High Performance Computer System Lab

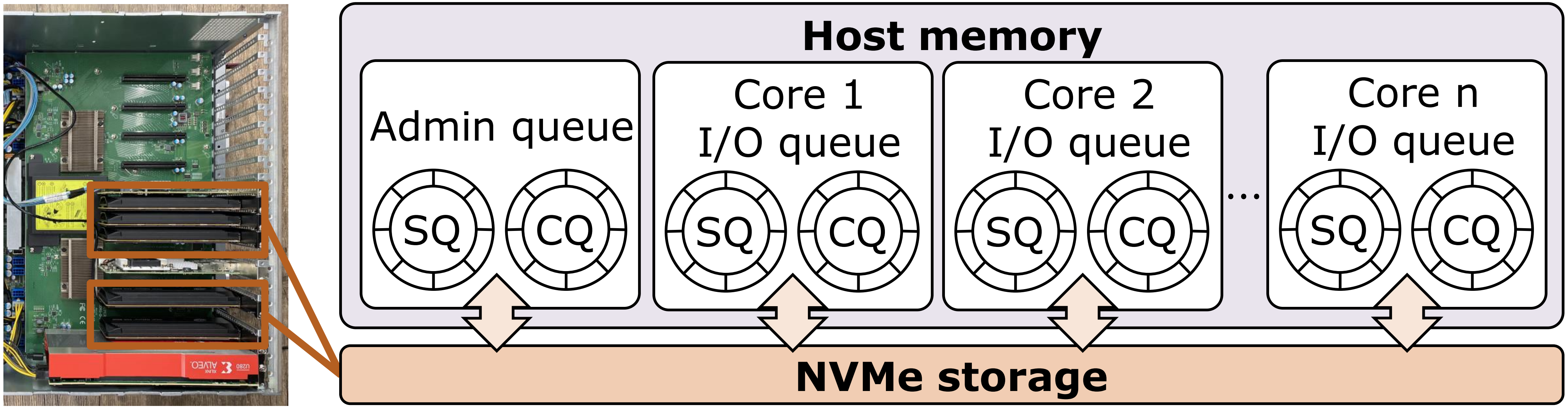
FVM: FPGA-assisted Virtual Device Emulation for Fast, Scalable, and Flexible Storage Virtualization

Dongup Kwon[1,2], Junehyuk Boo[1], Dongryeong Kim[1], and Jangwoo Kim[1,2]

[1] Department of Electrical and Computer Engineering, Seoul National University
[2] Memory Solutions Lab, Samsung Semiconductor Inc.

Background: NVM Express (NVMe) Storage

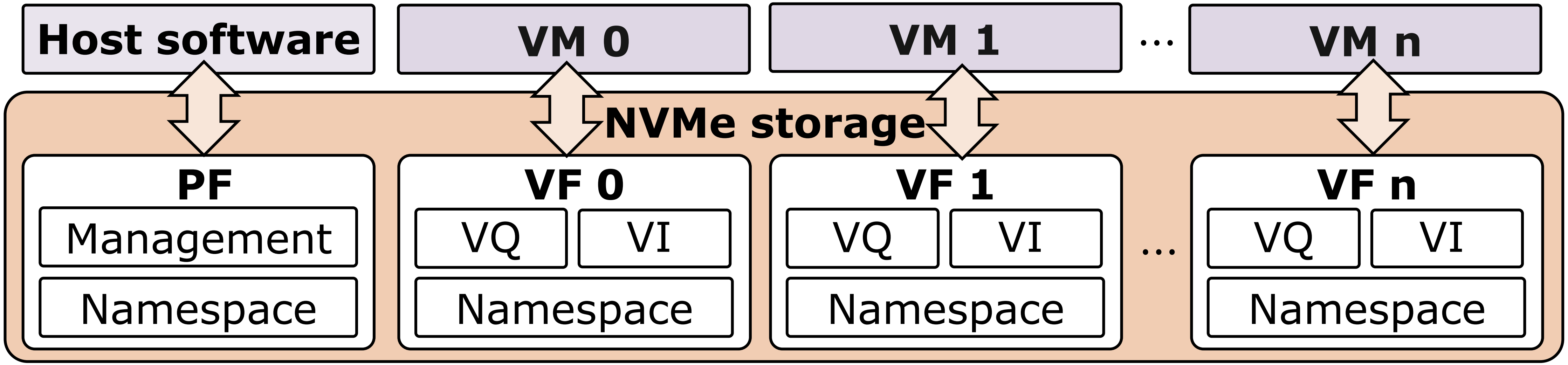
- **Provide high I/O performance through PCIe**
  - Utilize multiple I/O submission/completion queue (SQ/CQ) pairs
  - Enable highly parallel I/O processing on multiple CPU cores



NVMe storage is widely used in modern datacenters to accelerate I/O

Background: HW-assisted NVMe Virtualization

- **Utilize single-root I/O virtualization (SR-IOV)**
  - Create multiple physical/virtual functions (PFs/VFs) internally
  - Assign each VF to a VM exclusively and allow direct access to HW
  - Assignable resources: virtual queues (VQs), virtual interrupts (VIs)



SR-IOV can provide near-native storage performance to multiple VMs

Background: Limitations of SR-IOV

- **Limited VM-management features and use cases**
  - No interposition layer bewteen VMs and storage
  - Inflexible storage resource allocation
- **Limited compatibility**
  – Vendor-specific and hard-wired implementations

Category	*Feature*	*SR-IOV*
Storage configuration	*Consolidation*	✓
	Aggregation	✗
	Caching	✗
Resource management	*Replication*	△
	Throttling	△
Administration	*Migration*	✗
	Metering	△

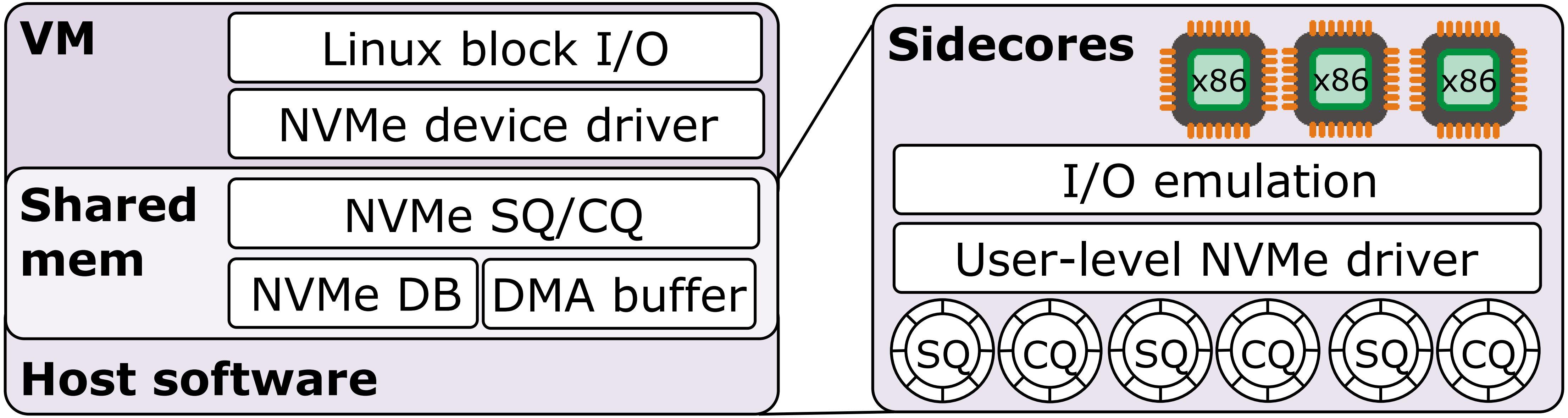
SR-IOV loses flexibility to implement critical VM-management features

Outline

- Background
- **Motivation**
  - SW-based host sidecore / on-device sidecore approaches
- FVM: FPGA-assisted Storage Virtualization
- Evaluation
- Conclusion

Alternative #1: SW-based Host Sidecore Approach

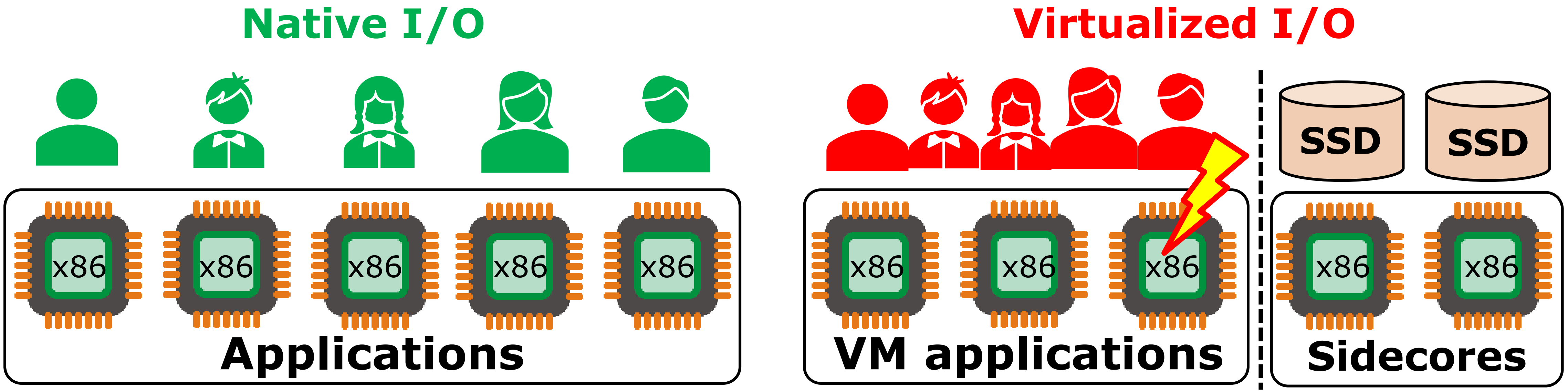
- **Dedicate CPU cores to emulate virtual devices**
- **Accelerate storage virtualization layers**
  - Avoid expensive traps to a hypervisor and cache pollution



Host sidecore approaches accelerate virtualization by dedicating CPU cores

Limitations of Host Sidecores

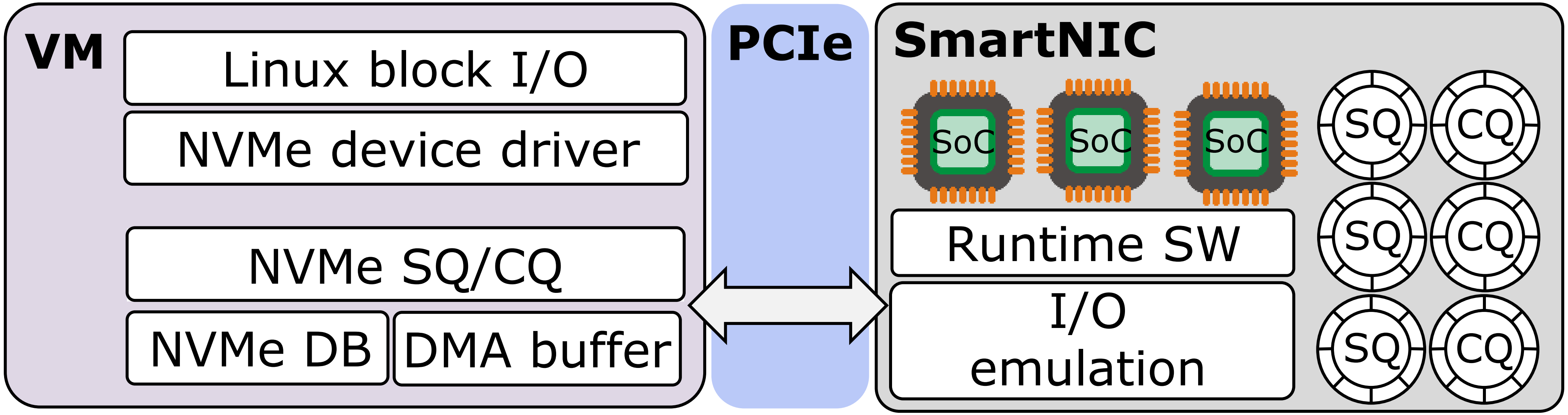
- **Expensive and non-scalable virtualization**
  - Polling guest I/O activities + indirect interrupt injection
  - Demand 40%-60% more CPU resources than native I/O operations
  - Limited VM performance or scalability due to lack of CPU resources



Host sidecore approaches should pay the expensive virtualization tax

Alternative #2:
On-device Sidecore Approach

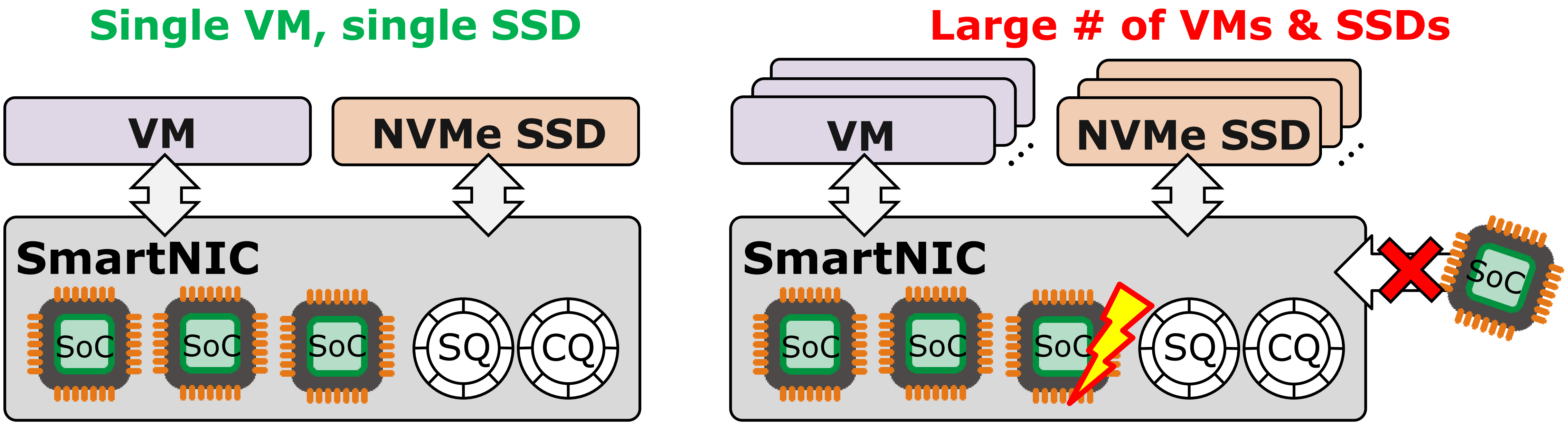
- **Offload a virtualization layer to SoC cores**
  - Emulate guest I/O via SoC cores in other peripheral devices
- **Save the host resources required for storage virtualization**



On-device sidecores can reduce the virtualization tax

Limitations of On-device Sidecore

- **Weak computing power of SoC cores**
  - Cannot support a large number of VMs, virtual/physical devices



On-device sidecores suffer from limited performance and scalability

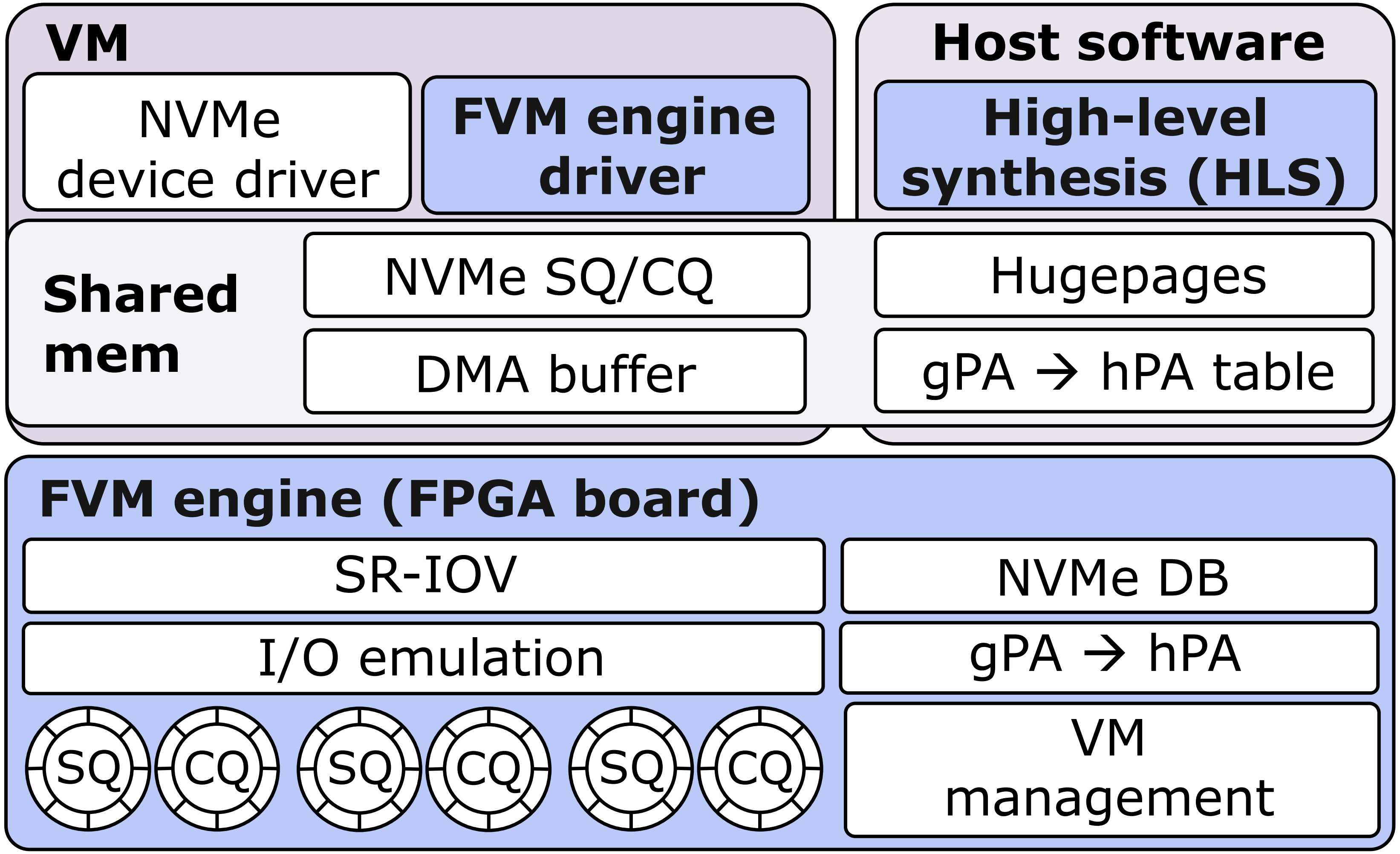
Design Goals

	SR-IOV	Sidecore CPU	Sidecore SoC	FVM
1 Performance	✓	✓	△	✓
2 Host efficiency	✓	✗	✓	✓
3 Scalability	✓	△	✗	✓
4 Flexibility	✗	✓	✓	✓
5 Compatibility	△	✓	✓	✓

Outline

- Background
- Motivation
- **FVM: FPGA-assisted Storage Virtualization**
- Evaluation
- Conclusion

Key Ideas and Benefits

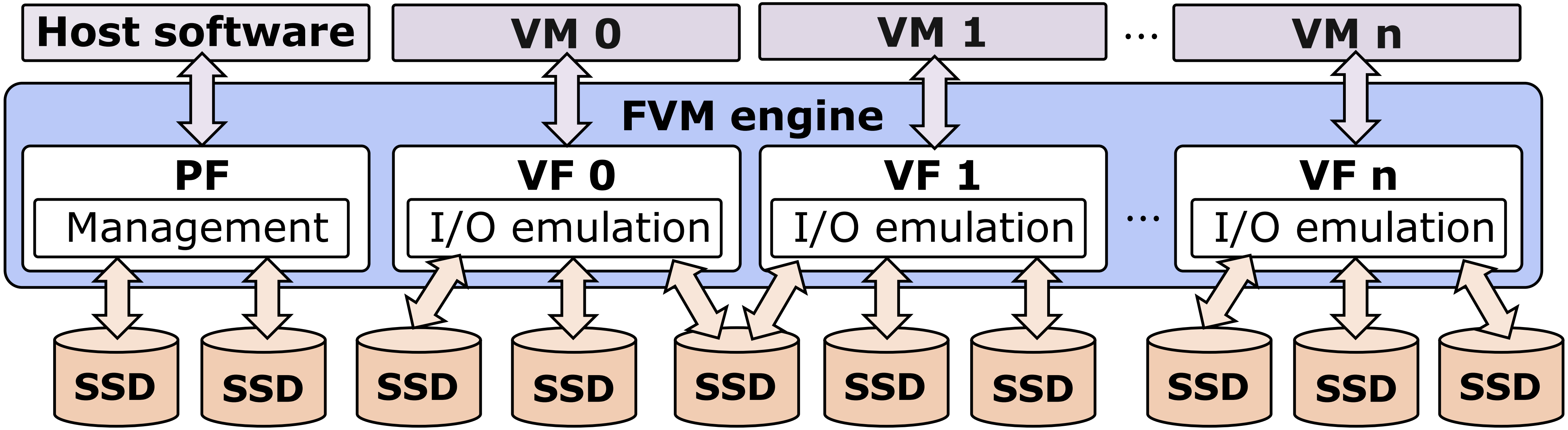


- **FPGA-assisted virtualization**
  - HW-based
  - Host-decoupled
  - Scalable
  - Flexible
  - Programmable

FVM enables fast, scalable, and flexible storage virtualization

Key Idea #1:
HW-level Virtualization Layer

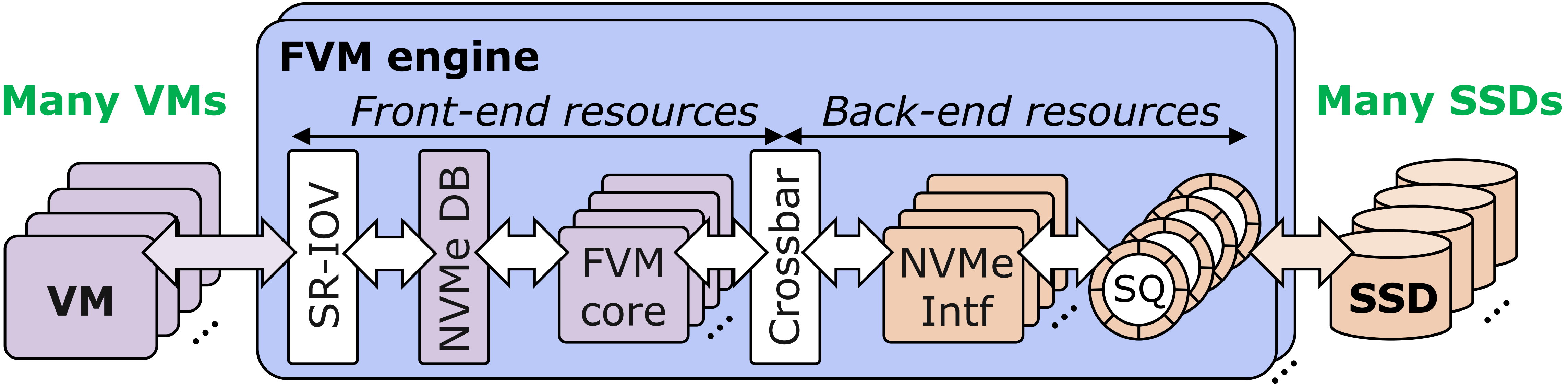
- **Utilize a decoupled FPGA for device emulation**
- **Allow direct access to FVM engine from a VM environment**
  - Save the host resource and enable fast virtualized I/O paths



FVM emulates virtual storage devices without software arbitration

Key Idea #2: Scalable Virtualization Layer

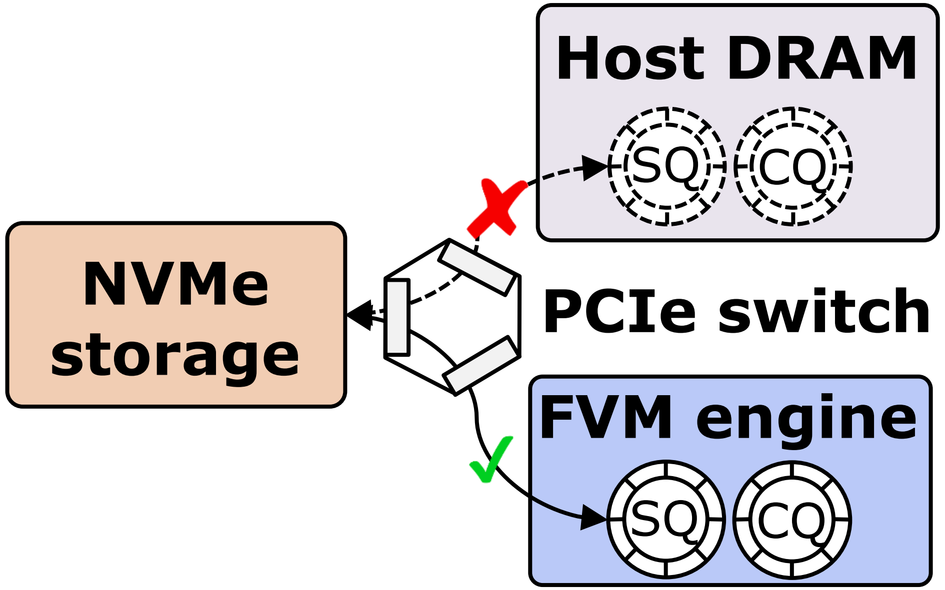
- **Create many front-end / back-end resources**
  - **Front-end:** FVM core – Poll and emulate guest I/O operations
  - **Back-end:** NVMe interface – Manage and control SSDs through PCIe
  - Can scale with a large number of VMs and SSDs

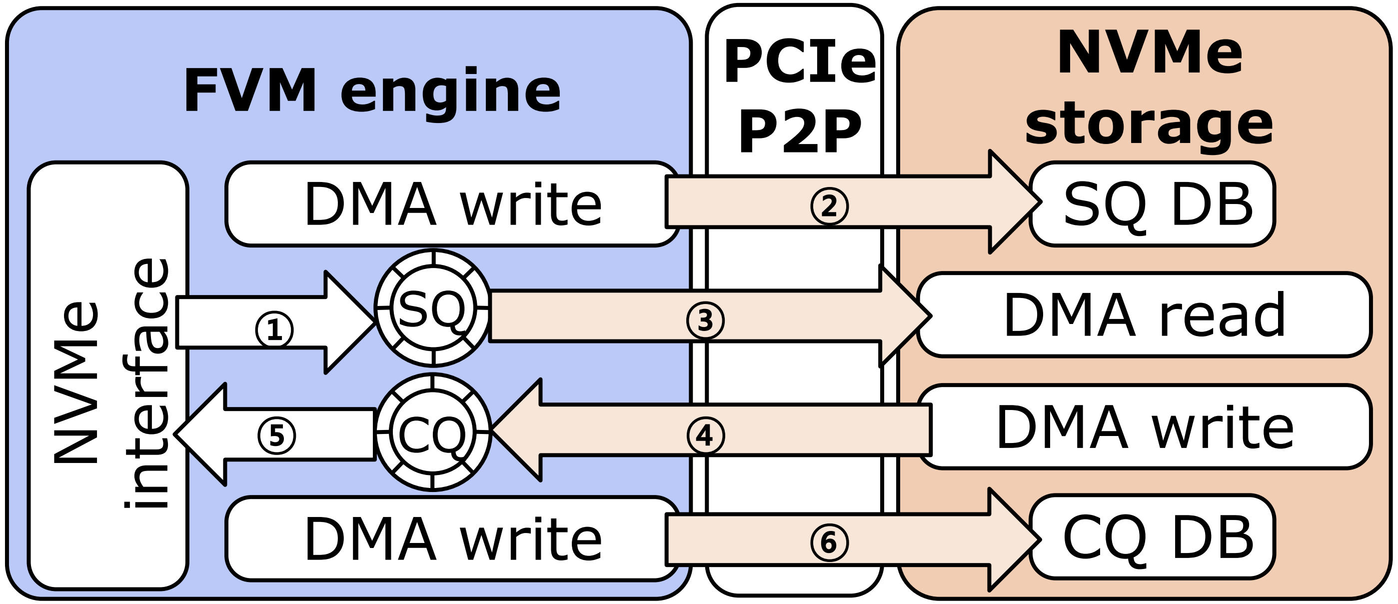


FVM can scale up the virtualization resources with a target storage system

Key Idea #3: Direct Device-Control Mechanism

- **Implement NVMe interfaces on FVM engine**
- **Issue and handle NVMe commands / completions**
  - Interact with NVMe storage devices at the hardware level





FVM manages physical NVMe devices through PCIe P2P

Key Idea #4: HLS-based Design Flow

- **Support C/C++ high-level languages**
- **Allow users to extend virtualization features easily**
- **Exmaple features**
  - Consolidation
  - Caching
  - Replication
  - Throttling
  - Direct (D2D) copy

Category	*Feature*	*SR-IOV*	*FVM*	*(LoC)*
Storage configuration	*Consolidation*	✓	✓	*(40)*
	Caching	✗	✓	*(220)*
Resource management	*Replication*	△	✓	*(15)*
	Throttling	△	✓	*(70)*
Administration	*Direct copy*	✗	✓	*(570)*

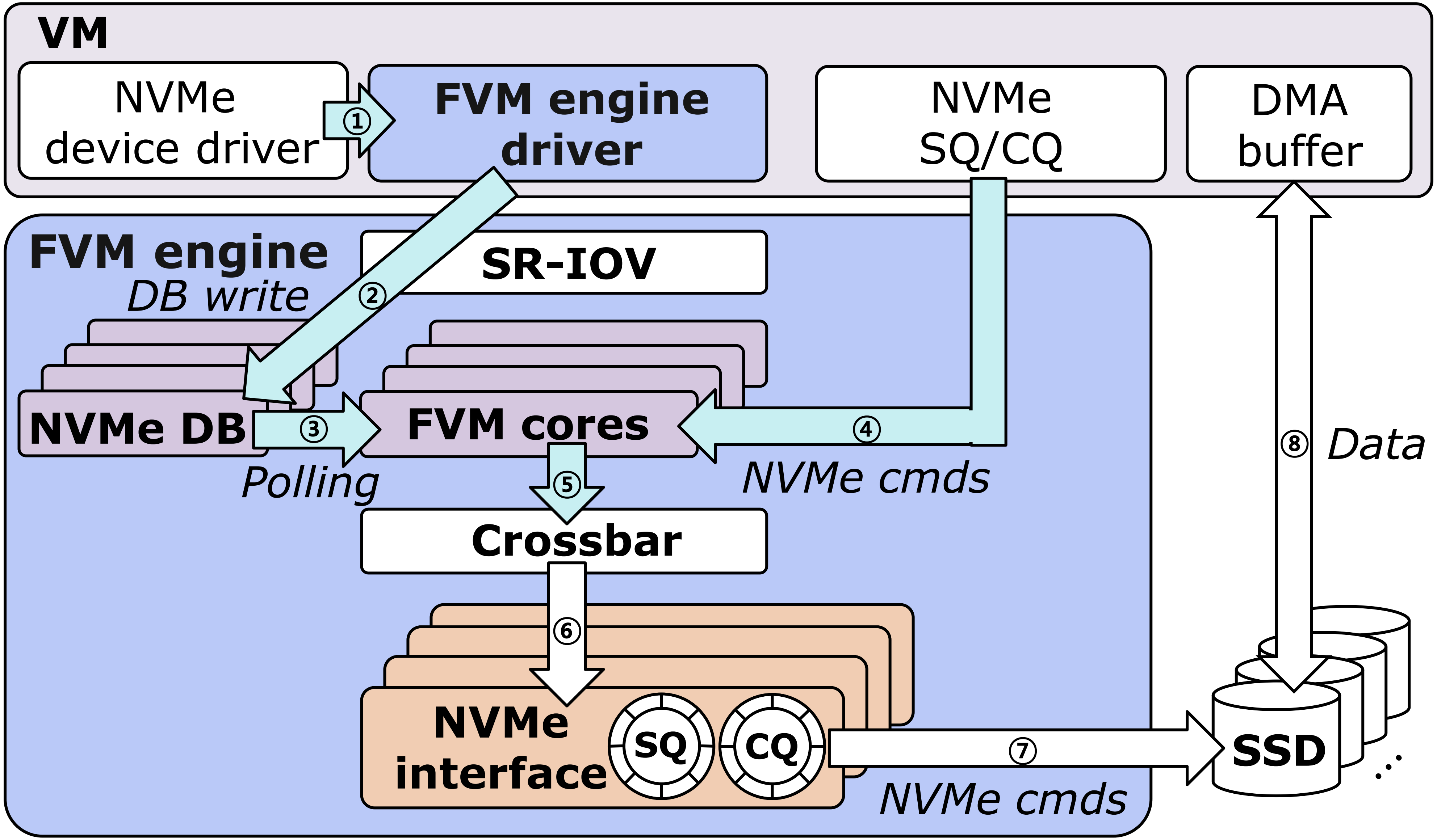
FVM supports easy VM management and feature programmability

HPCS High Performance Computer System Lab

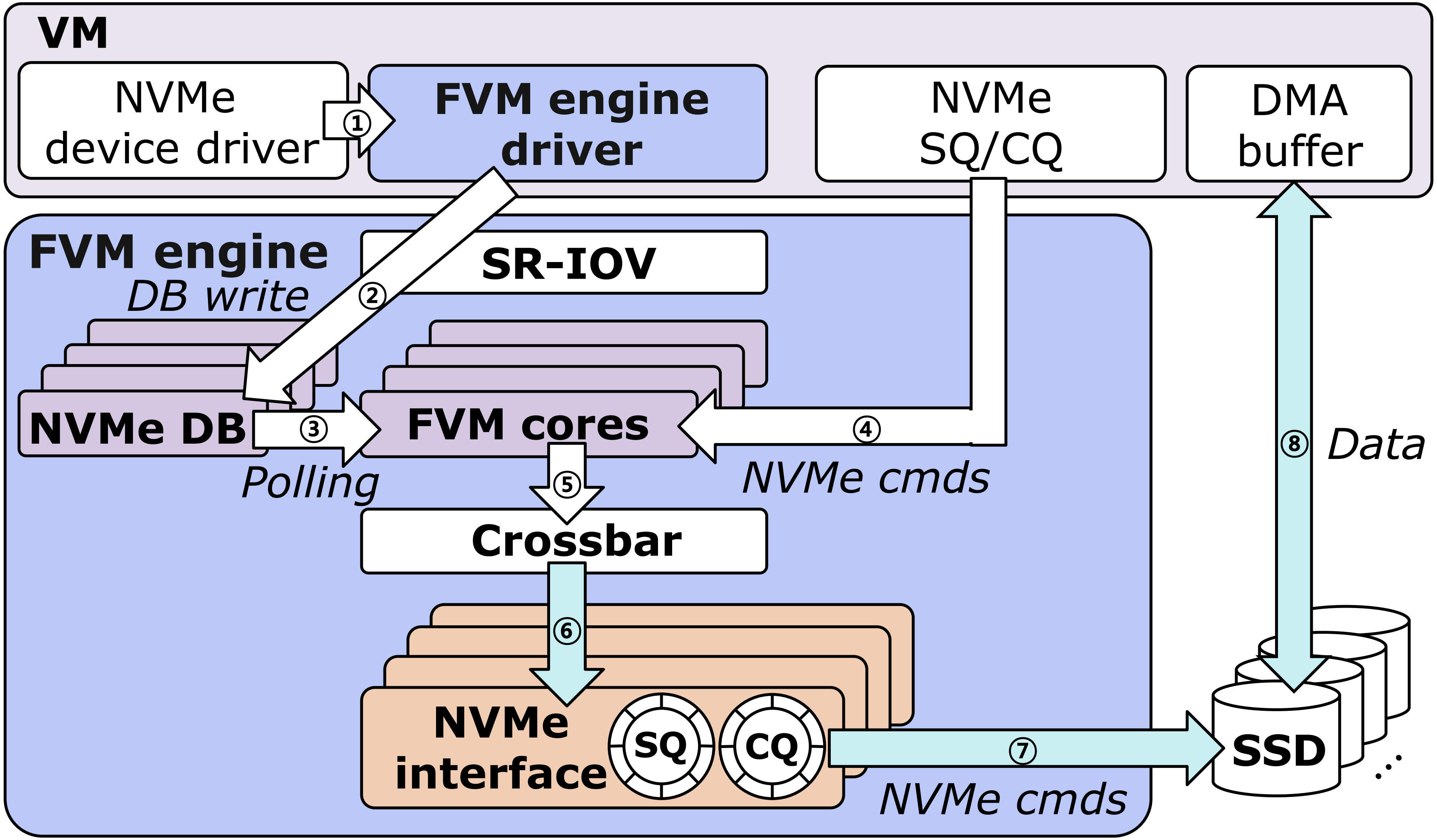
Outline

- Background
- Motivation
- **FVM: FPGA-assisted Storage Virtualization**
  - Key ideas / end-to-end I/O paths
- Evaluation
- Conclusion

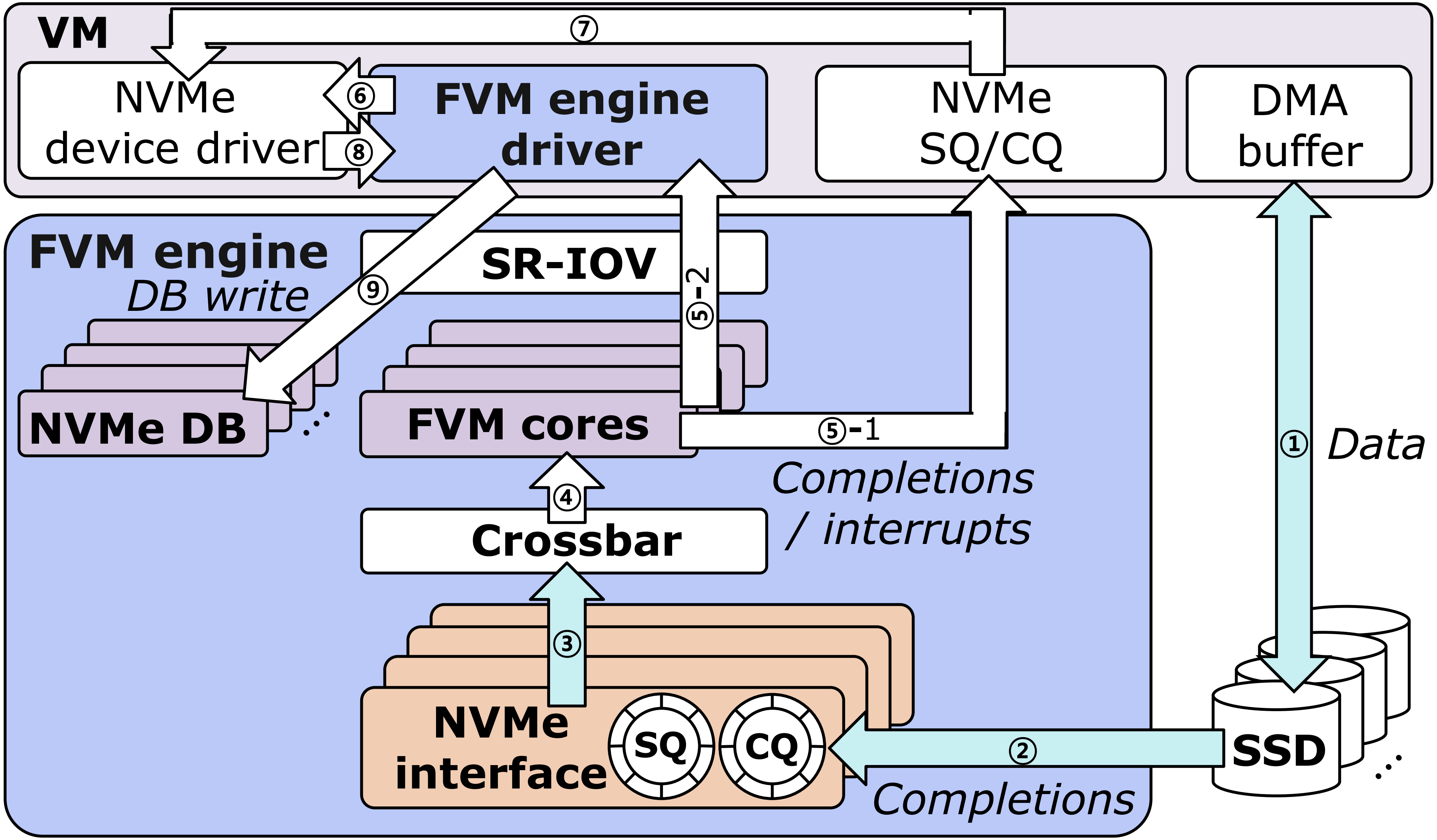
End-to-End Submission Path (1/2)



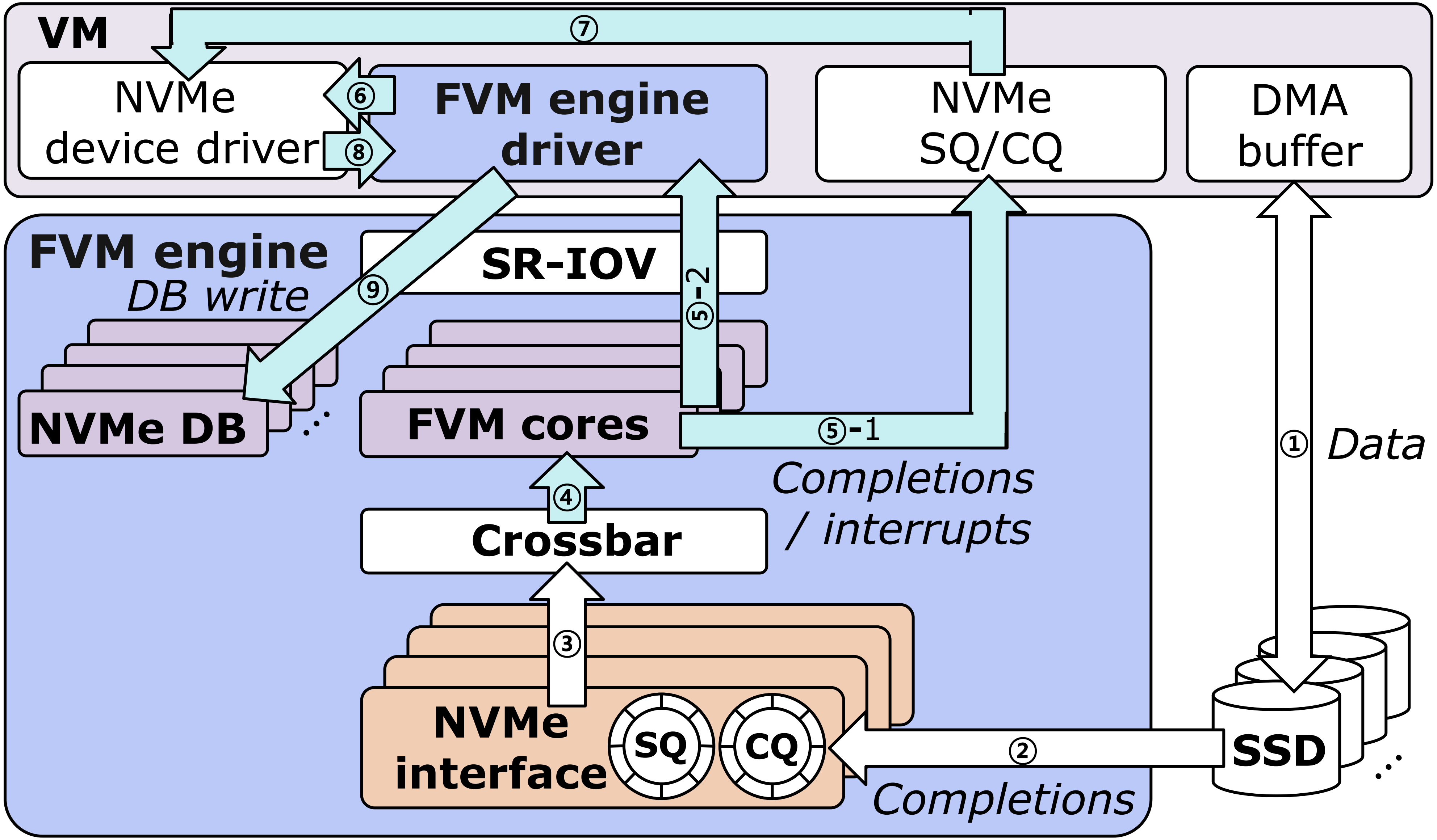
End-to-End Submission Path (2/2)



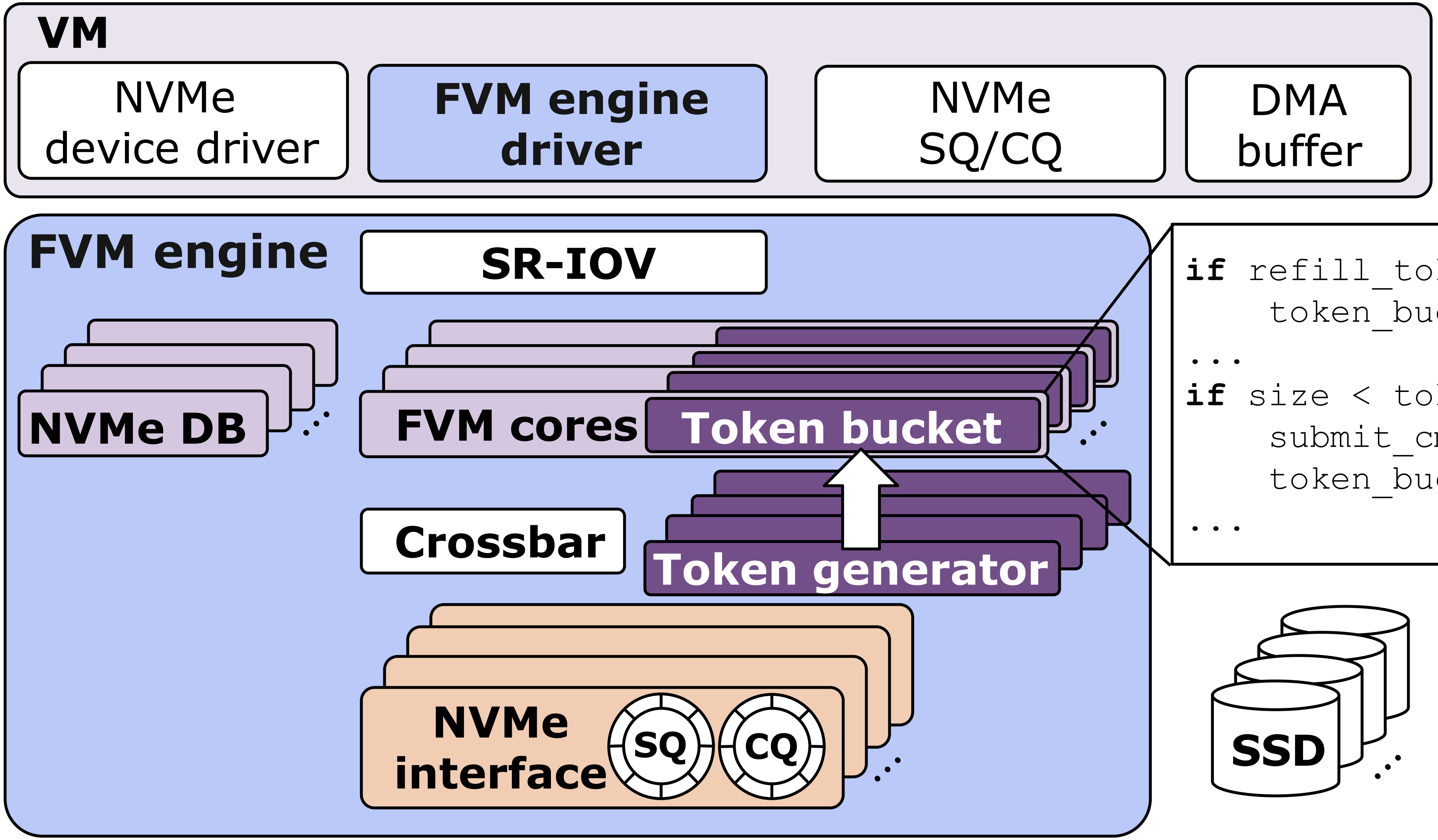
End-to-End Completion Path (1/2)



End-to-End Completion Path (2/2)

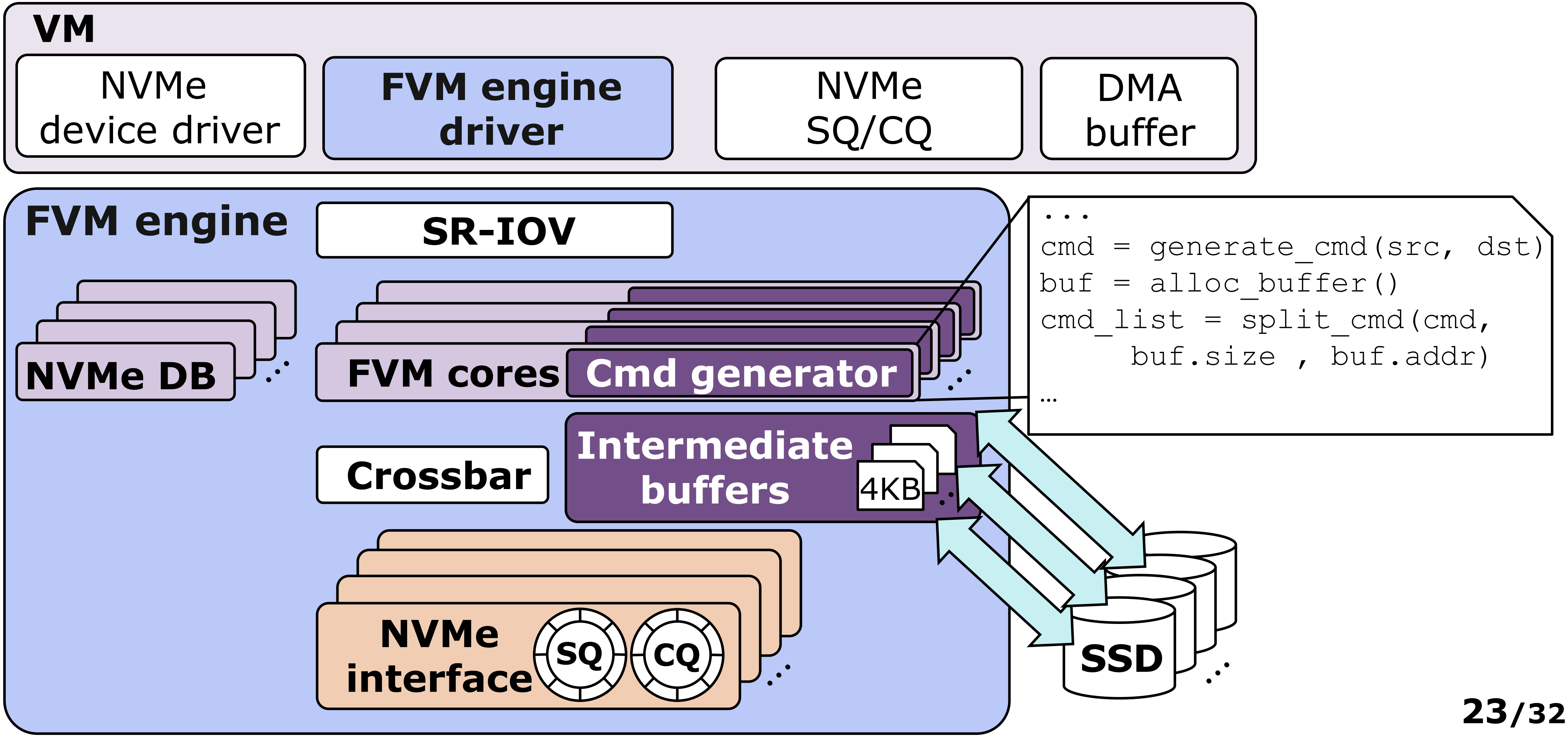


VM-management Feature: Throttling

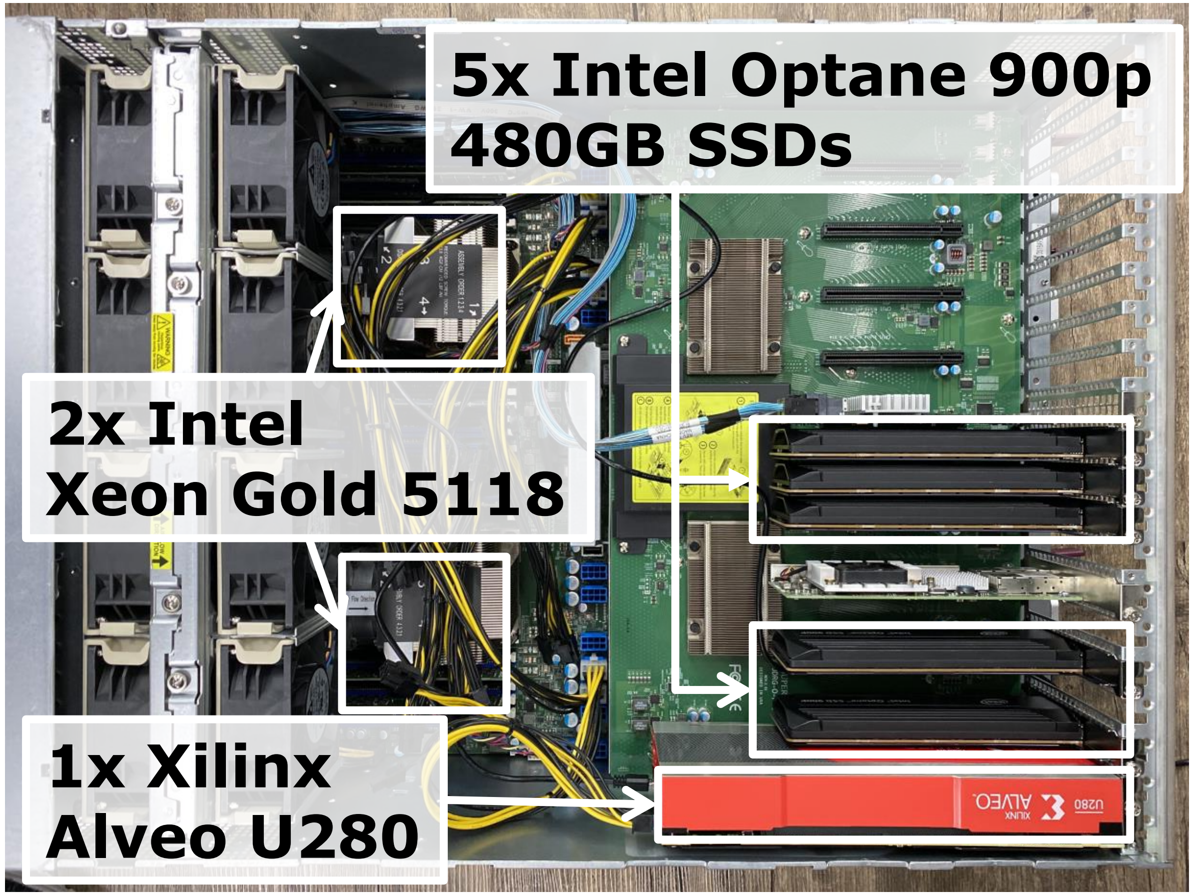



```
if refill_token then
    token_bucket += token
...
if size < token_bucket then
    submit_cmd(cmd)
    token_bucket -= size
...
```

VM-management Feature: D2D copy



Implementation



- **Prototype**
  - 2x 12-core Xeon 5118 / 256GB
  - 5x 480GB NVMe SSDs
  - Xilinx Alveo U280 Card
- **Based on open-source SW frameworks**
  - Linux kernel v5.3
  - KVM/QEMU v3.0
  - SPDK vhost-nvme v20.01

Outline

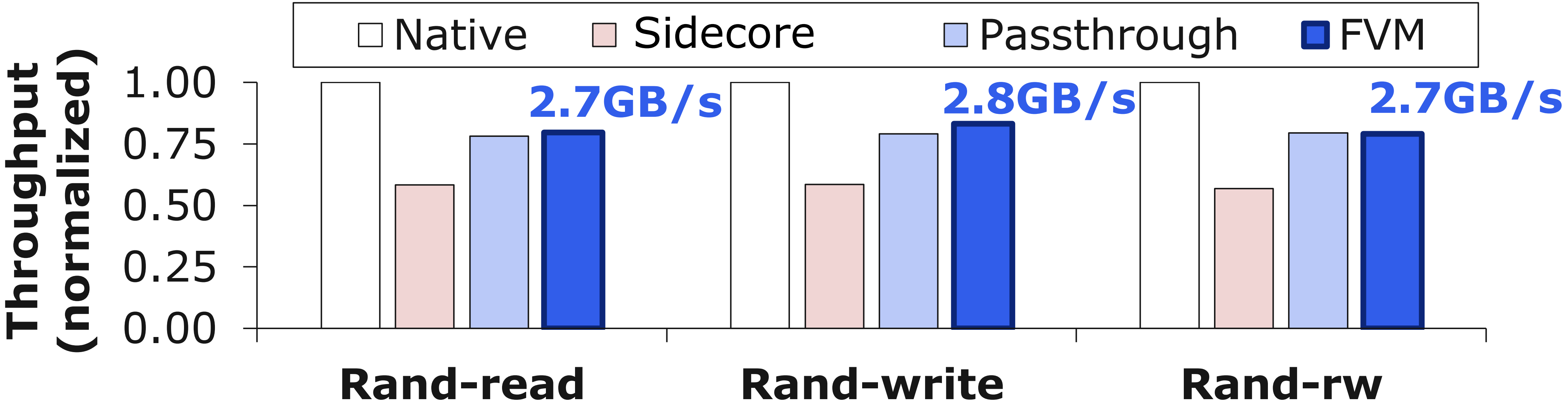
- Background
- Motivation
- FVM: FPGA-assisted Storage Virtualization
- **Evaluation**
- Conclusion

Evaluation Methodology

- **FVM vs host sidecore, passthrough (ideal perf.)**
- **Random I/O performance from VMs**
  - FIO random-read/write/rw (4 threads, 32 queue depth)
  - I/O throughput, host CPU usage measurement
- **RocksDB performance with multiple threads from VMs**
  - (A) 50% read, (B) 95% read, (C) read-only, (D) read-latest, (E) short-range, (F) read-modifiy-write workloads
  - RocksDB operation throughput measurement
- **Scability test with multiple VMs and SSDs**

Random I/O Performance

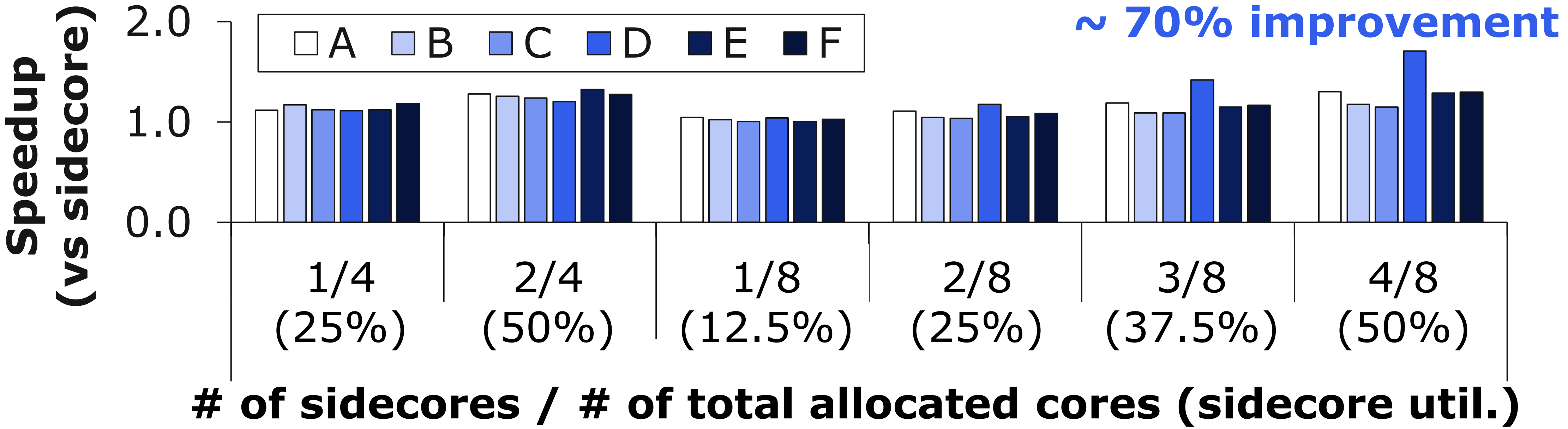
- **Random I/O with limited CPU usage (4 cores)**
  - **Host sidecore:** incur CPU contention between VMs and sidecores
  - **Passthrough/FVM:** decouple virtualization from host resources



FVM achieves 1.37x – 1.42x higher I/O throughput than sidecore approaches

RocksDB Operation Throughput

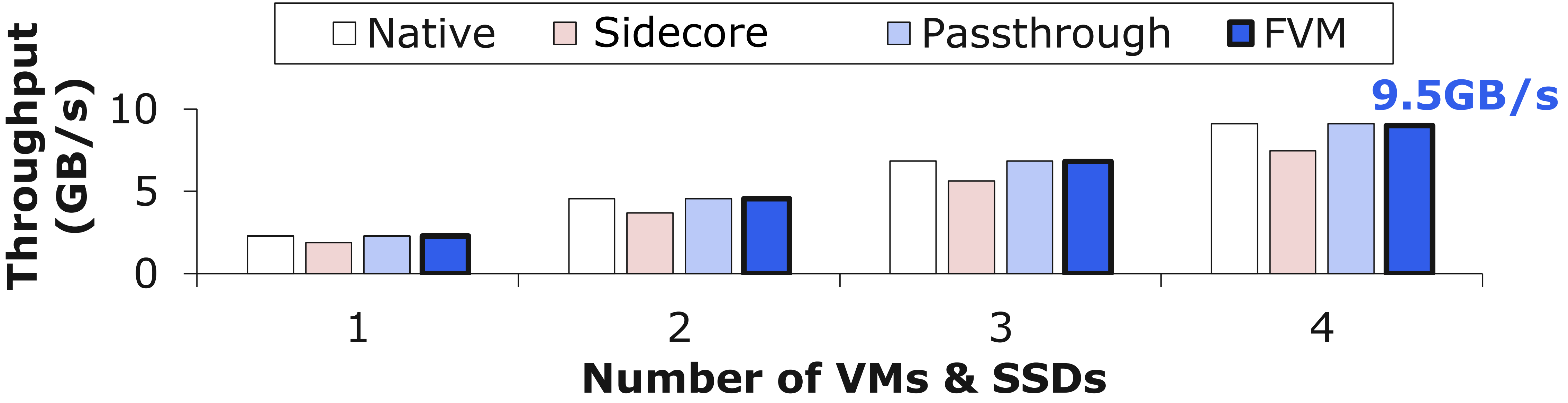
- **RocksDB workloads with 4-8 CPU cores**
  - **Host sidecore:** limit VM performance due to lack of host resources
  - **FVM:** save host resources and offer more compute power to VMs



With FVM, host CPUs are better spent for VM workloads and user applications

Scalability Test

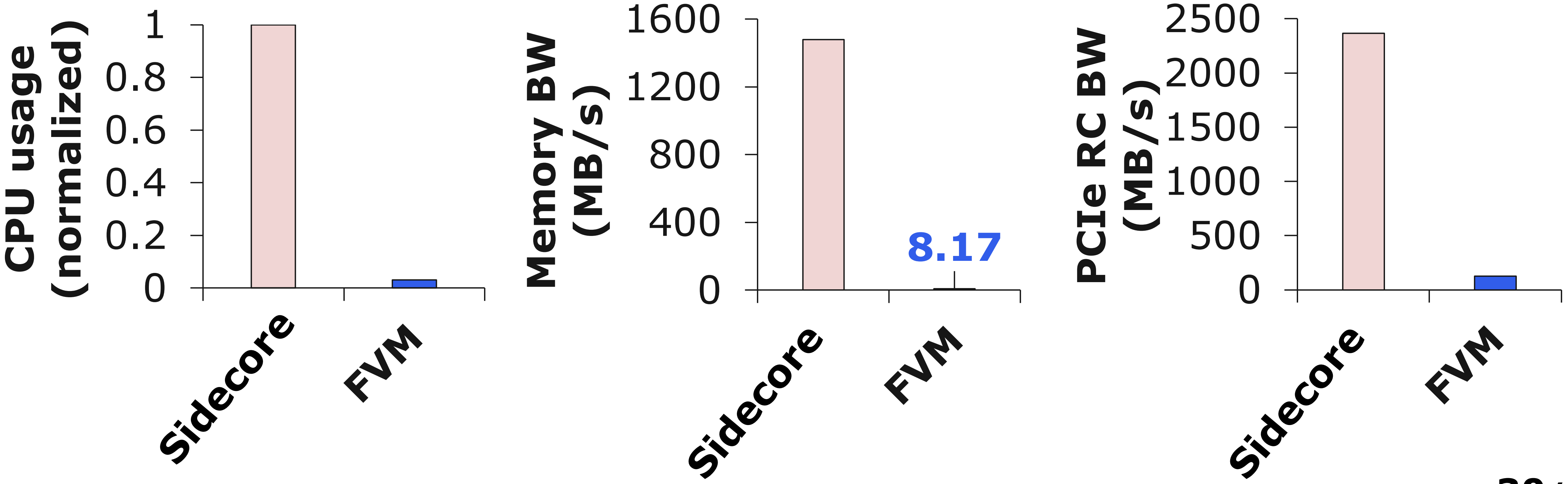
- **An increasing # of VMs (1 SSD/4 cores/VM)**
  - **Host sidecore:** pay the virtualization tax with an increasing # of VMs
  - **FVM:** scale up the virtualization resources with a target # of VMs/SSDs



FVM provides scalable performance by adding more FVM cores or FPGAs

Example Feature: D2D Copy

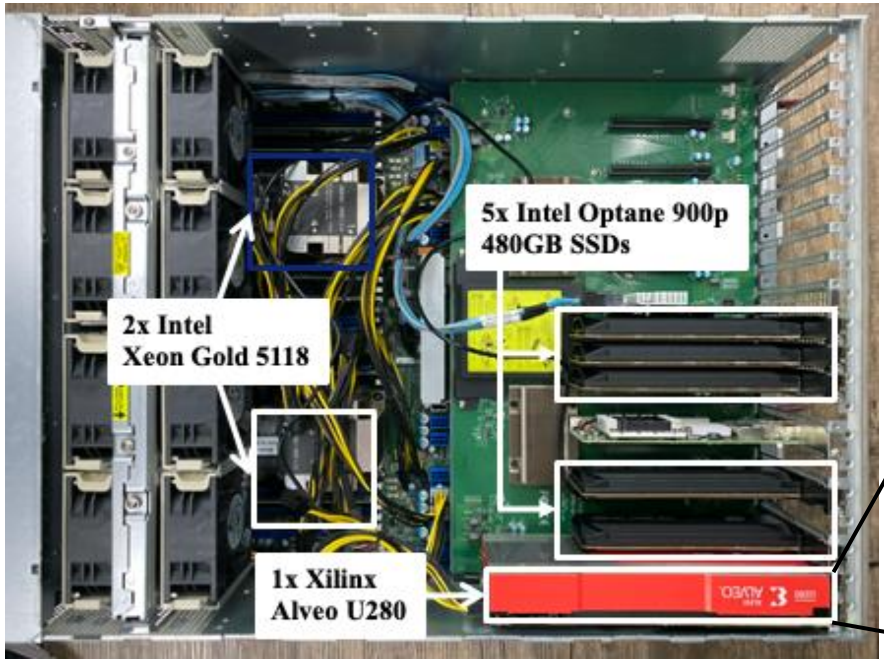
- **Direct device-to-device copy through P2P**
  - Vs. SW-based indirect data copy
  - Host resource saving: CPU, host memory / root complex BW

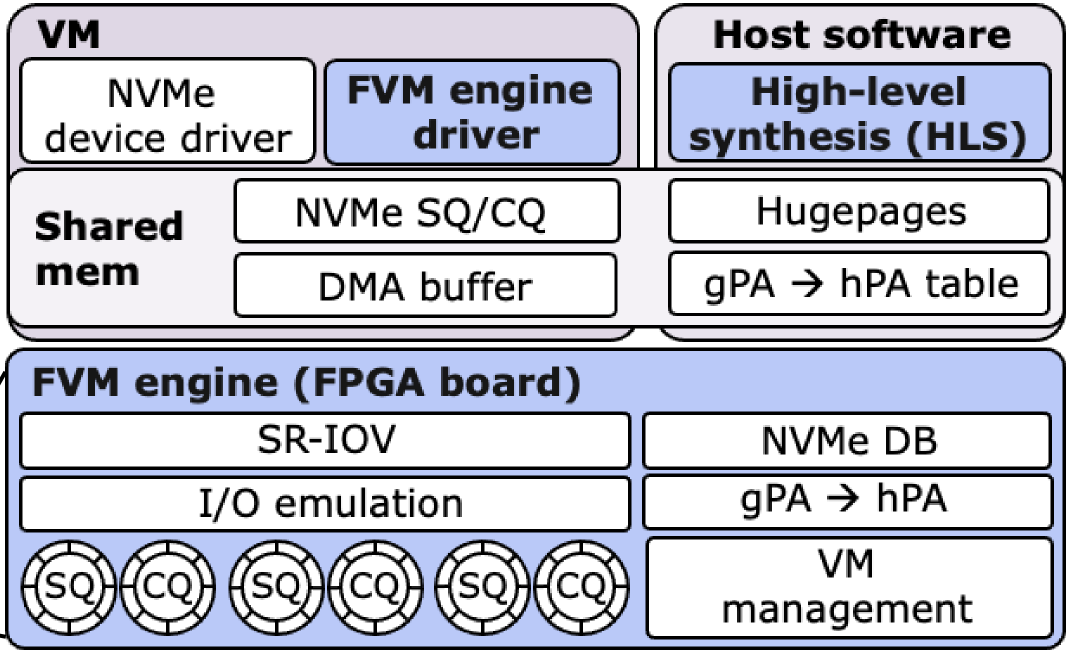


Discussion & Conclusion

- **Cost analysis**
  - CPU saving: 20 cores in a 64-core machine ≈ $2000 - $6400
  - Small FPGA resource usage for FVM engine
- **FVM: FPGA-assisted storage virtualization**
  - HW-based and scalable virtualization layer
  - Direct device-control mechanism
  - HLS-based design flow
- **Implementation with off-the-shelf FPGA/SSD devices**
  - 1.37x – 1.42x higher I/O performance than sidecore approaches
  - 9.5 GB/s aggregate throughput with 4 NVMe SSDs

Thank You!





FVM: FPGA-assisted Virtual Device Emulation for Fast, Scalable, and Flexible Storage Virtualization

Dongup Kwon, dongup@snu.ac.kr