

# zns-tools: An eBPF-powered, Cross-Layer Storage Profiling Tool for NVMe ZNS SSDs

Nick Tehrany, Krijn Doekemeijer, and **Animesh Trivedi**  
Vrije Universiteit Amsterdam

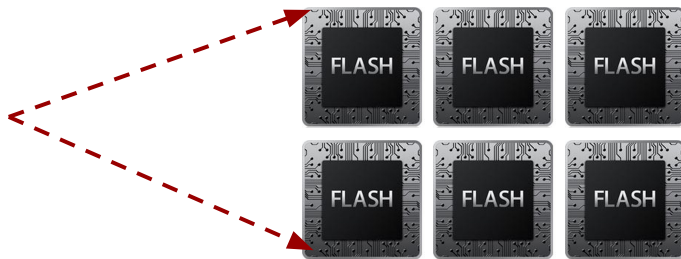


<https://github.com/stonet-research/zns-tools>

# Cloud Computing + Storage



# Challenges with SSDs



Read anywhere  
Write anywhere

Write **once**  
Write **sequentially**  
Erase (or RESET)  
**Garbage collection (GC)**  
**Finite P/E cycle**  
Errors  
Chip management

**Complexity**  
**Unpredictability**  
**Interference**

*(all hidden from us)*

# Lots of research



## Fantastic SSD Int Learn and

Nanqin Lin  
University of Chicago and

Mil  
Univ

## SDF: Software-Defined Flash for Web-Scale Internet Storage Systems

Jian Ouyang Shiding Lin  
Baidu, Inc.  
{ouyangjian, linshiding}@baidu.com

Song Jiang\* Zhe  
Peking University and  
Wayne State University  
sjiang@wayne.edu

### Design Tradeoffs for SSD Performance

Nitin Agrawal\*, Vijayan Prabhakaran, Tej  
John D. Davis, Mark Manasse, Rina Panatier  
Microsoft Research, Silicon Valley  
\*University of Wisconsin-Madison

#### Abstract

Solid-state disks (SSDs) have the potential to revolutionize the storage system landscape. However, there is little published work about their internal organization or the design choices that SSD manufacturers face in pursuit of optimal performance. This paper presents a taxonomy of such design choices and analyzes the likely performance of various configurations using a trace-driven simulator and workload traces extracted from real systems. We find that SSD performance and lifetime is highly workload-sensitive, and that complex system problems that normally appear higher in the storage stack, or even in distributed systems, are relevant to device firmware.

#### 1 Introduction

The advent of the NAND-flash based solid-state storage device (SSD) is certain to represent a sea change in the architecture of computer storage subsystems. These devices are capable of producing not only exceptional bandwidth, but also random I/O performance that is orders of magnitude better than that of rotating disks. Moreover, SSDs offer both a significant savings in power budget and an absence of moving parts, improving system reliability.

Although solid-state disks cost significantly more per unit capacity than their rotating counterparts, there are numerous applications where they can be applied to great benefit. For example, in transaction-processing systems, disk capacity is often wasted in order to improve operation throughput. In such configurations, many small (cost inefficient) rotating disks are deployed to increase I/O parallelism. Large SSDs, suitably optimized for random read and write performance, could effectively replace whole farms of slow, rotating disks. At this writing, small SSDs are starting to appear in laptop configurations because of their reduced power-profile and reliability in portable environments. As the cost of flash continues to decline, the potential application space for solid-state disks will certainly continue to grow.

Despite the promise that SSDs hold, there is little in the literature about the architectural tradeoffs inherent in

their design. We remains the intent as a consequence of a given performance choice. In this paper that are relevant than analyze ssd based disk simulator. We find that SSD performance and lifetime is highly workload-sensitive, and that complex system problems that normally appear higher in the storage stack, or even in distributed systems, are relevant to device firmware.

We find that design approach appeared higher problems, choice. We also relevant to SSD

- **Data placement:** the data placement load balance
- **Parallelism:** any given time period must be considered
- **Write order:** present had randomly
- **Workload:** workload-sensors that "real" workloads do not seek

As SSDs inevitably become in

increases I/O bandwidth by 50% on average SSD-based system used at Baidu, China's largest Internet search company. Currently only 40% or less of the raw bandwidth of the flash memory in the SSDs is delivered by the storage system to the applications. Moreover, because of space over-provisioning in the SSD to accommodate non-sequential or random writes, and additionally, parity coding across flash channels, typically only 50-70% of the raw capacity of a commodity SSD can be used for user data. Given the large scale of Baidu's data center, making the most effective use of its SSDs is of great importance. Specifically, we seek to maximize both bandwidth and usable capacity.

To achieve this goal we propose software-defined flash (SDF), a hardware/software co-designed storage system to maximally exploit the performance characteristics of flash memory in the context of our workloads. SDF exposes individual flash channels to the host software and eliminates space over-provisioning. The host software, given direct access to the raw flash channels of the SSD, can effectively organize its data and schedule its data access to better realize the SSD's raw performance potential.

Currently more than 3000 SDFs have been deployed in Baidu's storage system that supports its web page and image repository systems. Our measurements show that SDF can deliver approximately 95% of the raw flash bandwidth and provide 99% of the flash capacity for user data. SDF

increases I/O bandwidth by 50% on average SSD-based system used at Baidu, China's largest Internet search company. Currently only 40% or less of the raw bandwidth of the flash memory in the SSDs is delivered by the storage system to the applications. Moreover, because of space over-provisioning in the SSD to accommodate non-sequential or random writes, and additionally, parity coding across flash channels, typically only 50-70% of the raw capacity of a commodity SSD can be used for user data. Given the large scale of Baidu's data center, making the most effective use of its SSDs is of great importance. Specifically, we seek to maximize both bandwidth and usable capacity.

To achieve this goal we propose software-defined flash (SDF), a hardware/software co-designed storage system to maximally exploit the performance characteristics of flash memory in the context of our workloads. SDF exposes individual flash channels to the host software and eliminates space over-provisioning. The host software, given direct access to the raw flash channels of the SSD, can effectively organize its data and schedule its data access to better realize the SSD's raw performance potential.

Currently more than 3000 SDFs have been deployed in Baidu's storage system that supports its web page and image repository systems. Our measurements show that SDF can deliver approximately 95% of the raw flash bandwidth and provide 99% of the flash capacity for user data. SDF

increases I/O bandwidth by 50% on average SSD-based system used at Baidu, China's largest Internet search company. Currently only 40% or less of the raw bandwidth of the flash memory in the SSDs is delivered by the storage system to the applications. Moreover, because of space over-provisioning in the SSD to accommodate non-sequential or random writes, and additionally, parity coding across flash channels, typically only 50-70% of the raw capacity of a commodity SSD can be used for user data. Given the large scale of Baidu's data center, making the most effective use of its SSDs is of great importance. Specifically, we seek to maximize both bandwidth and usable capacity.

\*This work was performed during his visiting professorship at Peking University.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permission from permissions@acm.org.  
ASPLOS '14, March 01 - 05 2014, Salt Lake City, UT, USA.  
Copyright © 2014 ACM 978-1-4503-2306-5/14/03...\$15.00.  
http://dx.doi.org/10.1145/2542940.2542959

## The Unwritten Contract of Solid State Drives

Jun He Sudarsun Kannan Andrea C. Arpaci-Dusseau Remzi H. Arpaci-Dusseau  
Department of Computer Sciences, University of Wisconsin-Madison

### LightNVM: The

Matias Björling

†CNEX Lab

#### Abstract

As Solid-State Drives (SSDs) become data-centers and storage arrays, they demand for predictable latency. Traditional block I/Os, fail to meet this demand. High-level of abstraction at the cost of performance and suboptimal resource utilization pose that SSD management trade-offs is through *Open-Channel Reservations*, a new design gives hosts control over their internal, experience building *LightNVM*, the Linux SSD subsystem. We introduce a new *Flash* interface that exposes SSD page media characteristics. *LightNVM* is designed storage stacks, while also enabling to take advantage of the new I/O primitives results demonstrate that *LightNVM* overhead, that it can be tuned variability and that it can be used to predictably I/O latencies.

#### 1 Introduction

Solid-State Drives (SSDs) are projected to be the dominant form of secondary storage in the next few years [18, 19, 31]. Despite their superior performance, SSDs suffer well-documented issues: log-on-log [37, 57], large tail-latency predictable I/O latency [12, 28, 30], an utilization [1, 11]. These shortcomings hardware limitations: the non-volatile core of SSDs provide predictable I/O at the cost of constrained operations duration/reliability. It is less than of new chips are managed within an SSD, per block I/O interface as a magnetic disk these shortcomings [5, 52].

A new class of SSDs, branded as *Open-Channel SSDs*, are designed to provide predictable I/O latencies. These SSDs expose the underlying NAND flash characteristics to the host software, enabling the host to manage the SSD's internal state and to predictably I/O latencies.

USENIX Association

#### Abstract

We perform a detailed vertical analysis of application performance atop a range of modern file systems and SSD FTLs. We formalize the "unwritten contract" that clients of SSDs should follow to obtain high performance, and conduct our analysis to uncover application and file system designs that violate the contract. Our analysis, which utilizes a highly detailed SSD simulation underneath traces taken from real workloads and file systems, provides insight into how to better construct applications, file systems, and FTLs to realize robust and sustainable performance.

#### 1. Introduction

In-depth performance analysis lies at the heart of systems research. Over many years, careful and detailed analysis of memory systems [26, 81], file systems [36, 50, 51, 66, 84, 87], parallel applications [91], operating system kernel structure [35], and many other aspects of systems [25, 29, 37, 41, 65] has yielded critical, and often surprising, insights into systems design and implementation.

However, perhaps due to the rapid evolution of storage systems in recent years, there exists a large and important gap in our understanding of I/O performance across the storage stack. New data-intensive applications, such as LSM-based (Log-Structured Merge-tree) key-value stores, are increasingly common [6, 14]; new file systems, such as ZFS [62], have been created for an emerging class of flash-based Solid State Drives (SSDs); finally, the devices themselves are rapidly evolving, with aggressive flash-based translation layers (FTLs) consisting of a wide range of optimizations. How well do these applications work on these modern file systems, when running on the most recent class of SSDs? What aspects of the current stack work well, and which do not?

The goal of our work is to perform a detailed vertical analysis of the application/file-system/SSD stack to answer the aforementioned questions. We frame our study around the file-system/SSD interface, as it is critical for achieving high performance. While SSDs provide the same interface

as hard drives, how higher layers utilize said interface can greatly affect overall throughput and latency.

Our first contribution is to formalize the "unwritten contract" between file systems and SSDs, detailing how upper layers must treat SSDs to extract the highest instantaneous and long-term performance. Our work here is inspired by Schlosser and Ganger's unwritten contract for hard drives [82], which includes three rules that must be tacitly followed in order to achieve high performance on Hard Disk Drives (HDDs); similar rules have been suggested for SMR (Shingled Magnetic Recording) drives [46].

We present five rules that are critical for users of SSDs. First, to exploit the internal parallelism of SSDs, SSD clients should issue large requests or many outstanding requests (*Request Scale* rule). Second, to reduce translation-cache misses in FTLs, SSDs should be accessed with locality (*Locality* rule). Third, to reduce the cost of converting page-level to block-level mappings in hybrid-mapping FTLs, clients of SSDs should start writing at the aligned beginning of a block boundary and write sequentially (*Aligned Sequentiality* rule). Fourth, to reduce the cost of garbage collection, SSD clients should group writes by the likely death time of data (*Grouping By Death Time* rule). Fifth, to reduce the cost of wear-leveling, SSD clients should create data with similar lifetimes (*Uniform Data Lifetime* rule). The SSD rules are naturally more complex than their HDD counterparts, as SSD FTLs (in their various flavors) have more subtle performance properties due to features such as wear leveling [30] and garbage collection [31, 71].

We utilize this contract to study application and file system pairings atop a range of SSDs. Specifically, we study the performance of four applications – LevelDB (a key-value store), RocksDB (a LevelDB-based store optimized for SSDs), SQLite (a more traditional embedded database), and Varmal (an email server benchmark) – running atop a range of modern file systems – Linux ext [69], XFS [88], and the flash-friendly ZFS [62]. To perform the study and extract the necessary level of detail our analysis requires, we build *Wisc5E*, an analysis tool, along with *Wisc5Esim*, a detailed and extensively evaluated discrete-event SSD simulator that can model a range of page-mapped and hybrid FTL designs [48, 54, 57, 74]. We extract traces from each application/file-system pairing, and then, by applying said traces to *Wisc5Esim*, study and understand details of system performance that previously were not well understood. *Wisc5E* and *Wisc5Esim* are available at <http://research.cs.wisc.edu/adsl/Software/wisc5ee/>.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permission from permissions@acm.org.  
ASPLOS '14, April 22 - 26, 2014, Singapore, S'gapore.  
© 2014 Copyright held by the author(s). Publication rights reserved by ACM.  
10.1145/2542940.2542959  
DOI: http://dx.doi.org/10.1145/2542940.2542959

15th USENIX Conference on File and Storage Technologies 359

# Rise of the Open SSD Interfaces



**Idea: make SSD internal state and operations more ...**

- Visible
- Under the control of the host system software (OS)

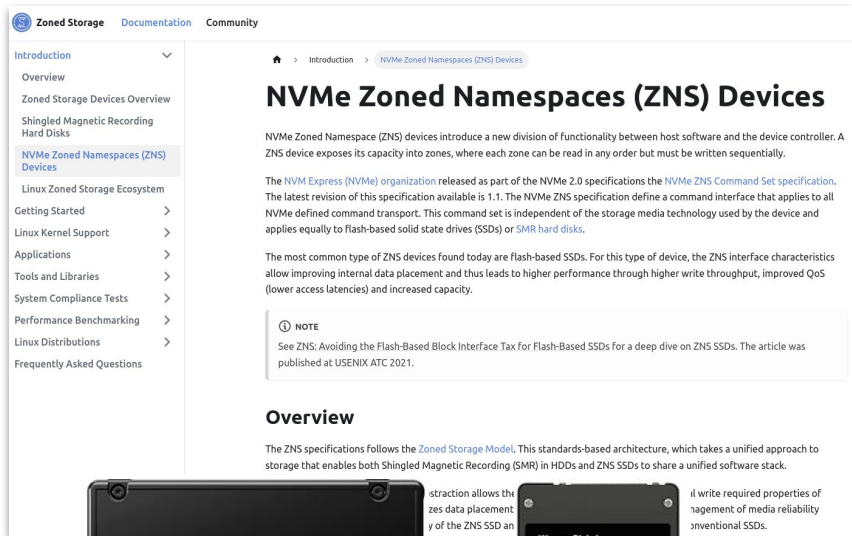
Multiple design points: Software-defined Flash, Open Channel SSD, Stream SSD...

**Today: Zone Namespace SSDs or ZNS SSD**

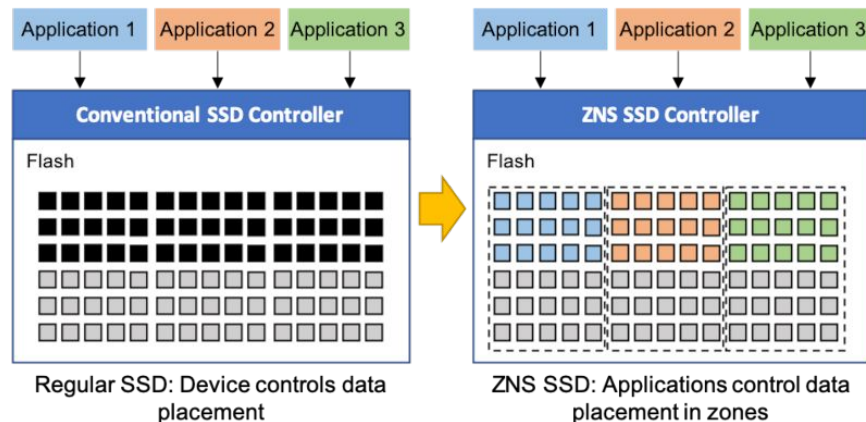
**Benefits:**

- An OS can decide how store data and manage SSD storage
- An OS can plan (resource management) for a particular QoS
- An OS can explain why performance suffered
- Simplified SSD internals

# ZNS: The New Storage Interface and Capabilities



The screenshot shows the 'NVMe Zoned Namespaces (ZNS) Devices' page from the Zoned Storage documentation. The page title is 'NVMe Zoned Namespaces (ZNS) Devices'. The main content includes an overview of ZNS devices, their introduction, and their characteristics. A note mentions that the latest revision of the NVMe ZNS Command Set specification is 1.1. The page also includes a sidebar with navigation links and a footer with the text 'Standardized in the NVMe 1.4, July 2021'.



<https://zonedstorage.io/docs/introduction/zns>

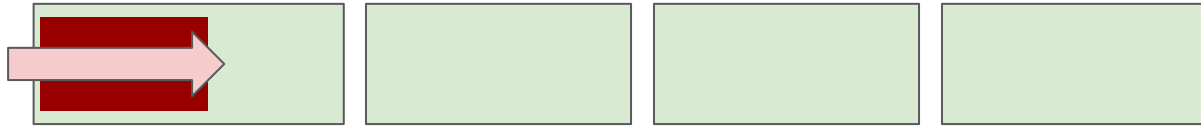
Standardized in the NVMe 1.4, July 2021

# ZNS Introduction: New I/O and Management Operations



1

Zones (written sequentially)

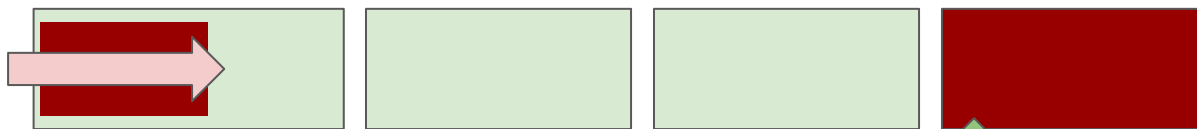


# ZNS Introduction: New I/O and Management Operations



1

Zones (written sequentially)



2

Fully written zone is Erased by the host (OS)

A new command : RESET



# ZNS Introduction: New I/O and Management Operations



1

Zones (written sequentially)



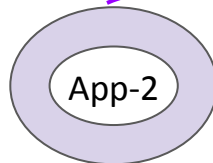
2

Fully written zone is Erased by the host (OS)

A new command : RESET

3

Application-controlled data placements in zones

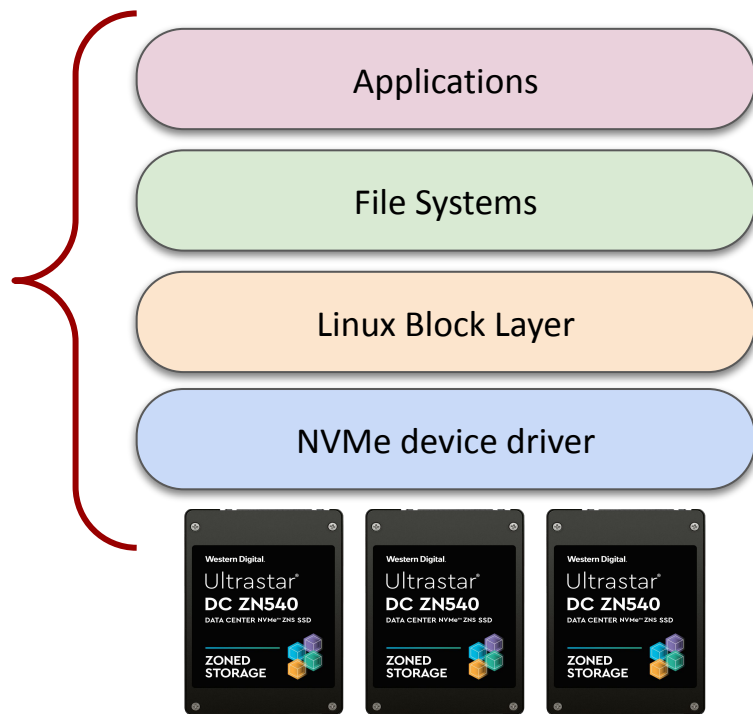


# What Does This Mean?

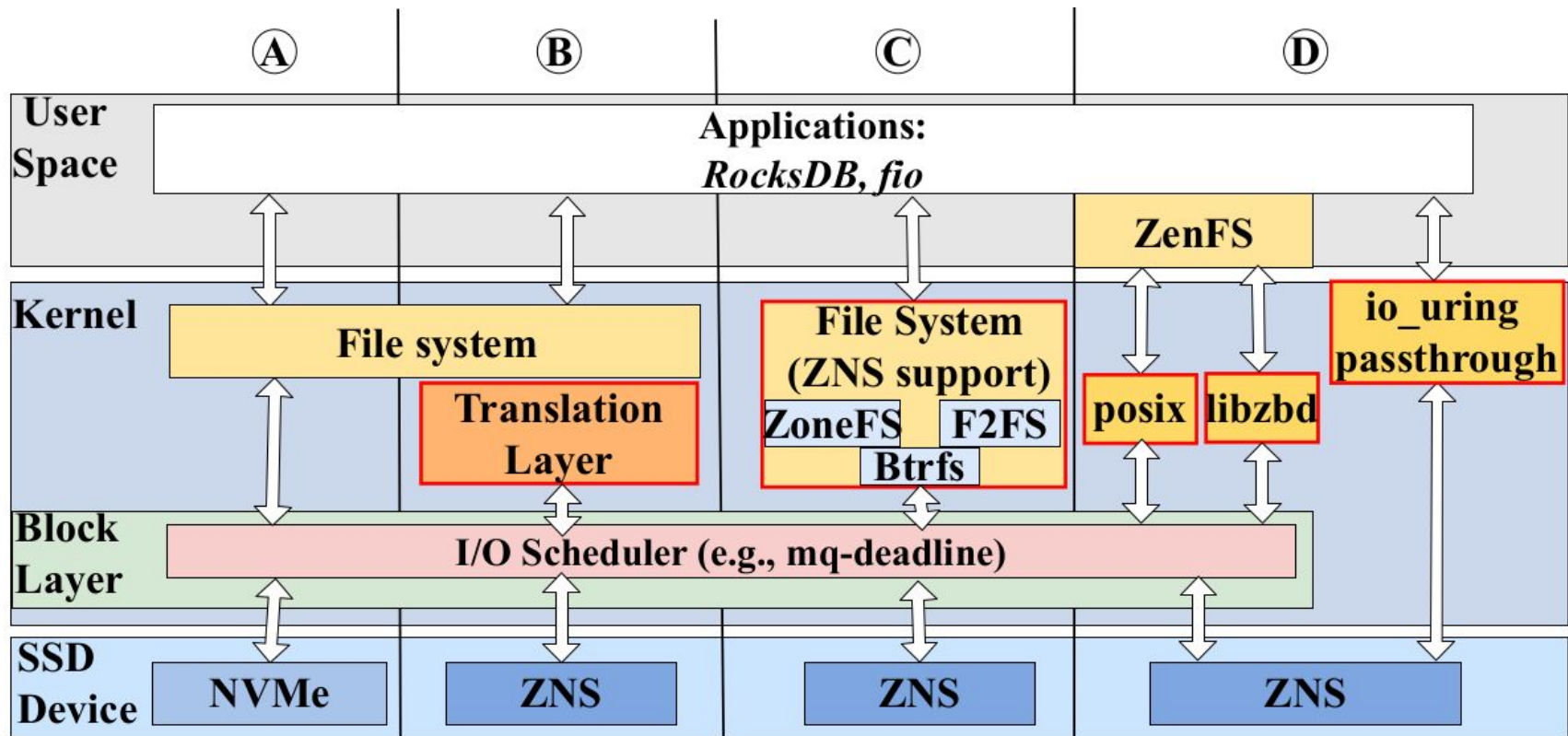
Better control (visibility) on ...

1. Data placement in zones
2. Parallelism management
3. Garbage collection
4. Zone resets

With this visibility, we have an opportunity to build a more expressive, complete & comprehensive **Data Lifecycle Event tracing framework!**



# ZNS : Software Integration Challenges (non-Trivial)



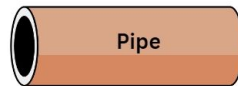
# zns-tools: An eBPF-powered Profiling Tool for NVMe ZNS SSDs

## Do one thing well (the UNIX pipe philosophy)

- Collection of tools
- (WiP) `user@system:~$ tool1 | tool2 | tool3`



Memory Buffer



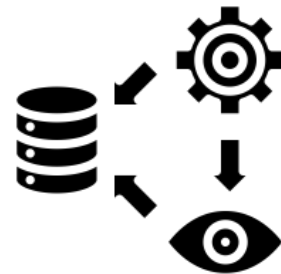
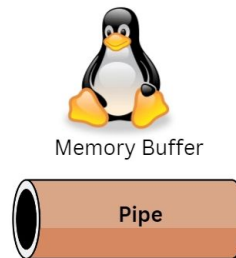
# zns-tools: An eBPF-powered Profiling Tool for NVMe ZNS SSDs

## Do one thing well (the UNIX pipe philosophy)

- Collection of tools
- (WiP) `user@system:~$ tool1 | tool2 | tool3`

## Keep it modular and standardized

- Follows the Model-View-Controller model
- Decouples trace gathering, processing, and visualization
- BPFtrace output and JSON format (but extensible)



# zns-tools: An eBPF-powered Profiling Tool for NVMe ZNS SSDs

## Do one thing well (the UNIX pipe philosophy)

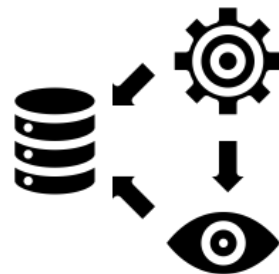
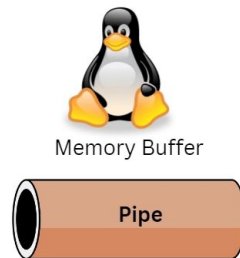
- Collection of tools
- (WiP) `user@system:~$ tool1 | tool2 | tool3`

## Keep it modular and standardized

- Follows the Model-View-Controller model
- Decouples trace gathering, processing, and visualization
- BPFtrace output and JSON format (but extensible)

## Keep it lightweight

- eBPF !
- Supports complex data structure walks, and parsing



# [1/3] `zns-tools.nvme`: Profiling the ZNS Device

## NVMe driver (ZNS) and the Linux block layer profiler

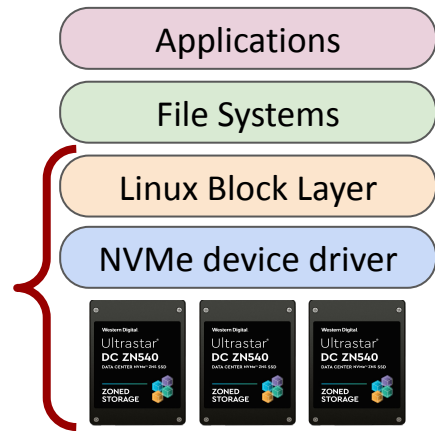
### What to trace?

- All I/O commands: Read, Writes, Appends
- All management commands: Reset, Finish
- Size of the payload, timestamp ...

Data is then grouped on per-zone basis

### Insights:

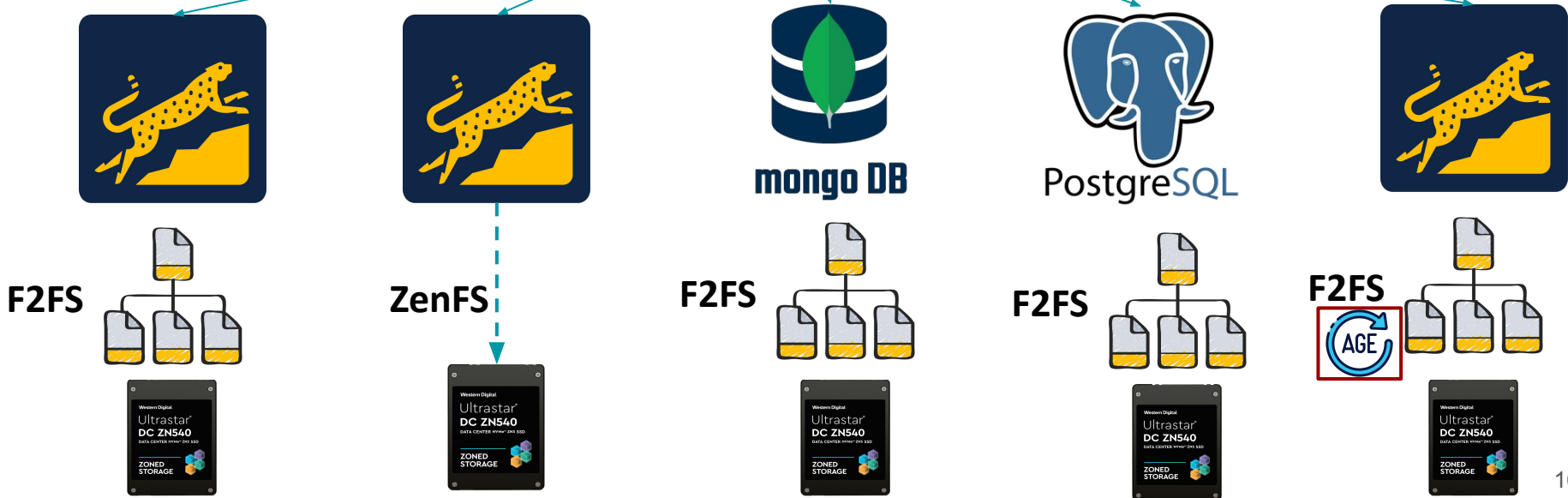
- *Are all zones uniformly used (wear-leveling)?*
- *Are there heavily over written or read zones?*



```
user@system:~$ zns-tools.nvme nvme2n1
```

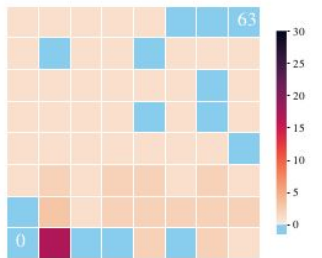
# Exploring the ZNS Software Integration Options

YCSB, Workload-A (50% read, 50% write)

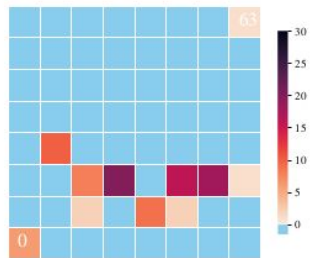




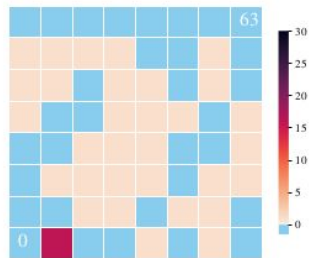
# Insight: Not all the ZNS Integration options are the same!



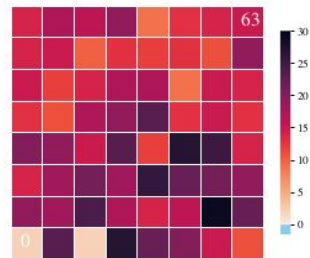
(a) RocksDB + F2FS



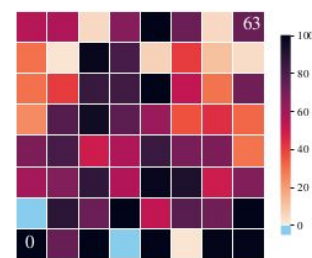
(b) RocksDB + ZenFS



(c) MongoDB + F2FS

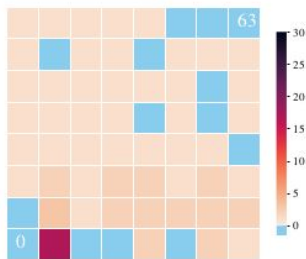


(d) PostgreSQL + F2FS

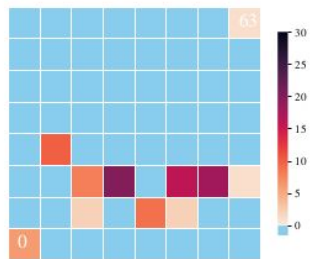


(e) RocksDB + aged F2FS

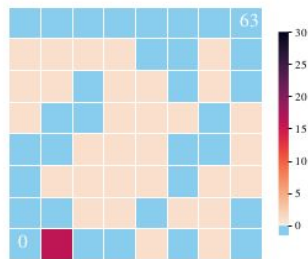
# Insight: Not all the ZNS Integration options are the same!



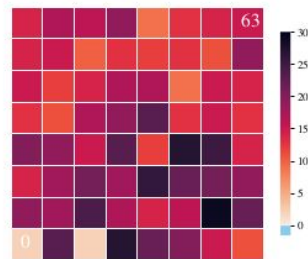
(a) RocksDB + F2FS



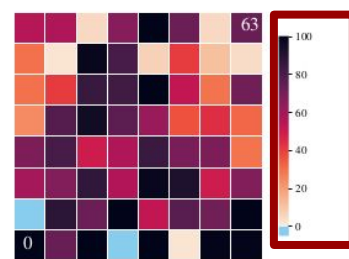
(b) RocksDB + ZenFS



(c) MongoDB + F2FS



(d) PostgreSQL + F2FS



(e) RocksDB + aged F2FS

F2FS has an even zone usage

Domain-specific, ZenFS does not any wear-leveling on zones

PostgreSQL issues a lot more zone Resets

Aged F2FS issues significantly more Resets

# [2/3] zns-tools.fs: Profiling the File System

## Linux file system and the VFS-level event tracer

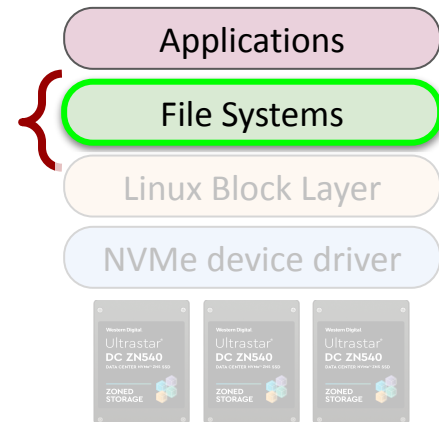
### What to trace?

- **File-level** I/O events (open, read, write, close)
- **File system-level** life cycle events  
(garbage collections for LFS, file hotness levels)

Three subtools: `fs.fiemap`, `fs.imap`, and `fs.segmap` [[github](#)]

### Insights:

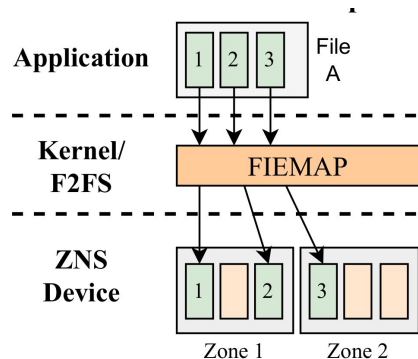
- *In which zones my files are stored, what is the fragmentation level?*
- *Is my file stored in hot or cold ZNS zone?*
- *How many hot files are in this directory?*



# zns-tools.fs: Two Specific Responsibilities

## [Any FS] Locate a file (name) to its zone storage location

- Uses FIEMAP (ioctl) call to extract file extents (LBA address, length)
- Generate: address-to-zone mappings, extent distribution (min, max, percentiles), hole statistics, zone-level placement information



## [F2FS-specific, semantic] Identify hotness classifications of F2FS segments and ZNS zones

- **Data:** file data and file metadata (inode), **Temperate classes:** Hot, Warm, Cold
- Read F2FS metadata (/proc, F2FS checkpoints for NAT)
- Any file/directory → set {F2FS segments, zones, temperature}
- Any zone → set {F2FS file segments, inode segments, file names, offsets}

## [ 3/3 ] `nvme . app`: Profiling the Whole System

Profile any user defined events functions with {kernel, userspace} eBPF probes

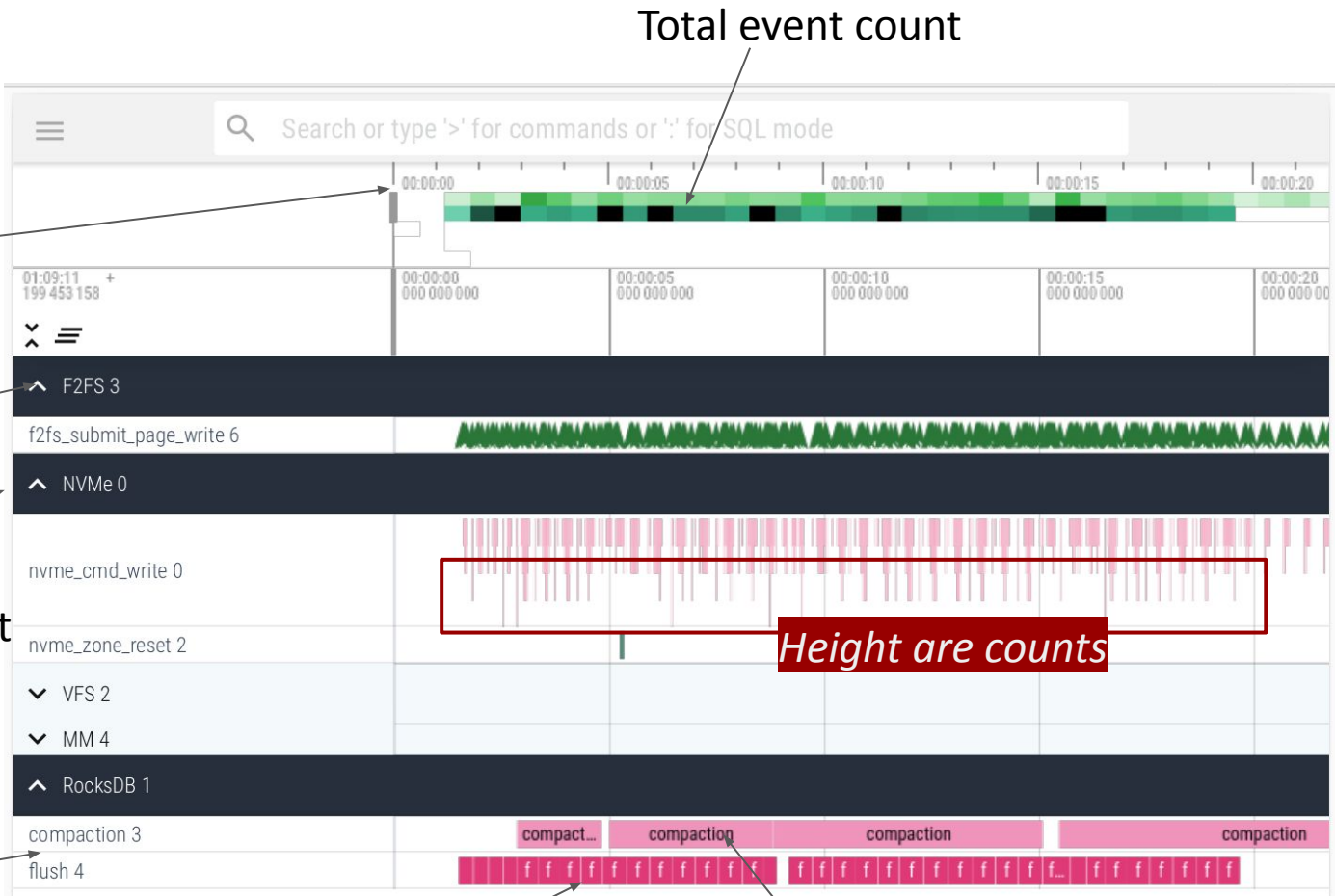
What to trace?

- **Application-level:** (LSM KV) compaction, garbage collection, WAL and memtables flushes
- **File system-level:** All previously discussed file system related events
- **NVMe-level:** All previously nvme, block-level events

**Insights:**

- *Why does my application data got mixed with other application data?*
- *Why there was a reset issues to this particular zone?*
- *Why did my P99 latencies increased from the baseline?*

# Example Run



Total event count

Timeline stamps

Which layer

Expanded version with tracing of write and reset

Height are counts

Application-level events

“flush” probes

“compaction” probes

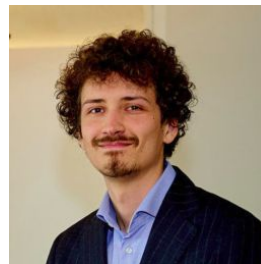
## Future Work

- **Implement the UNIX pipe design**
- Deep integration with F2FS and Btrfs
- Common abstraction to trace requests across the stack (currently time based)
- Expand to new applications
- NVMe FDP support
- Performance evaluation to high-capacity ZNS arrays (scaling challenge)
- Interactive visualization
- Storage traces in databases for more expressive analysis

# Thank you



Nick Tehrany



Krijn Doekemeijer



<https://github.com/stonet-research/zns-tools>



<https://atlarge-research.com/pdfs/2024-zns-tools.pdf>

**Acknowledgments:** The Dutch Research Council (NWO) and Western Digital



# Overheads

