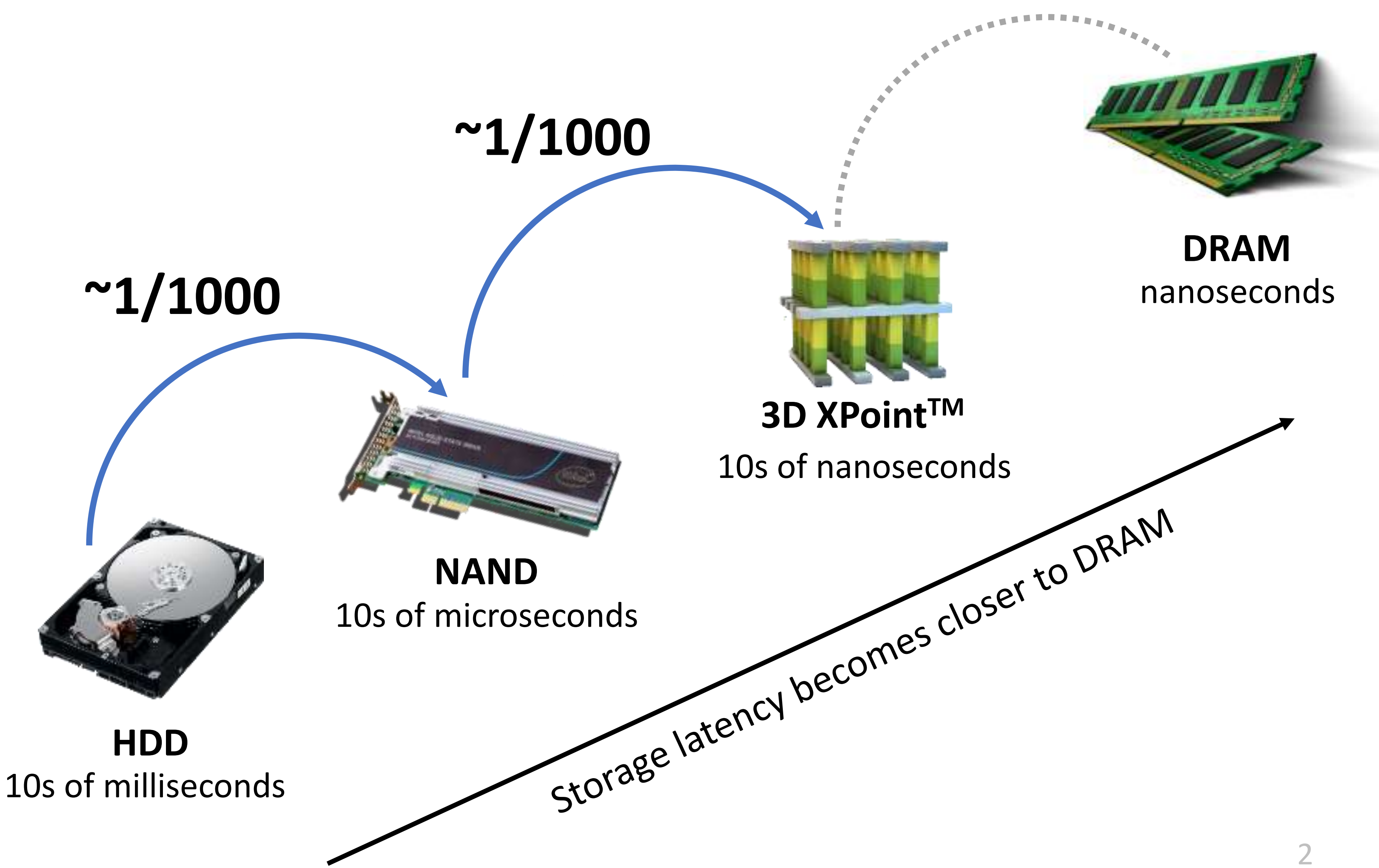
HotStorage ’16
June 20, 2016

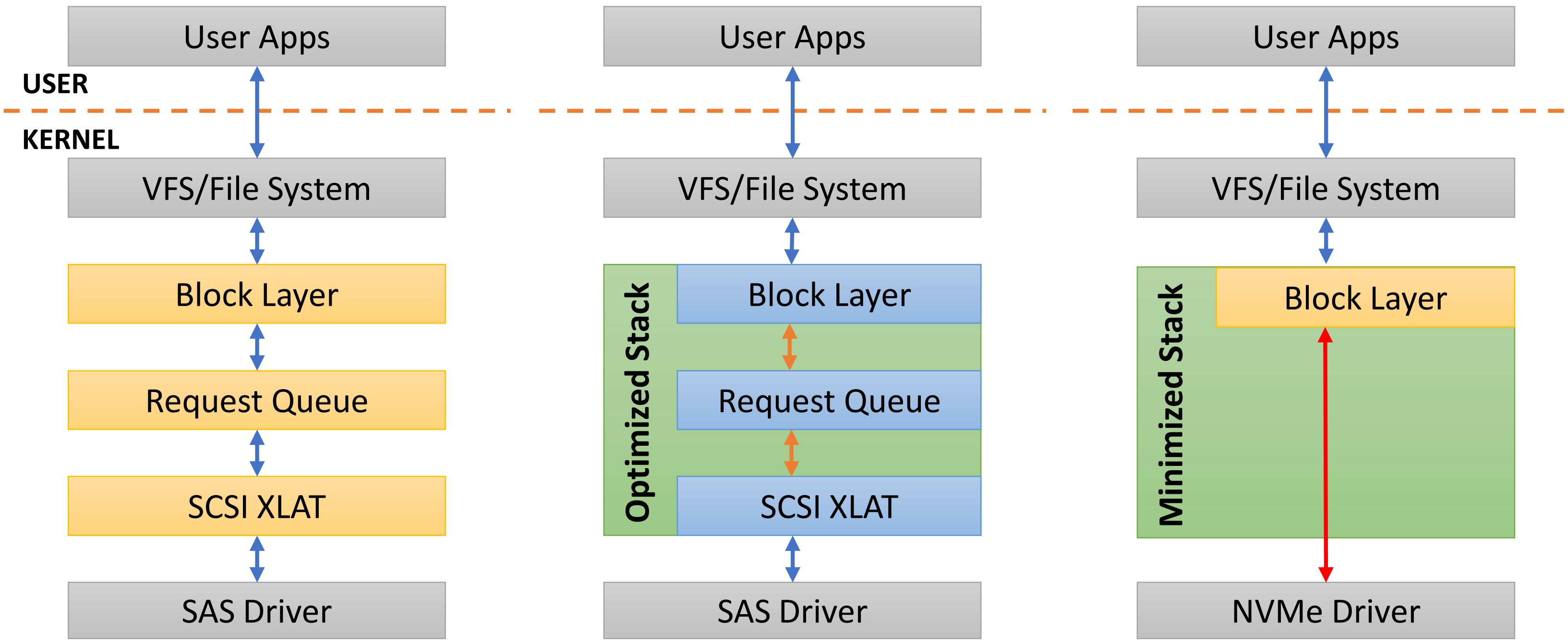
NVMeDirect: A User-Space I/O Framework for Application-specific Optimization on NVMe SSD

Hyeong-Jun Kim, Jin-Soo Kim
Sungkyunkwan University
Young-Sik Lee
KAIST

Evolution of storage device



Evolution of storage I/O stack



HDD (~10ms)

SAS SSD (~150µs)

NVMe SSD (<100µs)

Previous approaches – Optimizing the kernel storage stack

- Use of polling for the fast I/O completion [Yang et al. FAST 2012]
- Optimization of a low-level hardware abstraction layer [Shin et al. ATC 2014]
  - Reducing the translation overhead between abstraction layers
- Optimizations to fully exploit the performance of fast storage devices [Yu et al. ACM TOCS 2014]
  - Polling, request merging, double buffering and reducing context switches

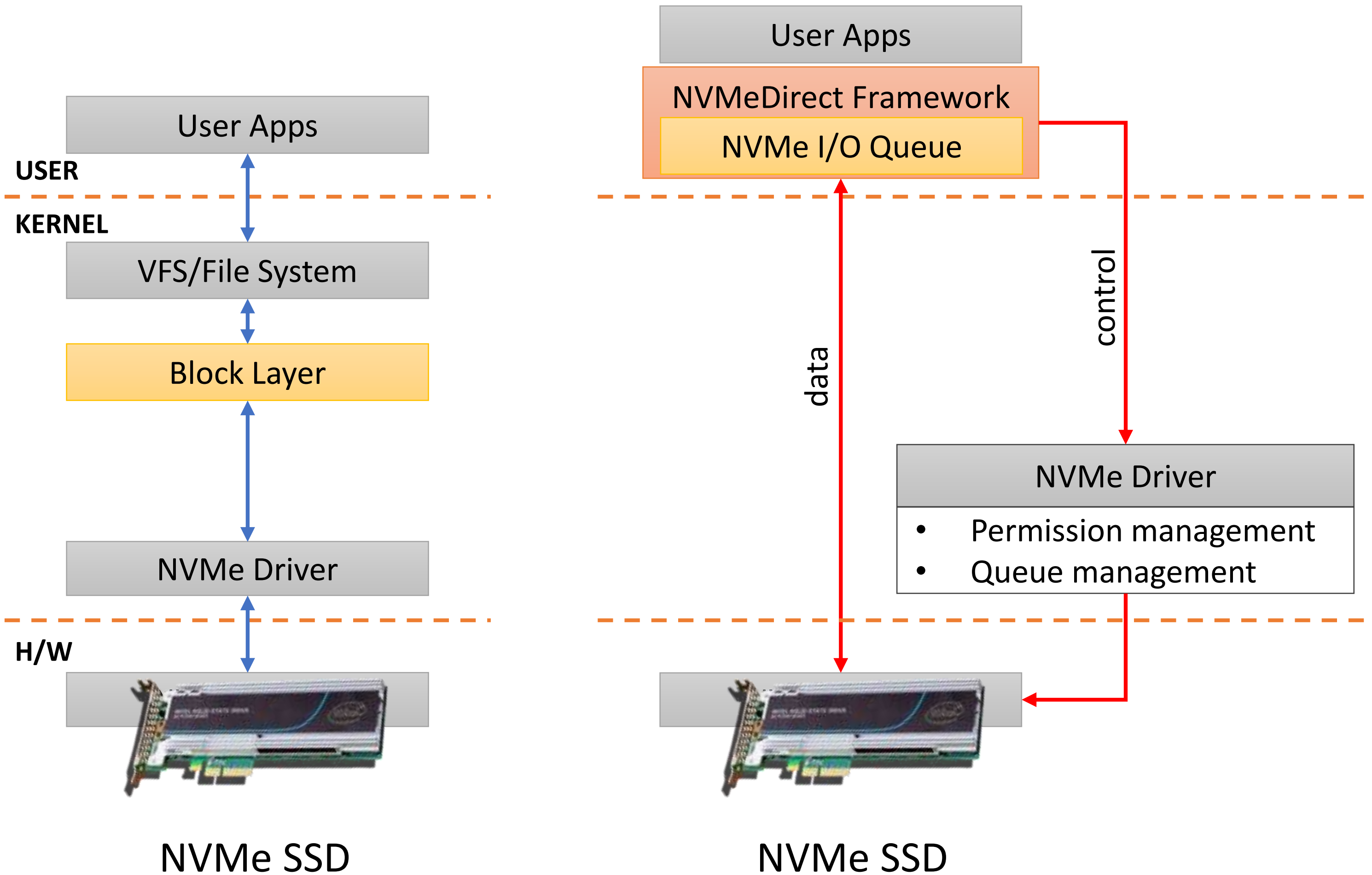
Limitations - Optimizing the kernel storage stack

- Kernel should be general to provide an abstraction layer
- Kernel cannot implement any policy that favors a certain application
- Updating kernel requires a constant effort to port application-specific optimizations

Previous approaches – Direct access to storage device

- Direct access to the special storage device [Caulfield et al. ASPLOS 2012]
  - Special hardware is required
- Direct access to NVMe device
  - Intel Storage Performance Development Kit – SPDK (Sep 2015)
  - Micron Userspace NVMe driver project – UNVMe (Feb 2016)
  - Device dedicated to a single user process
  - Provides just simple read & write interface based on polling
  - Not sufficient to port existing applications

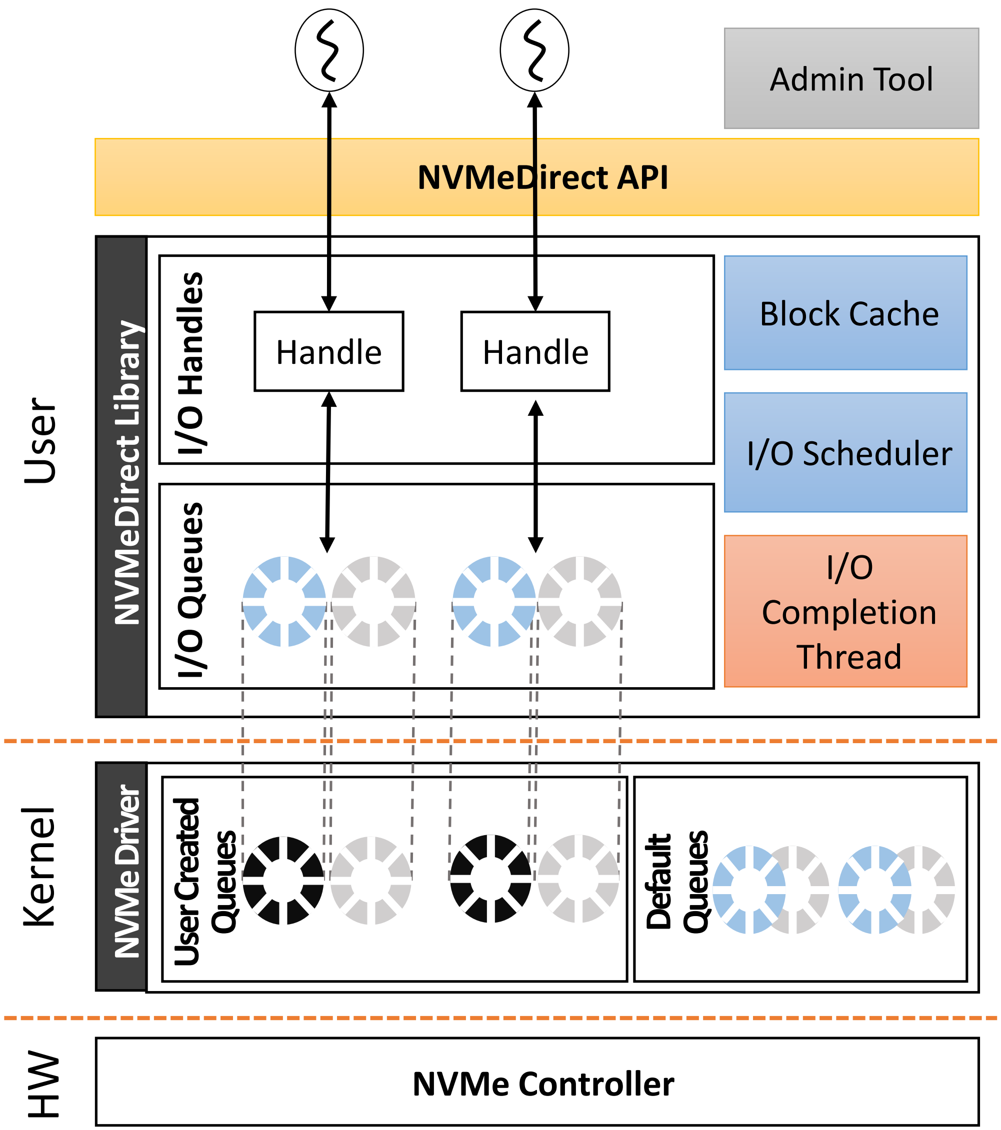
Our Approach: NVMeDirect



NVMeDirect Overview

- Allows user-space applications to directly access NVMe SSDs without any hardware modifications
  - Achieves high performance by avoiding storage stack overhead
- Supports various I/O policies
  - Applications can be optimized according to their I/O characteristics
  - Selective use of block cache, I/O scheduler, or I/O completion thread
  - Asynchronous I/O vs. Synchronous I/O
  - Buffered I/O vs. Direct I/O
- Designed to maximize performance for trusted applications
  - Storage appliance, private clouds, etc.

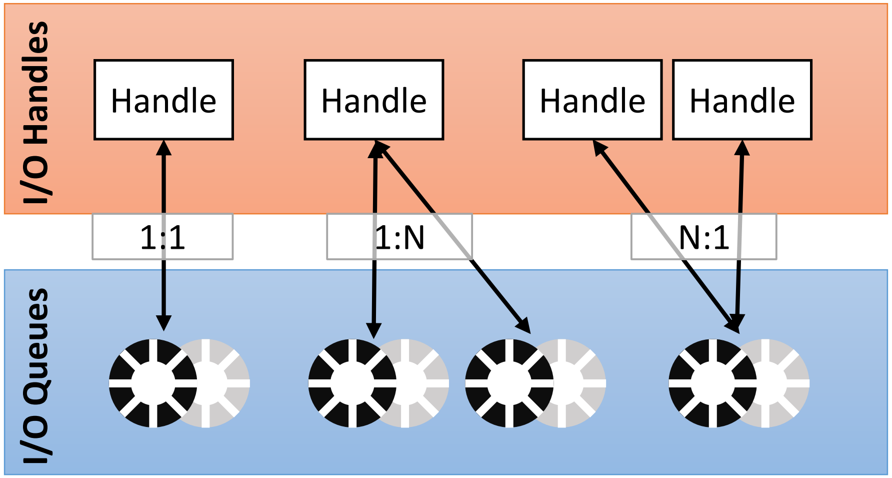
NVMeDirect Design



- **NVMeDirect Management**
  - Kernel driver
  - Admin tool
- **NVMeDirect I/O**
  - I/O Handles
  - User-space I/O Queues
- **NVMeDirect I/O Framework**
  - Block Cache
  - I/O Scheduler
  - I/O Completion Thread

NVMeDirect Design – Queues and Handles

- **User-space I/O Queues**
  - Memory-mapped address space for NVMe I/O Queues created in the kernel address space
- **I/O Handles**
  - Used to send I/O requests to NVMe I/O Queue(s)
  - A thread can create one or more I/O Handles
  - Each Handle can be configured to use different features : caching, I/O scheduling, I/O completion, etc.



NVMeDirect Design - APIs

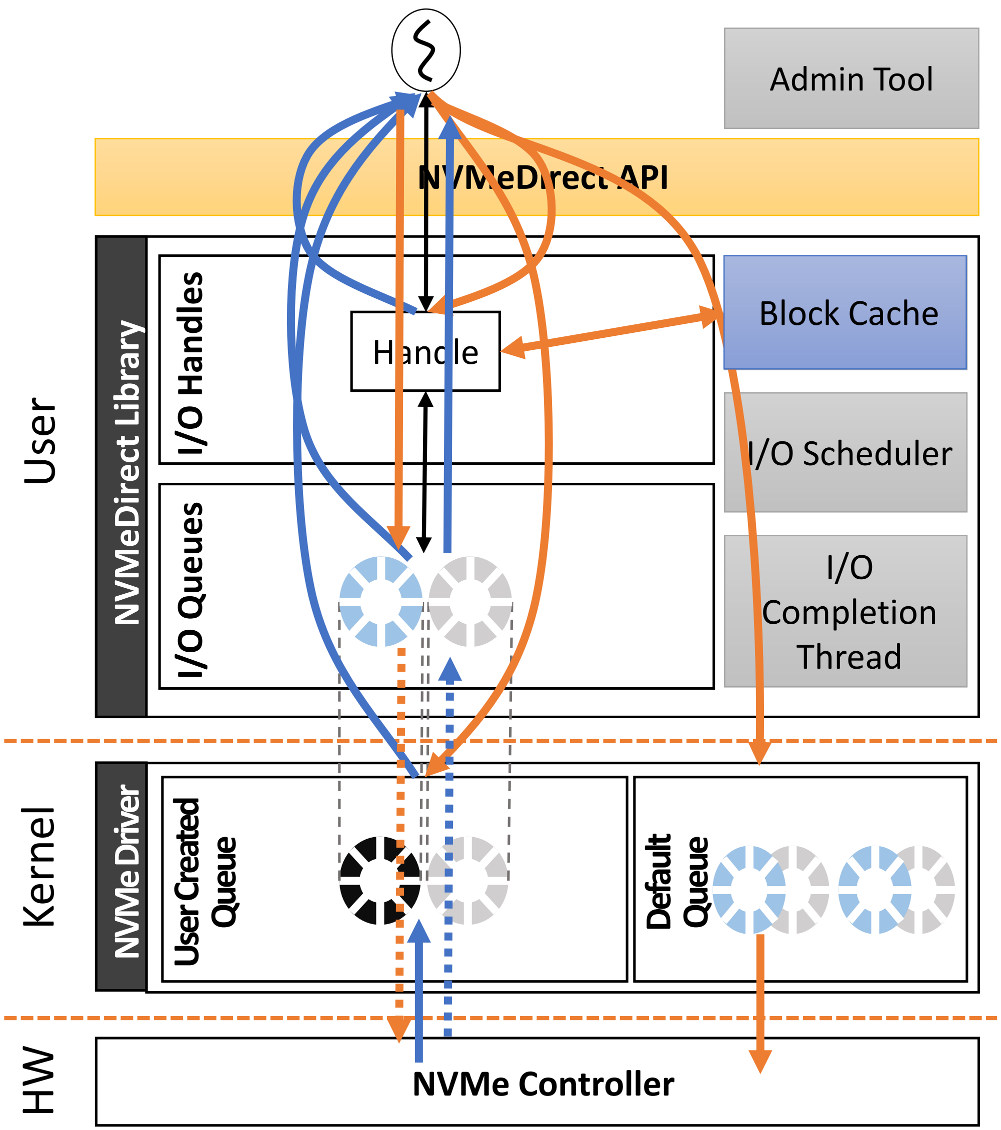
Driver Connection	`struct nvmed* nvmed_open(char*);` `int nvmed_close(struct nvmed*);`
Queue Management	`struct nvmed_queue* nvmed_create_queue(struct nvmed*);` `int nvmed_destroy_queue(struct nvmed_queue*);`

I/O Handle Management	`struct nvmed_handle* nvmed_create_handle(struct nvmed_queue*);`
	`struct nvmed_handle* nvmed_create_mq_handle(struct nvmed_queue**);`
	`int nvmed_destroy_handle(struct nvmed_handle*);`
	`int nvmed_set_param(struct nvmed_handle*, int, int);`

Buffer Management	`void* nvmed_get_buffer(struct nvmed_handle*, unsigned int);` `int nvmed_put_buffer(void*);`

I/O	`off_t nvmed_lseek(struct nvmed_handle*, off_t, int);`
	`ssize_t nvmed_read(struct nvmed_handle*, void*, unsigned int);`
	`ssize_t nvmed_write(struct nvmed_handle*, void*, unsigned int);`
	`int nvmed_flush(struct nvmed_handle*);`
	`int nvmed_discard(struct nvmed_handle*, off_t, size_t);`

Example of I/O using NVMeDirect



1) Open device

```
nvmed = nvmed_open("/proc/nvme0/n1");
```

2) Create queue

```
queue = nvmed_queue_create(nvmed);
```

3) Create handle

```
handle = nvmed_handle_create(queue);
```

4) Perform I/O

```
size = nvmed_read(handle, buf, len);
```

5) Configure Handle

```
ret = nvmed_set_param
        (handle, BUFFERED_IO, TRUE);
```

Advantages of NVMeDirect

- Enables high performance I/O
  - Low latency and high throughput
- Easy to support new interfaces
  - Weighted queue, multi-stream, etc.
- Easy to develop and debug
- Provides various I/O policies
- Free from kernel update
- Co-exists with legacy kernel I/O

Evaluation

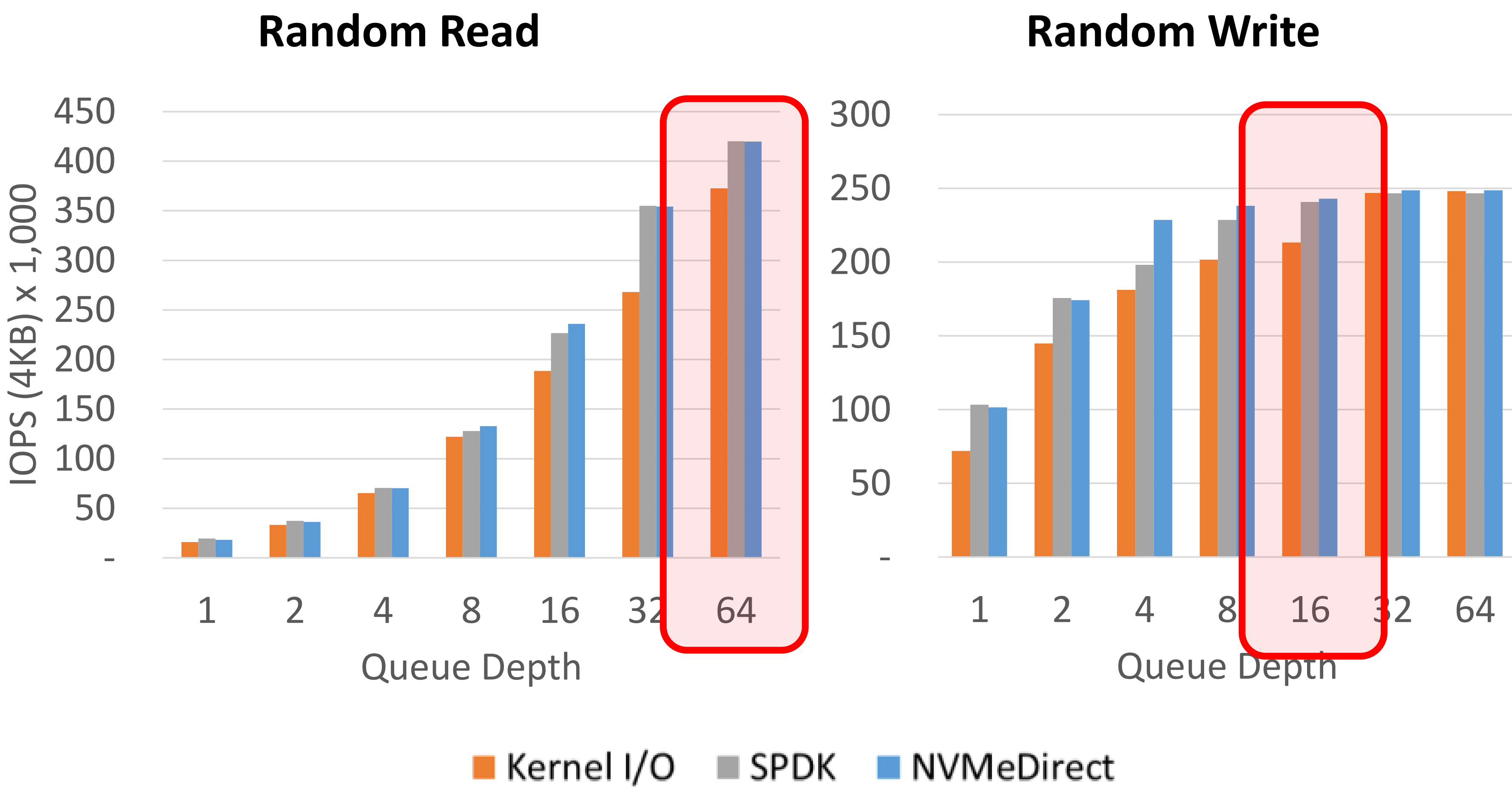
- Implementation on the Linux kernel 4.3.3

- Experimental setup
  - Ubuntu 14.04 LTS
  - 3.3GHz Intel Core i7 CPU (6 cores) & 64GB of DRAM
  - Intel 750 Series 400GB NVMe SSD

- Comparison with
  - Kernel I/O
  - SPDK
  - NVMeDirect

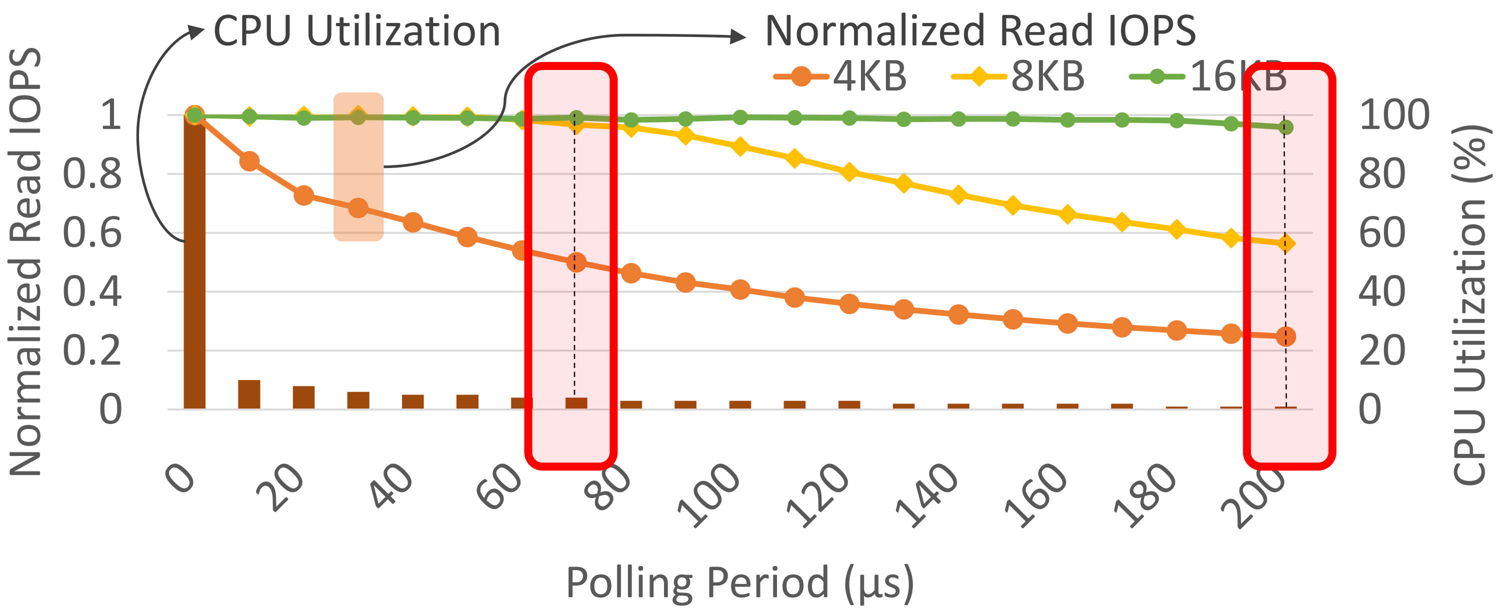
Baseline performance

- Asynchronous random I/O performance using FIO



Impact of the Polling Period

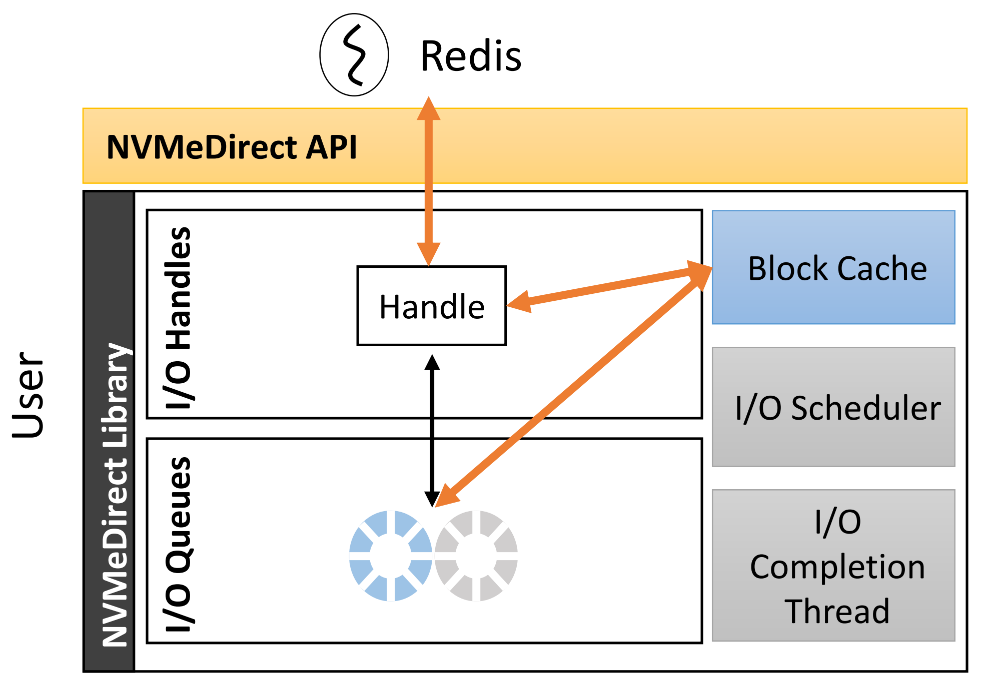
- Polling is not efficient on bandwidth sensitive workload due to the significant increase in the CPU load
- Significant performance degradation occurs in a certain polling period



- Control Polling Period dynamically based on I/O size or hints from applications

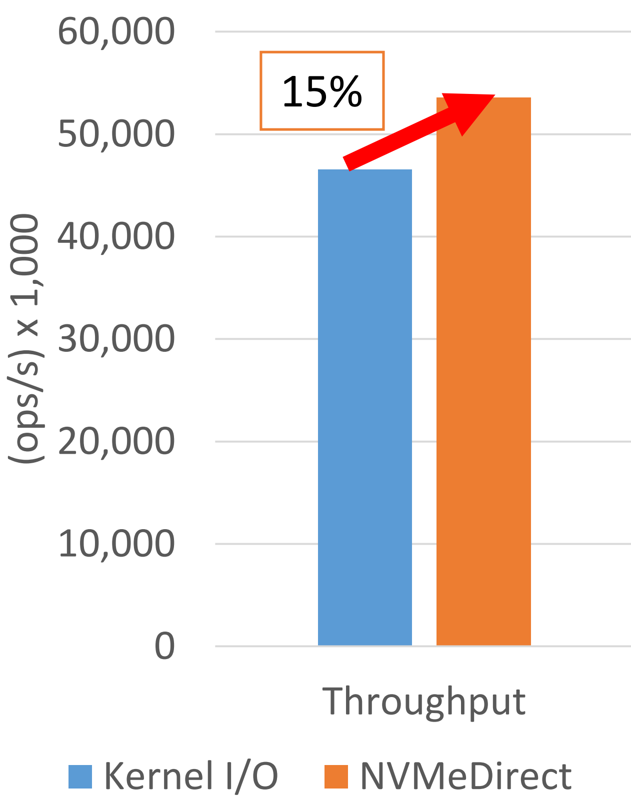
Latency-sensitive Application

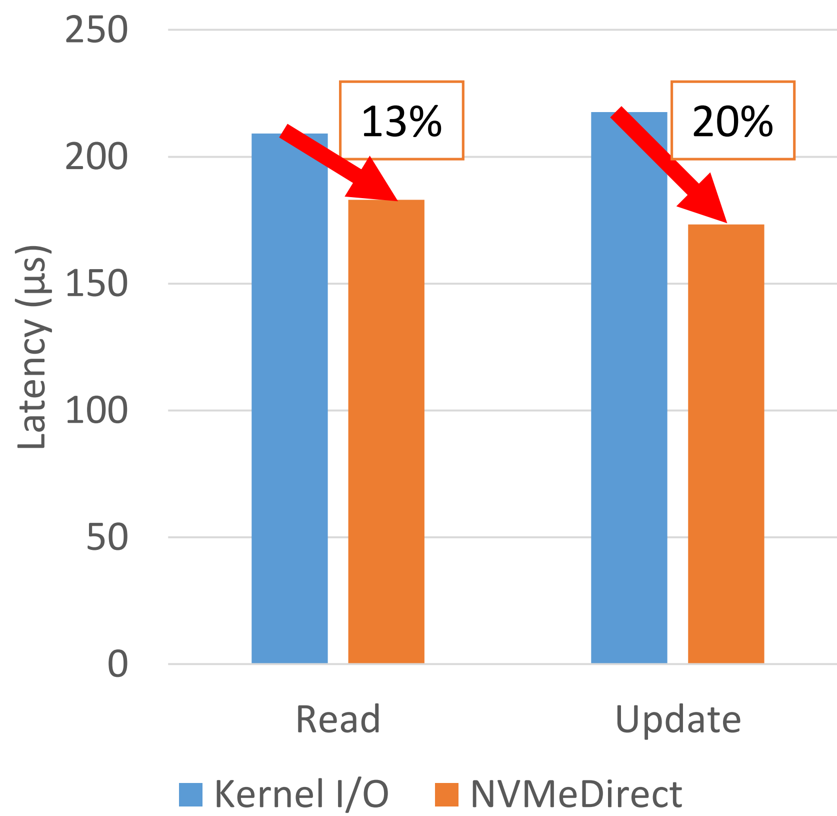
- Redis: in-memory data structure store
- Logging every operation for persistency
- Logs are 10 to 100 bytes in size
- Write buffer is required due to small-size data
  - Difficult to run on SPDK without significant code modification



Latency-sensitive Application

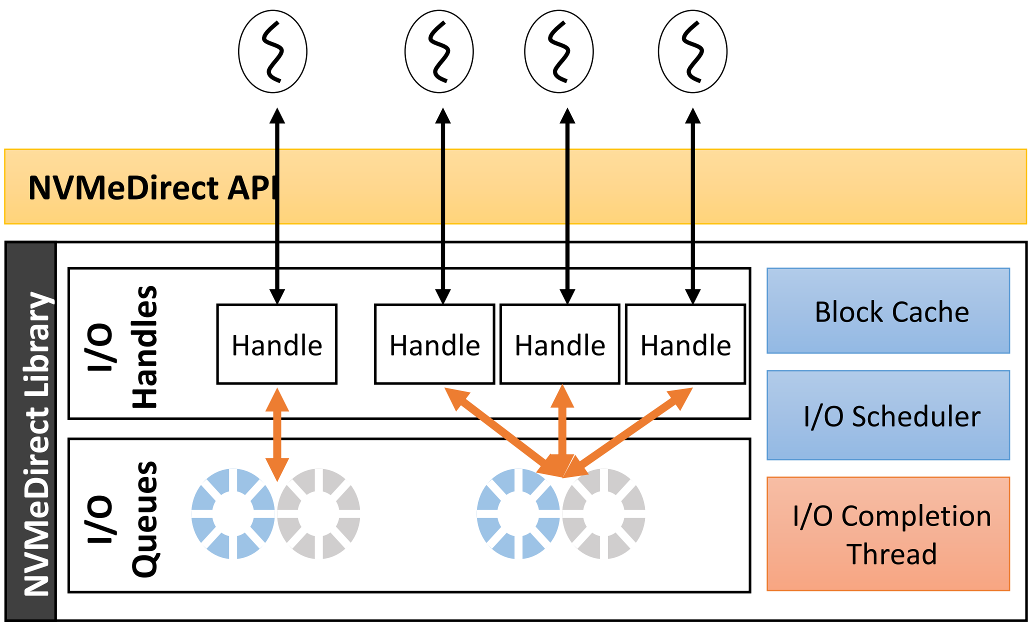
- Using workload-A in YCSB on Redis
- Update-heavy workload with Zipf distribution





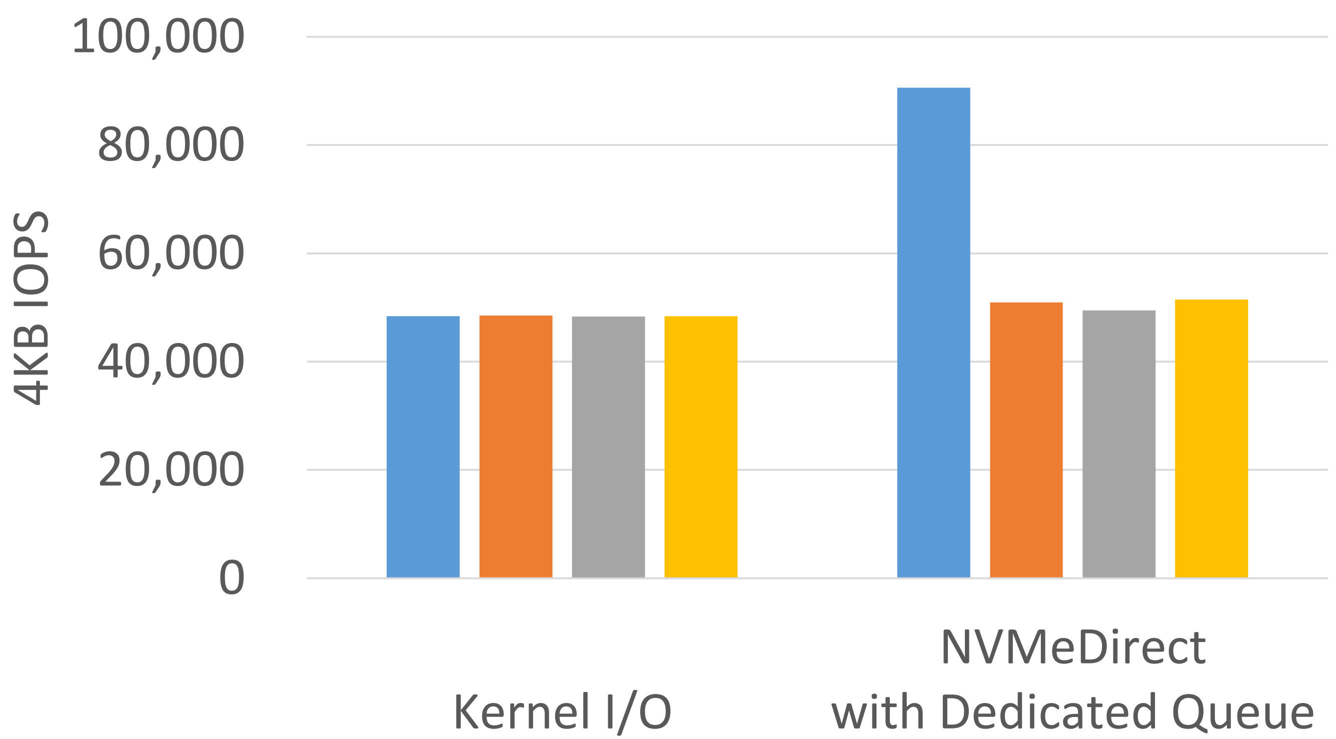
Differentiated I/O Service

- NVMeDirect supports prioritized I/O without H/W features
  - Prioritized I/O without a weighted round-robin scheduler
  - Using flexible binding between Handles and Queues
  - Sharing a single Queue with multiple Handles



Differentiated I/O Service

- One prioritized thread with a dedicated queue,
  Three threads with a shared queue
- Each thread performs 4KB random write



Conclusion

- NVMeDirect
  - First full framework for
    I/O in the user-space based on stock NVMe devices
  - Can be easily applied to many applications
  - Useful for emerging storage devices, e.g. 3D XPoint™, etc.
  - Available as open-source at
    https://github.com/nvmedirect (July 2016)

- Future work
  - User-level file systems
  - Porting diverse data-intensive applications over NVMeDirect
  - Protecting the system from illegal access

Thank you

hjkim@csl.skku.edu