

Modern NVMe SSD Architecture:

Enabling host↔device communication: Zoned Namespaces (ZNS)
and Flexible Data Placement (FDP)

Dev Purandare

Center for Research in Systems and Storage
UC Santa Cruz

UC SANTA CRUZ
BaskinEngineering



Center for Research
in Systems and Storage

Layout of the talk

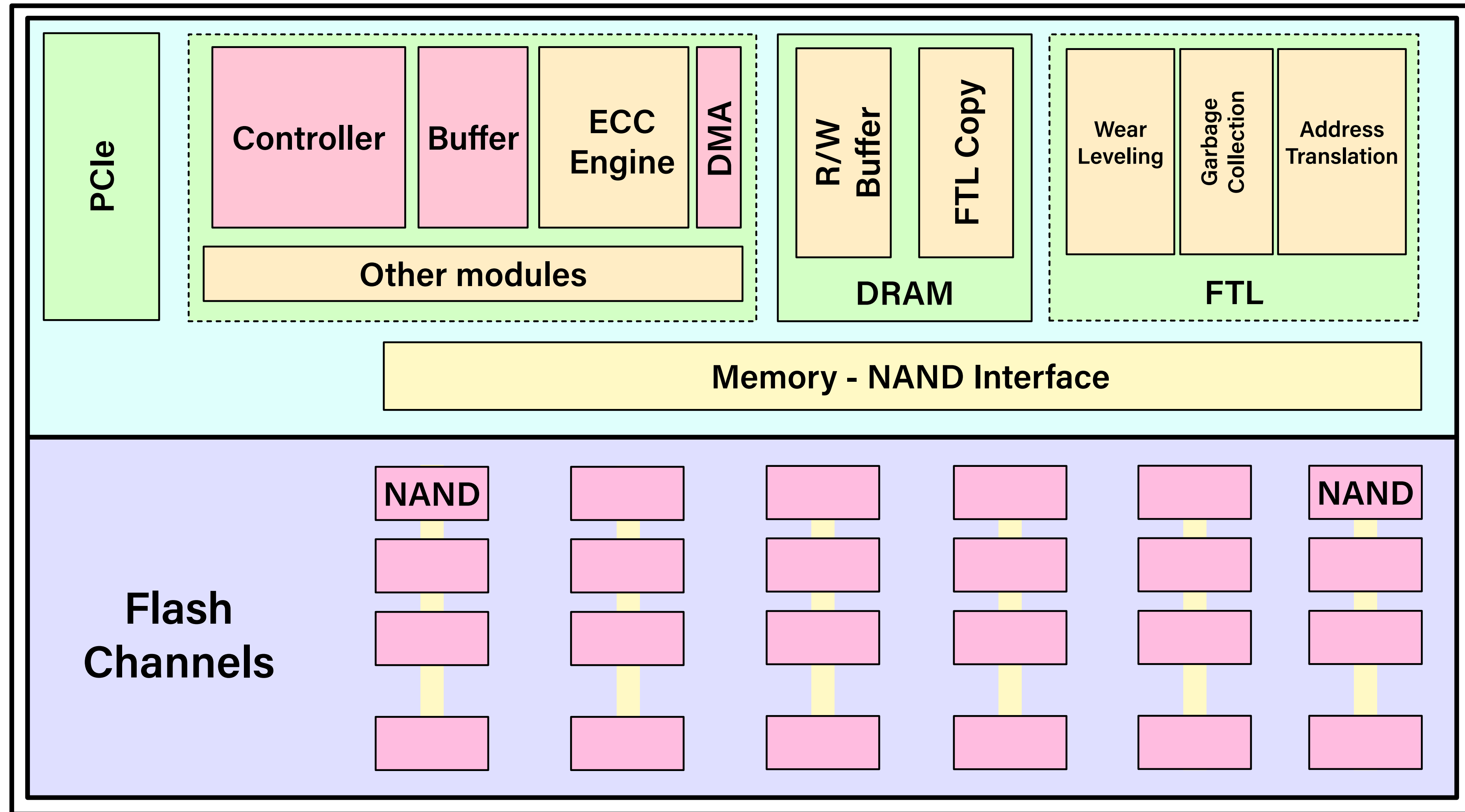
1. Traditional SSDs and the motivation to change
2. Modern NVMe Command Sets: ZNS, and FDP
3. How to think of data layout
4. Available hardware, emulators, and testing
5. Tools, libraries, and Kernel Support
6. Filesystems, Applications, and popular projects
7. Blueprints for deploying modern SSDs

Everything referenced in this talk is also listed at: <https://github.com/devashishp/FMS>

“SSDs will continue to improve by some metrics (notably density and cost per bit), but everything else about them is poised to get worse.”

— The Bleak Future of NAND Flash Memory (2012)

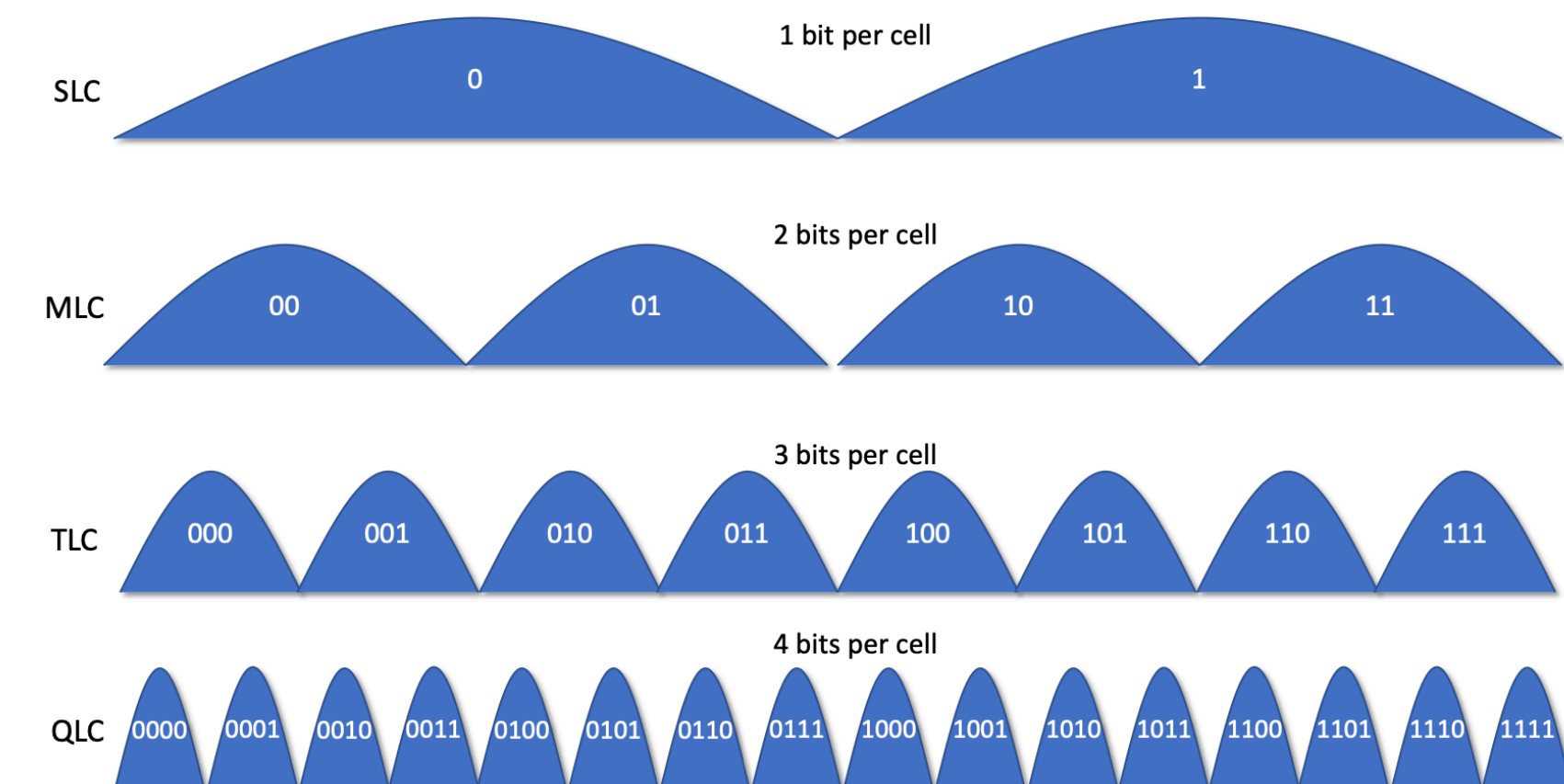
Structure of an SSD



Density comes at a cost

- ▶ Flash storage has grown in capacity faster than any other kind of storage propelled by:
 - ▶ Die shrink
 - ▶ Packing more bits in each cell
 - ▶ 3D stacking
- ▶ Each increase in density of a cell comes at a cost of performance and durability

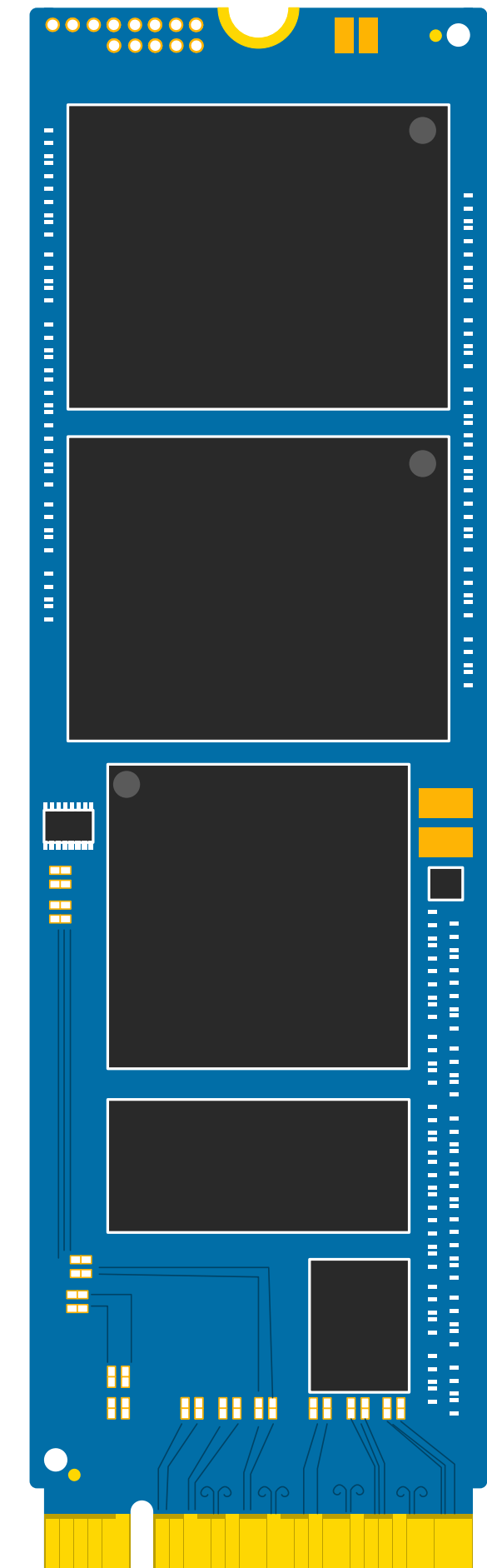
NAND Flash SSD Technology



Source: IDC, 2018

We can't change physics but,

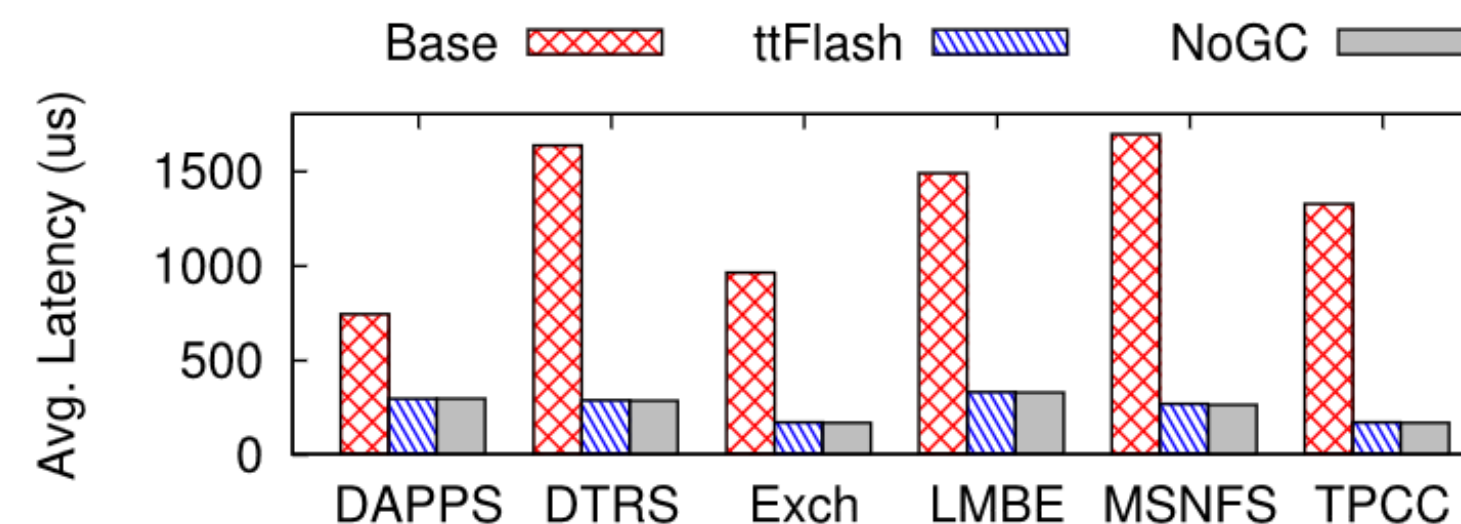
- ▶ We can help the devices perform better and last longer
- ▶ NAND-based flash cannot perform in-place updates
 - ▶ Random Writes are mapped to new location on the device
 - ▶ Causes metadata overhead
- ▶ The erase unit of an SSD is 10-100× the write unit
 - ▶ Valid blocks need to be moved on erase
 - ▶ This affects cost, performance, and lifetime
- ▶ If we reduce in-place updates and co-locate data that will be deleted together, we can reduce this overhead



Garbage Collection

- Requires moving valid data to new blocks before erase
- **Impacts cost**
 - Overprovisioning (7% to 28%)
 - Extra DRAM needed — 1GB per 1TB of flash
- **Impacts performance**
 - Increased tail latency
 - Degraded throughput
- **Impacts Lifetime**
 - Write Amplification Factor >1 (2× - 5×)

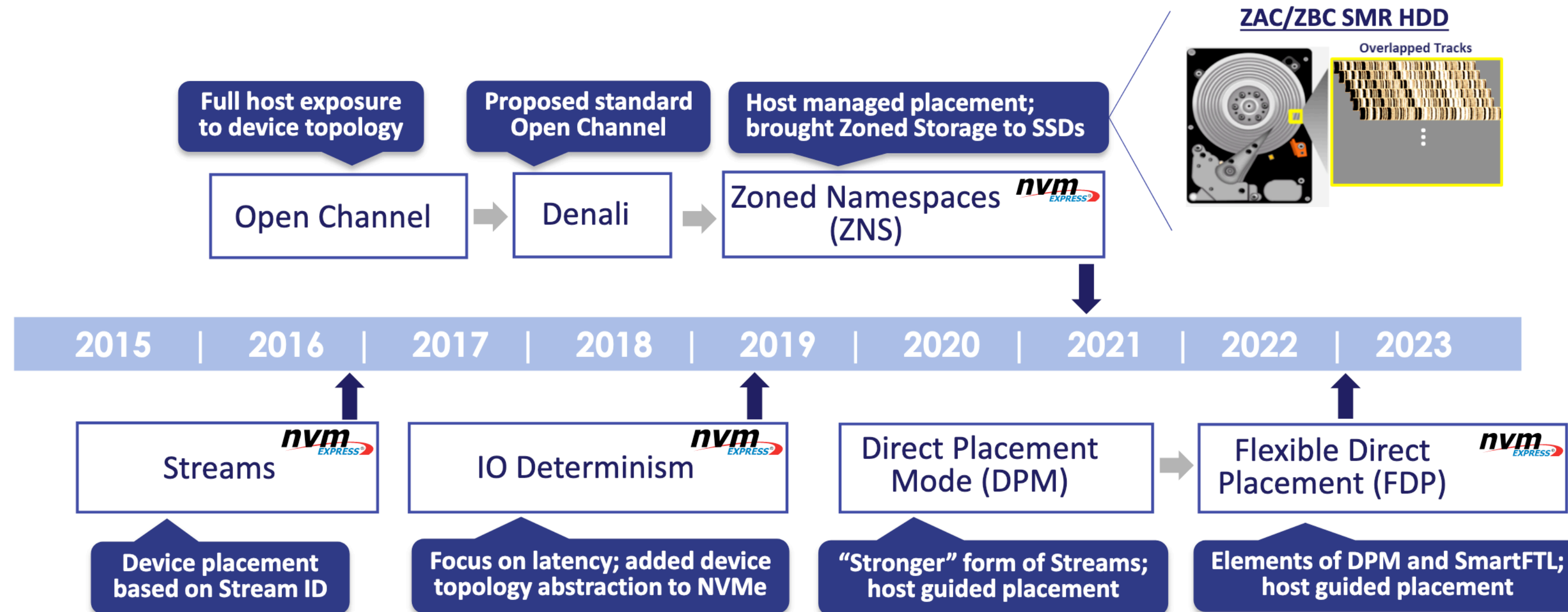
Latencies	
Page Read (flash-to-register)	40μs
Page Write (register-to-flash)	800μs
Page data transfer (via channel)	100μs
Block erase	2ms



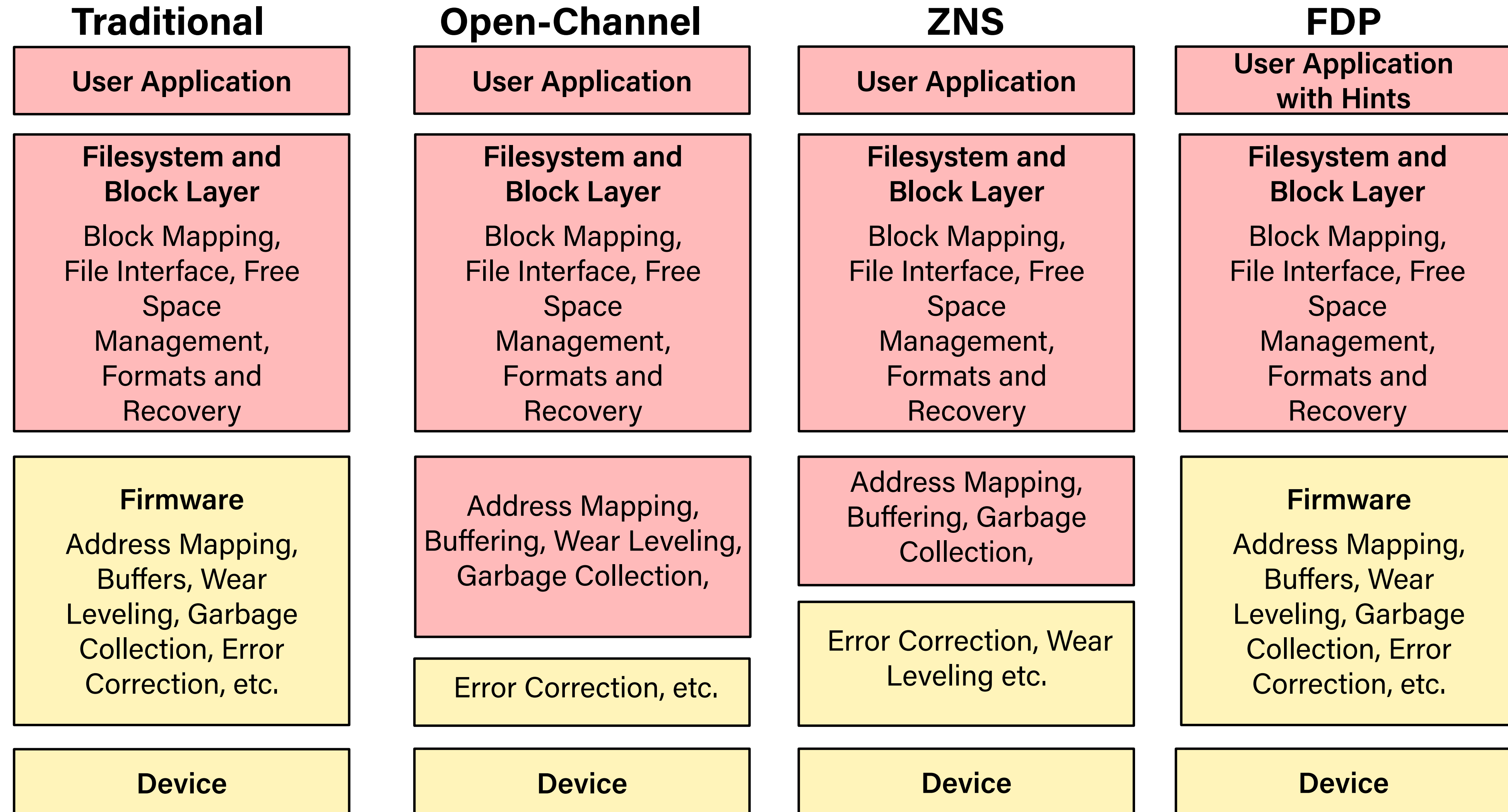
Tiny-Tail Flash: Near-Perfect Elimination of Garbage Collection Tail Latencies in NAND SSDs

Host-Device Communication

A Brief History of NVMe Data Placement Debates



Comparison (Host vs. Device)

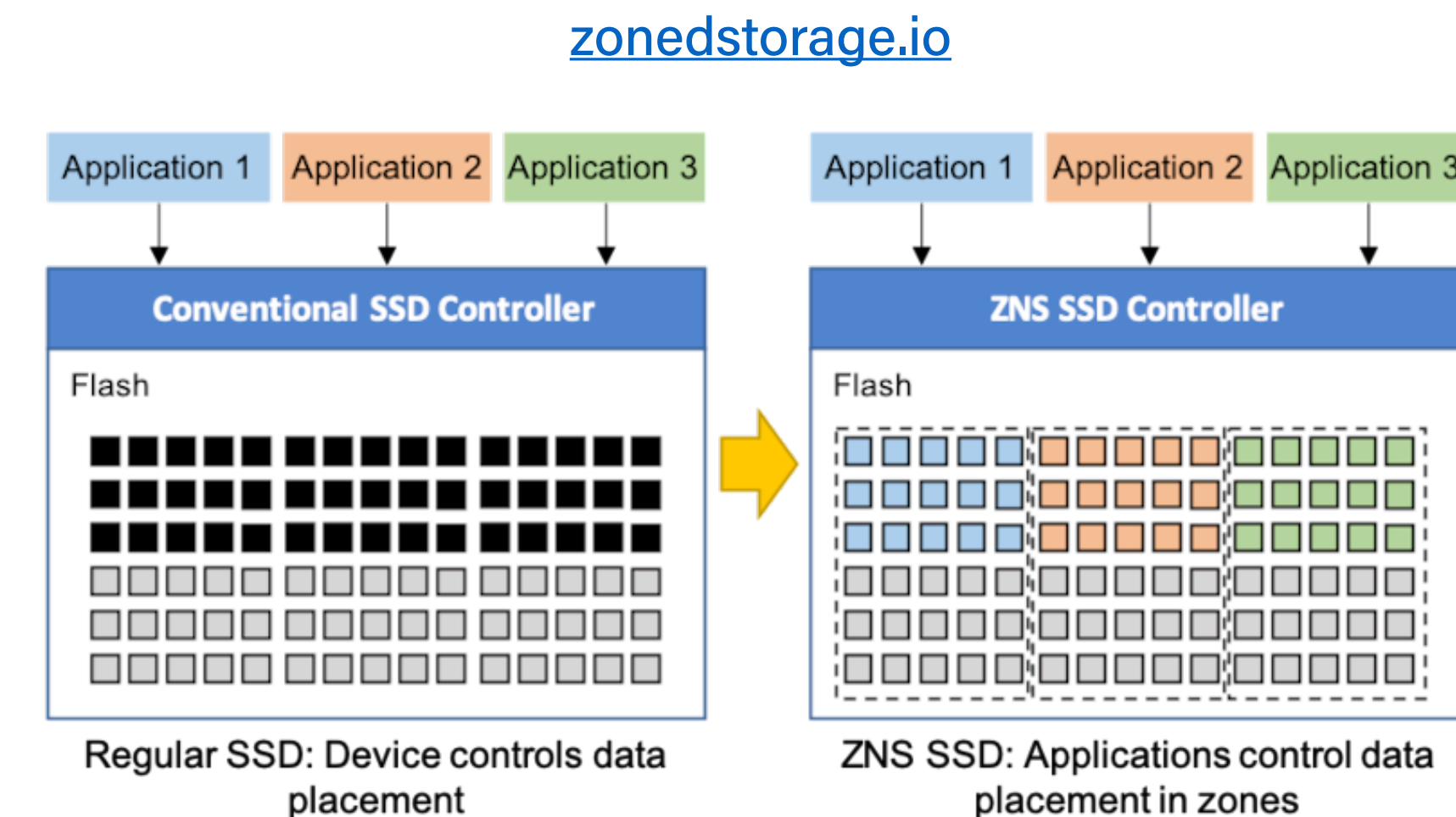


ZNS and FDP: an overview

- ▶ ZNS splits the address space in equal sized append-only regions called 'zones'
- ▶ **Simple device:** lightweight FTL, low overprovisioning, lower DRAM needed
- ▶ **Complex interface:** needs append-only writes, host performs garbage collection
- ▶ FDP routes data to the right 'reclaim units' depending on the hint
- ▶ **Complex device:** Full FTL, same overprovisioning and DRAM as standard SSDs
- ▶ **Simple interface:** hints are optional, backwards compatible with current SSDs

Zoned Namespace SSDs

- ▶ **Address space partitioned into 'zones'**
 - ▶ Equal sized append-only regions
- ▶ **Multi-tenancy**
 - ▶ Can pick zones for different writers
- ▶ **Data co-location**
 - ▶ Related data in the same zone
- ▶ **Garbage Collection**
 - ▶ By the host with 'Zone Reset'



Venue	# Pubs.	Simpl	Appr	Res	Orth
<i>FAST</i>	126	9	8	23	8
<i>OSDI</i>	164	3	0	4	0
<i>SOSP</i>	77	2	2	2	0
<i>MSST</i>	98	10	7	16	10
Total	465	24	17	45	18

Table 1: Impact of ZNS adoption on existing work on flash-based SSDs. Columns are counts. #Pubs. indicates the total publications in the venue over the last 5 years.

ZNS Architecture

- ▶ ZNS partitions the device into equal sized append-only regions
- ▶ Zone Size vs Zone Capacity
- ▶ Write pointer
 - ▶ Maintains the last written Logical Block Address

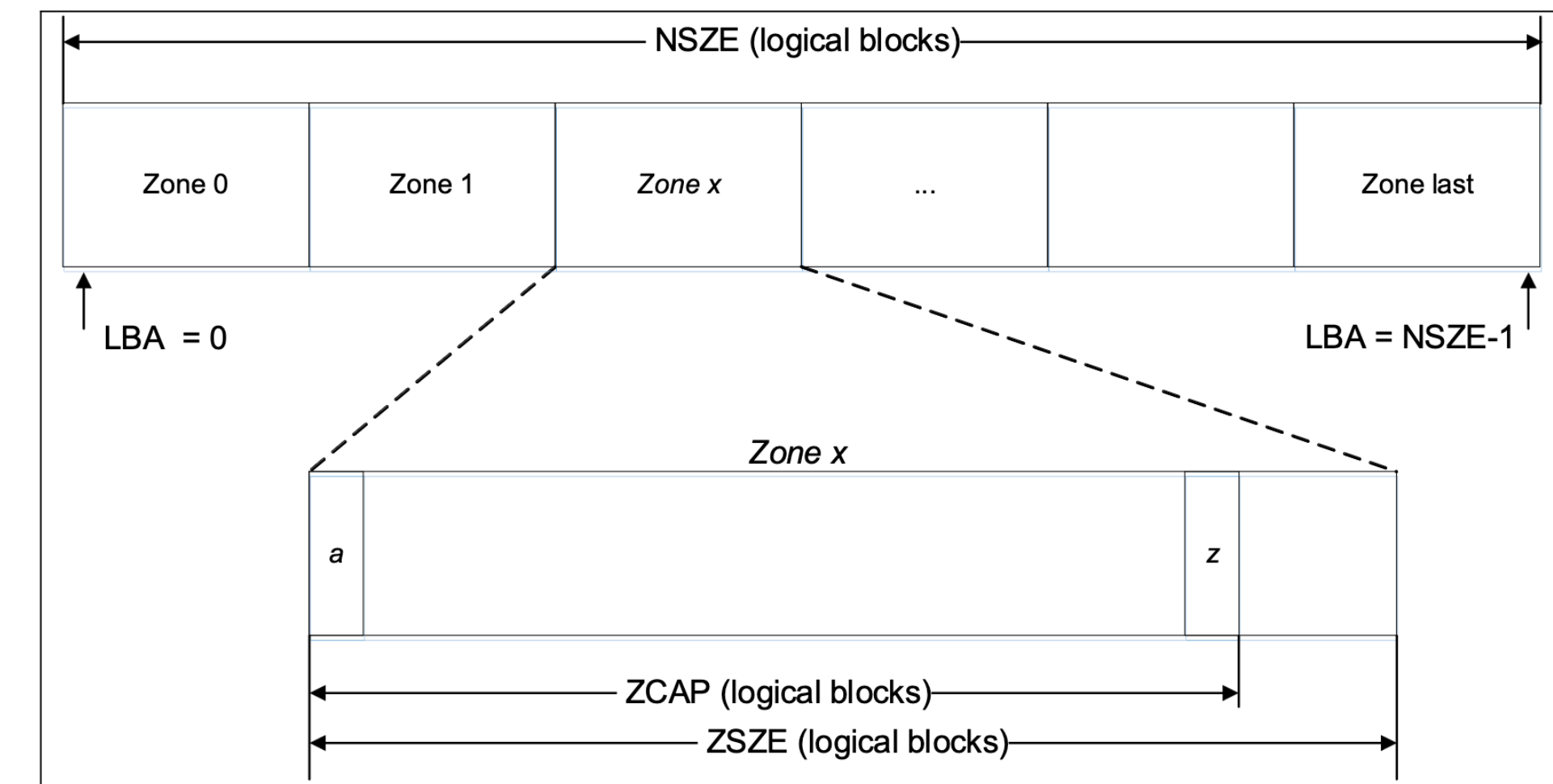
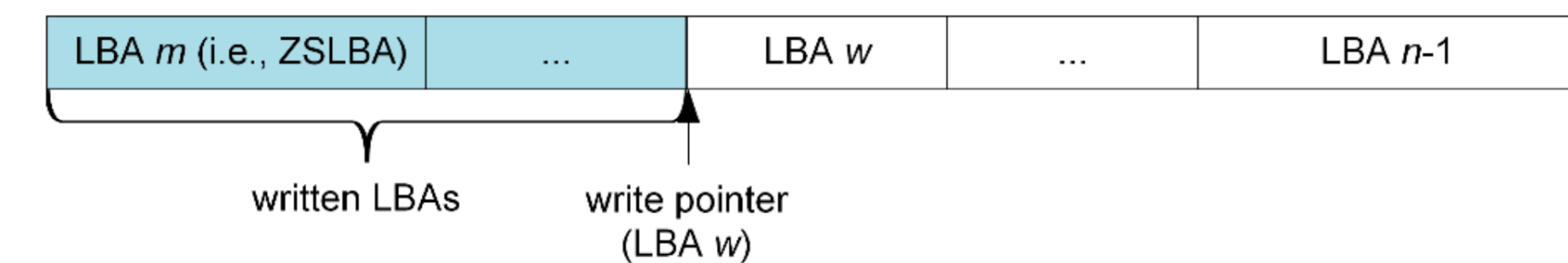
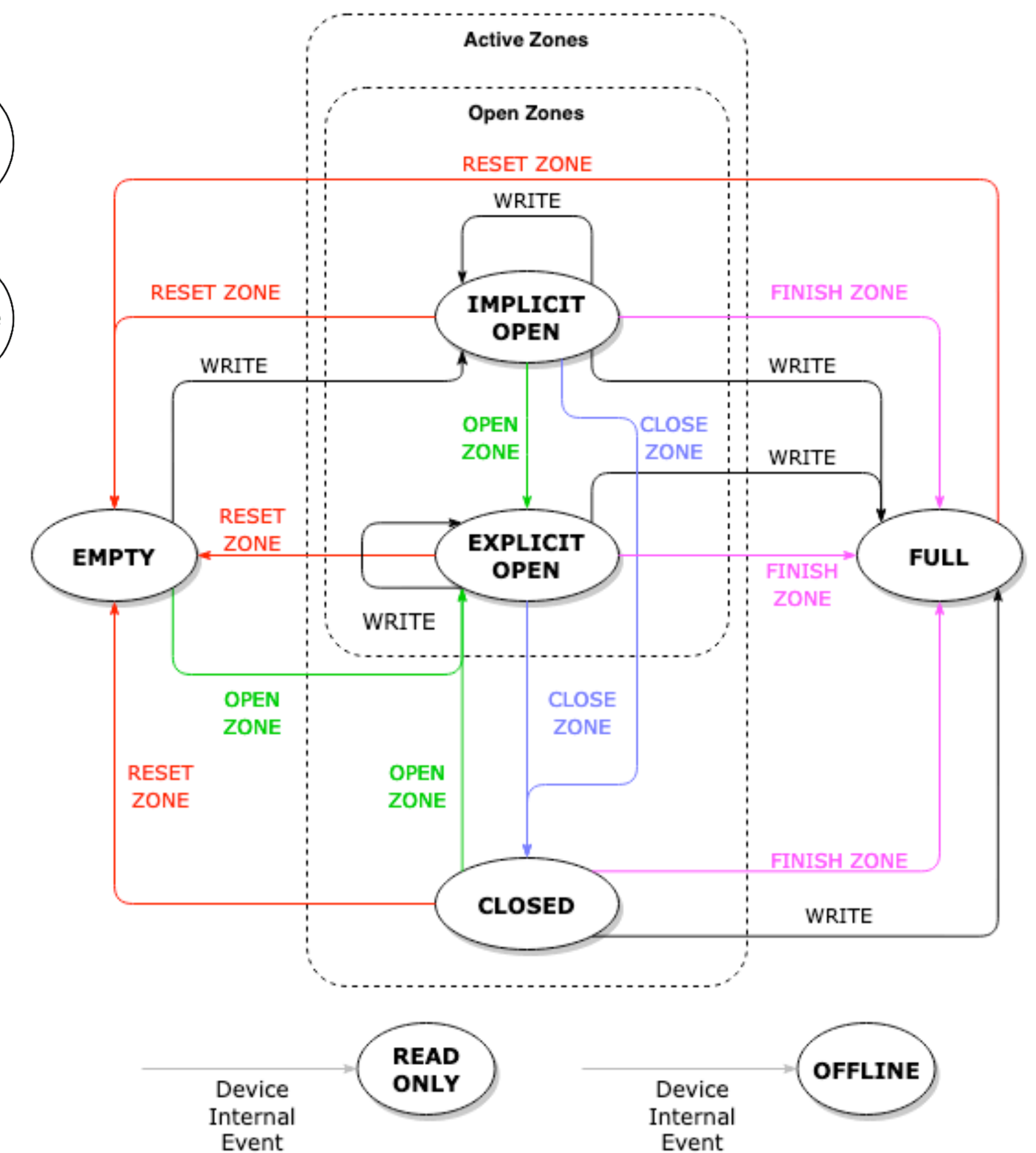


Figure 6: Write Pointer in a Partially Written Zone



zonedstorage.io

- ### Figure 7: Zone State Machine



<https://zonedstorage.io/docs/introduction/zoned-storage>

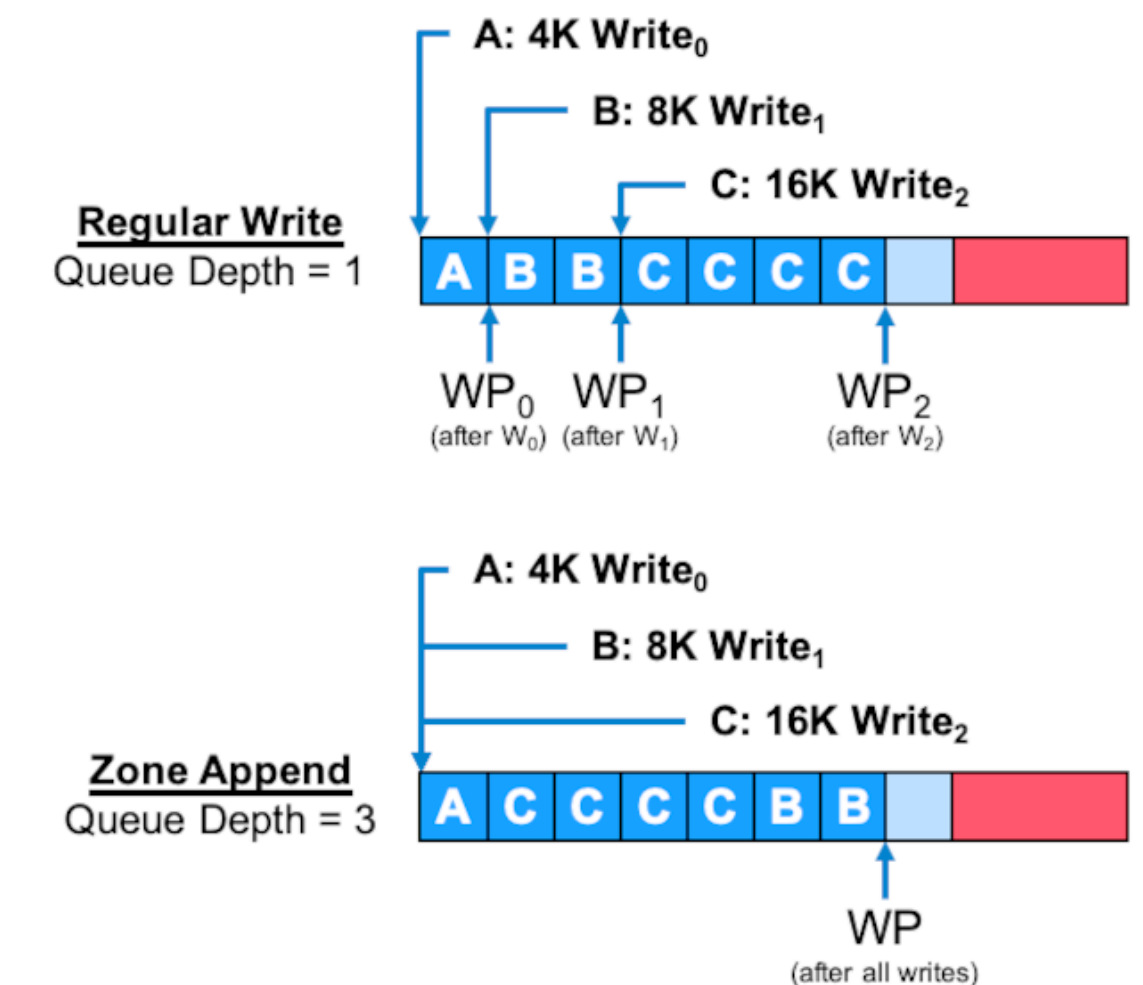
New Commands

▸ Zone Append

- Writes data to the tail of the log
- Returns the lowest logical block address
- Write ordering according to the device
- Solves contention over write pointer
- Allows >1 Queue Depth writes to the device

▸ Simple Copy

- Copy without involving host (like XCOPY)



ZNS: Programming

- ▶ **ZNS Unlocks new capabilities for applications**
 - ▶ Co-locate related data and reduce WAF
 - ▶ Improve performance with isolation
 - ▶ Host-managed GC
- ▶ **ZNS introduces the following challenges:**
 - ▶ Sequential writes only*
 - ▶ Active management of write resources
- ▶ **ZNS is well suited for log-structured applications**

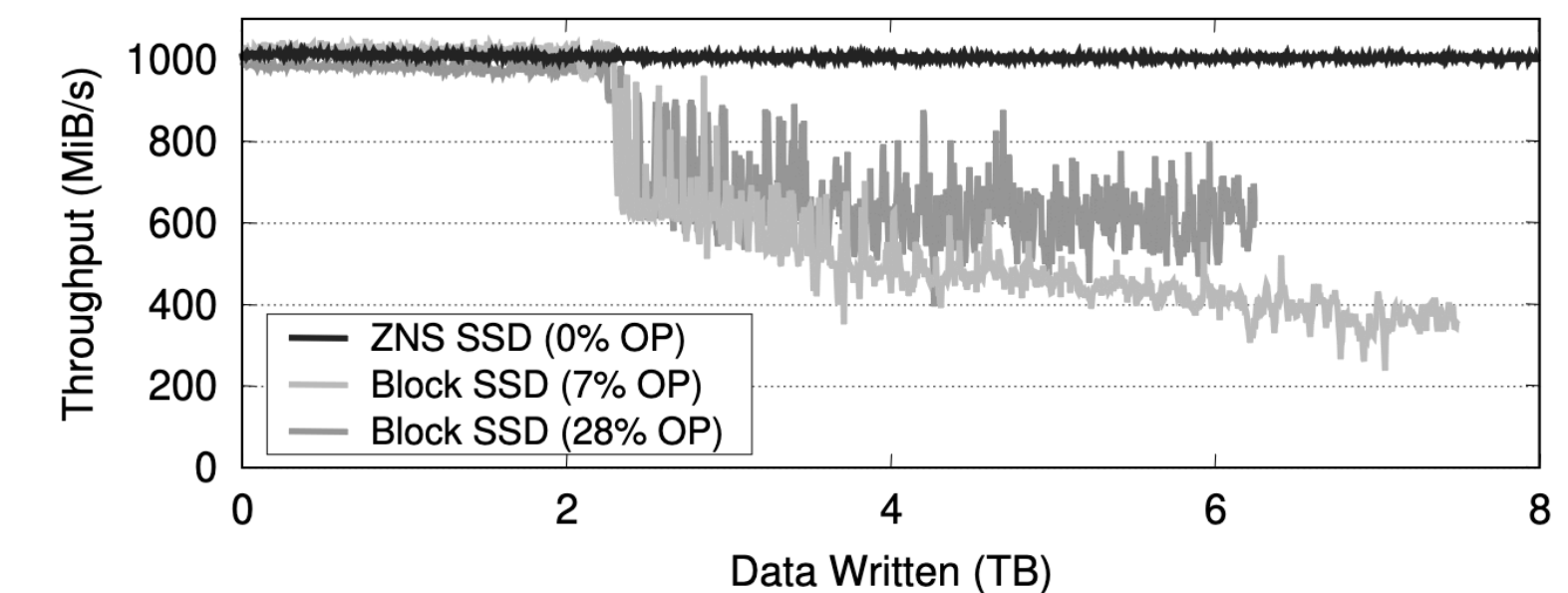


Figure 1: Throughput of a multi-threaded write workload that overwrites usable SSD capacity four times. The SSDs all have 2 TB raw media and share the same hardware platform.

Getting Started: Emulation

▸ null_blk

- Null block device
- Can be backed by memory
- Useful to test compatibility

```
# modprobe null_blk nr_devices=1 \  
    zoned=1 \  
    zone_nr_conv=4 \  
    zone_size=64 \  
...
```

▸ Qemu

- Can emulate a zoned device
- Fully compliant

```
# /usr/local/bin/qemu-system-x86_64 \  
...  
-device nvme,id=nvme0,serial=deadbeef,zoned.zasl=5 \  
-drive file=${znsimg},id=nvmezns0,format=raw,if=none \  
-device nvme-ns,drive=nvmezns0,bus=nvme0,nsid=1,\  
    logical_block_size=4096,\  
    physical_block_size=4096,zoned=true,zoned.zone_size=64M,zoned.\  
    zone_capacity=62M,zoned.max_open=16,zoned.max_active=32  
...
```

▸ ConfZNS

- Timing accurate configurable emulator
- Based on FEMU
- Allows to pick channels, dies, etc.

<https://github.com/DKU-StarLab/ConfZNS>

Libraries

<https://github.com/westerndigitalcorporation/libzbd>

▸ **libzbd**

- Provides support for zoned devices through the kernel interface using ioctls
- libzbd calls are for zoned operations while standard systemcalls are used for reads and writes

▸ **libnvme**

- Library for interacting with nvme devices
- nvme-cli uses this library
- Allows OS-mapping of NVMe commands like id-ns, id-ctrl, zone-append, mgmt-send, mgmt-receive, write etc.

xNVMe - Cross-platform

- ▶ **xNVMe** includes libxnvme_znd with the zoned command set
- ▶ Can be used with a variety of backends: Linux, AIO, SPDK, io_uring, freebsd, ioctl
- ▶ Supports all NVMe commands, similarly to libnvme

```
err = xnvme_buf_fill(buf, buf_nbytes,
                    cli->args.data_input ? cli->args.data_input : "anum");
if (err) {
    xnvme_cli_perr("xnvme_buf_fill()", err);
    goto exit;
}

xnvme_cli_pinf("Initializing async. context + alloc/init requests");
err = xnvme_queue_init(dev, qd, 0, &queue);
if (err) {
    xnvme_cli_perr("xnvme_queue_init()", err);
    goto exit;
}
xnvme_queue_set_cb(queue, cb_pool, &cb_args);

xnvme_cli_pinf("Append uri: '%s', qd: %d", cli->args.uri, qd);
xnvme_spec_znd_descr_pr(&zzone, XNVME_PR_DEF);

xnvme_cli_timer_start(cli);

payload = buf;
for (uint64_t curs = 0; (curs < zone.zcap) && !cb_args.ecount;) {
    struct xnvme_cmd_ctx *ctx = xnvme_queue_get_cmd_ctx(queue);

submit:
    err = xnvme_znd_append(ctx, nsid, zone.zslba, 0, payload, NULL);
    if (err) {
        goto submit;
    }
}
```



https://github.com/OpenMPDK/xNVMe/blob/main/lib/xnvme_znd.c

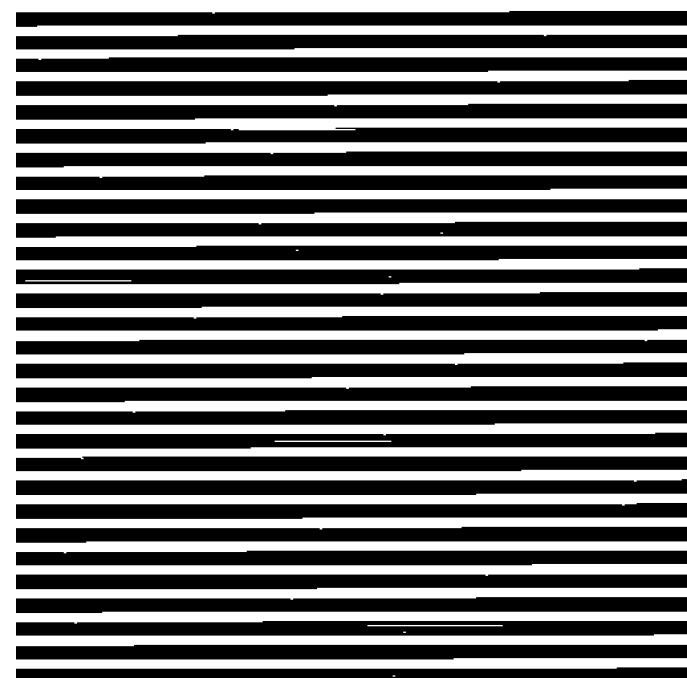
Filesystems: POSIX

▸ BTRFS

- Support for ZNS (experimental)
- Fixed extent sizes can cause issues
- No support for mixed groups, RAID or NOCOW

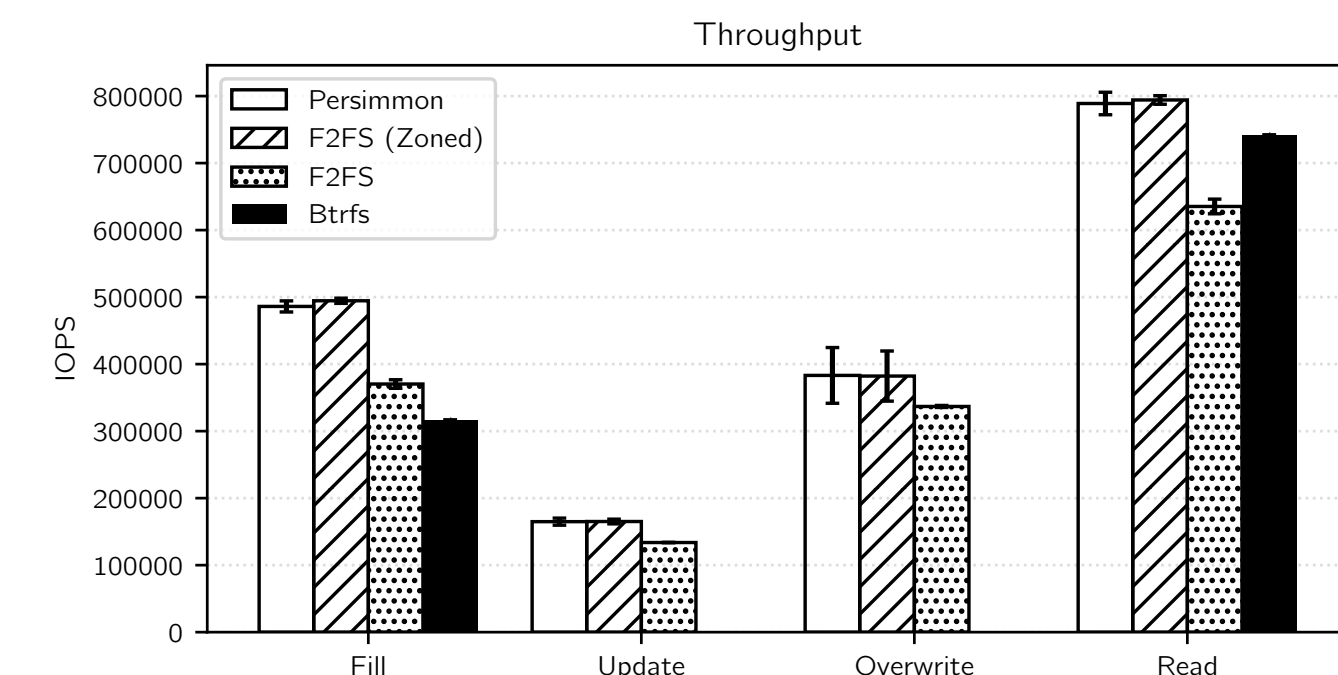
▸ F2FS

- Support for sequential zones in the data region (not for metadata or checkpoints)
- Needs to span multi-namespace or device



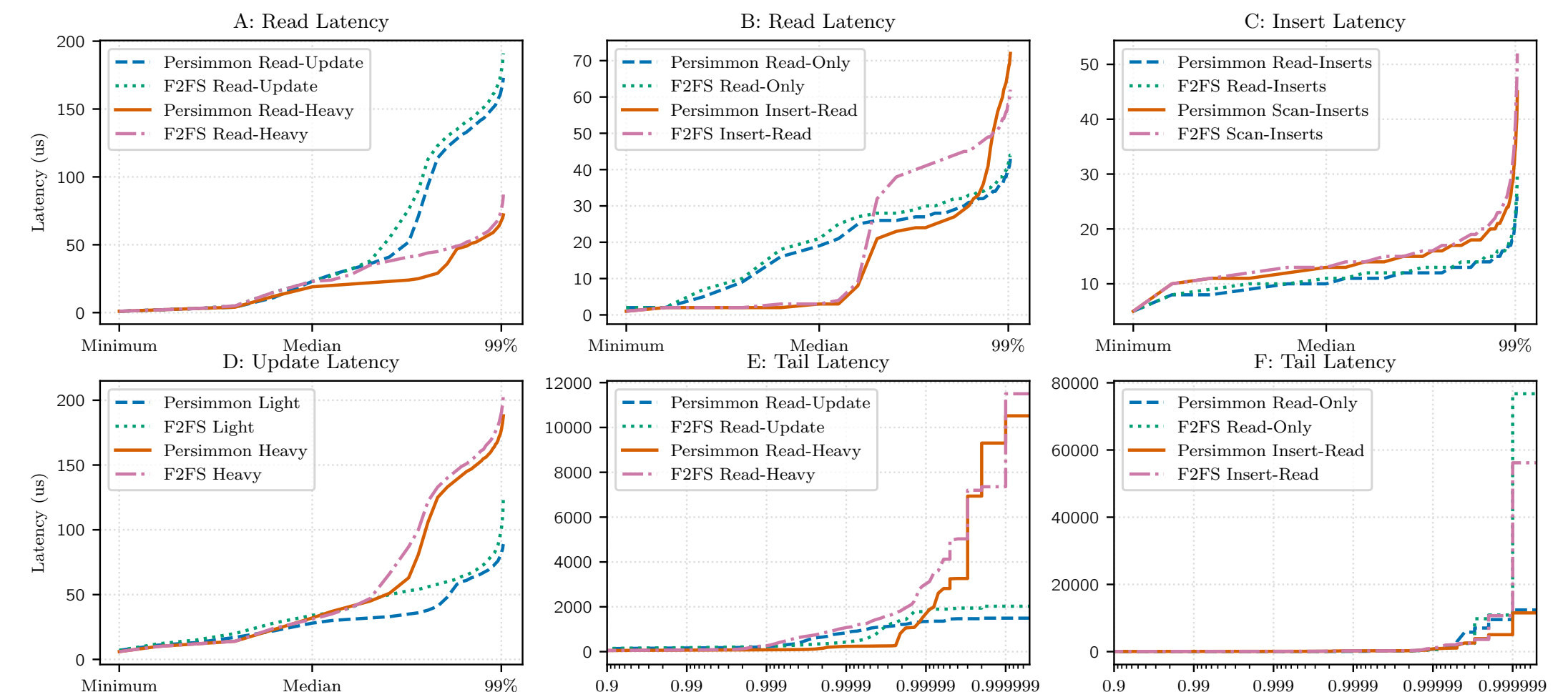
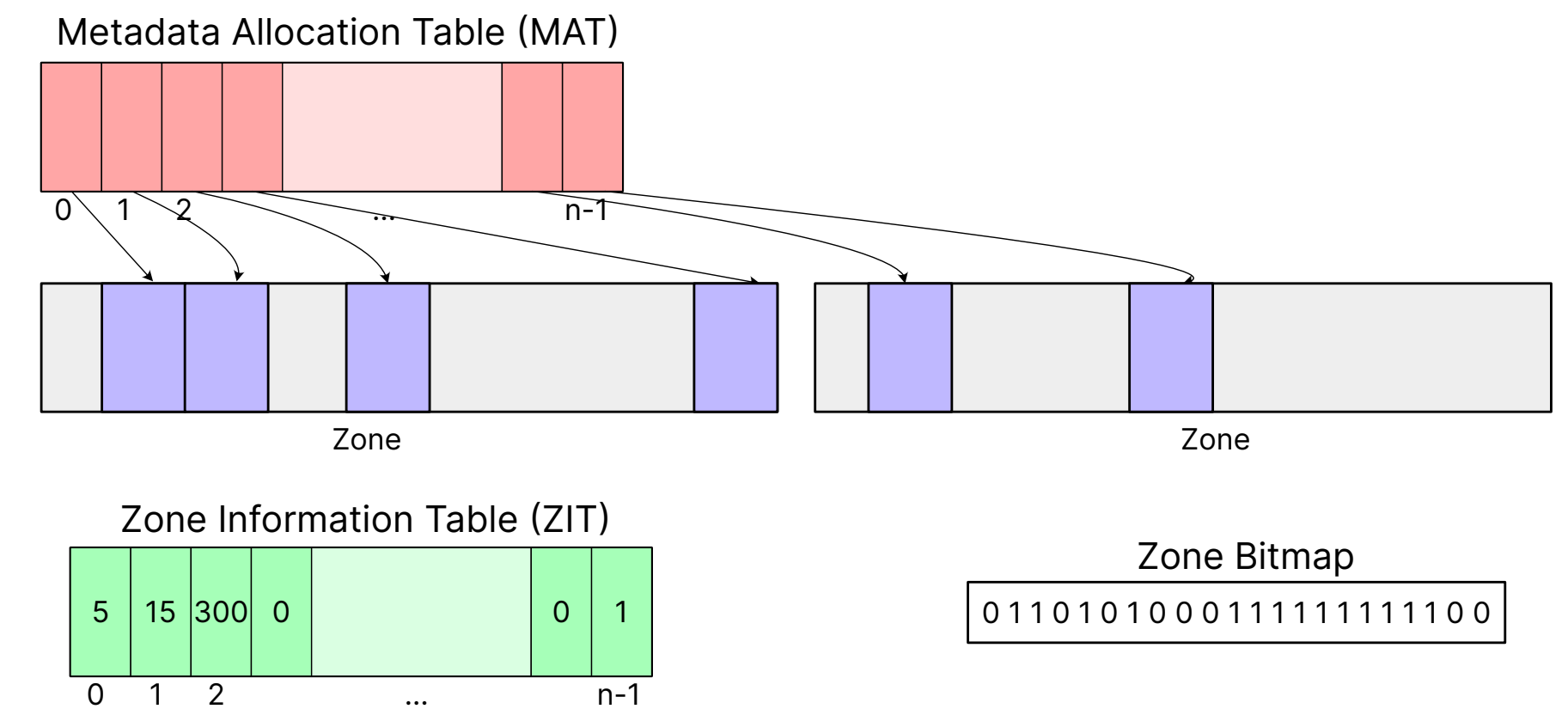
BTRFS' partially filled extents leave gaps
And cause space amplification

Relative performance, Persimmon is a fork
of F2FS with append-only metadata and
checkpoints (my work)



Persimmon

- ▶ Fork of F2FS with append-only metadata
- ▶ Does not require conventional zones
- ▶ Reduces tail latency, garbage collection overhead
- ▶ Offers better space utilization
- ▶ Maintains performance of F2FS



Filesystems: ZoneFS

- ▶ Exposes each zone as a file
- ▶ Block layer representation
- ▶ Write can append to the file
- ▶ Truncate Garbage collects the zone
- ▶ Great for experimentation!
- ▶ Writes are unbuffered
- ▶ Reads are unchanged
- ▶ Sysfs reports device details

```
#define _GNU_SOURCE
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

#define BLOCKSIZE 409600

int main() {
    char buf[BLOCKSIZE] = {1};
    for (int i = 0; i < BLOCKSIZE; i++) {
        buf[i] = 1;
    }
    int fd = open("/mnt/zonefs/seq/16", O_WRONLY | O_DIRECT | O_CREAT | O_APPEND,
        | 0666);
    if (fd == 0) {
        perror("Open");
    }
    int ret = write(fd, buf, BLOCKSIZE);
    if (ret < 0) {
        printf("Err: %s\n", strerror(errno));
        perror("File");
    }
    close(fd);
    return 0;
}
```

A simple C program to write 100 pages of "1" to zone 16 on a mounted zonefs filesystem.

```
dev + twilight ~/ exa /mnt/zonefs
seq
dev + twilight ~/ exa /mnt/zonefs/seq/
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29
```

```
dev + twilight ~/ ls /sys/fs/zonefs/dm-4/
max_active_seq_files max_wro_seq_files nr_active_seq_files nr_wro_seq_files
dev + twilight ~/ cat /sys/fs/zonefs/dm-4/nr_active_seq_files
0
```

Highlighted works: ZenFS

- ▶ ZenFS is a storage backend for RocksDB
- ▶ Can place files on raw zoned block devices or ZoneFS
- ▶ Does not support random writes and performs lazy GC

ZNS: Avoiding the Block Interface Tax for Flash-based SSDs

Matias Bjørling^{*}, Abutalib Aghayev[◇], Hans Holmberg^{*}, Aravind Ramesh^{*}, Damien Le Moal^{*},
Gregory R. Ganger[†], George Amvrosiadis[†]

^{*}Western Digital [◇]The Pennsylvania State University [†]Carnegie Mellon University

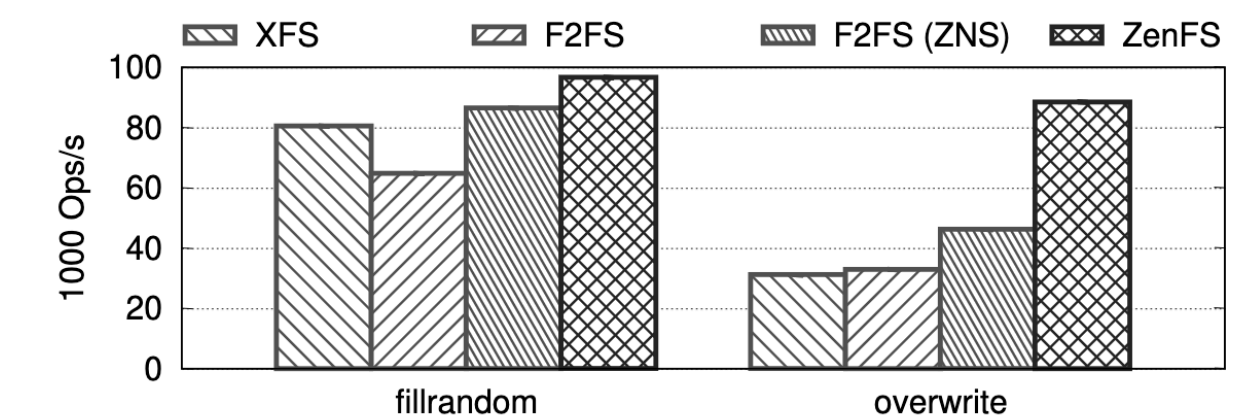
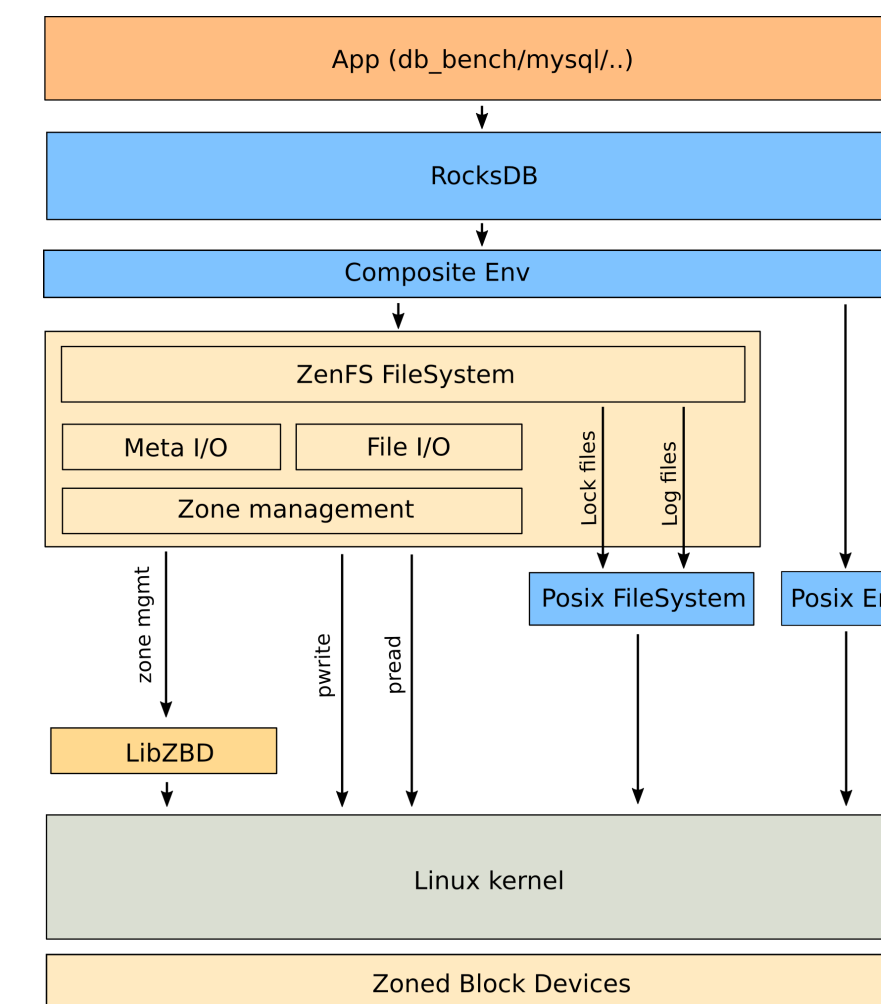


Figure 6: Throughput of RocksDB with write-heavy benchmarks—*fillrandom* followed by *overwrite* using the block-interface SSD with 28% OP and the ZNS SSD.

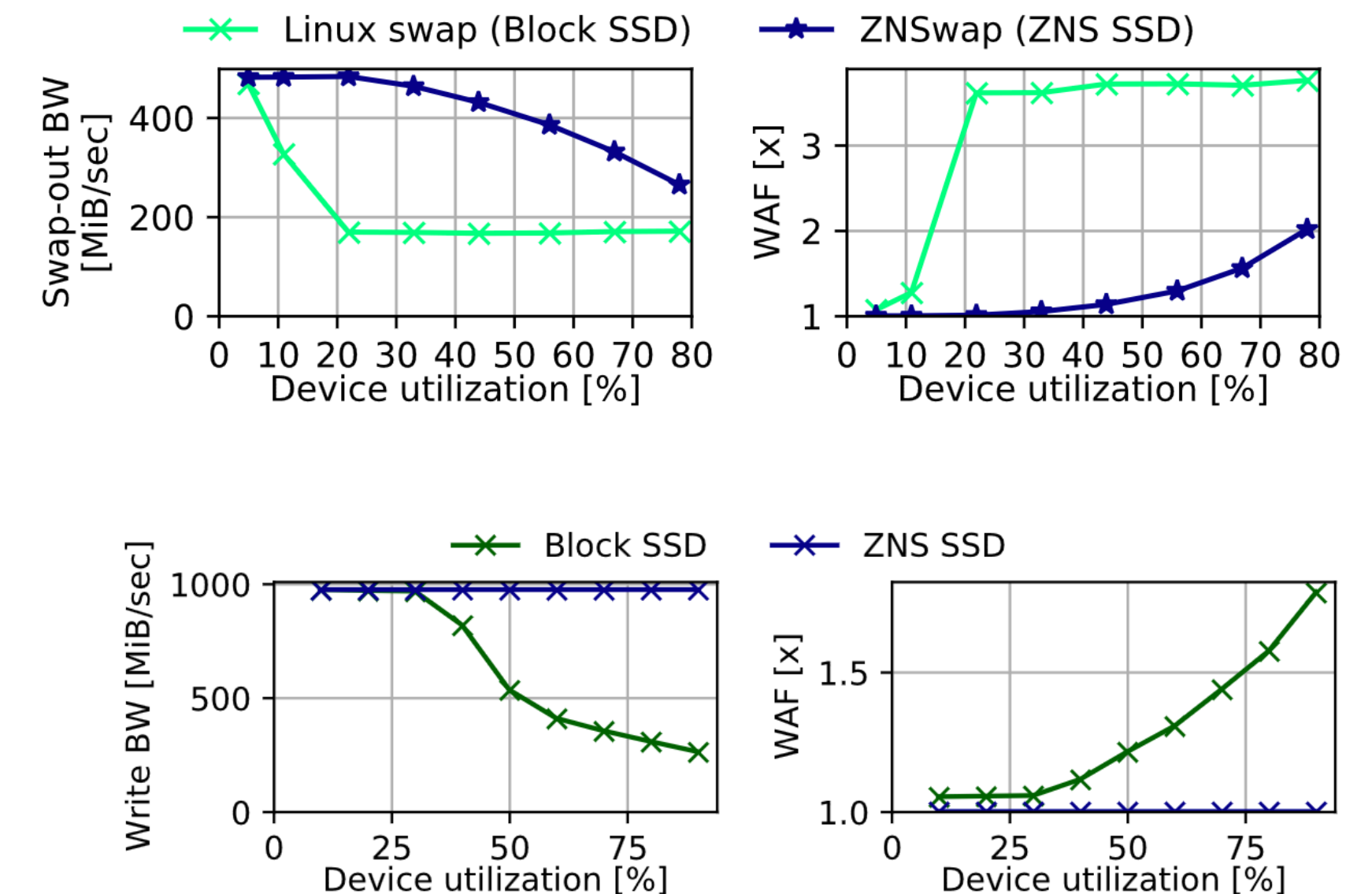
<https://github.com/westerndigitalcorporation/zenfs>

ZNSwap

- ▶ ZNS Optimized swap subsystem
 - ▶ GC co-designed with OS swap logic
 - ▶ Stable throughput
 - ▶ 10× lower p99 latency
 - ▶ 5× higher throughput

ZNSwap: un-Block your Swap

SHAI BERGMAN, Technion
NIKLAS CASSEL and MATIAS BJØRLING, Western Digital
MARK SILBERSTEIN, Technion



Motivation for FDP SSDs

- ZNS imposes a *restrictive* contract:
 - Can only write to a write pointer
 - Append-only writes, random writes-overwrites not possible
 - Host manages data placement and garbage collection
 - Requires a new interface and rewriting filesystems/applications
- FDP presents *backwards compatibility*
 - Backwards compatible interface to traditional SSDs
 - Optional hints that the device can leverage

Evolving NVMe Streams

- ▶ Stream SSDs (2016) added support for multiple streams of different temperatures
- ▶ Streams have limitations
 - ▶ Eg. SHORT is application and workload dependent
- ▶ FDP use arbitrary Reclaim Groups as hints

RWH_WRITE_LIFE_SHORT	Data written with this flag is expected to have a high overwrite rate, or life time.
RWH_WRITE_LIFE_MEDIUM	Longer life time than SHORT
RWH_WRITE_LIFE_LONG	Longer life time than MEDIUM
RWH_WRITE_LIFE_EXTREME	Longer life time than LONG

NVMe FDP

- Google and Meta merged their suggestions for Smart FTL, and Direct Placement Mode into FDP
- **On-device:**
 - Placement on the indicated super block
 - Management of super blocks
- **At host:**
 - Addition of a placement identifier
- No other functionality is changed

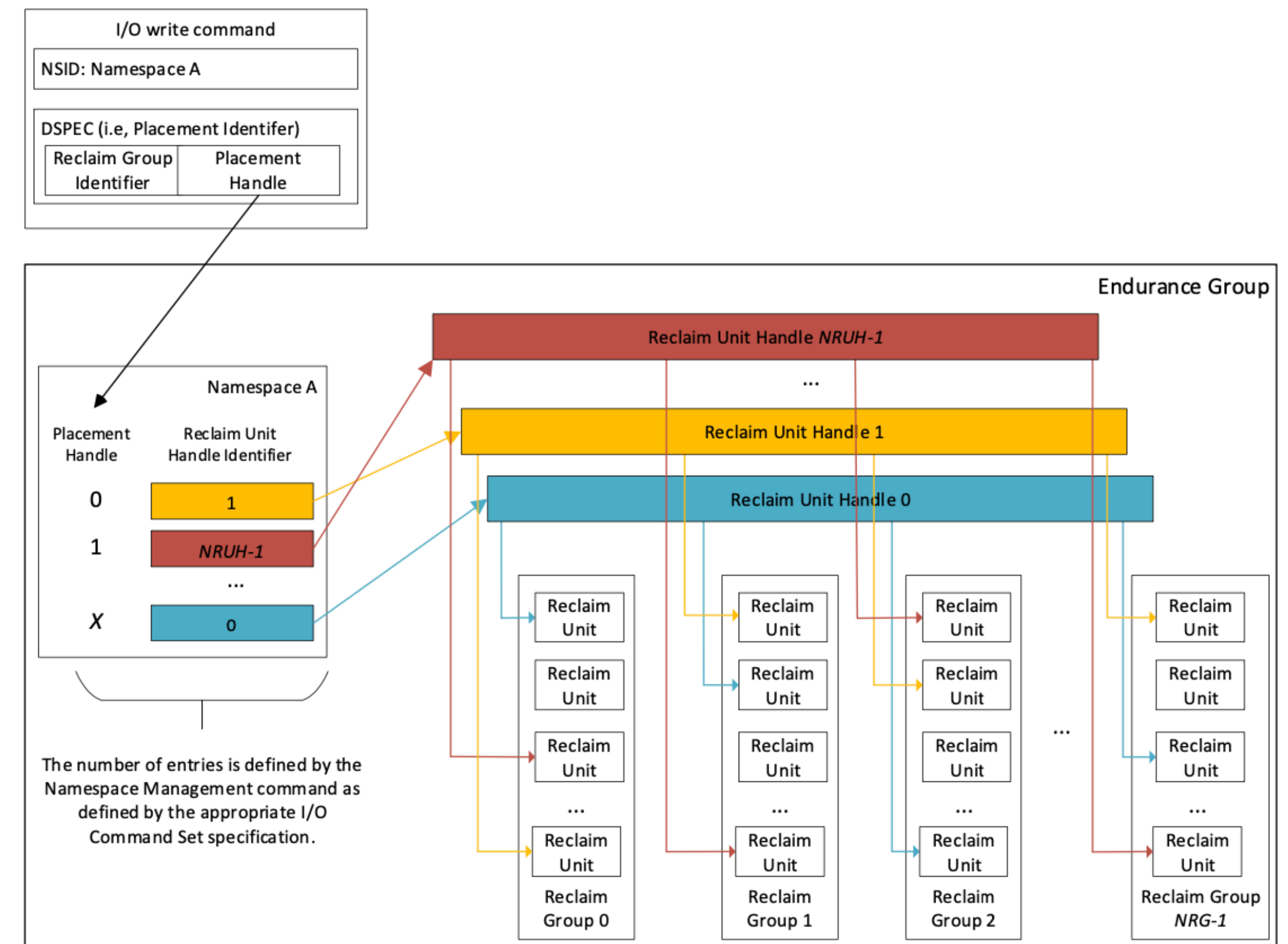
Core Concepts

► Grouping

- Reclaim Groups (RGs)
- Reclaim Units (RUs)
- Reclaim Unit Handles (RUHs)

► Placement

- Placement Identifier
- Placement Handle



Bits	Description
31:24	Reserved
23:20	Directive Type (DTYPE): Specifies the Directive Type associated with the Directive Specific field (refer to the Directives section in the NVM Express Base Specification).
19:16	Reserved
15:00	Number of Logical Blocks (NLB): This field specifies the number of logical blocks to become uncorrectable. This is a 0's based value.

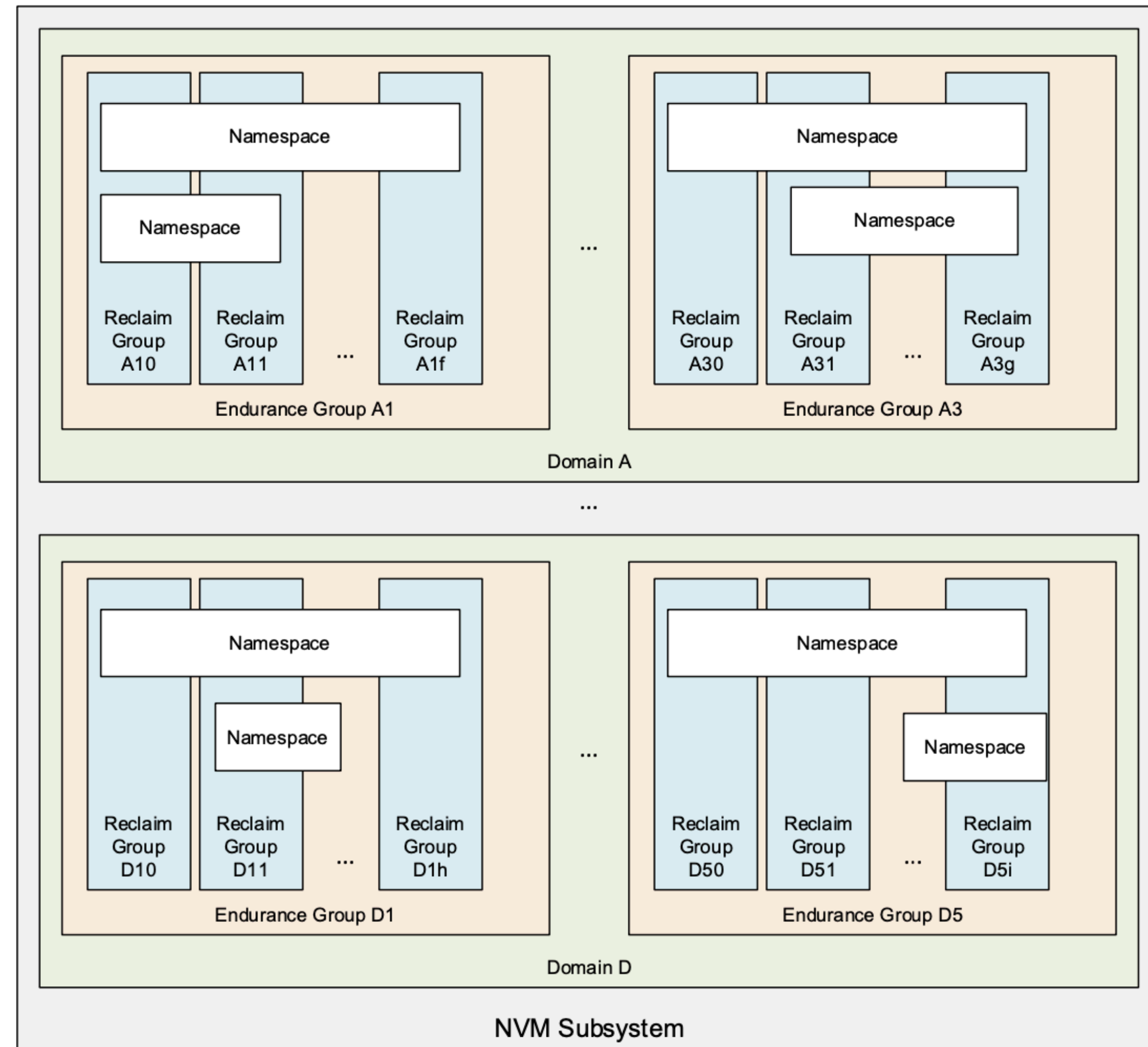
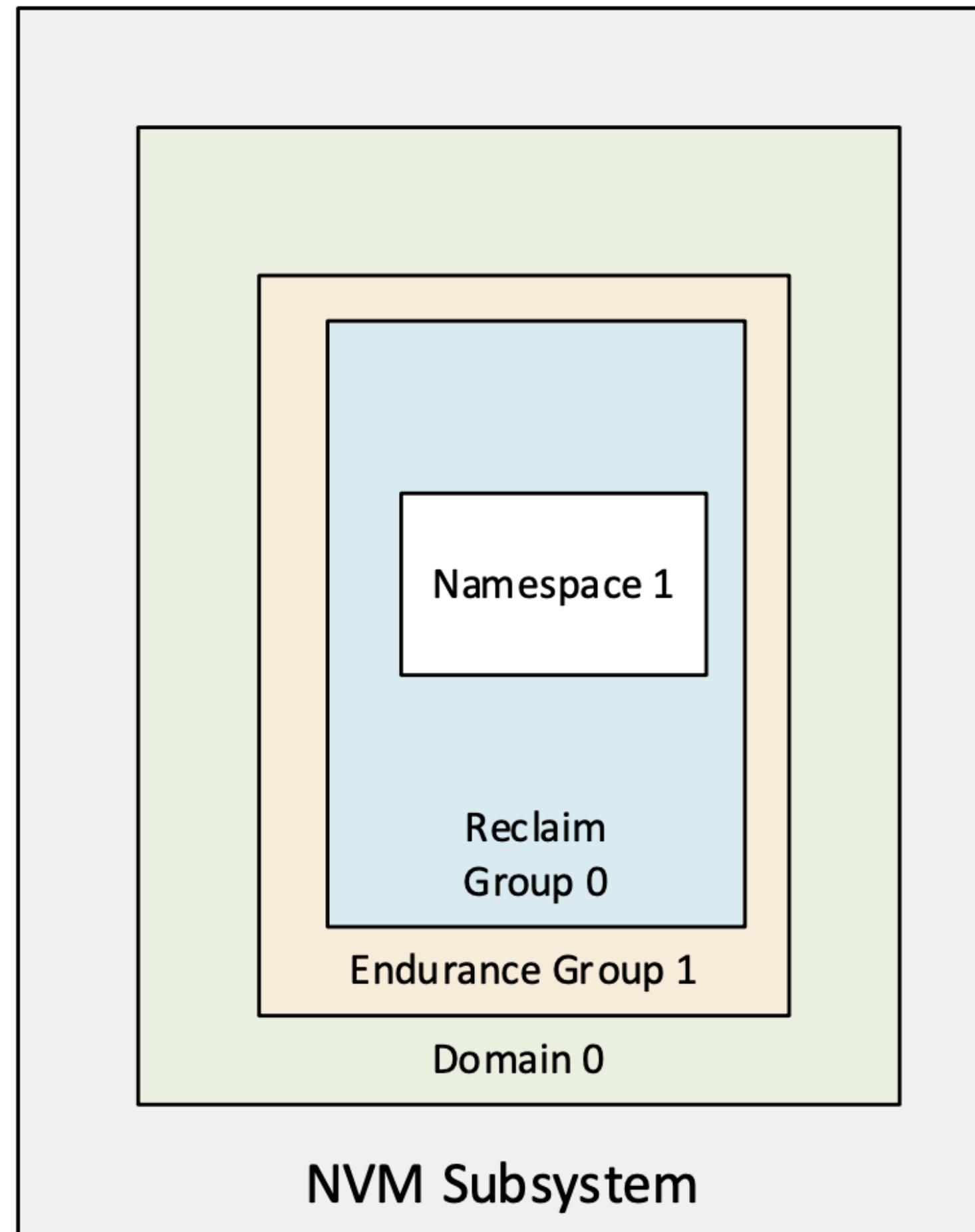
FDP Benefits

- ▶ **Without application support:**
 - ▶ Writes could still be tagged into groups per-application
 - ▶ Gets partial performance isolation
 - ▶ Gets most, but not all benefits
- ▶ **With explicit application support:**
 - ▶ Writes can be grouped by delete characteristics
 - ▶ WAF=1 possible with good grouping

	Write Amplification		
	Conv SSD	FDP SSD	%
FIO	1.63	1.02	-37%
JEDEC (512B)	3.95	3.8	-4%
JEDEC (4KB)	3.73	3.48	-7%
Full Bechmarks	In Progress		

From OCP 2023 Storage talks
<https://www.opencompute.org/events/past-events/2023-ocp-storage-tech-talks>

Simple vs Complex Hierarchy



Selecting Reclaim Groups

- ▶ FDP can be used as a placement handle to indicate reclaim unit
- ▶ Placement handle is an integer hint (unsigned 16 bit)

```
[global]
rw=randwrite
size=2M
iodepth=1
bs=4K
thread=1
fdp=1
fdp_pli=4,5
```

A sample fio spec using pli 4 and 5

```
int nvme_ns_write_uncorrectable(nvme_ns_t n, off_t offset, size_t count)
{
    struct nvme_io_args args = {
        .args_size = sizeof(args),
        .fd = nvme_ns_get_fd(n),
        .nsid = nvme_ns_get_nsid(n),
        .control = 0,
        .dsm = 0,
        .dspec = 0,
        .reftag = 0,
        .apptag = 0,
        .appmask = 0,
        .storage_tag = 0,
        .data_len = 0,
        .data = NULL,
        .metadata_len = 0,
        .metadata = NULL,
        .timeout = NVME_DEFAULT_IOCTL_TIMEOUT,
        .result = NULL,
    };
};
```

Possible Hint Interfaces

▸ RWH_Lifetime hints

- Used by the stream interface
- The third parameter can be an integer allowing the host to set a write hint for a particular file descriptor

▸ `fadvise(2)`

- Allows application to indicate to the kernel the expected workload
- Typically used to perform cache-based optimization
- Could be utilized for RUHs

```
fcntl(fd, F_SET_RW_HINT, RW_WRITE_LIFE_MEDIUM);
```

```
fadvise64(15, 0, 0, POSIX_FADV_RANDOM)
```

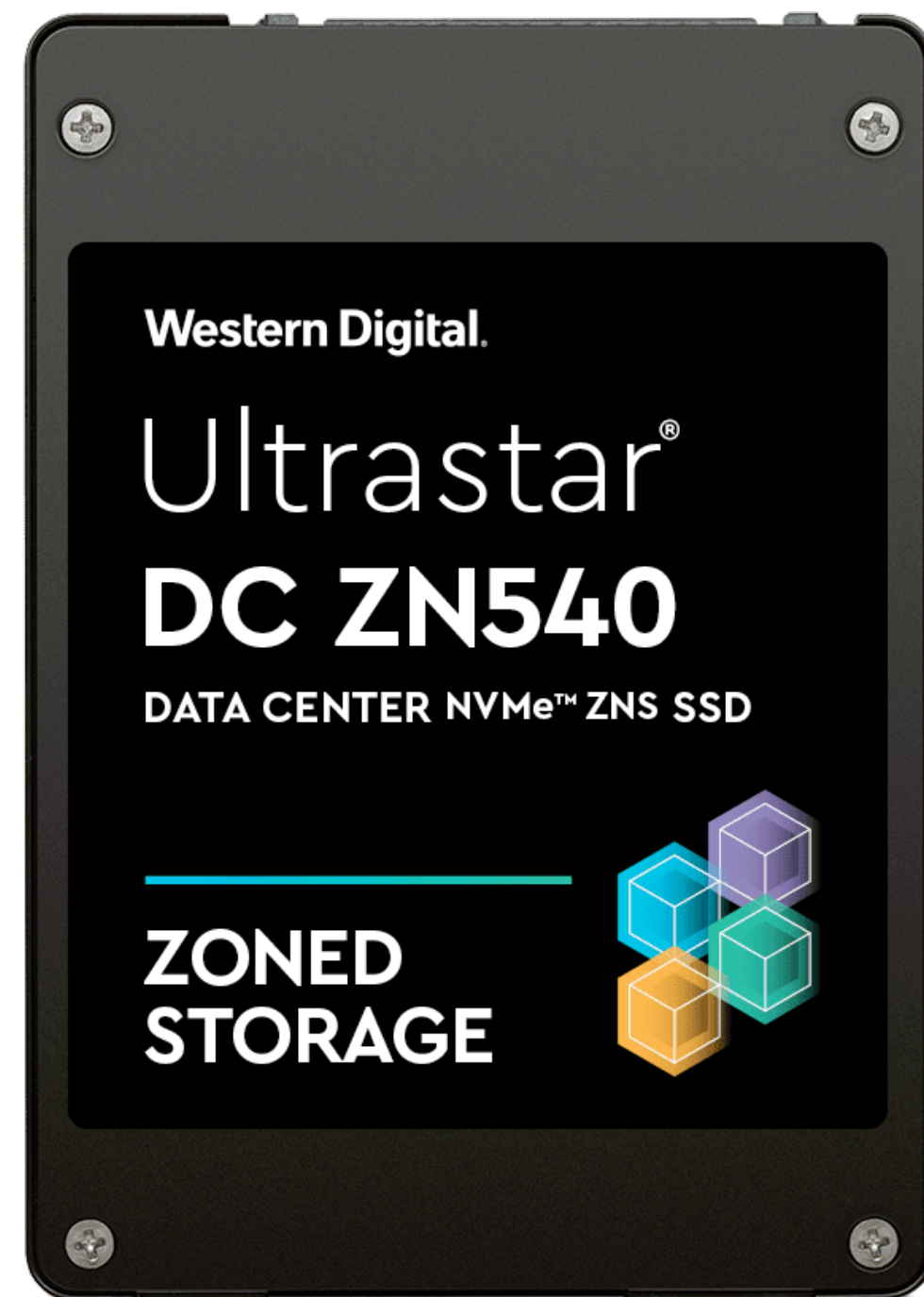
Hardware and Emulation

- ▶ FDP is ratified as a part of NVMe 2.0 spec (TP 4146)
- ▶ Linux Kernel Support since 5.19
- ▶ xNVMe support
- ▶ QEMU 8.0 supports FDP emulation for 1 endurance group
- ▶ fio supports FDP
- ▶ nvme-cli has fdp-specific commands

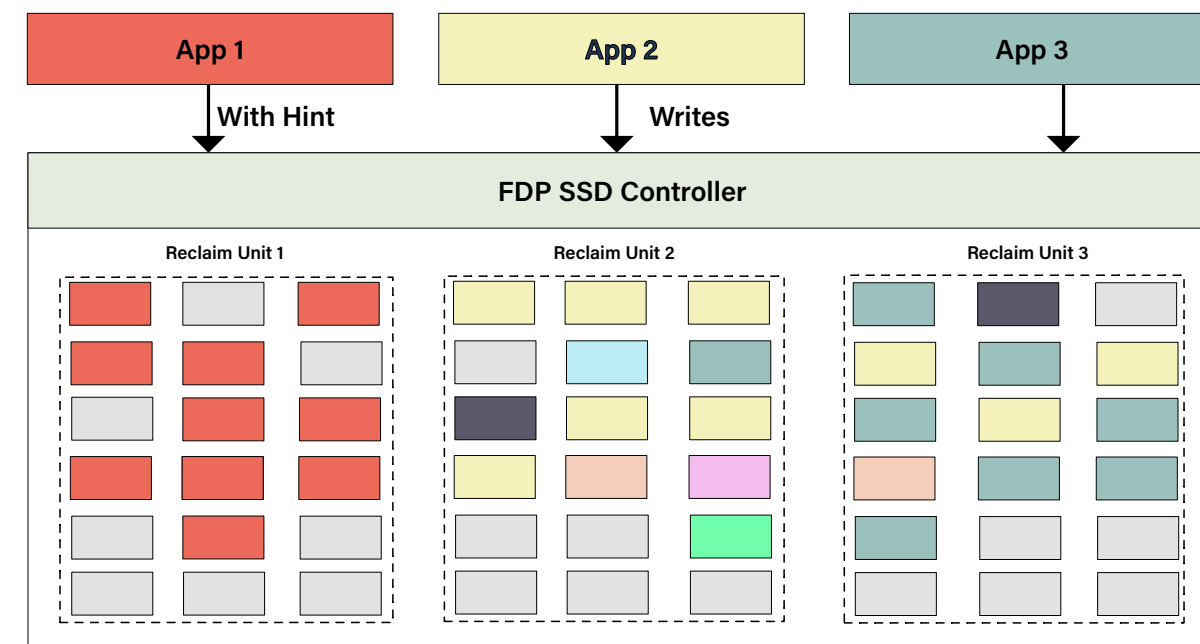
```
-device nvme-subsys,id=nvme-  
subsys-0,nqn=subsys0,fdp=on,fdp.nruh=16
```

Setting up a FDP device in QEMU

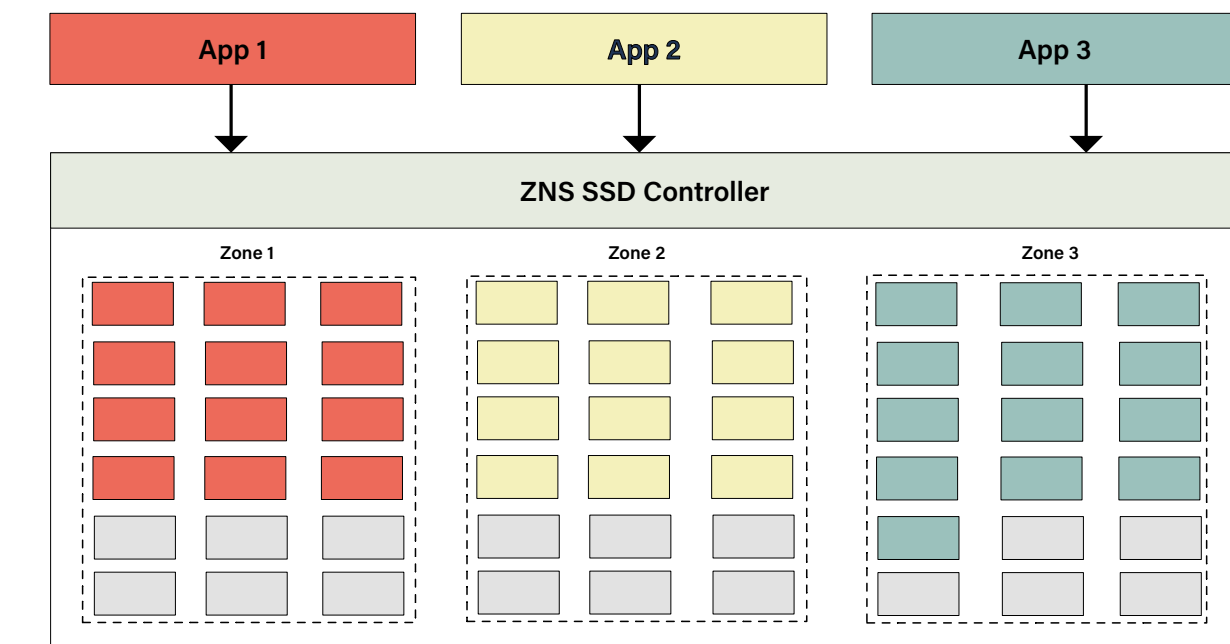
Drives ZNS and FDP



The fork: ZNS and FDP



FDP	ZNS
Standard Interface	Append-only Interface
Sequential, random, overwrite	Sequential writes only
Standard OP	0% OP
Standard DRAM	Reduced DRAM
Host changes optional	Host changes required
WAF 1 possible	WAF = 1
Host provides hints	Hosts picks zone
On-device GC	Host-performed GC
Reclaim Unit	Zone
Full FTL	Simple FTL
Stateless	Stateful
Static resource allocation	Dynamic resource allocation

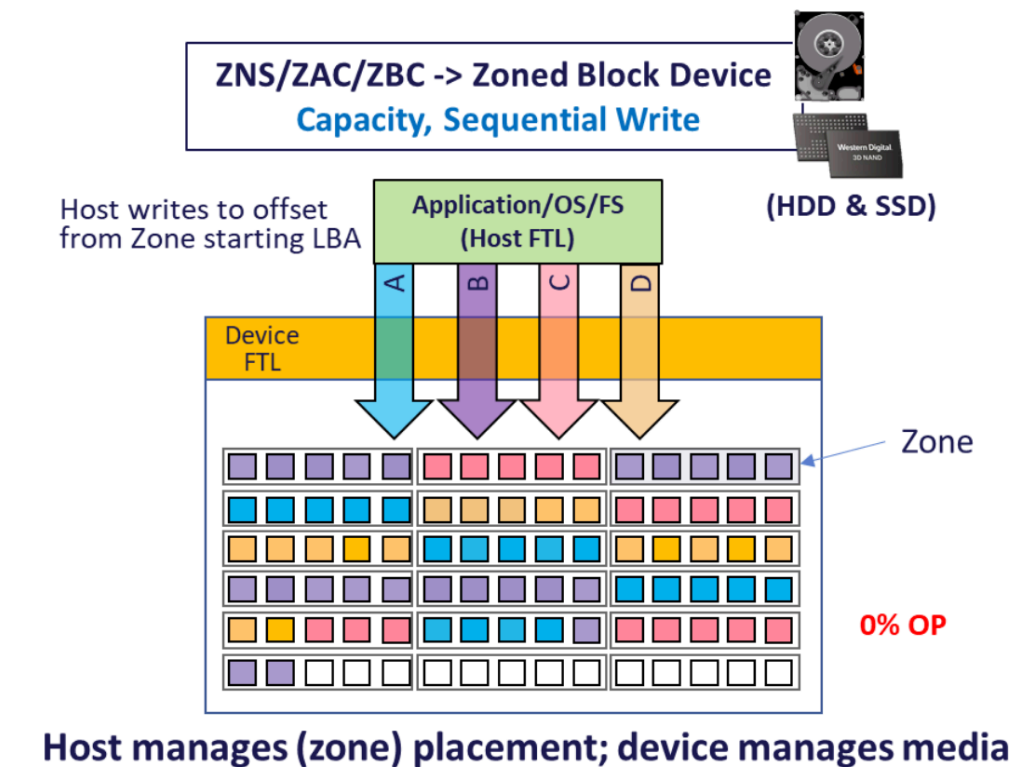
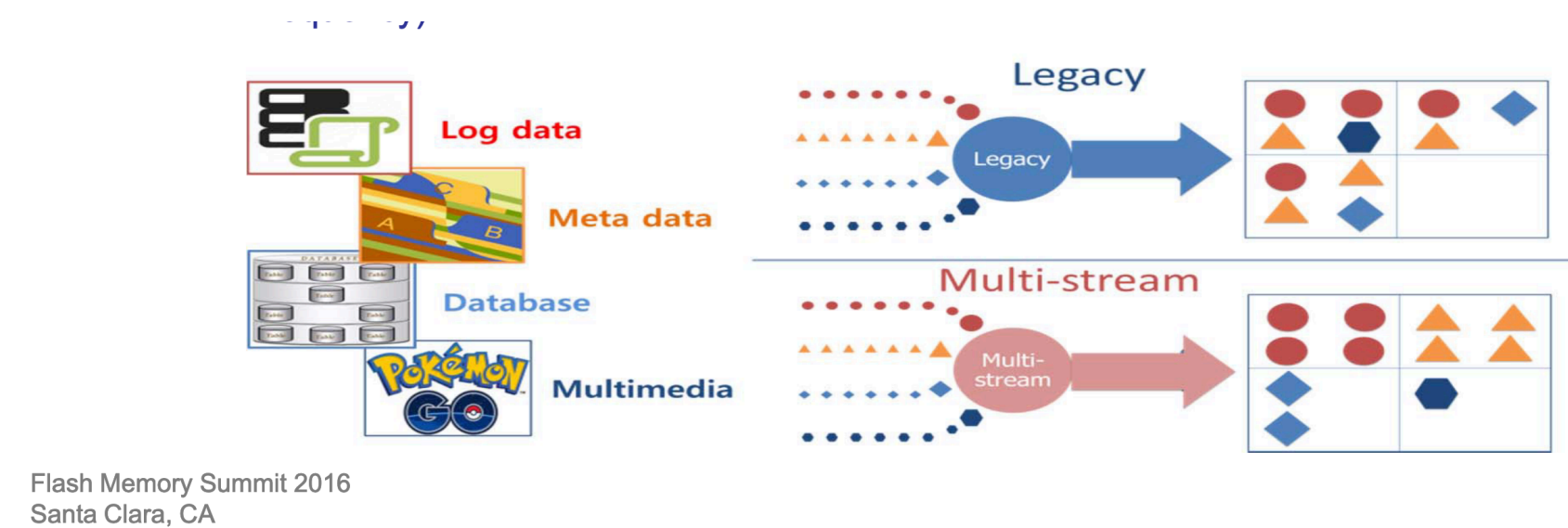


Open Problems: Hint Generation

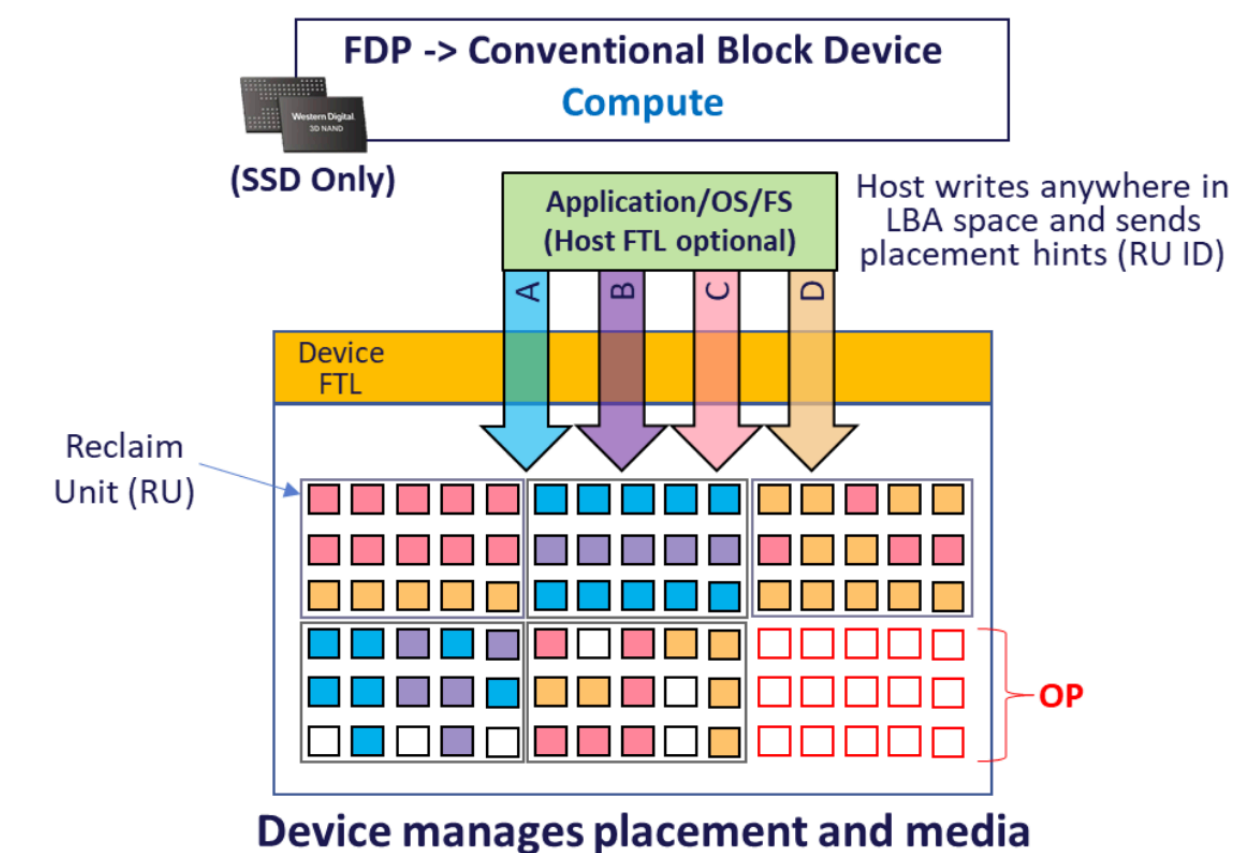
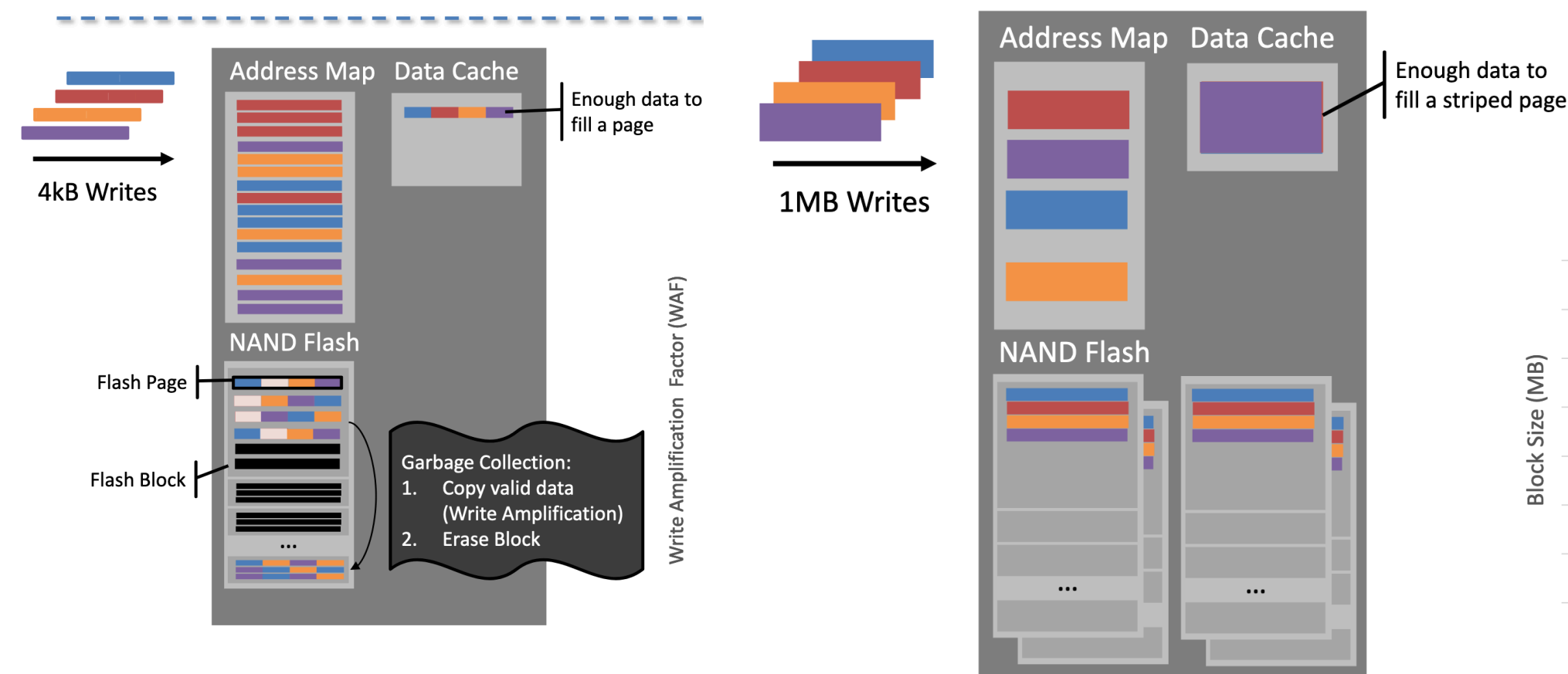
History of host-managed SSD

FMS 2019 (ZNS)

FMS 2016 (Stream)



SSD Architecture

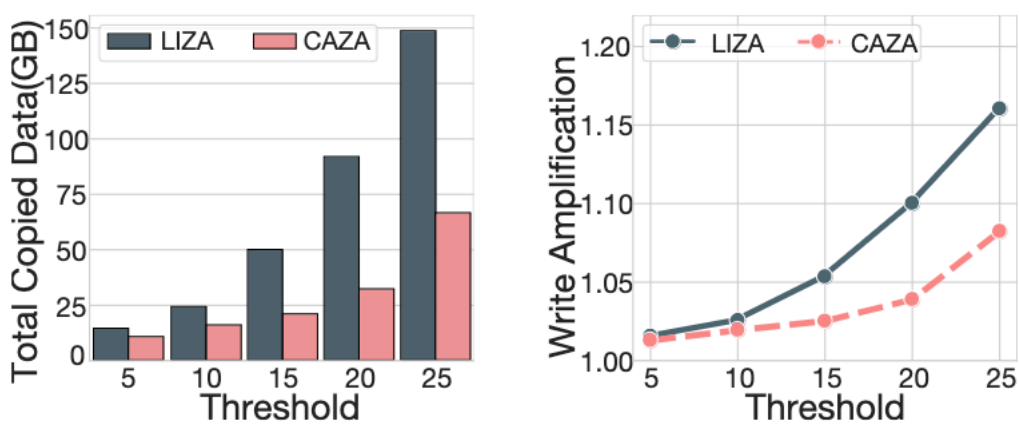


OCP 2018 (Denali)

FMS 2022 (FDP)

Existing Work

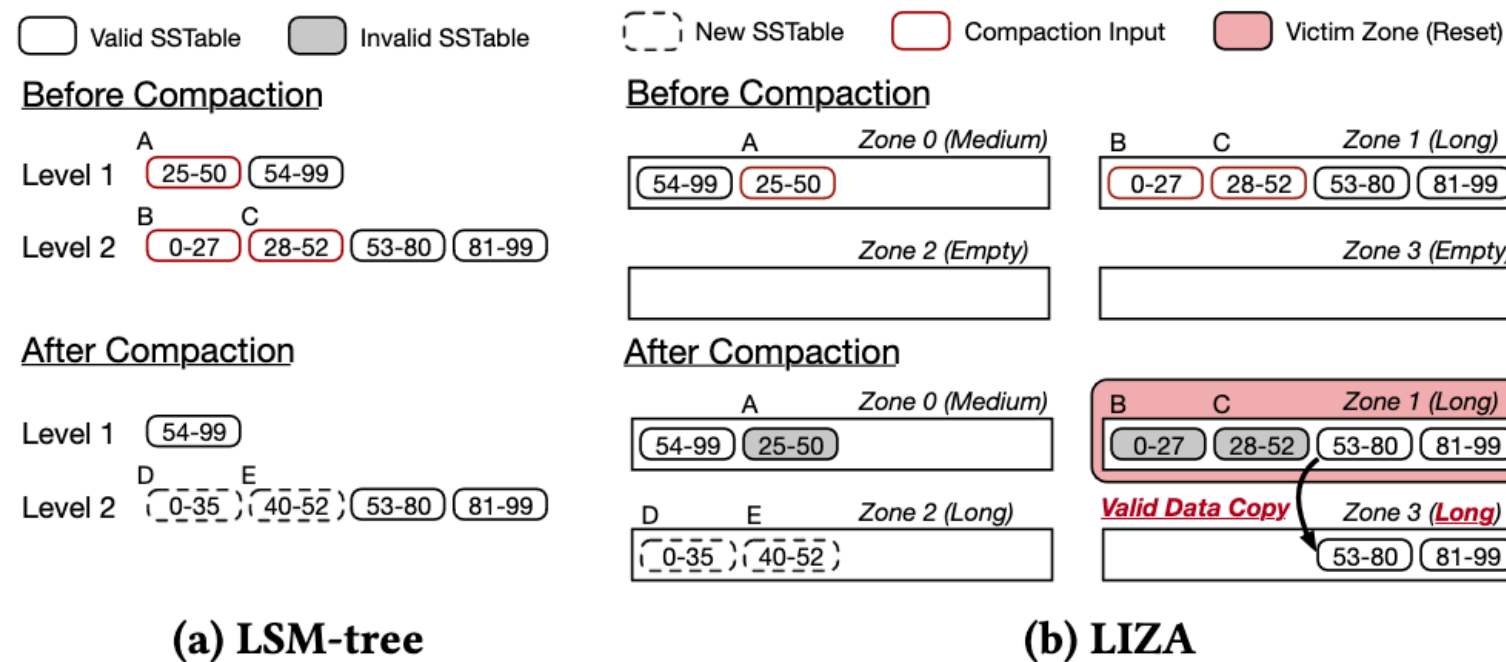
- ▶ Separate journaling from writes
- ▶ ZenFS can use write lifetime hints to place data
- ▶ RocksDB can supply these hints based on levels



(a) Total Copied Data (b) Write Amplification

Figure 3: WA analysis of LIZA and CAZA

Compaction-Aware Zone Allocation for LSM based Key-Value Store on ZNS SSDs



HotStorage'22, June 27–28, 2022, Virtual Event, USA

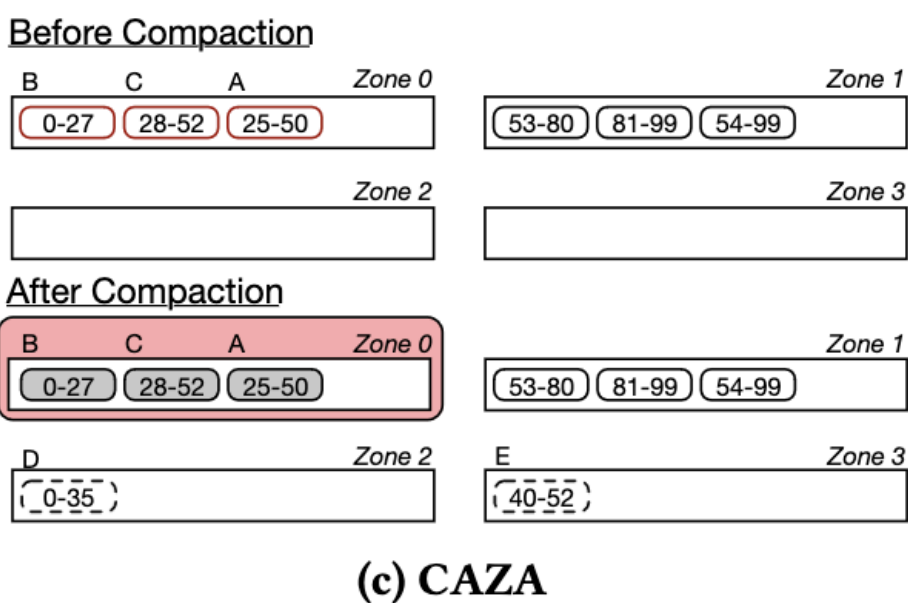
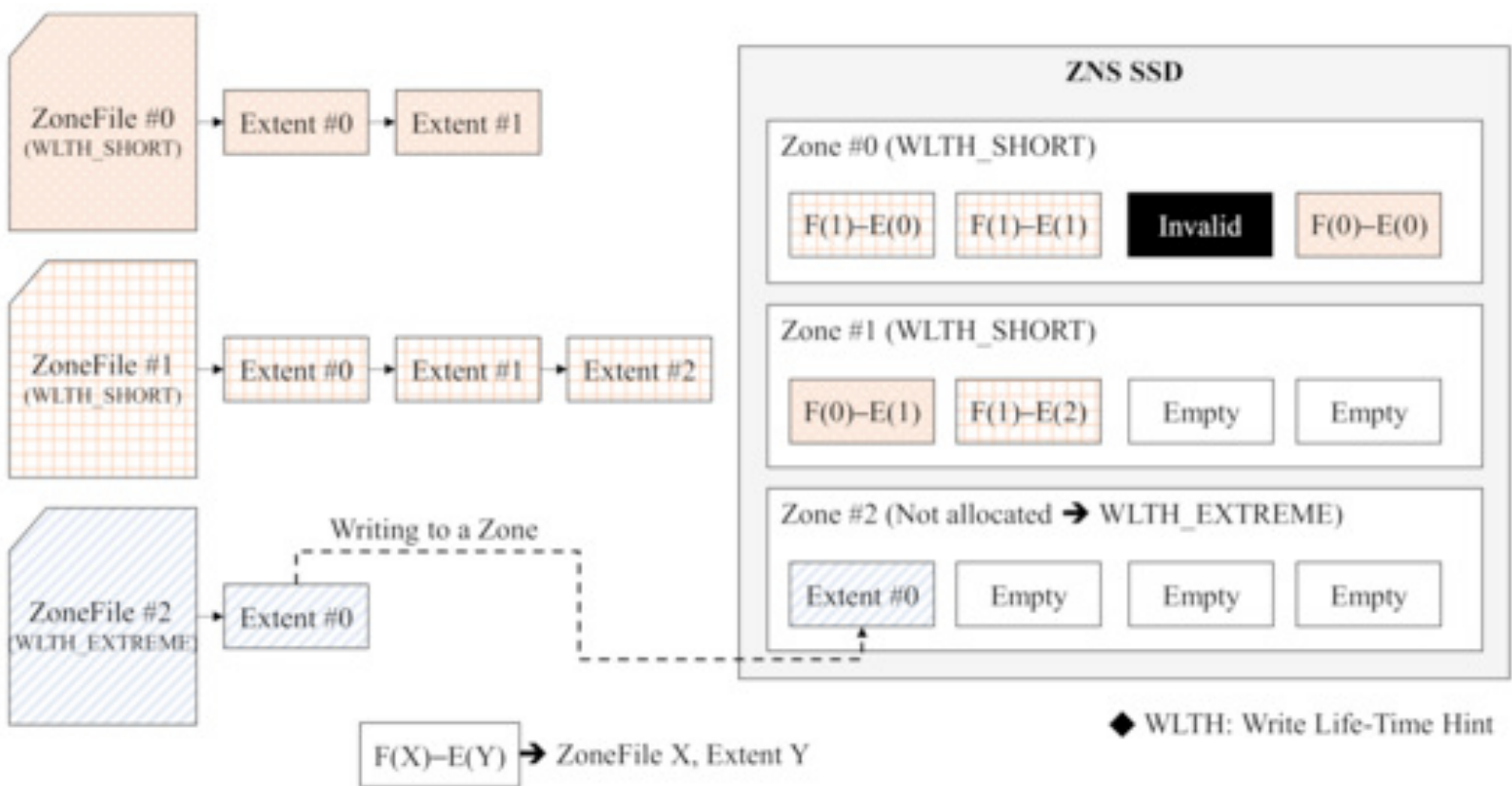


Figure 2: Examples showing the efficiency of zone cleaning of CAZA compared to LIZA



Efficient Key-Value Data Placement for ZNS SSD

Blueprints for effective hints

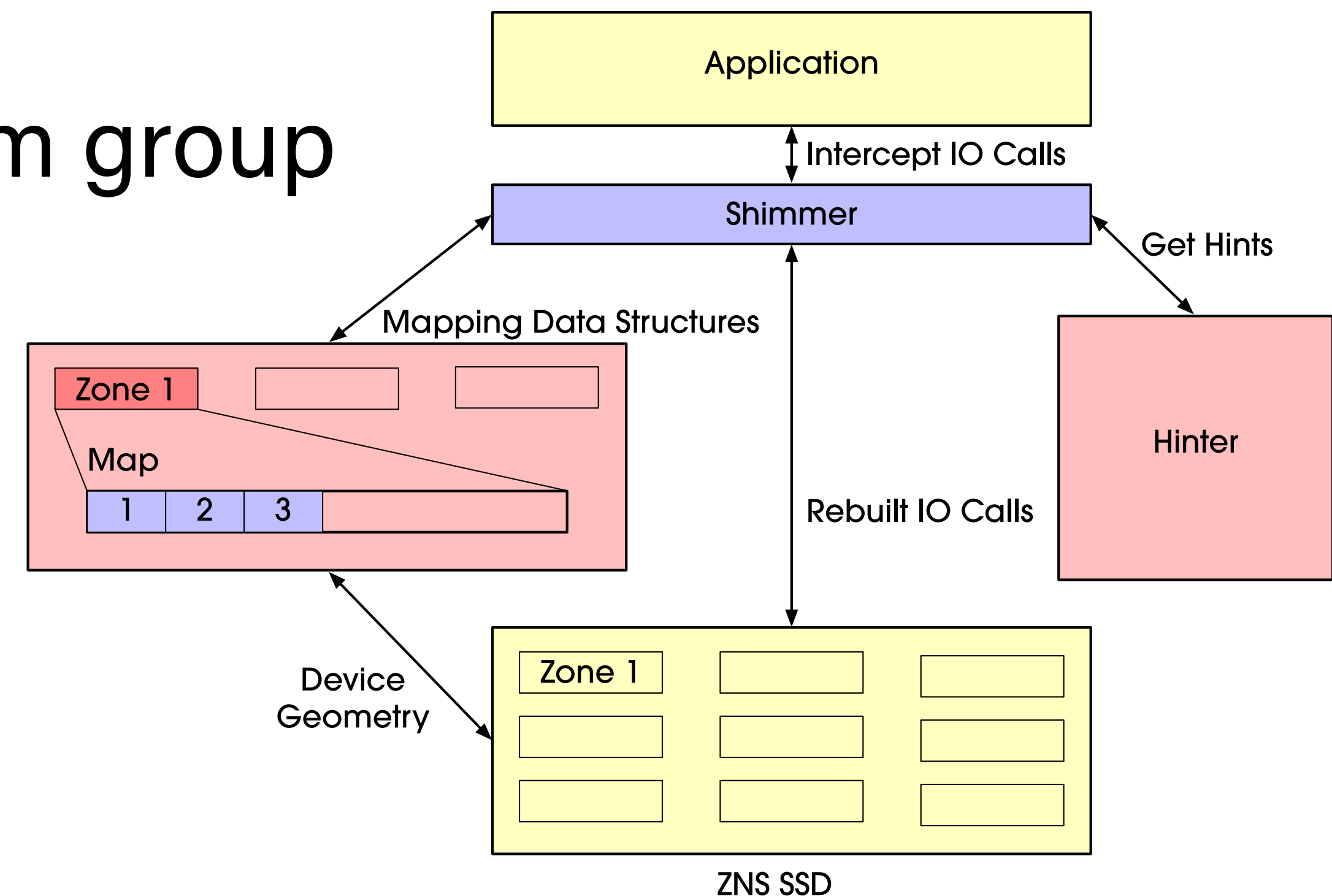
- ▶ **Application:**
 - ▶ Is unaware of hardware layout
 - ▶ Unaware of other application
- ▶ **Filesystem:**
 - ▶ Unaware of application workload characteristics
 - ▶ Applications need to be rewritten

An ideal hint generator

- Aware of storage hierarchy
- Aware of application and workload
- Decoupled from application, filesystem for greater visibility
- Lightweight and efficient
- Could be implemented as a *shim layer*:
 - Possible to use WASI/eBPF to inject hints?
 - Or dynamic libraries?

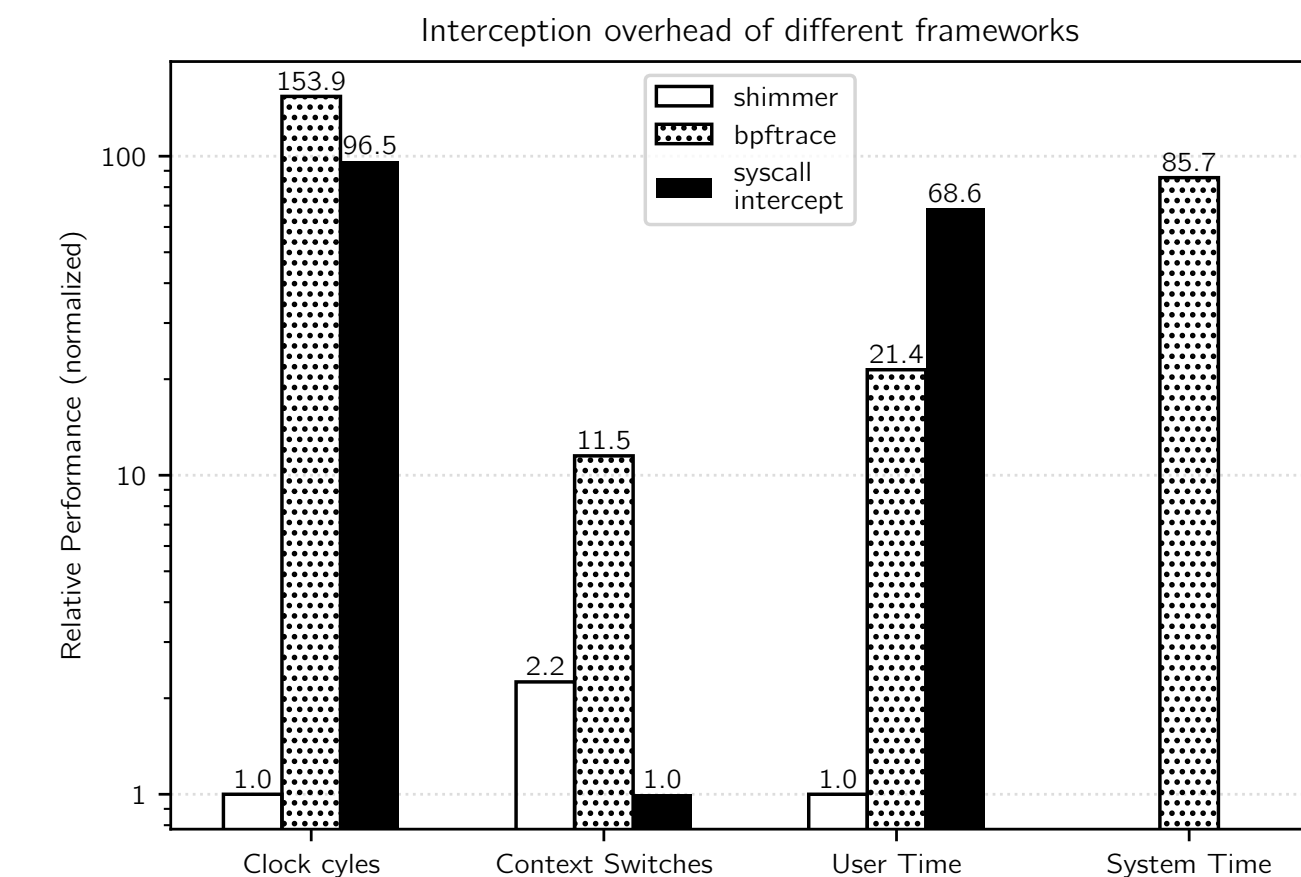
Idea: Dynamic interception

- Write() calls are intercepted
 - A trained model is used to generate hint
 - In ZNS files are mapped to zones
 - In FDP, just need to supply reclaim group
 - Model trained on deletion



Shimmer

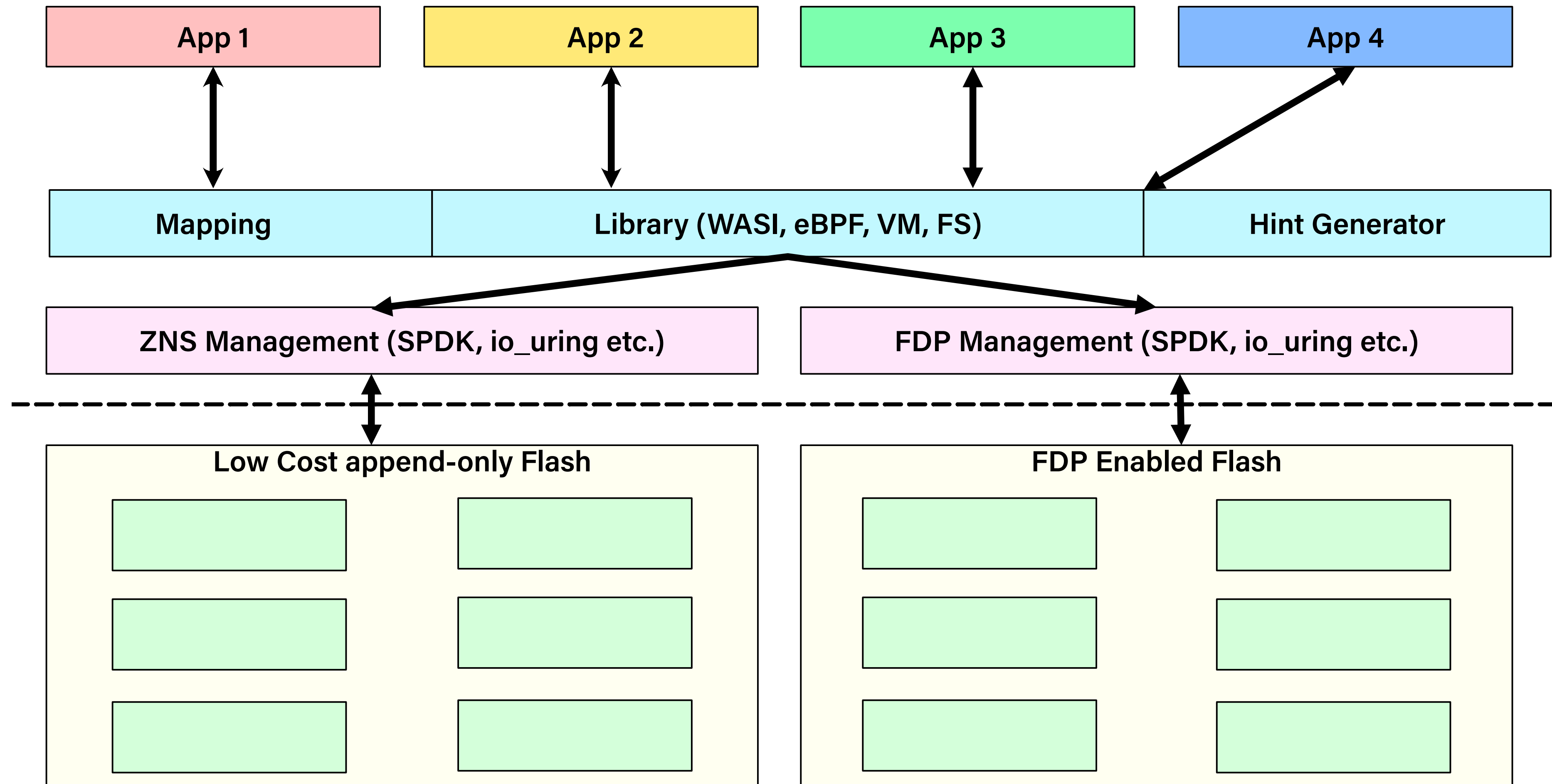
- ▶ Uses dynamic library loading to modify system calls to read() and write()
- ▶ Injects hints based on a trained k-nearest-neighbors model on file lifetimes
- ▶ Groups related files together in the same zone/resource group
- ▶ Work in progress
- ▶ **Check out the poster at FMS!**



A SHIMMER overloaded write() call that adds a hint for the Stream SSD interface.

```
unsafe fn write(  
    &mut self,  
    fd: c_int,  
    buf: *const c_char,  
    nbytes: size_t,  
) -> c_int {  
    let path = get_path(fd);  
    if !self.hinted.contains(&path) {  
        self.hinted.insert(path)  
        let hint = get_hint(path);  
        let _ = fcntl(fd, F_SET_RW_HINT, &hint);  
    }  
}
```

Large-Scale Deployment



Opportunities

- Leverage **fast path optimizations**
 - eBPF, SPDK, io_uring for reducing kernel overhead
- Maintain **backwards compatibility**
 - Interception using VMs, Filesystems, eBPF, dynamic libraries
- Leverage **observability and machine learning**
 - Dynamic application-specific, workload-specific hints
- Improve performance, lifetime, and reduce cost

Questions?

Everything referenced in this talk is also listed at: <https://github.com/devashishp/FMS>

About Me:

Dev Purandare

Center For Research in Storage and Systems
Baskin School of Engineering
University of California, Santa Cruz

devashish@ucsc.edu

LinkedIn: devashishp

Website: <https://sincerely.dev>

I am currently looking for full-time cutting-edge data management positions.

