HotStorage'22

What You Can't Forget: Exploiting Parallelism for Zoned Namespaces

Hanyeoreum Bae, Jiseon Kim, Miryeong Kwon, Myoungsoo Jung

Computer Architecture and MEmory systems Laboratory

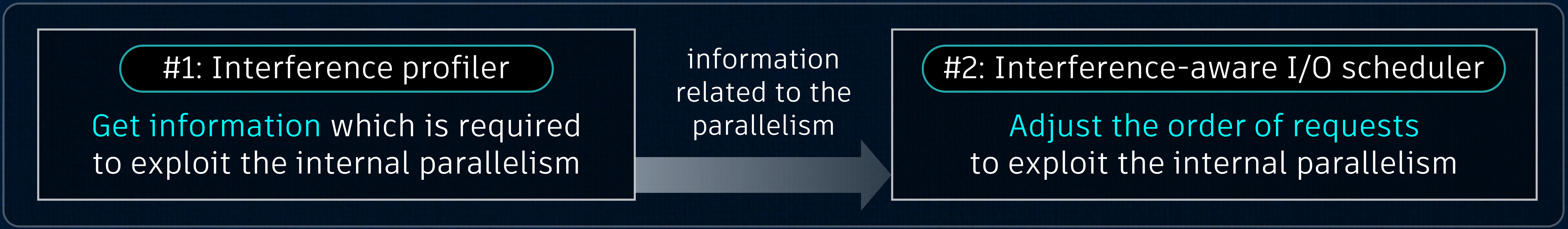
KAIST CAMEL

High-Level Summary

We analyze the problem of zoned namespaces (ZNS), by using two production ZNS SSDs

Problem: It is hard to exploit the internal parallelism of SSDs
(Reason: ZNS does not provide abstraction required to manage the parallelism)

We propose two simple modules



#1: Interference profiler

Get information which is required to exploit the internal parallelism

information related to the parallelism

#2: Interference-aware I/O scheduler

Adjust the order of requests to exploit the internal parallelism

1. Background – Zoned Namespaces

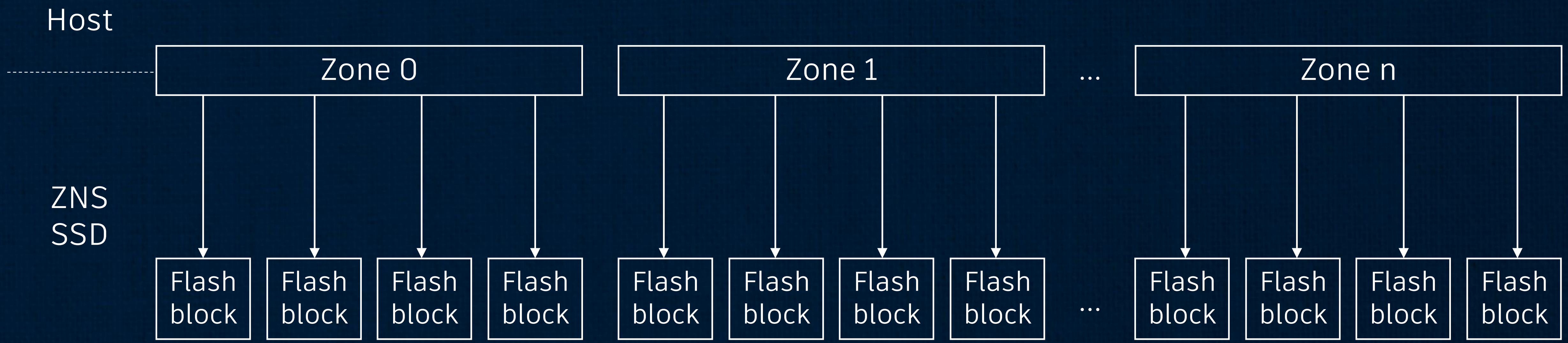
2. Challenge

3. Solution

4. Evaluation

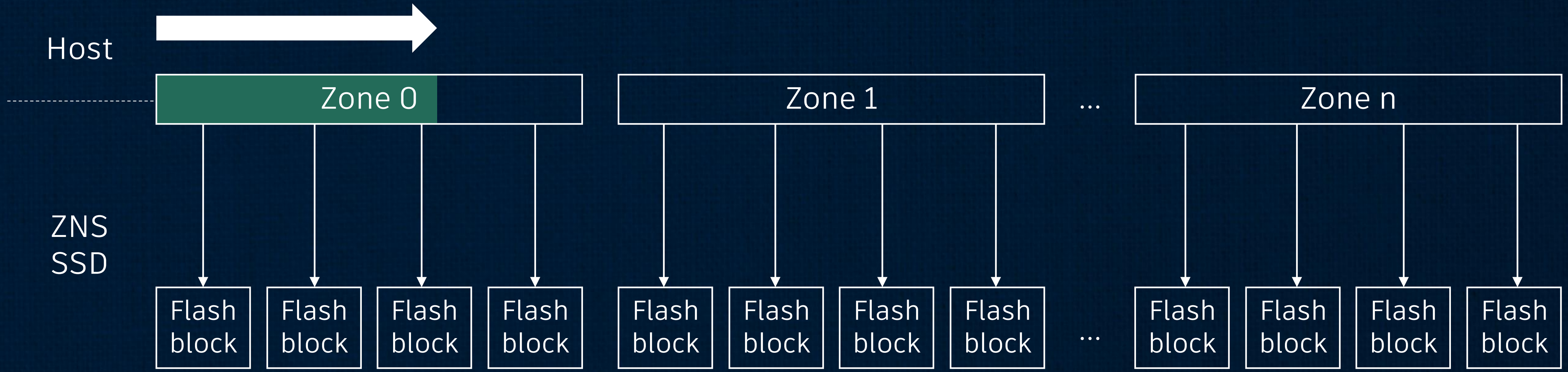
What Is ZNS?

- Zoned namespaces (ZNS): Emerging storage interface
  - Divide logical address space into multiple zones
  - In general, each zone is mapped to one or more flash blocks



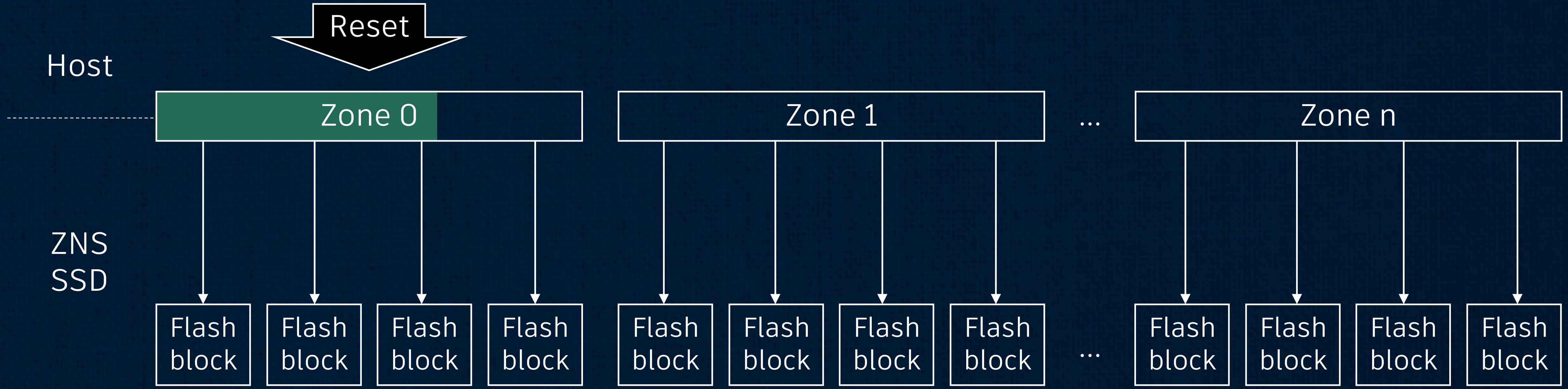
What Is ZNS?

- Two constraints on each zones
  - Constraint #1: Sequential write

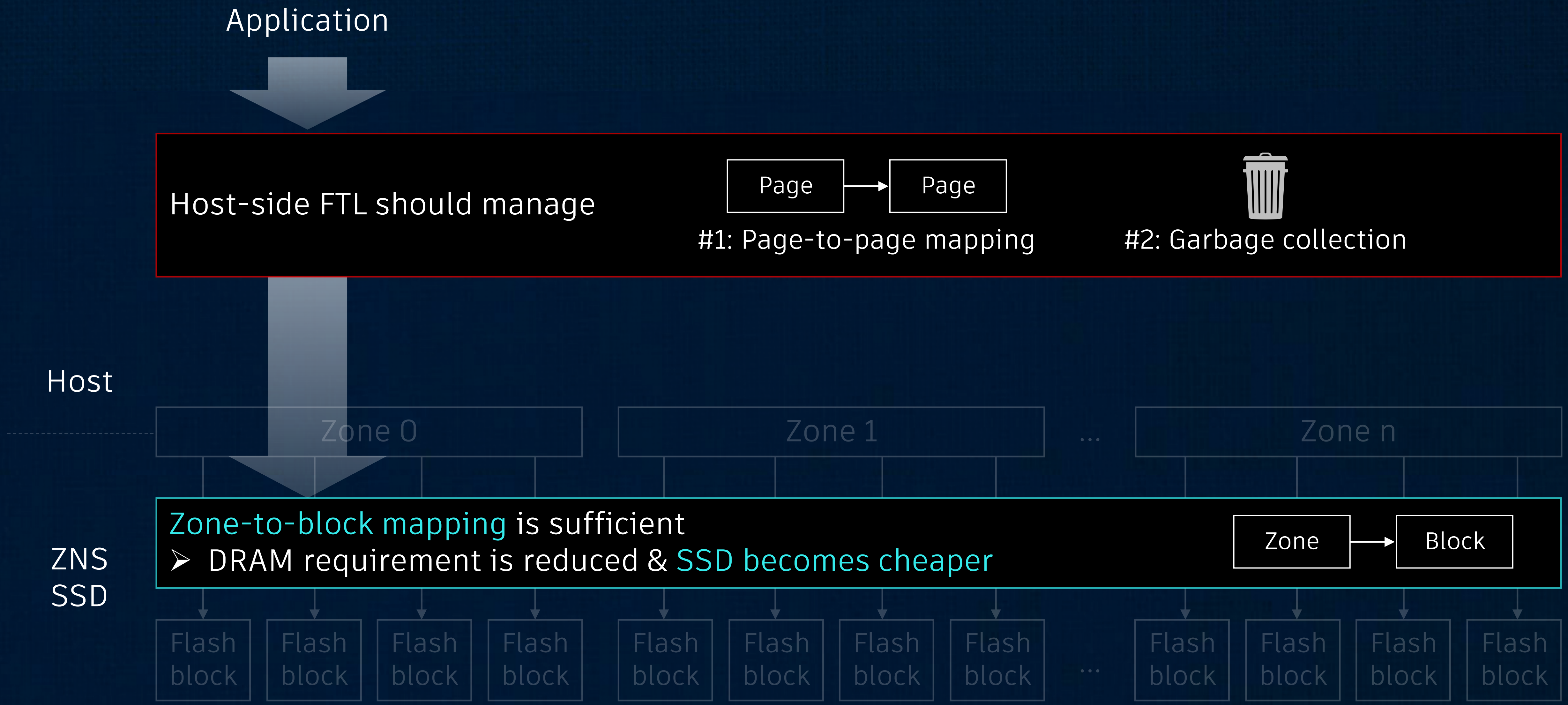


What Is ZNS?

- Two constraints on each zones
  - Constraint #1: Sequential write
  - Constraint #2: Erase(reset)-before-write



What Is ZNS?



Host-side FTL should manage

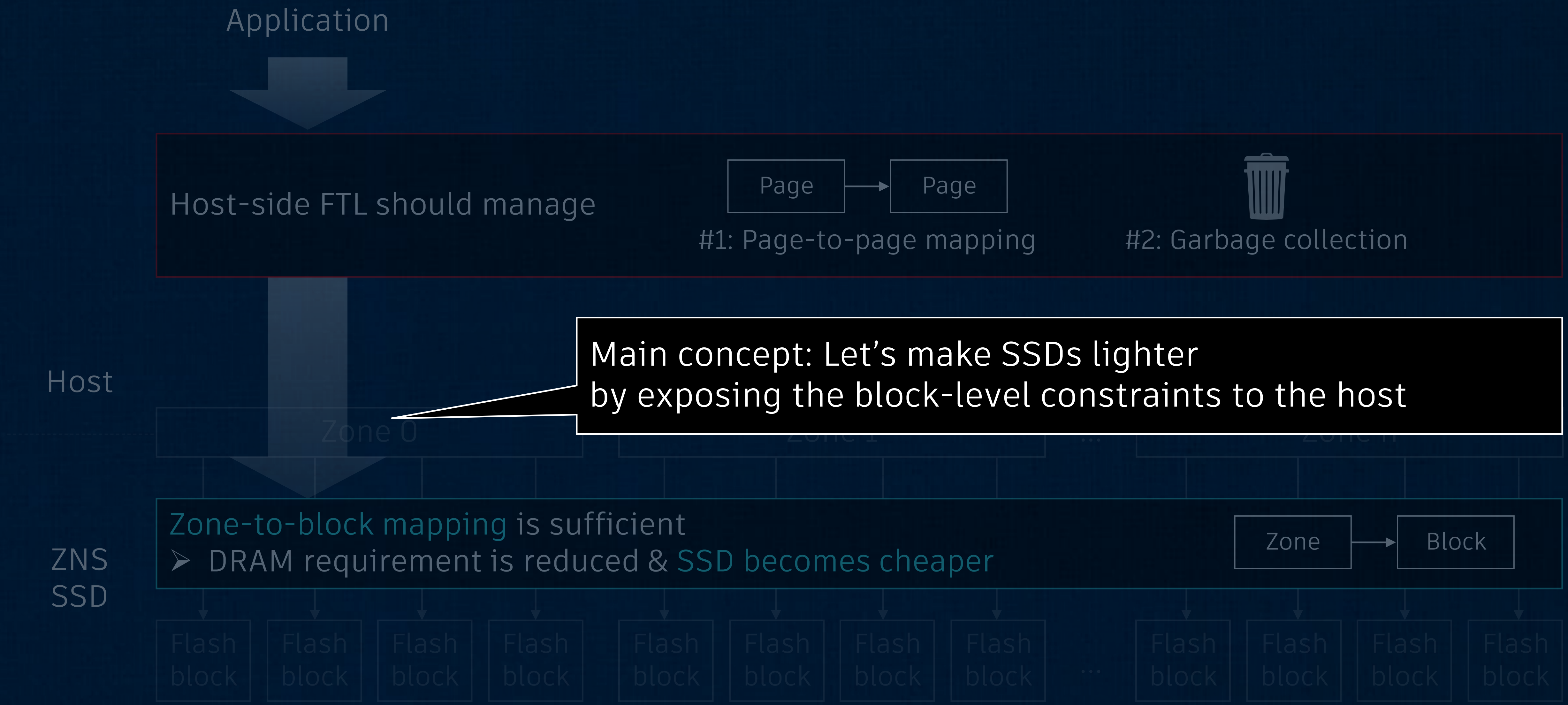
#1: Page-to-page mapping

#2: Garbage collection

Zone-to-block mapping is sufficient

➢ DRAM requirement is reduced & SSD becomes cheaper

What Is ZNS?



Application

Host-side FTL should manage

#1: Page-to-page mapping

#2: Garbage collection

Host

Main concept: Let’s make SSDs lighter
by exposing the block-level constraints to the host

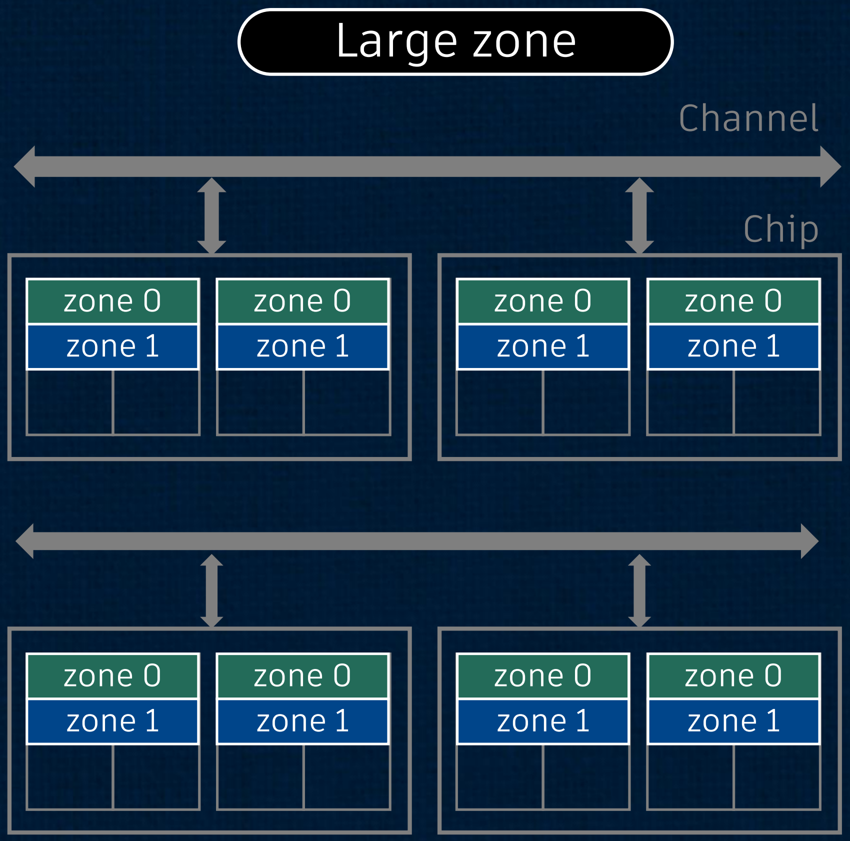
ZNS SSD

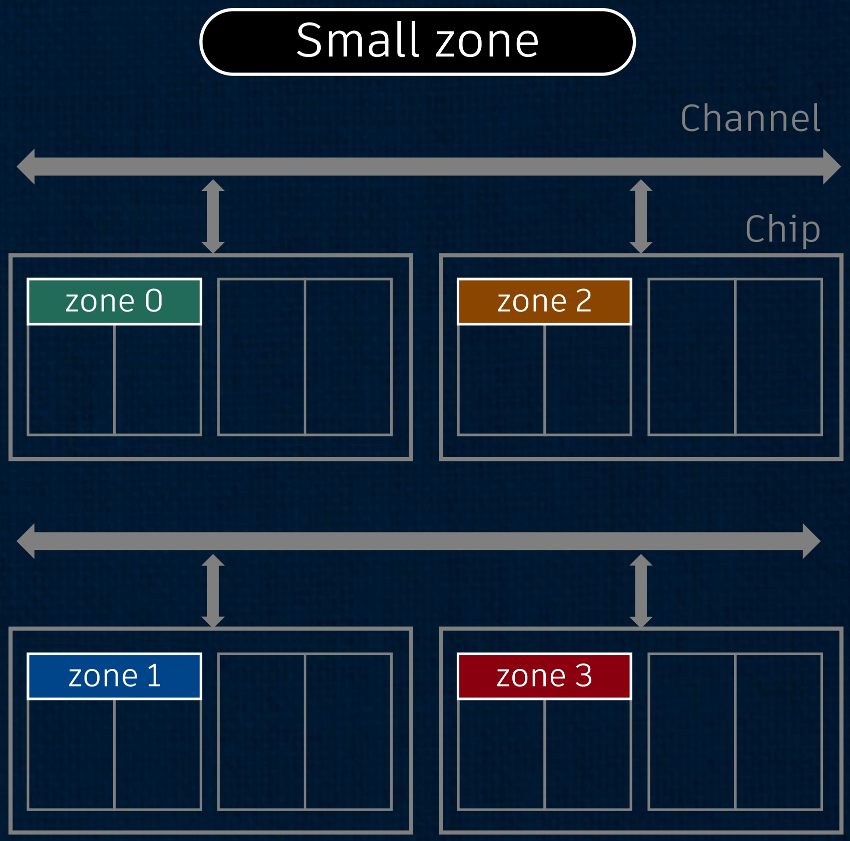
Zone-to-block mapping is sufficient

➢ DRAM requirement is reduced & SSD becomes cheaper

How Should Zones Be Configured?

- No constraint on zone size

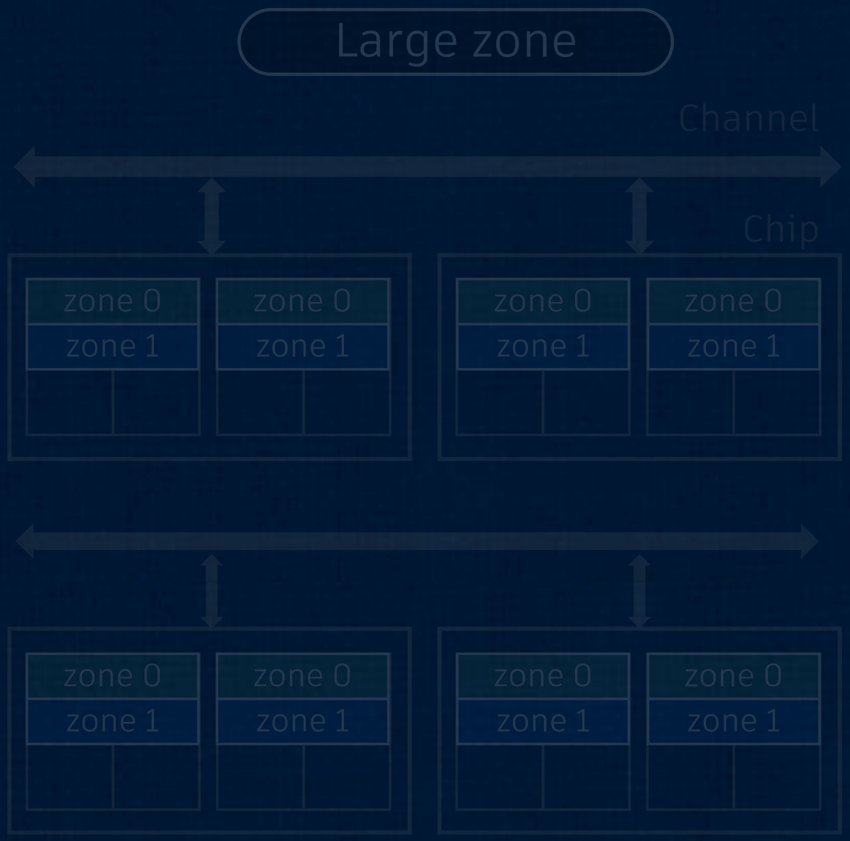


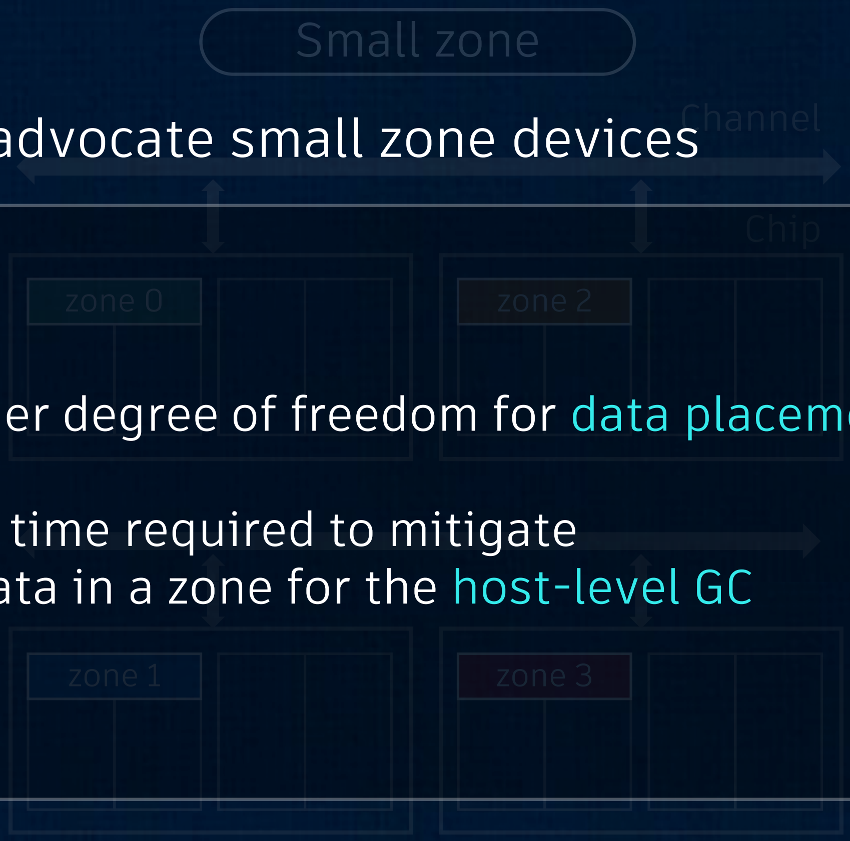


** Above figure is different from
the configuration of the production ZNS SSDs that we used in this paper

How Should Zones Be Configured?

- No constraint on zone size





➢ We advocate small zone devices

+) Higher degree of freedom for data placement

+) Less time required to mitigate
valid data in a zone for the host-level GC

1. Background – Zoned Namespaces

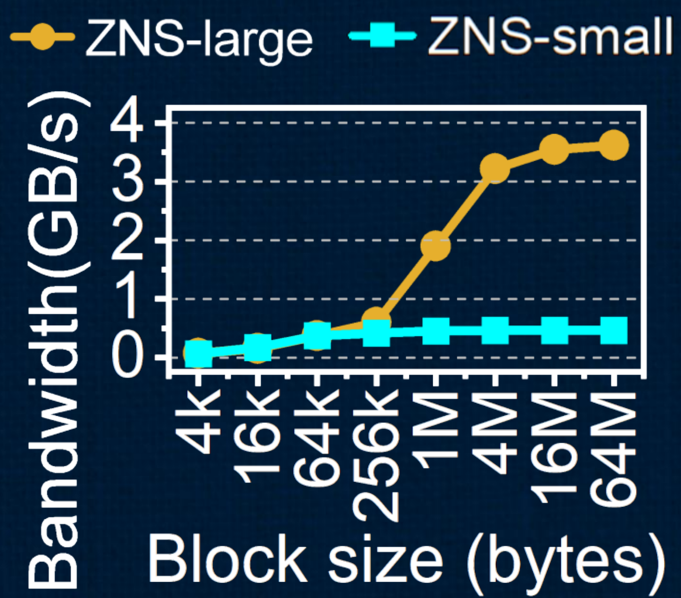
2. Challenge

3. Solution

4. Evaluation

Disadvantage of Small Zone Devices

Disadvantage:
They show much worse performance than large zone devices, when the request size is large

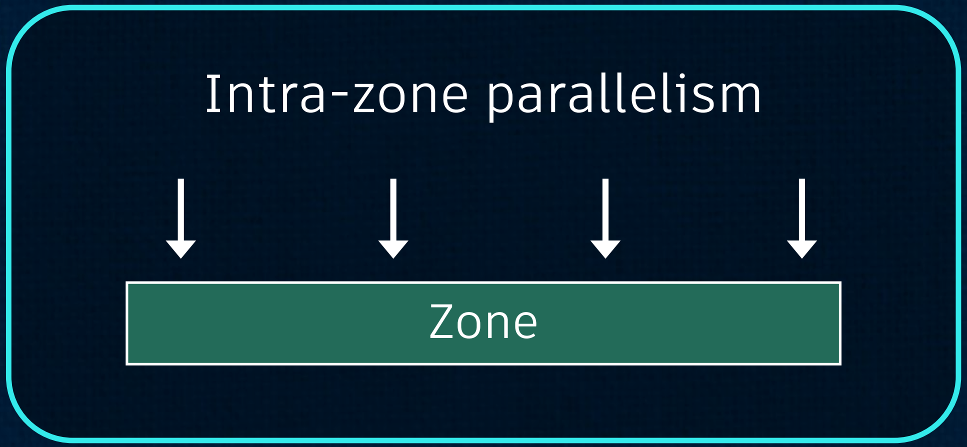


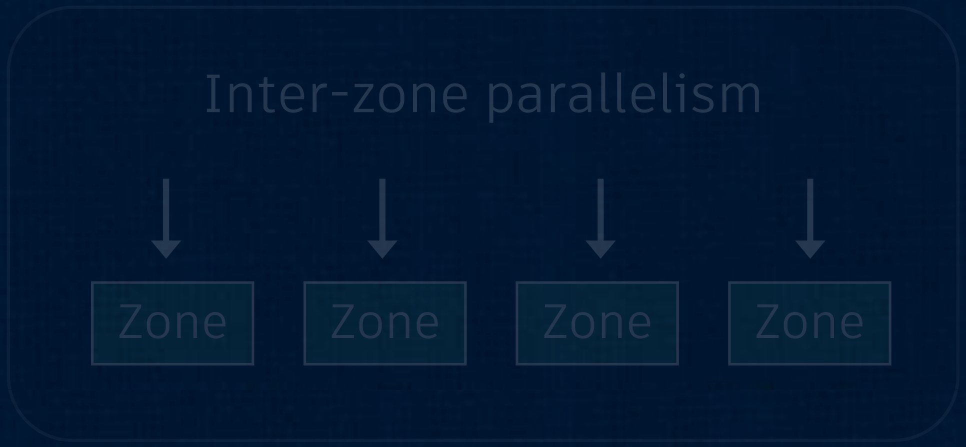
- Sequential read w/ single process
- Used two production ZNS SSDs
  - ZNS-large: ZNS SSD with large zones (2.18GB/zone)
  - ZNS-small: ZNS SSD with small zones (96MB/zone)

** Both SSDs utilize the same flash package
** PCIe3.0 x4 (max. bandwidth = 3.94GB/s)

➢ Reason: Internal parallelism (especially, intra-zone parallelism)

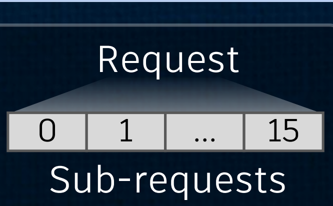
Internal Parallelism in ZNS SSDs



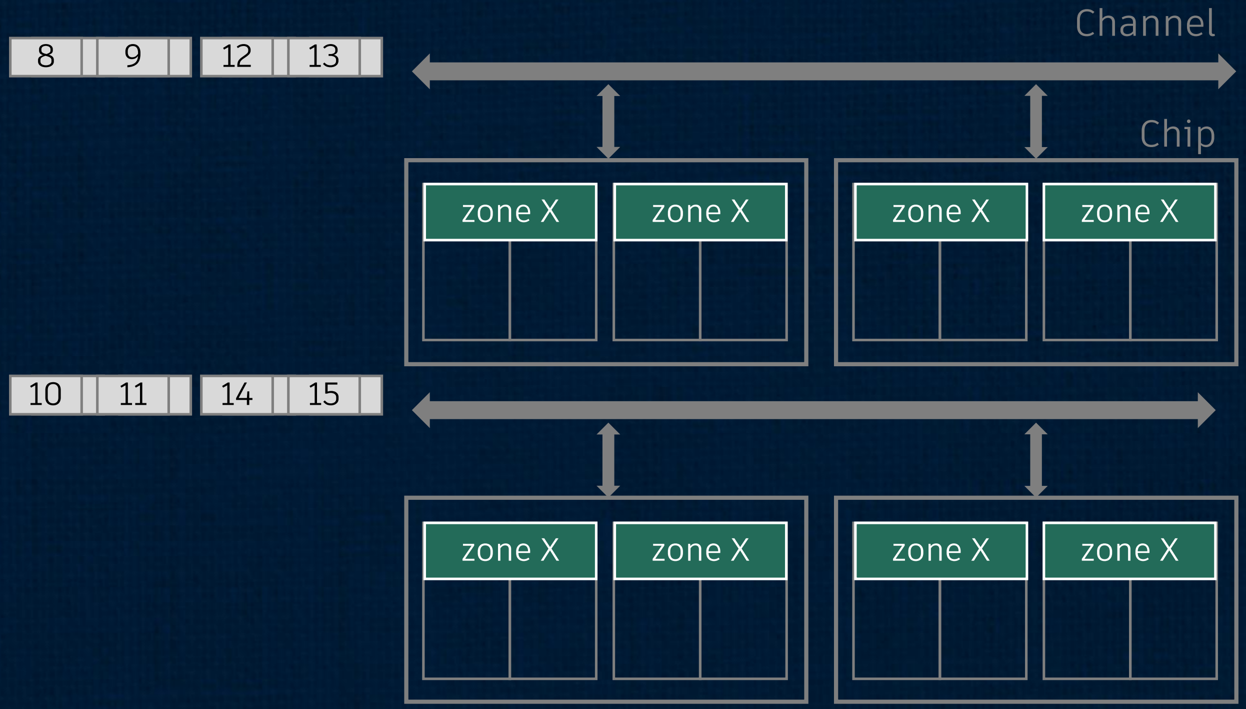


Intra-zone Parallelism

Way to exploit the intra-zone parallelism
: Increase the request size

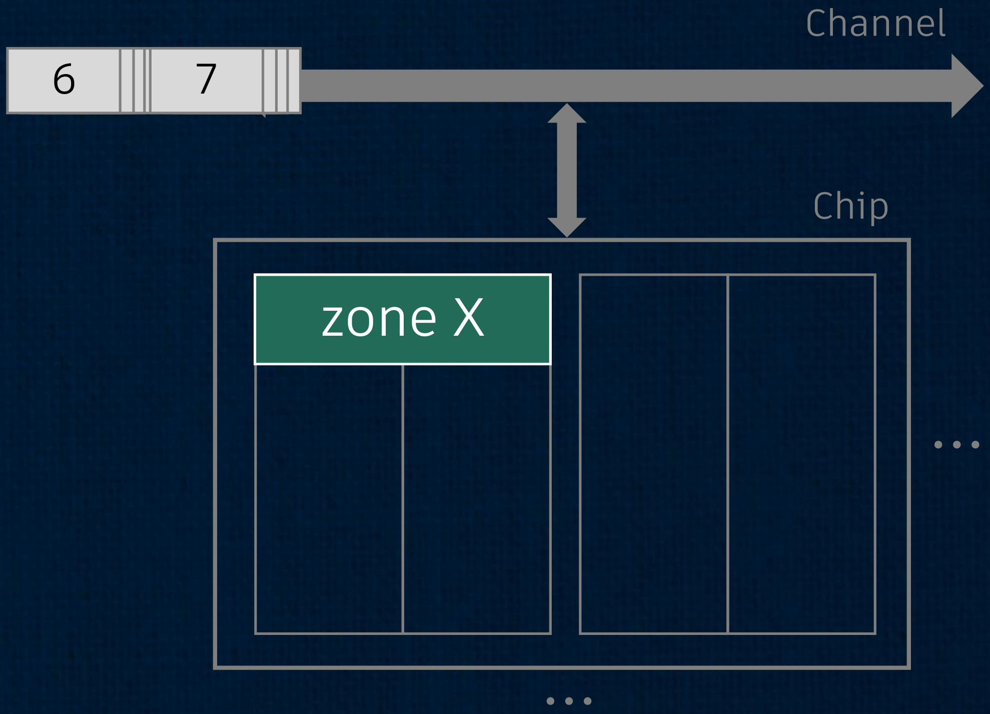


Large zone
(Intra-zone parallelism ↑)



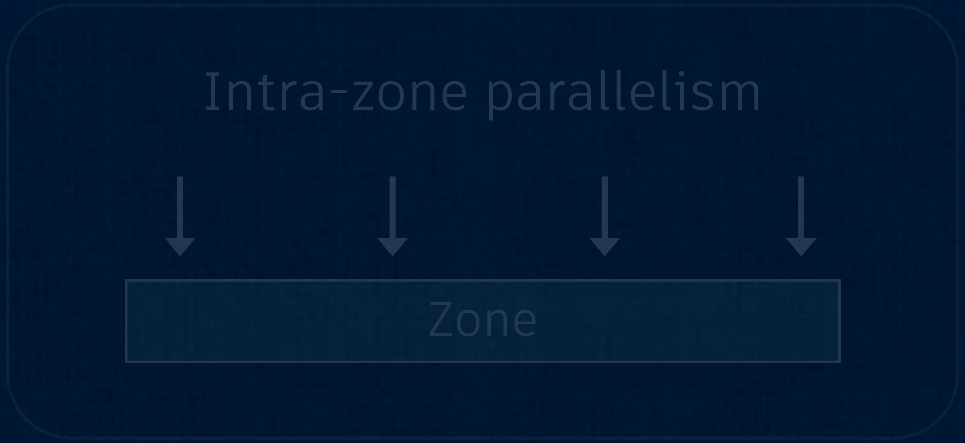
Able to get a high performance
by increasing the request size

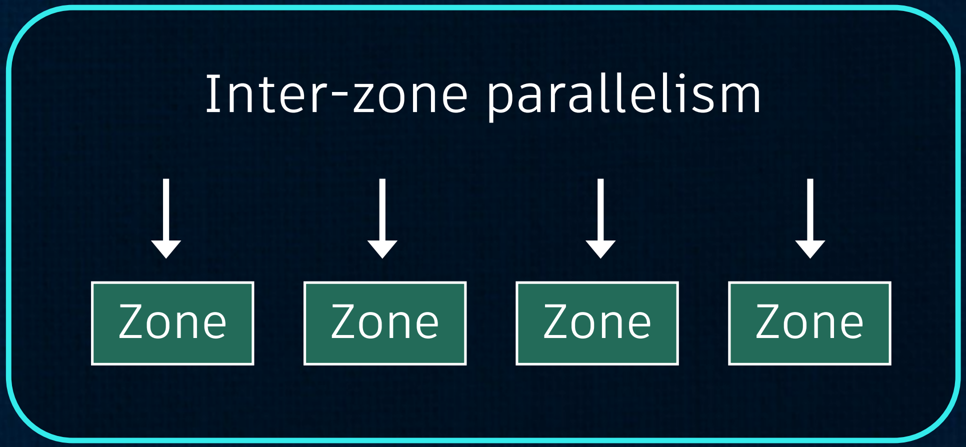
Small zone
(Intra-zone parallelism ↓)



Unable to get a high performance
by increasing the request size

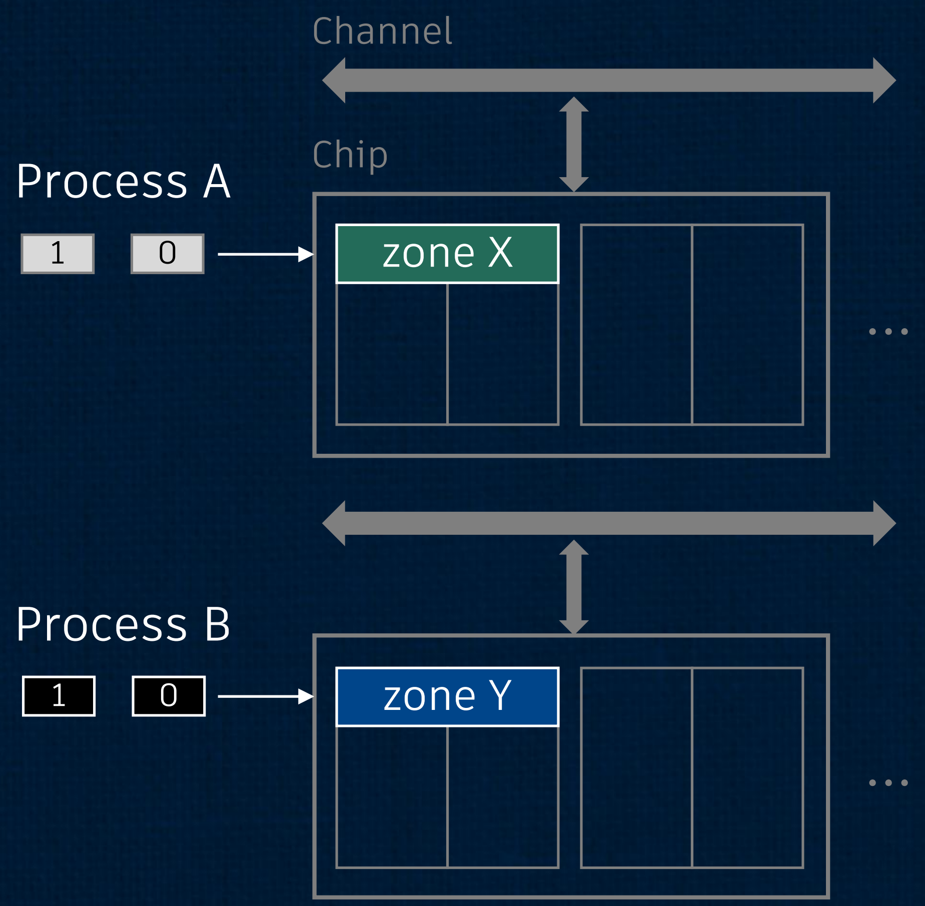
Internal Parallelism in ZNS SSDs





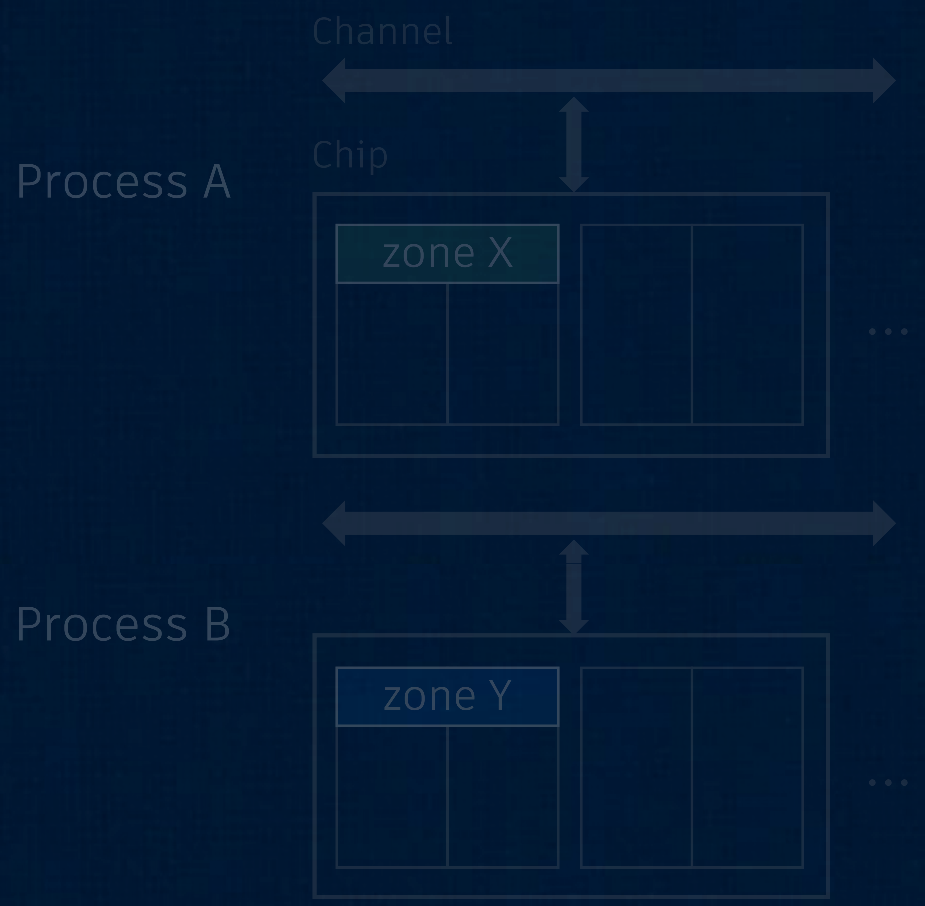
Inter-zone Parallelism

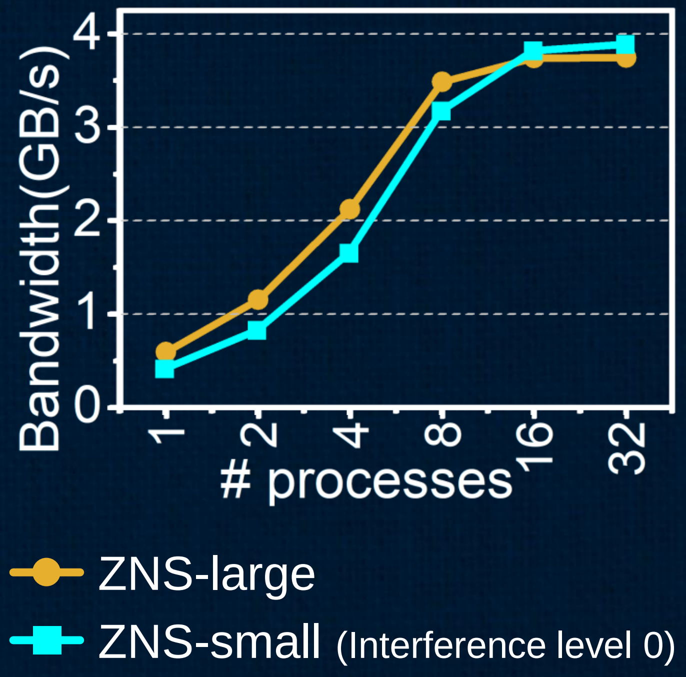
Way to exploit the inter-zone parallelism
: Send requests to different zones at the same time



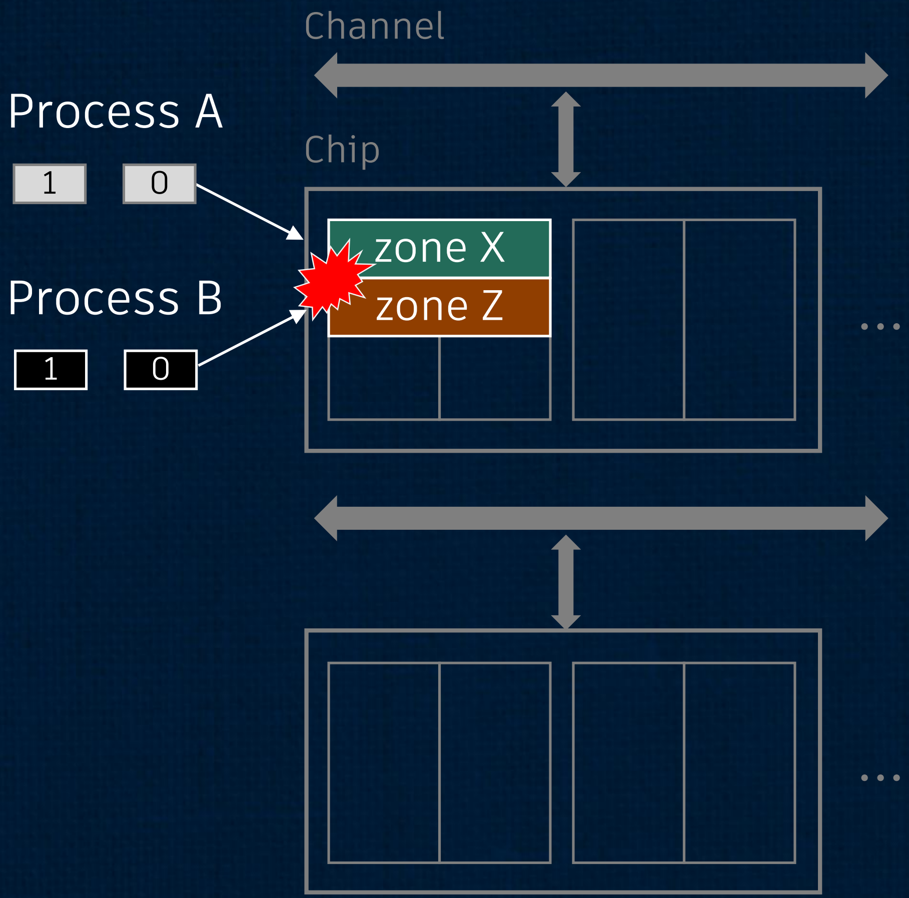
Inter-zone Parallelism

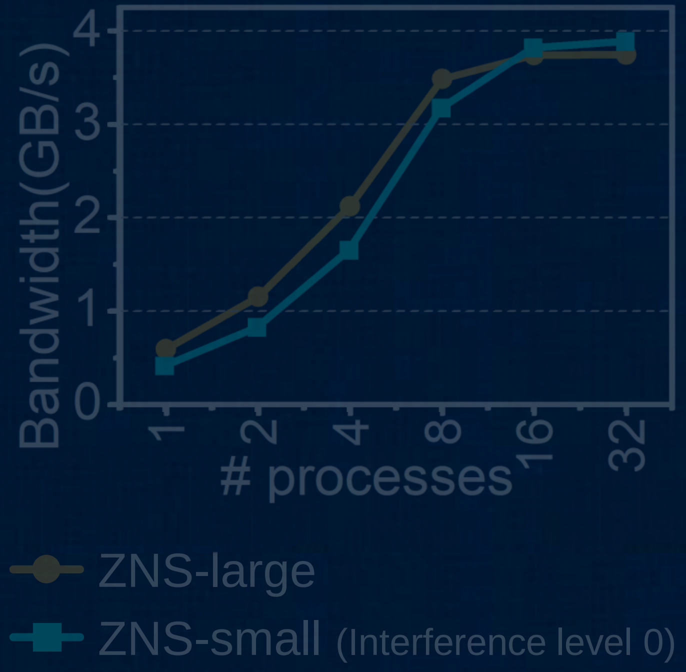
Way to exploit the inter-zone parallelism
: Send requests to different zones at the same time





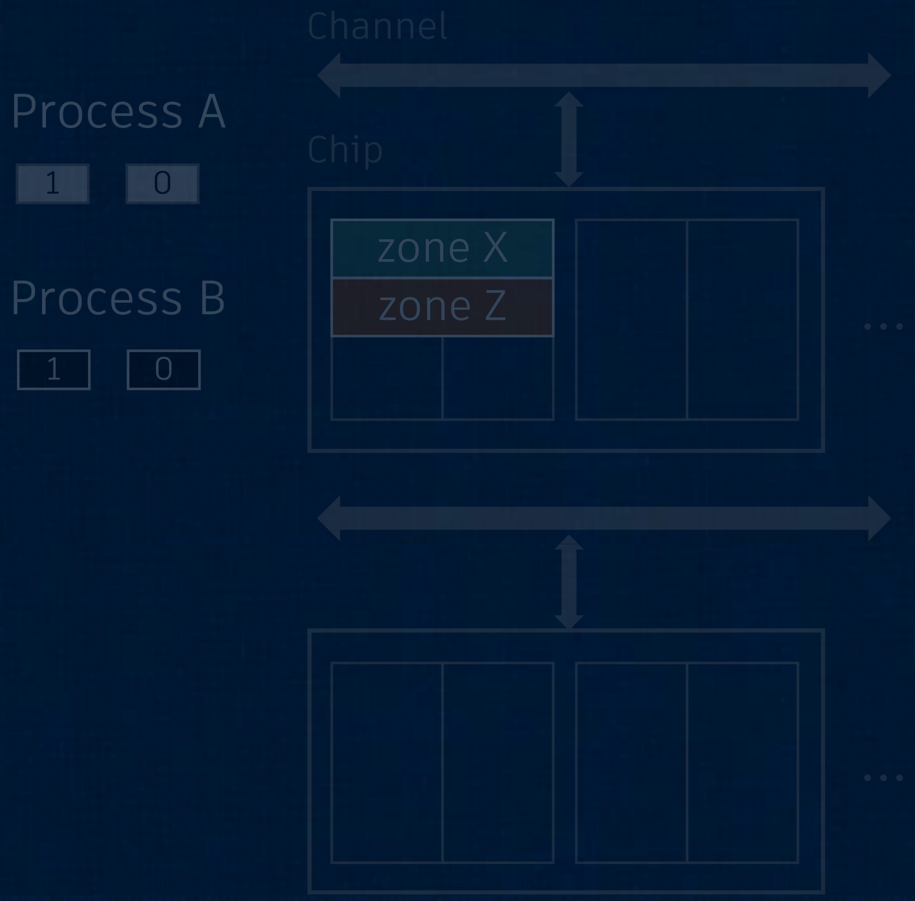
Challenge: Inter-Zone Interference

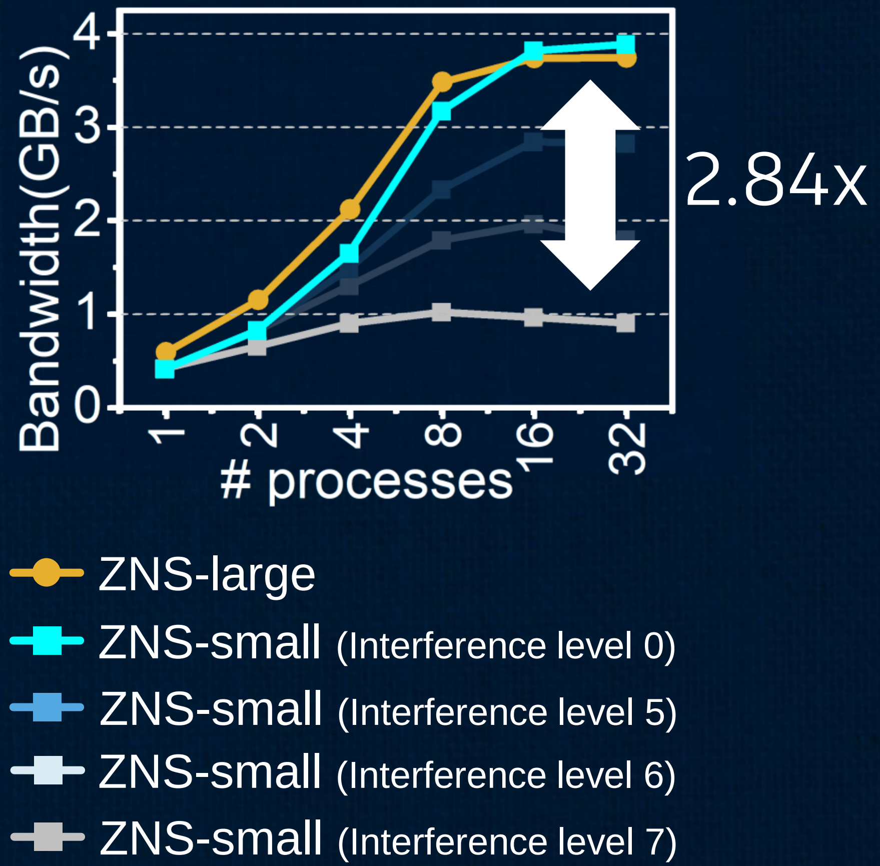




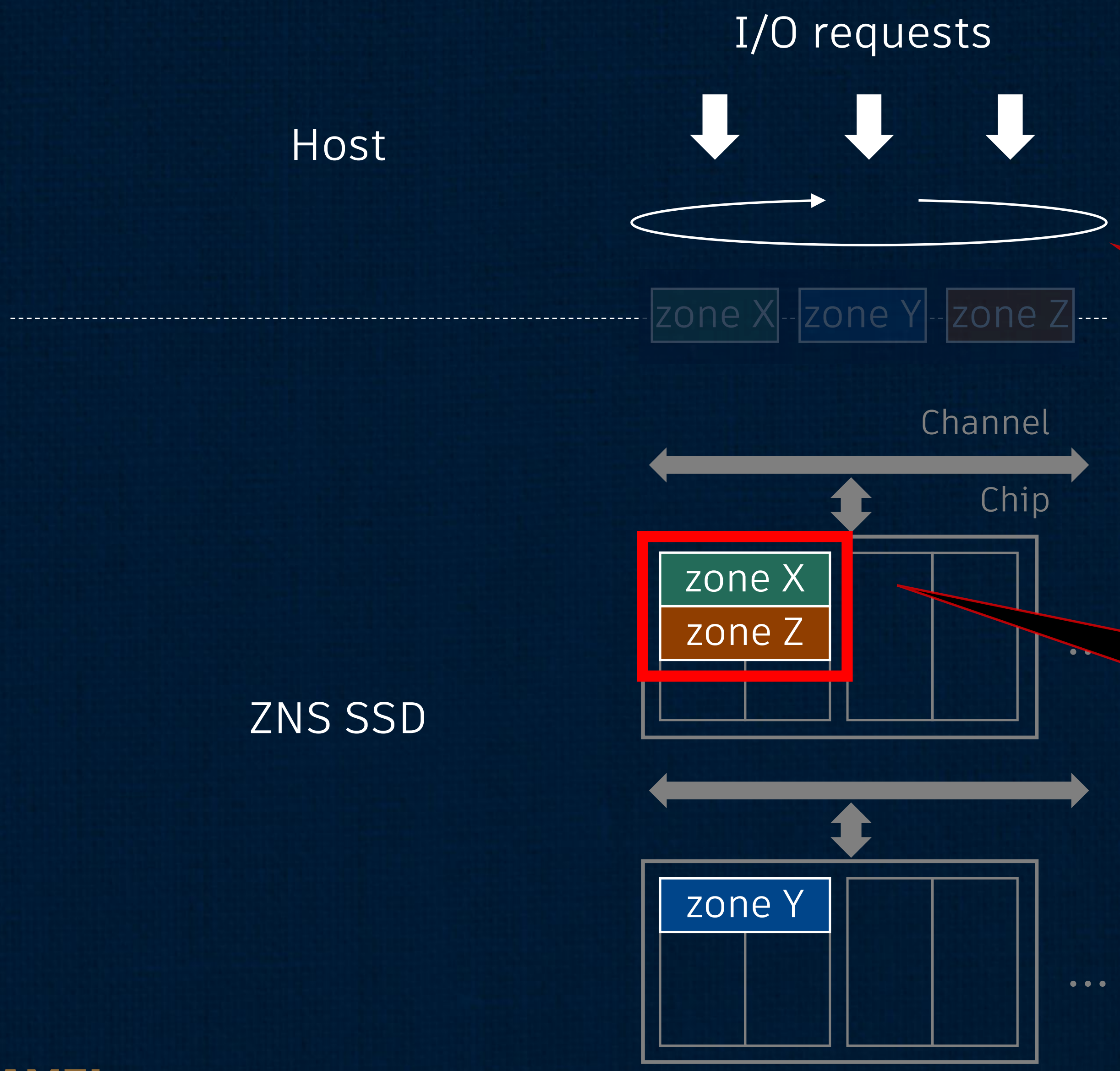
Challenge: Inter-Zone Interference

The host must be aware of the inter-zone interference, since it can cause a serious performance degradation





Requirements to Prevent the Interference

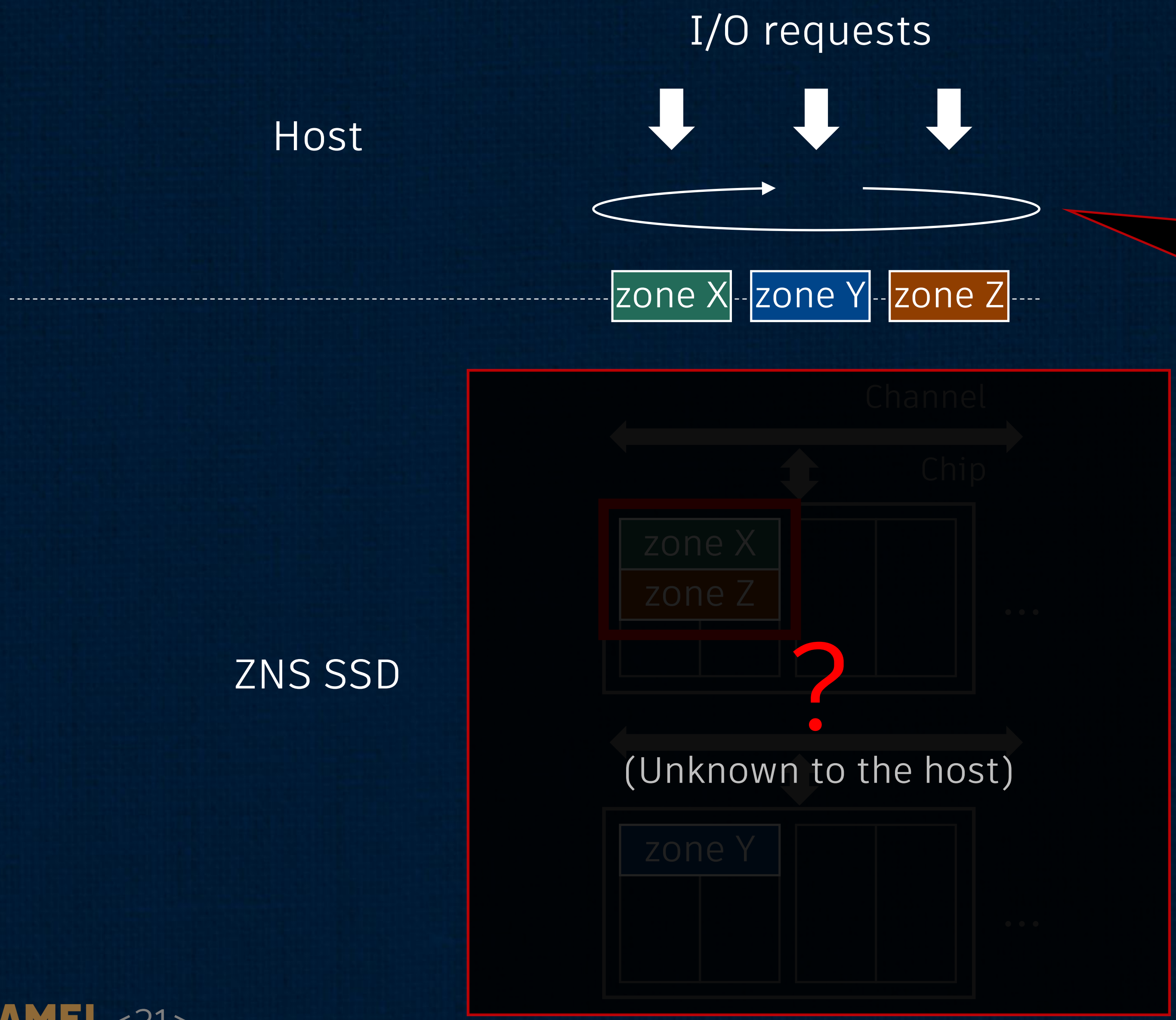


#2: The host needs to schedule I/O requests appropriately, based on the information

(e..g., Prevent zone X and zone Z from being accessed simultaneously)

#1: The host must recognize the zone-to-zone relationship generating the interference

Challenge: Not Enough Abstraction



The host cannot prevent the interference since there is no information related to the interference/parallelism

Problem:
ZNS does not provide enough abstraction about hardware configuration

1. Background – Zoned Namespaces

2. Challenge

3. Solution

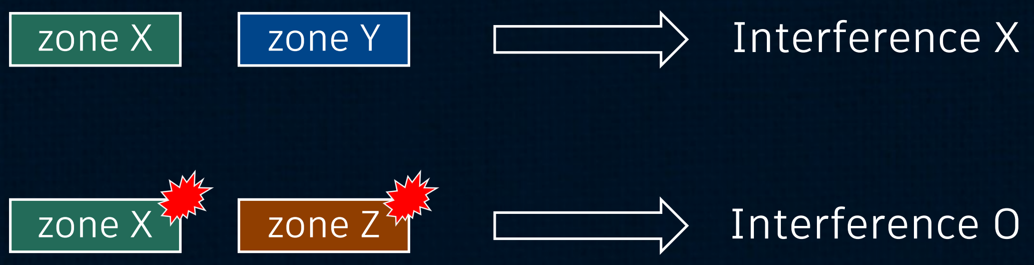
4. Evaluation

Overview

Problem:

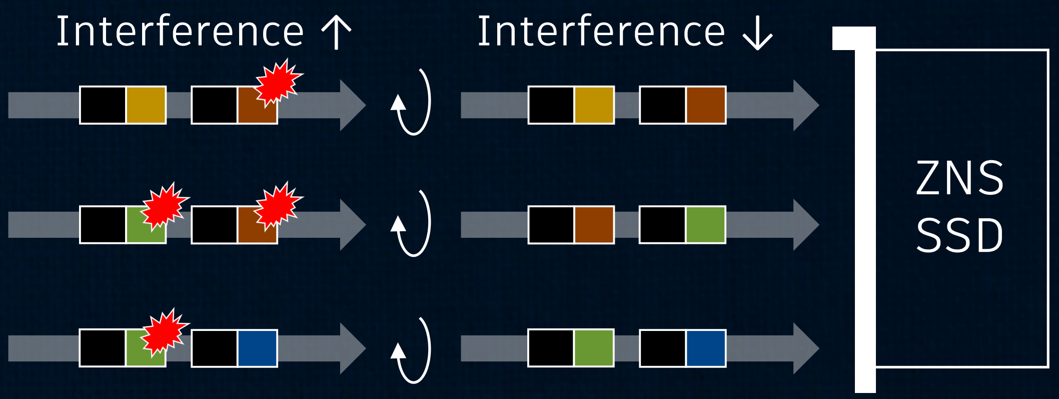
Not enough abstraction about hardware ➔ Cannot prevent performance degradation due to the interference

Interference profiler



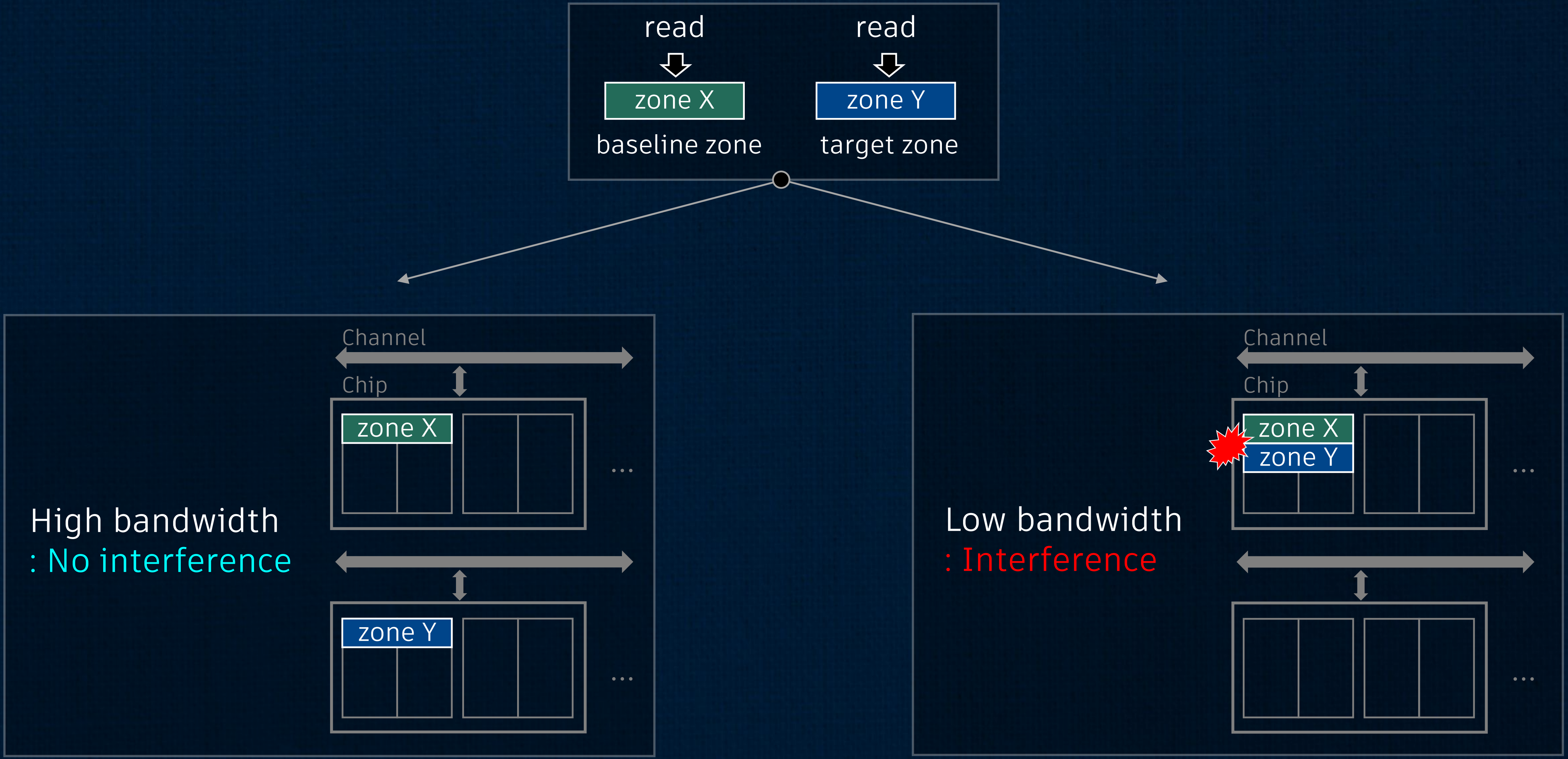
Detect zone-to-zone relationships generating the interference

Interference-aware I/O Scheduler



Adjust the order of requests to reduce the level of inter-zone interference

Interference Profiler – Main Idea



read → zone X

baseline zone

read → zone Y

target zone

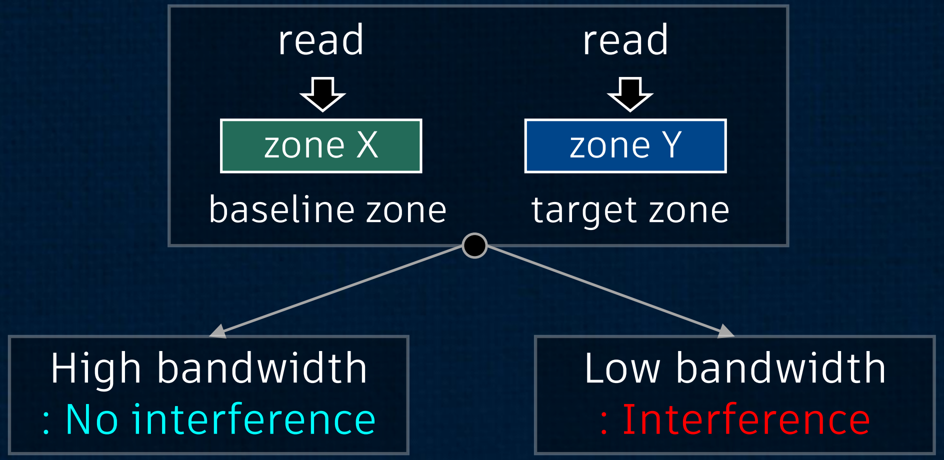
High bandwidth

: No interference

Low bandwidth

: Interference

Interference Profiler



read

zone X

baseline zone

read

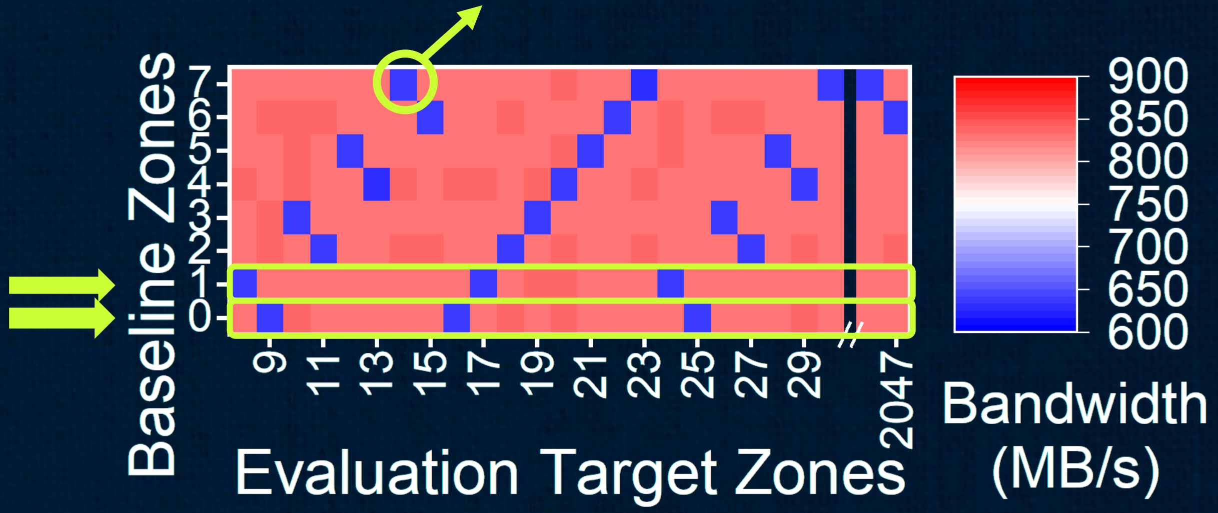
zone Y

target zone

High bandwidth
: No interference

Low bandwidth
: Interference

Low bandwidth: Interference



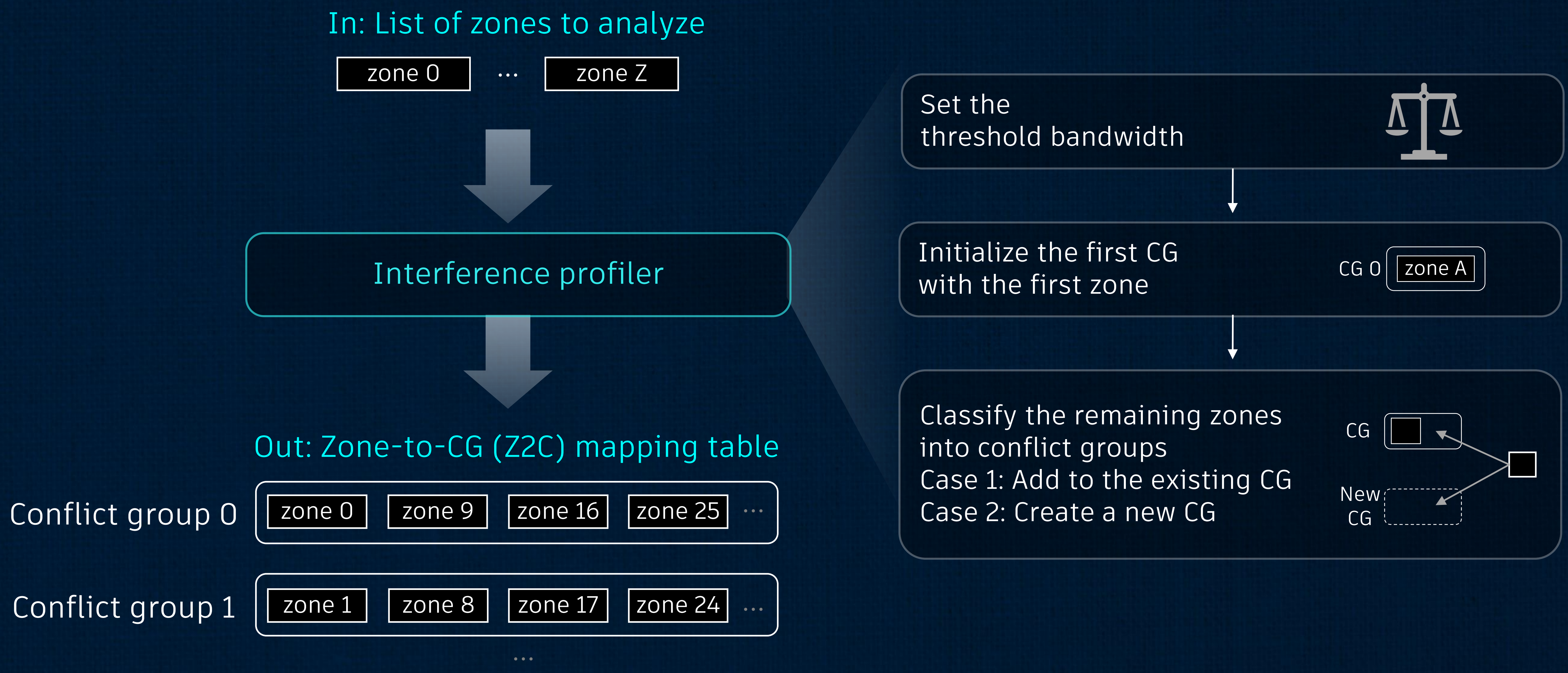
Conflict group 0: zone 0 | zone 9 | zone 16 | zone 25 | …

Conflict group 1: zone 1 | zone 8 | zone 17 | zone 24 | …

…

*** Conflict group: Set of zones interfering with each other*

Interference Profiler



In: List of zones to analyze

zone 0 ··· zone Z

Interference profiler

Out: Zone-to-CG (Z2C) mapping table

Conflict group 0	zone 0	zone 9	zone 16	zone 25	···
Conflict group 1	zone 1	zone 8	zone 17	zone 24	···

···

Set the threshold bandwidth

Initialize the first CG with the first zone

Classify the remaining zones into conflict groups

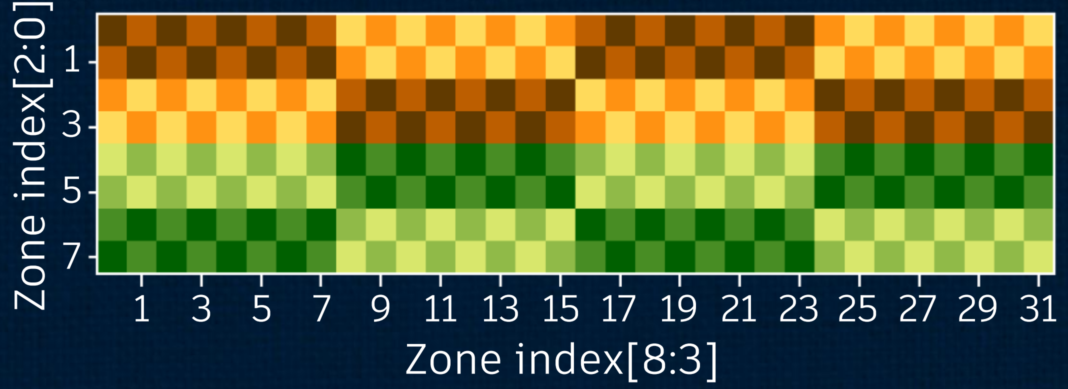
Case 1: Add to the existing CG

Case 2: Create a new CG

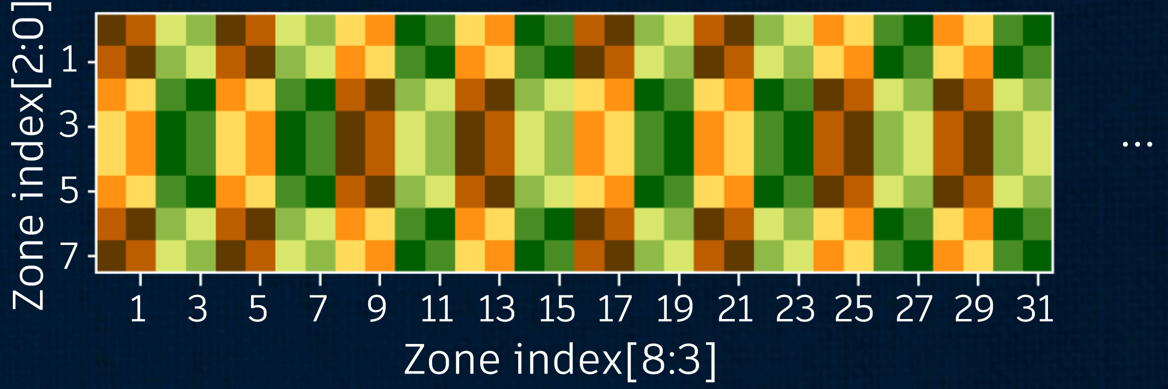
Interference Profiler - Results

Conflict group 0 1 2 3 4 5 6 7

Case #1



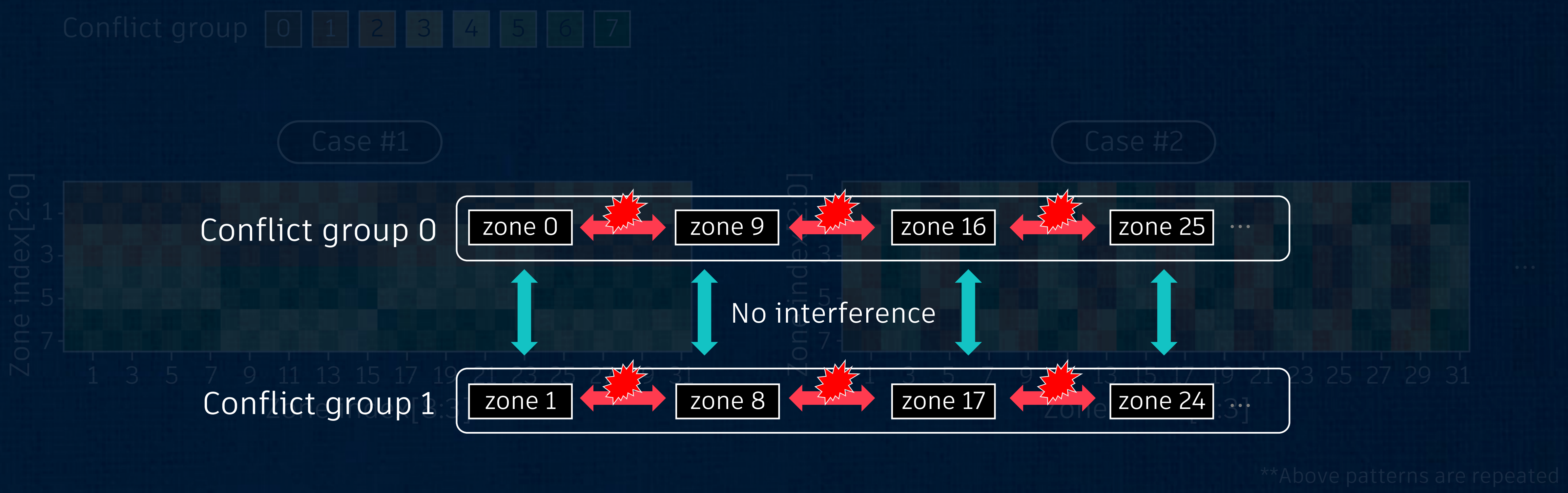
Case #2



**Above patterns are repeated

- Z2C mapping can vary based on the order in which zones are written

Interference Profiler - Results



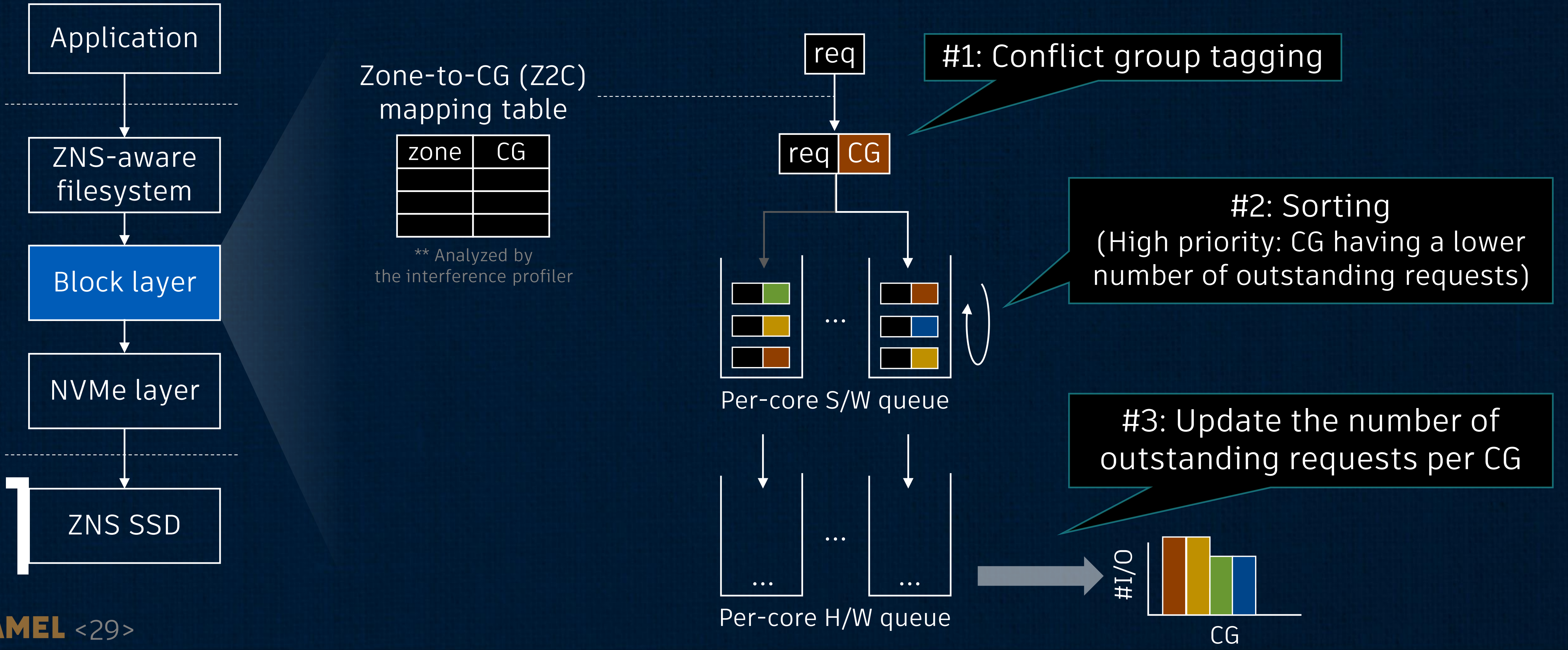
We can exploit the paralleism by accessing zones from different CGs

- Z2C mapping can vary based on the order in which zones are written

Interference-aware I/O Scheduler

Goal: Schedule I/O requests coming from different CGs as many as possible

➢ Able to exploit the internal parallelism



Zone-to-CG (Z2C) mapping table

zone	CG

** Analyzed by the interference profiler

#1: Conflict group tagging

#2: Sorting (High priority: CG having a lower number of outstanding requests)

#3: Update the number of outstanding requests per CG

1. Background – Zoned Namespaces

2. Challenge

3. Solution

4. Evaluation

Evaluation Setting

Schedulers

- blk_mq
  - Multi-queue I/O scheduler of Linux
- zns_mq
  - Multi-queue I/O scheduler that utilizes our interference profiling information

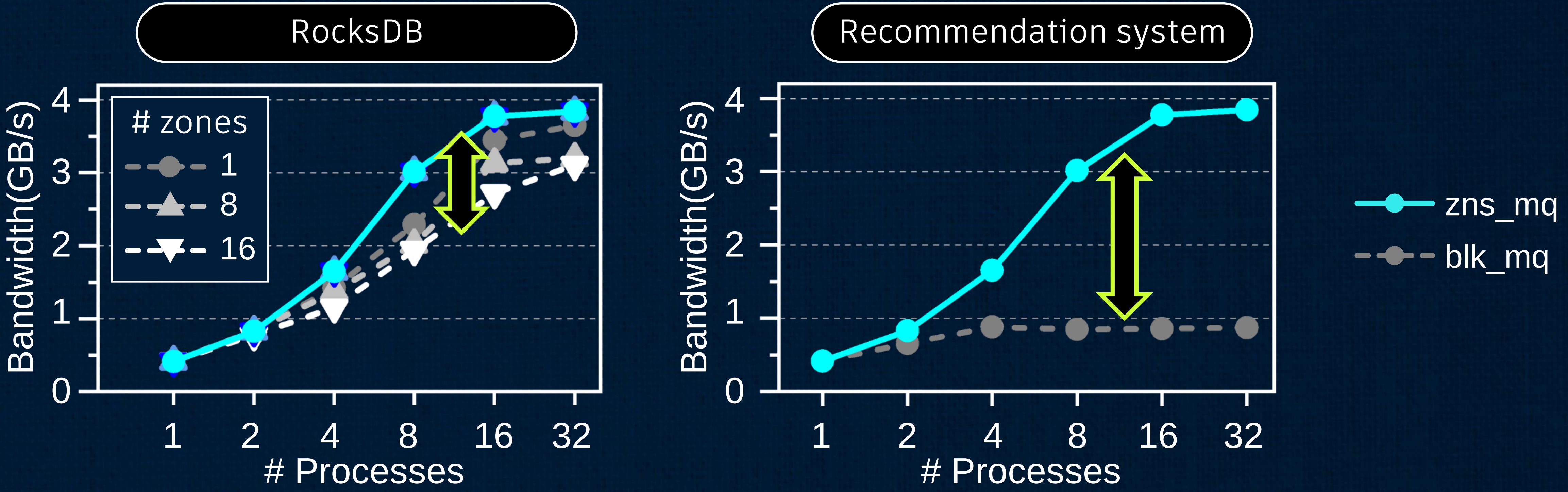
Workloads

- RocksDB
  - 1~16 zones / SST file
- Recommendation system
  - 128 zones / embedding table
  - Embedding table contains 50M indices with 64 dimensions

Environments

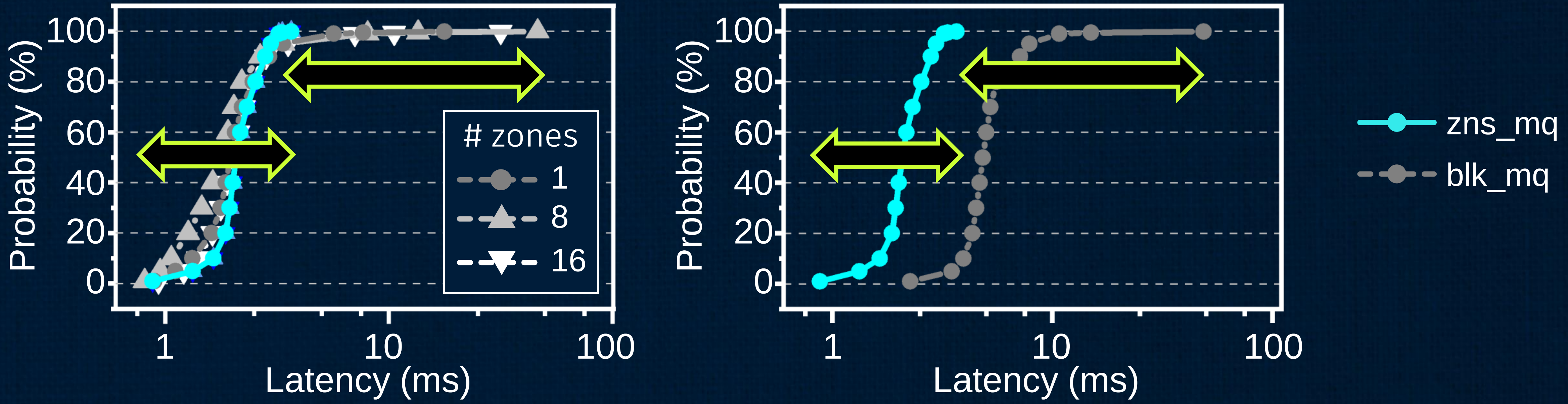
- ZNS-small
  - production ZNS SSD
  - 96MB/zone
  - TLC-based flash
- Intel Xeon CPU
  - 2.3GHz, 20cores, 40 vCPUs

Evaluation Results - Bandwidth



- znsmq improves the bandwidth of blk_mq by 1.98x, on average

Evaluation Results – Tail Latency


RocksDB

Recommendation system

- zns_mq shows narrower width of the distribution
  : All I/O requests experience similar interference levels
- zns_mq exhibits 11x shorter three nine (99.9%) tail latency, on average
- zns_mq exhibits 2.2x shorter latency, on average

Conclusion

- By using two production ZNS SSDs, we quantitively analyze the performance degradation due to the inter-zone interference
- We propose two simple modules to exploit the internal parallelism of ZNS SSDs
  - Interference profiler
  - Interference-aware I/O scheduler
- Our evaluation results show that our mechanism can improve the bandwidth and latency, significantly

Thank You

Contact: Hanyeoreum Bae (hyr.bae@camelab.org)