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

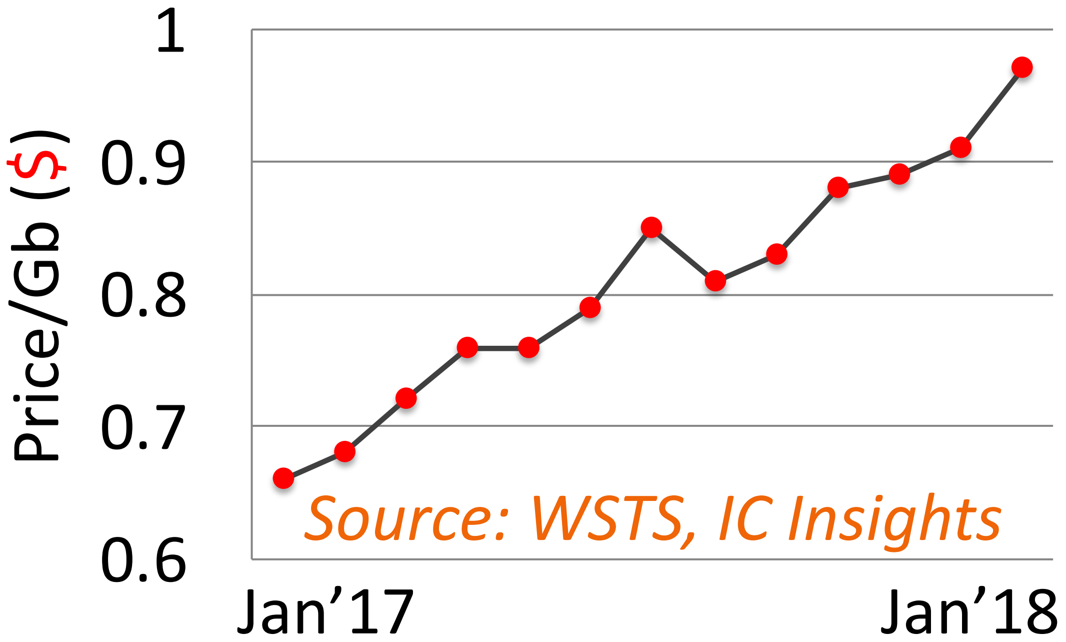
VUB VRIJE UNIVERSITEIT BRUSSEL

Google

Main memory capacity expansion

DRAM → Charge storage a scaling limitation

Manufacturing complexity makes DRAM pricing volatile



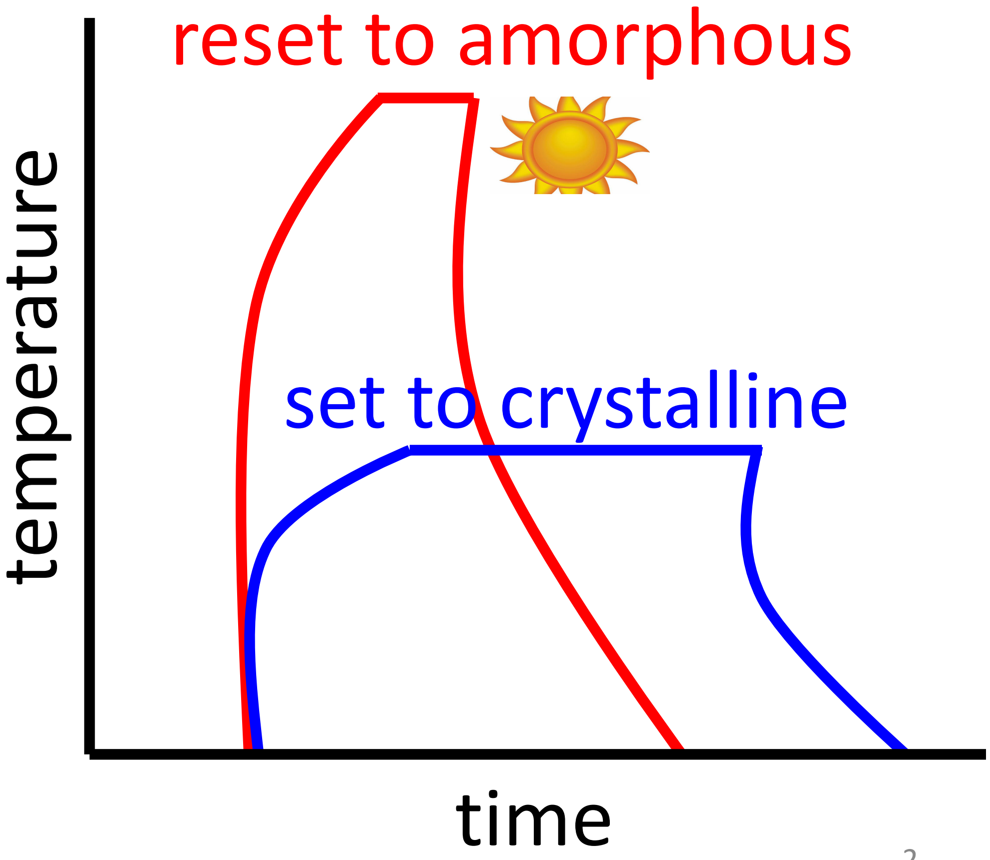
Phase change memory (PCM)

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

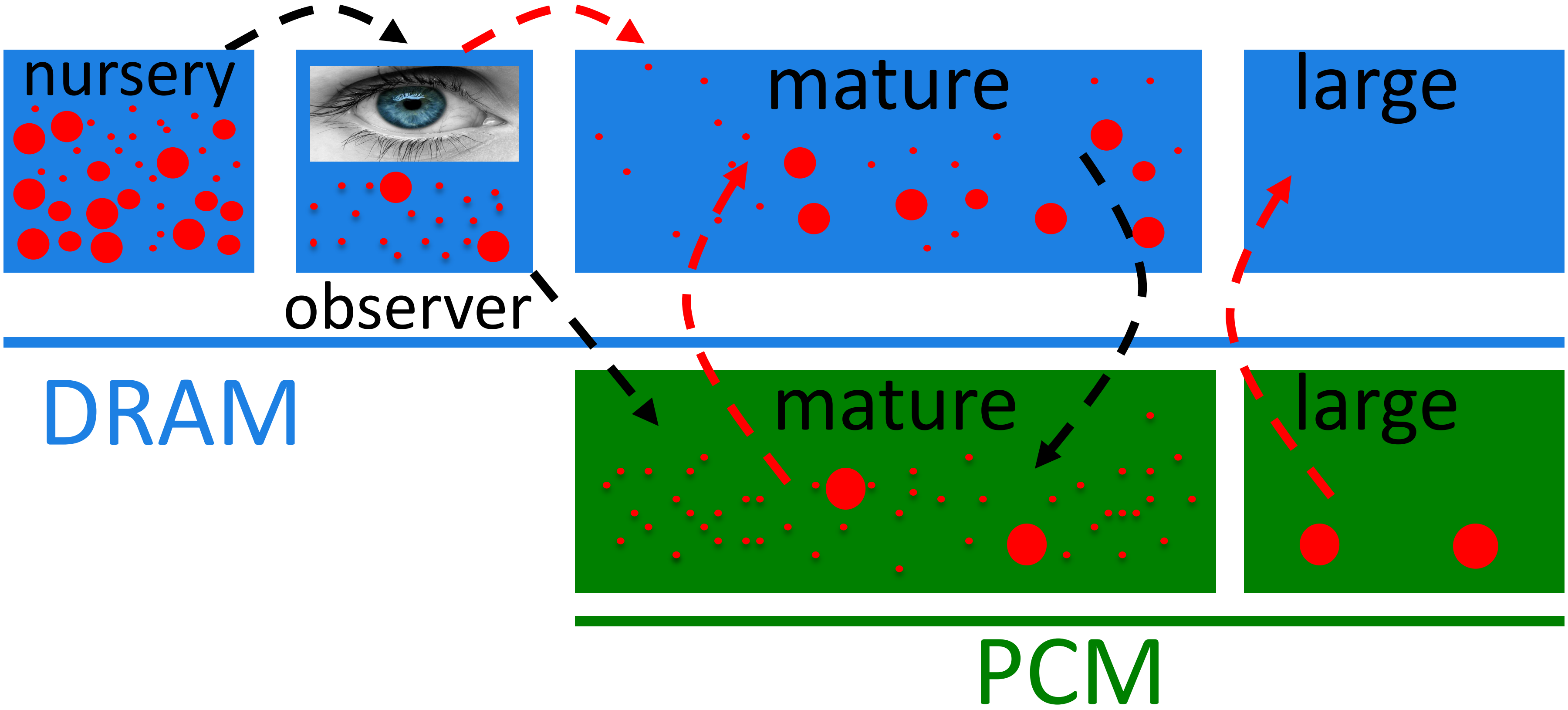
Application

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

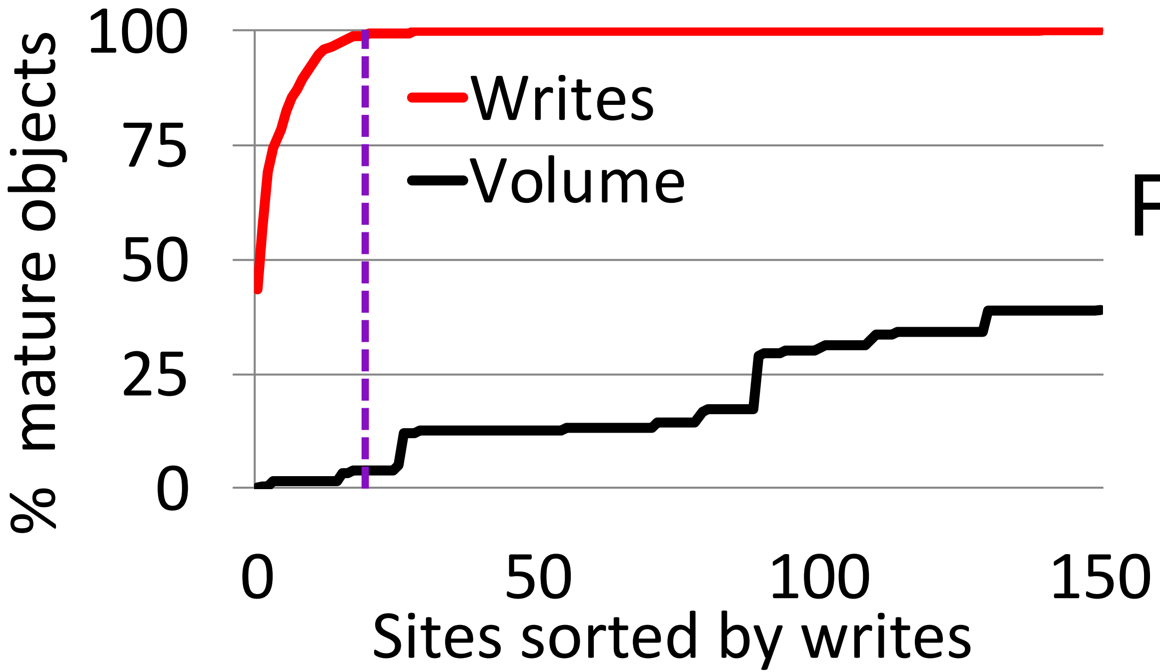
```
a = new Object()
b = new Object()
c = new Object()
d = new Object()
```

```
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()
```

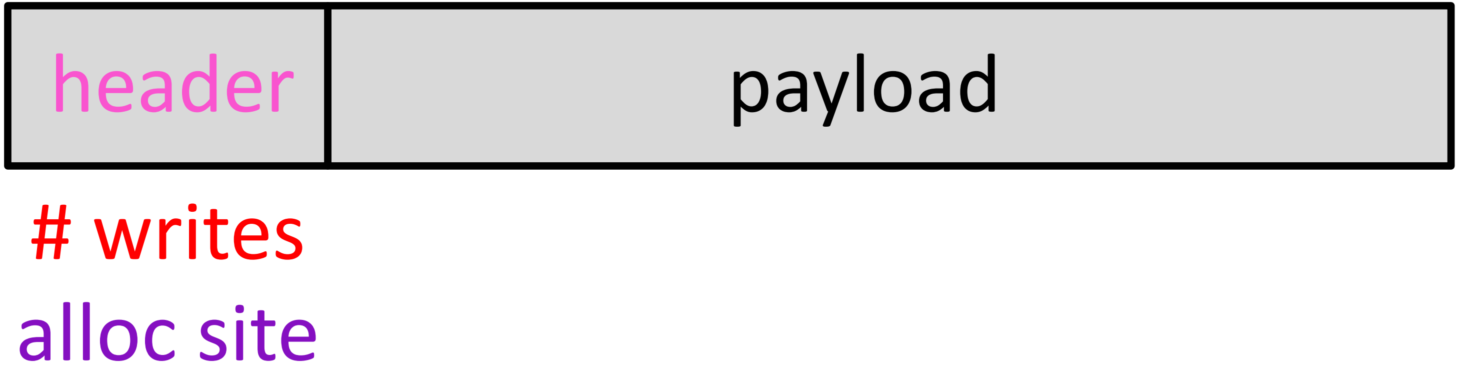
Application profiling (offline)

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



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()*

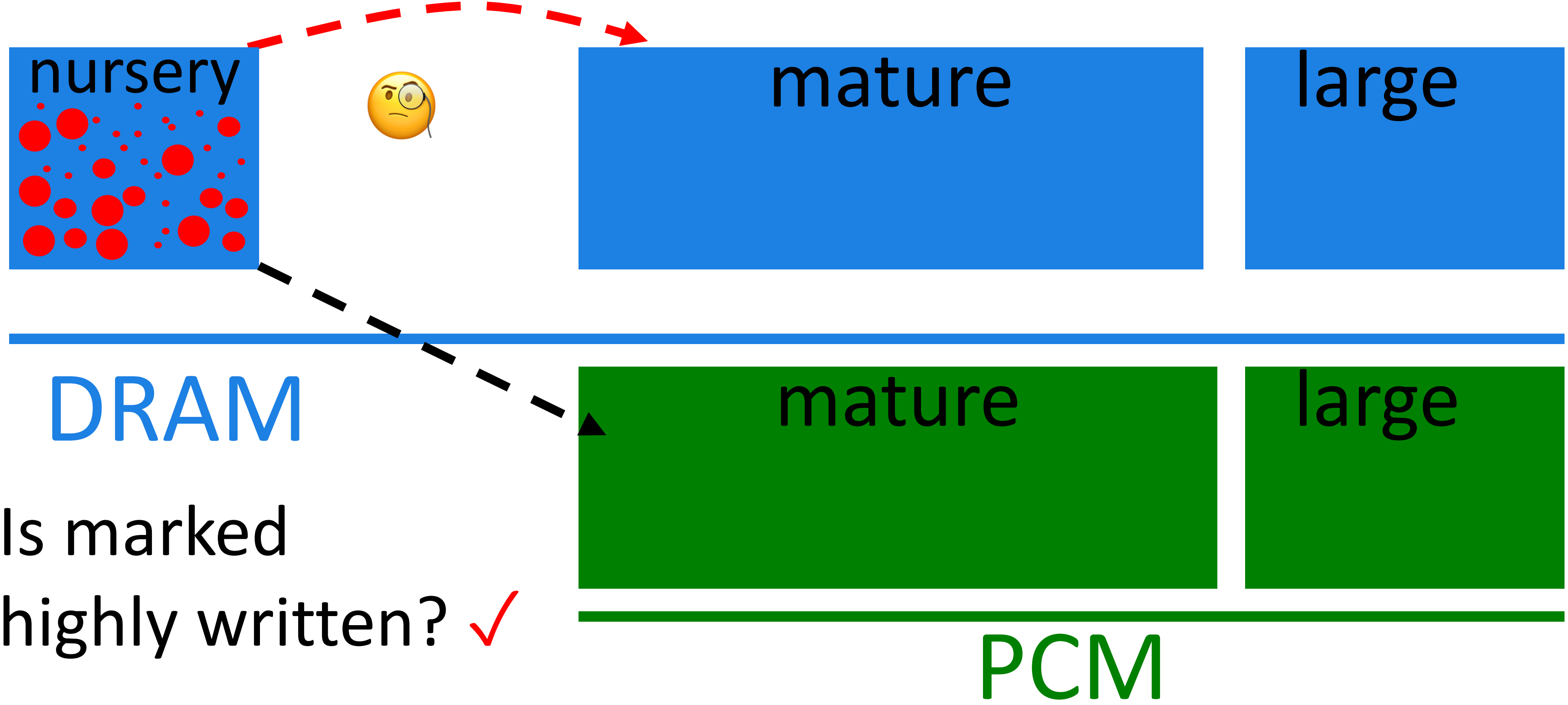
Bytecode rewriter modifies DRAM sites to use *new_dram()*

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

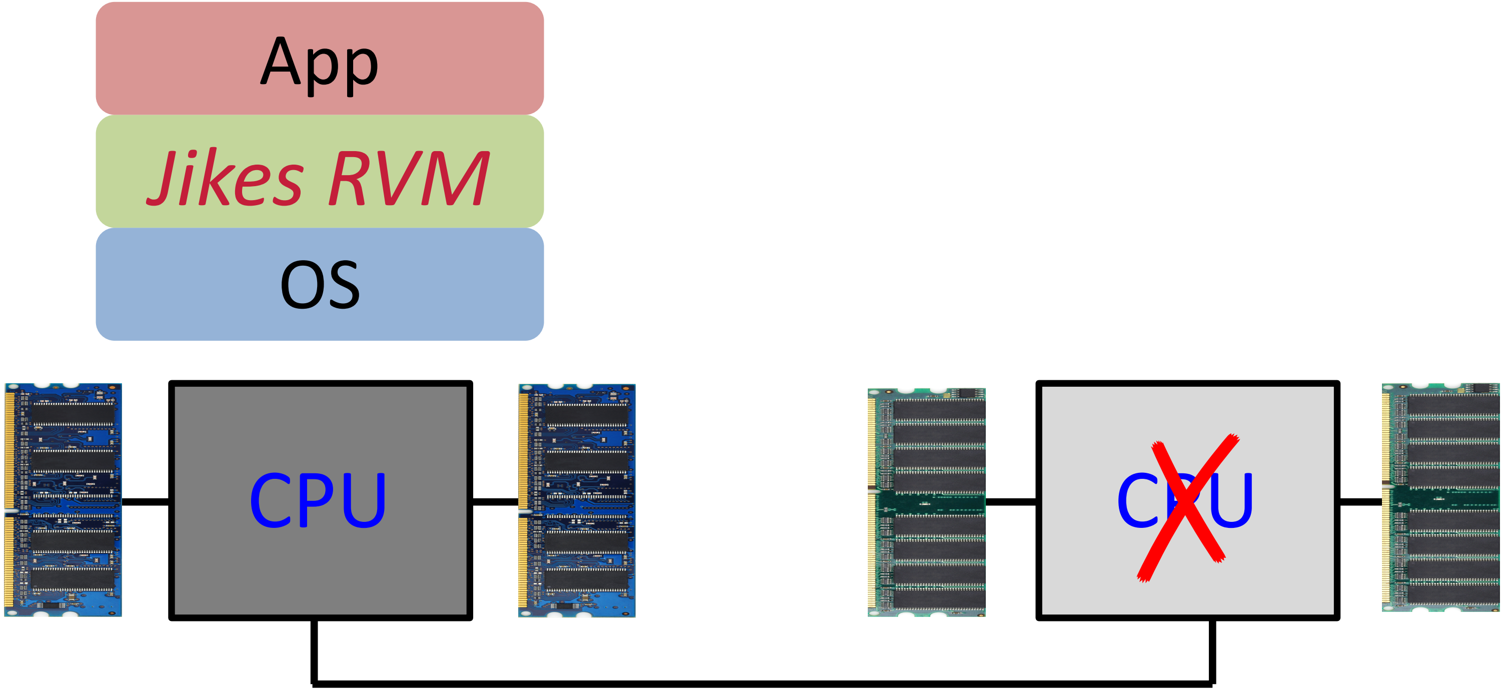
15 Applications → DaCapo, GraphChi, SpecJBB

Medium-end server platform

Different inputs for production and advice

Jikes RVM

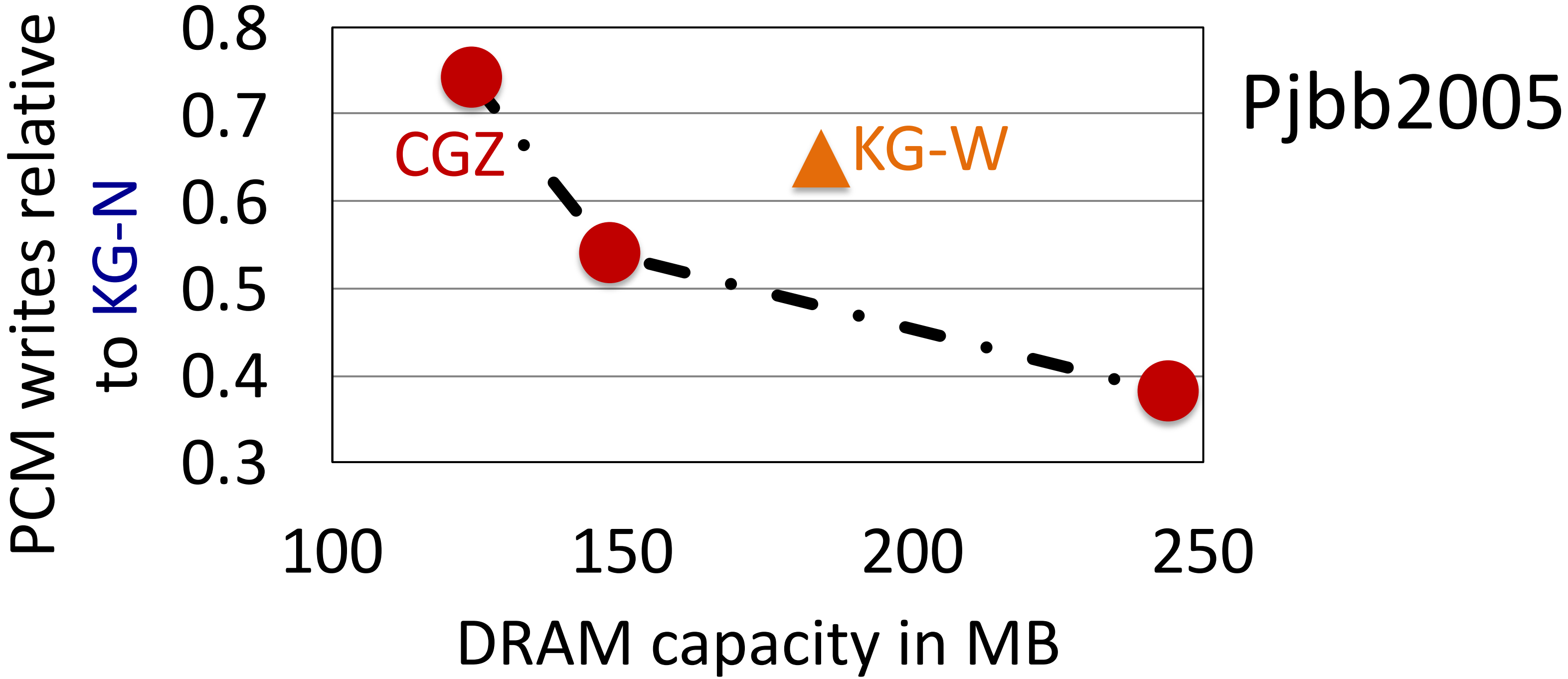
Emulation on NUMA hardware



16 hardware threads and 20 MB L3

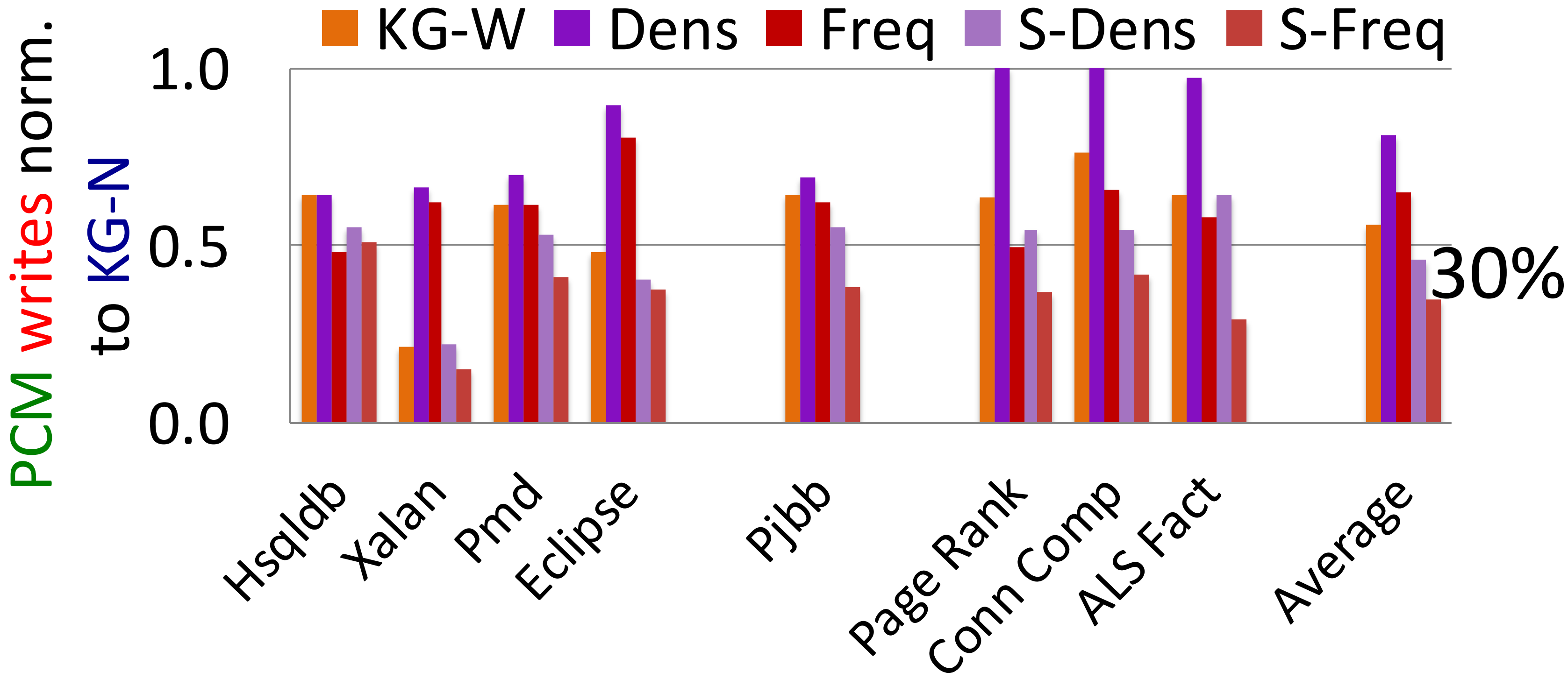
Use Intel pcm-memory.x to get per-socket write rate

Lifetime versus DRAM capacity



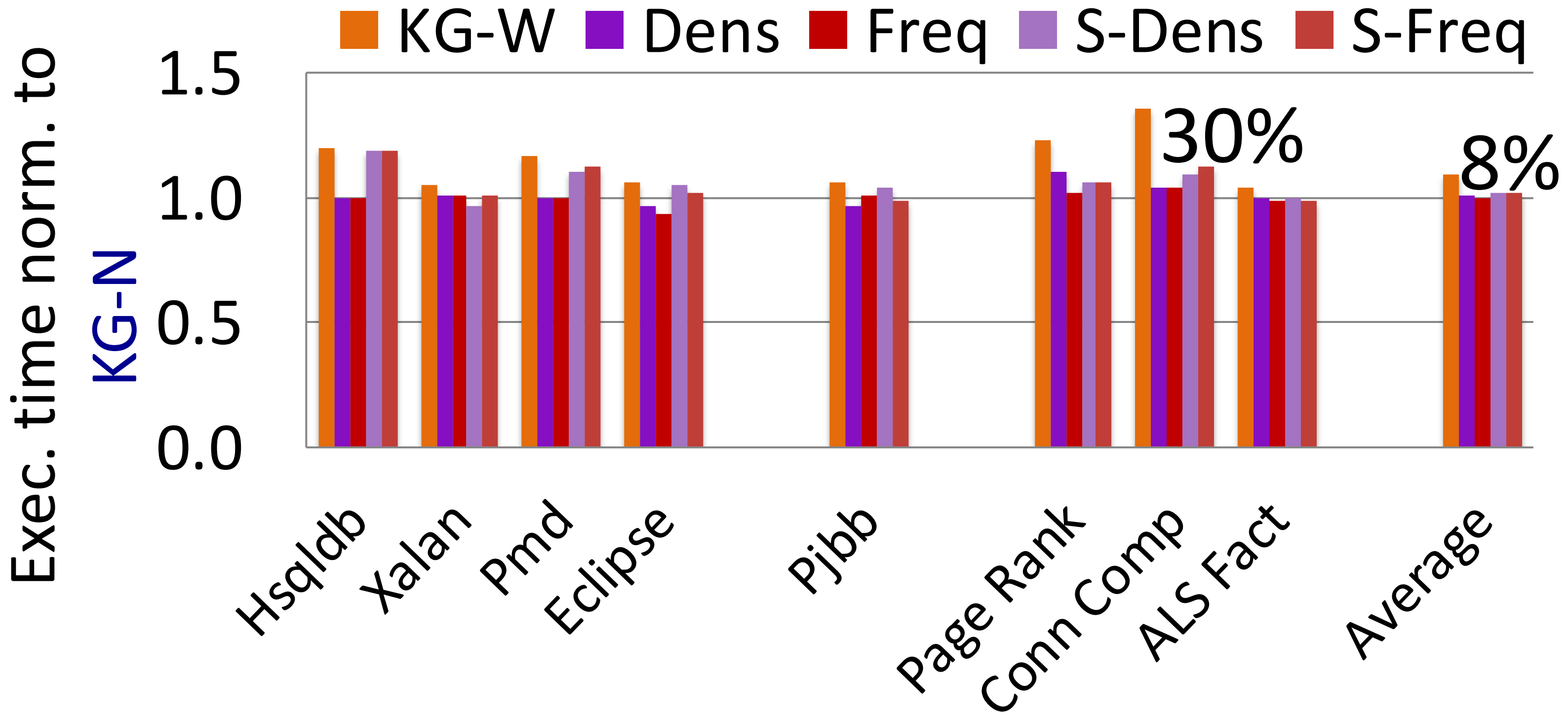
Crystal Gazer provides Pareto-optimal choices

PCM Writes



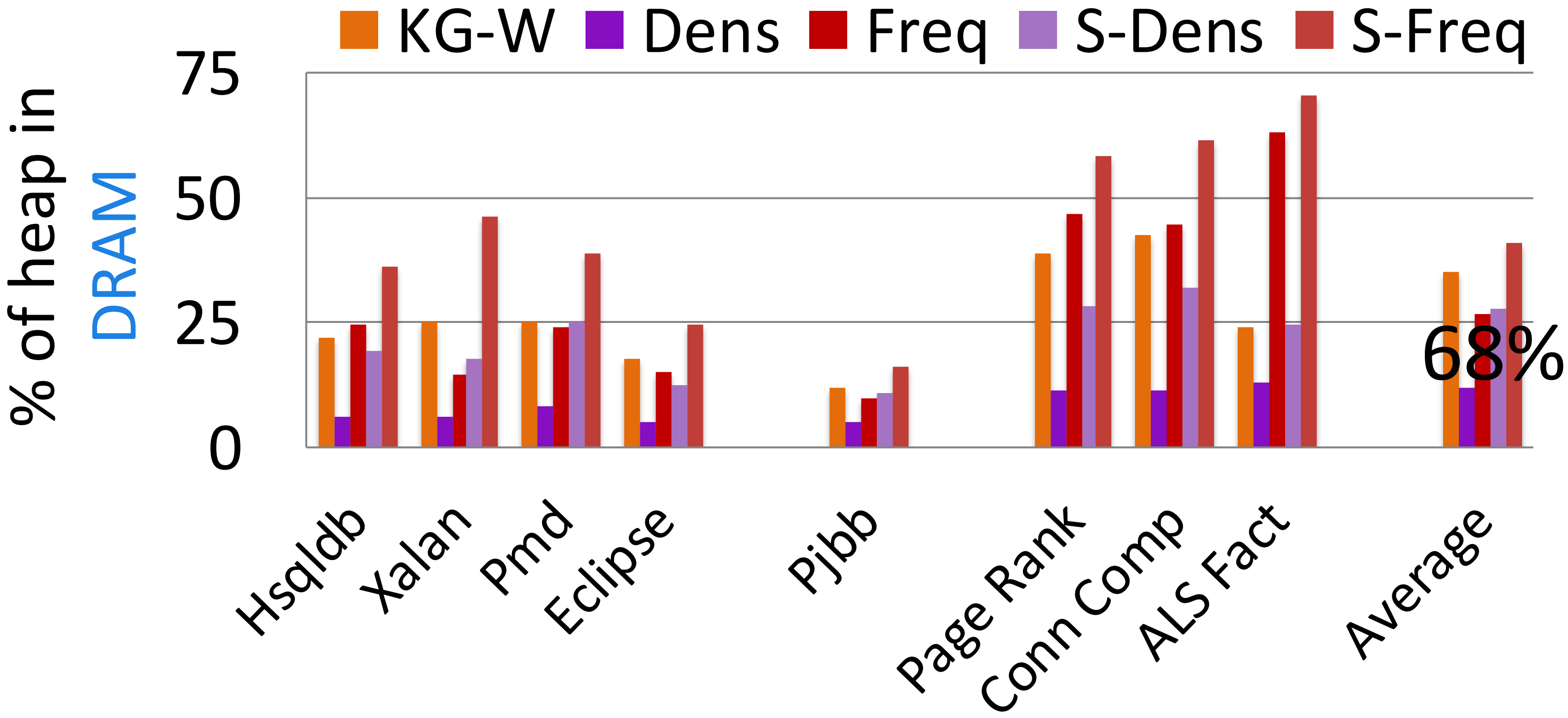
To optimize for lifetime, use Freq & survivors

Execution time



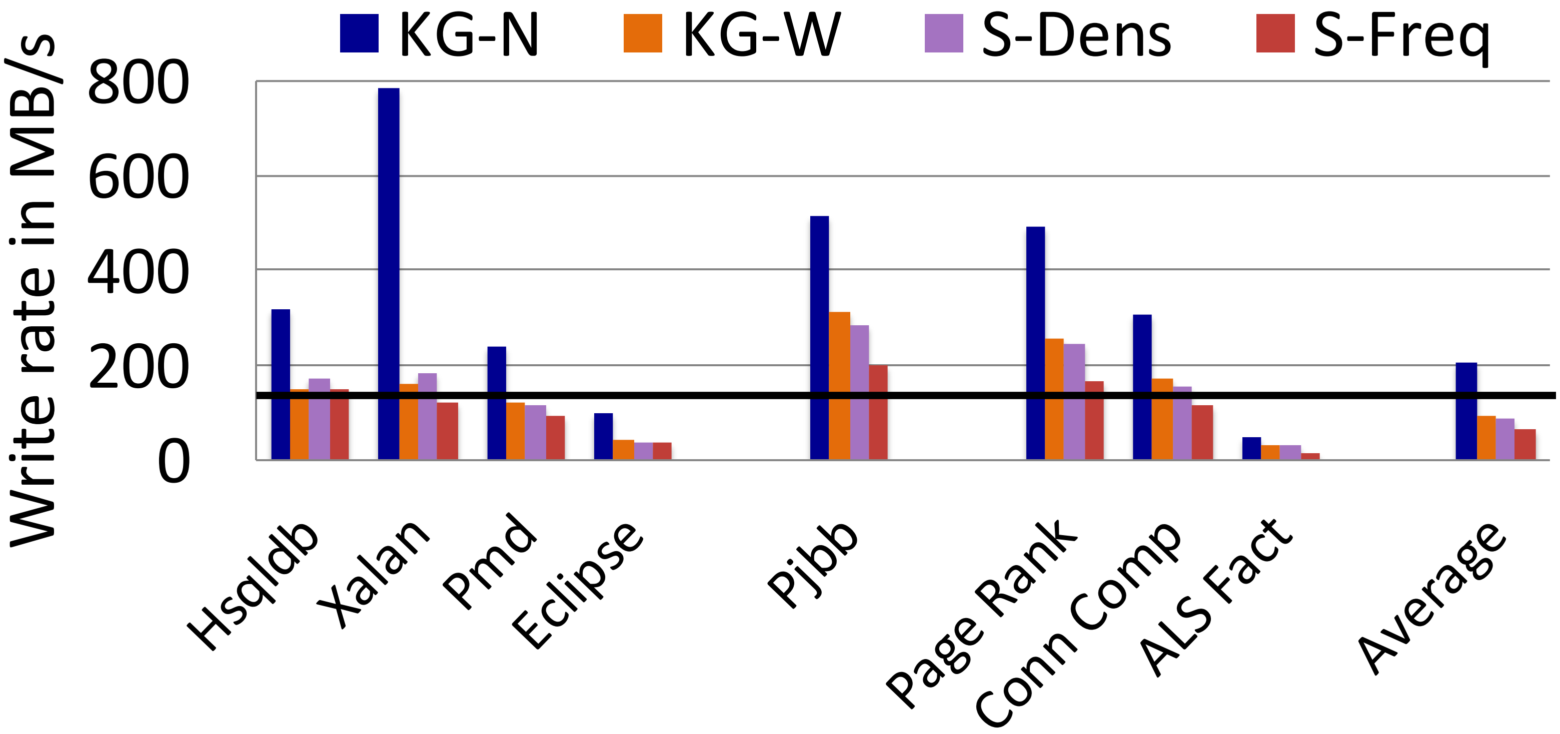
To optimize for performance, use Freq or Dens

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