Application-Managed Flash

Sungjin Lee, Ming Liu, Sangwoo Jun, and Shuotao Xu, Jihong Kim, Arvind

14th USENIX Conference on File and Storage Technologies (FAST '16). 2016.

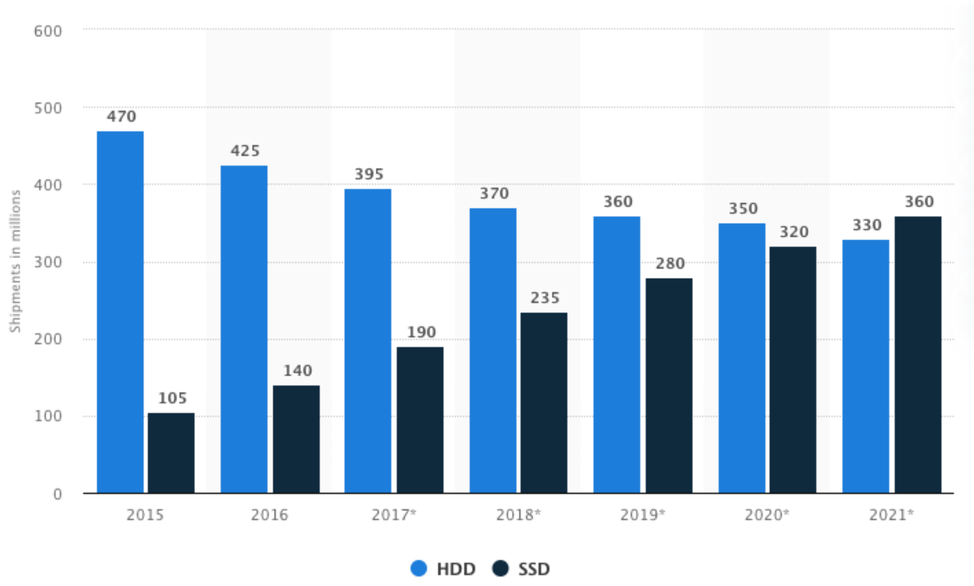
2021. 06. 02

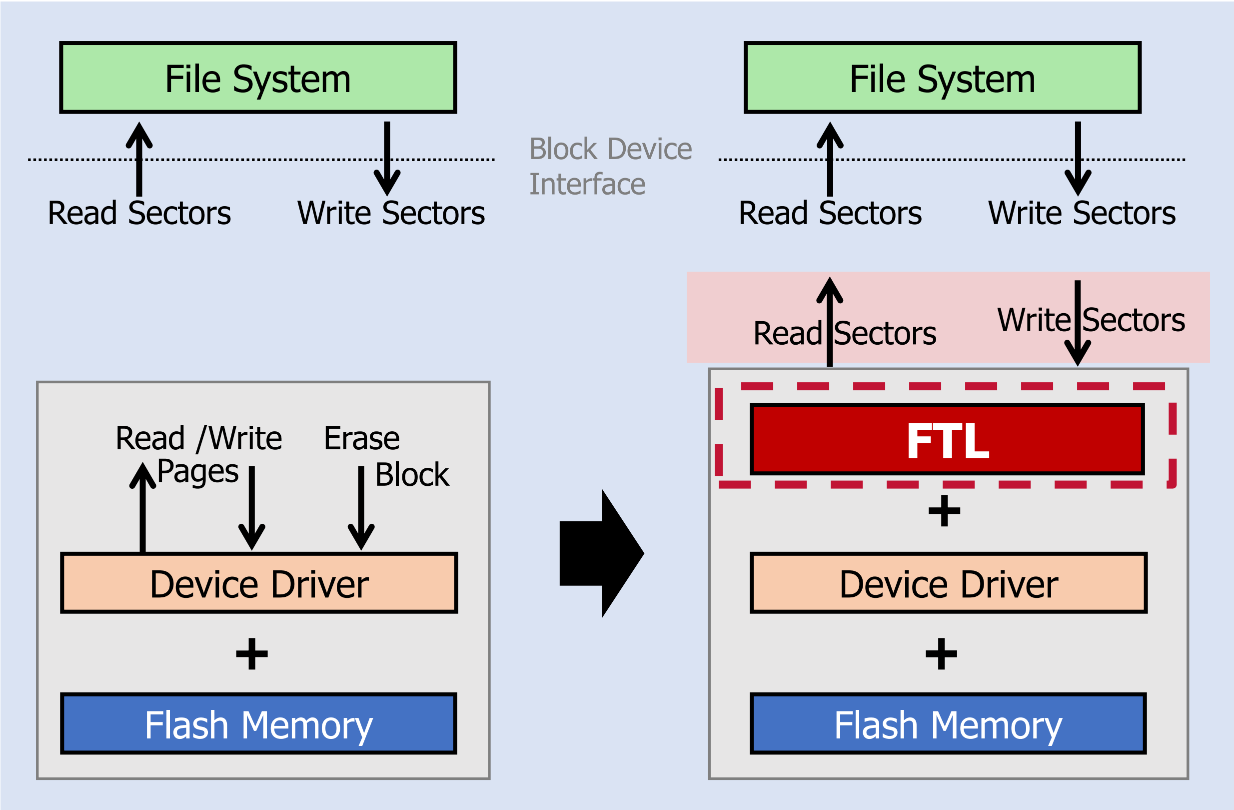
Presentation by Han, Yejin

hyj0225@dankook.ac.kr

Contents

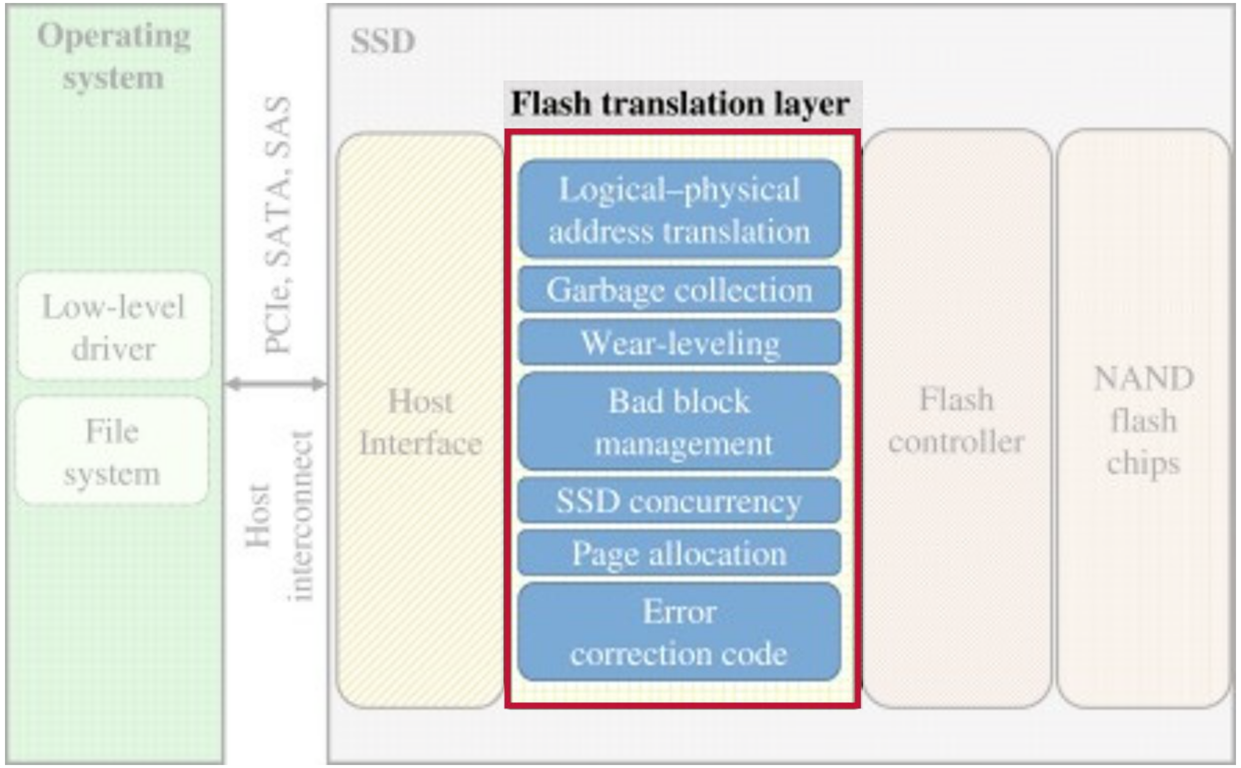
- NAND flash SSD become the preferred storage device in consumer electronics and data centers
- FTL provide an block I/O abstraction for interoperability with HDDs





What are the purposes of the FTL?

- Managing address translation, Garbage collection, wear-leveling, …



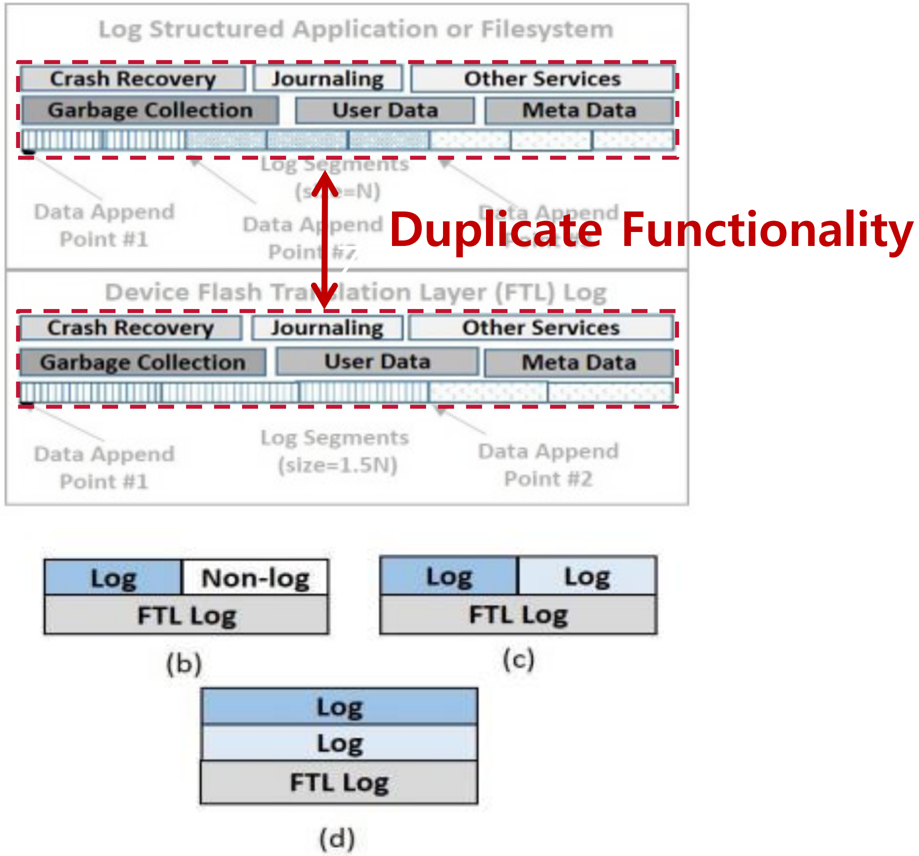
1) Require significant hardware resources
2) Extra I/Os for flash management (GC)
3) Host applications cannot predict the bahavior of flash storage

What are more issues of the FTL?

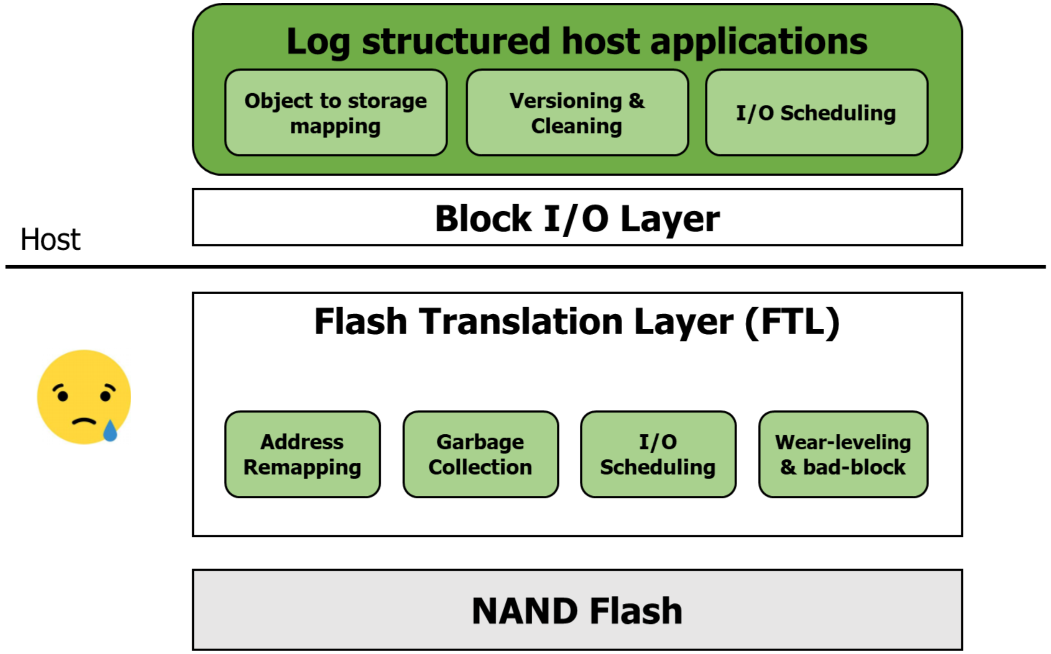
- Double logging between the FTL and the Host (double logging)
- Wastes hardware resource and increases write pressure to flash devices

Don't stack your Log on my Log

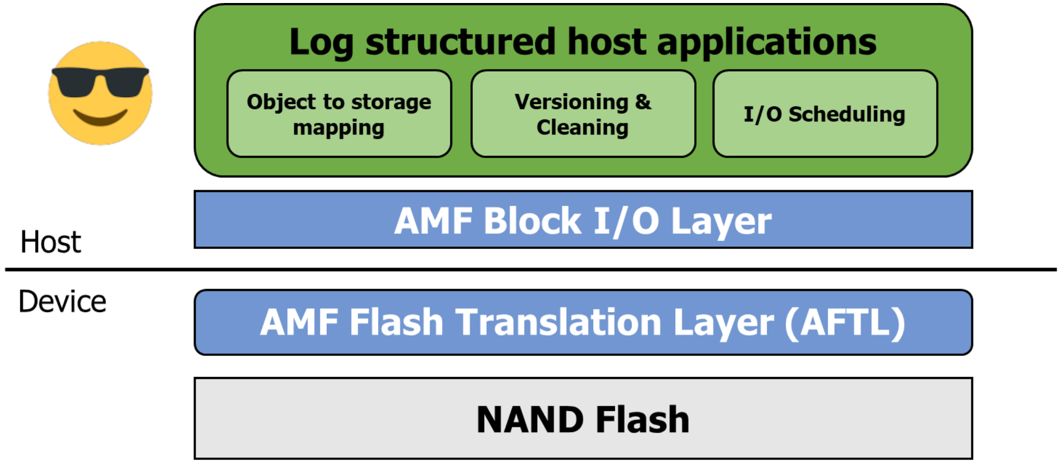
Jingpei Yang, Ned Plasson, Greg Gillis, Nisha Talagala, Swaminathan Sundararaman
SanDisk Corporation



Device-managed Flash



Application-managed Flash



moves flash management from the device to applications

Application-Managed Flash

1) Block I/O interface

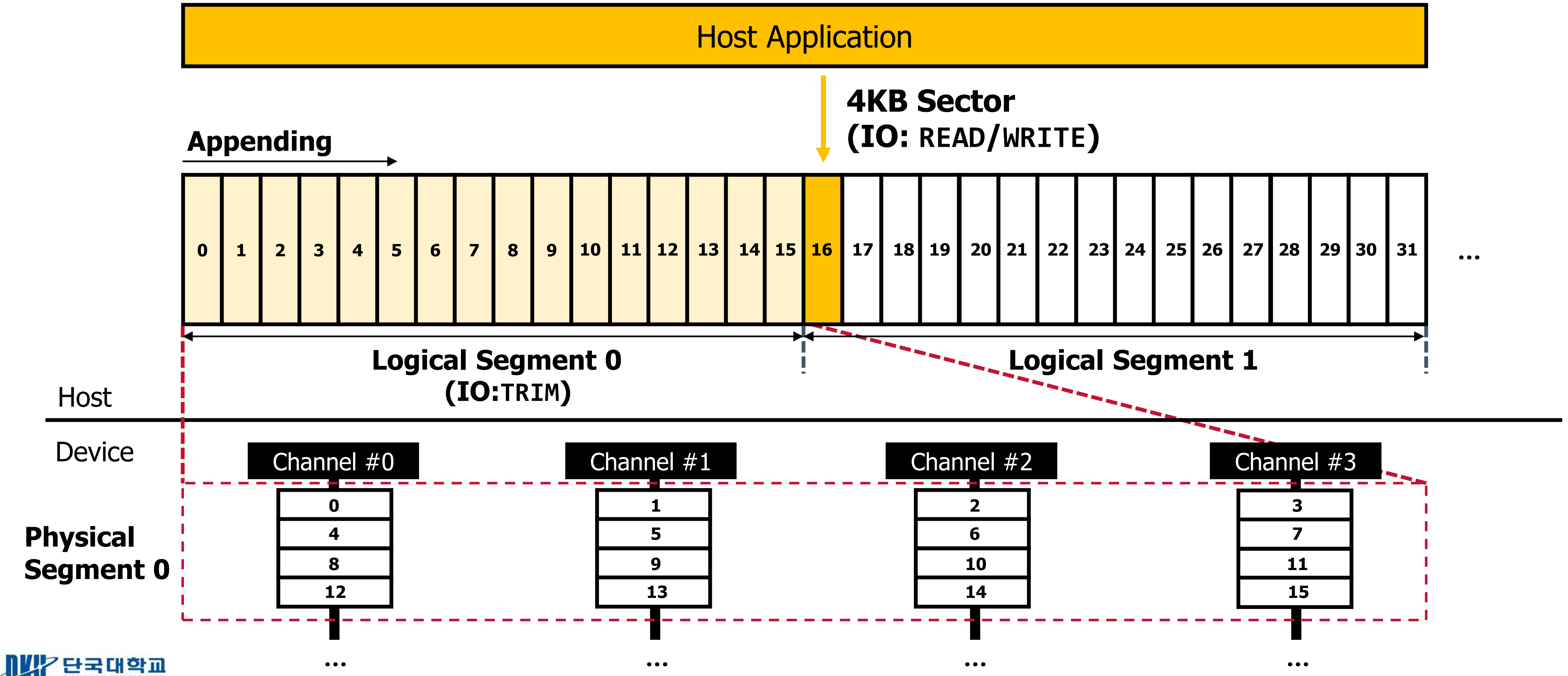
2) ALFS (AMF Log-structured File System)

3) AFTL (AMF Flash Translation Layer)

Embedded System Lab.

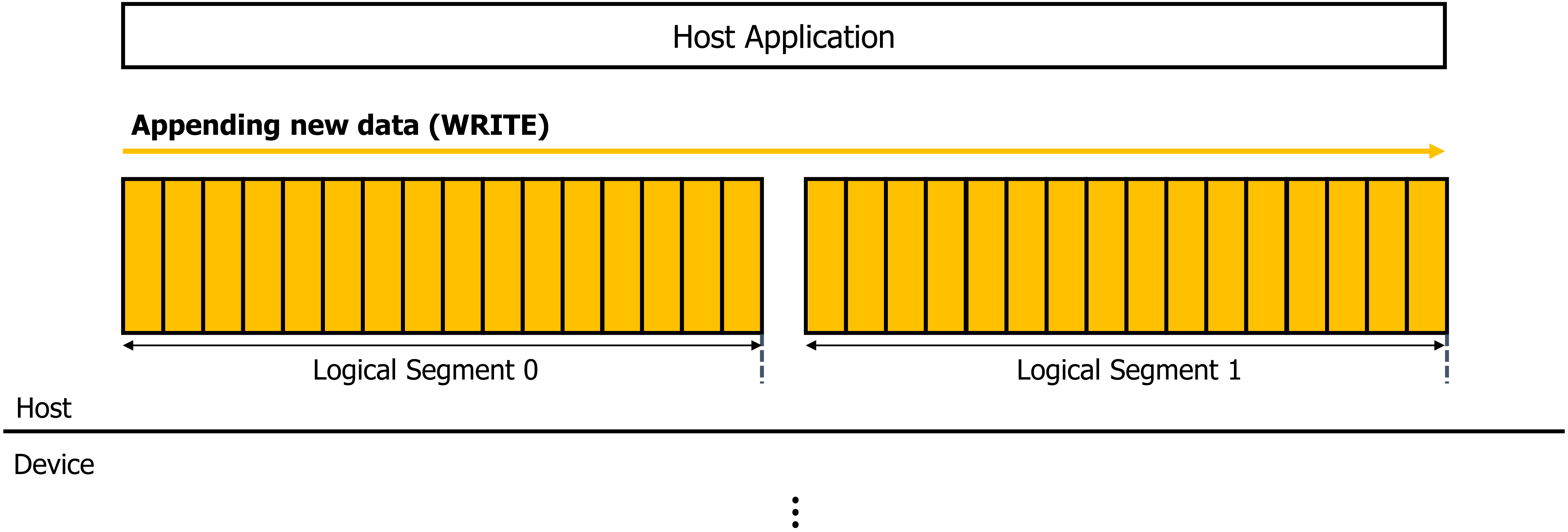
Block I/O Interface

- AMF I/O is based on conventional block I/O



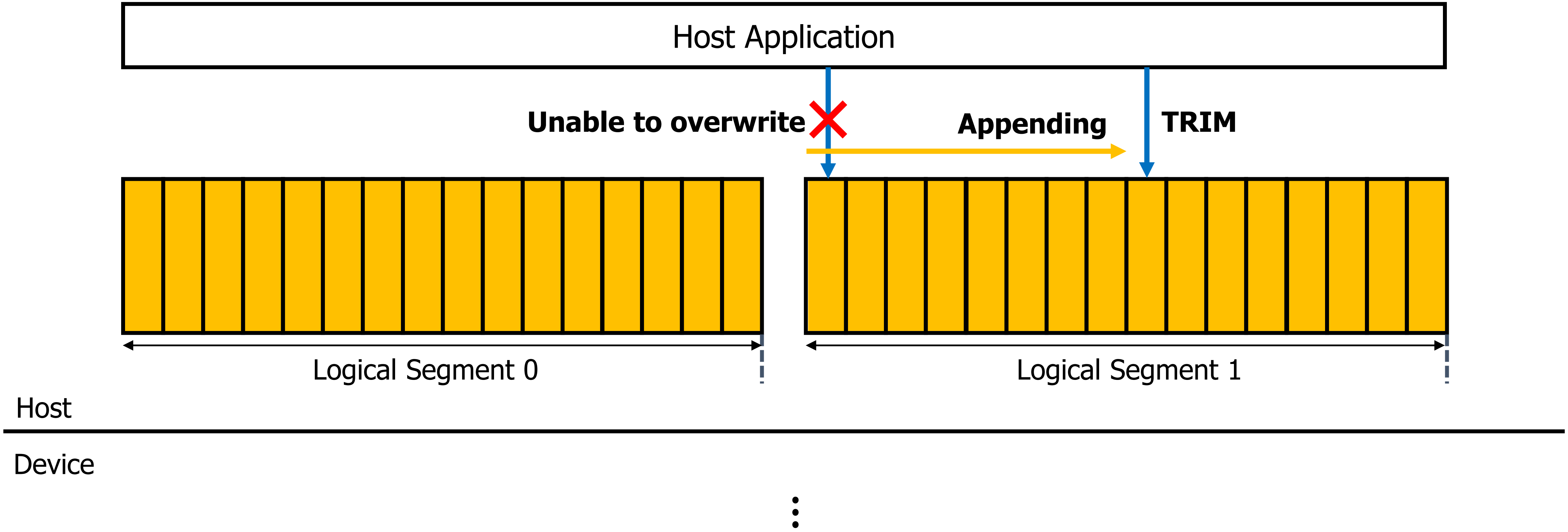
Block I/O Interface

- Append-only Write



Block I/O Interface

- Only sequential writes with no in-place updates
- Sectors can be reused after the segment has been deallocated by TRIM



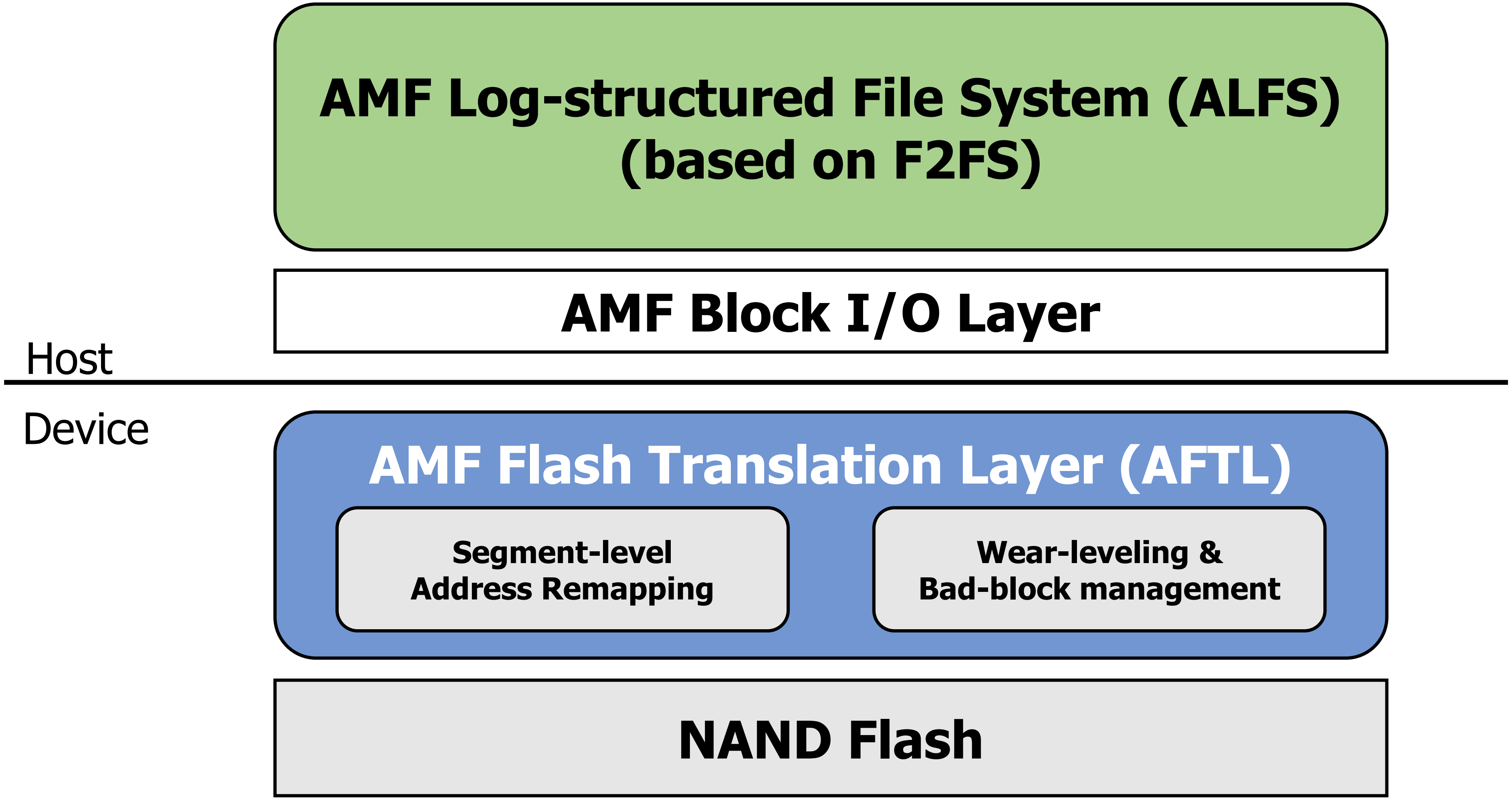
Application-Managed Flash

1) Block I/O interface

2) ALFS (AMF Log-structured File System)

3) AFTL (AMF Flash Translation Layer)

ALFS (AMF Log-structured File System)



ALFS (AMF Log-structured File System)

- File Layout and Operation

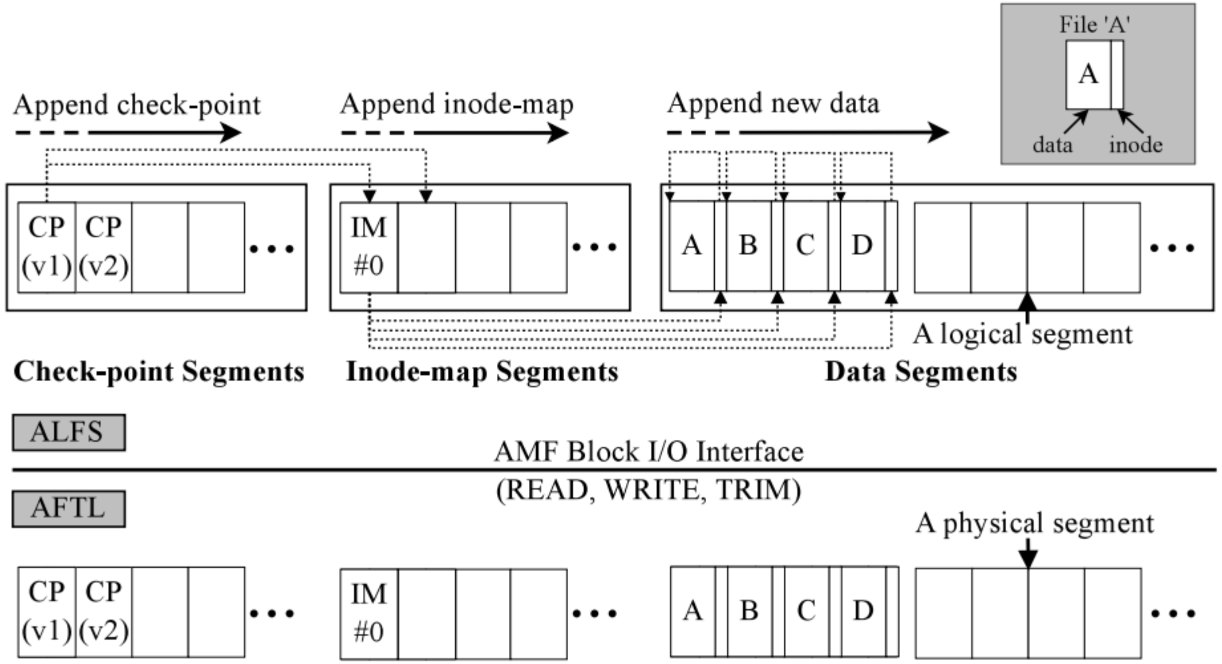
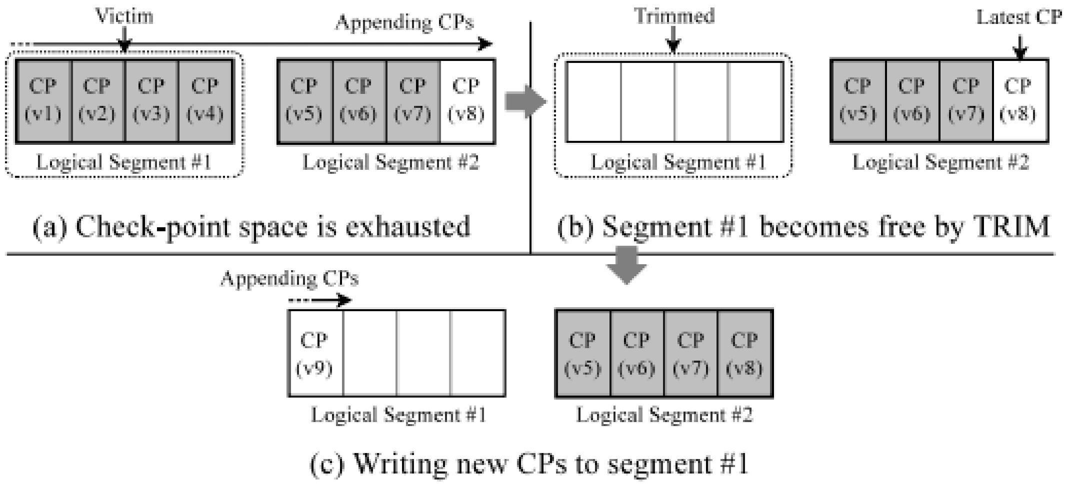


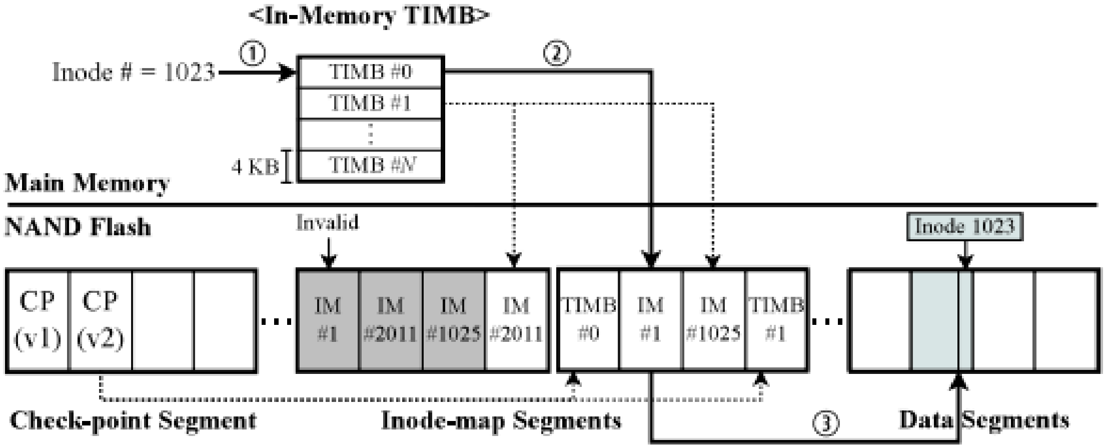
Figure 2: The upper figure illustrates the logical layout of ALFS. There is an initial check-point CP(v1). Four files are appended to data segments along with their inodes in the following order: A, B, C and D. Then, an inode map IM#0 is written which points to the locations of the inodes of the files. Finally, the check-point CP(v2) is written to check-point segments. The bottom figure shows the physical segments corresponding to the logical segments. The data layout of a logical segment perfectly aligns with its physical segment.

ALFS (AMF Log-structured File System)

- Check-point segments for quick mount and recovery
- Inode-map segments for fast searches of inodes



Check-point segment handling



Managing inode-map block

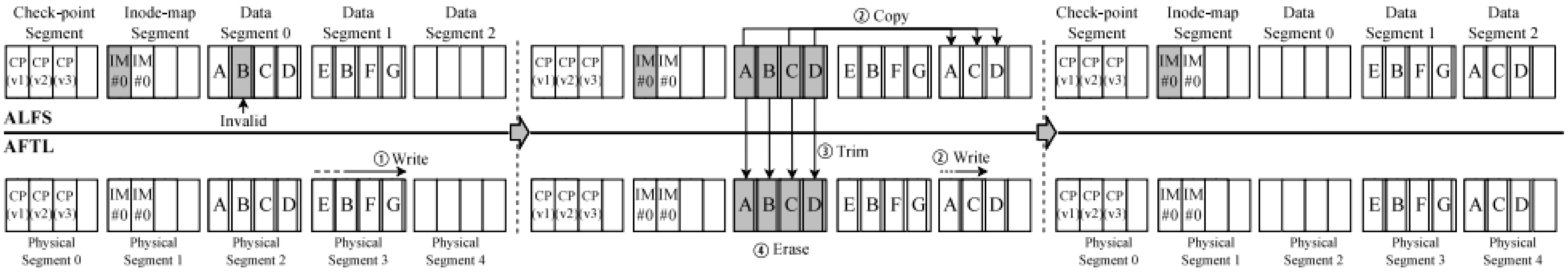
Embedded System Lab.

ALFS (AMF Log-structured File System)

- Comparison with conventional LFS

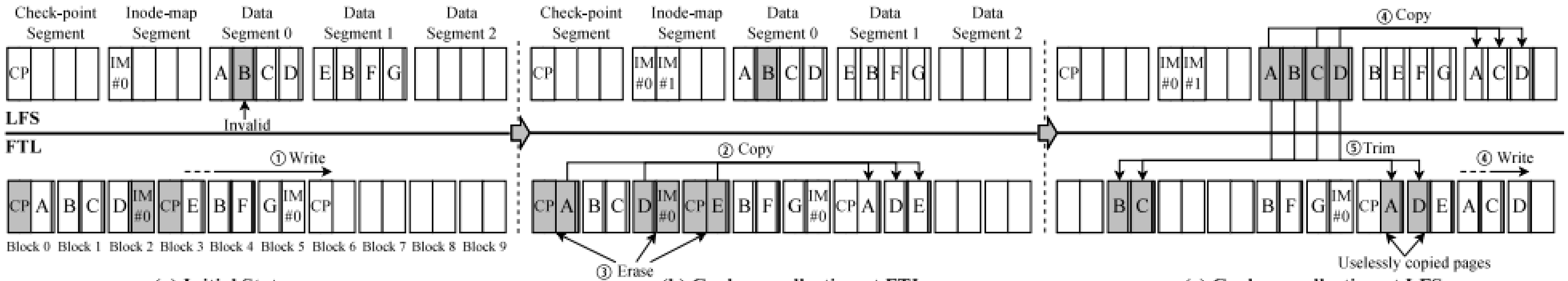
3 page copies + 2 block erasures

ALFS with AFTL



(a) Initial State (b) Garbage collection at ALFS (c) After garbage collection

LFS with FTL



(a) Initial State (b) Garbage collection at FTL (c) Garbage collection at LFS

6 page copies + 3 block erasures

Application-Managed Flash

1) Block I/O interface

2) ALFS (AMF Log-structured File System)

3) AFTL (AMF Flash Translation Layer)

AFTL (AMF flash Translation Layer)

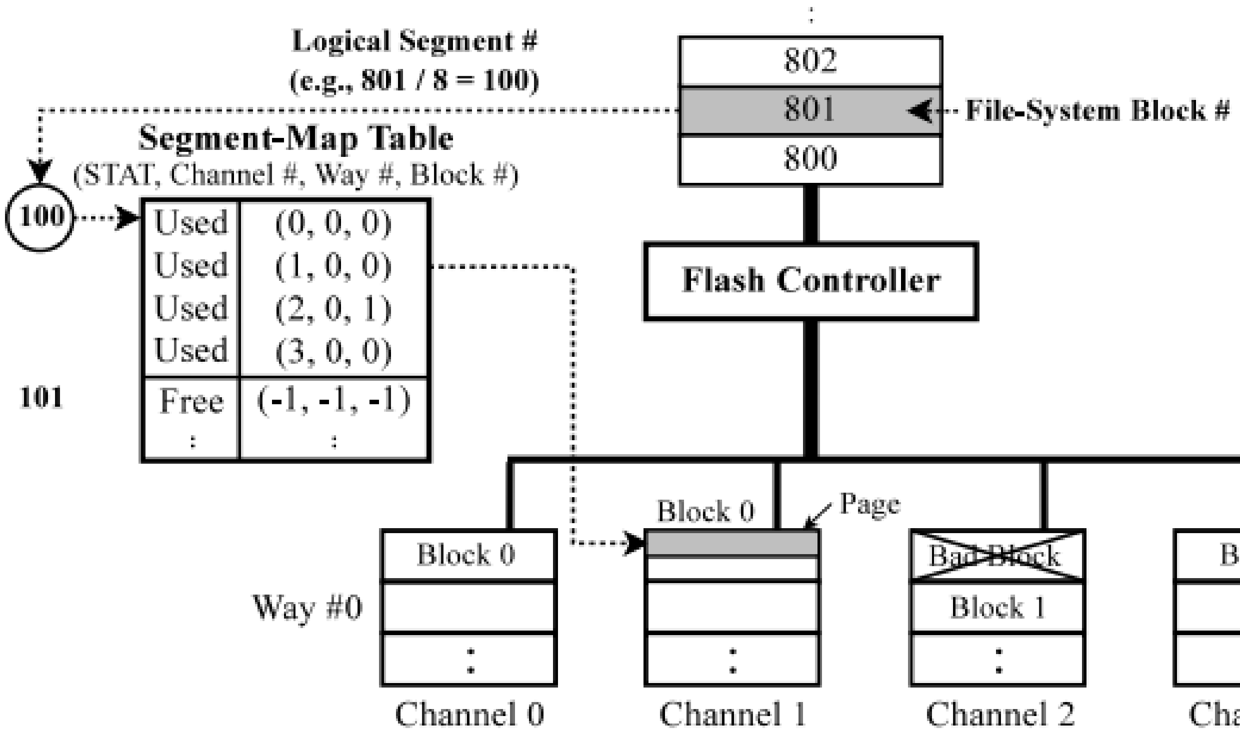


Figure 7: An example of how AFTL handles writes

Capacity	Block-level FTL	Hybrid FTL	Page-level FTL	AMF	
				AFTL	ALFS
512 GB 1 TB	4 MB 8 MB	96 MB 186 MB	512 MB 1 GB	4 MB 8 MB	5.3 MB 10.8 MB

Table 2: A summary of memory requirements

AFTL maintains a smaller mapping table than the page-level FTL

Data structures including TIMB need tiny amount of host DRAM

Performance measurment

- EXT4: only the FTL performed GC
- F2FS: both F2FS and the FTL performed GC
- AMF: only ALFS performed GC

- PFTL: no extra swapping I/Os
- AFTL: no extra swapping I/Os
- DFTL: incurred extra I/Os

4. Evaluation

Category	Workload	Description
File System	FIO	A synthetic I/O workload generator
	Postmark	A small and metadata intensive workload

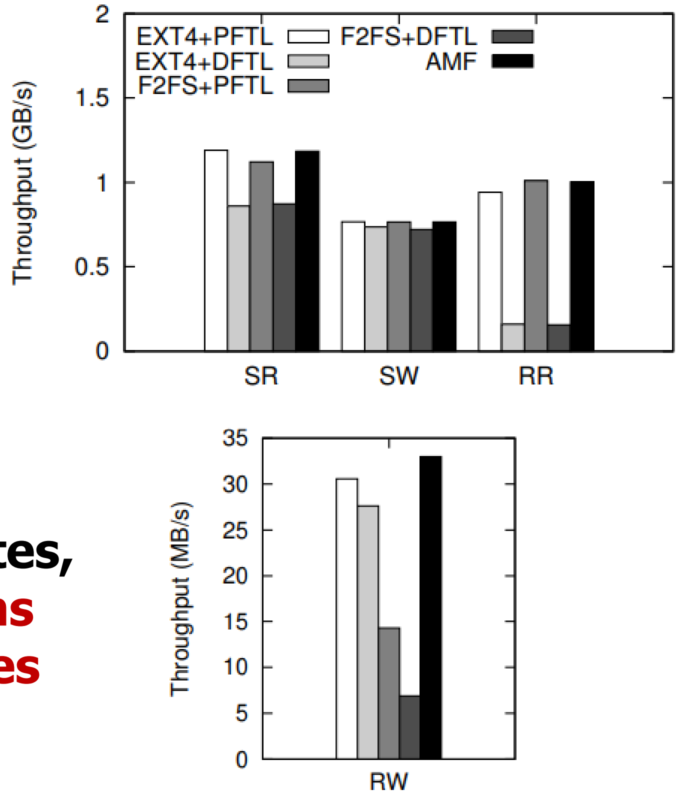


Figure 9: Experimental results with FIO

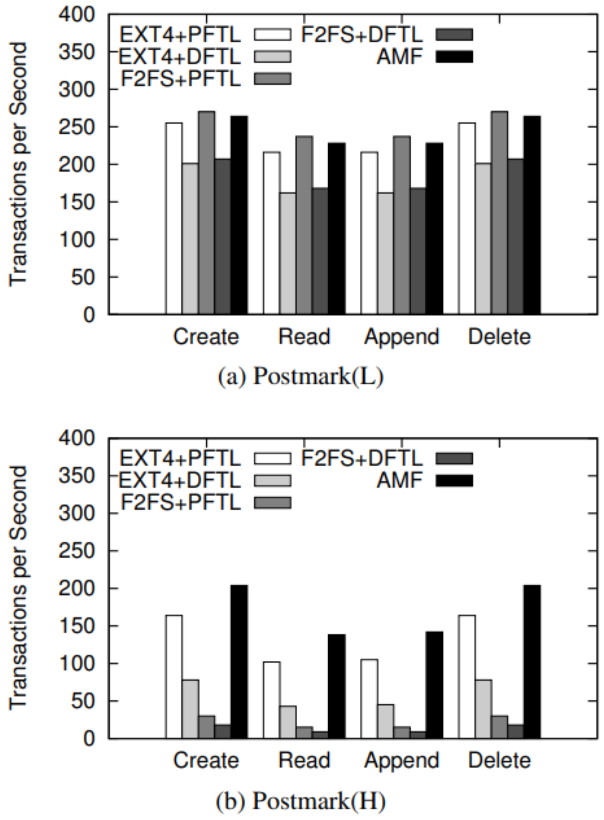


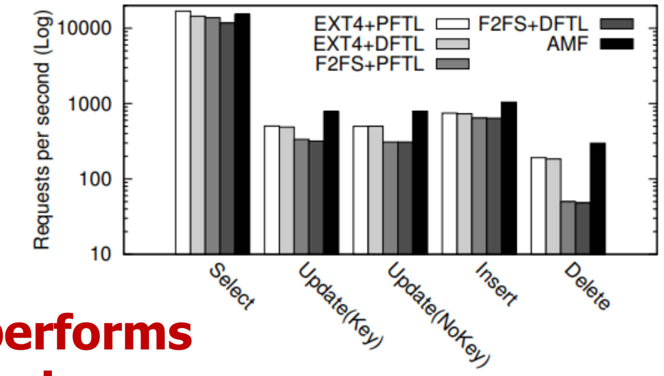
Figure 10: Experimental results with Postmark

For random writes,
AMF outperforms all other schemes

For the heavy workload,
AMF achieves the best performance

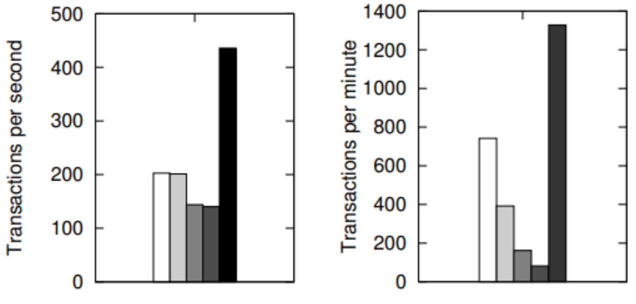
Database	Non-Trans	A non-transactional DB workload
	OLTP	An OLTP workload
	TPC-C	A TPC-C workload

Hadoop	DFSIO	A HDFS I/O throughput test application
	TeraSort	A data sorting application
	WordCount	A word count application



AMF outperforms all other schemes

(a) Non-Trans



(b) OLTP (c) TPC-C

Figure 11: Experimental results with database apps.

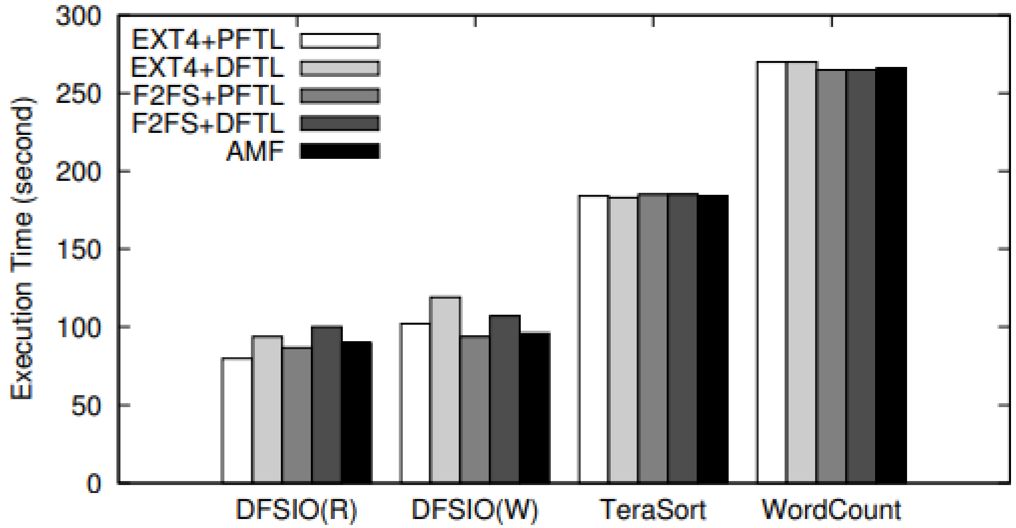


Figure 12: Experimental results with Hadoop apps.

All five storage configurations show similar performance

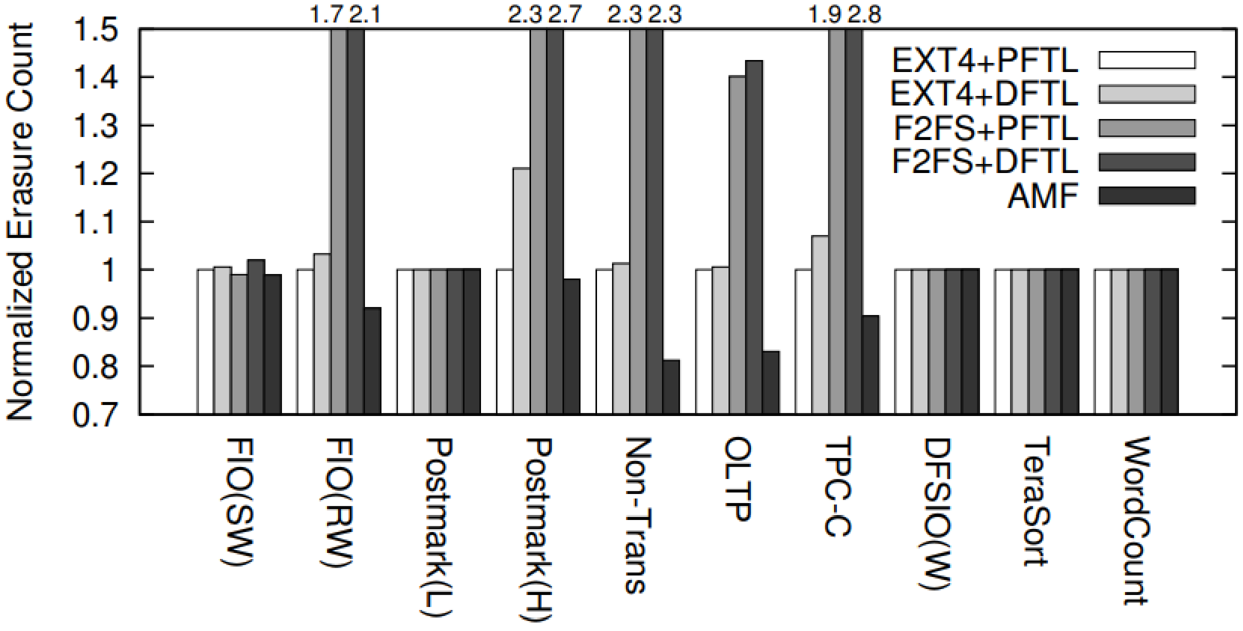


Figure 13: Erasure operations normalized to EXT4+PFTL

AMF incurs 28% fewer erase operations overall compared to F2FS+DFTL

The CPU utilization of AMF is similar to F2FS and EXT4

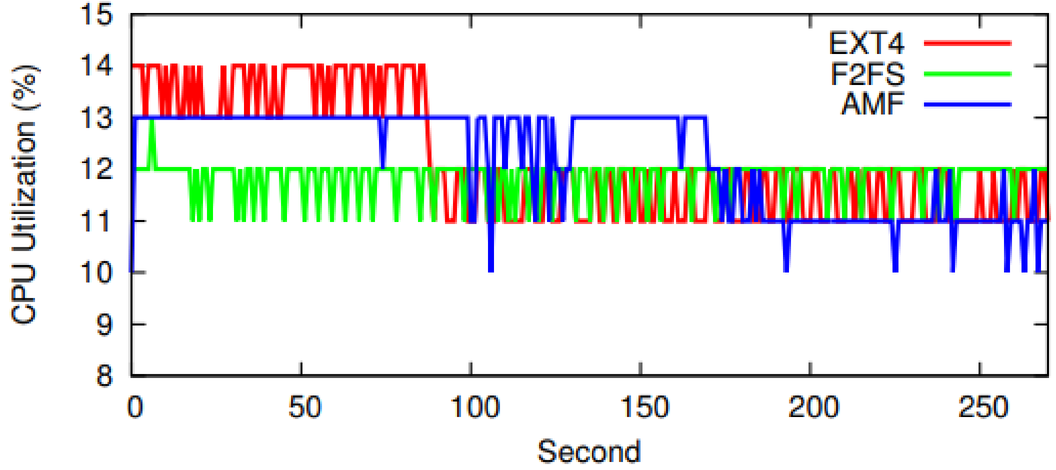


Figure 15: CPU utilization (%)

AMF has the shortest I/O response times with small fluctuations

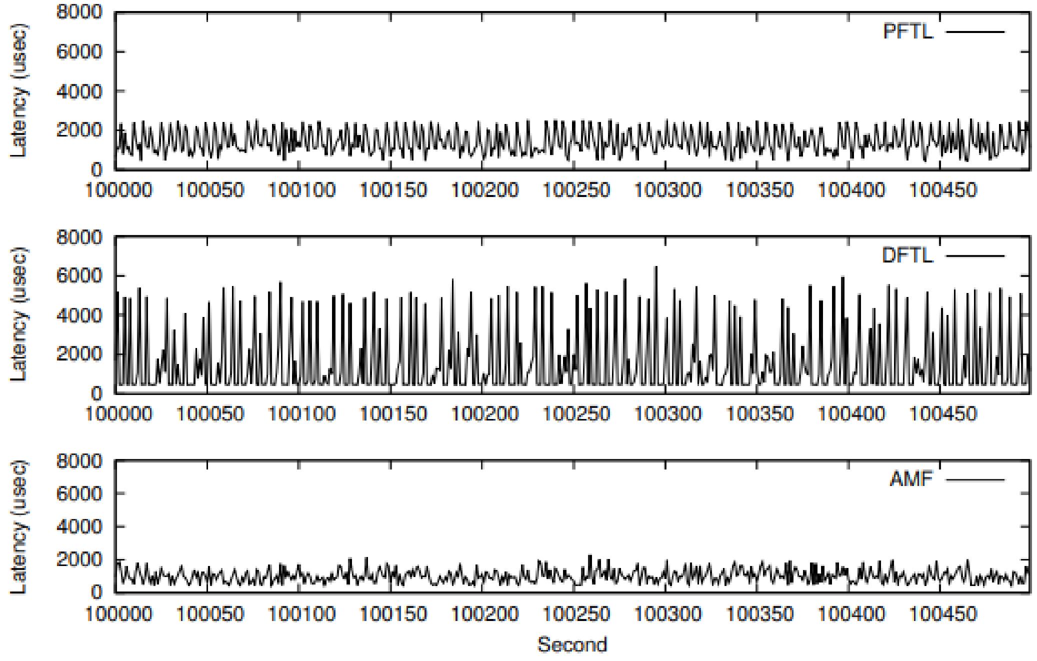


Figure 16: Write latency (μsec)

AMF

- **New block I/O interface exposing flash storage as append-only segments**
- **Implementing new FTL scheme (AFTL) and a new file system (ALFS)**
- **Reducing DRAM in the flash controller by 128X, impoving performa nce of the file system by 80%**

https://github.com/chamdoo/bdbm_drv.git
https://github.com/sangwoojun/bluedbm.git

Application-Managed Flash

Sungjin Lee, Ming Liu, Sangwoo Jun, and Shuotao Xu, Jihong Kim, Arvind

14th USENIX Conference on File and Storage Technologies (FAST '16). 2016.

Thank You!

2021. 06. 02

Presentation by Han, Yejin
hyj0225@dankook.ac.kr