NVMeVirt:
A Versatile Software-defined Virtual NVMe Device

NVRAMOS'23

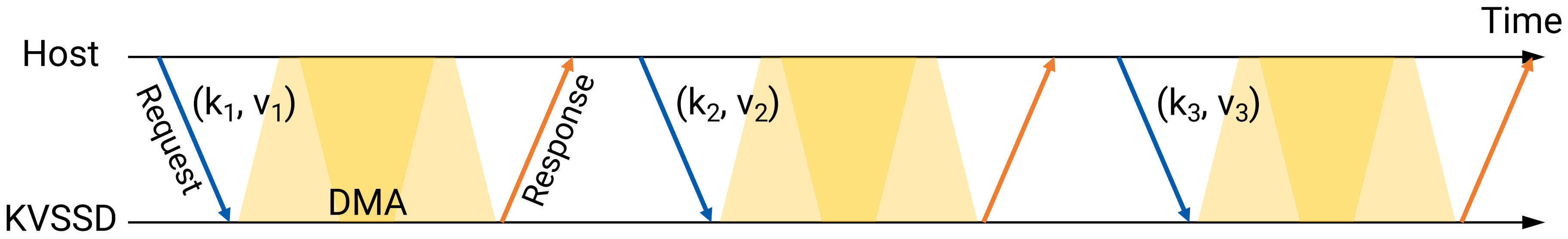
Sang-Hoon Kim*, Jaehoon Shim†, Euidong Lee†
Seongyeop Jeong†, Ilkueon Kang†, Jin-Soo Kim†

* Ajou University (아주대학교, 1973)

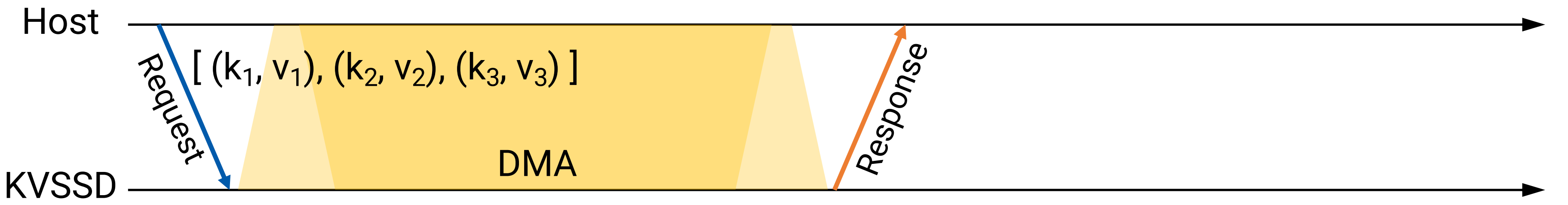
† VERITAS LUX MEA

Once upon a time in our research…

- We were evaluating a key-value SSD
- Found each KV operation is independently processed
  - High interfacing overhead for small KV operations



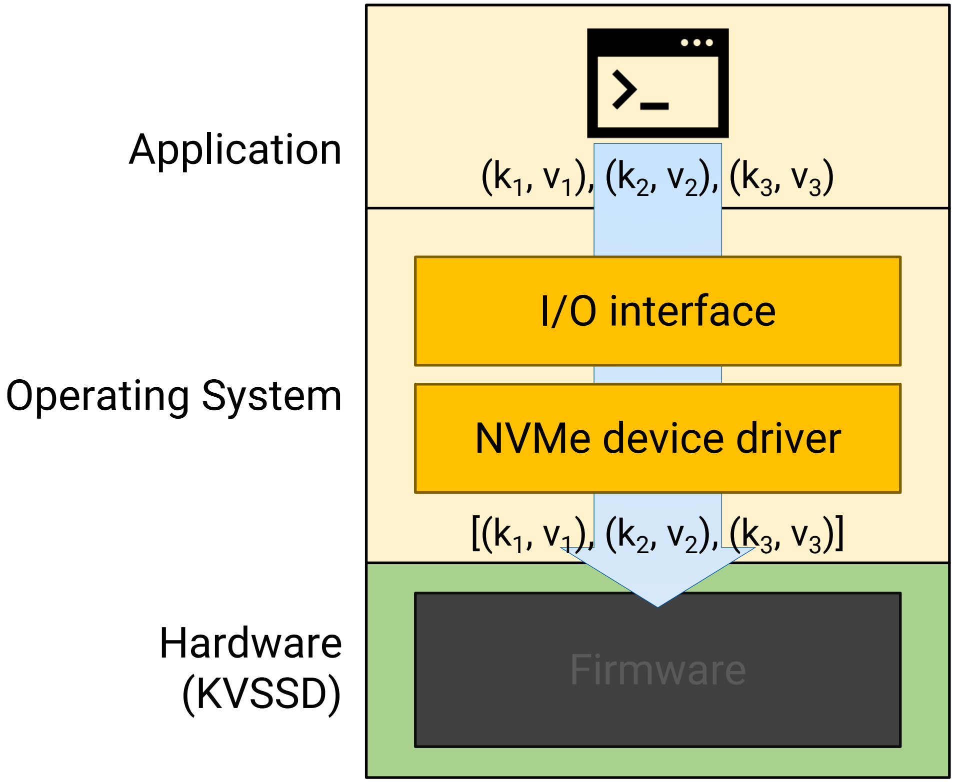
- What if we can gather multiple KV operations in a single command?

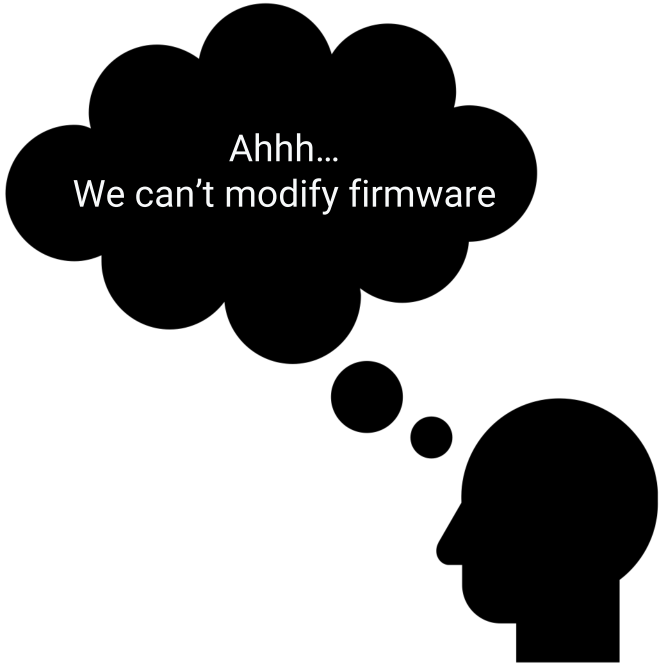


[Transaction Support Using Compound Commands in Key-Value SSDs (HotStorage'19)]

Once upon a time in our research…

- Turned out that we should change the firmware of KVSSD, which was beyond our control
  - Code availability, engineering efforts, research resources, legal matter, …





Innovations around NVMe

- Multi-stream
- OpenChannel SSD
- KV-SSD
- ZNS SSD
- CMB
- IO Determinism (TP4003c)
- PMR (TP4032)
- NVM Sets (TP4052c)
- FDP (TP4146)
- SPDK
- NVMe over Fabric (NIC-SSD)
- GPUDirect Storage (GPU-SSD)
- SmartSSD (FPGA-SSD)
- Computational Storage (TP4091)
- ...

1973
AJOU UNIV.

Problems

- Building a hardware prototype requires considerable time and engineering effort
- The existing hardware prototyping tools are inflexible and hard to get adapted to the fast evolution
- Requirements may change at any time
- Trade-offs should be assessed swiftly
- Want to evaluate performance impact by running real applications

- What about using an **emulator?**

Dilemma of Emulator

- Emulators can facilitate advanced storge research by ***actualizing*** novel device concepts
  - Open-Channel SSD, NVM SSD, KVSSD, Zoned Namespace (ZNS) SSD, computational storage, …
  - Can implement the concepts in software
    - No need to wait until they become available at retailor shops
    - $$$

- Cannot support some I/O models and storage configurations that are frequently used for building modern storage systems

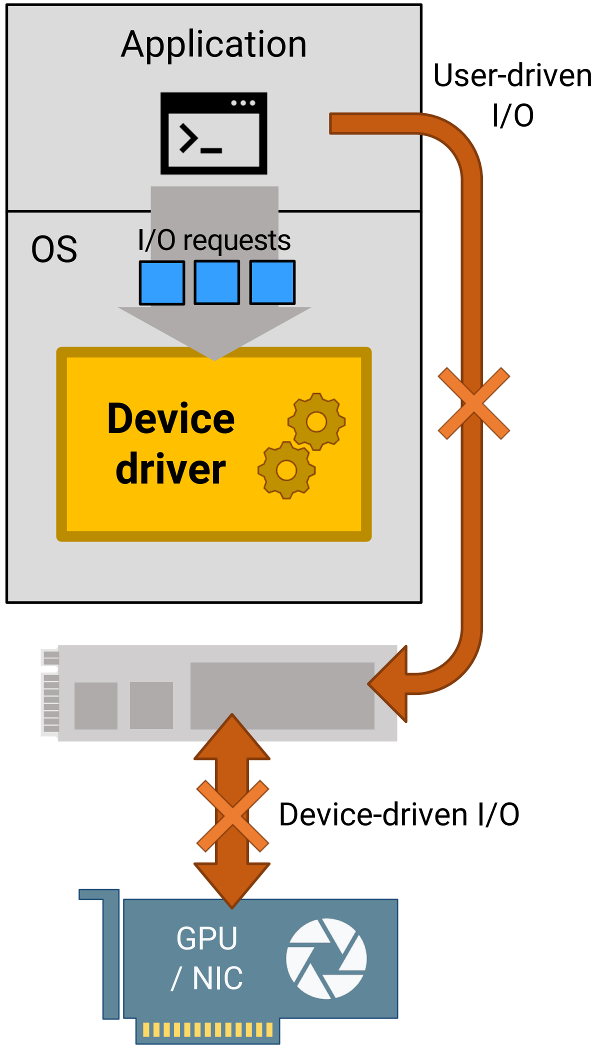
Previous: Device Driver-level Approaches

- Catch I/O requests at the block/NVMe device driver and emulate the requests
  - DavidFAST11, FlexDriveHPCC16, …

- Can only process ‘*regular*’ I/O requests

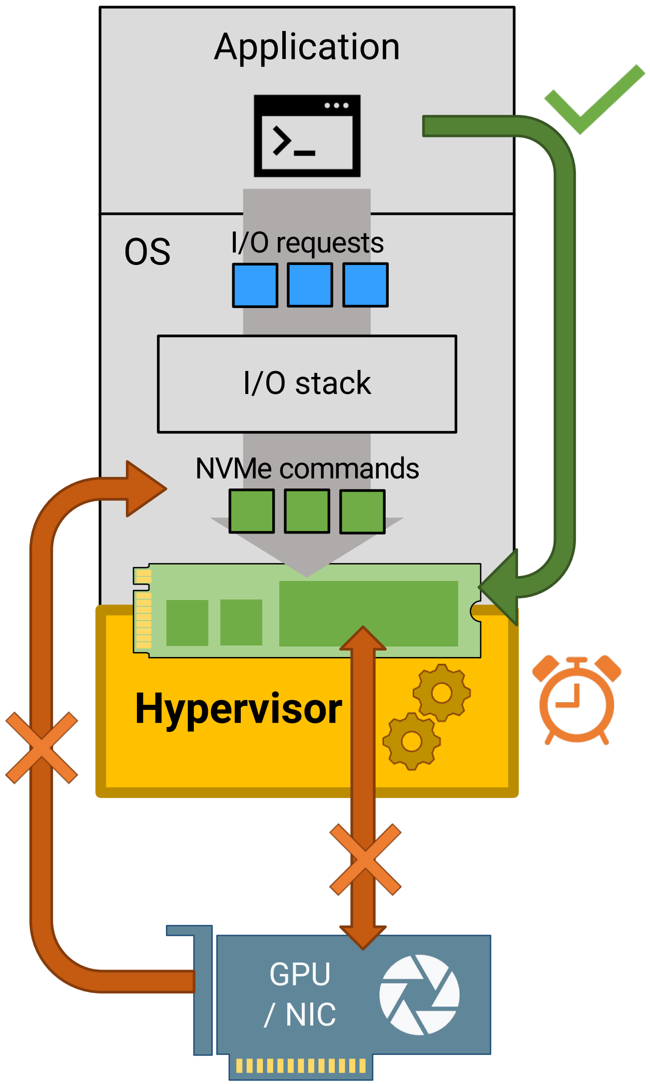
- Unable to support user-driven I/O: Kernel bypassing with SPDK

- Neither for device-driven I/O
  - RDMA target for NVMe-oF, PCI peer-to-peer DMA



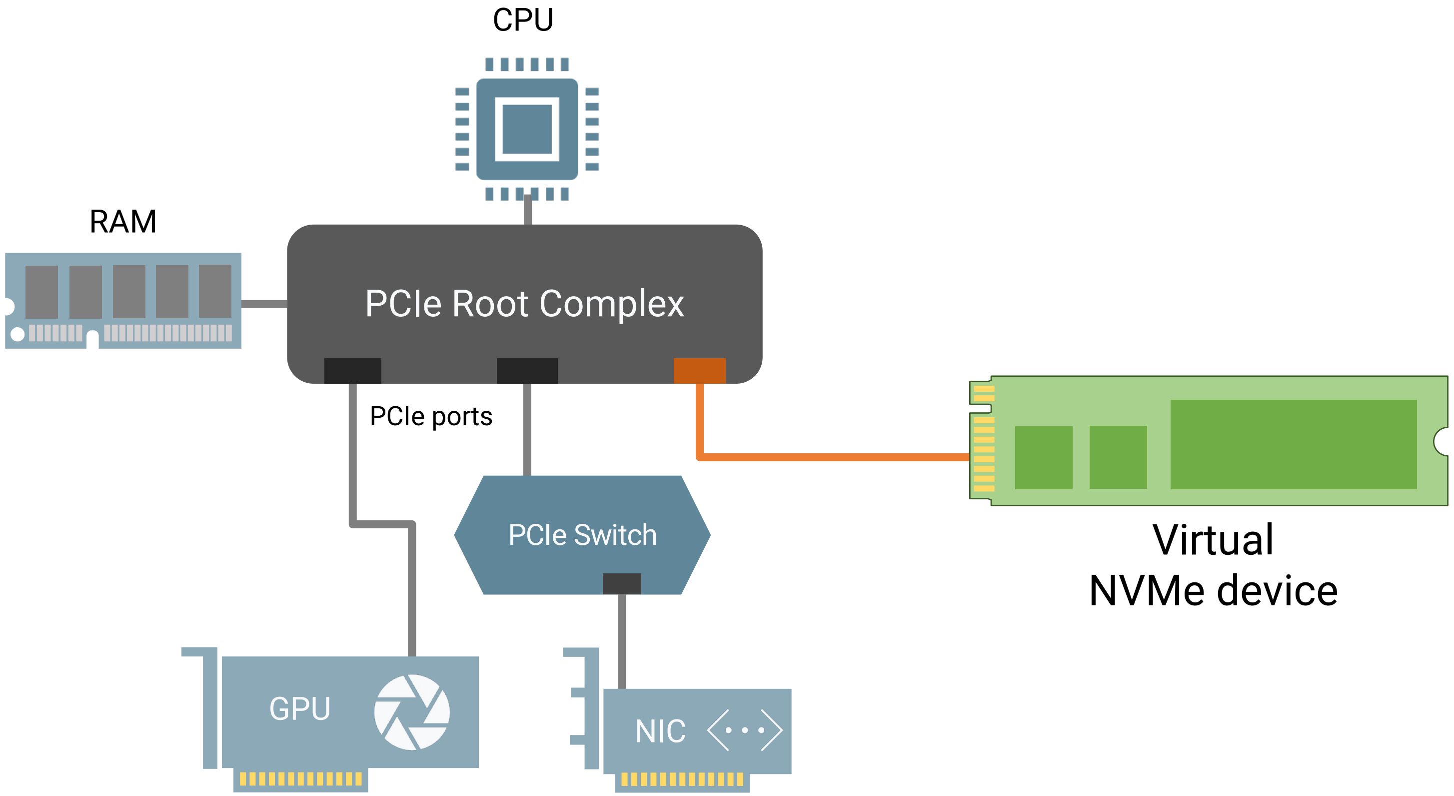
Previous: Virtualization-based Approaches

- Hypervisor emulates a virtual device exposed to the guest OS
  - VSSIM[MSST13], FEMU[FAST18], ZNS+[OSDI21] , …
- Can support the user-driven I/O
- Cannot support device-driven I/O configurations
  - No way to contact the virtual device from real devices on the host
  - Complicated memory layout in VM environments makes RDMA infeasible
- Virtualization overhead limits and/or impacts on the performance characteristics of target devices



NVMeVirt: Virtual NVMe Device in Software

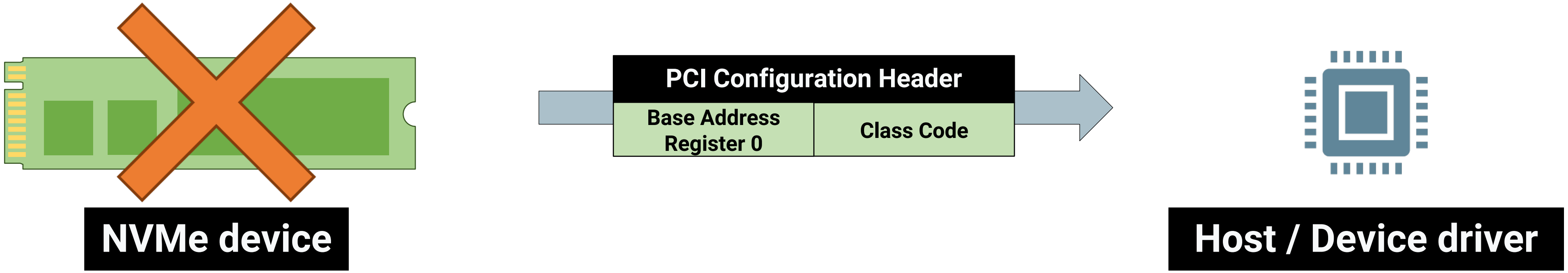
- A light-weight kernel module that presents **a native NVMe device** to the **entire system**
  - Support any storage configurations!



- Conventional SSD
- NVM SSD
- ZNS SSD
- KVSSD

Challenges for Virtual PCI/NVMe Devices

- Challenge 1: How to create a virtual PCI device instance in the system
  - The real device initiates the initialization
  - We don't have the physical device that can initiate the initialization
  - We don't want to mess up with the existing PCI subsystem implementation



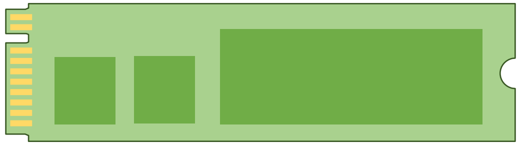
Challenges for Virtual PCI/NVMe Devices

- Challenge 1: How to create a virtual PCI device instance in the system
  - The real device initiates the initialization
  - We don't have the physical device that can initiate the initialization
  - We don't want to mess up with the existing PCI subsystem implementation

- Solution: Make a PCI device instance indirectly through PCI bus
  - Create a virtual PCI bus that presents the PCI configuration header of virtual device to the PCI subsystem
  - No modification is needed in the Linux kernel

Challenges for Virtual PCI/NVMe Devices

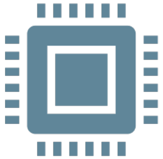
- Challenge 2: Cannot rely on the PCI mechanism to detect the requests from the host-side
  - Updates to the control block and doorbells are notified to the device as PCI transactions



NVMe device

	NVMe Control Block	
00h	Controller Capabilities	
08h	Version	Interrupt Mask Set
10h	Interrupt Mask Clear	Controller Configuration
18h	Reserved	Controller Status
20h	Subsystem Reset	Admin Q Attributes
28h	Admin SQ Base Address	
30h	Admin CQ Base Address	
38h	CMB Location	CMB Size

	Doorbells			
1000h				
1010h				
1020h				

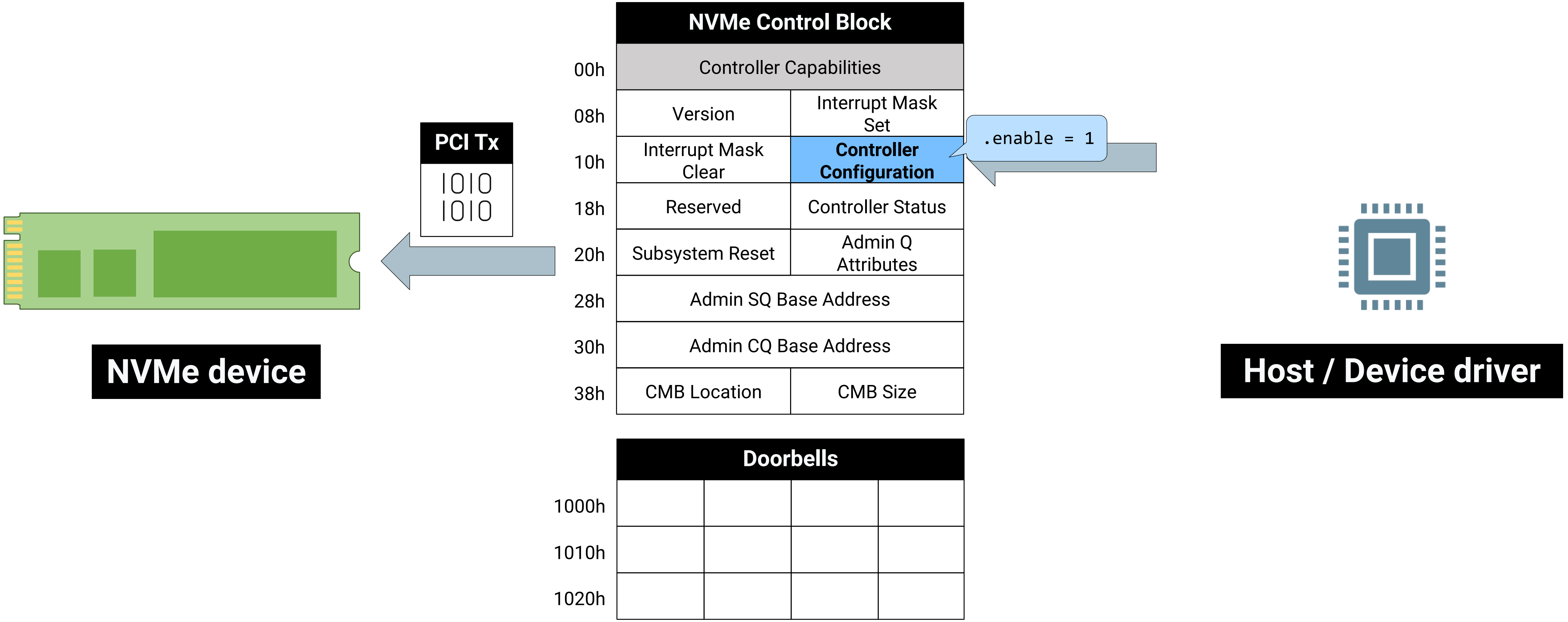


Host / Device driver

Device memory mapped to the host's address space

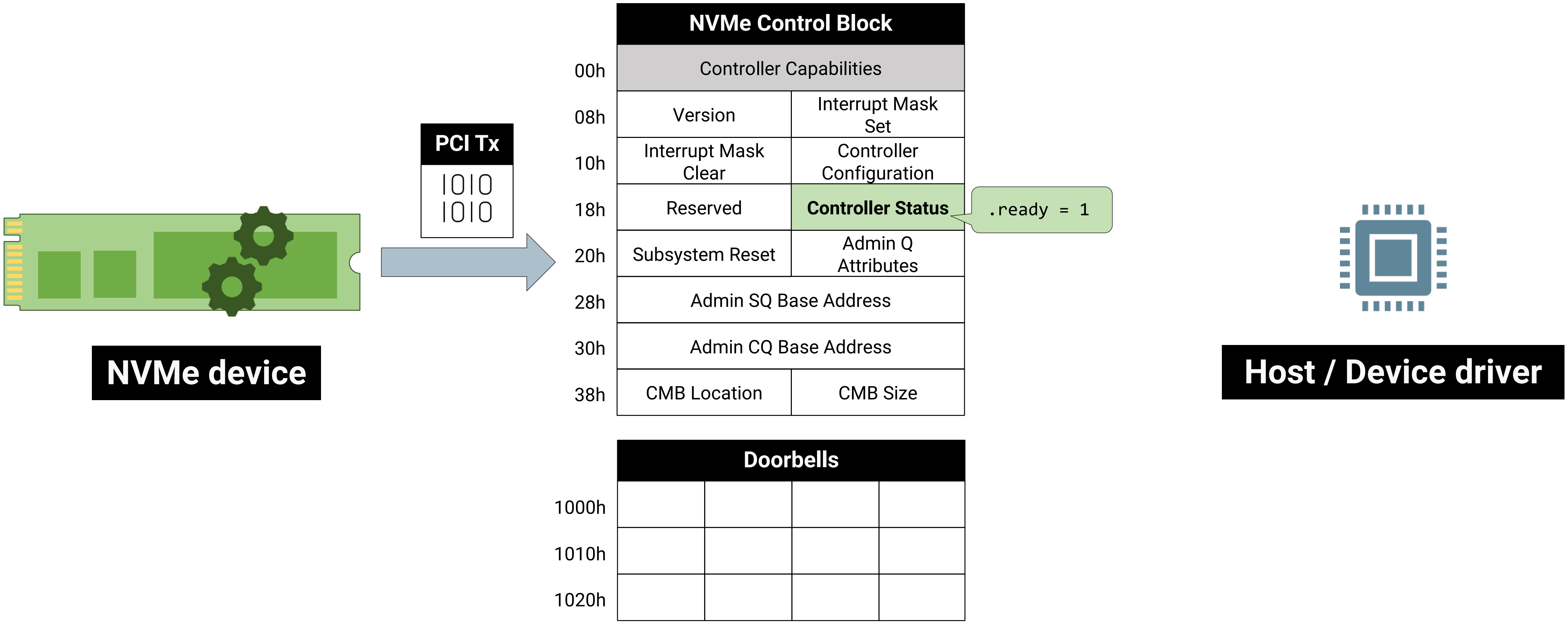
Challenges for Virtual PCI/NVMe Devices

- Challenge 2: Cannot rely on the PCI mechanism to detect the requests from the host-side
  - Updates to the control block and doorbells are notified to the device as PCI transactions



Challenges for Virtual PCI/NVMe Devices

- Challenge 2: Cannot rely on the PCI mechanism to detect the requests from the host-side
  – Updates to the control block and doorbells are notified to the device as PCI transactions



Challenges for Virtual PCI/NVMe Devices

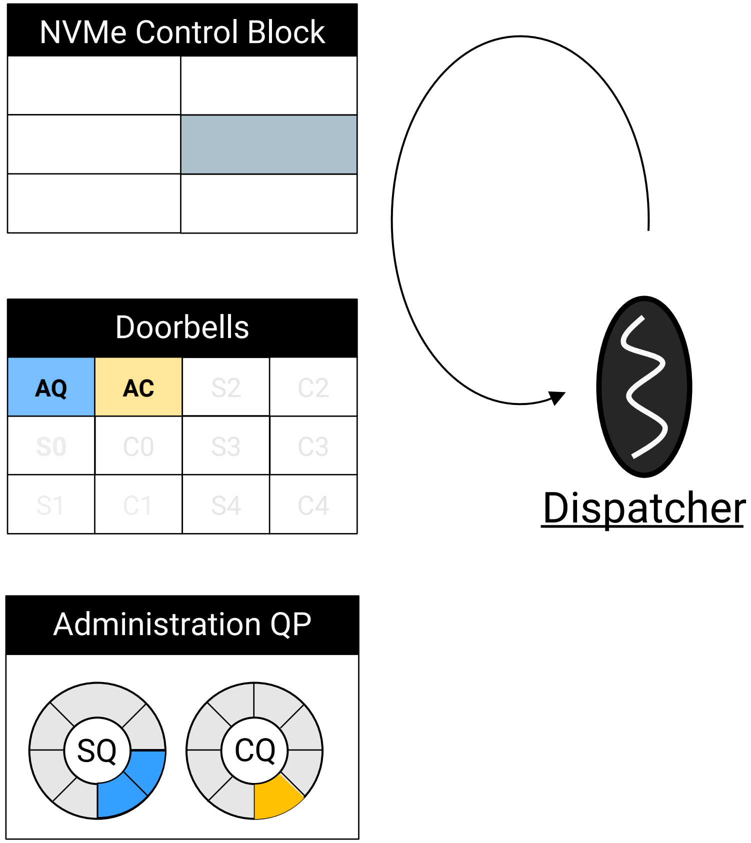
- Challenge 2: Cannot rely on the PCI mechanism to detect the requests from the host-side
  - Updates to the control block and doorbells are notified to the device as PCI transactions

  → Changes are applied silently as normal memory writes

- Solution: Dedicate a thread that scans the control block and doorbells to find any updates

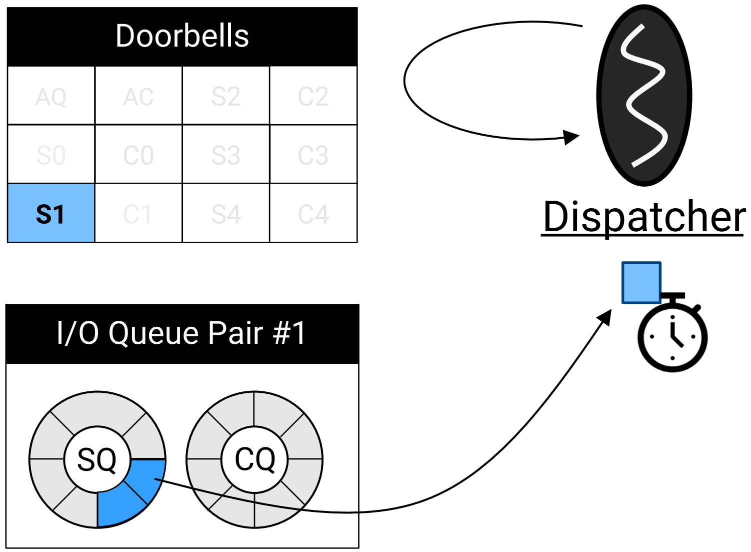
1973
AJOU UNIV.

Emulating NVMe Device: Configuration Requests

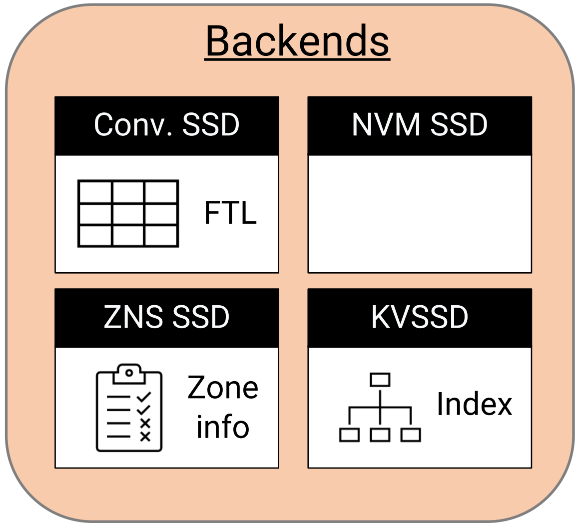


- Dispatcher directly processes configuration requests
  - Enable/shutdown device
  - Identify device and namespaces
  - Setup administration queue pair
  - Set/get features (e.g., # of queues)
  - Allocate/deallocate I/O queues
- Handle completion doorbells
  - Perform housekeeping

Emulating NVMe Device: I/O Requests

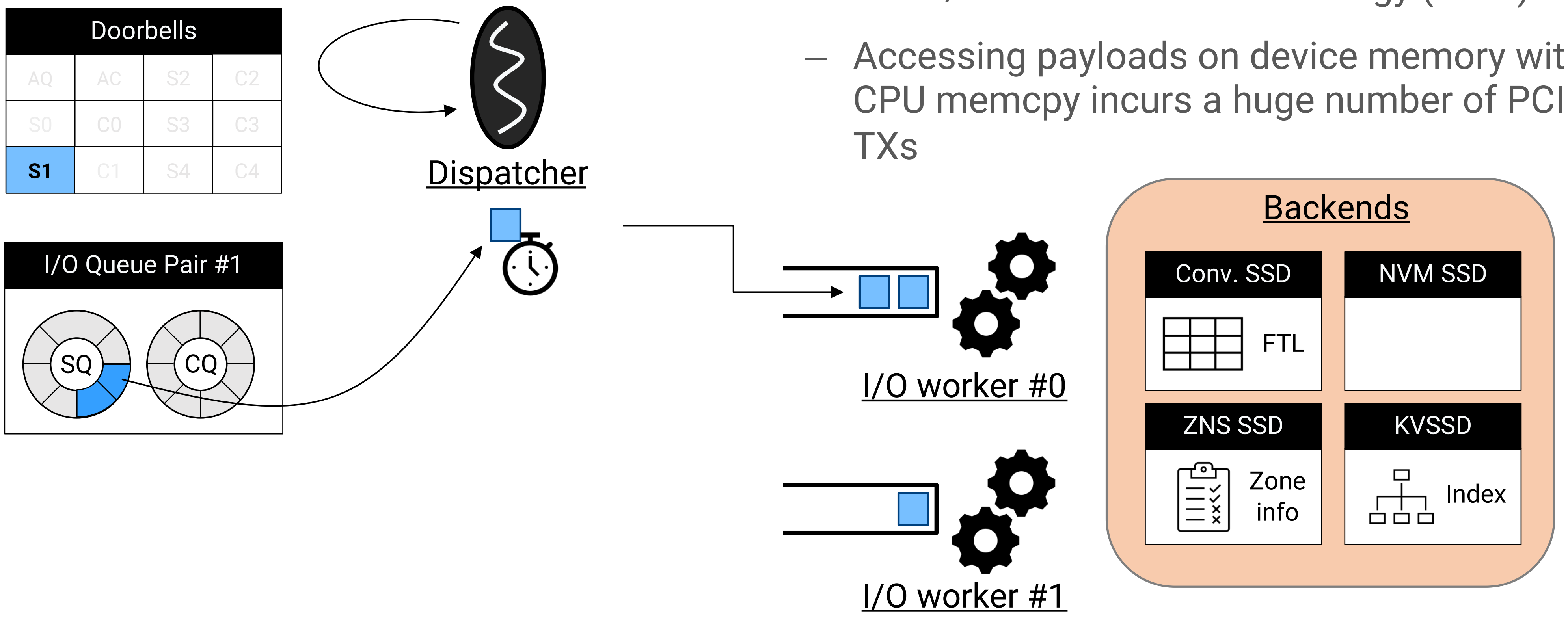


- I/O requests are divided into backend operations
  - According to the configured backend type
- Attach timestamps on the backend operations
  - Requested time, expected completion time



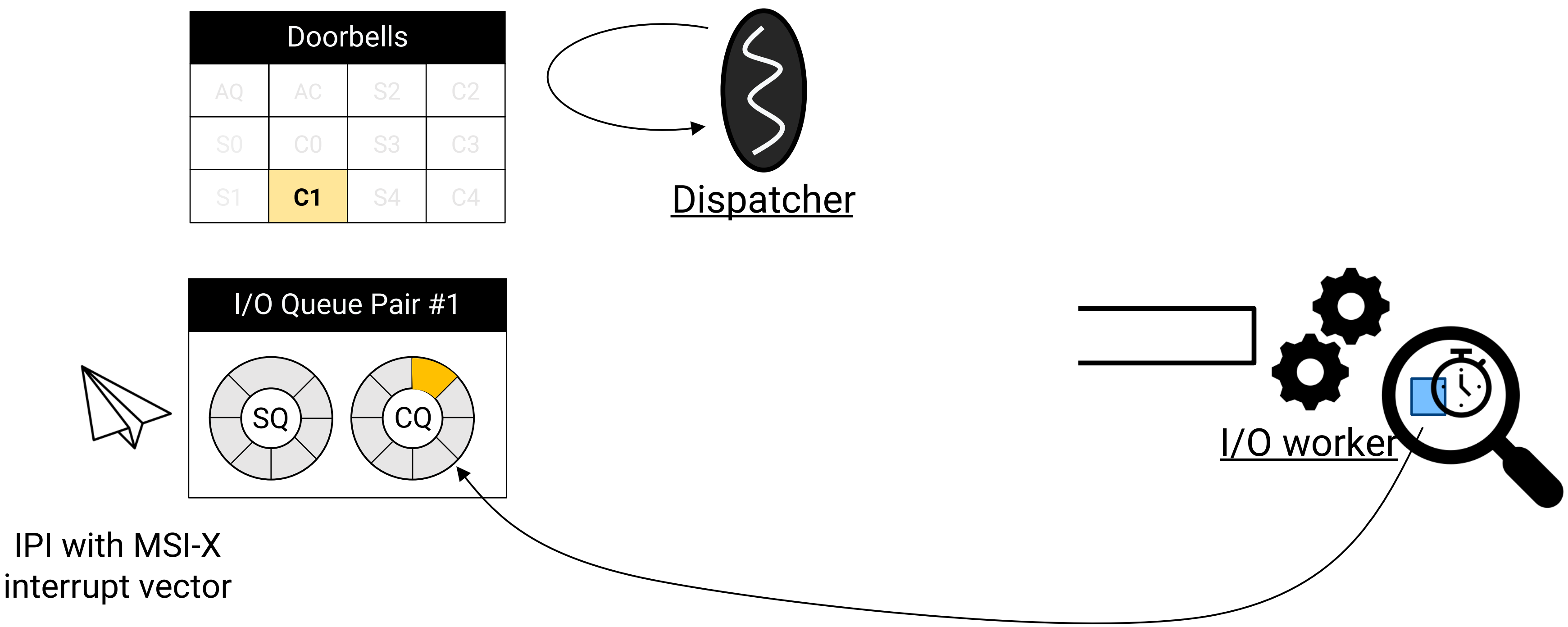
Emulating NVMe Device: I/O Requests

- Backend operations are dispatched to I/O workers
- I/O worker moves data using DMA engine
  - Intel I/O Acceleration Technology (IOAT)
  - Accessing payloads on device memory with CPU memcpy incurs a huge number of PCI TXs



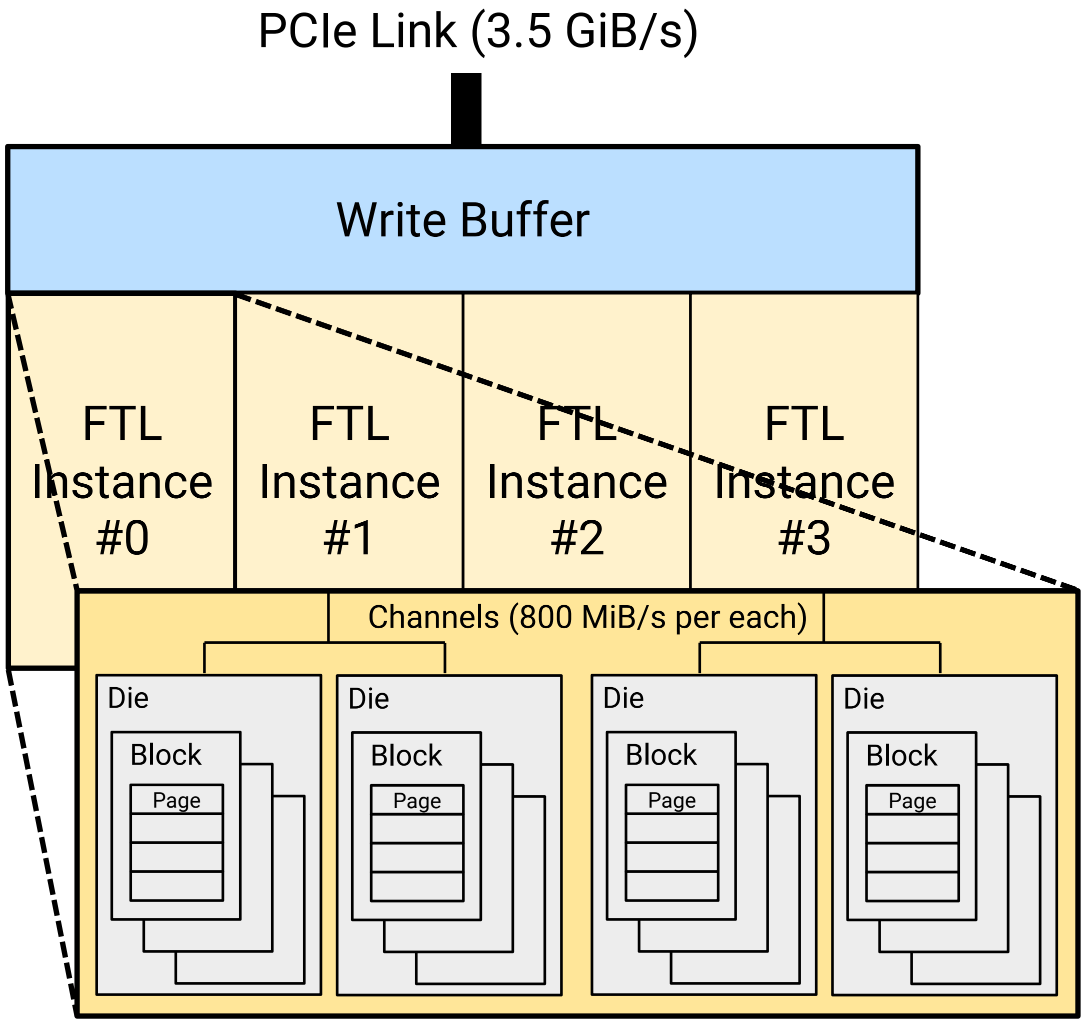
Emulating NVMe Device: I/O Requests

- Notify of the I/O completion through IPI with MSI-X interrupt vector



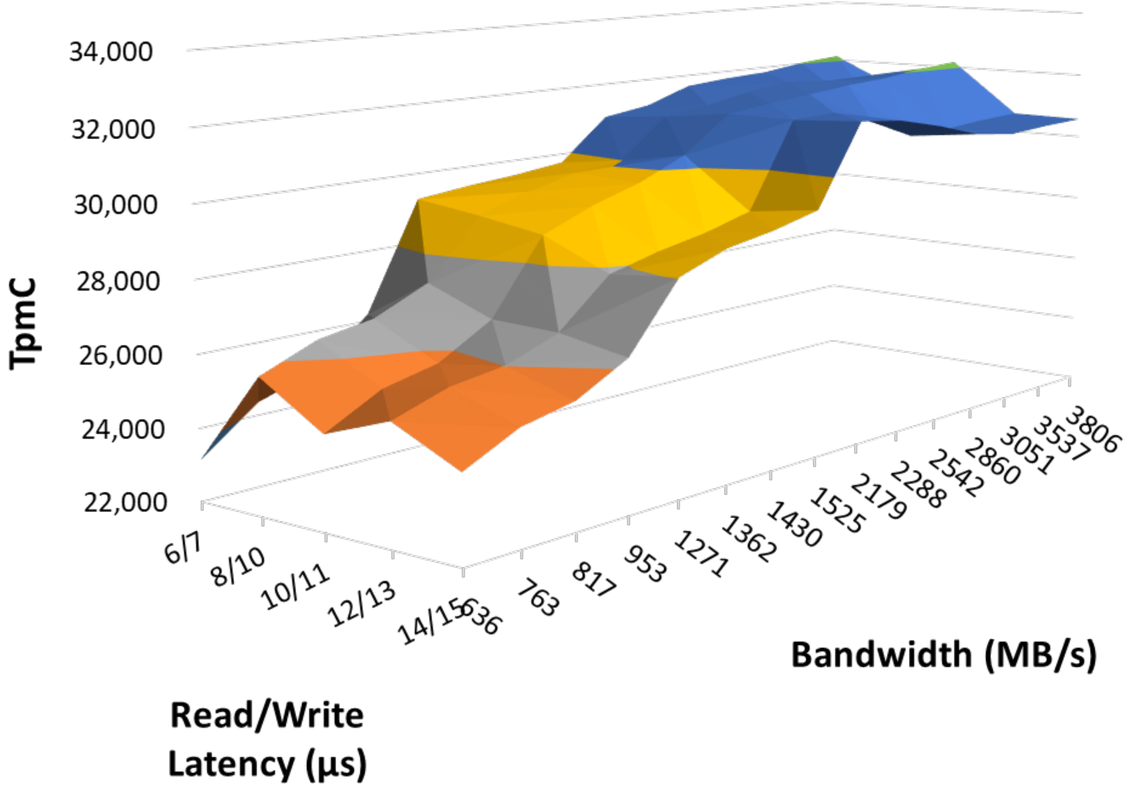
Performance Models

- Simple model for NVM SSDs
- Parallel model for conventional SSDs
  - A full-scale page-mapped FTL with GC
  - Model the on-device write buffer
  - Adopt the one-shot programming scheme
  - Model the parallel architectures in modern SSDs
    - Multiple FTL instances
    - Multiple dies and channels that operate independently
    - PCIe link and channels with limited aggregate bandwidth



Simple Performance Model

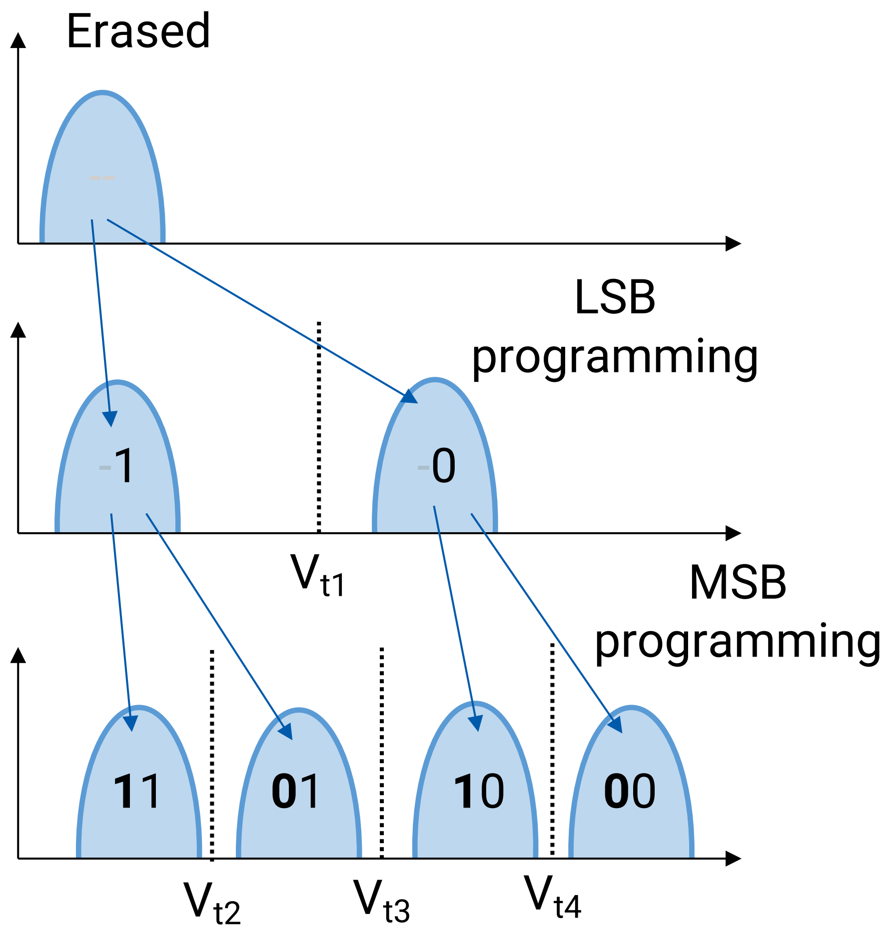
- Models a set of parallel I/O units where each I/O unit handles a sequence of I/O operations
  - Timing parameters are computed from target_latency and target_bandwidth
    - Can be independently specified
    - E.g. Optane SSD:
      Read 12μs @ 2.4GiB/s,
      Write 14μs @ 2.0GiB/s
- Used for Optane-like NVM SSDs and KV-SSDs

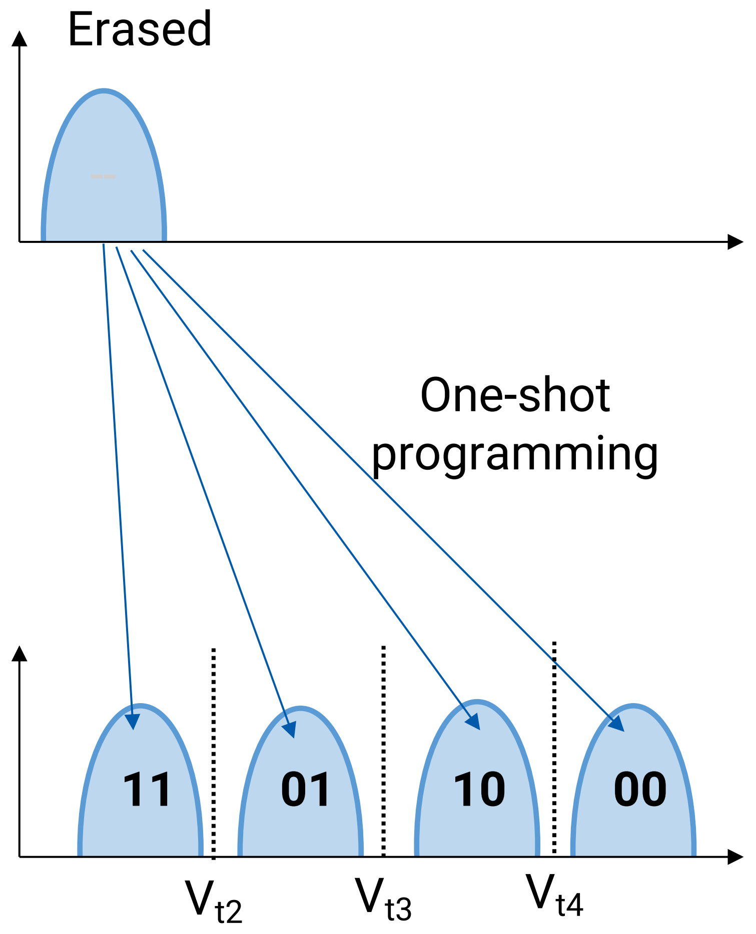


1973
AJOU UNIV.

Advanced Performance Model

- Adopt one-shot programming model with on-device write buffer



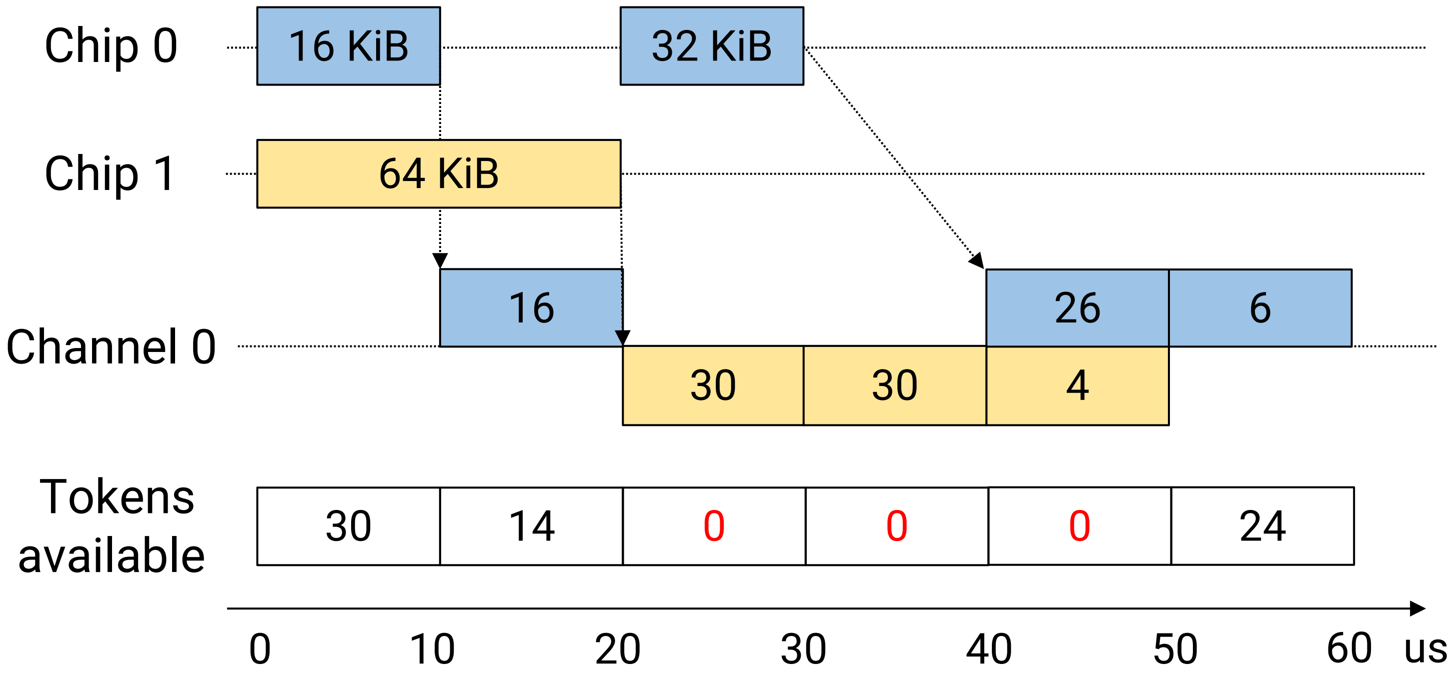


[Achieving Defect-Free Multilevel 3D Flash Memories with One-Shot Program Design, DAC'18]

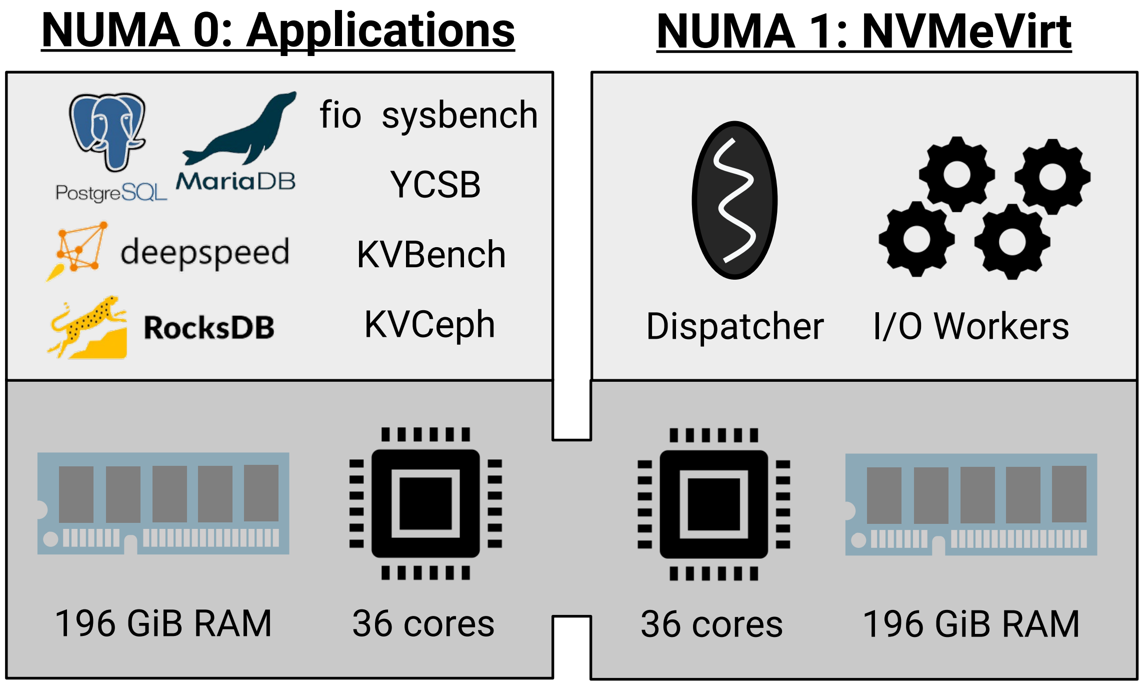
Advanced Performance Model

- Token-based contention model

1 token = 1 KiB
30 tokens per 10 us ~= 3 GiB/s



Evaluation



- Implemented in the Linux kernel 5.15+ (~10,000 LoC)
- Intel Xeon Gold 6240 x2
- 392 GiB RAM
- Debian Bullseye 11.5
- MariaDB 10.5
- PostgreSQL 13

Samsung 970 Pro

- Conventional SSD
- 512 GB

Intel P4800X

- OptaneDC NVM SSD
- 350 GB

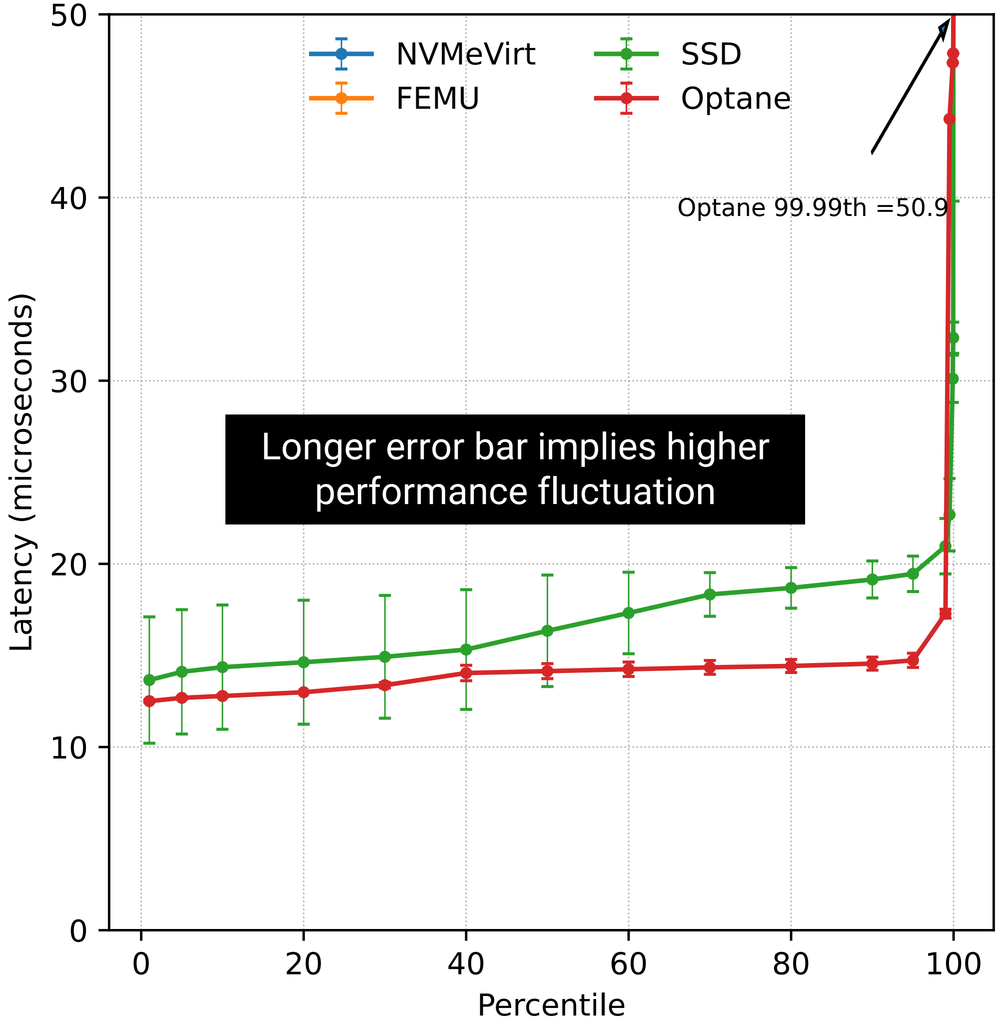
Samsung KVSSD

- 3.84 TB

Prototype ZNS SSD

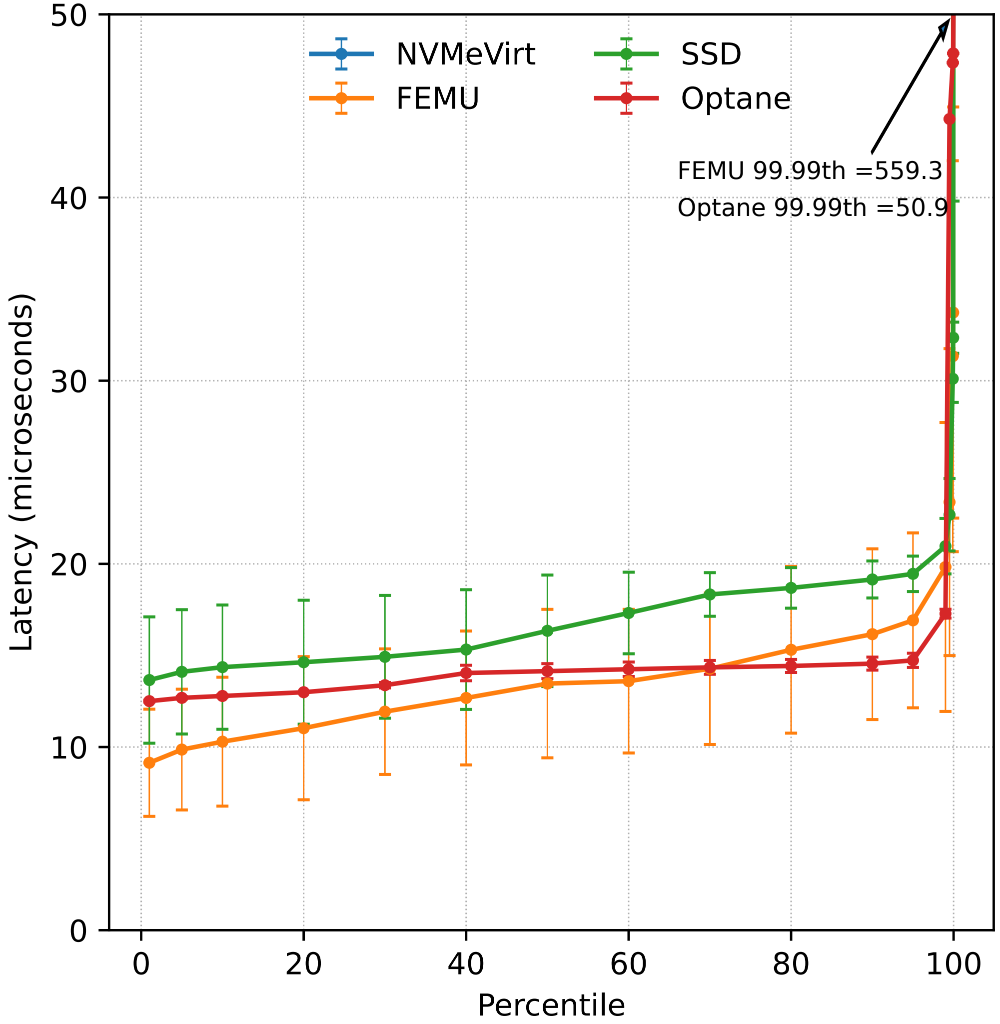
- 96 MiB zones
- 192 KiB write unit
- 32 TB

Emulation Quality: Performance Variance



- Distribution of percentiles for 10 runs
  - Each run does 4 KiB random writes with fio
  - Error bar indicates the standard deviation for the percentile

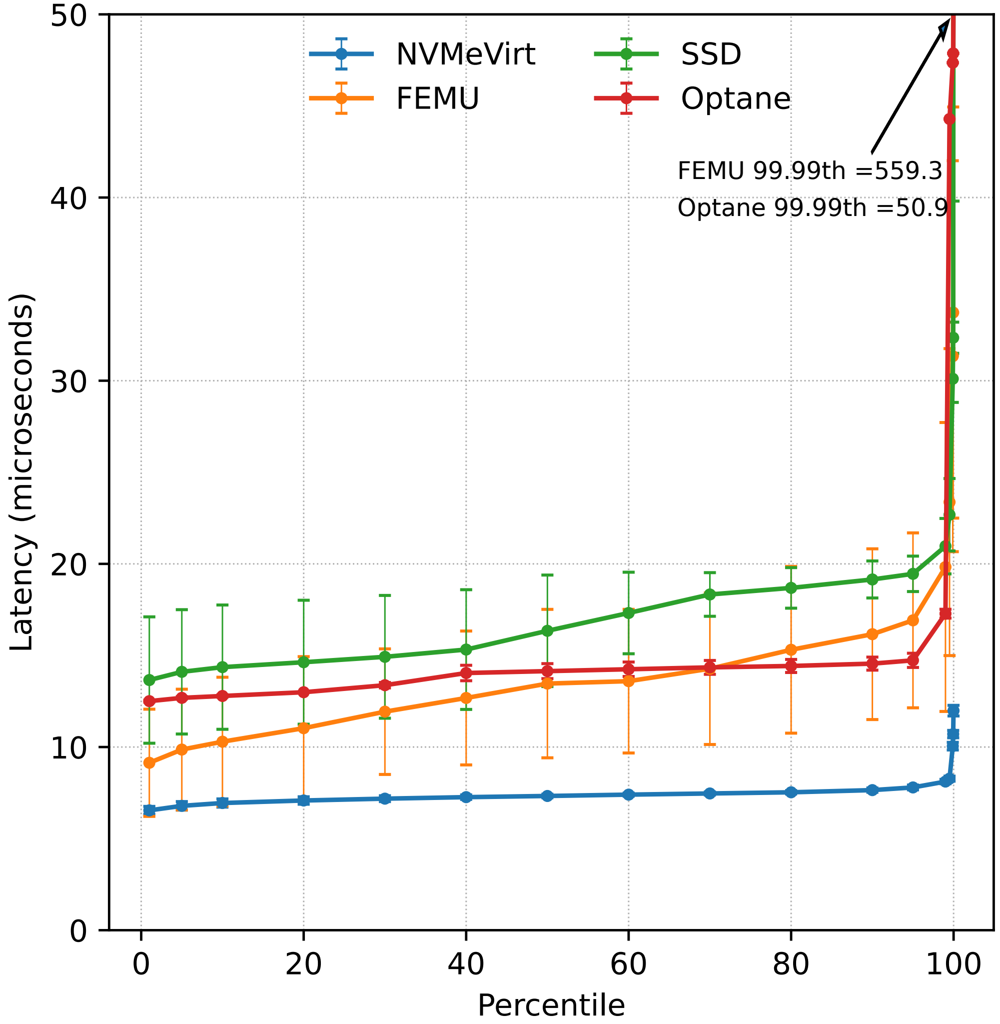
Emulation Quality: Performance Variance



- Distribution of percentiles for 10 runs
  - Each run does 4 KiB random writes with fio
  - Error bar indicates the standard deviation for the percentile
- FEMU exhibits a long tail latency and high run-by-run performance fluctuation
- FEMU would not be able to consistently emulate high-performance NVM SSDs

1973
AJOU UNIV.

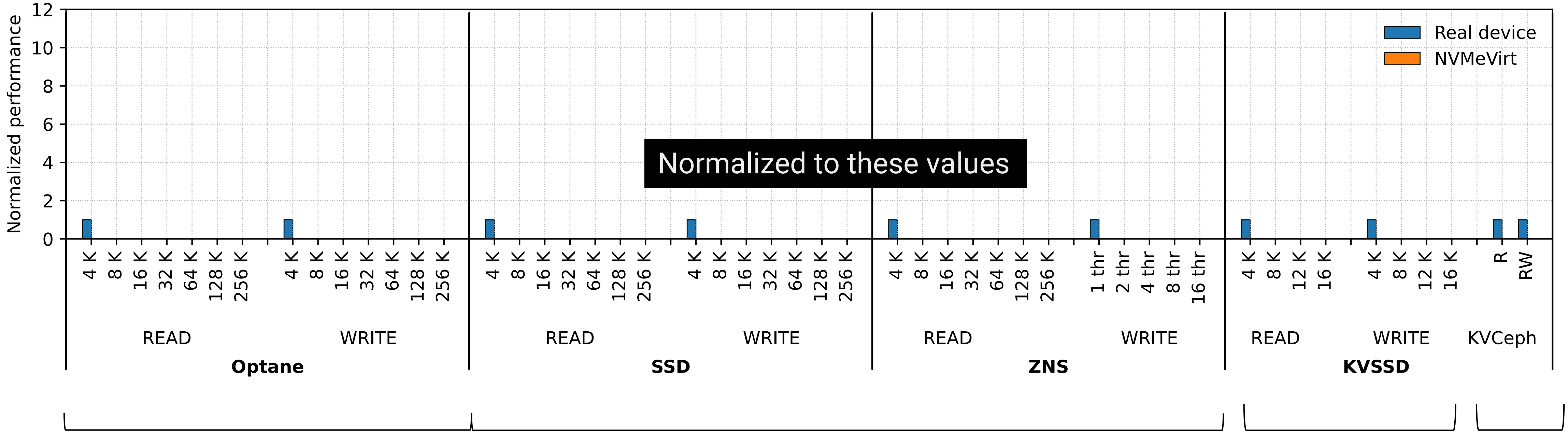
Emulation Quality: Performance Variance



- Distribution of percentiles for 10 runs
  - Each run does 4 KiB random writes with fio
  - Error bar indicates the standard deviation for the percentile
- FEMU exhibits a long tail latency and high run-by-run performance fluctuation
- FEMU would not be able to consistently emulate high-performance NVM SSDs
- NVMeVirt provides low latency with little performance variation

1973
AJOU UNIV.

Performance Comparison to Real Devices



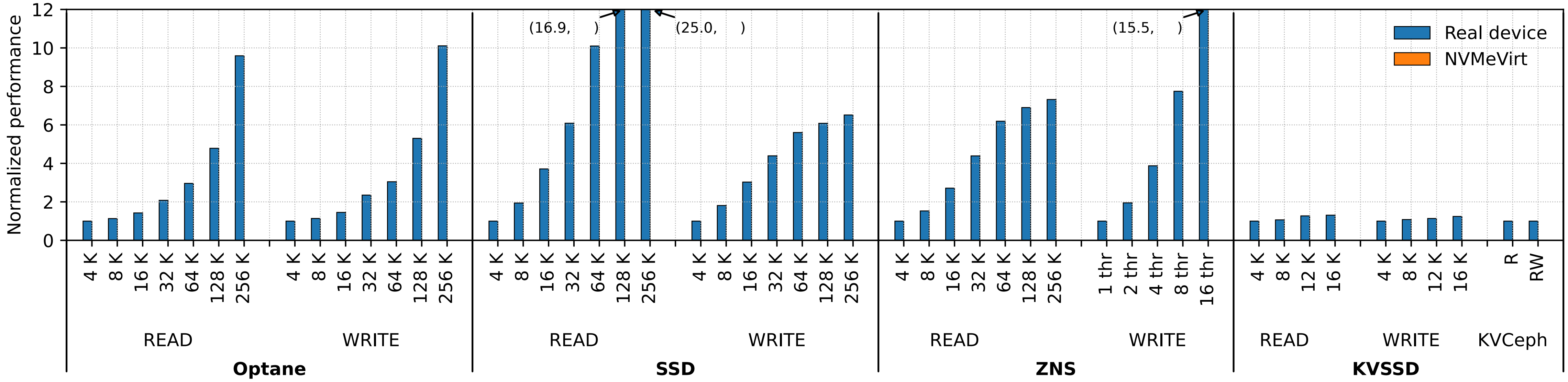
fio random access latency

fio random access bandwidth

OpenMPDK KVBench agg. BW

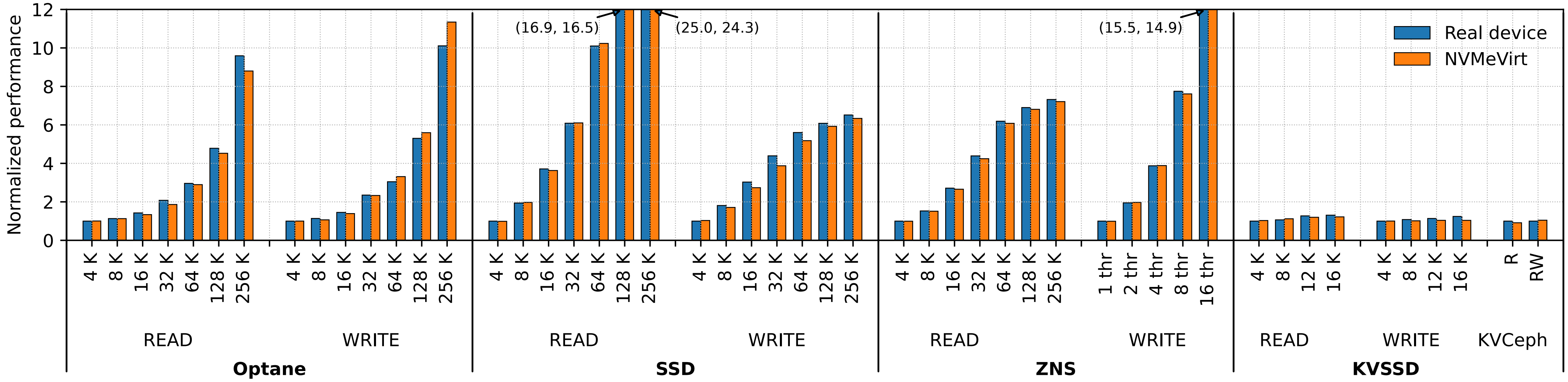
KVCeph agg. BW

Performance Comparison to Real Devices



1973
AJOU UNIV.

Performance Comparison to Real Devices

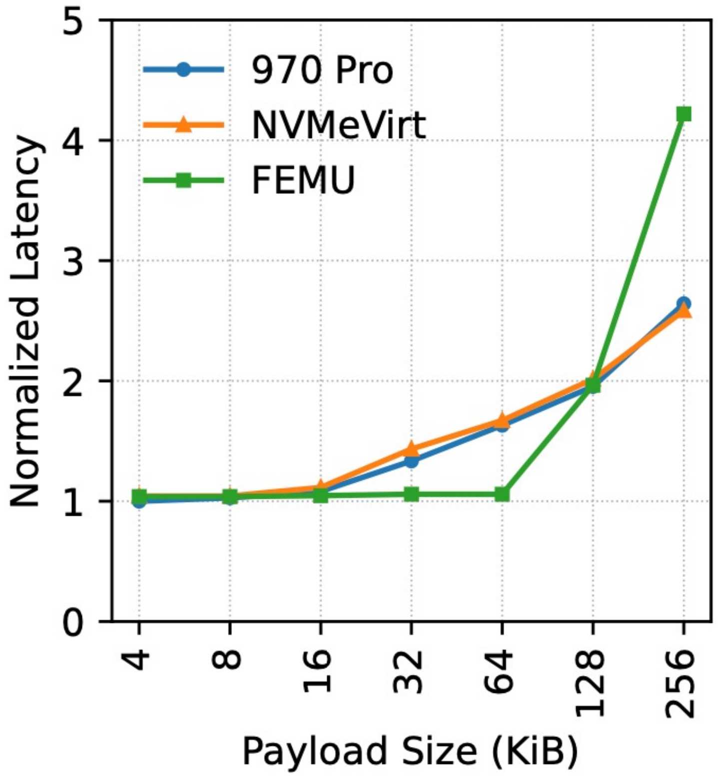


NVMeVirt can replicate the real devices' performance closely

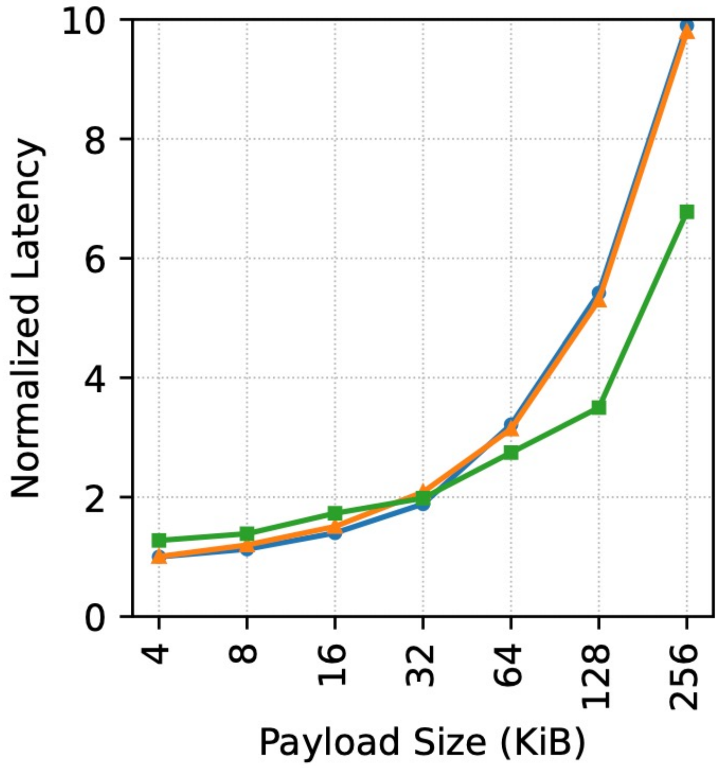
Harmonic mean of performance differences = 1.17%

Effects of Contention Modeling

- Fio with increasing payload sizes



Read



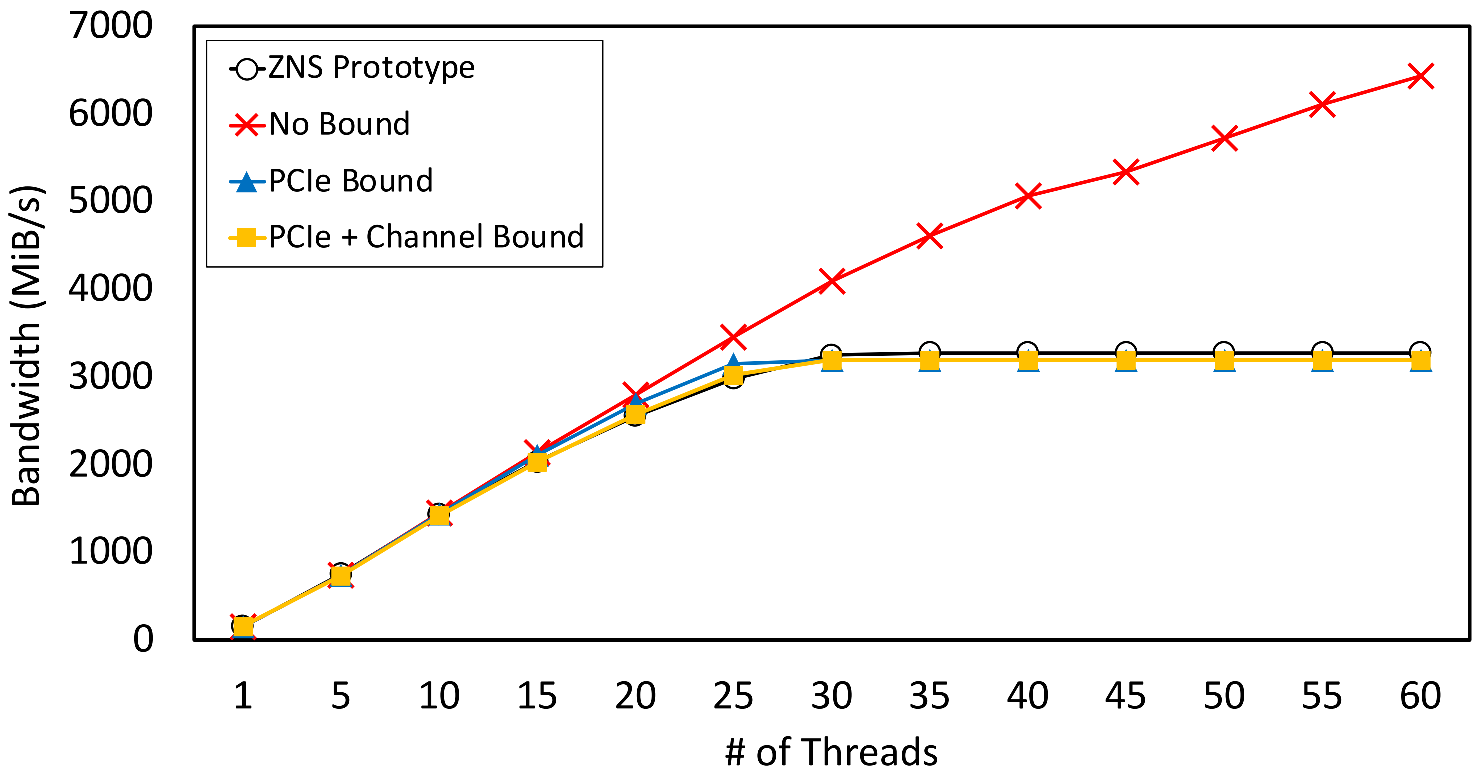
Write

- 32 KiB pages
- 8 channels
- 2 chips channel

- 3360 MiB/s PCIe
- 800 MiB/s channel

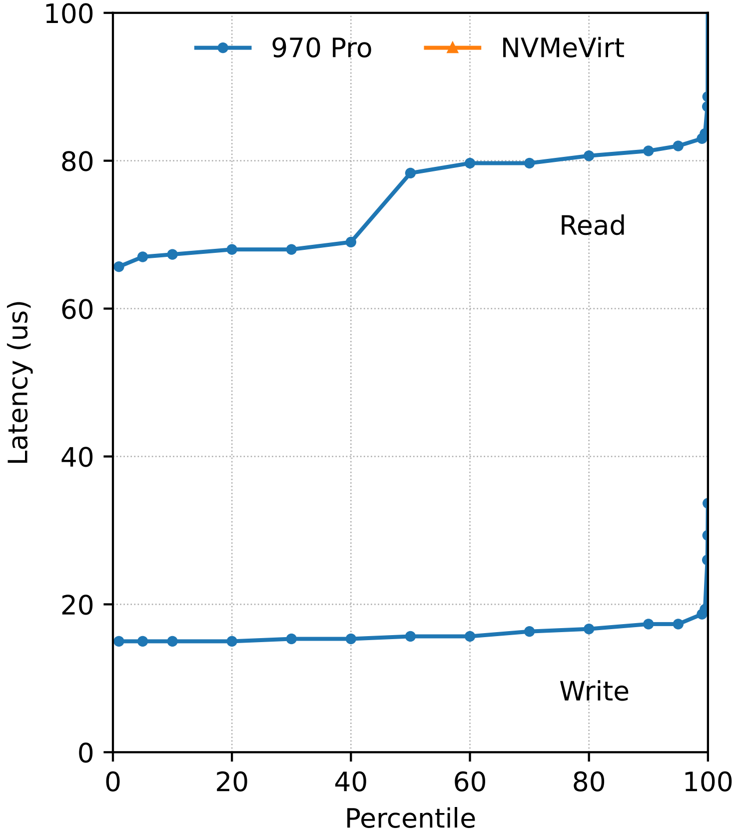
Effects of Contention Modeling

- Fio with increasing number of threads



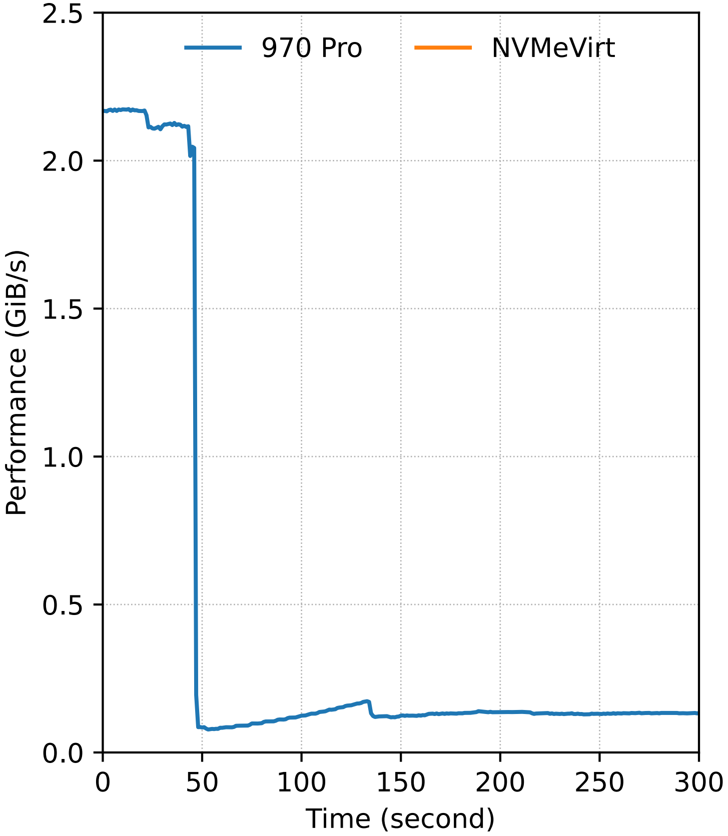
16KiB random read bandwidth

Performance Characteristics Compared to Real Devices



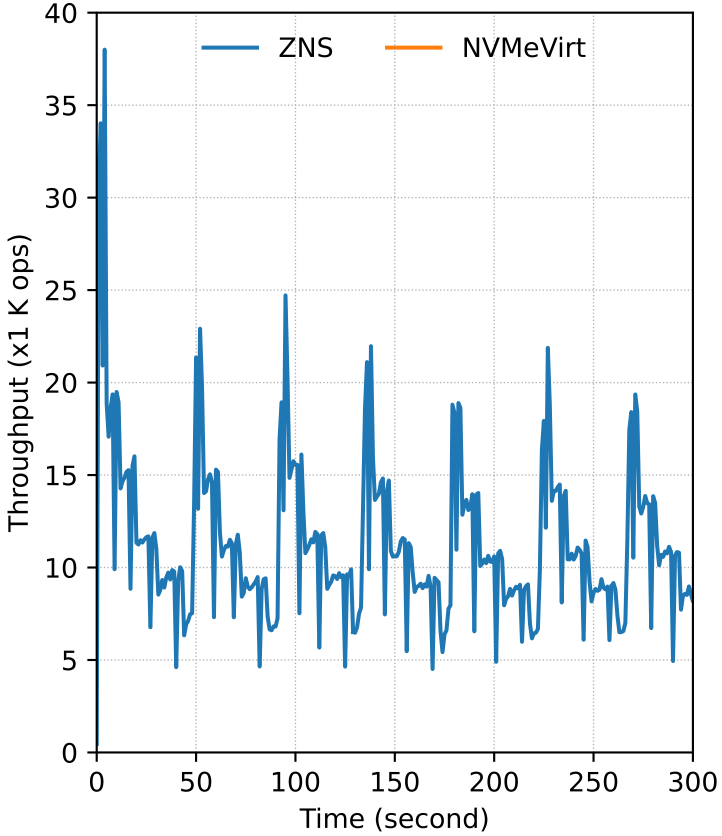
Distributions of latencies

- fio 16 KiB



Performance impact of GC

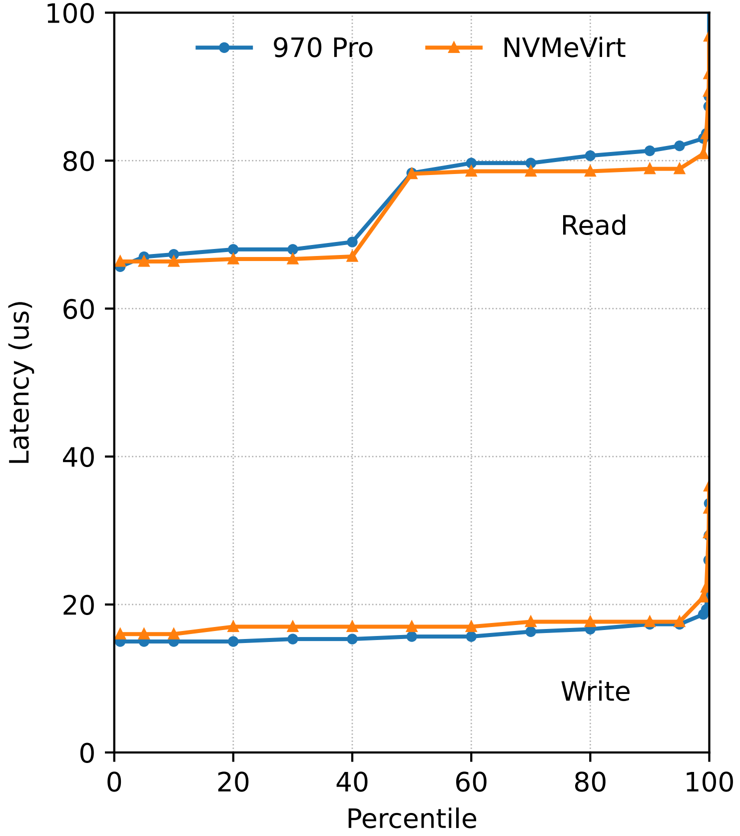
- Fill storage with sequential writes
- Perform random writes to trigger GC

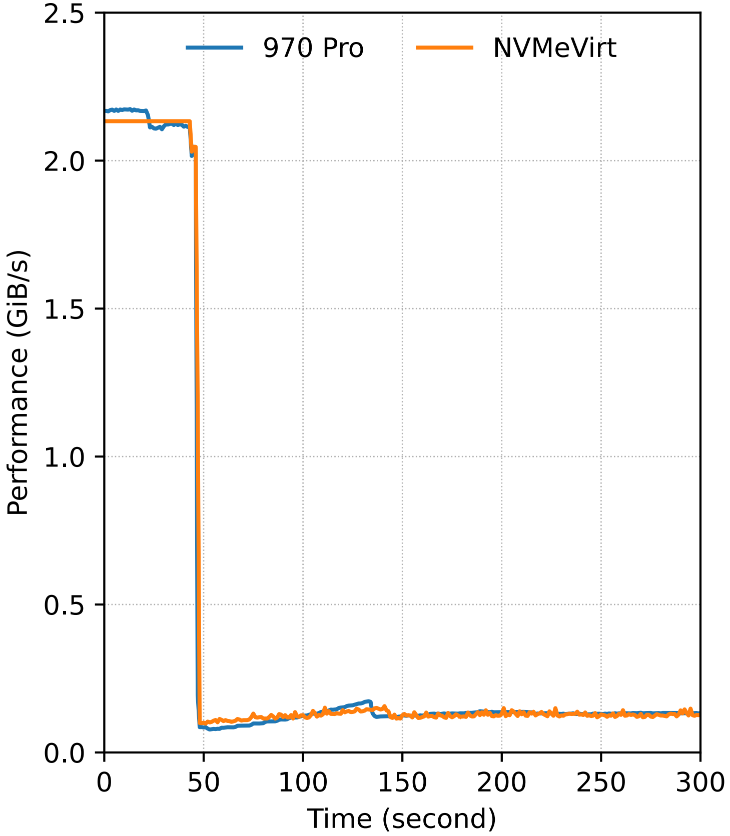


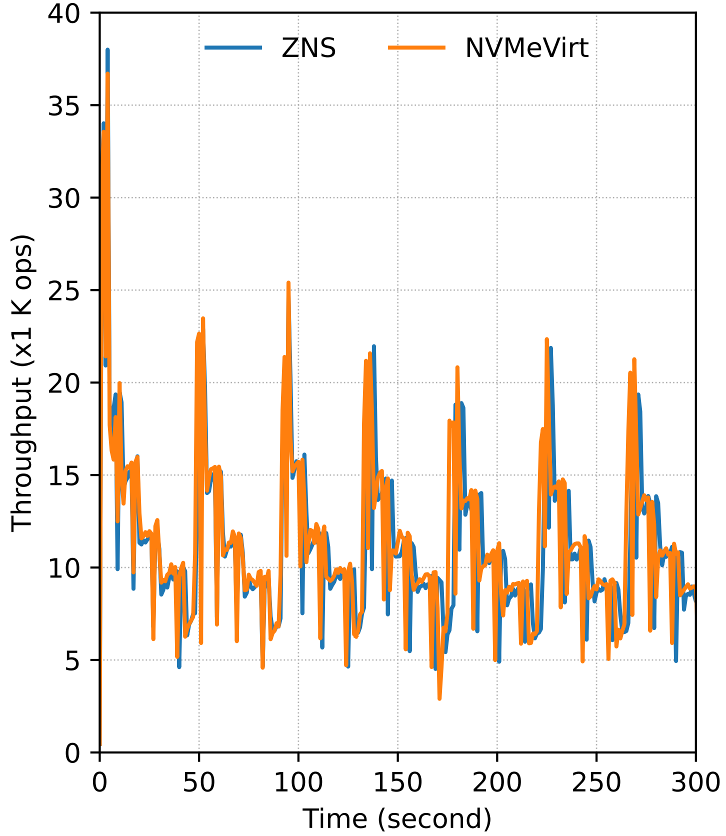
Throughput over time

- YCSB-A on RocksDB (50:50 read:update)

Performance Characteristics Compared to Real Devices







Distributions of latencies

- fio 16 KiB

Performance impact of GC

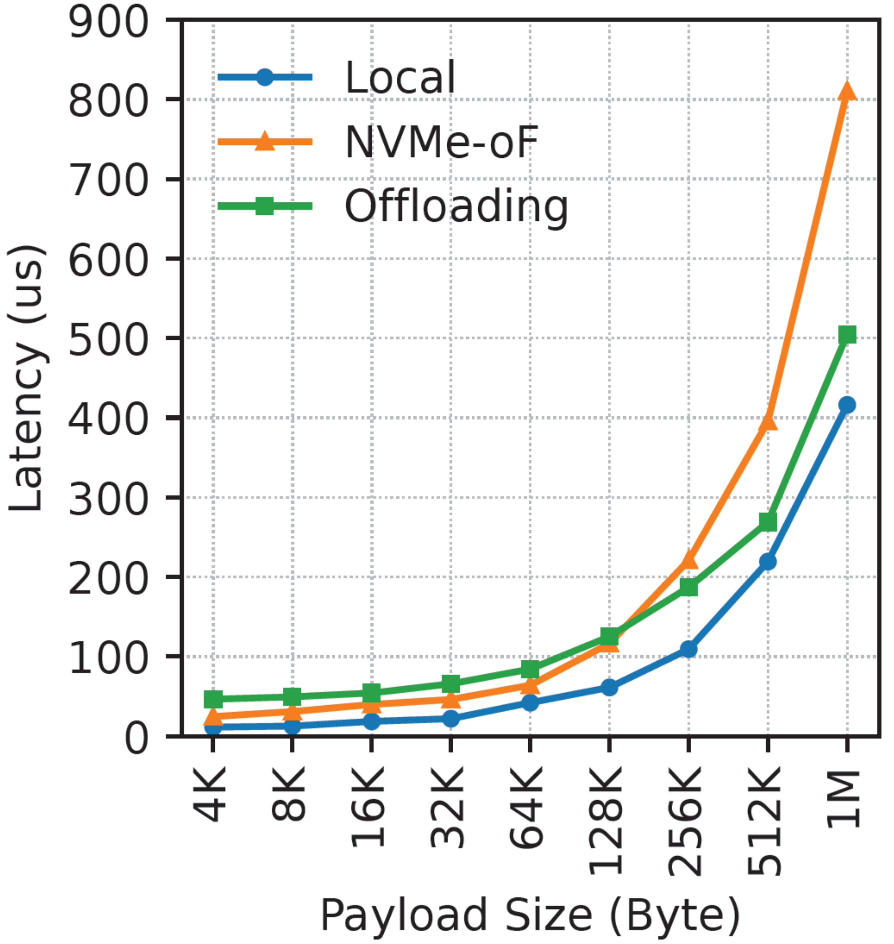
- Fill storage with sequential writes
- Perform random writes to trigger GC

Throughput over time

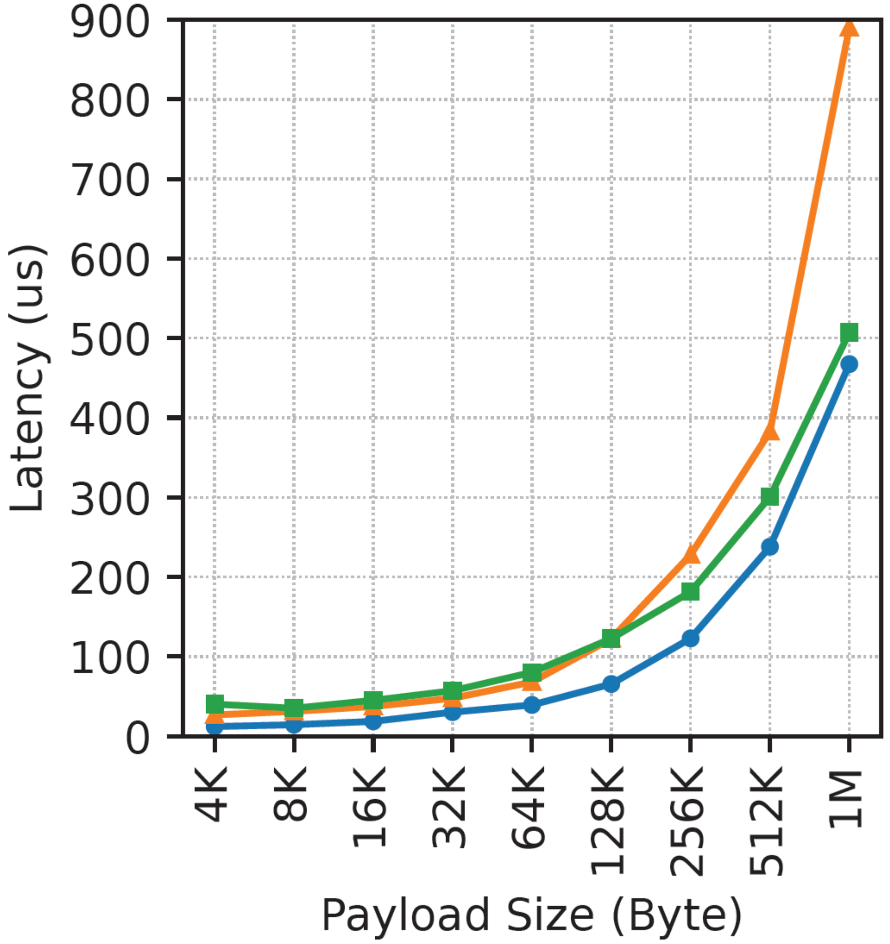
- YCSB-A on RocksDB (50:50 read:update)

1973
AJOU UNIV.

NVMe-oF Write Latency



Optane

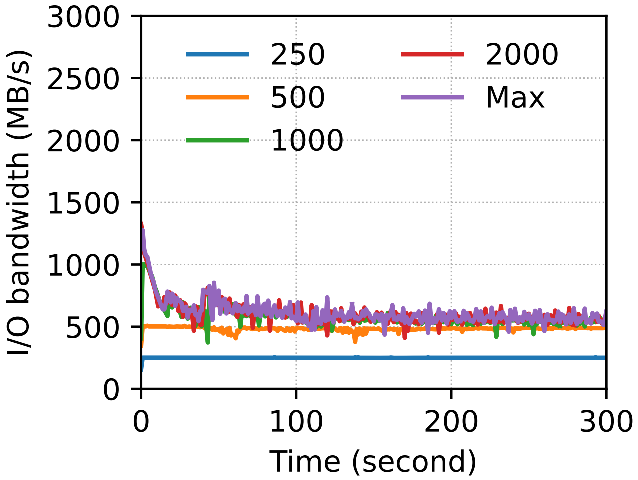


NVMeVirt

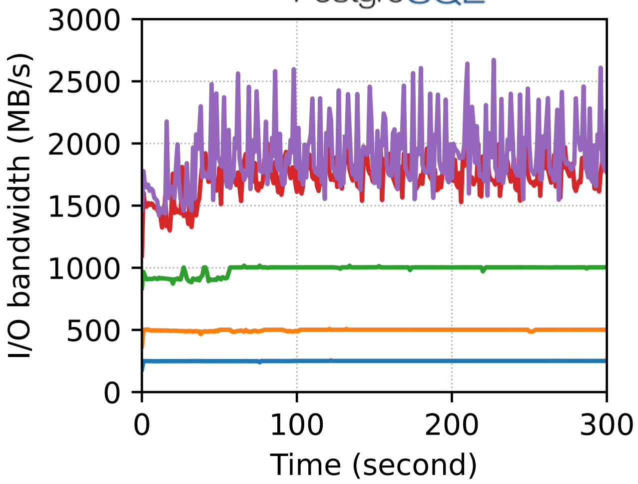
Case Study 1: DBMS on Various Storage Configurations

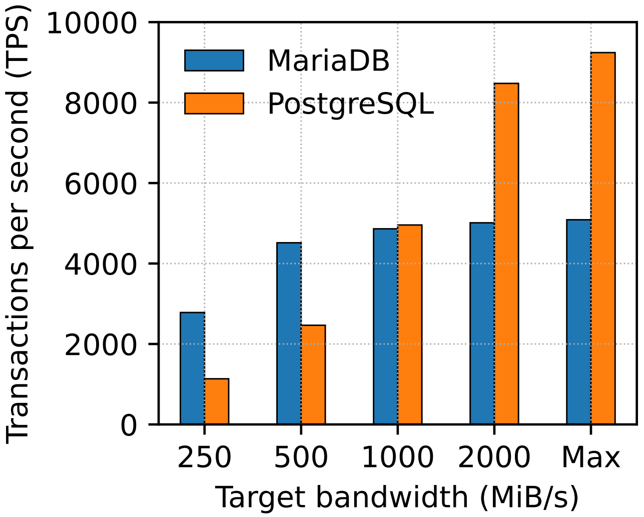
- Sysbench with various bandwidth limits

MariaDB



PostgreSQL

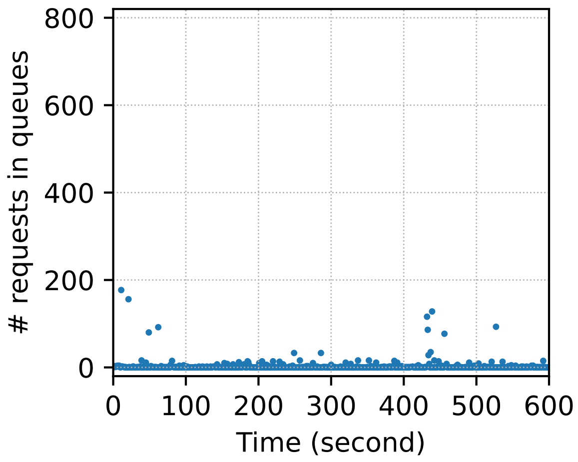




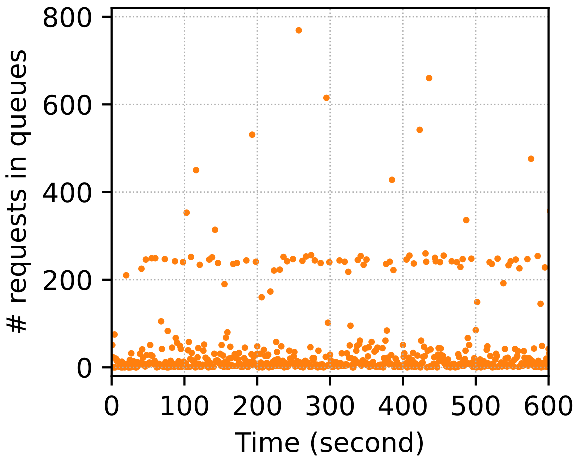
Case Study 1: DBMS on Various Storage Configurations

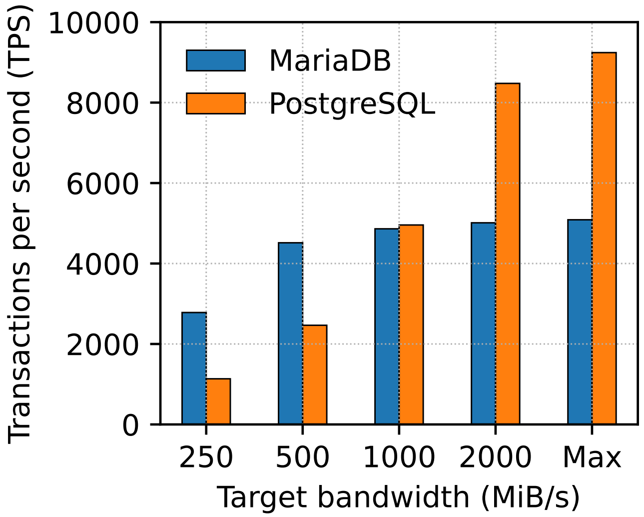
- Sysbench with various bandwidth limits

MariaDB



PostgreSQL

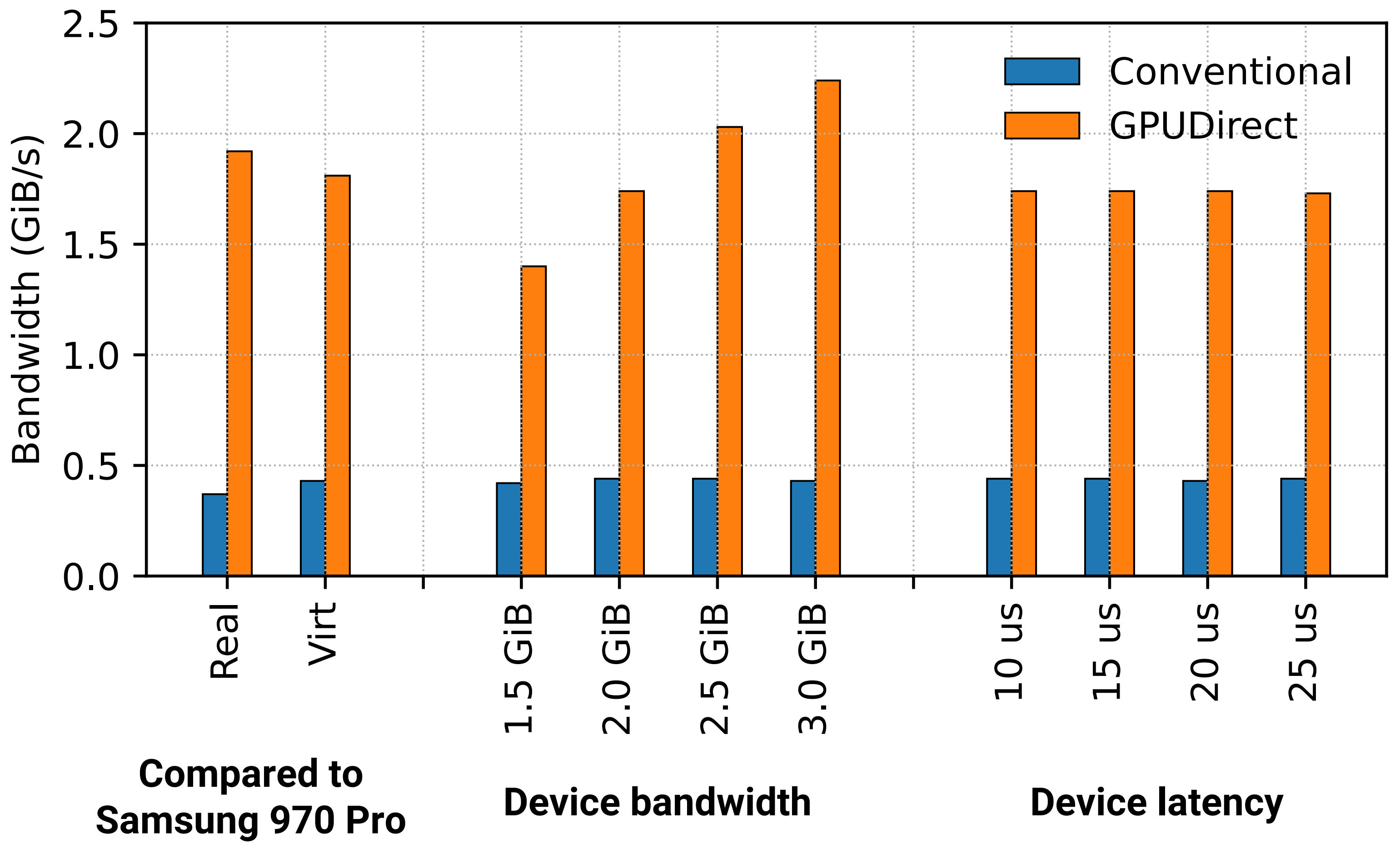




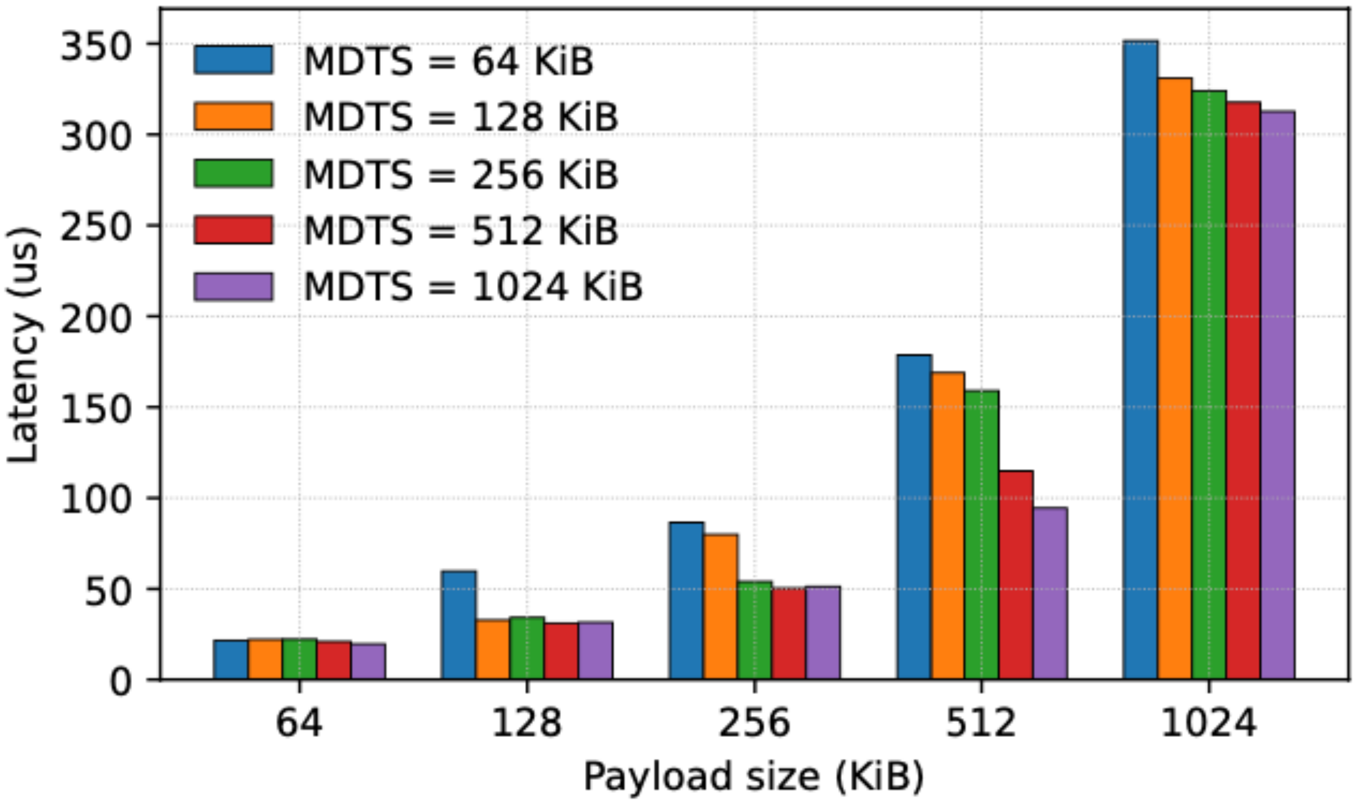
1973
AJOU UNIV.

Case Study 2: AI Application and PCIe P2P DMA

- Performance of checkpointing of Megatron DeepSpeed



Effect of MDTS value

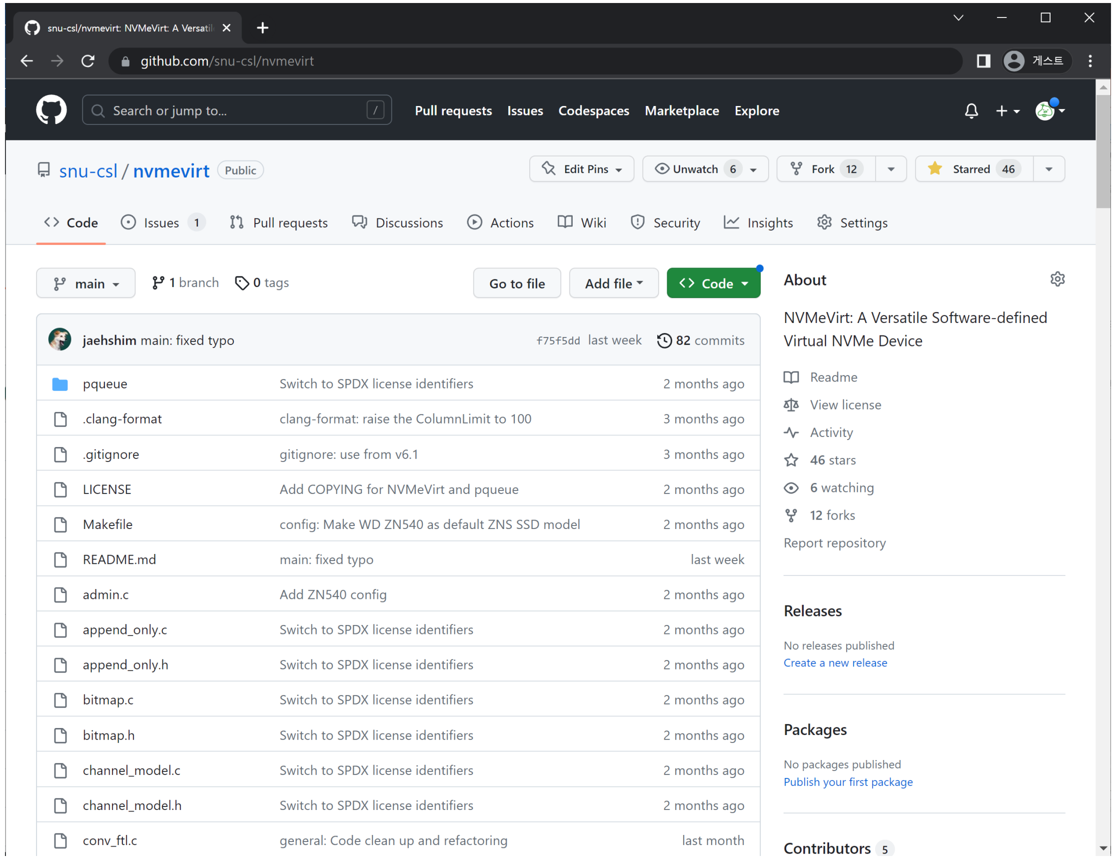


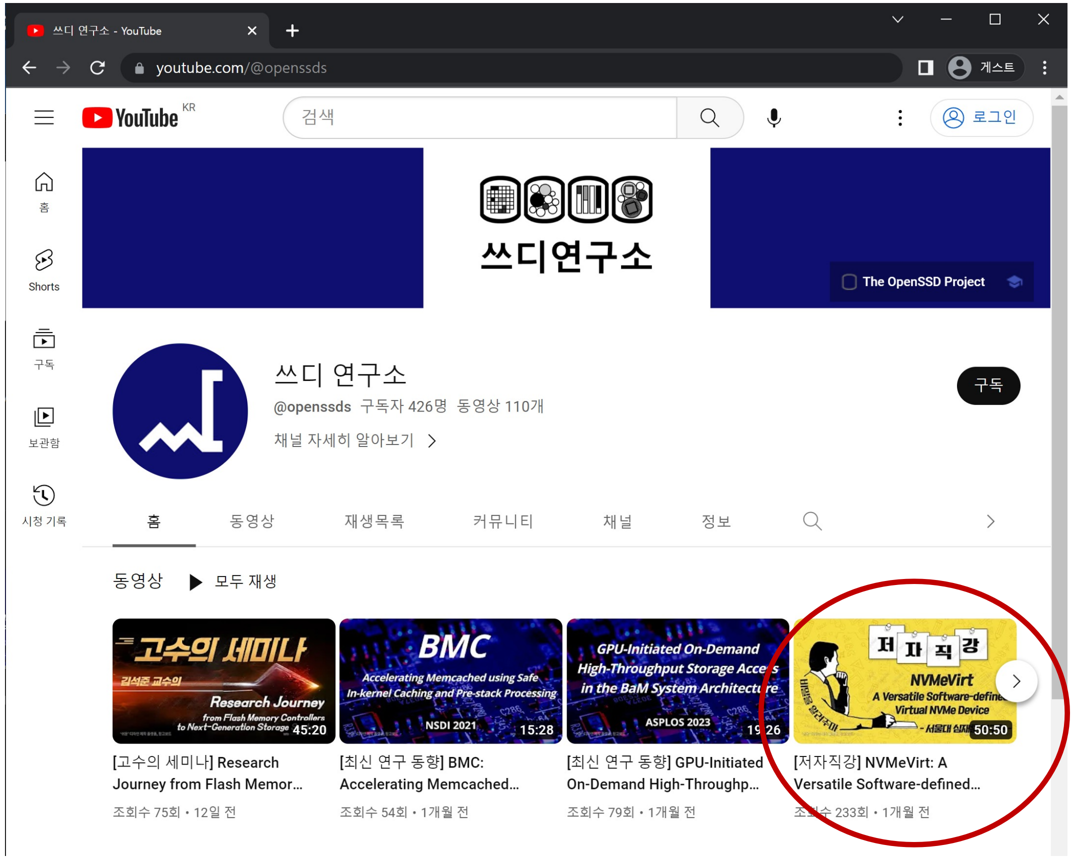
More Use Cases

- Fast prototyping for new NVMe interface extensions
  - E.g., FDP, Computational storage
- Finding and improving a software bottleneck in the storage stack
- Developing a new device-centric architecture
- Analyze the application's scalability on future high-performance storage devices
- Investigating performance impact of hardware parameters (e.g. MDTS, # queues)
- Benchmarking and performance/reliability testing

Give it a Try! (We are on Youtube too!)

- https://github.com/snu-csl/nvmevirt





1973
AJOU UNIV.

Conclusion

- NVMeVirt presents a virtual NVMe device
- Support all the modern storage configurations and device types
  - Configurations: Kernel bypass, PCI P2P DMA, and RDMA
  - Types: Conventional SSD, NVM SSD, ZNS SSD, and KVSSD
- Code is available at Github: https://github.com/snu-csl/nvmevirt

1973
AJOU UNIV.