Software Wear Management for Persistent Memories

Vaibhav Gogte, William Wang[1], Stephan Diestelhorst[1], Aasheesh Kolli[2,3],
Peter M. Chen, Satish Narayanasamy, Thomas F. Wenisch

University of Michigan

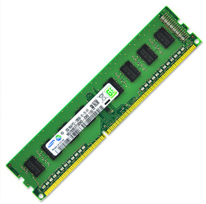
PennState [2]

arm [1]

vmware [3]

FAST'19
02/26/2019

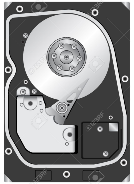
Promise of persistent memory (PM)



Performance

+

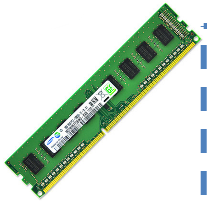
Density



+

Non-volatility

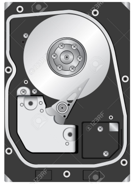
Promise of persistent memory (PM)



Performance

+

Density



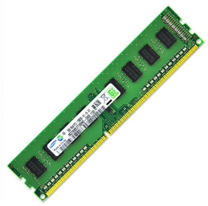
+

Non-volatility

PM for capacity expansion

PM cheaper and denser than DRAM

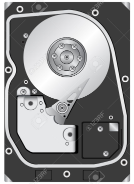
Promise of persistent memory (PM)



Performance

+

Density



+

Non-volatility

PM for capacity expansion

PM cheaper and denser than DRAM

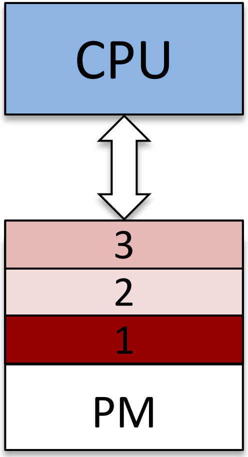
PM as storage

PM enables faster storage via load-store interface

PMs have low write endurance

- PM cells wear out after $10^7 – 10^9$ writes [Lee '09]

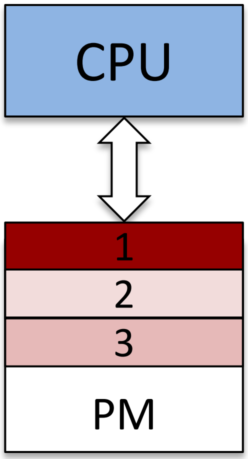
Wear-leveling mechanisms



PMs have low write endurance

- PM cells wear out after $10^7 – 10^9$ writes [Lee '09]

Wear-leveling mechanisms



Remap locations to uniformly distribute writes

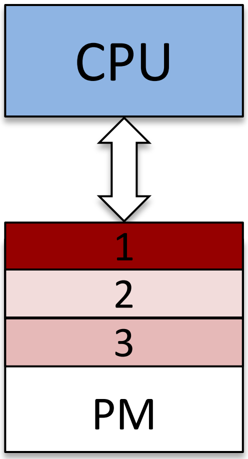
MichiganEngineering

arm

PMs have low write endurance

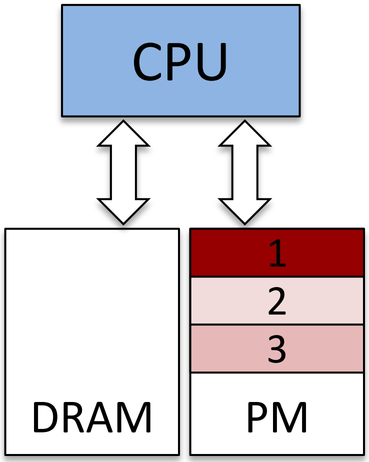
- PM cells wear out after $10^7 - 10^9$ writes [Lee '09]

Wear-leveling mechanisms



Remap locations to uniformly distribute writes

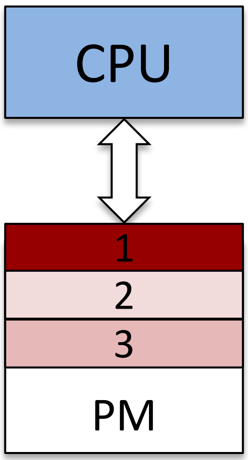
Wear-reduction mechanisms



PMs have low write endurance

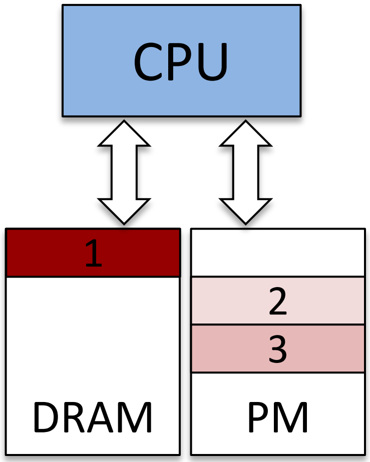
- PM cells wear out after $10^7 - 10^9$ writes [Lee '09]

Wear-leveling mechanisms



Remap locations to uniformly distribute writes

Wear-reduction mechanisms



Map heavily written locations to DRAM

Wear management in software

- Prior proposals measure PM wear in hardware [Qureshi '09, Ramos '11, …]
  - *Wear leveling*: Add latency of additional translation layer
  - *Wear reduction*: Require specialized memory controllers

- Our proposal: **Wear-aware virtual memory system**
  - Employ virtual-to-physical page mappings to manage wear
  - Eliminates need for another translation layer
  - Require no special hardware to measure PM wear

Challenge: Measurement of PM writes at a page granularity in software

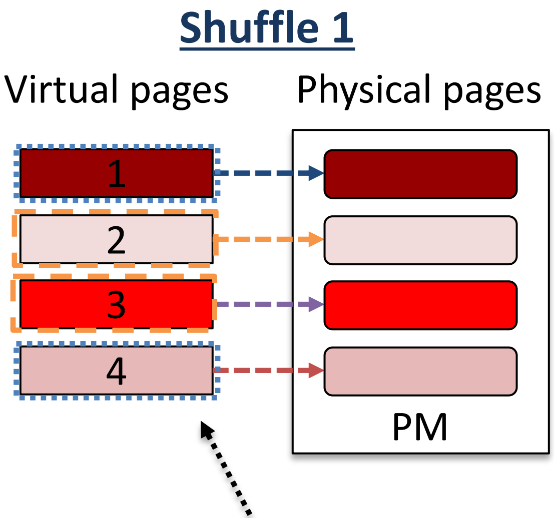
Contributions

Analytical framework	Simple remapping achieves near-uniform wear
⇩	
Wear leveling	Periodically remaps virtual-to-physical mappings
Wear estimation	Estimates per page wear in software
⇩	
Wear reduction	Migrates heavily written pages to high endurance mem.

Kevlar achieves PM **lifetime target of 4 years** with **1.2% performance overhead**

Wear leveling uniformly wears out PM pages

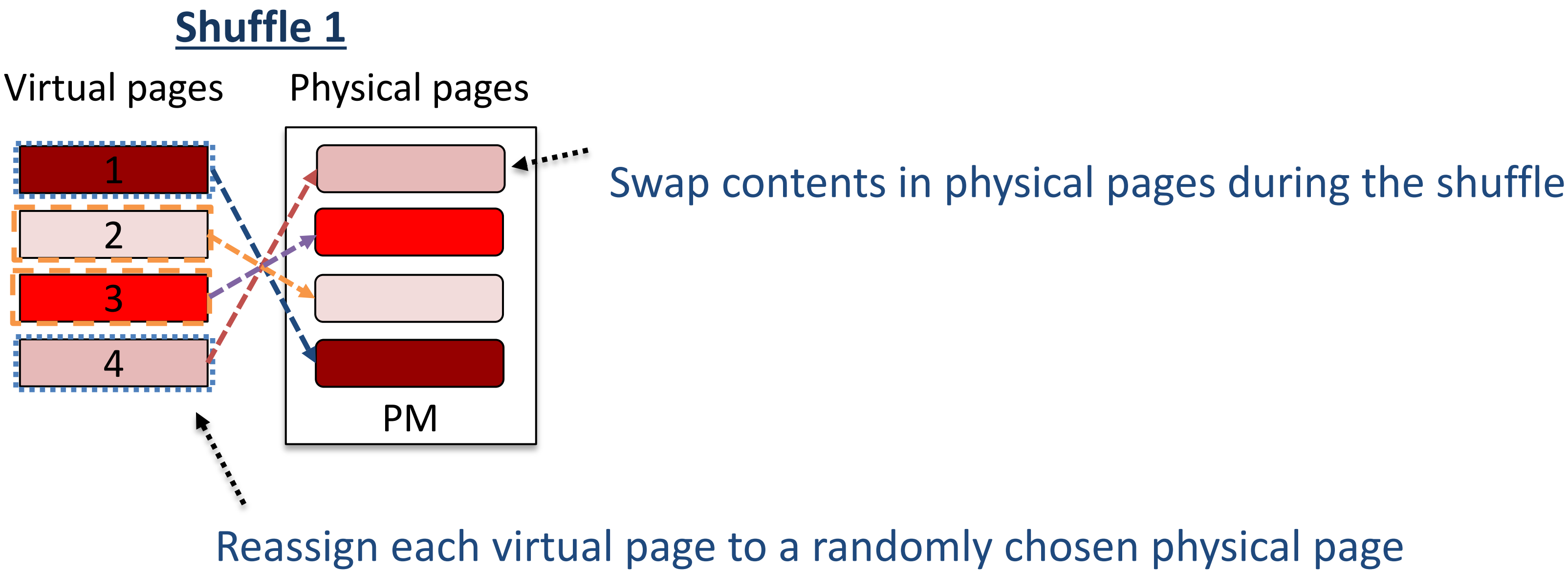
- Periodically *shuffle* memory footprint to spread writes uniformly in PM



Reassign each virtual page to a randomly chosen physical page

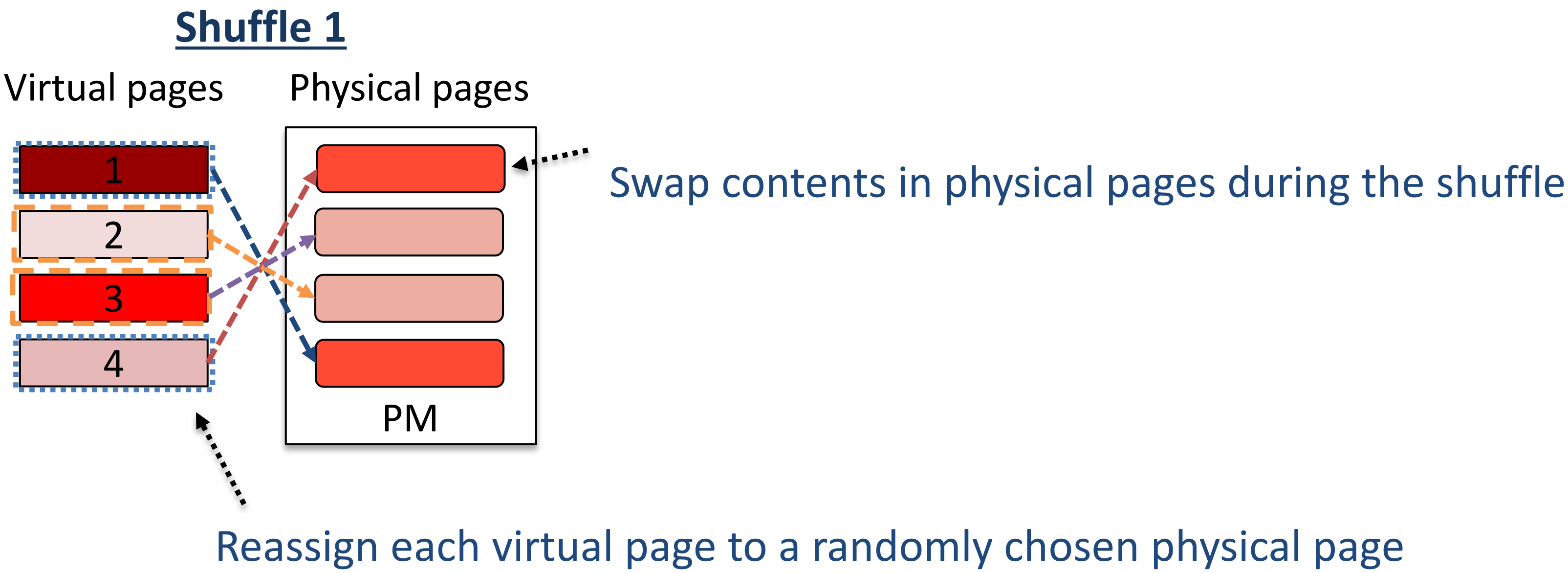
Wear leveling uniformly wears out PM pages

- Periodically *shuffle* memory footprint to spread writes uniformly in PM



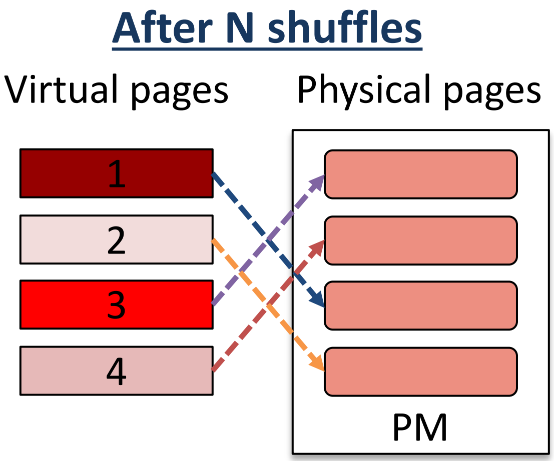
Wear leveling uniformly wears out PM pages

- Periodically *shuffle* memory footprint to spread writes uniformly in PM



Wear leveling uniformly wears out PM pages

- Periodically *shuffle* memory footprint to spread writes uniformly in PM



Disparity in page wear shrinks as shuffles increase

- **Does not require measurement of per page wear**
- **Depends on average PM write bandwidth**

Are random shuffles enough to achieve near-uniform wear?

PM lifetime due to random shuffles

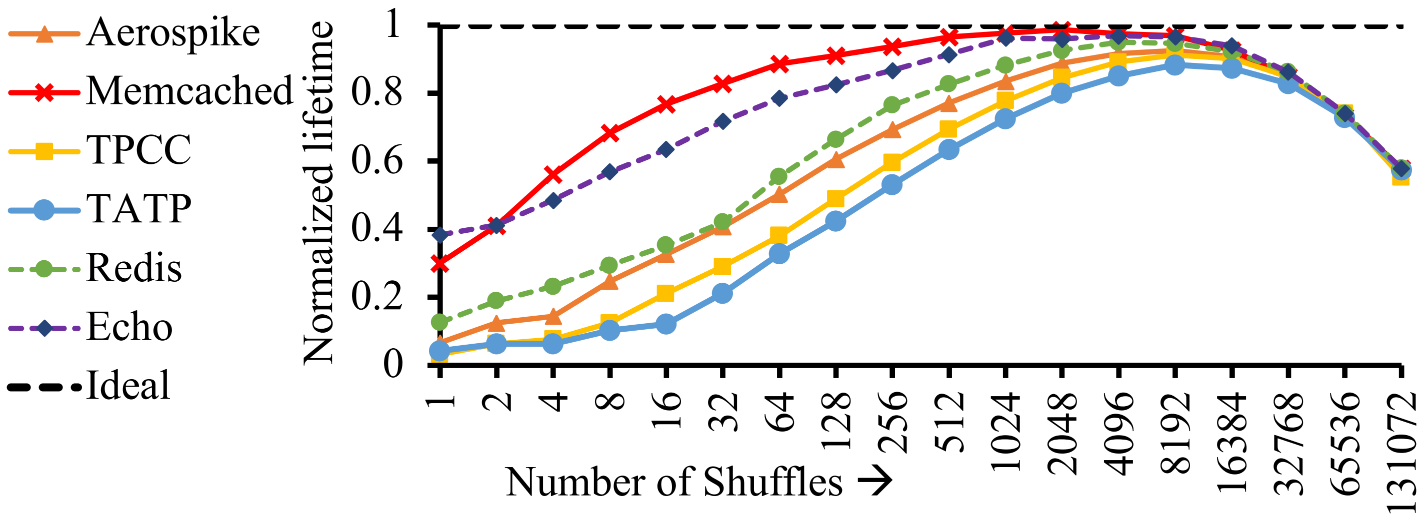
- Using **analytical framework** to determine no. of shuffles
  - Get write traces of applications using instrumentation
  - Evaluate wear to pages as number of shuffles increase

More details in the paper!

PM lifetime due to random shuffles

- Using **analytical framework** to determine no. of shuffles
  - Get write traces of applications using instrumentation
  - Evaluate wear to pages as number of shuffles increase

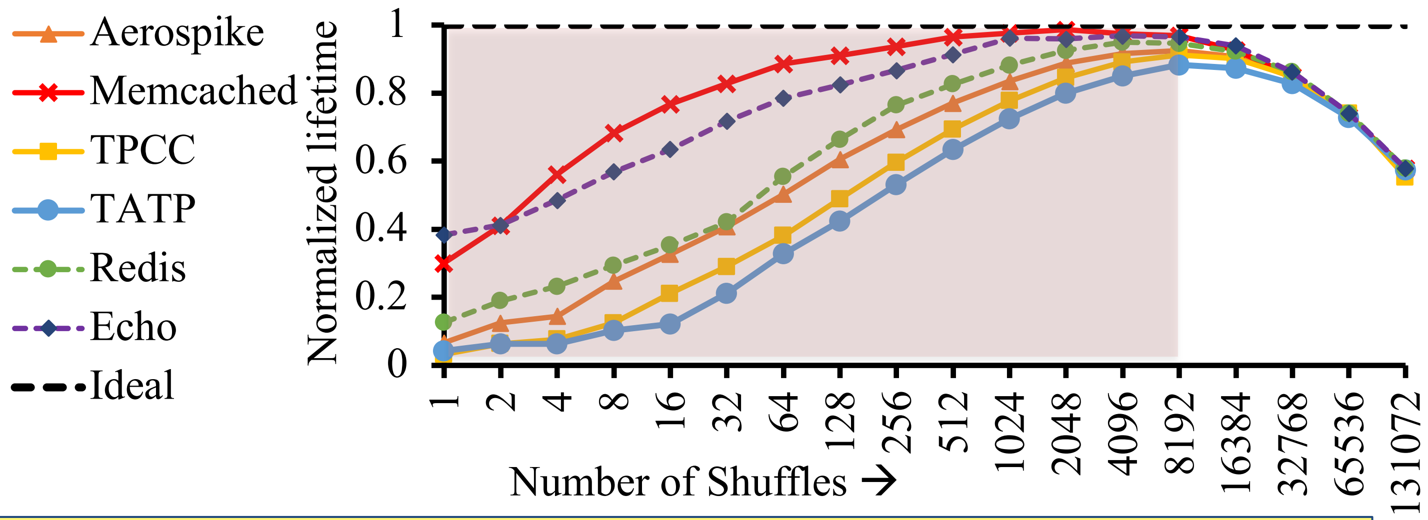
More details in the paper!



PM lifetime due to random shuffles

- Using **analytical framework** to determine no. of shuffles
  - Get write traces of applications using instrumentation
  - Evaluate wear to pages as number of shuffles increase

More details in the paper!

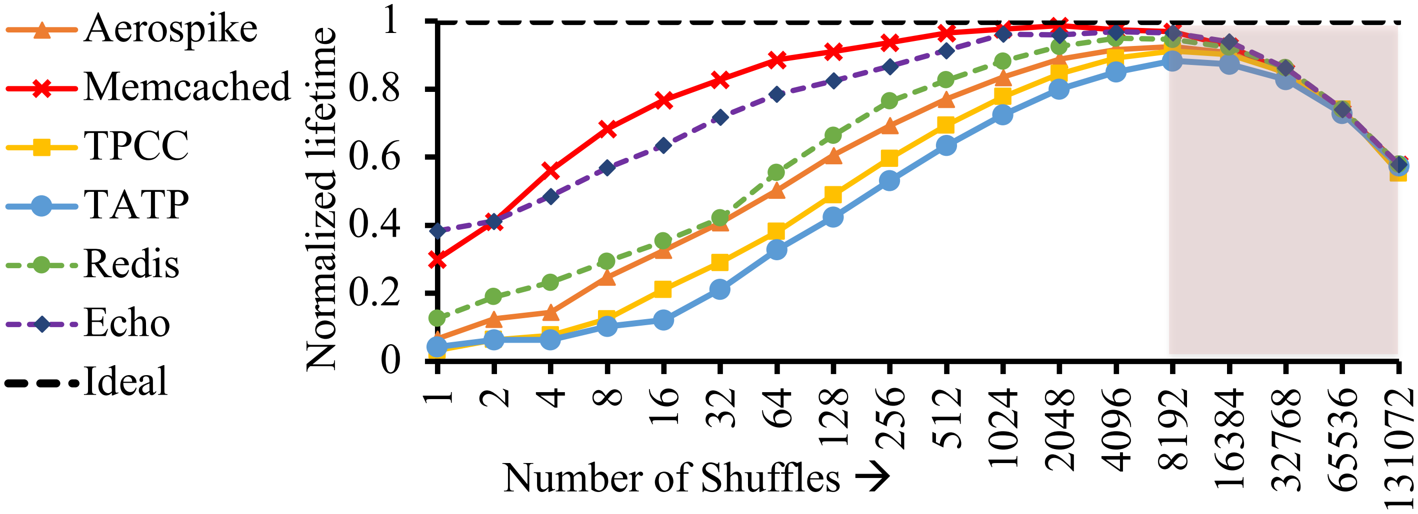


Lifetime improves with the increasing number of shuffles < 8192

PM lifetime due to random shuffles

- Using **analytical framework** to determine no. of shuffles
  - Get write traces of applications using instrumentation
  - Evaluate wear to pages as number of shuffles increase

More details in the paper!

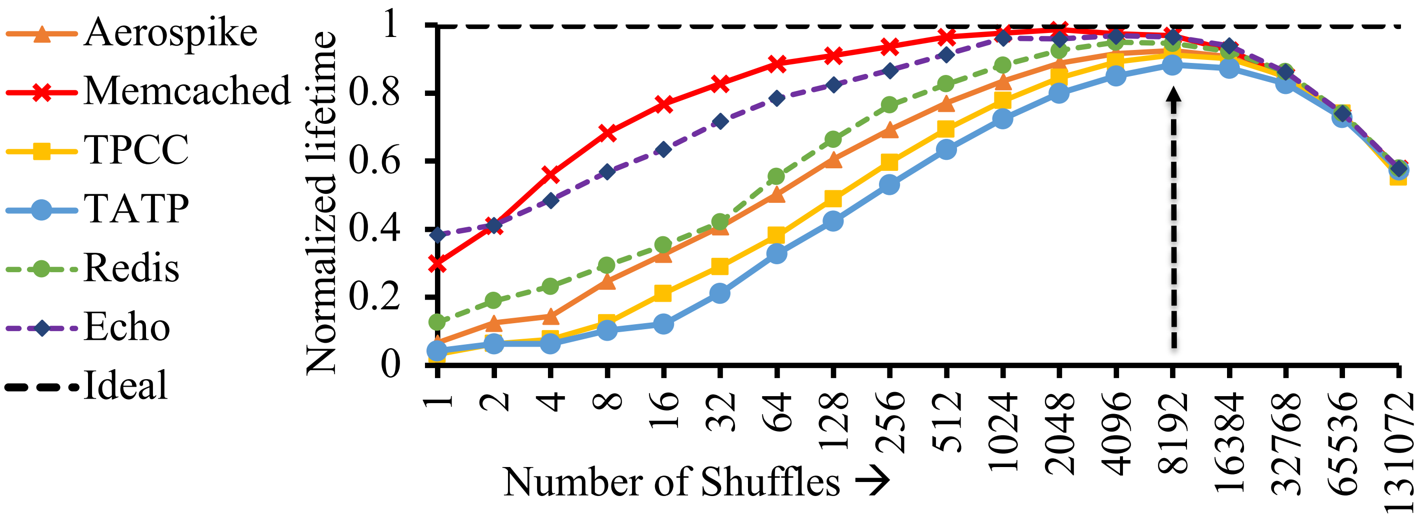


Writes due to shuffles dwarf application writes for > 8192 shuffles

PM lifetime due to random shuffles

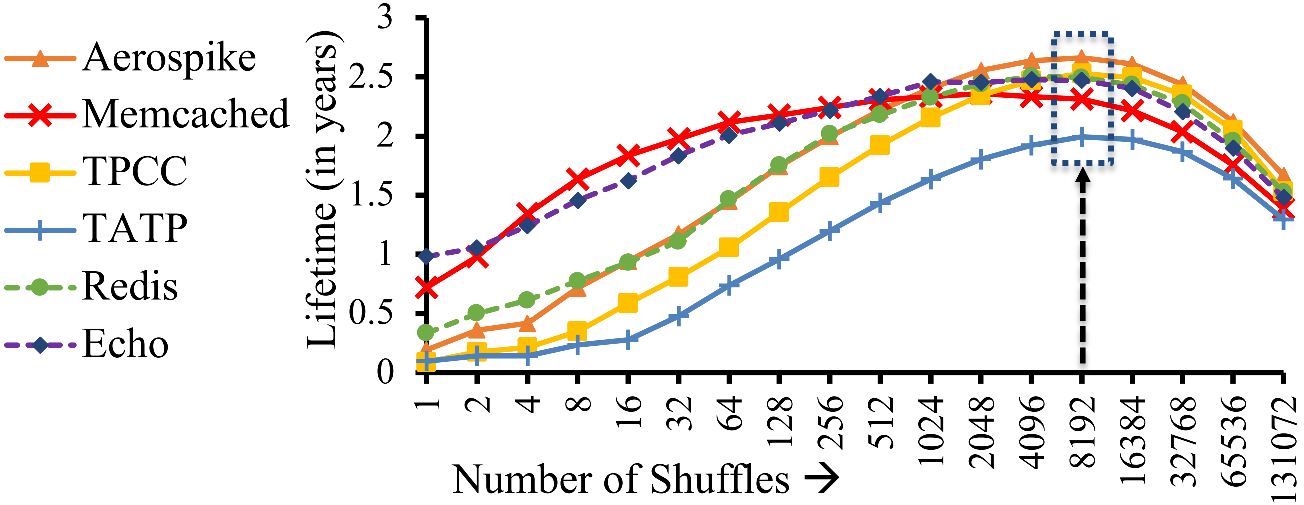
- Using **analytical framework** to determine no. of shuffles
  - Get write traces of applications using instrumentation
  - Evaluate wear to pages as number of shuffles increase

More details in the paper!



Kevlar achieves 94% ideal-wear with 8192 shuffles over PM lifetime

Wear leveling alone is not enough



- Wear leveling improves PM lifetime to 2.0 – 2.8 years
  - Insufficient to meet system lifetime targets (eg. 4 or 6 years)

Lifetime achieved due to wear leveling alone is limited by PM write bandwidth

MichiganEngineering

arm

Wear reduction in Kevlar

- Improves PM lifetime to a **configurable target**
- Limits PM write bandwidth to meet lifetime target

Eg. For desired lifetime = 4yrs, PM endurance = 10^7:

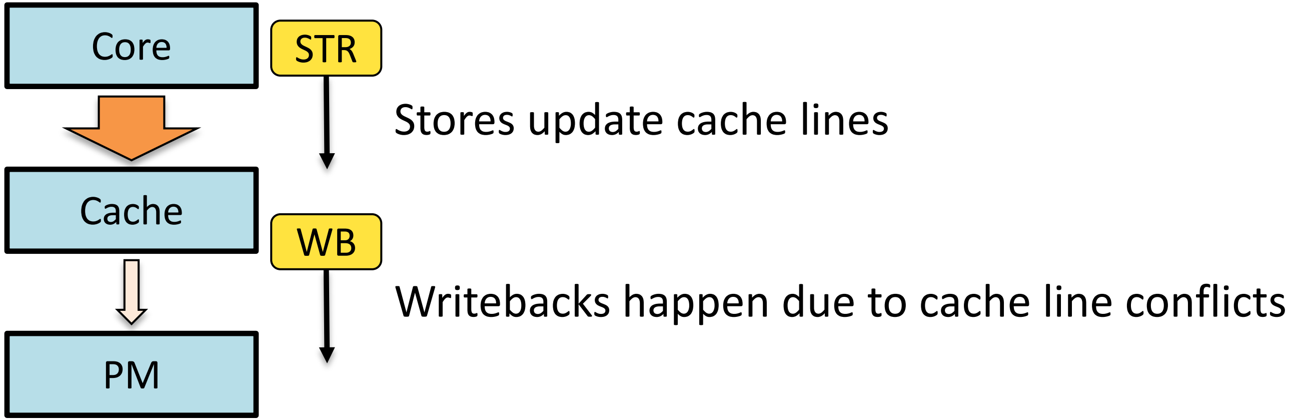
$$PM\_bandwidth = \frac{Endurance \times n\_pages}{Lifetime} = \textbf{20K writes/sec/GB}$$

- Performs **page migrations** to high endurance memory

Kevlar requires per page writeback rate to perform page migrations

Measuring PM page writes is challenging

- PM writes are a result of **cache writebacks**



Existing systems provide no mechanisms to measure per-page writebacks

Modeling caches to measure PM writebacks

- Precise modeling of caches in software expensive
- Kevlar builds an approximate cache model

Observe stores using hardware performance counters

↓

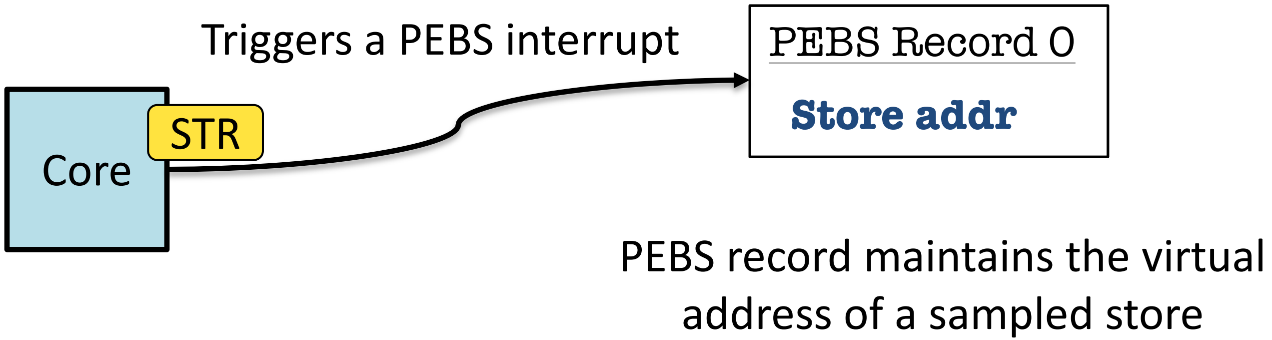
Approximately track set of dirty cache blocks using bloom filter

↓

Estimate PM writebacks using cache blocks in bloom filter

Using PEBS to sample stores

- Employs Intel's Precise-Event-Based-Sampling (PEBS) counters
- Configures PEBS to record arch. state for retiring stores

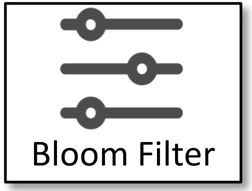


Optimization: Samples one every 17th stores to reduce monitoring overhead

Kevlar approximates caches to estimate wear

- Estimates temporal locality in application's access pattern
- Uses bloom filter to track dirty blocks in hardware cache

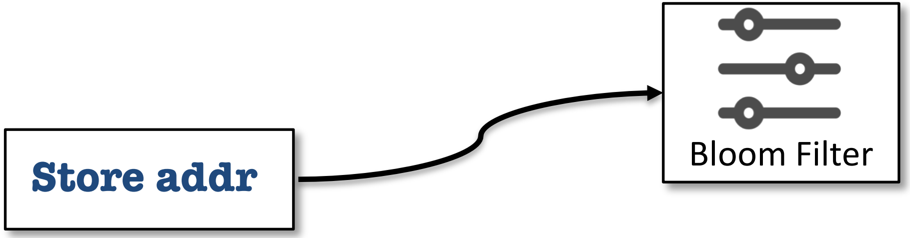
1. Sizes bloom filter to match LLC size



Kevlar approximates caches to estimate wear

- Estimates temporal locality in application’s access pattern
- Uses bloom filter to track dirty blocks in hardware cache

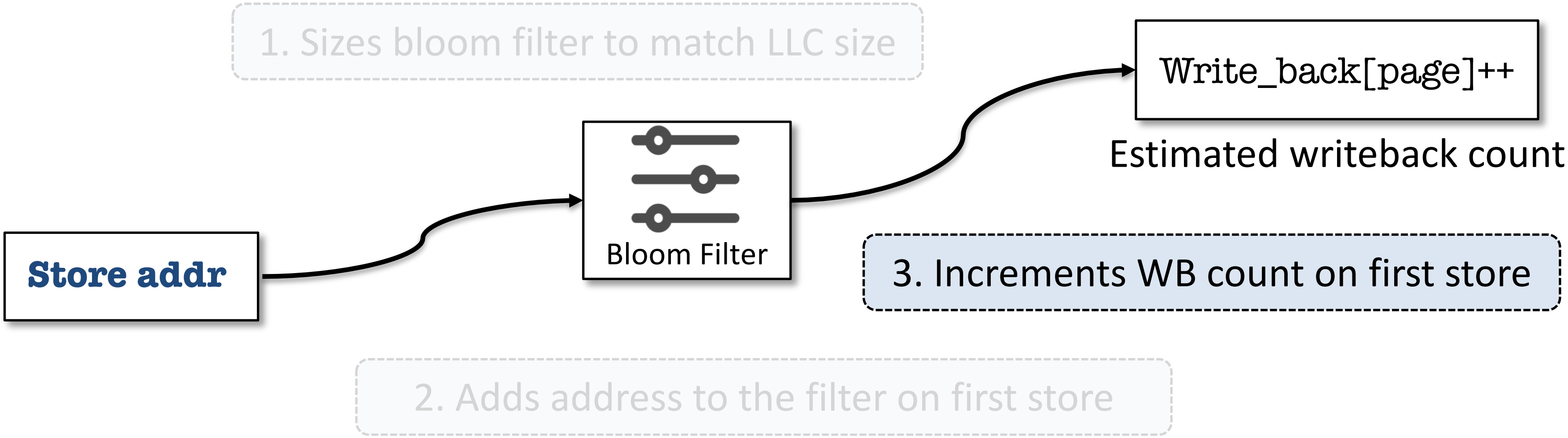
1. Sizes bloom filter to match LLC size



2. Adds address to the filter on first store

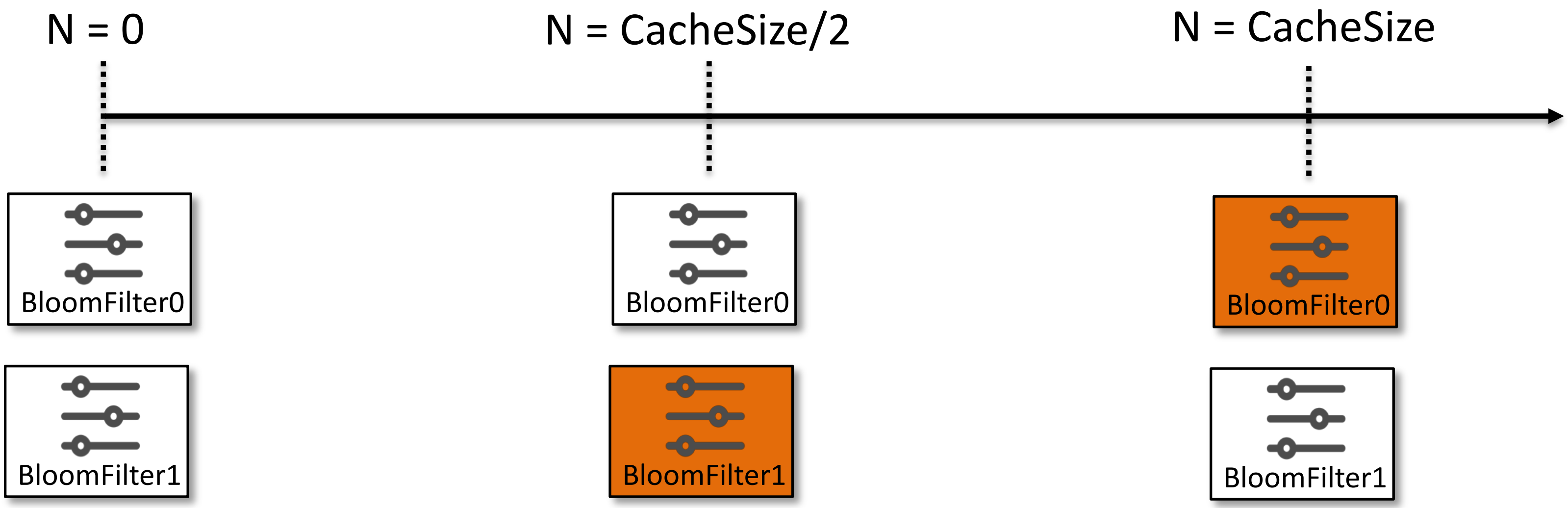
Kevlar approximates caches to estimate wear

- Estimates temporal locality in application's access pattern
- Uses bloom filter to track dirty blocks in hardware cache



Bloom filters cleared when they are full

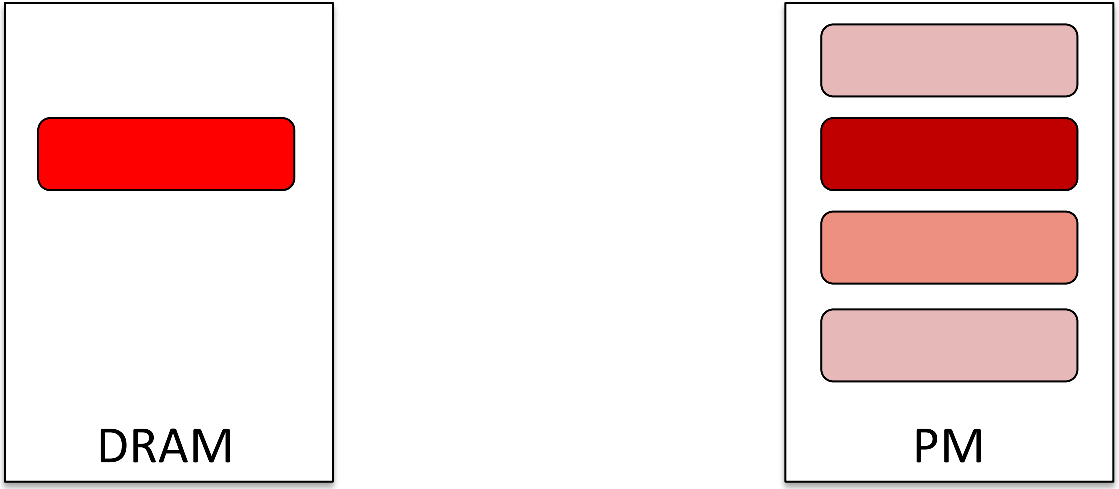
- Maintains number of cache blocks equal to size of last-level cache
  - Clearing bloom filter causes false spike in measured writebacks



Kevlar uses estimated writebacks per page to perform page migrations

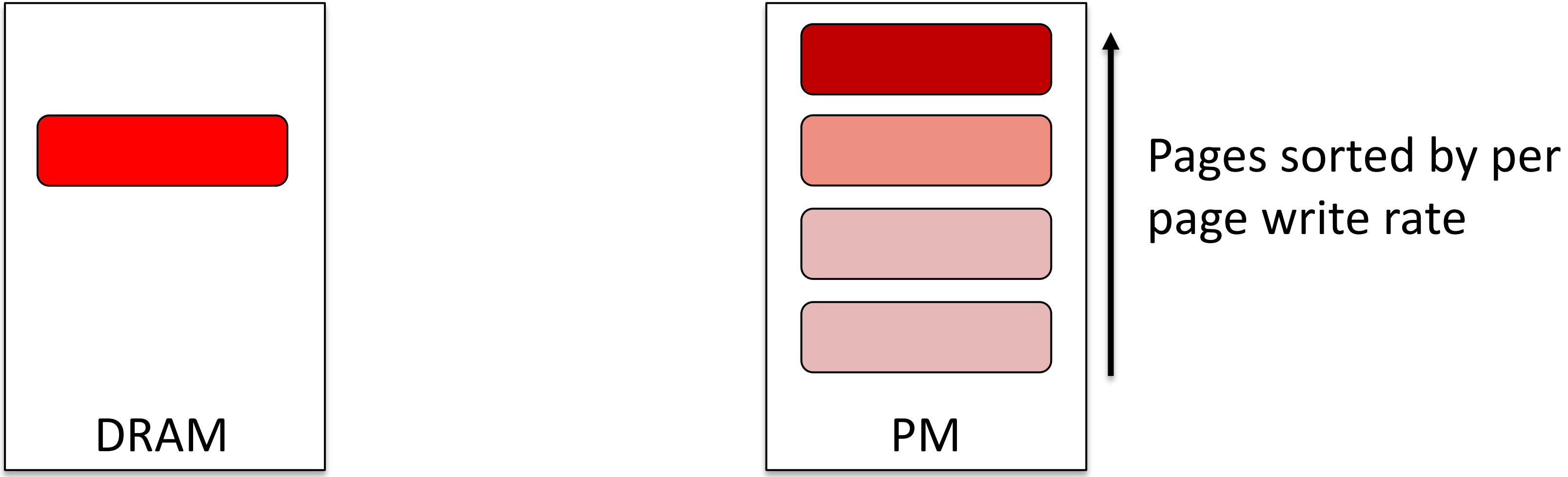
Kevlar migrates heavily written pages to DRAM

- Limits PM write bandwidth to 20K writes/sec for 4 year lifetime target
- Migrates top 10% freq. written pages to DRAM



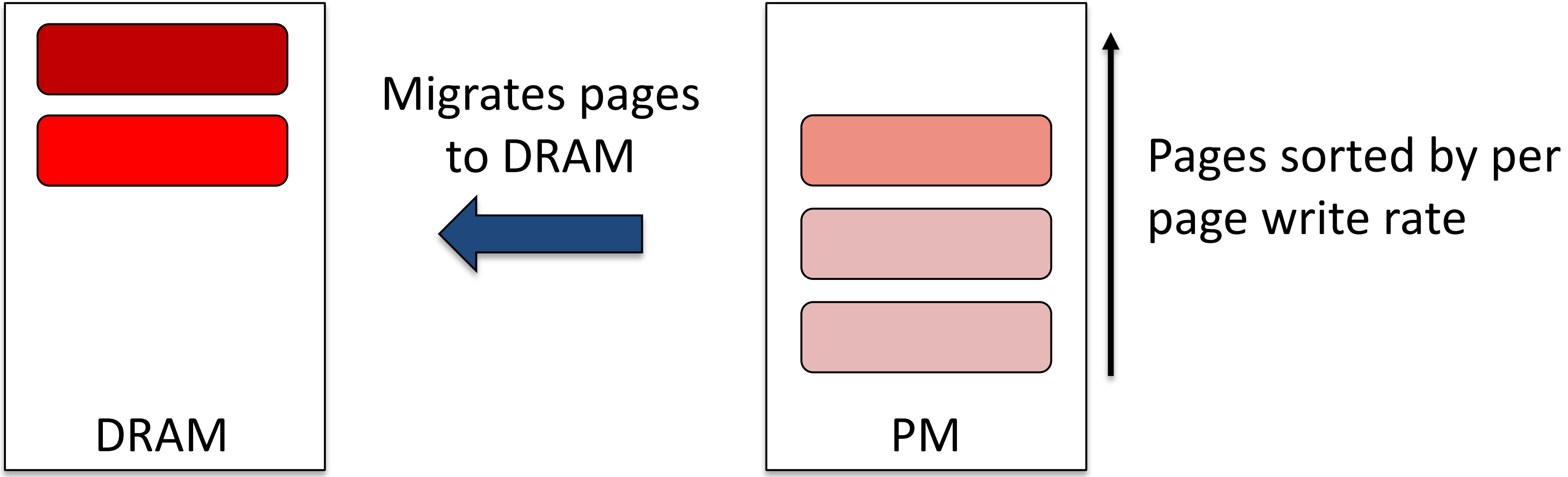
Kevlar migrates heavily written pages to DRAM

- Limits PM write bandwidth to 20K writes/sec for 4 year lifetime target
- Migrates top 10% freq. written pages to DRAM



Kevlar migrates heavily written pages to DRAM

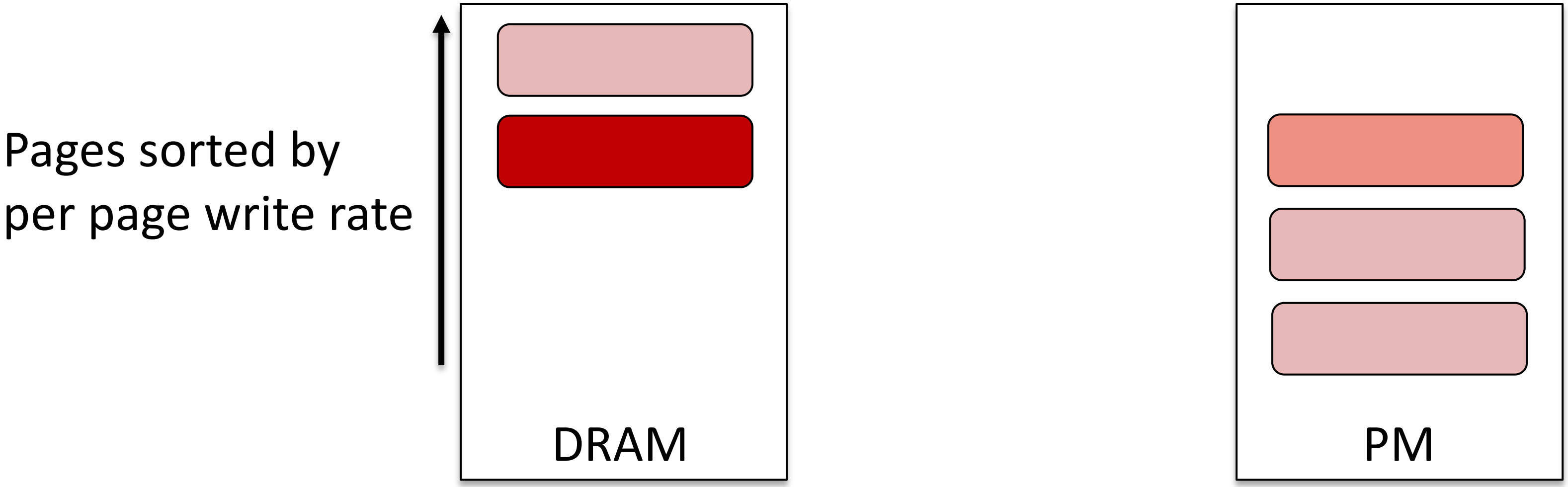
- Limits PM write bandwidth to 20K writes/sec for 4 year lifetime target
- Migrates top 10% freq. written pages to DRAM



Optimization: Kevlar disables PEBS counters when write rate is < 20K writes/sec

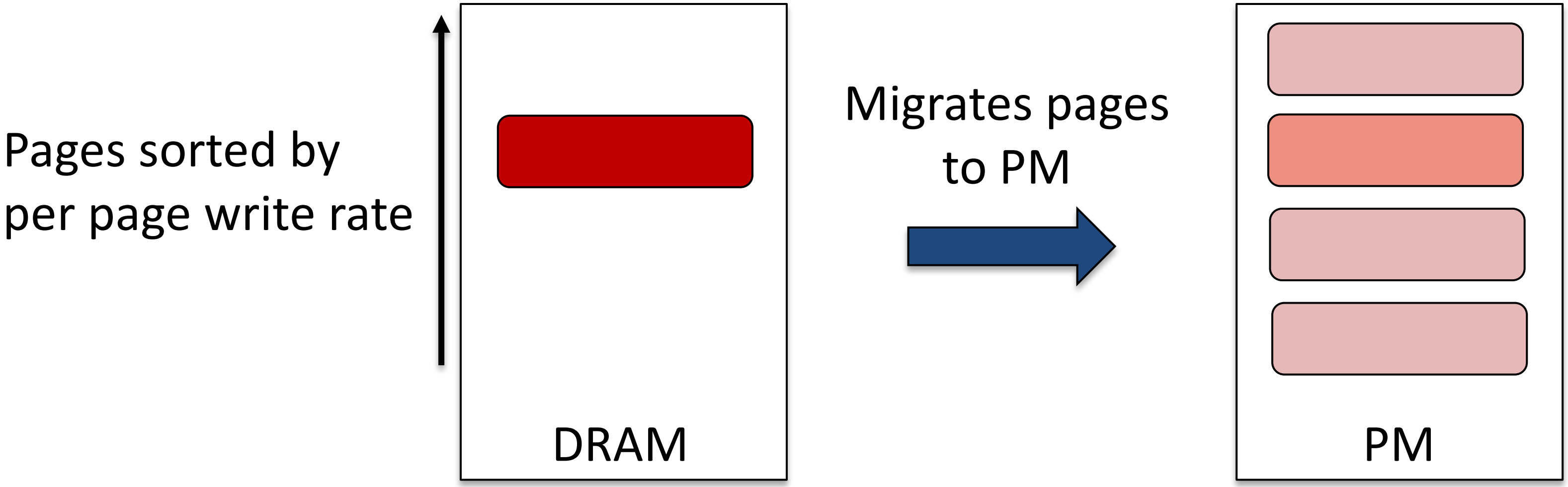
Kevlar detects changes in access pattern

- Detects PM write rate below 20K writes/sec for 5 consecutive intervals
- Re-enables PEBS monitoring to migrate least 10% written pages to PM



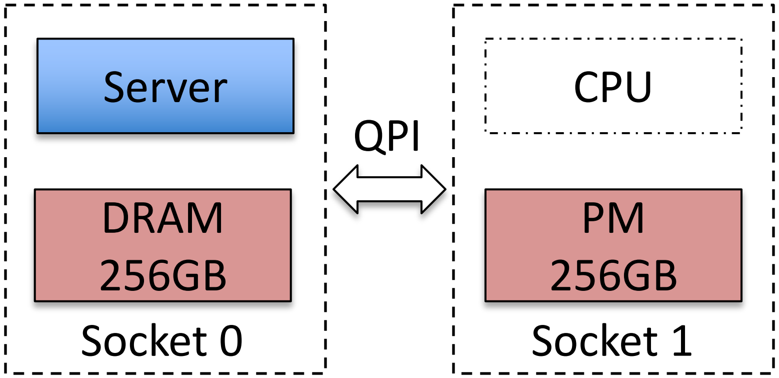
Kevlar detects changes in access pattern

- Detects PM write rate below 20K writes/sec for 5 consecutive intervals
- Re-enables PEBS monitoring to migrate least 10% written pages to PM

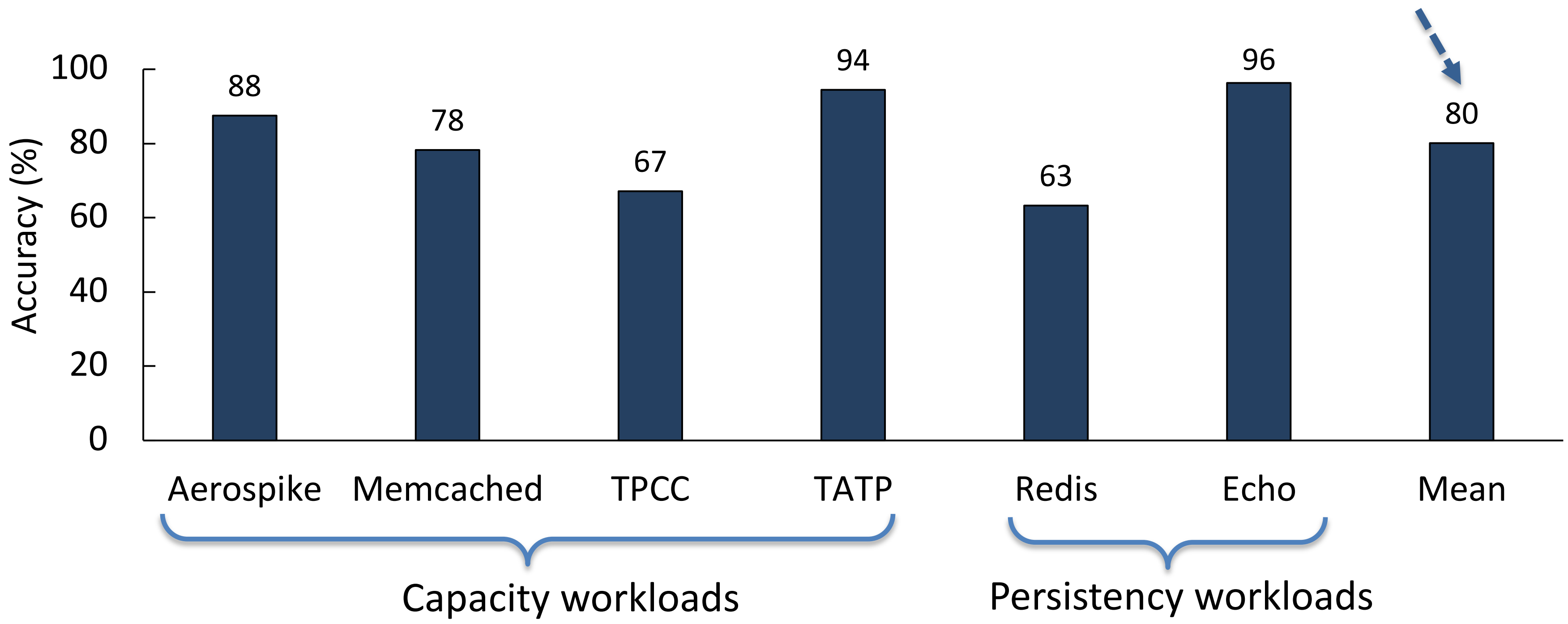


Methodology

- Prototyped in Linux 4.5
- Intel Xeon E5-2699 v3, 72 hardware threads
- Caches: 32KB L1 D&I, 256KB L2, 45MB LLC
- Linux cgroups to isolate cores and memory for server threads
- PM fails after 1% pages suffer 10^7 writes

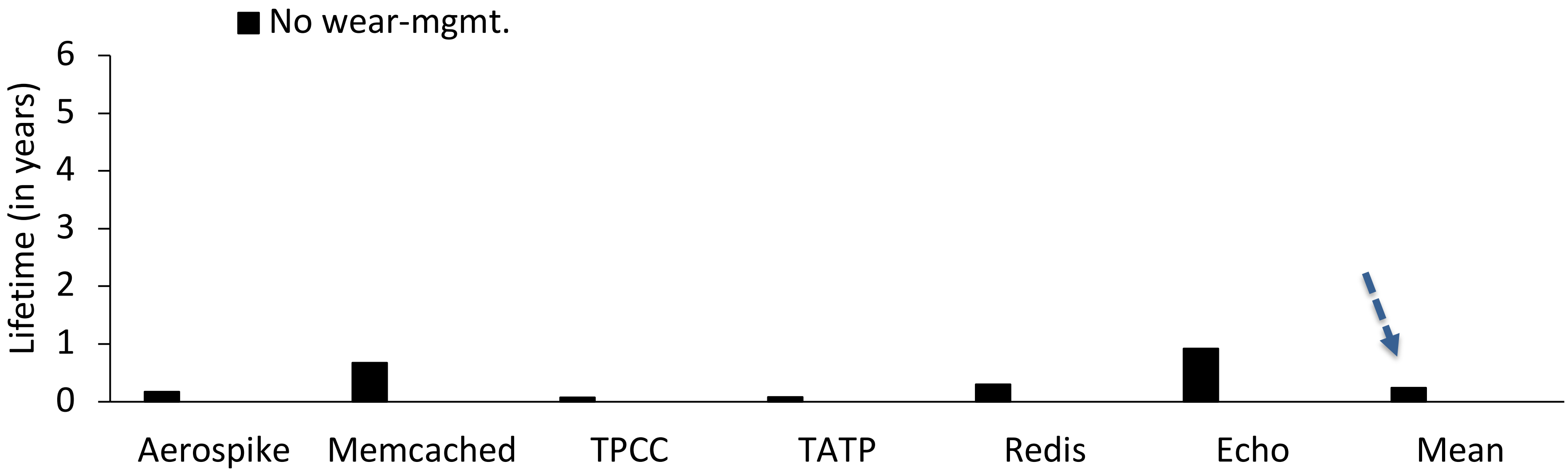


Accuracy of wear estimation



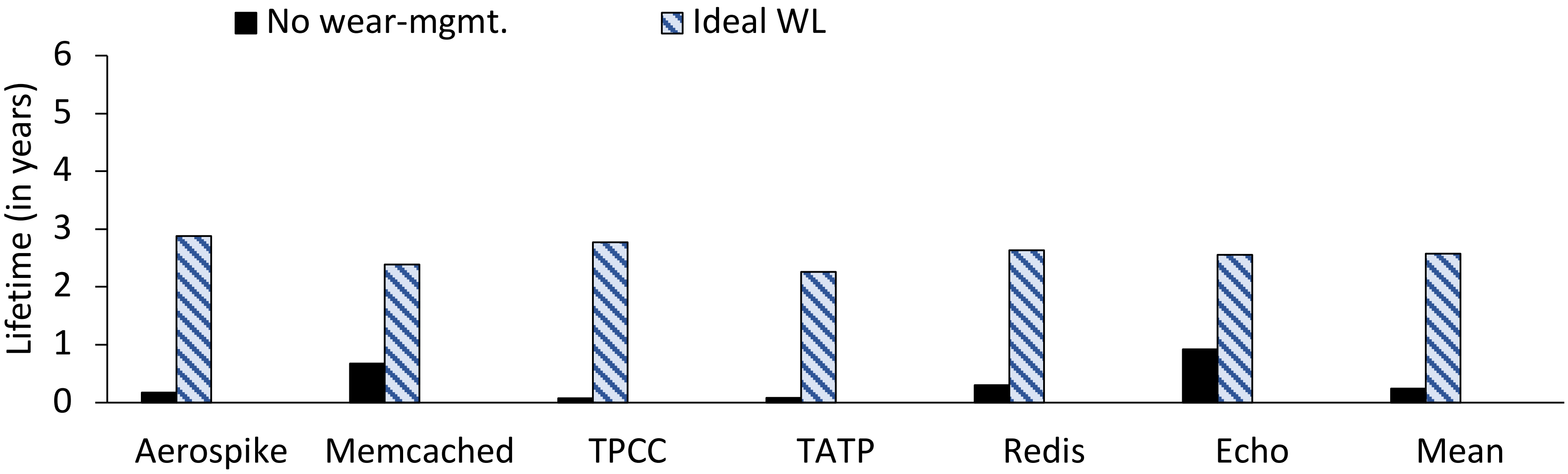
Kevlar can correctly detect 80% of top 10% heavily written pages in PM

PM device lifetime



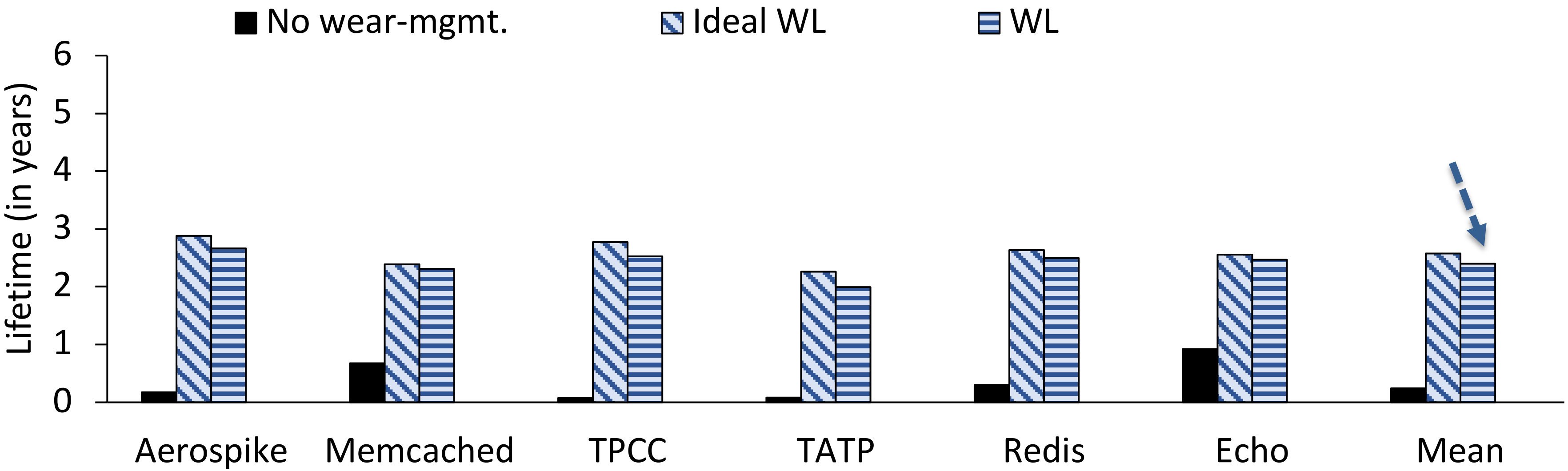
PM wears out in 1.1 months in absence of wear-management mechanisms

PM device lifetime



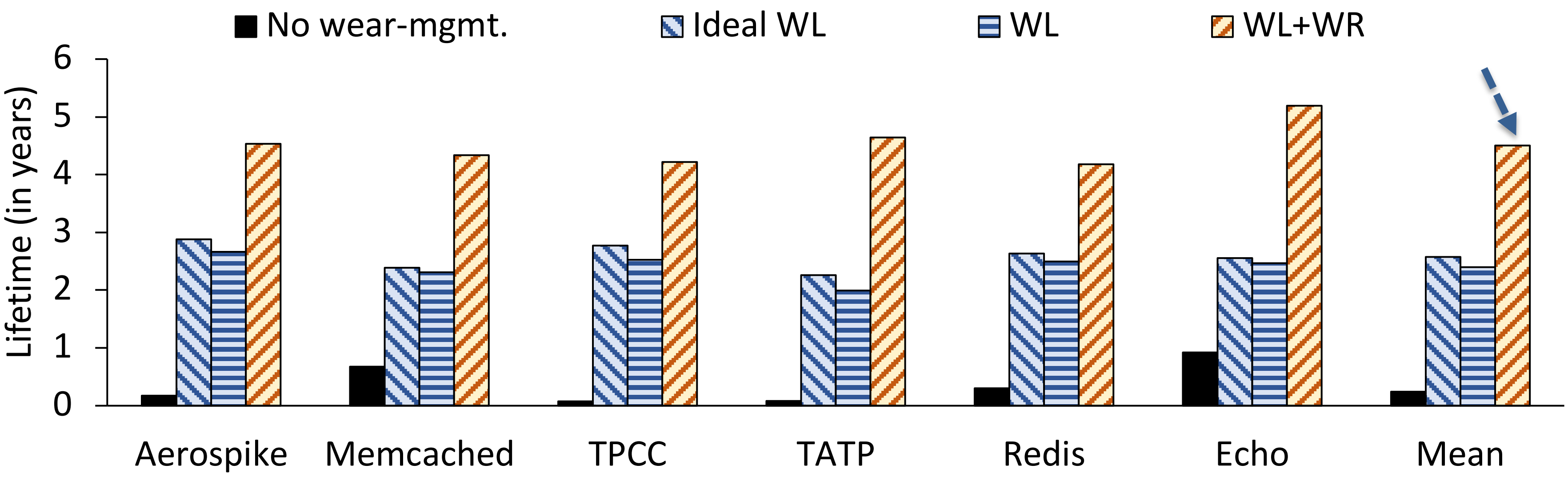
Ideal wear leveling shows lifetime for an oracle design that achieves uniform wear

PM device lifetime



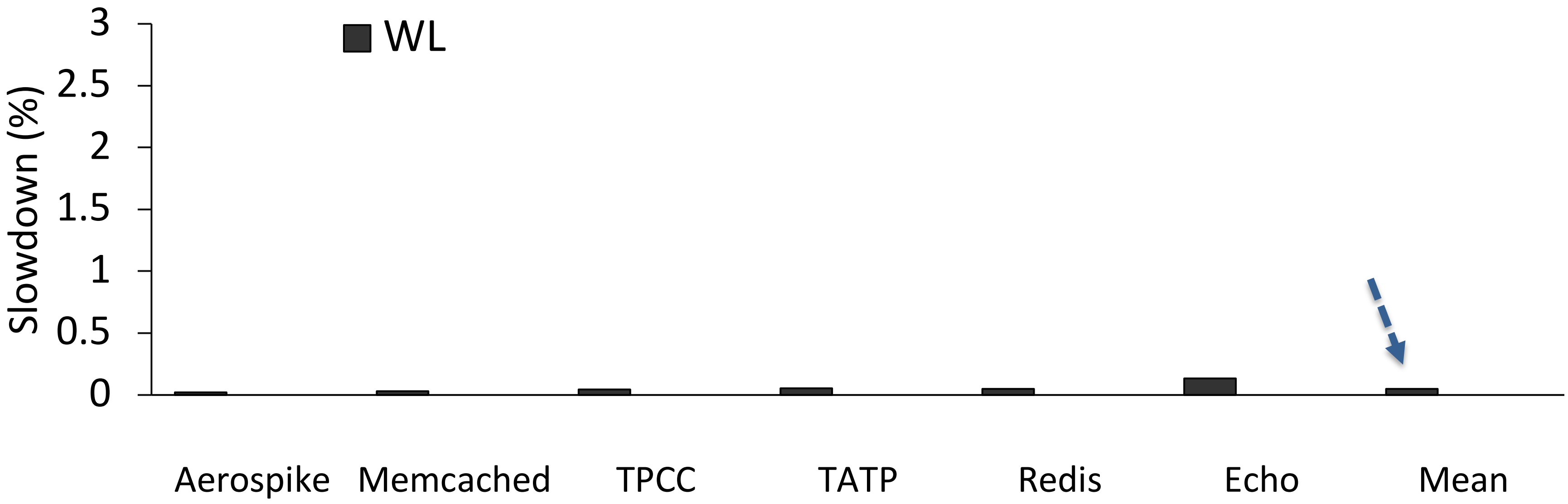
Kevlar improves PM lifetime by 9.8x as compared to the design without wear-mgmt.

PM device lifetime



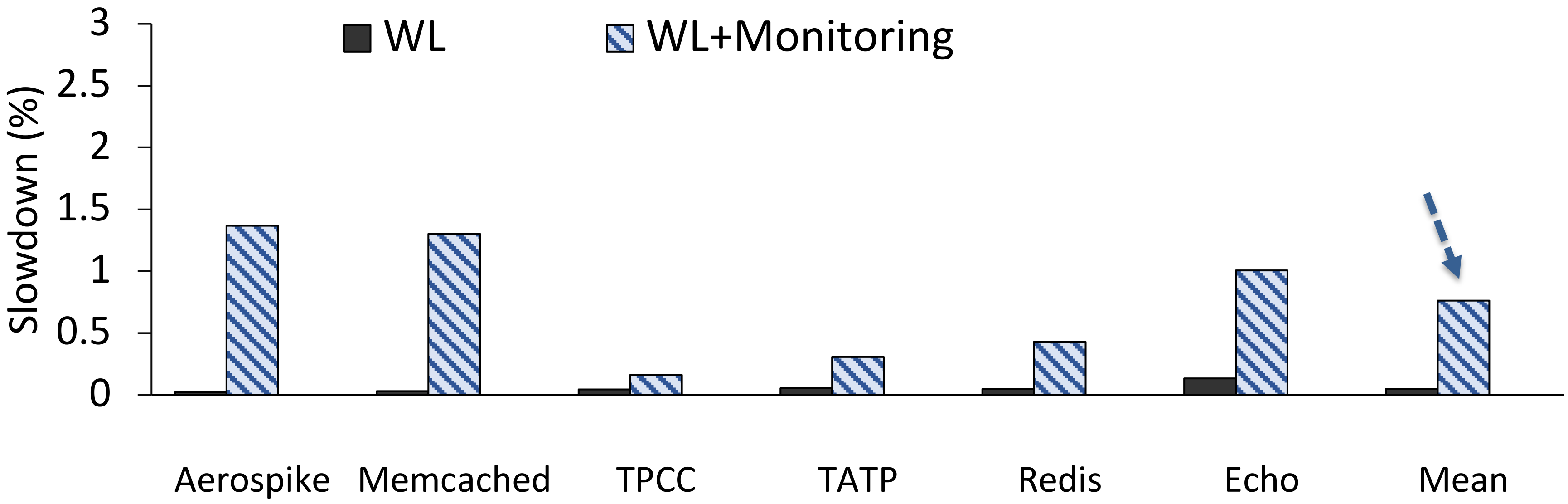
Kevlar limits PM write bandwidth to achieve lifetime target of 4 years

Kevlar performance overhead



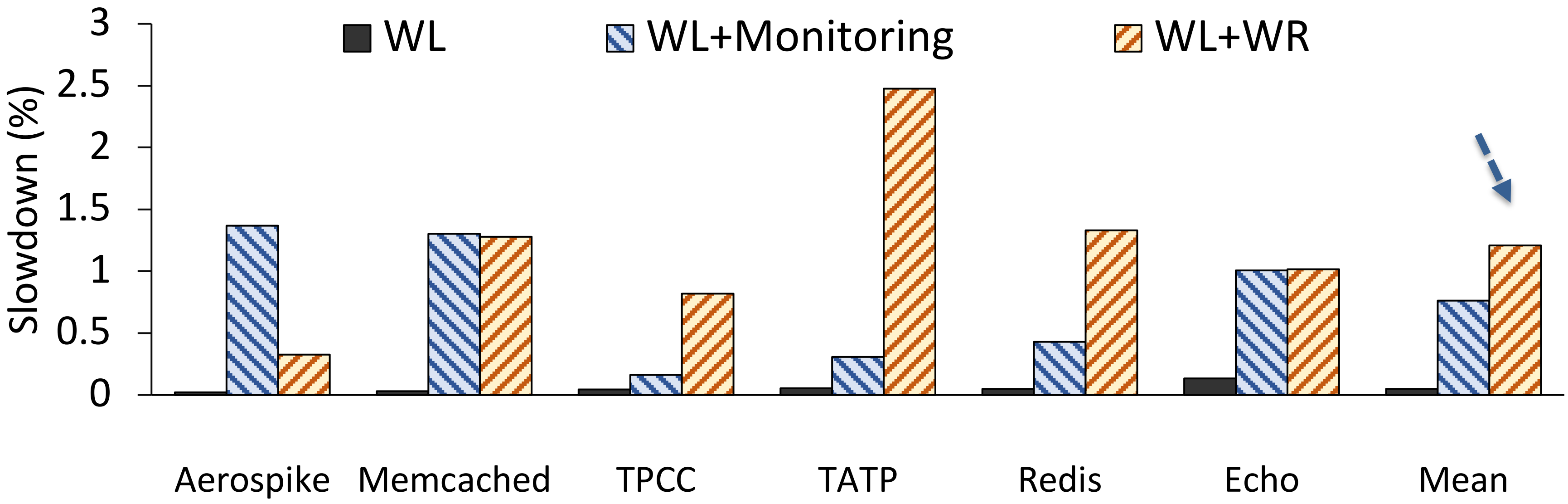
Wear leveling alone incurs a negligible performance overhead of 0.04%

Kevlar performance overhead



Kevlar's monitoring based on PEBS counters incur a performance overhead of 0.8% (avg.)

Kevlar performance overhead



Kevlar additionally incurs a 1.2% slowdown due to page migrations between DRAM and PM

MichiganEngineering

arm

Conclusion

Analytical framework	Simple remaps achieve near-ideal wear
⇩	
Wear leveling	Remaps pages in PM
Wear estimation	Estimates per page wear
⇩	
Wear reduction	Performs page migrations

Simple software mechanisms achieve **> 4yr lifetime** with **1.2% perf. overhead**

Software Wear Management for Persistent Memories

Vaibhav Gogte, William Wang[1], Stephan Diestelhorst[1], Aasheesh Kolli[2,3], Peter M. Chen, Satish Narayanasamy, Thomas F. Wenisch

University of Michigan

PennState [2]

arm [1]

vmware [3]

FAST'19

02/26/2019