We Ain’t Afraid of No File Fragmentation: Causes and Prevention of Its Performance Impact on Modern Flash SSDs

*Jun, Yuhun, Shinhyun Park, Jeong-Uk Kang, Sang-Hoon Kim,
and Euiseong Seo.*

USENIX FAST’24

2024. 08. 14

Presented by Juhyun Kim & Yongmin Lee

jhk@kiost.ac.kr , nascarf16@dankook.ac.kr

DANKOOK UNIVERSITY

Dankook University
System Software Lab.

Contents

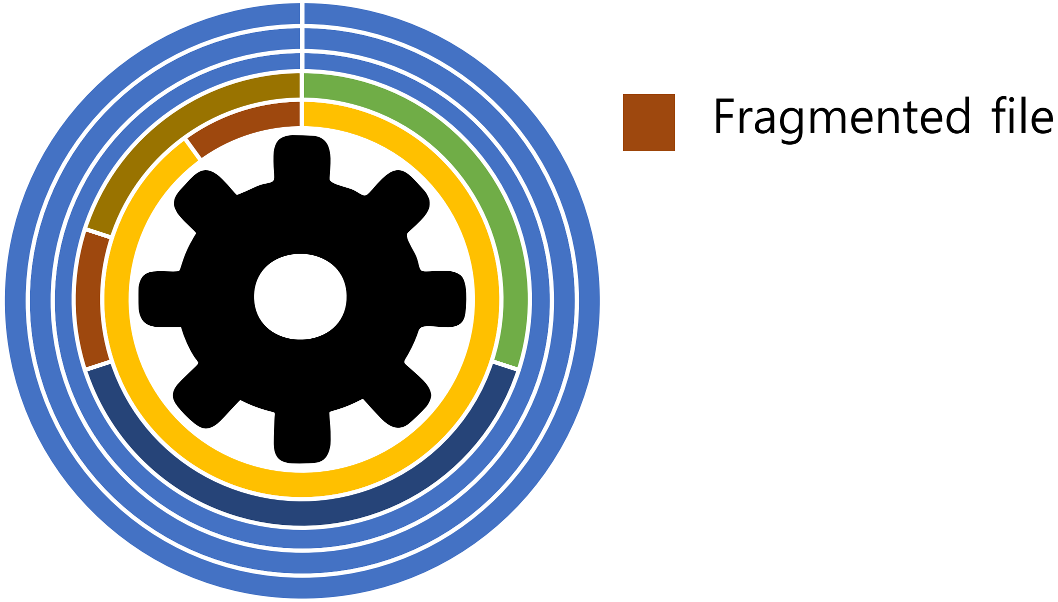
1. Fragmentation in HDD
2. Fragmentation in SSD
3. Normal SSD
4. Analysis of File Fragmentation
5. Die-level collisions
6. Approach
7. Evaluation
8. Conclusion

Fragmentation in HDD

- **Discontinuous data blocks**
  - Random access to scattered fragment

    → **Read performance bad !!!**
- **Existing tool**
  - Delay, pre allocation etc …
  - But simultaneously multiple write or long time before additional file write

    → **Impossible avoid to fragmentation**
- **Main degradation**
  - Kernel I/O path, storage device interface, storage media access



< HDD >

Fragmentation in SSD

- **File Systems Fated for Senescence? Nonsense, Says Science!**

  Alex Conway, et al. FAST'17

  → SSD have 2 to 5 times slower read performance when accessing fragmented files

Fragmentation in SSD

- **File Systems Fated for Senescence? Nonsense, Says Science!**

  Alex Conway, et al. FAST'17

  → SSD have 2 to 5 times slower read performance when accessing fragmented files

- **FragPicker: A New Defragmentation Tool for Modern Storage Devices**

  Park, Jonggyu, and Young Ik Eom. ACM SIGOPS'21

  → Claims that SSD's performance degradation is mainly from request splitting

Fragmentation in SSD

- **File Systems Fated for Senescence? Nonsense, Says Science!**
  Alex Conway, et al. FAST'17
  → SSD have 2 to 5 times slower read performance when accessing fragmented files
- **FragPicker: A New Defragmentation Tool for Modern Storage Devices**
  Park, Jonggyu, and Young Ik Eom. ACM SIGOPS'21
  → Claims that SSD's performance degradation is mainly from request splitting

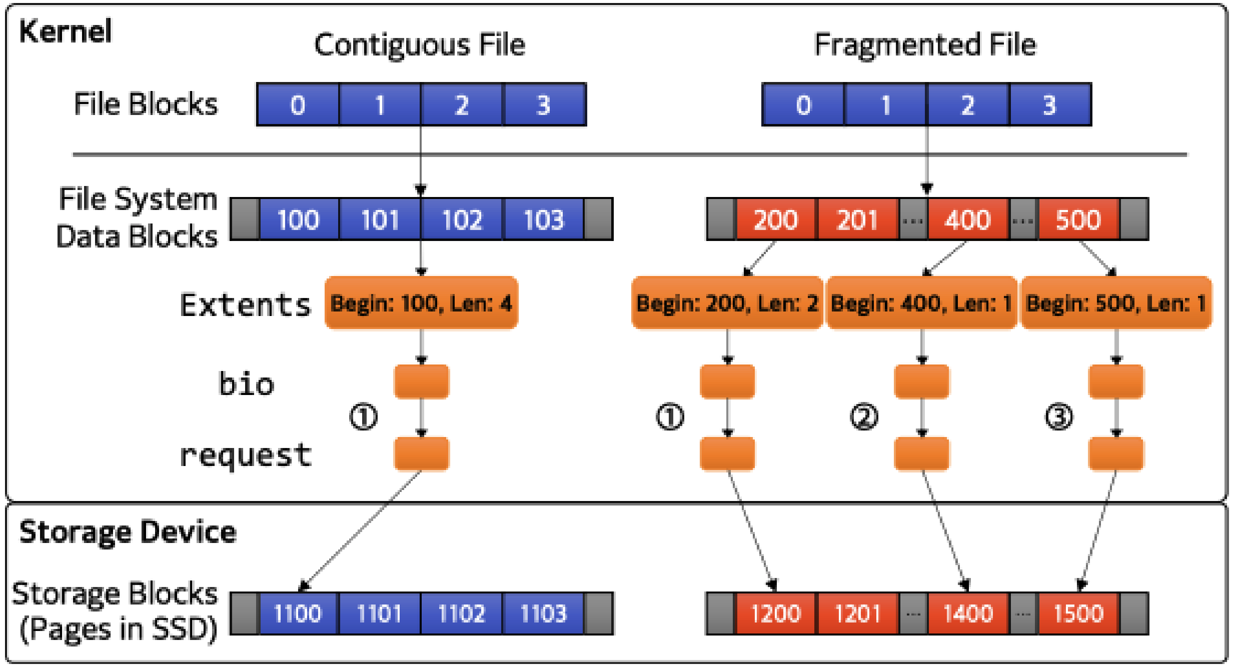
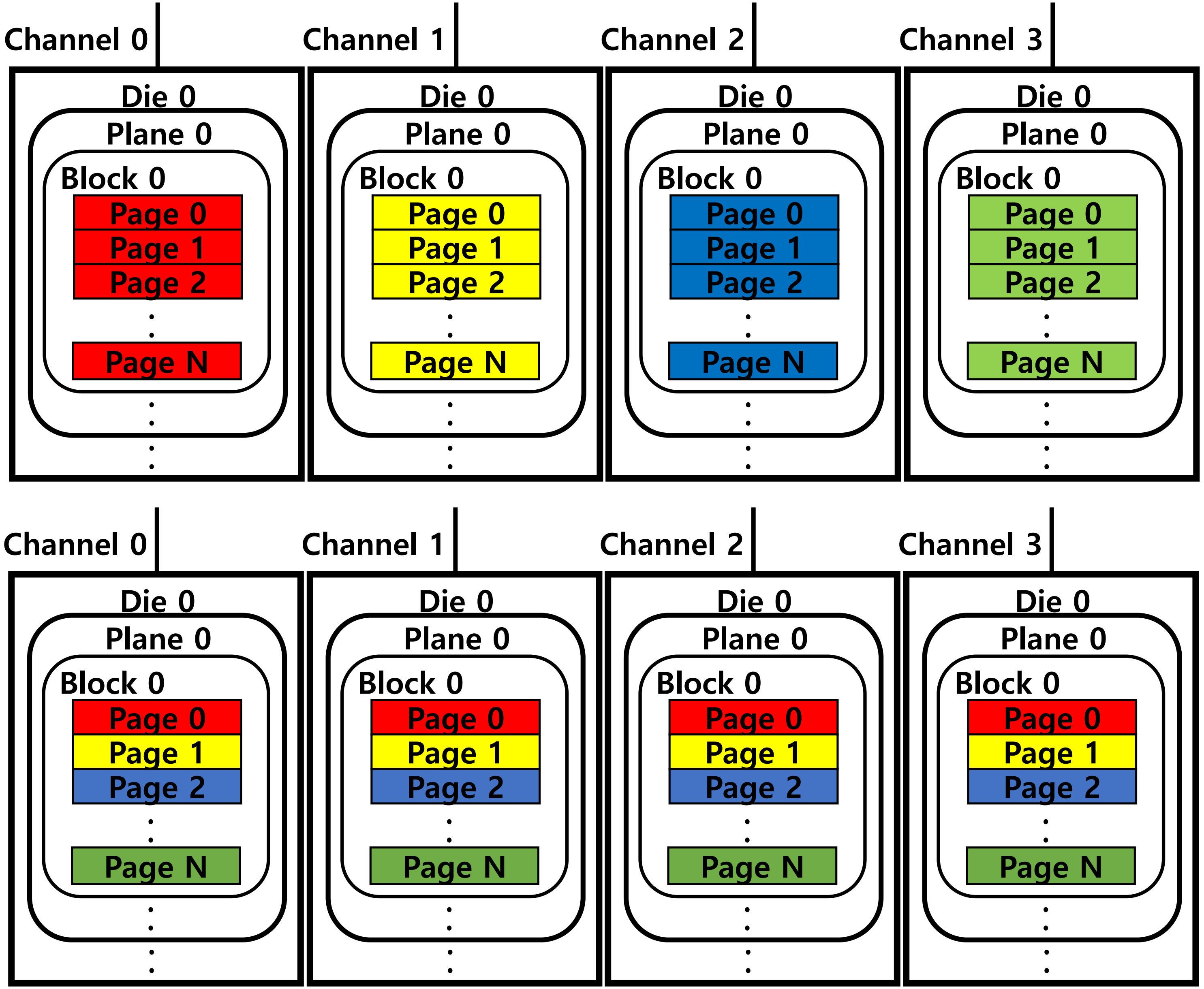


Figure 1: A sequential access to a contiguous file is translated to a single device command while that to a fragmented file ends up with multiple requests.

Single I/O operations translated into multiple device commands

Normal SSD

- Parallelism process
  - Using die
  - Assign in round-robin



Channel 0 | Channel 1 | Channel 2 | Channel 3

Die 0
Plane 0
Block 0
Page 0
Page 1
Page 2
Page N

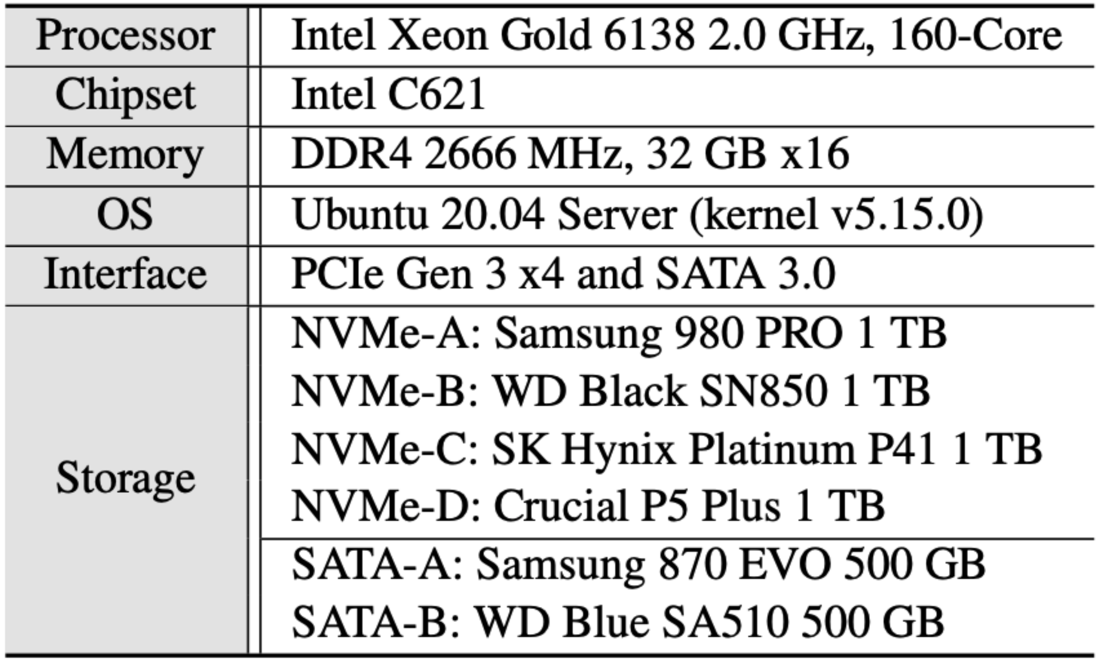
Channel 0 | Channel 1 | Channel 2 | Channel 3

Die 0
Plane 0
Block 0
Page 0
Page 1
Page 2
Page N

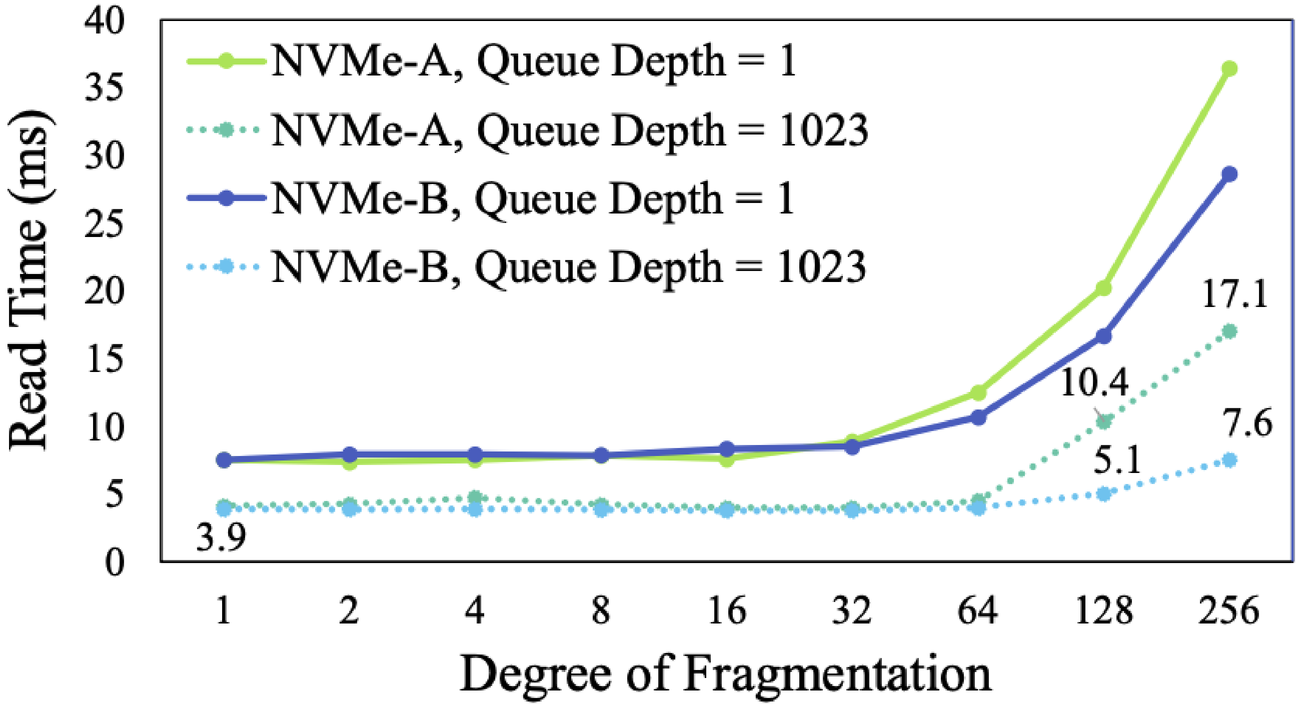
Analysis of File Fragmentation

- **Performance degradation by Degree of Fragmentation (DOF)**

Table 1: System configurations for experiments.



Processor	Intel Xeon Gold 6138 2.0 GHz, 160-Core
Chipset	Intel C621
Memory	DDR4 2666 MHz, 32 GB x16
OS	Ubuntu 20.04 Server (kernel v5.15.0)
Interface	PCIe Gen 3 x4 and SATA 3.0
Storage	NVMe-A: Samsung 980 PRO 1 TB NVMe-B: WD Black SN850 1 TB NVMe-C: SK Hynix Platinum P41 1 TB NVMe-D: Crucial P5 Plus 1 TB
	SATA-A: Samsung 870 EVO 500 GB SATA-B: WD Blue SA510 500 GB

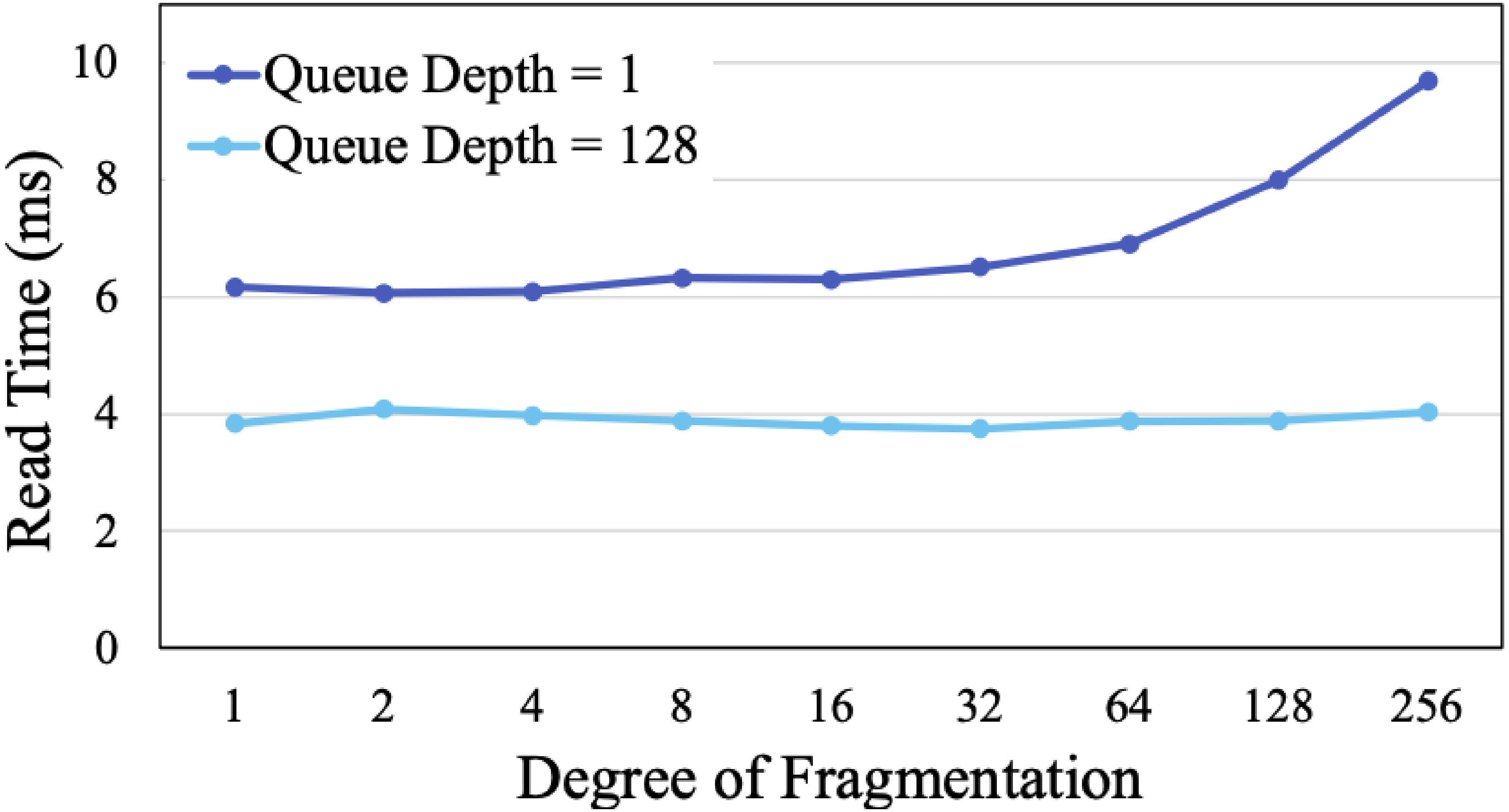


→ DOF causes performance degradation

Analysis of File Fragmentation

- **Impact Caused by Request Splitting**
  - **Do not affect to read time in RAMdisk**

    → No impact from request splitting



Analysis of File Fragmentation

- **Impact Caused by Request Splitting**

  → Request time increased proportionally With the increase in the DOF

  → Request can be masked by queue depth

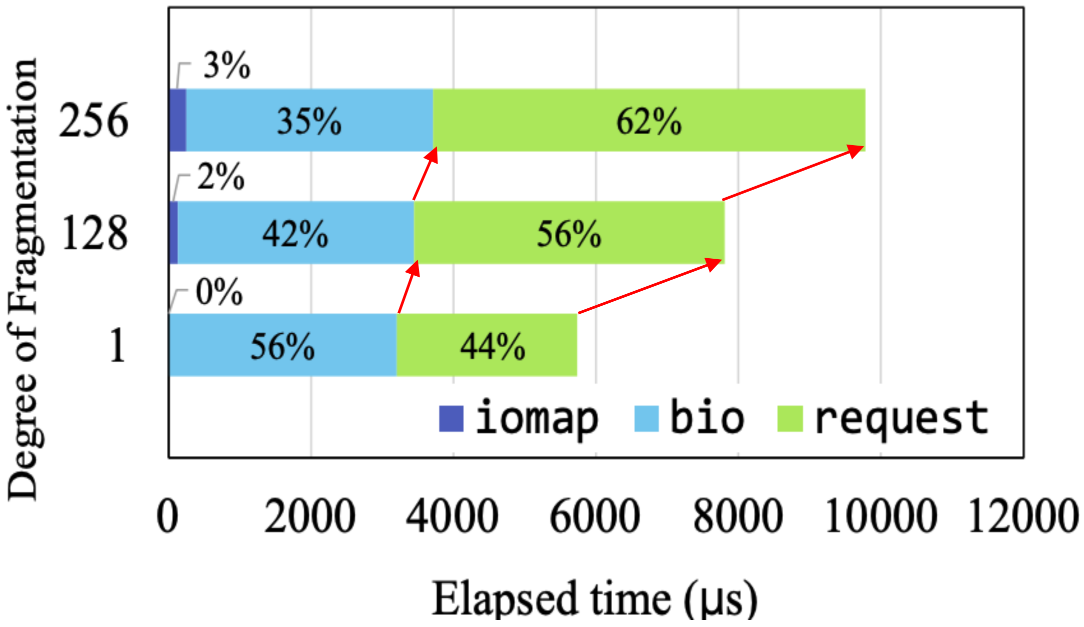


Figure 5: Time composition for creating request data structures in the kernel I/O path depending on File's DoF.

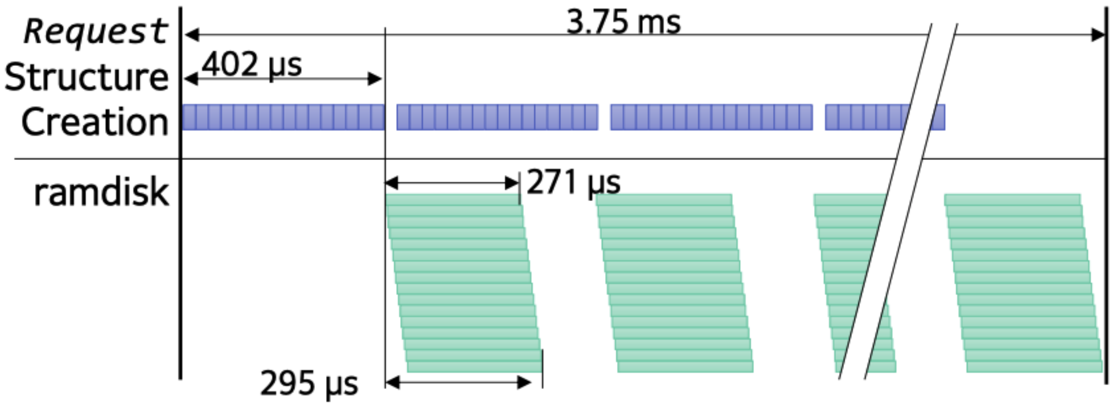
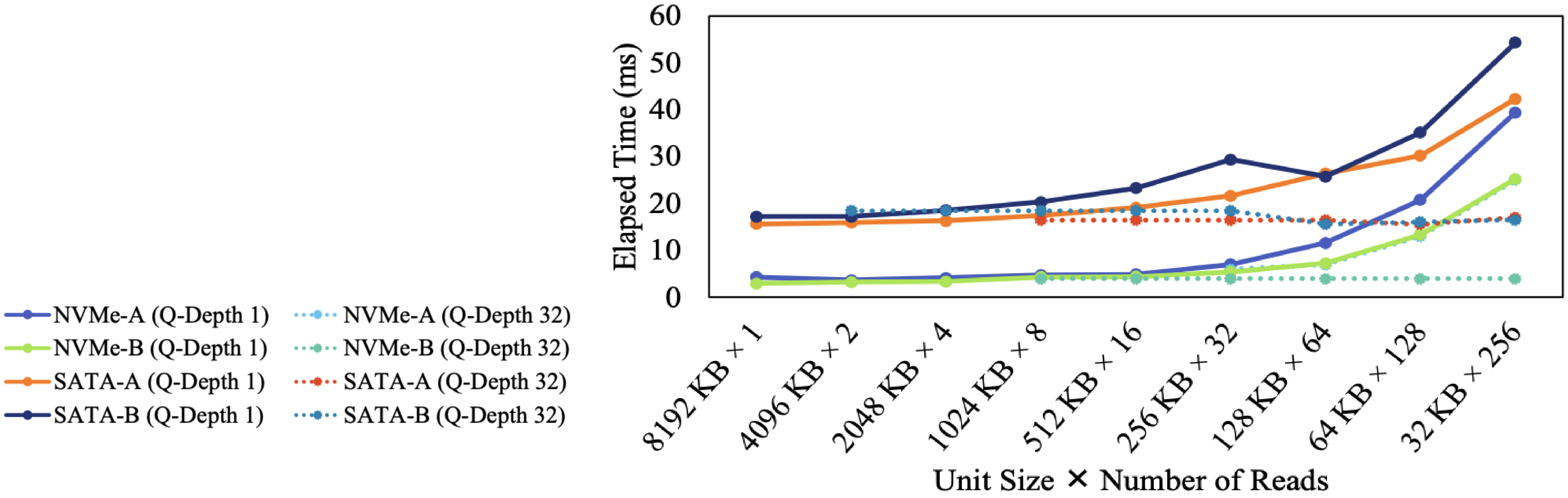


Figure 6: Reduction of read time due to the overlap of storage operations and `request` creation when File's DoF is 128.

Analysis of File Fragmentation

- **Impact Caused by Request Splitting**

  → Elapsed time increase

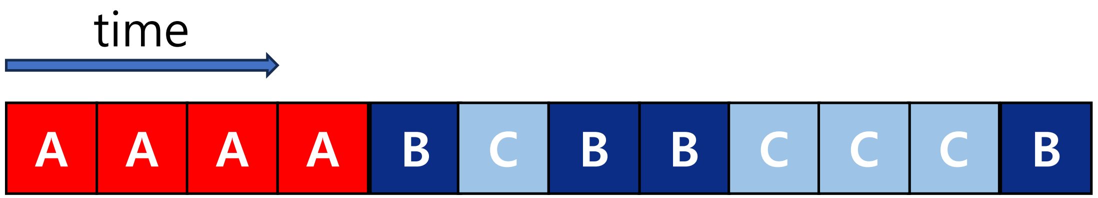


< Elapsed time for reading 8 MB of data depending on the unit size and the number of reads>

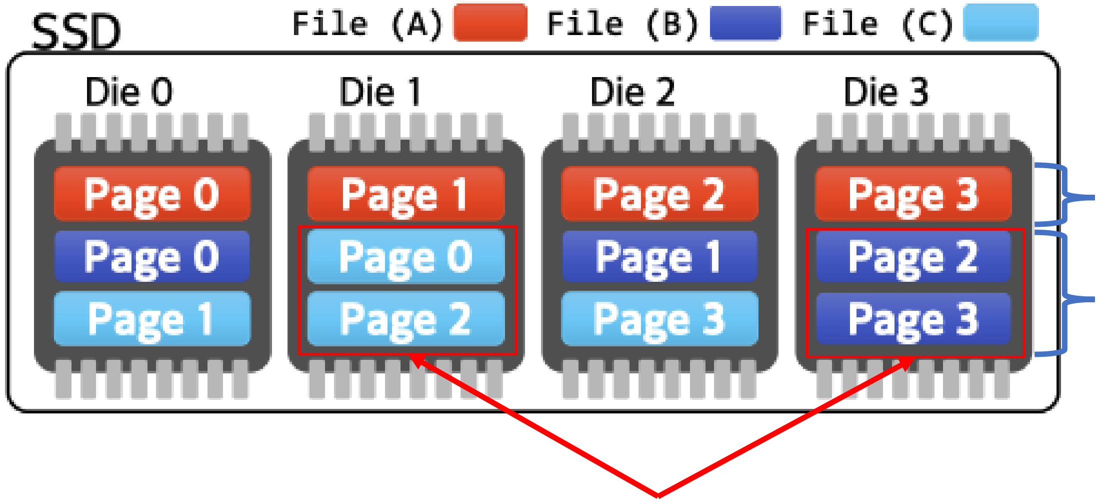
Analysis of File Fragmentation

- **Page misalignment**

Write sequence:



Write in round-robin manner



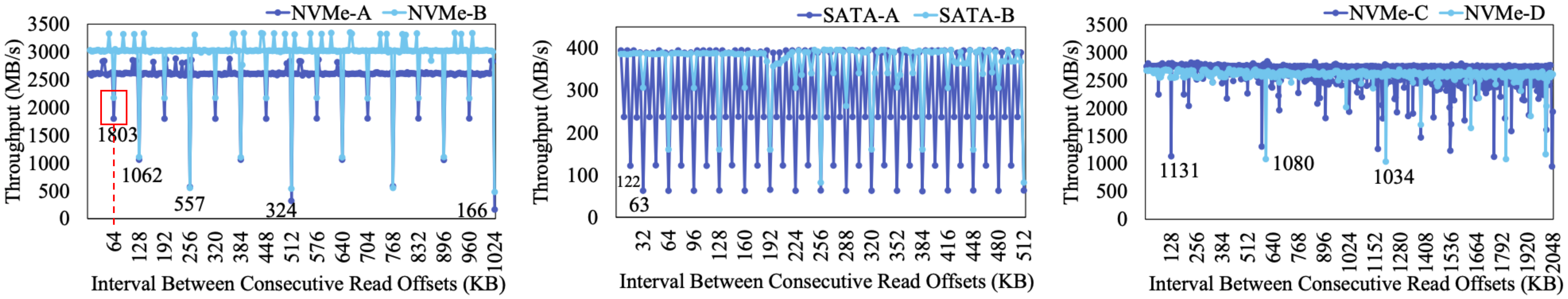
File A: up to 4x more bandwidth than single die

File B and C: 2x slower than File A

Die-level collision

Analysis of File Fragmentation

- **Page misalignment**



(a) NVMe-A/B (b) SATA (c) NVMe-C/D

Figure 8: Throughput while varying the interval between starting points of consecutive read operations.

→ In SSD, file fragmentation leads to additional die-level collisions

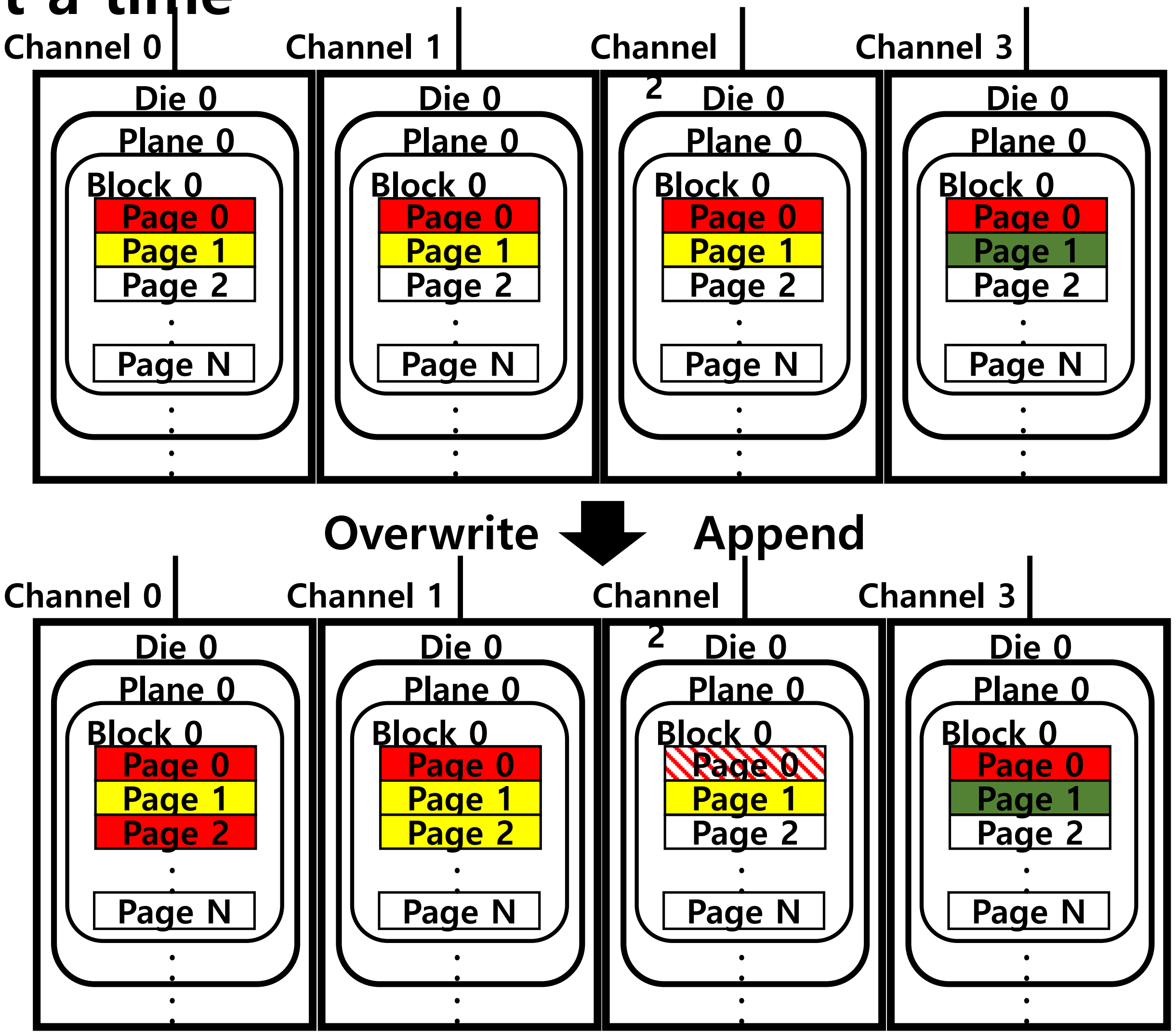
Die-level collisions

- Die can only process one request at a time
  - Overwrite & Append

    Page Page

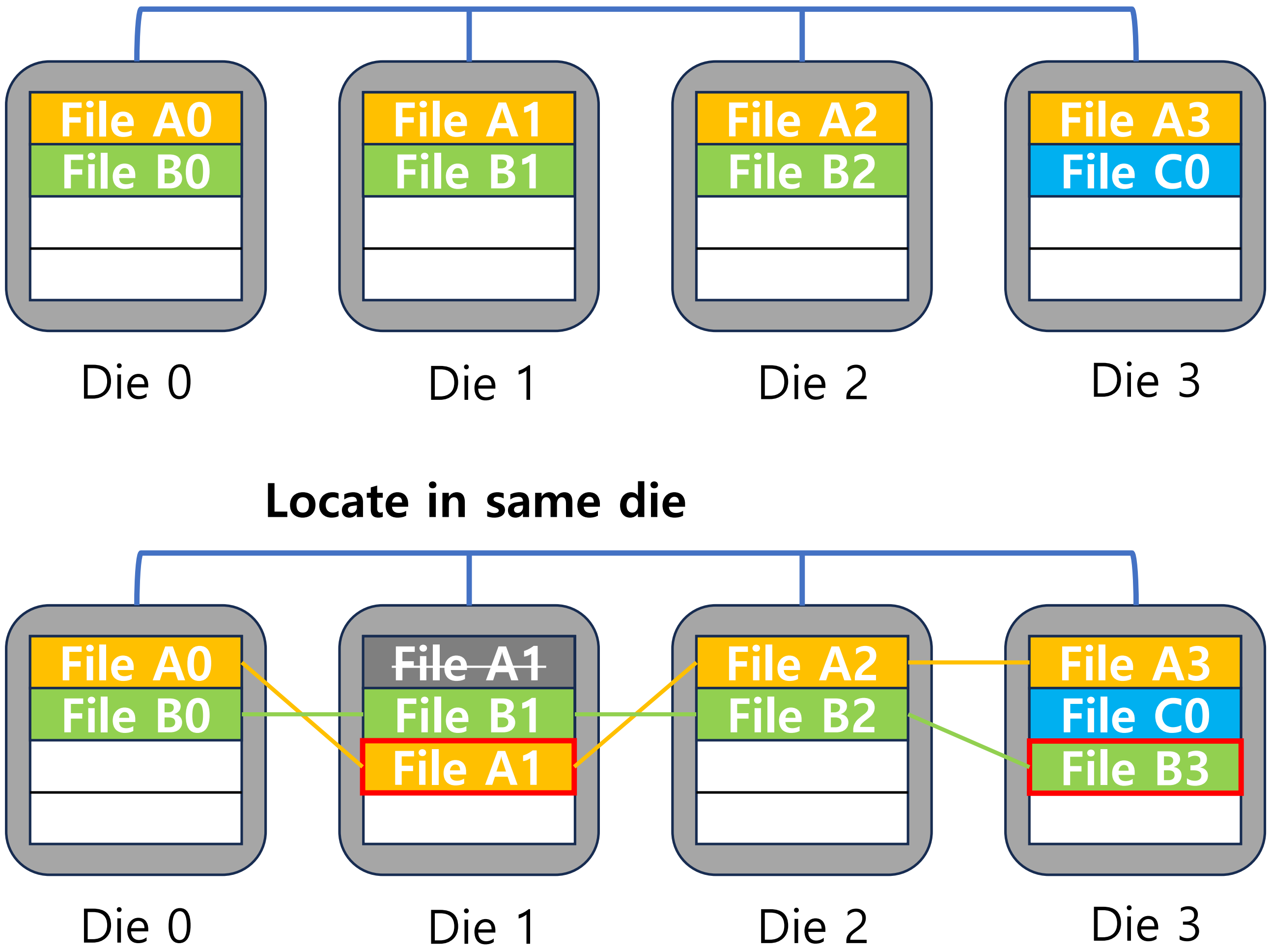
    → Occur die level collisions

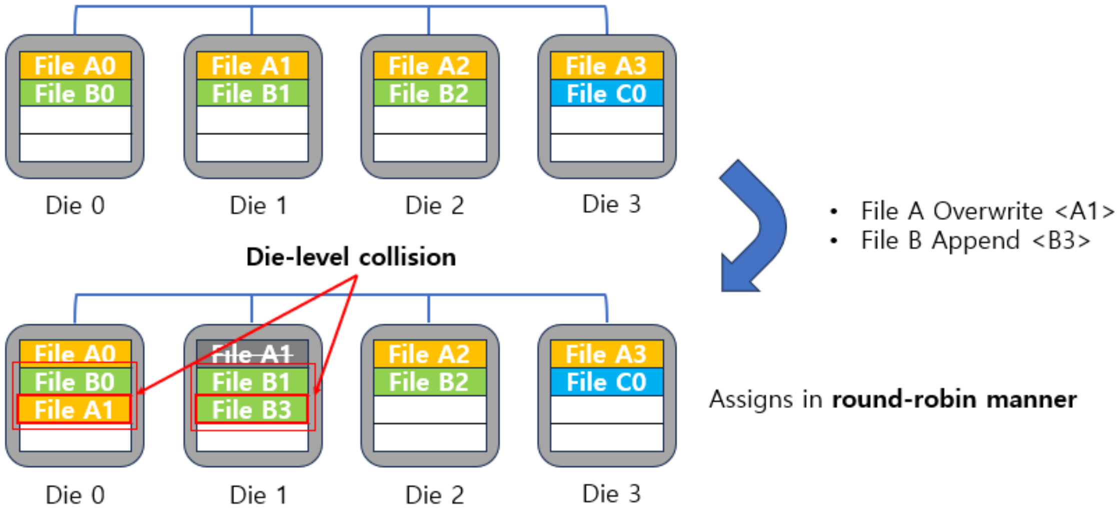
    → Read performance degradation



Approach

- **Using the given approach**





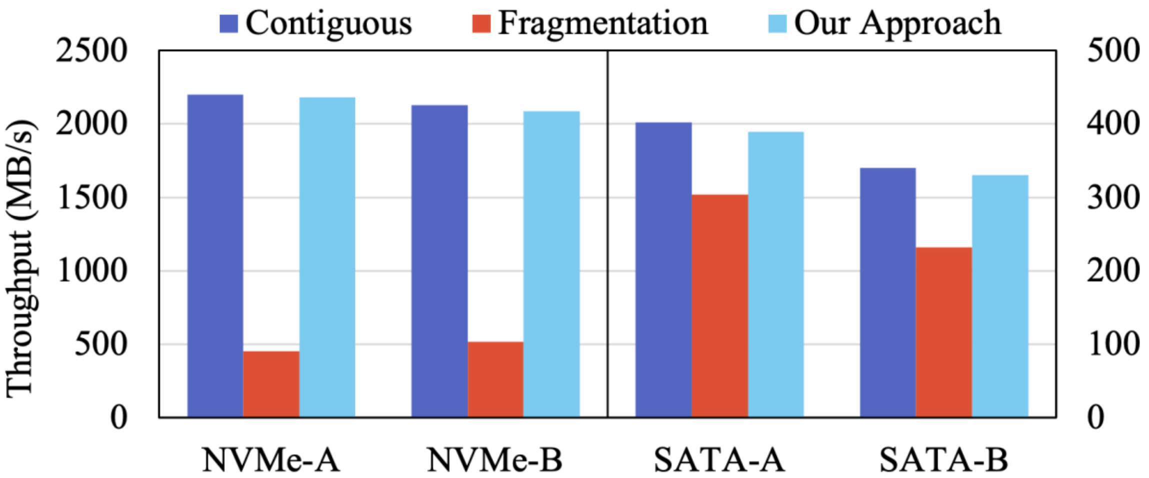
- File A Overwrite <A1>
- File B Append <B3>

Assigns in **round-robin manner**

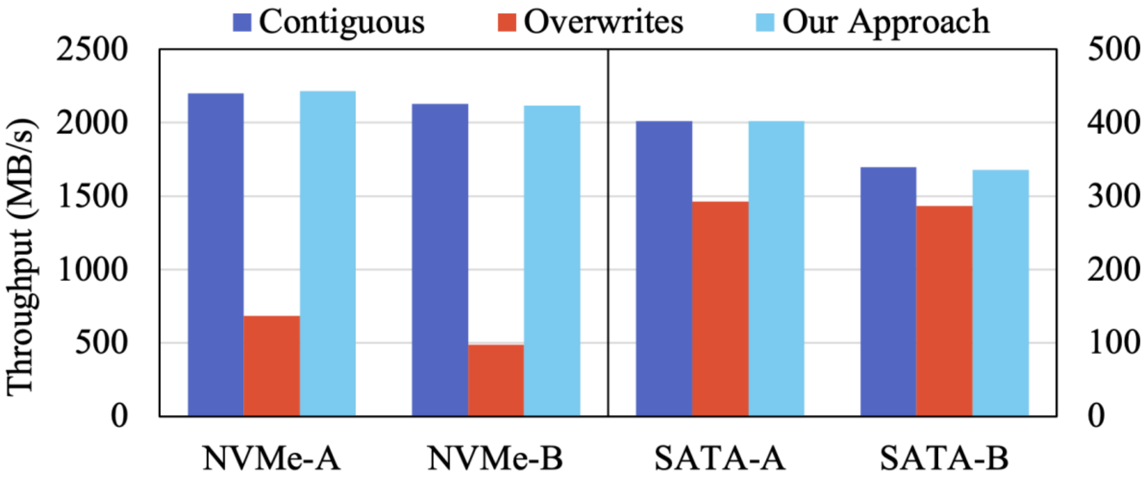
File A Overwrite <A1>
File B Append <B3>

Evaluation

- **Modified write patterns & Showing read throughput**
  - Form a file by append 256 segments
  - Each segement size
    → SSD's die allocation granularity
  - Total file size = 8MB



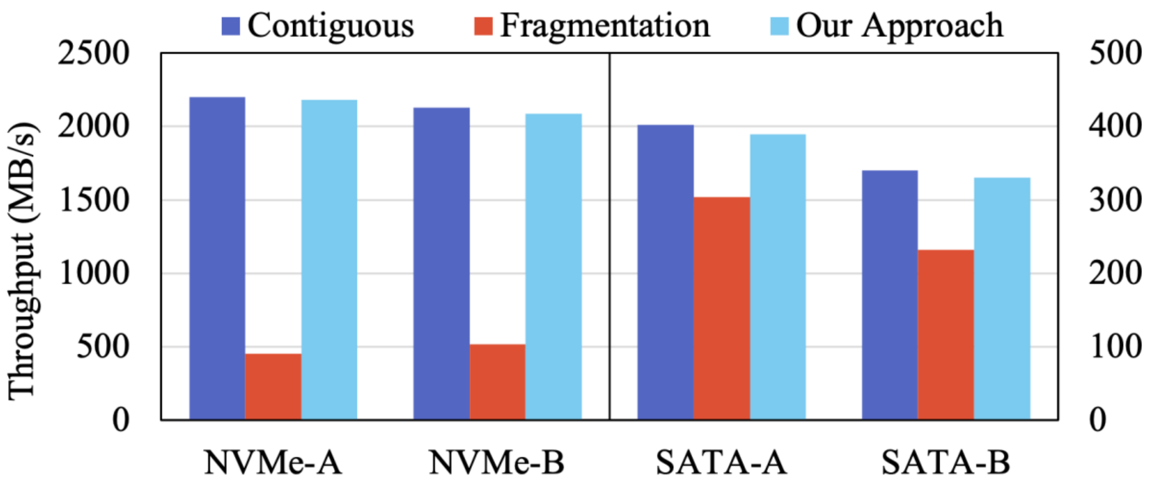
(a) Append Write



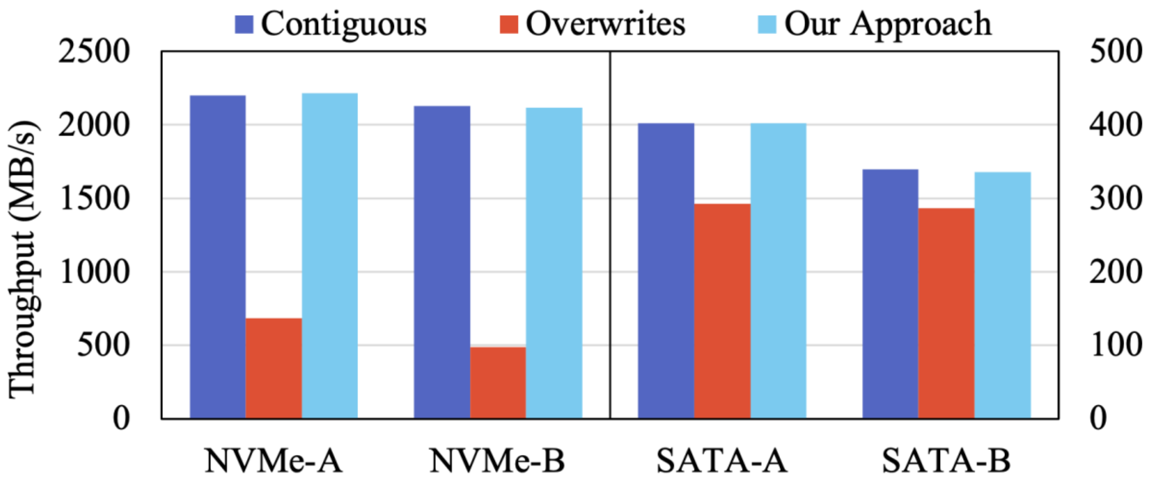
(b) Overwrite

Evaluation

- **Why does SATA SSDs performance degradation is less severe than NVMe?**
  - SATA3 Maximum throughput = **600MB/s**
  - Smaller die allocation granularities in SATA SSD
  - Adjusted final append's size to fit 8MB
  - So only the initial segment of the file became fragmenated in SATA SSD



(a) Append Write

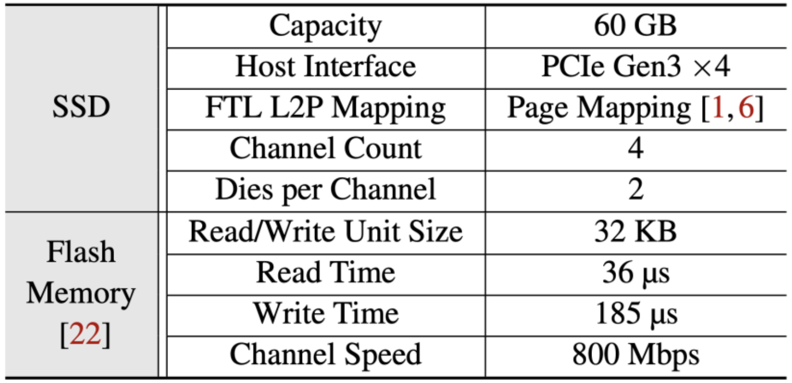


(b) Overwrite

Evaluation

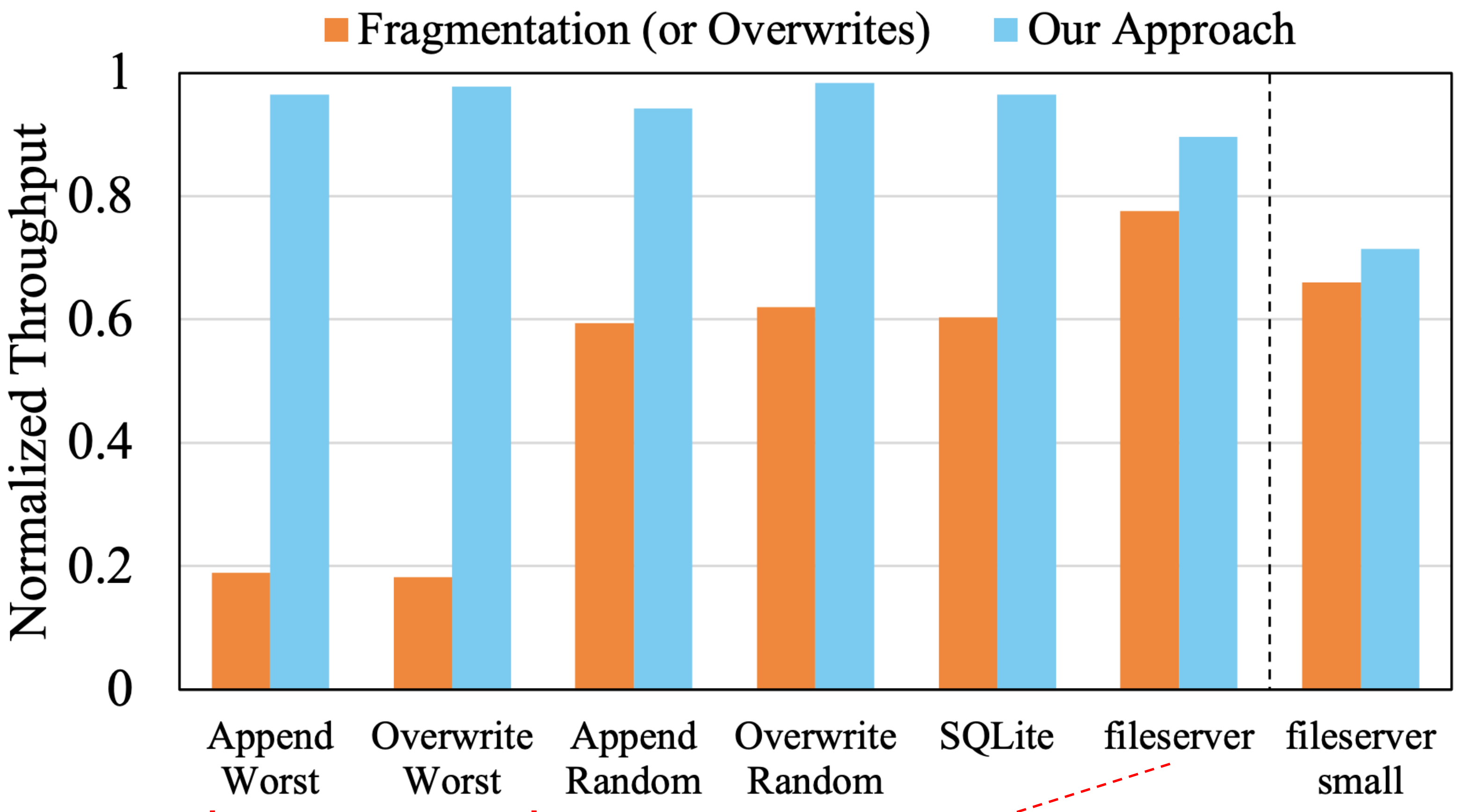
- **NVMeVirt**

Table 2: Parameters used for NVMe emulation.



SSD	Capacity	60 GB
	Host Interface	PCIe Gen3 ×4
	FTL L2P Mapping	Page Mapping [1, 6]
	Channel Count	4
	Dies per Channel	2
Flash Memory [22]	Read/Write Unit Size	32 KB
	Read Time	36 µs
	Write Time	185 µs
	Channel Speed	800 Mbps

Mirrors the settings of NVMe-B



10 threads, 32KB size append writes

Worst case: located in single die

Reduced to 16KB

Conclusion

- File fragmentation can indeed declines in read performance in SSD
  - Because of die-level collisions rather than request splitting
  - Misalignments also happens when files are overwritten

- Proposed NVMe command extension for better die-level parallelism
  - Provide hints to SSD

  → prevent additional die-level collisions caused by both file fragmentation and overwrites
  - Effectively suppresses the read performance degradation

Thank You !