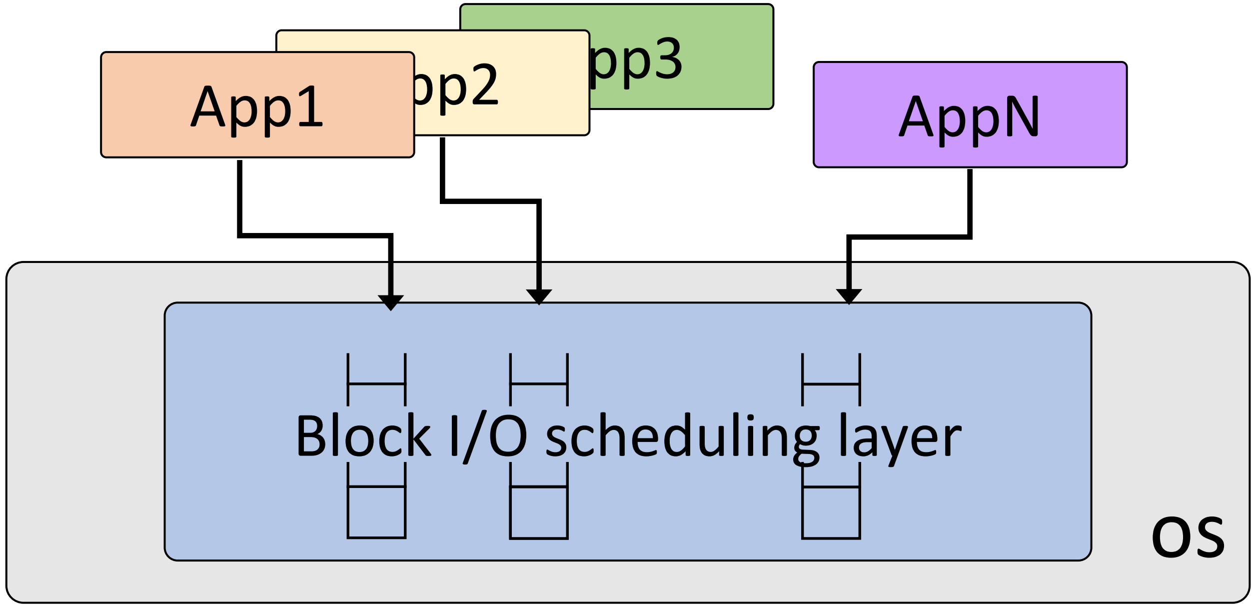
FAST 2021

D2FQ: Device-Direct Fair Queueing for NVMe SSDs

Jiwon Woo, Minwoo Ahn, Gyusun Lee and Jinkyu Jeong
Sungkyunkwan University (SKKU)
Computer Systems Laboratory

SUNG KYUN KWAN UNIVERSITY

Conventional I/O Scheduling

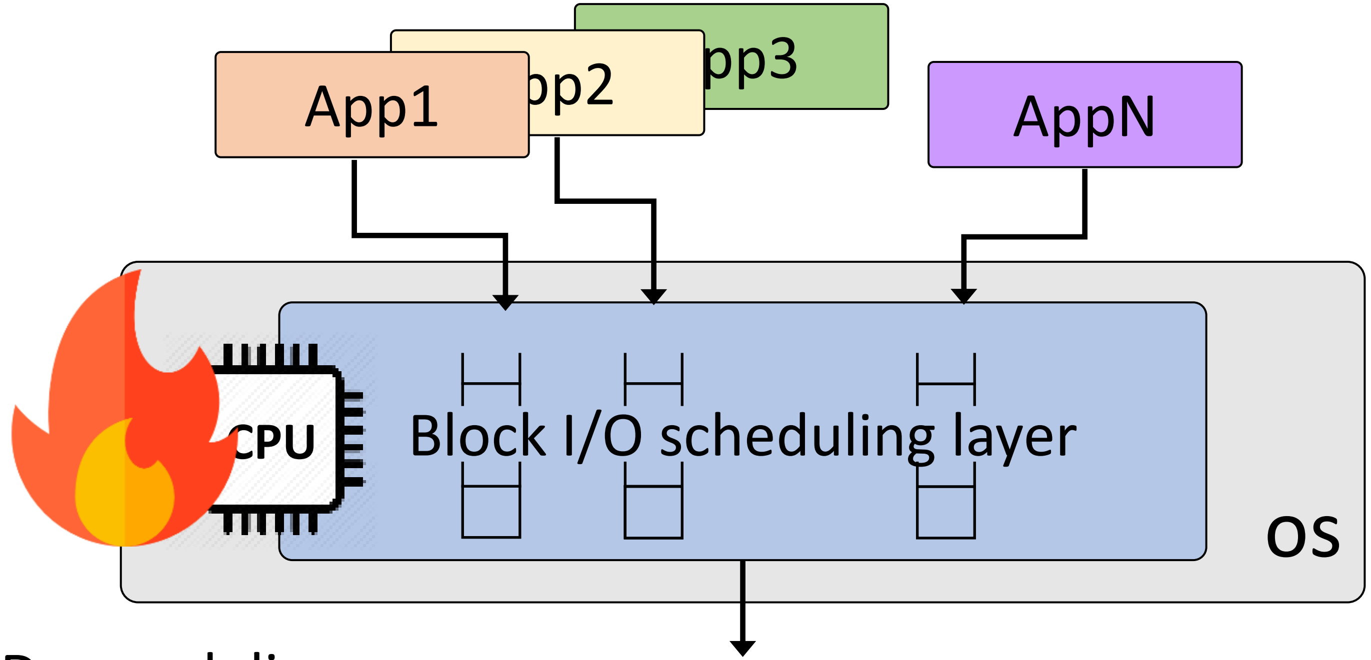


SSDs can deliver

Million IOPS

Being able to handle requests from multi-tenants

Conventional I/O Scheduling



SSDs can deliver
Million IOPS

CFQ [Linux]

BFQ [Linux]

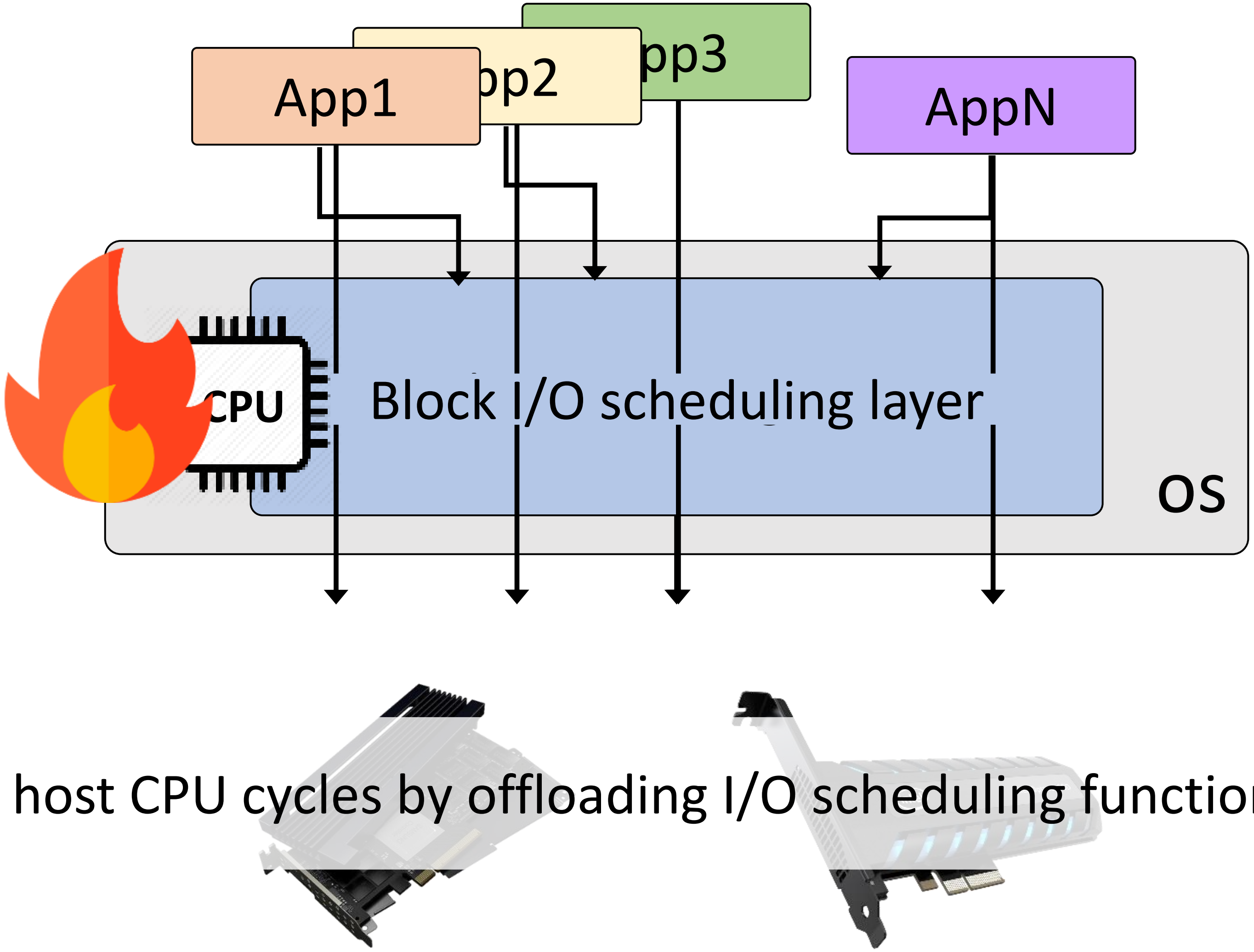
FlashFQ [ATC ‘13]

…

MQFQ [ATC ‘19]

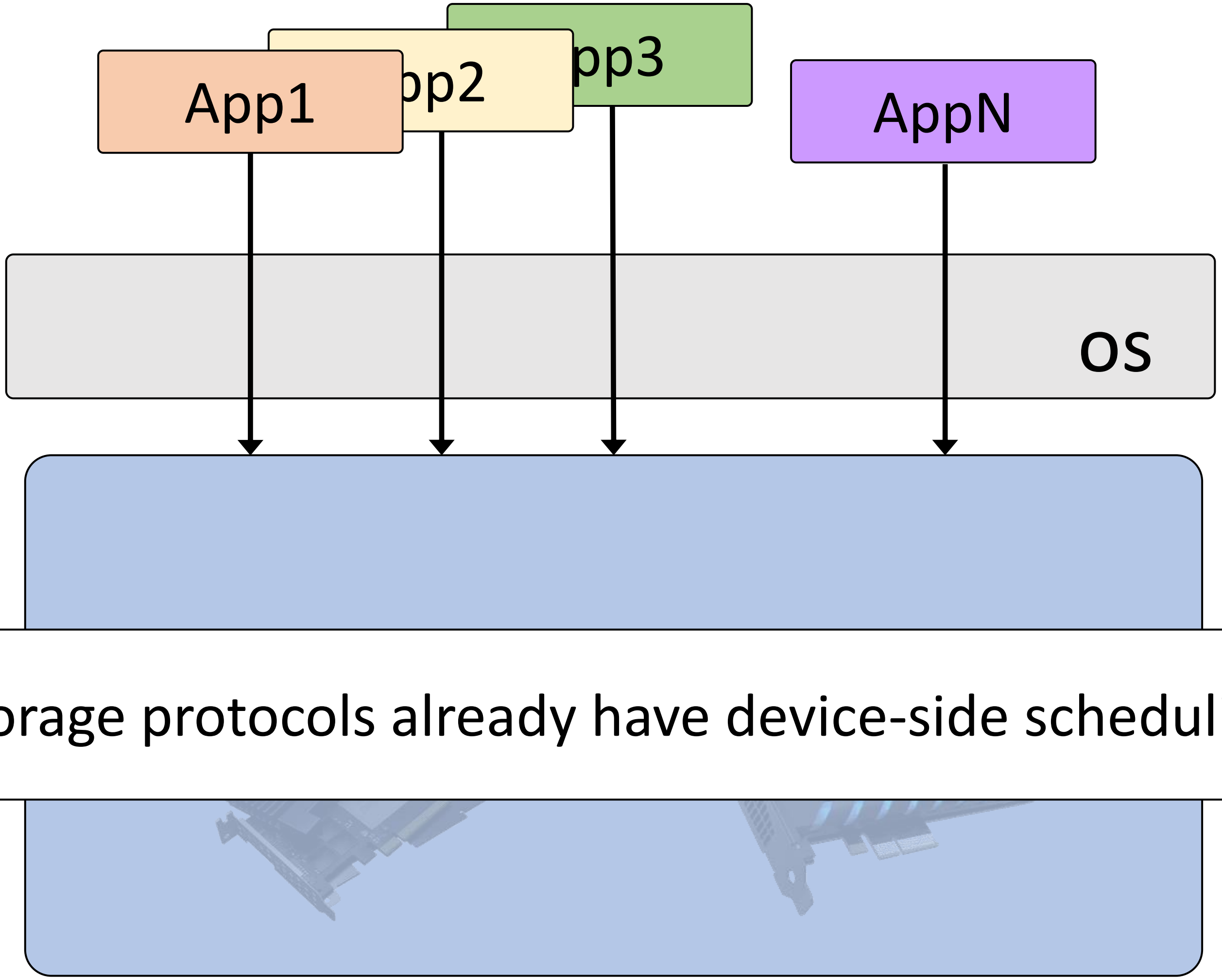
High CPU overhead

Device-side I/O Scheduling



Saving host CPU cycles by offloading I/O scheduling function to device

Device-side I/O Scheduling



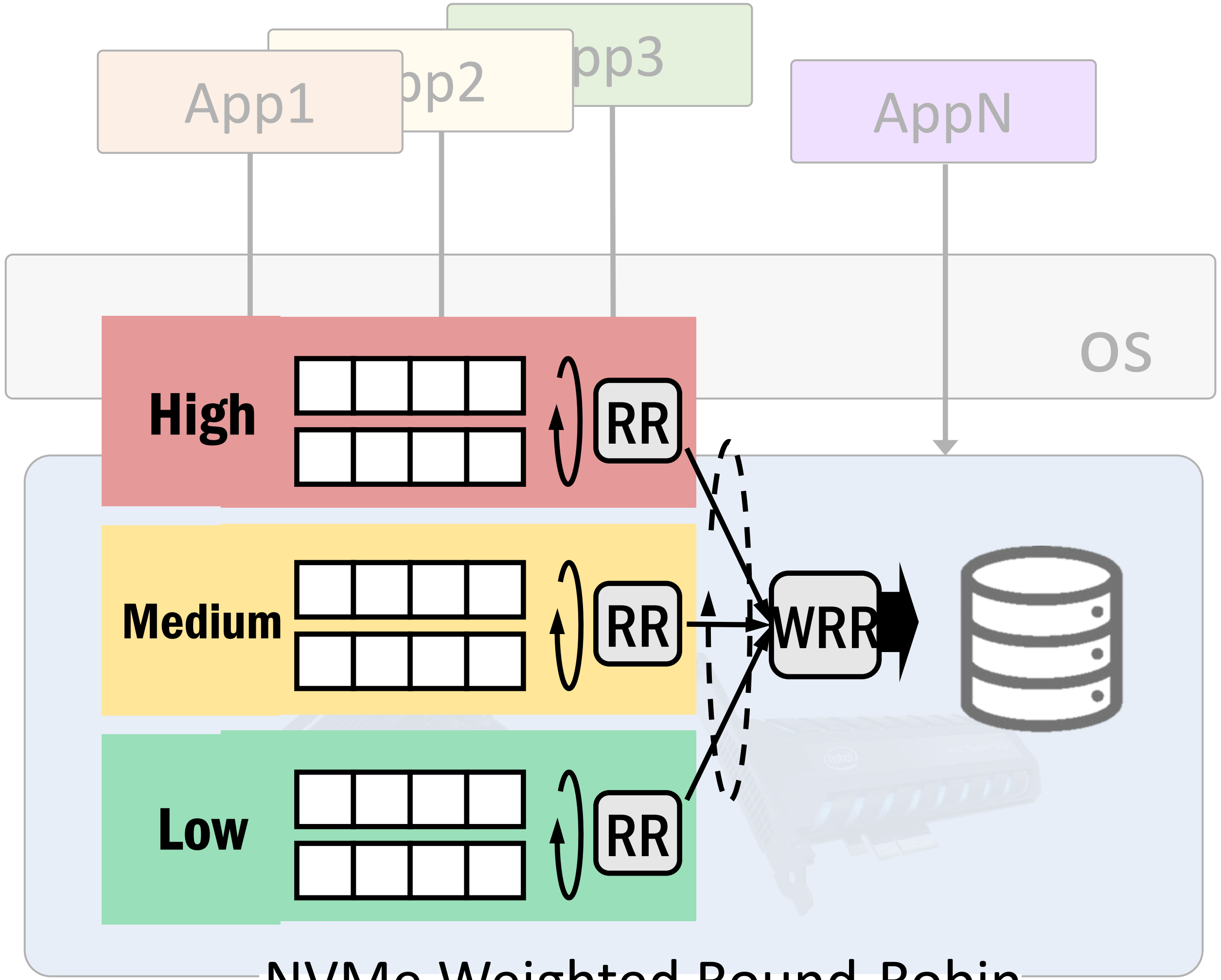
Some storage protocols already have device-side scheduling features

HIOS [SIGARCH ‘14]

FLIN [ISCA ‘18]

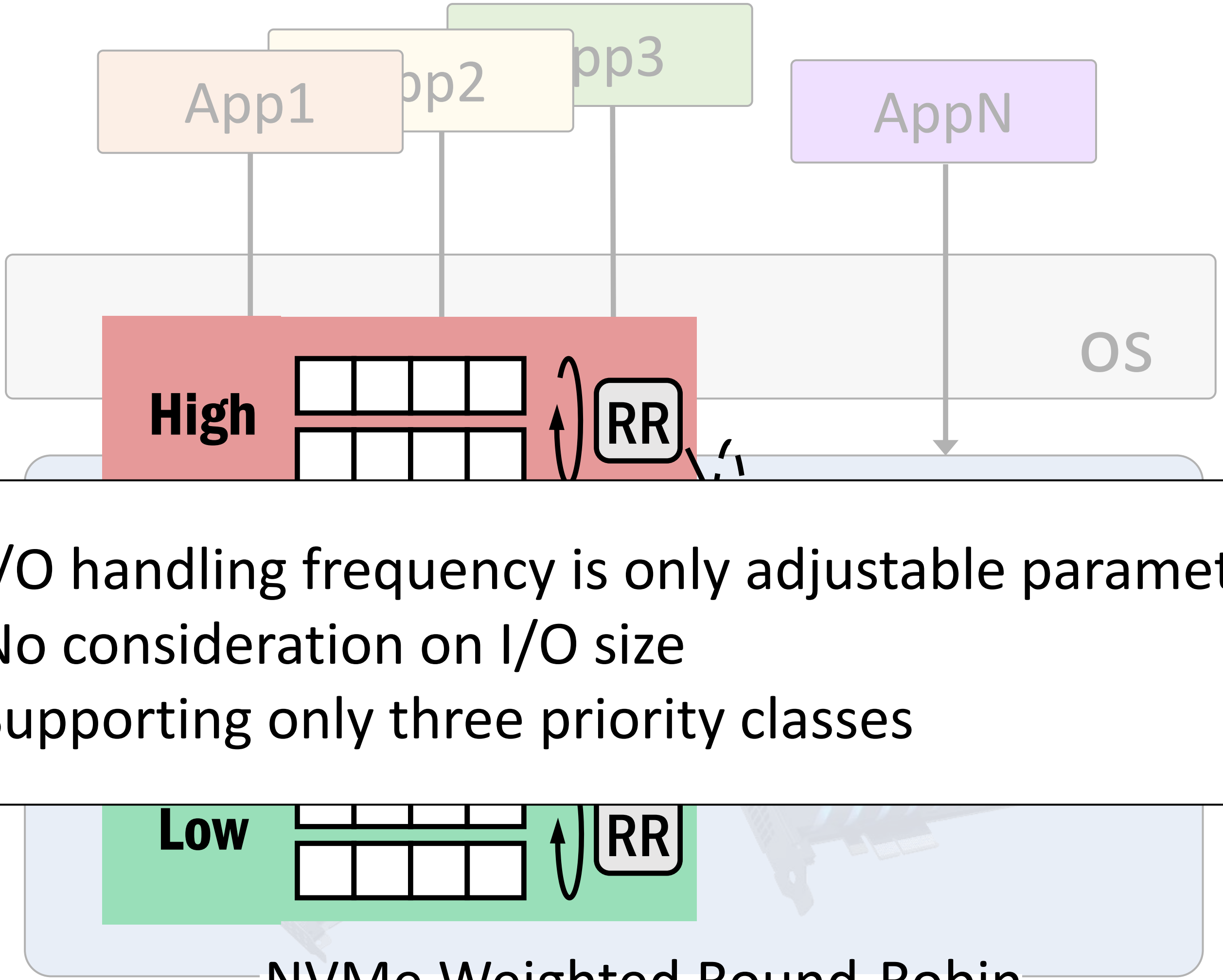
⋮

Device-side I/O Scheduling



NVMe Weighted Round-Robin

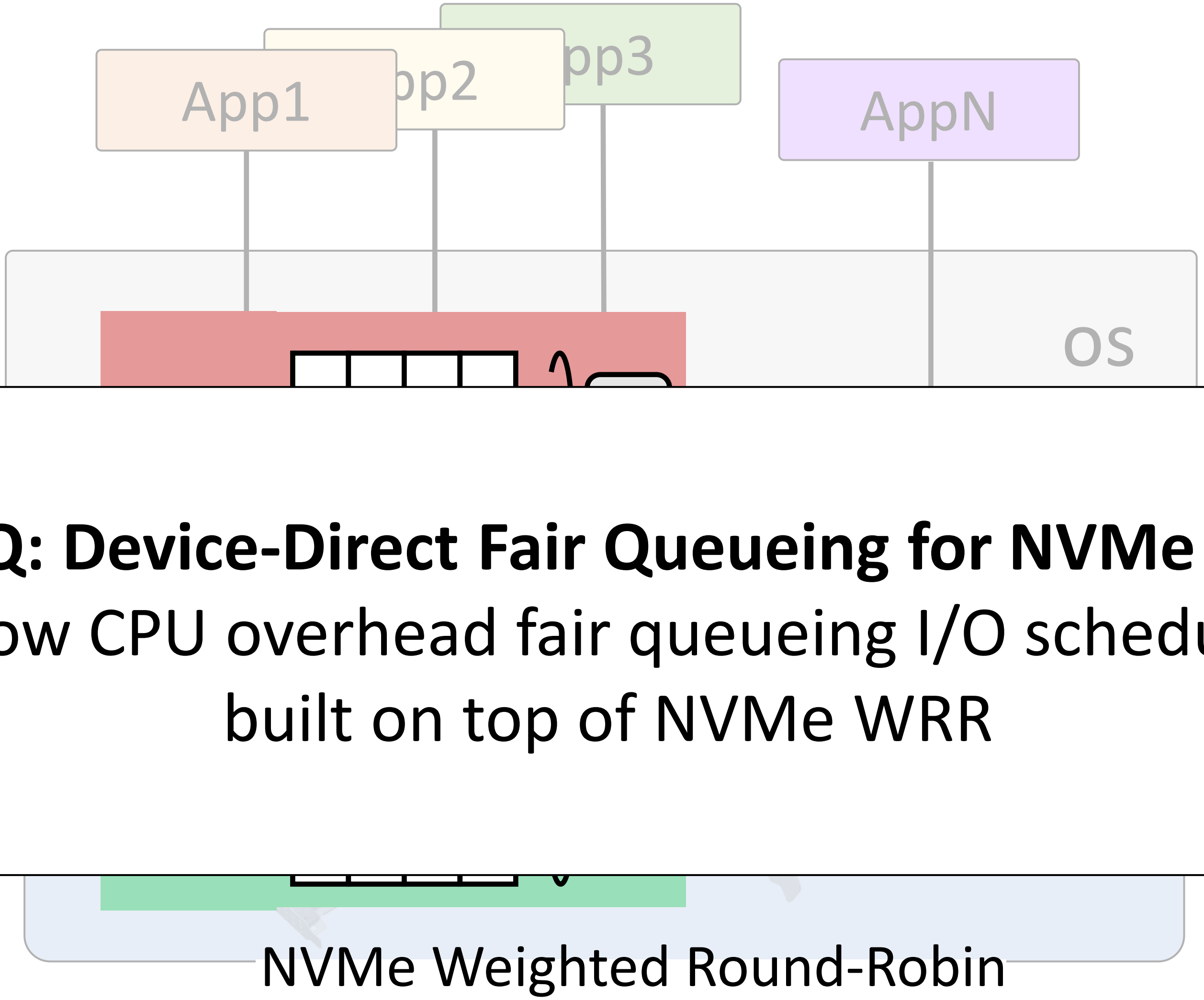
Device-side I/O Scheduling



1. I/O handling frequency is only adjustable parameter
2. No consideration on I/O size
3. Supporting only three priority classes

NVMe Weighted Round-Robin

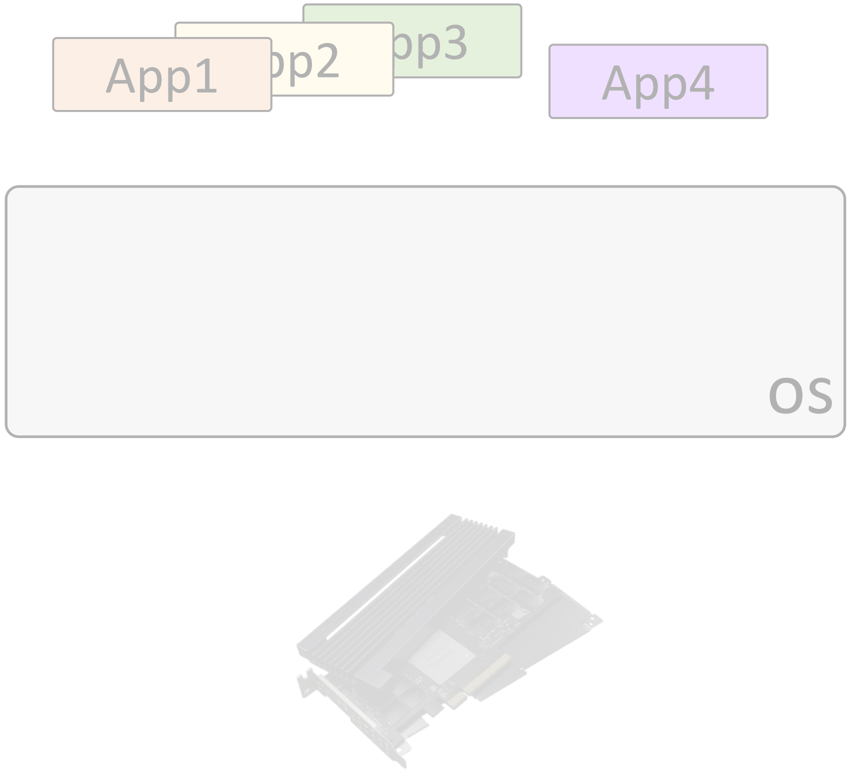
Our Approach



D2FQ: Device-Direct Fair Queueing for NVMe SSDs

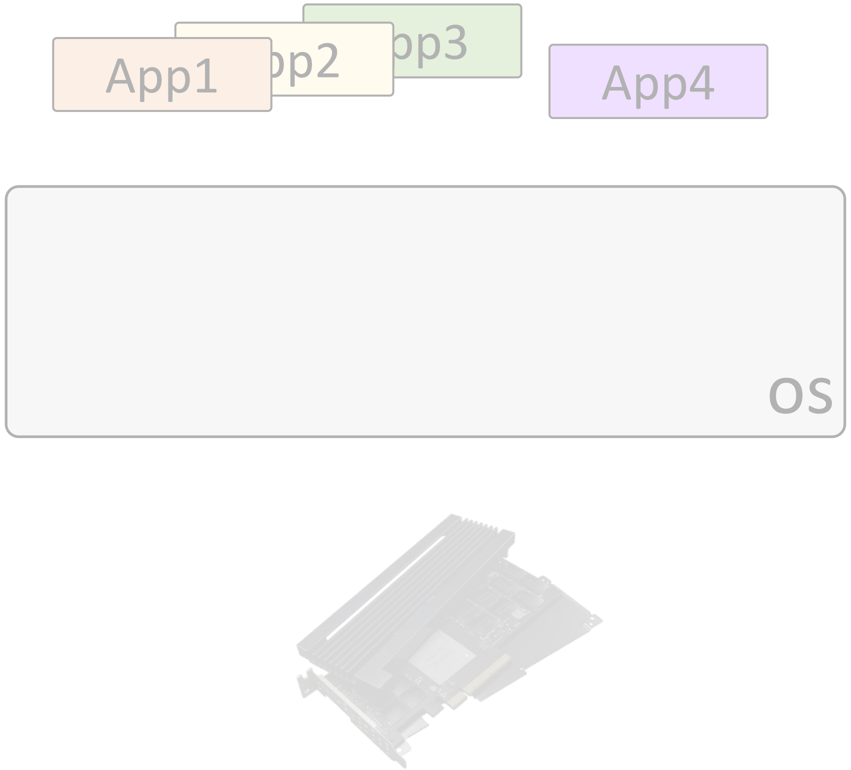
A low CPU overhead fair queueing I/O scheduler built on top of NVMe WRR

Virtual Time-based Fair Queueing

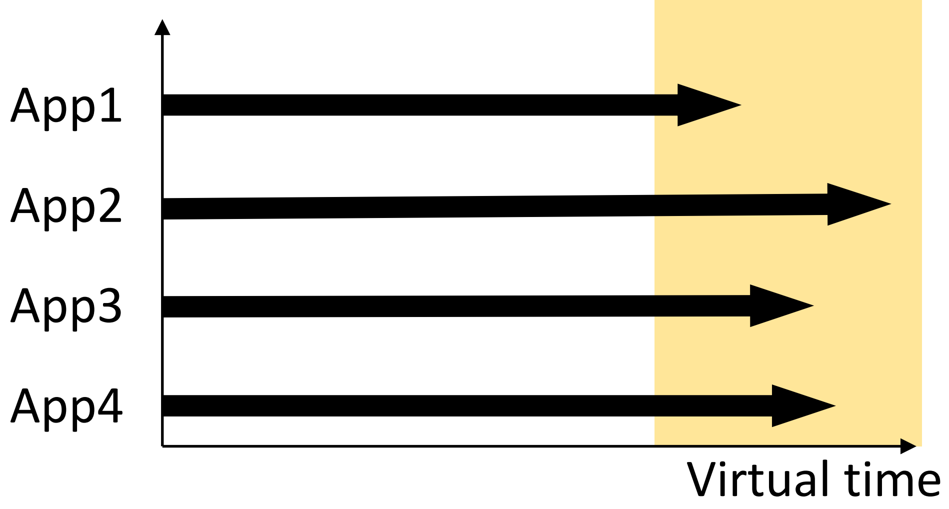


$$Virtual\ time = \frac{\sum I/O\ size_{completed}}{I/O\ weight}$$

Virtual Time-based Fair Queueing

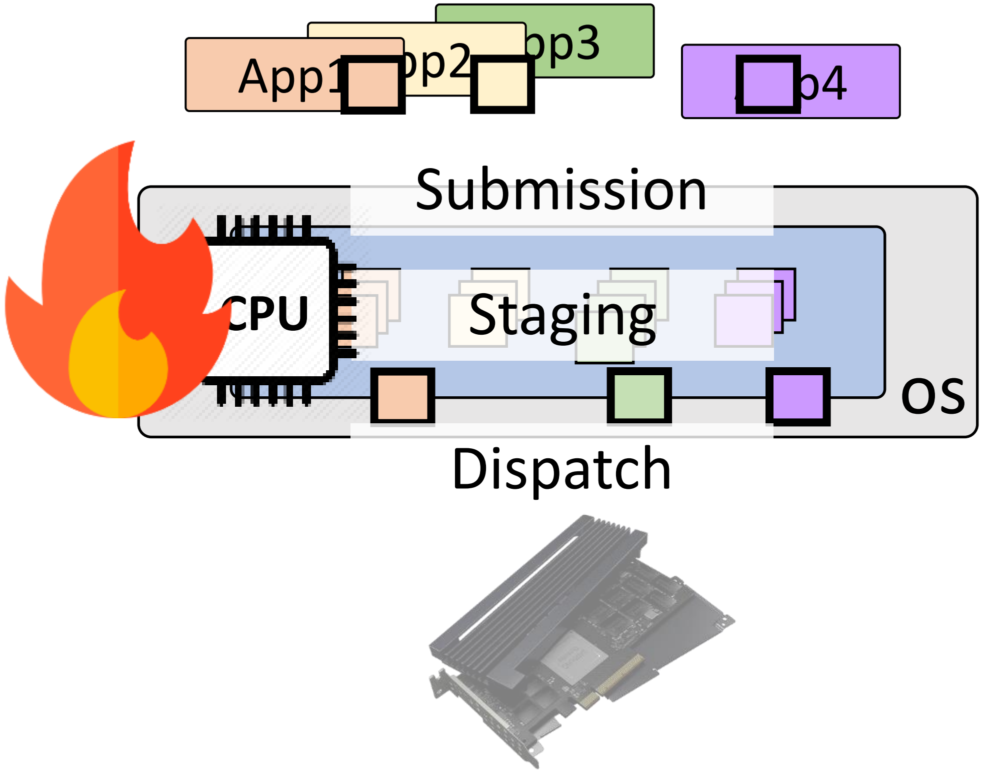


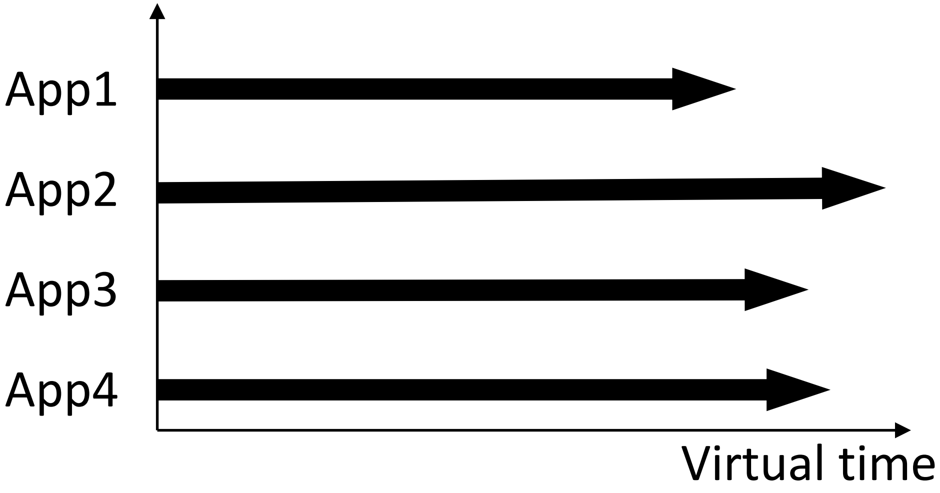
Satisfy fairness by equalizing virtual time of flows



$$Virtual\ time = \frac{\sum I/O\ size_{completed}}{I/O\ weight}$$

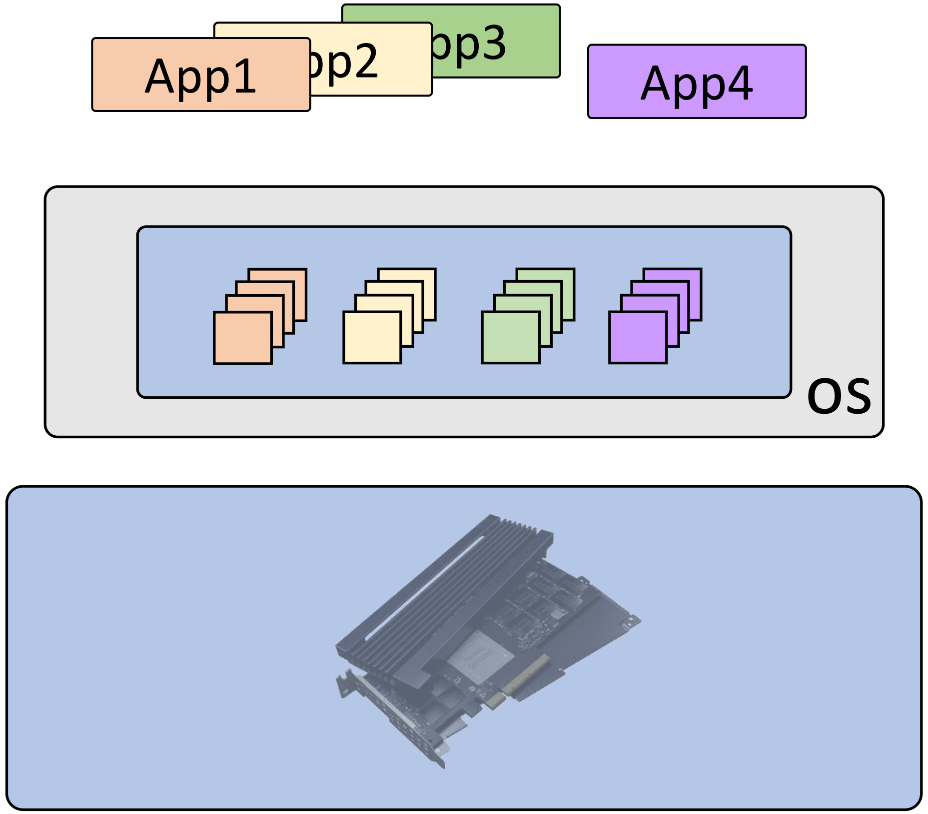
Virtual Time-based Fair Queueing

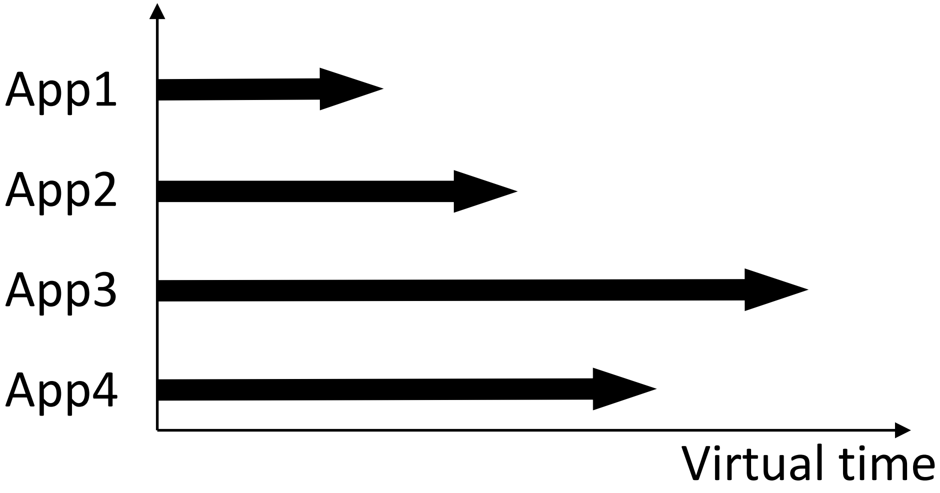




$$Virtual\ time = \frac{\sum I/O\ size_{completed}}{I/O\ weight}$$

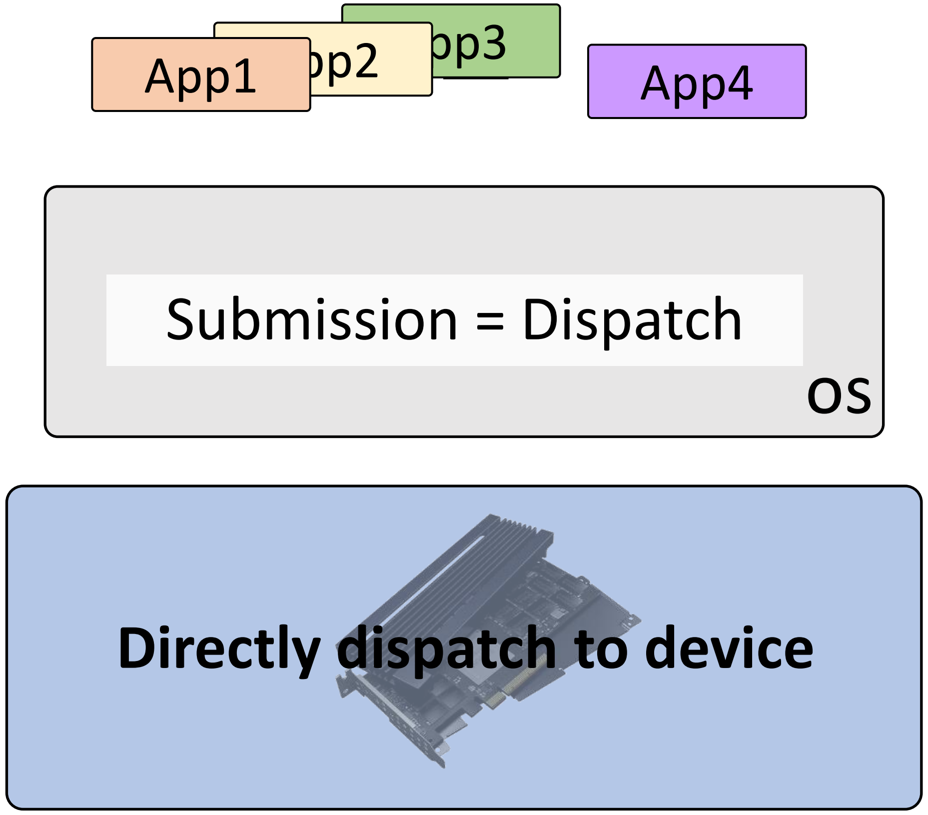
Virtual Time-based Fair Queueing – D2FQ

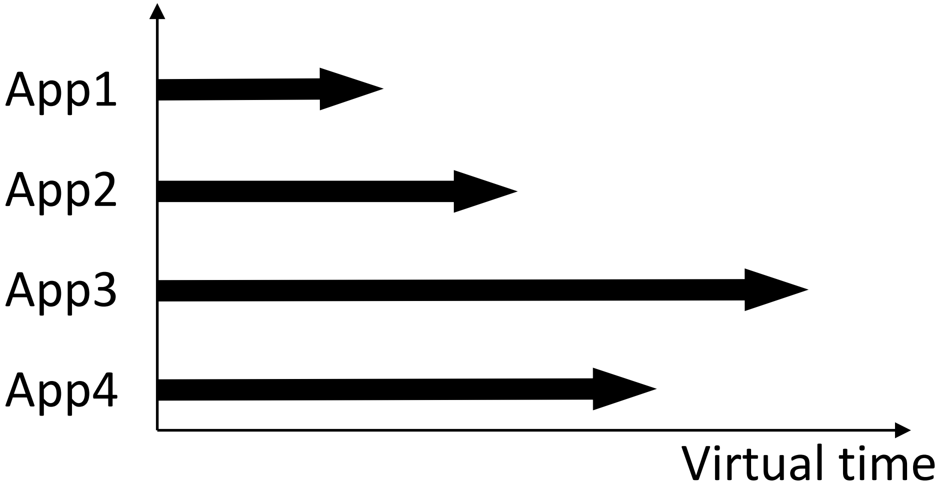




$$Virtual\ time = \frac{\sum I/O\ size_{completed}}{I/O\ weight}$$

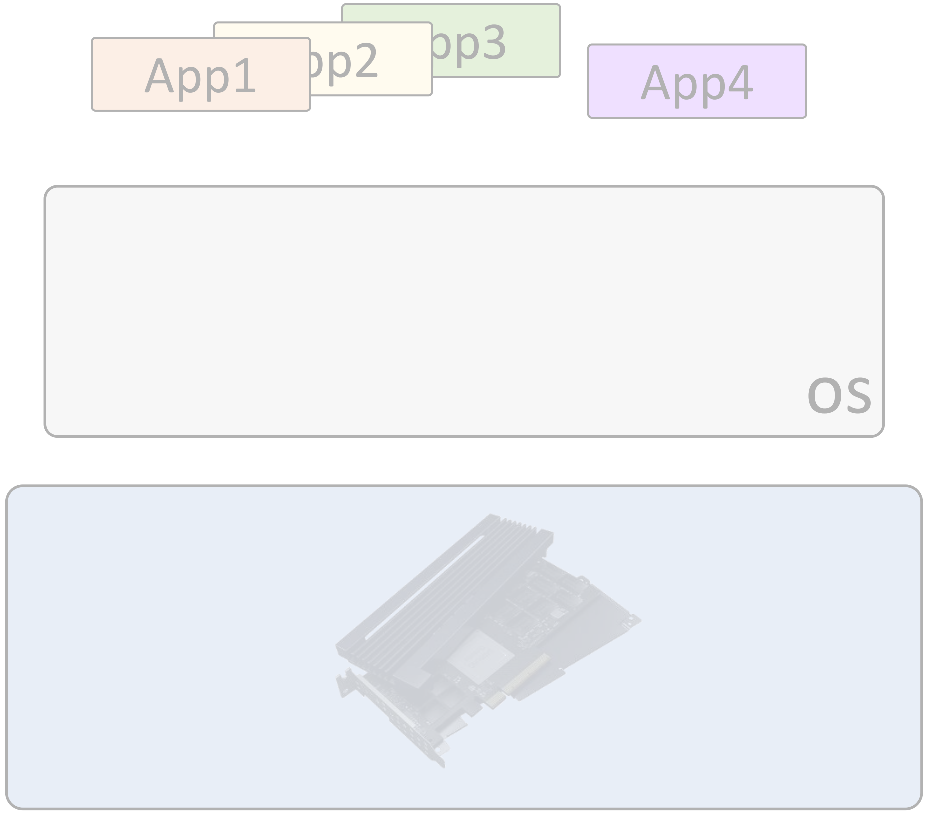
Virtual Time-based Fair Queueing – D2FQ

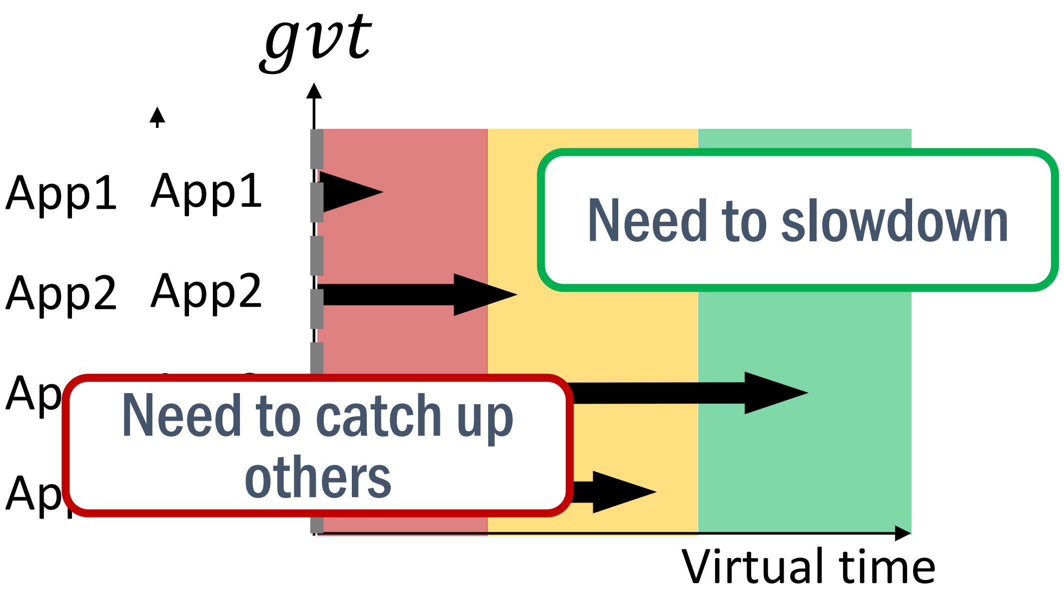




$$Virtual\ time = \frac{\sum I/O\ size_{completed}}{I/O\ weight}$$

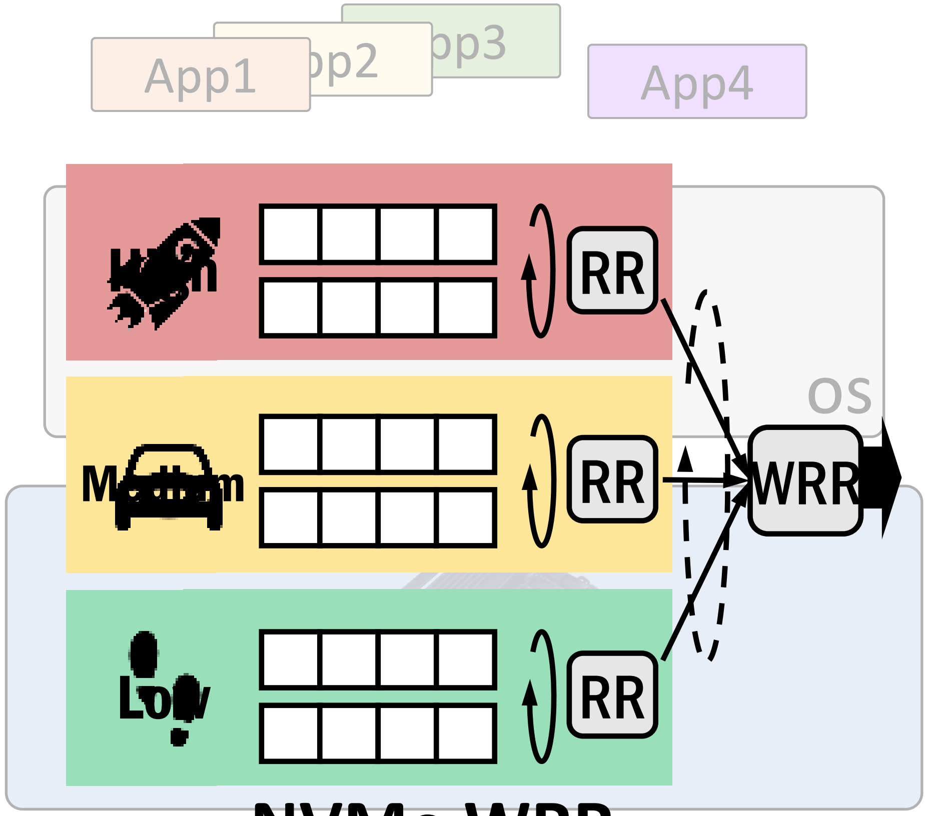
Virtual Time-based Fair Queueing – D2FQ



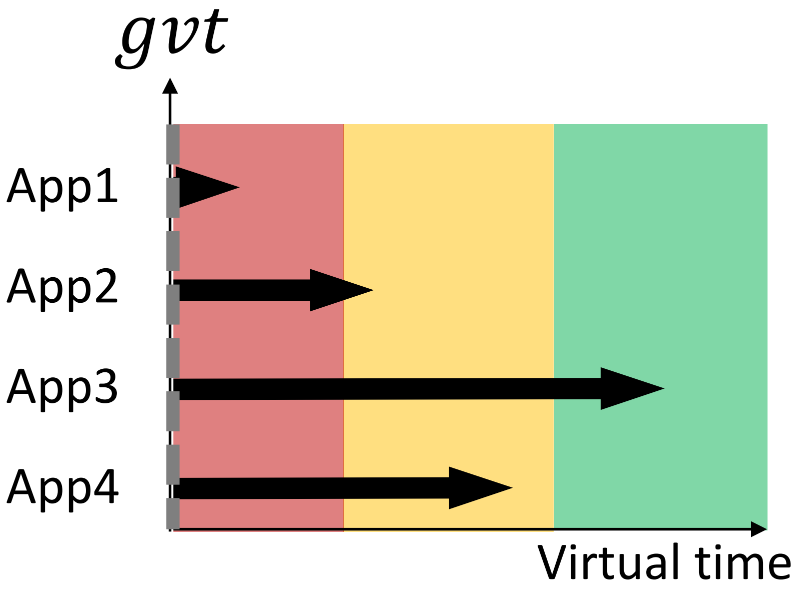


Throttle flows whose virtual time is far ahead of gvt

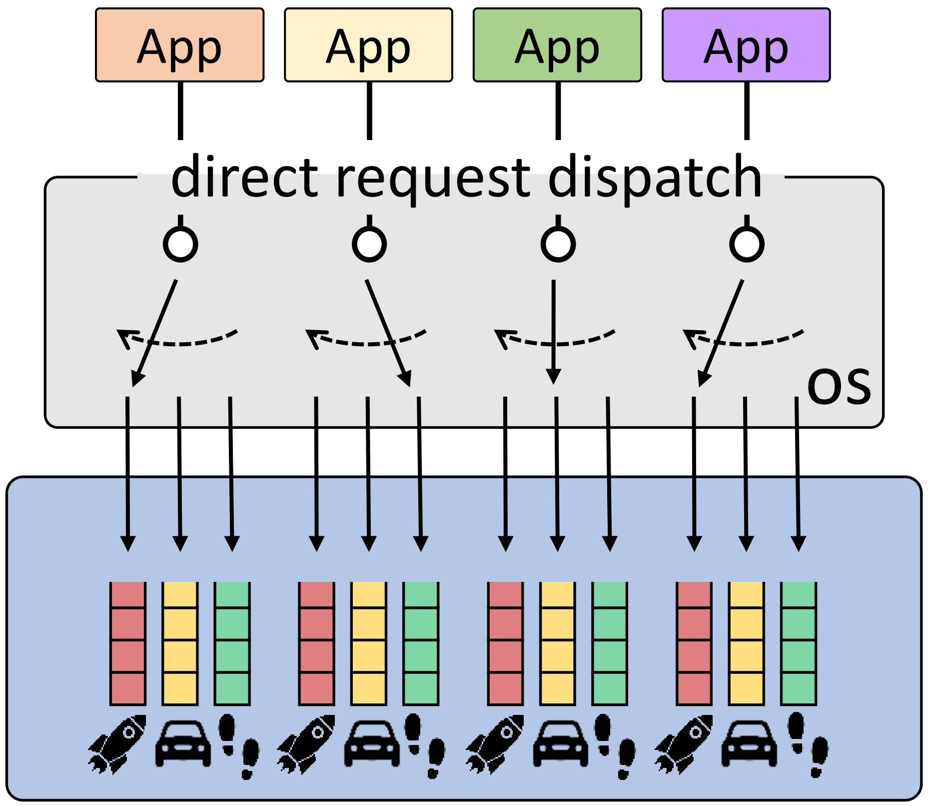
Virtual Time-based Fair Queueing – D2FQ

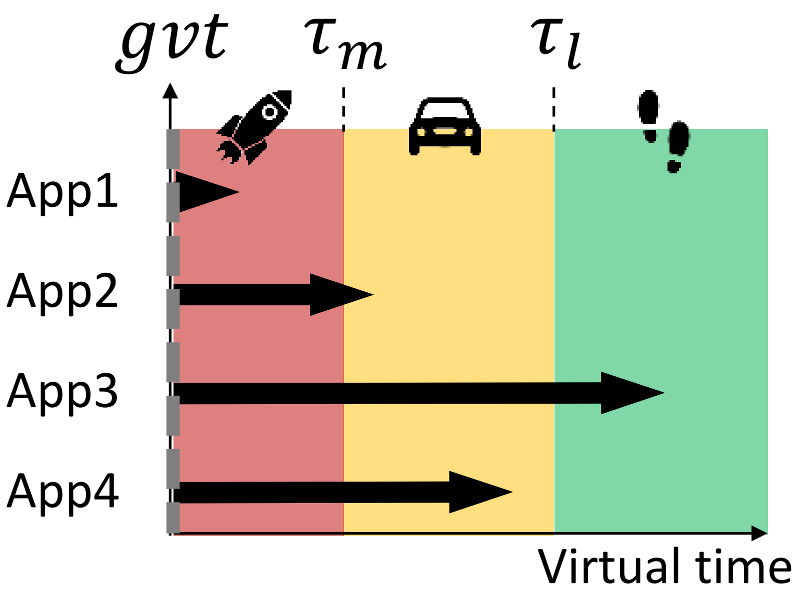


NVMe WRR



Virtual Time-based Fair Queueing – D2FQ





$$\begin{cases} \text{(rocket)} & if, vt_{flow} - gvt < \tau_m \\ \text{(car)} & else\ if, vt_{flow} - gvt < \tau_l \\ \text{(footprints)} & Otherwise \end{cases}$$

D2FQ Challenges

How to obtain sufficient I/O processing speed difference

Dynamic HL ratio adjustment

Which flow should be selected for I/O throttling?

Setting the queue class thresholds (τ_m, τ_l)

Please see the paper

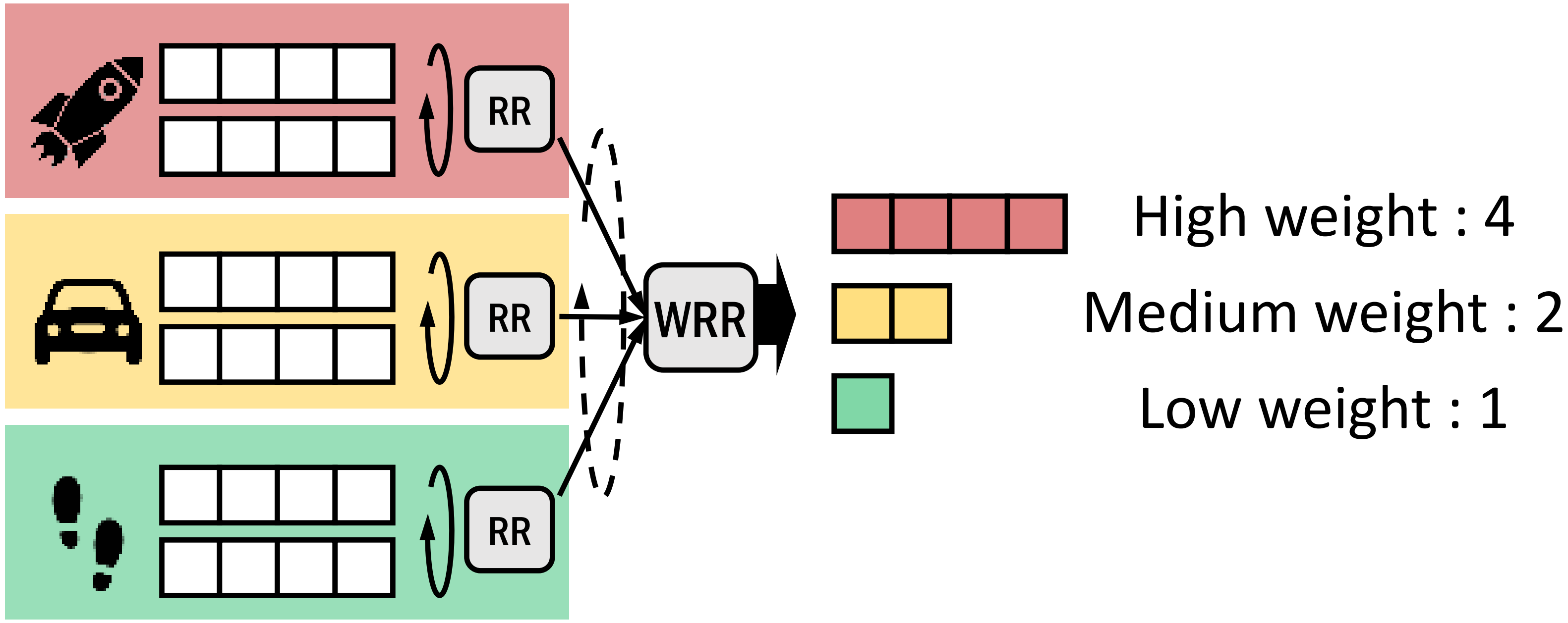
How to manage gvt scalably?

Sloppy minimum tracking

Dynamic HL Ratio Adjustment

- **HL ratio**
  - Ideal ratio of I/O processing speed between high and low queues
  - Ability to regulate virtual time process

e.g.)



Dynamic HL Ratio Adjustment

- **HL ratio**
  - Ideal ratio of I/O processing speed between high and low queues
  - Ability to regulate virtual time process

e.g.)

High weight : 4

Most important factor to achieve I/O fairness

Low weight : 1

Dynamic HL Ratio Adjustment

- **HL ratio**
  - Ideal ratio of I/O processing speed between high and low queues
  - Ability to regulate virtual time process

Low HL ratio

Small ability to regulate

may violate fairness

High HL ratio

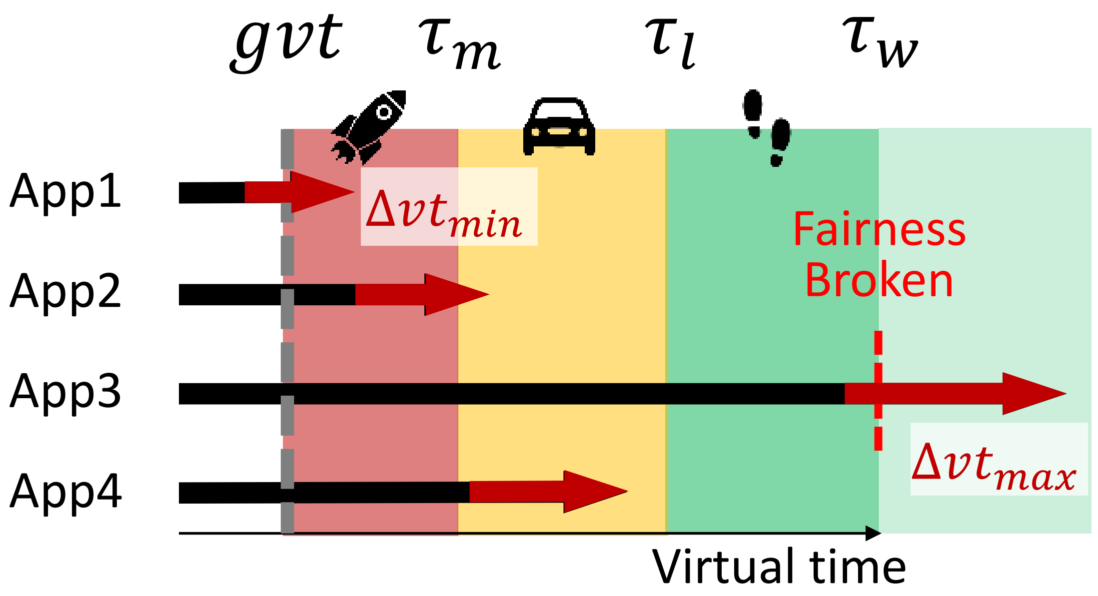
Takes too long to process

may incurs high tail latency

Need to set a proper HL ratio value dynamically

Dynamic HL Ratio Adjustment

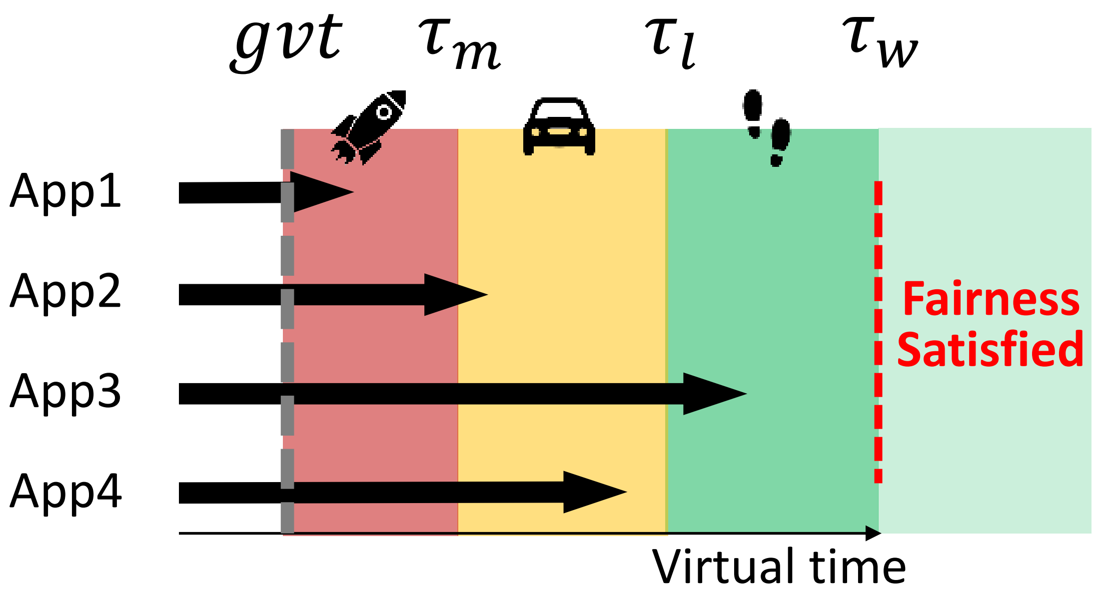
- **Increasing HL ratio**
  - Detect unfairness with τ_w
  - Calculate the additional I/O throttling capability to provide fairness
    - Calculate the delta of virtual time (Δvt) last time period
    - Current system requires at least $\frac{\Delta vt_{max}}{\Delta vt_{min}}$ times additional throttling capability



$$HL\ Ratio_{next} = \left\lfloor \frac{\Delta vt_{max}}{\Delta vt_{min}} \times HL\ Ratio_{prev} \right\rfloor + 1$$

Dynamic HL Ratio Adjustment

- **Decreasing HL ratio**
  - Occur when fairness is satisfied
    - Maximum virtual time gap is below τ_w
  - Calculate slowdown of each flow
    - Required throttling capability of system to satisfy fairness between a flow and the slowest flow
  - Set next HL ratio as the largest slowdown among all active flows



$$\text{slowdown}(f) = \frac{\textit{Estimated bandwidth of flow f when using high queues only}}{\textit{Actual bandwidth of flow f}}$$

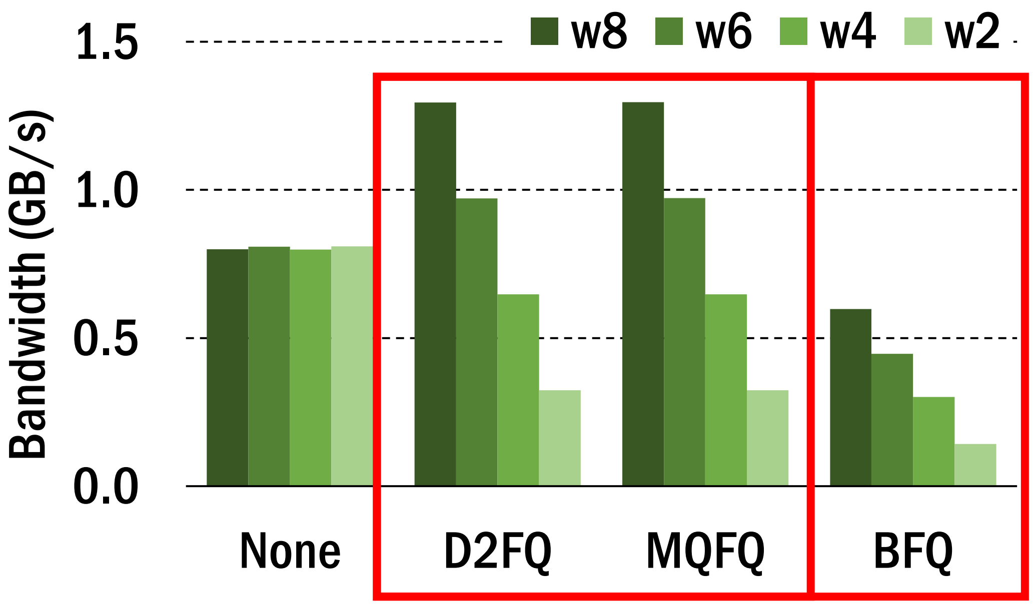
$$H/Lratio_{next} = \boldsymbol{MAXIMUM}(\,\text{slowdown}(f)\,)$$

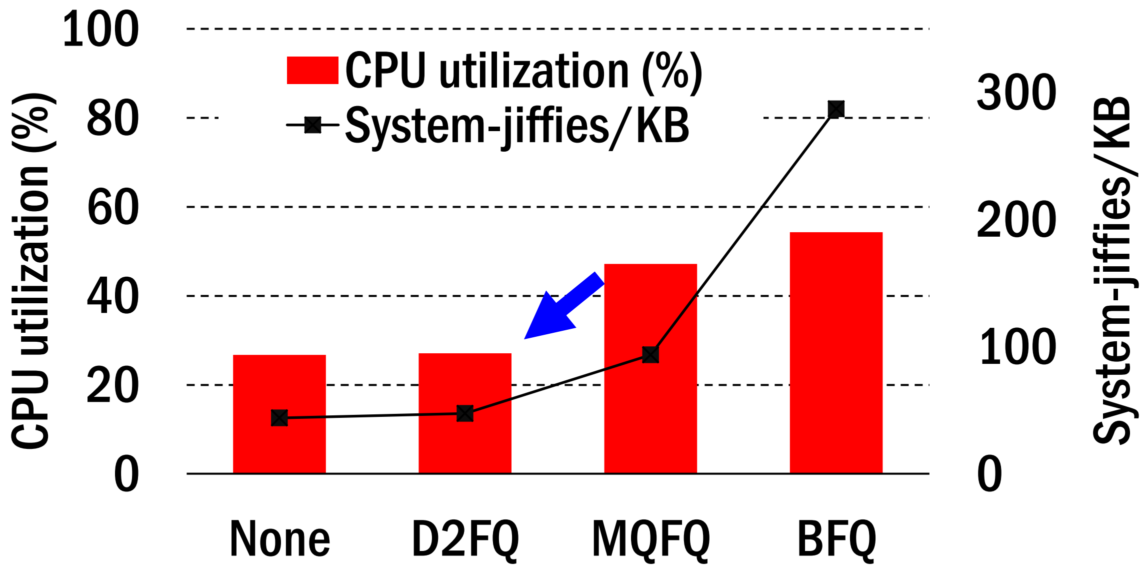
Evaluation

- **Experimental configuration**

CPU	Intel Xeon Gold 5112 3.6 GHz 8 physical cores (Hyperthreading off)
OS	Ubuntu 18.04.4
Base kernel	Linux 5.3.10
Memory	DDR4 192 GB
Storage device	Samsung SZ985 800 GB Z-SSD
Target fair I/O schedulers	None / **D2FQ** / MQFQ[ATC'19] / BFQ [Linux]
Workloads	Microbenchmark: FIO (libaio engine) Realistic workload: YCSB on RocksDB

Evaluation on Fairness



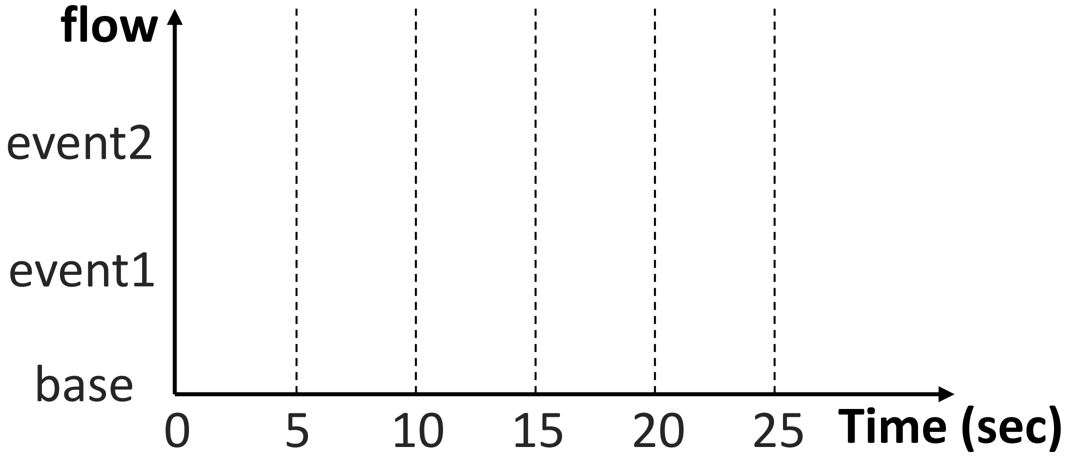


- MQFQ and D2FQ achieve fairness while fully utilizing device bandwidth
- D2FQ reduced CPU utilization by up to 45% compared to MQFQ

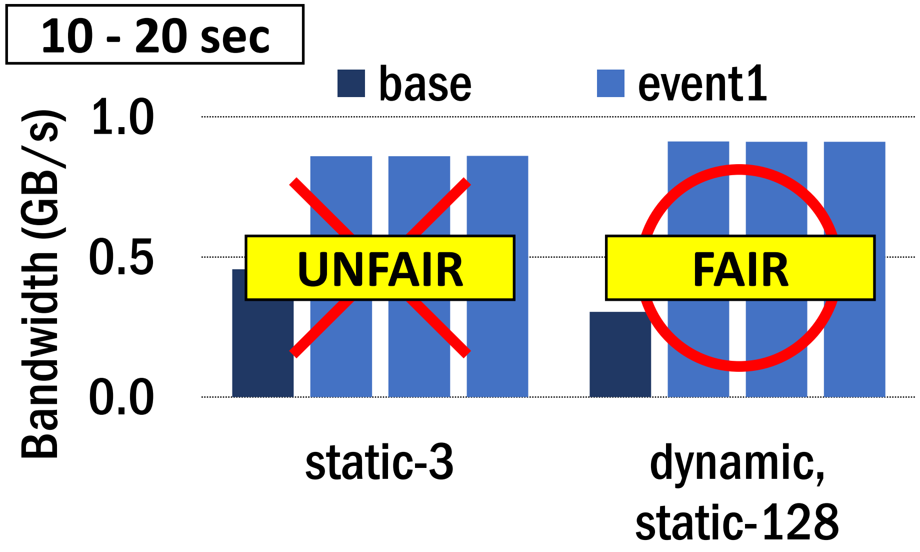
Dynamic HL Ratio Adjustment

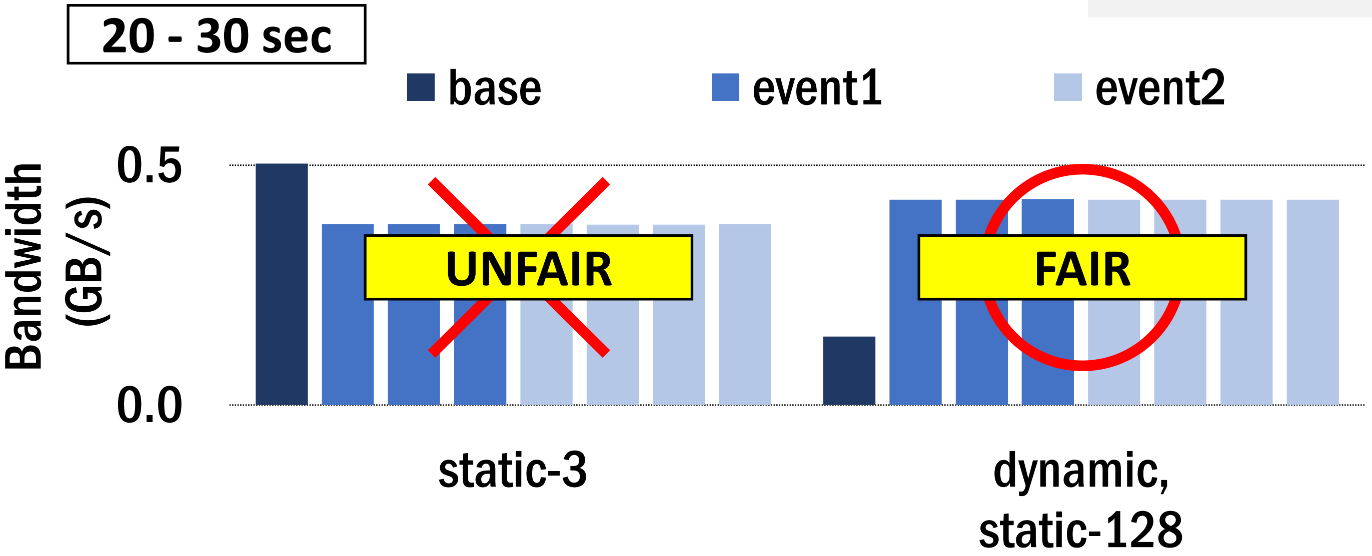
- **Compare I/O performance with three HL ratio setups**
  - Static-3, Static-128, dynamic (D2FQ-default)
  - # of flows increase with event1 and event2
  - Flows have different weights (1 vs 3)

	run time (sec)	I/O weight	# of flows
base	0 – end	1	1
event1	10 – end	3	3
event2	20 - end	3	4



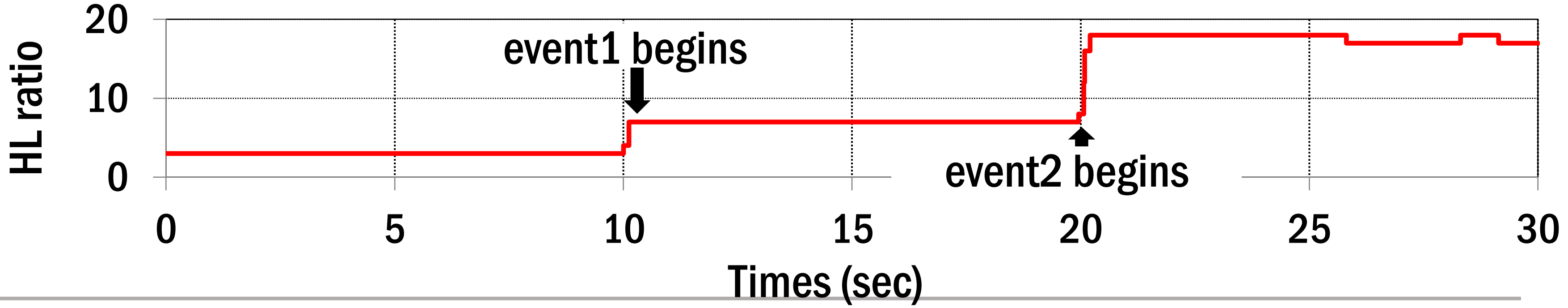
Dynamic HL Ratio Adjustment





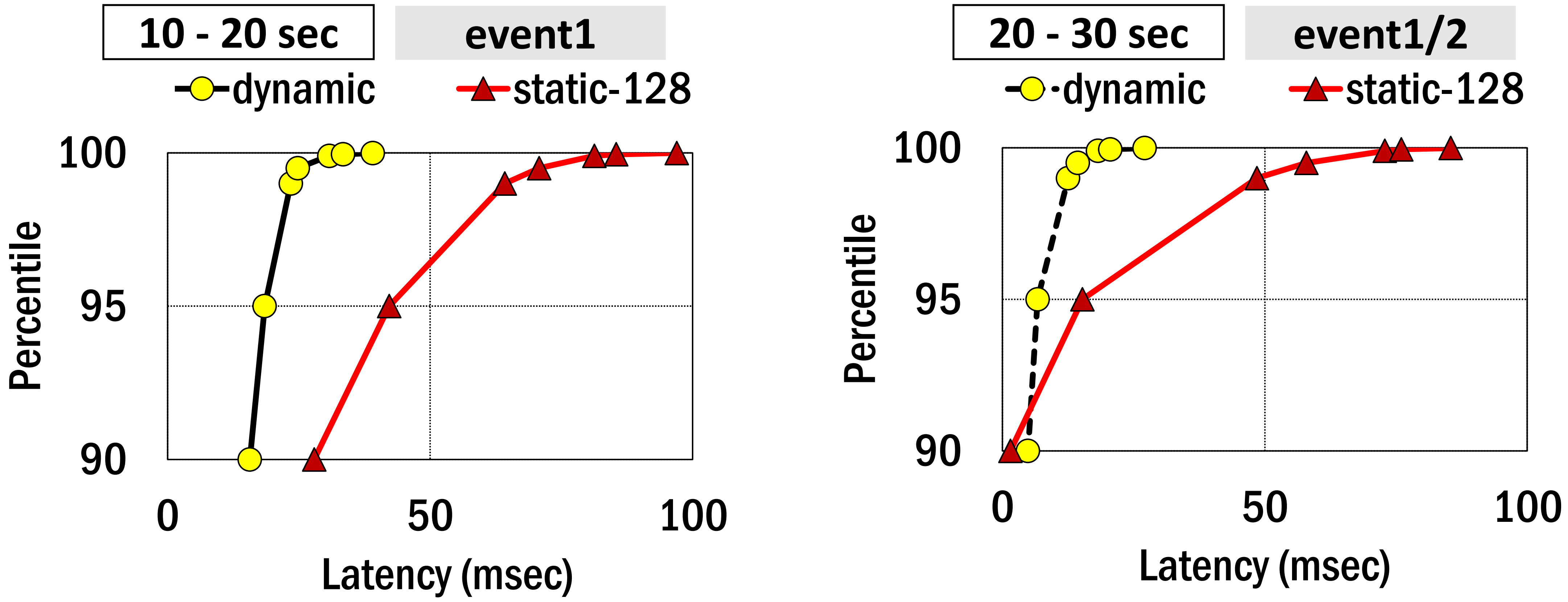
- Static-128 and our scheme (dynamic) achieve fairness
- Static-3 fails to achieve fairness because HL ratio of 3 is too small

- Runtime change of HL ratio in our scheme (dynamic)



Dynamic HL Ratio Adjustment

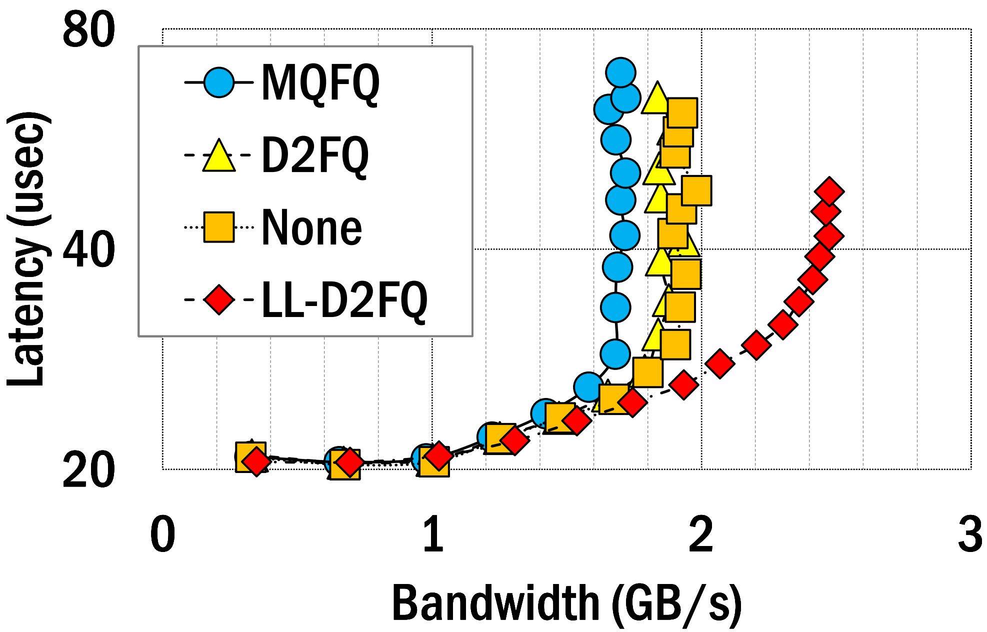
- Tail latency

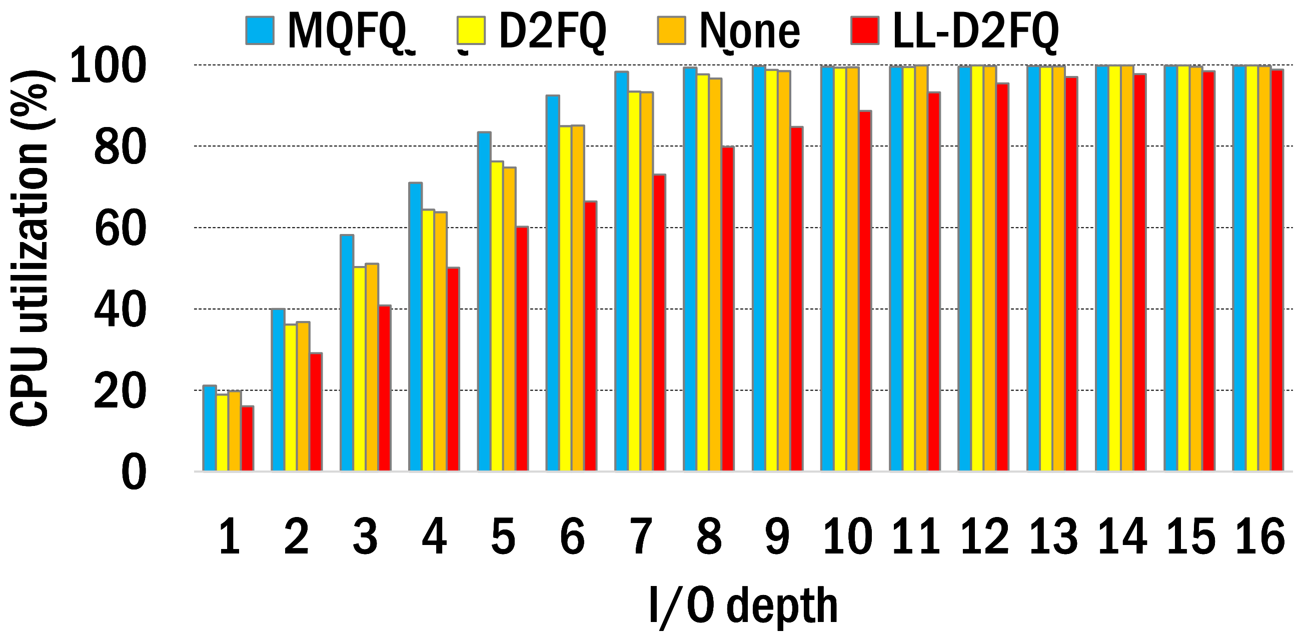


- Dynamic shows low tail latency as compared to static-128

I/O Performance

- Single thread high queue-depth I/O performance

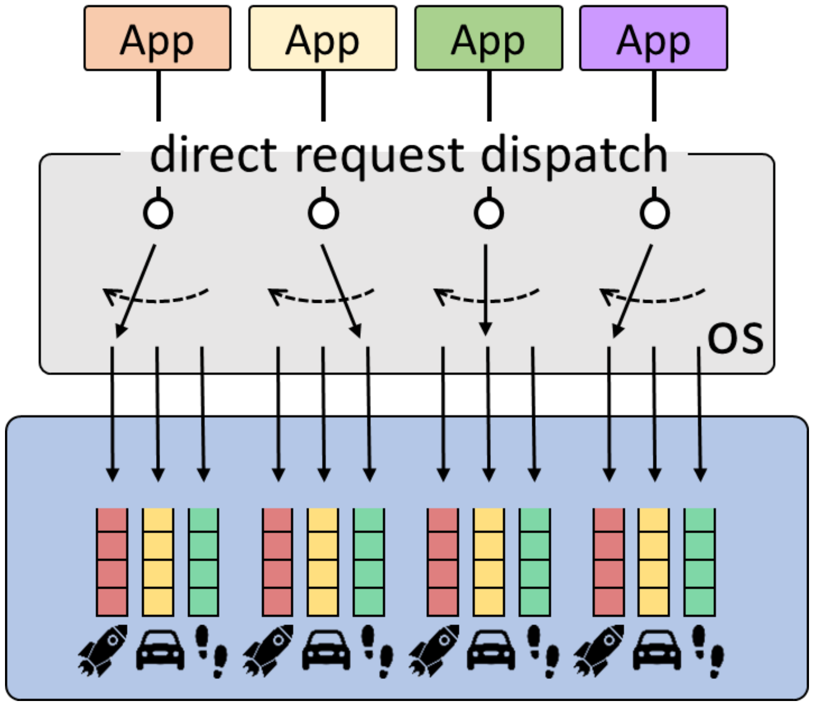




- D2FQ shows low CPU usage & high I/O performance (latency and bandwidth)
- D2FQ can be combined with AIOS [ATC'19], low-latency block-layer bypassing scheme
  - LL-D2FQ shows lowest CPU usage and highest I/O performance

D2FQ Conclusion

- **A low CPU overhead fair queueing I/O scheduler built on top of NVMe WRR**
- **Fair queueing with high scheduling performance**
  - Reducing CPU utilization by up to 45%
  - Fully utilizing bandwidth & showing low latency
  - Enhanced scalability
- **Vitalizing block-layer-bypass schemes (e.g., AIOS [ATC'19])**
  - Their low-latency I/O performance is now augmented with fair I/O scheduling



Source Code:
https://github.com/skkucsl/d2fq

Thank you

SUNG KYUN KWAN UNIVERSITY

Contact:

Jiwon Woo - jiwon.woo@csi.skku.edu
Minwoo Ahn - minwoo.ahn@csi.skku.edu
Gyusun Lee - gyusun.lee@csi.skku.edu
Jinkyu Jeong - jinkyu@skku.edu

Source Code:

https://github.com/skkucsl/d2fq