ACCELERATING NVME-OF TARGET SERVICE VIA SPDK/DPDK

Ziye Yang

NPG, DCG, Intel

Agenda

- What is SPDK?
- Accelerated NVMe-oF via SPDK
- Conclusion

Agenda

- What is SPDK?
- Accelerated NVMe-oF via SPDK
- Conclusion

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

Benefits of using **SPDK**

SPDK
more performance from Intel CPUs, non-volatile media, and networking

Up to **10X MORE** IOPS/core for NVMe-oF* vs. Linux kernel

Up to **8X MORE** IOPS/core for NVMe vs. Linux kernel

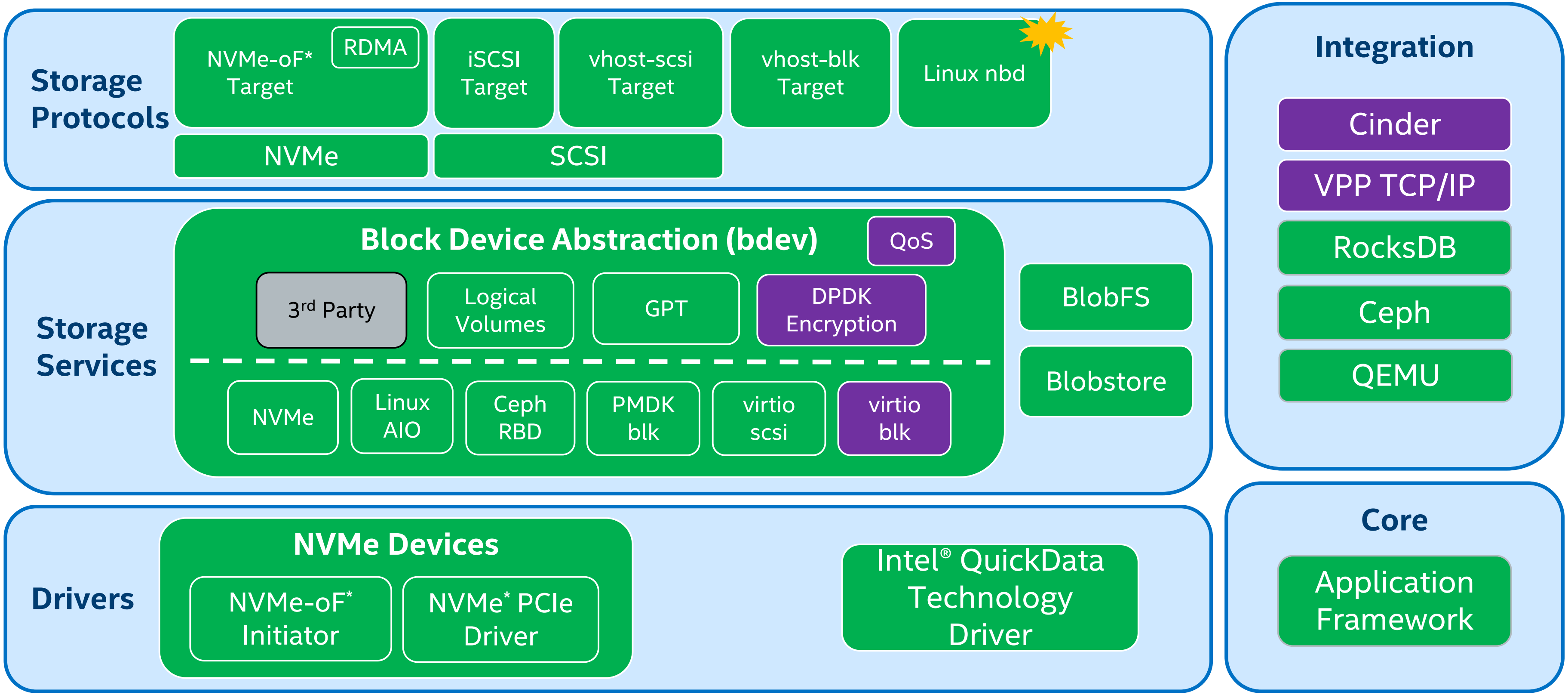
Up to **350%** **BETTER** Tail Latency for RocksDB workloads

FASTER TTM/ **LESS** RESOURCES than developing components from scratch

Provides Future Proofing as NVM technologies increase in performance

ARCHITECTURE

- Released
- New release 18.01
- 1H‘18



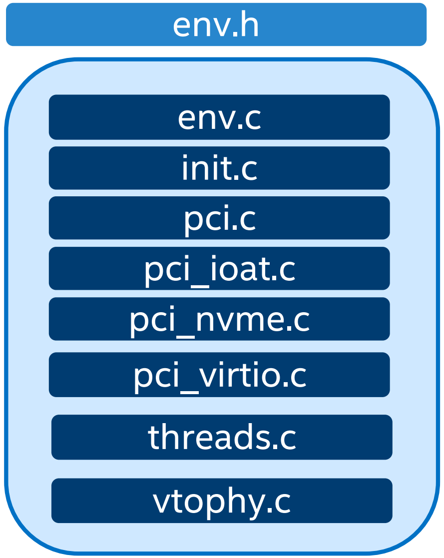
SPDK ENVIRONMENT ABSTRACTION

WHY AN ENVIRONMENT ABSTRACTION?

FLEXIBILITY FOR USER

ENVIRONMENT ABSTRACTION

- Memory allocation (pinned for DMA) and address translation
- PCI enumeration and resource mapping
- Thread startup (pinned to cores)
- Lock-free ring and memory pool data structures



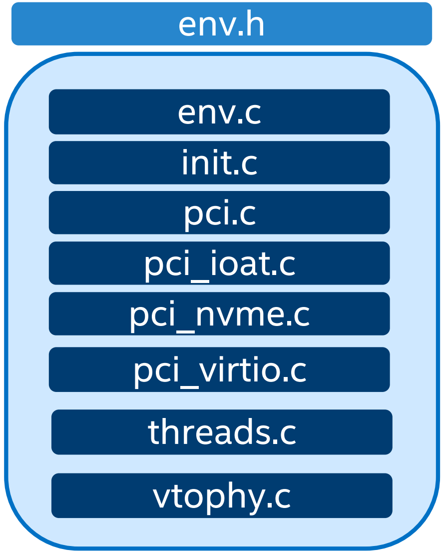
ENVIRONMENT ABSTRACTION

Configurable:

```
./configure --with-env=...
```

Interface defined in `spdk/env.h`

Default implementation uses **DPDK** (`lib/env_dpdk`)



FLEXIBILITY: DECOUPLING AND DPDK ENHANCEMENTS

APPLICATION FRAMEWORK

HOW DO WE COMBINE SPDK COMPONENTS?

THE SPDK APP FRAMEWORK PROVIDES THE GLUE

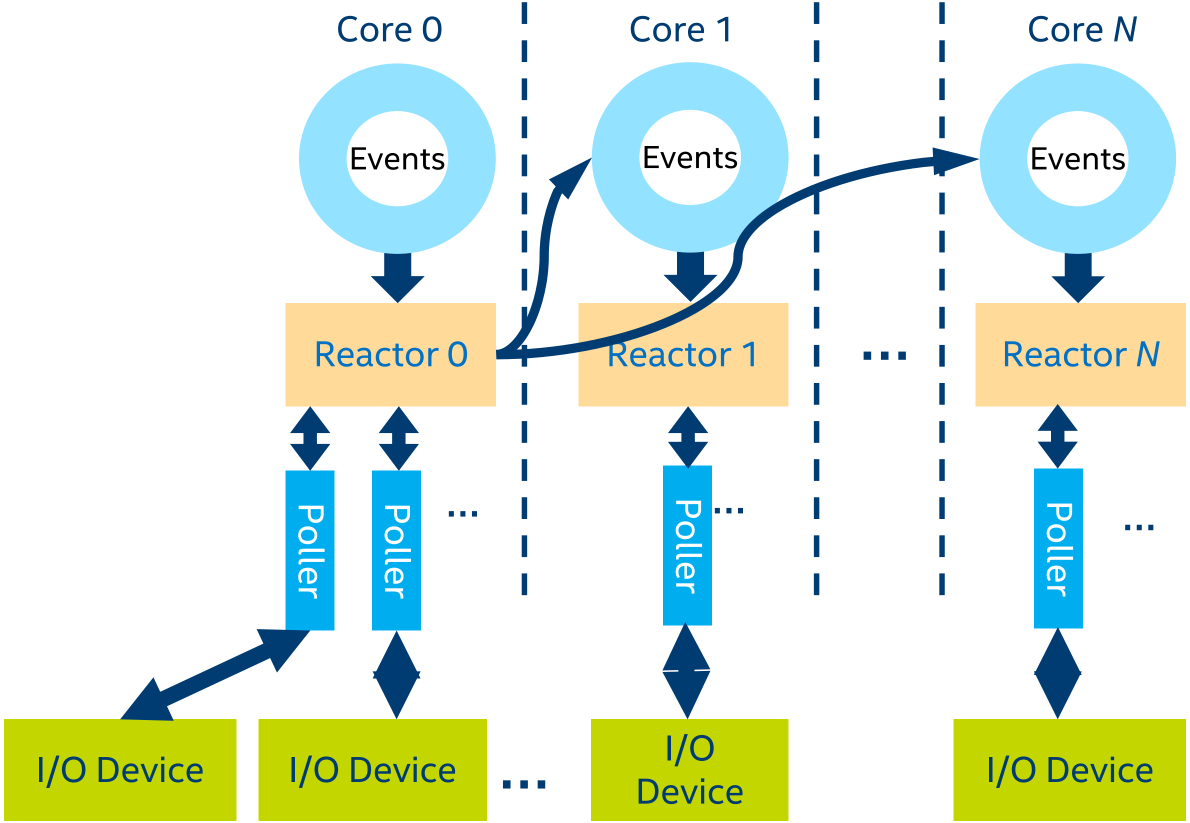
APP FRAMEWORK COMPONENTS

REACTOR

POLLER

EVENT

I/O CHANNEL


Core 0
Core 1
Core N
Events
Events
Events
Reactor 0
Reactor 1
…
Reactor N
Poller
Poller
…
Poller
…
Poller
…
I/O Device
I/O Device
…
I/O Device
I/O Device

POLLER

Essentially a “task” running on a reactor

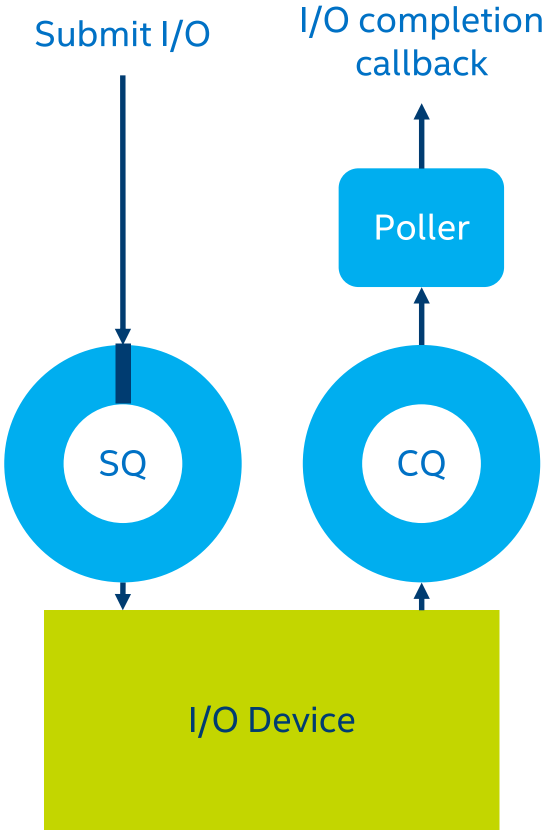
Primarily checks hardware for async events

Can run periodically on a timer

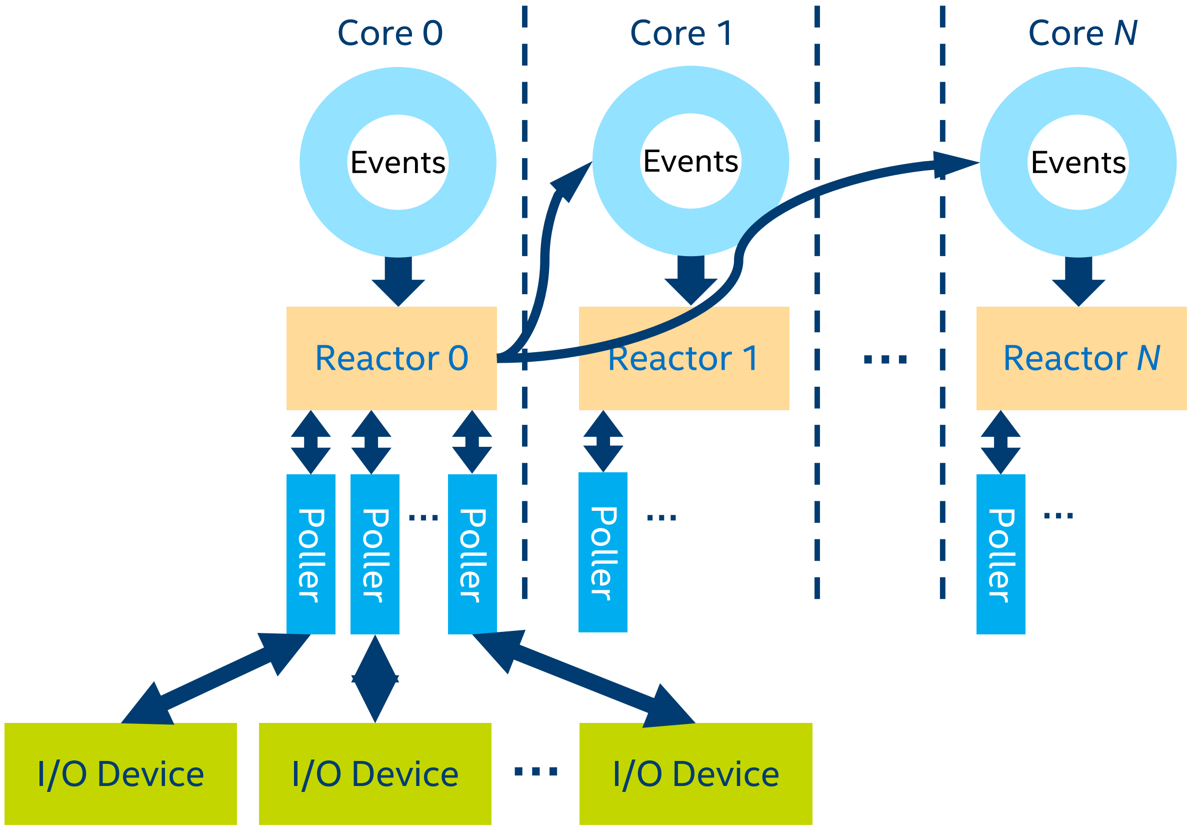
Example: poll completion queue

Callback runs to completion on reactor thread

Completion handler may send an event



EVENT



Core 0
Core 1
Core N
Events
Events
Events
Reactor 0
Reactor 1
…
Reactor N
Poller
Poller
…
Poller
Poller
…
Poller
…
I/O Device
I/O Device
…
I/O Device

EVENT

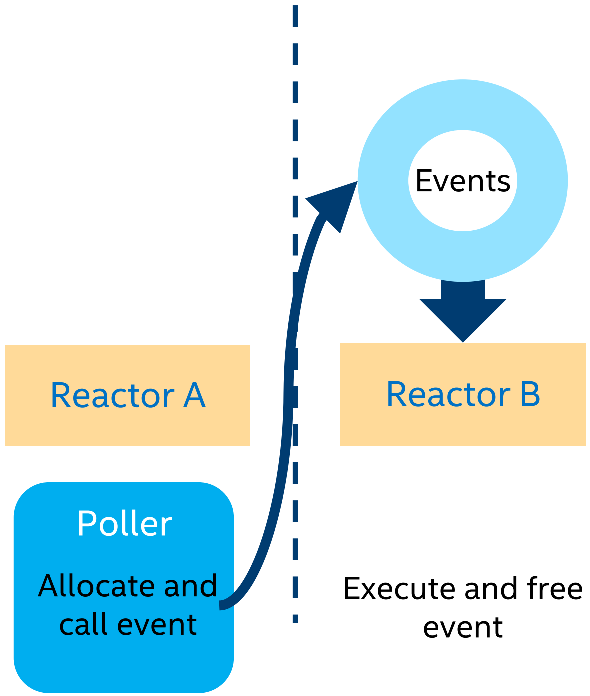
Cross-thread communication

Function pointer + arguments

One-shot message passed between reactors

Multi-producer/single-consumer ring

Runs to completion on reactor thread



I/O CHANNEL

Abstracts hardware I/O queues

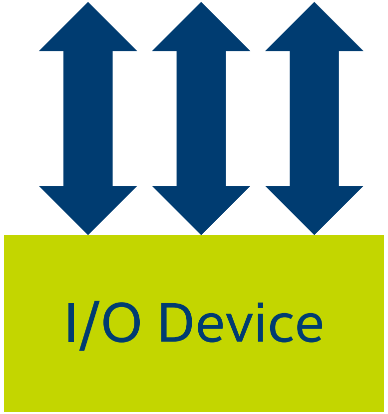
Register I/O devices

Create I/O channel per thread/device combination

Provides hooks for driver resource allocation

I/O channel creation drives poller creation

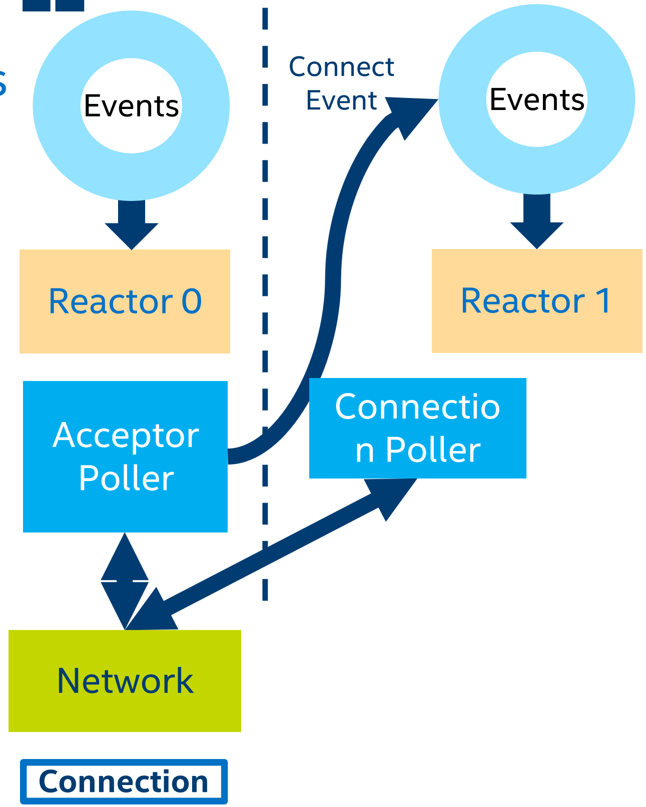
Pervasive in SPDK



NVME OVER FABRICS TARGET EXAMPLE

Acceptor network poller handles connect events

Connection event registers new poller



NVME OVER FABRICS TARGET EXAMPLE

Acceptor network poller handles connect events

Connection event registers new poller

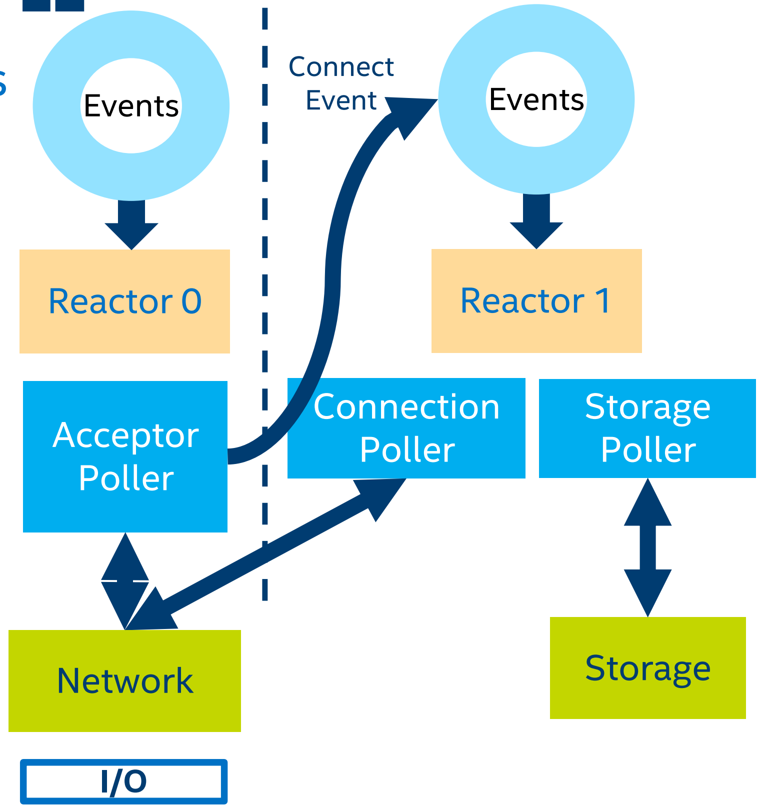
I/O request arrives over network

I/O submitted to storage

Storage device poller checks completions

Response sent

ALL ASYNCHRONOUS WORK IS DRIVEN BY POLLERS



Agenda

- What is SPDK?
- Accelerated NVMe-oF via SPDK
- Conclusion

SPDK NVMe-oF Components

NVMe over Fabrics Target

- Released July 2016 (with spec)
- **Hardening:**
  - Intel test infrastructure
    Discovery simplification
    Correctness & kernel interop
- **Performance improvements:**
  - Read latency improvement
    Scalability validation (up to 150Gbps)
  - Event Framework enhancements
  - Multiple connection performance improvement

NVMe over Fabrics Host (Initiator)

- New component added in Dec 2016
- Performance improvements
  - Eliminate copy: now true zero-copy
  - SGL (single SGL element)

SPDK NVMe-oF transport work

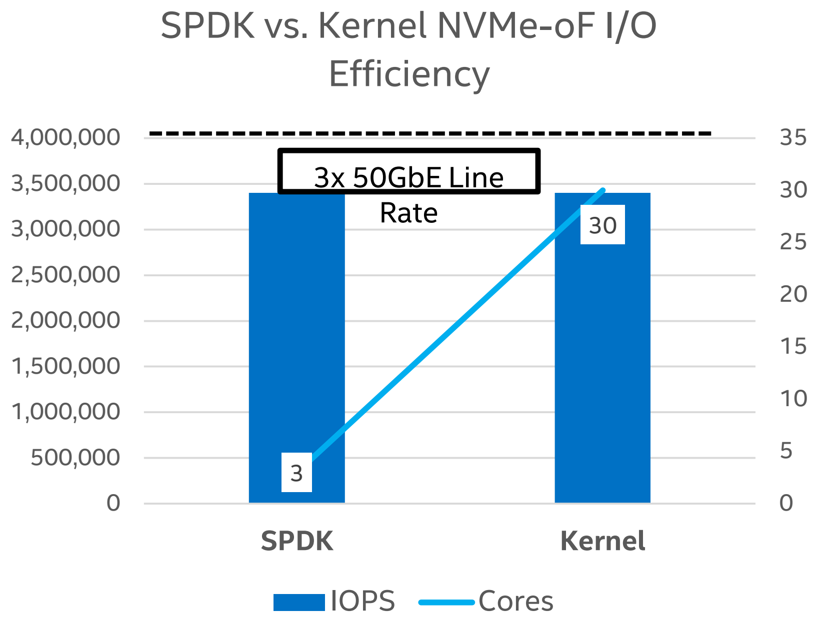
Existing work: RDMA transport

- **DPDK components used which is encapsulated in libspdk_env_dpdk.a, e.g.,**
  - PCI device management
  - CPU/thread scheduling
  - Memory management (e.g., lock free rings)
  - Log management

Upcoming work: TCP transport

- Kernel based TCP transport
- VPP/DPDK based user space TCP transport
  - Use DPDK Ethernet PMDs
  - Use user space TCP/IP stack (e.g., VPP)

NVMe-oF Target Throughput Performance (RDMA)

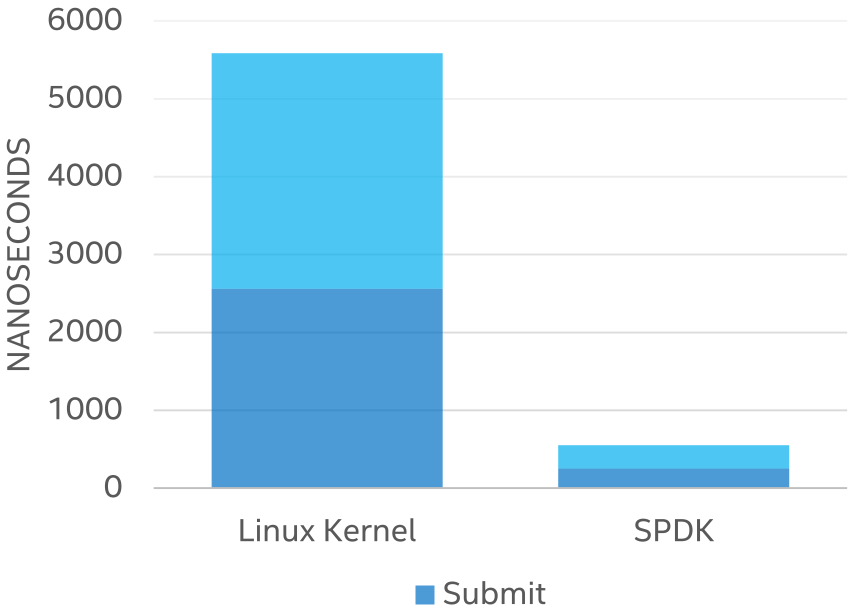


NVMe* over Fabrics Target Features	Realized Benefit
Utilizes NVM Express* (NVMe) Polled Mode Driver	Reduced overhead per NVMe I/O
RDMA Queue Pair Polling	No interrupt overhead
Connections pinned to CPU cores	No synchronization overhead

SPDK reduces NVMe over Fabrics software overhead up to 10x!

System Configuration: Target system: Supermicro SYS-2028U-TN24R4T+, 2x Intel® Xeon® E5-2699v4 (HT off), Intel® Speed Step enabled, Intel® Turbo Boost Technology enabled, 8x 8GB DDR4 2133 MT/s, 1 DIMM per channel, 12x Intel® P3700 NVMe SSD (800GB) per socket, -1H0 FW; Network: Mellanox* ConnectX-4 LX 2x25Gb RDMA, direct connection between initiators and target; Initiator OS: CentOS* Linux* 7.2, Linux kernel 4.10.0, Target OS (SPDK): Fedora 25, Linux kernel 4.9.11, Target OS (Linux kernel): Fedora 25, Linux kernel 4.9.11 Performance as measured by: fio, 4KB Random Read I/O, 2 RDMA QP per remote SSD, Numjobs=4 per SSD, Queue Depth: 32/job. SPDK commit ID: 4163626c5c

NVM Express* Driver Software Overhead

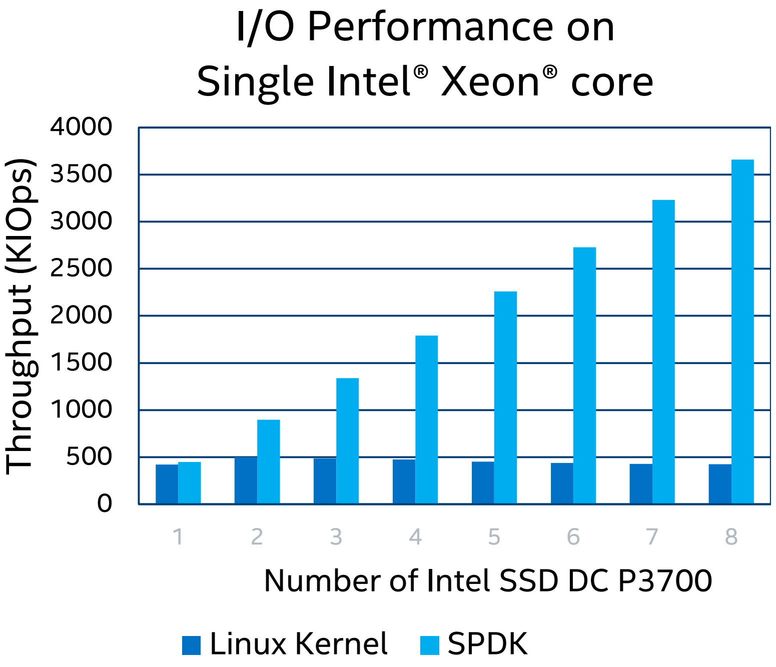


Kernel Source of Overhead	SPDK Approach
Interrupts	Asynchronous Polled Mode
Synchronization	Lockless
System Calls	User Space Hardware Access
DMA Mapping	Hugepages
Generic Block Layer	Specific for Flash Latencies

SPDK reduces NVM Express* (NVMe) software overhead up to 10x!

System Configuration: 2x Intel® Xeon® E5-2695v4 (HT off), Intel® Speed Step enabled, Intel® Turbo Boost Technology disabled, 8x 8GB DDR4 2133 MT/s, 1 DIMM per channel, CentOS* Linux* 7.2, Linux kernel 4.7.0-rc1, 1x Intel® P3700 NVMe SSD (800GB), 4x per CPU socket, FW 8DV10102, I/O workload 4KB random read, Queue Depth: 1 per SSD, Performance measured by Intel using SPDK overhead tool, Linux kernel data using Linux AIO

NVM Express* Driver Throughput Scalability

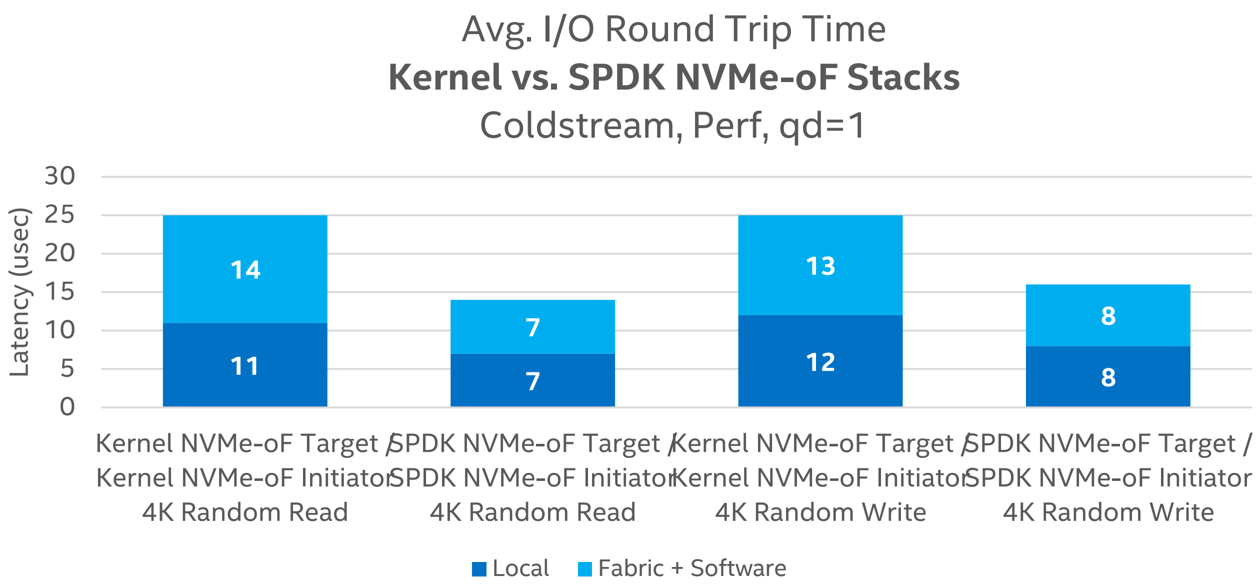


- Systems with multiple NVM Express* (NVMe) SSDs capable of millions of I/O per second
- Results in many cores of software overhead with kernel-based interrupt-driven driver model
- SPDK enables:
  - more CPU cycles for storage services
  - lower I/O latency

SPDK saturates 8 NVMe SSDs with a single CPU core!

System Configuration: 2x Intel® Xeon® E5-2695v4 (HT off), Intel® Speed Step enabled, Intel® Turbo Boost Technology disabled, 8x 8GB DDR4 2133 MT/s, 1 DIMM per channel, CentOS* Linux* 7.2, Linux kernel 4.10.0, 8x Intel® P3700 NVMe SSD (800GB), 4x per CPU socket, FW 8DV101H0, I/O workload 4KB random read, Queue Depth: 128 per SSD, Performance measured by Intel using SPDK perf tool, Linux kernel data using Linux AIO

SPDK Host + Target vs. Kernel Host + Target

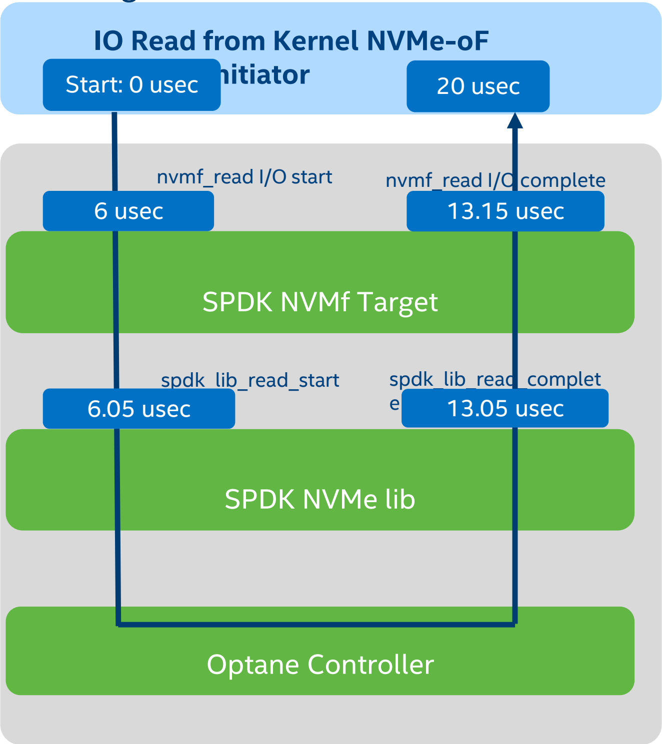


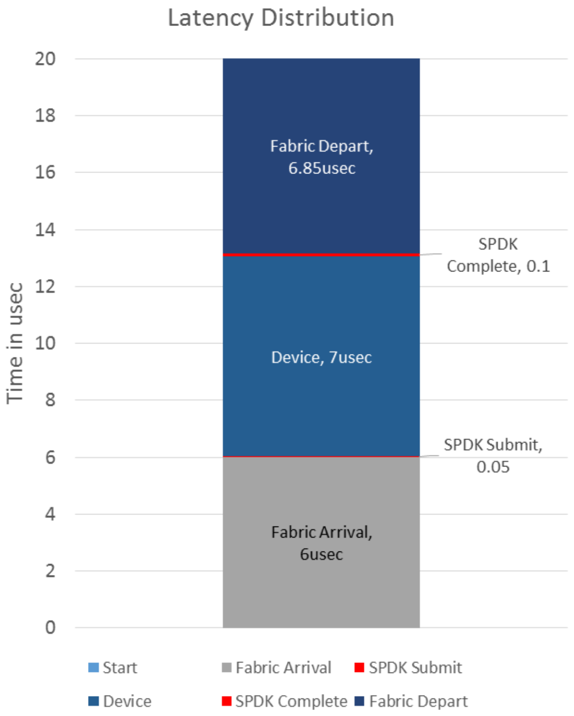
SPDK reduces Optane NVMe-oF latency by 44%, write latency by 36%!

System Configuration: 2x Intel® Xeon® E5-2695v4 (HT on, Intel® Speed Step enabled, Intel® Turbo Boost Technology enabled, 64GB DDR4 Memory, 8x 8GB DDR4 2400 MT/s, Ubuntu 16.04.1, Linux kernel 4.10.1, 1x 25GbE Mellanox 2P CX-4, CX-4 FW= 14.16.1020, mlx5_core= 3.0-1 driver, 1 ColdStream, connected to socket 0, 4KB Random Read I/0 1 initiators, each initiator connected to bx NVMe-oF subsystems using 2P 25GbE Mellanox. Performance measured by Intel using SPDK perf tool, 4KB Random Read I/O, Queue Depth: 1/NVMe-oF subsystem. numjobs 1, 300 sec runtime, direct=1, norandommap=1, FIO-2.12, SPDK commit # 42eade49

NVMe-oF IO Latency Model, 4KB Random Read (Intel Optane SSD DC P4800X)

SPDK Target + Kernel NVMe-oF Initiator



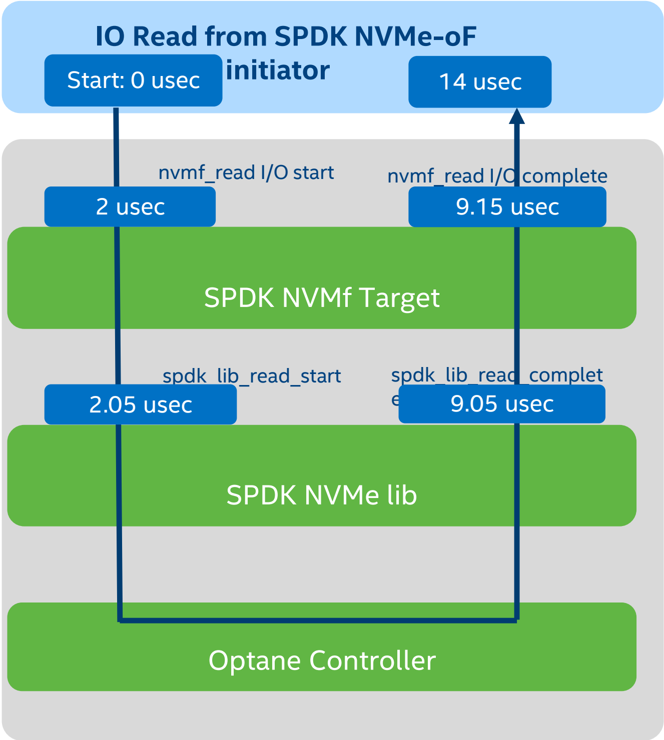


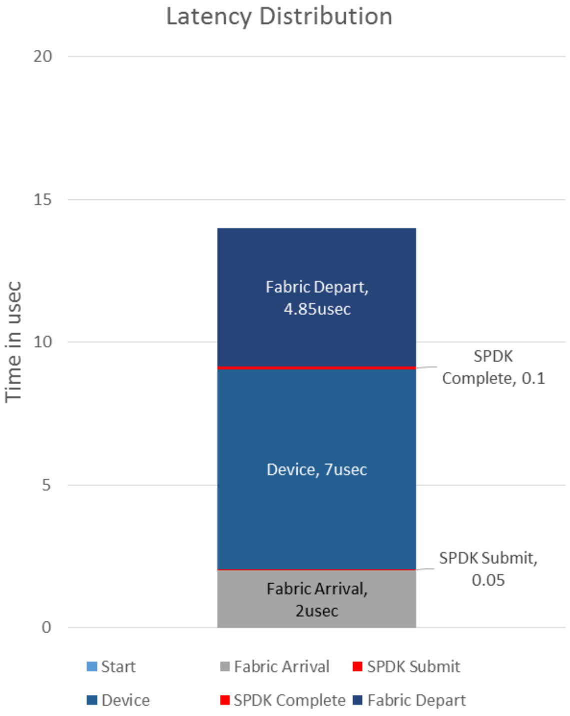
- 20 usec round trip time measured from NVMe-oF initiator
- Out of 20usec, ~7 usec spent in NVMe controller
- 12-13 usec measured time in the fabric and kernel NVMe-oF initiator
- SPDK NVMf target adds just 100-200 nsec to fabric overhead

System Configuration: 2x Intel® Xeon® E5-2695v4 (HT on, Intel® Speed Step enabled, Intel® Turbo Boost Technology enabled, 64GB DDR4 Memory, 8x 8GB DDR4 2400 MT/s, Ubuntu 16.04.1, Linux kernel 4.10.1, 1x 25GbE Mellanox 2P CX-4, CX-4 FW= 14.16.1020, mlx5_core= 3.0-1 driver, 1 ColdStream, connected to socket 0, 4KB Random Read I/0 1 initiators, each initiator connected to bx NVMe-oF subsystems using 2P 25GbE Mellanox. Performance measured by Intel using SPDK perf tool, 4KB Random Read I/O, Queue Depth: 1/NVMe-oF subsystem. numjobs 1, 300 sec runtime, direct=1, norandommap=1, FIO-2.12, SPDK commit # 42eade49

NVMe-oF IO Latency Model, 4KB Random Read (Intel Optane SSD DC P4800X)

SPDK Target + SPDK NVMe-oF Initiator





- 14 usec round trip time measured from NVMf client
- Out of 14usec, ~7 usec spent in NVMe controller
- 7 usec measured time in the fabric and SPDK NVMe-oF initiator
- SPDK NVMf target adds just 100-200 nsec to fabric overhead

System Configuration: 2x Intel® Xeon® E5-2695v4 (HT on, Intel® Speed Step enabled, Intel® Turbo Boost Technology enabled, 64GB DDR4 Memory, 8x 8GB DDR4 2400 MT/s, Ubuntu 16.04.1, Linux kernel 4.10.1, 1x 25GbE Mellanox 2P CX-4, CX-4 FW= 14.16.1020, mlx5_core= 3.0-1 driver, 1 ColdStream, connected to socket 0, 4KB Random Read I/0 1 initiators, each initiator connected to bx NVMe-oF subsystems using 2P 25GbE Mellanox. Performance measured by Intel using SPDK perf tool, 4KB Random Read I/O, Queue Depth: 1/NVMe-oF subsystem. numjobs 1, 300 sec runtime, direct=1, norandommap=1, FIO-2.12, SPDK commit #42eade49

Agenda

- What is SPDK?
- Accelerated NVMe-oF via SPDK
- Conclusion

Conclusion

- In this presentation, we introduce
  - SPDK library
  - The accelerated NVMe-oF target built from SPDK library
- SPDK proves to be useful to accelerate storage applications equipped with NVMe based devices
- Call for action:
  - Welcome to use SPDK in storage area (similar as using DPDK in network) and contribute into SPDK community.

Q&A

intel®

experience what's inside™