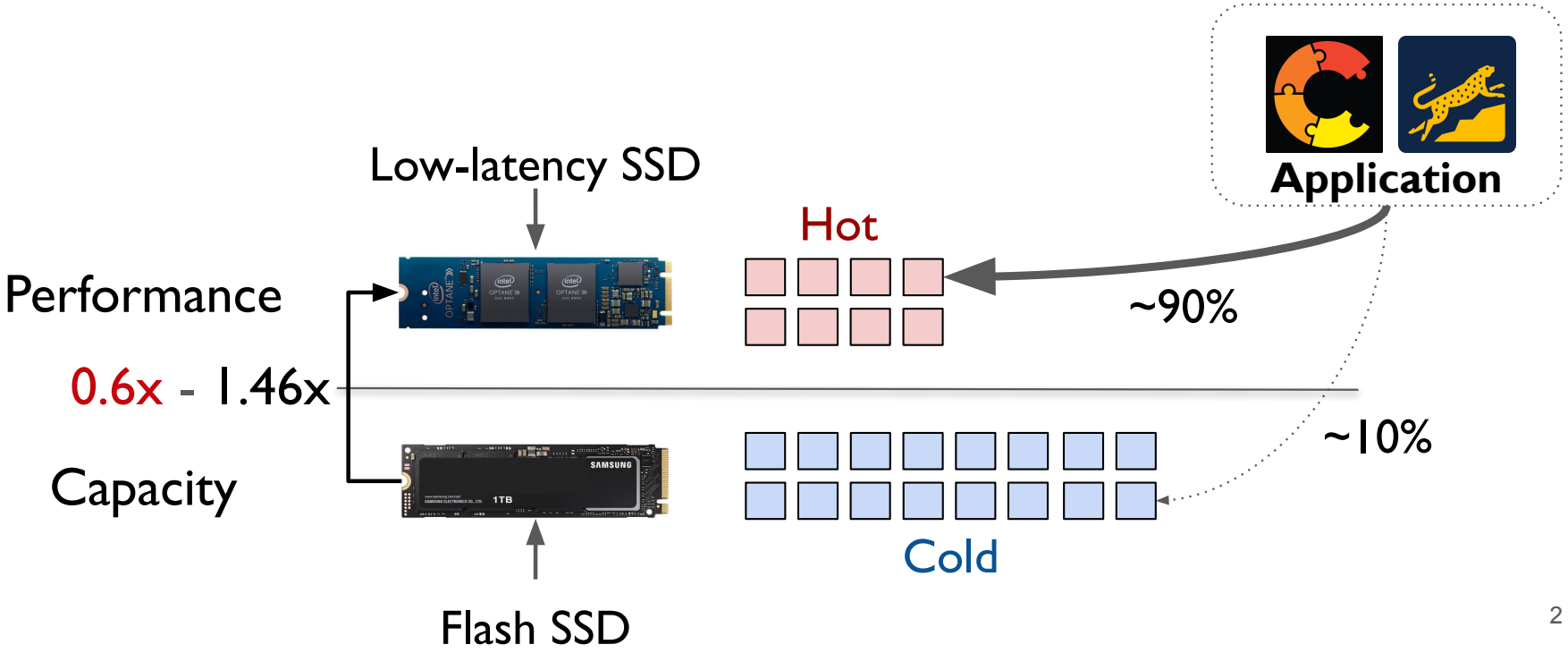
Getting the MOST out of your Storage Hierarchy with Mirror-Optimized Storage Tiering

Kaiwei Tu, Kan Wu, Andrea Arpaci-Dusseau and Remzi Arpaci-Dusseau

Storage Hierarchy is Less Hierarchical

Capacity tier is severely **underutilized**

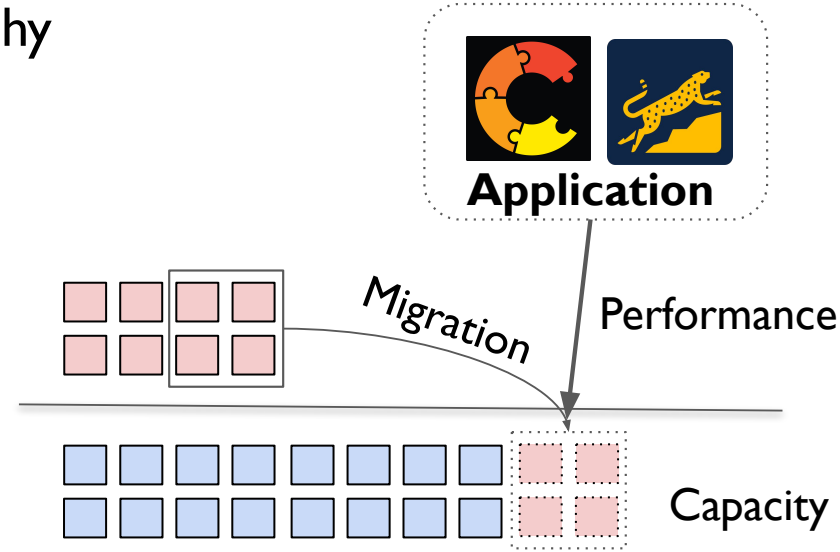


Why does Migration not work?

Existing approaches rely on migration (e.g. Colloid[1] and BATMAN[2])

Migration is expensive in storage hierarchy

- ❌ Interferes with frontend application
- ❌ Hard to handle bursty workload
- ❌ Reduces device lifetime



Question: Can we achieve load balancing without paying high cost of migration?

Mirror-Optimized Storage Tiering (MOST)

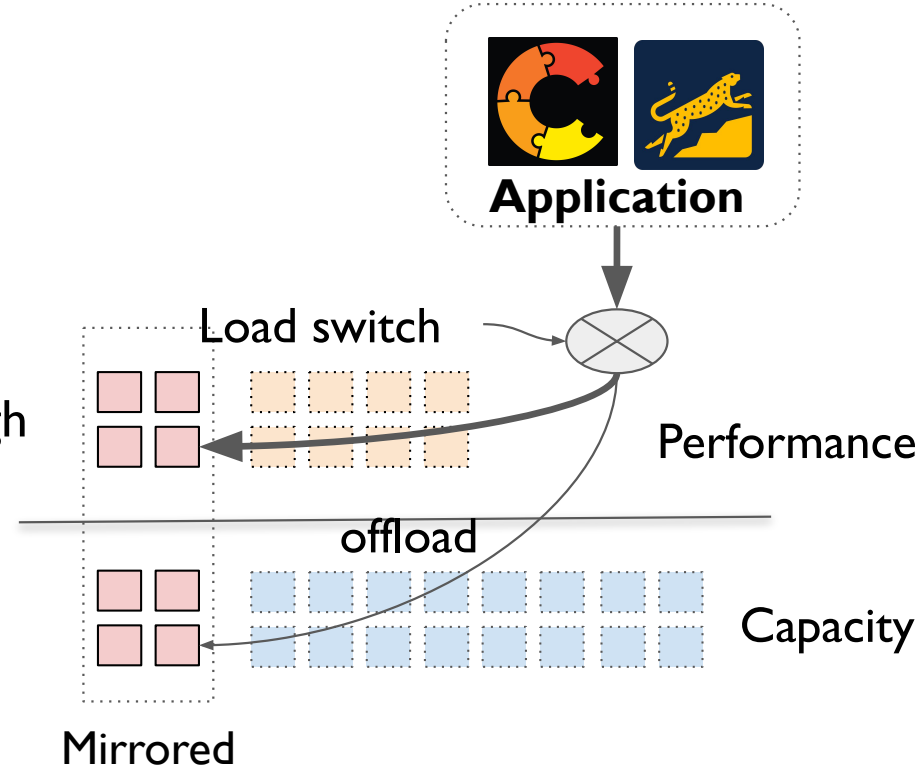
Cross-tier **mirroring** is the key

Mirroring enables routing-based balancing

- Access pattern is naturally skewed
- Small amount of mirroring is enough

Key Results

- Up to **2.3x** higher throughput
- Reduces writes by up to 84%



Outline

Design: Mirror-Optimized Storage Tiering (MOST)

- MOST Layout
- Read and Write Balancing
- Read/Write Interaction

Implementation & Evaluation

Conclusion

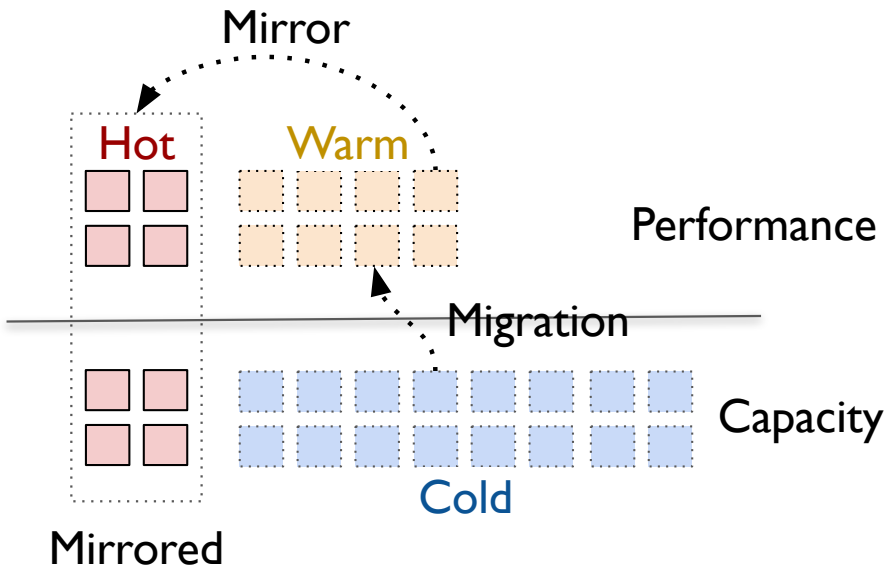
MOST Layout

Data is classified into three classes

- Hot data is mirrored
- Warm and cold data are tiered

Hotness tracking is same as classic approach

- Read/write counter to track hotness



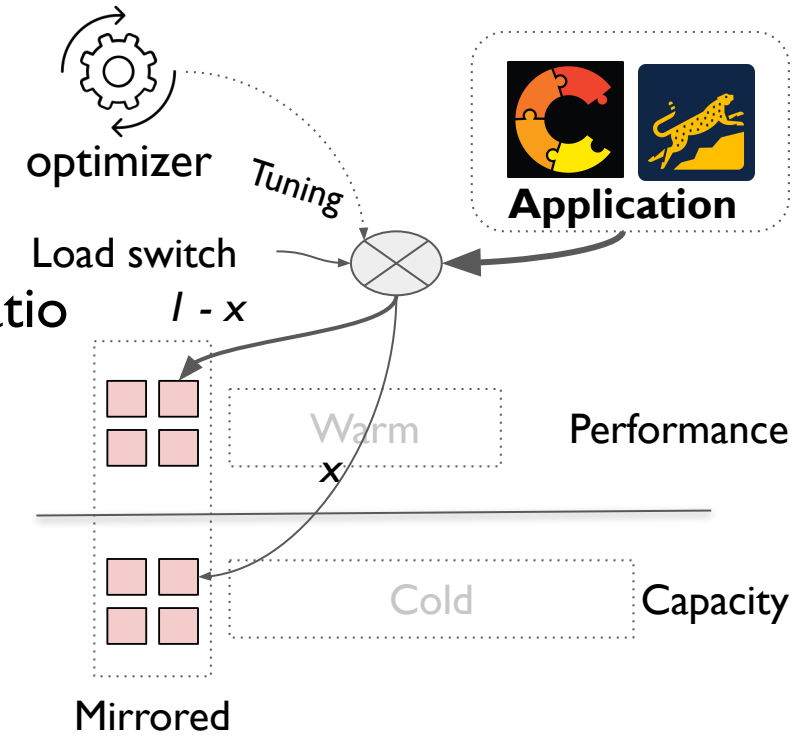
Read Balancing

Load switch: distribute traffic across tiers

- (1 - x)% traffic to performance tier
- x% traffic offloaded to capacity tier

An optimizer dynamically tunes offload ratio

- No prior knowledge of workload/device
- Objective: equalize access latency



Read Balancing

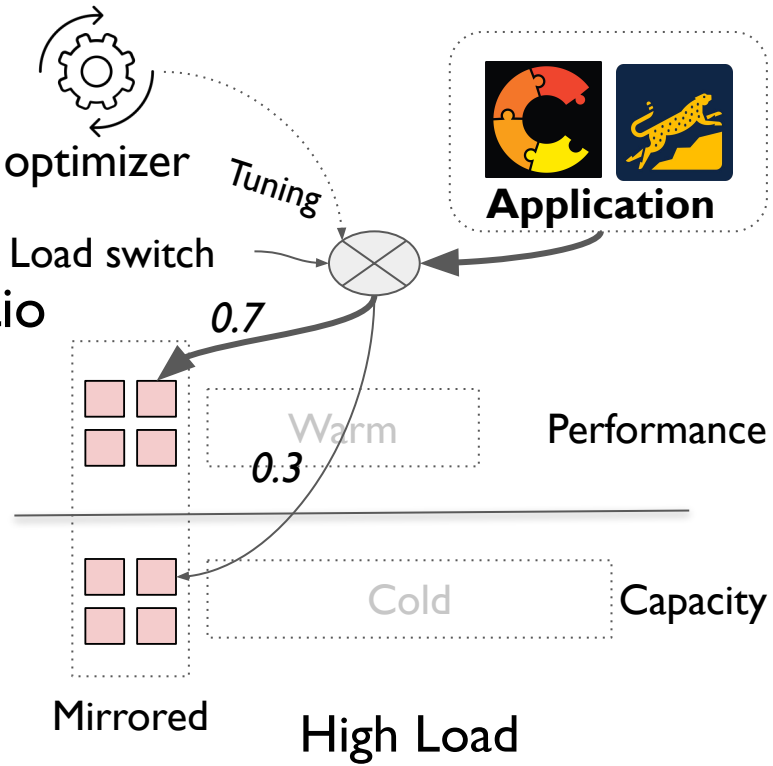
Load switch: distribute traffic across tiers

- (1 - x)% traffic to performance tier
- x% traffic offloaded to capacity tier

An optimizer dynamically tunes offload ratio

- No prior knowledge of workload/device
- Objective: equalize access latency

High load: offload traffic to capacity tier



Read Balancing

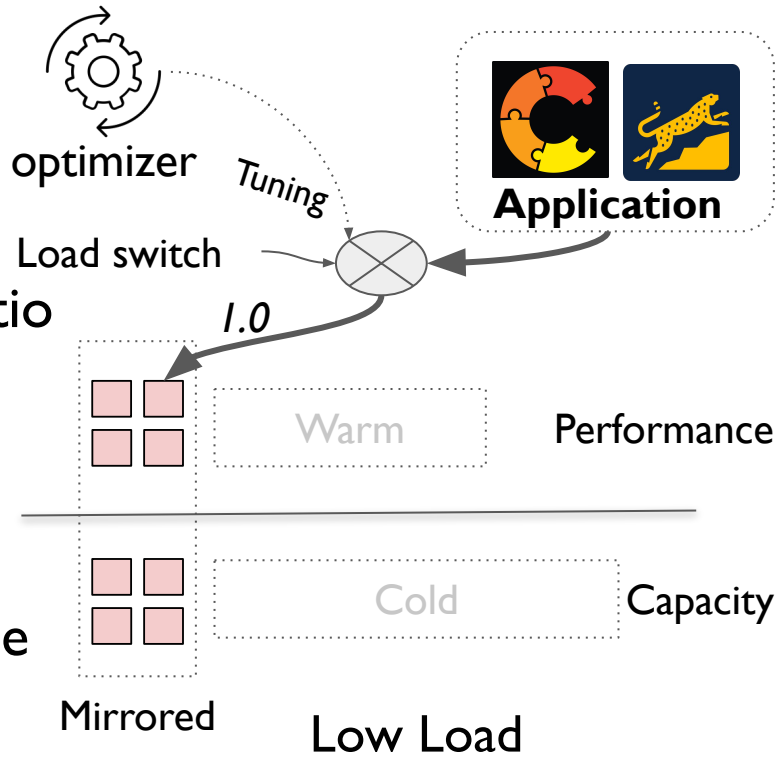
Load switch: distribute traffic across tiers

- (1 - x)% traffic to performance tier
- x% traffic offloaded to capacity tier

An optimizer dynamically tunes offload ratio

- No prior knowledge of workload/device
- Objective: equalize access latency

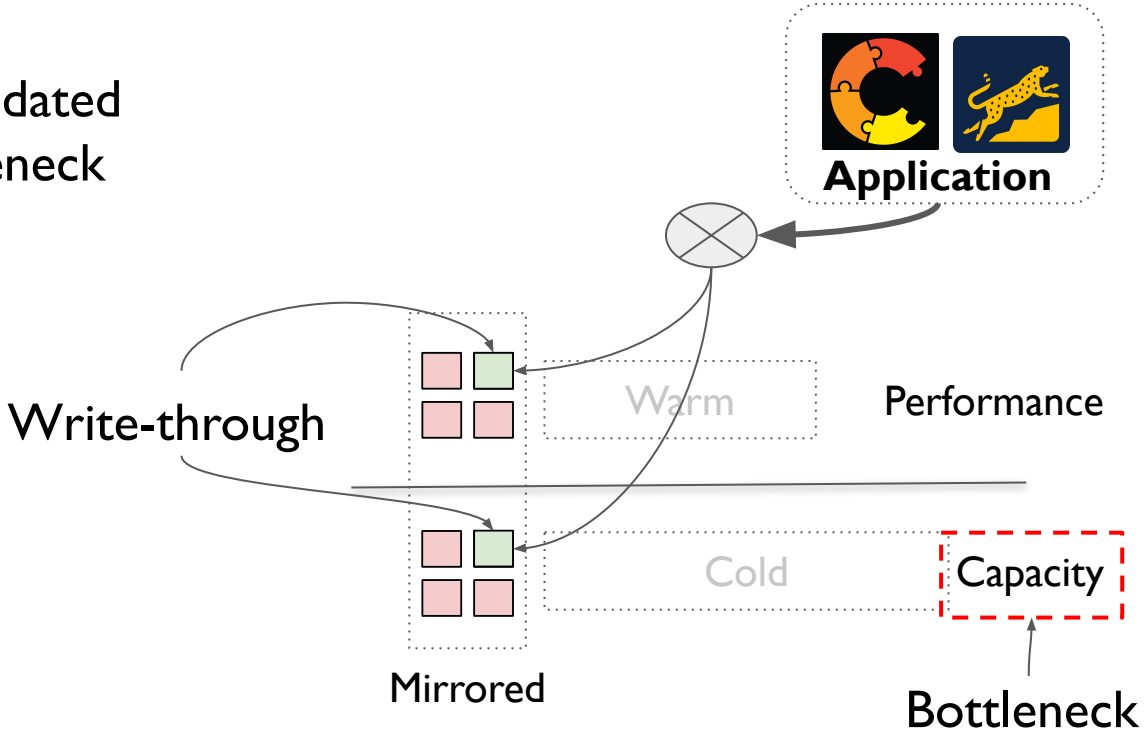
High load: offload traffic to capacity tier
Low load: bring traffic back to performance tier



Write Balancing

Write-through is expensive

- Both copies need to be updated
- Slower device is the bottleneck



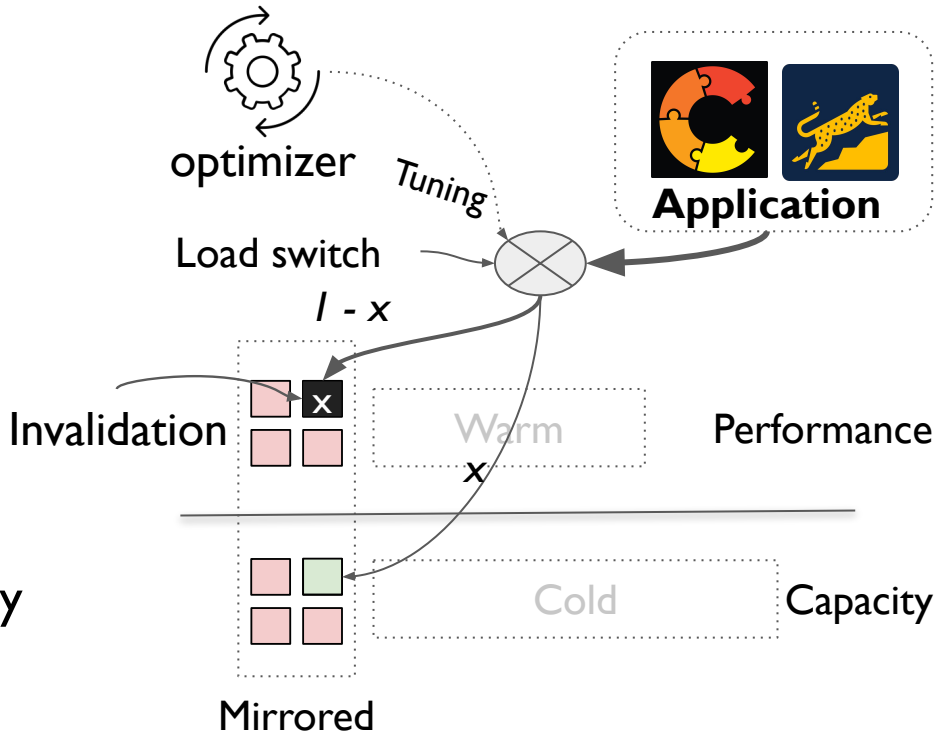
Write Balancing

Write-through is expensive

- Both copies need to be updated
- Slower device is the bottleneck

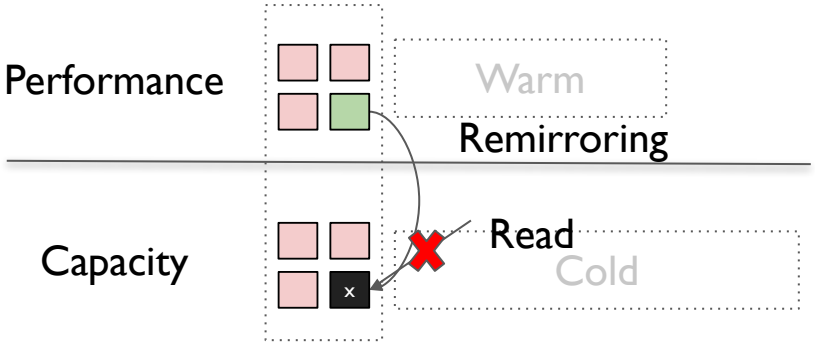
MOST only update one copy

- Write is balanced as read
- BUT writes invalidate another copy



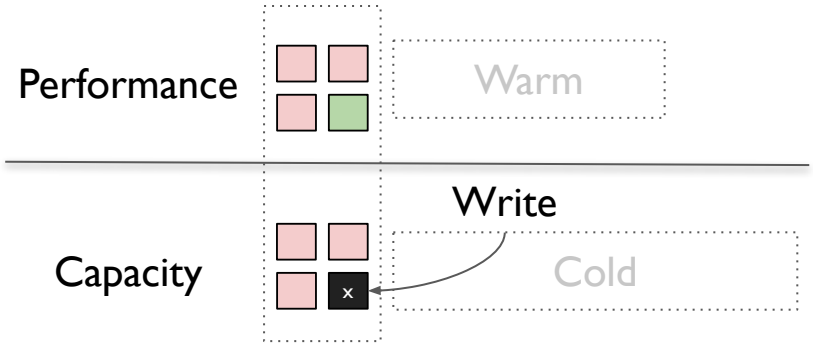
Read/Write Interaction

Write invalidation disables read balancing



Remirroring

Case 1: Read-heavy Data



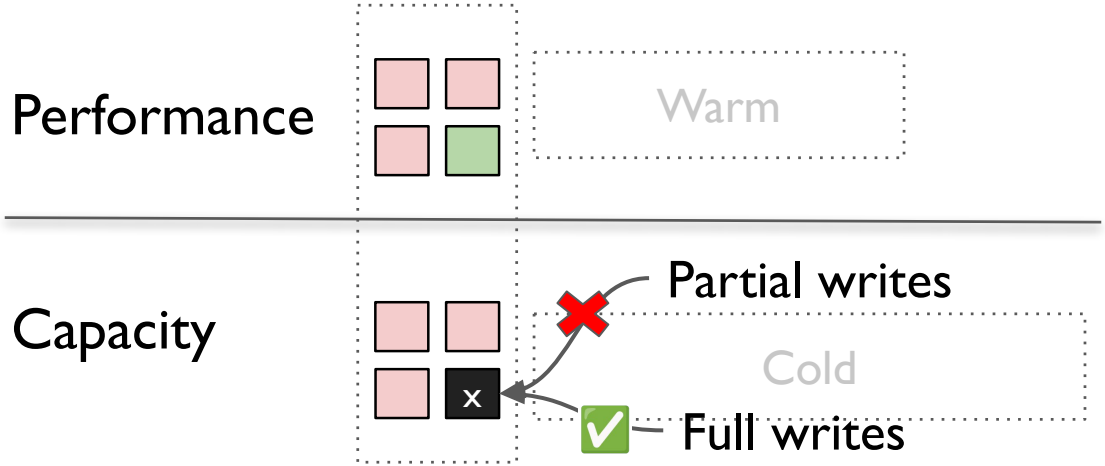
Write As a Balancer

Case 11: Write-heavy Data

Challenge of Partial Writes

✅ Overwrites can be sent to any copy

❌ Partial writes can only be sent to valid copy

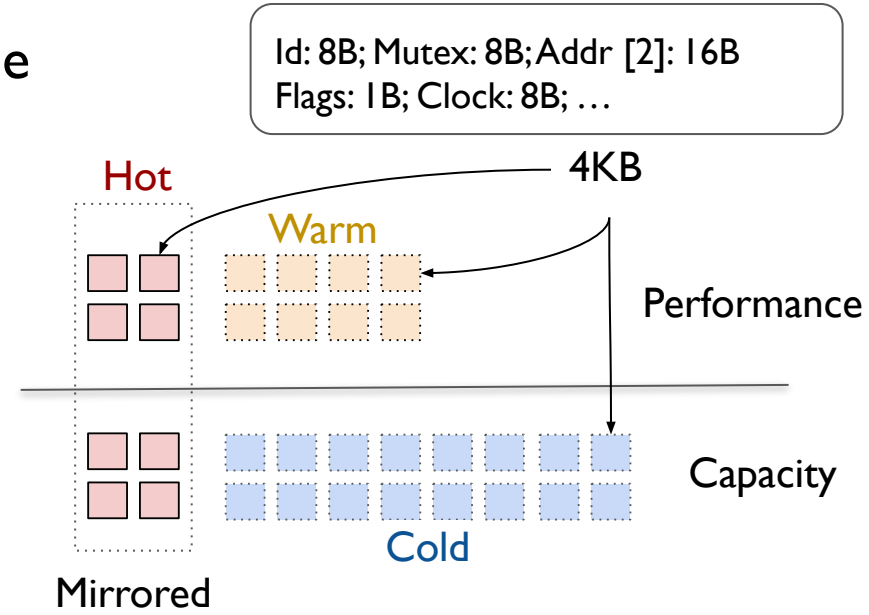


How to Avoid Partial Writes?

Application I/O naturally aligns to device page size (e.g. 4KB)

Managing data in 4KB is expensive

- ~30GB metadata for 2TB hierarchy

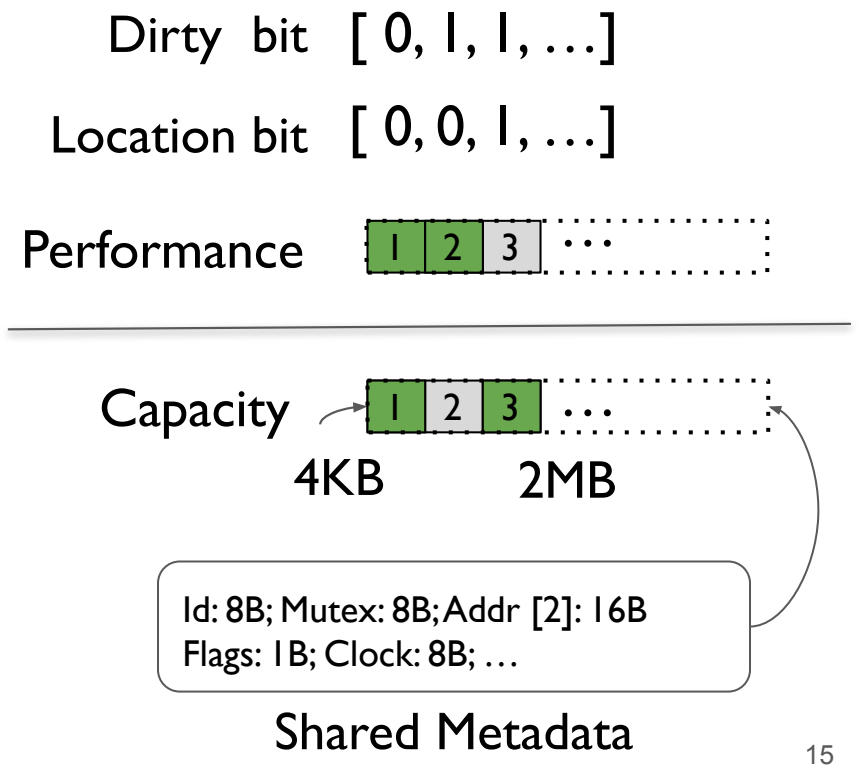


Subpage Management

MOST manages blocks in 2MB

✅ Avoid partial writes

✅ Reduce metadata overhead (~140 MB metadata for 2 TB hierarchy)



Outline

Design: Mirror-Optimized Storage Tiering (MOST)

- Data Layout
- Read and Write Balancing
- Read/Write Interaction

Implementation & Evaluation

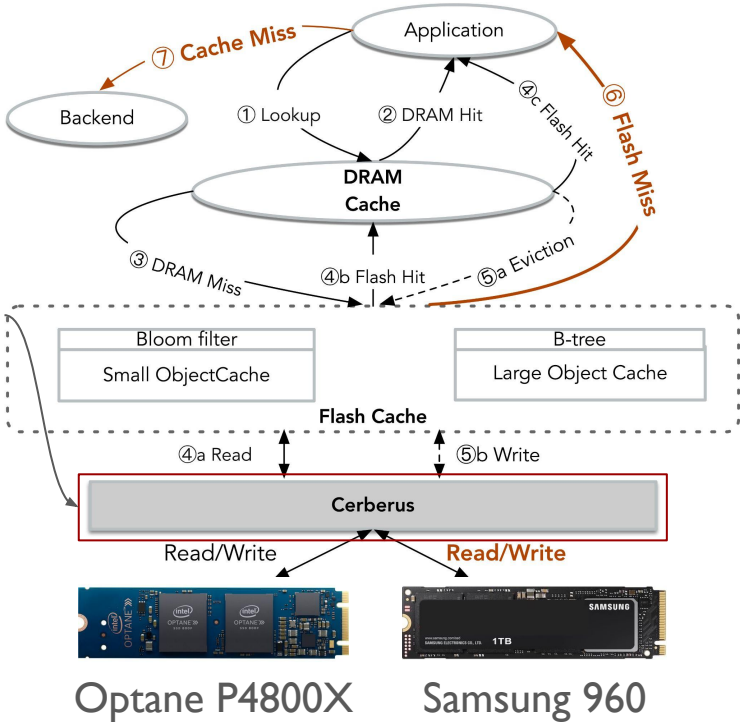
Conclusion

Implementation - Cerberus

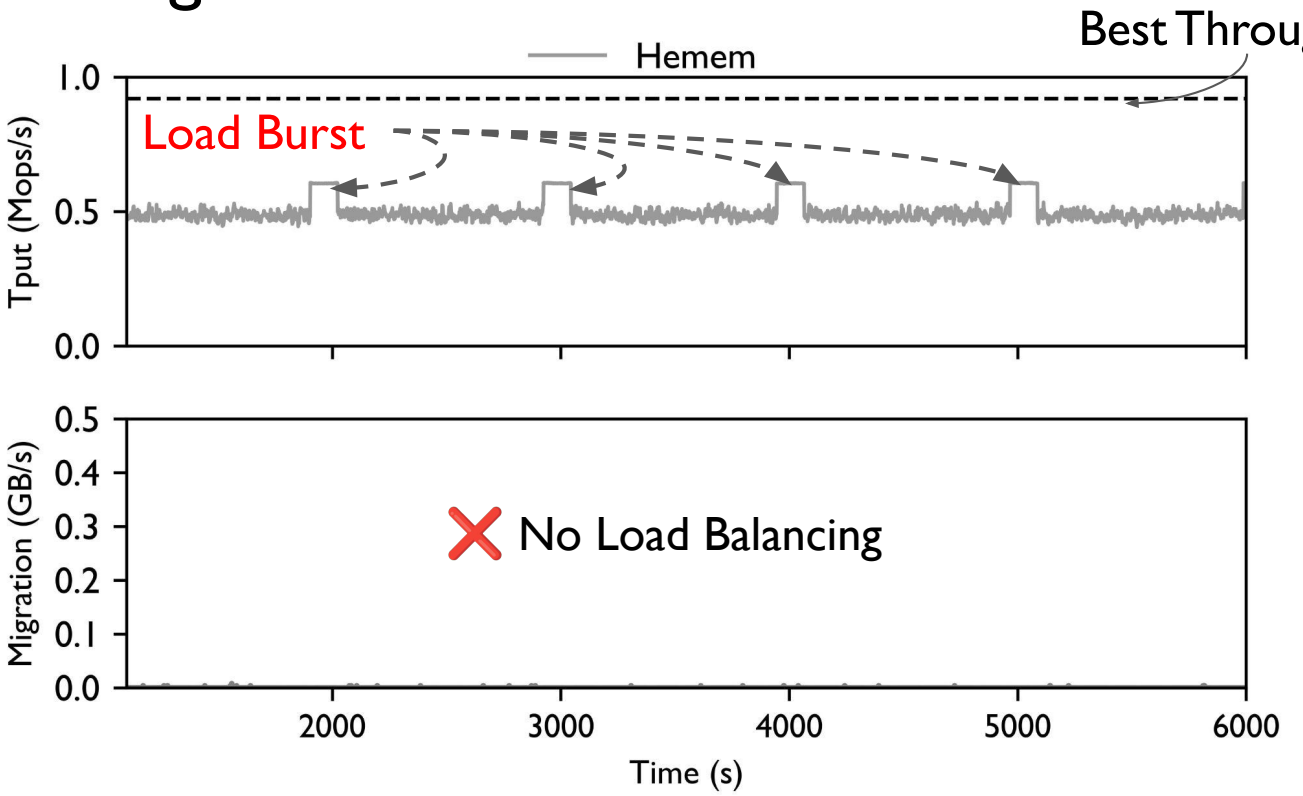
MOST is implemented as block management layer named **Cerberus** in CacheLib

Counterpart systems:

- Striping (default)
- Tiering: HeMem[4] (SOSP' 21), BATMAN (MEMSYS '17) and Colloid (SOSP' 24)
- Caching: Orthus (FAST' 21)[3]

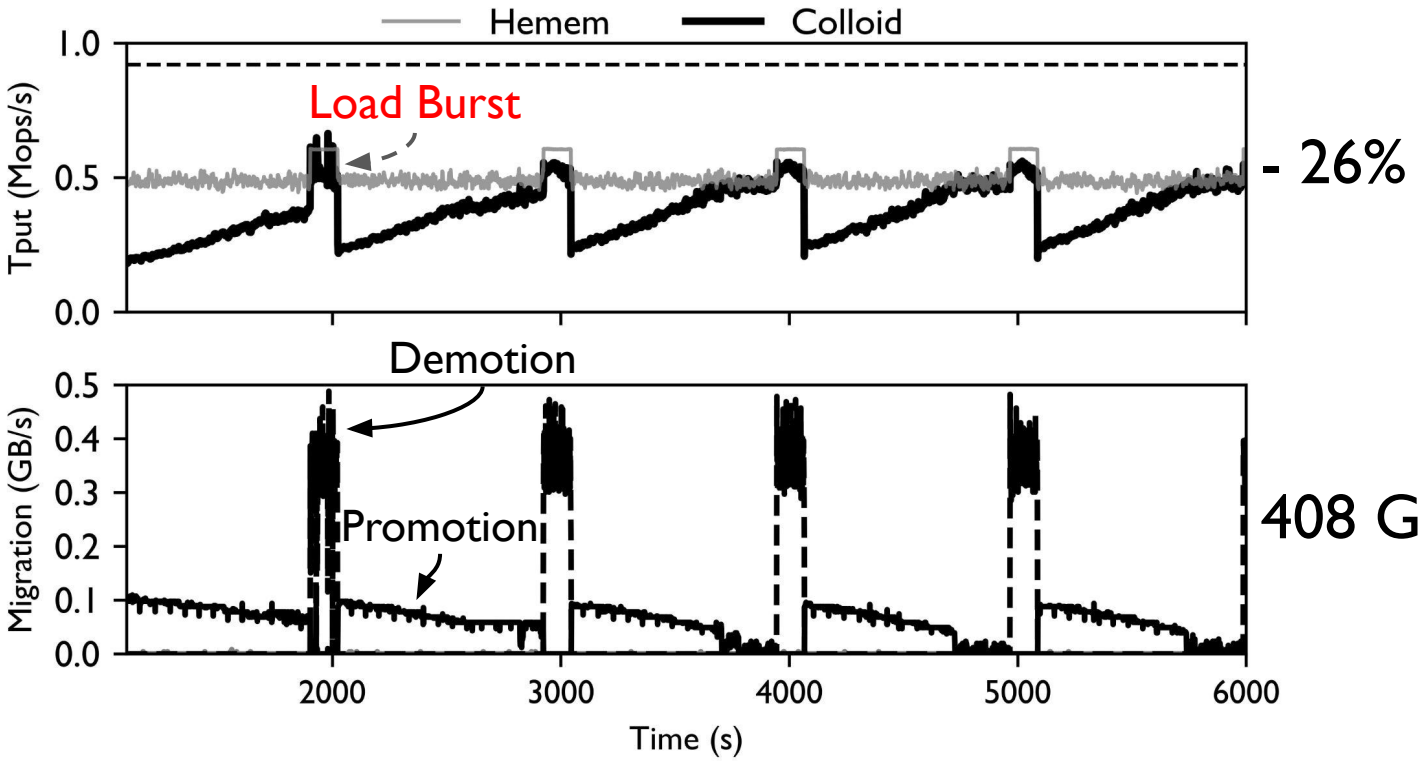


Classic Tiering Doesn't Balance



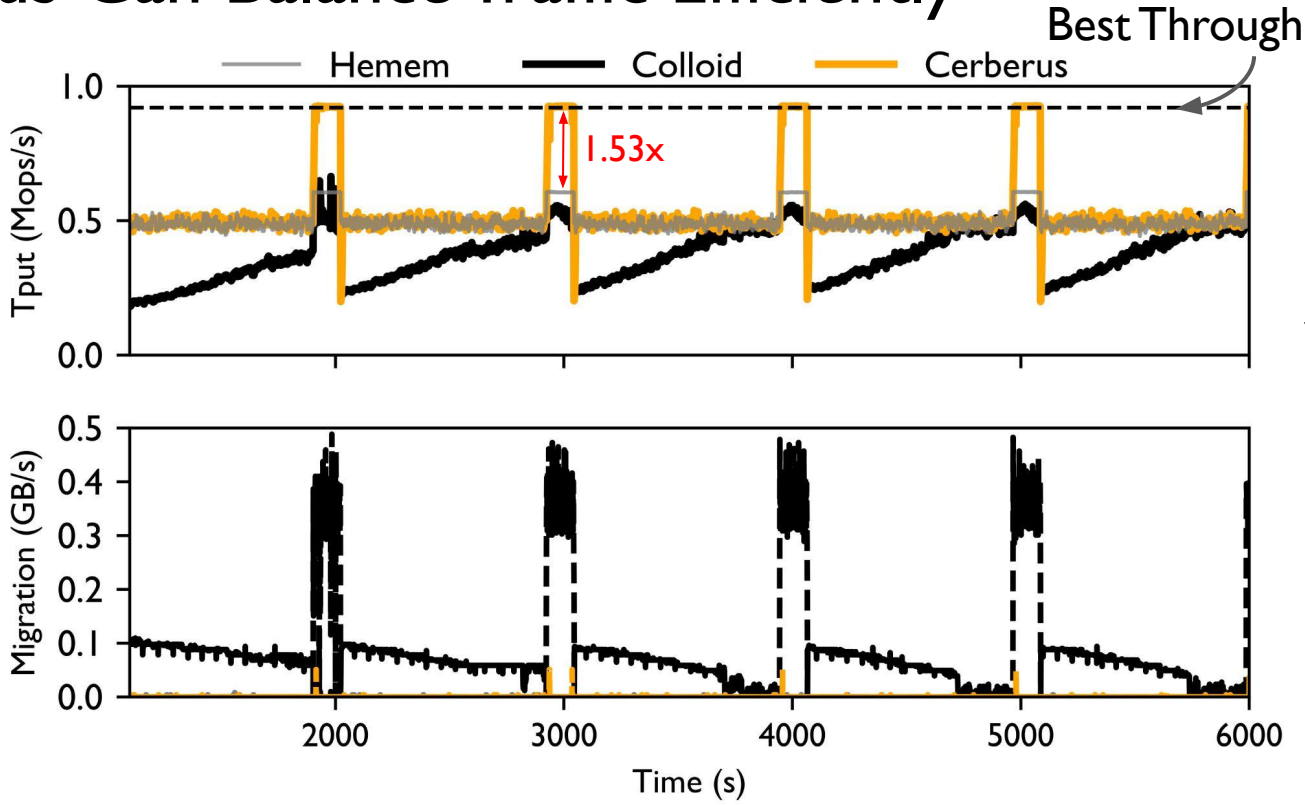
4KB random read, 1.2 TB, 2 mins burst every 15 mins

What About Migration-based Approach?



4KB random read, 1.2 TB, 2 mins burst every 15 mins

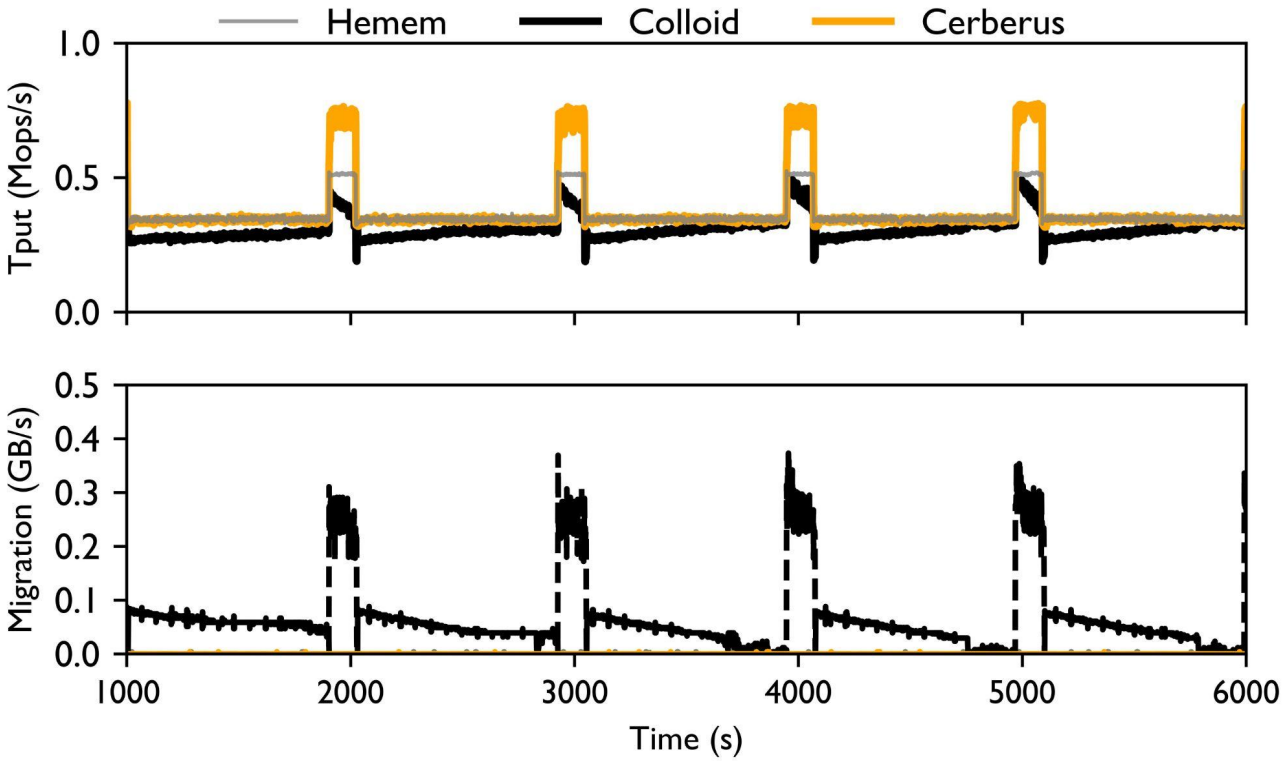
Cerberus Can Balance Traffic Efficiently



+45%
vs. Colloid

4KB random read, 1.2 TB, 2 mins burst every 15 mins

What About Write?



+23%
vs. Colloid

4KB random write, 1.2 TB, 2 mins burst every 15 mins

Other Experiments in the Paper

Cerberus improves workloads across different intensity levels

Cerberus improves CacheBench (lookaside cache, OSDI '20) workloads:

- Workloads with varying value size and write ratios
- Four production workloads (e.g. in-memory caching, block caching)

Cerberus improves YCSB workloads:

- A, B, C, D, and F (E omitted due to lack of range-query support)

…

Outline

Design: Mirror-Optimized Storage Tiering (MOST)

- Data Layout
- Read and Write Balancing
- Read/Write Interaction

Implementation & Evaluation

Conclusion

Conclusion

Migration-based load balancing in tiered storage is inefficient

We introduce Mirror-Optimized Storage Tiering (MOST)

- Mirror a small amount of hot data to achieve efficient load balancing

Hybrid storage system may offer further benefits

Thank You!

Q & A

Contact: kaiweitu@cs.wisc.edu

References

[1] M. Vuppalapati and R. Agarwal, "Tiered Memory Management: Access Latency is the Key!," in *Proc. ACM SIGOPS 30th Symposium on Operating Systems Principles (SOSP '24)*, 2024, pp. 79–94.

[2] C.-C. Chou, A. Jaleel, and M. K. Qureshi, "BATMAN: Techniques for Maximizing System Bandwidth of Memory Systems with Stacked-DRAM," in *Proc. International Symposium on Memory Systems (MEMSYS 2017)*, 2017, pp. 268–280.

[3] K. Wu, Z. Guo, G. Hu, K. Tu, R. Alagappan, R. Sen, K. Park, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "The Storage Hierarchy is Not a Hierarchy: Optimizing Caching on Modern Storage Devices with Orthus," in *Proc. 19th USENIX Conf. on File and Storage Technologies (FAST '21)*, 2021, pp. 307–323.

[4] A. Raybuck, T. Stamler, W. Zhang, M. Erez, and S. Peter, "HeMem: Scalable Tiered Memory Management for Big Data Applications and Real NVM," in *Proc. ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP '21)*, 2021, pp. 392–407.