Enabling NVMe WRR support in Linux Block Layer

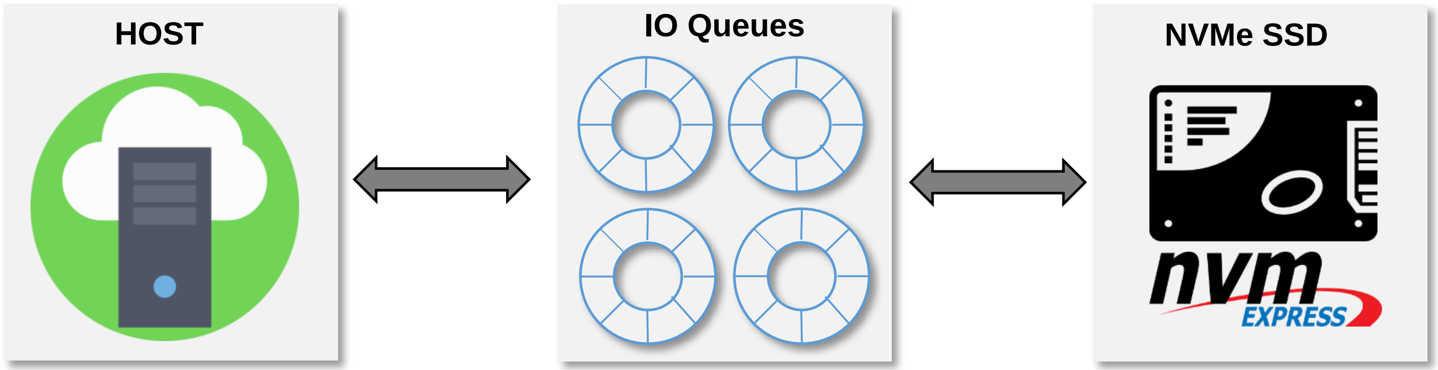
USENIX HotStorage’17

Kanchan Joshi, Praval Choudhary, Kaushal Yadav
Memory solutions, Samsung Semiconductor India R&D

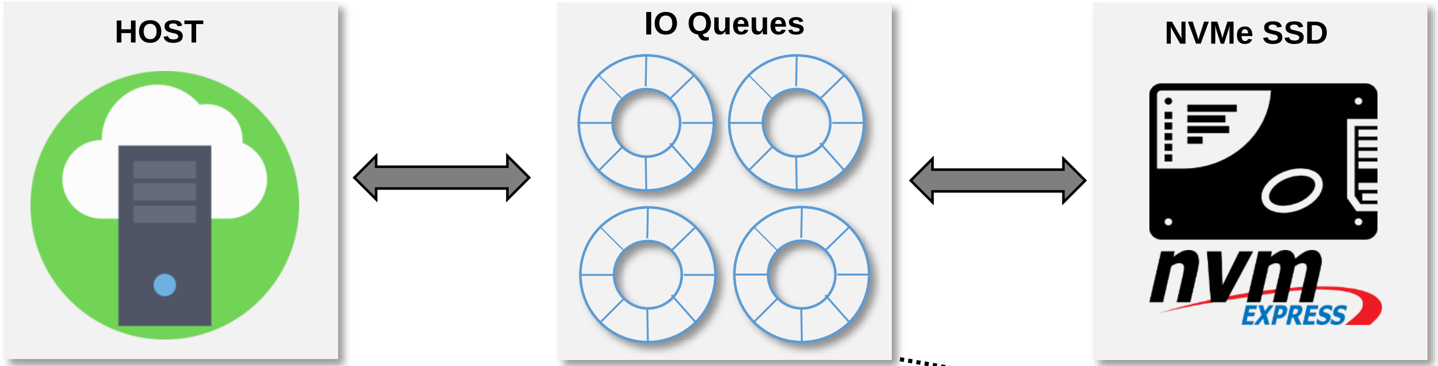
Outline

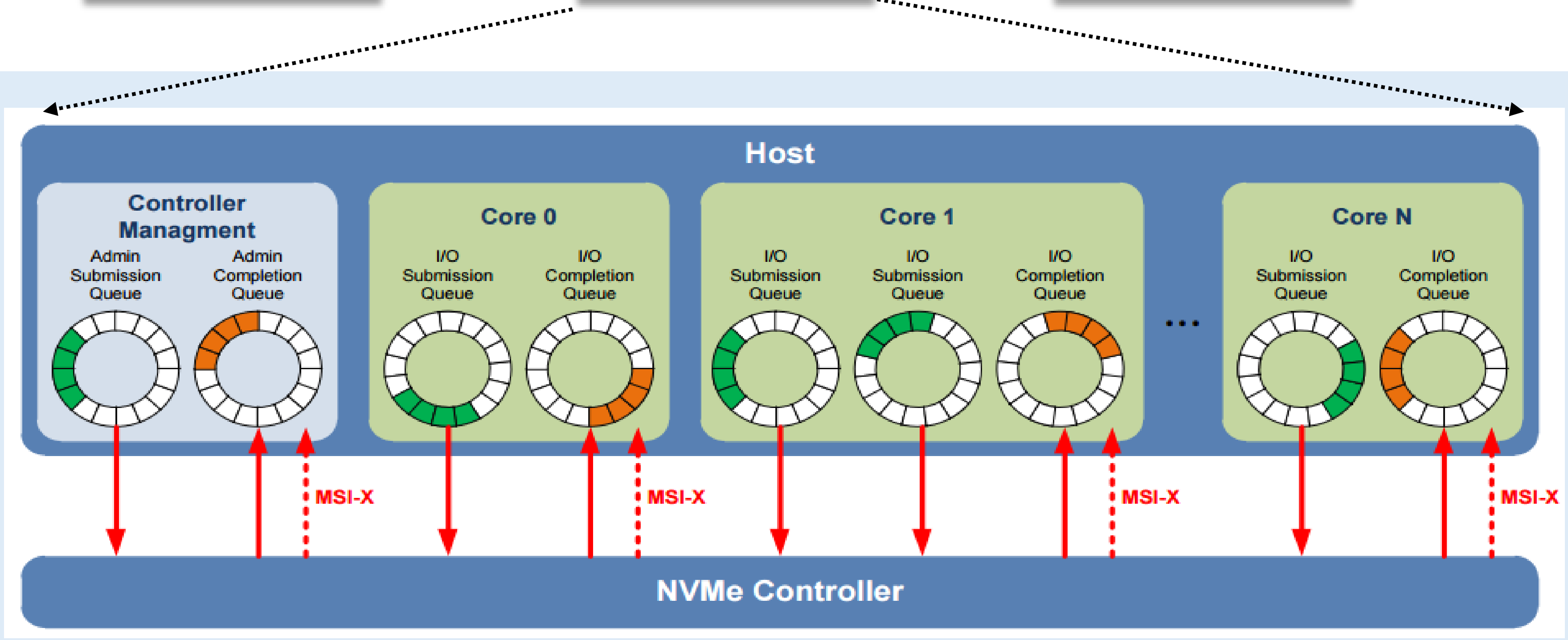
- NVMe I/O queues
- Arbitration methods and WRR
- What it takes to build differentiated I/O service
- Affinity based method and its drawback
- Proposed method
- Results
- Summary

NVMe I/O Queues



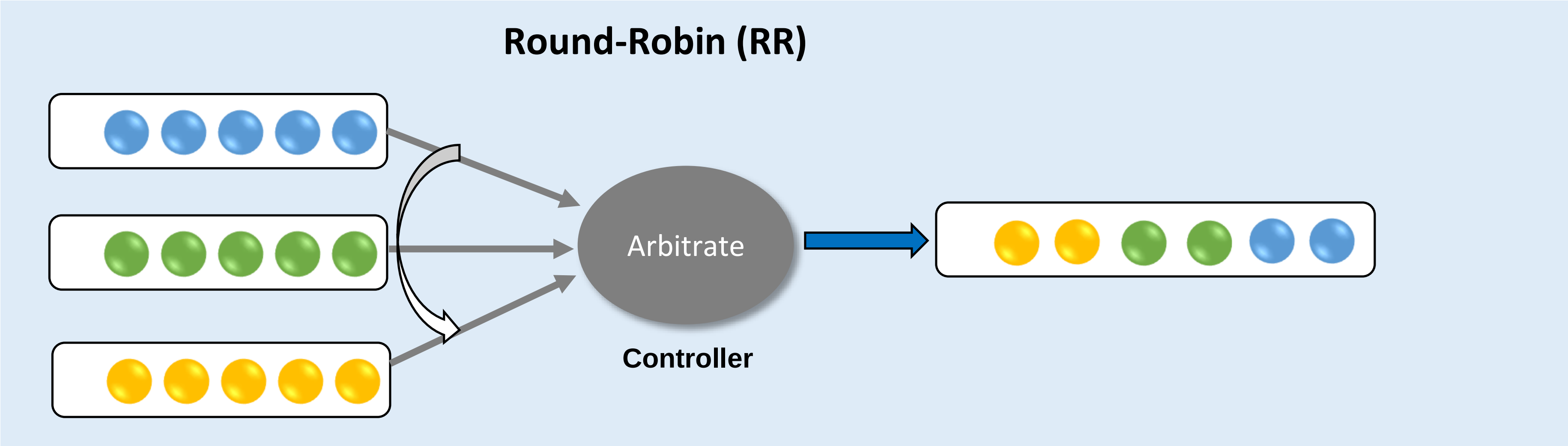
NVMe I/O Queues



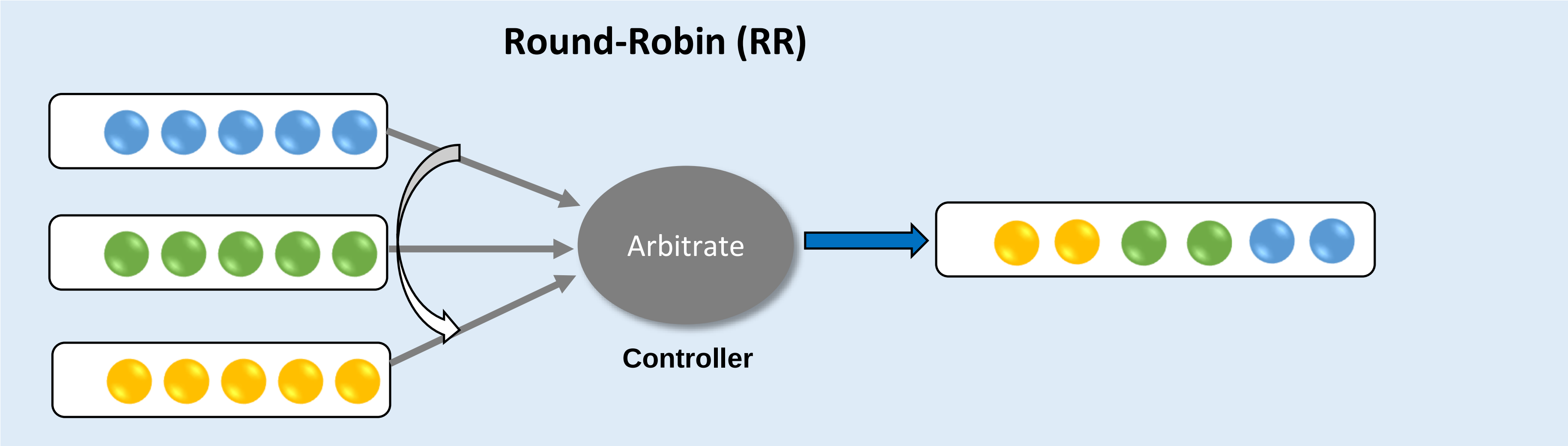


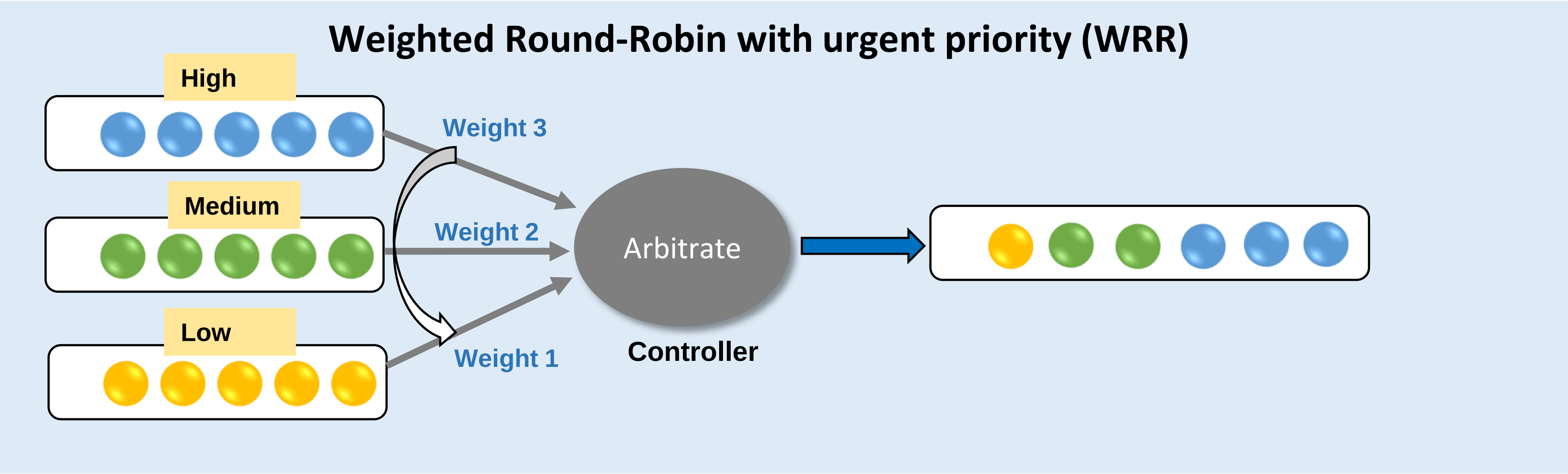
- Per-CPU queue pair
- Parallel I/O distribution
- Fast core-local path

Arbitration Methods



Arbitration Methods





Problem Statement

SSIR
SAMSUNG SEMICONDUCTOR INDIA R&D CENTER

How to make prioritization capability (WRR) benefits reach to Applications!

WRR Support Requirements

SSIR
SAMSUNG SEMICONDUCTOR INDIA R&D CENTER

I/O Prioritization

- Need to create prioritized I/O queues
- Retain NUMA-friendly path

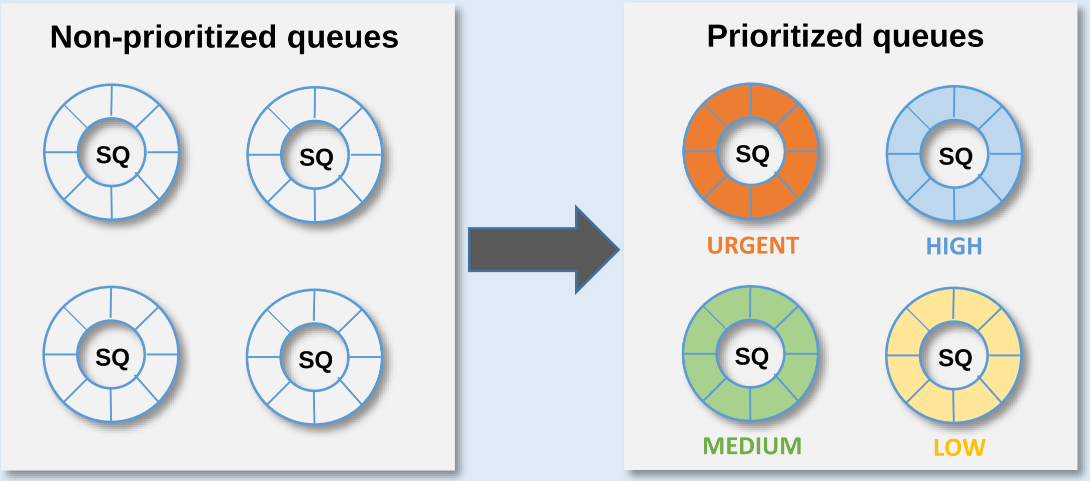
I/O classification

- How application can specify I/O service?
- Per-application or per I/O?

WRR Support Requirements

I/O Prioritization

- Need to create prioritized I/O queues
- Retain NUMA-friendly path



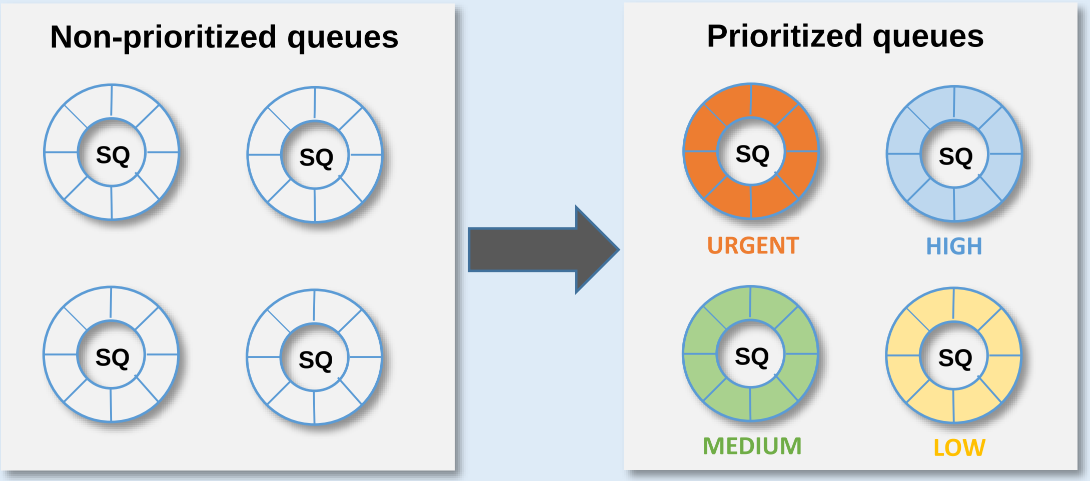
I/O classification

- How application can specify I/O service?
- Per-application or per I/O?

WRR Support Requirements

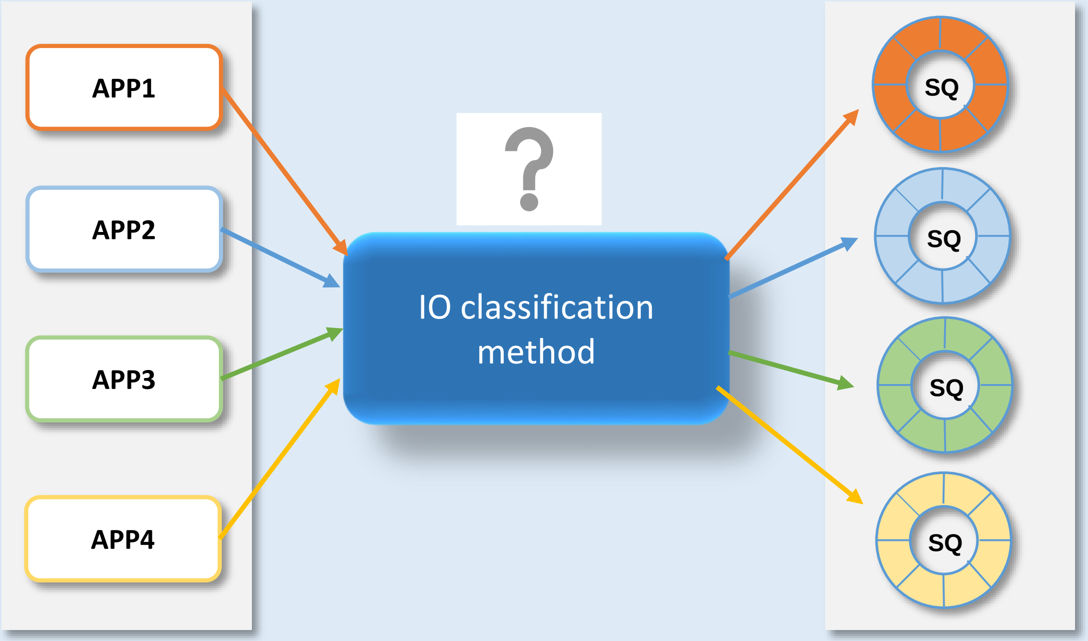
I/O Prioritization

- Need to create prioritized I/O queues
- Retain NUMA-friendly path



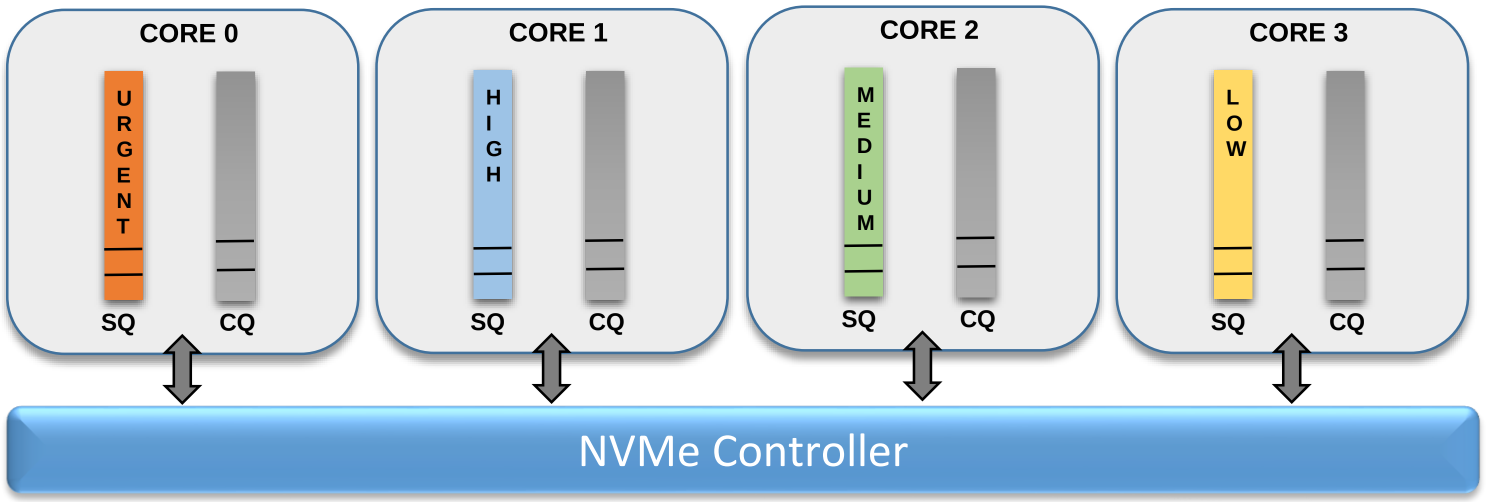
I/O classification

- How application can specify I/O service?
- Per-application or per I/O?



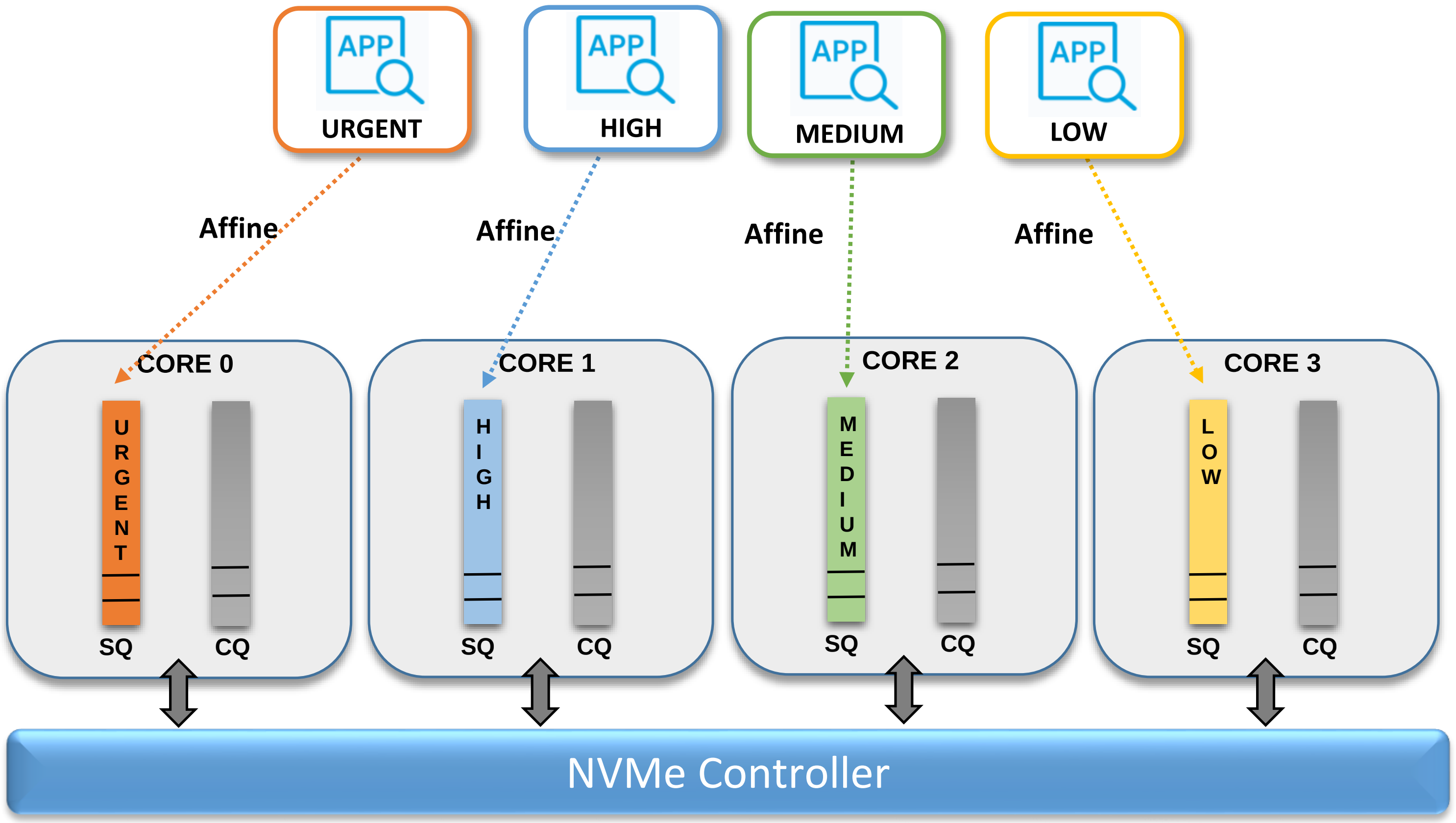
Affinity-based Method

- **Prioritization method:** Each core hosts one type of submission queue (1:1 mapping)
- **Classification method:** Affine applications to particular core(s)



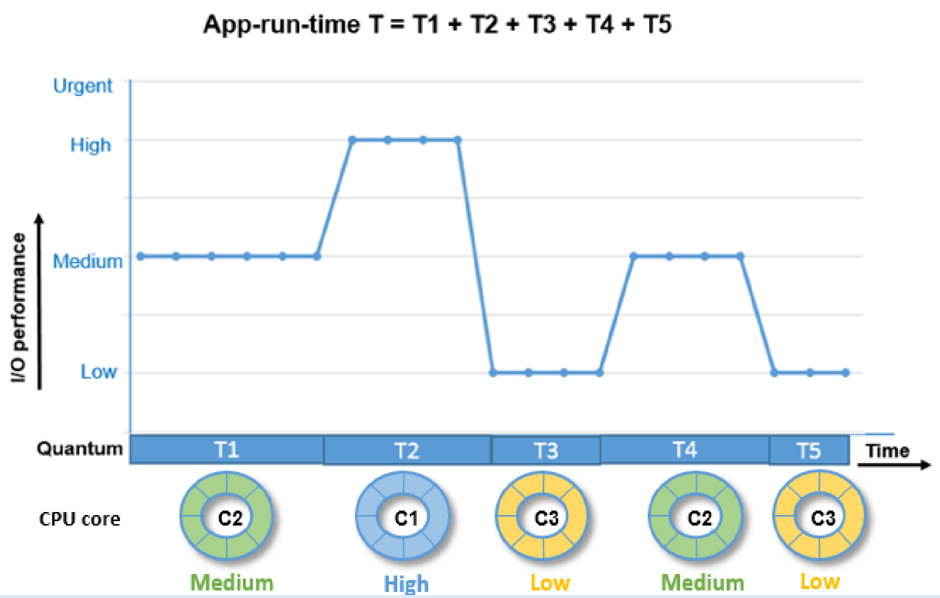
Affinity-based Method

- **Prioritization method:** Each core hosts one type of submission queue (1:1 mapping)
- **Classification method:** Affine applications to particular core(s)



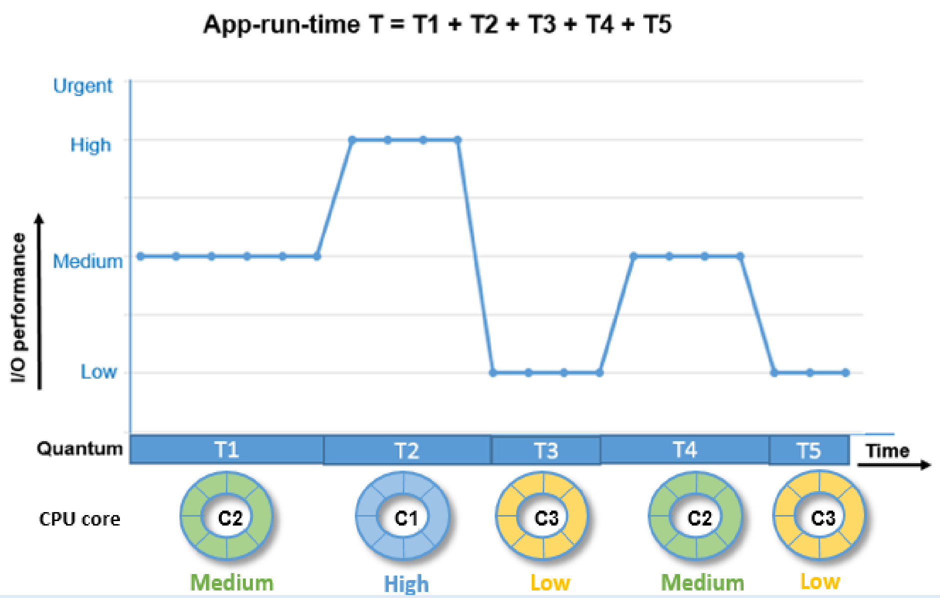
Drawbacks

- **All running applications must be affined (Arbitrary I/O performance otherwise)**

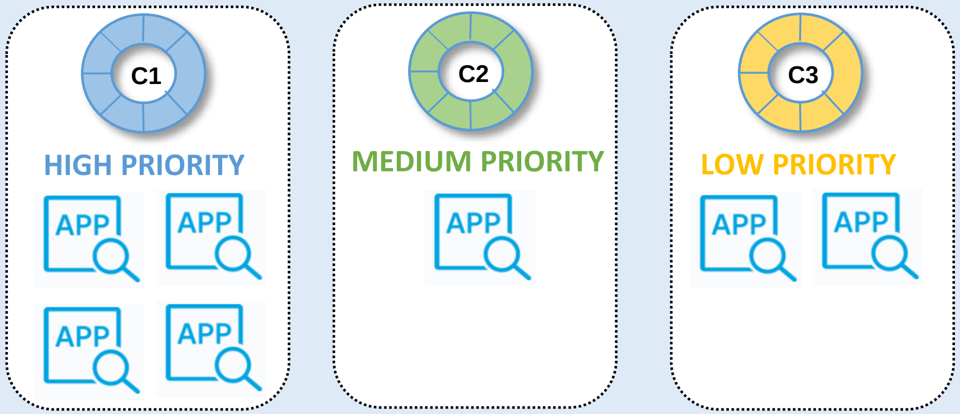


Drawbacks

- All running applications must be affined (Arbitrary I/O performance otherwise)



- Reduction in compute-ability
- Mandatory affinity leading to asymmetric core-utilization



Proposed Method: I/O Priority-based

I/O Prioritization

- Create prioritized I/O queues on each core
- Retain NUMA-friendly path

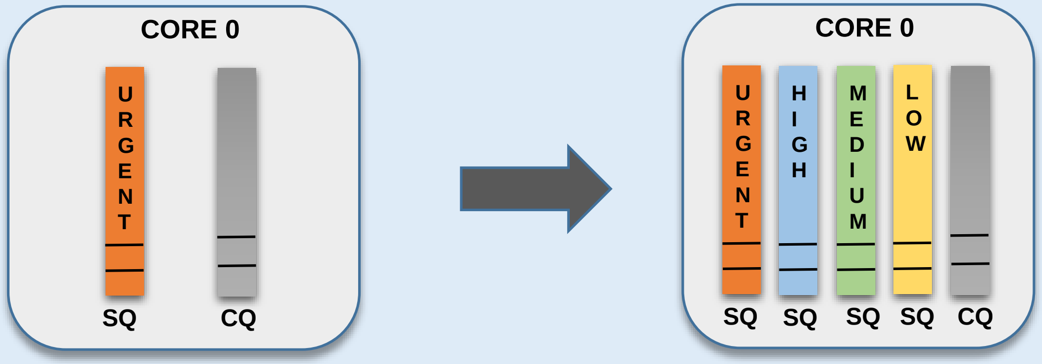
I/O Classification

- Link NVMe priorities to existing I/O priority classes
- Per-application

Proposed Method: I/O Priority-based

I/O Prioritization

- Create prioritized I/O queues on each core
- Retain NUMA-friendly path



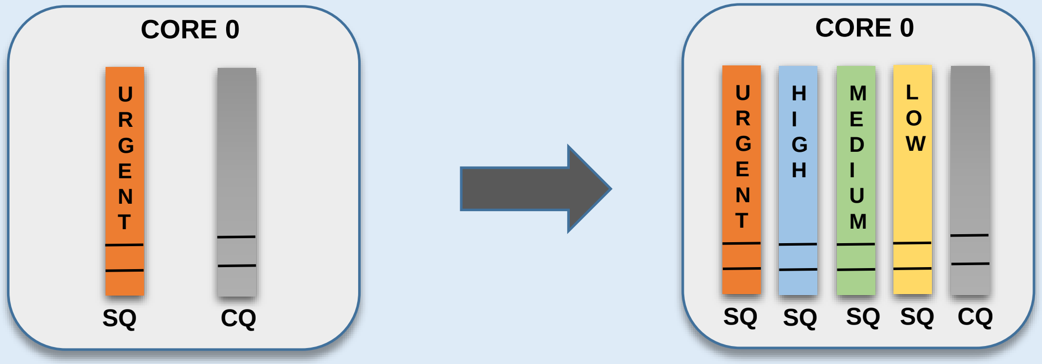
I/O Classification

- Link NVMe priorities to existing I/O priority classes
- Per-application

Proposed Method: I/O Priority-based

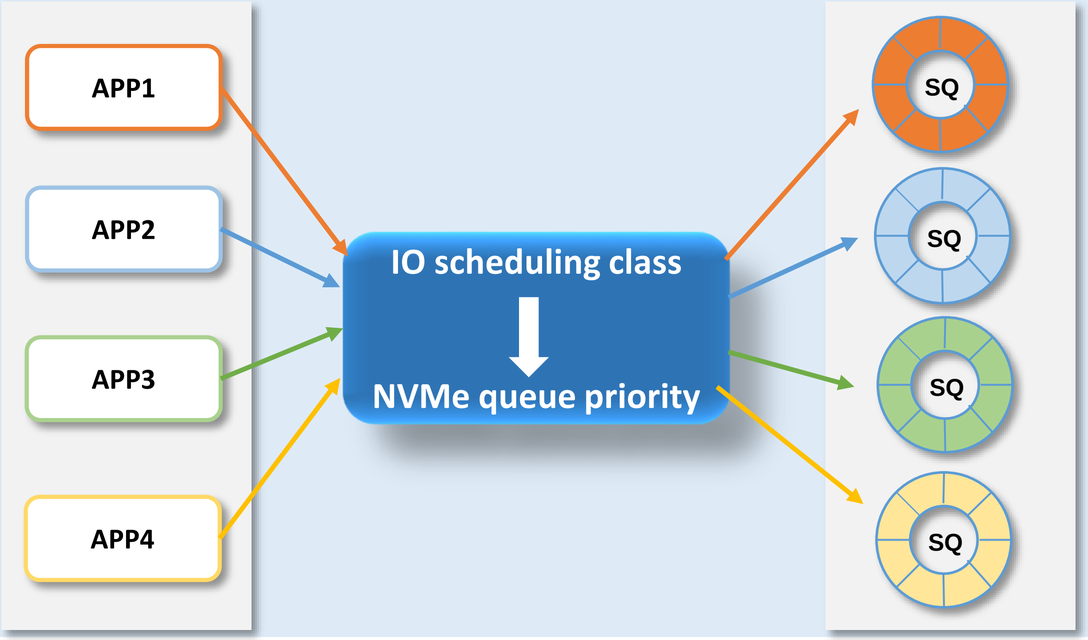
I/O Prioritization

- Create prioritized I/O queues on each core
- Retain NUMA-friendly path



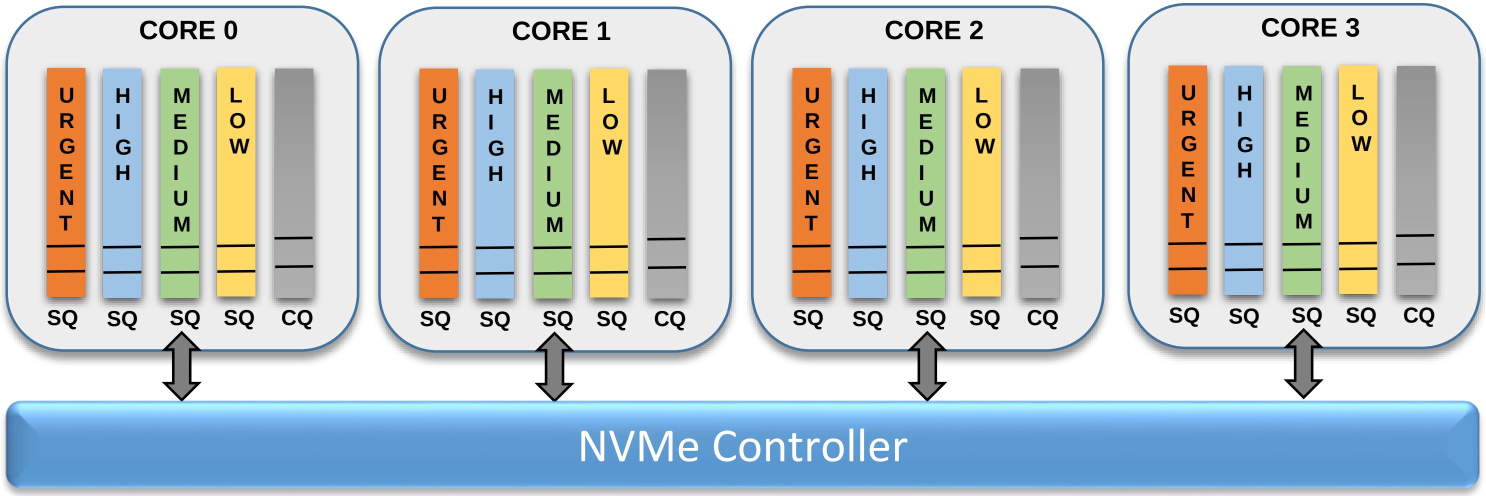
I/O Classification

- Link NVMe priorities to existing I/O priority classes
- Per-application



I/O Priority-based Method

- **Prioritization Method:** Each core hosts four type of submission queues (4:1 mapping)
- **Classification Method:** Reuse existing I/O scheduling classes

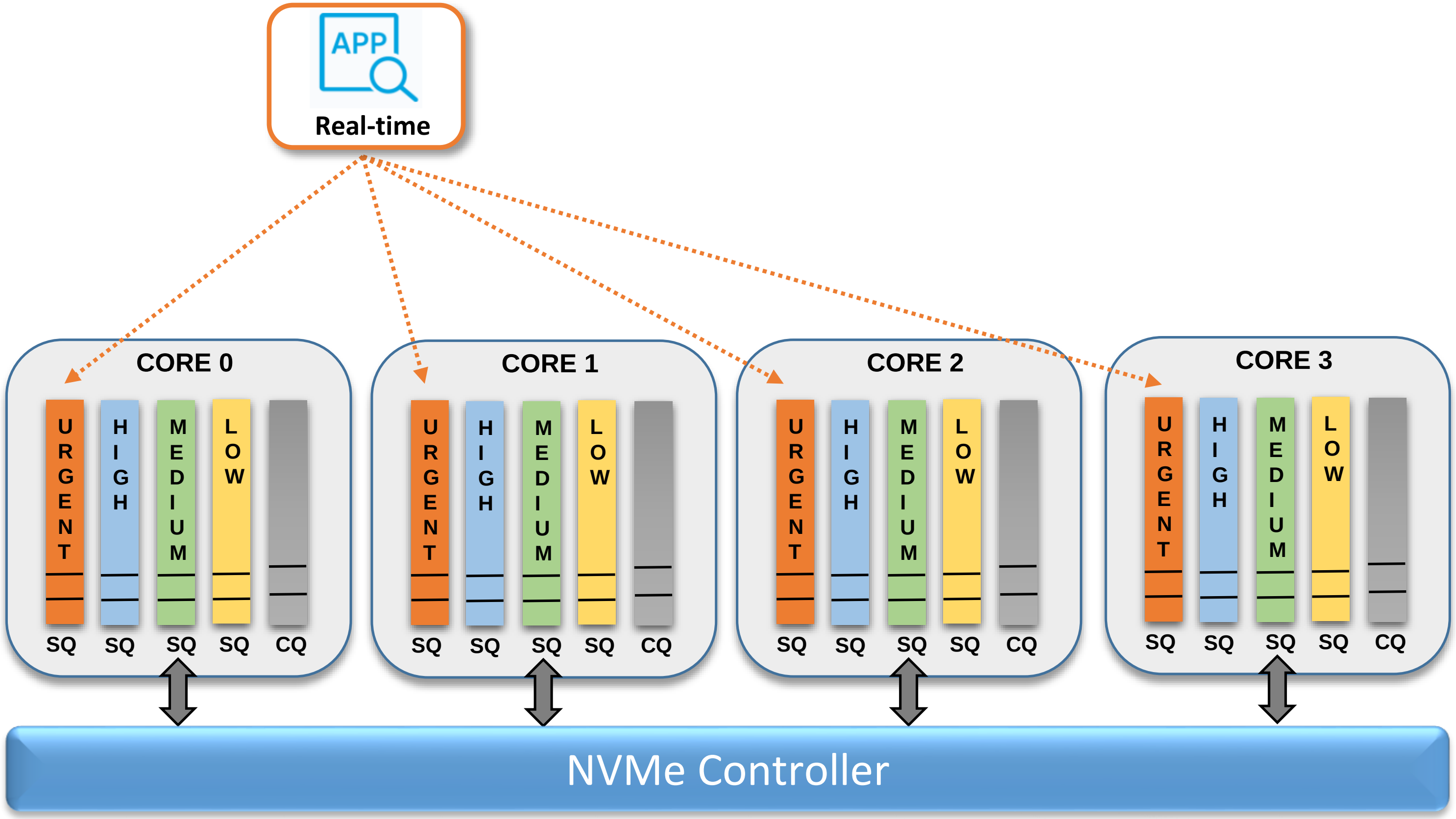


✓ **Compute-ability unaffected**

✓ **Does not require modifying applications**

I/O Priority-based Method

- **Prioritization Method:** Each core hosts four type of submission queues (4:1 mapping)
- **Classification Method:** Reuse existing I/O scheduling classes



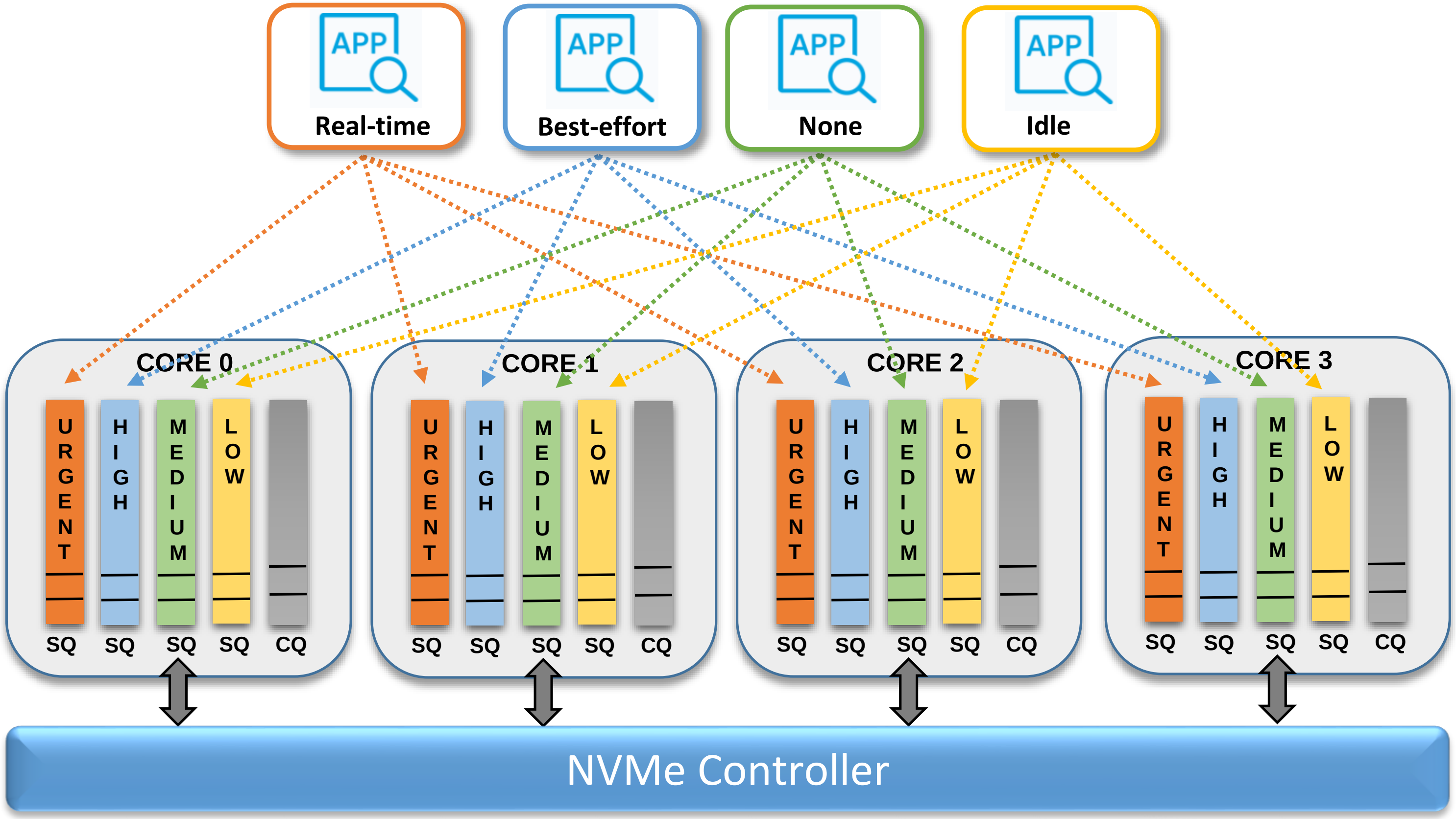
✓ **Compute-ability unaffected**

✓ **Does not require modifying applications**

I/O Priority-based Method

SSIR SAMSUNG SEMICONDUCTOR INDIA R&D CENTER

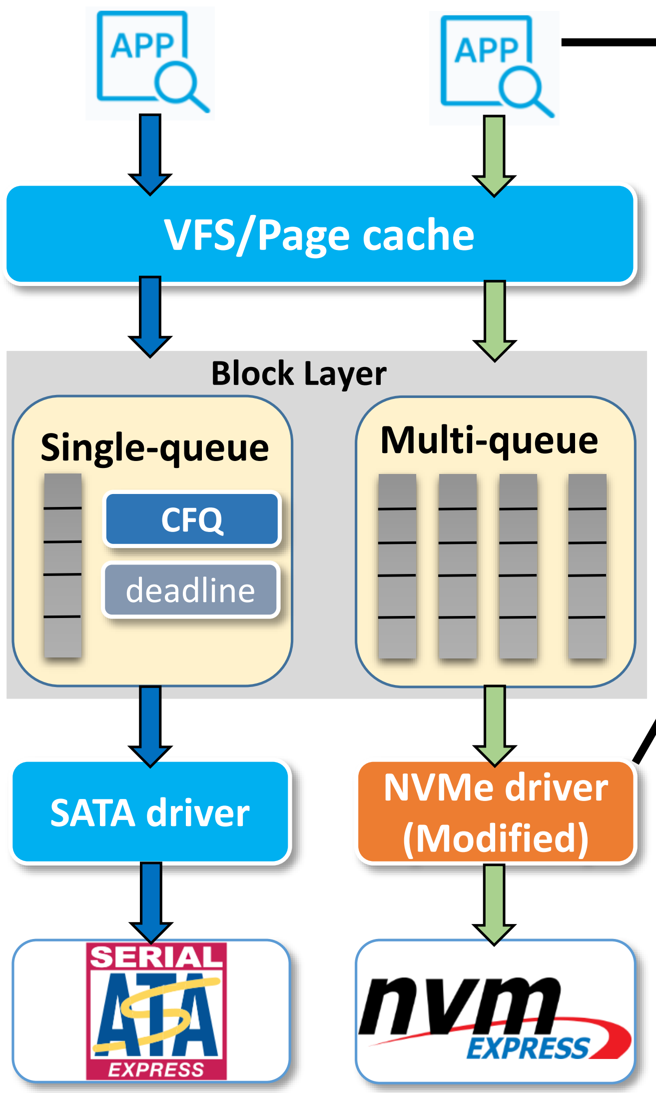
- **Prioritization Method:** Each core hosts four type of submission queues (4:1 mapping)
- **Classification Method:** Reuse existing I/O scheduling classes



✓ Compute-ability unaffected

✓ Does not require modifying applications

Modified NVMe Stack (4.10 Kernel)



- Specify io-priority class value while running (ionice)
- This is stored in io_context inside task_struct

- Obtain io-class value from io_context (or from request)
- Map io-class to queue-priority value and place command in corresponding SQ

Real-time	Urgent
Best-effort	High
None	Medium
Idle	Low

Ionice example on NVMe

Best-effort

```
[root@localhost fio]# ionice -c 2 fio randread
random-read: (g=0): rw=randrw, bs=4K-4K/4K-4K, ioengine=libaio, iodepth=64
...
random-read: (g=0): rw=randrw, bs=4K-4K/4K-4K, ioengine=libaio, iodepth=64
fio 1.59
Starting 4 processes
Jobs: 4 (f=4): [rrrr] [10.6% done] [841.7M/0K /s] [210K/0  iops] [eta 04m:29s]
```

High

210K

None

```
[root@localhost fio]# ionice -c 0 fio randread
random-read: (g=0): rw=randrw, bs=4K-4K/4K-4K, ioengine=libaio, iodepth=64
...
random-read: (g=0): rw=randrw, bs=4K-4K/4K-4K, ioengine=libaio, iodepth=64
fio 1.59
Starting 4 processes
Jobs: 4 (f=4): [rrrr] [10.3% done] [572.7M/0K /s] [143K/0  iops] [eta 04m:30s]
```

Medium

143K

Idle

```
[root@localhost fio]# ionice -c 3 fio randread
random-read: (g=0): rw=randrw, bs=4K-4K/4K-4K, ioengine=libaio, iodepth=64
...
random-read: (g=0): rw=randrw, bs=4K-4K/4K-4K, ioengine=libaio, iodepth=64
fio 1.59
Starting 4 processes
Jobs: 4 (f=4): [rrrr] [10.0% done] [303.2M/0K /s] [75.8K/0  iops] [eta 04m:31s]
```

Low

75.8K

Experimental Setup

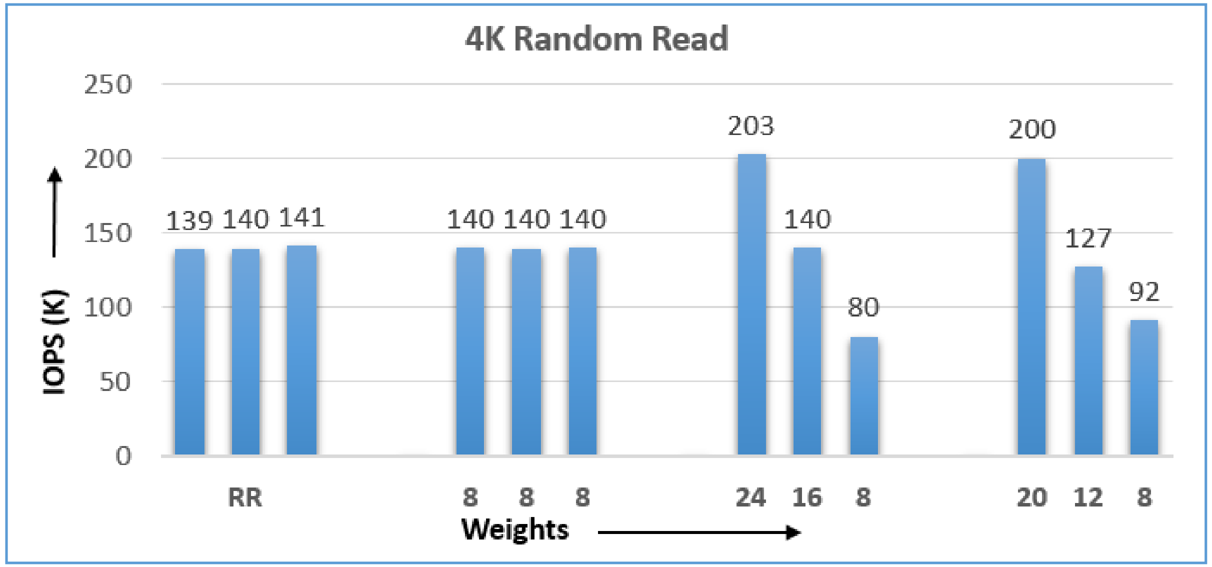
- Linux 4.10 Kernel
  (Modified NVMe Driver)

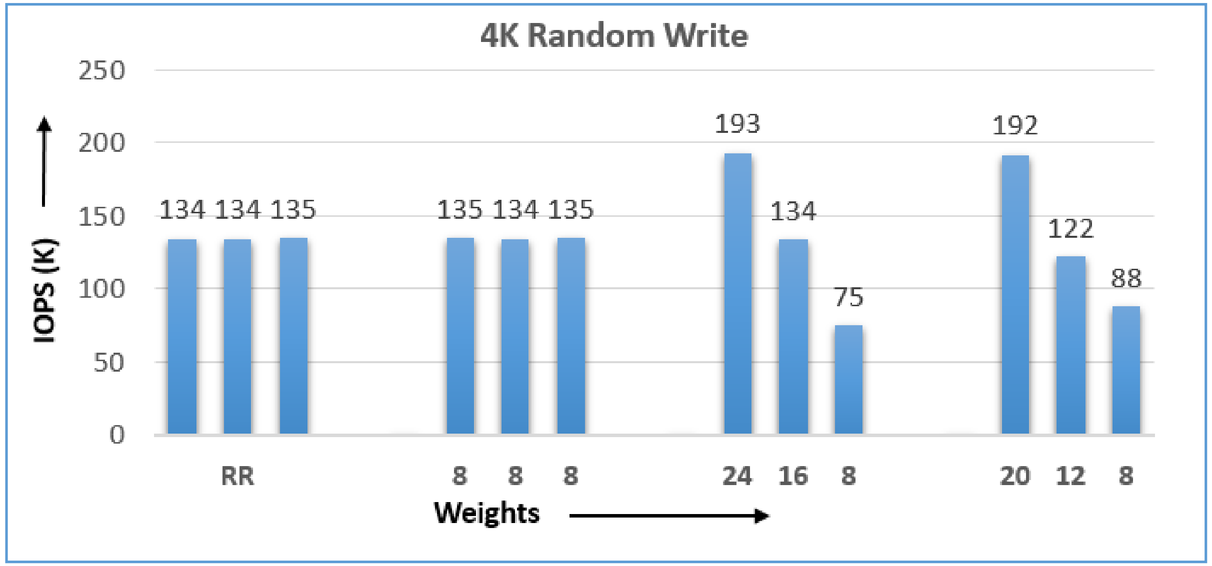
- Dell R720 server
  - 32 CPUs (2 NUMA nodes)
  - 32 GB RAM

- Samsung PM1725a SSD
  (With WRR arbitration)

Result #1

❑ IOPS distribution among 3 applications



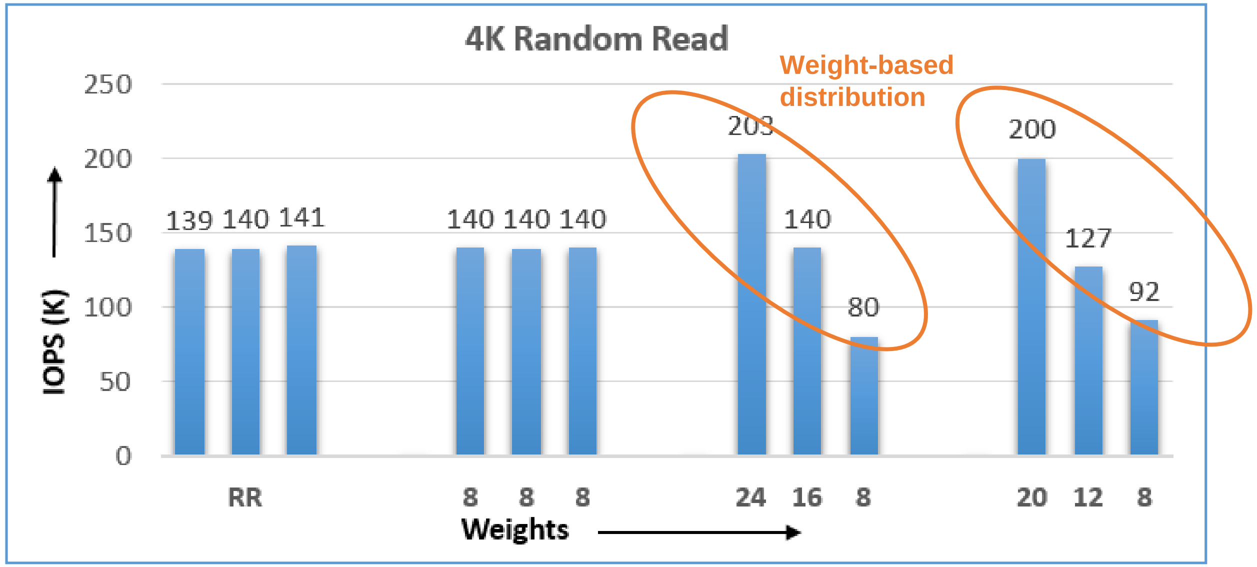


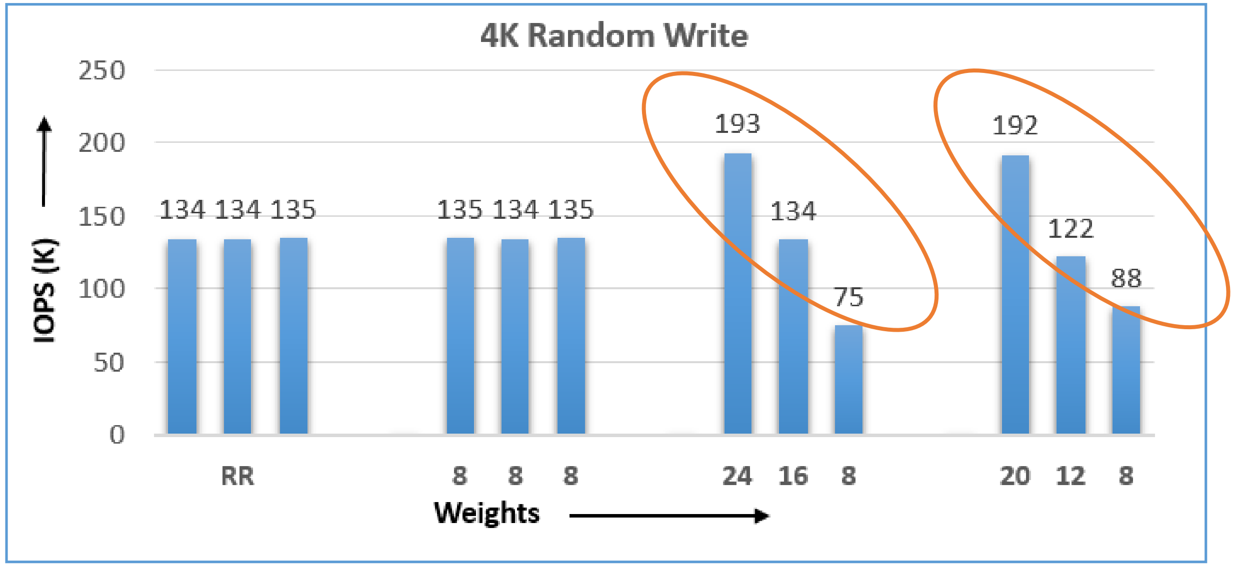
Application configuration
- 4 FIO jobs
- QD 64
- 4K record

Result #1

❑ IOPS distribution among 3 applications

Application configuration
- 4 FIO jobs
- QD 64
- 4K record

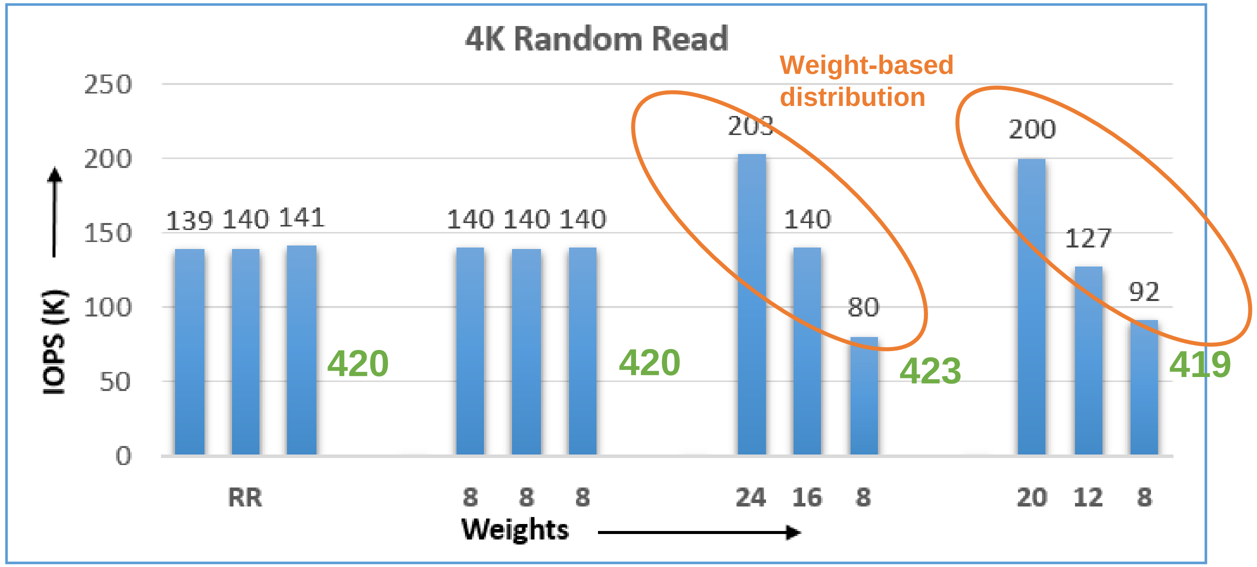


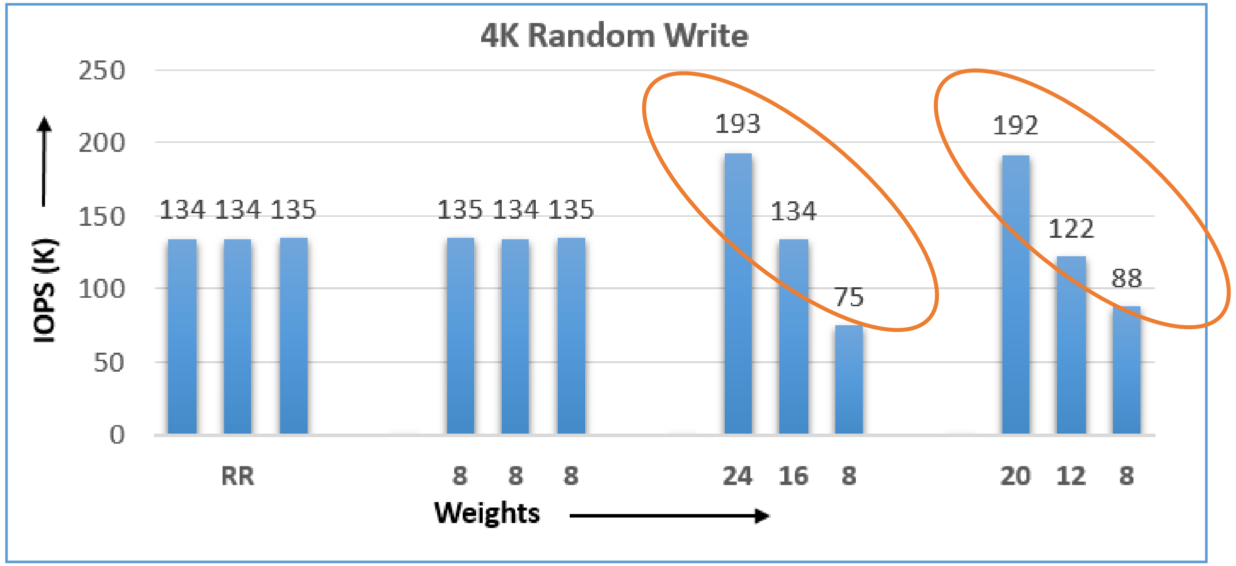


Result #1

❑ **IOPS distribution among 3 applications**

Application configuration
- 4 FIO jobs
- QD 64
- 4K record

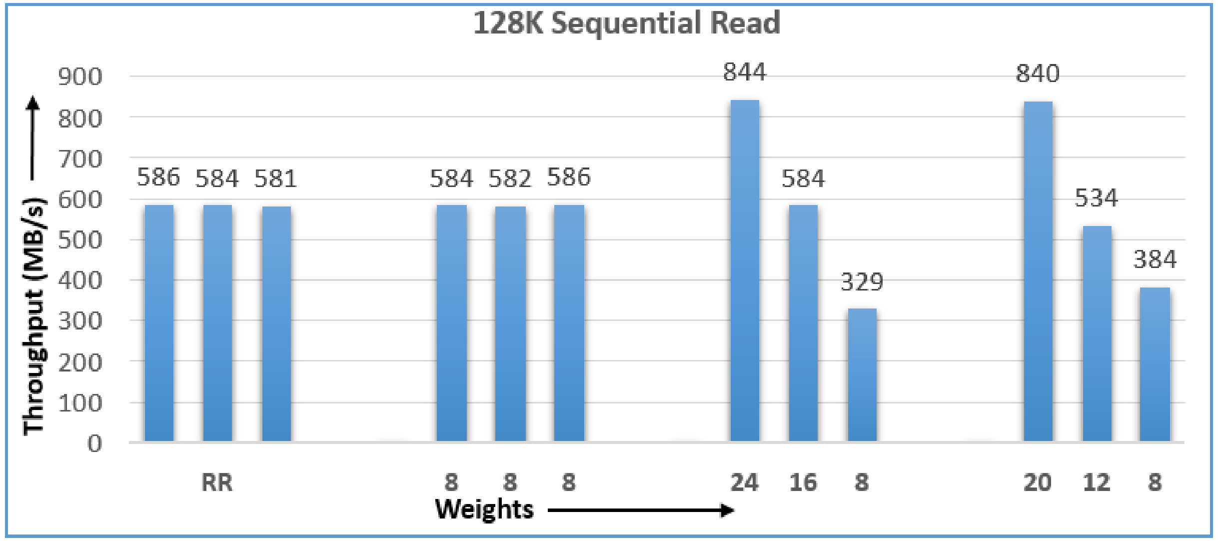


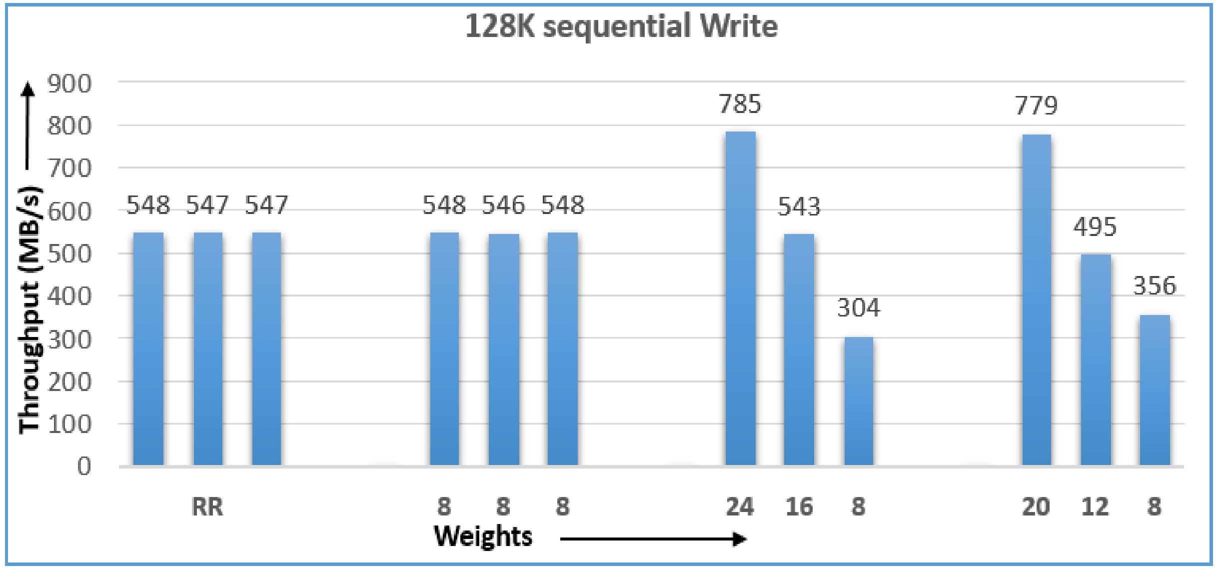


Result #2

- ❑ **Bandwidth distribution among 3 applications**

Application configuration
- 4 FIO jobs
- QD 64
- 128K record



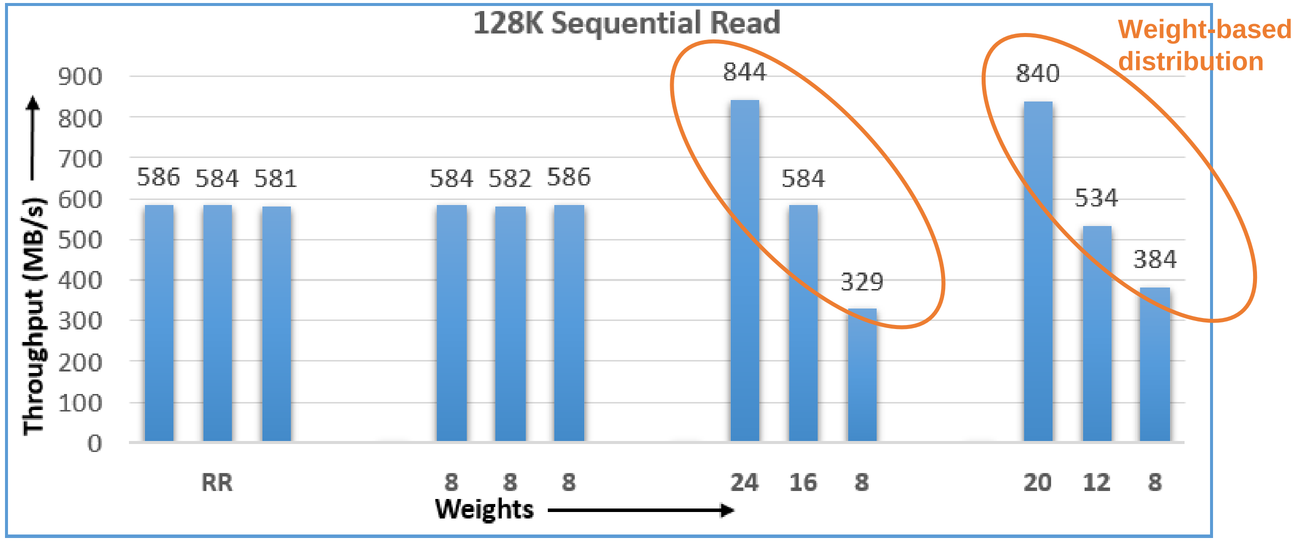


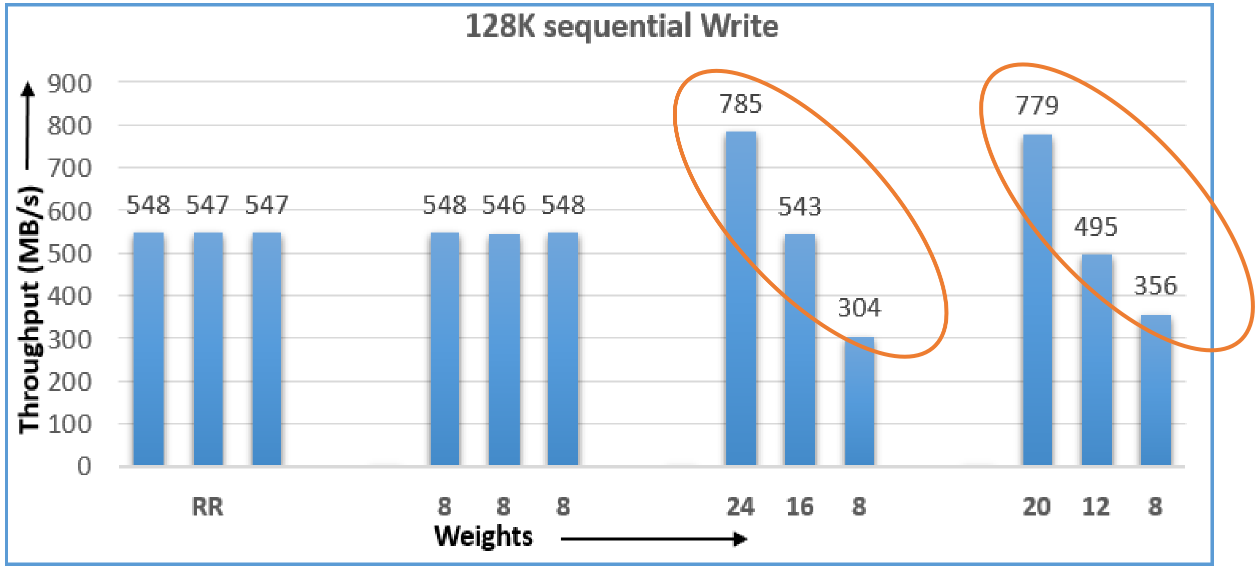
Result #2

- Bandwidth distribution among 3 applications

Application configuration

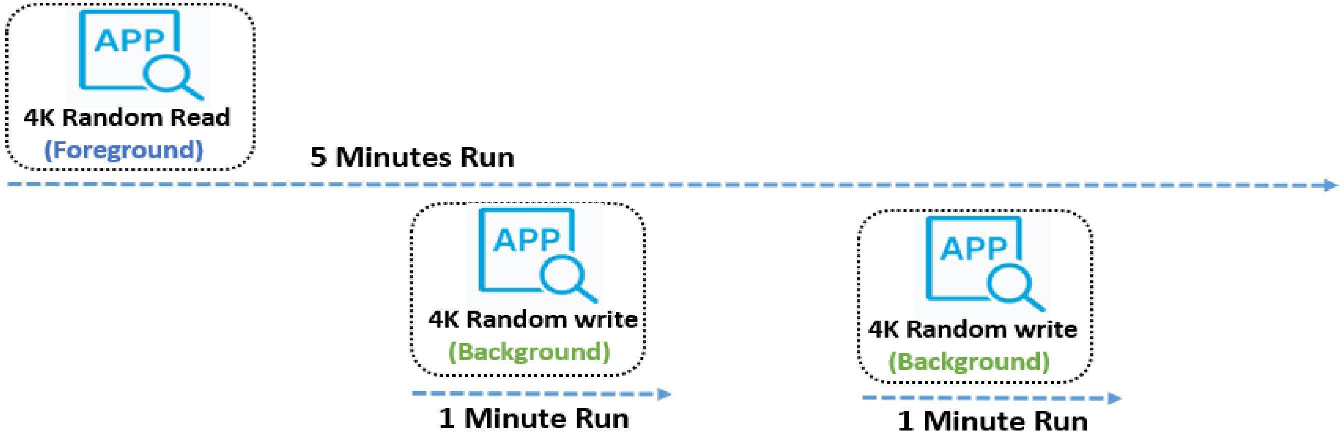
- 4 FIO jobs
- QD 64
- 128K record





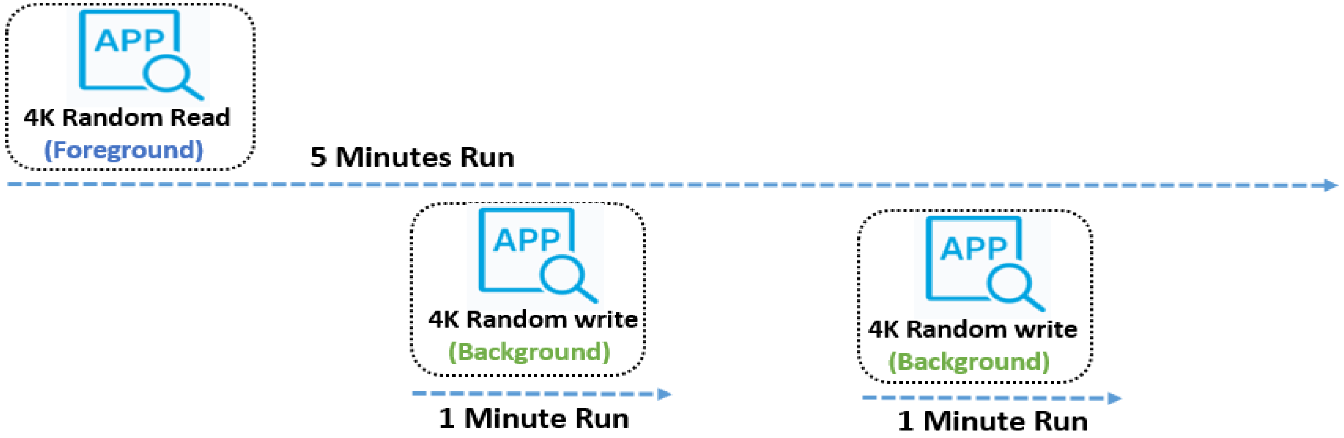
Result #3

- Foreground/Background IO control

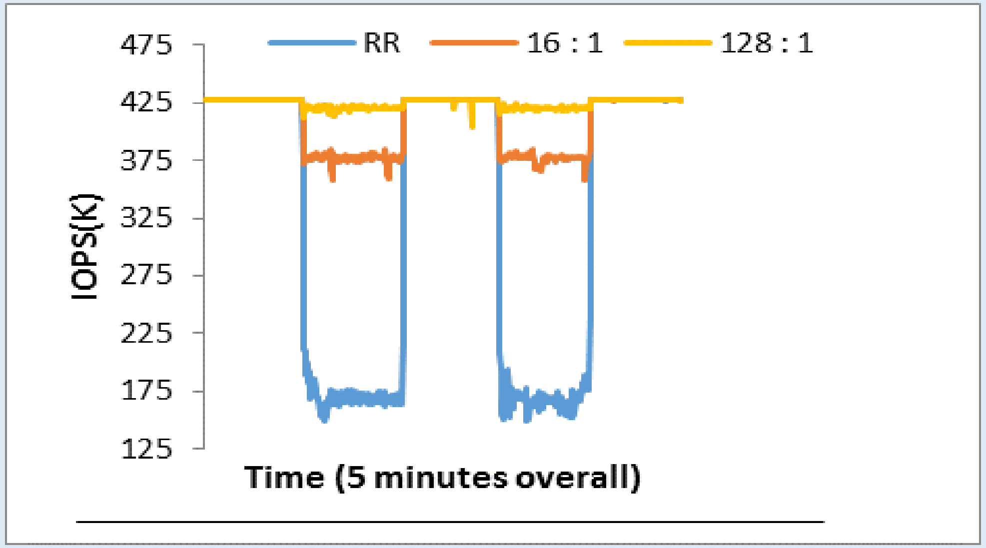


Result #3

Foreground/Background IO control



Foreground Read IOPS



- **RR mode**
  - Sharp decline in IOPS
  - Background process cannot be throttled
- **WRR mode**
  - Background process can be throttled
  - 16:1 = Throttle BG process
  - 128:1 = Further throttling. Retains foreground performance

Summary

- Differentiated I/O service for applications can be built using WRR arbitration

- Scheduler-independent prioritization: Applications get the advantage of the prioritization natively present inside the device

- Proposed method does not reduce compute-ability of applications

- By not introducing new interface/API, need of rebuilding application is avoided

- Future work
  - Kernel patch
  - Sysfs support for run-time WRR configuration

Acknowledgements

Rajesh Sahoo, Anshul Sharma, Sungyoung Ahn, Manoj Thapliyal, Vikram Singh, and Seunguk Shin