Adapting System Software to Evolving Memory Architectures

NVRAMOS '25

Soongsil University

Eunji Lee

Contents

- Avoiding Pitfalls in Networked Key-Value Store for Tiered Memory (CLOUD ’25)
- Revisiting Trim for CXL Memory (HotStorage ‘25)

Avoiding Pitfalls in Networked Key-Value Store for Tiered Memory

IEEE CLOUD '25

[1]Seungmin Shin, [1]Leeju Kim, [1]Wookyung Lee, Eyee Hyun Nam, Seungmin Kim, Bryan S. Kim, Sungjin Lee, Eunji Lee

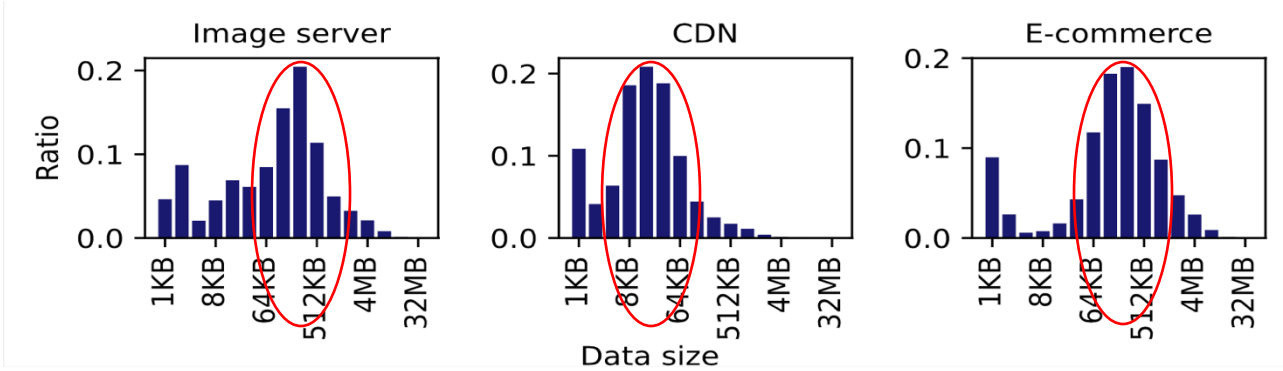
[1] **Equal Contribution / Soongsil University**

In-memory Key-value Store

- Critical component in modern applications
- Redis, Memcached, MemC3
- Maintain ALL data in memory (DRAM)
- Widely used in web/ML service providers

Objects are Getting Larger!

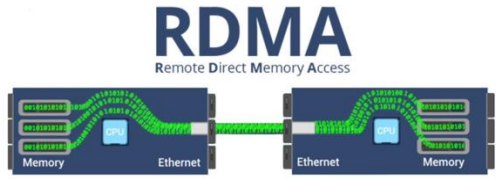
- Size of data is increasing to several KB
  - Object size distribution in CDN (provided by WineSOFT)

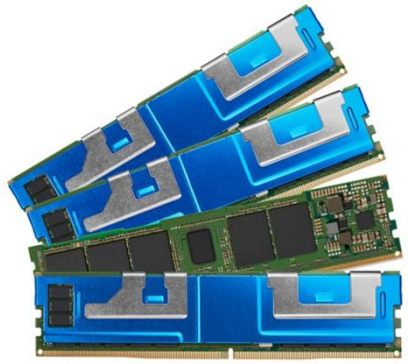


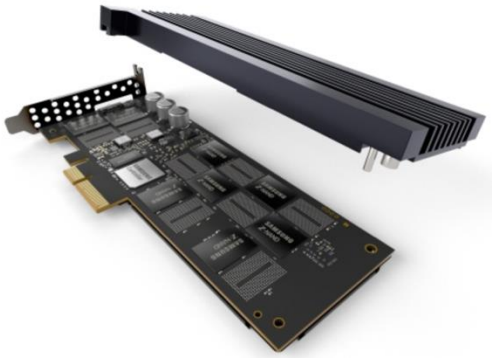
- Also evident in ML-based systems
  - GPTCache leverages KVS to cache embedding vectors
    ➔ 64–128 bytes (2018) to 16KB (2024)
  - UP2X, a distributed KVS in Meta, handles profile data for AI/ML inferences
    ➔ average object size of 3.6KB, some exceeding 100KB

Tiered Memory is Commonplace

- Memory capacity can no longer meet data-centric application demands
- Proper data placement across heterogeneous memory is becoming important
  - hot / cold separation
  - promotion / demotion / migration







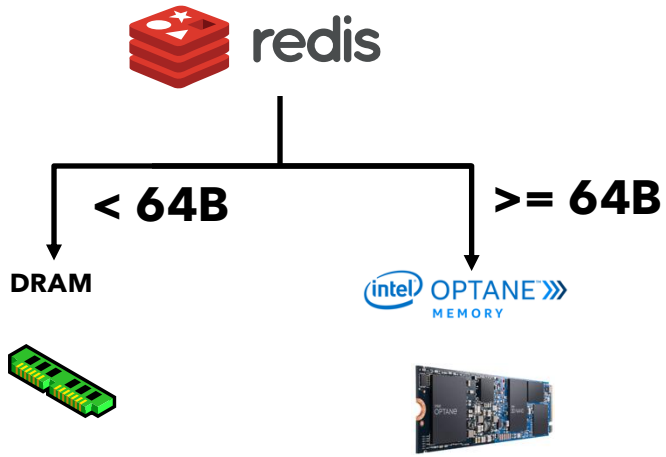
Motivation

- KVS for Tiered Memory

KVS	Impl.	Performance Tier	Capacity Tier	Threshold
Tair-Pmem [10]	Variant of Redis [11]	Index	Value, Index	N/A
Pmem-redis [12]	Variant of Redis [11]	Index, Small value	Large value	64 bytes
TieredMemDB [13]	Variant of Redis [11]	Index, Small value	Large value	64 bytes
X-mem [14]	Integrated into MemC3 [15]	Index, Small value	Large value	64 bytes
BonsaiKV [16]	Custom	Index	Index, Small value in log, Large value	256 bytes
FlatStore [17]	Custom	Index, Small value in buffer	Small value in log, Large value	256 bytes
ListDB [18]	Custom	Index, Value	Index, Value	N/A
Pmem-rocksdb [19]	RocksDB [20]	-	Index, Value	N/A
BadgerDB [21]	Wisckey [22]	Index, Small value	Large value	4 KB
Memcached Extstore [23]	Memcached [24]	Index	Value	N/A
Cachelib [25]	Memcached [24]	Index, Frequently accessed value	Less frequently accessed value	N/A

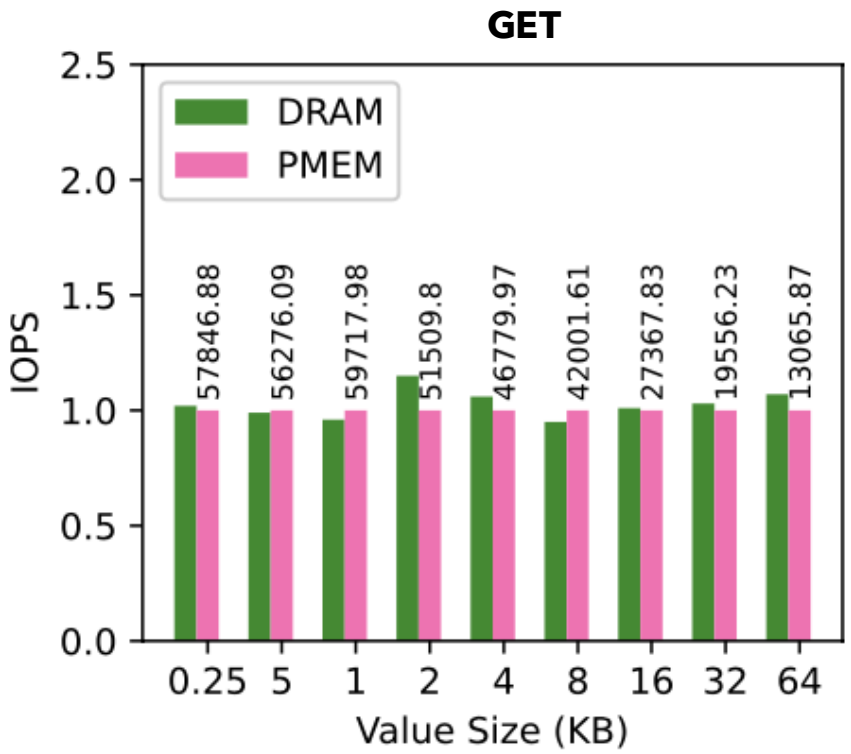
TABLE I: Data placement policies in key-value stores for tiered memory.

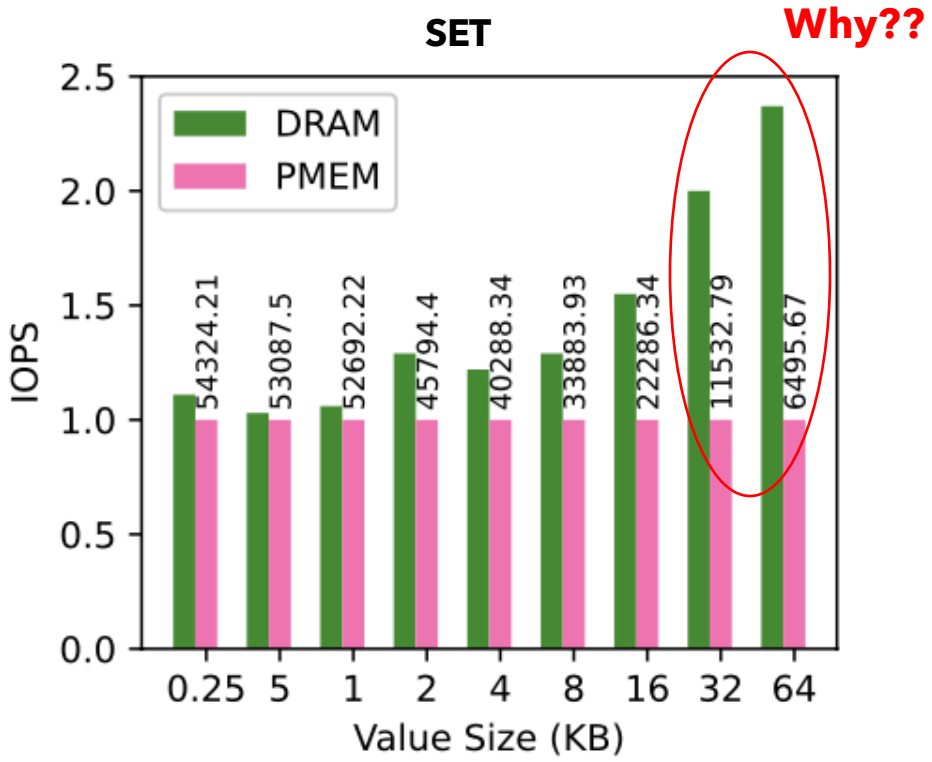
- Empirical study with Redis as case study
  - Pmem-redis (developed by Intel)
  - Supermicro SYS-1029U-TRT
  - 128GB DRAM / 512GB DCPMM
  - Connected with 56 Gb Ethernet



Performance Analysis

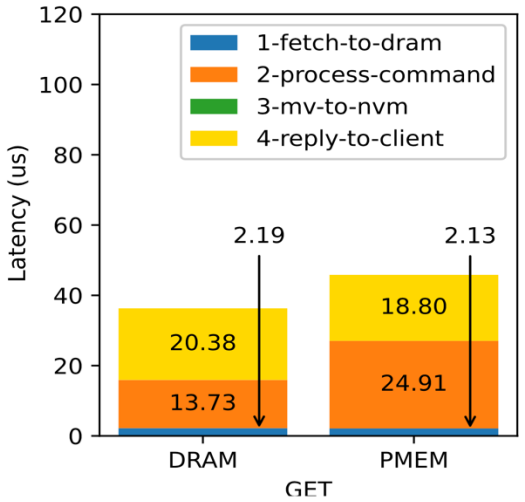
- Get operations are nearly identical
- Set operations are significantly different – *large-sized data suffers from serious performance drop*

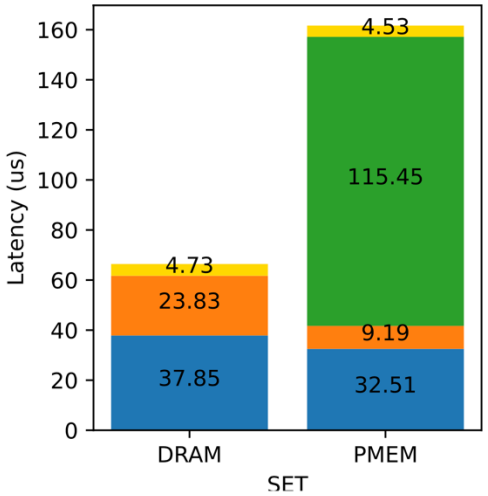




Latency Breakdown

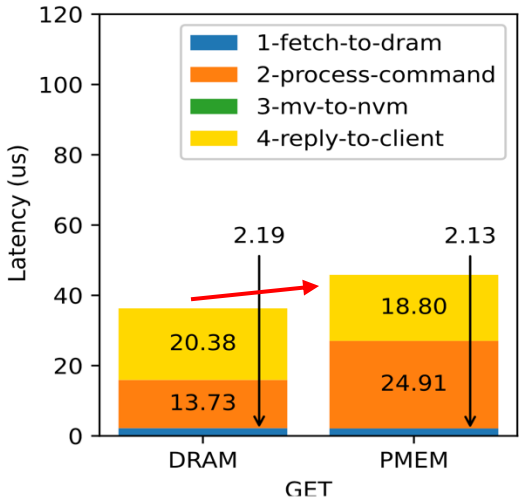
- 64KB data size, average latency of 40K Ops.
- Steps
  1. **Fetch-to-dram** : Read packets from kernel's socket buffer
  2. **Process-command** : Parse and handle data according to command
  3. **Mv-to-nvm** : Copy the data to DCPMM if data size exceed threshold
  4. **Reply-to-client** : Send reply to cleint after processing operations

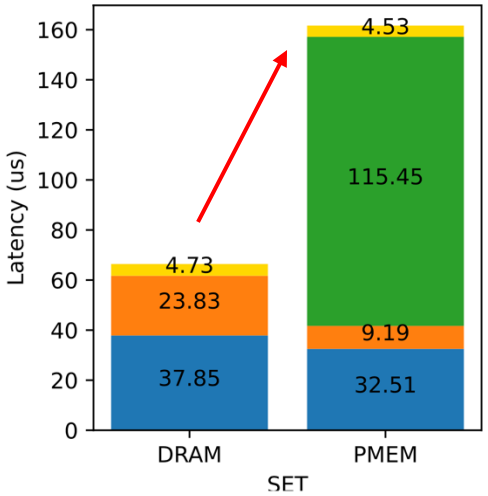




Latency Breakdown

- 64KB data size, average latency of 40K Ops.
- Steps
  1. **Fetch-to-dram** : Read packets from kernel's socket buffer
  2. **Process-command** : Parse and handle data according to command
  3. **Mv-to-nvm** : Copy the data to DCPMM if data size exceed threshold
  4. **Reply-to-client** : Send reply to client after processing operations

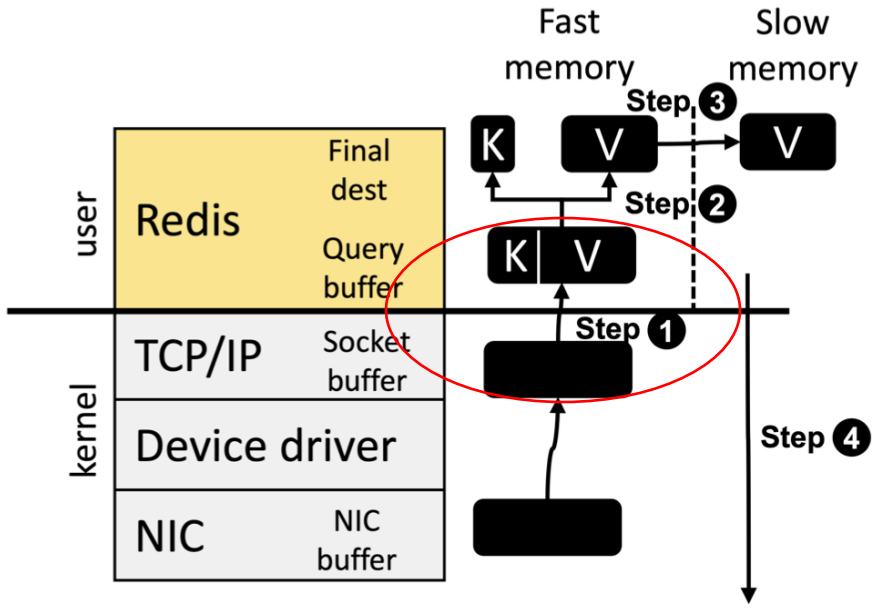




Three Performance-Degrading Pitfalls

1) Transient Data Staging

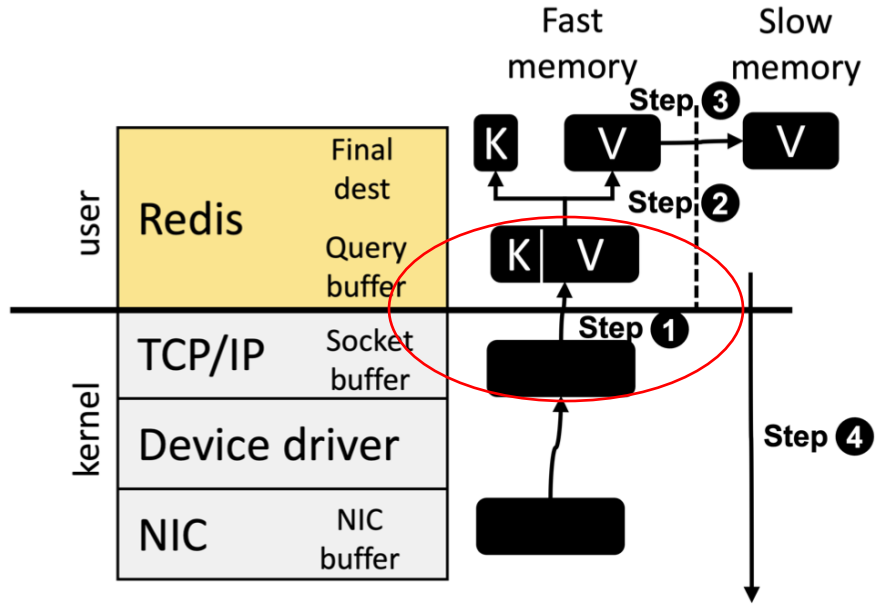
- Requests are delivered via the network, encapsulated in *packets* in modern KVS
- Determining final destination of a packet requires specific information (*e.g. value size*)
- Socket interface hides these details until packets reaches user space
- Data for other tiers is redundantly copied in DRAM!

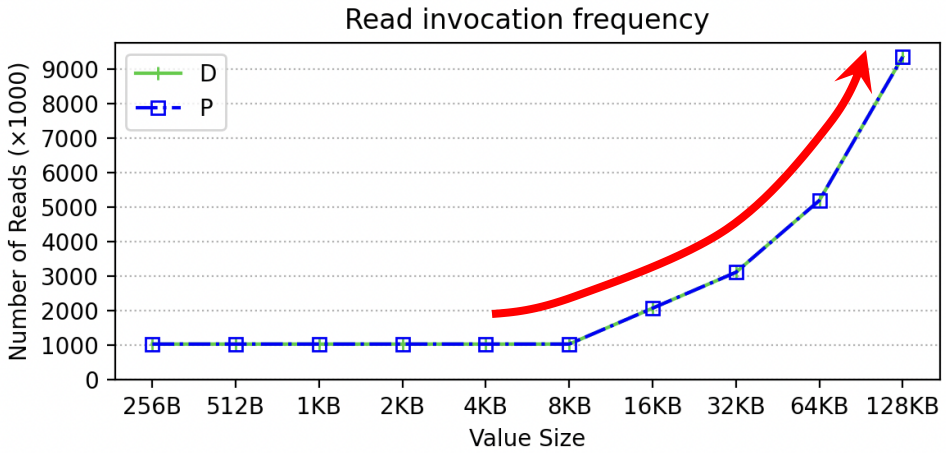


Three Performance-Degrading Pitfalls

2) Repetitive Kernel/User Crossings

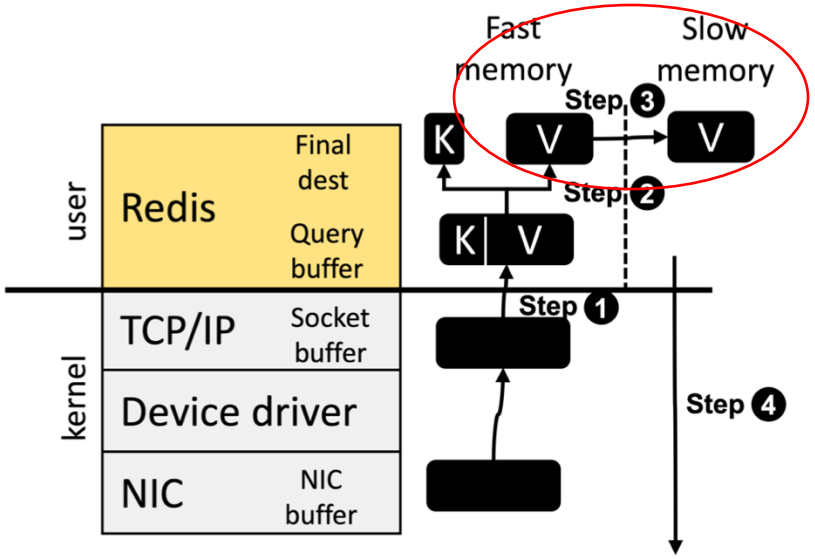
- Unknown data size!
- Heuristic buffer management
  - 16KB / 128KB / 32KB
  - Reallocating and merging memory for consecutive objects is costly
- Repetitive read system calls
  - Redis fixes read calls at 16 KB
- Socket interface hides these details until packets reaches user space!





Three Performance-Degrading Pitfalls

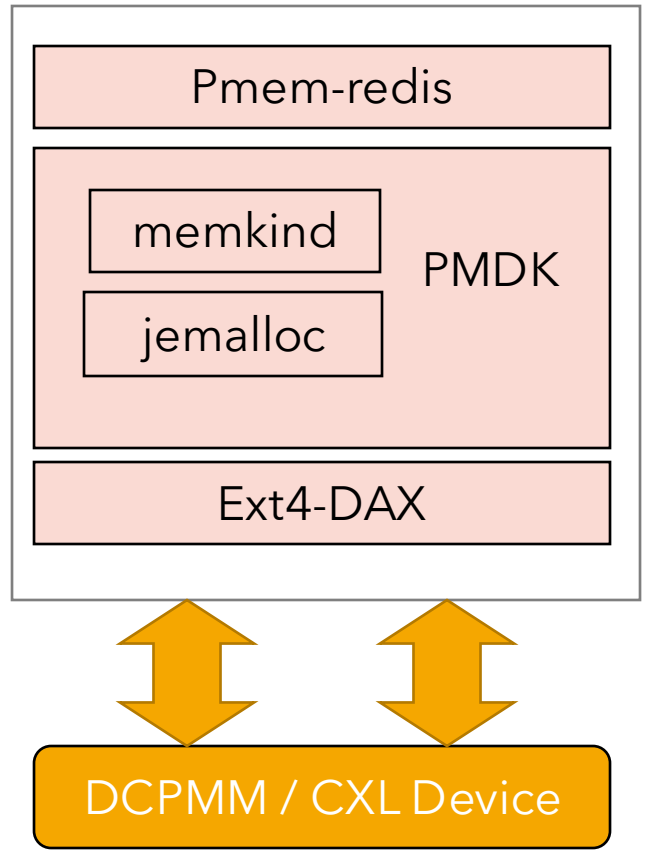
3) Zero-write bomb!



Three Performance-Degrading Pitfalls

3) Zero-write bomb!

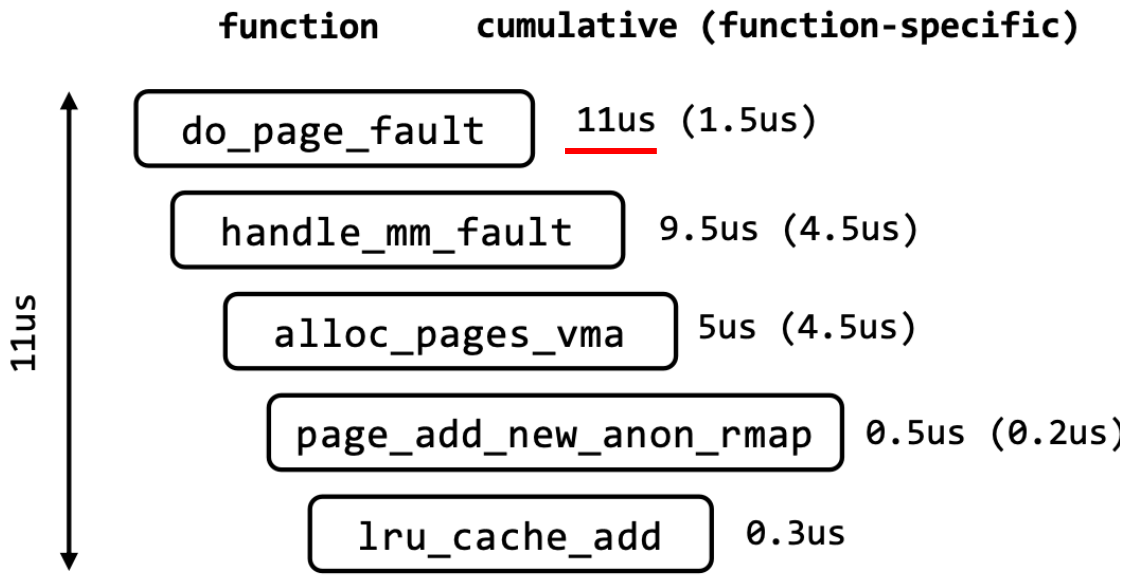
- Memory initialization for anonymous pages
- Use memory-mapped file as a memory space in DCPMM (`posix_ftruncate/mmap`)
- Extending file size using `posix_fallocate` (by 2MB huge page)
- Page fault on first access causes physical block allocation and hundreds of zero-writes to block device for initialization!



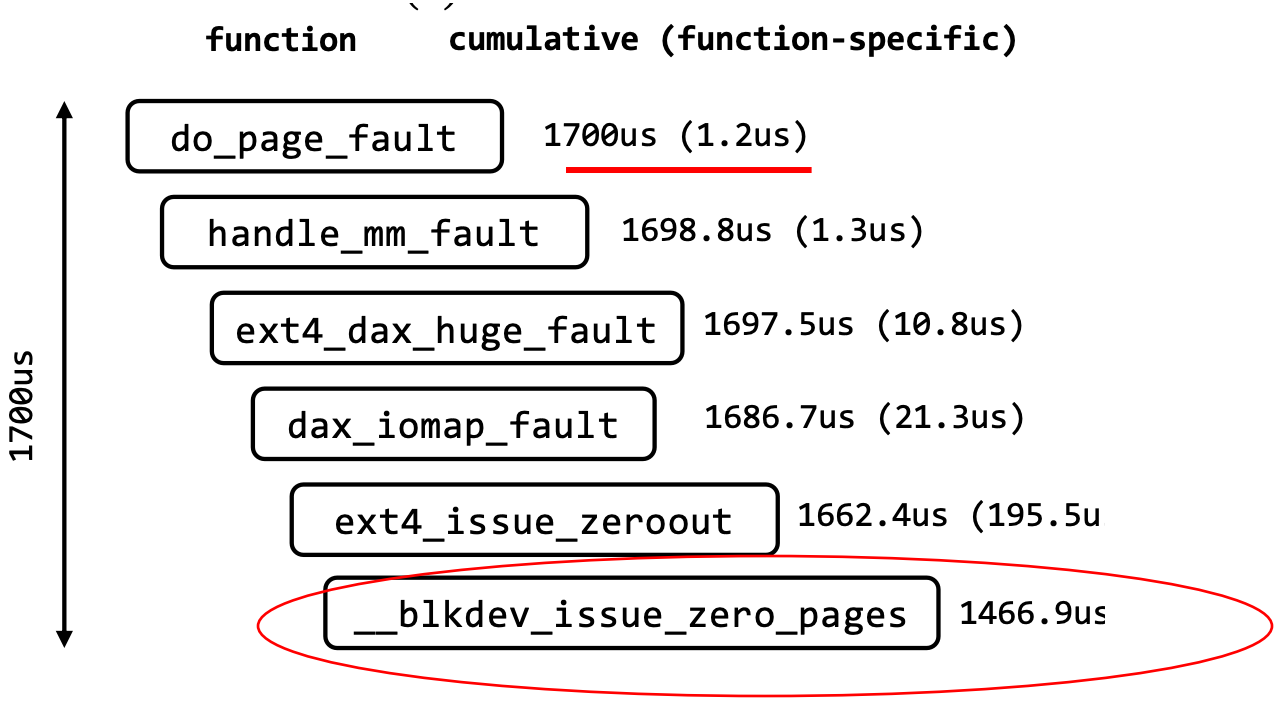
Software stack for DAX device

Three Performance-Degrading Pitfalls

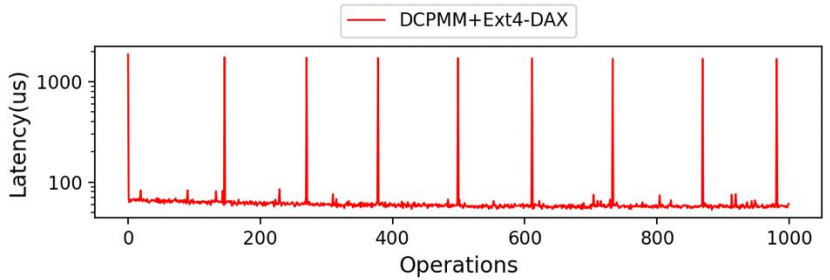
- **3) Zero-write bomb!**



(a) DRAM



(b) DCPMM

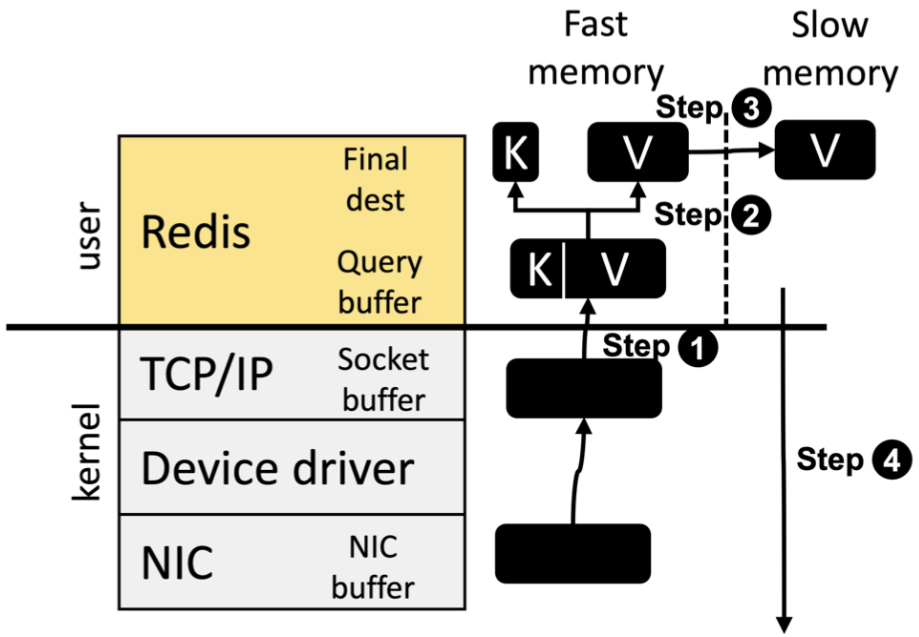


X-redis

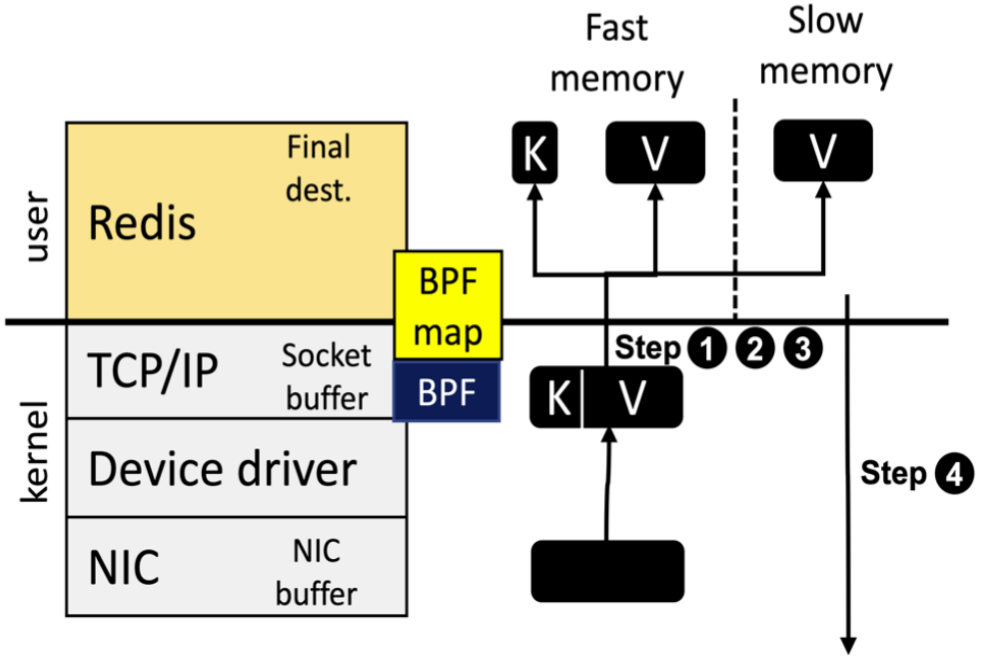
- PPF (Packet Peek and Forward)
  - Peek at packets in kernel layer with eBPF
  - Streamline data placement decisions on tiered memory
- OMA (Opportune Memory Allocator)
  - Move zeroing off the critical path with opportune pre-initialization

PPF: Packet Peek and Forward

- Peek at packets in the kernel layer using Linux eBPF
- Get the information for data placement without popping
- Eliminate unnecessary data copies and read system calls



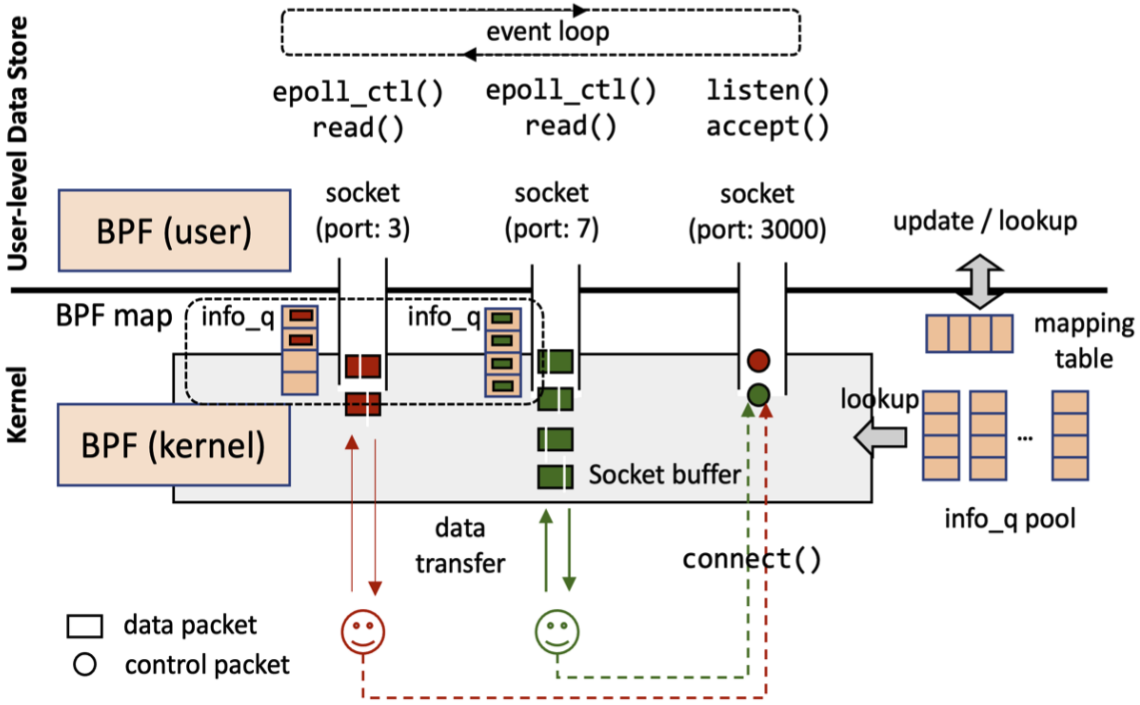
Pmem-redis (w.o. PPF)



X-redis (w. PPF)

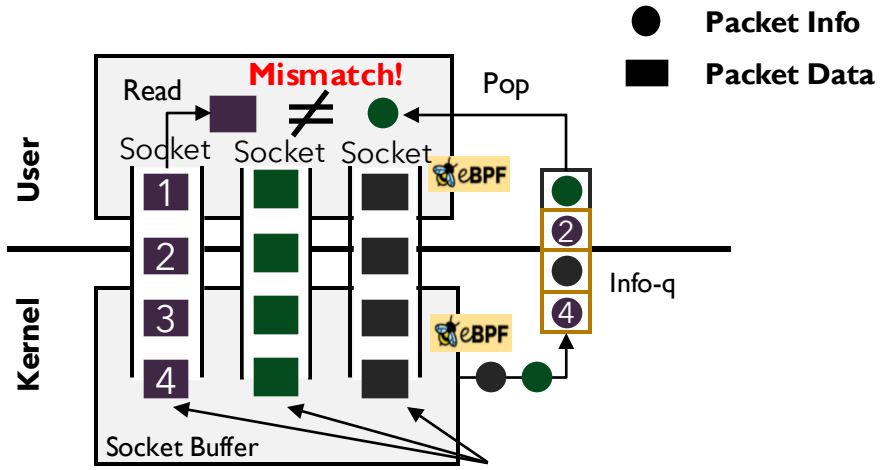
PPF: Packet Peek and Forward

- BPF Component Type
  - BPF_PROG_TYPE_SOCKET_FILTER
  - Fully leverage in-kernel network layer
- BPF Map
  - Accessible to both user and kernel
  - Store value size and layout in eBPF map
  - BPF_MAP_TYPE_QUEUE (info_q)
  - BPF_MAP_TYPE_HASH (mapping table)

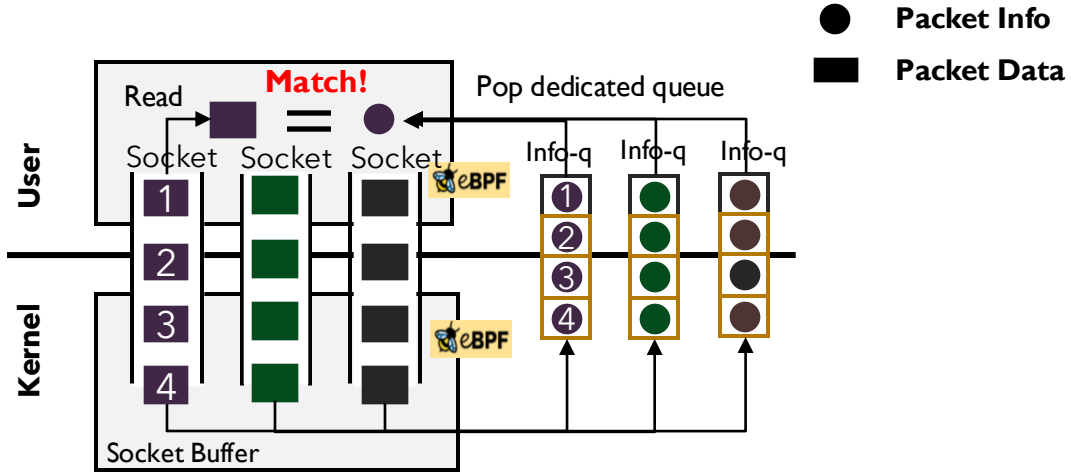


PPF: Packet Peek and Forward

- Challenge 1. Order of Packet Processing
  - Information dequeued from Info-q must match packet waiting for processing
  - Packet has no ID → *Hard to match!*
  - Allocate an info-queue per socket using its port number upon connection establishment



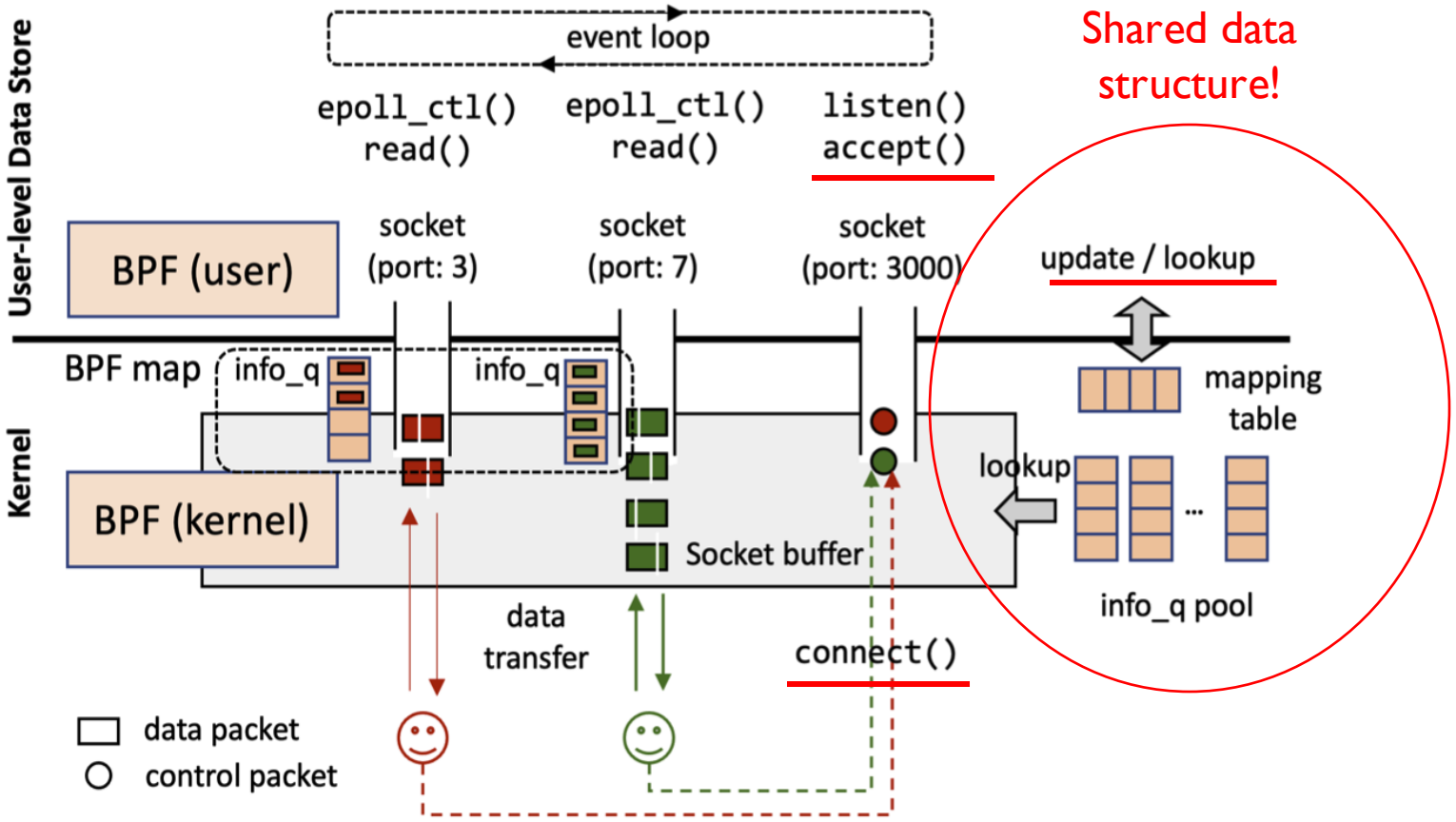
Shared info-q



Separate info-q per socket

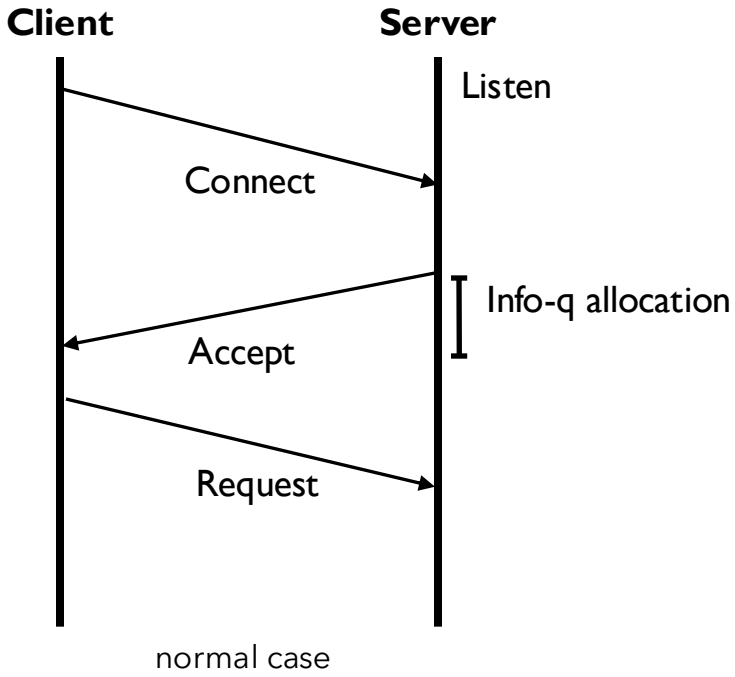
PPF: Packet Peek and Forward

- Challenge 2. Concurrency control to shared eBPF map
  - Dynamic allocation of info-q incurs R/W contention to mapping table

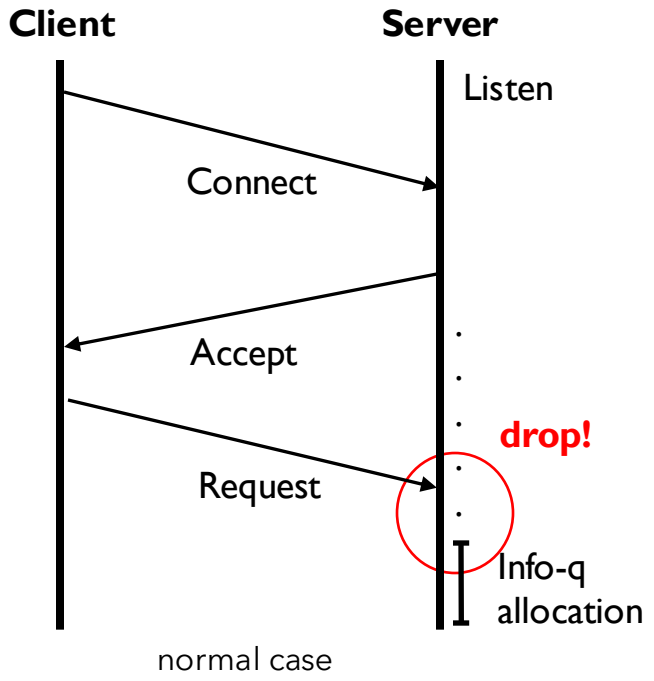


PPF: Packet Peek and Forward

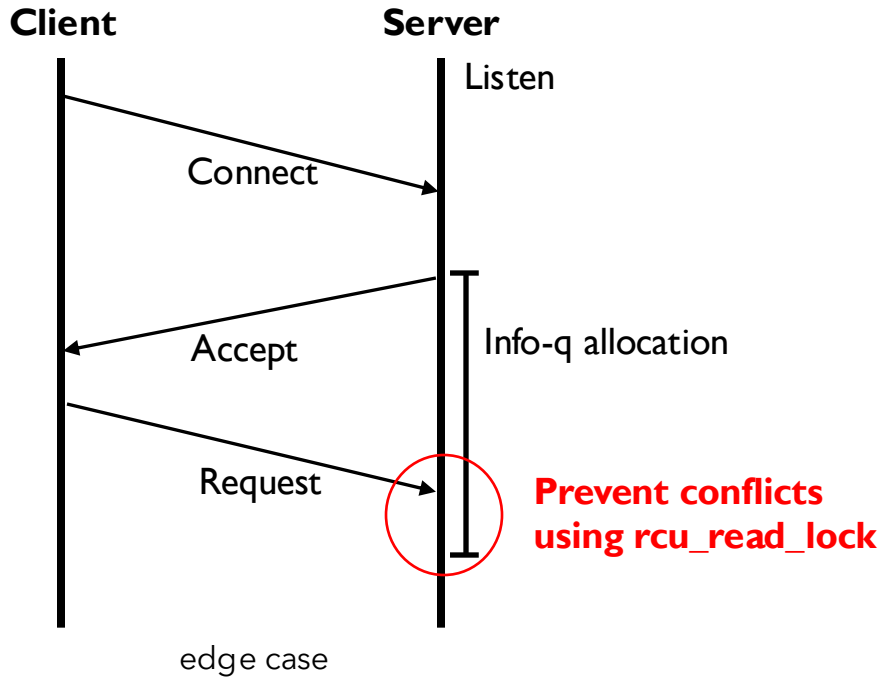
- Challenge 2. Concurrency control to shared eBPF map
  - Dynamic allocation of info-q incurs R/W contention to mapping table



Packet arrives after allocation



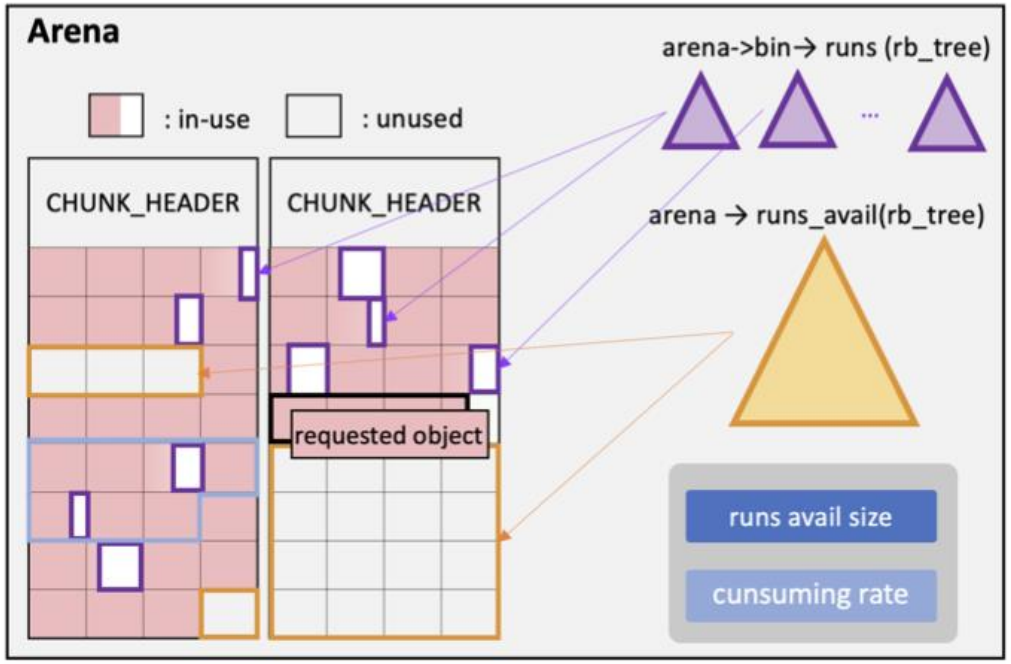
Packet arrives before allocation



Packet arrives during allocation

OMA: Opportune Memory Allocator

- Lump-sum approach – initialization all at boot time or deallocation
  - Incur unnecessary writes / external fragmentation!
- OMA
  - Continuously monitors memory usage
  - Proactively allocates and initializes memory before available free space is exhausted
- Integrated into *jemalloc*!
  - Hard work



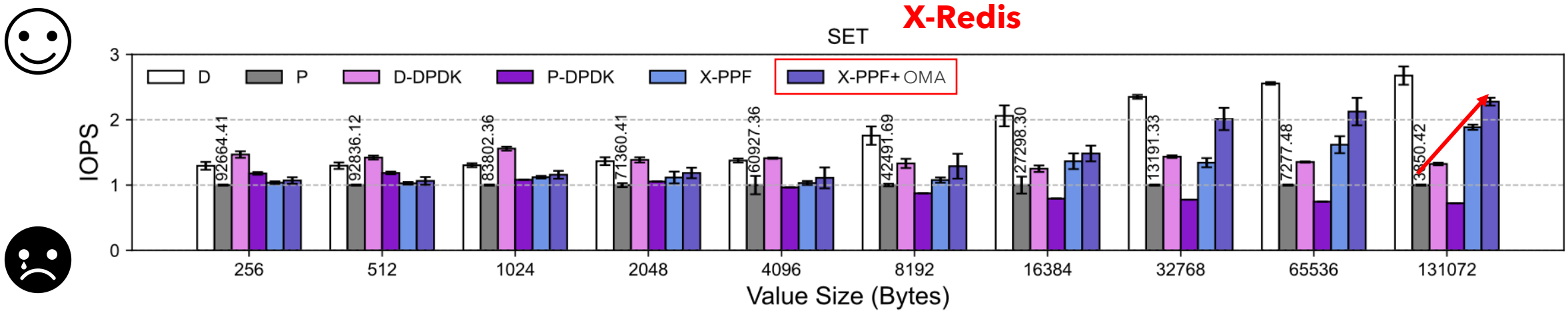
Internal structure of jemalloc

Evaluation

- Workloads
  - Memtier Benchmark
  - Twitter cache trace
  - 8 threads
- Configurations
  - D : Original Redis
  - P : Pmem-Redis (baseline)
  - D-DPDK : Original Redis with DPDK
  - P-DPDK : Pmem-redis with DPDK
  - X-PPF : P with PPF, without OMA
  - X-PPF+OMA : P with PPF and OMA (X-Redis)

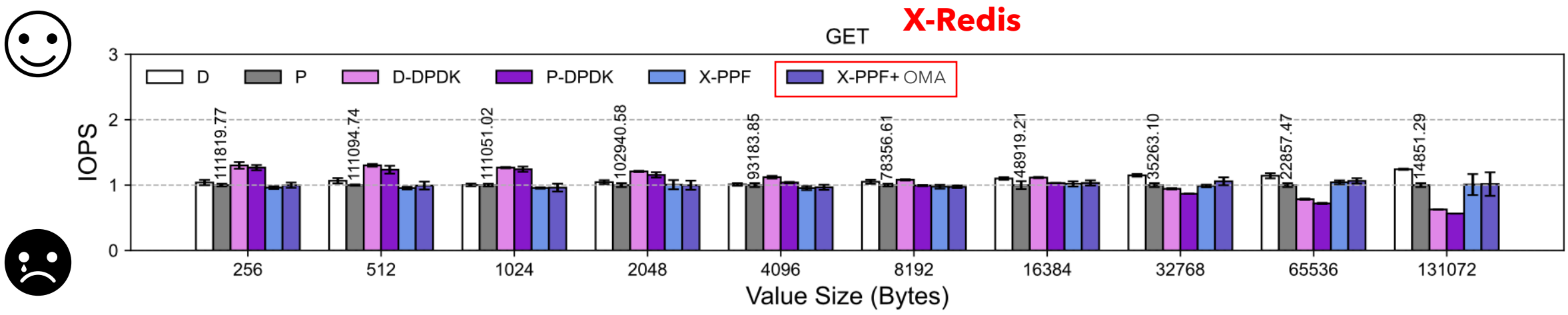
Memtier Benchmark (SET)

- X-Redis outperforms P (Redis using DRAM and DCPMM) in all range
  - Improve performance by 2.28x when value size reach 128KB
- X-Redis performs better than D/P-DPDK for large-sized writes
  - Small-sized buffer of f-stack negatively impacts large write performance
  - Single-threaded packet processing in DPDK when integrated into Redis



Memtier Benchmark (GET)

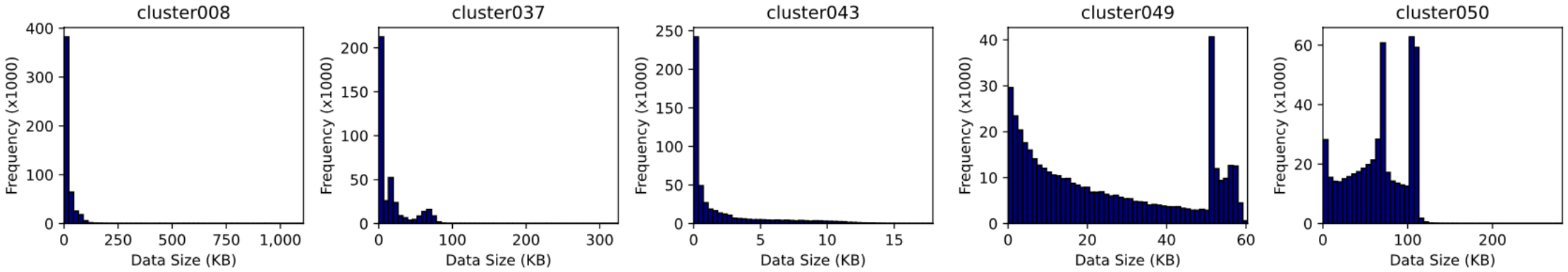
- Performance differences across versions is marginal
- No performance penalty for read requests in X-Redis



Twitter Cache Trace

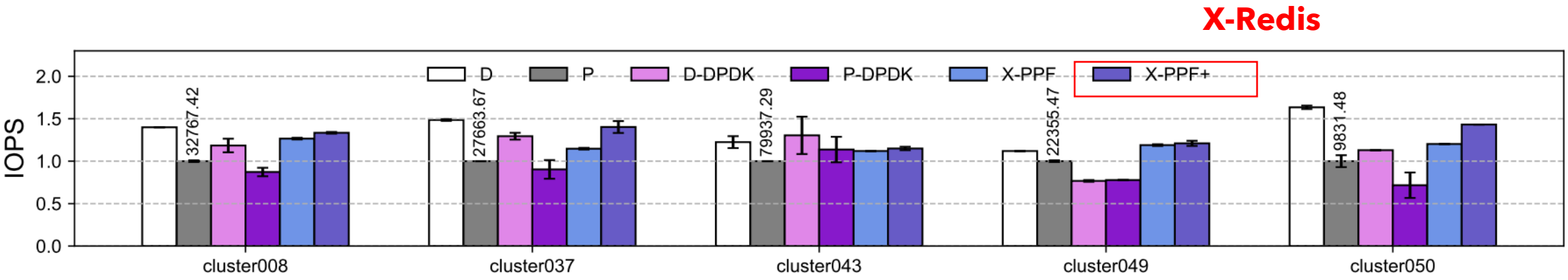
- 4 workload with high set ratio (cluster8, cluster37, cluster49, cluster50)
- 1 workload with small value size (cluster43)
- Mixed set and get operations

Trace	Category	Key Size	Avg. Value Size	SET:GET
C008	Computation	23B	18.2KB	50:50
C037	Computation	72B	16.4KB	27:63
C043	Computation	44B	1.9KB	50:50
C049	Storage	44B	25.4KB	43:57
C050	Computation	18B	67.8KB	48:52



Twitter Cache Trace

- X-Redis provides average 32.7% higher IOPS compared to P
- X-Redis shows performance degradation of only 3.7% compared to DRAM-only redis



Conclusion

- Uncover pitfalls in networked tiered-memory KVS
  - Unnecessary data fetches caused by limited socket interfaces
  - Bulky block writes for on-demand memory initialization
- X-Redis – an enhanced KVS for tiered memory employing two techniques
  - PPF: Packet Peek and Forward
  - OMA: Opportune Memory Allocation
  - Improve write performance substantially while mitigating latency spikes
- Not limited to specific configuration
  - Memcached's CacheLib / Extstore
  - RDMA, CXL Memory, NUMA node
  - Data size / other policies

HotStorage 2025

Revisiting Trim for CXL Memory

Hayan Lee[1], Jungwoo Kim[2], Wookyung Lee[1], Juhyung Park[2], Sanghyuk Jung[3], Jinki Han[3], Bryan S. Kim[4], Sungjin Lee[5], Eunji Lee[1]

[1]Soongsil University, [2]DGIST, [3] Eeum, [4]Syracuse University, [5]POSTECH

Contents

Rising Demand for High-capacity Memory in AI Era

- **Fueled by data-centric and AI/ML applications**

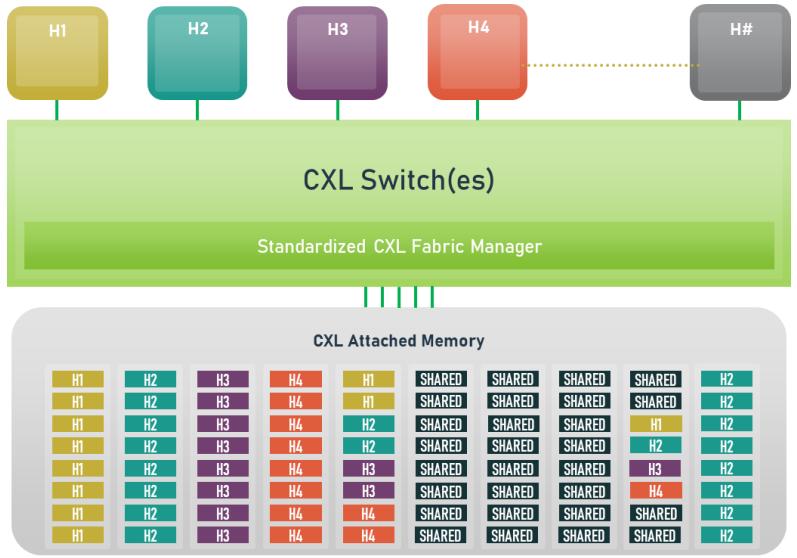
- **Memory disaggregation emerges as a key trend**
  - Realize scalable memory capacity by pooling distributed memory resources
  - Interconnect technologies are attracting significant attention

CXL-Flash

- **Compute Express Link (CXL)**
  - Driven by Intel (2019)
  - Cache-coherent interconnect technology
  - Enables multi hosts and devices to share a common memory space

- **CXL-enabled flash memory**
  - Provide high capacity
  - Hide long latency with intelligent prefetching and data placement
  - E. g. Samsung CMM-H

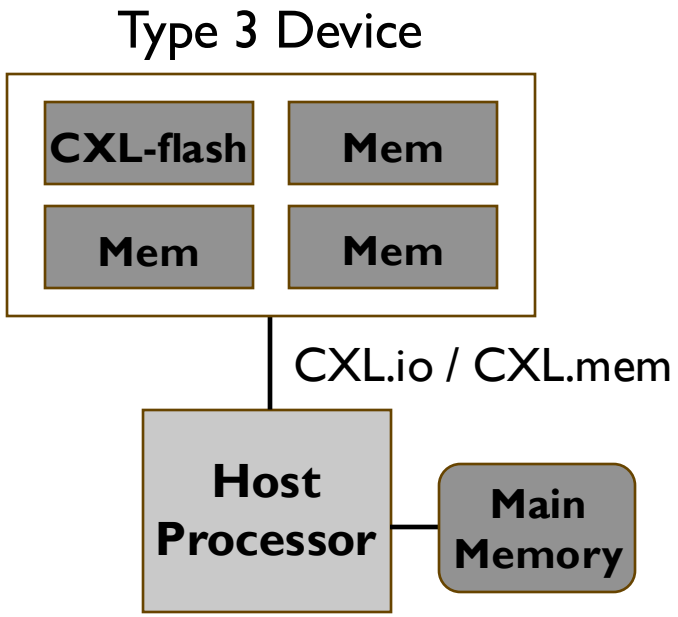
CXL Consortium



https://computeexpresslink.org/blog/explaining-cxl-memory-pooling-and-sharing-1049/

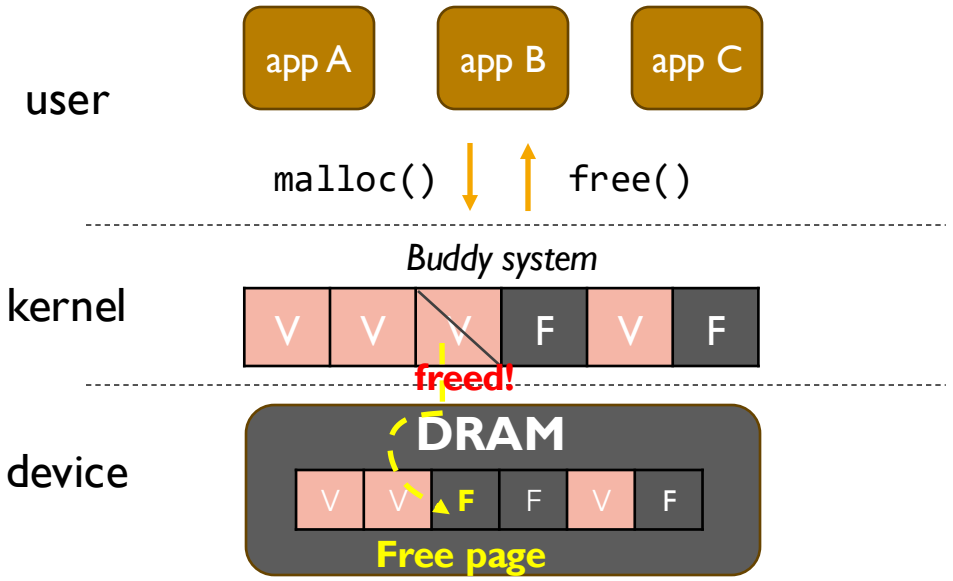
CXL-Flash

- **CXL Type 3 Device**
- **CXL.io / CXL.mem protocols**
- **Accessed in cache-line units (i.e., 64B) by host processor**



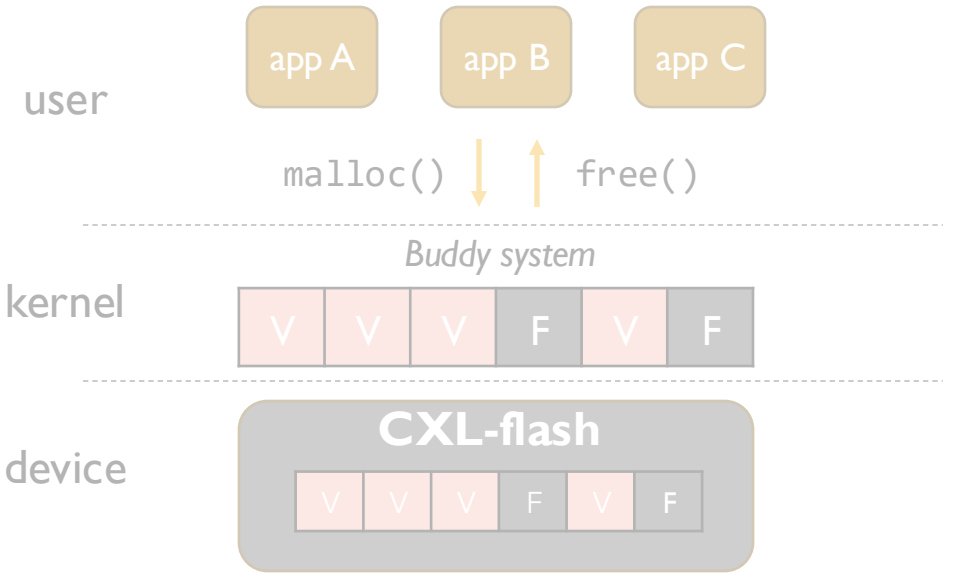
Challenges of CXL-Flash as Memory Module

When DRAM used

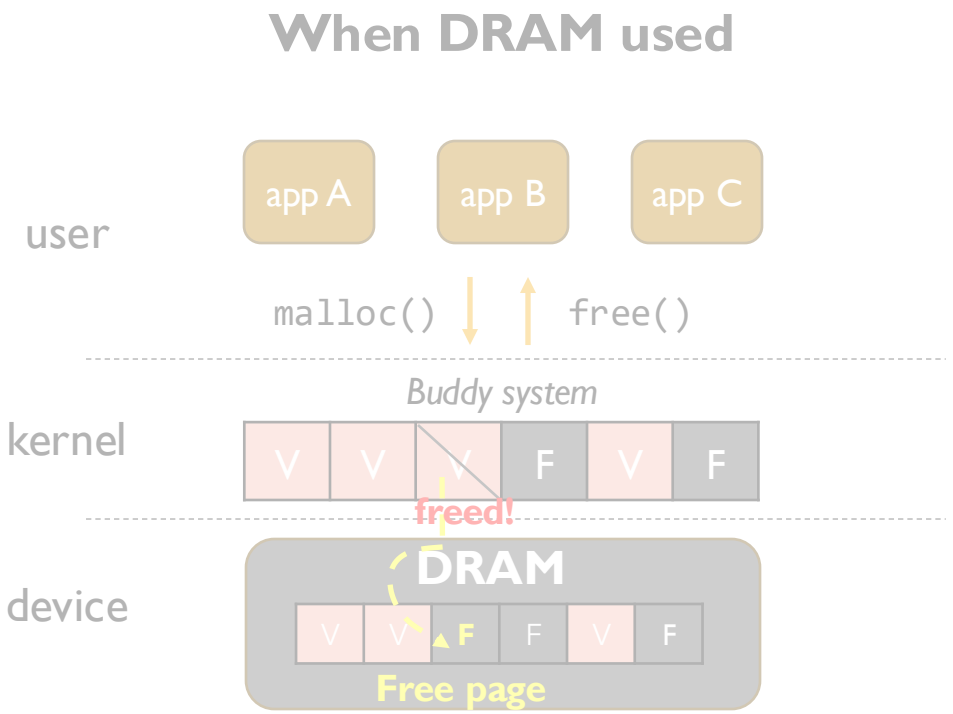


Zero cost for retaining invalid data

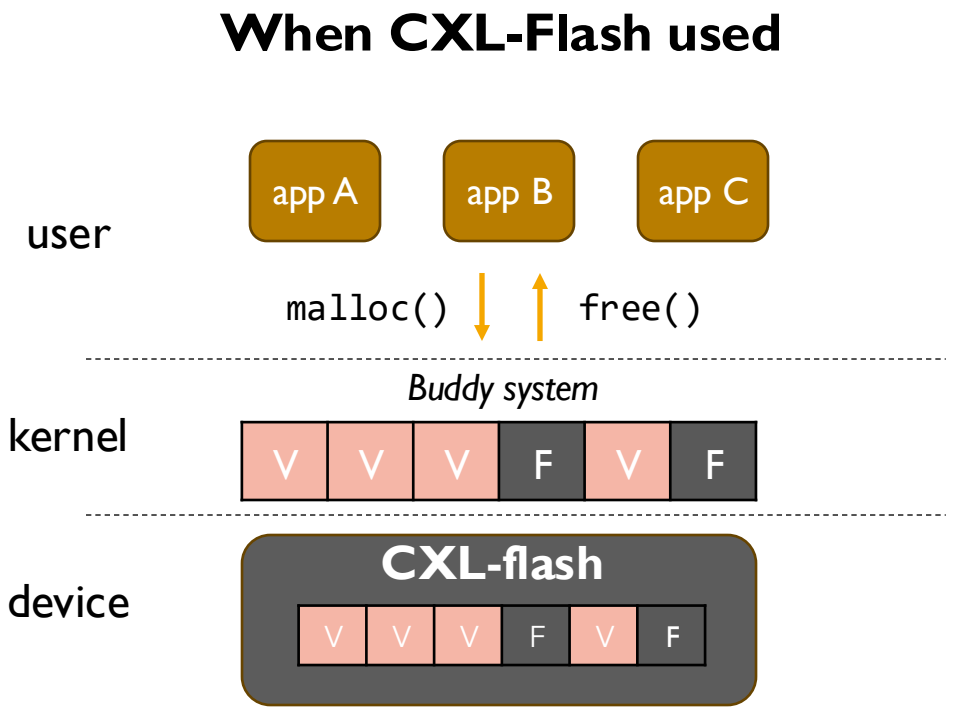
When CXL-Flash used



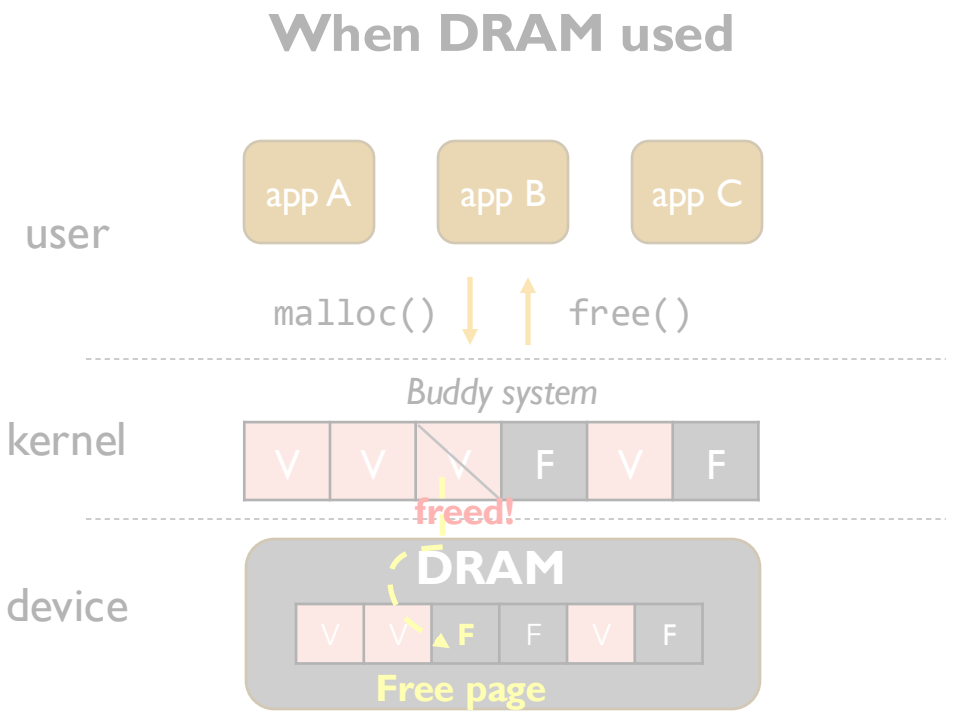
Challenges of CXL-Flash as Memory Module

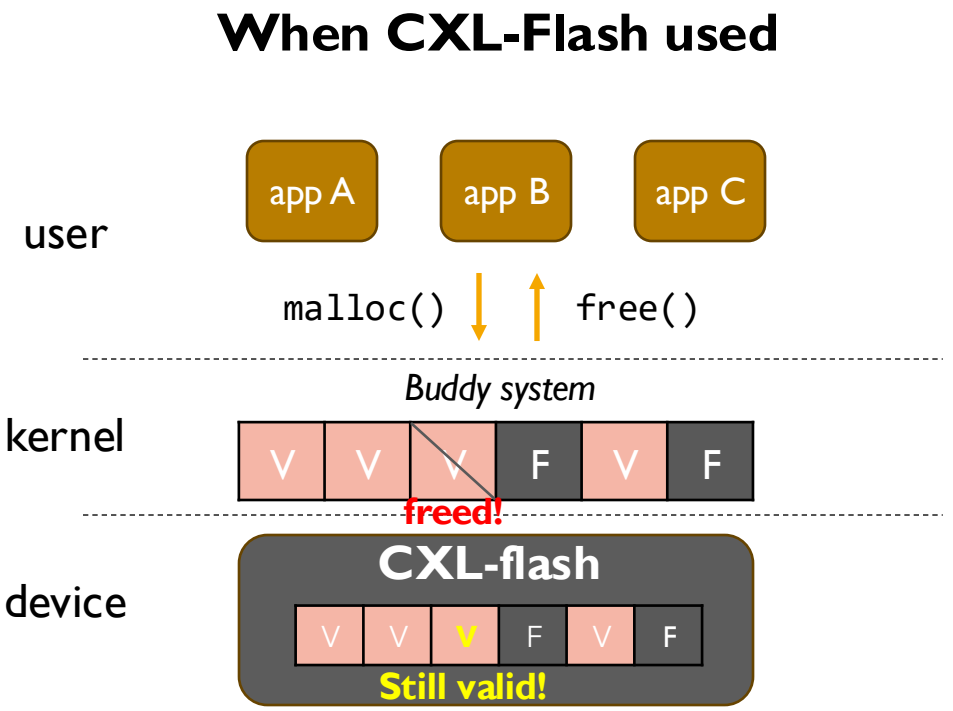


Zero cost for retaining invalid data



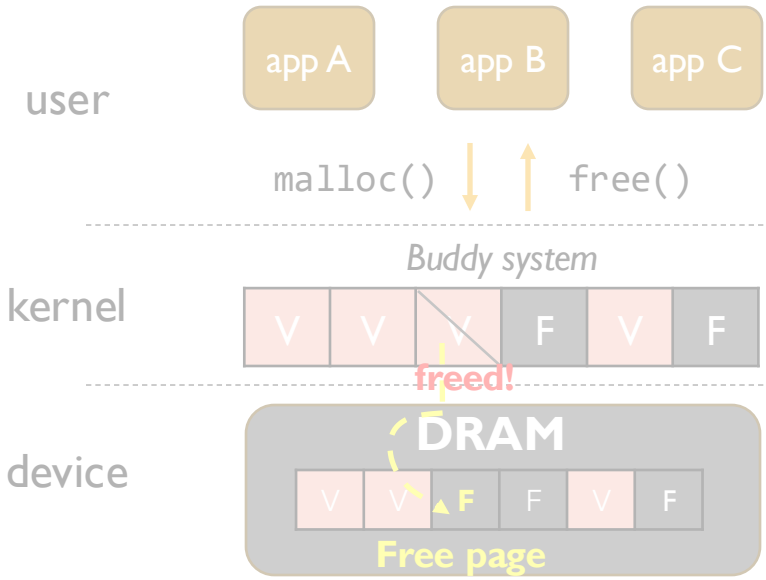
Challenges of CXL-Flash as Memory Module





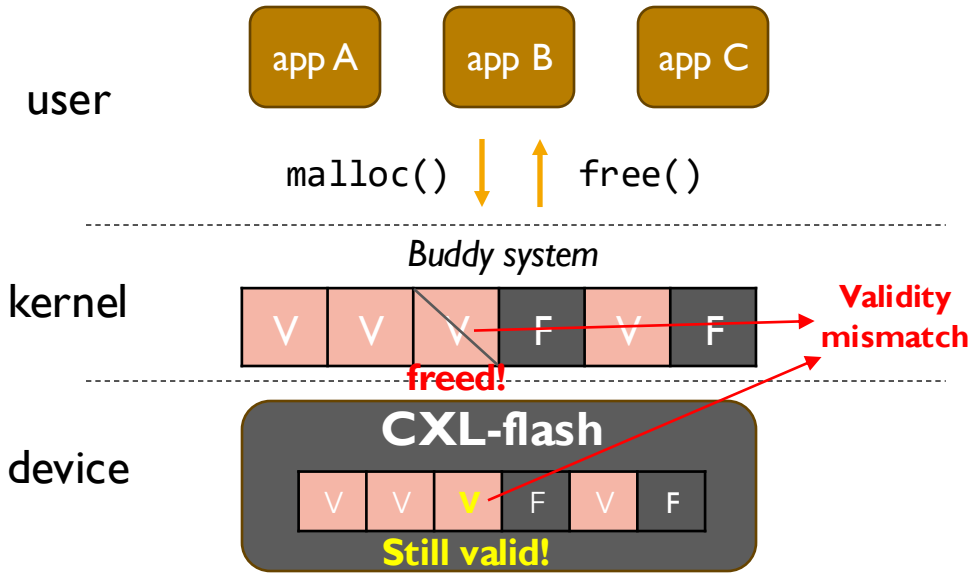
Challenges of CXL-Flash as Memory Module

When DRAM used



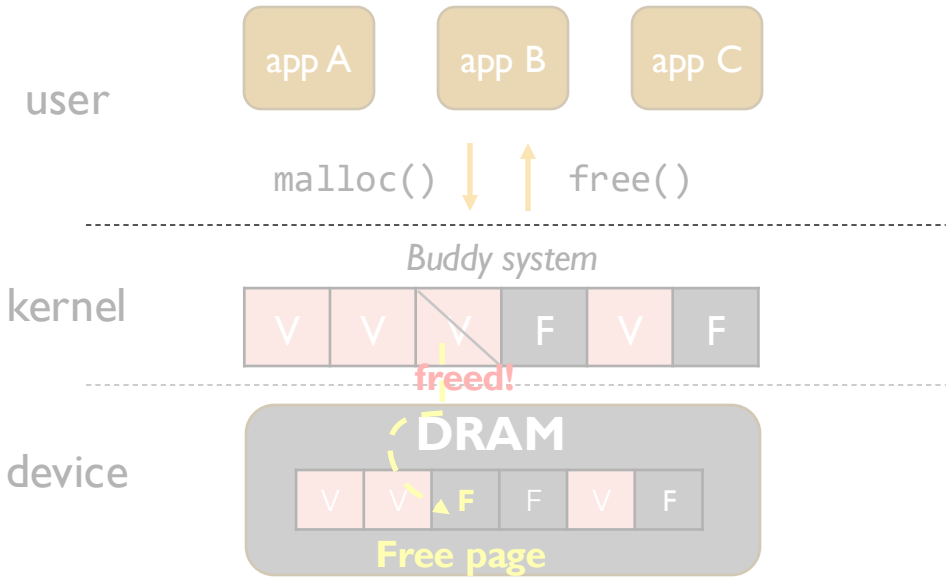
Zero cost for retaining invalid data

When CXL-Flash used



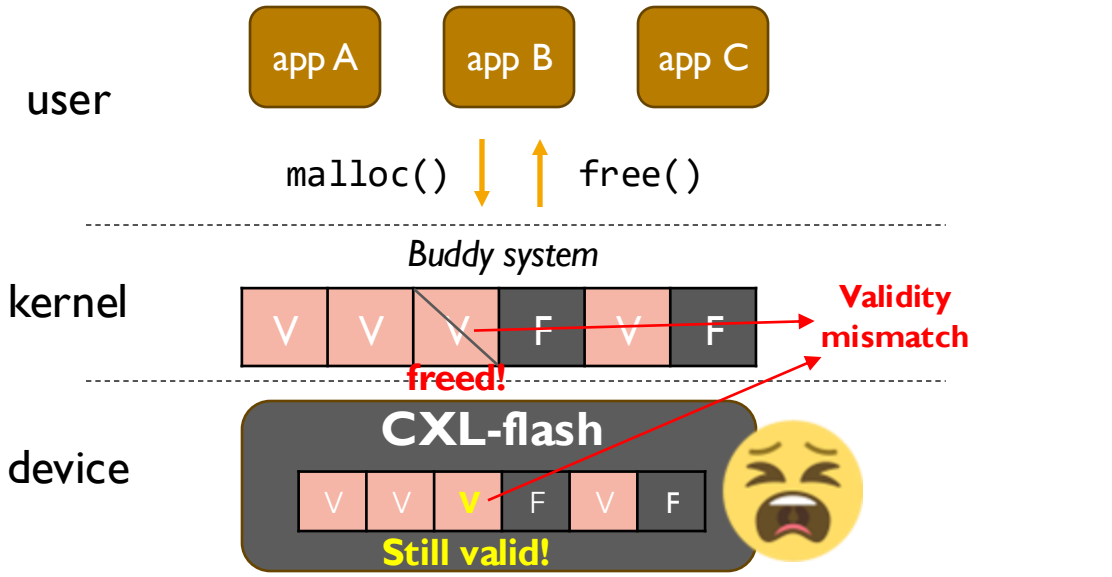
Challenges of CXL-Flash as Memory Module

When DRAM used



Zero cost for retaining invalid data

When CXL-Flash used



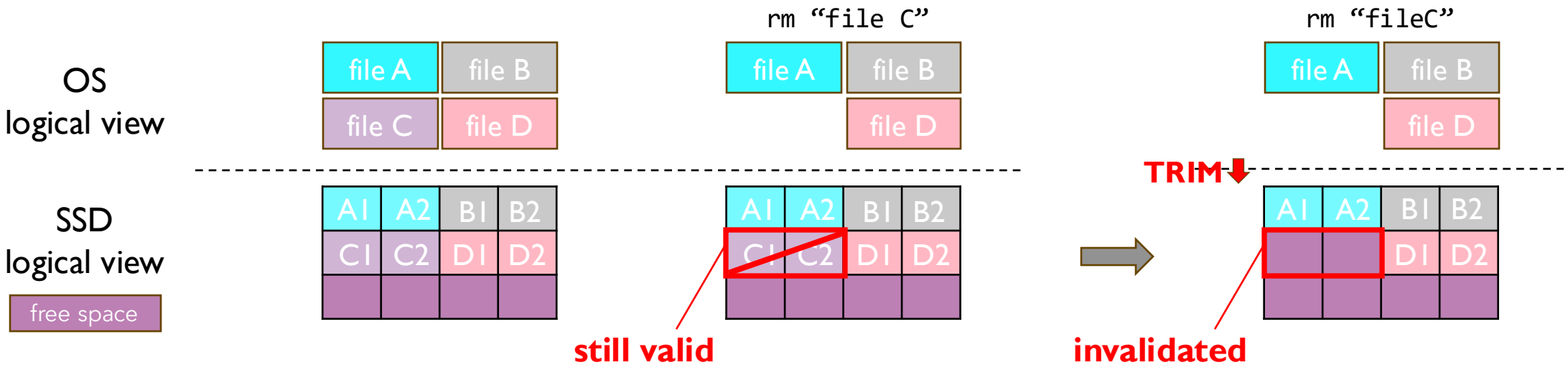
Will be copied during GC !

Need additional cost for invalid data!

- **Write amplification problem**
- **Fatal to lifetime-limited flash**

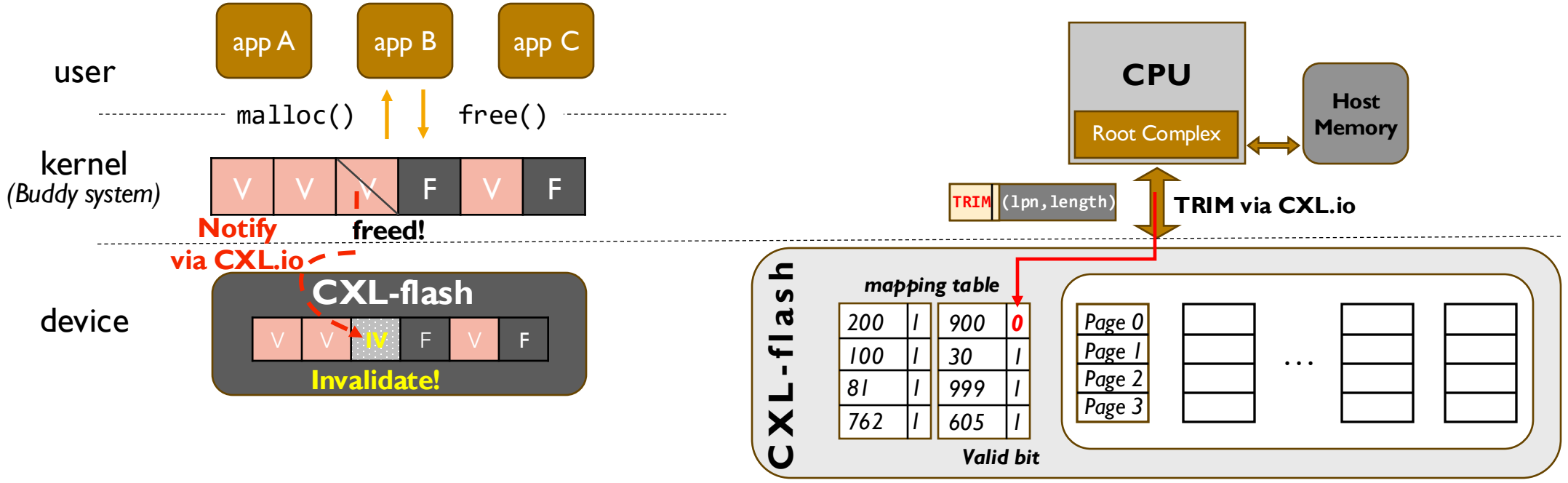
TRIM for Conventional SSD

- **TRIM command**
  - Handle mismatch between file data and actual data on SSD
  - Share validity information of data with the underlying device



Designing TRIM for CXL-Flash

- **TRIM triggered by buddy system upon reclamation**
  - Privileged operation
  - Asynchronously via CXL.io protocol



Difficulties in Assessing TRIM Impact

- **Simulation/emulation are not suitable for CXL-Flash**
  - Hard to capture traces during meaningful time window
    - Excessive memory access
    - # of Memory access >>> # of Storage access
  - Emulators suffer from limited functionality or slow execution
    - OpenCIS, Flight Simulator, CXLMemSim
    - NUMA emulation
- **Hard to conduct measurement study**
  - CXL-compliant devices have limited availability
  - Profiling TRIM-related operations is challenging

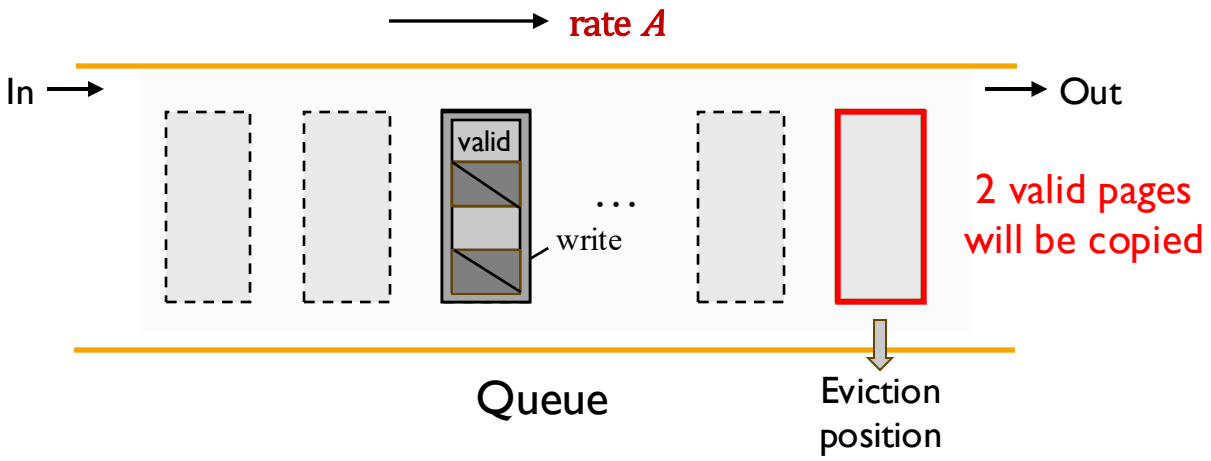
OpenCIS

MemVerge

CMM-H

Analytic Model for CXL-flash

- **Previous work: Analytic Modeling of SSD Write Performance (SYSTOR' 12)**
  - Modeling write amplification based on overprovisioning ratio with LRU cleaning policy
  - No considerations on TRIM command



총 발생한 쓰기량

$$A(WAF) = \frac{N_p}{N_p - E(v)}$$

실제 사용 자가 쓴 량

$$= \frac{N_p}{N_p - \left(1 - \frac{1}{UN_p}\right)^{\frac{TN_p}{A}} Np}$$

Cleaning 시점에 valid page 수의 기댓값

$$= \frac{1}{1 - (1 - \frac{1}{UN_p})^{\frac{TNp}{A}}}$$

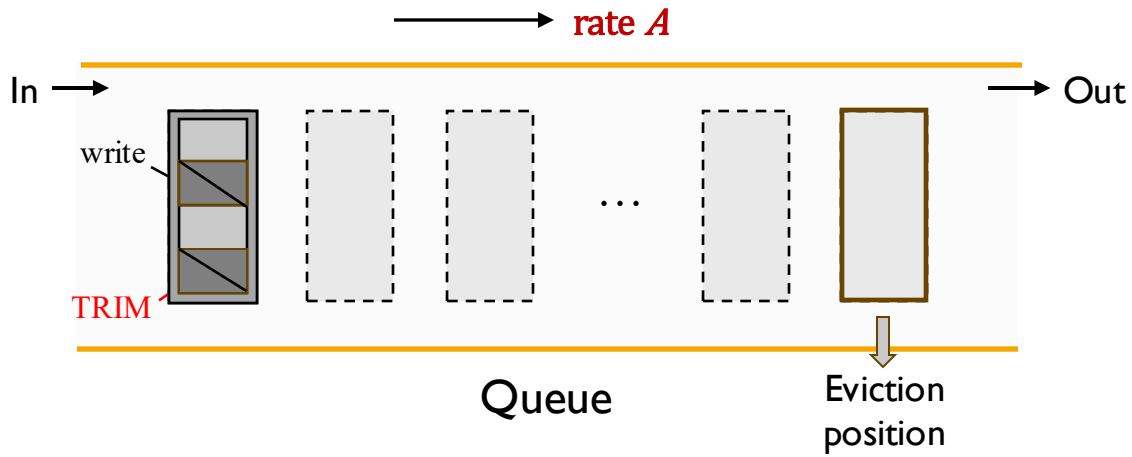
Lambert's W 함수를 이용해 풀면..

$$\mathrm{A}(WAF) = \frac{\boldsymbol{\alpha}}{\boldsymbol{\alpha} + \boldsymbol{W}(-\boldsymbol{\alpha e^{-\alpha}})}$$

(overprovisioning ratio α)

Our Analytic Model for CXL-flash

- **Extend analytical model to support TRIM**
  - New terminology D_r: ratio of Trim traffic to write traffic
  - Write amplification(A) prediction using # of valid pages in block



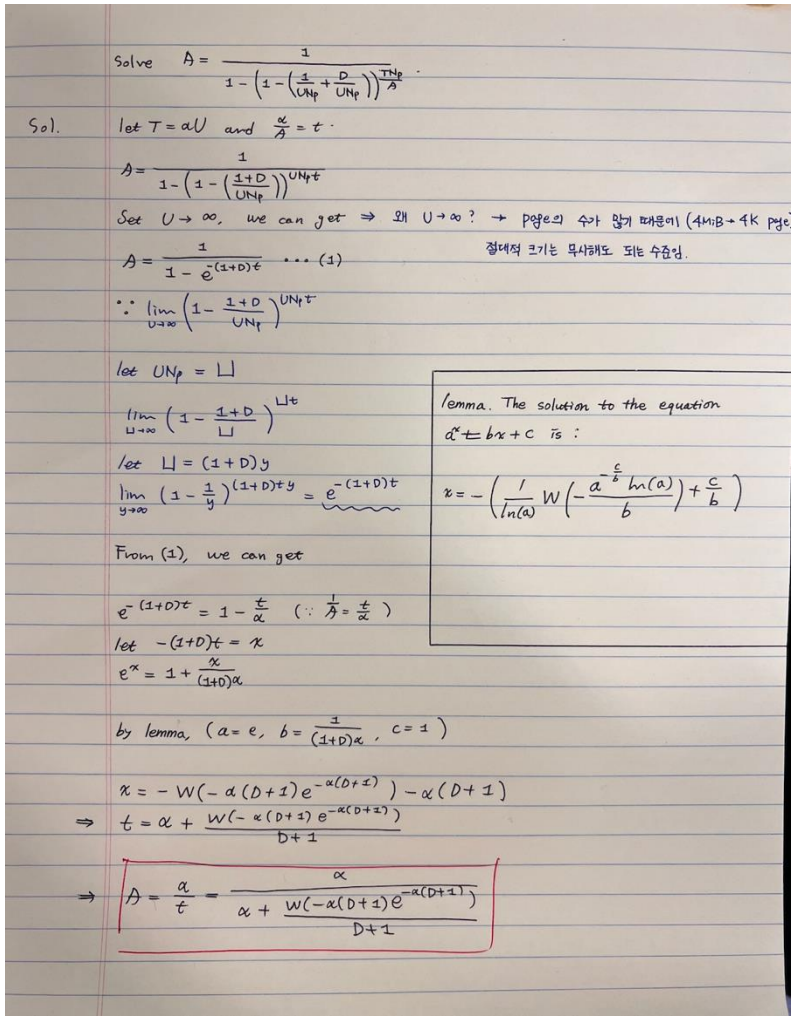
Our model:

$$A = \frac{1}{1-\left(1-\left(\frac{1}{UN_p}+\frac{D}{UN_p}\right)\right)^{\frac{TNp}{A}}}$$

TRIM 이 발생하면
invalidate 될 확률이 증가

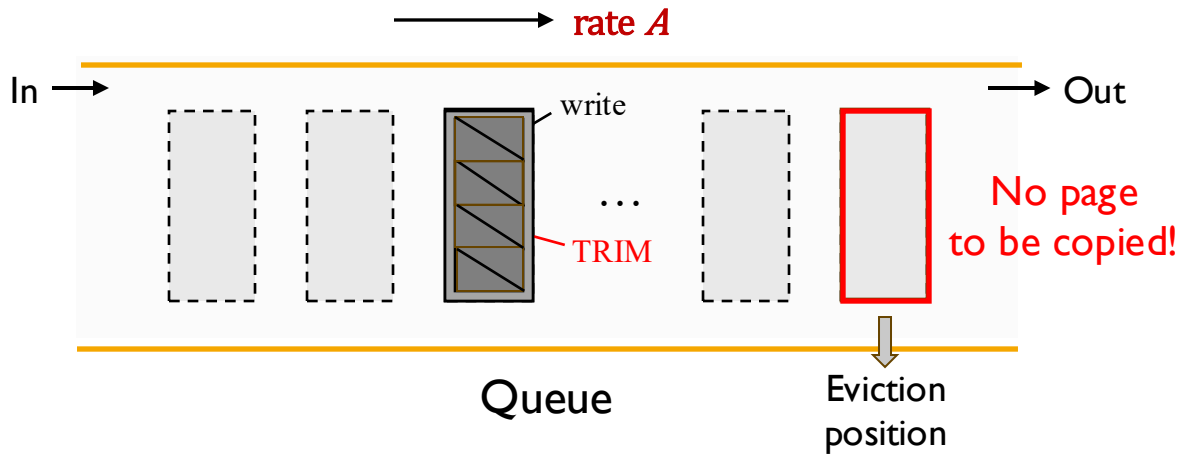
Our Analytic Model for CXL-flash

- 협업의 중요성...



Our Analytic Model for CXL-flash

- **Extend analytical model to support TRIM**
  - New terminology D_r: ratio of Trim traffic to write traffic
  - Write amplification(A) prediction using # of valid pages in block



Higher chance that pages in block get invalidated than the original model

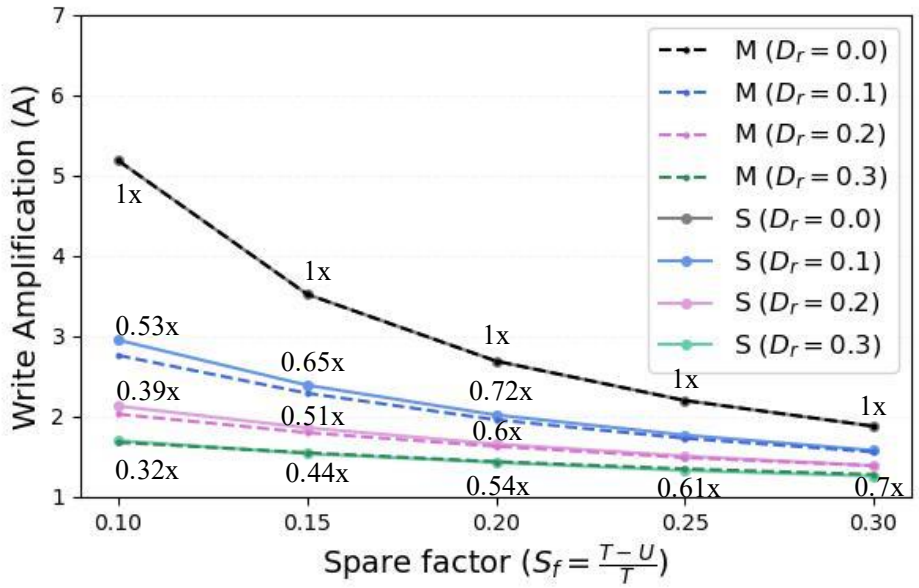
Our model:

$$A(WAF) = \frac{\alpha}{\alpha + \frac{W(-\alpha(1+D_r)e^{-\alpha(1+D_r)})}{1+D_r}}$$

(overprovisioning ratio α)

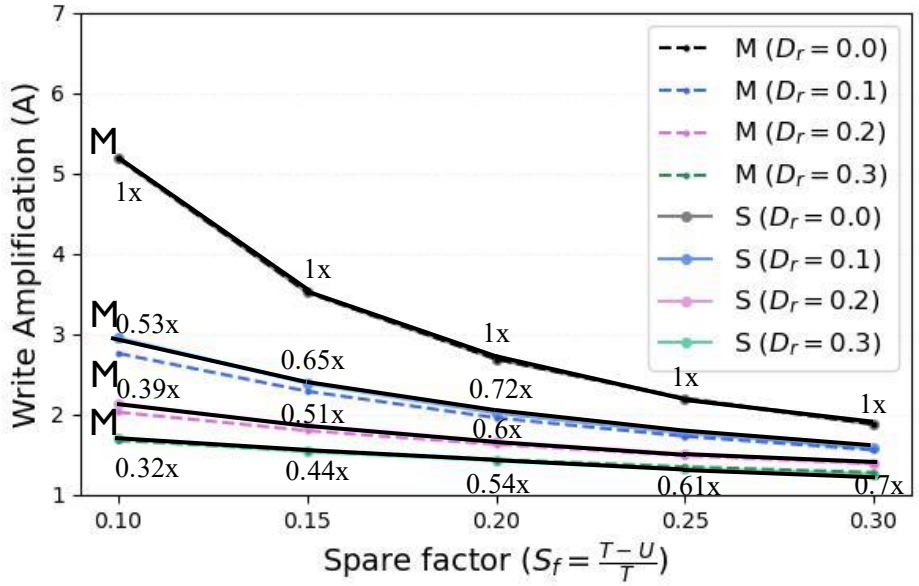
Validation of Analytical Model

- **Comparison between modeled(M) and simulated(S) WAF**
- **Methodology**
  - **Implement TRIM command in FTLSim** (SYSTOR' 12)
  - 10^6 logical blocks, 128 pages per block
  - Synthetic workloads
    - (1) Write entire logical space for warm-up
    - (2) Uniformly distributed writes (1/3 of total capacity)
      + TRIM issued every 10 writes (D_r)



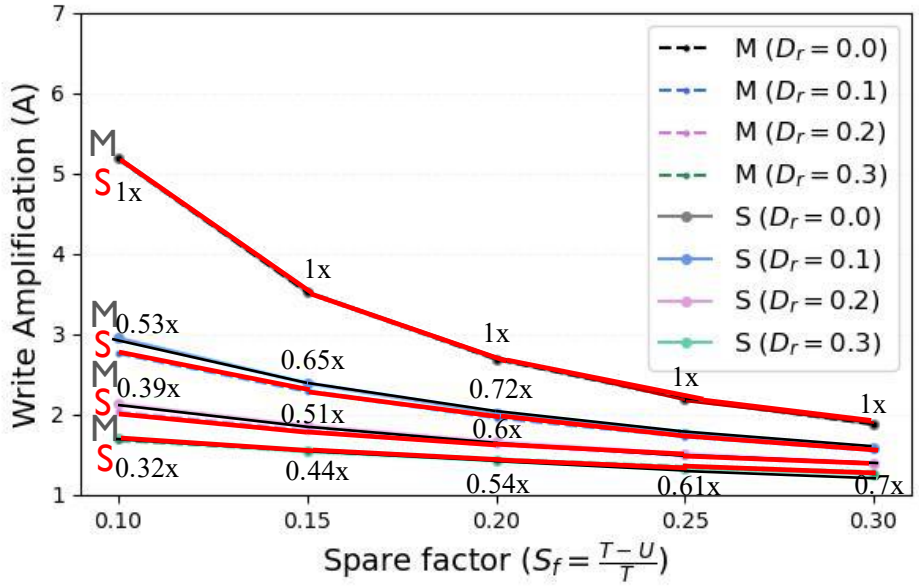
Validation of Analytical Model

- **Comparison between modeled(M) and simulated(S) WAF**
- **Methodology**
  - **Implement TRIM command in FTLSim** (SYSTOR' 12)
  - 10^6 logical blocks, 128 pages per block
  - Synthetic workloads
    - (1) Write entire logical space for warm-up
    - (2) Uniformly distributed writes (1/3 of total capacity)
      + TRIM issued every 10 writes (D_r)



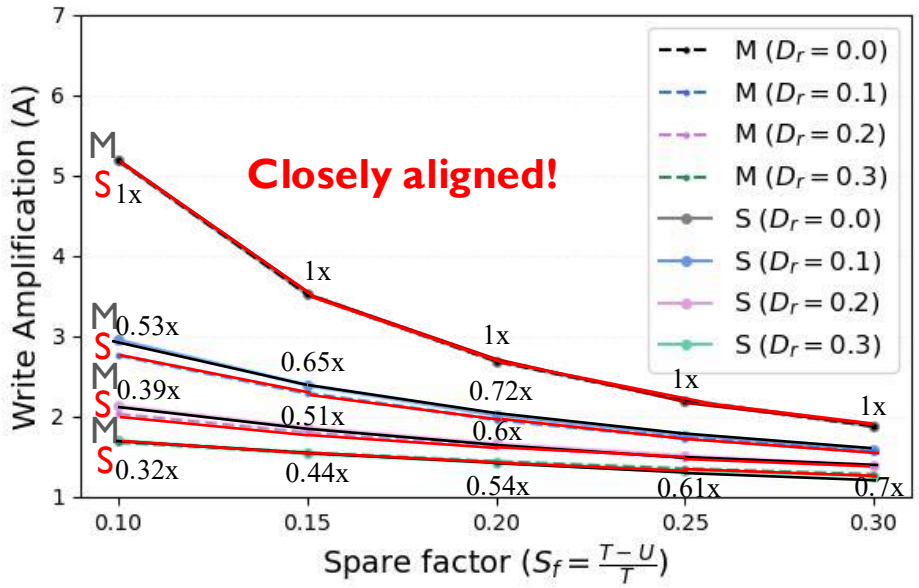
Validation of Analytical Model

- **Comparison between modeled(M) and simulated(S) WAF**
- **Methodology**
  - **Implement TRIM command in FTLSim** (SYSTOR' 12)
  - 10^6 logical blocks, 128 pages per block
  - Synthetic workloads
    - (1) Write entire logical space for warm-up
    - (2) Uniformly distributed writes (1/3 of total capacity)
      + TRIM issued every 10 writes (D_r)



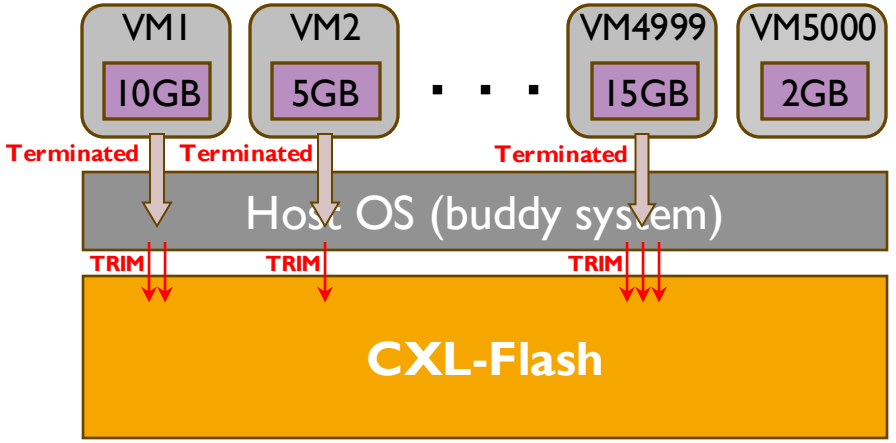
Validation of Analytical Model

- **Comparison between modeled(M) and simulated(S) WAF**
- **Methodology**
  - **Implement TRIM command in FTLSim** (SYSTOR' 12)
  - 10^6 logical blocks, 128 pages per block
  - Synthetic workloads
    - (1) Write entire logical space for warm-up
    - (2) Uniformly distributed writes (1/3 of total capacity)
      + TRIM issued every 10 writes (D_r)



Estimation of TRIM's Effects in Real World

- **Data centers running numerous VMs the need for CXL-Flash stands out most.**
- **Microsoft Azure VM Traces 5000 VMs sampled**
  - Each VM's Lifetime, Memory (GB)
  - YCSB (A-F) / DLRM (Train/Infer)
- **TRIM is issued upon VM termination**

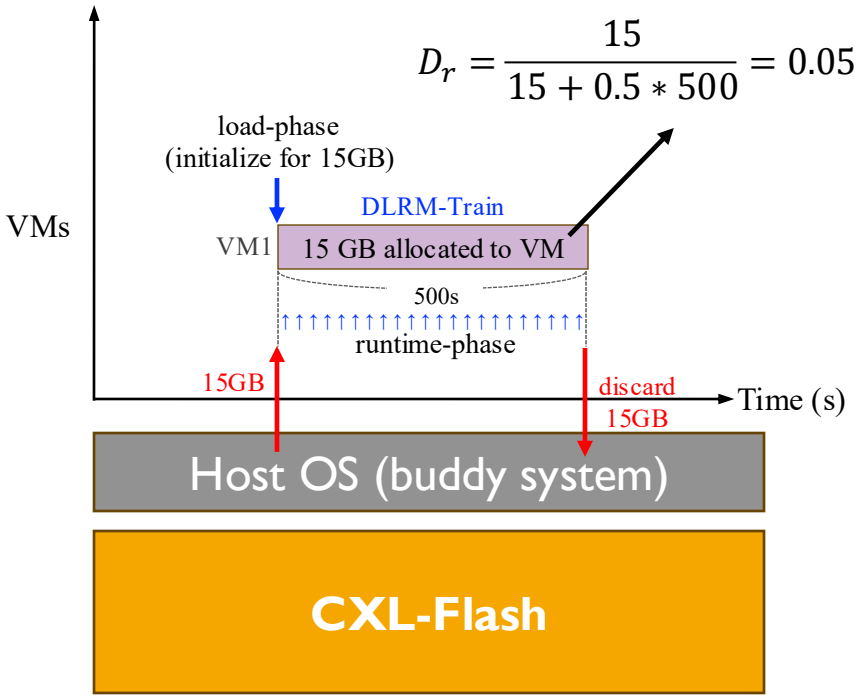


Estimation of TRIM's Effects in Real World (cont')

- **Our model:** $A = \frac{\alpha}{\alpha + \frac{W(-\alpha(1+D_r)e^{-\alpha(1+D_r)})}{1+D_r}}$
  - Defined as $D_r = \frac{discarded\ size}{load\ phase\ traffic + runtime\ phase\ traffic}$
  - $discarded\ size$ assumed to be equal to $load\ phase\ traffic$
    - VM's Memory(GB)
  - Profiling `LLC_misses.mem_write` with linux perf tool
    - No data of each VM's memory write traffic
  - Random mapping write traffic to VMs
    - YCSB + DLRM mixed at different ratio 6:4 / 3:7

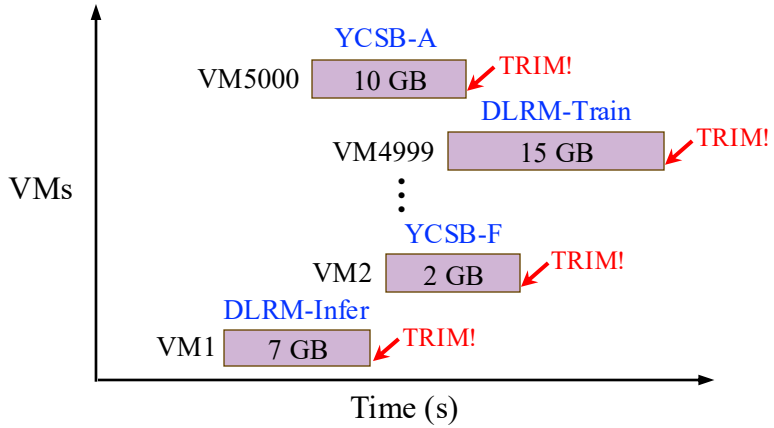
WK	A	B	C	D	E	F	TR	IF
R	262	296	226	524	79	190	1200	380
W	147	154	94	388	40	101	500	267

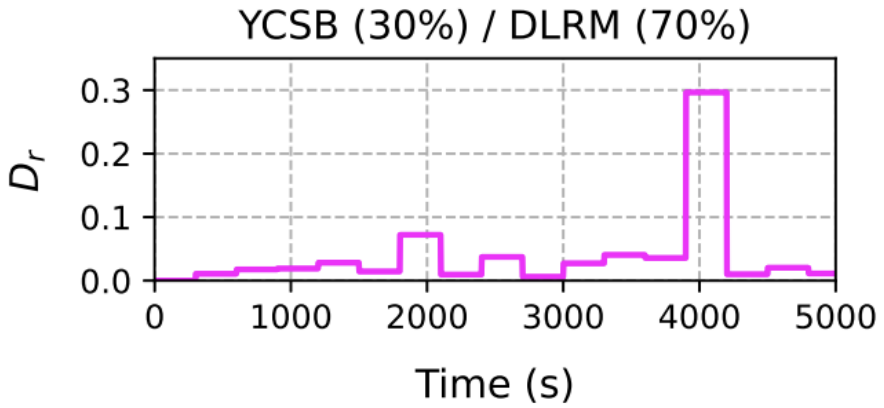
Memory access traffic (MB/s)

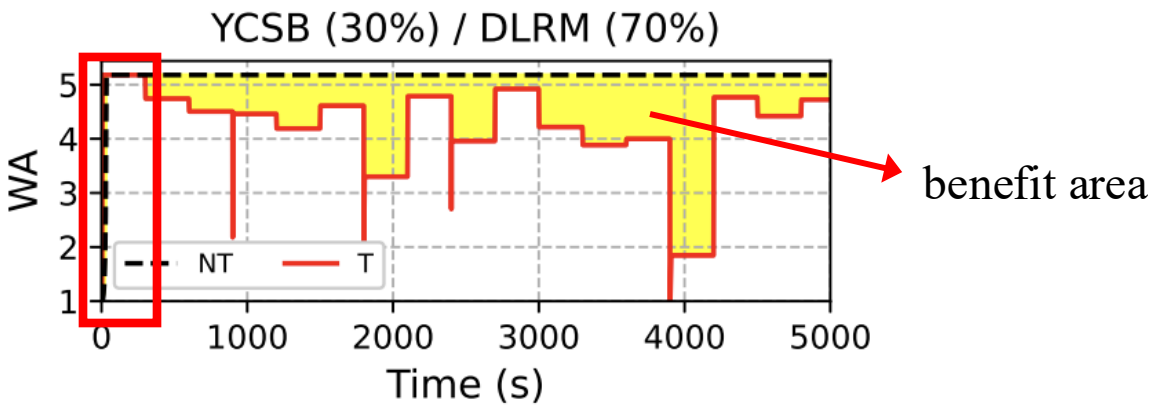


Estimation of TRIM's Effects in Real World (cont')

- **TRIM effect on CXL-Flash running VMs over time**
  - NT(No Trim): reaching a state fully occupied by fake valid data
  - T(Trim): reduction in WAF

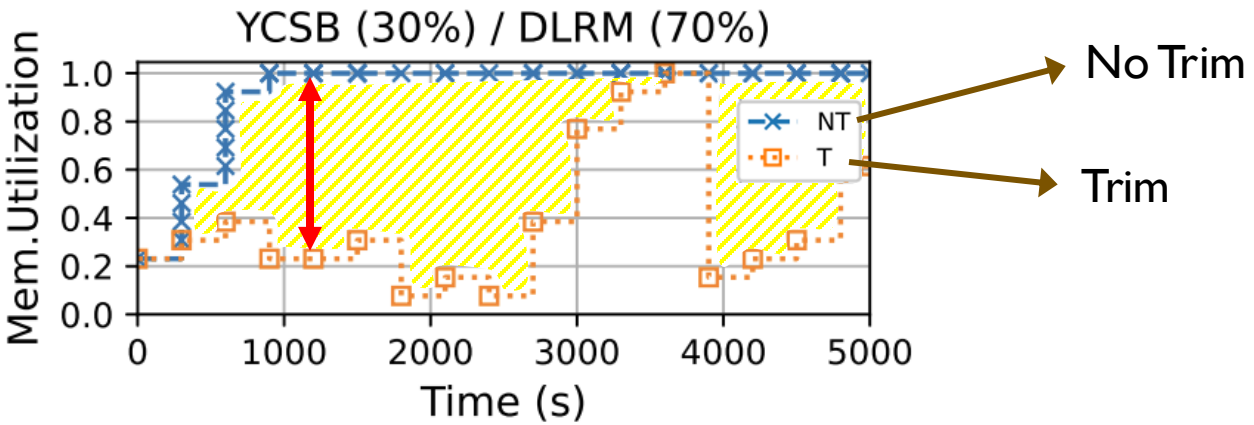






Estimation of TRIM's Effects in Real World (cont')

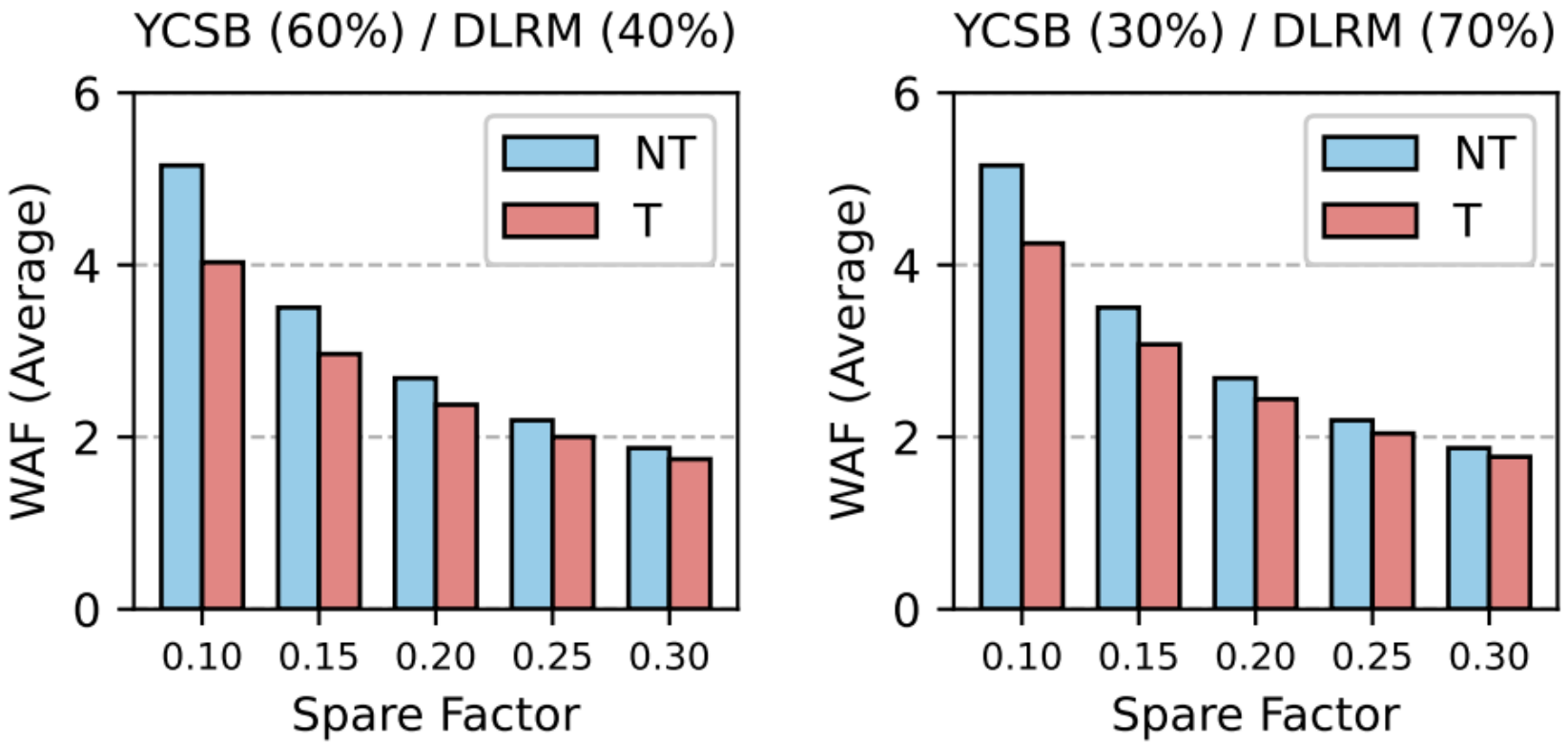
- **Memory Utilization : TRIM (T) vs. No TRIM (NT)**
  - Size of CXL-Flash (the max-memory simultaneously used): 22.75 GB
  - Spare factor of CXL-Flash is set to 0.1
- **CXL-Flash perceives available spaces are fully utilized beyond a certain point**



Memory Utilization over time

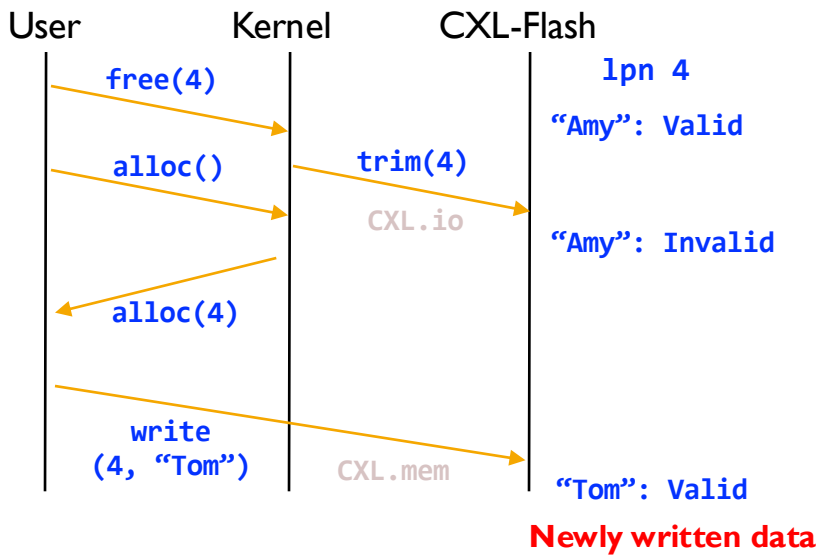
Estimation of TRIM's Effects in Real World (cont')

- **Average WAF in CXL-Flash running VMs**
  - WAF reduced by 11.56% on average, reaching a maximum of 19.69%

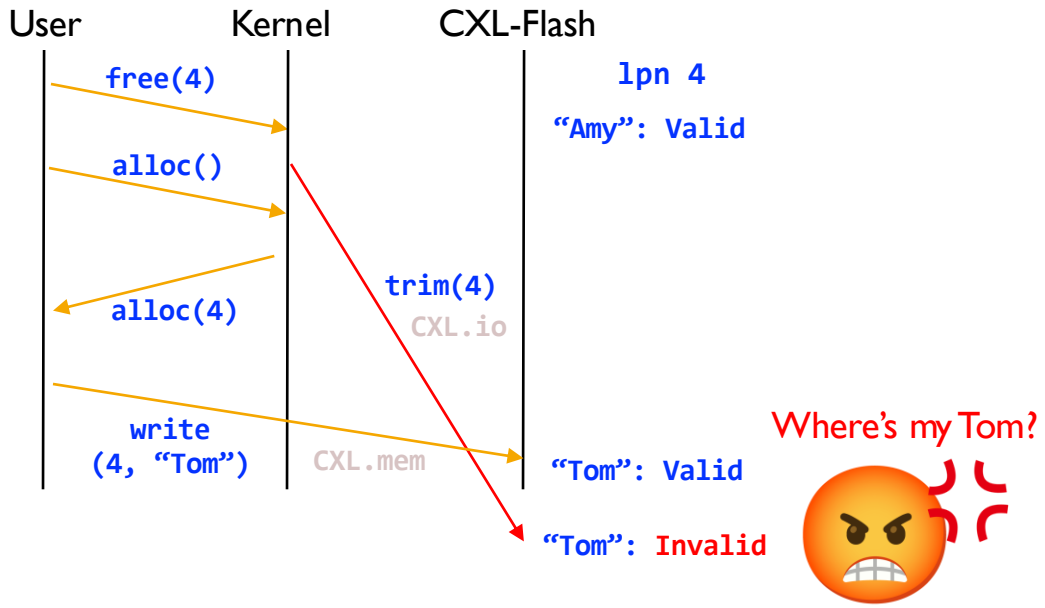


Discussion

- **Proper coordination is needed to prevent data loss**
  - Kernel must defer reallocating the TRIM-pending area until ACK is received
  - Device must notify the kernel upon TRIM completion



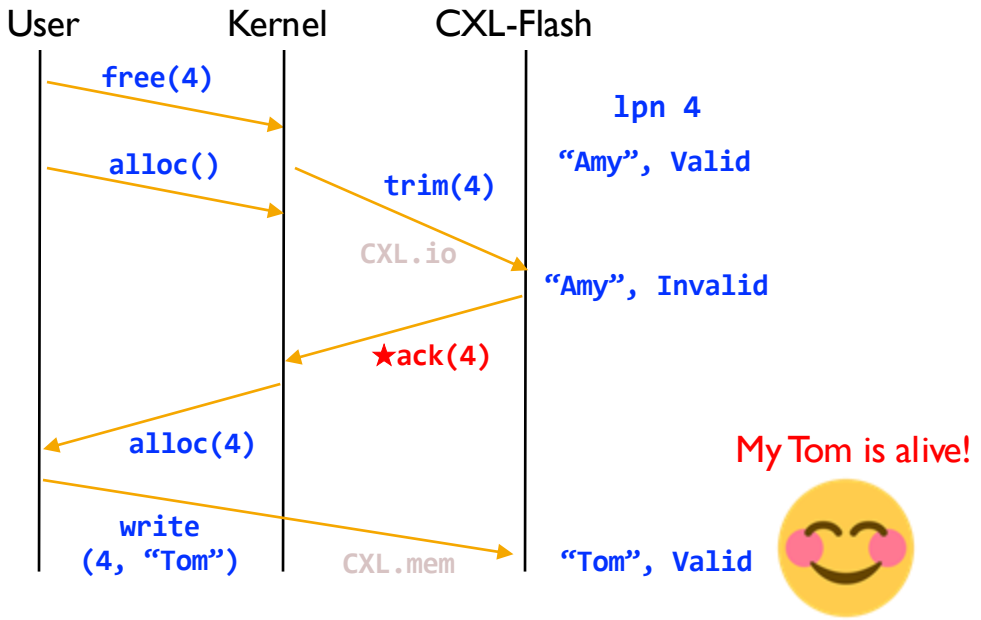
In-order execution



Out-of-order execution

Discussion

- **Proper coordination is needed to prevent data loss**
  - Kernel must defer reallocating the TRIM-pending area until ACK is received
  - Device must notify the kernel upon TRIM completion



Coordinated out-of-order executio n

Conclusion

- **Summary**
  - TRIM via CXL.io to invalidate fake valid data in CXL-Flash
  - Near-exact extended analytical model for TRIM effect evaluation
  - Demonstration of the necessity of TRIM on real-world scenarios using our model
  - TRIM-like mechanism is necessary for CXL-Flash!
    - Synchronization issues should be considered

- **Future work**
  - Identification of optimal host-side TRIM initiator and analysis of TRIM overhead
  - Analysis of long-term impact of TRIM on memory performance and endurance in CXL-Flash
  - Estimation of TRIM effectiveness via Workload- and GC policy-aware analytical model in real-world scenario