Ziggurat: A Tiered File System for Non-Volatile Main Memories and Disks

Shengan Zheng[†], Morteza Hoseinzadeh[§], Steven Swanson[§]

[†] *Shanghai Jiao Tong University*
[§] *University of California, San Diego*

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

NVSL

Background

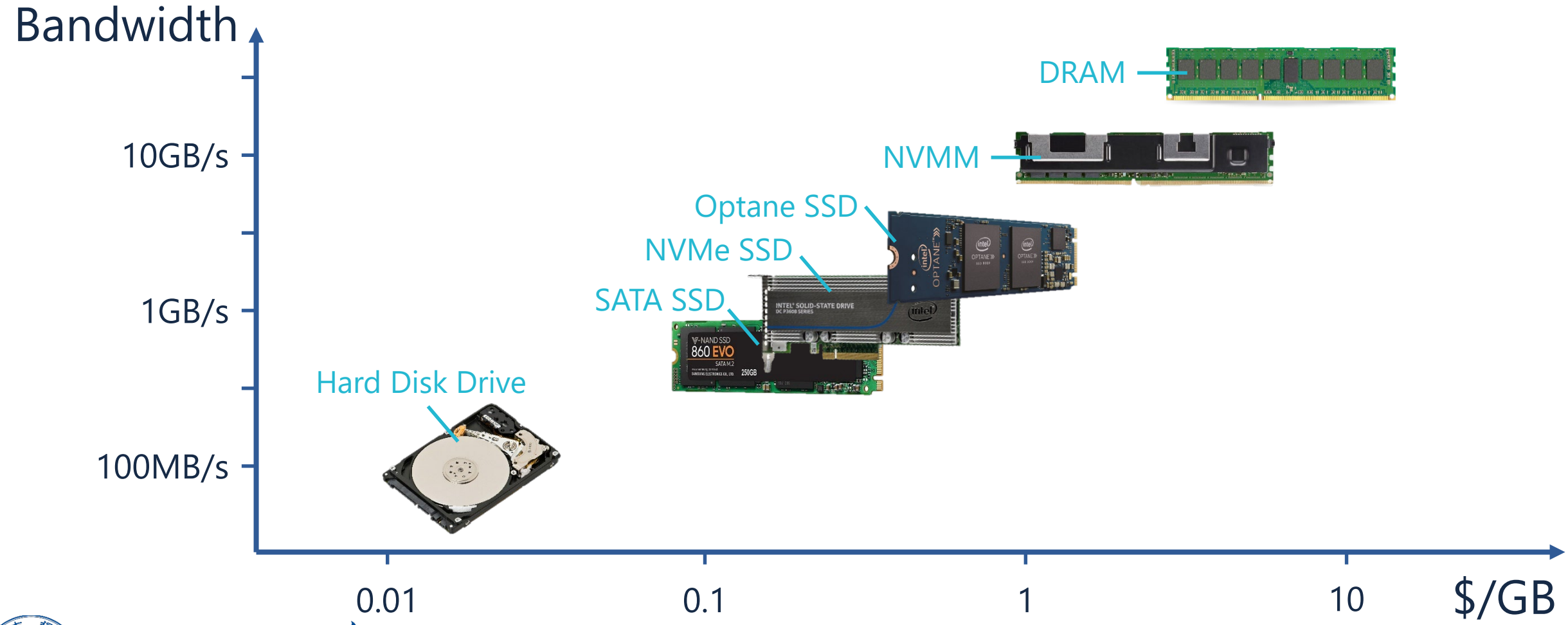
- Non-volatile main memory (NVMM)
  - Byte-addressability
  - Persistence
  - Direct access (DAX)

3D-XPoint NVDIMM

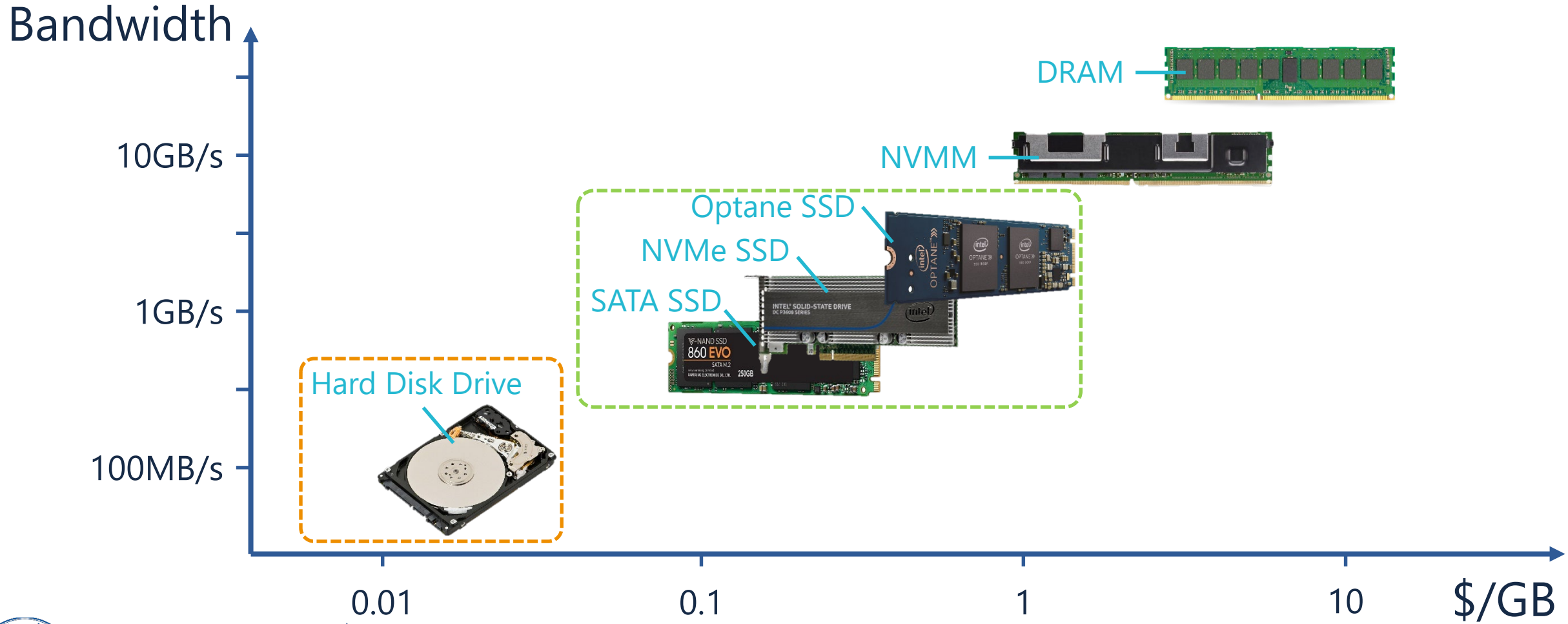
- NVMM file systems
  - PMFS, SCMFS, NOVA
  - EXT4-DAX, XFS-DAX
  - Capacity?

DRAM + Flash NVDIMM

Motivation


Bandwidth
10GB/s
1GB/s
100MB/s
DRAM
NVMM
Optane SSD
NVMe SSD
SATA SSD
Hard Disk Drive
0.01
0.1
1
10
$/GB

Motivation

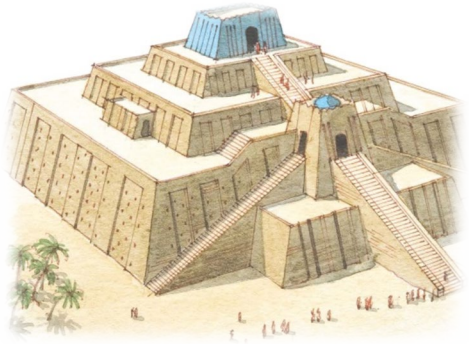

Bandwidth
10GB/s
1GB/s
100MB/s
DRAM
NVMM
Optane SSD
NVMe SSD
SATA SSD
Hard Disk Drive
0.01
0.1
1
10
$/GB

Tiered Storage System

- SSD for speed
- HDD for capacity

Tiered Storage System

- NVMM for speed
- Disks for capacity



Ziggurat Overview

- Intelligent data placement policy
  - Send writes to the most suitable tier
  - High NVMM space utilization
- Accurate predictors
  - Predict the synchronicity of each file (synchronicity predictor)
  - Predict the size of future writes to each file (write size predictor)
- Efficient migration mechanism
  - Only migrate cold data in cold files
  - Migrate file data in groups

Outline

- Motivation
- Data placement policy
- Migration mechanism
- Evaluation
- Conclusion

Data Placement Policy

- *Although NVMM is the fastest tier in Ziggurat, file writes should not always go to NVMM.*

Data Placement		Synchronicity predictor	
		Synchronously-updated	Asynchronously-updated
Write size predictor	Large writes	NVMM 	Disk 
	Small writes	NVMM 	NVMM 

Synchronicity Predictor

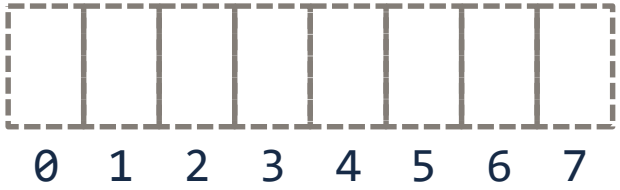
- *Predict whether the future accesses are likely to be **synchronous***

```
write(0,2);
fsync();
write(2,2);
fsync();
write(4,2);
fsync();
```

File log

Data blocks written: 0 / 4

File data



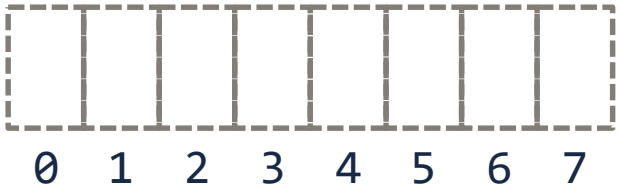
Synchronous

```
write(0,2);
write(2,2);
write(4,2);
fsync();
```

File log

Data blocks written: 0 / 4

File data



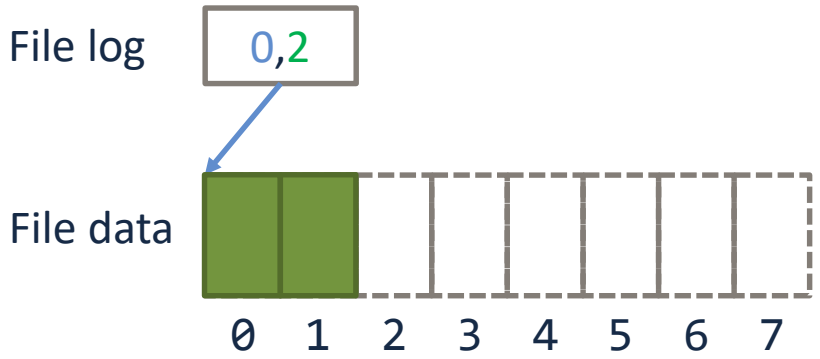
Synchronous

Write entry offset, length

Synchronicity Predictor

- *Predict whether the future accesses are likely to be **synchronous***

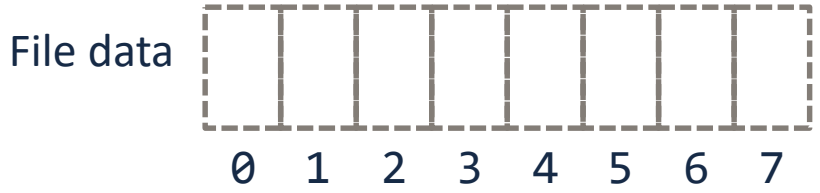
```
write(0,2);
fsync();
write(2,2);
fsync();
write(4,2);
fsync();
```



Data blocks written: 2 / 4

Synchronous

```
write(0,2);
write(2,2);
write(4,2);
fsync();
```



Data blocks written: 0 / 4

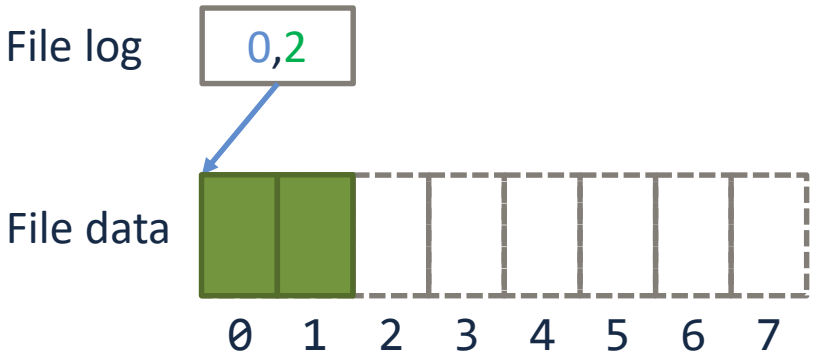
Synchronous

Write entry offset, length

Synchronicity Predictor

- *Predict whether the future accesses are likely to be **synchronous***

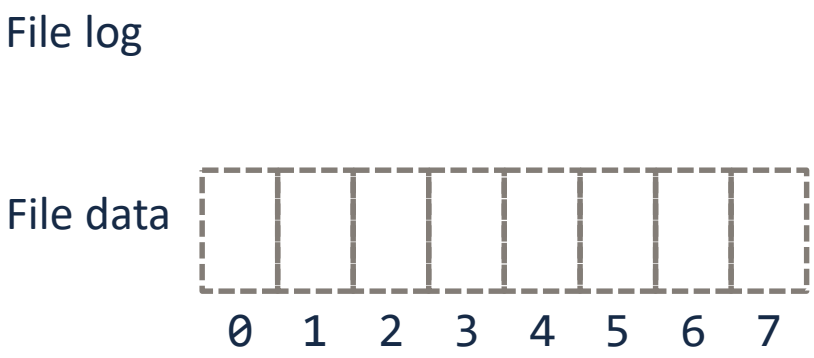
```
write(0,2);
fsync();
write(2,2);
fsync();
write(4,2);
fsync();
```



Data blocks written: 0 / 4

Synchronous

```
write(0,2);
write(2,2);
write(4,2);
fsync();
```



Data blocks written: 0 / 4

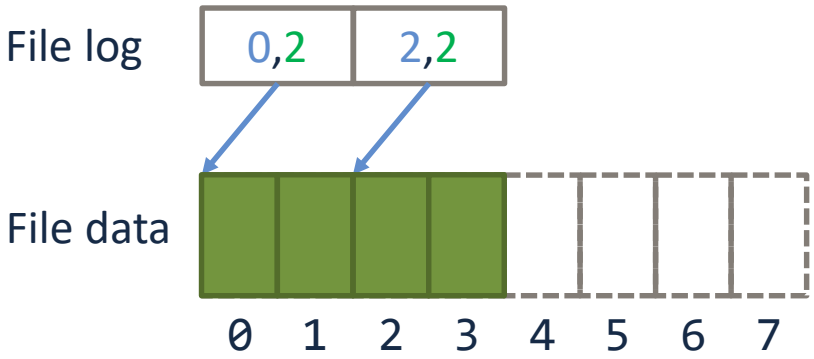
Synchronous

Write entry offset, length

Synchronicity Predictor

- *Predict whether the future accesses are likely to be **synchronous***

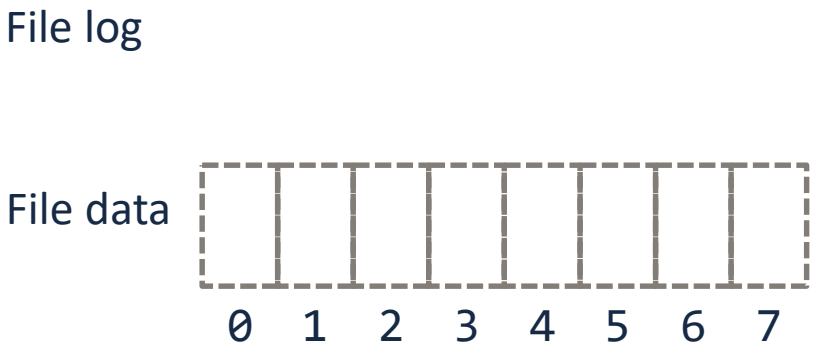
```
write(0,2);
fsync();
write(2,2);
fsync();
write(4,2);
fsync();
```



Data blocks written: 2 / 4

Synchronous

```
write(0,2);
write(2,2);
write(4,2);
fsync();
```



Data blocks written: 0 / 4

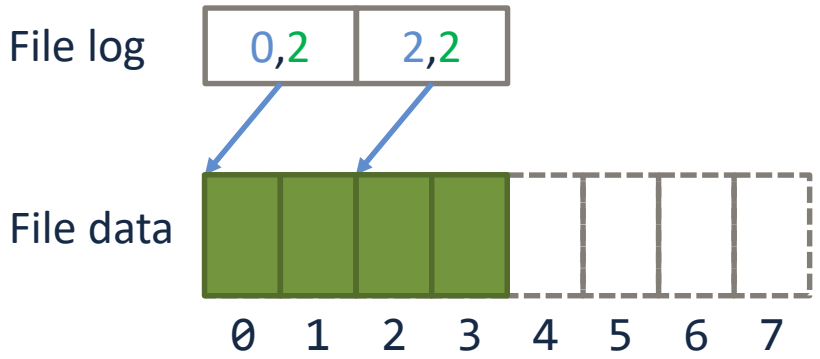
Synchronous

Write entry offset, length

Synchronicity Predictor

- *Predict whether the future accesses are likely to be **synchronous***

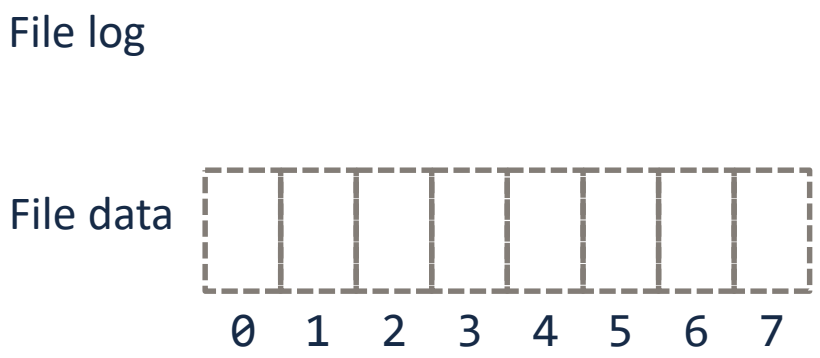
```
write(0,2);
fsync();
write(2,2);
fsync();
write(4,2);
fsync();
```



Data blocks written: 0 / 4

Synchronous

```
write(0,2);
write(2,2);
write(4,2);
fsync();
```



Data blocks written: 0 / 4

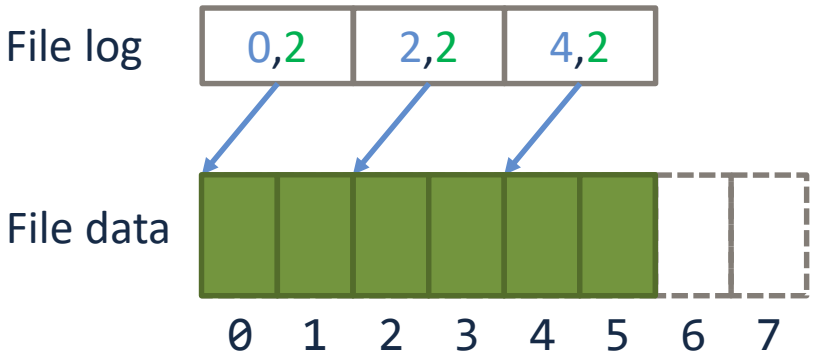
Synchronous

Write entry offset, length

Synchronicity Predictor

- *Predict whether the future accesses are likely to be **synchronous***

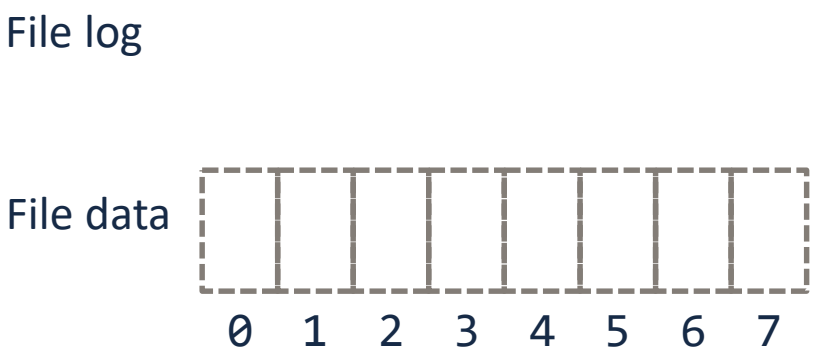
```
write(0,2);
fsync();
write(2,2);
fsync();
write(4,2);
fsync();
```



Data blocks written: 2 / 4

Synchronous

```
write(0,2);
write(2,2);
write(4,2);
fsync();
```



Data blocks written: 0 / 4

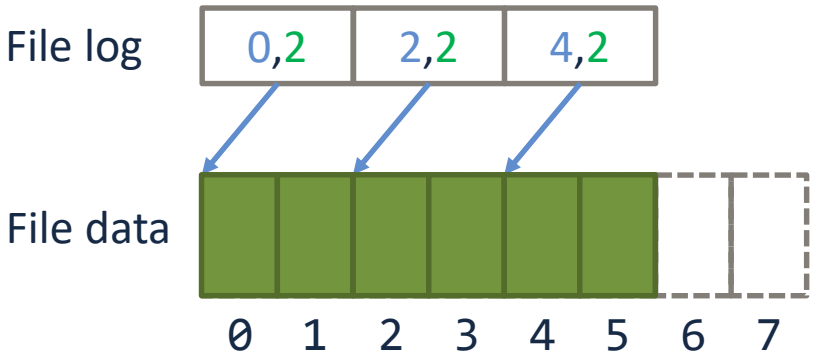
Synchronous

Write entry offset, length

Synchronicity Predictor

- *Predict whether the future accesses are likely to be **synchronous***

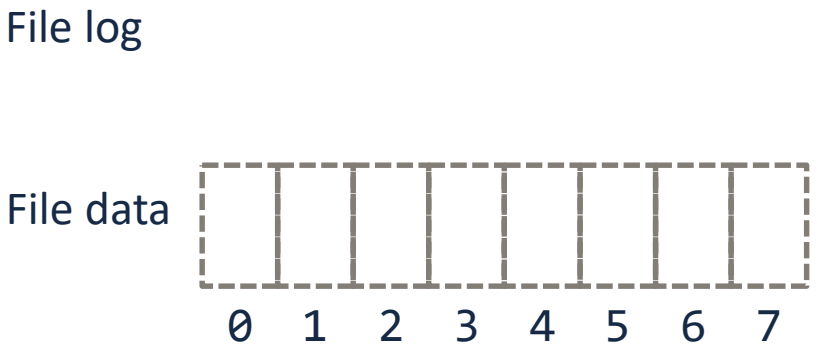
```
write(0,2);
fsync();
write(2,2);
fsync();
write(4,2);
➡fsync();
```



Data blocks written: 0 / 4

Synchronous

```
write(0,2);
write(2,2);
write(4,2);
fsync();
```



Data blocks written: 0 / 4

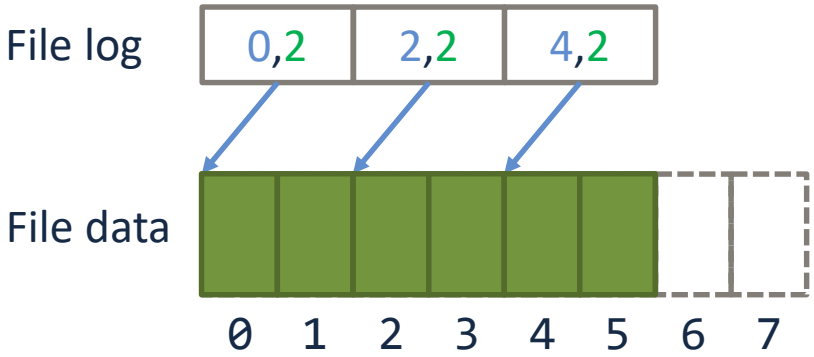
Synchronous

Write entry offset, length

Synchronicity Predictor

- *Predict whether the future accesses are likely to be **synchronous***

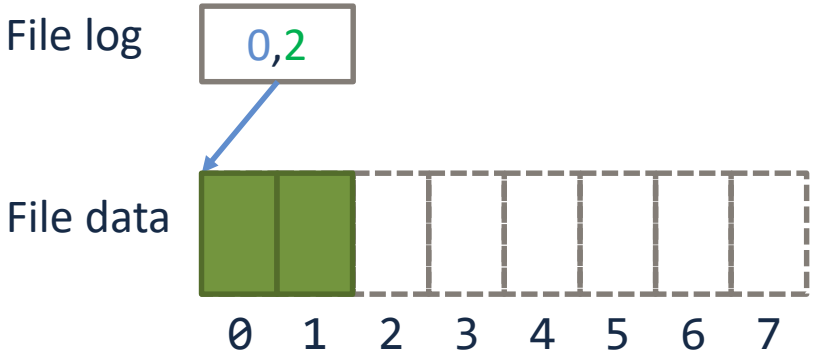
```
write(0,2);
fsync();
write(2,2);
fsync();
write(4,2);
fsync();
```



Data blocks written: 0 / 4

Synchronous

```
➡write(0,2);
write(2,2);
write(4,2);
fsync();
```



Data blocks written: 2 / 4

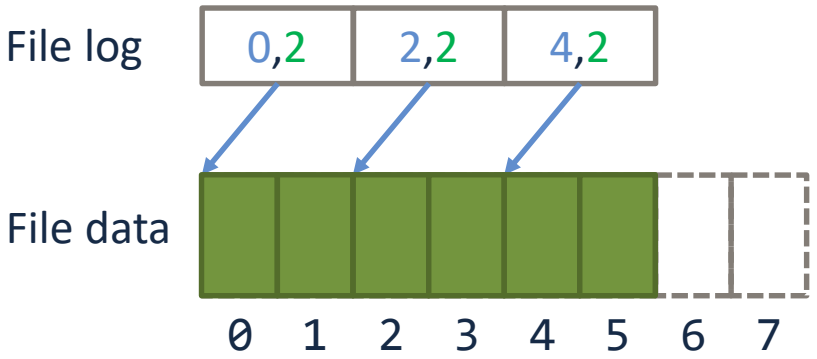
Synchronous

Write entry offset, length

Synchronicity Predictor

- *Predict whether the future accesses are likely to be **synchronous***

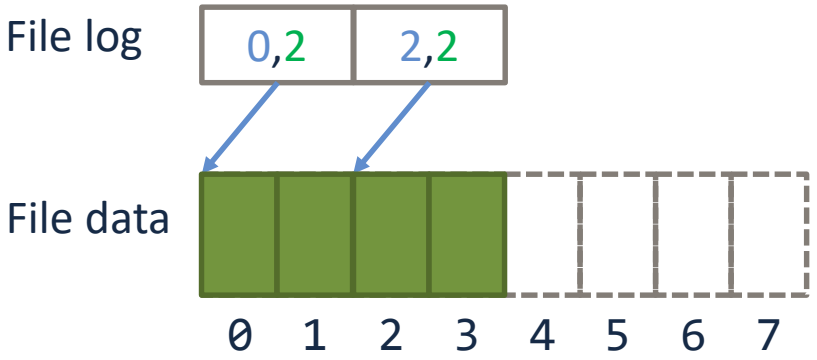
```
write(0,2);
fsync();
write(2,2);
fsync();
write(4,2);
fsync();
```



Data blocks written: 0 / 4

Synchronous

```
write(0,2);
➡write(2,2);
write(4,2);
fsync();
```



Data blocks written: 4 / 4

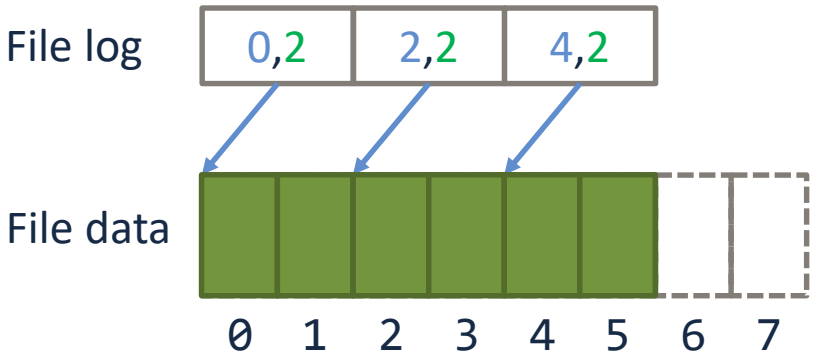
Synchronous

Write entry | offset, length

Synchronicity Predictor

- *Predict whether the future accesses are likely to be **synchronous***

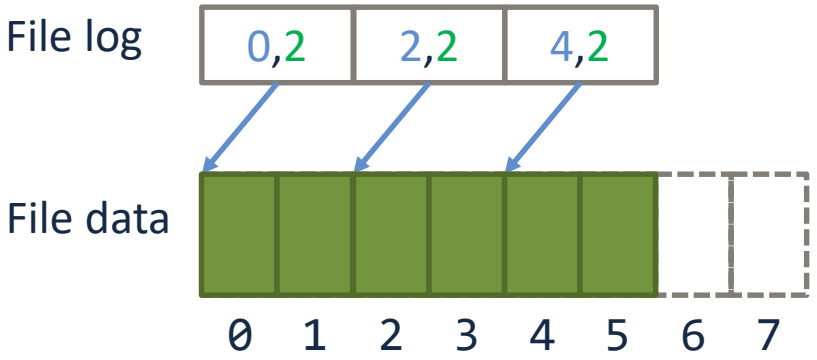
```
write(0,2);
fsync();
write(2,2);
fsync();
write(4,2);
fsync();
```



Data blocks written: 0 / 4

Synchronous

```
write(0,2);
write(2,2);
➡write(4,2);
fsync();
```



Data blocks written: 6 / 4

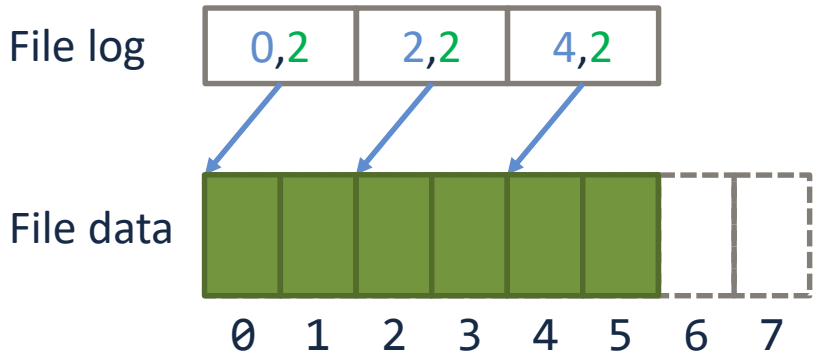
Synchronous

Write entry offset, length

Synchronicity Predictor

- *Predict whether the future accesses are likely to be **synchronous***

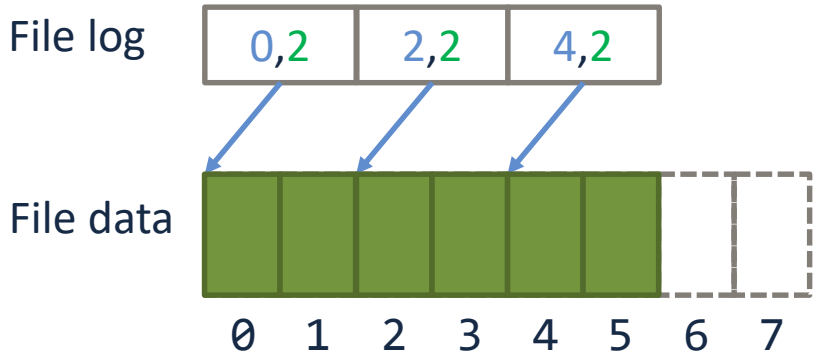
```
write(0,2);
fsync();
write(2,2);
fsync();
write(4,2);
fsync();
```



Data blocks written: 0 / 4

Synchronous

```
write(0,2);
write(2,2);
write(4,2);
➡fsync();
```



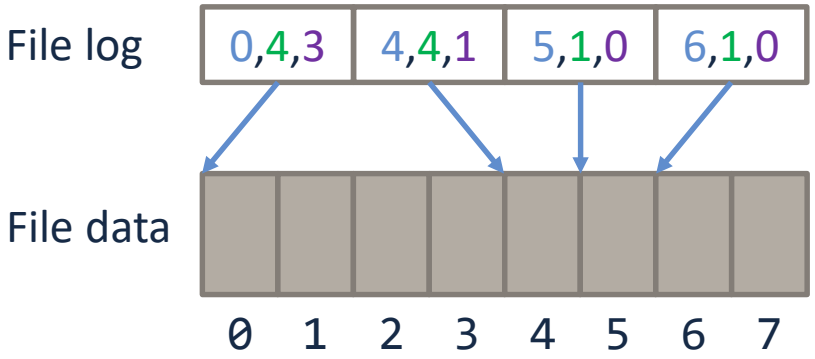
Data blocks written: 0 / 4

Asynchronous

Write entry offset, length

Write Size Predictor

- *Predict whether the incoming writes are both **large** and **stable***

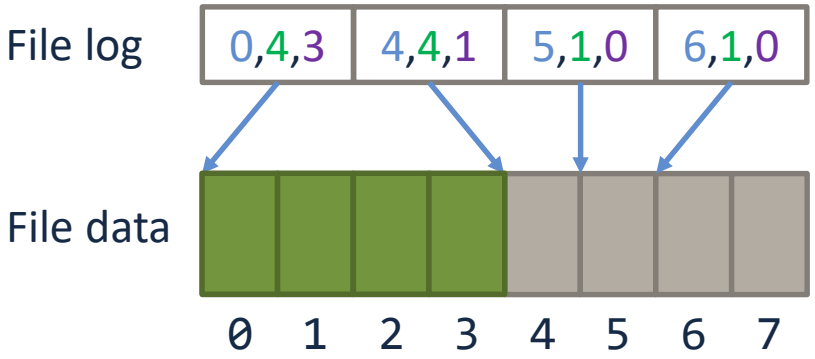



```
write(0,4);
write(6,1);
write(4,4);
```

Write entry | offset, length, counter

Write Size Predictor

- *Predict whether the incoming writes are both **large** and **stable***



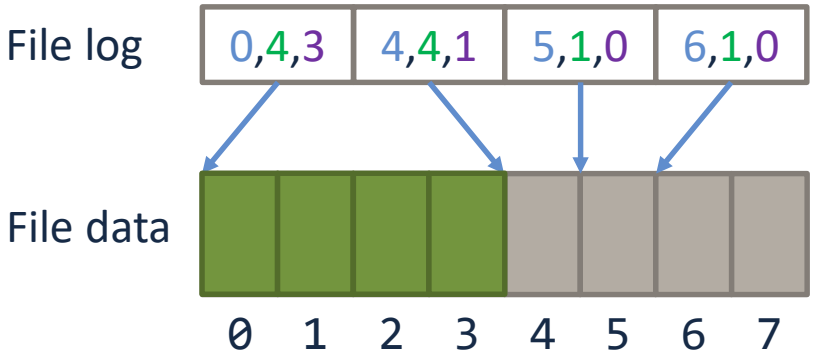
0,4,?

```
write(0,4);
write(6,1);
write(4,4);
```

Write entry | offset, length, counter

Write Size Predictor

- *Predict whether the incoming writes are both **large** and **stable***




```
write(0,4);
write(6,1);
write(4,4);
```

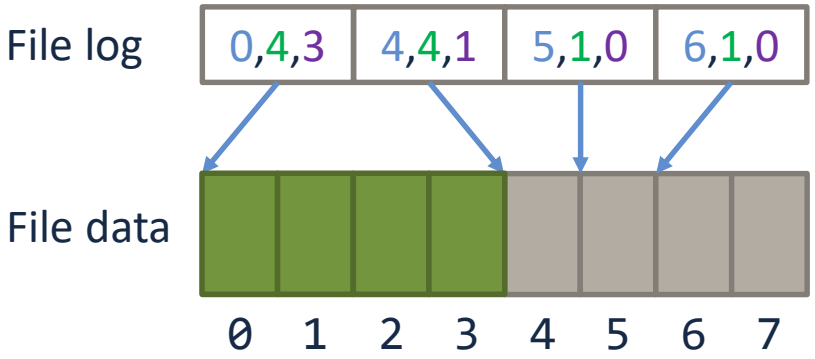
0,4,? Length ≥ 4 → Large

Predecessor found → Stable

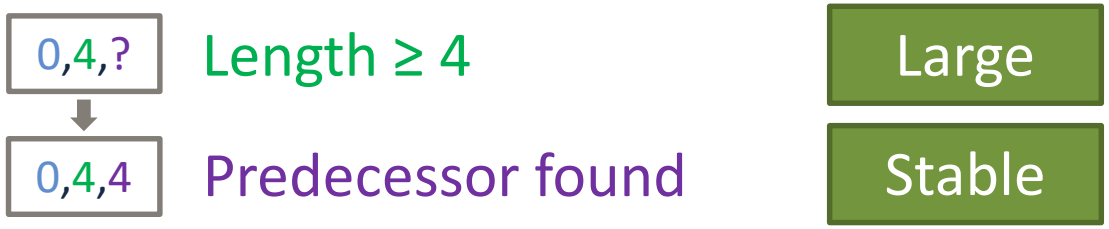
Write entry: offset, length, counter

Write Size Predictor

- *Predict whether the incoming writes are both **large** and **stable***



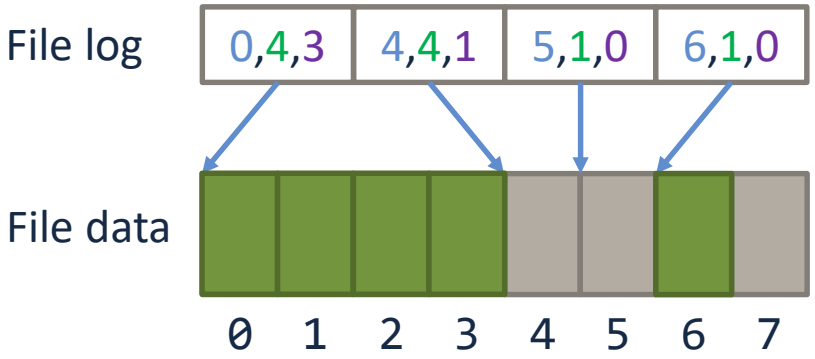

```
write(0,4);
write(6,1);
write(4,4);
```



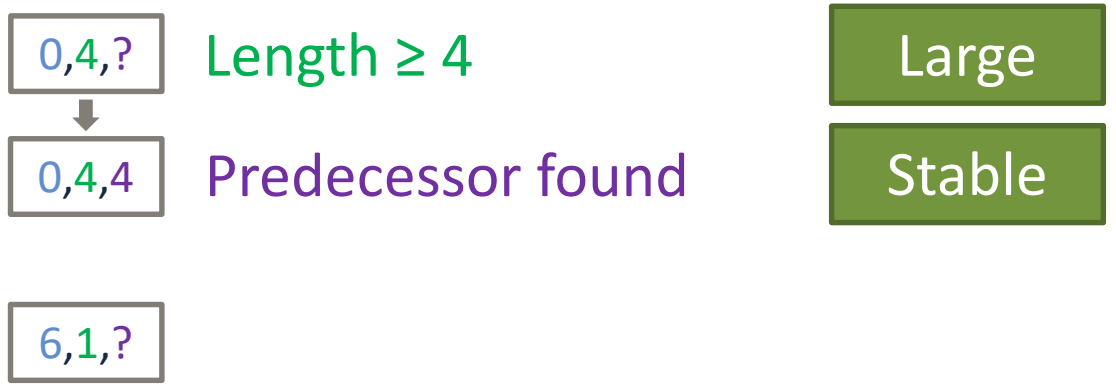
Write entry | offset, length, counter

Write Size Predictor

- *Predict whether the incoming writes are both **large** and **stable***



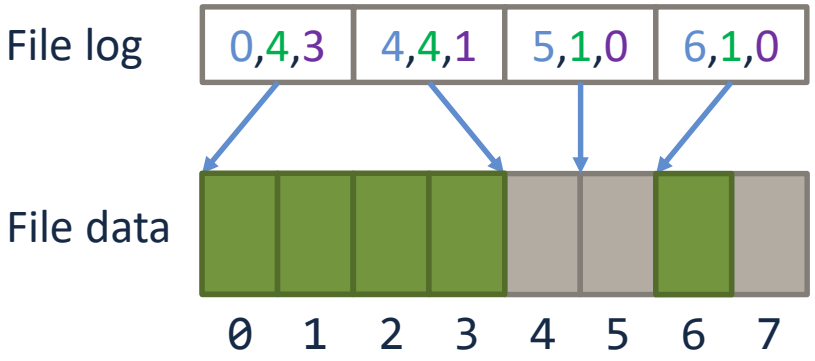

```
  write(0,4);
➡ write(6,1);
  write(4,4);
```



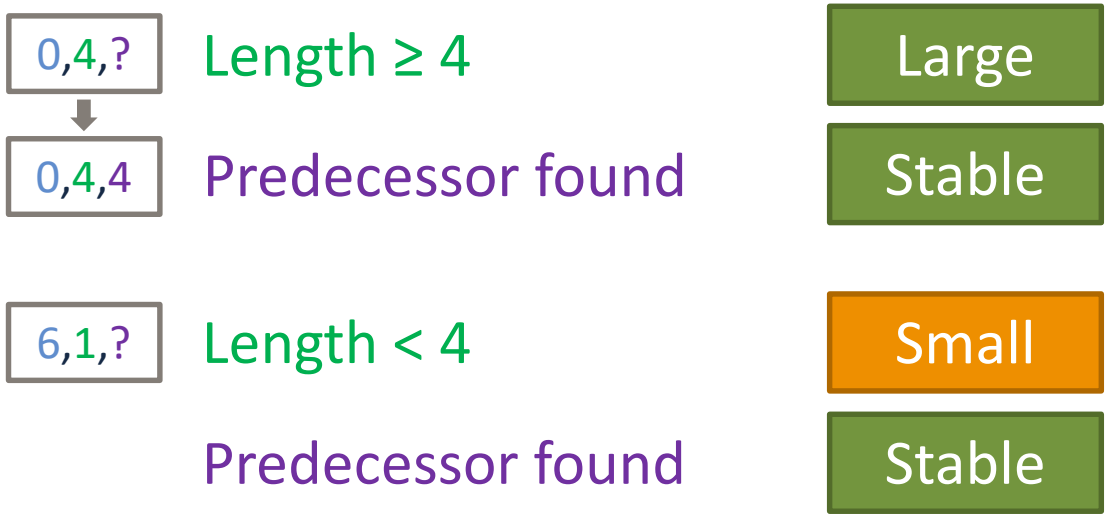
Write entry [offset, length, counter]

Write Size Predictor

- *Predict whether the incoming writes are both **large** and **stable***



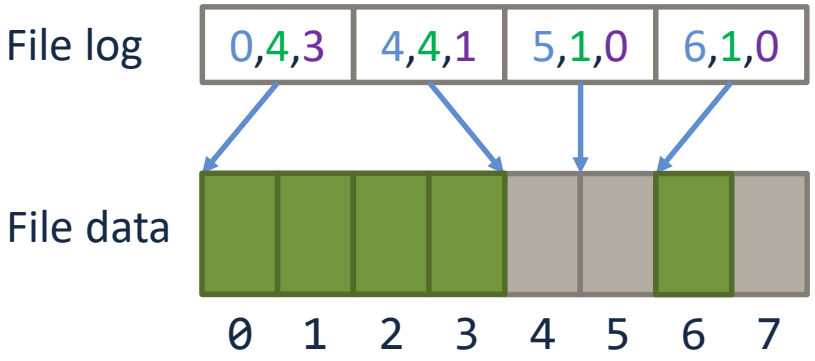

```
  write(0,4);
➡ write(6,1);
  write(4,4);
```



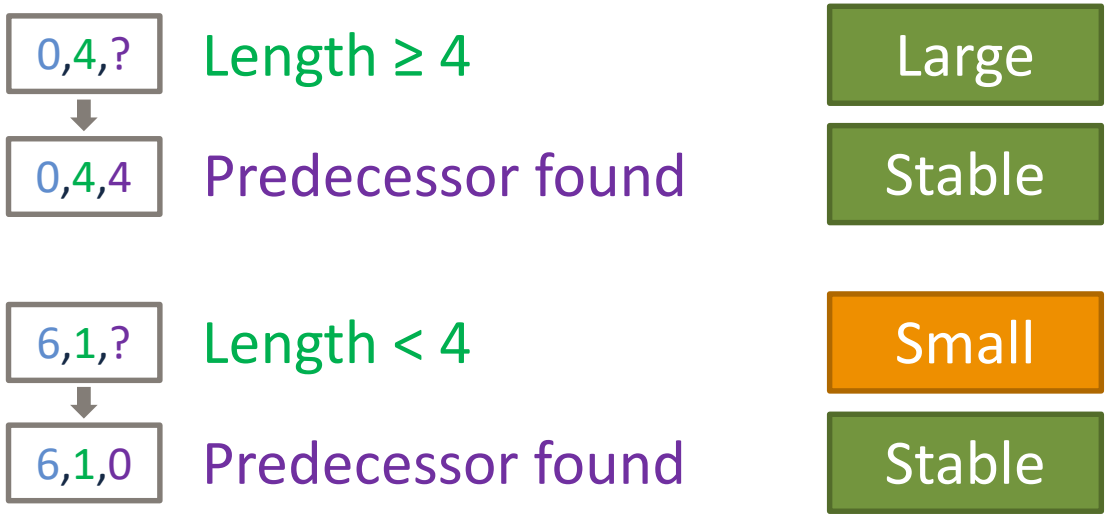
Write entry [offset, length, counter]

Write Size Predictor

- *Predict whether the incoming writes are both **large** and **stable***



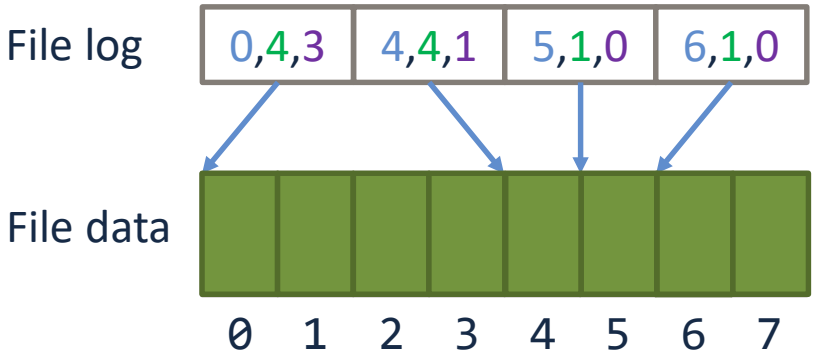

```
  write(0,4);
➡ write(6,1);
  write(4,4);
```



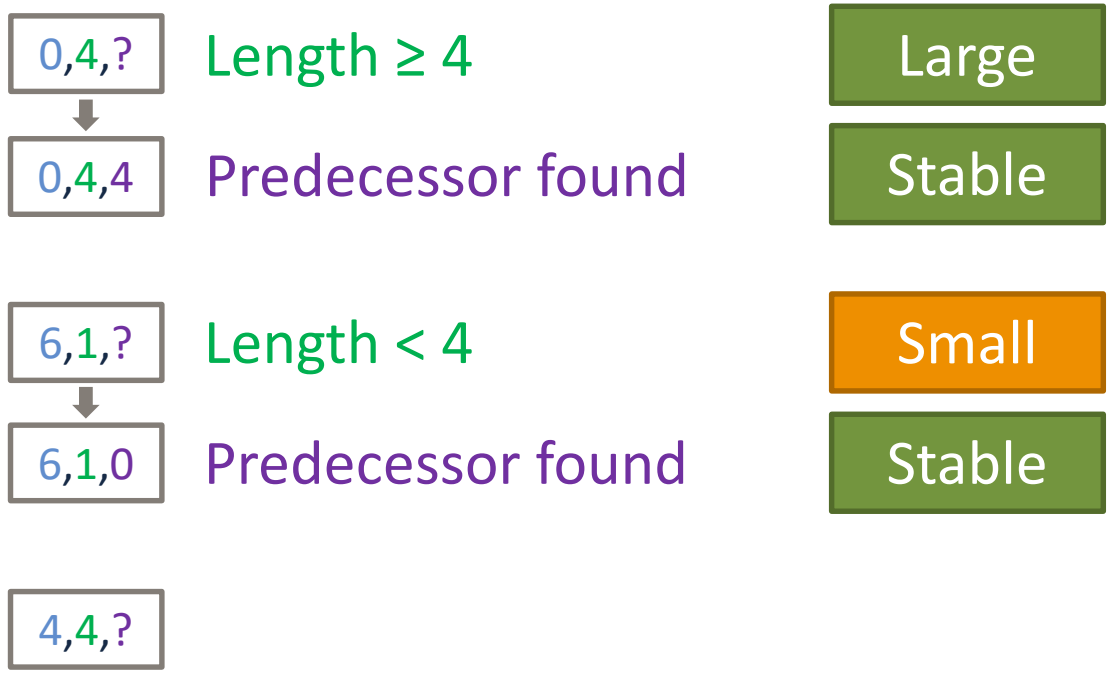
Write entry | offset, length, counter

Write Size Predictor

- *Predict whether the incoming writes are both **large** and **stable***



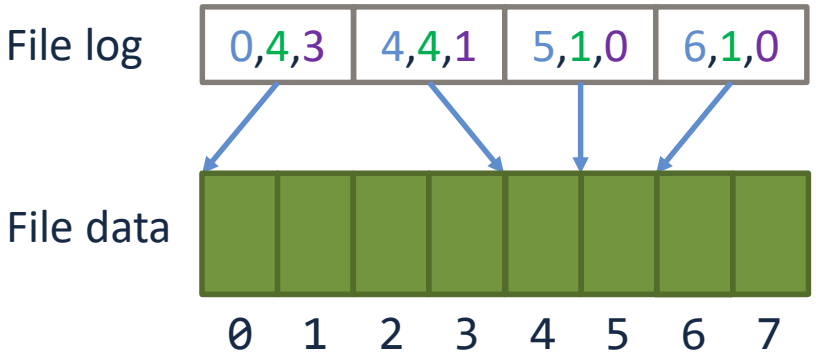

```
write(0,4);
write(6,1);
write(4,4);
```

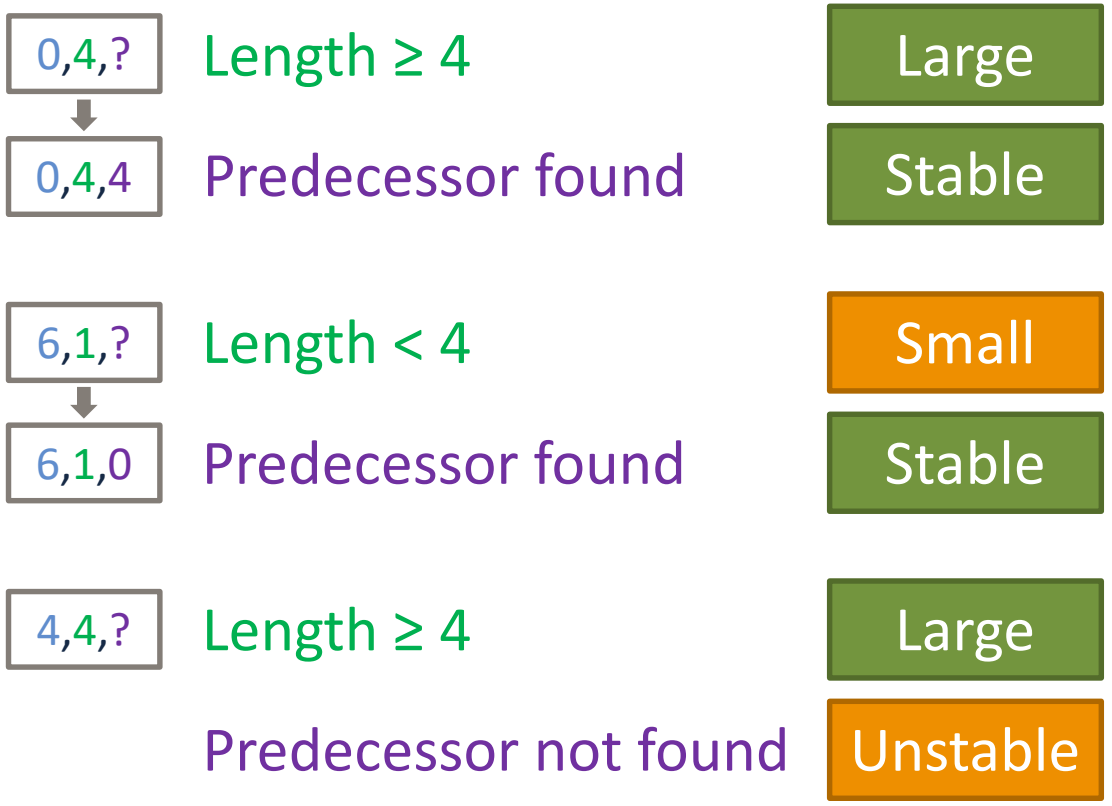


Write entry | offset, length, counter

Write Size Predictor

- *Predict whether the incoming writes are both **large** and **stable***

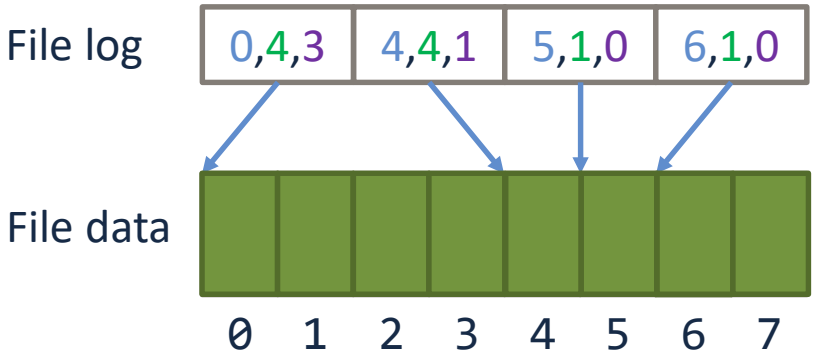




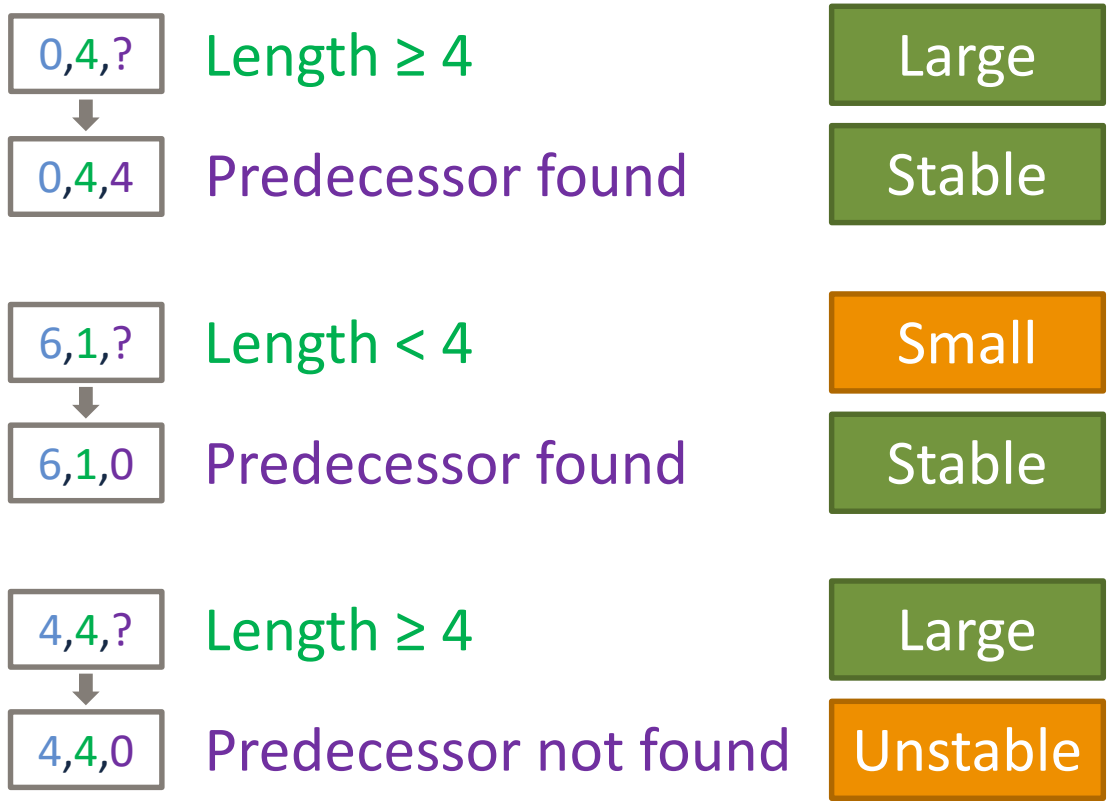
Write entry | offset, length, counter

Write Size Predictor

- *Predict whether the incoming writes are both **large** and **stable***



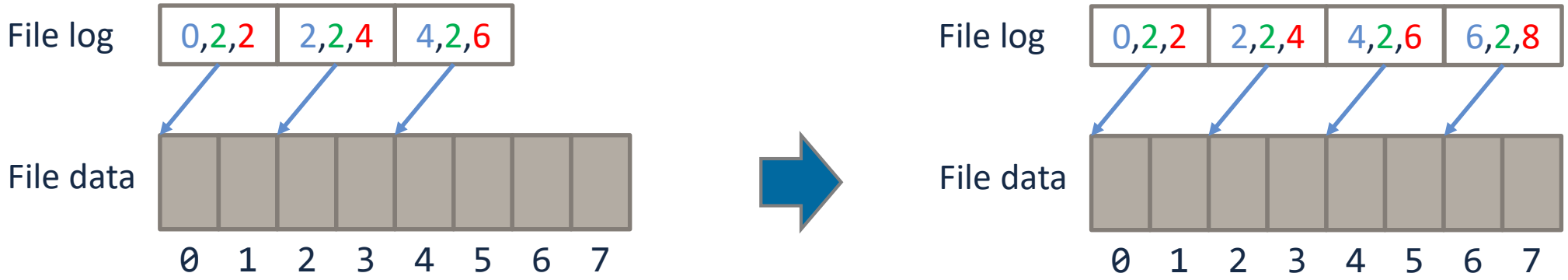

```
write(0,4);
write(6,1);
write(4,4);
```



Write entry | offset, length, counter

Cold Data Identification

- Average modification time (amtime)
  - Updated differentially



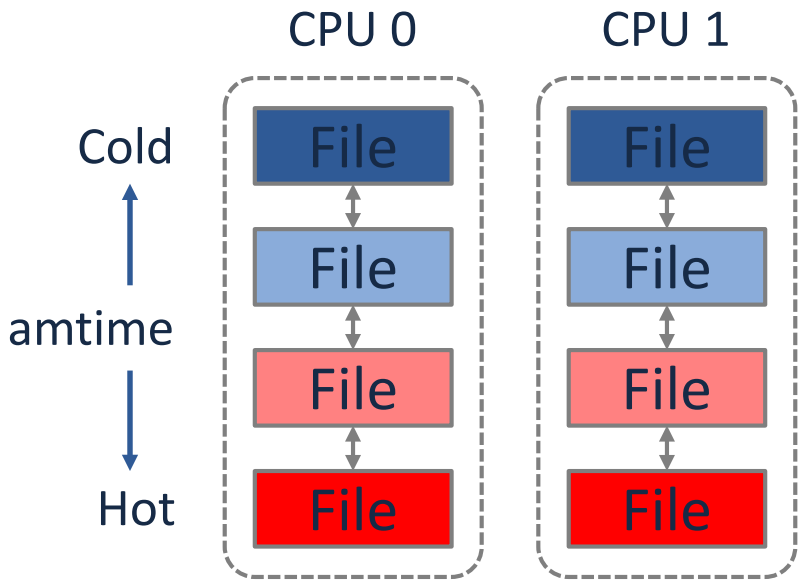
$$\text{amtime} = \frac{2*2+4*2+6*2}{2+2+2} = 4$$

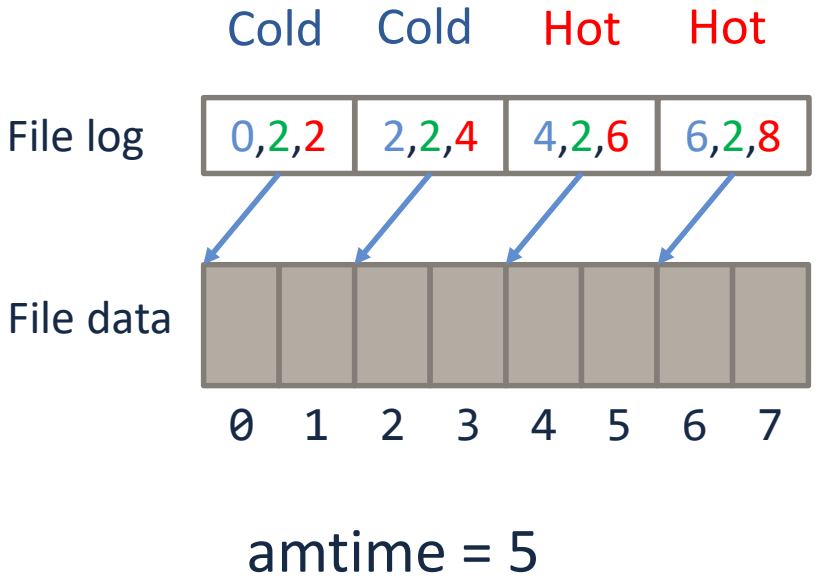
$$\text{amtime} = \frac{4*6+8*2}{6+2} = 5$$

Write entry | offset, length, mtime

Cold Data Identification

- Among files
  - Cold lists sorted by amtime
- Within each file
  - Cold data blocks relative to amtime



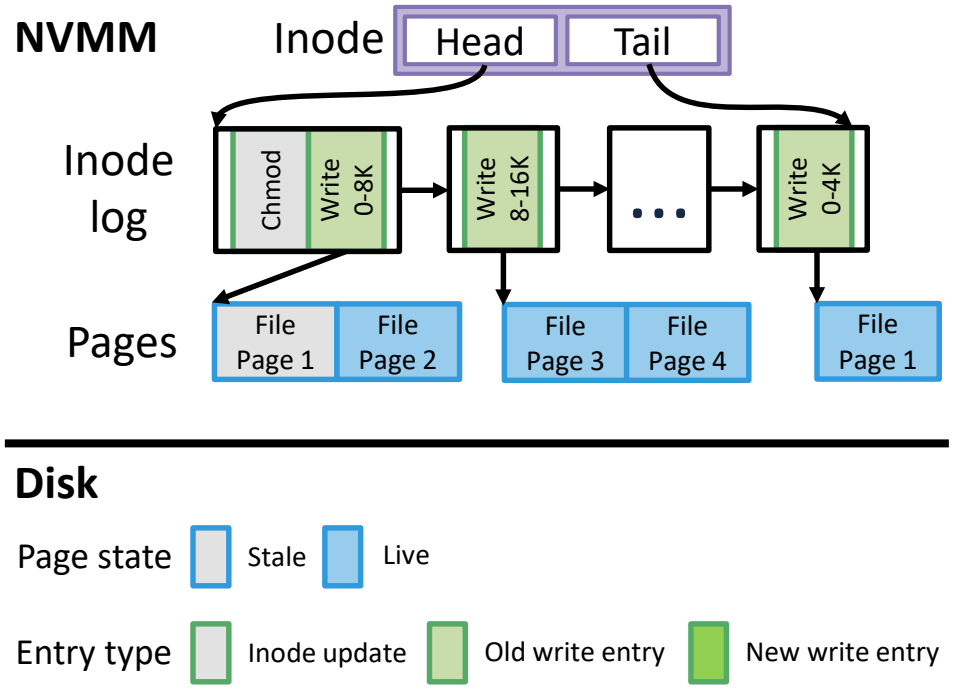


amtime = 5

Write entry | offset, length, mtime

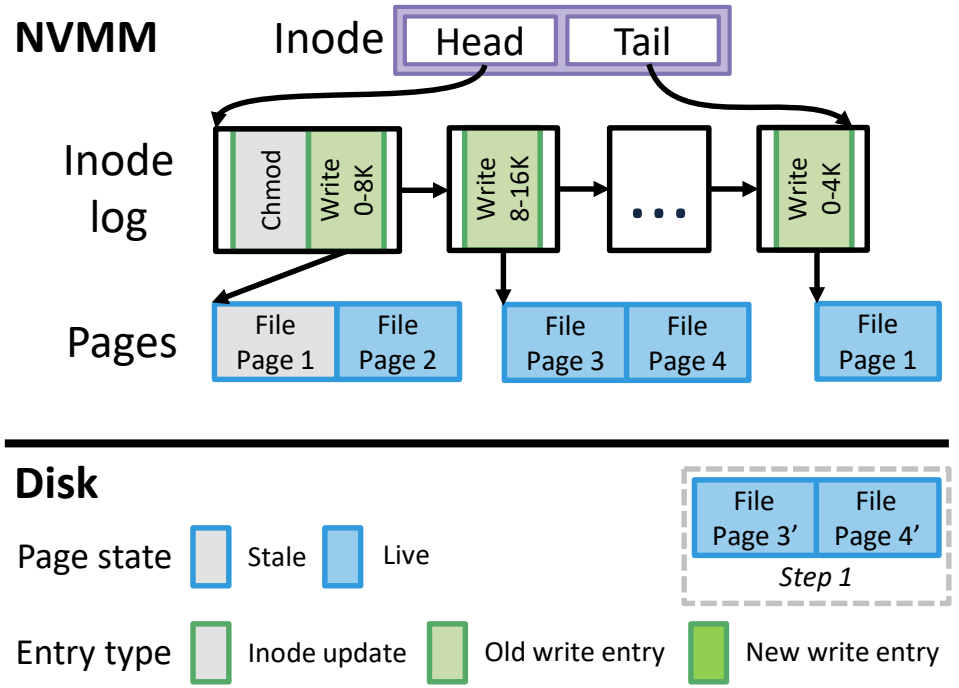
Migration Mechanism

- Basic migration
  - Migrate the coldest data to disk
  - Consistency is ensured
- Reverse migration
  - Migrate from disk to NVMM
  - Handle mmap
  - Read-dominated workloads



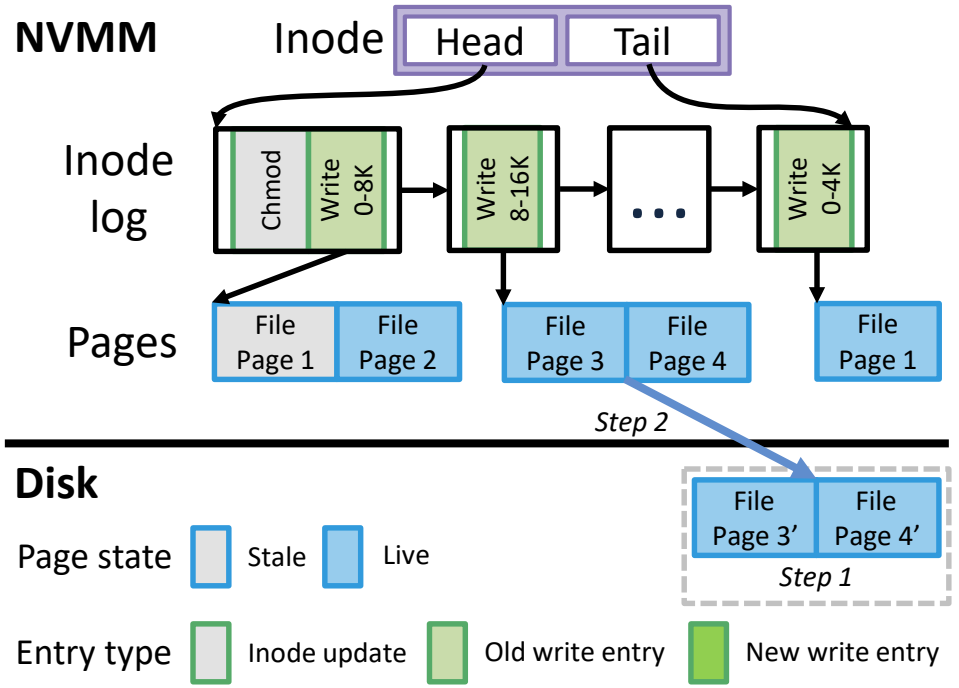
Migration Mechanism

- Basic migration
  - Migrate the coldest data to disk
  - Consistency is ensured
- Reverse migration
  - Migrate from disk to NVMM
  - Handle mmap
  - Read-dominated workloads



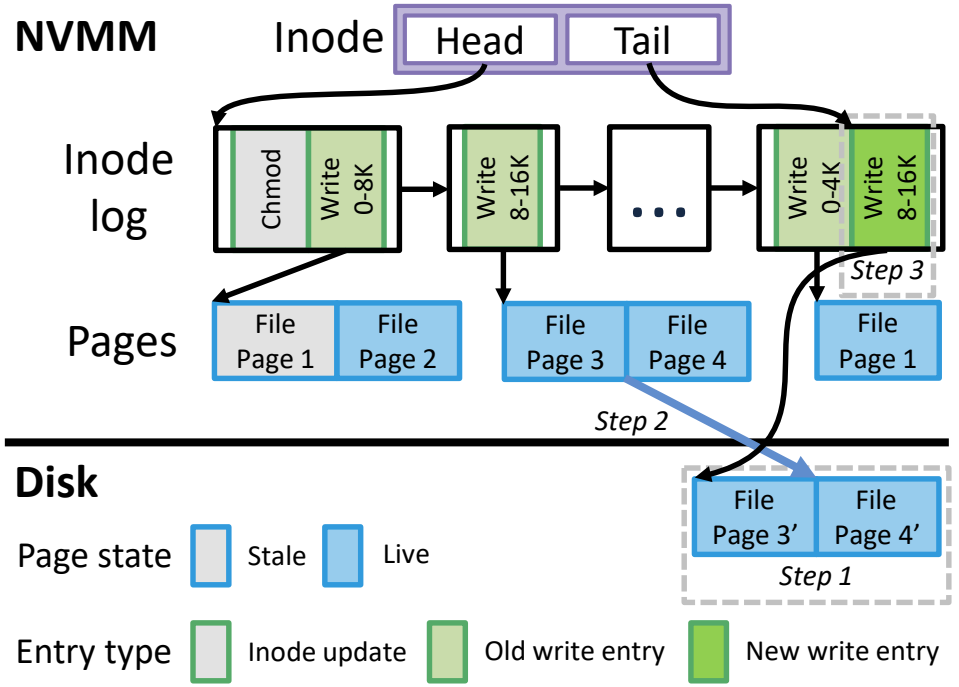
Migration Mechanism

- Basic migration
  - Migrate the coldest data to disk
  - Consistency is ensured
- Reverse migration
  - Migrate from disk to NVMM
  - Handle mmap
  - Read-dominated workloads



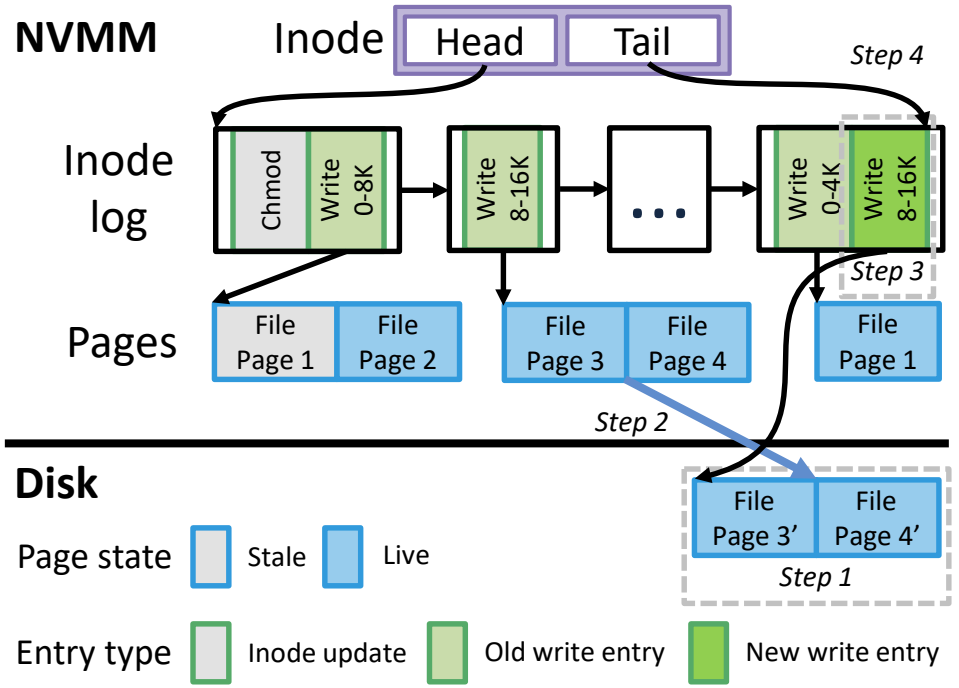
Migration Mechanism

- Basic migration
  - Migrate the coldest data to disk
  - Consistency is ensured
- Reverse migration
  - Migrate from disk to NVMM
  - Handle mmap
  - Read-dominated workloads



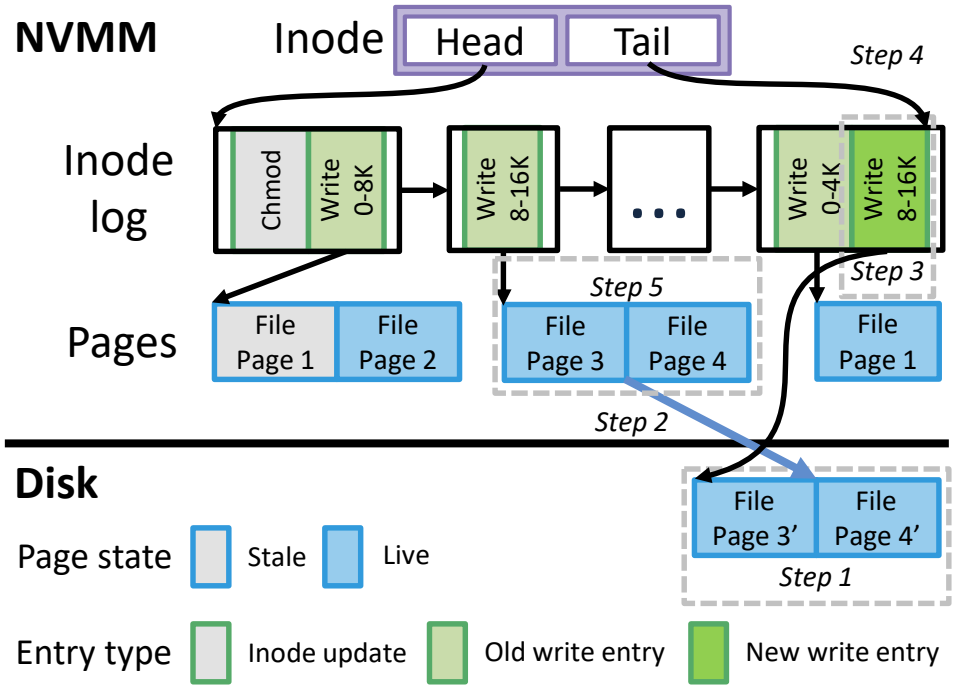
Migration Mechanism

- Basic migration
  - Migrate the coldest data to disk
  - Consistency is ensured
- Reverse migration
  - Migrate from disk to NVMM
  - Handle mmap
  - Read-dominated workloads



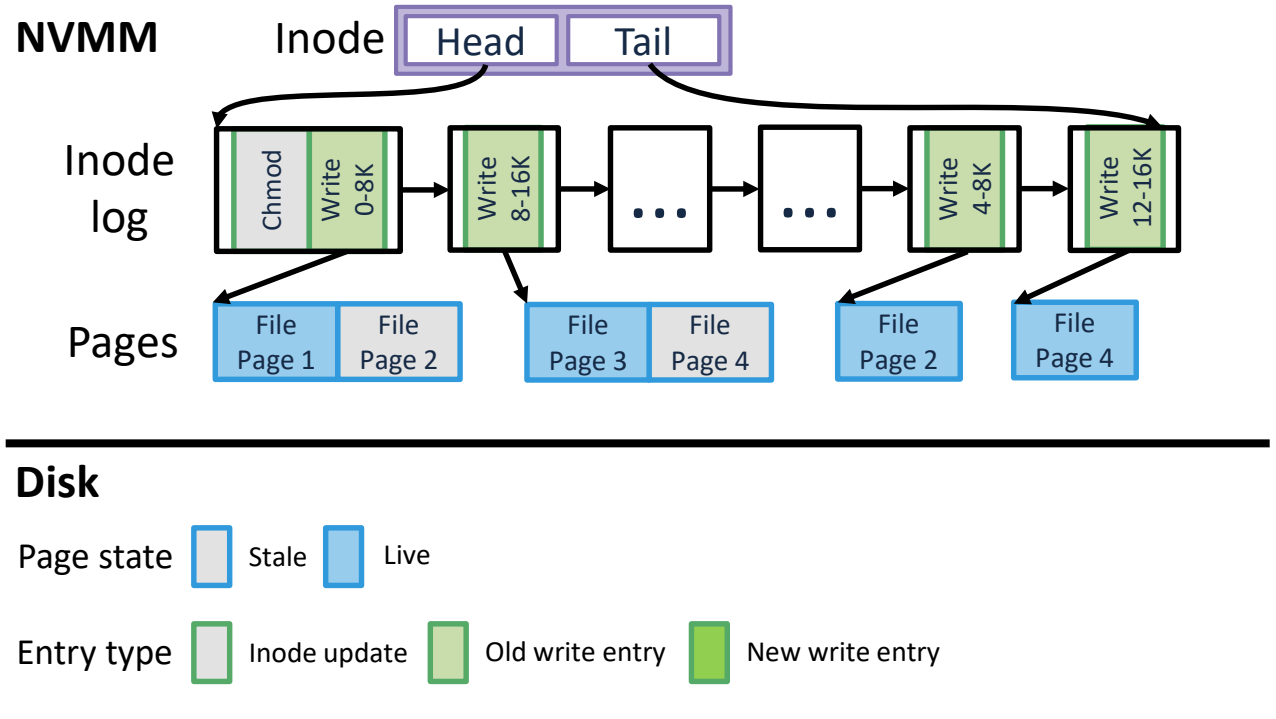
Migration Mechanism

- Basic migration
  - Migrate the coldest data to disk
  - Consistency is ensured
- Reverse migration
  - Migrate from disk to NVMM
  - Handle mmap
  - Read-dominated workloads



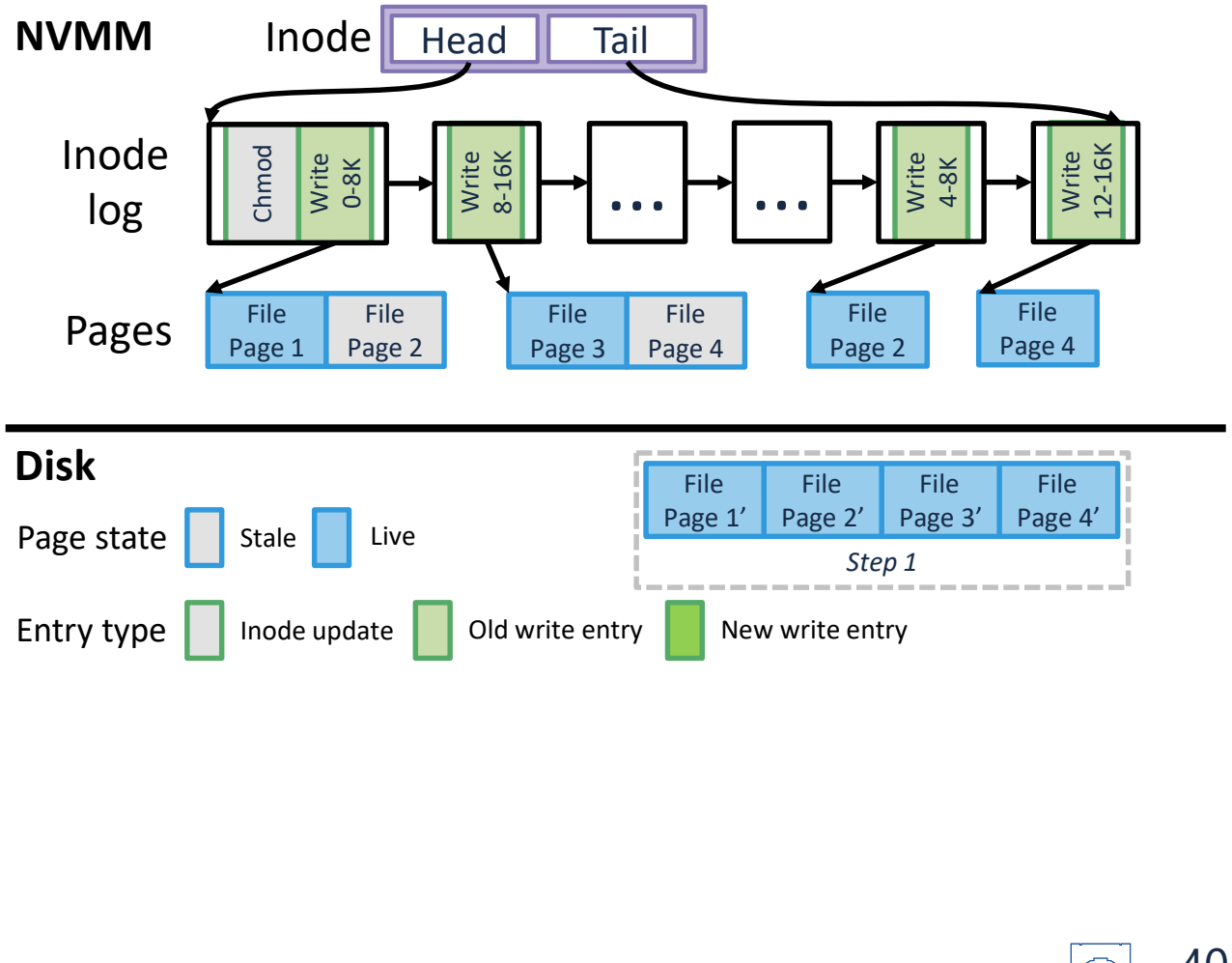
Migration Mechanism

- Group migration
  - Coalesce adjacent write entries
  - Utilize the high sequential bandwidth of disks
- Benefits
  - Improve migration efficiency
  - Accelerate future reads
  - Reduce log size
  - Moderate disk fragmentation



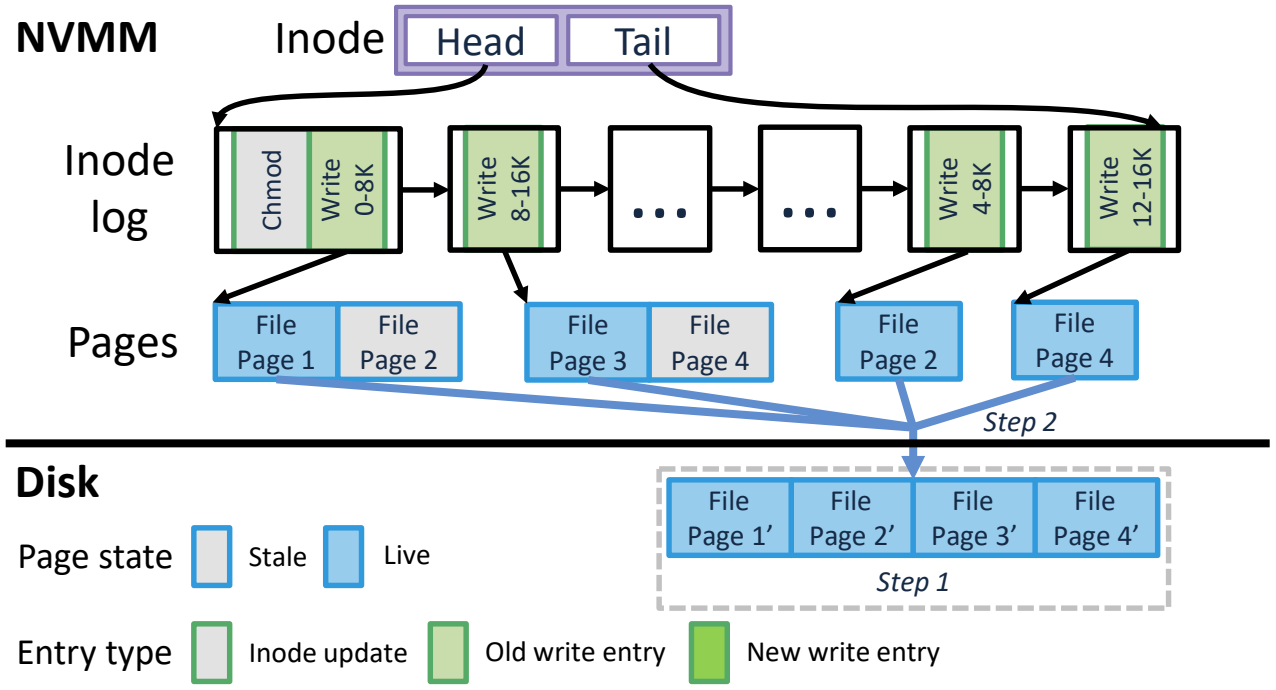
Migration Mechanism

- Group migration
  - Coalesce adjacent write entries
  - Utilize the high sequential bandwidth of disks
- Benefits
  - Improve migration efficiency
  - Accelerate future reads
  - Reduce log size
  - Moderate disk fragmentation



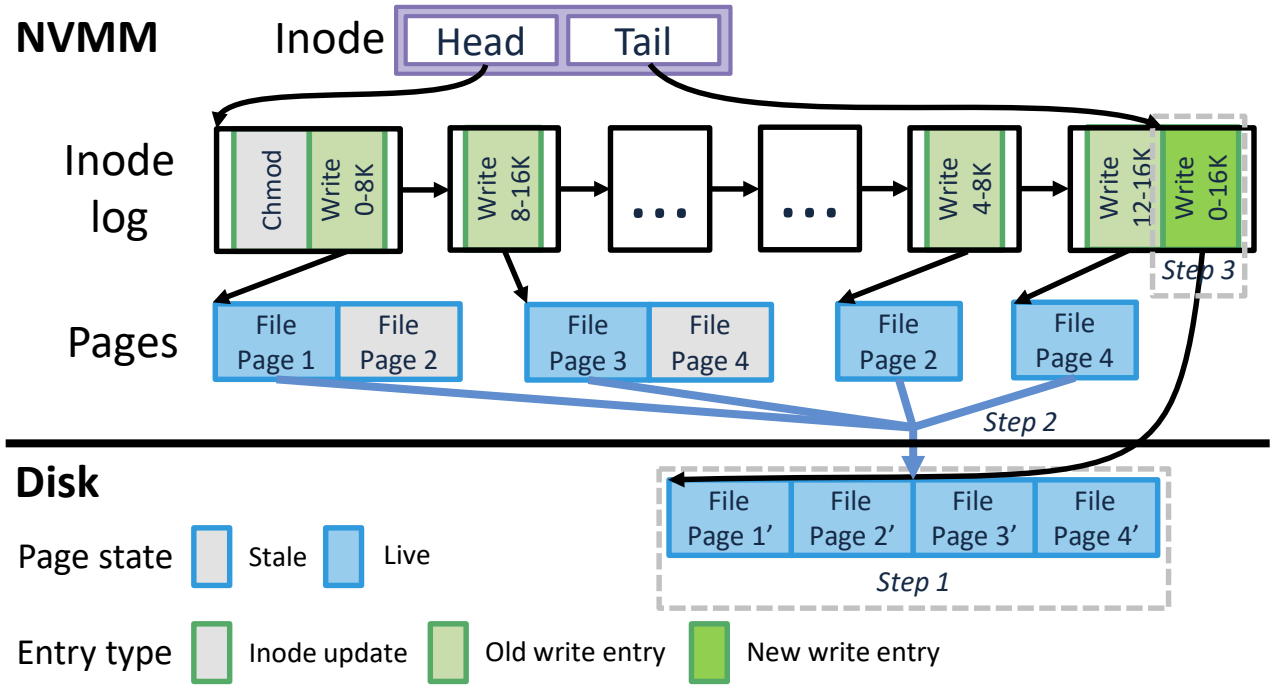
Migration Mechanism

- Group migration
  - Coalesce adjacent write entries
  - Utilize the high sequential bandwidth of disks
- Benefits
  - Improve migration efficiency
  - Accelerate future reads
  - Reduce log size
  - Moderate disk fragmentation



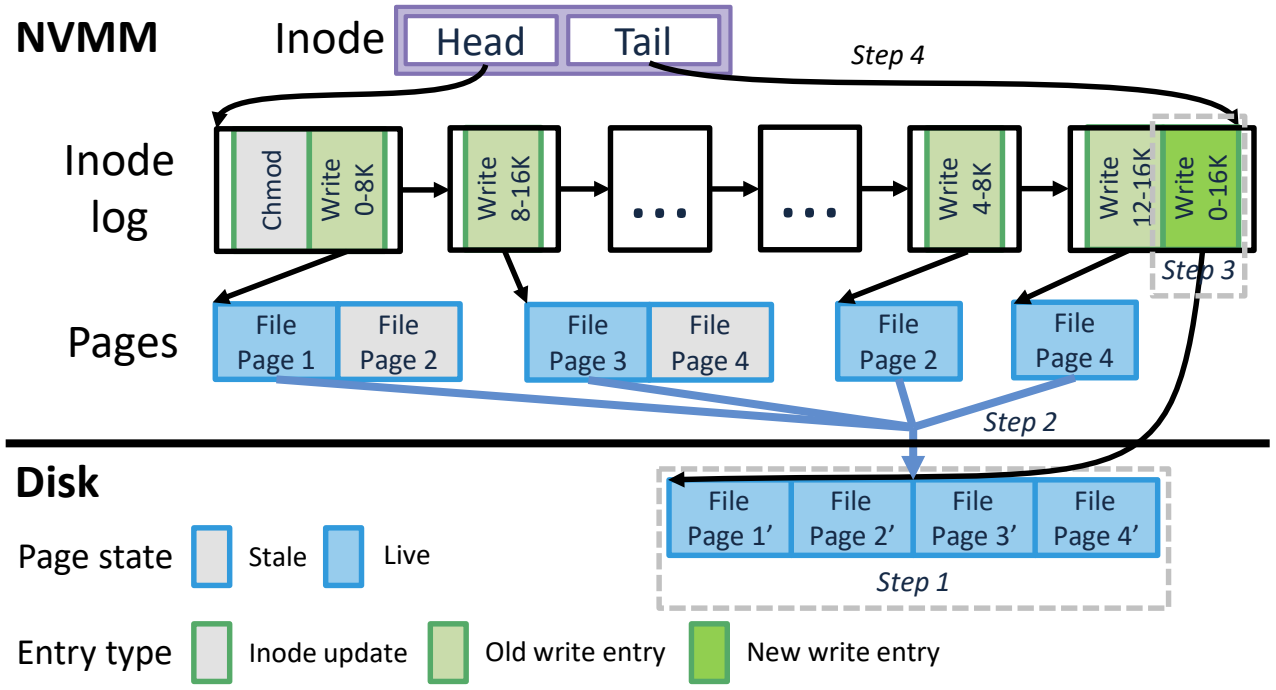
Migration Mechanism

- Group migration
  - Coalesce adjacent write entries
  - Utilize the high sequential bandwidth of disks
- Benefits
  - Improve migration efficiency
  - Accelerate future reads
  - Reduce log size
  - Moderate disk fragmentation



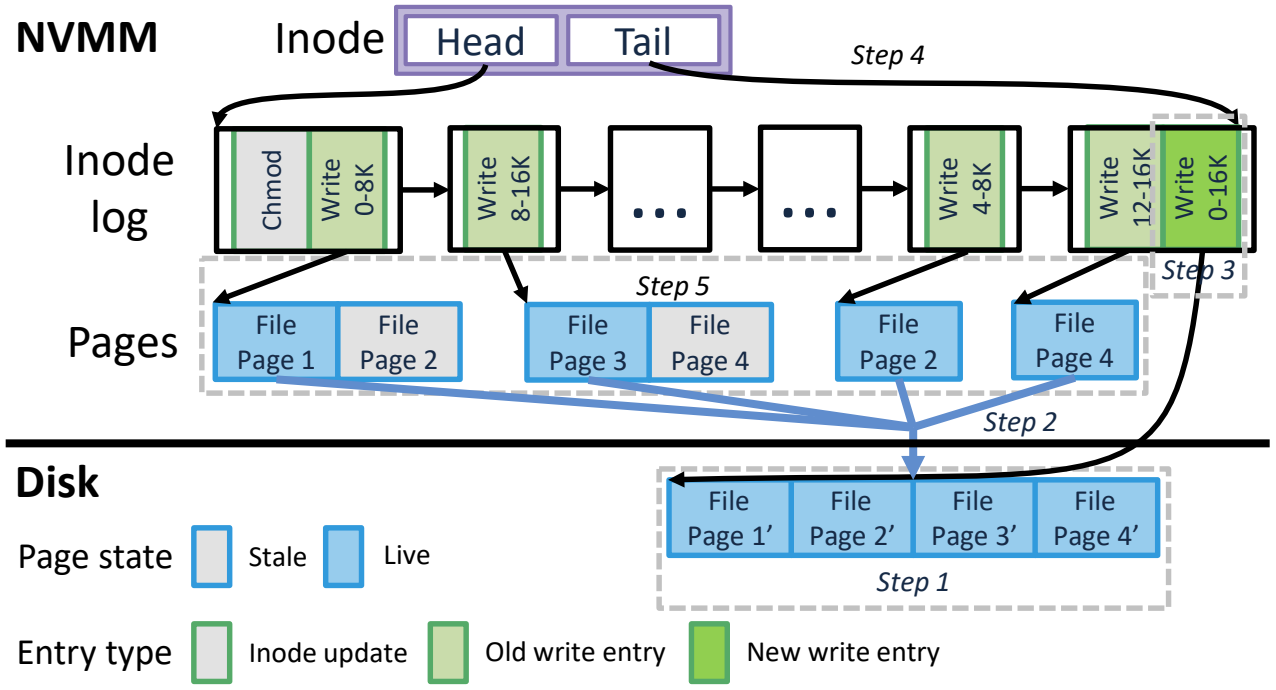
Migration Mechanism

- Group migration
  - Coalesce adjacent write entries
  - Utilize the high sequential bandwidth of disks
- Benefits
  - Improve migration efficiency
  - Accelerate future reads
  - Reduce log size
  - Moderate disk fragmentation



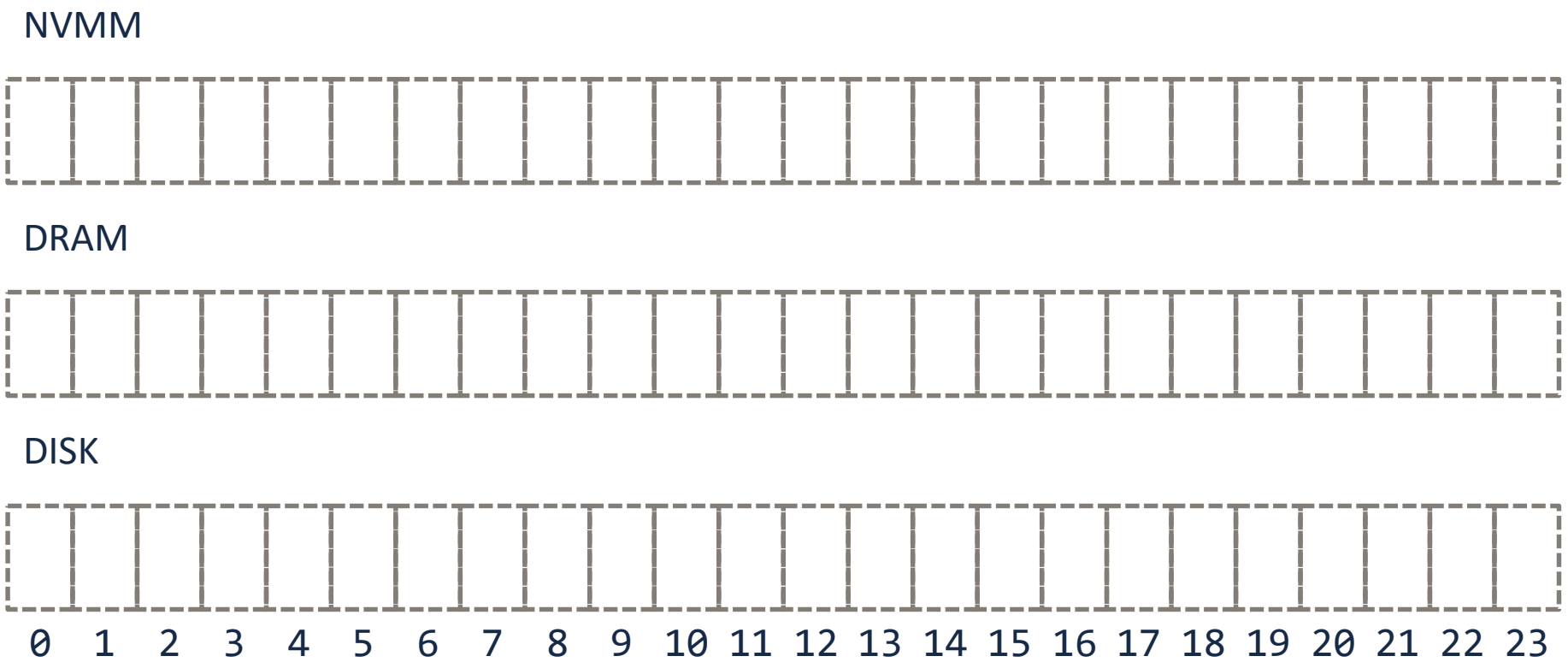
Migration Mechanism

- Group migration
  - Coalesce adjacent write entries
  - Utilize the high sequential bandwidth of disks
- Benefits
  - Improve migration efficiency
  - Accelerate future reads
  - Reduce log size
  - Moderate disk fragmentation



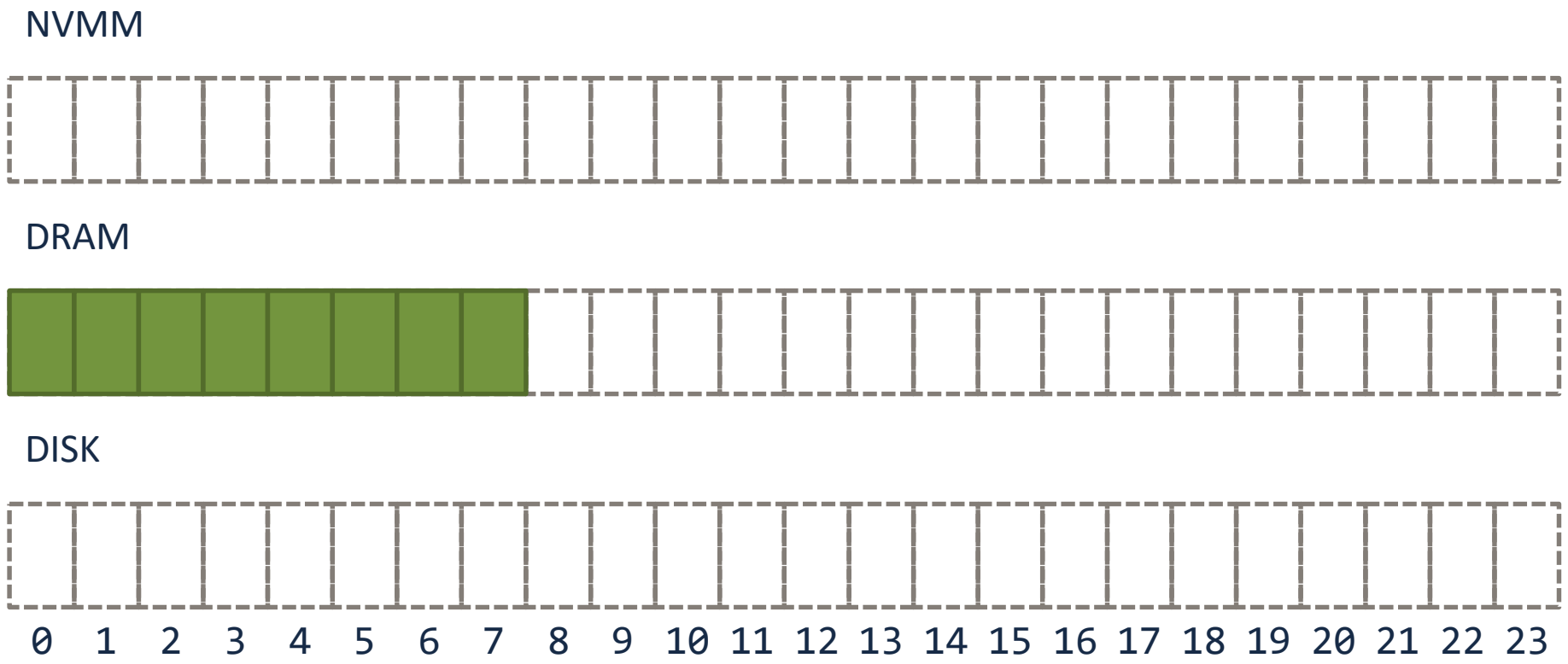
File Operations

```
write(0,8);
write(8,8);
fsync();
append(16,1);
append(17,1);
...
append(23,1);
......
migrate();
......
read(16,8);
mmap(0,8);
```



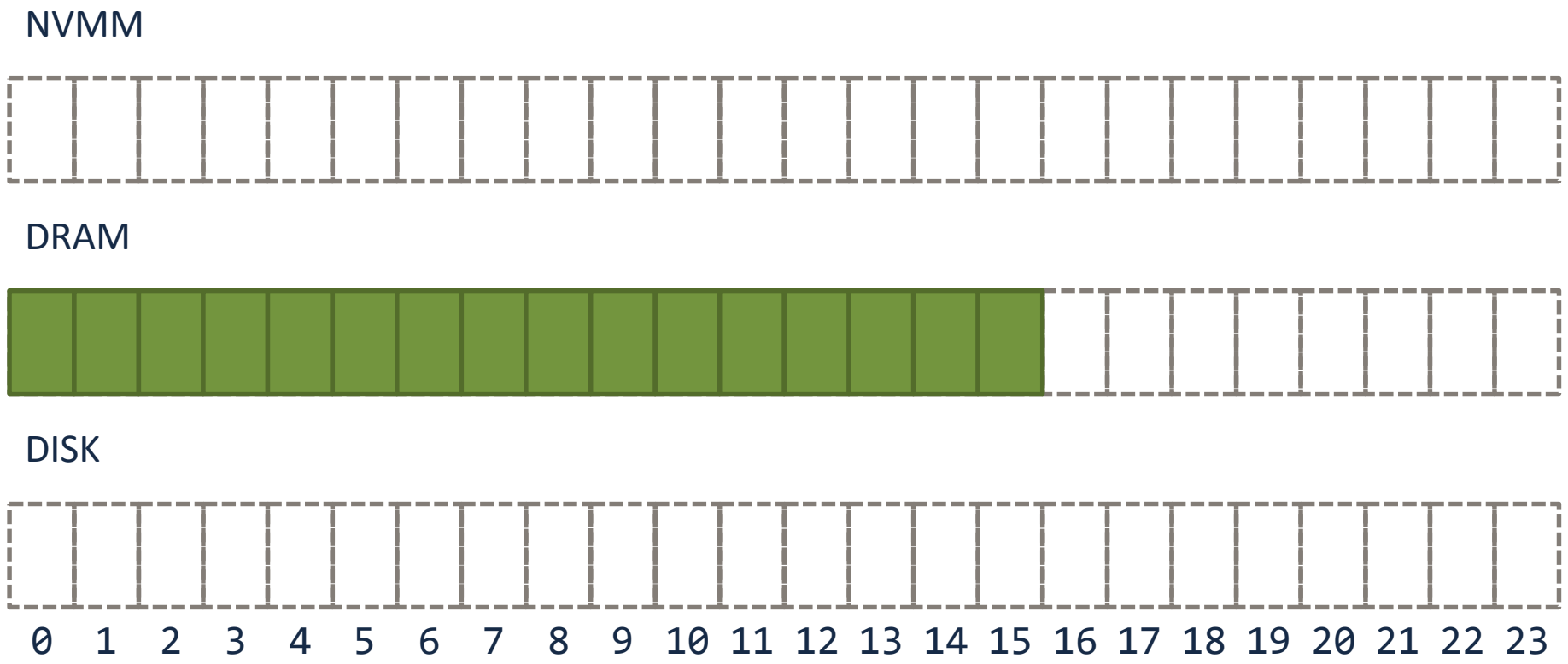
File Operations

```
write(0,8);
write(8,8);
fsync();
append(16,1);
append(17,1);
...
append(23,1);
......
migrate();
......
read(16,8);
mmap(0,8);
```



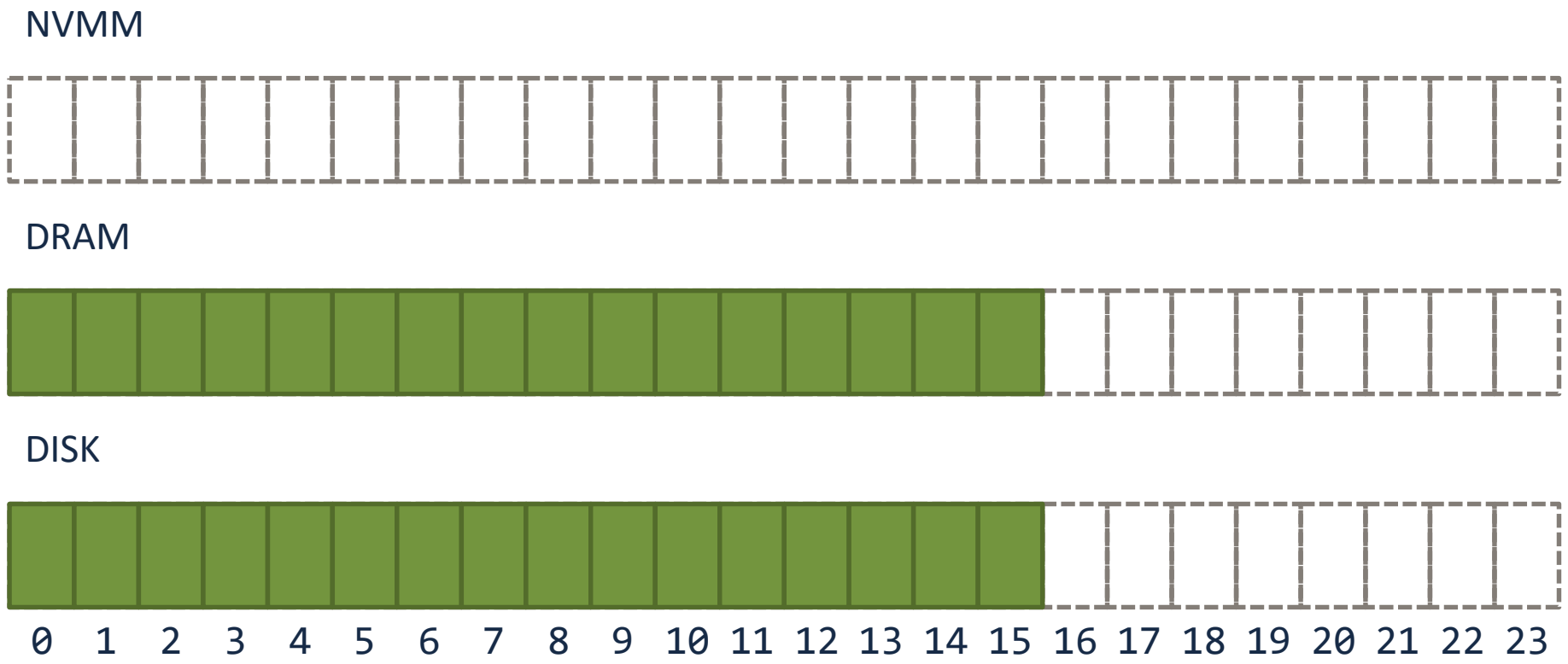
File Operations

```
write(0,8);
write(8,8);
fsync();
append(16,1);
append(17,1);
...
append(23,1);
......
migrate();
......
read(16,8);
mmap(0,8);
```



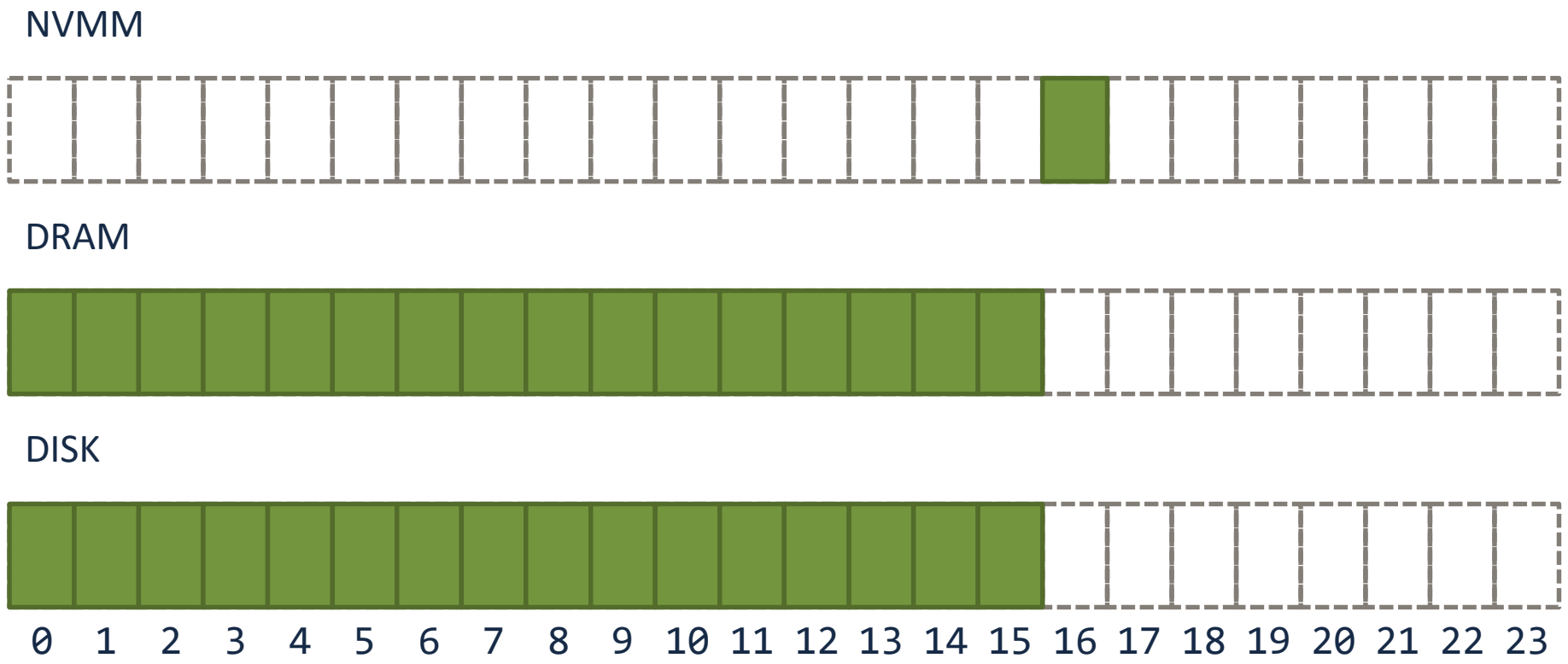
File Operations

```
write(0,8);
write(8,8);
fsync();
append(16,1);
append(17,1);
...
append(23,1);
......
migrate();
......
read(16,8);
mmap(0,8);
```



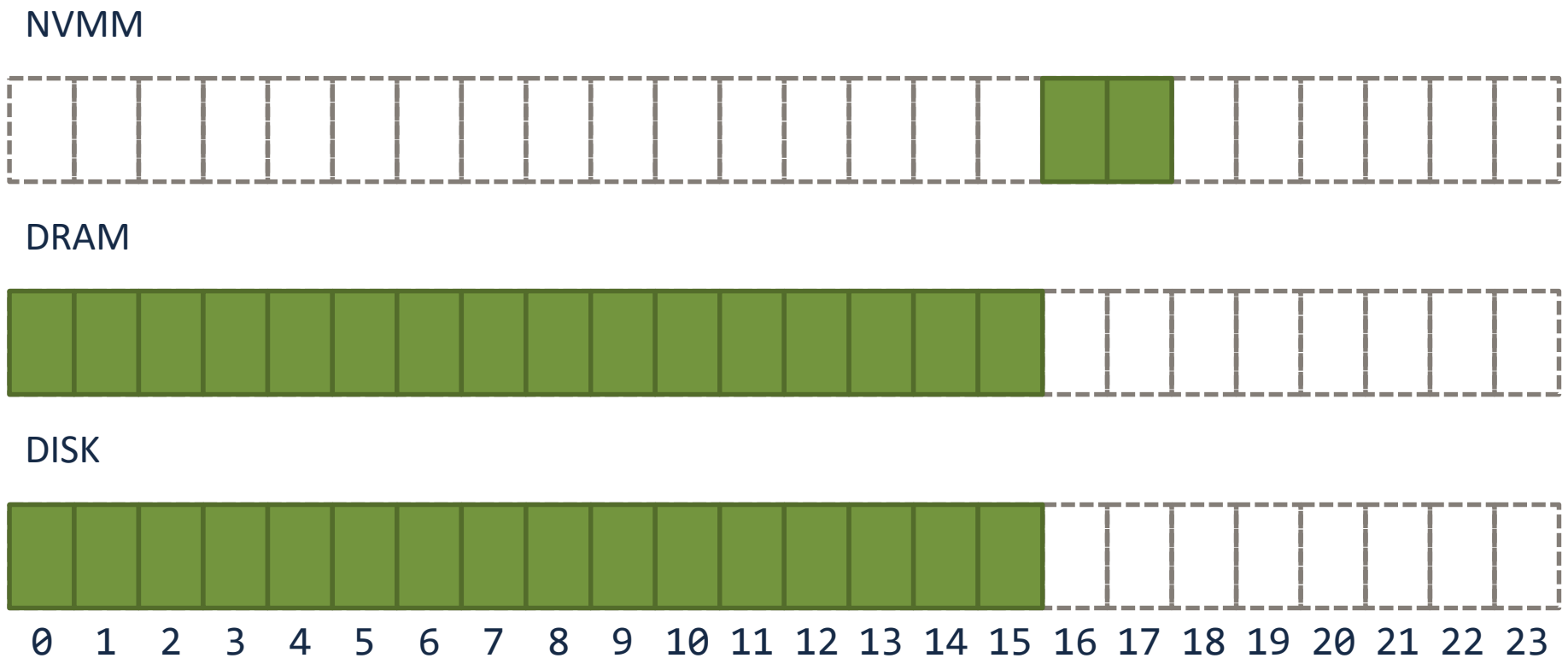
File Operations

```
write(0,8);
write(8,8);
fsync();
append(16,1);
append(17,1);
...
append(23,1);
......
migrate();
......
read(16,8);
mmap(0,8);
```



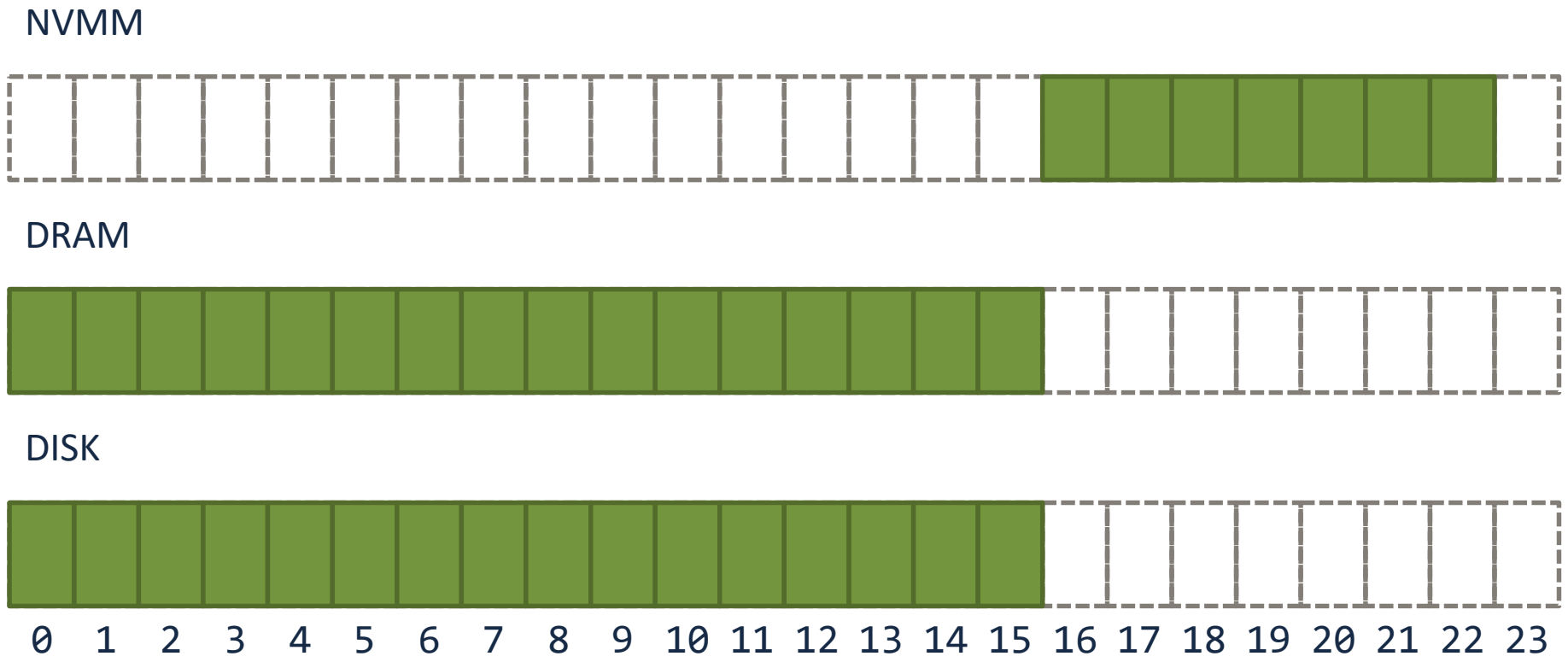
File Operations

```
write(0,8);
write(8,8);
fsync();
append(16,1);
append(17,1);
...
append(23,1);
......
migrate();
......
read(16,8);
mmap(0,8);
```



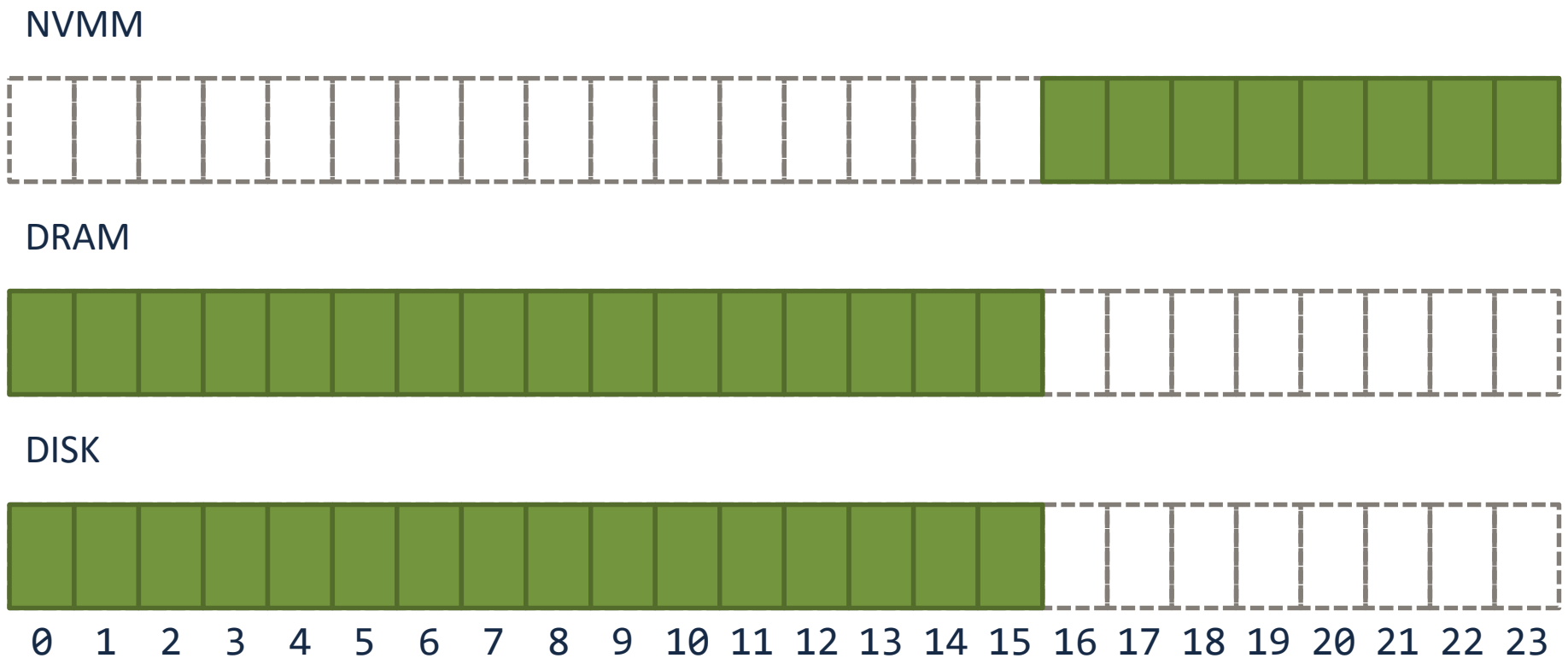
File Operations

```
write(0,8);
write(8,8);
fsync();
append(16,1);
append(17,1);
...
append(23,1);
......
migrate();
......
read(16,8);
mmap(0,8);
```



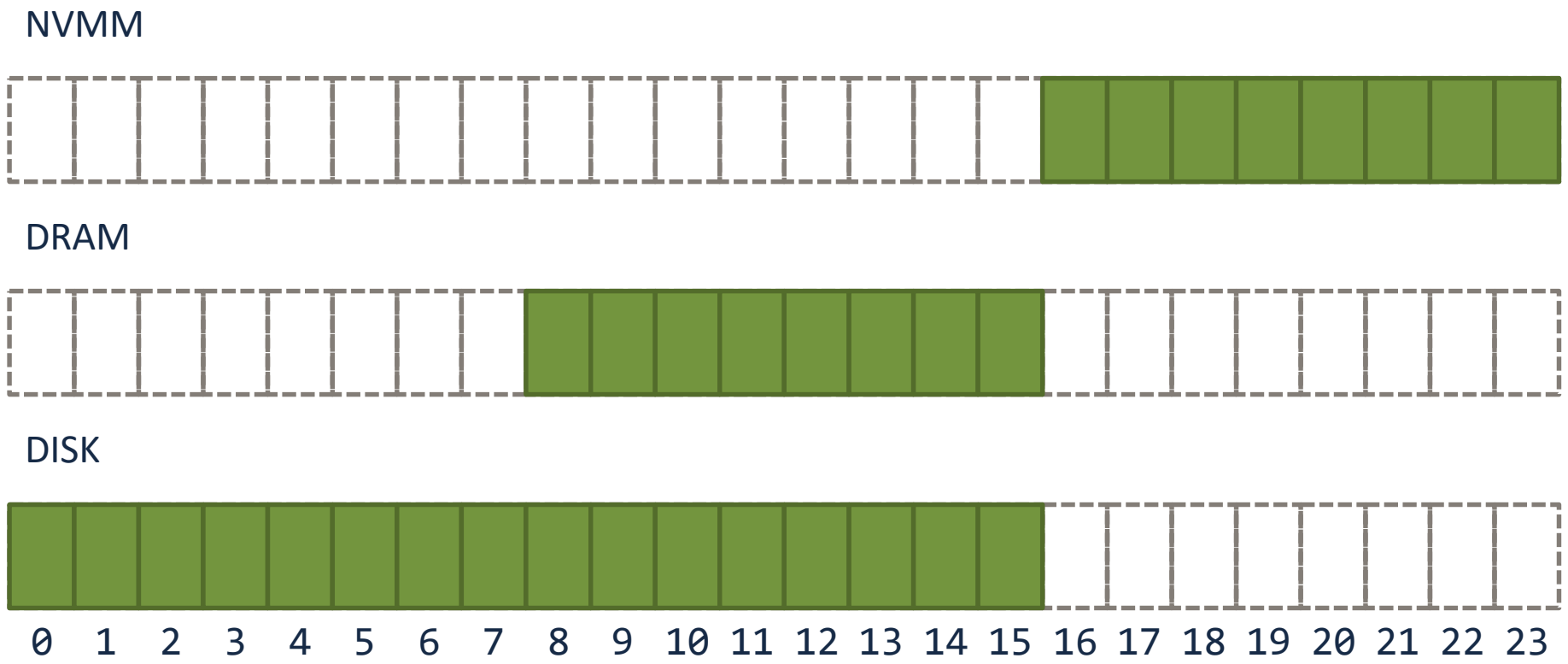
File Operations

```
write(0,8);
write(8,8);
fsync();
append(16,1);
append(17,1);
...
append(23,1);
......
migrate();
......
read(16,8);
mmap(0,8);
```



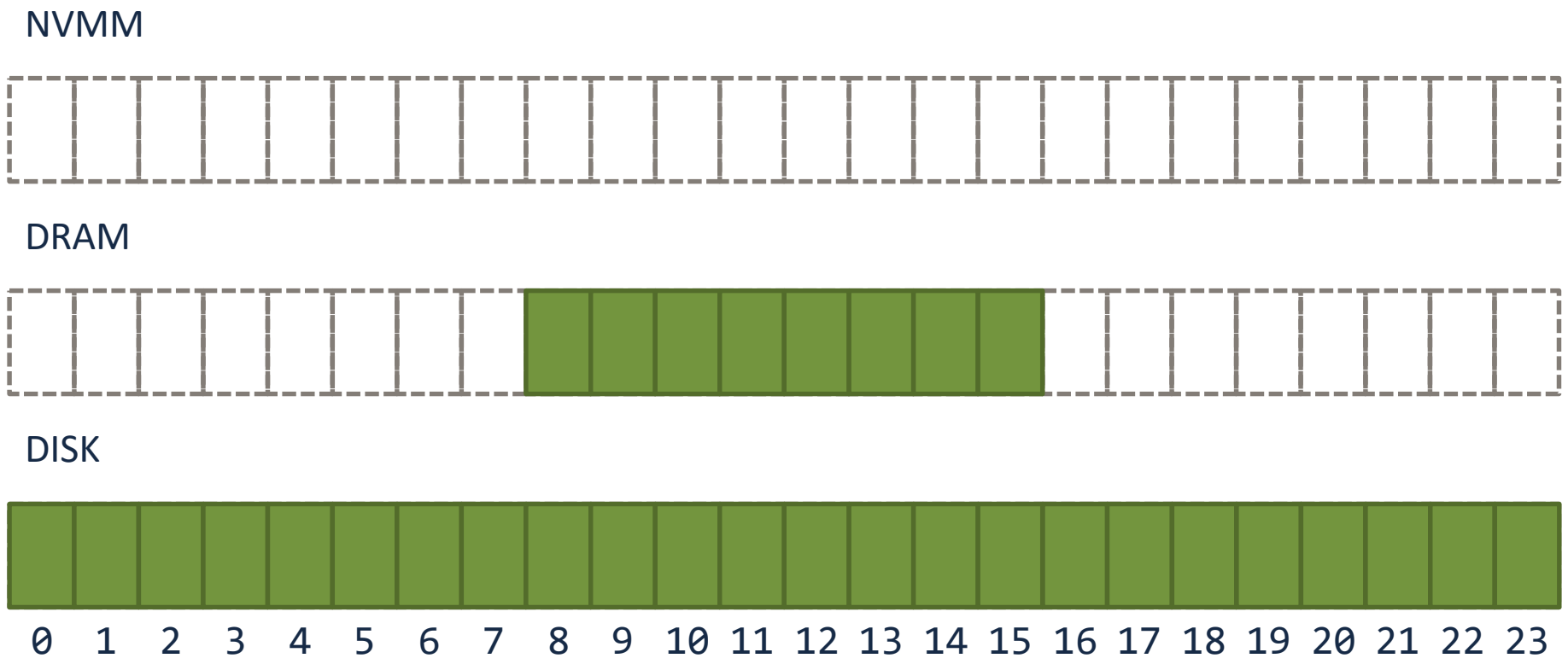
File Operations

```
write(0,8);
write(8,8);
fsync();
append(16,1);
append(17,1);
...
append(23,1);
......
migrate();
......
read(16,8);
mmap(0,8);
```



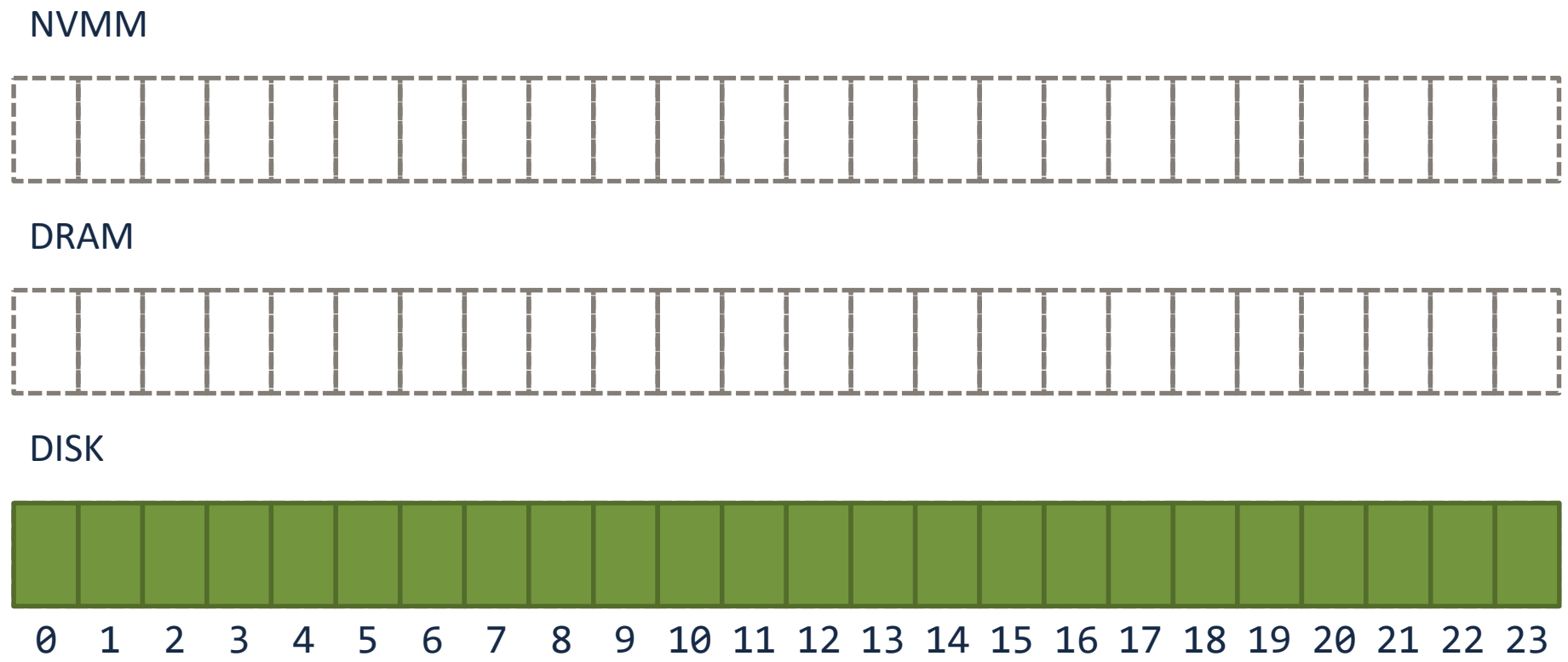
File Operations

```
write(0,8);
write(8,8);
fsync();
append(16,1);
append(17,1);
...
append(23,1);
......
migrate();
......
read(16,8);
mmap(0,8);
```



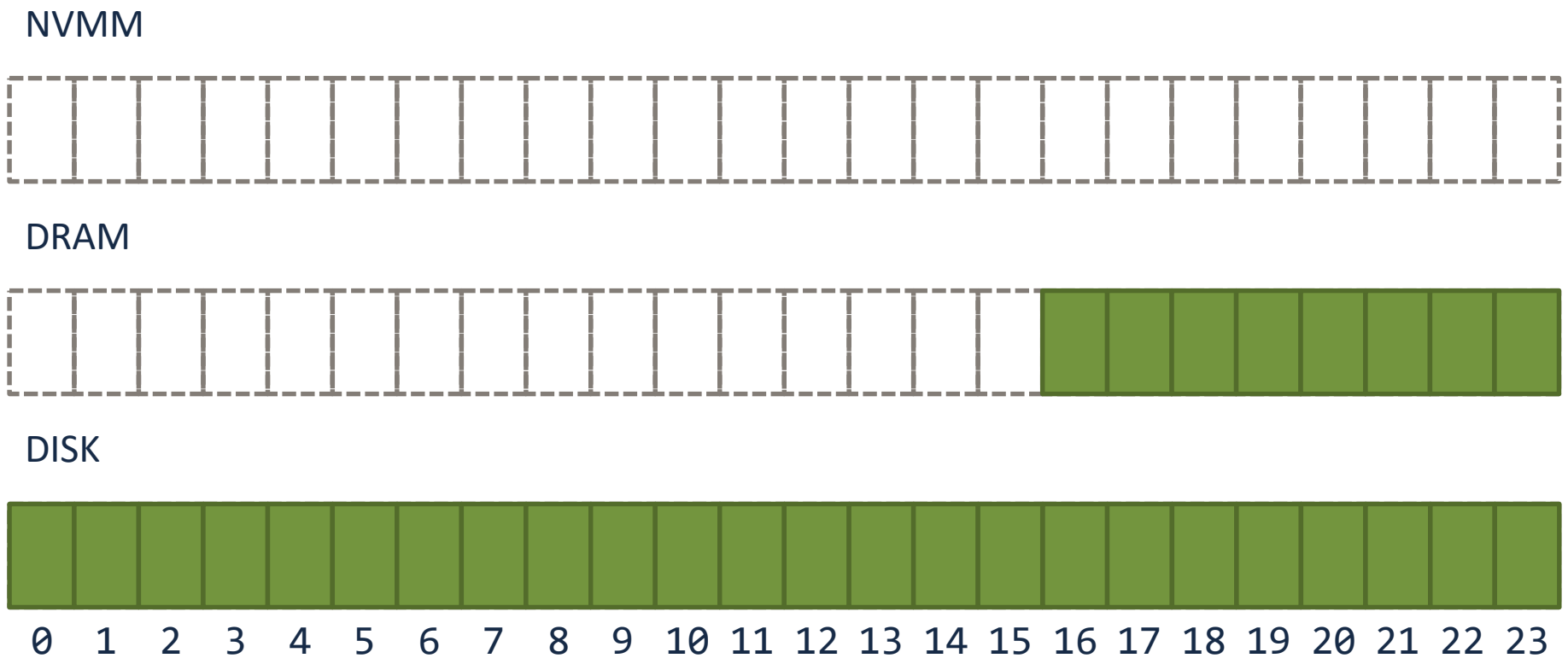
File Operations

```
write(0,8);
write(8,8);
fsync();
append(16,1);
append(17,1);
...
append(23,1);
......
migrate();
......
read(16,8);
mmap(0,8);
```



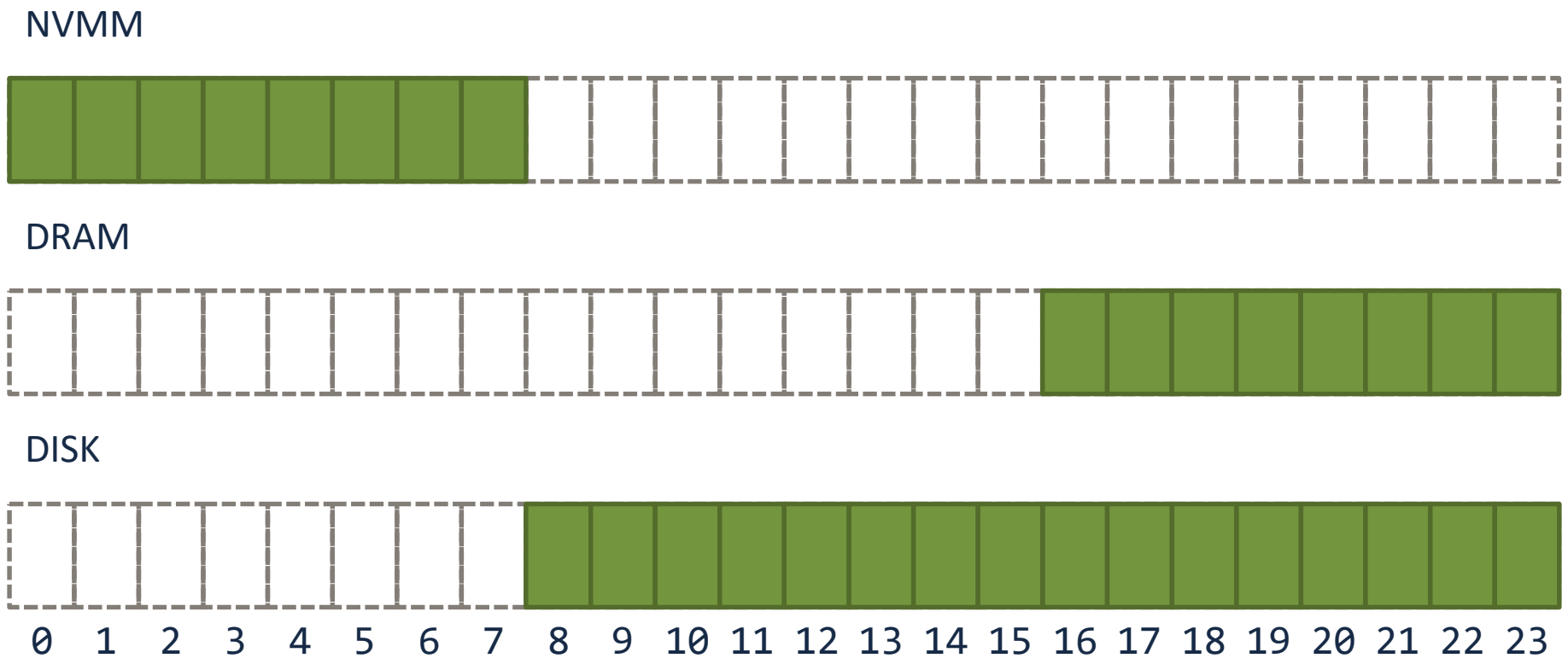
File Operations

```
write(0,8);
write(8,8);
fsync();
append(16,1);
append(17,1);
...
append(23,1);
......
migrate();
......
read(16,8);
mmap(0,8);
```



File Operations

```
write(0,8);
write(8,8);
fsync();
append(16,1);
append(17,1);
...
append(23,1);
......
migrate();
......
read(16,8);
mmap(0,8);
```

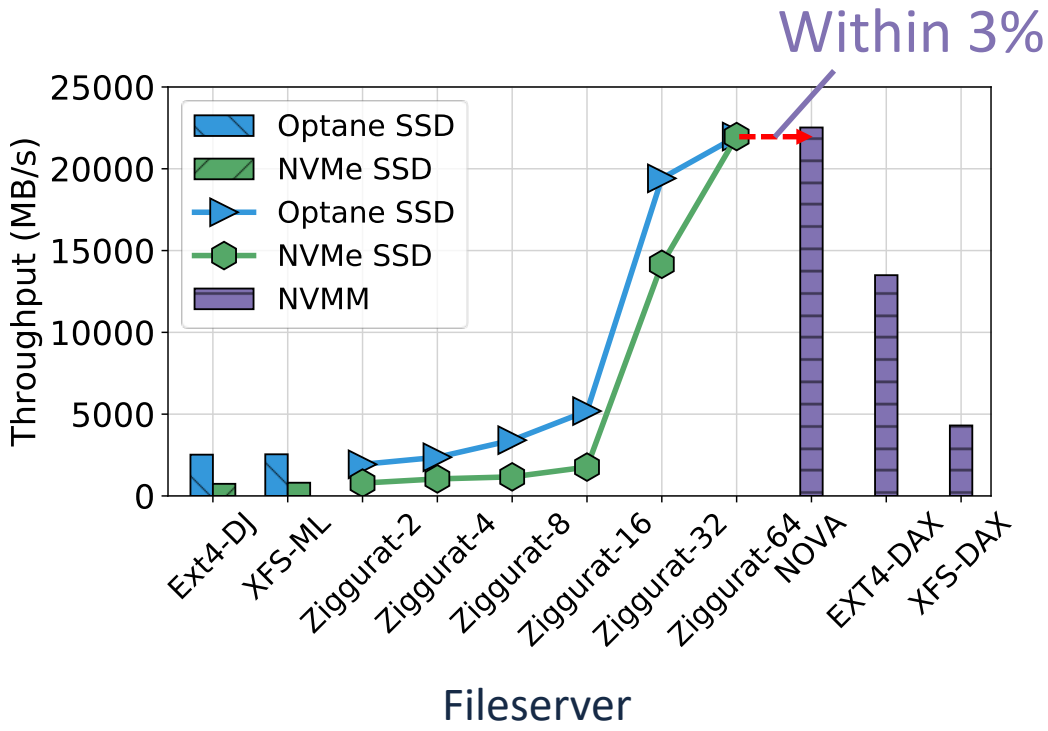


Evaluation

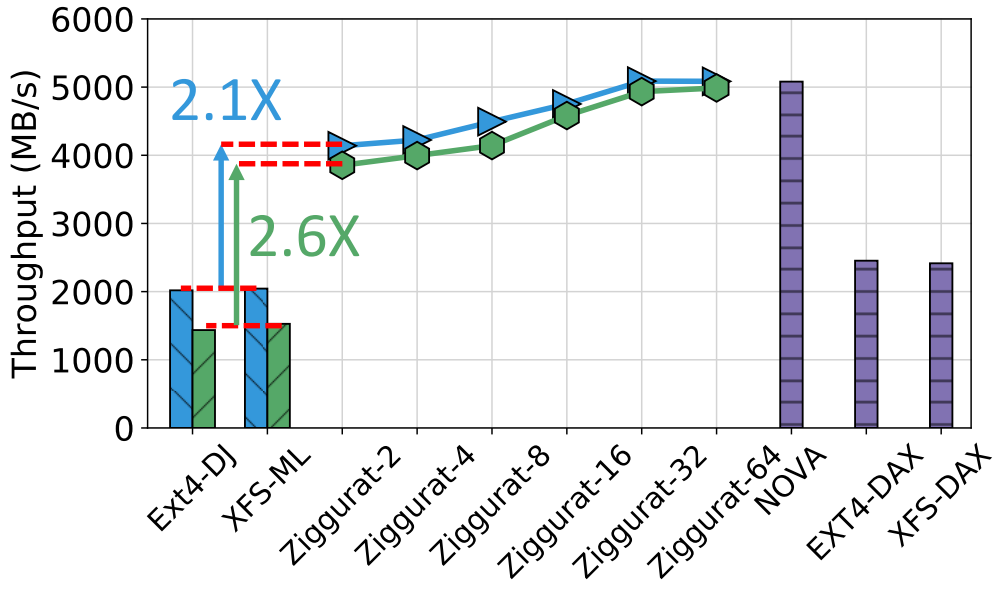
- Experimental setup
  - Dual-socket Intel Xeon E5 server
  - 256 GB DRAM
  - 187 GB Intel DC P4800X Optane SSD
  - 400 GB Intel DC P3600 NVMe SSD
  - Ubuntu 16.04 LTS, Linux kernel 4.13.0
- Emulated NVMM
  - NUMA effect on DRAM
  - NUMA node 1: NVMM emulation
  - NUMA node 0: Processors and memory for applications
- Disk-based file systems
  - EXT4-DJ (data journaling)
  - XFS-ML (metadata logging)
- Ziggurat
  - Ziggurat-X (Optane/NVMe SSD + X GB of NVMM)
- NVMM-based file systems
  - NOVA, EXT4-DAX, XFS-DAX

Filebench

- With large amount of NVMM, Ziggurat's performance nearly matches that of NVMM-only file systems.



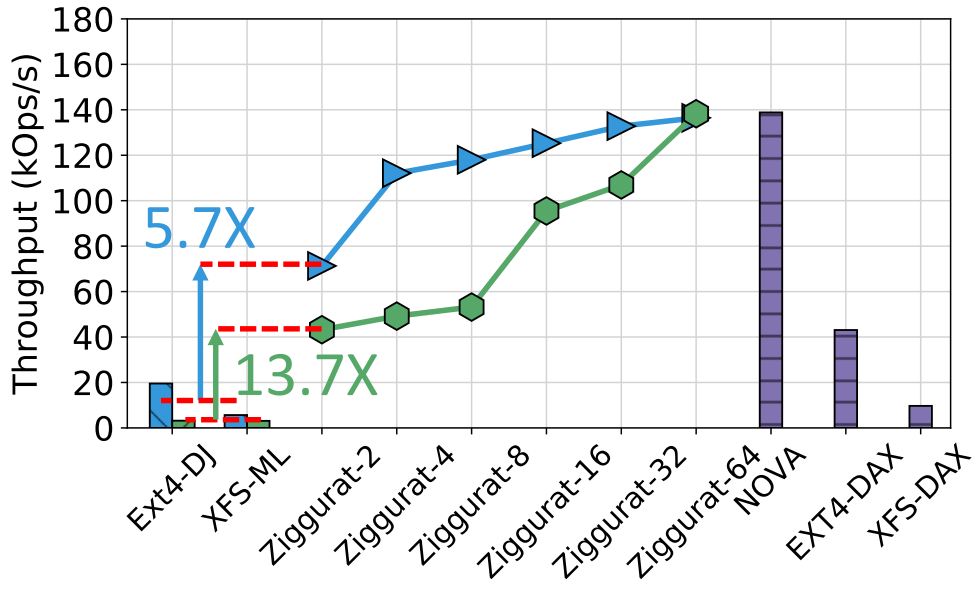
Fileserver



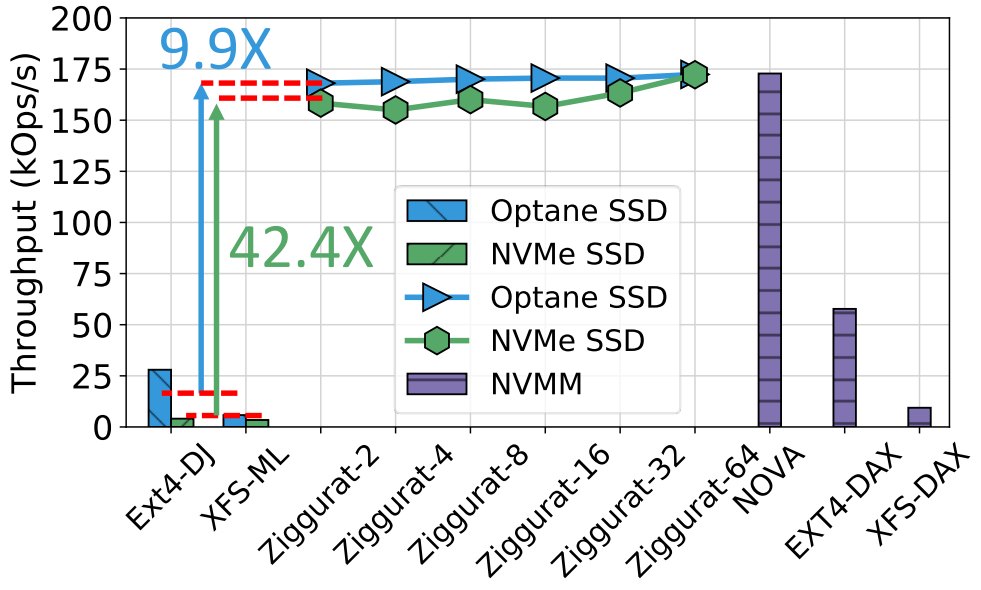
Varmail

Rocksdb

- Ziggurat shows good performance for inserting file data with write-ahead logging (WAL).



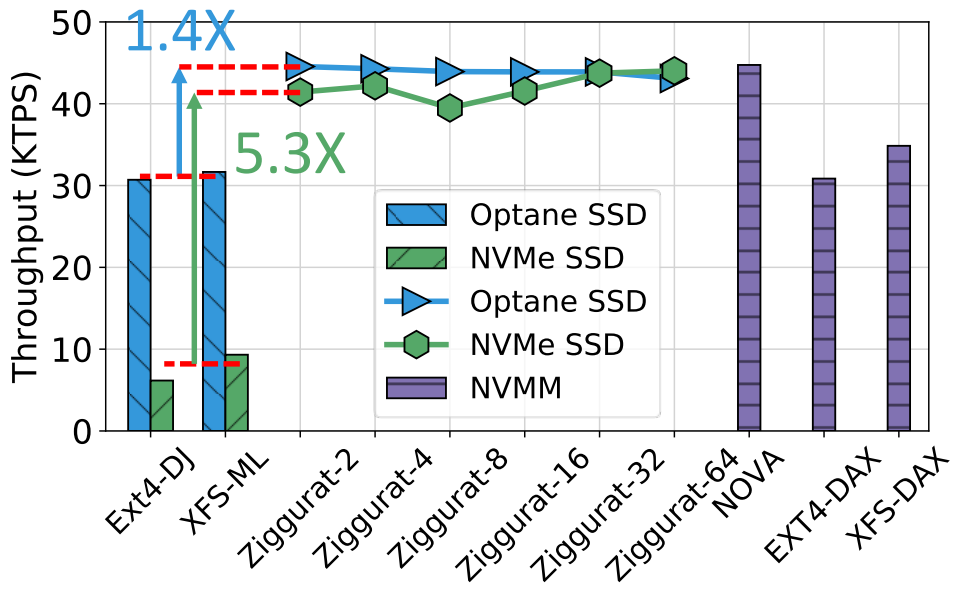
Random insert



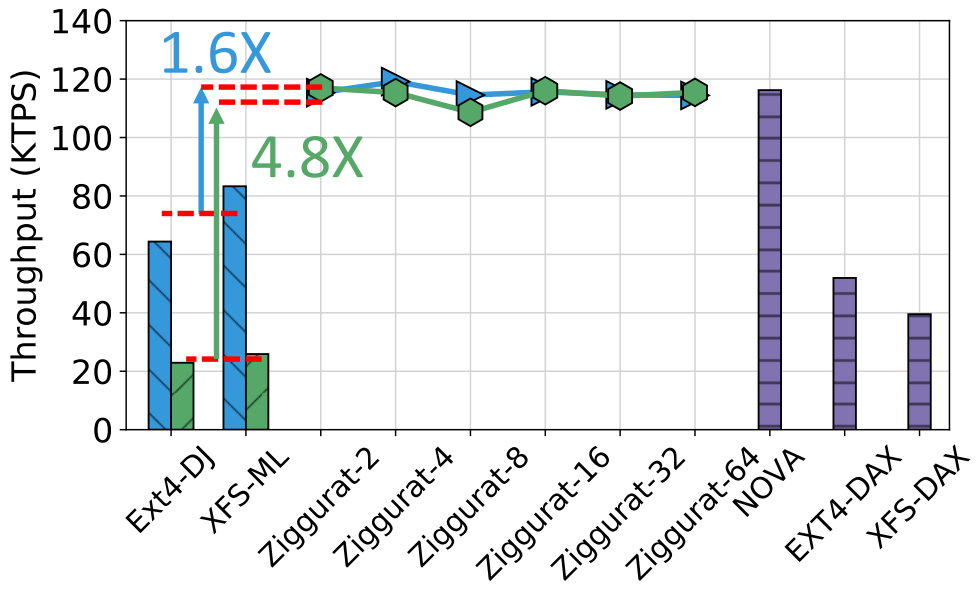
Sequential insert

SQLite

- Ziggurat maintains near-NVMM performance because the hot journal files are frequently updated in NVMM.



PERSIST



WAL

Conclusion

- Ziggurat fully utilizes the strengths of NVMM and disks to offer high file performance for a wide range of access patterns.
- [**Prediction**] Ziggurat steers the incoming writes to the most suitable tier based on the prediction results.
- [**Migration**] Ziggurat coalesces adjacent data blocks and migrates them in large chunks to disks.
- [**Evaluation**] Ziggurat achieves up to 38.9X and 46.5X throughput improvement compared with EXT4 and XFS running on an SSD alone, respectively.

Thank you