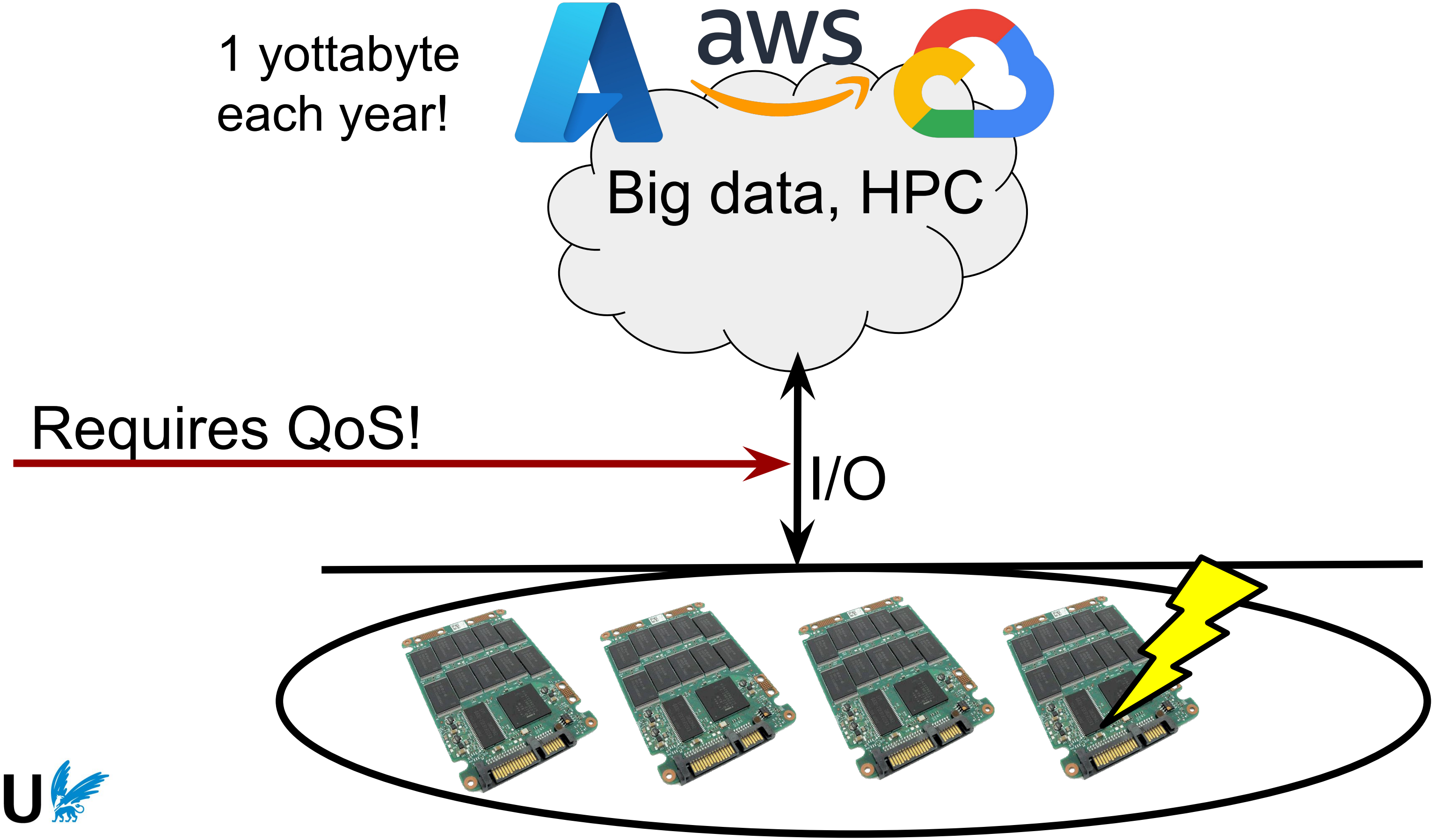
@Large Research
Massivizing Computer Systems

VRIJE UNIVERSITEIT AMSTERDAM

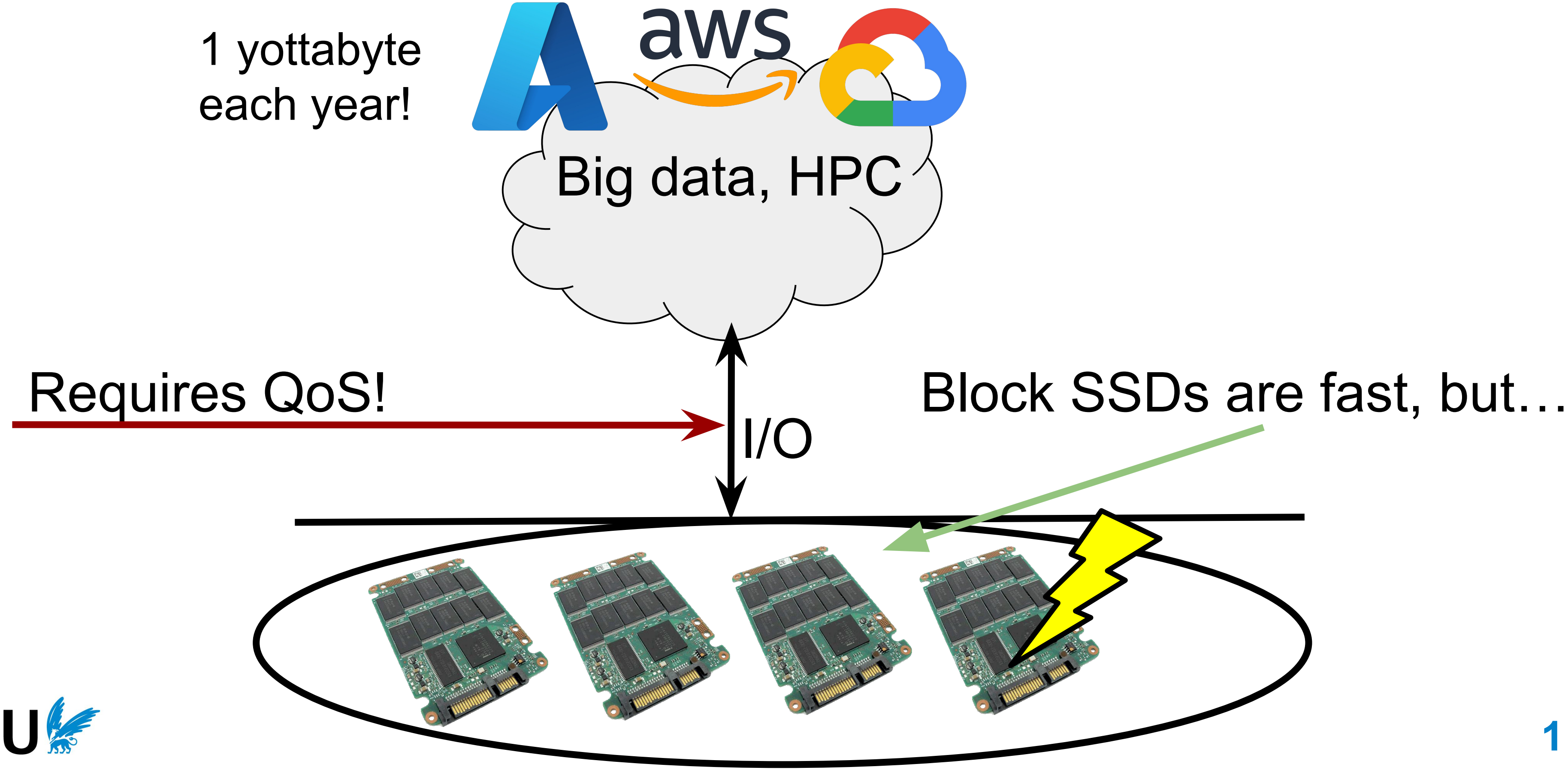
Exploring I/O Management Performance in ZNS with ConfZNS++

Krijn Doekemeijer, Dennis Maisenbacher, Zebin Ren, Nick Tehrany, Matias Bjørling, and Animesh Trivedi

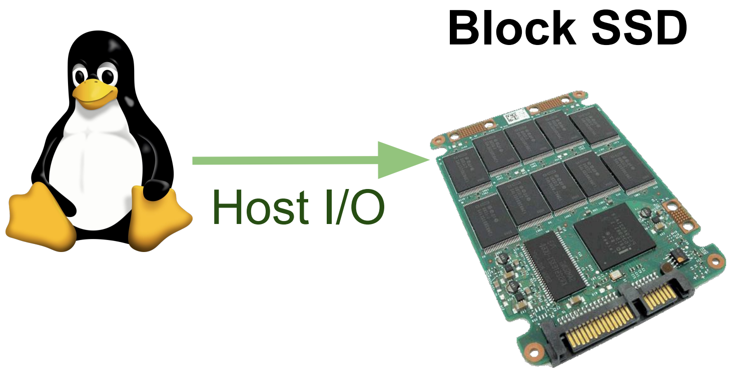
Storage QoS demands are increasing

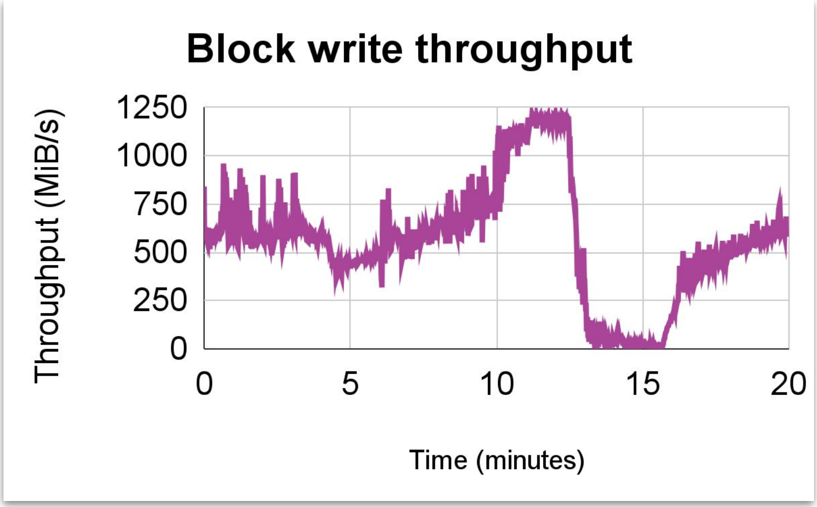


Storage QoS demands are increasing



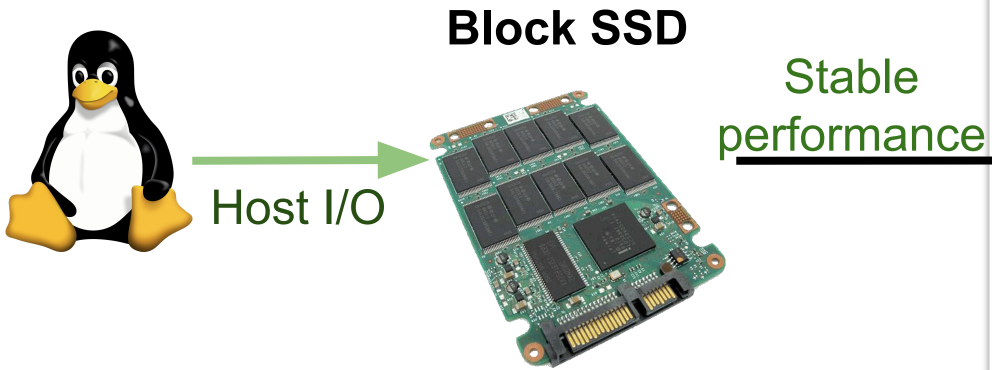
Block SSDs do not deliver QoS

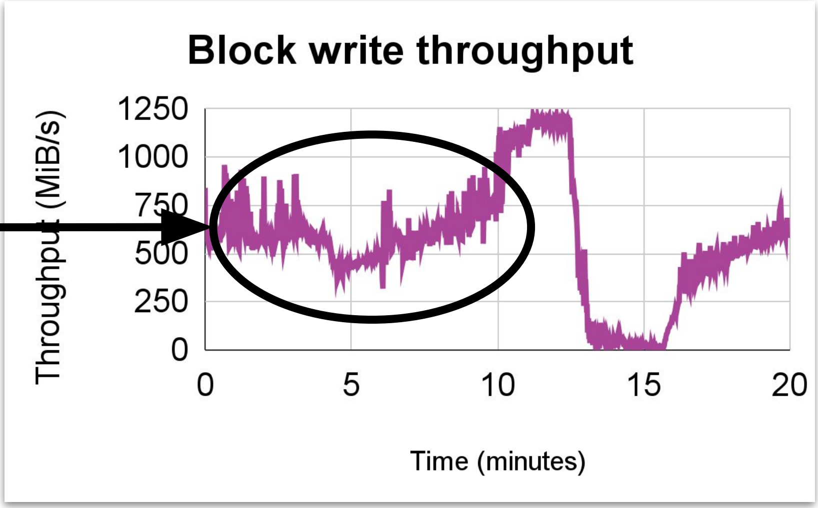




[1] IEEE CLUSTER’23, Krijn Doekemeijer; Nick Tehrany; Balakrishnan Chandrasekaran; Matias Bjørling; Animesh Trivedi, Performance characterization of NVMe Flash Devices with Zoned Namespaces (ZNS)

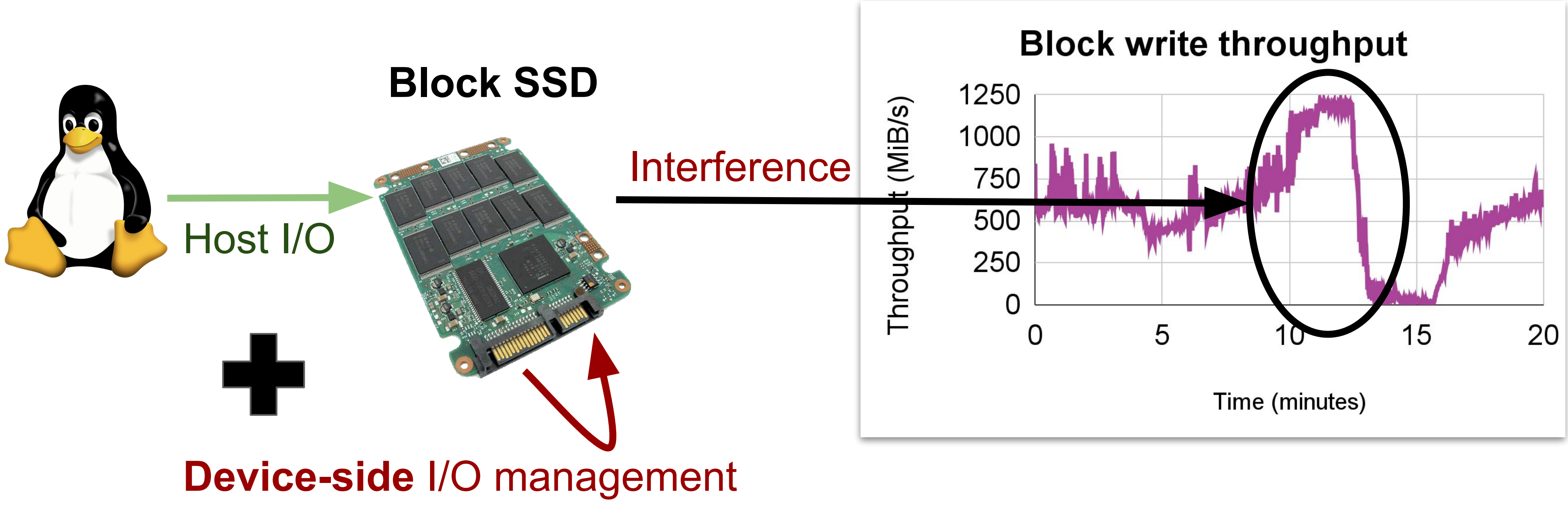
Block SSDs do not deliver QoS





[1] IEEE CLUSTER'23, Krijn Doekemeijer; Nick Tehrany; Balakrishnan Chandrasekaran; Matias Bjørling; Animesh Trivedi, Performance characterization of NVMe Flash Devices with Zoned Namespaces (ZNS)

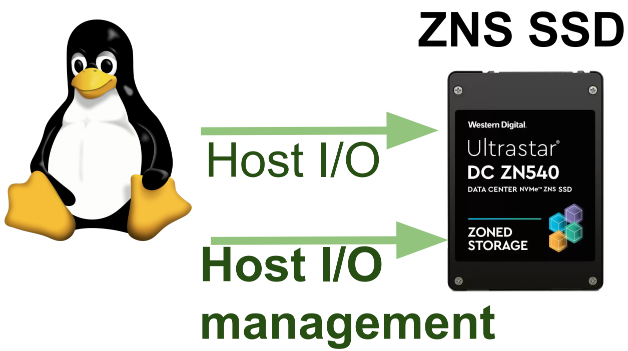
Block SSDs do not deliver QoS

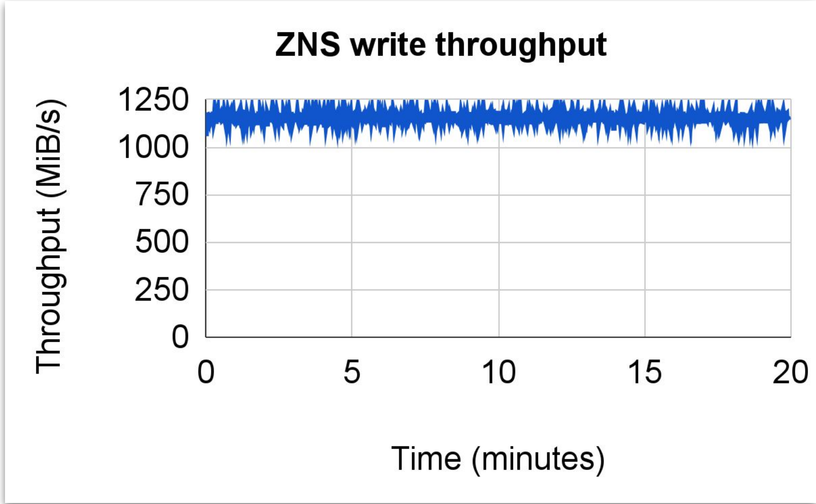


[1] IEEE CLUSTER’23, Krijn Doekemeijer; Nick Tehrany; Balakrishnan Chandrasekaran; Matias Bjørling; Animesh Trivedi, Performance characterization of NVMe Flash Devices with Zoned Namespaces (ZNS)

Solution: data-placement SSDs

- Expose I/O management to host
- Example: ZNS, FDP
- Achievable stable I/O performance

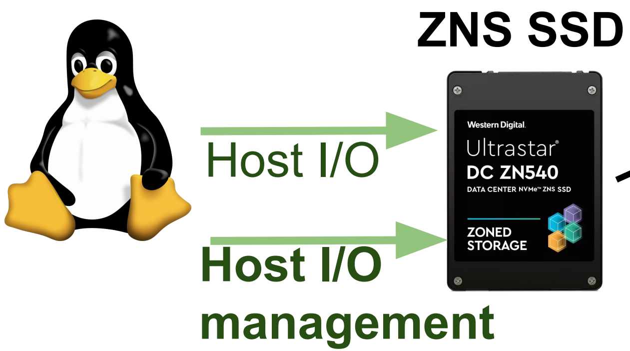


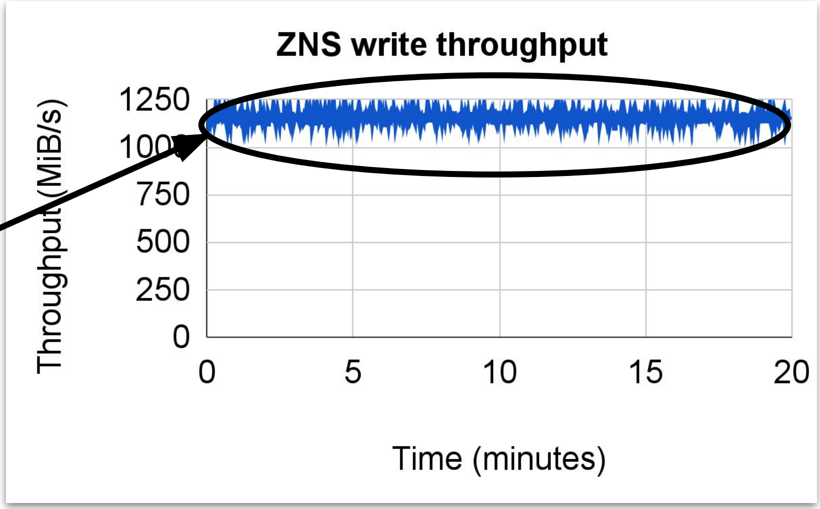


[1] IEEE CLUSTER'23, Krijn Doekemeijer; Nick Tehrany; Balakrishnan Chandrasekaran; Matias Bjørling; Animesh Trivedi, Performance characterization of NVMe Flash Devices with Zoned Namespaces (ZNS)

Solution: data-placement SSDs

- Expose I/O management to host
- Example: ZNS, FDP
- Achievable stable I/O performance





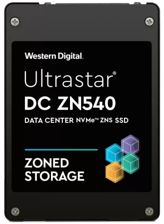
[1] IEEE CLUSTER’23, Krijn Doekemeijer; Nick Tehrany; Balakrishnan Chandrasekaran; Matias Bjørling; Animesh Trivedi, Performance characterization of NVMe Flash Devices with Zoned Namespaces (ZNS)

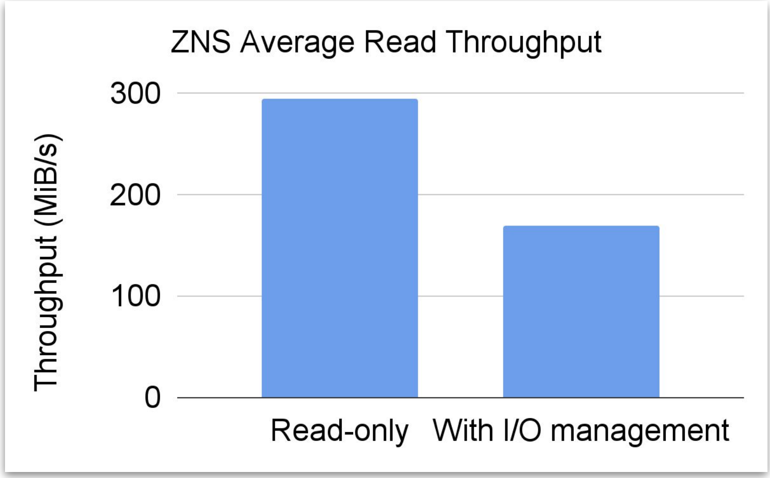
Host I/O management interferes

- We observed all ZNS I/O management operations to interfere

Host I/O management interferes

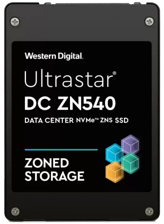
- We observed all I/O management operations to interfere with I/O
- For example on read:

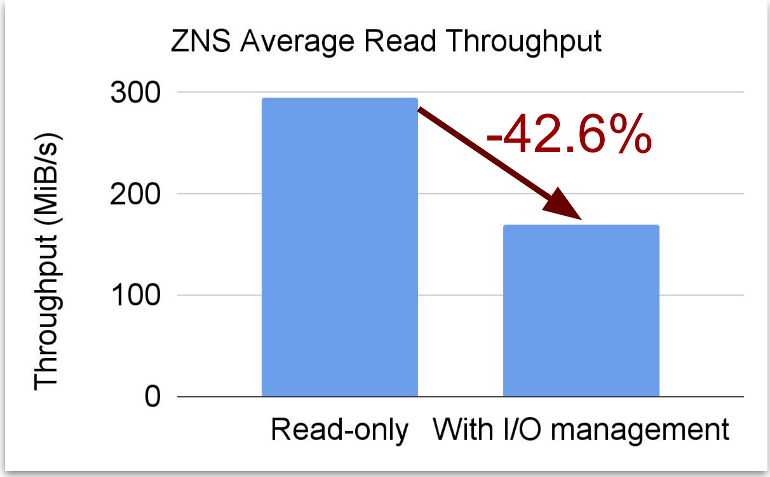




Host I/O management interferes

- We observed all I/O management operations to interfere with I/O
- For example on read:





- **ZNS performance is only stable if I/O management is done efficiently**

What we will discuss today

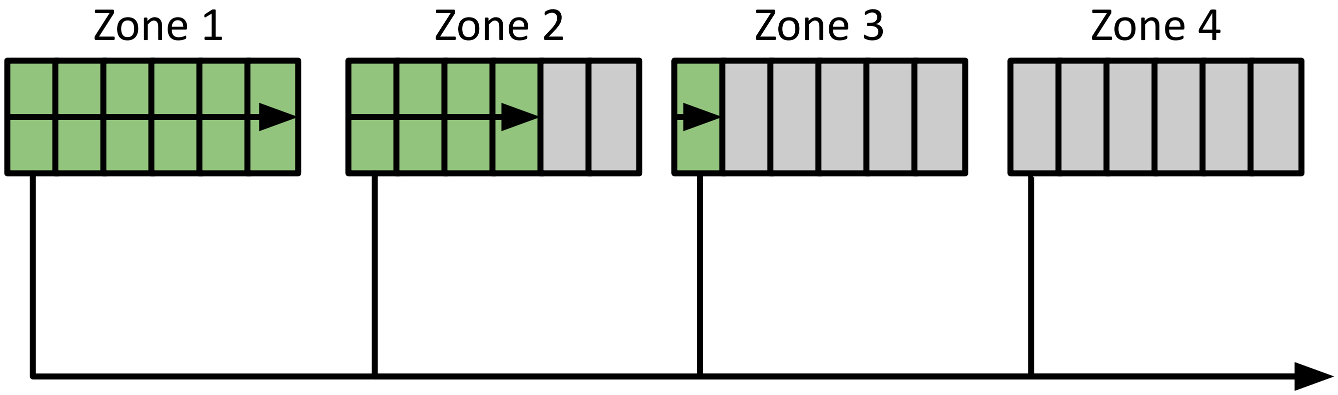
Key problem: How to deal with I/O management performance interference?

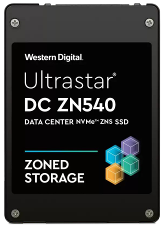
Our solutions:

- **S1 – Characterize**: ZNS I/O management interference
- **S2 – Emulate**: ConfZNS++
- **S3 – Mitigate:** 2x host solutions

Background: ZNS SSDs

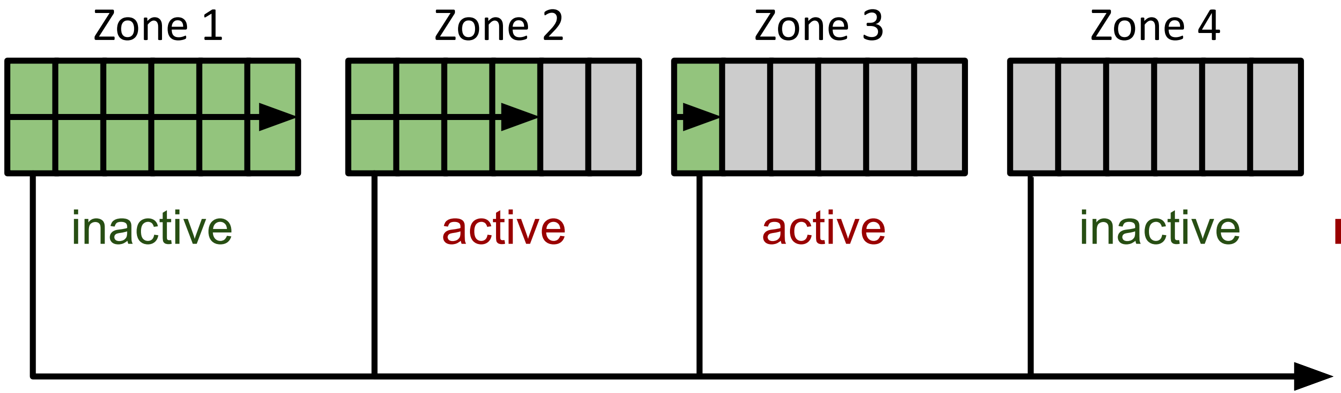
- ZNS: Storage as a series of **disjoint** sequential write zones



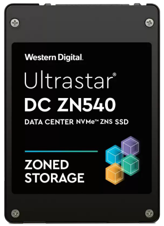


Background: ZNS SSDs

- ZNS: Storage as a series of **disjoint** sequential write zones
- Limited resources: **limited number of active zones**



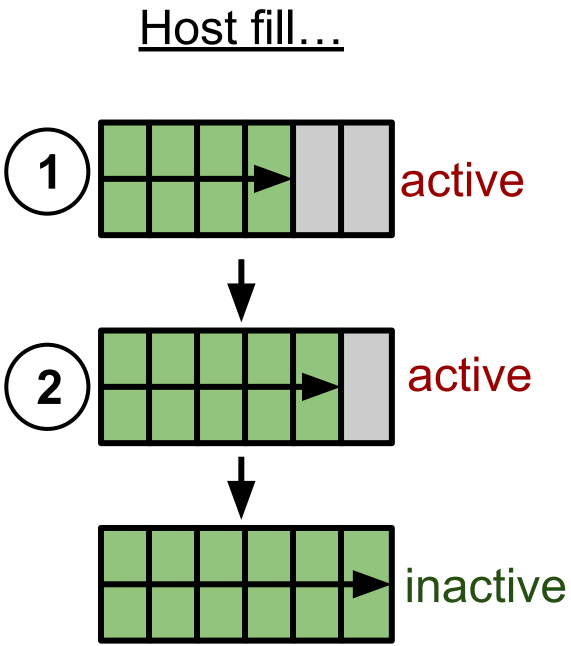
max 2 active



VU

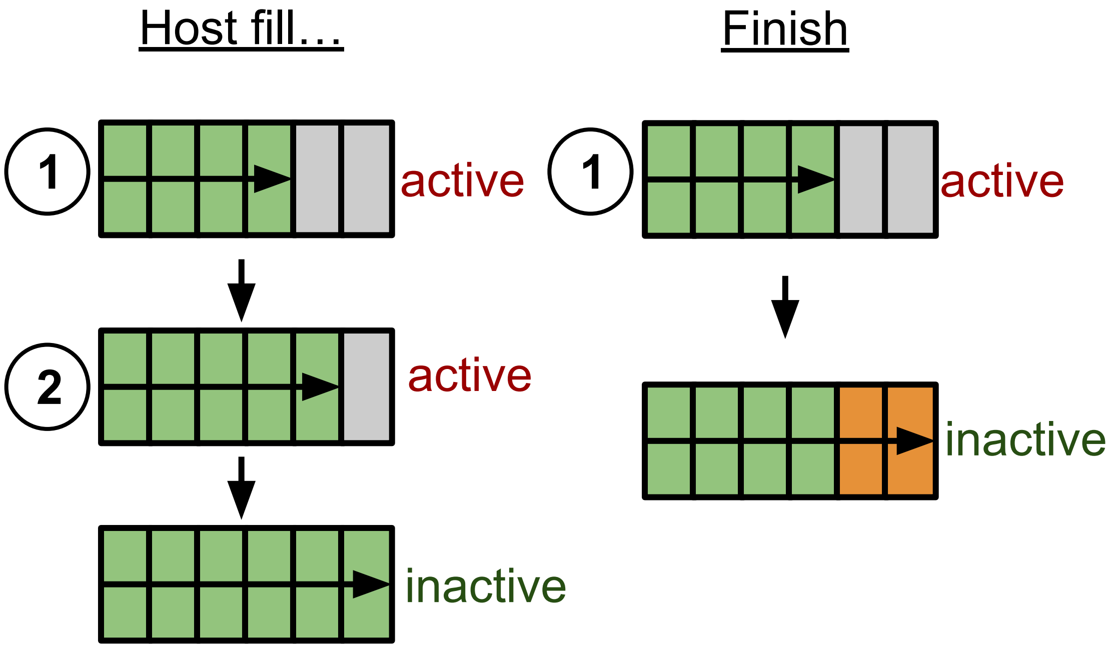
Background: I/O management

- How do we release a zone's resources?



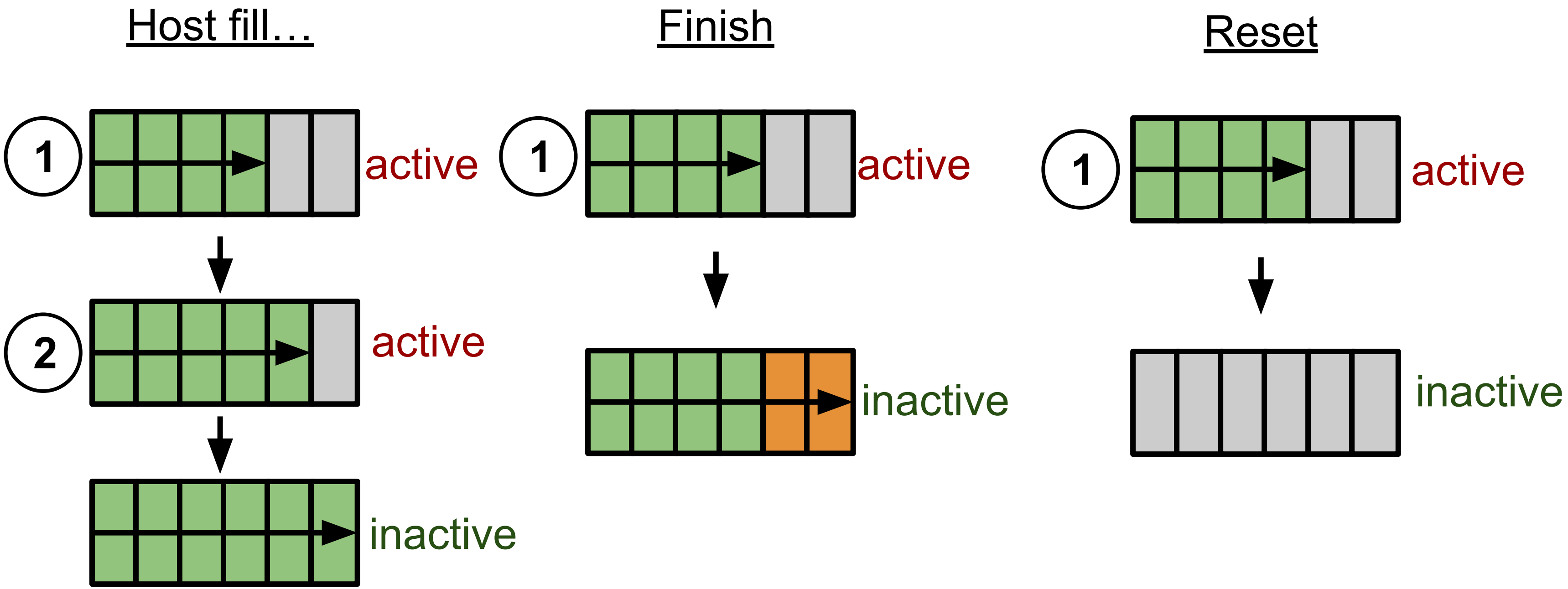
Background: I/O management

- How do we release a zone's resources?
- What if we do not have data → **I/O management operations!**

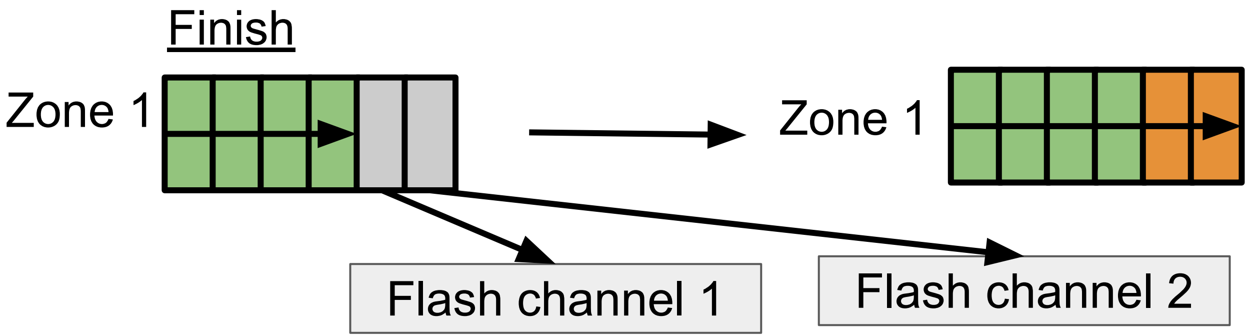


Background: I/O management

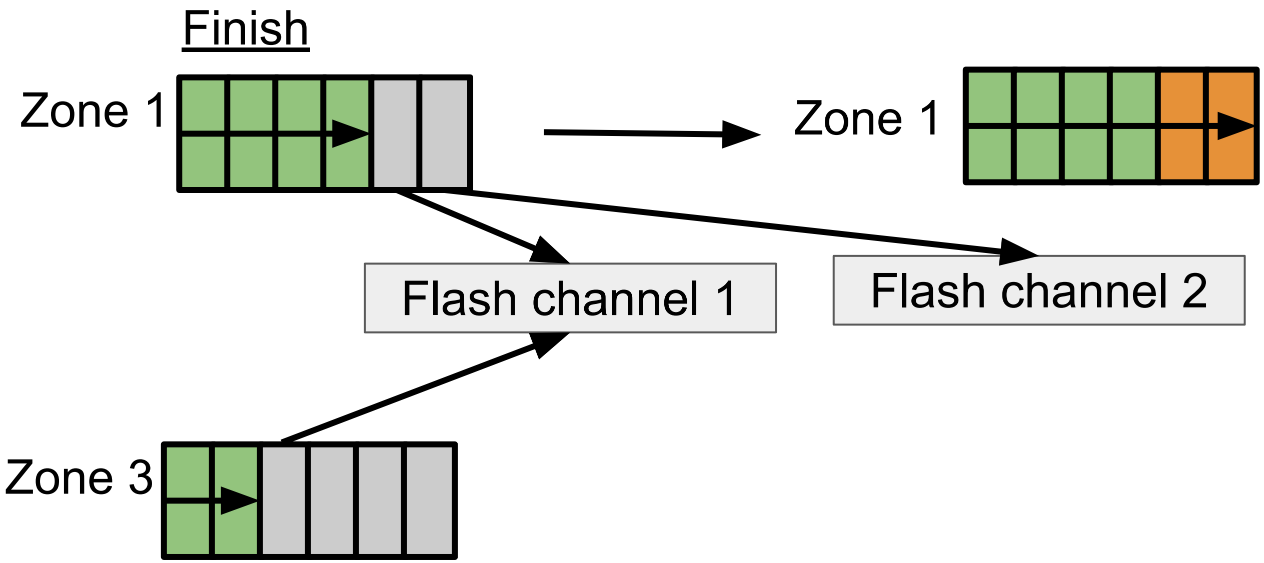
- How do we release a zone's resources?
- What if we need to delete → **I/O management operations!**



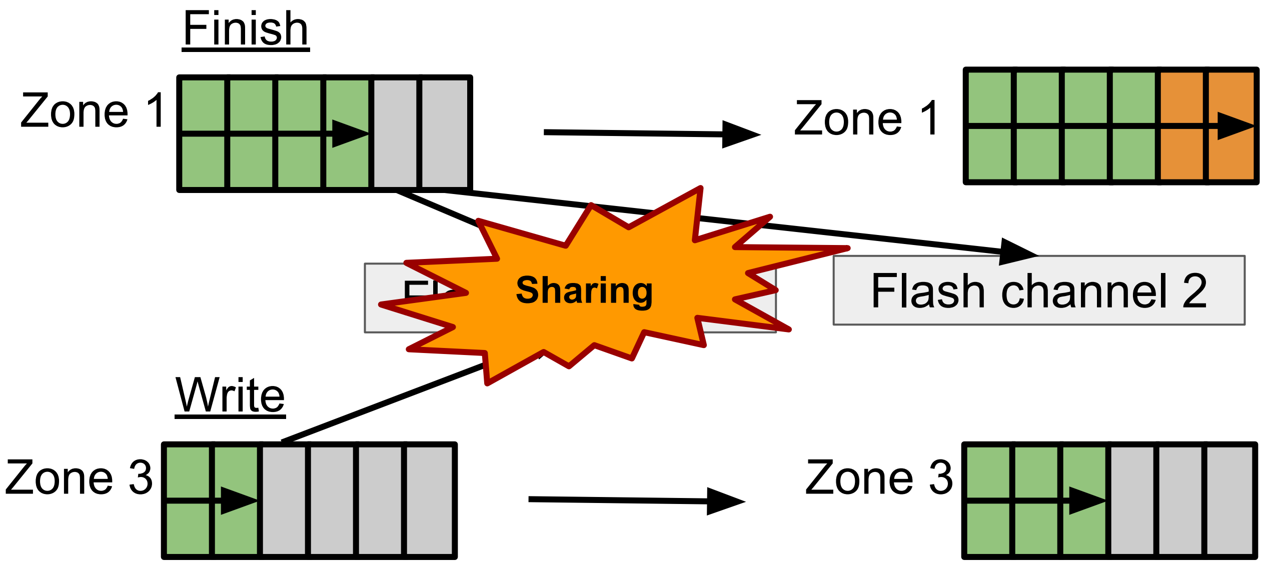
Background: I/O management interference



Background: I/O management interference



Background: I/O management interference

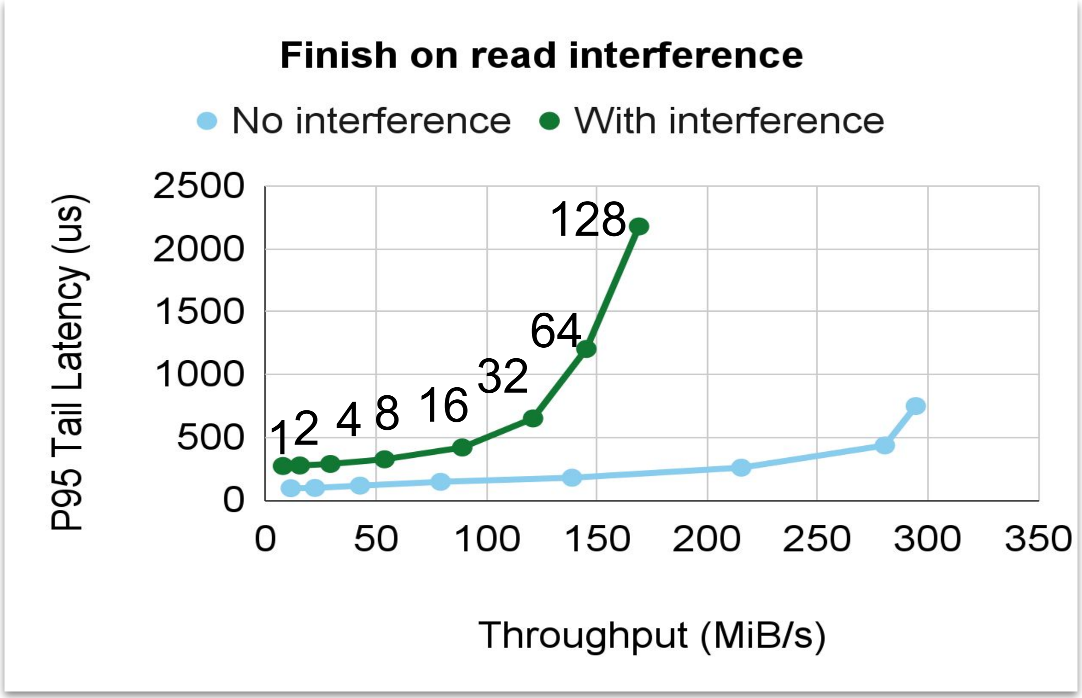


S1 - Finish interference on I/O

Experiment: Reading from ZNS with concurrent finish operations

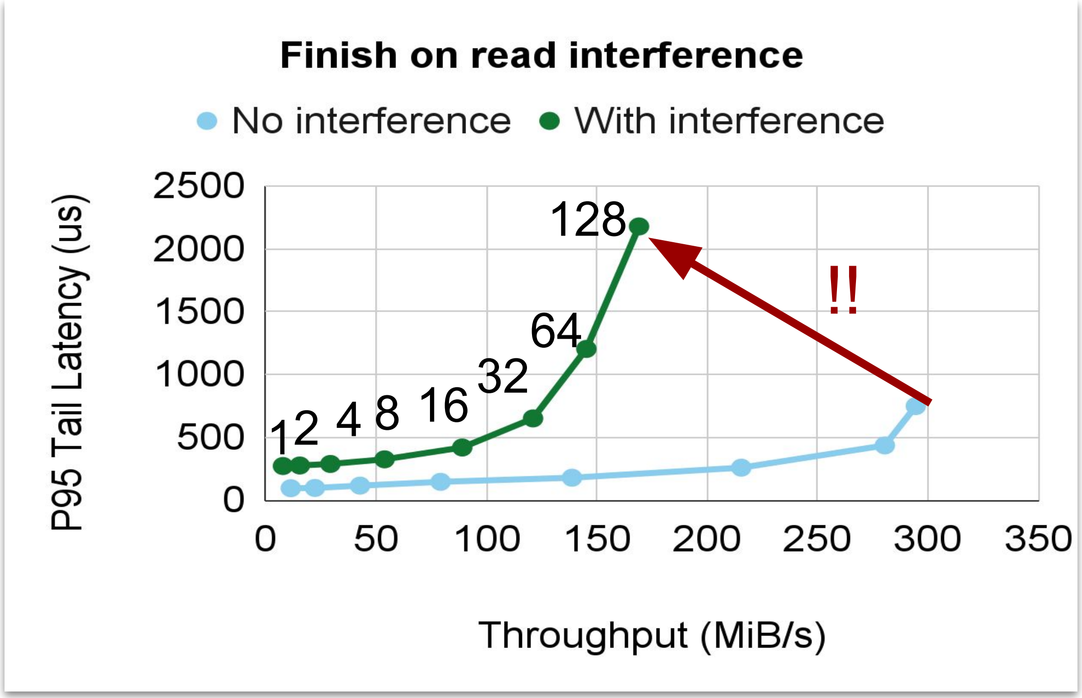
S1 - Finish interference on I/O

Experiment: Reading from ZNS with concurrent finish operations



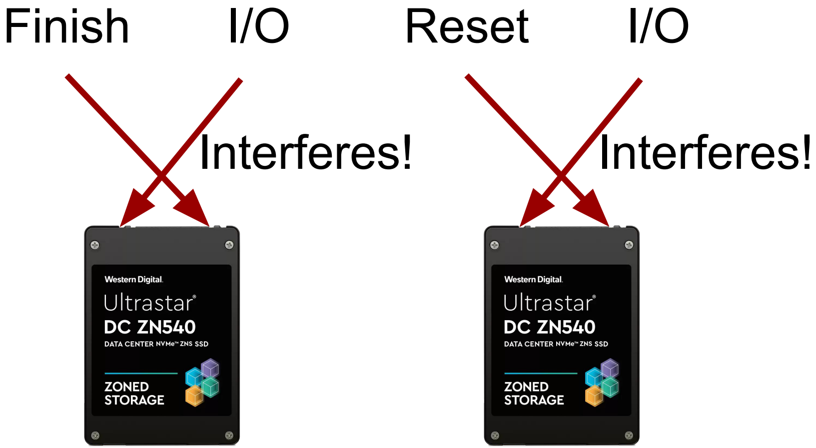
S1 - Finish interference on I/O

Experiment: Reading from ZNS with concurrent finish operations



S1 - What else did we characterize?

- **8 key performance observations!** (see paper)
- **Finish** interferes significantly on:
  - Write
  - Read
- **Reset** interferes significantly on:
  - Write



S2 - Emulation: ConfZNS++

Problem: No emulator has function-realistic management performance
Consequence: Host software performance is not representative
Solution: Our ConfZNS++ emulator

Supported performance models

Emulator	**I/O: Read/Write**	**Reset**	**Finish**	**Zone mapping**
ConfZNS	✓	—	✗	✗
ConfZNS++	✓	✓	✓	✓

[1] SYSTOR'23, Inho Song, Myounghoon Oh, Bryan Suk Joon Kim, Seehwan Yoo, Jaedong Lee, and Jongmoo Choi, ConfZNS: A Novel Emulator for Exploring Design Space of ZNS SSDs

S2 - ConfZNS++: finish design

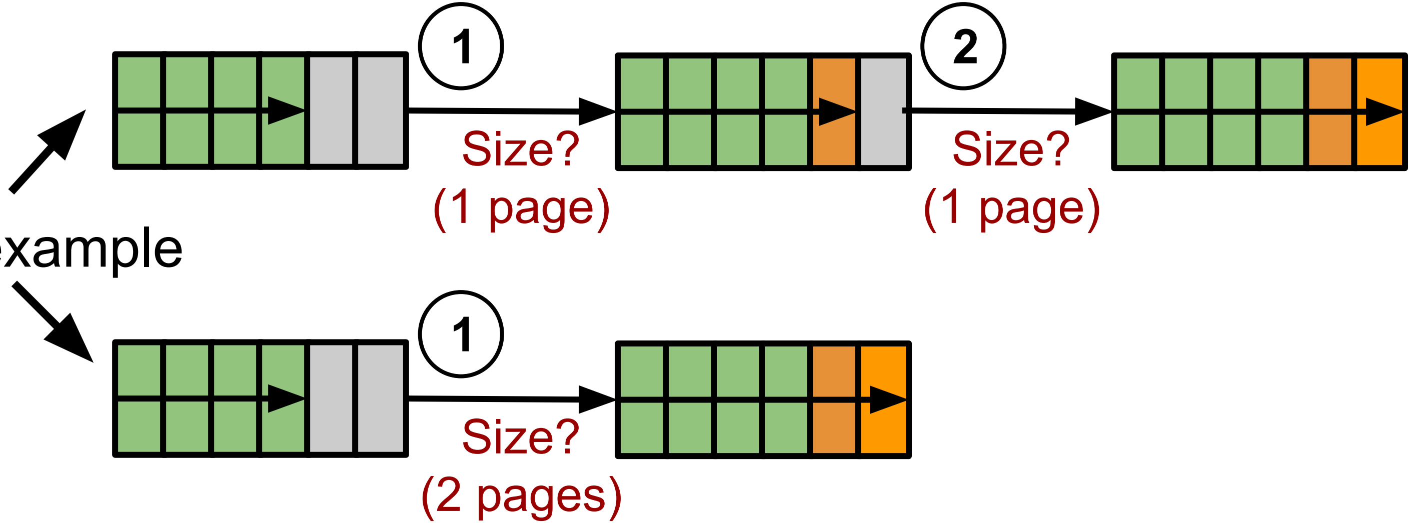
S2 - ConfZNS++: finish design

1. What request size?
2. Pause between requests?
3. Preempt on concurrent I/O?

S2 - ConfZNS++: finish design

1. What request size?
2. Pause between requests?
3. Preempt on concurrent I/O?

For example

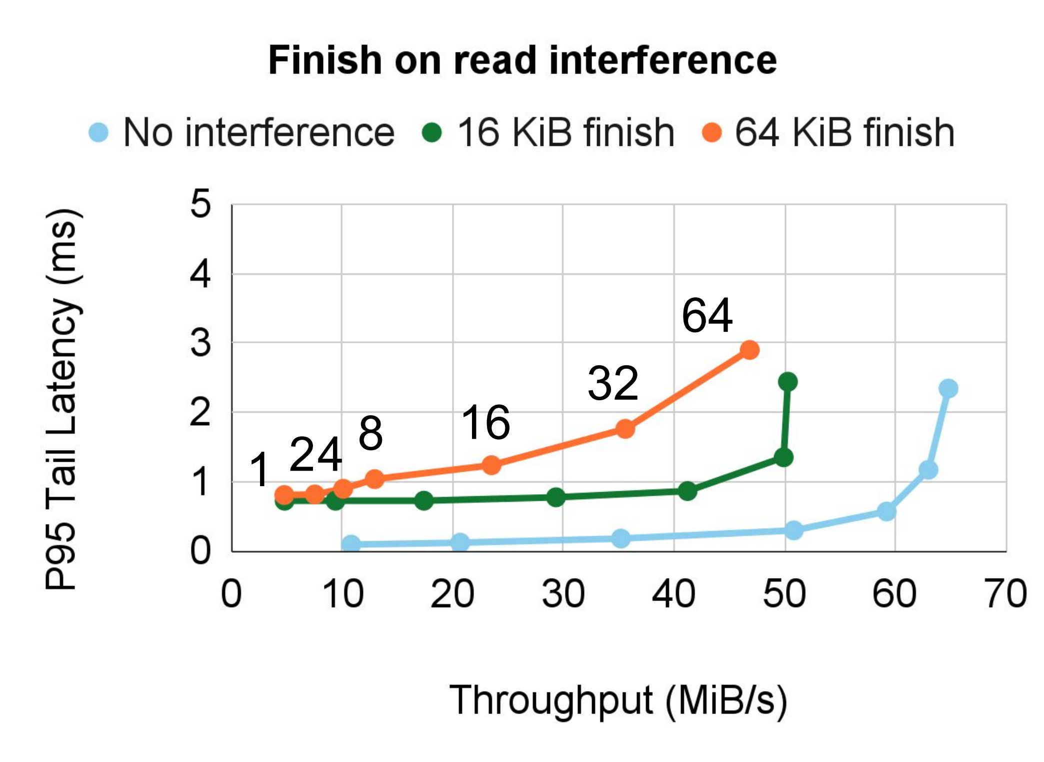


S2 - ConfZNS++: finish interference

Experiment: Reading from ZNS with concurrent finish operations

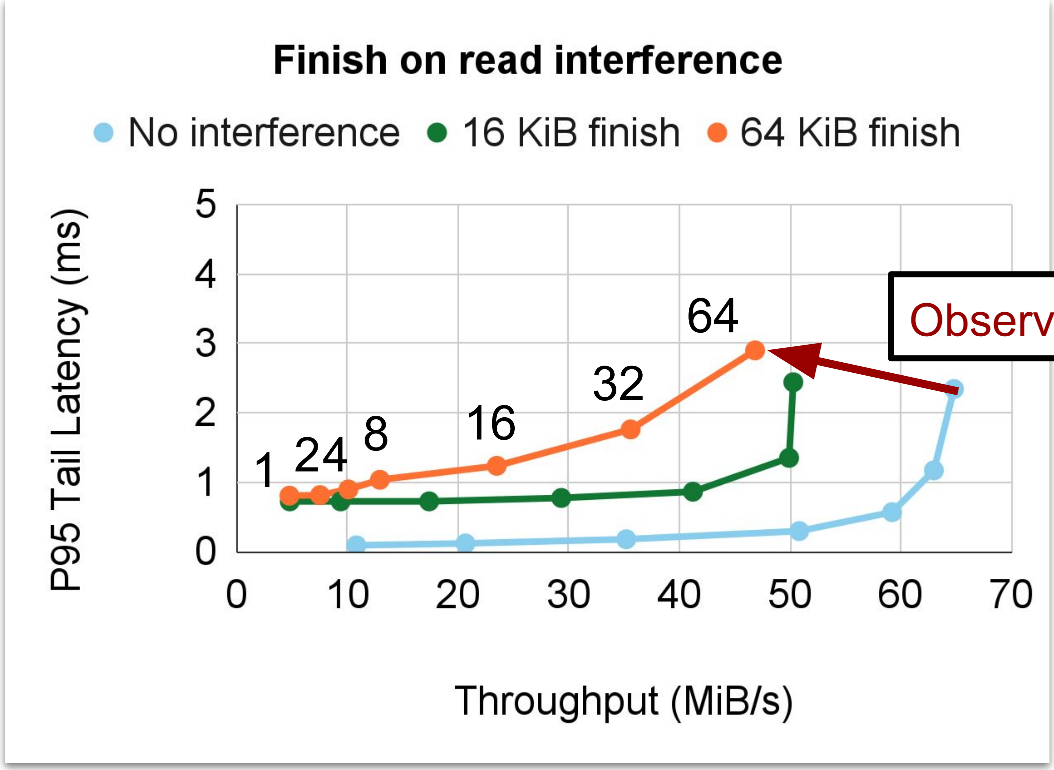
S2 - ConfZNS++: finish interference

Experiment: Reading from ZNS with concurrent finish operations



S2 - ConfZNS++: finish interference

Experiment: Reading from ZNS with concurrent finish operations

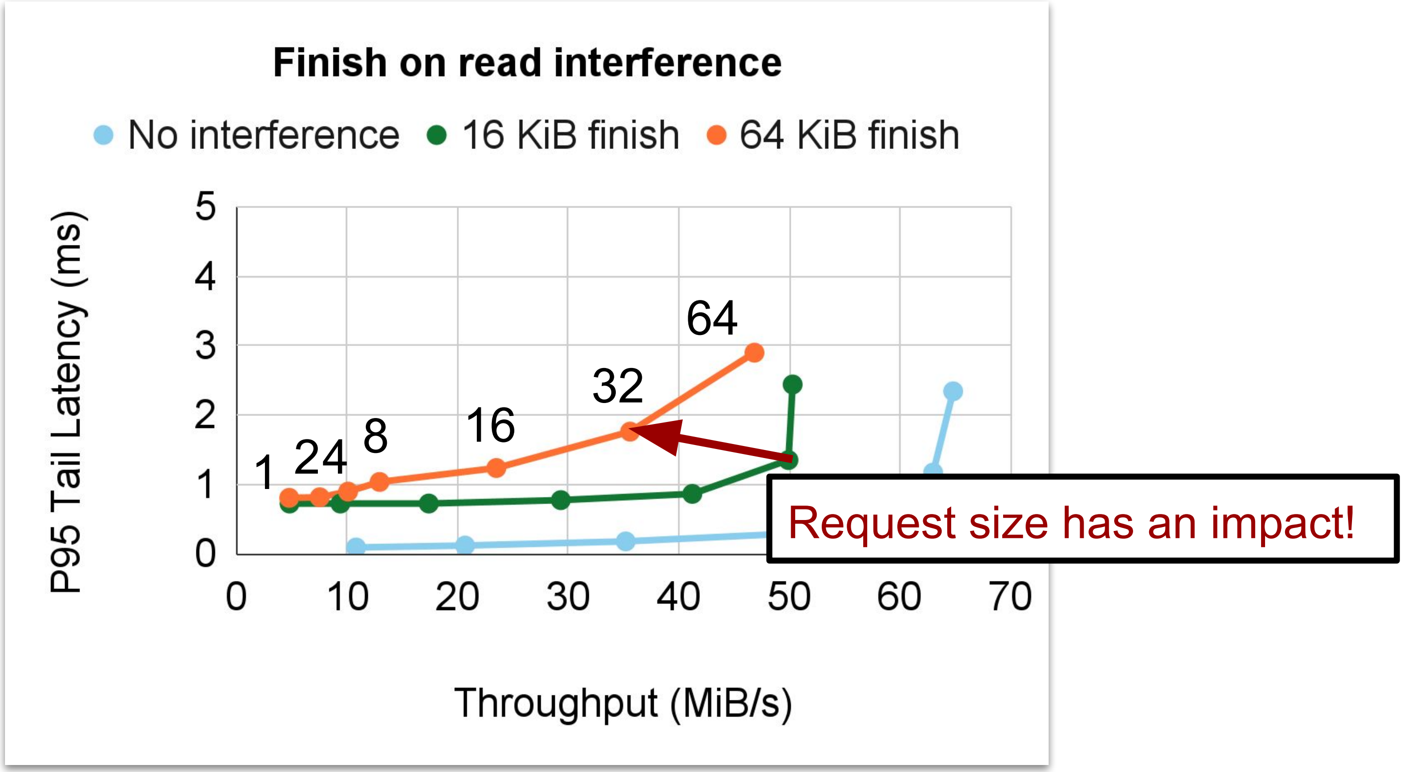


Observable Interference!

VU

S2 - ConfZNS++: finish interference

Experiment: Reading from ZNS with concurrent finish operations

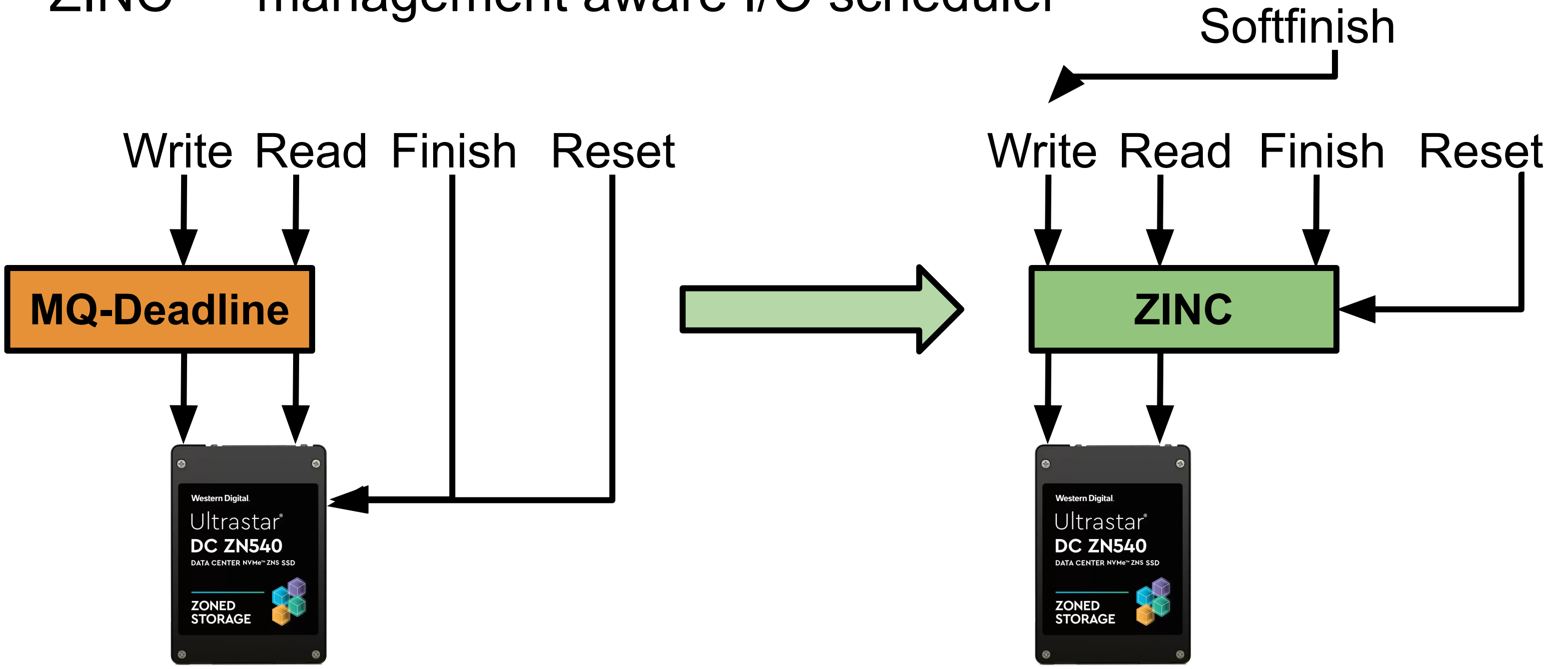


S2 - ConfZNS++: what else?

- ConfZNS++ has:
  1. 3x Finish designs
  2. 5x Reset designs
  3. 2x Zone mapping designs
- Highly configurable to support adding new decisions

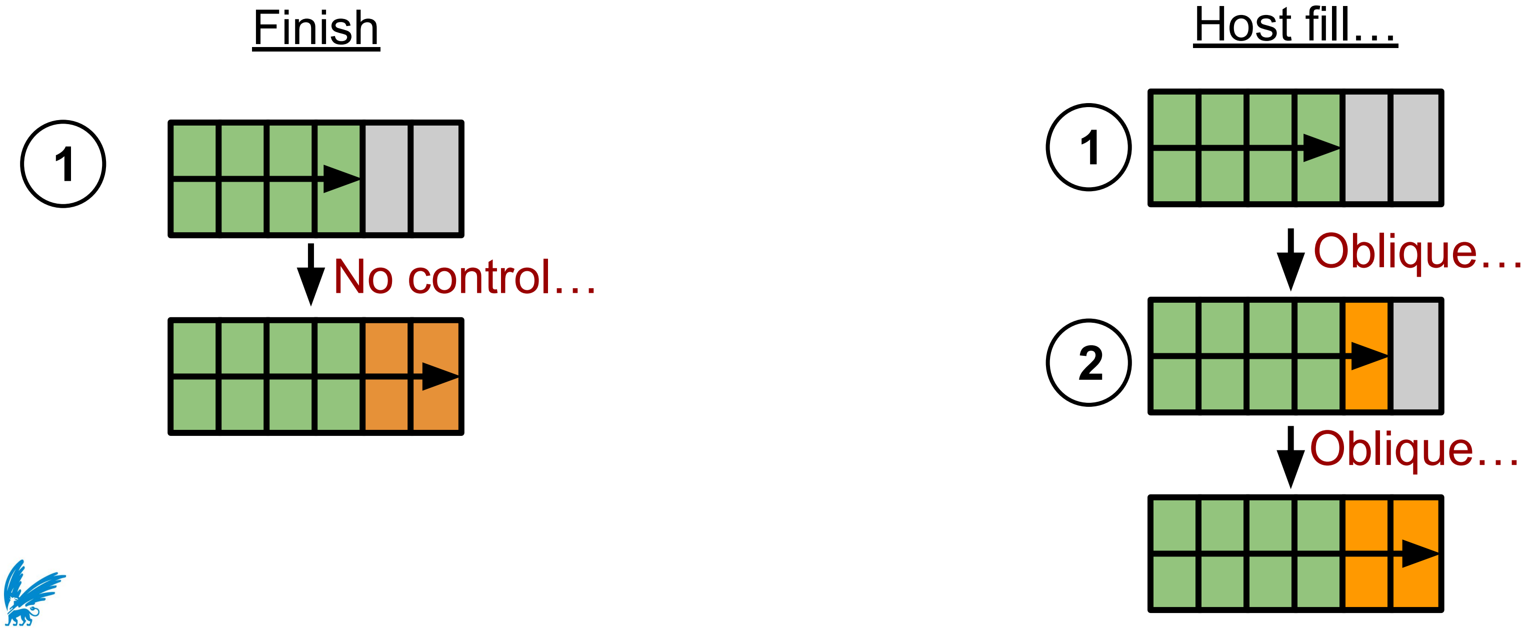
S3 - Reducing interference

- **Demonstration** of reducing interference:
  1. Softfinish — host-managed finish
  2. ZINC — management aware I/O scheduler



S3 - Softfinish background

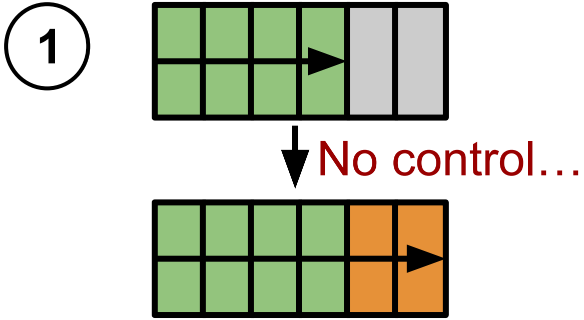
- **Finish control knobs**?



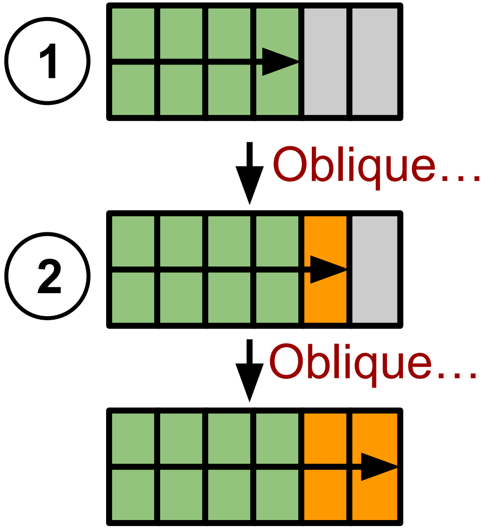
S3 - Softfinish background

- **Finish control knobs**?
  - Explicit finish: too transparent
  - Host fill: too oblique

Finish

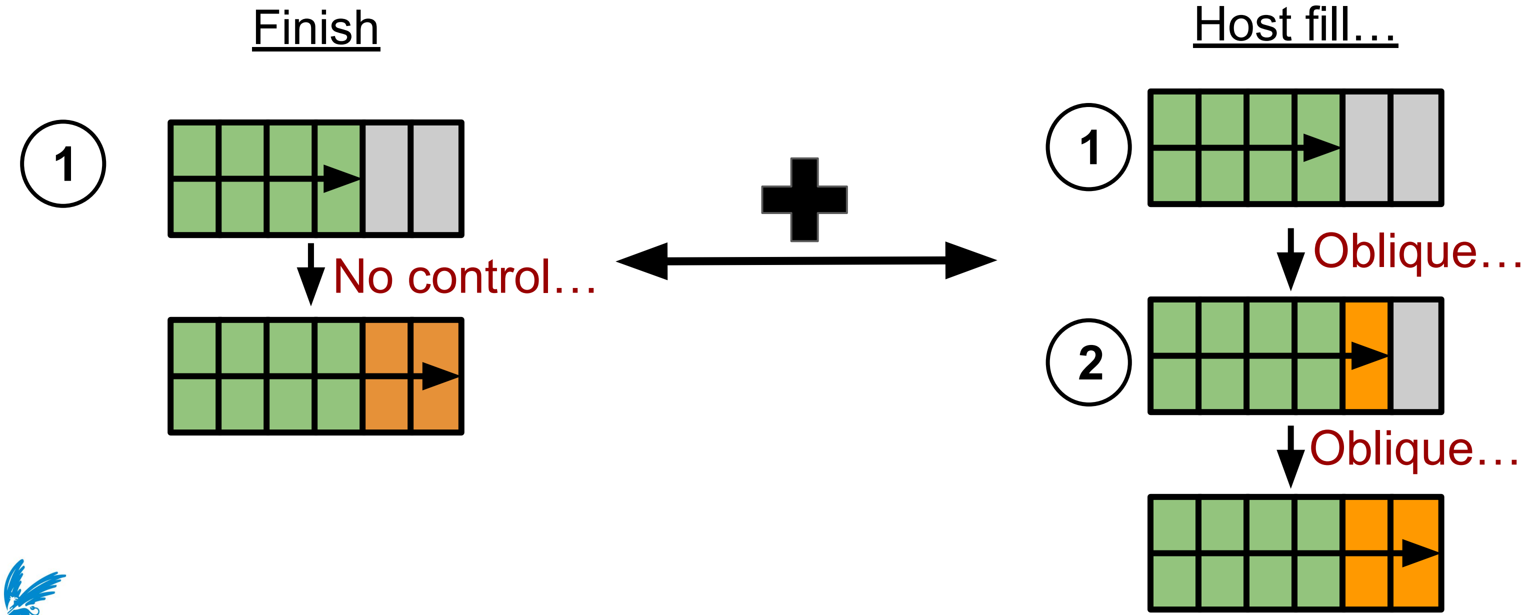


Host fill…



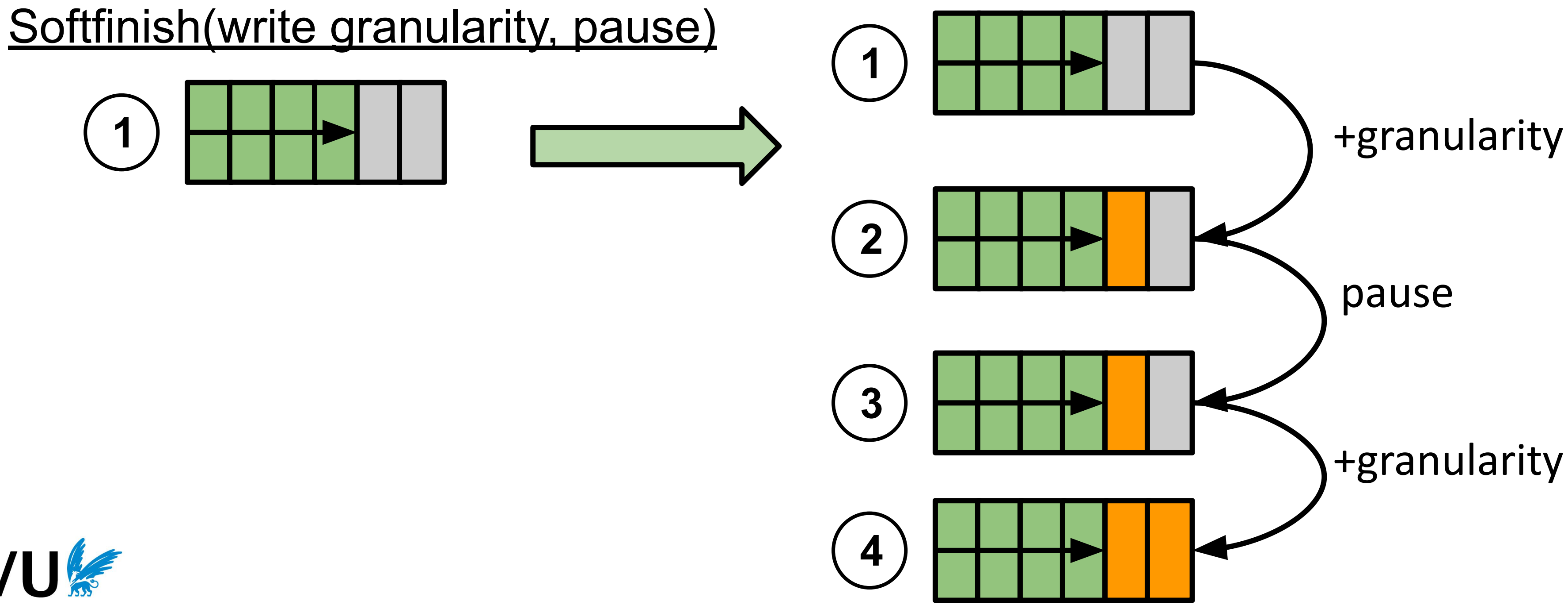
S3 - Softfinish background

- **Finish control knobs**?
  - Explicit finish: too transparent
  - Host fill: too oblique
- Idea: Combine finish and host filling



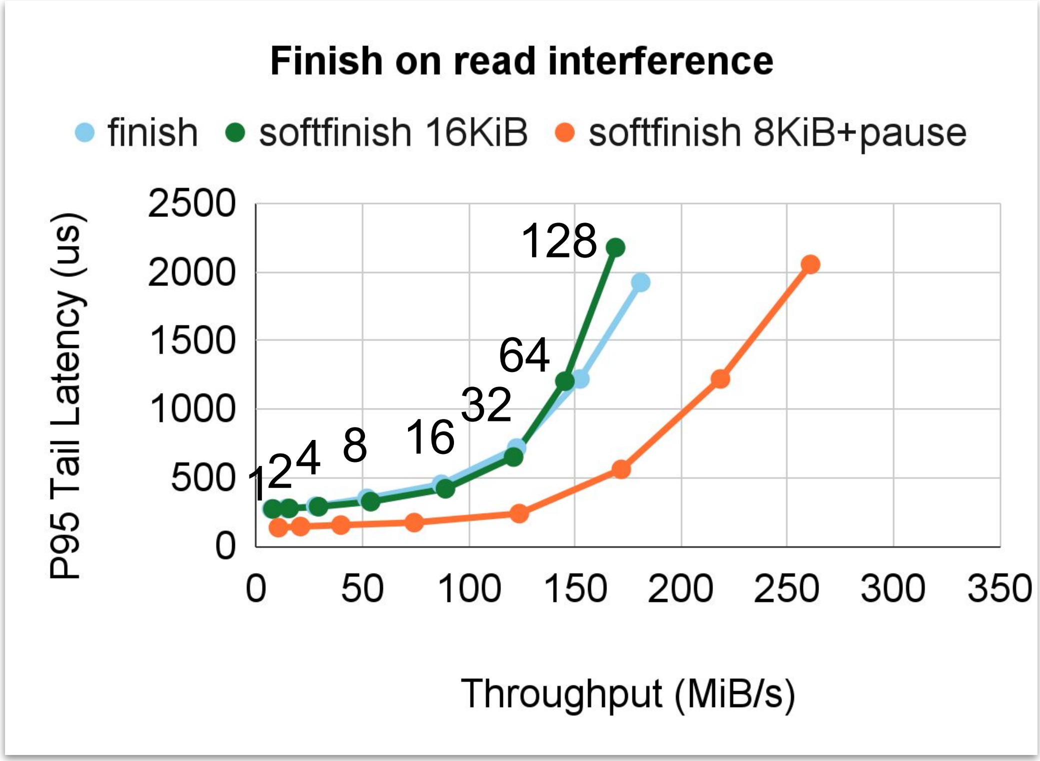
S3 - Softfinish design

- Transparent host filling with control knobs



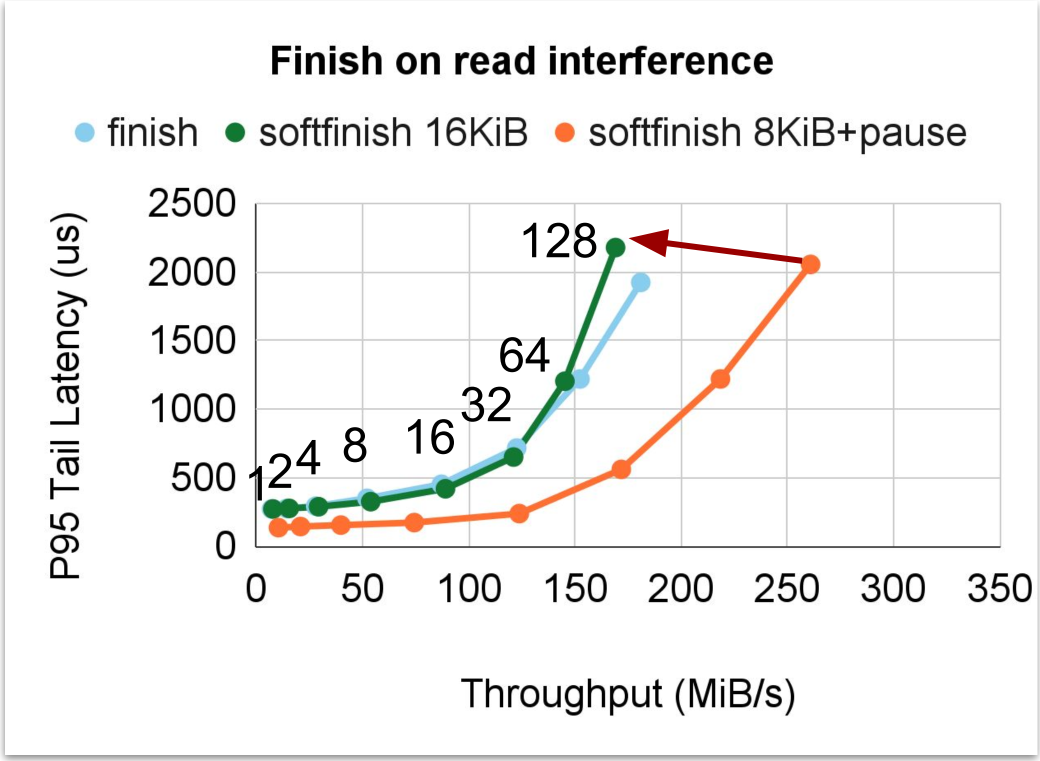
S3 - Softfinish results

Experiment: Softfinish at different granularities on read performance



S3 - Softfinish results

Experiment: Softfinish at different granularities on read performance



VU

More details/results in the paper ...

Exploring I/O Management Performance in ZNS with ConfZNS++

Krijn Doekemeijer
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands

Dennis Maisenbacher
Western Digital
Copenhagen, Denmark

Zebin Ren
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands

Nick Tehrany*
BlueOne Business Software LLC
Beverly Hills, California, USA

Matias Bjørling
Western Digital
Copenhagen, Denmark

Animesh Trivedi*
IBM Research Europe
Zurich, Switzerland

ABSTRACT

Flash-based storage is known to suffer from performance unpredictability due to interference between host-issued I/O and device-side I/O management. SSDs with data placement capabilities, such as Zoned Namespaces (ZNS) and Flexible Data Placement (FDP), expose selective device-side I/O management operations to the host to provide predictable performance. In this paper, we demonstrate that these host-issued I/O management operations lead to performance interference with host-issued I/O. Indeed, we find that the I/O management operations introduced by ZNS and FDP create I/O interference, leading to significant performance losses. Despite the performance implications, we observe that ZNS research frequently uses emulators (over 20 recently published papers), but no emulator currently has function-realistic models for I/O management. To address this gap, we identify ten ZNS I/O management designs, explain how they interfere with I/O, and introduce ConfZNS++, a function-realistic emulator with native I/O management support, providing future research with the capability to explore these designs. Additionally, we introduce two actionable host-managed solutions to reduce ZNS management interference: ZINC, an I/O scheduler prioritizing I/O over I/O management, and the `softfinish` operation, a host-managed implementation of the `finish` operation. In our experiments, ZINC reduces `reset` interference by 56.9%, and `softfinish` reduces `finish` interference by 50.7%.

*Work done while the author was at the Vrije Universiteit Amsterdam.

SYSTOR '24, September 23–24, 2024, Virtual, Israel

ACM ISBN 979-8-4007-1181-7/24/09.
https://doi.org/10.1145/3688351.3689160

Table 1: *Performance models supported in ZNS emulators; models with "—" are incomplete.*

Model	*FEMU*	*NVMeVirt*	*ConfZNS*	*ConfZNS++*
`Read` and `write`	✗	✓	✓	✓
`Reset`	✗	—	—	✓
`Finish`	✗	✗	✗	✓
Zone mapping	✗	✗	✗	✓

CCS CONCEPTS

• **Information systems** → **Storage management**; *Flash memory*; • **Software and its engineering** → *Secondary storage*.

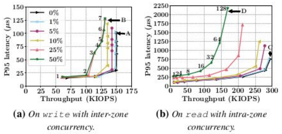
KEYWORDS

ZNS, Interference, NVMe Flash Storage, Emulation

1 INTRODUCTION

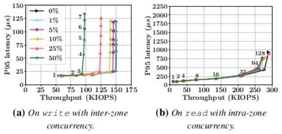
Solid-state drives (SSDs) have become the de facto standard for storing and processing data at high speeds. Today, SSDs can deliver microsecond access latencies, millions of I/O operations per second, and gigabytes of bandwidth per second [53]. However, delivering predictable SSD performance is challenging due to the significant required management effort for flash-based SSDs [2, 42] (e.g., garbage collection, parallelism management, wear-leveling). This flash management is traditionally hidden from the host behind the block interface, which exposes the SSD as a read/write anywhere device. To support this block-based interface, an SSD manages media transparently in the background but causes significant performance interference and, as a result, performance unpredictability in both latency and throughput [4, 15, 19, 20, 24, 33, 41].

To resolve this unpredictability, researchers have advocated for extending the conventional block-based SSD interface toward a more host-controlled interface. Examples of such interfaces include Software-Defined Flash (SDF), Open-Channel, Streams, and recently introduced Zoned Namespaces (ZNS) and Flexible Data Placement (FDP) [1, 4, 5, 29, 48]. We call SSDs, which support such interfaces, data placement SSDs.



(a) *On* `write` *with inter-zone concurrency.* **(b)** *On* `read` *with intra-zone concurrency.*

Figure 2: `Finish` *interference on (a)* `write`*; (b)* `read`*.*



(a) *On* `write` *with inter-zone concurrency.* **(b)** *On* `read` *with intra-zone concurrency.*

Figure 3: `Reset` *interference on (a)* `write`*; (b)* `read`*.*

Setup: Our benchmarking setup is shown in Tab. 2. We deploy all our benchmarking workloads on the ZN540 ZNS SSD. To confirm the generalizability of our `finish` interference results, we repeated our `finish` experiments on ZNS SSD "B" and made similar observations. We evaluate the interference of all I/O management (i.e., `finish` and `reset`) operations on all I/O operations exposed by ZNS. ZNS exposes two I/O operations for writing: `write` and `append`; and one operation for reading: `read`.

We benchmark I/O management interference by running a fio [25] process with two concurrent threads: a foreground thread running I/O and an interfering background thread running I/O management. Both threads are spatially separated (disjoint zones). We measure performance as the foreground thread's average throughput and P95 latency.

We scale the foreground thread's I/O using a setup similar to a previous study [14]. We define scalability in the *concurrency level* (CL), which is the number of concurrently issued operations. `Write` is inter-zone scalable (concurrent operations in disjoint zones), and `append` and `read` are intra-zone scalable (concurrent operations in a single zone). With inter-zone scalable operations we increase CL with the number of threads, each to a disjoint zone, and with intra-zone scalable operations we increase CL by increasing a single thread's queue depth to a single zone. For `write` and `append`, we use concurrency levels 1–7, and for `read` 1–128 in powers of 2 (based on each operation's saturation point).

We scale the background thread's management operations by increasing its *intensity*. Intensity is the maximum allowed throughput per second and is controlled by throttling an operation to a percentage of its peak performance (e.g., 50%). Before each workload, we first measure the operation's peak performance to determine the intensities to evaluate. In a workload, each `finish` is additionally preceded by a single page `write` as `finish` only affects zones with data; we assume this `write`'s interference is negligible.

For `reset` and `read` we prefill their assigned zones. We modify fio to support `finish` and `append` as workloads and exclusively use the *io_uring passthrough* mechanism [26]. *io_uring passthrough* delivers operations directly to the NVMe device driver, bypassing the block layer, which achieves performance close to the device hardware.

`Finish` interference: We now evaluate the interference of `finish` on I/O. Fig. 2 shows `finish` on I/O interference. The figure shows I/O performance in KIOPS (x-axis, higher is better) and P95 latency (y-axis, lower is better). The points on the lines represent the CL, and we observe that the throughput and latency increase monotonically with the CL (hence, one line is annotated). Each line is presented with a different percentage, indicating `finish` intensity. We measure peak `finish` throughput as 1.1 GiB/s (~1 IOPS); for example 25% equals approximately one operation every four seconds. In the plot, when an I/O operation saturates the device, a queuing effect takes place where the throughput remains stable, but latency increases sharply. We call this point the *saturation point*. For example, `write`'s saturation point is at the knee of the 0% line (CL=3, 149.3 KIOPS at 25.7 μs). Note that we do not plot `append` interference as it is comparable to `write` interference except for 50% past CL>2; `append` throughput decreases past this point (we do not know the reason for this anomaly). The similarity between `write` and `append` is expected as both issue the same operations to flash; they only differ in their implementation firmware (e.g., acceleration).

We observe that `finish` interferes significantly with all three I/O operations and make four observations. First (**Obs #1**), write operations (i.e., `write` and `append`) do not experience interference before their saturation point (CL=3 in Fig. 2a). Write performance at CL<3 is identical for all `finish` intensities. Second (**Obs #2**), write interference is significant beyond the saturation point and increases with the concurrency level. `Write` interference is highest at CL=7 (marked "A" and "B"), where it achieves 150.0 KIOPS and 101.9 μs in isolation and 131.1 KIOPS (12.6% lower) and 128.5 μs (20.7% higher) at 50% `finish`. Third (**Obs #3**), we observe that `finish` on `read` interference occurs irrespective of the saturation point (Fig. 2b), i.e., occurs at all concurrency levels. At CL=1, `read` achieves in isolation 11.3 KIOPS and 95.7 μs, and at 50% `finish`, 7.8 KIOPS (31.4% lower) and 272.4 μs (2.8× higher). Fourth (**Obs #4**),

Take-away message

1. **SSD I/O management interferes with I/O performance!**
2. Data placement SSDs (ZNS, FDP...) expose management
   - **Host-controllable interference!**
3. This Interference was not available in emulators…
   - ConfZNS++ adds support to ZNS emulators
4. Two solutions to reduce interference on the host

Paper:
https://atlarge-research.com/pdfs/2024-confznsplusplus.pdf
Source code:
https://github.com/stonet-research/systor-confznsplusplus-artifact

Supported by NWO, Western Digital, Graph Massivizer, and 6G FNS

Thank you for listening!

Contact information:

- Mail: k.doekemeijer@vu.nl
- GitHub: https://github.com/stonet-research/systor-confznsplusplus-artifact

Paper:
https://atlarge-research.com/pdfs/2024-confznsplusplus.pdf
Source code:
https://github.com/stonet-research/systor-confznsplusplus-artifact

Supported by NWO, Western Digital, Graph Massivizer, and 6G FNS