A storage-abstraction-layer
for **traditional**
and **emerging**
storage **devices**
and their **interfaces**
via a unified **API**
enabling I/O interface **independence**
with **minimal** abstraction-layer **cost**

Simon A. F. Lund
Samsung
Copenhagen, Denmark
simon.lund@samsung.com

Last update: 27.10.2023

Agenda

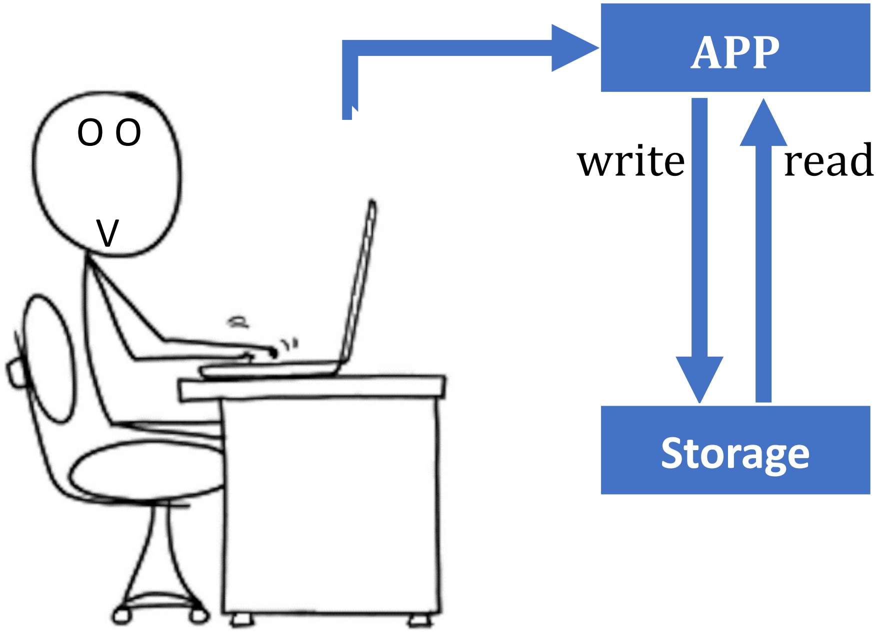
- The background and motivation for xNVMe
- The goal: I/O Interface Independence
- Brief overview of the API
- Performance evaluation
  - Command latency ➔ abstraction overhead
  - Peak IOPS / Physical CPU core ➔ efficiency when CPU bound

Background

Background

Traditional

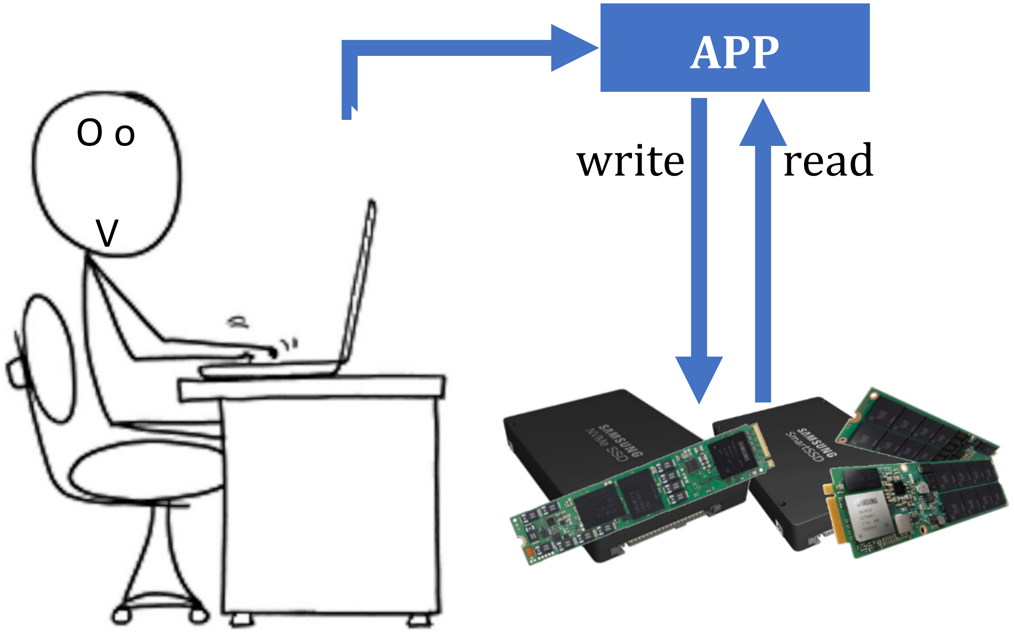
- Operating System Managed
- I/O is just reading and writing
- Storage device is the bottleneck



Background

Traditional + NVMe

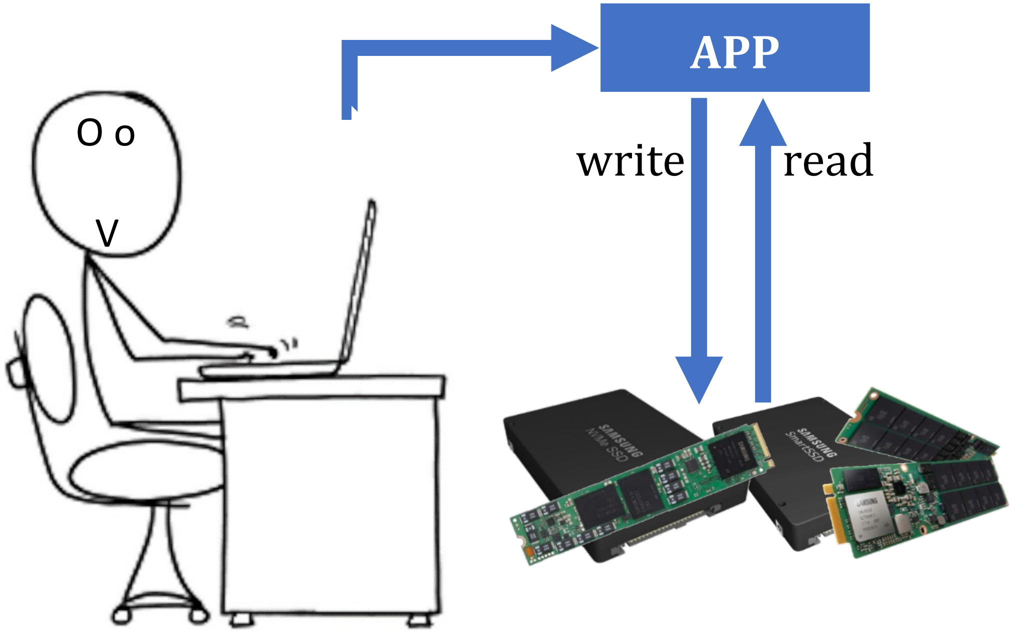
- Operating System Managed
- I/O is just reading and writing
- ~~Storage device is the bottleneck~~



Background

Reduce the cost of crossing the address-space boundary; system-call overhead, context-switching and memory mapping

Traditional + NVMe
- Operating System Managed
- I/O is just reading and writing
- ~~Storage device is the bottleneck~~



User Space

read()/write()
pread()/pwrite()
readv()/writev()
➔ **Threadpool for scale**

Kernel Space

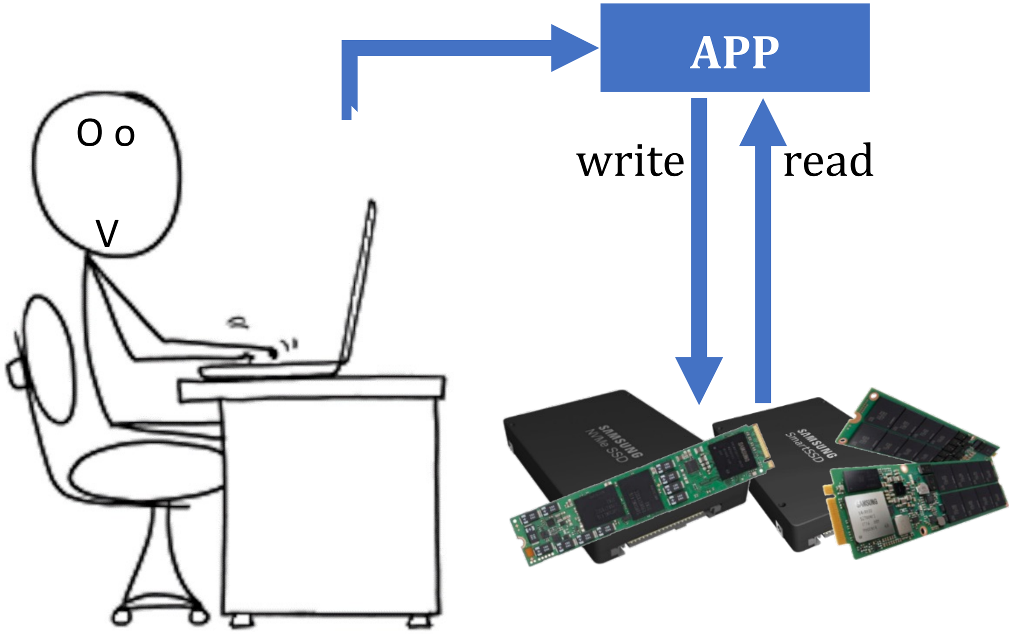
vfs

Block Layer

Background

Reduce the cost of crossing the address-space boundary;
system-call overhead, context-switching and memory mapping

Traditional + NVMe
- Operating System Managed
- I/O is just reading and writing
- ~~Storage device is the bottleneck~~



User Space | **Kernel Space**

read()/write()
pread()/pwrite()
readv()/writev()
➔**Threadpool for scale**
POSIX aio
Linux libaio
Windows IOCP
➔ **Interrupt Driven**
io_uring

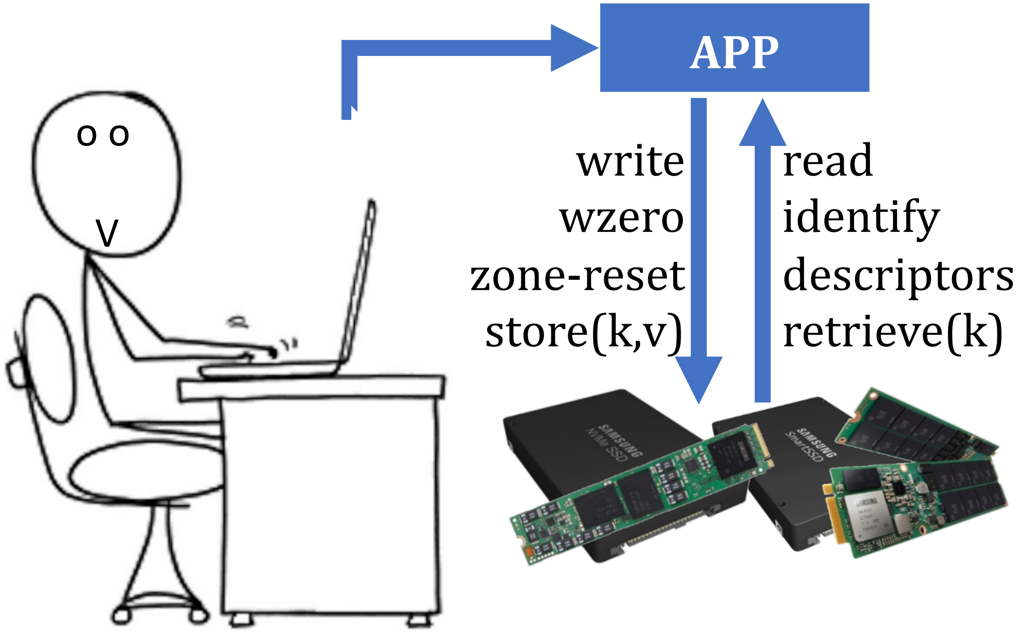
vfs
Block Layer
shared
shared
thread: sq poll
thread: driver-poll

Background

Reduce the cost of crossing the address-space boundary; system-call overhead, context-switching and memory mapping

Traditional + NVMe ZNS + KV

- Operating System Managed
- ~~I/O is just reading and writing~~
- ~~Storage device is the bottleneck~~



User Space		**Kernel Space**
read()/write()		vfs
pread()/pwrite()		Block Layer
readv()/writev()		
➔Threadpool for scale		
POSIX aio	shared	
Linux libaio		
Windows IOCP		
➔ Interrupt Driven		
io_uring	shared	thread: sq poll thread: driver-poll
ioctl() / devfs /sysfs		NVMe

Background

Reduce the cost of crossing the address-space boundary;
system-call overhead, context-switching and memory mapping

Traditional + NVMe ZNS + KV

- ~~Operating System Managed~~
- ~~I/O is just reading and writing~~
- ~~Storage device is the bottleneck~~

APP

write
wzero
zone-reset
store(k,v)

read
identify
descriptors
retrieve(k)

User Space | **Kernel Space**

read()/write()
pread()/pwrite()
readv()/writev()
➔ **Threadpool for scale**

vfs
Block Layer

POSIX aio
Linux libaio
Windows IOCP
➔ **Interrupt Driven**

shared

io_uring

shared

thread: sq poll
thread: driver-poll

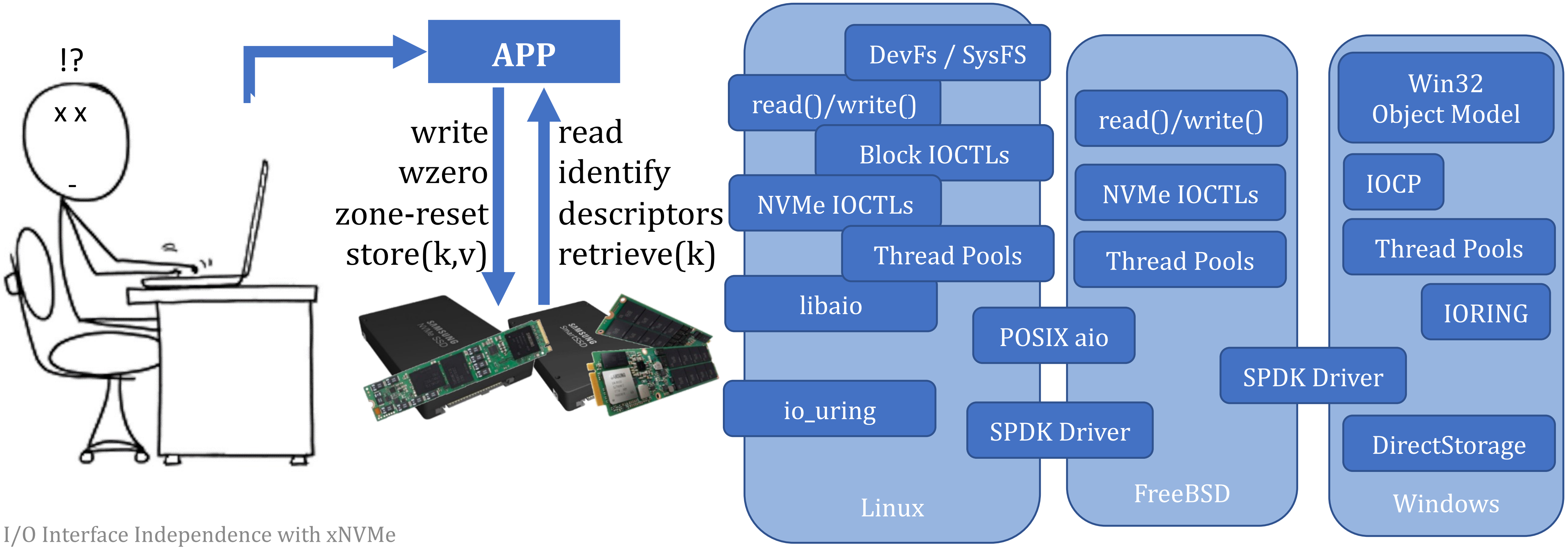
ioctl() / devfs /sysfs
SPDK/NVMe
(user space driver)
➔ **Kernel Bypass**

NVMe
vfio-pci /uio-generic

Background

I/O interface innovation

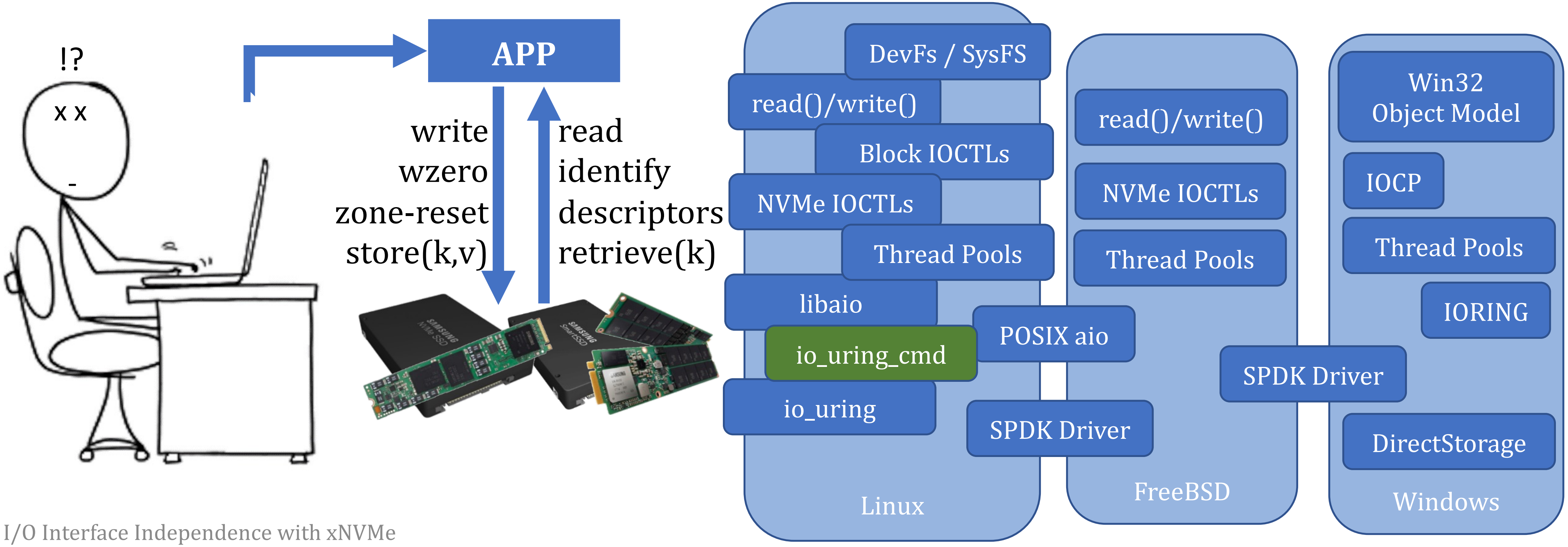
- ~~Operating System Managed~~
- ~~I/O is just reading and writing~~
- ~~Storage device is the bottleneck~~



Background

I/O interface innovation

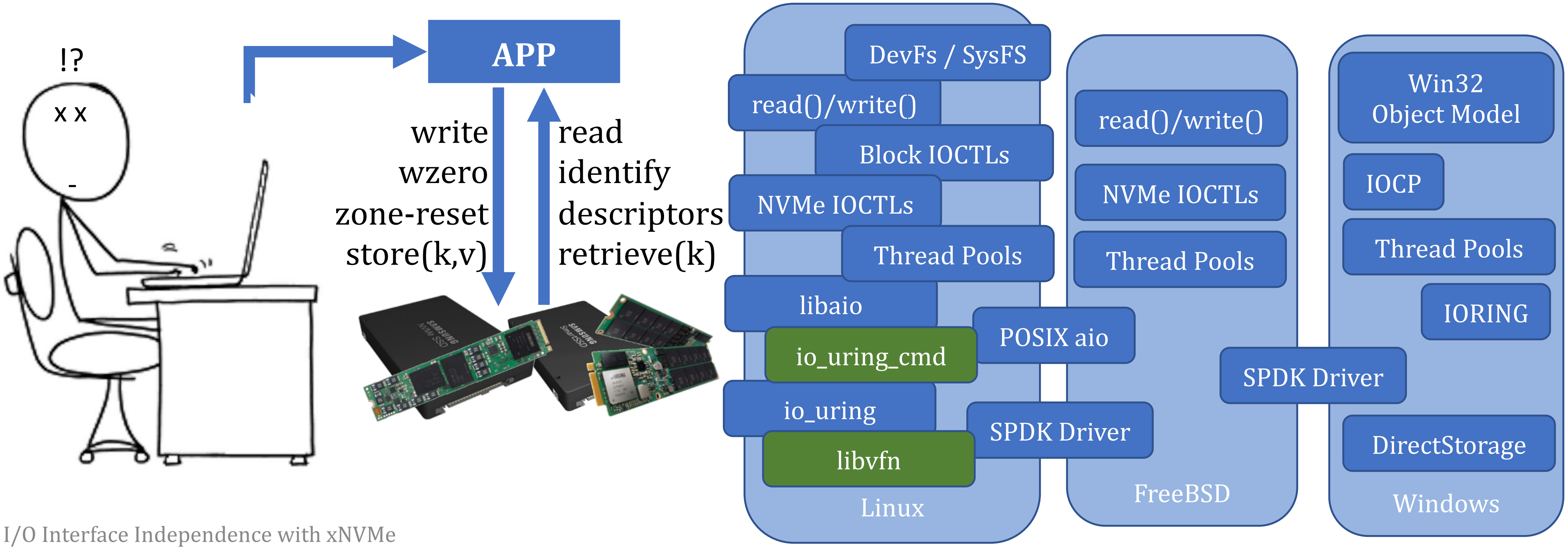
- ~~Operating System Managed~~
- ~~I/O is just reading and writing~~
- ~~Storage device is the bottleneck~~



Background

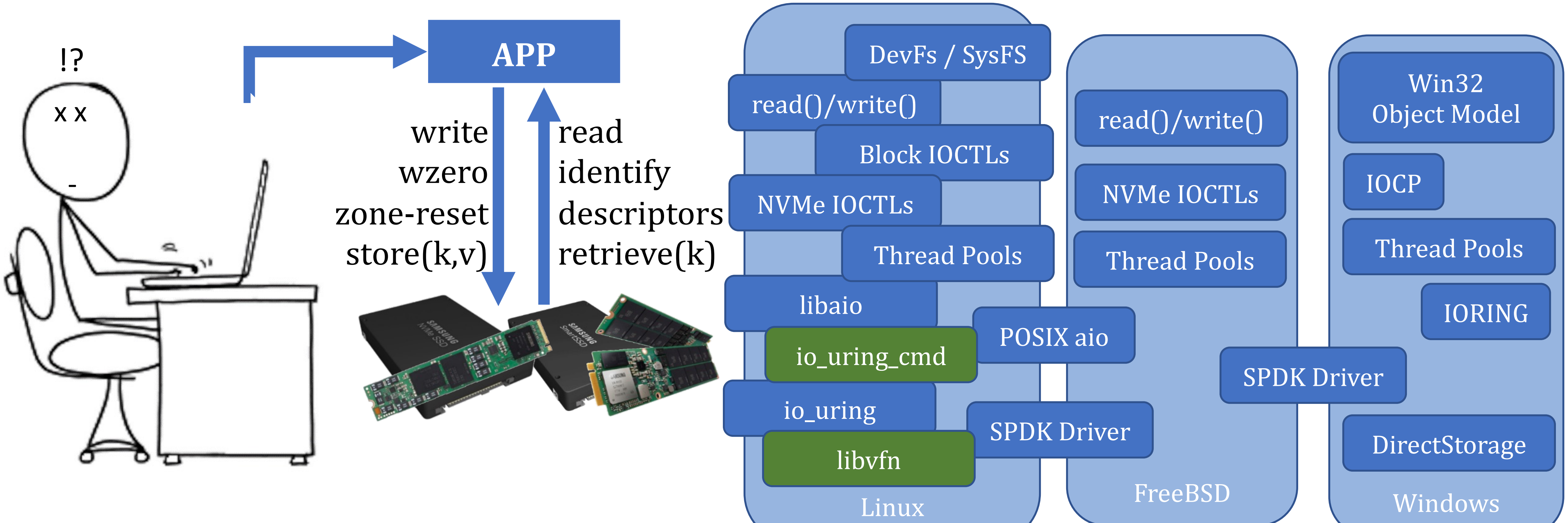
I/O interface innovation

- ~~Operating System Managed~~
- ~~I/O is just reading and writing~~
- ~~Storage device is the bottleneck~~



Background: the problem

- We are in a time of interesting system interface changes, fluctuating from operating system managed, unikernels and OS bypass.
- Additionally, storage device interfaces are expanding with new command sets
- **Question:** How do you manage, and leverage, I/O interface innovation?



Background: the problem

We denote **I/O interface independence** the following property of a data-intensive system: *changing I/O interface does not require refactoring the rest of the system*.

Our hypothesis is that I/O interface independence can be achieved at negligible performance cost.

Background: the problem

- **Negligible** performance cost, how much is that?

Background: the problem

- **Negligible** performance cost, how much is that?
- Ideally less than other means of **I/O routing**
  - I/O routing through PCIe switch ~**150 nsec**
  - I/O routing through PCH ~**865 nsec**
  - I/O routing through OS storage stack ~**1500 nsec**
- In relation to media access times
  - I/O access on "fast" NAND in an NVMe SSD ~**7.000 nsec**
  - I/O access on "slow" NAND in an NVMe SSD is ~**60.000 nsec**

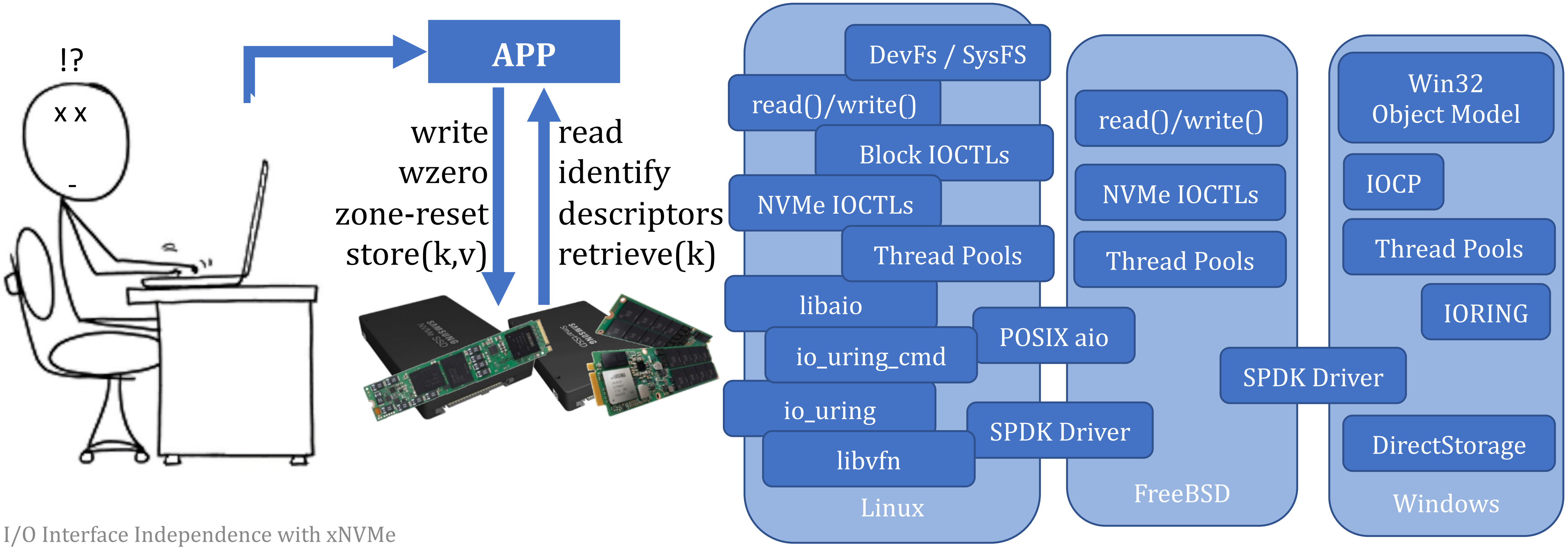
- A small fraction of media-access time,
  relative to other means of I/O routing ➔ low hundreds

4k random read at QD1	Latency (**nsec**)
Connected via **PCIe slot** **Lanes directly to CPU**	6455

4k random read at QD1	Latency (**nsec**)
Connected via **M.2** port **Lanes via PCH to CPU**	7376

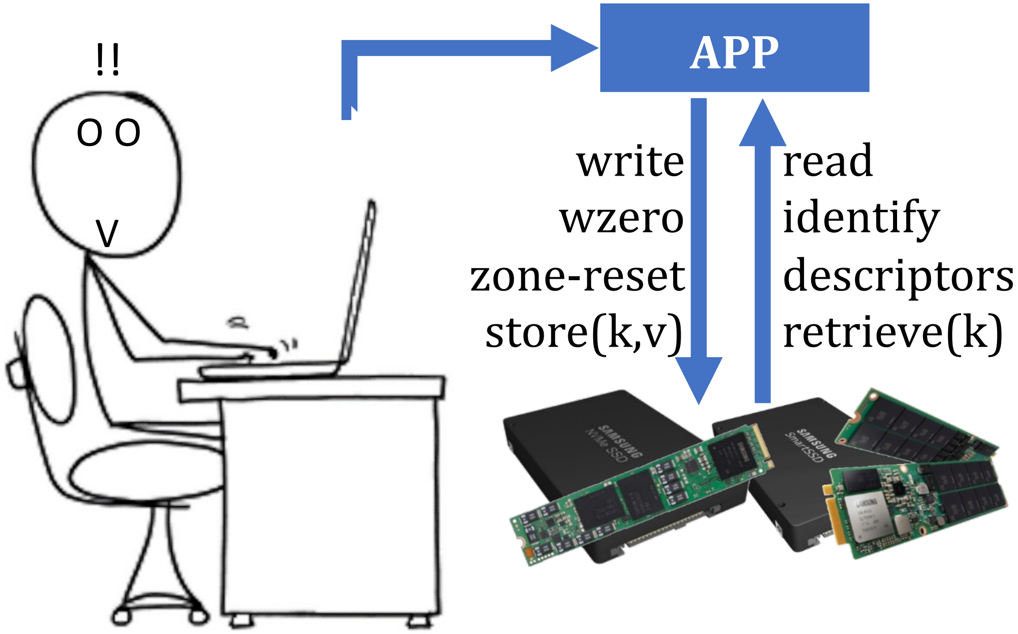
Background: the problem

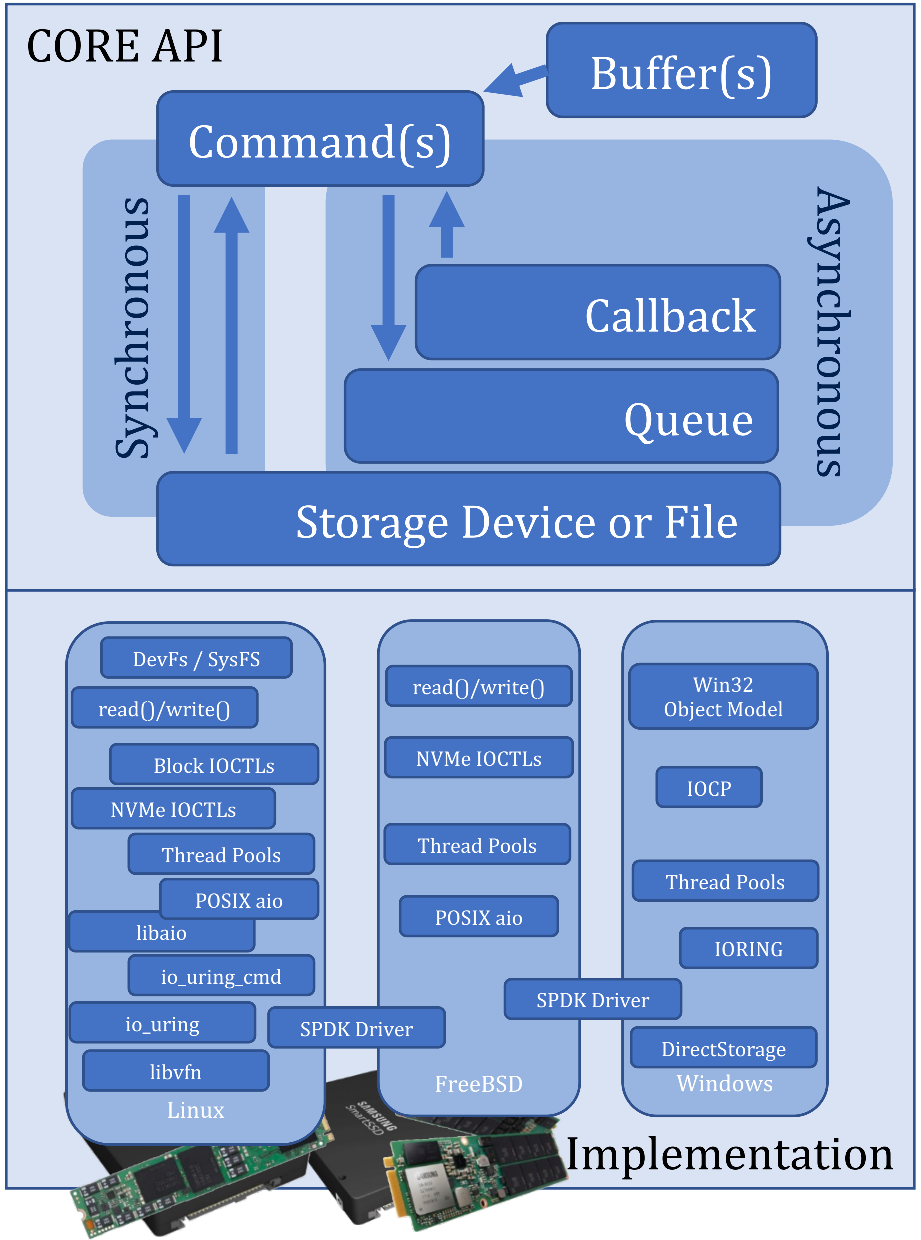
- **Questions**
  - Is I/O interface independence possible? And at what cost?
  - How do you manage, and leverage, I/O interface innovation?



I/O Interface Independence with **xNVMe**

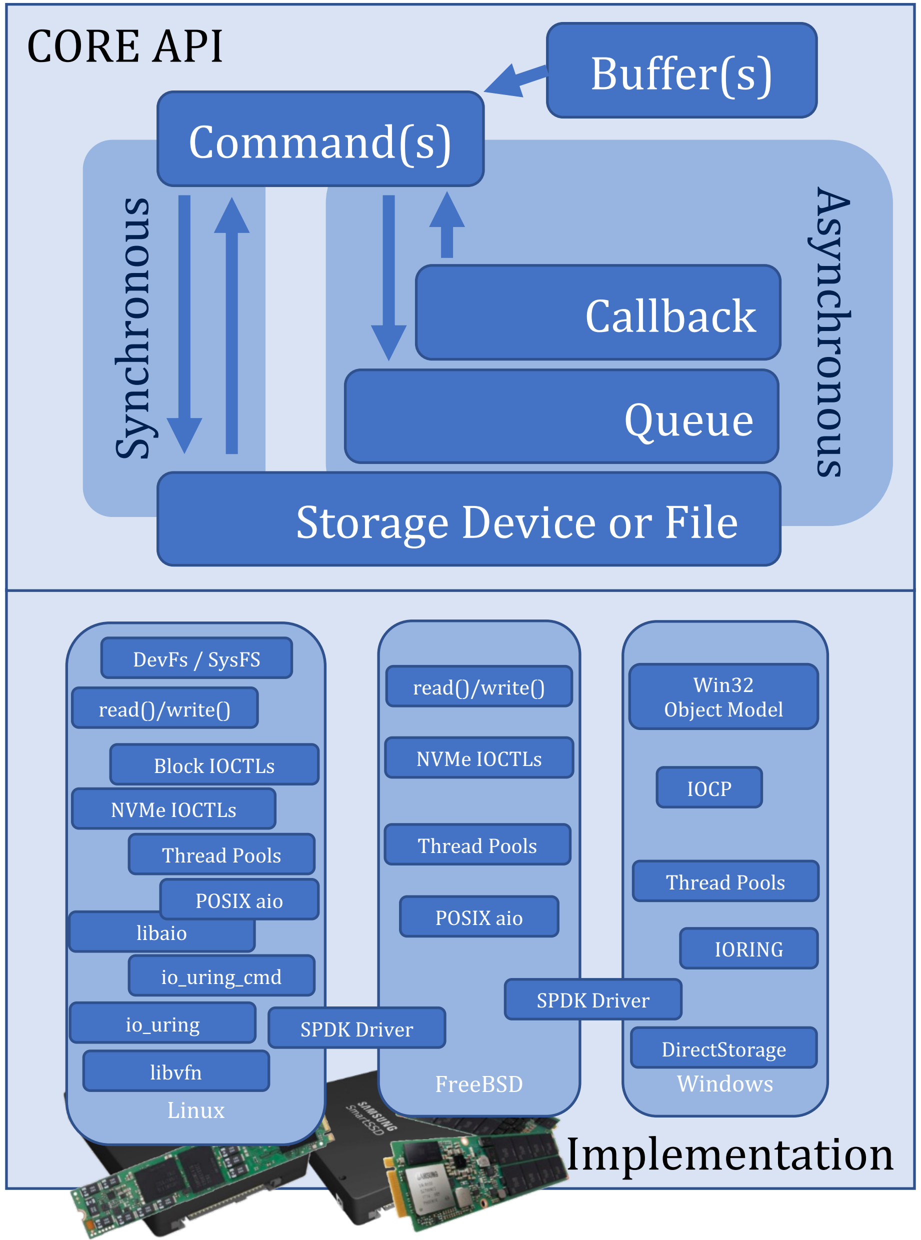
- I/O interface independence with negligible performance cost
  - Extensible, Simple and Uniform
- Minimal spanning-layer





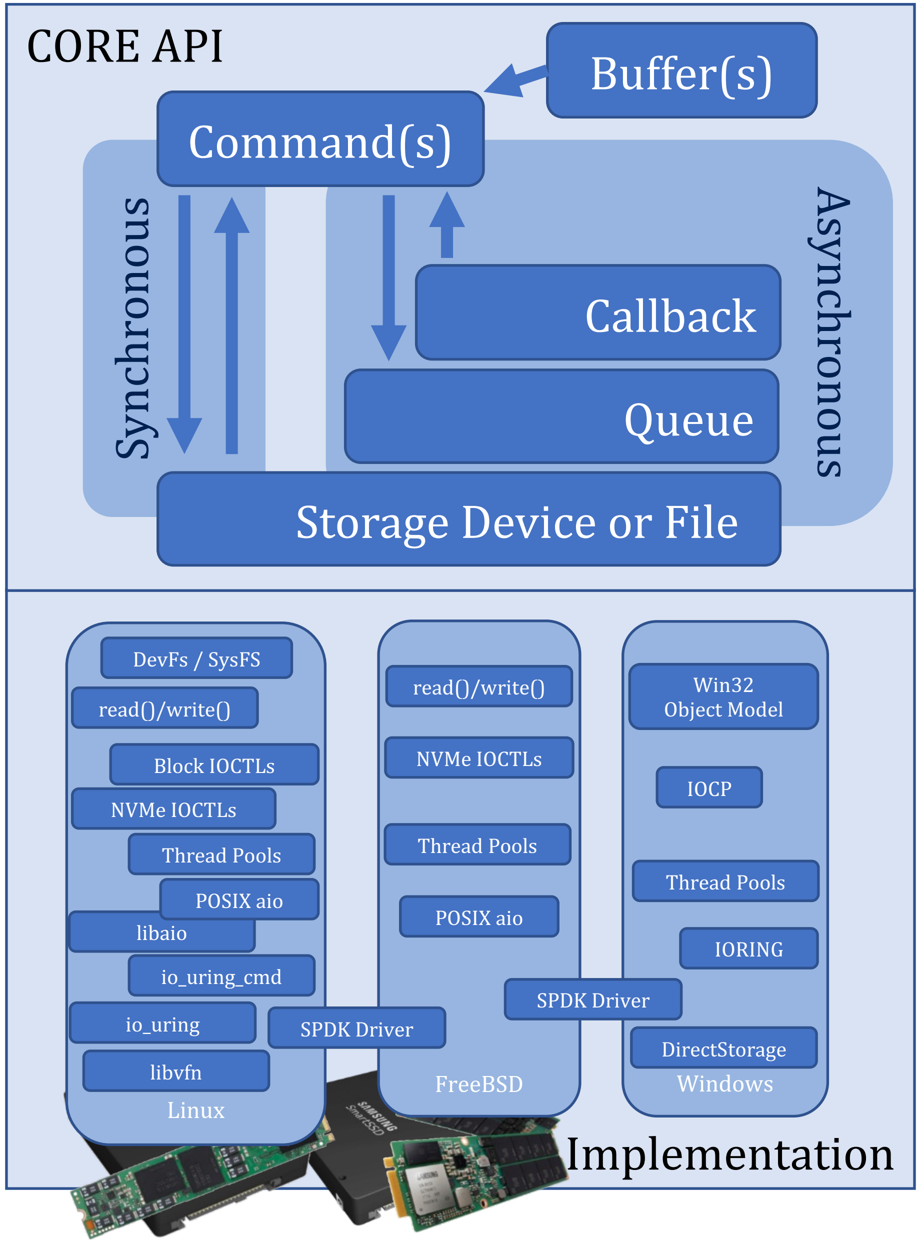
I/O Interface Independence with **xNVMe:** API

- Device Handles
- Buffers
- Commands
  - Synchronous
  - Asynchronous



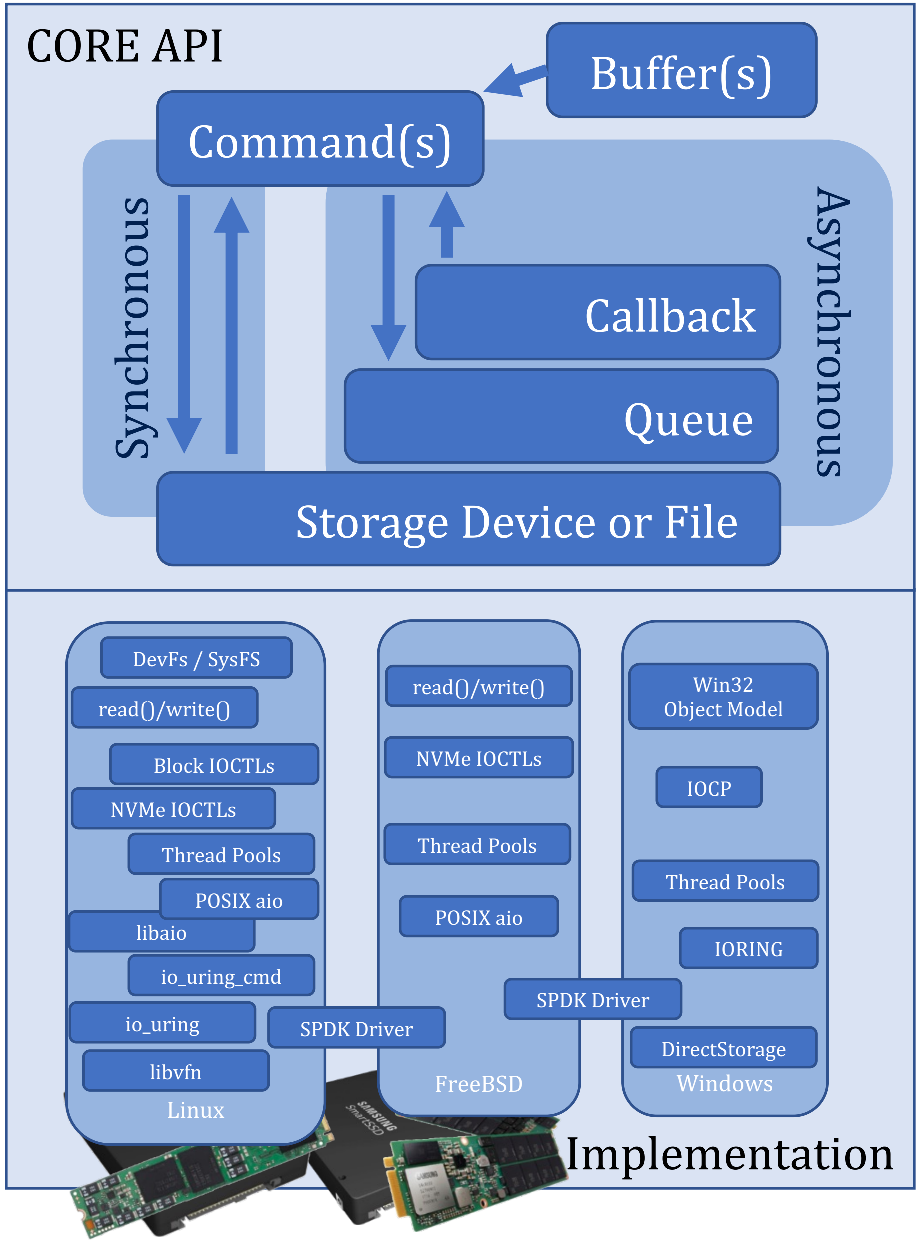
I/O Interface Independence with **xNVMe:** API

- **Device Handles**
- Buffers
- Commands
  - Synchronous
  - Asynchronous



I/O Interface Independence with **xNVMe:** API

- **Device Handles**
  - `xnvme_enumerate(uri, opts, cb, args)`
  - `xnvme_dev_open(uri, opts)`



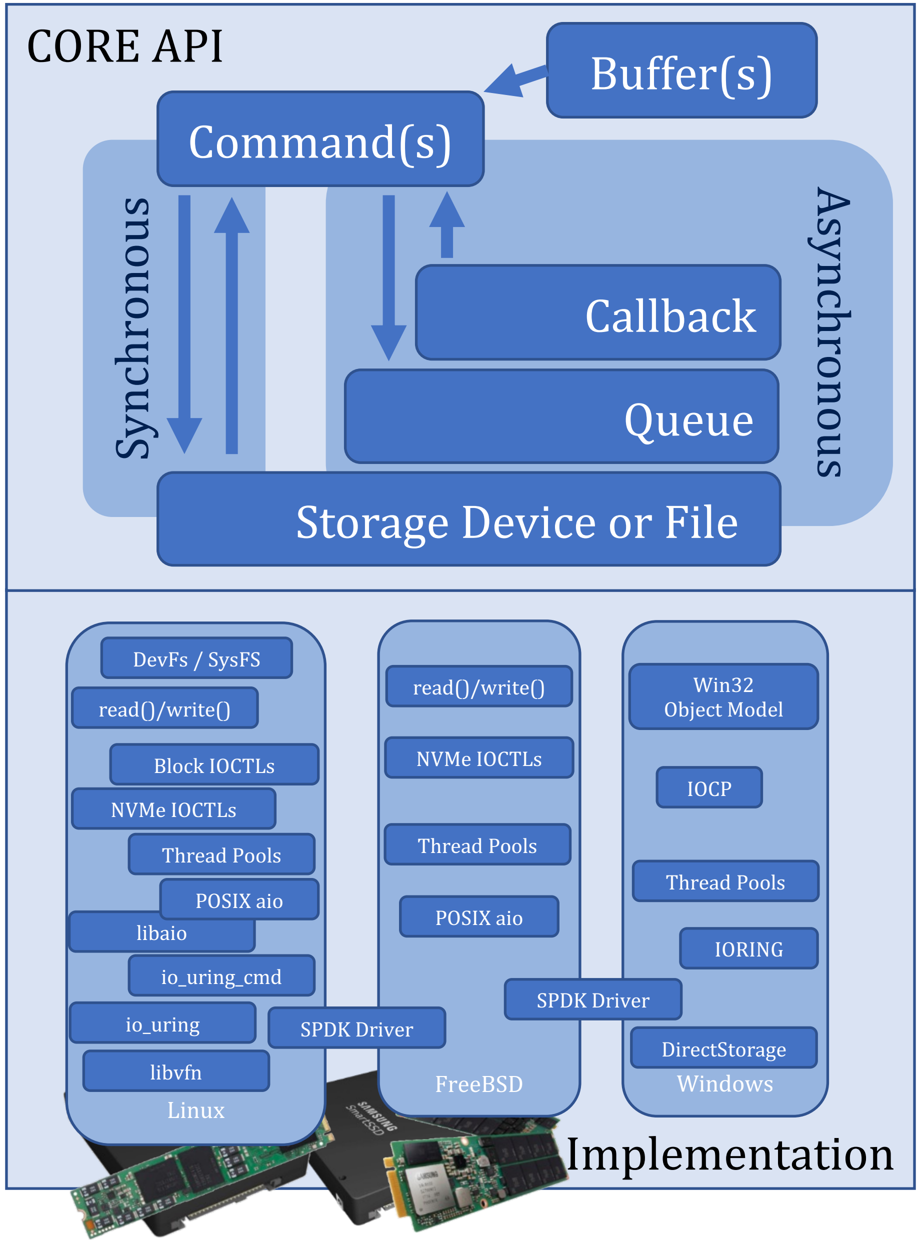
I/O Interface Independence with **xNVMe:** API

- **Device Handles**
  - `xnvme_enumerate(uri, opts, cb, args)`

Invoked for each device
`cb(dev, args)`

`NULL`
Local system

`“10.11.12.185:4420”`
Fabrics Transport



I/O Interface Independence with **xNVMe:** API

- **Device Handles**
  - `xnvme_enumerate(uri, opts, cb, args)`

NULL

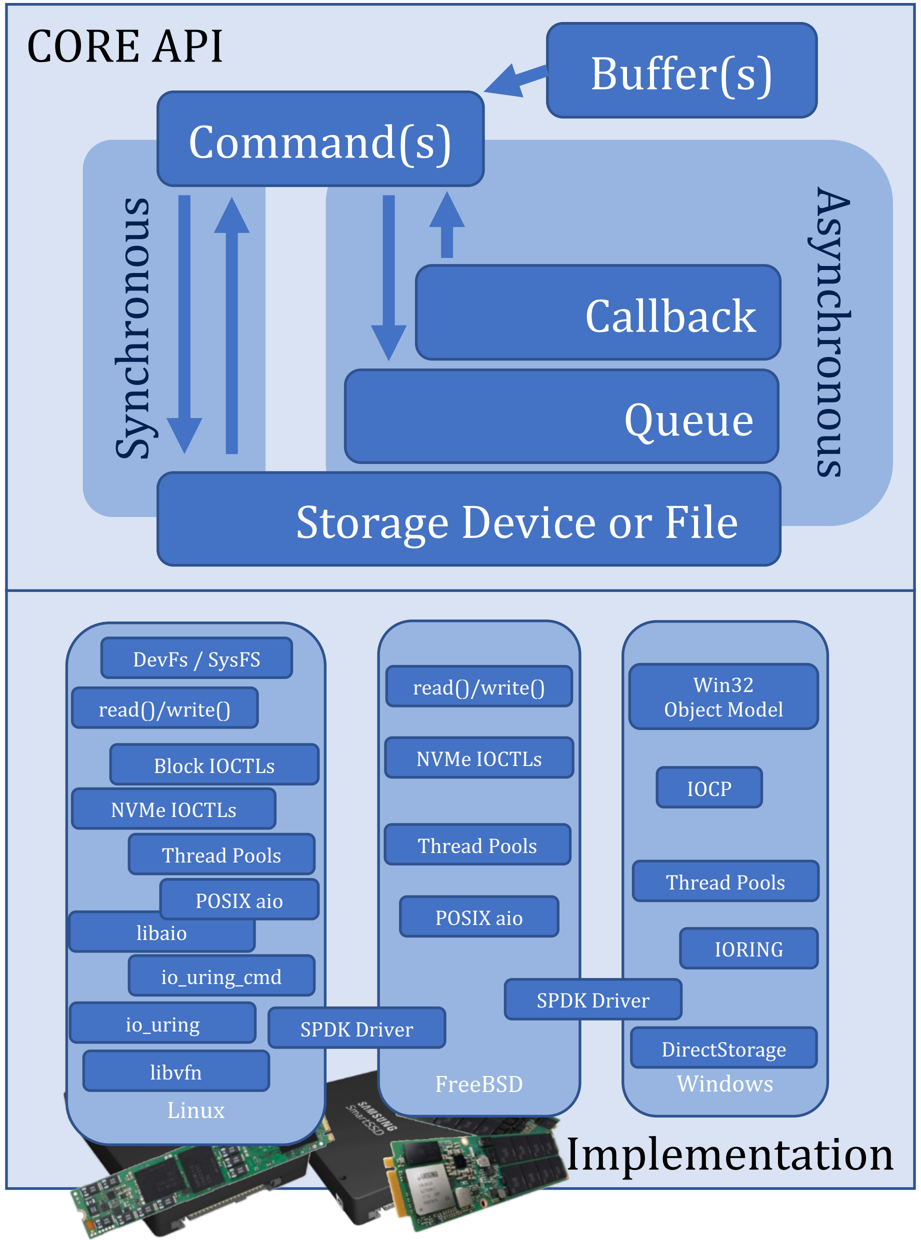
Local system

User space NVMe Driver

```
root@corei5:~# xnvme enum
xnvme_enumeration:
  - {uri: '0000:04:00.0', dtype: 0x2, nsid: 0x1, csi: 0x0}
  - {uri: '/dev/nvme0n1', dtype: 0x2, nsid: 0x1, csi: 0x0}
  - {uri: '/dev/ng0n1', dtype: 0x2, nsid: 0x1, csi: 0x0}
```

OS Managed NVMe NS (Block Device)

OS Managed NVMe NS (Char Device)

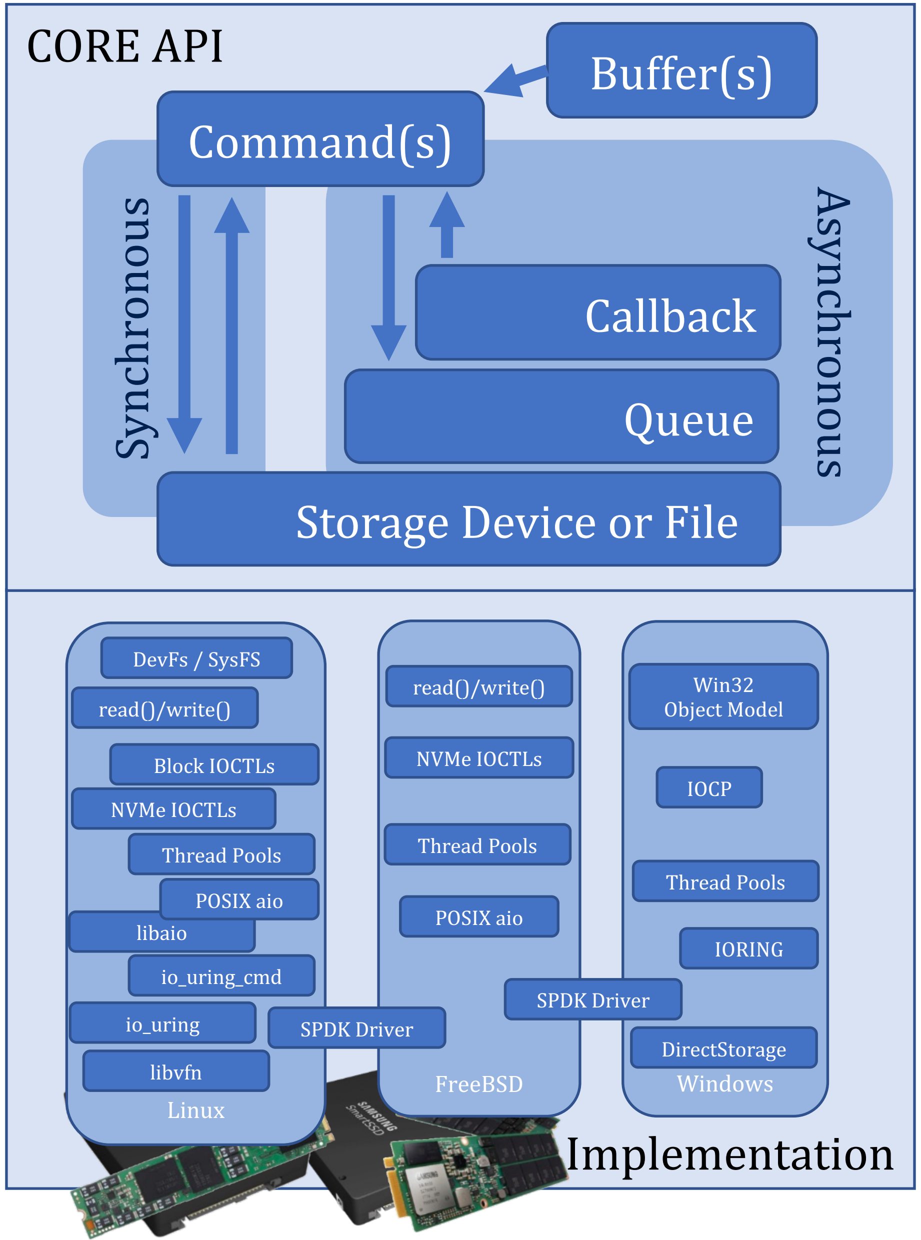


I/O Interface Independence with **xNVMe:** API

- **Device Handles**
  - xnvme_enumerate(uri, opts, cb, args)

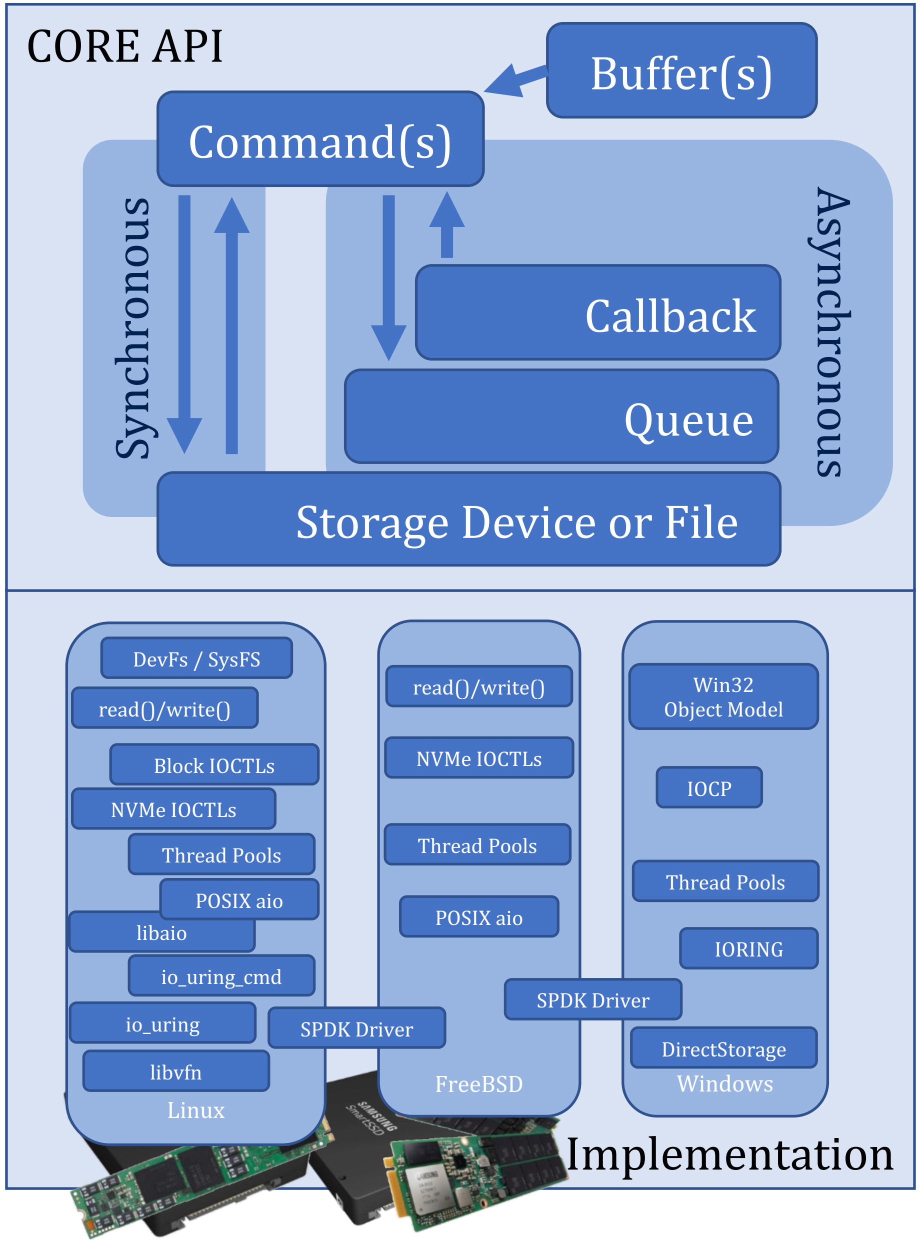
"10.11.12.185:4420"
Fabrics Transport

```
safl@debtop:~$ xnvme enum --uri 10.11.12.185:4420
xnvme_enumeration:
  - {uri: '10.11.12.185:4420', dtype: 0x2, nsid: 0x1, csi: 0x0}
safl@debtop:~$
```



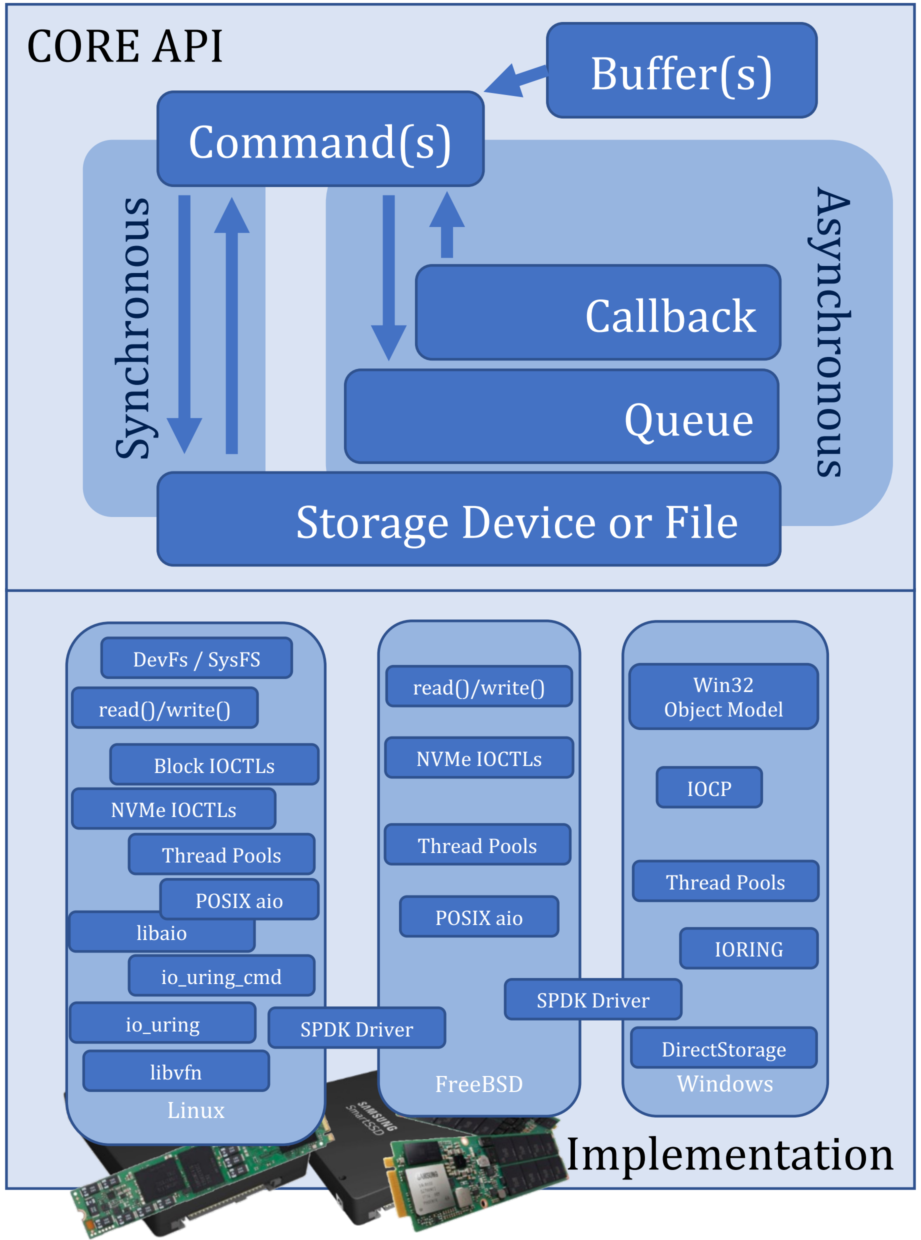
I/O Interface Independence with **xNVMe:** API

- **Device Handles**
  - `xnvme_enumerate(uri, opts, cb, args)`
  - `**dev** = xnvme_dev_open(uri, opts)`



I/O Interface Independence with **xNVMe:** API

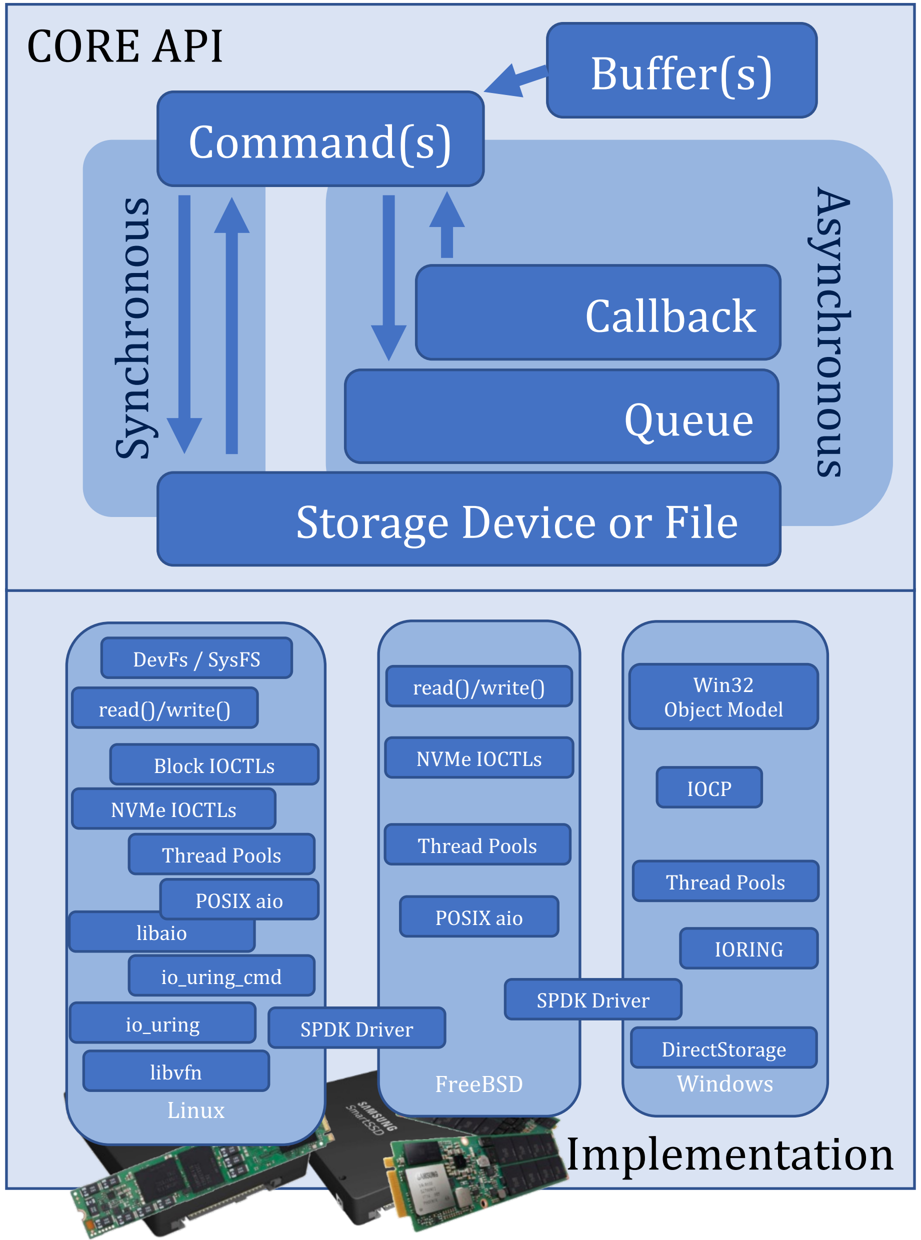
- **Device Handles**
  - `xnvme_enumerate(uri, opts, cb, args)`
  - `dev = xnvme_dev_open(uri, opts)`
  - URI Examples (CLI tool)



I/O Interface Independence with **xNVMe:** API

- **Device Handles**
  - `xnvme_enumerate(uri, opts, cb, args)`
  - `dev = xnvme_dev_open(uri, opts)`
  - URI Examples (CLI tool)

```
xnvme info /dev/ng0n1 --dev-nsid 0x1
xnvme info 0000:04:00.0 --dev-nsid 0x1
xnvme info 10.11.12.185:4420 -dev-nsid 0x1
xnvme info /dev/sda
xnvme info /dev/nullb0
```

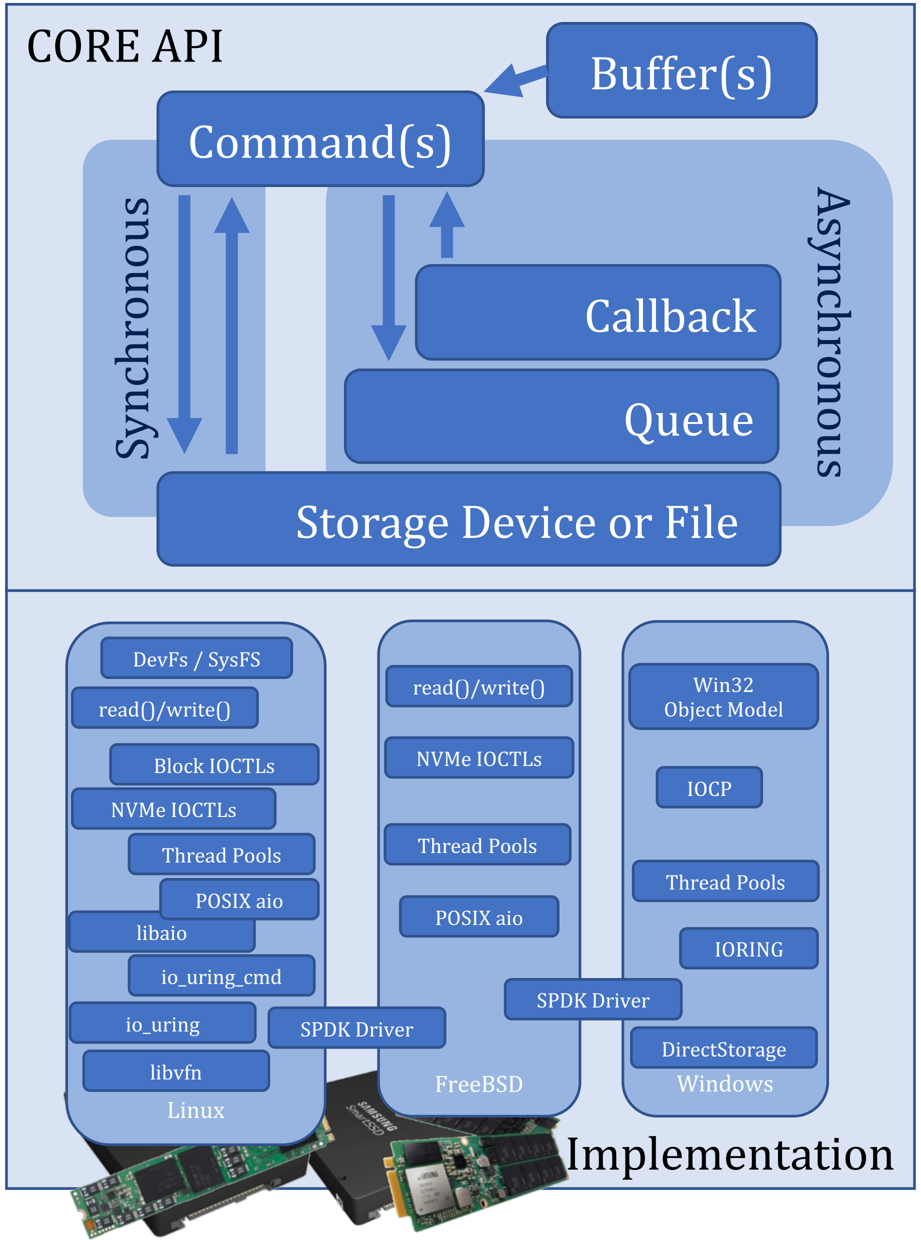


I/O Interface Independence with **xNVMe:** API

- **Device Handles**
  - `xnvme_enumerate(uri, opts, cb, args)`
  - `**dev** = xnvme_dev_open(uri, opts)`
  - URI Examples (CLI tool)

```
xnvme info /dev/ng0n1 --dev-nsid 0x1
xnvme info 0000:04:00.0 --dev-nsid 0x1
xnvme info 10.11.12.185:4420 -dev-nsid 0x1
xnvme info /dev/sda
xnvme info /dev/nullb0
```

Traditional { xnvme info /dev/sda, xnvme info /dev/nullb0 }

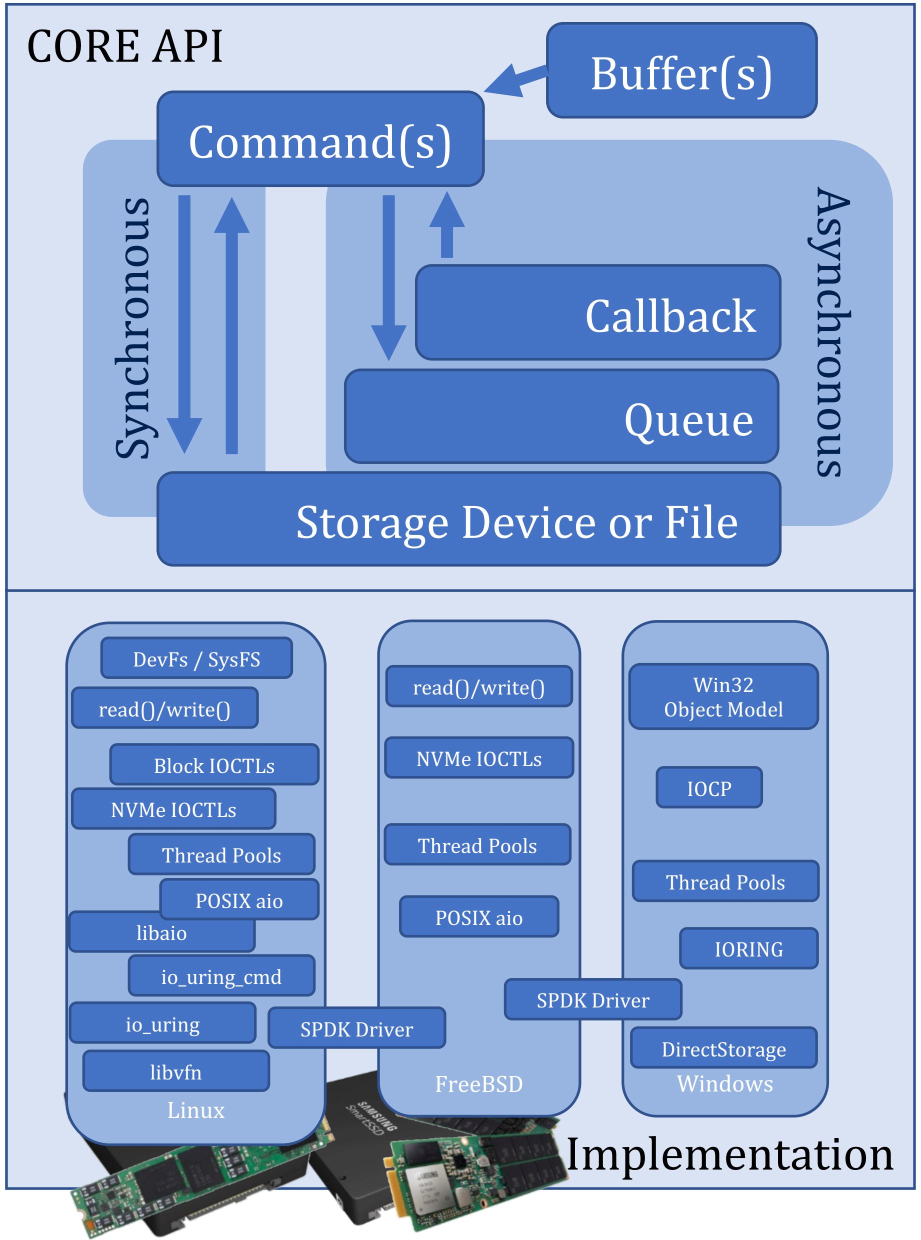


I/O Interface Independence with **xNVMe:** API

- **Device Handles**
  - `xnvme_enumerate(uri, opts, cb, args)`
  - `**dev** = xnvme_dev_open(uri, opts)`
  - URI Examples (CLI tool)

NVMe {
```
xnvme info /dev/ng0n1 --dev-nsid 0x1
xnvme info 0000:04:00.0 --dev-nsid 0x1
xnvme info 10.11.12.185:4420 -dev-nsid 0x1
```

Traditional {
```
xnvme info /dev/sda
xnvme info /dev/nullb0
```



I/O Interface Independence with **xNVMe:** API

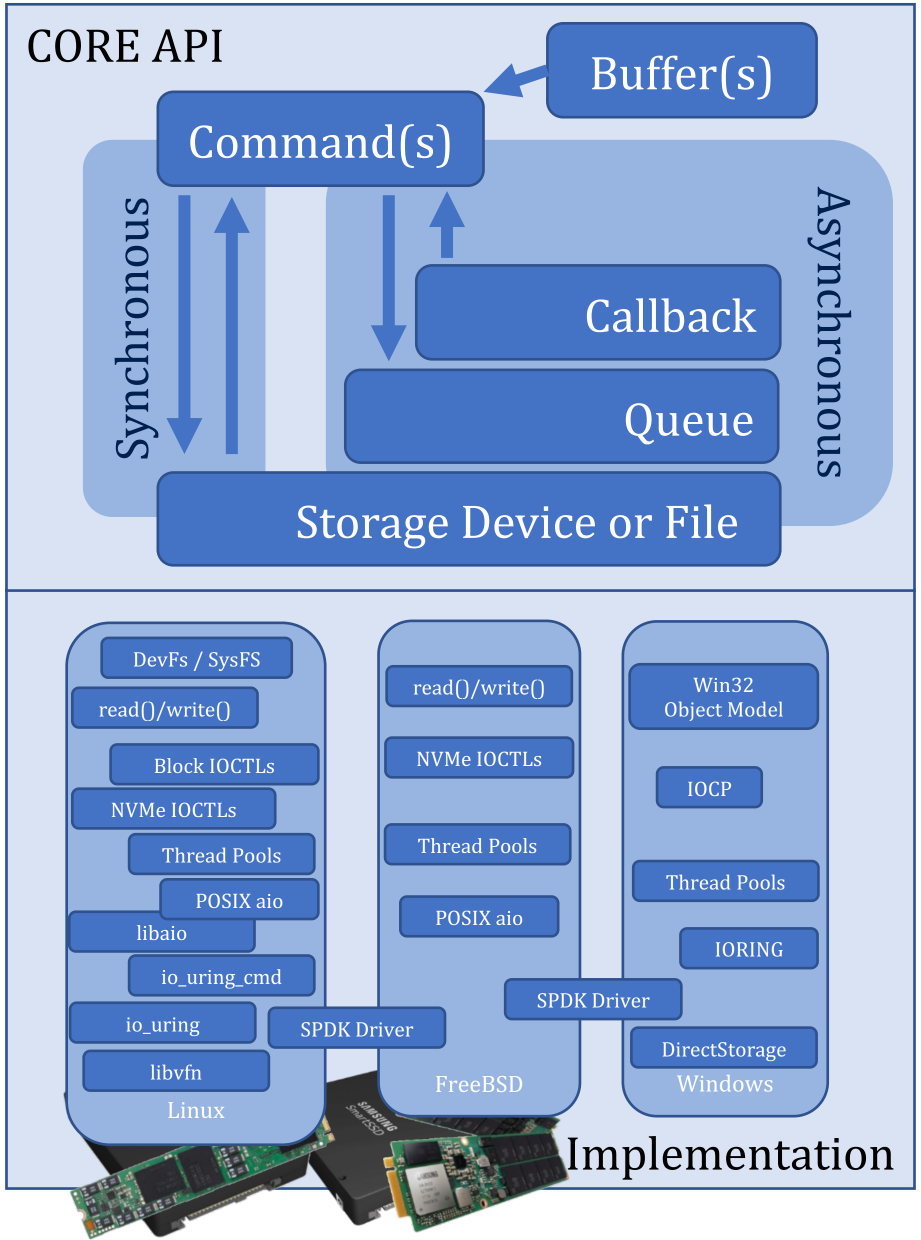
- **Device Handles**
  - `xnvme_enumerate(uri, opts, cb, args)`
  - `**dev** = xnvme_dev_open(uri, opts)`
  - URI Examples (CLI tool)

NVMe
```
xnvme info /dev/ng0n1 --dev-nsid 0x1
xnvme info 0000:04:00.0 --dev-nsid 0x1
xnvme info 10.11.12.185:4420 -dev-nsid 0x1
```

Traditional
```
xnvme info /dev/sda
xnvme info /dev/nullb0
```

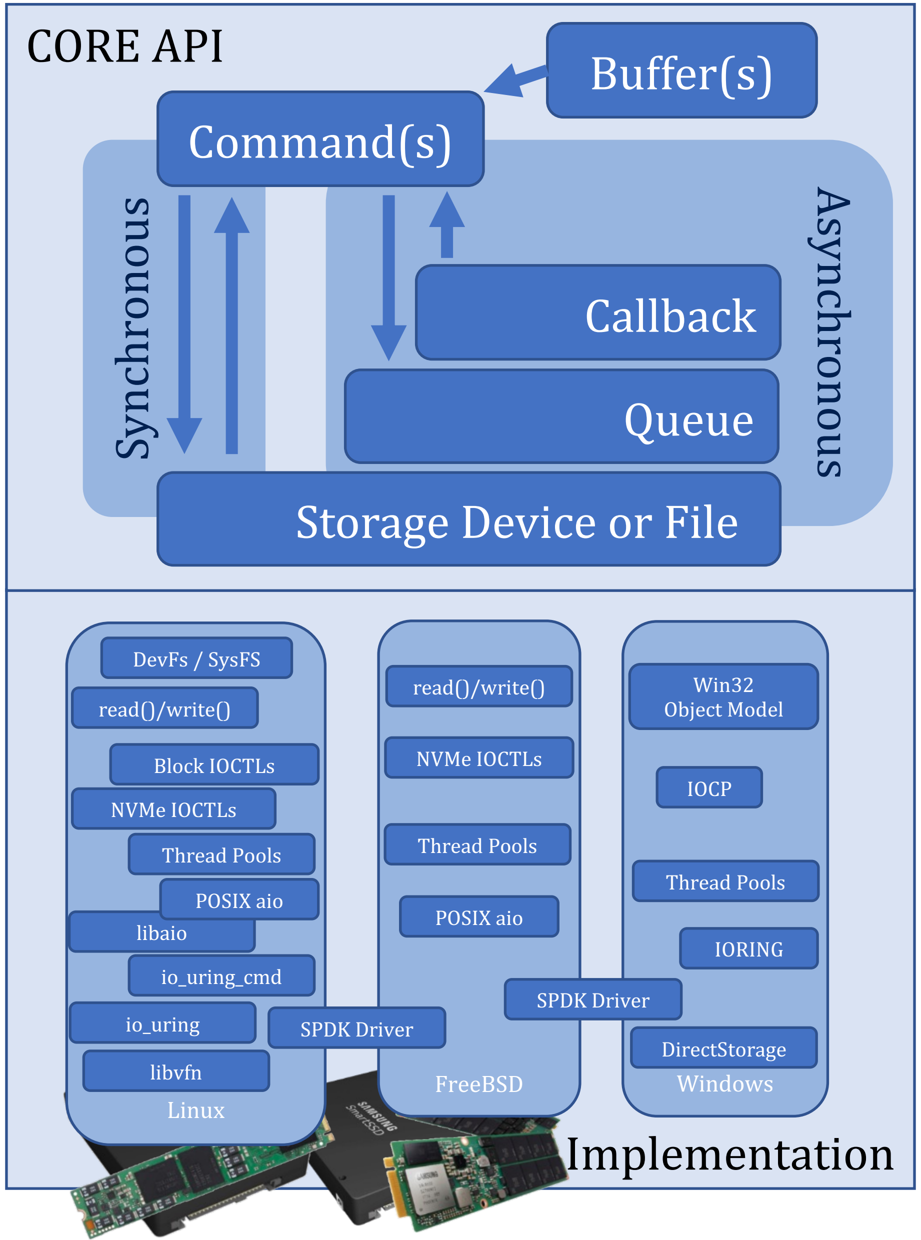
  - OPTS Examples (C API)

```
opts = { .async = ”io_uring” }
opts = { .async = ”libaio” }
opts = { .async = ”thrpool”, .sync = ”nvme” }
opts = { .async = ”thrpool”, .sync = ”psync” }
```



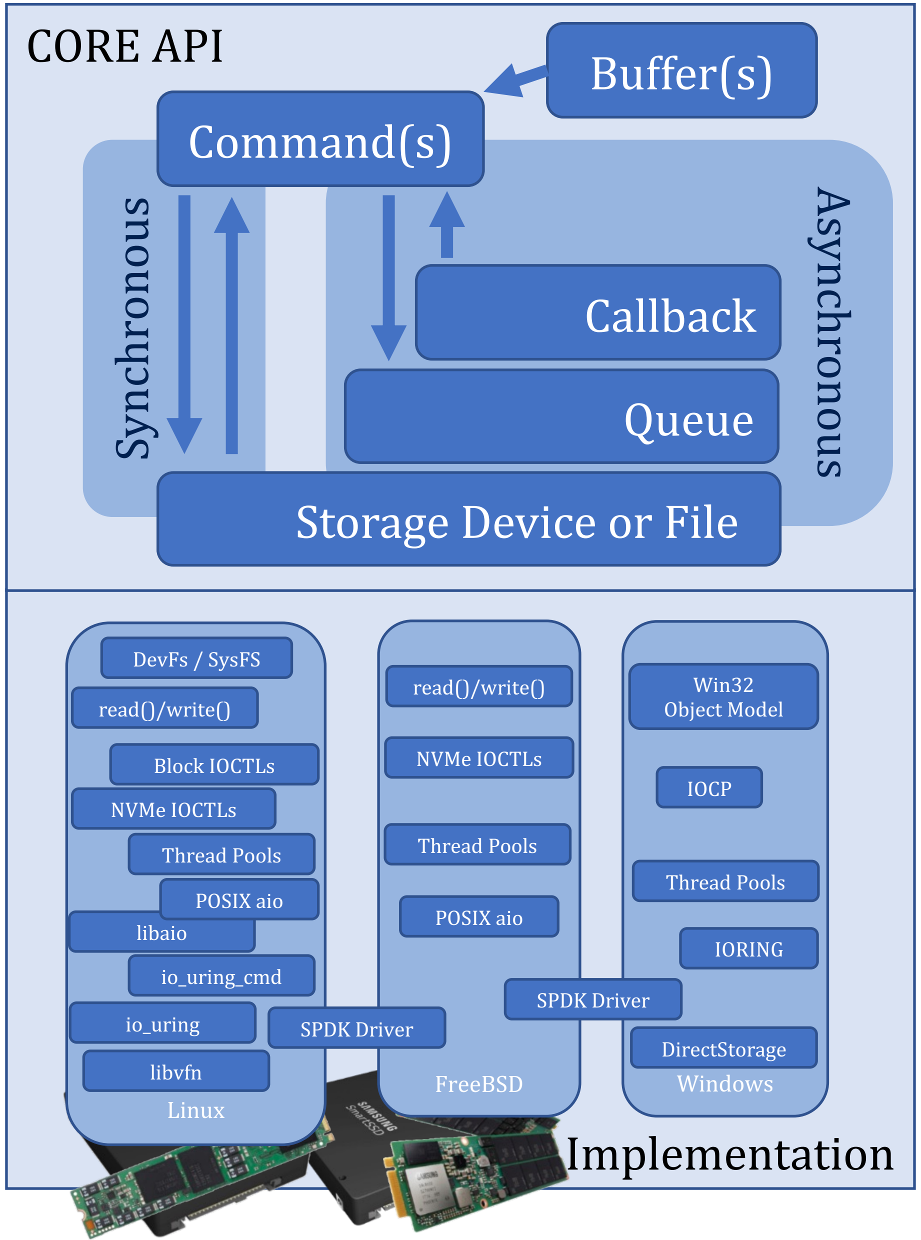
I/O Interface Independence with **xNVMe:** API

- **Device Handles**
- Buffers
- Commands
  - Synchronous
  - Asynchronous



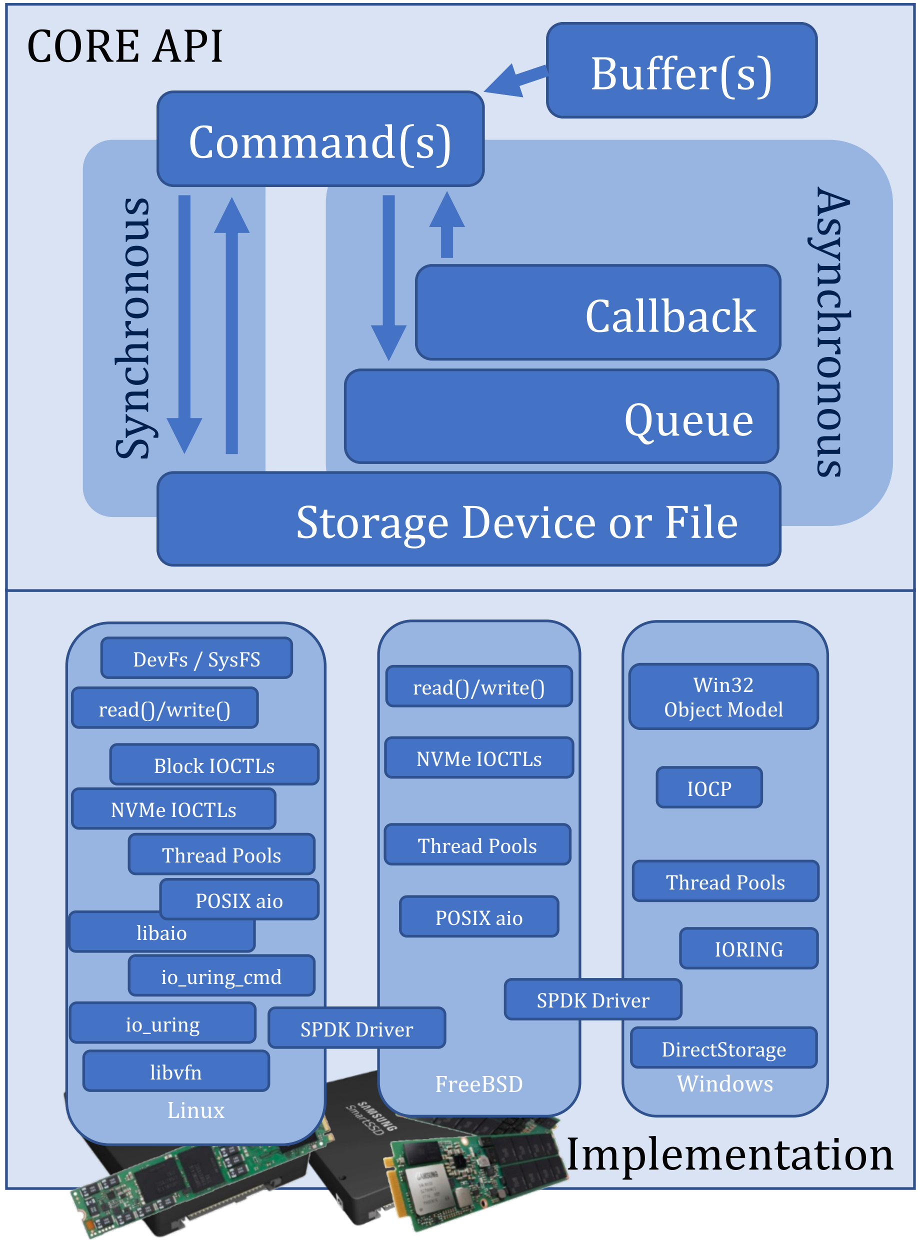
I/O Interface Independence with **xNVMe:** API

- Device Handles
- **Buffers**
- Commands
  - Synchronous
  - Asynchronous



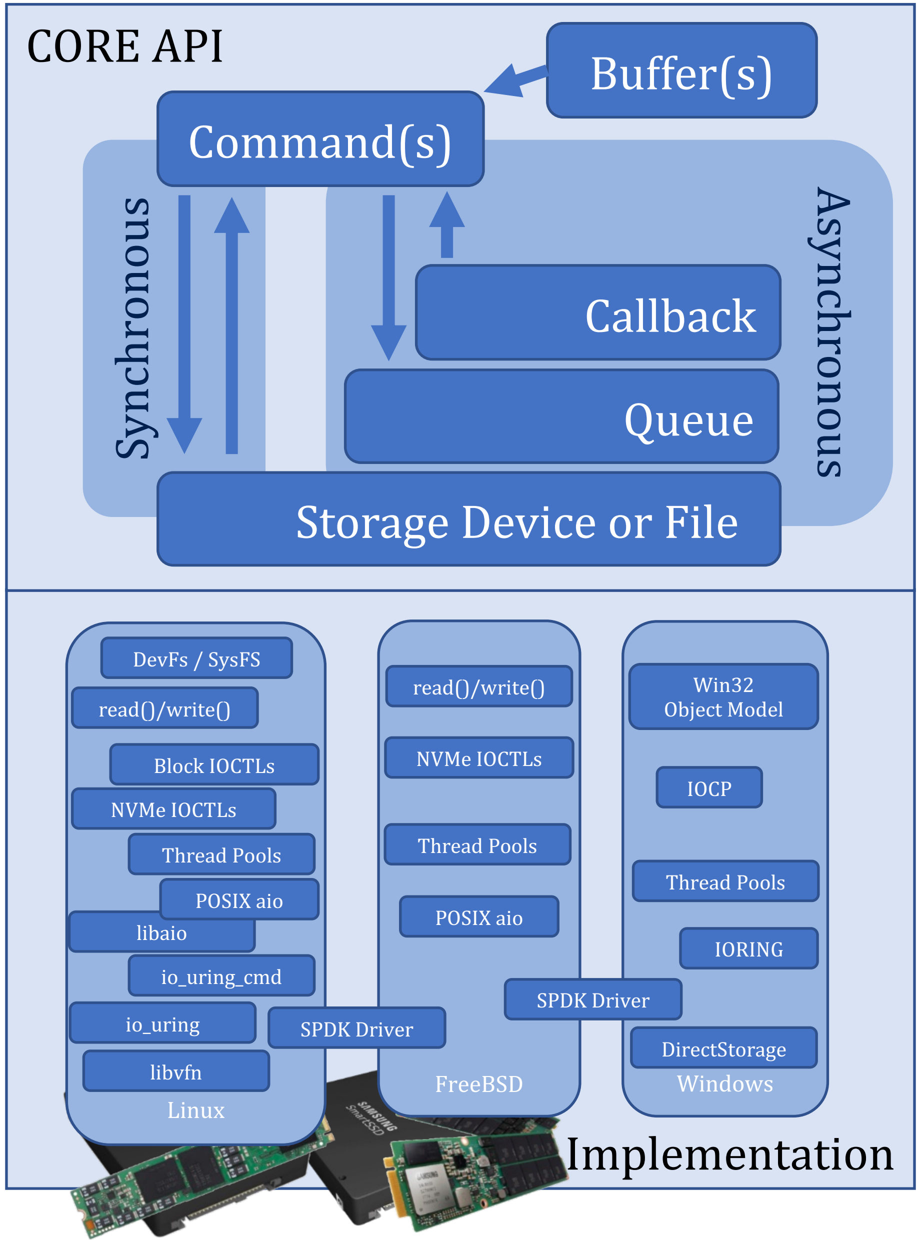
I/O Interface Independence with **xNVMe:** API

- **Buffers**



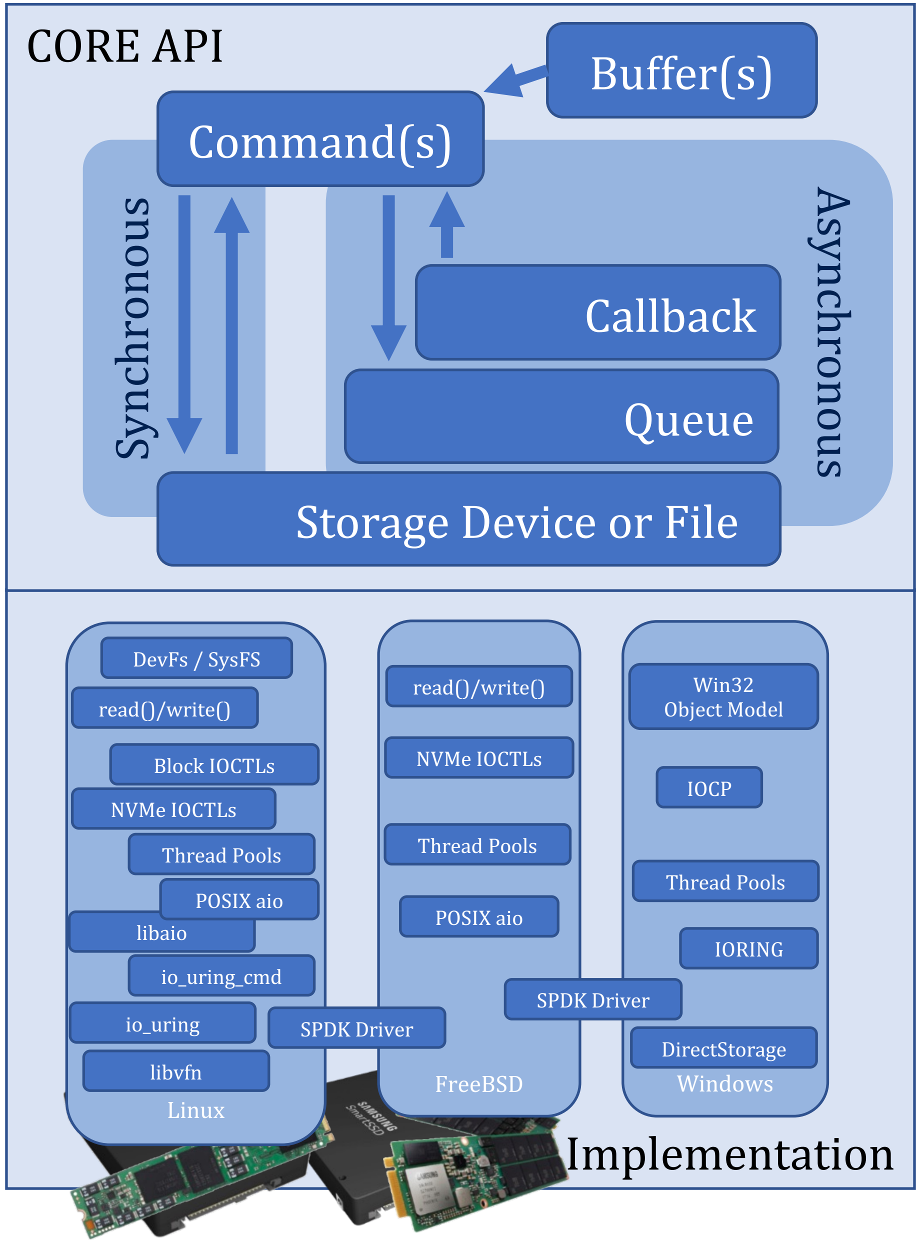
I/O Interface Independence with **xNVMe:** API

- **Buffers**
  - Contigous (`* void`)



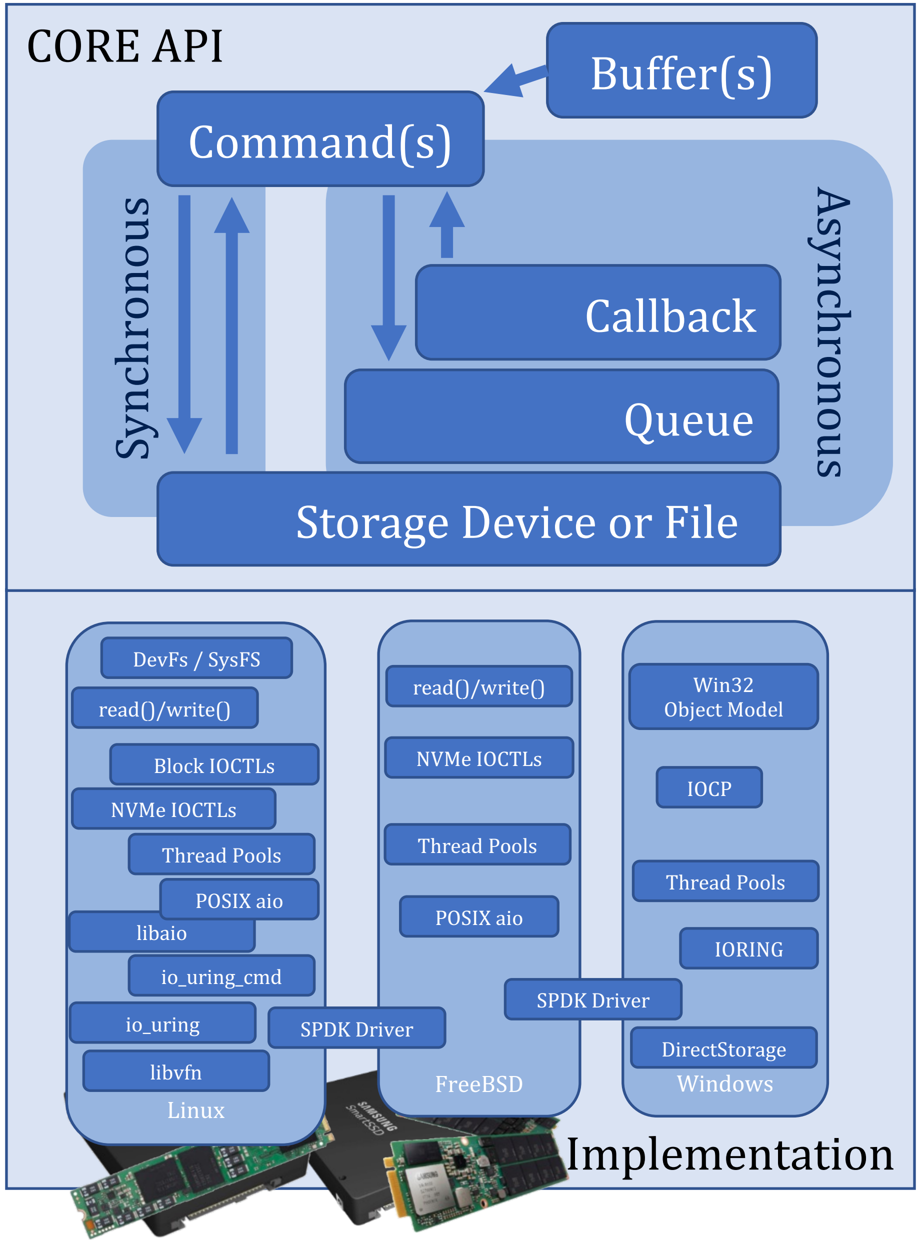
I/O Interface Independence with **xNVMe:** API

- **Buffers**
  - Contigous (`* void`)
  - Vectored (`struct iovec`)



I/O Interface Independence with **xNVMe:** API

- **Buffers**
  - Contigous (`* void`)
  - Vectored (`struct iovec`)
  - `buf = xnvme_buf_alloc(dev, nbytes)`

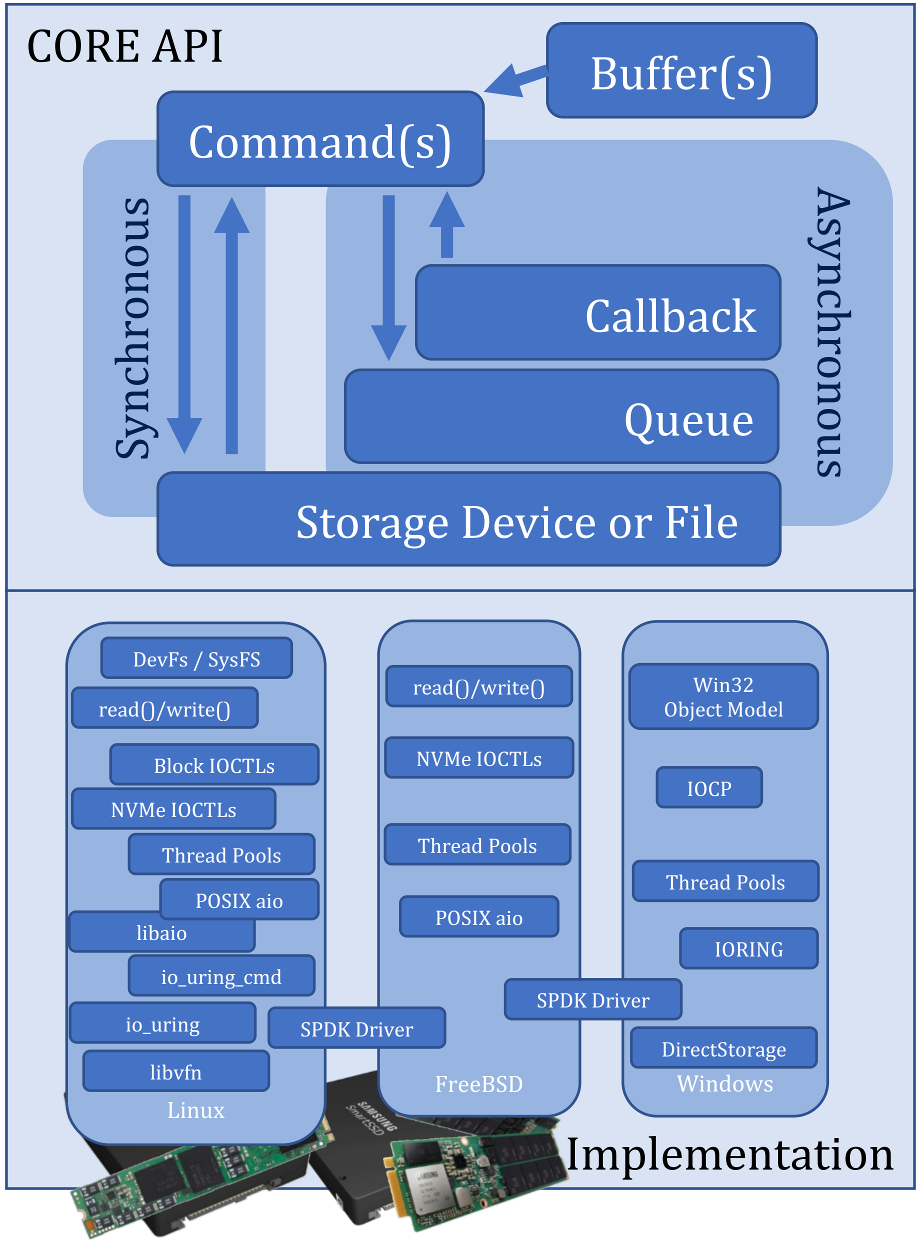


I/O Interface Independence with **xNVMe:** API

- **Buffers**
  - Contigous (`* void`)
  - Vectored (`struct iovec`)
  - `buf = xnvme_buf_alloc(dev, nbytes)`

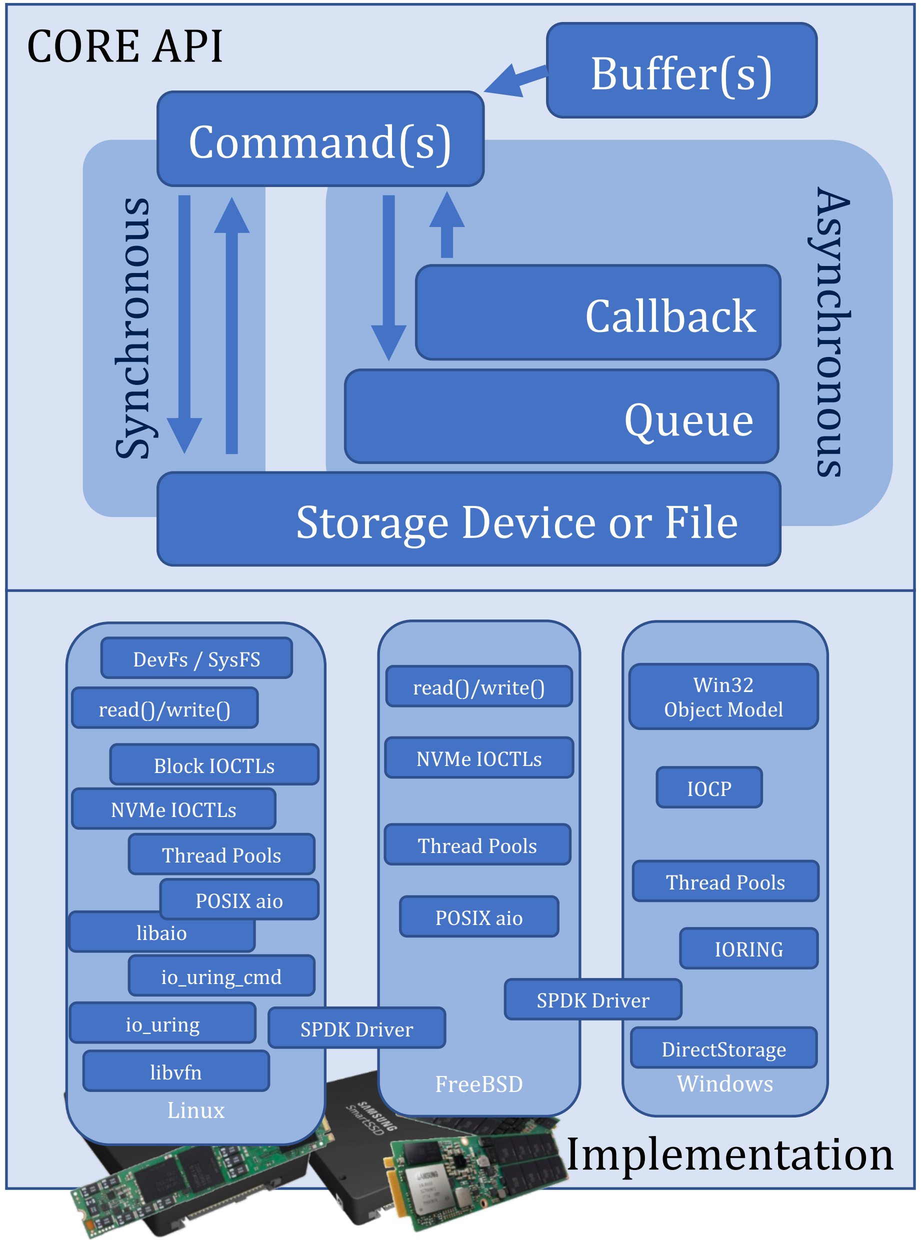
Ensure alignment constraints are met
- Page-alignment requirements for I/O interface and platform
- For I/O with given **dev**

Ensure correct memory allocator is used
- Virtual memory for OS managed
- DMA transferable for User Space NVMe Driver(s)



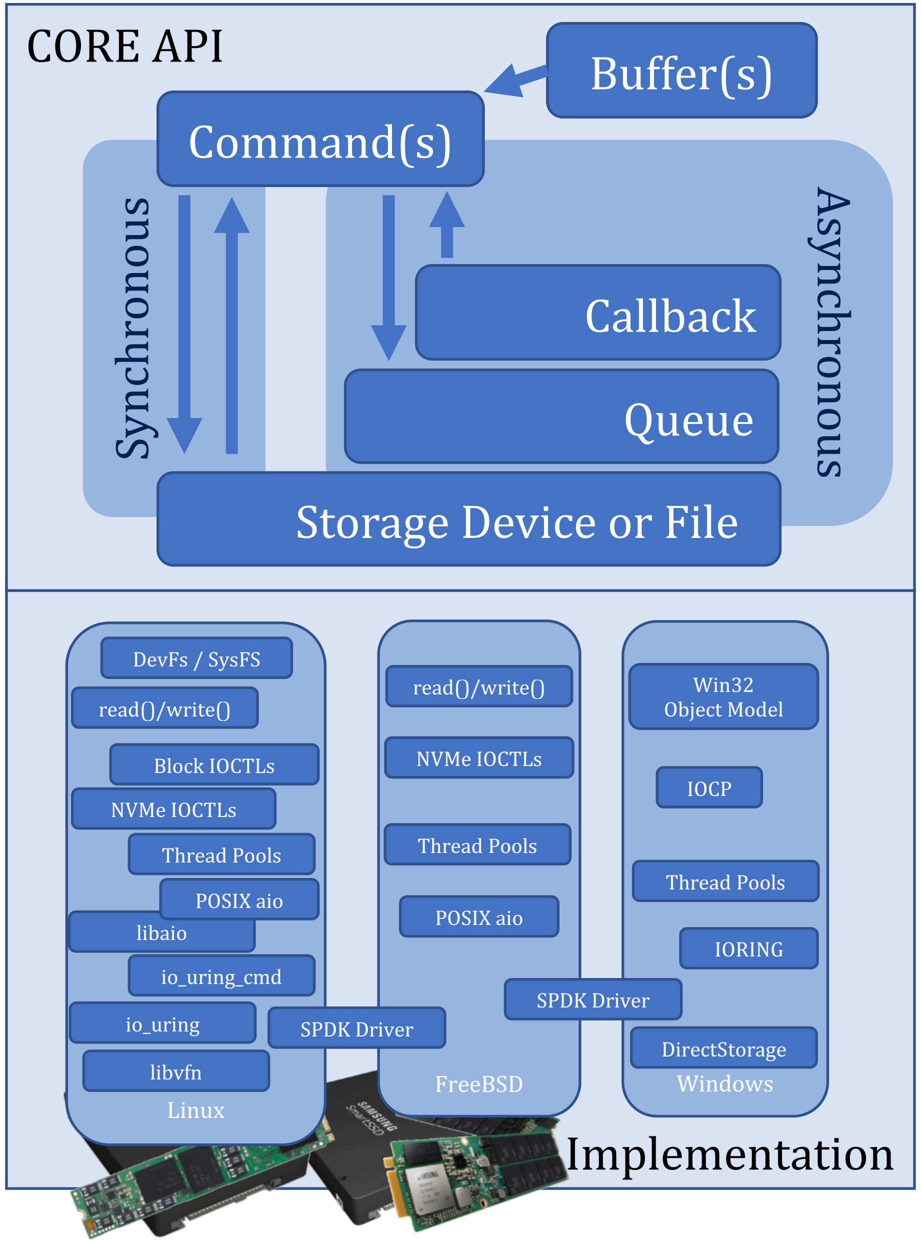
I/O Interface Independence with **xNVMe:** API

- Device Handles
- Buffers
- **Commands**
  - Synchronous
  - Asynchronous



I/O Interface Independence with **xNVMe:** API

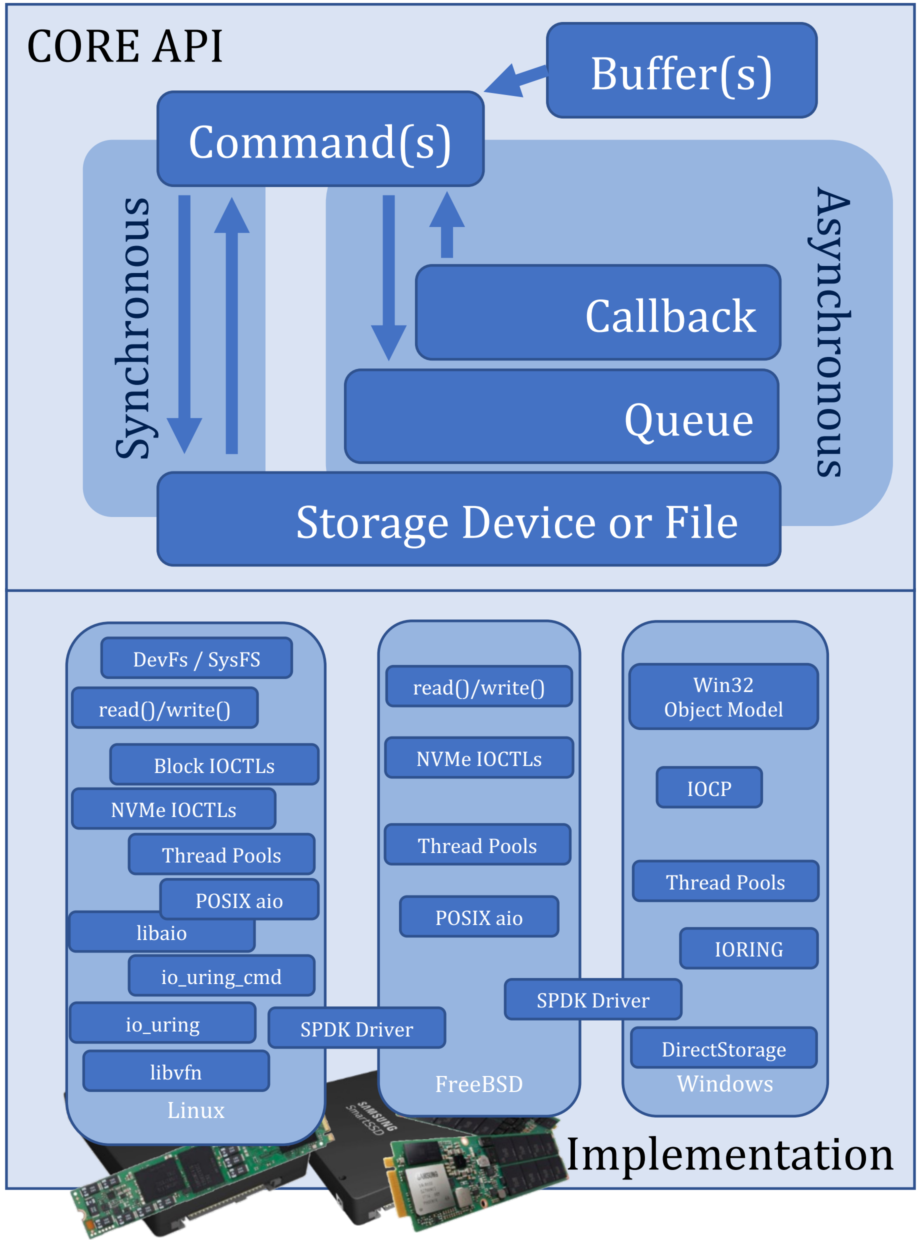
- **Commands**
  - `xnvme_cmd_passv(ctx, vec[], ...)`
  - `xnvme_cmd_pass(ctx, buf, ...)`



I/O Interface Independence with **xNVMe:** API

- **Commands**
  - `xnvme_cmd_passv(ctx, vec[], ...)`
  - `xnvme_cmd_pass(ctx, buf, ...)`

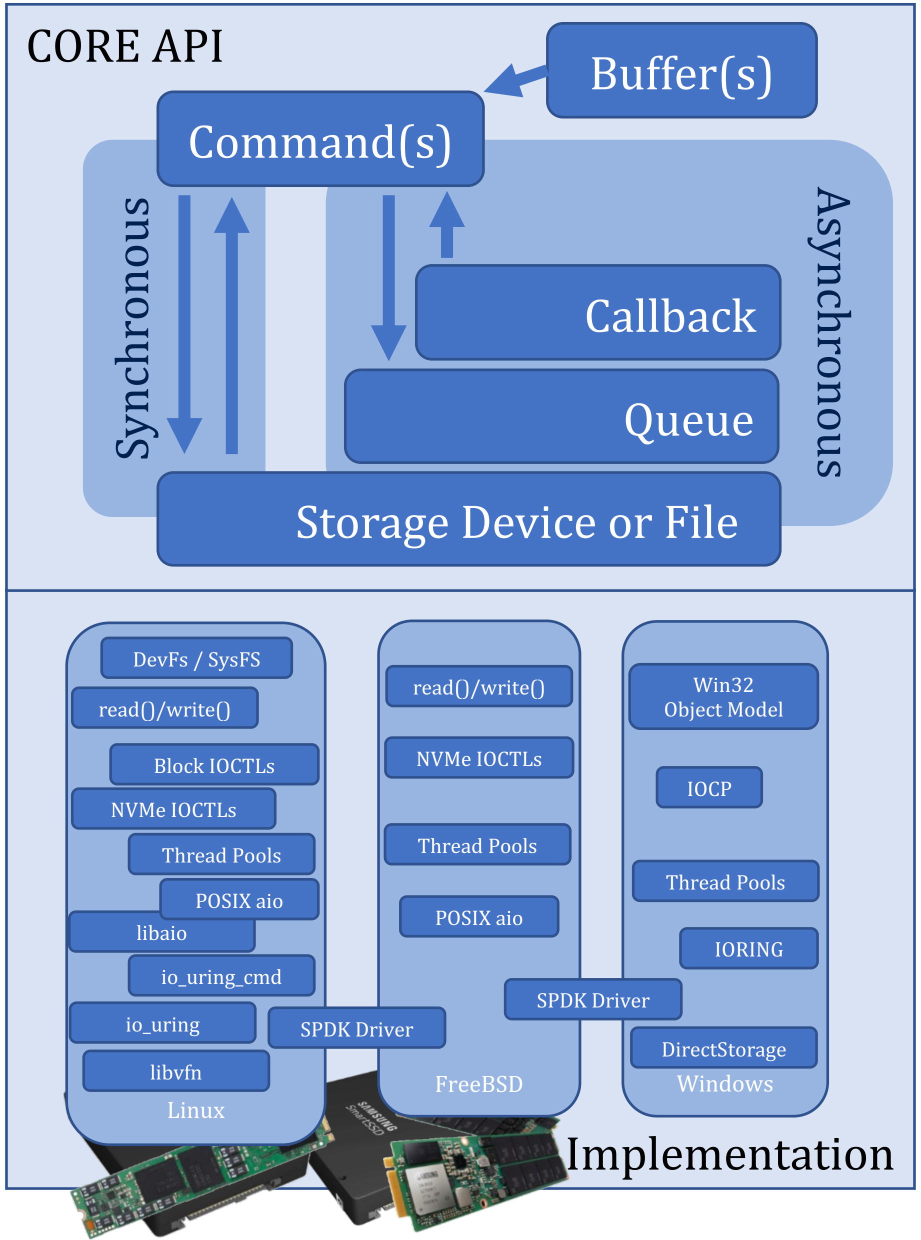
Payload description: number of iovecs, size of contig. buf, etc.



I/O Interface Independence with **xNVMe:** API

- **Commands**
  - `xnvme_cmd_passv(ctx, vec[], ...)`
  - `xnvme_cmd_pass(ctx, buf, ...)`
- Command Context
  - NVMe Command/Completion (sqe/cqe)
  - Auxilary Information (Device & I/O path)

Payload description: number of iovecs, size of contig. buf, etc.



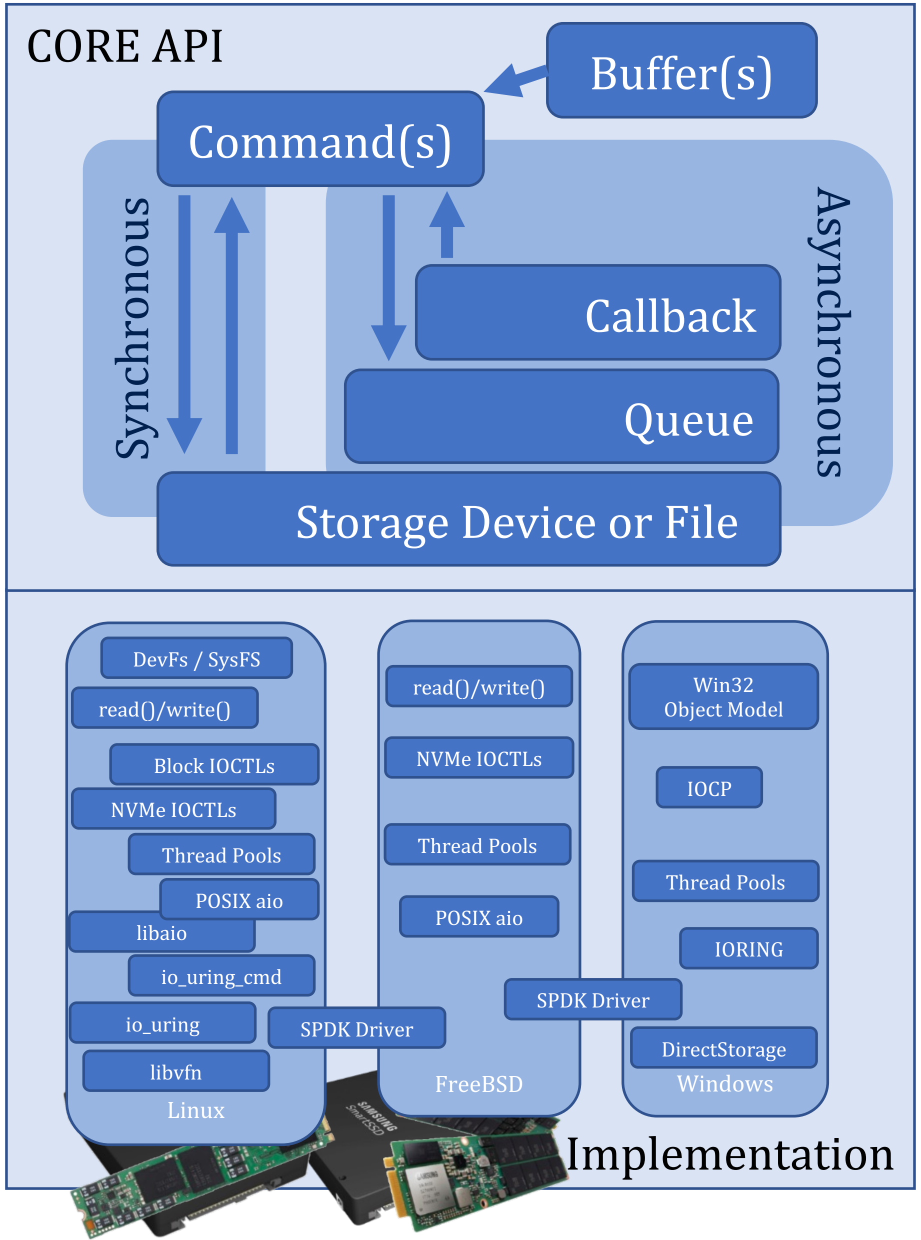
I/O Interface Independence with **xNVMe:** API

- **Commands**
  - xnvme_cmd_passv(ctx, vec[], ...)
  - xnvme_cmd_pass(ctx, buf, ...)

Payload description: number of iovecs, size of contig. buf, etc.

- Command Context
  - NVMe Command/Completion (sqe/cqe)
  - Auxilary Information (Device & I/O path)

```
int
xnvme_znd_append(struct xnvme_cmd_ctx *ctx, uint32_t nsid, uint64_t zslba,
                 uint16_t nlb, const void *dbuf, const void *mbuf)
{
        void *cdbuf = (void *)dbuf;
        void *cmbuf = (void *)mbuf;

        size_t dbuf_nbytes = cdbuf ? ctx->dev->geo.lba_nbytes * (nlb + 1) : 0;
        size_t mbuf_nbytes = cmbuf ? ctx->dev->geo.nbytes_oob * (nlb + 1) : 0;

        ctx->cmd.common.opcode = XNVME_SPEC_ZND_OPC_APPEND;
        ctx->cmd.common.nsid = nsid;
        ctx->cmd.znd.append.zslba = zslba;
        ctx->cmd.znd.append.nlb = nlb;

        return xnvme_cmd_pass(ctx, cdbuf, dbuf_nbytes, cmbuf, mbuf_nbytes);
}
```

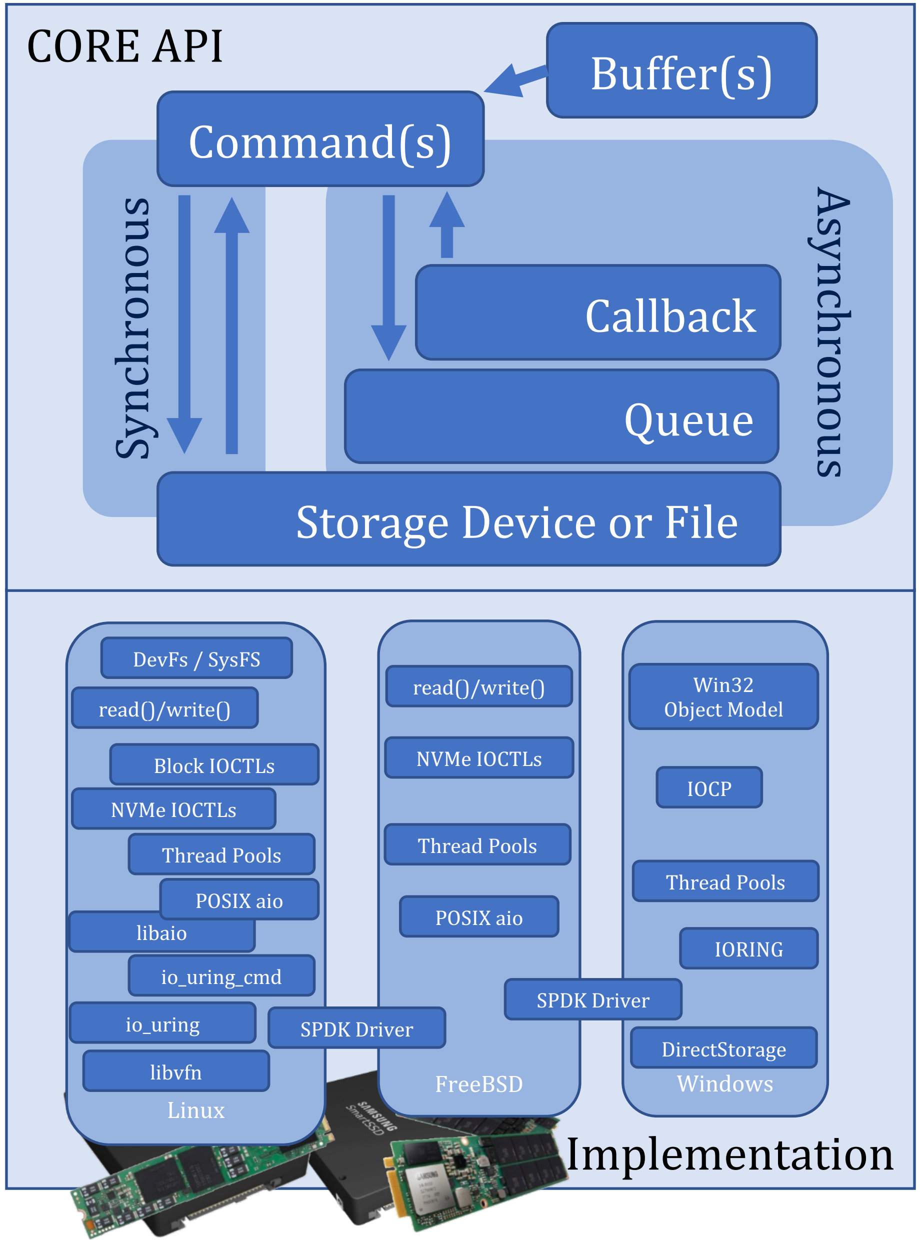


I/O Interface Independence with **xNVMe:** API

Payload description: number of iovecs, size of contig. buf, etc.

- **Commands**
  - `xnvme_cmd_passv(ctx, vec[], ...)`
  - `xnvme_cmd_pass(ctx, buf, ...)`
- Command Context
  - NVMe Command/Completion (sqe/cqe)
  - Auxilary Information (Device & I/O path)
- **Synchronous**

```
ctx = xnvme_cmd_ctx_from_dev(dev)
  ... setup ctx.cmd (sqe) ...
```

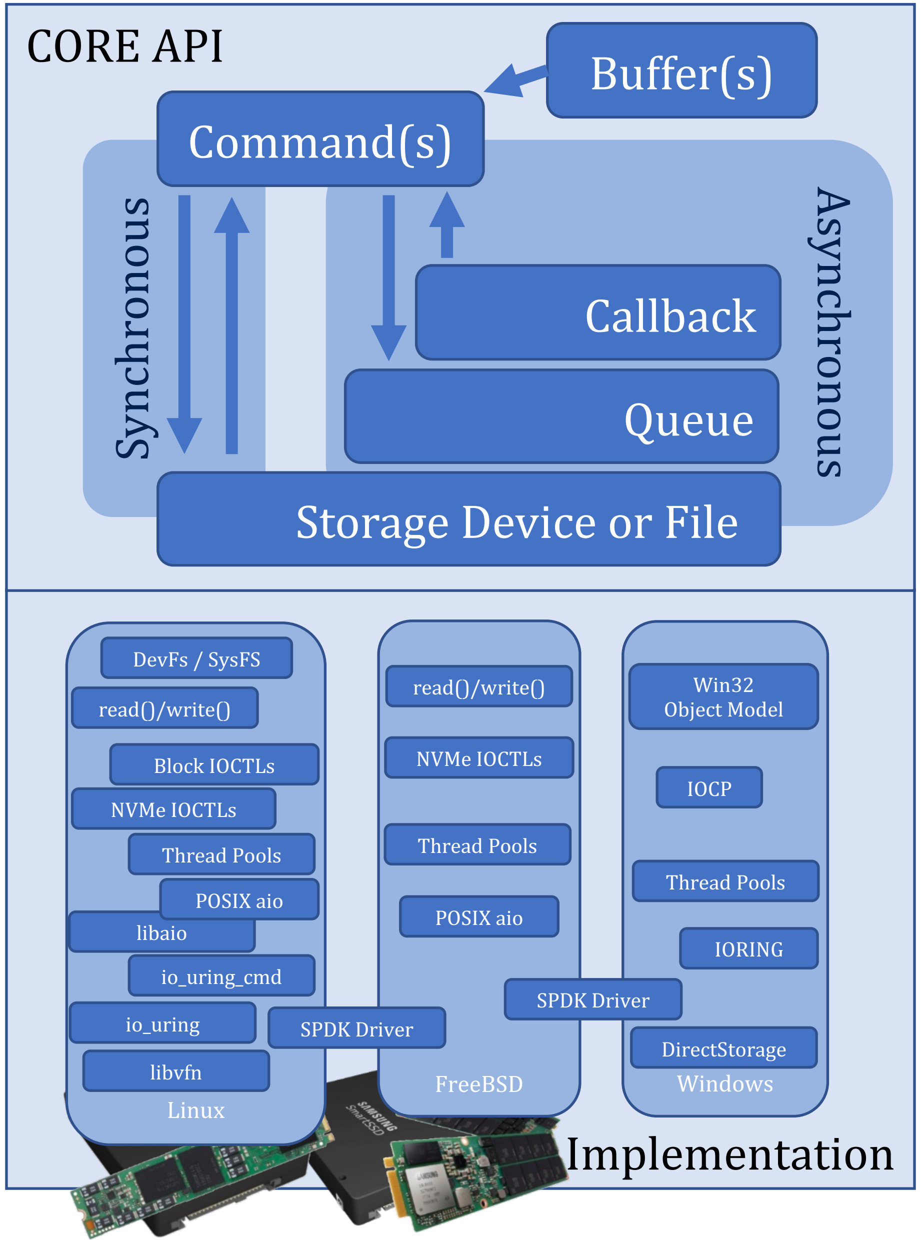


I/O Interface Independence with **xNVMe:** API

Payload description: number of iovecs, size of contig. buf, etc.

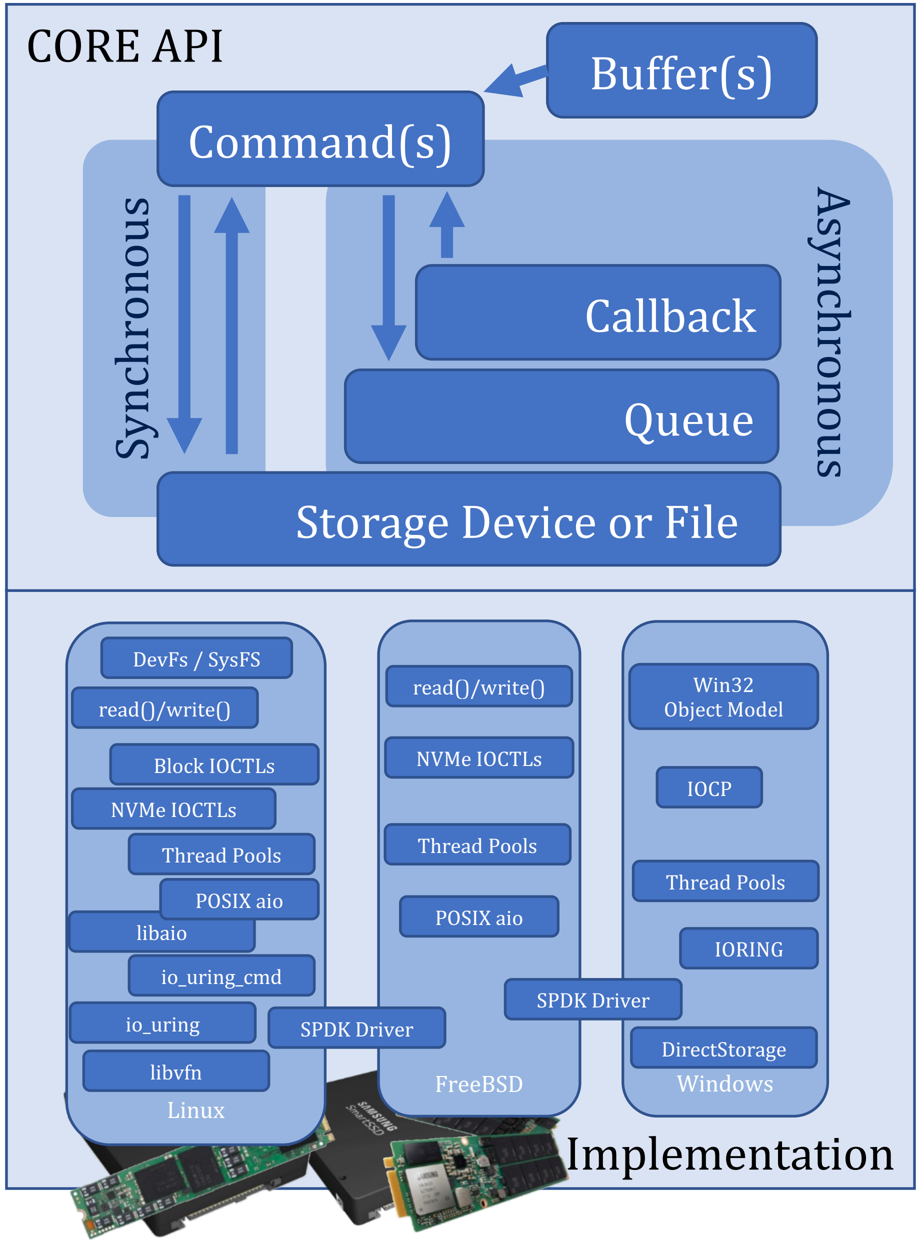
- **Commands**
  - `xnvme_cmd_passv(ctx, vec[], ...)`
  - `xnvme_cmd_pass(ctx, buf, ...)`
- Command Context
  - NVMe Command/Completion (sqe/cqe)
  - Auxilary Information (Device & I/O path)
- **Synchronous**

```
ctx = xnvme_cmd_ctx_from_dev(dev)
  ... setup ctx.cmd (sqe) ...
xnvme_cmd_pass(ctx, buf, ...)
  ... inspect ctx.cpl (cqe) ...
```



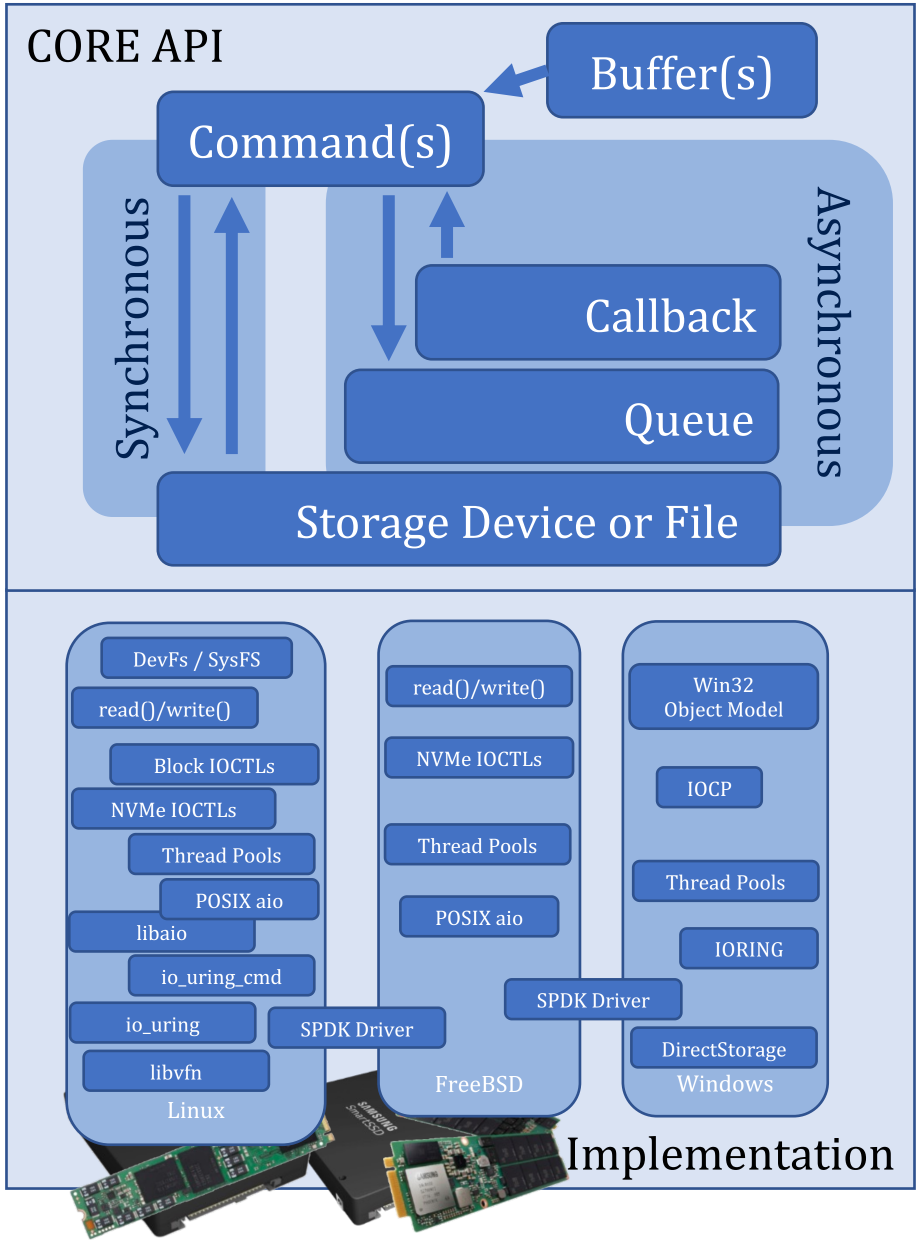
I/O Interface Independence with **xNVMe:** API

- Device Handles
- Buffers
- Commands
  - **Synchronous**
  - Asynchronous



I/O Interface Independence with **xNVMe:** API

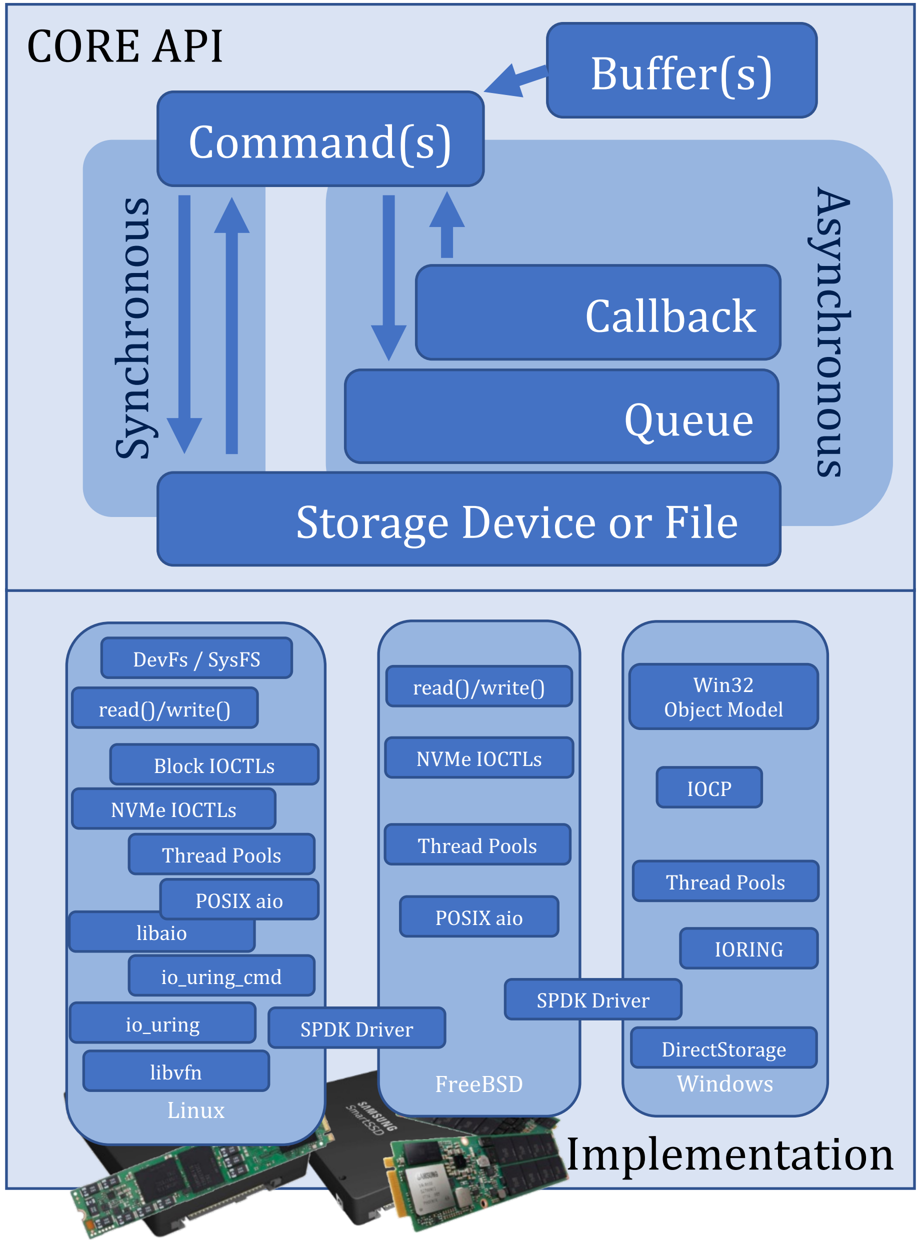
- Device Handles
- Buffers
- Commands
  - Synchronous
  - **Asynchronous**



I/O Interface Independence with **xNVMe:** API

- **Asynchronous**

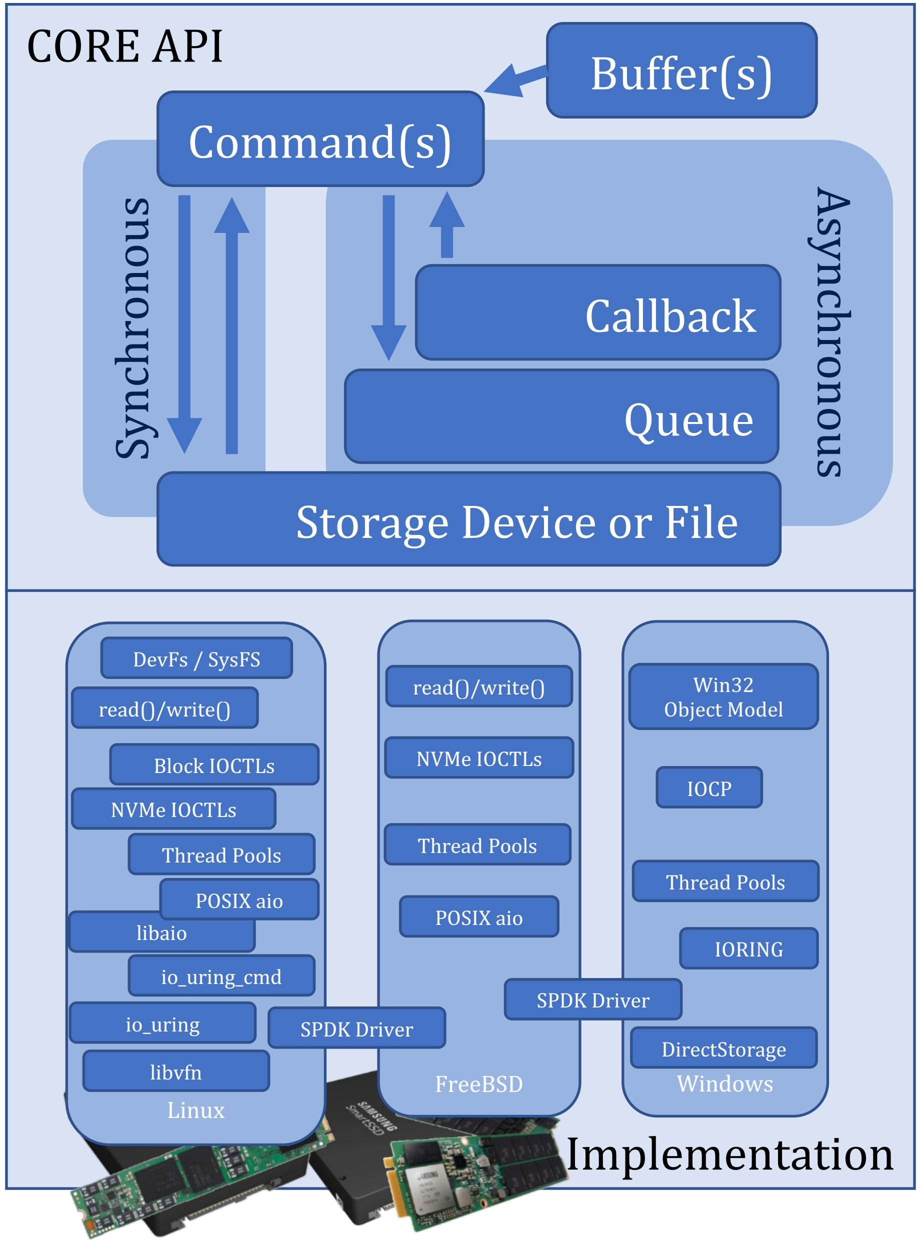
  `xnvme_queue_init(`**`dev`**`, cap, **`**`q`**`, ..)`



I/O Interface Independence with **xNVMe:** API

- **Asynchronous**

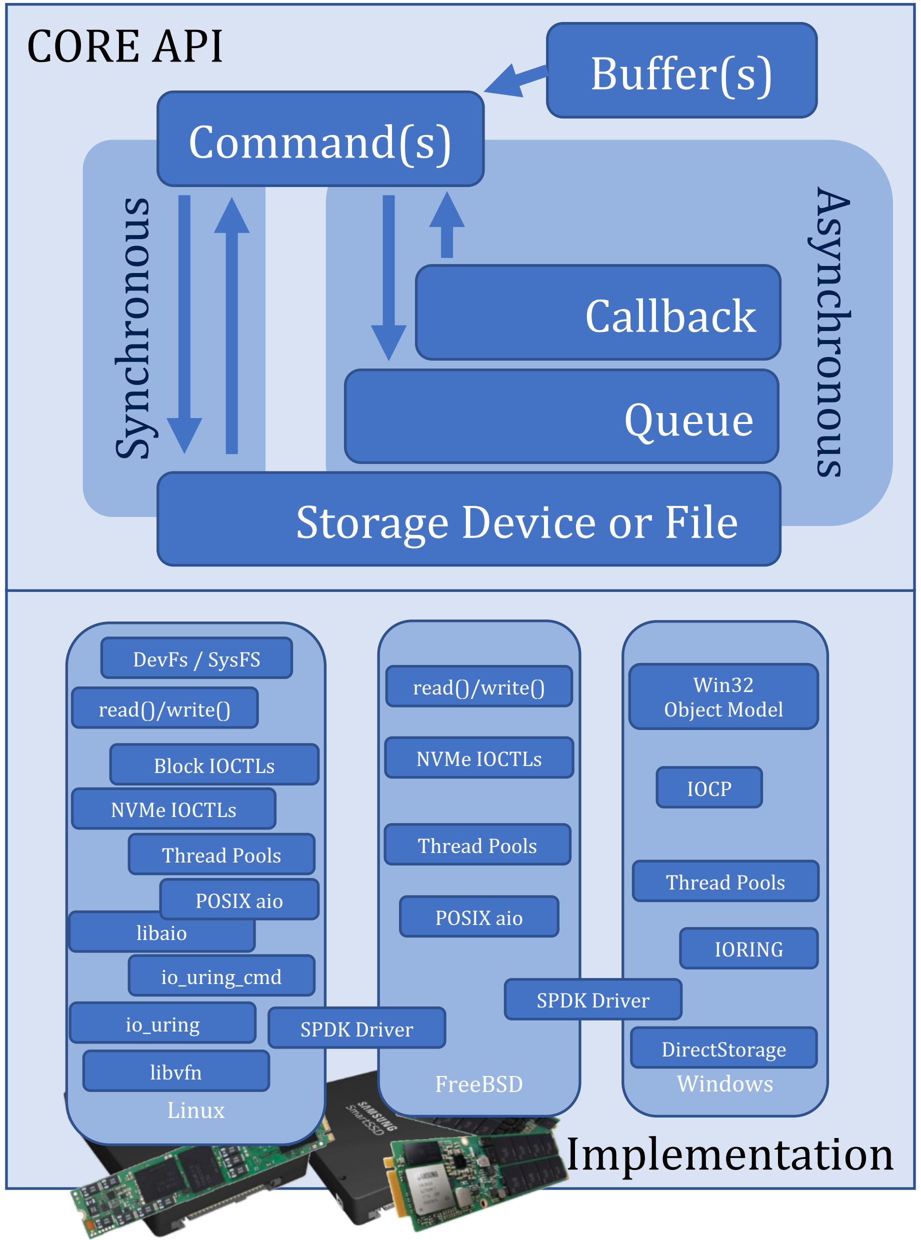
```
xnvme_queue_init(dev, cap, **q, ..)

ctx = xnvme_cmd_ctx_from_queue(q)
  ... setup ctx.cmd (sqe) ...
xnvme_cmd_pass(ctx, buf, ...)
```



I/O Interface Independence with **xNVMe:** API

- **Asynchronous**

```
xnvme_queue_init(dev, cap, **q, ..)

ctx = xnvme_cmd_ctx_from_queue(q)
  ... setup ctx.cmd (sqe) ...
xnvme_cmd_pass(ctx, buf, ...)

xnvme_queue_poke(q, max)
xnvme_queue_drain(q)
```



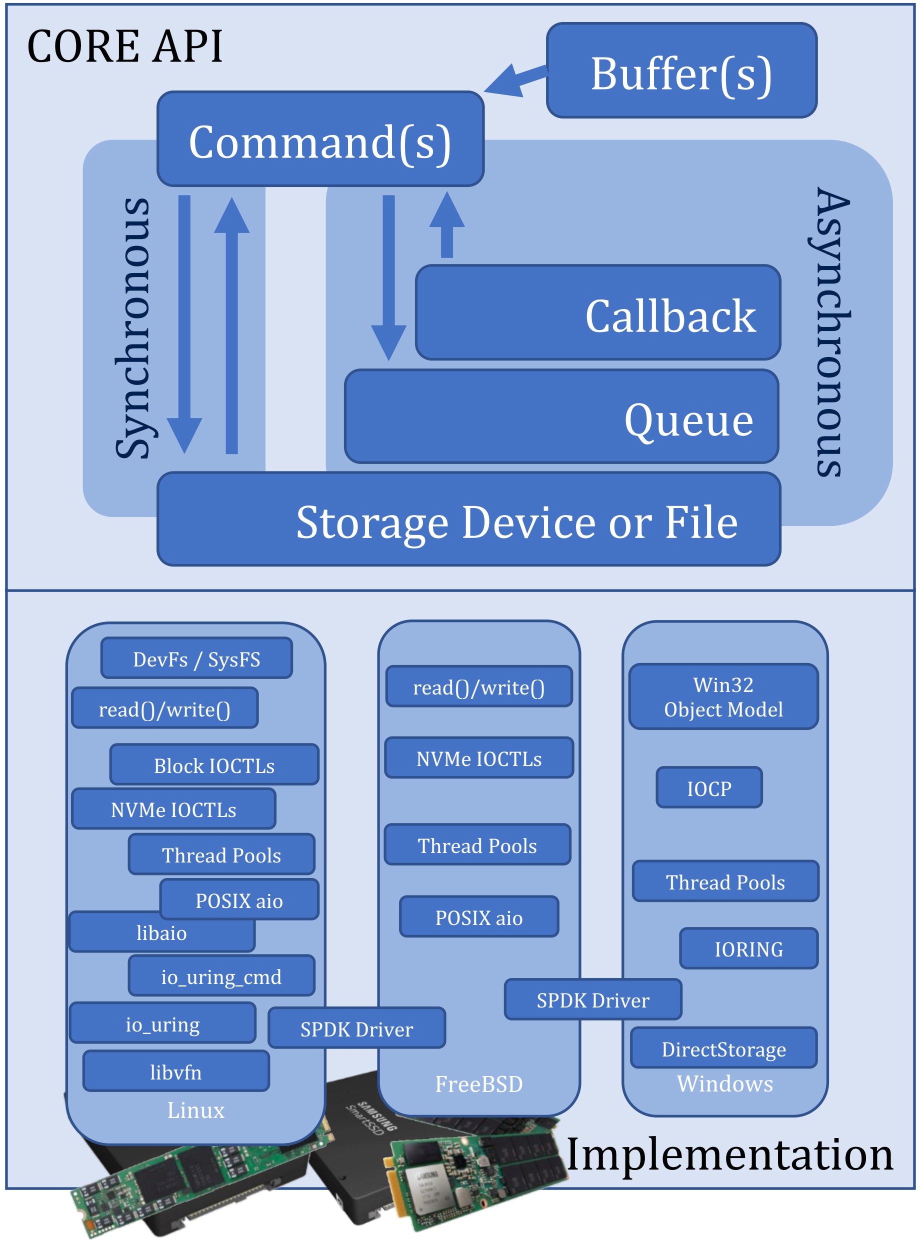
I/O Interface Independence with **xNVMe:** API

- **Asynchronous**

```
xnvme_queue_init(dev, cap, **q, ..)

ctx = xnvme_cmd_ctx_from_queue(q)
  ... setup ctx.cmd (sqe) ...
xnvme_cmd_pass(ctx, buf, ...)
```

Process at most **max** completions

```
xnvme_queue_poke(q, max)
```



I/O Interface Independence with **xNVMe:** API

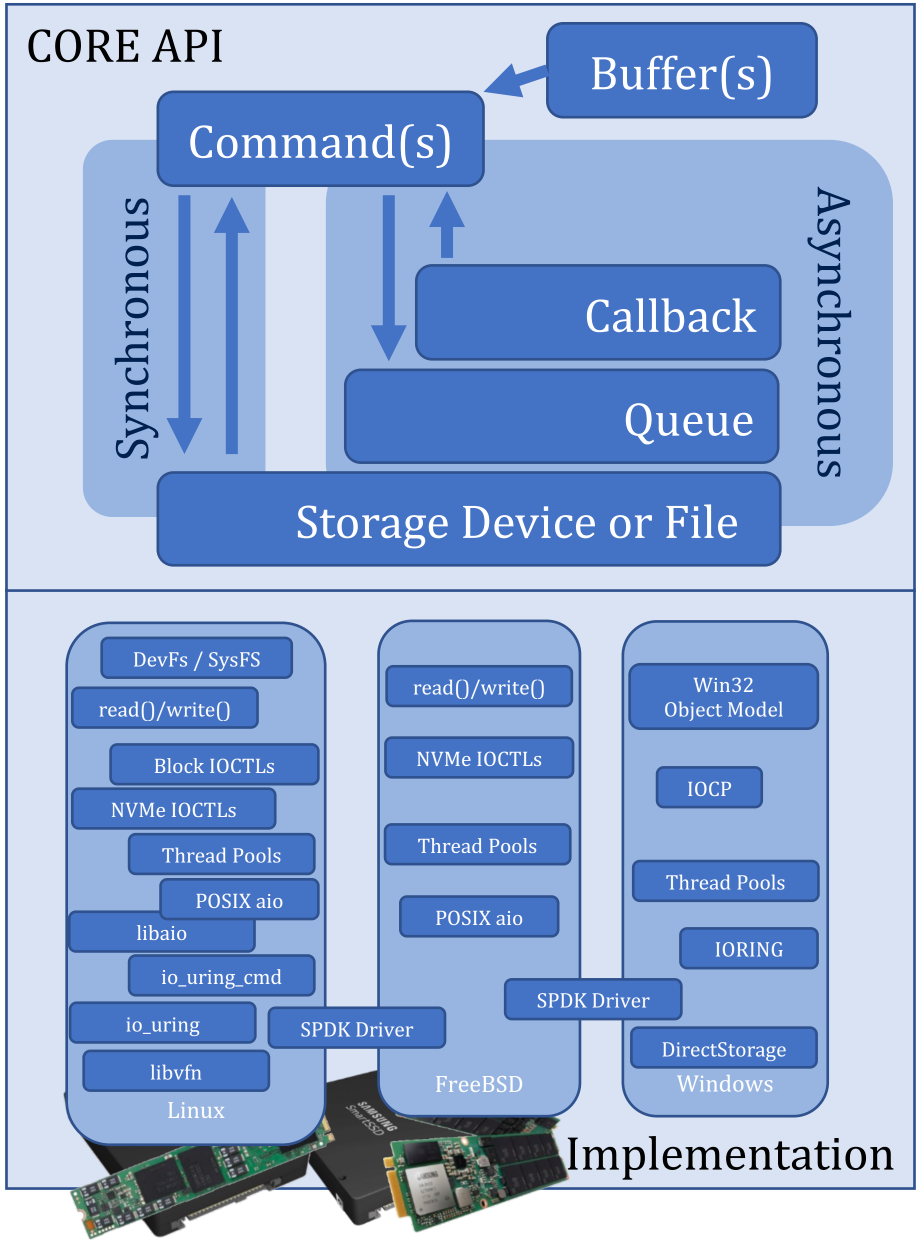
- **Asynchronous**

```
xnvme_queue_init(dev, cap, **q, ..)

ctx = xnvme_cmd_ctx_from_queue(q)
  ... setup ctx.cmd (sqe) ...
xnvme_cmd_pass(ctx, buf, ...)
```

Process at most **max** completions

```
xnvme_queue_poke(q, max)
```

```
  ctx.callback(ctx, ctx.args)
    ... inspect ctx.cpl (cqe) ...
```



I/O Interface Independence with **xNVMe:** API

- **Asynchronous**

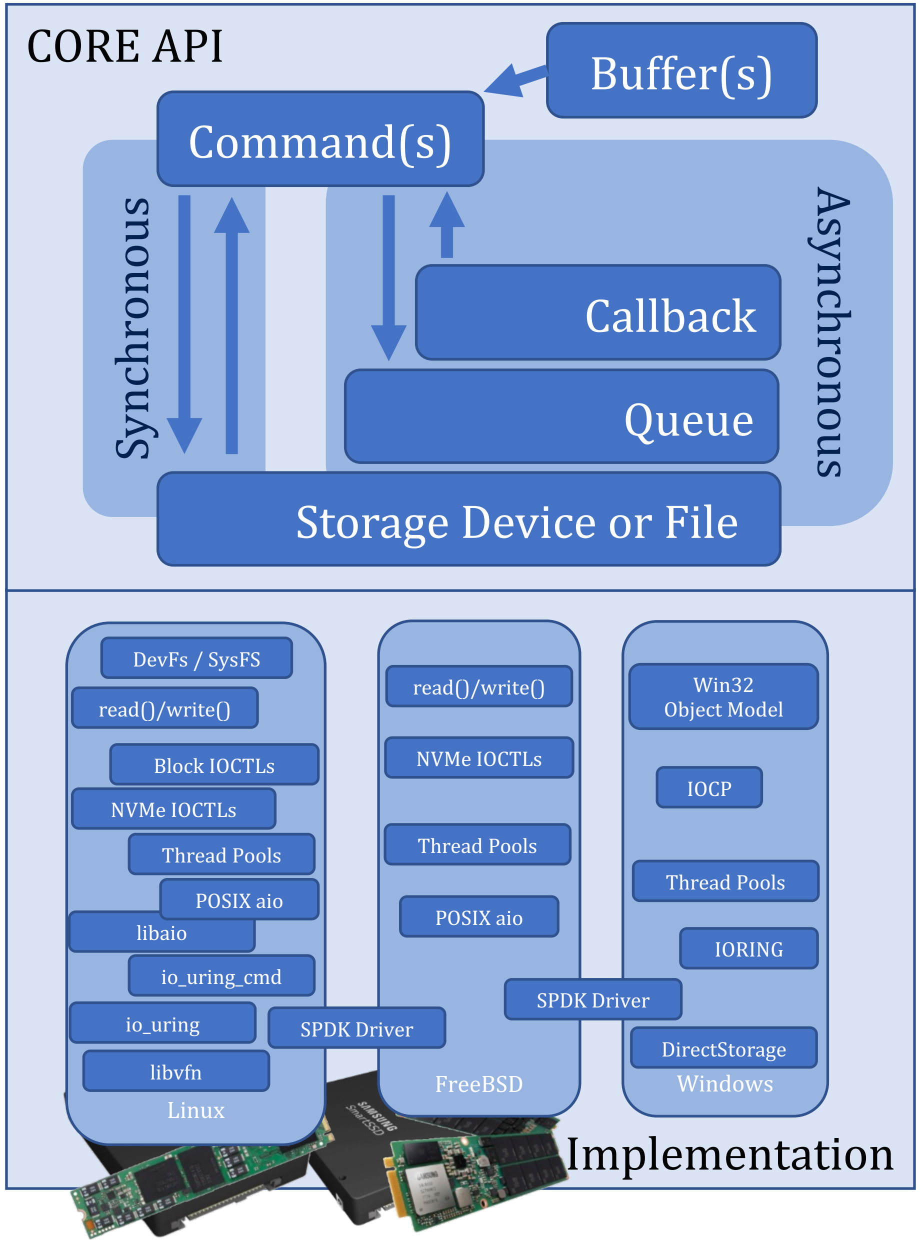
```
xnvme_queue_init(dev, cap, **q, ..)

ctx = xnvme_cmd_ctx_from_queue(q)
  ... setup ctx.cmd (sqe) ...
xnvme_cmd_pass(ctx, buf, ...)
```

Process at most **max** completions

```
xnvme_queue_poke(q, max)
xnvme_queue_drain(q)
```

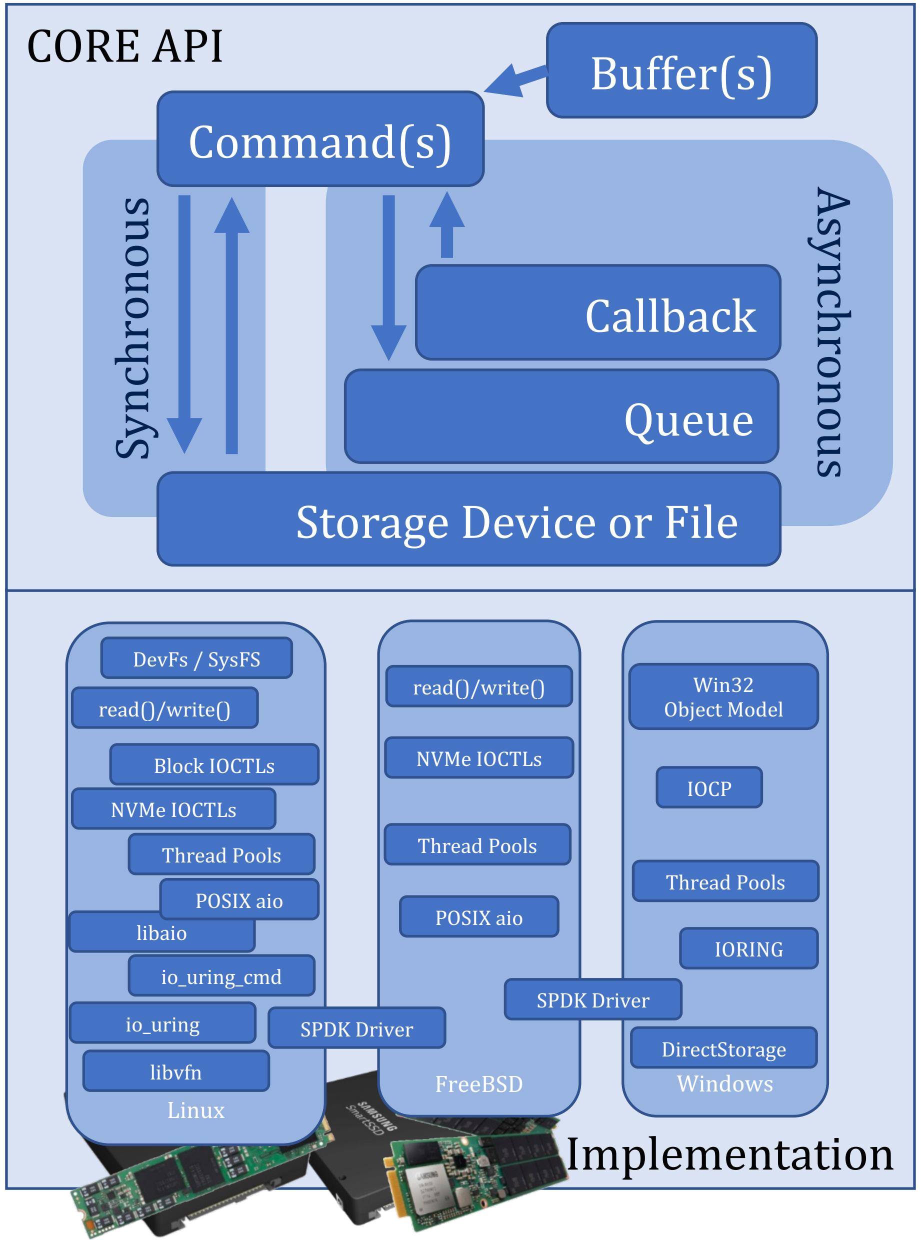
Process completions until queue is **empty**

```
   ctx.callback(ctx, ctx.args)
     ... inspect ctx.cpl (cqe) ...
```



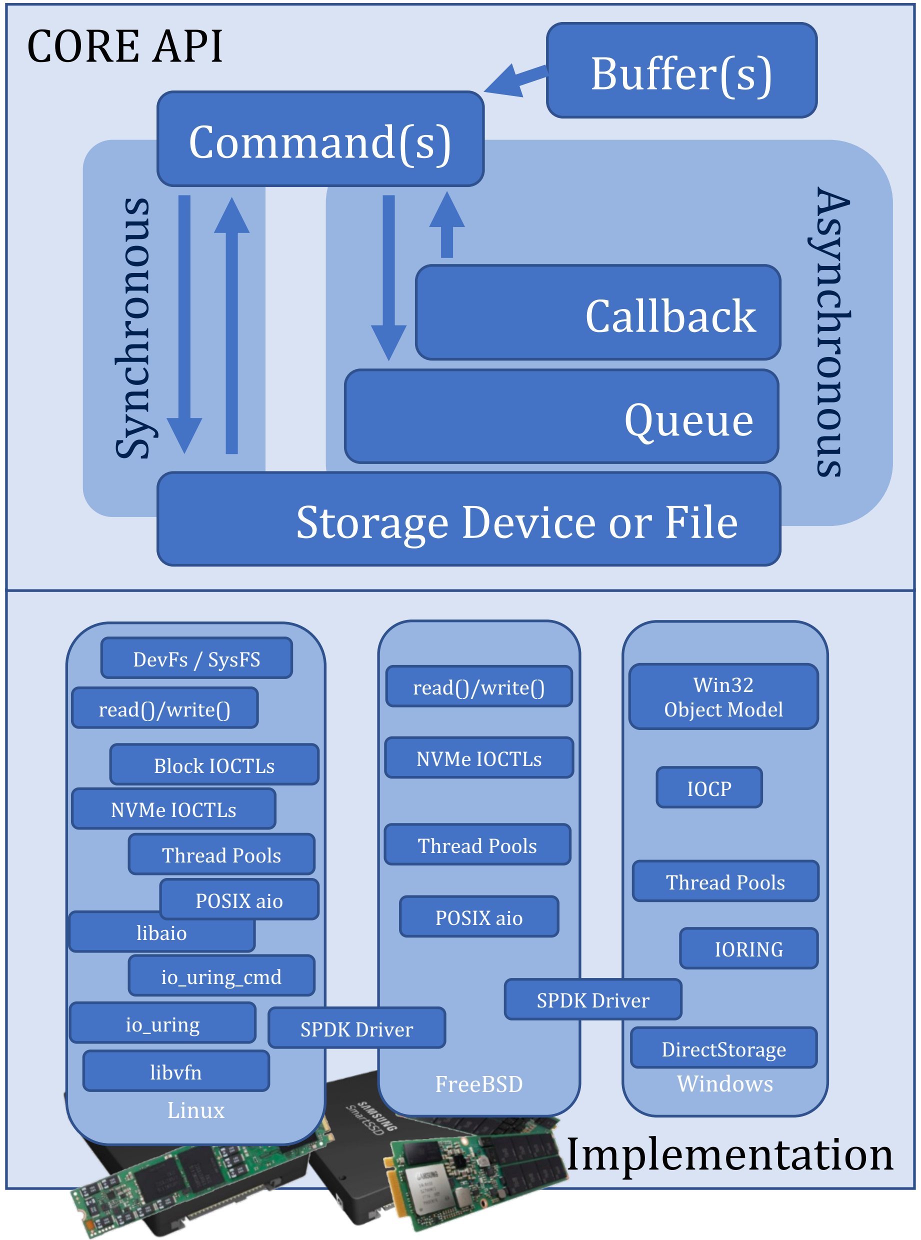
I/O Interface Independence with **xNVMe:** API

- Device Handles
- Buffers
- Commands
  - Synchronous
  - **Asynchronous**



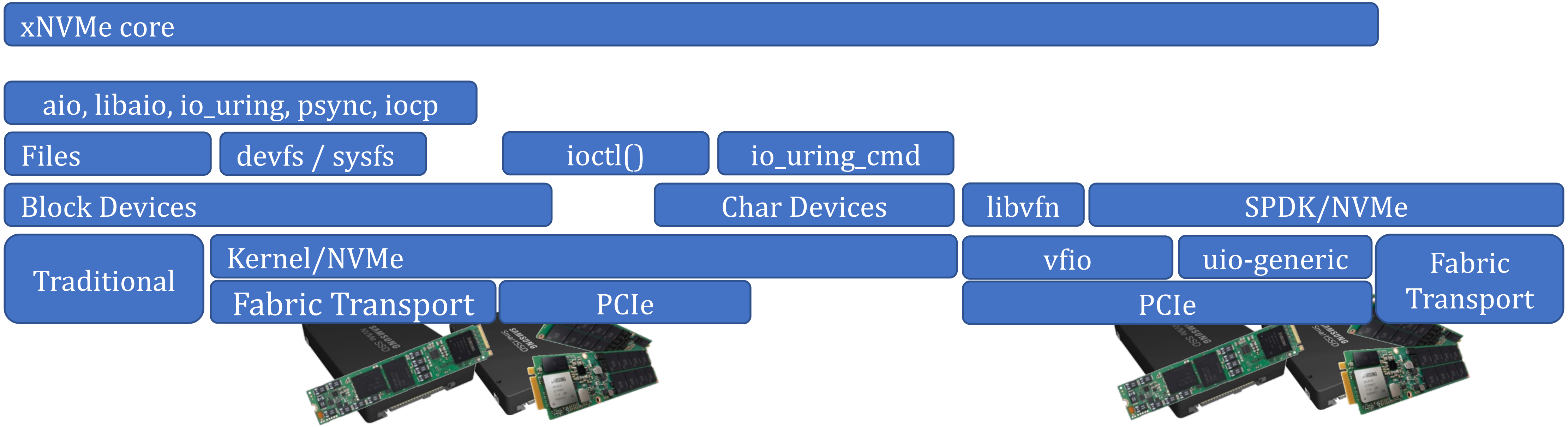
I/O Interface Independence with **xNVMe:** API

- Device Handles
- Buffers
- Commands
  - Synchronous
  - Asynchronous
- **For details, docs are available**
  - C API
    https://xnvme.io/docs/latest/capis/
  - C API Examples
    https://xnvme.io/docs/latest/examples/



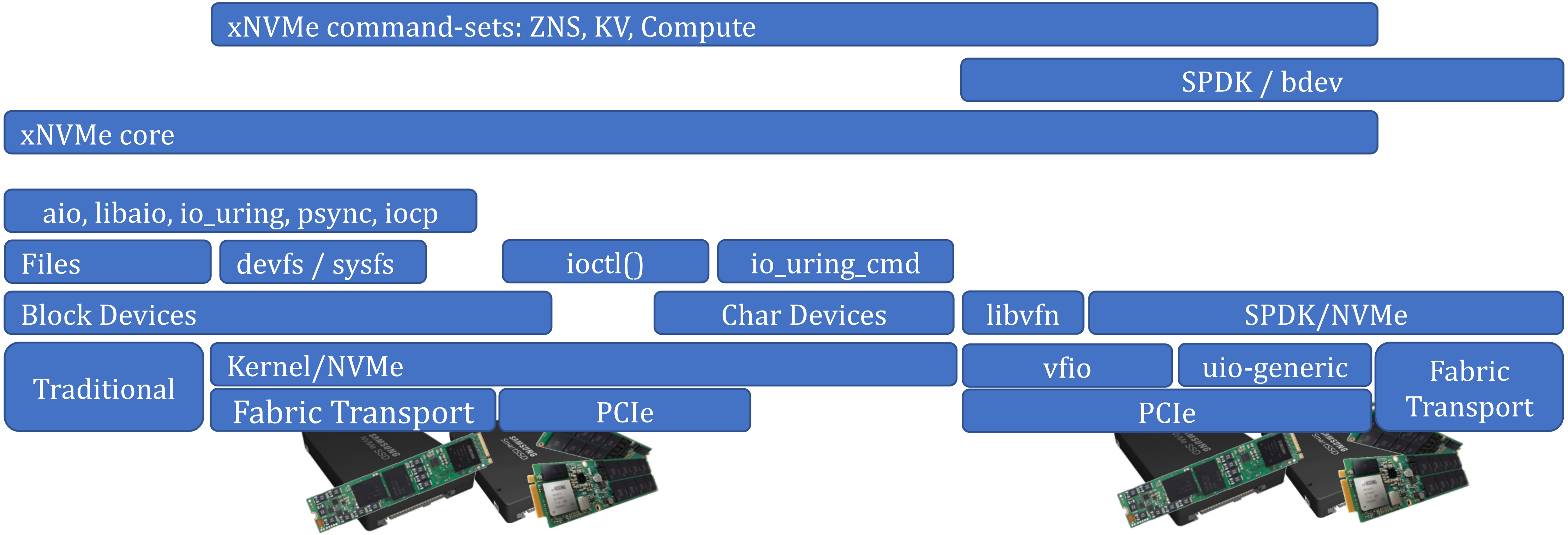
I/O Interface Independence with **xNVMe**

- A minimal **encapsulation** of system-interfaces and user-space drivers into a **unified** API for device handles, buffers, commands and their submission in **synchronous** and **asynchronous** mode



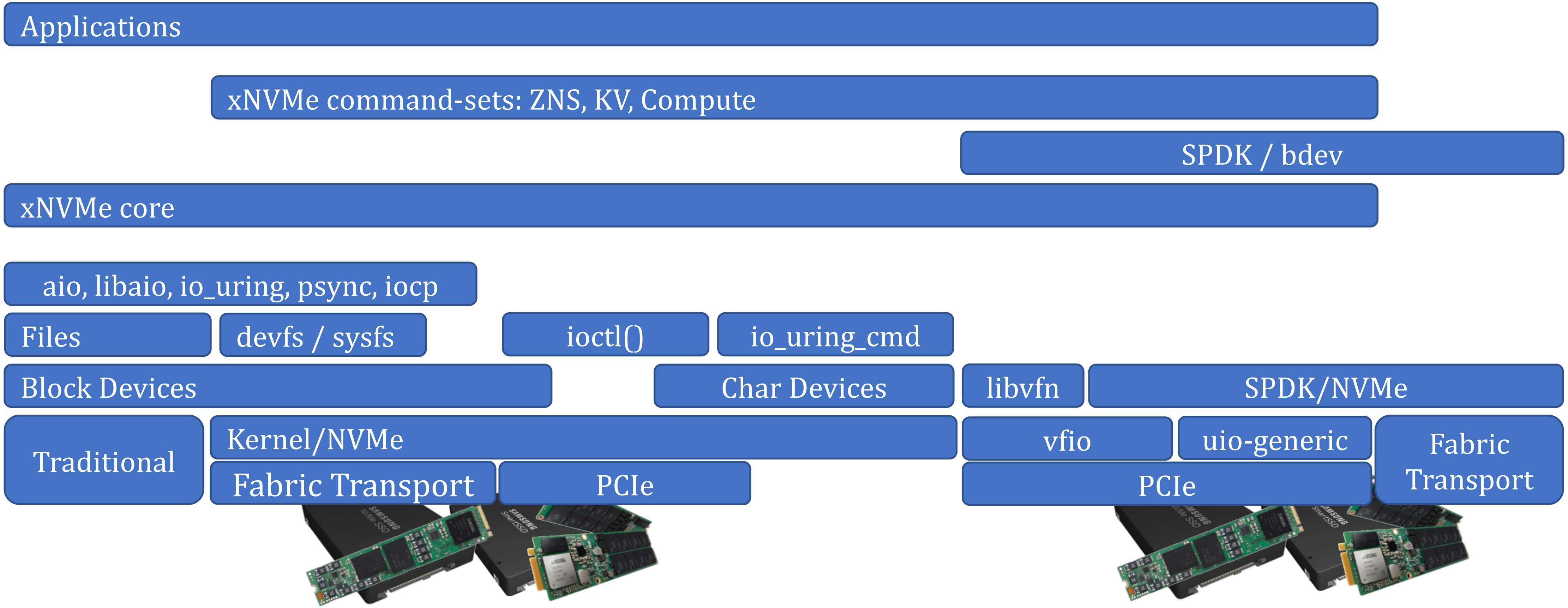
I/O Interface Independence with **xNVMe**

- **Extensibility**: a single, simple command construction



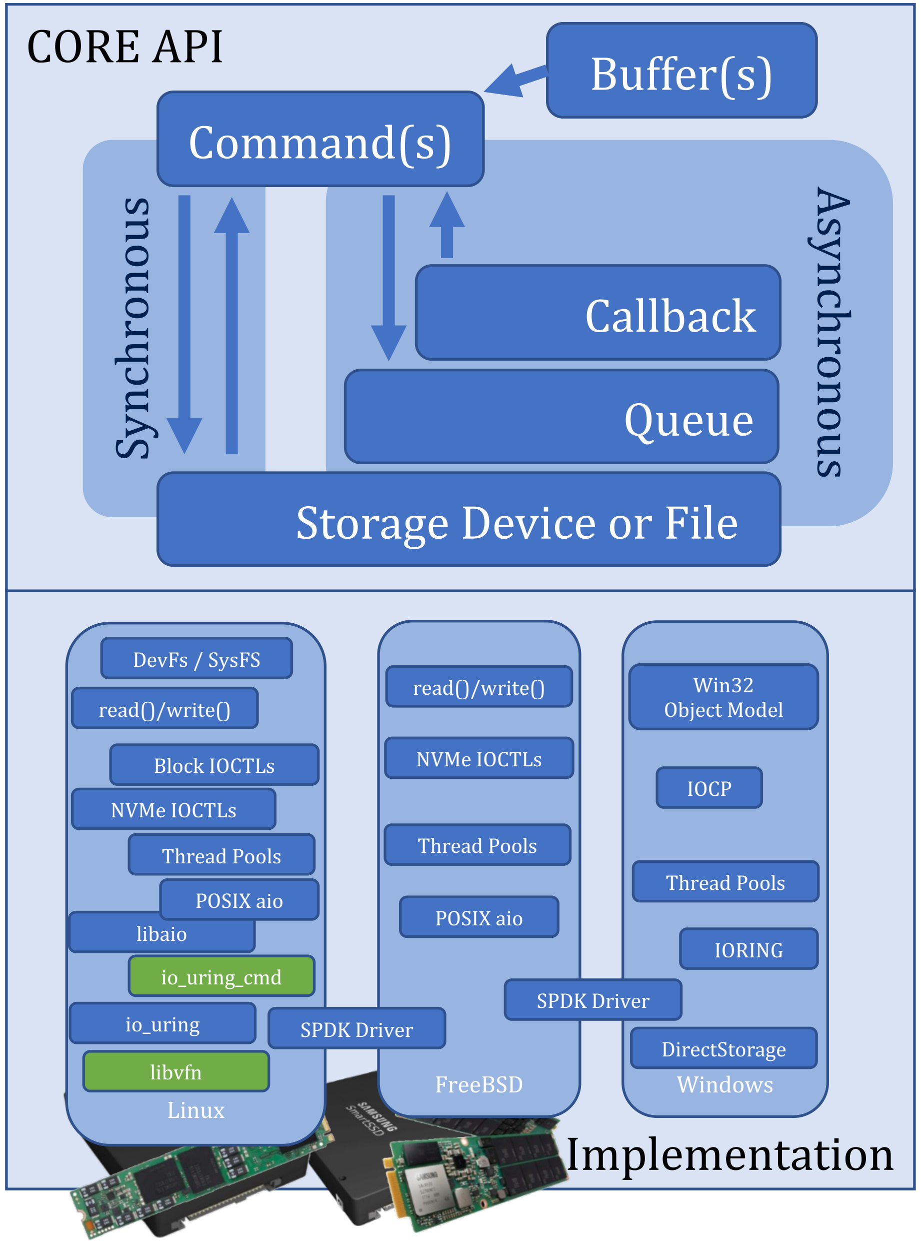
I/O Interface Independence with **xNVMe**

- **Extensibility**: a single, simple command construction
- **Applications**: use command-set helpers or directly to the core



Extensibility: a recent example

- Support for Linux async. NVMe Passthru
  - Aka **io_uring_cmd** / async. ioctl()
- Linux Changes
  - Generic namespace char-devices **/dev/ng0n1** 5.15
  - Extension of **io_uring** big-sqe & big-cqe
  - **NVMe sqe/cqe** embedded in **ring-sqe/cqe** 5.18
  - Non-NVM Command-sets ➔ **efficiently**
- xNVMe
  - System interface handled by library backend
  - **No** changes to CORE API
  - **No** changes to upper-layers
  - **No** changes to the **application**



Performance Evaluation

1. xNVMe Abstraction Layer **Overhead**
   ➔Per command latency increase

2. xNVMe Abstraction Layer **Efficiency**
   ➔ Peak IOPS on a single physical **CPU**

Relative to reference implementations in **fio** and **SPDK / bdevperf**

Performance Evaluation: abstraction **overhead**
xNVMe in **FIO**

Comparing **io-engine** implementations using **`fio`**

Linux + FreeBSD POSIX aio vs xnvme (be_async=posix)
Linux aio vs xnvme (be_async=libaio)
Linux io_uring vs xnvme (be_async=io_uring)
SPDK/NVMe vs vs xnvme (be_async=spdk)
Windows IOCP vs xnvme (be_async=iocp)

Measure: per command latency **delta**

Performance Evaluation: framework

- Quantify performance penalty of xNVMe
  1. **Baseline** overhead; non-I/O interface and non-device specific
  2. For each I/O **Interface** compare overhead using an NVMe device
  3. **Scalability**; for each I/O interface using an NVMe device: verify that the overhead remains constant when scaling up I/O payload size and queue-pressure

Performance Evaluation: framework

- Quantify performance penalty of xNVMe
- Commodity hardware for **reproducibility**

Hardware	Model
CPU	Intel Core i5-9400 2.9Ghz
Memory	Corsair 2x 16GB DDR4 3200Mhz CL18
Board	MSI MPG Z390I Gaming Edge AC
SSD	Intel Optane Memory M10 Series (MEMPEK1J016GAL)
Software	**Model**
FreeBSD	Version 12.1
fio	Version 3.27
gcc	Version 10.2.1
clang	Version 12.0.1
SPDK	Version 21.04
xNVMe	Version 0.0.26

Performance Evaluation: framework

- Quantify performance penalty of xNVMe
- Commodity hardware for **reproducibility**
- Optane NVMe SSD advertises low and predictable I/O latency (~**7000 nsec**).

Hardware	Model
CPU	Intel Core i5-9400 2.9Ghz
Memory	Corsair 2x 16GB DDR4 3200Mhz CL18
Board	MSI MPG Z390I Gaming Edge AC
SSD	Intel Optane Memory M10 Series (MEMPEK1J016GAL)
Software	**Model**
FreeBSD	Version 12.1
fio	Version 3.27
gcc	Version 10.2.1
clang	Version 12.0.1
SPDK	Version 21.04
xNVMe	Version 0.0.26

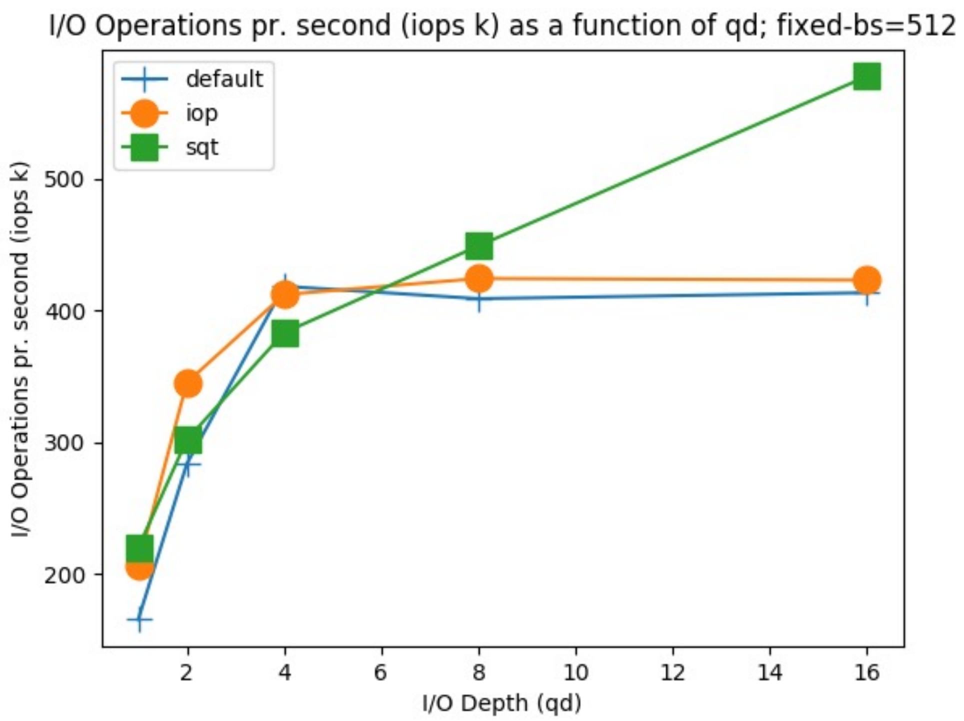
Performance Evaluation: framework

- Quantify performance penalty of xNVMe
- Commodity hardware for **reproducibility**
- Optane NVMe SSD advertises low and predictable I/O latency (~**7000 nsec**).
- **fio** is utilized to do relative comparison
  - xNVMe I/O interface implementations vs state-of-the-art reference implementations
  - Random-read spanning the entire device

Hardware	Model
CPU	Intel Core i5-9400 2.9Ghz
Memory	Corsair 2x 16GB DDR4 3200Mhz CL18
Board	MSI MPG Z390I Gaming Edge AC
SSD	Intel Optane Memory M10 Series (MEMPEK1J016GAL)
Software	**Model**
FreeBSD	Version 12.1
fio	Version 3.27
gcc	Version 10.2.1
clang	Version 12.0.1
SPDK	Version 21.04
xNVMe	Version 0.0.26

Performance Evaluation: framework

- Quantify performance penalty of xNVMe
- Commodity hardware for **reproducibility**
- Optane NVMe SSD advertises low and predictable I/O latency (~**7000 nsec**).
- **fio** is utilized to do relative comparison
  - xNVMe I/O interface implementations vs state-of-the-art reference implementations
  - Random-read spanning the entire device
- **io_uring** tunables; using submission-queue-thread-polling, register files + buffers, contig-buffer payloads



Performance Evaluation: **baseline**

- Quantify performance penalty of xNVMe
- Establish a baseline by running **without** a device
- Fio random-read at qd=1, bs=4k
  - built-in I/O engine **NULL**
  - xNVMe I/O engine using **–async=nil**

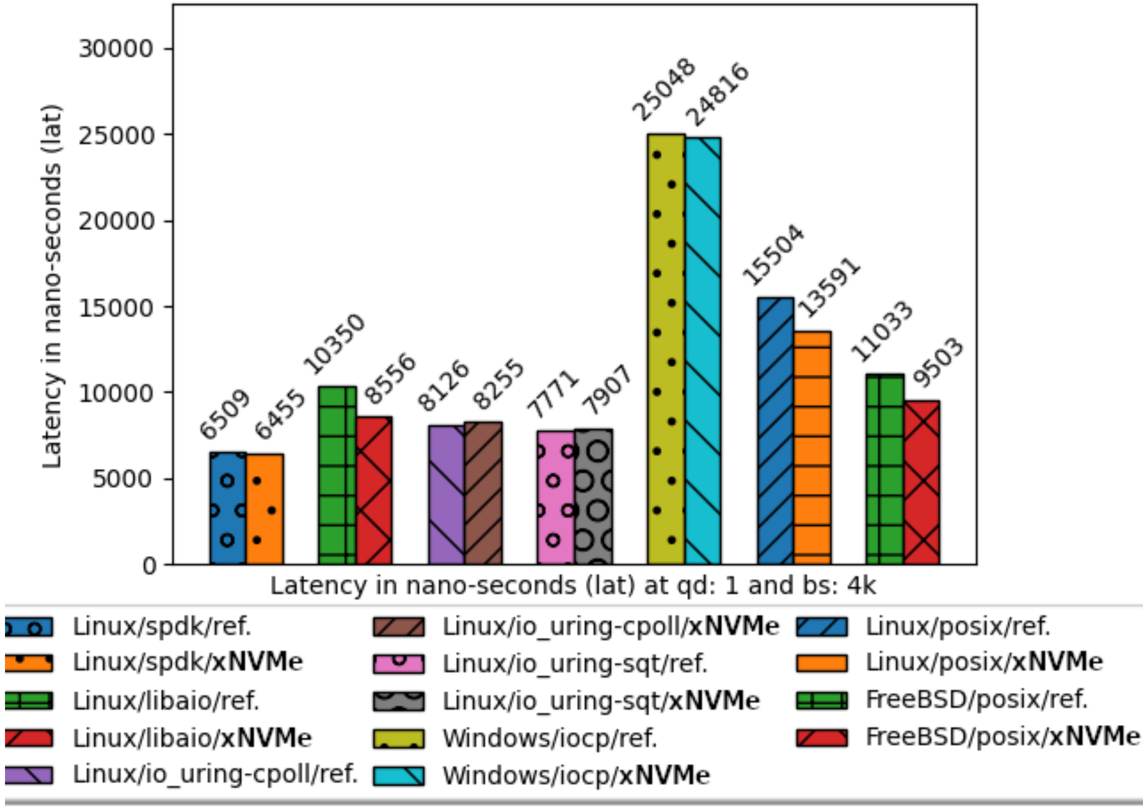
engine \ latency (nsec)	Avg.	Min.	Max.	Std.dev.
NULL	36	8	17916	48
xNVMe[async=nil]	90	82	15844	74

1) xNVMe does not impact variance, thus, we consider avg. lat.
2) Baseline overhead = 90 – 36 = **54 nsec** per I/O

- We will now explore how xNVMe behaves when accessing an SSD through the following I/O interfaces: POSIX aio (FreeBSD + Linux), libaio, IOCP, io_uring and SPDK/NVMe.

Performance Evaluation: interface qd=1, bs=4k

- Quantify performance penalty of xNVMe
- **expected** penalty = reference latency + baseline + I/O specific
- Expectation is met for io_uring
  - Penalty = ~**136 nsec**
- Otherwise, same/less ➔ Why?
- Interrupt-driven I/O interfaces
  - xNVMe spins instead of waiting for interrupt/wakup
- SPDK/NVMe
  - Different IO engine, doing more work
  - Hooks in at a higher-level in the driver



Performance Evaluation: scalability check

- Varying **queue-depth** (qd)=[1,2,4,8]; fixed block-size (bs)=4k
- Varying **block-size** (bs)=[512,4k,32k]; fixed queue-depth (qd) =1

- The above visualized as plots of latency as a function of the varied parameter

Performance Evaluation: scalability check

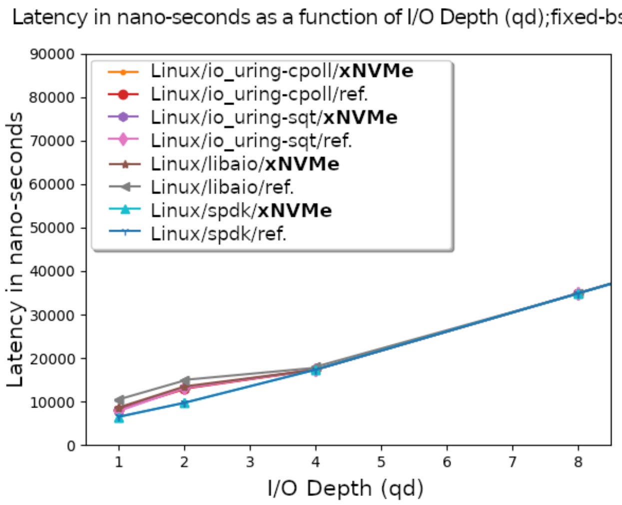
- Varying **queue-depth** (qd)=[1,2,4,8]; fixed block-size (bs)=4k
- Varying **block-size** (bs)=[512,4k,32k]; fixed queue-depth (qd) =1

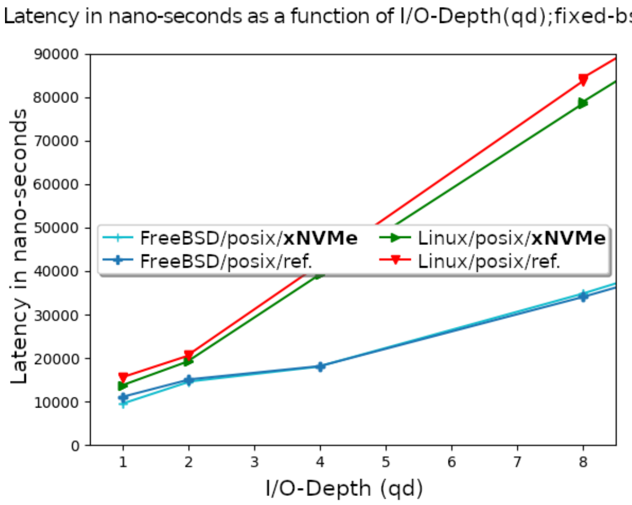
- The above visualized as plots of latency as a function of the varied parameter

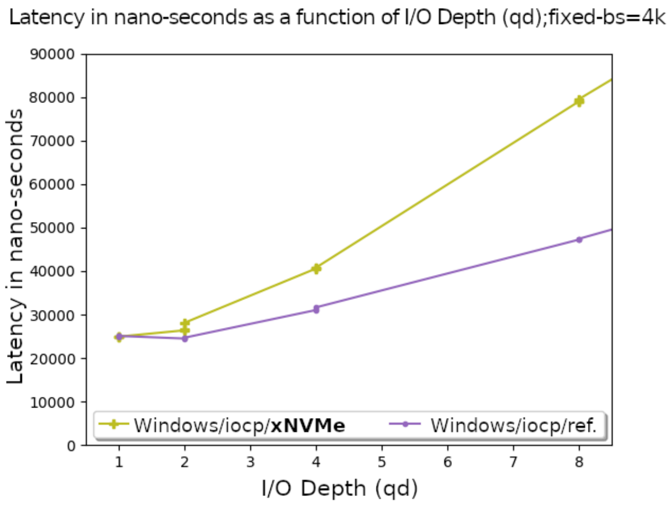
- A **perfect** result would illustrate xNVMe and the reference implementation as lines parallel to each other
  ➔Thus, the xNVMe overhead does not degrade with increasing queue depth or block size

Performance Evaluation: scalability check

- Varying **queue-depth** (qd)=[1,2,4,8]; fixed block-size (bs)=4k



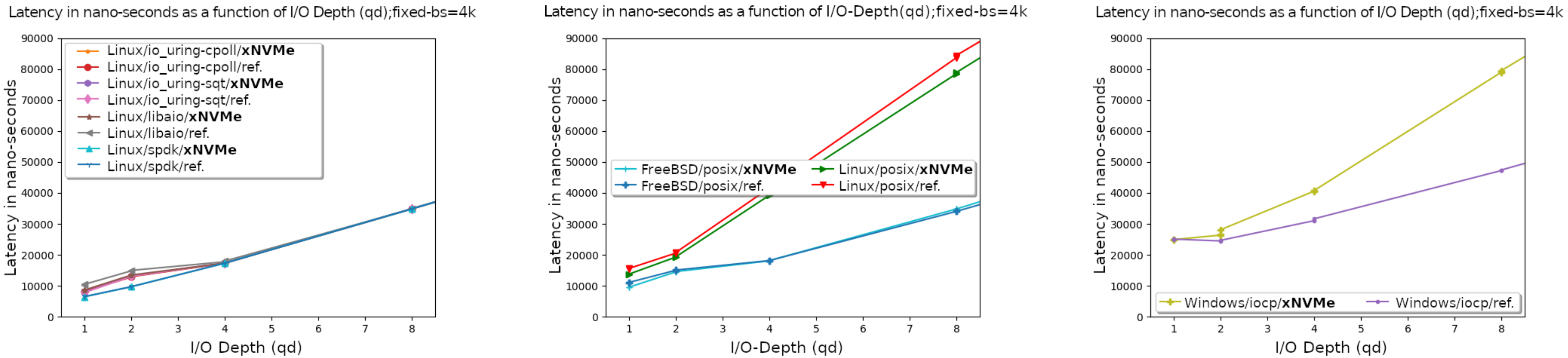




- A near **perfect** result is achieved on all accounts for the xNVMe implementations, except for the Windows I/O interface, this has been identified as a short-coming in the backend implementation

Performance Evaluation: scalability check

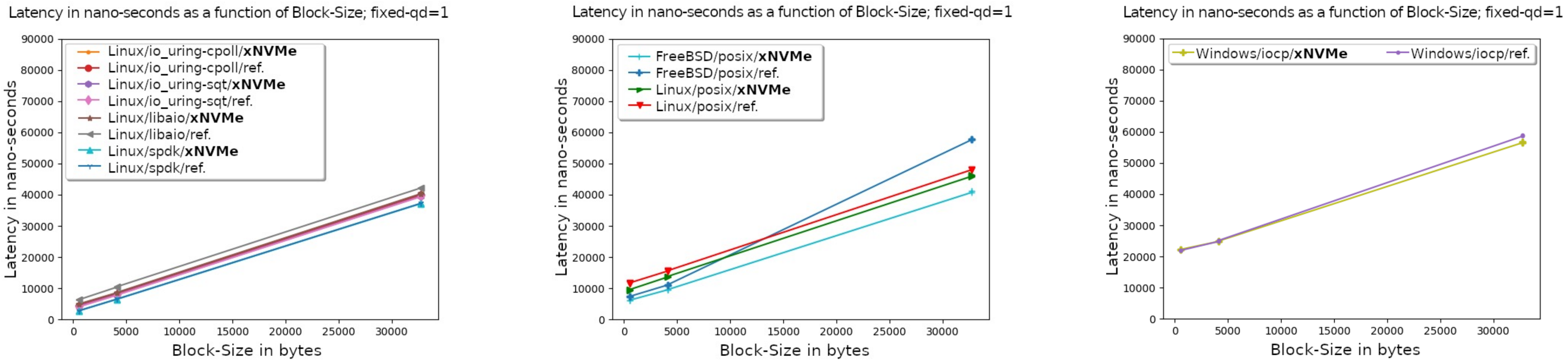
- Varying **queue-depth** (qd)=[1,2,4,8]; fixed block-size (bs)=4k



- A near **perfect** result is achieved on all accounts for the xNVMe implementations, except for the Windows I/O interface, this has been identified as a short-coming in the backend implementation
- Observations **unrelated** to xNVMe:
  - POSIX **aio** does dramatically better on **FreeBSD** than on it does on **Linux**.
  - On Linux, **io_uring**, **libaio** and **SPDK** saturates the device at **QD4**.

Performance Evaluation: scalability check

- Varying **block-size** (bs)=[512,4k,32k]; fixed queue-depth (qd) =1



- A near **perfect** result is achieved on all accounts for the xNVMe implementations, and thus the xNVMe penalty is constant in this regard.
- Observations **unrelated** to xNVMe:
  - POSIX aio on FreeBSD has issues with larger block-sizes.

Performance Evaluation: conclusion on **overhead**

- Quantify performance penalty of xNVMe
- Baseline penalty ~ **54 nsec** per I/O
- io_uring penalty ~ **129 nsec** to**136 nsec**
- Interrupt-driven; **less** than reference due to completion-processing
- User space; **less** due to minor difference io-engine implementation
- The **penalty** is constant when scaling I/O depth and block-size
  - Except for Windows IOCP
- **Future-work**:
  - ~~Add option to disable completion-status polling on interrupt-driven I/O~~
  - ~~Address Windows IOCP shortcomings~~

Performance Evaluation: abstraction **efficiency**

xNVMe in **SPDK**

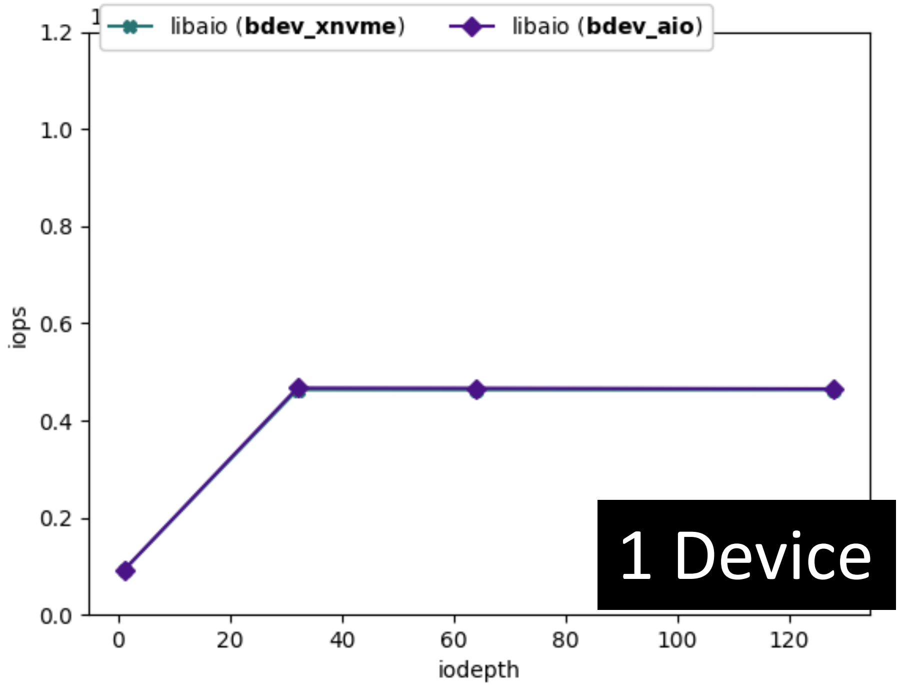
Comparing **bdev** implementations using `bdevperf`

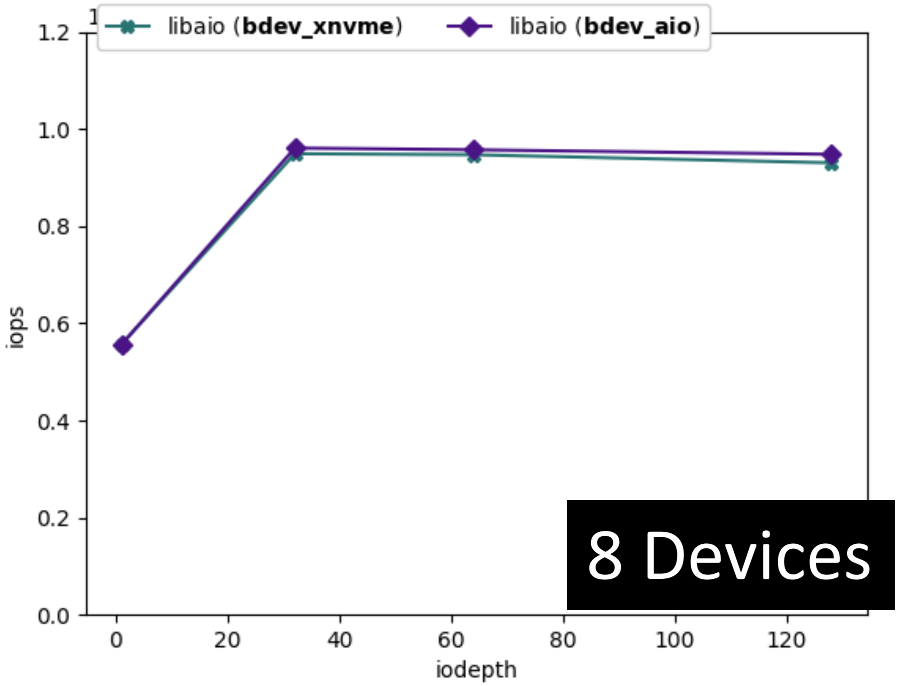
bdev_**aio** vs bdev_**xnvme** (io_mechanism=**libaio**)
bdev_**uring** vs bdev_**xnvme** (io_mechanism=**io_uring**)
bdev_**uring** vs bdev_**xnvme** (io_mechanism=**io_uring_cmd**)

Measure: Peak **IOPS** on a single physical **CPU** core

Performance Evaluation: SPDK **bdev_xnvme**

bdev_**aio** vs bdev_**xnvme** (io_mechanism=**libaio**)

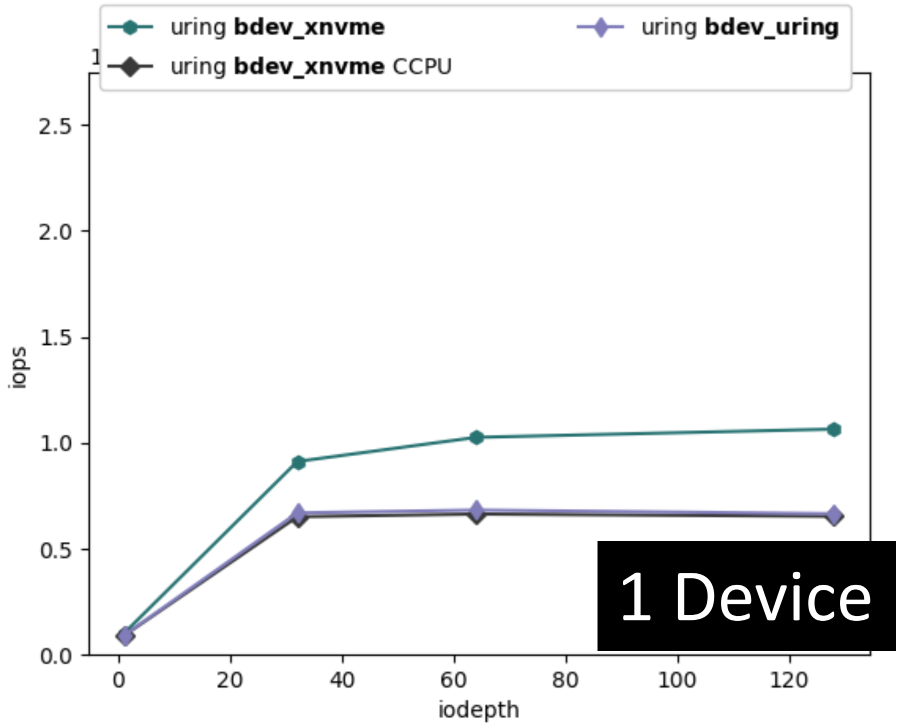


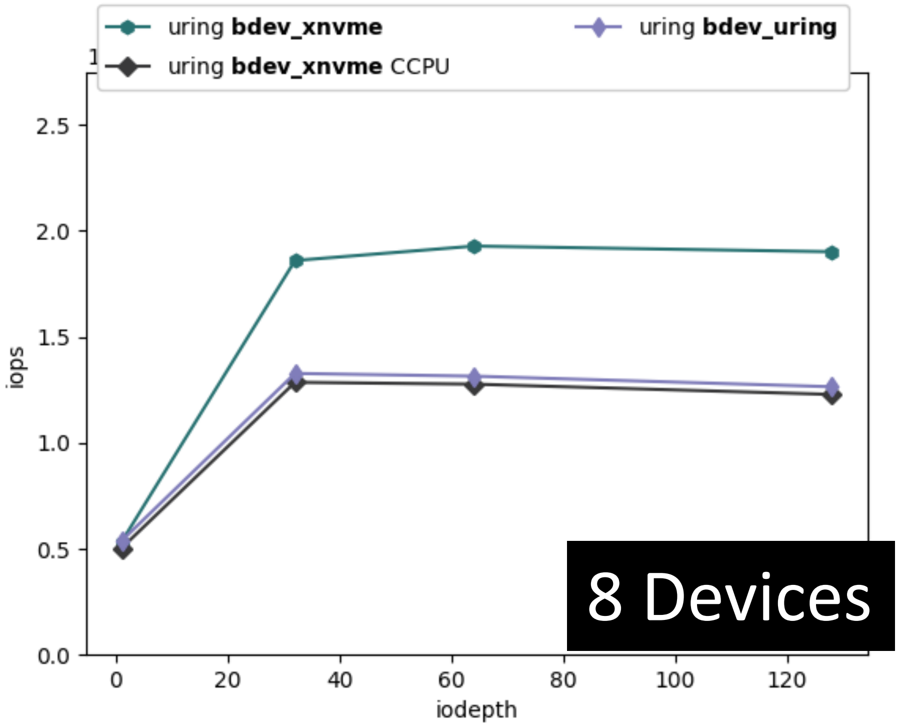


- bdev_**xnvme** at scale with bdev_**aio**

Performance Evaluation: SPDK **bdev_xnvme**

bdev_**uring** vs bdev_**xnvme** (io_mechanism=**io_uring**)

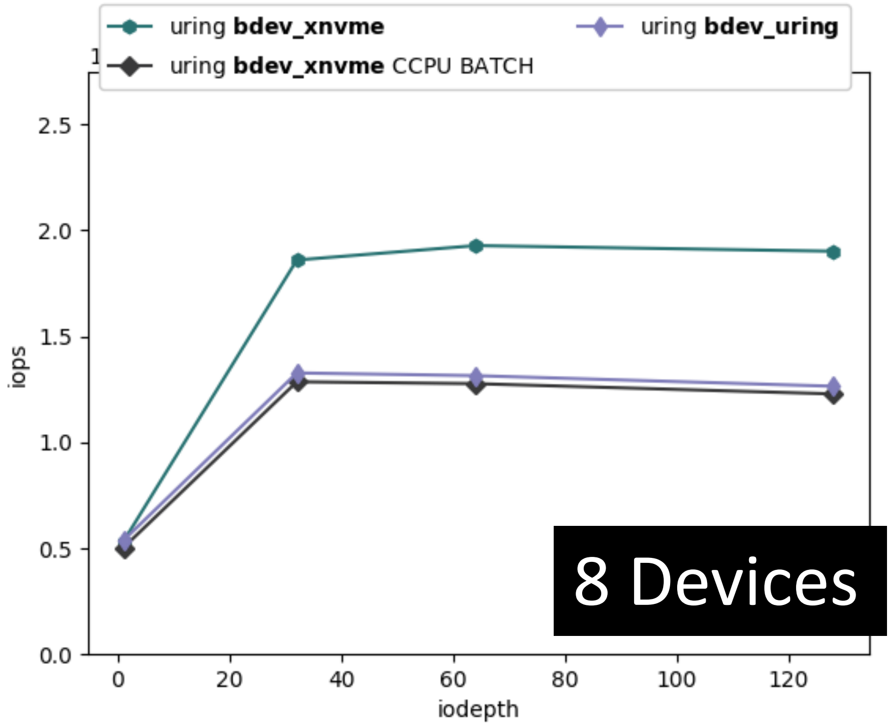


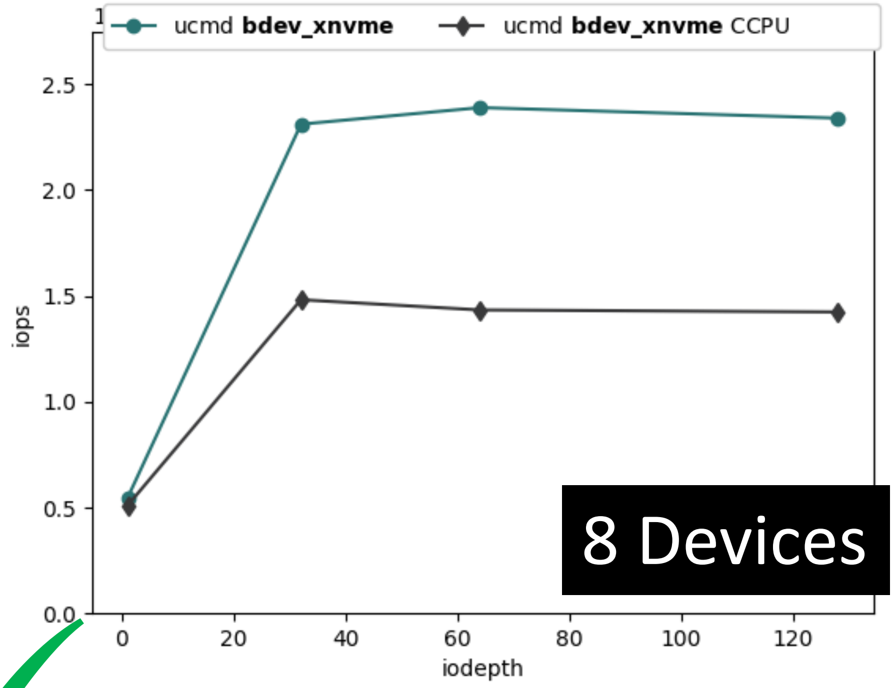


- bdev_**xnvme** at scale with bdev_**uring** ✓
- bdev_**xnvme** "out-scales" bdev_**uring** with **IOPOLL** enabled ✓

Performance Evaluation: SPDK **bdev_xnvme**

bdev_**uring** vs bdev_**xnvme** (io_mechanism=**io_uring_cmd**)





- bdev_**xnvme** (io_uring_cmd) **>** bdev_**uring**
  - Both with and without **IOPOLL**
- bdev_**xnvme** (io_uring_cmd) **>** bdev_**xnvme** (io_uring)

Integrations and Interoperability 1/2

- **Fio**
  - xNVMe is merged in upstream **fio**
  - Fio io-engine name: `xnvme`
  - Available since **fio v3.31**
- **SPDK**
  - xNVMe is merged in upstream **SPDK**
  - SPDK bdev: `bdev_xnvme`
  - Available since **SPDK v22.09**
- **Work-in-Progress**
  - Automated performance evaluation
  - Integration into **nvme-cli** and **libblkio**

Integrations and Interoperability 2/2

- **libvfn** backend
  - Linux vfio-based user space NVMe driver for low-level tinkering
  - See: https://github.com/OpenMPDK/libvfn
  - Available since xNVMe **v0.5.0**
- **Python** Bindings
  - Based on **ctypes**
  - Available from xNVMe **v0.7.2**
- **Rust** Bindings
  - Based on **bindgen**
  - Available from xNVMe **v0.7.2**

Summary

- I/O Interface Independence is achievable with **xNVMe** for a cost of **54** to **136 nsec** per I/O

- **Unified API** for the continuing innovation of I/O interfaces
  - In **C**, **Python**, and **Rust**
- **Fio**, available now and since **v3.31**
- **SPDK,** available now and since **v22.09**

- Discord: https://discord.com/invite/XCbBX9DmKf
- Documentation: https://xnvme.io/docs/
- Repository: https://github.com/OpenMPDK/xNVMe
- SYSTOR22 Article: https://dl.acm.org/doi/10.1145/3534056.3534936
- SDC23 Presentation: https://storagedeveloper.org/events/agenda/session/553