Accelerating virtual machine access with Storage performance development kit

Anu H Rao

Storage Software Product line Manager

Datacenter Group, Intel® Corp

Notices & Disclaimers

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. Check with your system manufacturer or retailer or learn more at intel.com.

No computer system can be absolutely secure.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/benchmarks .

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit http://www.intel.com/benchmarks .

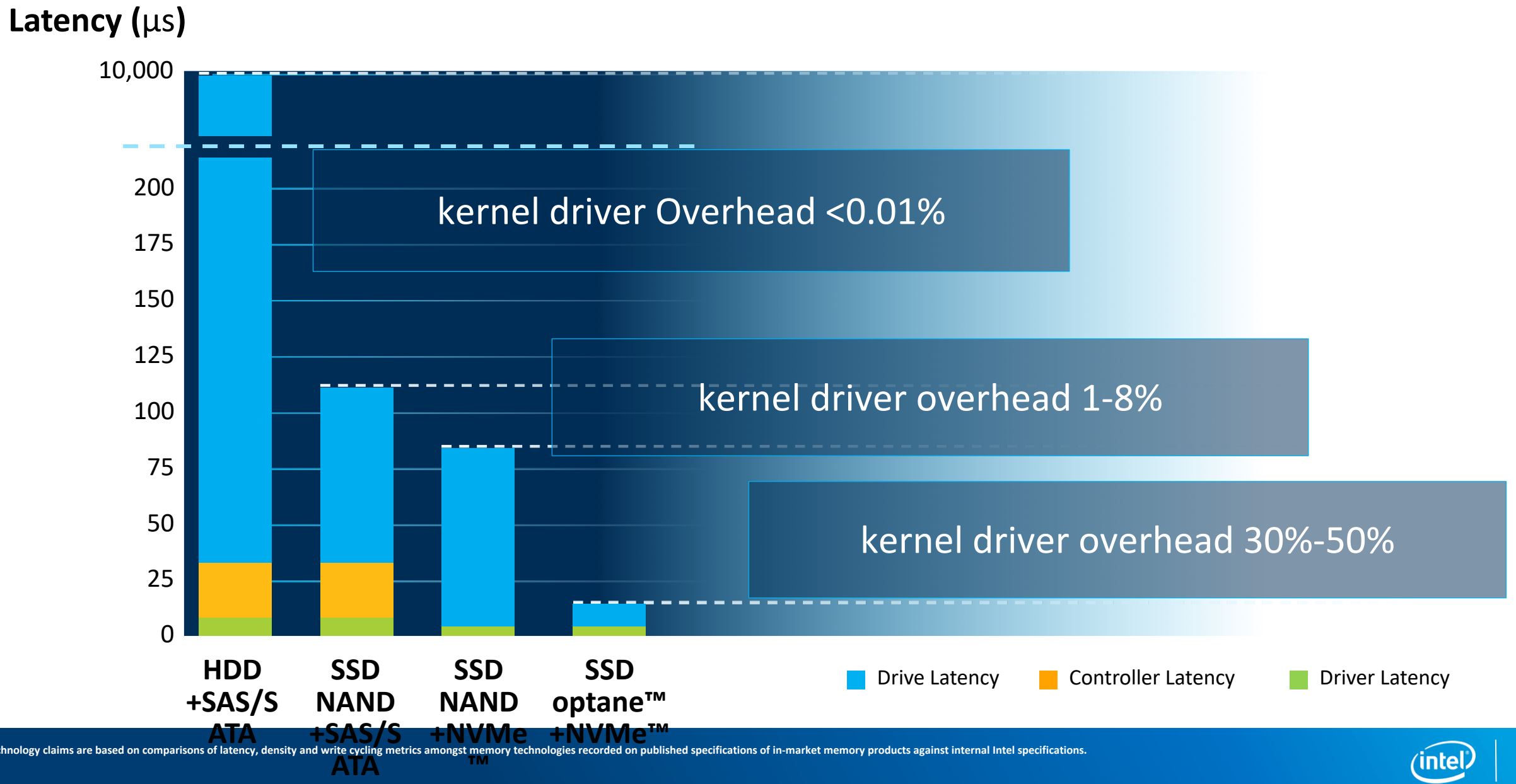
Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Cost reduction scenarios described are intended as examples of how a given Intel-based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.

Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

introduction

The Challenge: Media Latency



Storage Performance Development Kit

Scalable and Efficient Software Ingredients

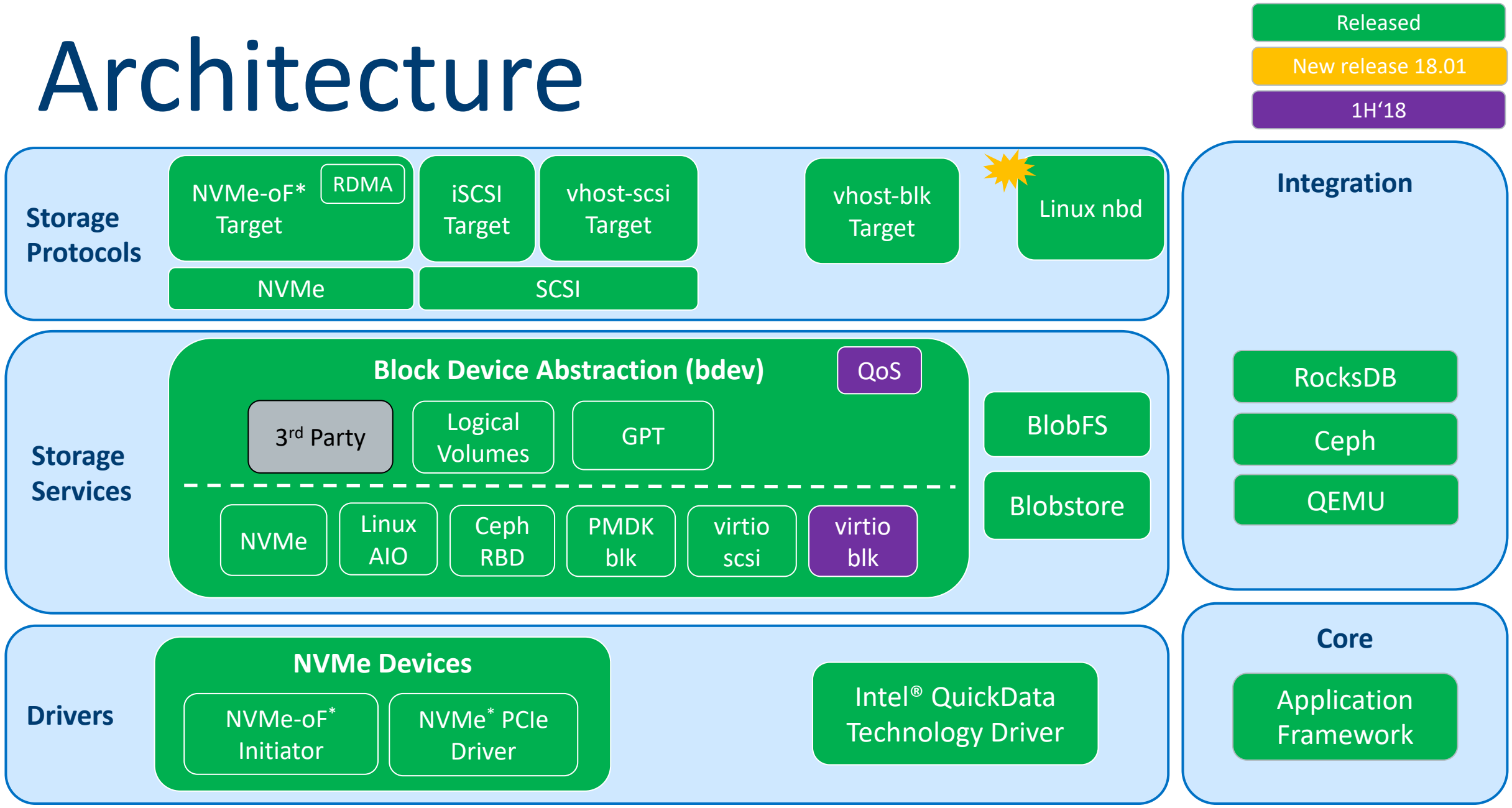
- User space, lockless, polled-mode components
- Up to millions of IOPS per core
- Designed to extract maximum performance from non-volatile media

Storage Reference Architecture

- Optimized for *latest generation CPUs and SSDs*
- Open source composable building blocks (BSD licensed)
- Available via SPDK.io
- Follow @SPDKProject on twitter for latest events and activities

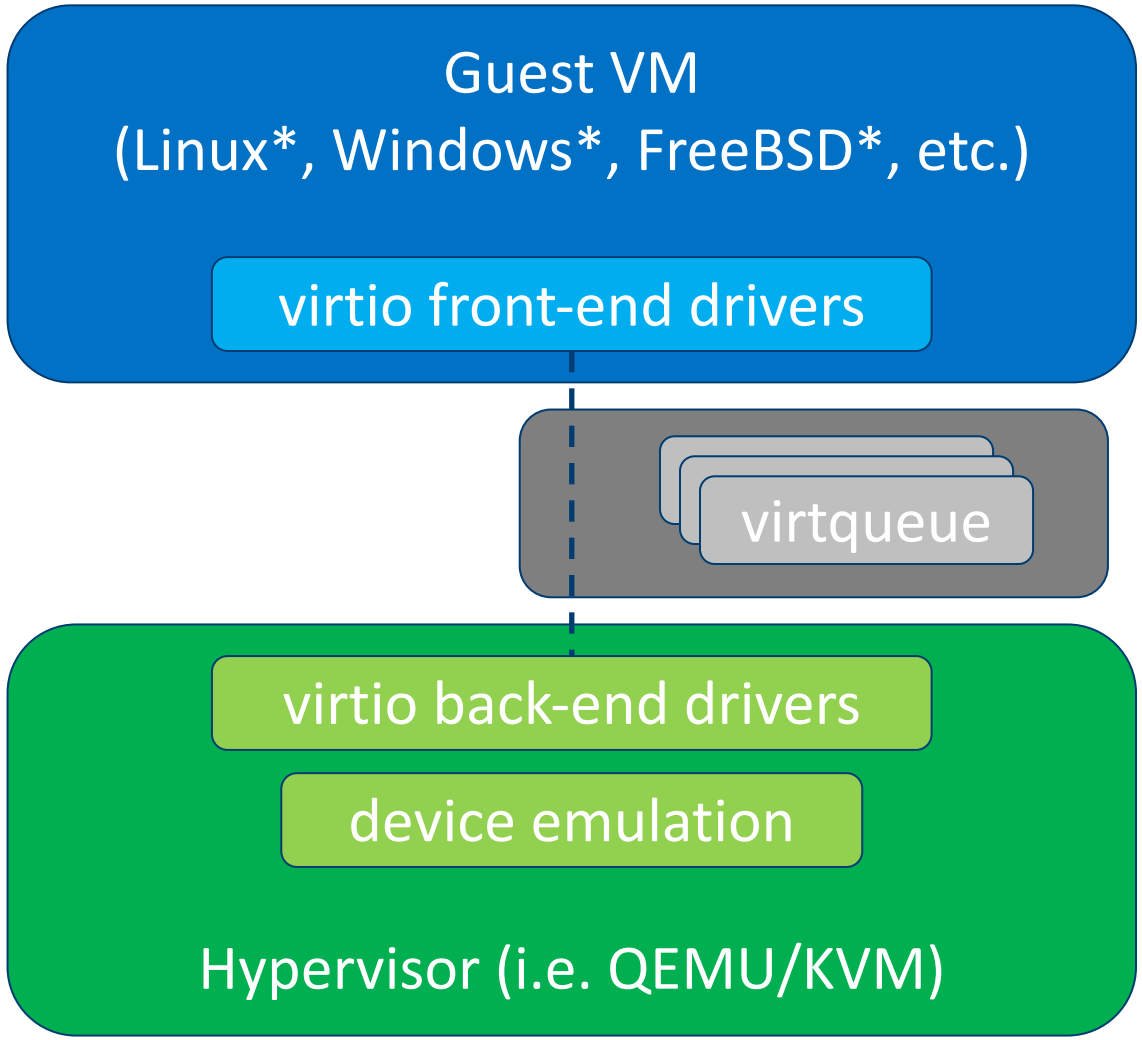
architecture

Architecture



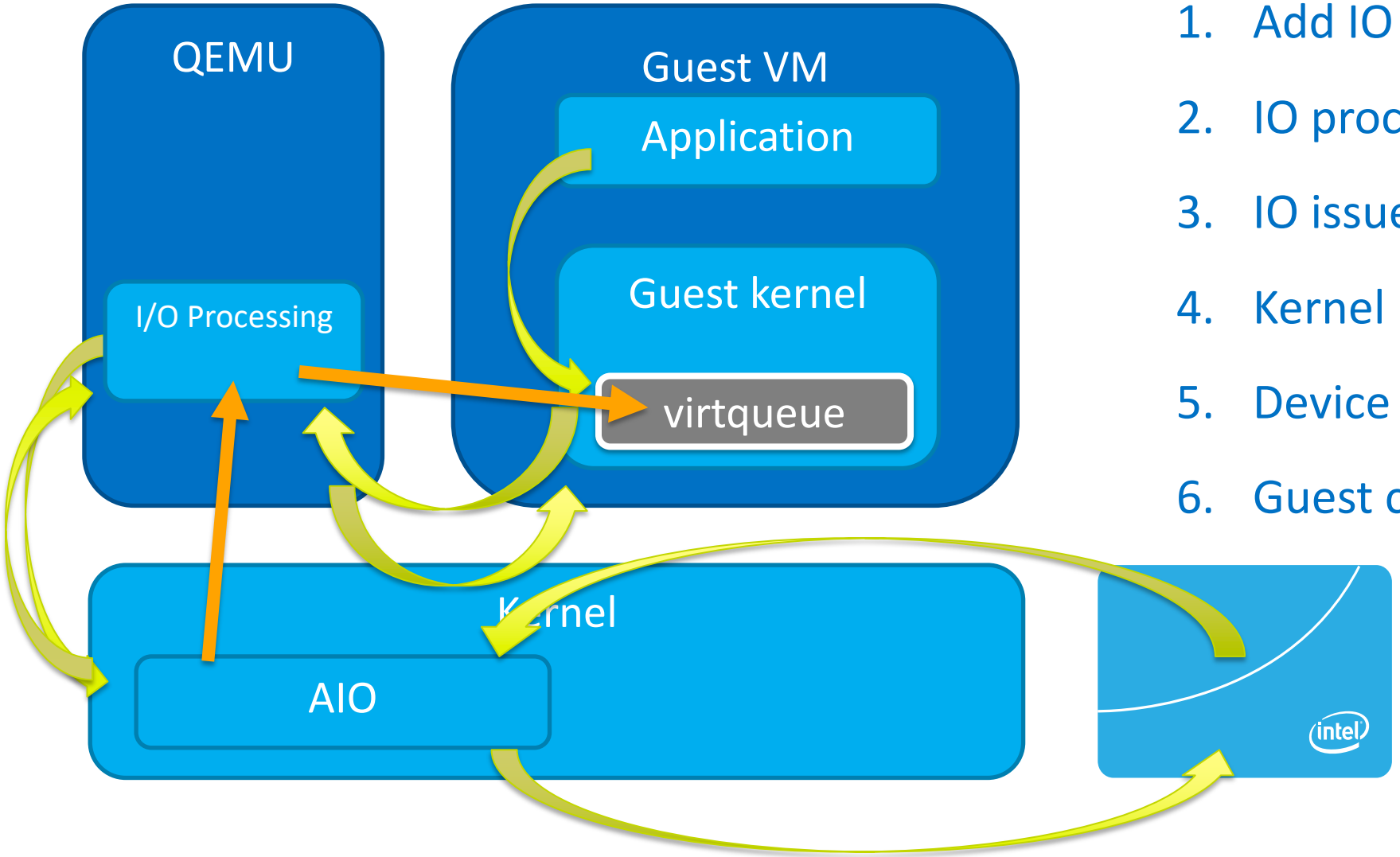
VHOst deep dive

Virtio



- Paravirtualized driver specification
- Common mechanisms and layouts for device discovery, I/O queues, etc.
- virtio device types include:
  - virtio-net
  - virtio-blk
  - virtio-scsi
  - virtio-gpu
  - virtio-rng
  - virtio-crypto

QEMU VirtIO SCSI

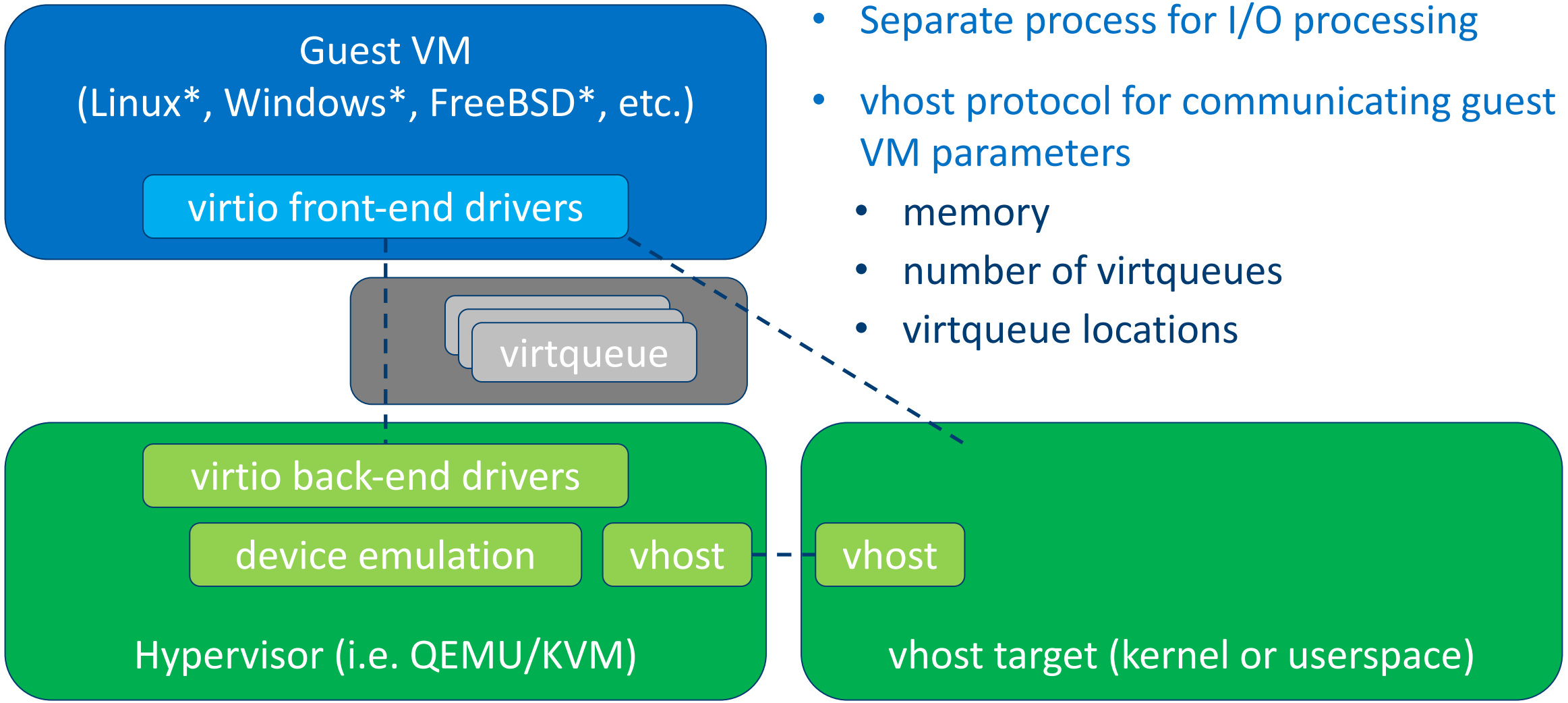


1. Add IO to virtqueue
2. IO processed by QEMU
3. IO issued to kernel
4. Kernel pins memory
5. Device executes IO
6. Guest completion interrupt

QEMU VIRTIO

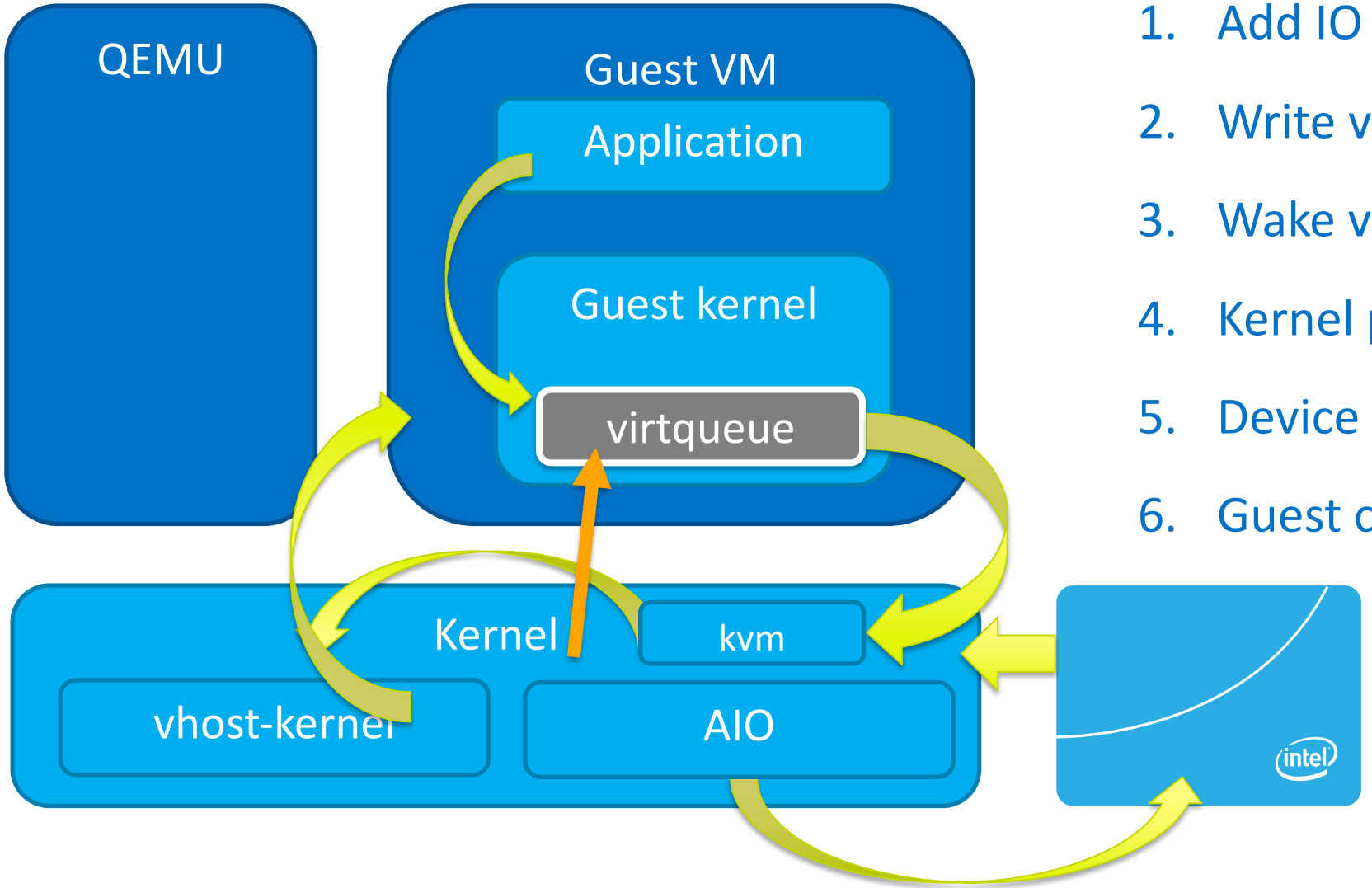
Vhost (KERNEL)

Vhost



- Separate process for I/O processing
- vhost protocol for communicating guest VM parameters
  - memory
  - number of virtqueues
  - virtqueue locations

Kernel VHOST



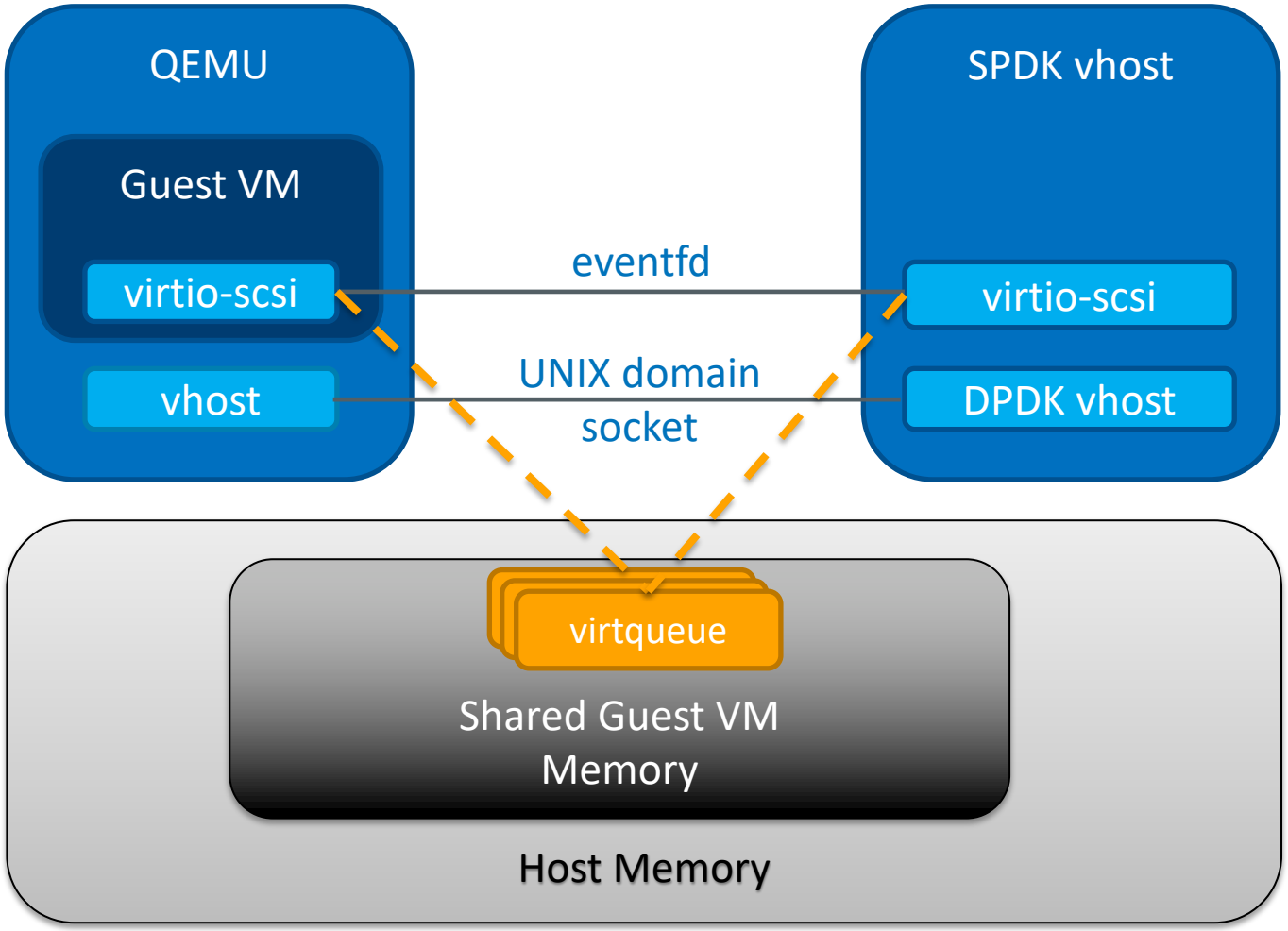
1. Add IO to virtqueue
2. Write virtio doorbell
3. Wake vhost kernel
4. Kernel pins memory
5. Device executes IO
6. Guest completion interrupt

QEMU VIRTIO

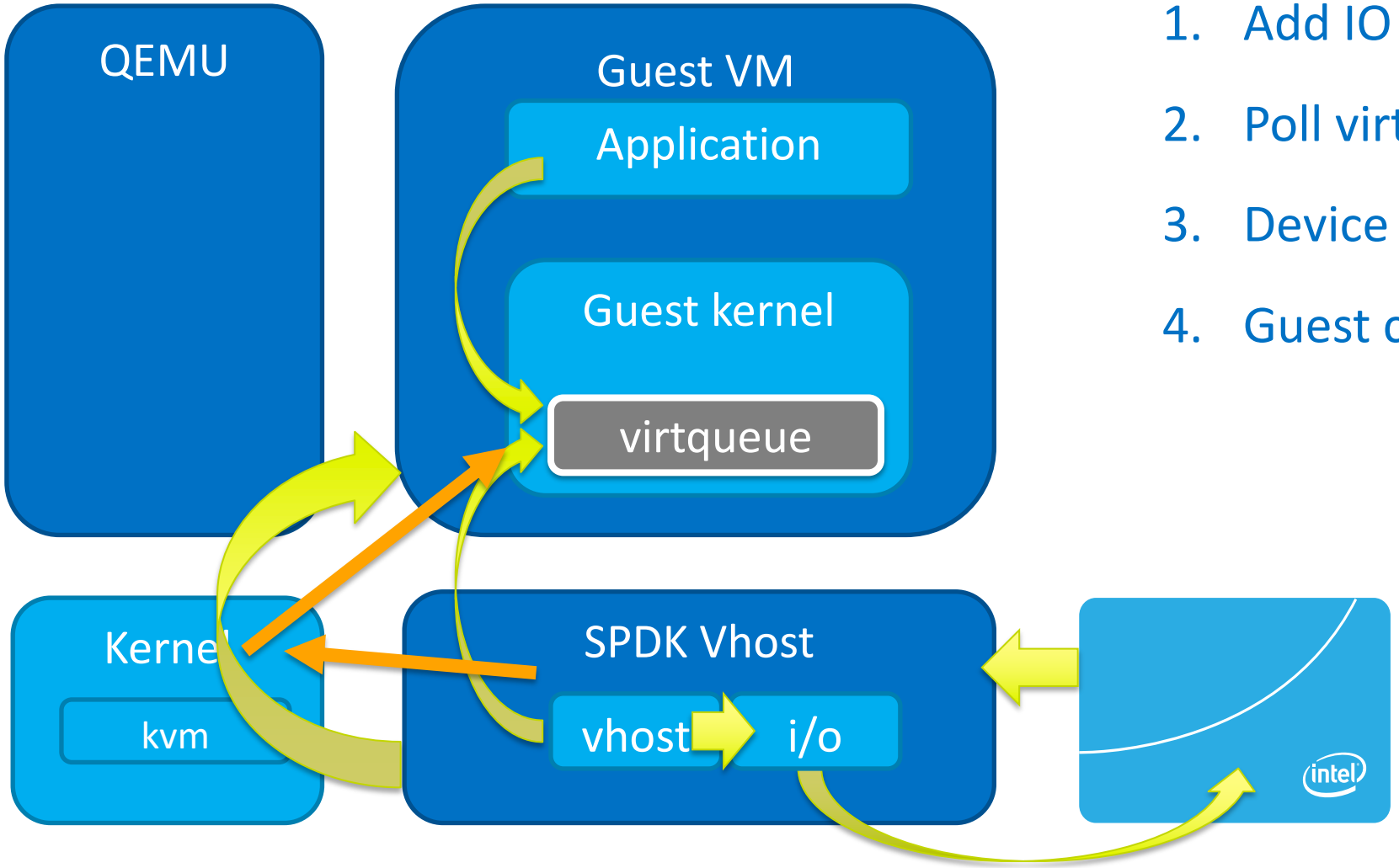
Vhost (KERNEL)

vhost (USERSPACE)

SPDK VHOST Architecture

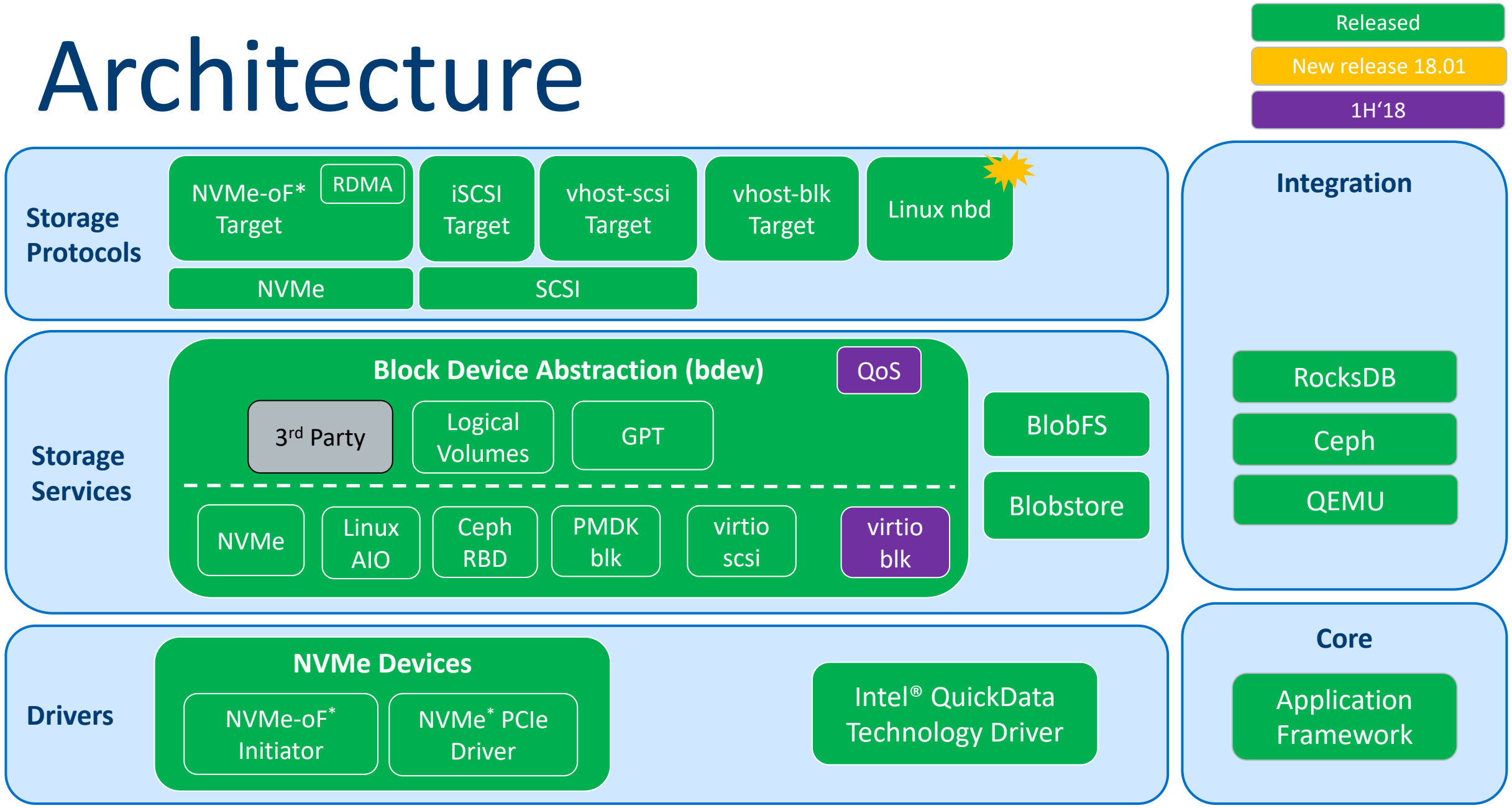


SPDK VHOST



1. Add IO to virtqueue
2. Poll virtqueue
3. Device executes IO
4. Guest completion interrupt

Architecture



Sharing SSDs in userspace

Typically not 1:1 VM to local attached NVMe SSD

- otherwise just use PCI direct assignment

What about SR-IOV?

- SR-IOV SSDs not prevalent yet
- precludes features such as snapshots

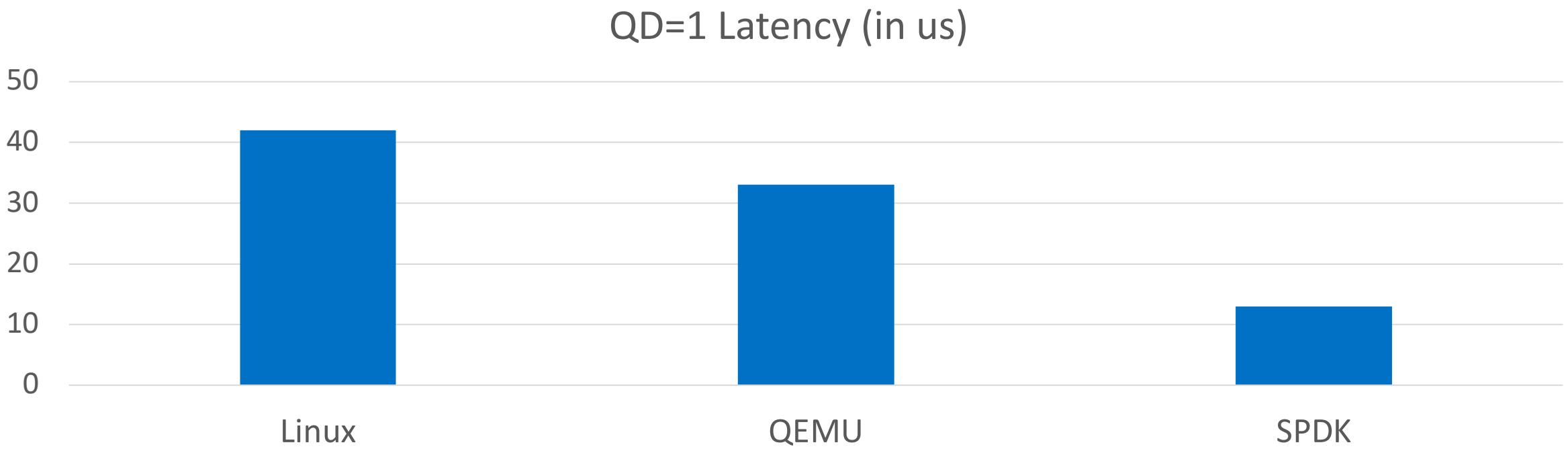
What about LVM?

- LVM depends on Linux kernel block layer and storage drivers (i.e. nvme)
- SPDK wants to use userspace polled mode drivers

SPDK Blobstore and Logical Volumes!

KVM/QEMU Virtualization Performance & efficiency

SPDK vhost Performance



SPDK up to 3x better efficiency and latency

System Configuration: 2S Intel® Xeon® Platinum 8180: 28C, E5-2699v3: 18C, 2.5GHz (HT off), Intel® Turbo Boost Technology enabled, 12x16GB DDR4 2133 MT/s, 1 DIMM per channel, Ubuntu* Server 16.04.2 LTS, 4.11 kernel, 23x Intel® P4800x Optane SSD – 375GB, 1 SPDK lvolstore or LVM lvgroup per SSD, SPDK commit ID c5d8b108f22ab, 46 VMs (CentOS 3.10, 1vCPU, 2GB DRAM, 100GB logical volume), vhost dedicated to 10 cores

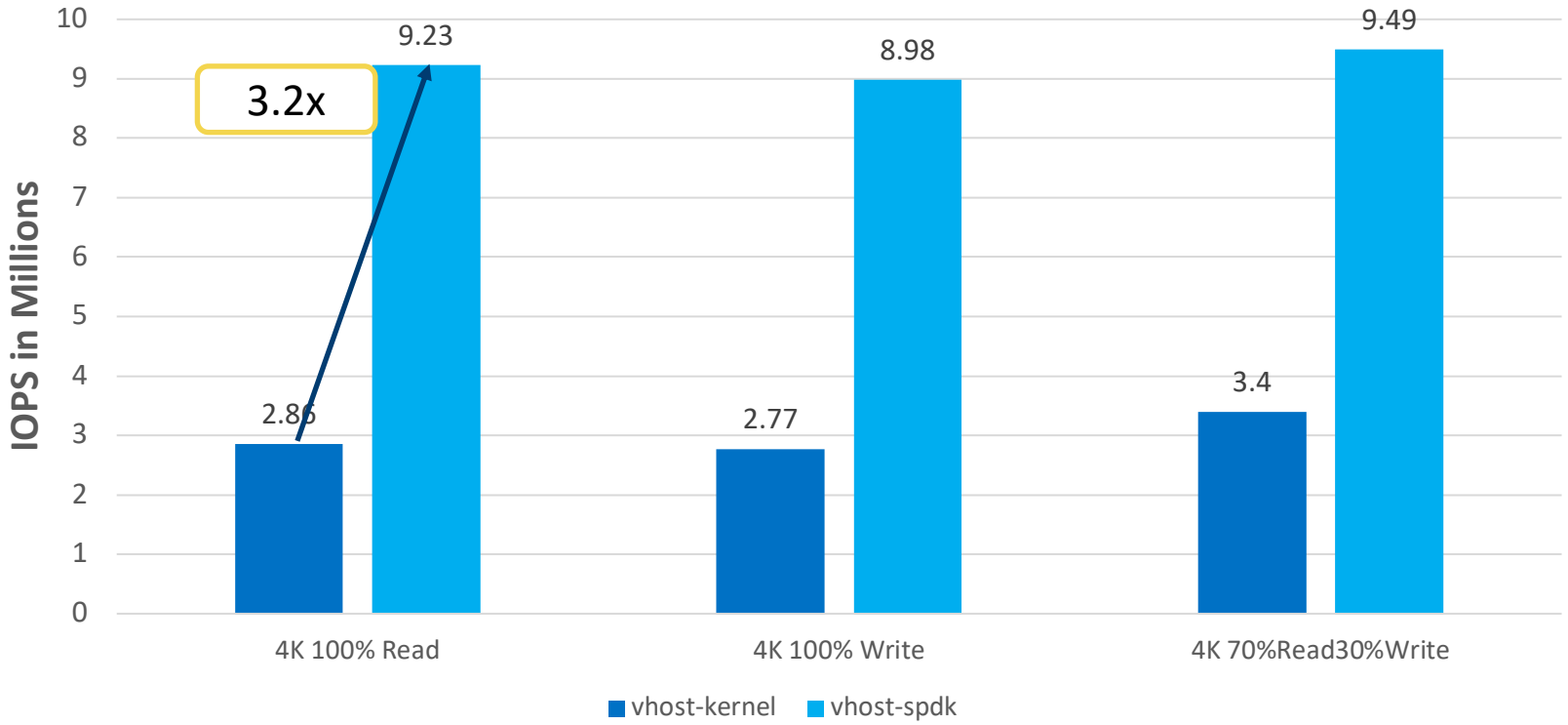
As measured by: fio 2.10.1 – Direct=Yes, 4KB random read I/O, Ramp Time=30s, Run Time=180s, Norandommap=1, I/O Engine = libaio, Numjobs=1

Legend: Linux: Kernel vhost-scsi QEMU: virtio-blk dataplane SPDK: Userspace vhost-scsi

48 VMs: vhost-scsi performance (SPDK vs. Kernel)

Intel Xeon Platinum 8180 Processor, 24x Intel P4800x 375GB

2 partitions per VM, 10 vhost I/O processing cores



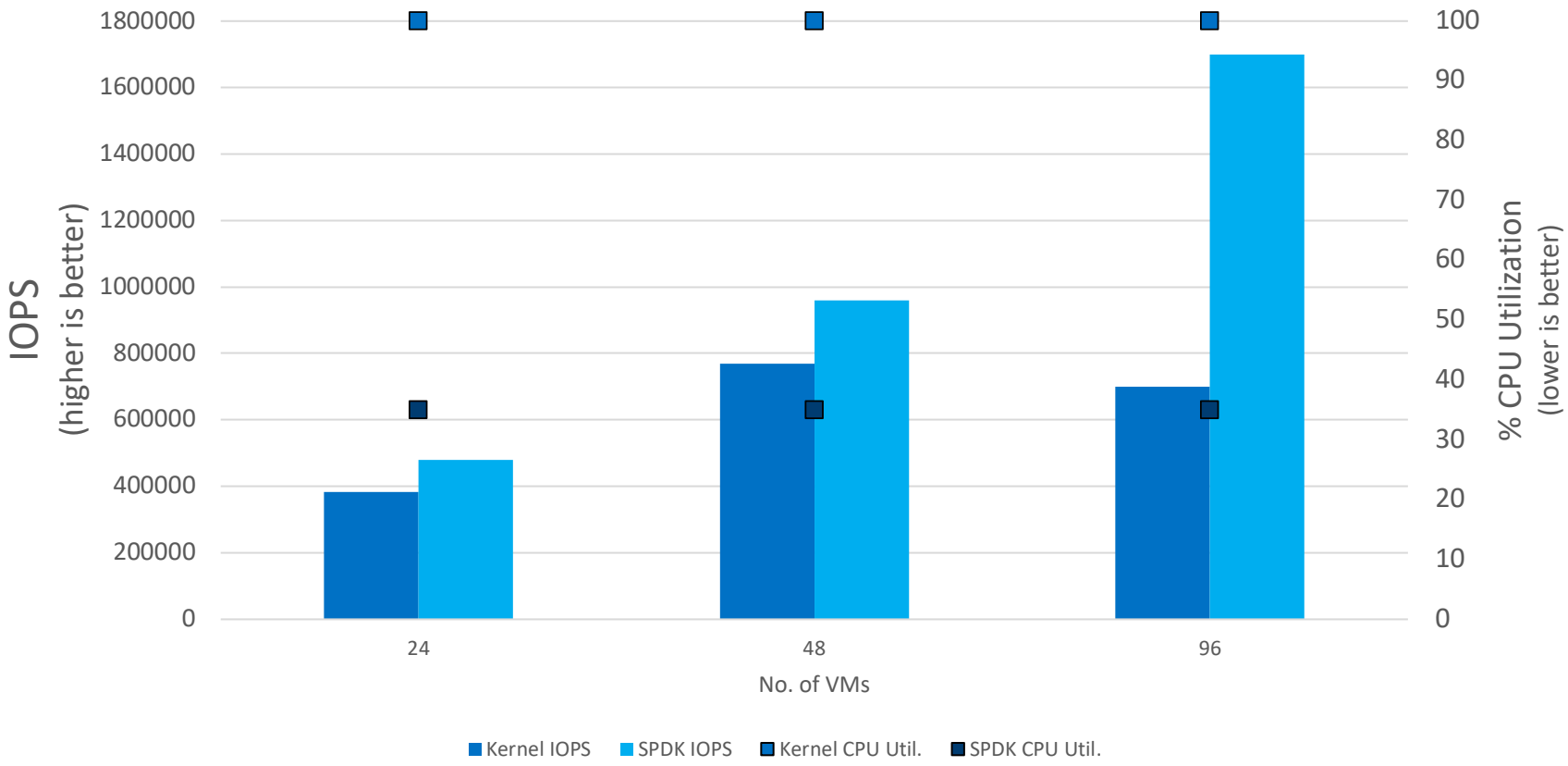
- Aggregate IOPS across all 48x VMs reported. All VMs on separate cores than vhost-scsi cores.
- 10 vhost-scsi cores for I/O processing
- SPDK vhost-scsi up to 3.2x better with 4K 100% Random read I/Os
- Used cgroups to restrict kernel vhost-scsi processes to 10 cores

System Configuration:Intel Xeon Platinum 8180 @ 2.5GHz. 56 physical cores 6x 16GB, 2667 DDR4, 6 memory Channels, SSD: Intel P4800x 375GB x24 drives, Bios: HT disabled, p-states enabled, turbo enabled, Ubuntu 16.04.1 LTS, 4.11.0 x86_64 kernel, 48 VMs, number of partition: 2, VM config : 1core 1GB memory, VM OS: fedora 25, blk-mq enabled, Software packages: Qemu-2.9, libvirt-3.0.0, spdk (3bfecec994), IO distribution: 10 vhost-cores for SPDK / Kernel. Rest 46 cores for QEMU using cgroups, FIO-2.1.10 with SPDK plugin, io depth=1, 8, 32 numjobs=1, direct=1, block size 4k

VM Density: Rate Limiting 20K IOPS per VM

Intel Xeon Platinum 8180 Processor, 24x Intel P4800x 375GB

10 vhost-scsi cores

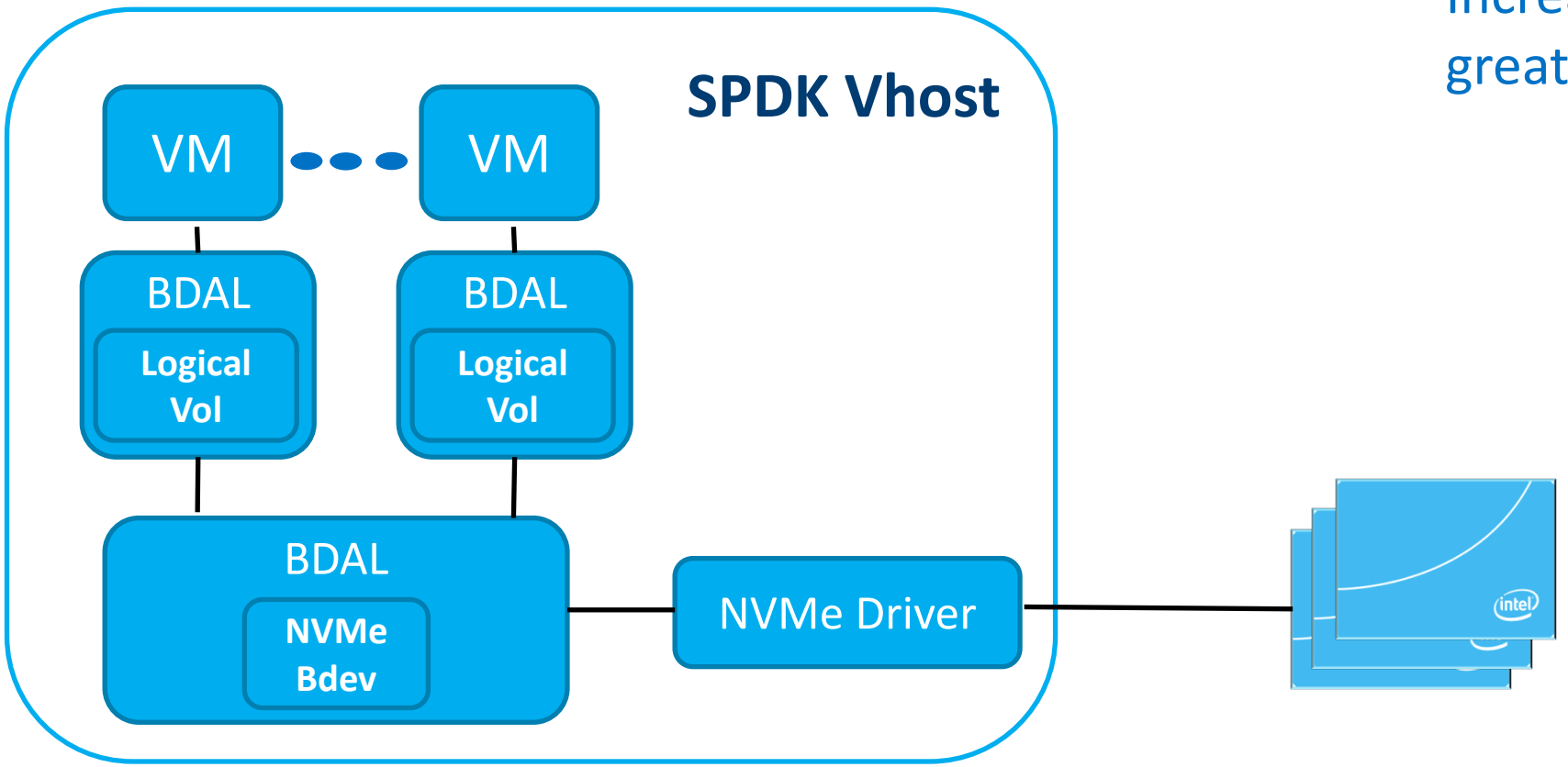


- % CPU utilized shown from VM side
- Each VM was running queue depth=1, 4KB random read workload
- Hyper threading enabled to allow 112 cores.
- Each VM rate limited to 20K IOPS using cgroups
- SPDK able to scale to 96 VMs, supporting 20K per VM. Kernel scale till 48 VMs. Beyond 48 VMs, 10 vhost-cores seem bottleneck

System Configuration:Intel Xeon Platinum 8180 @ 2.5GHz. 56 physical cores 6x 16GB, 2667 DDR4, 6 memory Channels, SSD: Intel P4800x 375GB x24 drives, Bios: HT disabled, p-states enabled, turbo enabled, Ubuntu 16.04.1 LTS, 4.11.0 x86_64 kernel, 48 VMs, number of partition: 2, VM config : 1core 1GB memory, VM OS: fedora 25, blk-mq enabled, Software packages: Qemu-2.9, libvirt-3.0.0, spdk (3bfecec994), IO distribution: 10 vhost-cores for SPDK / Kernel. Rest 46 cores for QEMU using cgroups, FIO-2.1.10 with SPDK plugin, io depth=1, 8, 32 numjobs=1, direct=1, block size 4k

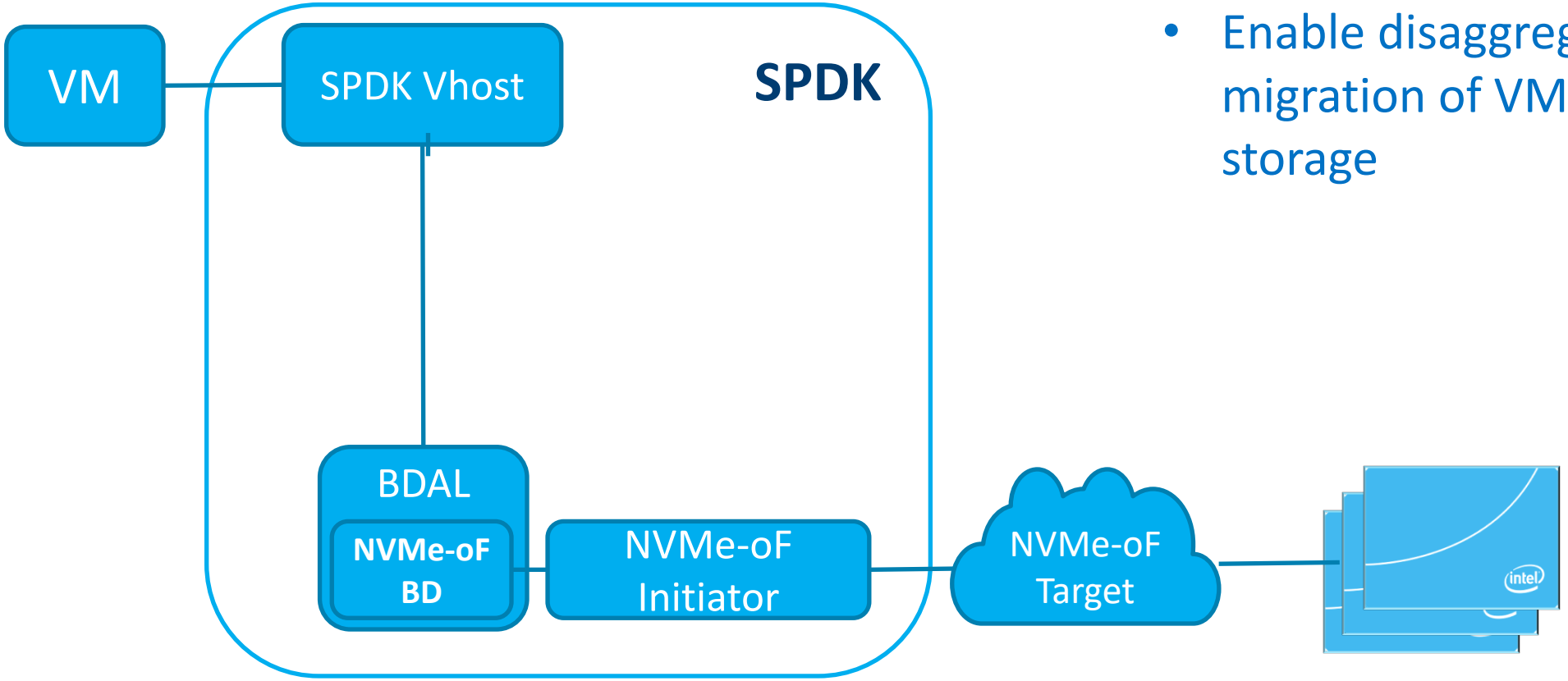
Use Cases for vhost

VM EPHEMERAL STORAGE



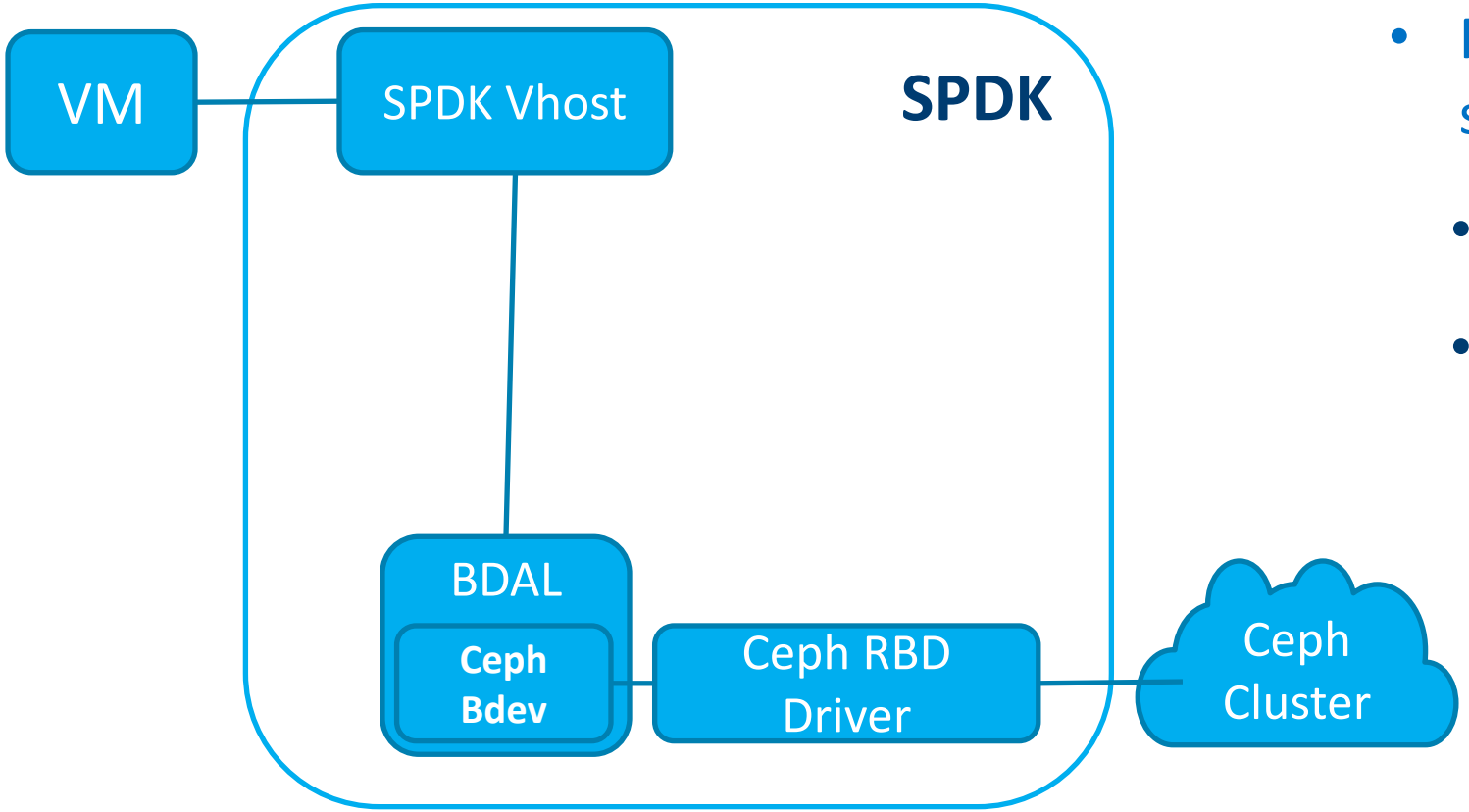
- Increased efficiency yields greater VM density

VM Remote Storage



- Enable disaggregation and migration of VMs using remote storage

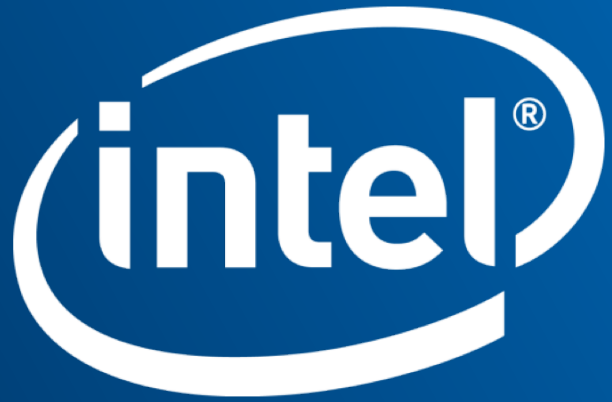
VM CEph Storage



- Potential for innovation in data services
  - Cache
  - Deduplication

For More information on SPDK

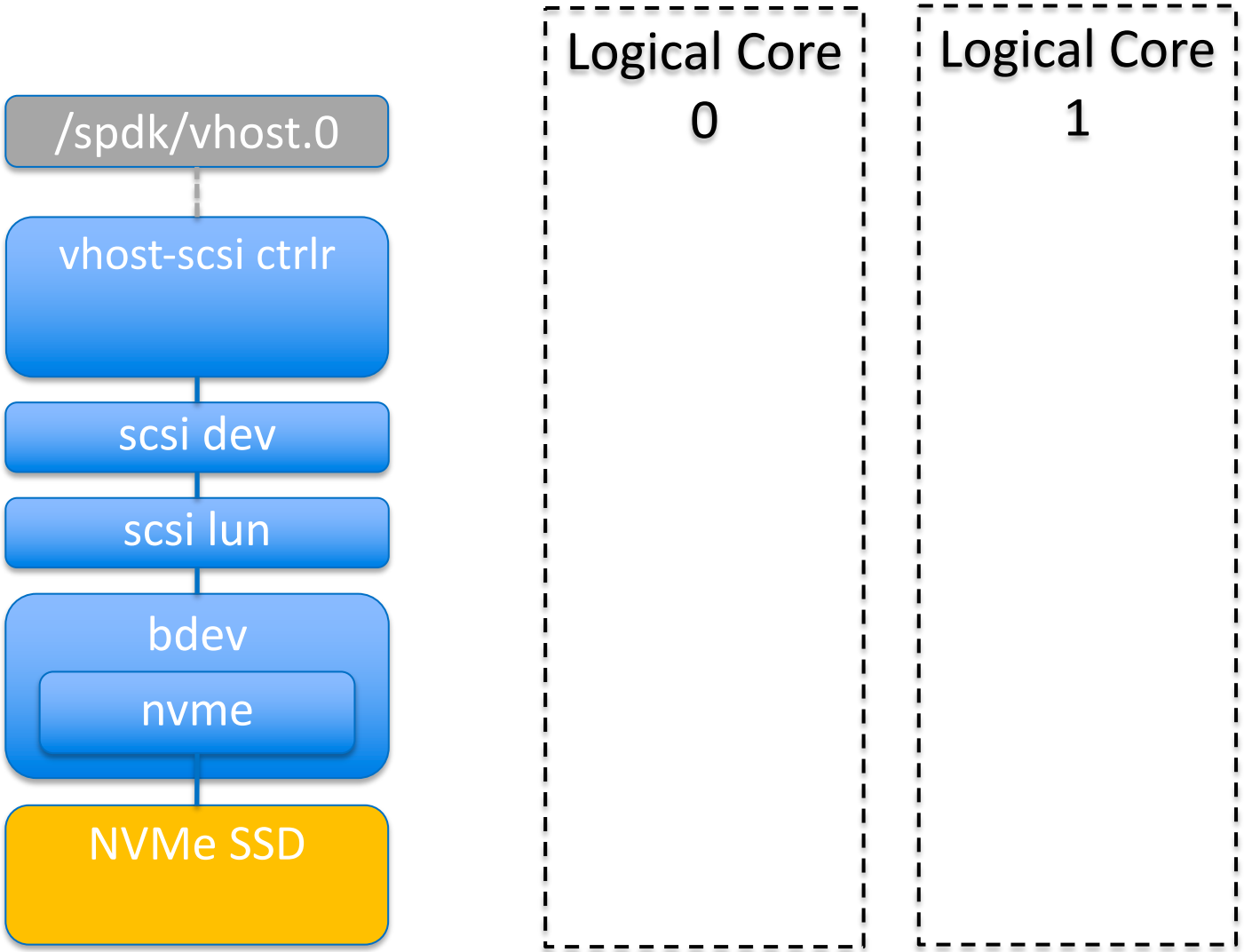
- Visit ***SPDK.io*** for tutorials and links to github, maillist, IRC channel and other resources
- Follow ***@SPDKProject on twitter*** for latest events, blogs and other SPDK community information and activities


intel®

Basic Architecture

Configure vhost-scsi controller

- JSON RPC
- creates SPDK constructs for vhost device and backing storage
- creates controller-specific vhost domain socket



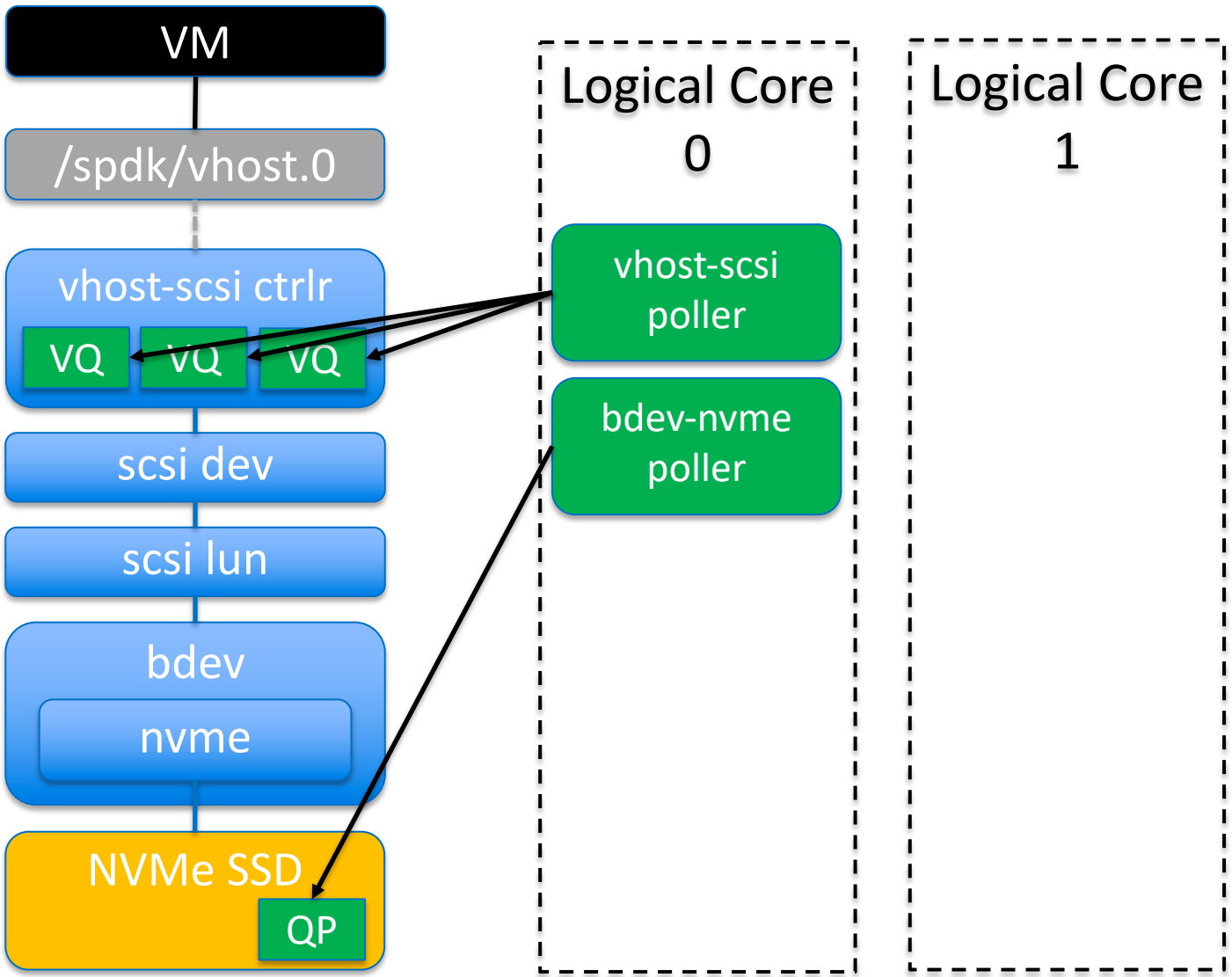
Basic Architecture

Launch VM

- QEMU connects to domain socket

SPDK

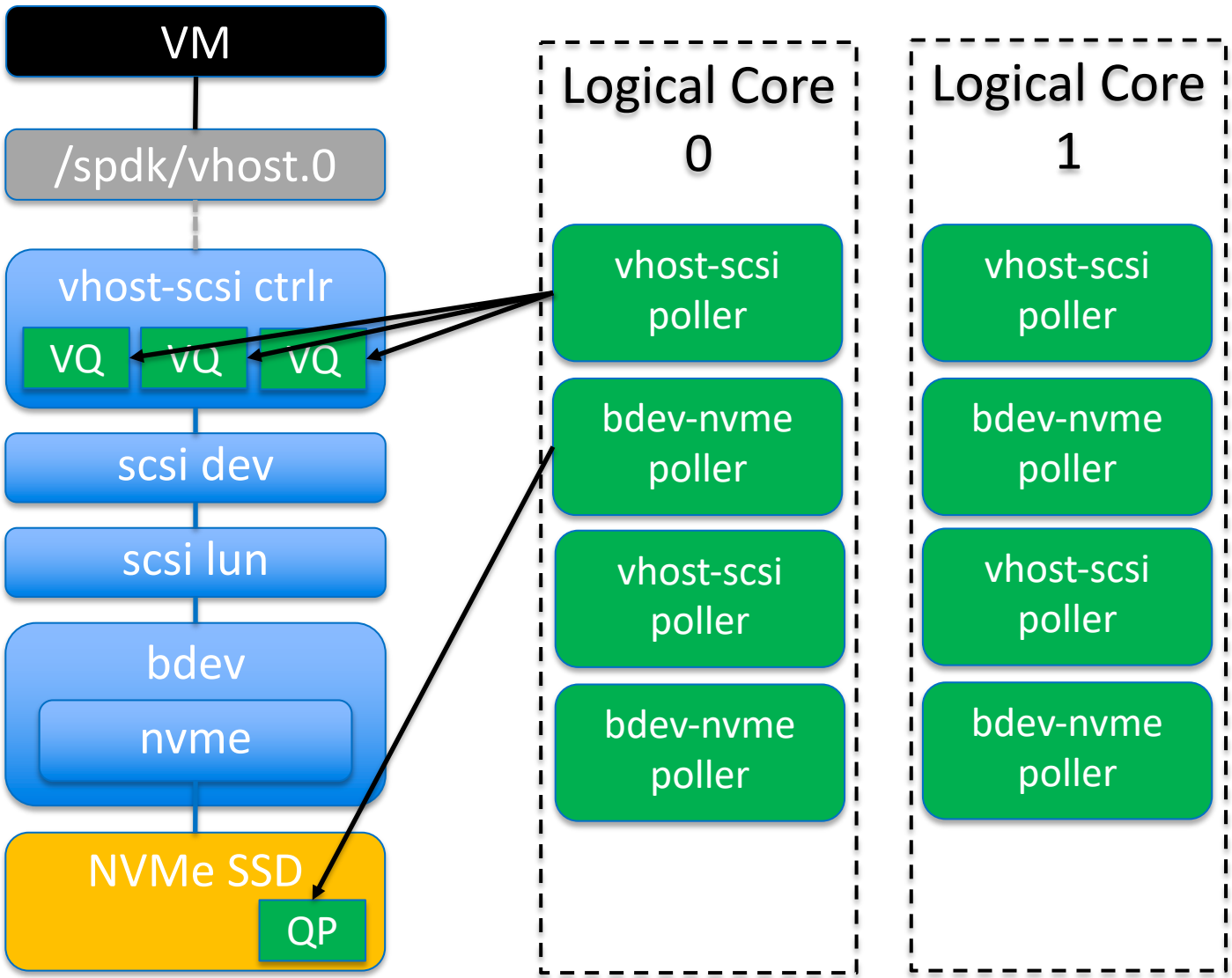
- Assigns logical core
- Starts vhost dev poller
- Allocates NVMe queue pair
- Starts NVMe poller



Basic Architecture

Repeat for additional VMs

- pollers spread across available cores



Logical volumes and blobstore

Blobstore Design – Design Goals

- Minimalistic for targeted storage use cases like Logical Volumes and RocksDB
- Deliver only the basics to enable another class of application
- Design for fast storage media

Blobstore Design – High Level

Application interacts with chunks of data called blobs

- Mutable array of pages of data, accessible via ID

Asynchronous

- No blocking, queuing or waiting

Fully parallel

- No locks in IO path

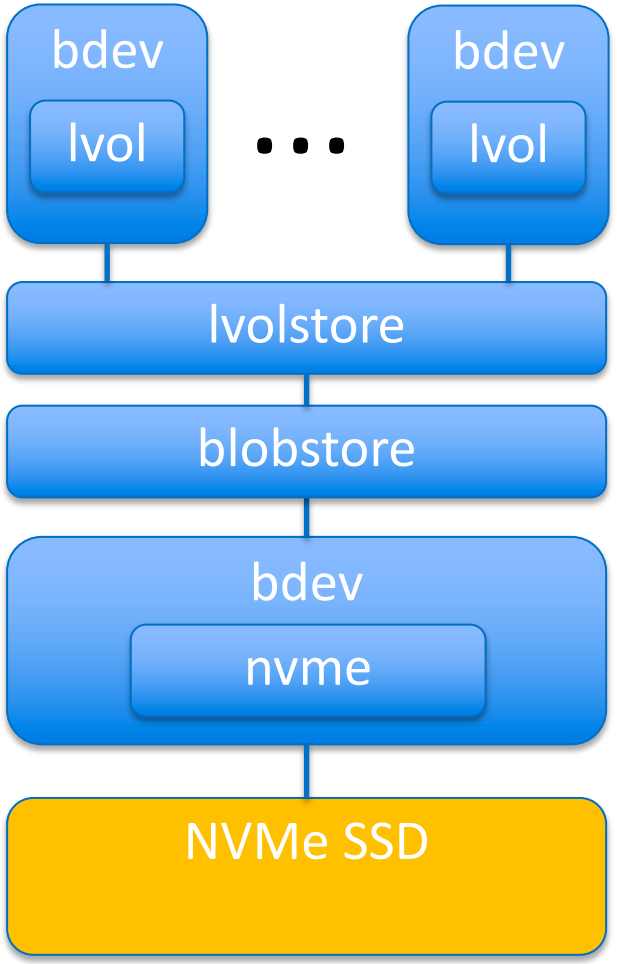
Atomic metadata operations

- Depends on SSD atomicity (i.e. NVMe)
- 1+ 4KB metadata pages per blob

Logical Volumes

Blobstore plus:

- UUID xattr for lvolstore, lvols
- Friendly names
  - lvol name unique within lvolstore
  - lvolstore name unique within application
- Future
  - snapshots (requires blobstore support)



Asynchronous Polling

Poller execution

- Reactor on each core
- Iterates through pollers round-robin
- vhost-scsi poller
  - poll for new I/O requests
  - submit to NVMe SSD
- bdev-nvme poller
  - poll for I/O completions
  - complete to guest VM