RFLUSH: Rethink the Flush

USENIX FAST'18

Jeseong Yeon[1], Minseong Jeong[1], Sungjin Lee[2], Eunji Lee[1,3]

[1]Chungbuk National University, [2]DGIST, [3]University of Wisconsin–Madison

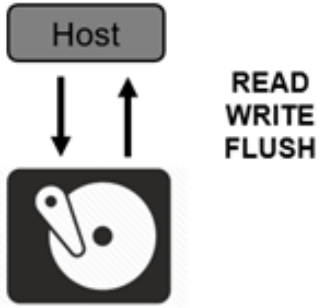






Storage Interfaces are Evolving!

- **Standard block interface**
  - Commands over logical block address space
  - READ, WRITE and FLUSH
  - Abstract of a storage device allows a host to use a device in an efficient manner



Storage Interfaces are Evolving!

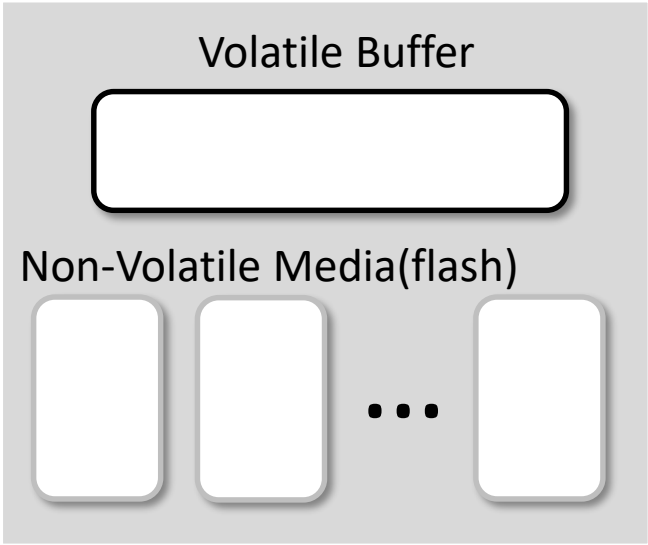
- **Extend interfaces for new media**
  - **Command** – TRIM, Nameless Write, Remap, In-storage Copy, Atomic Update, Multi-stream SSD.
  - **Protocol** – NVMe / PCI-express
  - **Architecture** – Host-manageable SSD, User-programmable SSD



Rethink the Flush

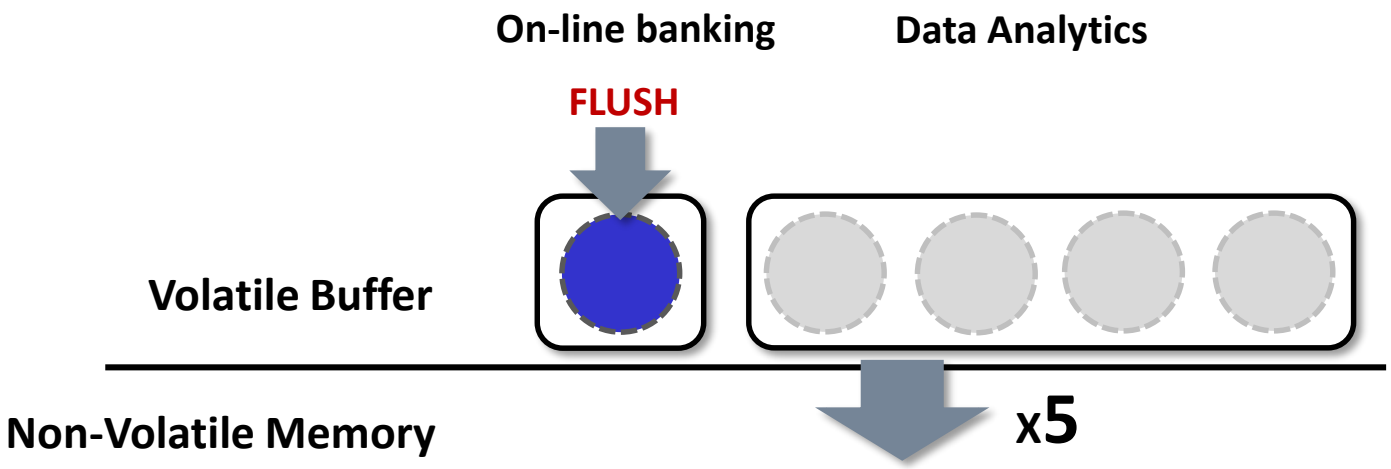
- **Volatile buffer in storage**
  - Absorb write and minimize seeks in HDD
  - Improve random write performance and mask limited endurance in SSD
  - **Data loss** and **improper** ordering in power outage

- **FLUSH command**
  - Force all pending writes immediately
  - Ensure persistence and proper serialization



Rethink the Flush

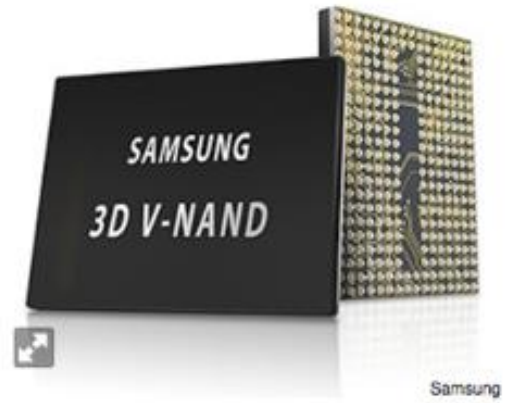
- **Lump-sum approach**



- **(-) Slow down** non-volatile materialization of the writes that need to be made durable
- **(-) Deprive** non-urgent data of an opportunity to **absorb future write** and **coalesce in a buffer**

SSD Trends

- **FLUSH cost rises up** with next-gen SSD
- **Larger buffer** to compensate for steadily increasing latencies and endurance limits
  - 512MB to 2GB
  - Samsung 3D V-NAND 16 TB has 32GB DRAM



Samsung's 48-layer V-NAND chips used in the PM1633a 15.36TB SSD.

- **DRAM-less SSD**
  - Use host memory (Few GBs)
  - Limited form-factor
  - Competitive price

Alternatives

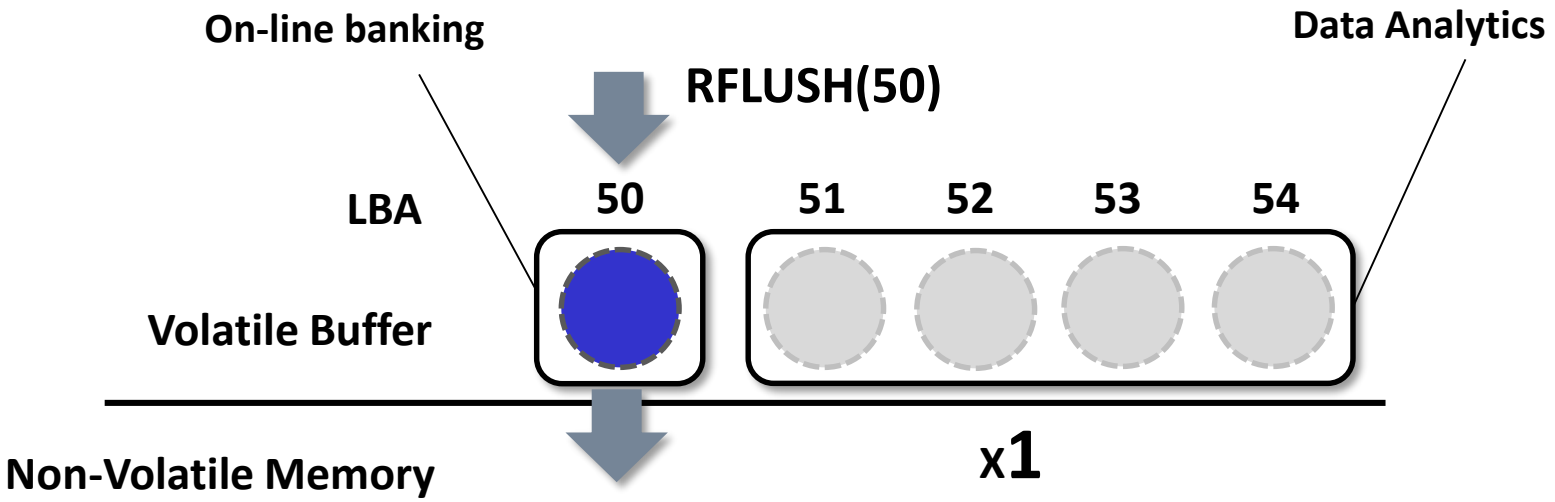
- **Supercapacitor**
  - Provide energy to force all dirty data in volatile buffer at power outage
- **Non-volatile memory**
  - Volatile read cache and non-volatile write buffer
- **Challenges**
  - Increase manufacturing cost
  - Write endurance
  - Limited form factor

Range Flush Primitive

- **RFLUSH**
  - **Fine-grained** control over forcing buffered data
  - Transfer **a range of LBAs** to persist with it

(+) Speed up non-volatile materialization of target data

(+) Enhance buffering and coalescing of other dirty data



Getting Real

- Realizing RFLUSH is not without **challenges**
  - How to **implement** RFLUSH?
  - **Where** to use RFLUSH?
  - **How much** performance gain can be achieved?

- **Prototype** Implementation
  - RFLUSH is implemented in **open-source flash** development platform (BlueDBM)
  - **F2FS** is modified to use RFLUSH during handling of fsync/fdatasync

Contents

- Motivation
- Problem and Solution
- **Implementation**
- Performance Evaluation
- Conclusion

Challenges

- **Where to Use RFLUSH**
- How to Address the Associated Data
- How to Handle Metadata
- How to Integrate into a Storage Protocol

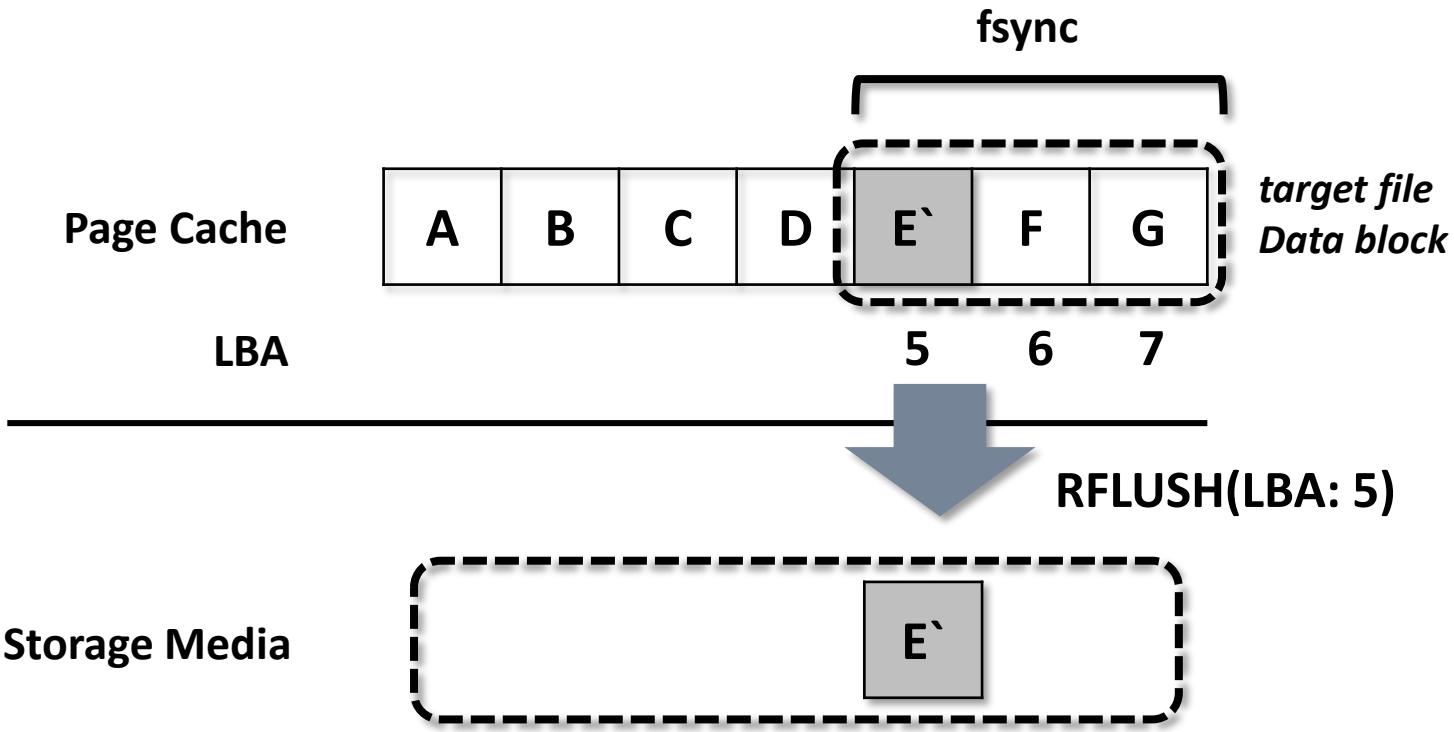
Where to use RFLUSH

- More general than FLUSH
- Allow **fine-grained** control over what to flush
- **FLUSH usage cases**
  - E.g., Transaction commit in file system/ DB / KV-store

- Our Study – **fsync** / **fdatasync**
  - No application modifications – ***fsync*** semantics are preserved
  - Easy to determine associate data for RFLUSH
  - Noticeable performance gains when isolating regions to flush

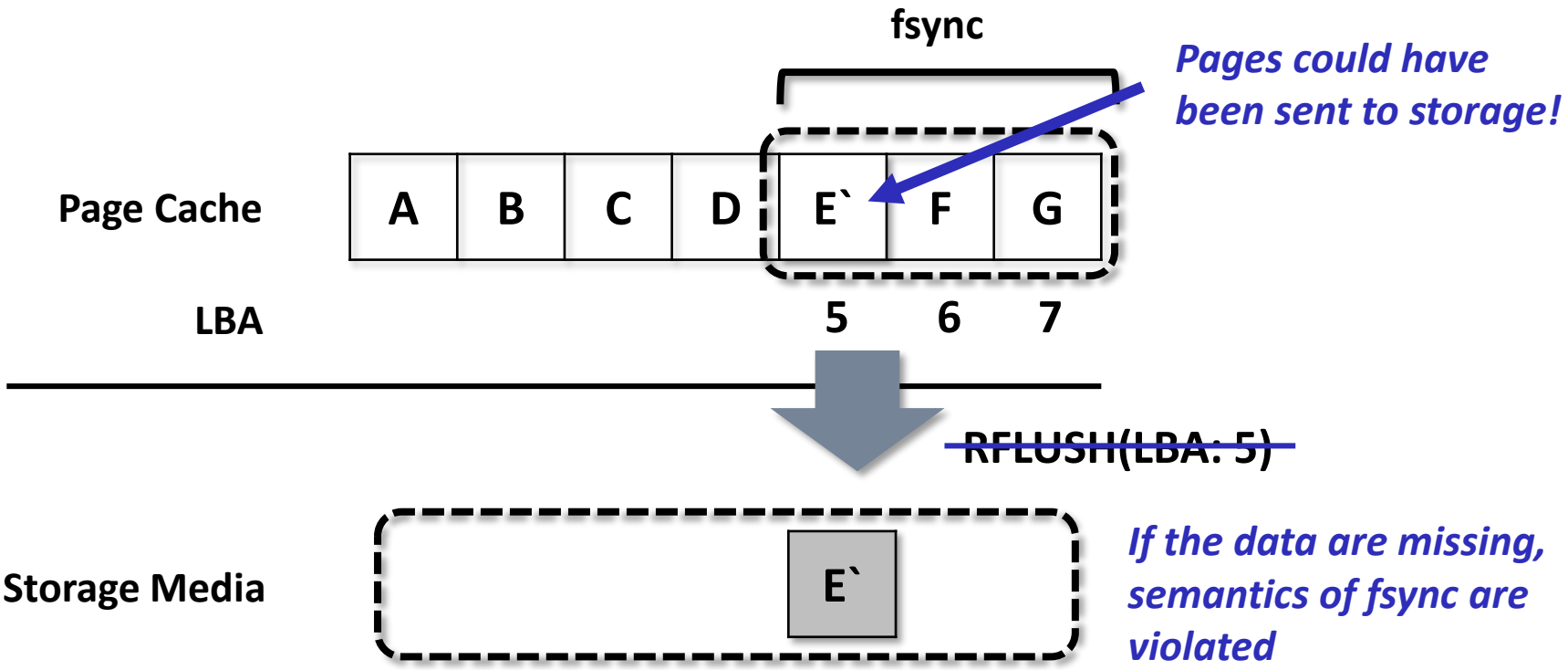
How to Specify Associated Data

- **fsync**
  - Transfer LBAs of dirty pages of target file



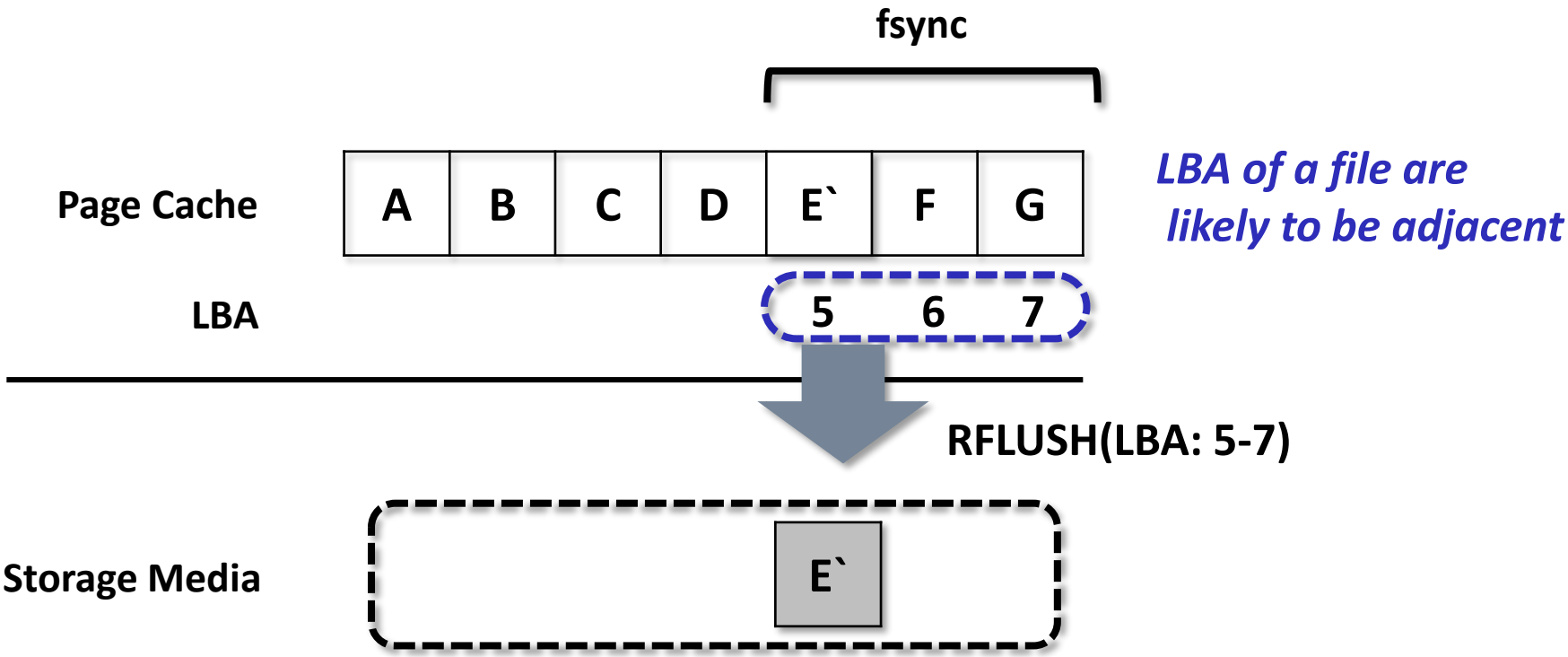
How to Specify Associated Data

- **fsync**
  - Transfer LBAs of dirty pages - ***Incorrect***



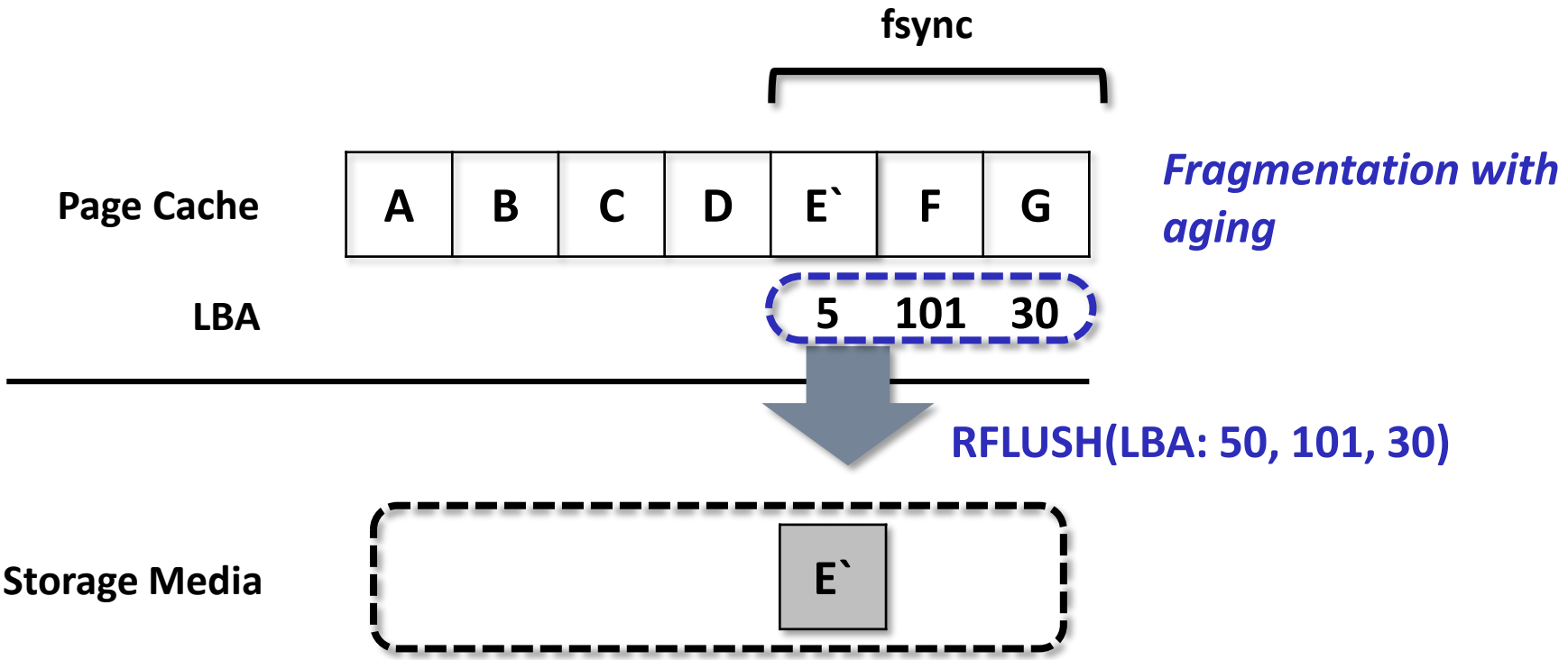
How to Specify Associated Data

- **fsync**
  - Transfer LBAs of dirty pages – ***Incorrect***
  - Transfer **ALL** LBAs of the file



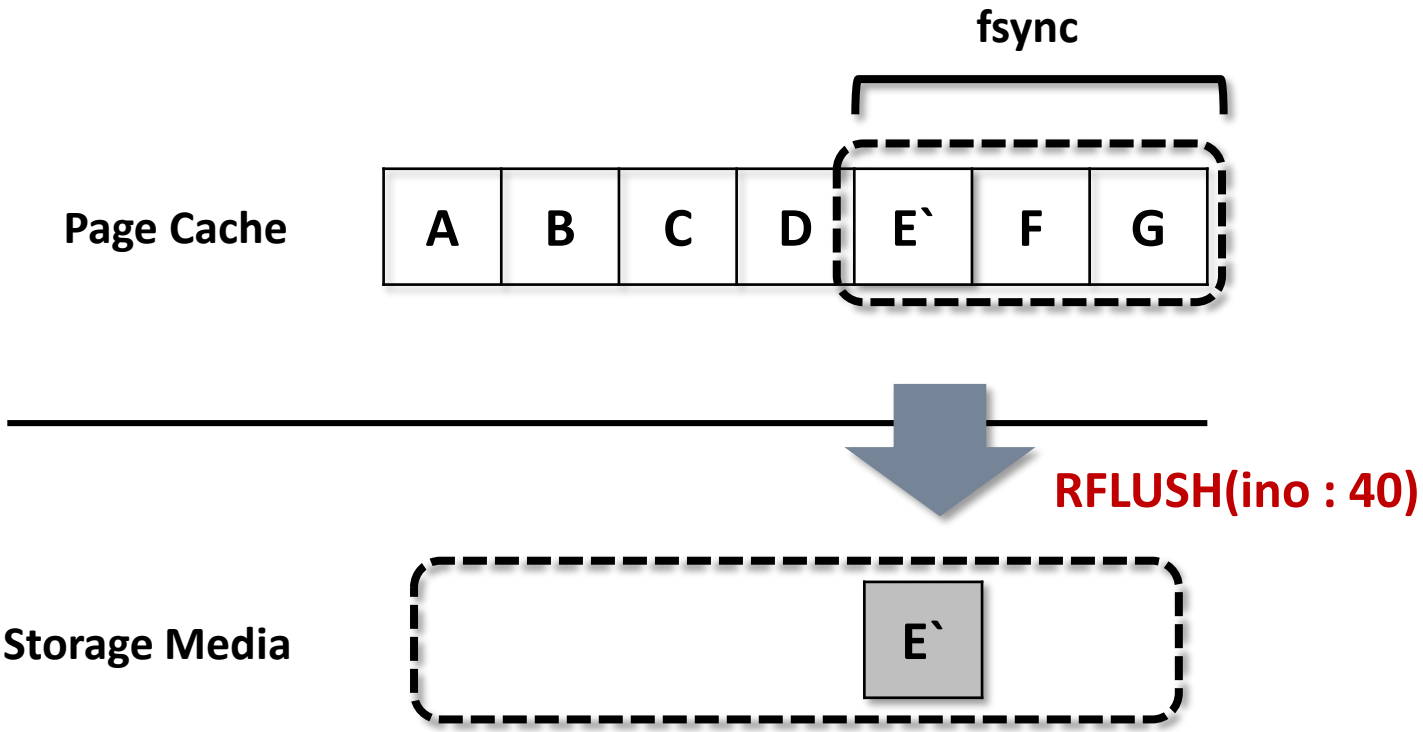
How to Specify Associated Data

- **fsync**
  - Transfer LBAs of dirty pages – ***Incorrect***
  - Transfer **ALL** LBAs of the file – **Correct**, but **costly**



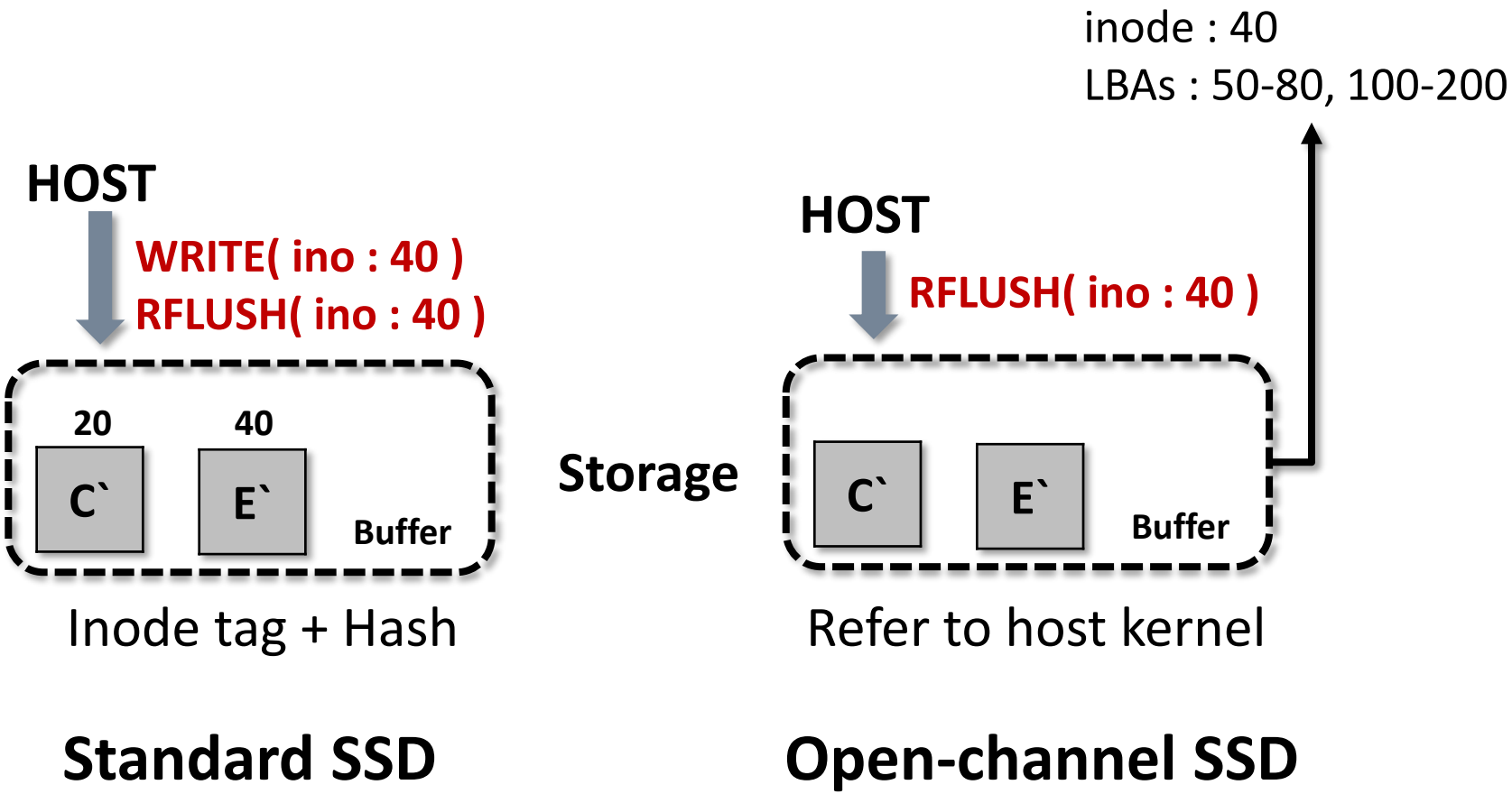
How to Specify Associated Data

- **fsync**
  - Transfer LBAs of dirty pages – ***Incorrect***
  - Transfer **ALL** LBAs of the file – **Correct**, but **costly**
  - Transfer **Inode number** of the target file



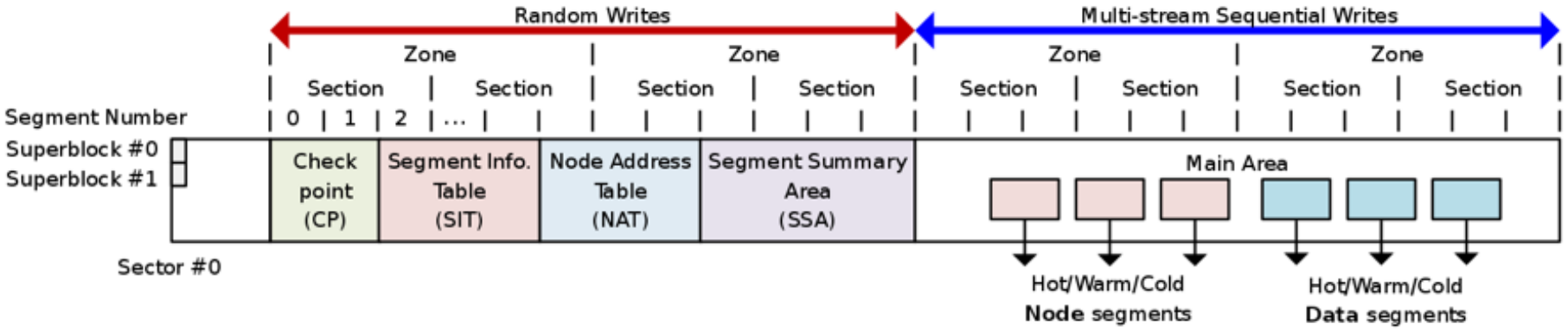
How to Specify Associated Data

- **Ino-based RFLUSH protocol**



How to Handle Metadata

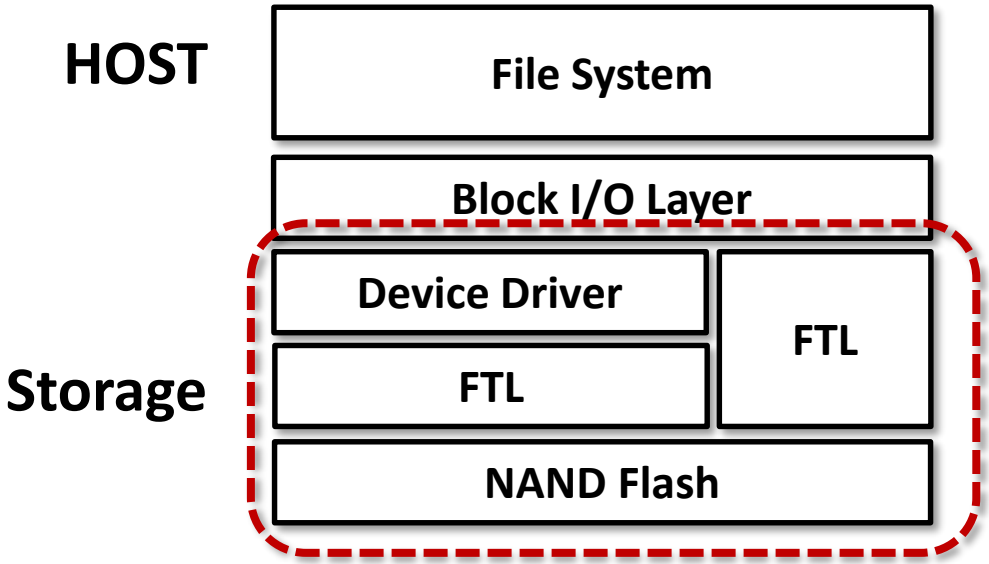
- **fsync**
  - Proper synchronization of dependent metadata
  - Small and random metadata updates
  - **Encode a full range of metadata area**

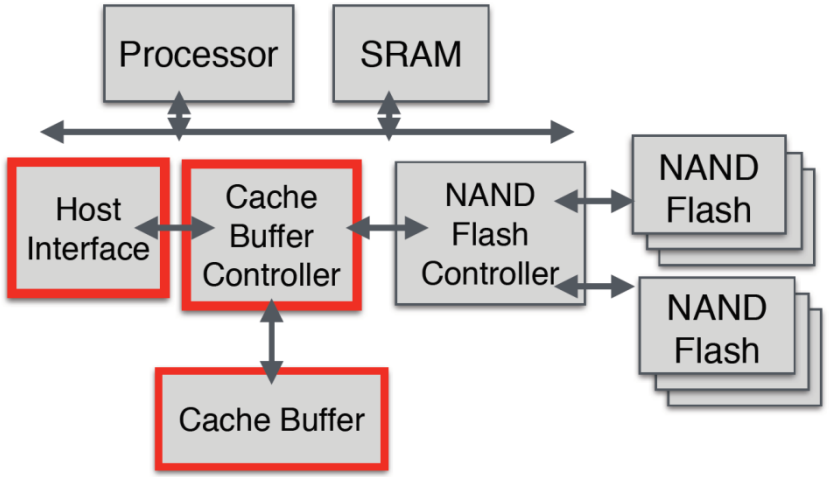


F2FS On-storage Layout (USENIX FAST '15)

Integration into a Storage Protocol

- **Open-channel SSD : BlueDBM**
  - Developed by MIT CSAIL
  - SSD emulator with allocated memory





Contents

- Motivation
- Problem and Solution
- Implementation
- **Performance Evaluation**
- Conclusion

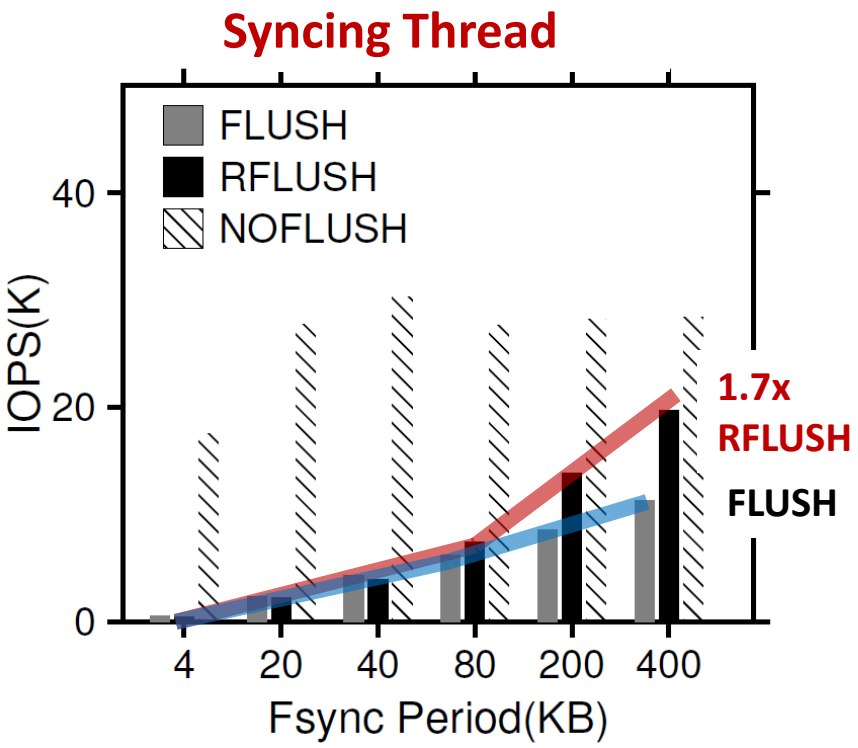
Performance Evaluation

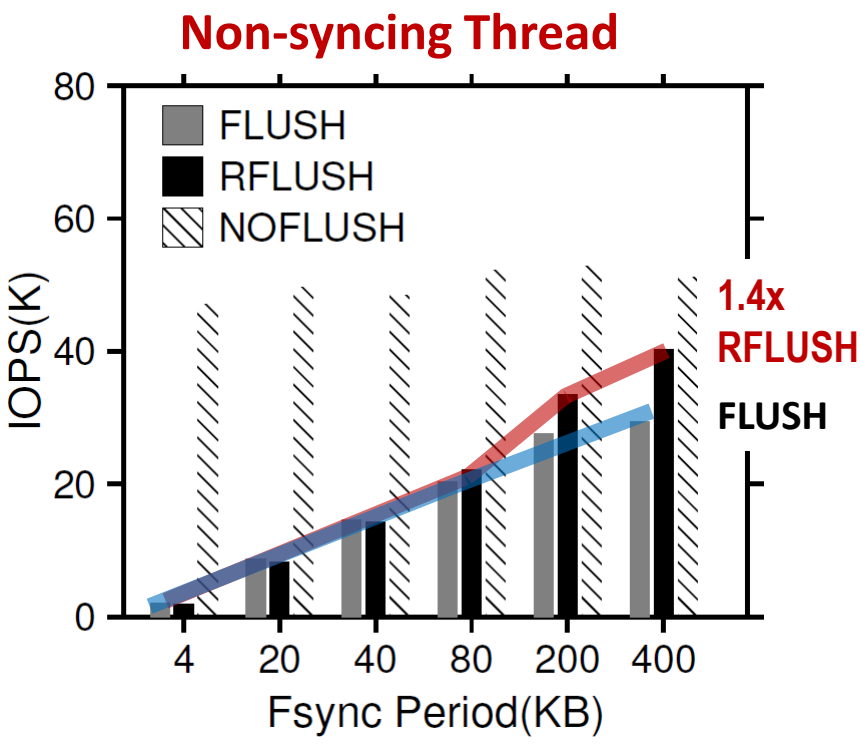
- Linux 4.7.2
- Benchmark – FIO, Linkbench, TPC-C, Fileserver
- SSD Platform Setup

Configuration	Settings
Page / Block size	4KB / 64 Pages
Read / Write Latency	100us / 1300us
Block Erase Latency	1.5ms
Data Transfer Latency	100us (for 4KB)
Overprovisioning Ratio	3%
SSD capacity	37 GB
In-storage Buffer	256 MB / 1 GB

FIO Benchmark - IOPS

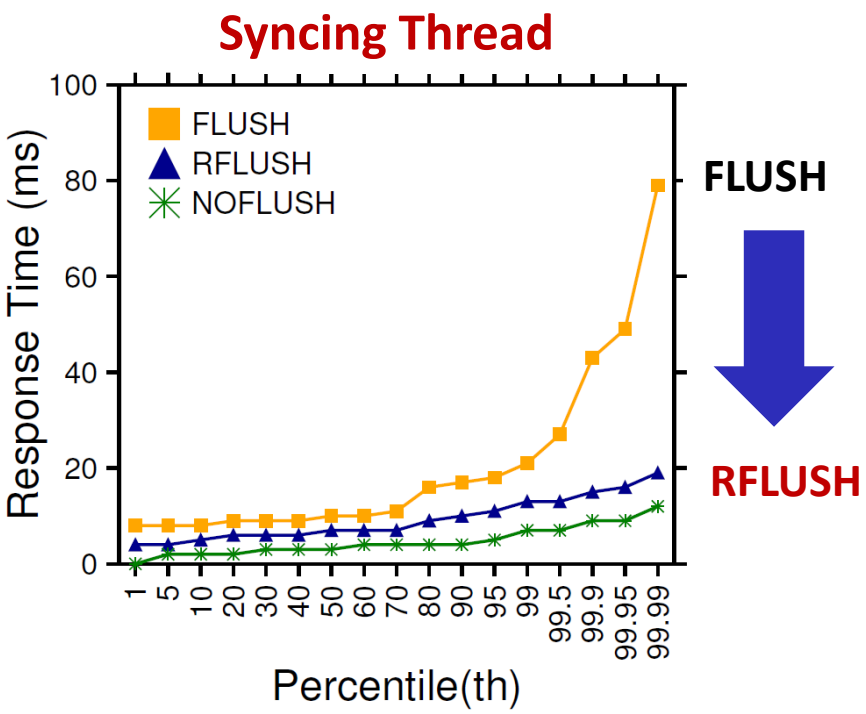
- Single synching thread : write + periodic sync
- Multiple non-syncing threads : write

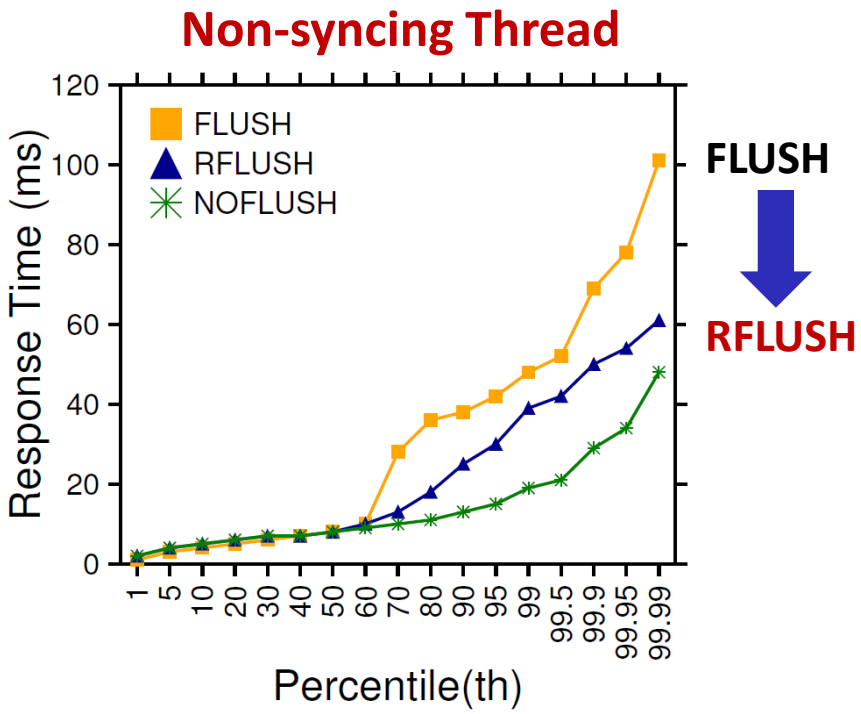




Tail Latency

- Response time(FIO Benchmark)

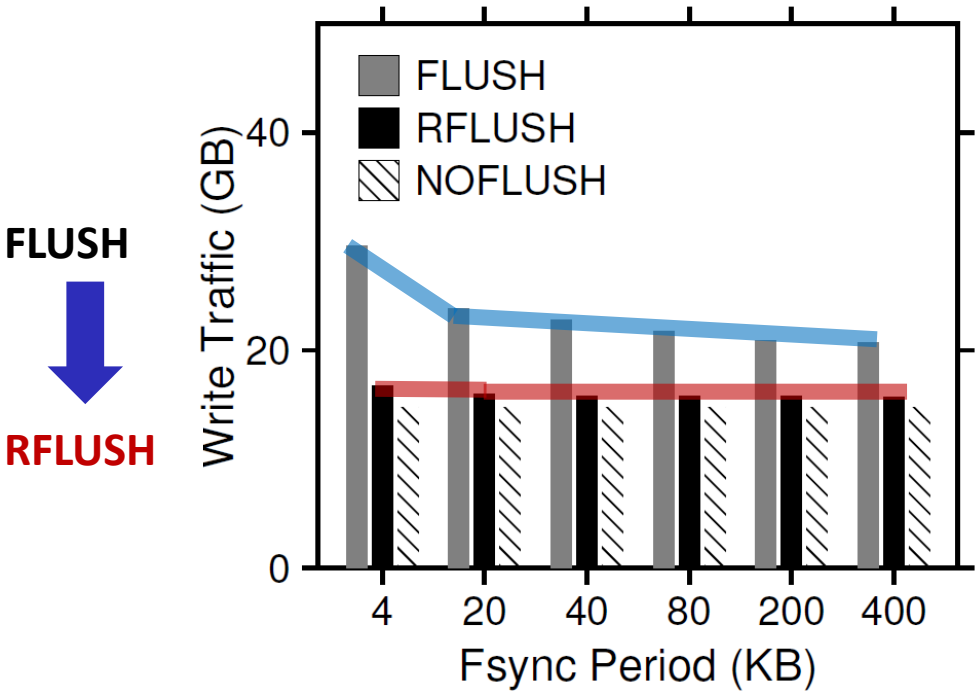




RFLUSH is critical to providing a **consistent performance** from a storage device

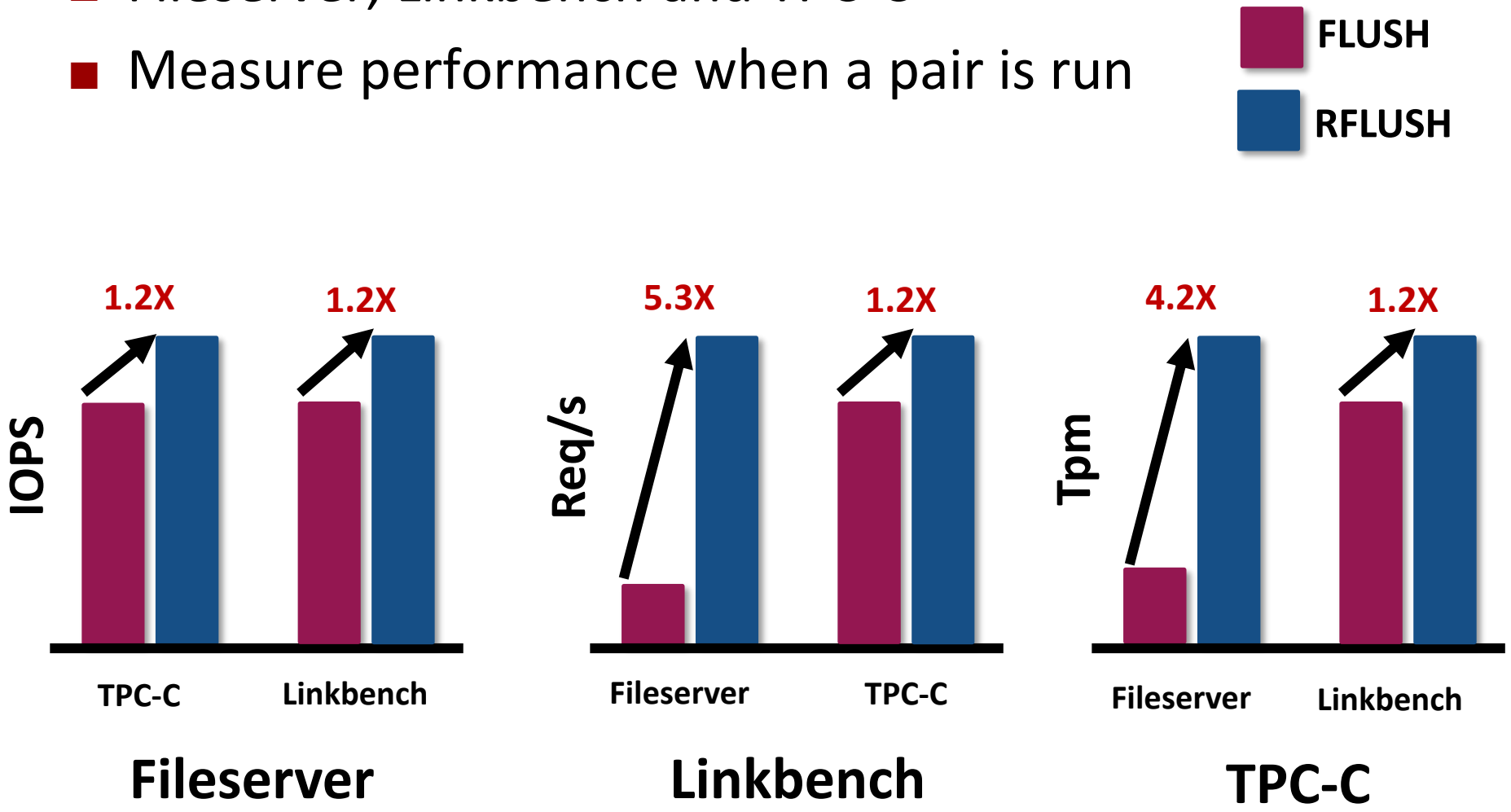
Write Traffic

- Reduce write traffic by 24% to 43%



Real-world Benchmark

- Fileserver, Linkbench and TPC-C
- Measure performance when a pair is run



Conclusion

- Raise the issue of the **negative performance** impact of a **lump-sum** approach to persisting buffered data within storage
- Present **RFLUSH** that allows fine-grained persistence control
- RFLUSH prototype is implemented in F2FS and open-channel SSD
- Increase IOPS up to 5.3x, reduce write traffic by 43%

Thank you

Q&A