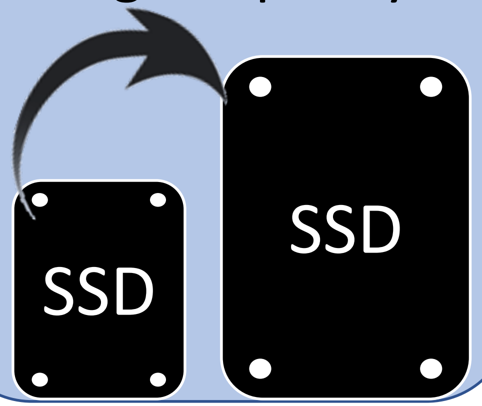
Getting More Performance with Polymorphism from Emerging Memory Technologies

Iyswarya Narayanan, Aishwarya Ganesan, Anirudh Badam, Sriram Govindan, Bikash Sharma, Anand Sivasubramaniam

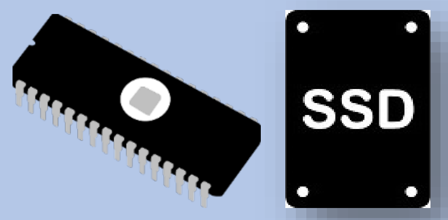
PennState Microsoft WISCONSIN UNIVERSITY OF WISCONSIN–MADISON

Resource needs of cloud storage applications span multiple aspects

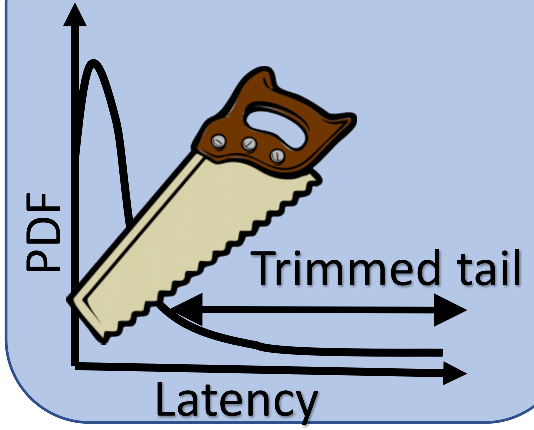
High Capacity



Volatile and
persistent accesses

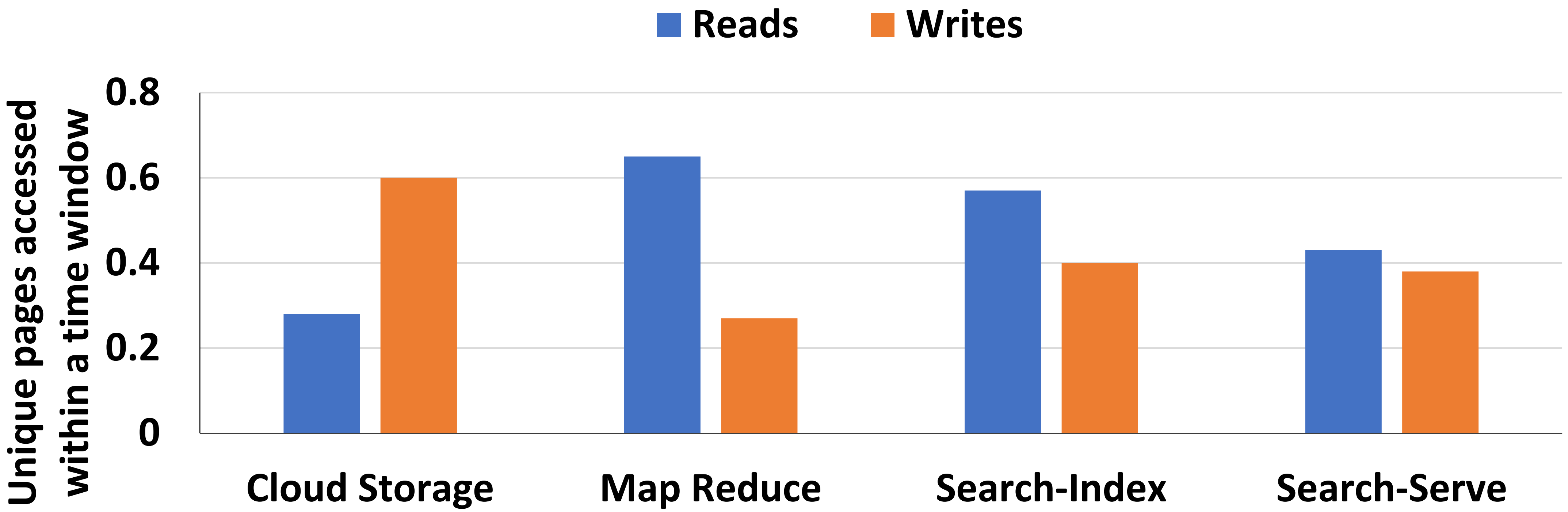


High performance



Cloud applications are diverse!

In terms of their capacity needs for both volatile reads and persistent writes



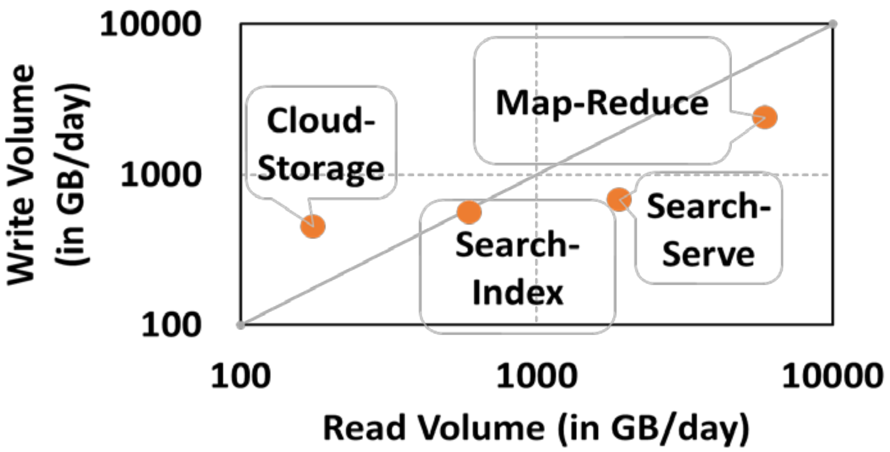
Write intensive

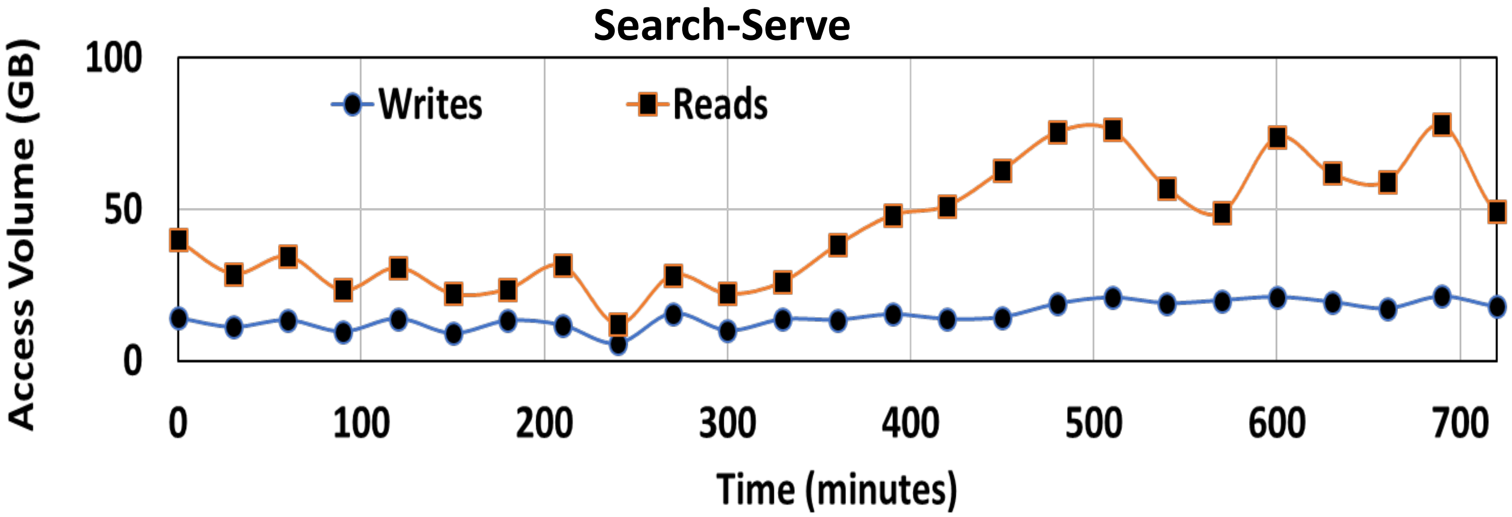
Read intensive

Read and write intensive

Cloud applications are diverse!

In terms of volume of read and write accesses



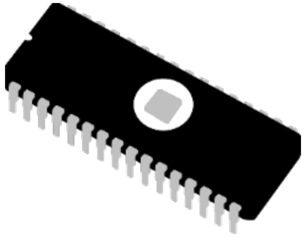


Across different applications **Temporally within same applications**

How to effectively provision memory and storage resources for diverse cloud storage applications?

DRAM and SSD are memory and storage resources

Volatile
Low latency
Low capacity

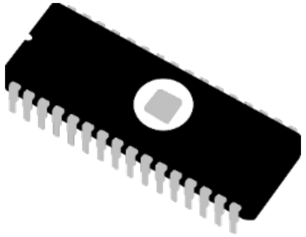


Persistent
High latency
High capacity

Latency ⬆
Cost ⬇
Capacity ⬆

They are rigid in their performance characteristics

Volatile
Low latency
Low capacity



In Function: Memory or Storage

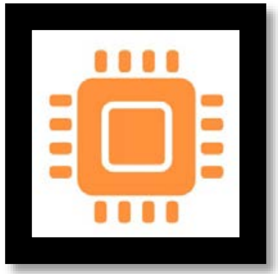
In Latency: fast DRAM vs. slow SSD

In Capacity: Based on server SKU

Persistent
High latency
High capacity

Latency ↑
Cost ↓
Capacity ↑

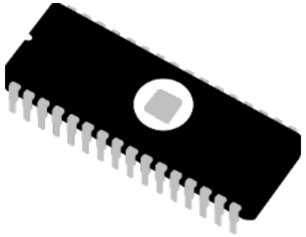
Can emerging memories help meet diverse resource needs for cloud storage apps across several dimensions?

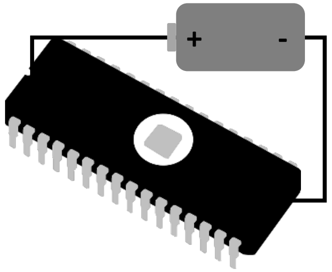


Non-volatile
Lower latencies w.r.t SSD
Larger and flexible: capacity and latency

Volatile
Low capacity
Low latency

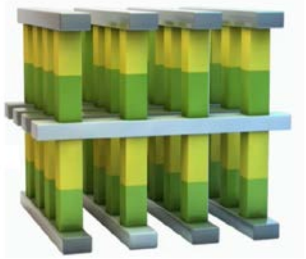
Persistent
High capacity
High latency

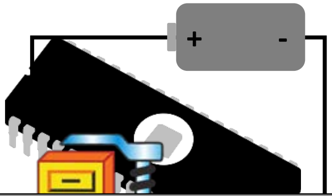




Battery Backed DRAM

3D XPoint





Compressed

Latency

Can we exploit these emerging memory technologies to overcome drawbacks of existing resources?

What are the design choices to integrate emerging memory technologies in cloud servers?

Persistent memory programming

NVM based file systems

Transparent cache (memory or storage)

Benefit volatile and persistent accesses

No changes to applications

Low cost and transparent

Intrusive code changes to applications!

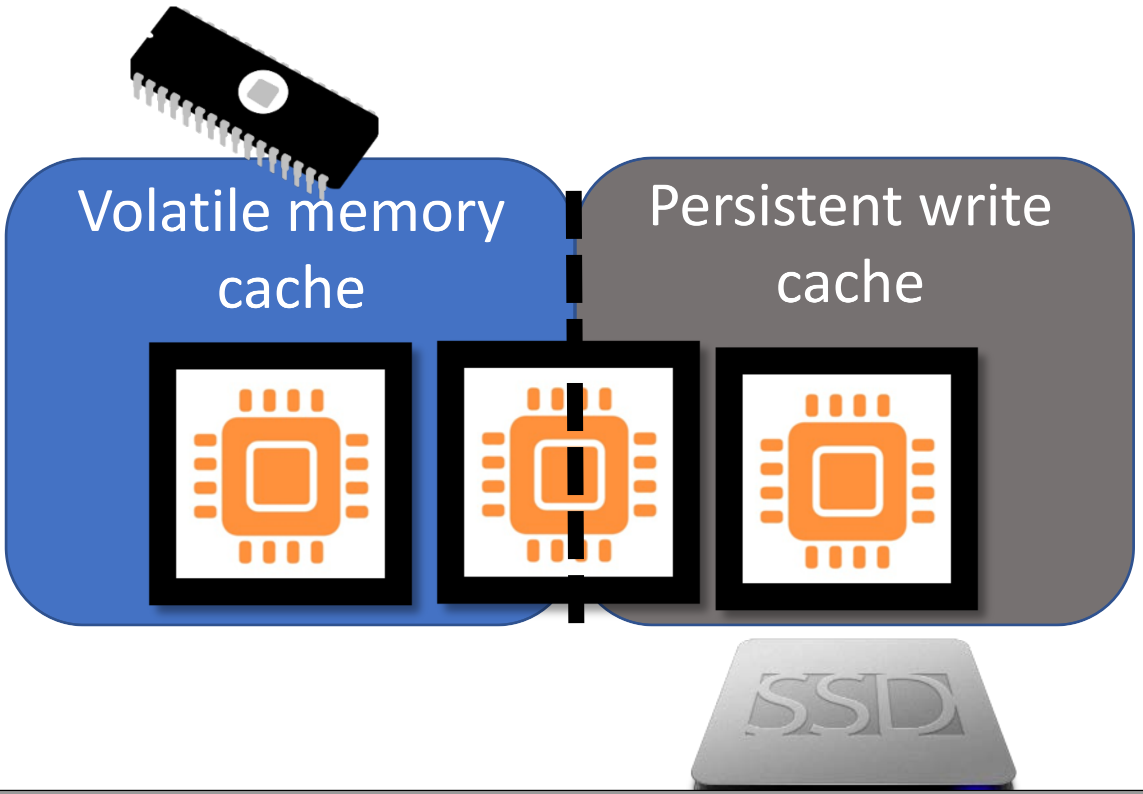
High NVM provisioning cost for entire storage needs!
Intrusive code changes to OS and FS!

Benefits reads or writes and not both!

Emerging memory technologies are polymorphic

They can function as both memory and storage!

Memory extension (direct access via loads, stores)



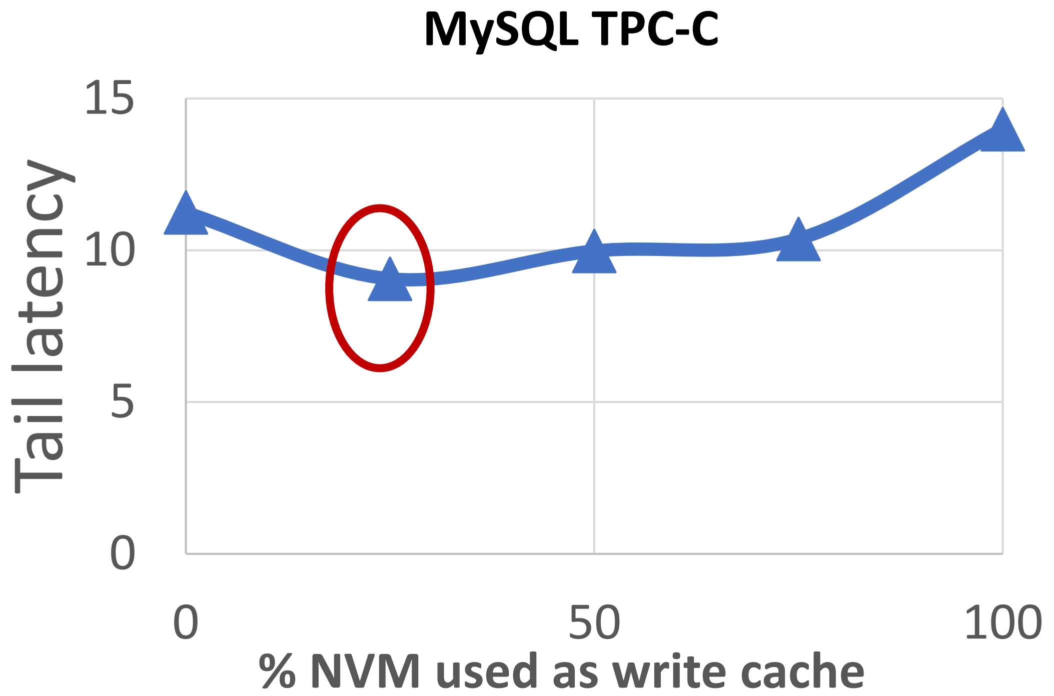
Transparent write cache above SSD (via block interface)

Can we exploit functional polymorphism knob?

Functional polymorphism can benefit applications with competing volatile and persistent flows

dm-cache to use a part of NVM as write cache

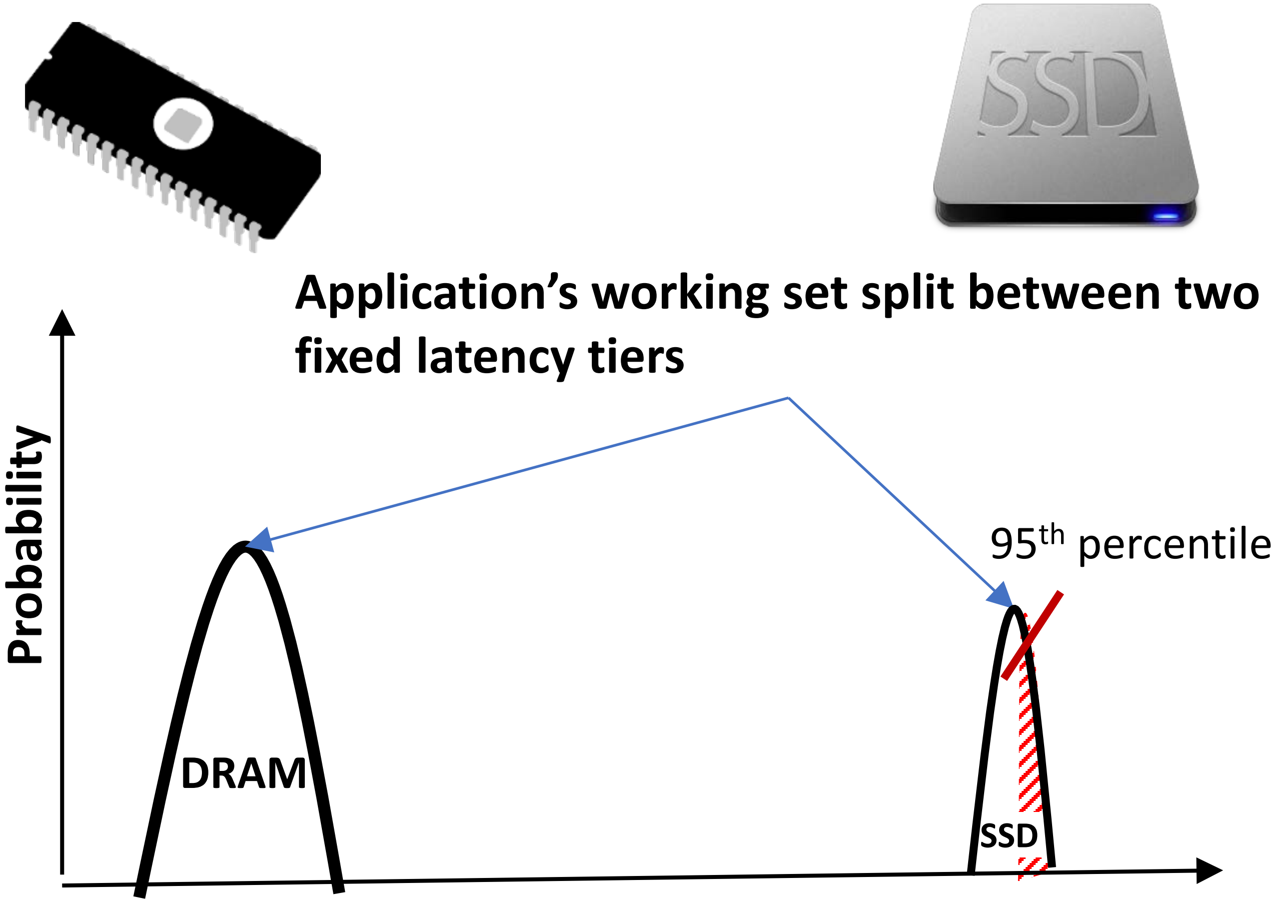
Rest – additional memory accessible via load/stores



Partitioning NVM between memory and storage reduces latency

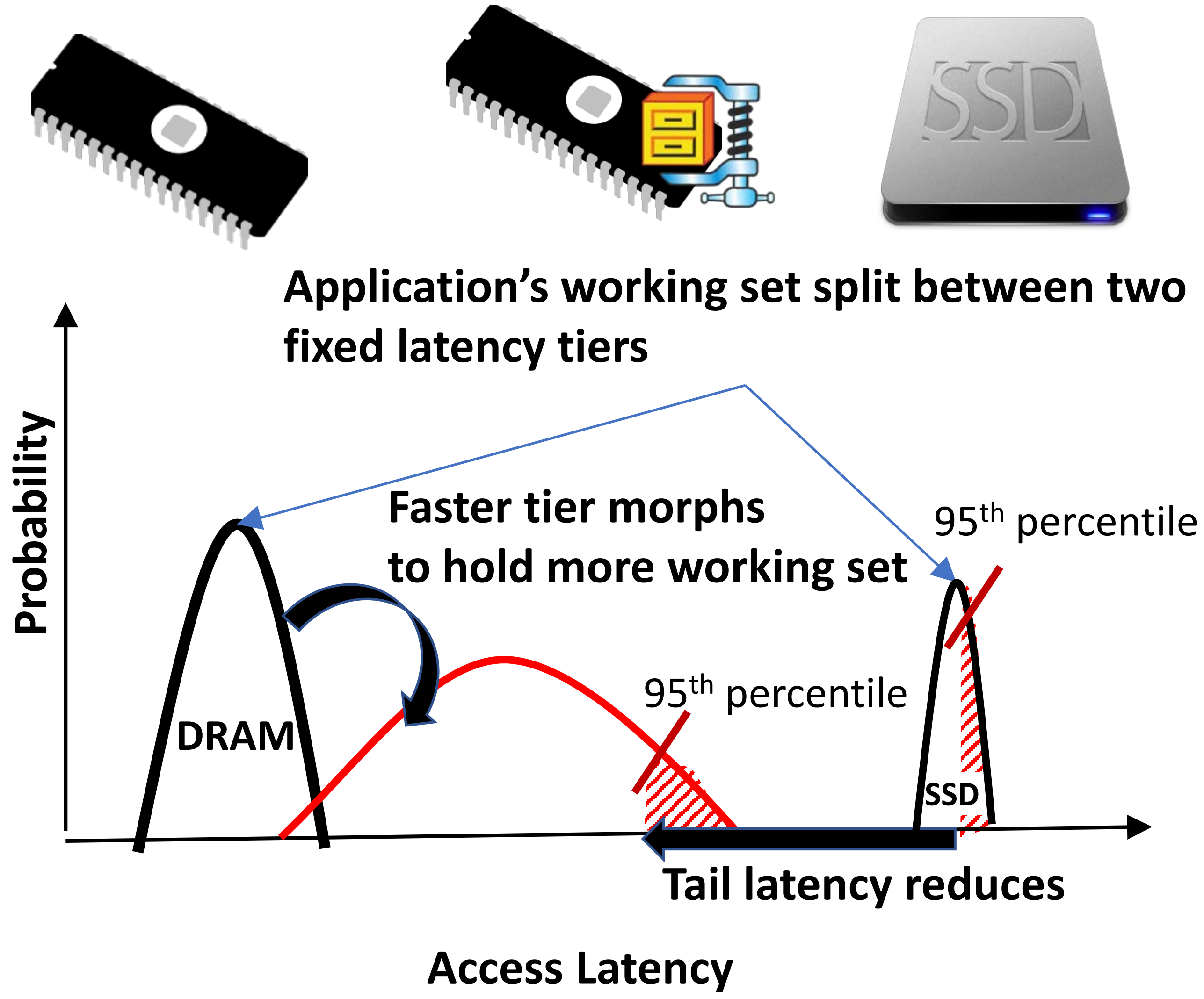
What if the working set exceeds physical memory/write-cache capacity?

Impact of insufficient physical capacity + fixed resource characteristics on application performance

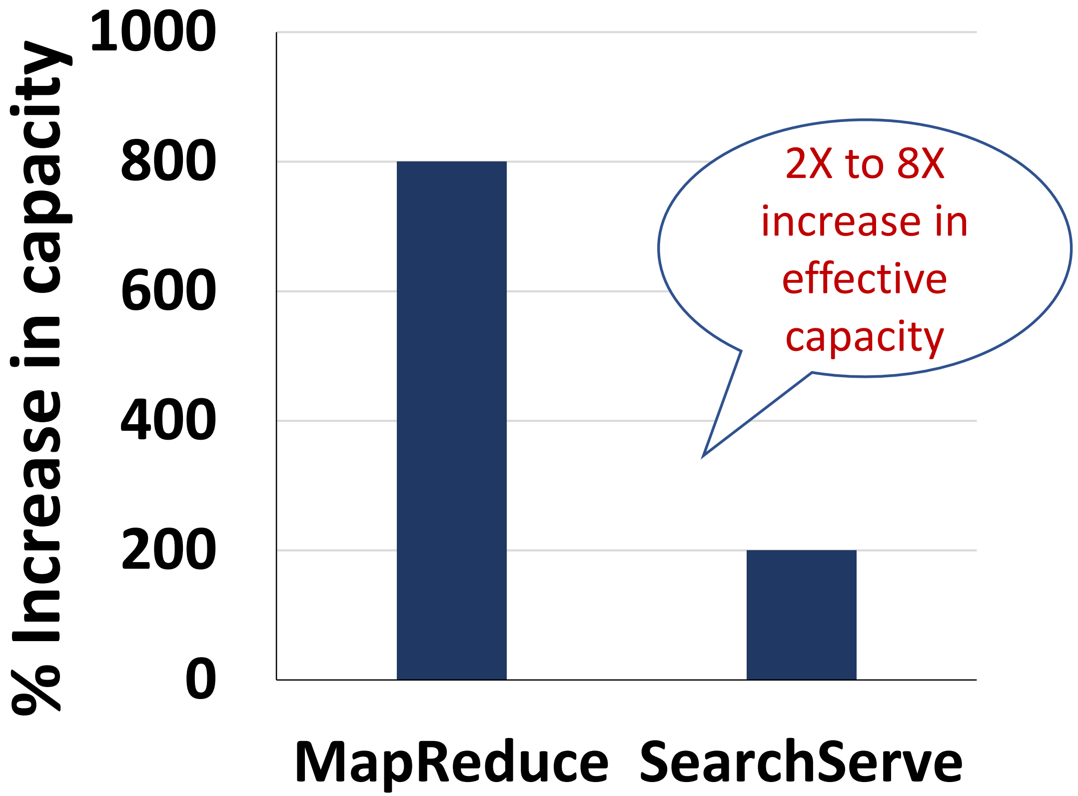


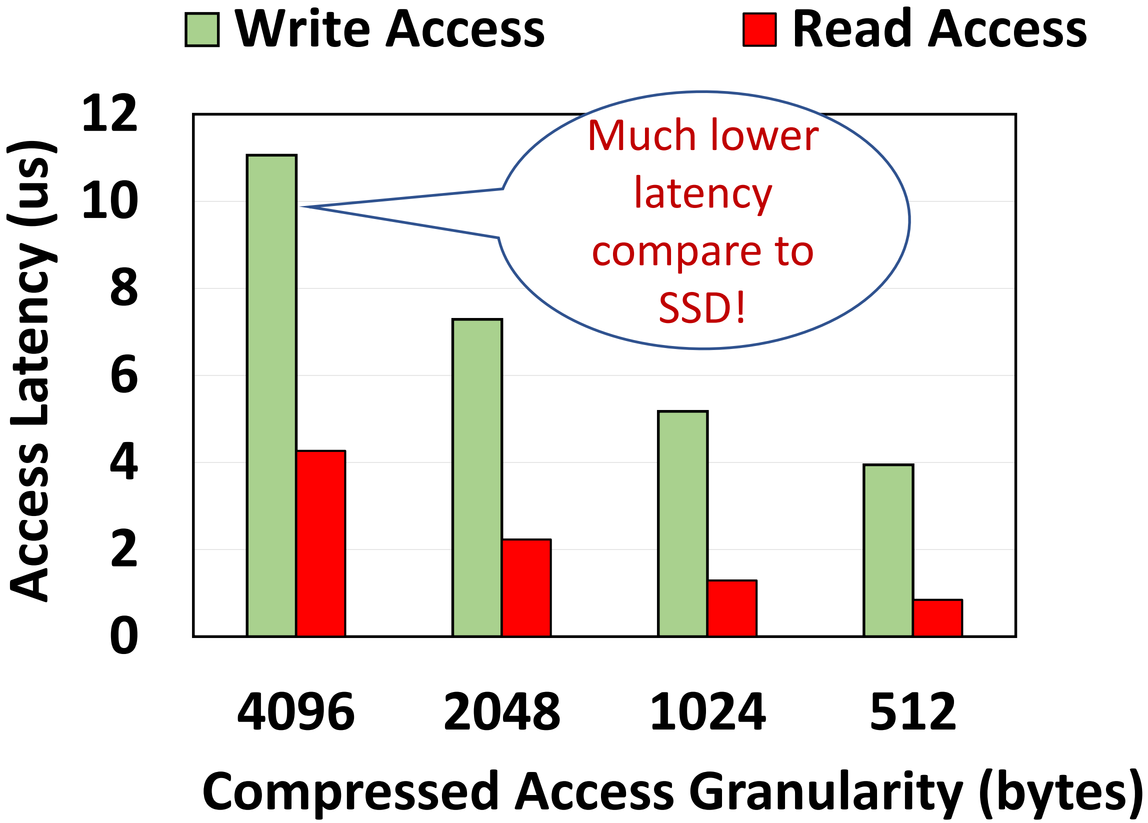
Tail latency is determined by the slowest tier

Representational polymorphism knob to tune latency and capacity



Representational polymorphism can benefit applications

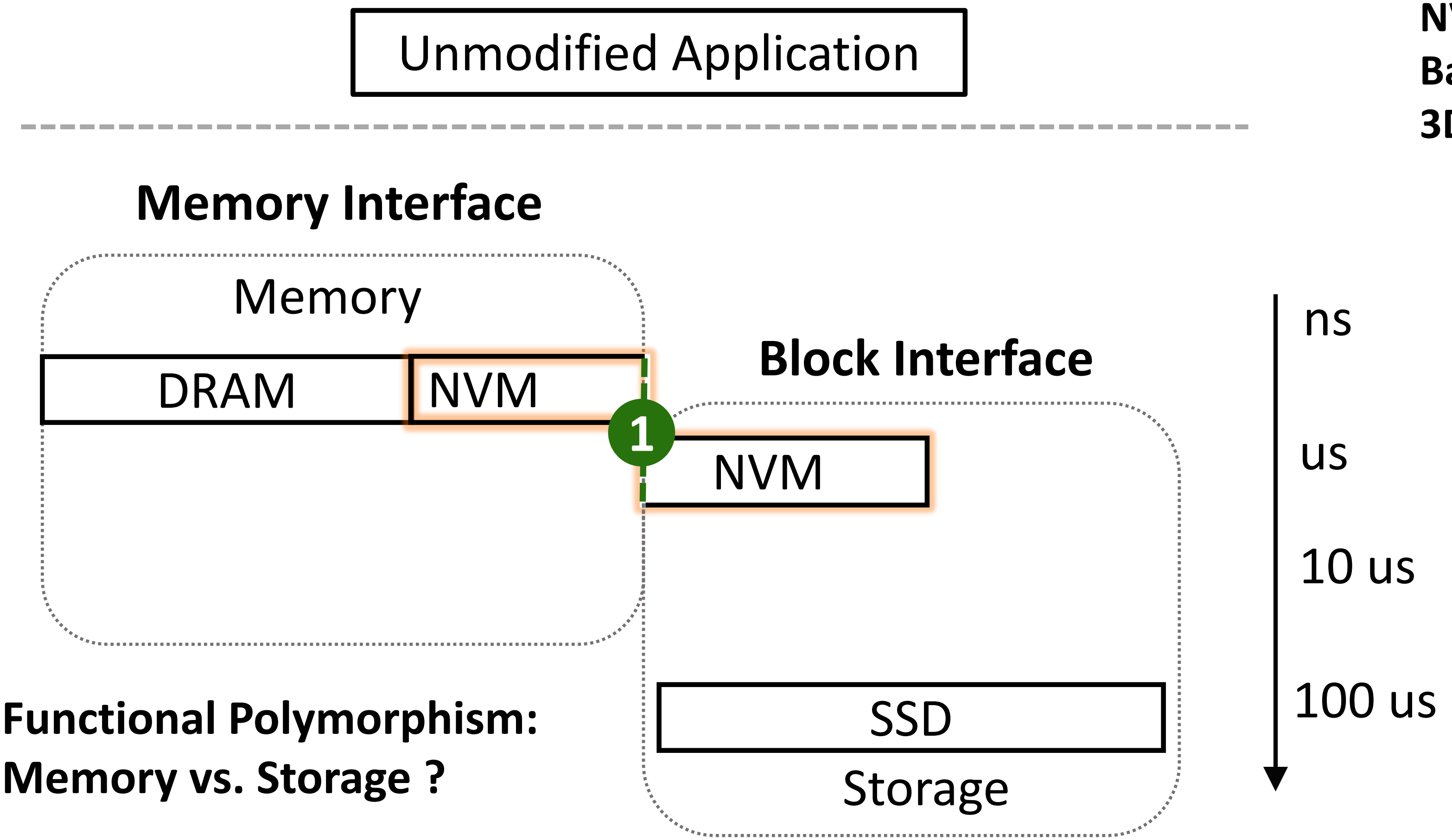




Our goal: Effectively serve diverse cloud applications using polymorphic emerging memory based cache

PolyEMT: Polymorphic Emerging Memory Technology based cache

NVM can be
Battery-backed DRAM,
3D-Xpoint, etc.

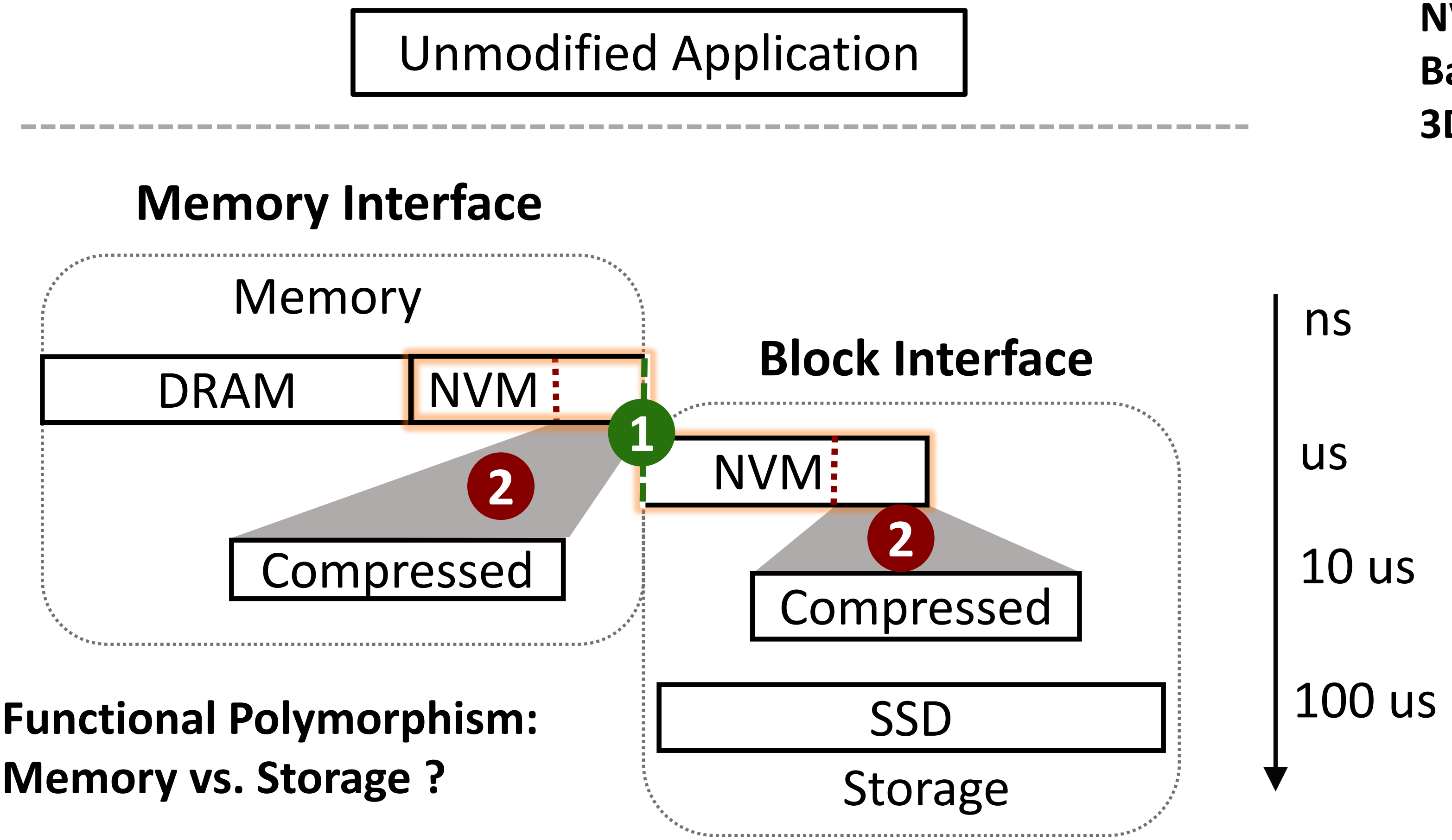


1 **Functional Polymorphism: Memory vs. Storage ?**

Cloud applications are diverse: One partition size does not fit all!

PolyEMT: Polymorphic Emerging Memory Technology based cache

NVM can be Battery-backed DRAM, 3D-Xpoint, etc.



1 **Functional Polymorphism: Memory vs. Storage ?**

Representational Polymorphism:

We need to navigate performance trade-off across capacity, latency, and persistence dimensions!

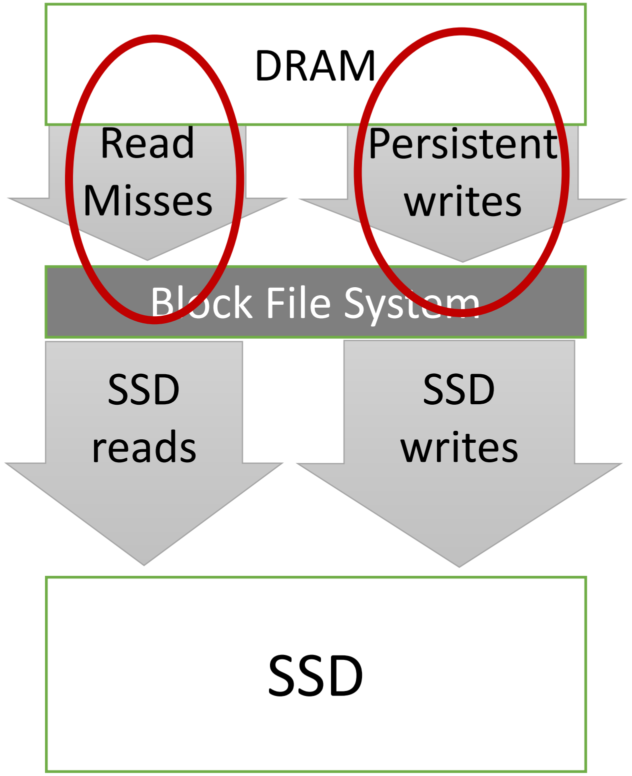
Key idea of PolyEMT cache

- Address the most significant bottleneck first using the emerging memory based cache
- Then gradually morph its characteristics to further improve performance

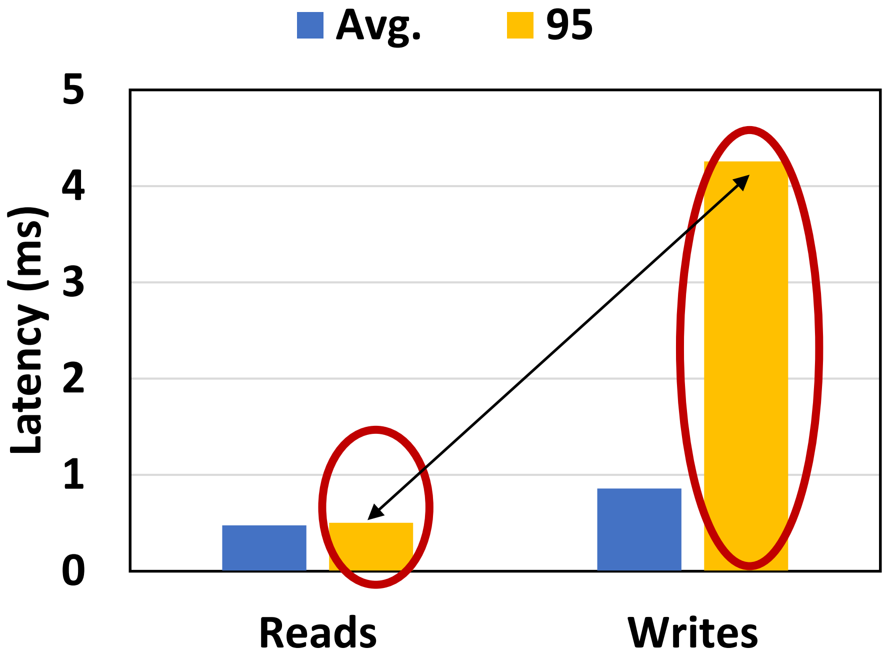
What is the most significant bottleneck for a generic application with mixes of reads and writes ?

Persistent writes (file writes, flushes, msyncs) incurs high latency in existing systems

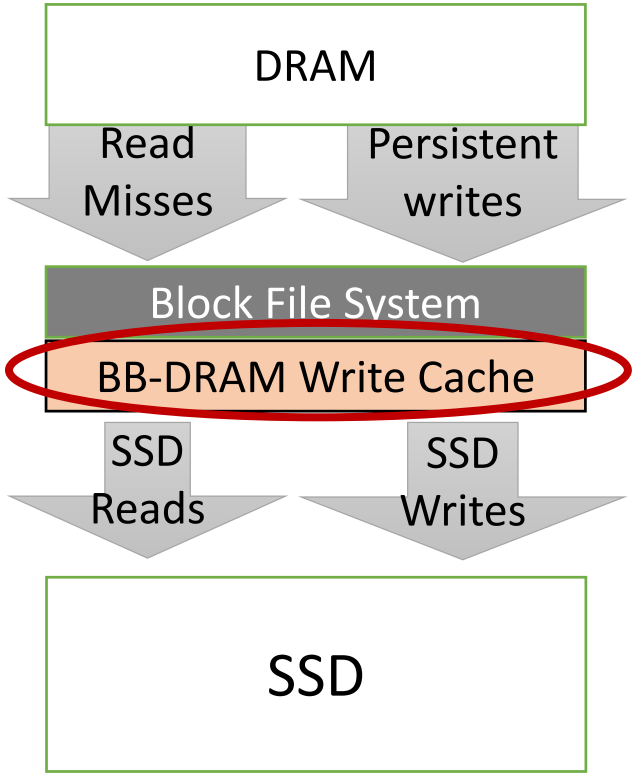
Persistent tier is much slower



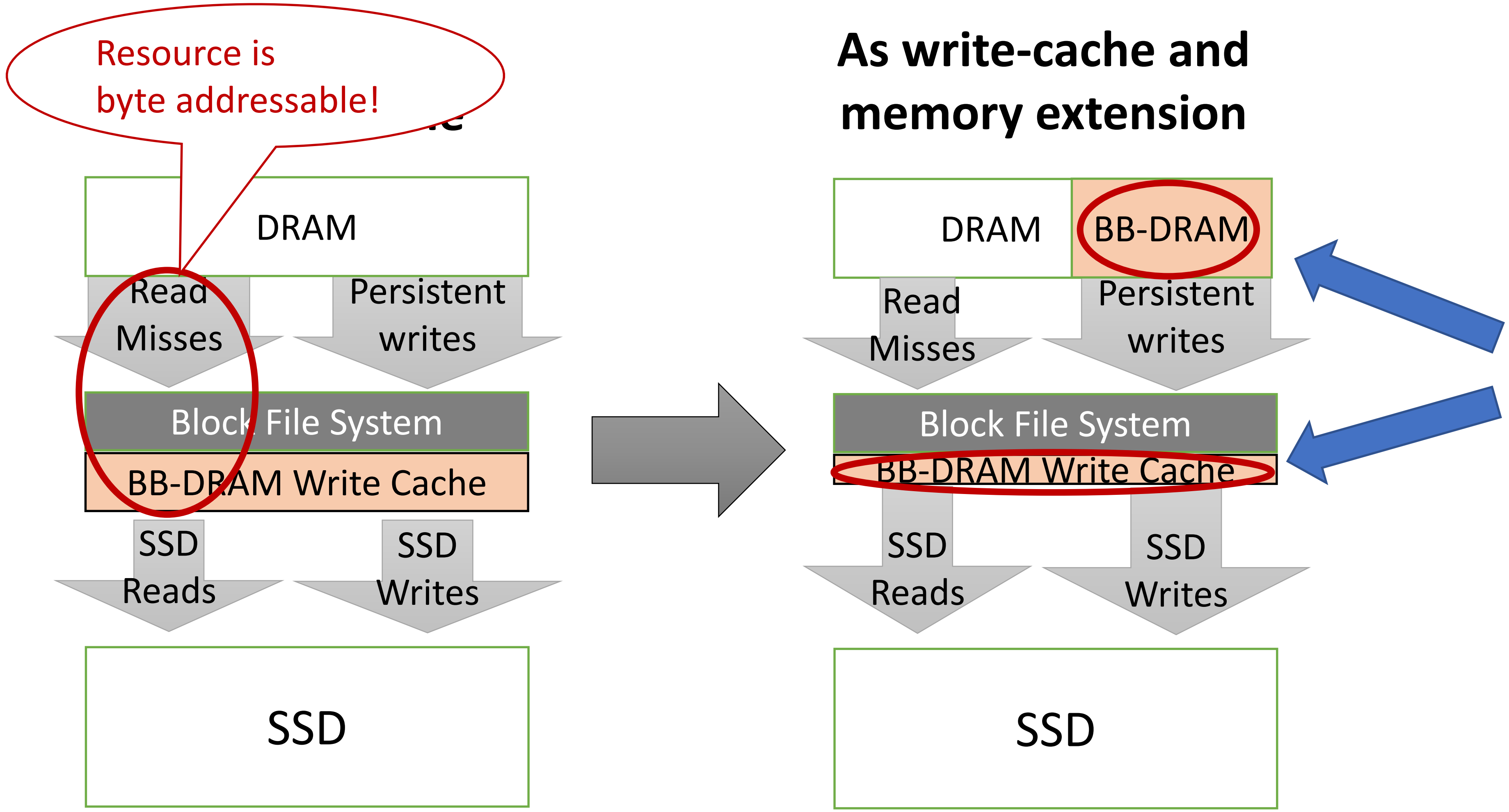
And, SSDs are asymmetric in their read/write latency



Use BB-DRAM as Write-Cache to SSD

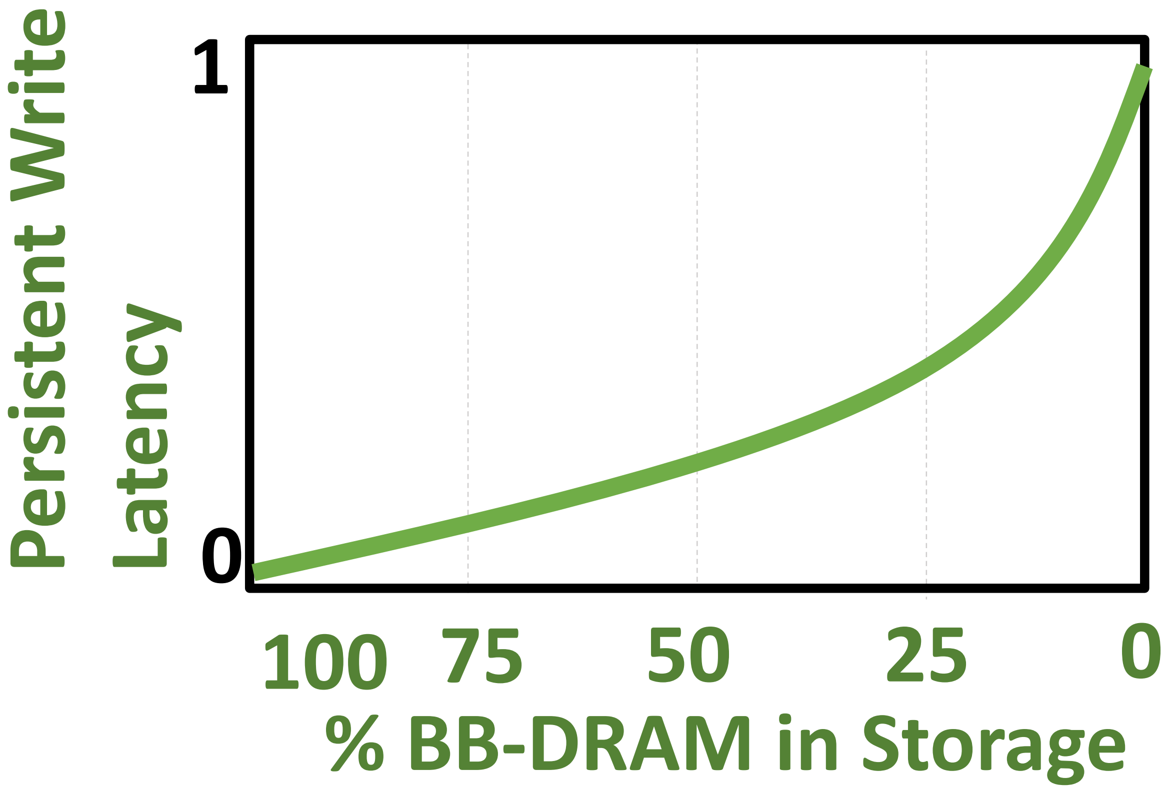


EMT entirely in Write-Cache is inefficient usage for read accesses as they are byte addressible

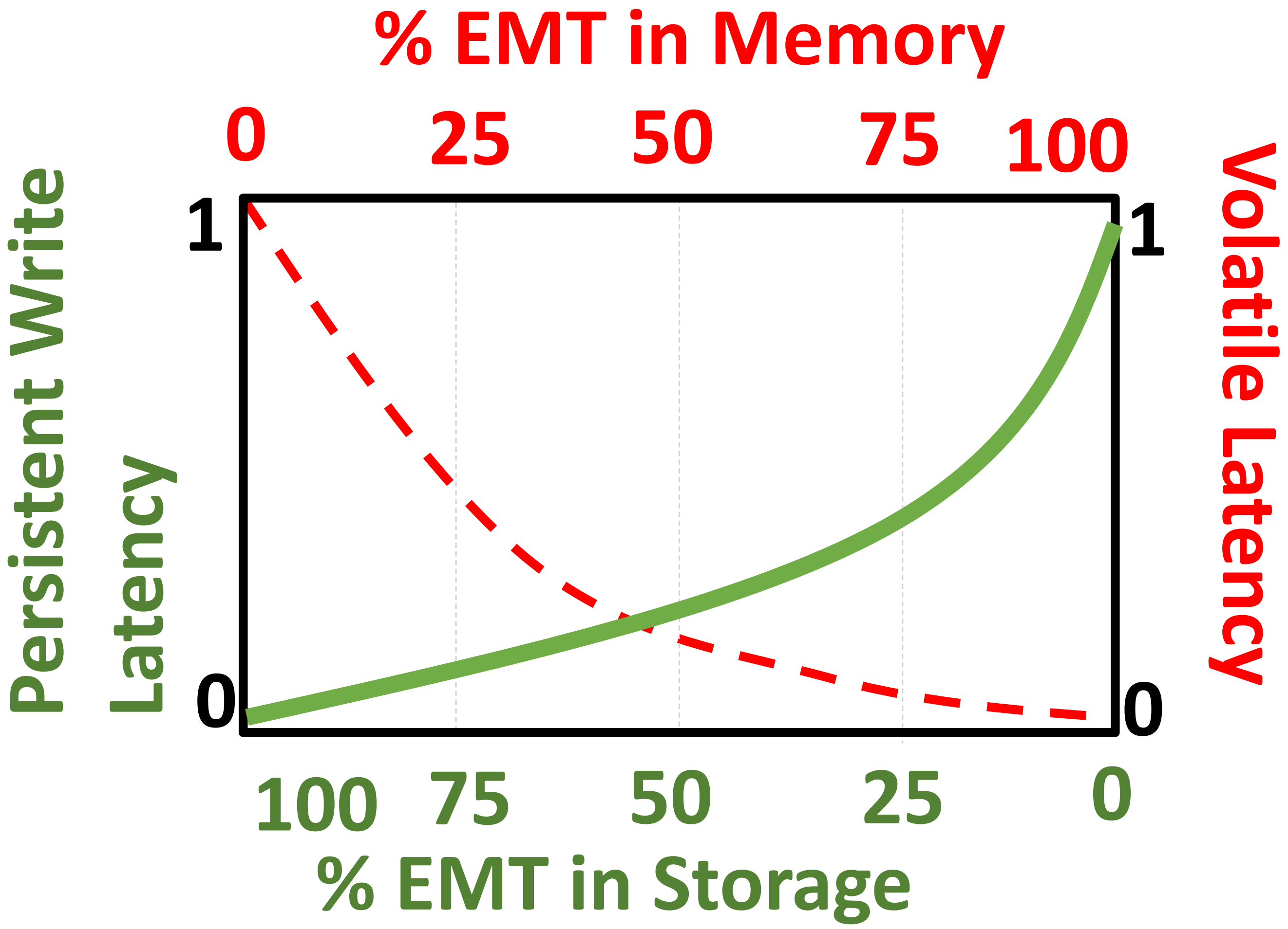


How to apportion NVM capacity between memory and Storage functions?

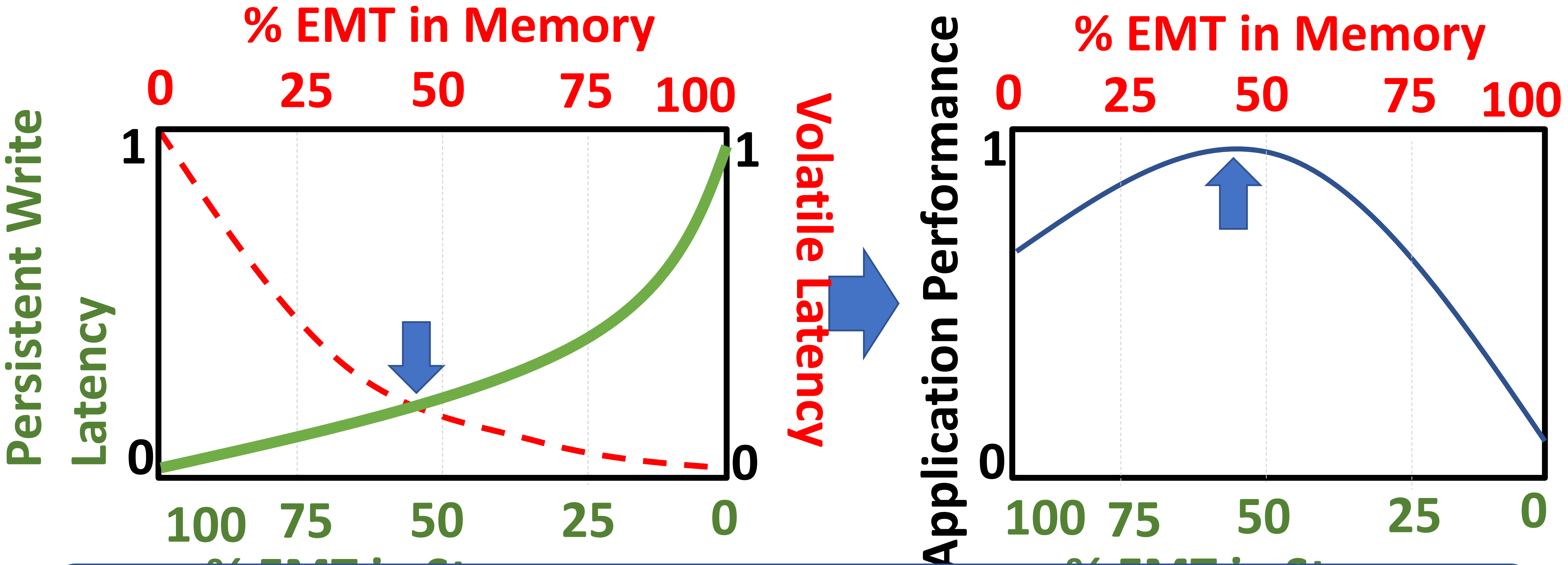
Tuning write-cache capacity in the presence of competing read and write flows



Tuning write-cache capacity in the presence of competing read and write flows

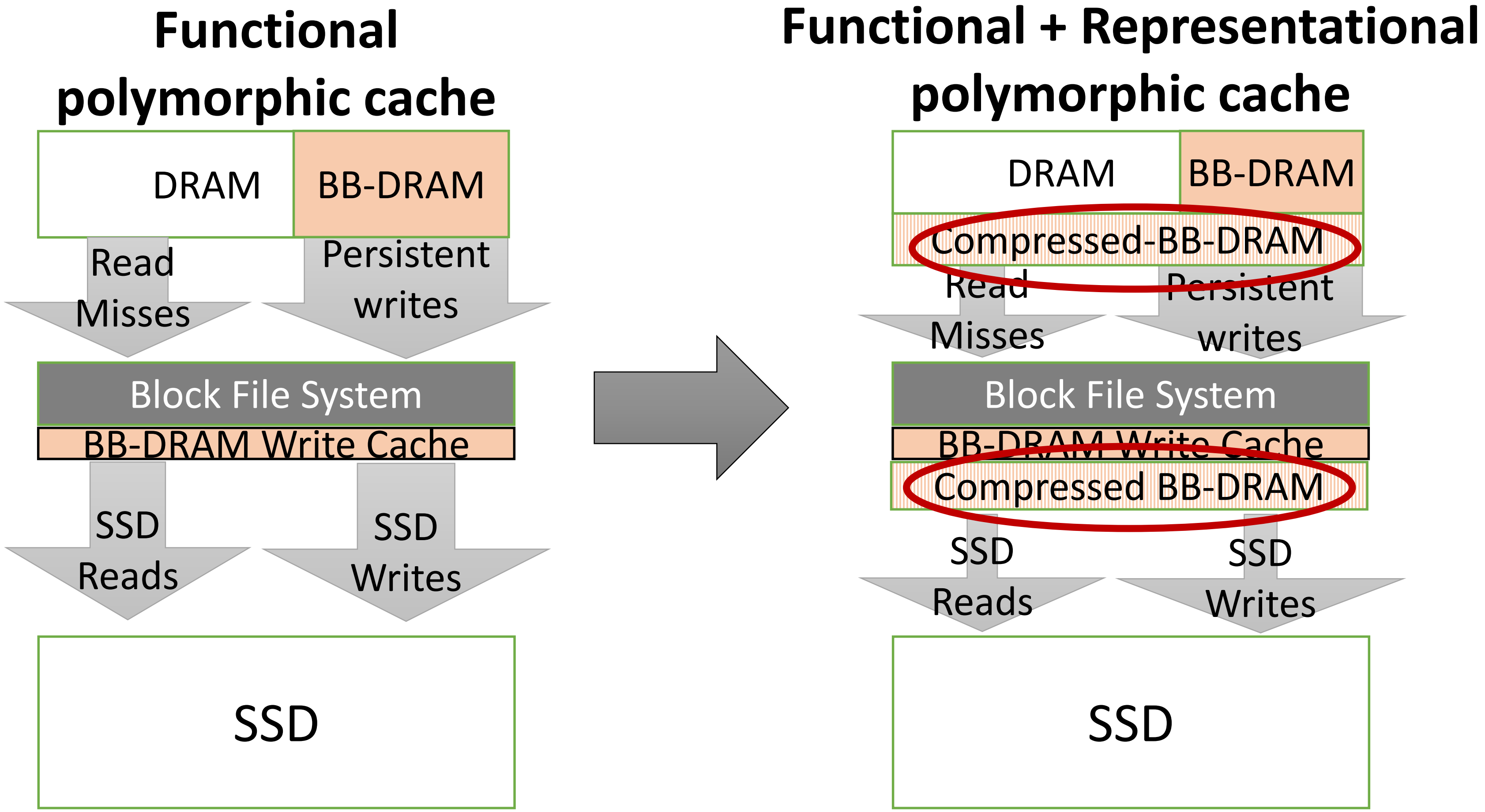


Balance the overall impact of read and write accesses



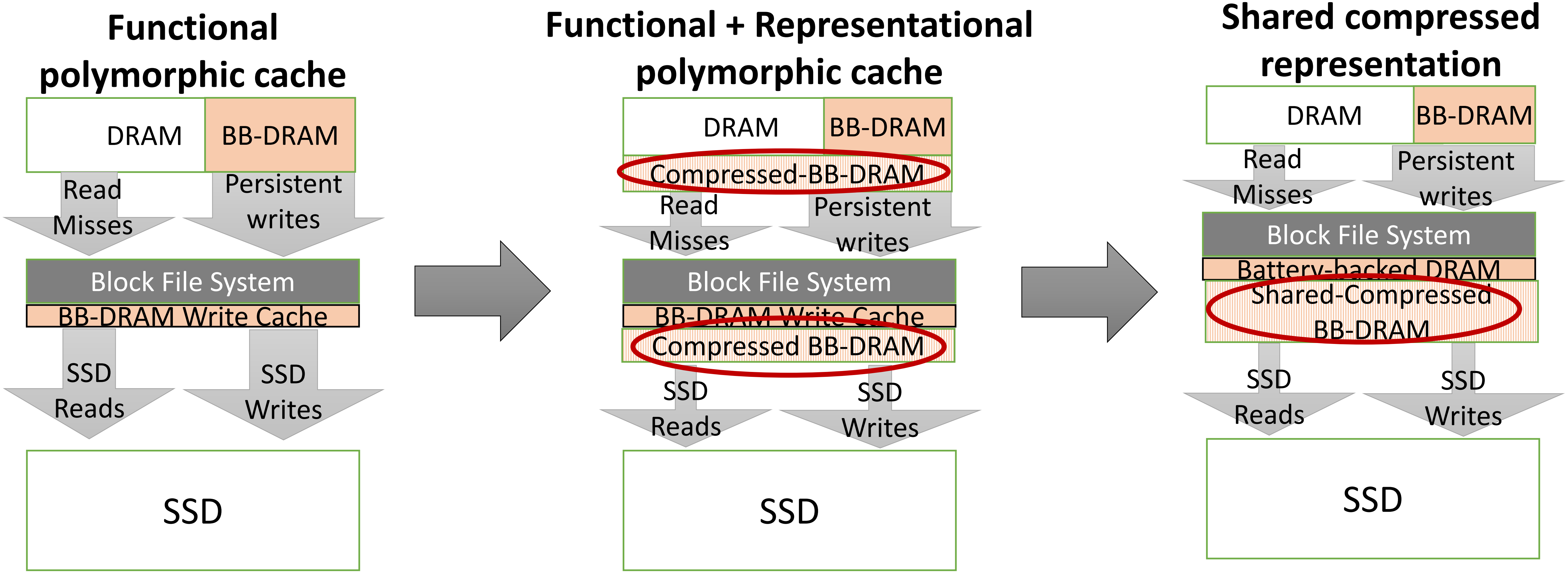
Incrementally repurpose Write-Cache blocks as memory pages to balance read/write performance.

When the physical capacity is insufficient, exploit representational polymorphism



No latency benefits by separating memory and storage functions!

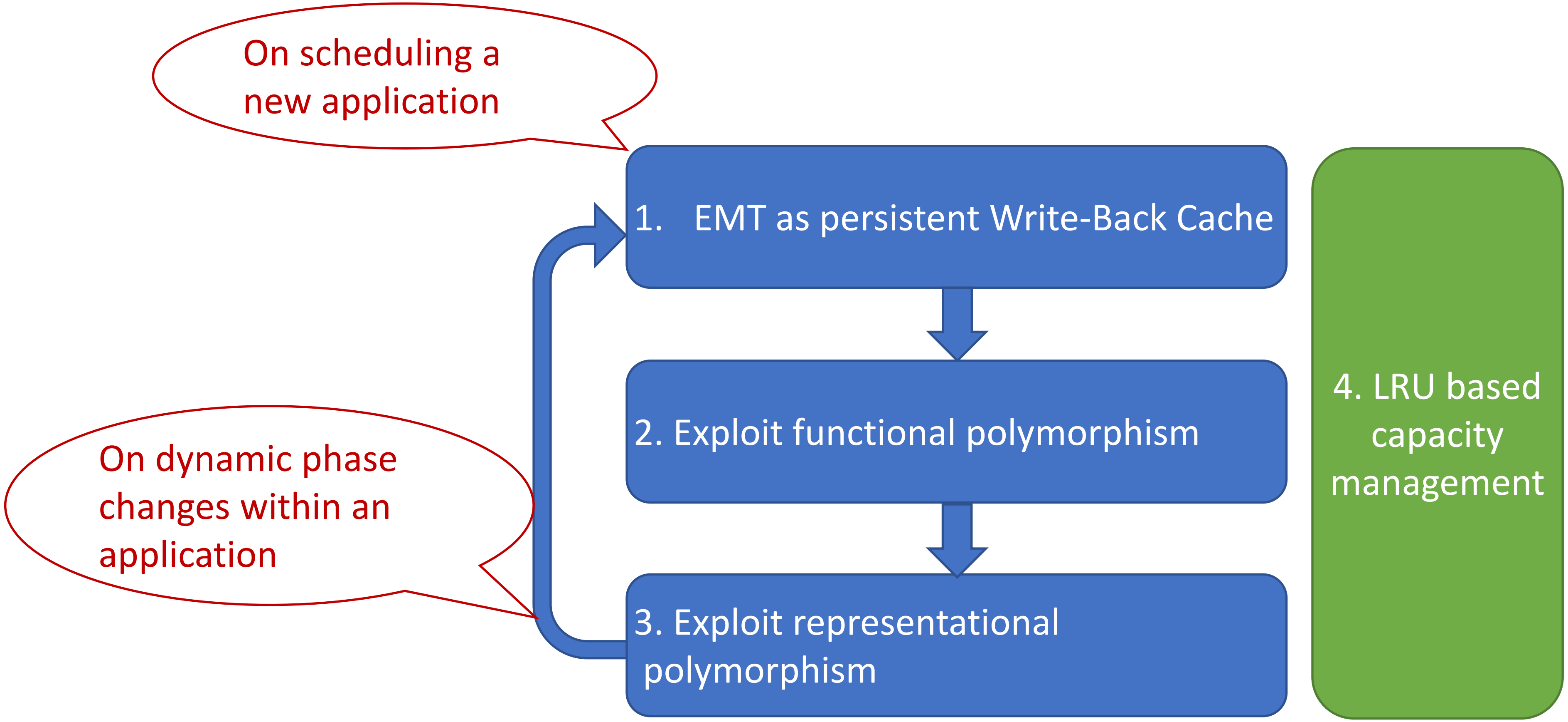
When the physical capacity is insufficient, exploit representational polymorphism



No latency benefits by separating memory and storage functions!

Shared compression layer reduces compute requirements too!

PolyEMT optimization steps at a glance



PolyEMT prototype

- PolyEMT library and runtime
  - mmap(): native load/store access
  - msync(): persist dirty data to NVM write cache
    persist data to SSD in background

- More implementation details in the paper

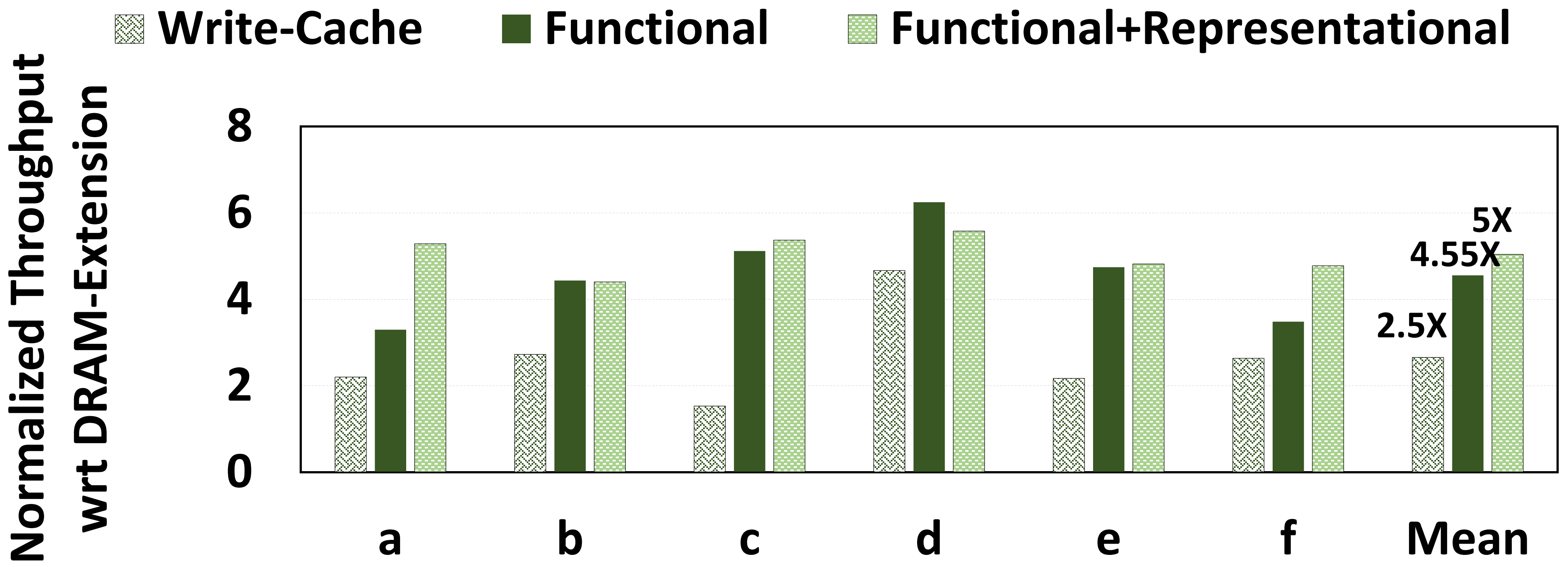
Evaluation Setup

- Azure VM
  - DRAM (26GB)
  - Battery Backed –DRAM (6GB)
  - SSD
  - CPU based compression
- Redis Key-Value store with persistence capability
- Data set size:
  - 38GB much higher than DRAM+BB-DRAM capacity
- YCSB benchmarks

Transparent integration policies under evaluation

- Dram-Extension
- Write-Cache
- Write-Cache + Functional polymorphism
- Write-Cache + Functional polymorphism + Representational polymorphism

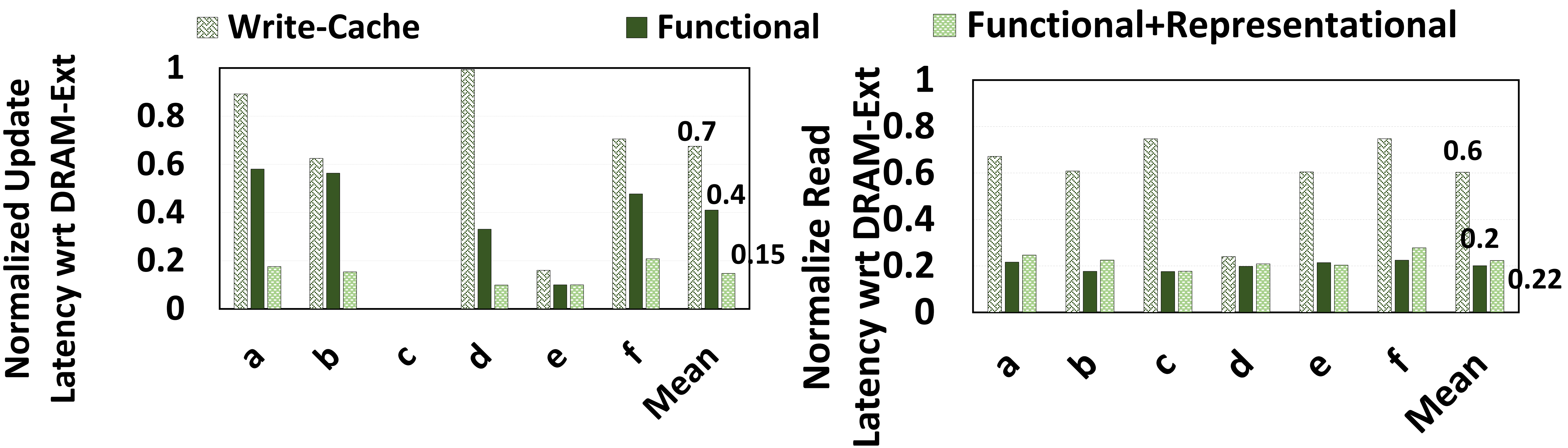
Performance benefits of PolyEMT on throughput



Addressing the most significant bottleneck improves performance by 2.5X

Exploiting polymorphisms further improves performance by 70% and 90%

Performance benefits of PolyEMT on tail latency

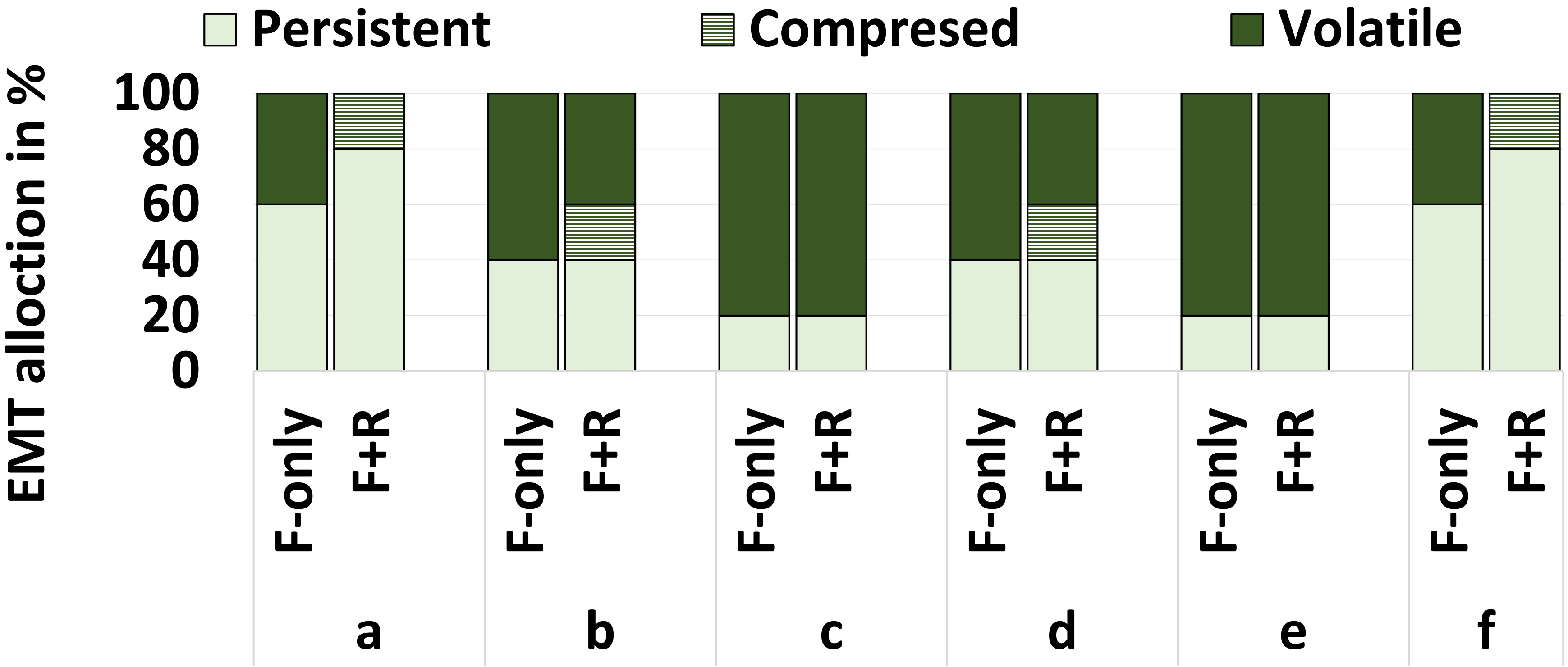


EMT based write cache reduces write and read tail latency by 30% and 40%

Functional polymorphism reduces write and read tail latency by 60% and 80%

Combining morphing reduces write and read tail latency by 85% and 78%

PolyEMT achieves performance by apportioning polymorphic resource across multiple dimensions



PolyEMT benefits diverse cloud applications via careful apportioning of polymorphic cache across three dimensions!

Diverse storage applications + Polymorphic EMT cache = High performance

To conclude,

- Explore emerging memory technologies to augment SSD performance
  - For diverse cloud applications
  - In a cost efficient and transparent way

- Our contributions:
  - Functional and representational polymorphism knobs of emerging memories
  - EMT design as a cache for SSD
  - Transparent mechanism to integrate this cache
  - Policy to morph this cache across to improve performance

- Software defined memory and storage resource provisioning to extract better performance per cost