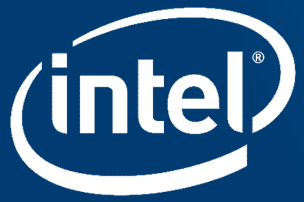


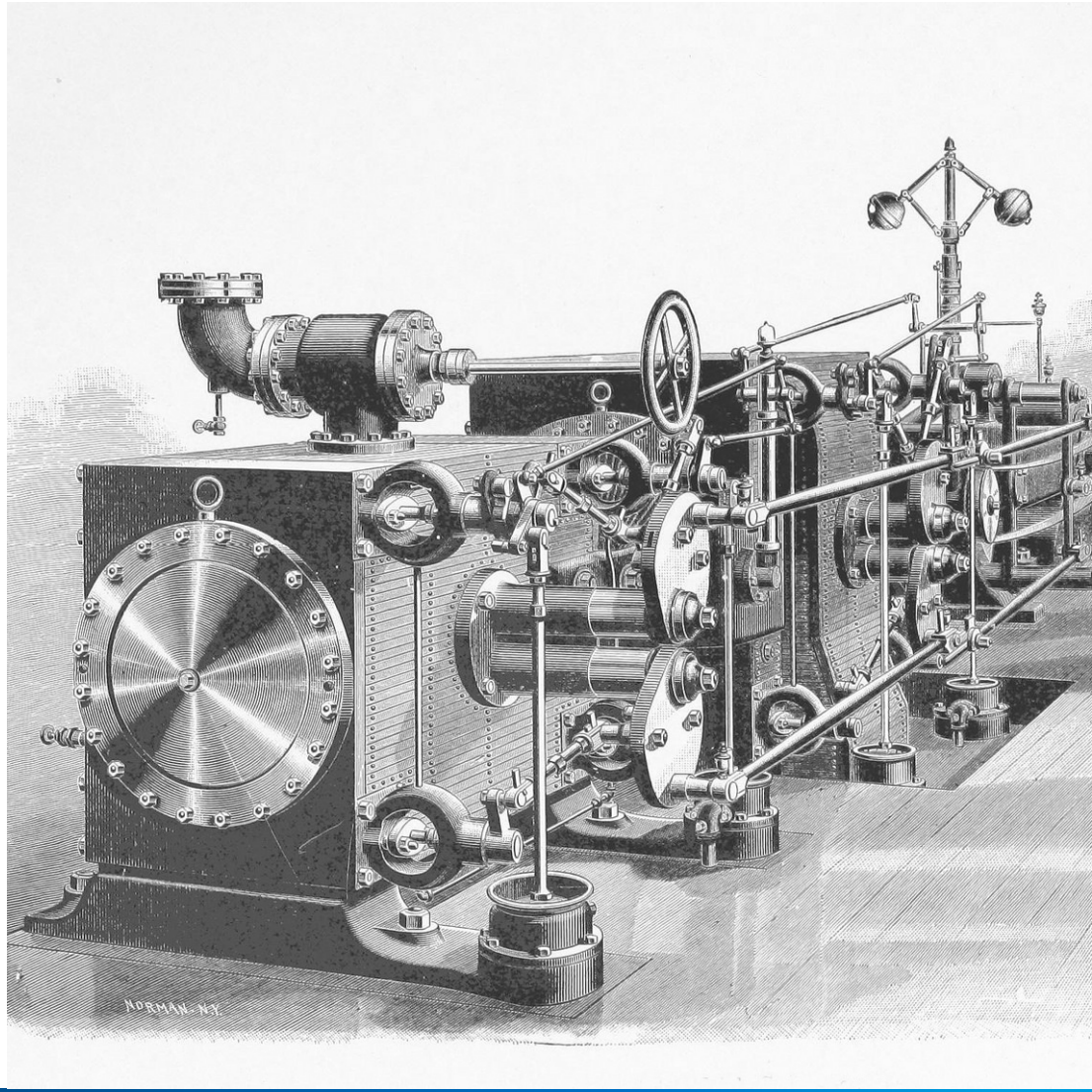
SPDK: UNDER THE HOOD

Storage Performance Development Kit (SPDK)

Daniel Verkamp, Software Engineer

AGENDA

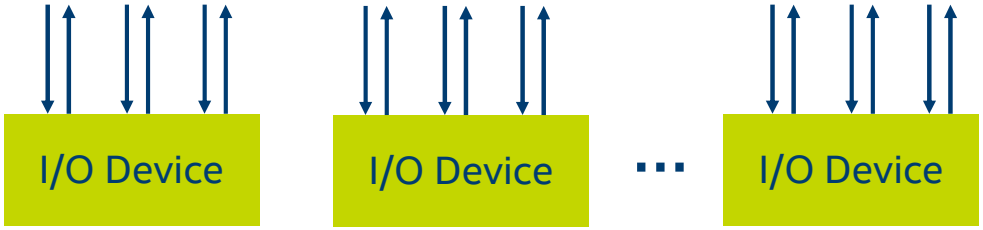
- THREADING MODEL DISCUSSION
- SPDK ENVIRONMENT LAYER
- SPDK APPLICATION FRAMEWORK
- SPDK BLOCKDEV LAYER
- SPDK EXAMPLE APPS



MOTIVATION: PERFORMANCE VIA CONCURRENCY

Modern CPUs provide many cores

Modern I/O devices provide many independent queues



Goal: Architect software to match the hardware

CONTEXT SWITCHING AND INTERRUPTS

OS-provided multitasking was important on single-core machines

Modern machines have many cores

Instead of context switching, dedicate core(s) to specific tasks

Avoid interrupt handler overhead and latency by polling

Instead of locks, pass messages

THREADING MODEL OPTIONS

Model	Pros	Cons	Example
One connection per thread with blocking I/O	Simple programming model	Interrupt driven High memory overhead	Apache worker MPM
Many connections per thread with I/O event multiplexing (select(), ...)	Low memory overhead Less context switching	Interrupt driven Inefficient polling	Apache event MPM, nginx, libuv, ...
Many connections per thread with polled asynchronous I/O	Low memory overhead No interrupts No context switching	More complex programming model	SPDK

WHAT THREADING MODEL DOES SPDK TARGET?

ASYNCHRONOUS POLLED I/O

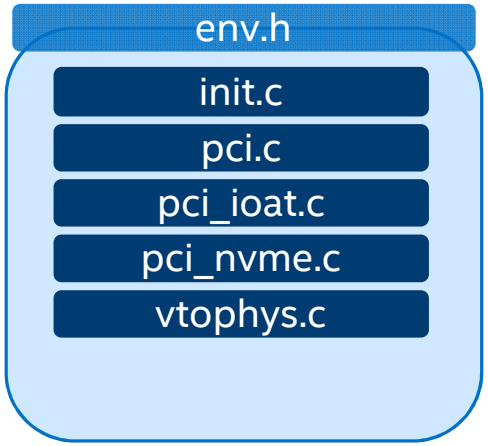
SPDK ENVIRONMENT ABSTRACTION

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



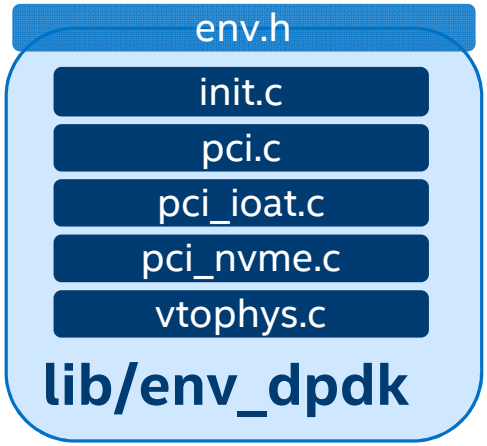
ENVIRONMENT ABSTRACTION

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?

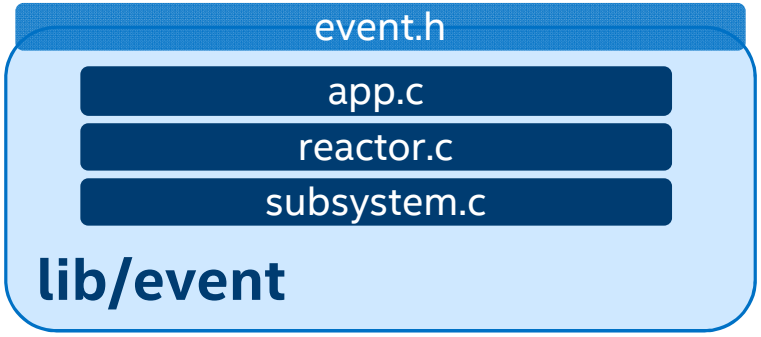
THE SPDK APP FRAMEWORK PROVIDES THE GLUE

APPLICATION FRAMEWORK

Builds on the environment abstraction

Example of how to glue other SPDK components together

Libraries (lib/*) vs. applications (app/*)



APP FRAMEWORK COMPONENTS

REACTOR

POLLER

EVENT

I/O CHANNEL

REACTOR

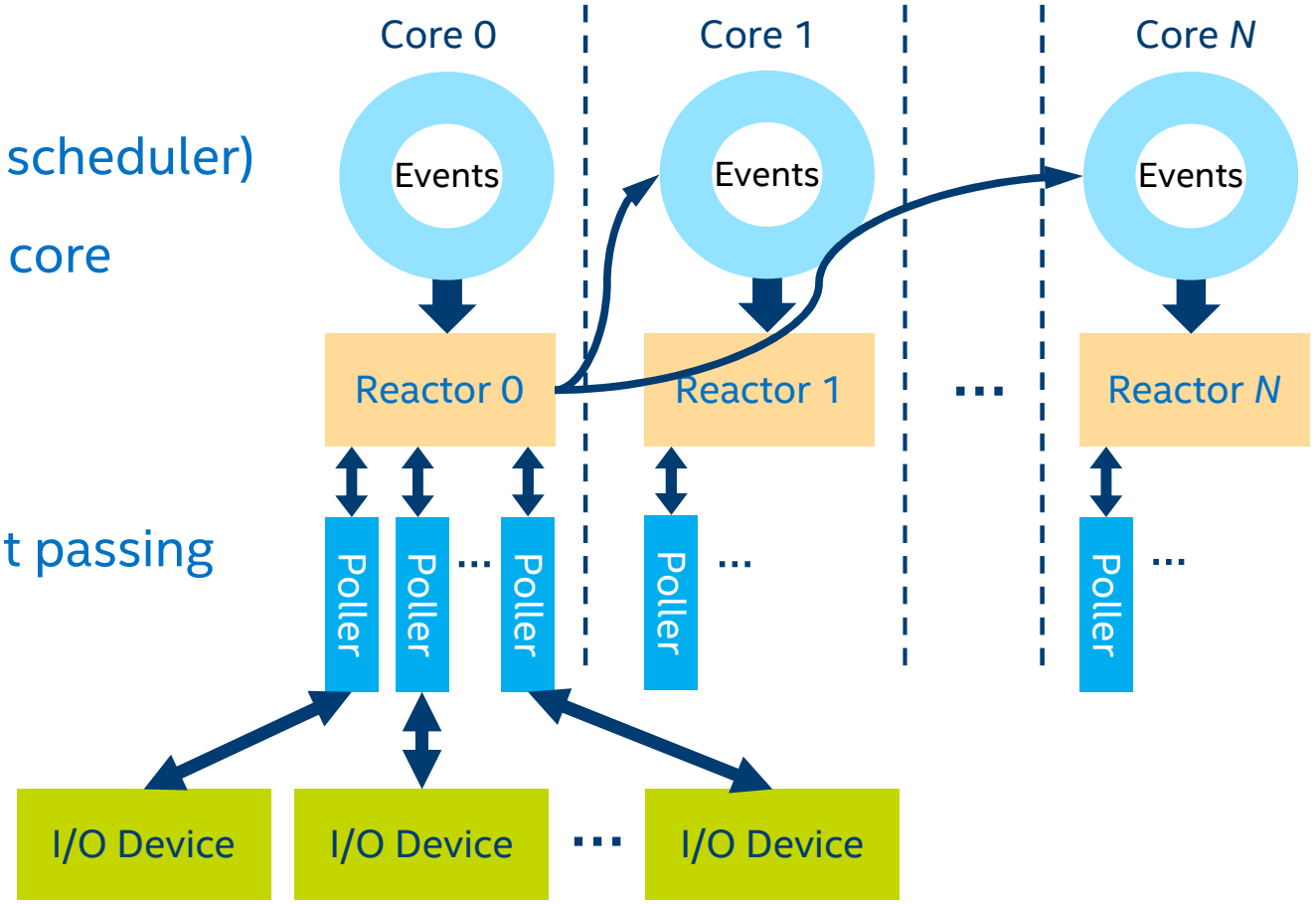
Event loop (essentially a scheduler)

Pinned to a specific CPU core

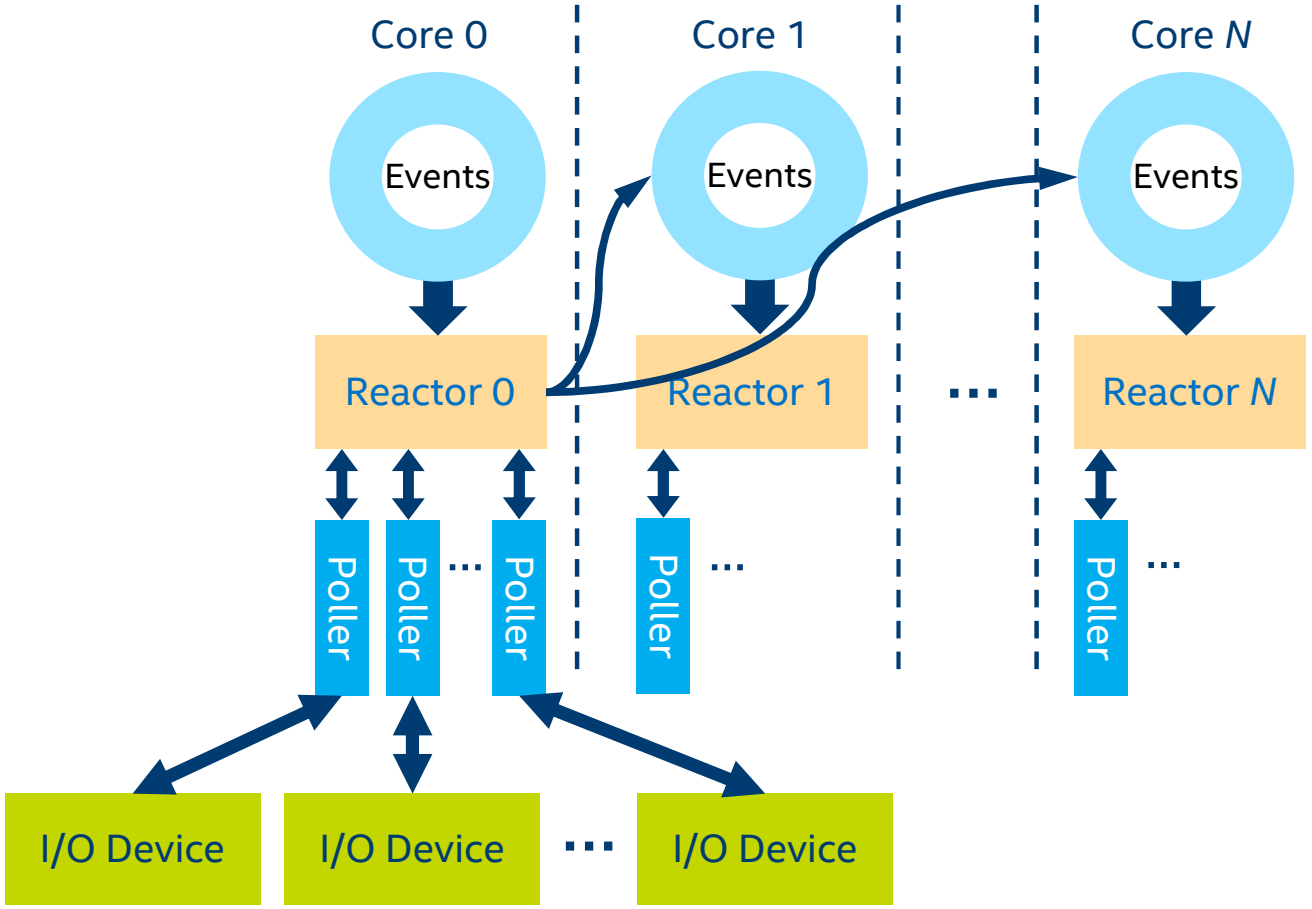
Polls I/O devices

Polls event ring

Communication via event passing



POLLER


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

POLLER

Essentially a “task” running on a reactor

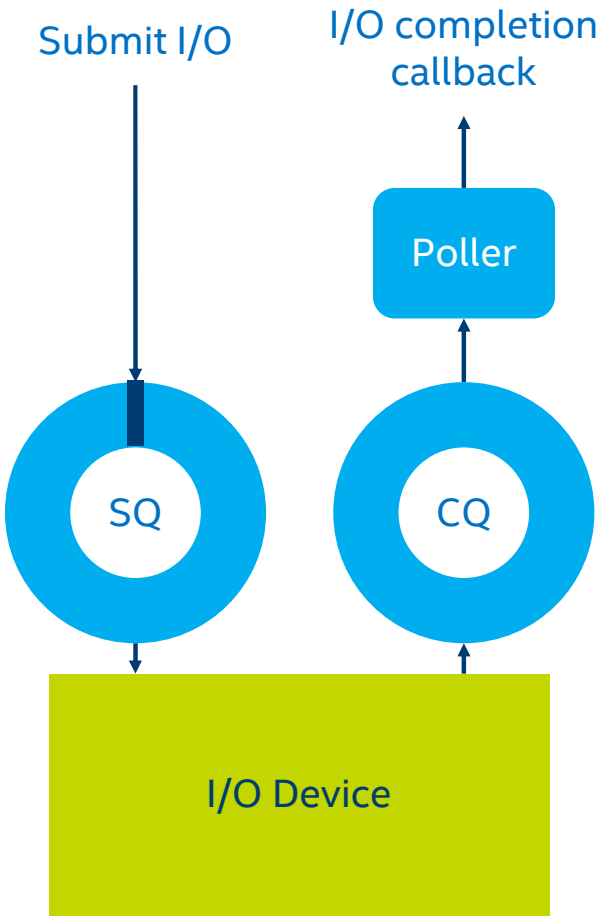
Primarily checks hardware for async events

Can run periodically on a timer

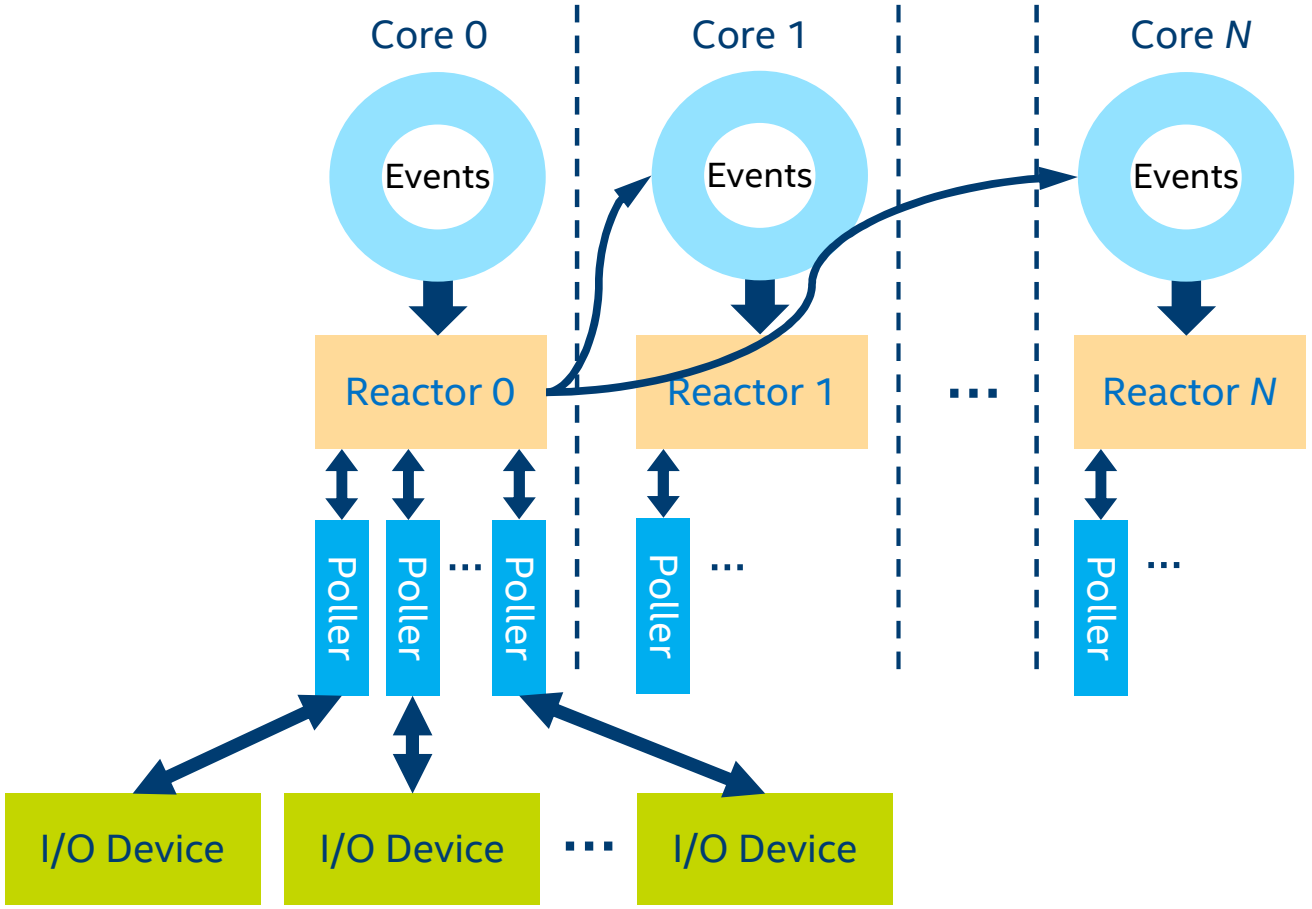
Example: poll completion queue

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

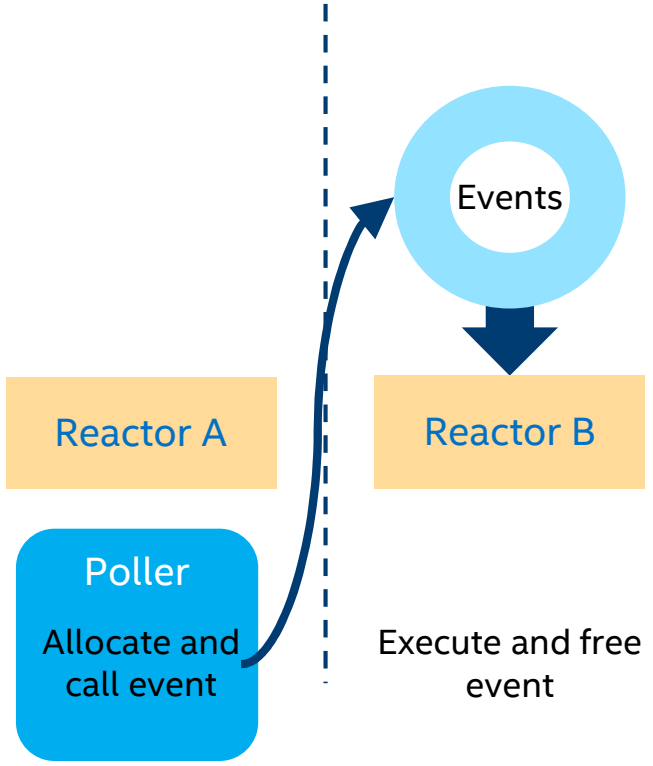
Cross-thread communication

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

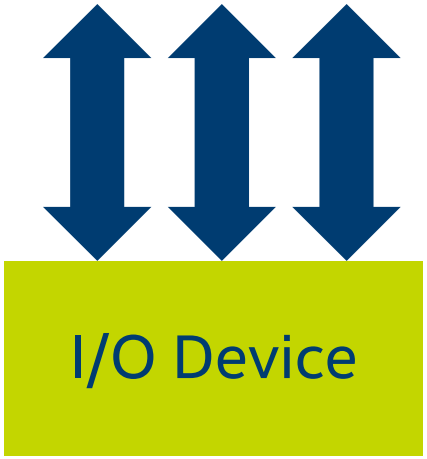
Register I/O devices

Create I/O channel per thread/device combination

Provides hooks for driver resource allocation

I/O channel creation drives poller creation

Pervasive in SPDK



BLOCKDEV LAYER

BLOCK DEVICE LAYER

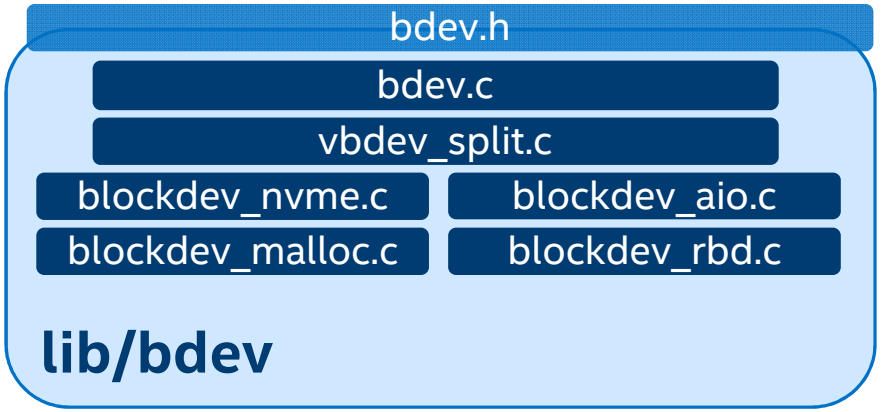
Block device driver abstraction

Async read, write, flush, deallocate

SGL support (readv/writev)

I/O channel integration

Layering (virtual blockdevs)



BDEV DRIVERS

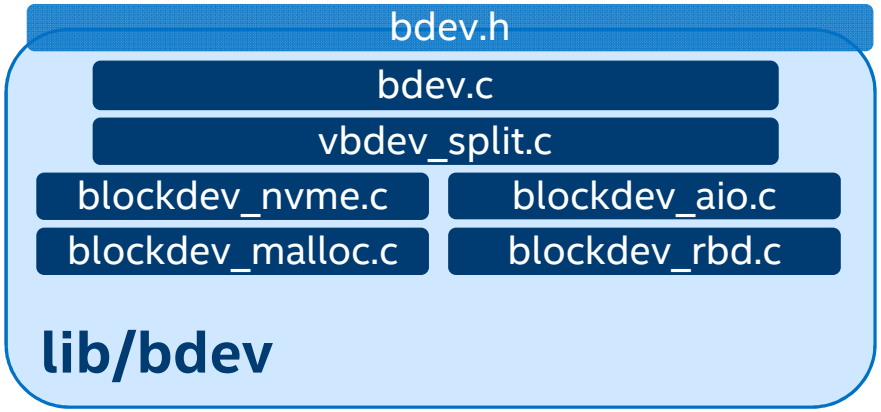
NVMe* (local, remote)

Malloc (RAM disk)

Linux libaio

Ceph RBD

Potential future work: pmem (NVML)



*Other names and brands may be claimed as the property of others.

BDEV LAYERING

Virtual blockdev drivers

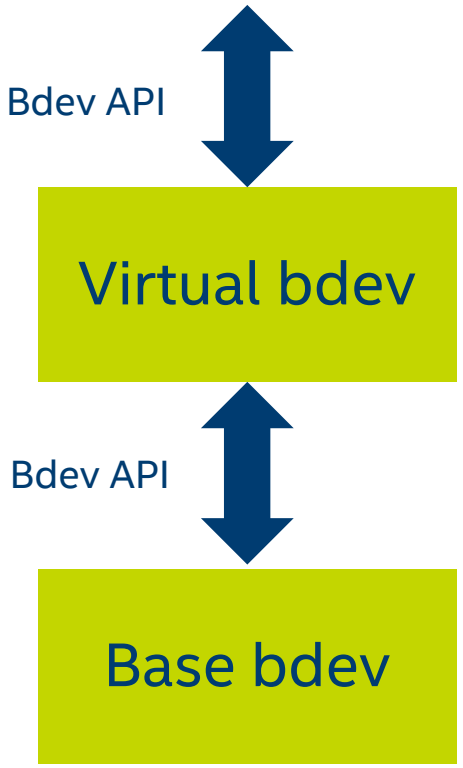
Claim base bdev(s)

Produce virtual bdev(s)

Provide storage services

Example: vbdev_split

Coming soon: Blob bdev

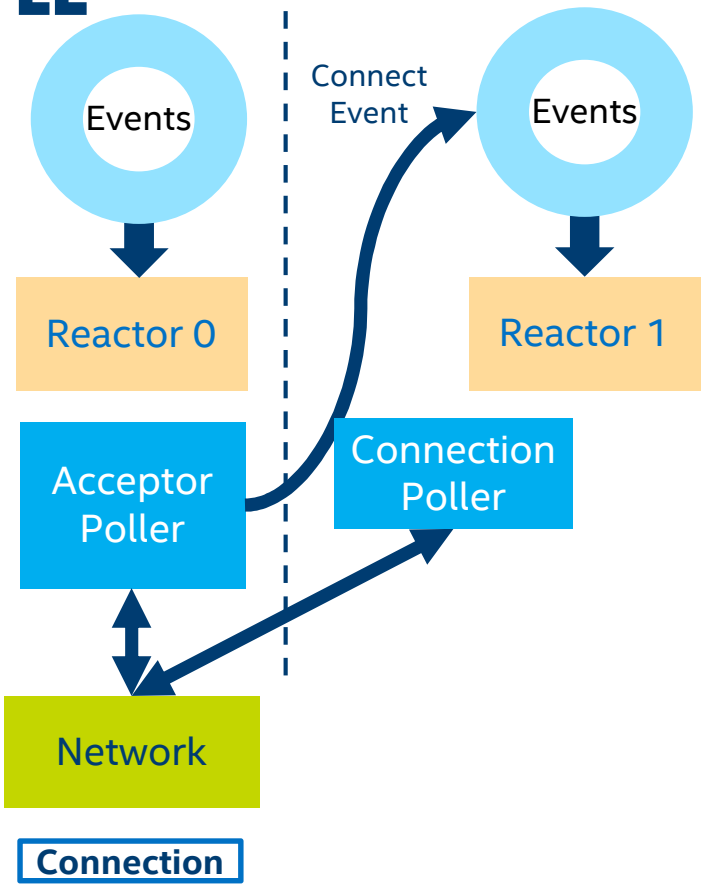


EXAMPLE APP WALKTHROUGHS

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

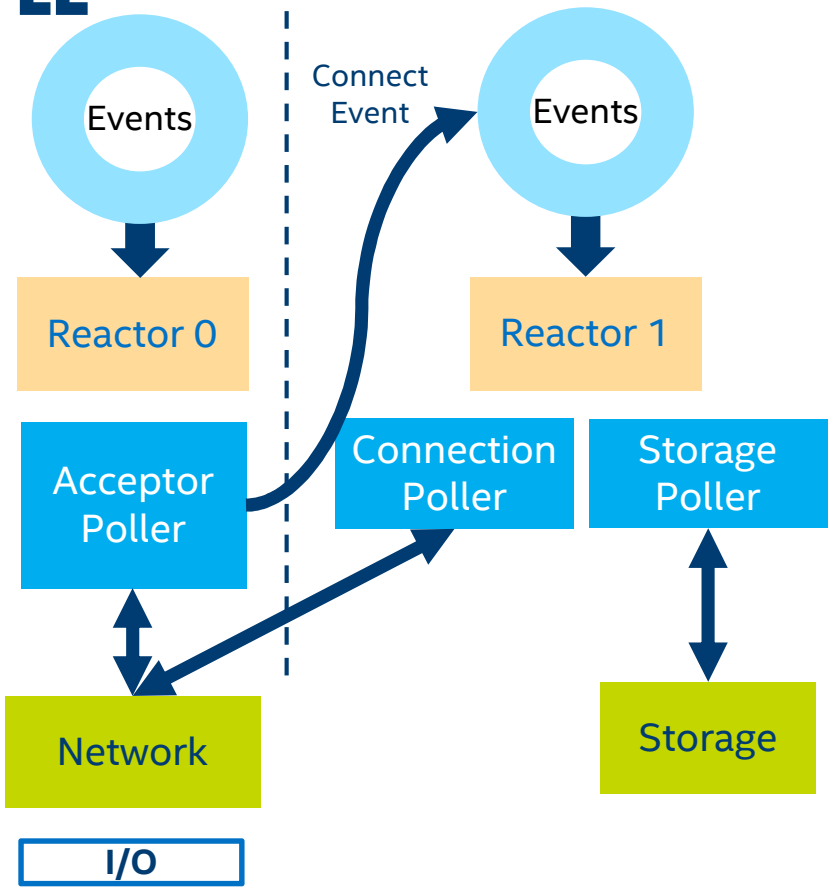
I/O request arrives over network

I/O submitted to storage

Storage device poller checks completions

Response sent

ALL ASYNCHRONOUS WORK IS DRIVEN BY POLLERS



VHOST-SCSI EXAMPLE

VM guest adds task to shared-memory queue

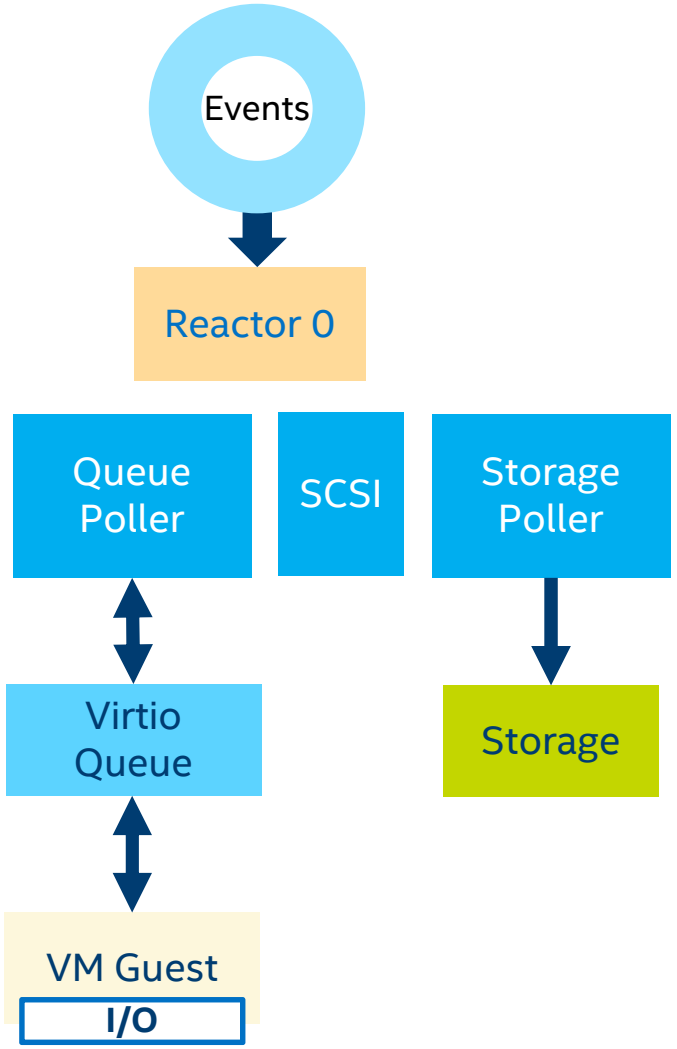
Task retrieved from queue and passed to SCSI

I/O submitted to storage

Storage poller completes I/O

SCSI layer signals completion by sending an event

Event completes I/O back to VM



EFFICIENCY

Software design follows from hardware capabilities

SIMPLICITY

Building blocks to manage asynchronous I/O

FLEXIBILITY

Swappable environment abstraction

Notices and Disclaimers

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.

No computer system can be absolutely secure.

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/performance**.


intel®
experience
what's inside™