

# ZNS+ Simulator

Prof. Dongkun Shin ([dongkun@skku.edu](mailto:dongkun@skku.edu))

TA – Youngjae Lee([yjlee4154@gmail.com](mailto:yjlee4154@gmail.com))

TA – Jeeyoon Jung([wjdwldbsl@skku.edu](mailto:wjdwldbsl@skku.edu))

Embedded Software Laboratory

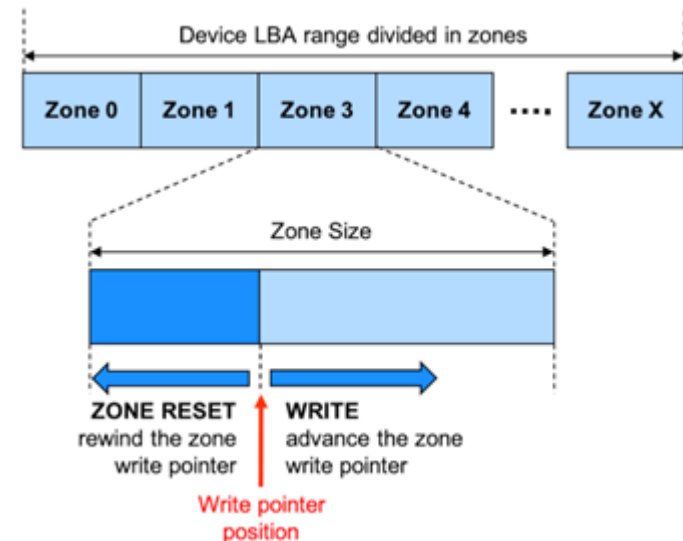
Sungkyunkwan University

<http://nyx.skku.ac.kr>

# ZNS SSD

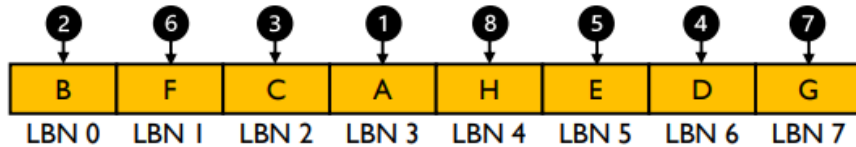
# Zoned Namespace

- SSD interface exposing NAND flash characteristics
- Zoned Interface
  - Logical address space is divided into fixed-size zones
  - Each zone must be written in sequential order
  - User explicitly reset zone to erase data
- Benefits
  - GC-less
    - Predictable latency
  - Less resource usage
    - DRAM
    - Over-provision



# Why ZNS? Small L2P Table

Random Write



LBN: Logical Block Number

LBN-level L2P translation table

LBN	Fblock/Fpage
0	0/1
1	1/1
2	0/2
3	0/0
4	1/3
5	1/0
6	0/3
7	1/2

Logical Block Size = 4KB

Fblock 0

Fpage 0	A
Fpage 1	B
Fpage 2	C
Fpage 3	D

Sequential Write

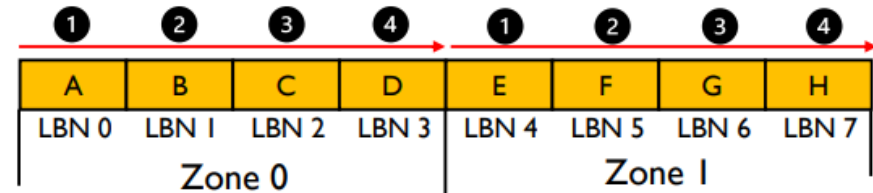
Fblock 1

Fpage 0	E
Fpage 1	F
Fpage 2	G
Fpage 3	H

Sequential Write

**Regular SSD**

Sequential Write



Zone 0

Zone 1

Zone-level L2P translation table

Zone	Fblock
0	0
1	1

Logical Zone Size = xxMB

Fblock 0

Fpage 0	A
Fpage 1	B
Fpage 2	C
Fpage 3	D

Sequential Write

Fblock 1

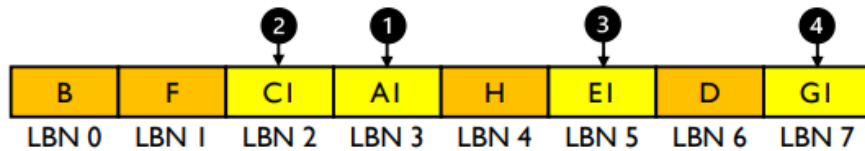
Fpage 0	E
Fpage 1	F
Fpage 2	G
Fpage 3	H

Sequential Write

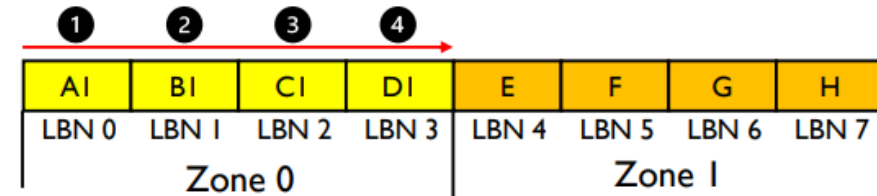
**ZNS SSD**

# Why ZNS? GC-less, Predictable

Random Write



Sequential Write



LBN-level L2P translation table

LBN	Fblock/Fpage
0	3/0
1	3/2
2	2/1
3	2/0
4	3/3
5	2/2
6	3/1
7	2/3

Fblock 0

Fpage 0	<del> </del>
Fpage 1	B
Fpage 2	<del> </del>
Fpage 3	D

Fblock 2

Fpage 0	AI
Fpage 1	CI
Fpage 2	EI
Fpage 3	GI

Fblock 1

Fpage 0	<del> </del>
Fpage 1	F
Fpage 2	<del> </del>
Fpage 3	H

Fblock 3 (OP)

Fpage 0	
Fpage 1	
Fpage 2	
Fpage 3	

Regular SSD

GC: valid page copy

→ write amplified, unexpected delay

Zone-level L2P translation table

Zone	Fblock
0	2
1	1

GC-less, No OP  
WAF  $\approx$  1  
(write amp. factor)

Fblock 0

Fpage 0	<del> </del>
Fpage 1	<del> </del>
Fpage 2	<del> </del>
Fpage 3	<del> </del>

Fblock 2

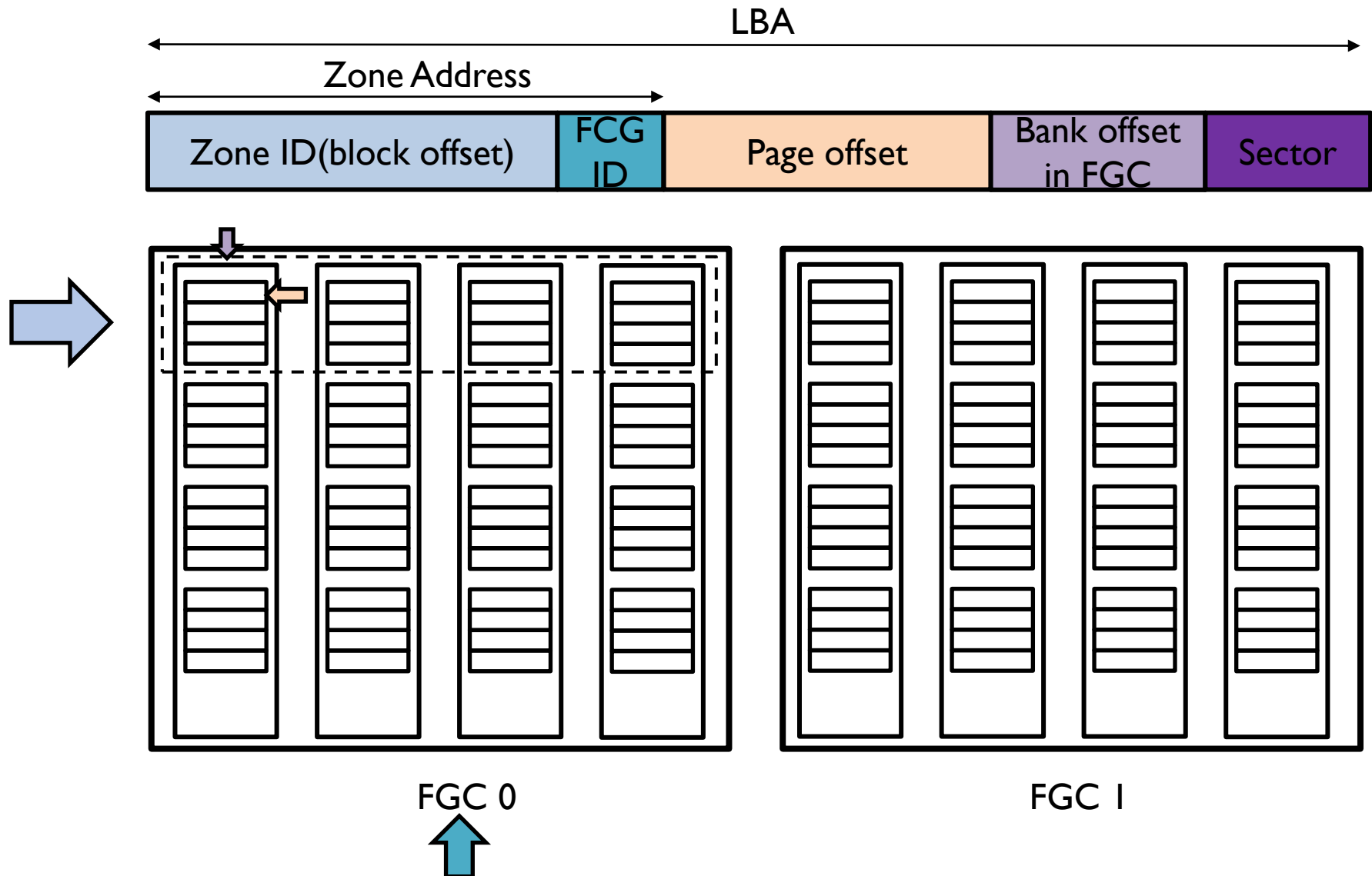
Fpage 0	AI
Fpage 1	BI
Fpage 2	CI
Fpage 3	DI

Fblock 1

Fpage 0	E
Fpage 1	F
Fpage 2	G
Fpage 3	H

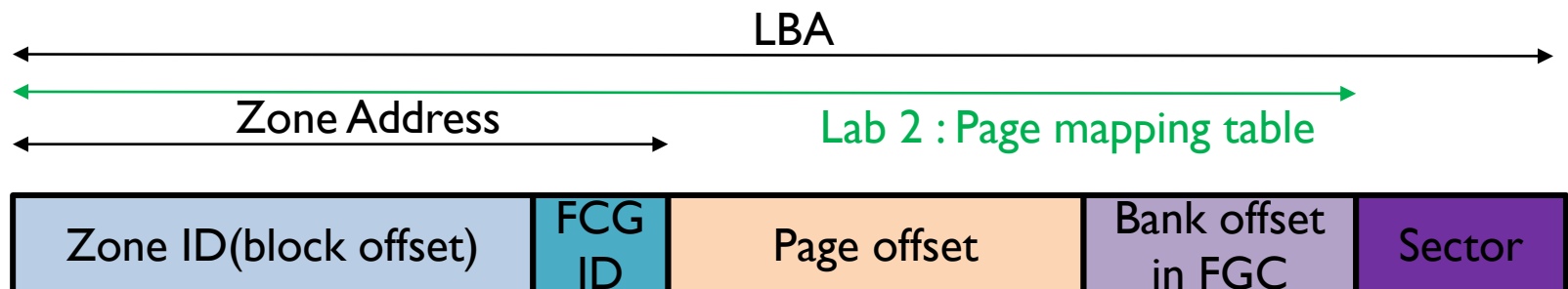
ZNS SSD

# ZNS striping policy



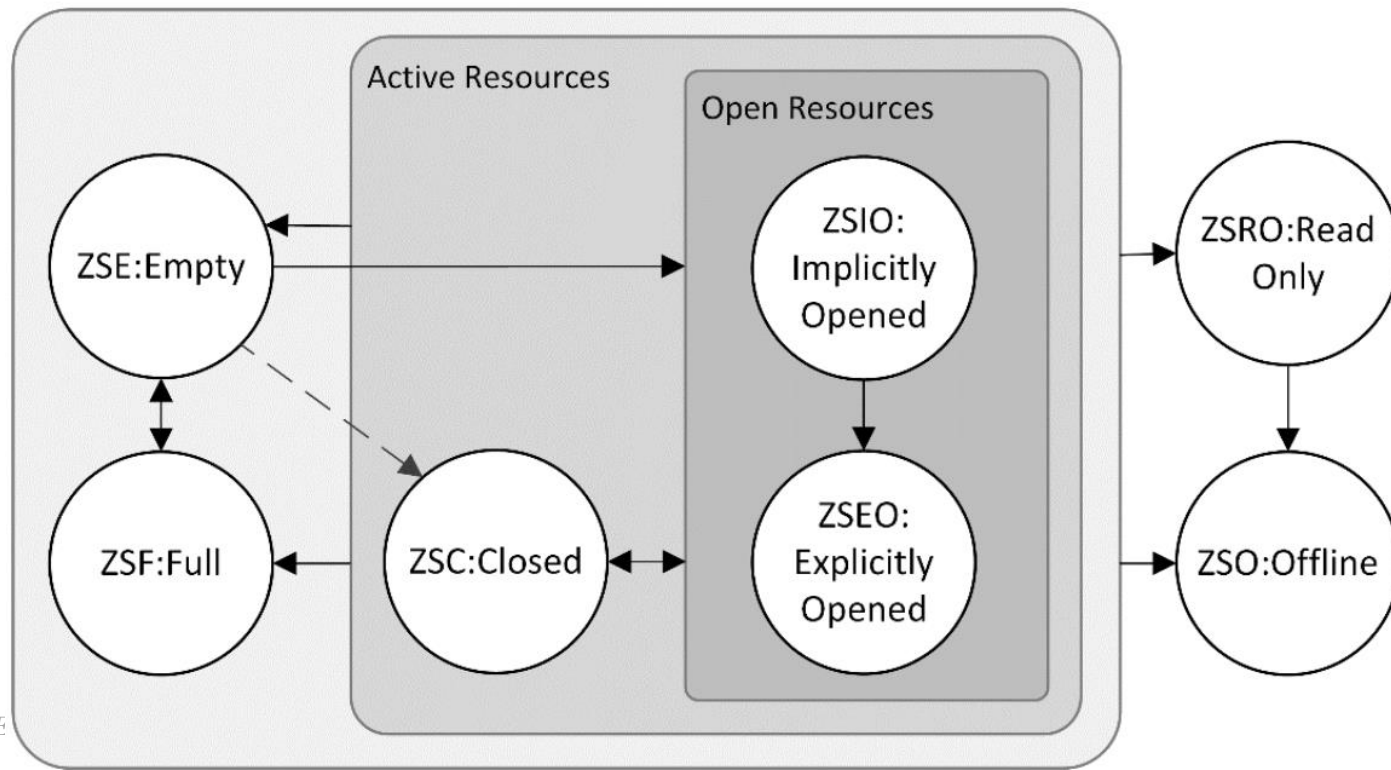
# ZNS striping policy

- Must use Zone Address  $\rightarrow$  Physical Address mapping table in SSD side DRAM
  - Lab2 : page mapping table use  $\times 2^{(\text{Page offset} + \text{Bank offset in FGC})}$  DRAM than ZNS simulator
- Total bank :  $2^{(\text{FCG ID} + \text{Bank offset in FGC})}$



# Zone State Machine

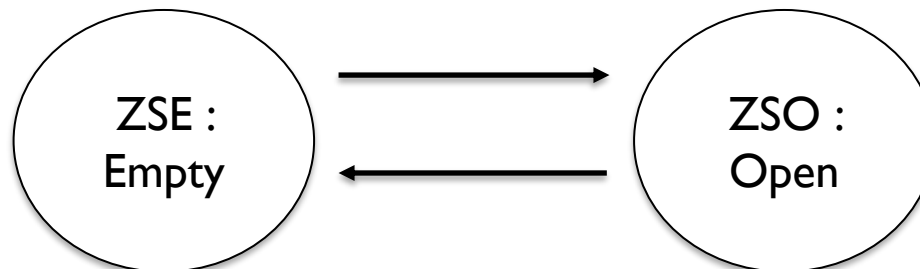
- In NVMe spec
  - Manage total 7 state
  - Devide Open Resources & Active Resources
    - Max open Zone & Max Active Zone





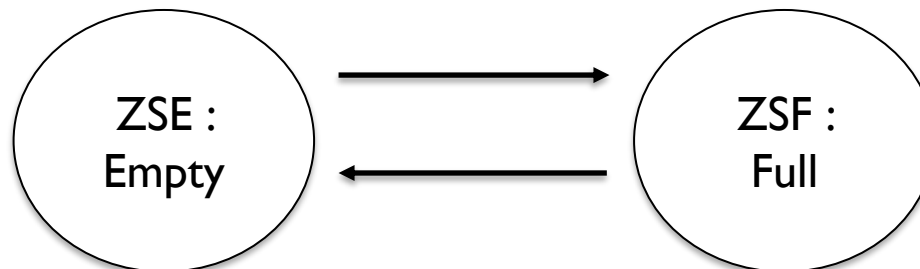
# Zone State Machine

- Empty → Open
  - Host writes first LBA of zone, Zone opened automatically
  - Host sends **Zone Management Send** Command (Empty → Open)
  - If open resource is full, then report error
- Open → Empty
  - Host sends **Zone Management Send** Command (Open → Empty)



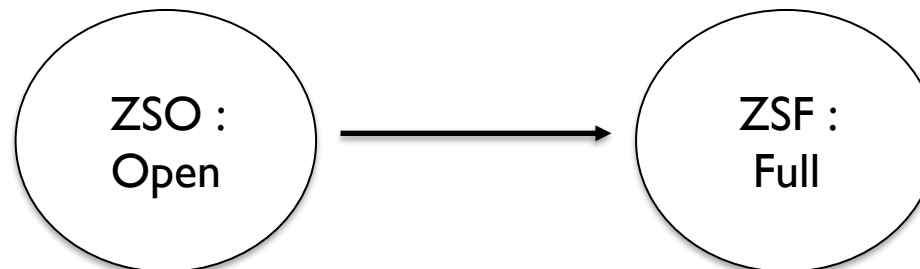
# Zone State Machine

- Empty  $\rightarrow$  Full
  - Host sends **Zone Management Send Command** (Empty  $\rightarrow$  Full)
- Full  $\rightarrow$  Empty
  - Host sends **Zone Management Send Command** (Full  $\rightarrow$  Empty)



# Zone State Machine

- Open → Full
  - Host sends **Zone Management Send** Command (Open → Full)
  - Host writes last LBA of zone, Zone Finished automatically
- Full → Open
  - Full state zone must be reset and open
  - It can't be happened, if this occur return error



# Zone Management Command

- Zone\_Descriptor : must manage in DRAM to manage Zone State
  - Manage Zone's Start LBA, Size, Write Pointer, State
- Zone management Send & Receive
  - Zone Management Send
    - Change state of the zone
  - Zone Management Receive
    - Return Zone Descriptor

# DRAM usage

- Zone\_Descriptor
- Logical Zone to Physical Zone mapping table
- Data buffer(must keep not page aligned data)
  - Keep # max open zone Data buffer(size = 1 page)

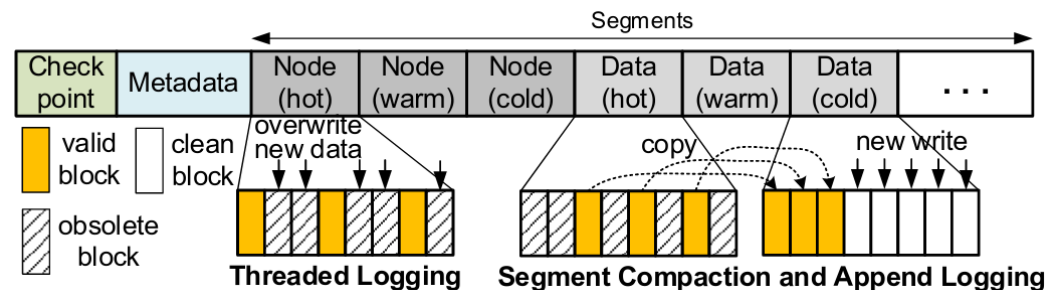
# ZNS+

# ZNS+

- **ZNS+:Advanced Zoned Namespace Interface for Supporting In-Storage Zone Compaction**
  - **OSDI'21**
    - <https://www.usenix.org/conference/osdi21/presentation/han>
  - **ZNS extension for F2FS file system**
    - **Offload file system management to ZNS SSD**

# F2FS (Flash-Friendly File System)

- F2FS
  - One of actively maintained Log-structured File Systems (LFS)
  - Six types of segments: hot, warm, and cold segments for each node/data
  - Multi-head logging
  - Internal GC to reclaim invalid blocks
    - Append logging (AL)
    - Threaded logging (TL)
  - Support ZNS SSD





# F2FS (Flash-Friendly File System)

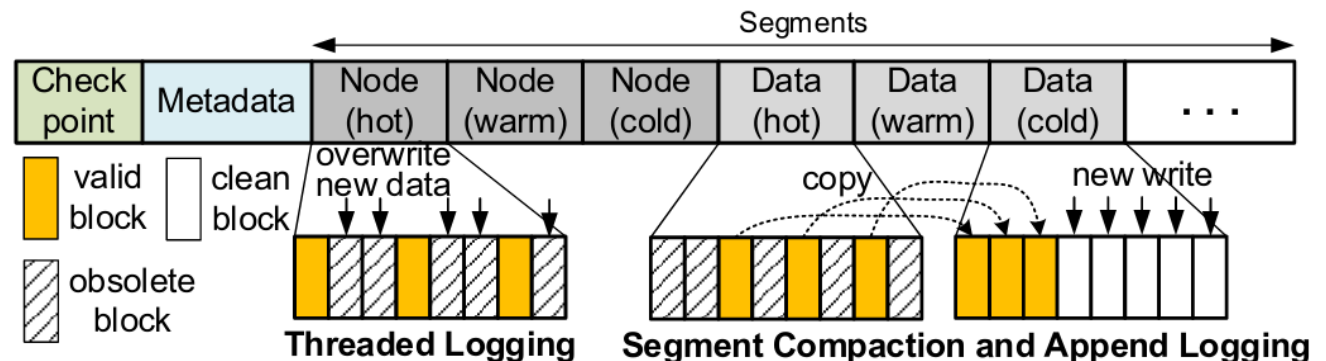
- Garbage Collection

- Segment Compaction

- Select a victim segment with lowest compaction cost
- Copy valid data to new free segment via host-initiated R/W
- Update metadata

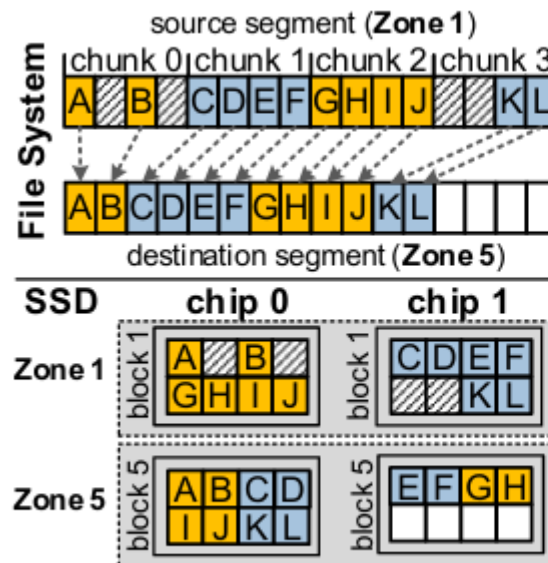
- Threaded Logging

- Enabled when free segment become insufficient
- Overwrite invalid blocks with new data
- Disabled for ZNS SSD

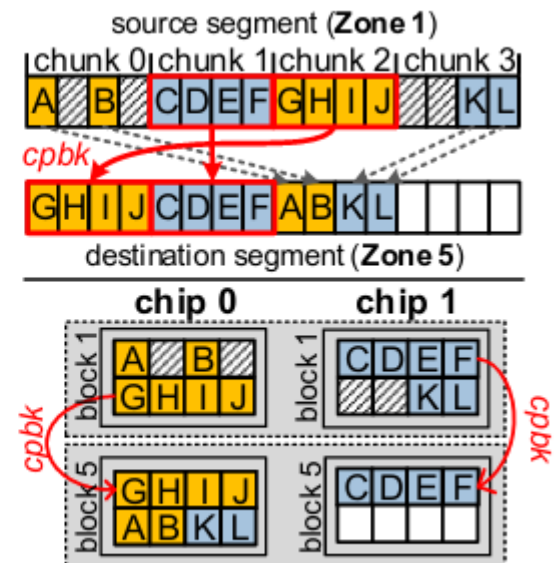


# ZNS+: LFS-Aware ZNS

- **Internal Zone Compaction**
  - Accelerate segment compaction
  - Copy blocks within SSD
    - Reduce host-device traffic



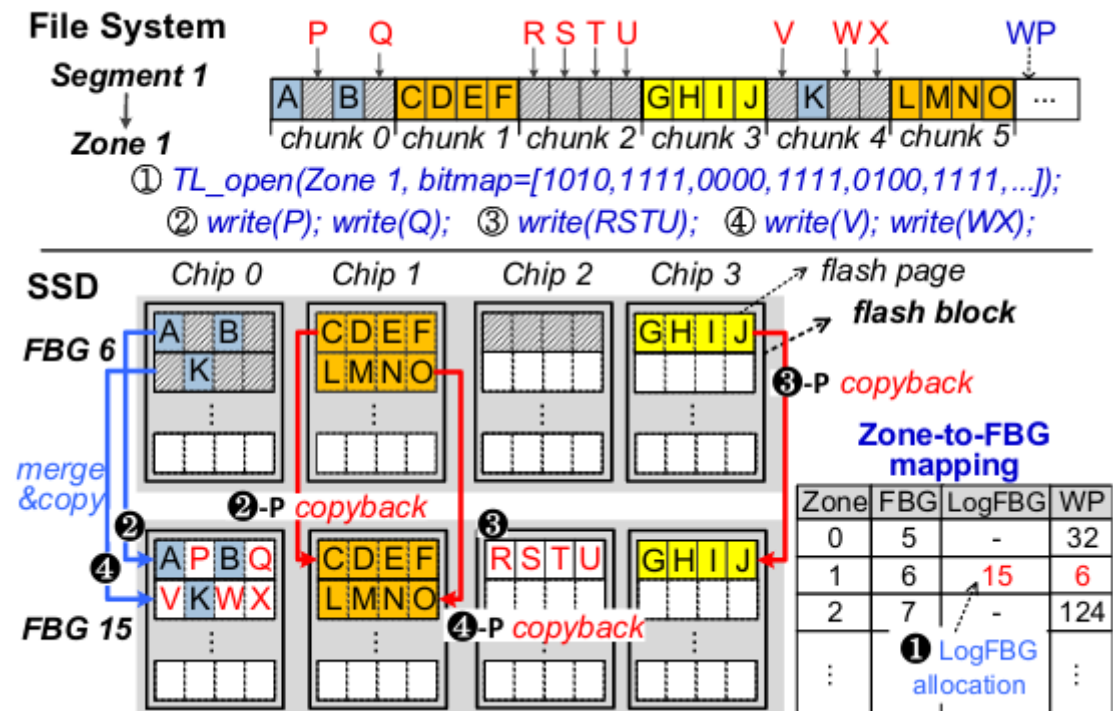
(a) normal block allocation



(b) copyback-aware allocation

# ZNS+: LFS-Aware ZNS

- **Sparse Sequential Write**
  - Host can overwrite a zone sparsely
    - Support F2FS threaded logging on ZNS SSD



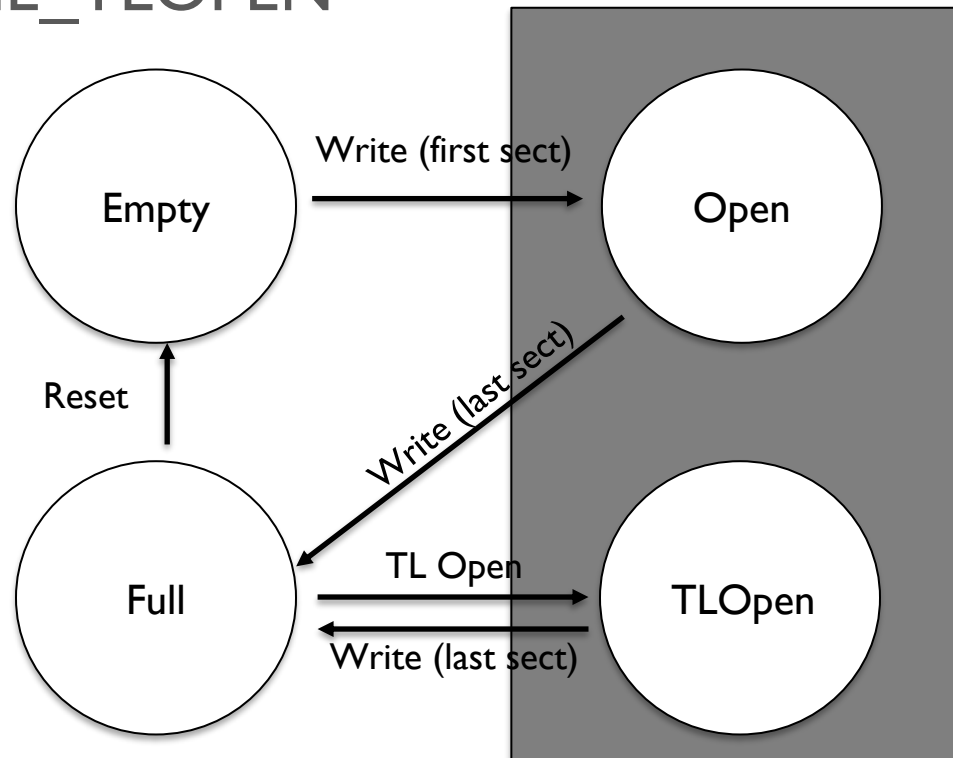
# Lab 5 : ZNS+ Simulator

# ZNS+ Simulator

- Develop a ZNS+ simulator
  - Simulate the operations ZNS SSD
    - Zoned I/O interface: Read, Write, Reset
    - Zone metadata
  - We also implement ZNS+ operations
    - Zone Compaction
    - Threaded Logging

# ZNS+ Simulator

- Zone State for our Assignment
  - Total 4 states
  - Manage Open resource for ZONE\_OPE and ZONE\_TLOPEN



# ZNS Operations

- `zns_init(nbank, nblk, npage, dzone)`
  - Initialize ZNS simulator
    - Zone descriptors (state, write pointer, start lba)
    - Buffer for unaligned write
    - Internal metadata (zone-to-fbg, threaded logging, free fbg list)
  - `nbank, blk, npage`: NAND flash dimension
    - `nbank, npage`: power-of-two
  - Dzone: zone interleaving degree
    - Degree is guaranteed to be a power-of-two

# ZNS Operations

- `zns_write(lba, nsect, data)`
  - If ``lba`` does not match to the write pointer or zone is in ``ZONE_FULL`` state
    - Write fails and return -1
  - If ``lba`` is start of the zone
    - Change the zone state to ``ZONE_OPEN`` and increase open count
    - If total open zone count has already exceeded ``MAX_OPEN_ZONE``, then write fails and return -1
    - Allocate FBG
      - Free FBG should be managed in FIFO manner

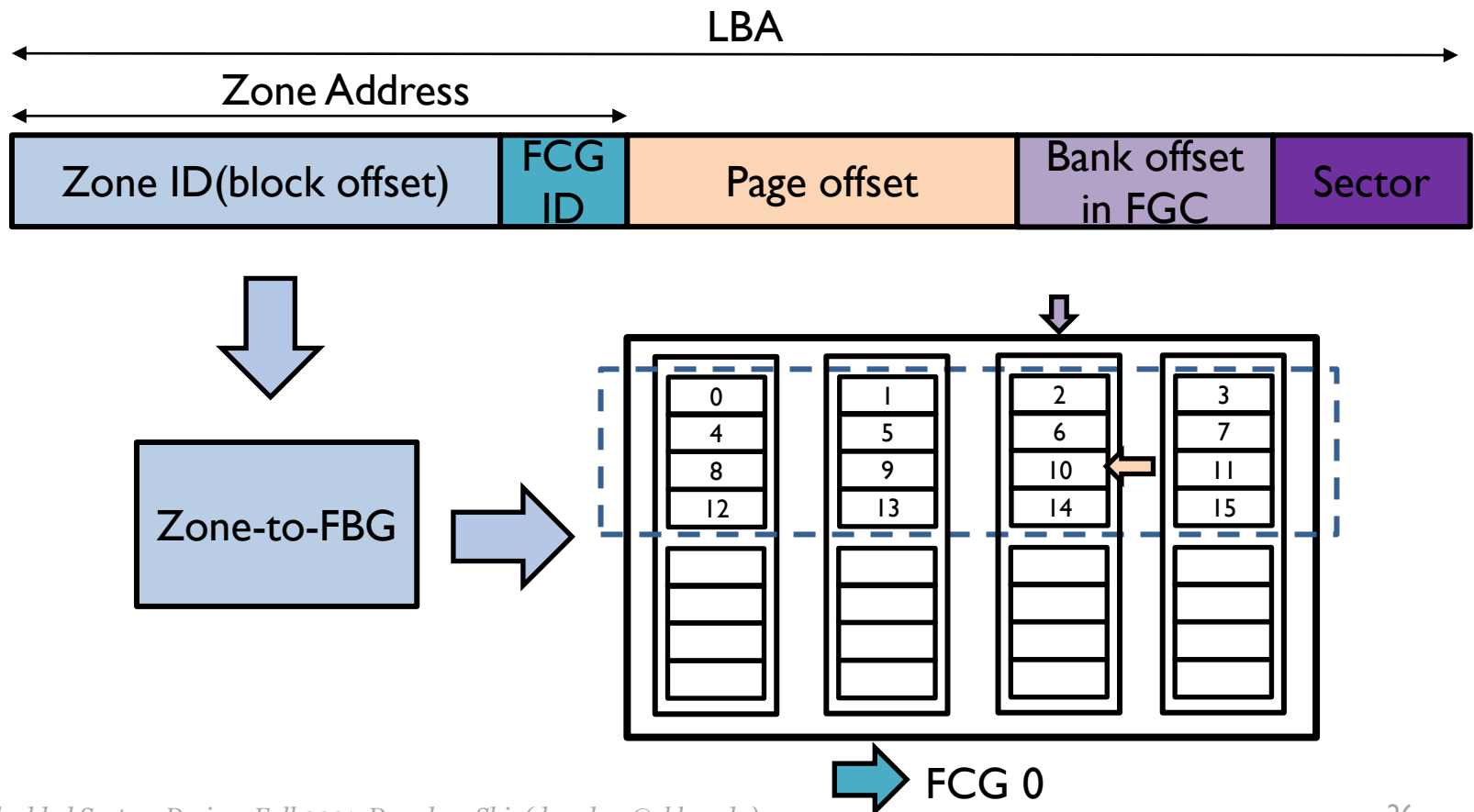


# ZNS Operations

- `zns_write(lba, nsect, data)`
  - Write data to the NAND
    - Stripes data across the banks mapped to the zone
    - If the data does not fit into the NAND page, store in buffer
    - Increase ``wp`` by ``nsect``
    - If the last sector of zone is written, change state to `ZONE_FULL`
  - `[lba, lba+nsect)` range is guaranteed to be included within single zone

# ZNS Operations

- `zns_write(lba, nsect, data)` (cont.)
  - Write striping



# ZNS Operations

- `zns_read(lba, nsect, data)`
  - Fill ``data`` with written data and return
  - Read has no restriction (max open, writer pointer)
  - If target sector is not written yet, fill all bytes with ``0xff``
  - You should also check the data stored in the buffer
  - `[lba, lba+nsect)` range is guaranteed to be included within single zone

# ZNS Operations

- `zns_reset(lba)`
  - Reset the zone starting with given ``lba``
    - Erase all NAND blocks in target zone
  - Return -1 if target zone is not in `ZONE_FULL` state
  - Zone metadata
    - Reset ``wp`` to the ``slba``
    - Change ``state`` to ``ZONE_EMPTY``

# ZNS Operations

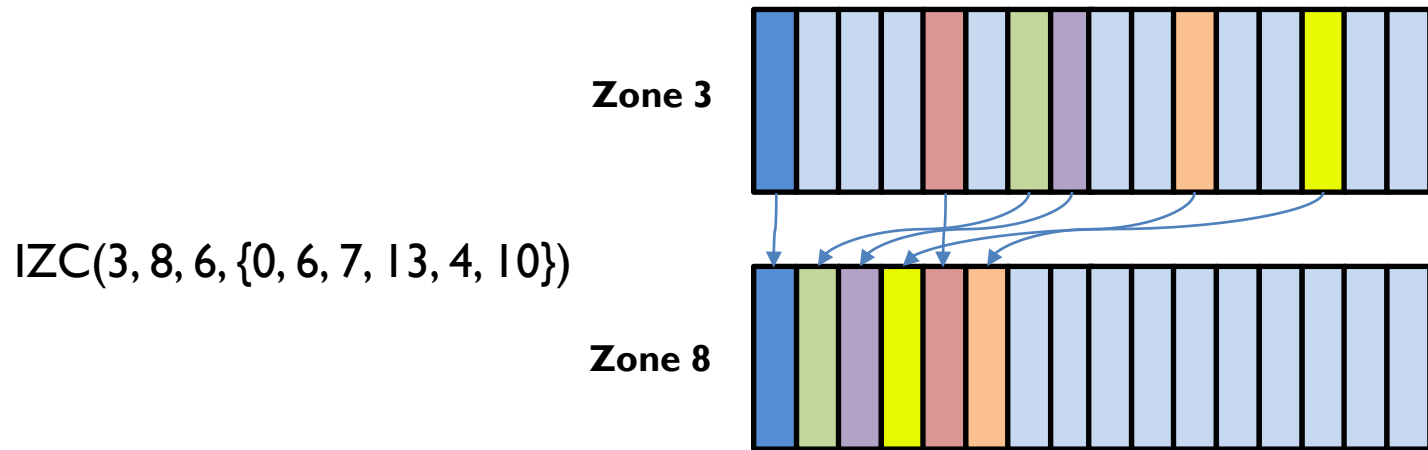
- `zns_get_desc(szone, nzone, desc)`
  - Fill ``descs`` with the zone descriptors
    - ``state``: zone state
    - ``slba``: start address of the zone
    - ``wp``: write pointer of the zone
  - ``nzone`` descriptors from ``szone`` to ``szone+nzone-1``
  - The length of the ``descs`` array is ``nzone``

# ZNS+ Operations

- `zns_izc(src_zone, dest_zone, copy_len, copy_list)`
  - Copy sectors from source zone to destination zone
  - Append sectors in ``copy_list`` from the beginning of the destination zone
    - Each entry is sector offset from the beginning of the zone
  - Reset source zone
  - If total open zone count has already exceeded ``MAX_OPEN_ZONE``, then write fails and return -1
  - Return -1 if target zones are not in suitable state
    - `src_zone != dest_zone`
    - `src_zone: ZONE_FULL, target_zone: ZONE_EMPTY`

# ZNS+ Operations

- `zns_izc(src_zone, dest_zone, copy_len, copy_list)`
  - Example



# ZNS+ Operations

- `zns_tl_open(zone, valid_arr)`
  - Open a zone for threaded logging
    - Allow sparse overwrite `zns_write` on TL opened zone
    - Increase open zone count same as normal open
      - Allocate one more FBG for target zone
    - Sectors specified in bitmap will be skipped by host write
      - Copy skipped sectors from original NAND blocks
    - Target zone should be in `'ZONE_FULL'` state
      - If not, return -1
  - Each bank has `'MAX_OPEN_ZONE'` OP blocks for TL
  - After the zone becomes FULL, erase source NAND blocks in FBG



# ZNS+ Operations

- `zns_tl_open(zone, valid_arr)` (cont.)

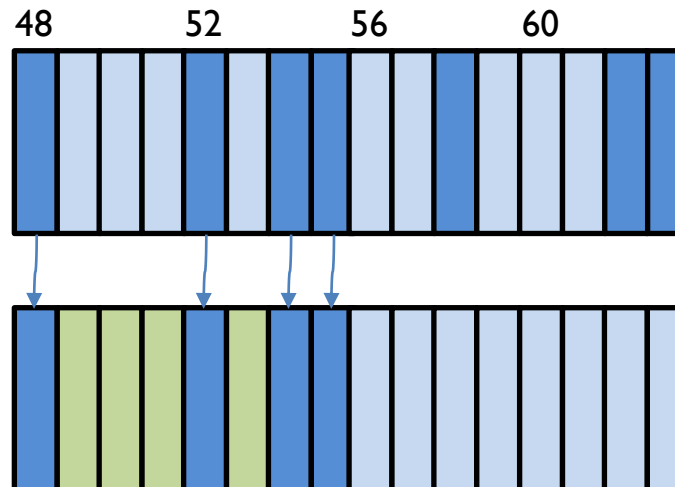
`TL_OPEN(3, {1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1})`

`WRITE(49, 3)`

`WRITE(53, 1)`

`READ(48, 4)` from new zone

`READ(56, 4)` from old zone



Zone becomes FULL after write sector 61

# Grading Policy

- Recommended environment : GCC on Linux
  - You can do it in Windows, but be sure that your work also runs in Linux
  - Use only standard C library or POSIX C library functions
- Personal Project
- Submissions will be graded based on the number of test cases passed
  - We will use larger test cases for scoring
- Submit to the icampus
  - Due: 10/31 (Sun.) 23:59:00
  - Submission file name: ``<student_id>.tar.gz`` (includes ``zns.c`` only)
  - Modify student id in ``Makefile`` and use ``make submit`` command
- Late penalty : -20 % / day (Up to 3 days)

# Q&A