An Empirical Study of File-System Fragmentation in Mobile Storage Systems

Cheng Ji [§] **, Li-Pin Chang**[※]**, Liang Shi**[¶]**, Chao Wu** [§]
Qiao Li[¶]**, Chun Jason Xue** [§]

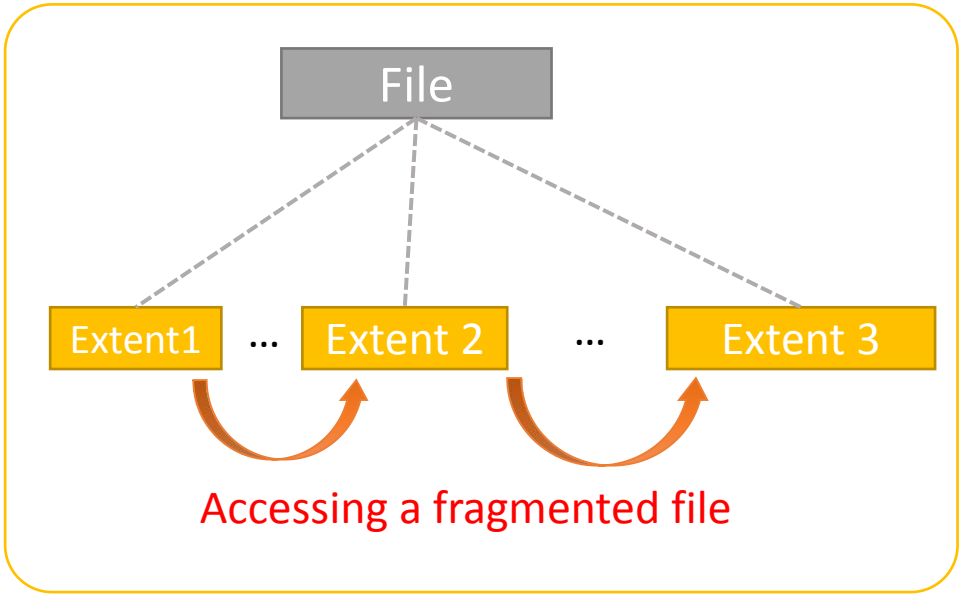
[§] City University of Hong Kong
[※]National Chiao-Tung University
[¶]Chongqing University

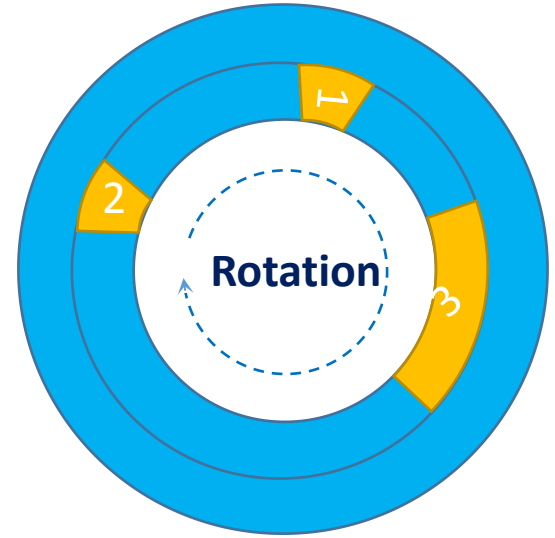
Overview

- Background and Motivation
- Characteristics of Fragmentation
- Evaluation of Fragmentation
- Pilot Solutions
- Conclusions

Background – File System Fragmentation

- A common problem in file systems
  - Emerges when file cannot get continuous free space
- Negatively affects I/O performance of hard disks
  - Increasing the frequency of disk head movements





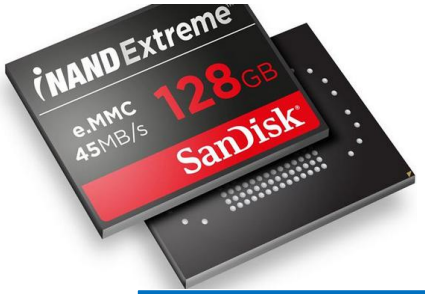
Increased disk seek time

Background – Mobile Devices and Flash Storage

- Mobile devices are becoming the necessities for daily life
  - They run file systems on top of flash storage, e.g., eMMC



Mobile devices



Flash storage

- I/O performance is crucial to system performance
  - Performance drops when operations of database and filesystem are combined [LEE et al., EMSOFT'12] , e.g., "JOJ" problem [Kim et al. FAST'14]

However, no work targets to study fragmentation impact on mobile devices.

Motivating Questions

- Q1：Whether there is fragmentation on mobile devices?
- Q2：Does fragmentation impact the performance of mobile devices?
- Q3：How to resolve fragmentation problem for mobile storages?

Overview

- Background and Motivation
- Characteristics of Fragmentation
- Evaluation of Fragmentation
- Pilot Solutions
- Conclusions

Fragmentation in Aged Smartphones

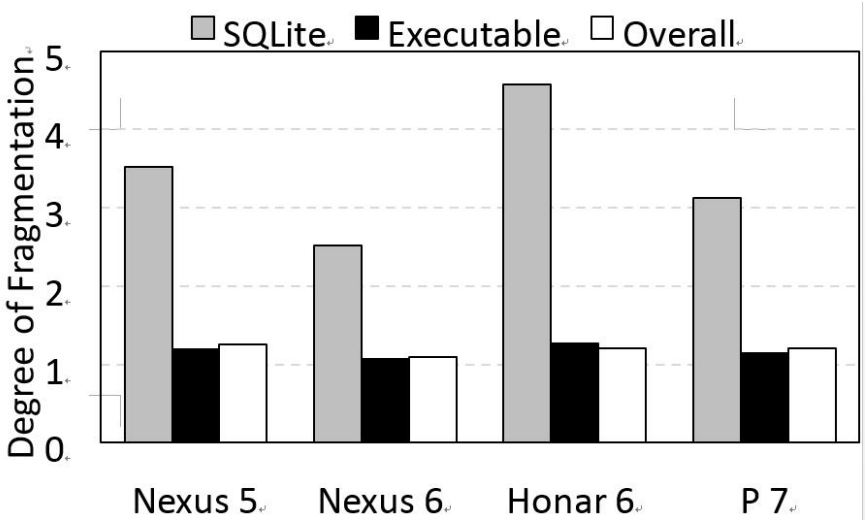
- 4 Randomly selected smartphones were Investigated
  - Common usages including web surfing, sending/receiving emails, social networking, and taking pictures

	Nexus 5	Nexus 6	Honor 6	Ascend P7
Usage Time	1 year	6 months	6 months	1 year
Linux Kernel	3.4.0	3.10.40	3.10.30	3.0.8
Android Version	4.4.4	5.5.1	4.4.2	4.4.2
`Data` Capacity	26.8 GB	26 GB	11.6 GB	11.8 GB
`Data` Utilization	93%	57%	44%	90%

Characteristics of the Aged Smartphones

Fragmentation in Aged Smartphones

- Single File Fragmentation are reported
  - Degree of Fragmentation (DoF) = N_{ext} / N_{lowest_ext}



Database files are severely fragmented

1. Most are those with extensions **.db** and **.db-journal**
2. Small fragment size, ranging from tens to hundreds of kilobytes

Fragments on SQLite Files

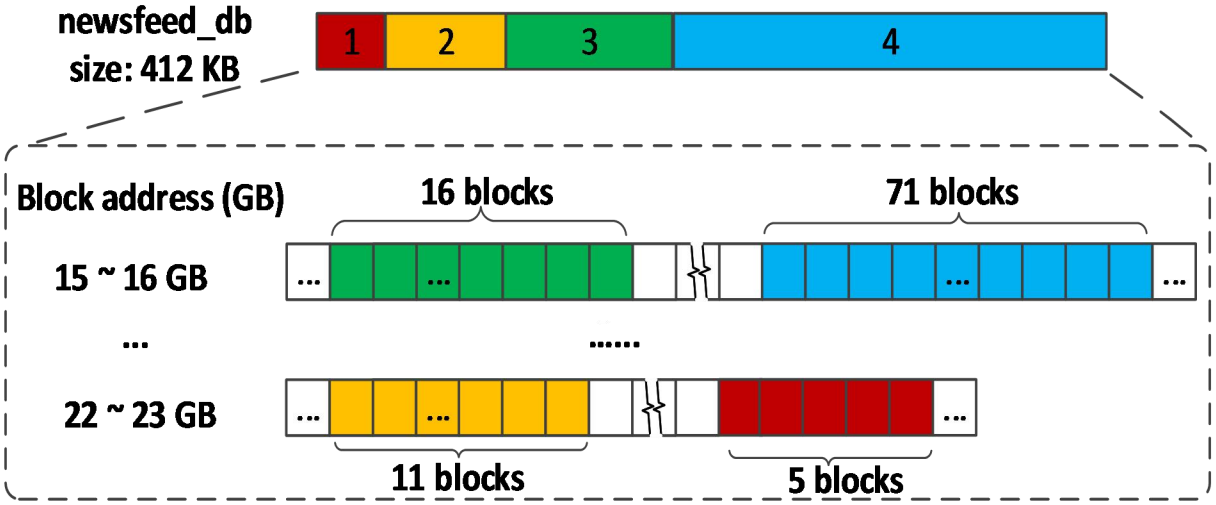
- Reasons for fragmented SQLite files
  - Filesystem aging
    - Small “holes” make file fragments inevitable under high space utilization
  - Writing behaviors
    - Multiple database files in the same directory are growing in parallel

Fragments on SQLite Files

- Reasons for fragmented SQLite files
  - Filesystem aging
    - Small “holes” make file fragments inevitable under high space utilization
  - Writing behaviors
    - Multiple database files in the same directory are growing in parallel
- No, it is not caused by buggy apps or artificial fragmentation!
  - Observing fragmentation under a newly formatted file system
    - Run apps on Nexus 5 with low space utilization 19% (previous 93%)

Fragments on SQLite Files

- Experiment results
  - Facebook: 48 files have 190 fragments (extends)
    - Fragmented pieces were dispersed over 1 GB address space (9 out of 48)
    - Caused by the SQLite writing behavior



Files are still severely fragmented even under low space utilization!!!

Overview

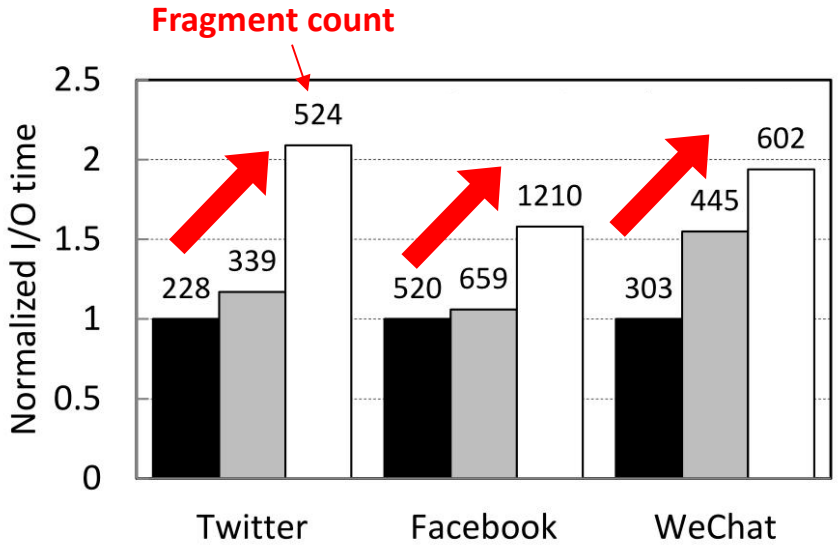
- Background and Motivation
- Characteristics of Fragmentation
- Evaluation of Fragmentation
- Pilot Solutions
- Conclusions

Overall Problem

- I/O latency vs. fragmentation
  - Emulate an aged file system on Ascend P7
    - Fill large files and small files alternatively
    - Delete small file randomly to create small “holes”
  - Use *Mobibench* to replay the file-system traces in the aged file system

Overall Problem

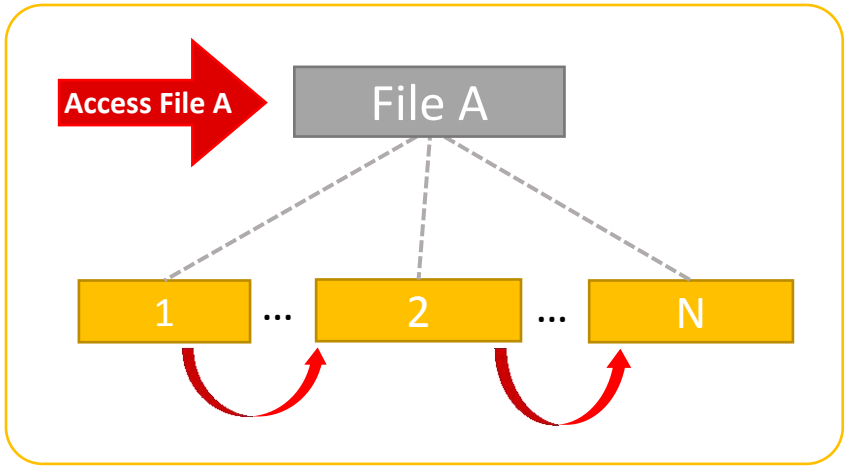
- I/O latency vs. fragmentation
  - Emulate an aged file system on Ascend P7
    - Fill large files and small files alternatively
    - Delete small file randomly to create small “holes”
  - Use *Mobibench* to replay the file-system traces in the aged file system
- But wait… shouldn't flash storage be immune to fragmentation?



I/O latencies are proportional to degree of fragmentation !

Two Impacts Caused by Fragmentation (1)

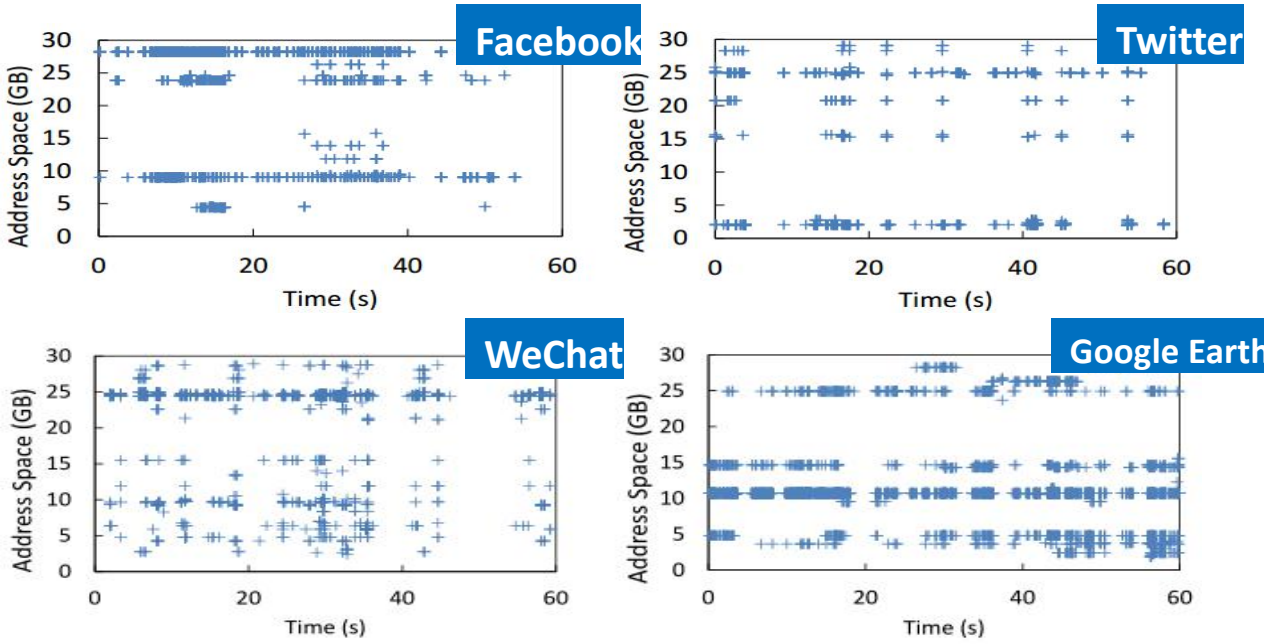
- First: ***Increased Block I/O Frequency***
  - Accessing fragmented file requires more block I/O frequency
  - 10% read performance drop← 13% more block I/O



Higher block I/O frequency

Two Impacts Cause by Fragmentation (2)

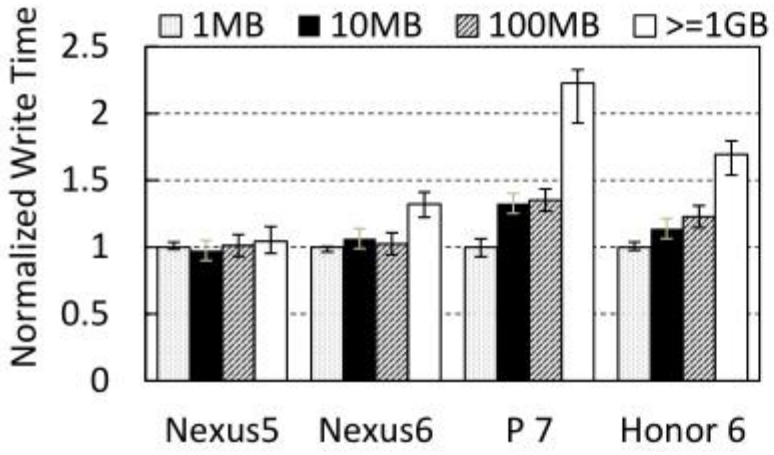
- Second: ***Dispersed Block I/O Patterns***
  - Observation: I/O traces associated with database files



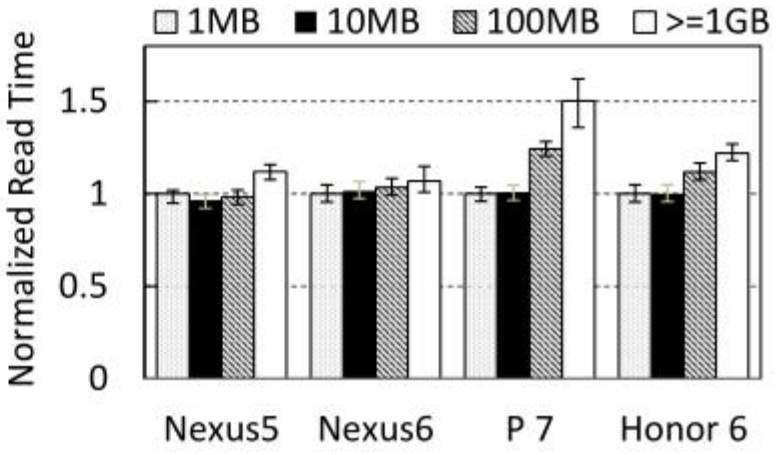
  - DB files contribute to 70% of block writes [LEE et al., EMSOFT'12]!

Dispersed Block I/O Patterns

- Shouldn't flash storage be immune to fragmentation?
- Real measurement says no!
  - Random read/write over different disk regions
  - 1 GB: highly dispersed, 1MB: concentrated



(a) Random sync writes

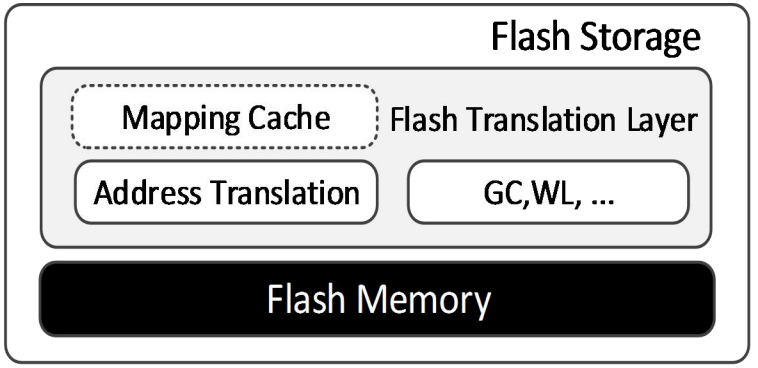


(b) Random reads

Not only write latency, but also read latency becomes higher when the I/Os are highly dispresed!

I/O Patterns vs. Mapping Cache

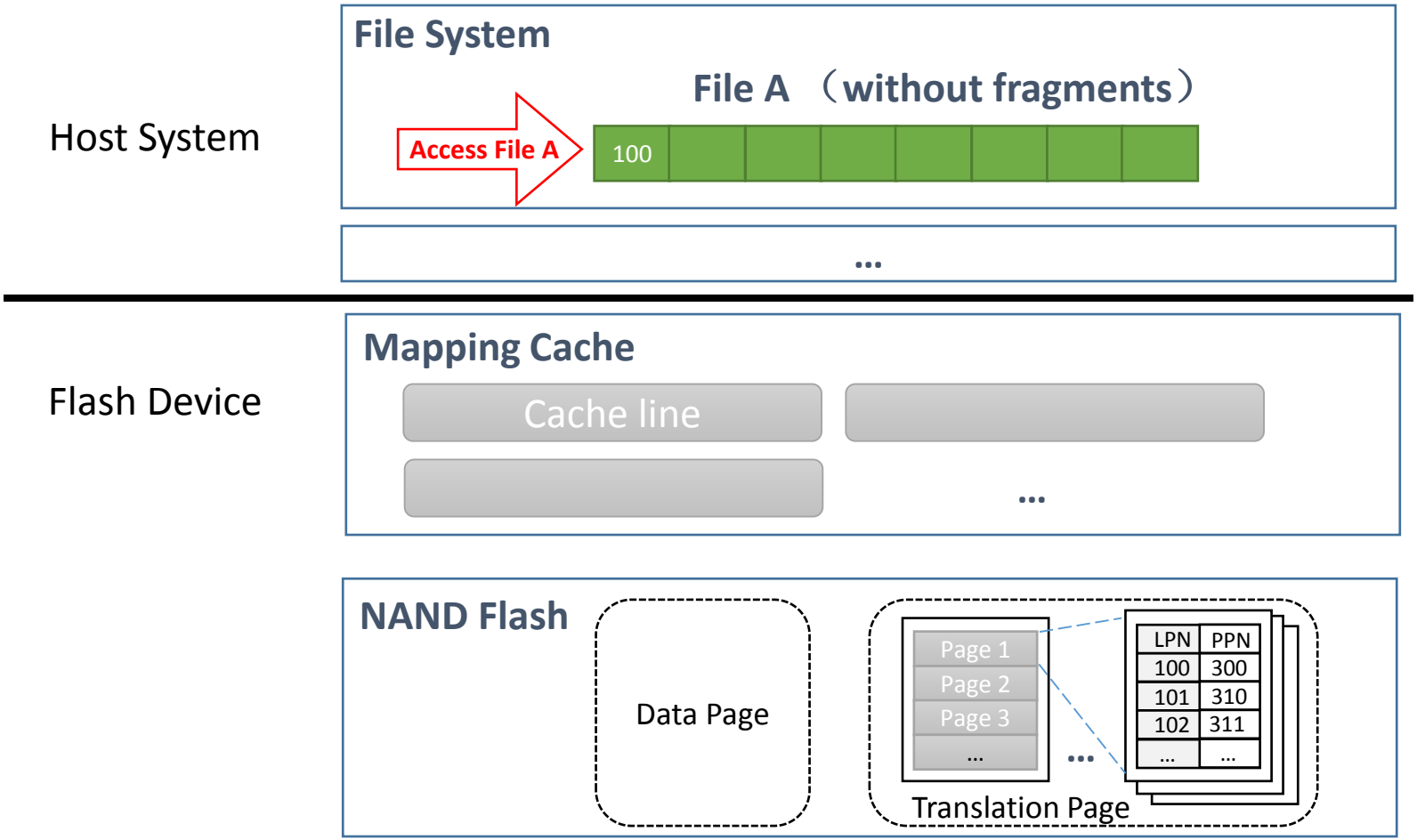
- What is the cause of the slow access under dispersed I/Os?
  - Based on our industry experience, many mobile storage devices (eMMCs) employ demand-based mapping cache
    - Caching a small active portion of mapping table
    - Spatial locality exploitation:
      - Prefetching multiple mapping entries
      - Prefetching is seriously harmful under random access



Mapping cache overhead will be highly related to I/O latencies on flash storage

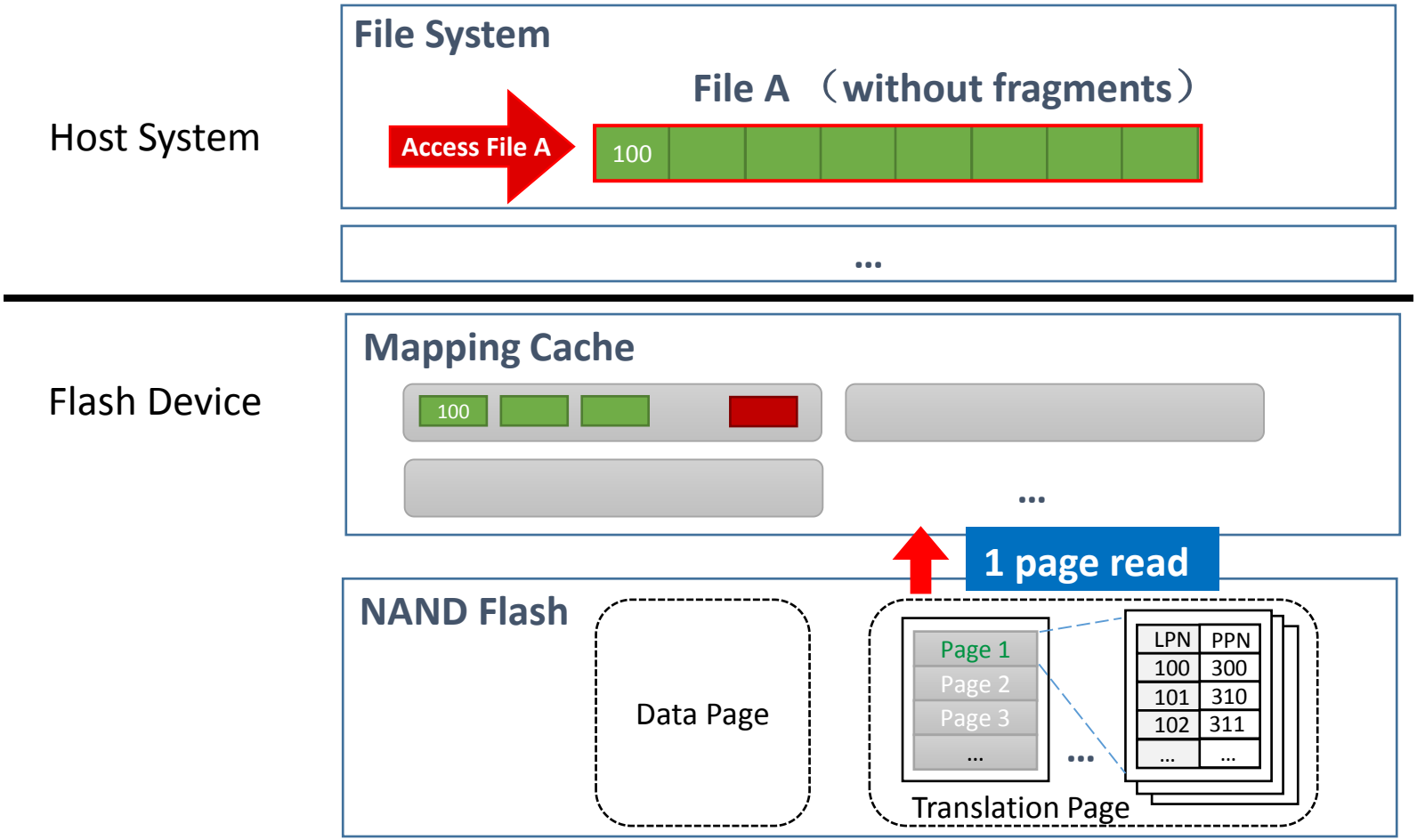
Mapping Cache Simulation

- Simulation Settings
  - Cache line size (4 KB), replacement algorithm (LRU)



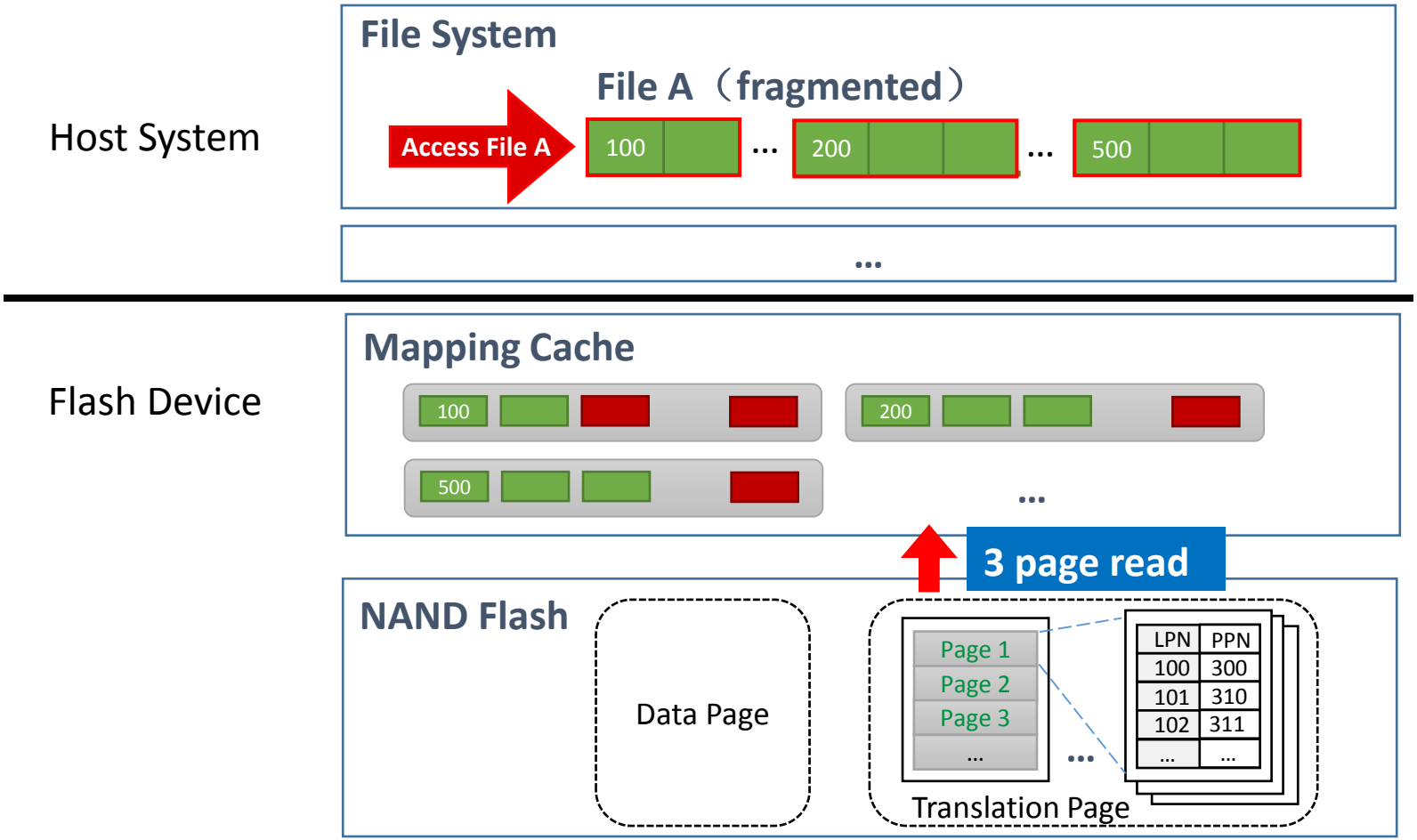
Mapping Cache Simulation

- Accessing file without fragments



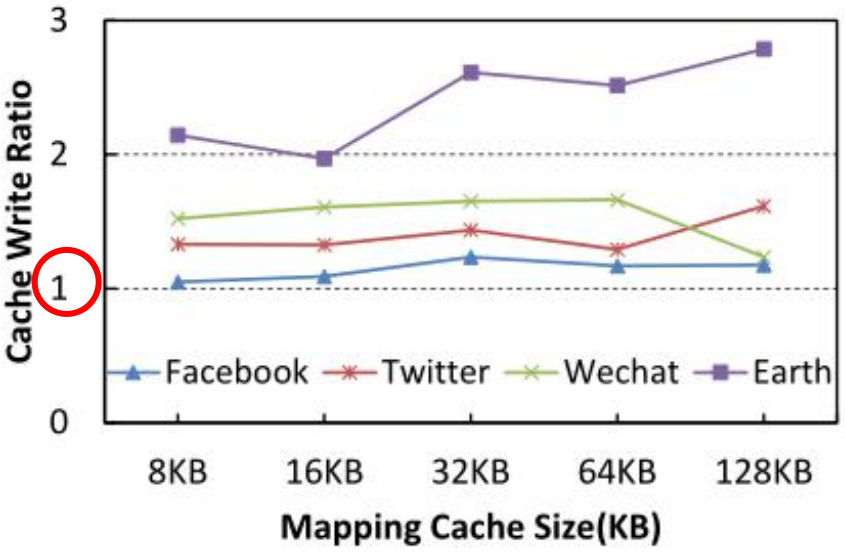
Mapping Cache Simulation

- Accessing fragmented file

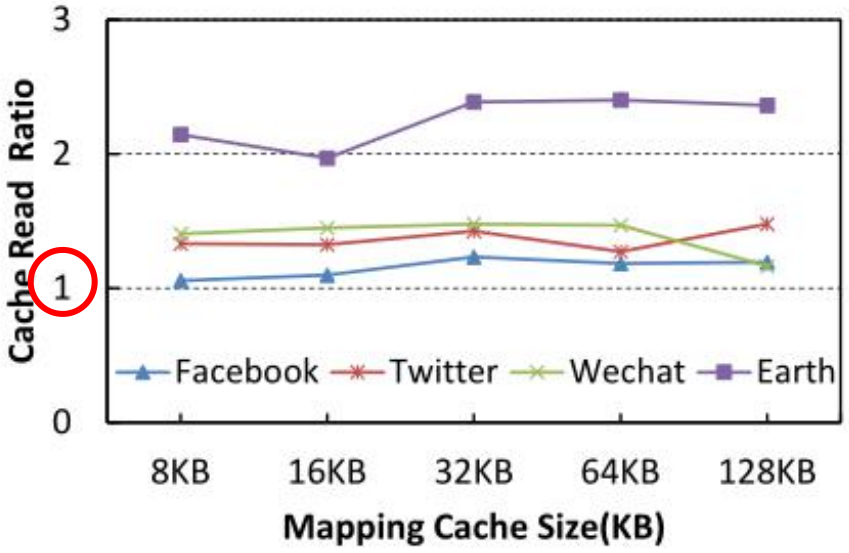


Mapping Cache Simulation

- Block trace replay
  - Before and after defragmenting database files
- The higher the ratio is (>1), the higher the pressure on cache is



(a) Cache Write Ratio



(b) Cache Read Ratio

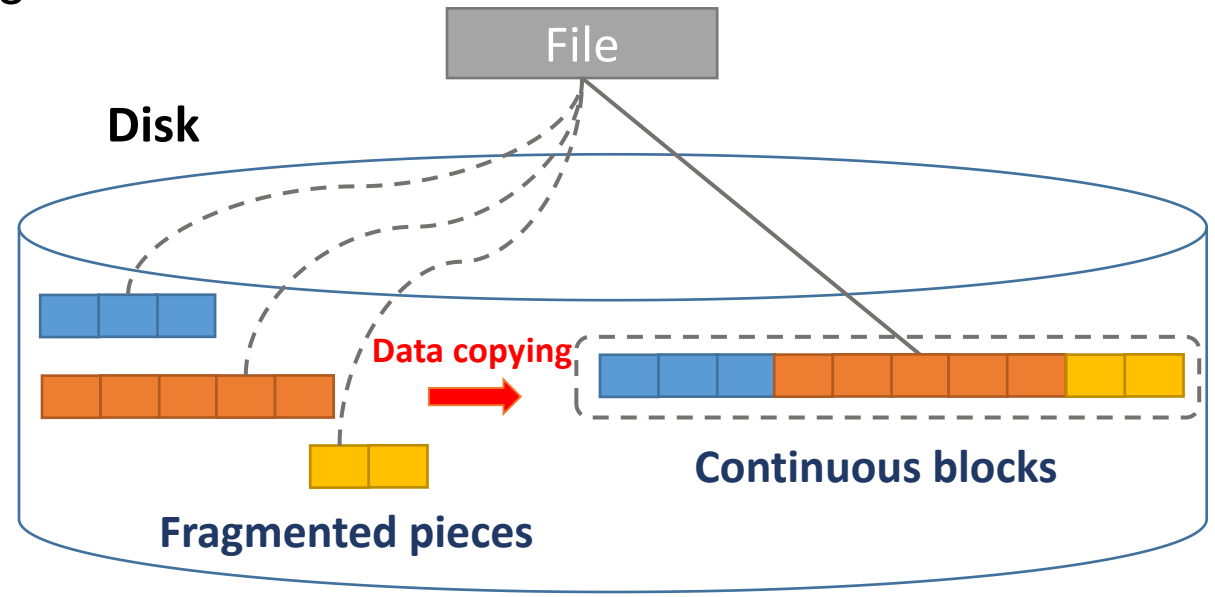
Dispersed access patterns impose a high pressure on the mapping cache

Overview

- Background and Motivation
- Characteristics of Fragmentation
- Evaluation of Fragmentation
- Pilot Solutions
- Conclusions

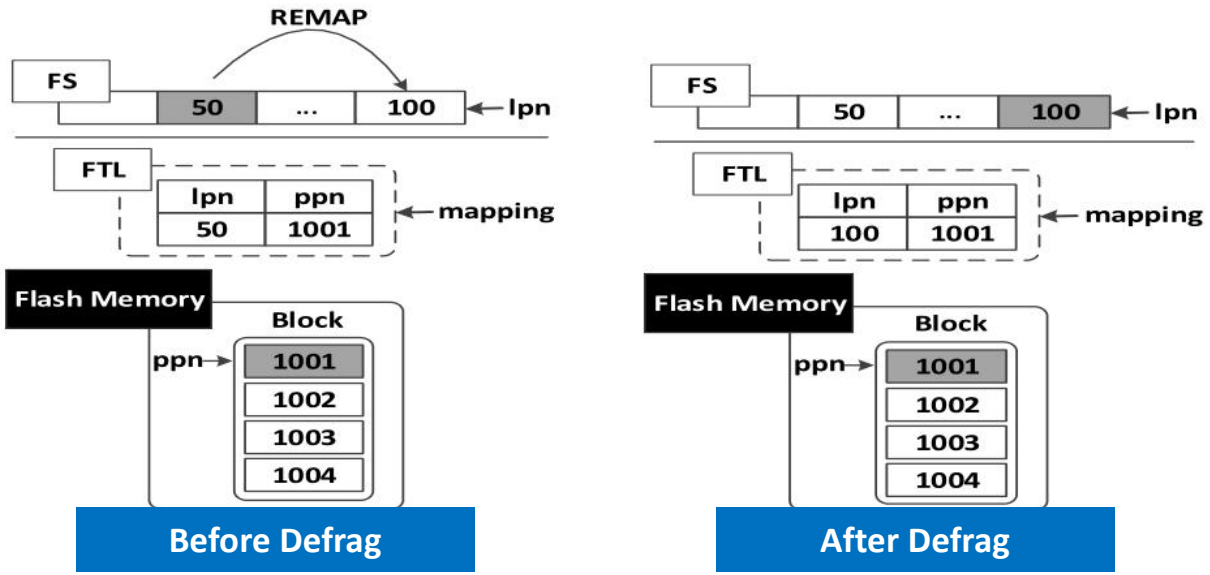
Conventional Defragmentation

- Migrating fragmented file pieces to sequential free space
  - Massive data copying
  - A passive solution
- No defragmentation schemes have been suggested for flash storage



Solution 1 - Defragmenting Flash Storage

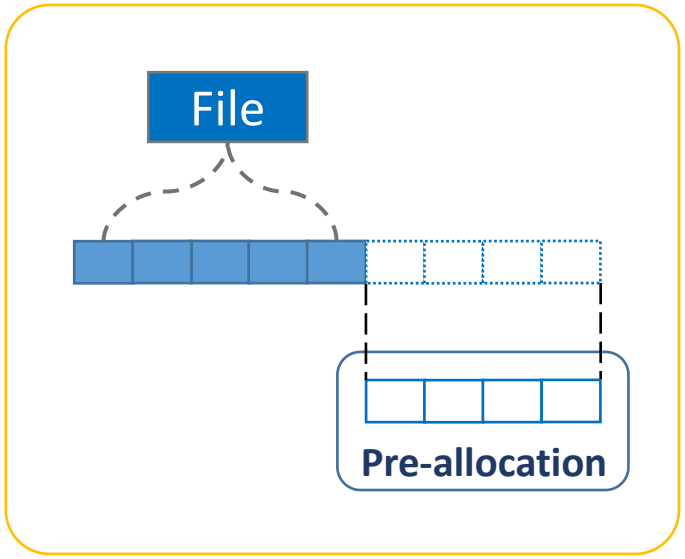
- Exploit the mapping scheme in the FTL
  - Adjusting the mapping cache, instead of copying data
  - Through a new block command

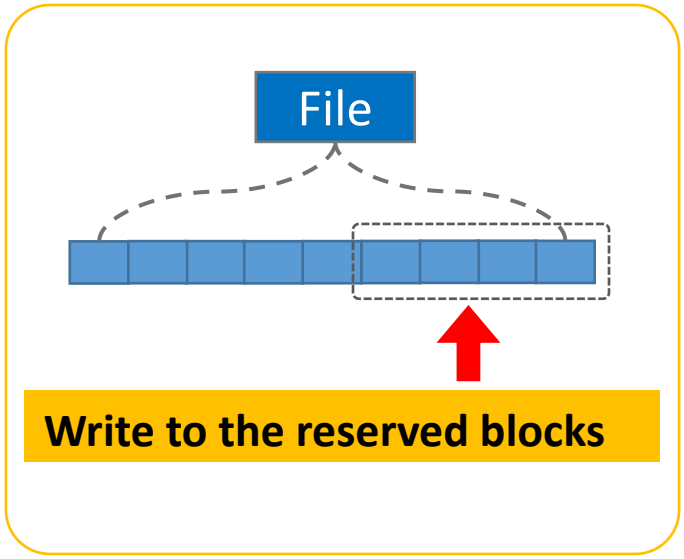


Restoring the logical sequentiality without actually copying data in flash!

Solution 2 - Proactive Fragmentation Avoidance

- Proactively avoid fragmentation by allocating sufficient continuous space for files to grow
  - Target files: database files 
  - Small pre-allocated space would be sufficient





Overview

- Background and Motivation
- Characteristics of Fragmentation
- Evaluation of Fragmentation
- Pilot Solutions
- Conclusions

Conclusion

- SQLite database files are severely fragmented in Android
- File fragments impact the I/O performance of mobile devices
  - Amplified I/O frequency
  - Increased mapping cache cost → a unique problem with flash storage
- Two pilot solutions to optimize file fragmentation
- Future work
  - Characterizing the production of fragments,
  - Validating our mapping cache simulation against real eMMCs

Thank you

Questions?