ByteExpress: A High-Performance and Traffic-Efficient Inline Transfer of Small Payloads over NVMe

Junhyeok Park, Junghee Lee, and Youngjae Kim

Presenter: Junhyeok Park

SOGANG UNIVERSITY

The 17th ACM Workshop on
Hot Topics in Storage and File Systems

July 10-11
Boston, MA

KOREA UNIVERSITY

Inside Today's NVMe SSDs

Recent advances in SSD technology have led to increasingly powerful devices, with modern NVMe SSDs now equipped with substantial internal DRAM and multi-core ARM processors.

Samsung V-NAND V5
2 chips @ 4 Tbit

Samsung LPDDR4-1866 DRAM
features a 1 GB capacity

Samsung Phoenix Controller
features 5 ARM Cortex-R7 cores

[1] TechPowerUp. "Samsung 970 EVO Plus 1 TB SSD Specs." *TechPowerUp*, accessed June 20, 2025. [link]

Inside Today's NVMe SSDs

Recent advances in SSD technology have led to increasingly powerful devices, with modern NVMe SSDs now equipped with substantial internal DRAM and multi-core ARM processors.

Samsung V-NAND V5

As these capabilities continue to evolve, SSDs are gradually shifting from simple storage solutions to intelligent data-processing devices,

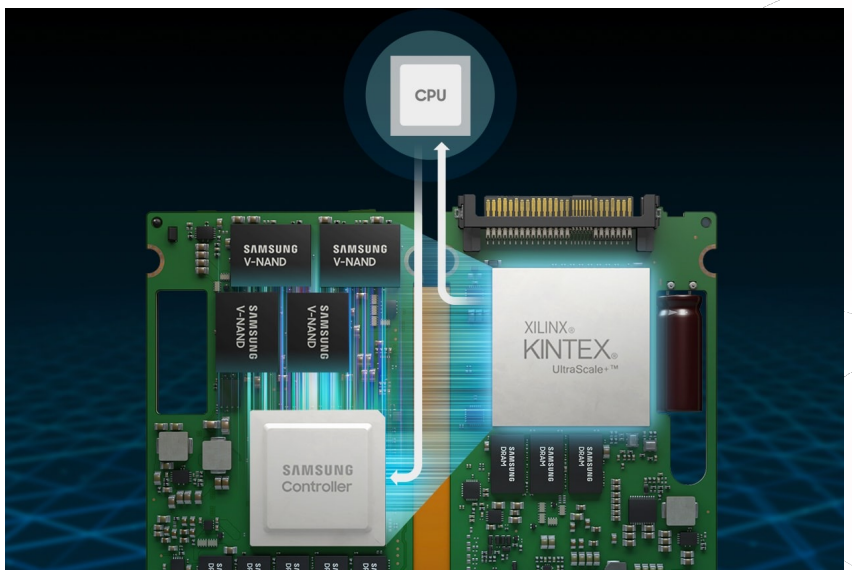
opening new opportunities for offloading and executing host-side tasks.

[1] TechPowerUp. "Samsung 970 EVO Plus 1 TB SSD Specs." *TechPowerUp*, accessed June 20, 2025. [link]

Computational Storage Devices

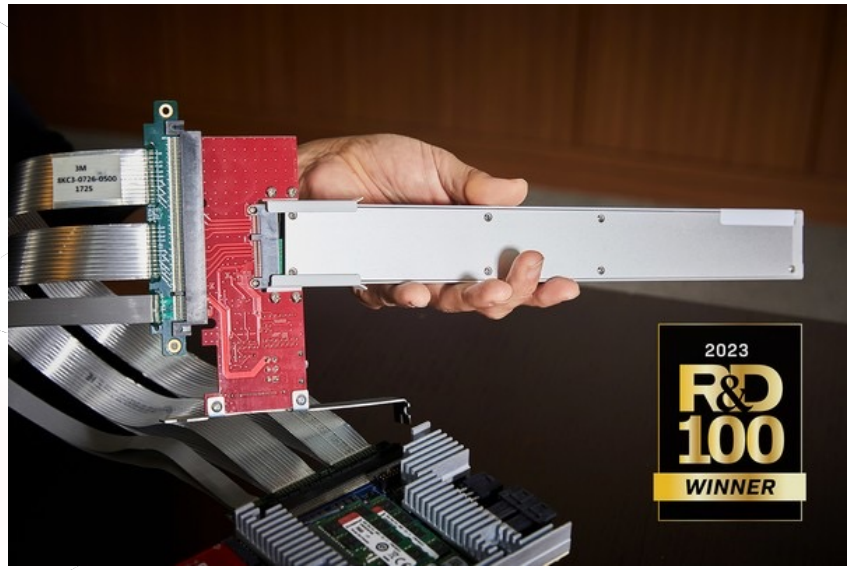
These trends have led to the emergence of new classes of computational storage devices, including …

- **Computational SSDs (CSDs)** that can execute user's analytics tasks such as SQL filters inside the SSD.
- **Key-Value SSDs (KV-SSDs)** that natively support key-value operations within the device by bypassing traditional file systems.



Samsung SmartSSD can achieve 2.8x faster SQL query execution on Parquet data [2]

SK hynix KV-CSD showed up to 10.6x lower write latency than RocksDB [3]



[2] Samsung Electronics. *Samsung SmartSSD Computational Storage Drive*. Brochure, accessed June 20, 2025. [link]
[3] I. Park et al., "KV-CSD: A Hardware-Accelerated Key-Value Store for Data-Intensive Applications", 2023 IEEE International Conference on Cluster Computing (CLUSTER). [link]

NVMe-Based New Storage Interface

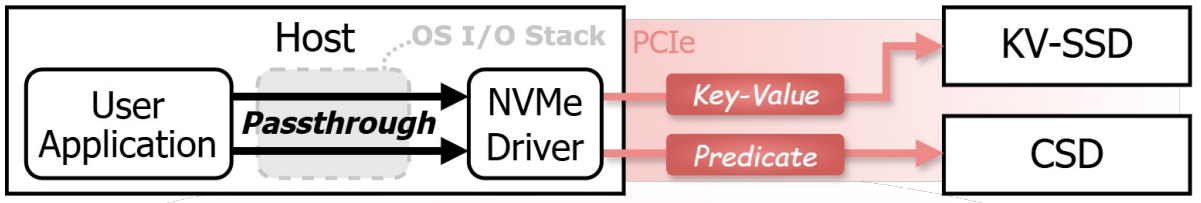
To interact with these new types of devices, users commonly employ the NVMe passthrough, wherein application-level requests, such as …

- **SQL predicates** for CSDs or
- **key-value pairs** for KV-SSDs,

are encoded as custom NVMe Commands (CMDs) and sent directly to the device.

```
86  int nvme_submit_passthru64(int fd, unsigned long ioctl_cmd,
87                             struct nvme_passthru_cmd64 *cmd,
88                             __u64 *result)
89  {
90          int err = ioctl(fd, ioctl_cmd, cmd);
91
92          if (err >= 0 && result)
93                  *result = cmd->result;
94          return err;
95  }
```

NVMe Passthru Function
defined in libnvme/blob/master/src/nvme/ioctl.c



NVMe Passthru-based CSD & KV-SSD Interface
➔ enables seamless data exchange with these devices without significantly modifying the kernel I/O stack

NVMe-Based New Storage Interface

To interact with these new types of devices, users commonly employ the NVMe passthrough, wherein application-level requests, such as …

- **SQL predicates** for CSDs or
- **key-value pairs** for KV-SSDs,

are encoded as custom NVMe Commands (CMDs) and sent directly to the device.

Interestingly, a closer look at this new storage interface reveals that the actual data payloads (*i.e., SQL predicates and key-value pairs****) transferred in such requests are often small.***

defined in libnvme/blob/master/src/nvme/ioctl.c

The Advent of Small, Direct I/O

CSDs process filter operations based on computation task specifications that typically require only a table identifier and predicates since the SSD is already aware of table schema information [4].

```
SELECT min(vertex_id) AS VID, min(x) as X, min(y) as Y, min(z) as Z, avg(e) AS E
FROM 'xx_36785.parquet'
WHERE x > 1.5 AND x < 1.6 AND y > 1.5 AND y < 1.6 AND z > 1.5 AND z < 1.6
GROUP BY vertex_id ORDER BY E;
```

Example SQL Query from the Laghos 3D Mesh Dataset [5]
computes per-vertex average energy in a small 3D region

Table Name

```
'xx_36785.parquet'
```

Predicate Clause

```
x > 1.5 AND x < 1.6 AND y > 1.5 AND y < 1.6 AND z > 1.5 AND z < 1.6
```

Extracted 'Table Name' and 'Predicate Clause'

1001 1101 0101 0110 ...
(just exemplar binary codes)

Payload

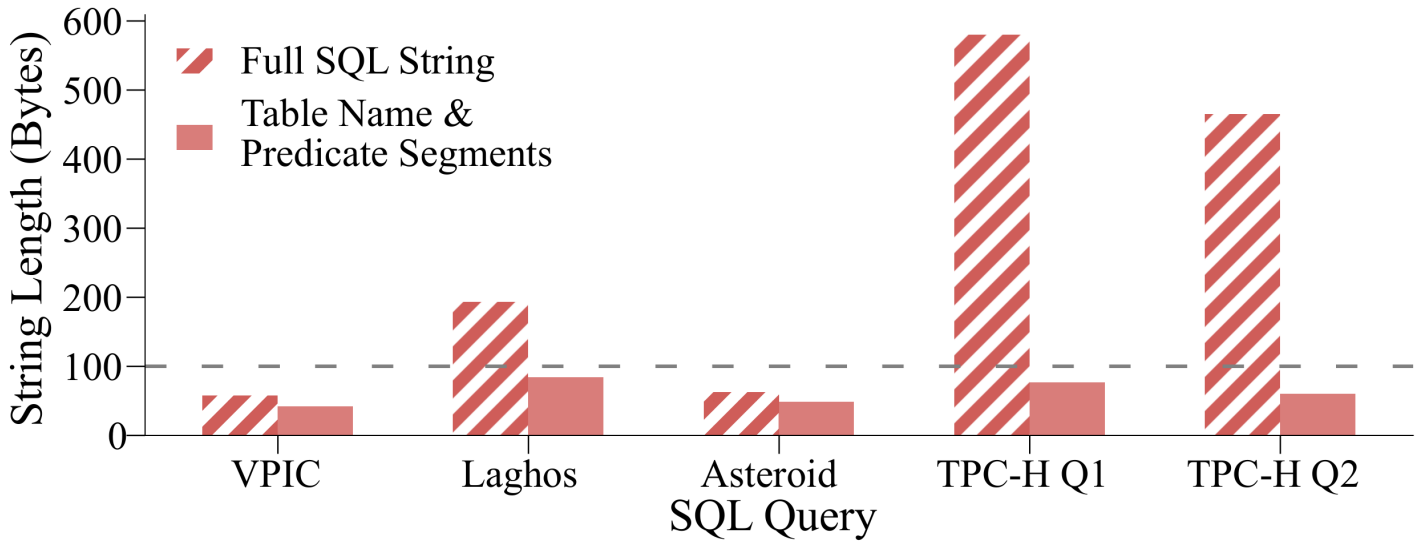
Binary Encoded Message
specifies computation task (SQL filtering)

[4] I. Jo et al., "YourSQL: a high-performance database system leveraging in-storage computing". Proc. VLDB Endow. 9, 12 (August 2016), 924–935. [link]
[5] Los Alamos National Laboratory. *Laghos Sample Dataset*. GitHub repository, accessed June 20, 2025. [link]

The Advent of Small, Direct I/O

This results in payloads of just tens to hundreds of bytes.

➔ Even when expressed as a completely unoptimized SQL string without any binary encoding, the total payload size remains small.

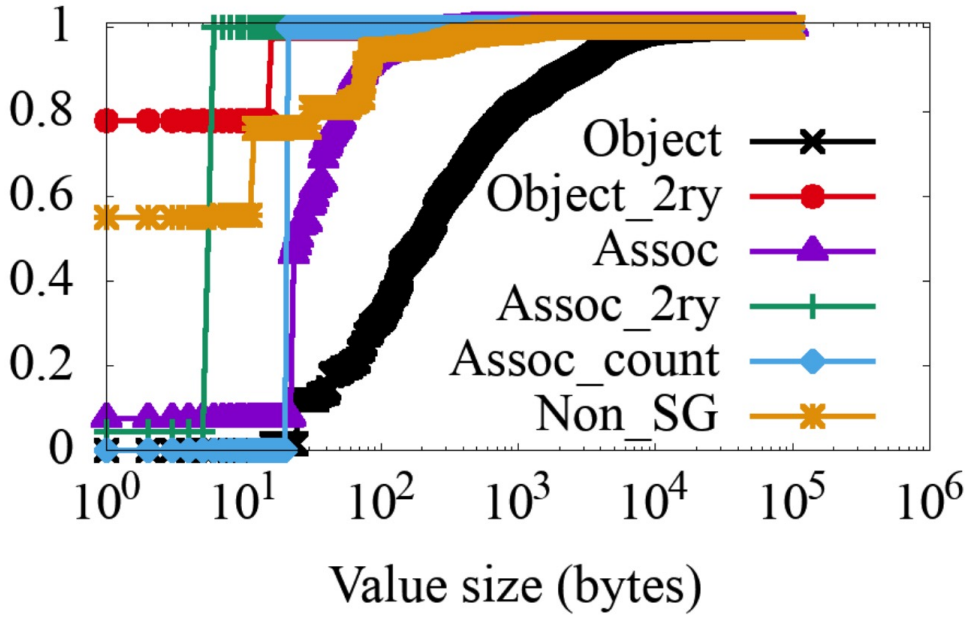


Example Queries Used in CSD Works [4 - 8]
showing the lengths of full string and table/predicate segments

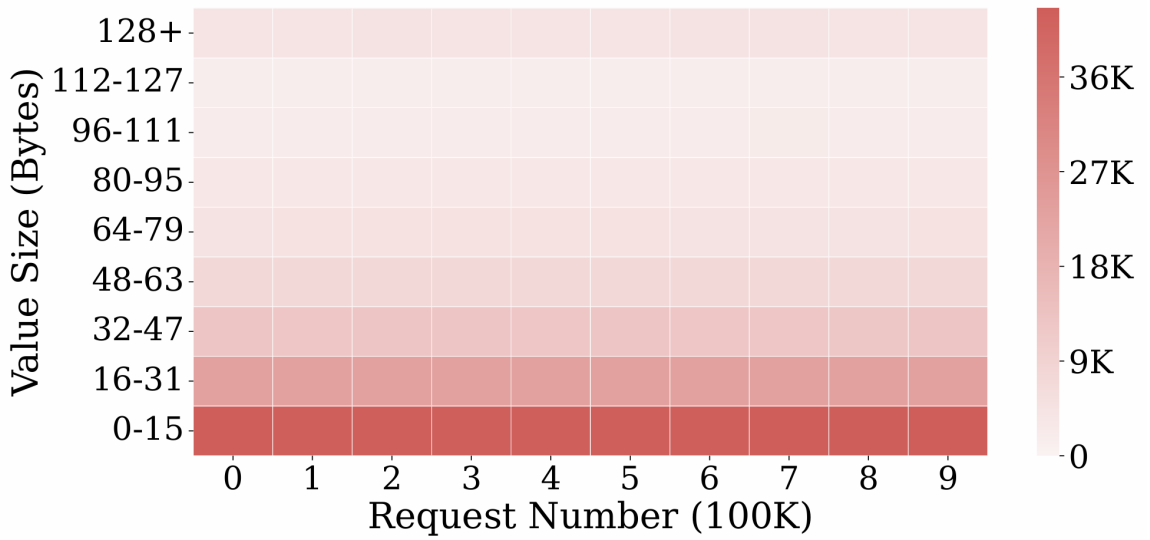
[4] I. Jo et al., "YourSQL: a high-performance database system leveraging in-storage computing". Proc. VLDB Endow. 9, 12 (August 2016), 924–935. [link]
[5] Los Alamos National Laboratory. *Laghos Sample Dataset*. GitHub repository, accessed June 20, 2025. [link]
[6] Los Alamos National Laboratory. *DeepWater Impact Dataset*. GitHub repository, accessed June 20, 2025. [link]
[7] Los Alamos National Laboratory. *VPIC: Vector Particle-In-Cell Simulation Code*. GitHub repository, accessed June 20, 2025. [link]
[8] J. Kim, "Accelerating Data Analytics Using Object-based Computational Storage System in HPC“. In Supercomputing Conference (SC23), Denver, CO, USA, 2023. [link]

The Advent of Small, Direct I/O

Similarly, the values processed during real-world key-value operations are often just a few dozen bytes in size, as shown by internal workload analyses from Meta [9] and Twitter [10].



Value Size CDF for RocksDB as a MySQL Storage Layer in Meta's Data Centers

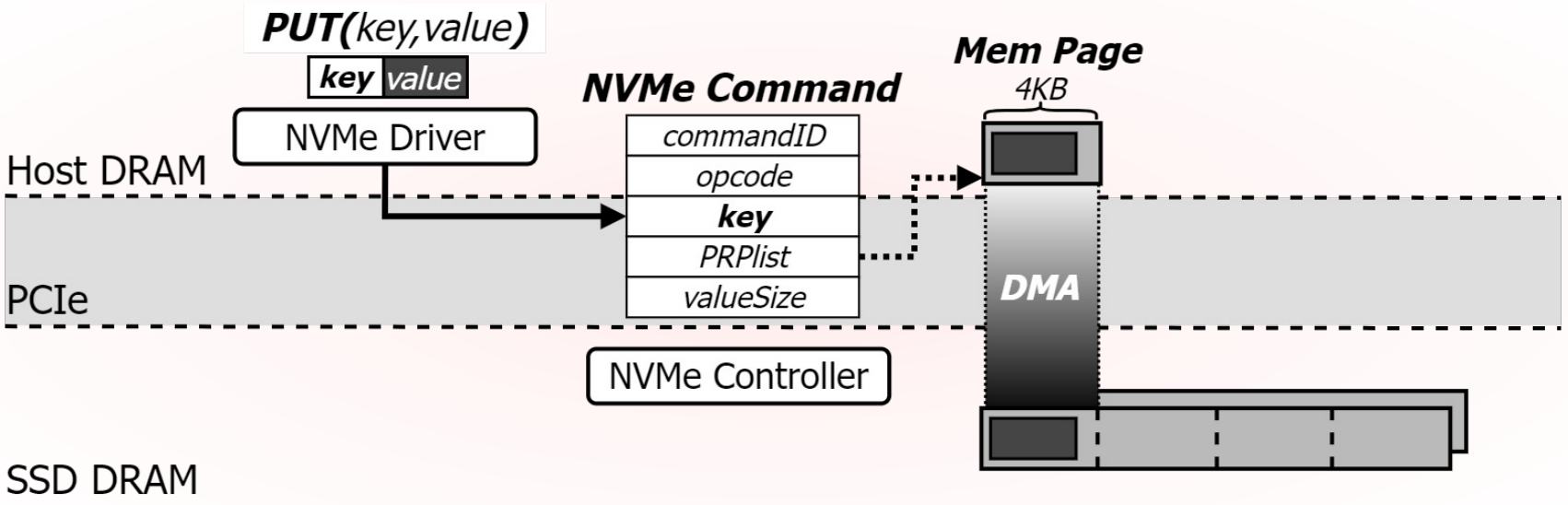


Value Size Heatmap from MixGraph All_random with Its Default Parameter Settings

[9] Z. Cao et al., "Characterizing, Modeling, and Benchmarking RocksDB Key-Value Workloads at Facebook". In Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST'20) [link]
[10] 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) [link]

The Advent of Small, Direct I/O

These small values are persisted individually in KV-SSDs, in accordance with the key-value pair-level transaction model defined in the NVMe key-value extension [11].

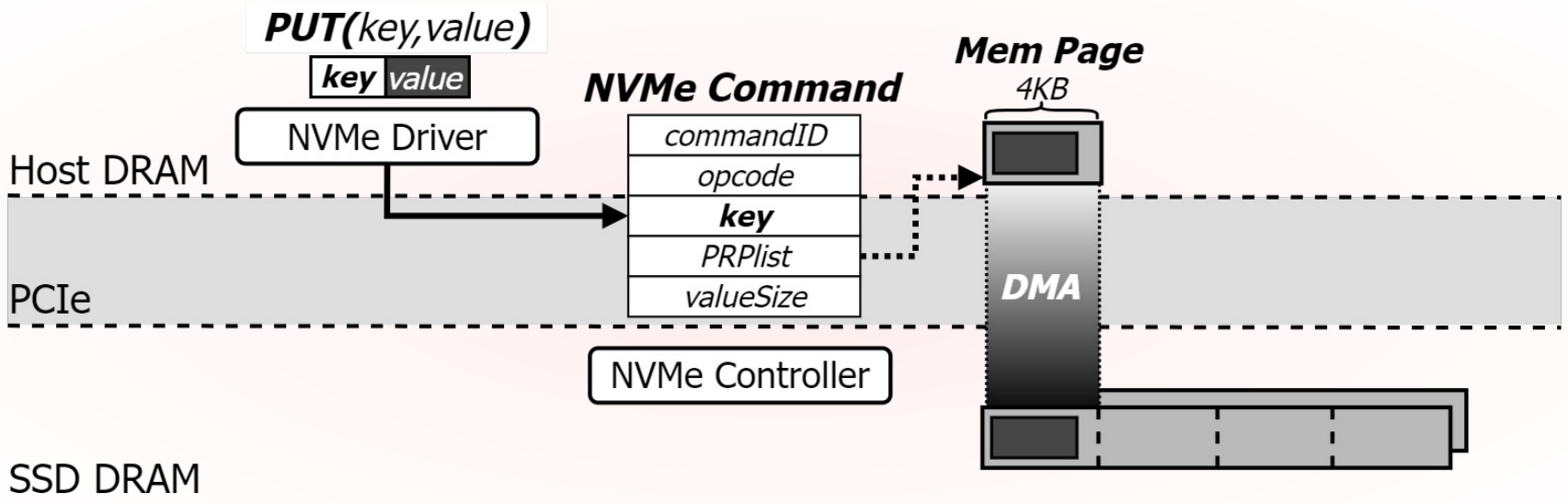


- As a result, small direct I/O operations are a natural and frequent part of KV-SSD behavior in real-world scenarios.
  - Although batching multiple key-value pairs into a single bulk PUT has been explored in some prior work, such approaches may not always be applicable, particularly in use cases where fine-grained persistence is desired for each key-value pair [12].

[11] NVM Express Inc. 2021. NVM Express Key Value Command Set Specification. Last Accessed: 2024-09-12. [link]
[12] etcd Authors. 2023. etcd Raft Log Durability and Performance. Last Accessed: 2025-05-25. [link]

The Advent of Small, Direct I/O

These small values are persisted individually in KV-SSDs, in accordance with the key-value pair-level transaction model defined in the NVMe key-value extension [11].

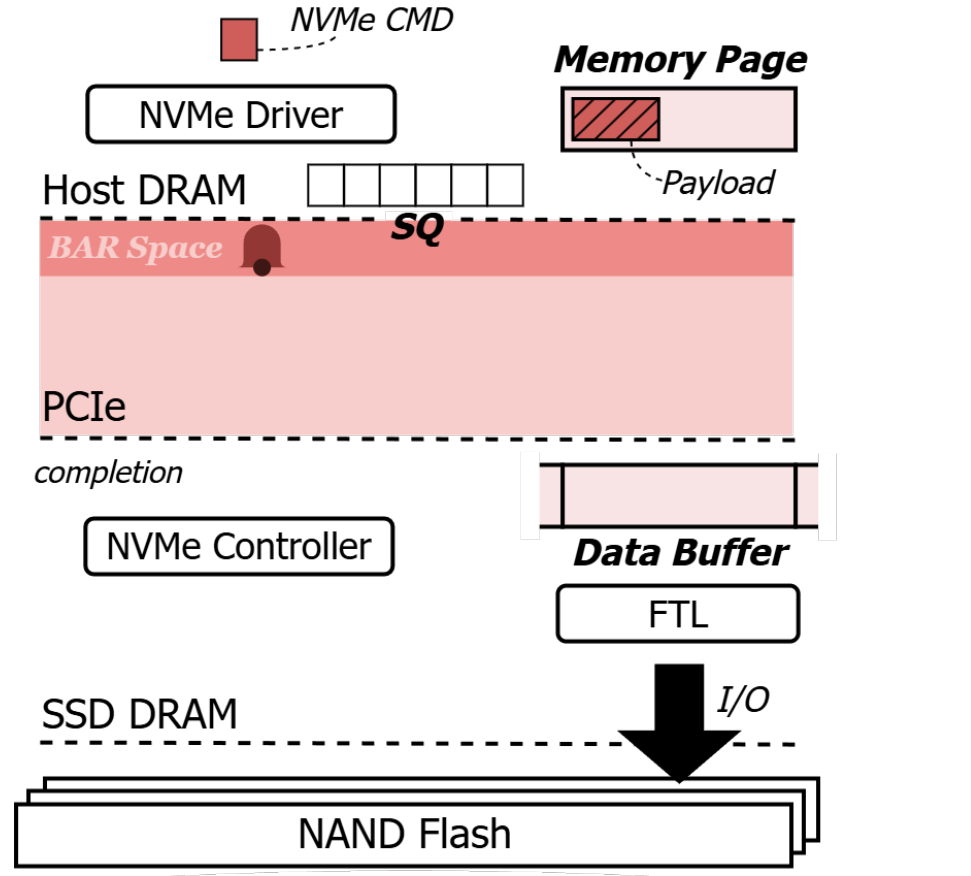


However, despite the prevalence of small payloads, the conventional NVMe protocol is ironically ill-suited to handle them efficiently.

[11] NVM Express Inc. 2021. NVM Express Key Value Command Set Specification. Last Accessed: 2024-09-12. [link]
[12] etcd Authors. 2023. etcd Raft Log Durability and Performance. Last Accessed: 2025-05-25. [link]

NVMe is Not Small I/O Friendly!

Specifically, NVMe employs Physical Region Pages (PRPs) for data transfer, which requires data to be transferred in 4 KB memory page units.

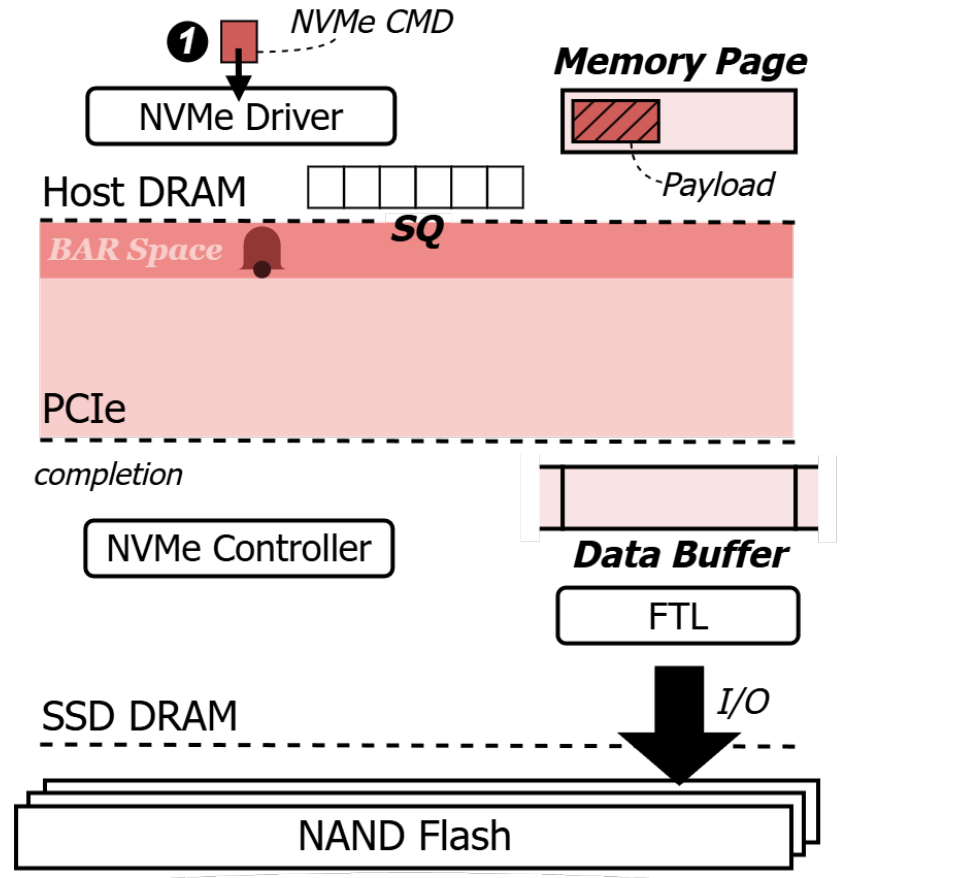


NVMe PRP-Based Payload Transfer Mechanism

The host prepares the data in memory pages and constructs an **NVMe CMD** that specifies the (1) address and (2) number of pages to transfer.

NVMe is Not Small I/O Friendly!

Specifically, NVMe employs Physical Region Pages (PRPs) for data transfer, which requires data to be transferred in 4 KB memory page units.

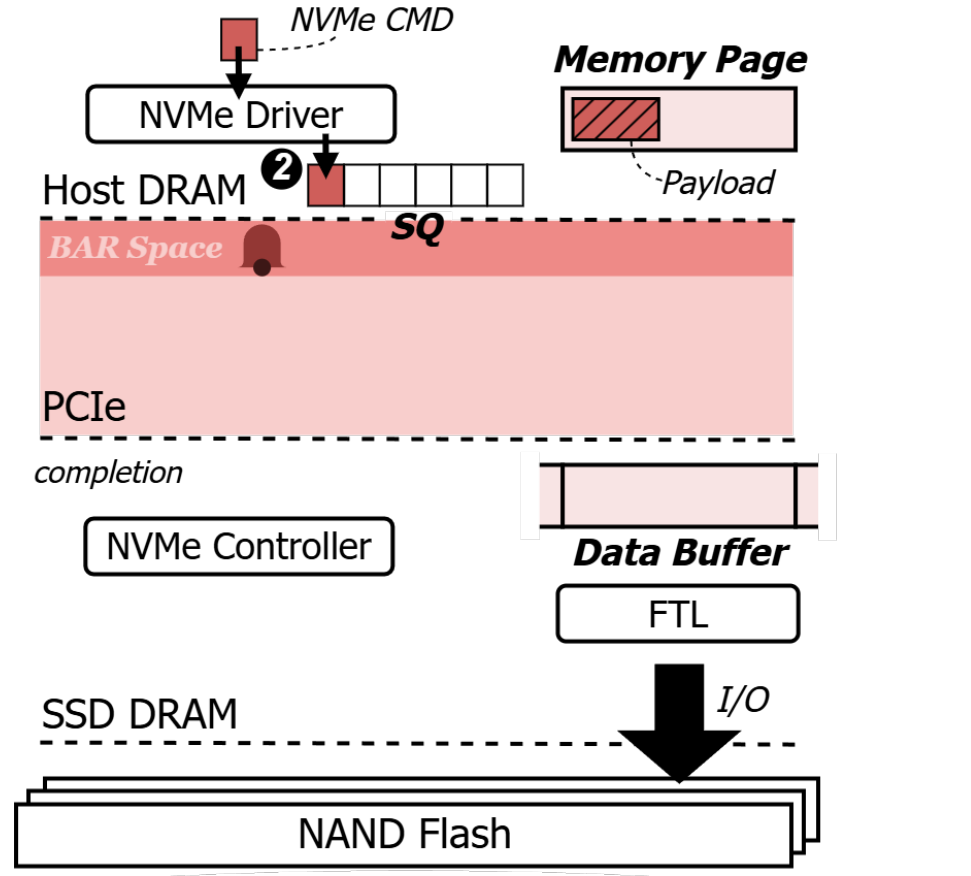


NVMe PRP-Based Payload Transfer Mechanism

1 The CMD is submitted to the NVMe driver.

NVMe is Not Small I/O Friendly!

Specifically, NVMe employs Physical Region Pages (PRPs) for data transfer, which requires data to be transferred in 4 KB memory page units.

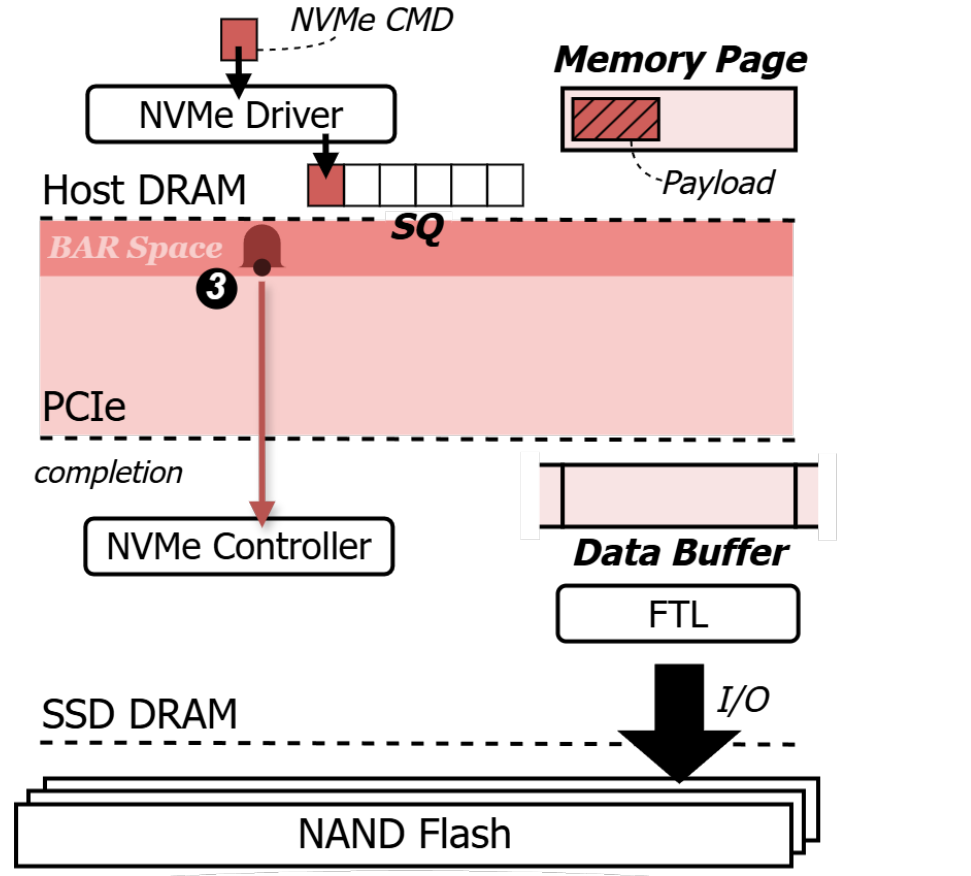


NVMe PRP-Based Payload Transfer Mechanism

2 The driver inserts the CMD into the **NVMe Submission Queue (SQ).**

NVMe is Not Small I/O Friendly!

Specifically, NVMe employs Physical Region Pages (PRPs) for data transfer, which requires data to be transferred in 4 KB memory page units.

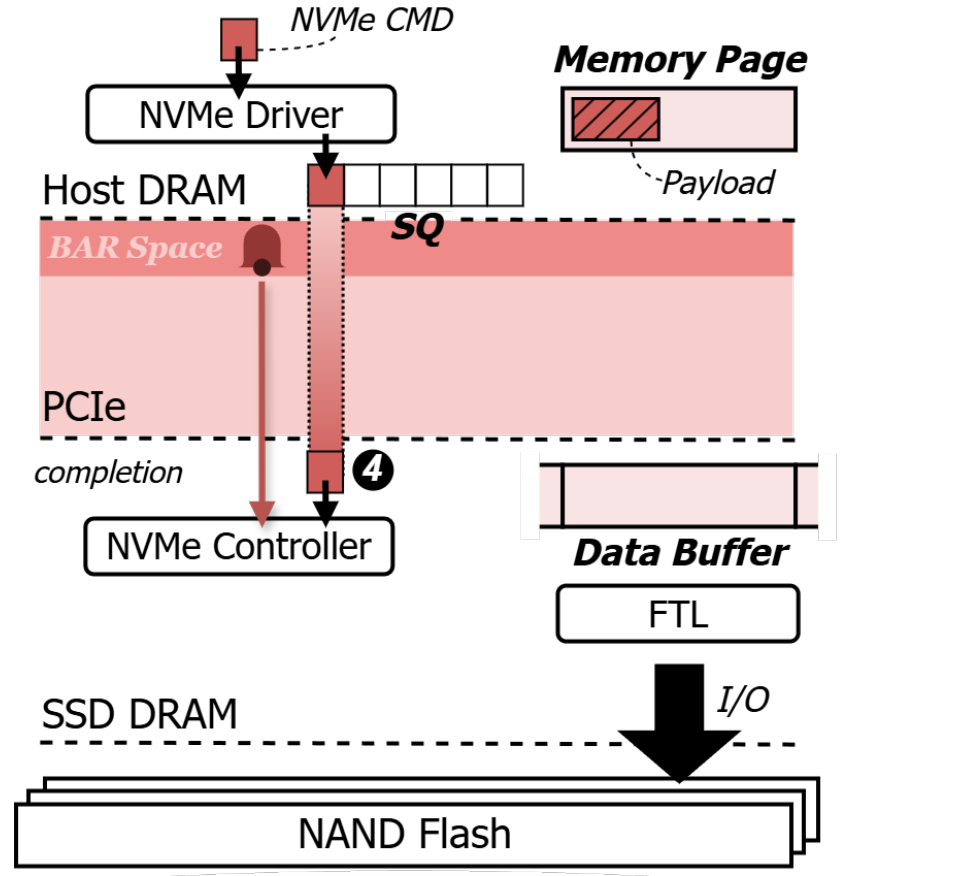


NVMe PRP-Based Payload Transfer Mechanism

3 The driver triggers **the doorbell register** in the PCIe Base Address Register (BAR) space to notify the device of a new submission.

NVMe is Not Small I/O Friendly!

Specifically, NVMe employs Physical Region Pages (PRPs) for data transfer, which requires data to be transferred in 4 KB memory page units.

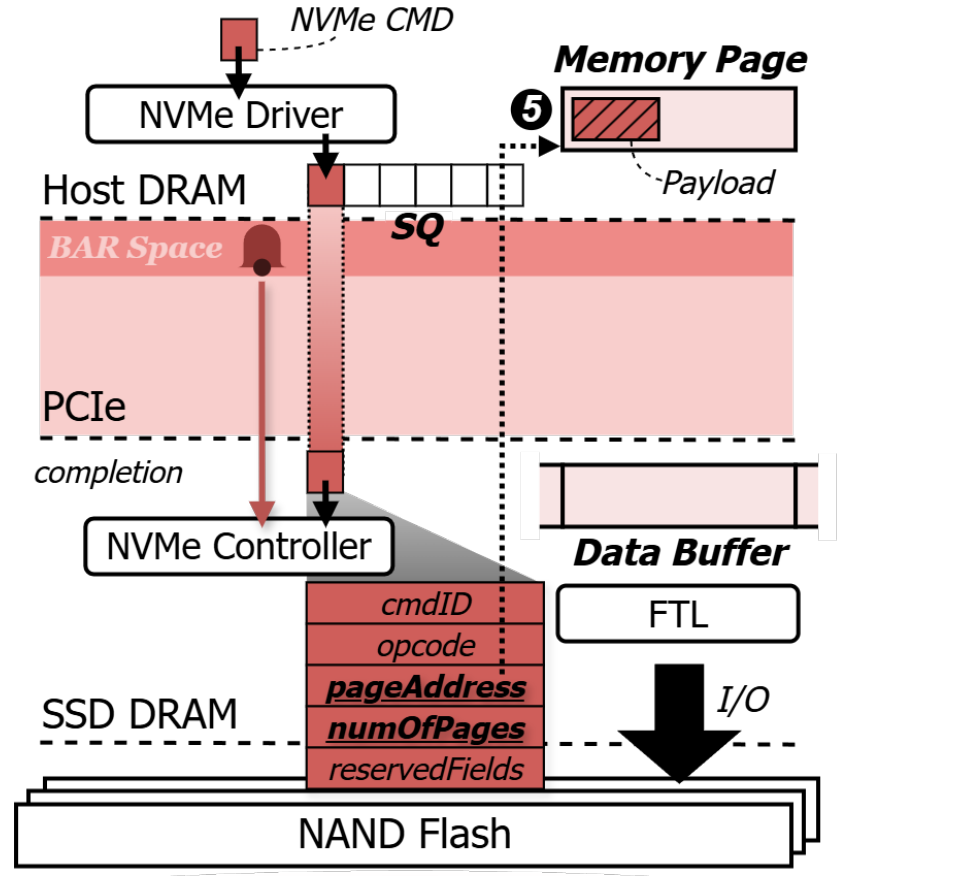


NVMe PRP-Based Payload Transfer Mechanism

4 The device, polling the doorbell register, detects the submission and performs a **64-byte Direct Memory Access (DMA)** fetch of the CMD.

NVMe is Not Small I/O Friendly!

Specifically, NVMe employs Physical Region Pages (PRPs) for data transfer, which requires data to be transferred in 4 KB memory page units.

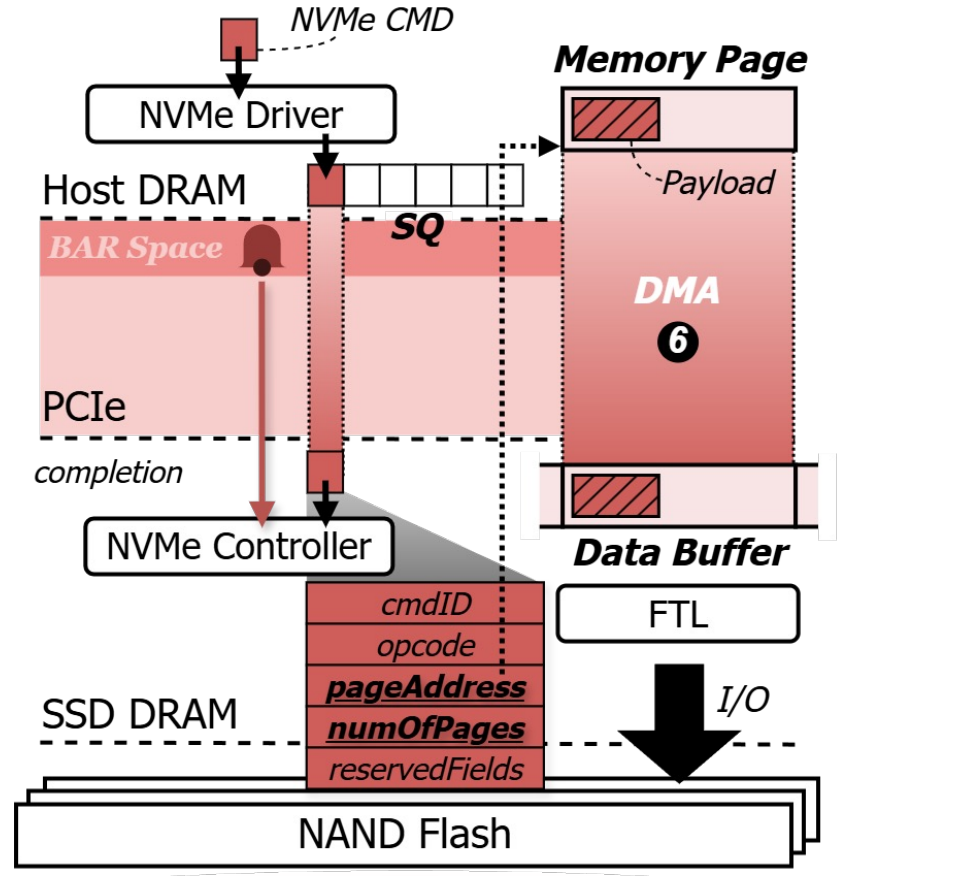


NVMe PRP-Based Payload Transfer Mechanism

5 The controller then reads (identifies) the indicated host **page addresses** and **page counts** within PRP entries.

NVMe is Not Small I/O Friendly!

Specifically, NVMe employs Physical Region Pages (PRPs) for data transfer, which requires data to be transferred in 4 KB memory page units.

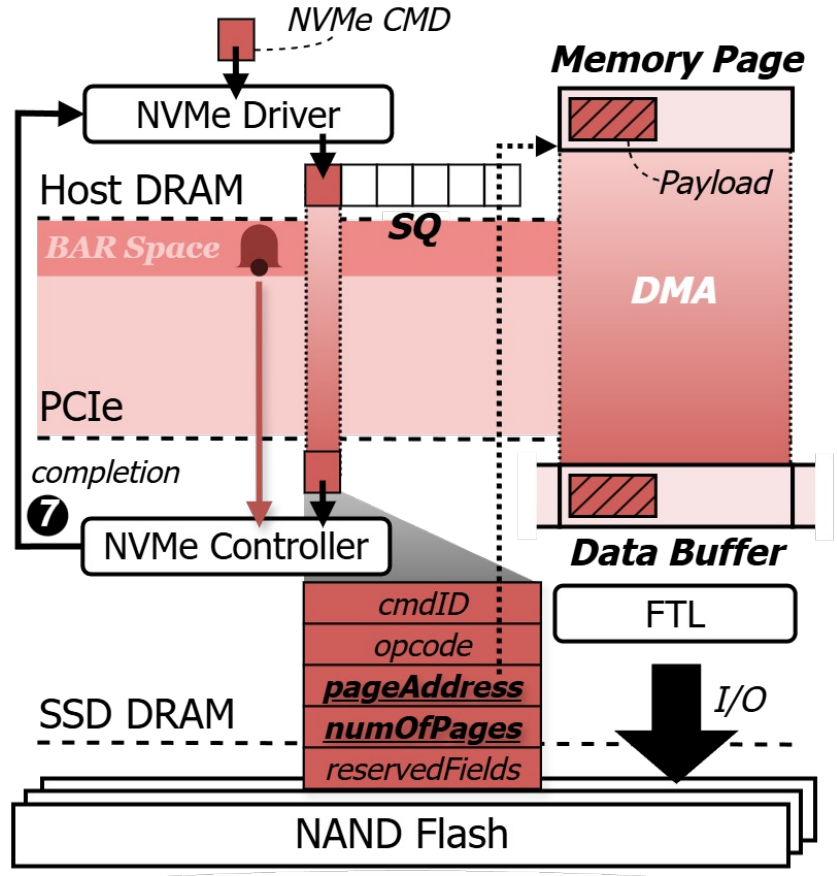


NVMe PRP-Based Payload Transfer Mechanism

6 The controller copies the corresponding pages into device DRAM via 4 KB page unit DMAs.

NVMe is Not Small I/O Friendly!

Specifically, NVMe employs Physical Region Pages (PRPs) for data transfer, which requires data to be transferred in 4 KB memory page units.

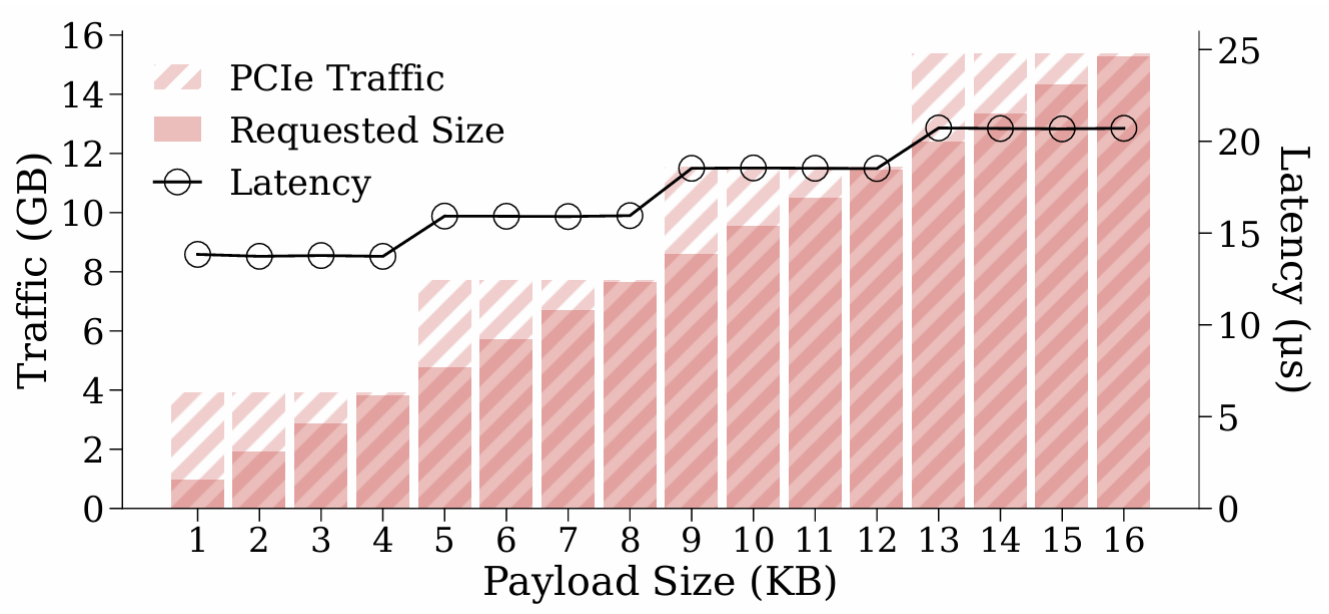


NVMe PRP-Based Payload Transfer Mechanism

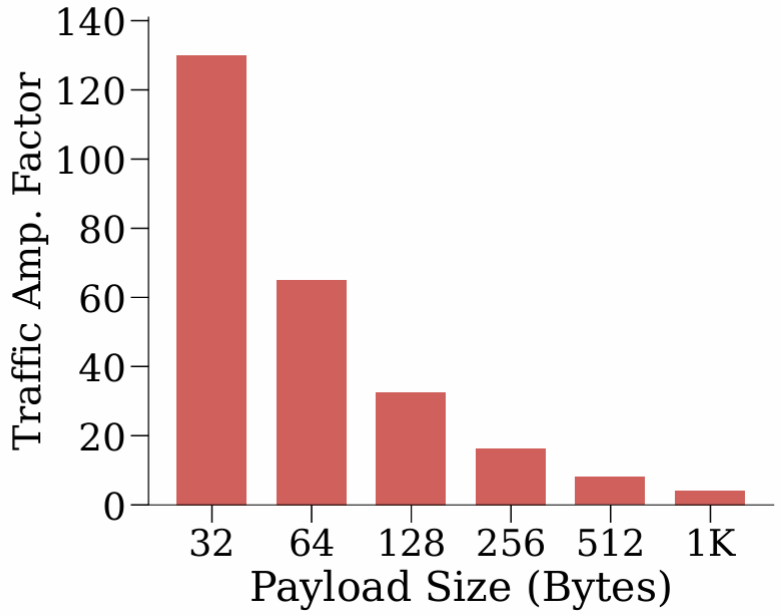
7 Finally, the device signals completion to the host.

NVMe is Not Small I/O Friendly!

As a result, even a 32-byte payload incurs 4 KB of PCIe traffic, more than 130× greater than the requested size.



PCIe Traffic and Transfer Latency



Traffic Amplification Factor

➔ This significant traffic bloating can lead to increased latency and unnecessary power consumption, making it a critical bottleneck for frequent, small direct I/Os.

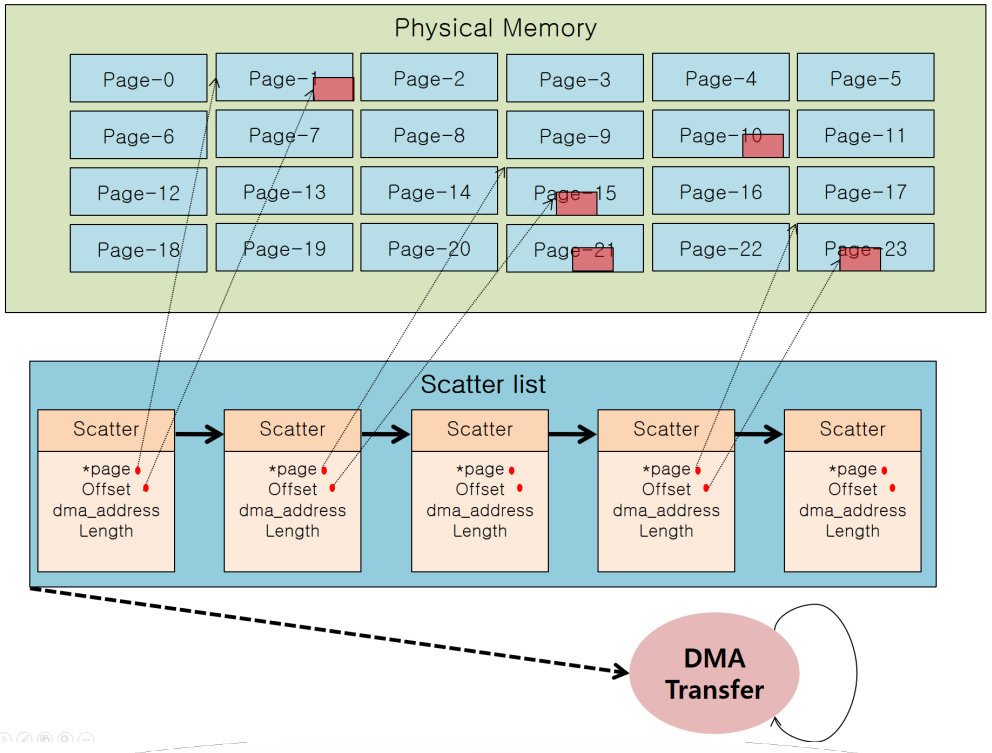
NVMe Scatter-Gather List ?

NVMe also supports Scatter-Gather Lists (SGLs), which can enable fine-grained DMA transfer for small payloads.

However, SGLs are not supported across all NVMe devices (because it is not mandatory), and even when supported, the Linux kernel is configured by default to use them only for payloads larger than 32 KB.

```
76      static unsigned int sgl_threshold = SZ_32K;
77      module_param(sgl_threshold, uint, 0644);
78      MODULE_PARM_DESC(sgl_threshold,
79                       "Use SGLs when average request segment size is larger or equal to "
80                       "this size. Use 0 to disable SGLs.");
```

Linux Kernel's NVMe SGL Size Threshold
defined in [drivers/nvme/host/pci.c]



NVMe SGL-Based Payload Transfer Mechanism

➔ ***Accordingly, this work focuses on optimizing PRP-based transfers.***

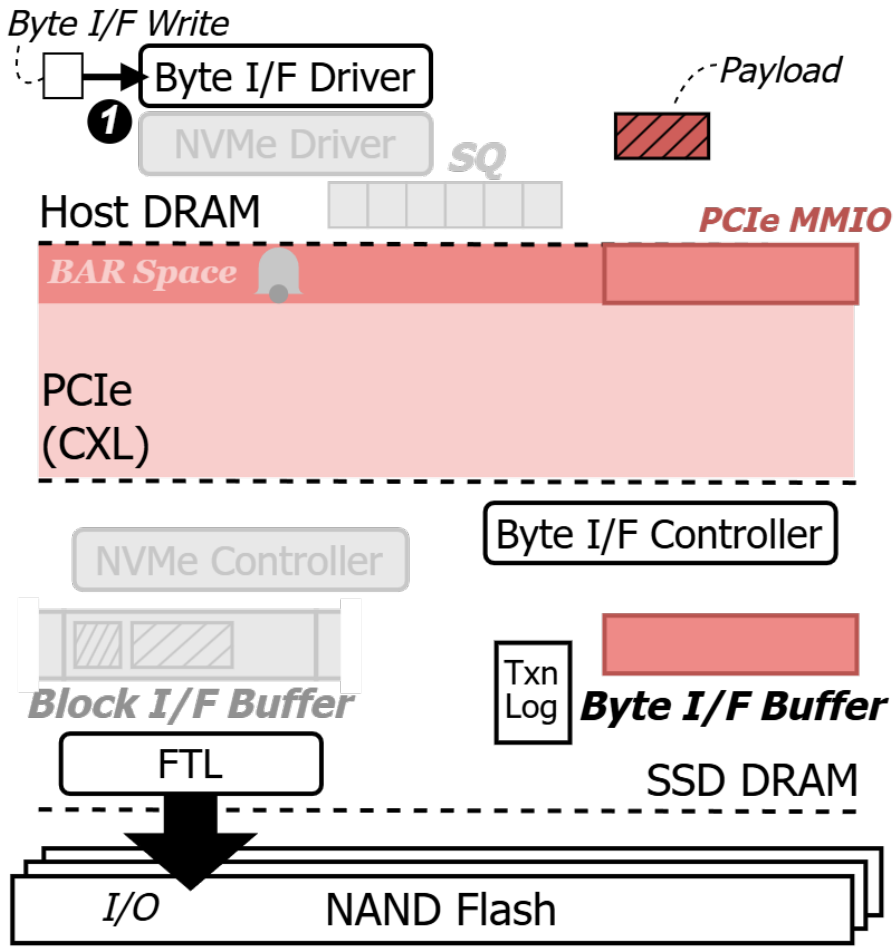
Existing Approaches Work! But…

Prior works have attempted to address this issue using two primary approaches:

(1) Bypassing the NVMe stack via **PCIe Memory-Mapped I/O (MMIO)** to enable byte-level writes directly to the device.

- **Ex)** 2B-SSD [ISCA `18], ByteFS [ASPLOS `25]

❶ If a user application issues a Byte Interface write request,



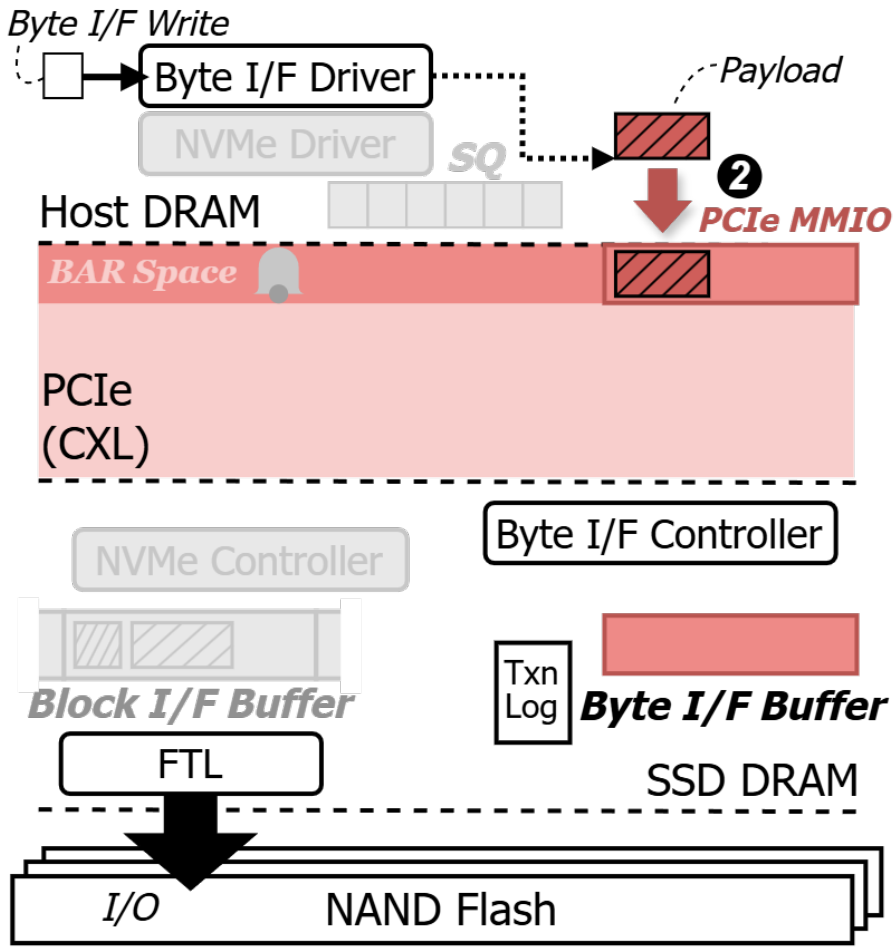
Existing Approaches Work! But…

Prior works have attempted to address this issue using two primary approaches:

(1) Bypassing the NVMe stack via **PCIe Memory-Mapped I/O (MMIO)** to enable byte-level writes directly to the device.

- **Ex)** 2B-SSD [ISCA `18], ByteFS [ASPLOS `25]

❷ the data is memory-copied to the device-mapped PCIe BAR region.



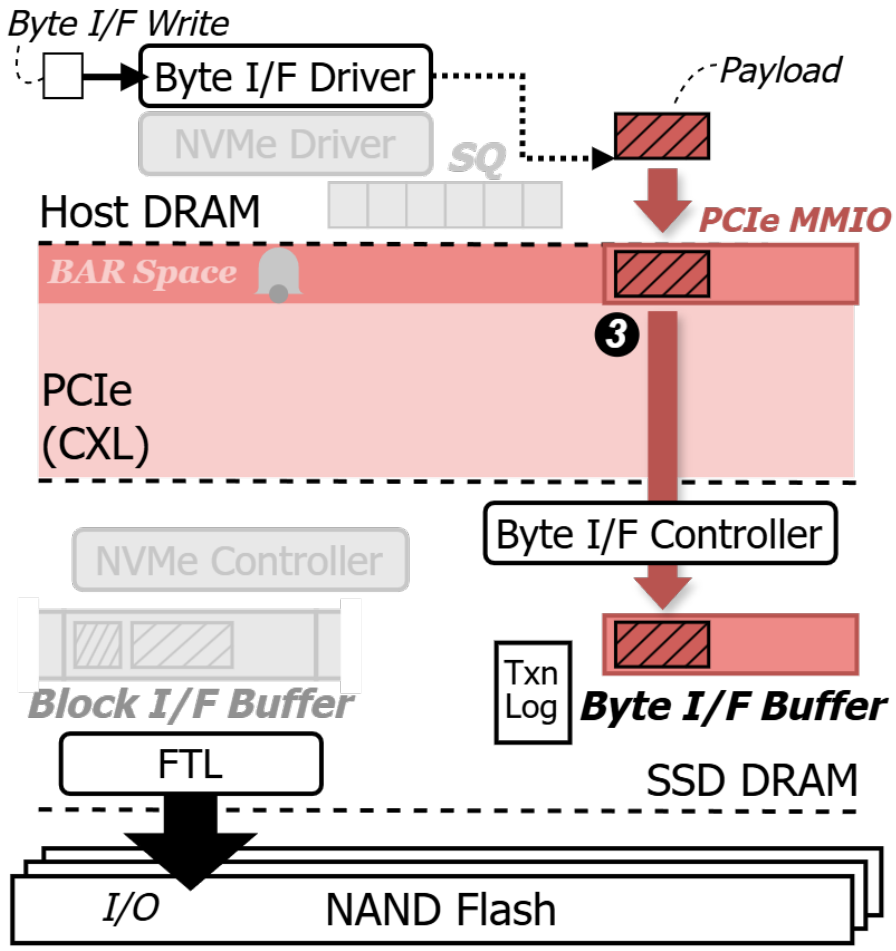
Existing Approaches Work! But…

Prior works have attempted to address this issue using two primary approaches:

(1) Bypassing the NVMe stack via **PCIe Memory-Mapped I/O (MMIO)** to enable byte-level writes directly to the device.

- **Ex)** 2B-SSD [ISCA `18], ByteFS [ASPLOS `25]

❸ Writing to the device-mapped PCIe BAR region results in a memory write transaction over PCIe, which **automatically updates** the corres-ponding address in device memory.



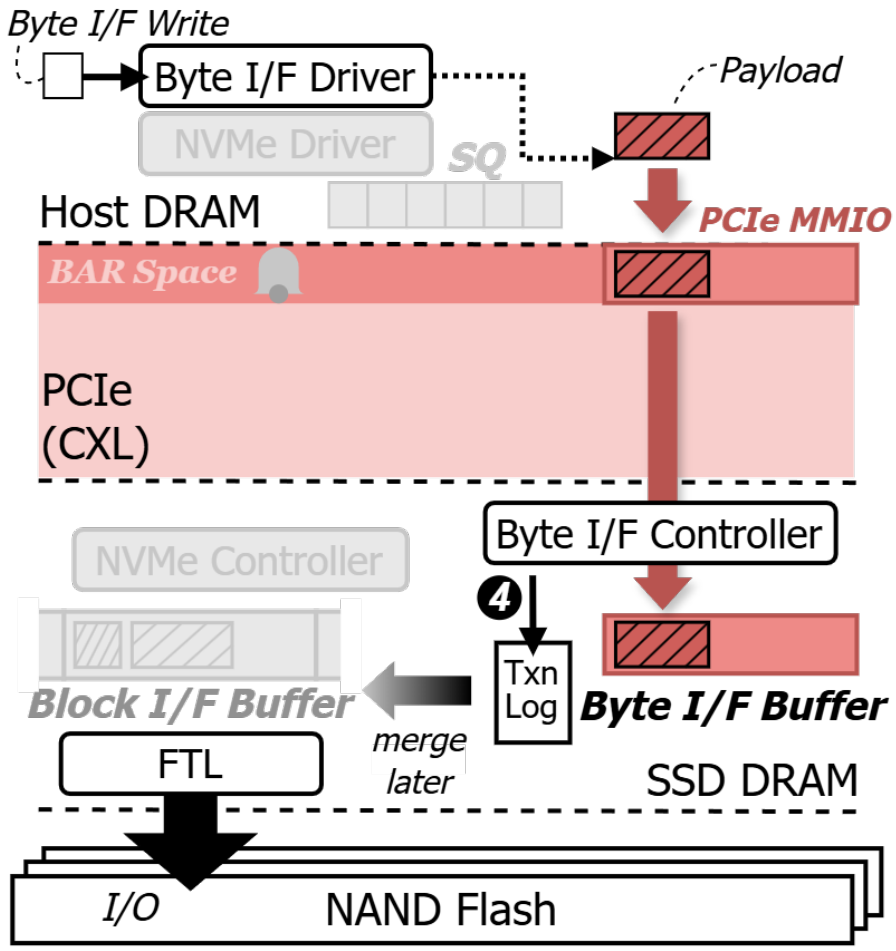
Existing Approaches Work! But…

Prior works have attempted to address this issue using two primary approaches:

(1) Bypassing the NVMe stack via **PCIe Memory-Mapped I/O (MMIO)** to enable byte-level writes directly to the device.

- **Ex)** 2B-SSD [ISCA `18], ByteFS [ASPLOS `25]

❹ If needed, the written data is coordinated with data written through the Block Interface, thru transaction logs or similar methods to maintain consistency.



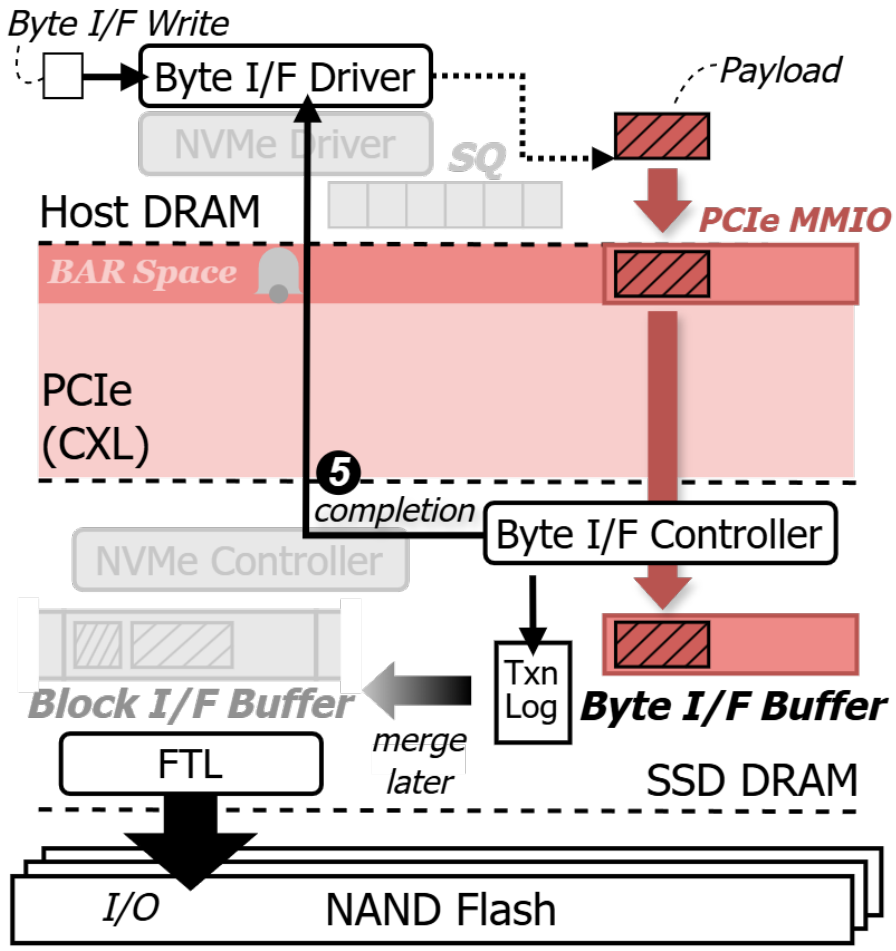
Existing Approaches Work! But…

Prior works have attempted to address this issue using two primary approaches:

(1) Bypassing the NVMe stack via **PCIe Memory-Mapped I/O (MMIO)** to enable byte-level writes directly to the device.

- **Ex)** 2B-SSD [ISCA `18], ByteFS [ASPLOS `25]

5 Then, it notifies the host of the completion.



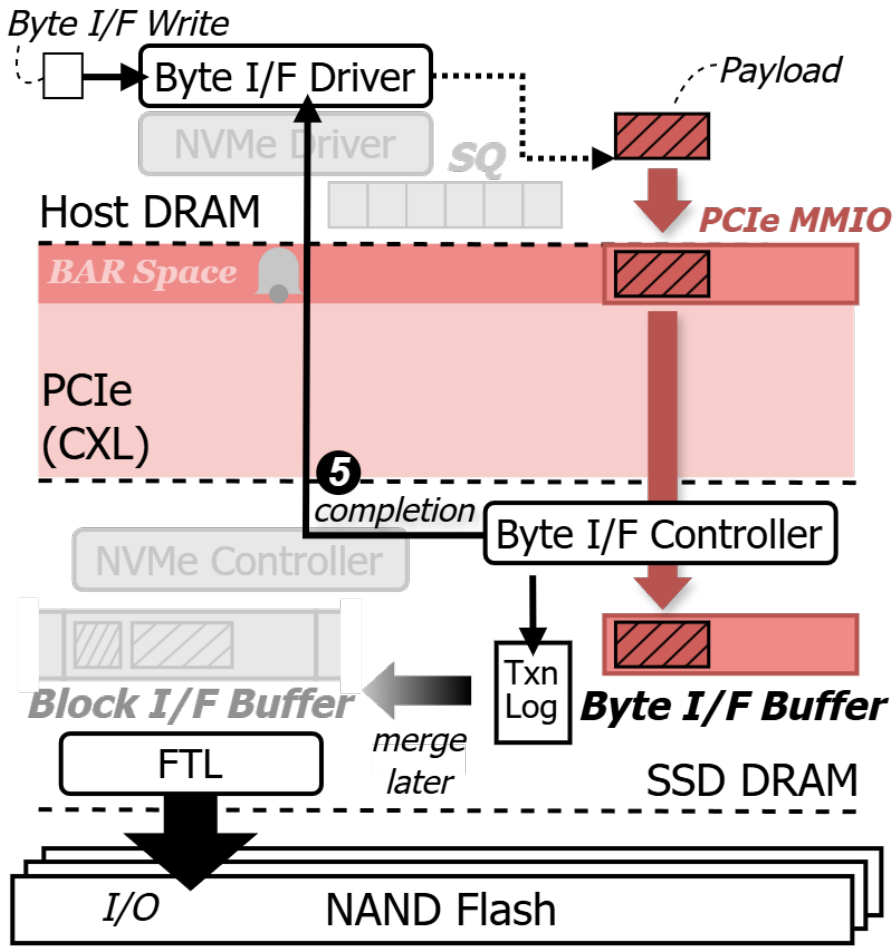
Existing Approaches Work! But…

Prior works have attempted to address this issue using two primary approaches:

(1) Bypassing the NVMe stack via **PCIe Memory-Mapped I/O (MMIO)** to enable byte-level writes directly to the device.

- **Ex)** 2B-SSD [ISCA `18], ByteFS [ASPLOS `25]

- **Pros)**
  - This design allows for low-latency, fine-grained data transfers [13] ➔ High-Performance
  - Can be integrated into a NVMe Controller Memory Buffer (CMB) concept ➔ Opportunity for Standardization
  - Can be extended to utilize PCIe-based memory expansion protocols such as Compute Express Link (CXL).



[13] 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). [link]

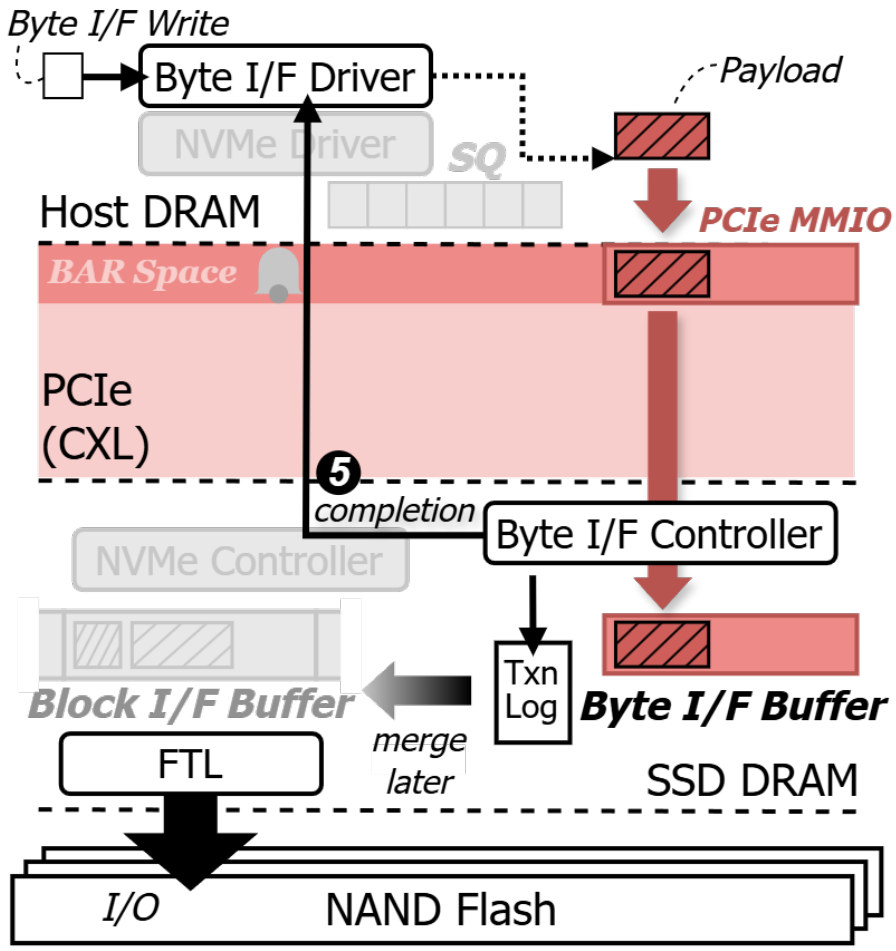
Existing Approaches Work! But…

Prior works have attempted to address this issue using two primary approaches:

(1) Bypassing the NVMe stack via **PCIe Memory-Mapped I/O (MMIO)** to enable byte-level writes directly to the device.

- **Ex)** 2B-SSD [ISCA `18], ByteFS [ASPLOS `25]

- **Cons)**
  - The device must be (1) significantly modified to include a new dedicated buffer memory, and (2) maintain transactional coordination between the new Byte Interface for small payloads and the existing block-based I/O path for large payloads.
    ➔ High Design and Manufacturing Costs
  - On the host-side, NVMe passthrough-based APIs cannot be reused, requiring the development of a new interface layer.
    ➔ Interface Redesign Overhead

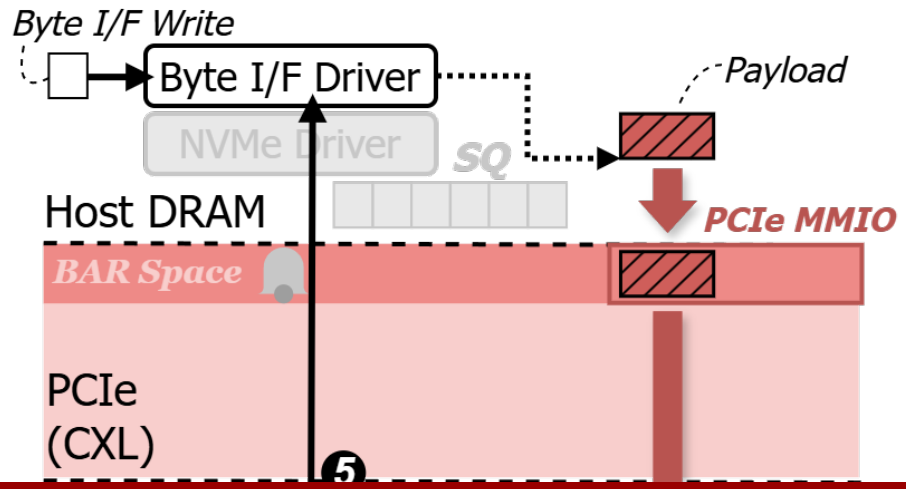


Existing Approaches Work! But…

Prior works have attempted to address this issue using two primary approaches:

(1) Bypassing the NVMe stack via **PCIe Memory-Mapped I/O (MMIO)** to enable byte-level writes directly to the device.

- **Ex)** 2B-SSD [ISCA `18], ByteFS [ASPLOS `25]
- **Cons)**



This method requires substantial modifications to SSD architecture and controller firmware, *making it difficult to integrate with existing NVMe-based CSDs or KV-SSDs.*

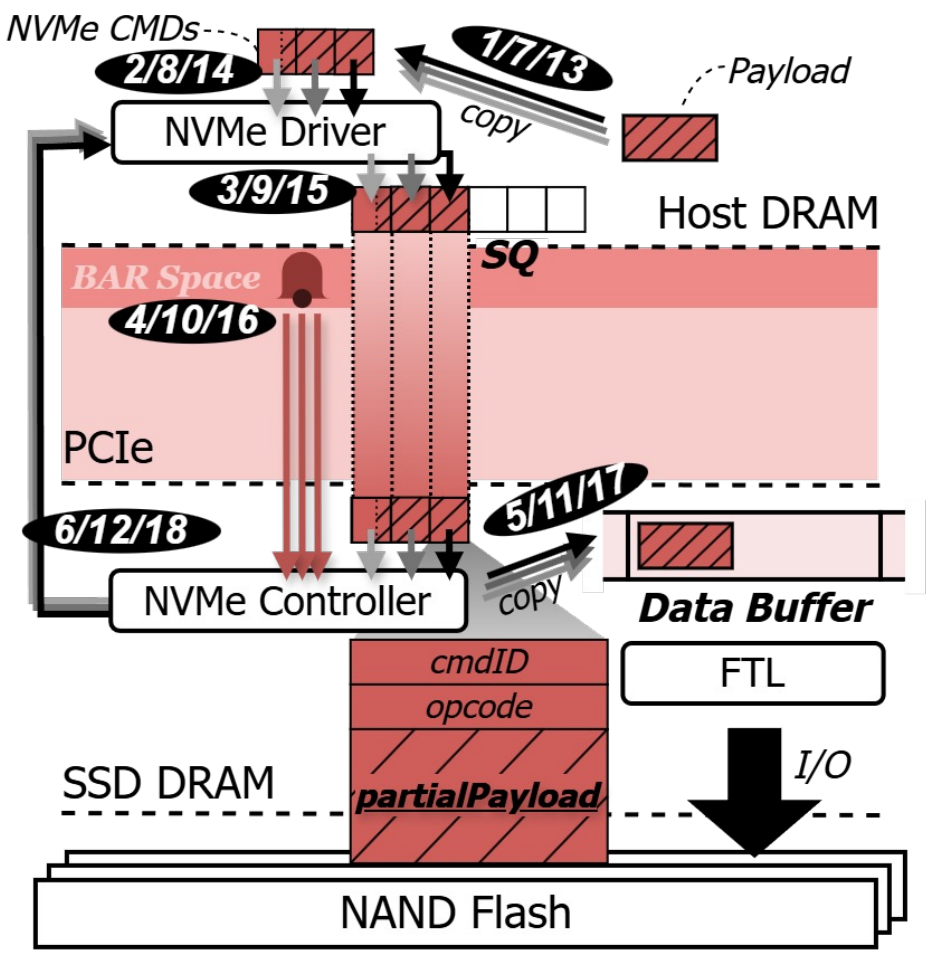
Existing Approaches Work! But…

Prior works have attempted to address this issue using two primary approaches:

(2) Embedding the payload inline within one or more custom NVMe CMDs and transmitting them in fragments.

- **Ex)** BandSlim [ICPP `24], [IEEE Micro `25]

It embeds small payload fragments directly into CMD fields and issues **a sequence of NVMe CMDs (each carrying a portion of the data)** to create fine-grained PCIe traffic patterns.



Existing Approaches Work! But…

Prior works have attempted to address this issue using two primary approaches:

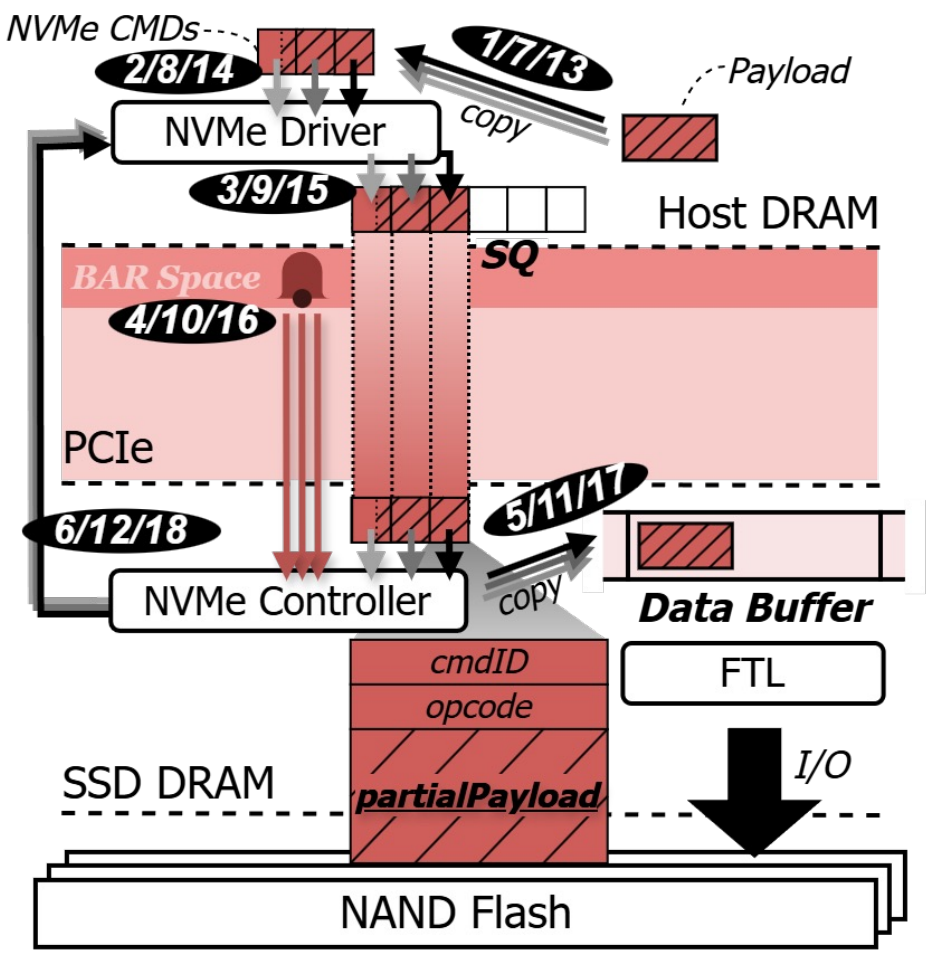
(2) Embedding the payload inline within one or more custom NVMe CMDs and transmitting them in fragments.

- **Ex)** BandSlim [ICPP `24], [IEEE Micro `25]

- **Pros)**
  - By utilizing the NVMe passthrough, this method offers the advantage of requiring no significant modifications to SSD architectures and protocols. ➔ Relatively Easy Integration

- **Cons)**
  - Frequent issuance of CMDs significantly increases protocol overhead, particularly as the number of fragments grows.

    ➔ As a result, when the payload size exceeds just a few dozen bytes (e.g., 64 bytes), the cost of repeated CMD submission becomes a major performance bottleneck.

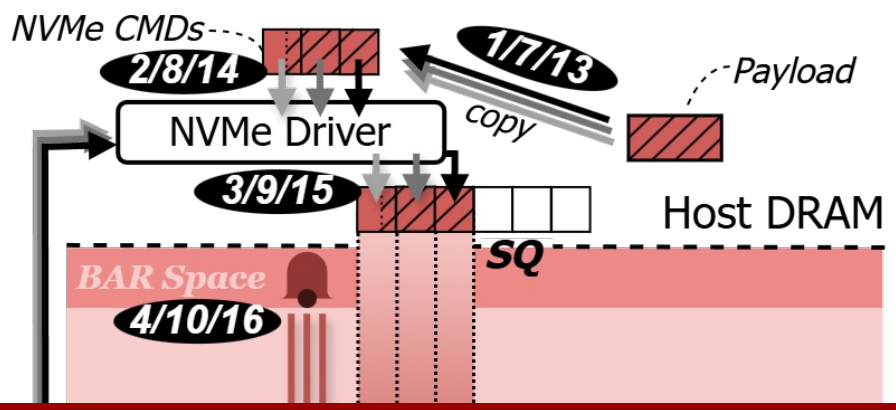


Existing Approaches Work! But…

Prior works have attempted to address this issue using two primary approaches:

(2) Embedding the payload inline within one or more custom NVMe CMDs and transmitting them in fragments.

- **Ex)** BandSlim [ICPP `24], [IEEE Micro `25]



This method loses scalability for larger payloads
due to repeated CMD generation and processing overhead caused by its mandatory serialization.

dozen bytes (e.g., 64 bytes), the cost of repeated CMD submission becomes a major performance bottleneck.

NAND Flash

Proposed Solution: ***ByteExpress***

In our paper, we present ByteExpress, a novel mechanism that efficiently transmits small payloads while avoiding the said drawbacks of existing solutions.

At the heart of ByteExpress is a concise yet powerful insight:

> ***"NVMe already enables fine-grained data delivery over PCIe."***
>
> ➔ This capability is embedded in the design of the NVMe SQ, where the host places a CMD in memory and the device performs a 64-byte DMA fetch from the SQ's tail.

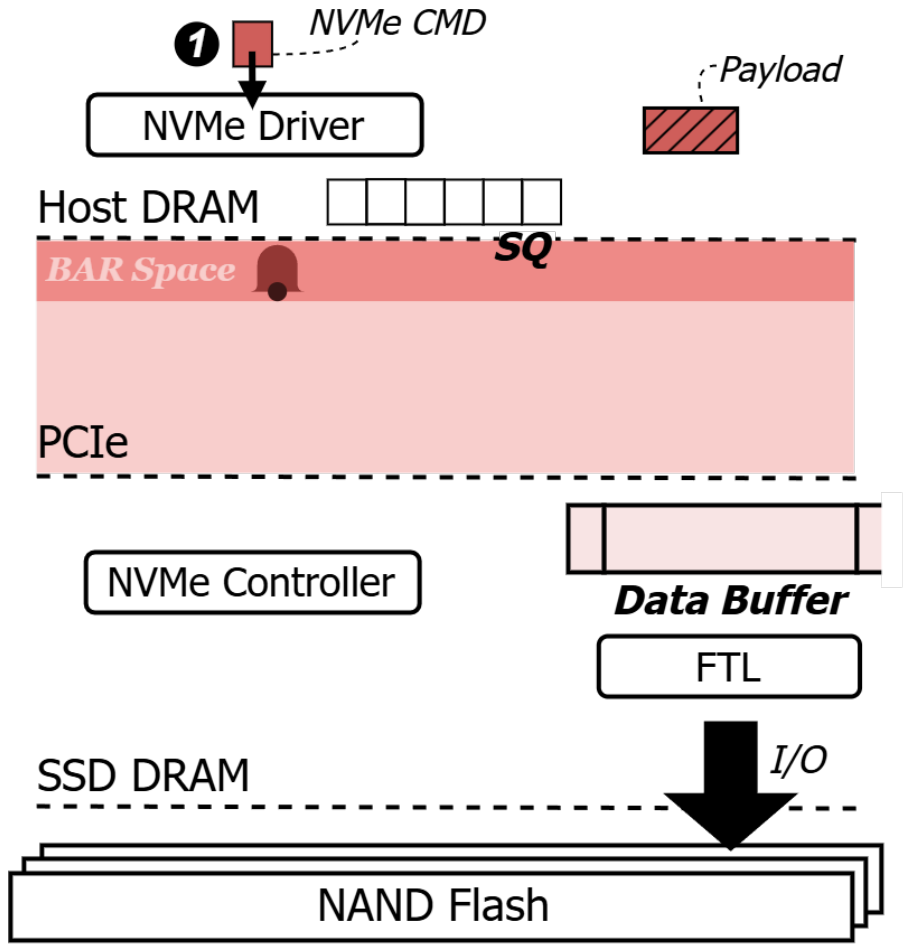
If we reinterpret the CMD itself as a payload or a portion of payload, this mechanism effectively becomes **a built-in fine-grained I/O path**.

➔ ByteExpress builds on this insight by placing the actual payload into the SQ in 64-byte chunks.

Proposed Solution: ***ByteExpress***

ByteExpress directly places the small payload, in 64-byte chunks, into the NVMe Submission Queue (SQ) after the CMD itself.

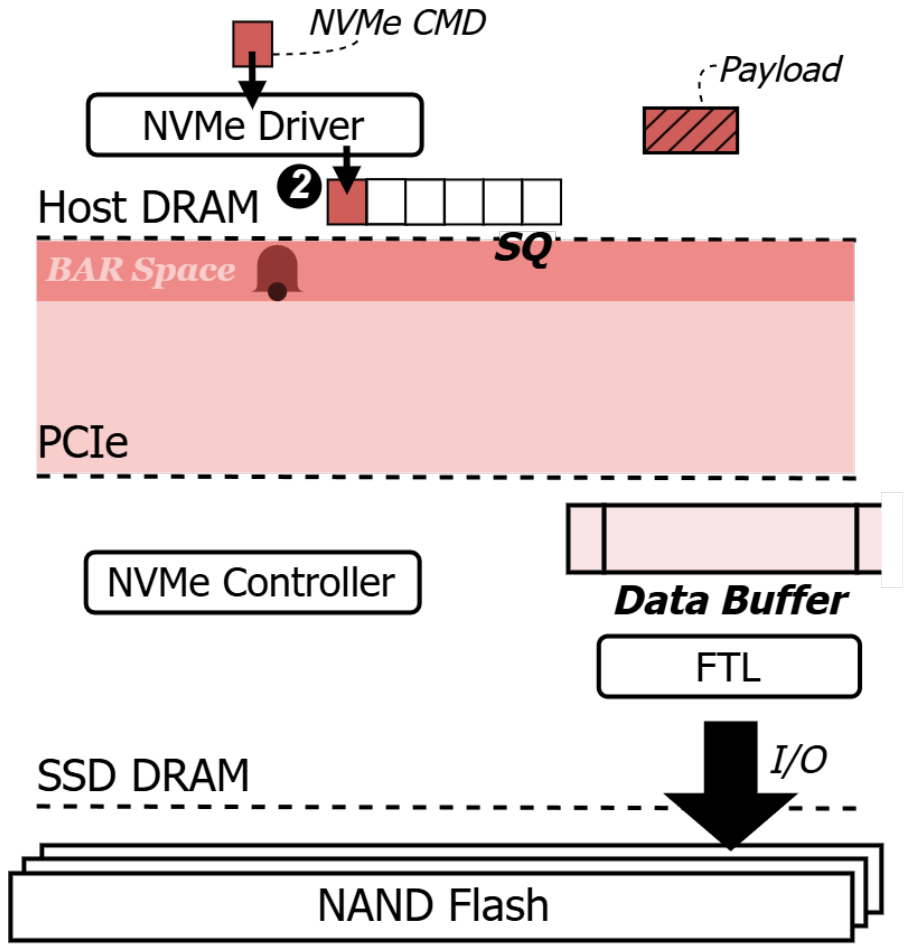
1. The NVMe CMD is issued to the NVMe driver **as usual**.



Proposed Solution: ***ByteExpress***

ByteExpress directly places the small payload, in 64-byte chunks, into the NVMe Submission Queue (SQ) after the CMD itself.

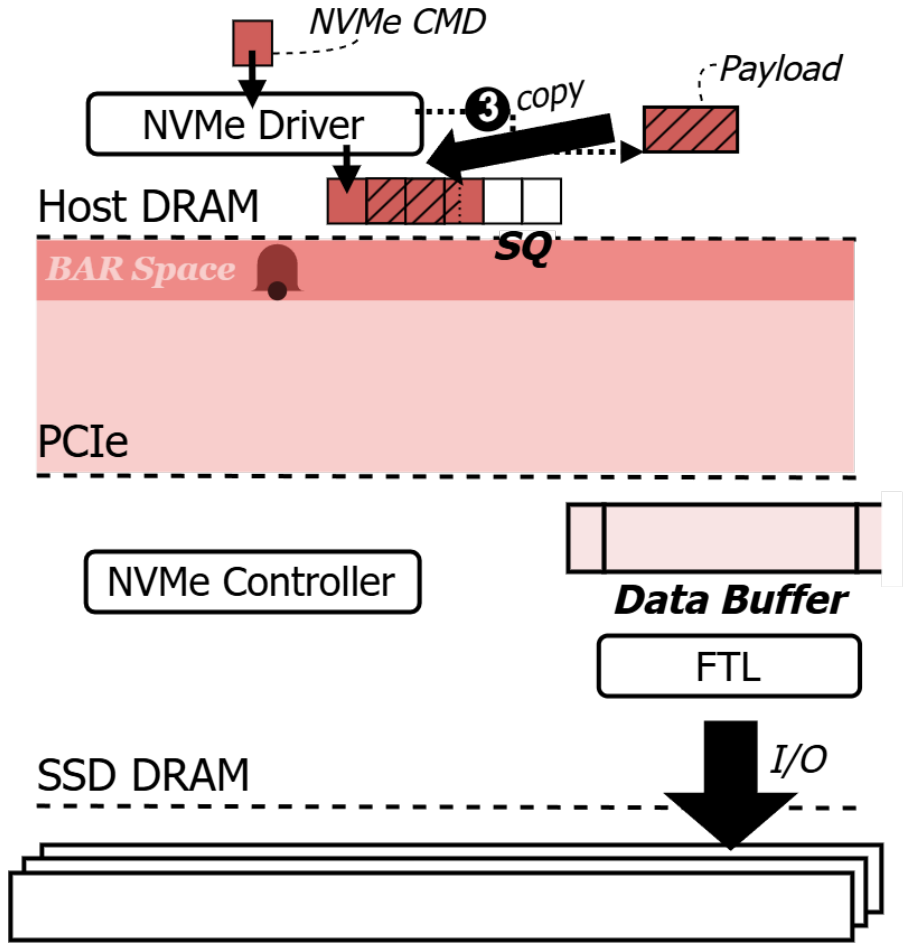
❷ The NVMe CMD is submitted to the SQ **as usual**.



Proposed Solution: ***ByteExpress***

ByteExpress directly places the small payload, in 64-byte chunks, into the NVMe Submission Queue (SQ) after the CMD itself.

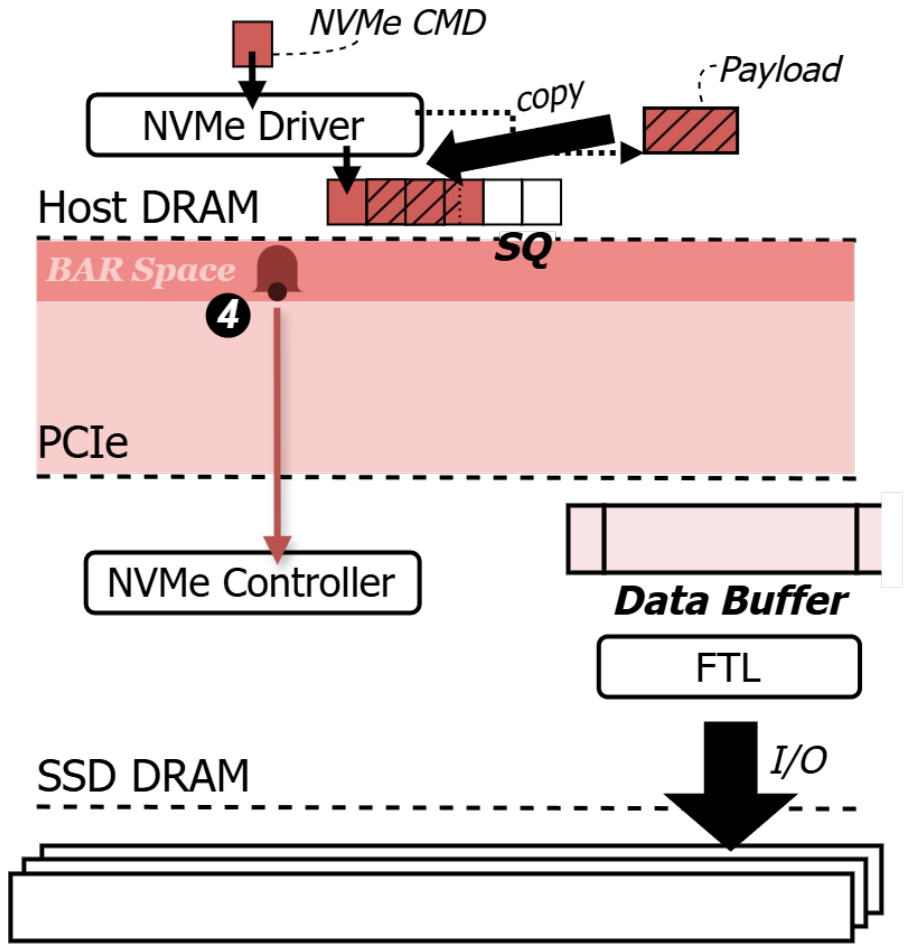
❸ (Here's where it differs) the **payload** is immediately **appended** to the SQ right after the just-submitted CMD, in 64B units.



Proposed Solution: ***ByteExpress***

ByteExpress directly places the small payload, in 64-byte chunks, into the NVMe Submission Queue (SQ) after the CMD itself.

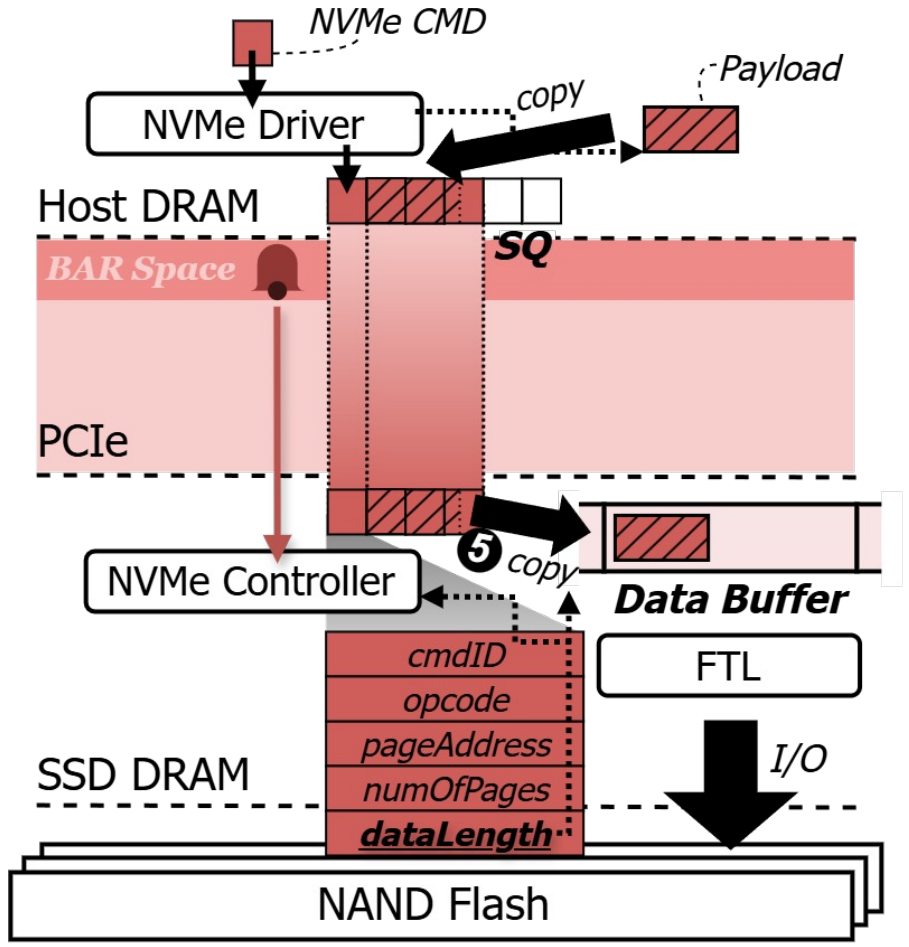
❹ The driver **rings the doorbell register** to notify the device of a new submission, just as usual.



Proposed Solution: ***ByteExpress***

ByteExpress directly places the small payload, in 64-byte chunks, into the NVMe Submission Queue (SQ) after the CMD itself.

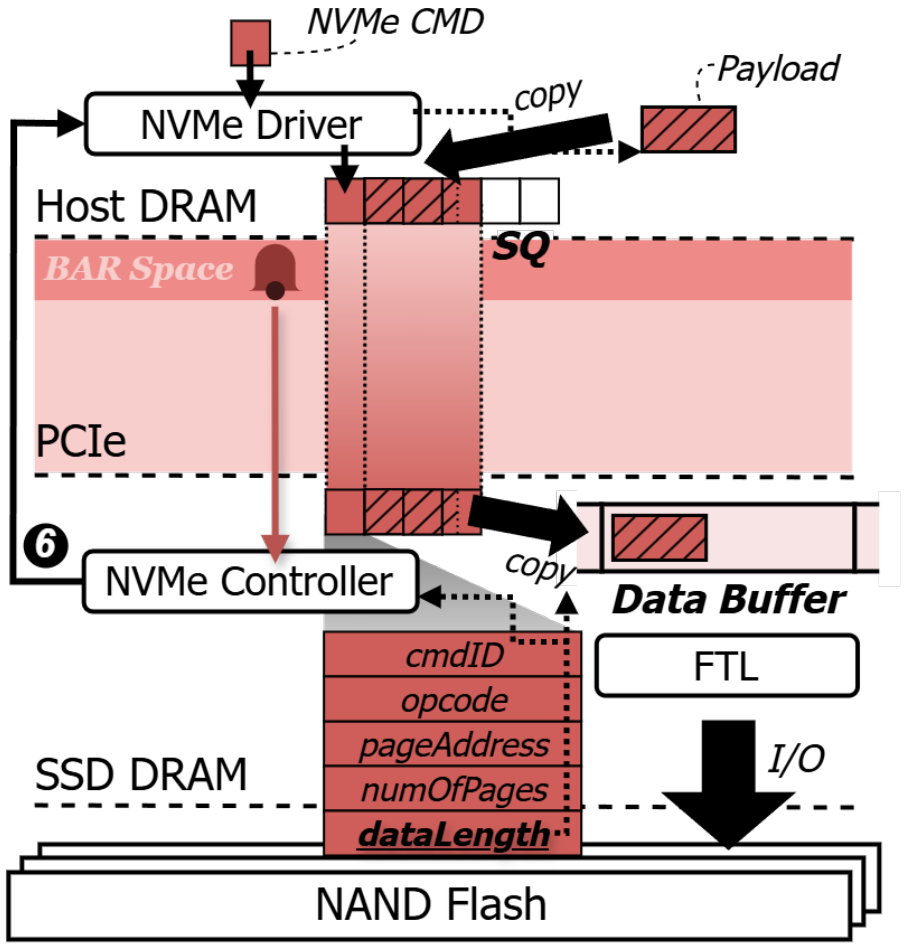
5 If the device detects ByteExpress semantics when fetching a CMD, it **fetches the following payload chunks from the same SQ** in *one shot* and copies them directly into the data buffer.



Proposed Solution: ***ByteExpress***

ByteExpress directly places the small payload, in 64-byte chunks, into the NVMe Submission Queue (SQ) after the CMD itself.

6 Finally, the device signals completion to the host, **as usual**.



ByteExpress – Key Design Challenges

Challenge#1: Identifying the Payload

- The NVMe driver already has full knowledge of the payload at submission time.
  - **The payload size** is encoded in the data length field of the NVMe I/O CMD.
  - **The address** is specified through the PRP entry fields during CMD construction.
    - The field values provided via the **nvme_passthru_cmd** structure are preserved in the corresponding **bio** request, which is fully accessible to the NVMe driver.

```
911         spin_lock(&nvmeq->sq_lock);
912
913         size_t data_len = blk_rq_bytes(req);
914         struct nvme_command *cmd = &iod->cmd;
915
916         if (cmd->common.opcode == 0xA8) { // ByteExpress Command
917                 int slots_needed = 1 + DIV_ROUND_UP(data_len, 64); // 1 for cmd + N payloads
918                 nvme_sq_copy_cmd(nvmeq, &iod->cmd);
919
920                 void *kaddr = bio_data(req->bio); // PRP-mapped buffer address
921 Address         if (!kaddr) {
922                         printk(KERN_ERR "Failed to get host buffer address from PRP\n");
923                         spin_unlock(&nvmeq->sq_lock);
924                         return BLK_STS_IOERR;
925                 }
```

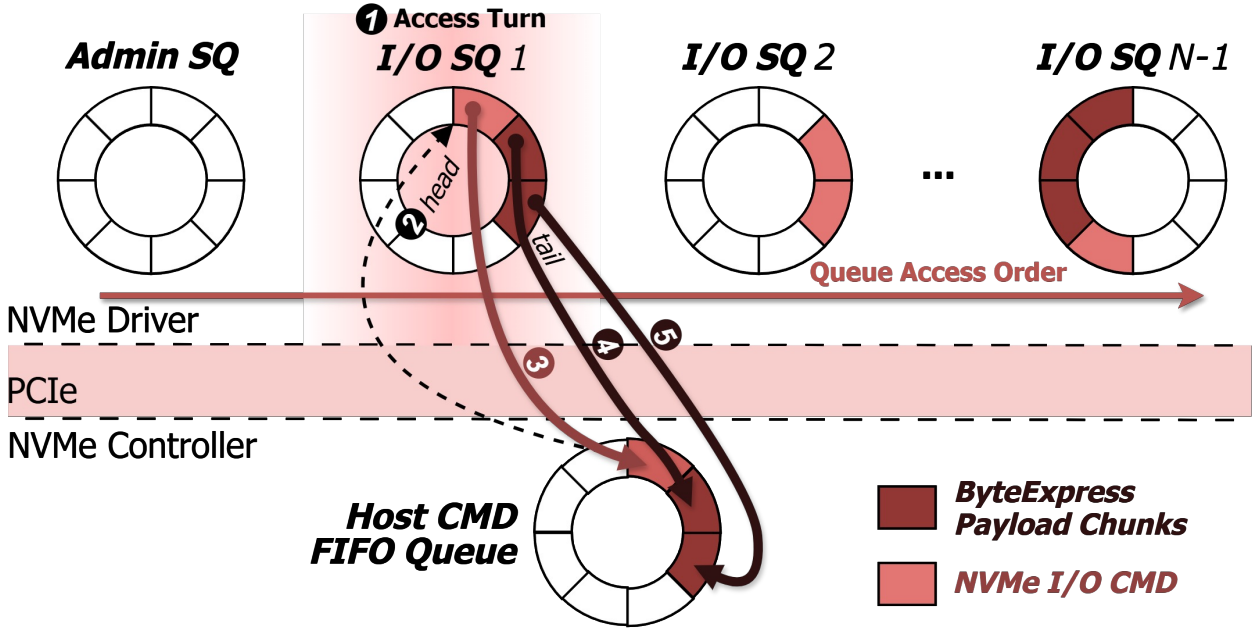
Payload Size

ByteExpress-Extended Linux Kernel NVMe Driver
open-sourced at https://github.com/junttang/byteexpress

ByteExpress – Key Design Challenges

Challenge#2: Preserving Data Ordering

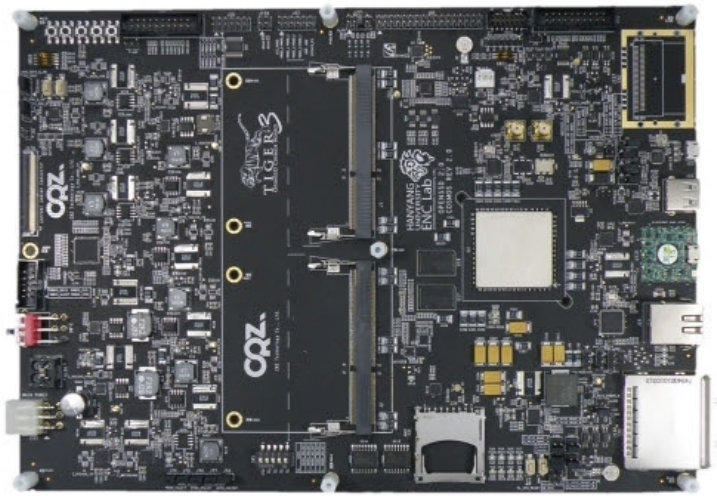
- In the current HotStorage version, ByteExpress preserves ordering between the NVMe CMD and its associated 64-byte payload chunks through two complementary mechanisms.
  - **First,** on the host side, the NVMe driver guarantees exclusive access to each SQ using spin locks. We leverage this by inserting both the CMD and its payload chunks while holding the lock.
  - **Second,** on the device side, the controller preserves inter-SQ ordering by fetching subsequent entries exclusively from the same SQ once a ByteExpress-applied CMD is detected



ByteExpress – Evaluation

We implemented* and evaluated ByteExpress on a Linux host machine connected to the Cosmos+ OpenSSD platform (PCIe Gen2 8-lane connection).

- **Host Machine**
  - 64-Core Intel Xeon Gold 6226R CPU
  - 384 GB of DDR4 Memory
- **Cosmos+ OpenSSD**
  - Xilinx Zynq-7000 FPGA
  - 1 TB NAND Flash
  - 1 GB On-Board DDR3 Memory

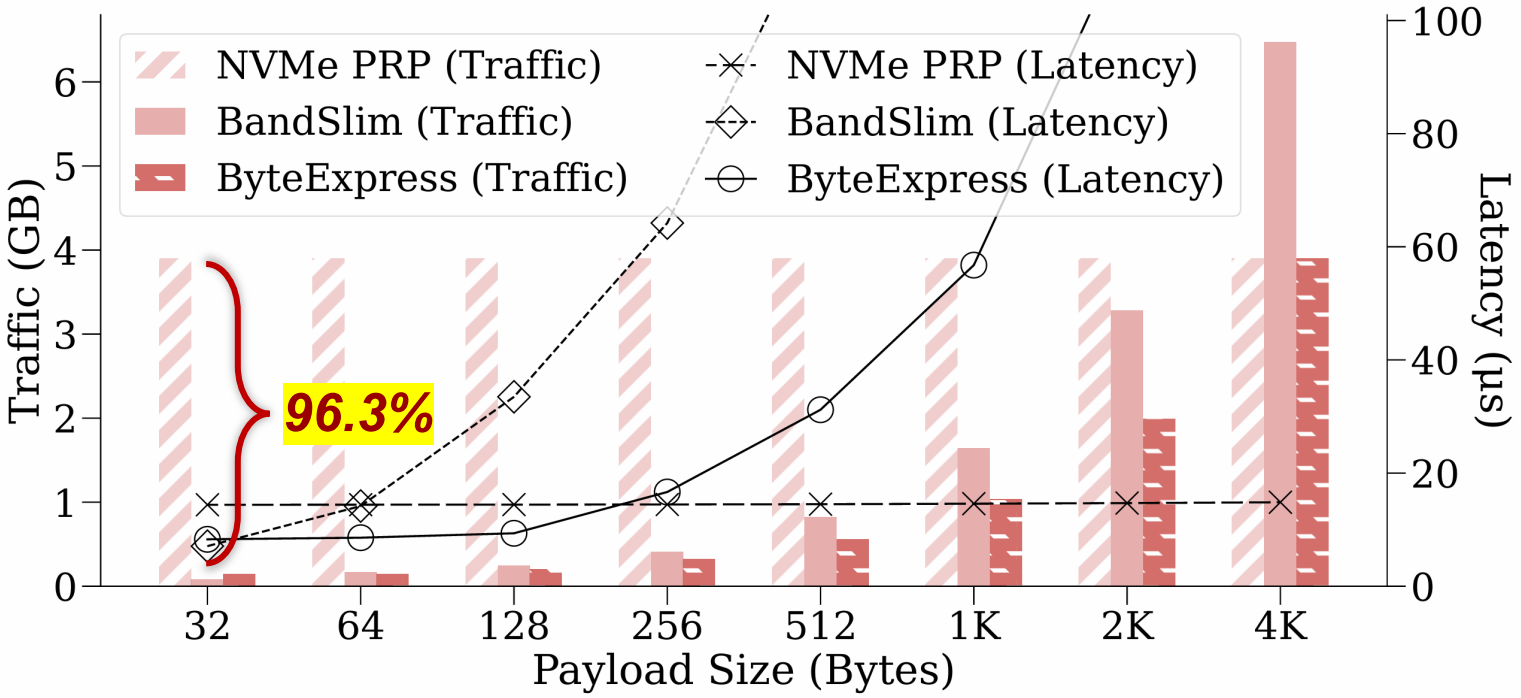


Cosmos+ OpenSSD Platform
a widely-used open-source SSD prototype board

* Open-source github repository: https://github.com/junttang/byteexpress

ByteExpress – Benefits

We evaluated PCIe traffic* and transfer latency (with NAND I/O disabled on the OpenSSD) across various payload sizes using NVMe passthrough, issuing 1 million writes per configuration with NVMe PRP, BandSlim, and ByteExpress.



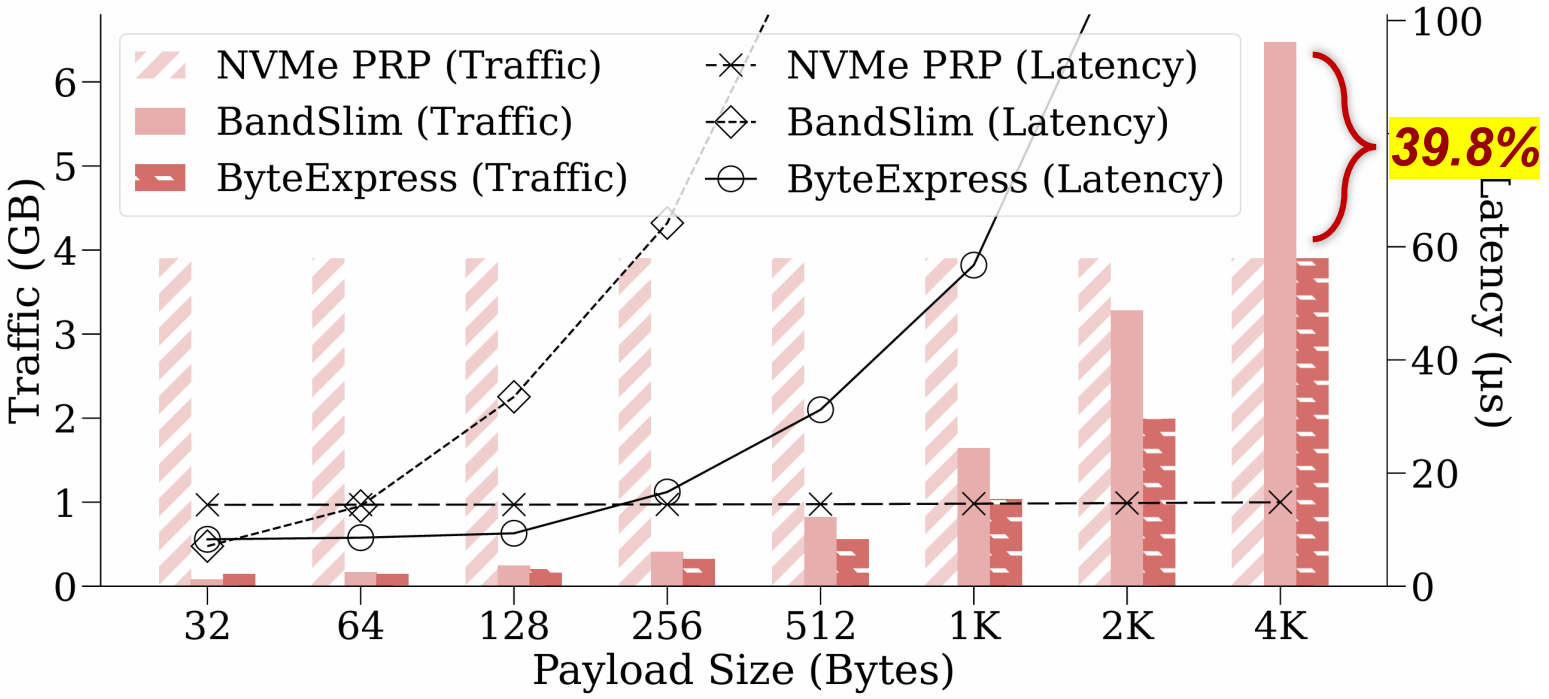
Both ByteExpress and BandSlim **significantly reduced PCIe traffic** for payloads smaller than 4 KB compared to PRP.

➔ **Up to 96.3% of reduction**

* PCIe traffic was measured via Intel Performance Counter Monitor (PCM)

ByteExpress – Benefits

We evaluated PCIe traffic* and transfer latency (with NAND I/O disabled on the OpenSSD) across various payload sizes using NVMe passthrough, issuing 1 million writes per configuration with NVMe PRP, BandSlim, and ByteExpress.

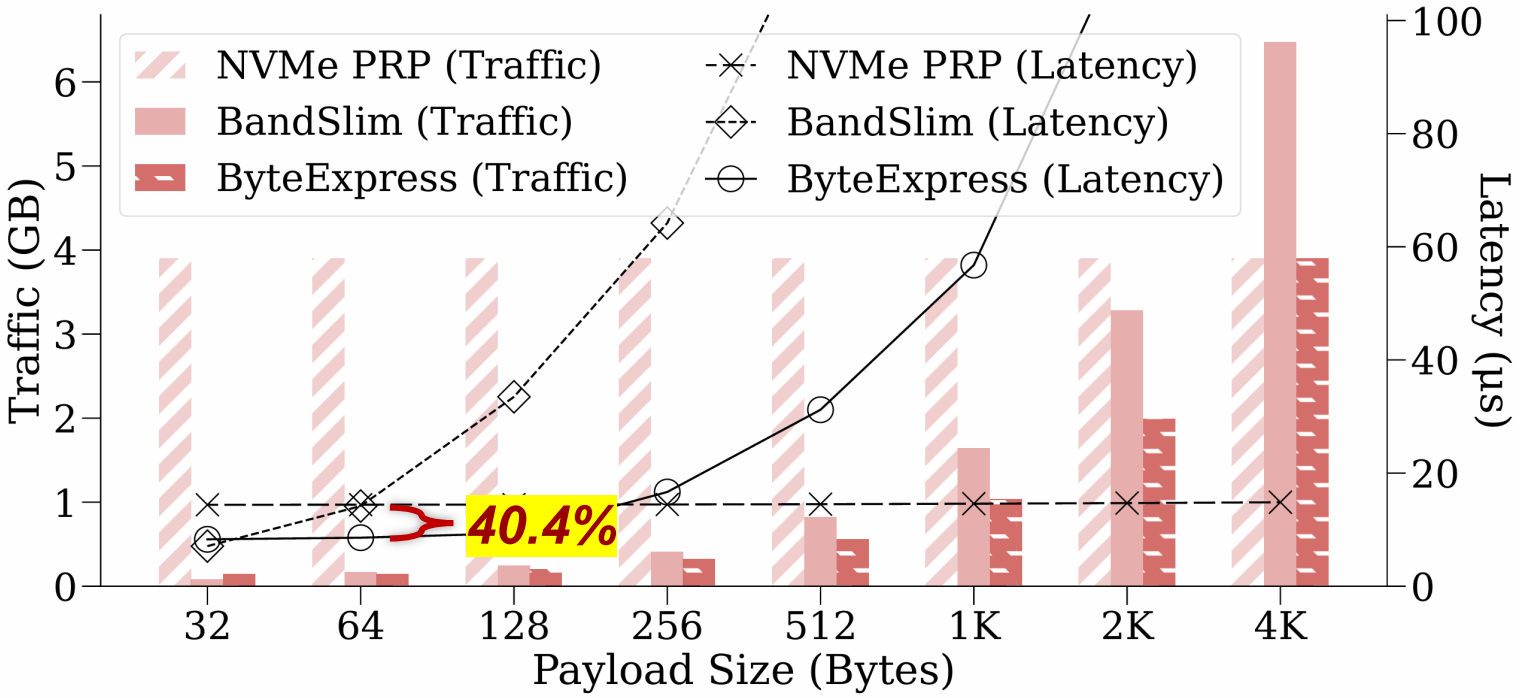


Interestingly, in the 64-byte to 4 KB range, ByteExpress outperformed BandSlim by **up to 39.8%** in traffic reduction.

* PCIe traffic was measured via Intel Performance Counter Monitor (PCM)

ByteExpress – Benefits

We evaluated PCIe traffic* and transfer latency (with NAND I/O disabled on the OpenSSD) across various payload sizes using NVMe passthrough, issuing 1 million writes per configuration with NVMe PRP, BandSlim, and ByteExpress.

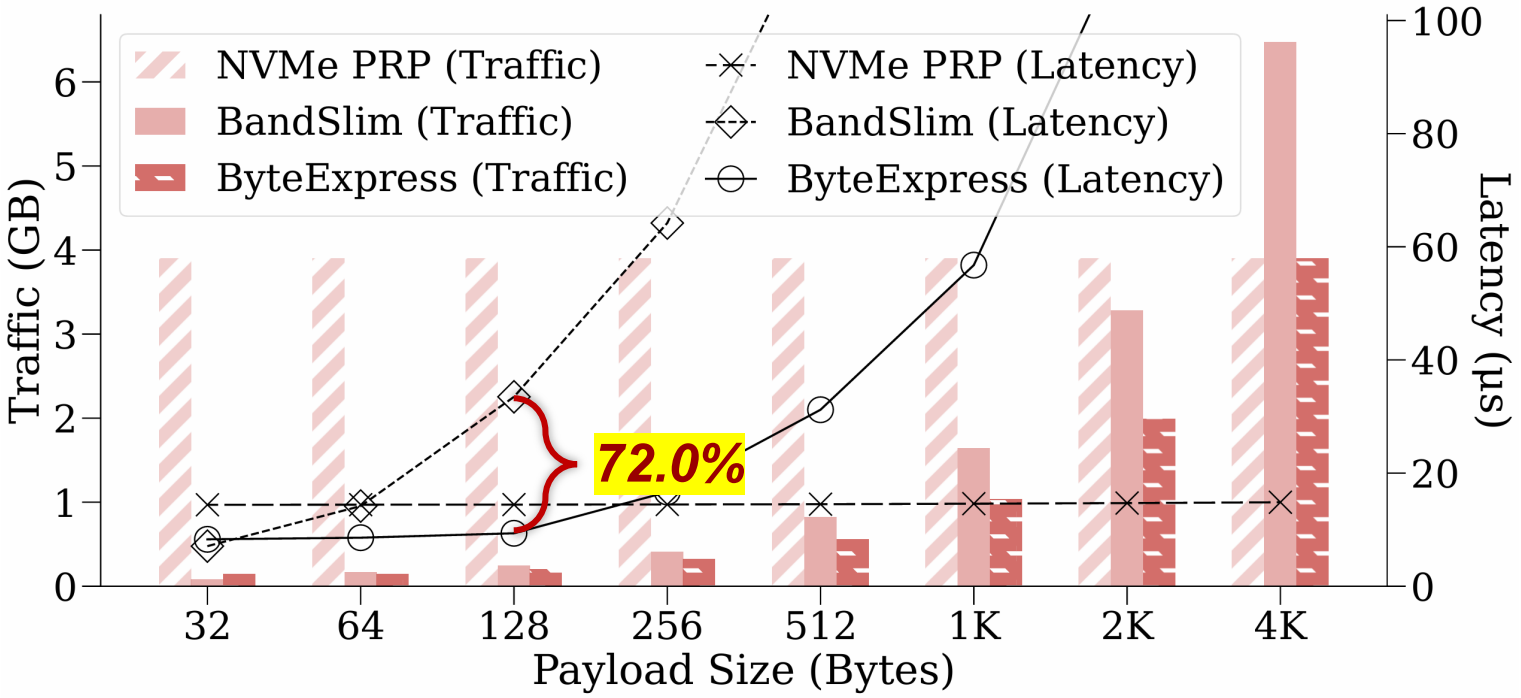


For transfer latency, ByteExpress reduced latency by **up to 40.4%** over NVMe PRP in the 32–128 byte range.

* PCIe traffic was measured via Intel Performance Counter Monitor (PCM)

ByteExpress – Benefits

We evaluated PCIe traffic* and transfer latency (with NAND I/O disabled on the OpenSSD) across various payload sizes using NVMe passthrough, issuing 1 million writes per configuration with NVMe PRP, BandSlim, and ByteExpress.

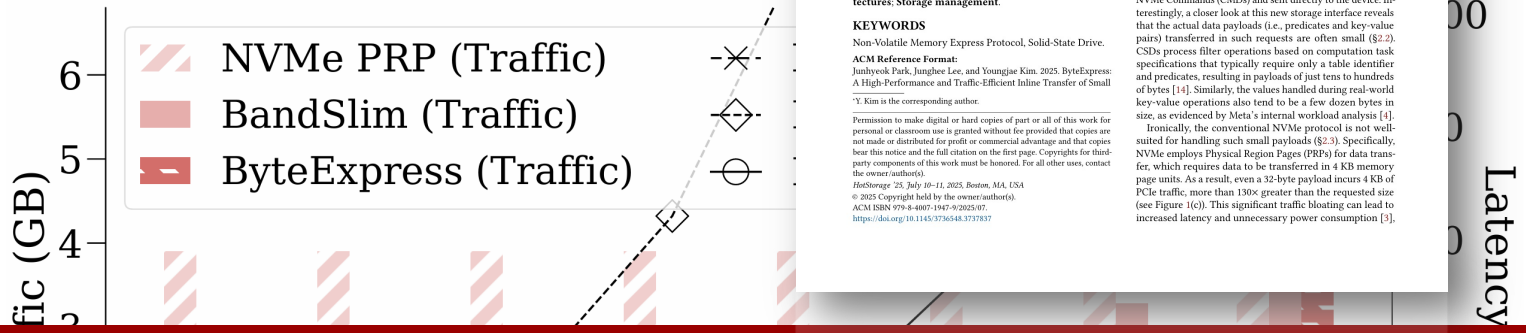


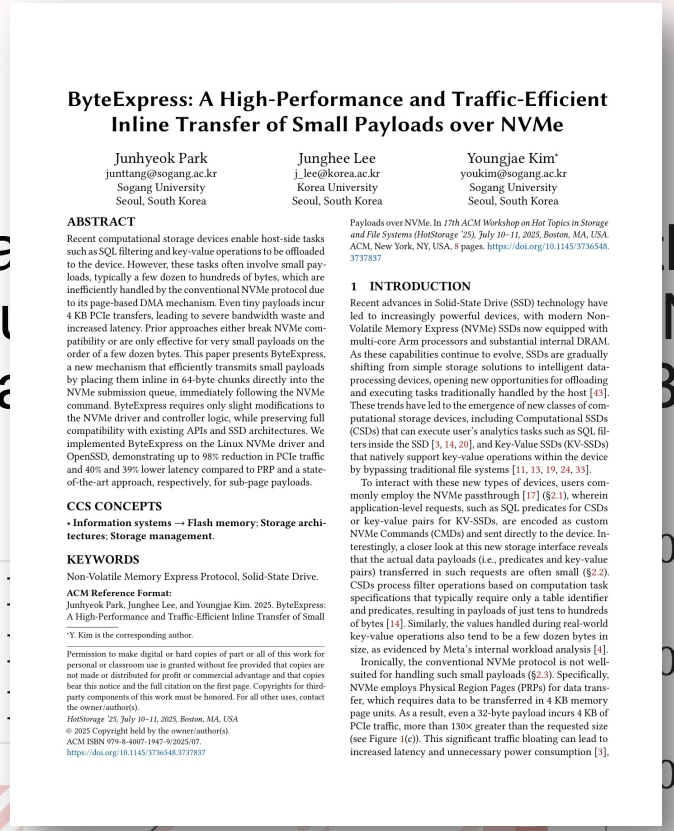
Furthermore, ByteExpress outperformed BandSlim beyond 64 bytes, for instance, achieving a **72% reduction** at 128 bytes.

* PCIe traffic was measured via Intel Performance Counter Monitor (PCM)

ByteExpress

We evaluated PCIe traffic* a h NAND I/O disabled on
the OpenSSD) across variou NVMe passthrough, issuing
1 million writes per configura BandSlim, and ByteExpress.





ByteExpress: A High-Performance and Traffic-Efficient Inline Transfer of Small Payloads over NVMe

Junhyeok Park
junttang@sogang.ac.kr
Sogang University
Seoul, South Korea

Junghee Lee
j_lee@korea.ac.kr
Korea University
Seoul, South Korea

Youngjae Kim*
youkim@sogang.ac.kr
Sogang University
Seoul, South Korea

ABSTRACT

Recent computational storage devices enable host-side tasks such as SQL filtering and key-value operations to be offloaded to the device. However, these tasks often involve small payloads, typically a few dozen to hundreds of bytes, which are inefficiently handled by the conventional NVMe protocol due to its page-based DMA mechanism. Even tiny payloads incur 4 KB PCIe transfers, leading to severe bandwidth waste and increased latency. Prior approaches either break NVMe compatibility or are only effective for very small payloads on the order of a few dozen bytes. This paper presents ByteExpress, a new mechanism that efficiently transmits small payloads by placing them inline in 64-byte chunks directly into the NVMe submission queue, immediately following the NVMe command. ByteExpress requires only slight modifications to the NVMe driver and controller logic, while preserving full compatibility with existing APIs and SSD architectures. We implemented ByteExpress on the Linux NVMe driver and OpenSSD, demonstrating up to 98% reduction in PCIe traffic and 40% and 39% lower latency compared to PRP and a state-of-the-art approach, respectively, for sub-page payloads.

CCS CONCEPTS

• **Information systems** → **Flash memory**; **Storage architectures**; **Storage management**.

KEYWORDS

Non-Volatile Memory Express Protocol, Solid-State Drive.

ACM Reference Format:

Junhyeok Park, Junghee Lee, and Youngjae Kim. 2025. ByteExpress: A High-Performance and Traffic-Efficient Inline Transfer of Small Payloads over NVMe. In *17th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage '25), July 10–11, 2025, Boston, MA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3736548.3737837

*Y. Kim is the corresponding author.

HotStorage '25, July 10–11, 2025, Boston, MA, USA

ACM ISBN 979-8-4007-1947-9/2025/07.
https://doi.org/10.1145/3736548.3737837

1 INTRODUCTION

Recent advances in Solid-State Drive (SSD) technology have led to increasingly powerful devices, with modern Non-Volatile Memory Express (NVMe) SSDs now equipped with multi-core Arm processors and substantial internal DRAM. As these capabilities continue to evolve, SSDs are gradually shifting from simple storage solutions to intelligent data-processing devices, opening new opportunities for offloading and executing tasks traditionally handled by the host [43]. These trends have led to the emergence of new classes of computational storage devices, including Computational SSDs (CSDs) that can execute user's analytics tasks such as SQL filters inside the SSD [3, 14, 20], and Key-Value SSDs (KV-SSDs) that natively support key-value operations within the device by bypassing traditional file systems [11, 13, 19, 24, 33].

To interact with these new types of devices, users commonly employ the NVMe passthrough [17] (§2.1), wherein application-level requests, such as SQL predicates for CSDs or key-value pairs for KV-SSDs, are encoded as custom NVMe Commands (CMDs) and sent directly to the device. Interestingly, a closer look at this new storage interface reveals that the actual data payloads (i.e., predicates and key-value pairs) transferred in such requests are often small (§2.2). CSDs process filter operations based on computation task specifications that typically require only a table identifier and predicates, resulting in payloads of just tens to hundreds of bytes [14]. Similarly, the values handled during real-world key-value operations also tend to be a few dozen bytes in size, as evidenced by Meta's internal workload analysis [4].

Ironically, the conventional NVMe protocol is not well-suited for handling such small payloads (§2.3). Specifically, NVMe employs Physical Region Pages (PRPs) for data transfer, which requires data to be transferred in 4 KB memory page units. As a result, even a 32-byte payload incurs 4 KB of PCIe traffic, more than 130× greater than the requested size (see Figure 1(c)). This significant traffic bloating can lead to increased latency and unnecessary power consumption [3],

Furthermore, ByteExpress outperformed BandSlim beyond 64 bytes, for

You can find more design and evaluation details in our paper.

We'd love for you to check it out!

* PCIe traffic was measured via Intel Performance Counter Monitor (PCM)

Conclusion

Summary

- ByteExpress offers a lightweight, practical solution to the long-standing inefficiency of small, direct data transfer over NVMe.
- By repurposing the NVMe submission queue to carry 64-byte payload chunks, ByteExpress enables efficient, fine-grained small-data transmission without modifying SSD-internal architecture or NVMe passthrough-based APIs.

Future Work

- We are planning to extend our work to support read operations by leveraging the NVMe completion queue in a similar fashion.
- Additionally, we aim to enhance the design by removing the queue-level confinement and enabling parallel chunk transfers.
- A comprehensive comparison with SGL under various PCIe link speeds is also underway.

Thank You

Q&A

Presenter: Junhyeok Park

Contact: *junttang@sogang.ac.kr*