A Dive into Computer Systems Research

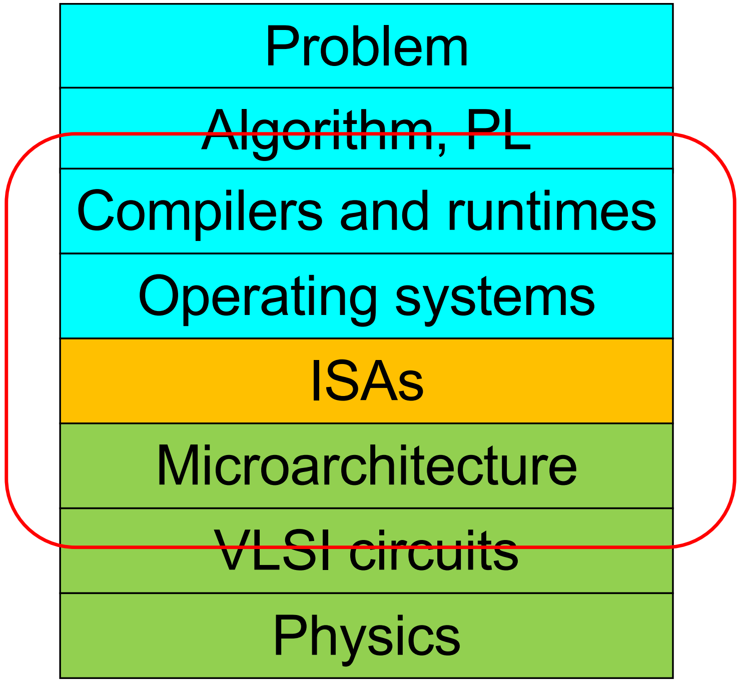
Shoaib Akram
shoaib.akram@anu.edu.au

Australian National University

What is considered systems research?

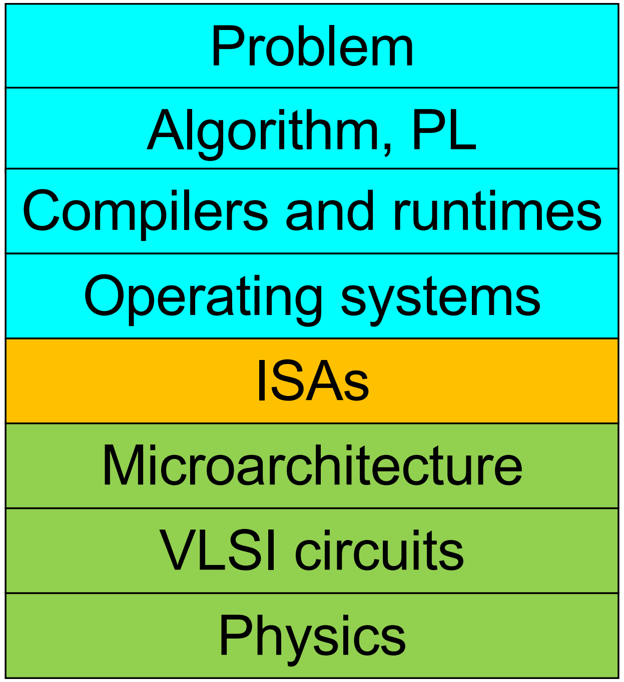
The Transformation Hierarchy

- We use a systematic transformation hierarchy to solve complex problems
  - From English to movement of electrons
- The **“system of transformations”** is built to satisfy **“user constraints”**
  - Device size, cost, energy, reliability
- What is systems research?
  - How to enable the transformation?
  - **Qualified answer:** How best to enable an optimal design point in a complex space
  - Show by building a real system



What is a computer system?

- Sequence of transformations
- Hardware + software
- Compute + storage
- CPU, memory, and disk → computer
- Network of computers → Datacenter
- Network of datacenters → Cloud
- Network of CPU and accelerators → system on a chip



Problem

Algorithm, PL

Compilers and runtimes

Operating systems

ISAs

Microarchitecture

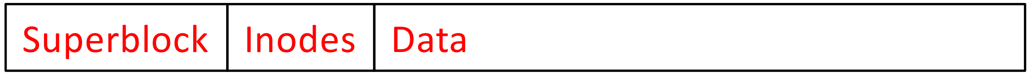
VLSI circuits

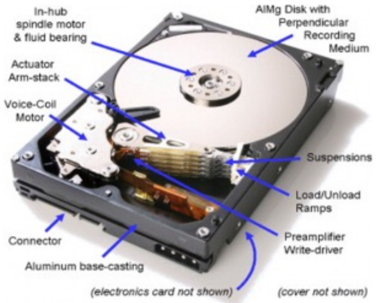
Physics

Two Historical Examples

- Two examples
  - Storage and file systems
  - Processor microarchitecture

Fast File System (FFS)

- Unix OS is introduced. Ken Thompson wrote the first filesystem
- Simple and elegant (?) 
  Superblock | Inodes | Data
- Unfortunately, performance was terrible
- Kirk McKusick measured it could utilize only 2% of disk bandwidth
- **Problem:** Filesystem was written as if the underlying device was a random access memory (like physical memory)


In-hub spindle motor & fluid bearing
Actuator Arm-stack
Voice-Coil Motor
AlMg Disk with Perpendicular Recording Medium
Suspensions
Load/Unload Ramps
Connector
Preamplifier Write-driver
Aluminum base-casting
(electronics card not shown)
(cover not shown)

- But, disk is not a random access device
- It has mechanical components. Arm movement. Rotational disk
- Sequential accesses are faster than random access
- A group at Berkeley wrote the fast “disk-aware” filesystem
- **Key constraint:** Not enough details of the device are exposed to the system
- **Key realization:** Exploit device organization/physics whatever is known about it. “Keep related stuff together”

Out of Order Execution

- **1960s and 70s:** It is established that the programming model of a Von Neuman machine is intuitive for the programmers
- And that such machines are practical to build on large scale
- **Problem:** One operation per clock cycle in program order (as specified by the Von Neuman model) is very restrictive
  - Need to concurrently execute many instructions in one clock cycle to gain higher performance
- **Solution: (**a marvel of human ingenuity**)**
  - **Key constraint:** Instructions have dependences, so how can one find conc.
  - **Key realization:** With some effort one can find independent insts. in programs
  - **Dynamic scheduling:** Fetch instruction in order, but execute instructions whenever their operands are ready (dataflow machine with seq. model)
  - Control Data and IBM the early innovators
  - Improved over many decades (branch prediction, precise interrupts)

Lessons

- **FFS was possible** because the team that built it realized that it is critical to look one layer below the OS abstraction layer
  - They realized early on that device physics shapes the system
  - They also realized the need for good abstraction, so they did not change what was exposed to the users of FFS
  - Modern file systems still use the same file system API

- **OOO was possible** because early systems researchers at CDC and IBM studied program behavior and program interaction with machines
  - **They were innovating at many layers:** ISA, OS, microarchitecture, compilers, design, PL, algorithms, management
  - In this specific instance, a different debate emerged. OOO in hardware is too complex. Why can't compiler do it? Compiler/uARCH both innovated.

Computer Architecture Ideas

- Venues
  - ISCA, MICRO, ASPLOS, HPCA

- Not much in traditional OOO processor microarchitecture
- Memory systems: caching, coherence, consistency, multicore
- DRAM reliability
- Mitigating security vulnerabilities
- Processing in memory
- New storage technologies
- ML accelerators
- ML for systems

Operating Systems Ideas

- Venues
  - ASPLOS, OSDI, SOSP

- True OS papers: Very hard to find

- Garbage collection
- Data-intensive systems
- NoSQL stores
- Persistent memory programming models
- Compute in NICs
- Computational storage

Why should you consider it?

- Key enabler of new and “emerging” applications
  - Millisecond-scale real-time analytics over social media

- Broad applicability
  - 1% improvement in GPU throughput for ML

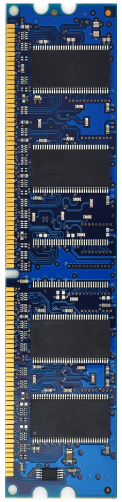
- Building systems is fun although “challenging”

- Lots of room to work at different abstraction layers
  - Same problem can have a variety of solutions: Compiler vs. managed runtime vs. OS vs. hardware

- Can help produce better algorithms, think new problems, move technology

Ongoing Research

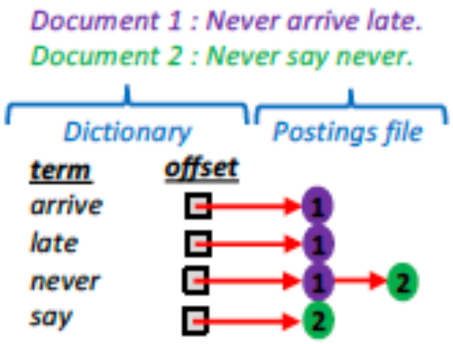
Motivation



- Lot of pressure on physical memory (DRAM)
- Technology is not scaling as fast as it used to
- But applications demand more memory
- **Key realization**
  - Data is expensive to cache and store for fast delivery
  - **Meta-data is more expensive**
    - Counter-intuitive. Why?

Example 1: Search Engines

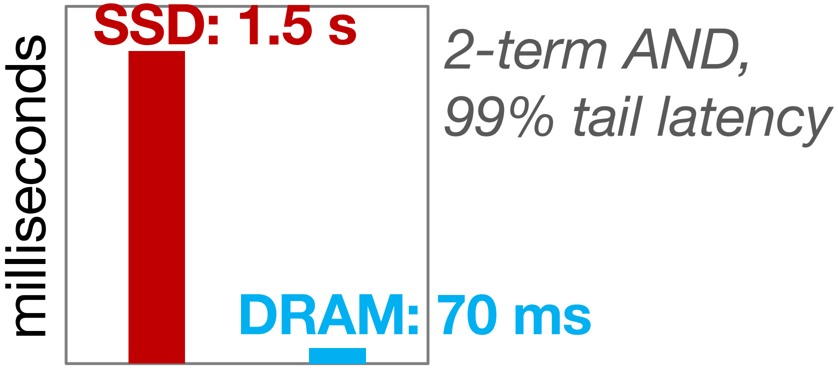
- Key data structure that enable fast search
  - Inverted index


Document 1 : Never arrive late.
Document 2 : Never say never.
Dictionary
Postings file
term
offset
arrive
late
never
say
1
1
1
2
2

- Think of a massive hash table

- Every time we create a new website or tweet, something gets added to the hash table

- **Hash table placement and query response time**



2-term AND, 99% tail latency

Example 2: ML Analytics

- Iterative computation until a condition is met

- Each iteration produces a transformation of a massive dataset

- **Two options**
  - Recompute the transformation whenever needed (possibly in every transformation)
  - Cache it in memory or disk

- Cache capacity to avoid recomputation **10X** of actual dataset!

Some Ongoing Projects

- Huge heaps without increasing GC overhead
- Rethinking software stacks for emerging memories
  - Search engines, databases, caches
- Accelerators for proteomics discovery
- Secure and practical memory systems
- Storing and querying very large indices in memory

Aim of a research project

What is the aim of a research project?

- I will give you five keywords to post in your workspace
- Remember them like the stages of instruction processing in a basic CPU pipeline

Fetch	Decode	Execute	Memory	Writeback

- Aim of a research project

Question	Answer	Data	Argument	Revise

- Ask a **question worth answering**
- Find an **answer that you can support** with **good reasons**
- Find **good data** that you can use as **reliable evidence** to support your reasons
- Draft an **argument** that makes a **good case** for your answers
- **Revise** the draft until reader would think you **meet the first four goals**
- It is important to realize how **best to utilize** your mentor for each step
  - **Hindsight:** Wished had engaged mentor more for Question, Argument, Revise

Formula for Questions

- Three step process

1) Topic: I am working on X (history of ANU school of computing)
    2) Question: because I want to find out Y (why students love it so much)
        3) Significance: so I can help others understand Z (how can ANU SOCO practices help other schools in the region attract more students)

Why is the question worth asking?

1) Topic: I am working on machine learning analytics
    2) Question: because I want to find out how it performs on modern GPUs
        3) Significance: so I can help others understand how to architect GPUs to accelerate ML analytics

- In systems research, we build artifacts, so typically, we use the understanding to build stuff (there is an additional step)
    - We cannot build stuff without **"understanding."** That is ANTI-RESEARCH
- So, if you sit in a talk where someone begins with, "I built X." **Ask:** "What informs the design and architecture of X?" Do we understand the behavior of existing systems that do X? Why did you built X? Who benefits? **Why** does X work?

More Example Questions

- Three step process

1) Topic: I am working on memory management

    2) Question: because I want to find out the overhead of malloc() on Linux

        3) Significance: so I can help others understand how to build high-performance memory allocators

        4) Finally, I use the understanding to build **kangaroomalloc()**

1) Topic: I am working on branch prediction

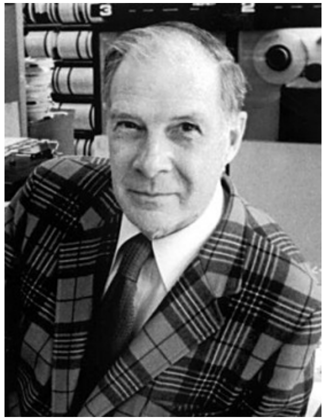
    2) Question: because I want to find out how it behaves for Java workloads

        3) Significance: so I can help others understand how to build new branch predictors for object-oriented languages like Java

        4) Finally, I use the understanding to build **kangaroopredictor()**

Wrong

- I propose kangaroopredictor
- It exhibits 2% more accuracy for Java workloads
- It uses state of the art machine learning
- **Trust me:** It beats everything else!



"The purpose of computing is insight, not numbers."
- Richard Hamming (Turing Award)

Hamming, "You and Your Research"

Hamming, "You and Your Research" (June 6, 1995)

198K views 10 years ago Richard Hamming: "Learning to Learn"
Intro: I have given a talk with this title many times, and it turns out from discussions after the talk I could have just as well have called it "You and Your Engineering Career," or even "You and Your Career." But I left the word "Research" in the title because that is what I have most studied. From the previous chapters you have an adequate background for how I made the study, and I need not mention again the names of the famous people I have studied closely. The earlier chapters are, in sense, just a great expansion, with much more detail, of the original talk. This chapter is, in a sense, a summ Show more

https://www.youtube.com/watch?v=a1zDuOPkMSw&t=1086s&ab_channel=securitylectures

Wrong

- I propose kangaroopredictor
- It exhibits 2% more accuracy for Java workloads
- It uses state of the art machine learning
- **Trust me:** It beats everything else

"What transfers is insight, not academic design, not performance numbers."

- Bill Dally

- In January 2009 he was appointed chief scientist of Nvidia. He worked full-time at Nvidia, while supervising about 12 of his graduate students at Stanford.
- In 2009, he was elected to the National Academy of Engineering for contributions to the design of high-performance interconnect networks and parallel computer architectures.
- He received the 2010 ACM/IEEE Eckert–Mauchly Award for "outstanding contributions to the architecture of interconnection networks and parallel computers."

Right!

- We find that frequent jumps in object-oriented code due to
  - X, Y, Z, ...
    - result in high misprediction rates
    - 20% of all mispredictions are due to X
    - 30% due to Y
    - 10% due to Z

Good systems problems can be solved in different ways. At different layers. Ok to do it based on your philosophy. But don't dismiss other approaches. Sometimes there is no precedent to solve problem at a specific layer. Good research community enables a variety of solutions.

- One could rewrite code to eliminate X, Y, Z, but that requires extra programming effort. One could add a compiler optimization pass
- We propose **kangaroopredictor** that tackles X, Y, and Z to do better prediction in hardware
- **Note:** The **excitement** is **NO LONGER** in **kangaroopredictor** (it's now the last bullet) but in **"understanding"** the behavior of existing predictors and more importantly, interaction b/w OO programs and hardware

Good ideas cannot be dismissed

- **Instruction set architectures**
  - RISC had clear advantages. MIPS a **great** ISA. **MIPS R10K a great microarchitecture**
  - CISC (Intel x86) became the de facto in high performance computing (some history)
  - Technology (physics) trends **eventually betrayed.** CISC decoding consumes too much power. (Even hardware speculation is being questioned (Meltdown). VLIW return?)
  - **Today:** RISC-V emerged as a popular open-source alternative

- **Memory management**
  - **Predominant opinion as late as early 2000s:** Programmers should manually manage memory for high-performance and memory-efficient code
  - C vs. languages with automatic memory managers (aka garbage collection)
  - Memory became cheaper. Technology scaling lead to high density
  - Programs became too complex (programming burden)
  - **Java became the standard for developing major data processing applications**
    - Search engines, analytics, graph processing, 90% of Apache software

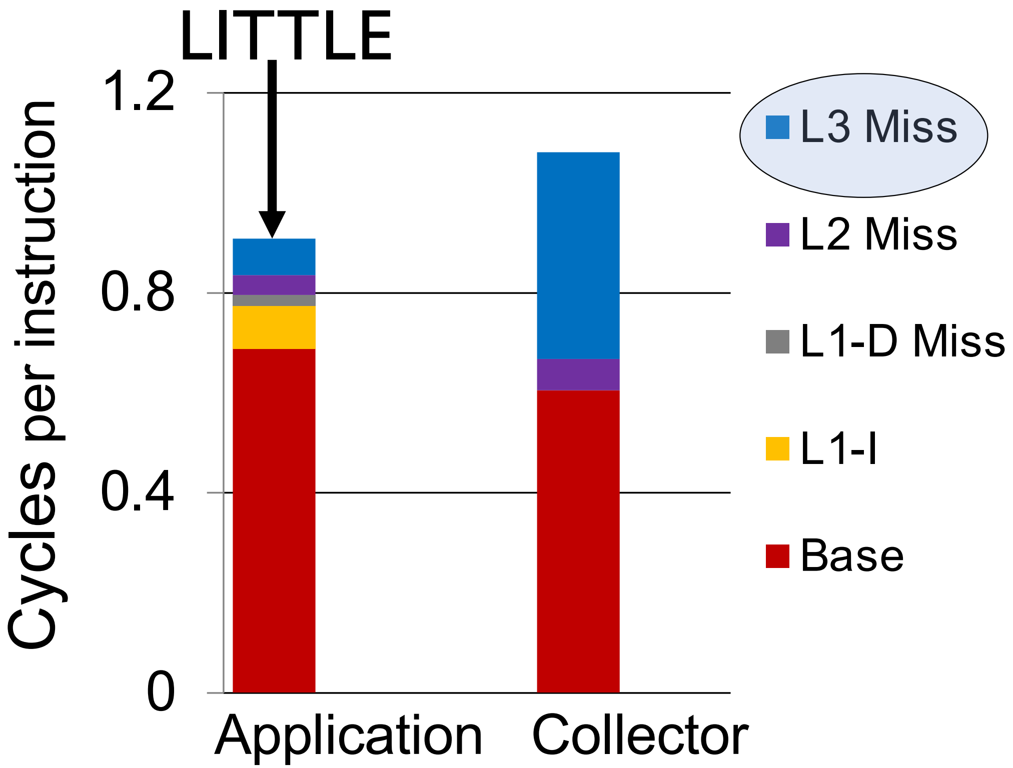
Importance of Hypothesis

- You should have a theory to answer the question
  - Current predictors are inaccurate because of large # methods
  - Current allocators incur high latency because applications allocate objects with variable sizes leading to fragmentation
  - Current CPUs are memory-bound for ML workloads

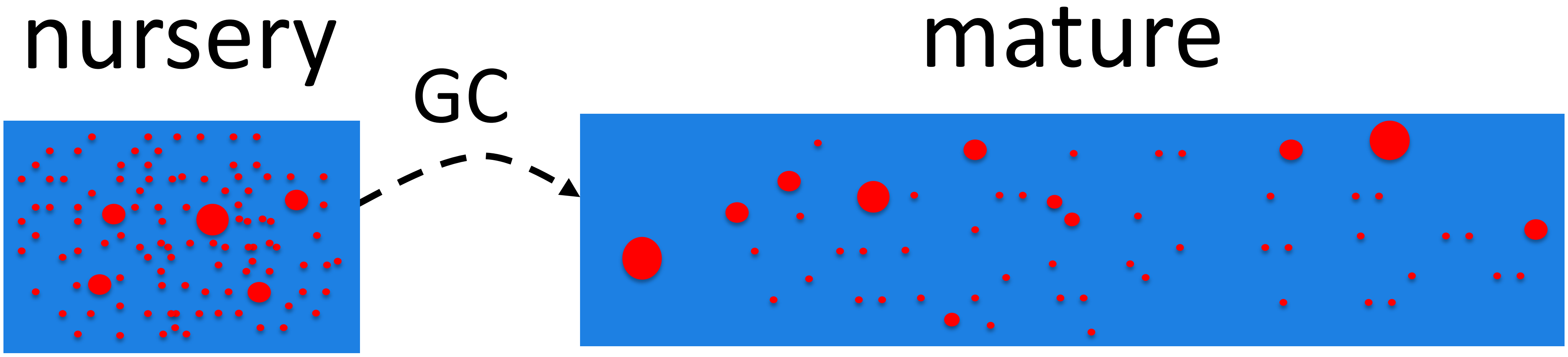
- Testing the hypothesis
  - Representative applications (benchmarks)
  - Real machine or simulator
  - Gain insight into program-machine interaction

Testing Hypothesis

- **CPI stack:** Breaks down execution time into different components at the microarchitectural level



Another Example



70%

of writes

22%

to 2% of objects

Picking Problems

- Must pick important problems. Ask Why frequently
- Questions that someone cares about
  - Hopefully, an entire community
- Enable new applications
- Keep an eye on where technology will go
- Aim high!

Aim High

- Try to contribute something novel as an undergrad
- You learn a lot. Research could lead to impact.

Best paper candidate, honors project

Analyzing and Improving the Scalability of In-Memory Indices for Managed Search Engines

Aditya Chilukuri
aditya.chilukuri@anu.edu.au
Australian National University
Canberra, ACT, Australia

Shoaib Akram
shoaib.akram@anu.edu.au
Australian National University
Canberra, ACT, Australia

Abstract

Managed search engines, such as Apache Solr and Elasticsearch, host huge inverted indices in main memory to offer fast response times. This practice faces two challenges. First, limited DRAM capacity necessitates search engines aggressively compress indices to reduce their storage footprint. Unfortunately, our analysis with a popular search library shows that compression slows down queries (on average) by up to 1.7× due to high decompression latency. Despite their performance advantage, uncompressed indices require 10× more memory capacity, making them impractical. Second, indices today reside off-heap, encouraging unsafe memory accesses and risking eviction from the page cache.

Emerging byte-addressable and scalable non-volatile memory (NVM) offers a good fit for storing uncompressed indices. Unfortunately, NVM exhibits high latency. We rigorously evaluate the performance of DRAM and NVM-backed compressed/uncompressed indices to find that an uncompressed index in a high-capacity managed heap memory-mapped over NVM provides a 36% reduction in query response times compared to a DRAM-backed compressed index in off-heap memory. Also, it is only 11% slower than the uncompressed index in a DRAM heap (fastest approach). DRAM and NVM-backed compressed (off-heap) indices behave similarly.

We analyze the narrow response time gap between DRAM and NVM-backed indices. We conclude that inverted indices demand massive memory capacity, but search algorithms exhibit a high spatial locality that modern cache hierarchies exploit to hide NVM latency. We show the scalability of uncompressed indices on the NVM-backed heap with large core counts and index sizes. This work uncovers new space-time tradeoffs in storing in-memory inverted indices.

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 the author(s) 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.
ISMM '23, June 18, 2023, Orlando, FL, USA
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0179-5/23/06.
https://doi.org/10.1145/3591195.3595272

CCS Concepts: • **Information systems** → **Search index compression**; Search engine indexing; • **Hardware** → **Memory and dense storage**; • **Software and its engineering** → *Garbage collection.*

Keywords: Text search, inverted index, persistent memory, compression, managed heap, garbage collection

ACM Reference Format:
Aditya Chilukuri and Shoaib Akram. 2023. Analyzing and Improving the Scalability of In-Memory Indices for Managed Search Engines . In *Proceedings of the 2023 ACM SIGPLAN International Symposium on Memory Management (ISMM '23), June 18, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3591195.3595272

1 Introduction

Search engines enable locating web pages on the internet and are a critical component of social media, professional networking, and e-commerce platforms. The key to retaining satisfied users is to offer low query response times. Amazon reports that even a 100 ms delay results in revenue drops [36]. Similar observations guide Google's search infrastructure.

The critical data structure search engines use for locating documents (web pages or social media posts) matching a word (term) is an inverted index. An inverted index maps unique terms to posting lists, where each posting stores an integer document identifier (ID) and meta-data (term frequency and position). Associating terms to posting lists using an inverted index speeds up query evaluation dramatically.

Today's standard practice is to host the inverted index in off-heap main memory. Recent work shows that even PCIe NVMe SSDs with byte-addressable 3D XPoint memory cannot deliver real-time response times [2]. Therefore, service providers keep indices in memory [60]. Unfortunately, as datasets grow, the inverted index grows proportionally, and large indices put increased pressure on DRAM. On the other hand, DRAM scaling cannot cope with the growth in datasets [20, 42]. Specifically, as data volume doubles yearly, the DRAM capacity only scales by 10% [24, 28]. The result is either the poor quality of service due to index lookups from storage or exorbitant memory-related expenditures.

Problem # 1 (High Decompression Latency): Compression is a crucial technique search engines use to store large indices in limited DRAM. For example, Apache Lucene uses a compression scheme that reduces index size by 85–90%,

under review at VLDB, research project, top conference

SPIRIT: Scalable and Persistent On-Heap Indices in Hybrid Memory for Real-Time Search

Shoaib Akram
Shoaib.Akram@anu.edu.au
Australian National University
Canberra, ACT, Australia

ABSTRACT

Today, real-time search over big microblogging data requires low indexing and query latency. Online services, therefore, prefer to host inverted indices in memory. Unfortunately, as datasets grow, indices grow proportionally, and with limited DRAM scaling, the main memory faces high pressure. Also, indices must be persisted on disks as building them is computationally intensive. Consequently, it becomes necessary to frequently move on-heap index segments to block storage, slowing down indexing. Reading storage-resident index segments necessitates filesystem calls and disk accesses during query evaluation, leading to high and unpredictable tail latency.

This work exploits (hybrid) DRAM and scalable non-volatile memory (NVM) to offer dynamically growing, and instantly searchable, large persistent indices in on-heap memory. We implement our proposal in SPIRIT, a real-time text inversion engine over hybrid memory. SPIRIT exploits the byte-addressability of hybrid memory to enable direct access to the index on a pre-allocated heap, eliminating expensive block storage accesses and filesystem calls during live operation. It uses fast persistent pointers in a global descriptor table to offer: ❶ instant segment availability to query evaluators upon fresh ingestion, ❷ low-overhead segment movement across memory tiers transparent to query evaluators, and ❸ decoupled segment movement into NVM from their visibility to query evaluators, enabling intelligent policies for mitigating high NVM latency. SPIRIT accelerates compaction with zero-copy merging and supports fast, graceful shutdown and instant recovery.

With 15% and 50% of the index in DRAM, it concurrently resolves queries only 13% and 6.44% slower, respectively, compared to DRAM alone. Our work generalizes to other data-intensive services that will benefit from direct on-heap access to large persistent indices.

PVLDB Reference Format:
Adnan Hasnat and Shoaib Akram. SPIRIT: Scalable and Persistent On-Heap Indices in Hybrid Memory for Real-Time Search. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/vics1/SPIRIT.

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

1 INTRODUCTION

Today, enabling fast real-time search over social media content is critical to the success of many enterprises, including Linkedin, Meta, and Twitter [8, 52]. Social media content is either queried explicitly for *relevance search*, similar to static web content, or implicitly by the service for *timeline retrieval* to populate a user's home feed. The latter generates queries based on the user's preferred topics or followers and is more frequent on social media platforms — the source of 50% of tweets recommended on Twitter's `For You` and `Following` tabs [69]. The queries run concurrently with an indexing engine that builds indices in real-time, coping with a massive volume of data, e.g., 500 million Tweets per day for Twitter [69]. Consequently, real-time indexing puts high pressure on the aggregate CPU and memory of the real-time cluster, and concurrent queries exacerbate the pressure further [3, 52].

The critical data structure search engines use for locating documents (web pages, social media posts, or tweets) matching a word (term) is an inverted index [84]. Offering real-time response times requires hosting the index in memory [28, 75]. Unfortunately, indices grow proportional to datasets, and large indices put pressure on DRAM. However, DRAM scaling cannot cope with the growth in datasets [1, 11, 14, 28, 29, 33, 57, 61, 72], increasing infrastructure cost [15, 72]. Scaling in-memory indices to large datasets demands dense memory technologies, complementing DRAM.

Hosting indices in memory is also at odds with persisting them in today's storage stack. Popular search engines, e.g., Apache Solr [22] and Elasticsearch [20] use a segmented index, and each fixed-size segment resides on the heap before being moved to the page cache. Segments are buffered in the cache (no synchronous I/O per segment) to amortize I/O overheads. Ultimately, calling `fsync` to bulk-persist segments on storage, an operation called *commit* in Elasticsearch, becomes necessary to avoid losing significant index updates on a crash [5, 20]. Unfortunately, if the index outgrows available DRAM, later reads of the persistent segment generate I/O transfers. Recent work shows that even the fastest PCIe NVMe SSDs cannot deliver the response times required by real-time search [3, 52, 69]. Furthermore, an `fsync` is costly and cannot be performed without a significant performance hit [20]. Existing systems either risk losing a significant state or paying a performance penalty.

Due to limited DRAM and the need for persistence, block storage is deeply integrated into real-time search clusters. Therefore, popular real-time search engines do not make on-heap segments visible to query evaluators [20]. Instead, query evaluators use multiple filesystem calls to access new segments from the OS cache or storage. (In Elastic search, one set of calls is to read the commit point and another to read the index segment [20].) These calls incur high overhead [9, 38], prohibiting real-time operation [20]. Furthermore,

Some Tenets of Systems Research

- Good abstractions are powerful. **In fact, this is why computer systems work. (**And why I start every semester with: Alice has an idea to save the world. How can she orchestrate the movement of electrons with English. She cannot. She uses?**)**

- Yet, many great ideas come from understanding the interaction between abstraction layers

- Insight is key. Go to class for insight. Read (critically) for insight. Do research for insight. Communicate to gain and give insight. If something "just works," and you do not understand **WHY**, it's useless. (When it breaks, you can't fix it.)

- Good engineering in systems research is necessary, but the goal of research is to communicate new insights. No one is interested in how you fixed bugs in your code. (**Analogy:** Fertilizer is critical for growing pretty roses, but we don't decorate our house with fertilizer. **Try it and no guests will come again.** Same with research, tell people about "boring" engineering details, and they won't listen to you again. **YET** must decide how much they need to know.)

- Designing new systems is somewhat of an art. (Technology "pull," and application "push.") Must learn from prior art/design, i.e., **precedents** (COMP2300/COMP2310/Microarch.). Must use **creativity** to adapt to new changing technology trends and new workloads. **Two things systems researcher must live with:** physics (speed of light, how small a transistor can me made, and still be used reliably, yield of an X mm^2 chip) and society (big data due to microblogging, social media, and online payments; use of AI/ML; purchasing power; Netflix vs. renting video; cloud vs. in-house)

- Device physics shapes the system
  - Early filesystem research. Moore's law and its impact on systems. Persistent memory. Distributed systems. What enabled multi-layer software stacks (think Scala)? What threatens them now? Shift to multicore. Disk vs. Flash

Systems Architect’s Toolbox: Design Side

- Know the precedents (what techniques worked in prior systems)?
  - Caching, prediction, ISA additions, speculation, write batching, sequential log, tracing collector, write barrier, spin lock, interrupt, MMU

- Know the “key” tradeoffs
  - Compression saves storage capacity but decompression incurs high latency
  - RISC ISA simplifies circuit complexity, but results in more instructions per C/C++ statement (pressure on instruction memory)
  - Disk is cheap but its latency is high
  - SRAM is fast but consumes more power

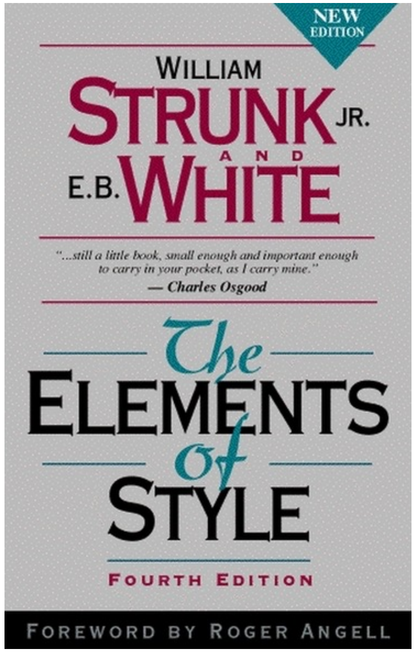
- Know the “critical” metrics
  - Performance, power, energy, reliability, security, extendibility, observability, manageability, cost, scalability, throughput, tail latency

Engineering Side

- Holistic view of system
  - Good comprehension of CPU, memory, and disk datapath. Byte-addressable vs. block addressable. Virtual memory. Virtualizing CPU.
- Good programming skills in one or more languages and ability to pick a new language quickly
- Good systems building skills (compiling the Linux kernel, using GCC/GDB, writing Makefile, hacking OpenJDK)
- Data structures and algorithms
- Performance debugging
  - Monitoring low-level processor performance
  - I/O traffic monitoring tools

Writing and Presentation

Advice on Writing

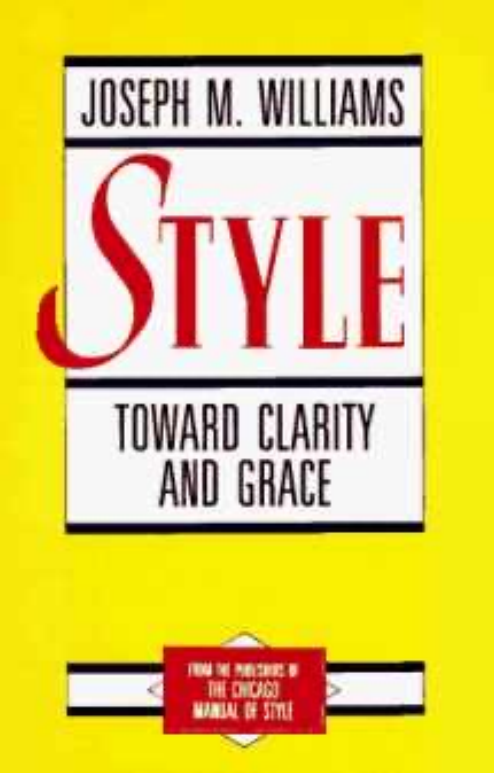

NEW EDITION
WILLIAM STRUNK JR. AND E.B. WHITE
"...still a little book, small enough and important enough to carry in your pocket, as I carry mine." — Charles Osgood
The ELEMENTS of STYLE
FOURTH EDITION
FOREWORD BY ROGER ANGELL

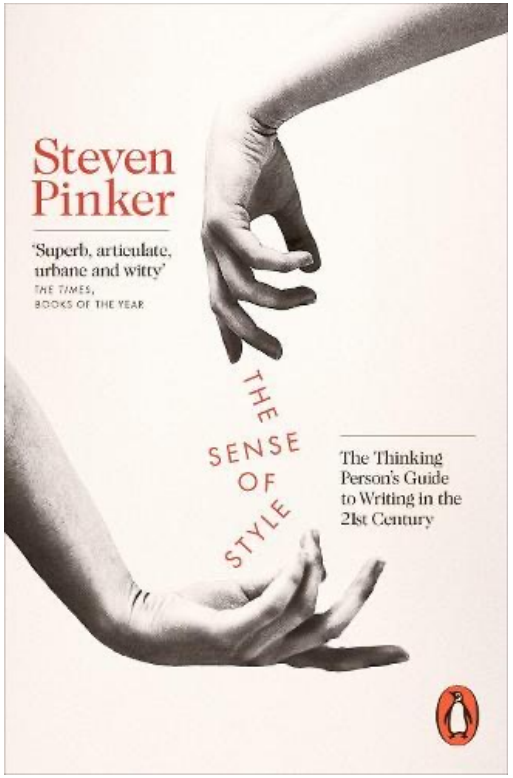
👎

Advice by "prescription"

"Trust me." Do X. Do Y

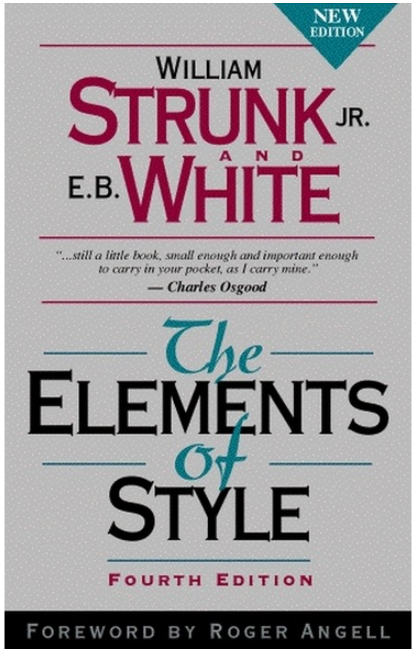
👍 👍




Steven Pinker
'Superb, articulate, urbane and witty'
THE TIMES, BOOKS OF THE YEAR
THE SENSE OF STYLE
The Thinking Person's Guide to Writing in the 21st Century

- Advice based on "insight"
- What is the purpose of writing?
- What do humans consider good writing?
- Why one writing style is more powerful than other?
- How **"attention mechanics"** work? Invoking stress

Advice on Writing



NEW EDITION

WILLIAM STRUNK JR.
AND
E.B. WHITE

"...still a little book, small enough and important enough to carry in your pocket, as I carry mine."
— Charles Osgood

The ELEMENTS of STYLE

FOURTH EDITION

FOREWORD BY ROGER ANGELL

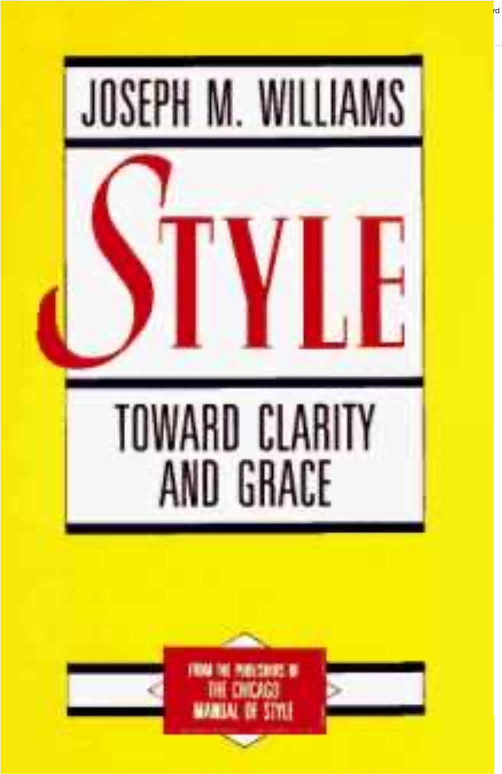
- Passive voice is best avoided
- Don’t end a sentence with preposition
- **And many more prescriptions**
  - **Analogy:** temporary relief, no pinpointing the real source of pain, no diagnosis

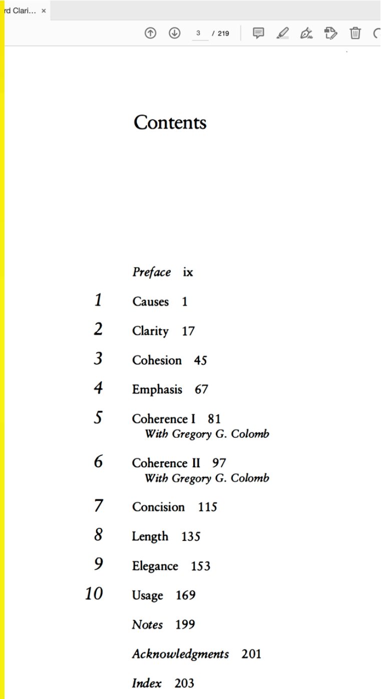
- **Reality**
  - Passive serves an important role
  - OK to end with prep.

- **Key realization in style community:** Passive and preps. alone don’t put people off. There are more fundamental issues to be dealt with. **And they relate to a system of style that must be understood**

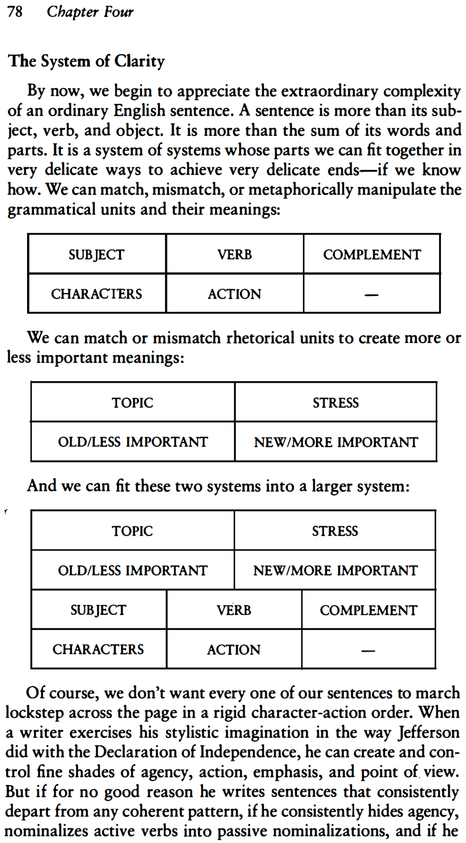
Advice on Writing

👍





Contents



78 *Chapter Four*

The System of Clarity

By now, we begin to appreciate the extraordinary complexity of an ordinary English sentence. A sentence is more than its subject, verb, and object. It is more than the sum of its words and parts. It is a system of systems whose parts we can fit together in very delicate ways to achieve very delicate ends—if we know how. We can match, mismatch, or metaphorically manipulate the grammatical units and their meanings:

SUBJECT	VERB	COMPLEMENT
CHARACTERS	ACTION	—

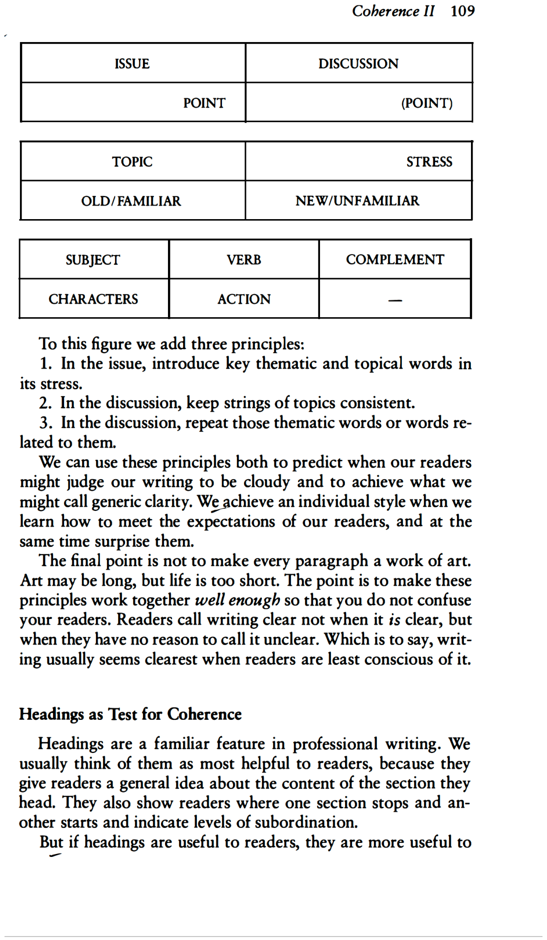
We can match or mismatch rhetorical units to create more or less important meanings:

TOPIC	STRESS
OLD/LESS IMPORTANT	NEW/MORE IMPORTANT

And we can fit these two systems into a larger system:

TOPIC		STRESS
OLD/LESS IMPORTANT		NEW/MORE IMPORTANT
SUBJECT	VERB	COMPLEMENT
CHARACTERS	ACTION	—

Of course, we don't want every one of our sentences to march lockstep across the page in a rigid character-action order. When a writer exercises his stylistic imagination in the way Jefferson did with the Declaration of Independence, he can create and control fine shades of agency, action, emphasis, and point of view. But if for no good reason he writes sentences that consistently depart from any coherent pattern, if he consistently hides agency, nominalizes active verbs into passive nominalizations, and if he



Coherence II 109

ISSUE	DISCUSSION
POINT	(POINT)

TOPIC	STRESS
OLD/FAMILIAR	NEW/UNFAMILIAR

SUBJECT	VERB	COMPLEMENT
CHARACTERS	ACTION	—

To this figure we add three principles:

1. In the issue, introduce key thematic and topical words in its stress.
2. In the discussion, keep strings of topics consistent.
3. In the discussion, repeat those thematic words or words related to them.

We can use these principles both to predict when our readers might judge our writing to be cloudy and to achieve what we might call generic clarity. We achieve an individual style when we learn how to meet the expectations of our readers, and at the same time surprise them.

The final point is not to make every paragraph a work of art. Art may be long, but life is too short. The point is to make these principles work together *well enough* so that you do not confuse your readers. Readers call writing clear not when it *is* clear, but when they have no reason to call it unclear. Which is to say, writing usually seems clearest when readers are least conscious of it.

Headings as Test for Coherence

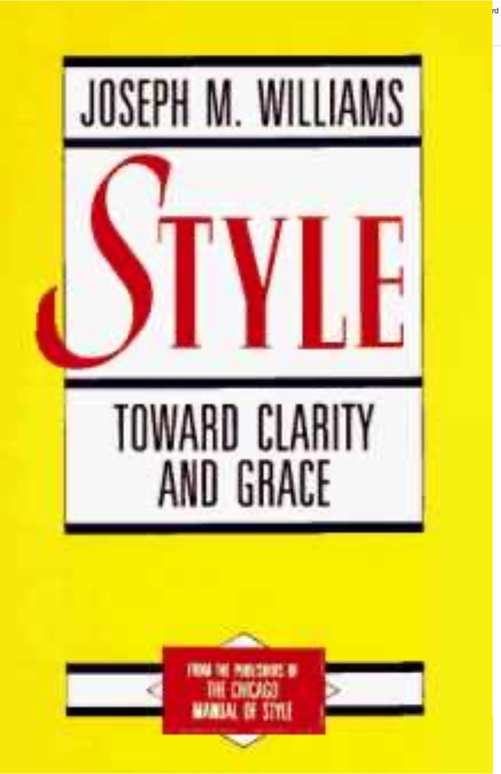
Headings are a familiar feature in professional writing. We usually think of them as most helpful to readers, because they give readers a general idea about the content of the section they head. They also show readers where one section stops and another starts and indicate levels of subordination.

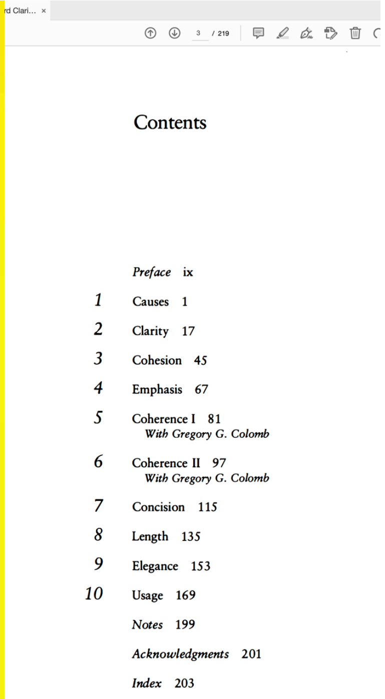
But if headings are useful to readers, they are more useful to

Advice based on "insight"

Advice on Writing

👍





Contents

Advice based on “insight”

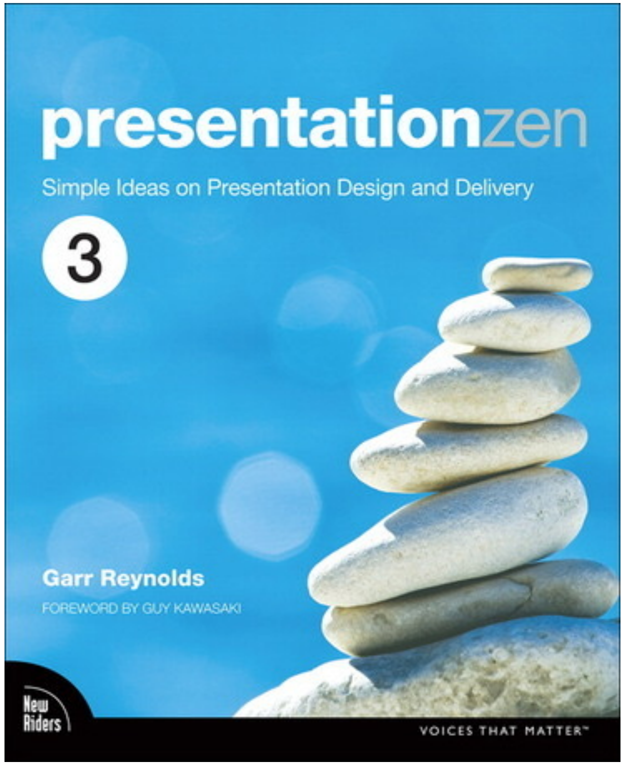
The First Two Principles of Clear Writing
Readers are likely to feel that they are reading prose that is dear and direct when

(1) the **subjects** of the sentences name the cast of **characters**, and

(2) the **verbs** that go with those subjects name the **crucial actions** those characters are part of.

Presenting Research Outcomes

👍



Advice based on “insight”

One main idea per slide

Few bullets

Good titles (some examples later)

Figures clearly annotated

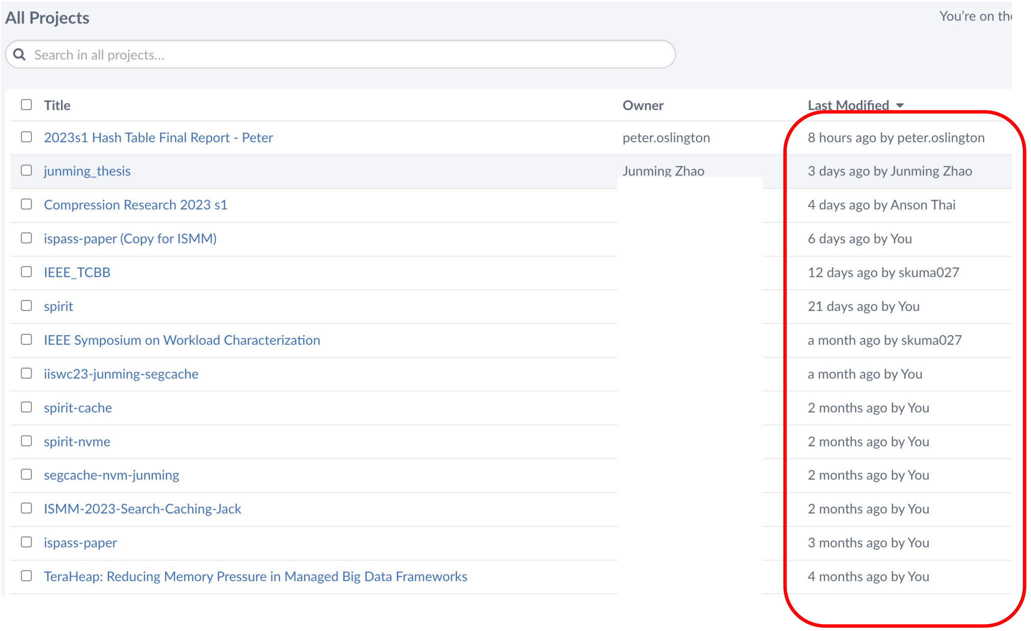
One slide to the next (story telling)

My students create a slide deck. I can write an entire paper without bugging them too much by just following the slide deck

Latex and Overleaf

Learn Latex

Start collaborating with your advisor on Overleaf



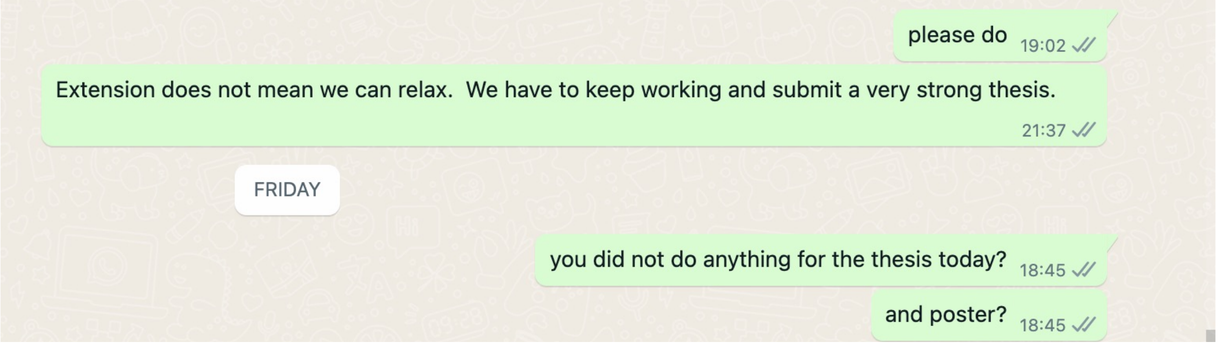
All Projects

You're on th

Search in all projects...

□ Title	Owner	Last Modified ▾
□ 2023s1 Hash Table Final Report - Peter	peter.oslington	8 hours ago by peter.oslington
□ junming_thesis	Junming Zhao	3 days ago by Junming Zhao
□ Compression Research 2023 s1		4 days ago by Anson Thai
□ ispass-paper (Copy for ISMM)		6 days ago by You
□ IEEE_TCBB		12 days ago by skuma027
□ spirit		21 days ago by You
□ IEEE Symposium on Workload Characterization		a month ago by skuma027
□ iiswc23-junming-segcache		a month ago by You
□ spirit-cache		2 months ago by You
□ spirit-nvme		2 months ago by You
□ segcache-nvm-junming		2 months ago by You
□ ISMM-2023-Search-Caching-Jack		2 months ago by You
□ ispass-paper		3 months ago by You
□ TeraHeap: Reducing Memory Pressure in Managed Big Data Frameworks		4 months ago by You

He will help you stay focused



please do 19:02

Extension does not mean we can relax. We have to keep working and submit a very strong thesis. 21:37

FRIDAY

you did not do anything for the thesis today? 18:45

and poster? 18:45

Systems Papers

Systems Papers: One Classification

- Architecture
- Runtimes for PL
- Memory management
- Distributed databases
- Graph analytics
- Compilers
- Many more areas ...

Systems Papers: Another Classification

- New “**systems idea”**
  - New mechanism
  - New policy

- Performance **analysis** and **evaluation**
  - Evaluate existing/emerging hardware
    - Specific features
    - Full system (holistic)
  - Evaluate existing/emerging workloads
    - Specific phases
    - Full workload

- Analytical and mechanistic modeling
  - Enable **new insights** (by fast exploration)
  - Enable **new policies** that are rigorously understood (contrast with “ML magic”)

Mechanism vs. Policy

- **Sharing a CPU among many users**
  - Mechanism: Changing PC_{user1} to PC_{user2} and other actions to switch to executing process from user 2
  - Policy: When to **switch** from one user to the next, which user to give priority, cloud vs. desktop

- **Using disk as an extension of main memory (swapping)**
  - Mechanism: Copying data from memory to disk, physical hardware changes, pins, wires, interrupts, system calls, all that jazz
  - Policy: When to **initiate a transfer** from memory to disk (when memory is critically low, when memory is 80% of capacity, ...)

- **Offloading computation to a GPU**
  - Mechanism: Introducing GPU in the system, setting up CPU-GPU communication, etc
  - Policy: What to offload? When to offload? If the GPU is busy, **what is the policy to offload another waiting task?**

Importance of Performance Evaluation

- Why do we evaluate performance?
  - To understand if we can build better systems for a specific workload
  - To understand if we are enabling needless features
  - To understand how can we improve the system

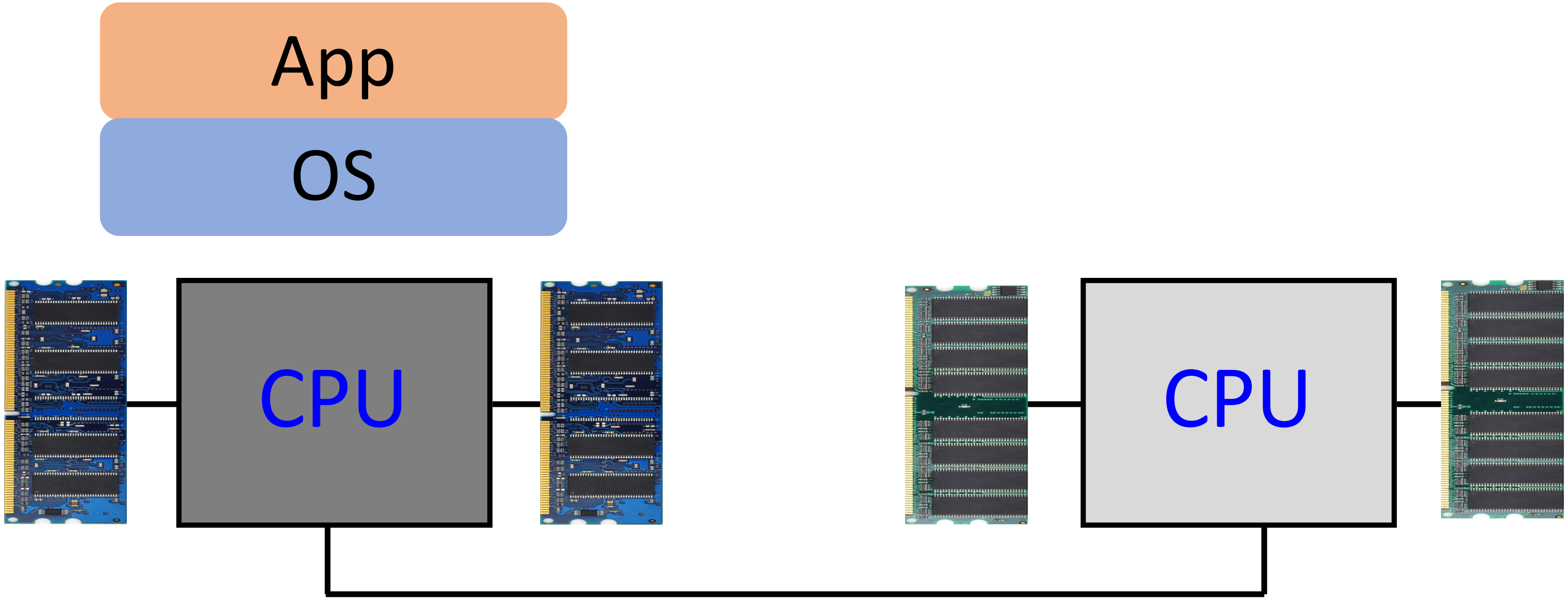
- Hardware is available
  - Do a real system study

- Hardware is not available
  - Use simulation (e.g., model the behavior of the system in C++)
    - Cycle accurate (very time consuming) vs. mechanistic model (fast but not very accurate)
  - Use emulation
    - Emulate the "unavailable system" using an existing system

Example of Emulation

- NUMA to model a hybrid DRAM-PCM system

- Frequency scaling to model a big.LITTLE system



Example of Simulation

- Sniper multicore simulator we use in Microarchitecture Course

Importance of Modeling

- Gain insight
  - How does a system work?
  - A high-level model of an out-of-order processor

uarch-indep branch predictor model [De Pestel, ISPASS'15]

total cycle count

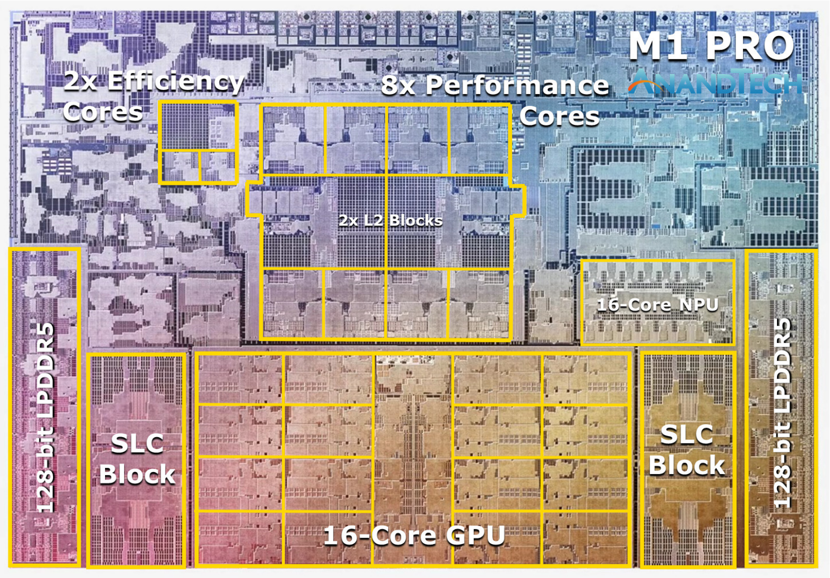
miss rates predicted using StatStack [Eklov and Hagersten, ISPASS'10]

$$C = \overbrace{\frac{N}{D_{\text{eff}}}}^{\textbf{Base}} + \underbrace{m_{\text{bpred}} \times (c_{\text{res}} + c_{\text{fr}})}_{\textbf{Branch}} + \overbrace{\sum_{\text{level}=i} m_{\text{ILi}} \times c_{\text{Li+1}}}^{\textbf{I-cache}} + \underbrace{\frac{m_{\text{LLC}} \times c_{\text{mem}}}{\text{MLP}}}_{\textbf{D-cache}}$$

uarch-indep MLP model [Van den Steen, CAL'18]

N = dynamic instruction count

D_{eff} = effective dispatch rate; is function of ILP, I-mix, ALU contention



Importance of Modeling

- Quickly explore large design space in early stage of design
  - Simulators are extremely slow
  - In early stages, only need to know relative performance
  - To filter out **parameter settings** (for example, cache size) that do not show good trends

New Idea Papers

- Let’s look at some top-tier idea papers from my recent work

- TeraHeap: Reducing Memory Pressure in Managed Big Data Frameworks
  - ASPLOS 2023

- Write-Rationing Garbage Collection for Hybrid Memories
  - PLDI 2019

- SPIRIT: Scalable and Persistent On-Heap Indices in Hybrid Memory for Real-Time Search
  - Under Review

What’s in a title?

- Succinct. To the point. Stress the key contribution. Good verbs. Good adjectives.
- Typically include software aspect and a hardware aspect
- Find a “decent” & memorable name. But if you can’t, don’t force a name, or have one that is pointless

- **TeraHeap:** Reducing Memory Pressure **in** Managed Big Data Frameworks
  - ASPLOS 2023

- Write-Rationing Garbage Collection **for** Hybrid Memories
  - PLDI 2019

- **SPIRIT:** Scalable and Persistent On-Heap Indices in Hybrid Memory **for** Real-Time Search
  - Under Review

Other Papers & Presentations

- Let’s see some other papers

- Let’s see some presentations

Write-Rationing Garbage Collection for Hybrid Memories

Shoaib Akram (Ghent), Jennifer B. Sartor (Ghent and VUB), Kathryn S. Mckinley (Google), and Lieven Eeckhout (Ghent)

Shoaib.Akram@UGent.be

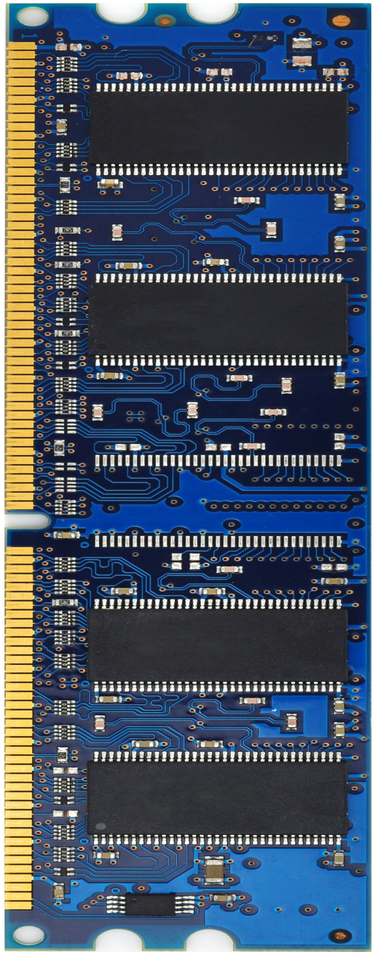
DRAM is facing challenges

Scalability

Cost

Energy

Reliability



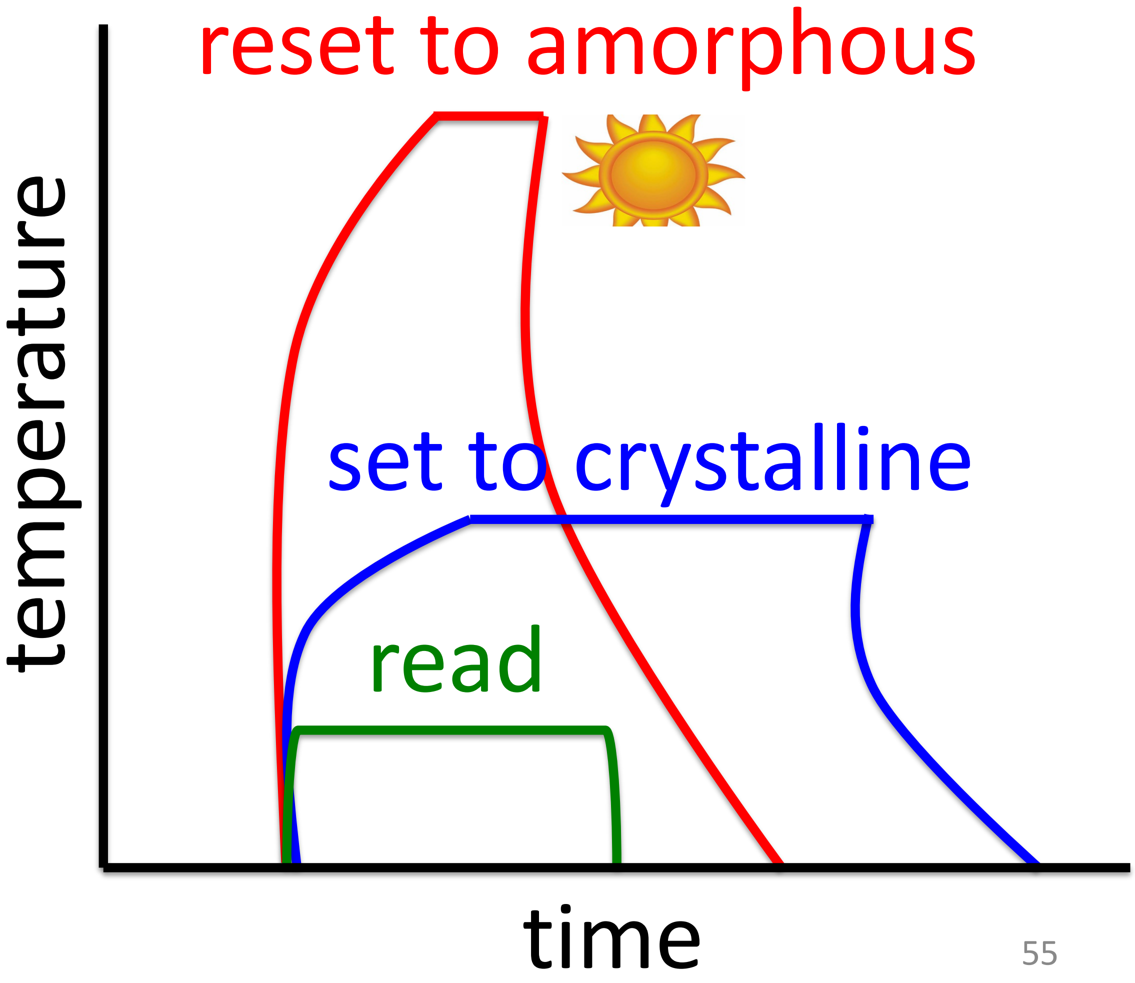
Phase change memory

Persistent

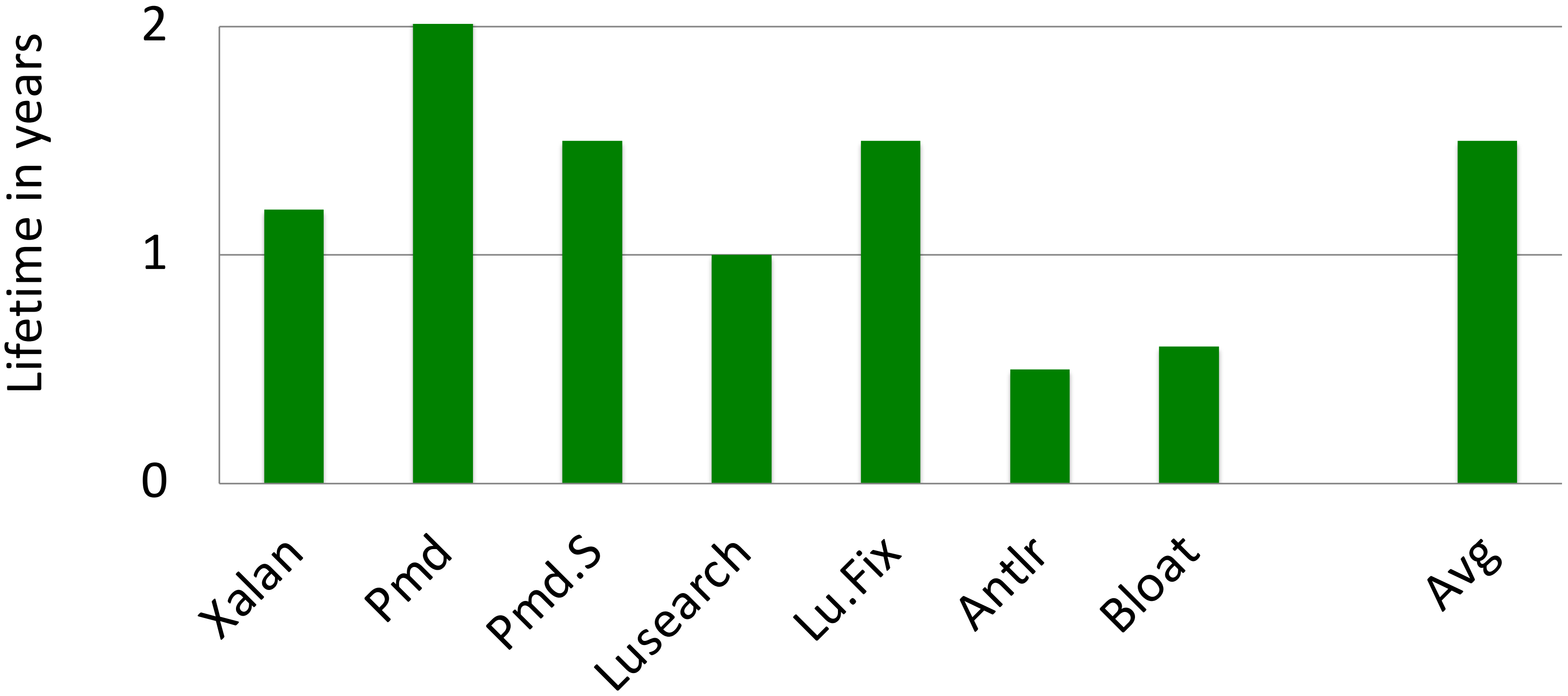
Byte addressable

High latency

Low endurance

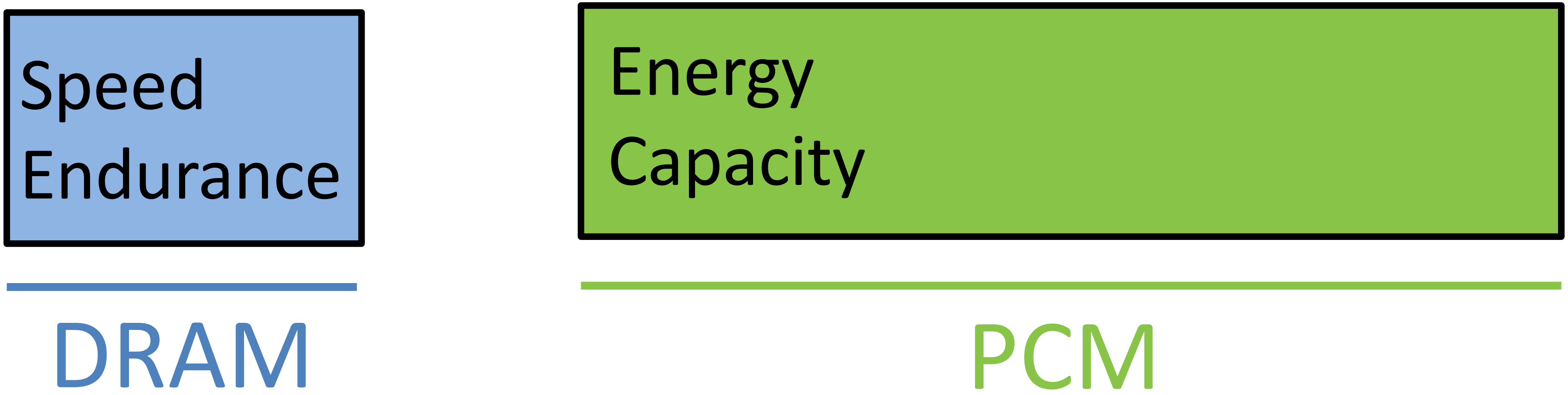


PCM only is not practical



32 GB PCM with hardware wear-levelling

Hybrid DRAM-PCM memory



Speed
Endurance

DRAM

Energy
Capacity

PCM

Challenges

Bridging the DRAM-PCM latency gap

Mitigating PCM wear-out

Prior art in mitigating PCM wear-out

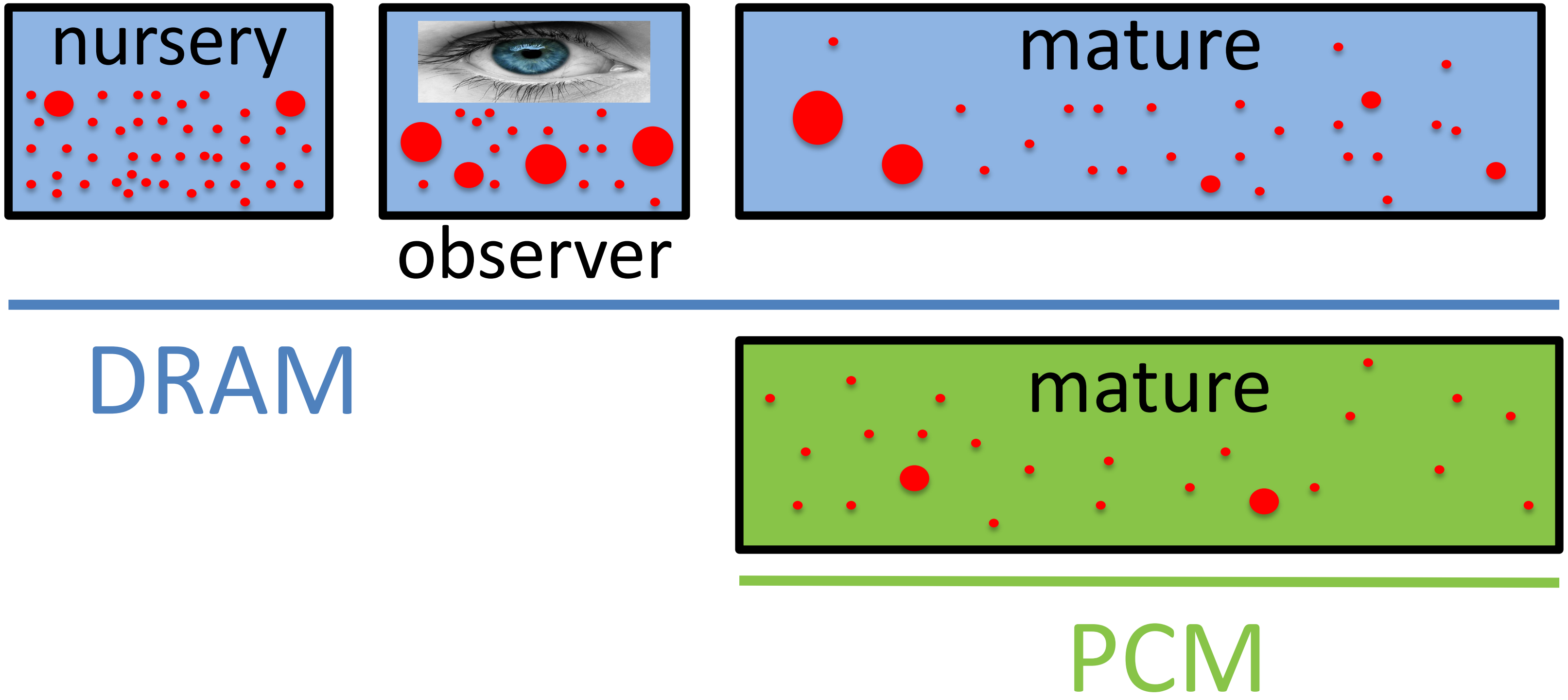
Hardware wear-leveling

- Spread writes out across PCM
- 32 GB PCM lasts only two years!

OS write partitioning

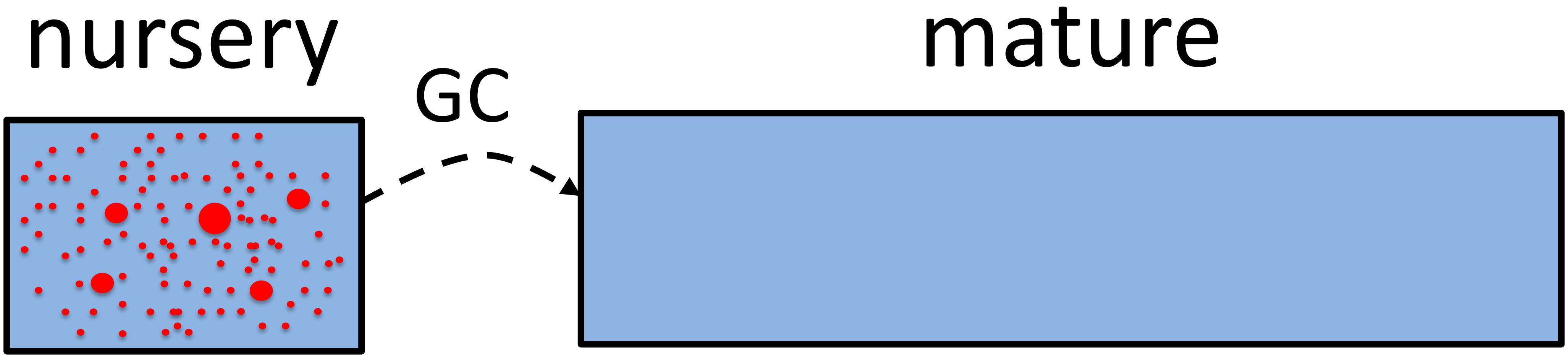
- Keep highly written pages in DRAM
- Coarse granularity
- Costly page migrations

Garbage collection for hybrid memory



This work uses GC to keep highly written objects in DRAM

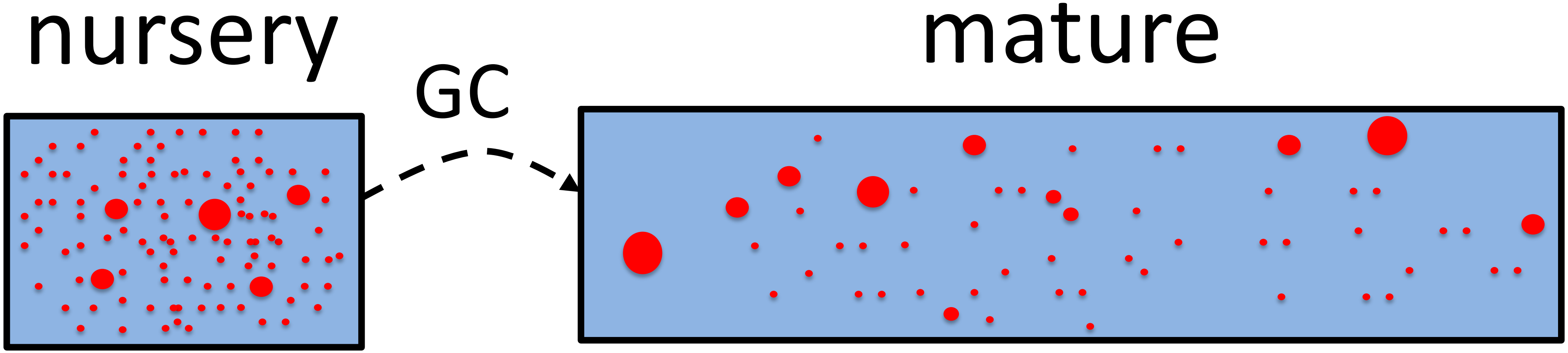
Distribution of writes in GC heaps



70%

of writes

Distribution of writes in GC heaps



nursery

GC

mature

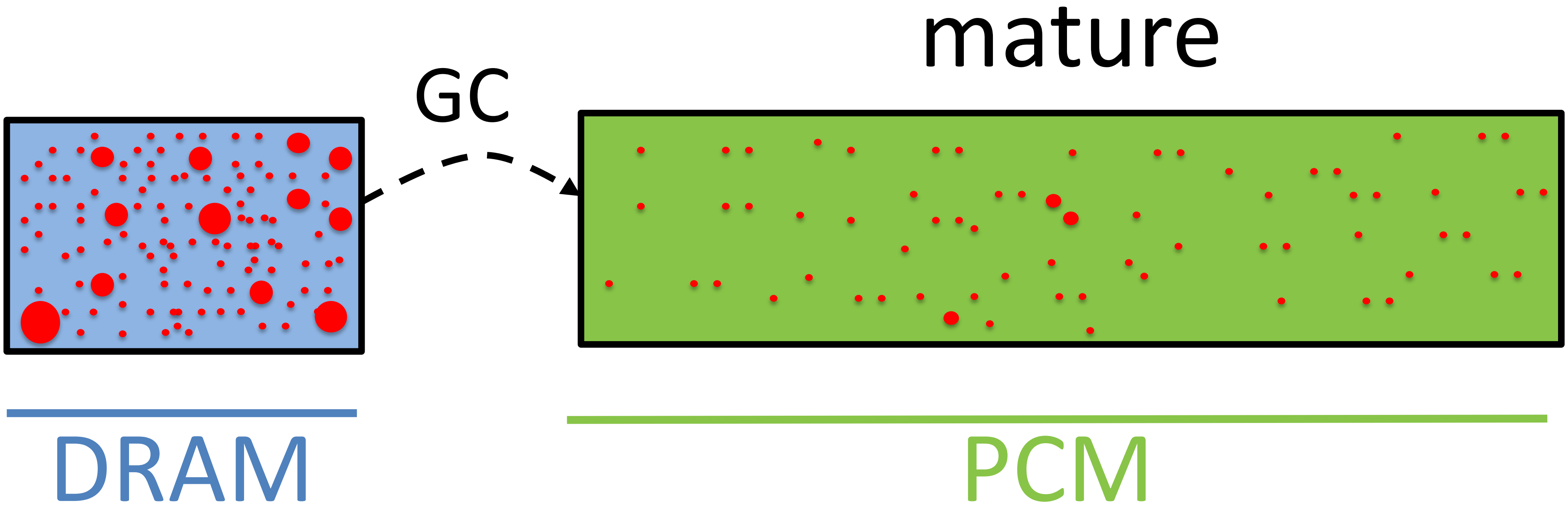
70%

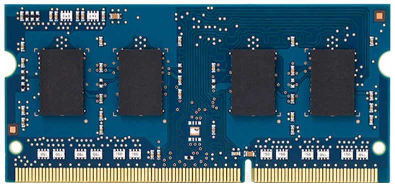
of writes

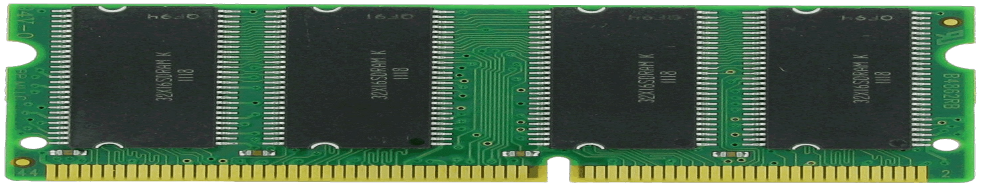
22%

to 2% of objects

Contribution
Write-Rationing Garbage Collectors





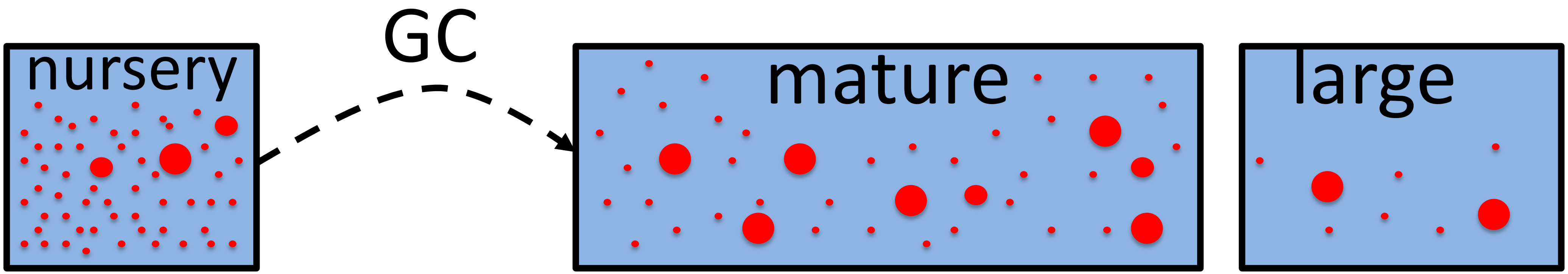


Two write-rationing garbage collectors

Kingsguard-Nursery

Kingsguard-Writers

Heap organization in DRAM



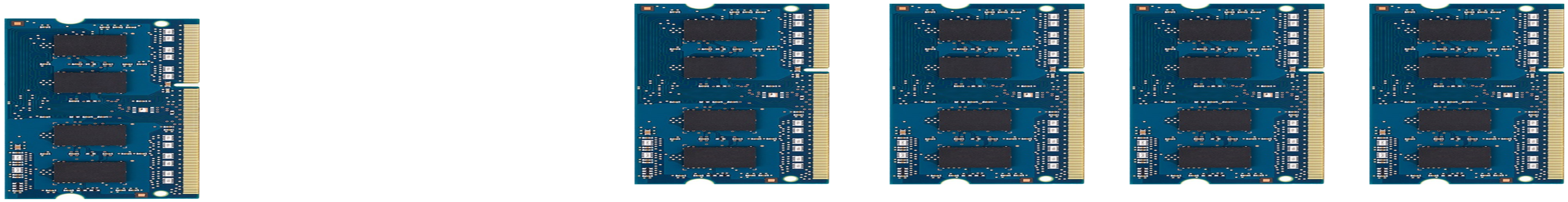
GC

nursery

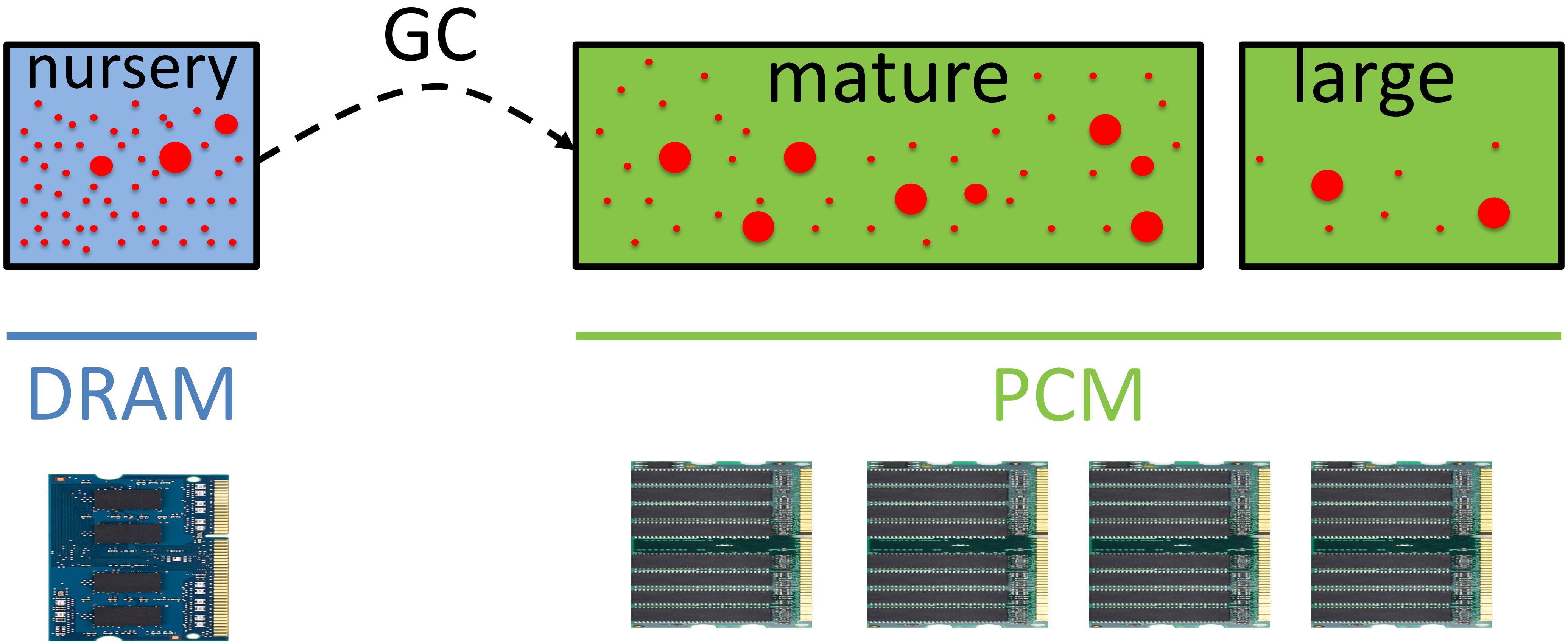
mature

large

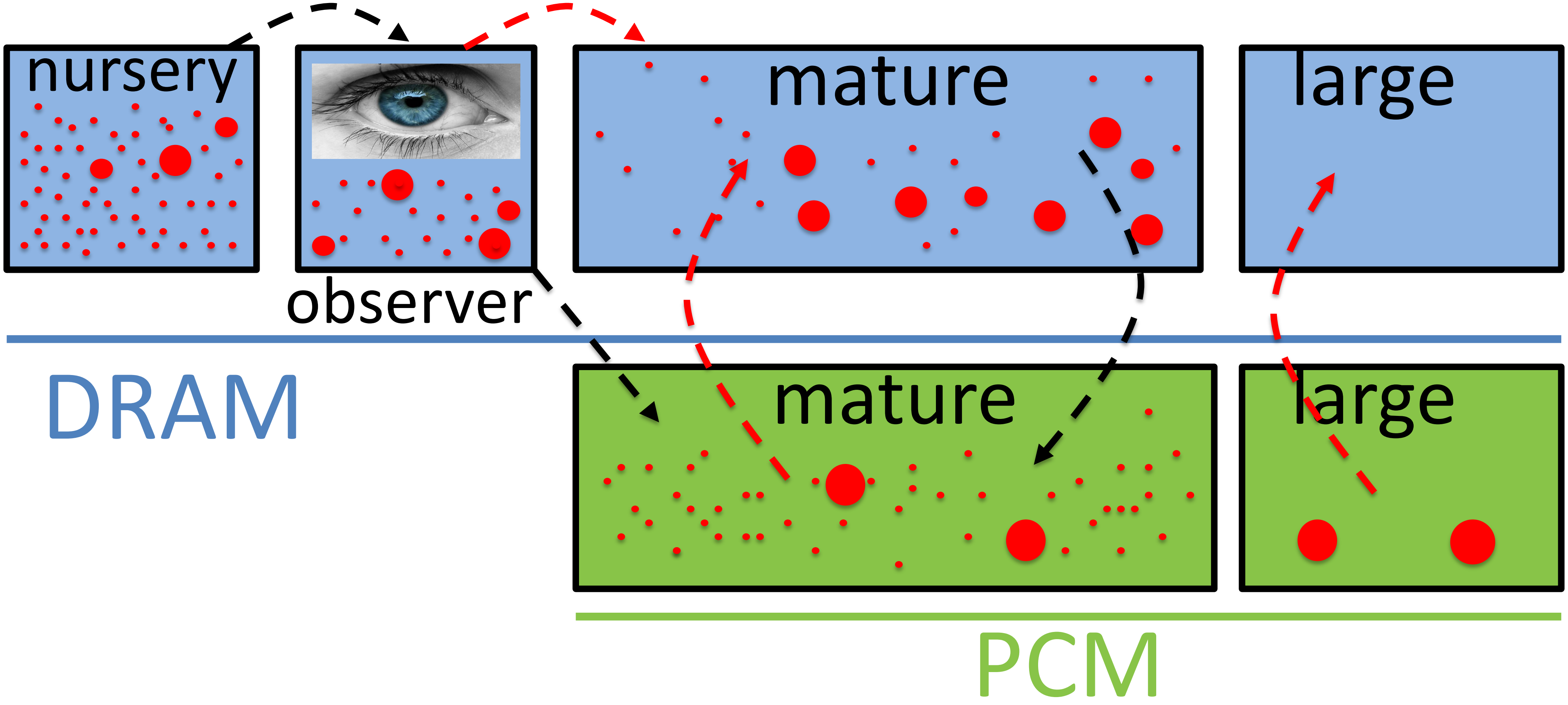
DRAM



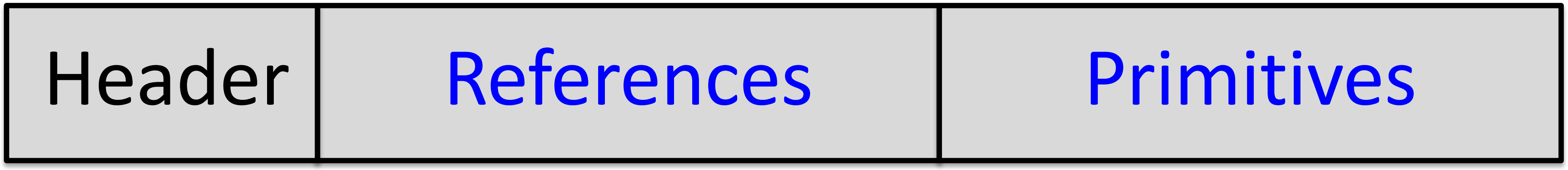
KG-N Kingsguard-Nursery



KG-W Kingsguard-Writers



Monitoring writes



Header | References | Primitives

On a write to an object

Write barrier sets a bit in header

Write barrier configurations

Monitor references

Monitor references and primitives

Two additional optimizations

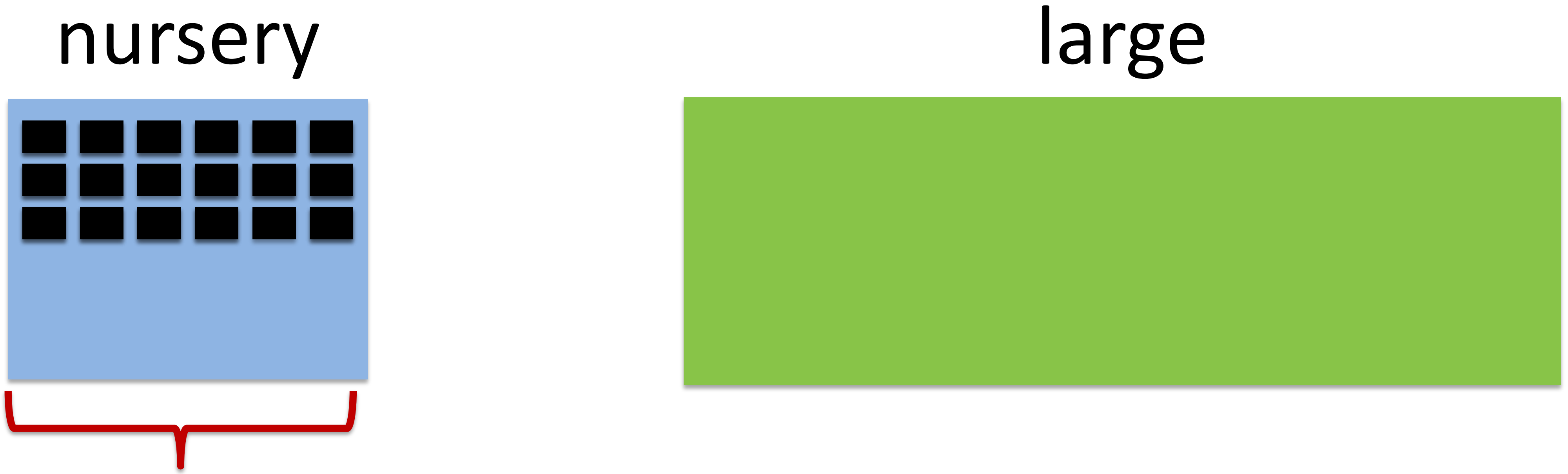
Large object optimization

Selectively allocate large objects in DRAM

Metadata optimization

Place mark bits of PCM objects in DRAM

Large object optimization



nursery

large

½ of remaining nursery

Monitor PCM write rate to tu[redacted]

Results

(1) Measurements on real hardware

(2) Simulation

Jikes research virtual machine

Java applications

Real hardware methodology

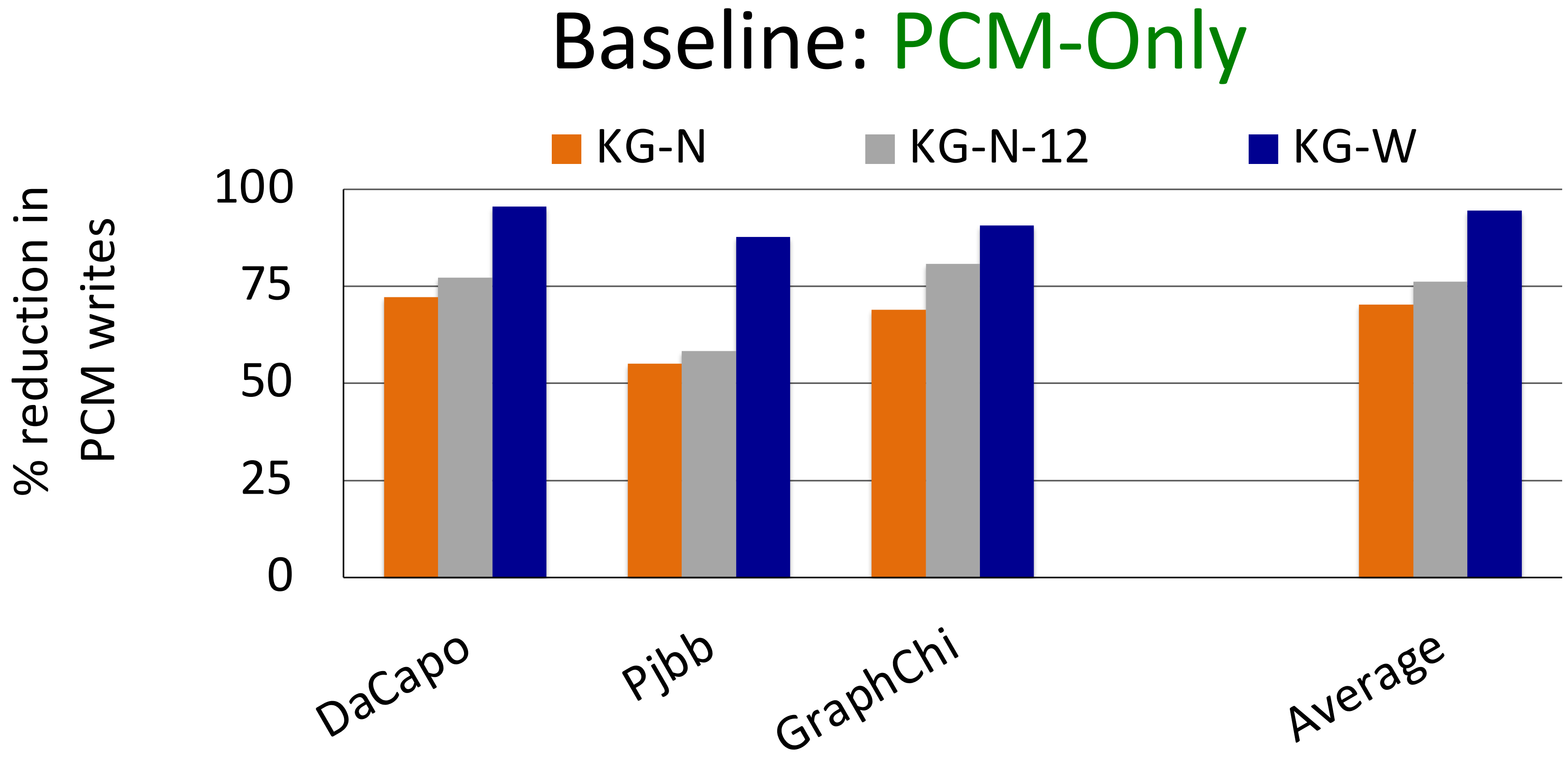
Use write barriers to count object writes

Applications: 12 DaCapo, 3 GraphChi, and Pjbb

Configurations

- KG-N : 4 MB nursery
- KG-W: 4 MB nursery, 8 MB observer
- KG-N : 12 MB nursery

Reduction in PCM writes



KG-W reduces 95% of writes to PCM

Simulation methodology

7 DaCapo applications

Measure lifetime, energy, and execution time in simulator

Memory systems

Homogeneous

- 32 GB DRAM
- 32 GB PCM

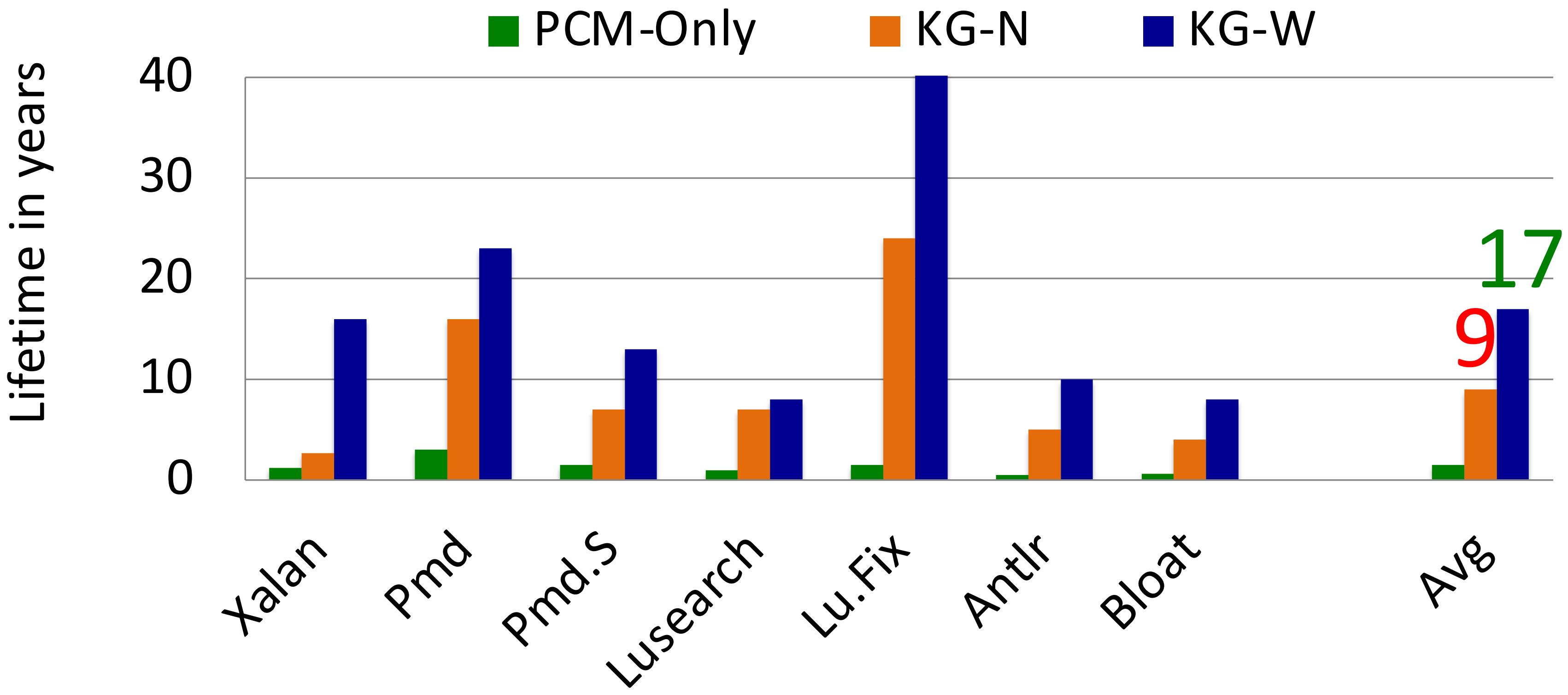
Hybrid

- 1 GB DRAM
- 32 GB PCM

PCM parameters

- 4X read latency
- 4X write energy
- 10 M writes/cell

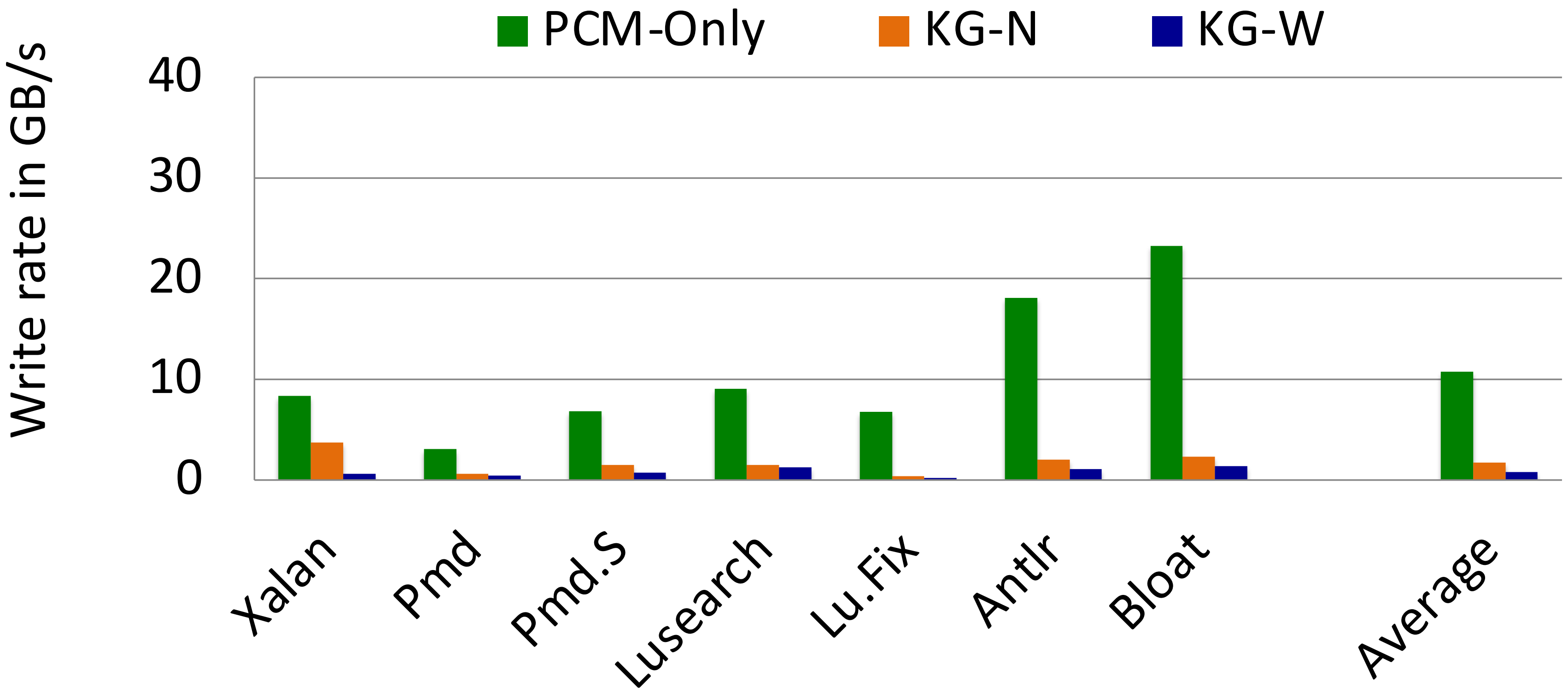
PCM lifetimes



PCM alone is not practical

PCM lasts more than 10 years with KG-W

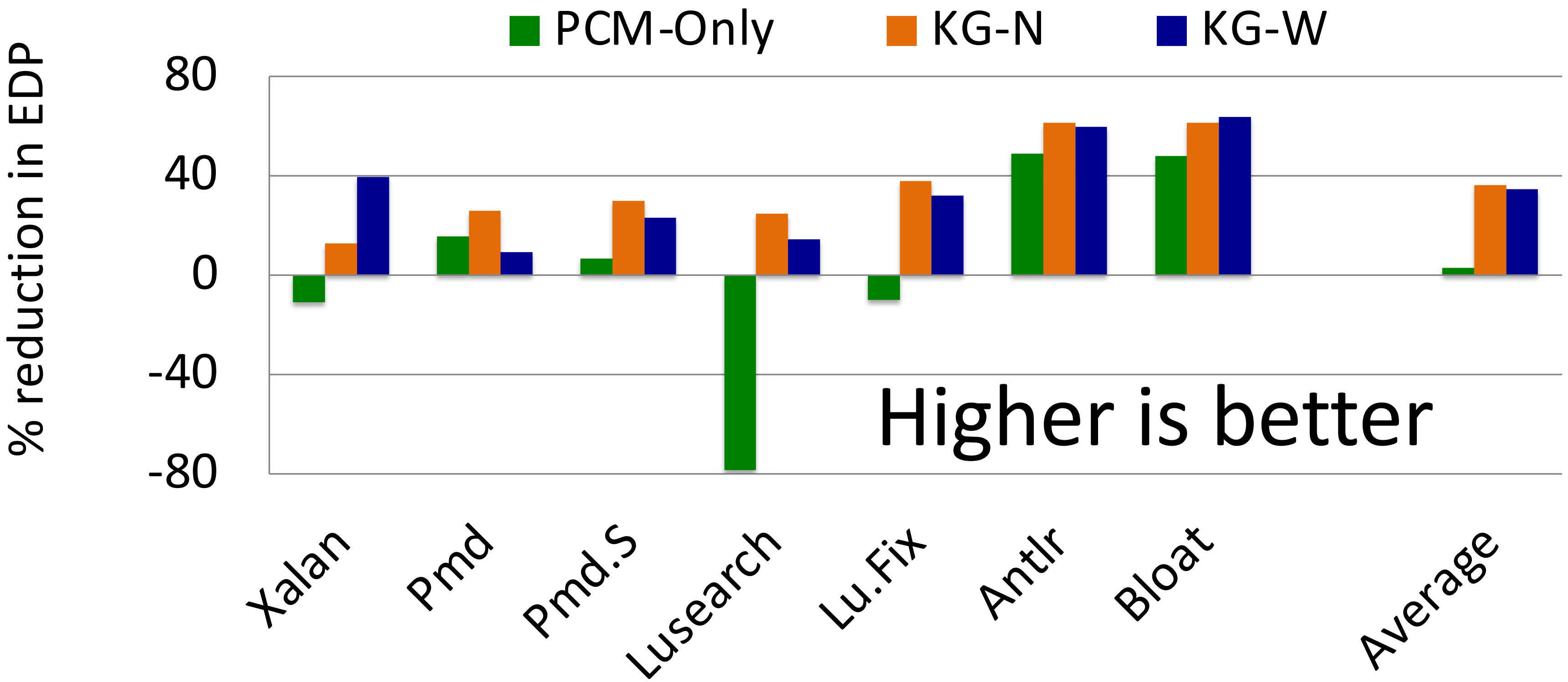
PCM write rates



KG-N reduces write rate by 6X over PCM-Only

KG-W reduces write rate by 2X over KG-N

EDP reduction compared to DRAM



EDP : Energy Delay Product

KG-W has 35% better EDP than DRAM-Only

In the paper

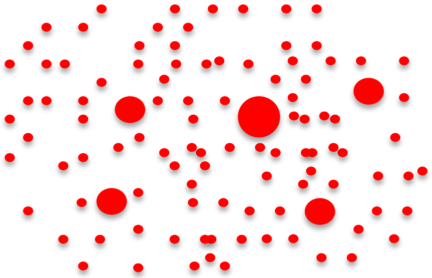
Execution time results

Breakdown of KG-W overheads

Object demographics

Comparison with OS approach

Write rationing garbage collection



Monitor fine grained write behavior of objects

Exploit managed runtimes to organize objects in hybrid memory

Kingsguard collectors improve PCM lifetime

Crystal Gazer: Profile-Driven Write-Rationing Garbage Collection for Hybrid Memories

Shoaib Akram (Ghent), Jennifer B. Sartor (Ghent and VUB), Kathryn S. McKinley (Google), and Lieven Eeckhout (Ghent)

Shoaib.Akram@UGent.be

GHENT UNIVERSITY

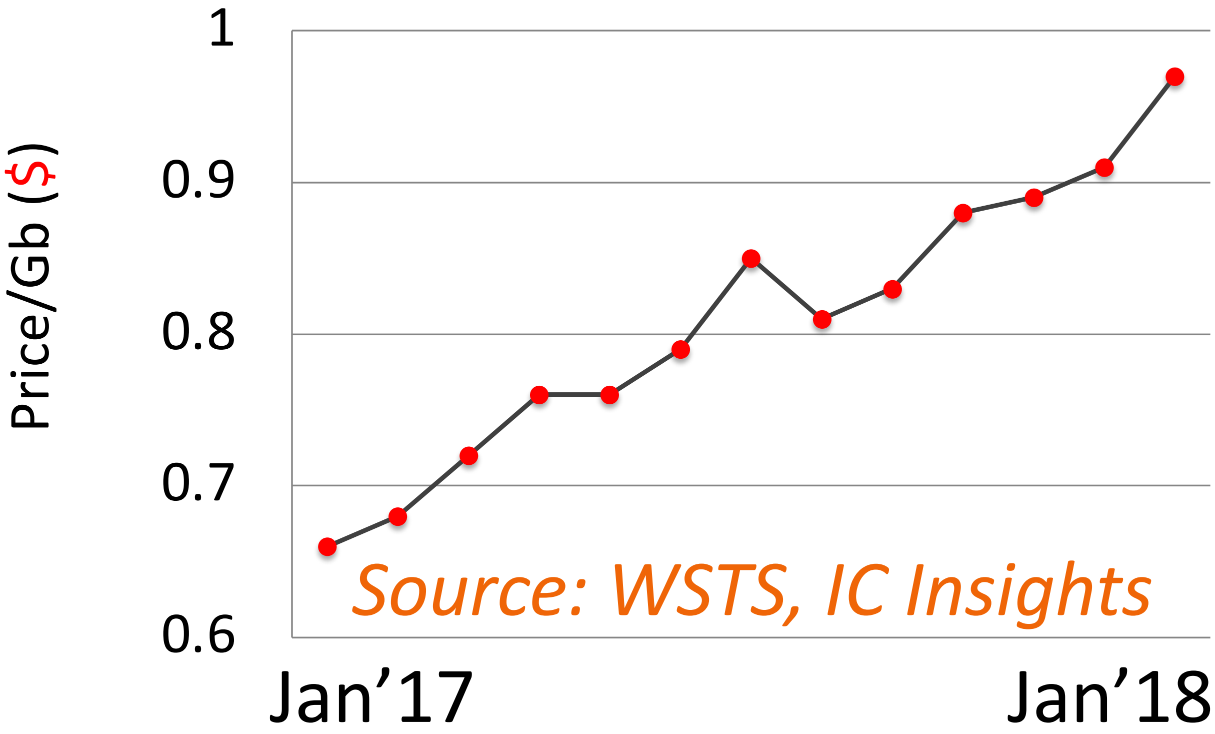
VUB VRIJE UNIVERSITEIT BRUSSEL

Google

Main memory capacity expansion

DRAM → Charge storage a scaling limitation

Manufacturing complexity makes DRAM pricing volatile



Source: WSTS, IC Insights

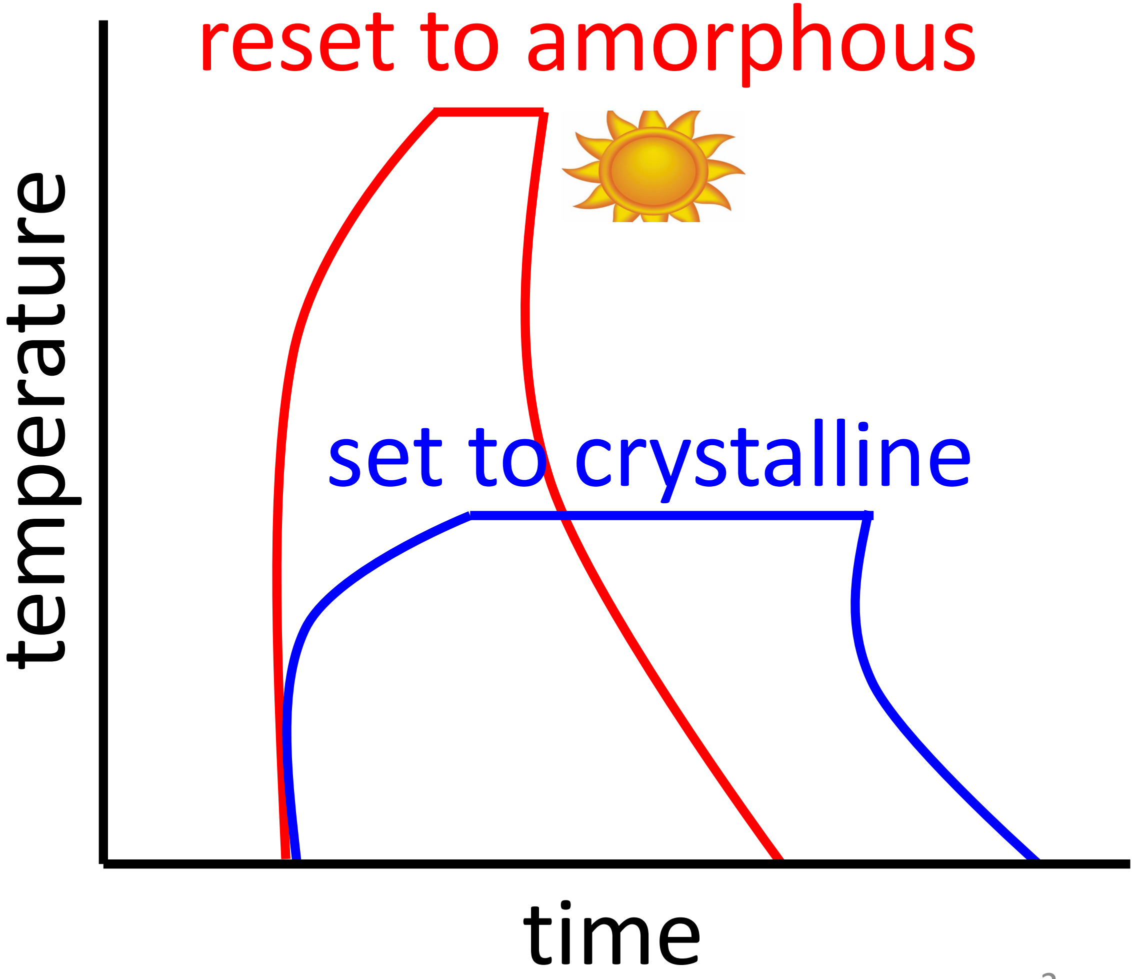
Phase change memory (PCM)

More Gb/$

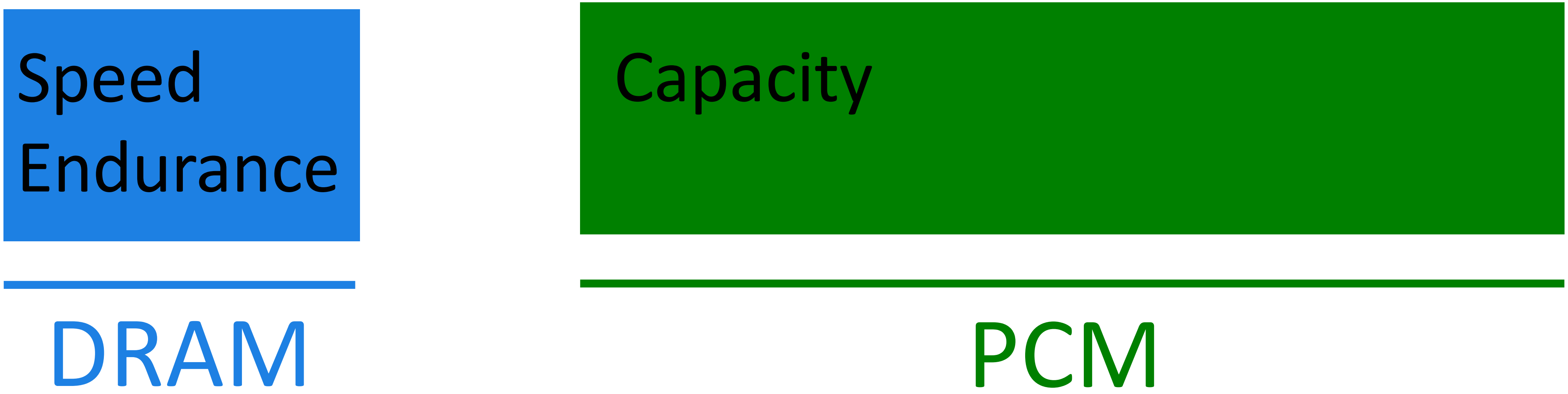
Byte addressable

Latency → DRAM

🙁 Write endurance



Hybrid DRAM-PCM memory



Speed

Endurance

DRAM

Capacity

PCM

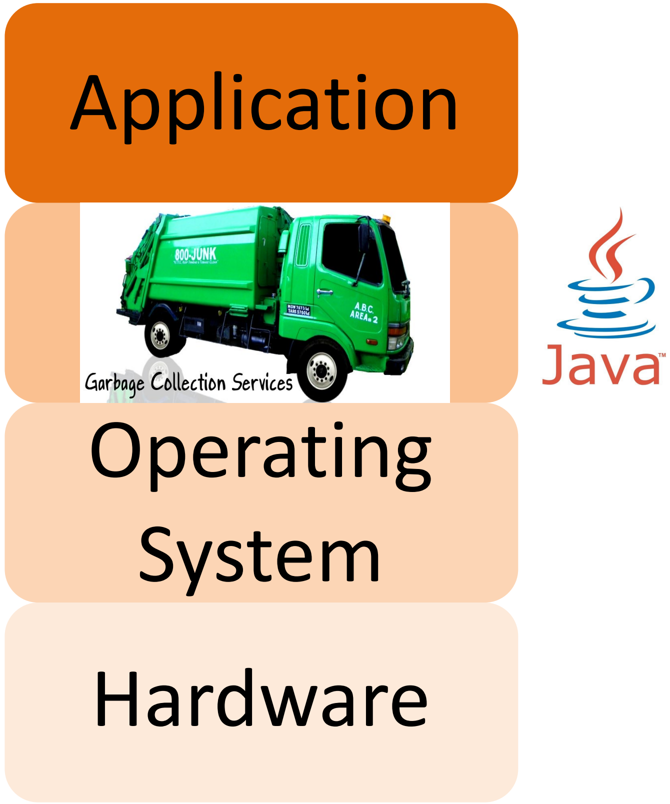
PCM alone can wear out in a few months time

This work → Use DRAM to limit PCM writes

Garbage Collection to limit PCM writes

GC understands memory semantics

A GC approach is *finer grained* than OS approaches



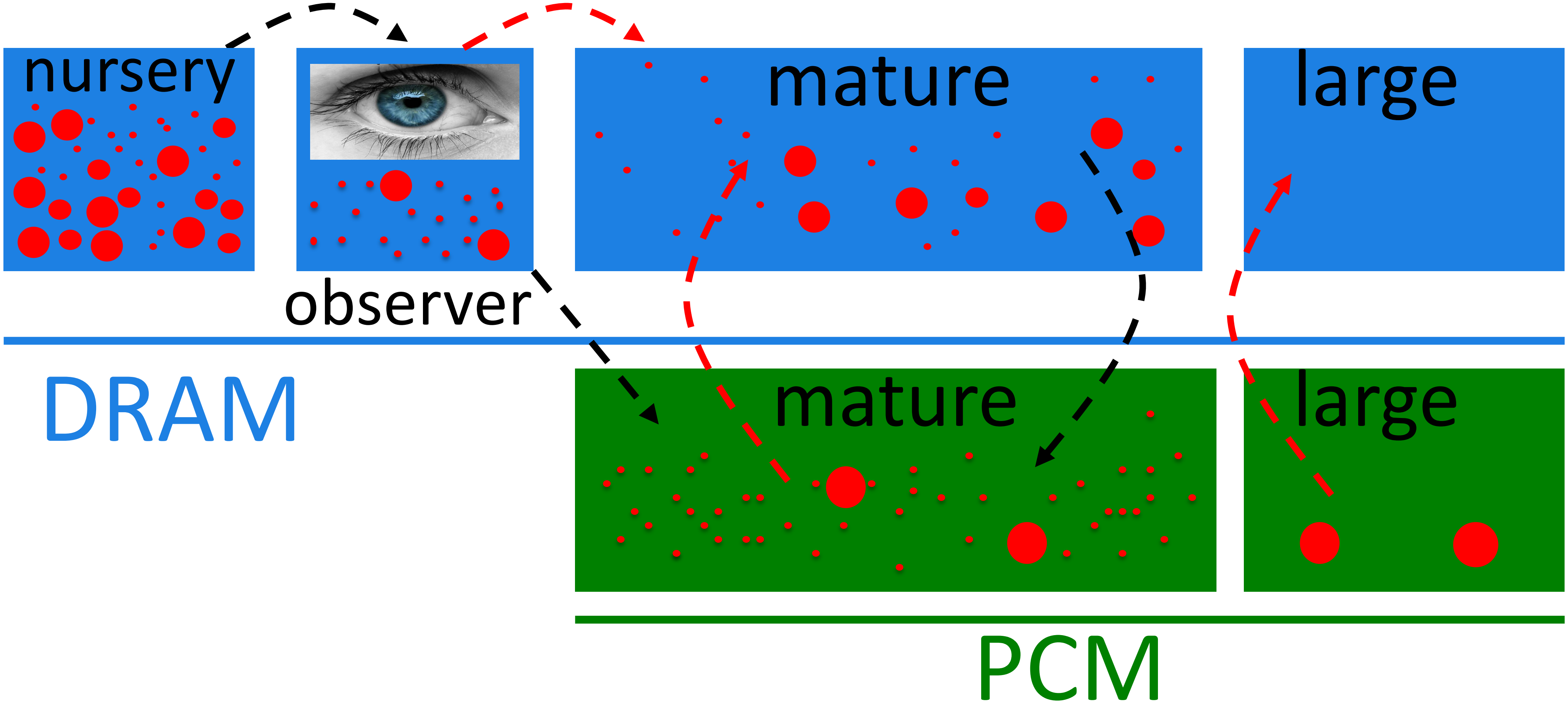
Application

Operating System

Hardware

Write-Rationing Garbage Collection for Hybrid Memories, PLDI, 2018

KG-W Kingsguard-Writers



KG-W drawbacks

Overhead of dynamic monitoring

Limited time window to predict write intensity
→ mispredictions

Excessive & fixed DRAM consumption

Predicting highly written objects without a DRAM observer

Crystal Gazer

Allocation site as a write predictor

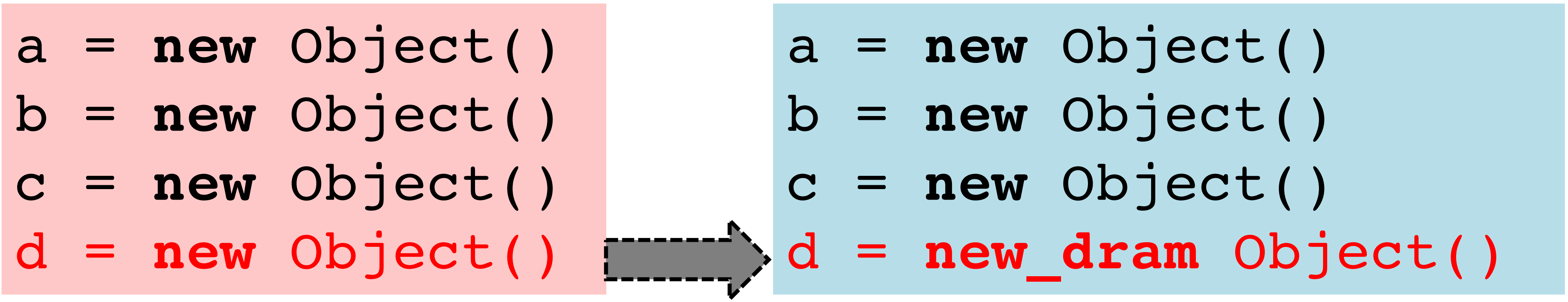
```
a = new Object()
b = new Object()
c = new Object()
d = new Object()
```

Allocation site as a write predictor

```
a = new Object()
b = new Object()
c = new Object()
d = new Object()
```

Uniform distribution 🙁

Allocation site as a write predictor



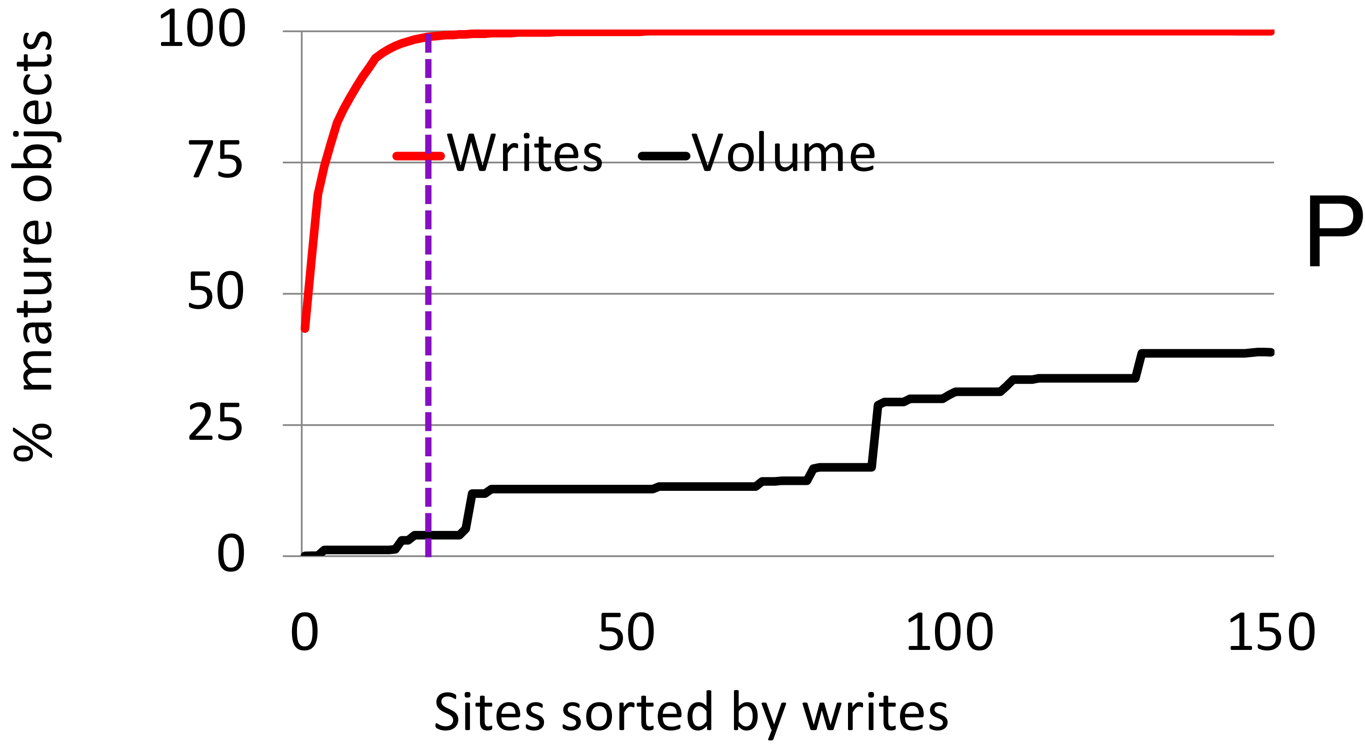
```
a = new Object()
b = new Object()
c = new Object()
d = new Object()
```

```
a = new Object()
b = new Object()
c = new Object()
d = new_dram Object()
```

Uniform distribution 🙁

Skewed distribution 🙂

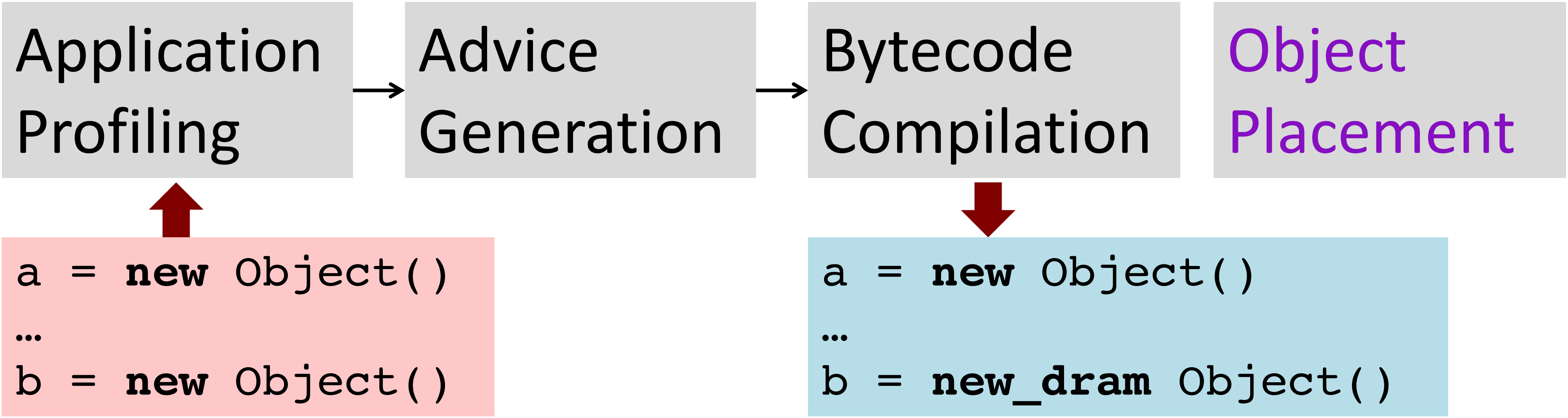
Write distribution by allocation site



Pjbb2005

A few sites capture majority of the writes

Crystal Gazer overview



Application Profiling → Advice Generation → Bytecode Compilation

Object Placement

```
a = new Object()
…
b = new Object()
```

```
a = new Object()
…
b = new_dram Object()
```

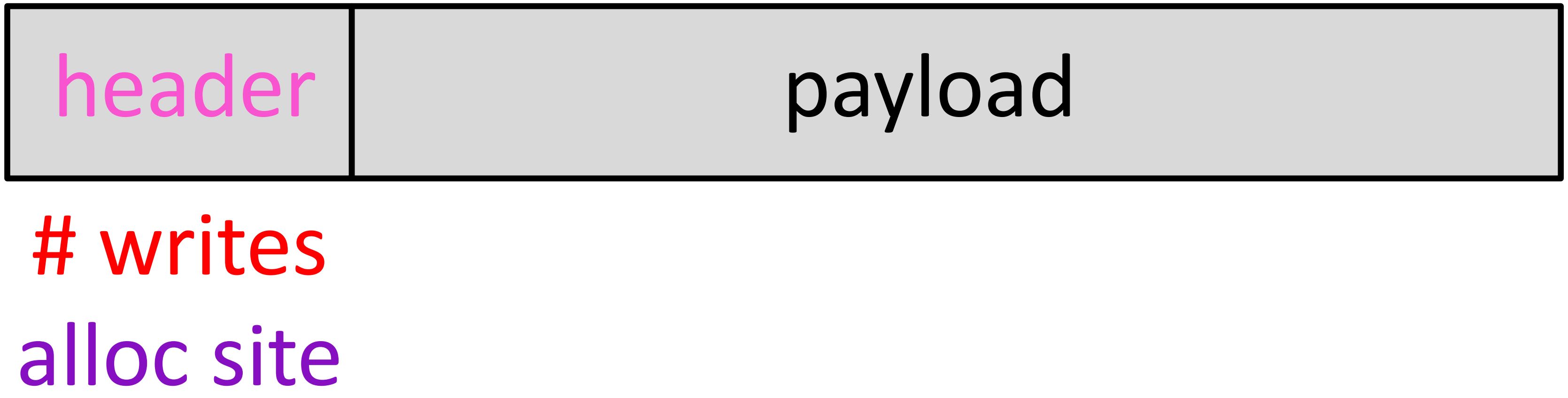
Application profiling (offline)

Goal: Generate a write intensity trace

Object Identifier	# Writes	# Bytes	Allocation site
O1	0	4	A() + 10
O2	0	4	A() + 10
O3	2048	4	A() + 10
O4	2048	4096	B() + 4

Tracking alloc sites and # writes

Object layout



header

payload

# writes

alloc site

Compiler inserts code to compute allocation sites

Write barrier tracks # writes to each object

Application Profiling

Minimize full-heap collections → 3 GB heap

Nursery size a balance b/w size of trace and mature object coverage

2.4X slowdown across 15+ applications

Advice generation

Goal: Generate <alloc-site, advice> pairs

advice → DRAM or PCM

input is a write-intensity trace

Two heuristics to classify allocation sites as DRAM or PCM

Alloc site classification heuristics

Freq: A *threshold* % of objects from a site get more than a *threshold* # writes → DRAM

🙂 Aggressively limits PCM writes

🙁 No distinction based on object size

Alloc site classification heuristics

Write density → Ratio of # writes to object size

Dens: A *threshold* % of objects from a site have more than a *threshold* write density → DRAM

Classification thresholds

Homogeneity threshold → 1%

Frequency threshold → 1

Density threshold → 1

Classification examples

Frequency threshold = 1

PCM writes = ?, DRAM bytes = ?

Object Identifier	# Writes	# Bytes	Allocation site
O1	0	4	A() + 10
O2	0	4	A() + 10
O3	128	4	A() + 10
O4	128	4096	B() + 4

Classification examples

Frequency threshold = 1

PCM writes = ?, DRAM bytes = ?

	Object Identifier	# Writes	# Bytes	Allocation site
	O1	0	4	A() + 10
	O2	0	4	A() + 10
→	O3	128	4	A() + 10
→	O4	128	4096	B() + 4

Classification examples

Frequency threshold = 1

PCM writes = 0/256, DRAM bytes = 5008

	Object Identifier	# Writes	# Bytes	Allocation site
	O1	0	4	A() + 10
	O2	0	4	A() + 10
→	O3	128	4	A() + 10
→	O4	128	4096	B() + 4

Classification examples

Density threshold = 1

PCM writes = ?, DRAM bytes = ?

Object Identifier	# Writes	# Bytes	Allocation site
O1	0	4	A() + 10
O2	0	4	A() + 10
O3	128	4	A() + 10
O4	128	4096	B() + 4

Classification examples

Density threshold = 1

PCM writes = ?, DRAM bytes = ?

Object Identifier	# Writes	# Bytes	Allocation site	
O1	0	4	A() + 10	
O2	0	4	A() + 10	
O3	128	4	A() + 10	→ 32
O4	128	4096	B() + 4	

Classification examples

Density threshold = 1

PCM writes = ?, DRAM bytes = ?

Object Identifier	# Writes	# Bytes	Allocation site	
O1	0	4	A() + 10	
O2	0	4	A() + 10	
O3	128	4	A() + 10	→ 32
O4	128	4096	B() + 4	→ < 1

Classification examples

Density threshold = 1

PCM writes = 128/256, DRAM bytes = 12

Object Identifier	# Writes	# Bytes	Allocation site	
O1	0	4	A() + 10	
O2	0	4	A() + 10	
O3	128	4	A() + 10	→ 32
O4	128	4096	B() + 4	→ < 1

Bytecode compilation

Introduce a new bytecode → *new_dram()*

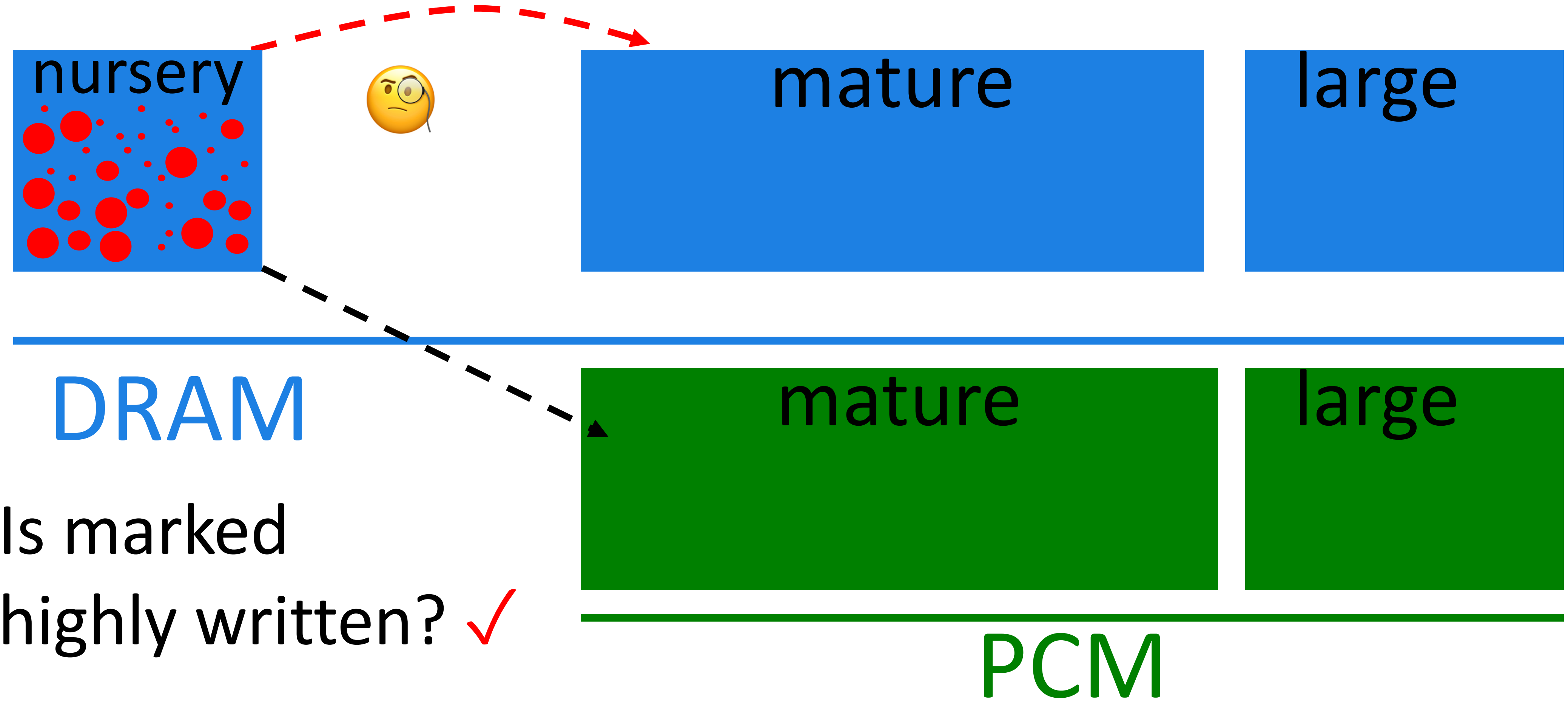
Bytecode rewriter modifies DRAM sites to use *new_dram()*

Object placement

new_dram() → Set a bit in the object header

GC → Inspect the bit on nursery collection to copy object in DRAM or PCM

Object placement



Is marked highly written? ✓

Key features of Crystal Gazer

Eliminate overheads of dynamic monitoring

Proactive → less mispredictions

Reduces DRAM usage & opens up pareto-optimal tradeoffs b/w capacity and lifetime

Evaluation methodology

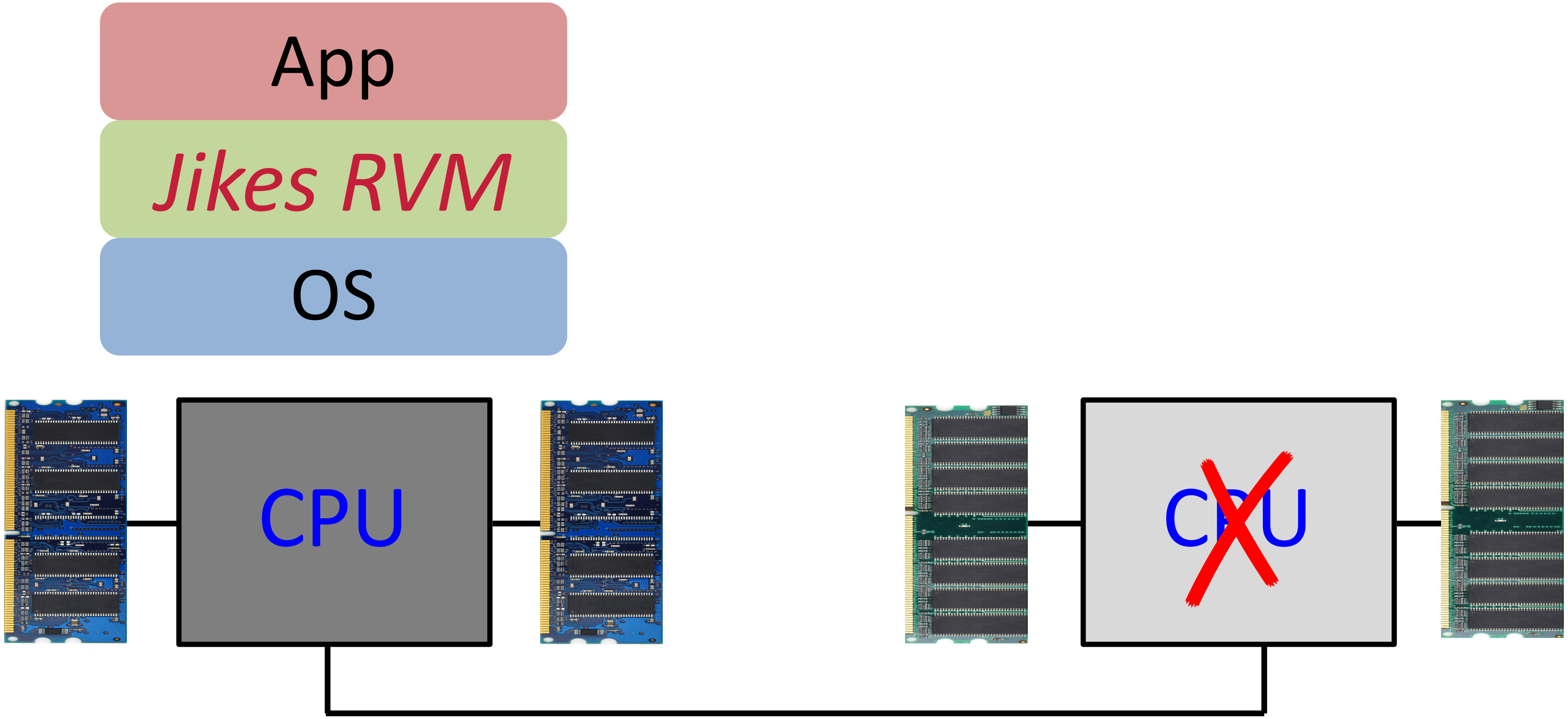
15 Applications → DaCapo, GraphChi, SpecJBB

Medium-end server platform

Different inputs for production and advice

Jikes RVM

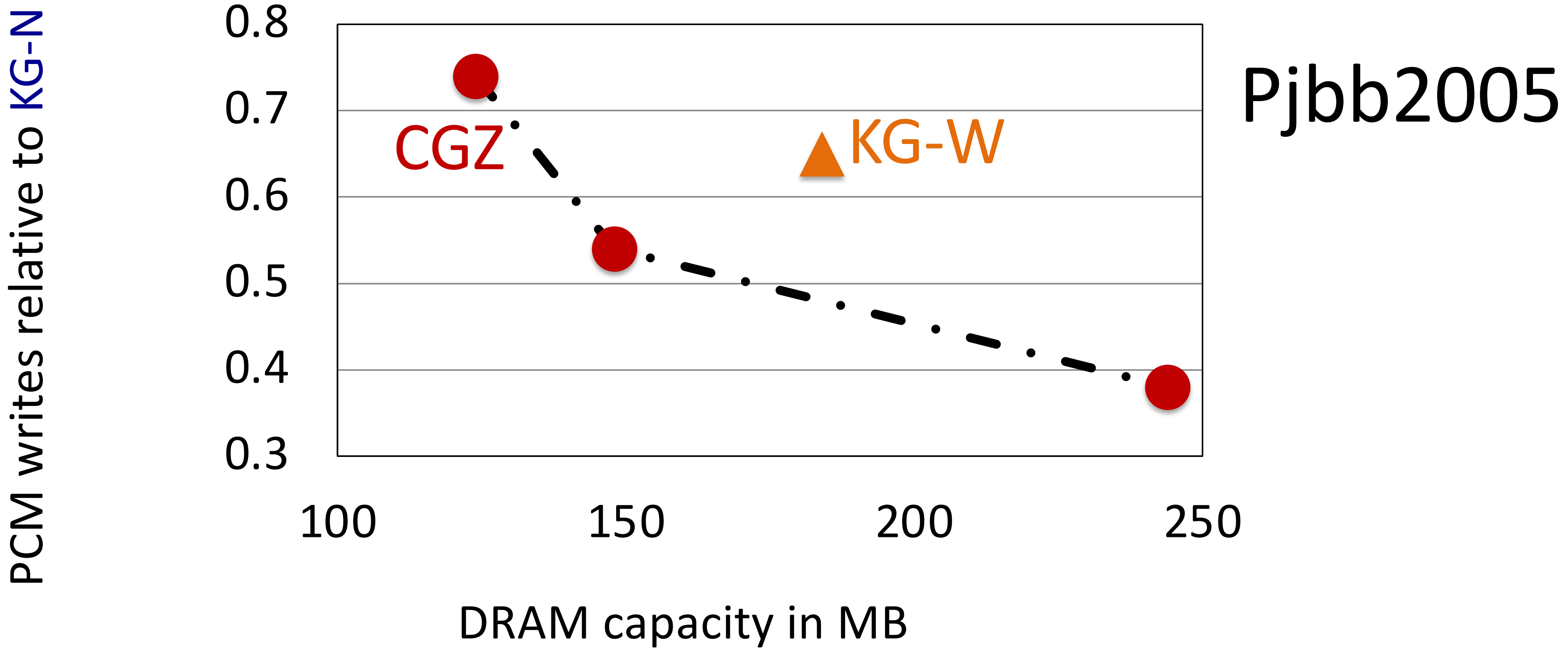
Emulation on NUMA hardware



16 hardware threads and 20 MB L3

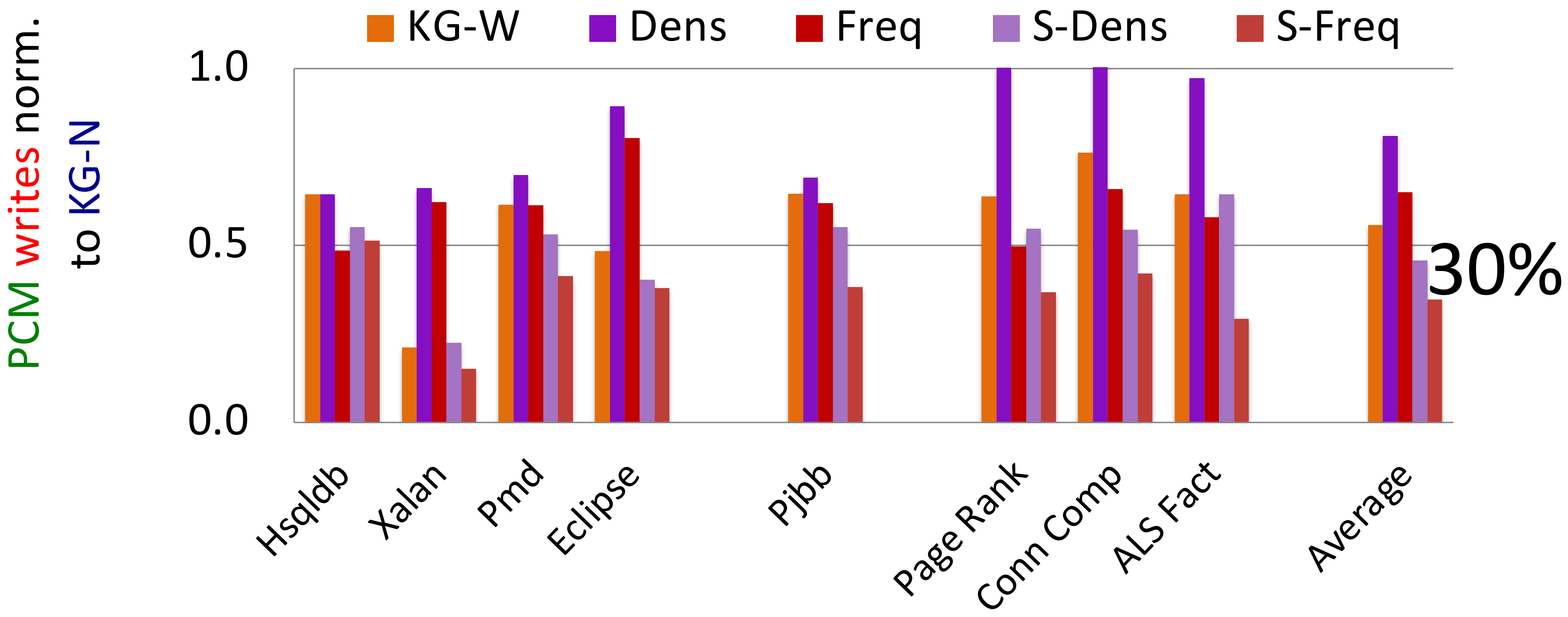
Use Intel pcm-memory.x to get per-socket write rate

Lifetime versus DRAM capacity



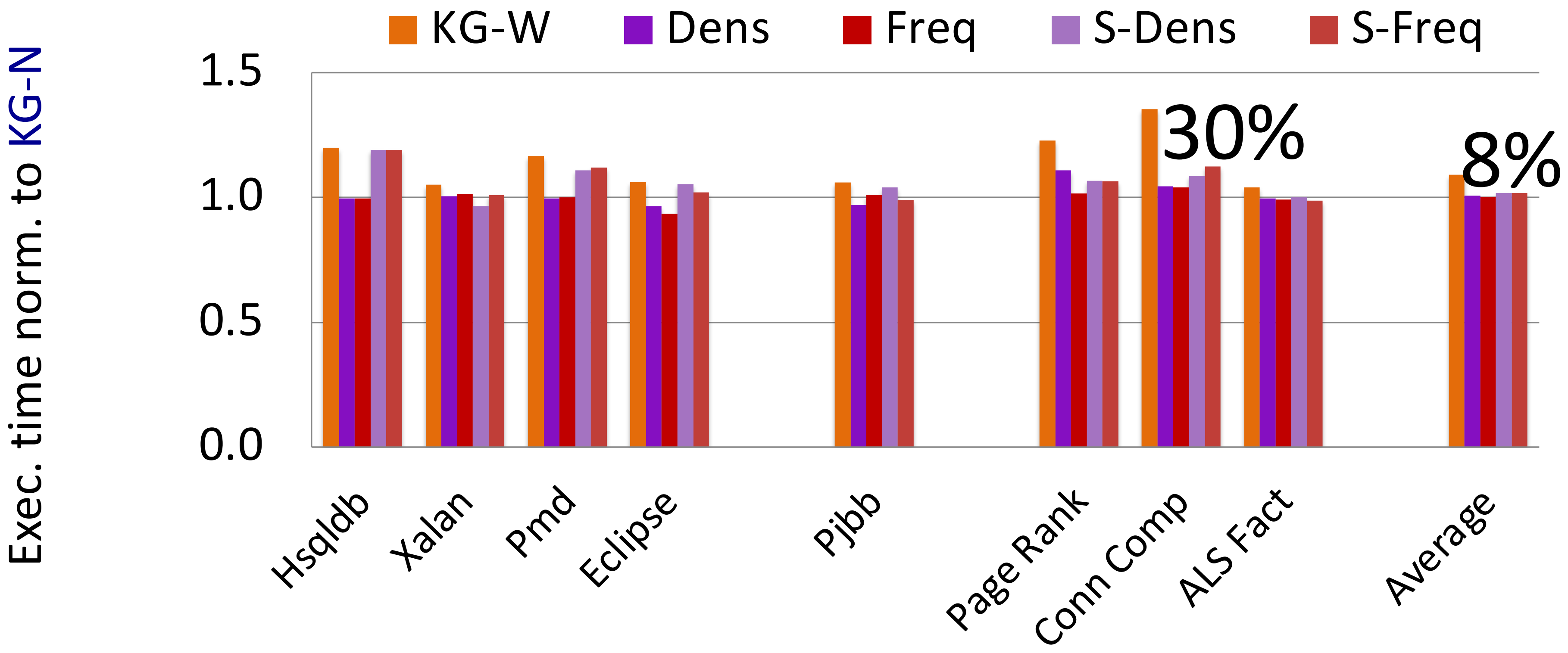
Crystal Gazer provides Pareto-optimal choices

PCM Writes



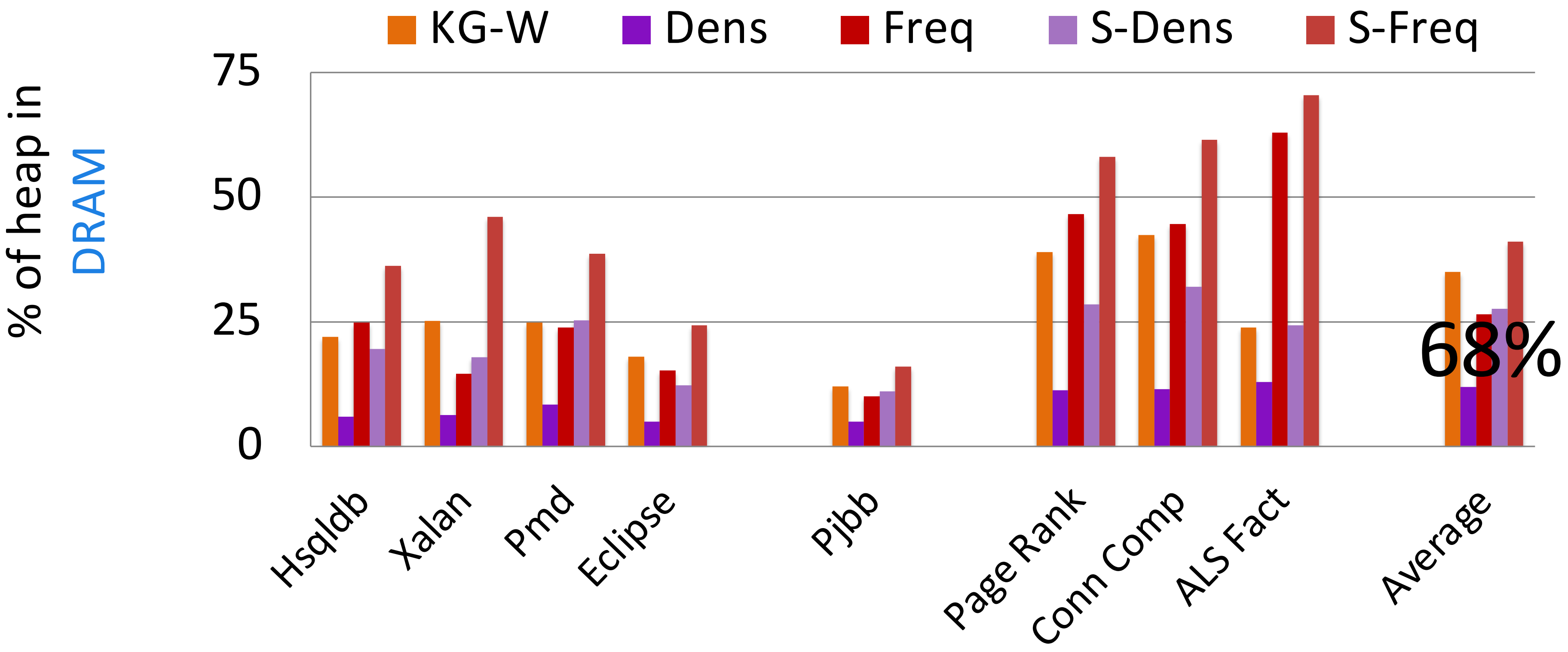
To optimize for lifetime, use Freq & survivors

Execution time



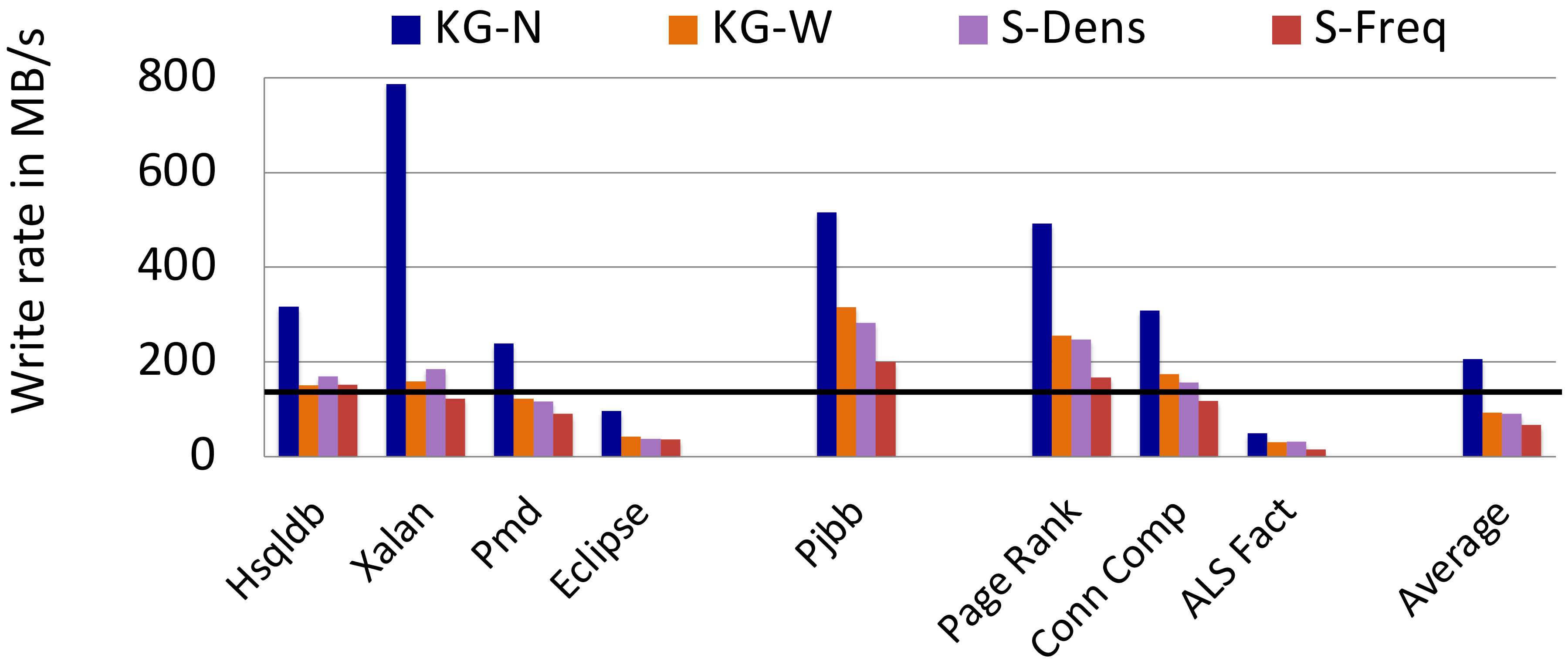
To optimize for performance, use Freq or Dens

DRAM capacity



To optimize for DRAM usage, use Dens

Write rates



Write-rationing GC makes PCM practical

Profile-driven write-rationing GC

Hybrid memory is inevitable

DRAM PCM

Allocation site a good predictor of writes

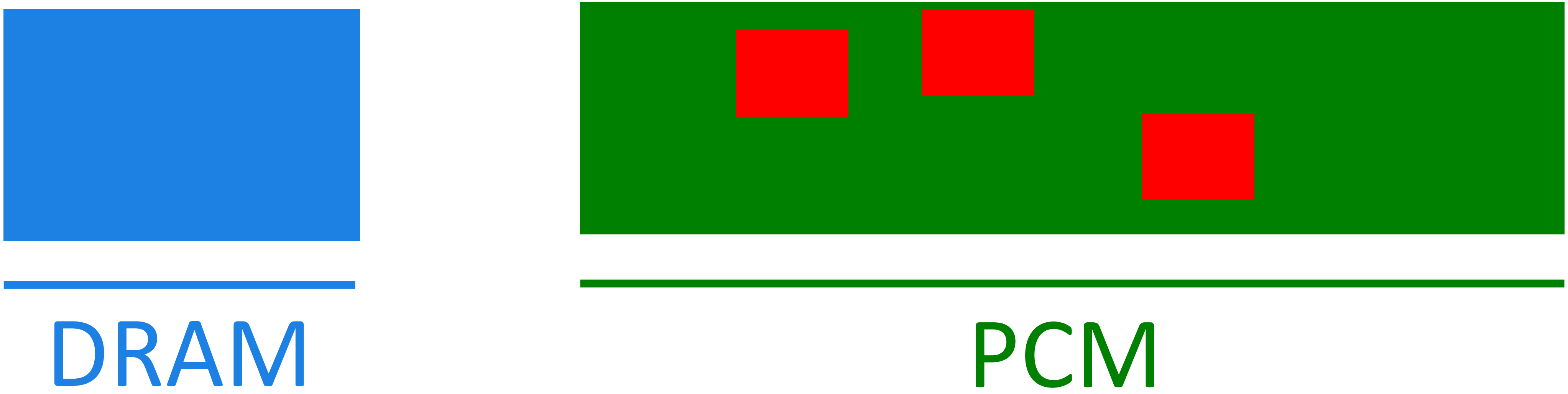
Static approach beats dynamic

→ Better performance

→ Reduced DRAM capacity

→ Better PCM lifetime

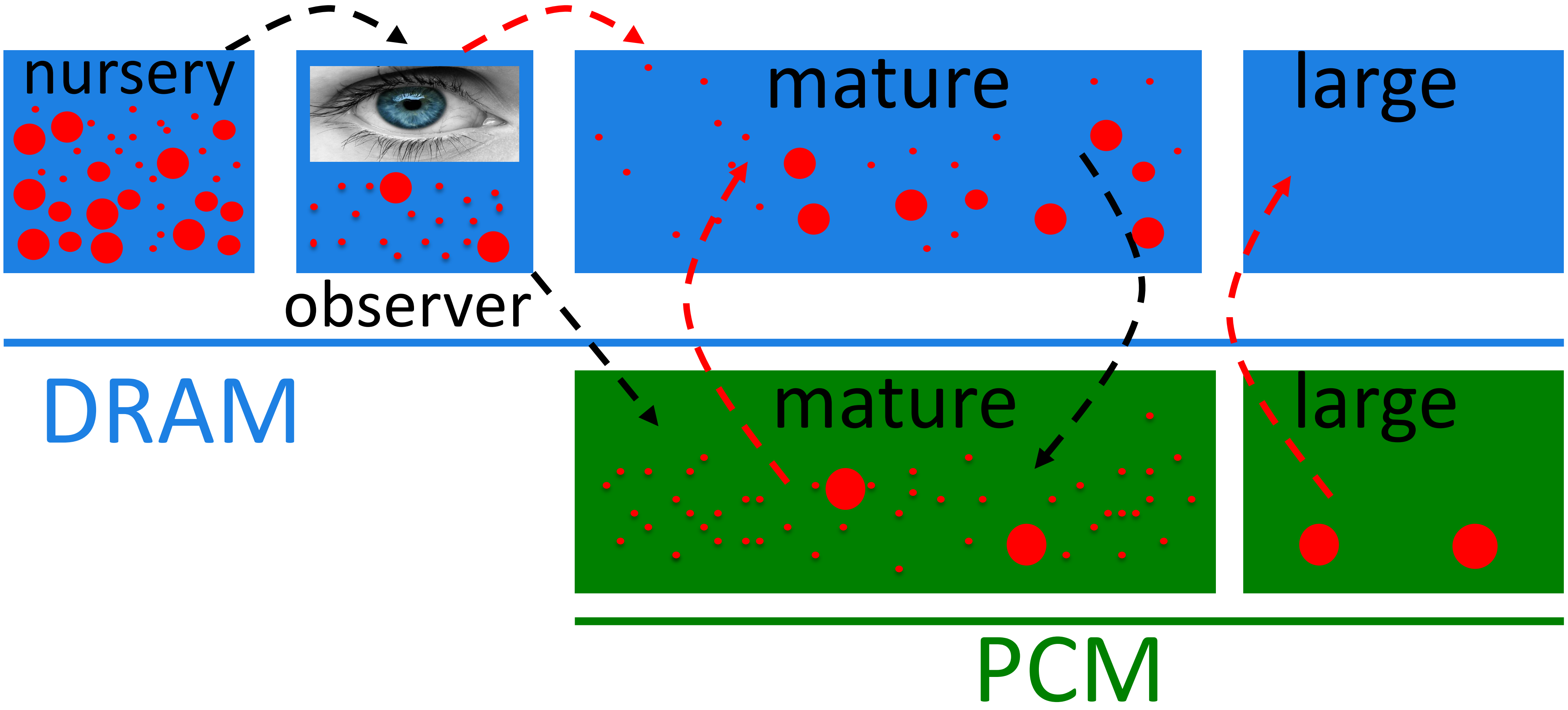
OS to limit PCM writes



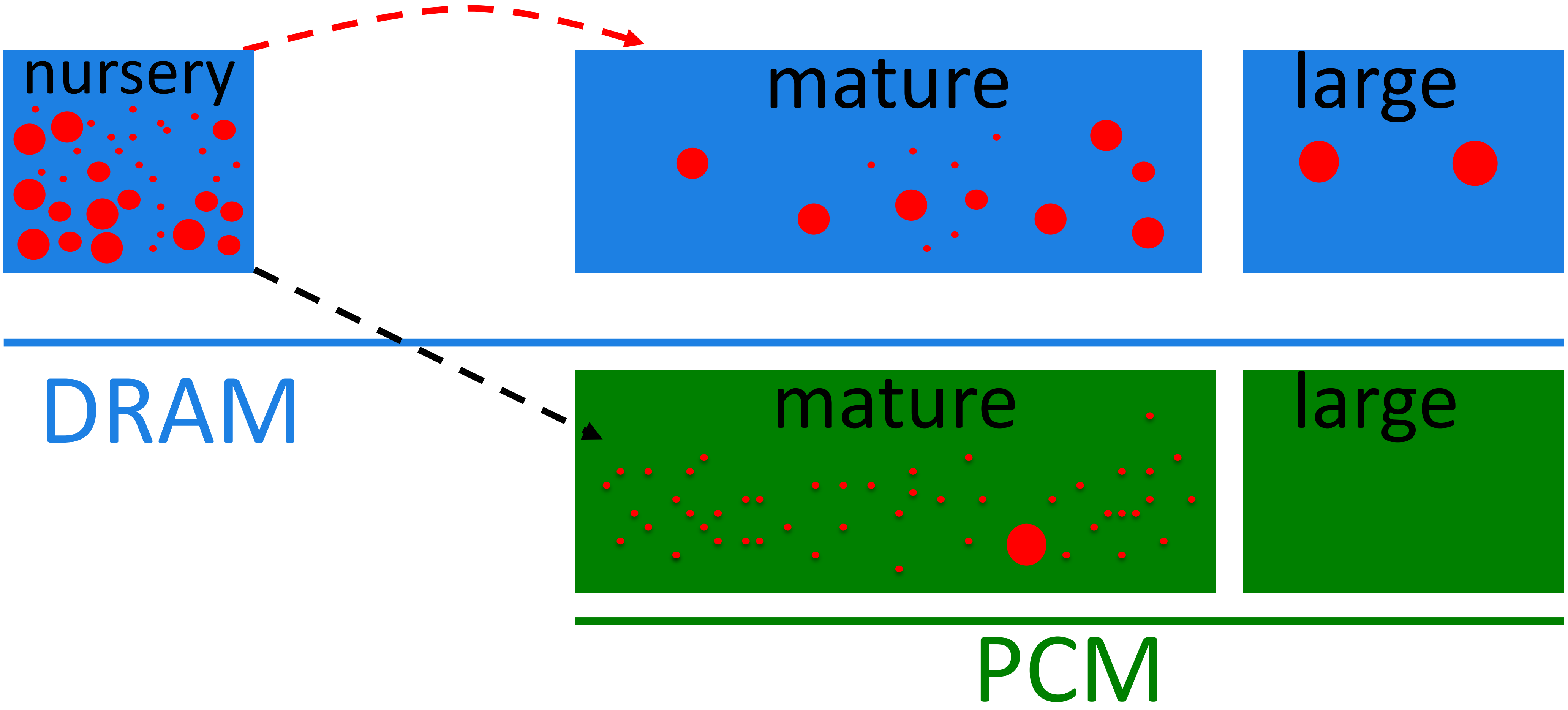
Coarse-grained data movement is inefficient

Page migrations hurt performance and lifetime

Object placement

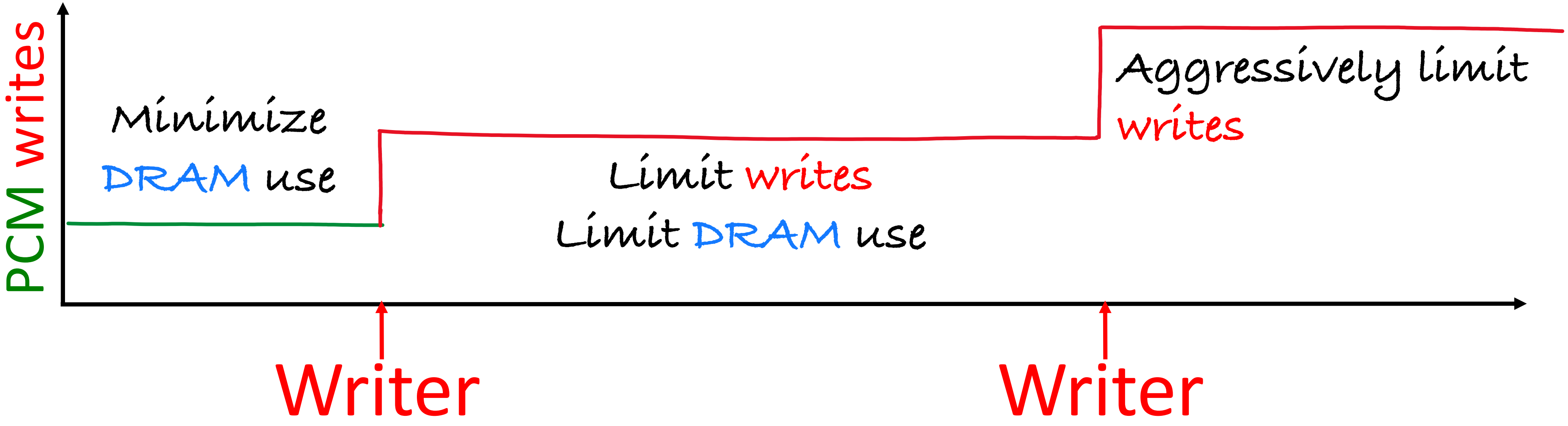


Object placement



Proteus: Workload-adaptive write-rationing GC

Problem: continuous workload

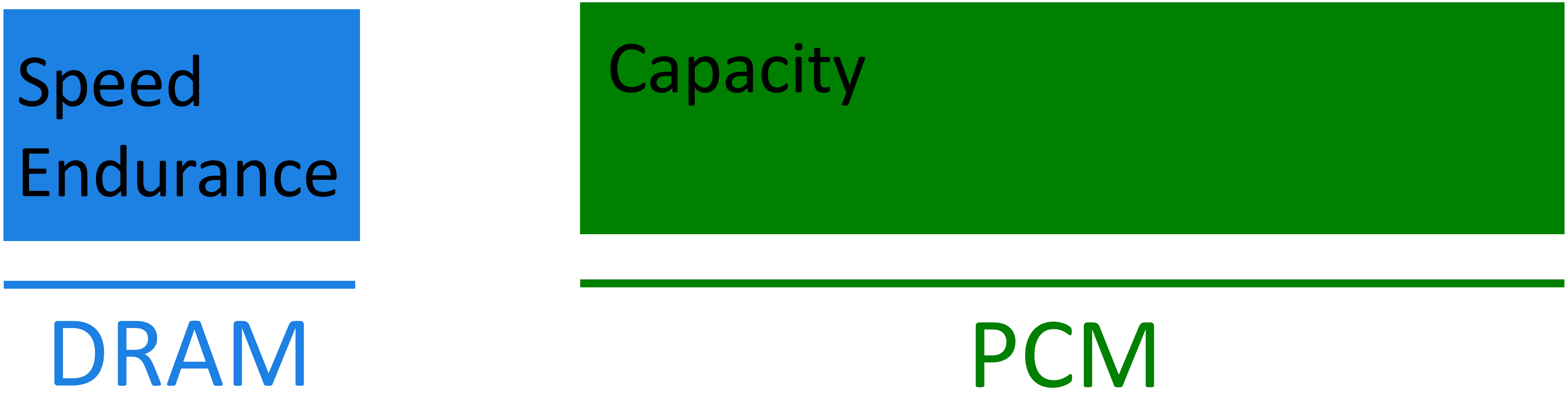


Proteus: encode advice for different scenarios in object headers

Hybrid DRAM-PCM memory

🙂 More GB/$ with Phase Change Memory

🙁 Higher latency *and* low endurance



Speed
Endurance

DRAM

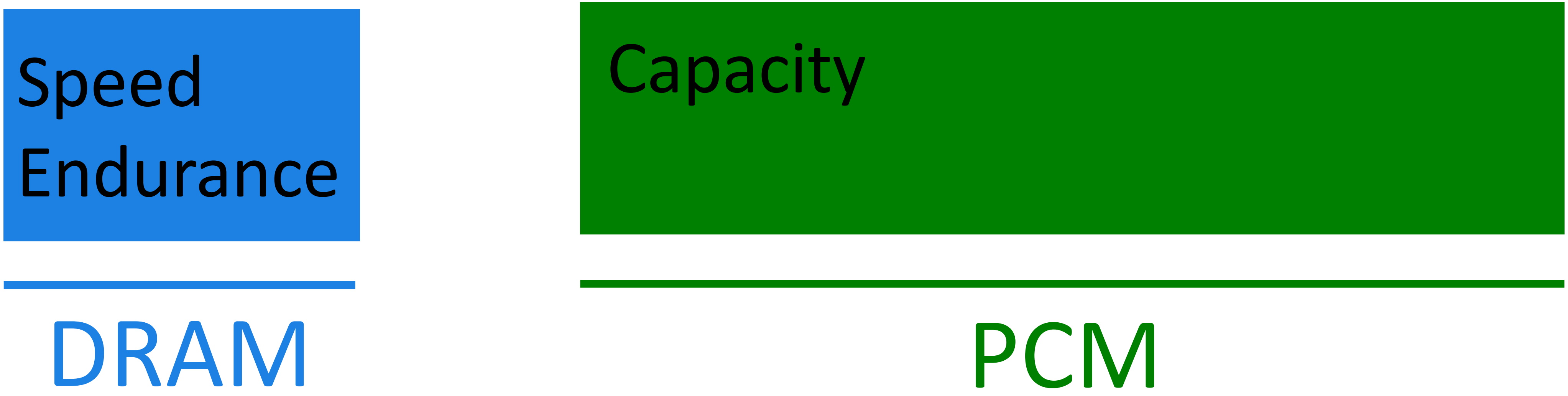
Capacity

PCM

Managing DRAM-PCM memory

Mitigate PCM wear-out

Bridge the DRAM-PCM latency gap



Speed

Endurance

Capacity

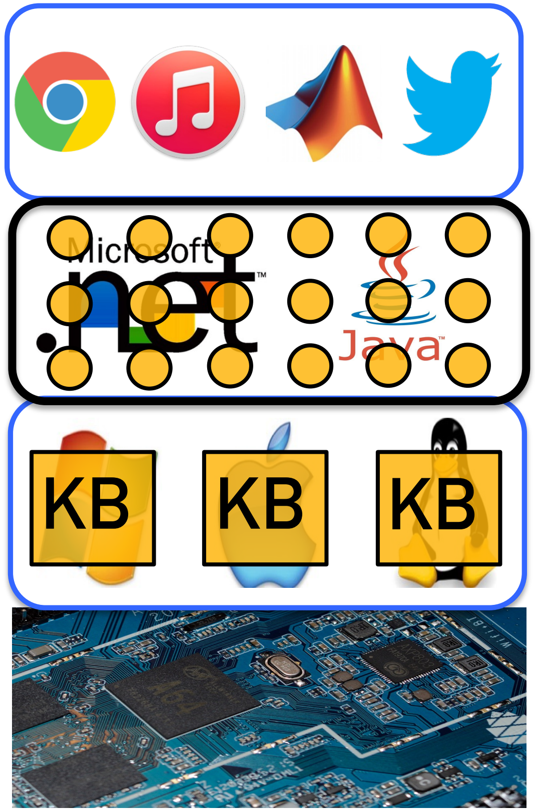
DRAM

PCM

Managing DRAM-PCM memory

PLDI Philadelphia 2018

Write-Rationing Garbage Collection for Hybrid Memory



Garbage collection

Proactive ☺

Fine-grained objects ● ● ● ●

Operating System

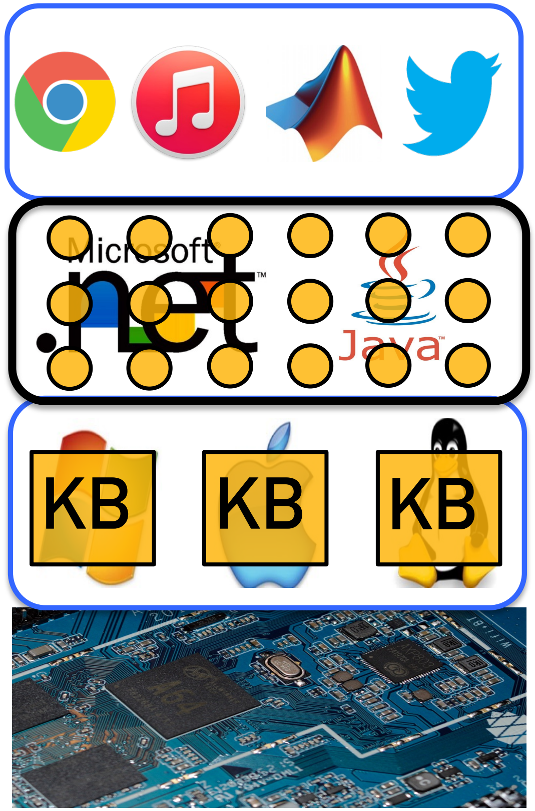
Coarse-grained pages KB

GC manages DRAM-PCM hybrid better than OS

Managing DRAM-PCM memory

PLDI Philadelphia 2018

Write-Rationing Garbage Collection for Hybrid Memory



Operating System

Coarse-grained pages KB

Garbage collection

Proactive ☺

Fine-grained objects ● ● ● ●

GC manages DRAM-PCM hybrid better than OS

Pros/cons of simulating DRAM-PCM

Gain insight

- What triggered the writeback to memory?

Study parameter sensitivity

Slow process

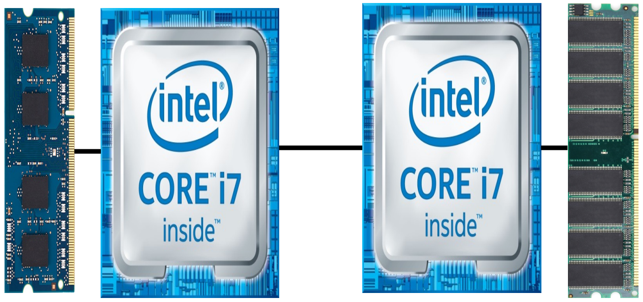
- Page Rank over twitter → hours versus months!

Incomplete model

- Missing OS or proprietary hardware features

Emulation for hybrid memory

Multi-socket NUMA for emulating DRAM-PCM hybrid memory



Fast evaluation of emerging workloads

- Several co-running BIG graph analytic applications written in Java

Existing emulation platforms

Focus is to evaluate explicit memory management in C/C++

Focus is to model the latency of PCM

Contribution: Emulation platform

DRAM-PCM emulation for managed applications

Comparison with Sniper using write-rationing garbage collectors

Contribution: Analysis of PCM writes

PCM writes and write rates

C++ versus Java

Impact of multiprogramming

Classic versus emerging applications

Is PCM practical as main memory?

Outline

Heap management

Kingsguard collectors

Comparison with simulation

Write analysis

Outline

Heap management

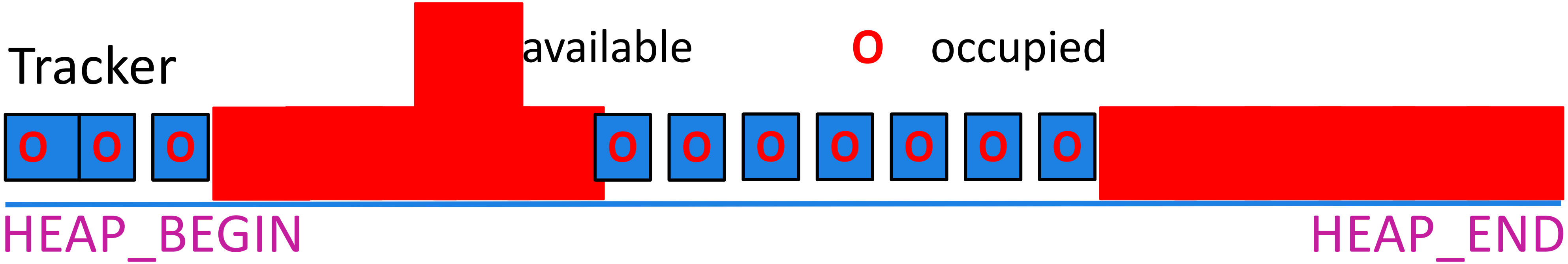
Kingsguard collectors

Comparison with simulation

Write analysis

DRAM heap management

Heap Tracker



available

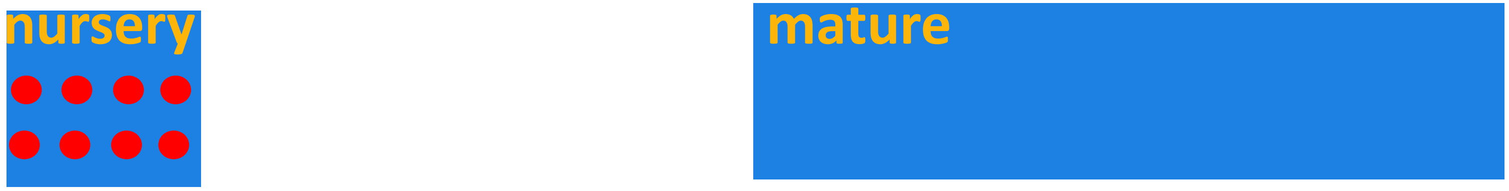
o occupied

o o o o o o o o o o

HEAP_BEGIN

HEAP_END

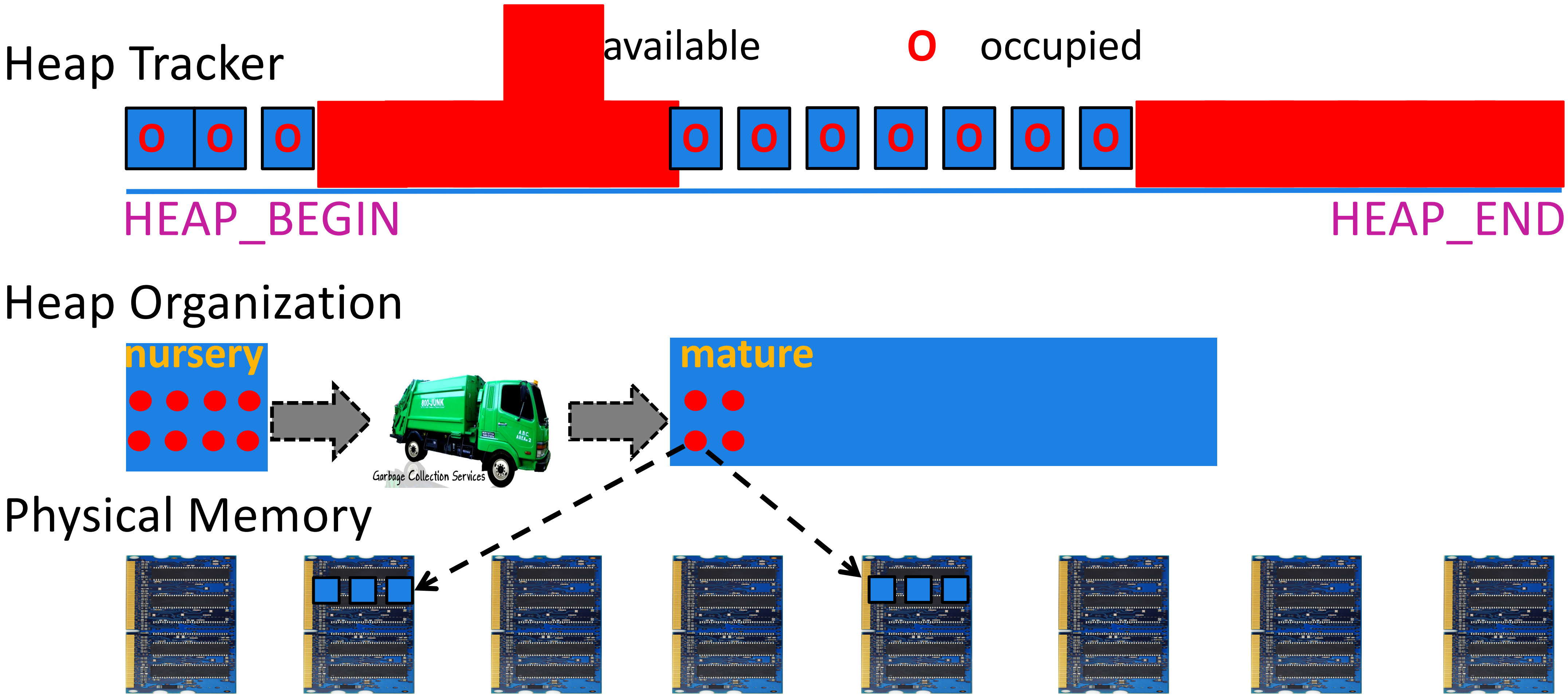
Heap Organization



nursery

mature

DRAM heap management



Heap Tracker

available

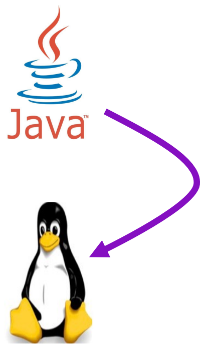
o occupied

Heap Organization

Physical Memory

DRAM-PCM heap management

JVM uses mbind() to inform the OS to map a space in DRAM or PCM



Anything else the JVM should do?

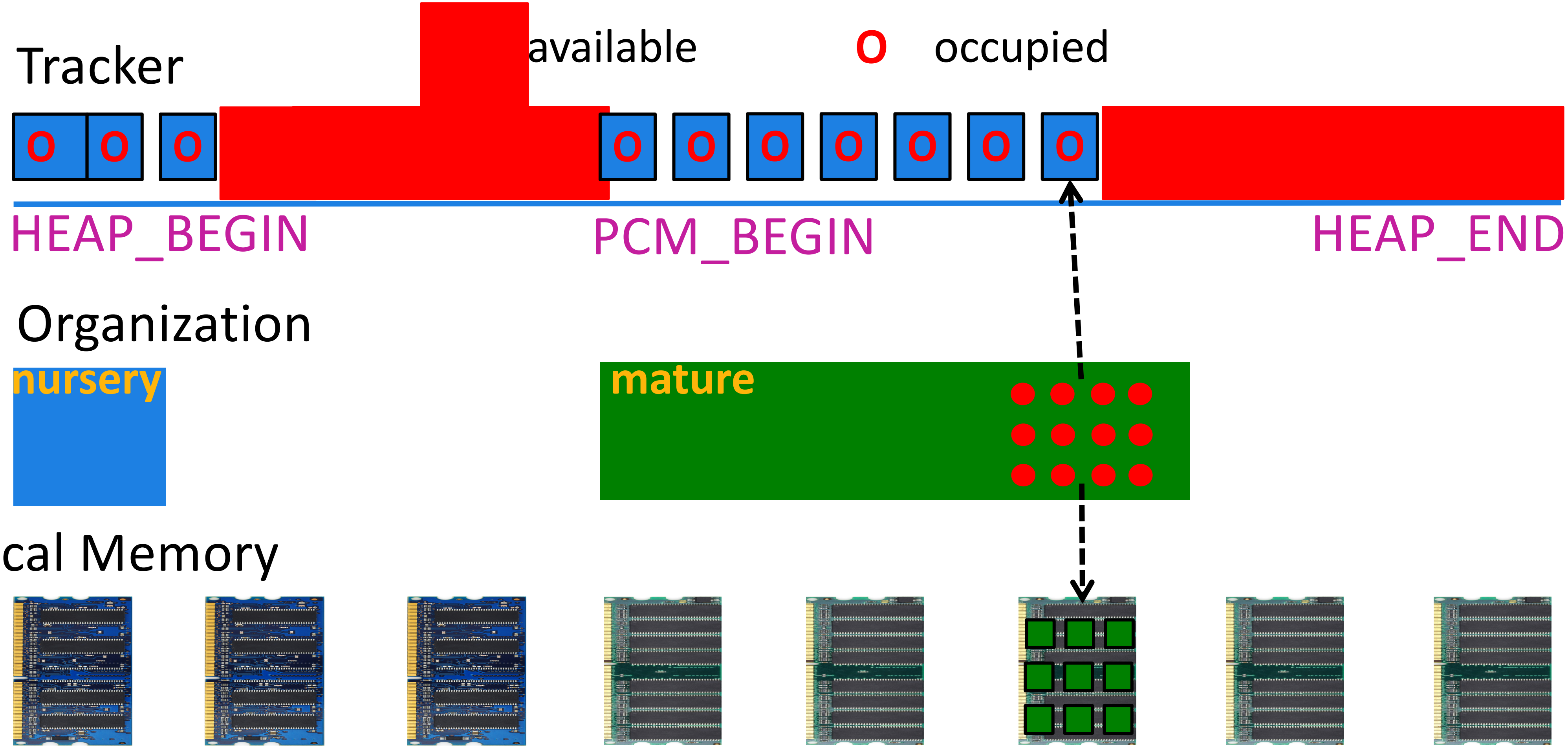
Next: Sanity check with a DRAM nursery and PCM mature

DRAM-PCM heap management

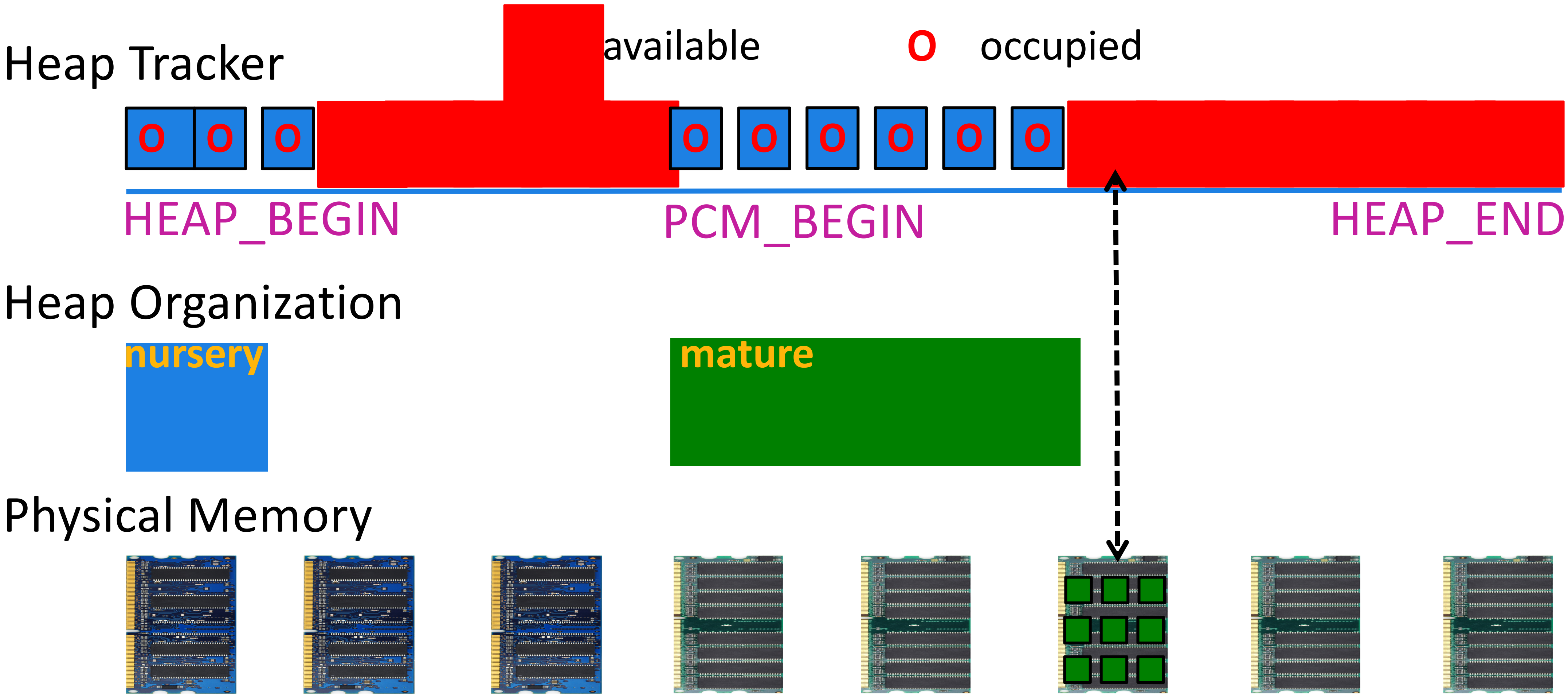
Heap Tracker

Heap Organization

Physical Memory



DRAM-PCM heap management



Heap Tracker

available

O occupied

HEAP_BEGIN

PCM_BEGIN

HEAP_END

Heap Organization

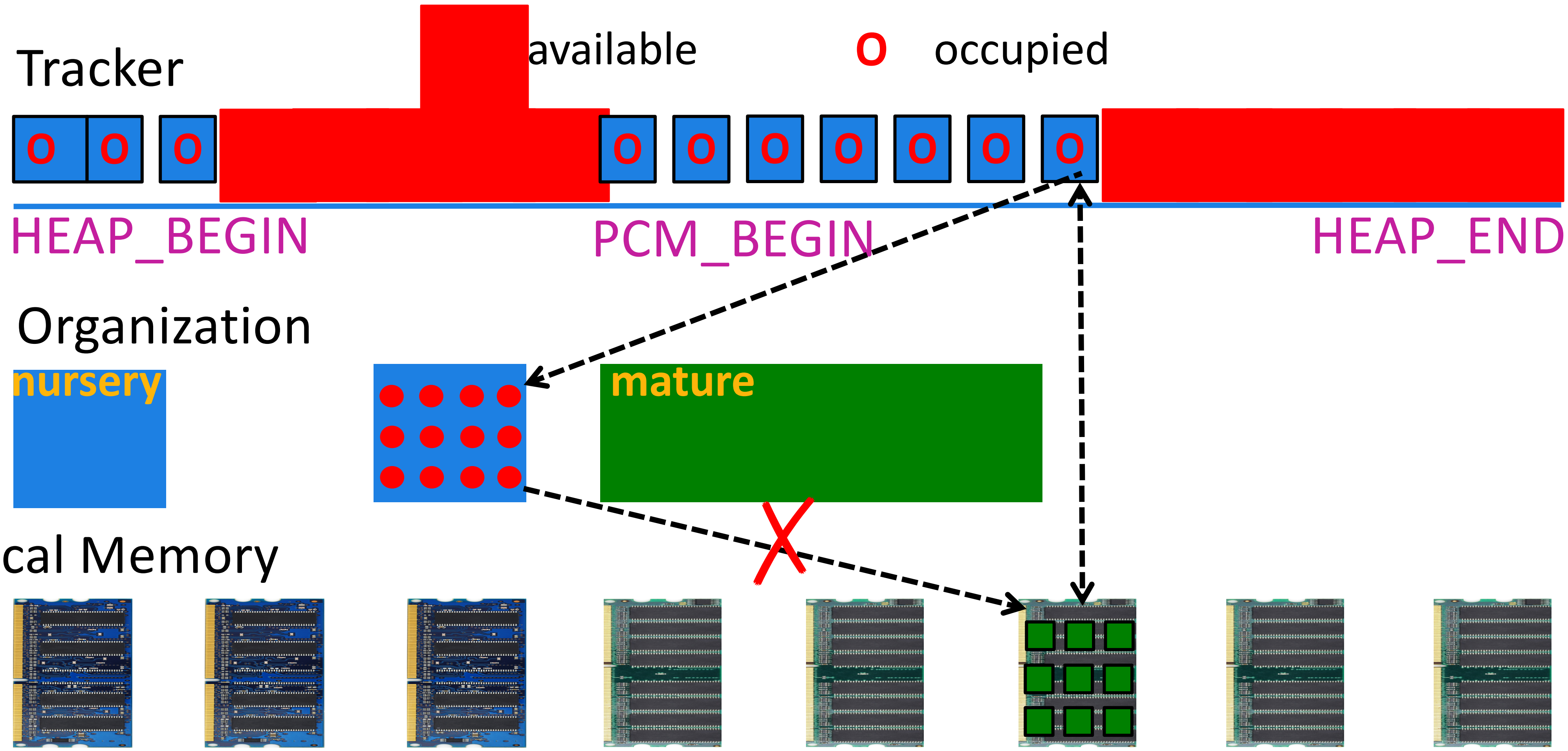
nursery

mature

Physical Memory

DRAM-PCM heap management

Heap Tracker



available

o occupied

o o o o o o o o o o

HEAP_BEGIN

PCM_BEGIN

HEAP_END

Heap Organization

nursery

mature

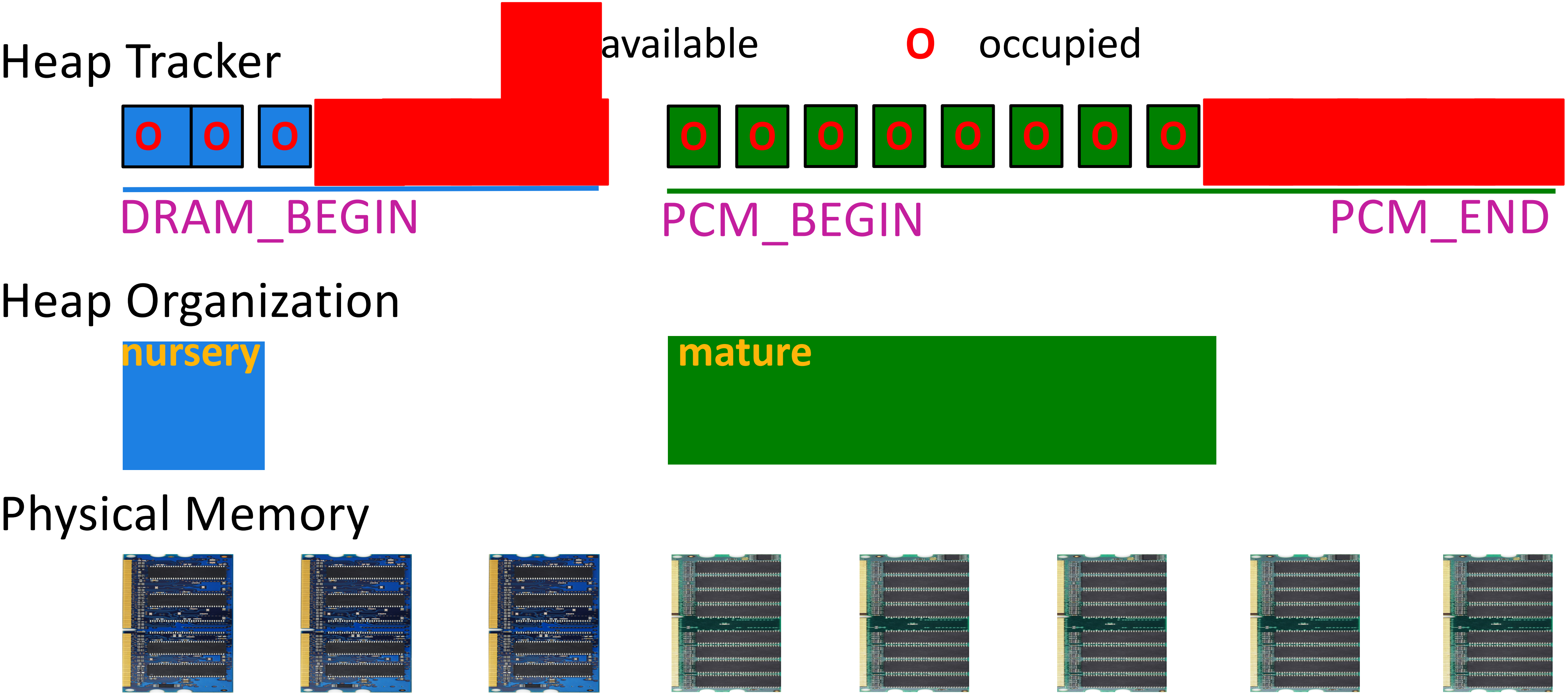
Physical Memory

DRAM-PCM heap management

Options

- Map/unmap pages in physical memory whenever space grows/shrinks
- Two free lists ✓

DRAM-PCM heap management



Heap Tracker

available o occupied

Heap Organization

Physical Memory

Outline

Heap management

Kingsguard collectors

Comparison with simulation

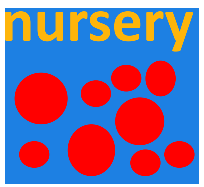
Write analysis

Kingsguard-Nursery (KG-N)

Write-rationing GC: concentrate writes in DRAM

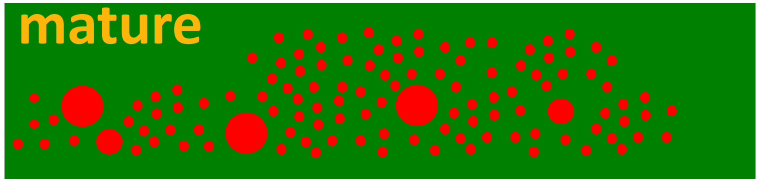
70%

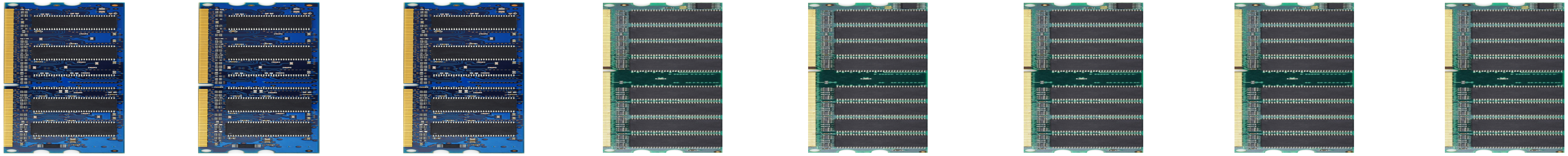
of writes



22%

to 2% of objects

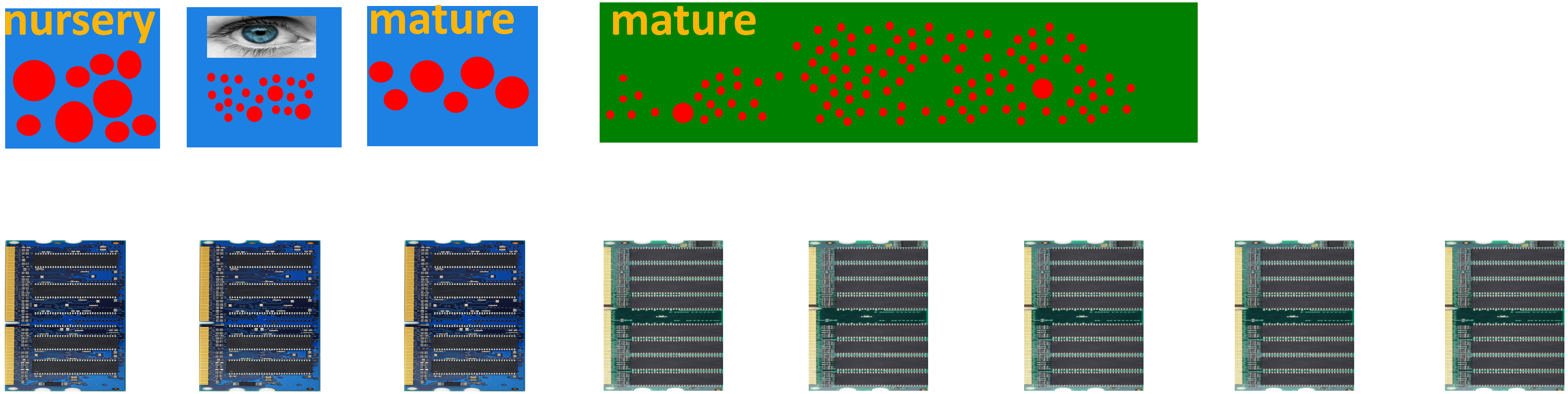




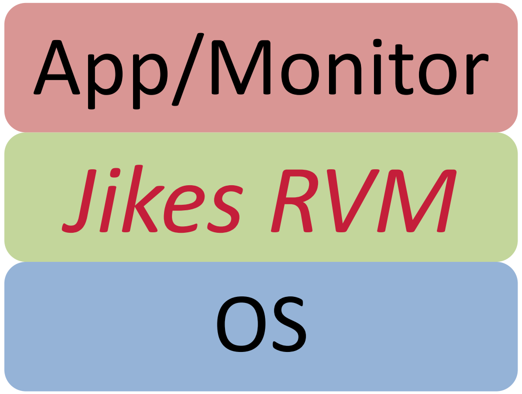
Kingsguard-Writers (KG-W)

KG-W monitors writes in a DRAM observer space

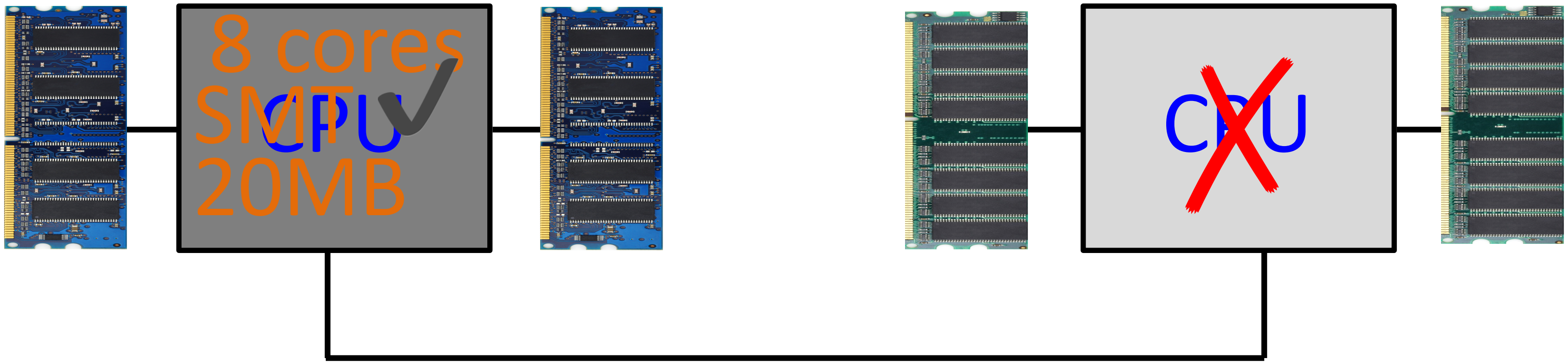
Trades off performance for better endurance



Emulation setup



Monitor: Intel pcm-memory.x to get per-socket write rate



Emulation versus simulation

PCM write reduction with KG-N and KG-W versus PCM-Only

Execution time increase with KG-W versus KG-N

No OS in simulation
Faithfully model emulator

Reduction in PCM writes with KG-N and KG-W versus PCM-Only

Kingsguard collectors limit PCM writes

KG-W much better than KG-N

	Simulation	Emulation
KG-N	4%	8%
KG-W	62%	64%

Increase in execution time with KG-W versus KG-N

KG-W is slower than KG-N because it monitors writes to objects

	Simulation	Emulation
KG-W	+7%	+10%

Graph workload evaluation

GraphChi: Analyze BIG graphs on a single machine

- Both Java and C++ implementations

Page Rank *and* Connected Components

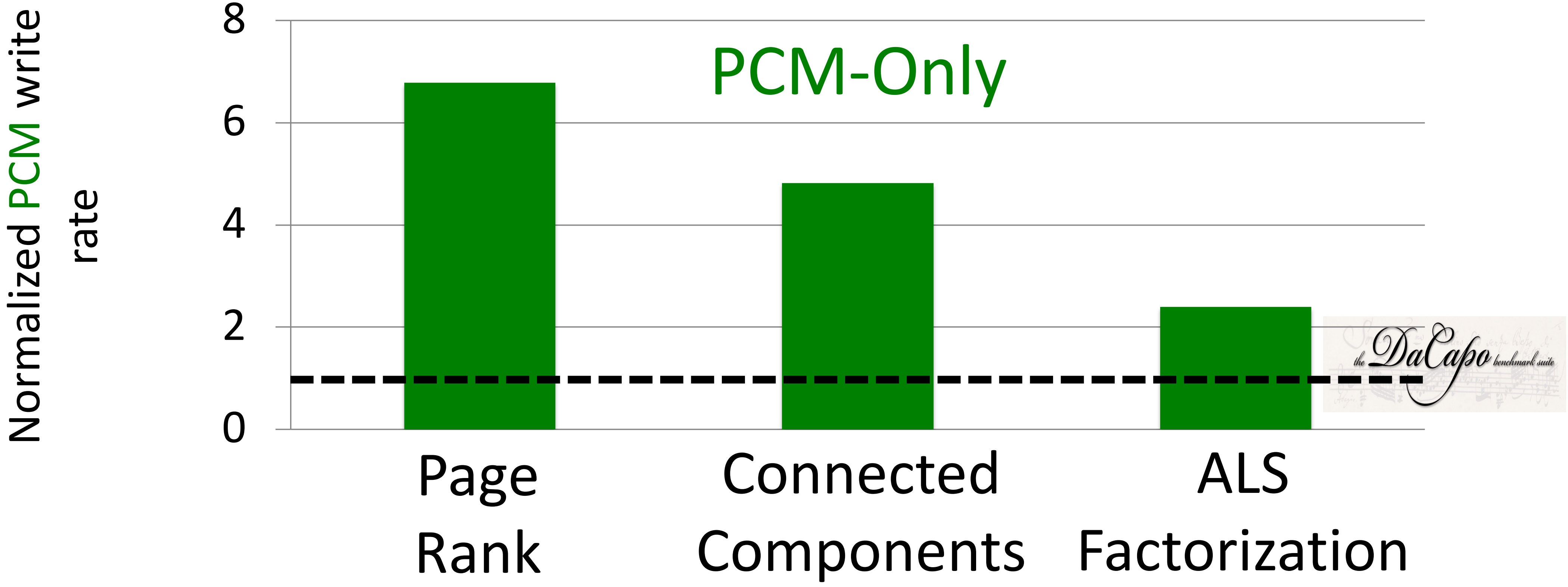
- LiveJournal social network

ALS Factorization

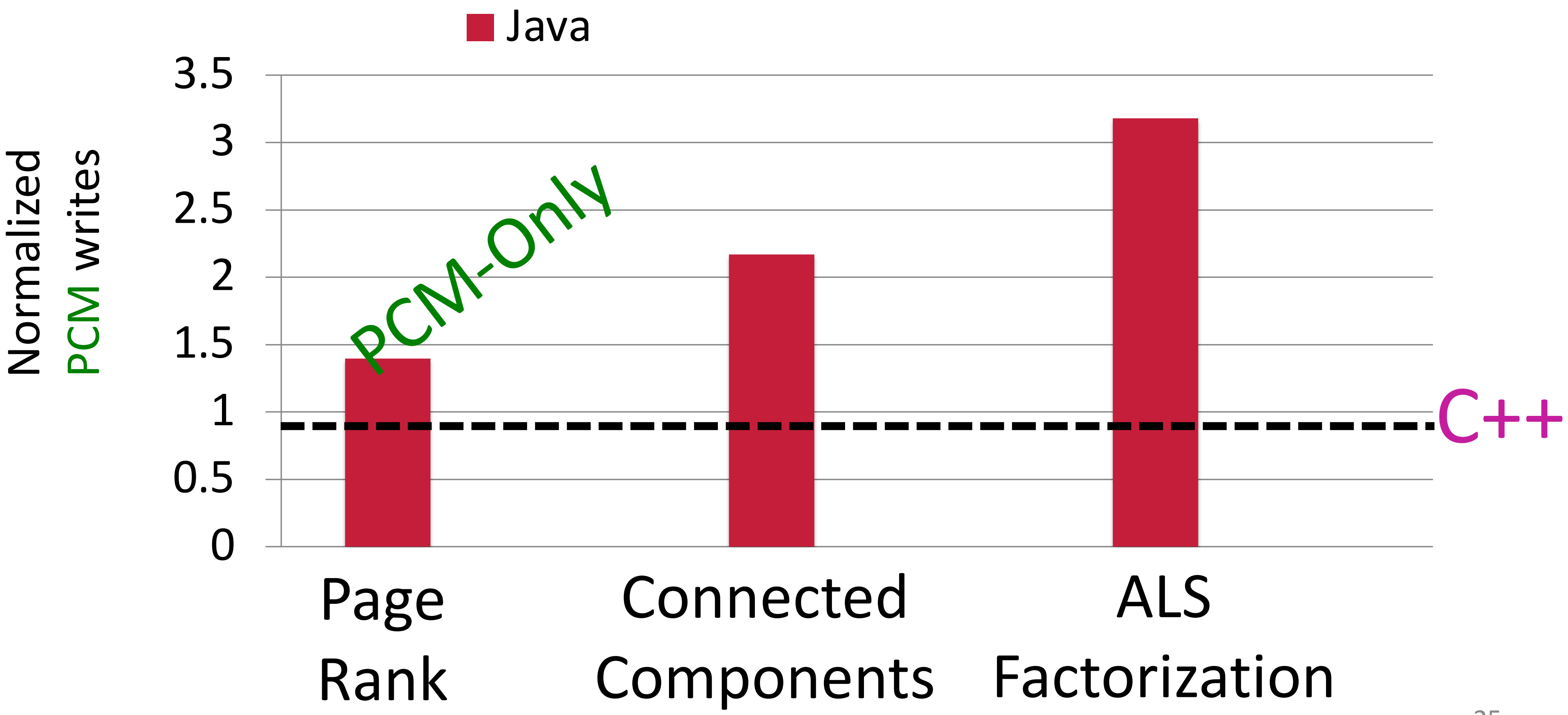
- Netflix challenge

Graph apps write more than DaCapo

Billions of vertices → Billions of objects



Java writes more to PCM than C++

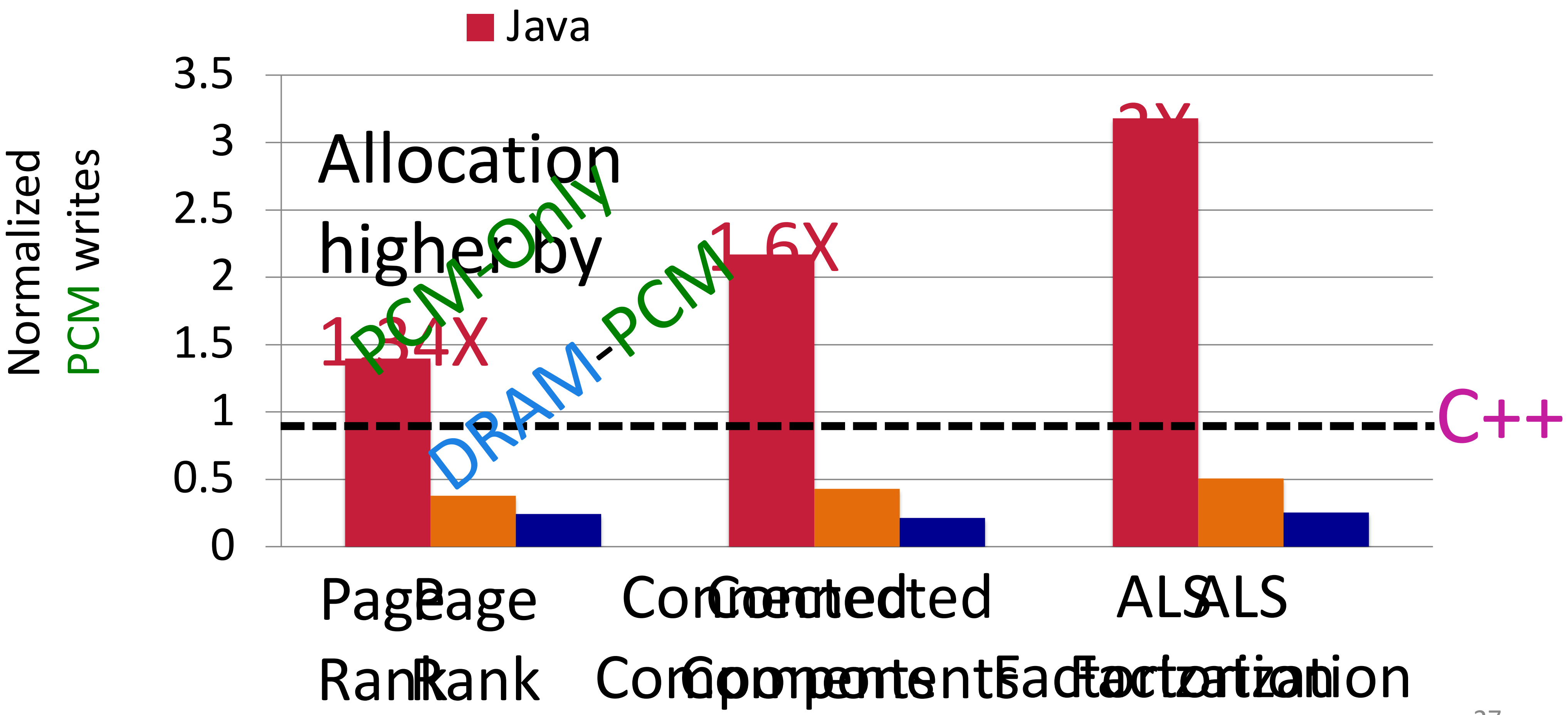


Java writes more to PCM than C++

Reasons

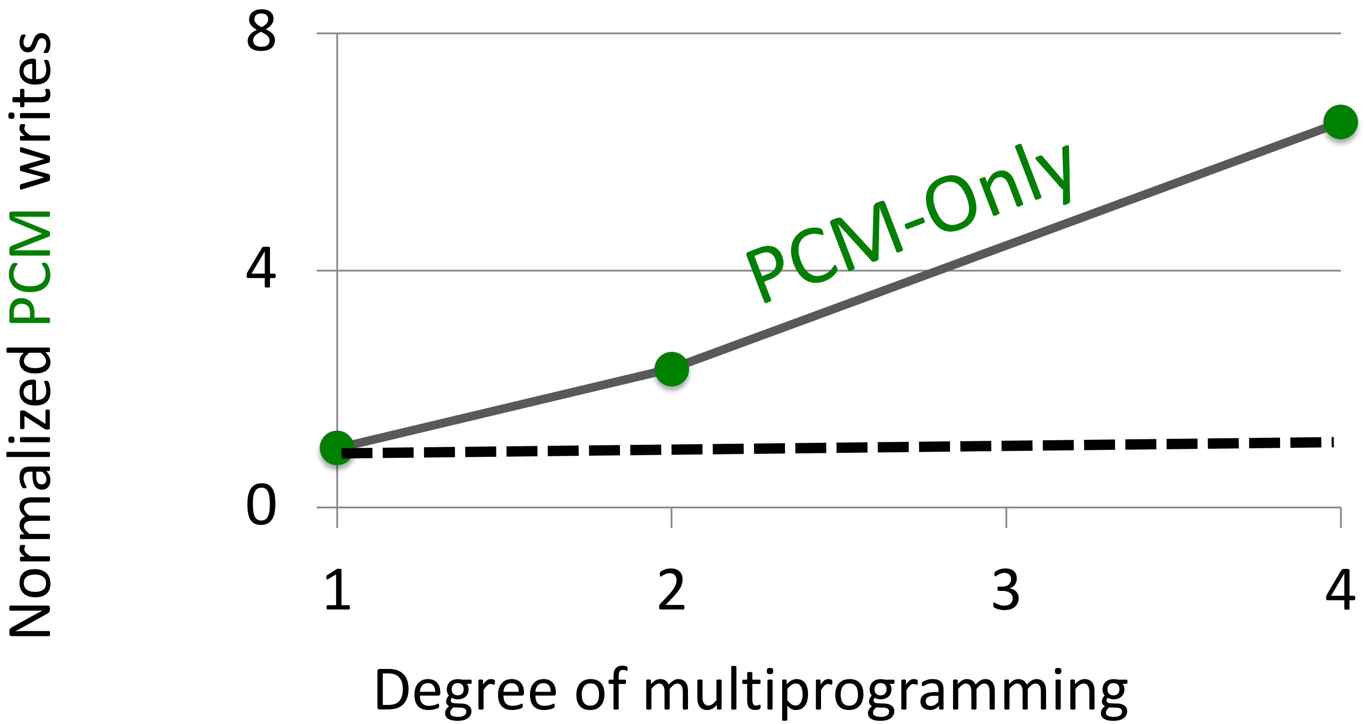
- Higher allocation volume →
- Copying between heap spaces
- Zeroing to provide memory safety

Java writes more to PCM than C++

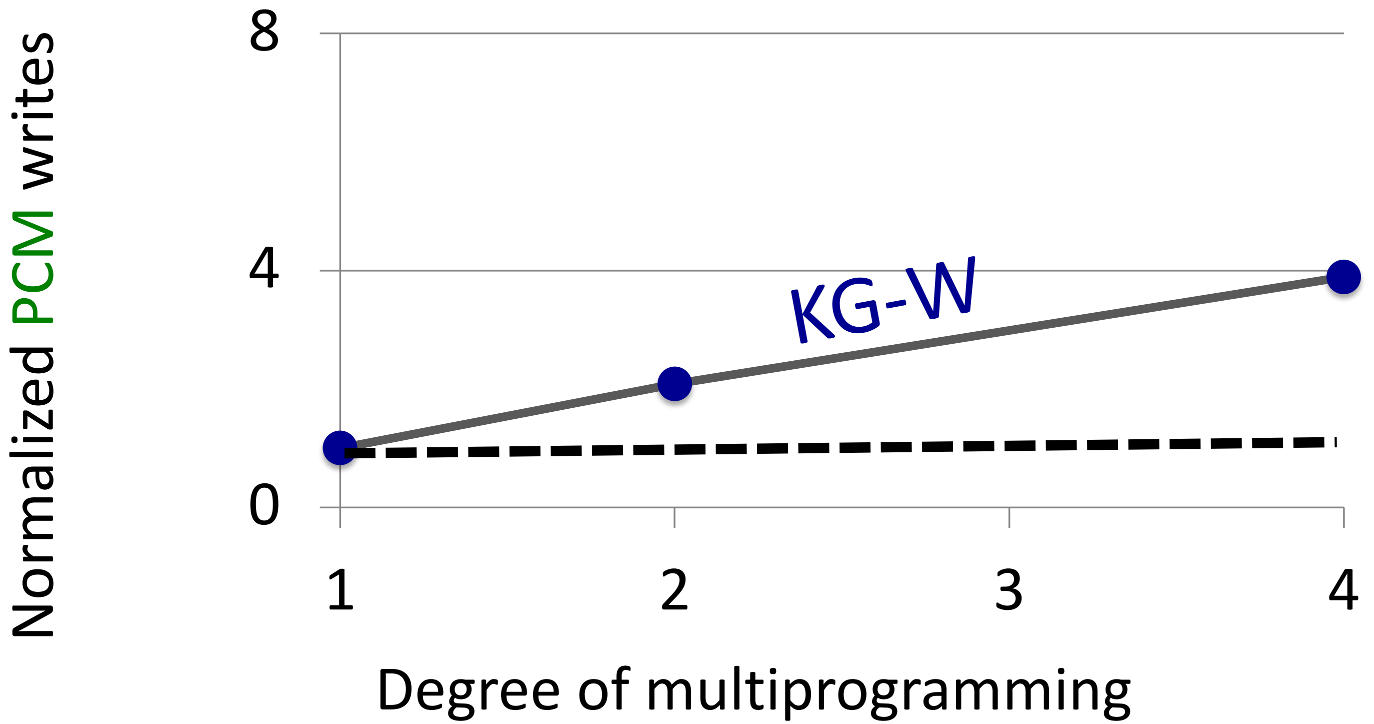


Allocation higher by

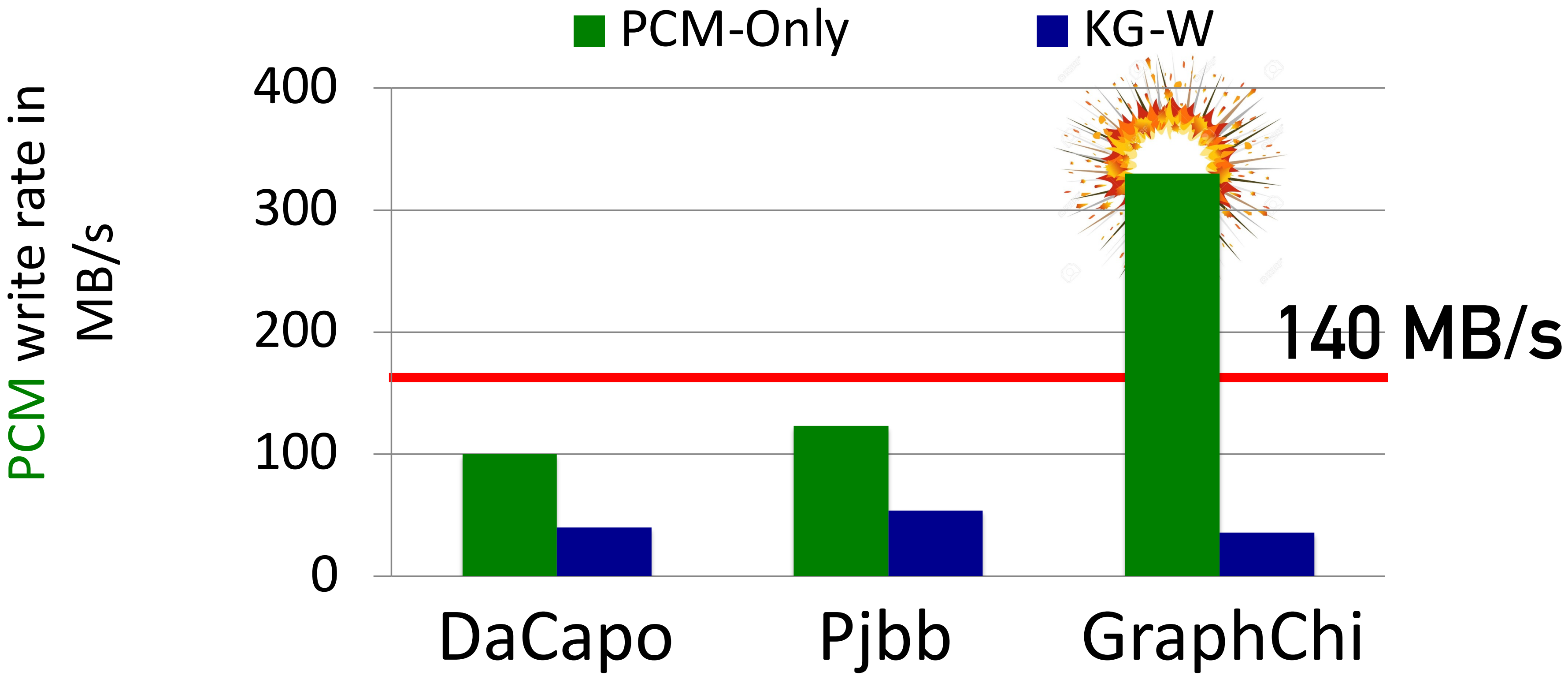
Writes increase **super-linearly** due to multiprogramming with PCM-Only



Writes increase linearly due to multiprogramming with KG-W

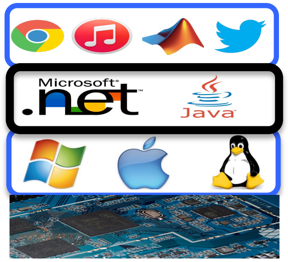


PCM-Only is not practical as main memory



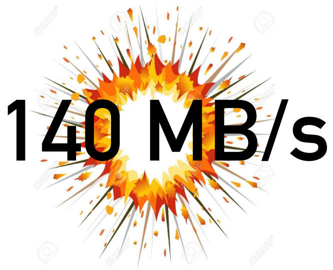
Conclusions

Across the stack emulation of hybrid memory



Similar outcomes with different evaluation methods

More research to make PCM practical as main memory



Emulating and Evaluating Hybrid Memory for Managed Languages on NUMA Hardware

Shoaib Akram (Ghent), Jennifer B. Sartor (Ghent and VUB), Kathryn S. Mckinley (Google), and Lieven Eeckhout (Ghent)

Shoaib.Akram@UGent.be

GHENT UNIVERSITY

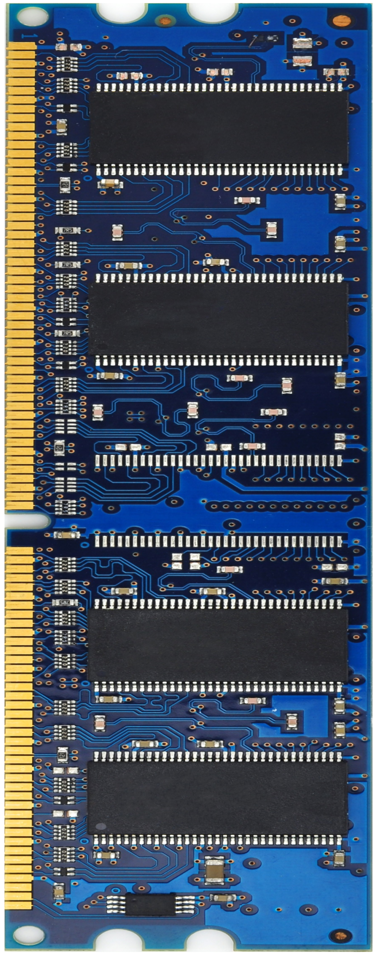
VUB VRIJE UNIVERSITEIT BRUSSEL

Google

DRAM is facing challenges

Scalability

Reliability

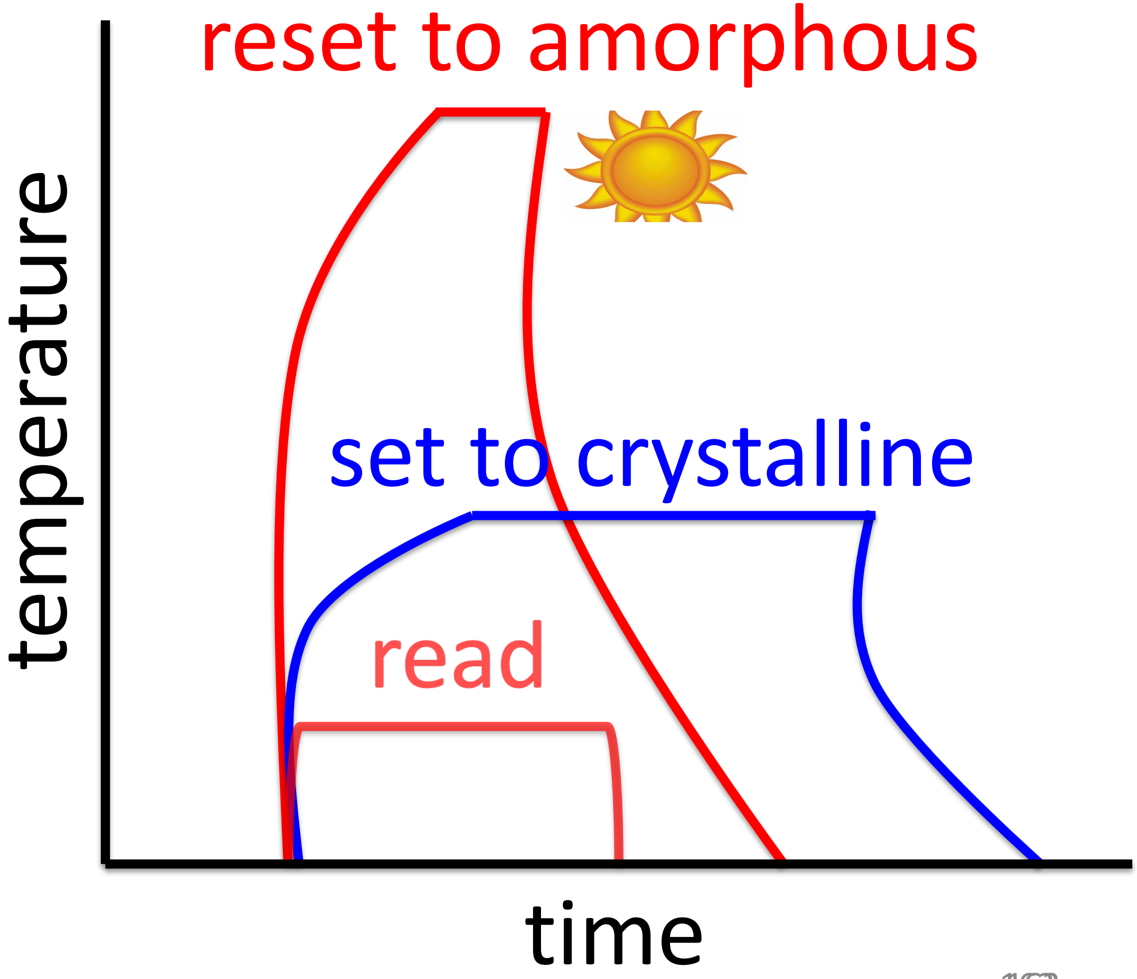


Phase change memory

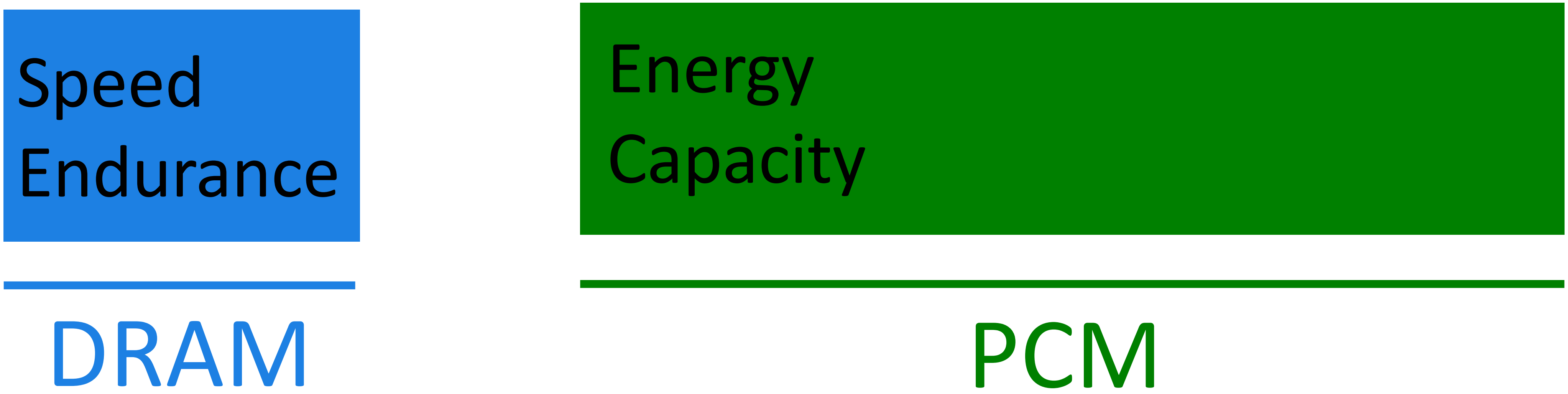
GB/$ ☺

Latency ☹

Endurance ☹



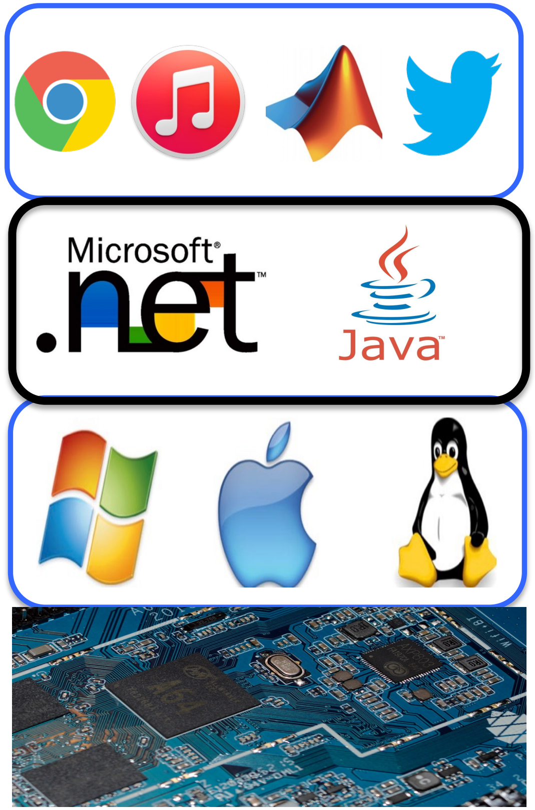
Hybrid DRAM-PCM memory



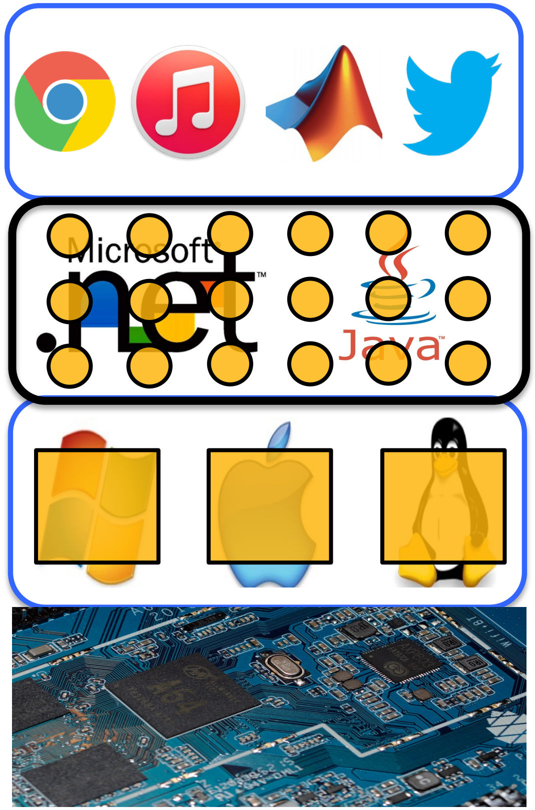
Mitigate PCM wear-out

Bridge the DRAM-PCM latency gap

Abstractions for hybrid memory



Abstractions for hybrid memory



Garbage Collection

Virtual Memory

DRAM Cache

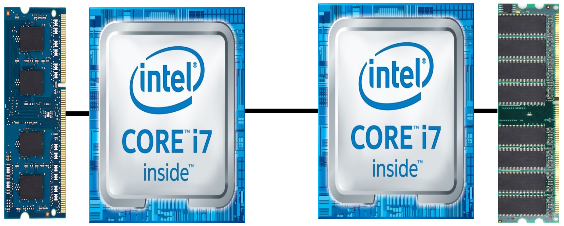
Wear Level

Hybrid memory evaluation

PageRank over Twitter network graph

GraphChi on commodity hardware

User-level simulation with mechanistic models

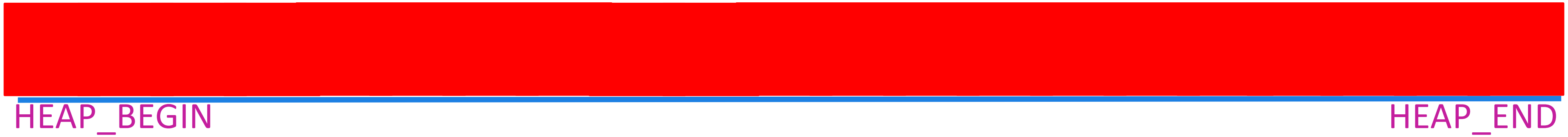


Emulation on NUMA machine

Prior art: latency ✓ C/C++ ✓

DRAM heap management

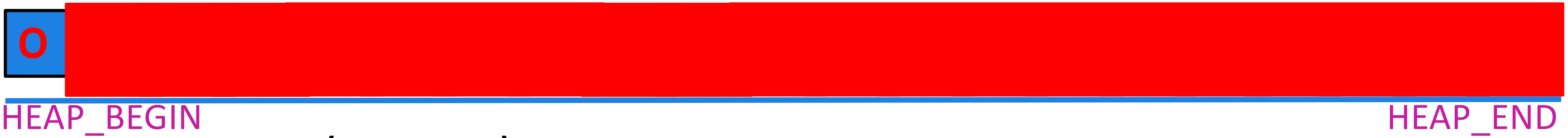
Heap Tracking (Chunks)



```
MegaCity m = new MegaCity(“Madison”)
```

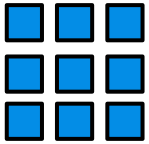
DRAM heap management

Heap Tracking (Chunks)

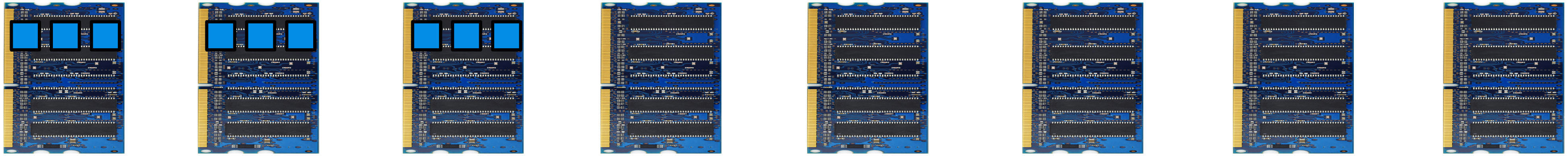


Heap Organization (Spaces)

OS Memory (pages)

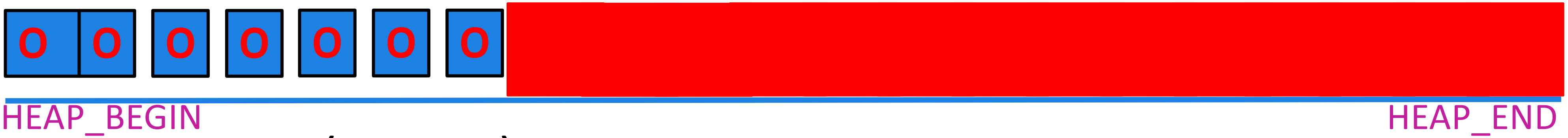


Physical Memory (frames)

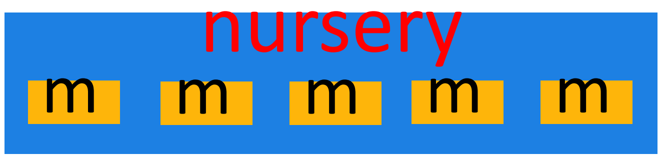


DRAM heap management

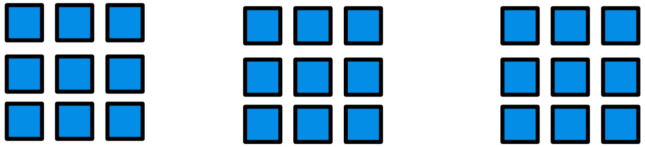
Heap Tracking (Chunks)



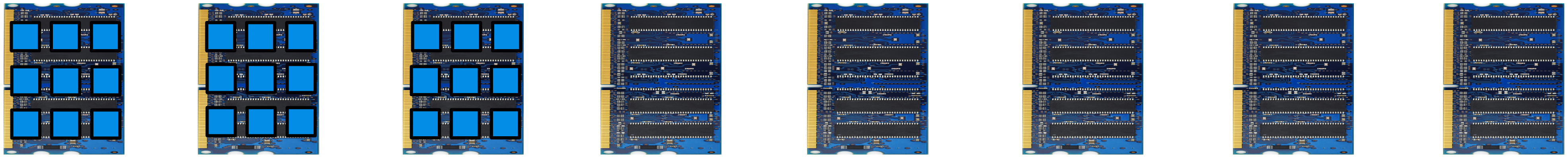
Heap Organization (Spaces)



OS Memory (pages)

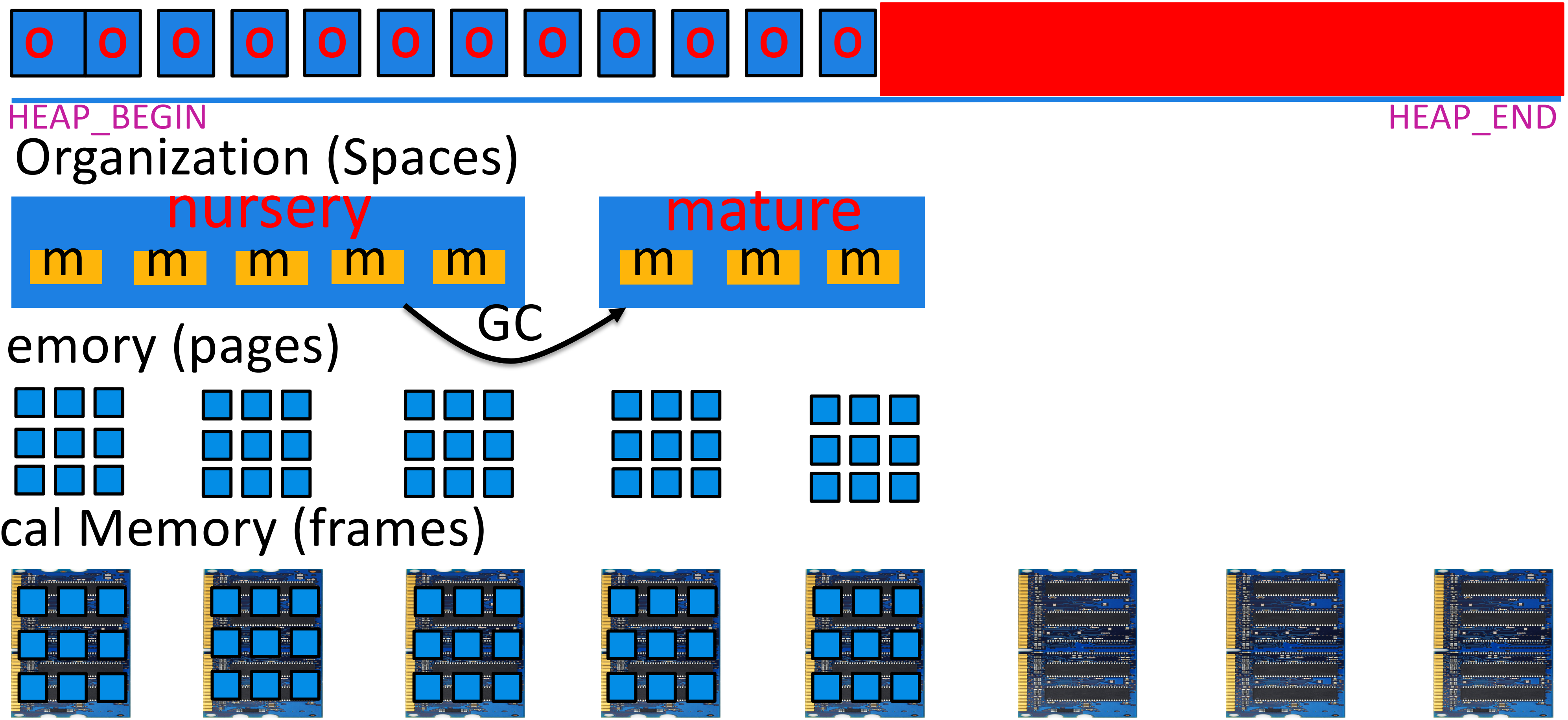


Physical Memory (frames)

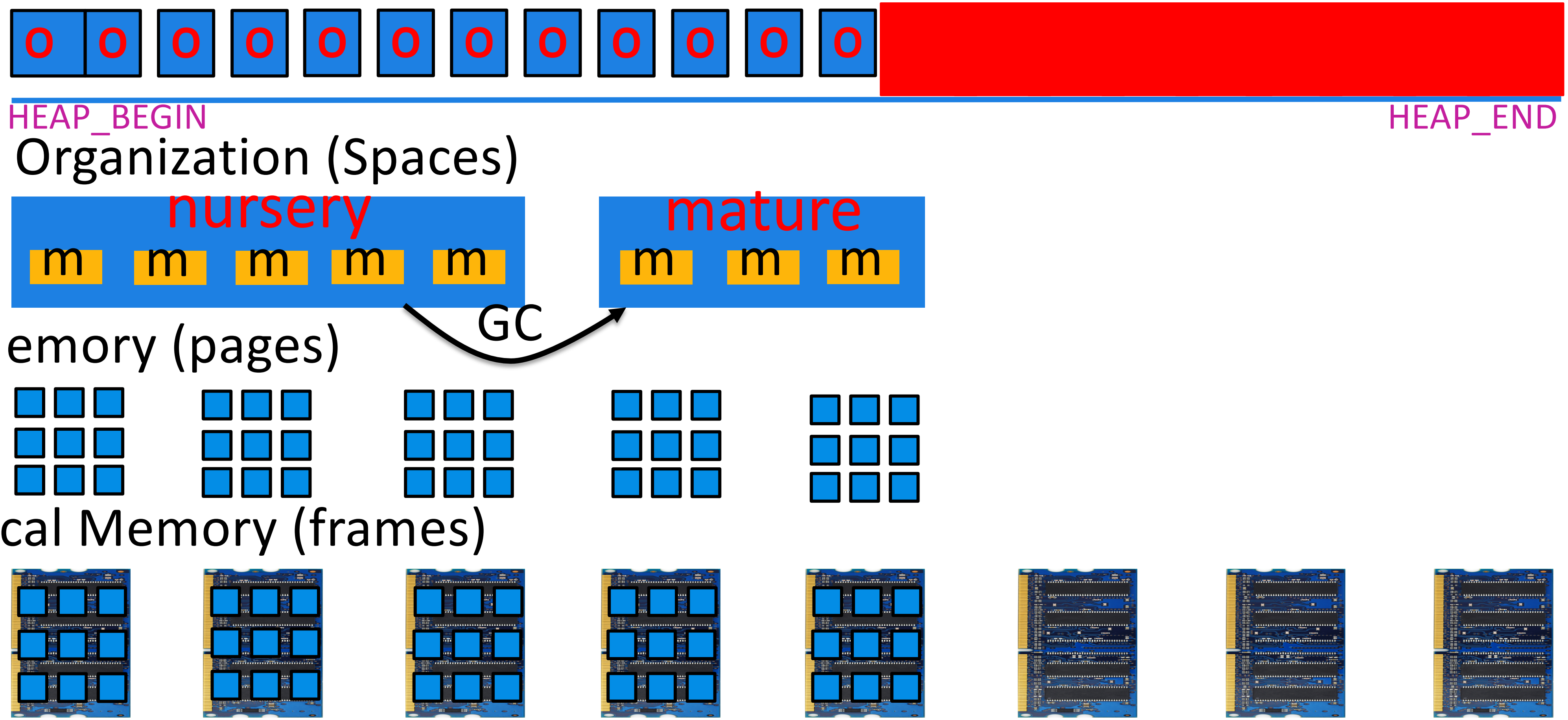


DRAM heap management

Heap Tracking (Chunks)



DRAM heap management



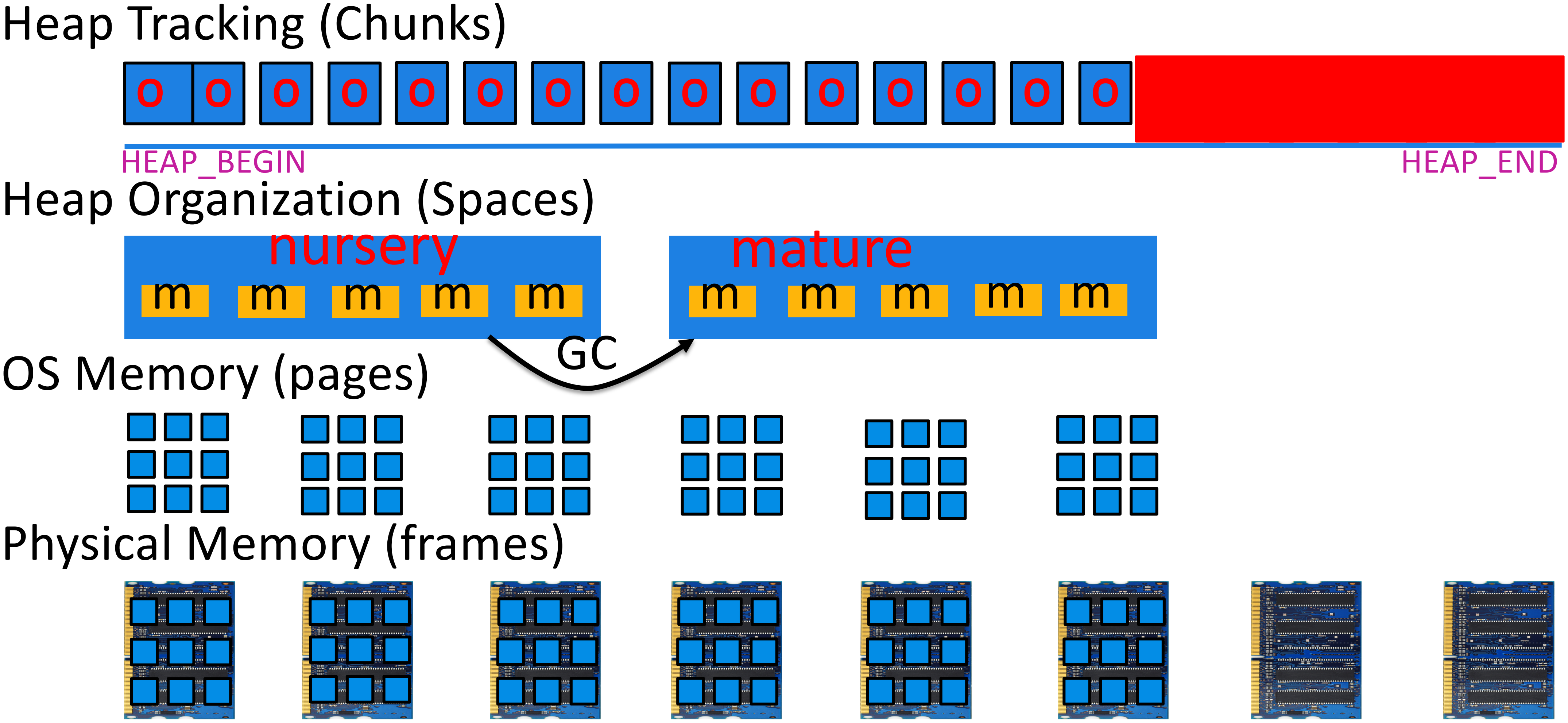
Heap Tracking (Chunks)

Heap Organization (Spaces)

OS Memory (pages)

Physical Memory (frames)

DRAM heap management



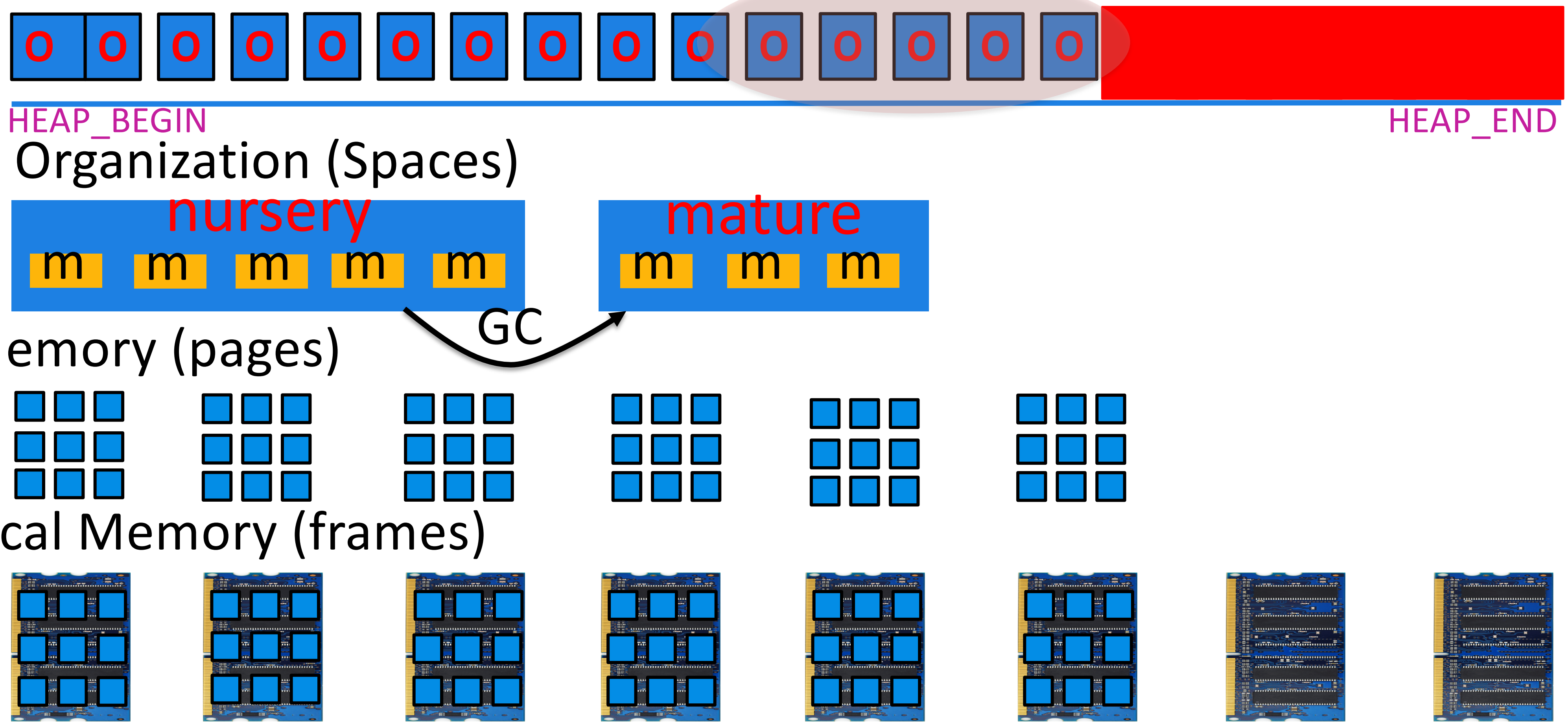
Heap Tracking (Chunks)

Heap Organization (Spaces)

OS Memory (pages)

Physical Memory (frames)

DRAM heap management



Heap Tracking (Chunks)

o o o o o o o o o o o o o o o

HEAP_BEGIN

HEAP_END

Heap Organization (Spaces)

nursery

m m m m m

mature

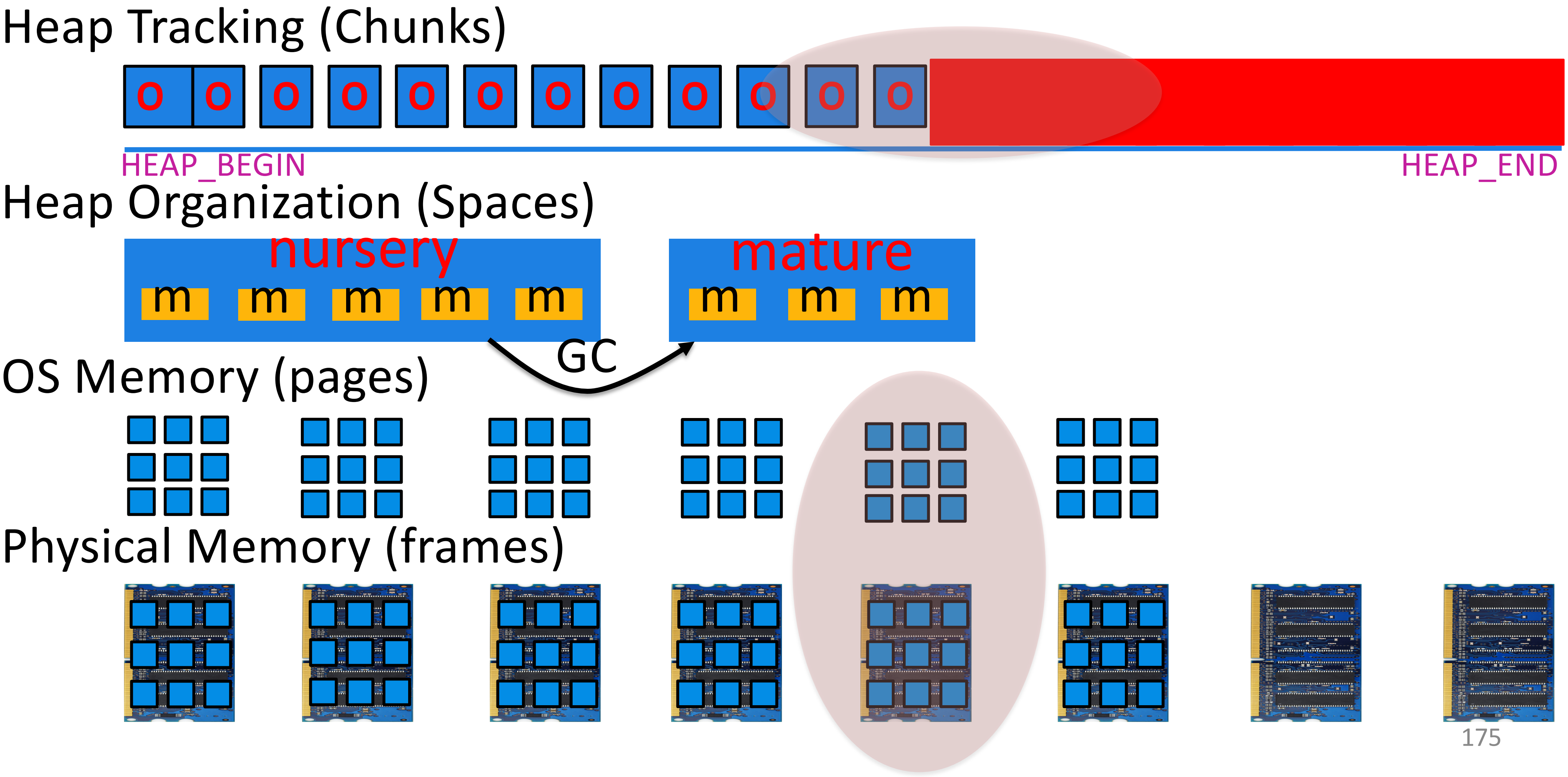
m m m

GC

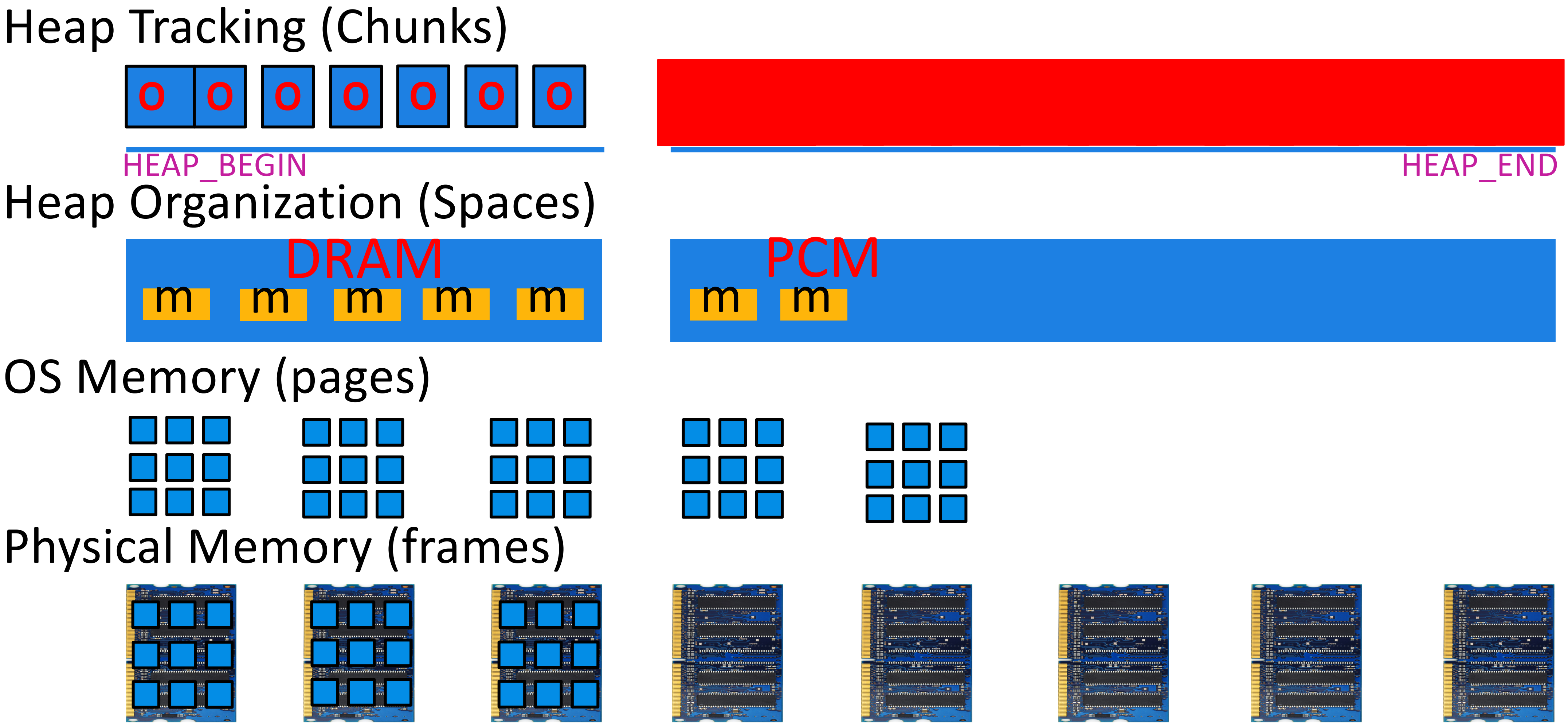
OS Memory (pages)

Physical Memory (frames)

DRAM heap management



DRAM heap management



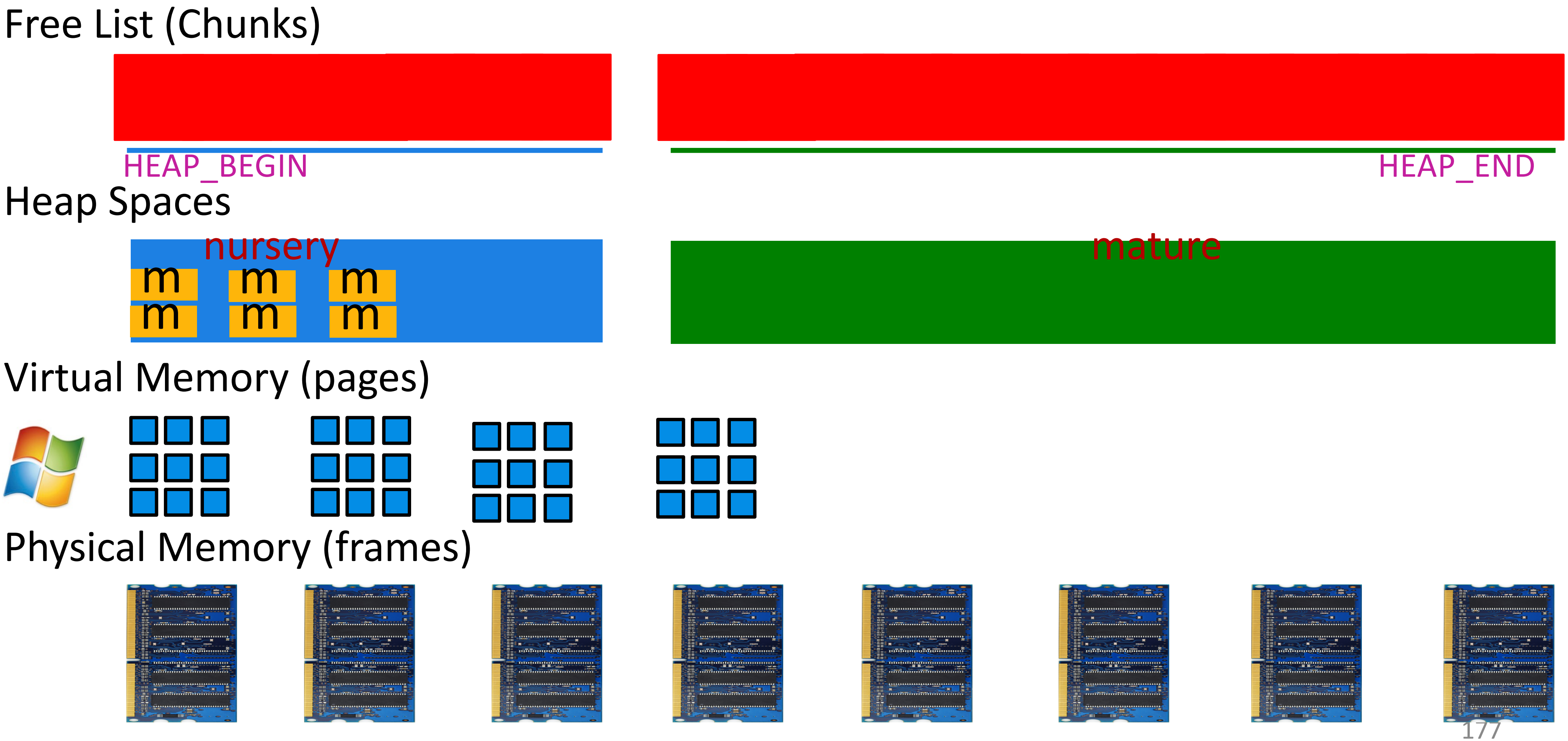
Heap Tracking (Chunks)

Heap Organization (Spaces)

OS Memory (pages)

Physical Memory (frames)

DRAM heap management



Free List (Chunks)

HEAP_BEGIN

HEAP_END

Heap Spaces

nursery

m m m

m m m

mature

Virtual Memory (pages)

Physical Memory (frames)

NUMA platform for emulation

Validation against Sniper

More PCM writes with Java than C++

Co-running apps increase PCM writes

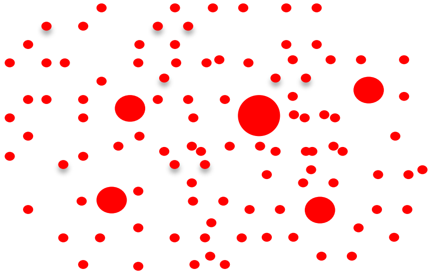
Graph apps write more than DaCapo

PCM-Only is impractical

KG-N and KG-W limit PCM writes

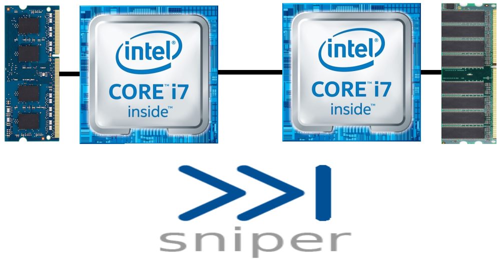
Takeaways

Monitoring heaps at a fine granularity is promising

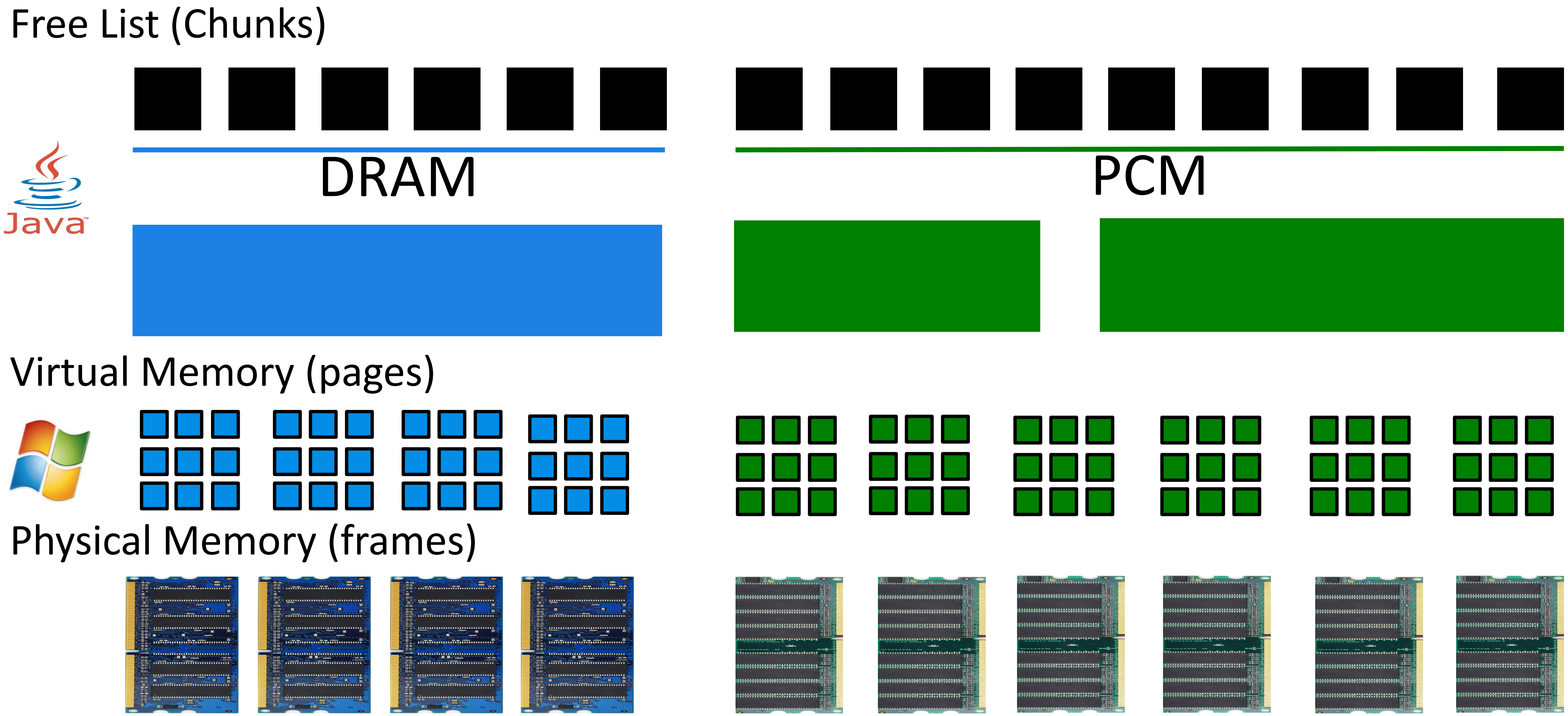


Write-rationing garbage collection make PCM practical as main memory

Similar conclusion with 3 distinct methods



Heap management in DRAM-Only



Free List (Chunks)

DRAM

PCM

Virtual Memory (pages)

Physical Memory (frames)

Hardware platform

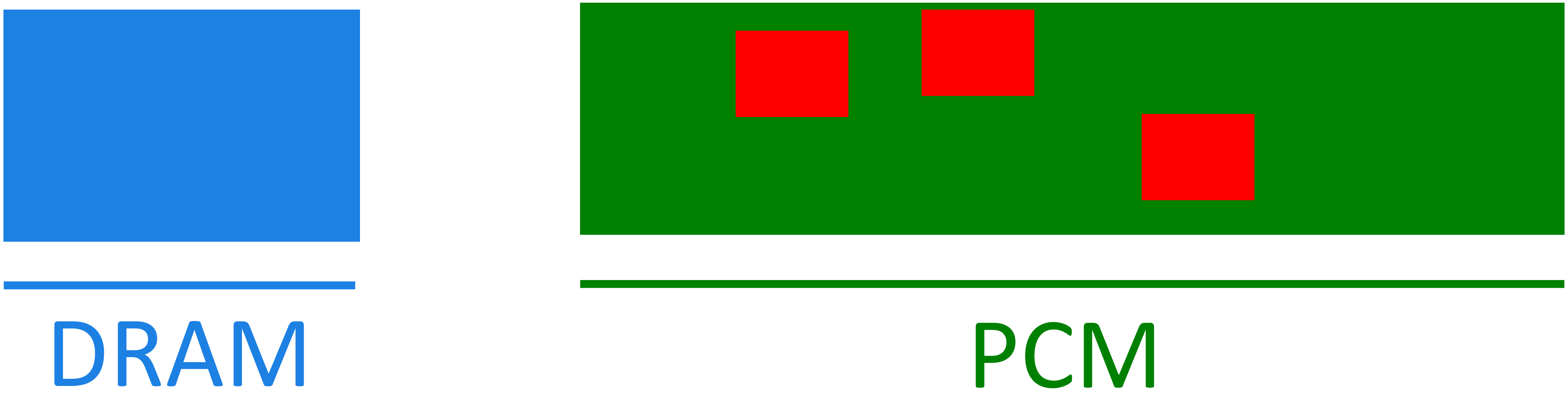
How to evaluate hybrid memory?

	Simulation	Emulation
Speed	Slow	Native
Diversity	Low	High
Full System	✗	✓
Realistic	✗	✓

How to evaluate hybrid memory?

	Simulation	Emulation
Full System	✗	
Realistic	✗	

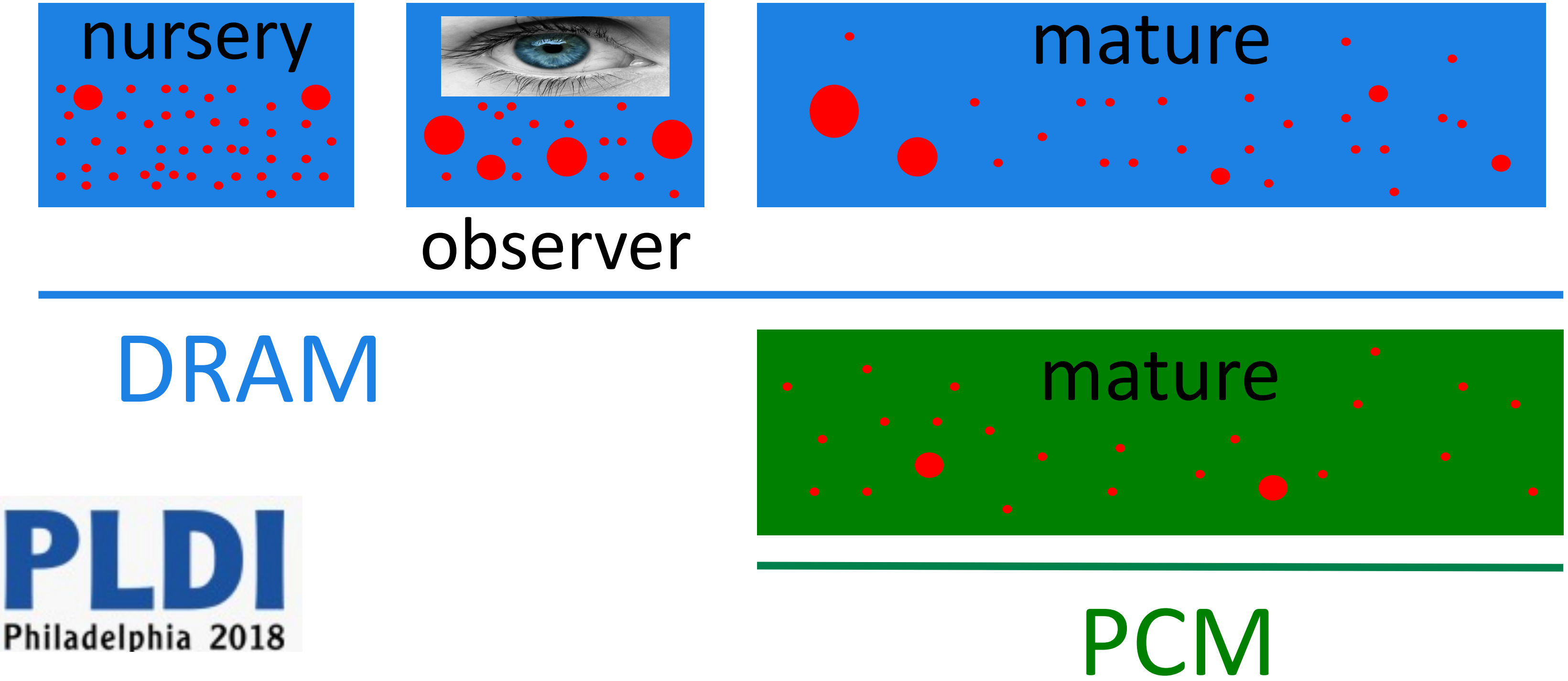
OS to limit PCM writes



Drawbacks

- Coarse granularity
- Costly page migrations

Managed runtime to limit PCM writes



nursery

observer

mature

DRAM

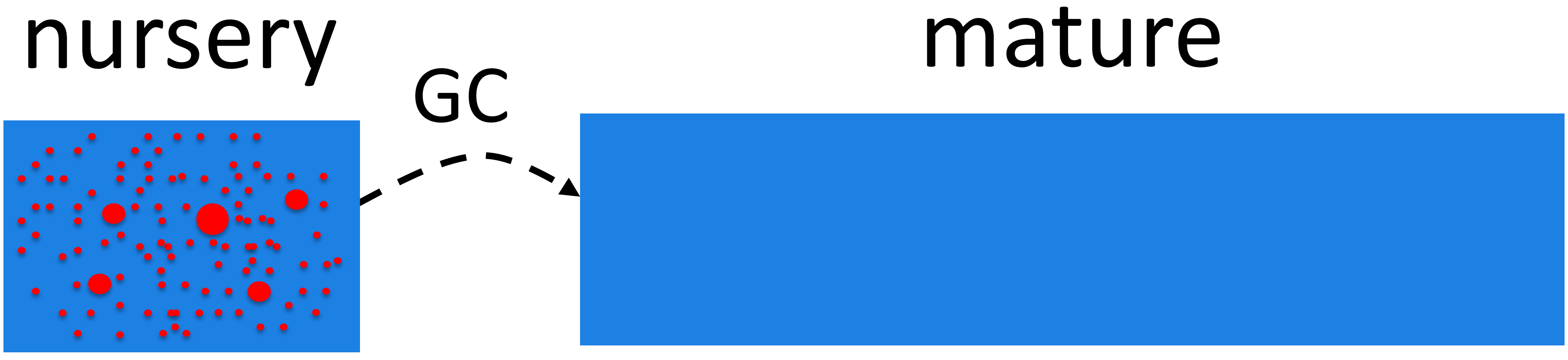
mature

PCM

PLDI Philadelphia 2018

Our work uses garbage collection to keep highly written objects in DRAM

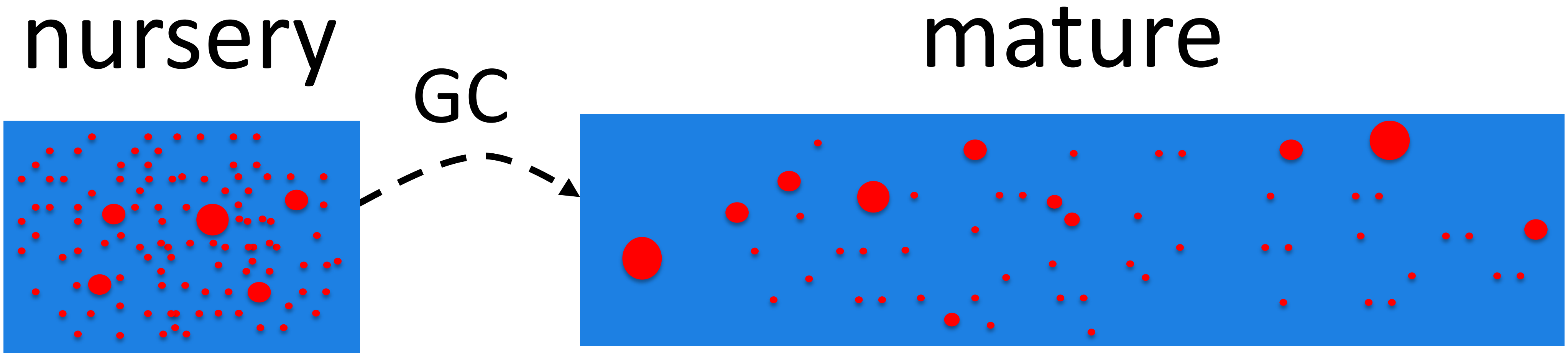
Distribution of writes in GC runtime



70%

of writes

Distribution of writes in GC runtime



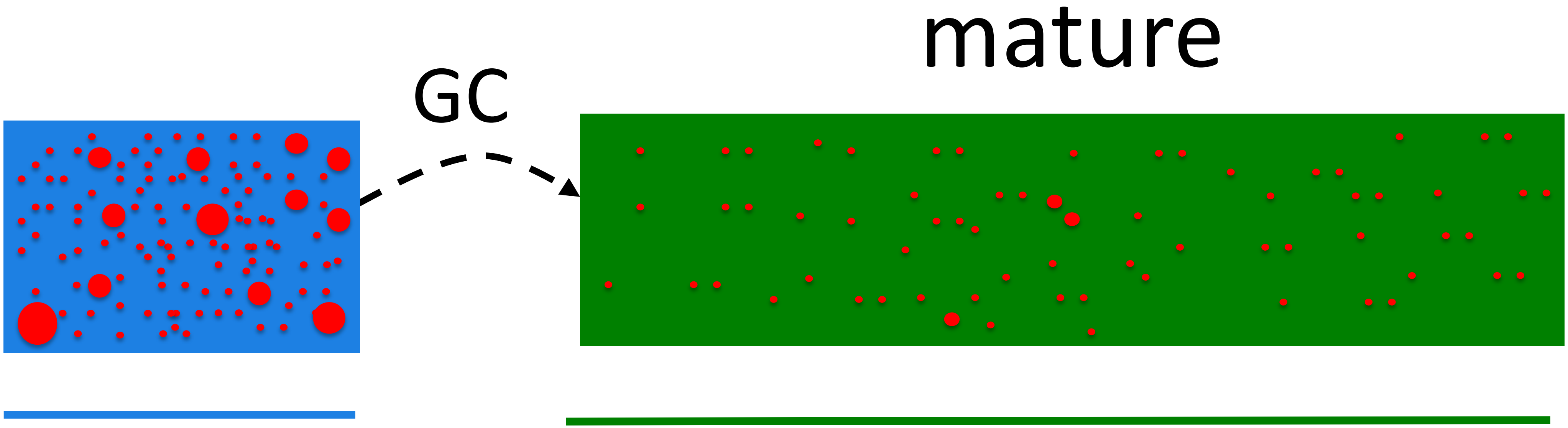
70%

of writes

22%

to 2% of objects

Contribution
Write-Rationing Garbage Collectors

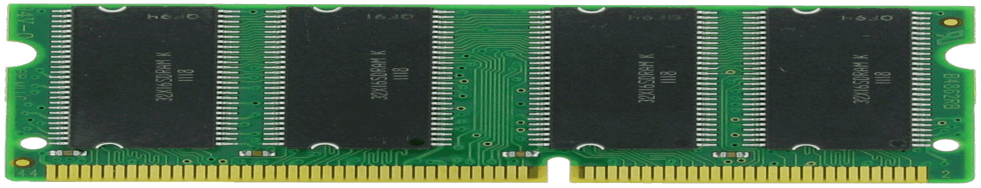


mature

GC

DRAM

PCM

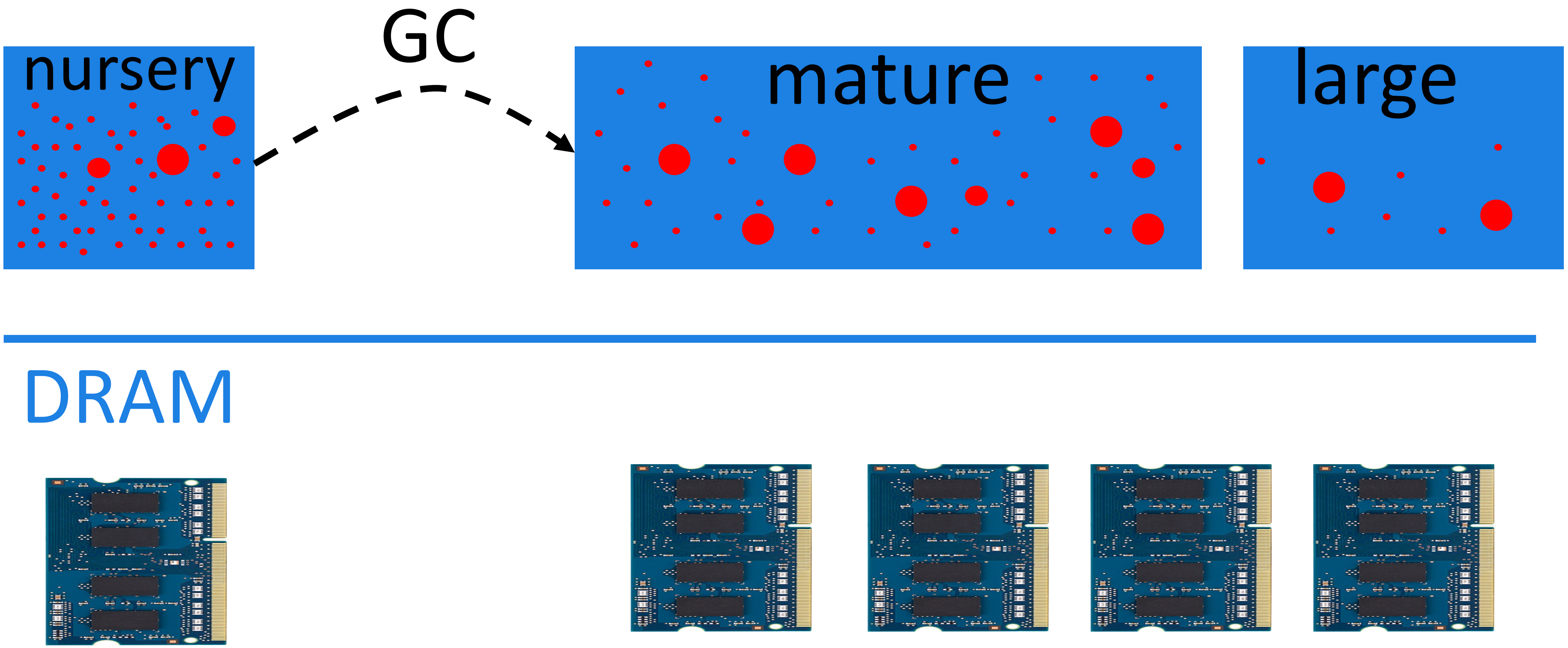


Two write-rationing garbage collectors

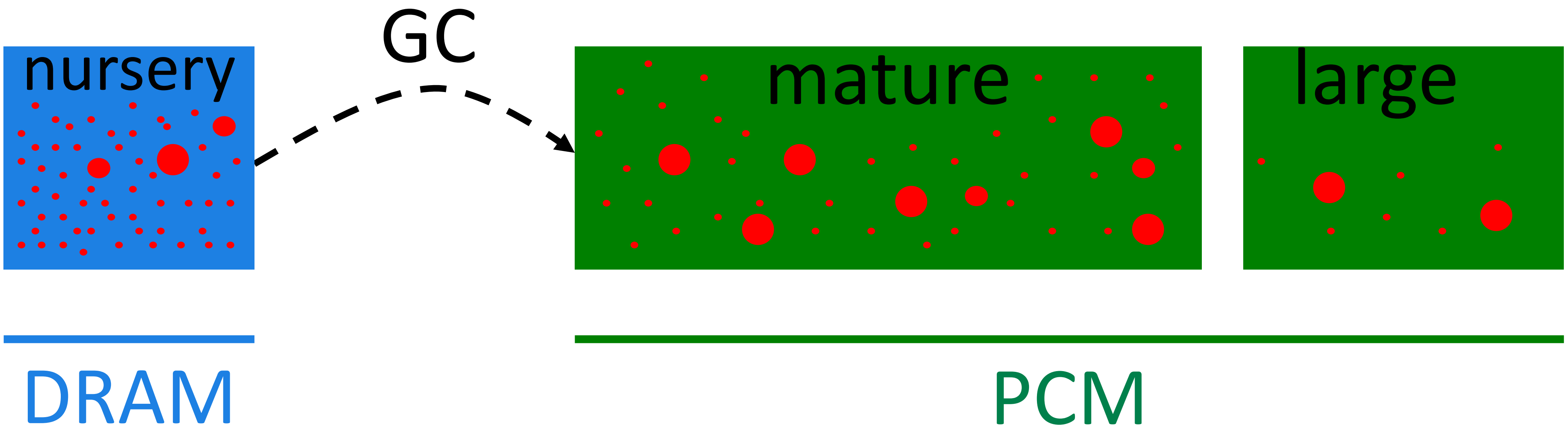
Kingsguard-Nursery

Kingsguard-Writers

Heap organization in DRAM



KG-N Kingsguard-Nursery



nursery

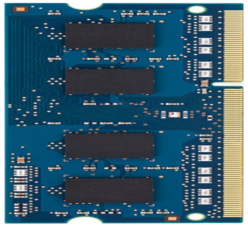
GC

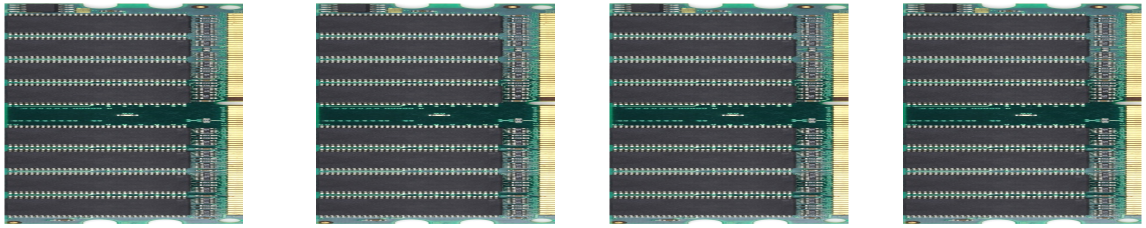
mature

large

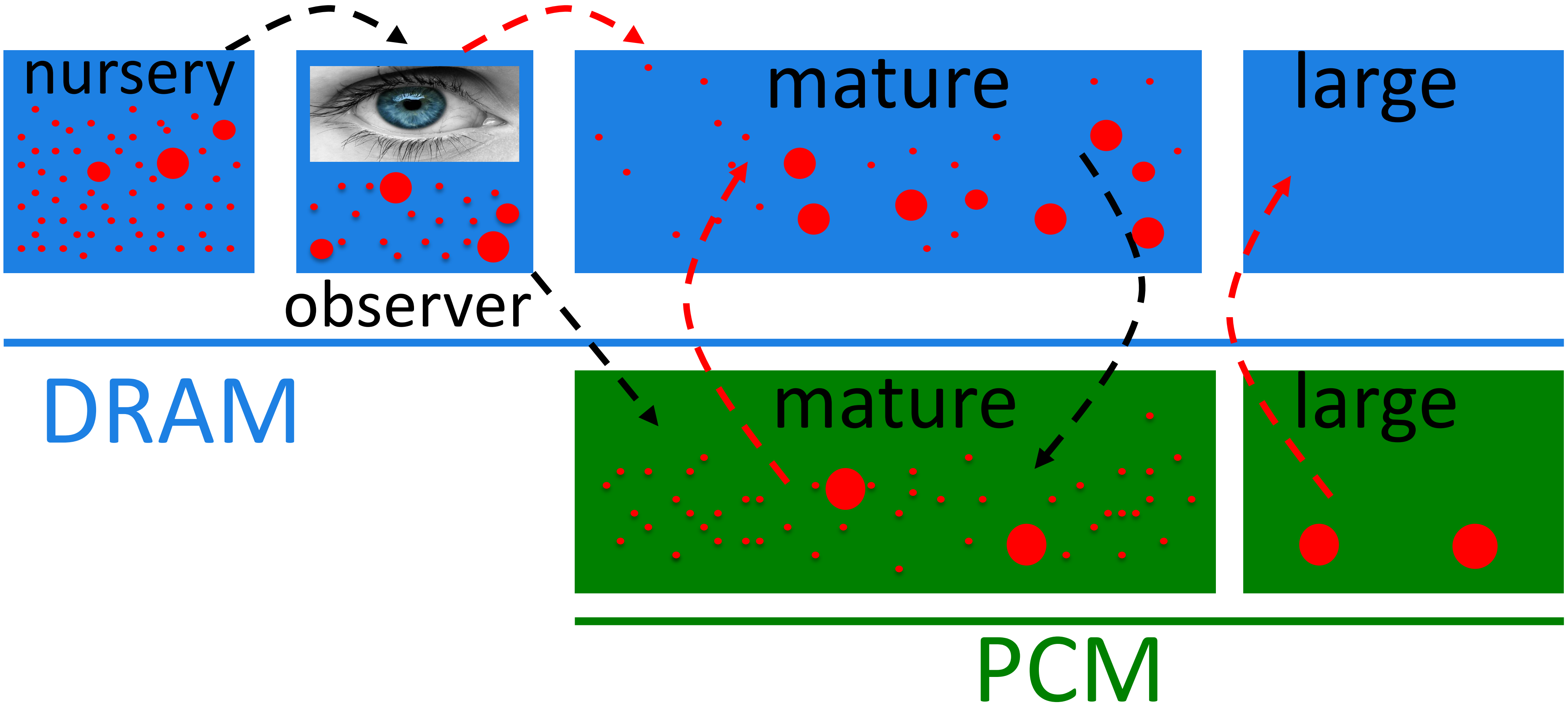
DRAM

PCM



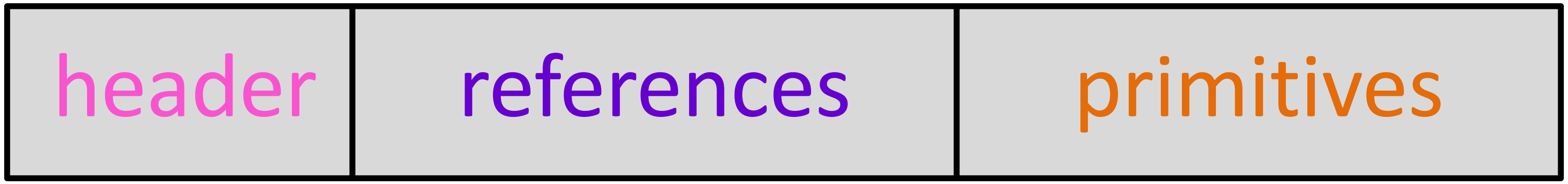


KG-W Kingsguard-Writers



Observing writes

Object format



header references primitives

Write barrier sets a header bit on object writes

Write barrier configurations

- Observe references
- Observe references and primitives

Two extra optimizations in KG-W

Large object optimization

- Allocate selected large objects in DRAM

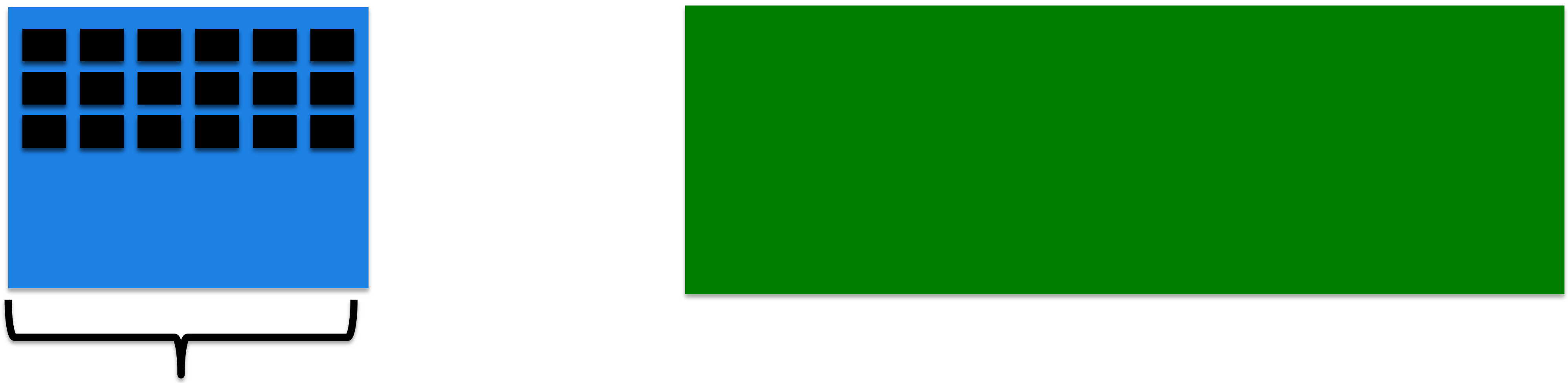
Metadata optimization

- Allocate PCM metadata in DRAM

Large object optimization

nursery

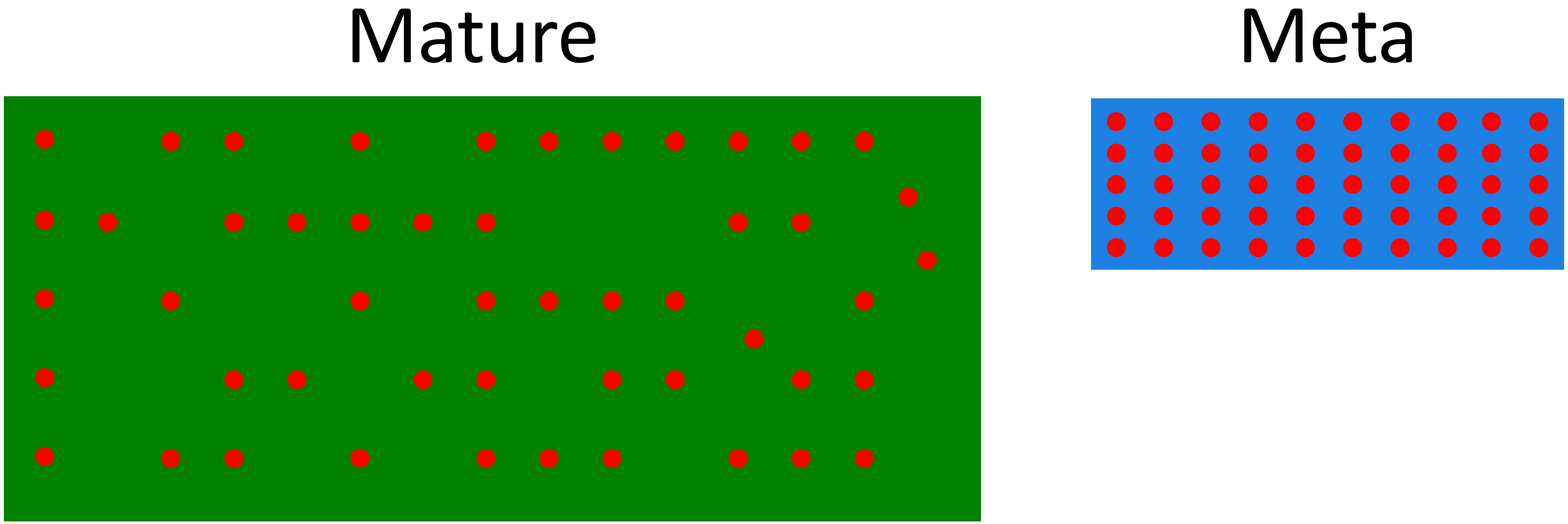
large



½ of remaining nursery

Monitor PCM write rate to tu[redacted]

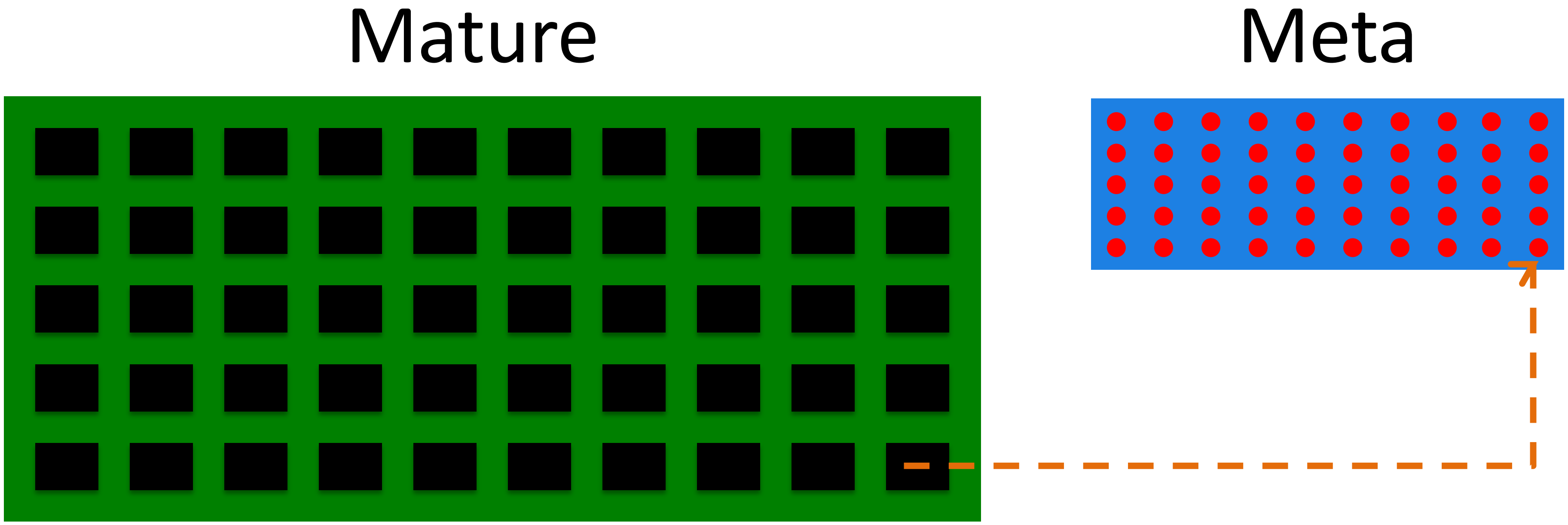
Metadata optimization



Full-heap GC: Mark live PCM objects

KG-W: Keep mark bits of PCM objects in DRAM

Metadata optimization



Full-heap GC: Mark live PCM objects

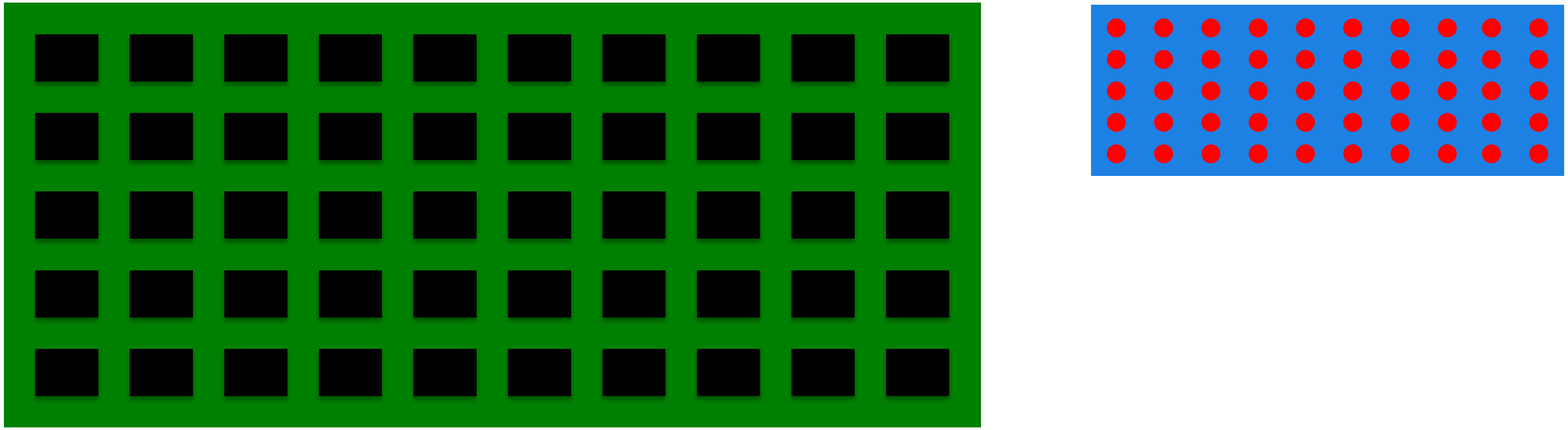
KG-W: Keep mark bits of PCM objects in DRAM

address_mark_bit = start_meta + idx_pcm_obj

DRAM metadata overhead

Mature

Meta



Smallest object size is 4 B: 25% overhead

Common case size is > 16 B: 6.25% overhead

KG-W: Only use side meta for objects > 16 B

Evaluation Methodology

Hardware

(1) Simulator

(2) Real hardware

Software

Jikes research virtual machine

Java applications

Real hardware measurements

Use write barriers to count object writes

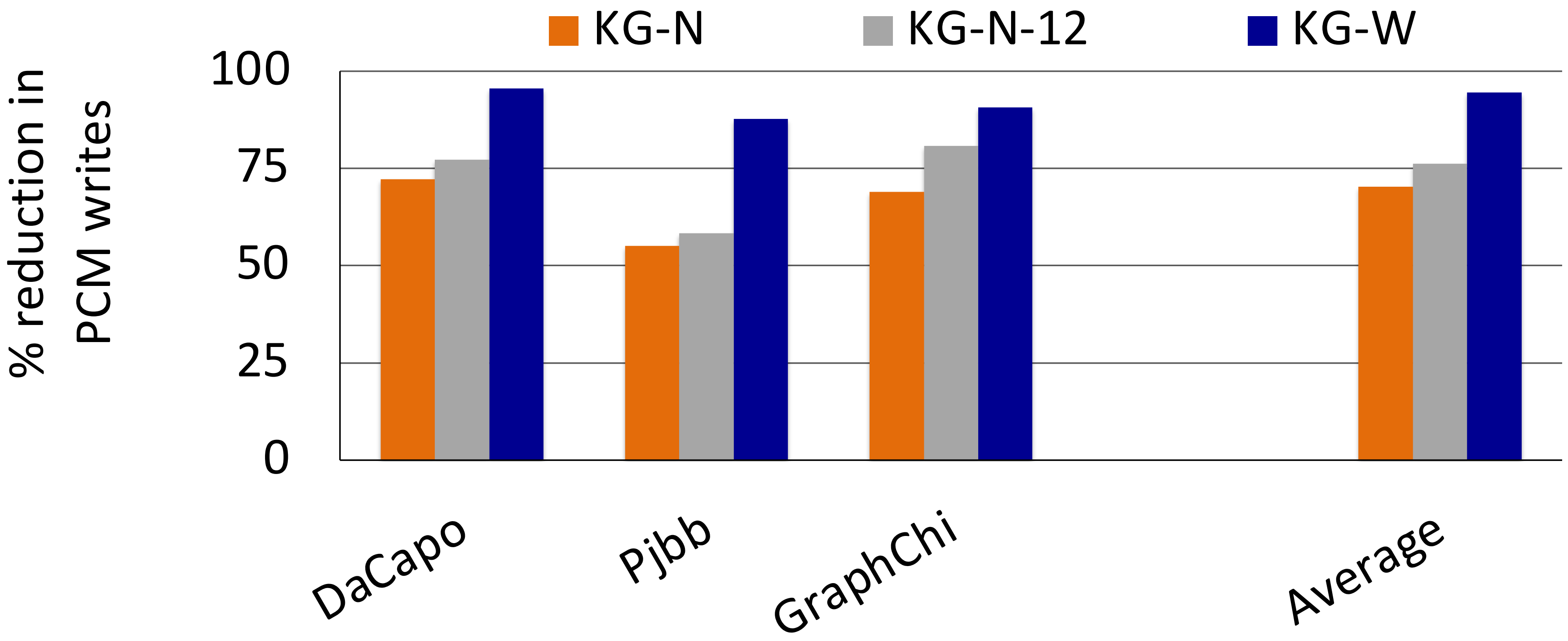
Applications: 12 DaCapo, 3 GraphChi, and Pjbb

Configurations

- KG-N : 4 MB nursery
- KG-W: 4 MB nursery, 8 MB observer
- KG-N : 12 MB nursery

Reduction in PCM writes

Baseline: PCM-Only



KG-W reduces 95% of writes to PCM

Simulation with Sniper

7 DaCapo applications

4 cores, 1 MB per core LLC

Scale simulated rates to a 32 core machine using trends from real hw

Memory systems

Homogeneous

- 32 GB DRAM
- 32 GB PCM

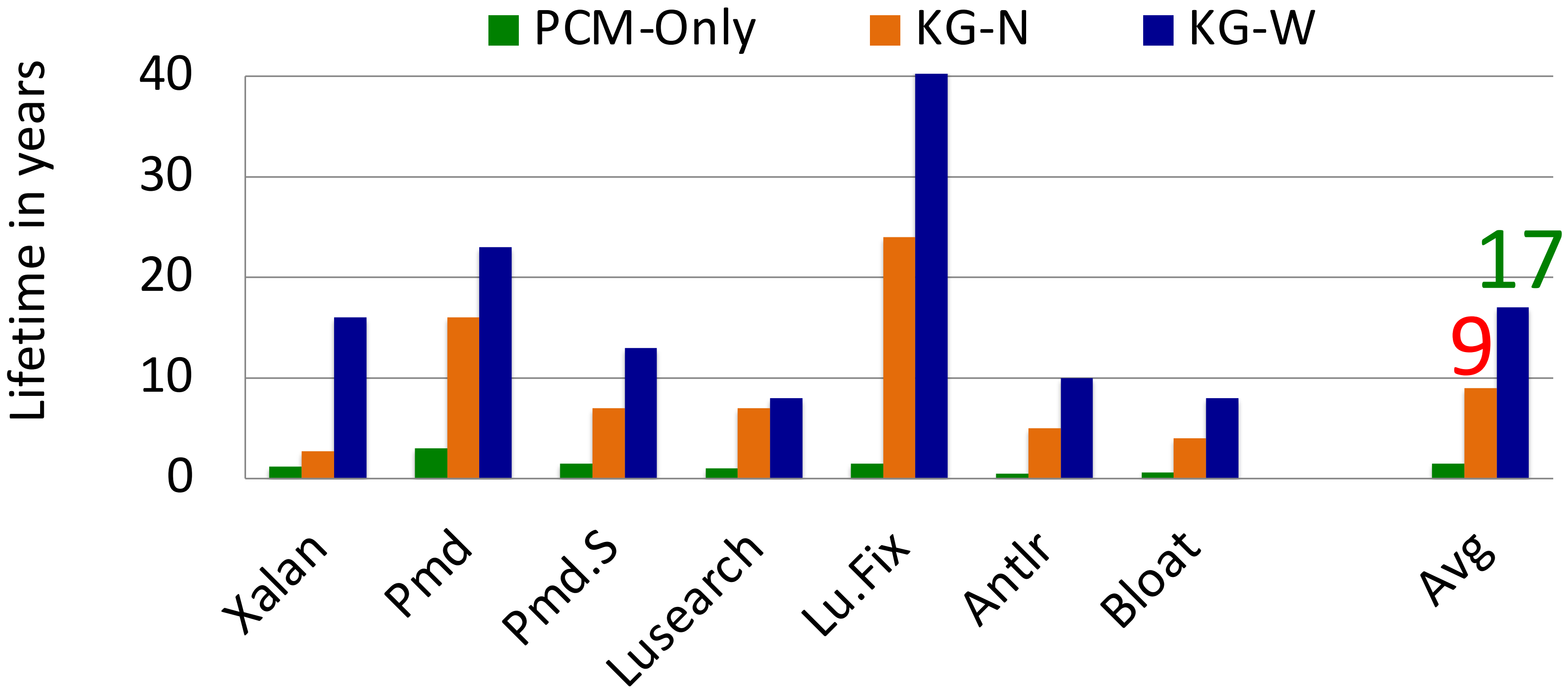
Hybrid

- 1 GB DRAM
- 32 GB PCM

PCM parameters

- 4X read latency
- 4X write energy
- 10 M writes/cell

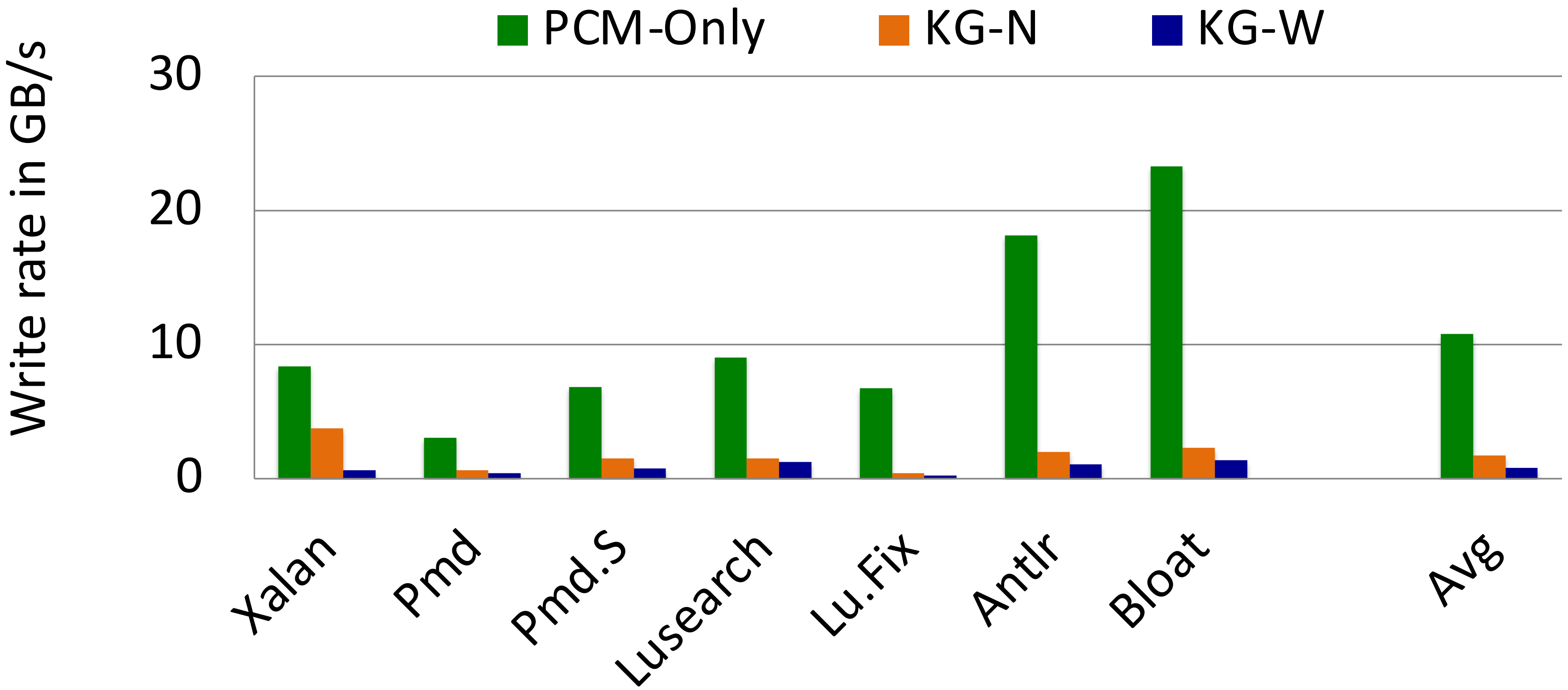
PCM lifetimes



PCM alone is not practical

PCM lasts more than 10 years with KG-W

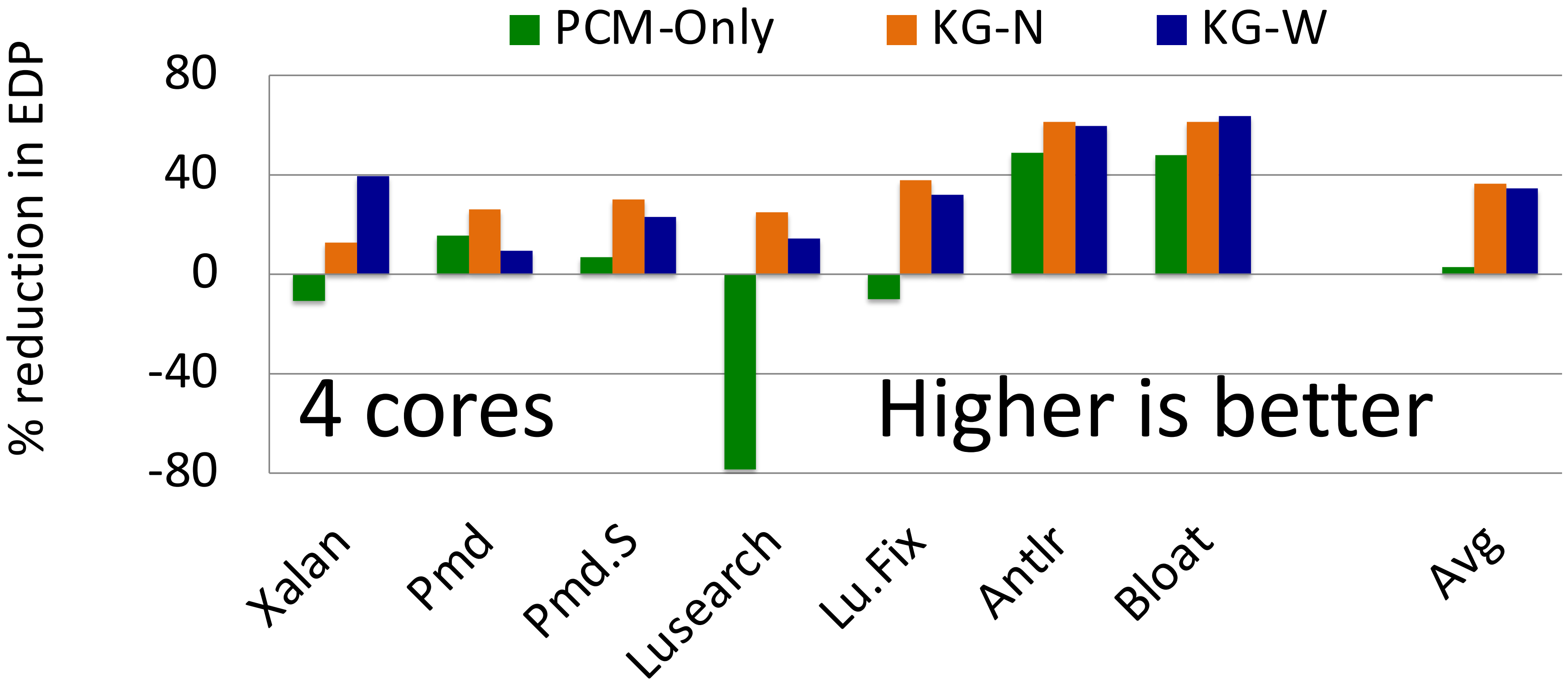
PCM write rates



KG-N reduces write rate by 6X over PCM-Only

KG-W reduces write rate by 2X over KG-N

EDP reduction compared to DRAM



EDP : Energy Delay Product

KG-W has 35% better EDP than DRAM-Only

Emulation on NUMA hardware

DRAM: Socket 0

PCM: Socket 1

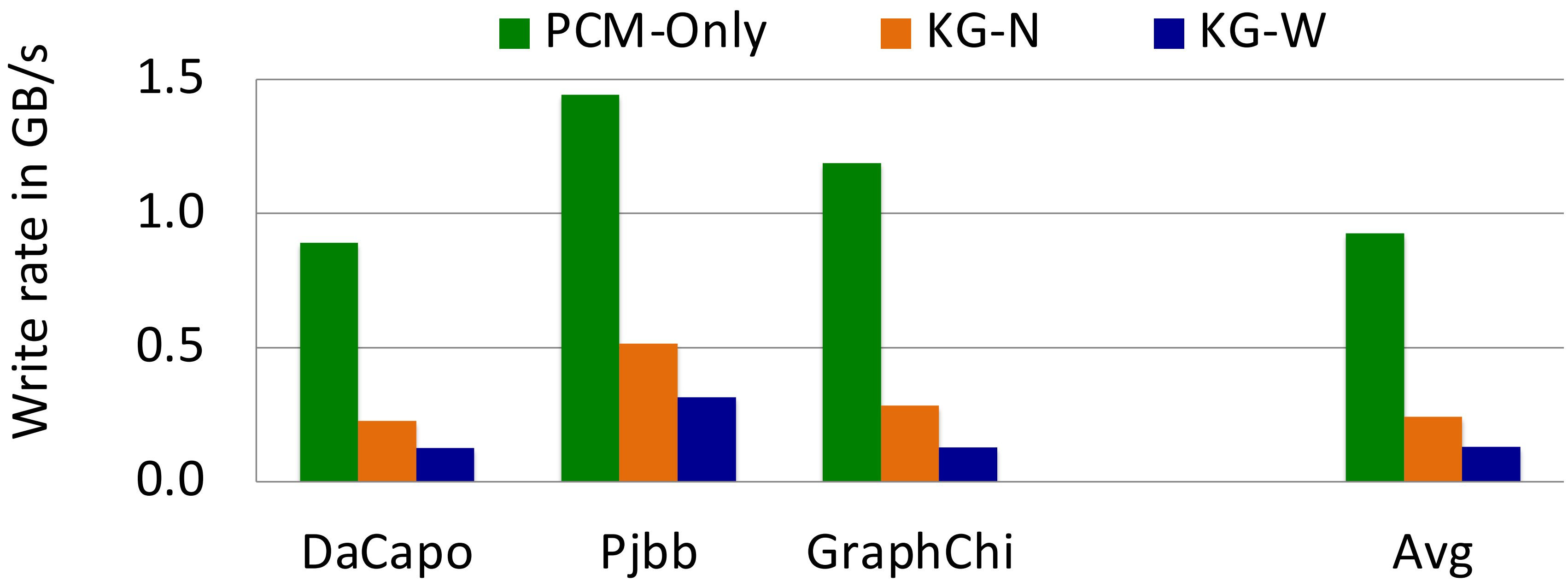
DRAM

DRAM

Modify

Use Intel perf monitor to measure writes

PCM write rates on NUMA hardware



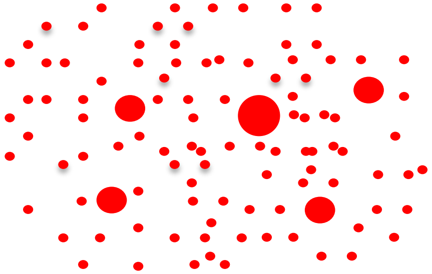
KG-N reduces write rate by 3.8X over PCM-Only

KG-W reduces write rate by 1.9X over KG-N

Crystal Gazer: Profile-Driven Write-Rationing Garbage Collection for Hybrid Memories

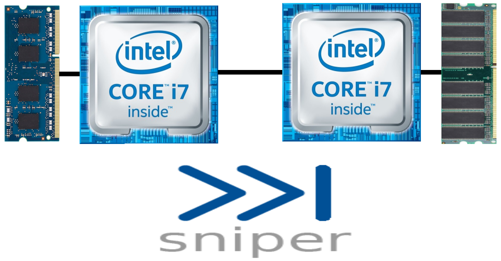
Takeaways

Monitoring heaps at a fine granularity is promising



Write-rationing garbage collection make PCM practical as main memory

Similar conclusion with 3 distinct methods



Exploiting Intel Optane Persistent Memory for Full Text Search

Shoaib Akram
ANU, Canberra
shoaib.akram@anu.edu.au

Australian National University

Full text search is ubiquitous

Web search

Google

Bing

Retail

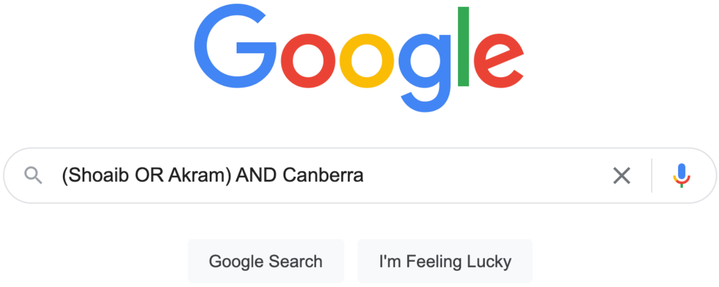
Social media

Search = Indexing + Query eval

Indexing builds an inverted index

word1 → document-list
word2 → document-list

Query evaluation searches for words

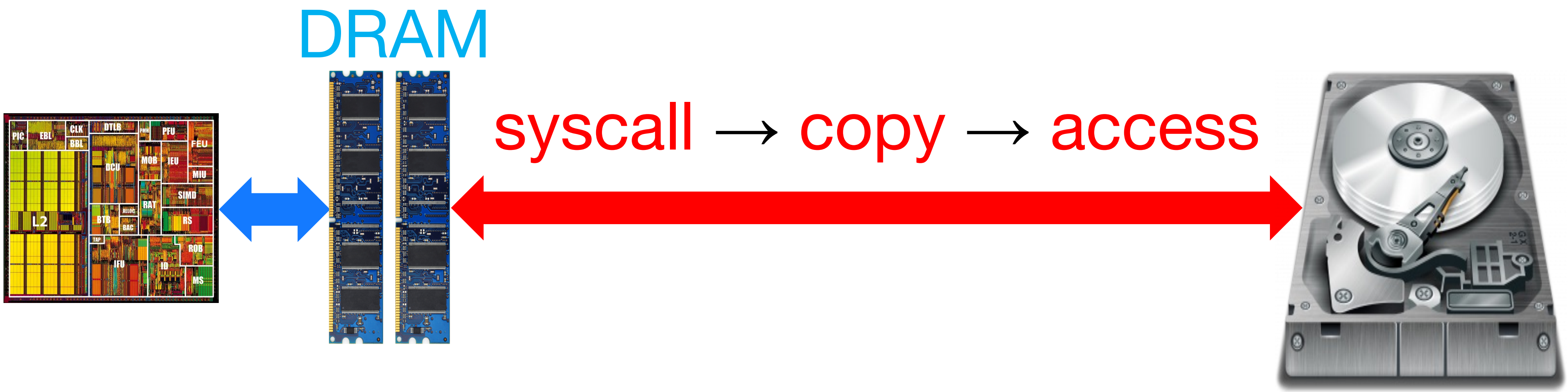


(Shoaib OR Akram) AND Canberra

Indexing speed increasingly **critical**

Challenge: I/O intensity

Writing & merging partial indices on storage takes up 40% of exec time

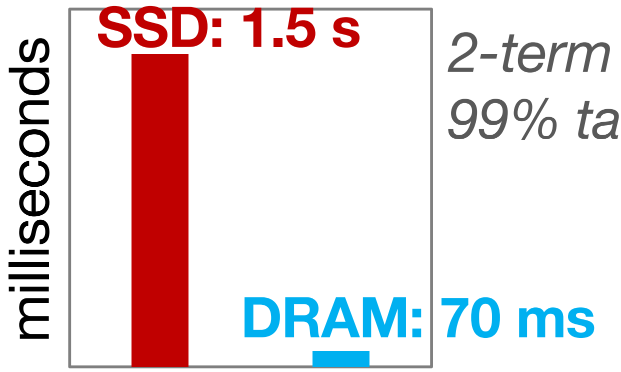


DRAM

syscall → copy → access

Challenge: DRAM capacity

NVMe SSD violates real time response constraint



2-term AND, 99% tail latency

🙁 Data growth outpaces DRAM scaling

Data volume → 2X

DRAM GB/$ → 20%

Today: Give up real time, or give up cost efficiency

Looking forward

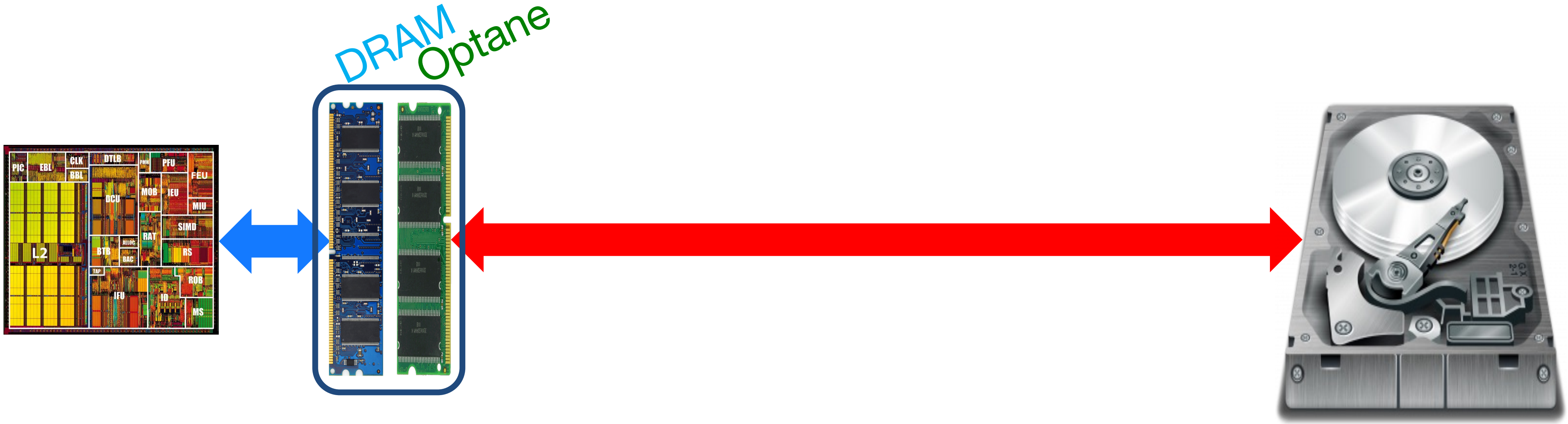
- Reduce I/O overhead
- Find a fresh memory scaling roadmap

Persistent memory (PM)

4X denser than DRAM

Load/store access

Non-volatile



Contribution: PM Search Engine

Exploiting PM for building/storing indices

→ Memory, storage, universal roles

→ Fine-grained crash consistent recovery

Extensive PM evaluation vs DRAM/SSD

→ Indexing perf, scalability, bottlenecks

→ Tail latency of query workloads

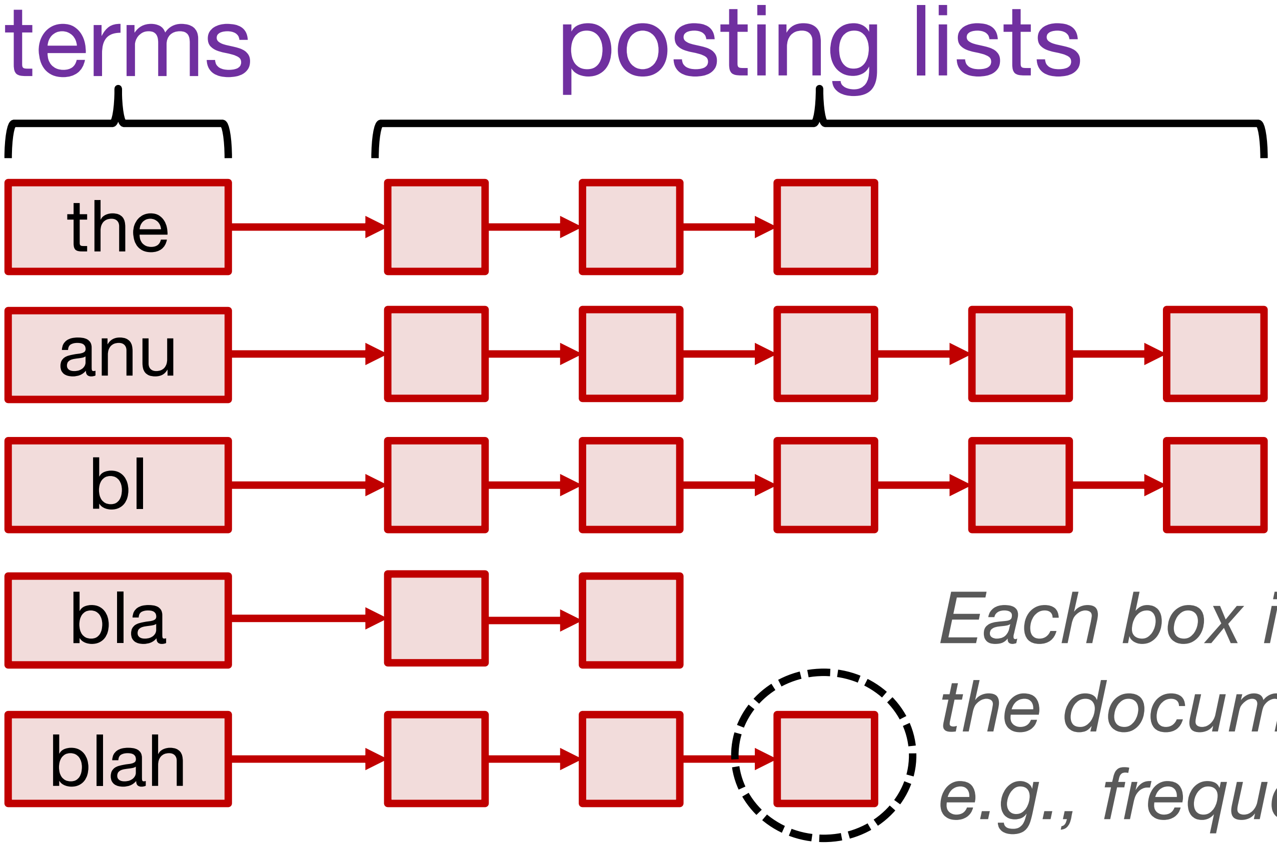
Rest of the talk

Building an index

Exploiting PM

Evaluation

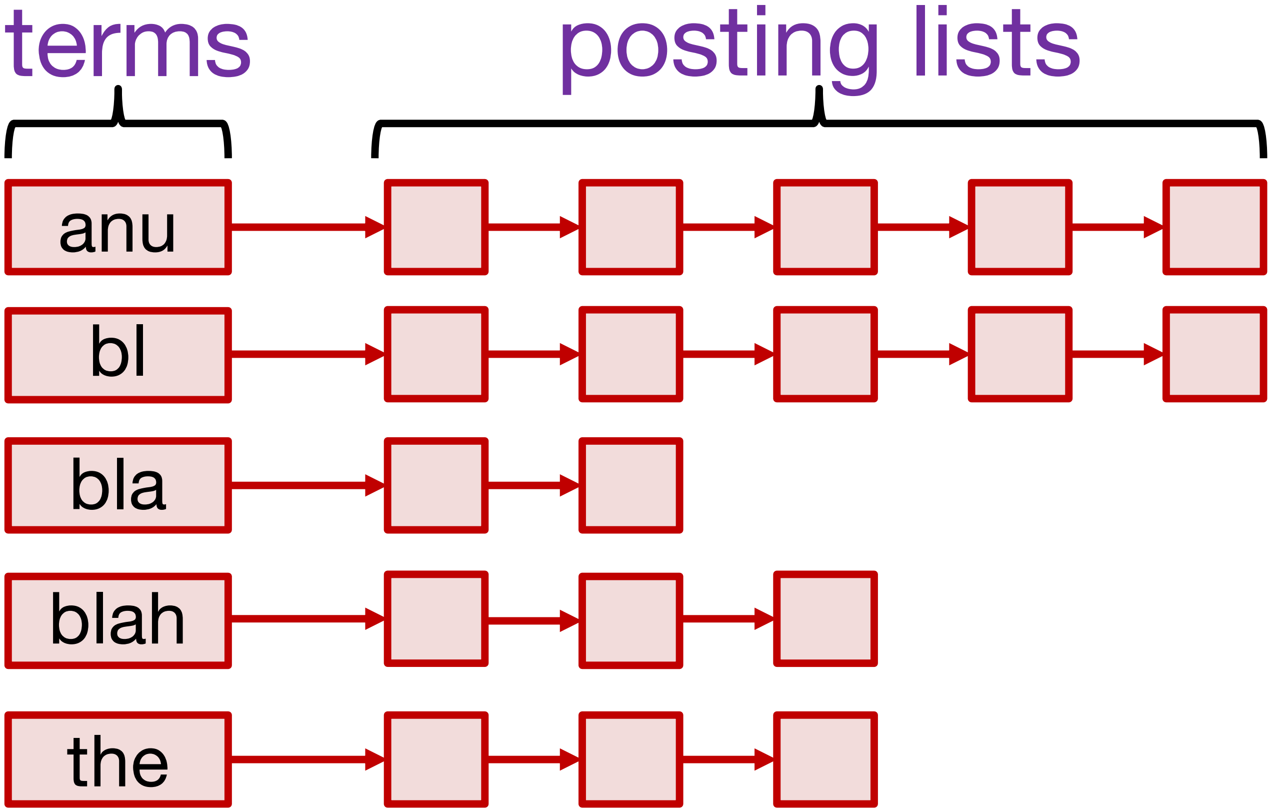
Step 1: Building the hash table



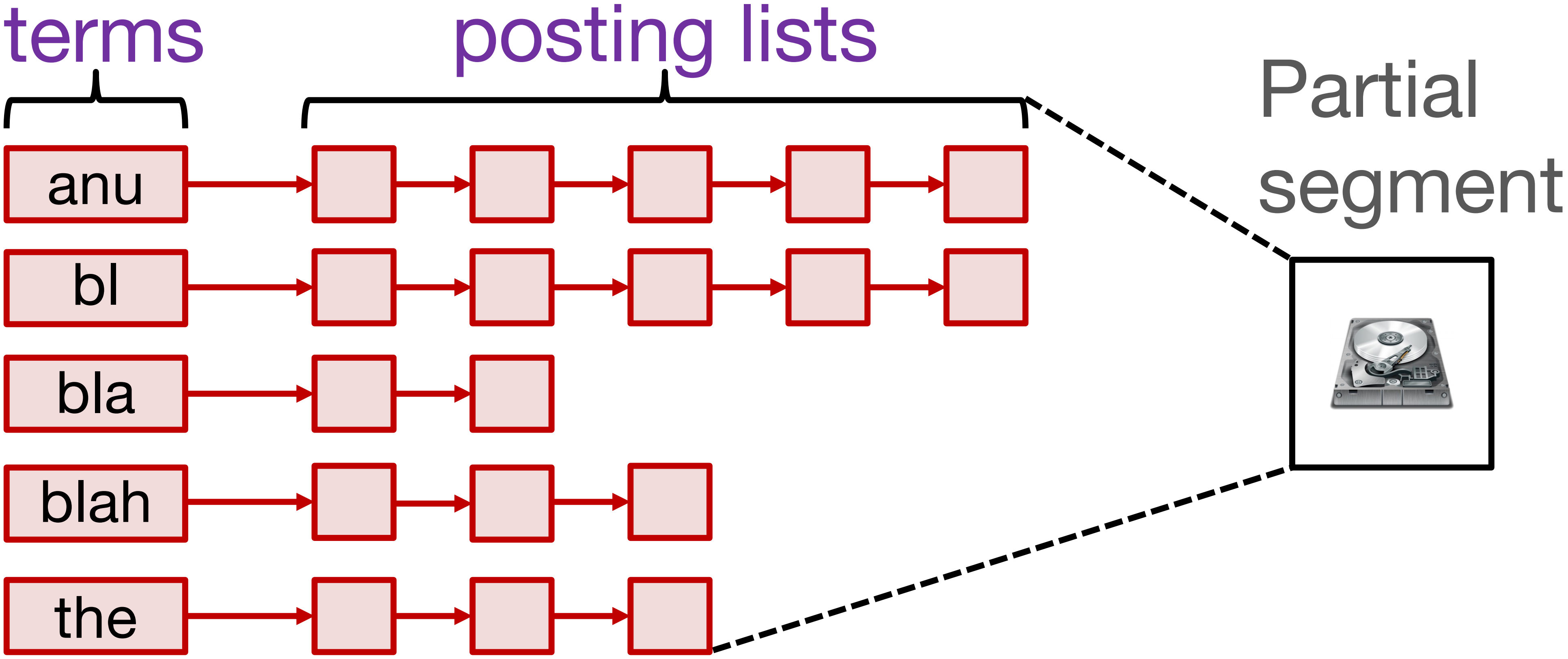
When the table is full → Step 2

Each box is a posting. It contains the document id plus meta-data, e.g., frequency and position of terms

Step 2: Sorting the hash table



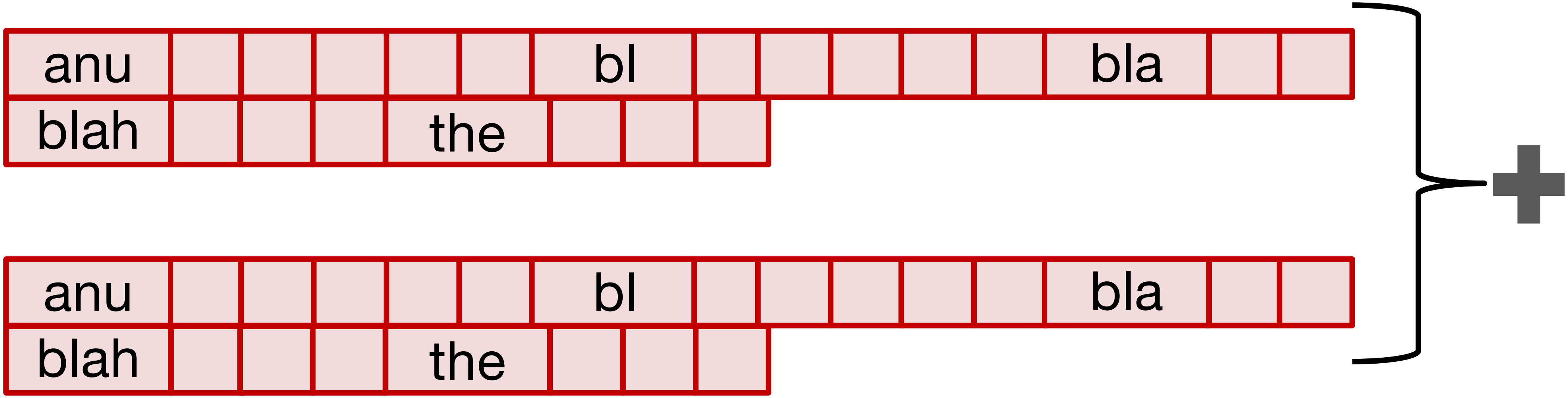
Step 3: Flushing the hash table



Flushing results in large amounts of sequentail I/O

Step 4: Merging segments

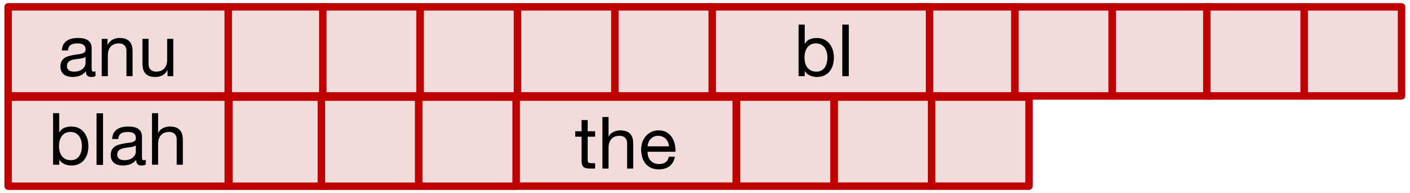
Merging segments is crucial for fast query evaluation



Merging results in large amounts of read/write I/O

Index = Segment + Dictionary

term	offset
anu	0
bl	6



Segment: Sequentially sorted postings on storage

Dictionary: To find posting lists in segments, indexers use a key-value store, such as, Berkeley DB

Different ways to exploit PM

Hash table, DRAM → PM

Partial segments, SSD → PM

Merged segments, SSD → PM

Dictionary, SSD → PM

PM configurations for indexing

Name of	Placement of Table, Postings, and Dictionary				Role of
Configuration	H Table	Partial St	Merged St	Dict	Optane PM
stock	DRAM	SSD	SSD	SSD	none
table-pm	PM	SSD	SSD	SSD	main memory
pm-only	PM	PM	PM	PM	universal
hybrid	DRAM	PM	PM	PM	storage
hybrid+	DRAM	PM	PM	SSD	storage

PM configurations for indexing

Name of	Placement of Table, Postings, and Dictionary				Role of
Configuration	H Table	Partial St	Merged St	Dict	Optane PM
stock	DRAM	SSD	SSD	SSD	none
table-pm	PM	SSD	SSD	SSD	main memory
pm-only	PM	PM	PM	PM	universal
hybrid	DRAM	PM	PM	PM	storage
hybrid+	DRAM	PM	PM	SSD	storage

PM configurations for indexing

Name of Configuration	Placement of Table, Postings, and Dictionary				Role of Optane PM
	H Table	Partial St	Merged St	Dict	
stock	DRAM	SSD	SSD	SSD	none
table-pm	PM	SSD	SSD	SSD	main memory
pm-only	PM	PM	PM	PM	universal
hybrid	DRAM	PM	PM	PM	storage
hybrid+	DRAM	PM	PM	SSD	storage

Crash consistent indexing

Crash consistent segment flushing

→ Use pmem_persist(segment)

→ Track progress (docIds)

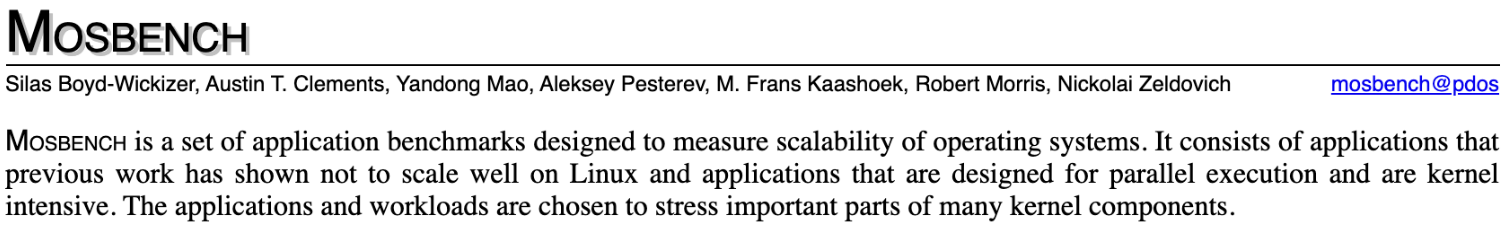
Crash consistent merging

→ Tracking progress is tricky

→ Details of “logging” in the paper

Baseline Engine

Psearchy



MOSBENCH

Silas Boyd-Wickizer, Austin T. Clements, Yandong Mao, Aleksey Pesterev, M. Frans Kaashoek, Robert Morris, Nickolai Zeldovich mosbench@pdos

MOSBENCH is a set of application benchmarks designed to measure scalability of operating systems. It consists of applications that previous work has shown not to scale well on Linux and applications that are designed for parallel execution and are kernel intensive. The applications and workloads are chosen to stress important parts of many kernel components.

Native, fast, and flexible

Easily integrated with Intel PMDK

Indexing Methodology

Dataset and measurement

→ Wikipedia English (DRAM)

→ Execution time

→ 1 GB HT per core, up to 32 cores

PM setup

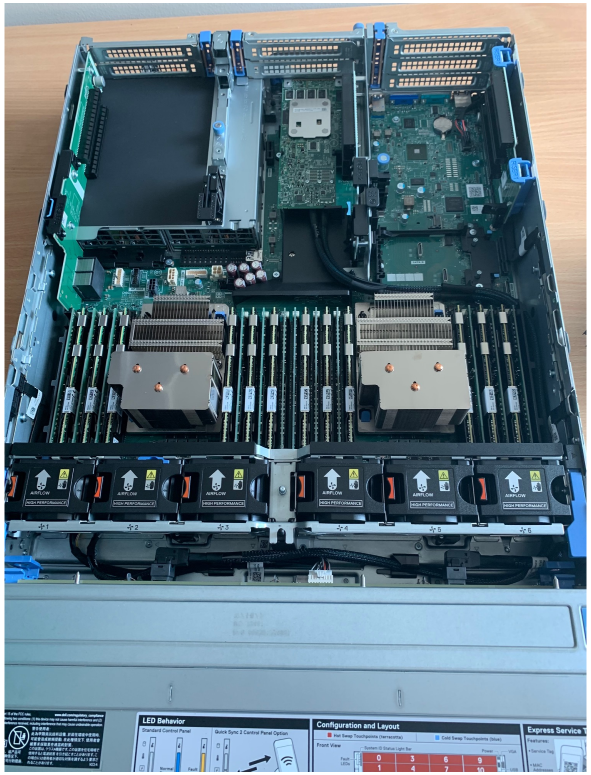
→ Interleaved, local, EXT4+**DAX**

→ pmemkv dictionary github.com/pmem/pmemkv

Experimental Platform

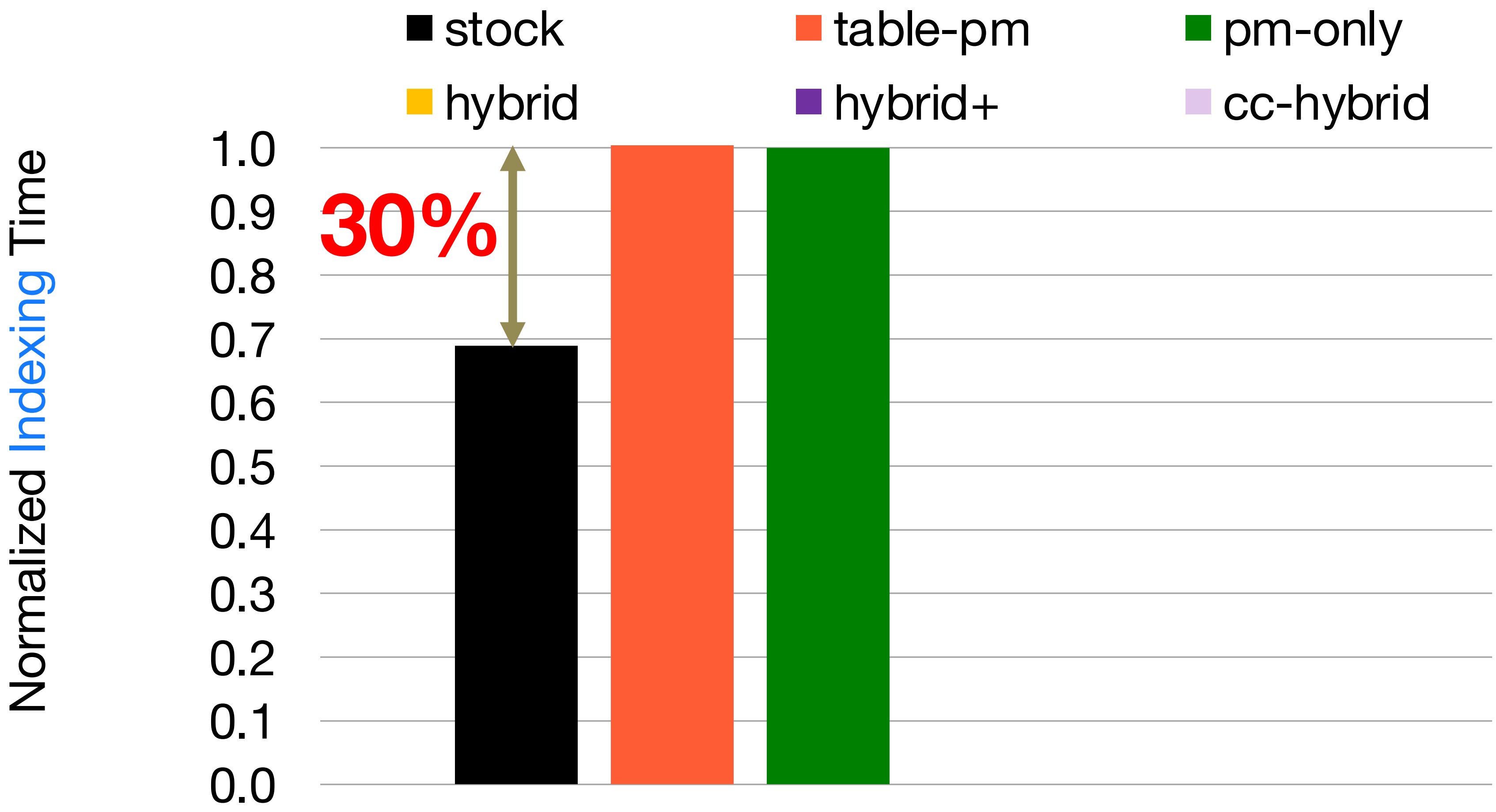
Our in-house server with DRAM, PM, & SSD

2 TB PM
0.5 TB DRAM
1.5 TB NVMe Optane SSD

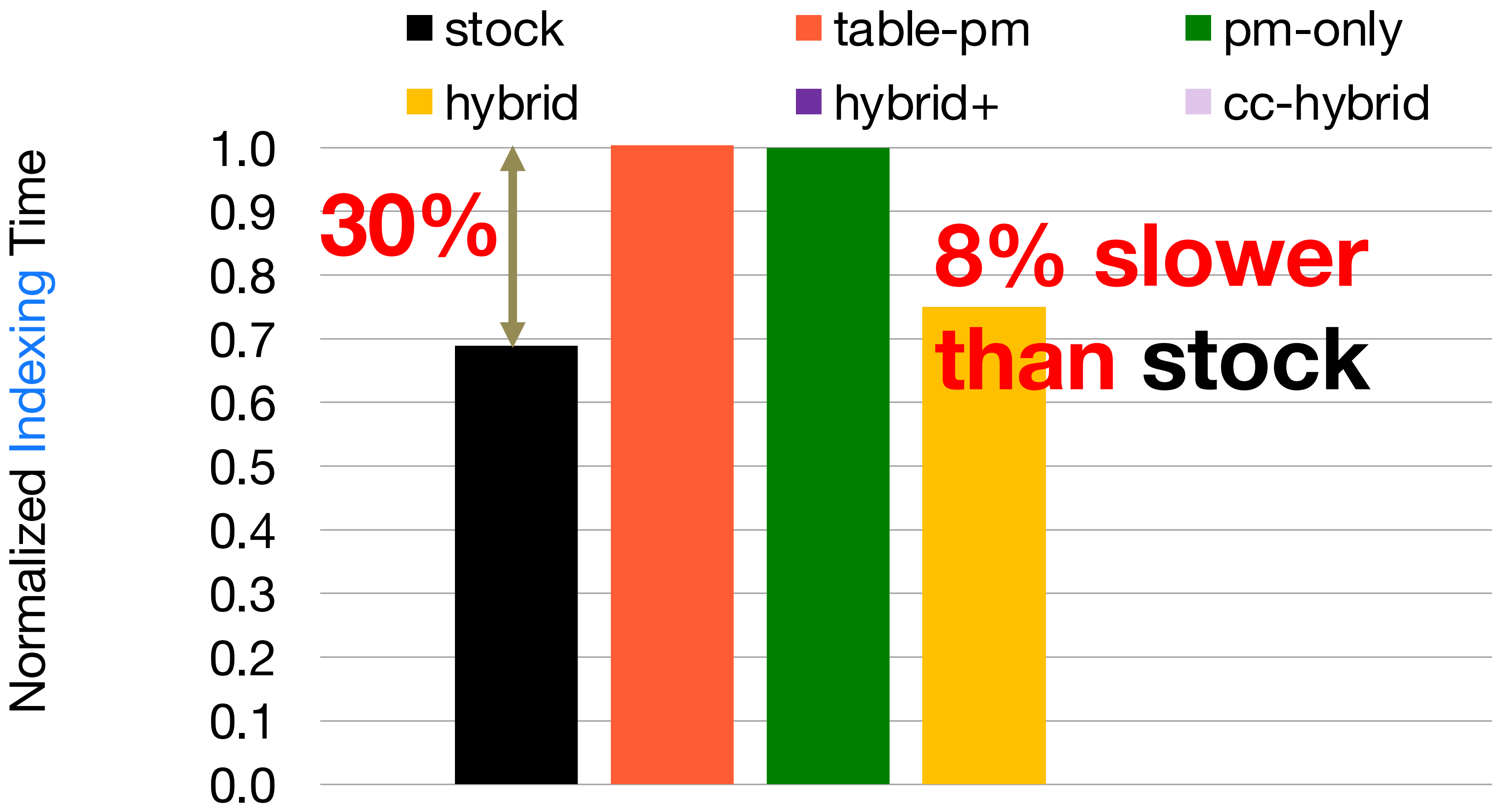


Indexing perf with one core

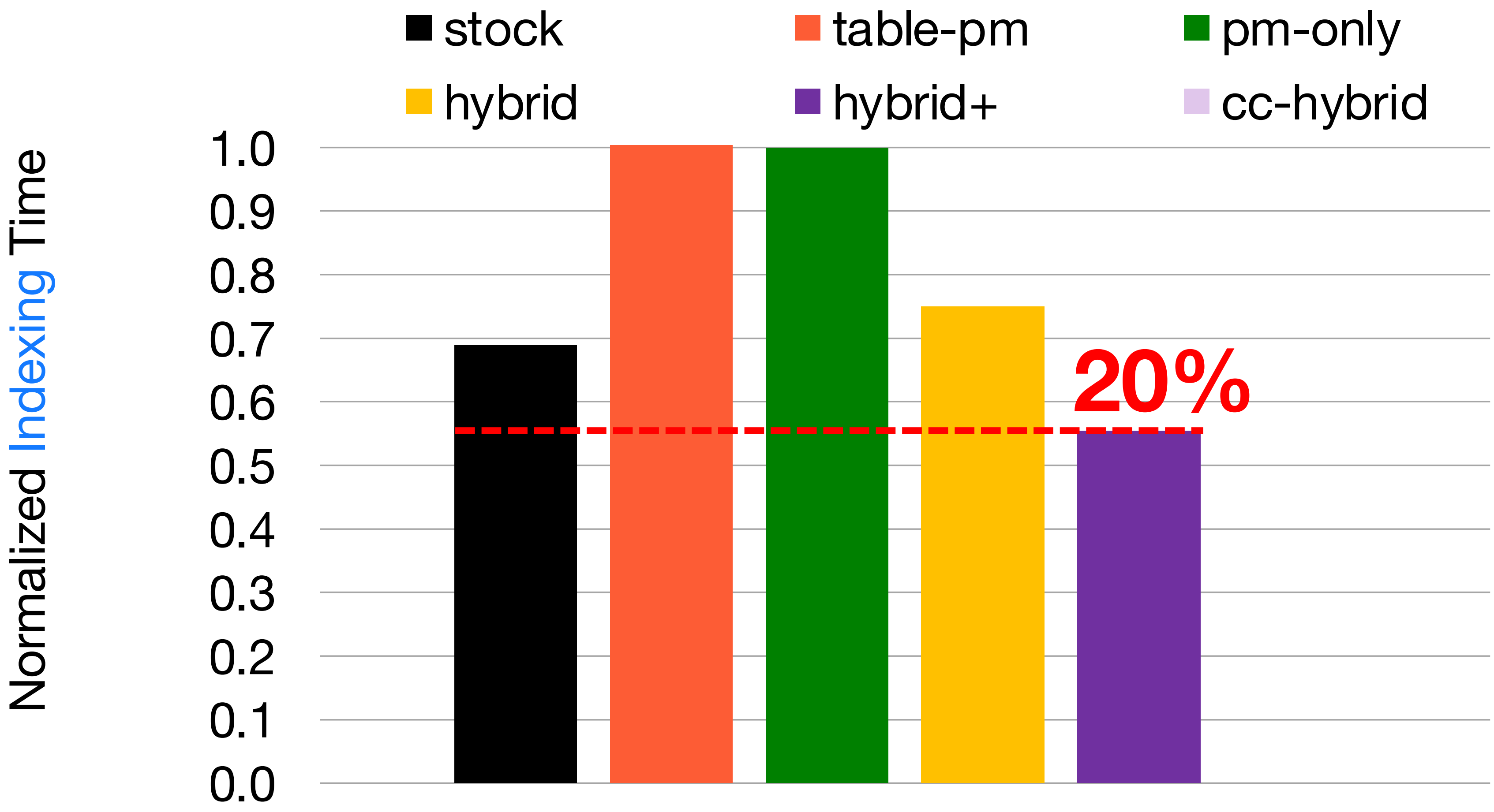
PM as main/only is 30% slower



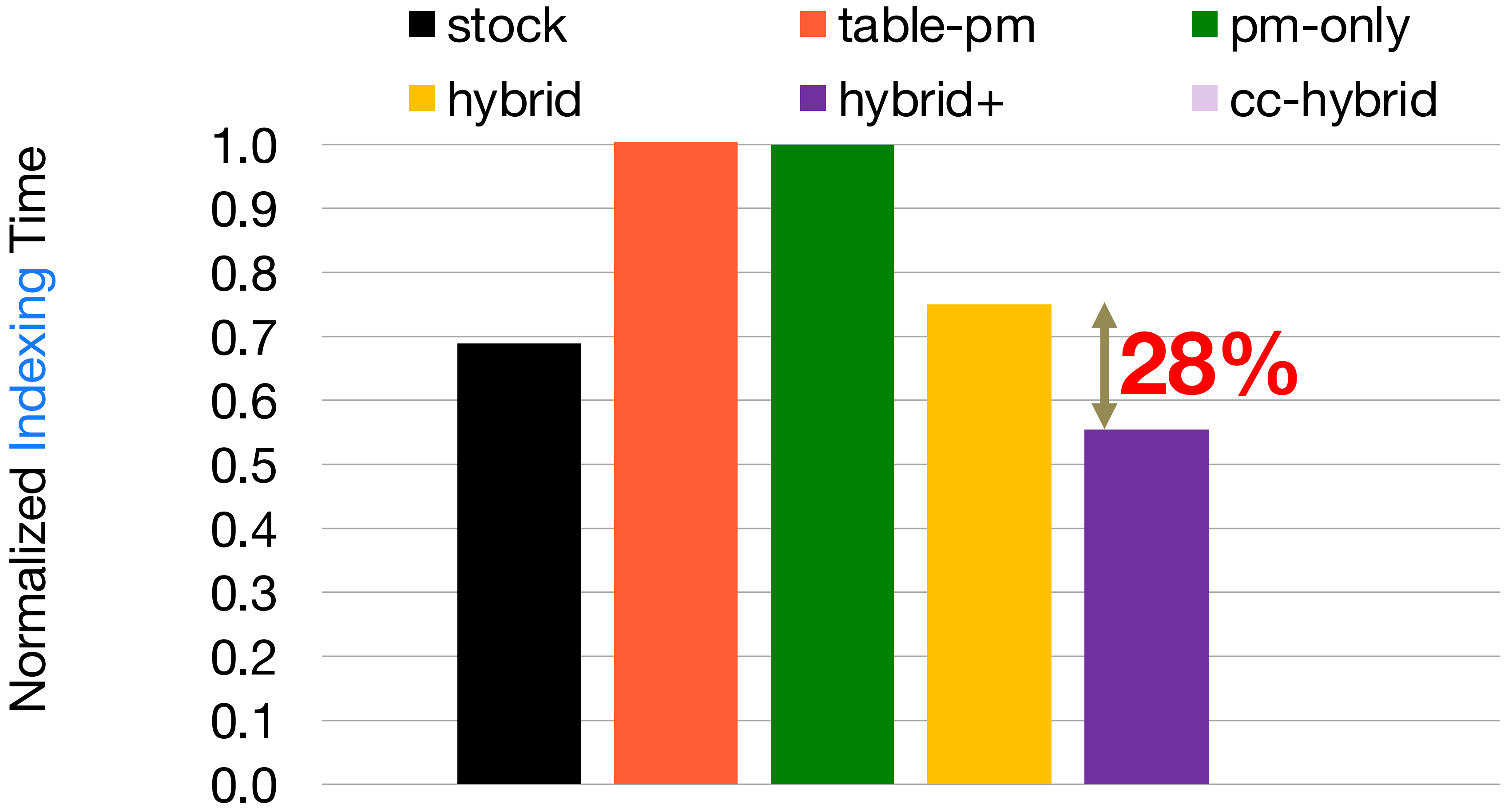
Hybrid is 8% slower than stock



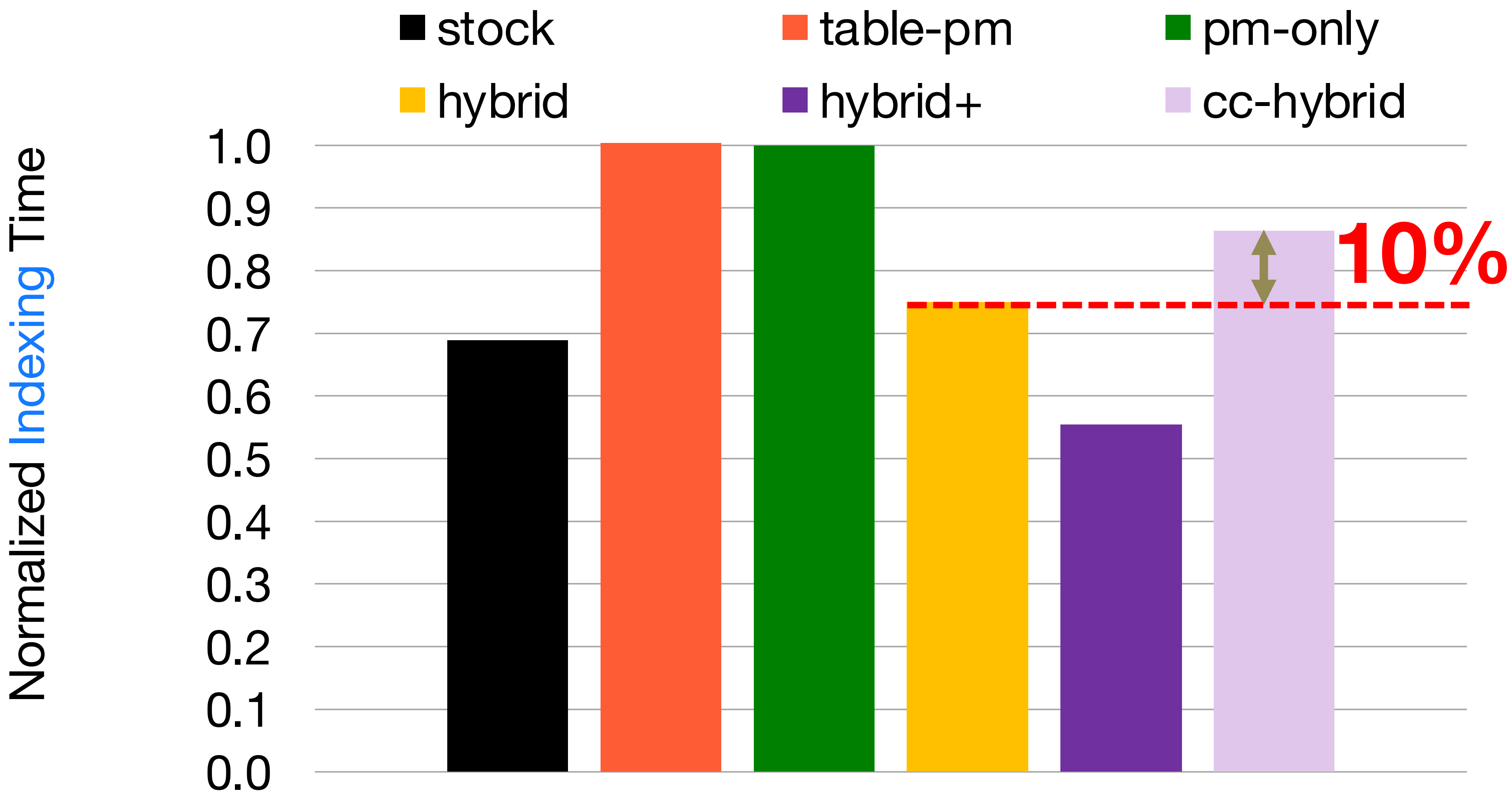
Hybrid+ is best, 20% over stock



Hybrid+ is best, pmkv costs 28%

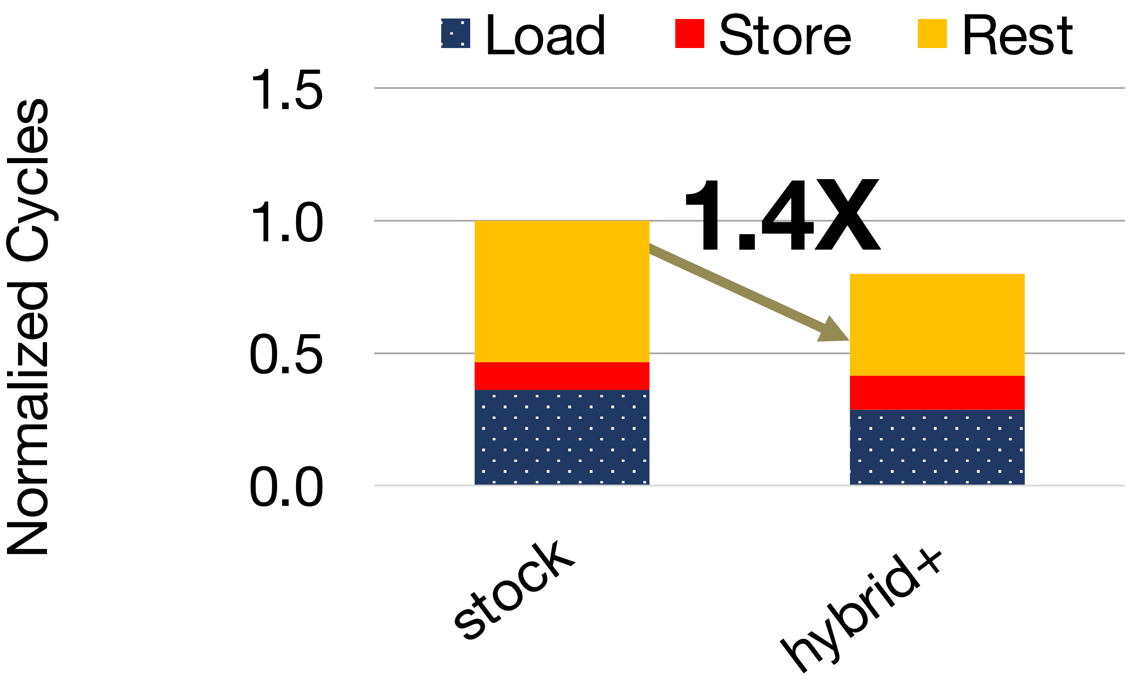


Crash consistency costs 10%

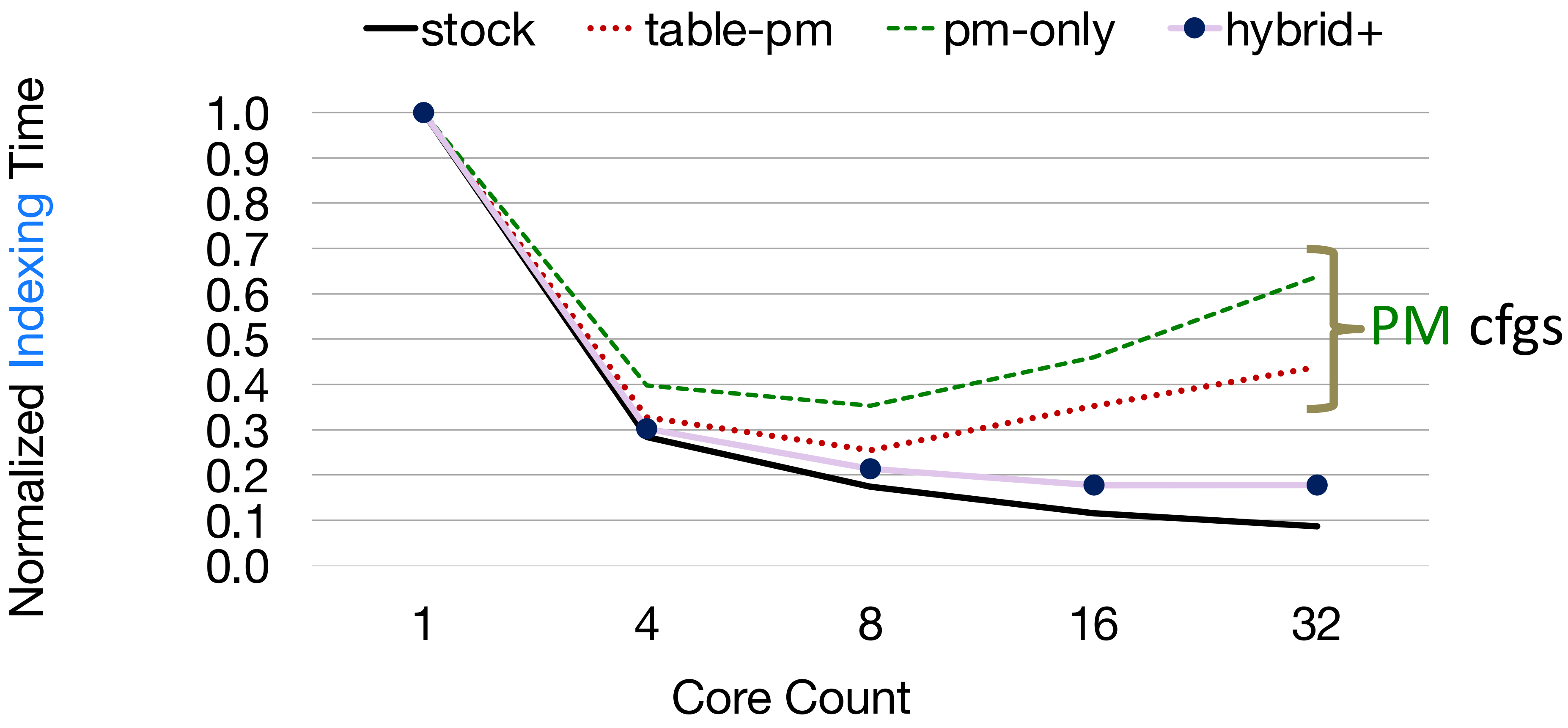


syscall → mmap is mainly why hybrid+ beats stock

Use perf counters to observe Load/Store stalls the multicore incurs

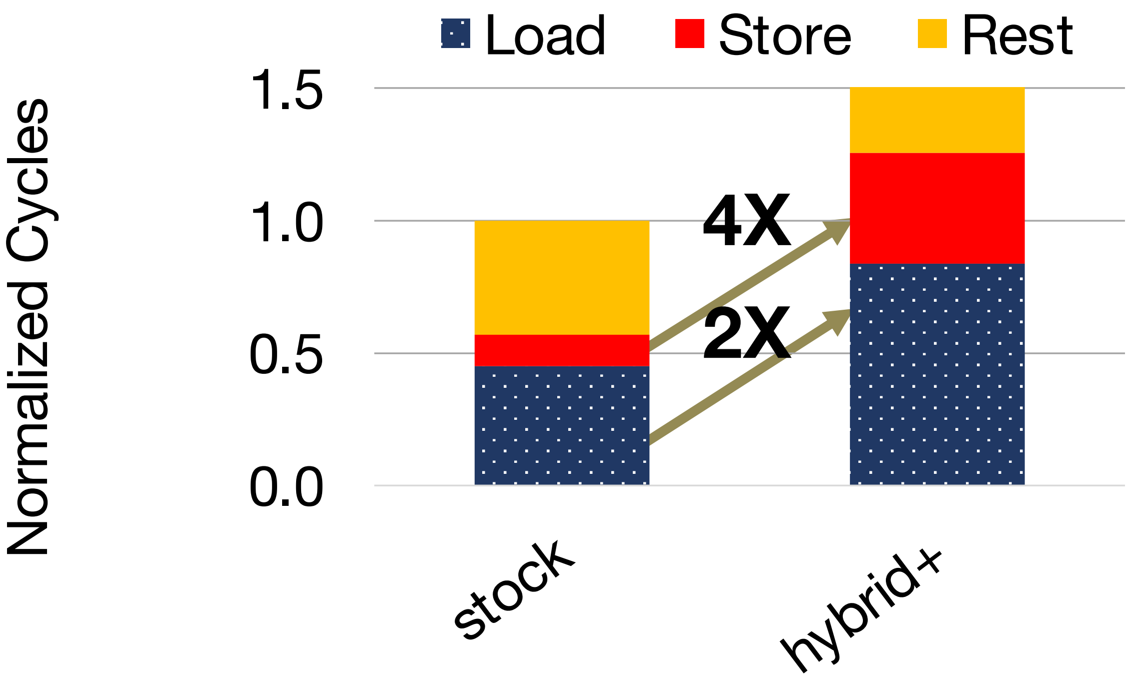


Indexing scalability



Hybrid+ incurs an increase in memory stalls (32 cores)

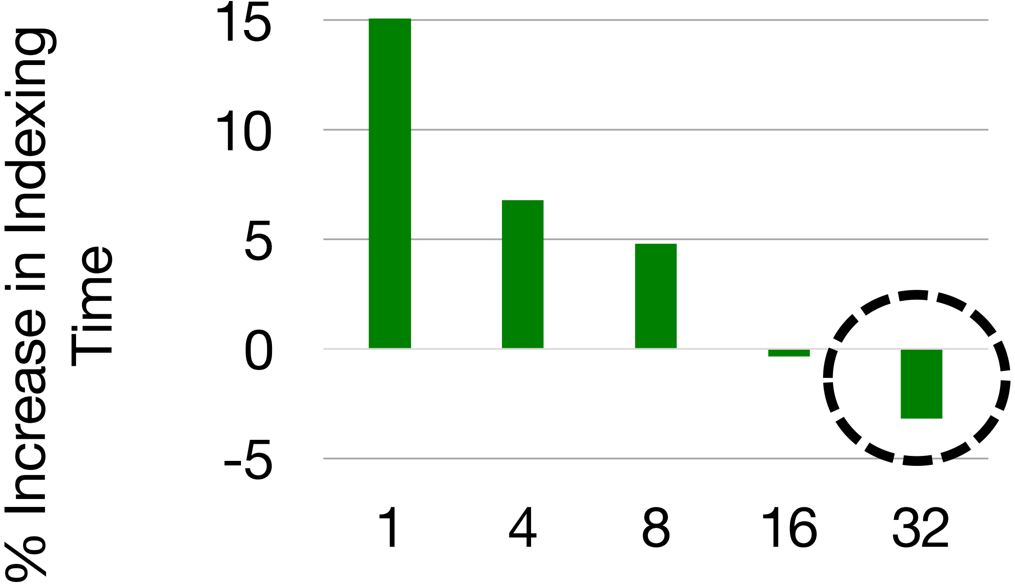
Use perf counters to observe Load/Store stalls the multicore incurs



Crash consistent indexing with 32 cores improves perf

32 cores: Invalidated cache lines become replacement candidates, improving LLC hit rate

Baseline: No pmem_persist()



Core Count

Query Evaluation Methodology

Tail latency of 100K concurrent queries

→ 1 term

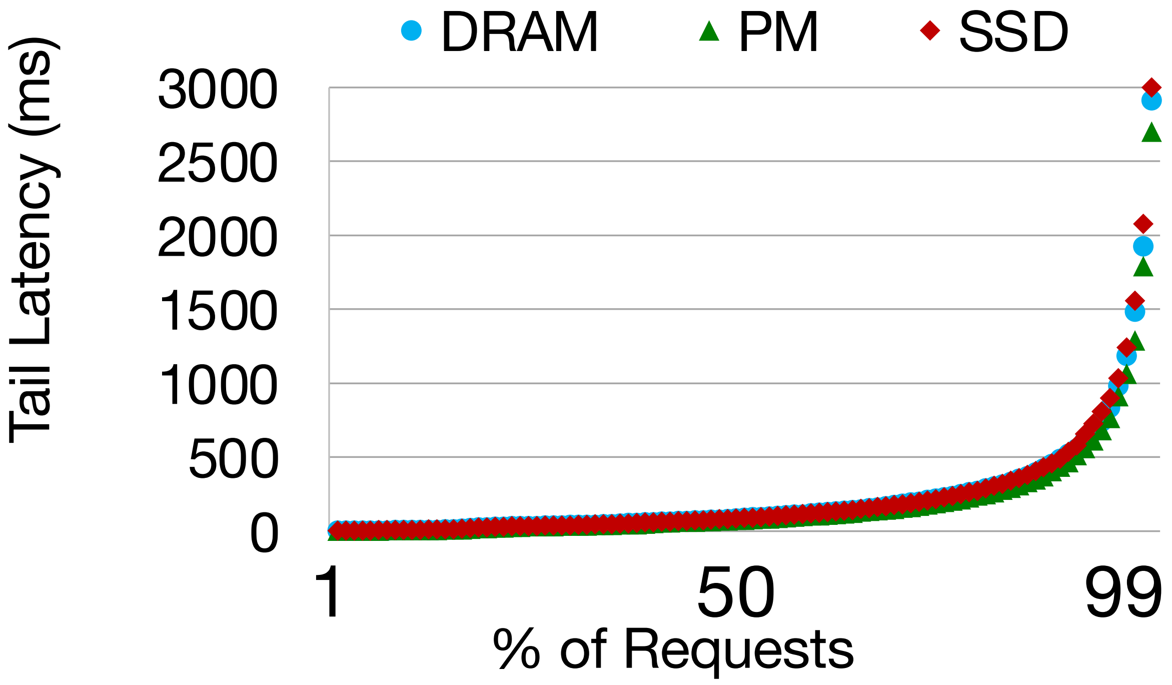
→ AND 2 terms

See paper for details

→ Term selection, variation, ranking

Tail latency of single-term queries
DRAM = PM = SSD

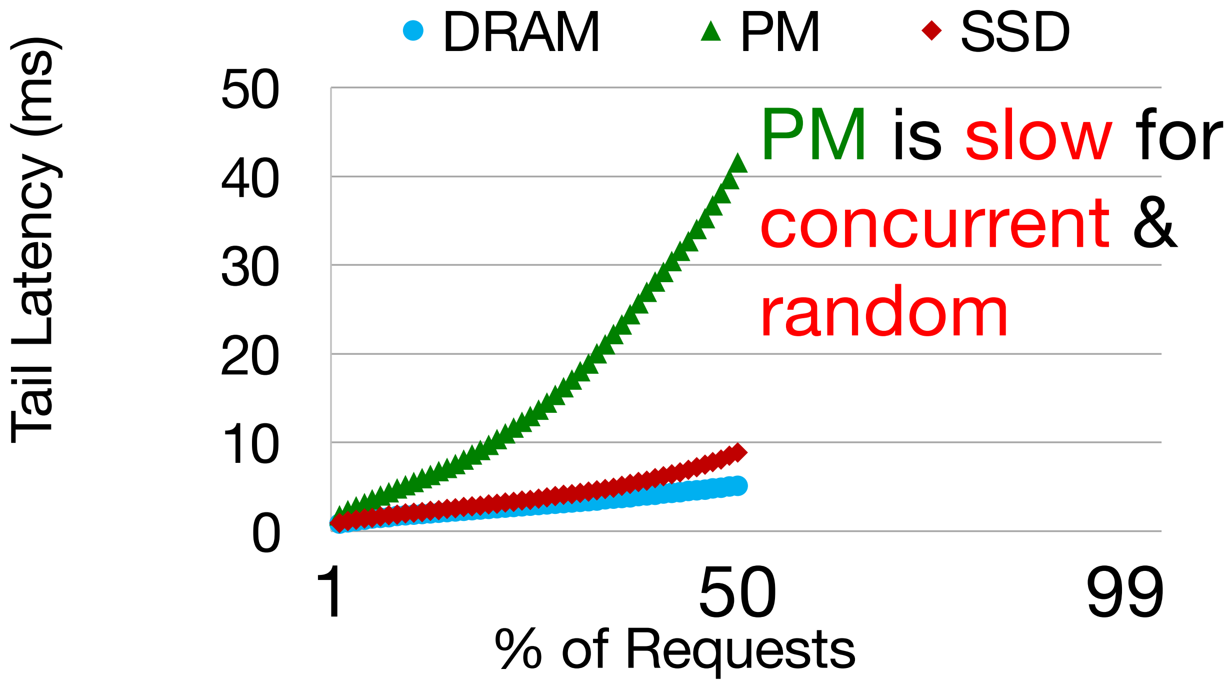
Accessing a single posting list results in a sequential access pattern



Tail latency of 2-term AND
Region 1: DRAM < SSD < PM

50% Shortest queries

Advancing two lists leads to random accesses

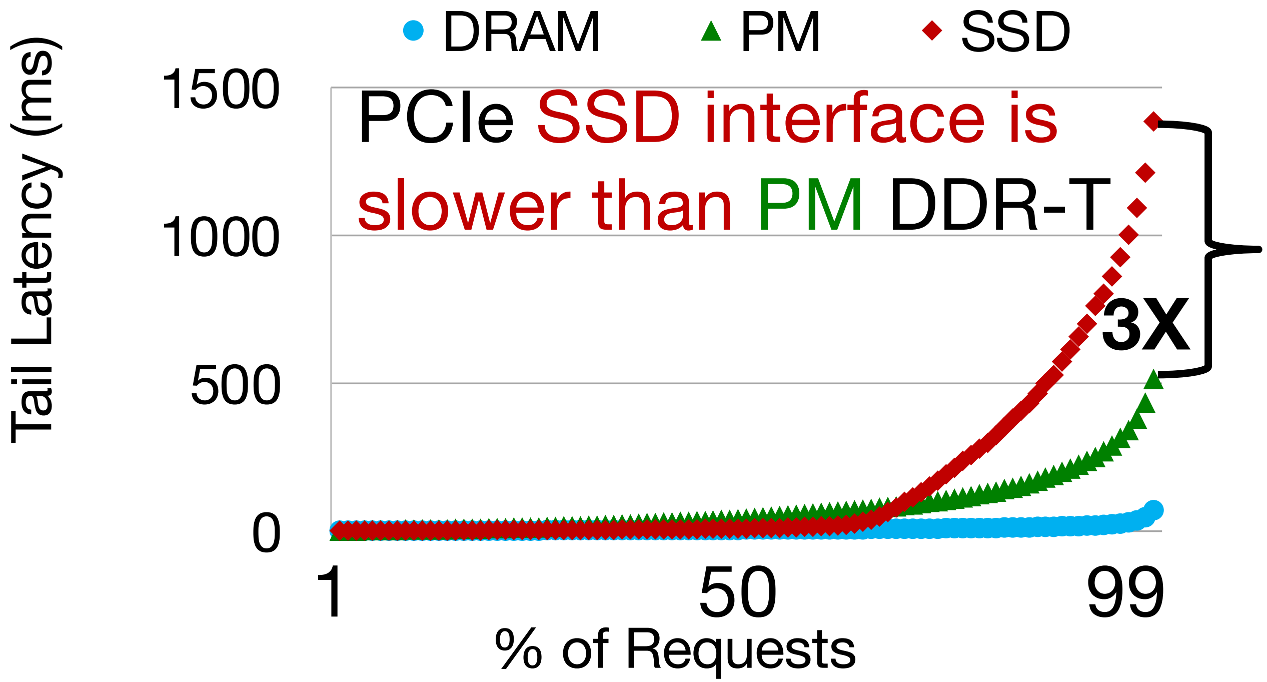


PM is slow for concurrent & random

Tail latency of 2-term AND
Region 2: DRAM < PM < SSD

50% Longest queries

These queries access the SSD media



PCIe SSD interface is slower than PM DDR-T

More analysis in the paper

Indexing: updates

Query eval: access patterns

Breakdowns: sort vs merge, load vs store

pmemkv: volatile map, binding

Other: OS caching impacts

Key Takeaways

PM does not scale well for write I/O bound indexing

PM shines for the latency-critical query evaluation

Contribution: PM Search Engine

Exploiting PM for building/storing indices

→ Memory, storage, universal roles

→ Fine-grained crash consistent recovery

Extensive PM evaluation vs DRAM/SSD

→ Indexing perf, scalability, bottlenecks

→ Tail latency of query workloads

UNIVERSITY OF CRETE Australian National University ISOVALENT Red Hat FORTH INSTITUTE OF COMPUTER SCIENCE

TeraHeap: Reducing Memory Pressure in Managed Big Data Frameworks

Iacovos G. Kolokasis
kolokasis@ics.forth.gr

Giannos Evdorou
evdorou@ics.forth.gr

Shoaib Akram
shoaib.akram@anu.edu.au

Christos Kozanitis
kozanitis@ics.forth.gr

Anastasios Papagiannis
anastasios@isovalent.com

Foivos Zakkak
fzakkak@redhat.com

Polyvios Pratikakis
polyvios@ics.forth.gr

Angelos Bilas
bilas@ics.forth.g
r

Analytics frameworks need large heaps

Analytics frameworks use managed runtimes

To process **large amounts of data** they need **large heaps**

Large heaps are **expensive (DRAM)** and **increase GC cost!**

- DRAM is expensive in dollar cost, energy, and power
- GC requires expensive scans over large heaps

For these reasons analytics frameworks avoid large heaps

Common practice: Move objects off-heap

Off-heap storage in this context means

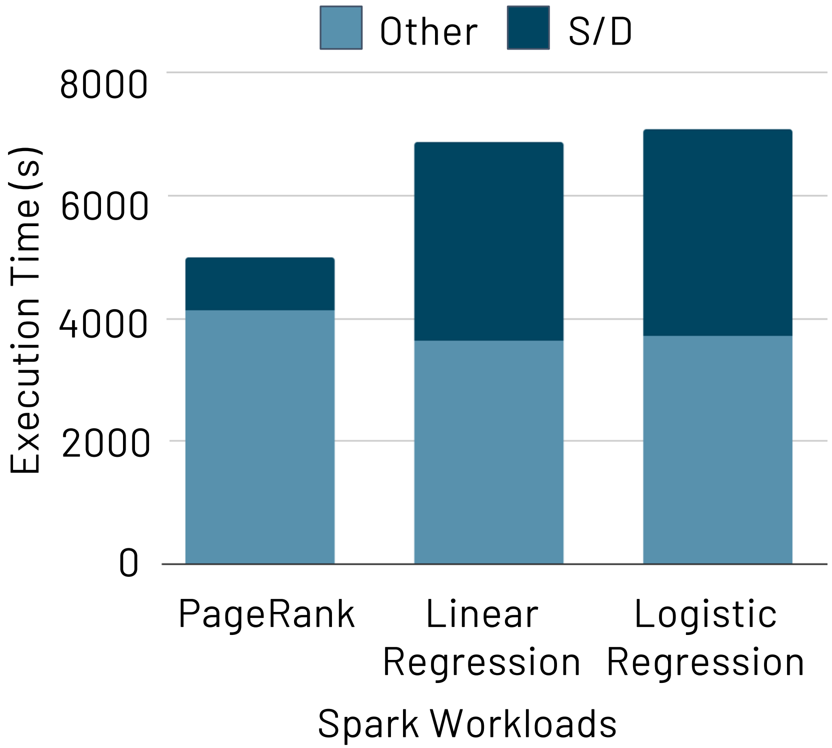
- Off DRAM → on fast storage
- Unmanaged → no GC scans

Off-heap demands serialization/deserialization (S/D)

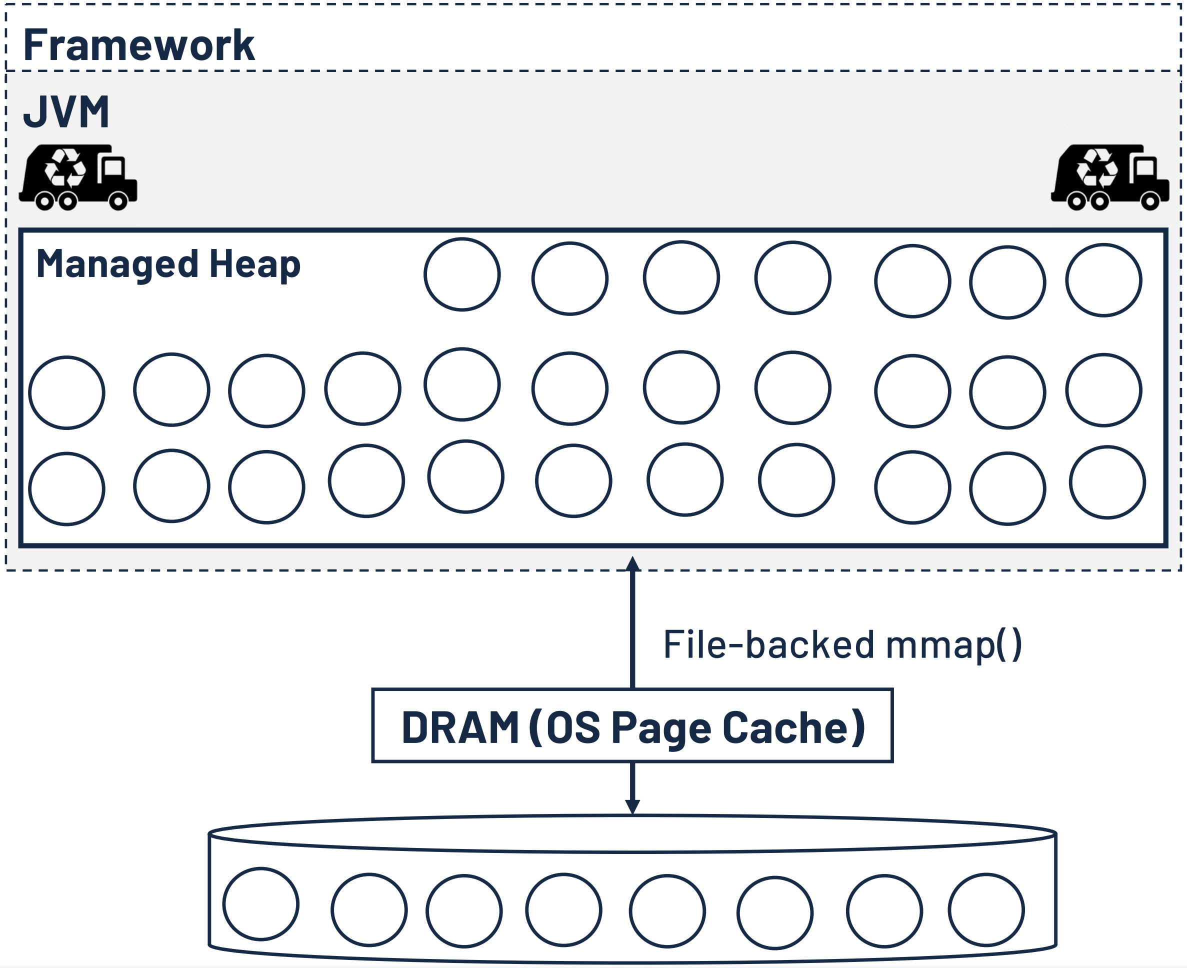
- Transform object closure into byte streams

S/D is significant problem!

- Takes up to 47% in Spark workloads
- Not everything is serializable!
- Off-heap can be unsafe



Eliminate S/D: Extend the heap over storage

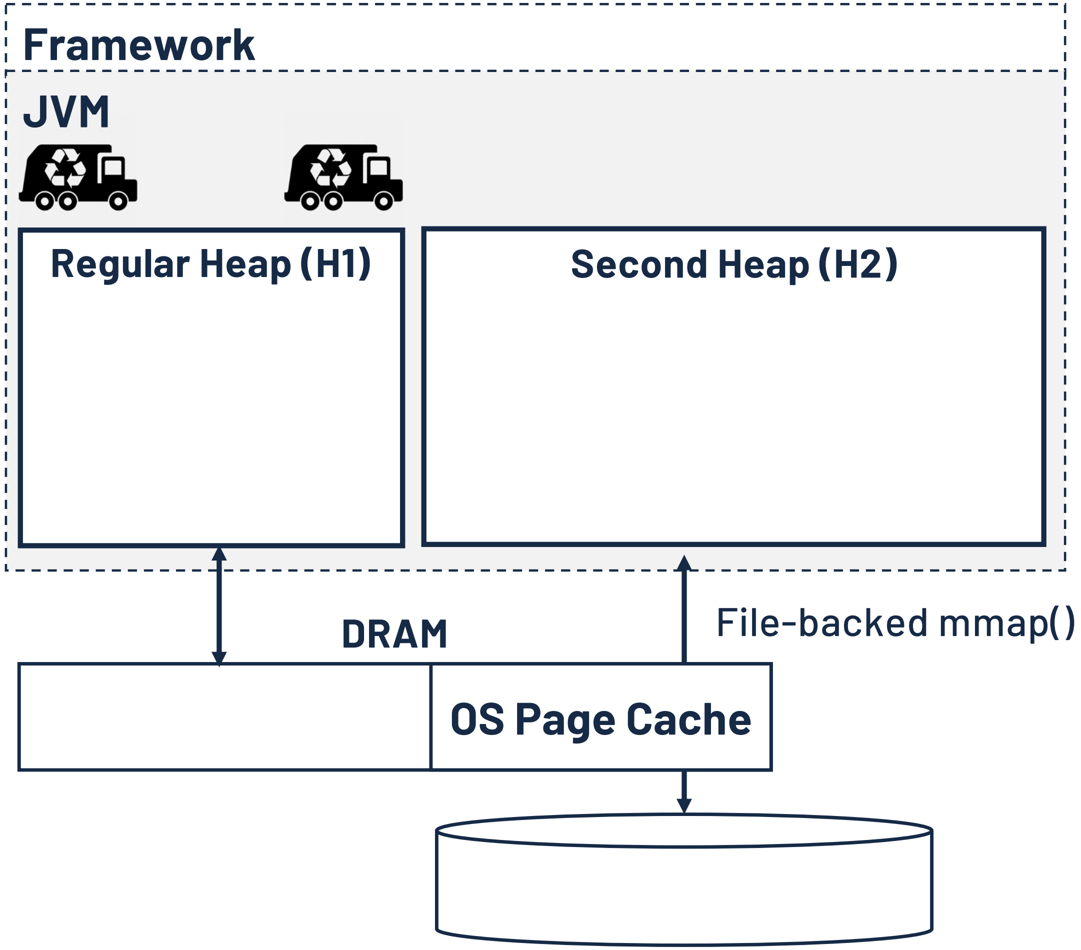


Today OpenJDK naively uses mmap()

GC cost increases dramatically!

- **Random accesses** over storage
- **Object compaction** over storage
- High I/O traffic

TeraHeap: Eliminate S/D without increasing GC cost



Provides the illusion of a single heap

Avoid GC scans over the device heap

Custom management for the device heap

- Lazy GC due to high storage capacity
- Minimizing I/O traffic

Outline

Motivation

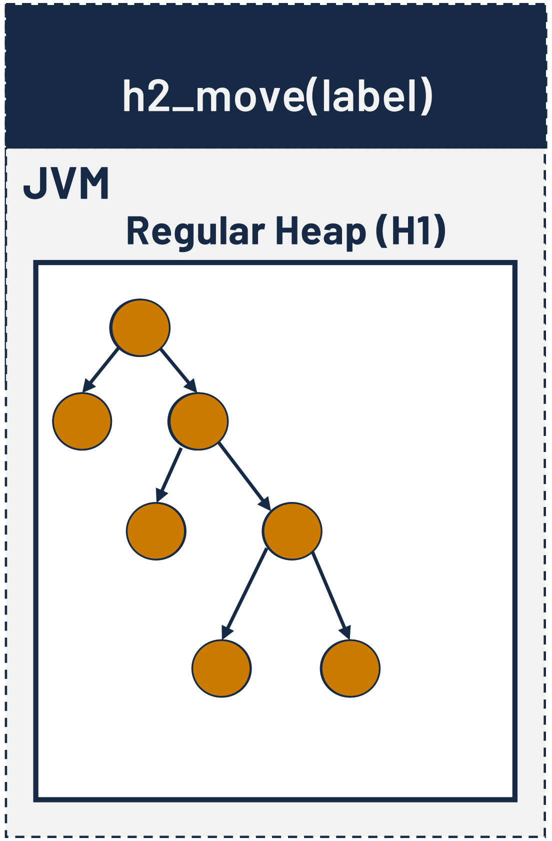
Design

- Identify objects for moving to H2
- Reclaim objects in H2 without GC scans
- Update cross-heap references with low I/O cost

Evaluation

Conclusions

Move off-heap objects to H2



☑ Goal: Find **large clusters** of objects with **similar lifetime**

Frameworks **move** partitions **off-heap**

Frameworks have **eventually immutable objects**

TeraHeap provides two hints

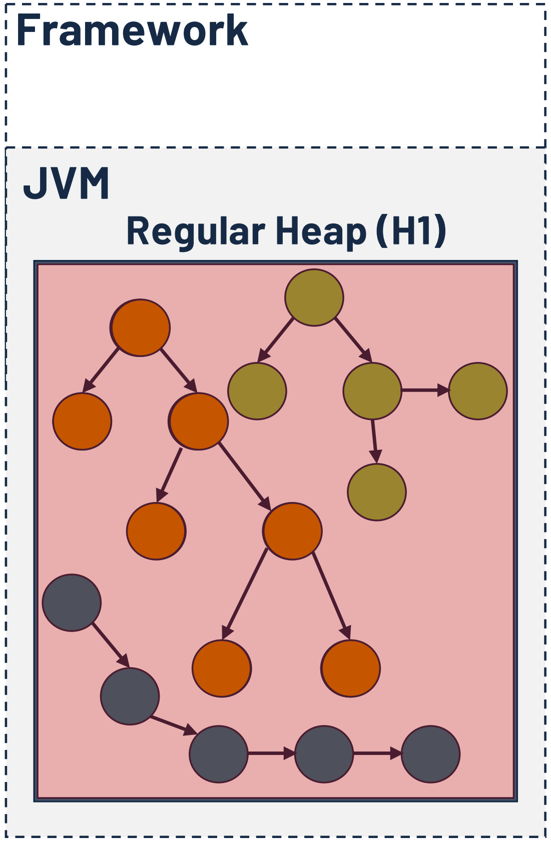
h2_mark_root(): Mark key object with a label

h2_move(): Advice when to move objects to H2

Move objects to H2 during GC

GC propagates the label from key object to all reachable

Can move objects to H2 eagerly



☑ Goal: Reduce **memory pressure in H1**

Increased memory pressure before transfer hint?

Eager transfers to H2 ⟶ decrease memory pressure in H1

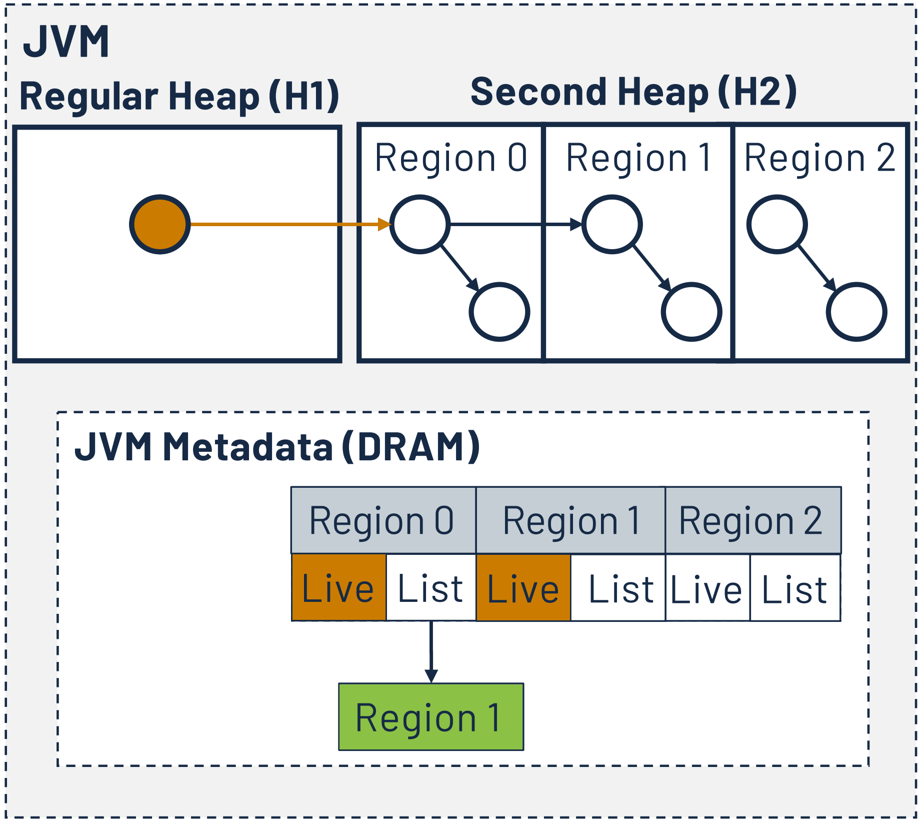
Use a high threshold to identify memory pressure

Bypass transfer hint

Move only a few marked objects to H2

Reduce **read-modify-write operations** in storage

Leverage storage capacity to free objects lazily



☑ Goal: **Reclaim** dead objects **without GC scans**

TeraHeap organizes H2 in fixed-sized regions

Objects with same label in the same region

Reclaim whole regions **(bulk free)**

Per region DRAM metadata **(avoid object access)**

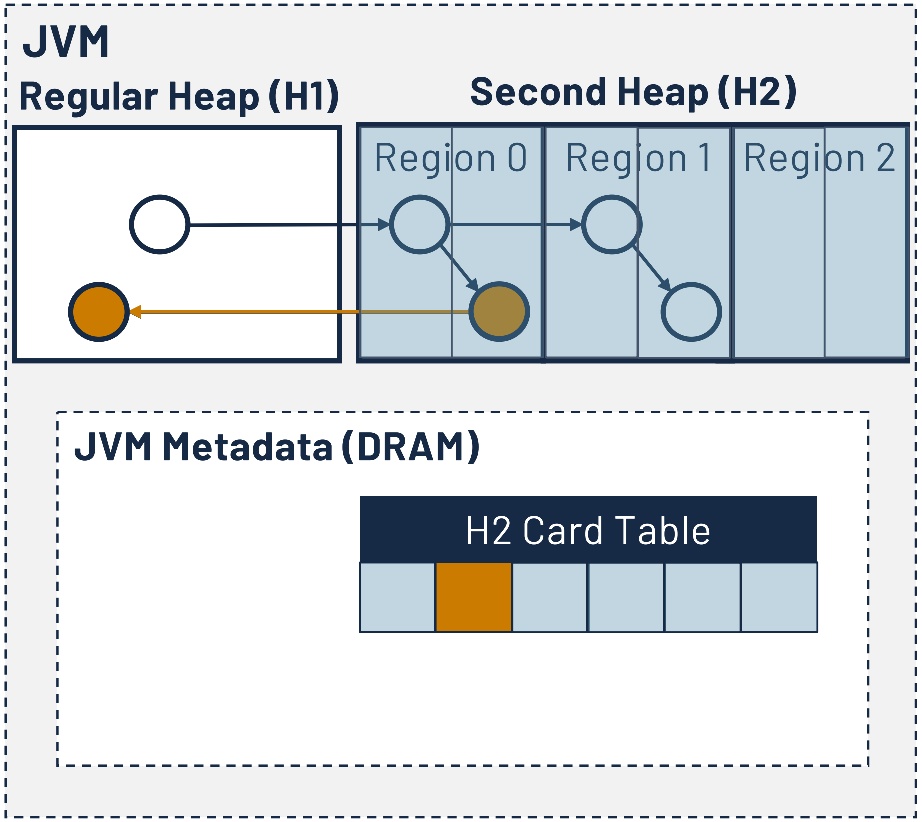
Live bit → region liveness

Dependency list → cross-region references

GC identifies H2 live regions

Free regions by zeroing regions metadata

Preserve correctness of object liveness



☑ Goal: **Track** H2 to H1 references with **low I/O cost**

Card table (byte array in DRAM)

One byte per fixed-size H2 segments

Large segments to reduce card table size

Categorize cards to scan less segments

Based on GC type, we scan specific segments

Testbed

We implement TeraHeap in OpenJDK 8 (we now support OpenJDK 17)

- Extend Parallel Scavenge garbage collector
- Extend interpreter, C1 and C2 (JIT) compilers to support updates in H2

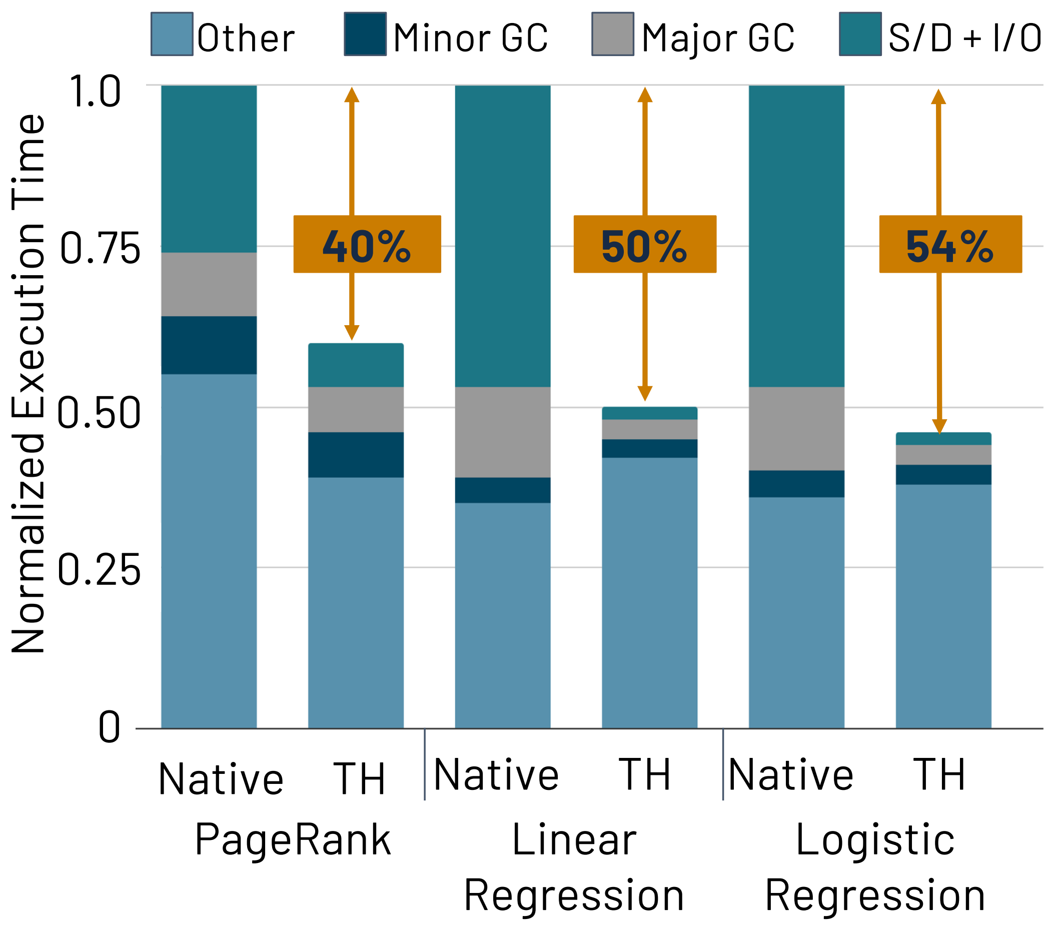
We use one servers with 2 TB NVMe SSD and 256 GB DRAM

- Also, we evaluate TeraHeap with NVM

- Real world applications
  - Spark with SparkBench suite
  - Giraph with Graphalytics benchmark suite

Limit DRAM capacity using cgroups

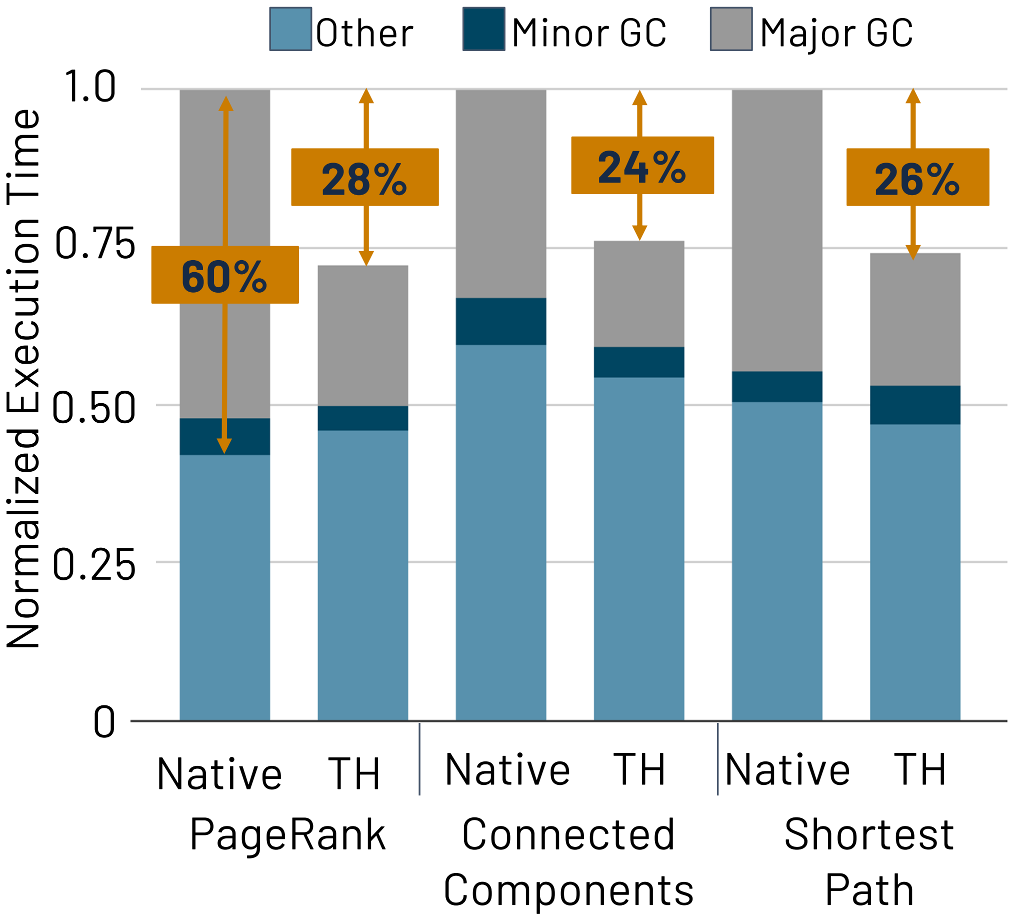
TeraHeap outperforms native Spark by up to 54%



Teraheap reduces S/D overhead

S/D in TeraHeap is due to shuffling

TeraHeap outperforms native Giraph by up to 28%



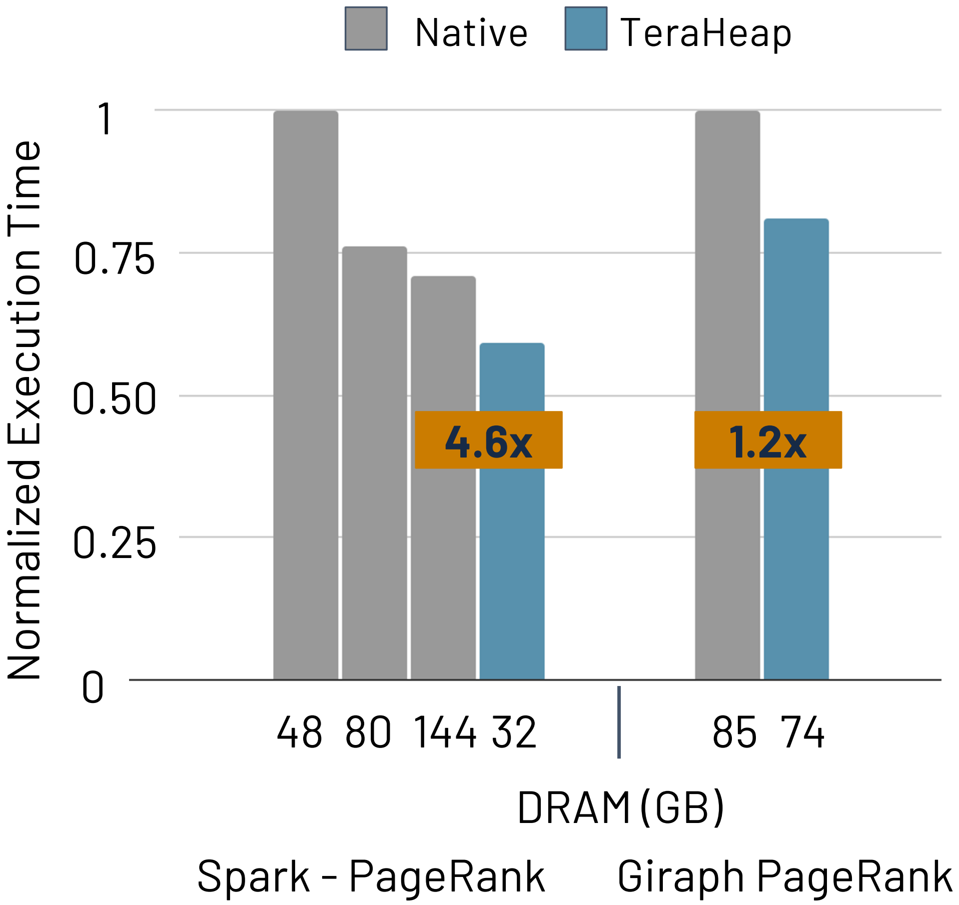
Main performance improvement

- Reduction of major GC (up to 50%)

Off-heap **reduces** heap pressure **temporarily**

- Giraph processes objects **only on-heap**
- Increases heap pressure → Increased GC!

TeraHeap reduces DRAM requirements



Provide direct access to H2 objects

Outperforms native Spark using 4.6x less DRAM

Outperforms native Giraph using 1.2x less DRAM

Key Takeaways

Analytics frameworks deal with large datasets using S/D

TeraHeap provides the illusion of single managed heap

- No S/D and no GC scans in the device heap for freeing space

Improves native Spark and Giraph performance by up to 54% and 28%

TeraHeap requires up to 4.6x less DRAM

Future work

- Eliminate hints by dynamically determining which objects to move to H2

TeraHeap: Reducing Memory Pressure for Managed Big Data Frameworks

Artifacts Available acm V1.1

Artifacts Evaluated acm Functional V1.1

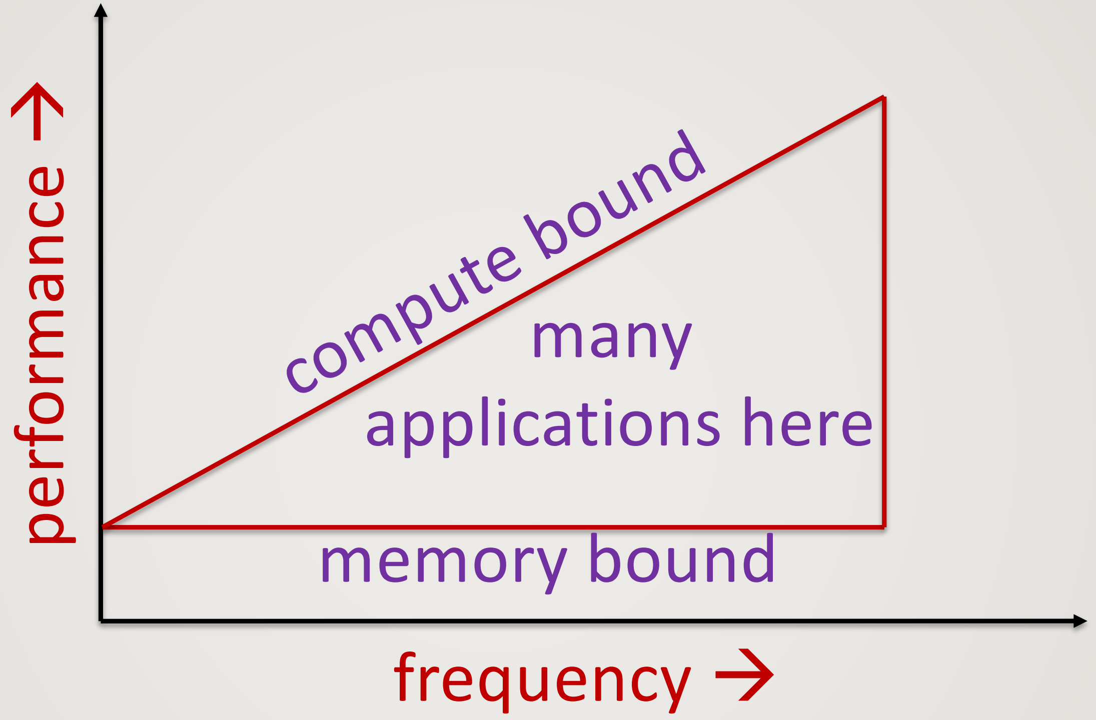
github.com/CARV-ICS-FORTH/teraheap

We thankfully acknowledge the support of the European Commision projects
EVOLVE (GA No 825061) and Eupex (GA No 101033975)
Iacovos G. Kolokasis is supported by the Meta Research PhD Fellowship (2022 – 2024)

DVFS PERFORMANCE PREDICTION FOR MANAGED MULTITHREADED APPLICATIONS

Shoaib Akram, Jennifer B. Sartor, Lieven Eeckhout
Ghent University, Belgium
Shoaib.Akram@elis.UGent.be

DVFS Performance Prediction



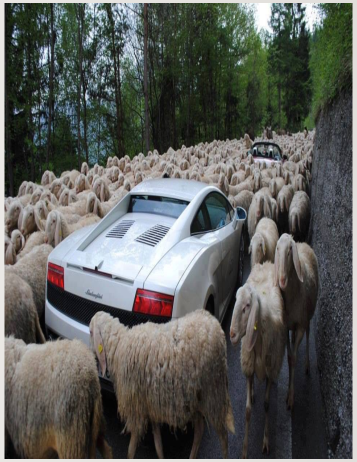
Sample at all DVFS states ☹

Estimate performance ☺

Managed Multithreaded Applications

Heterogeneity

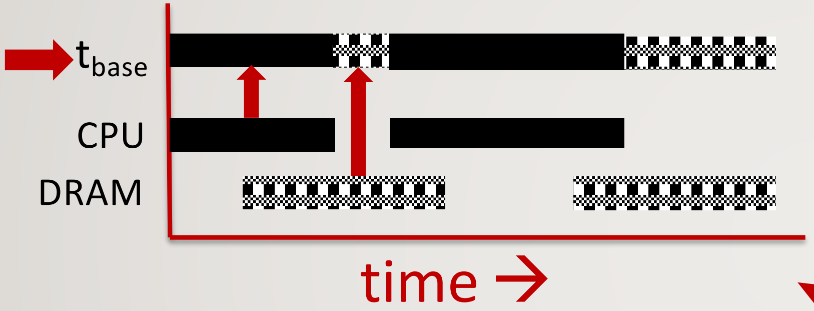
Synchronization



Store Bursts

Background

Base Frequency



- t_{base} sum of
  - –Scaling (S)
  - –Non-Scaling (NS)

Target Frequency

- r = Base/Target
- S → S * r
- NS → No change
- t_{target} = (S*r) + NS

- Not simple
- OOO+MLP

State of the Art

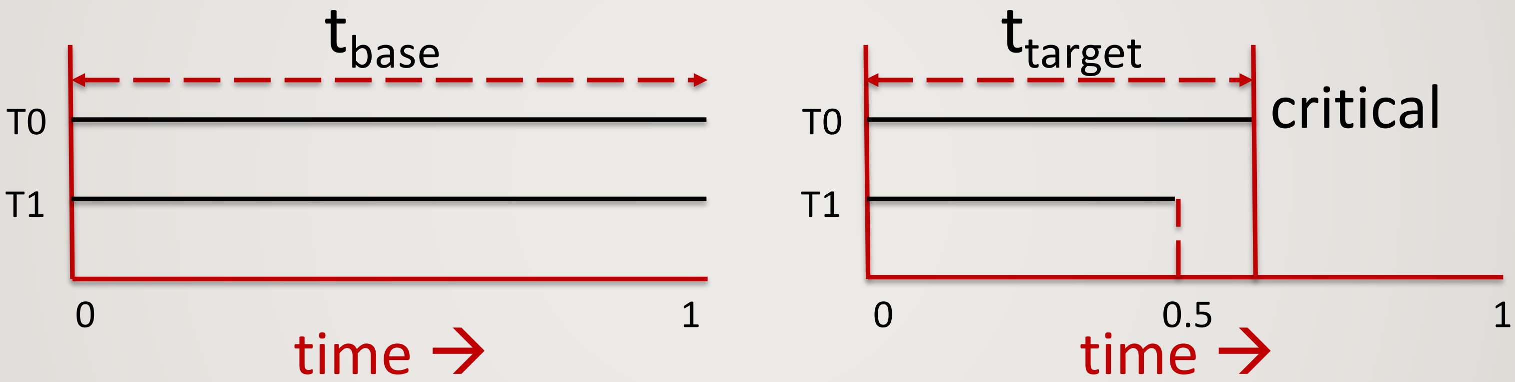
- **CRIT** estimates non-scaling by
  - Measuring critical path through loads
  - Ignoring store operations

R. Miftakhutdinov, E. Ebrahimi, and Y. N. Patt. Predicting performance impact of DVFS for realistic memory systems. MICRO, 2012.

Multithreaded CRIT (M+CRIT)

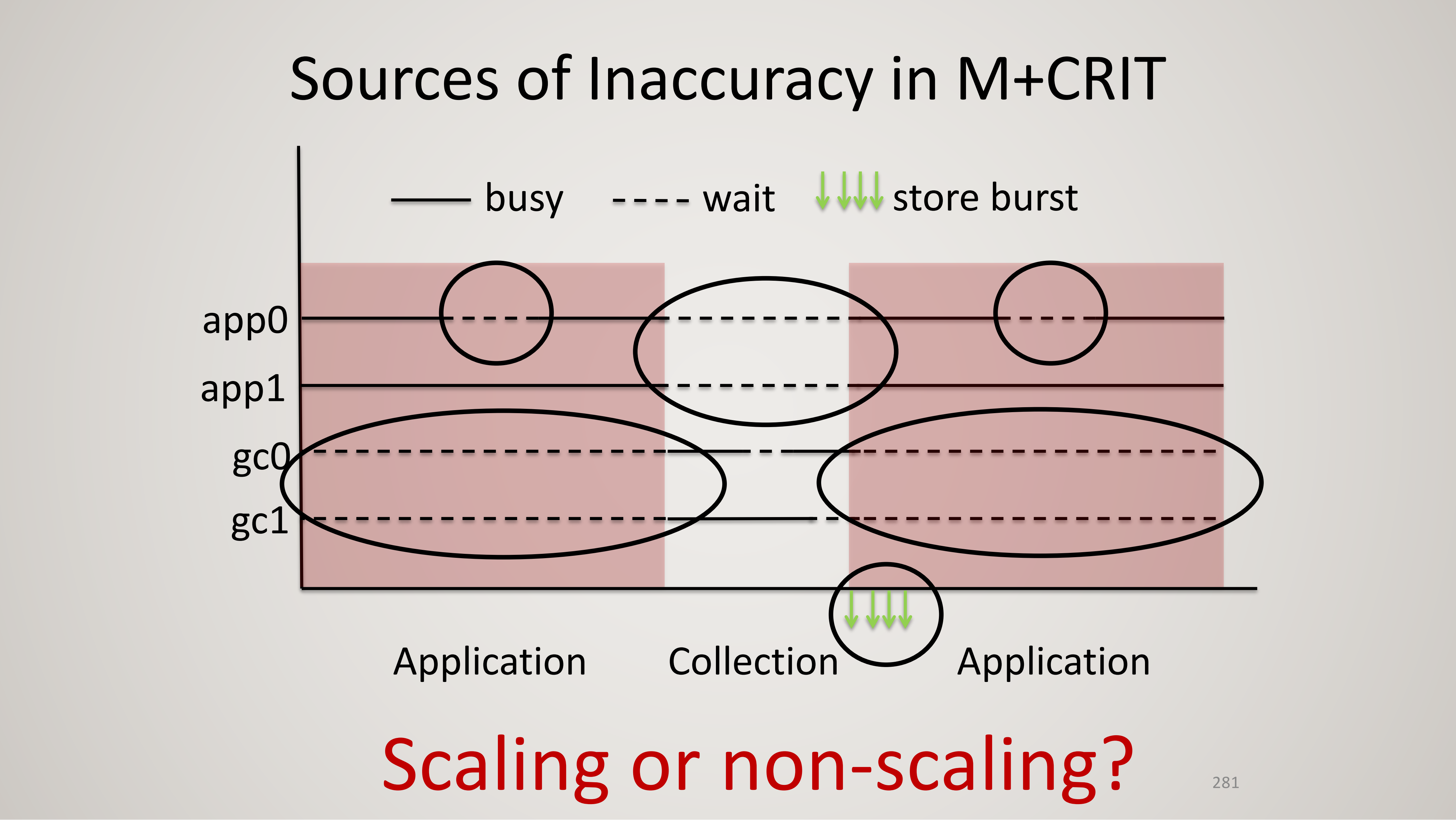
Base Frequency → Target Frequency

2X



Use CRIT to identify each thread's non-scaling

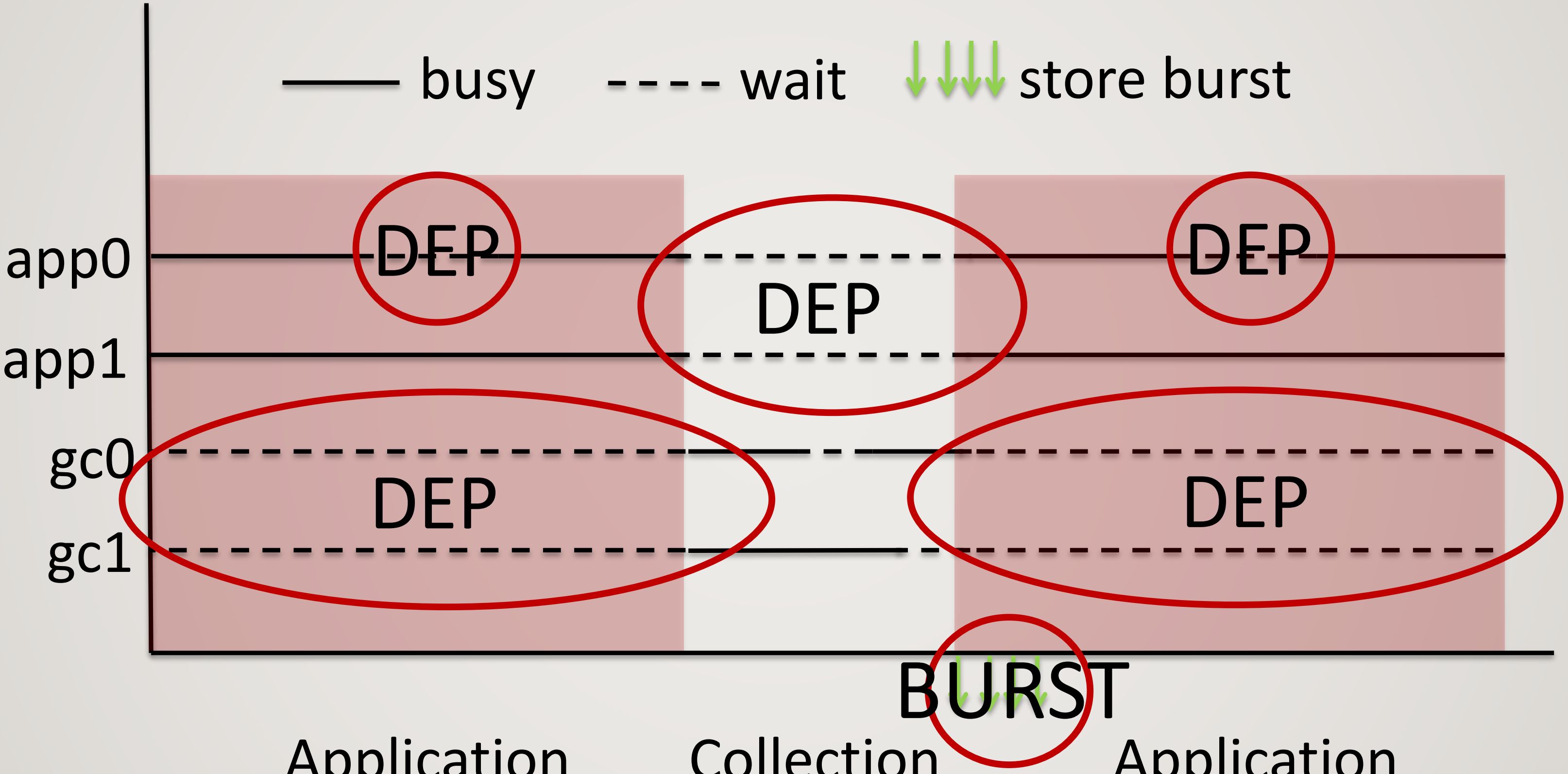
High error for multithreaded Java!



Sources of Inaccuracy in M+CRIT

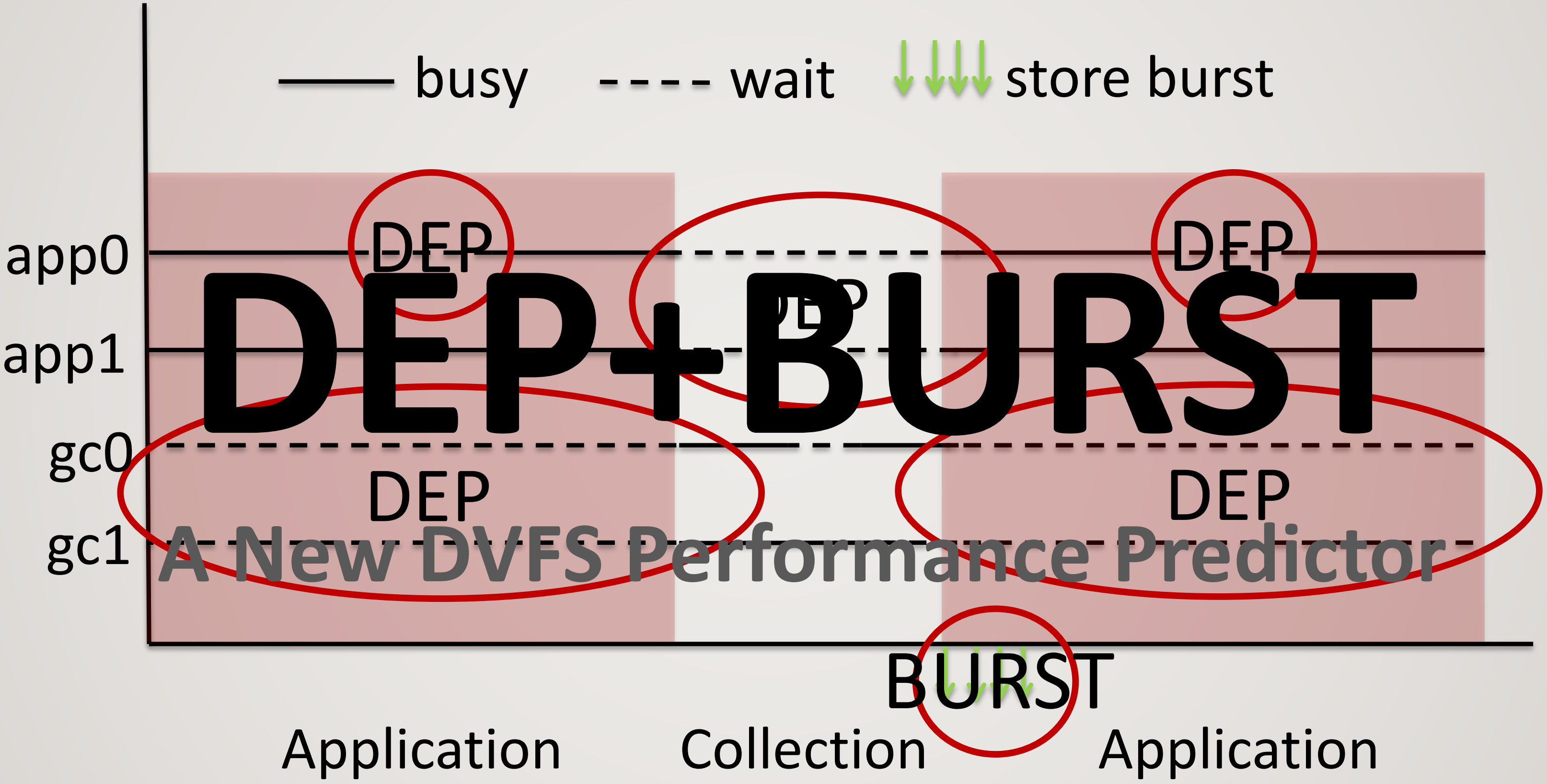
Scaling or non-scaling?

Sources of Inaccuracy in M+CRIT



Scaling or non-scaling?

Our Contribution



DEP+BURST

A New DVFS Performance Predictor

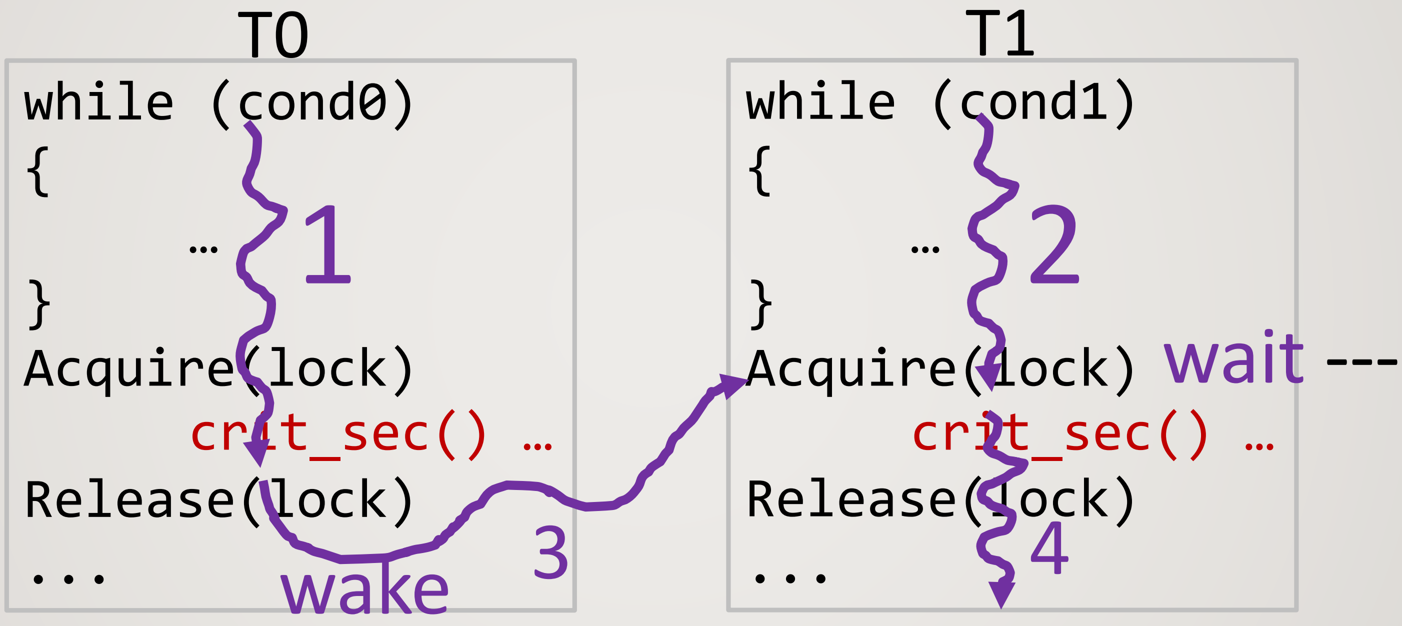
Scaling or non-scaling?

Our Contribution

DEP+BURST

A New DVFS Performance Predictor

Example: Inter-thread Dependences



T0

```
while (cond0)
{
      …
}
Acquire(lock)
      crit_sec() …
Release(lock)
...
```

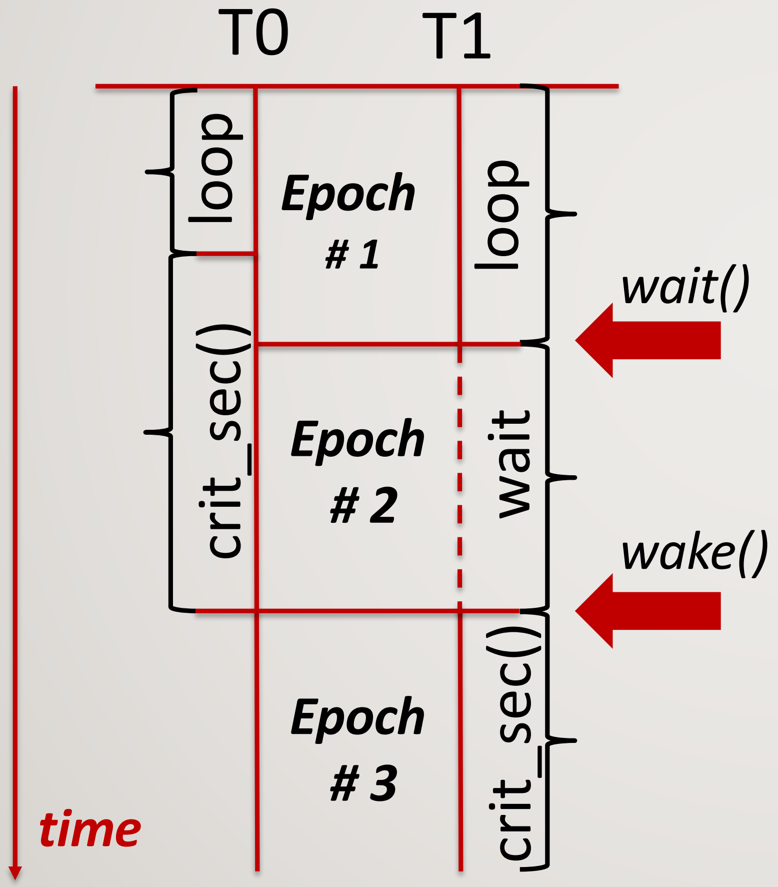
T1

```
while (cond1)
{
      …
}
Acquire(lock) wait ---
      crit_sec() …
Release(lock)
...
```

- Intercept synchronization activity
- Reconstruct execution at target frequency

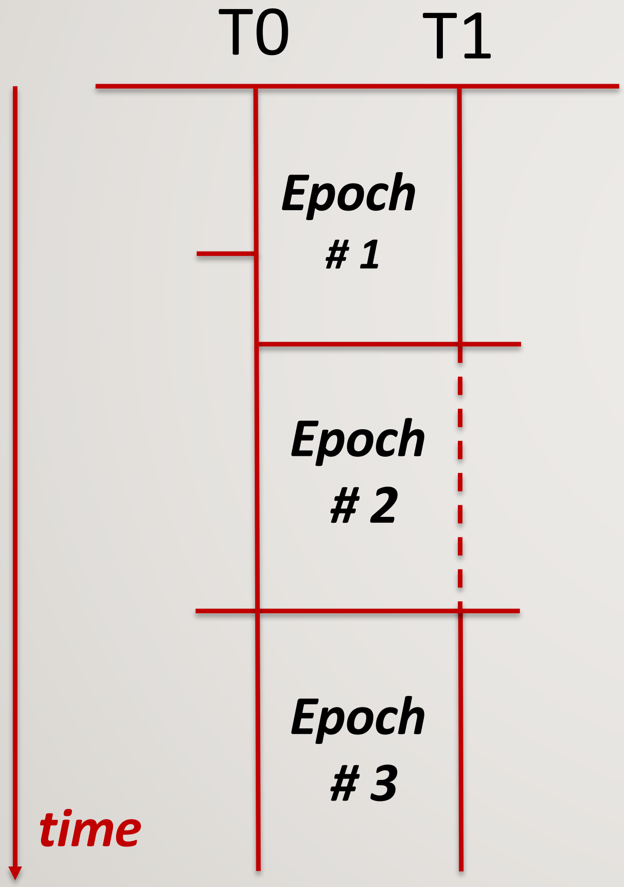
Identifying Synchronization Epochs

Base Frequency ➡ Target Frequency



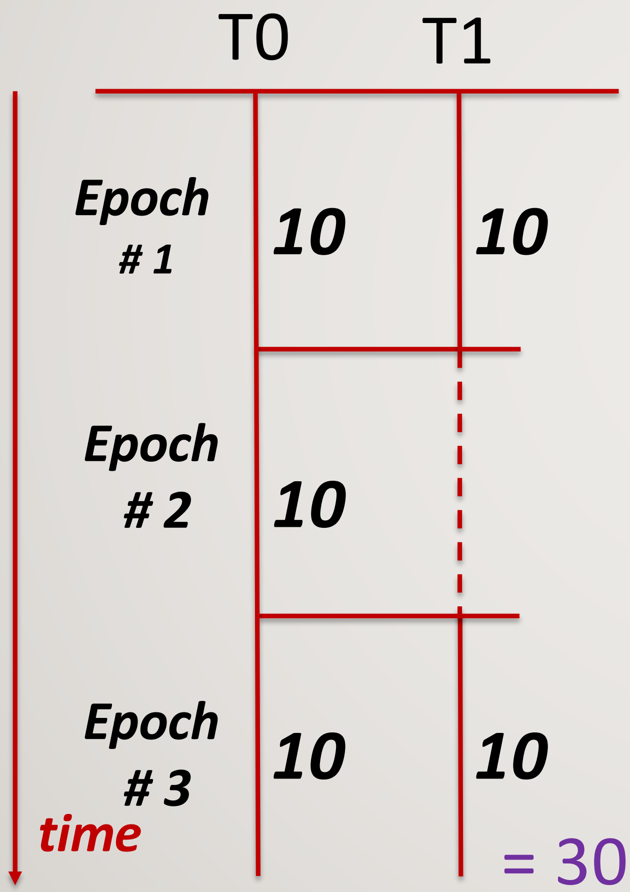
Identifying Synchronization Epochs

Base Frequency ➡ Target Frequency



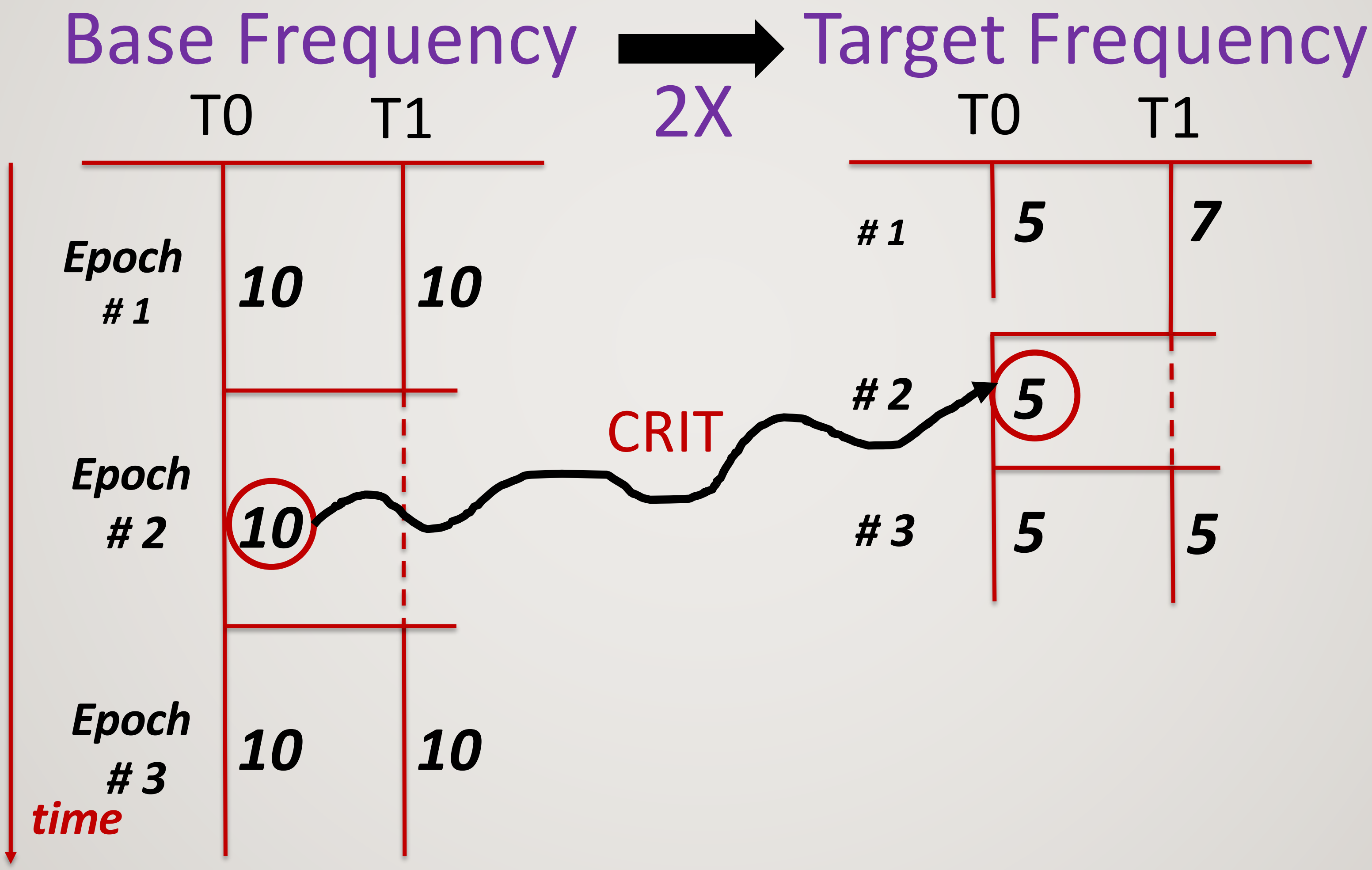
Identifying Synchronization Epochs

Base Frequency → Target Frequency



= 30 units

Reconstruction at Target Frequency



Base Frequency → 2X → Target Frequency

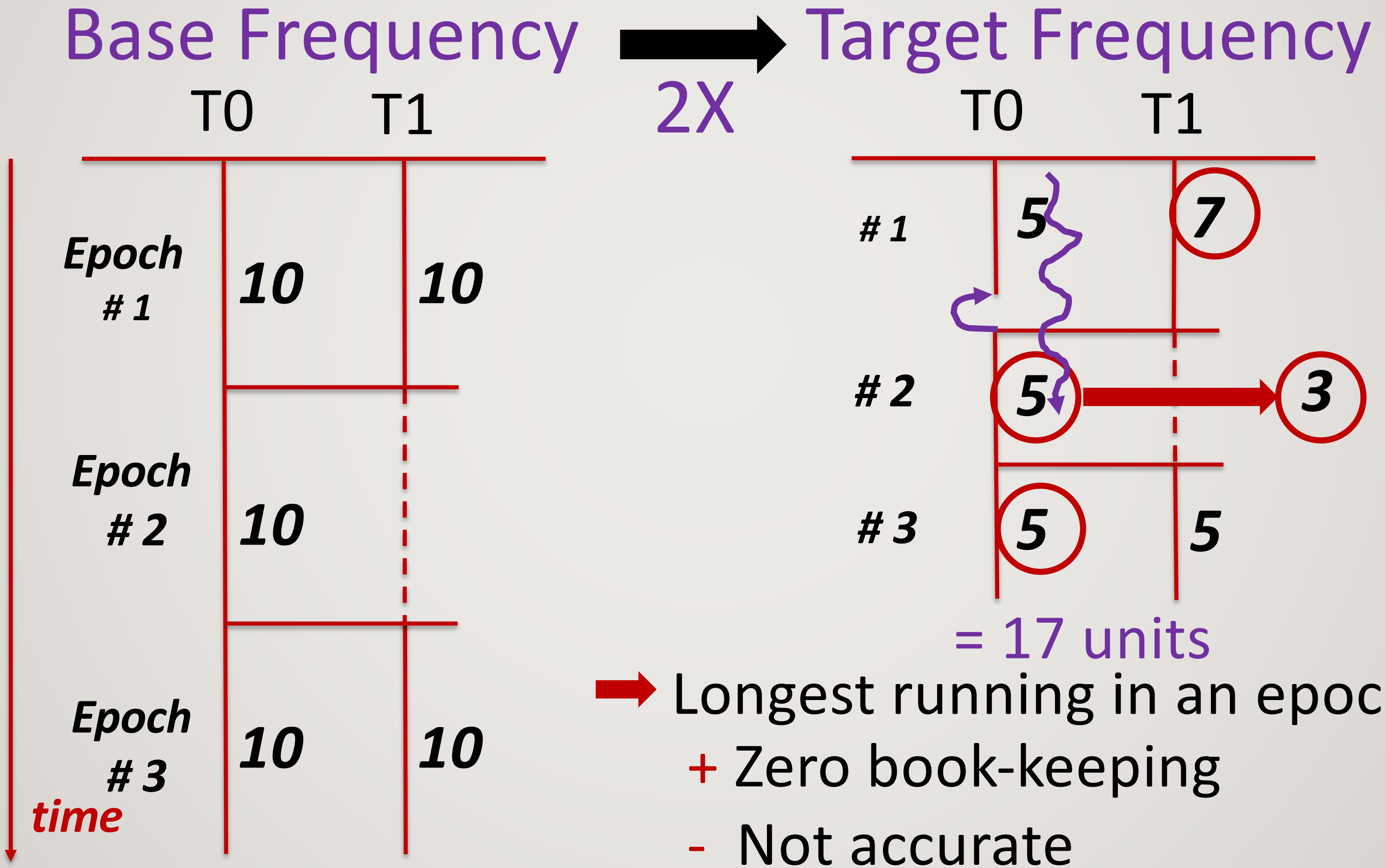
	T0	T1
Epoch # 1	10	10
Epoch # 2	10	
Epoch # 3	10	10

time

CRIT

	T0	T1
# 1	5	7
# 2	5	
# 3	5	5

Reconstruction at Target Frequency



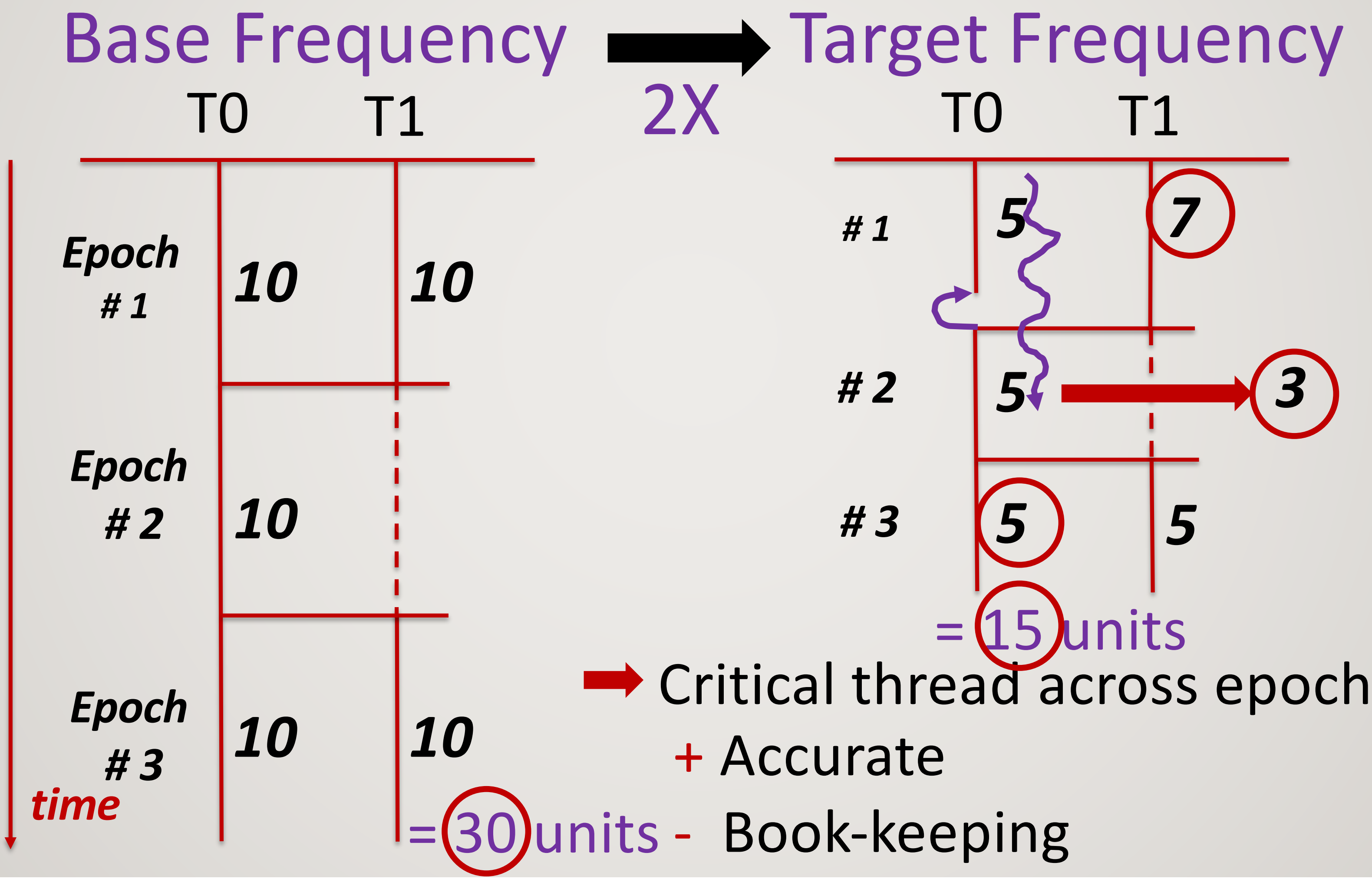
= 17 units

➡ Longest running in an epoch

+ Zero book-keeping

- Not accurate

Reconstruction at Target Frequency

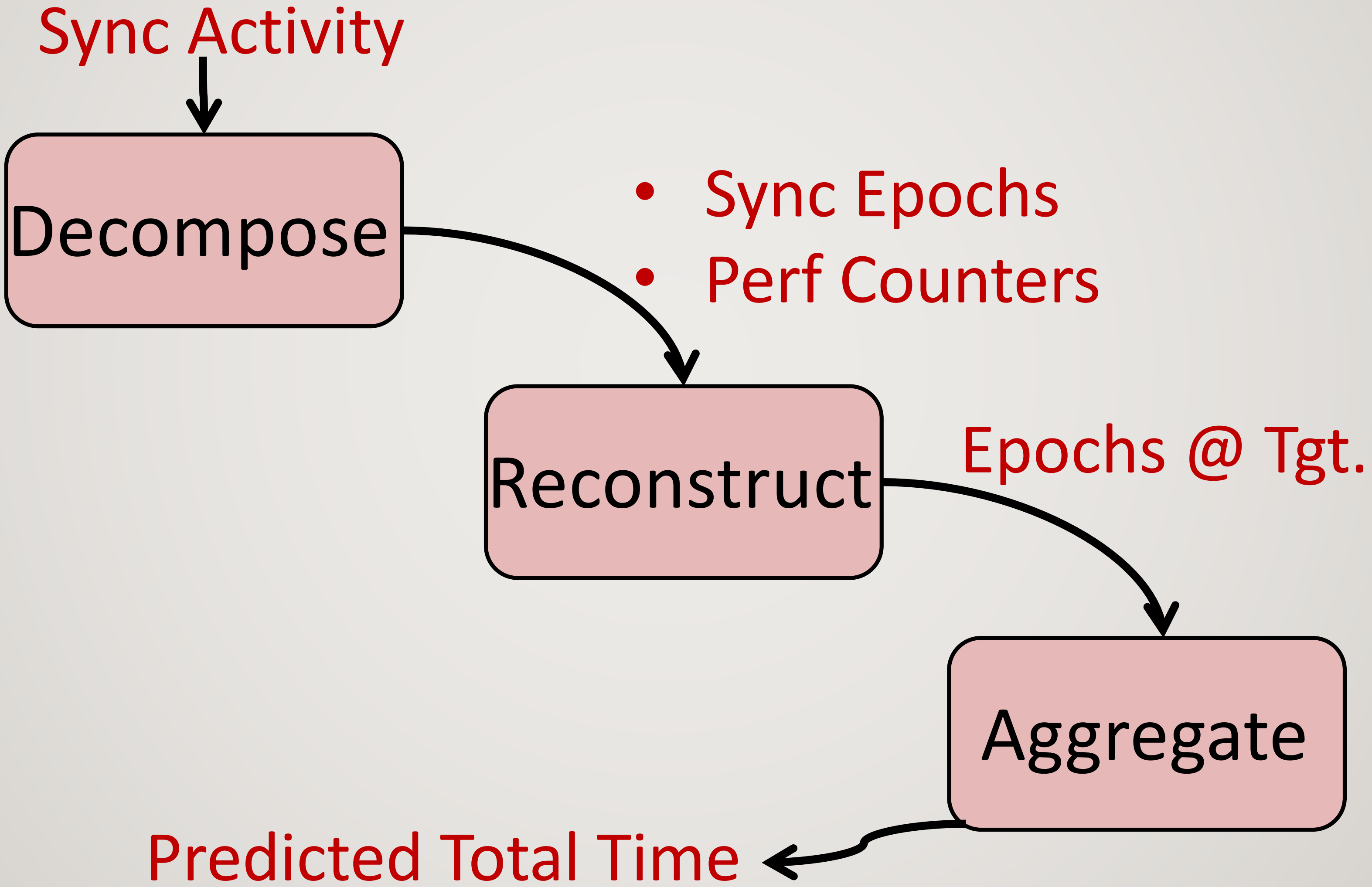


➡ Critical thread across epochs

+ Accurate

- Book-keeping

DEP: Summary



Sync Activity

Decompose

- Sync Epochs
- Perf Counters

Reconstruct

Epochs @ Tgt.

Aggregate

Predicted Total Time

Our Contribution

DEP+BURST

A New DVFS Performance Predictor

Our Contribution

DEP+BURST

A New DVFS Performance Predictor

Store Bursts

- Reasons
  - Zero initialization
  - Copying collectors
- Modeling Steps
  - Track how long the store queue is full
  - Add to the non-scaling component

Methodology

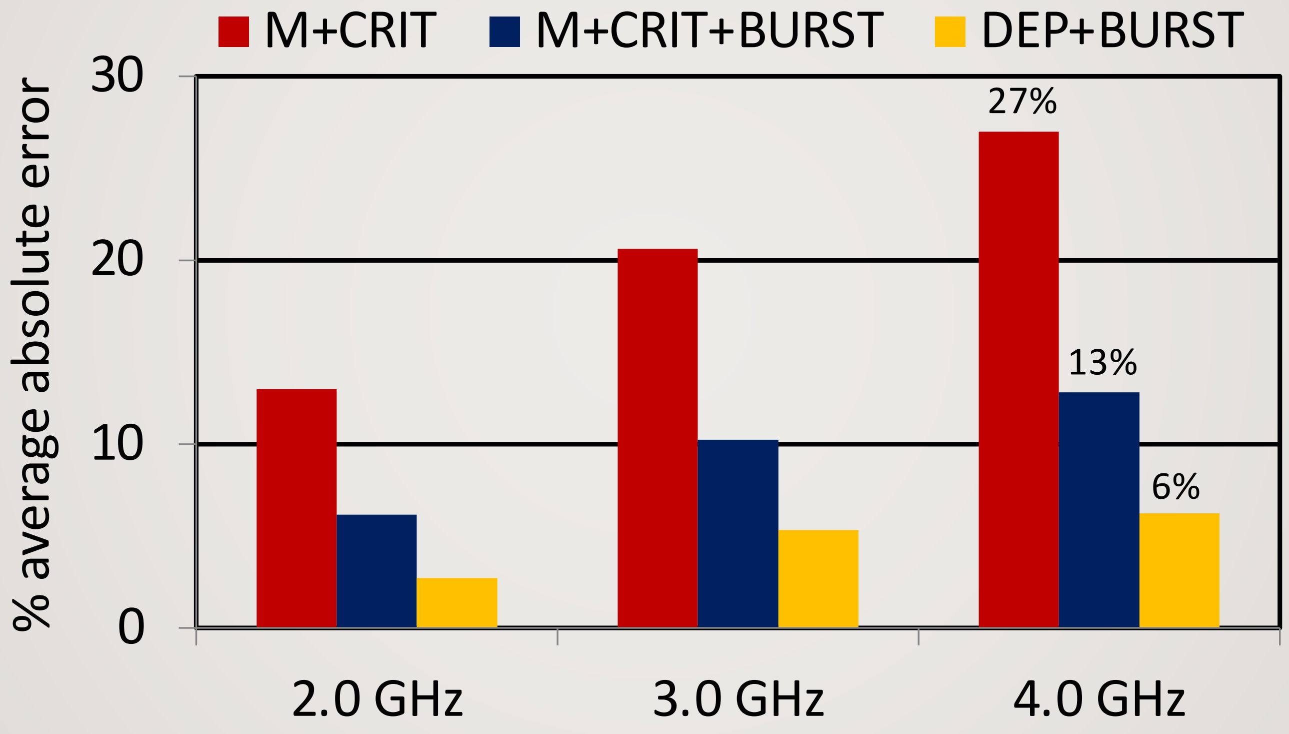
- Jikes RVM 3.1.2
- Production collector (Immix)
- # GC threads = 2
- 2x min. heap

Version 6.0

- 4 cores, 1.0 GHz → 4.0 GHz
- 3-level cache hierarchy
- LLC fixed to 1.5 GHz
- DVFS settings for 22 nm Haswell

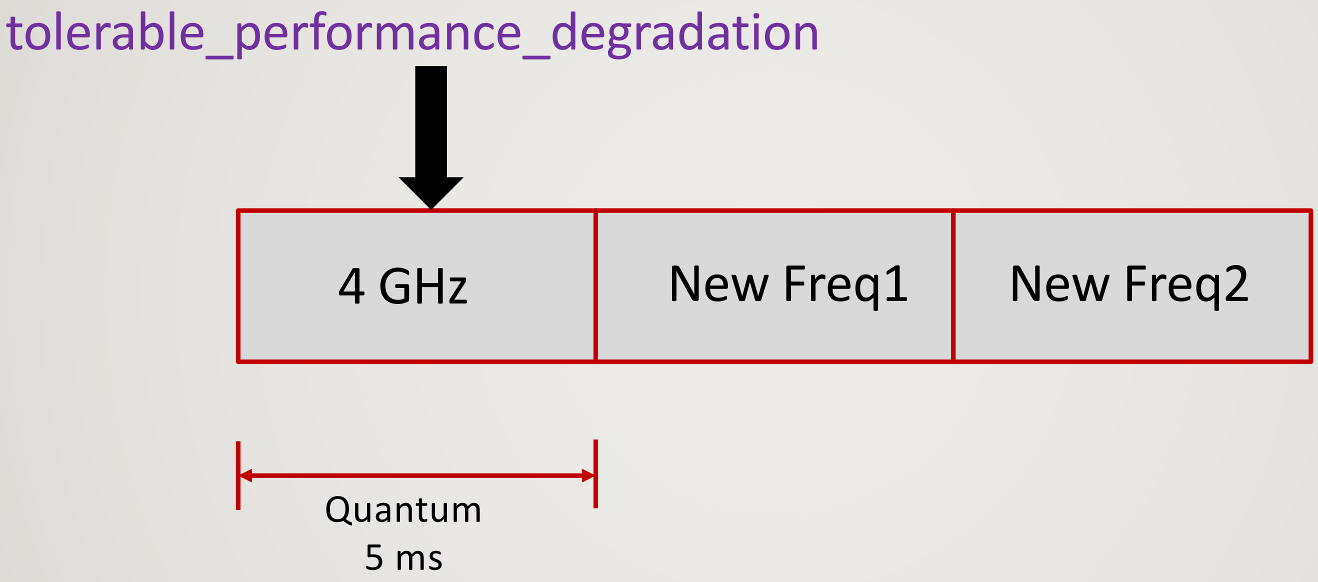
- Seven multithreaded benchmarks
- Four application threads

Accuracy



Baseline Frequency = 1.0 GHz

Energy Manager



tolerable_performance_degradation

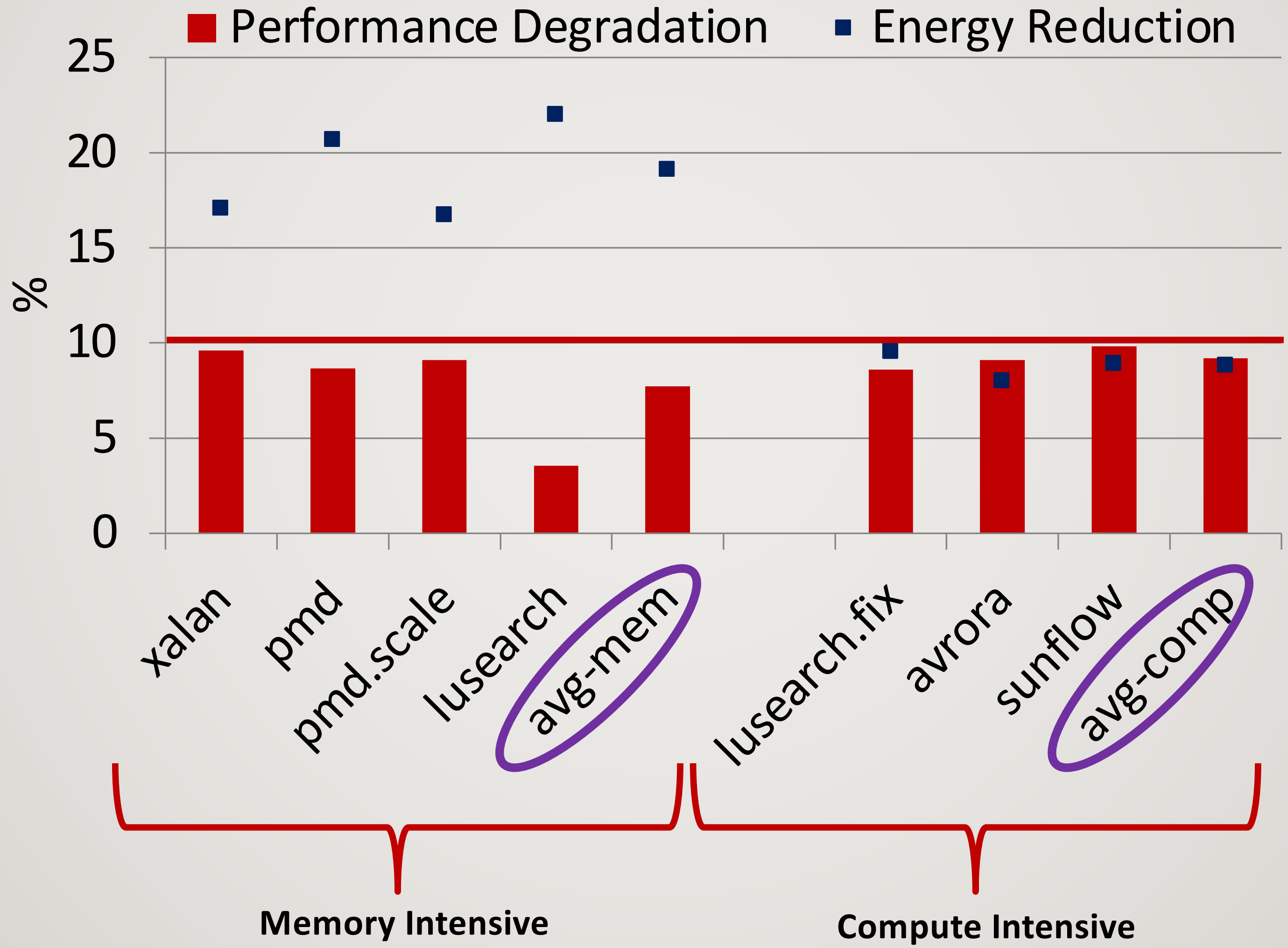
4 GHz

New Freq1

New Freq2

Quantum
5 ms

Energy Savings



Memory Intensive

Compute Intensive

Conclusions

- DEP+BURST: First predictor that accounts for
  - Application and service threads
  - Synchronization → inter-thread dependencies
  - Store bursts
- High accuracy
  - Less than 10% estimation error for seven Java bmarks.
- Negligible hardware cost
  - One extra performance counter
  - Minor book-keeping across epochs
- Demonstrated energy savings
  - 20 % avg. for a 10% slowdown (mem-intensive Java apps.)

DVFS PERFORMANCE PREDICTION FOR MANAGED MULTITHREADED APPLICATIONS

Thank You !

Shoaib.Akram@elis.UGent.be

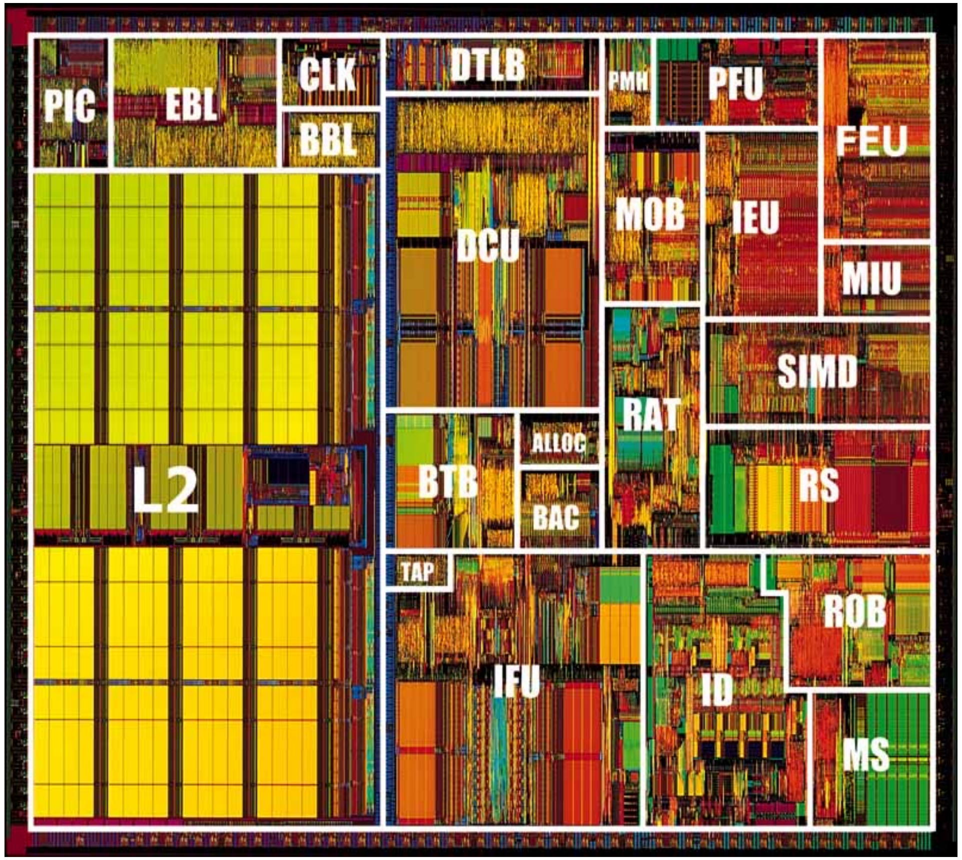
GHENT
UNIVERSITY

Managed Language Runtimes on Heterogeneous Hardware:

Optimizations for Performance, Efficiency and Lifetime Improvement

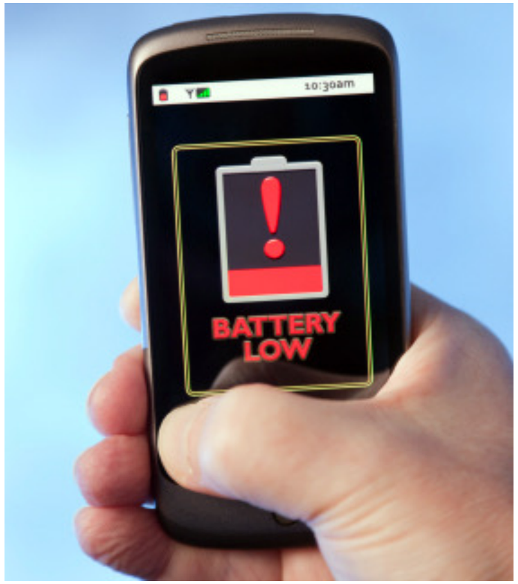
Programming Across the Stack Workshop
Invited Talk *by*
Shoaib Akram, Ghent University

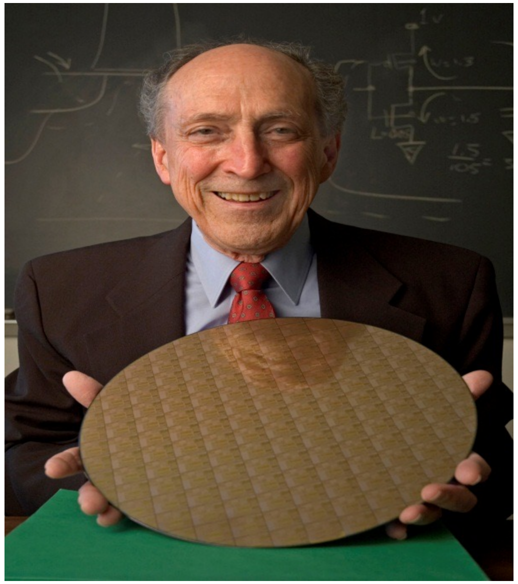
Circa 2000, hardware features were fixed at design time



One GHz
Big Core
DRAM

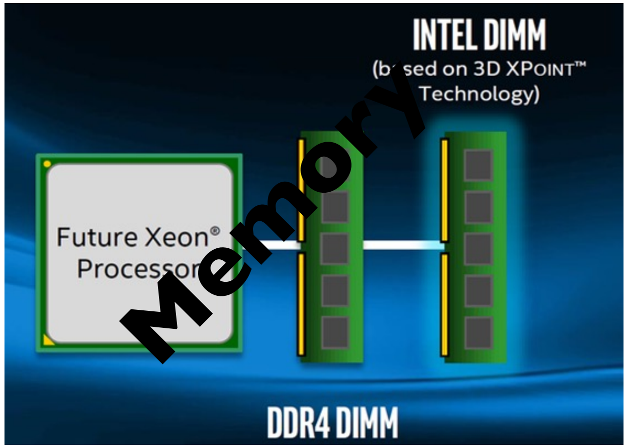
As time passed, efficiency became a first order concern

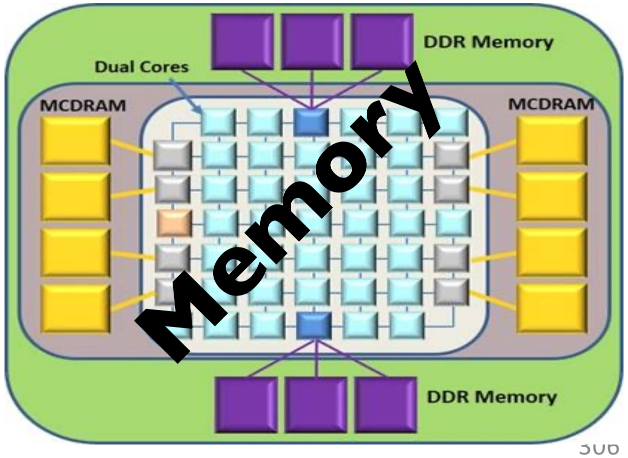




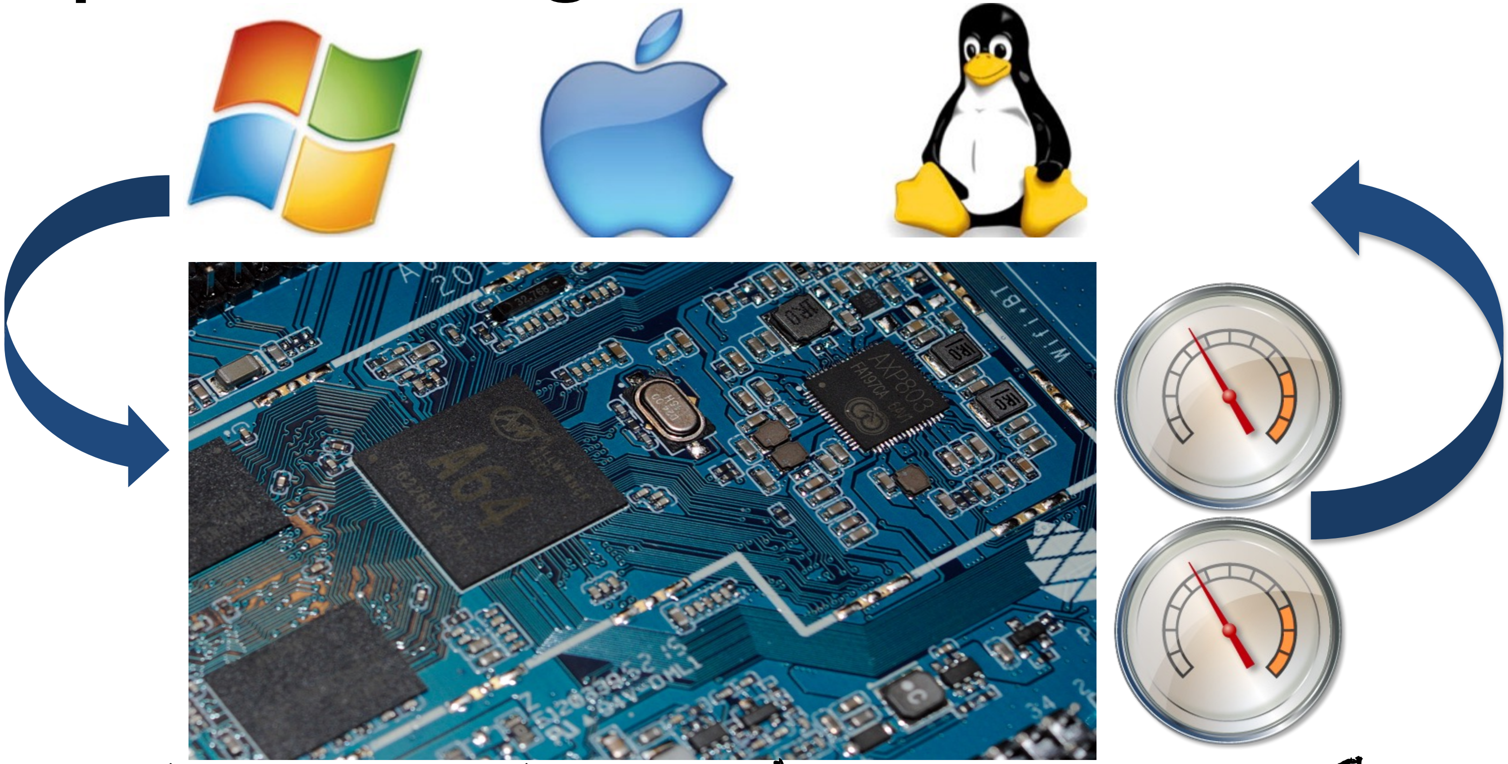
Billions of watts in data centre power
More search queries on mobiles
End of Dennard scaling

Hardware designers turned to flexibility for improving efficiency





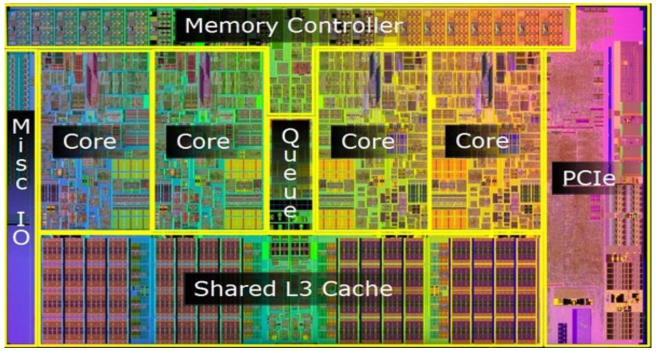
Event counters became the key to help OS configure hardware



Understand the behaviour of individual threads and adapt

What about software evolution?

Multithreading



Language runtimes

JavaScript

Microsoft® .net™

Prior Work in making software aware of hardware heterogeneity

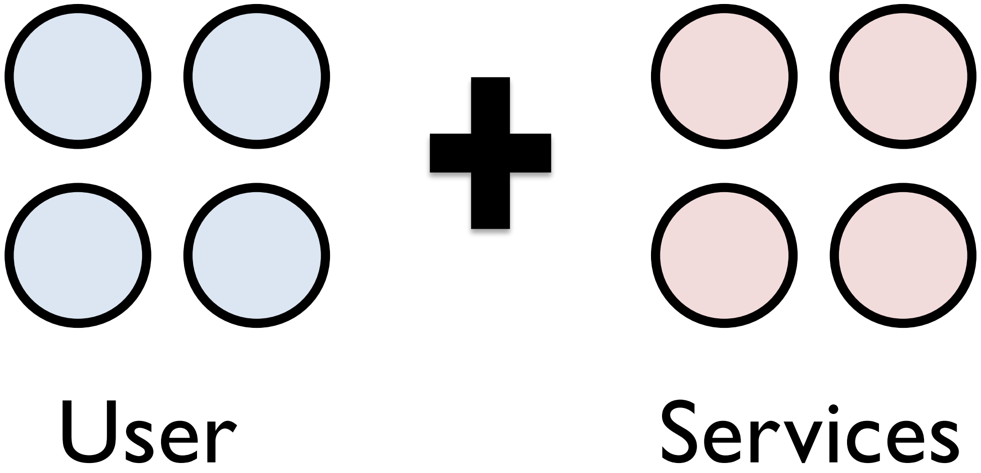
- ✧ Mostly for native applications
- ✧ No input from language runtime or user

Managed languages are popular due to their productivity advantage

The 2015 Top Ten Programming Languages, spectrum.ieee.org.

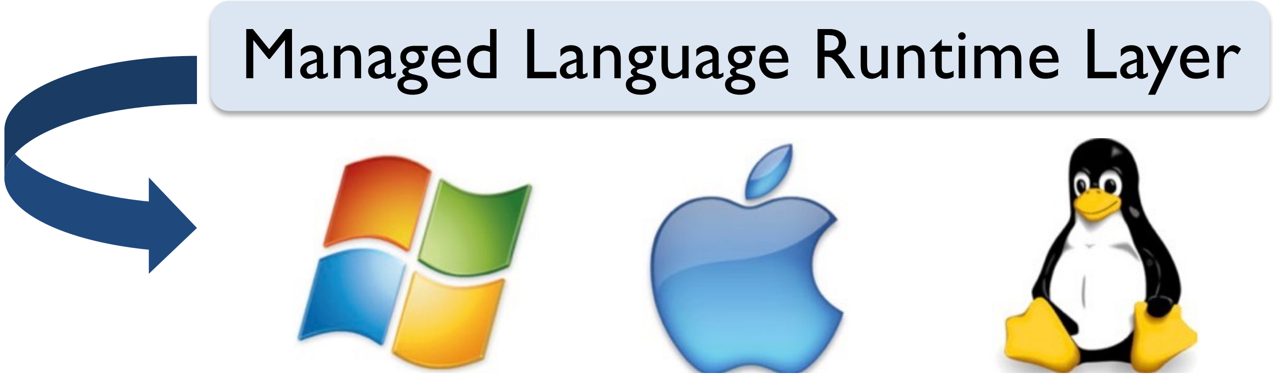
Research activity # 1: Behavior of managed multithreaded environs

- ✧ Scheduling user vs. service threads
- ✧ Understanding the impact of synchronization



Research activity #2: Include the runtime for better policy making

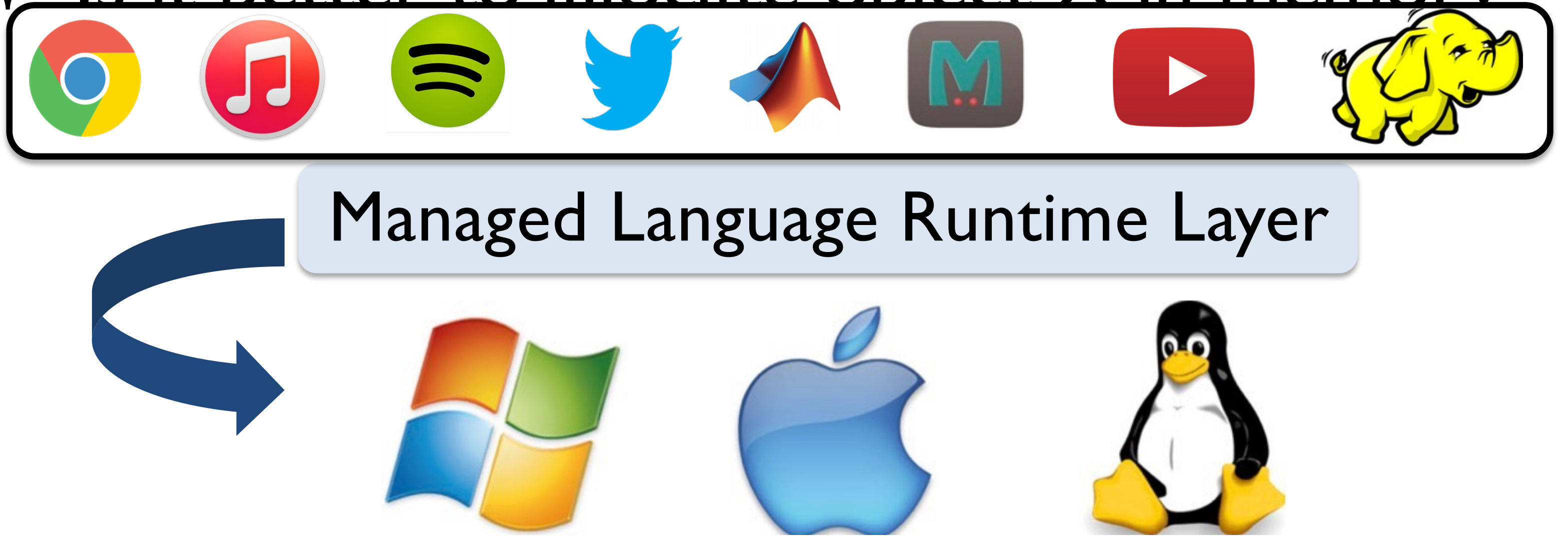
- Is any service thread critical to performance?
- Is it better to allocate object X in memory type T?



Managed Language Runtime Layer

Research activity #2: Include the runtime for better policy making

- Is any service thread critical to performance?
- Is it better to allocate object X in memory



Managed Language Runtime Layer

Bridge the application-OS gap

Agenda

1. Scheduling concurrent collection on heterogeneous multicores
2. Predicting the performance impact of DVFS for managed multithreaded applications
3. Using the garbage collector to guide object placement in hybrid memory

ACM Transactions on Architecture and Code Optimization (TACO), 2016

Boosting the Priority of Garbage:

Scheduling Collection on Heterogeneous Multicore Processors

Shoaib Akram, Jennifer B. Sartor, Kenzo Van Craeynest,
Wim Heirman, Lieven Eeckhout
Ghent University, Belgium
Shoaib.Akram@UGent.be

GHENT UNIVERSITY

VUB VRIJE UNIVERSITEIT BRUSSEL

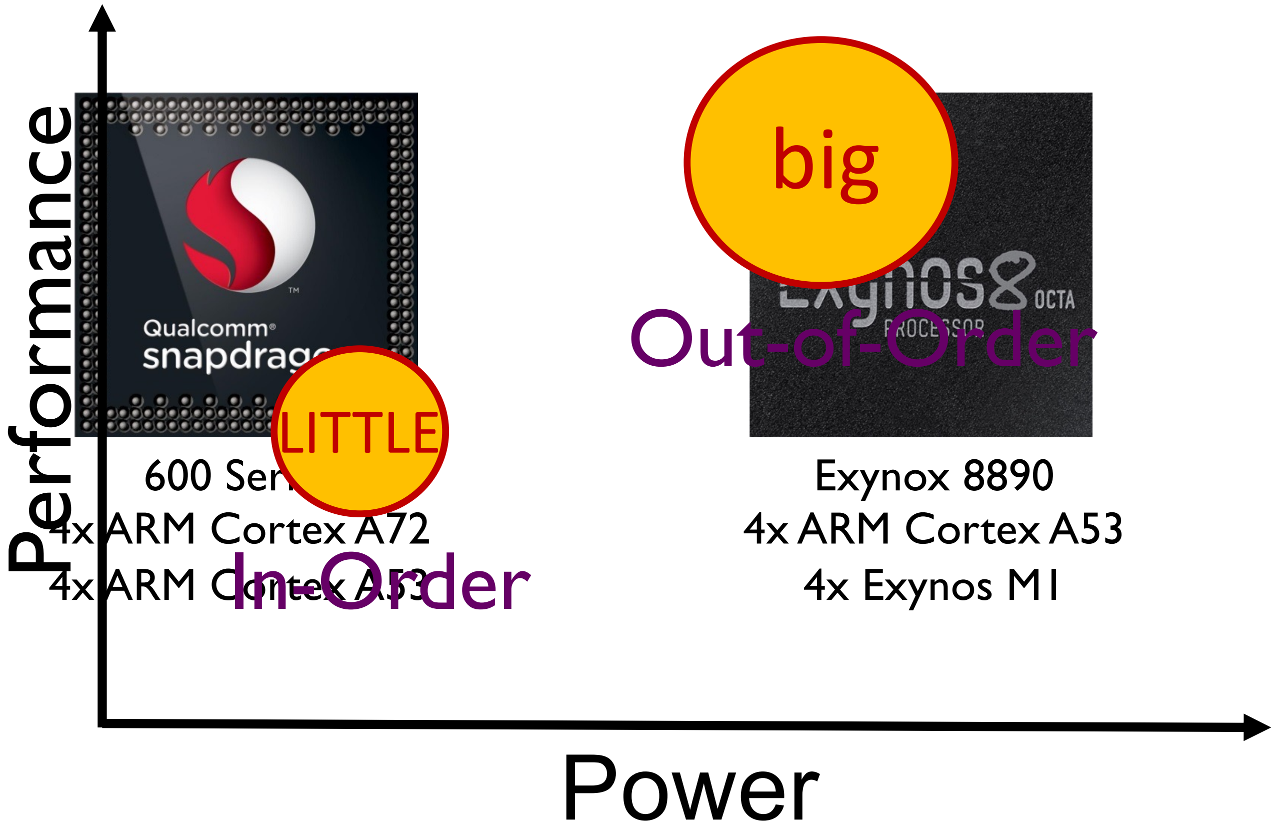
intel®

Garbage collector automatically reclaims memory for reuse

CPU usage is not negligible

Concurrent collectors fit for multicores

Heterogeneous multicores consist of different core types



Performance

big

Out-of-Order

LITTLE

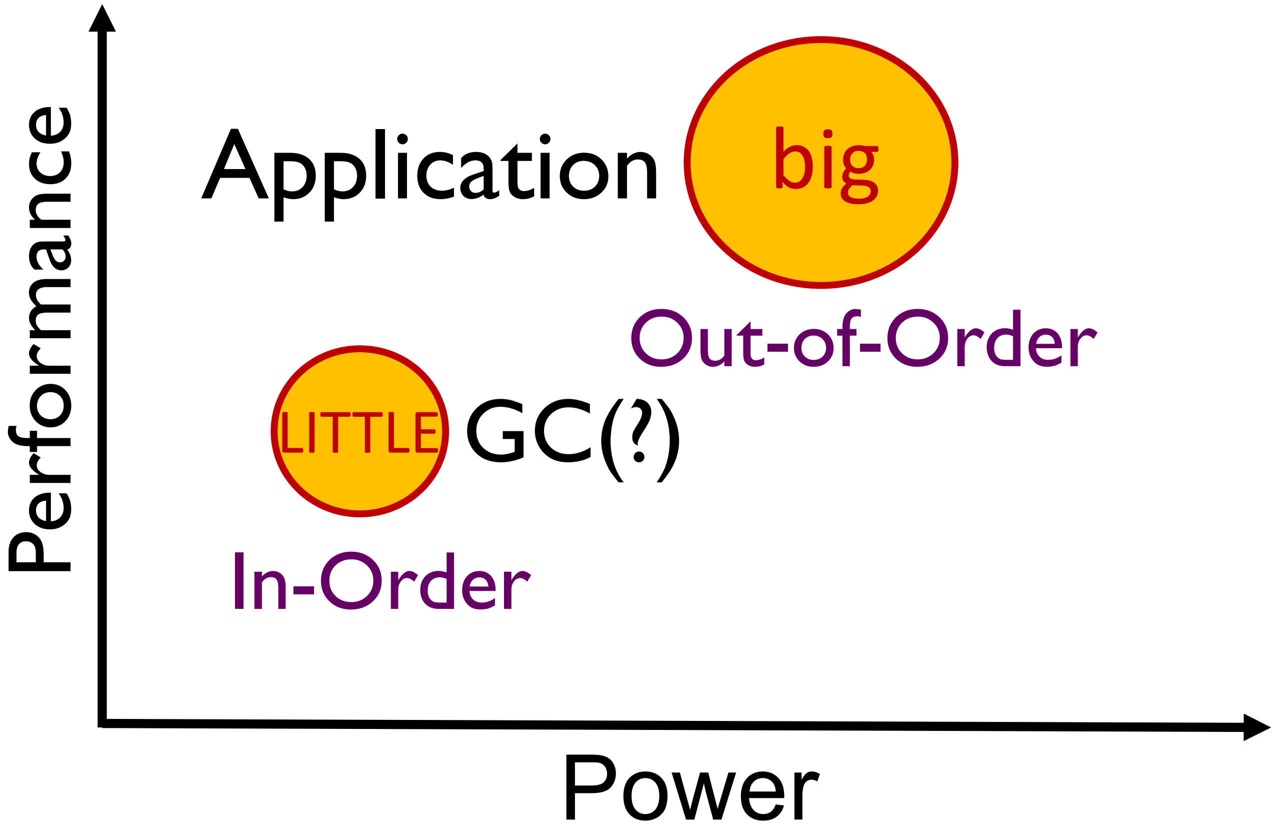
In-Order

600 Ser
4x ARM Cortex A72
4x ARM Cortex A53

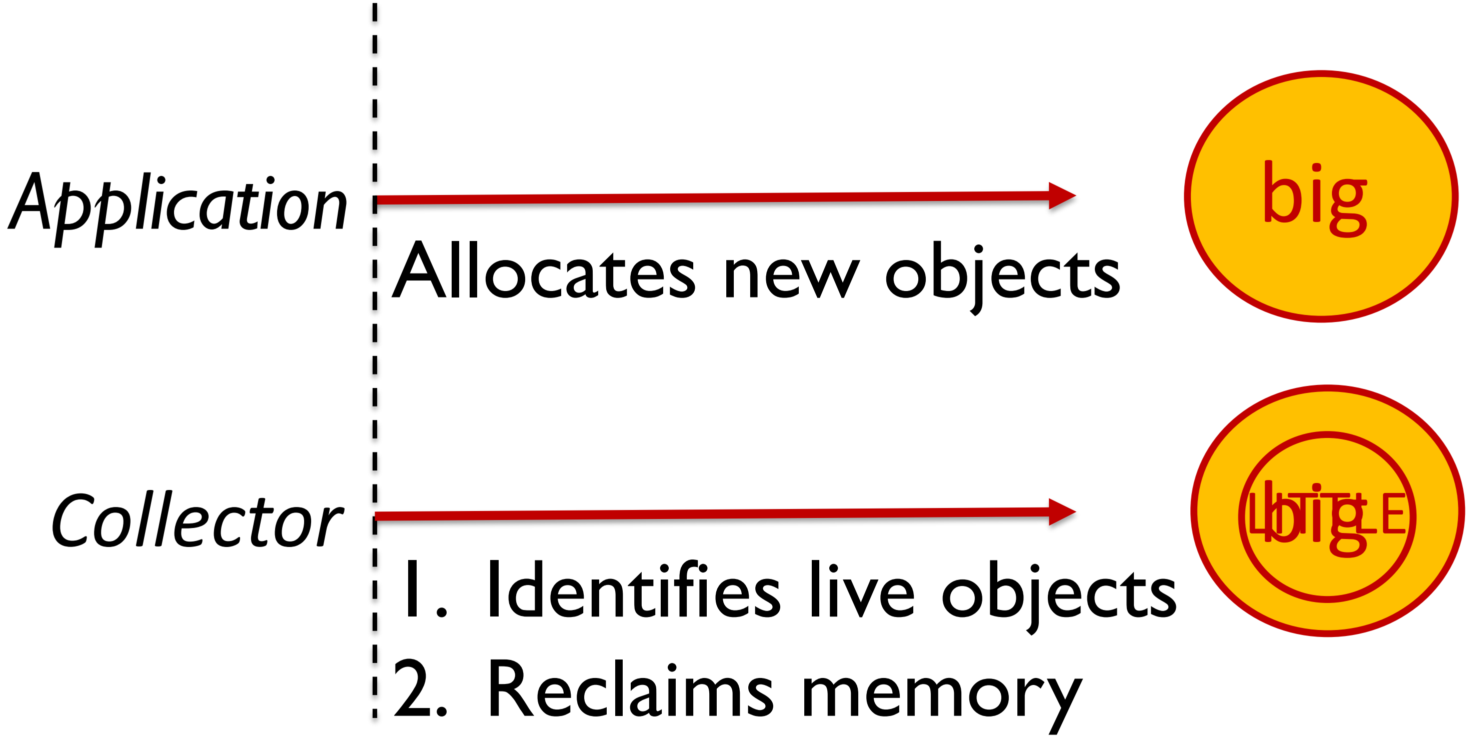
Exynox 8890
4x ARM Cortex A53
4x Exynos M1

Power

Which core type to run application versus the collector threads?



What is the performance impact of running collector on LITTLE core?



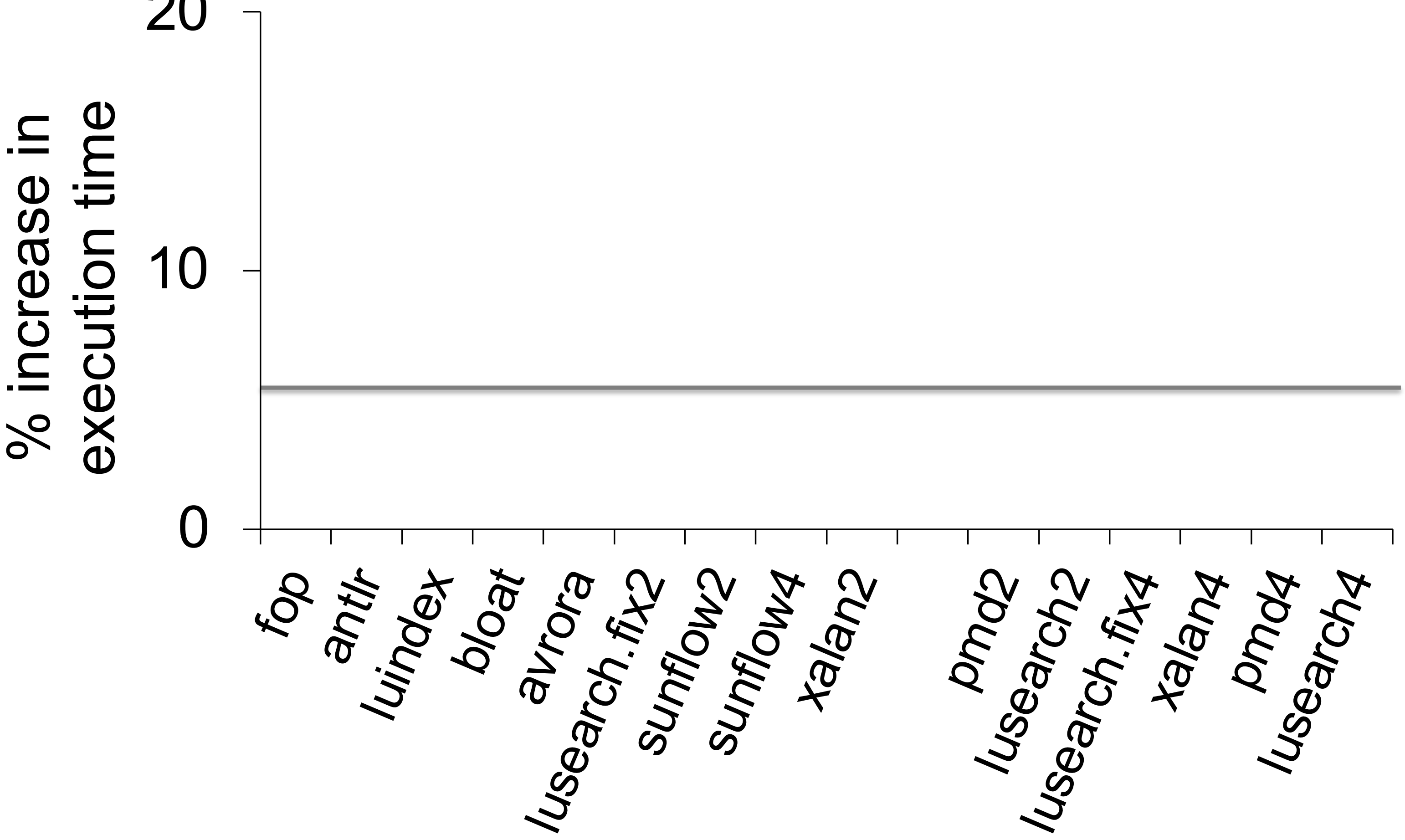
Application

Allocates new objects

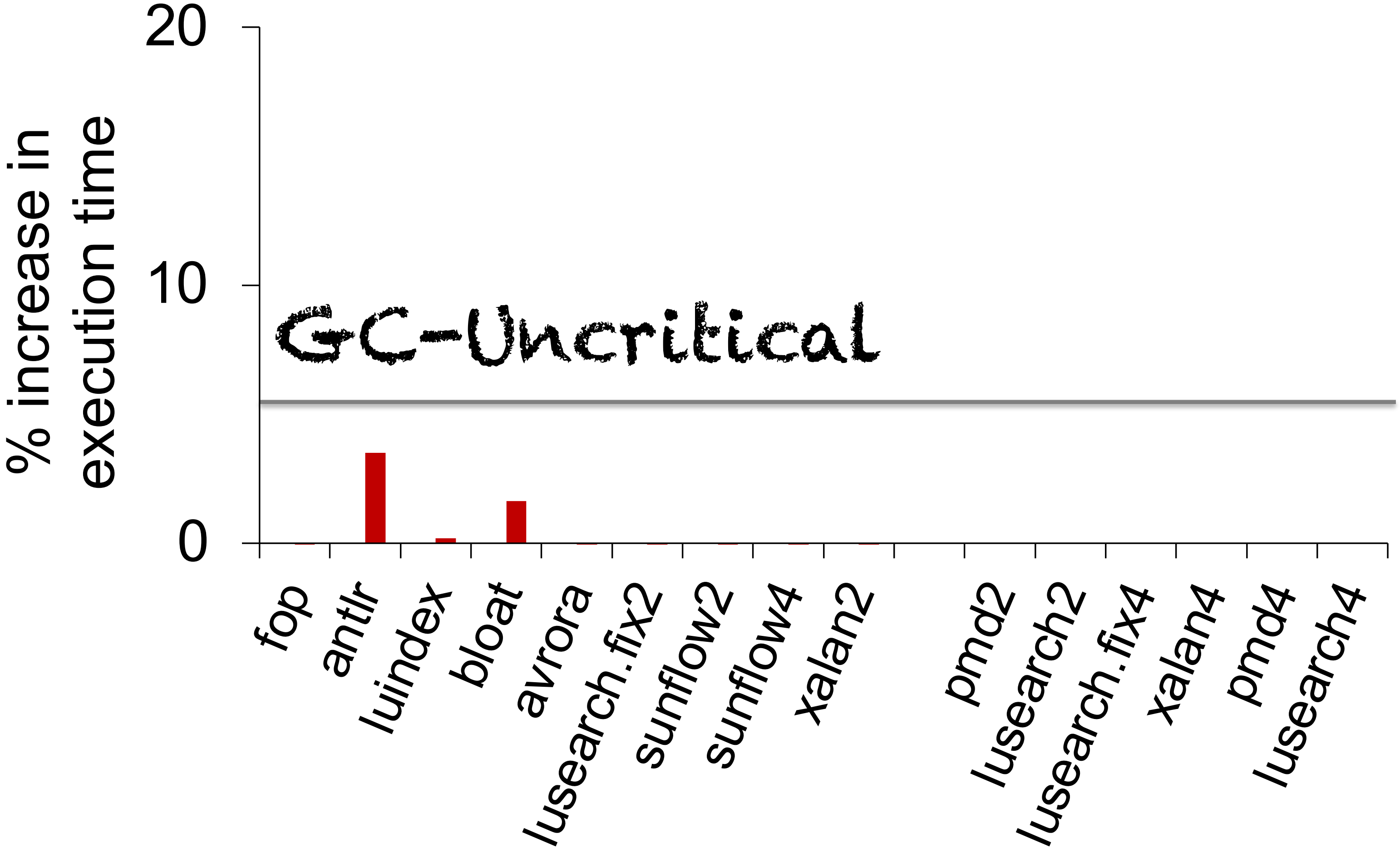
Collector

1. Identifies live objects
2. Reclaims memory

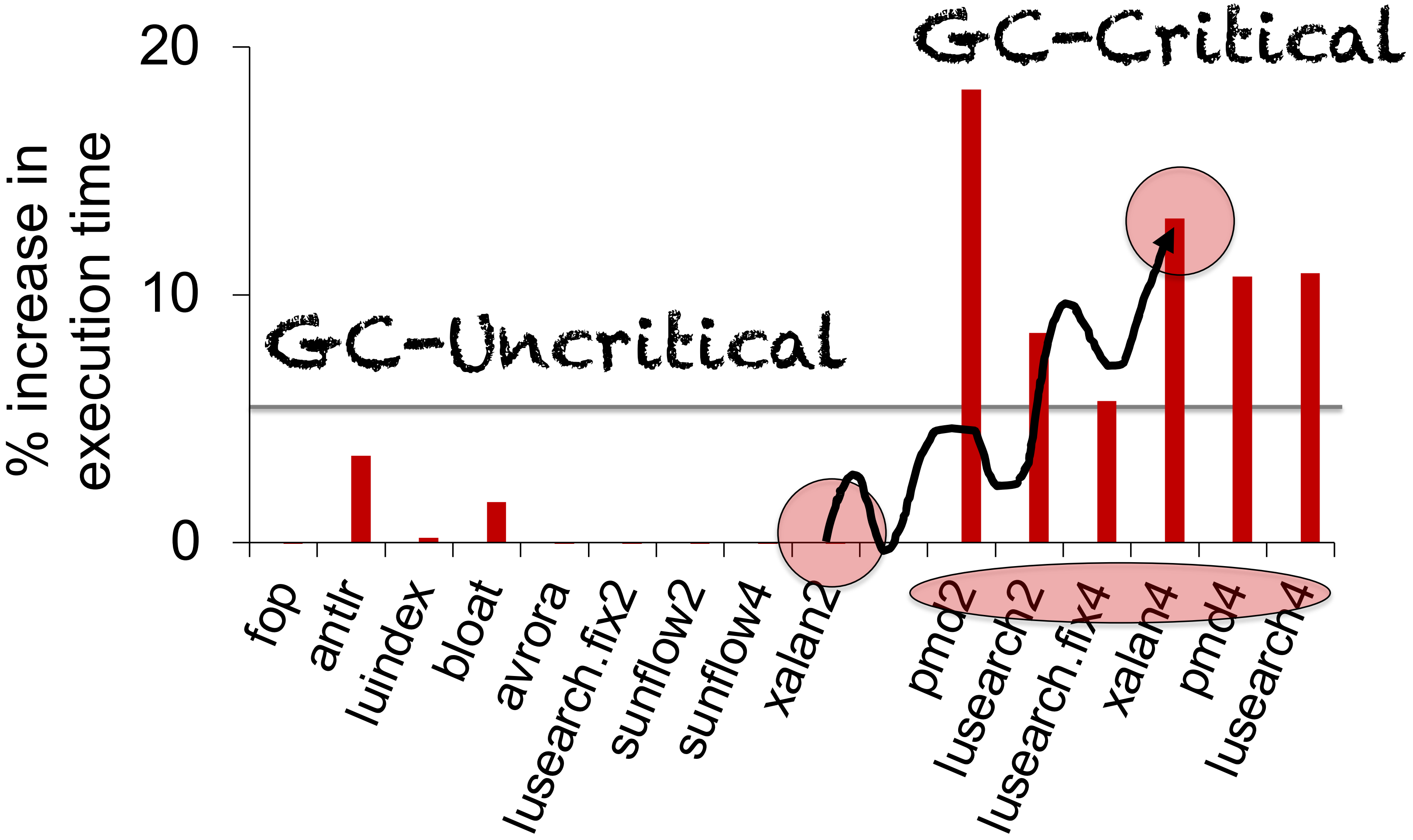
Running GC on LITTLE degrades performance of some applications



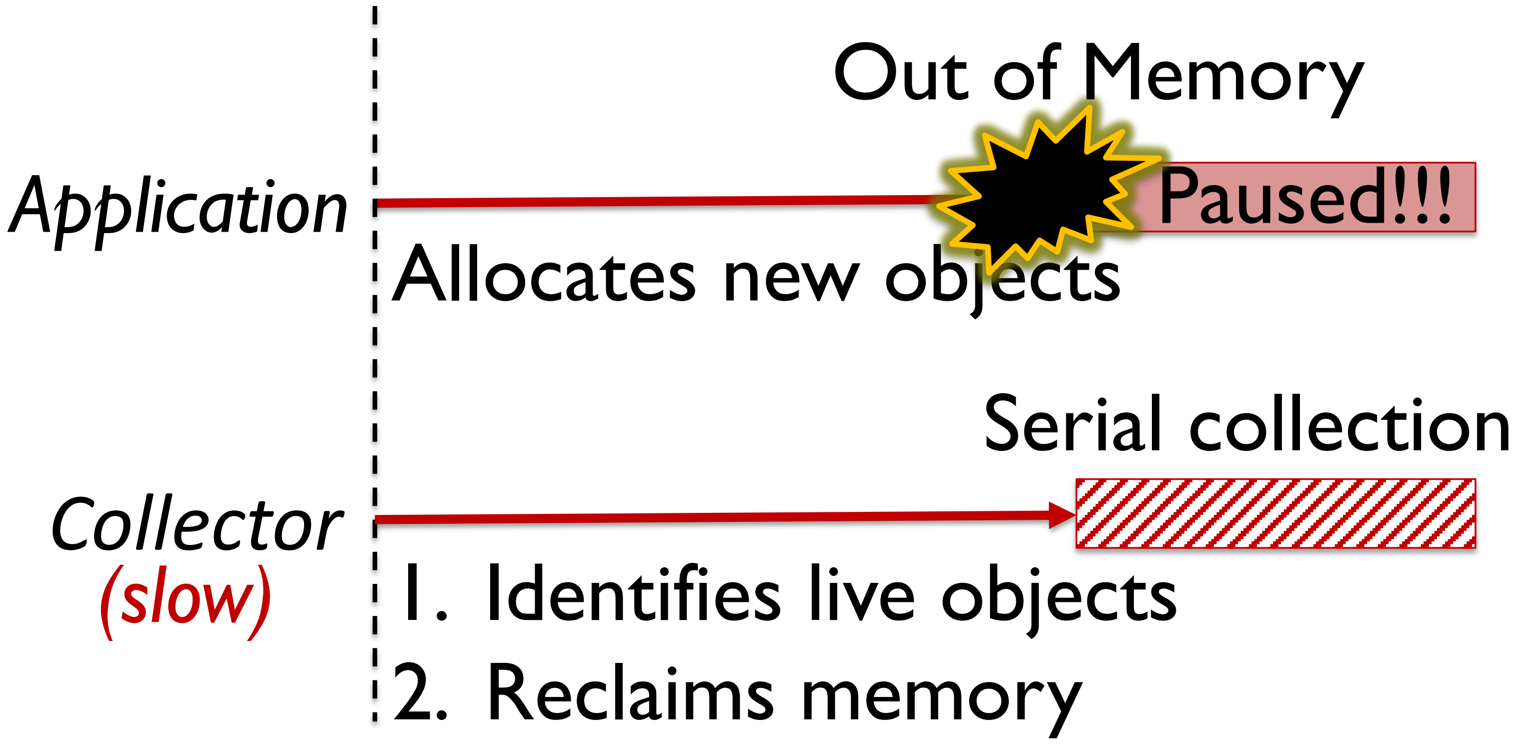
Running GC on LITTLE degrades performance of some applications



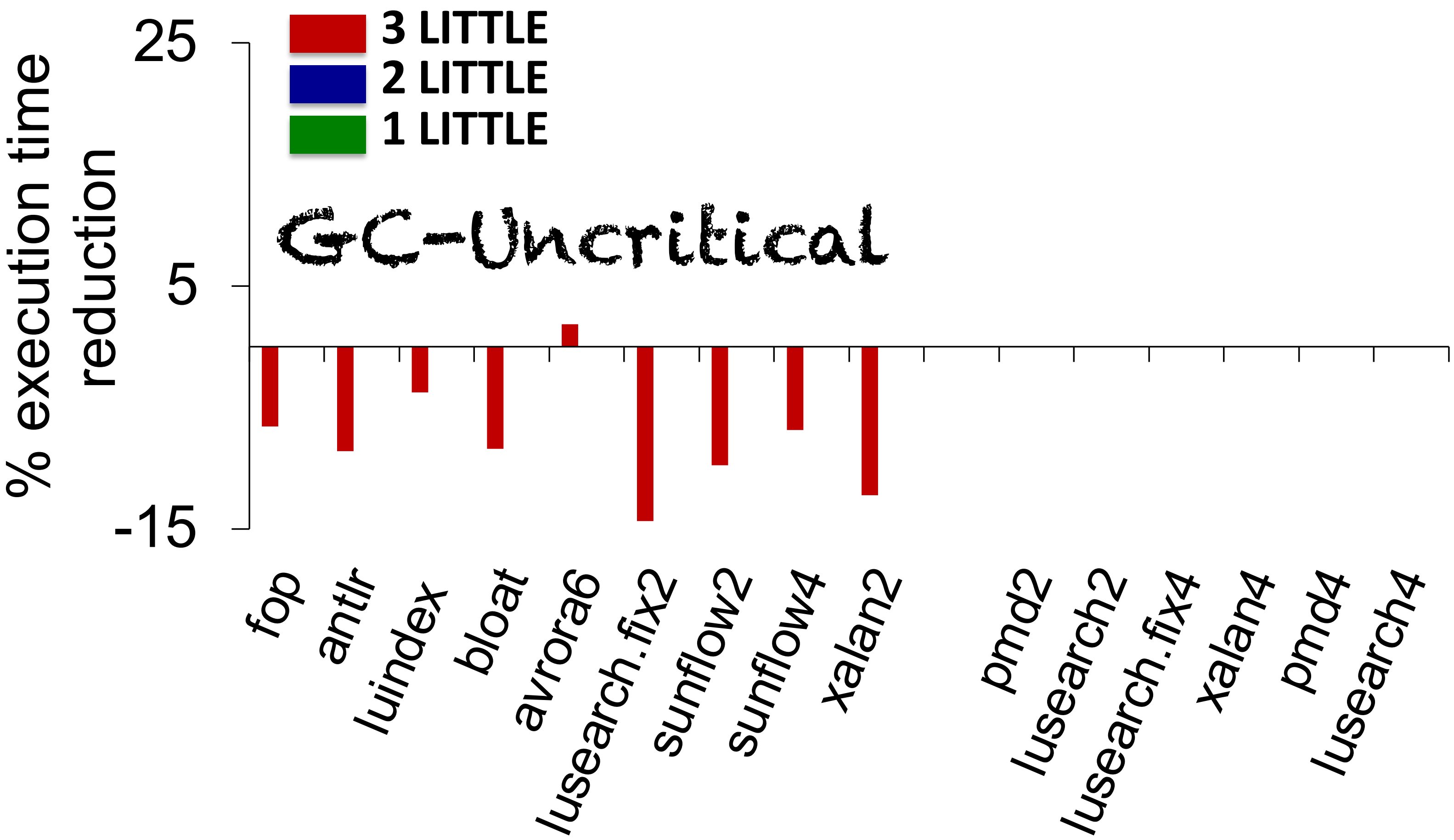
Running GC on LITTLE degrades performance of some applications



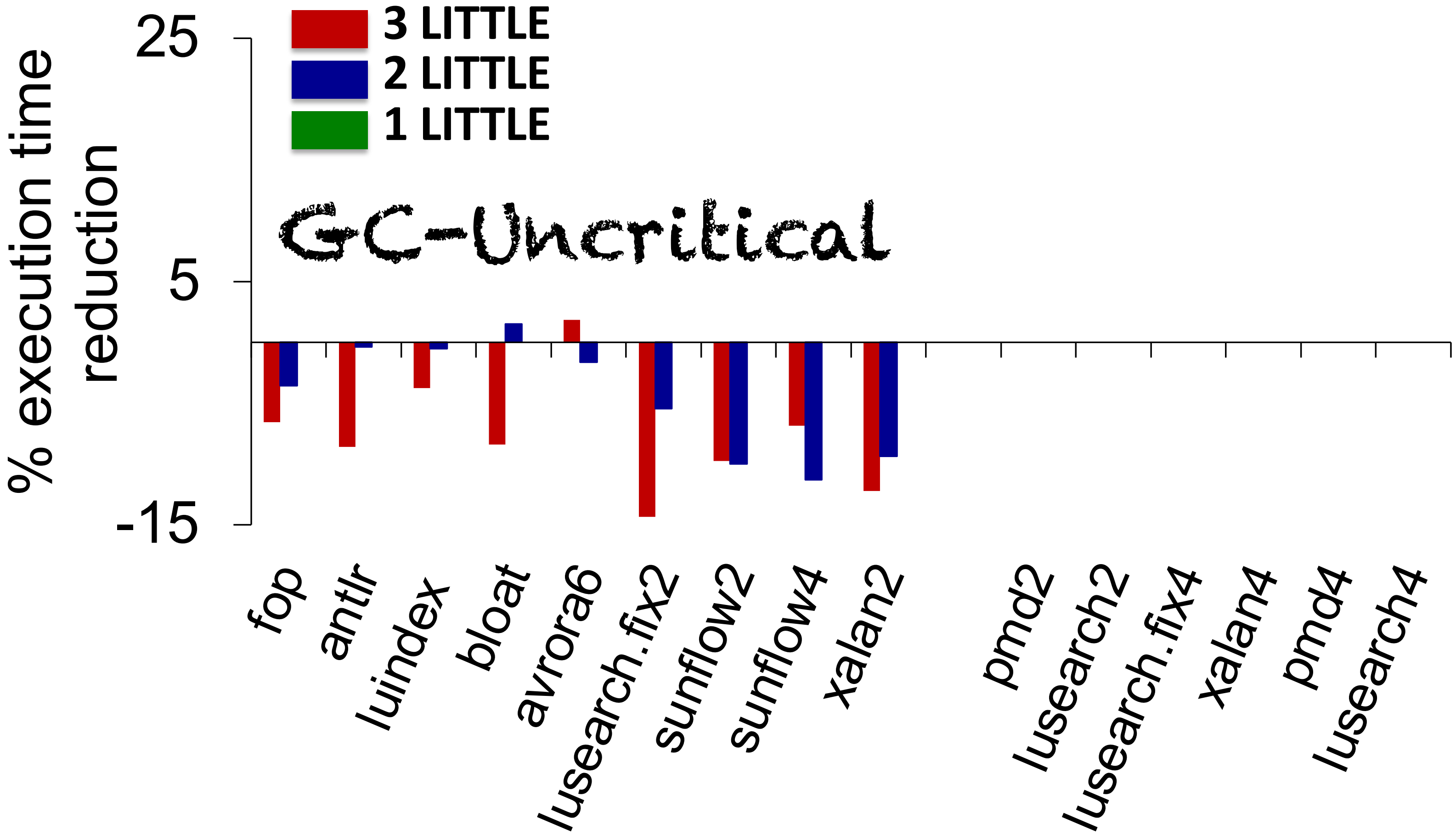
Why does the execution time of GC critical applications increase?



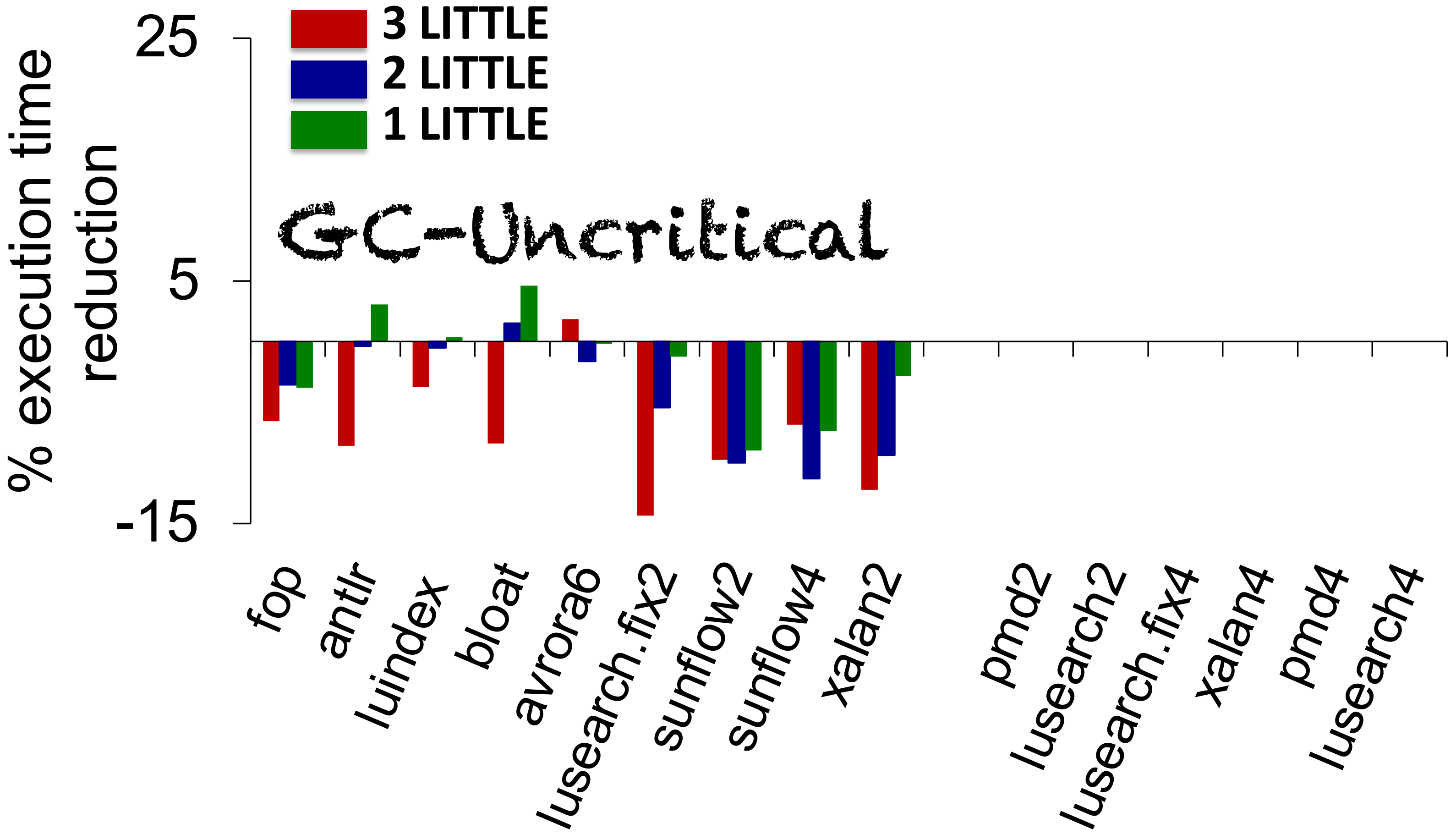
What happens if we give GC a fair share of the big core?



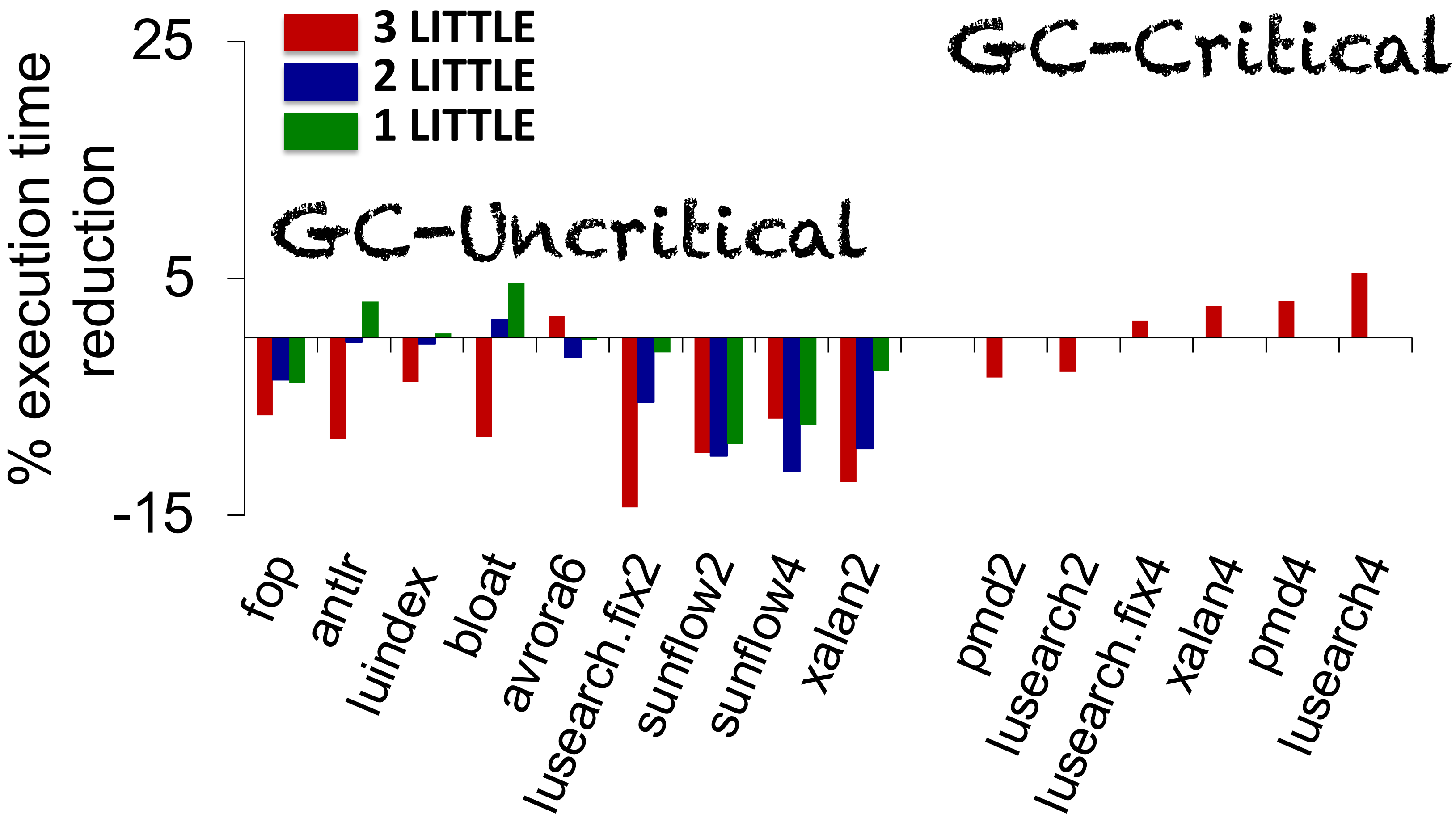
What happens if we give GC a fair share of the big core?



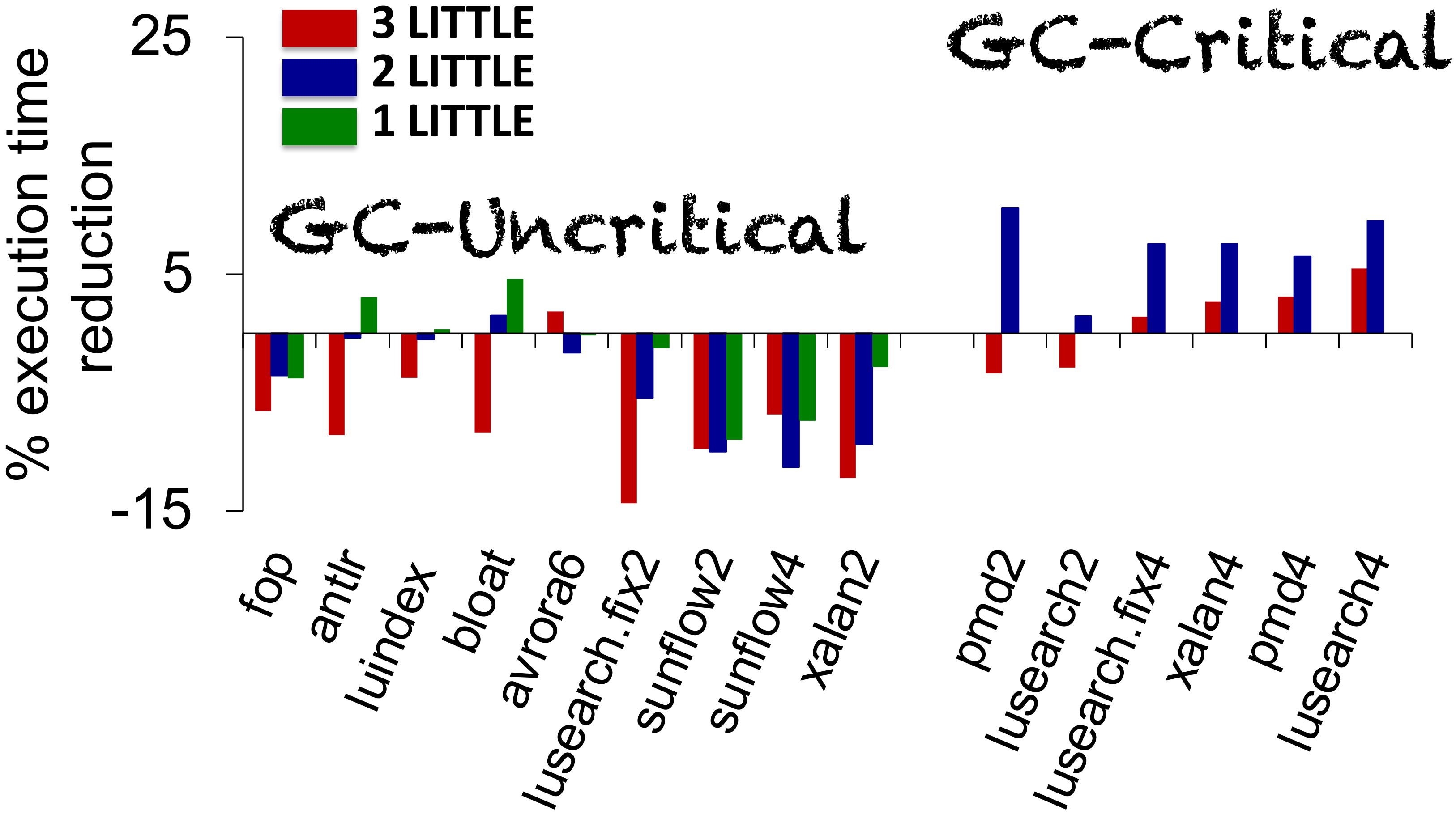
What happens if we give GC a fair share of the big core?



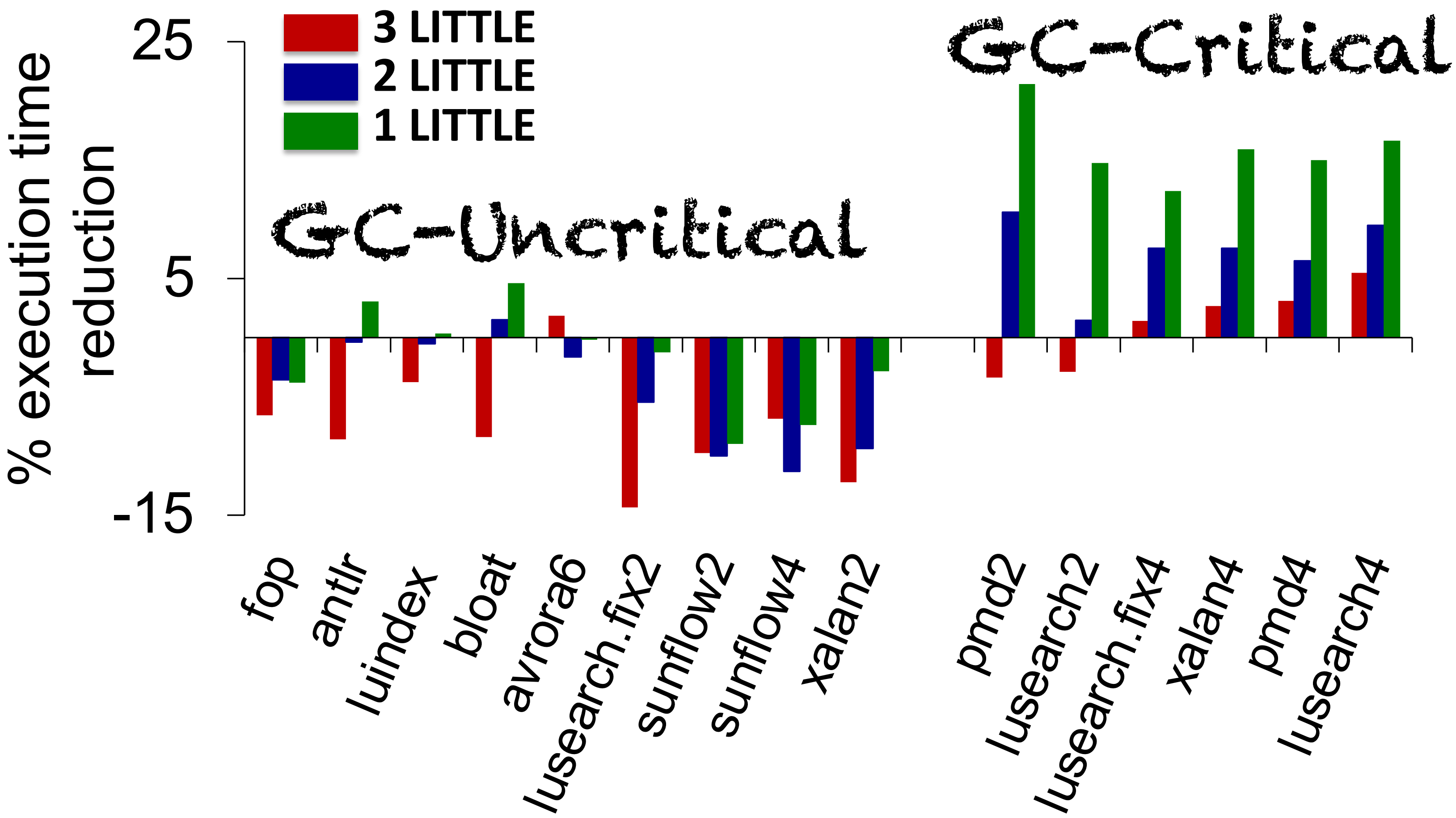
What happens if we give GC a fair share of the big core?



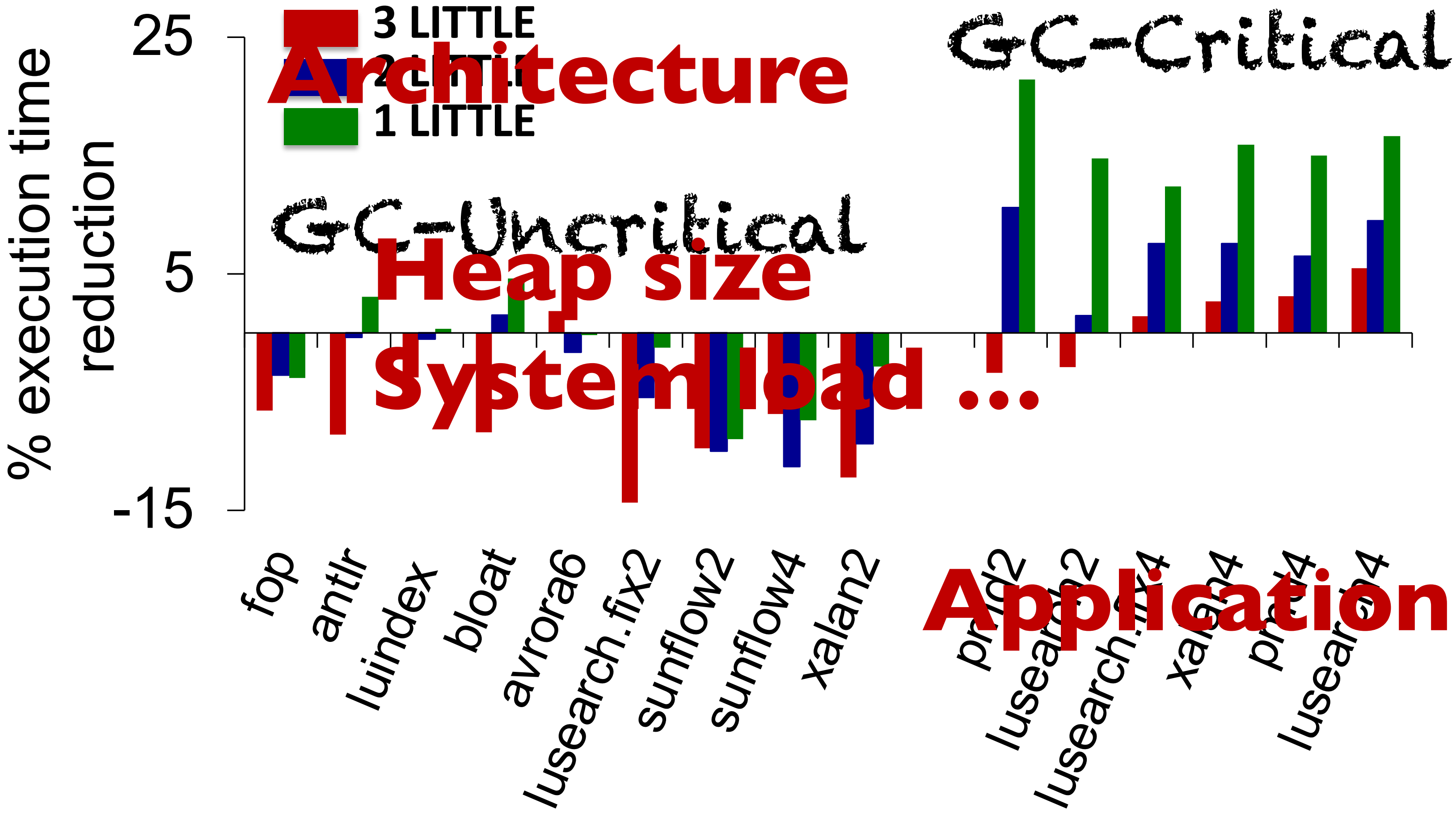
What happens if we give GC a fair share of the big core?



What happens if we give GC a fair share of the big core?



What happens if we give GC a fair share of the big core?

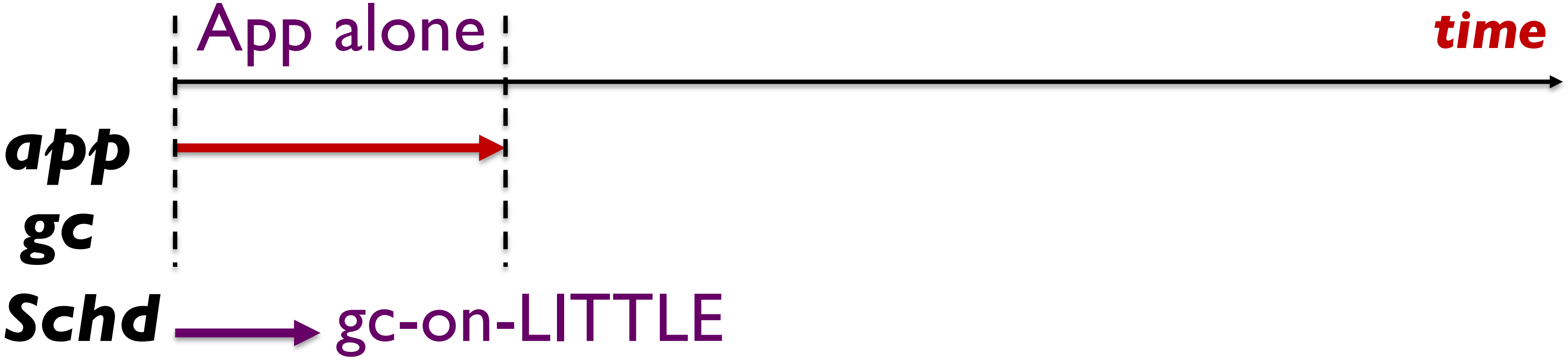


Our Contribution

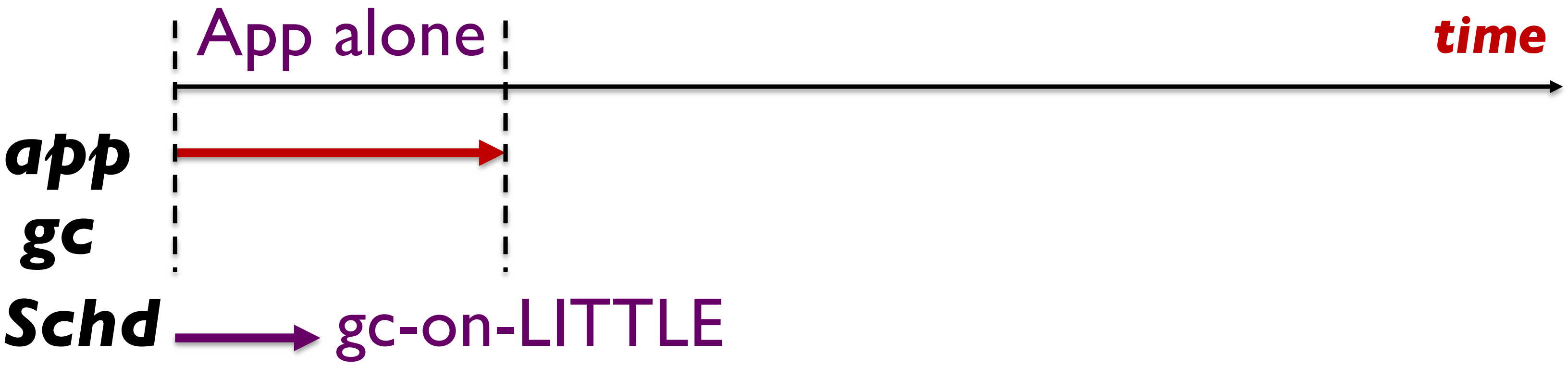
GC-Criticality-Aware Scheduler

Dynamically adjusts # big core cycles given to application versus GC

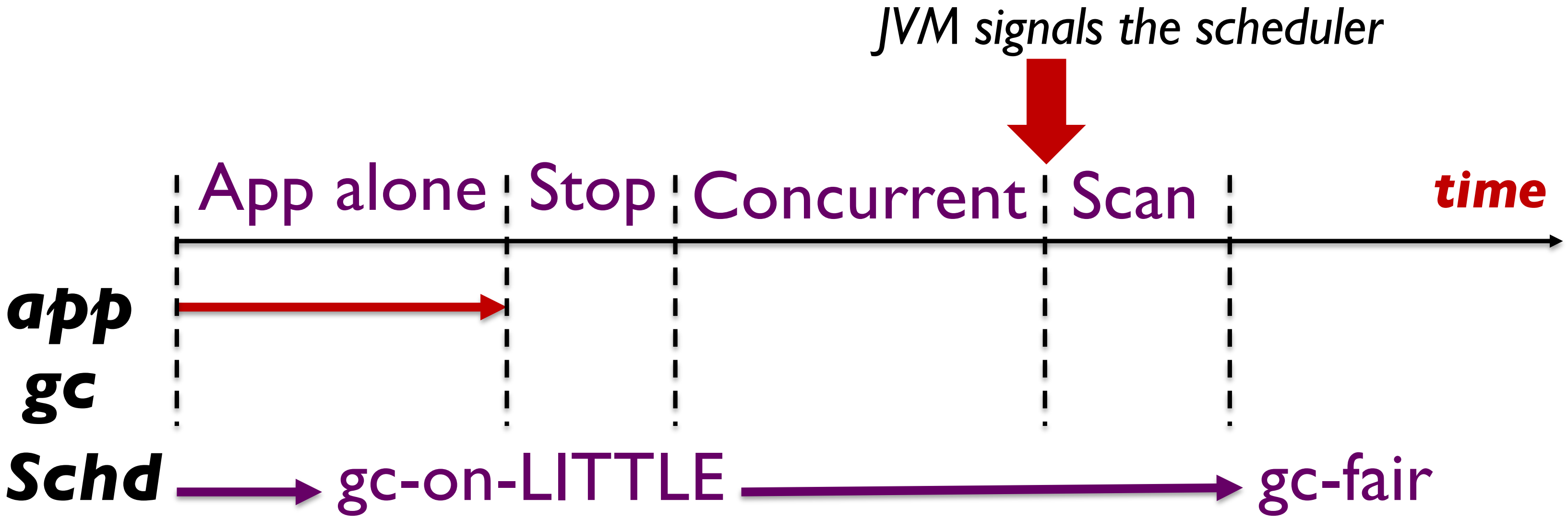
GC-Criticality-Aware Scheduler
Starting point is gc-on-LITTLE



GC-Criticality-Aware Scheduler
gc-on-LITTLE to gc-fair



GC-Criticality-Aware Scheduler gc-on-LITTLE to gc-fair



Stop pause to do book-keeping ignored

Scan stop pause: JVM signals scheduler

gc-fair gives equal priority to GC and app

GC-Criticality-Aware Scheduler

Boost States

Stop scan pauses observed even with gc-fair

Scheduler	How many quanta scheduled on the BIG core?
gc-on-LITTLE	First GC thread = 0, Second GC thread = 0
gc-fair	First GC thread = 1, Second GC thread = 1

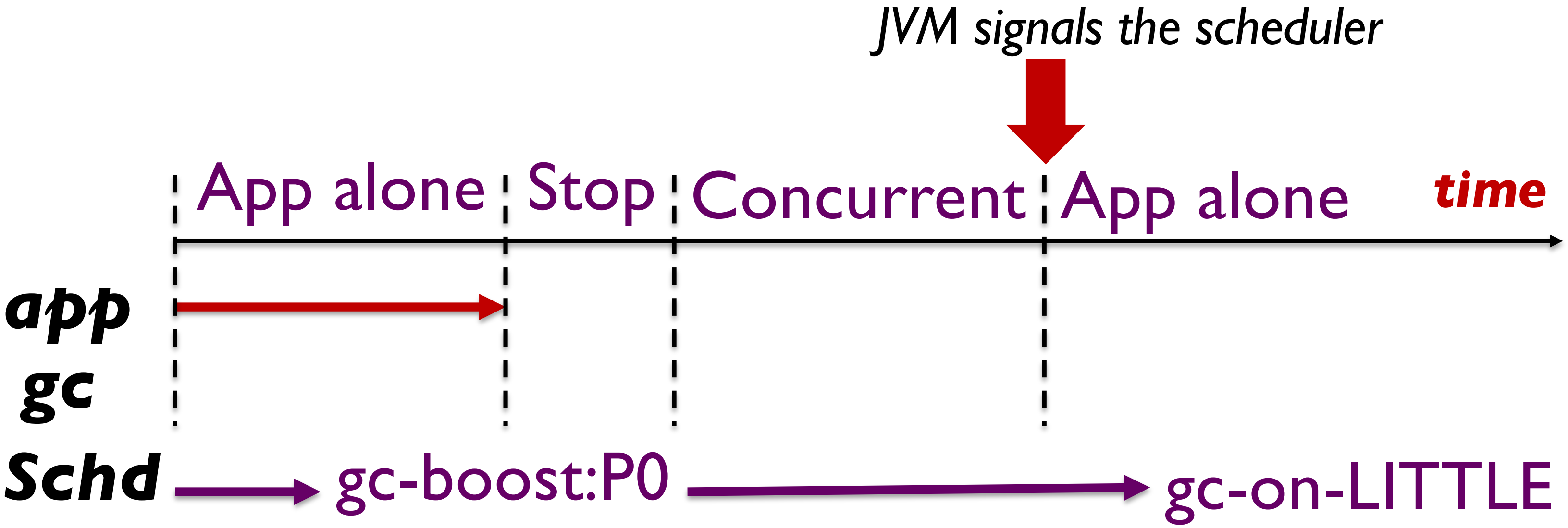
Boost the priority of garbage

Give GC more consecutive quanta on big

Scheduler	State	How many quanta scheduled on the BIG core?
gc-boost	P0	First GC thread = 1, Second GC thread = 1
gc-boost	P1	First GC thread = 1, Second GC thread = 2
	…	

Degrade boost state when no longer critical

GC-Criticality-Aware Scheduler gc-boost:P0 to gc-on-LITTLE



If no scan pause in state P0, go to gc-on-LITTLE

Can configure # zero stop scan intervals before returning to gc-on-LITTLE

Summary of gc-criticality-aware scheduling

1. JVM detects GC Criticality during execution
2. JVM communicates gc *criticality* to the scheduler
3. Scheduler adapts # big core cycles given to GC

Experimental Setup

the DaCapo benchmark suite

Jikes Research virtual machine

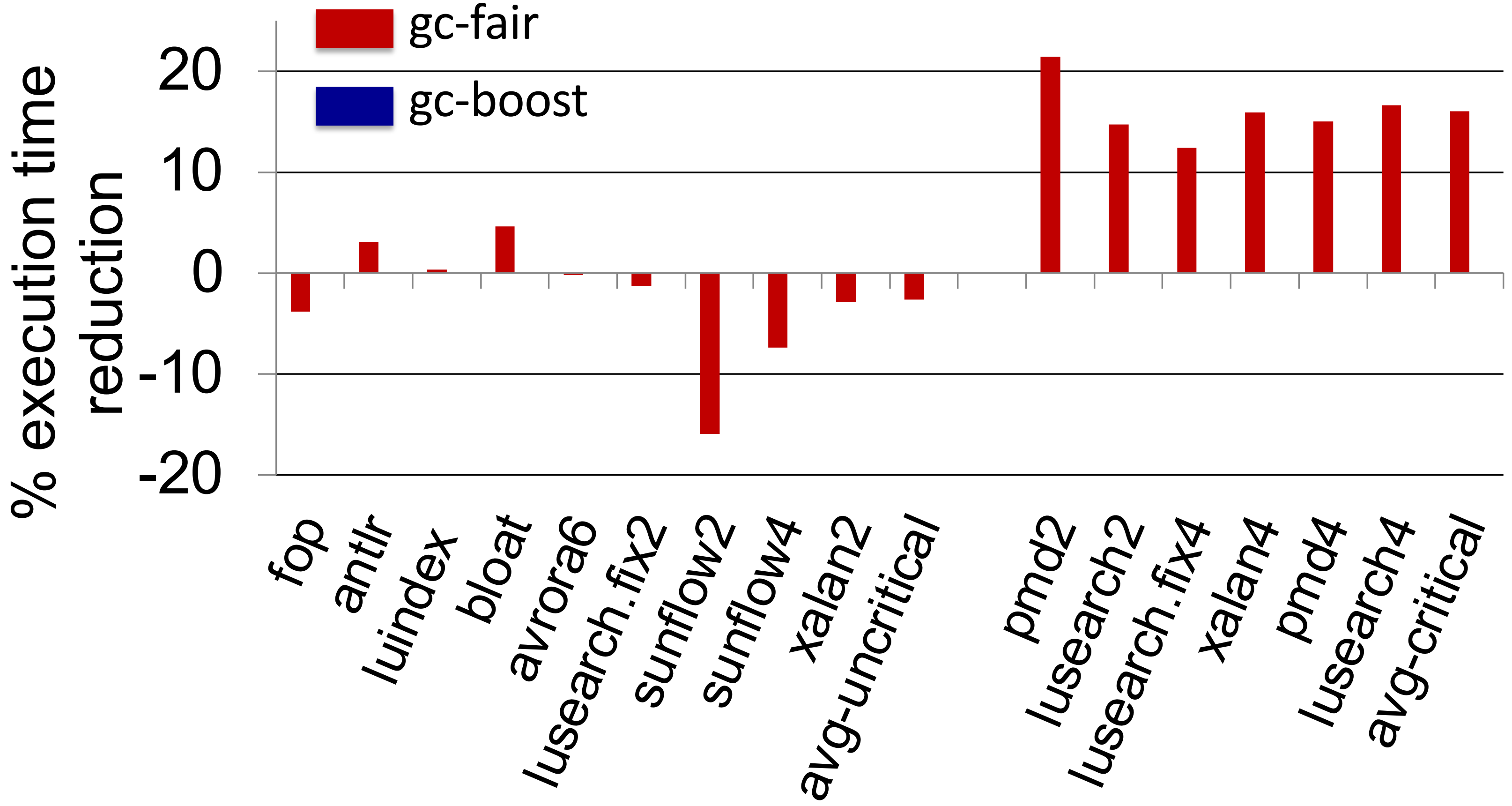
sniper

✧ How to tackle non-determinism?

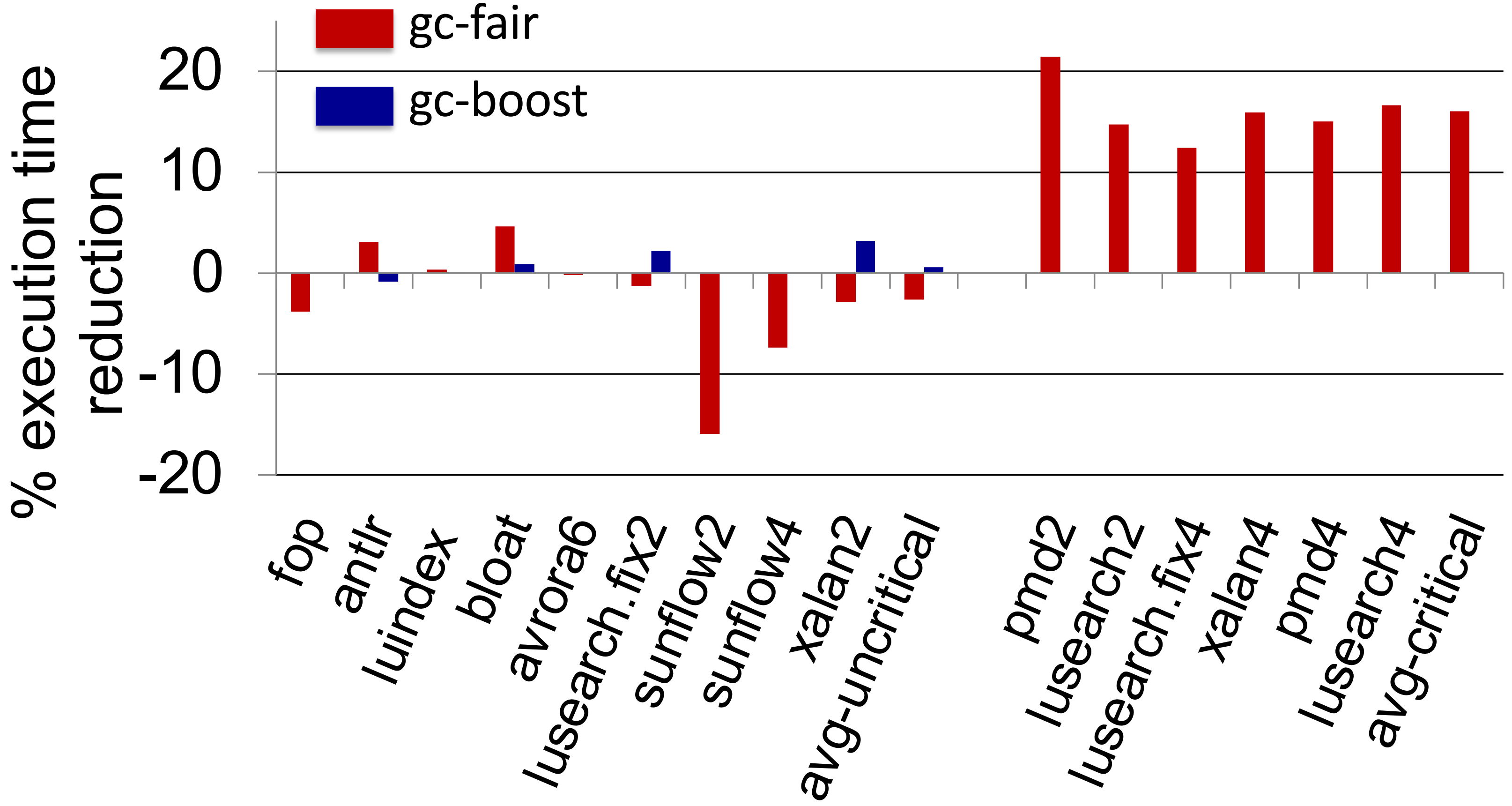
✧ CMS with heap 2x of minimum

✧ Model different architectures

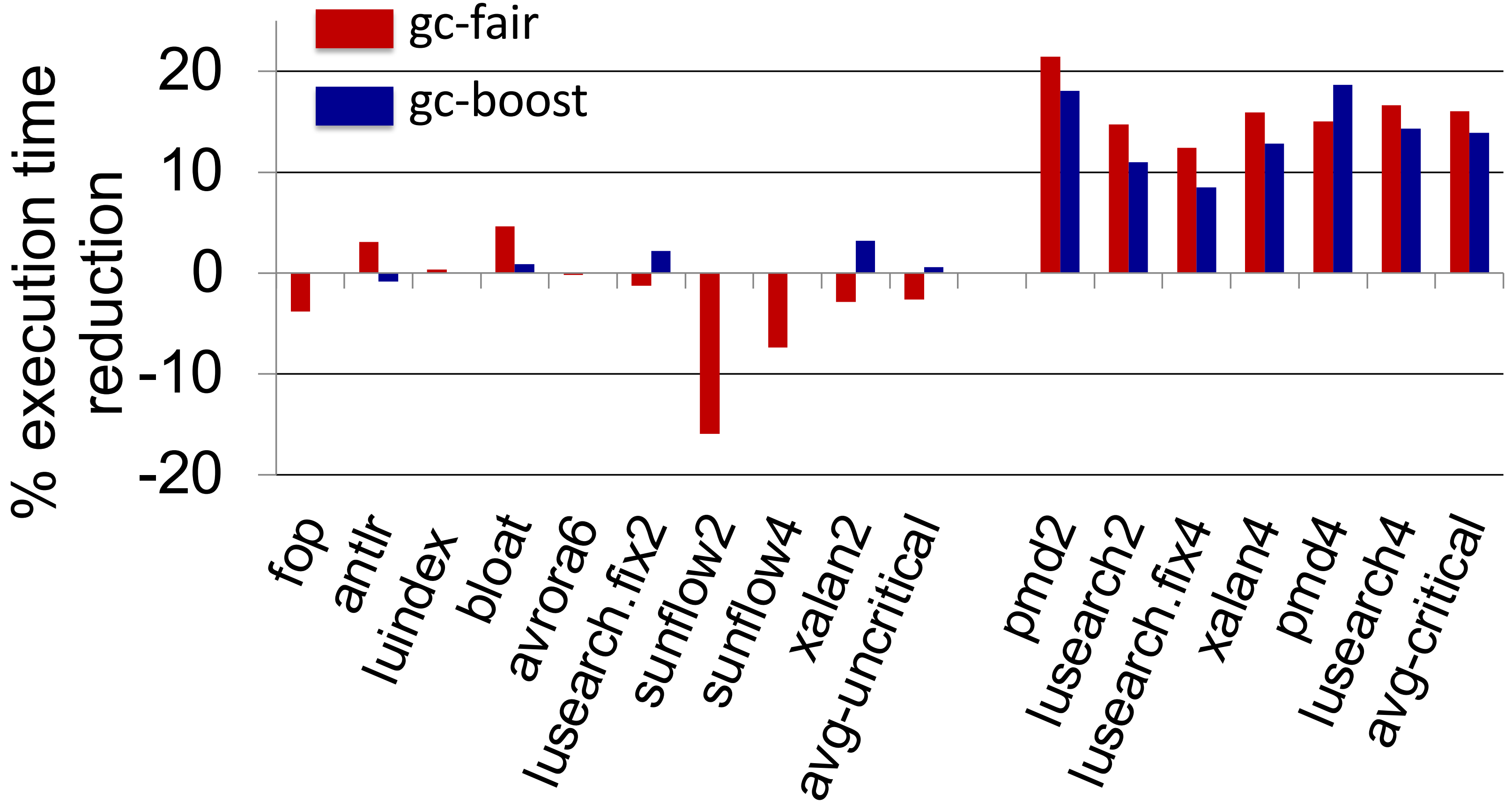
GC-Criticality-Aware scheduler is better performing vis-à-vis gc-fair



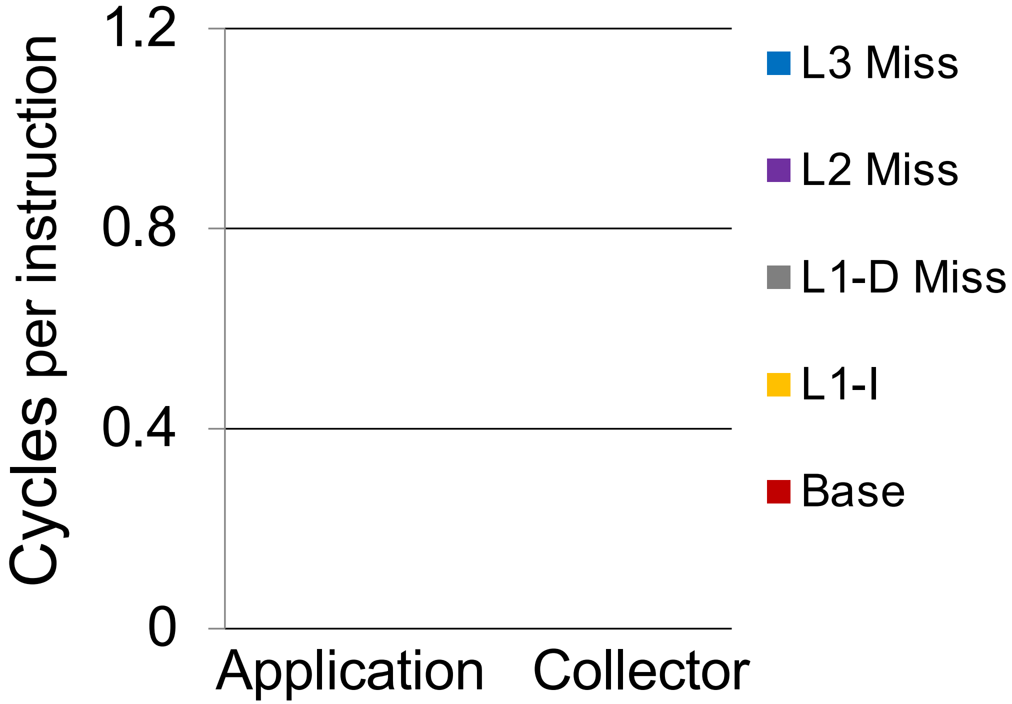
GC-Criticality-Aware scheduler is better performing vis-à-vis gc-fair



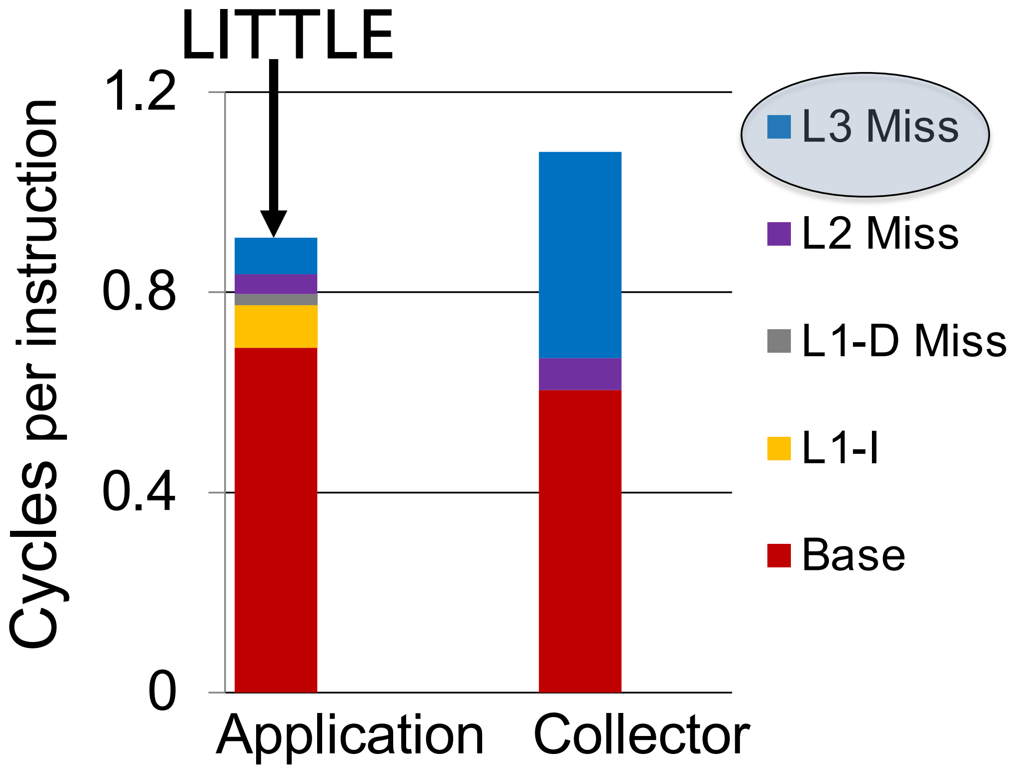
GC-Criticality-Aware scheduler is better performing vis-à-vis gc-fair



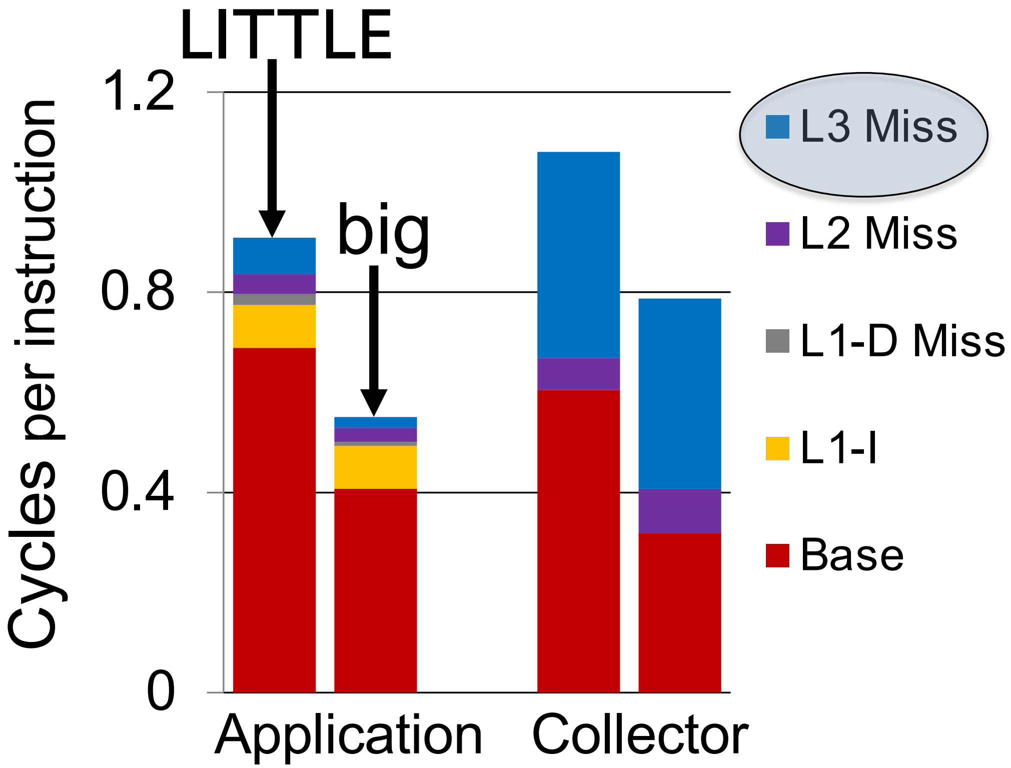
Where does the performance advantage of big core comes from?



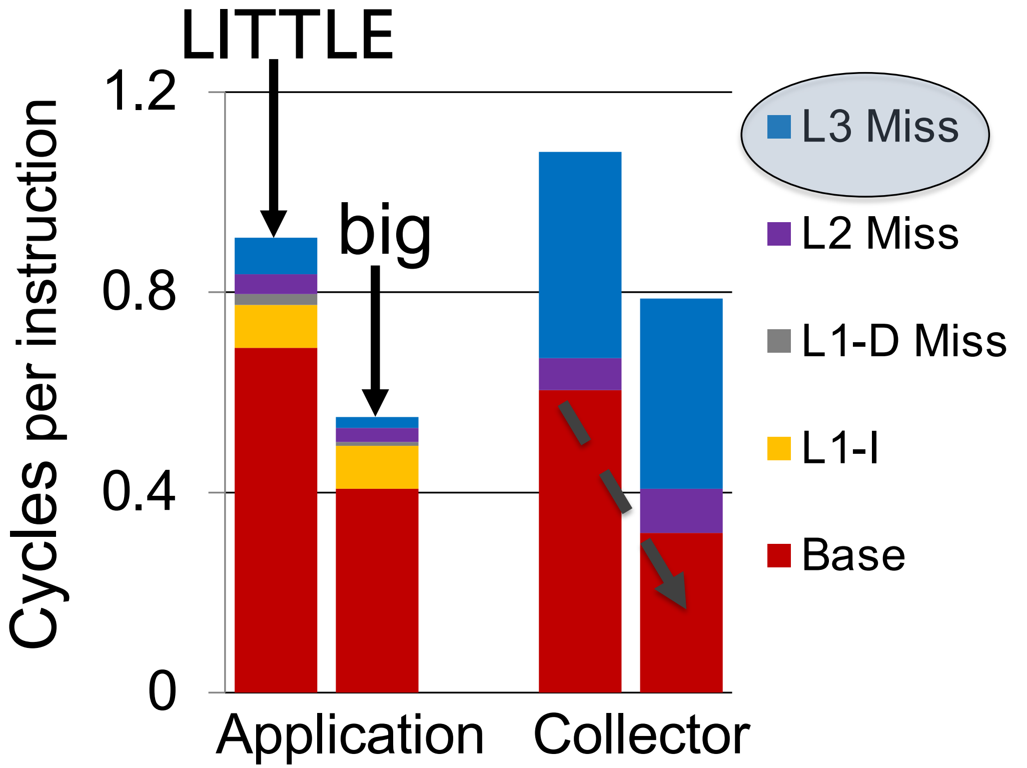
Where does the performance advantage of big core comes from?



Where does the performance advantage of big core comes from?



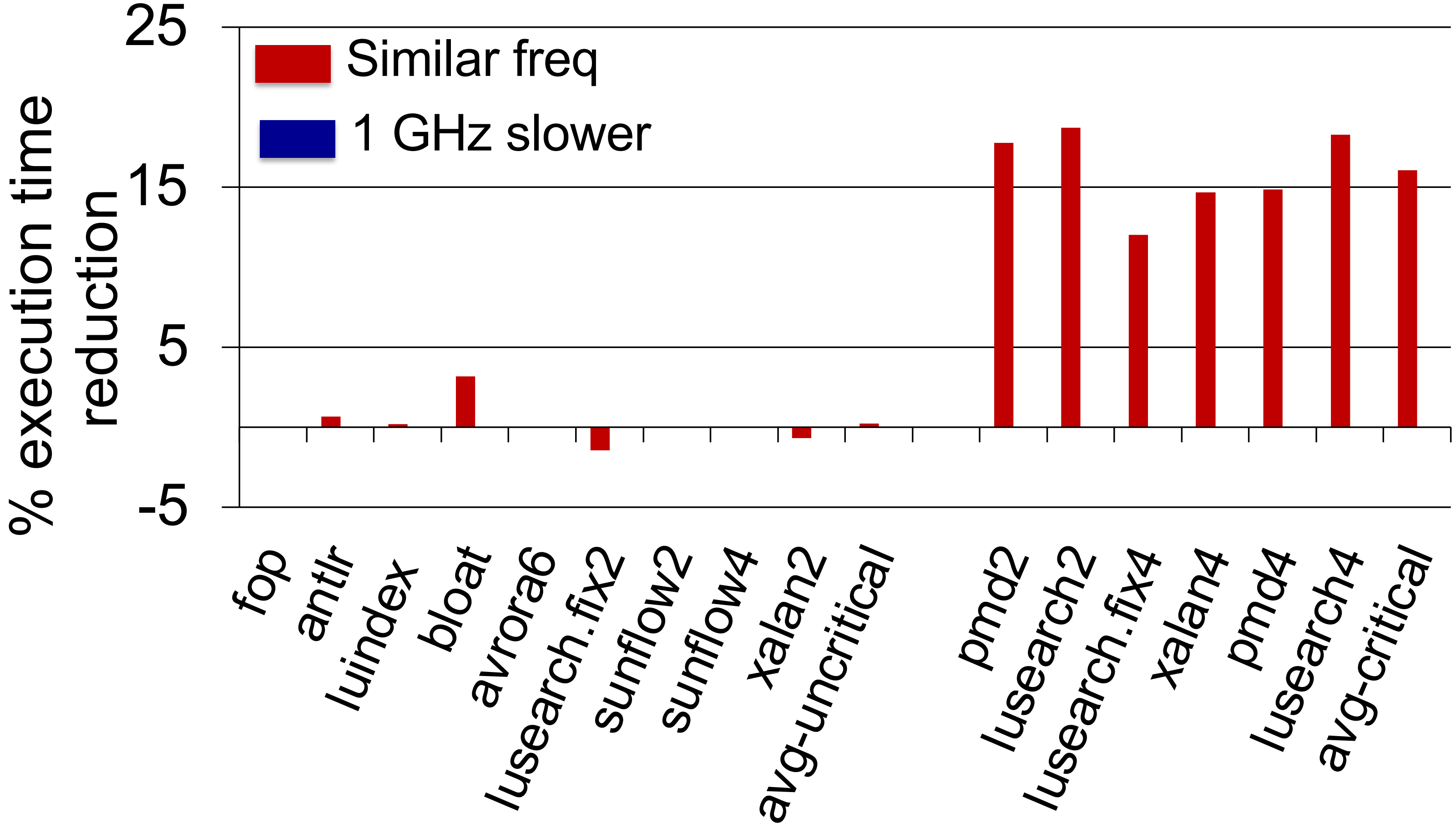
Where does the performance advantage of big core comes from?



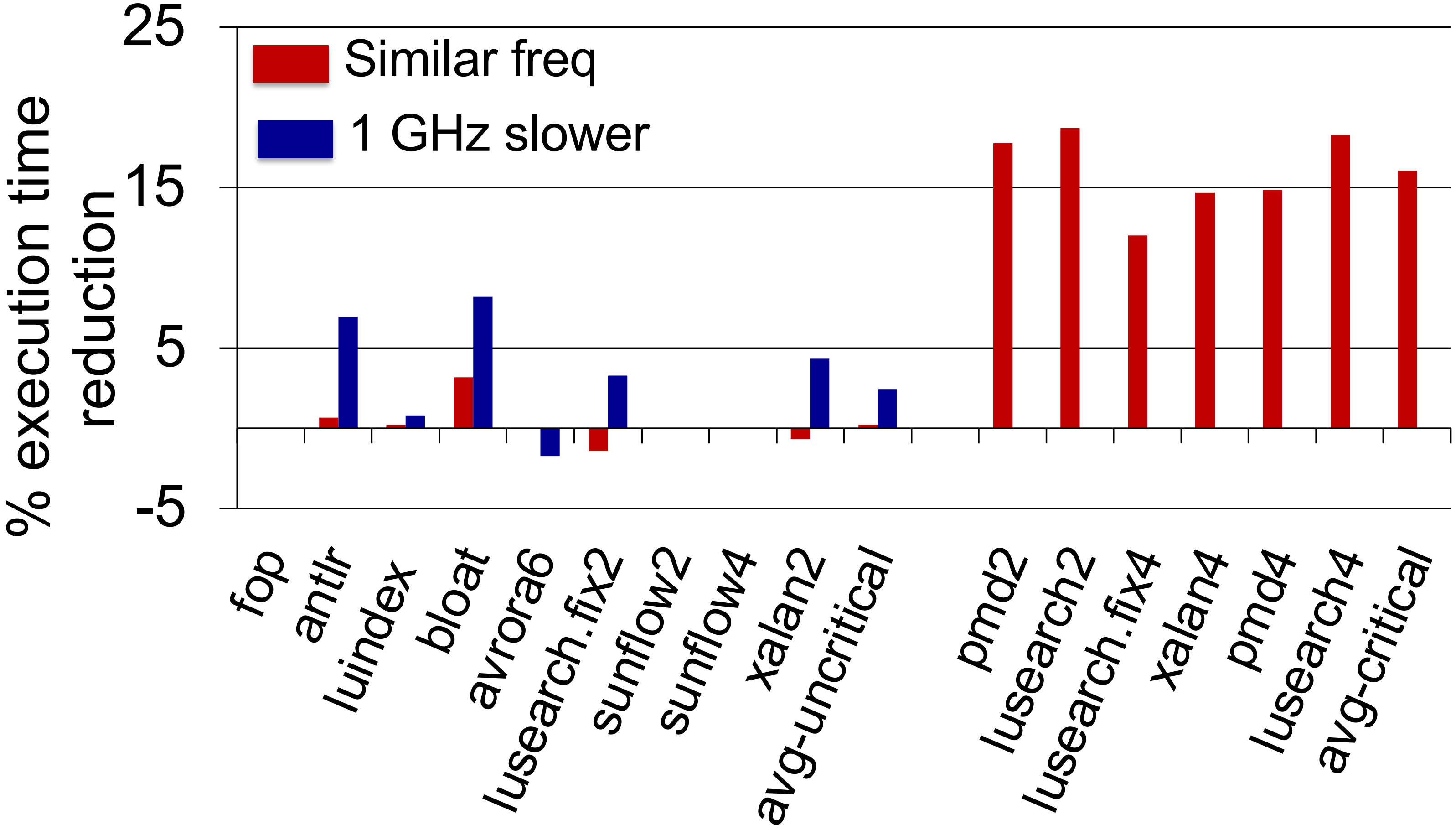
Instruction-level parallelism ☺

Memory-level parallelism ☹

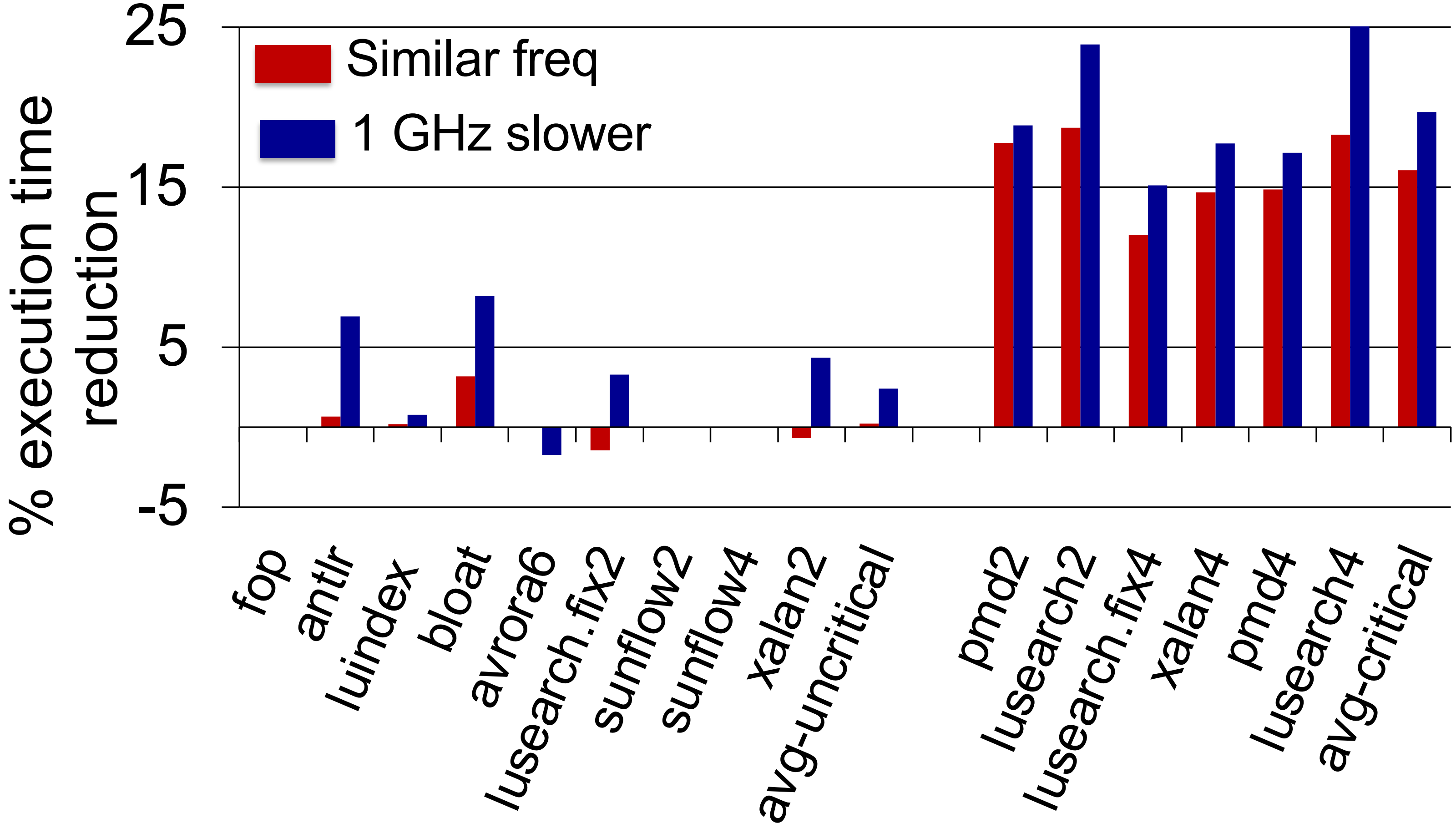
Lowering the frequency of LITTLE core makes GC even more critical



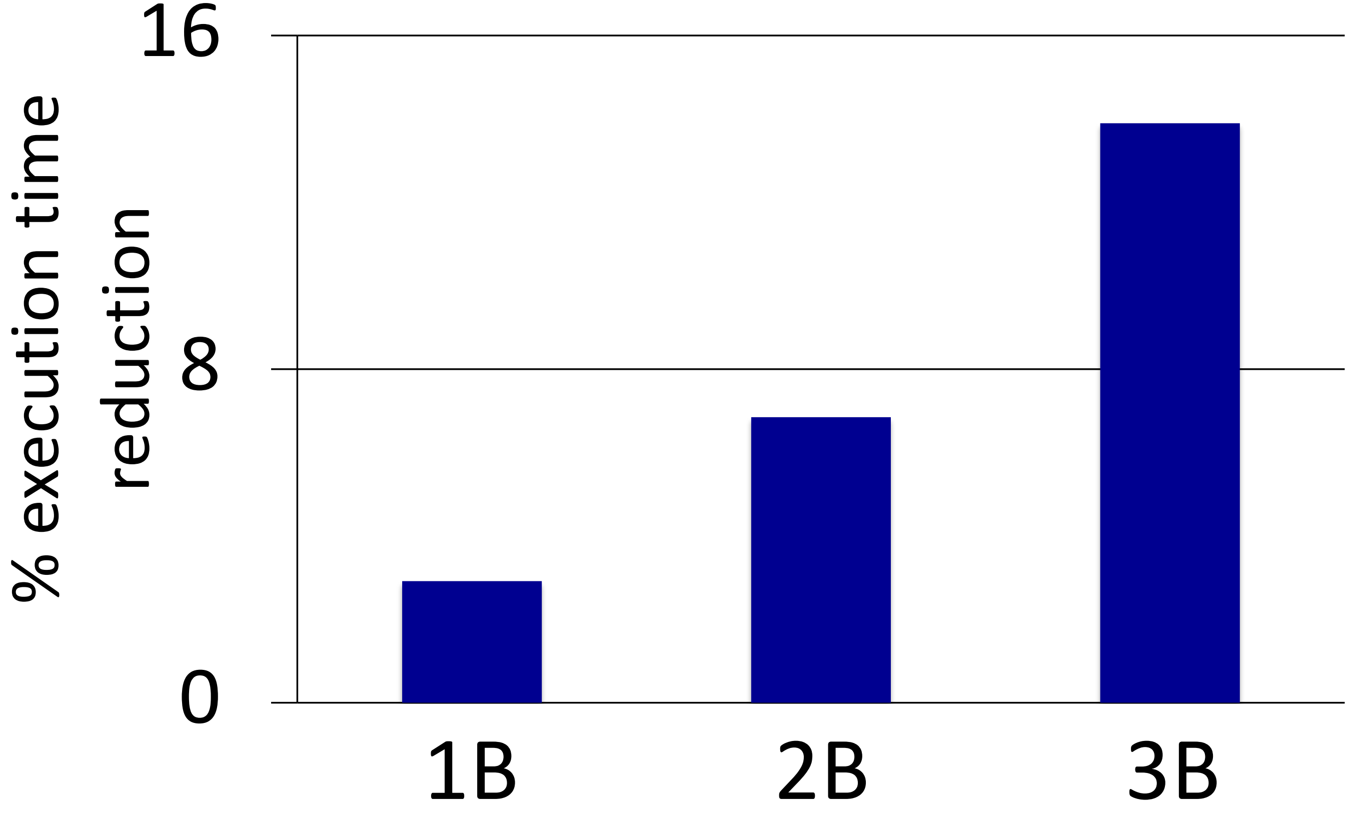
Lowering the frequency of LITTLE core makes GC even more critical



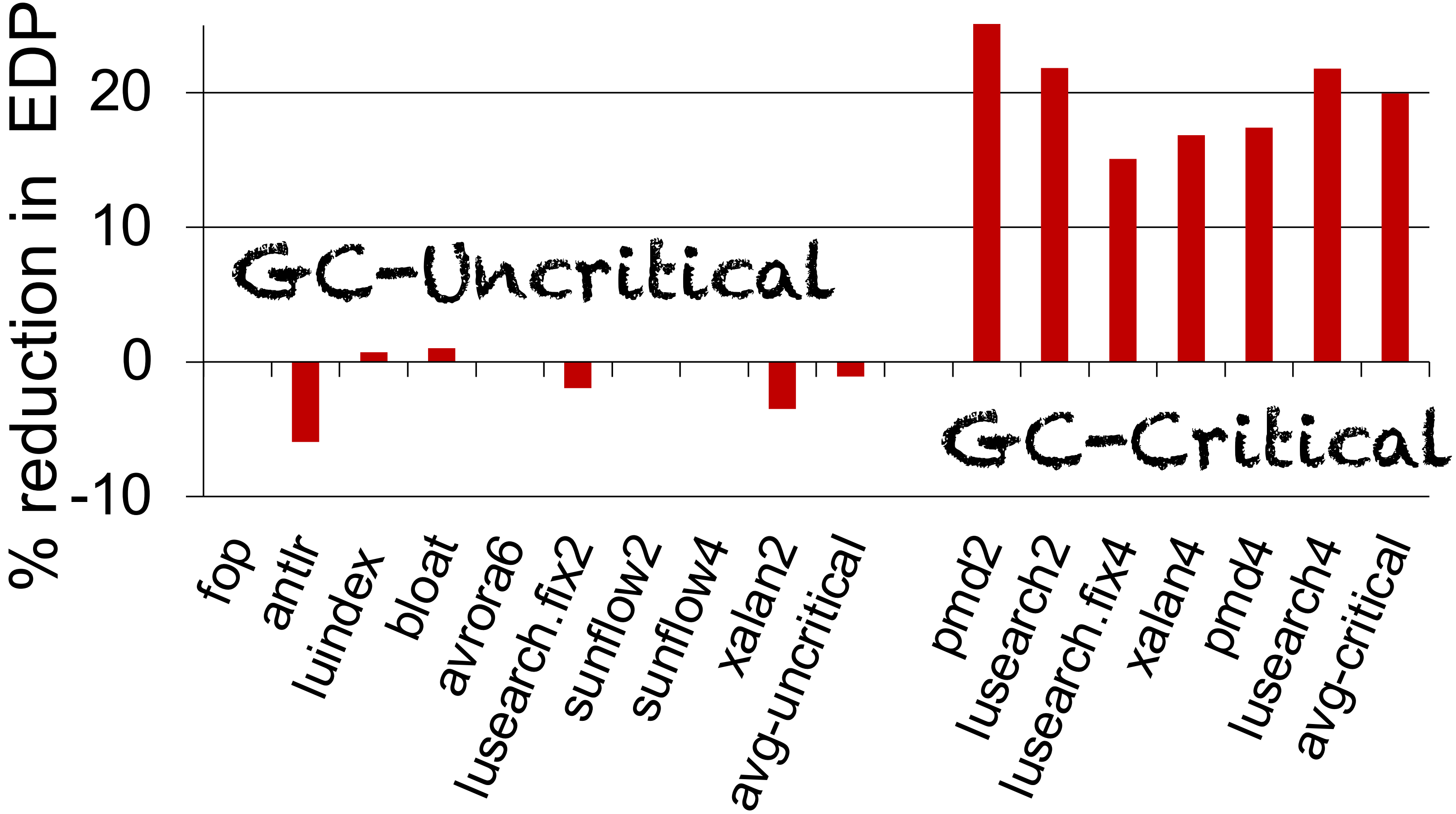
Lowering the frequency of LITTLE core makes GC even more critical



gc-boost provides greater gains for architectures with more big cores



Average EDP reduction of 20% for GC-Critical applications

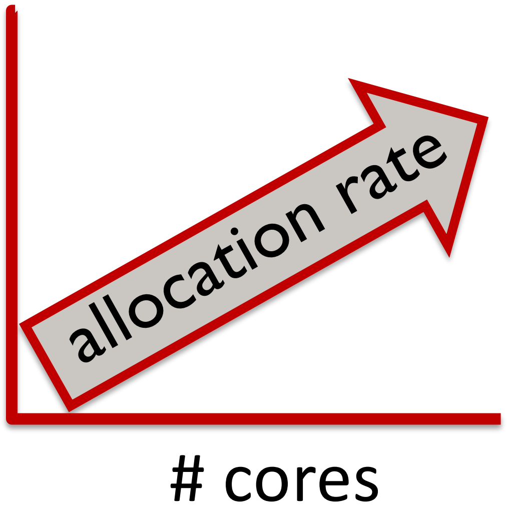


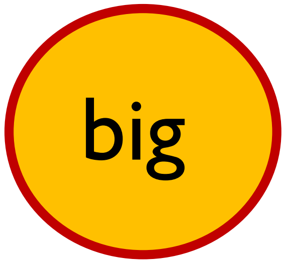
A few takeaway messages

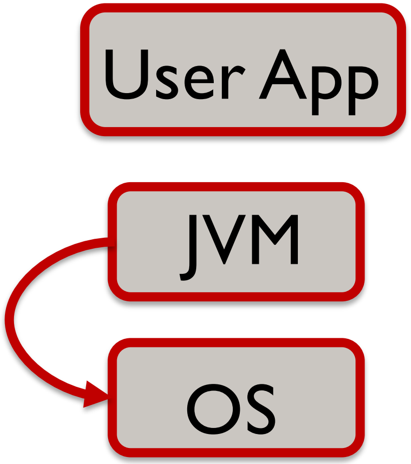
(1) Multithreaded applications could be GC critical

(2) GC benefits from big core features

(3) JVM support for scheduling GC improves efficiency


allocation rate
# cores




User App
JVM
OS