CS 764: Topics in Database Management Systems

Lecture 5: Buffer Management with NVM

Xiangyao Yu

9/22/2021

Today's Paper: Buffer Management with NVM

Spitfire: A Three-Tier Buffer Manager for Volatile and Non-Volatile Memory

Xinjing Zhou
zxjcarrot@gmail.com
Tencent Inc.

Joy Arulraj
arulraj@gatech.edu
Georgia Institute of Technology

Andrew Pavlo
pavlo@cs.cmu.edu
Carnegie Mellon University

David Cohen
david.e.cohen@intel.com
Intel

Abstract

The design of the buffer manager in database management systems (DBMSs) is influenced by the performance characteristics of volatile memory (*i.e.*, DRAM) and non-volatile storage (*e.g.*, SSD). The key design assumptions have been that the data must be migrated to DRAM for the DBMS to operate on it and that storage is orders of magnitude slower than DRAM. But the arrival of new non-volatile memory (NVM) technologies that are nearly as fast as DRAM invalidates these previous assumptions.

Researchers have recently designed Hymem, a novel buffer manager for a three-tier storage hierarchy comprising of DRAM, NVM, and SSD. Hymem supports cache-line-grained loading and an NVM-aware data migration policy. While these optimizations improve its throughput, Hymem suffers from two limitations. First, it is a single-threaded buffer manager. Second, it is evaluated on an NVM emulation platform. These limitations constrain the utility of the insights obtained using Hymem.

In this paper, we present Spitfire, a multi-threaded, three-tier buffer manager that is evaluated on real NVM hardware. We introduce a general framework for reasoning about data migration in a multi-tier storage hierarchy. We illustrate the limitations of the optimizations used in Hymem on Optane and then discuss how Spitfire circumvents them. We demonstrate that the data migration policy has to be tailored based on the characteristics of the devices and the workload. Given this, we present a machine learning technique for automatically adapting the policy for an arbitrary workload and storage hierarchy. Our experiments show that Spitfire works well across different workloads and storage hierarchies.

ACM Reference Format:
Xinjing Zhou, Joy Arulraj, Andrew Pavlo, and David Cohen. 2021. Spitfire: A Three-Tier Buffer Manager for Volatile and Non-Volatile Memory. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3448016.3452819

1 Introduction

The techniques for buffer management in a canonical DRAM-SSD hierarchy are predicated on the assumptions that: (1) the data must be copied from SSD to DRAM for the DBMS to operate on it, and that (2) storage is orders of magnitude slower than DRAM [4, 16]. But emerging non-volatile memory (NVM) technologies upend these design assumptions. They support low latency reads and writes comparable to DRAM, but with persistent writes and large storage capacity like an SSD. In a DRAM-SSD hierarchy, the buffer manager decides *what* pages to move between disk and memory and *when* to move them. However, with a DRAM-NVM-SSD hierarchy, in addition to these decisions, it must also decide *where* to move them (*i.e.*, which storage tier).

Prior Work. Recently, researchers have designed Hymem, a novel three-tier buffer manager for a DRAM-NVM-SSD hierarchy [37]. Hymem employs a set of optimizations tailored for NVM. It adopts a *data migration policy* consisting of four decisions: DRAM admission, DRAM eviction, NVM admission, and NVM eviction. ❶ Initially, a newly-allocated 16 KB page resides on SSD. When a transaction requests that page, Hymem *eagerly* admits the entire page to DRAM. ❷ DRAM eviction is the next decision that it takes to reclaim space. It uses the CLOCK algorithm for picking the victim page [34]. ❸ Next, it must decide whether that page must be admitted to the NVM buffer (if it is not already present in that buffer). Hymem seeks to identify *warm* pages. It maintains a queue of recently considered pages to make the NVM admission decision. It admits pages that were recently denied admission. Each time a page is considered for admission, Hymem checks whether the page is in the admission queue. If so, it is removed from the queue and admitted into the NVM buffer. Otherwise, it is added to the queue and directly moved to SSD, thereby bypassing the NVM buffer. ❹ Lastly, it uses the CLOCK algorithm for evicting a page from the NVM buffer.

Hymem supports cache-line-grained loading to improve the utilization of NVM bandwidth. Unlike SSD, NVM supports low-latency access to 256 B blocks. Hymem uses cache line-grained loading to extract only the hot data objects from an otherwise cold 16 KB page. By only loading those cache lines that are needed, Hymem lowers its bandwidth consumption.

Hymem supports a *mini page* layout for reducing the DRAM footprint of cache-line-grained pages. This layout allows it to efficiently keep track of which cachelines are loaded. When the mini page overflows (*i.e.*, all sixteen cache lines are loaded), Hymem transparently promotes it to a *full page*.

Limitations. These optimizations enable Hymem to work well across different workloads on a DRAM-NVM-SSD storage hierarchy. However, it suffers from two limitations. First, it is a single-threaded buffer manager. Second, it is evaluated on an NVM emulation platform. These limitations constrain the utility of the insights obtained using Hymem (§6.5). In particular, the data migration policy employed in Hymem is not the optimal one for certain workloads. We

SIGMOD '21, June 20–25, 2021, Virtual Event, China

ACM ISBN 978-1-4503-8343-1/21/06…$15.00
https://doi.org/10.1145/3448016.3452819

SIGMOD 2021

Managing Non-Volatile Memory in Database Systems

Alexander van Renen
Technische Universität München
renen@in.tum.de

Viktor Leis
Technische Universität München
leis@in.tum.de

Alfons Kemper
Technische Universität München
kemper@in.tum.de

Thomas Neumann
Technische Universität München
neumann@in.tum.de

Takushi Hashida
Fujitsu Laboratories
hashida.takushi@jp.fujitsu.com

Kazuichi Oe
Fujitsu Laboratories
ooe.kazuichi@jp.fujitsu.com

Yoshiyasu Doi
Fujitsu Laboratories
yosh-d@jp.fujitsu.com

Lilian Harada
Fujitsu Laboratories
harada.lilian@jp.fujitsu.com

Mitsuru Sato
Fujitsu Laboratories
msato@jp.fujitsu.com

ABSTRACT

Non-volatile memory (NVM) is a new storage technology that combines the performance and byte addressability of DRAM with the persistence of traditional storage devices like flash (SSD). While these properties make NVM highly promising, it is not yet clear how to best integrate NVM into the storage layer of modern database systems. Two system designs have been proposed. The first is to use NVM exclusively, i.e., to store all data and index structures on it. However, because NVM has a higher latency than DRAM, this design can be less efficient than main-memory database systems. For this reason, the second approach uses a page-based DRAM cache in front of NVM. This approach, however, does not utilize the byte addressability of NVM and, as a result, accessing an uncached tuple on NVM requires retrieving an entire page.

In this work, we evaluate these two approaches and compare them with in-memory databases as well as more traditional buffer managers that use main memory as a cache in front of SSDs. This allows us to determine how much performance gain can be expected from NVM. We also propose a lightweight storage manager that simultaneously supports DRAM, NVM, and flash. Our design utilizes the byte addressability of NVM and uses it as an additional caching layer that improves performance without losing the benefits from the even faster DRAM and the large capacities of SSDs.

ACM Reference Format:
Alexander van Renen, Viktor Leis, Alfons Kemper, Thomas Neumann, Takushi Hashida, Kazuichi Oe, Yoshiyasu Doi, Lilian Harada, and Mitsuru Sato. 2018. Managing Non-Volatile Memory in Database Systems. In *SIGMOD'18: 2018 International Conference on Management of Data, June 10–15, 2018, Houston, TX, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3183713.3196897

SIGMOD'18, June 10–15, 2018, Houston, TX, USA

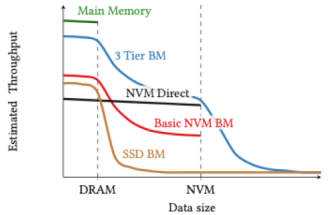


Figure 1: System designs under varying data sizes.

1 INTRODUCTION

Non-volatile memory (NVM), also known as Storage Class Memory (SCM) and NVRAM, is a radically new and highly promising storage device. Technologies like PCM, STT-RAM, and ReRAM have slightly different features [35], but generally combine the byte addressability of DRAM with the persistence of storage technologies like SSD (flash). Because commercial products are not yet available, the precise characteristics, price, and capacity features of NVM have not been publicly disclosed (and like all prior NVM research, we have to resort to simulation for experiments). What is known, however, is that for the foreseeable future, NVM will be slower (and larger) than DRAM and, at the same time, much faster (but smaller) than SSD [13]. Furthermore, NVM has an asymmetric read/write latency—making writes much more expensive than reads. Given these characteristics, we consider it unlikely that NVM can replace DRAM or SSD outright.

While the novel properties of NVM make it particularly relevant for database systems, they also present new architectural challenges. Neither the traditional disk-based architecture nor modern main-memory systems can fully utilize NVM without major changes to their designs. The two components most affected by NVM are logging/recovery and storage. Much of the recent research on NVM has optimized logging and recovery [5, 16, 22, 36, 45]. In this work, we instead focus on the storage/caching aspect, i.e., on dynamically deciding where data should reside (DRAM, NVM, or SSD).

SIGMOD 2018

Agenda

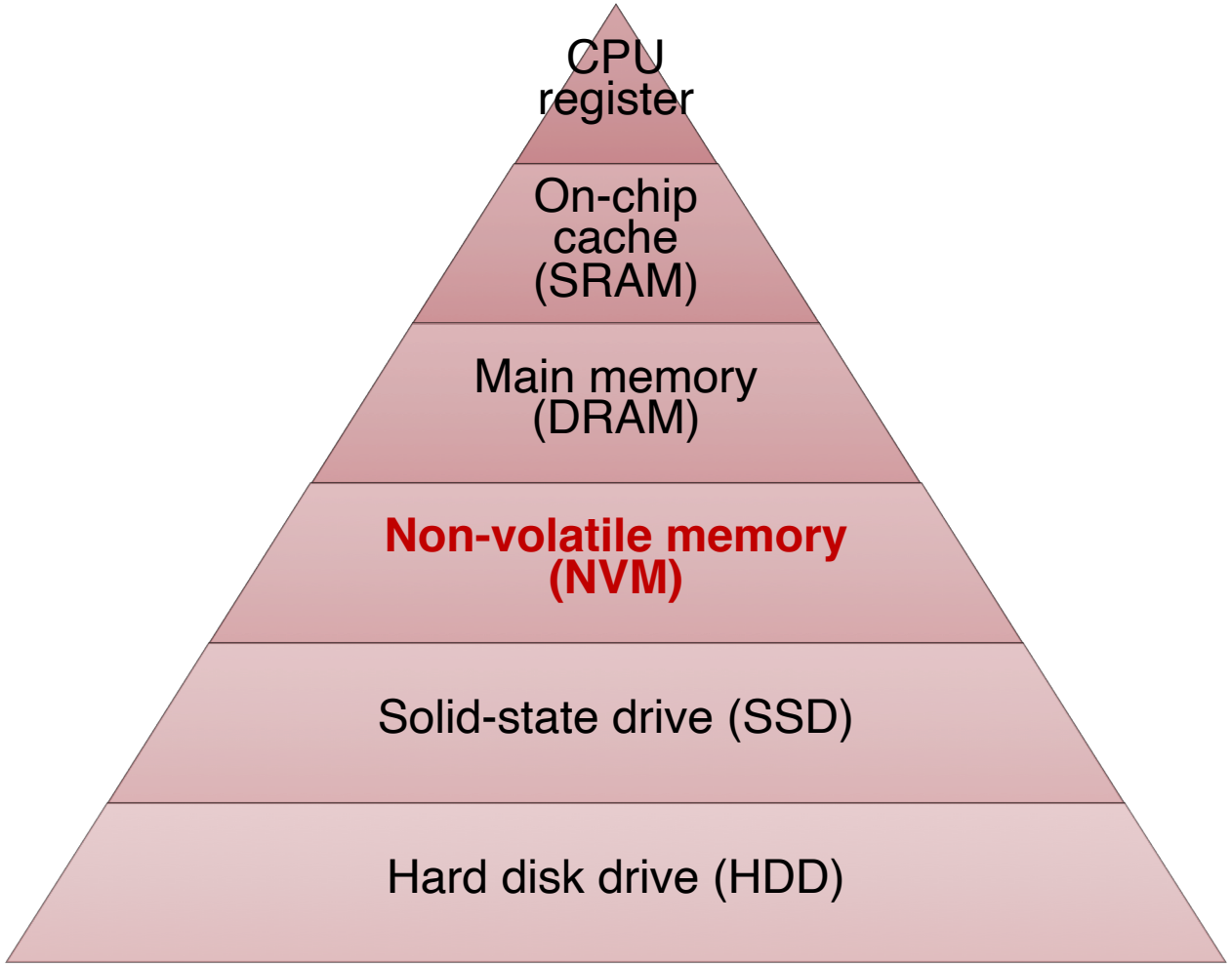
NVM Basics

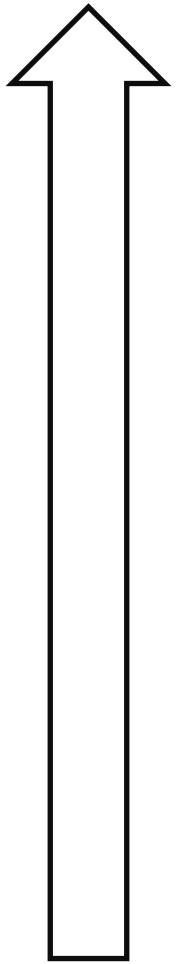
HYMEM Design (SIGMOD'18)

Spitfire Design (SIGMOD'20)

NVM Basics

Storage Hierarchy





Lower latency

Higher bandwidth

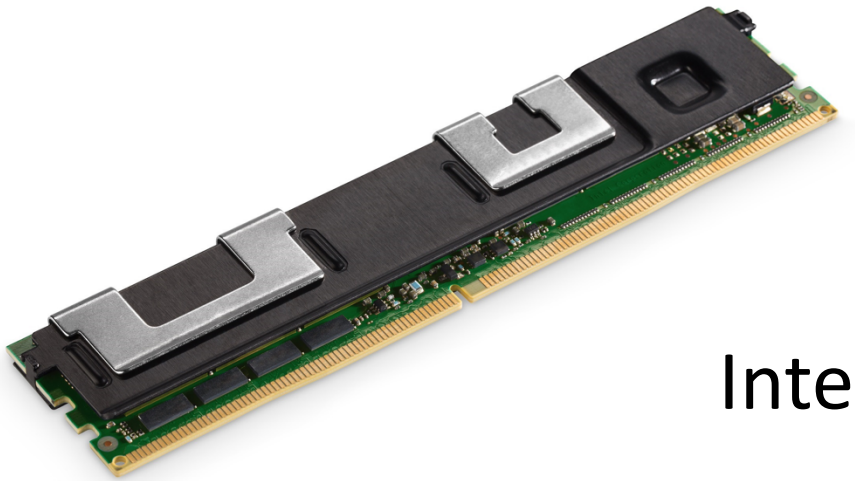
Smaller capacity

Higher cost

More energy

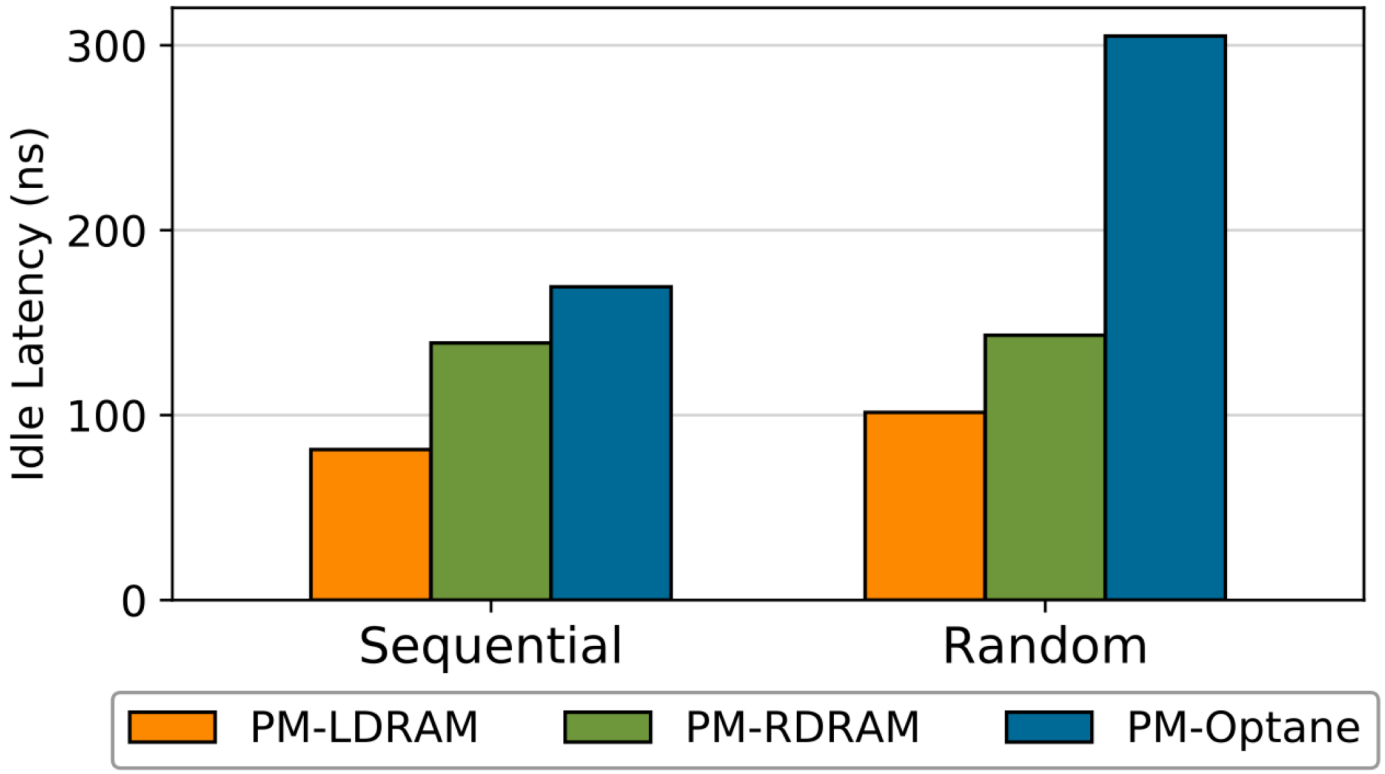
DRAM vs. NVM vs. SSD

	DRAM	NVM	SSD/HDD
Access granularity	**Byte addressable**	**Byte addressable**	Block storage
Durability	Volatile	**Non-volatile**	**Non-volatile**



Intel® Optane™ Memory

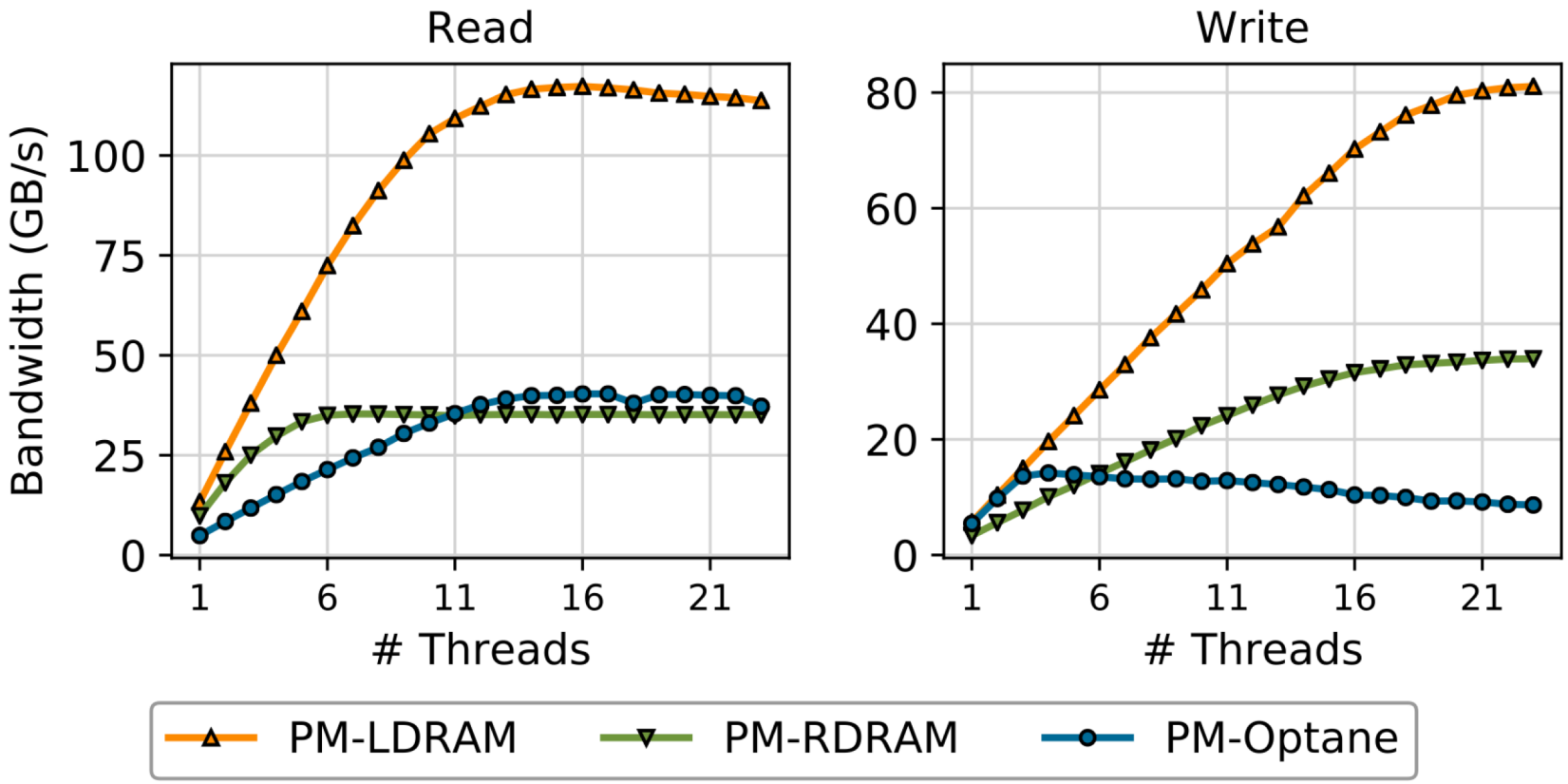
NVM Performance – Read Latency



Random read latency is 305 ns which is 3x slower than local DRAM
Sequential read latency is 2x better than random read latency

* Figure from *Basic Performance Measurements of the Intel Optane DC Persistent Memory Module*

NVM Performance – Bandwidth



Per-DIMM max read bandwidth: 6.6 GB/s, max write bandwidth: 2.3 GB/s
Read/write bandwidth gap is 2.9x for NVM (1.3x for DRAM)

* Figure from *Basic Performance Measurements of the Intel Optane DC Persistent Memory Module*

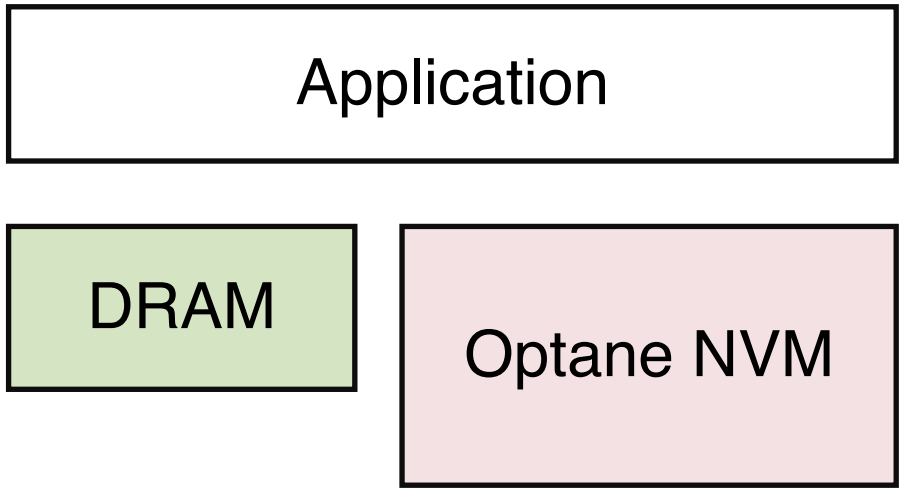
NVM Performance – Cost

Type	Density	Price	$/GB
DRAM	32GB	$374.71	$11.71
DRAM	64GB	$708.25	$11.07
DRAM	128GB	$1,913.21	$14.95
DRAM	256GB	$5,952.00	$23.25
Optane	**128GB**	**$577.00**	**$4.51**
Optane	**256GB**	**$2,125.00**	**$8.30**
Optane	**512GB**	**$6,751.00**	**$13.19**

Data from https://thememoryguy.com/intels-optane-dimm-price-model/

Operation Modes

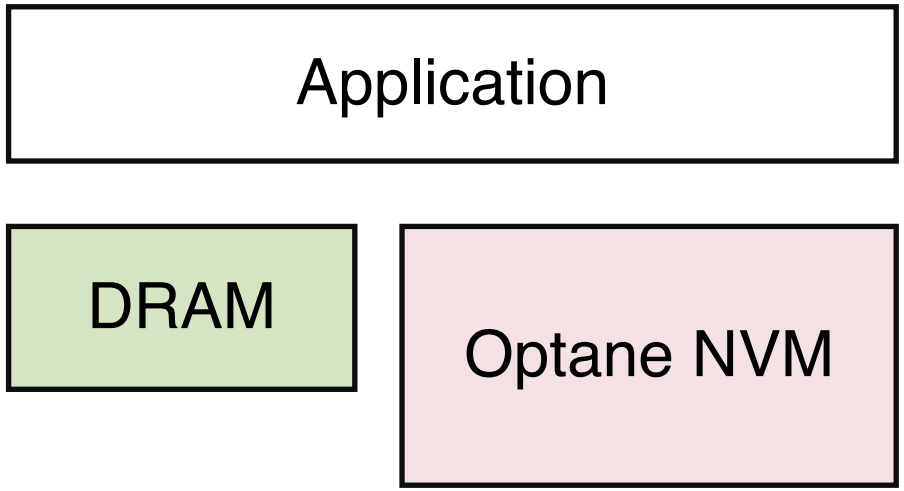
App direct mode



DRAM and NVM in different address space

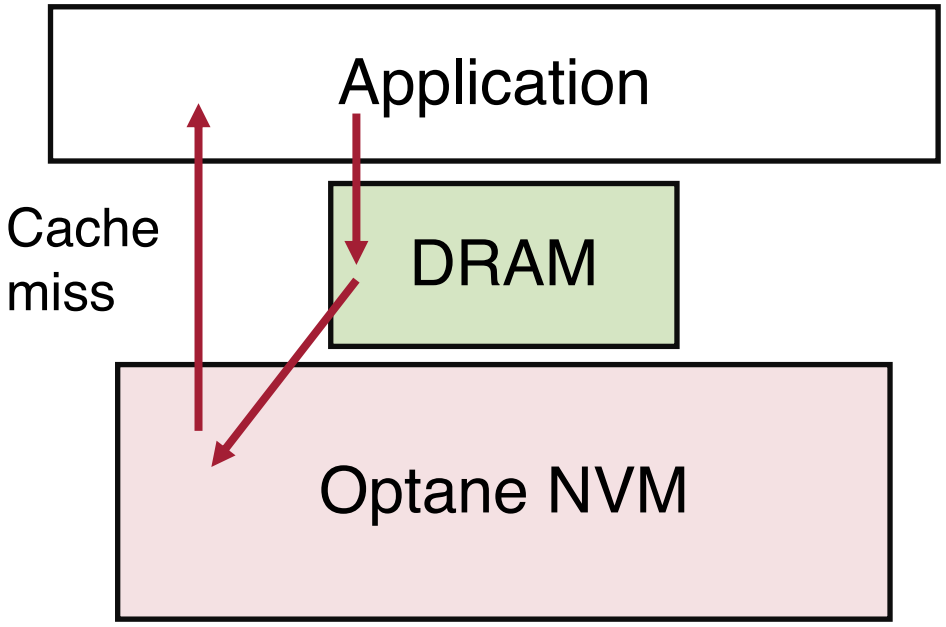
Operation Modes

App direct mode



DRAM and NVM in different address space

Memory Mode



DRAM as a cache managed by hardware

Buffer Management in Disk vs. NVM

Buffer management in SSD/HDD

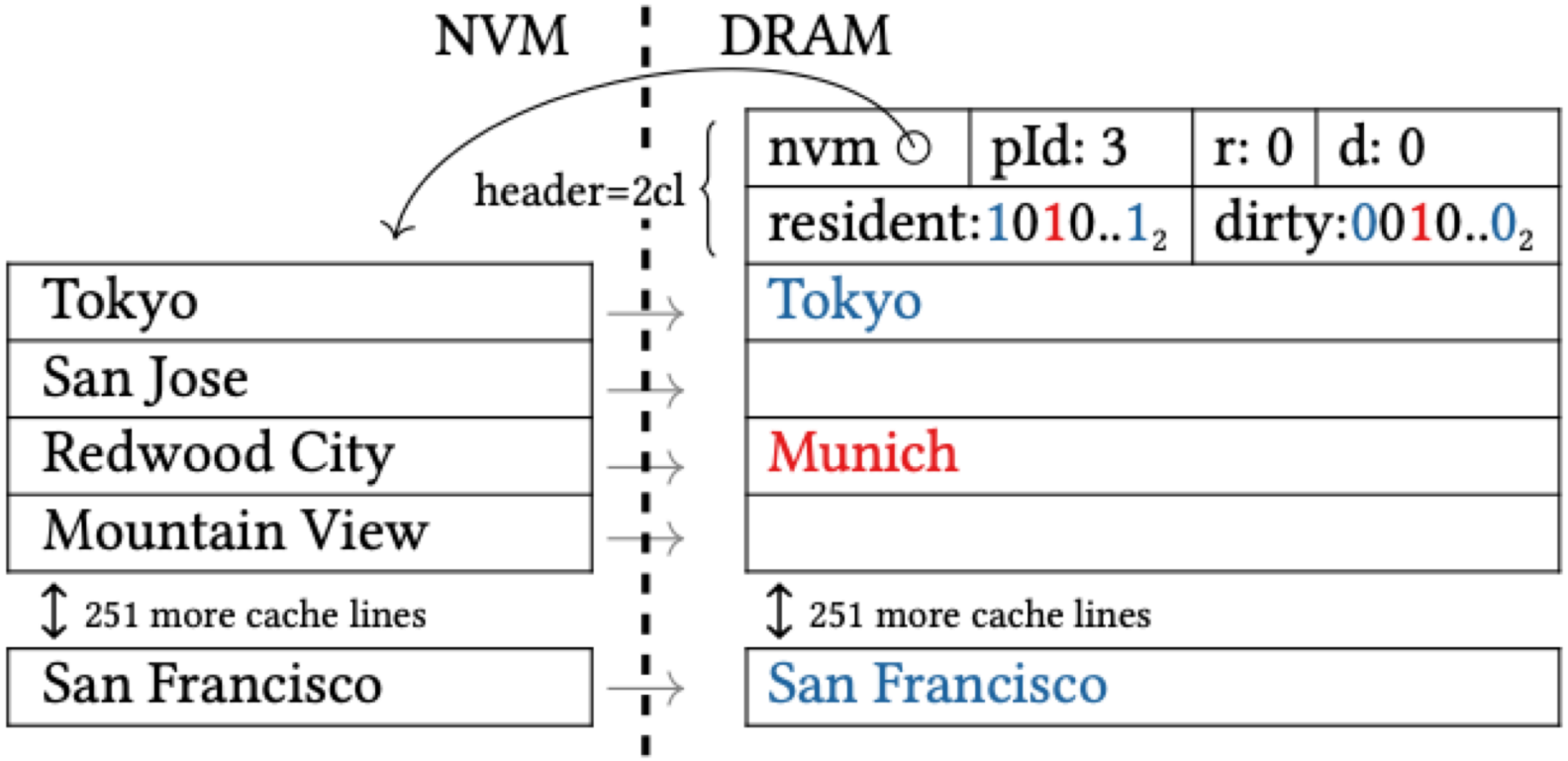
- Block storage
- Load a full page at a time (e.g., 16 KB)

Buffer management in NVM

- Byte addressable
- Waste of bandwidth if full pages are loaded
- Loading a cacheline at a time (64 B)

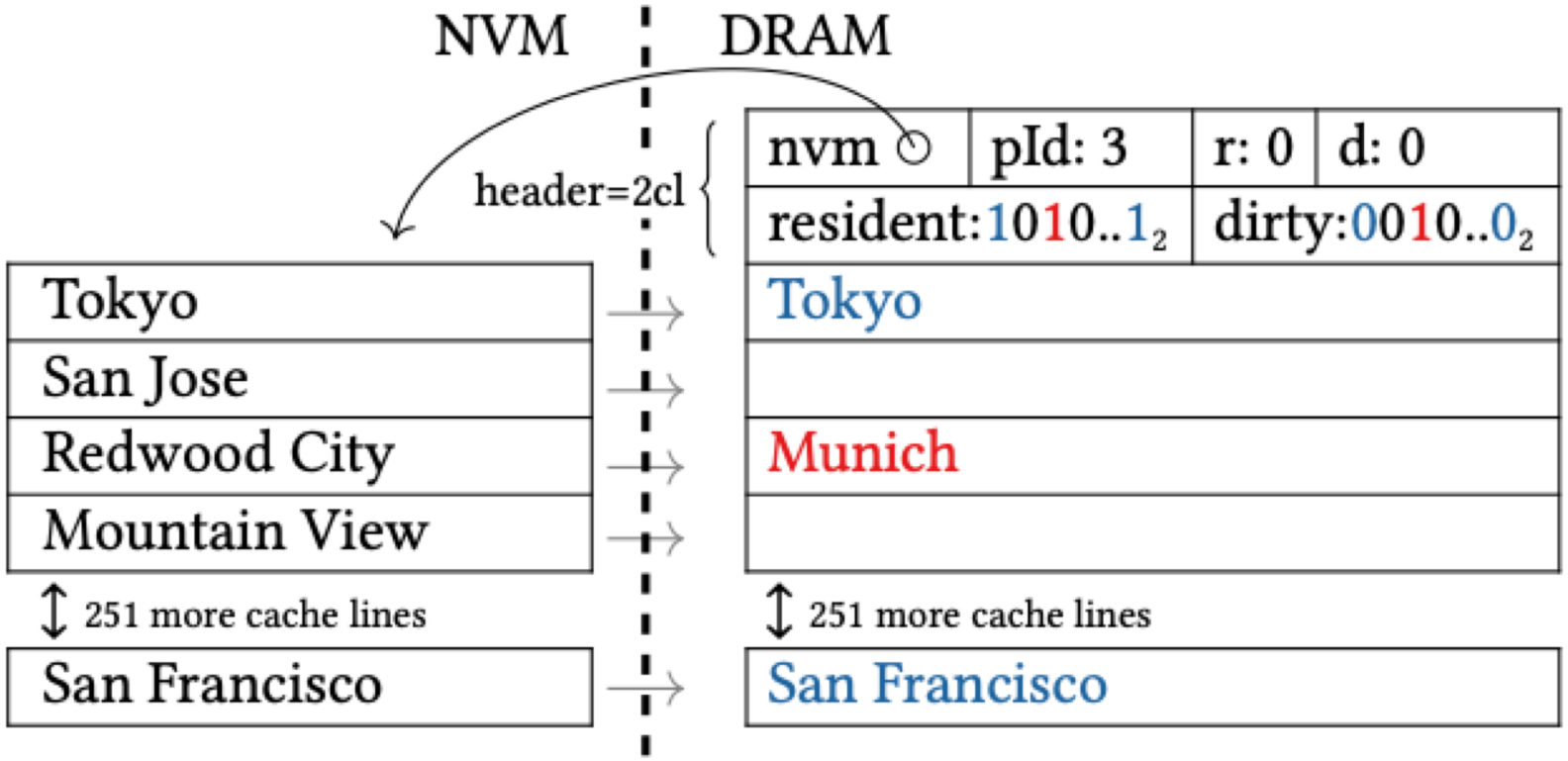
HYMEM Design

Cache-Line-Grained Pages



Page initially empty, cachelines loaded as they are accessed

Cache-Line-Grained Pages



Page initially empty, cachelines loaded as they are accessed

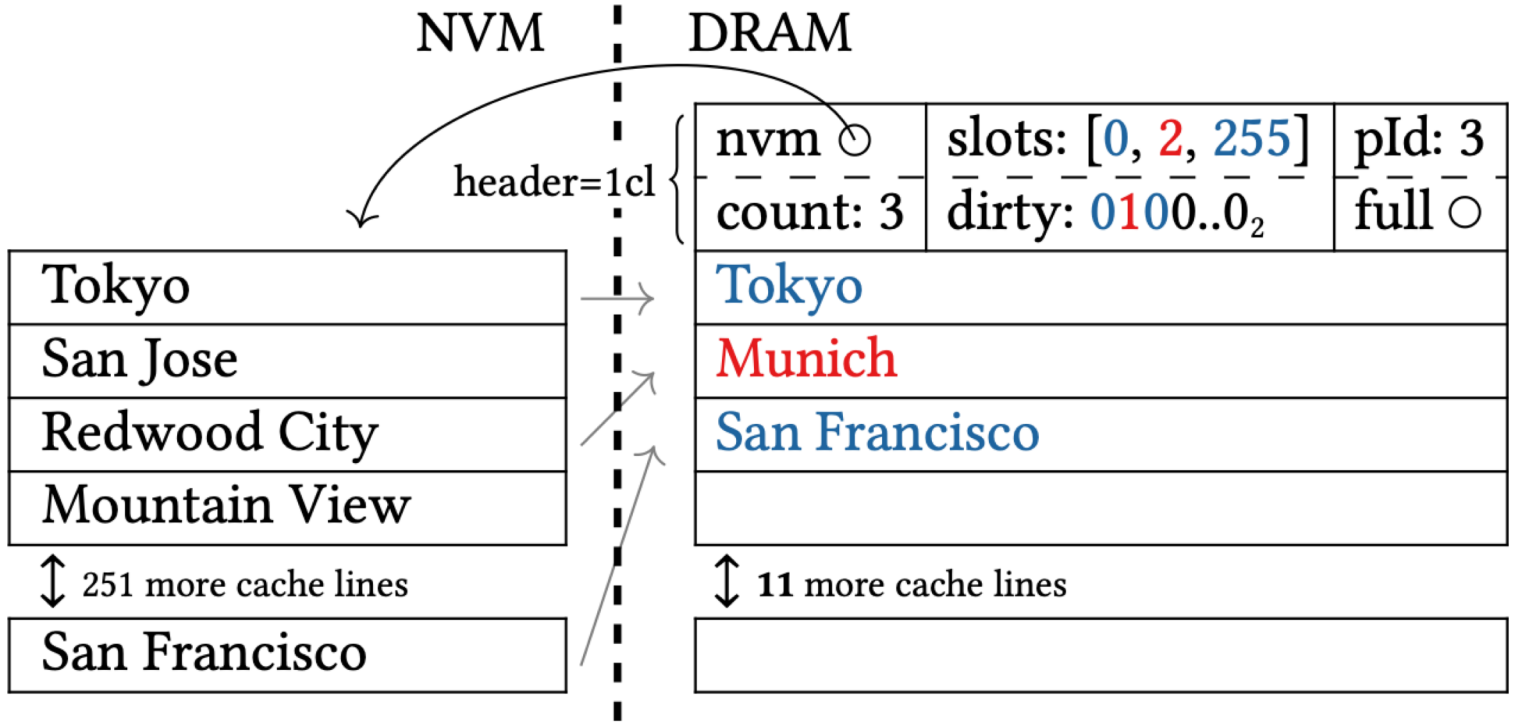
Overhead: each access checks/updates `resident/dirty` bits

Mini Pages

Page layout consumes more DRAM space than necessary

Mini Pages

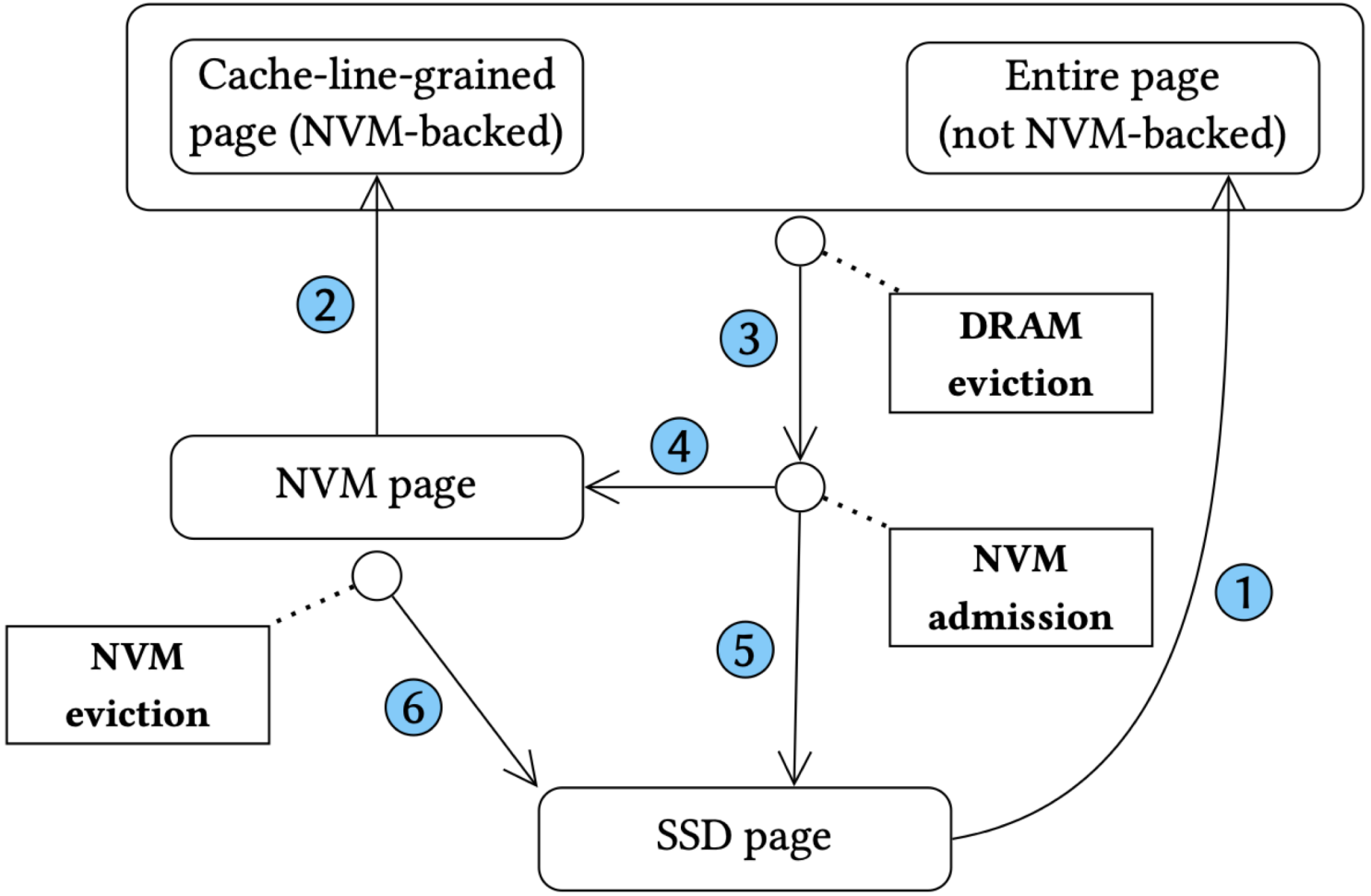
Page layout consumes more DRAM space than necessary



Mini page: a sparse representation of a page

- Cachelines are sorted
- Promote to full page when a mini-page is full

Three-Tier Buffer Management



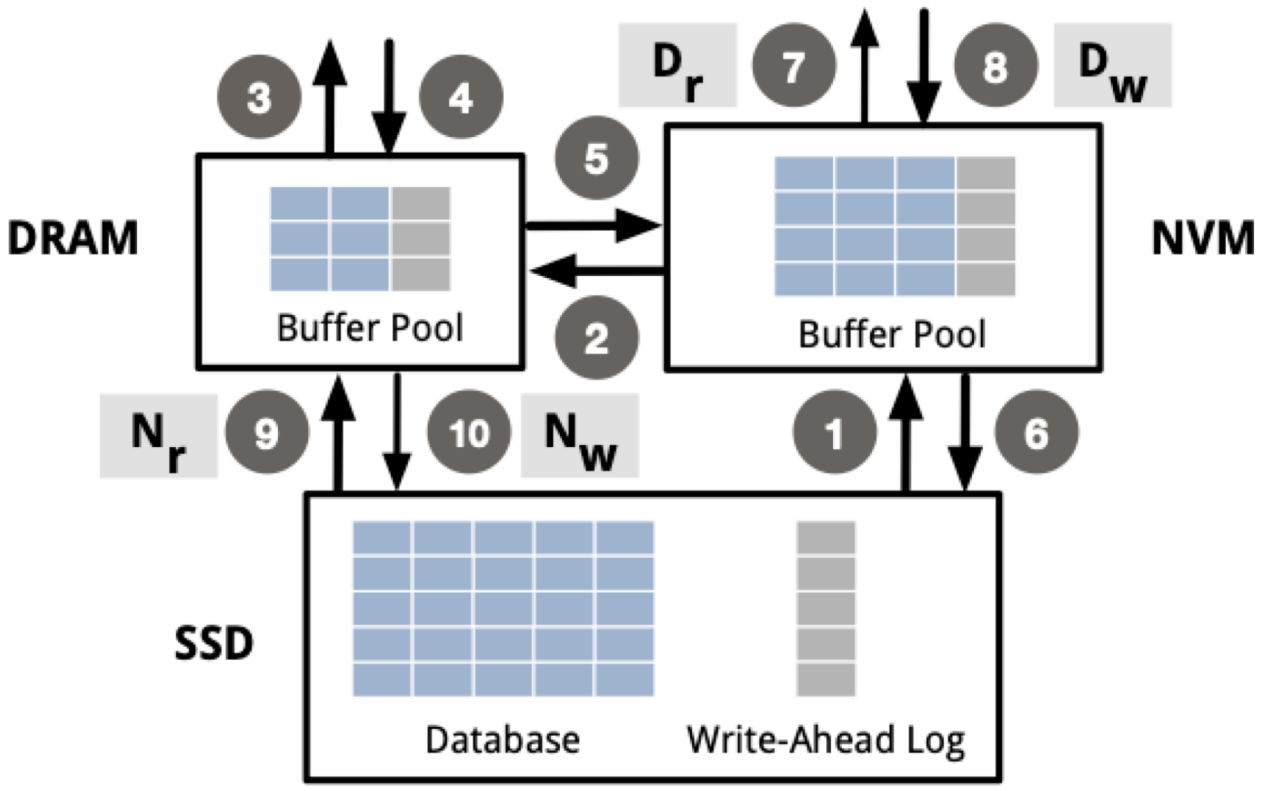
1. DRAM & NVM miss
   - Directly load to DRAM
2. NVM hit
   - Load cache-line-granted page to DRAM
3. DRAM eviction
   - Clock (second-chance)

4&5. NVM admission
   - Admission set (second-chance)

6. NVM eviction
   - Clock

Spitfire Design

Data Migration



- Bypass DRAM during reads
  - Access NVM directly without admitting the page into DRAM
- Bypass DRAM during writes
  - Log and checkpoint not cached in DRAM
- Bypass NVM during reads
  - HYMEM does this by default
- Bypass NVM during writes
  - DRAM evictions go to NVM with a probability
  - Simpler than HYMEM design

Novelty

CPU can directly read from NVM, bypassing DRAM

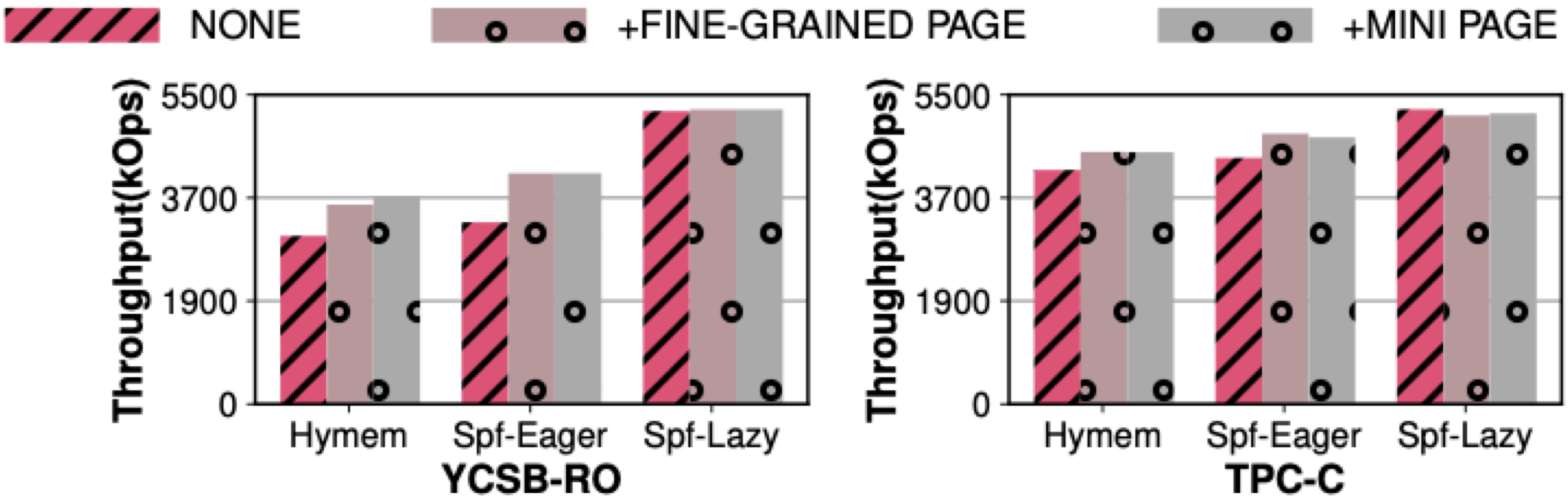
Probabilistic admission and replacement

Dynamically tuning of the probabilities

Policy	$\mathcal{D}_r$	$\mathcal{D}_w$	$\mathcal{N}_r$	$\mathcal{N}_w$
HYMEM [37]	1	1	0	AdmQueue
SPITFIRE-Eager	1	1	1	1
SPITFIRE-Lazy	0.01	0.01	0.2	1

Table 3: Migration Policies: List of policies used in the ablation study.

Evaluation


NONE
+FINE-GRAINED PAGE
+MINI PAGE
Throughput(kOps)
5500
3700
1900
0
Hymem
Spf-Eager
Spf-Lazy
YCSB-RO
Throughput(kOps)
5500
3700
1900
0
Hymem
Spf-Eager
Spf-Lazy
TPC-C

Q/A – Buffer Management with NVM

Need generalized algorithm for growing storage hierarchy?

- 2-tier -> 3-tier -> N-tier?

Disadvantage of probabilistic approach?

Disadvantage of self adapting/tuning?

Can we entirely replace SSD by NVM?

Do modern DBMS run on 3-tier buffer memories?

More baselines?

What is NUMA effect?

Group Discussion

Non-volatile memory (NVM) have (1) bandwidth and latency close to DRAM and (2) byte-addressability. How do NVM devices change buffer management in a DBMS?

- Inclusive caching?
- Is page the right management granularity?
- What if we treat NVM as a different tier of memory?

Before Next Lecture

Submit review for

Patricia G. Selinger, et al., Access Path Selection in a Relational Database Management System. SIGMOD, 1979