DISCOS
Data-Intensive AI Computing & Systems Laboratory

IHS

Optimizing NVMe-Based Key-Value Interfaces and Leveraging KVSSDs: A Comprehensive Study

Youngjae Kim, Ph.D.

Special Thanks to Junhyeok, Ki-Hwan, Hyunsun, and Seonghoon

Soongsil University Special Seminar
April 16, 2025

숭실대학교
Soongsil University

Contents

- Optimizing NVMe-Based Key-Value Interfaces
  - BandSlim: A Novel Bandwidth and Space-Efficient KV-SSD with an Escape-from-Block Approach, ICPP, 2024
  - ByteExpress: A High-Performance and Traffic-Efficient Inline Transfer of Small Payloads over NVMe, (under review)

- Leveraging KVSSDs to Overcome Performance Hurdles in Key-Value Stores
  - KVAccel: A Novel Write Accelerator for LSM-Tree-Based KV Stores with Host-SSD Collaboration, IPDPS, 2025 (To Appear)

DISCOS
Data-Intensive Computing & Systems Laboratory

IHS

Optimizing NVMe-Based Key-Value Interfaces

- BandSlim: A Novel Bandwidth and Space-Efficient KV-SSD with an Escape-from-Block Approach, ICPP, 2024
- ByteExpress: A High-Performance and Traffice-Efficient Inline Transfer of Small Payloads over NVMe, (under review)

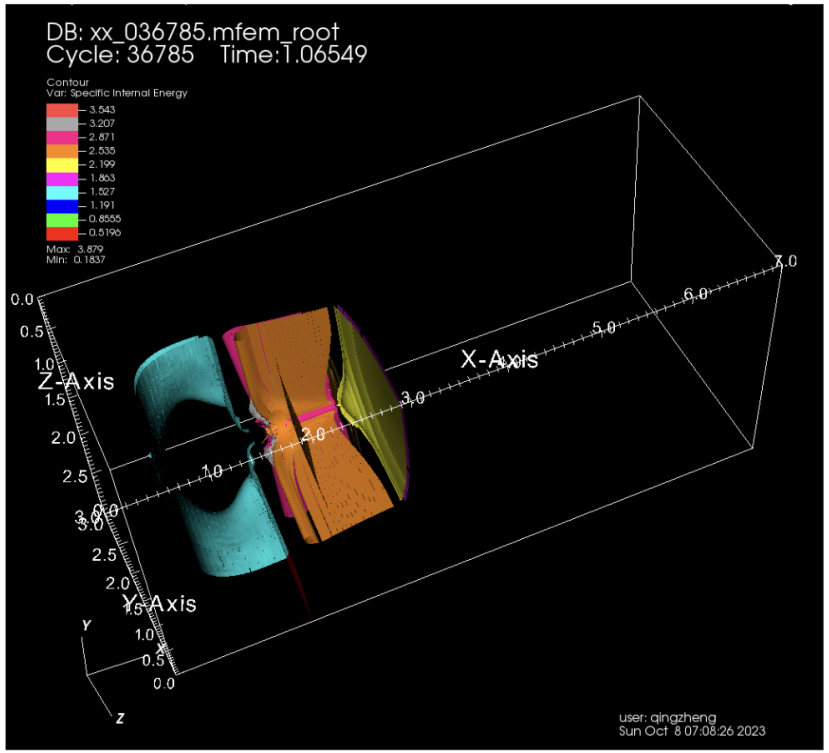
Background

Big Data Era

- A rapid adoption of Artificial Intelligence (AI), High-Performance Com--puting (HPC), Data Analytics, and Cloud Service in these days.
  - They handle **“Big Data”**.

ChatGPT

Microsoft Azure



What does Data look like?

- These Big Data applications do not merely handle Blocks; they manage variable-sized **Key-Value Pairs** or **Objects**.

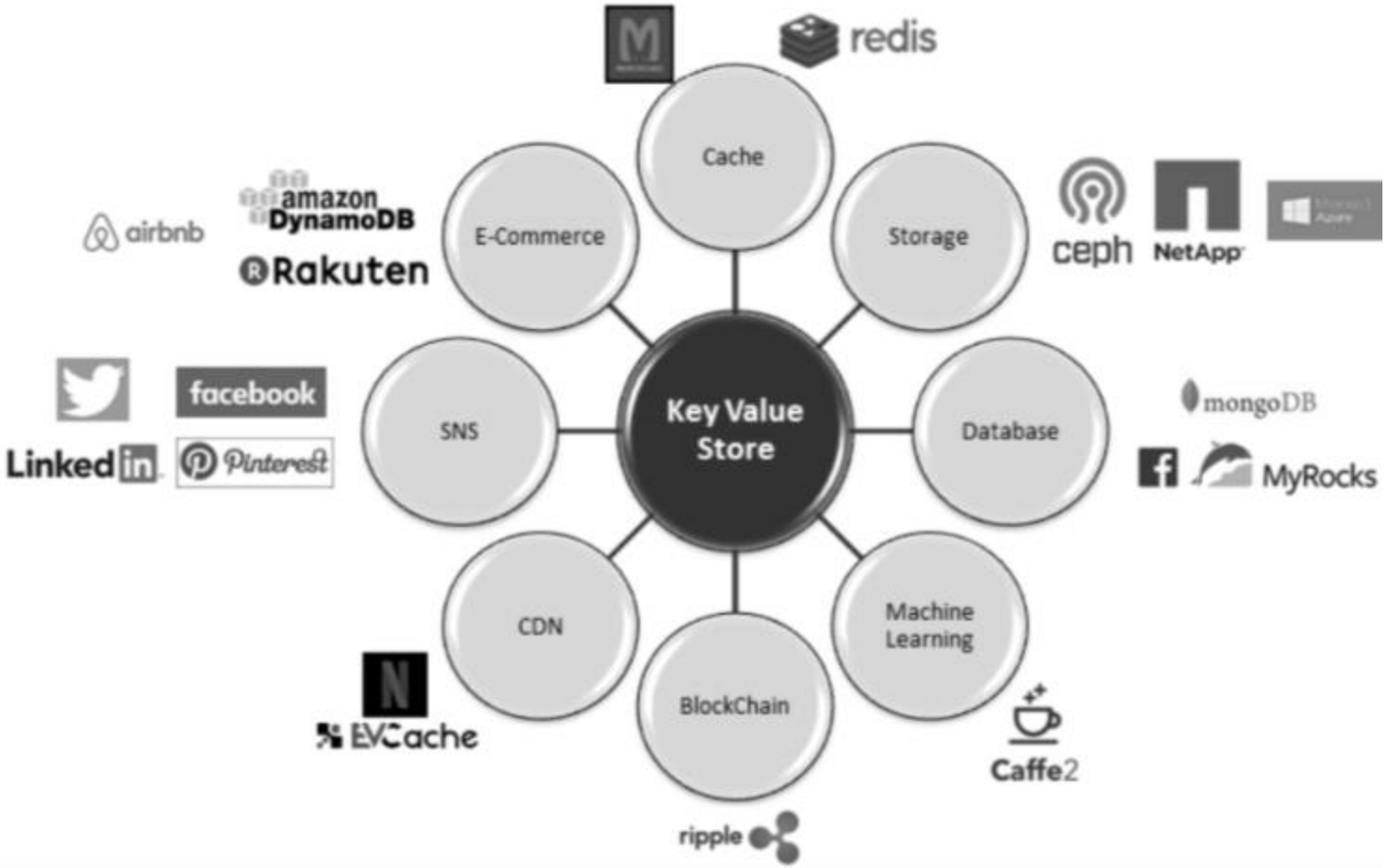
Fixed-sized

Variable-sized

* Picture from “Key Value SSD Explained – Concept, Device, System, and Standard” presented at SDC 2017 by S.-K.Yang,

Key-Value Store

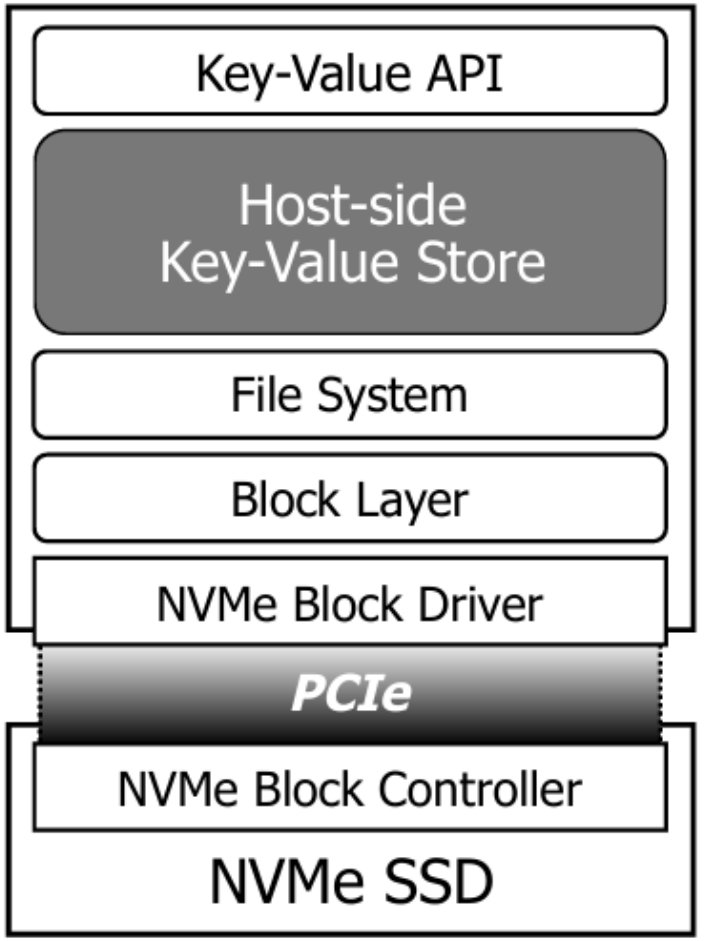
- Therefore, these Big Data applications typically operate by employing Key-Value Stores (e.g., RocksDB, Cassandra).



* Picture from "Key Value SSD Explained – Concept, Device, System, and Standard" presented at SDC 2017 by S.-K.Yang,

Software Stack Issue

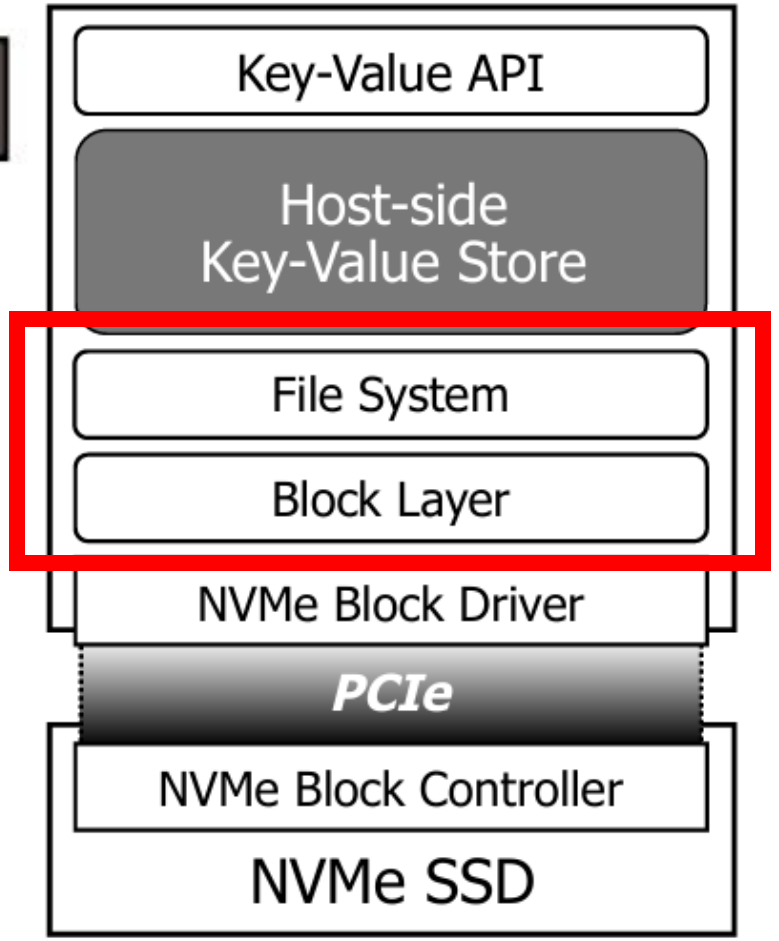
- Key-Value Stores run on top of file system & block layer, device driver and device controller.



Software Stack Issue

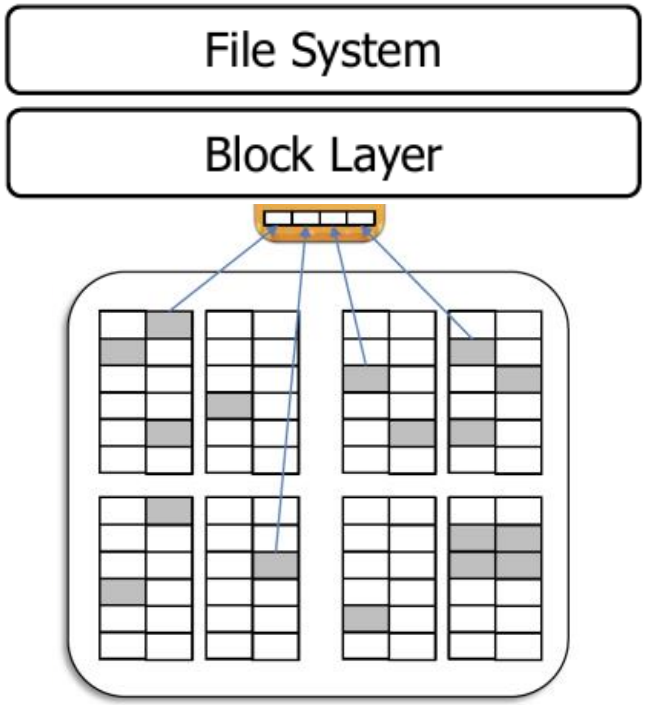
- Key-Value Stores run on top of file system & block layer, device driver and device controller.

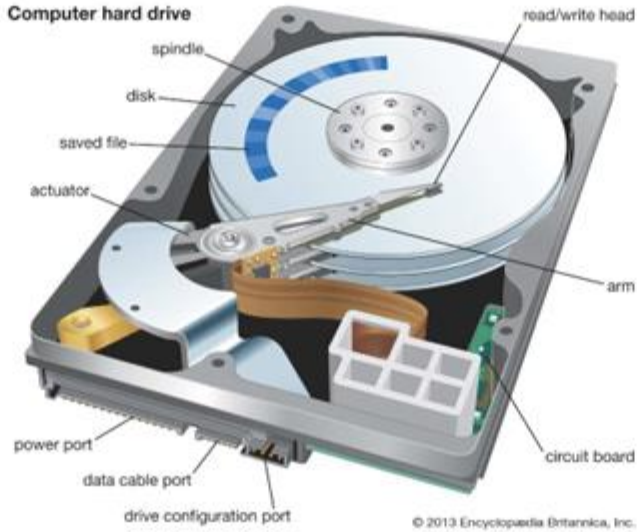
Do we really need these layers?



Software Stack Issue

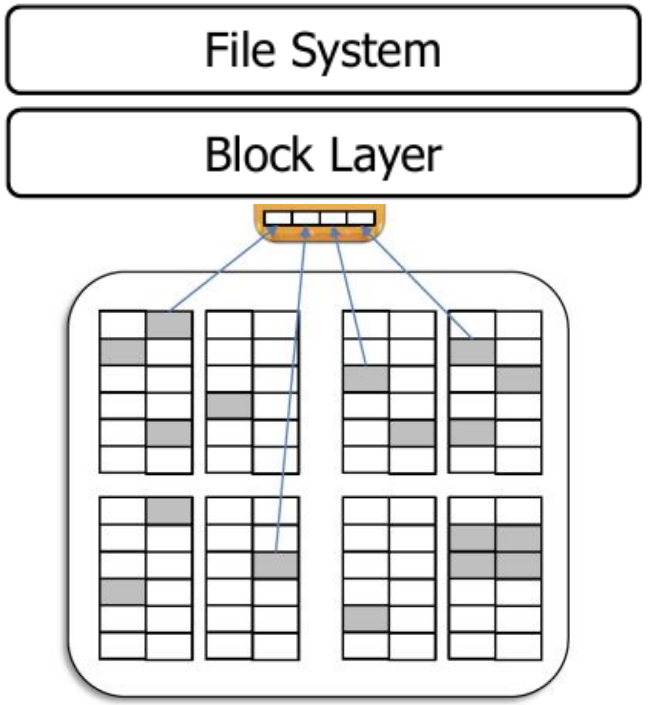
- These layers are in place to follow the **block interface**, which originated from the hard disk drives.





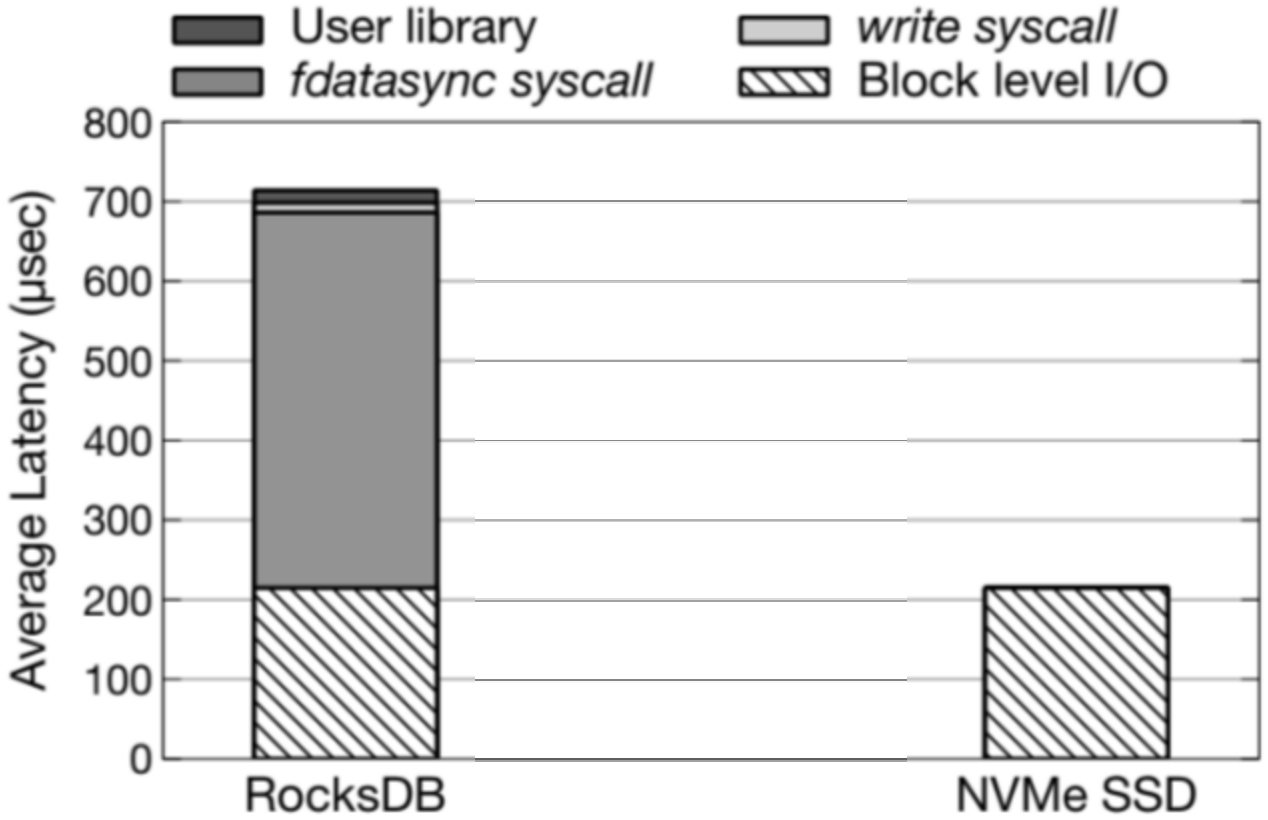
Software Stack Issue

- These layers are in place to follow the **block interface**, which originated from the hard disk drives.



Software Stack Issue

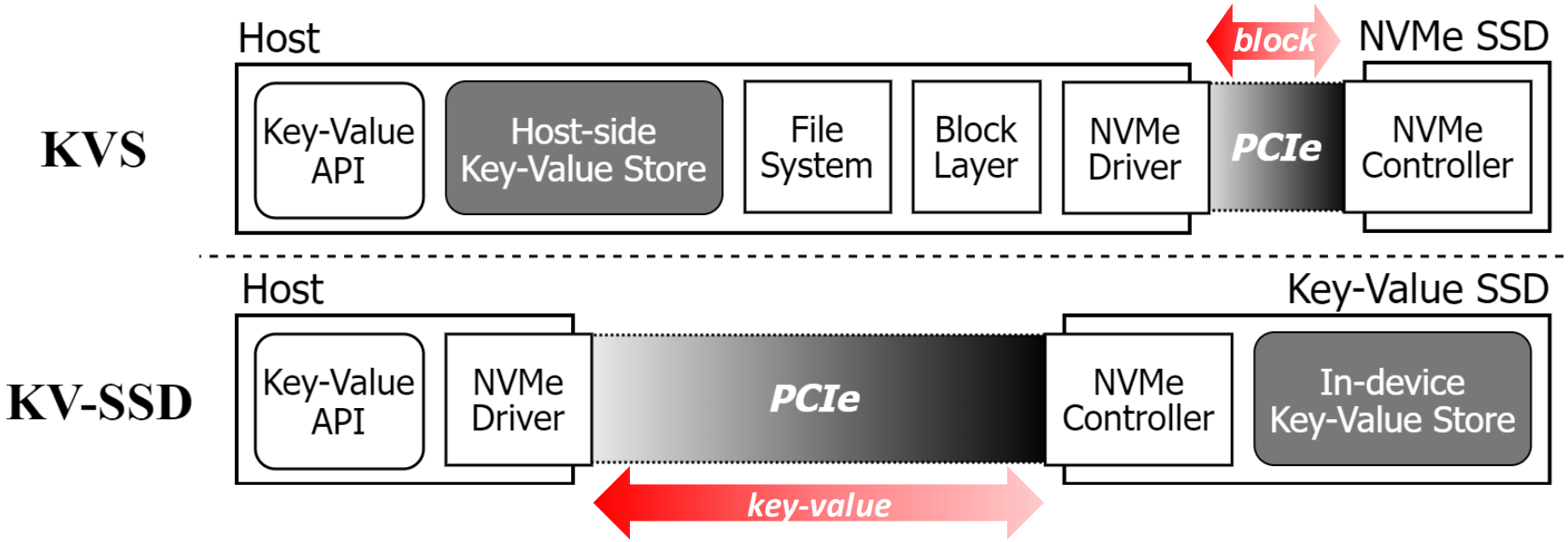
- The problem is that these layers account for a **significant portion** of the total response time in Key-Value Stores [1].



[1] Lee, C. G., Kang, H., Park, D., Park, S., Kim, Y., Noh, J., Chung, W., & Park, K. (2019). iLSM-SSD: An Intelligent LSM-Tree Based Key-Value SSD for Data Analytics. In Proceedings of the International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS).

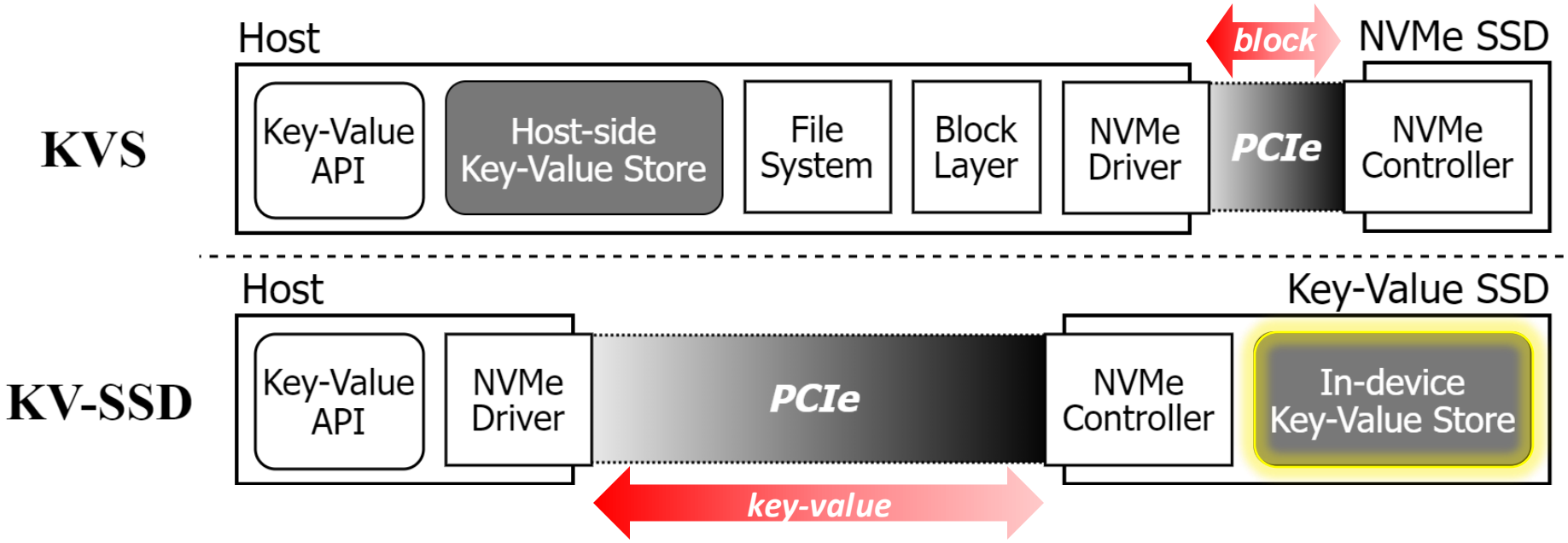
Key-Value Solid State Drive (KV-SSD)

- What about streamlining these layers from the storage stack?
  - By making a key-value pair as the unit of data communication interface
- KV-SSDs have renovated the storage interface by changing the unit of I/O transactions from the traditional block to key-value.



Key-Value Solid State Drive (KV-SSD)

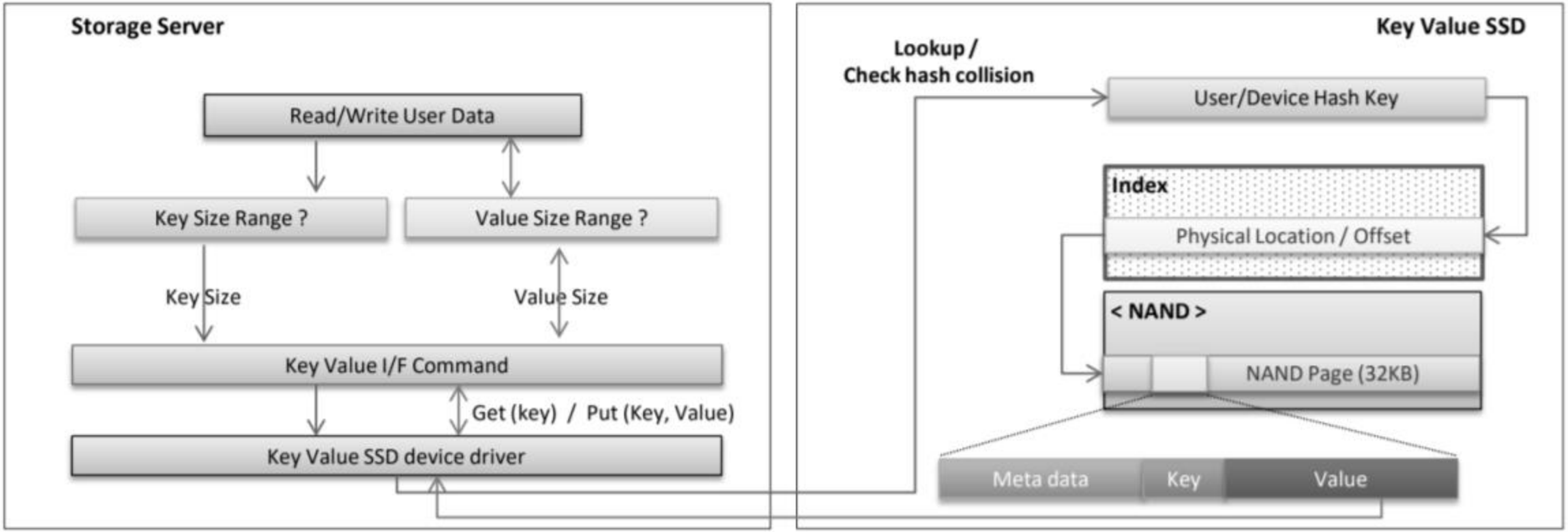
- What about streamlining these layers from the storage stack?
  - By making a key-value pair as the unit of data communication interface
- KV-SSDs have renovated the storage interface by changing the unit of I/O transactions from the traditional block to key-value.



➔ **lower latency & higher throughput**

Key-Value Solid State Drive (KV-SSD)

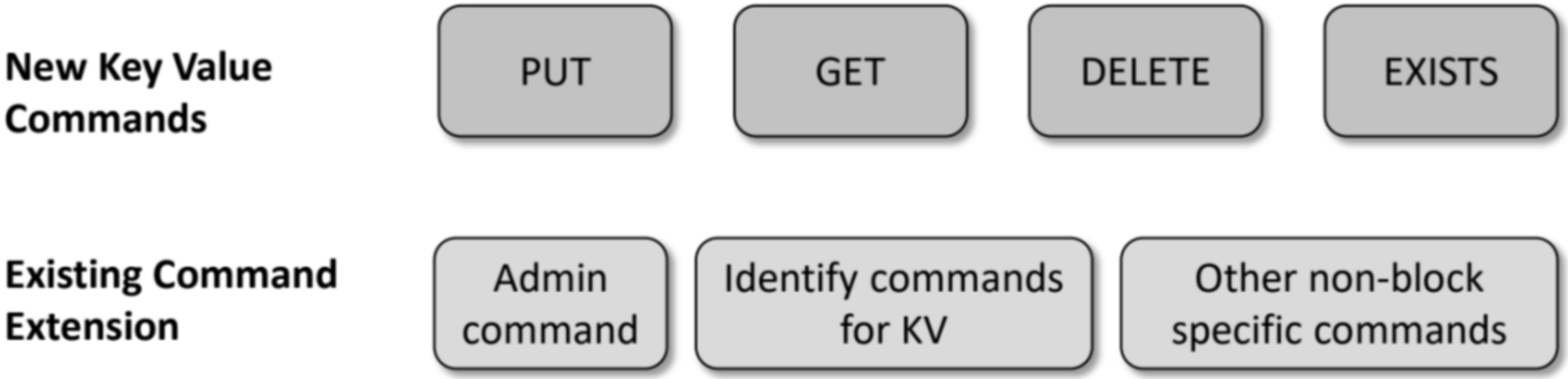
- KV-SSD supports key-value store operations like PUT and GET.
- KV-SSD maintains Key-to-Page mapping info by deploying index structures like Hash Table or LSM-tree.



* Picture from “Key Value SSD Explained – Concept, Device, System, and Standard” presented at SDC 2017 by S.-K.Yang,

NVMe Key-Value Command Set

- The NVMe protocol has introduced a key-value command set.



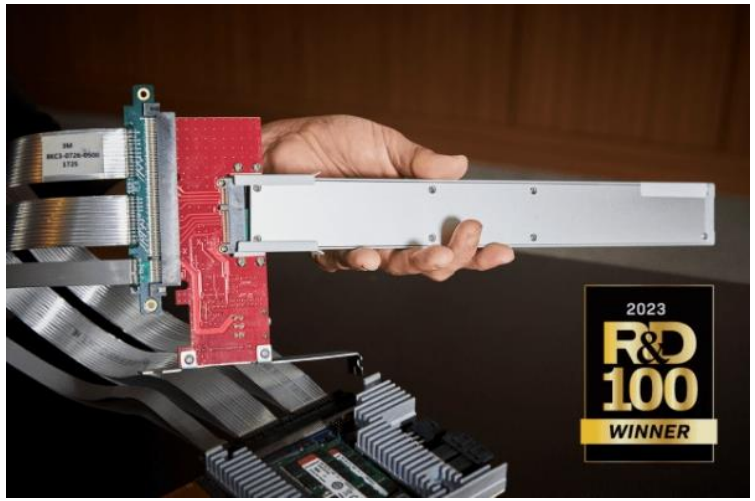
* Picture from “Key Value SSD Explained – Concept, Device, System, and Standard” presented at SDC 2017 by S.-K.Yang,

NVMe Key-Value Command Set

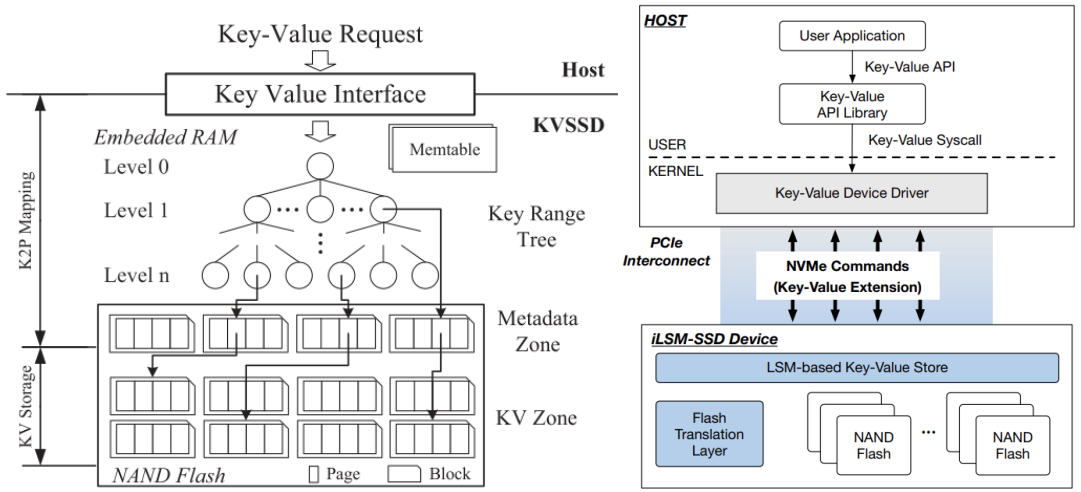
- The NVMe protocol has introduced a key-value command set.

- Most of commercially and academically released KV-SSDs have utilized the NVMe key-value command set to offer key-value interface.

***SK hynix KV-CSD* [2]**



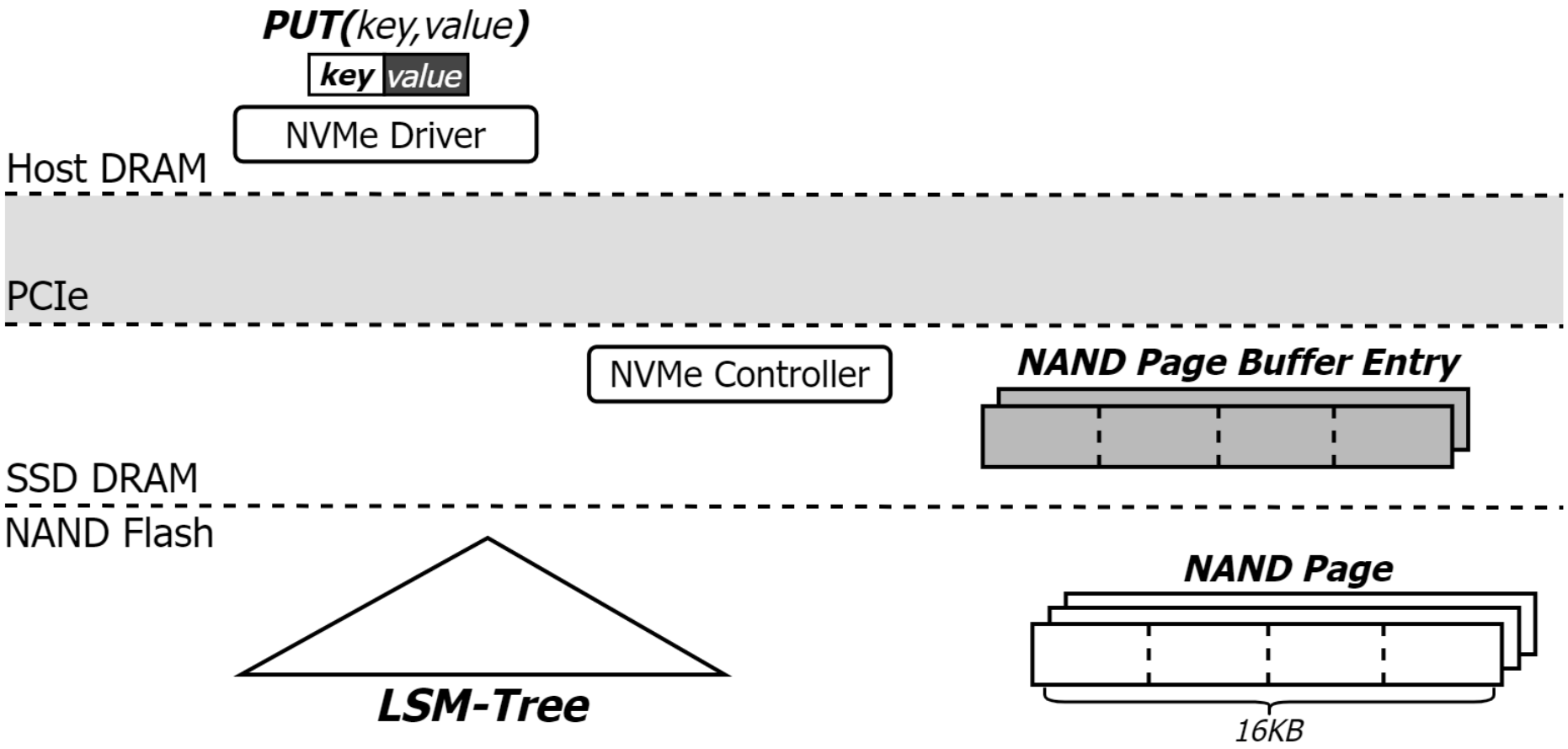
Academia



[2] Park, I., Zheng, Q., Manno, D., Yang, S., Lee, J., Bonnie, D., Settlemyer, B., Kim, Y., Chung, W., & Grider, G. (2023). KV-CSD: A Hardware-Accelerated Key-Value Store for Data-Intensive Applications. In Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER), 132–144.

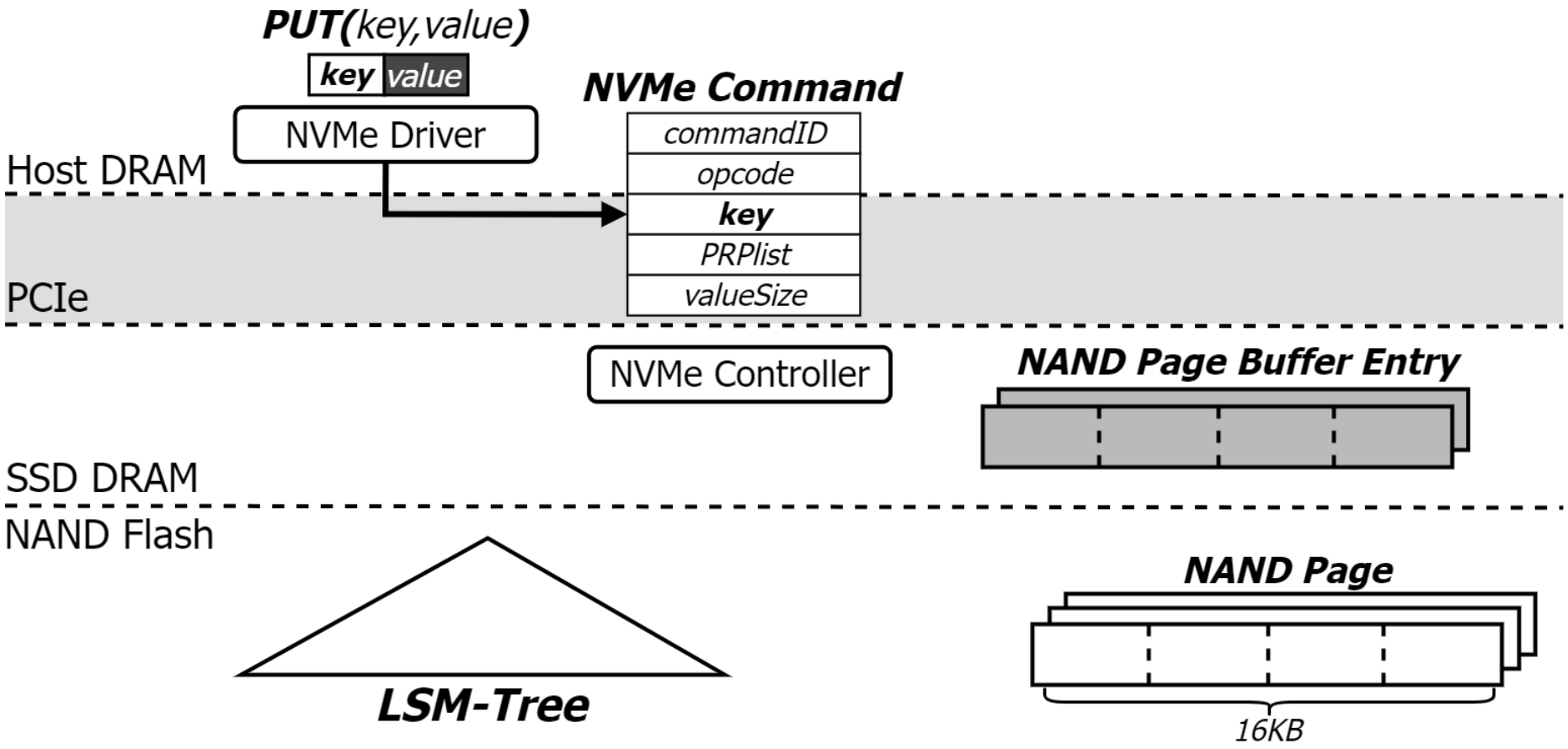
NVMe Key-Value Write Mechanism

- In a case of NVMe KV-SSD based on the LSM-tree with a key-value separation (e.g., iLSM-SSD, KV-CSD), when writing key-value pairs, ...



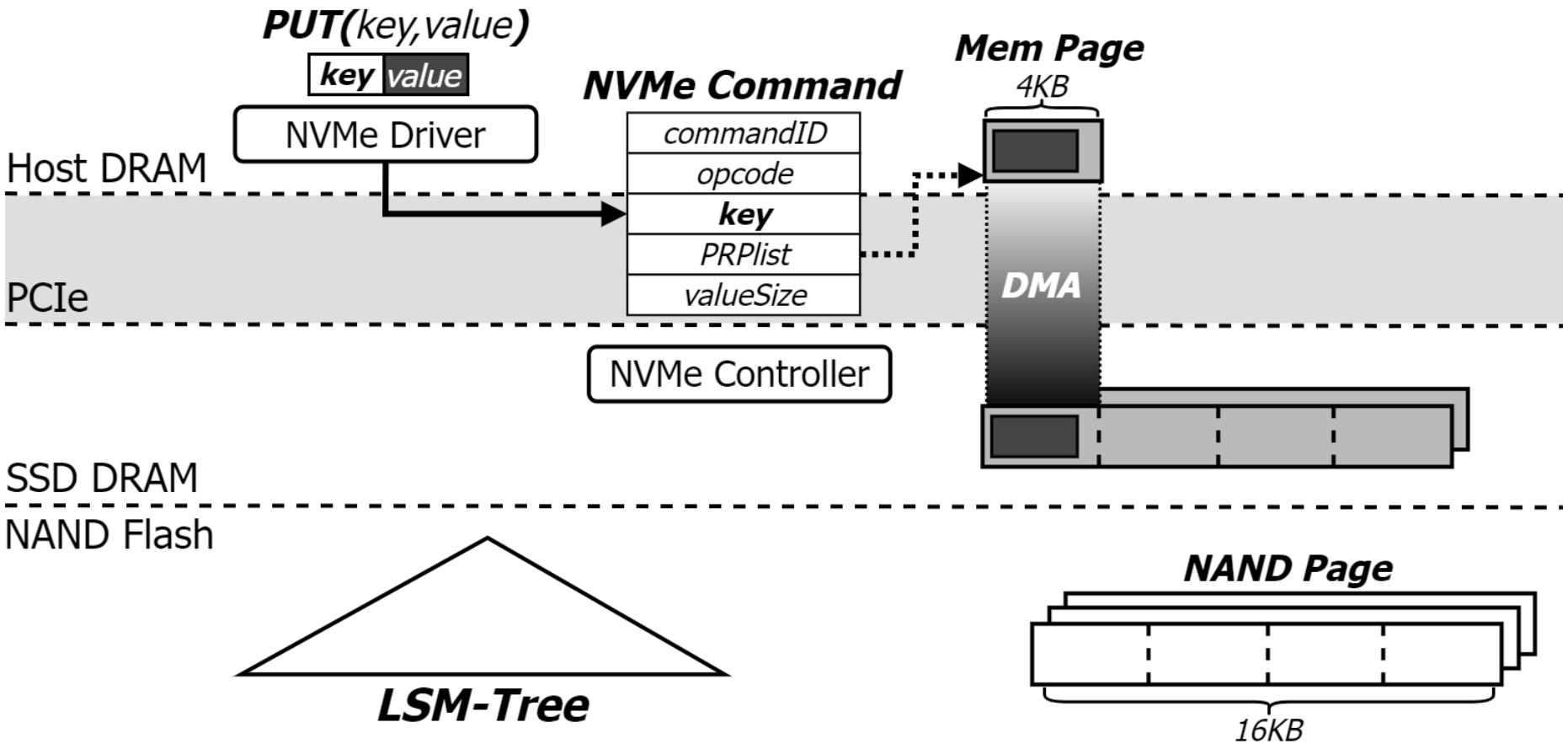
NVMe Key-Value Write Mechanism

- The NVMe driver stores a key and metadata in the NVMe command, and then submits the command to the SQ and rings the doorbell.



NVMe Key-Value Write Mechanism

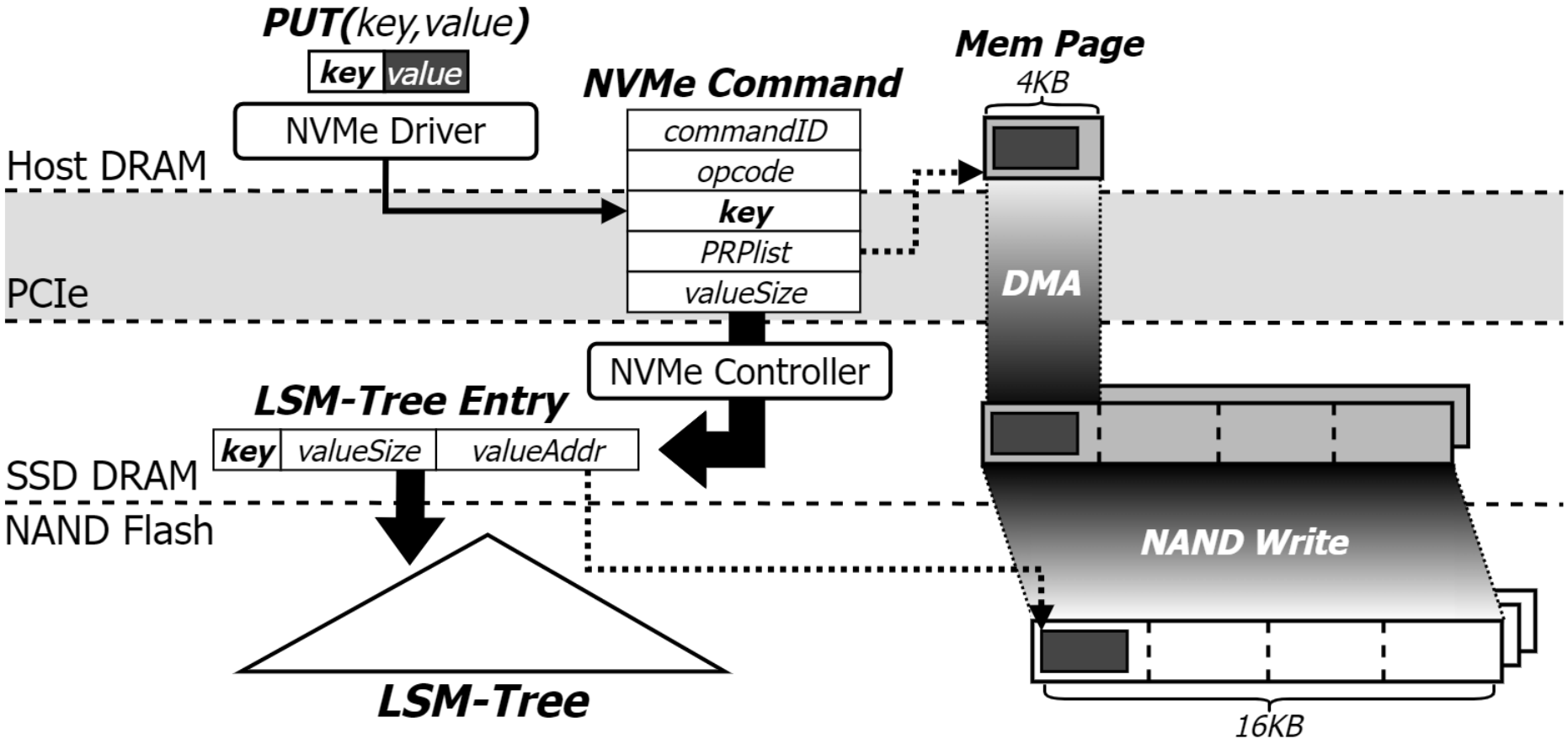
- The NVMe controller issues a DMA transaction to copy the payload (value) to the NAND page buffer within the device's DRAM.



NVMe Key-Value Write Mechanism

- The controller constructs the LSM-tree entry containing the key, value size, and value pointer, and programs the NAND page buffer entry.

(to show the flow clearly, it programs the NAND page buffer entry even though it's not full)



Motivation

Problem Definition

- According to Meta, their popular LSM KVS, RocksDB, in a production environment experiences the size of values nearly not reaching a hundred bytes on average [3], which is far less than the 4 KiB memory page size.

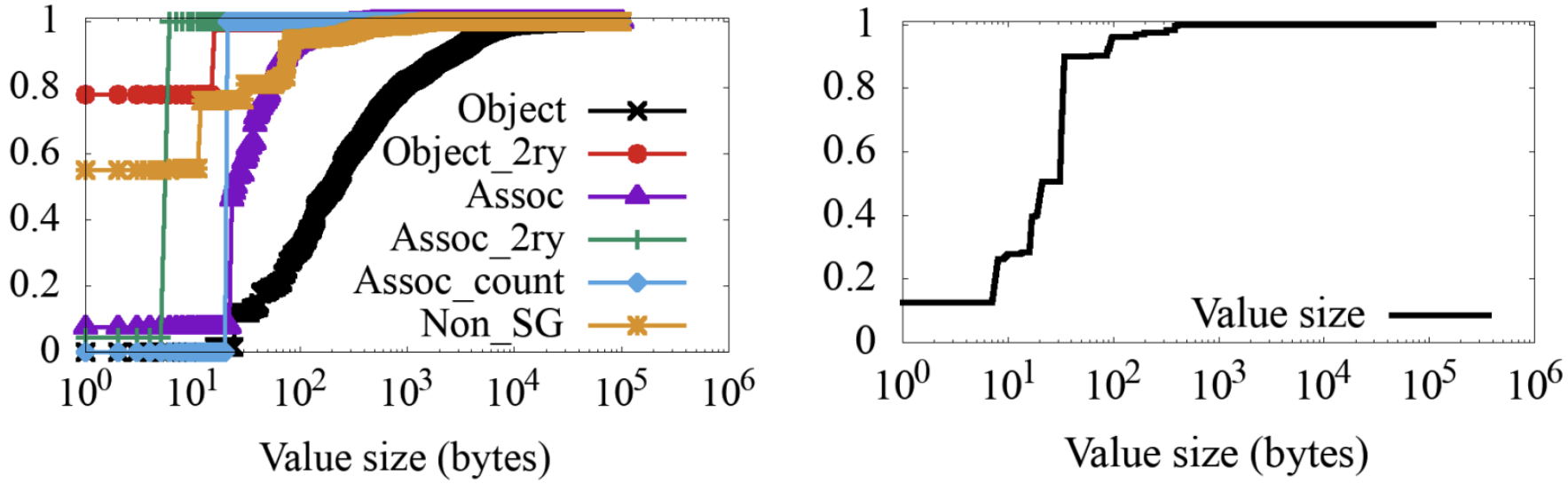
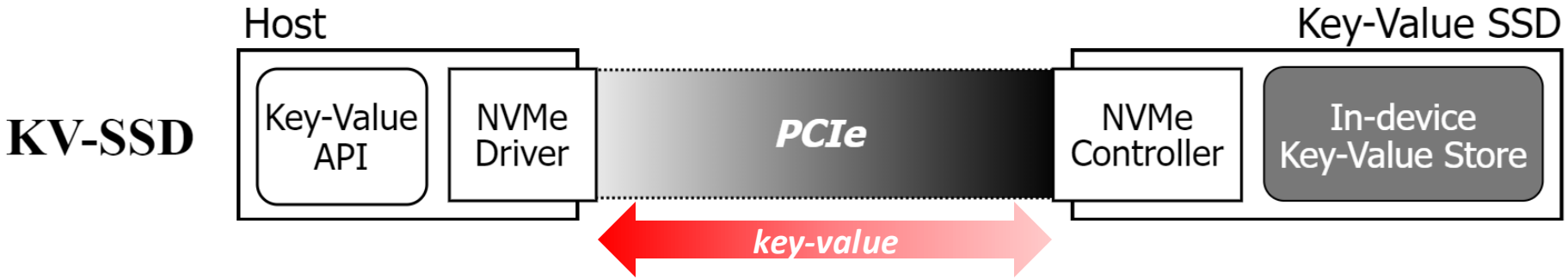


Figure – Value Size CDF for RocksDB as a MySQL storage layer (left) and RocksDB as a distributed KVS (right)

[3] Cao, Z., Dong, S., Vemuri, S., & Du, D. H. C. (2020). *Characterizing, modeling, and benchmarking RocksDB key-value workloads at Facebook. In Proceedings of the 18th USENIX Conference on File and Storage Technologies* (FAST '20) (pp. 1-14). Santa Clara, CA, USA.

Problem Definition

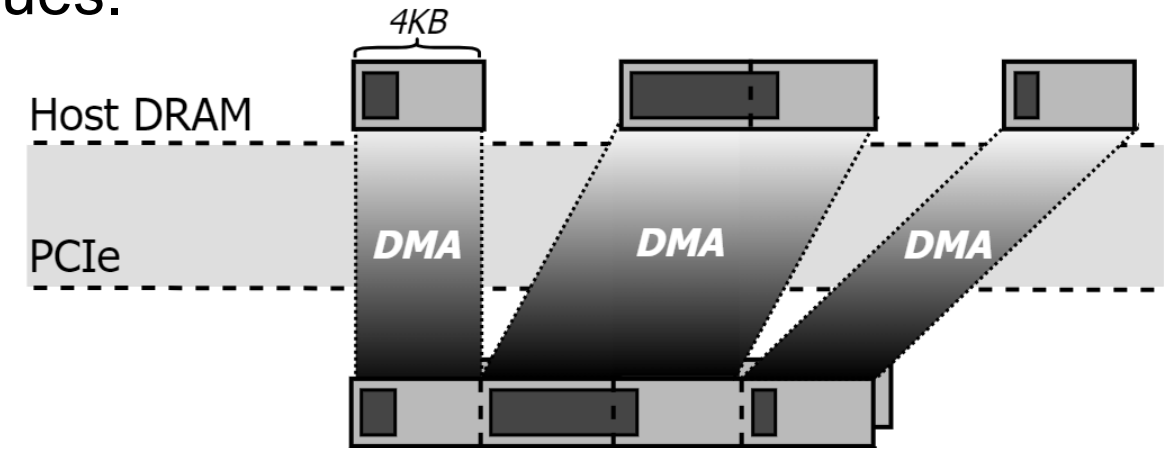
- The problem occurs with the fact that the NVMe key-value interface still cannot extricate itself from the deeply entrenched block-interface-assumed storage mechanisms and frameworks.



➔ **is it really a `*key-value*` interface?**

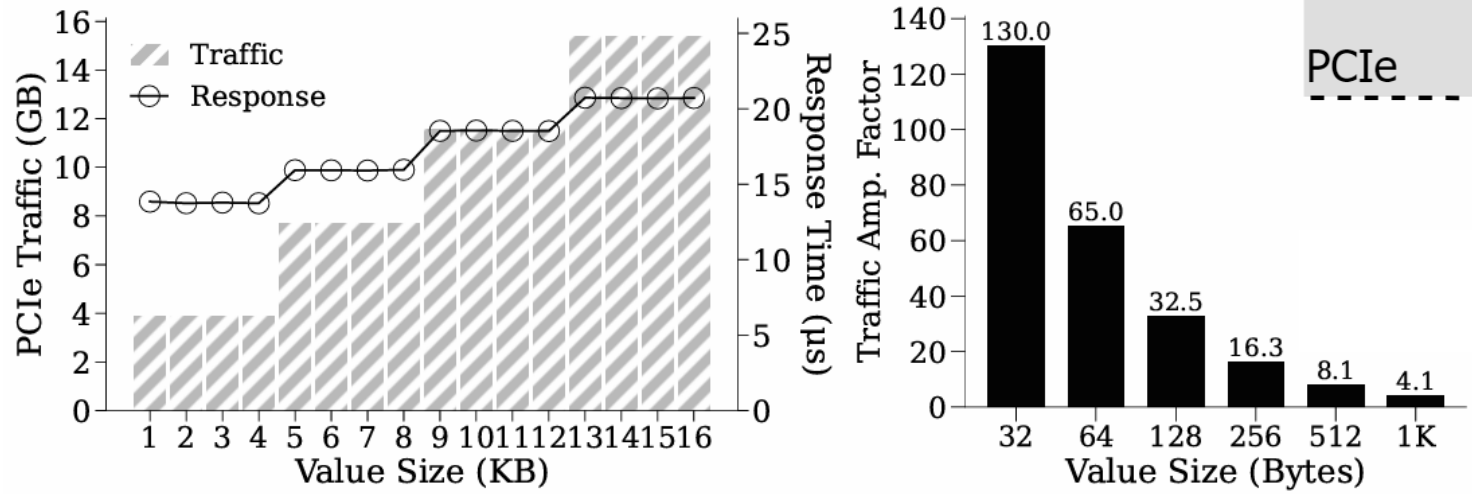
PCIe Traffic and Transfer Latency Bloating

- The NVMe's payload transfer method, PRP, restricts DMA transfers to occur in units of 4 KiB, a size of memory page.
  - This leads to the bloated PCIe traffic and latency during value transfers, especially for variable-sized, small values.

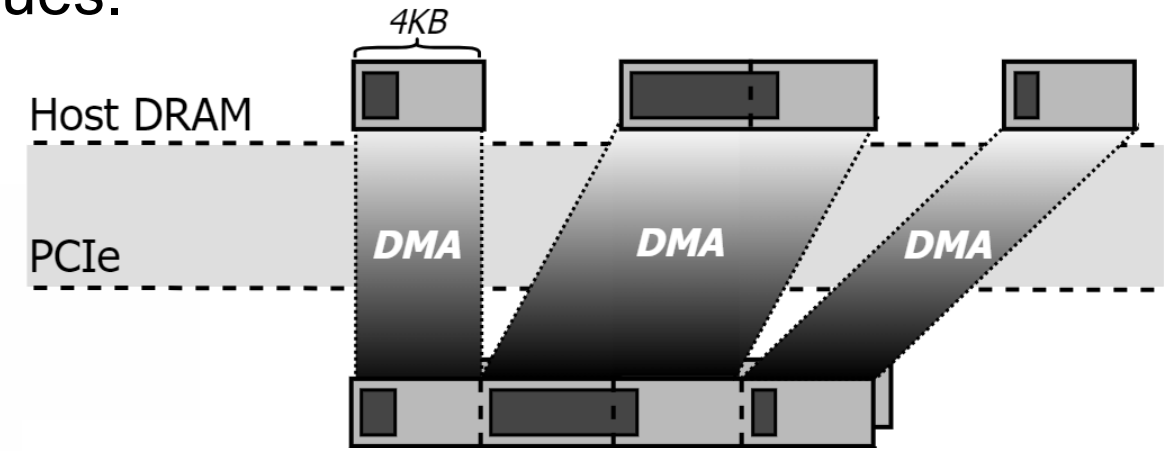


PCIe Traffic and Transfer Latency Bloating

- The NVMe's payload transfer method, PRP, restricts DMA transfers to occur in units of 4 KiB, a size of memory page.
  - This leads to the bloated PCIe traffic and latency during value transfers, especially for variable-sized, small values.



(a) Total PCIe Traffic & Avg. Resp. Time (b) Traffic Amplification

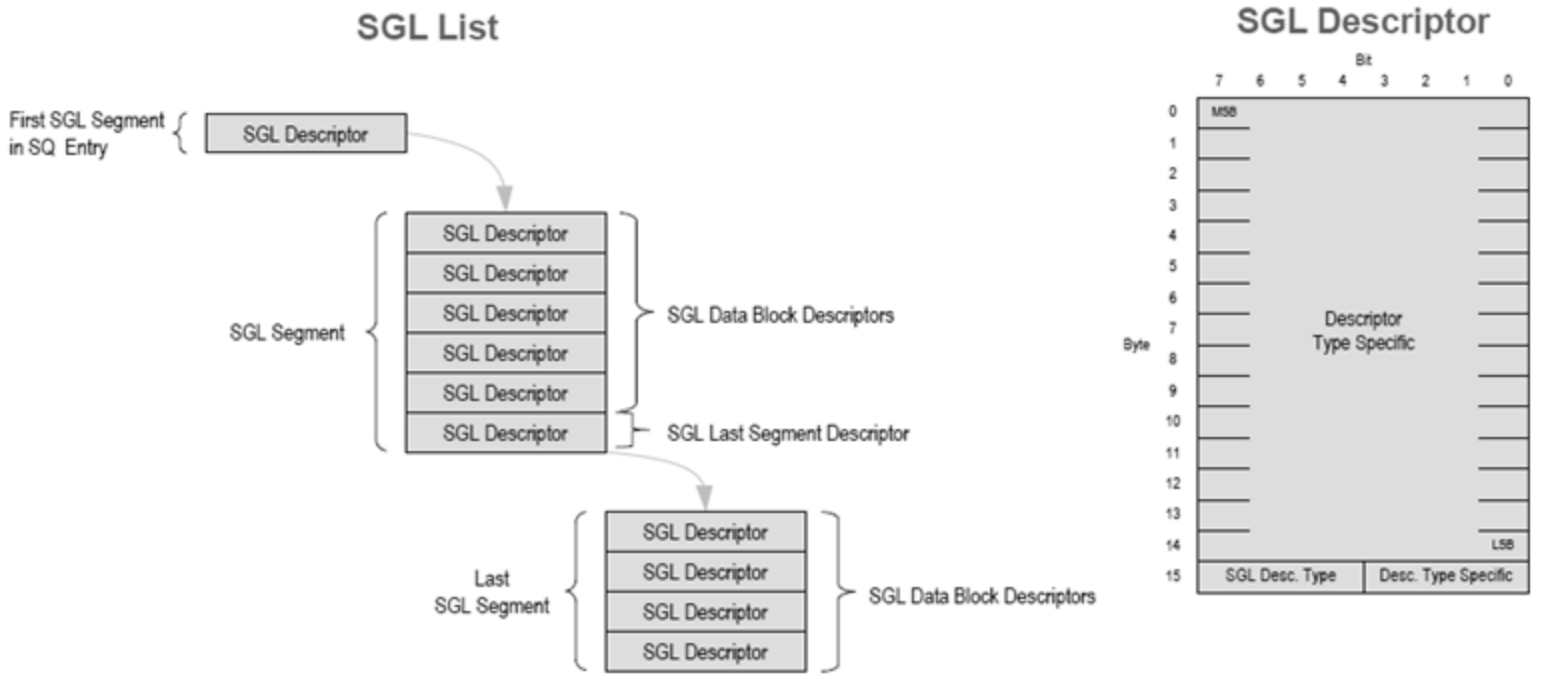


※ Traffic Amplification = (value size) / (PCIe traffic)

Setup	IterKVSSD (Systor '23) on Cosmos+ OpenSSD platform - feature: SOTA LSM-based KV-SSD - PCIe Gen2 x8 lane - 1GB of DRAM, 1TB of NAND (Toshiba), Xilinx zynq-7000
Workload	fillsequential of RocksDB's db_bench - number of PUTs: 1 million unique KV pairs - key size: 4 B

Limitation of Other Methods – NVMe SGL

- NVMe's another payload transfer mechanism, Scatter-Gather List (SGL), can support multiple variable-sized DMAs across scattered memory segments.



Limitation of Other Methods – NVMe SGL

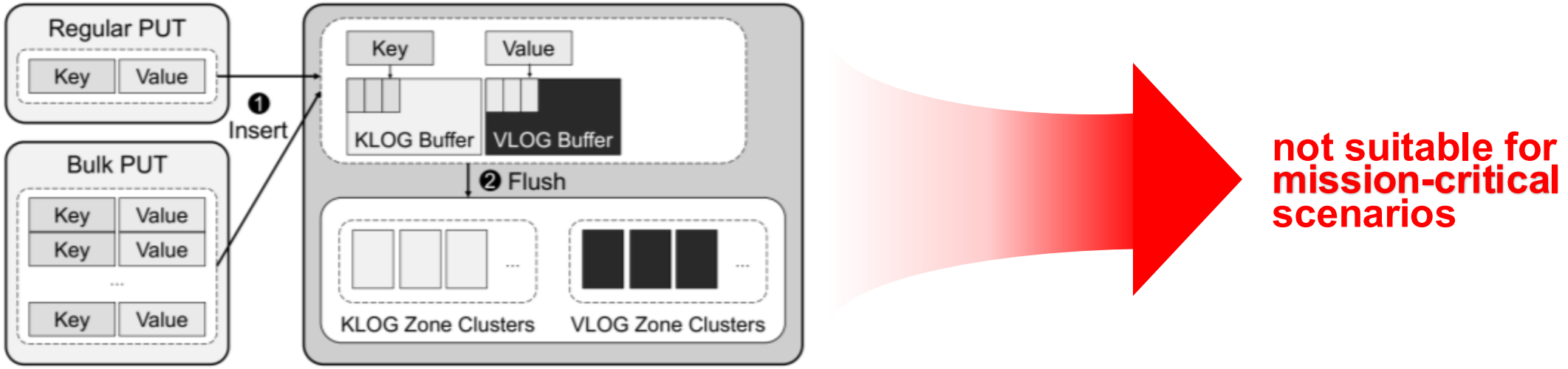
- However, it has been reported that the cost of enabling the SGL outweighs the benefit for I/O smaller than 32 KiB [4].
  - The Linux kernel thus establishes a minimum threshold for data transferred via SGL at 32 KiB [5], indicating that using SGL for small value transfers is not advisable.

```
60      static unsigned int sgl_threshold = SZ_32K;
61      module_param(sgl_threshold, uint, 0644);
62      MODULE_PARM_DESC(sgl_threshold,
63                      "Use SGLs when average request segment size is larger or equal to "
64                      "this size. Use 0 to disable SGLs.");
65
66      #define NVME_PCI_MIN_QUEUE_SIZE 2
67      #define NVME_PCI_MAX_QUEUE_SIZE 4095
68      static int io_queue_depth_set(const char *val, const struct kernel_param *kp);
69      static const struct kernel_param_ops io_queue_depth_ops = {
70              .set = io_queue_depth_set,
71              .get = param_get_uint,
72      };
```

[4] 2017. nvme : add Scatter-Gather List (SGL) support in NVMe driver. https://lore.kernel.org/all/04aaed5c-1a8a-f601-6c9c-88bf1cf66e8a@mellanox.com/T/
[5] The Linux Kernel source code. sgl_threshold. https://github.com/torvalds/linux/blob/master/drivers/nvme/host/pci.c

Limitation of Other Methods – Host Batching

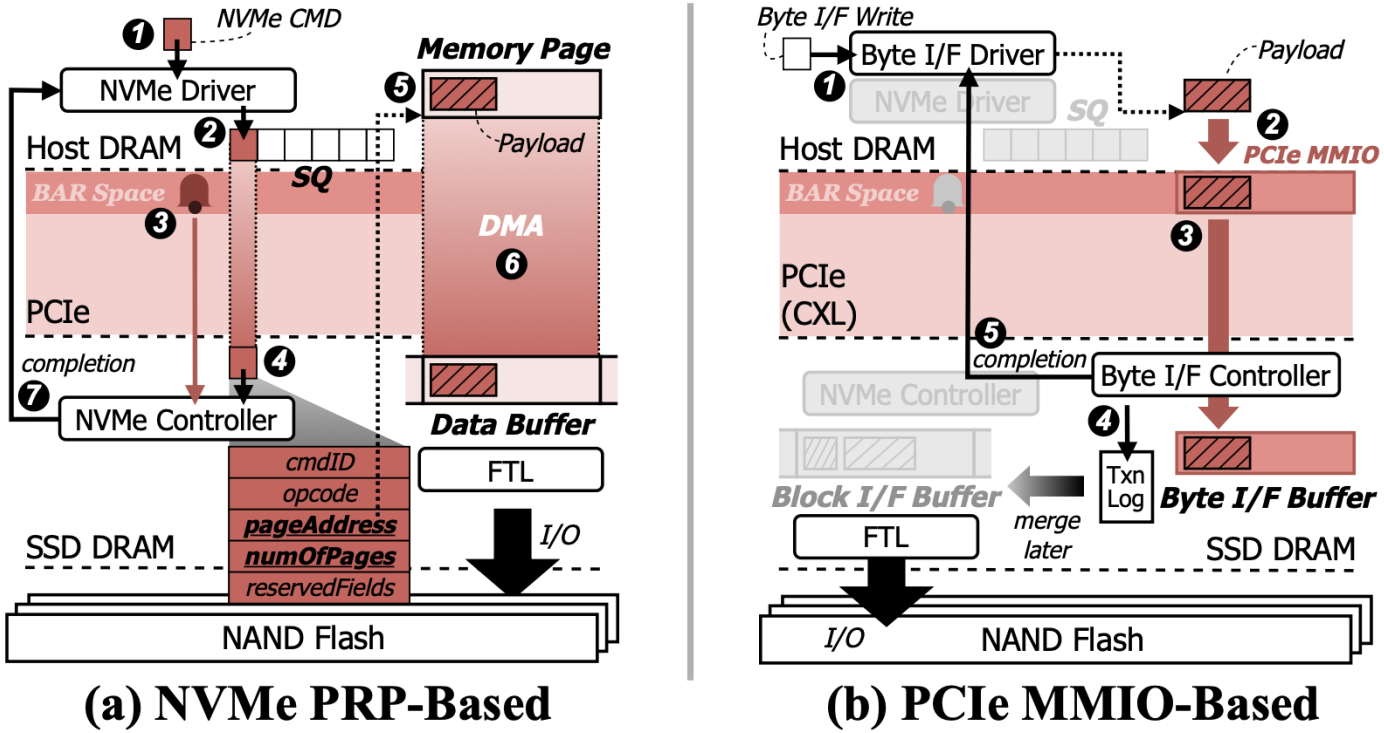
- KV-CSD and Dotori [6] have tackled this issue by implementing bulk PUT operation, which is host-side batching.
  - However, a fundamental issue with buffering the key-value entries on the host side is the risk of data loss on power failure.



[6] Duffy, C., Shim, J., Kim, S.-H., & Kim, J.-S. (2023). *Dotori: A Key-Value SSD Based KV Store. Proceedings of the VLDB Endowment,* 16(6), 1560–1572.

Limitation of Other Methods – PCIe MMIO

- PCIe MMIO-based transfer enables low-latency data exchange by letting hosts write small payloads directly into SSD memory via the BAR space.
  - Solutions like 2B-SSD [7] and ByteFS [8] use cache-line-level writes to bypass traditional block I/O paths.

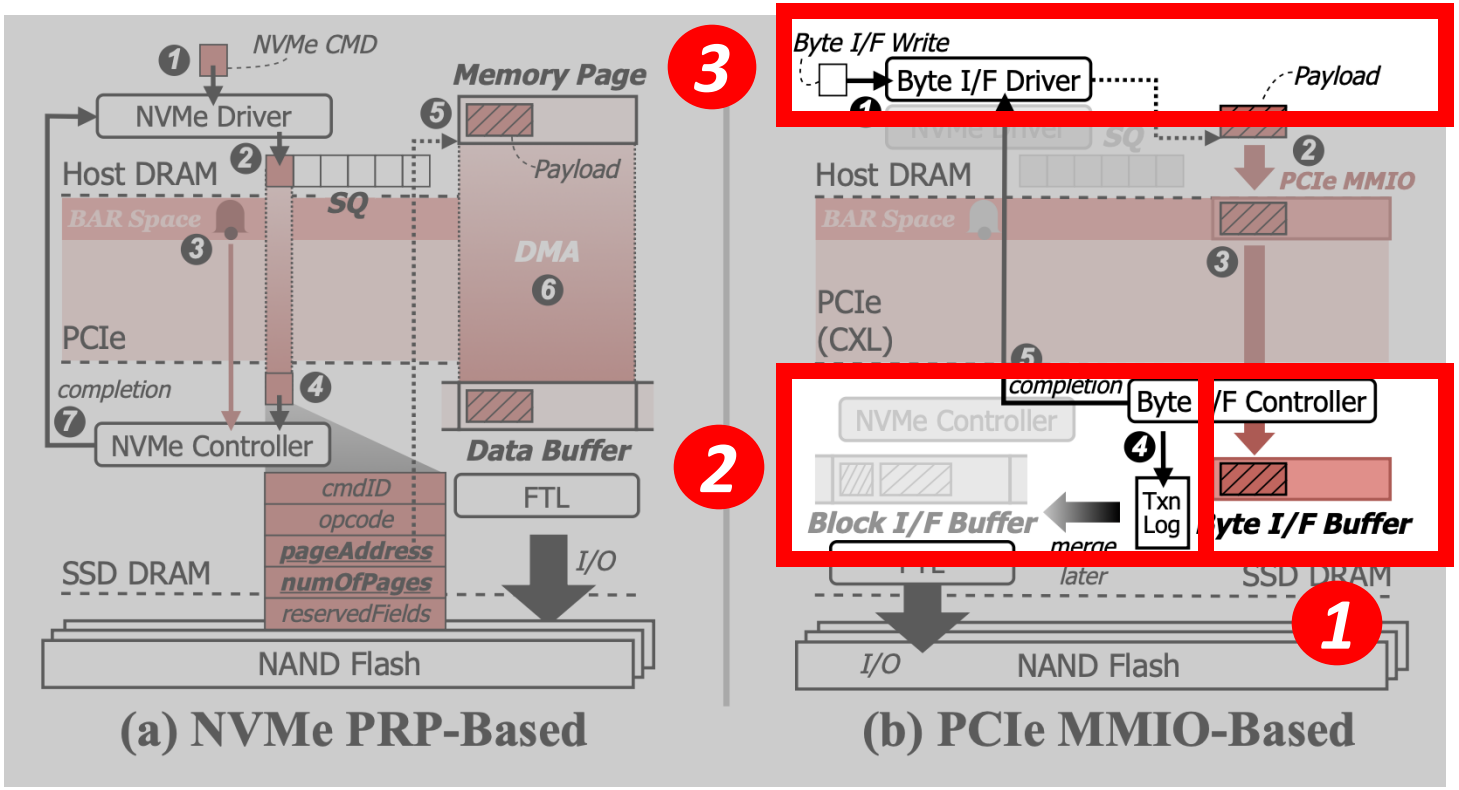


[7] D. -H. Bae et al., "2B-SSD: The Case for Dual, Byte- and Block-Addressable Solid-State Drives," 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA).

[8] S. Li et al., “ByteFS: System Support for (CXL-based) Memory-Semantic Solid-State Drives”. 2025 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '25).

Limitation of Other Methods – PCIe MMIO

- However, integrating this into existing NVMe devices is difficult due to the need for ❶ extra buffer memory, ❷ transactional coordination, and ❸ new host interfaces.
  - Significant modifications to SSD architectures and firmware logic.
  - Existing user-level APIs cannot be reused.



[7] D. -H. Bae et al., "2B-SSD: The Case for Dual, Byte- and Block-Addressable Solid-State Drives," 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA).

[8] S. Li et al., “ByteFS: System Support for (CXL-based) Memory-Semantic Solid-State Drives”. 2025 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '25).

Proposed Solutions

BandSlim & ByteExpress

Solutions: ***BandSlim & ByteExpress***

- To tackle the PCIe traffic amplification and latency bloating problem occurring in small key-value transfer under the NVMe key-value transaction, we introduce ***BandSlim[1]*** and ***ByteExpress[2]***.

[1] BandSlim: A Novel Bandwidth and Space-Efficient KV-SSD with an Escape-from-Block Approach, ICPP, 2024

[2] ByteExpress: A High-Performance and Traffic-Efficient Inline Transfer of Small Payloads over NVMe, (under review)

* ByteExpress is a proposed technique from a recently submitted research paper, currently under review, and serves as an improved version of BandSlim.

BandSlim's Fine-Grained Payload Transfer

- ***BandSlim*** employs a fine-grained inline value transfer mechanism that piggybacks values smaller than a memory page size to NVMe commands using the reserved fields (gray-colored in Figure (a)&(b)).

dword	description				
dword0	commandID	P		F	opcode
dword1	namespaceID				
dword2	key				
dword3					
dword4	metadataPointer (PRP)				
dword5					
dword6	PRPlistEntry1				
dword7					
dword8	PRPlistEntry2				
dword9					
dword10	valueSize				
dword11	reserved		option		keySize
dword12	reserved				
dword13					
dword14	key				
dword15					

(a) Write Command

dword	description				
dword0	commandID	P		F	opcode
dword1	namespaceID				
dword2	key				
dword3					
dword4	metadataPointer (PRP)				
dword5					
dword6	PRPlistEntry1				
dword7					
dword8	PRPlistEntry2				
dword9					
dword10	valueSize				
dword11	reserved		option		keySize
dword12	reserved				
dword13					
dword14	key				
dword15					

(b) Transfer Command

BandSlim's Fine-Grained Payload Transfer

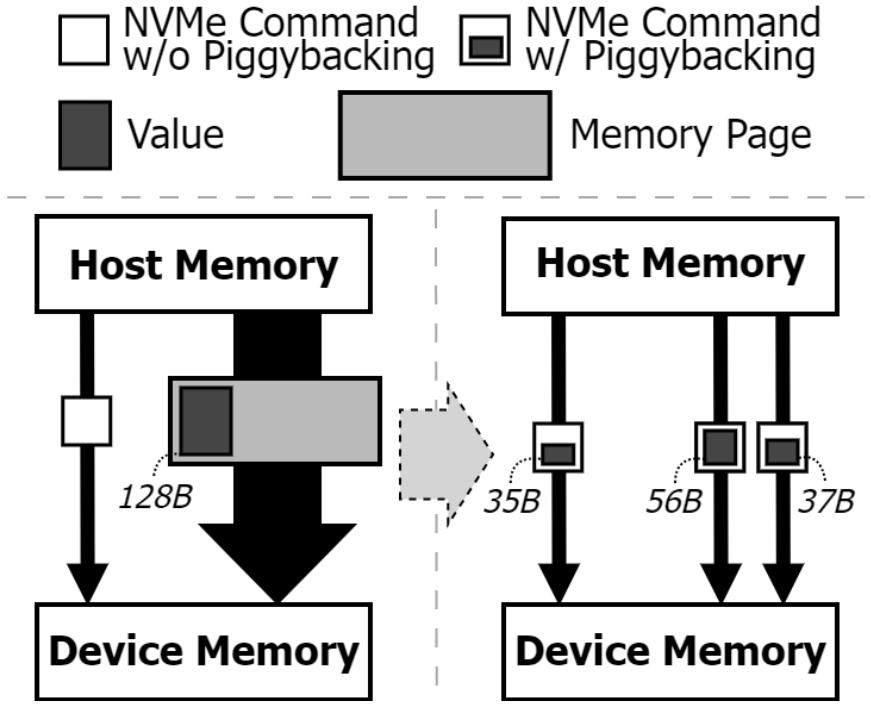
- ***BandSlim*** employs a fine-grained inline value transfer mechanism that piggybacks values smaller than a memory page size to NVMe commands using the reserved fields (gray-colored in Figure (a)&(b)).

dword	description				
dword0	commandID	P		F	opcode
dword1	namespaceID				
dword2	key				
dword3					
dword4	metadataPointer (PRP)				
dword5					
dword6	PRPlistEntry1				
dword7					
dword8	PRPlistEntry2				
dword9					
dword10	valueSize				
dword11	reserved		option		keySize
dword12	reserved				
dword13					
dword14	key				
dword15					

(a) Write Command

dword	description				
dword0	commandID	P		F	opcode
dword1	namespaceID				
dword2	key				
dword3					
dword4	metadataPointer (PRP)				
dword5					
dword6	PRPlistEntry1				
dword7					
dword8	PRPlistEntry2				
dword9					
dword10	valueSize				
dword11	reserved		option		keySize
dword12	reserved				
dword13					
dword14	key				
dword15					

(b) Transfer Command



BandSlim's Fine-Grained Payload Transfer

- ***BandSlim*** employs a fine-grained inline value transfer mechanism that piggybacks values smaller than a memory page size to NVMe commands using the reserved fields (gray-colored in Figure (a)&(b)).

dword	description				
dword0	commandID	P		F	opcode
dword1	namespaceID				
dword2	key				
dword3					
dword4	metadataPointer (PRP)				
dword5					
dword6	PRPlistEntry1				
dword7					
dword8	PRPlistEntry2				
dword9					
dword10	valueSize				
dword11	reserved		option		keySize
dword12	reserved				
dword13					
dword14	key				
dword15					

(a) Write Command

dword	description				
dword0	commandID	P		F	opcode
dword1	namespaceID				
dword2	key				
dword3					
dword4	metadataPointer (PRP)				
dword5					
dword6	PRPlistEntry1				
dword7					
dword8	PRPlistEntry2				
dword9					
dword10	valueSize				
dword11	reserved		option		keySize
dword12	reserved				
dword13					
dword14	key				
dword15					

(b) Transfer Command

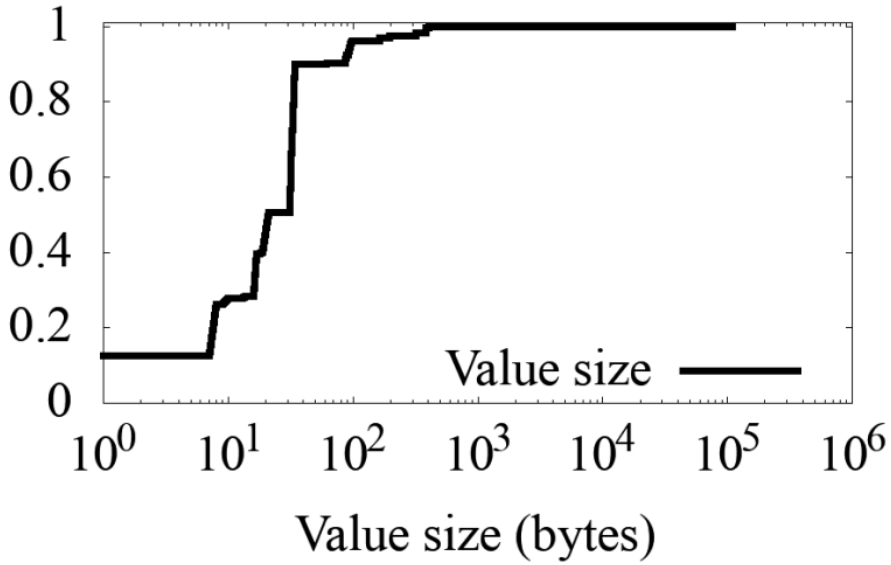


Figure – Value Size CDF for RocksDB in a production environment

BandSlim's Fine-Grained Payload Transfer

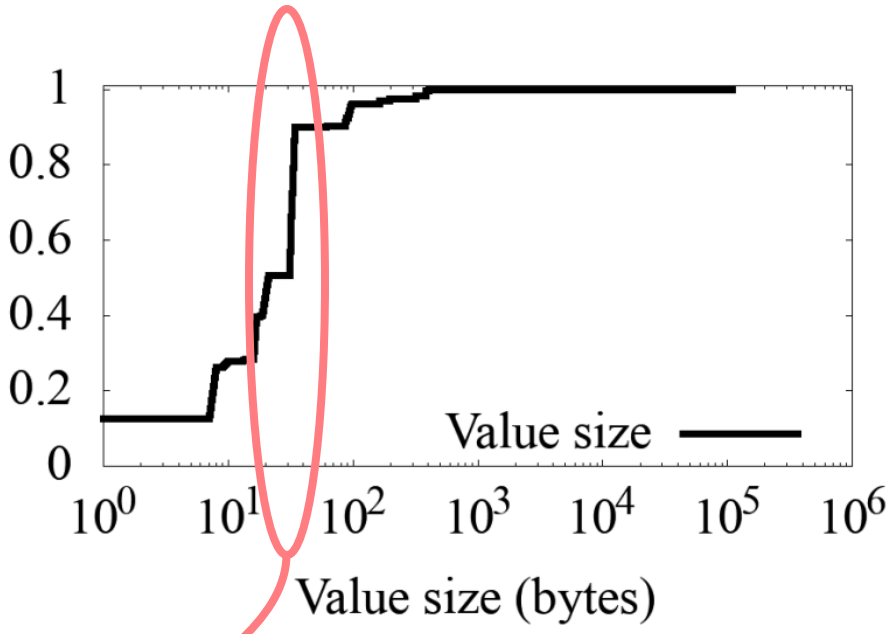
- ***BandSlim*** employs a fine-grained inline value transfer mechanism that piggybacks values smaller than a memory page size to NVMe commands using the reserved fields (gray-colored in Figure (a)&(b)).

dword	description				
dword0	commandID	P		F	opcode
dword1	namespaceID				
dword2	key				
dword3					
dword4	metadataPointer (PRP)				
dword5					
dword6	PRPlistEntry1				
dword7					
dword8	PRPlistEntry2				
dword9					
dword10	valueSize				
dword11	reserved		option		keySize
dword12	reserved				
dword13					
dword14	key				
dword15					

(a) Write Command

dword	description				
dword0	commandID	P		F	opcode
dword1	namespaceID				
dword2	key				
dword3					
dword4	metadataPointer (PRP)				
dword5					
dword6	PRPlistEntry1				
dword7					
dword8	PRPlistEntry2				
dword9					
dword10	valueSize				
dword11	reserved		option		keySize
dword12	reserved				
dword13					
dword14	key				
dword15					

(b) Transfer Command

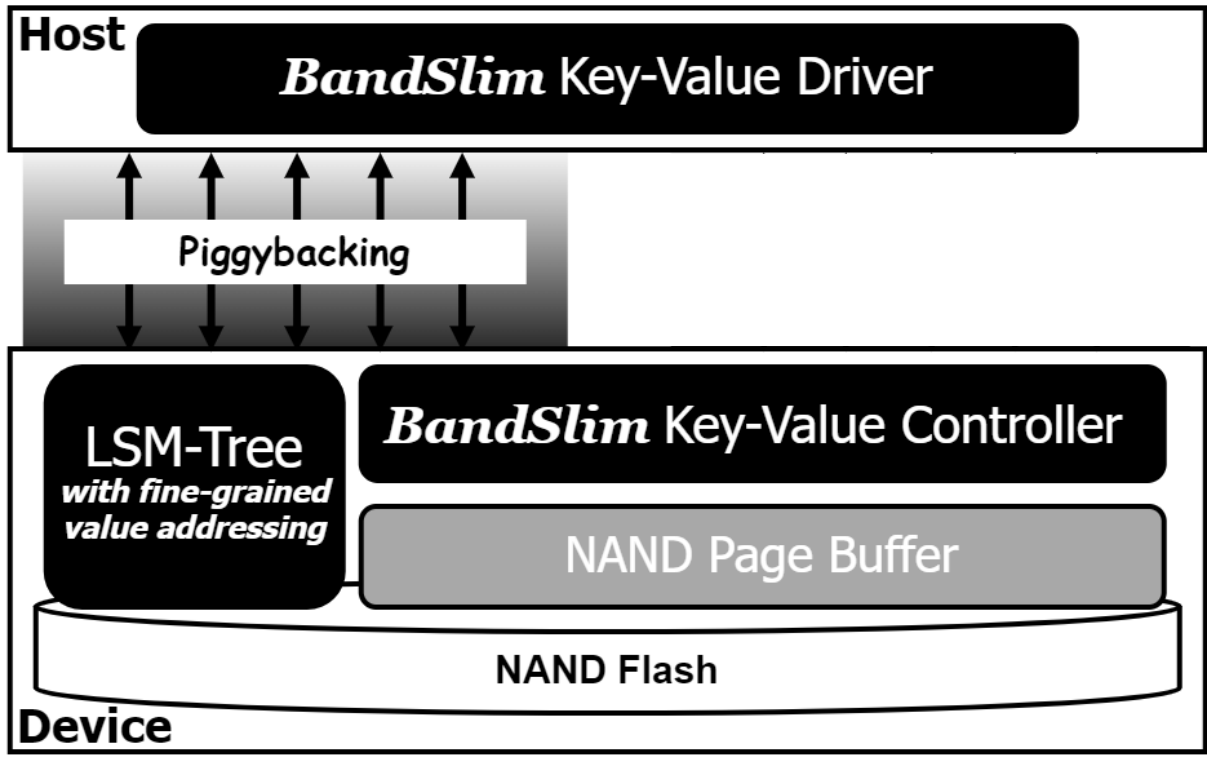


64 B NVMe command gives an opportunity

Figure – Value Size CDF for RocksDB in a production environment

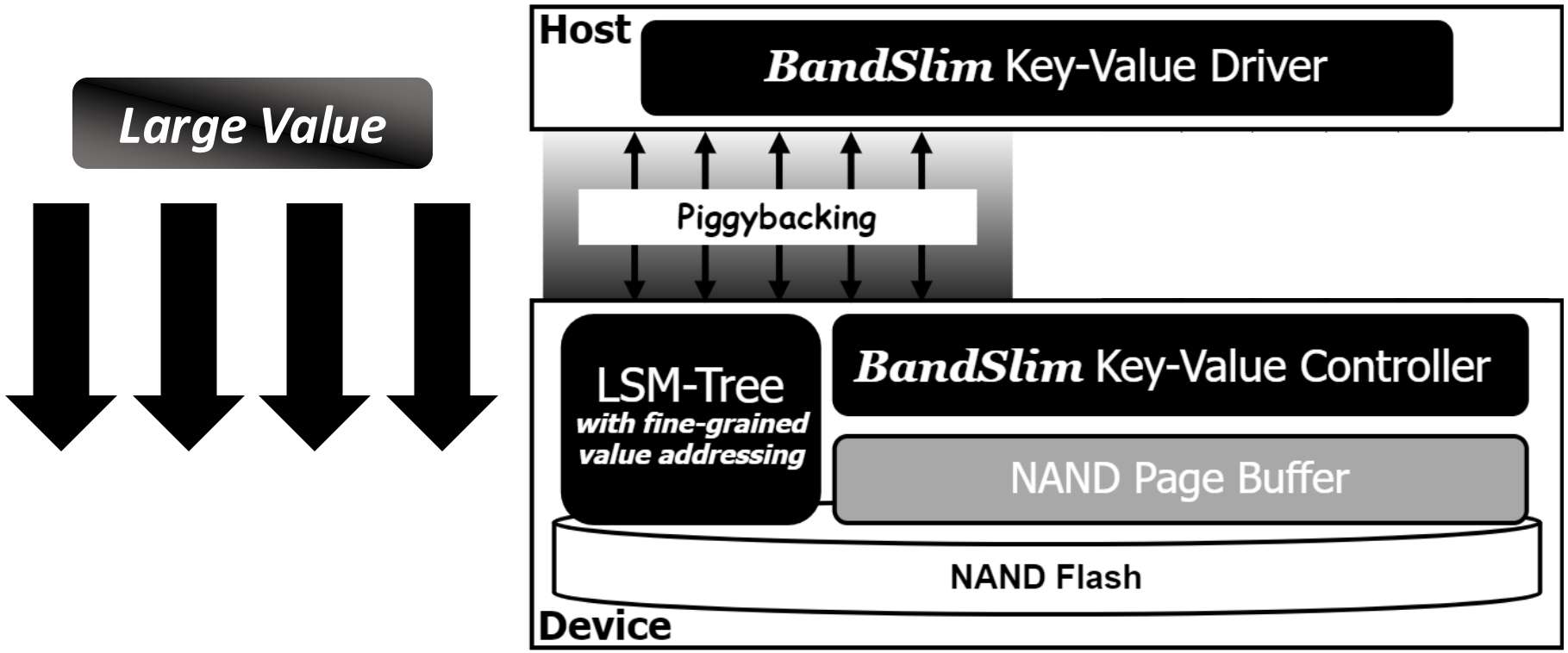
Adaptive Value Transfer Optimization

- When transmitting large values, generating and sending multiple NVMe commands in this manner can result in longer response times.
  - Thus, ***BandSlim*** also incorporates an *adaptive value transfer* strategy that switches back and forth piggybacking and page-unit DMA.



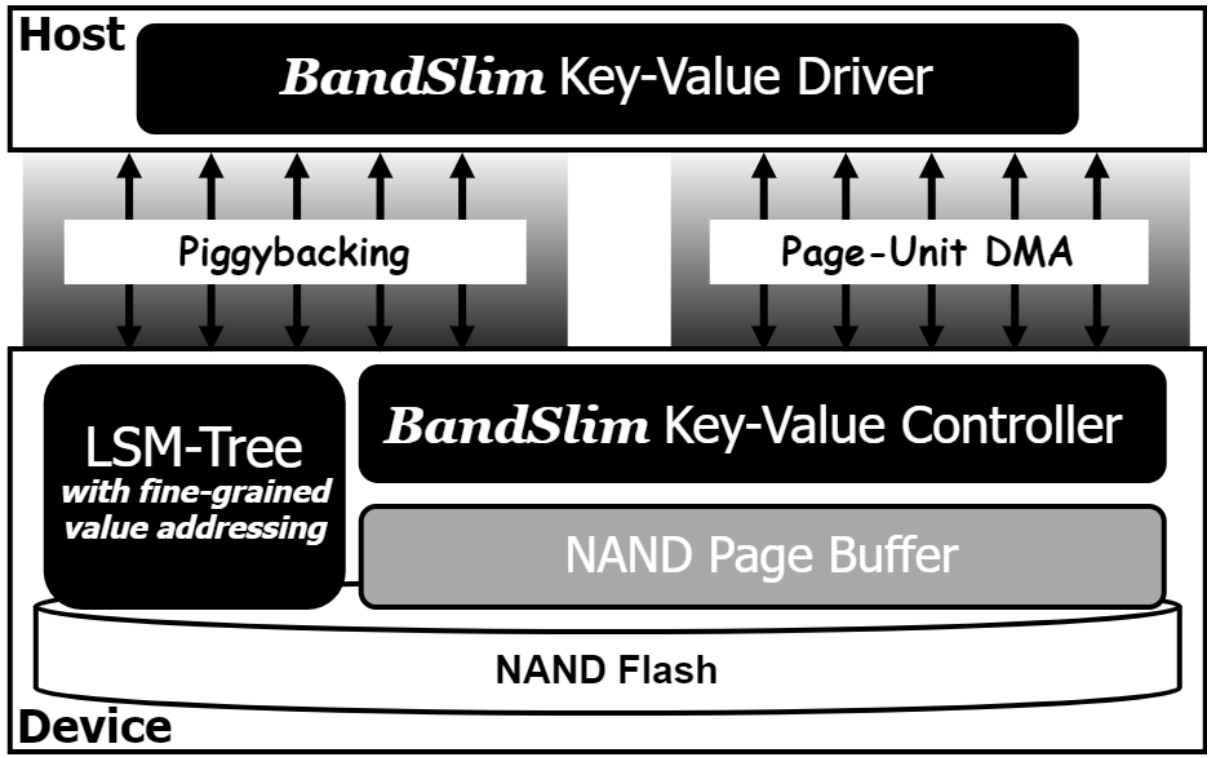
Adaptive Value Transfer Optimization

- When transmitting large values, generating and sending multiple NVMe commands in this manner can result in longer response times.
  - Thus, ***BandSlim*** also incorporates an *adaptive value transfer* strategy that switches back and forth piggybacking and page-unit DMA.



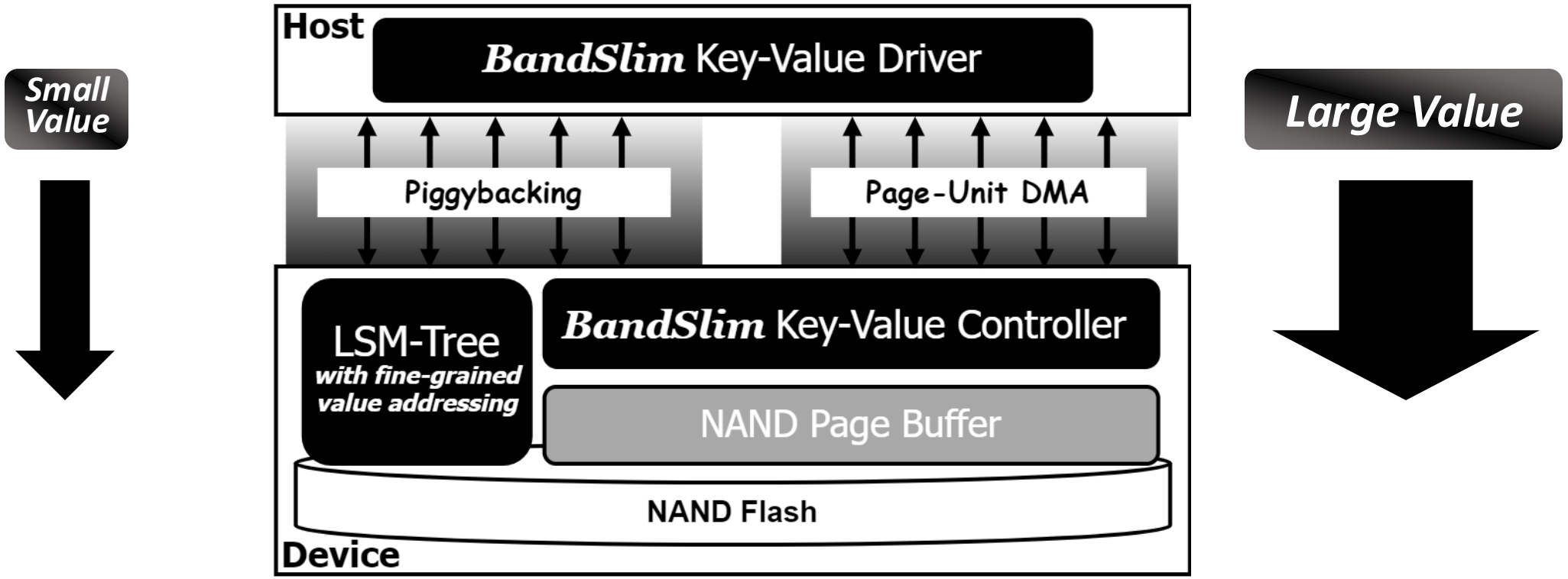
Adaptive Value Transfer Optimization

- When transmitting large values, generating and sending multiple NVMe commands in this manner can result in longer response times.
  - Thus, ***BandSlim*** also incorporates an *adaptive value transfer* strategy that switches back and forth piggybacking and page-unit DMA.



Adaptive Value Transfer Optimization

- When transmitting large values, generating and sending multiple NVMe commands in this manner can result in longer response times.
  - Thus, ***BandSlim*** also incorporates an *adaptive value transfer* strategy that switches back and forth piggybacking and page-unit DMA.



Evaluation Setup

- Testbed:

KV-SSD on Cosmos+ OpenSSD Platform

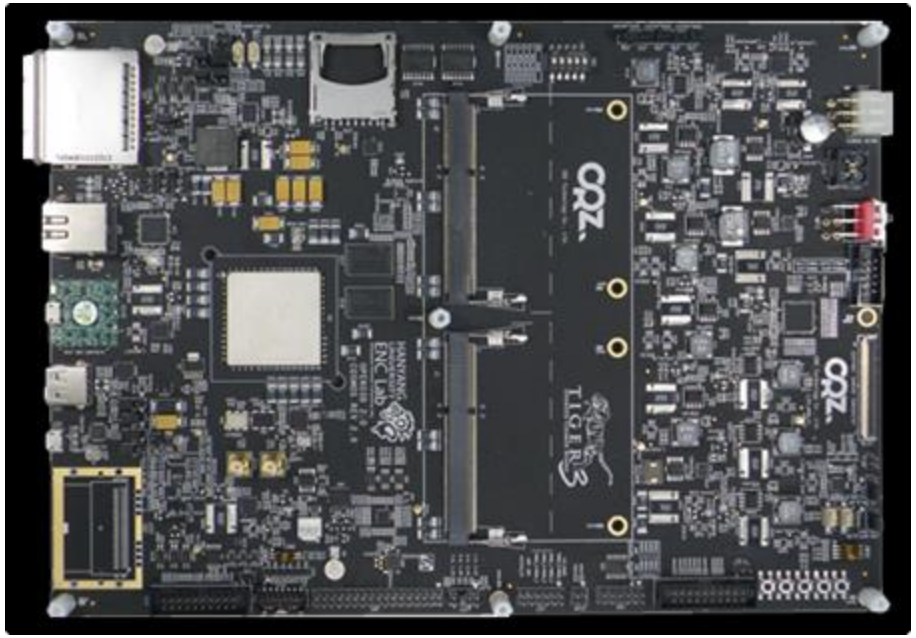


Table 1: HW/SW specifications of the OpenSSD platform.

SoC	Xilinx Zynq-7000 with ARM Cortex-A9 Core
NAND Module	1TB, 4 Channel & 8 Way
Interconnect	PCIe Gen2 ×8 End-Points

Table 2: HW/SW specifications of the host node.

CPU	Intel(R) Xeon(R) Gold 6226R CPU @ 2.90GHz (32 cores)
Memory	384GB DDR4
OS	Ubuntu 22.04

Evaluation Setup

- Test Configurations:

Baseline (NVMe PRP)	**State-of-the-art LSM-based NVMe KV-SSD, IterKVSSD (Systor '23).**
Piggyback (BandSlim)	**It transfers values using only piggybacking-based transfer method.**
Adaptive	**It transfers values using the adaptive value transfer method.**

Evaluation Setup

- Workloads (Meta's db_bench):

W(B)	**fillrandom, 1 million PUTs, value sizes of 8 B or 2 KiB at a 9:1 ratio.**
W(C)	**Same as *W(B)* but with the value size ratio reversed to 1:9.**
W(D)	**fillrandom, 1 million PUTs, values sizes of 8 B, 16 B, 32 B, 64 B, 128 B, 256 B, 512 B, 1 KiB, and 2 KiB with each size having an equal ratio.**
W(M)	**mixgraph (real-world workloads with a maximum value size of 1 KiB and almost 70% of values being under 35 B), 1 million PUTs.**

Evaluation Setup

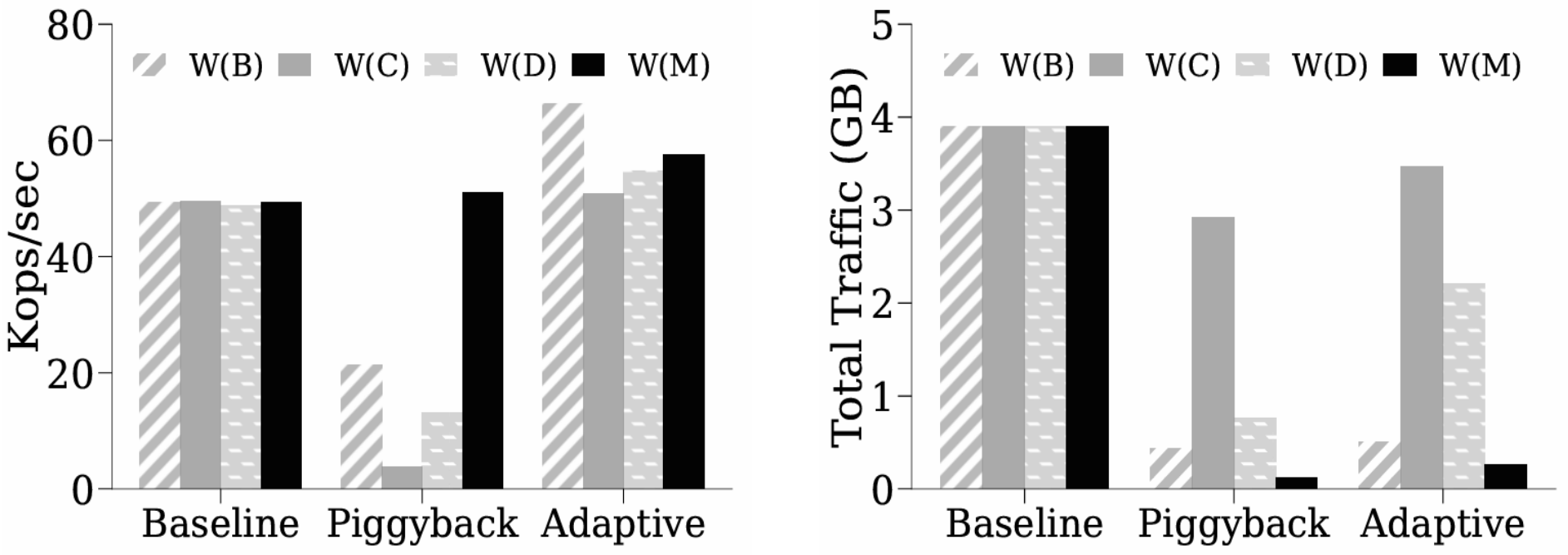
- Workloads (Meta's db_bench):

W(B)	→ **Small Value Dominant** (8B:2KB = 9:1)
W(C)	→ **Large Value Dominant** (8B:2KB = 1:9)
W(D)	→ **Balanced Value Size** (8B ~ 2KB)
W(M)	→ **Real-World Pattern** (mostly around 64B)

Fine-Grained Value Transfer

Various Workloads (*W(B) ~ W(M)*)

- *Piggyback (BandSlim)* significantly reduced total PCIe traffic across all workloads and improved the average throughput compared to *Baseline (NVMe PRP)* for ***W(****M*), which reflects real-world value size characteristics of RocksDB.



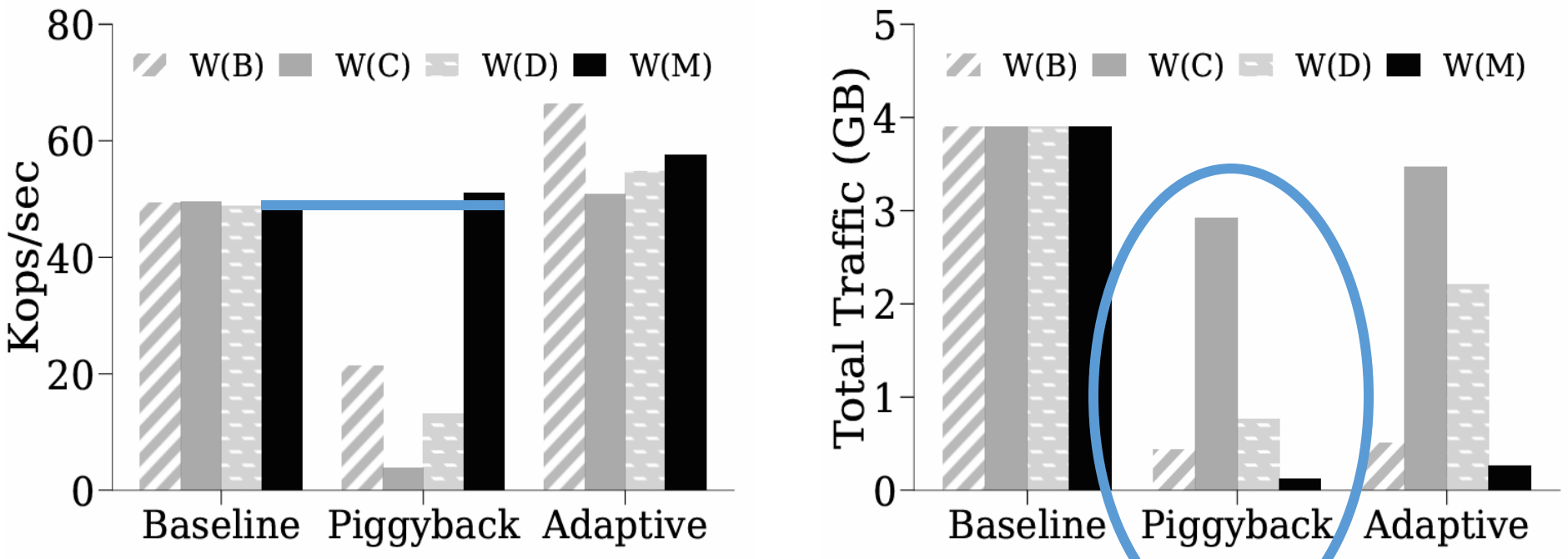
(a) Avg. Throughput (b) Total PCIe Traffic

Figure 2. Performance analysis of transfer methods.

Fine-Grained Value Transfer

Various Workloads (*W(B) ~ W(M)*)

- *Piggyback (BandSlim)* significantly reduced total PCIe traffic across all workloads and improved the average throughput compared to *Baseline (NVMe PRP)* for *W(M)*, which reflects real-world value size characteristics of RocksDB.



(a) Avg. Throughput (b) Total PCIe Traffic

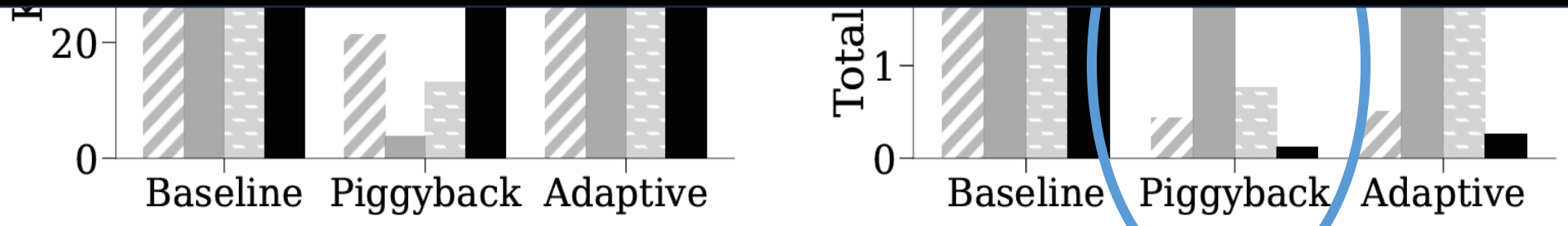
Figure 2. Performance analysis of transfer methods.

Fine-Grained Value Transfer

Various Workloads (*W(B) ~ W(M)*)

- *Piggyback (BandSlim)* significantly reduced total PCIe traffic across all workloads and improved the average throughput compared to *Baseline (NVMe PRP)* for *W(M)*, which reflects real-world value size characteristics of RocksDB.

The proposed approach performs better than the baseline under real-world workloads while reducing PCIe traffic significantly.



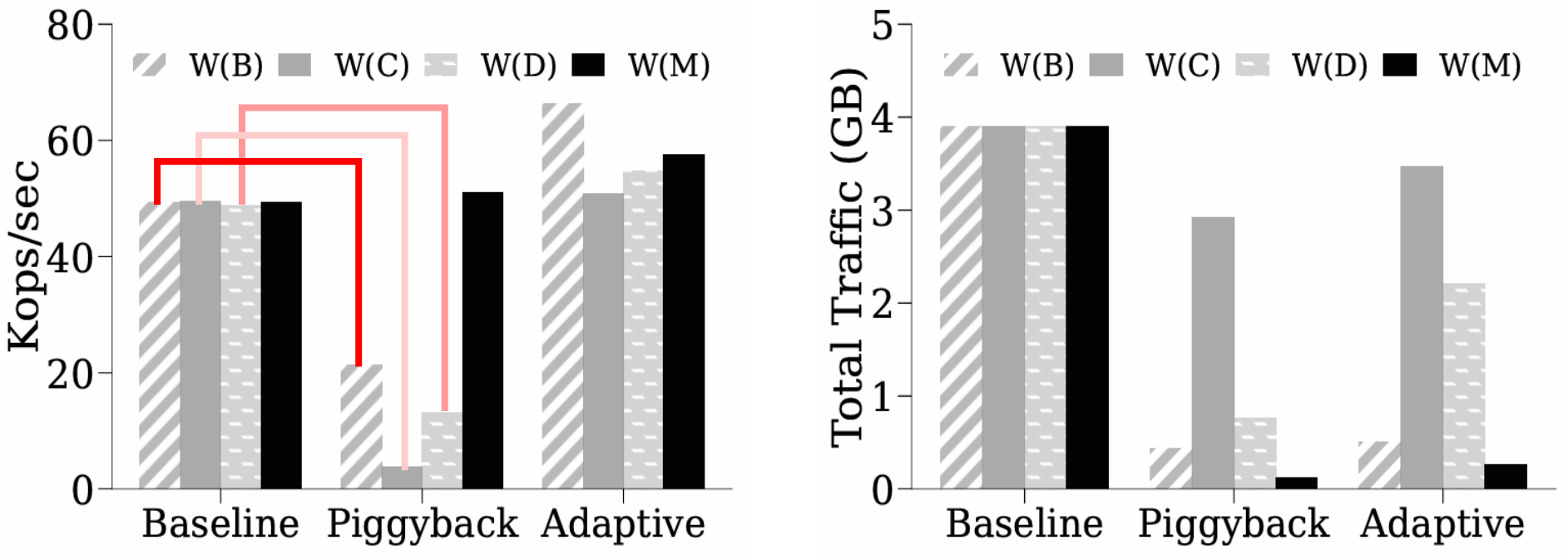
(a) Avg. Throughput (b) Total PCIe Traffic

Figure 2. Performance analysis of transfer methods.

Fine-Grained Value Transfer

Various Workloads (*W(B) ~ W(M)*)

- However, *Piggyback (BandSlim)* suffered significant performance degradation for ***W(****B*), ***W(****C*), and ***W(****D*) since its serialized NVMe command generation and processing for transferring larger values (64B~).
- Above all, *Adaptive* proves to be the best in all workloads.



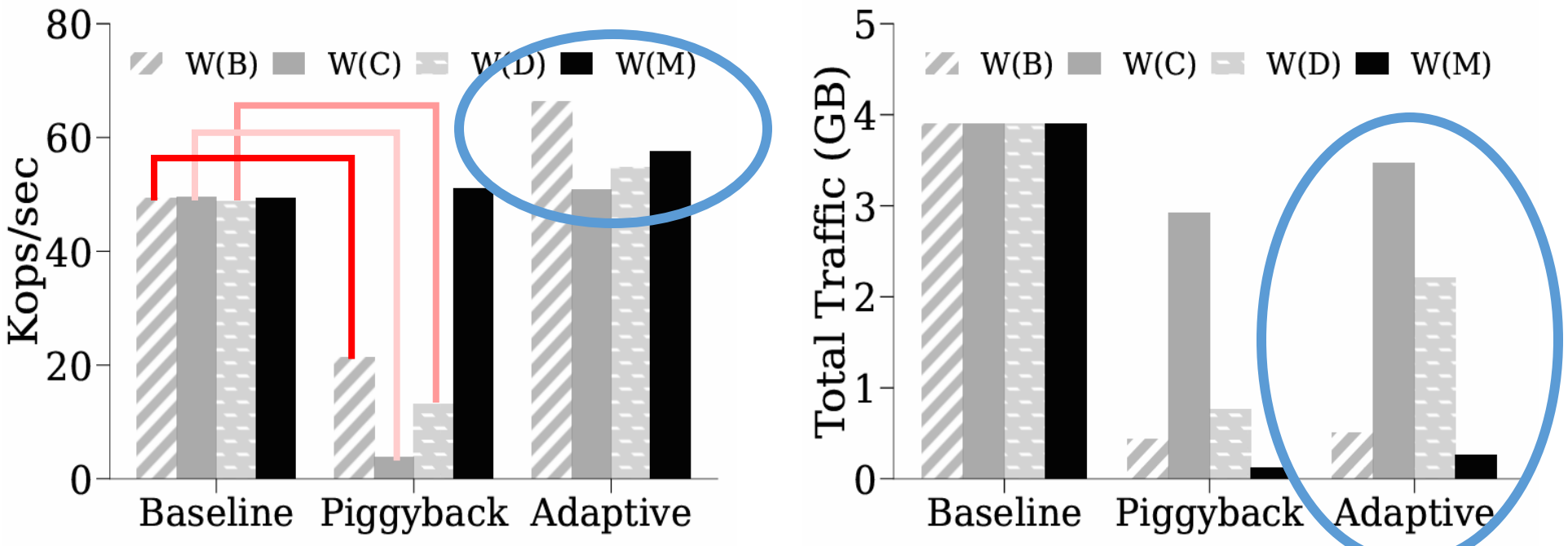
(a) Avg. Throughput (b) Total PCIe Traffic

Figure 2. Performance analysis of transfer methods.

Fine-Grained Value Transfer

Various Workloads (*W(B) ~ W(M)*)

- However, *Piggyback (BandSlim)* suffered significant performance degradation for ***W(****B*), ***W(****C*), and ***W(****D*) since its serialized NVMe command generation and processing for transferring larger values (64B~).
- Above all, *Adaptive* proves to be the best in all workloads.



(a) Avg. Throughput (b) Total PCIe Traffic

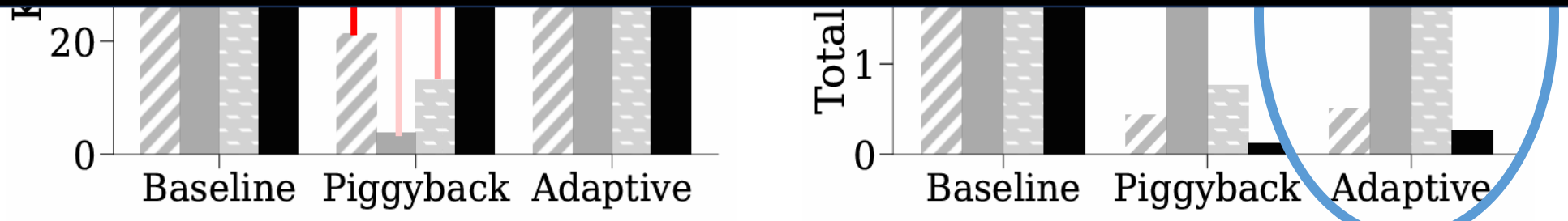
Figure 2. Performance analysis of transfer methods.

Fine-Grained Value Transfer

Various Workloads (*W(B) ~ W(M)*)

- However, *Piggyback (BandSlim)* suffered significant performance degradation for ***W(****B*), ***W(****C*), and ***W(****D*) since its serialized NVMe command generation and processing for transferring larger values (64B~).
- Above all, *Adaptive* proves to be the best in all workloads.

If we cover tiny values by piggybacking, and large values by fast PRP DMA, we can achieve an optimal transfer performance.



(a) Avg. Throughput (b) Total PCIe Traffic

Figure 2. Performance analysis of transfer methods.

Fine-Grained Value Transfer

Various Workloads (*W(B) ~ W(M)*)

- However, *Piggyback (BandSlim)* suffered significant performance degradation for ***W(****B*), ***W(****C*), and ***W(****D*) since its serialized NVMe command generation and processing for transferring larger values (64B~).

But BandSlim clearly has a limitation of suffering performance degradation for moderately large payloads (e.g., 64B ~ 256B) which are also common in other real-world deployments

(Twitter reported their average KV pair size is around 100B [9] *)*

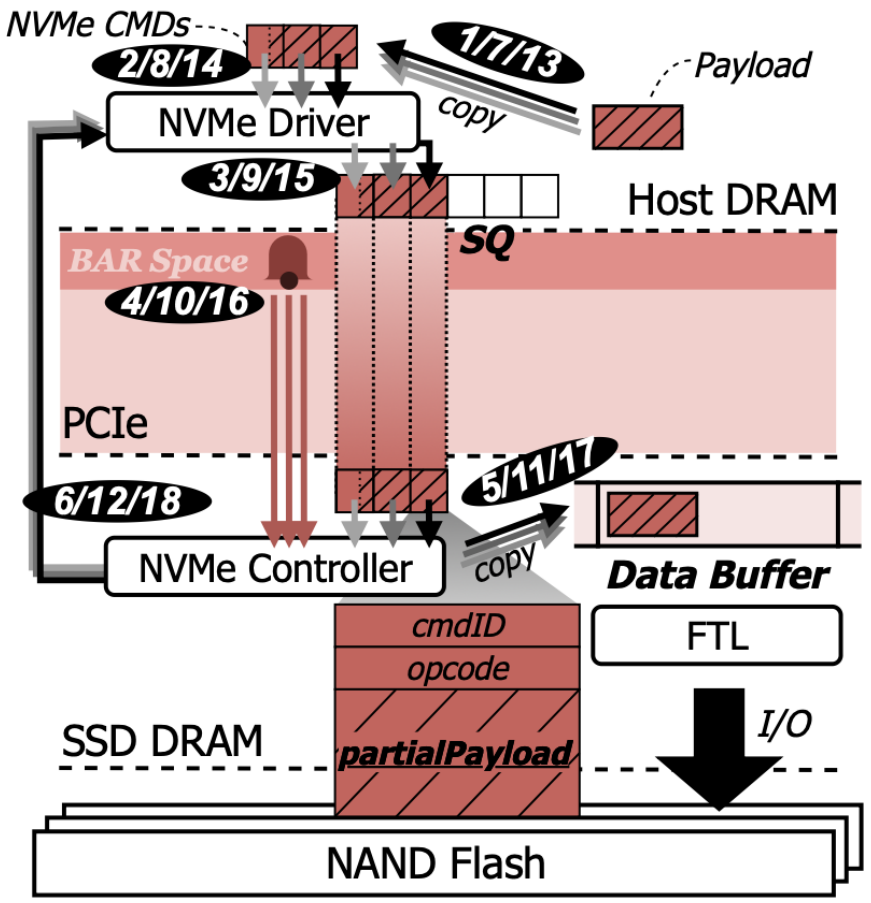
[9] W. Daelemans et al., "Overview of PAN 2019: Bots and Gender Profiling, Celebrity Profiling, Cross-Domain Authorship Attribution and Style Change Detection". In Experimental IR Meets Multilinguality, Multi- modality, and Interaction. Proceedings of the 10th International Conference of the CLEF Association (CLEF 2019)

(a) Avg. Throughput (b) Total PCIe Traffic

Figure 2. Performance analysis of transfer methods.

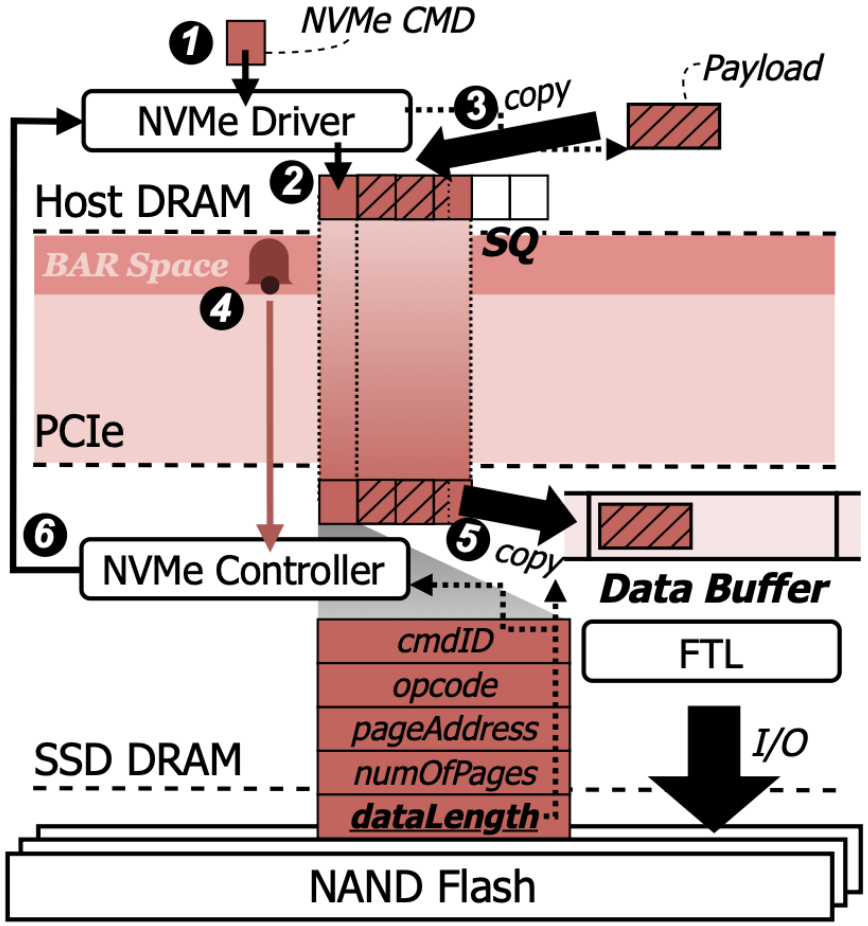
ByteExpress's Fine-Grained Payload Transfer

- The previous approach embeds payloads (values) in custom NVMe commands and achieves a fine-grained payload transfer.
  - However, it loses scalability for moderately larger payloads due to repeated NVMe command generation and processing caused by its mandatory serialization for data integrity of payloads.



ByteExpress's Fine-Grained Payload Transfer

- ***ByteExpress*** introduces a new method to transmit small payloads without modifying the NVMe interface or SSD architecture.
  - It appends 64-byte payload chunks directly after the command in the Submission Queue, avoiding PRP-based DMA inefficiencies and serialized command processing overheads.
  - The technical details are skipped since it's now under review.



ByteExpress's Fine-Grained Payload Transfer

- ***ByteExpress*** introduces a new method to transmit small payloads without modifying the NVMe interface or SSD architecture.
  - It ap
    after
    avoi
    seria
  - The
    since

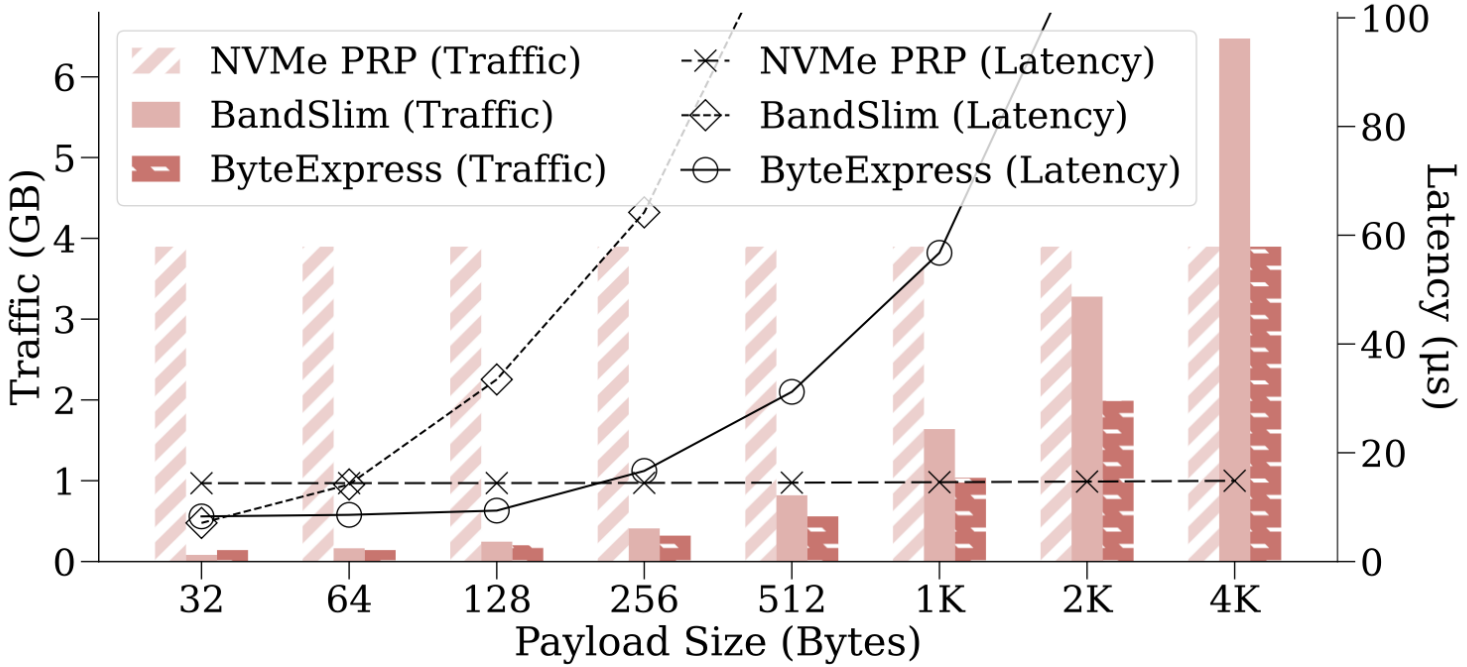
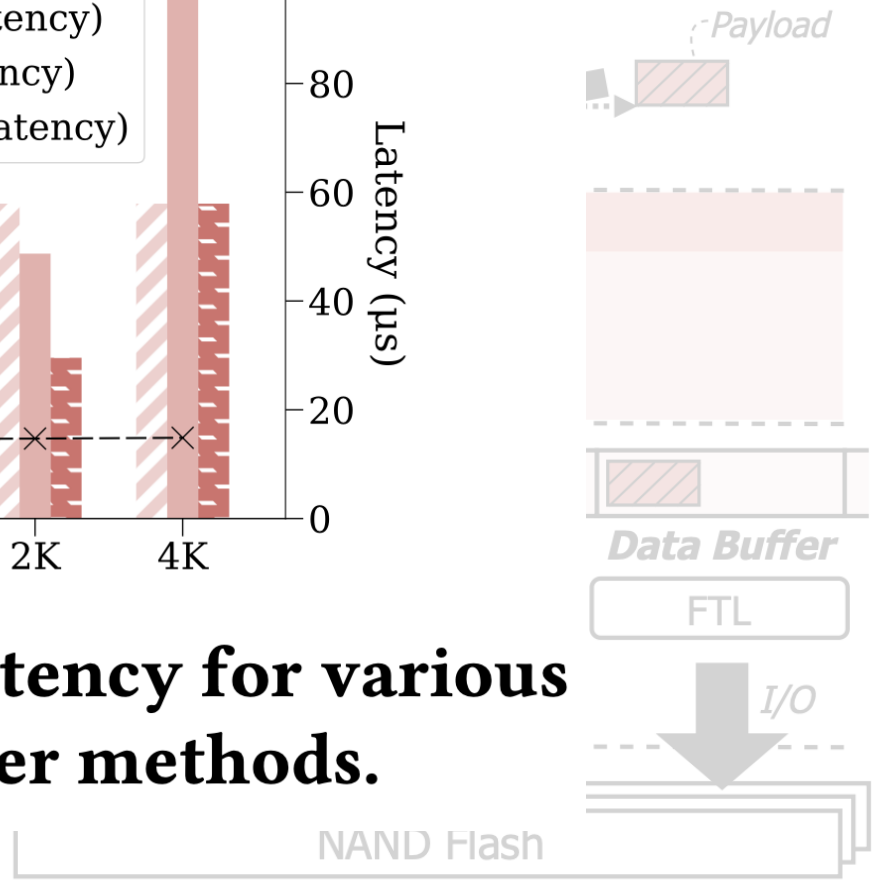


Figure 5: PCIe traffic and average latency for various payload sizes across different transfer methods.



* Workload: FillRandom with fixed sized payloads (32B to 4KB)

ByteExpress's Fine-Grained Payload Transfer

- ***ByteExpress*** introduces a new method to transmit small payloads without modifying the NVMe interface or SSD architecture.
  - It ap
    after
    avoi
    seria
  - The
    since

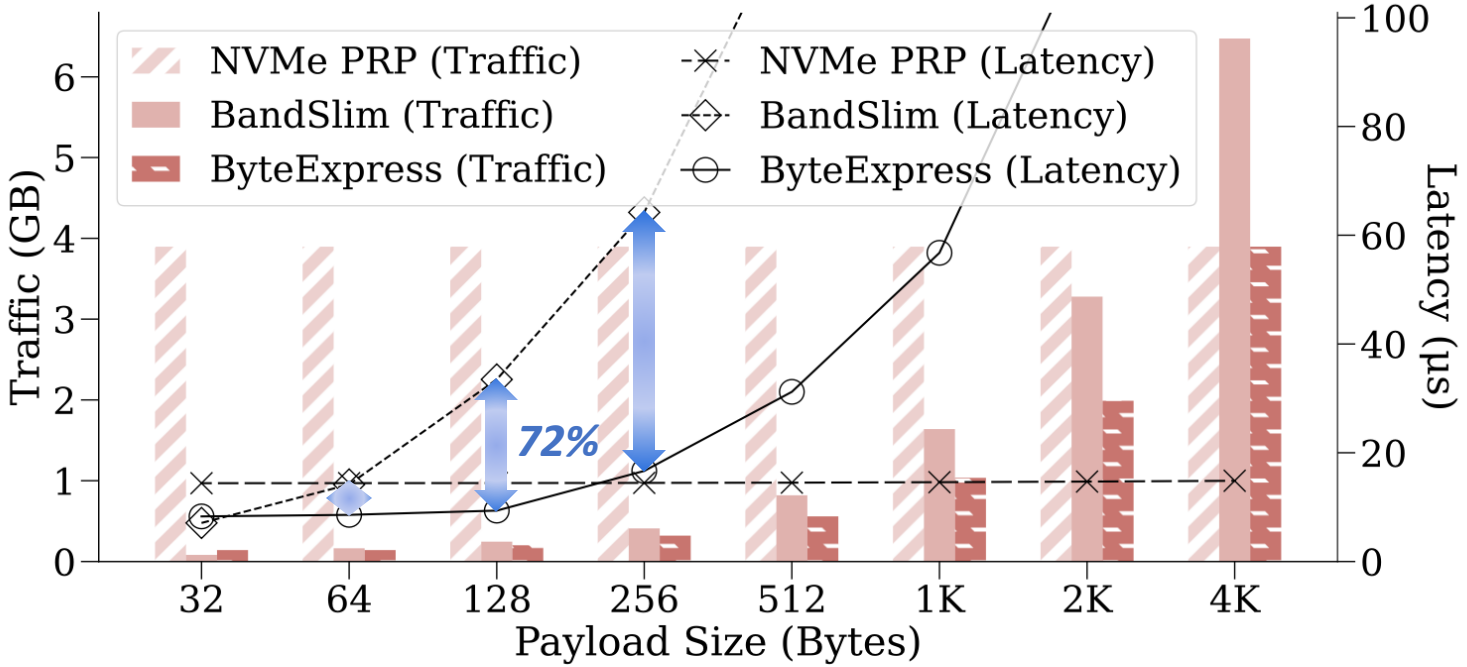
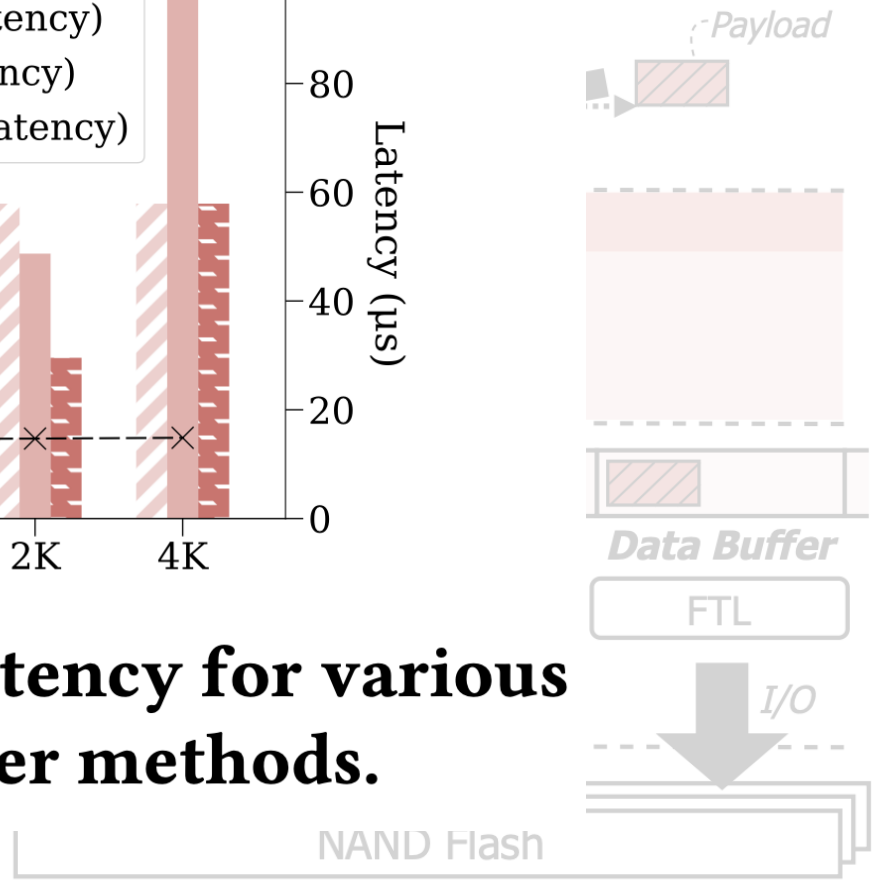


Figure 5: PCIe traffic and average latency for various payload sizes across different transfer methods.



* Workload: FillRandom with fixed sized payloads (32B to 4KB)

ByteExpress's Fine-Grained Payload Transfer

- ***ByteExpress*** introduces a new method to transmit small payloads without modifying the NVMe interface or SSD architecture.

With ByteExpress's NVMe SQ-Based Fine-Grained Transfer, we can efficiently cover not just tiny values but also moderately larger values (64B ~ 256B) which are frequent in real-world scenarios

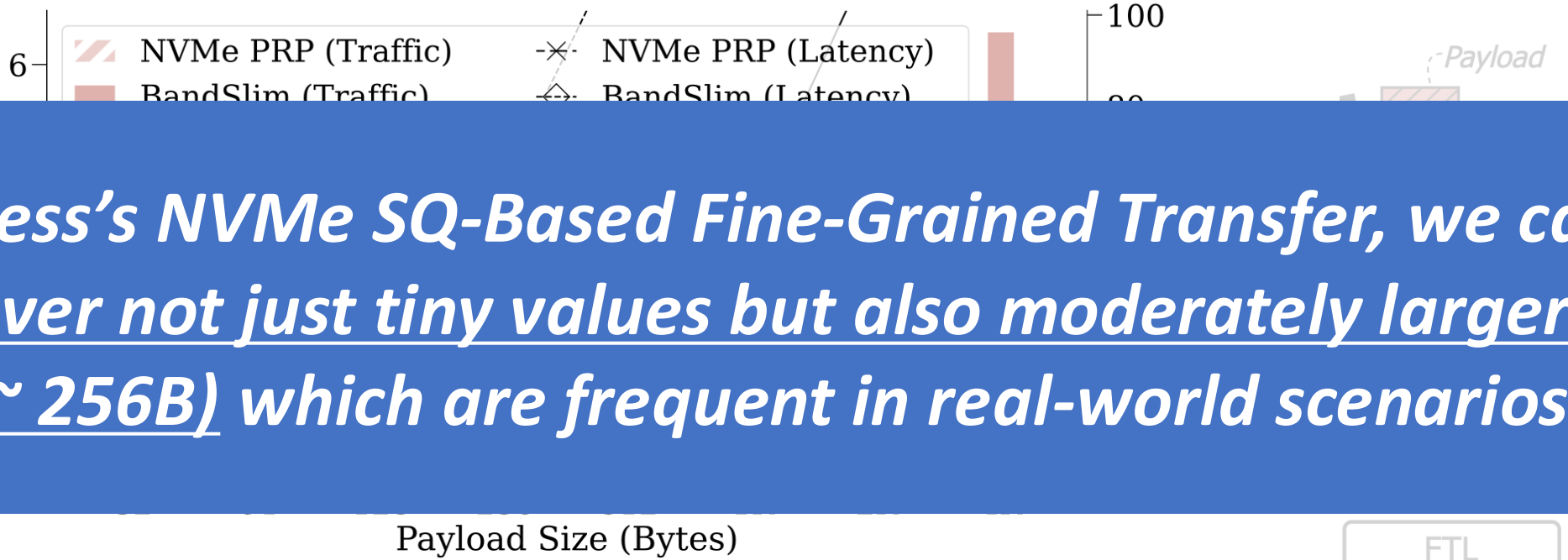
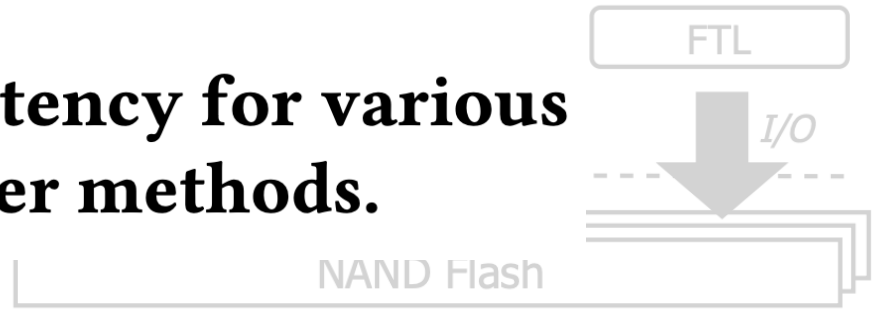


Figure 5: PCIe traffic and average latency for various payload sizes across different transfer methods.



* Workload: FillRandom with fixed sized payloads (32B to 4KB)

Conclusion

Conclusion

We introduce ***BandSlim*** and ***ByteExpress*** to address inefficiencies of traditional block-interfaced storage protocols (e.g., NVMe) for small payload transfers, especially in a domain of Key-Value SSDs.

The mismatch leads to excessive traffic on the PCIe interconnect and increased transfer latency, significantly degrading performance.

Our proposed solutions reduce PCIe traffic greatly and improve performance as well for small payloads-dominant scenarios.

DISCOS
Data-Intensive Computing & Systems Laboratory

IHS

Leveraging KVSSDs to Overcome Performance Hurdles in Key-Value Stores

- KVAccel: A Novel Write Accelerator for LSM-Tree-Based KV Stores with Host-SSD Collaboration, IPDPS, June 2025 (To Appear)

Background

LSM-Tree-Based Key-Value Stores

- LSM KVS (ex. RocksDB) stores data in an append-only manner in the active MemTable
- Data in MemTable is moved to and managed on disk through background jobs (Flush, Compaction)

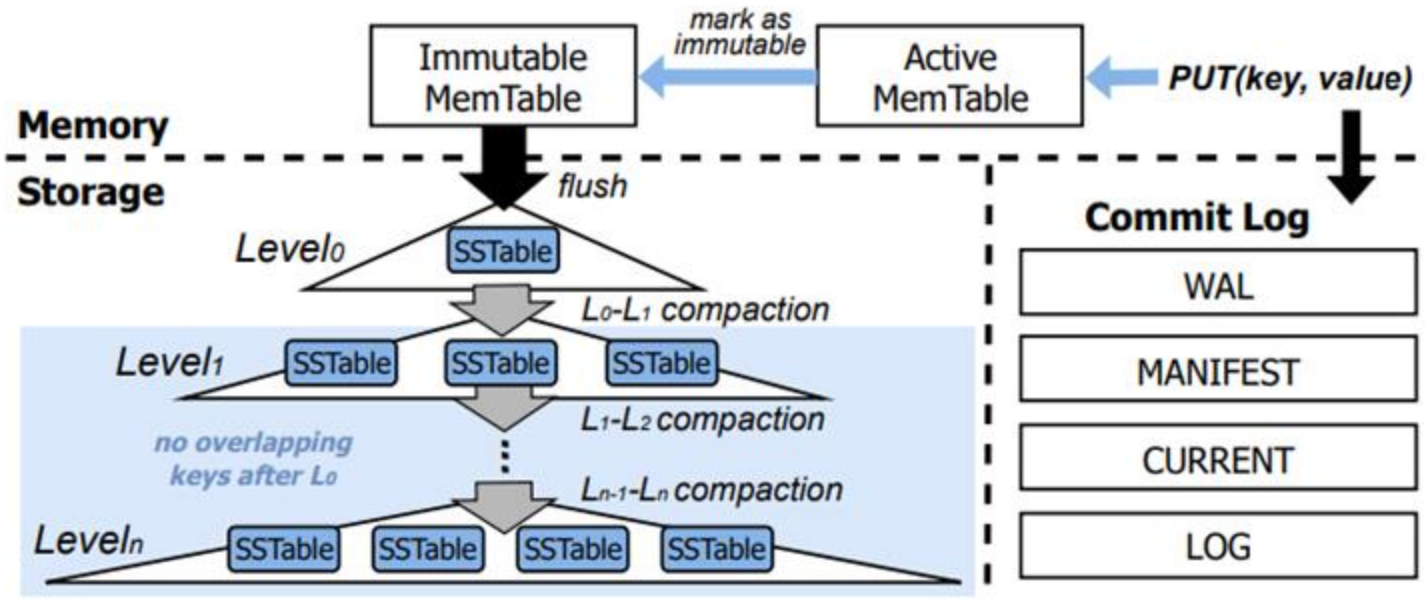


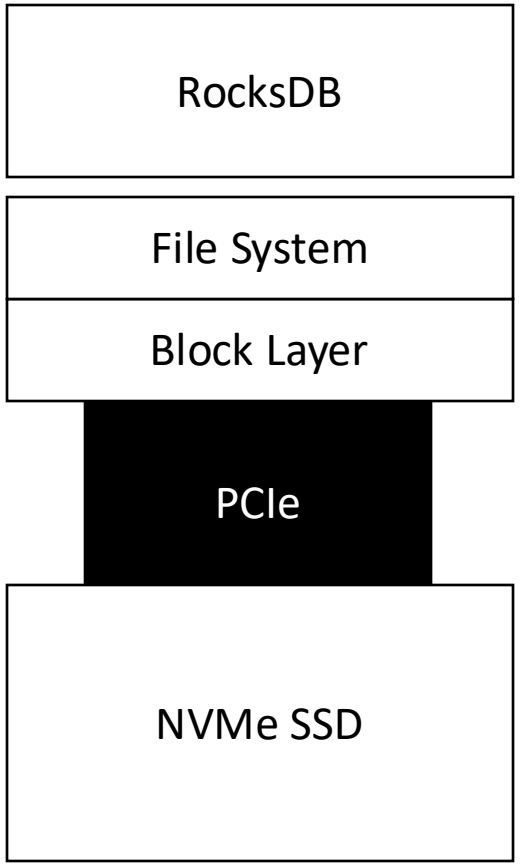
Fig. 1: An architecture of LSM-tree.

Write Stall Problem

- **Write Stall:** write operation blocked, due to bottlenecks in Flush, Compaction
- In RocksDB, write stall occurs under these 3 scenarios
  1) Incoming Writes > Flush
  2) Flush > Level 0 to Level 1 Compaction
  3) Pending deep level compaction size becomes heavier

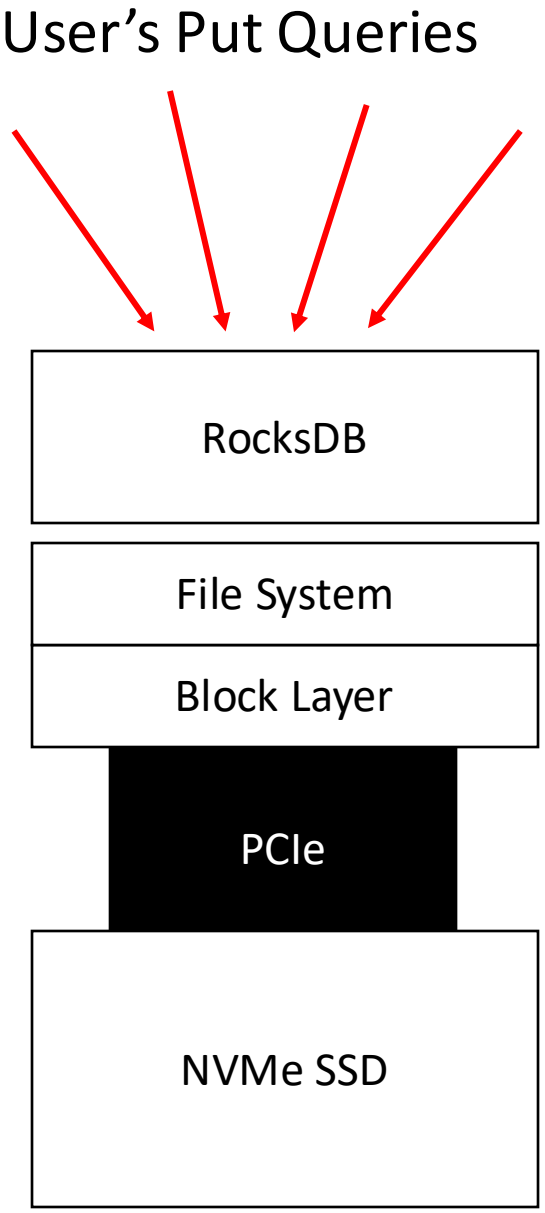
Motivation

Motivation



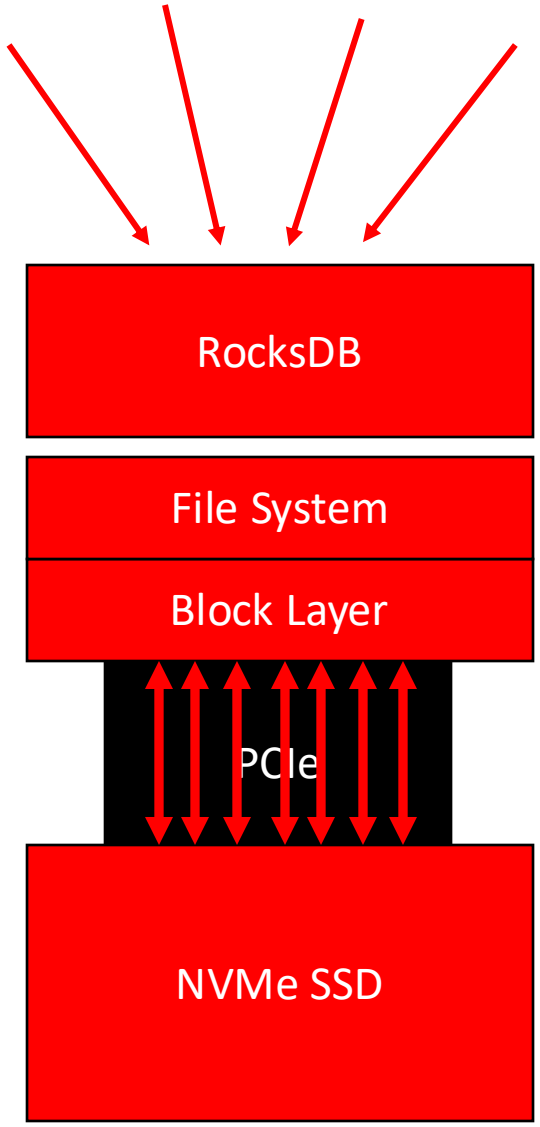
Motivation


RocksDB


User’s Put Queries
RocksDB
File System
Block Layer
PCIe
NVMe SSD

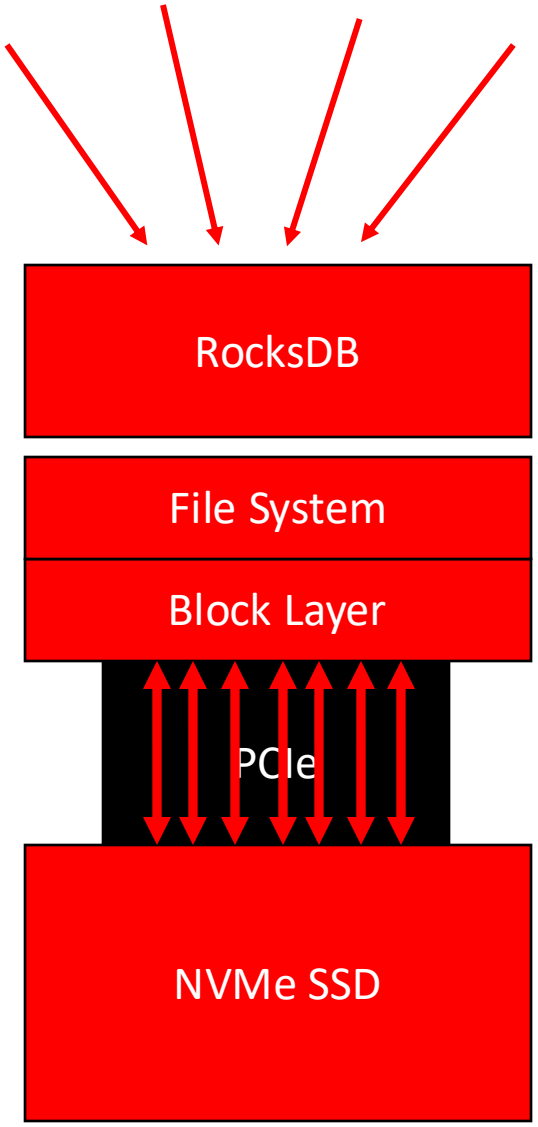
Motivation

RocksDB



Motivation

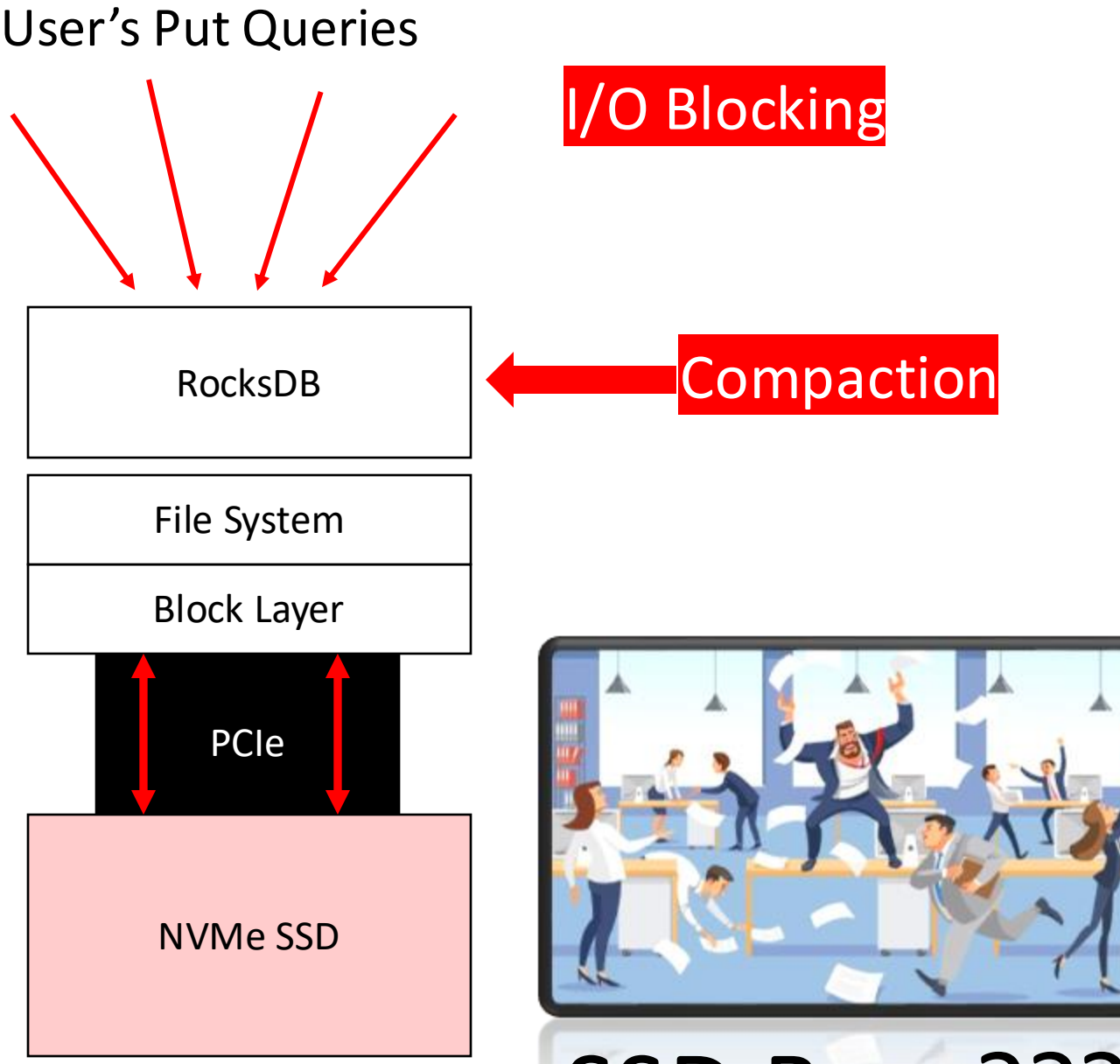
RocksDB



Compaction

Motivation

RocksDB



SSD Busy????

Slowdowns[1]: Inefficient Write Stall Solution

Observation #1

- RocksDB uses the *slowdown* method to prevent user writes from becoming completely blocked.
- The state-of-the-art solution ADOC[2] also uses *slowdowns*.

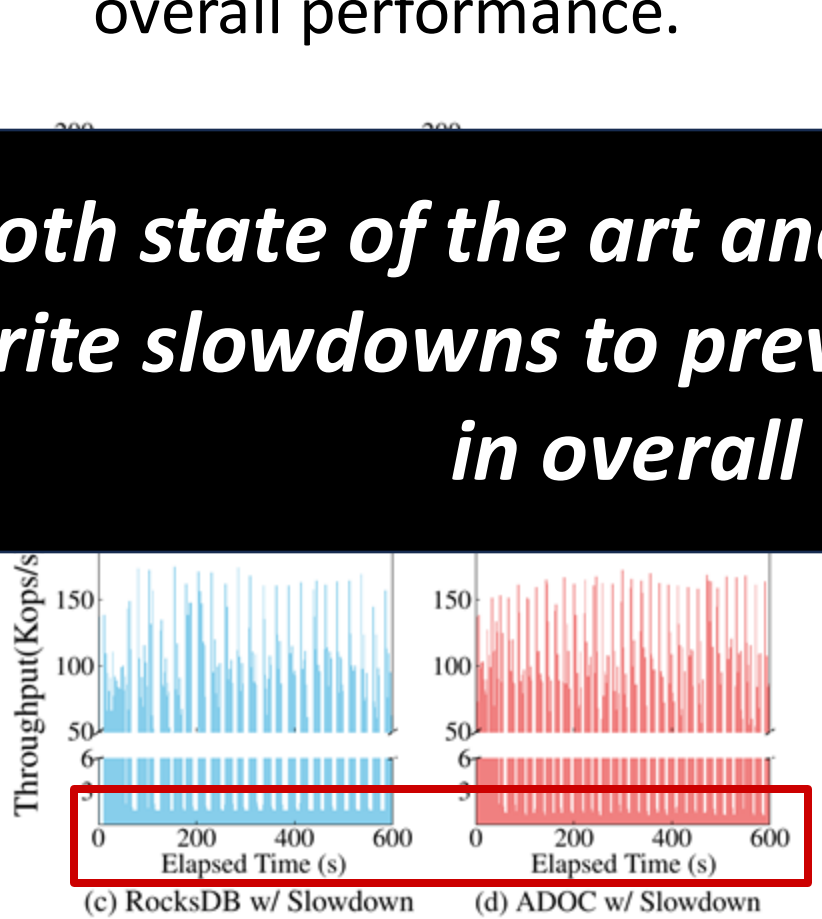
➡ Both RocksDB and ADOC[2] ultimately fall back to using *slowdown* to avoid a write stall.

[1]:https://github.com/facebook/rocksdb/wiki/Write-Stalls
[2]: ADOC: Automatically Harmonizing Dataflow Between Components in Log-Structured Key-Value Stores for Improved Performance, Jinghuan Yu et al. (USENIX FAST'23)

Slowdowns[1]: Inefficient Write Stall Solution

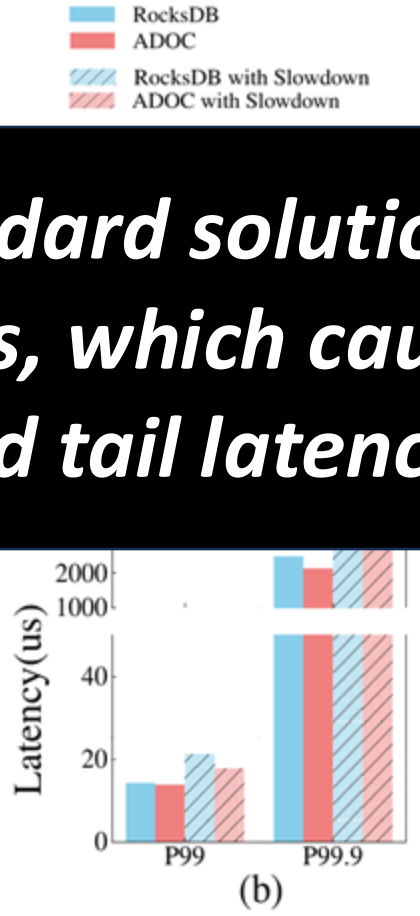
Observation #1

- *Slowdowns*, while preventing a complete write stall from occurring, harms overall performance.



Both state of the art and industry standard solutions make use of write slowdowns to prevent write stalls, which cause a sharp drop in overall throughput and tail latency.

preventing write stalls…



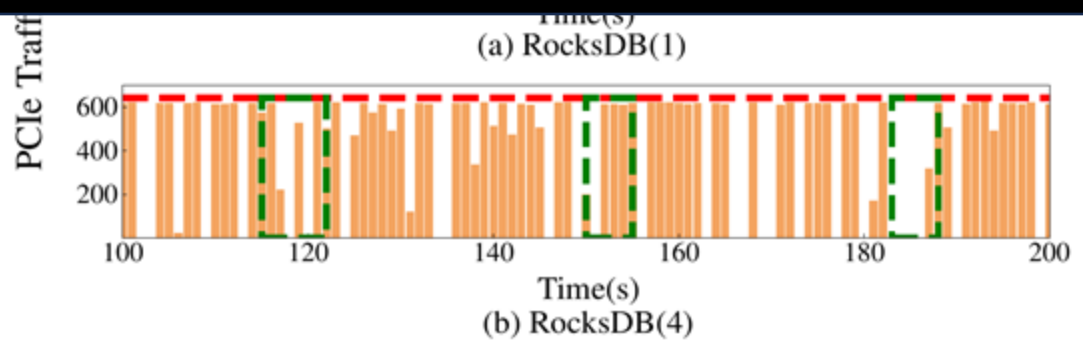
latency.

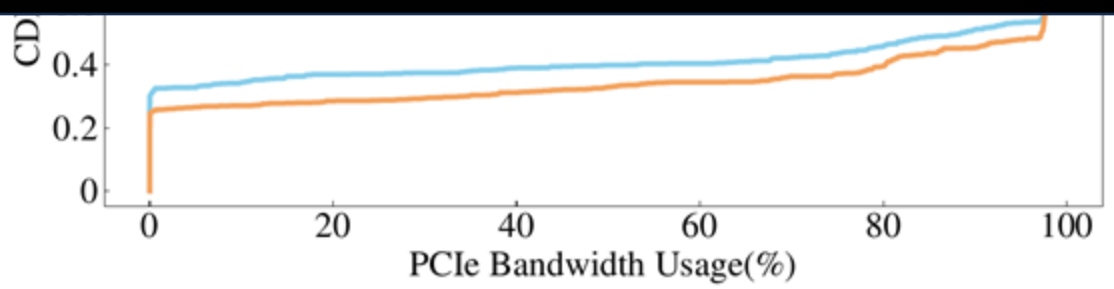
Under-utilization of PCIe Bandwidth

Observation #2

- PCIe Traffic drop sharply during a write stall, implying inefficient device resource usage.
  - RocksDB is shown to leave up to 90% of available PCIe bandwidth around

PCIe bandwidth is under-utilized during write stalls in industry standard LSM-KVS due to the compaction operation blocking device I/O.





The status quo

- **Observation 1.** ultimately leads to the following options for write stalls.

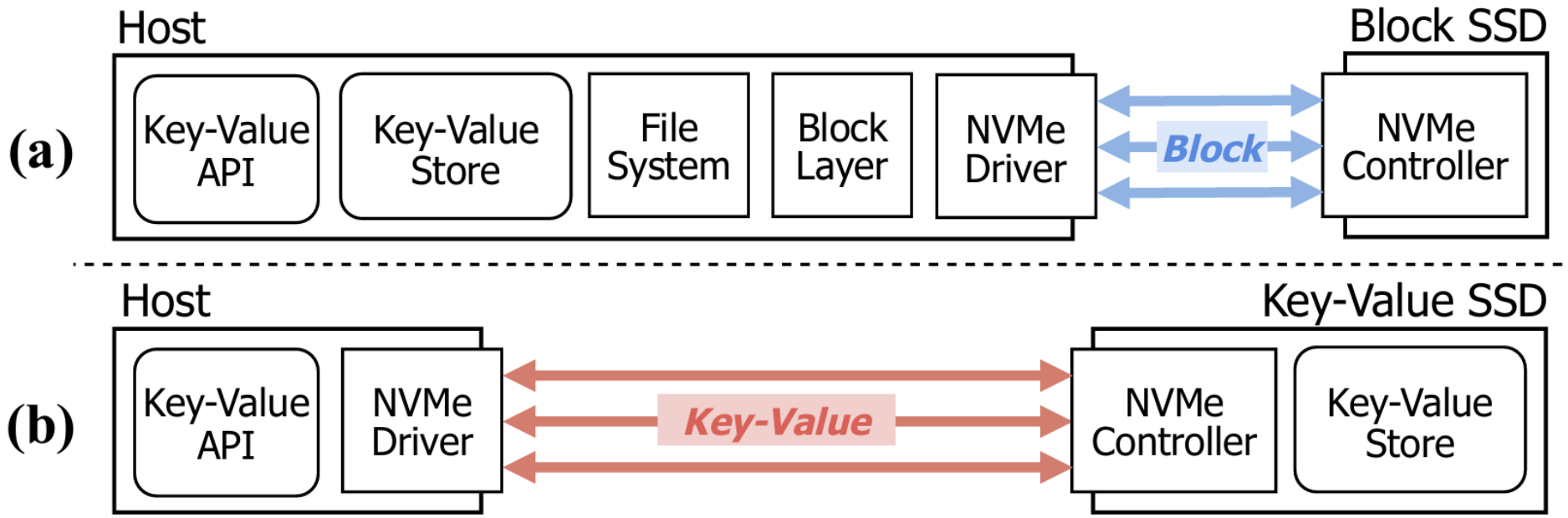
Slowdowns **Allowing Write Stalls**

Can write stalls be mitigated without sacrificing system resources by leveraging underutilized PCIe and device bandwidth during write stalls?

- **Observation 2.** reveals an unexploited resource to help mitigate write stalls and increase performance without sacrificing system resources: underutilized PCIe and device bandwidth during write stalls.

Utilizing Key-Value Interface of KV-SSD ?

- We can leverage the available bandwidth and processing capacity of SSD during write stalls in LSM-KVSs by temporarily redirecting pending write requests through the **key-value interface**.



Proposed Solution: *KVAccel*

Proposed Solution: ***KVAccel***

- ***KVAccel*** ’s design is based on two key factors:
  ➔ Disaggregation and Aggregation.

Disaggregation

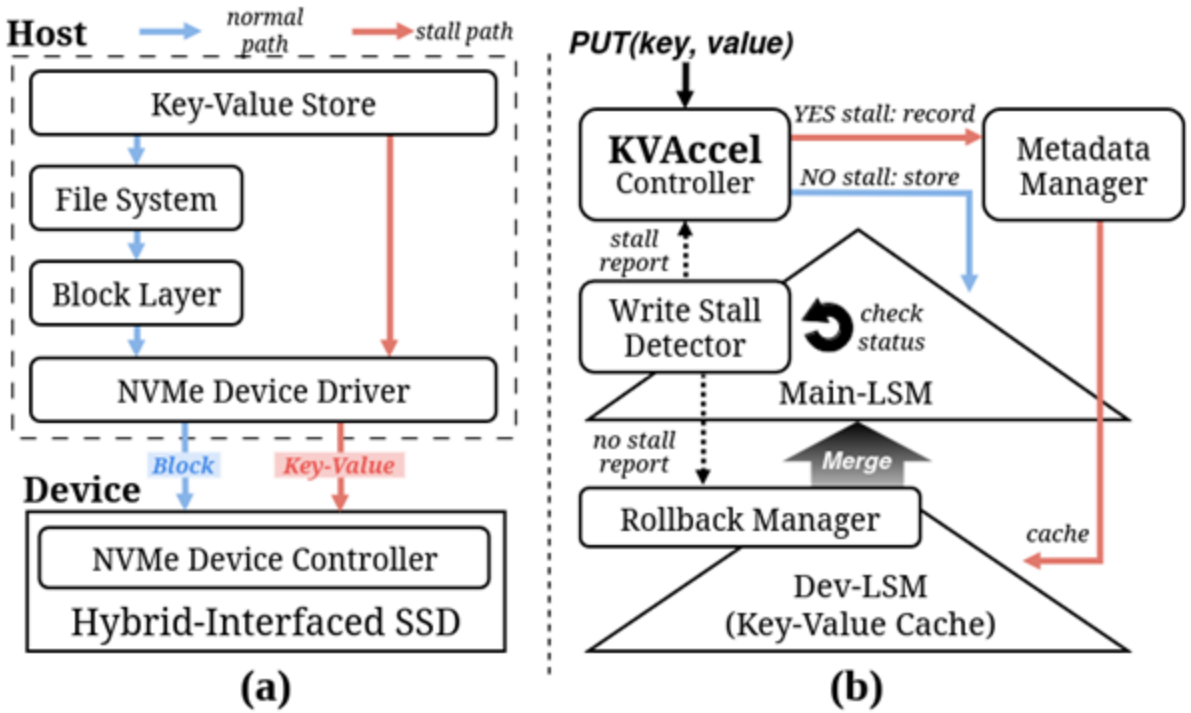
- Division of SSD into hybrid interface (block and key-value) and its required I/O paths
- Maintenance of each interface’s separate LSM-Tree

Aggregation

- Manage data from each interface as if it was one database instance
- Unify separate I/O commands and database state with rollback

Overview of ***KVAccel***

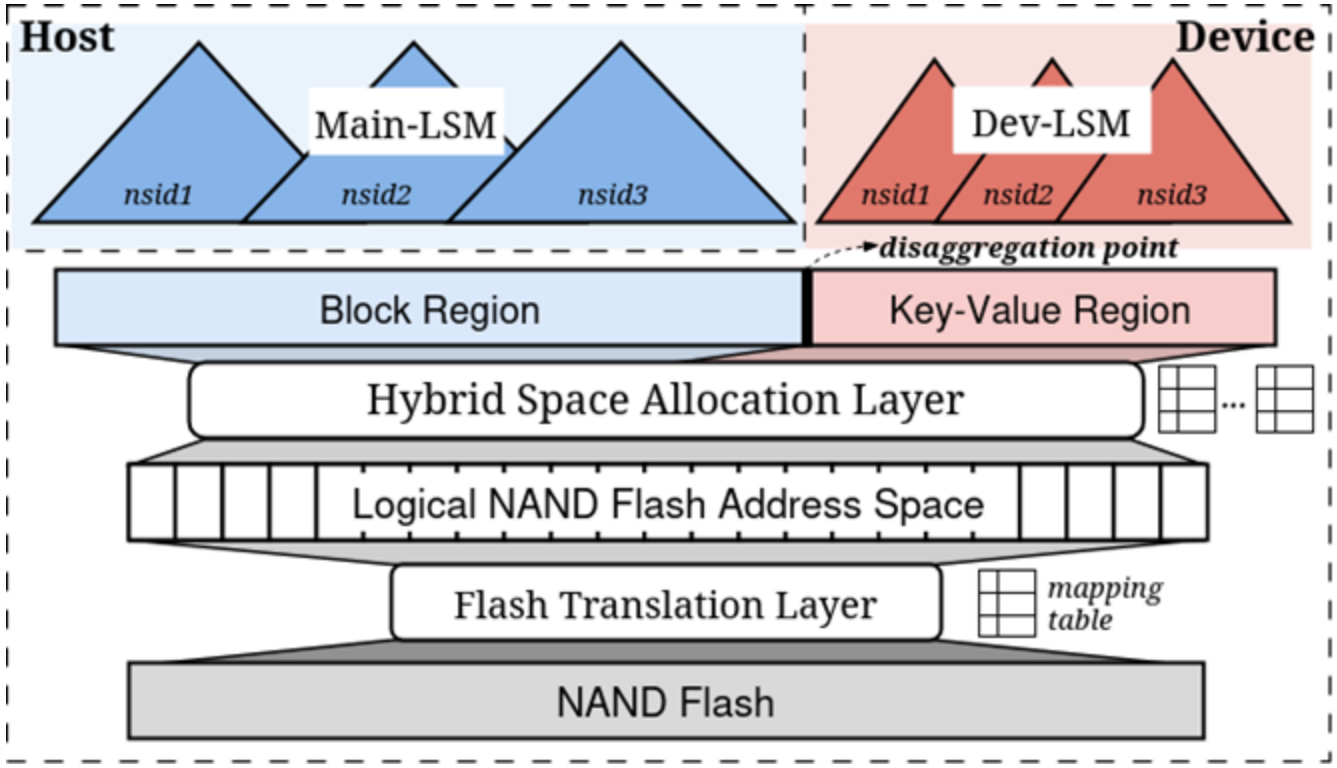
- Co-design of hardware & software provides two I/O paths
- Different I/O paths taken based on the presence of a write stall



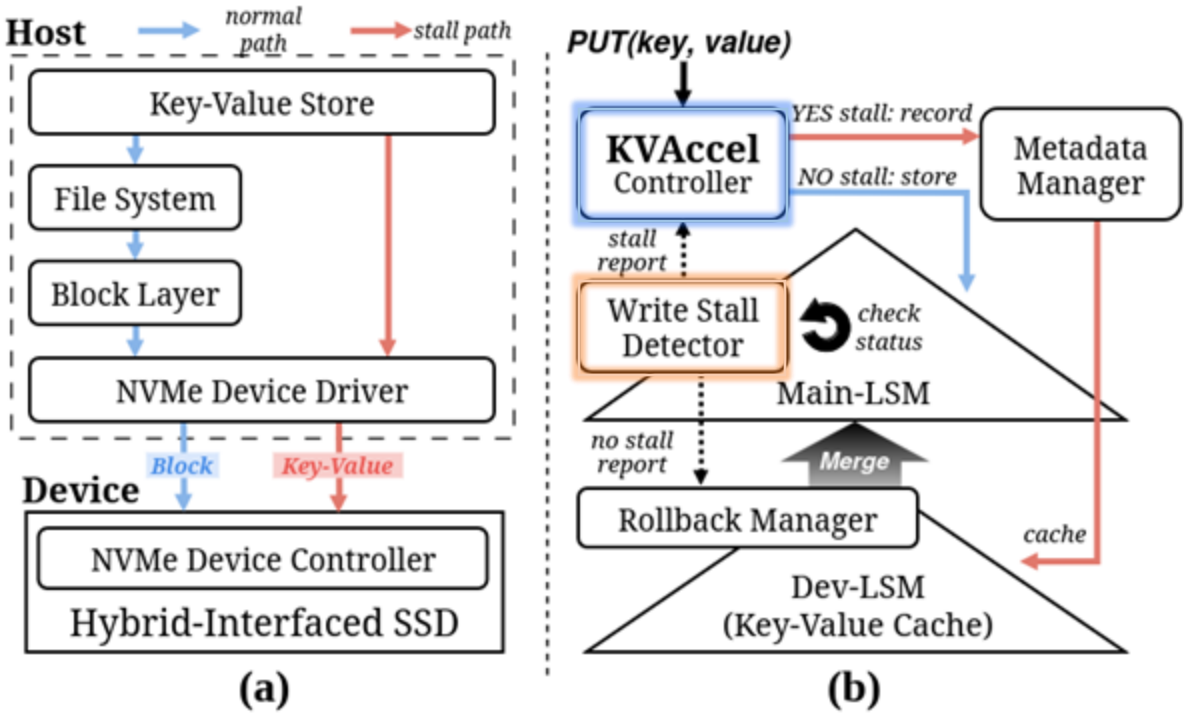
- **Main-LSM**
  - Block interface's LSM-Tree
  - Main-LSM I/O path taken when no write stalls are present.
- **Dev-LSM**
  - Key-value interface's LSM-Tree
  - Dev-LSM I/O path taken a write stall is occuring.

Hardware – Hybrid Dual-Interface SSD

- Hybrid interface SSD achieved by logical NAND flash address **disaggregation** via a specified address boundary.
  - Each interface is unaware of the division ➔ **no risk of overlapping** logical NAND pages.
  - SSD issues **different commands** for each interface based on the given NVMe opcode.

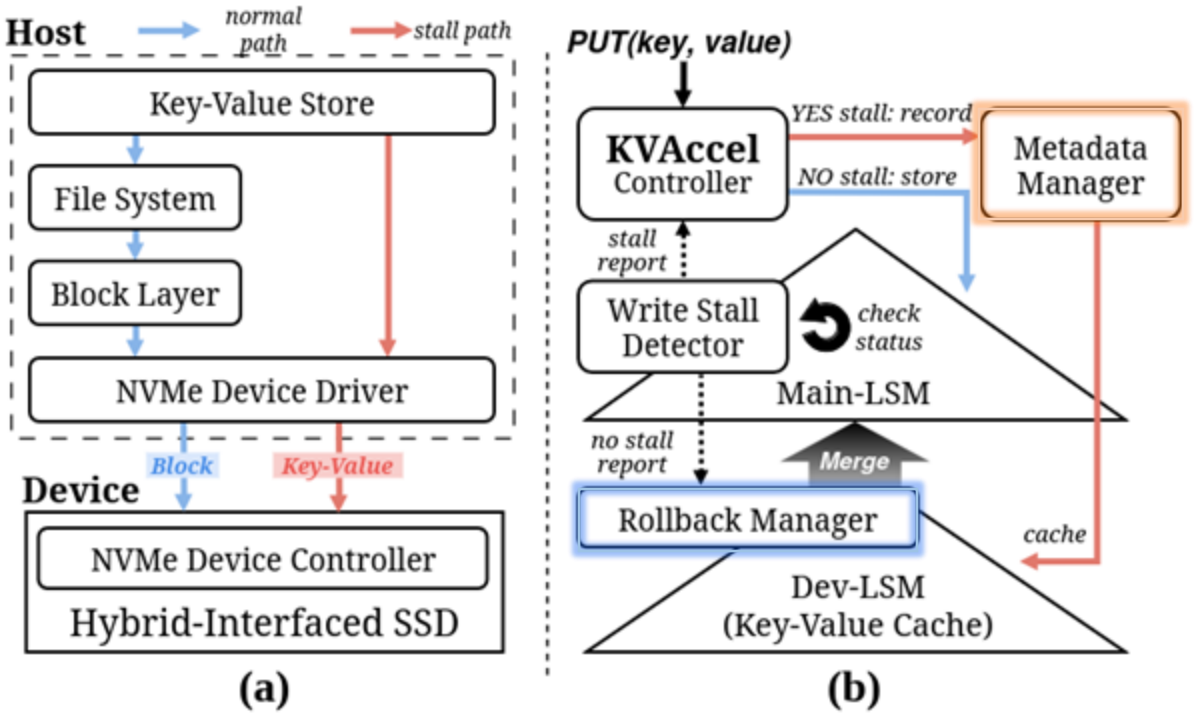


Software – Detector & Controller



- **Detector**
  - Detects write stalls checking 3 components
    - # of Level 0 SSTs
    - MemTable size
    - Pending compaction size
- **Controller**
  - Directs I/O commands to the correct interface based on the Detector's output.

Software – Metadata & Rollback Manager



- **Metadata Manager**
  - Keeps track of KV pairs located in Dev-LSM via a hash table for membership testing
- **Rollback Manager**
  - Initiates and performs the rollback operation based on the rollback scheduling policy and the Detector's output.

Rollback Operation: Scheduling

- Rollback refers to return the KV pairs in Dev-LSM back to Main-LSM into one LSM-KVS instance.
- Rollback operation can be scheduled *eagerly* or *lazily* based on workload characteristics.

Eager Rollback

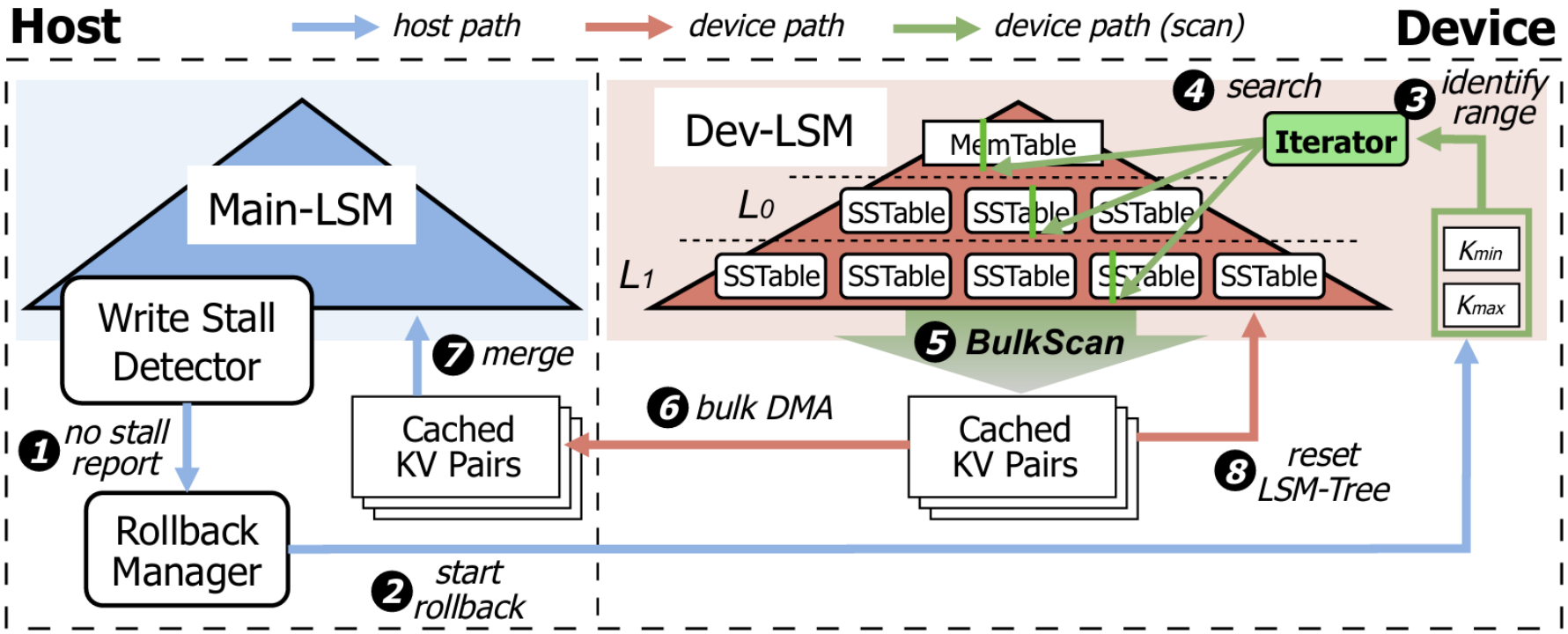
- Perform rollback as soon as there are enough resources available (by using L_0 file count threshold)
- Ideal for a read orientated workload to avoid slow Dev-LSM read operations

Lazy Rollback

- Delay rollback until the current write workload is completely finished
- Ideal for a write intensive workload to lower interference of rollback with write operations

Rollback Operation: Range Scan

- To accelerate rollback, KV pairs are read in bulk using a range scan operation.
  - Iterator reads Dev-LSM in its entirety and serializes the KV pairs.
  - KV pairs are then sent to the host by performing DMA multiple times.



Evaluation

Evaluation Setup

- Testbed:

KV-SSD on Cosmos+ OpenSSD Platform

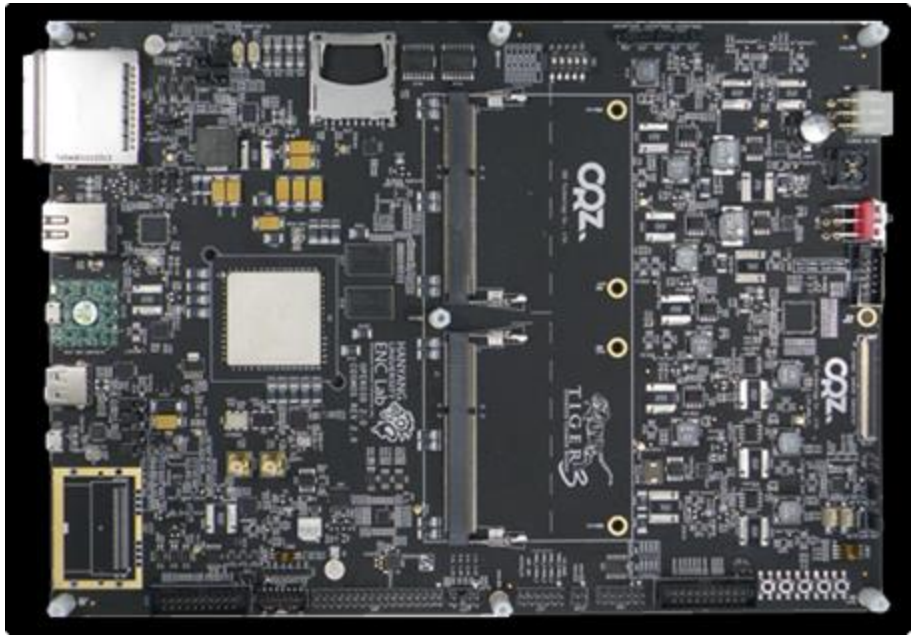


TABLE I: Specifications of the OpenSSD platform.

SoC	Xilinx Zynq-7000 with ARM Cortex-A9 Core
NAND Module	1TB, 4 Channel & 8 Way
Interconnect	PCIe Gen2 ×8 End-Points

TABLE II: Specifications of the host system.

CPU	Intel(R) Xeon(R) Gold 6226R CPU @ 2.90GHz (32 cores), CPU usage limited to 8 cores.
Memory	384GB DDR4
OS	Ubuntu 22.04.4, Linux Kernel 6.6.31

LSM-KVS and Benchmark Configurations

TABLE III: LSM-KVS configurations. For all figures, the numbers next to each LSM-KVS refer to compaction thread count. For KVACCEL, the settings refer to the Main-LSM.

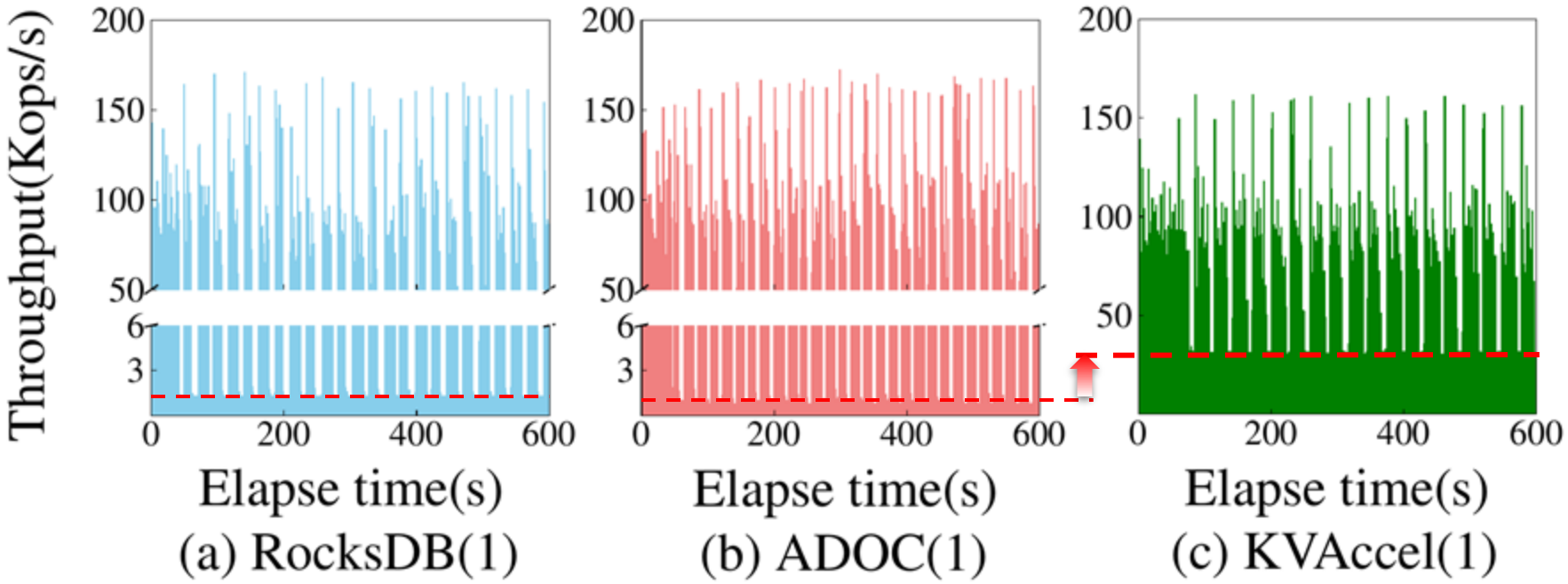
LSM-KVS	Compaction Threads (n)	MT Size
KVACCEL(n)	1	128 MB
	2	
	4	
RocksDB(n)	1	
	2	
	4	
ADOC(n)	1	
	2	
	4	

TABLE IV: *db_bench* workload configurations. Each benchmark was run with a 4 B key and 4 KB value size. Workload A,B,C were run for 600 seconds, and Workload D performed 60K read operations.

Name	Type	Characteristics	Notes (write/read ratio)
A	fillrandom	1 write thread	No write limit
B	readwhilewriting	1 write thread + 1 read thread	9:1
C			8:2
D	seekrandom	1 range query thread (Seek + 1024 Next)	Run after initial 20GB fillrandom

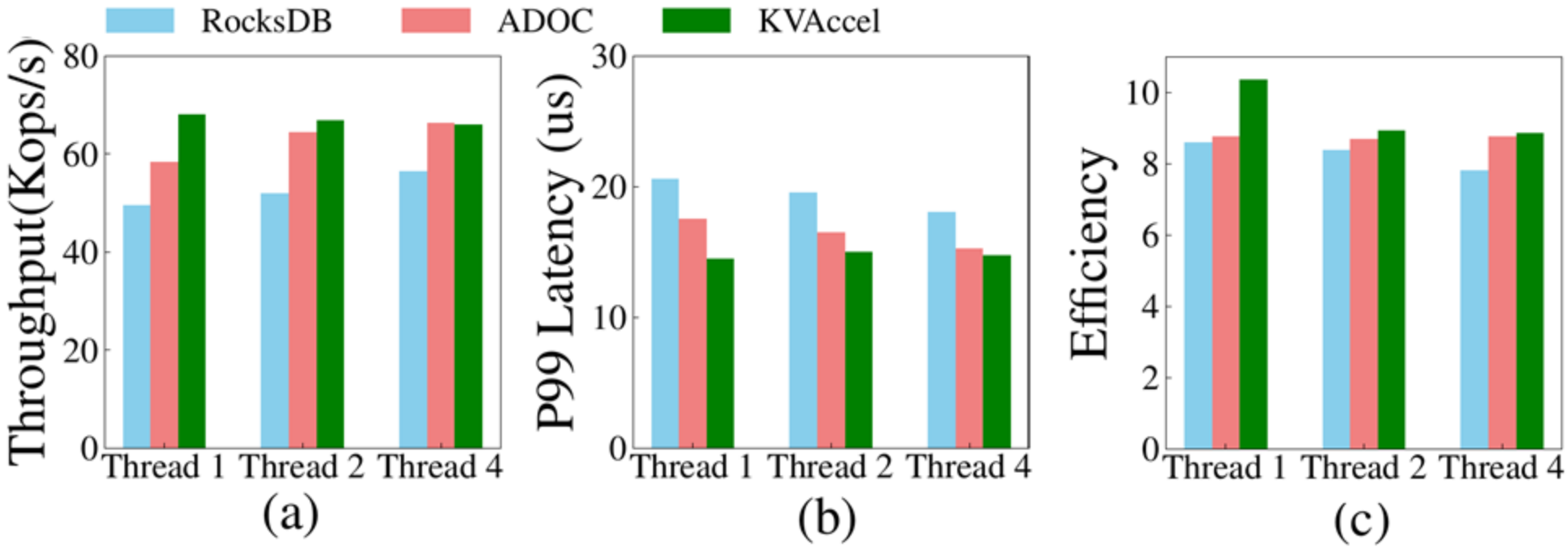
Write Stall Avoidance

- Throughput minimum values greatly increased, as ***KVAccel*** is designed to allow as much throughput as the SSD and system allows **without slowdowns**.



Performance Evaluation

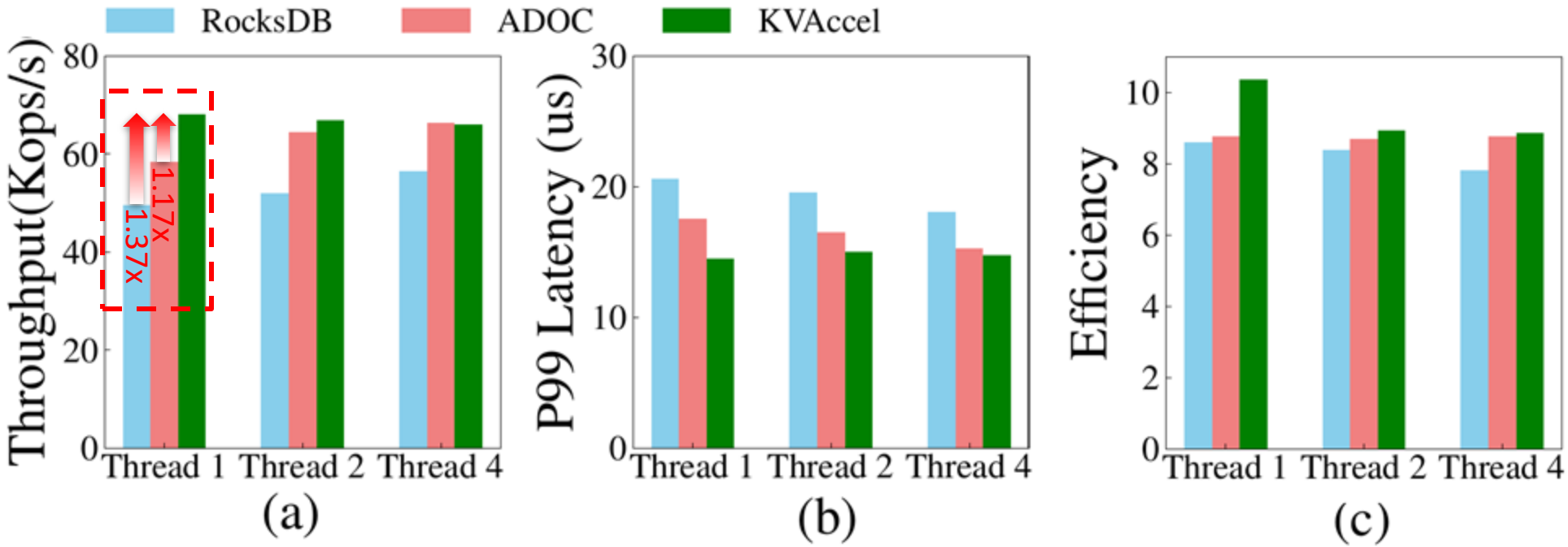
(a) Throughput **(b) P99 Latency** **(c) Efficiency**



Performance Evaluation

(a) Throughput

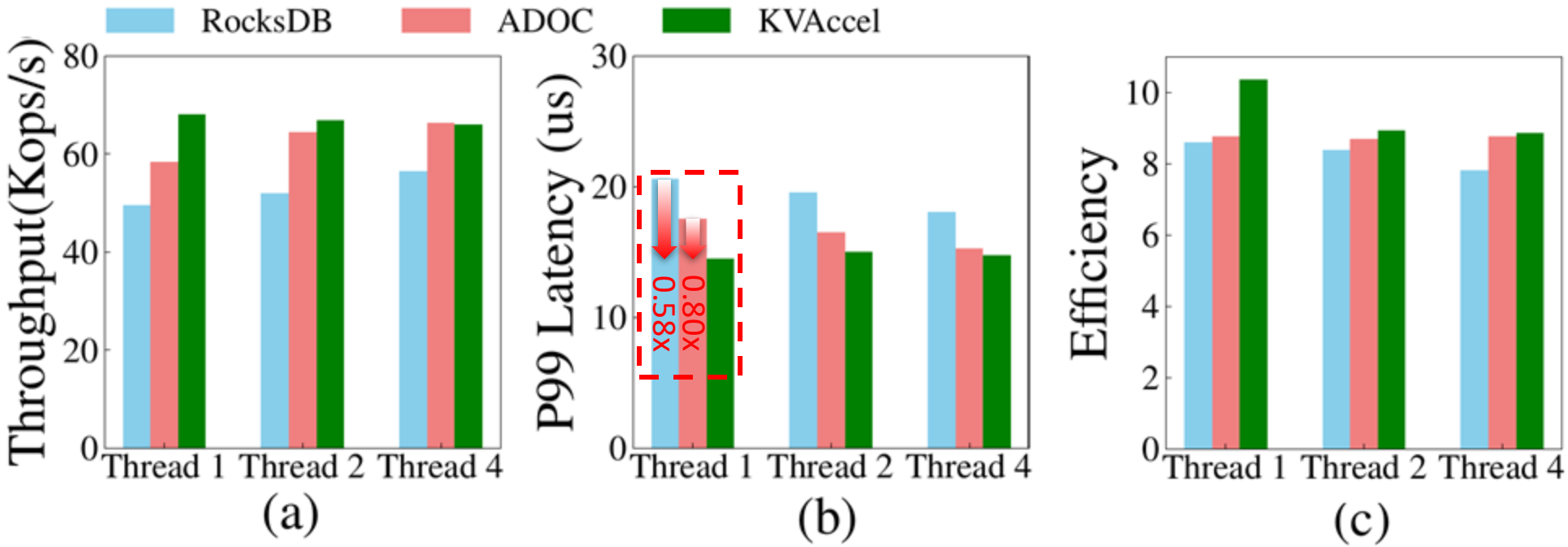
- ***KVAccel*** shows at most a 37% and 17% improvement over than RocksDB and ADOC, respectively.



Performance Evaluation

(b) P99 Latency

- Maximum of 42% and 20% decrease in latency was also observed between ***KVAccel*** and RocksDB, ADOC, respectively.

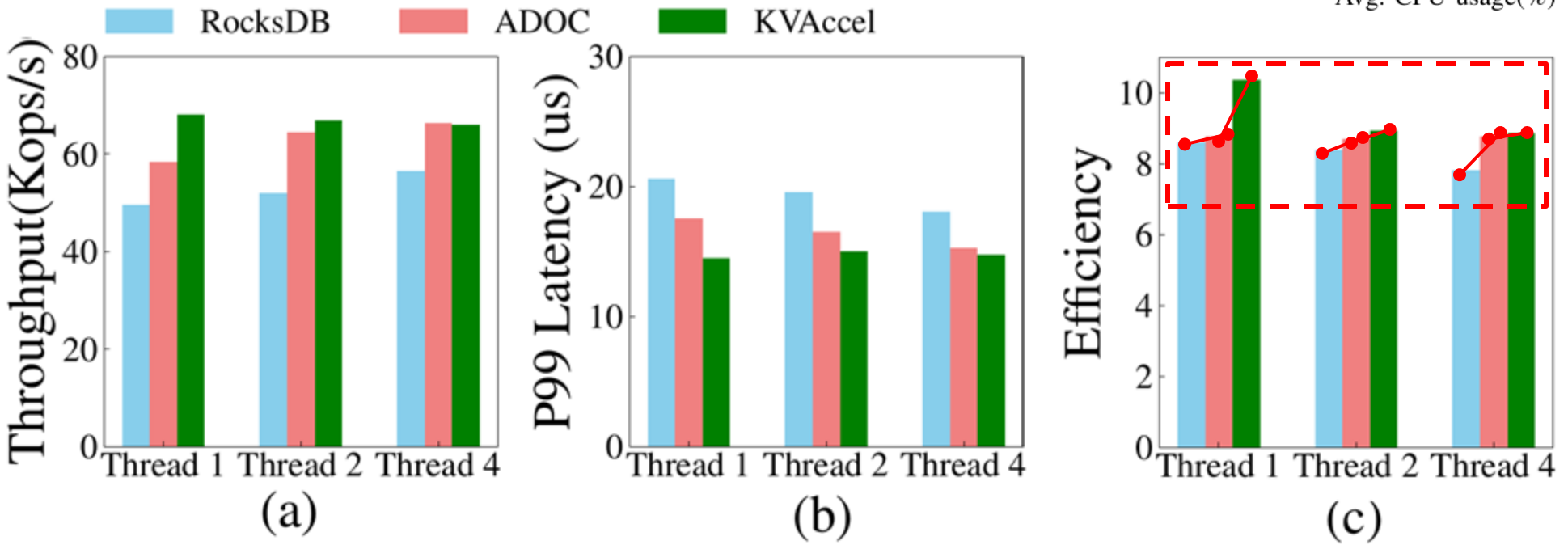


Performance Evaluation

(c) Efficiency = Avg. Throughput(MB/s) / Avg. CPU usage(%)

- ***KVAccel*** maintains the better efficiencies in host machine's resources between all LSM-KVS compared, with ***KVAccel***(1) shows the best efficiency over all configurations.

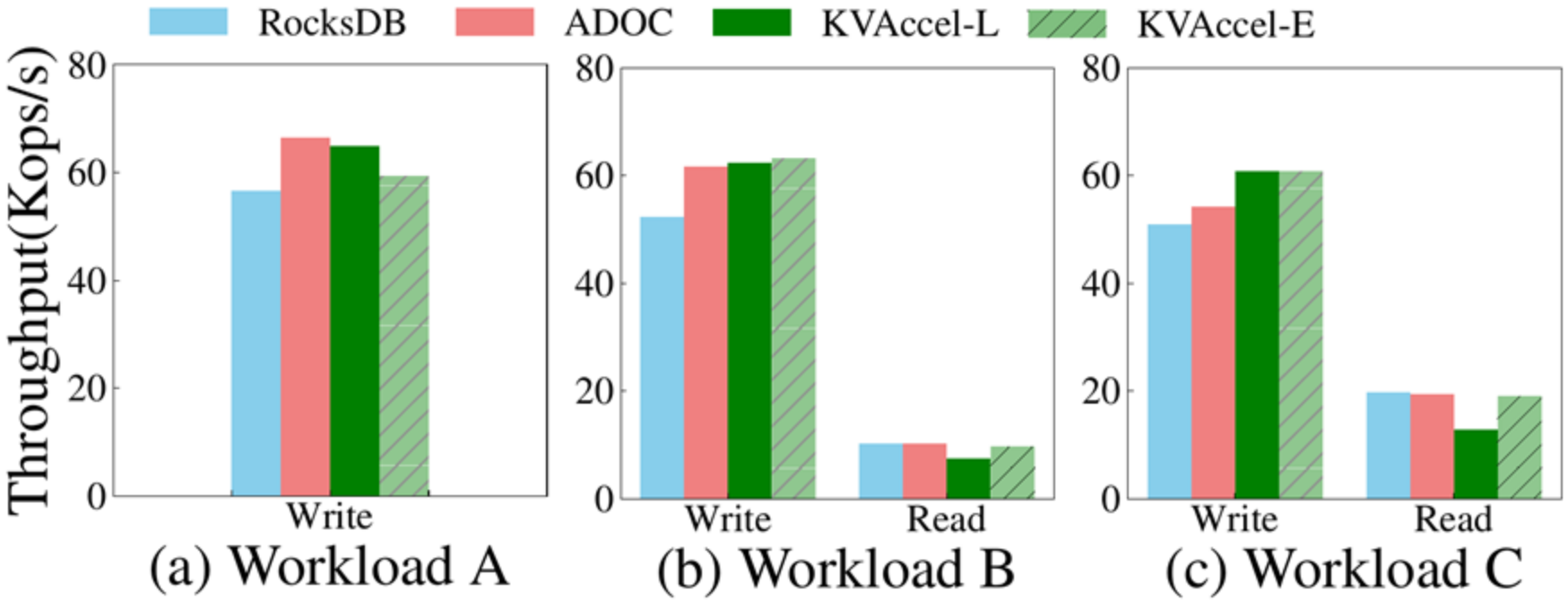
$$\text{Efficiency} = \frac{\text{Avg. Throughput(MB/s)}}{\text{Avg. CPU usage(\%)}}$$



Rollback Policies Evaluation

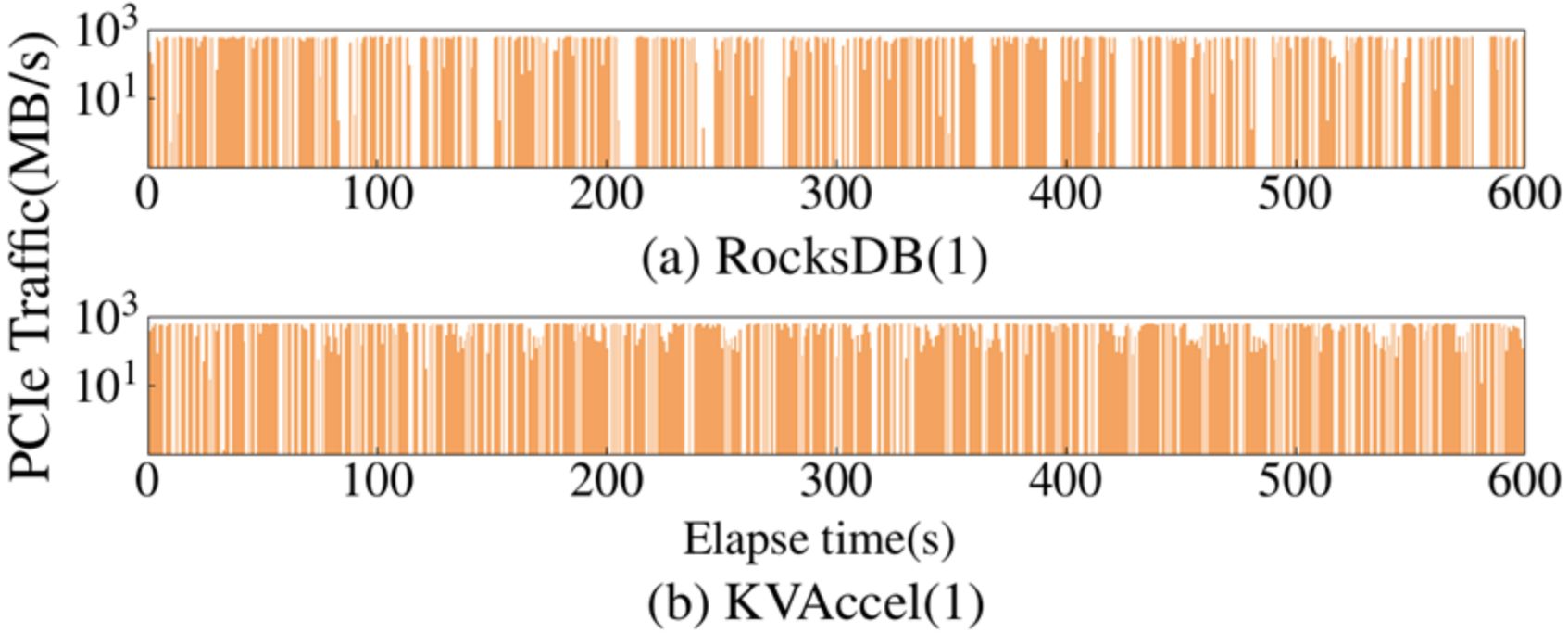
Eager vs Lazy Rollback analysis

- (b) and (c) present a read-write mix workload, where both rollback schemes achieve similar write throughput, both holding a lead of 36% and 51% over ADOC respectively.



PCIe Traffic Usage

- More available PCIe traffic exploited
- ***KVAccel*** takes advantage of its dual interface and demonstrate higher PCIe utilization over RocksDB.



Conclusion

Conclusion

- Current research on mitigating write stalls fall short in completely eliminating write stalls, while failing to incorporate both hardware and software holistically in their design.

- ***KVAccel*** demonstrates the effectiveness of hardware-software co-design for write stall mitigation by exploiting underutilized PCIe bandwidth and computational power of SSDs during compaction.

Thank You

Q&A

Presenter: Youngjae Kim

Contact: youkim@sogang.ac.kr