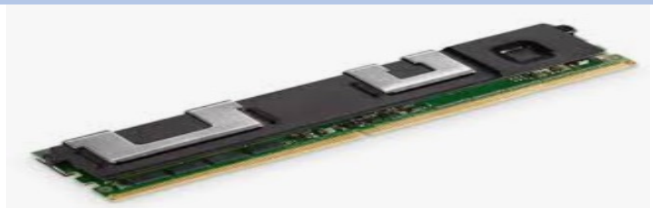
NVAlloc: Rethinking Heap Metadata Management in Persistent Memory Allocators

Zheng Dang [$] Shuibing He[$] Peiyi Hong [$] Zhenxin Li [$]
Xuechen Zhang[*] Xian-He Sun [#] Gang Chen[$]

$ 浙江大学 Zhejiang University
* Washington State University
# Illinois Tech

Persistent Memory is Emerging

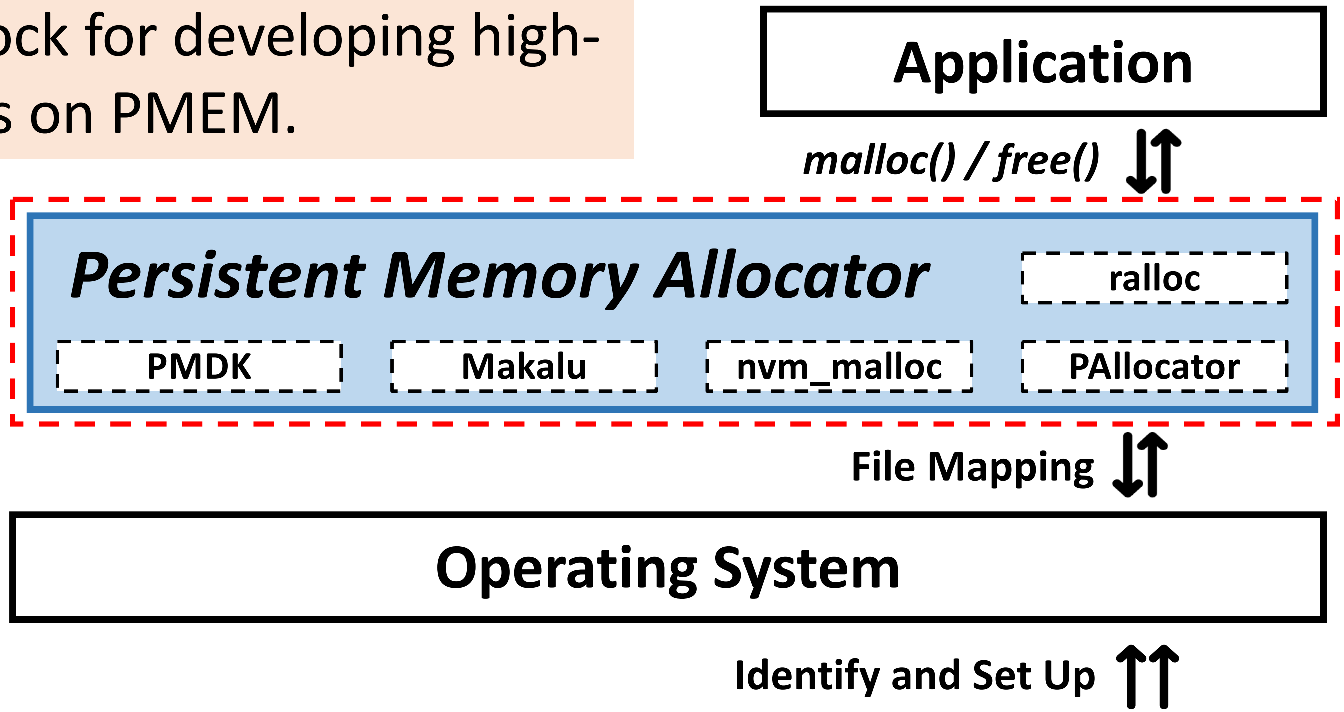


	Speed	Cost	Non-Volatility	Byte-Addressability	Power
Flash	Low	Decreasing	Yes	No	Low
DRAM	High	Increasing	No	Yes	High
PMEM	High*	Decreasing	Yes	Yes	Low

Speed Cost Non-Volatility Byte-Addressability Power

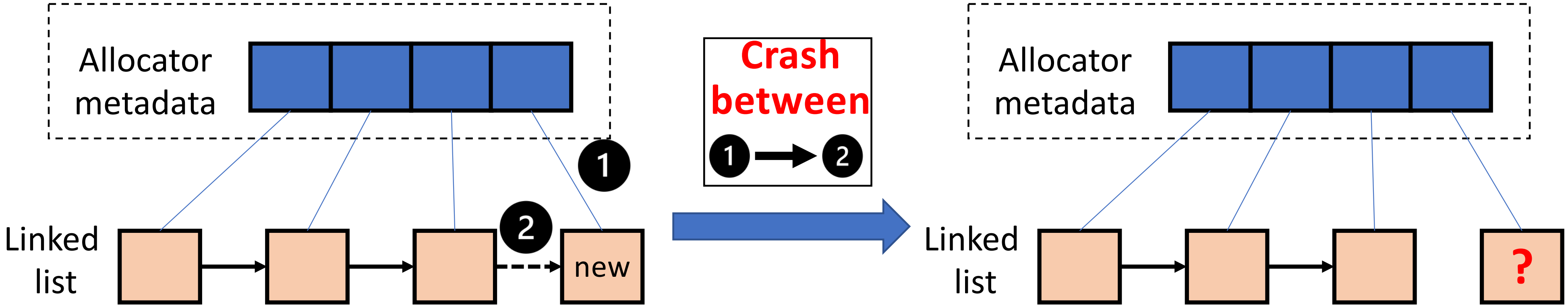
Persistent Memory Allocator

- Fundamental building block for developing high-performance applications on PMEM.
- Middleware that provides recoverable and fine-grained user-level heap memory management.



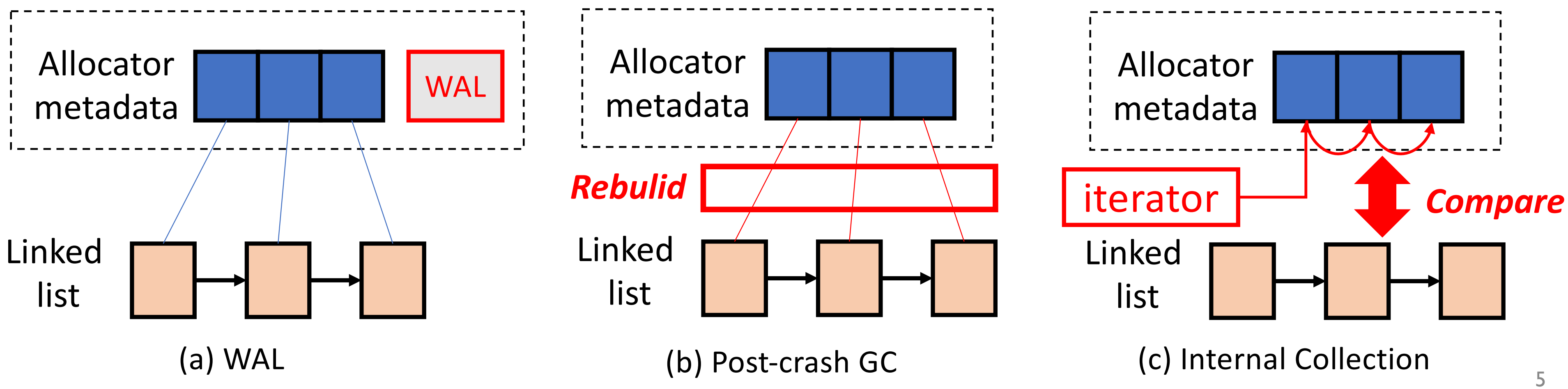
Persistent Memory Leaks

- Memory leaks are more problematic for persistent memory.
  - ➢ The leaks are persistent.
- Persistent memory faces a new class of memory leaks resulting from power failures.



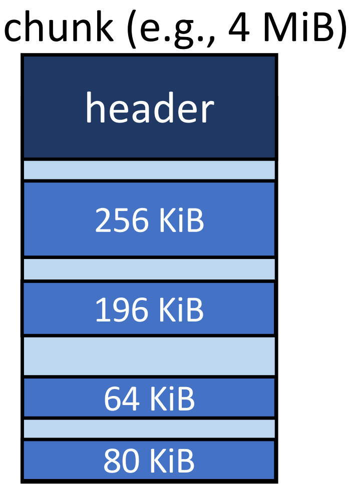
Consistency Models

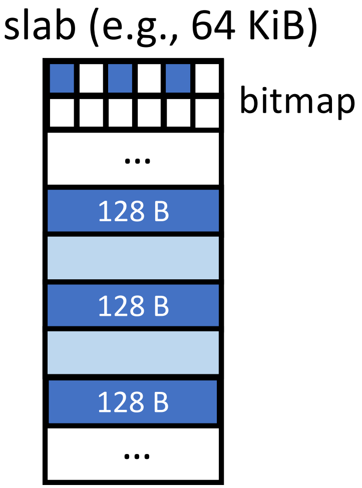
- Previous works use different consistency models to address persistent memory leaks:
  - Write-ahead log(WAL) based model (nvm_malloc, PAllocator).
  - Post-crash garbage collection based model (Makalu, ralloc).
  - Internal collection based model (PMDK).

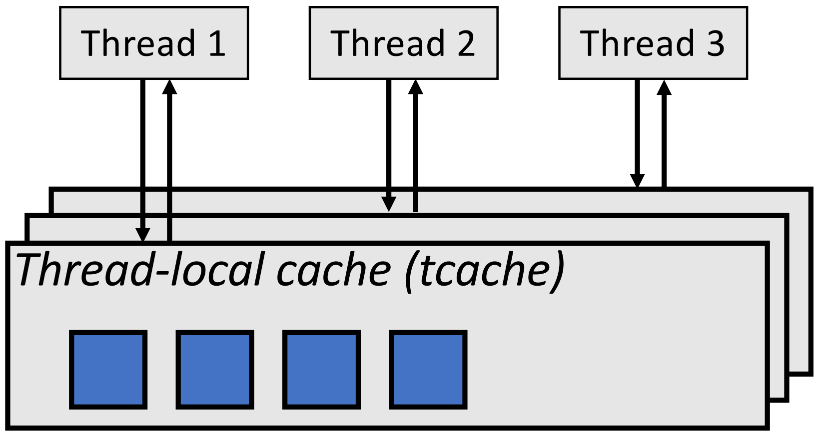


Persistent Heap Memory Management

- Existing persistent memory allocators inherit mature designs from volatile memory allocators.
  - Large allocations (e.g., ≥ 16 KiB) are served by large memory regions in ***chunks***.
  - Small allocations (e.g., < 16 KiB) are served by small memory regions in ***slabs***.
  - They use thread-local cache (or **tcache**) to reduce lock contention in small allocations.







Characteristics of Real PMEM Hardware

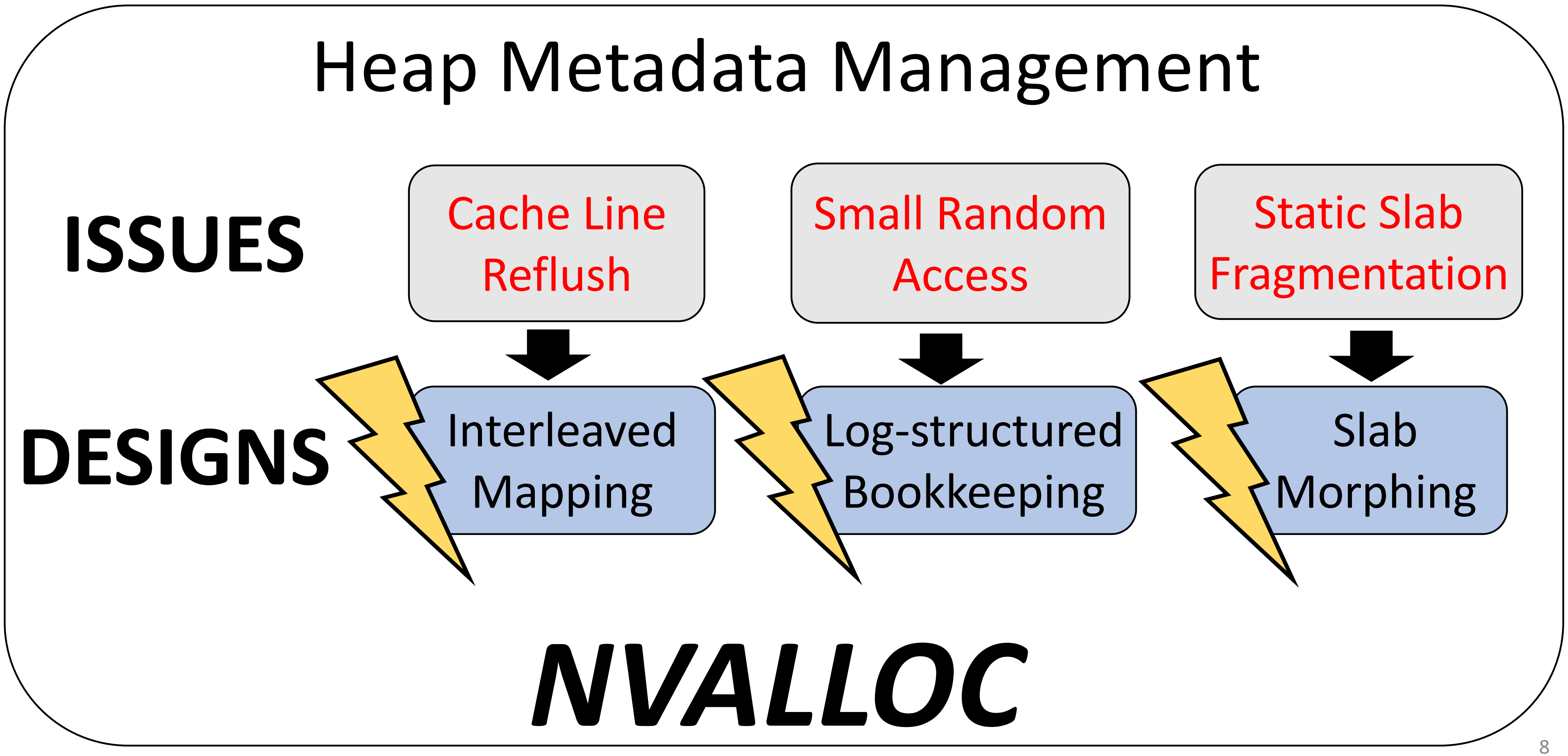
Intel® Optane™ DCPMM

- Better sequential access performance than random[1].
- Repeated cache line flush (reflush) latency is 8x higher[2].

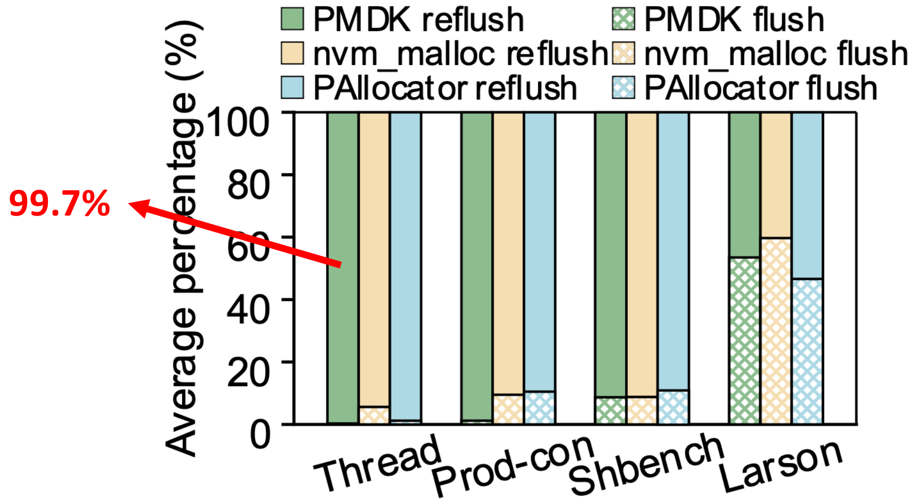
Existing allocators are not aware of these PMEM characteristics

[1] "An Empirical Guide to the Behavior and Use of Scalable Persistent Memory", FAST'20
[2] "FlatStore: An Efficient Log-Structured Key-Value Storage Engine for Persistent Memory", ASPLOS'20

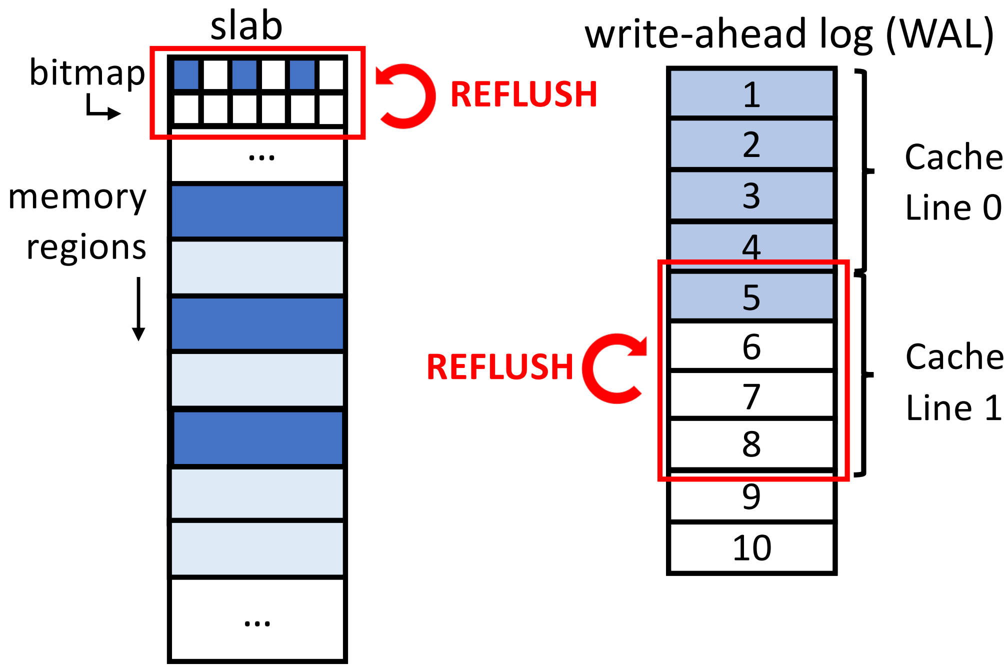
Objectives of Our Work



Observation I: Cache Line Reflush Frequently Happens in Small Allocations



Ratio of cache line reflush in well-known benchmarks

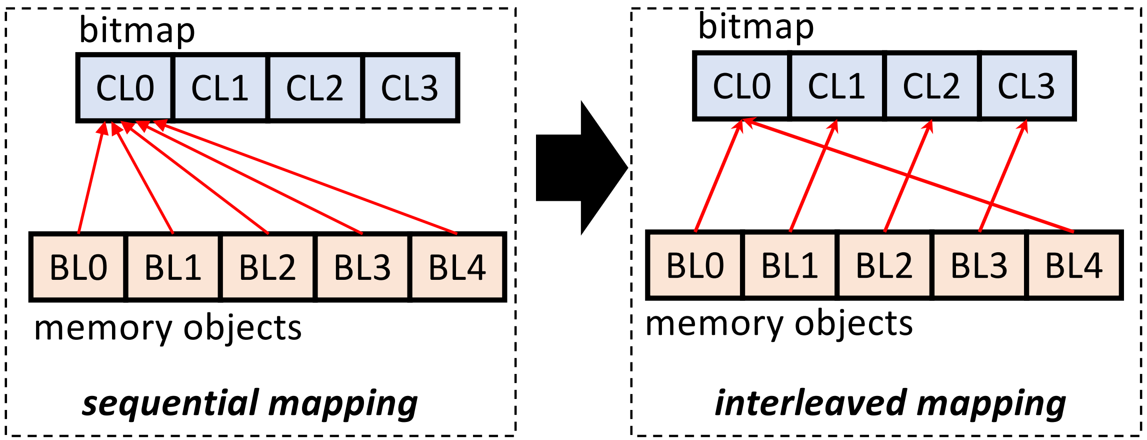


Reflush frequently happens for updating bitmaps in slab header and WALs.

Optimization I: Interleaved Mapping

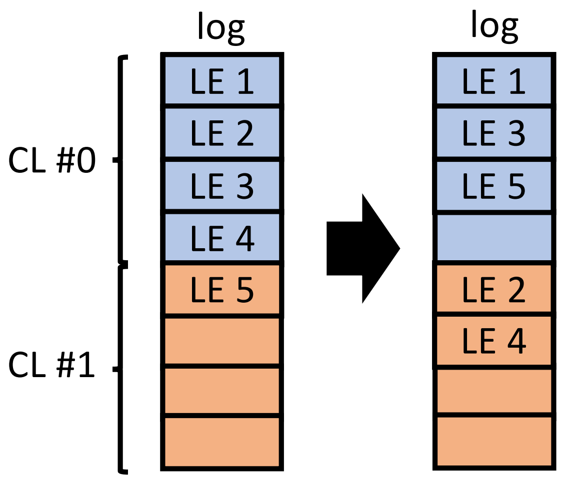
- Key insight: the metadata of consecutive objects do not need to be consecutive.
- *Interleaved tcache layout* => check the paper!

CL: Cache Line BL: Block



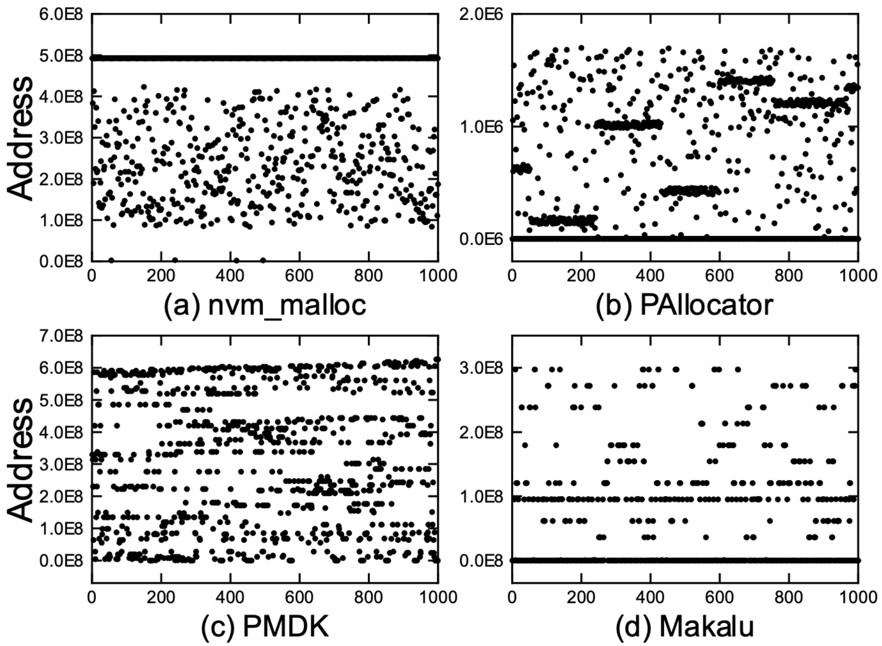
(a) Interleaved mapping in slab headers

LE: Log Entry

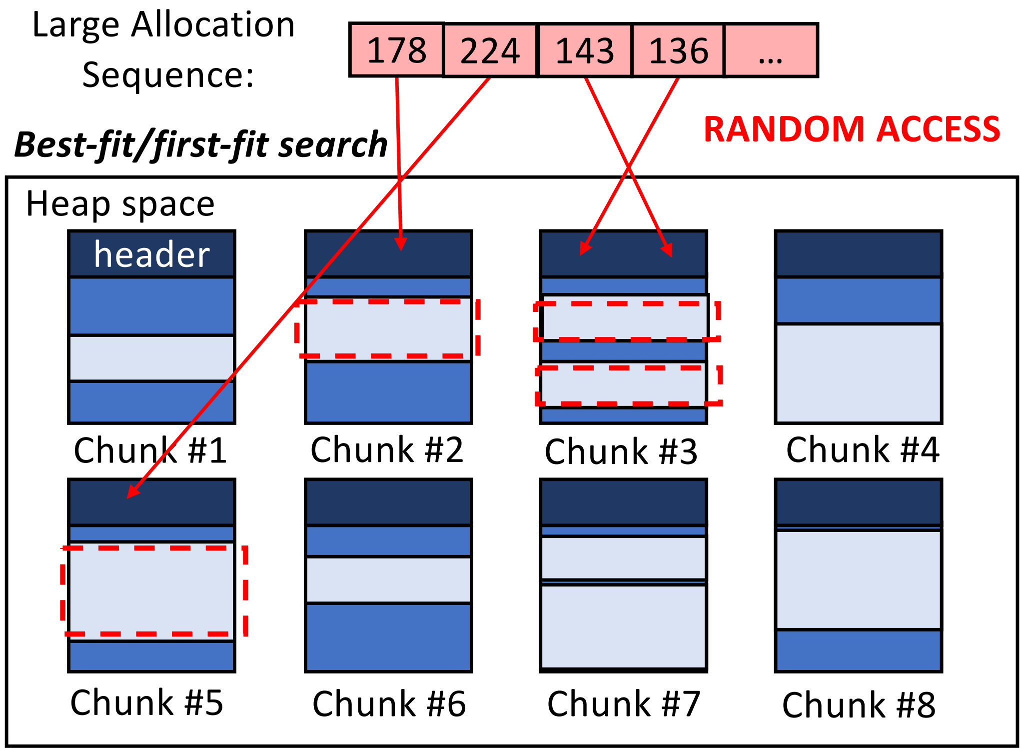


(b) Interleaved mapping in WALs

Observation II: Small Random Access Frequently Happens in Large Allocations



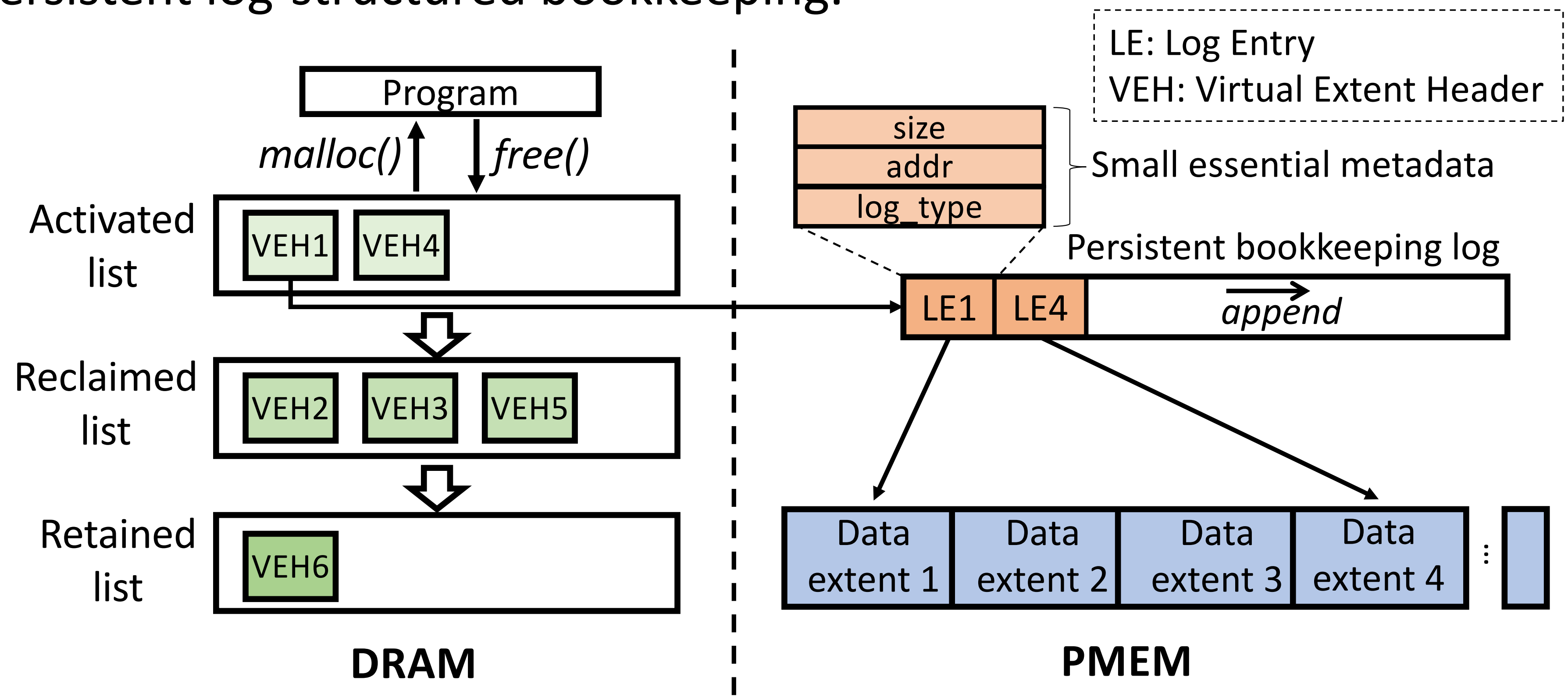
Small random access to PMEM in DBMStest



The allocation algorithms (e.g., best-fit) and in-place bookkeeping modification causes small random access.

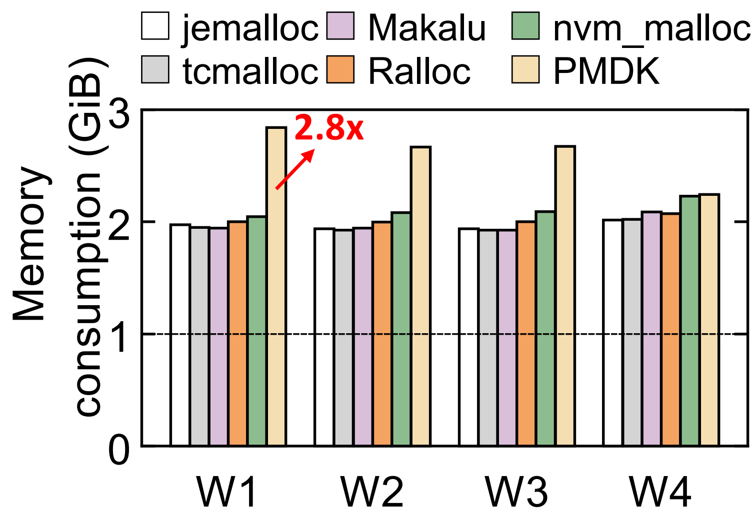
Optimization II: Log-structured Bookkeeping

- NVAlloc decouples large allocation metadata into volatile indexes and persistent log-structured bookkeeping.



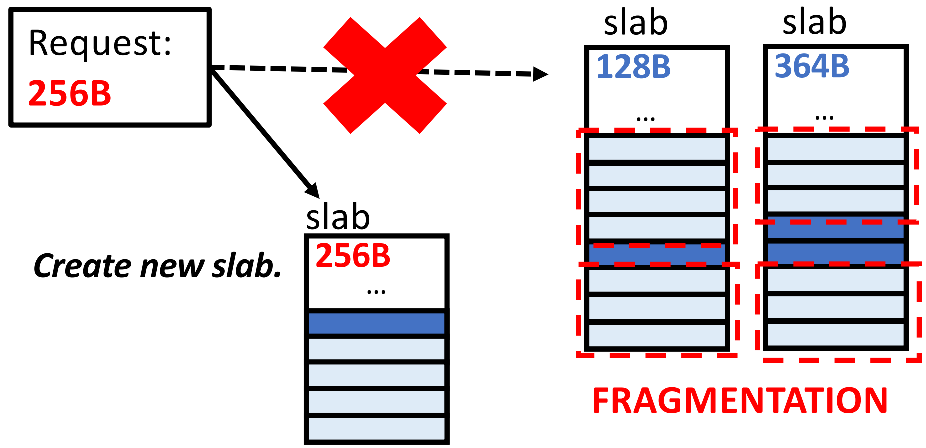
Observation III: Persistent Memory Fragmentation in Small Allocations

Fragmentation cannot be eliminated by restarting the system for PMEM



Memory usage in Fragbench (1GiB for real usage)

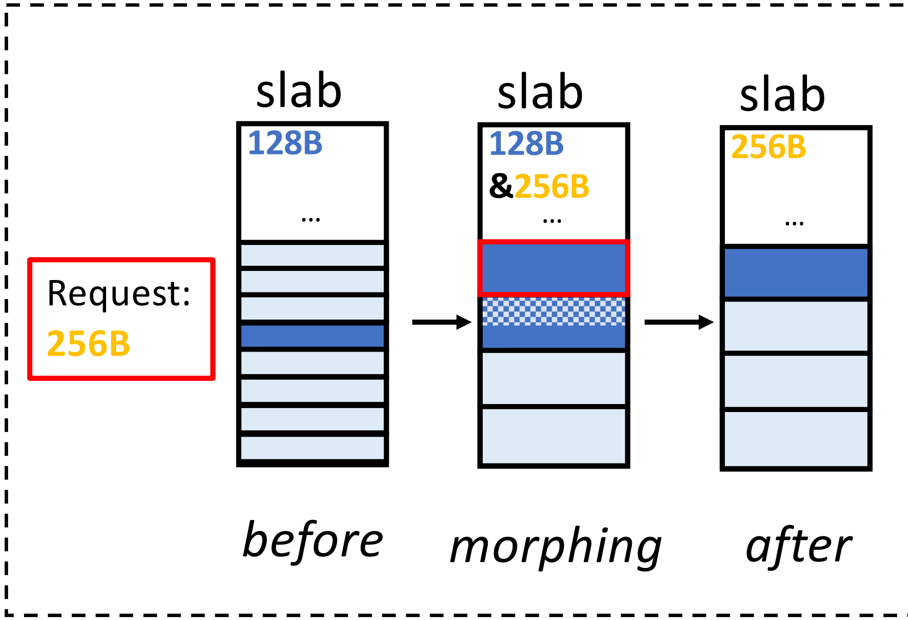
Mis-matched slabs cannot serve the request even they are mostly empty.



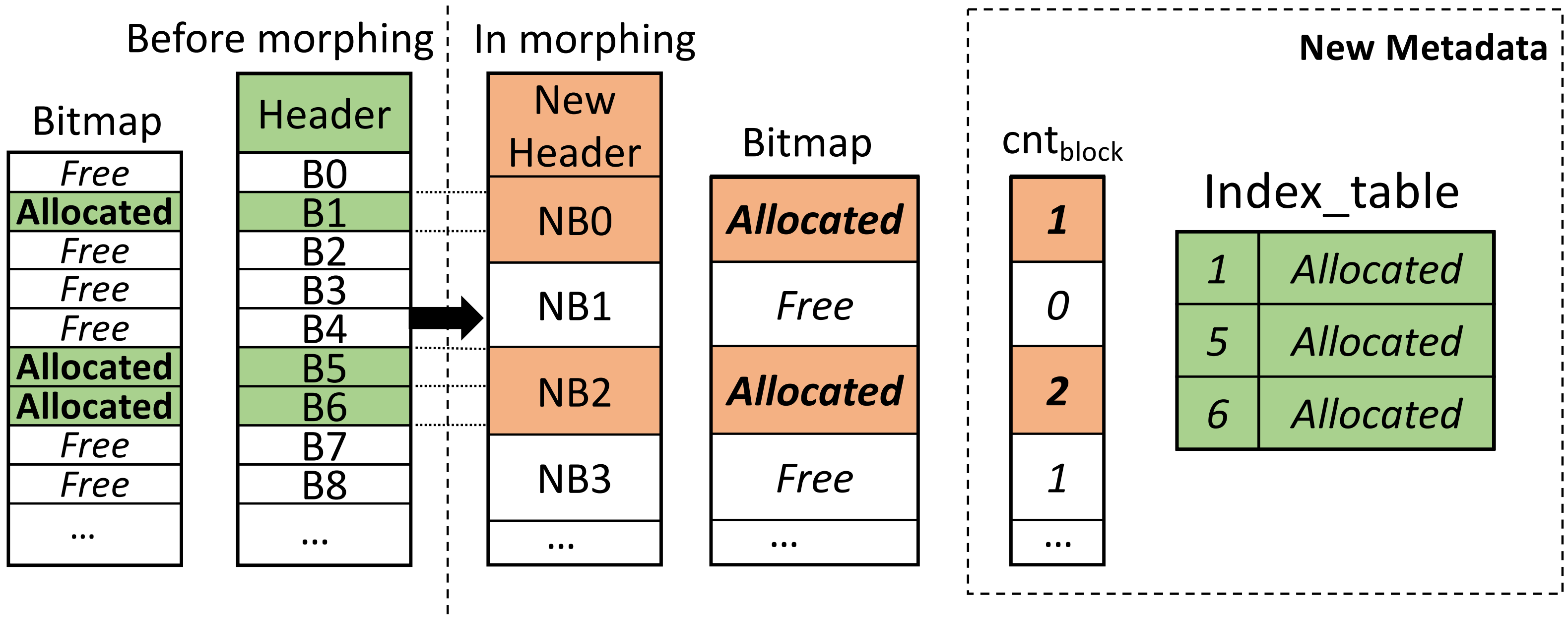
Static slab segregation causes fragmentation under varying allocation pattern.

Optimization III: Slab Morphing

- NVAlloc allows a slab of low memory usage to be transformed to a slab of another size class using *slab morphing*.
- During the transformation, the slab may store two types of data blocks of different sizes.
- Challenge:
  - Correctness of indexing two types of blocks in one slab.
  - Minimize the overhead.



Optimization III: Slab Morphing



- NVAlloc adds new metadata in slab header to manage the blocks during morphing.
- The release of old blocks (e.g., B0) has a low cost because they are small in number.
- The allocation and release of new blocks have no extra overhead because only the bitmap is used in the process.

Evaluation Setup

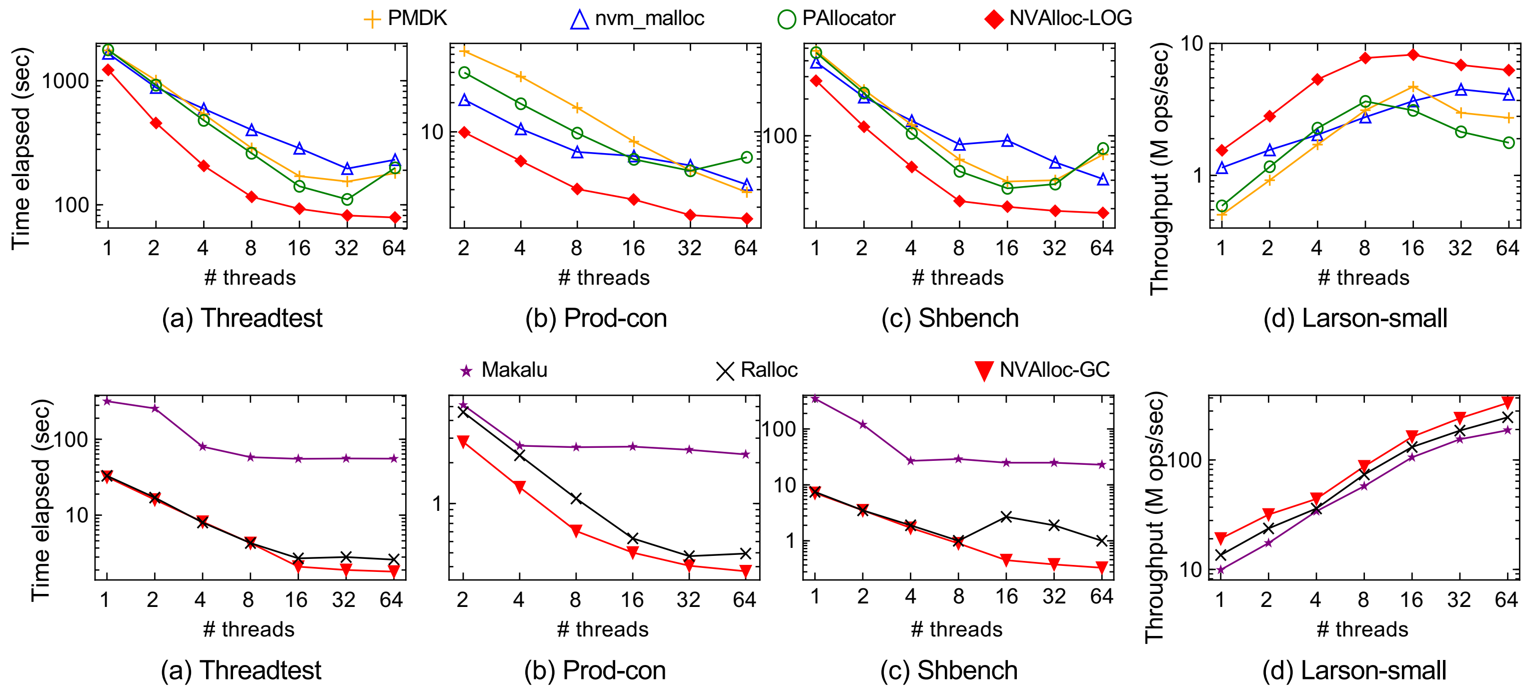
- Platform
  - CPU: Intel Xeon Gold 5218R
  - DRAM: 4*16 GB 2666 Mhz
  - PMEM: 2*128 GB Intel Optane DCPMMs
- Workloads

Categories	Workloads	Size
Small Allocation	Threadtest	64 B
	Prod_con	64 B
	Shbench	64~1000 B
	Larson (small)	64~256 B
Large Allocation	Larson (large)	32~512 KB
	DBMStest	32~512 KB
Memory Usage	Fragbench	100~2000B

- Compared Allocators

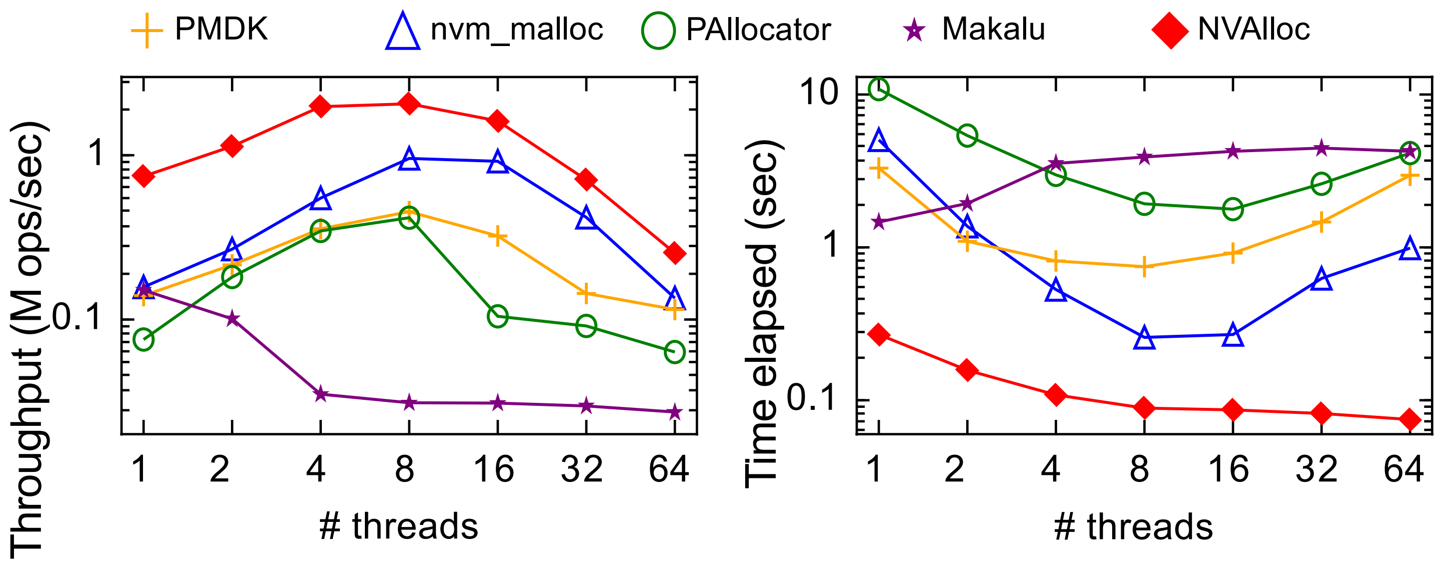
Consistency Model	Counterparts
WAL	nvm_malloc
	PAllocator
	NVAlloc-LOG
Post-crash GC	Makalu
	ralloc
	NVAlloc-GC
Internal Collection	PMDK
	NVAlloc-LOG

Performance Results: Small Allocation



For small allocation, NVAlloc outperforms existing allocators by up to 6.4x and 3x in average.

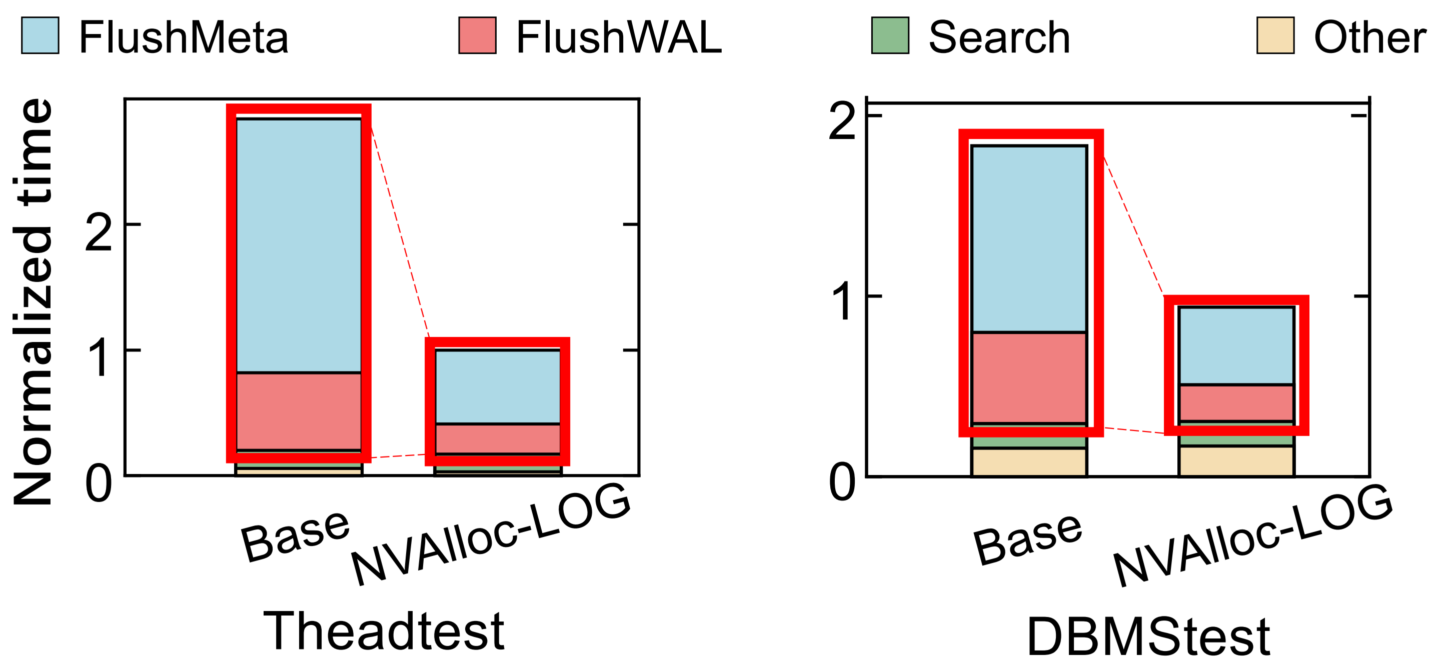
Performance Results: Large Allocation



(a) Larson-large (b) DBMStest

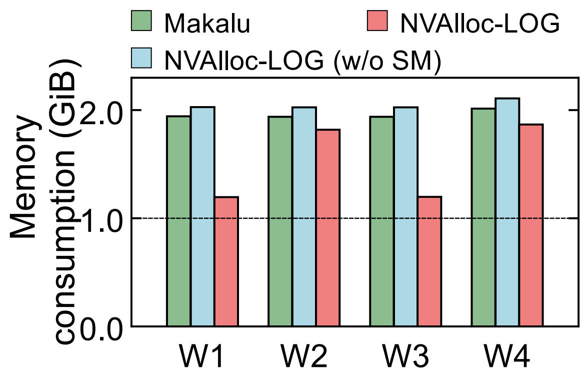
For large allocation, NVAlloc outperforms existing allocators by up to 57x and 5x in average.

Performance Breakdown

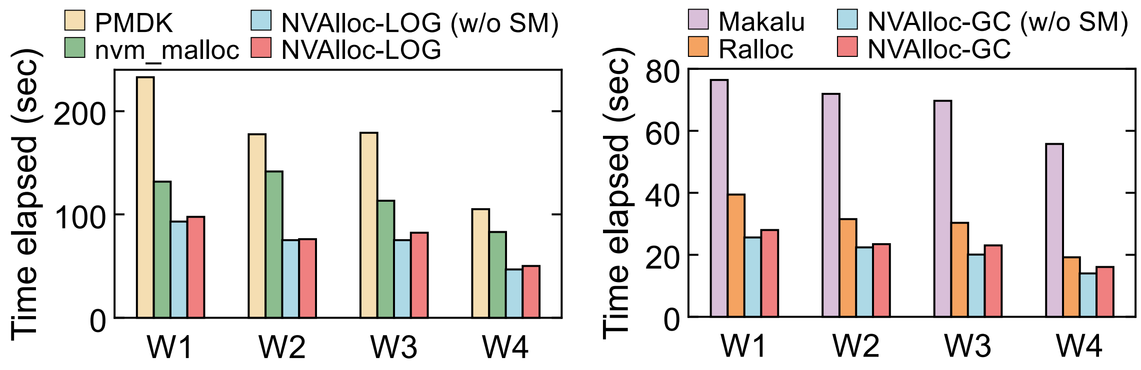


- The performance improvement is mainly due to the reduction of the flush time.
  - For small allocation, the cache line reflushes are eliminated.
  - For large allocation, the small random flushes to PMEM are eliminated.

Performance Results: Memory Usage



(a) Memory usage with Fragbench



(b) Performance with Fragbench

NVAlloc reduces memory usage by up to 57.8% with performance overhead less than 5%.

Summary

- We propose NVAlloc to address performance and memory usage issues in existing persistent memory allocators.
  - Interleaved mapping => cache line repeated flush.
  - Log-structured bookkeeping => small random access.
  - Slab morphing => static slab fragmentation.
- NVAlloc significantly speeds up small and large allocations and reduces memory usage.

- The source code for NVAlloc is available at https://github.com/ISCS-ZJU/NVAlloc with 8 KLOC.

NVAlloc: Rethinking Heap Metadata Management in Persistent Memory Allocators

Thanks for watching!

Contact me at: dangzheng@zju.edu.cn

浙江大学 Zhejiang University

Washington State University

Illinois Tech