Beyond Main Memory

Anti-Caching in Main Memory Database Systems

Justin DeBrabant
Brown University

BROWN
Database Group

Talk Overview

1. anti-caching architecture
   - larger than memory datasets in main memory DBMS
2. anti-caching + persistent memory
   - ditching the disk (finally!)

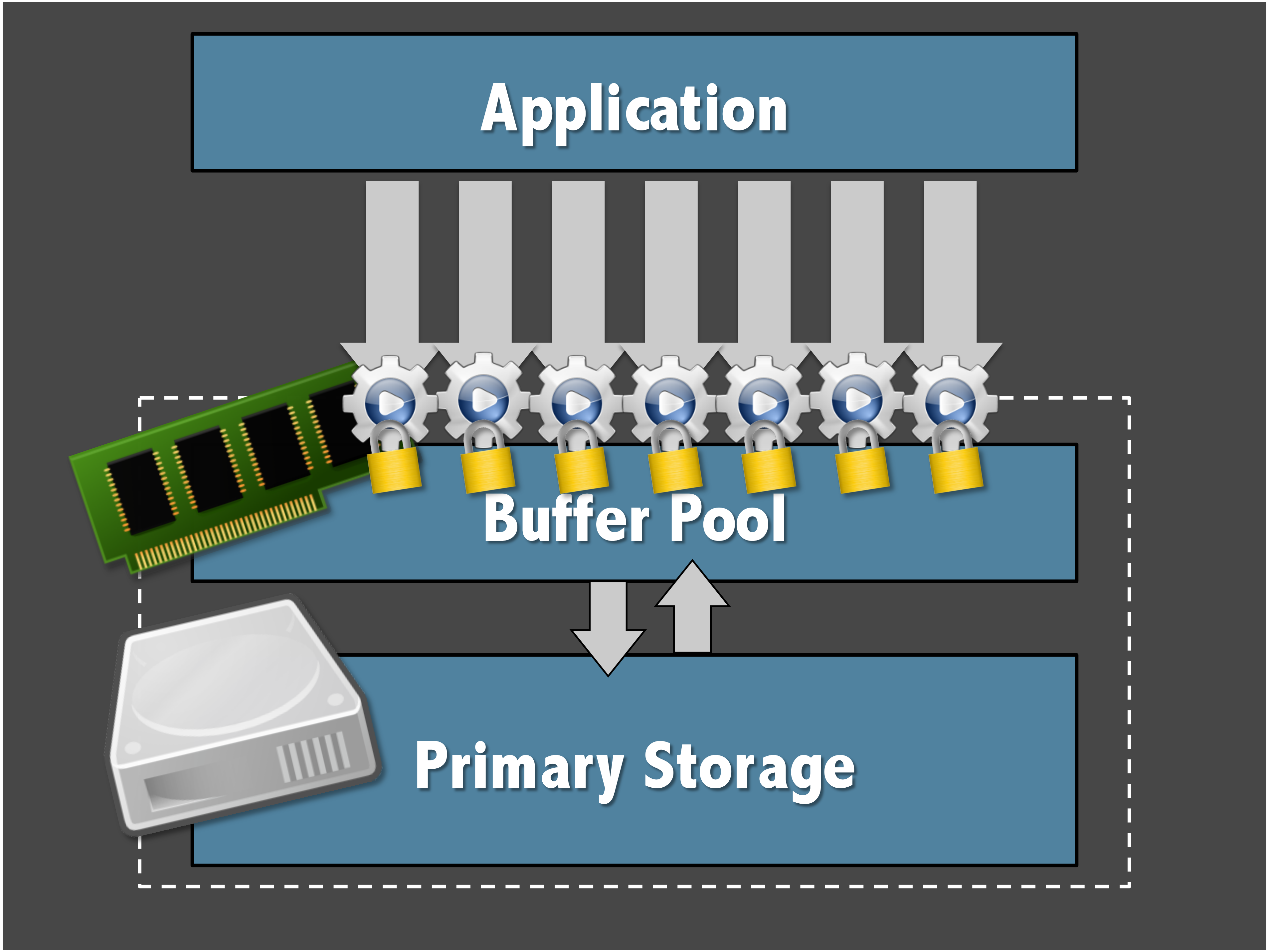
BROWN
Database Group

A bit of history…

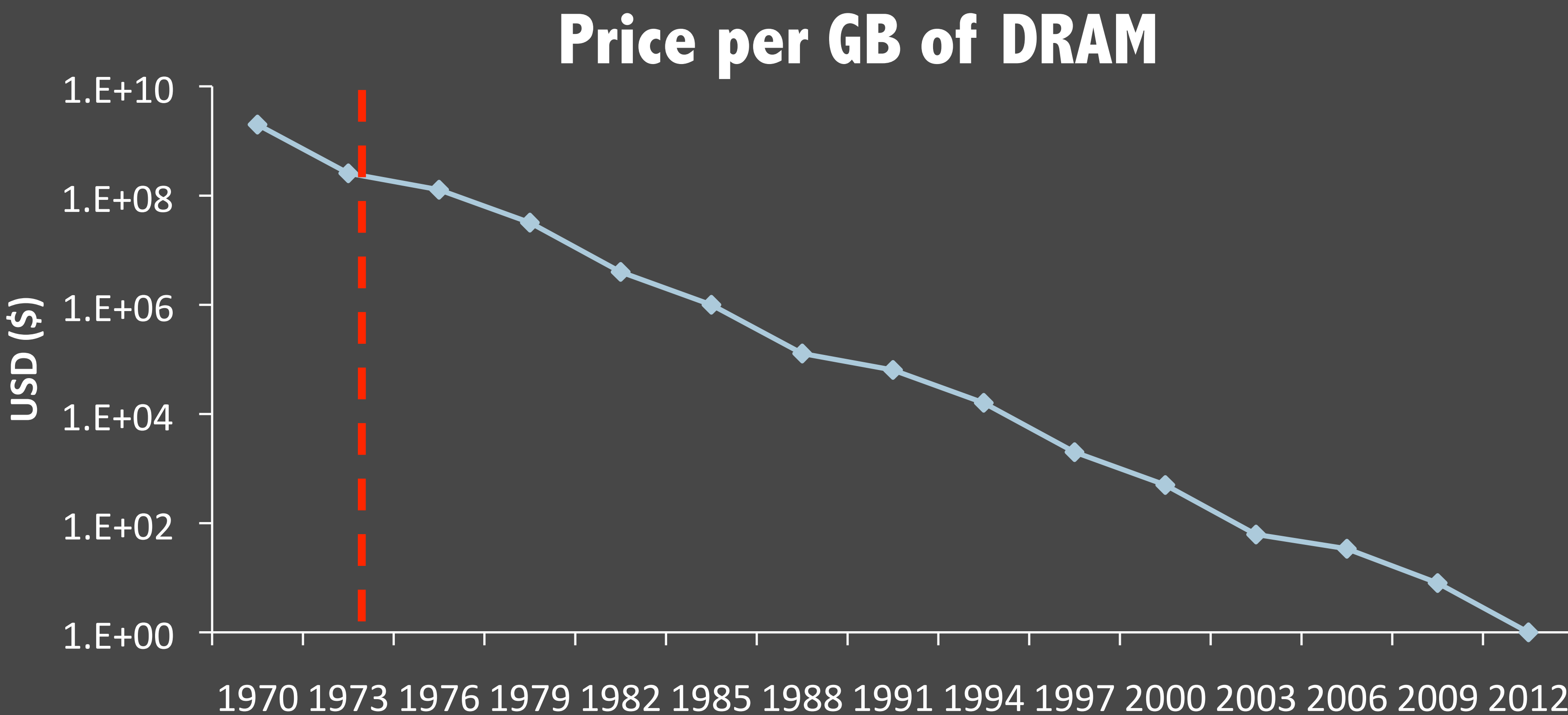
BROWN
Database Group

1974 – System R

- query optimization
- recovery
- transaction serialization
  - allows concurrent execution of transactions
  - lots of locks


Application
Buffer Pool
Primary Storage

Change is Good



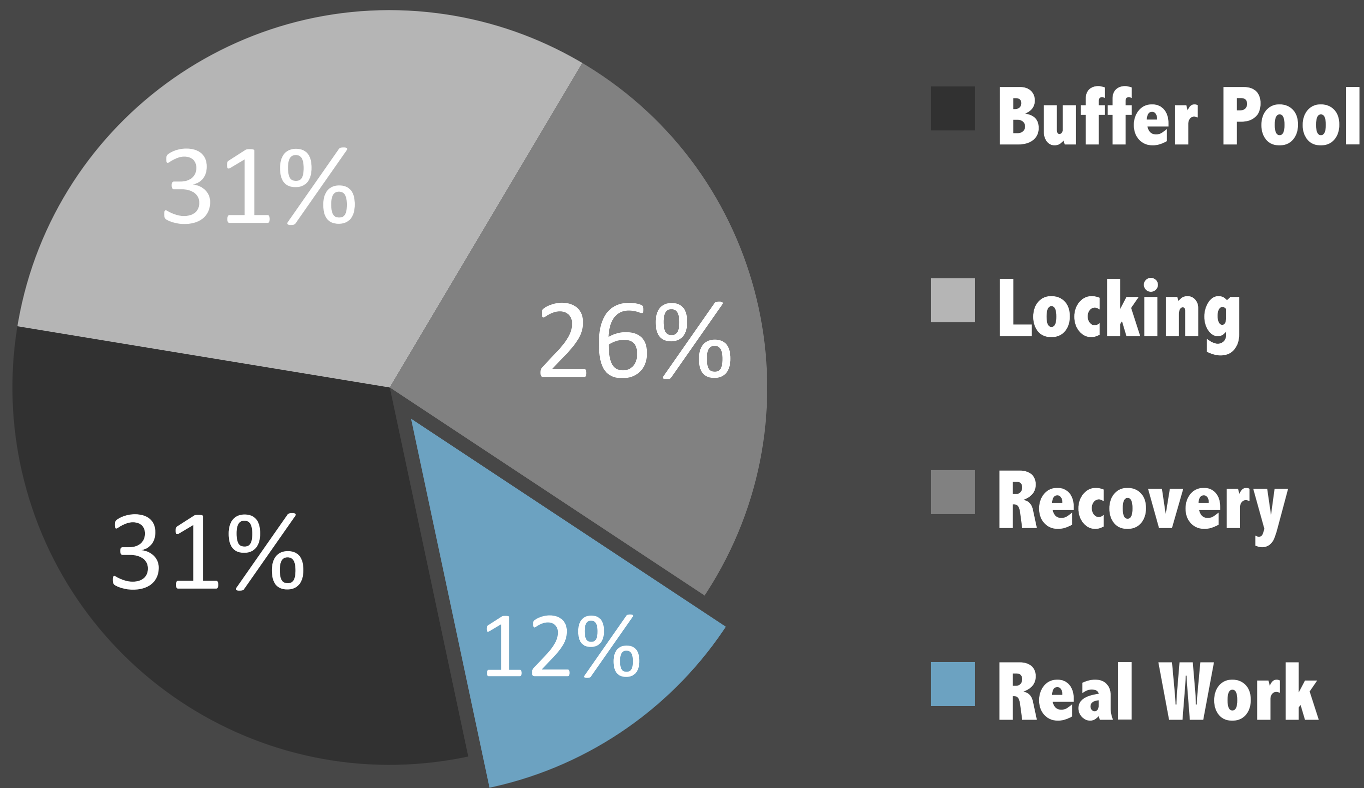
BROWN
Database Group

source: http://www.archivebuilders.com/whitepapers/22045p.pdf

Great, that's what the buffer pool is for…right?

More Memory → Higher Throughput?

- all data resides in memory (i.e. in buffer pool)
  - No disk stalls
- must still…
  - maintain buffer pool
  - lock/latch data
  - maintain ARIES-style recovery logs
- question: What is the overhead of all these things?



OLTP Through the Looking Glass, and What We Found There
SIGMOD '08

BROWN
Database Group

Ok, no buffer pool, how about a distributed cache?

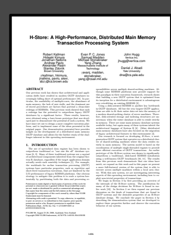
Distributed Caches

- e.g. memcached
- just in-memory key-value pair
- no inherent persistence
- application programmer must maintain consistency
  - or not!

So, we need a hybrid of these two architectures.

H-Store

Parallel Main Memory Transaction Processing System


H-Store: A High-Performance, Distributed Main Memory Transaction Processing System

H-Store: A High-Performance, Distributed Main Memory Transaction Processing System
VLDB 2008.

BROWN
Database Group

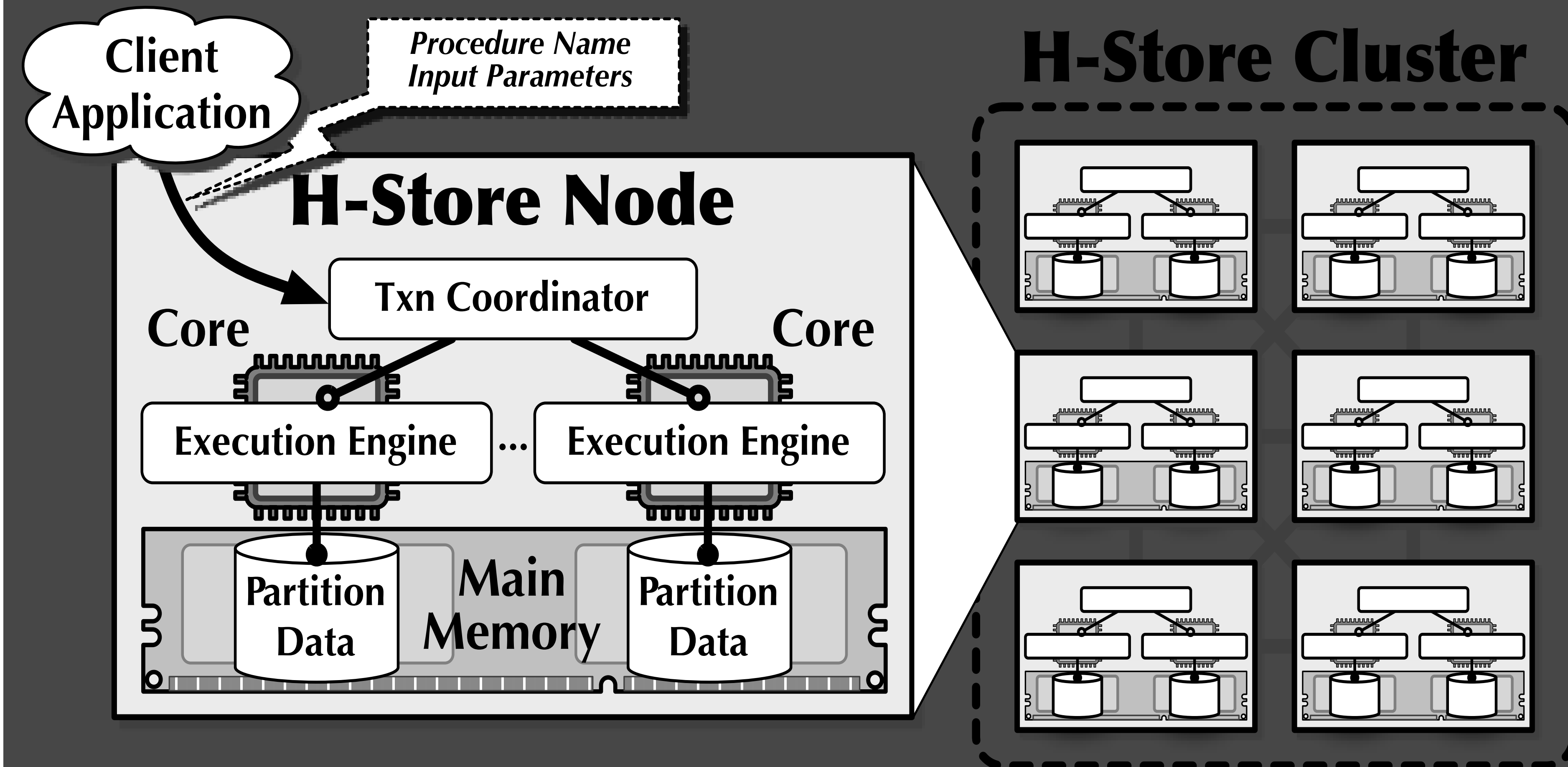
H-Store Architecture

- partitioned, shared-nothing
  - data is sharded across nodes
- single-threaded main memory execution
  - no need for locks and latches

H-Store Architecture (cont'd)

- stored procedures
  - no ad hoc queries in OLTP
  - command logging
- recovery
  - snapshots + command log

BROWN
Database Group

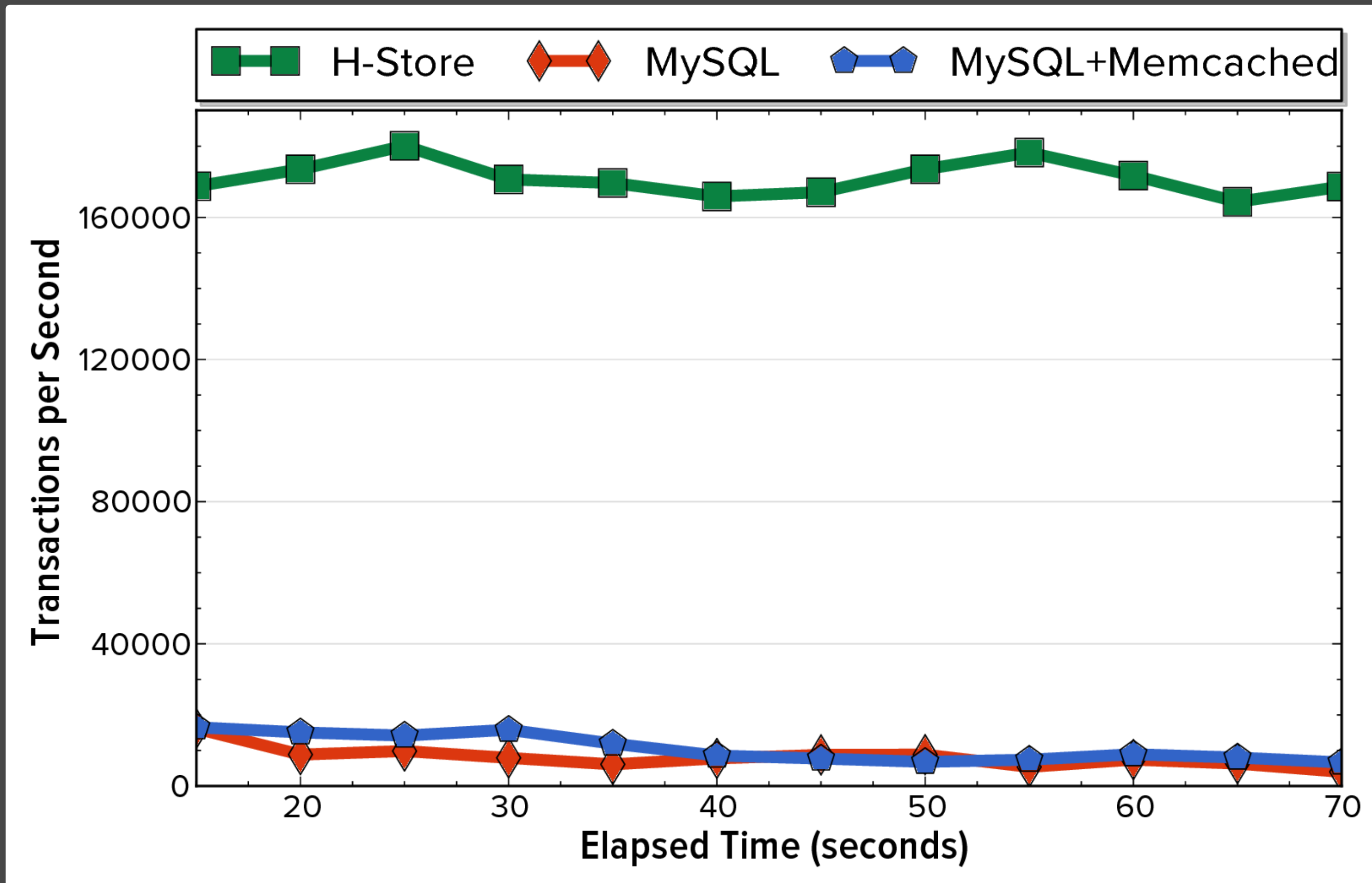


BROWN
Database Group

H-Store Assumptions

1. OLTP workload
    - short-lived transactions that touch only a few records at a time
2. mostly single-site transactions
    - distributed transactions need multi-node coordination
3. data fits in memory
    - virtual memory is bad!

YCSB, update-heavy, data < memory



BROWN
Database Group

Assumptions: Revisited

1. OLTP workload
   - "One Size Fits All": An Idea Whose Time Has Come and Gone
     - *ICDE '05*
   - The End of An Architectural Era: (Its Time for A Complete Rewrite)
     - *VLDB '07*

Assumptions: Revisited

2. mostly single-site transactions

- **Skew-Aware Automatic Database Partitioning In Shared-Nothing, Parallel OLTP Systems**
  - *SIGMOD '12*
- **On Predictive Modeling For Optimizing Transaction Execution in Parallel OLTP Systems**
  - *VLDB '11*

BROWN
Database Group

Assumptions: Revisited

BROWN
Database Group

Workload Skew Exists!

- hot data in memory
- cold data to disk
- goals
  - maintain transactional consistency
  - avoid blocking

Anti-Caching

BROWN
Database Group

Anti-Caching Phases

- evict
- pre-pass
- fetch
- merge

BROWN
Database Group

Evict

1. data > anti-cache threshold
2. dynamically construct anti-cache blocks of coldest tuples
3. asynchronously write to disk

Pre-Pass

1. a transaction enters pre-pass when evicted data is accessed
2. continues execution, creating list of evicted blocks
3. abort, queue blocks to be fetched

Fetch

1. data is fetched asynchronously from disk
   - avoids blocking
2. copied into merge buffer

BROWN
Database Group

Merge

1. previously aborted transaction is restarted
2. moves data from merge buffer to normal table
3. transaction executes normally

BROWN
Database Group

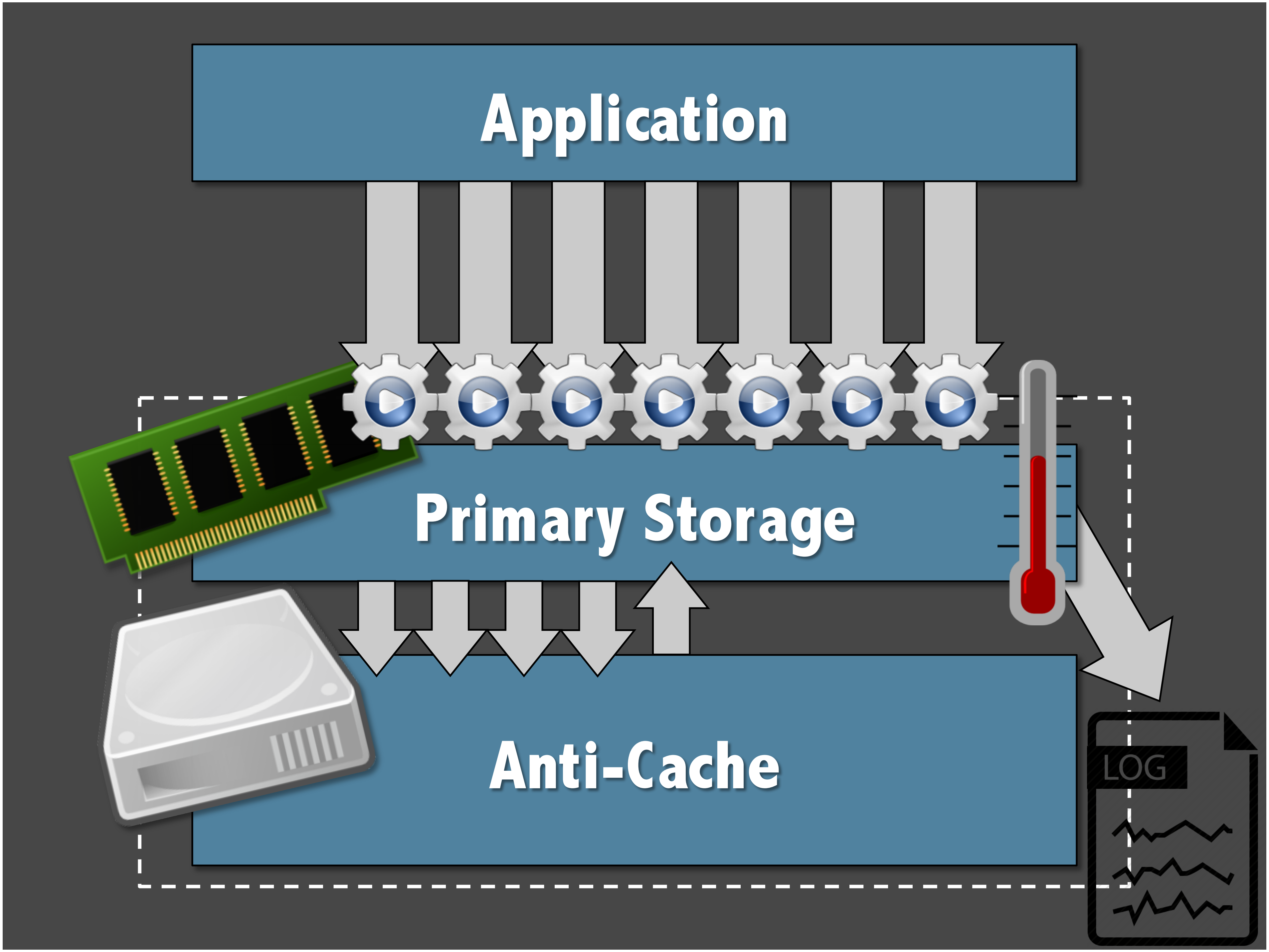
Multiple Restarts

- in-memory data for restarted transaction is relatively cold
  - mark tuples in pre-pass phase as hot
- data dependencies with evicted tuples
  - mark recently merged tuples as hot
- larger-than-memory queries still an issue
  - not in OLTP

Other Design Points

- LRU chain
  - embedded in tuple headers → *O(1)* updates
- EvictedTable
  - stores <tuple id, block id> pairs for evicted tuples
- anti-cache
  - BerkeleyDB

BROWN
Database Group


Application
Primary Storage
Anti-Cache
LOG

Sounds like swapping…

Anti-Caching vs. Swapping

- fine-grained eviction
  - blocks constructed dynamically
- non-blocking fetches
  - remove disk from critical path

Sounds like caching…

Anti-Caching vs. Caching

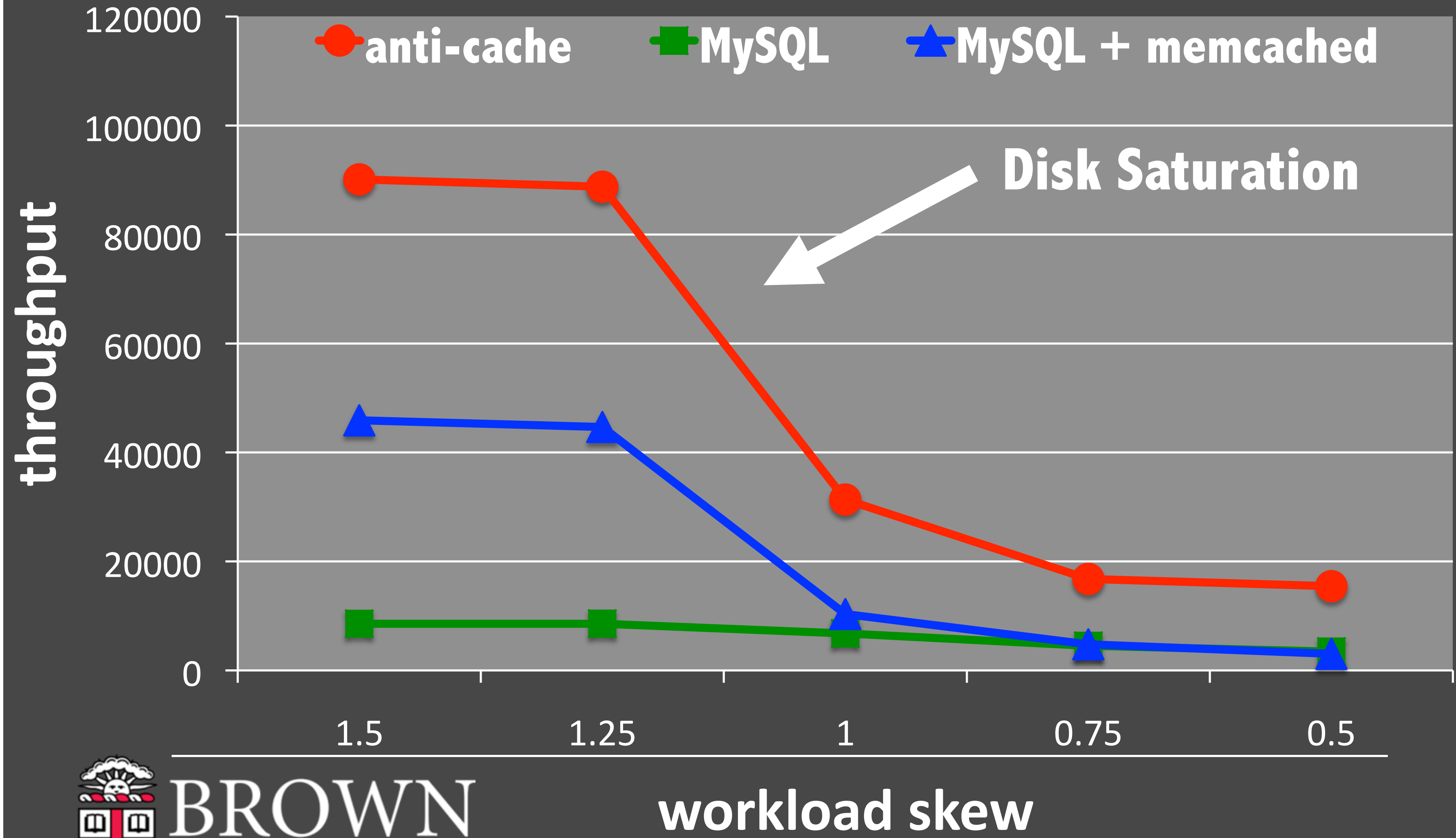
- data exists in exactly one location
  - caching architectures have multiple copies, must maintain consistency
  - data is moved, not copied
- goal is data size, not throughput

BROWN
Database Group

Benchmarking

- YCSB
- Zipfian skew
- data > memory
- read/write mix
- MySQL, MySQL + memcached

YCSB, read-only, data 8X memory



BROWN
Database Group

Conclusions

- 8-17X improvement for skewed workloads at 8X memory
  - avoids blocking for disk
  - fine-grained eviction
- disk becomes the bottleneck

Anti-Caching: A New Approach to Database Management System Architecture

Justin DeBrabant
Brown University
debrabant@cs.brown.edu

Andrew Pavlo
Brown University
pavlo@cs.brown.edu

Stephen Tu
MIT CSAIL
stephentu@csail.mit.edu

Michael Stonebraker
MIT CSAIL
stonebraker@csail.mit.edu

Stan Zdonik
Brown University
sbz@cs.brown.edu

ABSTRACT

The traditional wisdom for building disk-based relational database management systems (DBMS) is to organize data in heavily-encoded blocks stored on disk, with a main memory block cache. In order to improve performance given high disk latency, these systems use a multi-threaded architecture with dynamic record-level locking that allows multiple transactions to access the database at the same time. Previous research has shown that this results in substantial overhead for on-line transaction processing (OLTP) applications [15].

The next generation DBMSs seek to overcome these limitations with architecture based on main memory resident data. To overcome the restriction that all data fit in main memory, we propose a new technique, called *anti-caching*, where cold data is moved to disk in a transactionally-safe manner as the database grows in size. Because data initially resides in memory, an anti-caching architecture reverses the traditional storage hierarchy of disk-based systems. Main memory is now the primary storage device.

We implemented a prototype of our anti-caching proposal in a high-performance, main memory OLTP DBMS and performed a series of experiments across a range of database sizes, workload skews, and read/write mixes. We compared its performance with an open-source, disk-based DBMS optionally fronted by a distributed main memory cache. Our results show that for higher skewed workloads the anti-caching architecture has a performance advantage over either of the other architectures tested of up to 9× for a data size 8× larger than memory.

1. INTRODUCTION

Historically, the internal architecture of DBMSs has been predicated on the storage and management of data in heavily-encoded disk blocks. In most systems, there is a header at the beginning of each disk block to facilitate certain operations in the system. For example, this header usually contains a "line table" at the front of the block to support indirection to tuples. This allows the DBMS to reorganize blocks without needing to change index pointers. When a disk block is read into main memory, it must then be translated into main memory format.

Proceedings of the VLDB Endowment, Vol. 6, No. 14

DBMSs invariably maintain a buffer pool of blocks in main memory for faster access. When an executing query attempts to read a disk block, the DBMS first checks to see whether the block already exists in this buffer pool. If not, a block is evicted to make room for the needed one. There is substantial overhead to managing the buffer pool, since blocks have to be pinned in main memory and the system must maintain an eviction order policy (e.g., least recently used). As noted in [15], when all data fits in main memory, the cost of maintaining a buffer pool is nearly one-third of all the CPU cycles used by the DBMS.

The expense of managing disk-resident data has fostered a class of new DBMSs that put the entire database in main memory and thus have no buffer pool [11]. TimesTen was an early proponent of this approach [31], and more recent examples include H-Store [2, 18], MemSQL [3], and RAMCloud [25]. H-Store (and its commercial version VoltDB [4]) performs significantly better than disk-based DBMSs on standard OLTP benchmarks [29] because of this main memory orientation, as well as from avoiding the overhead of concurrency control and heavy-weight data logging [22].

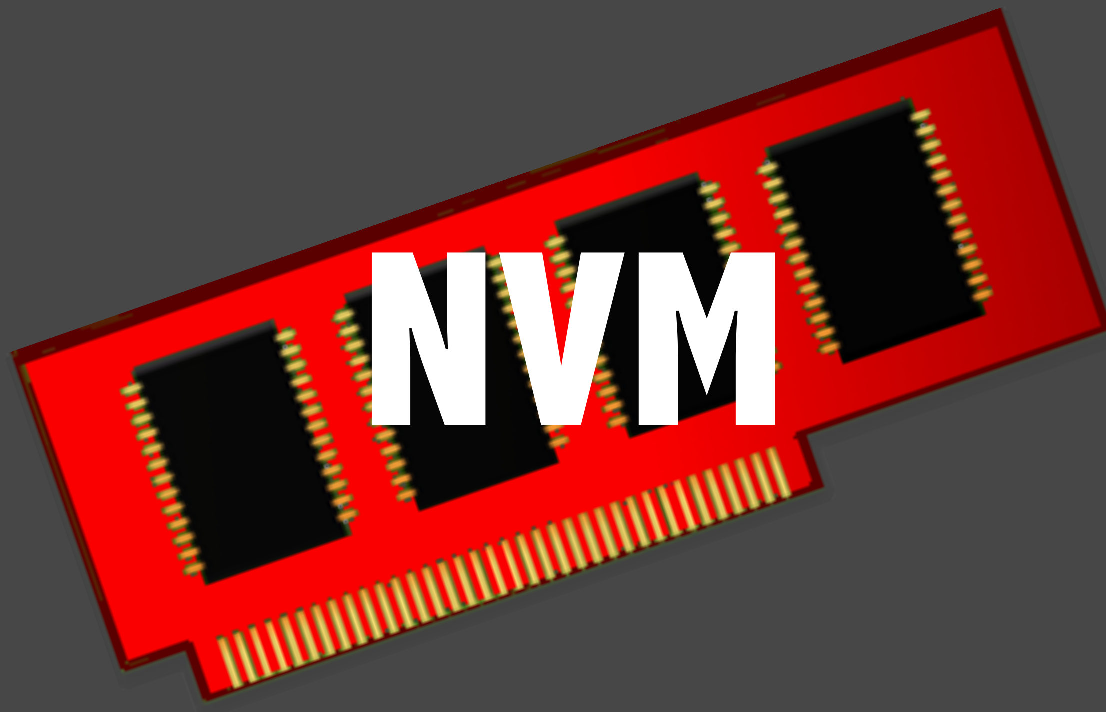
The fundamental problem with main memory DBMSs, however, is that this improved performance is only achievable when the database is smaller than the amount of physical memory available in the system. If the database does not fit in memory, then the operating system will start to page virtual memory, and main memory accesses will cause page faults. Because page faults are transparent to the user, in this case the main memory DBMS, the execution of transactions is stalled while the page is fetched from disk. This is a significant problem in a DBMS, like H-Store, that executes transactions serially without the use of heavyweight locking and latching. Because of this, all main memory DBMSs warn users not to exceed the amount of real memory [5]. If memory is exceeded (or if it might be at some point in the future), then a user must either (1) provision new hardware and migrate their database to a larger cluster, or (2) fall back to a traditional disk-based system, with its inherent performance problems.

One widely adopted performance enhancer is to use a main memory distributed cache, such as Memcached [14], in front of a disk-based DBMS. Under this two-tier architecture, the application first looks in the cache for the tuple of interest. If this tuple is not in the cache, then the application executes a query in the DBMS to fetch the desired data. Once the application receives this data from the DBMS, it updates the cache for fast access in the future. Whenever a tuple is modified in the database, the application must invalidate its cache entry so that the next time it is accessed the application will retrieve the current version from the DBMS. Many notable web

Anti-Caching: A New Approach to Database Management System Architecture

VLDB'13.

BROWN
Database Group


NVM

Memory Latencies (cycles)

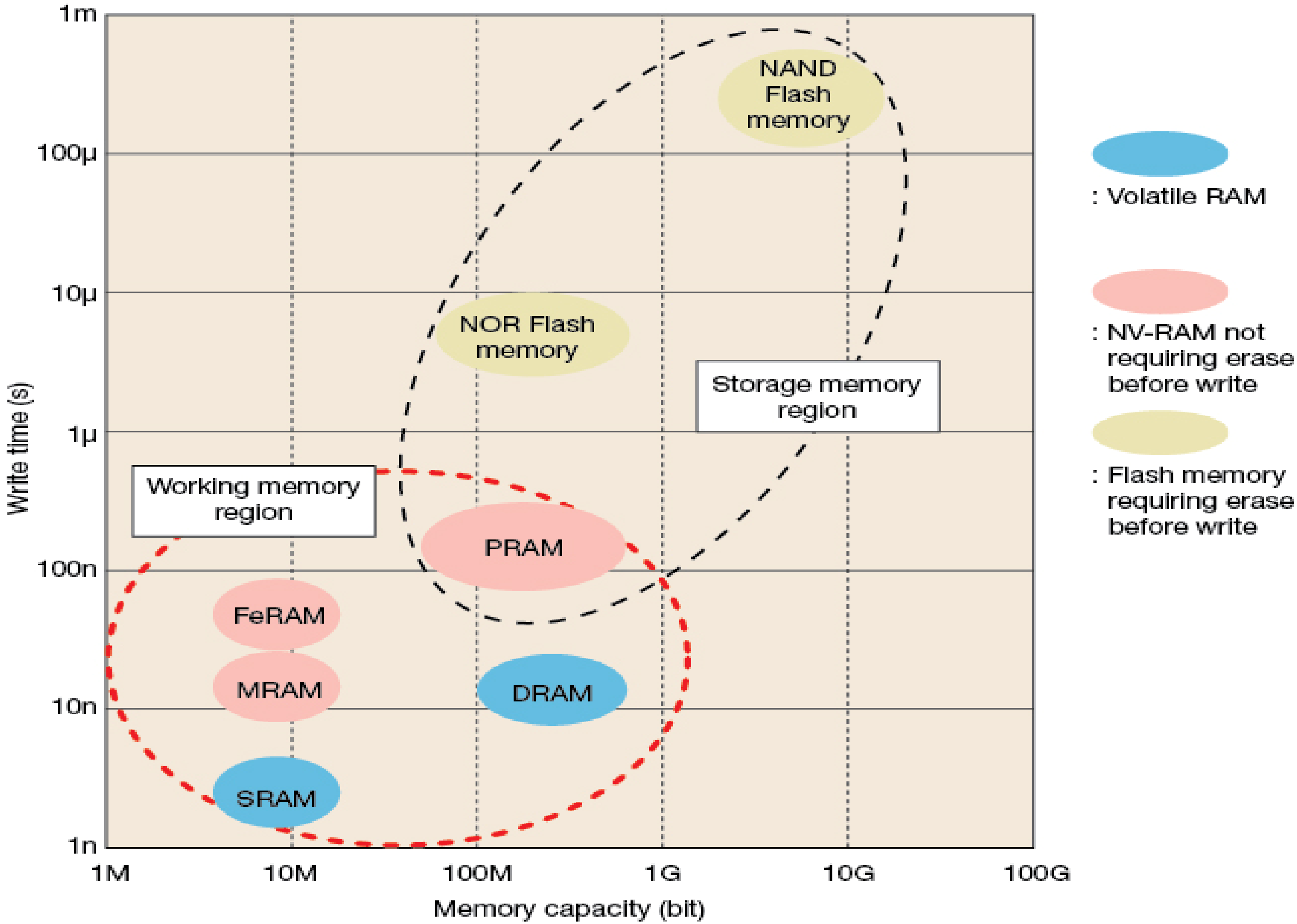
- SRAM: 1-30
- DRAM: 100-300
- flash: 25,000-2,000,000
- disk: 5,000,000+

NVM

- non-volatile
- random-access
- high write endurance
  - except flash
- byte-addressable
  - except flash

The Arms Race

- FeRAM
  - high write endurance
- MRAM
  - DRAM-like latency
- PCM (PRAM)
  - DRAM-like capacity



Courtesy: Motoyuki Ooishi

NVM Emulation

- goal: provide product-independent analysis
- test wide range of latency profiles
- automatically add specified latency
- built by collaborators at Intel

H-Store + NVM Architectures

1. **Anti-Cache to H-Store**

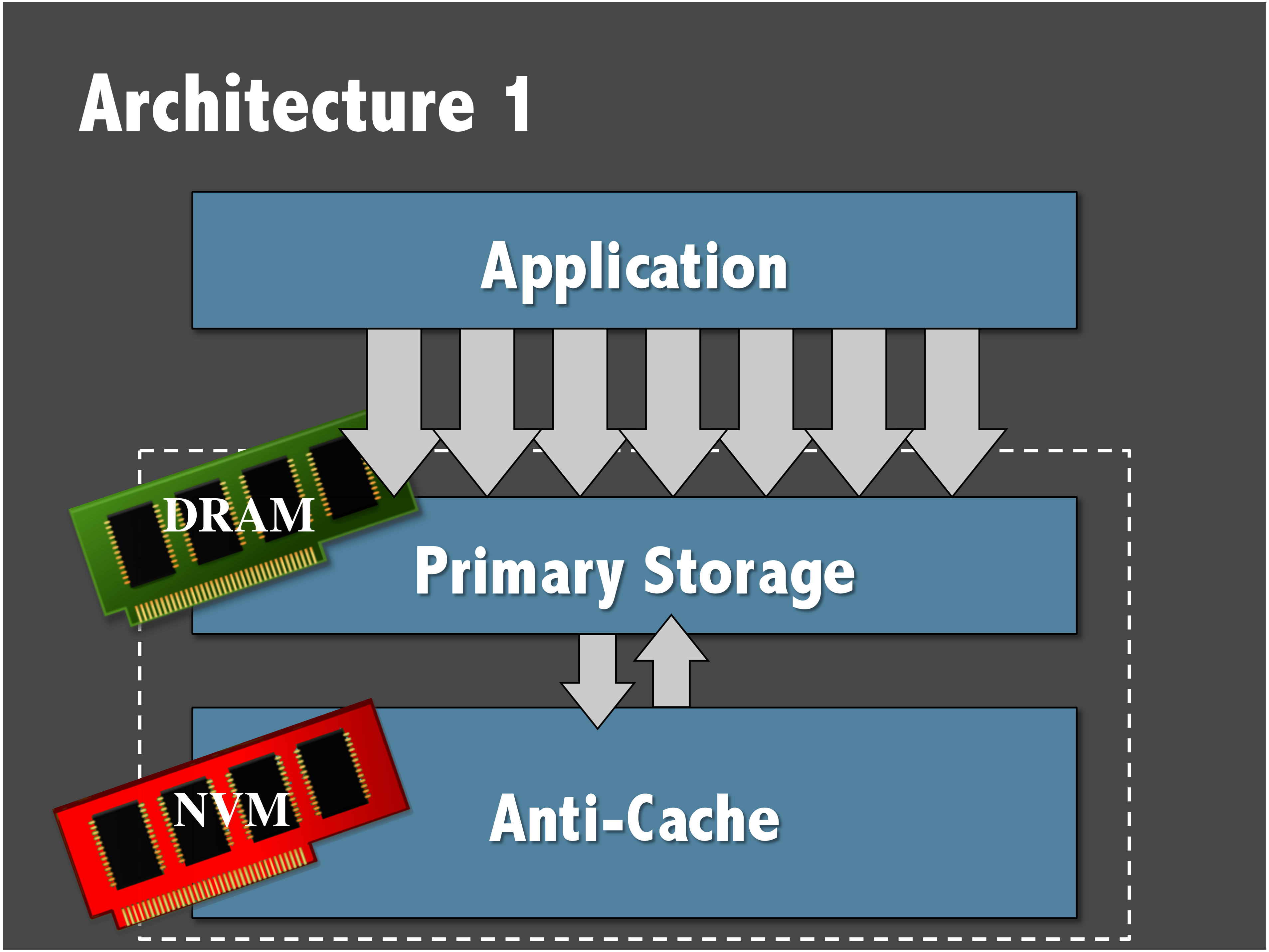
   + fully utilizes memory hierarchy

   - added memory overhead

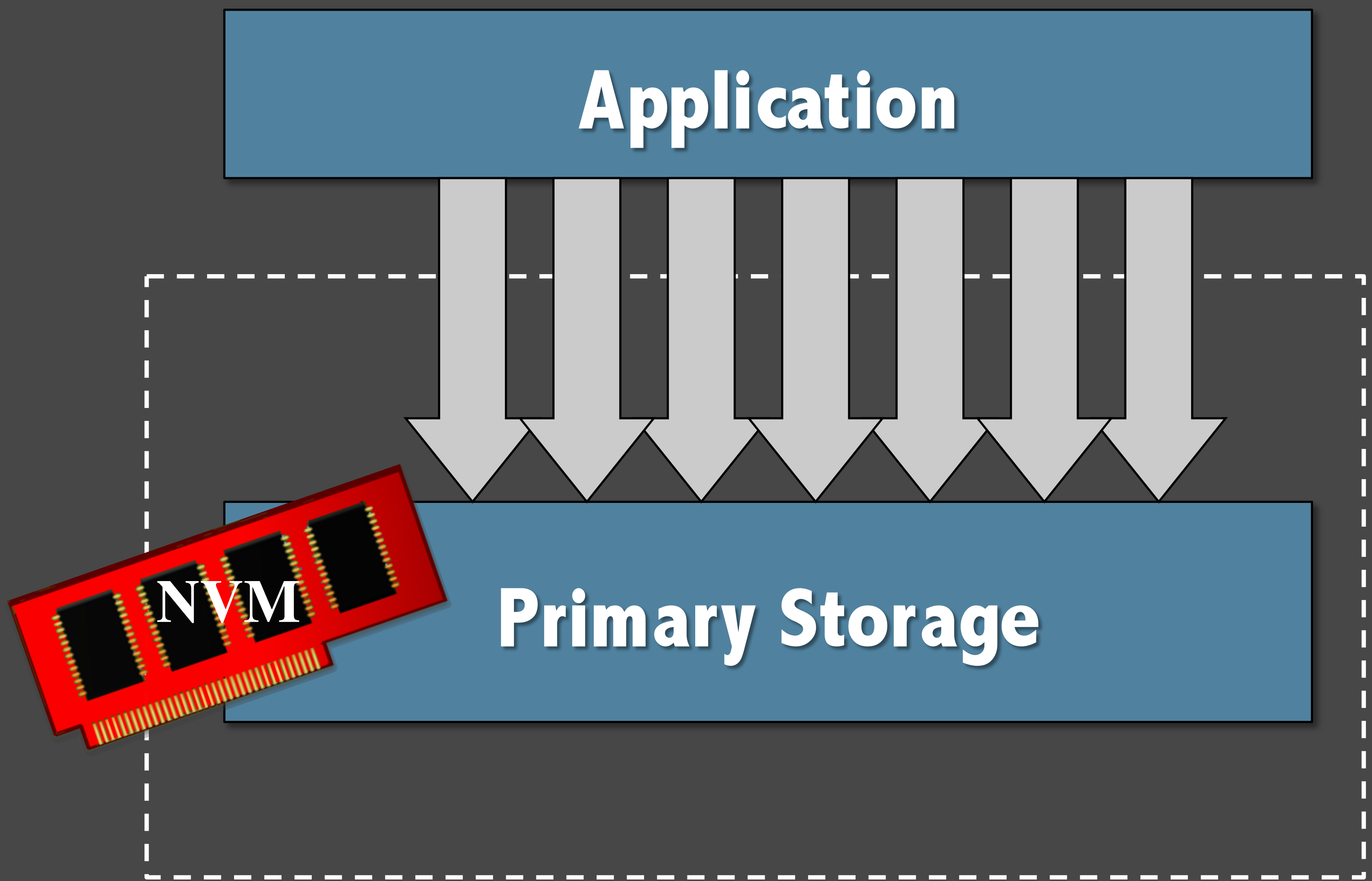
2. **H-Store on NVM**

   + no anti-cache overhead

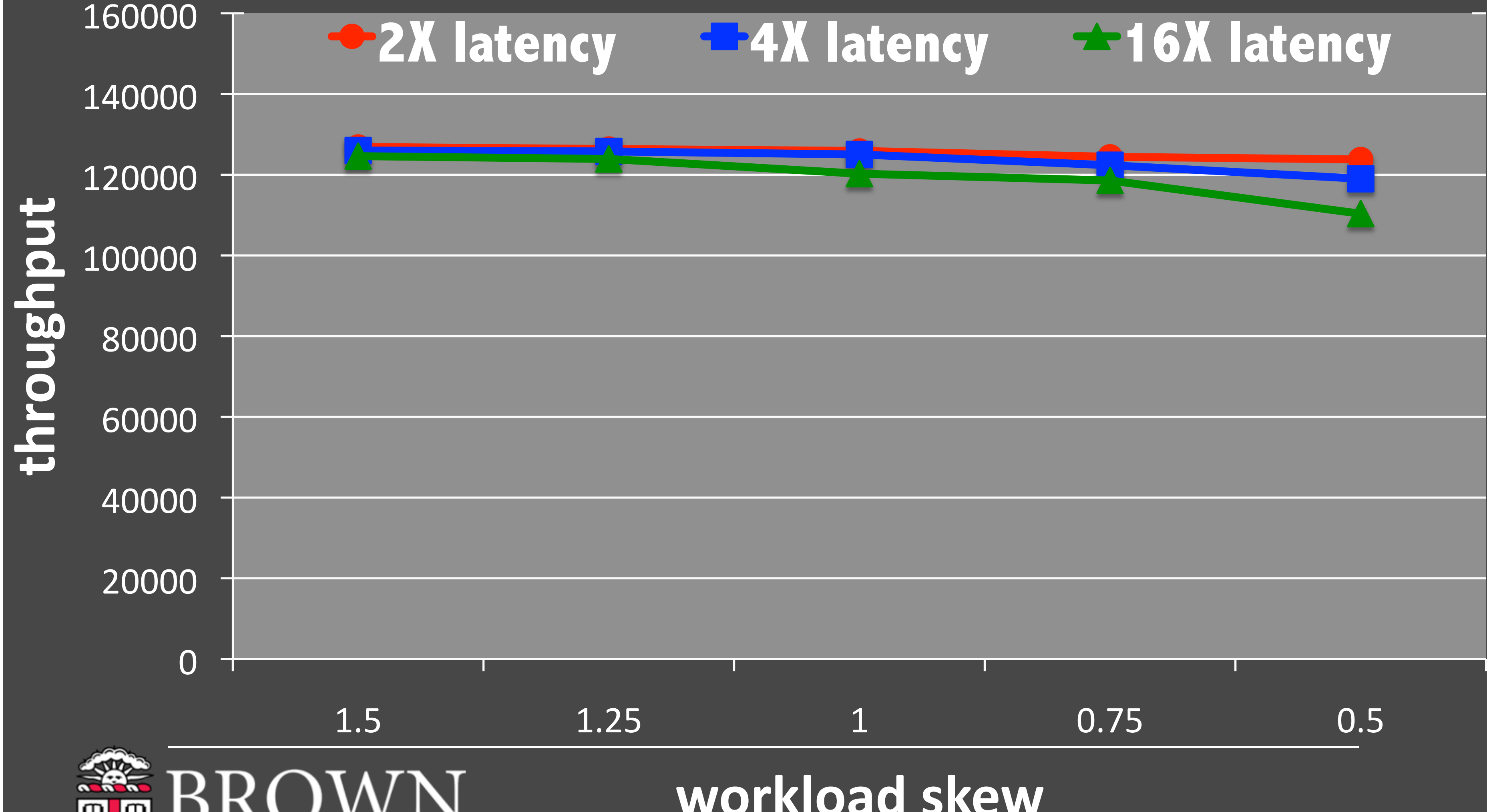
   - wear-leveling necessary


Architecture 1
Application
DRAM
Primary Storage
NVM
Anti-Cache

Architecture 2



Architecture 1, YCSB, read-only, data 8X memory



BROWN
Database Group

Work in Progress

- implementation of Architecture 2
  - also in H-Store
- further benchmarking
  - YCSB, TPC-C, possibly others
  - Architecture 1 vs. Architecture 2
- paper in preparation

Conclusions

- hardware has changed
- anti-caching to disk has 18-17X speedup over disk-based architecture
  - better utilization of available memory
- next-generation persistent memories extend the benefits of anti-caching beyond skewed workloads

BROWN
Database Group

Collaborators

Stan Zdonik (Brown)

Andy Pavlo (CMU)

Mike Stonebraker (MIT)

Stephen Tu (MIT)

Joy Arulraj (CMU)

ISTC BIG DATA

BROWN Database Group

Questions?

debrabant@cs.brown.edu

BROWN
Database Group