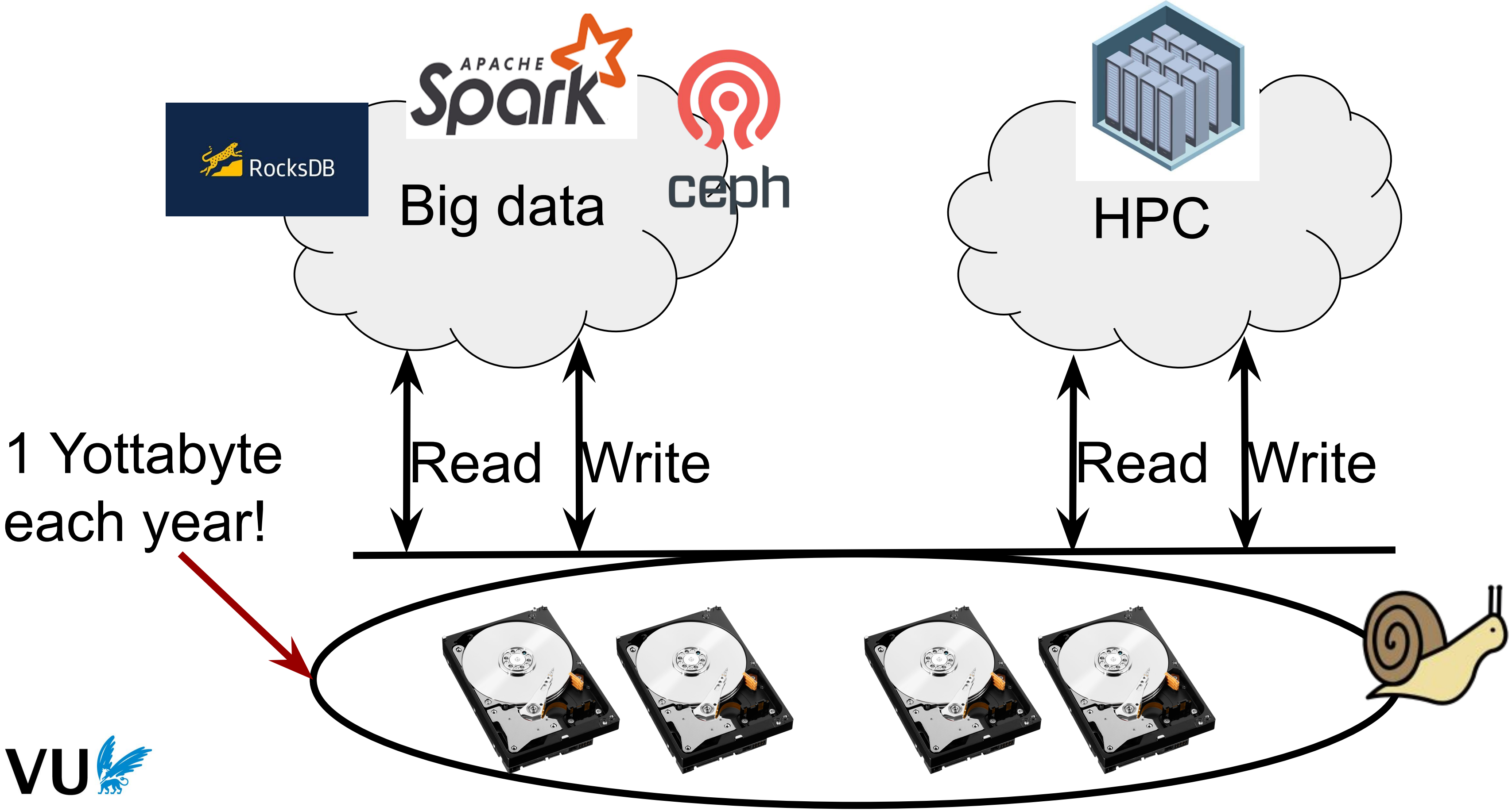
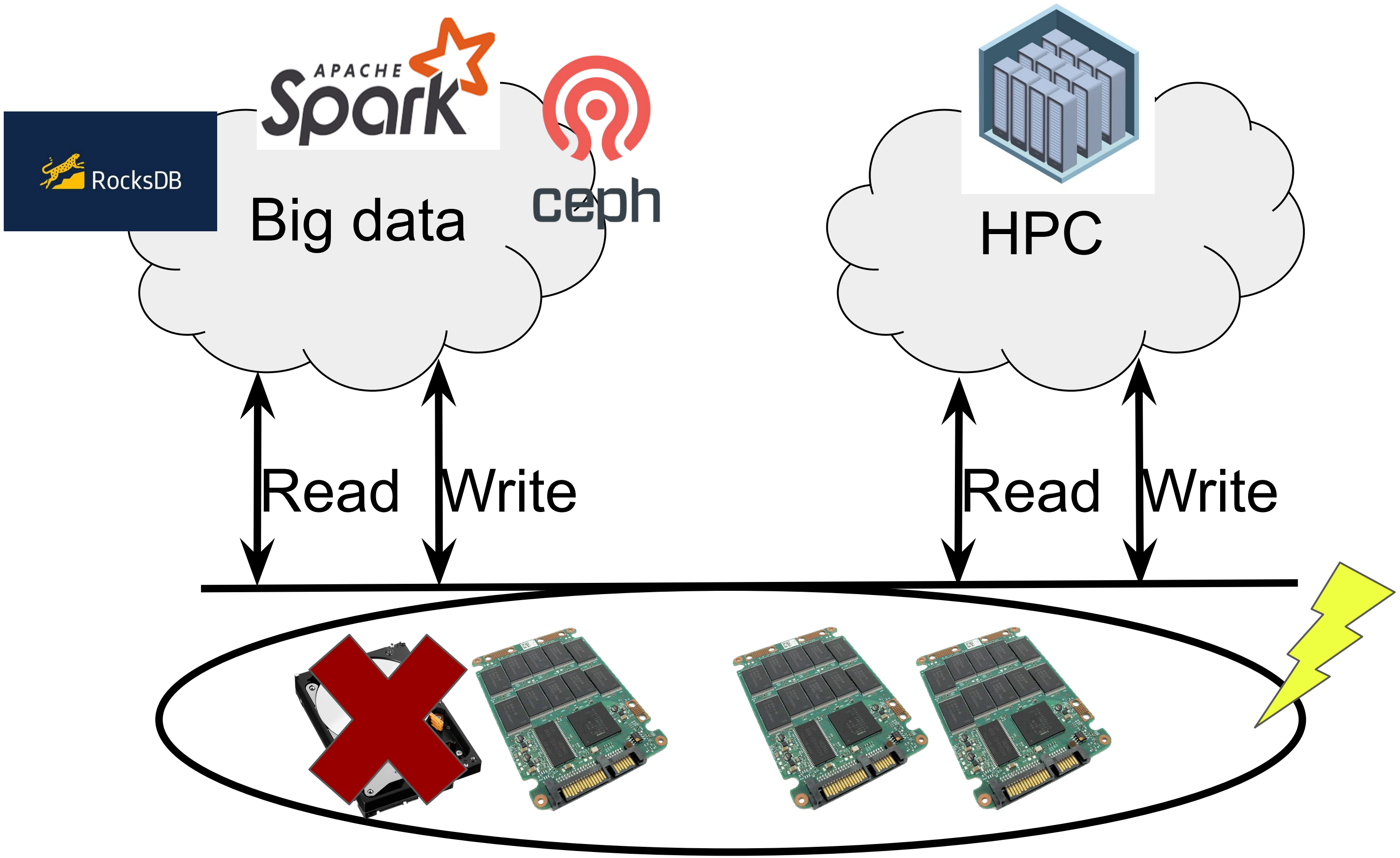
Performance Characterization of NVMe Flash Devices with Zoned Namespaces (ZNS)

Krijn Doekemeijer, Nick Tehrany, Balakrishnan Chandrasekaran, Mattias Bjørling, Animesh Trivedi

The amount of data is ever-increasing



Flash SSDs have become ubiquitous



What we will discuss today

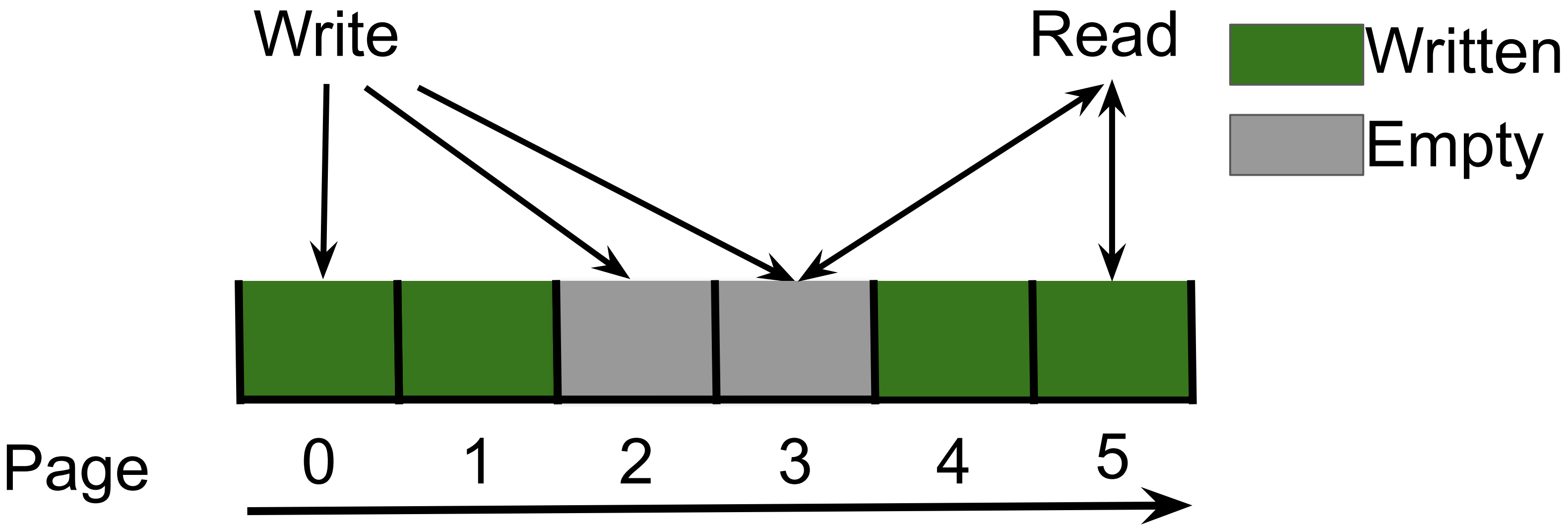
Problem:

1. Block interface does **not perform for flash**
2. **New** ZNS interface promises good performance
3. ZNS' performance characteristics are not known…

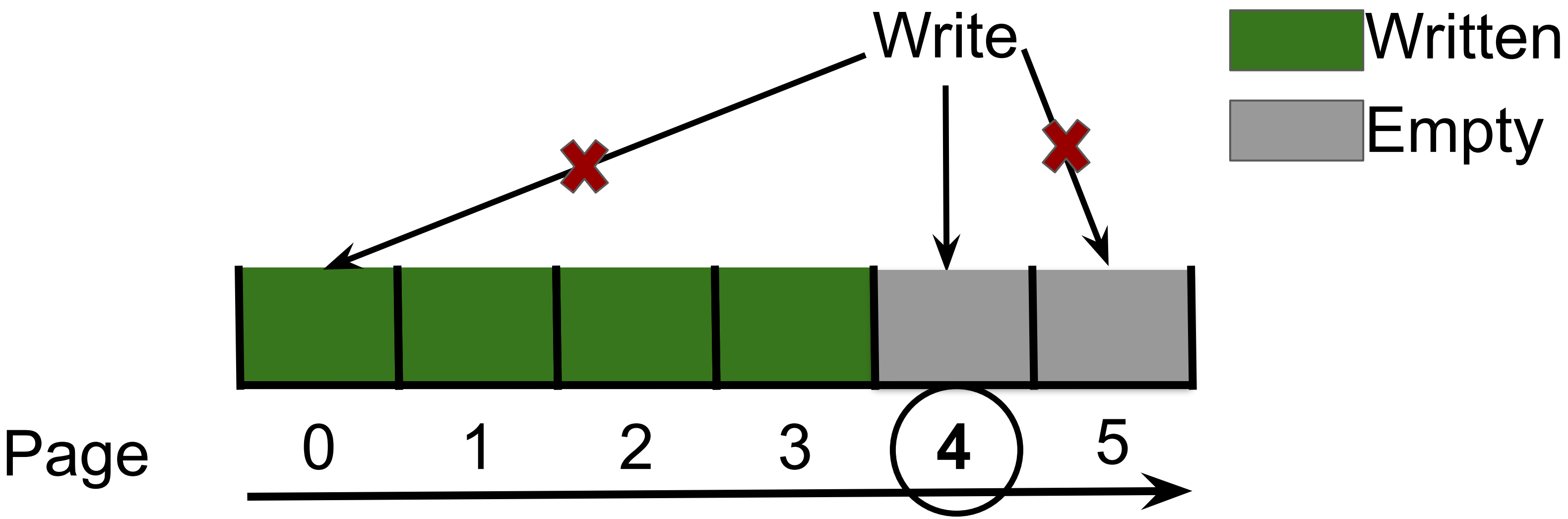
Goal: Infer ZNS performance characteristics

VU

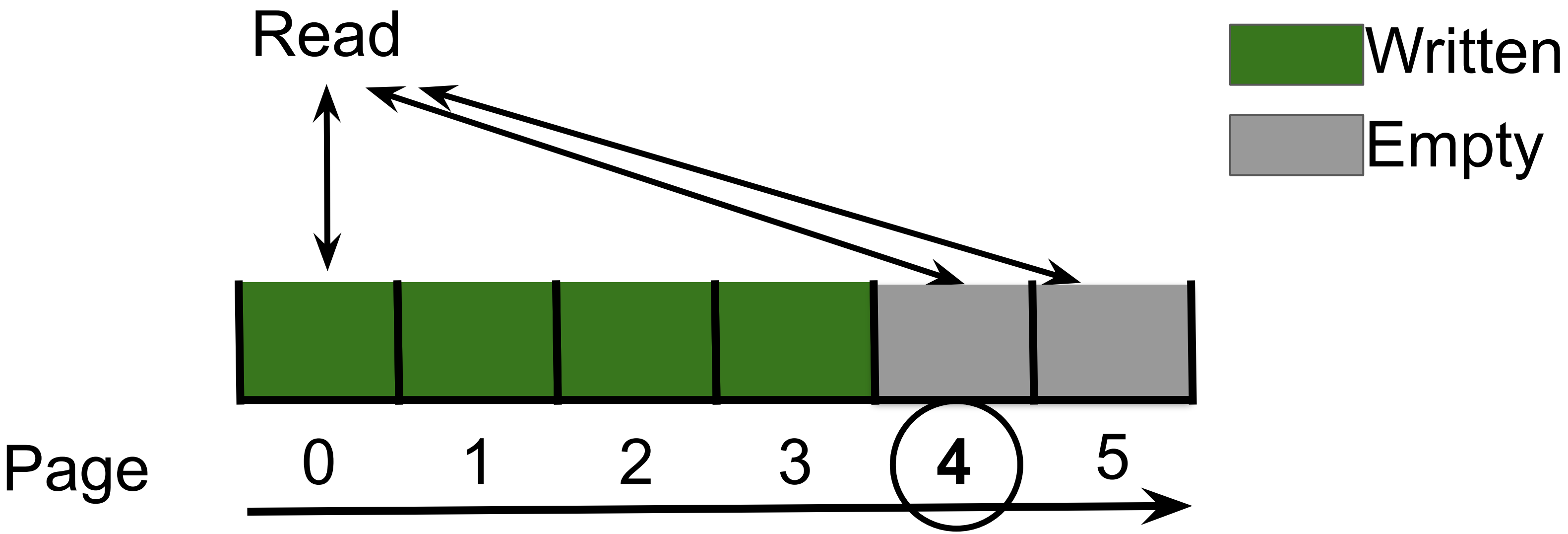
Block interface: Random write/read



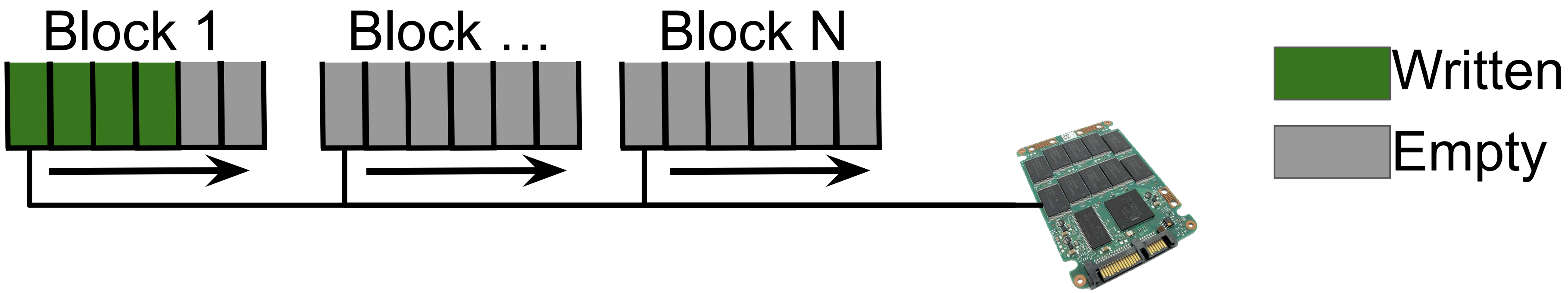
Flash is **different**: Sequential writes **only**



Flash is **different**: Random reads

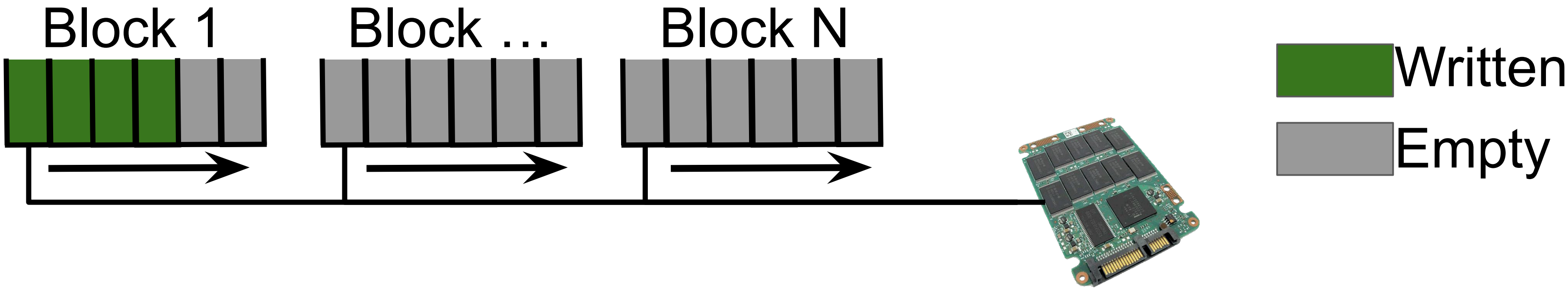


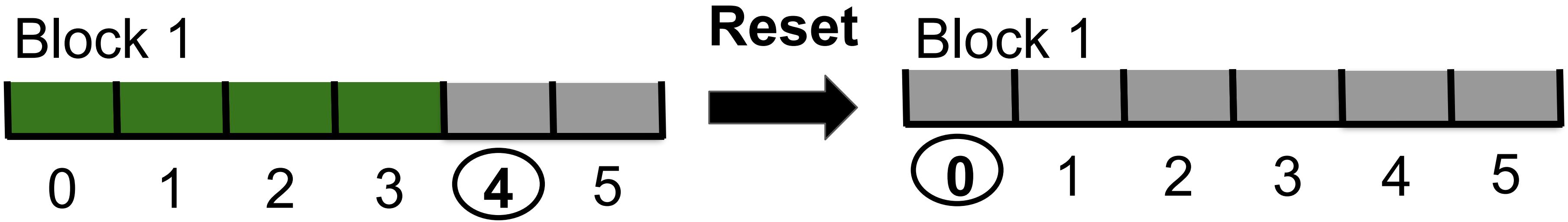
Flash is **different**: Reset at block level



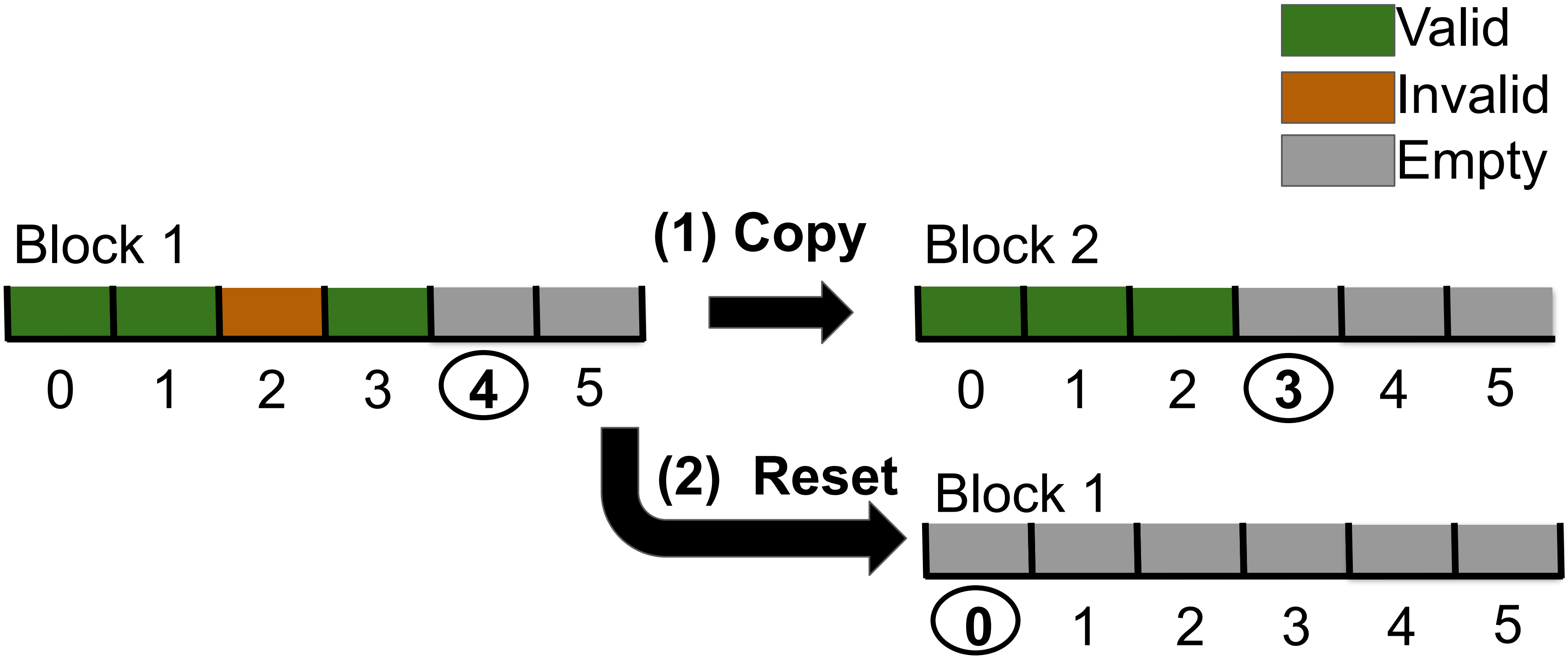
VU

Flash is **different**: Reset at block level





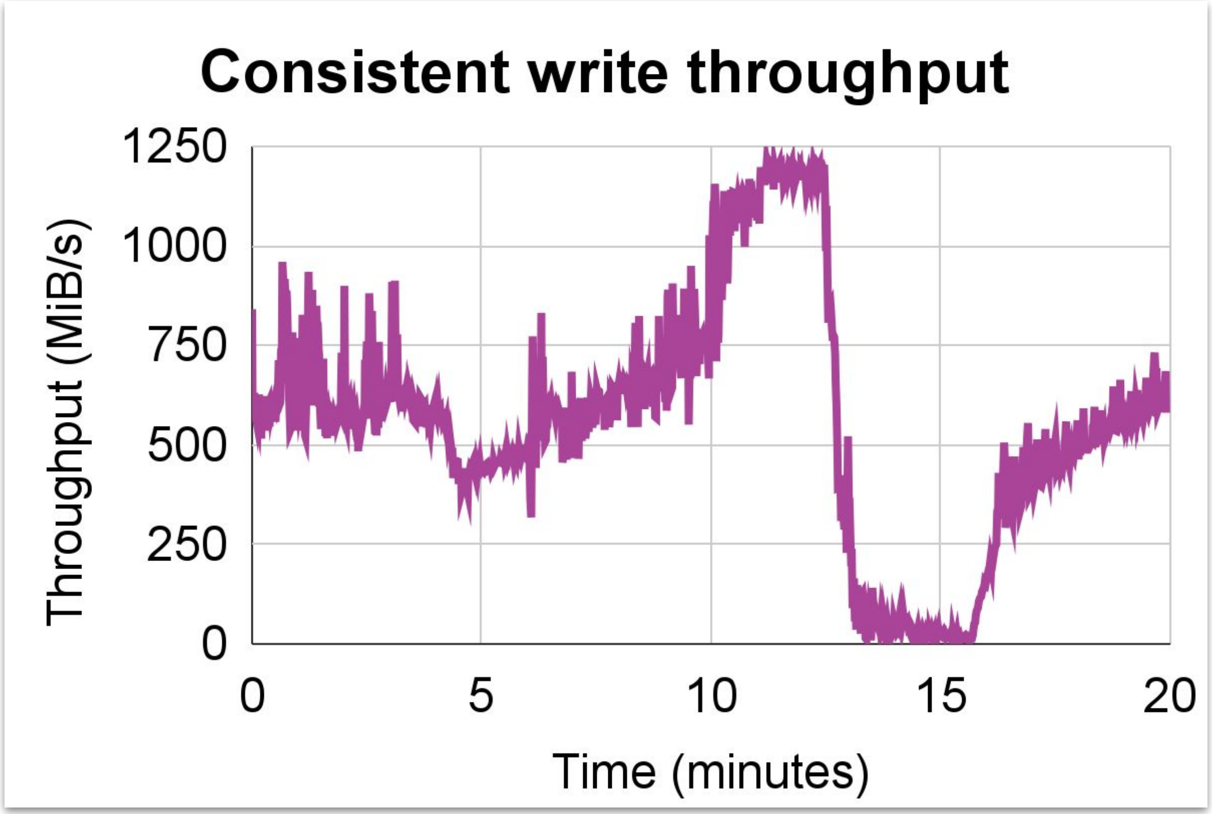
Block interface: Need to migrate data



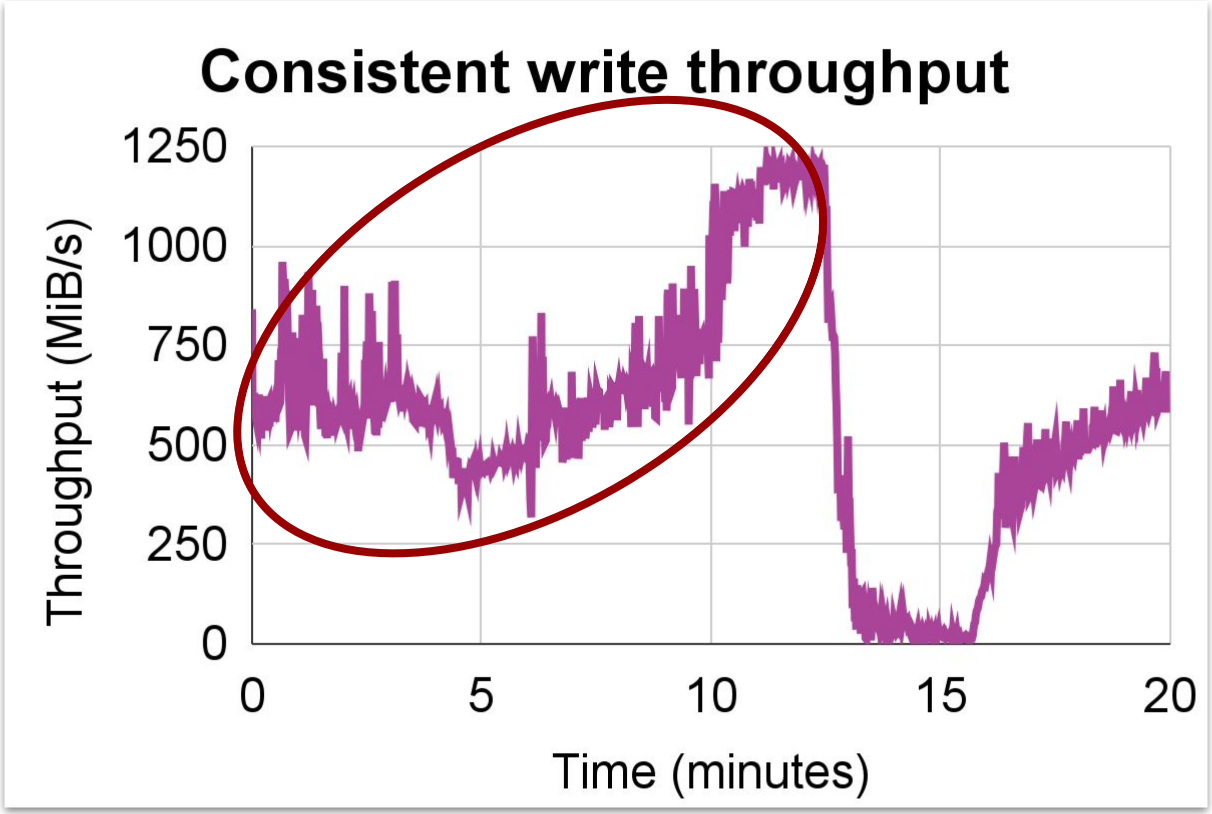
Flash: The problem of the block interface

Question?: What happens if we keep writing to the SSD? What about data migration?

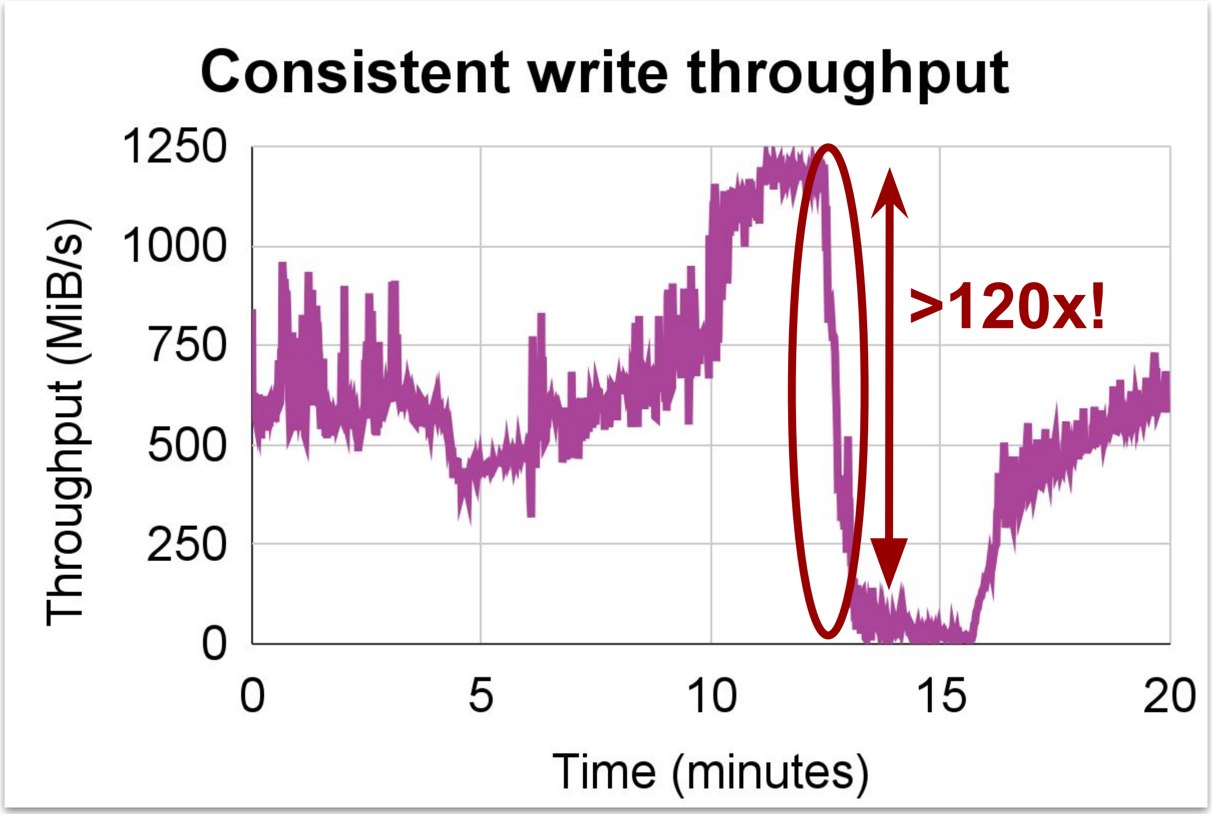
Flash: The problem of the block interface



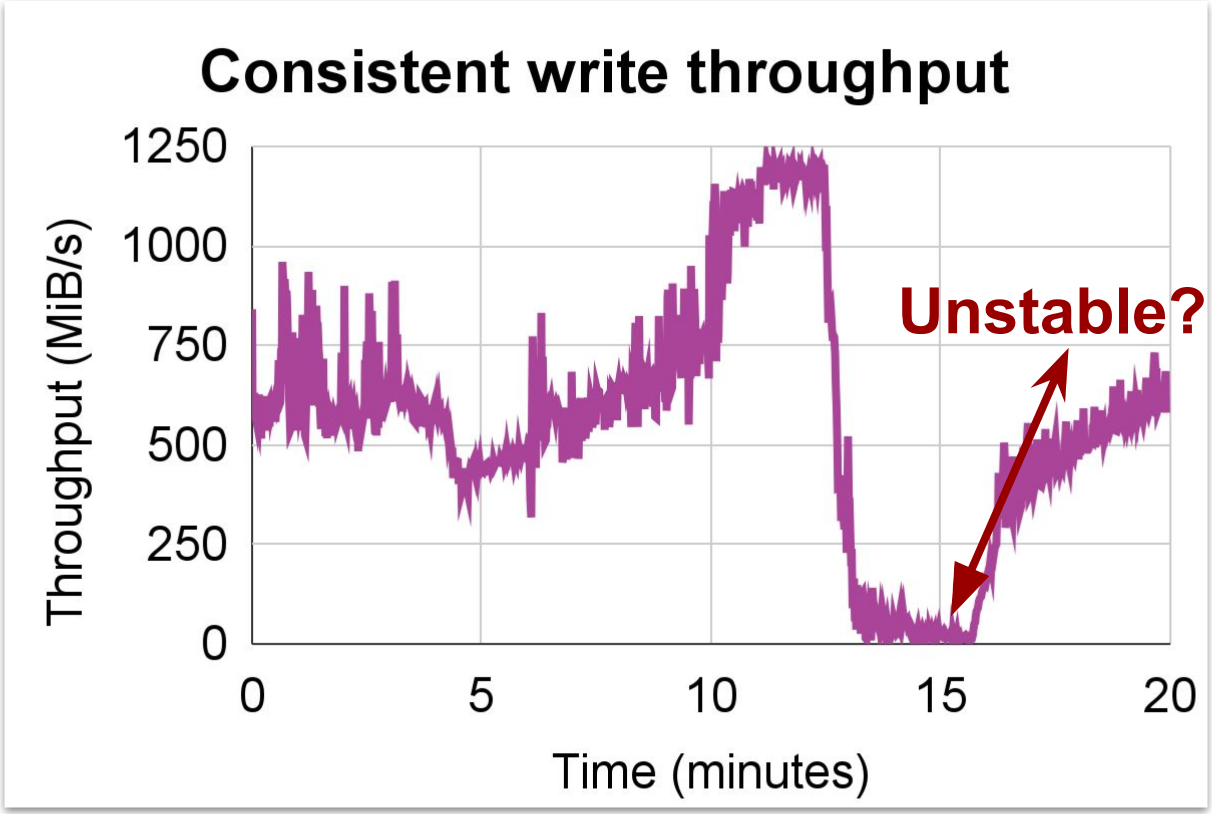
Flash: The problem of the block interface



Flash: The problem of the block interface

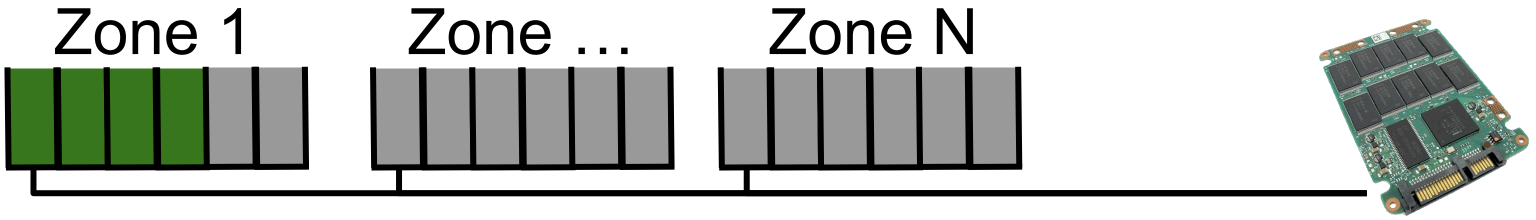


Flash: The problem of the block interface

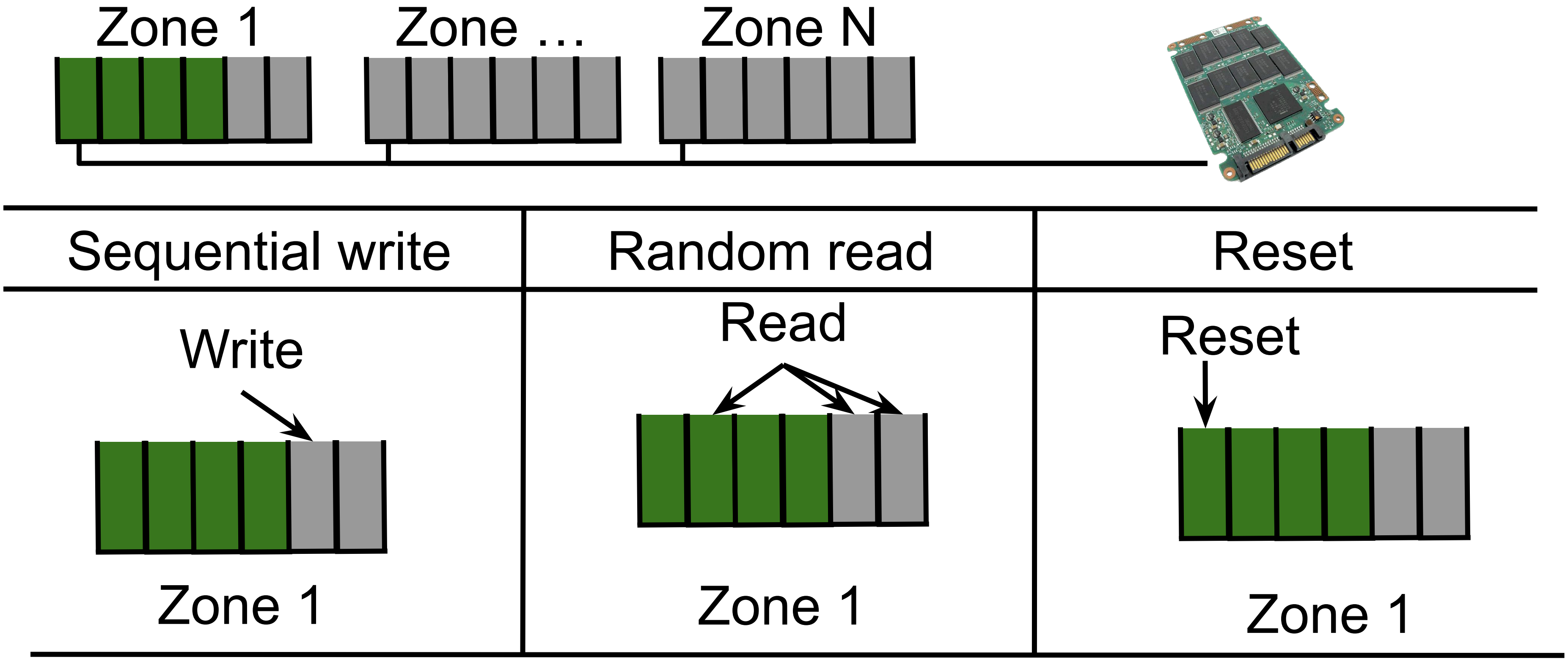


Is there a different interface
For flash storage?

Meet Zoned Namespace SSDs

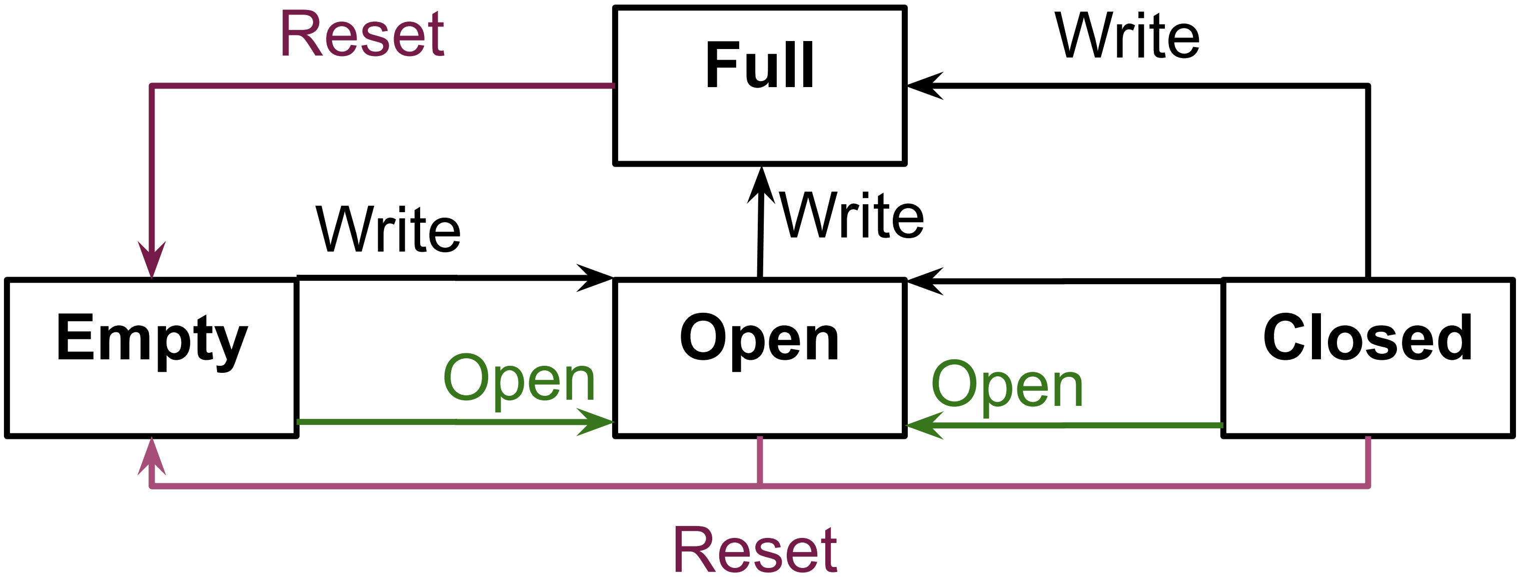


Meet Zoned Namespace SSDs



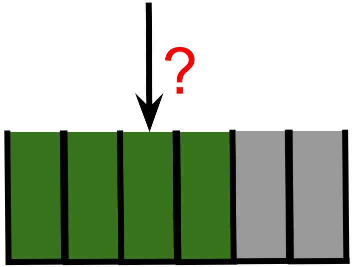
Zone state complexity

Possible zone states:



Research problem

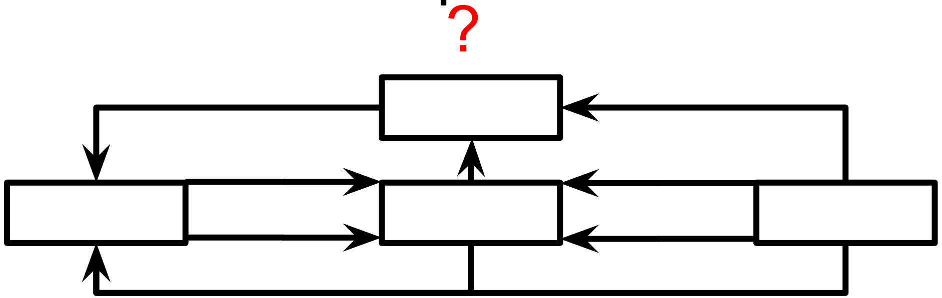
Write/Read/Reset



Stable?

?

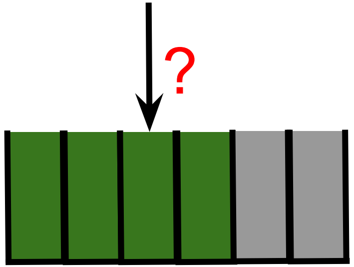
Zone operations



Research problem

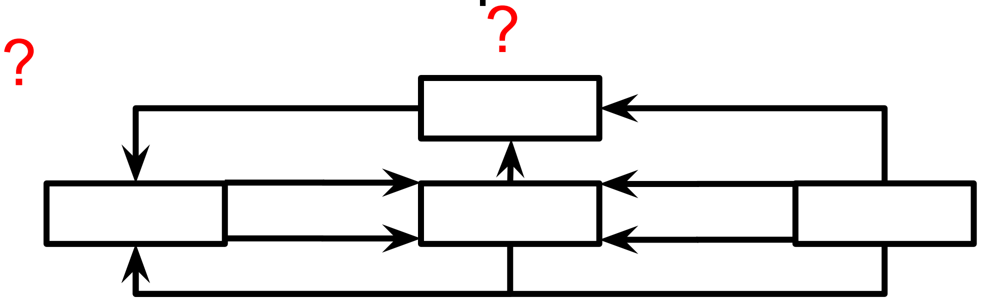
Question: What are ZNS performance characteristics?

Write/Read/Reset



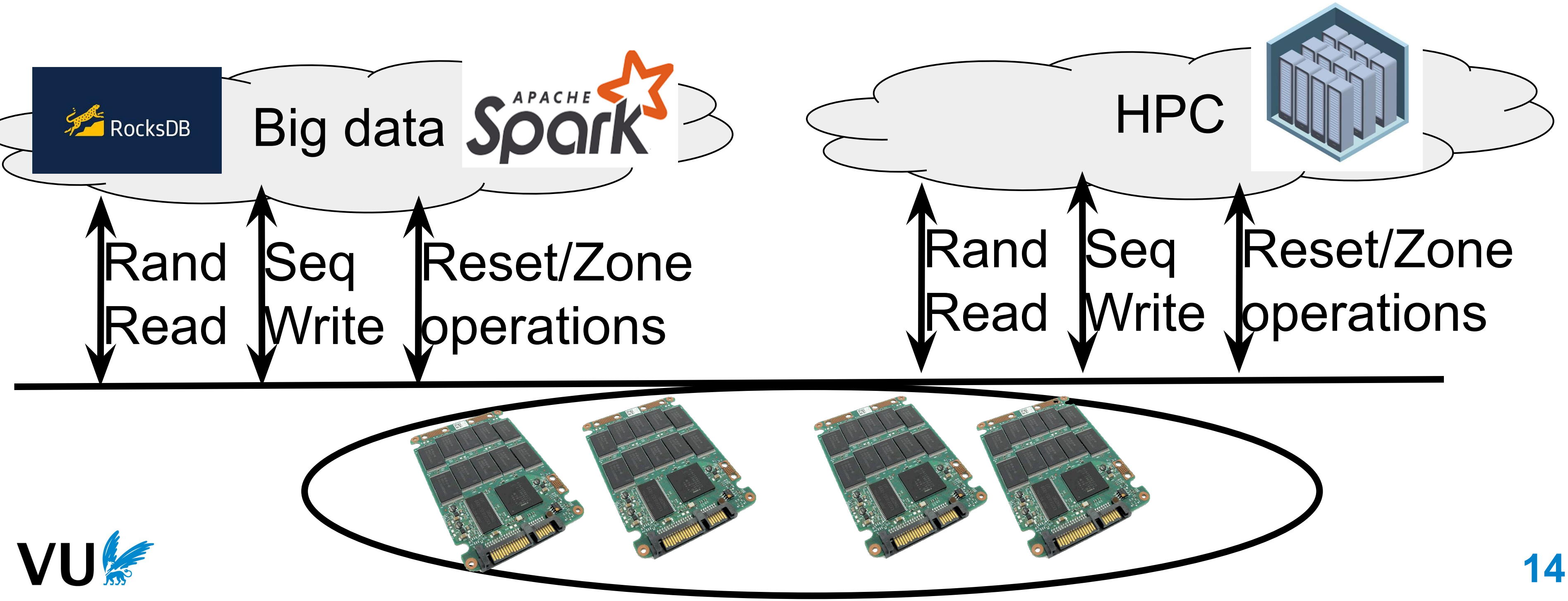
Stable?

Zone operations



Research problem

Before we use ZNS we **need** to characterize its performance characteristics



Solution: A characterization framework

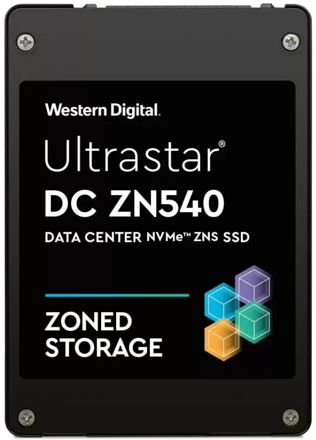
Goal: Infer performance characteristics

Microbenchmarks and scalability tests of:

- Appends/writes/reads
- **All** zone management operations

Interference tests of:

- Writes and reads
- Resets and writes/reads



Solution: A characterization framework

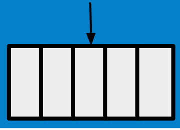
Results:
- 13 Key observations
- 5 Recommendations

We discuss the **4 biggest** observations

https://github.com/stonet-research/NVMeBenchmarks

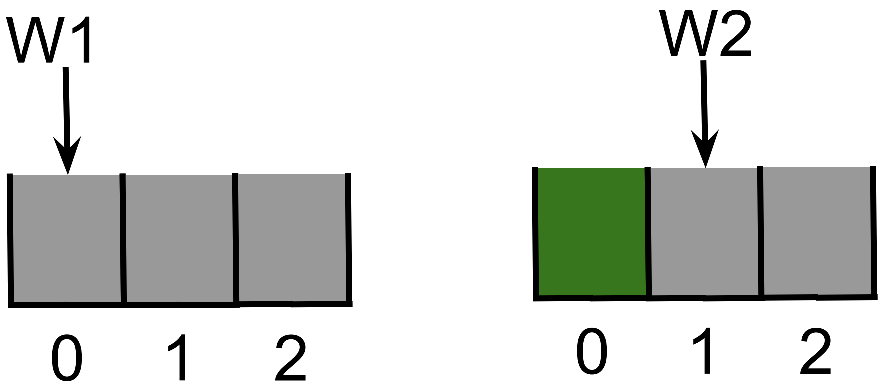
VU

Write operation methods

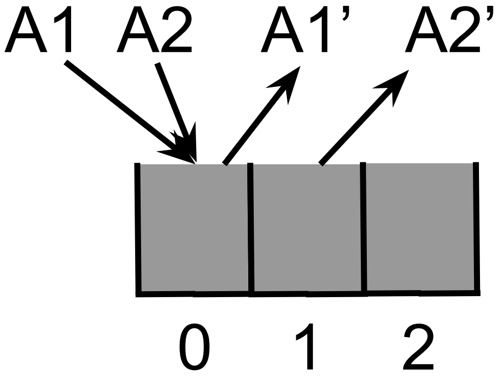


R1/R4

Sequential write	Appends



Host schedules one-by-one



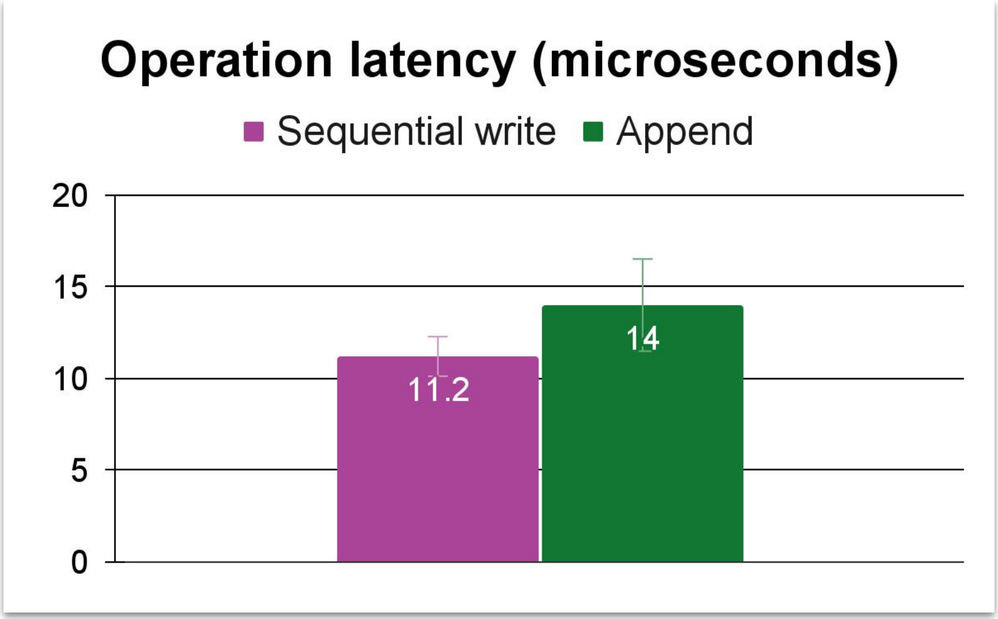
Device schedules and returns address

Latency: writes versus appends R1/R4

Motivation: What operation to use for low latency applications?

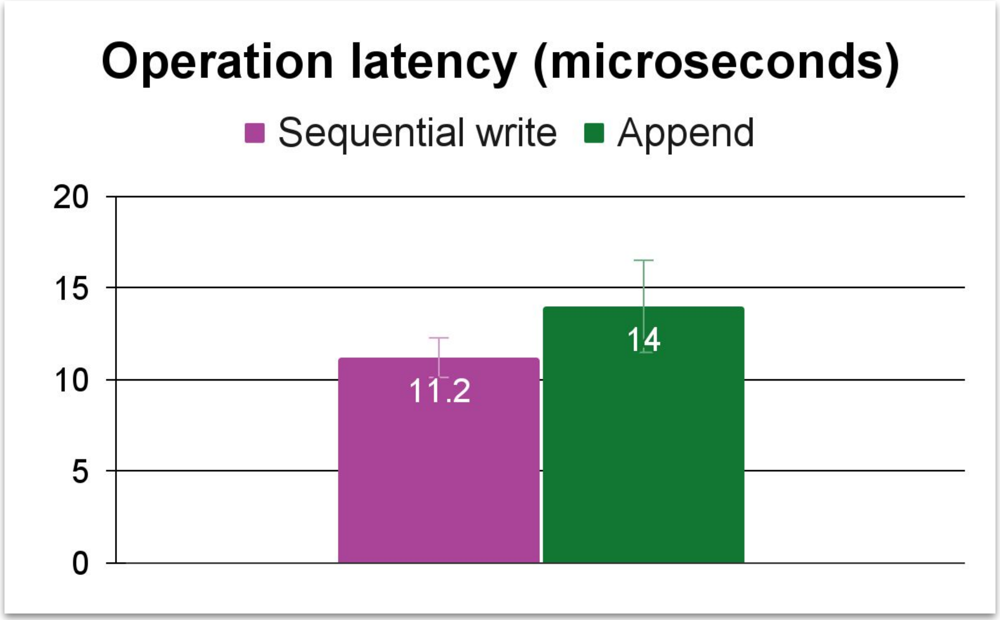
Latency: writes versus appends R1/R4

Motivation: What operation to use for low latency applications?



Latency: writes versus appends R1/R4

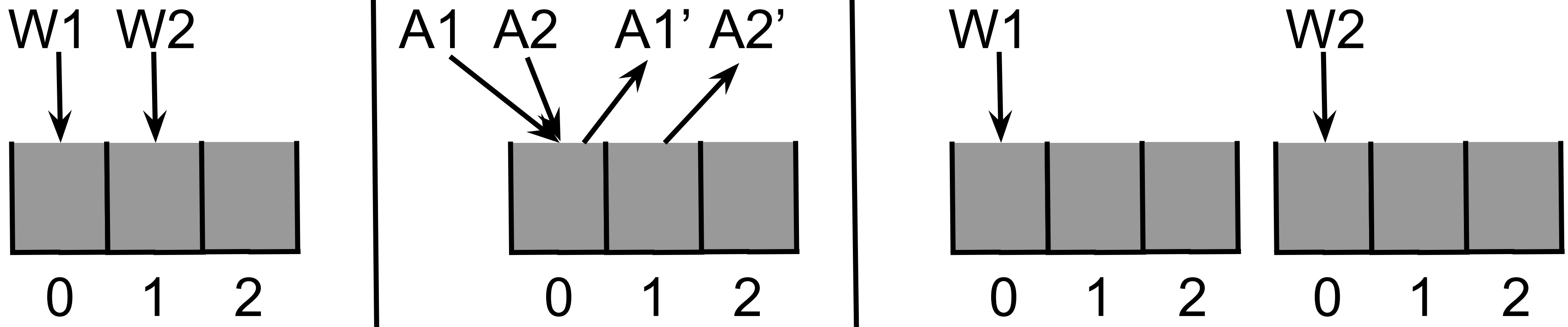
Motivation: What operation to use for low latency applications?

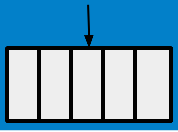


Recommendation: Use writes for lower latency

ZNS write parallelism methods R2/R4

Sequential writes	Zone appends	Write to multiple zones

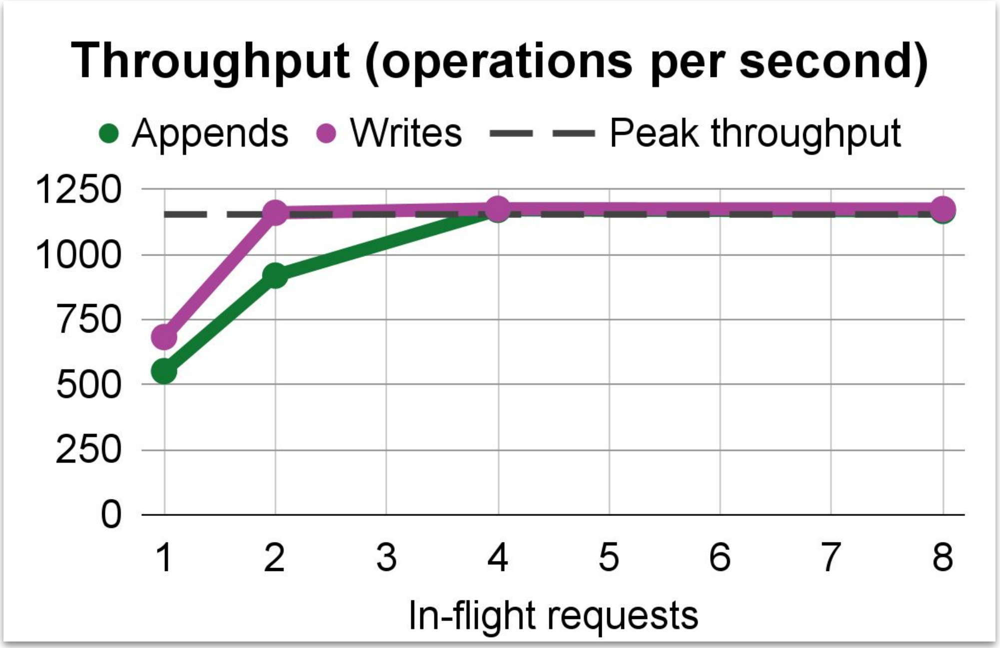


Throughput: ZNS writes  R2/R4

Motivation: What write operation to use for scaling throughput?

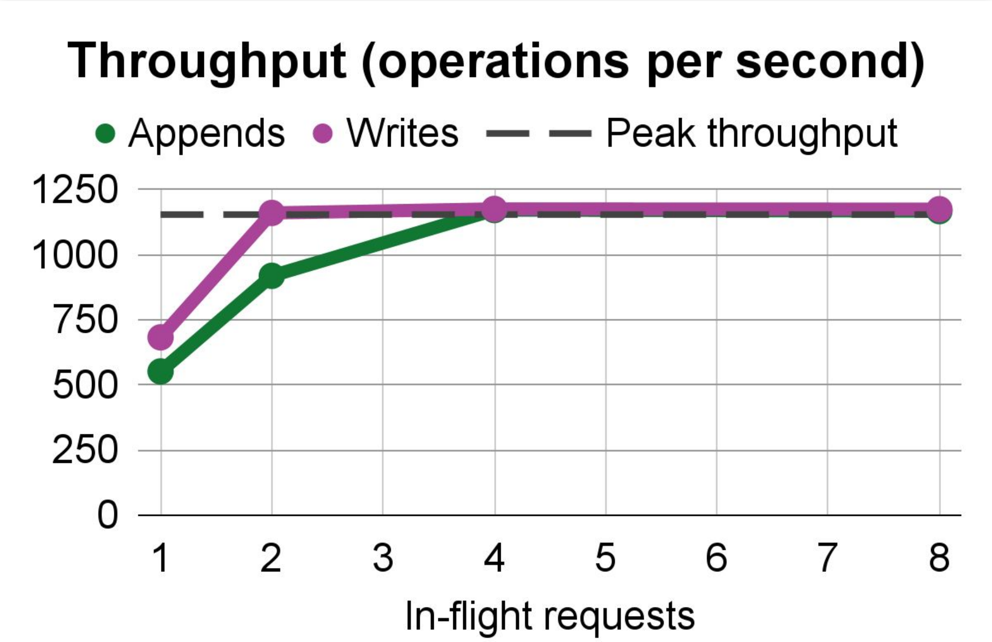
Throughput: ZNS writes R2/R4

Motivation: What write operation to use for scaling throughput?



Throughput: ZNS writes R2/R4

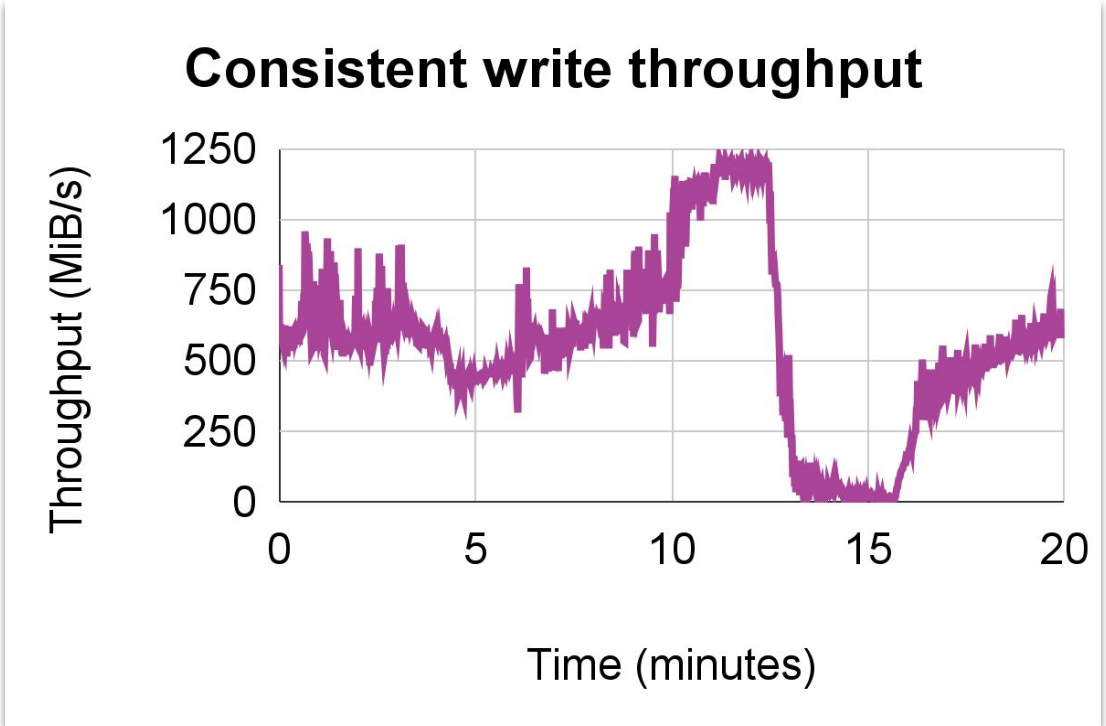
Motivation: What write operation to use for scaling throughput?



Recommendation: Use appends to scale throughput because of the open zone limit

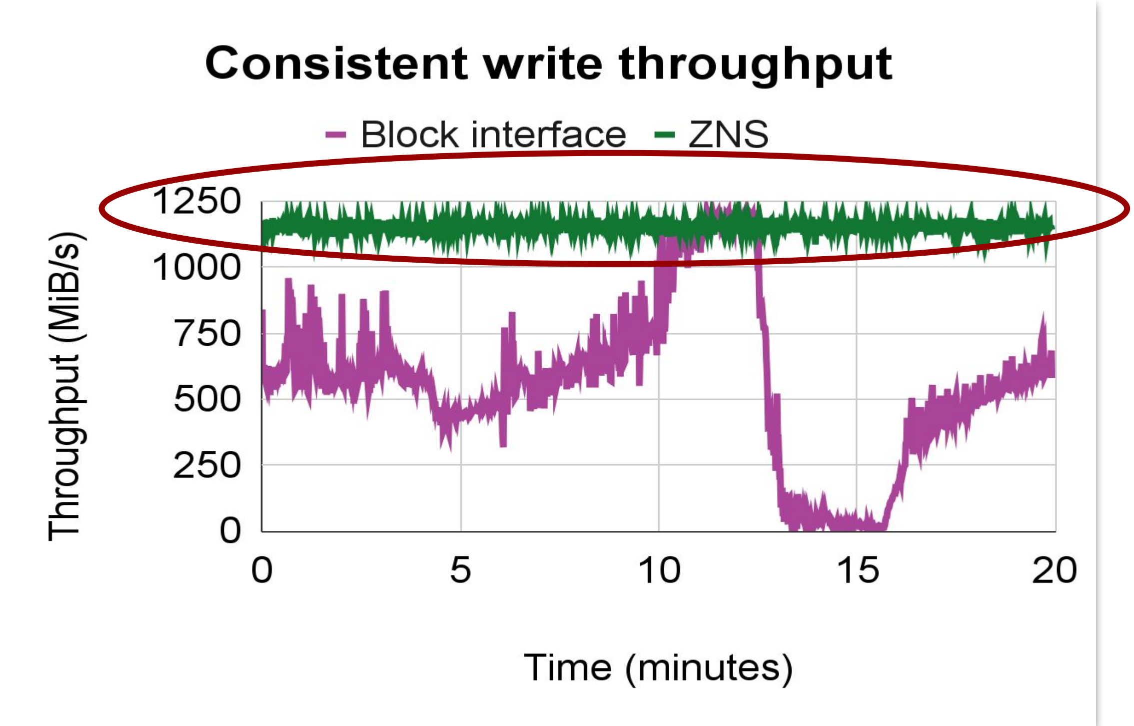
Predictable performance? Stable? R3/R4

Question: Does ZNS lead to the same variability as block devices?



Predictable performance? Stable? R3/R4

Question: Does ZNS lead to the same variability as block devices? **It does not**



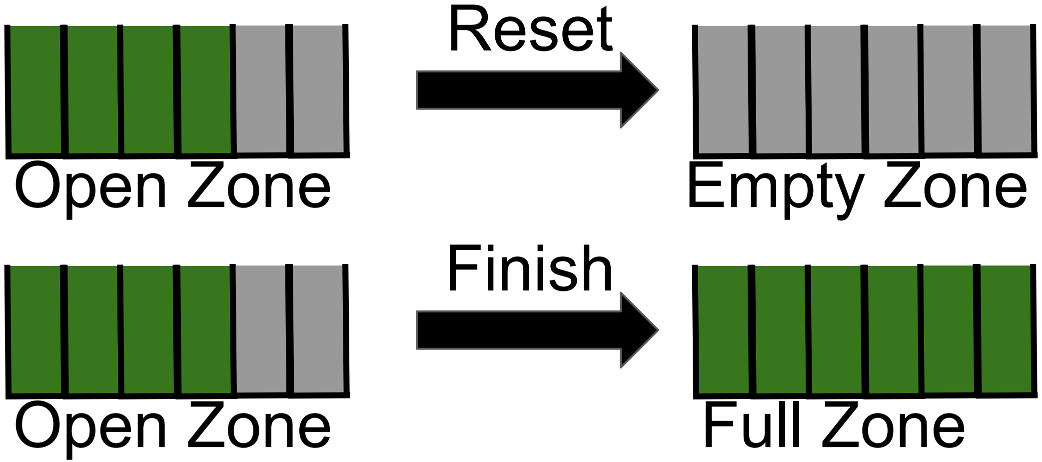
VU

Reset operation latency

Reset comes in two flavors:

(1) Pure Reset:

(2) Finish:



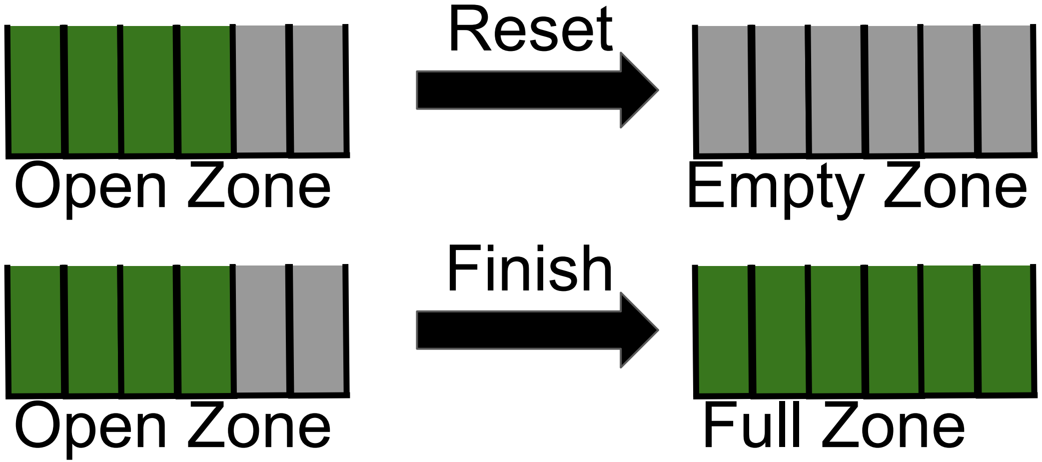
VU

Reset operation latency

Reset comes in two flavors:

(1) Pure Reset:

(2) Finish:



Finish is used to prevent reaching the **open zone limit**

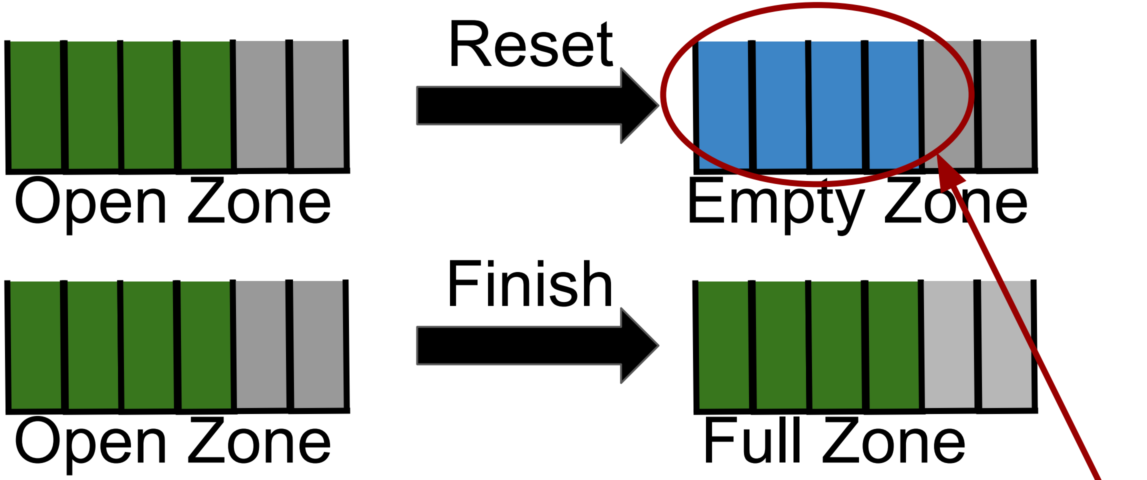
Reset operation latency

R4/R4

Reset comes in two flavors:

1) Pure Reset:

2) Finish:



Important question: does occupancy (% filled) matter?

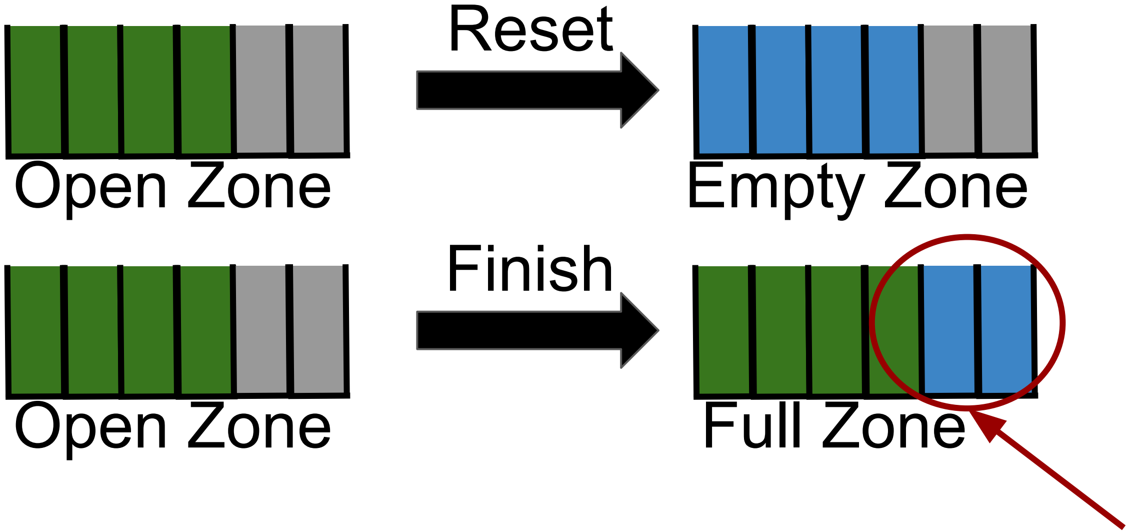
Reset operation latency

R4/R4

Reset comes in two flavors:

1) Pure Reset:

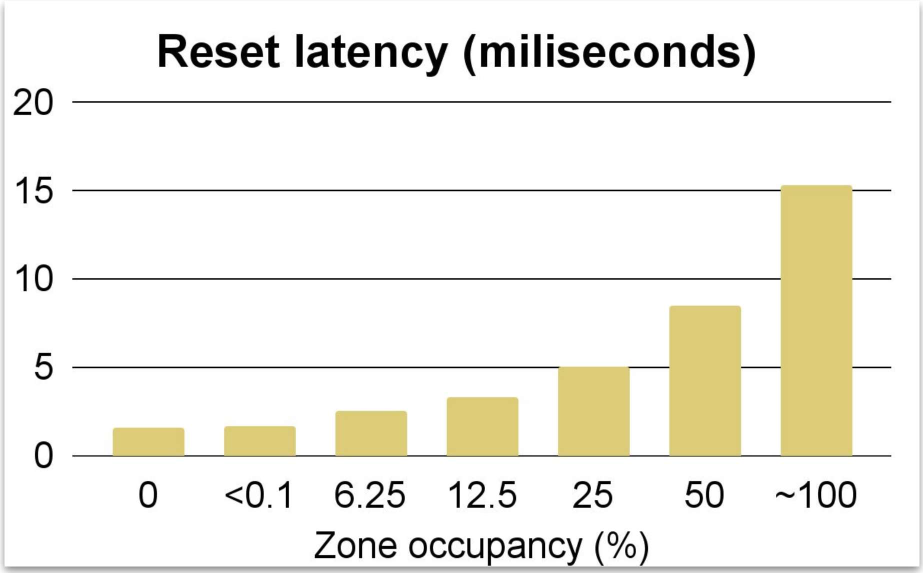
2) Finish:



Important question: does occupancy (% filled) matter?

Reset operation latency

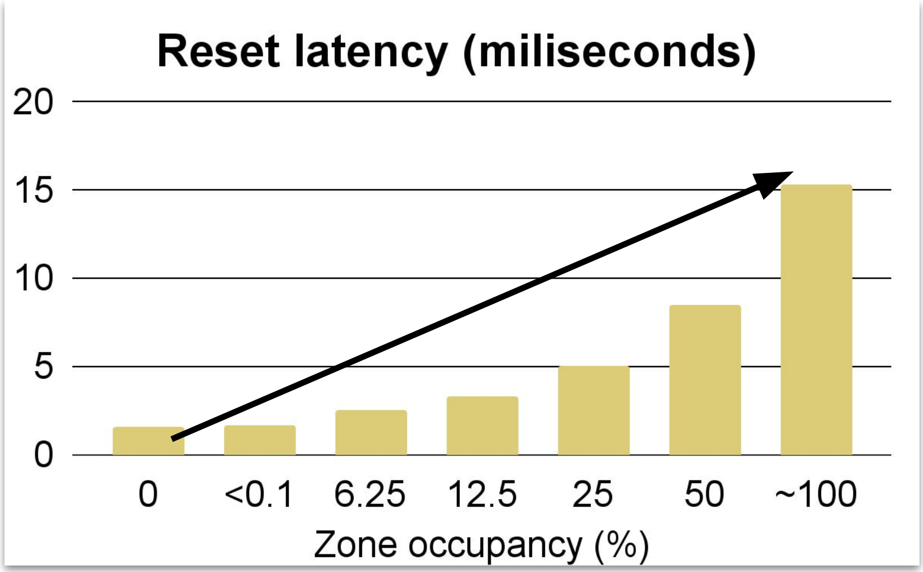
Motivation: Applications have to issue resets

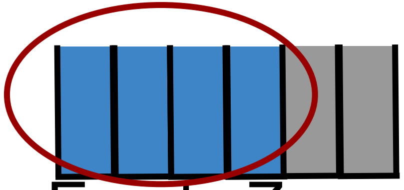


Reset operation latency

R4/R4

Motivation: Applications have to issue resets





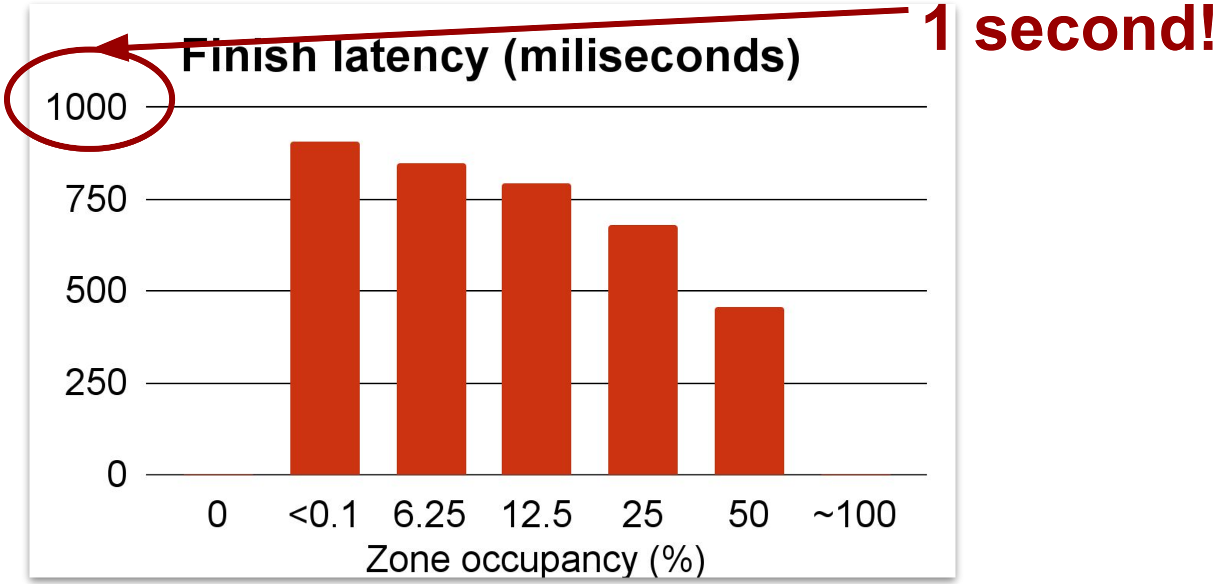
Empty Zone

Occupancy matters!

Reset operation latency

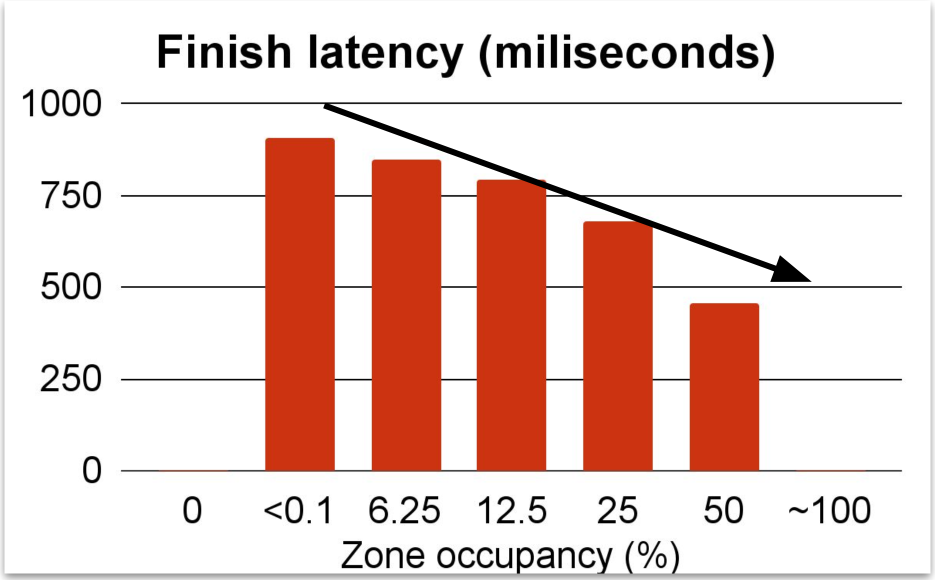
R4/R4

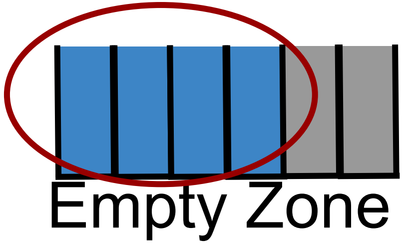
Motivation: Applications have to issue resets



Reset operation latency

Motivation: Applications have to issue resets



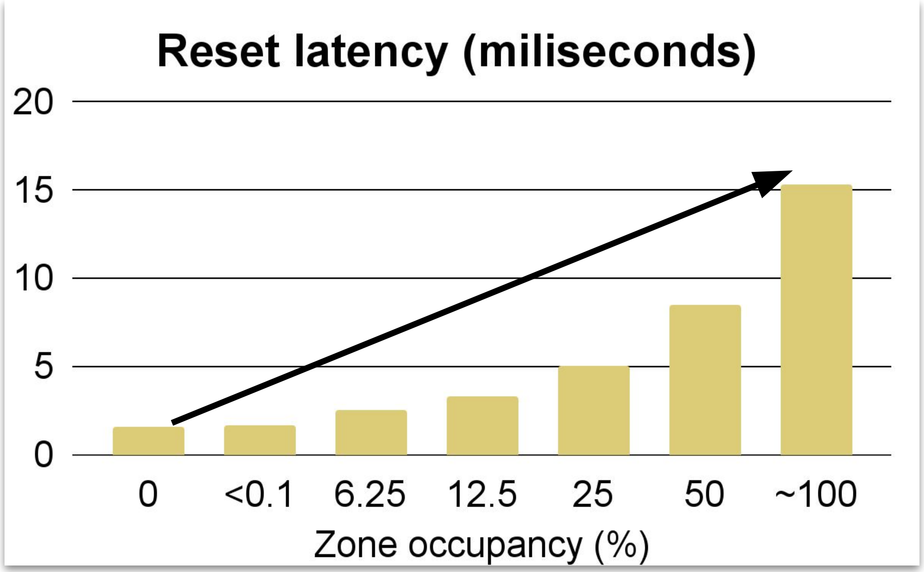


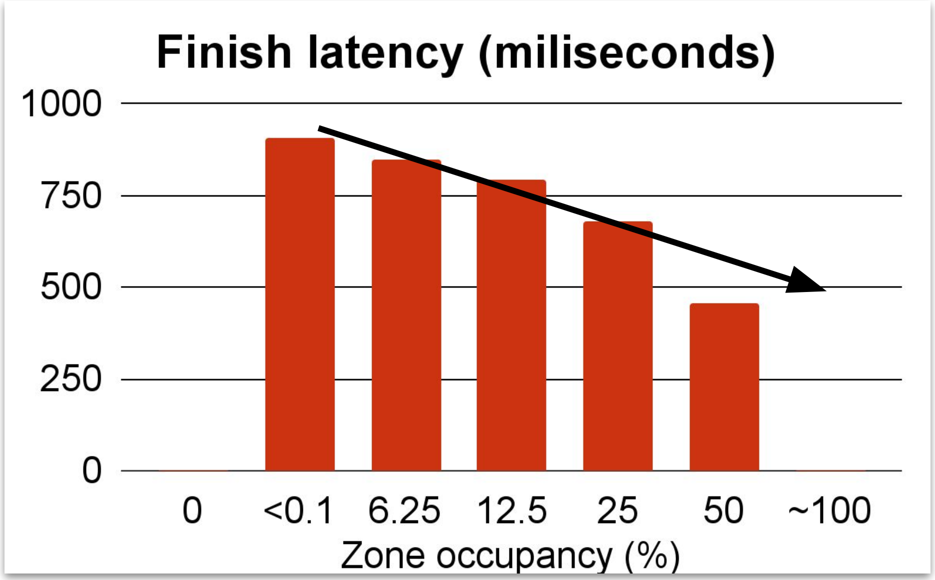
Occupancy matters!

Reset operation latency

R4/R4

Motivation: Applications have to issue resets

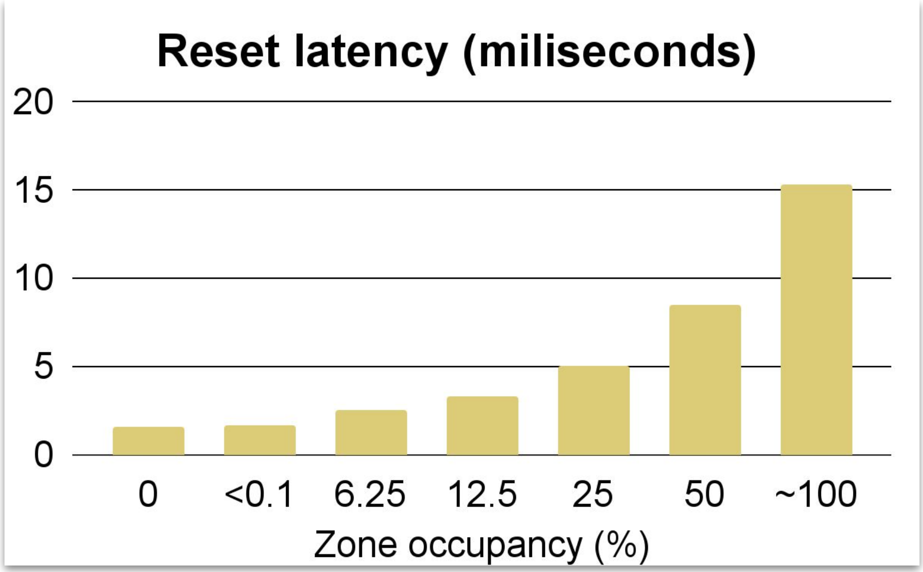


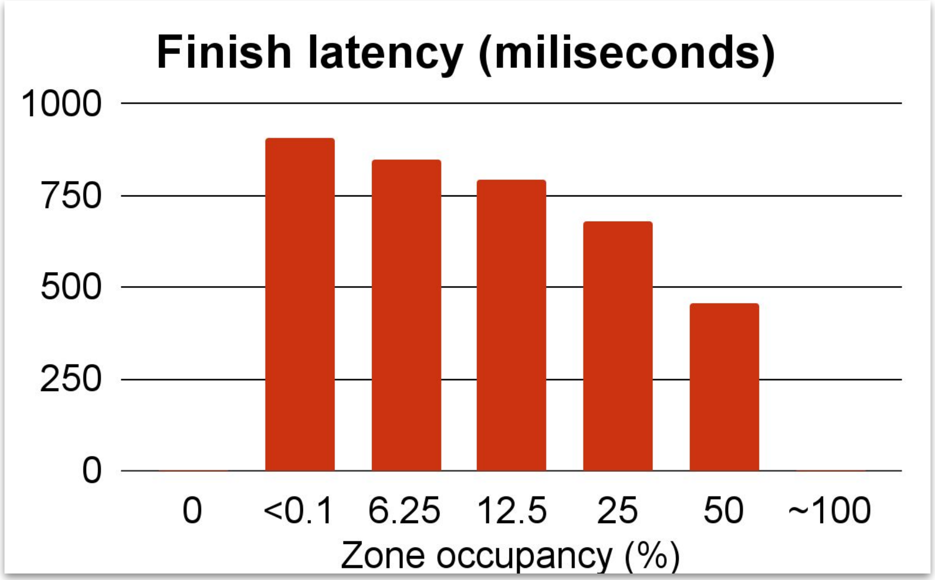


Reset operation latency

R4/R4

Motivation: Applications have to issue resets





Recommendation: Prevent issuing resets

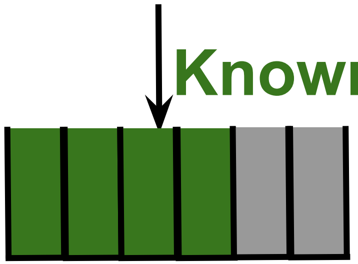
VU

Conclusion of discussed results

1. Low latency: use sequential writes
2. High throughput: use appends
3. ZNS has latency stability
4. ZNS reset operations should be avoided

Write/Read/Reset

Known!

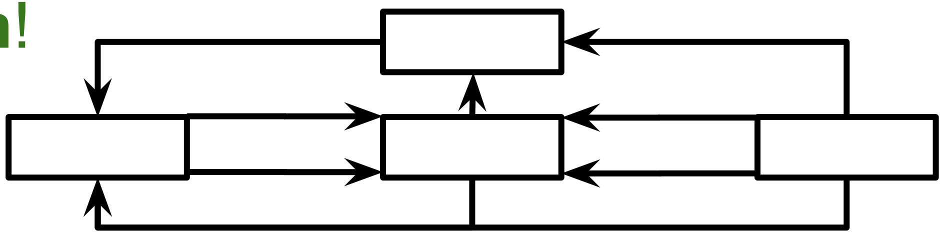


Stable?

Known!

Zone operations

Known!



VU

More details/results in the paper …

Performance Characterization of NVMe Flash Devices with Zoned Namespaces (ZNS)

Krijn Doekemeijer*[1], Nick Tehrany*[1,2], Balakrishnan Chandrasekaran[1], Matias Bjørling[3], and Animesh Trivedi[1]
[1]Vrije Universiteit Amsterdam, Amsterdam, the Netherlands
[2]Delft University of Technology, Delft, the Netherlands
[3]Western Digital, Copenhagen, Denmark
{k.doekemeijer, n.a.tehrany, b.chandrasekaran, a.trivedi}@vu.nl, matias.bjorling@wdc.com

***Abstract*—The recent emergence of NVMe flash devices with Zoned Namespace support, ZNS SSDs, represents a significant new advancement in flash storage. ZNS SSDs introduce a new storage abstraction of append-only *zones* with a set of new I/O (i.e., append) and management (zone state machine transition) commands. With the new abstraction and commands, ZNS SSDs offer more control to the host software stack than a non-zoned SSD for flash management, which is known to be complex (because of garbage collection, scheduling, block allocation, parallelism management, overprovisioning). ZNS SSDs are, consequently, gaining adoption in a variety of applications (e.g., file systems, key-value stores, and databases), particularly latency-sensitive big data applications. Despite this enthusiasm, there has yet to be a systematic characterization of ZNS SSD performance with its zoned storage model abstractions and I/O operations. This work addresses this crucial shortcoming. We report on the performance features of a commercially available ZNS SSD (13 key observations), explain how these features can be incorporated into publicly available state-of-the-art ZNS emulators, and recommend guidelines for ZNS SSD application developers. All artifacts (code and data sets) of this study are publicly available at https://github.com/stonet-research/NVMeBenchmarks.**

***Index Terms*—Measurements, NVMe storage, Zoned Namespace Devices**

I. Introduction

The emergence of fast flash storage in data centers, HPC, and commodity computing has fundamentally caused changes in every layer of the storage stack, and led to a series of new developments such as a new host interface (NVM Express, NVMe) [1], [2], [3], a high-performance block layer [4], [5], [6], [7], new storage I/O abstractions [8], [9], [10], [11], [12], [13], [14], and re/co-design of storage application stacks [15], [16], [17], [18], [19], [20], [21]. Today, flash-based solid-state drives (SSDs) can support very low latencies (i.e., a few microseconds), and multi GiB/s bandwidth with millions of I/O operations per second [22], [23], [24].

Despite these advancements, the conceptual model of a storage device remains unchanged since the introduction of hard disk drives (HDDs) more than half a century ago. A storage device supports only two necessary operations: write and read data in units of *sectors* (or blocks) [25]. Data can be read from and written to anywhere on the device, hence supporting random and sequential I/O operations. Though this model works with conventional HDDs, it is not apt for flash-based storage devices as flash internally does not support overwriting data [26], [27], [28]. Flash devices offer the illusion of "overwritable" storage via the *flash translation layer (FTL)*, a software component that runs within the device. The FTL enables easy integration of flash devices (by allowing them to masquerade as fast HDDs), albeit it introduces unpredictability in performance [29], [30], [31], [32], [33], [34] and complicates device lifetime management [35]. These challenges are defined as the *unwritten contracts* of SSDs [26]. As data centers have largely transitioned to SSDs for fast, reliable storage [36], [37], and modern big data applications have high QoS demands [38], [39], there is a dire need to address these unwritten contracts.

Researchers and practitioners advocate for open flash SSD interfaces beyond block I/O [40] to address these challenges. Examples include *Open-Channel SSDs* (OCSSD) [41], *multi-stream SSDs* [9], and, more recently, *Zoned Namespaces* (ZNS) [11]. The focus of this work is on NVMe devices that support ZNS, which are commercially available today [42], [43]. ZNS promises a low and stable tail latency [11] and a high device longevity, and, hence, addresses the needs of modern big data workloads. There is, unsurprisingly, a rich body of active and recent work on ZNS [44], [11], [45], [46], [47], [48], [49], [50], [51], [52], [53], [54], [55], [56]. Despite this enthusiasm, there has *not* been a systematic performance and operational characterization of ZNS SSDs. This lack of an extensive performance and operational characterization of ZNS SSDs severely limits the utilization and application of ZNS devices in big data workloads. In this work, we bridge this gap by presenting the performance characterization of a commercially-available NVMe ZNS device.

We complement this characterization of a physical device with an investigation of emulated ZNS devices, since they are widely used in research [51], [57], [58], [55]. Emulated devices enable researchers to explore the ZNS design space without being constrained by device-specific characteristics. Such unconstrained explorations are crucial since ZNS is a new interface and the selection of available configurations in a real SSD is, unsurprisingly, quite limited. The research validity of all of these works hinge on an emulator's ability to mimic

*Equal contributions, joint first authors. Nick was with TU Delft during this work.

Zones of a ZNS device have states (Fig. 1), which dictate the allowed operations on a specific zone. Since each zone operation (e.g., `read`, `write`, and `append`) consumes SSD resources (e.g., internal buffers), there are limits on the number of zones that can be concurrently opened and used. These limits are defined as the *maximum open zone limit* and *maximum active zone limit*, respectively. Applications must abide by these constraints, and explicitly manage the zone states and transitions. An application must, for instance, open a zone *before* it accepts `write`s or `append`s. State transitions can be internal to a device and *implicit* (e.g., a `write` to an empty zone transitions it to an open zone in Fig. 1), or *explicit* as a response to a user request.

ZNS offers several explicit zone management operations, which include `open`, `close` and `finish`. We skip discussing the first two, whose names reveal their functionalities, and focus on the last. The `finish` operation transforms an *open* zone directly into a *full* zone. It releases all resources attached to the zone (to stay within the maximum open zone limit). Then, the device can either fill the zone with data or mark the unused capacity with mapping updates (metadata updates) in the "finished" zone (Fig. 1). Mapping updates would require extra metadata to keep track of partially-filled zones. The `finish` operation has implications for performance, and the costs of this operation varies from one ZNS SSD implementation to another.

In summary, ZNS devices support a rich I/O interface that includes operations beyond the simple `read` and `write` operations seen in traditional flash storage. It is, therefore, crucial to understand and characterize the performance of these operations as they (and their state-machine transitions) are now part of the Linux storage software stack.

C. Software support

ZNS devices are fully supported in Linux since kernel version `5.9` [75]. Currently there is a limited number of applications that use ZNS and most that do, do not use all functionalities (e.g., no `finish` or `open`). Evaluating these applications would limit what ZNS properties we can measure and, therefore, in our work we use synthetic benchmarks as we need to understand all of ZNS' facets first. The results of our benchmarks can then be used for application design. Here, we briefly mention a number of prominent ZNS applications in research to get an overview of what is currently available. Currently, applications have access to ZNS-friendly file systems F2FS [76], Btrfs [77] and Ceph [78]. There is also support for a swap system known as ZNSwap [49] and a RAID system known as ZRAID [79] Lastly, KV-store RocksDB has ZenFS as a ZNS-capable file system back-end [11].

III. Experiments

In this paper, we characterize the performance and interference properties of the Western Digital Ultrastar DC ZN540 SSD, a large-zone ZNS SSD, using a series of controlled benchmarks. As of writing the number of commercially-available ZNS SSDs is limited, therefore, we focus our efforts on characterizing one SSD model and synthesize the performance questions to ask when evaluating a ZNS SSD. Tab. I summarizes our key findings.

TABLE I: *Overview of the key insights*

Category	*Insight*
`Append` vs. `write`	`Write` operations have up to 23% lower latencies than `append` operations (§III-C)
Scalability	Prefer intra-zone scalability (§III-D, §III-E)
Zone transitions	`Finish` is the most expensive operation; it takes up to hundreds of milliseconds (§III-E)
I/O interference	NVMe ZNS offers 3× higher read throughput under concurrent I/O operations than NVMe (§III-F)
I/O & GC interference	`Reset` latency increases by up to 78% under concurrent I/O operations, but `reset` operations themselves have no effect on `append`, `read` or `write` operations (§III-G)

A. Benchmarking setup

We use *fio* [80] for generating the workloads and benchmarking the ZNS device. We also employ custom *SPDK* benchmarks for benchmarking state transitions (§III-E) and `reset` interference (§III-G), since *fio* does not support them. We describe our benchmarking platform in detail in Tab. II.

We use two storage stacks for benchmarking: the Linux kernel block layer and the *SPDK* stack. The Linux block layer ships with the *mq-deadline* scheduler, which buffers multiple `write` operations to a single zone, merges writes to contiguous LBAs into one or multiple (larger) writes, and sequentially issues the merged requests. Applications can, hence, issue multiple `write` operations to a single zone. The *SPDK* stack, in contrast, is a bare-bones storage stack without any I/O scheduler. The rationale behind our storage stack selection is twofold. First, no storage stack currently supports all combinations of I/O and management operations that we aim to benchmark. We cannot, for instance, issue and benchmark `append` or zone management operations via *fio* and the Linux I/O stack. In a similar vein, we are restricted to issuing only one `write` per zone at a time with *SPDK*, since it lacks an I/O scheduler. Second, the selection enables us to compare the implications of state-of-the-practice—the Linux stack—and that of the state-of-the-art—*SPDK*—for ZNS application development.

We run experiments for 20 minutes and/or repeat them at least three times for deriving robust statistics. We pin our benchmarking code to the NUMA node containing the ZNS device. For the Linux storage stack, we use the *io_uring* engine with submission-queue polling enabled, following the recommended settings [14].

B. Performance metrics

We briefly describe the metrics we use to evaluate the performance of NVMe (ZNS) devices. Two indicators of I/O operation performance are throughput (i.e., the number of operations or bytes per second) and operation latency (i.e, the time each operation takes). We measure ZNS throughput in

VU

Take-home messages

ZNS SSDs deliver high-throughput stable performance

- ZNS has a unique performance model
- We synthesize ZNS' performance model

https://github.com/stonet-research/NVMeBenchmarks

Further reading

1. Bjørling et al., ZNS: Avoiding the block interface tax for flash-based SSDs, ATC'21
2. Shin et al., Exploring Performance Characteristics of ZNS SSDs: Observation and Implication, NVMSA'20
3. Bae et al., What You Can't Forget: Exploiting Parallelism for Zoned Namespaces, HotStorage'22
4. Im et al., Accelerating RocksDB for SmallZone ZNS SSDs by Parallel I/O Mechanism, Middleware'22
5. Jung et al., Preemptive Zone Reset Design within Zoned Namespace SSD Firmware, MDPI'23
6. Purandare et al. Append is near: Log-based data management on ZNS SSDs, CIDR'22
7. Bjørling et al., Zone Append: A New Way of Writing to Zoned Storage, Vault'20
8. **Tehrany et al.** Understanding (Un)Written Contracts of NVMe ZNS Devices with zns-tools, ArXiv (2023)
9. **Tehrany et al.** Understanding NVMe zoned namespace (ZNS) flash SSD storage devices**,** ArXiv (2023)
10. **Doekemeijer et al.** TropoDB: Design, Implementation and Evaluation of an Optimised KV-Store for NVMe Zoned Namespace Devices, MSc thesis (2022)
11. Kim et al., Performance Modeling and Practical Use Cases for Black-Box SSDs, TOS'21
12. He et al., The Unwritten Contract of Solid State Drives, EuroSys'17
13. Chen et al, Understanding Intrinsic Characteristics and System Implications of Flash Memory Based Solid State Drives, SIGMETRICS'09
14. Bjørling et al., LightNVM: The Linux ´ Open-Channel SSD Subsystem, FAST'17
15. Kang et al., The Multi-streamed Solid-State Drive, HotStorage'14

Also check out other research from the AtLarge research team!: https://atlarge-research.com/publications.html

- Backup slides -

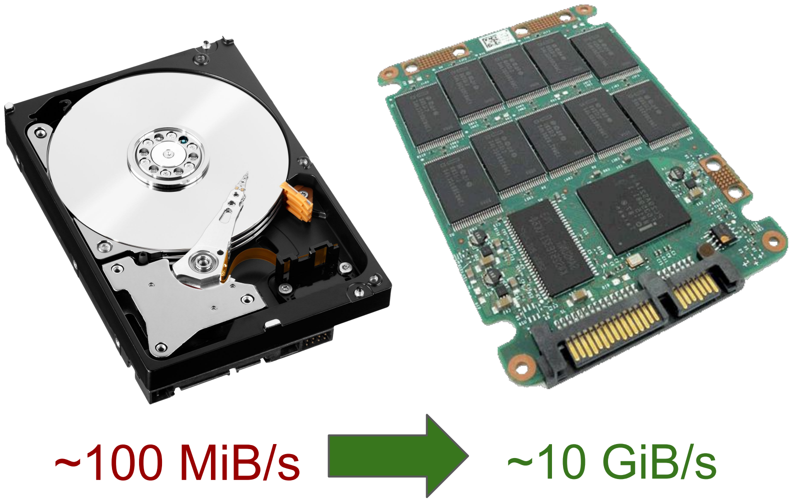
Flash SSDs are **ubiquitous**

Storage demands

- 1 yottabyte every year by 2030![1]
- Performance SLAs

Why flash SSDs?

- Fast storage
- Block interface requires NO changes to host applications



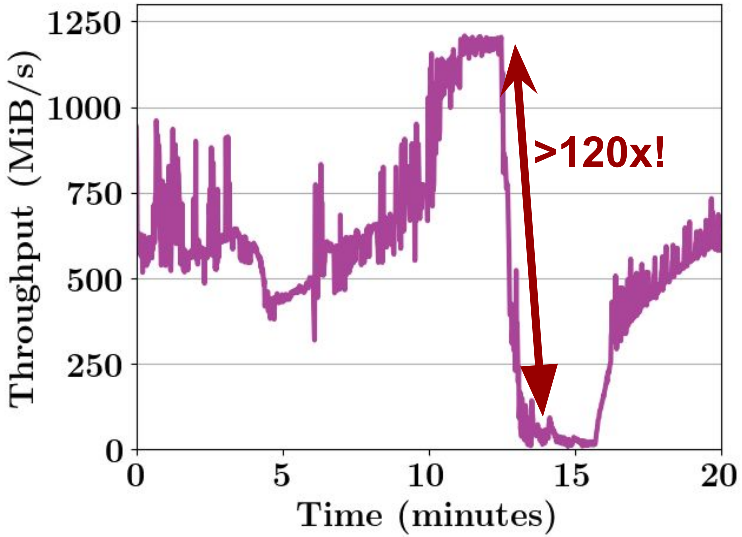
1. Huawei, 2021 https://www-file.huawei.com/-/media/corp2020/pdf/giv/industry-reports/computing_2030_en.pdf

The **problem** of the block interface

Block interface is a **mismatch** for flash

- Requires **firmware** that runs GC
- Unpredictable GC performance

Write throughput over time

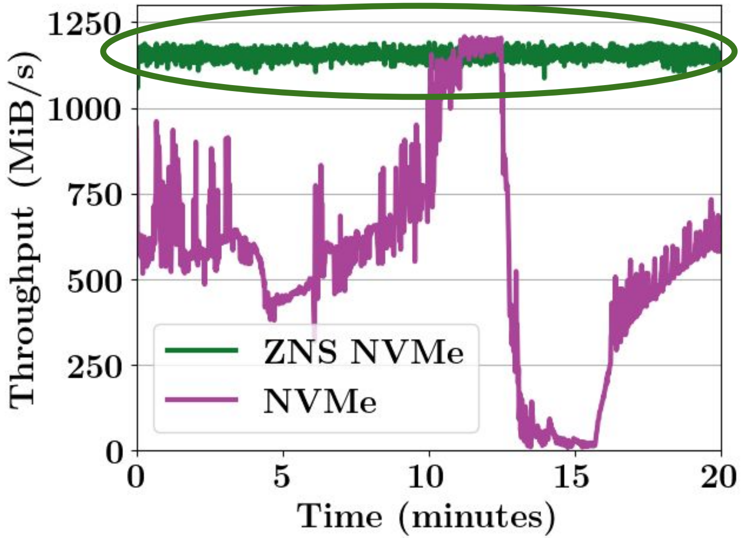


Meet NVMe Zoned NameSpace (ZNS)

ZNS is a **match** for flash

- **Application-managed** GC
- Applications have to be rewritten

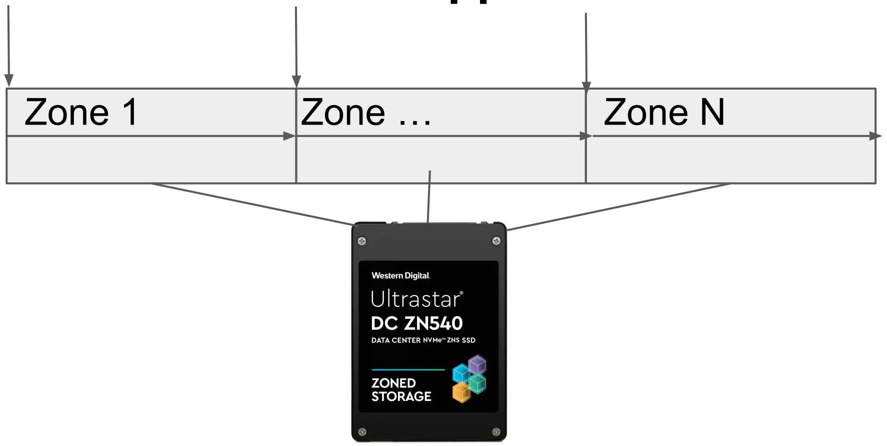
Write throughput over time



Problem: ZNS is **complex** and **novel**

- Device is split in **append-only, application-managed** zones
- 4 ZNS-unique zone management operations
- New zone append operation
- **We do NOT know any performance characteristic!**

NVMe ZNS Applications



How can we design for ZNS if we do not know its **performance model**?

Why Cluster?

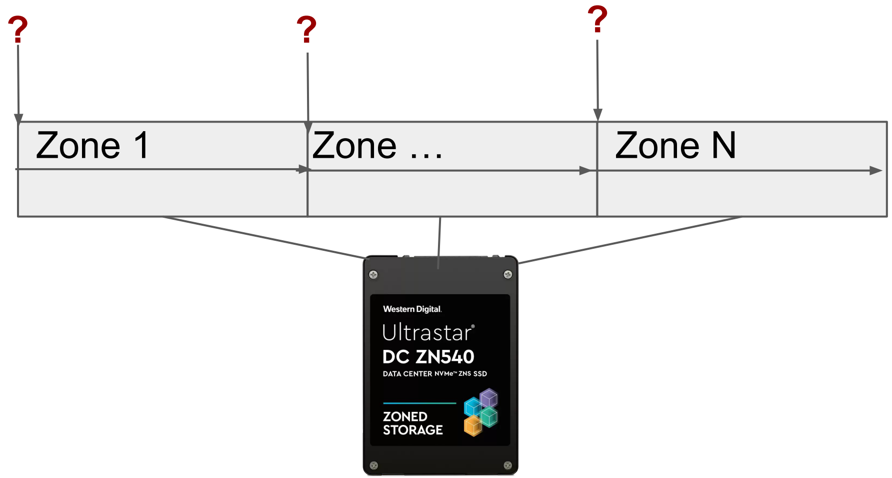
Convergence of HPC and big data

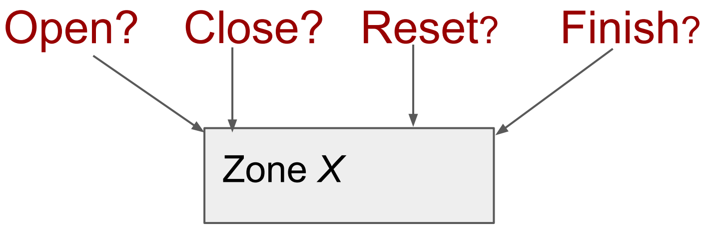
- Performance isolation for HPC
- Cheap, less overprovisioned flash storage for Big Data workloads
- ZNS promises to support both, but we **need** to model its performance first

IEEE CLUSTER 2023
Santa Fe, New Mexico, USA • 31 October–3 November

Performance model: What do we need?

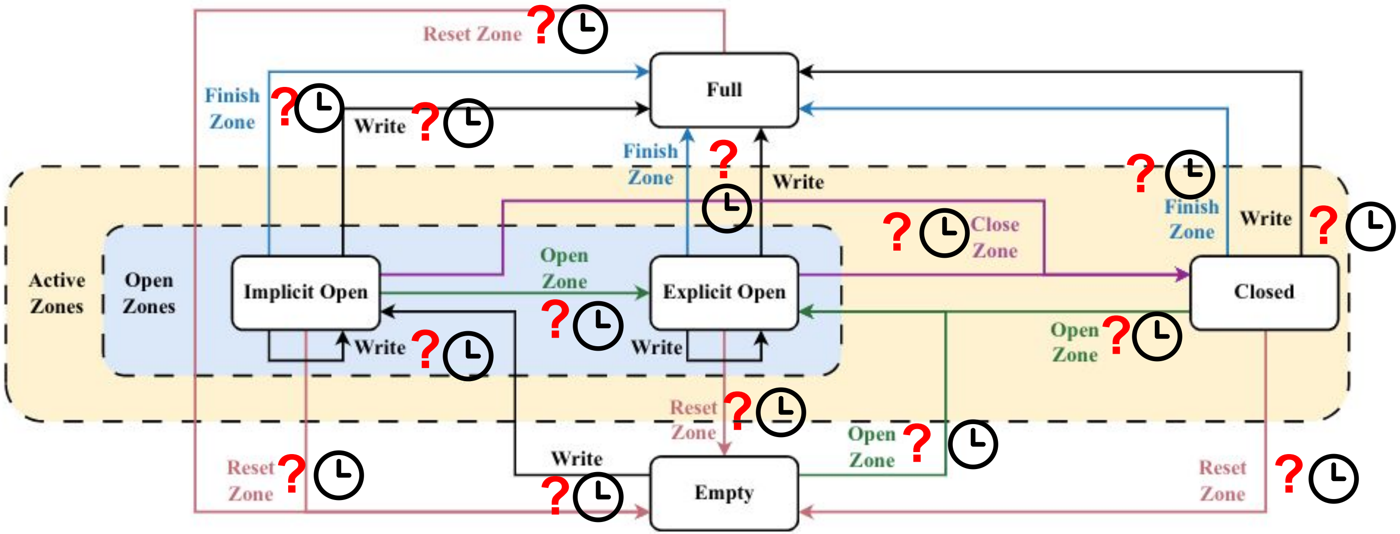
1. Performance scalability of zones
2. Performance of the 4 zone management operations
3. I/O request interference
4. Zone management request interference





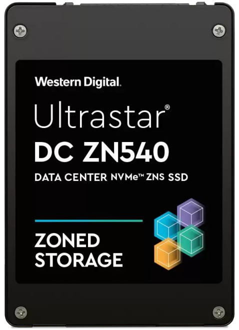
In short, we know none of these…

What is allowed on a zone?



Solution: A characterization framework

- We introduce a characterization framework:
  - Generic to support **any** ZNS device
  - **13 key observations!** (today we discuss only a few)
  - **5 key recommendations!**



https://github.com/stonet-research/NVMeBenchmarks

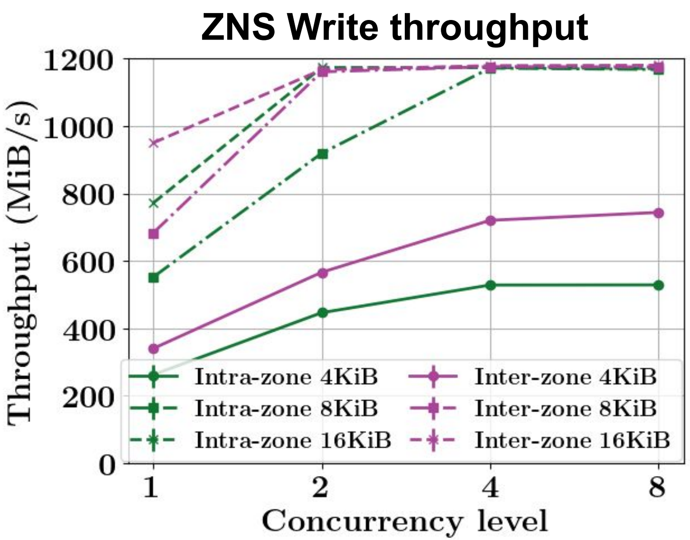
VU

ZNS highlight #1: Write scalability

ZNS Scalability methods:

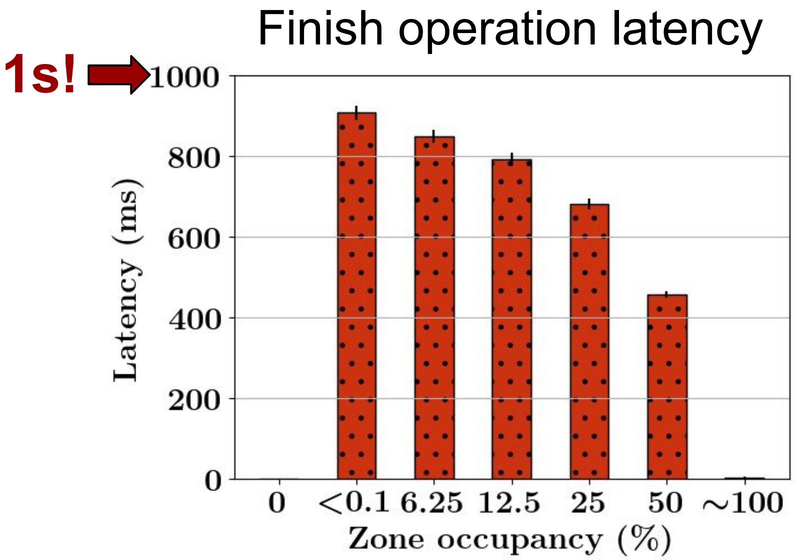
1. Intra-zone: issue zone appends to one zone
2. Inter-zone: write/zone appends to multiple zones concurrently

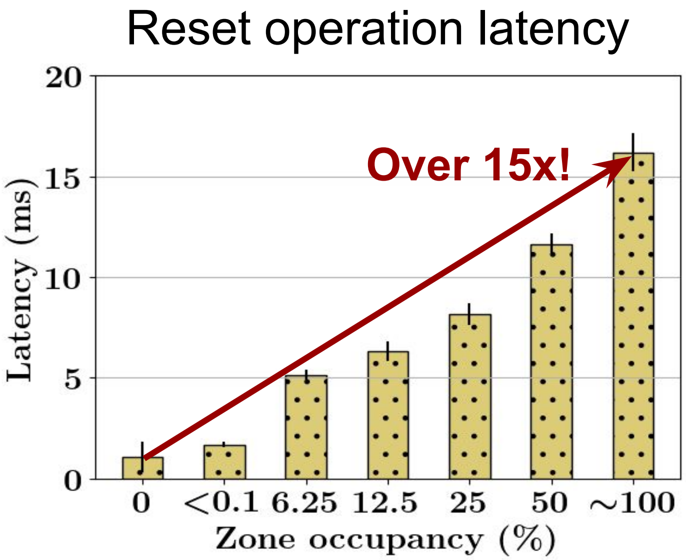
Problem: inter-zone is limited by **active zones!**



ZNS highlight #2: Zone management

- **Expensive** operations (writes are in **μs**)
- Cost depends on zone occupancy





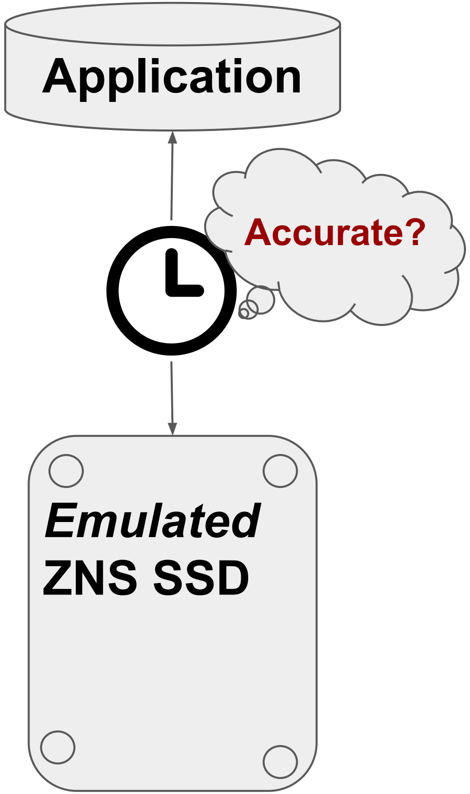
ZNS emulators

Problem:

- Researchers resort to **emulators (at least 4 papers)**
- Available emulators **do not** capture our observations
  - Applications are designed **wrongly!**

Solution:

- Emulators should be changed
- Applications should also be tested on real devices



Application design

We make the following **key recommendations**:

1. Use write instead of append for low latency
2. Prefer intra-zone scalability
3. Avoid finish operations!
4. There is no need to account for GC interference
5. Resets can be issued with concurrent I/O without performance hiccups

RocksDB

F2FS

What is next?

- Extend to more physical ZNS SSDs
  - **Any collaborators?**
- Incorporate our findings into emulators
  - We **need** this for future applications!
- Introduce a ZNS scheduler
- Extend to benchmarking applications

Take-home messages

- Flash SSDs are everywhere
- ZNS enables latency stability for flash SSDs
- ZNS has a unique performance model
- We synthesize this performance model
  - Use this model on your ZNS SSD!
- ZNS emulators are not accurate

Benchmark setup

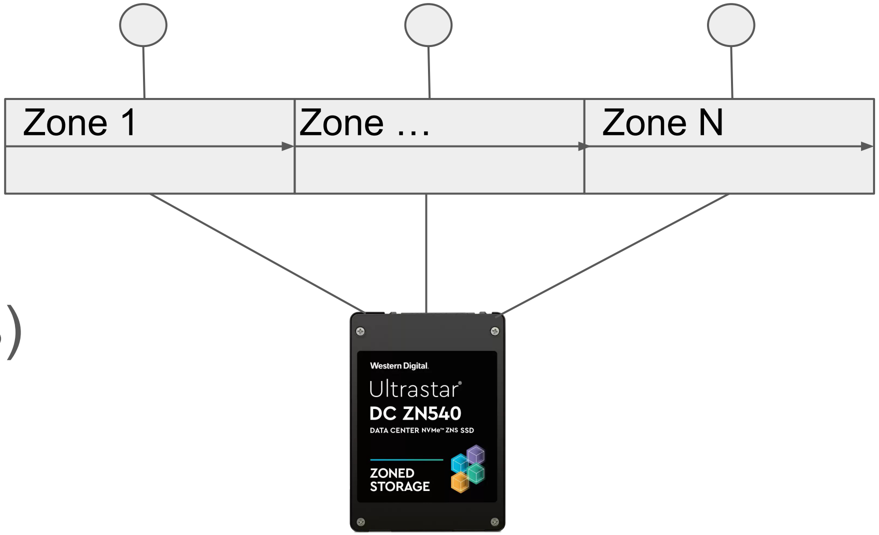
- **raw ZNS command** performance
- synthetic/controlled experiments

OS configuration	**Linux** 5.19, **Ubuntu** 22.04
Workload generator for I/O	***fio*** (3.32, git commit: db7fc8d)
Zone transition benchmarks	**Custom benchmarks** (C++)
Zone transition interference benchmarks	**Custom benchmarks** (C++)

ZNS: A new abstraction

A new abstraction:

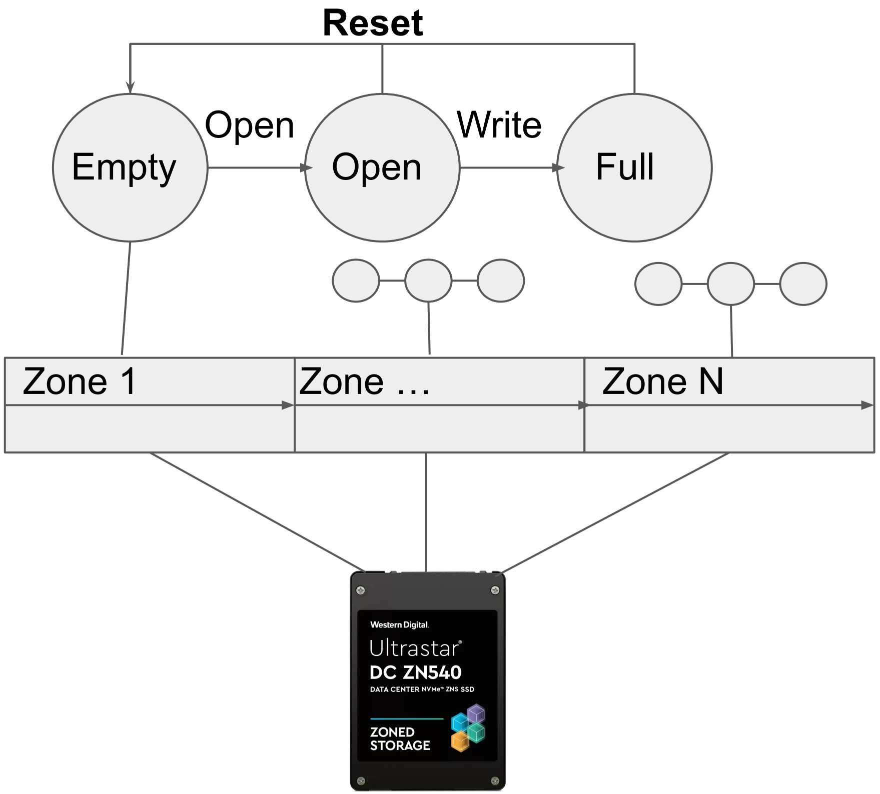
- Device is divided into zones
  - I/O is issued to zones
  - Append-only (**NO** overwrites)
  - Zones have state



ZNS: A new abstraction

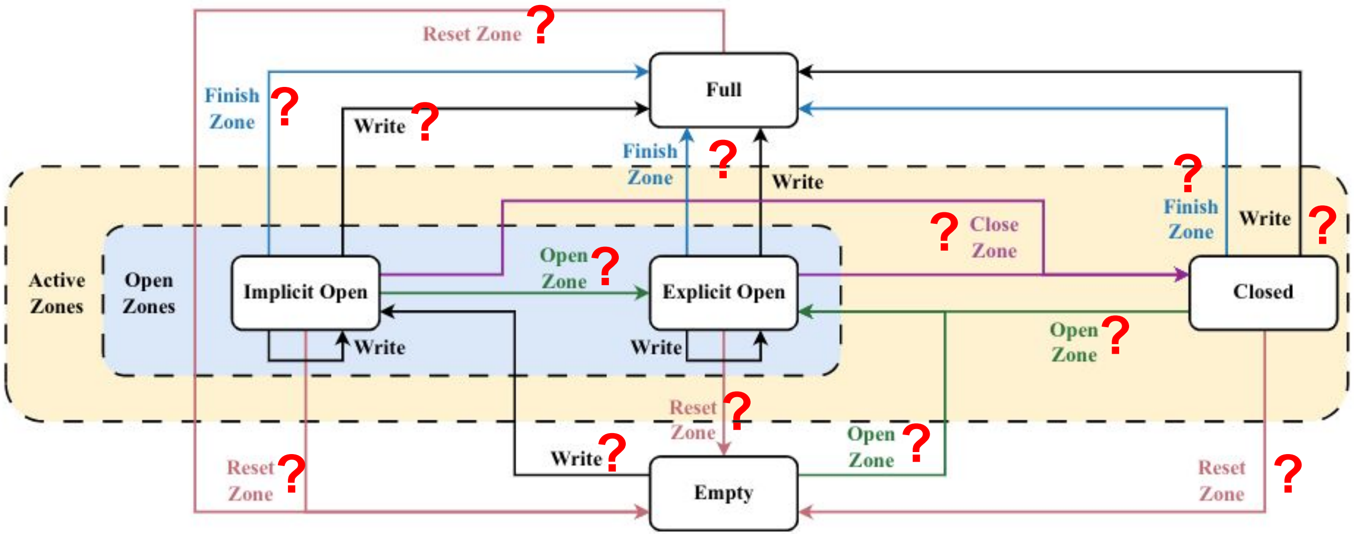
A new abstraction:

- Device is divided into zones
  - Append-only (**NO** overwrites)
  - Zones have state
- **Explicit** State management of zones
  - **Clients** do GC with reset operations
- *What is the performance of ZNS?*



VU

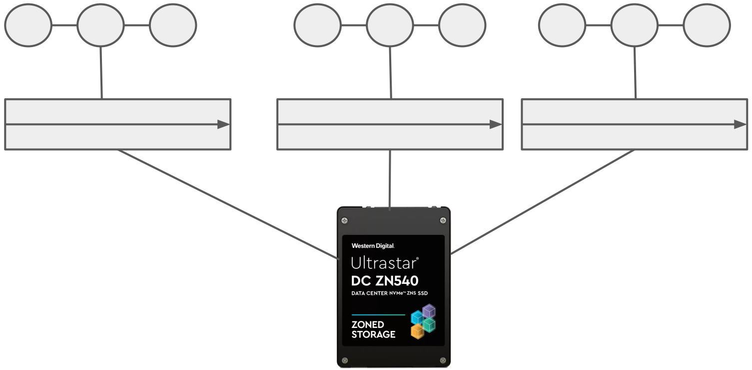
ZNS is complex!



What do we need to know?

What are the ZNS performance characteristics?

- How do we scale I/O and how scalable is ZNS?
- How expensive are zone transition operations?
- Does ZNS suffer from I/O interference?



*We can **not** optimize for ZNS if we do not know its performance characteristics!*

What we measured

What we measured (a lot):

- **Scalability:** Inter- and intra-zone scalability
- **Scalability:** Impact of request size
- **Zone transition overhead:** All zone transitions (reset, open, finish, close)
- **Interference:** Interference of reads/writes and zone transitions/writes/reads

We have **11 key observations**, we will explain **3** of them (**they are essential**!):

1. **Scalability:** Prefer Intra-zone scaling
2. **Zone transition overhead:** Finish operations are the most expensive operations
3. **Zone transition overhead:** Zone occupancy influences transition overhead

VU

ZNS write Scalability: how?

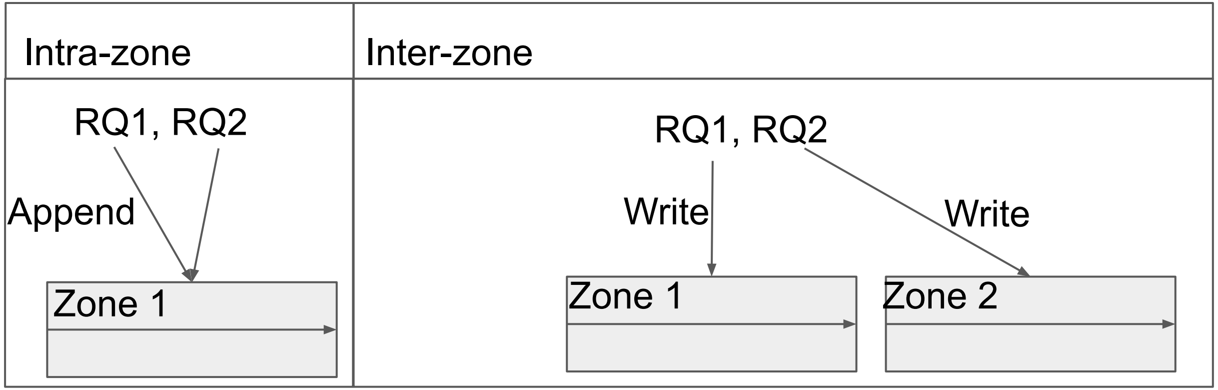
ZNS does not allow multiple writes to 1 zone!

Method 1. Intra-zone:

- **Append**, let the device reorder
  - Applications need to be rewritten…
- **Merge**, merge multiple "writes" on host

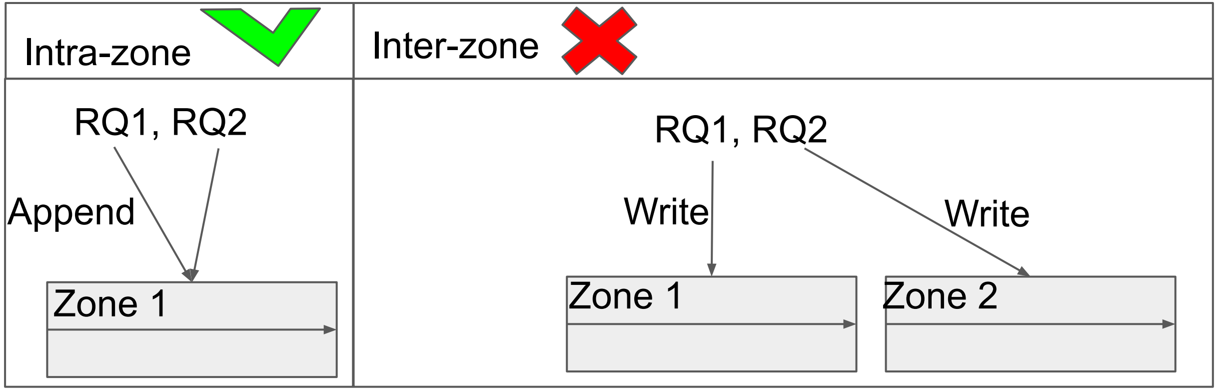
Method 2. Inter-zone:

- **Concurrent zones**
  - Limit **"max open zones"**

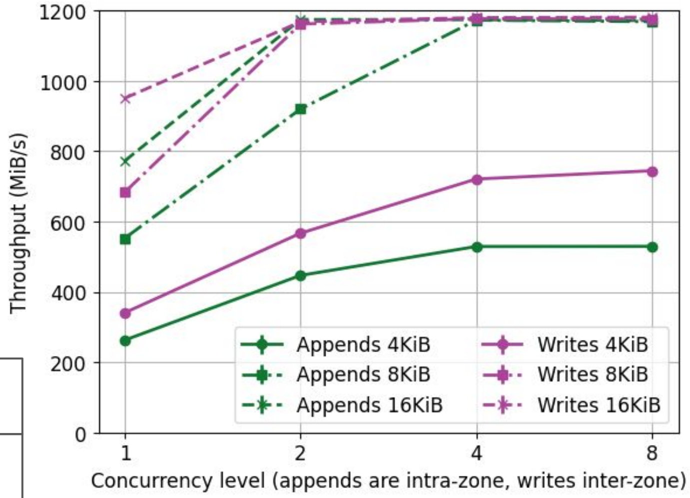


ZNS Scalability: bandwidth

- Both intra- and inter-zone reach device limit
  - Intra is appends, inter is concurrent zone writes
- **Request size** is very important
- Intra is **preferable**!
  - No max zone reached
  - Zones are shared between tenants… (Multi-tenancy)

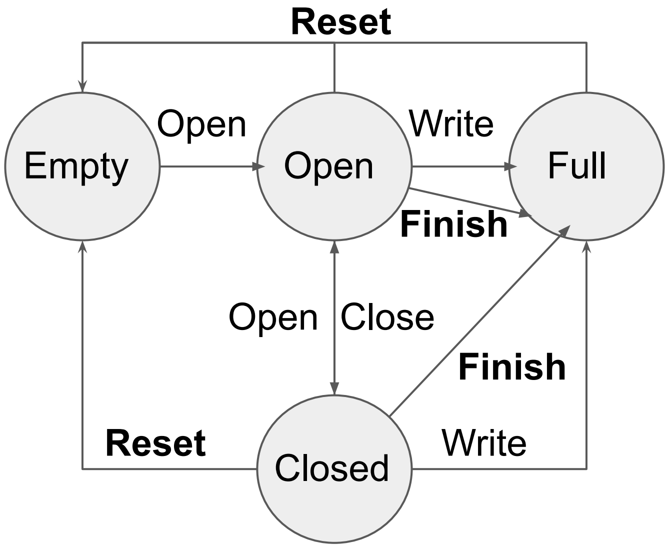


“Write” bandwidth



ZNS: State transitions

- Applications issue **all** transitions
  - Zones **need** to be opened to accept I/O
  - Zones **are limited**
- We evaluate **all** transition **latencies**
  - In isolation, one-by-one
  - Measure submission to completion
  - We fill zones for a percentage (1, …, 100%)
  - Repeated at least a thousand times
- **Important observations:**
  - **Finish** and **Reset** are **expensive**
  - Called regularly
  - These are not **negligible**!



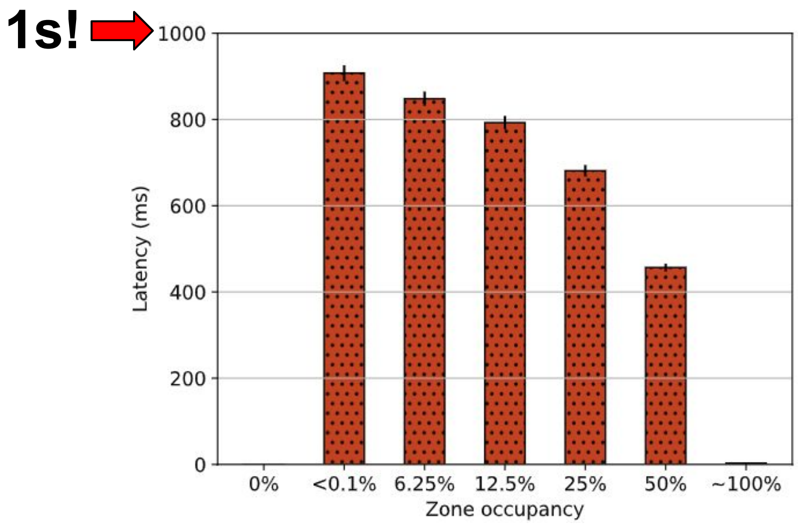
State transitions #1: finish operation

What are finish operations for?:

- Open zones to full zones
- Ensures max open zones is not reached

Results/recommendations:

- The **most** expensive operation!
- **Avoid finishing zones**
- **<u>Do not finish “empty” zones</u>**
- **Prefer intra-zone scalability**



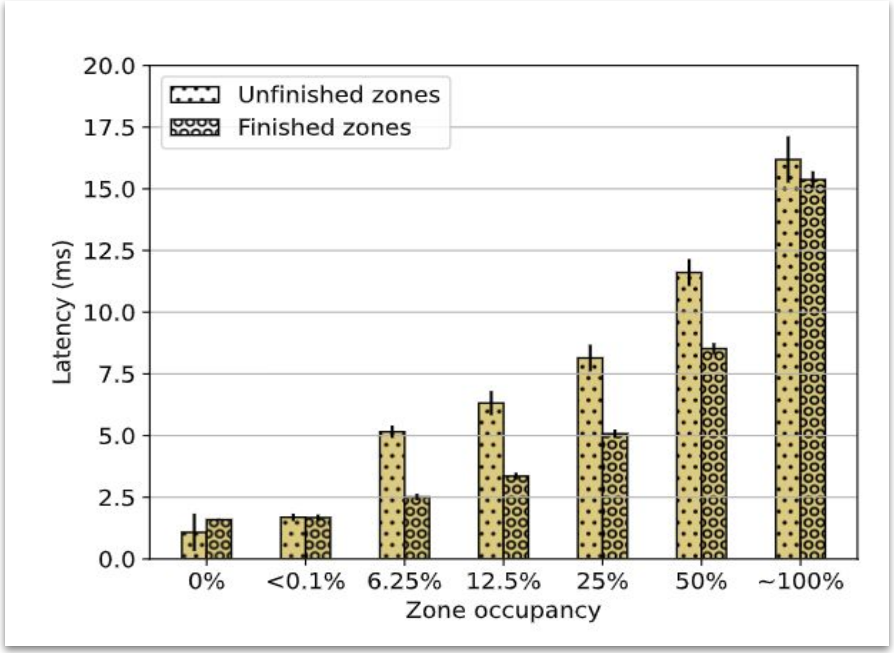
State transitions #2: reset operation

What are reset operations for:

- Zone garbage collection

Results/recommendations:

- Reset latency **correlates** with **zone occupancy**
- Resets are not free
- Resets should be **scheduled** on **zone occupancy**



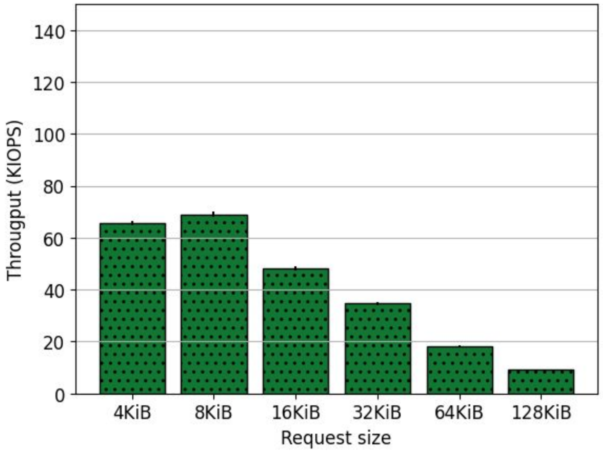
Other results/conclusions…

Please read the paper for more:

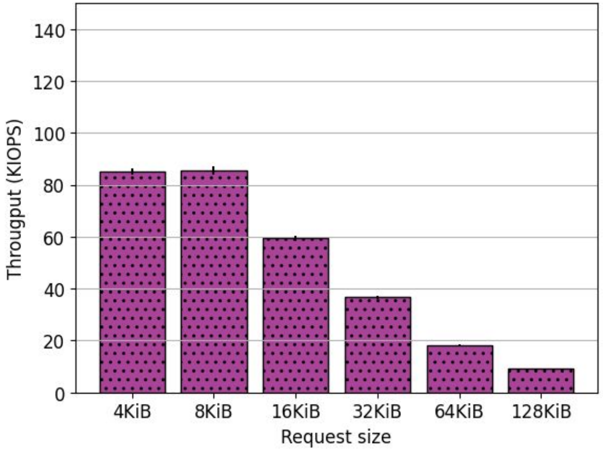
- The impact of I/O size…
- Open/close zone performance…
- I/O interference effects…
- ZNS-aware applications (and how to design them)
- …

Observation: Request size <u>always</u> matters!

Appends



Writes



Scalability #1: intra-zone

Method 1. Writes with scheduler:

- **NVMe operation**
- **One write to one zone allowed?**
- Merge I/O on host

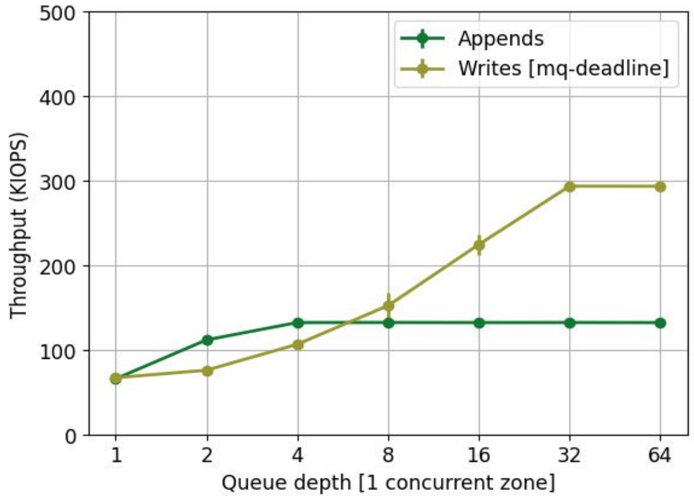
Method 2. Appends:

- **ZNS-specific operation!**
- **Multiple appends to one zone allowed!**

Results/recommendations:

- Prefer appends at low depth
- Use large requests

4KiB “Write” throughput



Scalability #2: inter-zone

Zone parallelism

- We can issue I/O to concurrent zones
- Limited by “max active zone” constraint

Results/recommendations:

- **Writes** have **better** inter-zone scalability

4KiB “Write” throughput