STORAGE DEVELOPER CONFERENCE

SDC 21

BY Developers FOR Developers

Virtual Conference
September 28-29, 2021

A SNIA Event

BFQ Linux IO Scheduler Optimizations for Multi-Actuator SATA Hard Drives

Presented by

Paolo Valente, Linaro

Tim Walker, Seagate Technologies

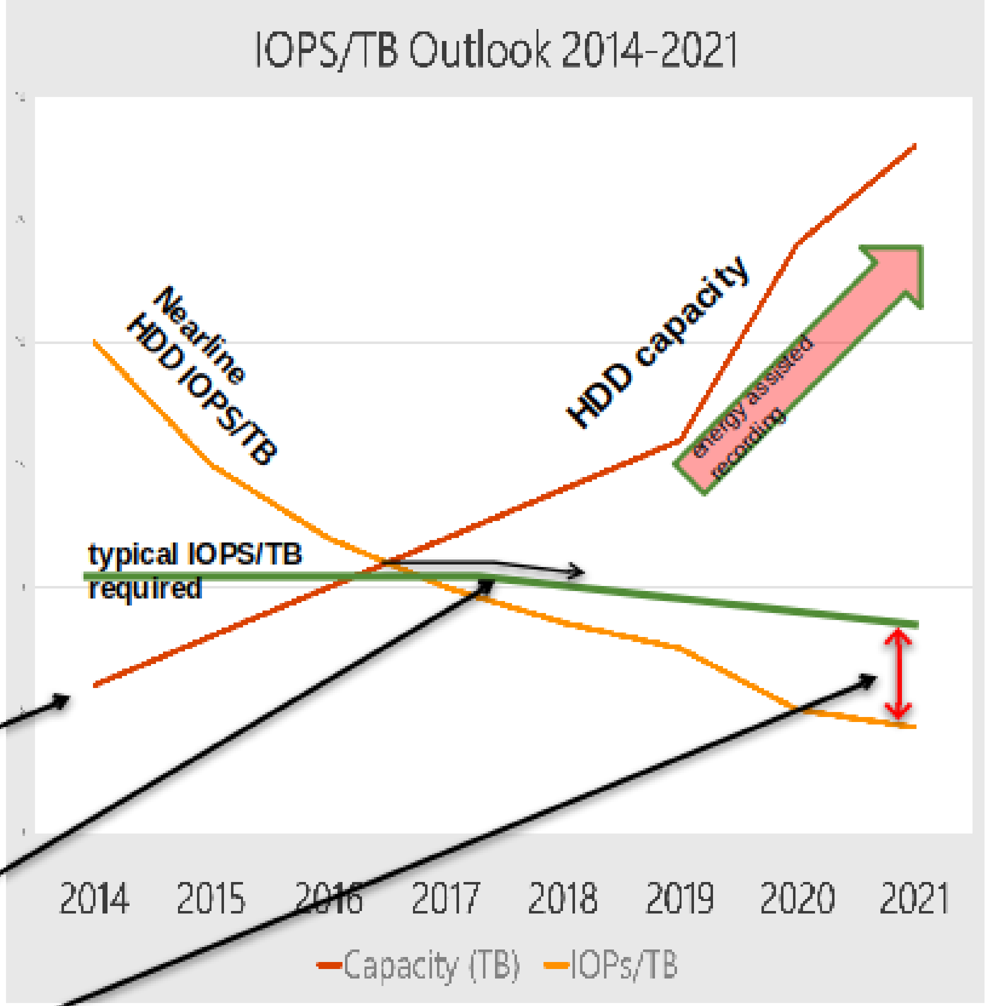
Content

- Multi-actuator drives: benefits and issues
- Using the BFQ I/O scheduler to control actuator load
- Performance results

HDDs for Cloud Duty: Stranded Unusable Capacity

IOPS / TiB erosion limits usable capacity

- New recording technology is driving HDD capacity to 60TB+ per spindle
- Servo-mechanical capability does not scale with areal density → reducing IOPS/TB
- Latency driven workloads cannot utilize the capacity gains as IOPs/TB drops below minimum workload QOS
- To meet read latency QOS, customers can:
  - Replace HDD with flash
  - Short-stroke the HDD, leaving unused capacity stranded
  - Deploy lower capacity drives
    - ***All detrimental to storage TCO***



HDD capacity increasing

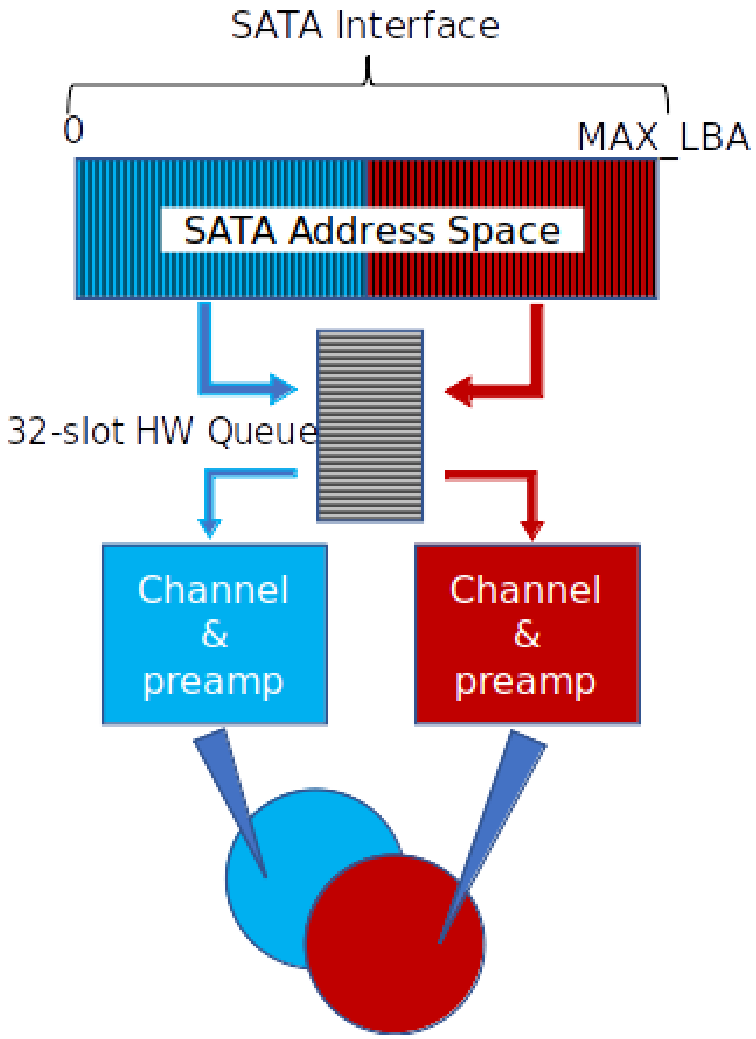
Data management advances trending minimum IOPS/TB requirements down

HDD IOPS/TB Performance Gap

Click to add Text

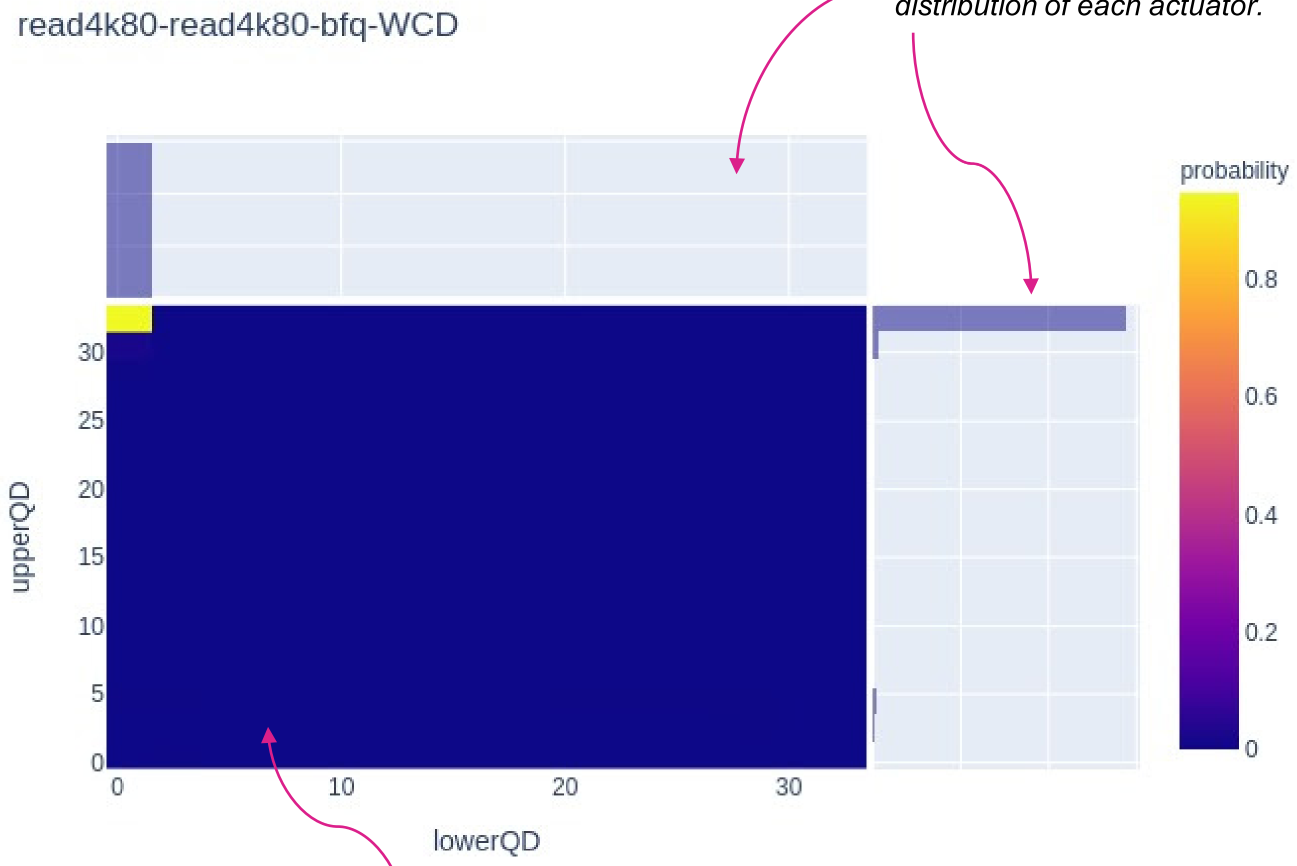
Single-namespace Split Actuator HDD

- The split-actuator approach divides the disks into groups, each addressed by an independent actuator. A given sector is reachable by only one actuator - none of the address space is shared.
- For example, Seagate's SATA version contains two actuators and maps the lower half of the SATA LBA space to the lower actuator and the upper half to the upper. There are no changes to the IO protocol, except for a log page to report the LBA:Actuator mapping.
- SATA allows 32 commands to be queued in the NCQ hardware queue. The 32-slot HW queue is a shared resource that services both actuators.
- Like any shared resource, we can either manage it, or it will manage us...



Note: The single-namespace, split-actuator design is also compatible with single LUN SAS.

In fact, if fed without per-actuator control ...



Histograms showing the queue depth distribution of each actuator.

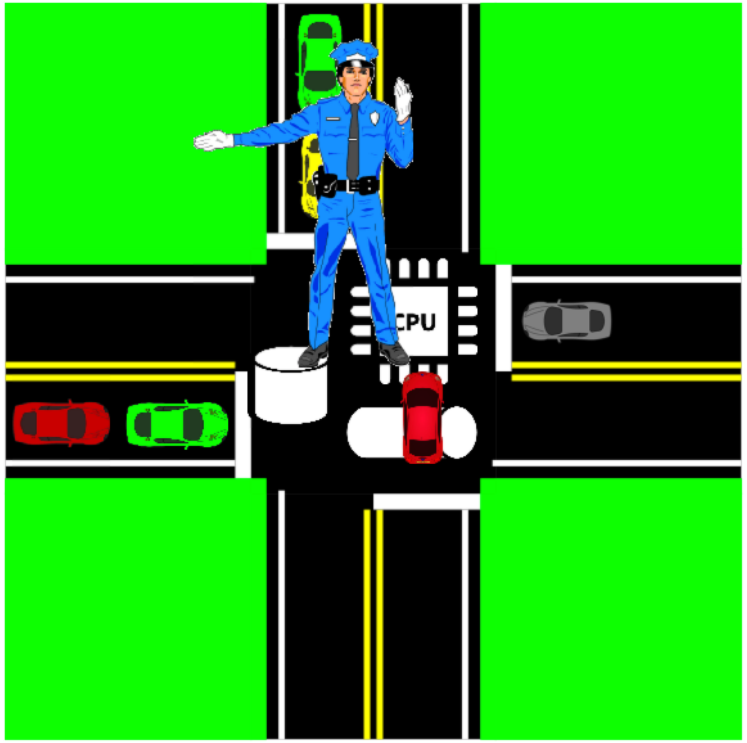
Heat map showing the relative distribution of simultaneous queue depths across the two actuators

- Some actuator may remain underutilized or even idle
- This may limit maximum throughput drastically

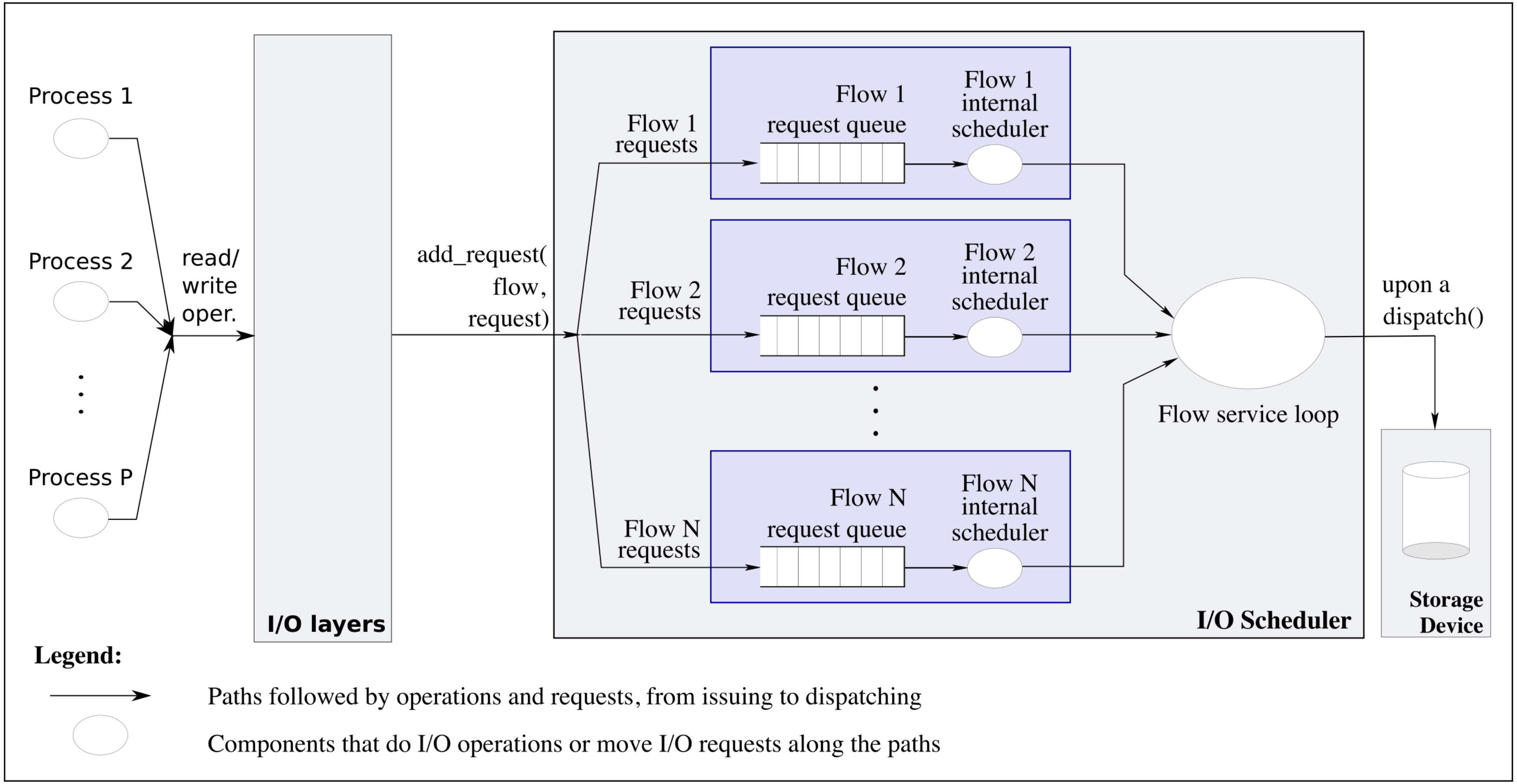
- Need to control actuator load
- The I/O stack already contains a component for controlling I/O
  - I/O scheduler

I/O schedulers

- Decide the order in which I/O requests are to be served
  - Order in which competing processes access storage
- So as to guarantee:
  - High I/O throughput
  - Low latency
  - High responsiveness - Low lag
  - Fairness
  - Other goals …
- Important for our problem:
  - The scheduler should provide a good ground for implementing flexible and accurate control on per-actuator load
- Available schedulers: *none, mq-deadline, kyber, BFQ*
- The BFQ I/O scheduler has a rich control-plane infrastructure

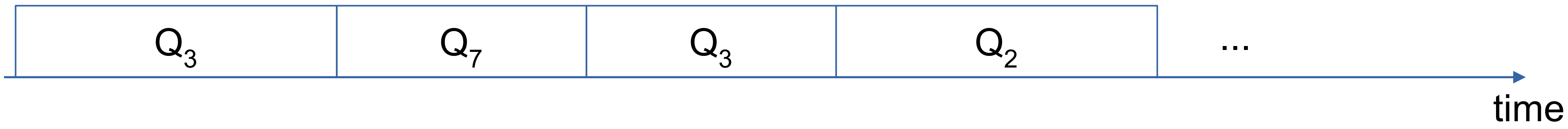


BFQ infrastructure



Queues and service loop

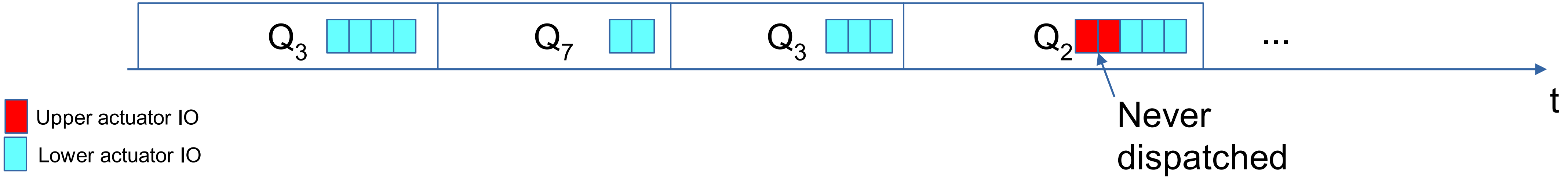
- BFQ usually considers the I/O of each process as a separate I/O flow
  - So it enqueues it in a separate request queue
  - With a weight associated with the process/queue
- Each queue is served for a while



- The service order is such that each queue is served, on average, at a rate proportional to the queue’s weight (the bandwidth share of each queue is proportional to the queue’s weight)

Loss of control on actuator load

- A process may generate requests for any actuator
- Scheduling logic does not take actuators into account
  - It does not guarantee any load balance among actuators
- The same process may generate requests even for both actuators
  - The internal scheduling logic inside each queue provides no guarantee that requests for a given actuators are served
- For example, only one actuator may be served in the previous schedule



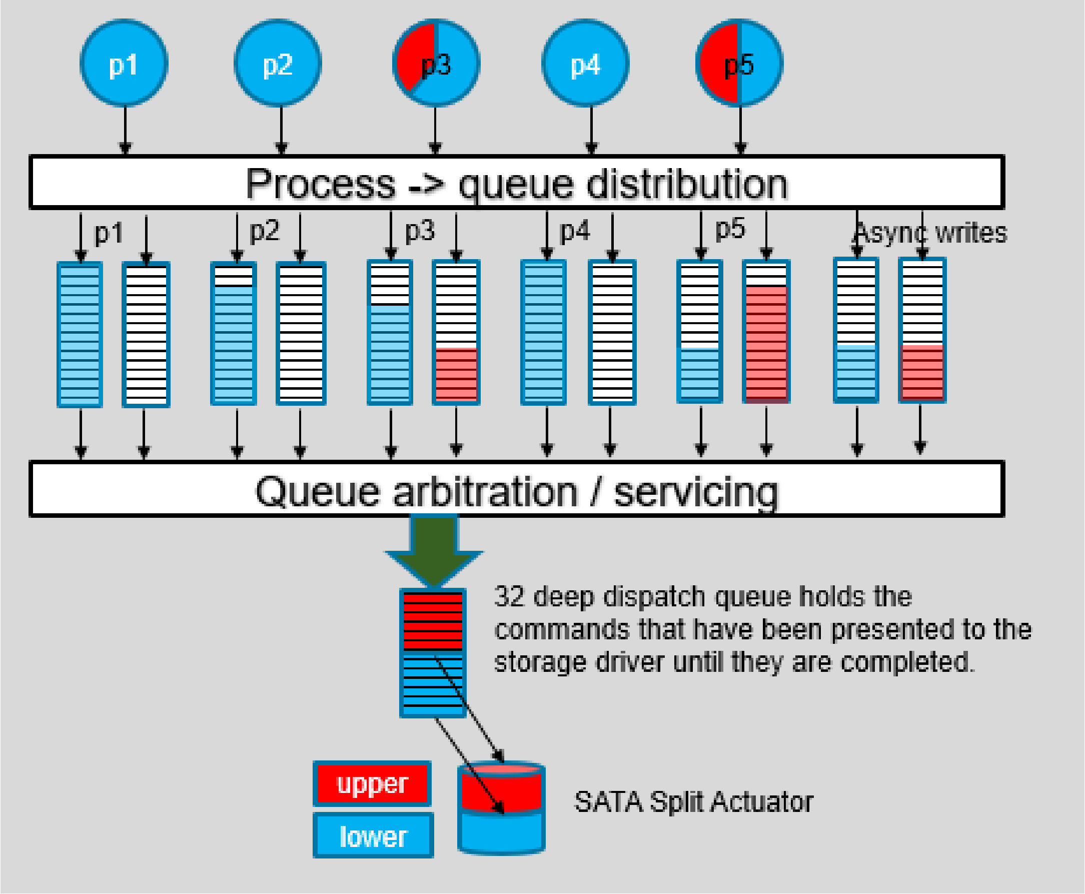
Never dispatched

🟥 Upper actuator IO

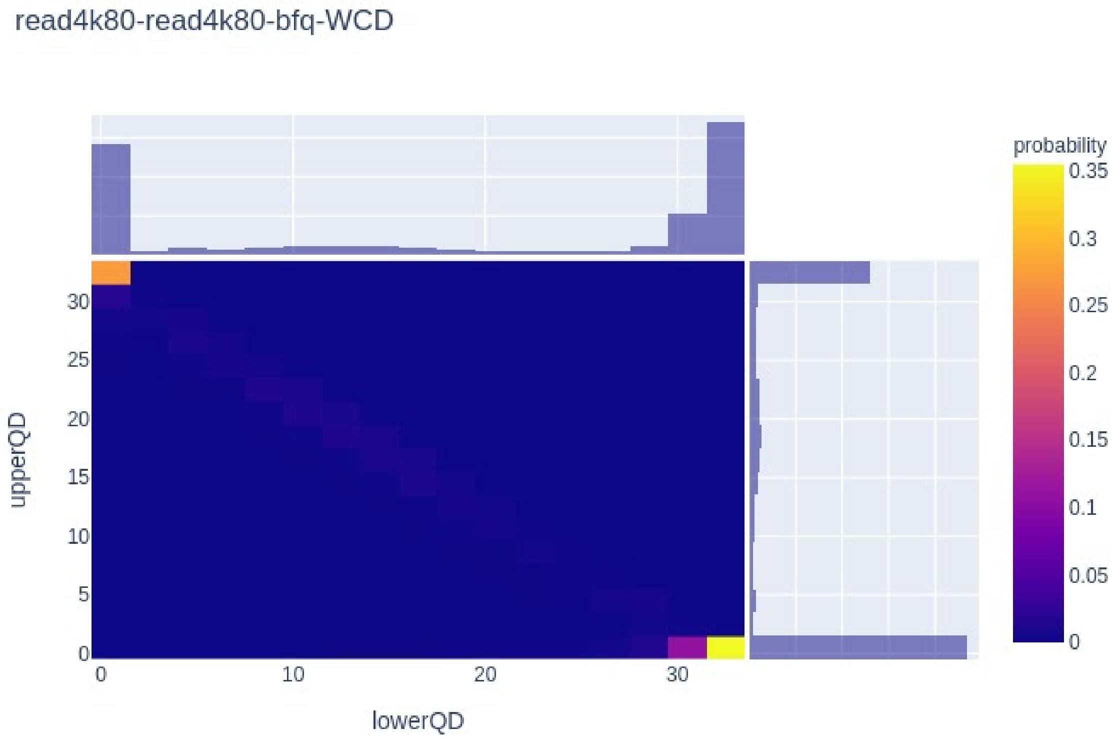
🟦 Lower actuator IO

Base idea: split queues

- Split each queue into one queue for each actuator
- So, each process will be associated with *N* separate queues, one for each actuator
- BFQ’s service loop will guarantee that all queues, and therefore all actuators, are served


p1
p2
p3
p4
p5
Process -> queue distribution
p1
p2
p3
p4
p5
Async writes
Queue arbitration / servicing
32 deep dispatch queue holds the commands that have been presented to the storage driver until they are completed.
upper
lower
SATA Split Actuator

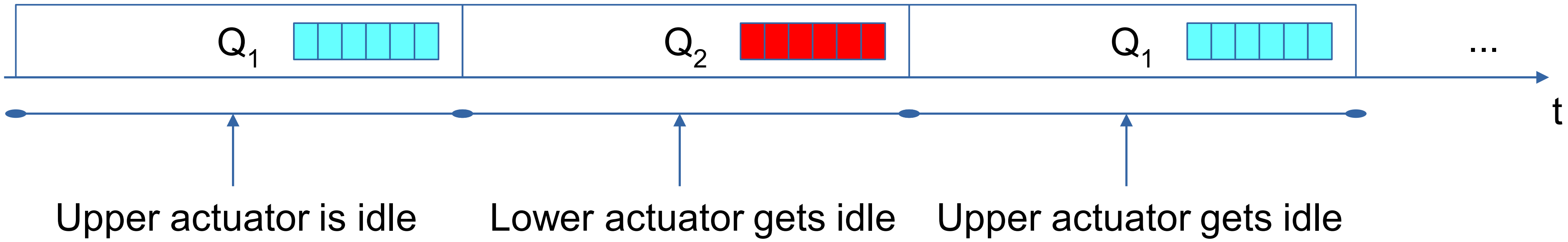
Result



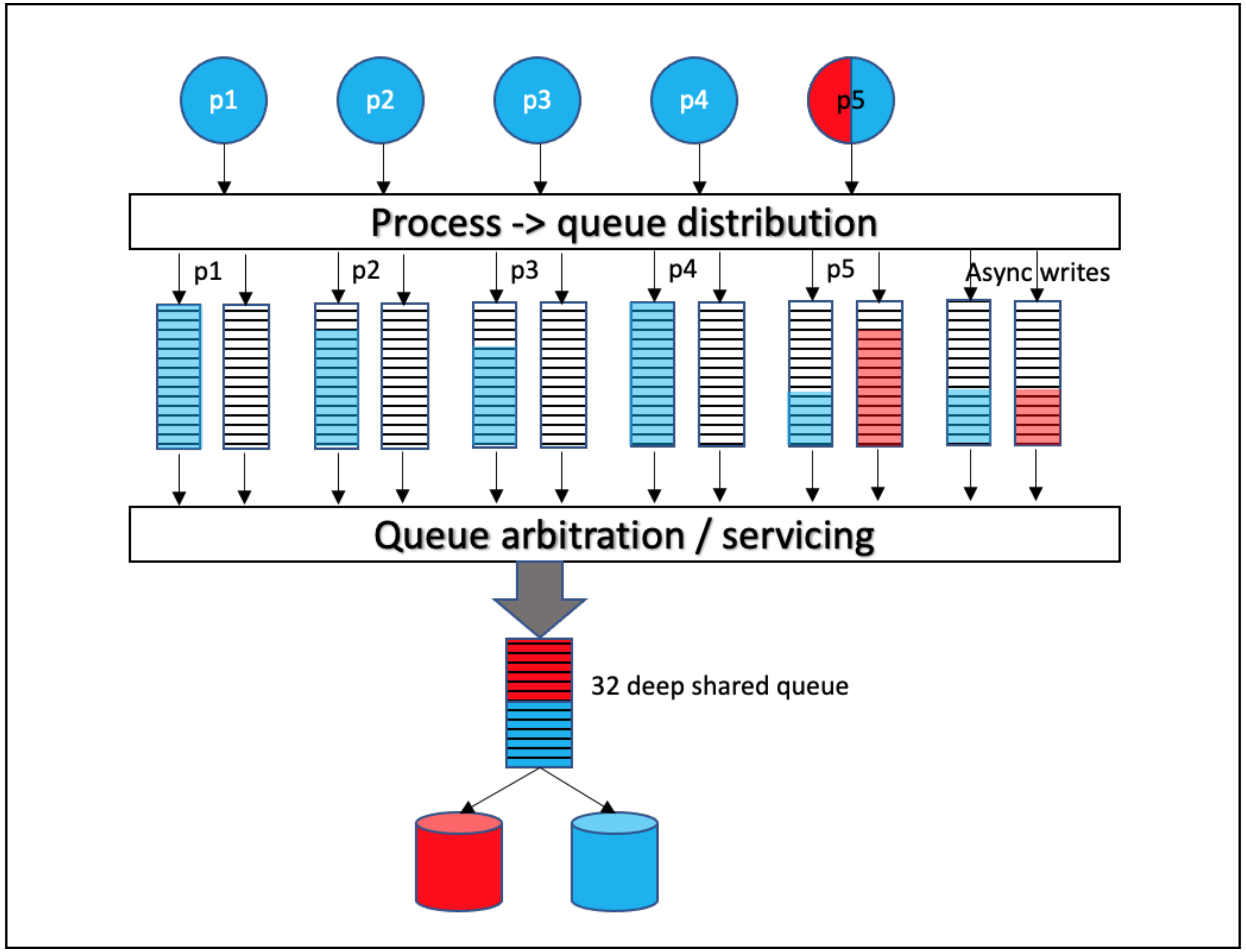
- Now two opposite states are equiprobable
- Still, either only the upper or only the lower actuator is busy
- Why?

One-queue-at-a-time service

- This workload is made of two sequential read streams
- With this workload, only one queue at a time happens to be served, for long
- This service scheme maximizes throughput with a single actuator
- But with a dual actuator:



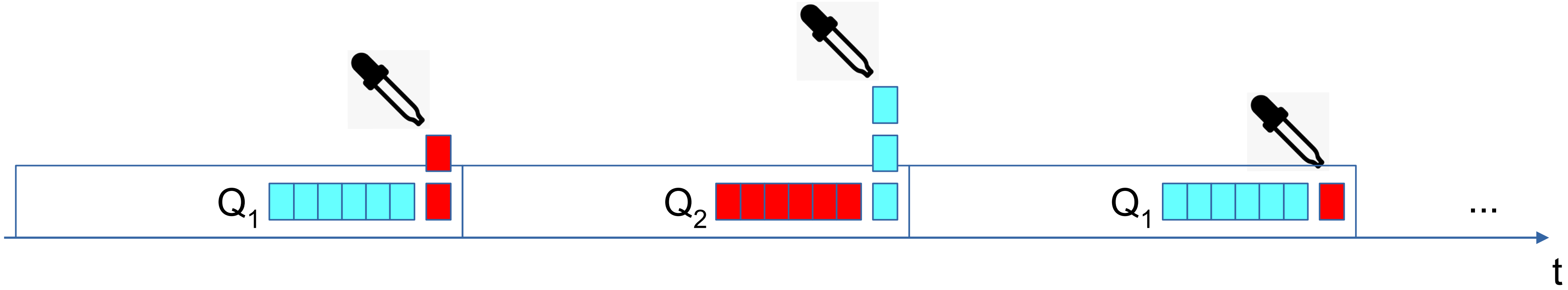
General problem



- There may be cases where some actuator is even more severely underutilized:
  - Many queues contain I/O for a given actuator, while few queues contain I/O for other actuators
  - One queue, containing I/O for a given actuator, has a much higher weight than queues that contain I/O for other actuators

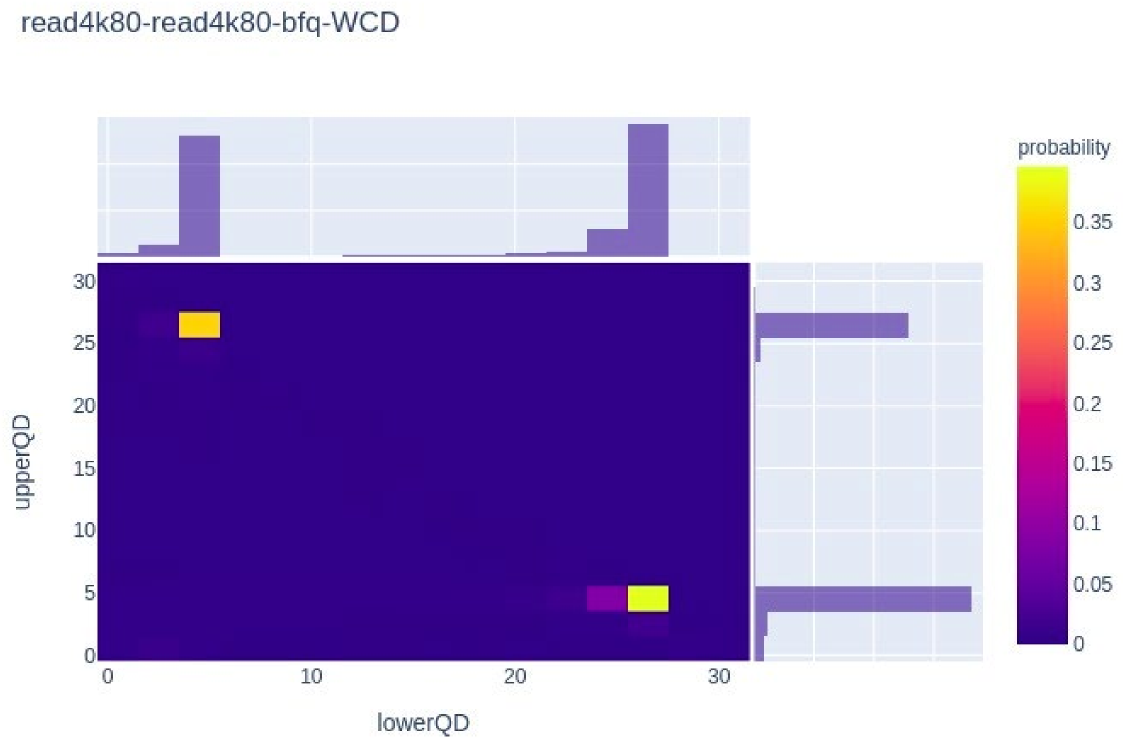
Improvement: inject I/O for the other actuators

- While serving a queue that contains I/O for a given actuator, inject (dispatch) some I/O for other actuators, if the the latter are underutilized



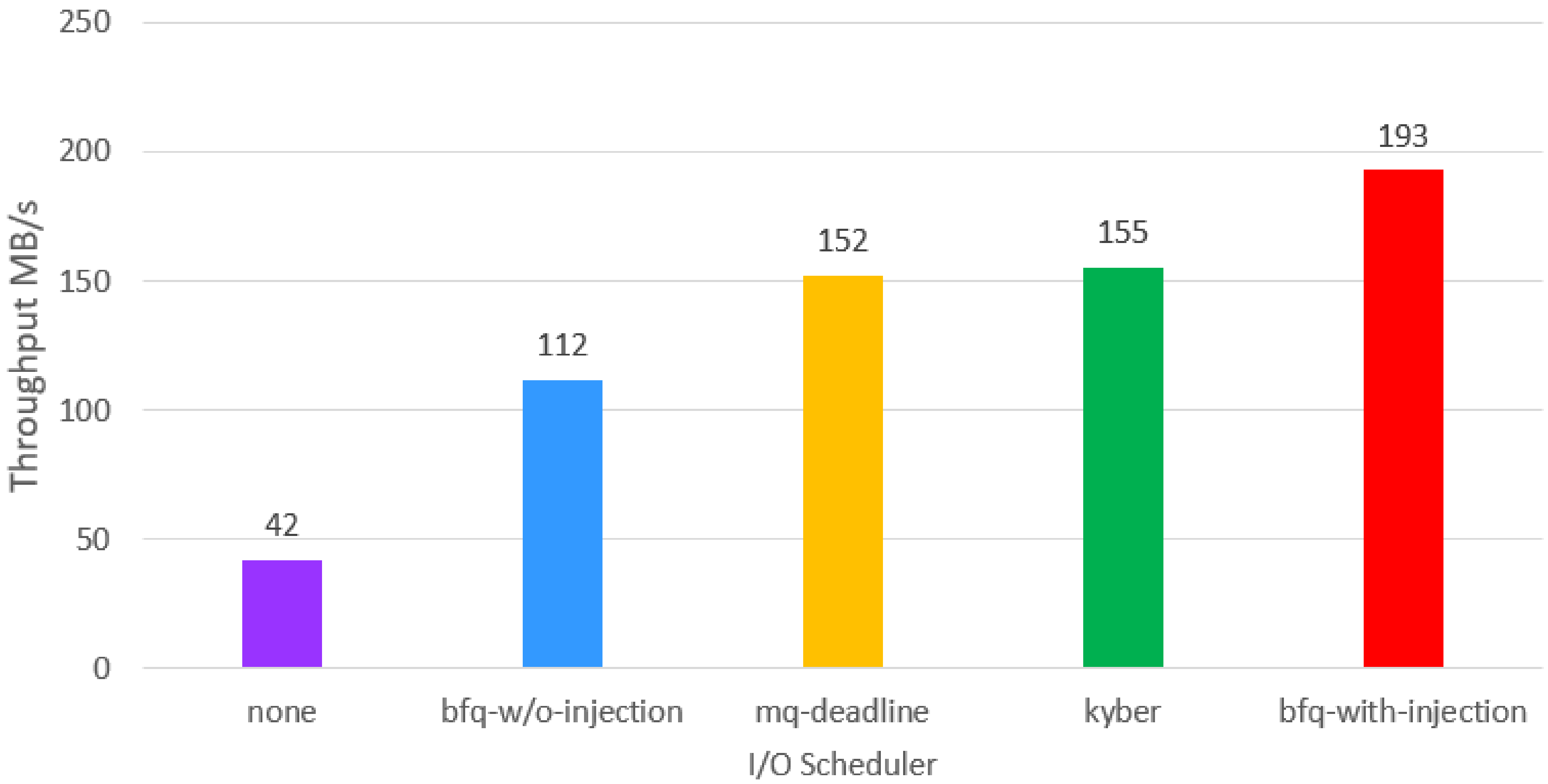
- Load threshold to decide whether or not to inject
  - Inject if below threshold
- Current threshold: 4

Results: minimum per-actuator load now guaranteed



- While one actuator is highly utilized, the load of the other is still around the threshold
- And viceversa

Results: throughput



- 25% higher throughput than the best-performing scheduler
- Stable performance, thanks to control (differently from the other schedulers)

Discussion

- Still a preliminary contribution
- Results shown in this presentation cover just one workload
- No production code available yet

Open issues:

- Injection does not take bandwidth distribution (weights) into account
- What is the best value for the injection threshold?
  - Most certainly, it depends on the workload
  - So it may be dynamic
  - Yet a good, static value could provide acceptable performance

Thank you
Please take a moment to rate this session.

Your feedback is important to us.