Zebra: Efficient Redundant Array of Zoned Namespace SSDs Enabled by Zone Random Write Area (ZRWA)

Tianyang Jiang[1], Guangyan Zhang[2], Xiaojian Liao[3], Yuqi Zhou[4]

[1]*Huawei Technologies* [2]*Tsinghua University*

[3]*Beihang University* [4]*China University of Geosciences*

Zoned Namespace (ZNS) SSD

ZNS SSDs enable **higher performance** and **lower cost** compared to conventional block-interface SSDs.

- No device-level garbage collection (GC)
- Coarse-grained flash translation layer
- Low flash capacity over-provisioning

Western Digital

Samsung

DapuStor

ZNS-based storage software community is increasingly **active**!

- Shifting responsibilities for **data placement** from devices to host software
- Replacing host-level GC with **uncontrollable device-level GC**
- **Filesystems**: F2FS, BtrFS
- **Databases**: RocksDB, MySQL

RocksDB MySQL

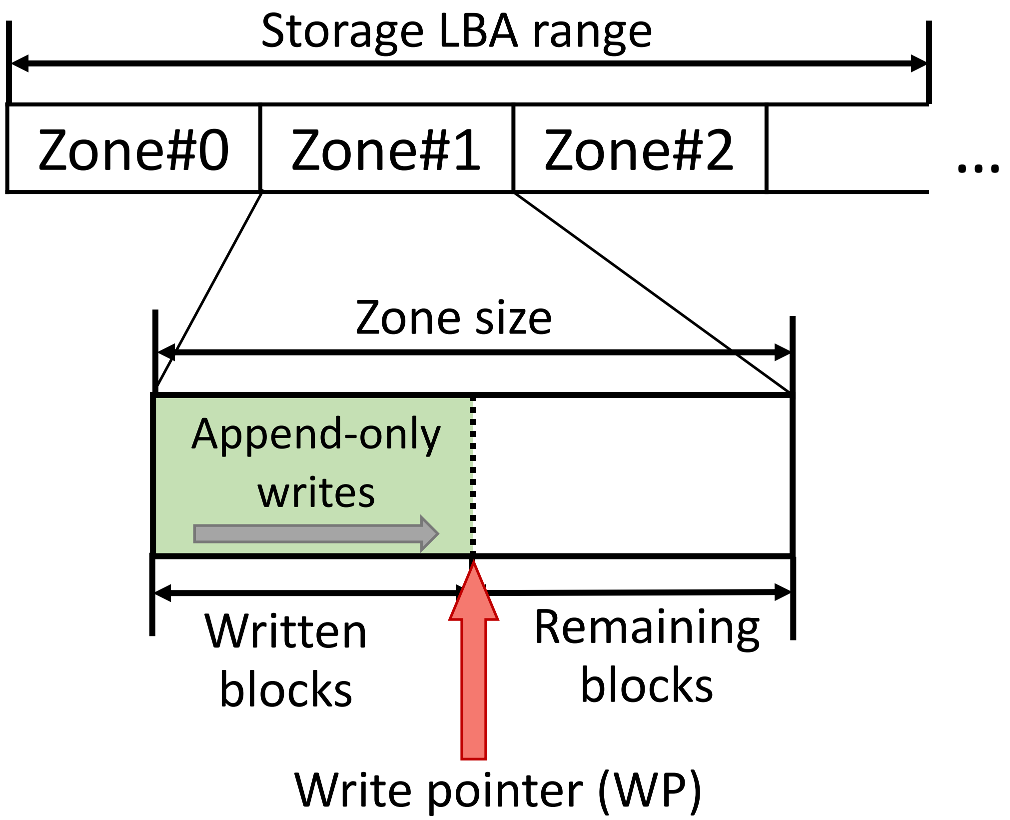
Zoned Namespace (ZNS) SSD

ZNS SSDs expose **ZONE** abstraction to storage applications.

- The logical block address space is divided into fixed-size zones.

What is the **ZONE**?

- Random reads
- **Append-only writes**, no overwrites
- Erase as a whole
- New writes must be appended after the write pointer (WP)



When scaling, RAID comes...

RAID: Redundant array of independent disks

Widely used in diverse domains

- Large-scale storage server in datacenters
- Disaggregated storage pool in cloud

Banks

Datacenters

Clouds

Building RAID with ZNS SSDs for ZNS-based applications

1. High aggregated bandwidth
2. Fault tolerance
3. Zone abstraction

Motivation: PPU v.s. ZNS

In-place updates in PPU is incompatible with **append-only** semantic in ZNS.

PPU: Partial parity updates

- When write request < chunk size, PPU happens in RAID.
- Long execution path
- **In-place updating** parity chunks

Storage workloads are dominated by small-sized write I/Os.

- 75% of writes < 16KiB in clouds

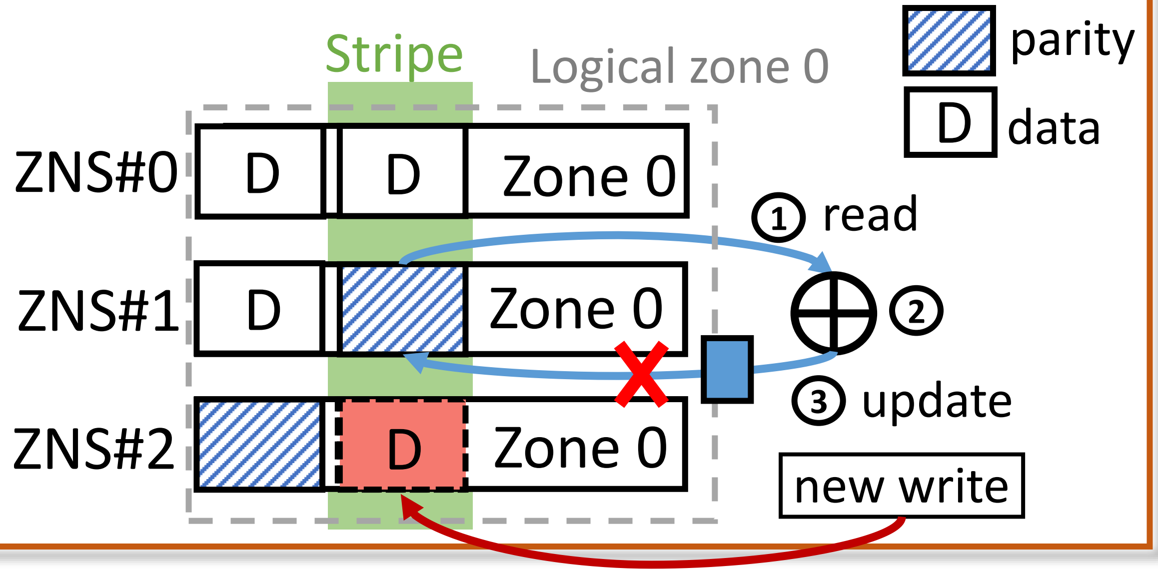
➢ **Low write performance** in conv. RAID

➢ More serious in ZNS RAID

ZNS: **Append-only** writes in zones

A ZNS RAID write request needs:

1. read old parity
2. calculate new parity
3. update parity



Existing solutions

Batching and issuing I/Os at stripe granularity:

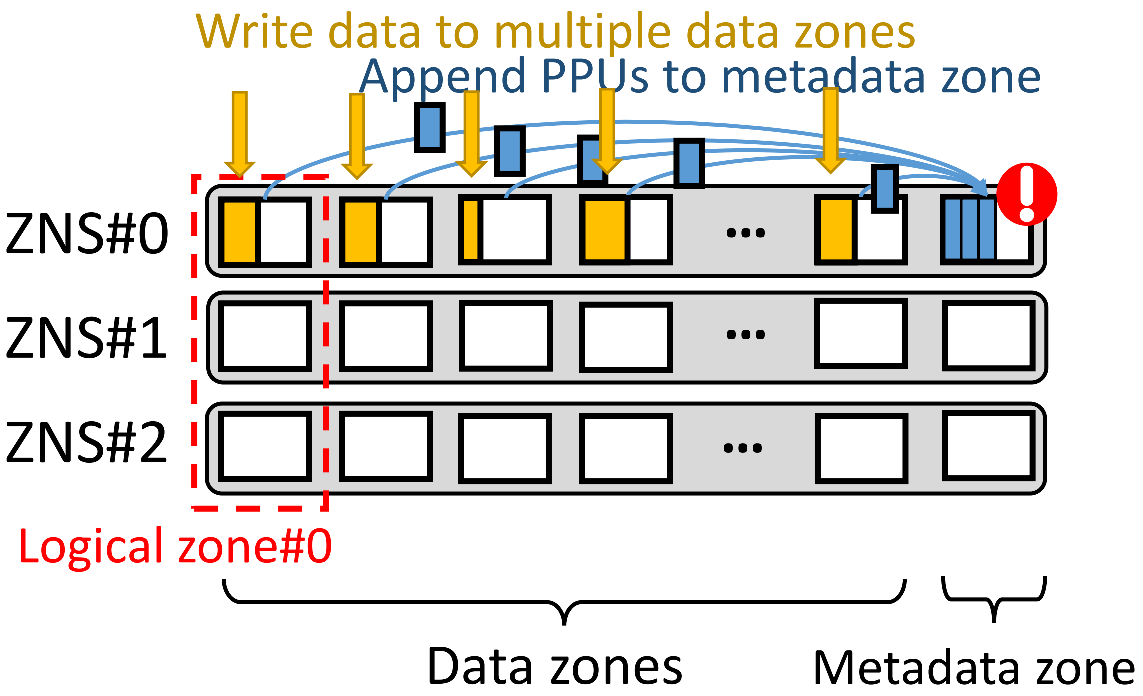
- ZapRAID [Apsys’23, TOS’24]
- High latency
- Lack of instant durability
- Degraded to PPU when fsync()

Allocating dedicated metadata zones for buffering PPUs:

- RAIZN [ASPLOS’23]
- Contention of multi-zone PPU aggregation
- RAID-level garbage collection

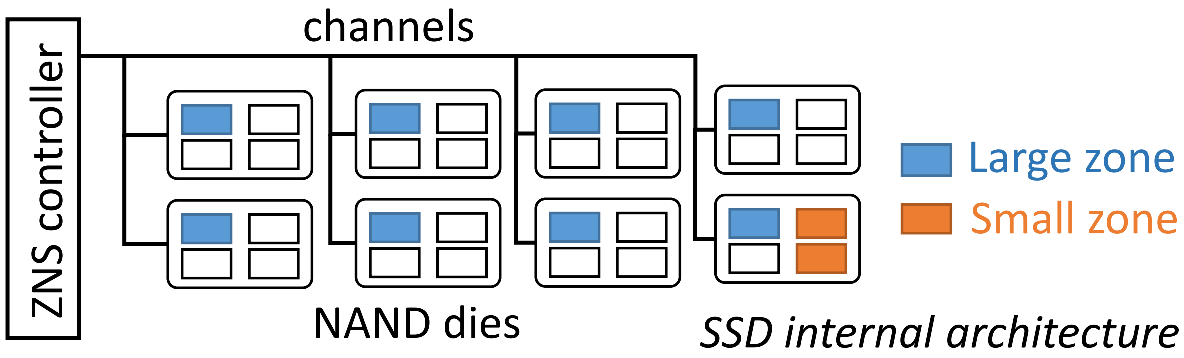
Issue 1: Contention of multi-zone PPU aggregation

Allocating 1 metadata zone to buffer PPUs from other data zones



Getting worse in small-zone ZNS RAID

- **Large-zone**: Striping across all channels
- **Small-zone**: Redirecting to 1 die
  - Bandwidth isolation between zones



Bandwidth: **Multiple** data zones v.s. **One** metadata zone

Throughput degradation:
Large-zone **19%**↓ Small-zone **76%**↓

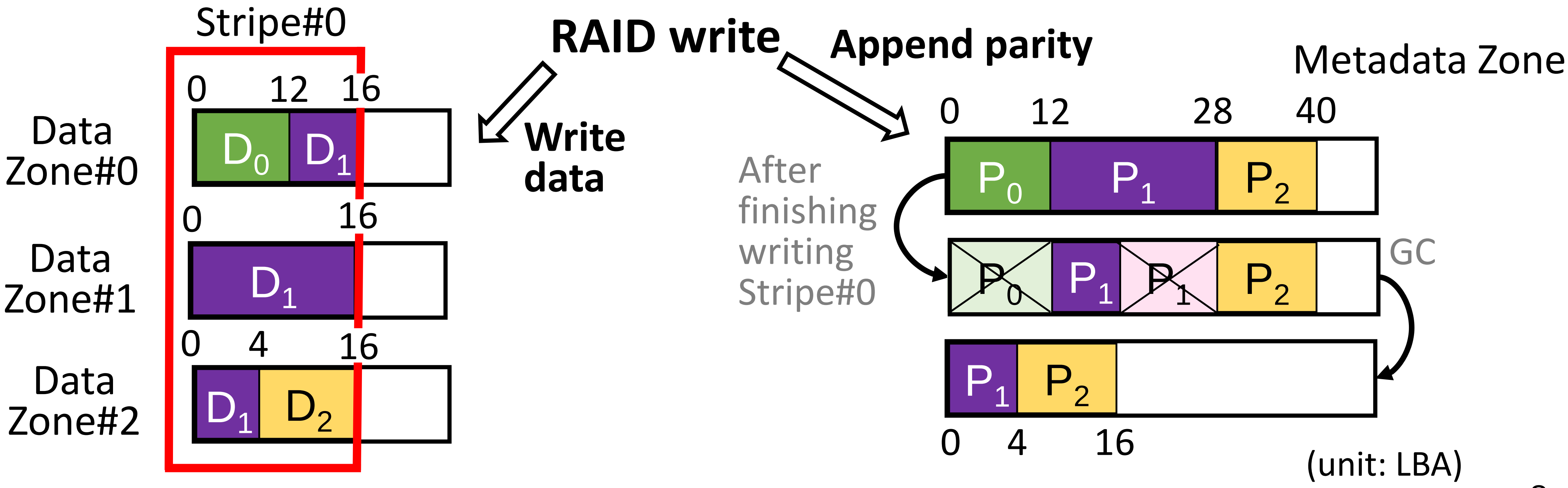
Issue 2: RAID-level Garbage collection

RAID-level GC consumes write bandwidth in ZNS RAID.

Extra space overhead by PPUs:

- Each RAID write generates a PPU in metadata zones

Reclaim obsolete PPUs periodically! ⇨ RAID-level GC

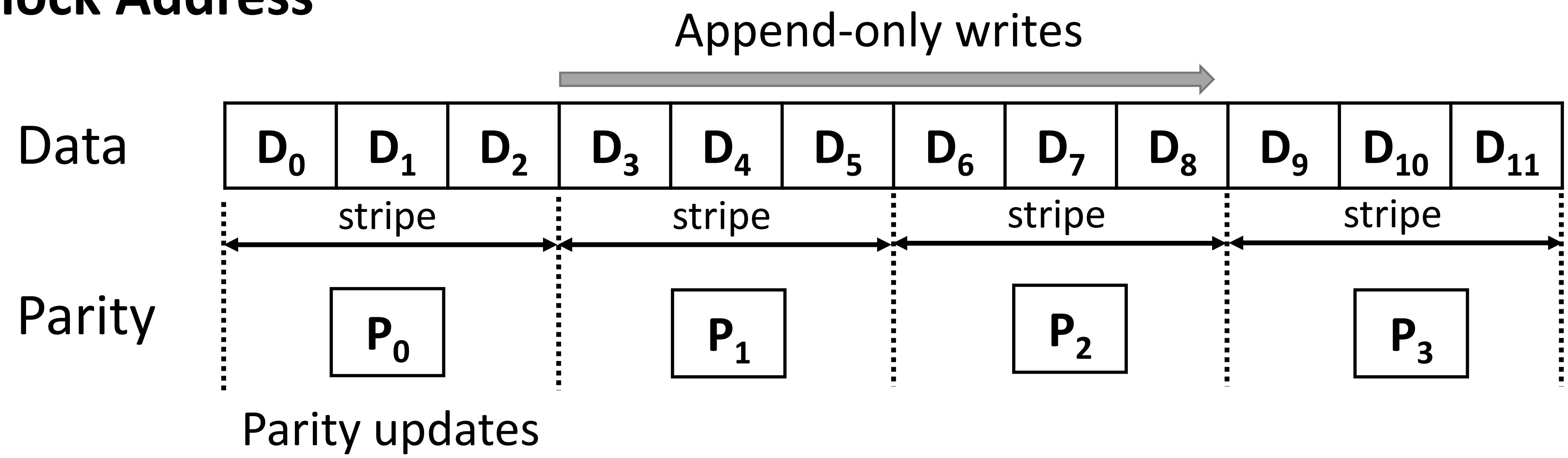


Throughput: **6.2% - 15%↓** P99 latency: **65%↑**

Observation

Parity chunks ***will not be updated*** *upon all the data blocks within the stripe have been written.*

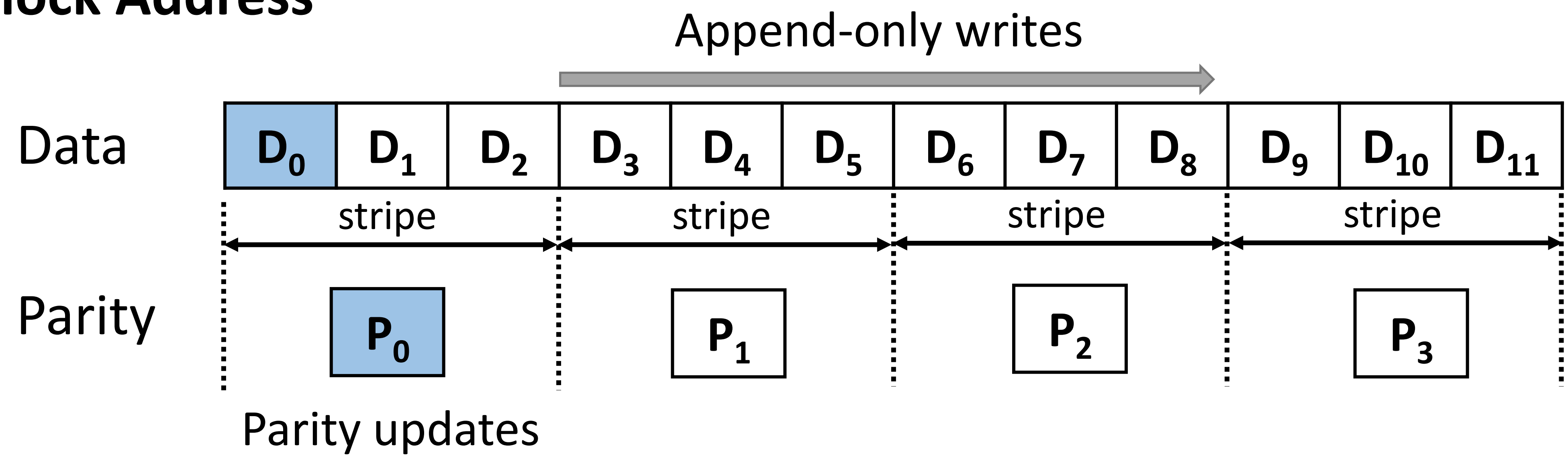
Logical Block Address



Observation

Parity chunks ***will not be updated*** *upon all the data blocks within the stripe have been written.*

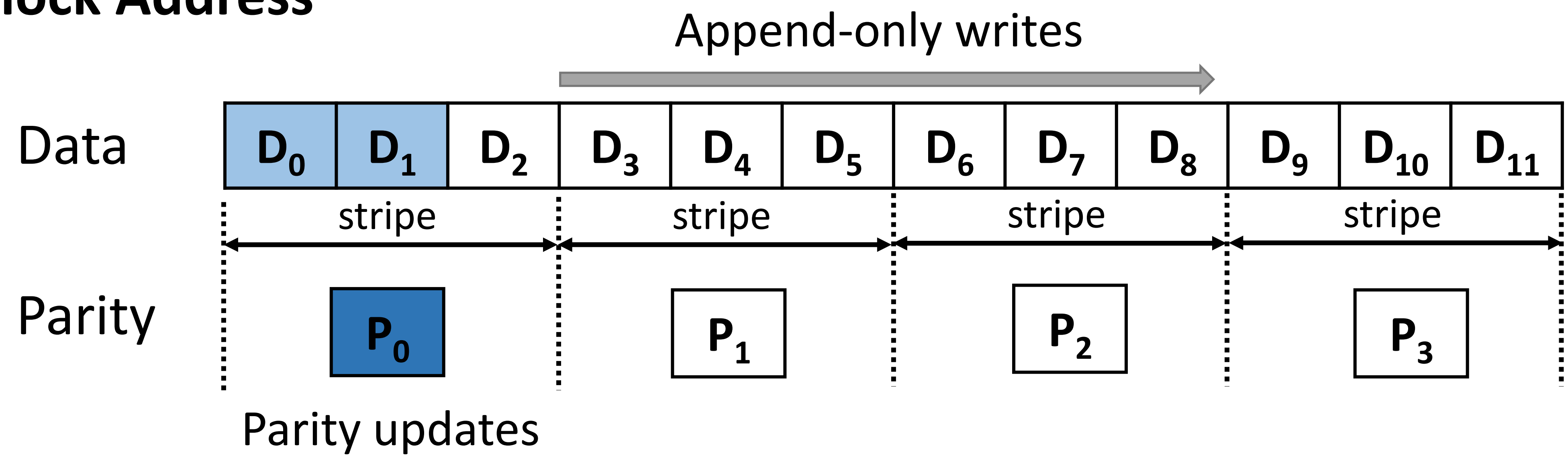
Logical Block Address



Observation

Parity chunks ***will not be updated*** *upon all the data blocks within the stripe have been written.*

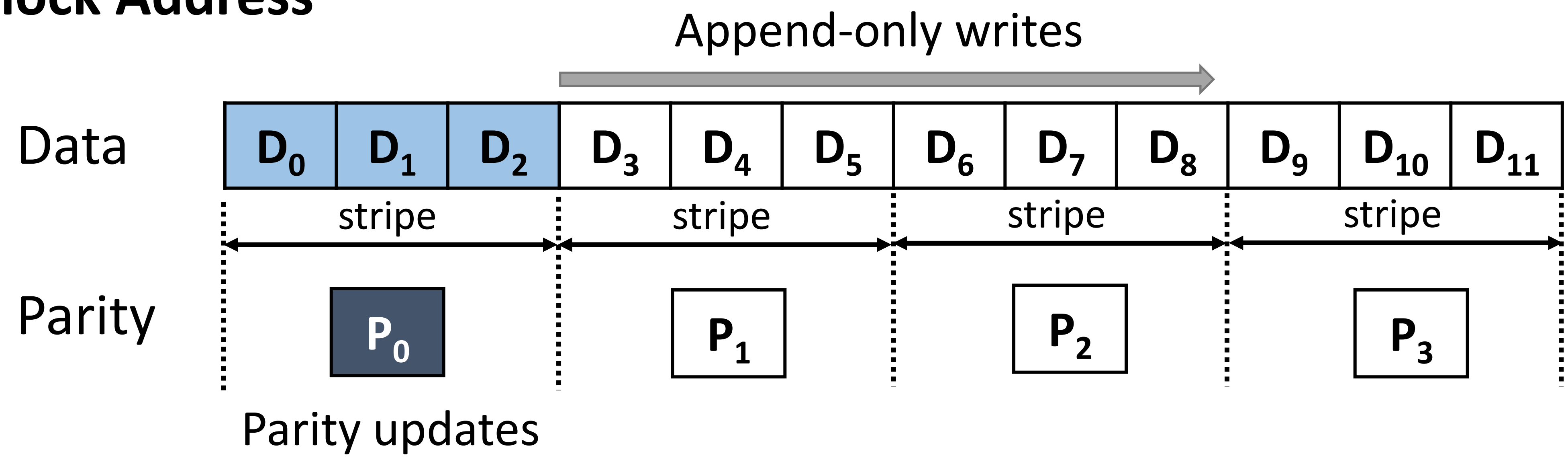
Logical Block Address



Observation

*Parity chunks **will not be updated** upon all the data blocks within the stripe have been written.*

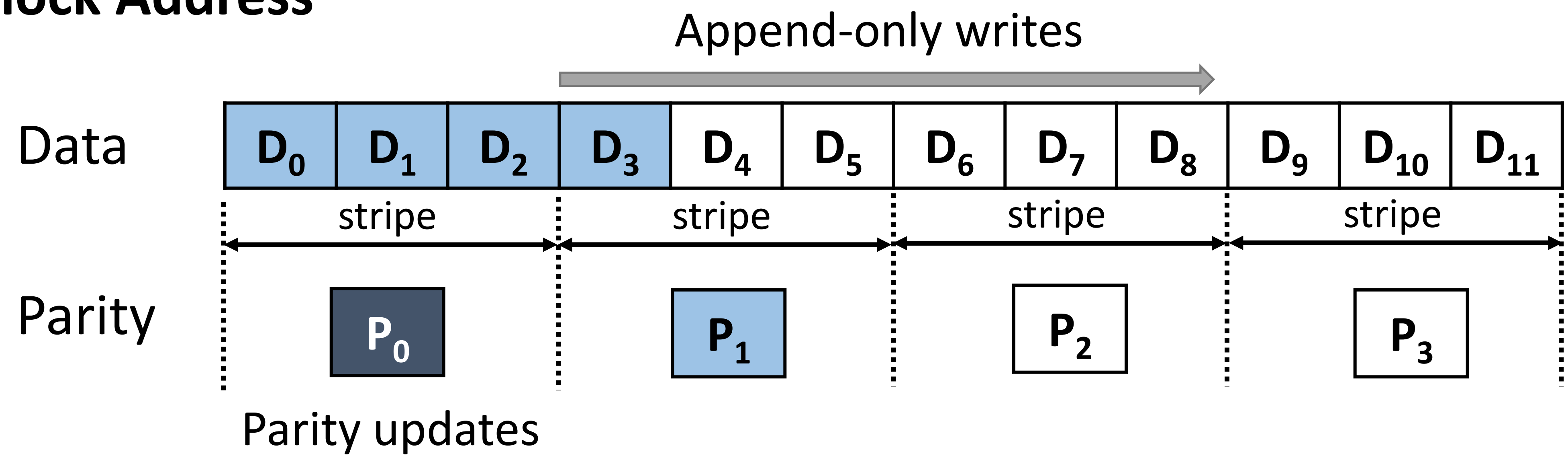
Logical Block Address



Observation

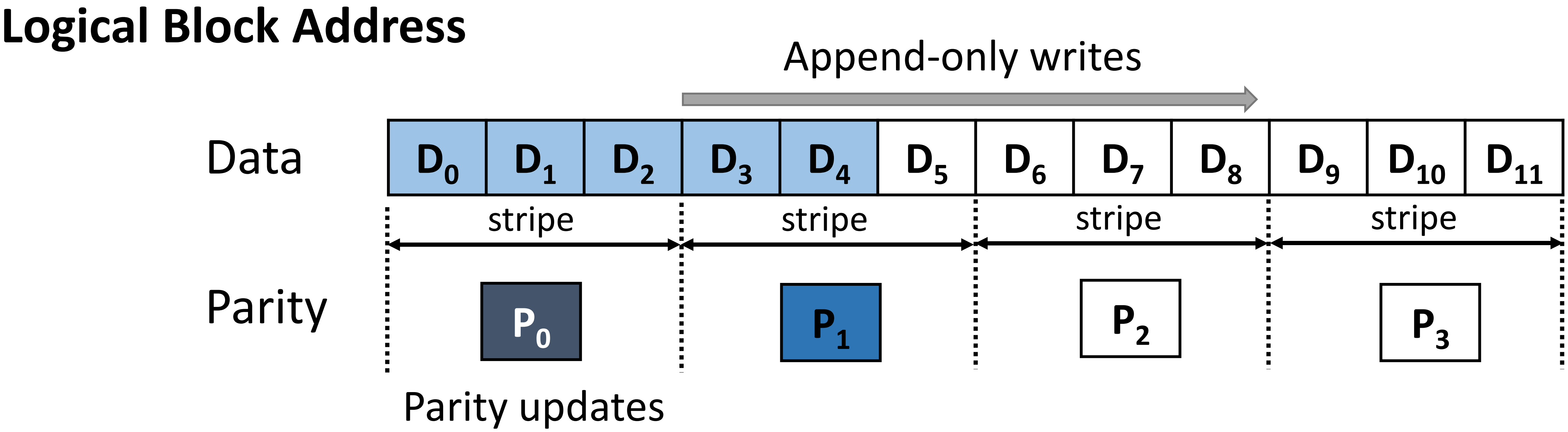
*Parity chunks **will not be updated** upon all the data blocks within the stripe have been written.*

Logical Block Address



Observation

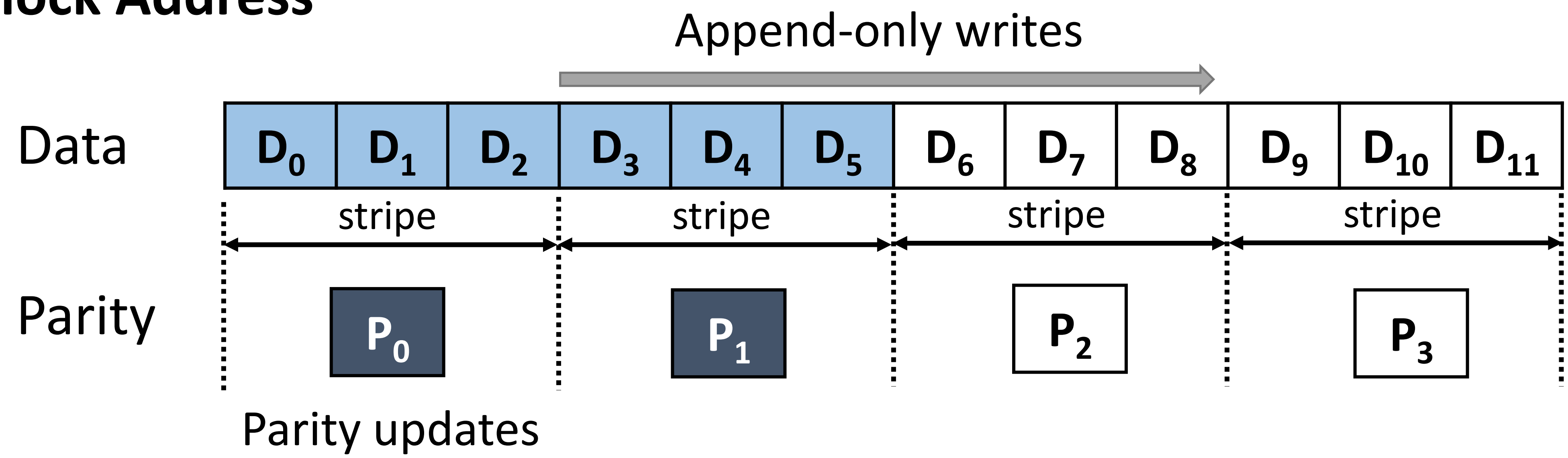
*Parity chunks **will not be updated** upon all the data blocks within the stripe have been written.*



Observation

Parity chunks ***will not be updated*** *upon all the data blocks within the stripe have been written.*

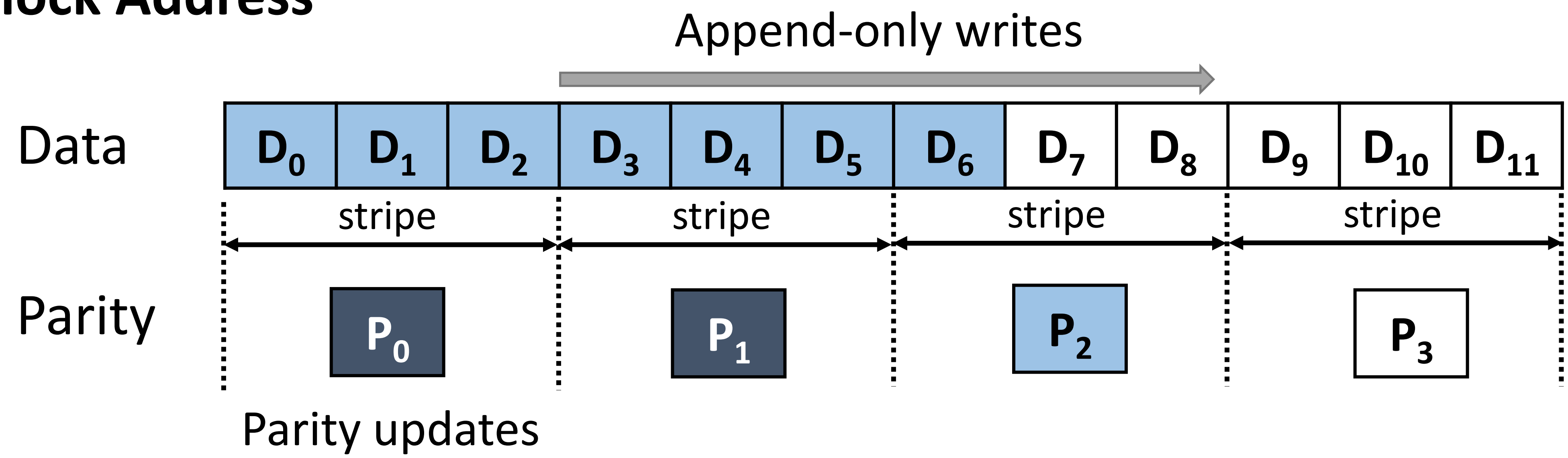
Logical Block Address



Observation

Parity chunks ***will not be updated*** *upon all the data blocks within the stripe have been written.*

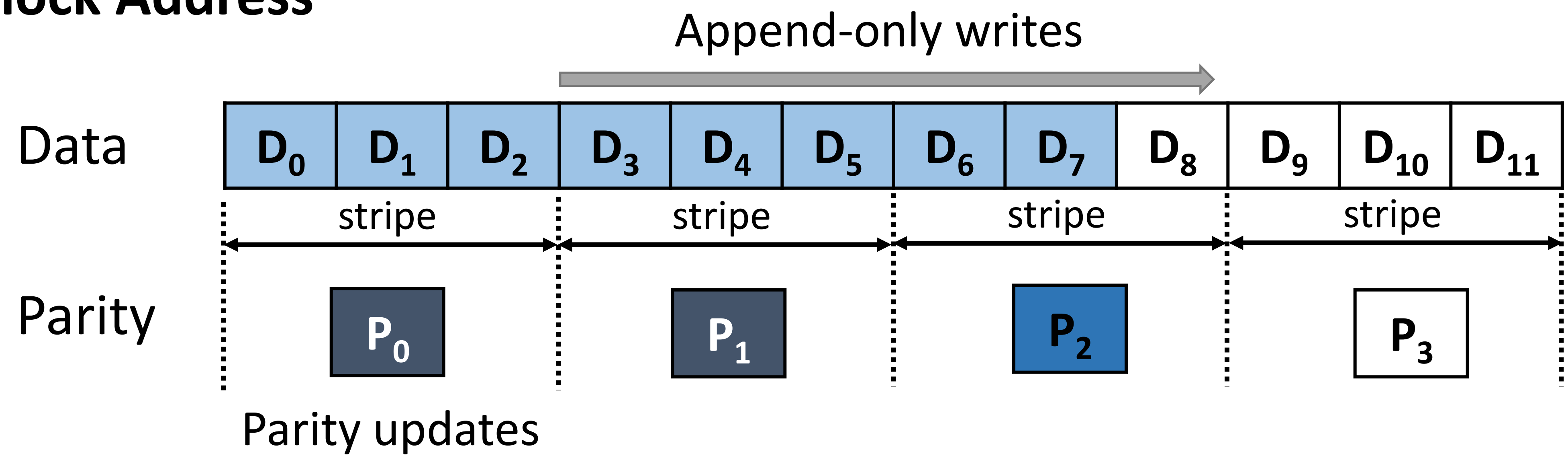
Logical Block Address



Observation

Parity chunks ***will not be updated*** *upon all the data blocks within the stripe have been written.*

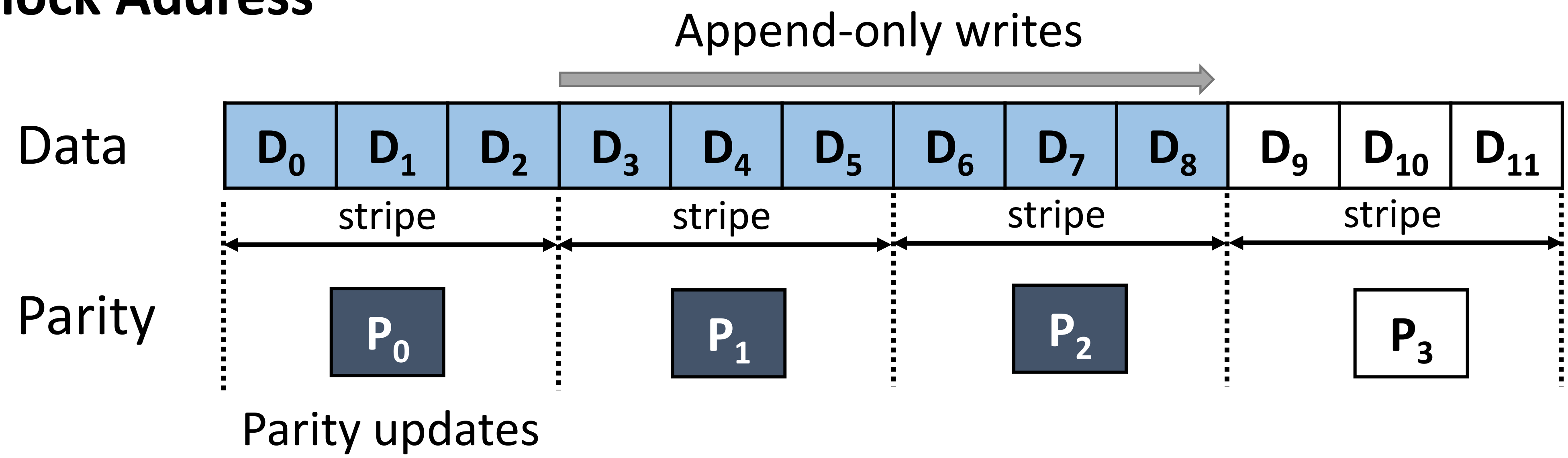
Logical Block Address



Observation

Parity chunks ***will not be updated*** *upon all the data blocks within the stripe have been written.*

Logical Block Address



Behaving like a **sliding window**!

What we want in ZNS RAID?

An area in ZNS:

1. Overwrite support
    - Processing PPUs in place
    - Avoiding RAID-level GC
2. De-centralized architecture
    - Private to each zone
    - No bandwidth contention between zones
3. Sufficiently large size
    - Holding parity chunks

Zone Random Write Area (ZRWA)

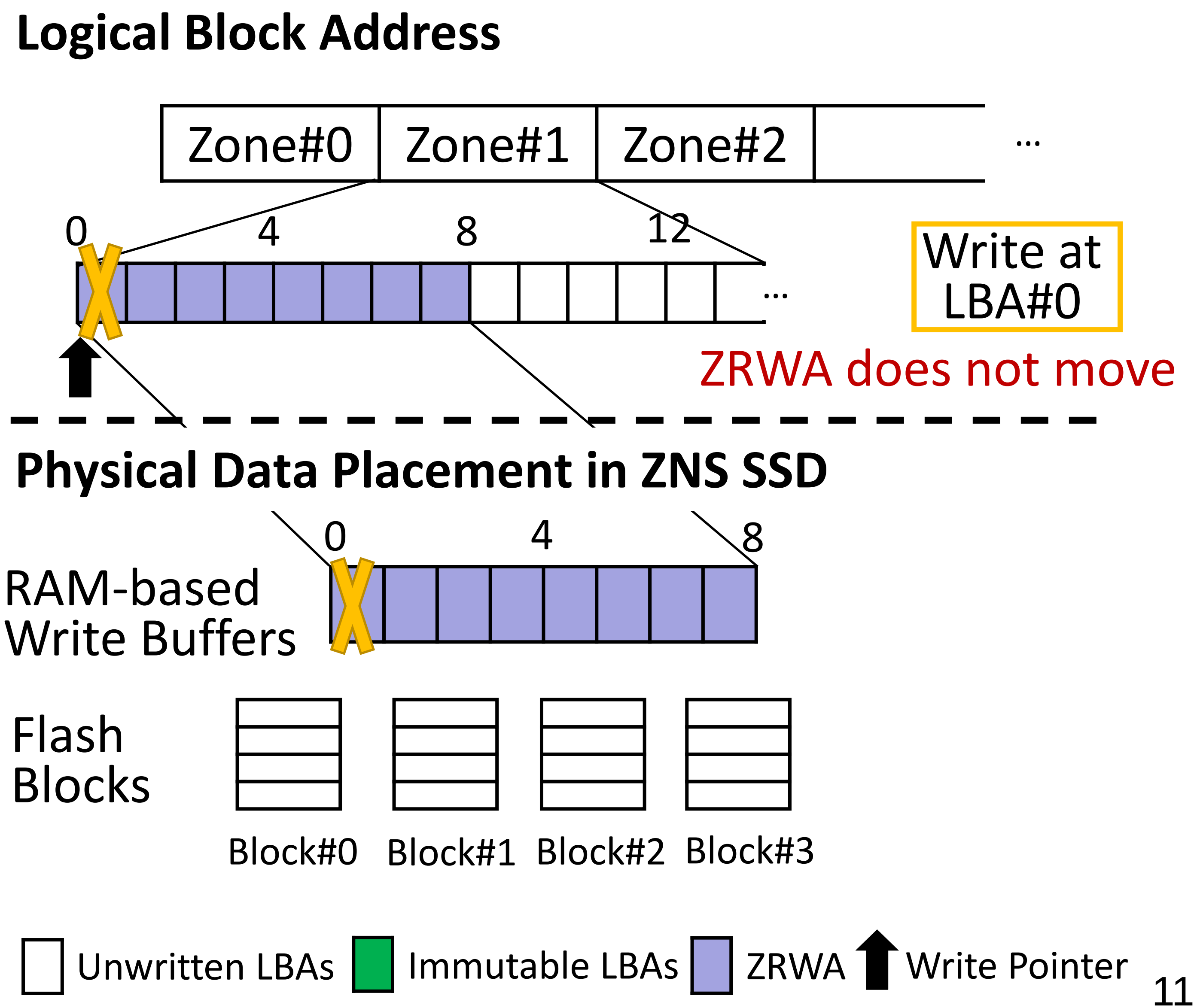
Included in NVMe specification

An area following WP

- Supporting **overwrites**
- **Moving** with data writing

Implementation:

- Exposing RAM-based write buffers in SSDs to applications
- Flushing data to flash when ZRWA moves



Zone Random Write Area (ZRWA)

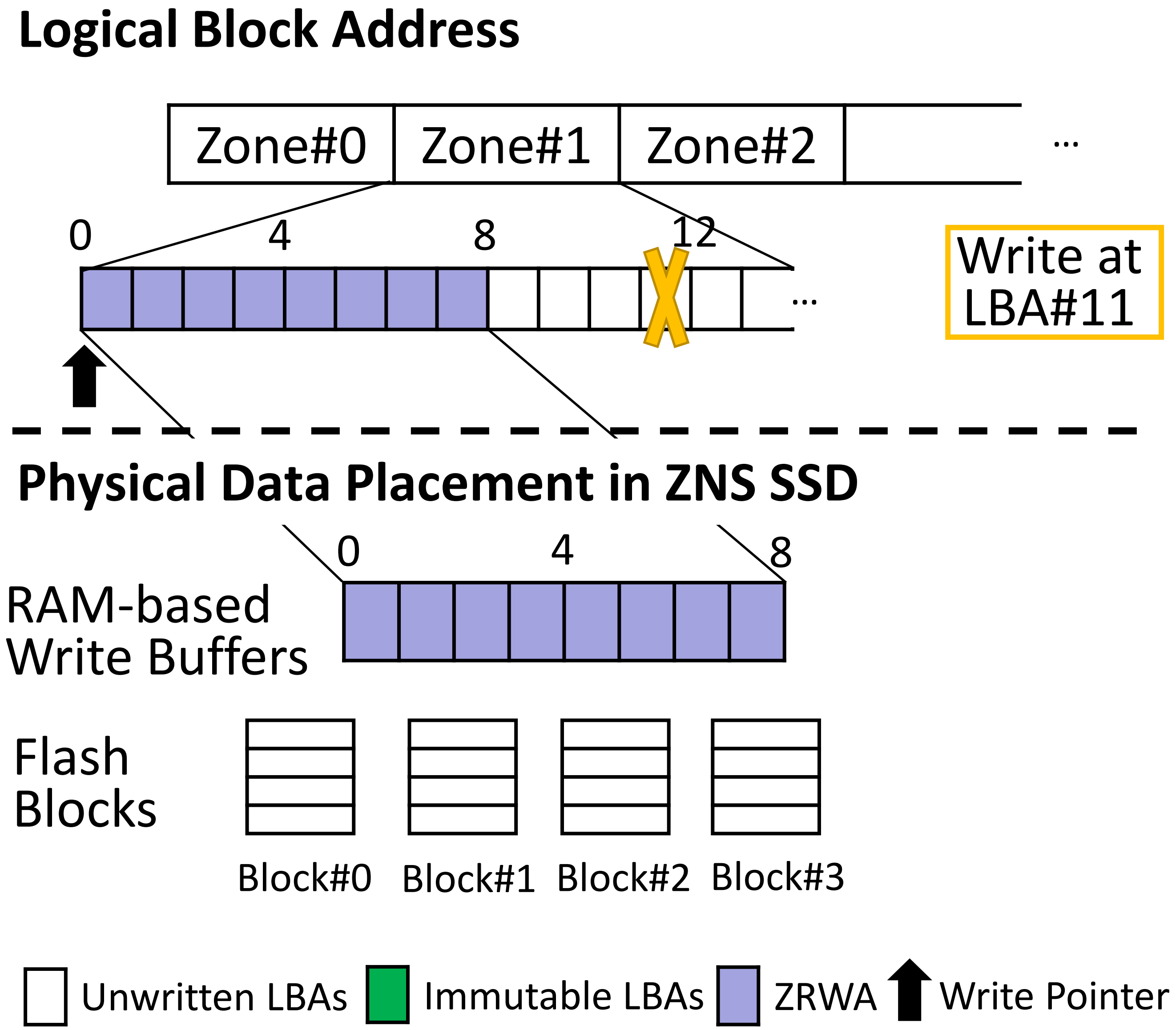
Included in NVMe specification

An area following WP

- Supporting **overwrites**
- **Moving** with data writing

Implementation:

- Exposing RAM-based write buffers in SSDs to applications
- Flushing data to flash when ZRWA moves



Zone Random Write Area (ZRWA)

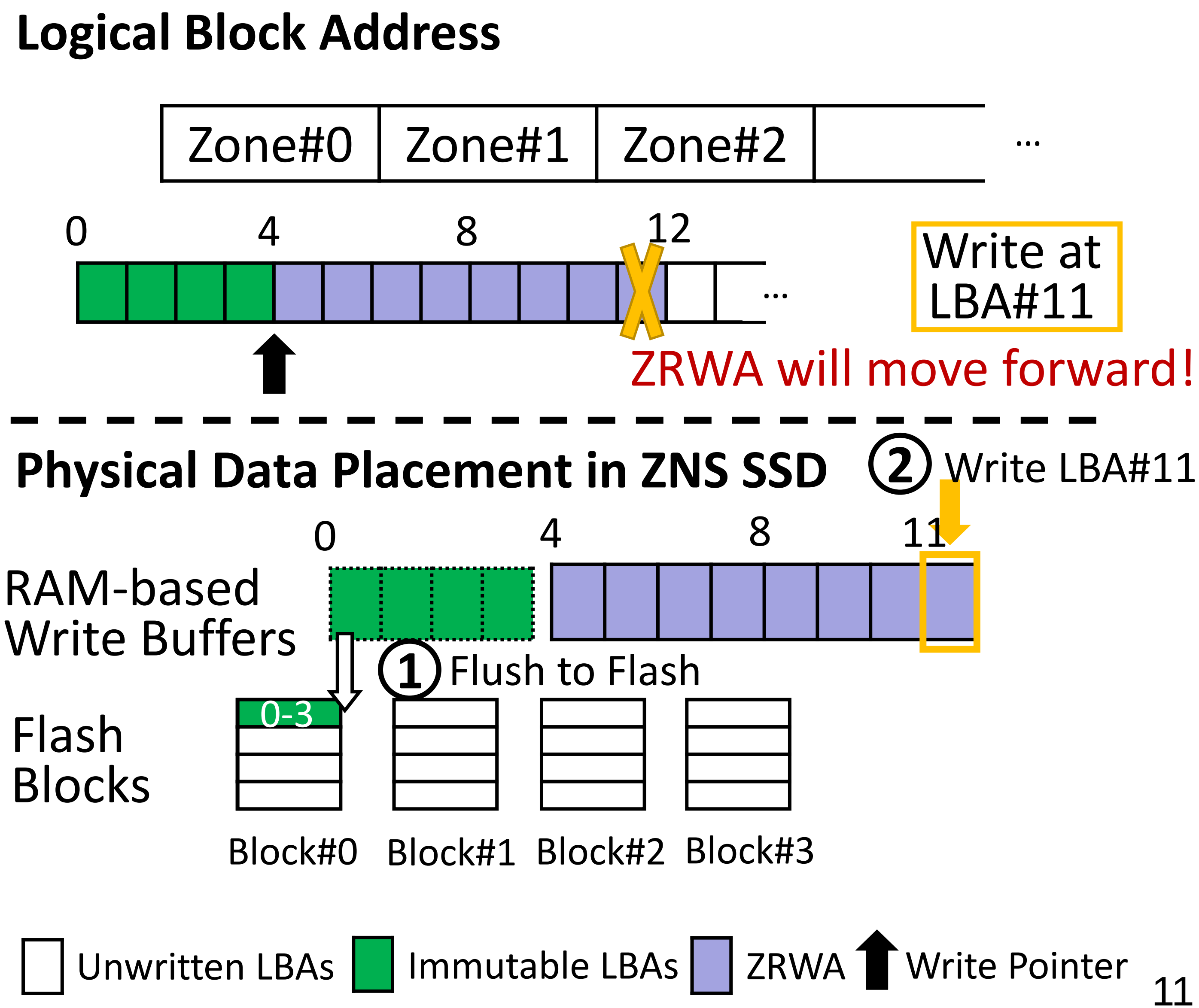
Included in NVMe specification

An area following WP

- Supporting **overwrites**
- **Moving** with data writing

Implementation:

- Exposing RAM-based write buffers in SSDs to applications
- Flushing data to flash when ZRWA moves



Why ZRWA fits ZNS RAID?

ZNS RAID requirements		ZRWA features
R1: Overwrite support for PPUs	⟺	Limited overwrite support Behaving like a **sliding window**
R2: De-centralized architecture	⟺	Each zone has a **private** ZRWA. Isolated in write buffer
R3: Sufficiently large size	⟺	ZRWA: **64KiB ~ 1MiB** per zone Parity chunk size: **≤ 64 KiB**

Perfect fit!

ZRWA-enabled RAID: Zebra

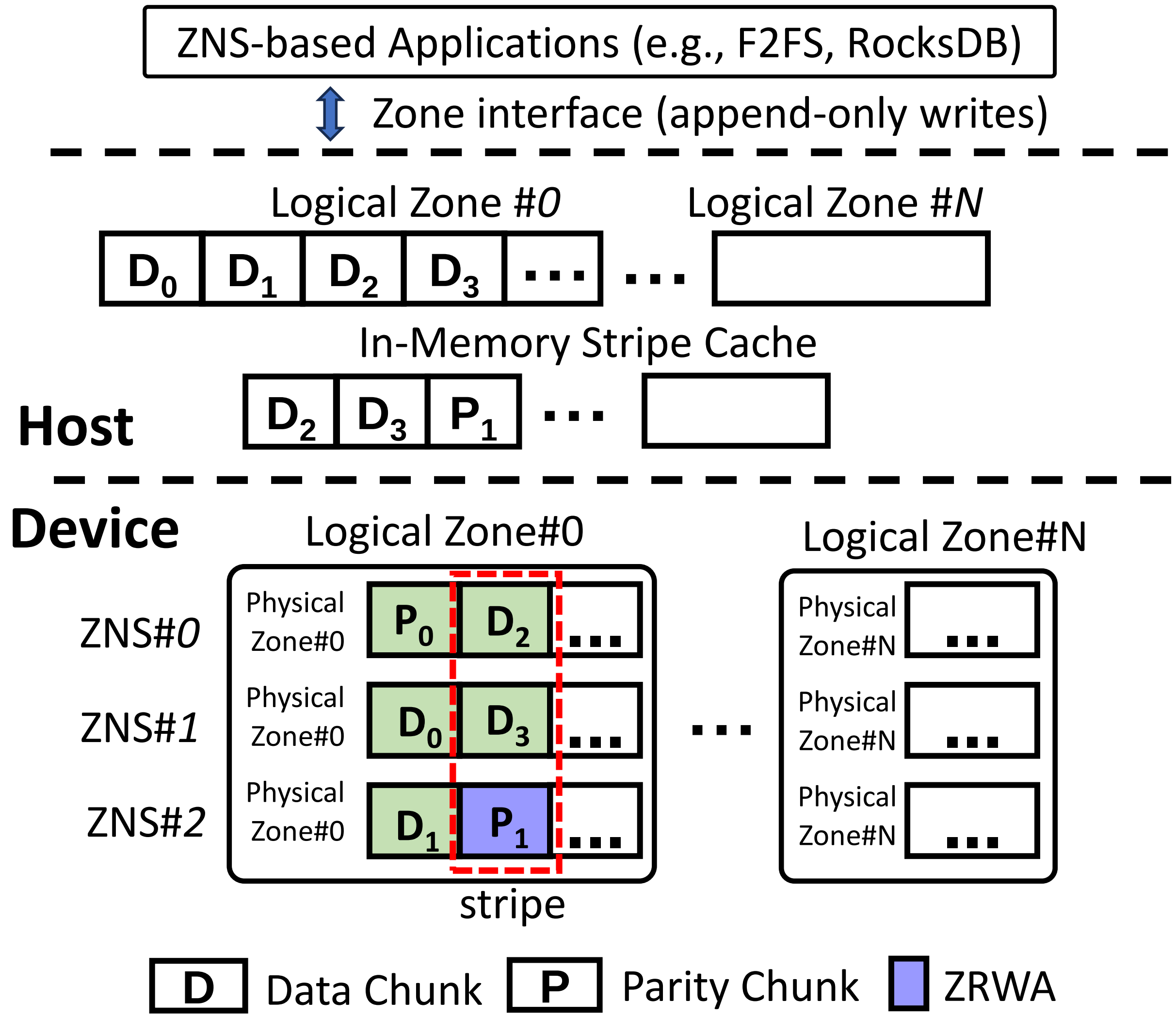
Idea: Holding parity chunks within ZRWA for in-place PPUs

Host side:

- Zone interface as a single device
- Logical zones
- Static L2P zone mappings
- In-memory stripe cache

Device side:

- Physical zones with ZRWA on
- Diverse RAID setups (e.g., RAID-5/6)



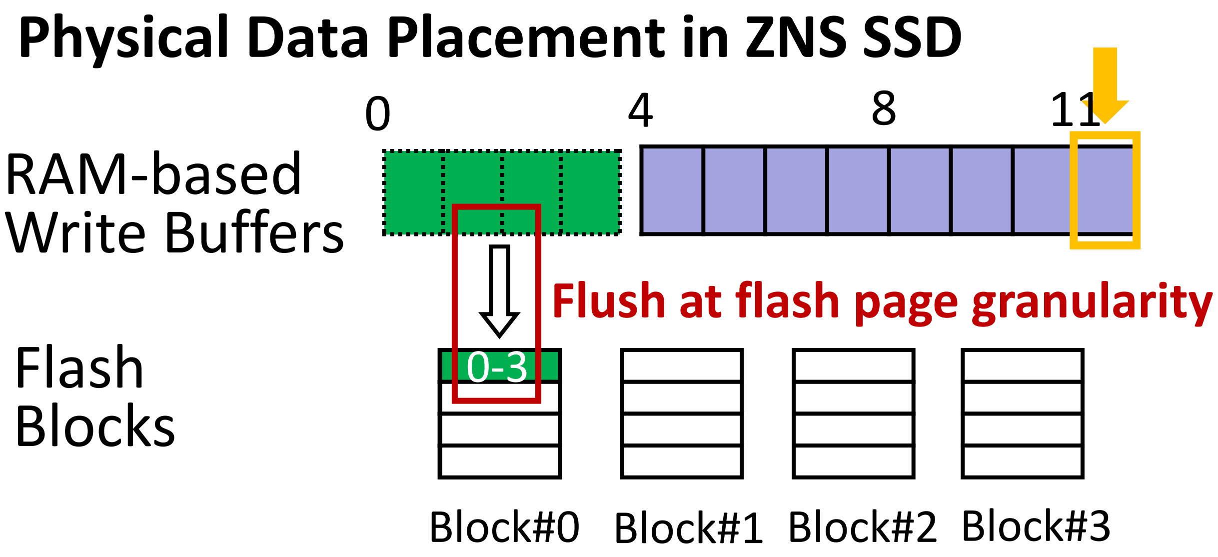
Challenge: Recovery from failure

Problem: During the recovery, zones with ZRWA cannot accurately identify the finished write position before the failure.

- Failure: A sudden power-off event

Distinction of moving granularity:

- ZRWA moves at **16KiB ~ 32KiB** granularity
- ZNS supports 1-LBA write (**4KiB**)



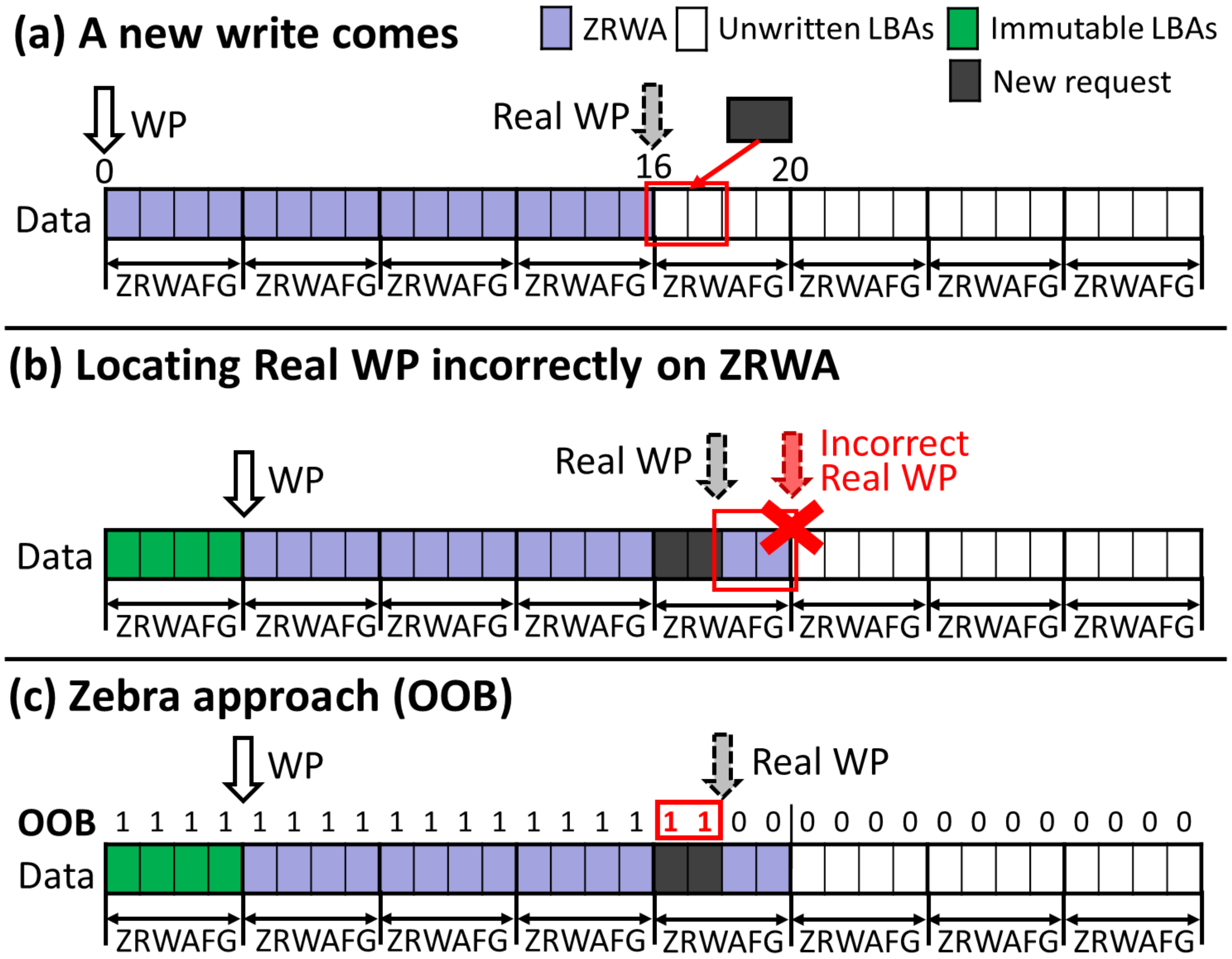
The flash page size of high-density NAND (MLC/TLC) **is inconsistent with** the write granularity of ZNS.

Locating WP with Lightweight Metadata

Real WP: Tail of successfully written data

Using **out-of-band (OOB) area** to record the data validity

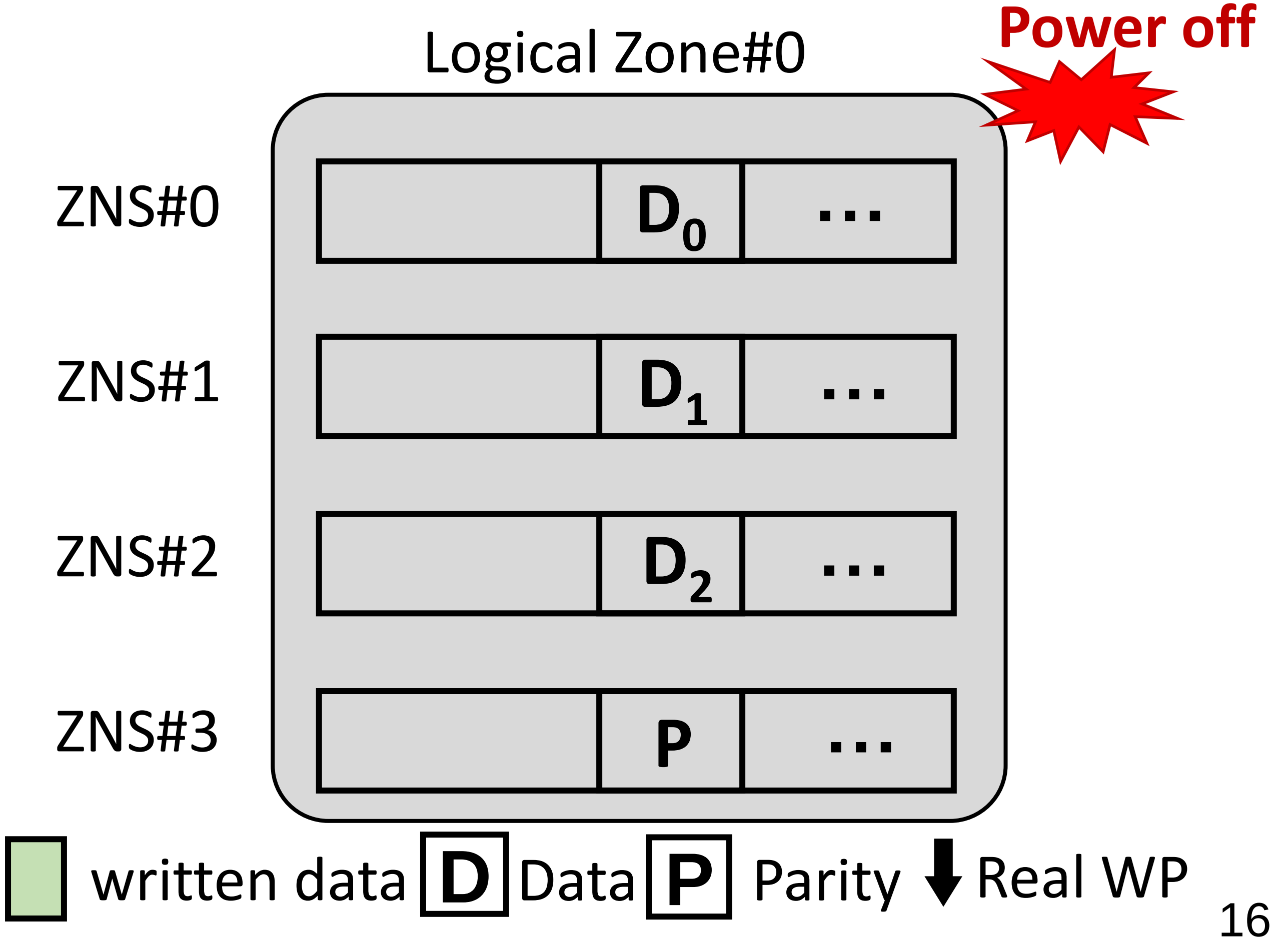
- OOB: per-page area for metadata
- Filled with 0 at first
- Set to 1 when page is written
- Back to 0 upon zone reset
- Space consumption: 1 bit/page



Recovery from a power-off event

Data consistency: No write holes in stripes, written data must be consecutive

Parity consistency: Parity consistent with data



Recovery from a power-off event

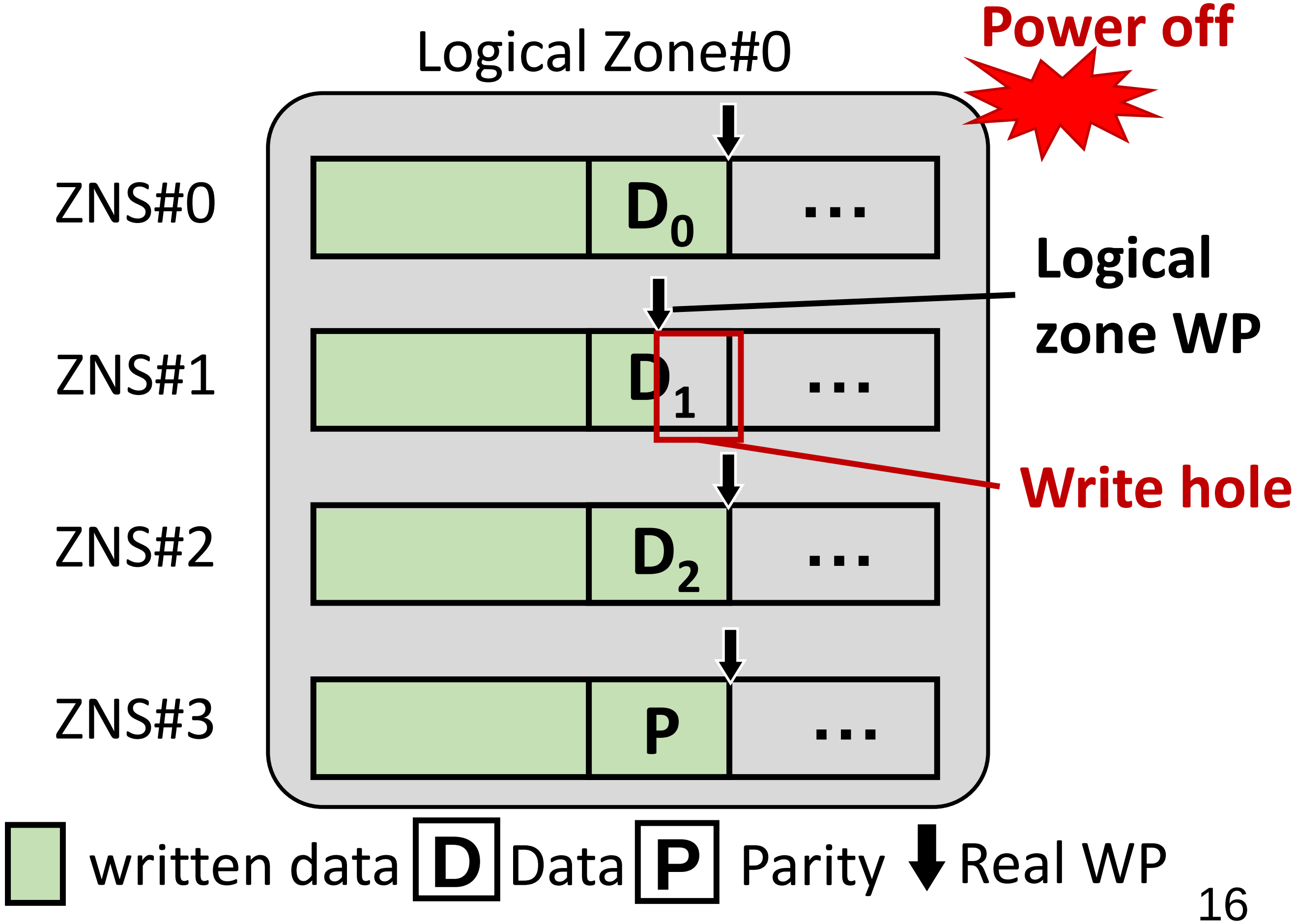
Data consistency: No write holes in stripes, written data must be consecutive

Parity consistency: Parity consistent with data

Step 1: Querying zone states

Step 2: Calculating Real WPs

- Physical zones Real WP
- Logical zone Real WP



Recovery from a power-off event

Data consistency: No write holes in stripes, written data must be consecutive

Parity consistency: Parity consistent with data

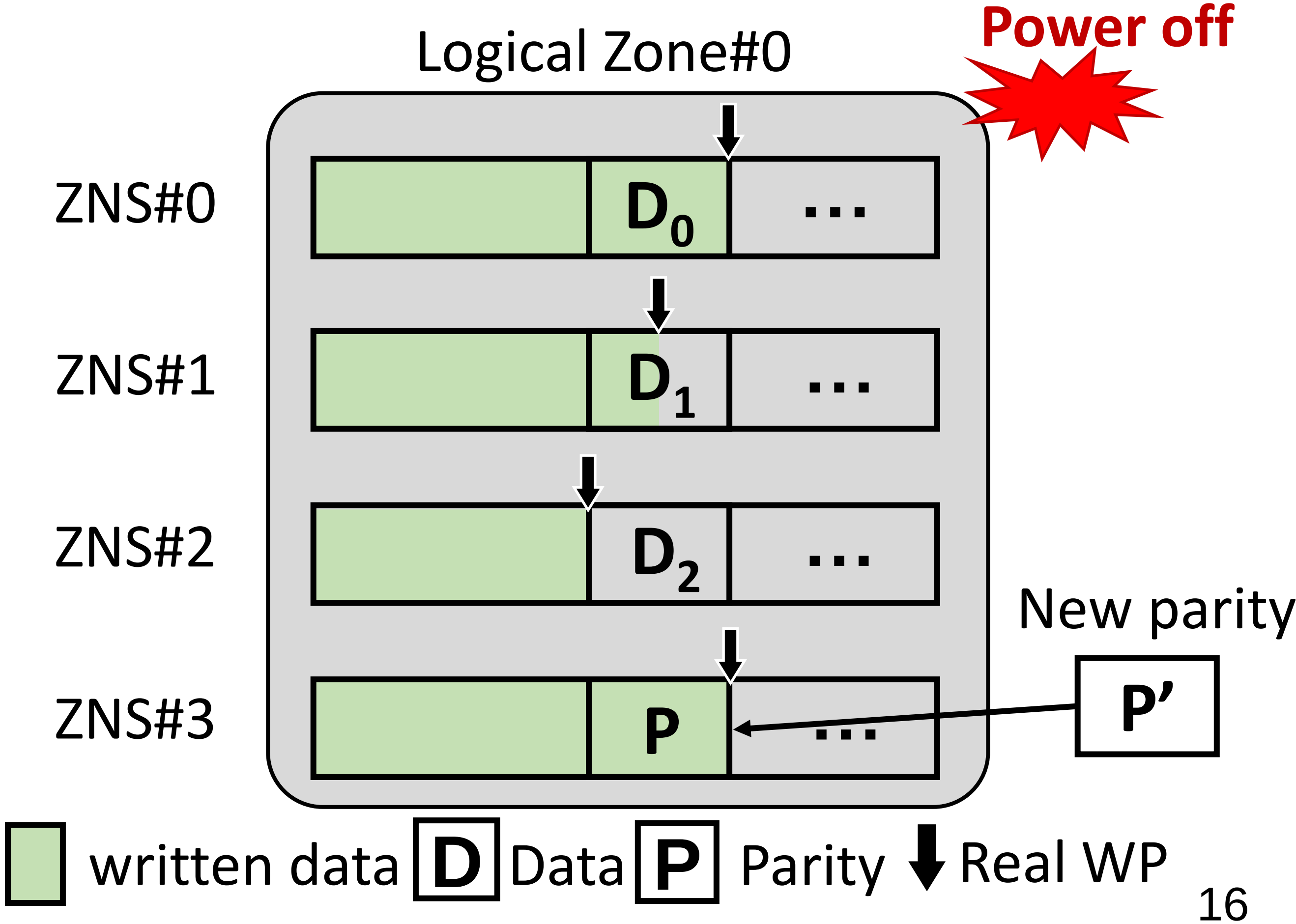
Step 1: Querying zone states

Step 2: Calculating Real WPs

- Physical zones Real WP
- Logical zone Real WP

Step 3: Synchronizing data & parity

- Overwriting new parity with ZRWA



Evaluation overview

Testbeds:

- Large-zone ZNS: (3+1) RAID-5 composed of 4 Western Digital ZN540
- Small-zone ZNS: (6+1) RAID-5 emulated by 7 ZNS SSDs via NVMeVirt

Micro Benchmarks:

- Read / Write / Mixed traces
- Real-world workloads: YCSB / TPC-C / SNIA traces

Application Benchmarks:

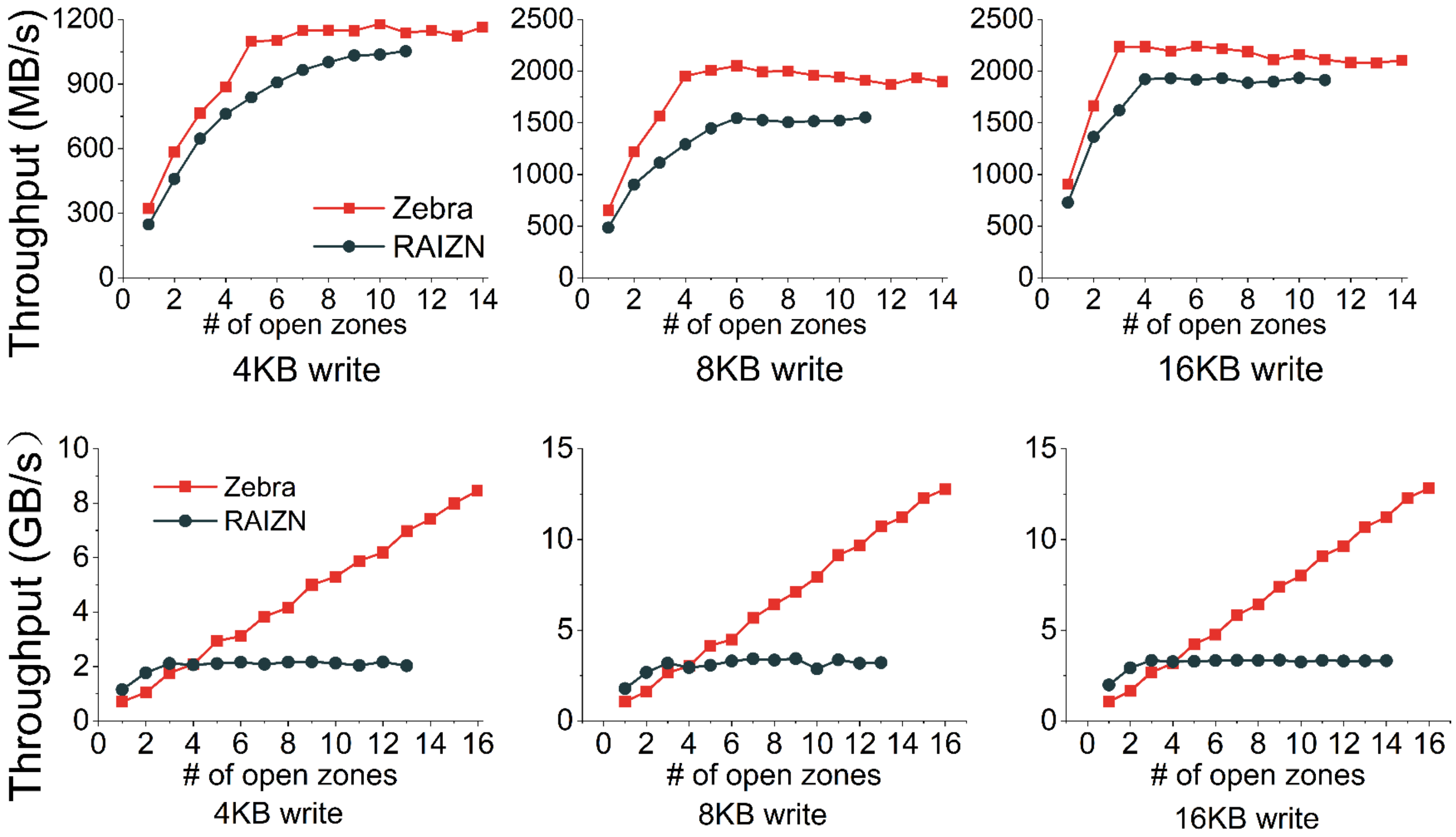
- RocksDB with db_bench

Peer system:

- RAIZN [ASPLOS’23]

Read & write performance

- Sequential & random read workloads: **similar** throughput
- Sequential write workloads: 4KiB / 8KiB / 16KiB

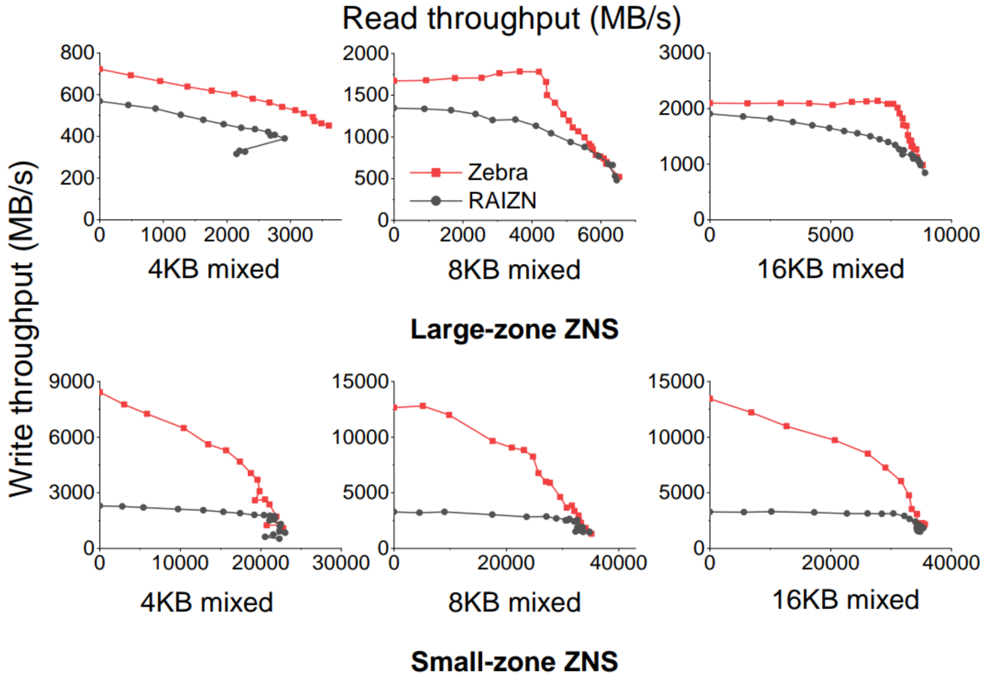


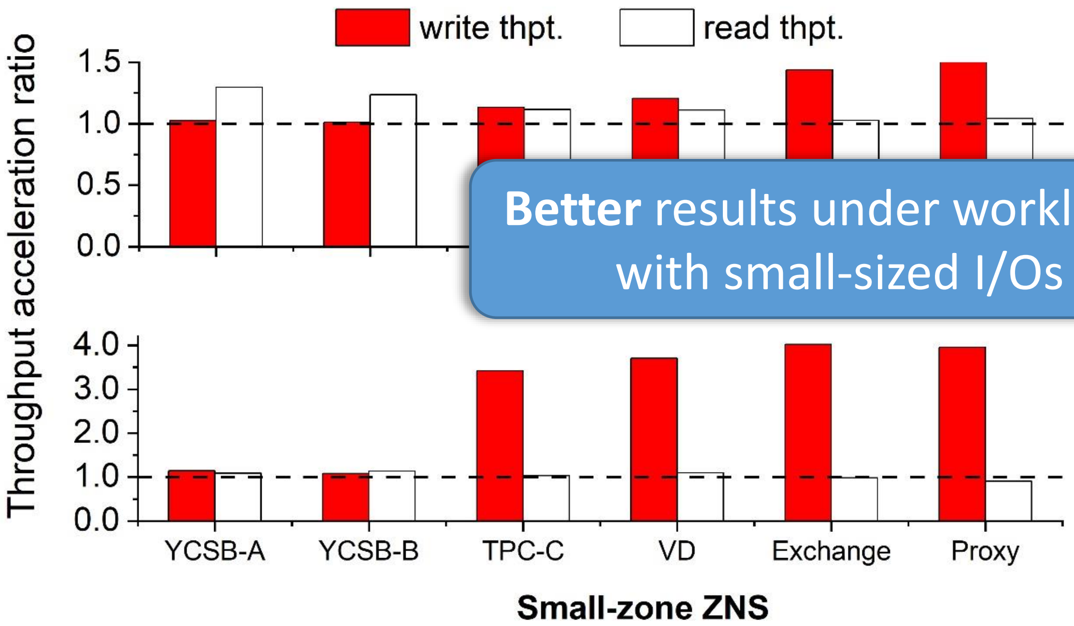
Large-zone ZNS SSD arrays: **6%-51%** throughput ⬆

Small-zone ZNS SSD arrays: **3.3X-4.2X** throughput ⬆

Performance under mixed & real-world traces

- Read-write-mixed workloads: varying write ratios
- Real-world traces:
  - captured from real applications, replaying on ZNS RAID systems





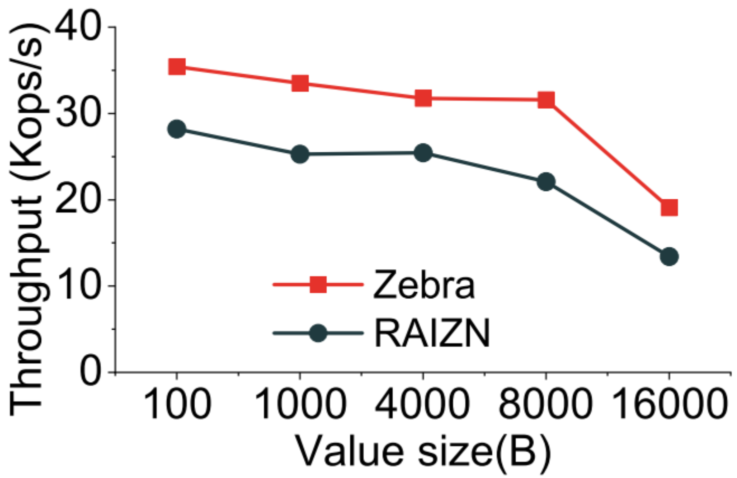
Better results under workloads with small-sized I/Os

2.2X write throughput ↑ under mixed workloads

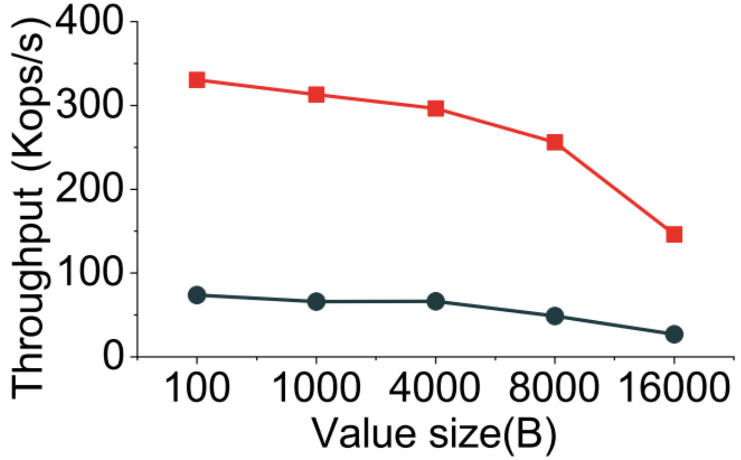
2.1X write throughput ↑ under real-world workloads

Application benchmarks

- Building RocksDB on ZNS RAID
- *fillsync* workload with db_bench



Large-zone ZNS



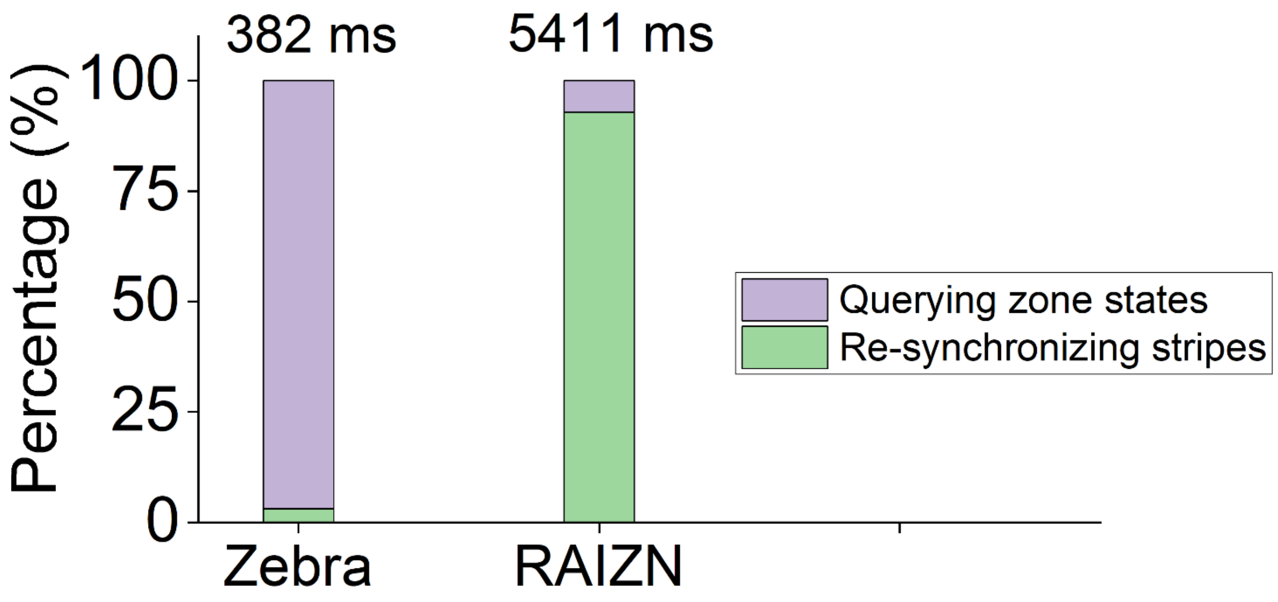
Small-zone ZNS

1.3X throughput ↑ on large-zone SSDs

4.9X throughput ↑ on small-zone SSDs

Recovery performance

- Recovery latency from a power-off event
- Zebra avoids the process of **reading metadata zones** during the recovery.



14.2X recovery acceleration on Zebra

Conclusions

- Problem
  - **Low write performance** of RAID systems based on ZNS SSDs
  - **In-place updates** in PPU is incompatible with **append-only** semantic in ZNS.
- Observation
  - Processing PPUs behaves like **a sliding window**, a natural fit for ZRWA feature.
- Key idea
  - Holding parity chunks within ZRWA for in-place PPUs
- Techniques
  - **Zebra:** a novel architecture of ZNS RAID **enabled by ZRWA**
  - **Lightweight** metadata management to locate WPs with **OOB**
  - **Recovery** process from a power-off event