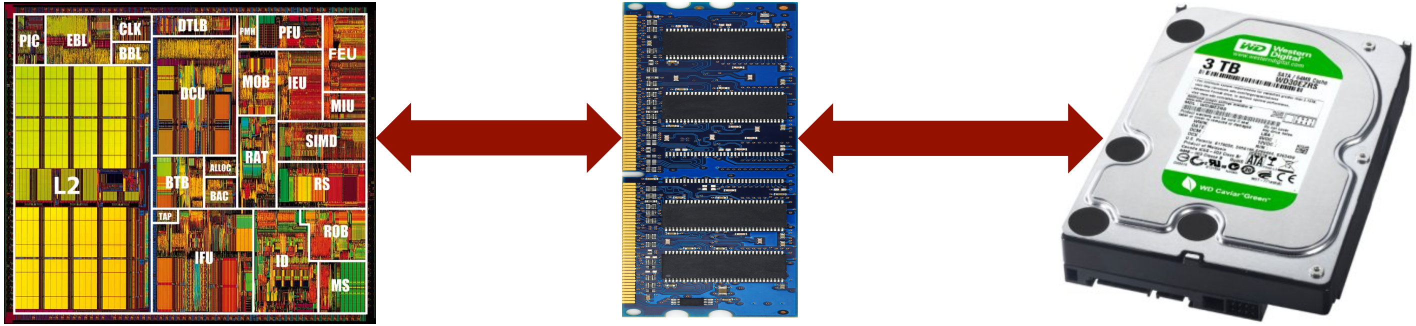
To expose, or not to expose, hardware heterogeneity to runtimes

Shoaib Akram

Ghent University, Belgium

Shoaib.Akram@UGent.be

Circa 2000: Hardware fixed at design



Out-of-order 1 GHz processor

Volatile DRAM main memory

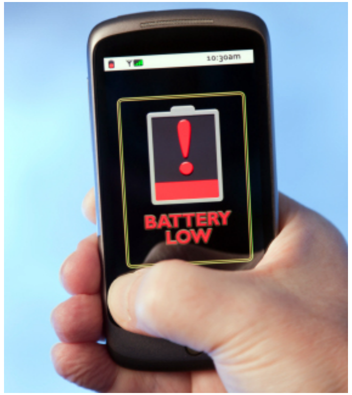
Persistent disk storage

The shift to power-aware computing

Dennard scaling has stopped

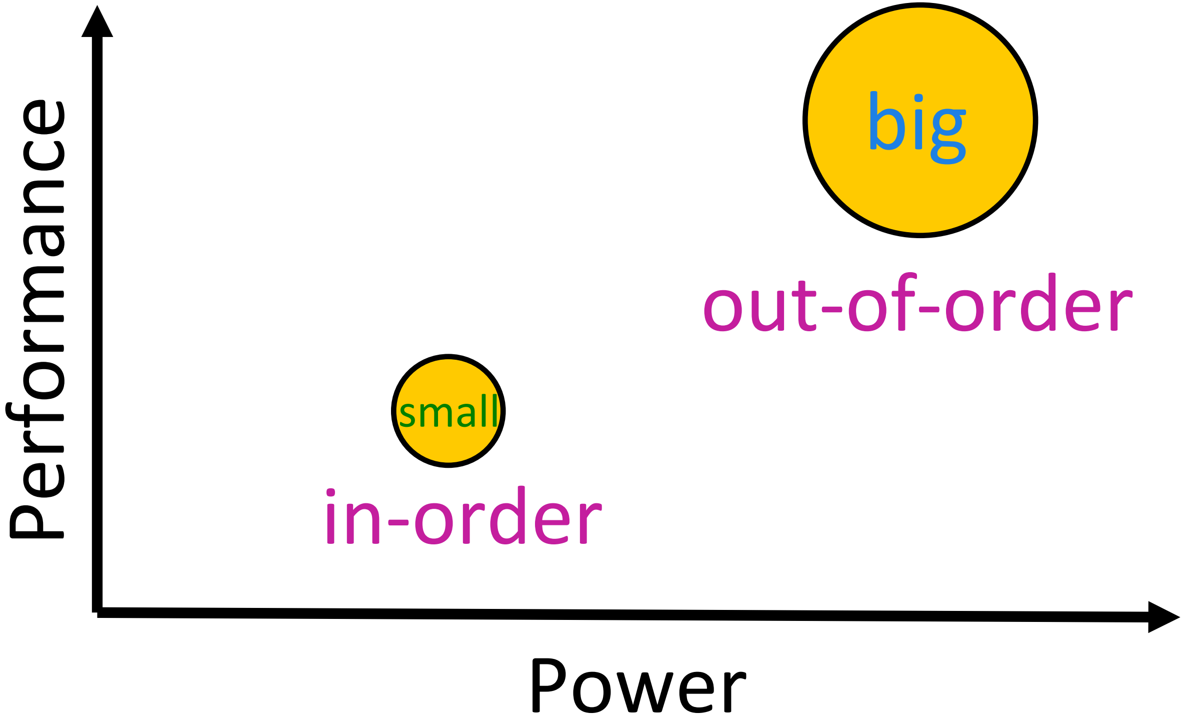
Constant power density as transistor size shrinks

Reliance on battery-operated devices

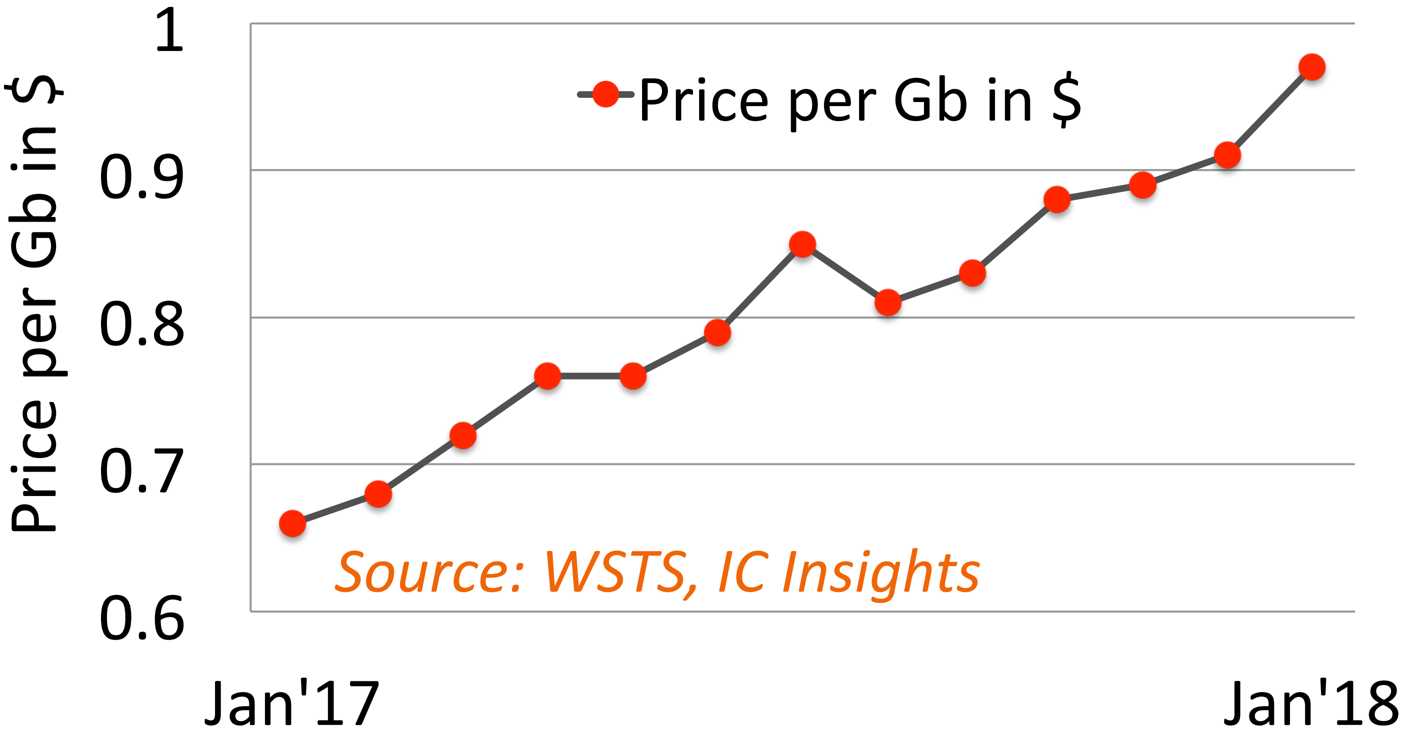


Heterogeneous Multicores

Each core has its own frequency domain (DVFS)



DRAM price and supply trends

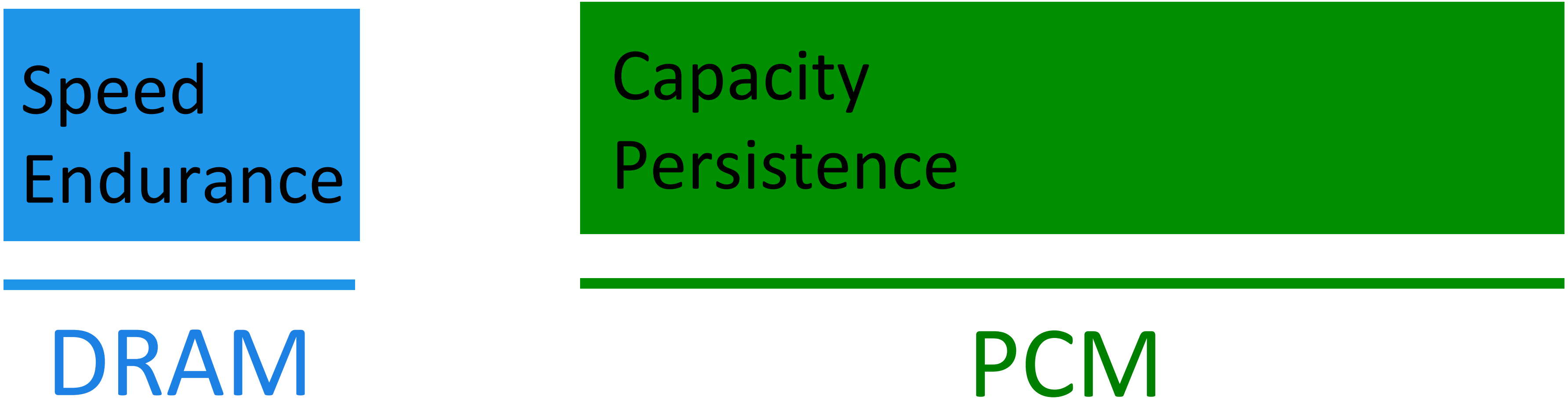


Source: WSTS, IC Insights

Hybrid DRAM-PCM memory

☺ More GB/$ with Phase Change Memory

☹ Higher latency *and* low endurance



Speed
Endurance

DRAM

Capacity
Persistence

PCM

Some challenges of heterogeneity

Schedule threads on big-small cores

Regulate DVFS

Mitigate PCM wear-out

Bridge DRAM-PCM latency-gap

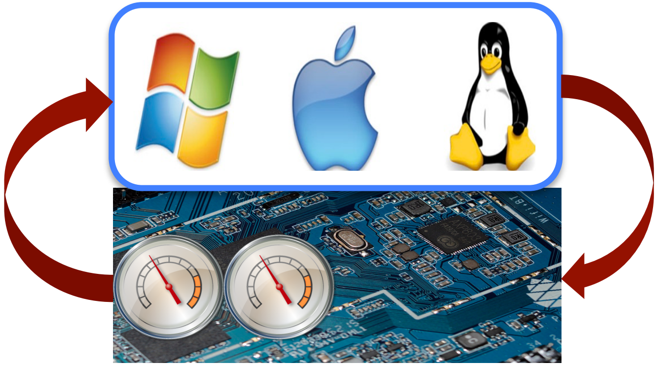
…

Hardware/OS only approaches

Hardware exposes counters

OS predicts how software behaves

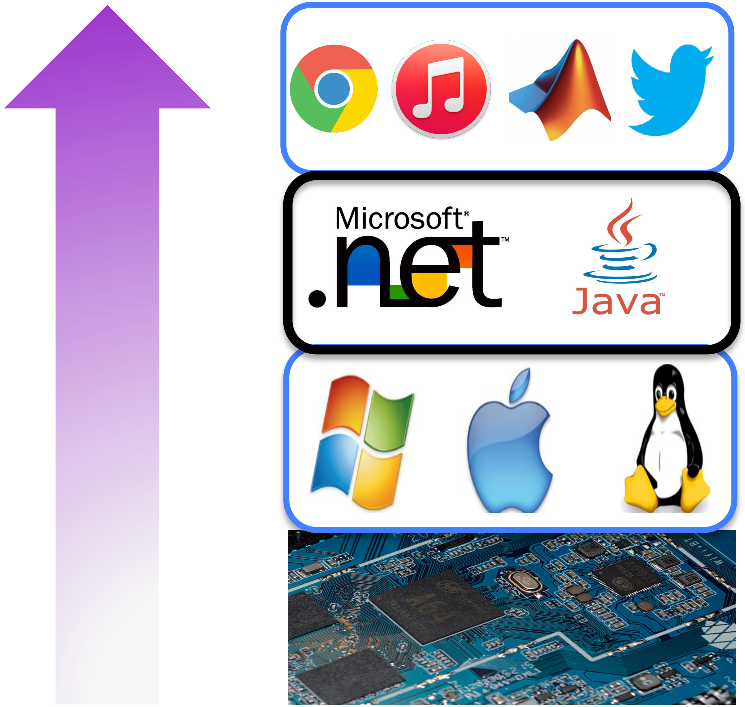
OS configures knobs and manages heterogeneity



Pros/cons of moving up the stack

Gain in semantic knowledge ☺

Loss in abstraction ☹



Exposing heterogeneity to runtimes

☺ Proactive

Thread $\mathcal{X}$ chases pointers (memory-bound)

$\mathcal{X}$→$\mathcal{Y}$ is producer→consumer

Memory region $\mathcal{X}$ is highly written

☺ Flexible granularity

Memory mgmt in OS fixed at page granularity

Exposing heterogeneity to runtimes

☹ Dependency

Hardware vendor relies on Microsoft/Oracle etc

OS is ubiquitous

☹ Software complexity

Gain insight into software behavior

Design, code, verify

☹ Native applications

C has a non-negligible fan base

Beyond heterogeneity

Hardware multithreading

Turbo boost

Prefetching

Variable page sizes

Cache and memory-bandwidth partitioning

Accelerators and FPGAs

3D Stacked memory

Outline

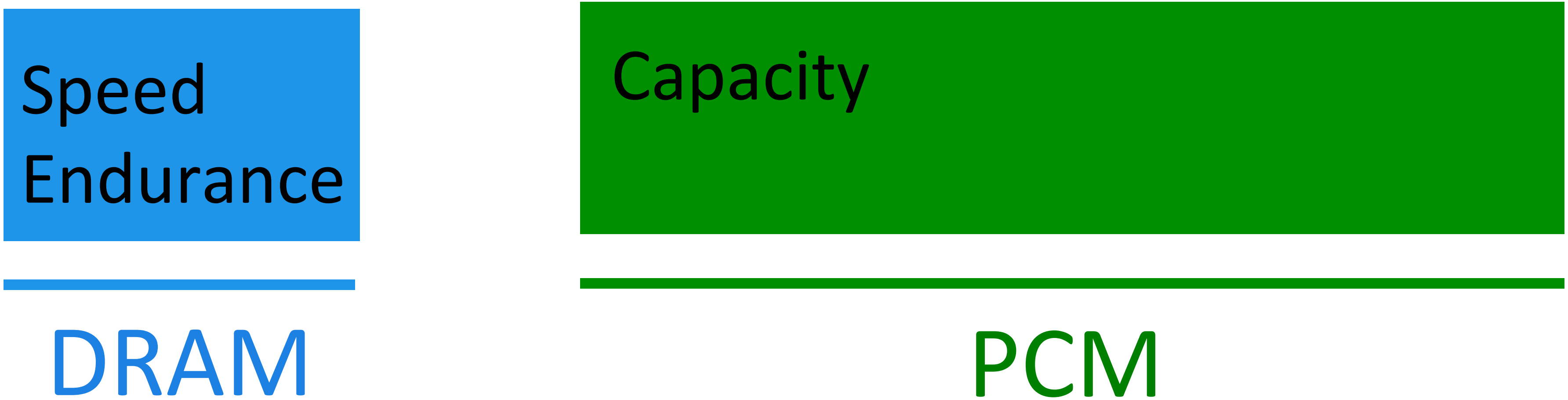
Garbage collection for hybrid memories

Concurrent collection on heterogeneous multicores

Managing DRAM-PCM memory

Mitigate PCM wear-out ✔

Bridge the DRAM-PCM latency gap



Speed
Endurance

Capacity

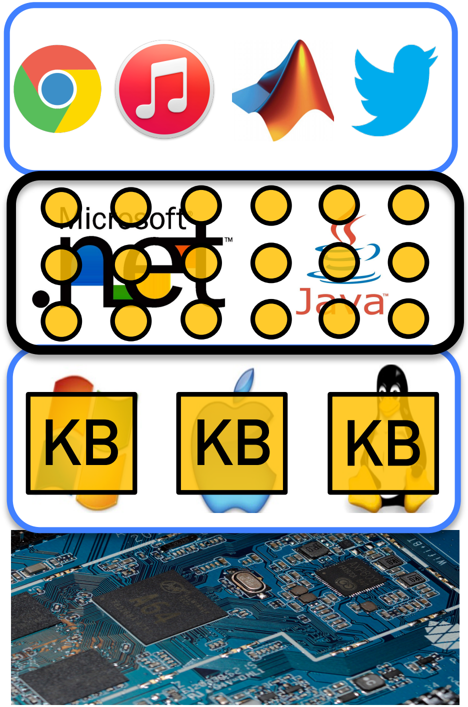
DRAM

PCM

Managing DRAM-PCM memory

PLDI Philadelphia 2018

Write-Rationing Garbage Collection for Hybrid Memory



Operating System

Coarse-grained

pages KB

Garbage collection

Proactive ☺

Fine-grained

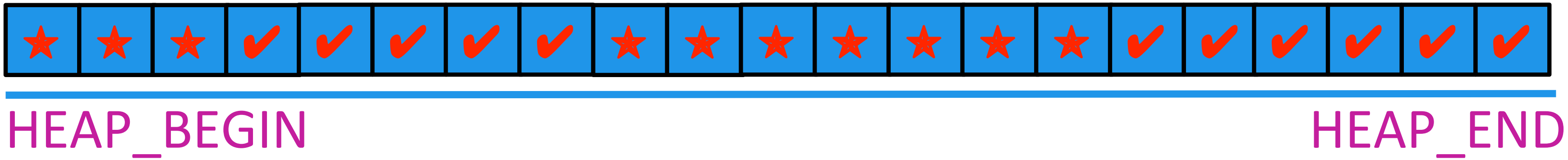
objects ● ● ● ●

GC manages DRAM-PCM hybrid better than OS

DRAM heap management

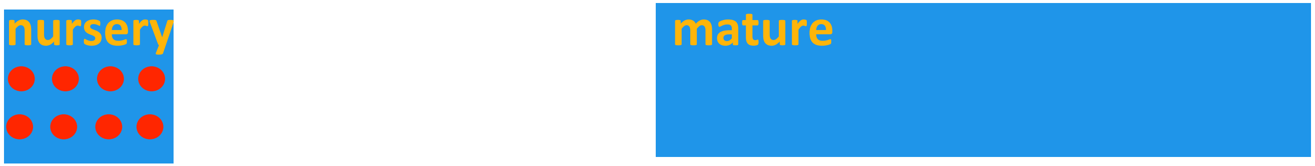
Heap Tracker

✔ available ★ occupied

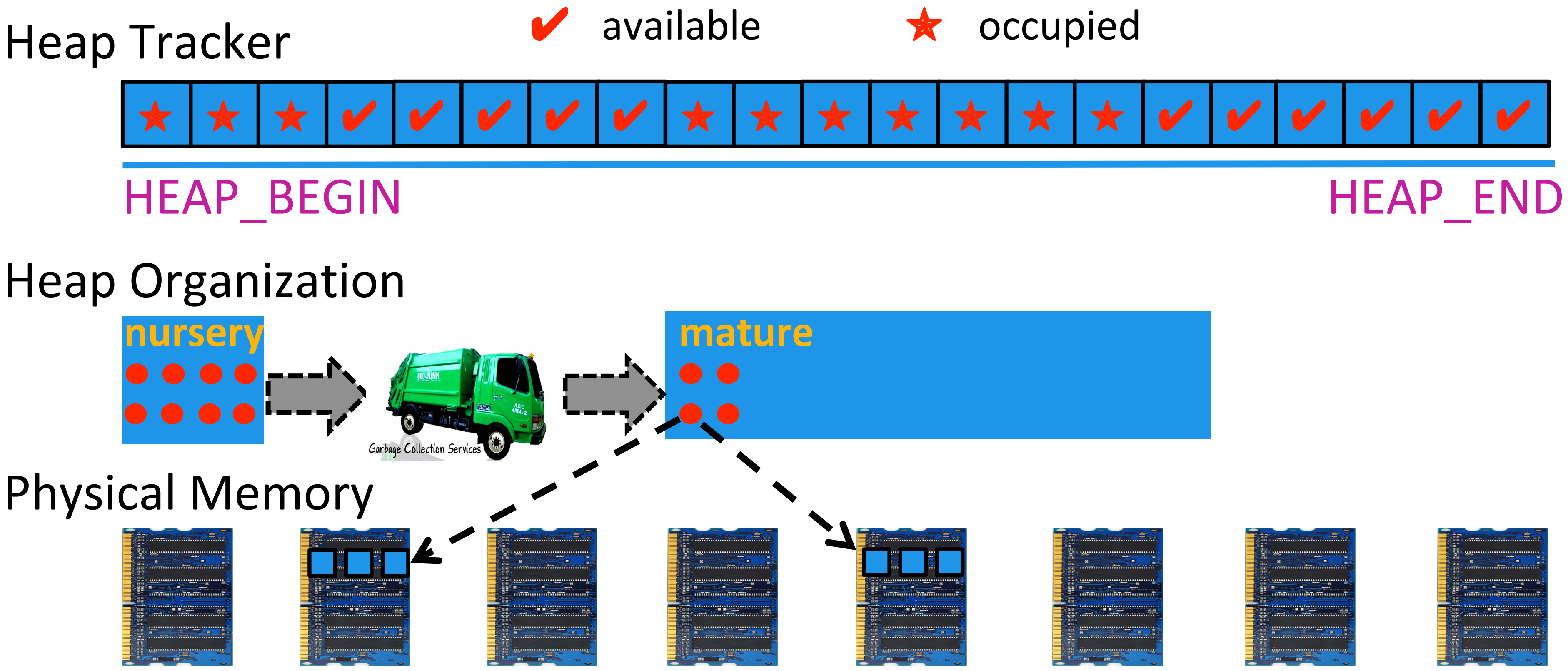


★ ★ ★ ✔ ✔ ✔ ✔ ✔ ★ ★ ★ ★ ★ ★ ★ ✔ ✔ ✔ ✔ ✔ ✔

Heap Organization



DRAM heap management



Heap Tracker

✔ available ★ occupied

★ ★ ★ ✔ ✔ ✔ ✔ ✔ ★ ★ ★ ★ ★ ★ ★ ✔ ✔ ✔ ✔ ✔ ✔

HEAP_BEGIN HEAP_END

Heap Organization

nursery

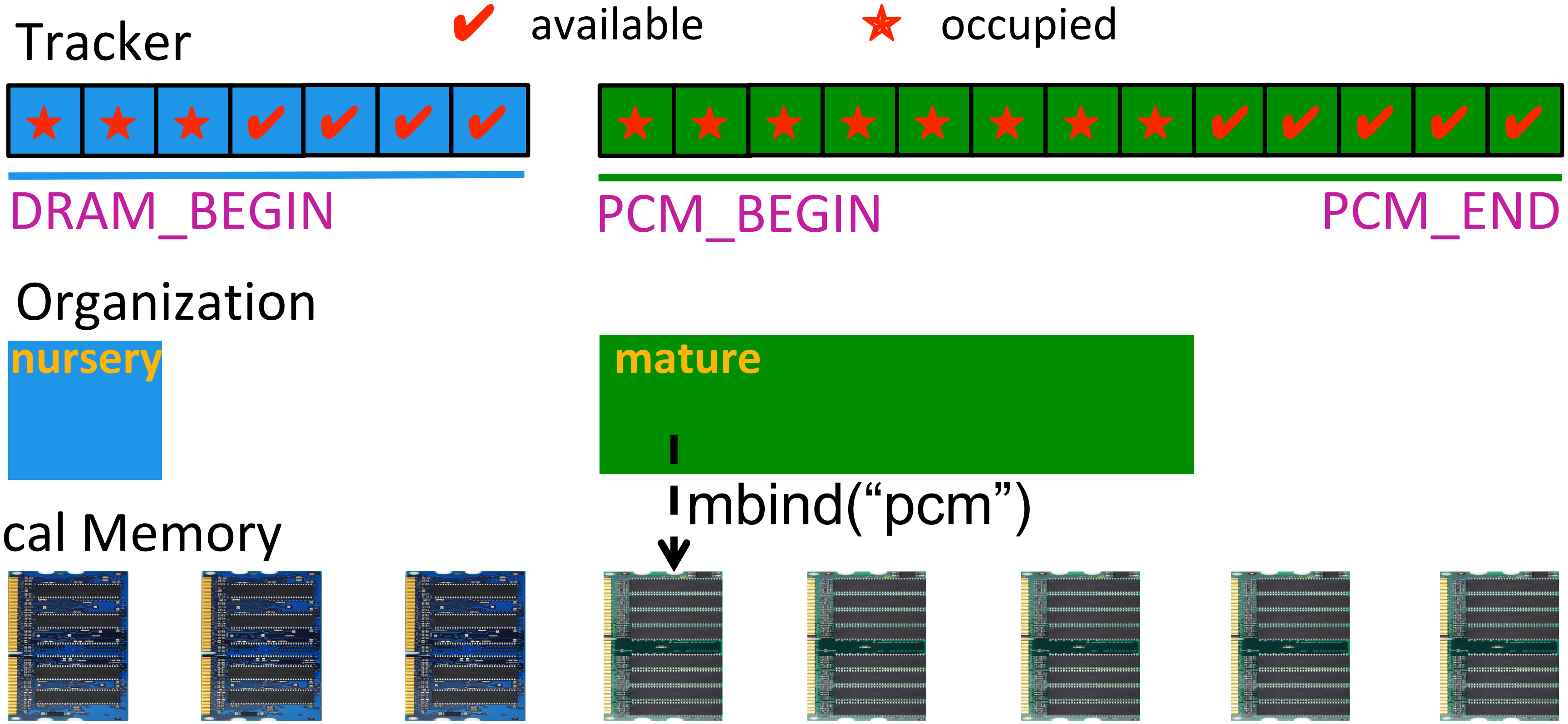
mature

Physical Memory

DRAM-PCM heap management

Heap Tracker

✔ available ★ occupied



★ ★ ★ ✔ ✔ ✔ ✔

★ ★ ★ ★ ★ ★ ★ ★ ✔ ✔ ✔ ✔ ✔

DRAM_BEGIN

PCM_BEGIN

PCM_END

Heap Organization

nursery

mature

mbind("pcm")

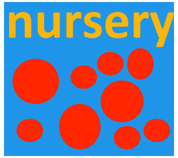
Physical Memory

Kingsguard-Nursery (KG-N)

Write-rationing GC: concentrate writes in DRAM

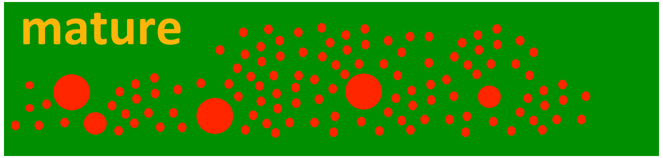
70%

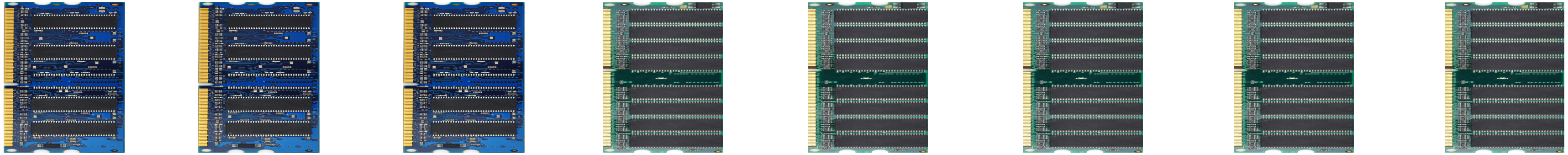
of writes



22%

to 2% of objects

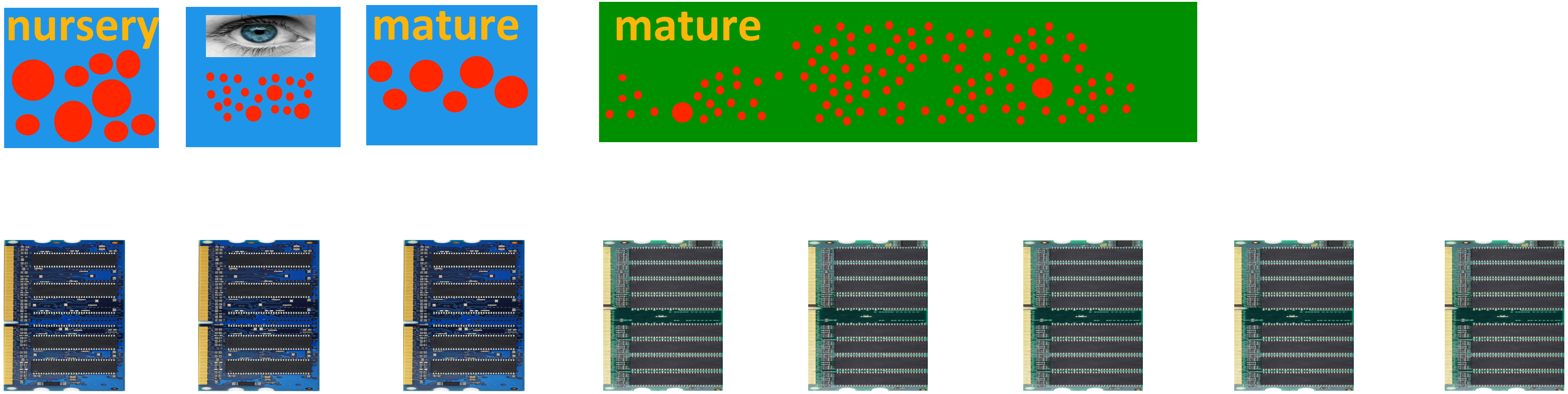




Kingsguard-Writers (KG-W)

KG-W monitors writes in a DRAM observer space

Trades off performance for better endurance



More optimizations in KG-W

Short-lived large objects in DRAM

Large data-structures cause writes to PCM

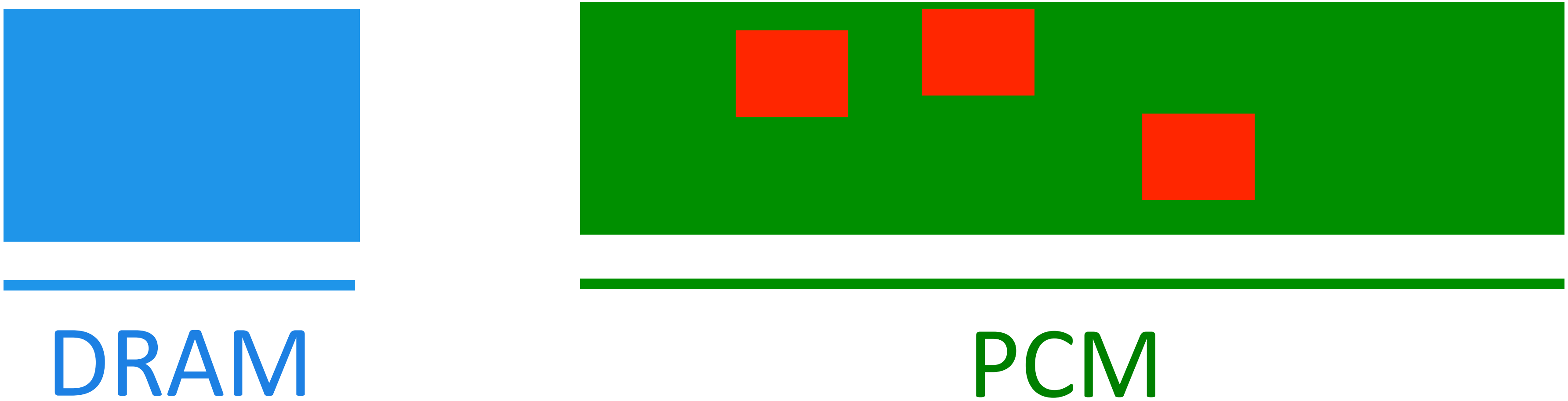
Keep them in DRAM

Object meta-data in DRAM

GC updates to mark bits lead to writes to PCM

Keep them in DRAM

OS versus Kingsguard

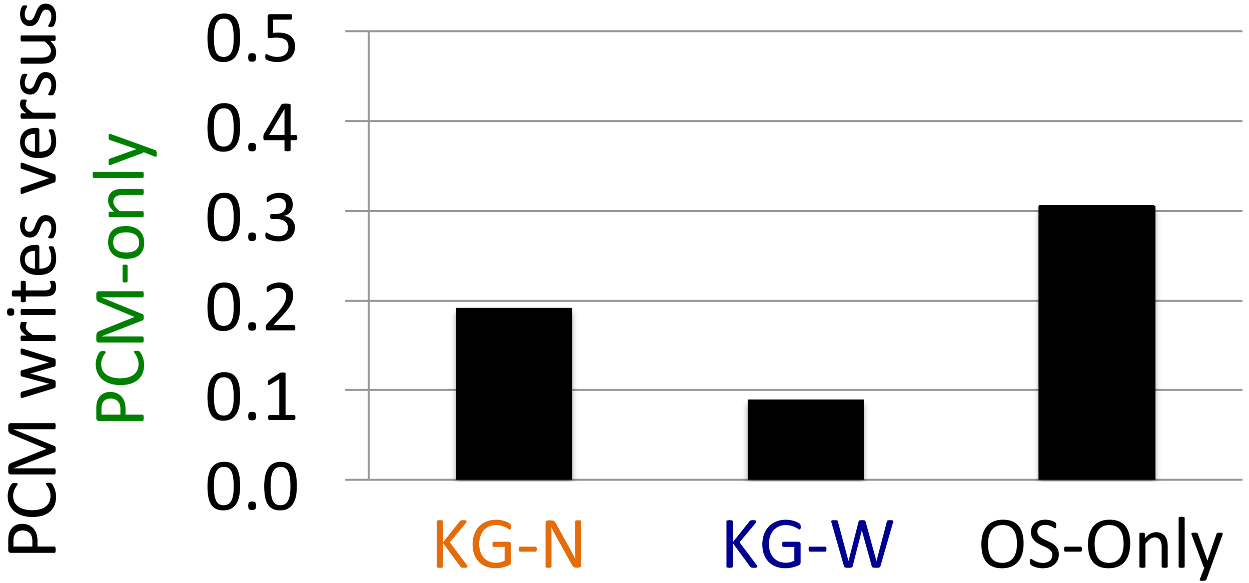


Rank pages according to writes

A page with $\mathcal{T}$ writes is a DRAM candidate

Adjust for phase behavior

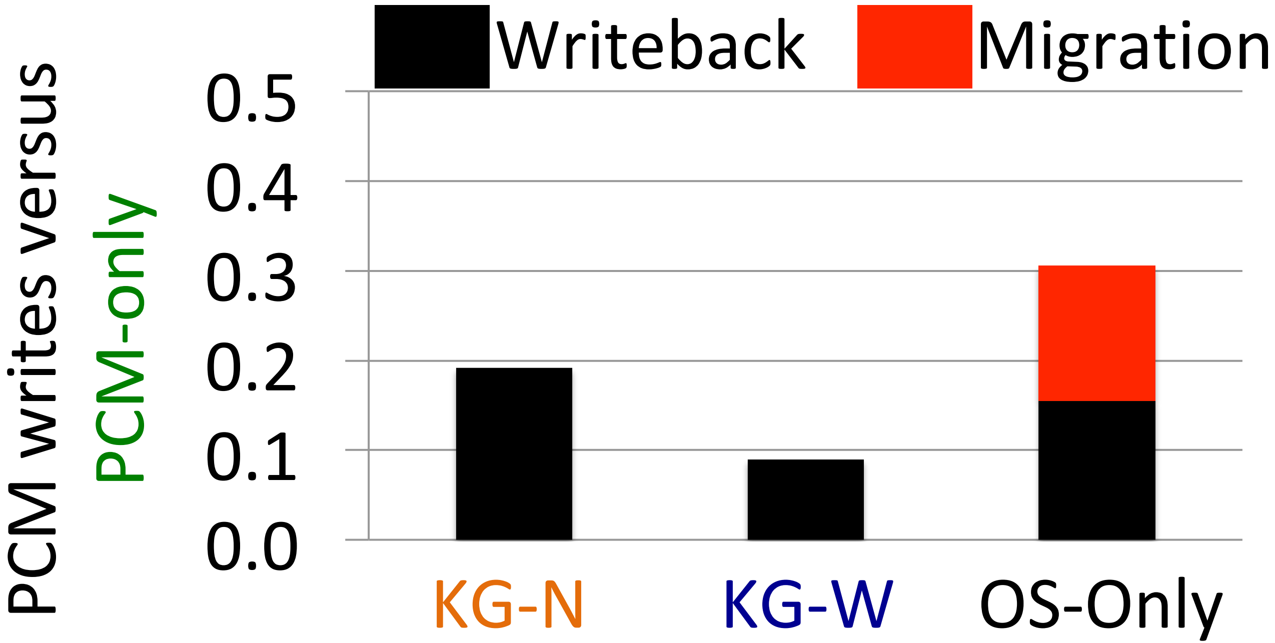
OS versus Kingsguard



Average for 7 DaCapo benchmarks in simulation

KG-W reduces 3X more writes than OS-Only

OS versus Kingsguard

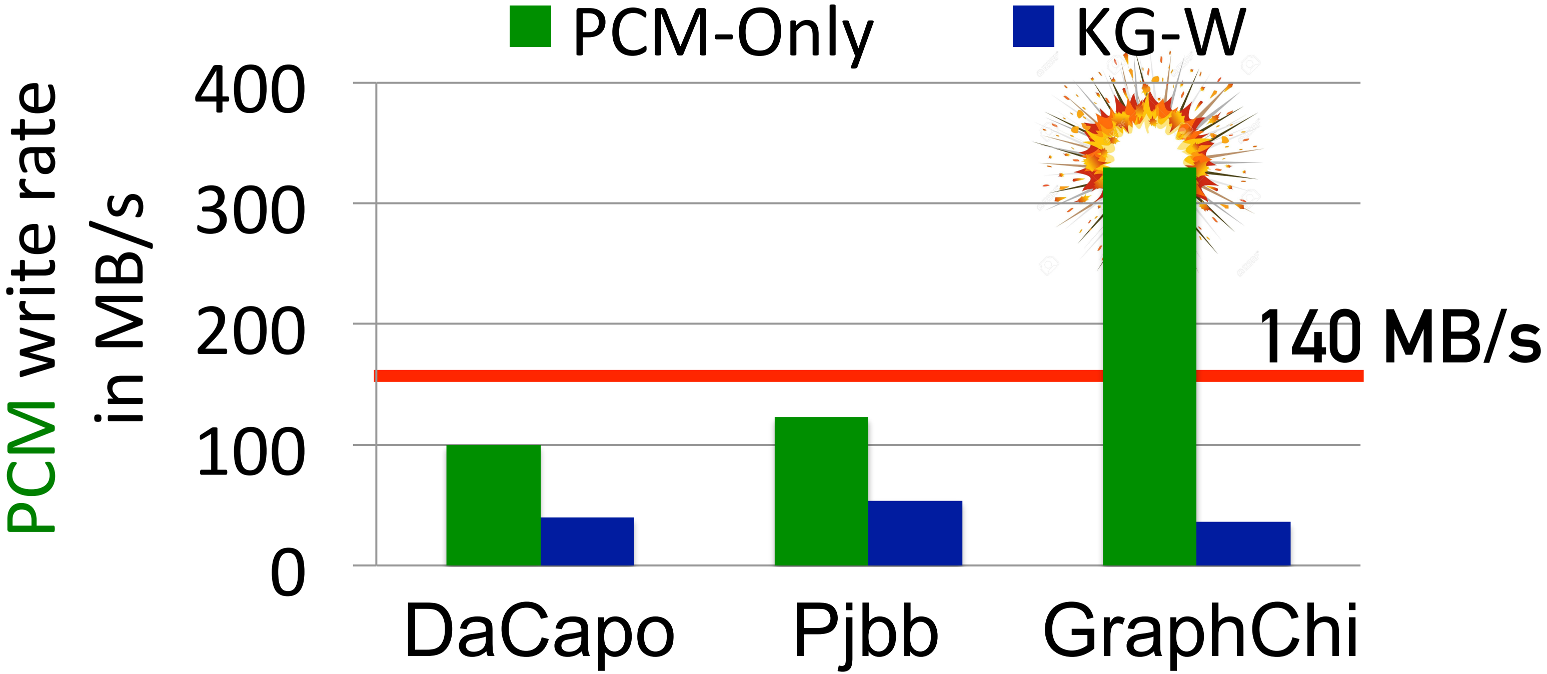


Average for 7 DaCapo benchmarks in simulation

OS predicts nursery pages but migrations harmful

Emulation on NUMA hardware

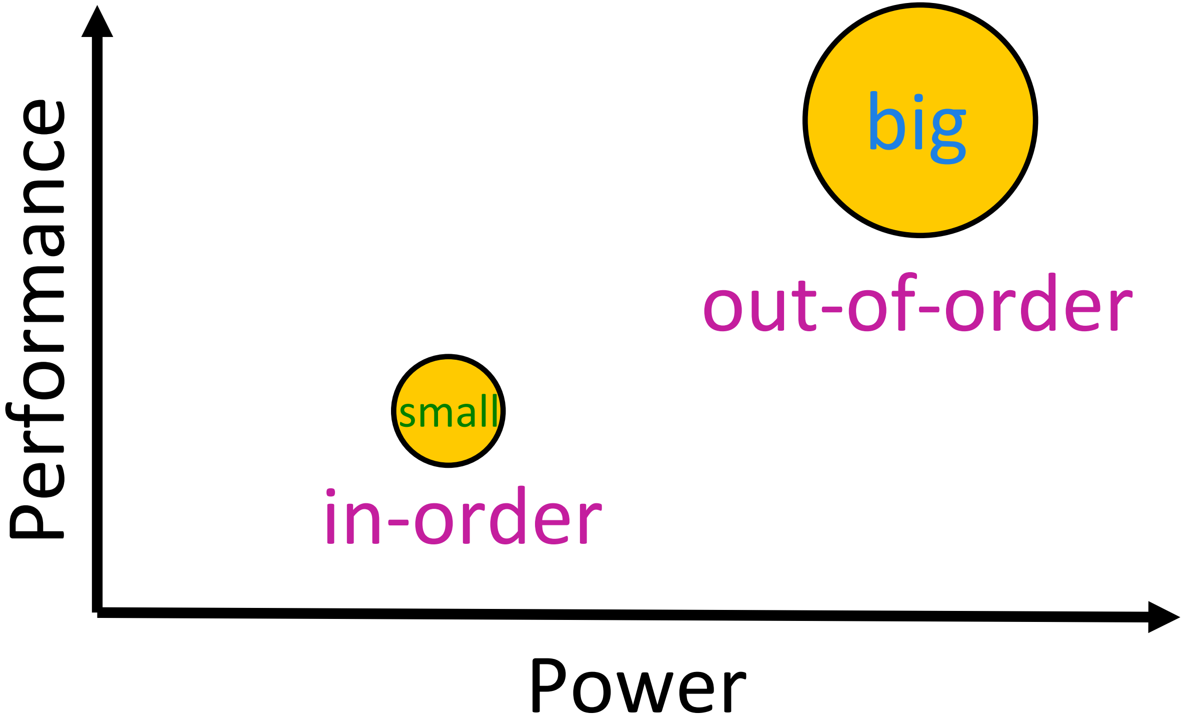
PCM-Only is not practical as main memory



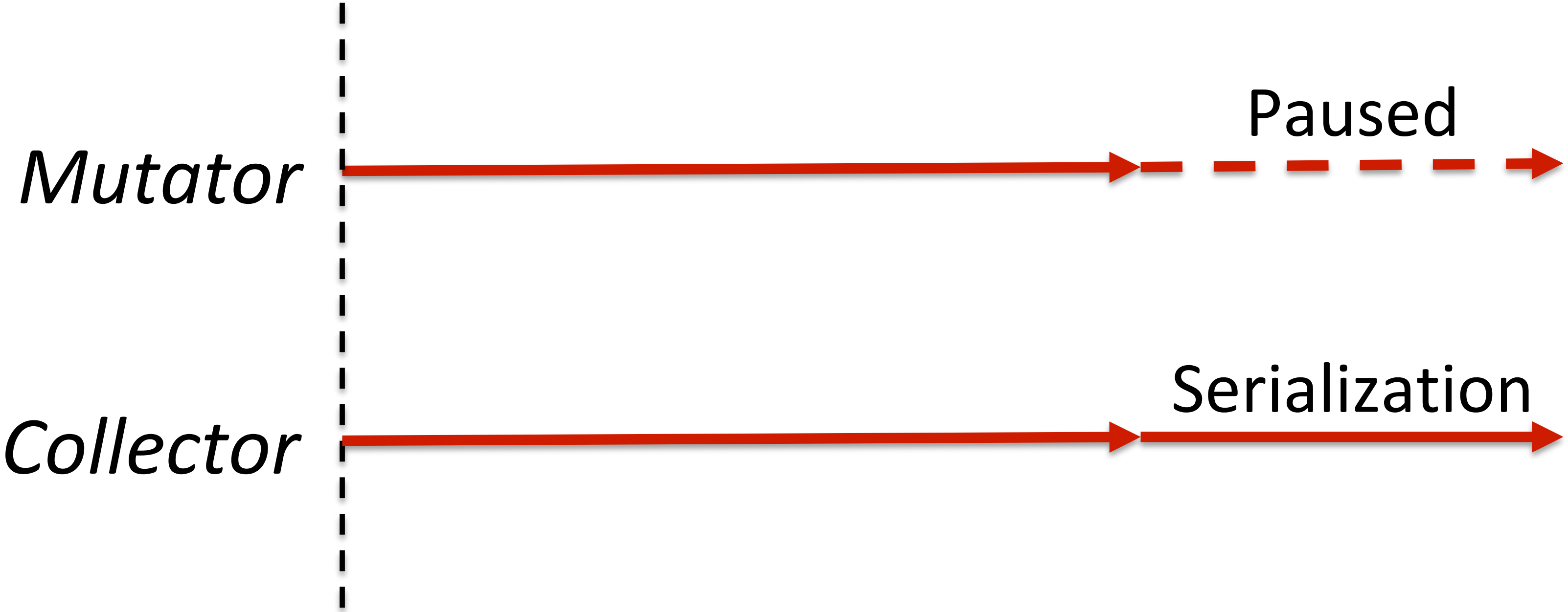
Crystal Gazer: Profile-Driven Write-Rationing Garbage Collection for Hybrid Memories

Heterogeneous multicore scheduling

Mutator→big, Concurrent GC → big or small?

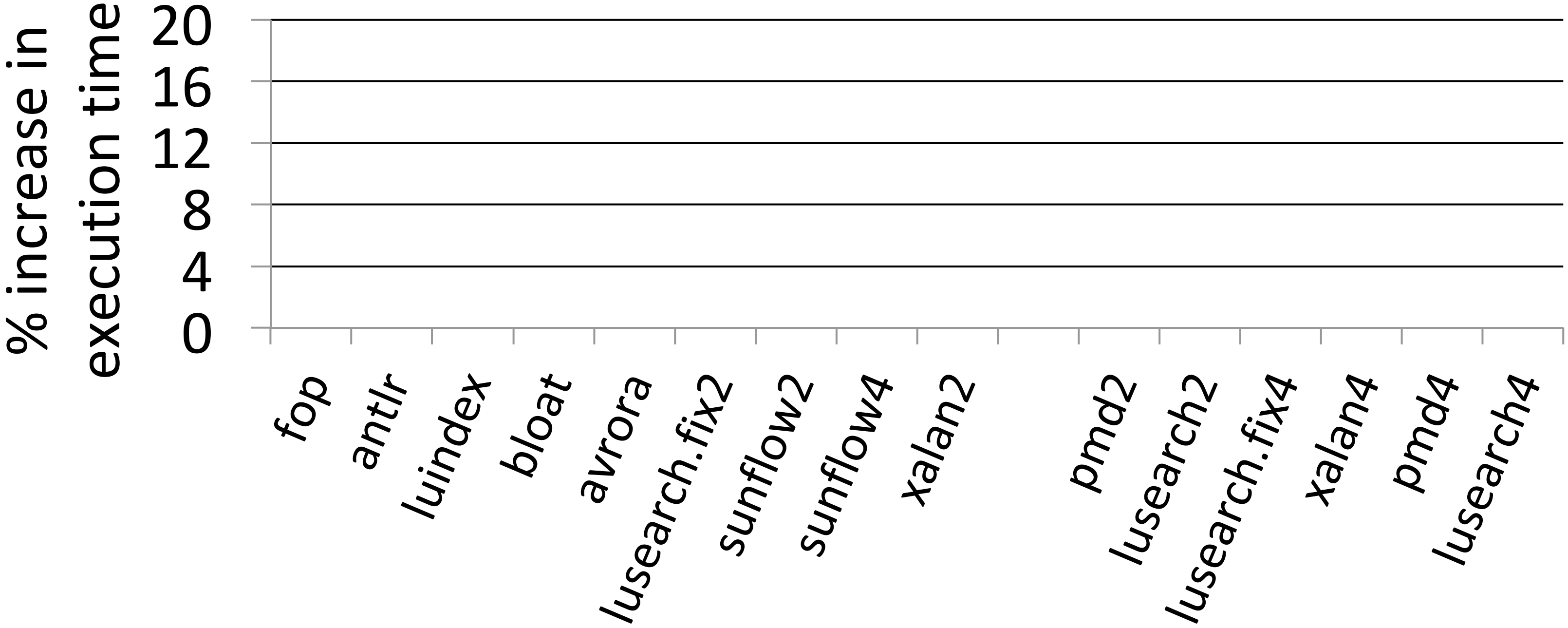


GC on big or small

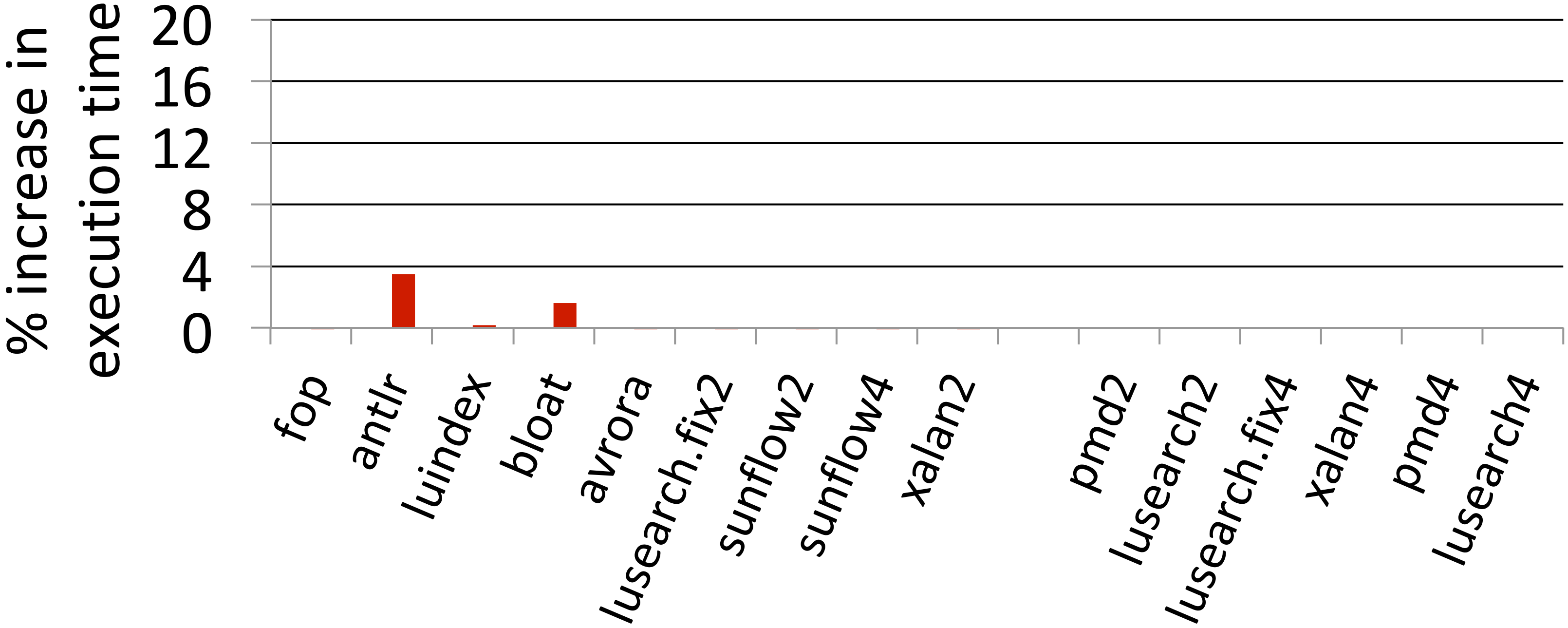


Slow GC pauses application due to no memory

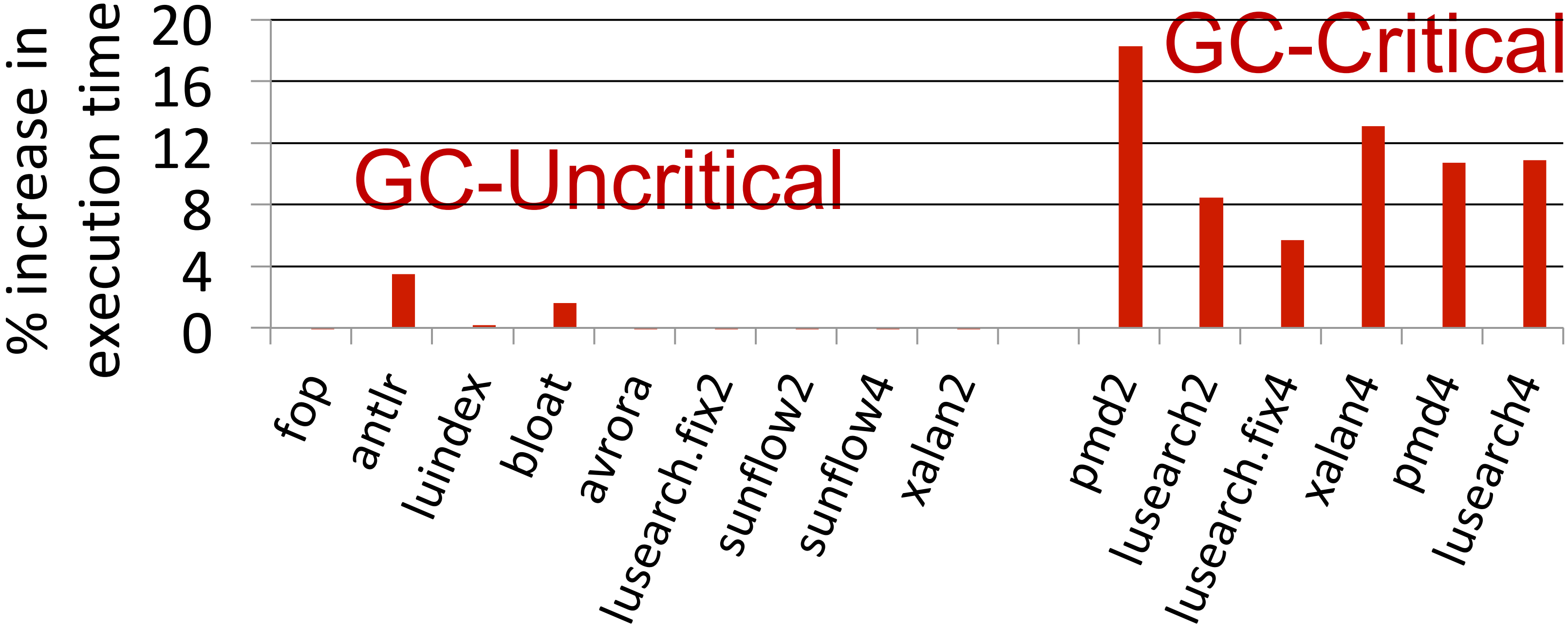
GC on big or small



GC on big or small



GC on big or small

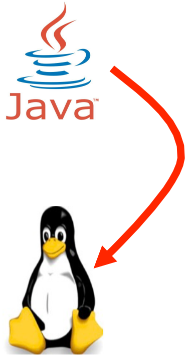


GC-criticality-aware scheduling

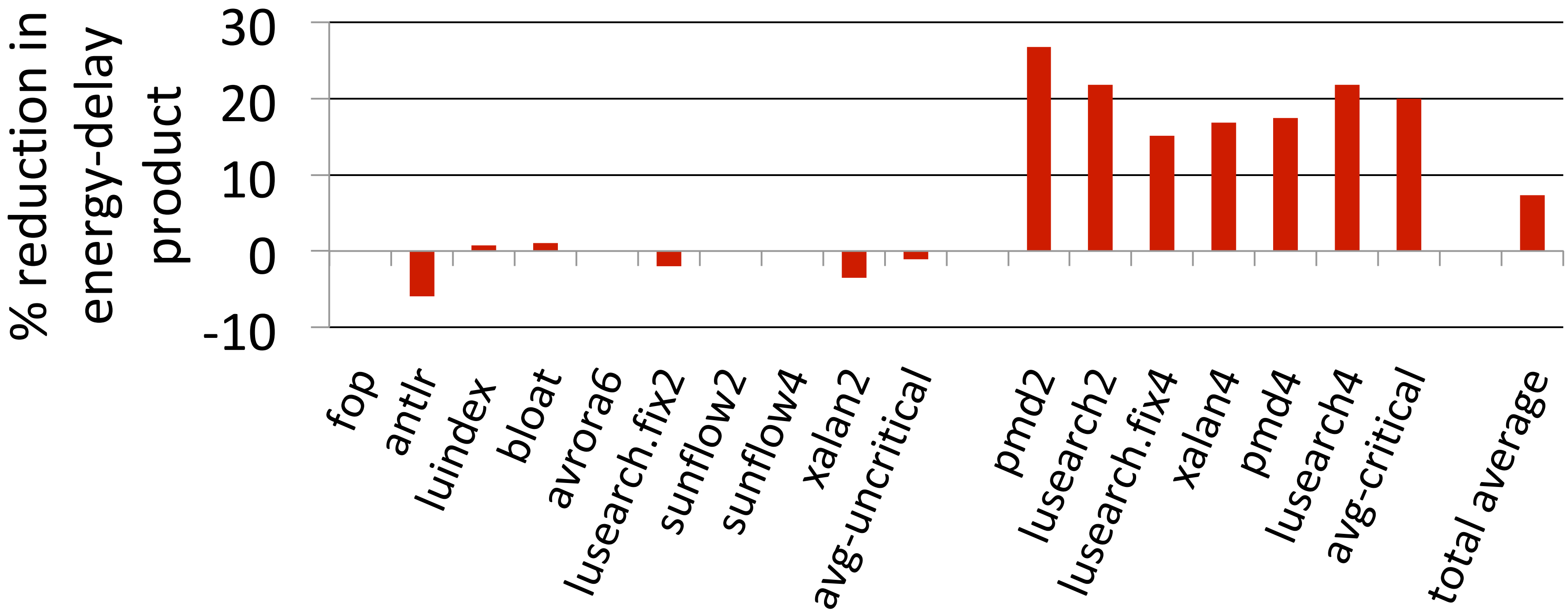
Runtime detects GC-criticality

Communicates criticality to the OS

OS adjusts GC priority



Better energy efficiency with GC-criticality-aware scheduling



To expose, or not to expose, hardware heterogeneity to runtimes

Always need OS (supervisory role)
Virtual memory, thread migration, and so on

Language runtimes can guide OS in forming the best policies to manage emerging hardware