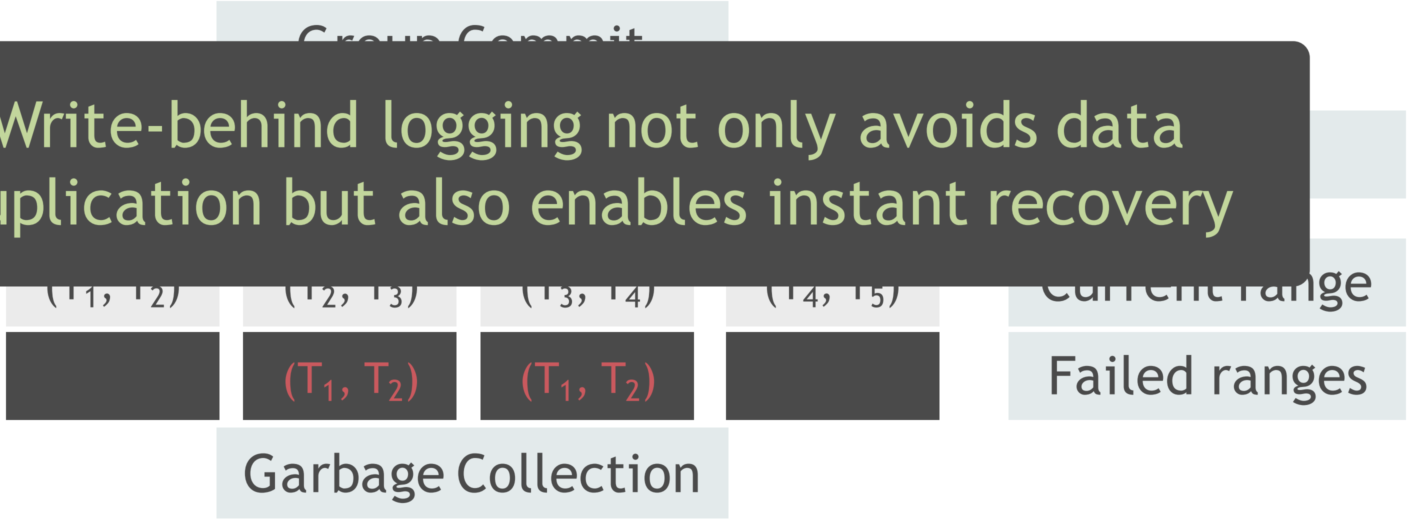
- DBMS assigns timestamps to transactions
  - All transactions in a particular group commit
  - Get timestamps within same group commit timestamp range
  - To ignore the effects of all in-flight transactions
- Idea: Use failed group commit timestamp range
  - DBMS uses this timestamp range during tuple visibility checks
  - Ignores tuples created or updated within this timestamp range
  - UNDO is, therefore, implicitly done via visibility checks

WRITE-BEHIND LOGGING

- Recovery consists of only analysis phase
  - Can immediately start processing transactions after restart
- Garbage collection eventually kicks in
  - Undoes effects of all uncommitted transactions
  - Using timestamp range information in write-behind log
  - After this finishes, no need to do extra visibility checks

METADATA FOR INSTANT RECOVERY

- Group commit timestamp range
  - Use it to ignore effects of transactions in failed group commit
  - Maintain list of failed timestamp ranges

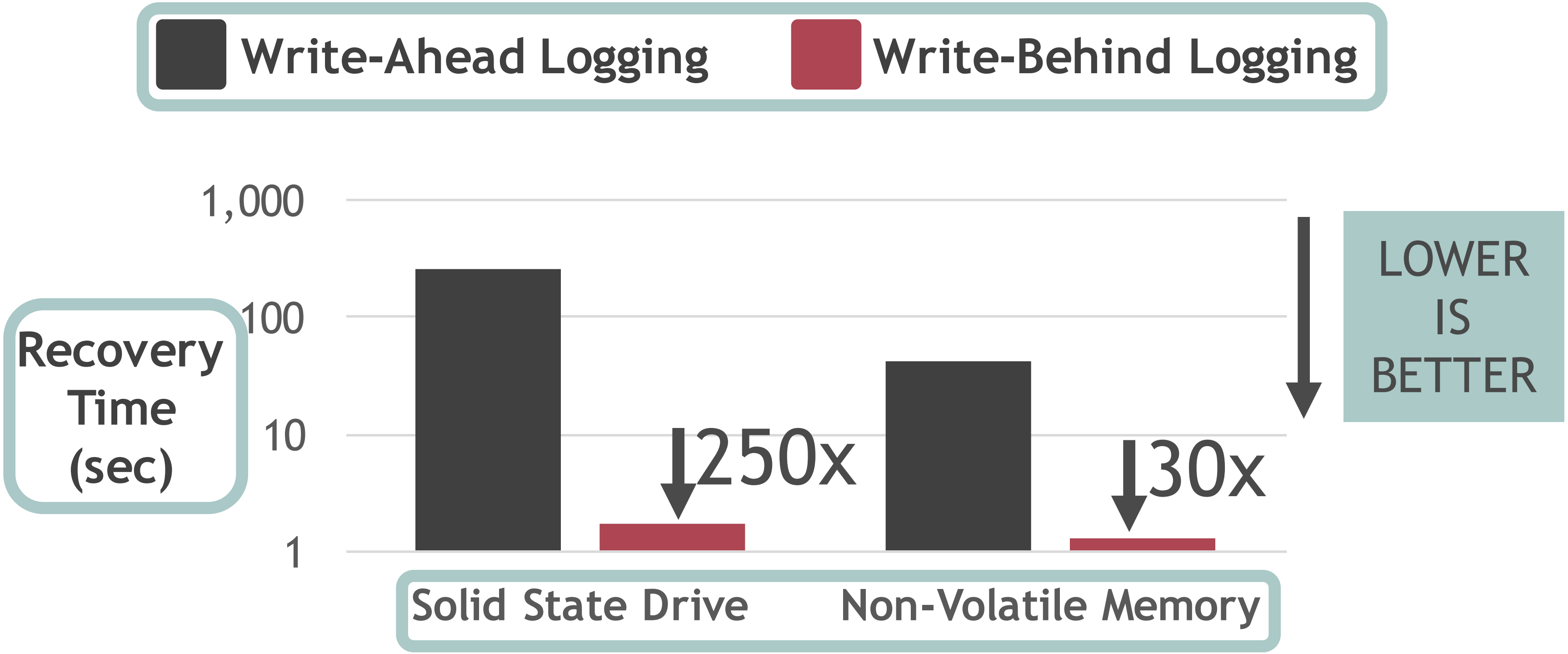


Write-behind logging not only avoids data duplication but also enables instant recovery

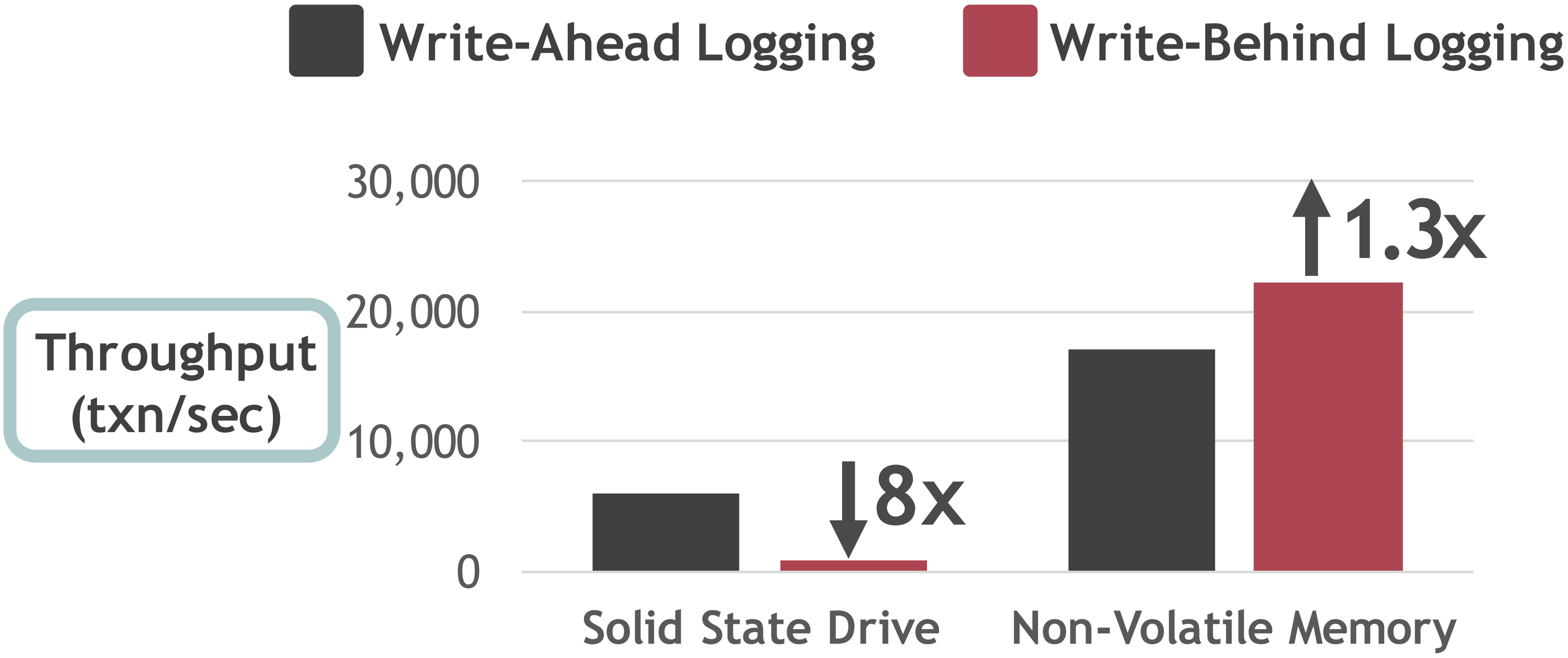
EVALUATION SETUP

- Compare logging protocols in Peloton DBMS
  - Write-Ahead logging
  - Write-Behind logging
- TPC-C benchmark
- Storage devices
  - Solid-state drive
  - Non-volatile memory

RECOVERY TIME



THROUGHPUT


Write-Ahead Logging
Write-Behind Logging
Throughput (txn/sec)
30,000
20,000
10,000
0
↓8x
↑1.3x
Solid State Drive
Non-Volatile Memory

RECAP: WRITE-BEHIND LOGGING

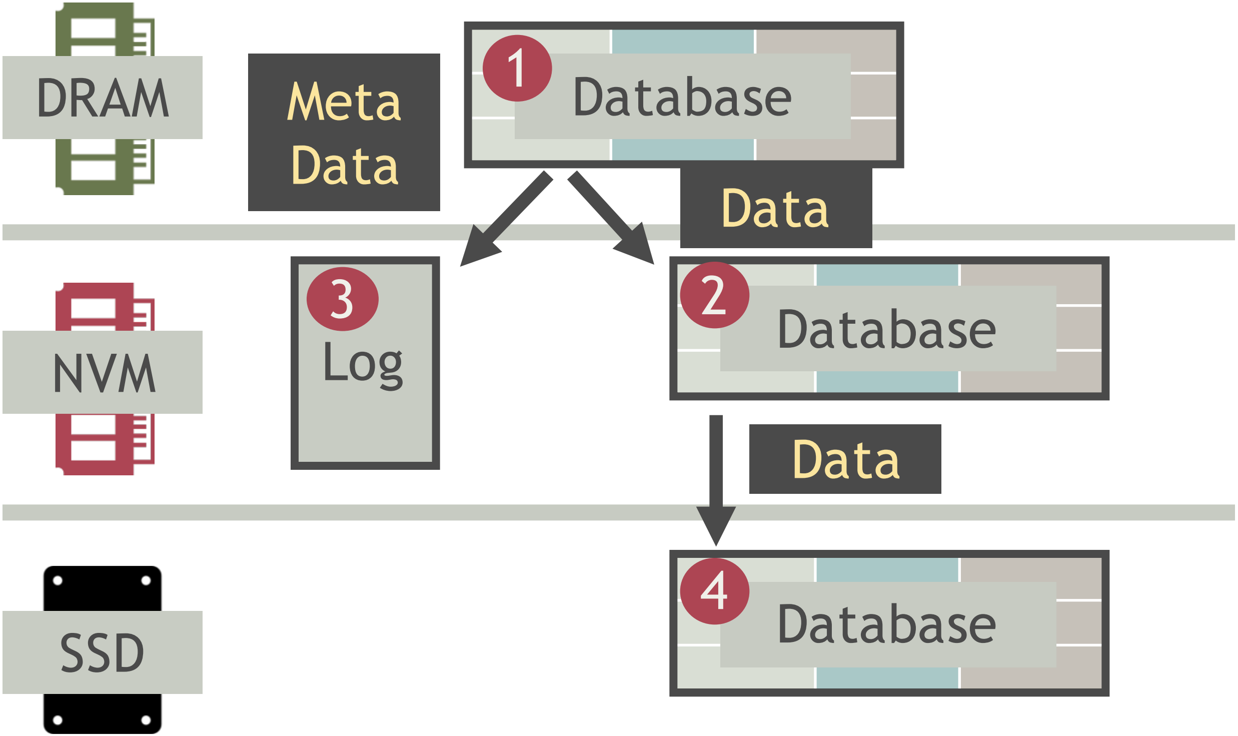
- Rethinking key algorithms
  - Write-behind logging enables instant recovery
  - Improves device utilization by reducing data duplication
  - Extends the device lifetime

DATA PLACEMENT

THREE-TIER STORAGE HIERARCHY

- Cost of first-generation NVM devices
  - SSD is still going to be in the picture
- Data placement
  - Three-tier DRAM + NVM + SSD hierarchy

THREE-TIER STORAGE HIERARCHY



DATA PLACEMENT

- Unlike SSD, can directly read data from NVM
  - No need to always copy data over to DRAM for reading
- Cache hot data in DRAM
  - Dynamically migrate cold data to SSD
  - And keep warm data on NVM

OPEN PROBLEM:
How do NVM capacity and access latencies affect the performance of DBMS?

ACCESS METHODS

NVM-AWARE ACCESS METHODS

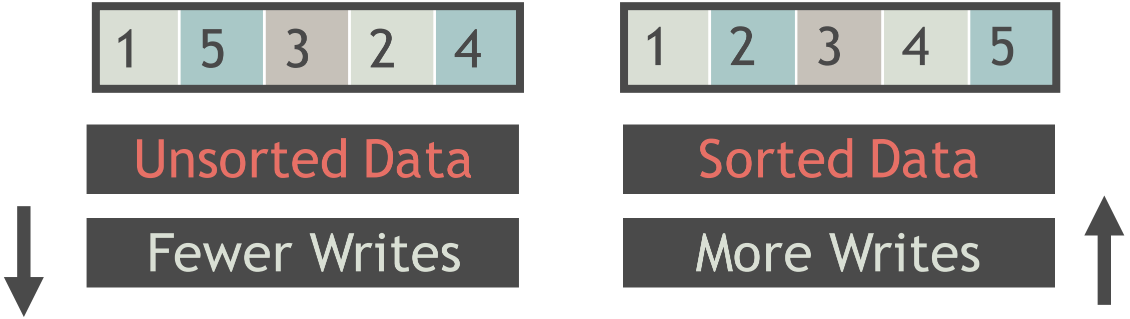
- Read-write asymmetry & wear-leveling
  - Writes might take longer to complete compared to reads
  - Excessive writes to a single NVM cell can destroy it
- Write-limited access methods
  - NVM-aware B+tree, hash table

Perform fewer writes, and instead do more reads

FPTREE: A HYBRID SCM-DRAM PERSISTENT AND CONCURRENT B-TREE FOR STORAGE CLASS MEMORY
SIGMOD 2016

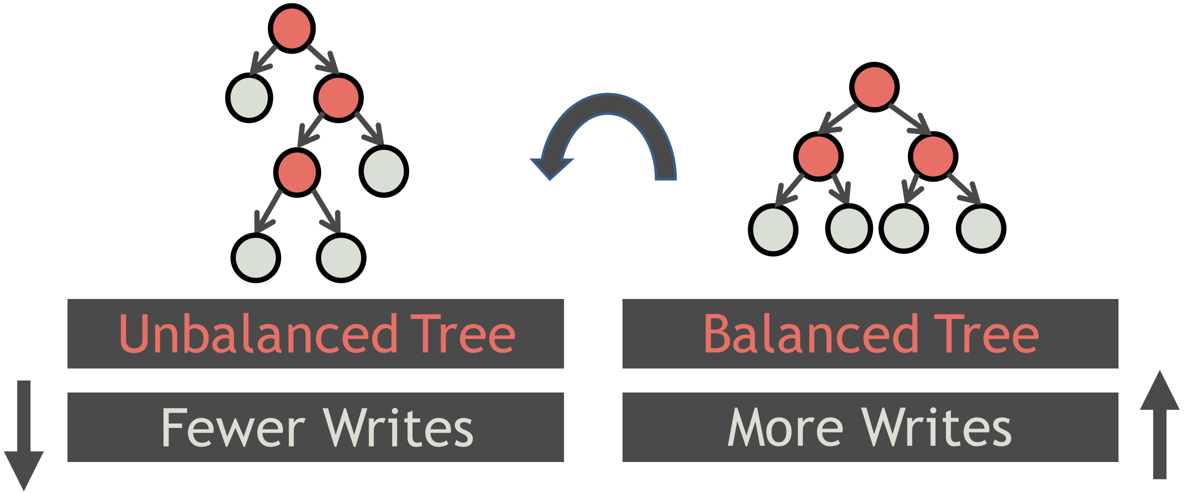
NVM-AWARE B+TREE

- Leave the entries in the leaf node unsorted
  - Require a linear scan instead of a binary search
  - But, fewer writes associated with shuffling entries



NVM-AWARE B+TREE

- Temporarily relax the balance of the tree
  - Extra node reads, fewer writes associated with balancing nodes



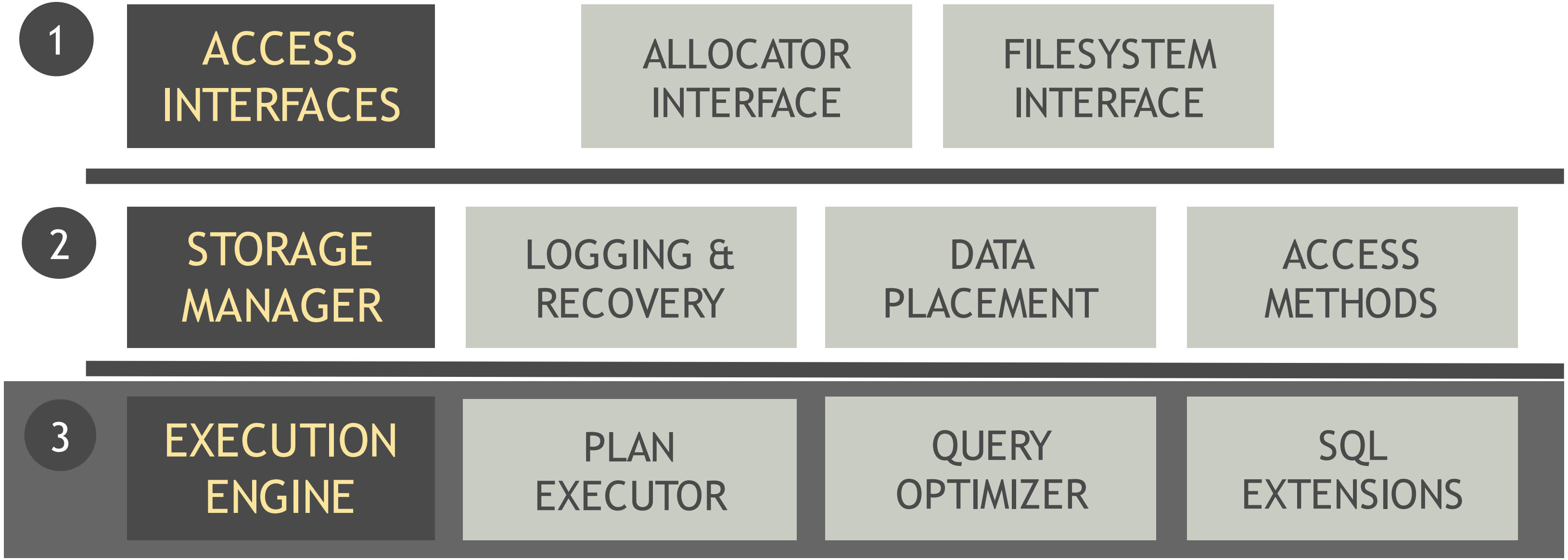
NVM-AWARE ACCESS METHODS

- More design principles will be covered in next tutorial

Data Structures Engineering For NVM
Ismail Oukid and Wolfgang Lehner, TU Dresden

OPEN PROBLEM:
Synthesizing other NVM-aware access methods.

BLUEPRINT OF AN NVM DBMS



HOW TO BUILD A NON-VOLATILE MEMORY DBMS
SIGMOD 2017 (TUTORIAL)

EXECUTION ENGINE

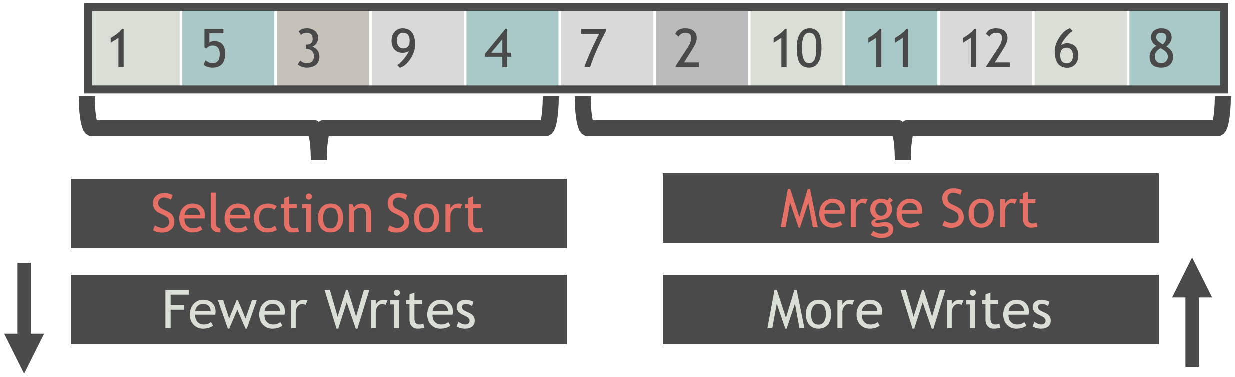
PLAN EXECUTOR

- Query processing algorithms
  - Sorting algorithm
  - Join algorithm
- Reduce the number of writes
  - Adjusting the write-intensivity knob
  - Write-limited algorithms

WRITE-LIMITED SORTS AND JOINS FOR PERSISTENT MEMORY
VLDB 2014

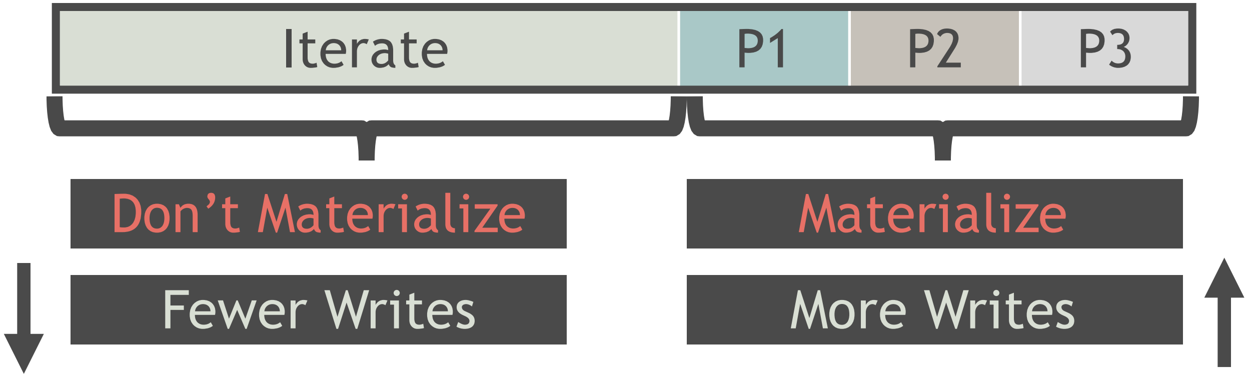
SEGMENT SORT

- Hybrid sorting algorithm
  - Run merge sort on a part of the input (segment): x%
  - Run selection sort on the rest of the input: (1-x)%
  - Adjust “x” to limit the number of writes



SEGMENT GRACE JOIN

- Hybrid join algorithm
  - Materialize a part of the input partitions: x%
  - Iterate over input for remaining partitions: (1-x)%
  - Adjust “x” to limit the number of writes



SQL EXTENSIONS

SQL EXTENSIONS

- Allow the user to control data placement on NVM
  - Certain performance-critical tables and materialized views

```
ALTER TABLESPACE nvm_table_space DEFAULT ON_NVM;
```

- Store only a subset of the columns on NVM
  - Exclude certain columns from being stored on NVM

```
ALTER TABLE orders ON_NVM EXCLUDE(order_tax);
```

NVM-RELATED SQL EXTENSIONS

- Need to construct new NVM-related extensions
  - Standardize these extensions

OPEN PROBLEM:
Need to construct new extensions and standardize them.

QUERY OPTIMIZER

QUERY OPTIMIZATION

- Cost-based query optimizer
  - Distinguish between sequential & random accesses
  - *But not between reads and writes*
- NVM-oriented redesign
  - *Differentiate between reads and writes in cost model*

MAKING COST-BASED QUERY OPTIMIZATION ASYMMETRY-AWARE
DAMON 2012

SEQUENTIAL SCAN

- Accounts for sequential access of all pages in table
  - Does not distinguish reads and writes

$$Cost(seqential\ scan) = Cost_{sequential}\ \|Table\|_{page\text{-}count}$$

- Updated cost function

$$Cost(seqential\ scan) = Cost_{sequential\text{-}reads}\ \|Table\|_{page\text{-}count}$$

HASH JOIN

- Function accounts for reading and writing all data once
  - Does not distinguish reads and writes

$$\text{Cost(hash join)} = (\text{Cost}_{\text{sequential}} + \text{Cost}_{\text{random}}) * (\ \|\text{Inner-Table}\|_{\#\text{pages}} + \|\text{Outer-Table}\|_{\#\text{pages}})$$

- Updated cost function

$$\text{Cost(hash join)} = (\text{Cost}_{\text{sequential-reads}} + \text{Cost}_{\text{random-writes}}) * (\ \|\text{Inner-Table}\|_{\#\text{pages}} + \|\text{Outer-Table}\|_{\#\text{pages}})$$

EVALUATION

- Compare different cost models on NVM emulator
  - Traditional cost model
  - NVM-aware cost model
- TPC-H benchmark on Postgres
- Performance impact
  - 50% speedup of queries
  - Maximum speedup: 500% (!)
  - Maximum slowdown: 1%

NVM-ORIENTED DESIGN

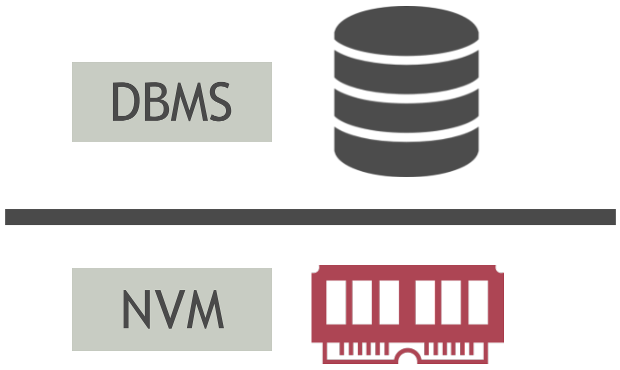
- Page-oriented cost functions
  - NVM is byte-addressable

OPEN PROBLEM:
Update cost model to factor in byte-addressability of NVM

LESSONS LEARNED

LESSONS LEARNED

- Important to reexamine the design choice
  - To leverage the raw device performance differential
  - Across different components of the DBMS
  - Helpful to think about an NVM-only hierarchy



LESSONS LEARNED

- NVM invalidates multiple long-held assumptions
  - Storage is several orders of magnitude slower than DRAM
  - Large performance gap between sequential & random accesses
  - Memory read and write latencies are symmetric

BLUEPRINT OF AN NVM DBMS

1	ACCESS INTERFACES	ALLOCATOR INTERFACE	FILESYSTEM INTERFACE	
2	STORAGE MANAGER	LOGGING & RECOVERY	DATA PLACEMENT	ACCESS METHODS
3	EXECUTION ENGINE	PLAN EXECUTOR	QUERY OPTIMIZER	SQL EXTENSIONS

HOW TO BUILD A NON-VOLATILE MEMORY DBMS
SIGMOD 2017 (TUTORIAL)

FUTURE WORK

- Highlighted a set of open problems
  - Data placement
  - Access methods
  - Query optimization
- Improvement in performance of storage layer
  - By several orders of magnitude over a short period of time
  - We anticipate high-impact research in this space

PELOTON

http://pelotondb.org

NVM Ready

Autonomous

Apache Licensed

END

@joy_arulraj & @andy_pavlo