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

(email) jhk@kiost.ac.kr , nascarf16@dankook.ac.kr

DANKOOK UNIVERSITY

Dankook University System Software Lab.

Contents

Fragmentation in HDD

- **Discontinue data blocks**

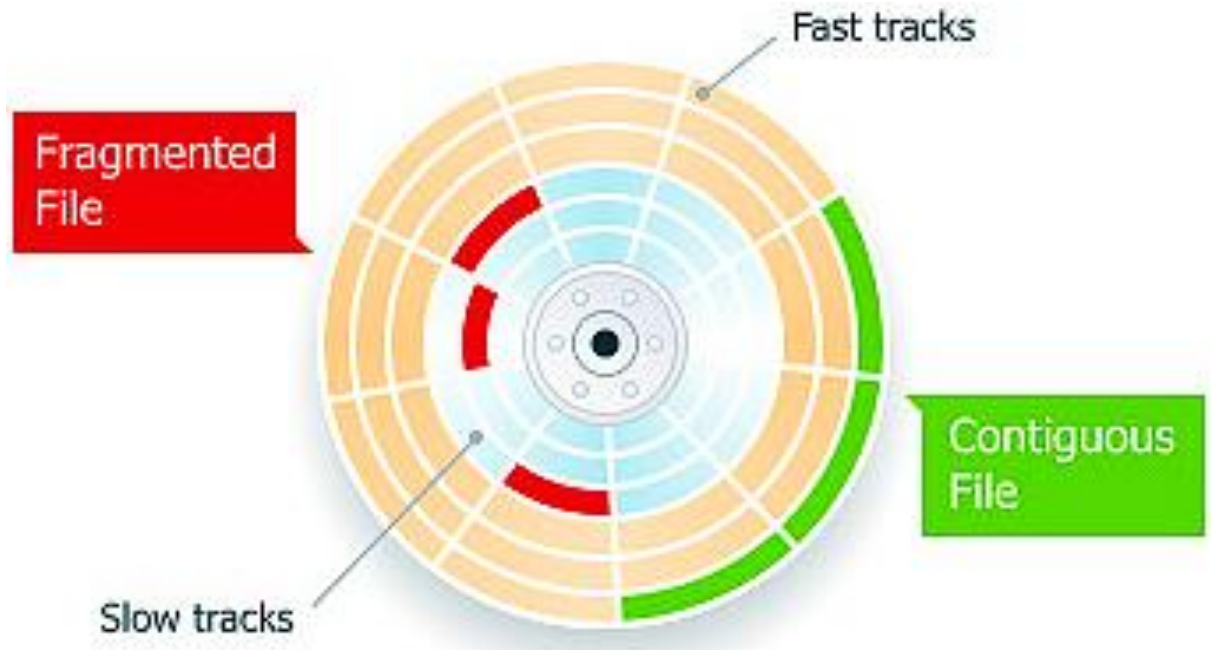
→ Random access to scattered fragment

→ Read performance bad !!!

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

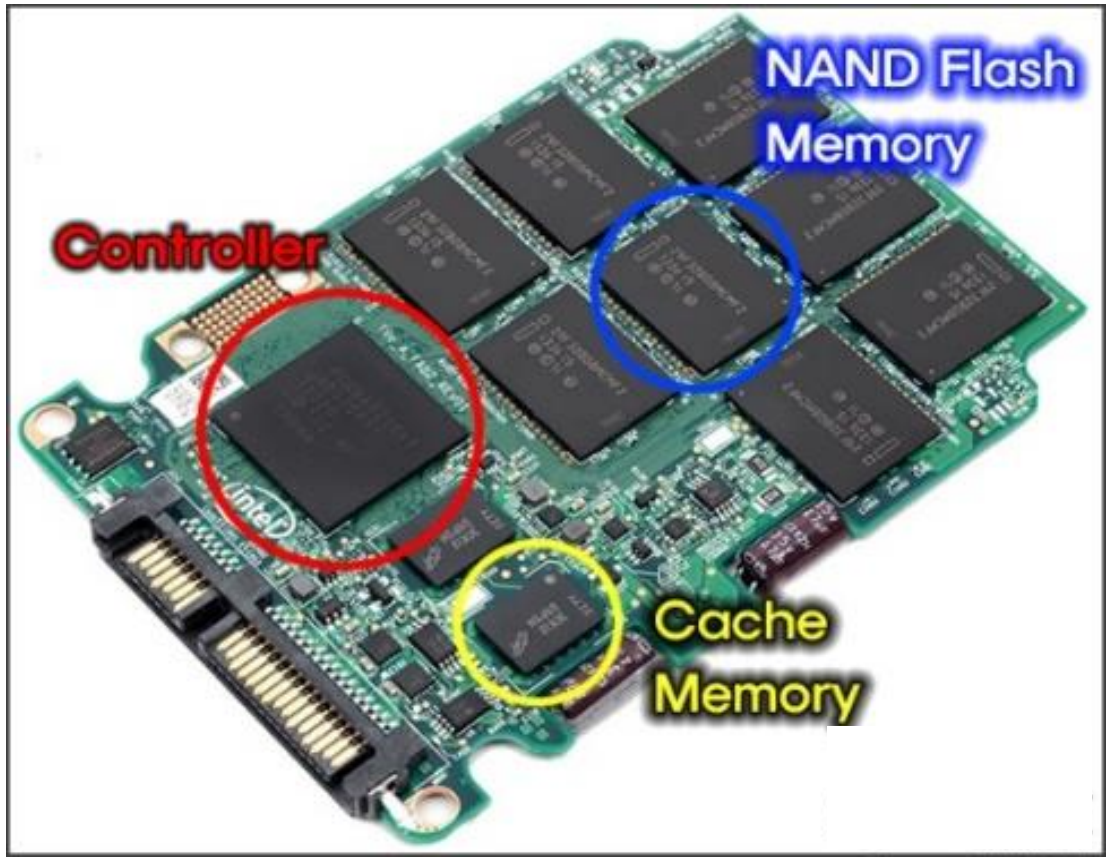
  → Impossible avoid to fragmentation

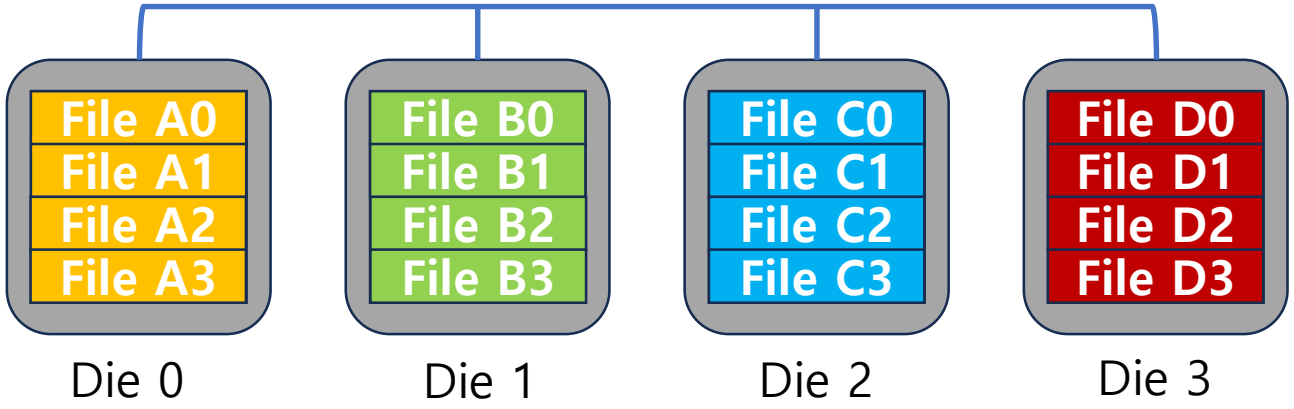
- **Main reason**
  - Kernel I/O path, storage device interface, storage media access



Normal SSD

- SSD

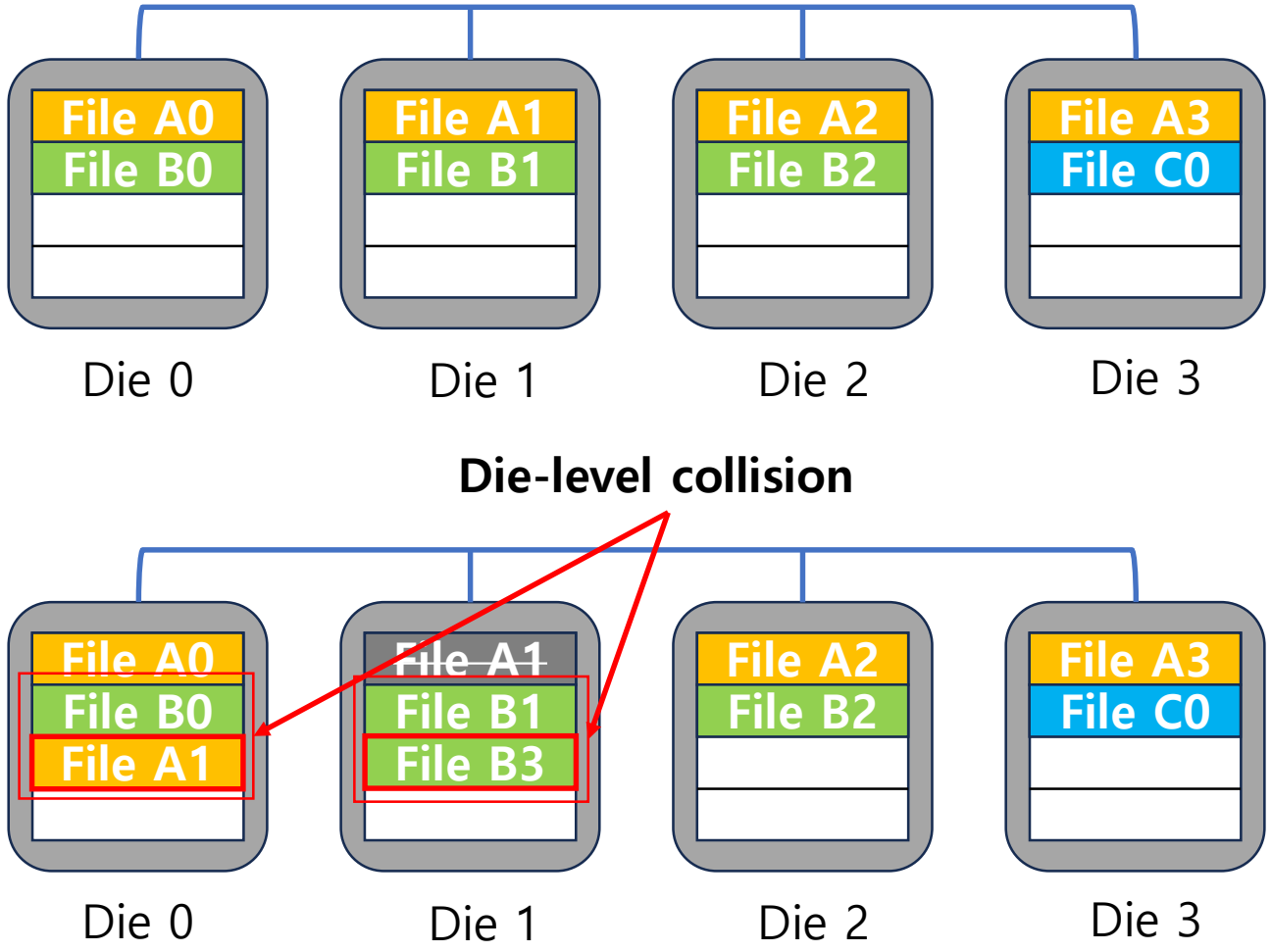




Is performance always good?

Normal SSD

- **Issue**



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

Assigns in **round-robin manner**

Fragmentation in SSD

- **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

- **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

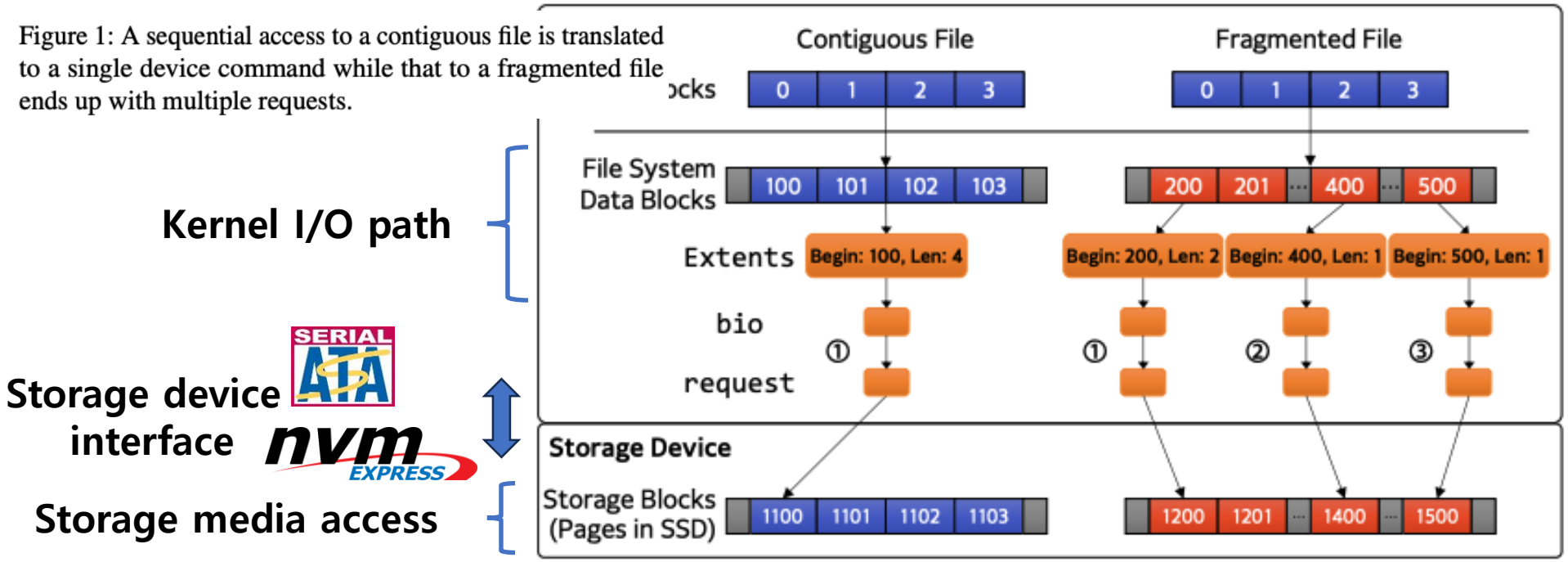
- **SSD**
  - **File Systems Fated for Senescence? Nonsense, Says Science!** *Alex Conway, et al. FAST'17*

  → 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.



Die-Level

- **Reason why? Die Level Collisions!**

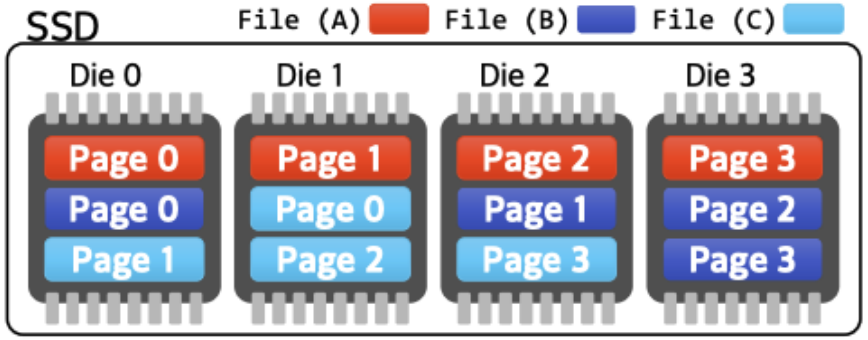
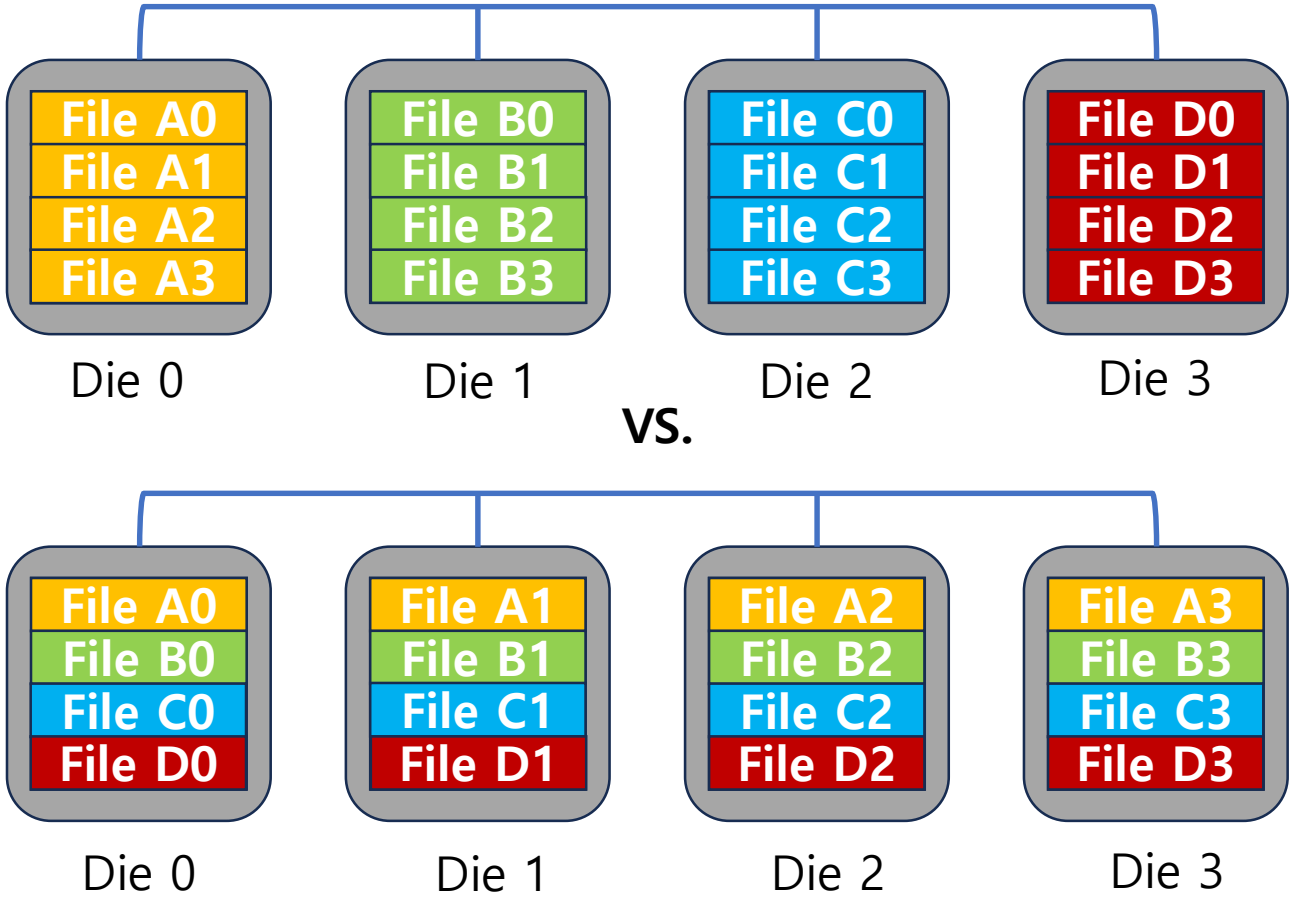


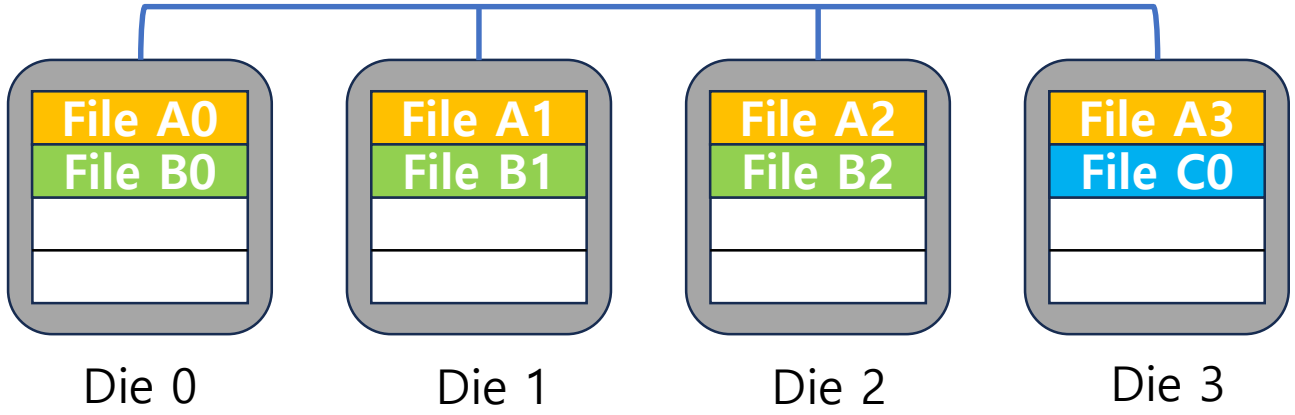
Figure 2: Data placement of three files in a flash SSD where one is contiguous and the other two are fragmented.

→ Die can only process one request at a time



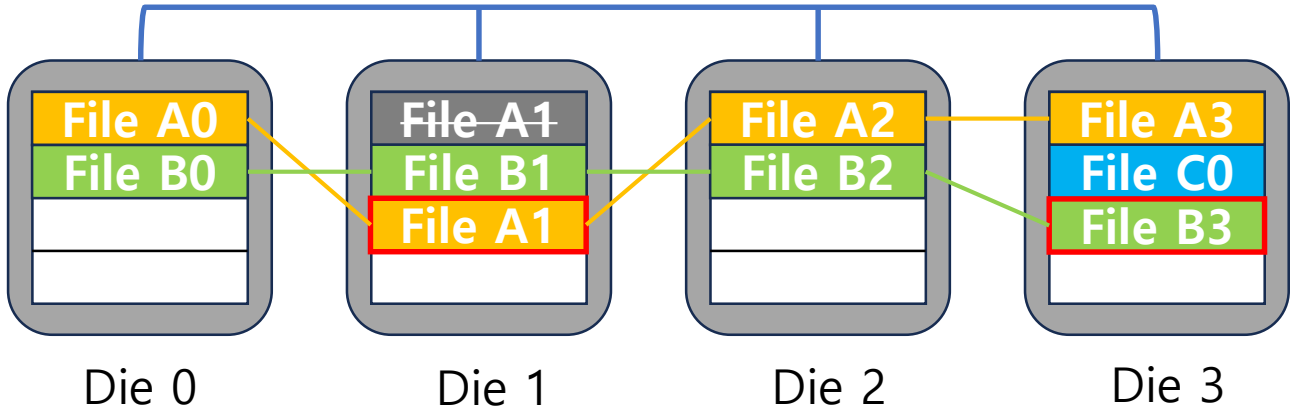
Approach

- **Using the given approach**



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

Locate in same die



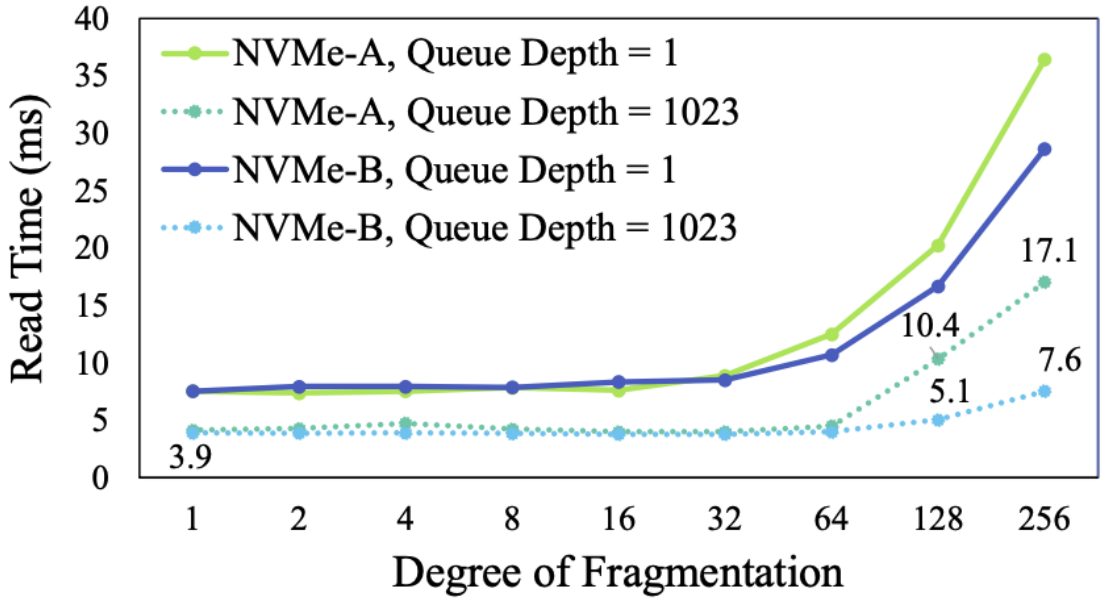
Good parallelism!

Locate in subsequent die

Analysis of File Fragmentation

Table 1: System configurations for experiments.

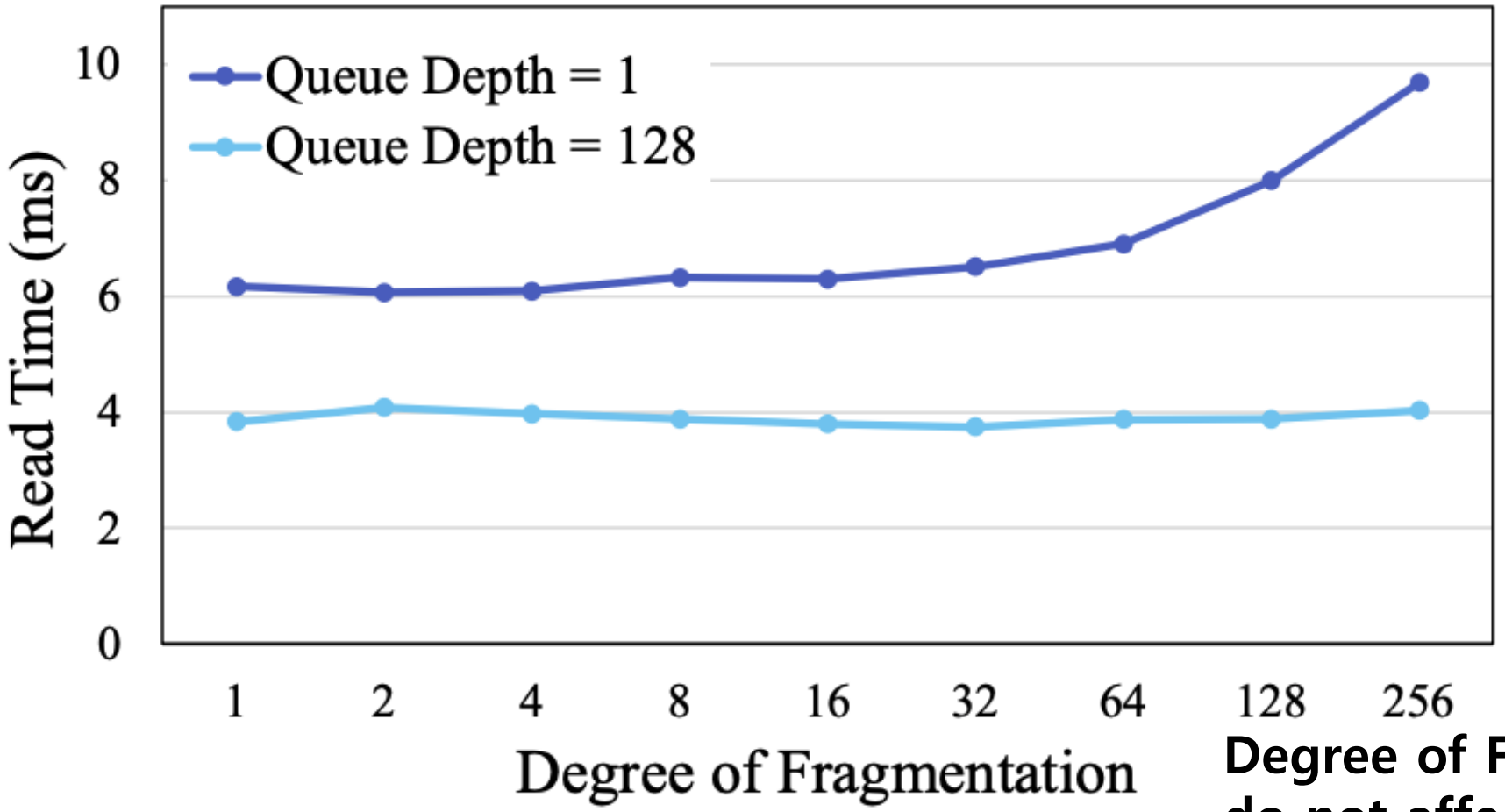
Processor	Intel Xeon Gold 6138 2.0 GHz, 160-Core
Chipset	Intel C621
Memory	DDR4 2666 MHz, 32 GB x16
OS	Ubuntu 20.04 Server (kernel v5.15.0)
Interface	PCIe Gen 3 x4 and SATA 3.0
Storage	NVMe-A: Samsung 980 PRO 1 TB NVMe-B: WD Black SN850 1 TB NVMe-C: SK Hynix Platinum P41 1 TB NVMe-D: Crucial P5 Plus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



Degree of Fragmentation causes performance degradation.

Analysis of File Fragmentation

- **RAMDisk**



Degree of Fragmentation
do not affect to read time in ramdisk
→ No impact from request splitting

Analysis of File Fragmentation

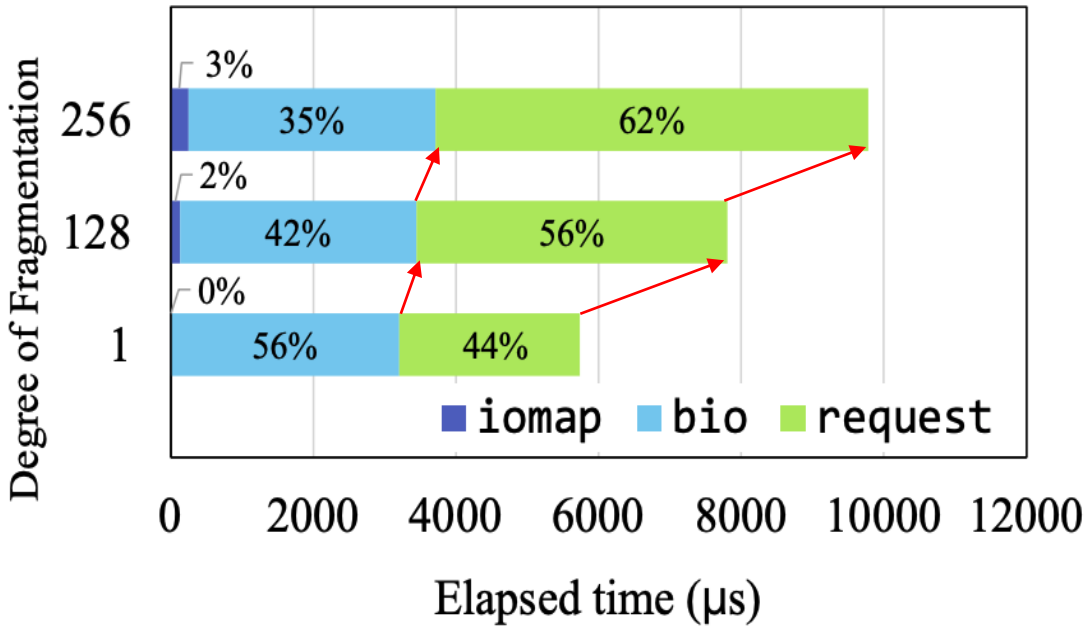


Figure 5: Time composition for creating request data structures in the kernel I/O path depending on File's DoF.

Request time increased proportionally
With the increase in the
Degree of Fragmentation(DoF)

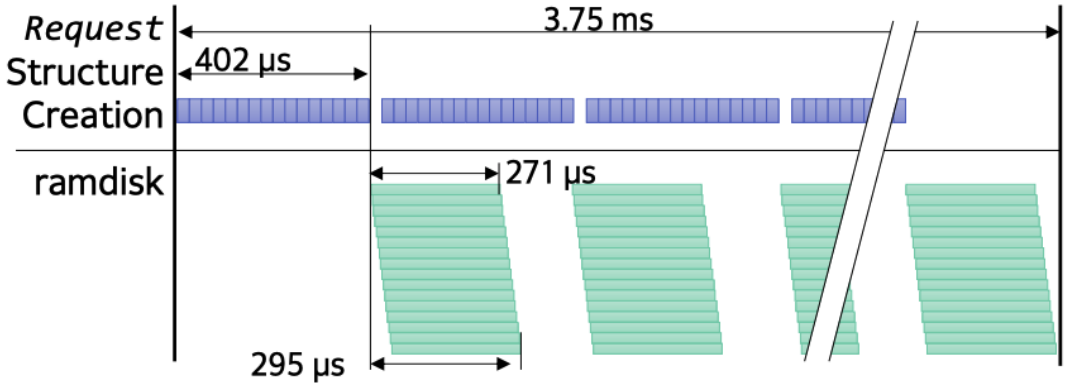
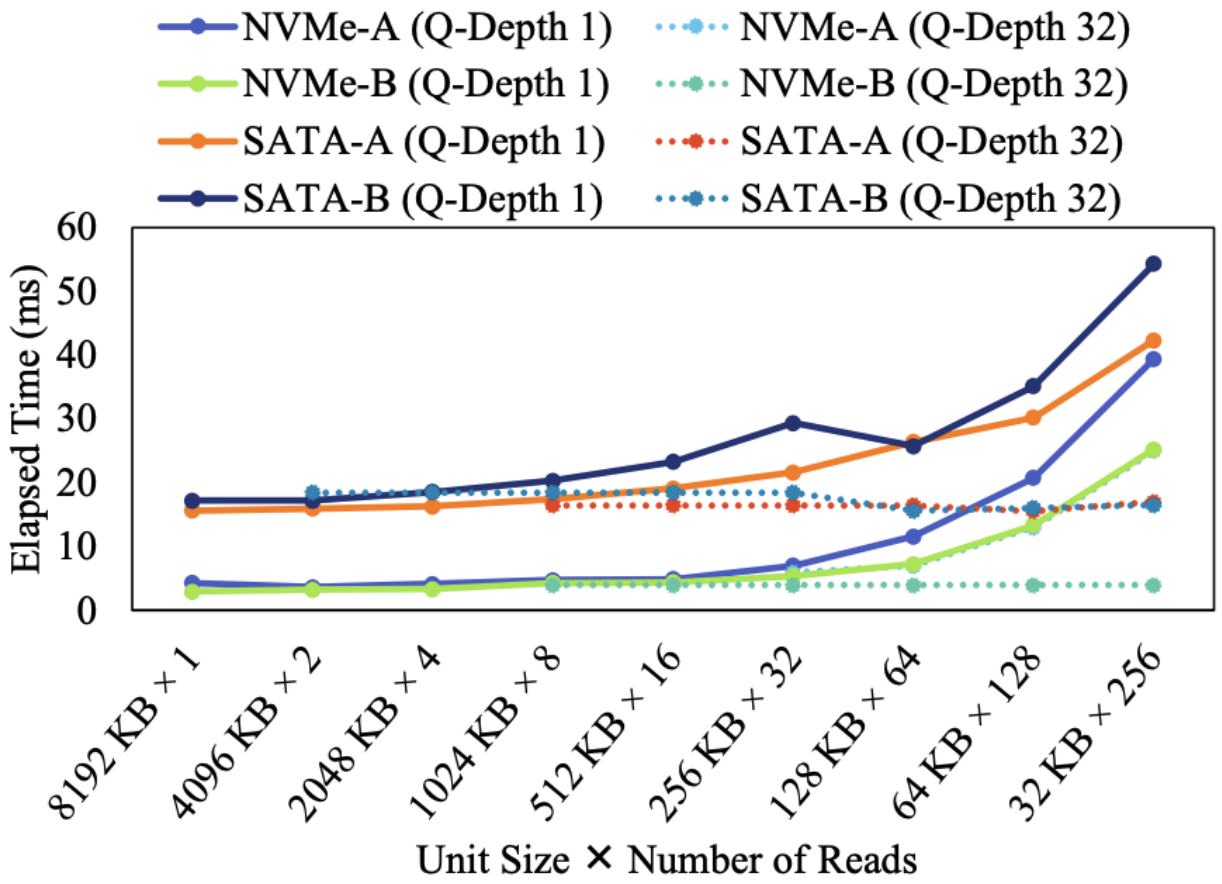


Figure 6: Reduction of read time due to the overlap of storage operations and `request` creation when File's DoF is 128.

"Kernel I/O path can be masked
by I/O queueing"

Analysis of File Fragmentation



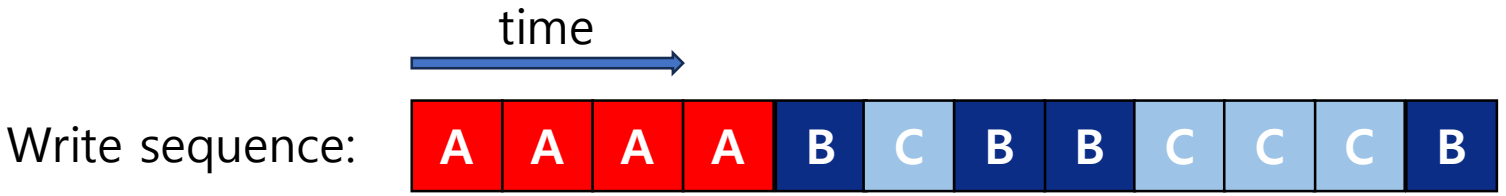
Result

→Request splitting overhead in the kernel I/O path is negligible

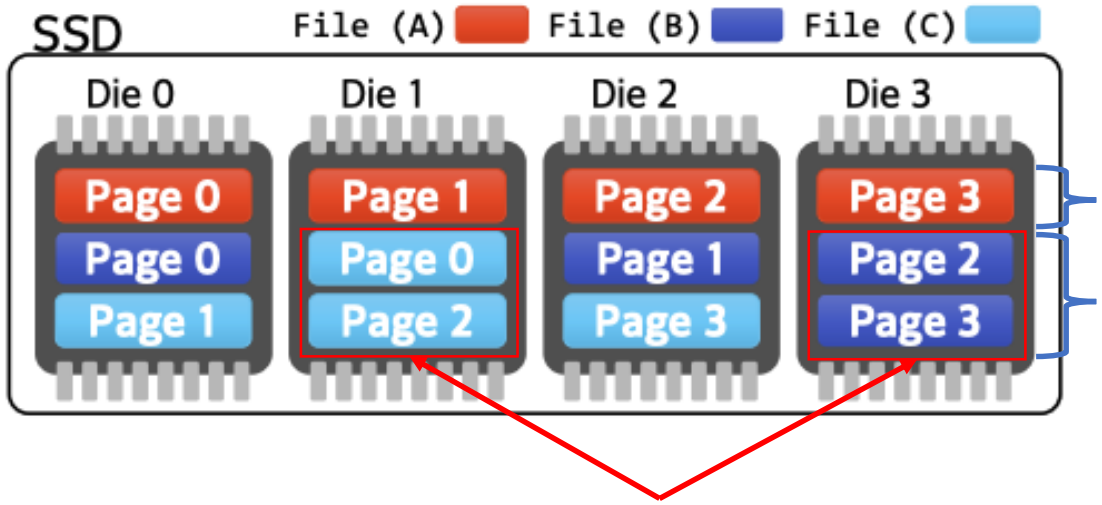
→ Request splitting overhead is mitigated wh en issuing I/O operations asynchronously through command queueing

Analysis of File Fragmentation

- **Page misalignment**



Write in round-robin manner



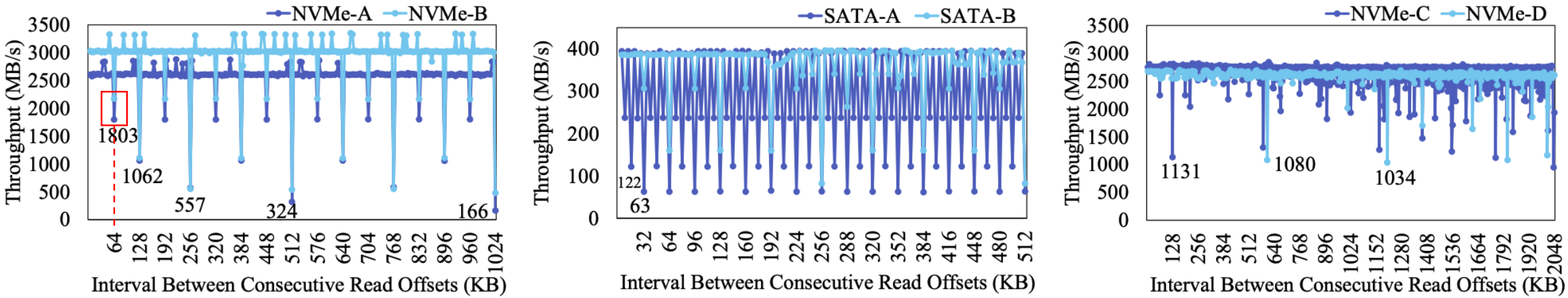
File A: up to 4x more bandwidth than single die

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

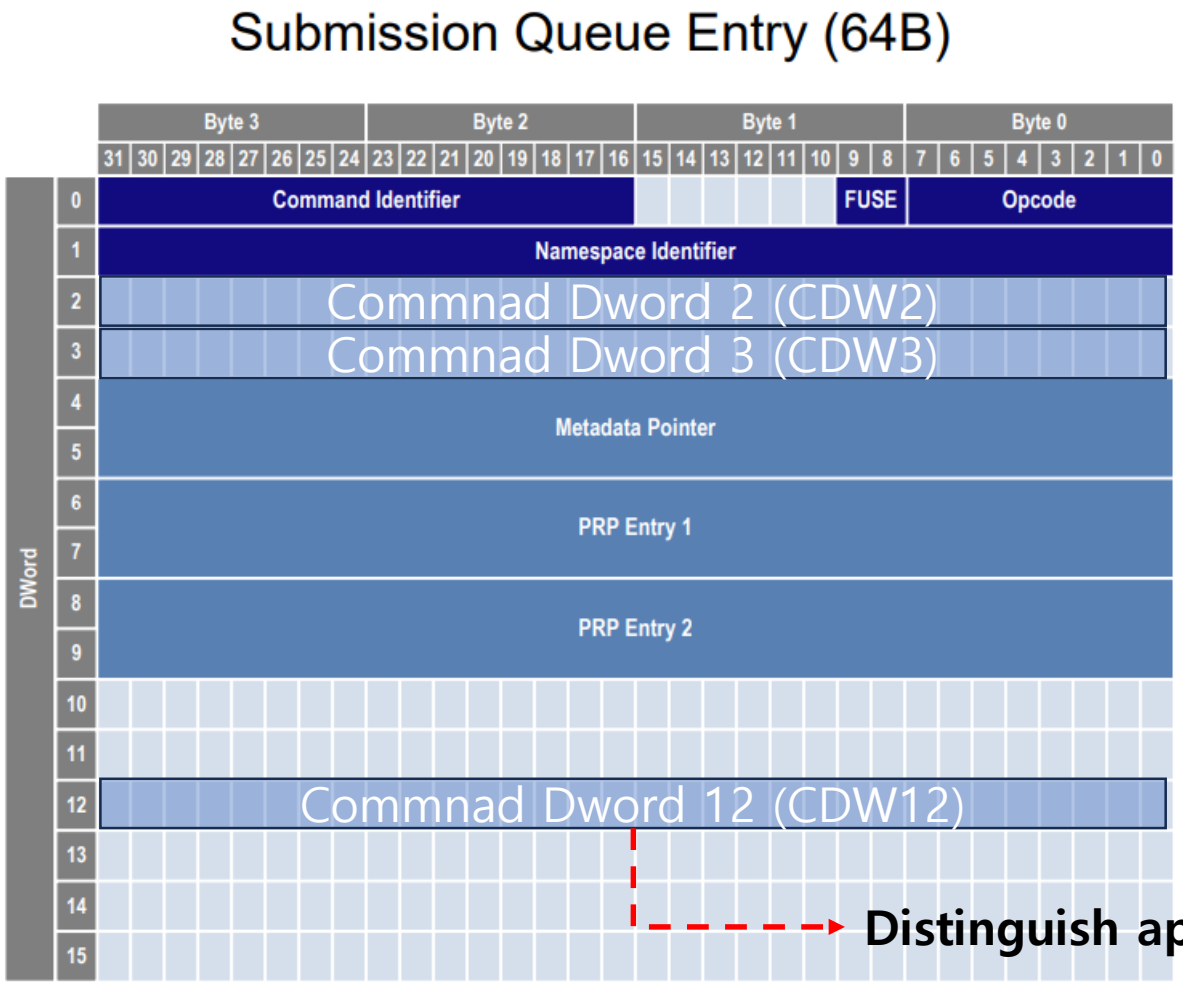
Figure 8: Throughput while varying the interval between starting points of consecutive read operations.

(a) Both NVMe SSD's page size is 16KB -> Allocates 2 pages per die

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

Approach

- **Uses NVMe protocol's write command**



Convey the LBA of the preceding file block

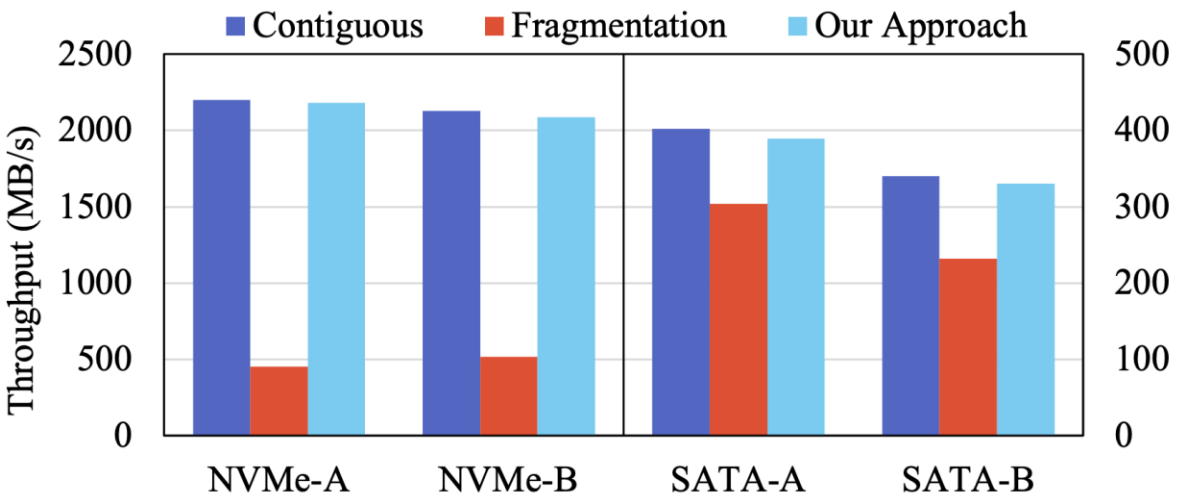
Distinguish append writes and overwrites from conventional writes

Ref: https://files.futurememorystorage.com/proceedings/2012/20120821_TD12_Onufryk.pdf, https://nvmexpress.org/wp-content/uploads/NVM-Express-Base-Specification-Revision-2.1-2024.08.05-Ratified.pdf

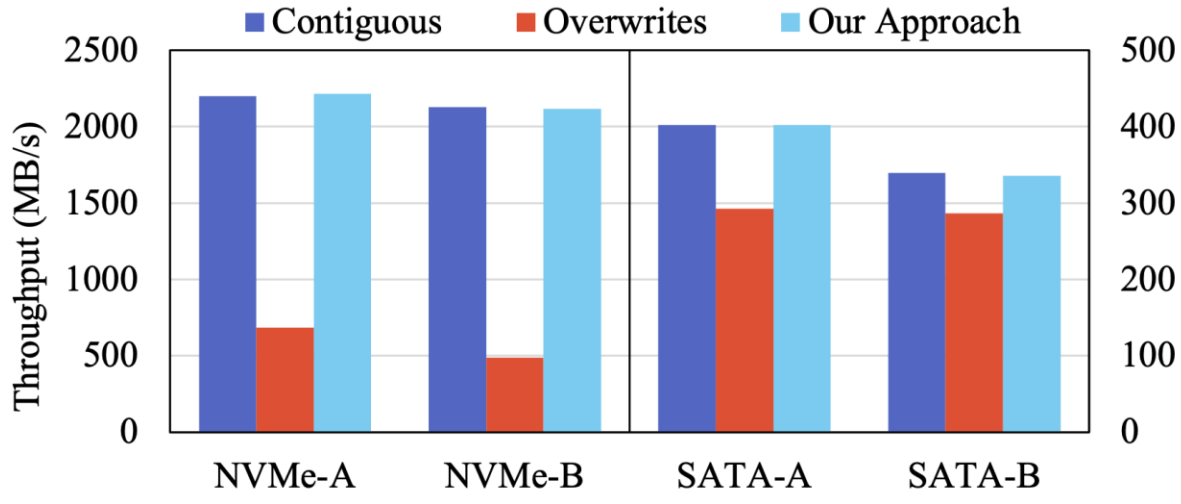
Evaluation

- **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**



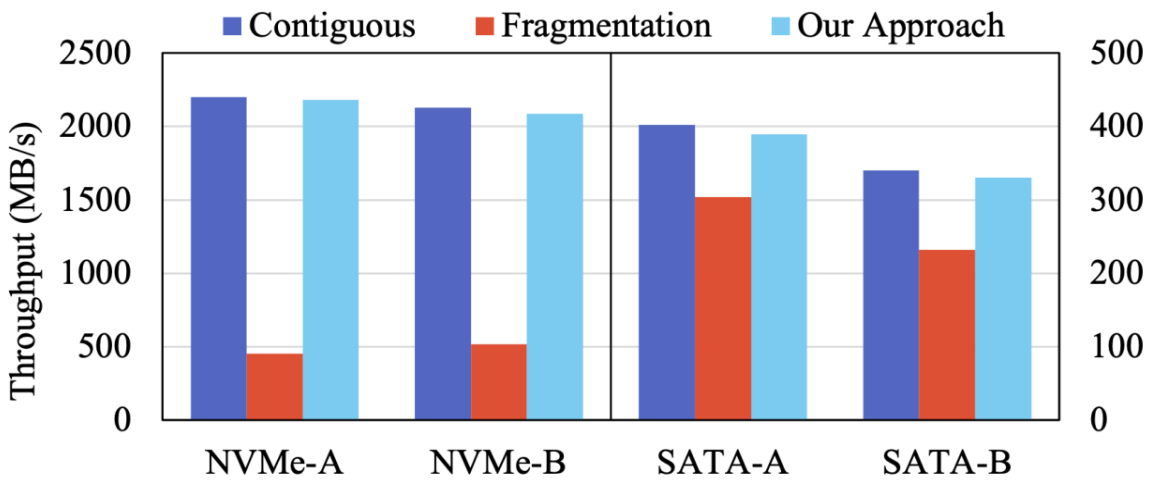
(a) Append Write



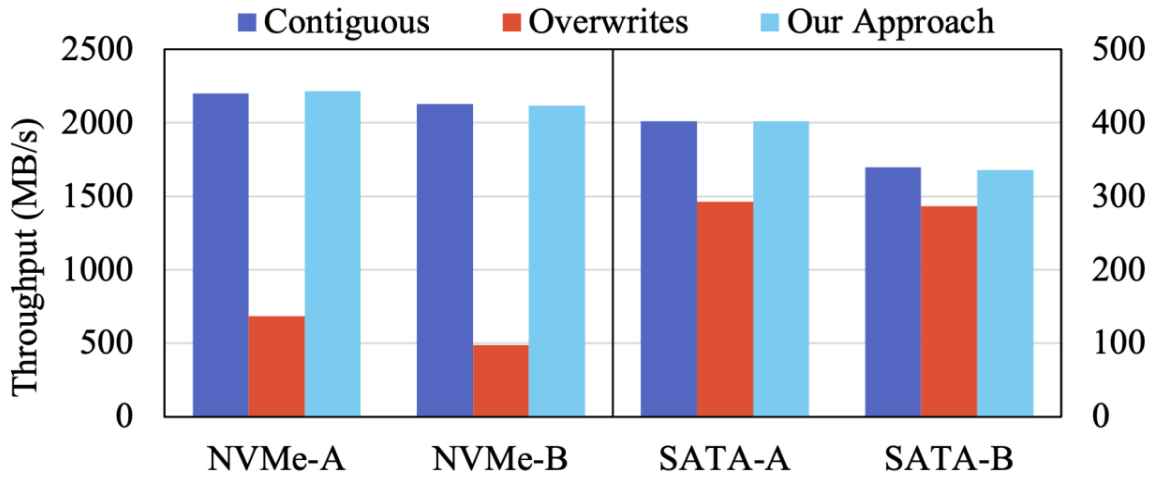
(b) Overwrite

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

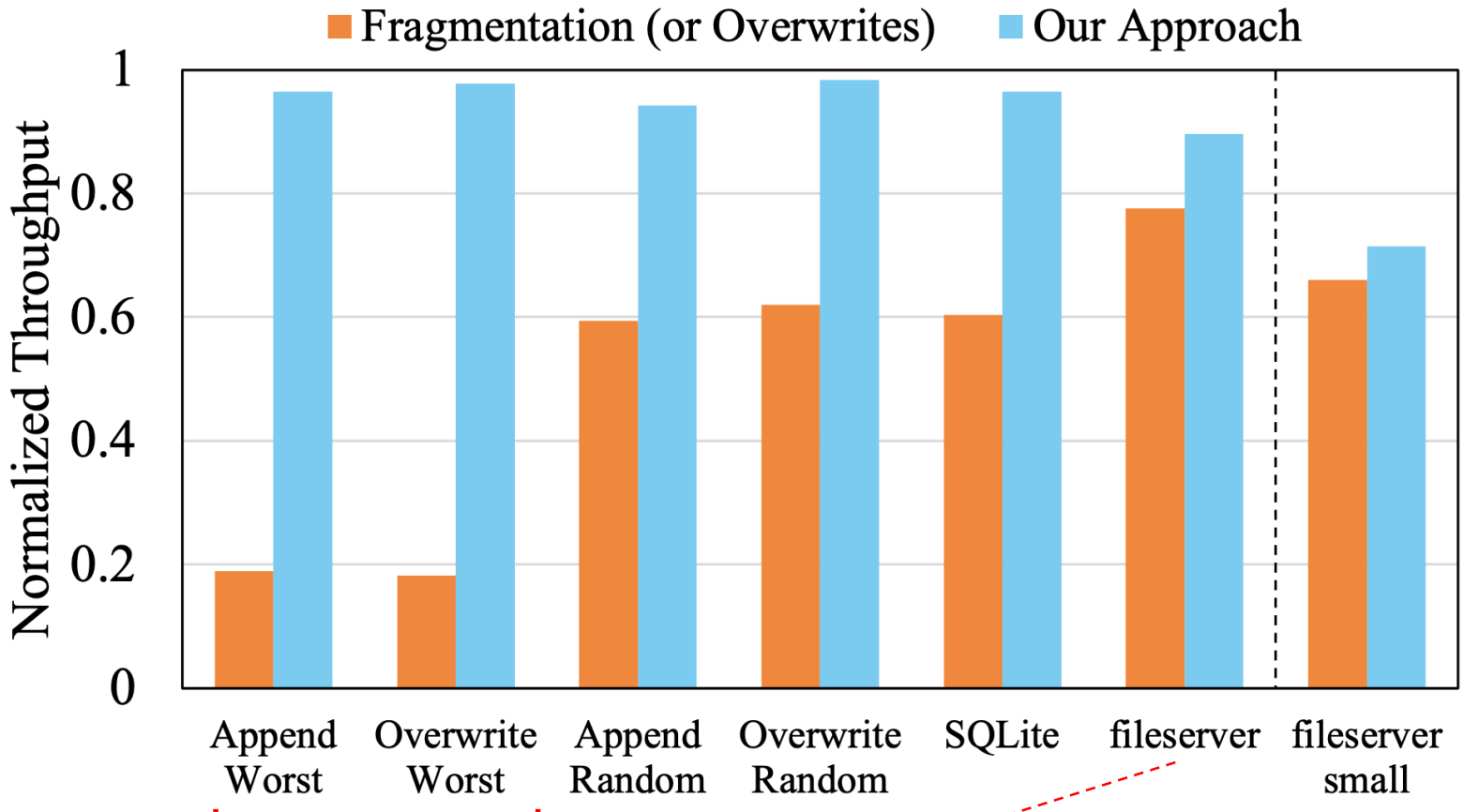
Evaluation

- **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 !

2024. 08. 14

Presented by Juhyun Kim & Yongmin Lee

(email) , nascarf16@dankook.ac.kr

Fragmentation

- **Discontinues data block**

→ Random access to scattered fragment

- **About HDD**
- **Tool for frag**

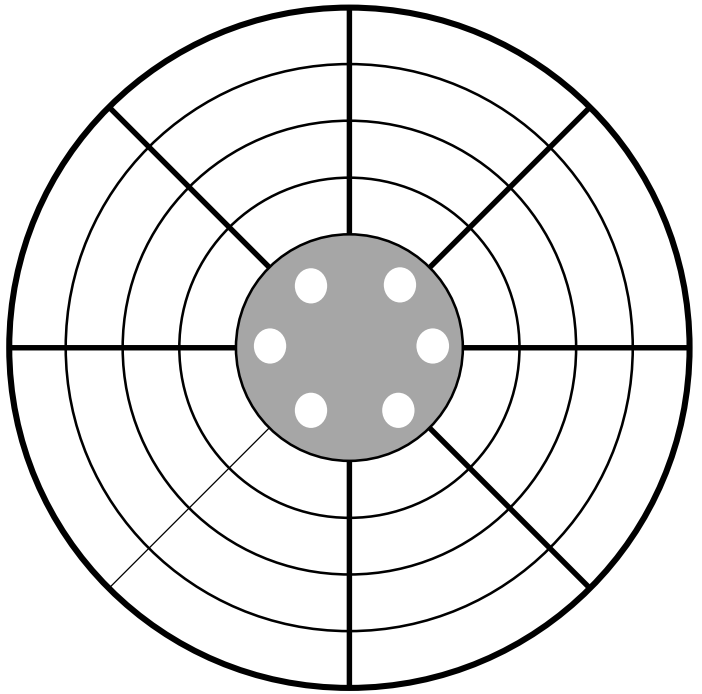
- 지연할당, 데이터 블록 사전 할당 … etc…

→ For 데이터 블록 간의 연속성 유지

→ But 동시 파일s write or long time before additional file write

→ 항상 연속된 데이터 블록 사용 불가능

File 1 :
File 2 :
File 3 :
File 4 :
…



Fragmentation

- **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

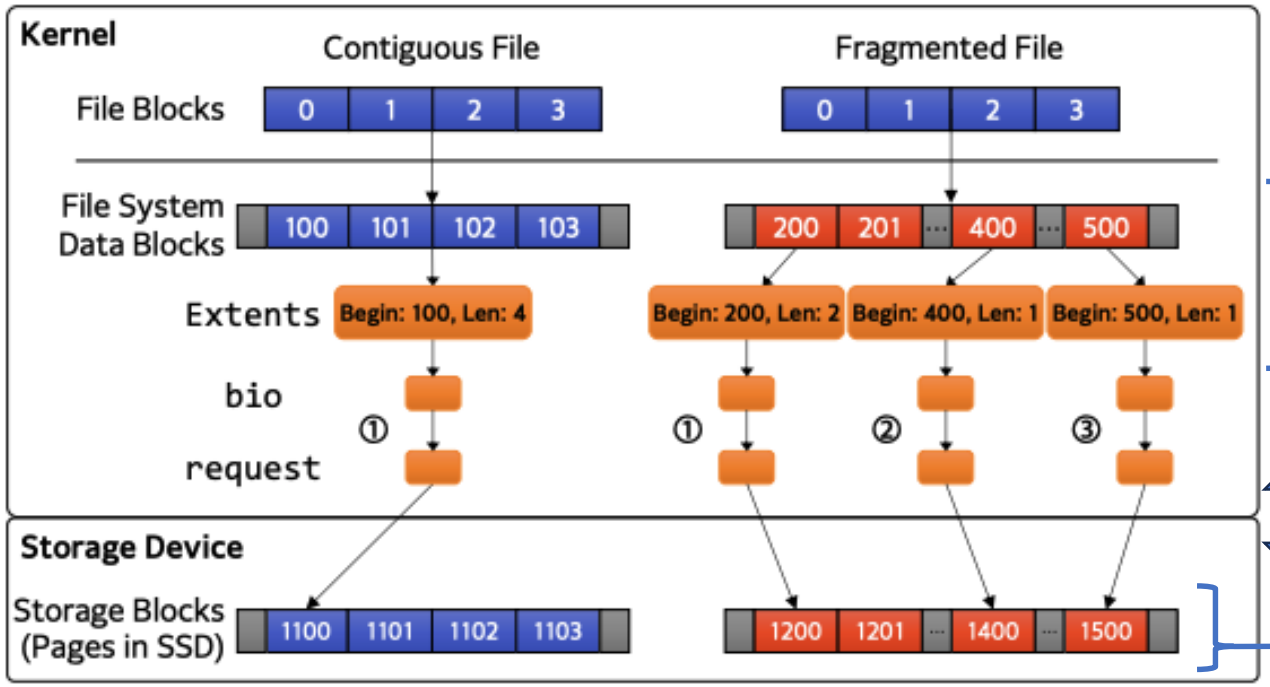


Figure 1: A sequential access to a contiguous file is translated to a single device command while that to a fragmented file ends up with multiple requests.

Kernel I/O path

Interface

Storage device interface

Storage media access

1. Introduction

- SSD
  - 본 논문에서는 조각화로 인한 성능 저하의 원인은 요청 분할이 아닌, SSD 내부의 병렬 처이 능력의 저하로 인한 다이 레벨 충돌!
  - Ssd 페이지 할당 문제
    - 플래시 메모리 페이지를 쓰여진 순서에 따라 라운드 로빈 방식으로 다이에 할당
    - → 파일이 frag 되면 연속된 다이에 배치되지 않고, 무작위 다이에 배치 → 읽기 성능 저하
    - → NVMe 프로토콜 확장과 페이지 투 다이 할당 알고리즘