Operating System Support for Multi-Tiered Memory
(A Case Study of Intel's Optane DCPMM)

Jeongseob Ahn

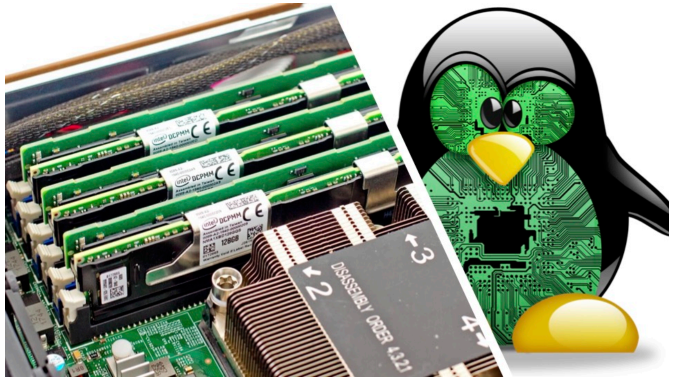
E-mail: jsahn@ajou.ac.kr
Web: https://jeongseob.github.io

AJOU UNIVERSITY

In this talk…

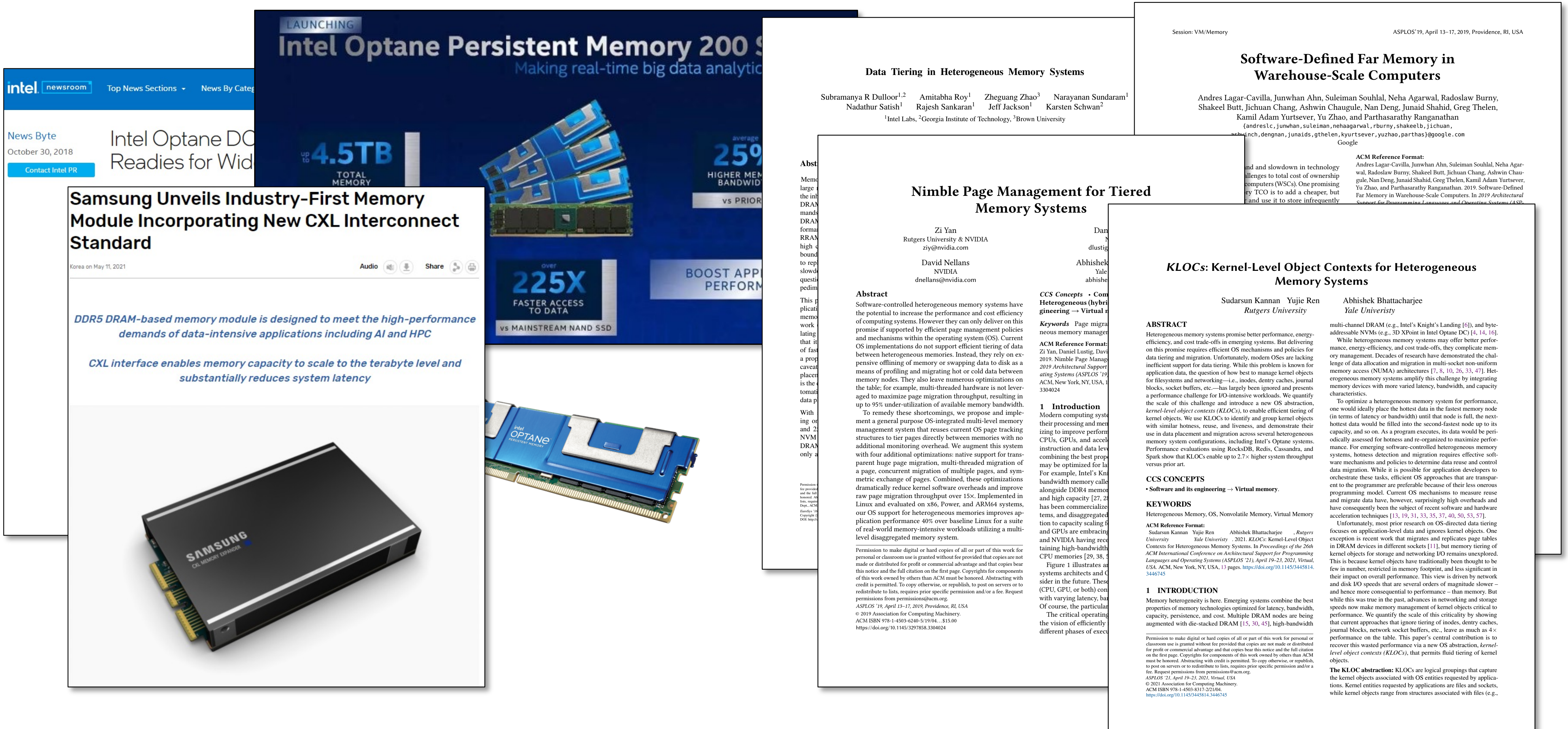
- Exploring the Design Space of Page Management for Multi-Tiered Memory Systems
  - Jonghyeon Kim, Wonkyo Choe, and Jeongseob Ahn
  - USENIX Annual Technical Conference (ATC), July 2021

- A Study of Memory Placement on Hardware-assisted Tiered Memory Systems
  - Wonkyo Choe, Jonghyeon Kim, and Jeongseob Ahn
  - IEEE Computer Architecture Letters (CAL), 19(2), July-December 2020



Artifact available at https://github.com/csl-ajou/AutoTiering

Large-scale memory systems



intel. newsroom Top News Sections News By Categ

News Byte
October 30, 2018
Contact Intel PR

Intel Optane DC
Readies for Wid

LAUNCHING
Intel Optane Persistent Memory 200 S
Making real-time big data analytic

up to 4.5TB TOTAL MEMORY

average 25% HIGHER MEM BANDWID vs PRIOR

over 225X FASTER ACCESS TO DATA vs MAINSTREAM NAND SSD

BOOST APP PERFORM

intel OPTANE PERSISTENT MEMORY

Samsung Unveils Industry-First Memory Module Incorporating New CXL Interconnect Standard

Korea on May 11, 2021

Audio Share

DDR5 DRAM-based memory module is designed to meet the high-performance demands of data-intensive applications including AI and HPC

CXL interface enables memory capacity to scale to the terabyte level and substantially reduces system latency

Data Tiering in Heterogeneous Memory Systems

Subramanya R Dulloor[1,2] Amitabha Roy[1] Zheguang Zhao[3] Narayanan Sundaram[1]
Nadathur Satish[1] Rajesh Sankaran[1] Jeff Jackson[1] Karsten Schwan[2]

[1]Intel Labs, [2]Georgia Institute of Technology, [3]Brown University

Abst

Session: VM/Memory ASPLOS'19, April 13–17, 2019, Providence, RI, USA

Software-Defined Far Memory in Warehouse-Scale Computers

Andres Lagar-Cavilla, Junwhan Ahn, Suleiman Souhlal, Neha Agarwal, Radoslaw Burny, Shakeel Butt, Jichuan Chang, Ashwin Chaugule, Nan Deng, Junaid Shahid, Greg Thelen, Kamil Adam Yurtsever, Yu Zhao, and Parthasarathy Ranganathan
{andreslc,junwhan,suleiman,nehaagarwal,rburny,shakeelb,jichuan,
ashwinch,dengnan,junaids,gthelen,kyurtsever,yuzhao,parthas}@google.com
Google

nd and slowdown in technology allenges to total cost of ownership computers (WSCs). One promising ry TCO is to add a cheaper, but r and use it to store infrequently

ACM Reference Format:
Andres Lagar-Cavilla, Junwhan Ahn, Suleiman Souhlal, Neha Agarwal, Radoslaw Burny, Shakeel Butt, Jichuan Chang, Ashwin Chaugule, Nan Deng, Junaid Shahid, Greg Thelen, Kamil Adam Yurtsever, Yu Zhao, and Parthasarathy Ranganathan. 2019. Software-Defined Far Memory in Warehouse-Scale Computers. In *2019 Architectural Support for Programming Languages and Operating Systems (ASP-*

Nimble Page Management for Tiered Memory Systems

Zi Yan
Rutgers University & NVIDIA
ziy@nvidia.com

David Nellans
NVIDIA
dnellans@nvidia.com

Dan
N
dlustig

Abhishek
Yale
abhishe

Abstract

Software-controlled heterogeneous memory systems have the potential to increase the performance and cost efficiency of computing systems. However they can only deliver on this promise if supported by efficient page management policies and mechanisms within the operating system (OS). Current OS implementations do not support efficient tiering of data between heterogeneous memories. Instead, they rely on expensive offlining of memory or swapping data to disk as a means of profiling and migrating hot or cold data between memory nodes. They also leave numerous optimizations on the table; for example, multi-threaded hardware is not leveraged to maximize page migration throughput, resulting in up to 95% under-utilization of available memory bandwidth.

To remedy these shortcomings, we propose and implement a general purpose OS-integrated multi-level memory management system that reuses current OS page tracking structures to tier pages directly between memories with no additional monitoring overhead. We augment this system with four additional optimizations: native support for transparent huge page migration, multi-threaded migration of a page, concurrent migration of multiple pages, and symmetric exchange of pages. Combined, these optimizations dramatically reduce kernel software overheads and improve raw page migration throughput over 15×. Implemented in Linux and evaluated on x86, Power, and ARM64 systems, our OS support for heterogeneous memories improves application performance 40% over baseline Linux for a suite of real-world memory-intensive workloads utilizing a multi-level disaggregated memory system.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ASPLOS '19, April 13–17, 2019, Providence, RI, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6240-5/19/04…$15.00
https://doi.org/10.1145/3297858.3304024

CCS Concepts • **Com Heterogeneous (hybri gineering → Virtual r**

Keywords Page migra neous memory managem

ACM Reference Format:
Zi Yan, Daniel Lustig, Davi 2019. Nimble Page Manag *2019 Architectural Support ating Systems (ASPLOS '19)* ACM, New York, NY, USA, 1 3304024

1 Introduction

Modern computing syste their processing and men izing to improve perform CPUs, GPUs, and accel instruction and data leve combining the best prop may be optimized for la For example, Intel's Kni bandwidth memory calle alongside DDR4 memor and high capacity [27, 2 has been commercialize tems, and disaggregated tion to capacity scaling f and GPUs are embracing and NVIDIA having rec taining high-bandwidth CPU memories [29, 38, 5

Figure 1 illustrates a systems architects and C sider in the future. These (CPU, GPU, or both) con with varying latency, ba Of course, the particular

The critical operating the vision of efficiently different phases of execu

***KLOCs*: Kernel-Level Object Contexts for Heterogeneous Memory Systems**

Sudarsun Kannan Yujie Ren
Rutgers University

Abhishek Bhattacharjee
Yale Univeristy

ABSTRACT

Heterogeneous memory systems promise better performance, energy-efficiency, and cost trade-offs in emerging systems. But delivering on this promise requires efficient OS mechanisms and policies for data tiering and migration. Unfortunately, modern OSes are lacking inefficient support for data tiering. While this problem is known for application data, the question of how best to manage kernel objects for filesystems and networking—i.e., inodes, dentry caches, journal blocks, socket buffers, etc.—has largely been ignored and presents a performance challenge for I/O-intensive workloads. We quantify the scale of this challenge and introduce a new OS abstraction, *kernel-level object contexts (KLOCs)*, to enable efficient tiering of kernel objects. We use KLOCs to identify and group kernel objects with similar hotness, reuse, and liveness, and demonstrate their use in data placement and migration across several heterogeneous memory system configurations, including Intel's Optane systems. Performance evaluations using RocksDB, Redis, Cassandra, and Spark show that KLOCs enable up to 2.7× higher system throughput versus prior art.

CCS CONCEPTS

• **Software and its engineering → Virtual memory**.

KEYWORDS

Heterogeneous Memory, OS, Nonvolatile Memory, Virtual Memory

ACM Reference Format:
Sudarsun Kannan Yujie Ren Abhishek Bhattacharjee , *Rutgers University* *Yale Univeristy* . 2021. *KLOCs*: Kernel-Level Object Contexts for Heterogeneous Memory Systems. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '21), April 19–23, 2021, Virtual, USA*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3445814.3446745

1 INTRODUCTION

Memory heterogeneity is here. Emerging systems combine the best properties of memory technologies optimized for latency, bandwidth, capacity, persistence, and cost. Multiple DRAM nodes are being augmented with die-stacked DRAM [15, 30, 45], high-bandwidth multi-channel DRAM (e.g., Intel's Knight's Landing [6]), and byte-addressable NVMs (e.g., 3D XPoint in Intel Optane DC) [4, 14, 16].

While heterogeneous memory systems may offer better performance, energy-efficiency, and cost trade-offs, they complicate memory management. Decades of research have demonstrated the challenge of data allocation and migration in multi-socket non-uniform memory access (NUMA) architectures [7, 8, 10, 26, 33, 47]. Heterogeneous memory systems amplify this challenge by integrating memory devices with more varied latency, bandwidth, and capacity characteristics.

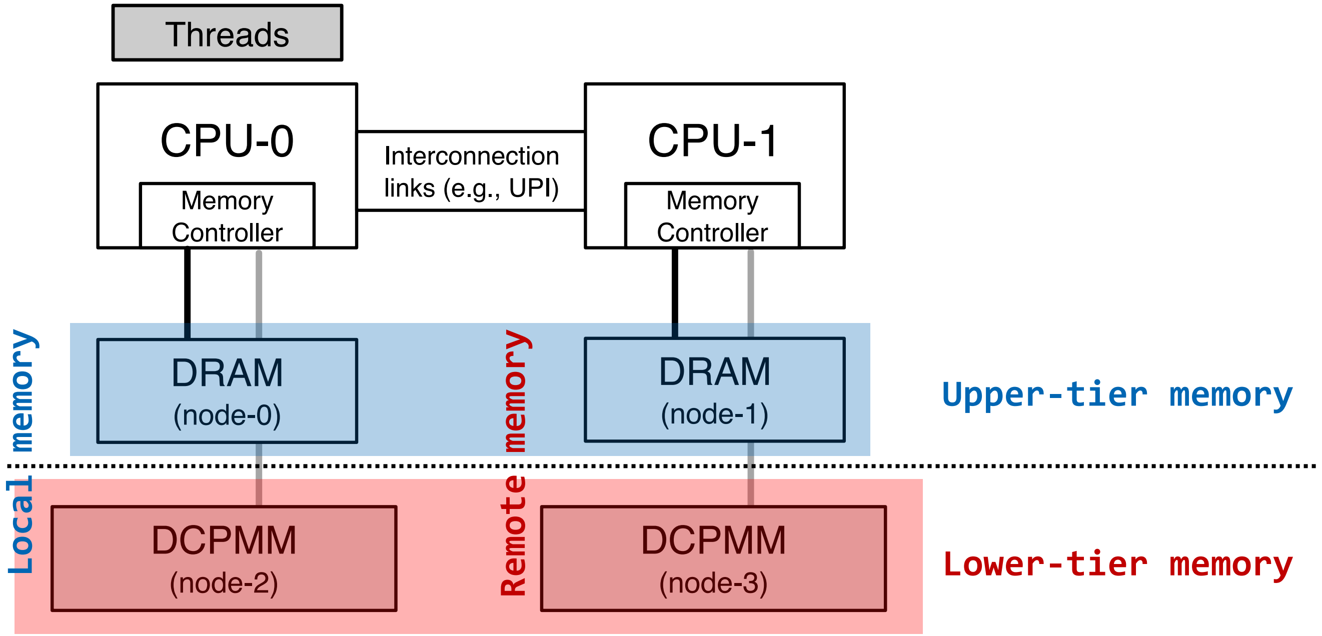
To optimize a heterogeneous memory system for performance, one would ideally place the hottest data in the fastest memory node (in terms of latency or bandwidth) until that node is full, the next-hottest data would be filled into the second-fastest node up to its capacity, and so on. As a program executes, its data would be periodically assessed for hotness and re-organized to maximize performance. For emerging software-controlled heterogeneous memory systems, hotness detection and migration requires effective software mechanisms and policies to determine data reuse and control data migration. While it is possible for application developers to orchestrate these tasks, efficient OS approaches that are transparent to the programmer are preferable because of their less onerous programming model. Current OS mechanisms to measure reuse and migrate data have, however, surprisingly high overheads and have consequently been the subject of recent software and hardware acceleration techniques [13, 19, 31, 33, 35, 37, 40, 50, 53, 57].

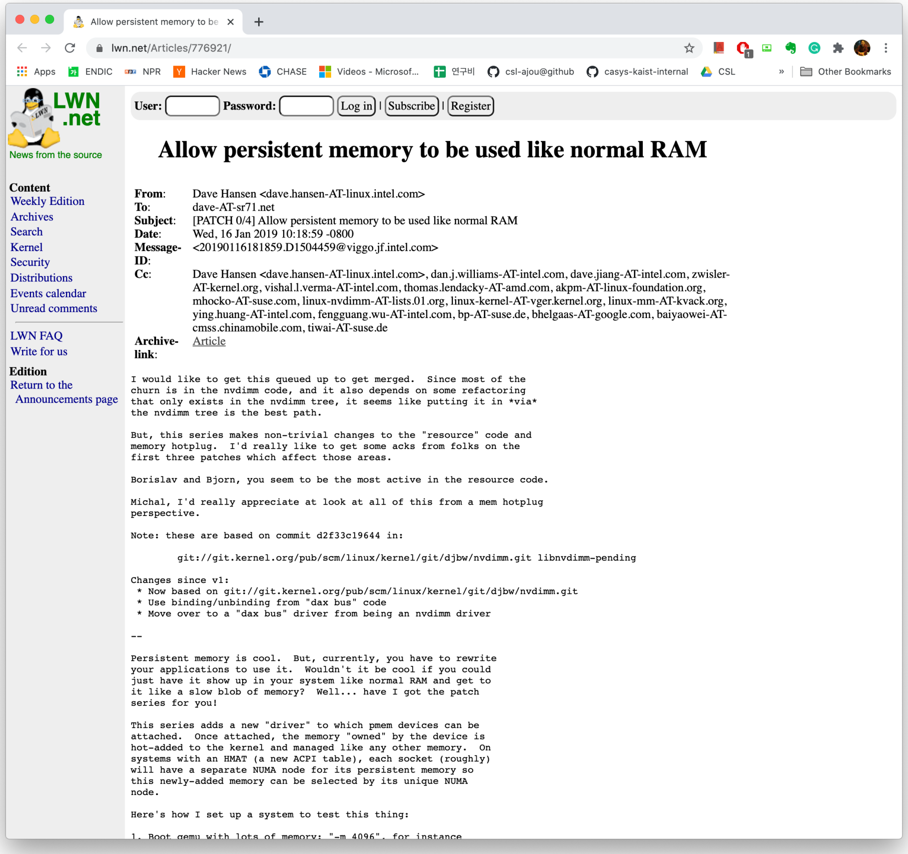
Unfortunately, most prior research on OS-directed data tiering focuses on application-level data and ignores kernel objects. One exception is recent work that migrates and replicates page tables in DRAM devices in different sockets [11], but memory tiering of kernel objects for storage and networking I/O remains unexplored. This is because kernel objects have traditionally been thought to be few in number, restricted in memory footprint, and less significant in their impact on overall performance. This view is driven by network and disk I/O speeds that are several orders of magnitude slower – and hence more consequential to performance – than memory. But while this was true in the past, advances in networking and storage speeds now make memory management of kernel objects critical to performance. We quantify the scale of this criticality by showing that current approaches that ignore tiering of inodes, dentry caches, journal blocks, network socket buffers, etc., leave as much as 4× performance on the table. This paper's central contribution is to recover this wasted performance via a new OS abstraction, *kernel-level object contexts (KLOCs)*, that permits fluid tiering of kernel objects.

The KLOC abstraction: KLOCs are logical groupings that capture the kernel objects associated with OS entities requested by applications. Kernel entities requested by applications are files and sockets, while kernel objects range from structures associated with files (e.g.,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ASPLOS '21, April 19–23, 2021, Virtual, USA
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8317-2/21/04.
https://doi.org/10.1145/3445814.3446745

A multi-tiered memory system



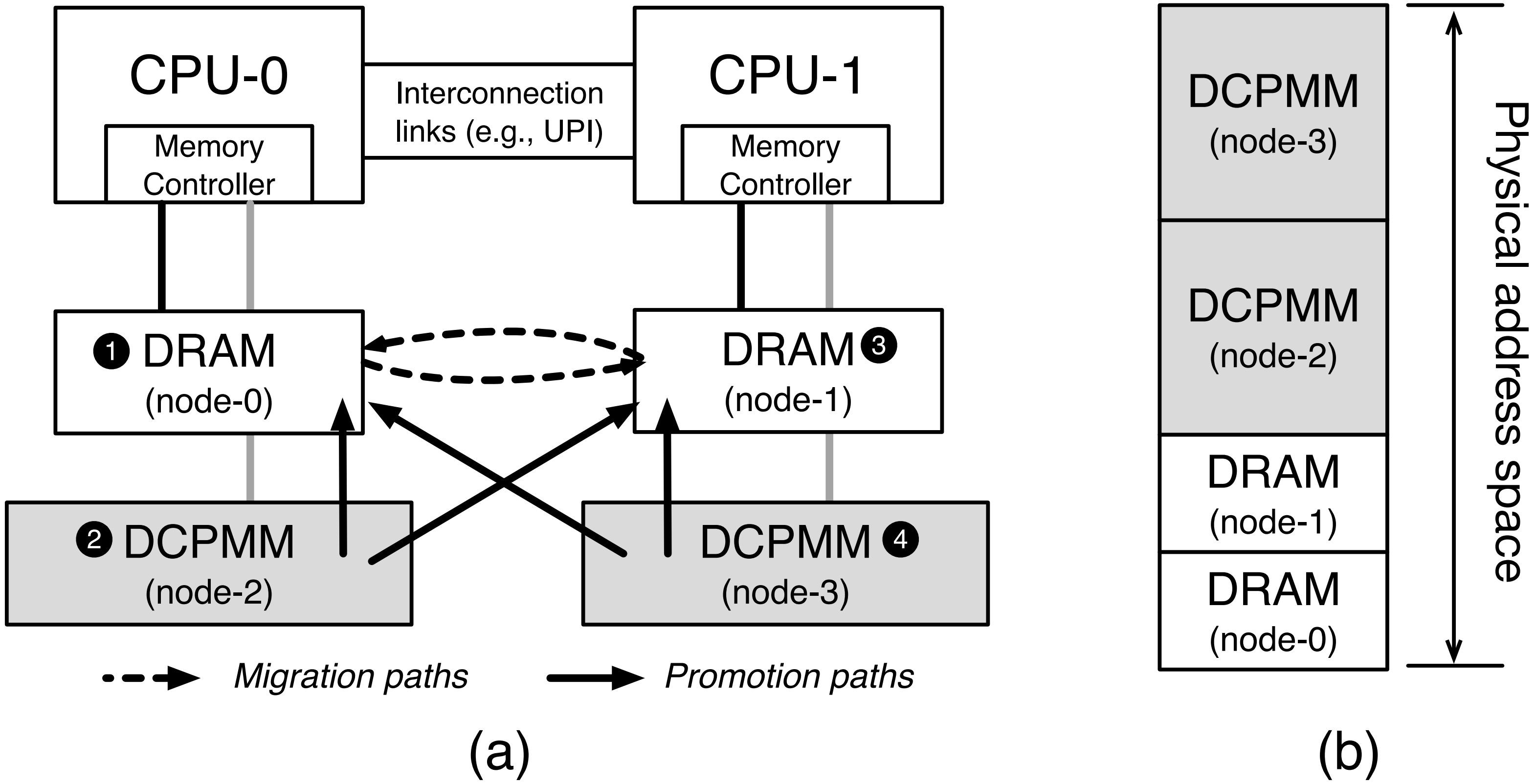


Allow persistent memory to be used like normal RAM

From: Dave Hansen <dave.hansen-AT-linux.intel.com>
To: dave-AT-sr71.net
Subject: [PATCH 0/4] Allow persistent memory to be used like normal RAM
Date: Wed, 16 Jan 2019 10:18:59 -0800
Message-ID: <20190116181859.D1504459@viggo.jf.intel.com>
Cc: Dave Hansen <dave.hansen-AT-linux.intel.com>, dan.j.williams-AT-intel.com, dave.jiang-AT-intel.com, zwisler-AT-kernel.org, vishal.l.verma-AT-intel.com, thomas.lendacky-AT-amd.com, akpm-AT-linux-foundation.org, mhocko-AT-suse.com, linux-nvdimm-AT-lists.01.org, linux-kernel-AT-vger.kernel.org, linux-mm-AT-kvack.org, ying.huang-AT-intel.com, fengguang.wu-AT-intel.com, bp-AT-suse.de, bhelgaas-AT-google.com, baiyaowei-AT-cmss.chinamobile.com, tiwai-AT-suse.de
Archive-link: Article

```
I would like to get this queued up to get merged.  Since most of the
churn is in the nvdimm code, and it also depends on some refactoring
that only exists in the nvdimm tree, it seems like putting it in *via*
the nvdimm tree is the best path.

But, this series makes non-trivial changes to the "resource" code and
memory hotplug.  I'd really like to get some acks from folks on the
first three patches which affect those areas.

Borislav and Bjorn, you seem to be the most active in the resource code.

Michal, I'd really appreciate at look at all of this from a mem hotplug
perspective.

Note: these are based on commit d2f33c19644 in:

        git://git.kernel.org/pub/scm/linux/kernel/git/djbw/nvdimm.git libnvdimm-pending

Changes since v1:
 * Now based on git://git.kernel.org/pub/scm/linux/kernel/git/djbw/nvdimm.git
 * Use binding/unbinding from "dax bus" code
 * Move over to a "dax bus" driver from being an nvdimm driver

--

Persistent memory is cool.  But, currently, you have to rewrite
your applications to use it.  Wouldn't it be cool if you could
just have it show up in your system like normal RAM and get to
it like a slow blob of memory?  Well... have I got the patch
series for you!

This series adds a new "driver" to which pmem devices can be
attached.  Once attached, the memory "owned" by the device is
hot-added to the kernel and managed like any other memory.  On
systems with an HMAT (a new ACPI table), each socket (roughly)
will have a separate NUMA node for its persistent memory so
this newly-added memory can be selected by its unique NUMA
node.

Here's how I set up a system to test this thing:

1. Boot qemu with lots of memory: "-m 4096", for instance
```

- Modern server systems have formed NUMA
- DRAM and DCPMM share the memory controller
- Since Linux 5.0, Optane DCPMM can be exposed as a normal RAM
    - A DCPMM memory node is treated as a CPU-less NUMA node

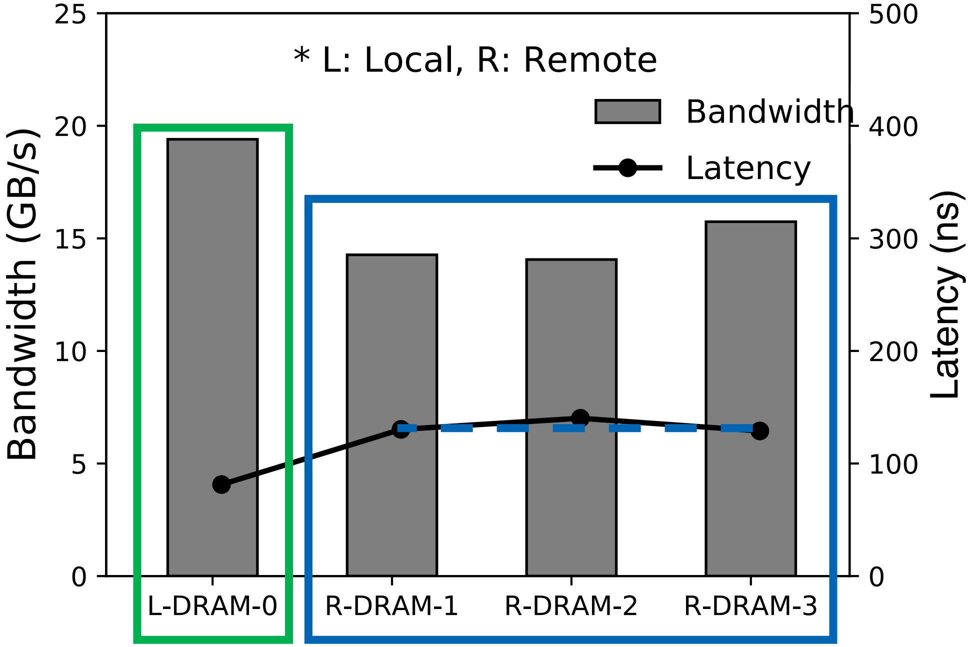


`SW-managed` tiered memory organization

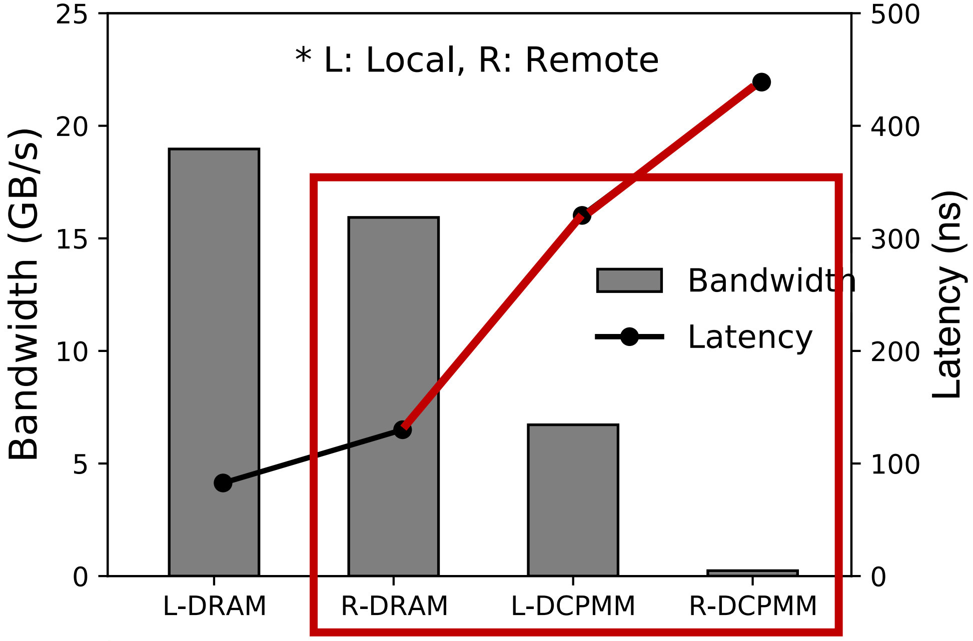
(A DCPMM memory node is treated as a CPU-less NUMA node)

Exploring the Design Space of Page Management for Multi-Tiered Memory Systems [USENIX ATC 2021]

Memory access latency and bandwidth



4-socket DRAM-based NUMA machine

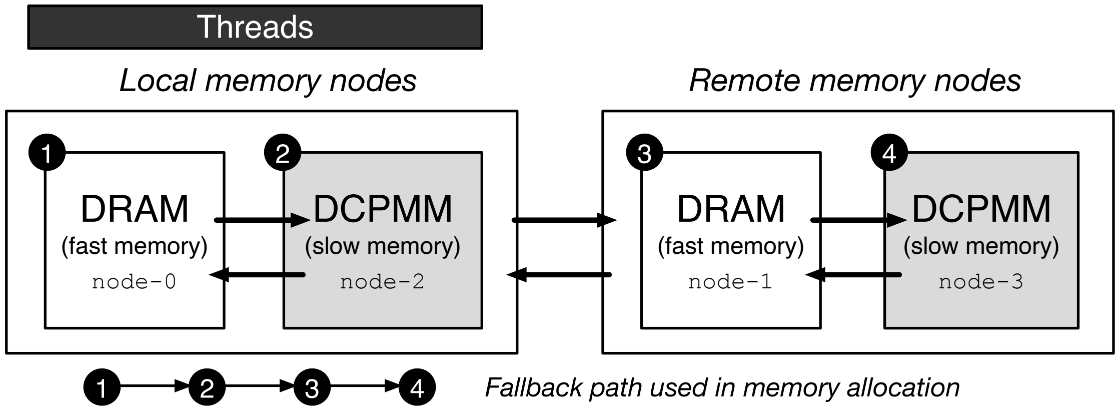


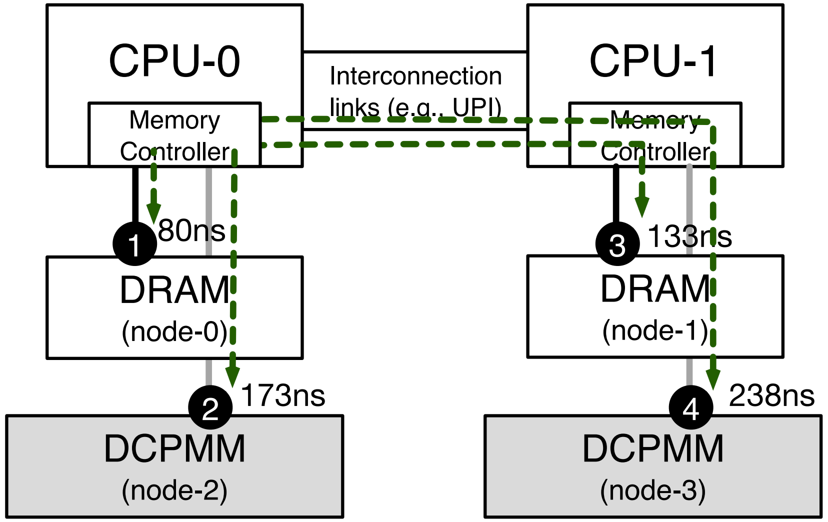
Multi-tiered NUMA machine

The critical factor in performance is not only **access locality** but also **access tier** of memory

Default memory placement: `local-first`

State-of-the-art Linux kernel* has not considered the characteristics of fast (DRAM) and slow (DCPMM) memory with NUMA properties



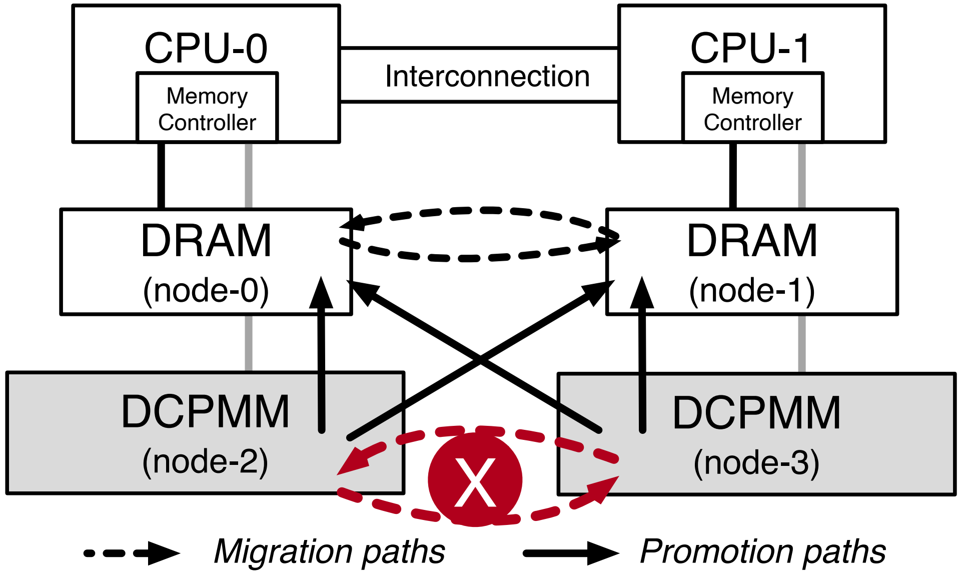


Q. Why don't you reorder the fallback node list according to the actual performance?

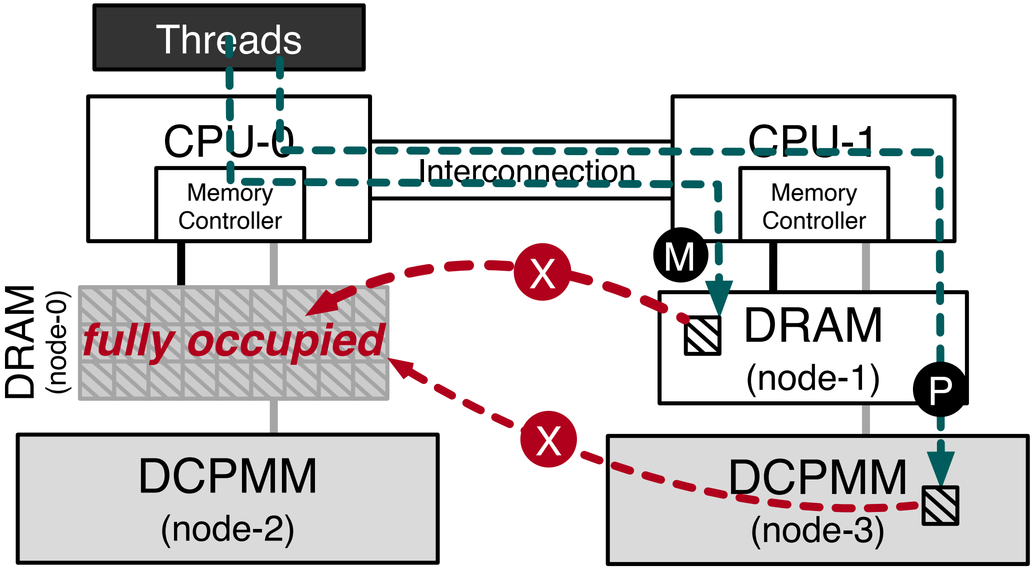
*As of conducting this study, we used the Linux kernel version 5.3

Limited page placement in current Linux

Why not `AutoNUMA`?

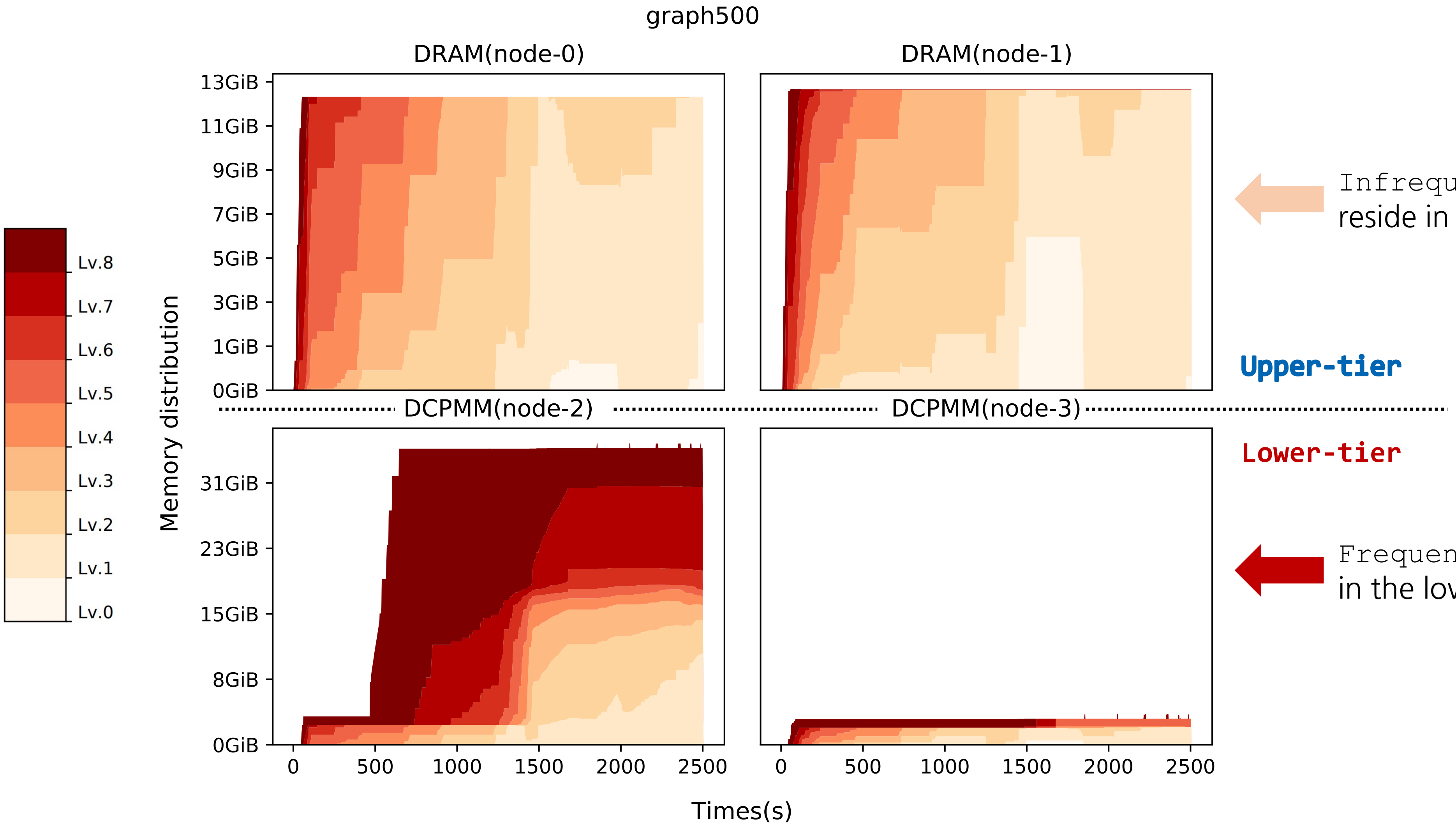


Page movement to **`CPU-less`** nodes (DCPMM) is prohibited in the current Linux



Page movement is allowed only when the target node has a free space in the current Linux

Need for page placement for tiered memory



`Infrequently` accessed pages reside in the upper-tier memory

`Frequently` accessed pages are found in the lower-tier memory

Stock Linux 5.3 version

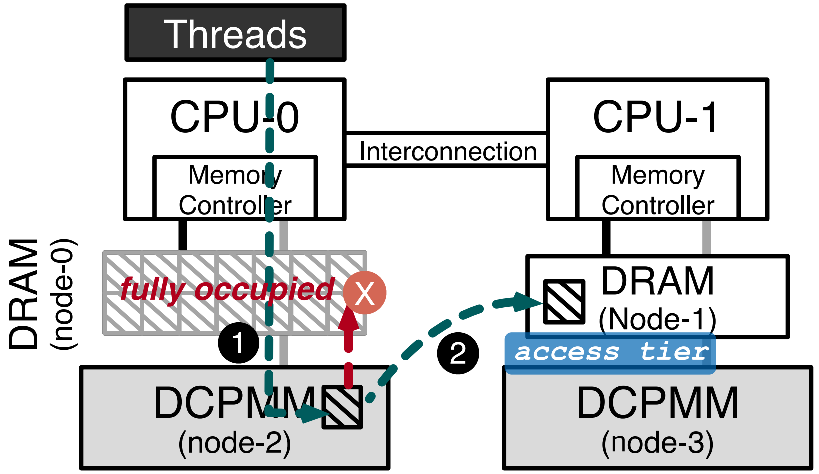
Problems with current page management

Problems (or limitations)

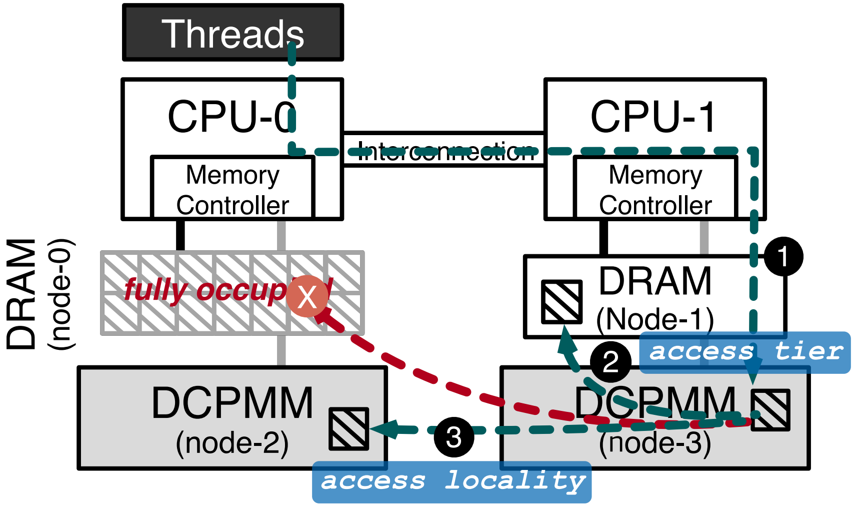
1. Allocation fallback does not consider access tier.
2. Pages are not promoted when upper-tier is full
3. Pages are never demoted or reclaimed to lower-tier memory
4. Page classification is too coarse-grained (binary: active or inactive)

Exploiting access tier first and then locality

Conservative Promotion or Migration → `AutoTiering-CPM`



Case-1: page promotion



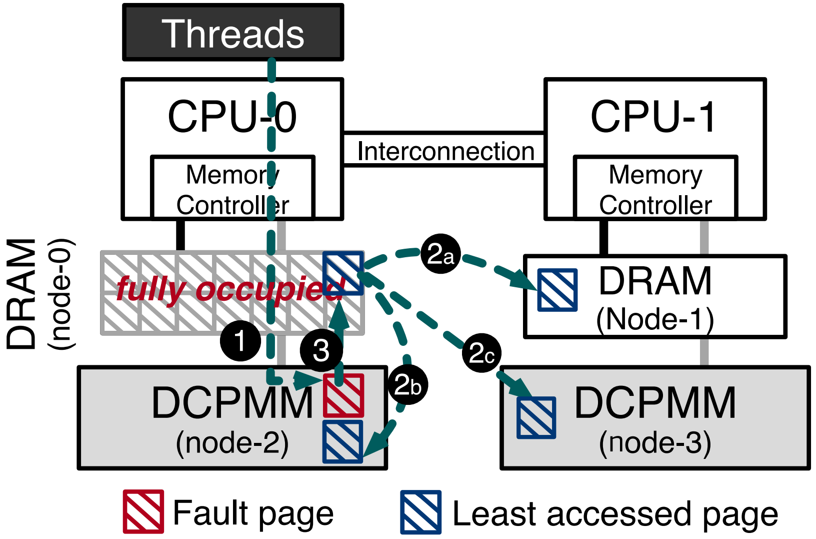
Case-2: page promotion or migration

`AutoTiering-CPM` provides alternatives for page migration failure due to fully occupied target memory node, leading to performance improvement

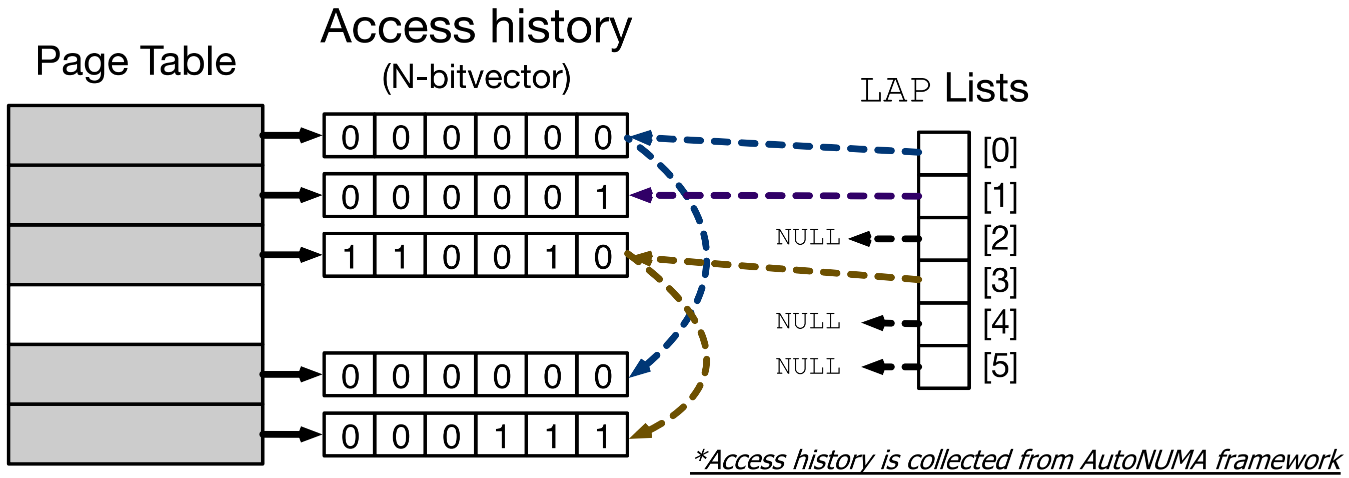
However, the upper-tier (DRAM) memory can still hold infrequently accessed data while frequently used pages reside in the lower-tier (DCPMM) memory

Enforcing page promotion and migration

<u>O</u>pportunistic <u>P</u>romotion or <u>M</u>igration → `AutoTiering-OPM`



- Finding the least accessed page (LAP)
  1. Inactive page from `file-backed` region
  2. LAP page from `anonymous` region



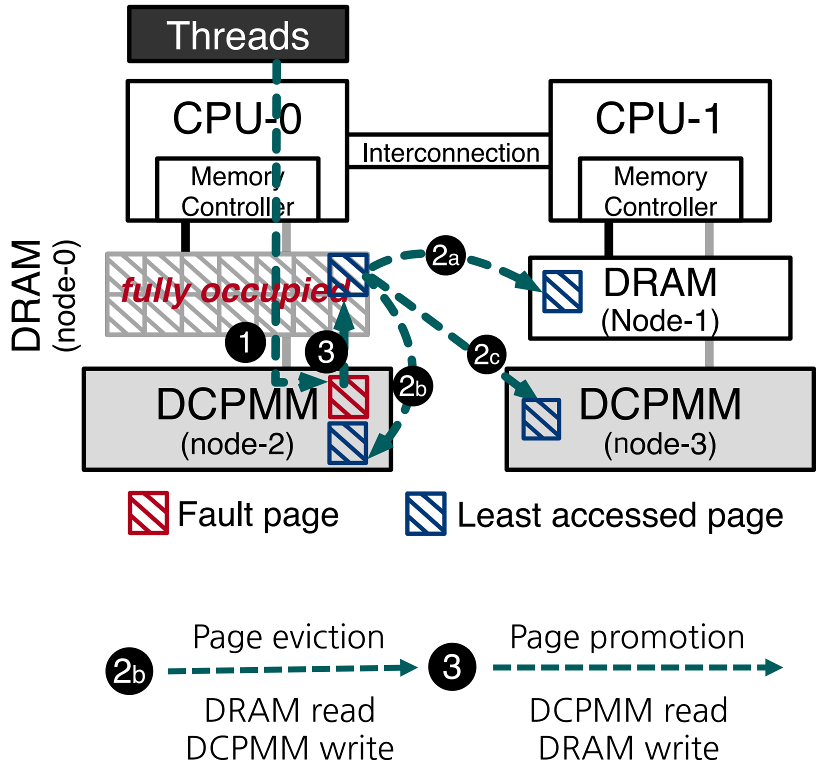
0 0 0 0 0 0
0 0 0 0 0 1
1 1 0 0 1 0
0 0 0 0 0 0
0 0 0 1 1 1

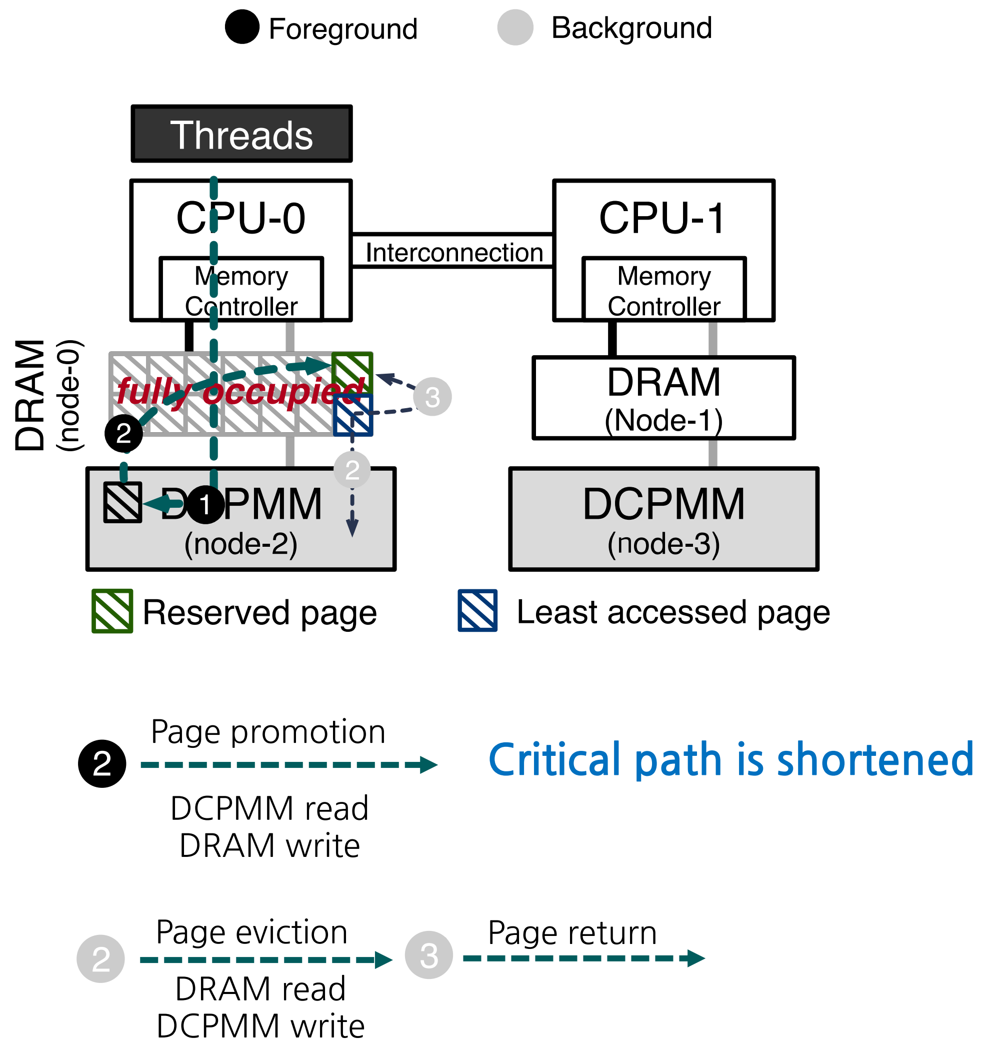
Access history is collected from AutoNUMA framework

With `AutoTiering-OPM`, we can achieve better utilization of the upper-tier memory

Hiding latency of page eviction

A software optimization comes to rescue

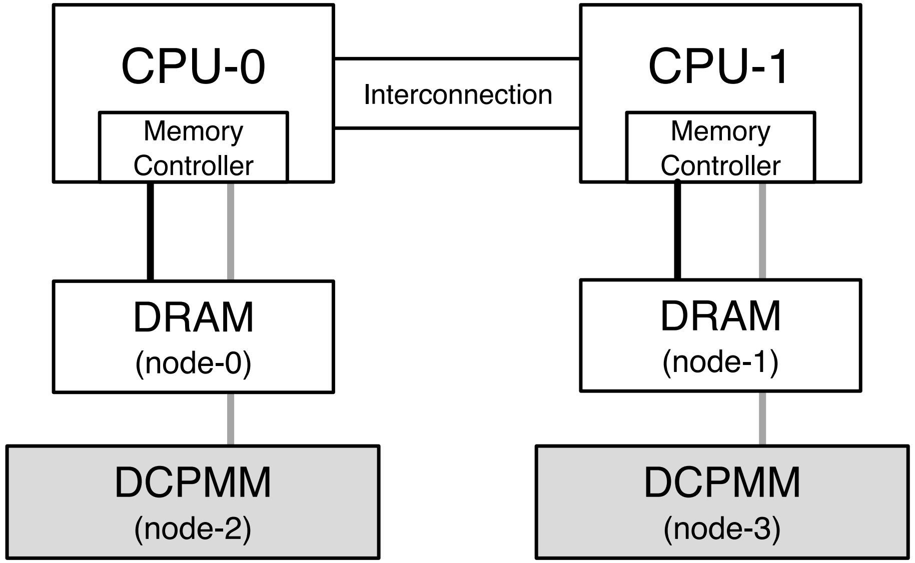




Critical path is shortened

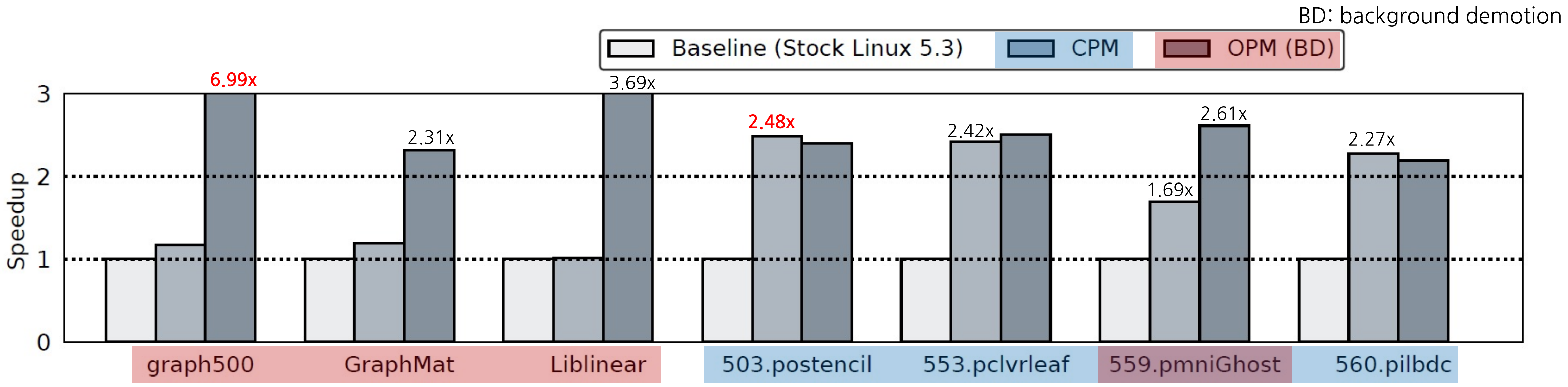
Experimental environments

- System
  - Intel(R) Xeon Gold 5218 CPU @ 2.30GHz x 2
  - 16GB DRAM x 2
  - 128GB Intel Optane DCPMM x 2
  - Linux kernel 5.3 with Ubuntu 18.04



- Benchmarks
  - SPECAccel (OpenMP)
  - GraphMat (PageRank)
  - Graph500 (BFS)
  - Liblinear

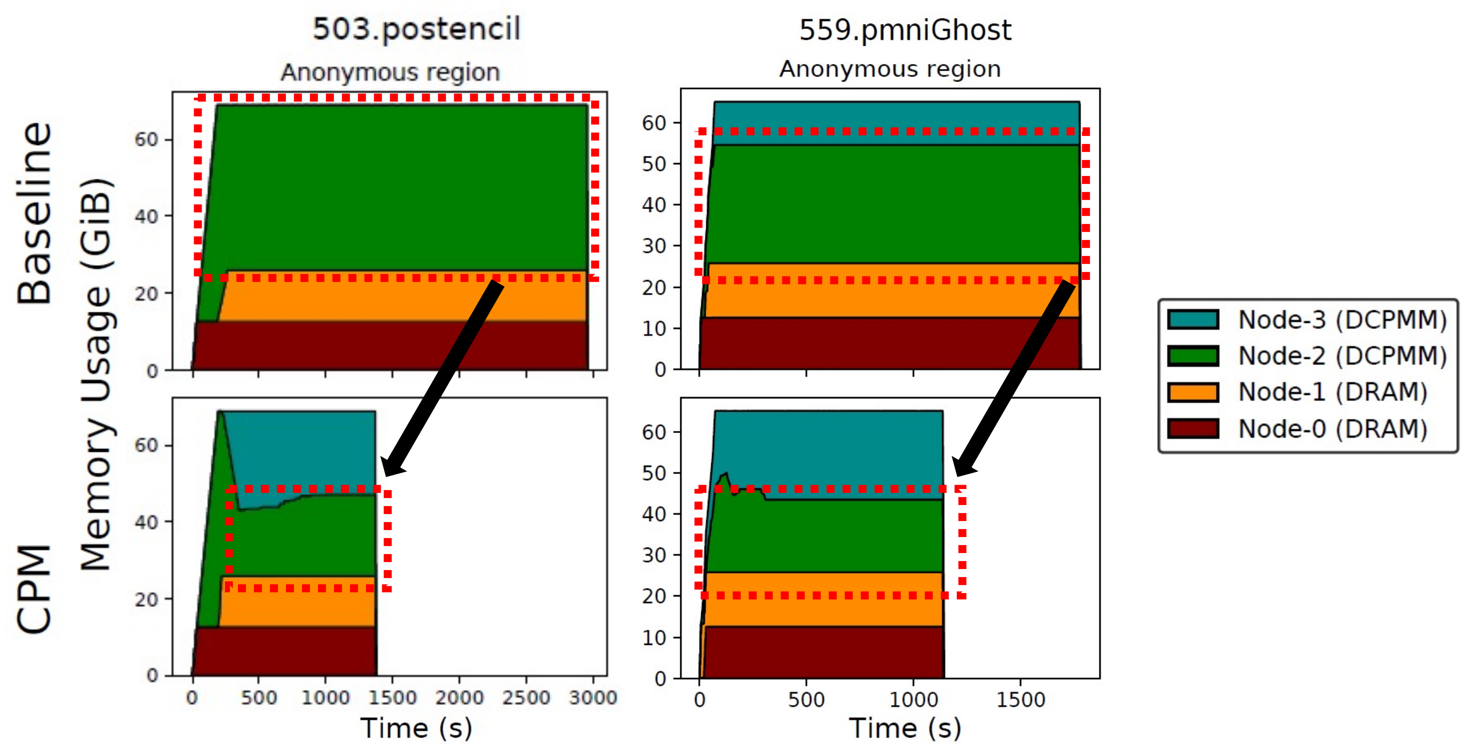
Performance evaluation



- ⁎ Most benchmarks are improved by `AutoTiering`
- ⁎ With `CPM`, speedup is up to 2.48x in **`503.postencil`**
- ⁎ With `OPM(BD)`, speedup is up to 6.99x in **`graph500`**

Effectiveness of AutoTiering-CPM

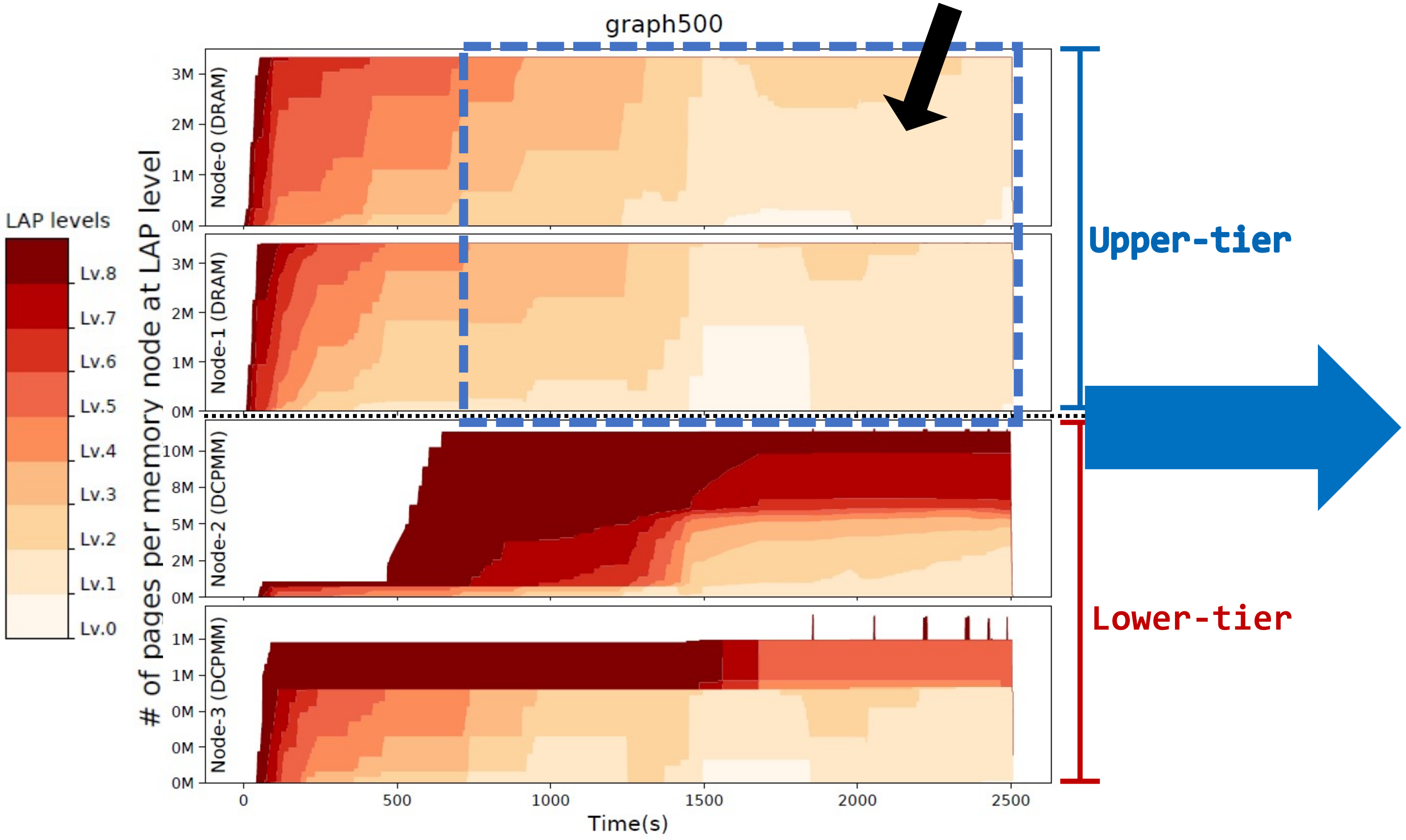
Distribution of Memory Usages



AutoTiering-CPM makes better use of multi-tiered memory

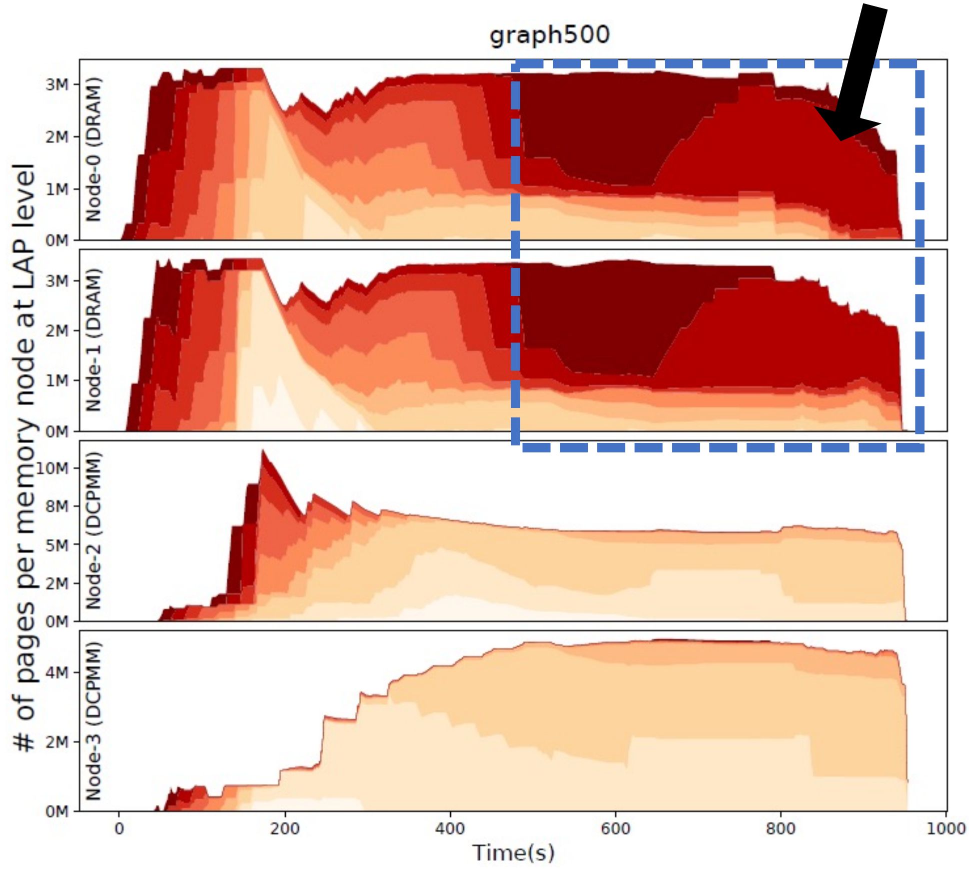
Effectiveness of LAP classification

Less effectively utilized



Stock Linux 5.3 version

More effectively utilized



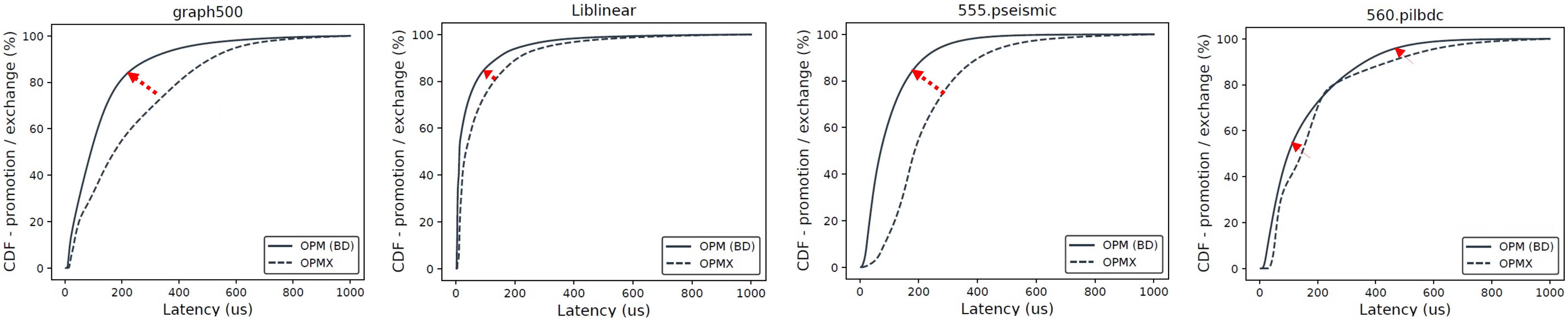
AutoTiering-OPM

AutoTiering-OPM can promote frequently accessed pages while demoting least accessed pages

Effect of hiding demotion latency

Measured page promotion latency (CDF) with `ftrace`

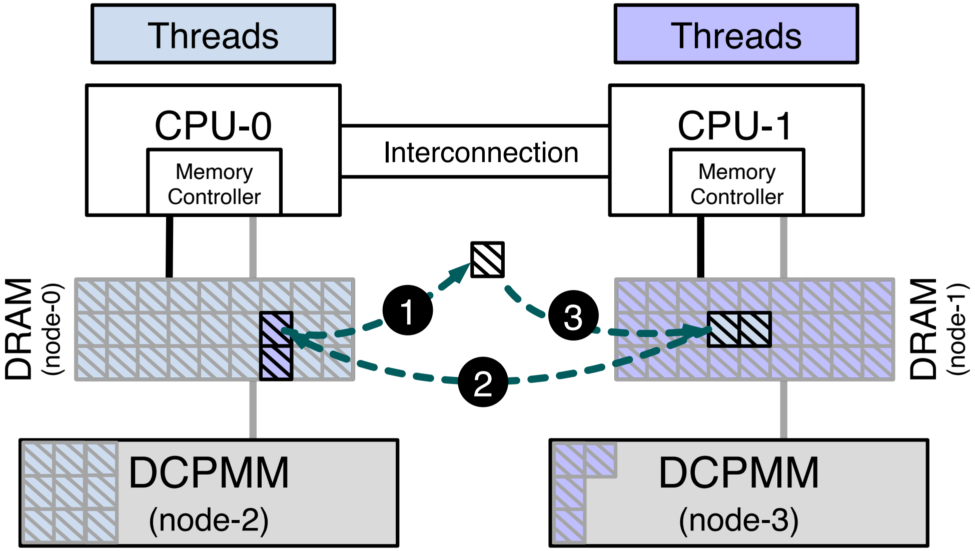
- `OPMX`: Opportunistic Page Migration with Exchange*



We can reduce the promotion latency by deferring the page demotion as background

*Nimble Page Management for Tiered Memory Systems [ASPLOS '19]

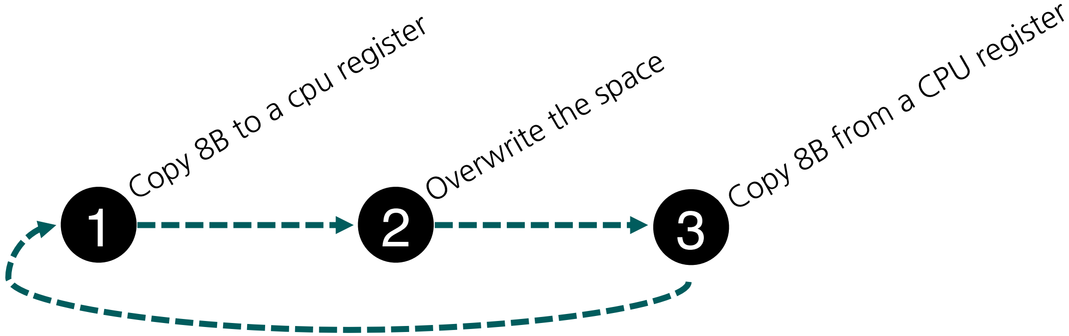
Is page exchange acutally needed?



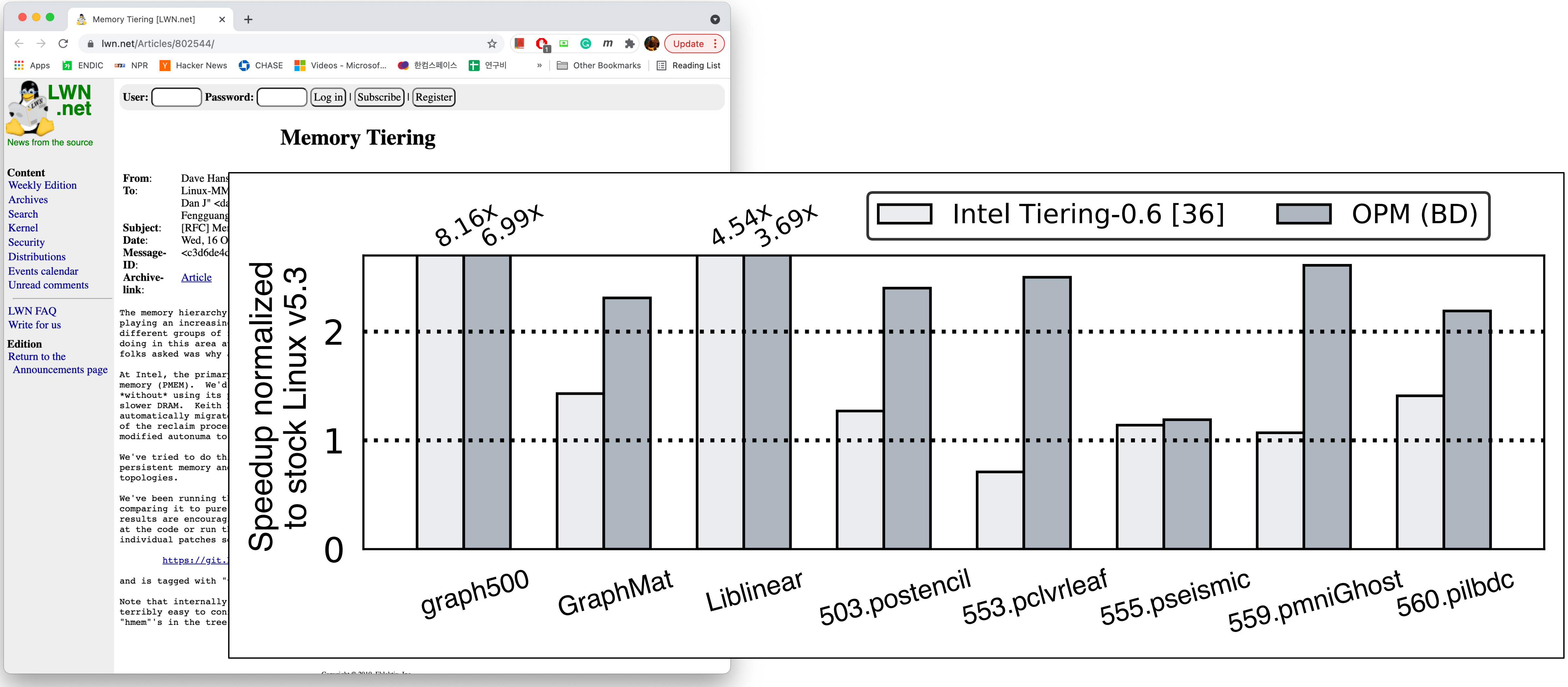
Page exchange scheme*

instead of copying data into new pages, we transfer data between each pair of pages using copy thread(s) that use CPU registers as the temporary storage for in-flight iterative data exchange operations. This use of registers allows our mechanism to avoid allocating a complete temporary page.

* Nimble Page Management for Tiered Memory Systems [ASPLOS '19]



Performance comparison with prior work



Content
Weekly Edition
Archives
Search
Kernel
Security
Distributions
Events calendar
Unread comments

LWN FAQ
Write for us

Edition
Return to the
Announcements page

User: Password: Log in | Subscribe | Register

Memory Tiering

From: Dave Hans
To: Linux-MM
Dan J" <da
Fengguang
Subject: [RFC] Me
Date: Wed, 16 O
Message-ID: <c3d6de4
Archive-link: Article

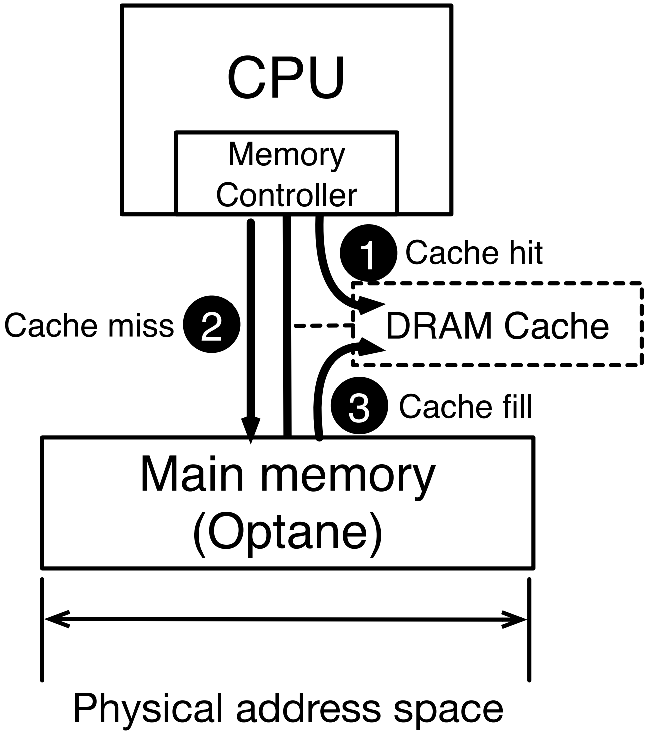
```
The memory hierarchy
playing an increasin
different groups of
doing in this area a
folks asked was why

At Intel, the primar
memory (PMEM).  We'd
*without* using its
slower DRAM.  Keith
automatically migrat
of the reclaim proce
modified autonuma to

We've tried to do th
persistent memory an
topologies.

We've been running t
comparing it to pure
results are encourag
at the code or run t
individual patches s

        https://git.

and is tagged with "

Note that internally
terribly easy to con
"hmem"'s in the tree
```

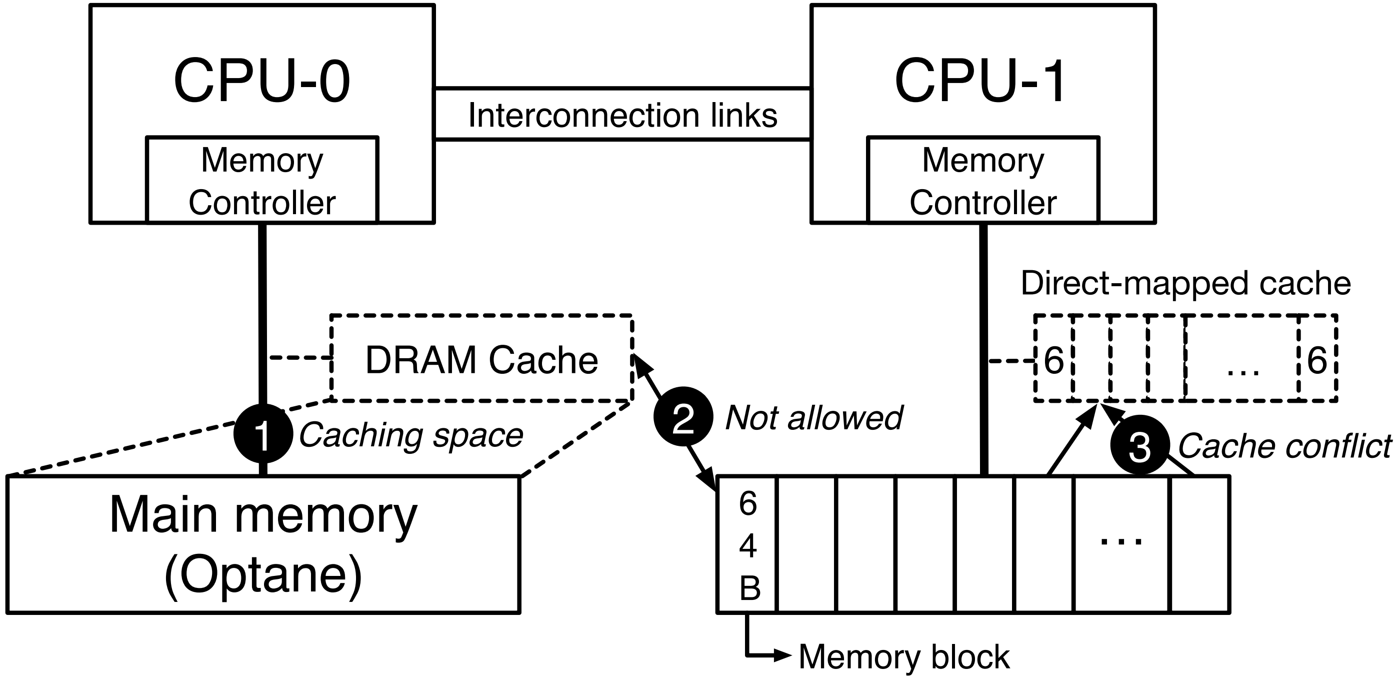
Summary

- Commodity OSes are not mature enough to support multi-tiered memory systems

- We explored new page placement schemes to extract the full benefits of multi-tiered memory systems

- Future work
    - Redesigning the kernel thread demoting page migration to DCPMM with the consideration of limited memory bandwidth of DCPMM
    - Adopting the newly added framework monitoring memory access pattern in Linux kernel called `DAMON` to reduce the access tracking overhead





HW-assisted tiered memory organization

(invisible DRAM to OS)

A Study of of Memory Placement on Hardware-assisted Tiered Memory Systems [IEEE CAL 2020]

HW-assisted tiered memory organization

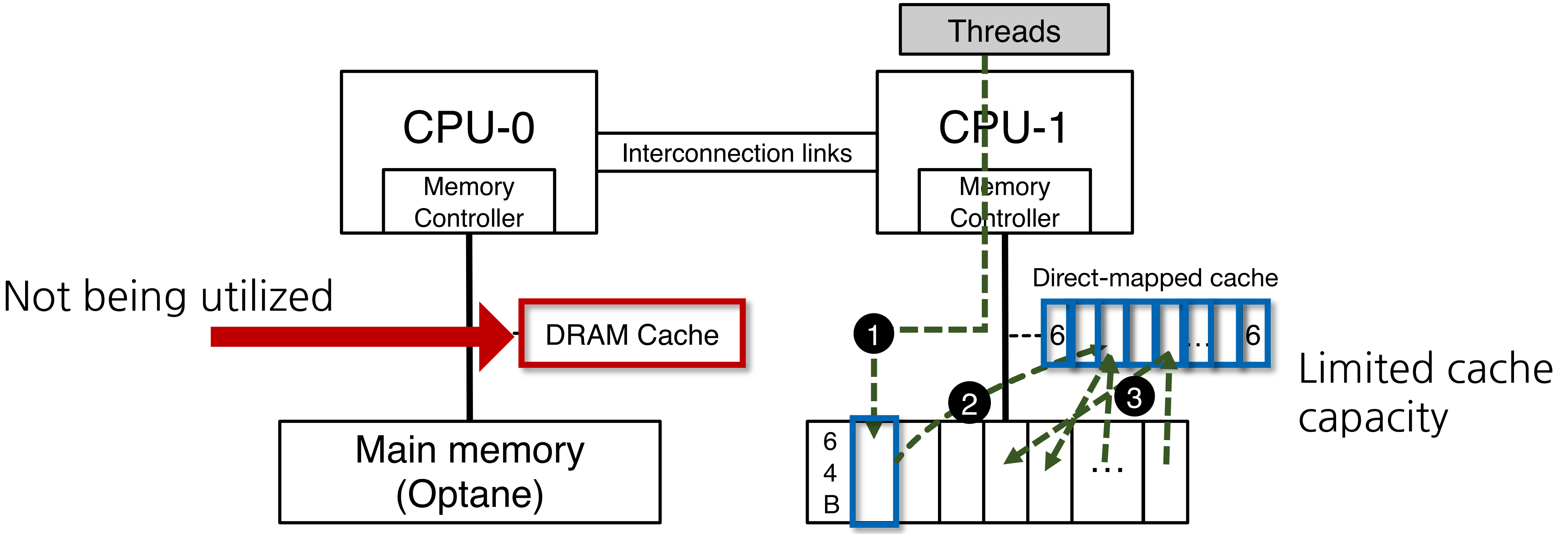
- Transparent to software
  - Any software modification is not required

- We are curious about the commodity operating systems work well
  - Modern memory management is highly optimized DRAM-only systems
    - Without consideration of heterogenous (hybrid or tiered) memory systems
    - Only NUMA characteristics are considered

- We revisit the design and implementation of operating systems

Recall memory placement: `local-first`

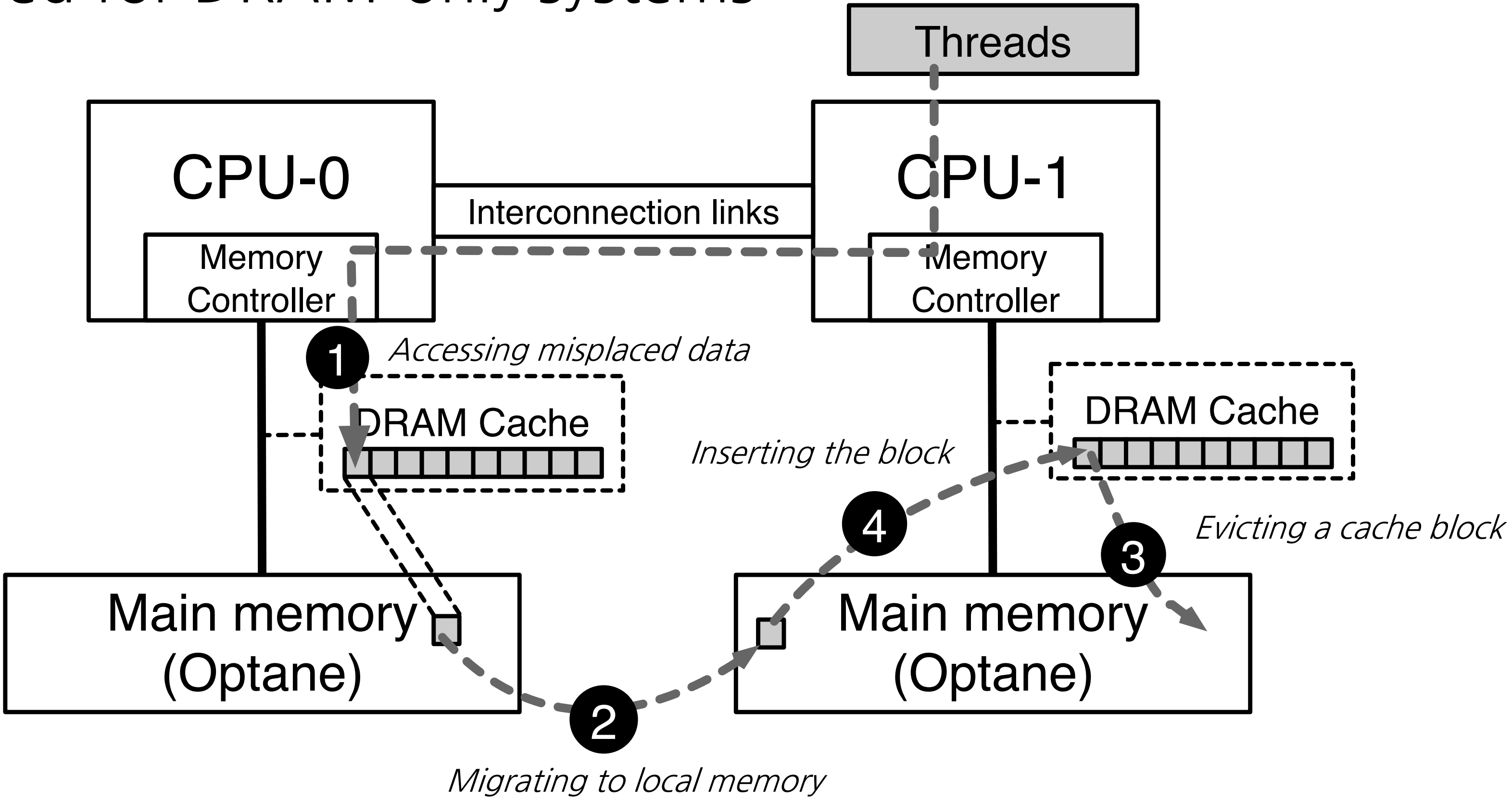
Let's see how the local-first policy works on the HW-assisted tiered memory system



The `local-first` placement policy leads to spending time back and forth between the local DRAM cache and the Optane main memory while the remote DRAM cache is idle

AutoNUMA is considered harmful

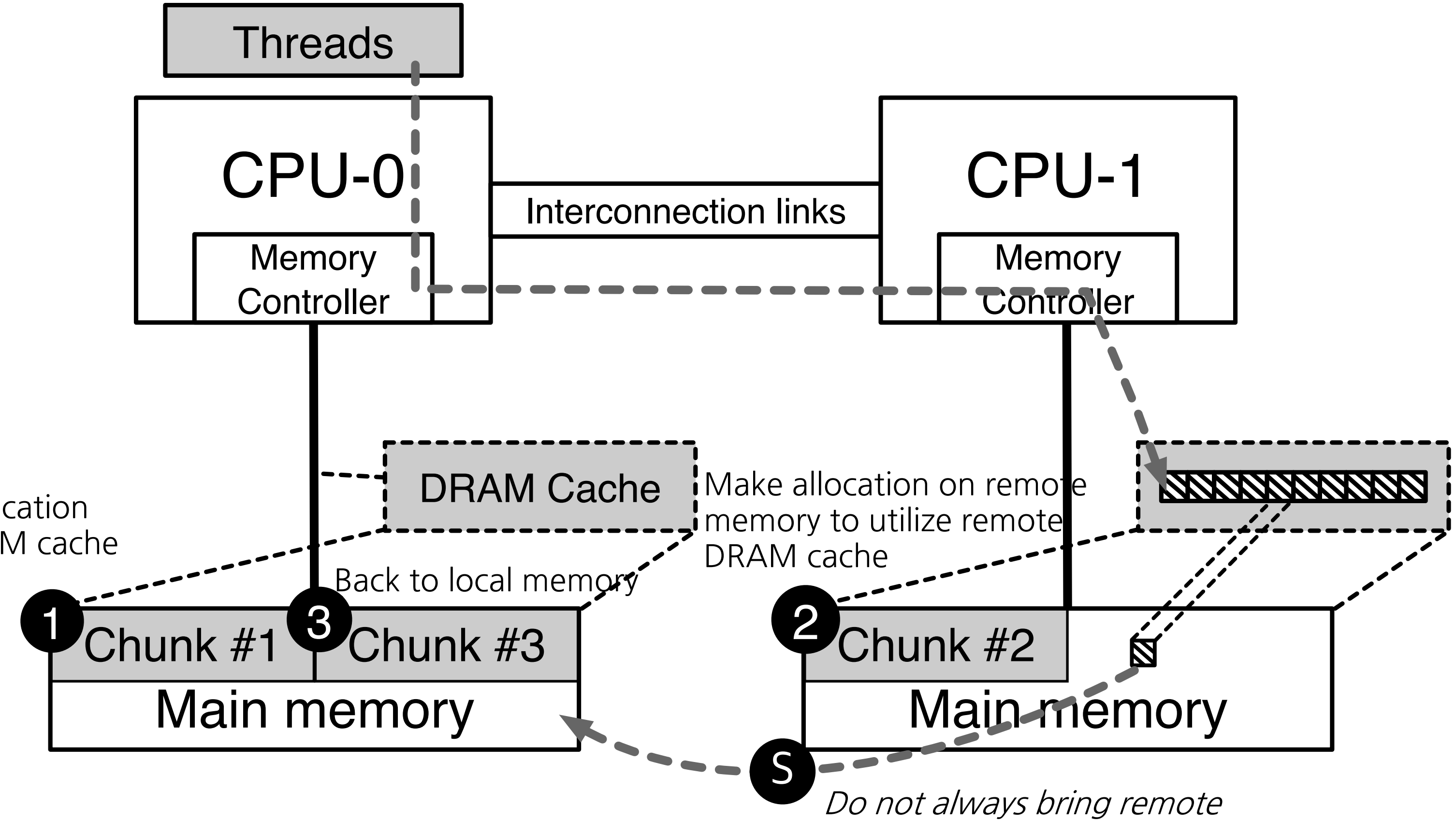
Again, it is designed for DRAM-only systems



AutoNUMA balancing may degrade performance on tiered memory systems. If the local DRAM cache does not have enough space, the application can experience frequent DRAM cache misses while not utilizing the remote DRAM cache

Our approach: `dram-first`

Exploiting such hardware characteristics in placing memory (pages)



Allow memory allocation that can fit in DRAM cache

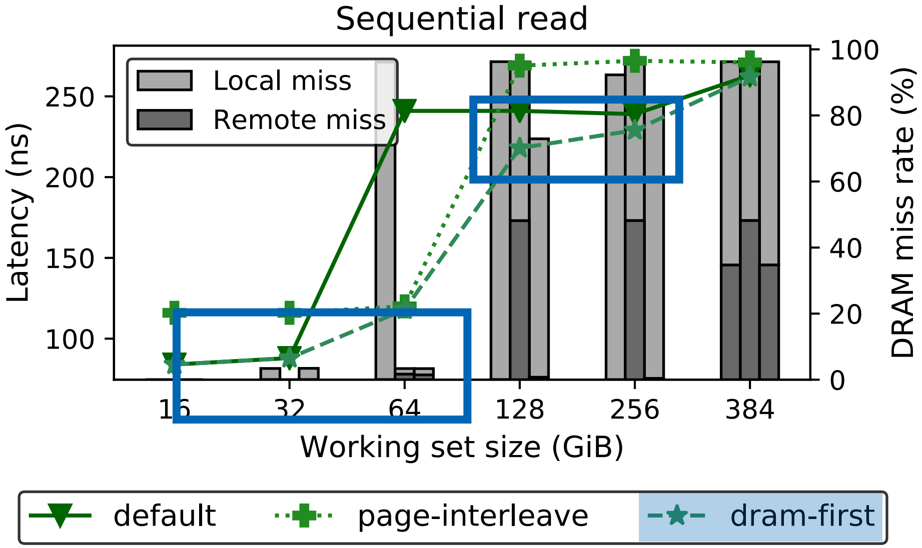
Back to local memory

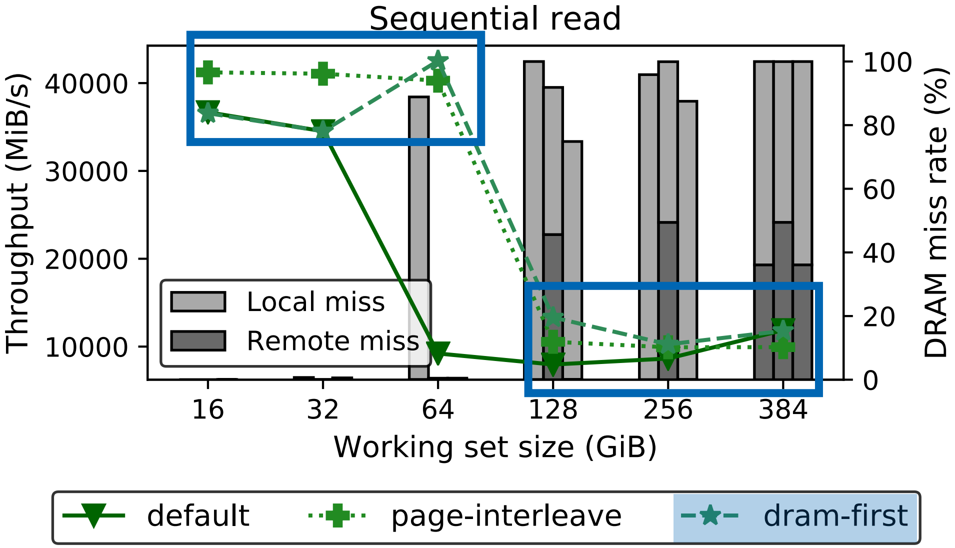
Make allocation on remote memory to utilize remote DRAM cache

Do not always bring remote (DRAM) pages

Preliminary results: latency & bandwidth

Experimental environments
- Intel Xeon Gold 5218 (16cores) x 2 sockets
- Two DRAM 16GB dimms and two DCPMM 128GB dimms per socket
- Linux kernel 5.3 with Ubuntu 18.04 server distribution

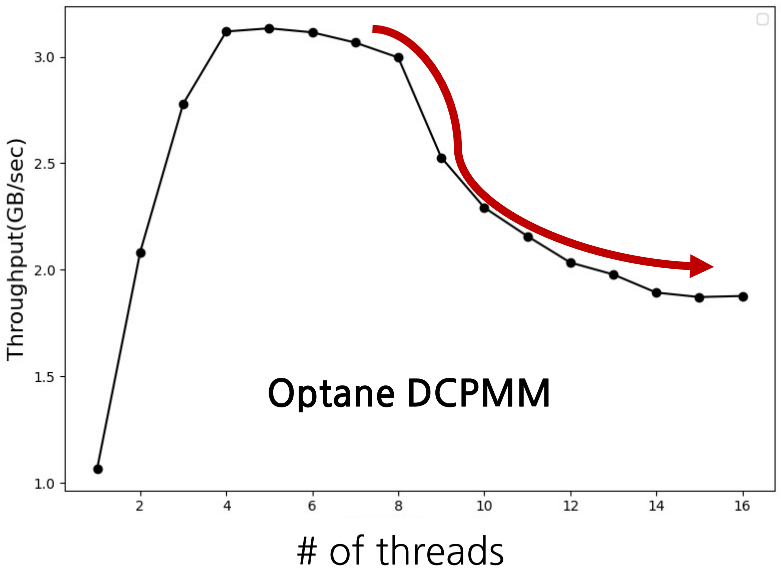


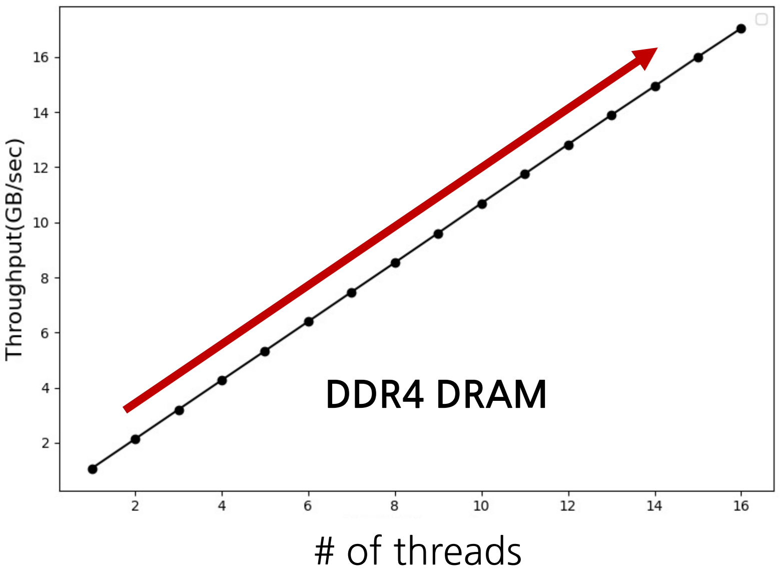


Remaining challenges in tiered memory systems

1. Demoting or migrating pages to Optane memory suffers from the limited memory bandwidth and leads to write amplification problems

*Random write with 256B granularity

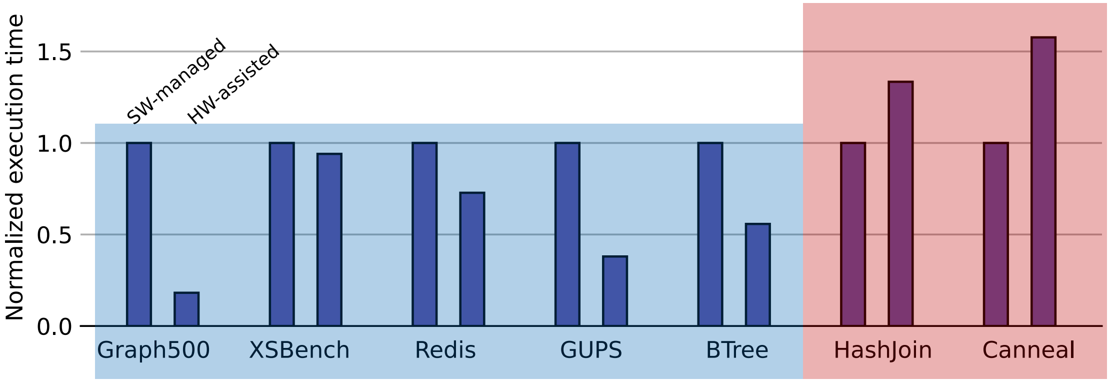




Remaining challenges in tiered memory systems

1. Demoting or migrating pages to Optane memory suffers from the limited memory bandwidth and leads to write amplification problems

2. Minimizing DRAM cache conflict misses within an Optane memory node
   - DRAM cache is organized as a direct-mapped cache
   - Two more memory blocks cannot be mapped to a single cache set
   - Note that the DRAM cache indexing scheme has not been disclosed



Thank You!

Artifact available at https://github.com/csl-ajou/AutoTiering

Jeongseob Ahn

E-mail: jsahn@ajou.ac.kr
Web: https://jeongseob.github.io

AJOU UNIVERSITY