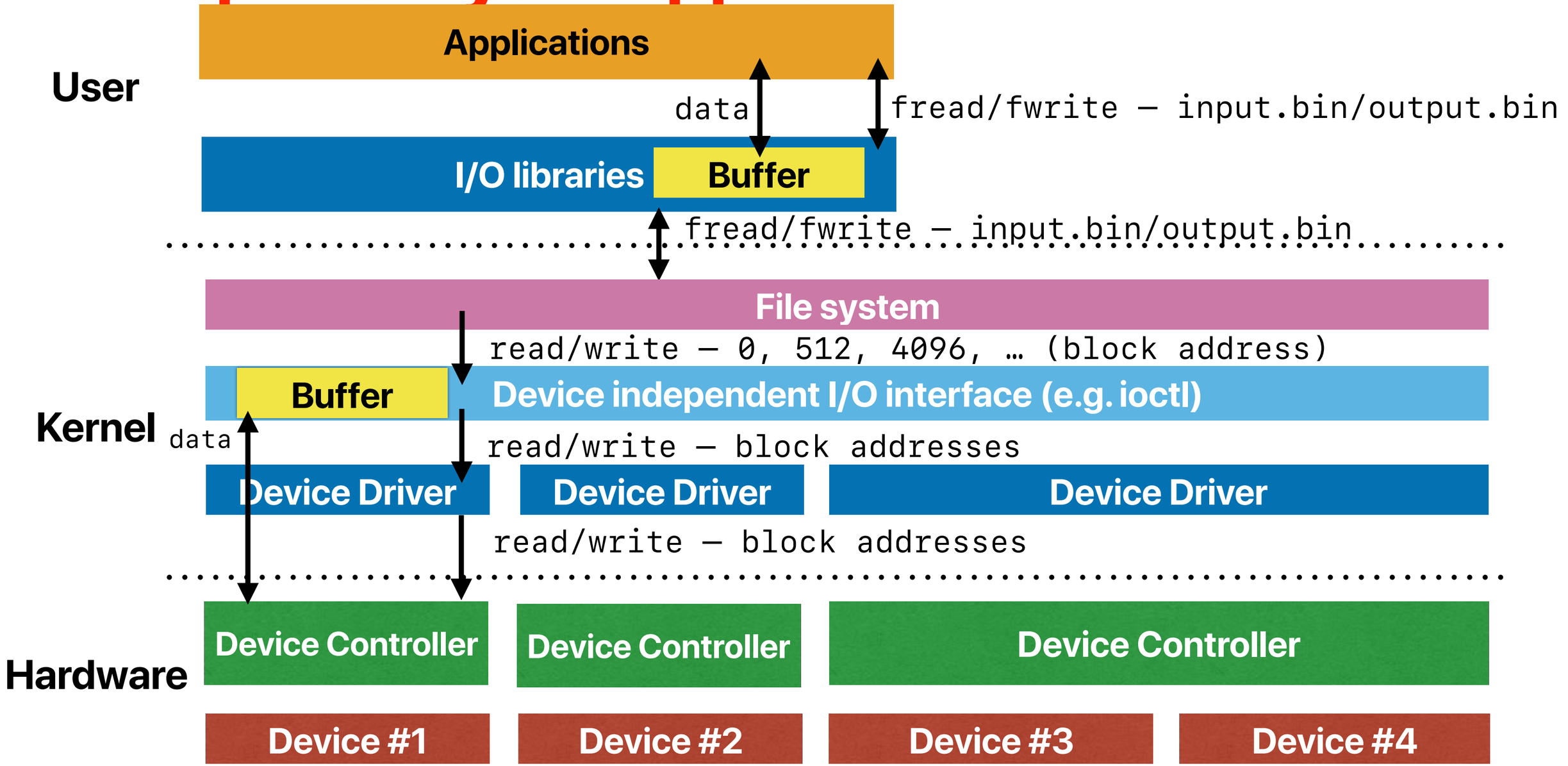
File Systems & The Era of Flash-based SSD

Hung-Wei Tseng

Recap: Abstractions in operating systems

- Process — the abstraction of a von Neumann machine
- Thread — the abstraction of a processor
- Virtual memory — the abstraction of memory
- File system — the abstraction of space/location on a storage device, the storage device itself, as well as other peripherals

Recap: How your application reaches H.D.D.



Recap: what BSD FFS proposes?

- Cylinder groups — improve spread-out data locations
- Larger block sizes — improve bandwidth and file sizes
- Fragments — improve low space utilization due to large blocks
- Allocators — address device oblivious
- New features
  - long file names
  - file locking
  - symbolic links
  - renaming
  - quotas

Recap: Performance of FFS

Table IIa. Reading Rates of the Old and New UNIX File Systems

Type of file system	Processor and bus measured	Speed (Kbytes/s)	Read bandwidth %	% CPU
Old 1024	750/UNIBUS	29	29/983 3	11
New 4096/1024	750/UNIBUS	221	221/983 22	43
New 8192/1024	750/UNIBUS	233	233/983 24	29
New 4096/1024	750/MASSBUS	466	466/983 47	73
New 8192/1024	750/MASSBUS	466	466/983 47	54

not the case for old FS

writes in FFS are slower than reads

Table IIb. Writing Rates of the Old and New UNIX File Systems

Type of file system	Processor and bus measured	Speed (Kbytes/s)	Write bandwidth %	% CPU
Old 1024	750/UNIBUS	48	48/983 5	29
New 4096/1024	750/UNIBUS	142	142/983 14	43
New 8192/1024	750/UNIBUS	215	215/983 22	46
New 4096/1024	750/MASSBUS	323	323/983 33	94
New 8192/1024	750/MASSBUS	466	466/983 47	95

CPU load is fine given that UFS is way too slow!

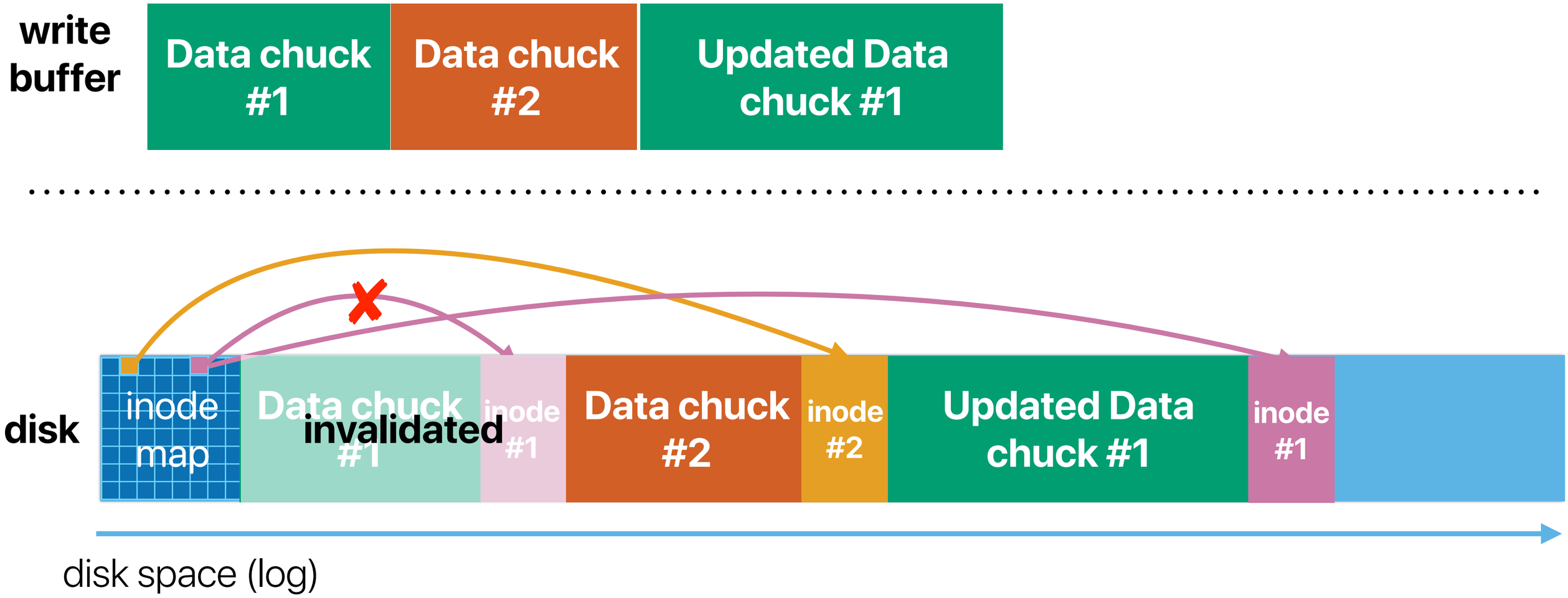
Recap: Why LFS?

- Writes will dominate the traffic between main memory and disks — Unix FFS is designed under the assumption that a majority of traffic is read
  - Who is wrong? **UFS is published in 1984**
  - As system memory grows, frequently read data can be cached efficiently
  - Every modern OS aggressively caches — use "free" in Linux to check
- Gaps between sequential access and random access
- Conventional file systems are not RAID aware

Recap: What does LFS propose?

- Buffering changes in the system main memory and commit those changes sequentially to the disk with fewest amount of write operations

Recap: LFS in motion



Segment cleaning/Garbage collection

- Reclaim invalidated segments in the log once the latest updates are checkpointed
- Rearrange the data allocation to make continuous segments
- Must reserve enough space on the disk
  - Otherwise, every writes will trigger garbage collection
  - Sink the write performance

Lessons learned

- Performance is closely related to the underlying architecture
  - Old UFS performs poorly as it ignores the nature of hard disk drives
  - FFS allocates data to minimize the latencies of disk accesses
- As architectural/hardware changes the workload, so does the design philosophy of the software
  - FFS optimizes for reads
  - LFS optimizes for writes — because we have larger memory now

Outline

- Modern file systems
- Flash-based SSDs and eNVy: A non-volatile, main memory storage system
- Don’t stack your log on my log

Modern file system design — Extent File Systems

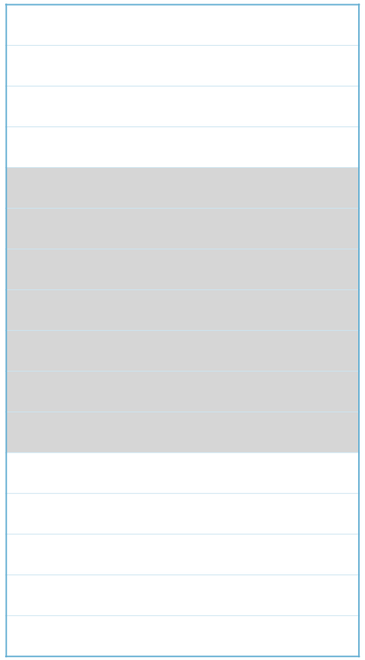
Extent file systems — ext2, ext3, ext4

- Basically optimizations over FFS + Extent + Journaling (write-ahead logs)

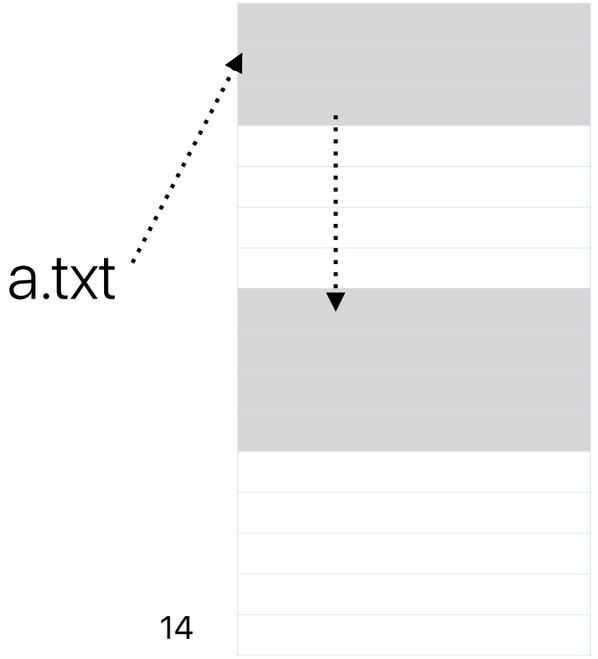
How do we allocate space?

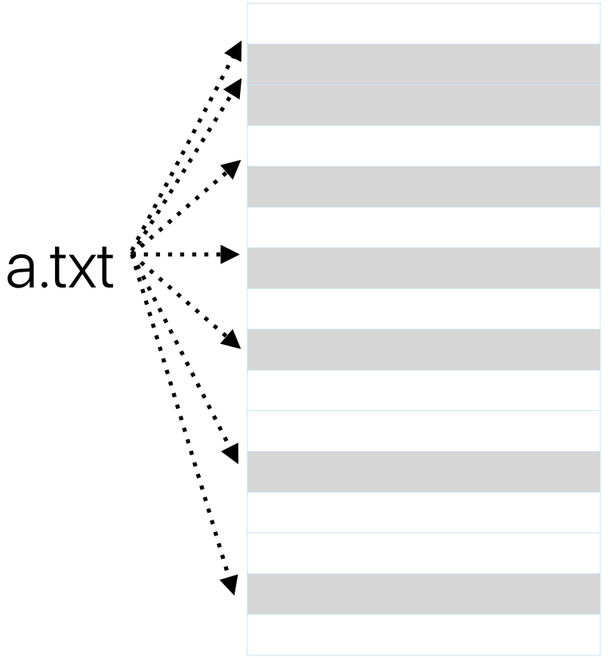
- Contiguous: the file resides in continuous addresses

a.txt



- Non-contiguous: the file can be anywhere
- Extents: the file resides in several group of smaller continuous address

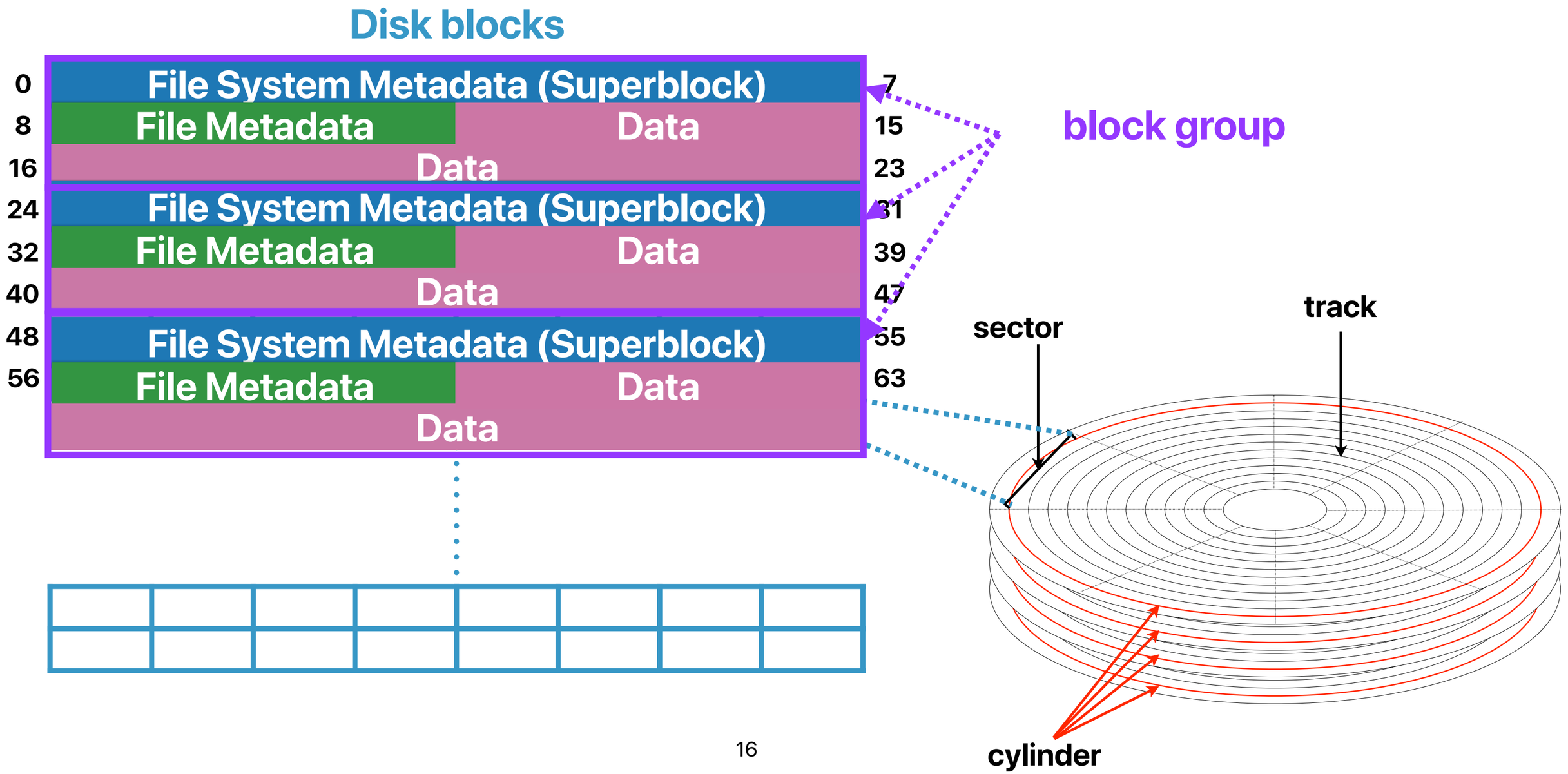




Using extents in inodes

- Contiguous blocks only need a pair <start, size> to represent
- Improve random seek performance
- Save inode sizes
- Encourage the file system to use contiguous space allocation

How ExtFS use disk blocks



Write-ahead log

- Basically, an idea borrowed from LFS to facilitate writes and crash recovery
- Write to log first, apply the change after the log transaction commits
  - Update the real data block after the log writes are done
  - Invalidate the log entry if the data is presented in the target location
  - Replay the log when crash occurs

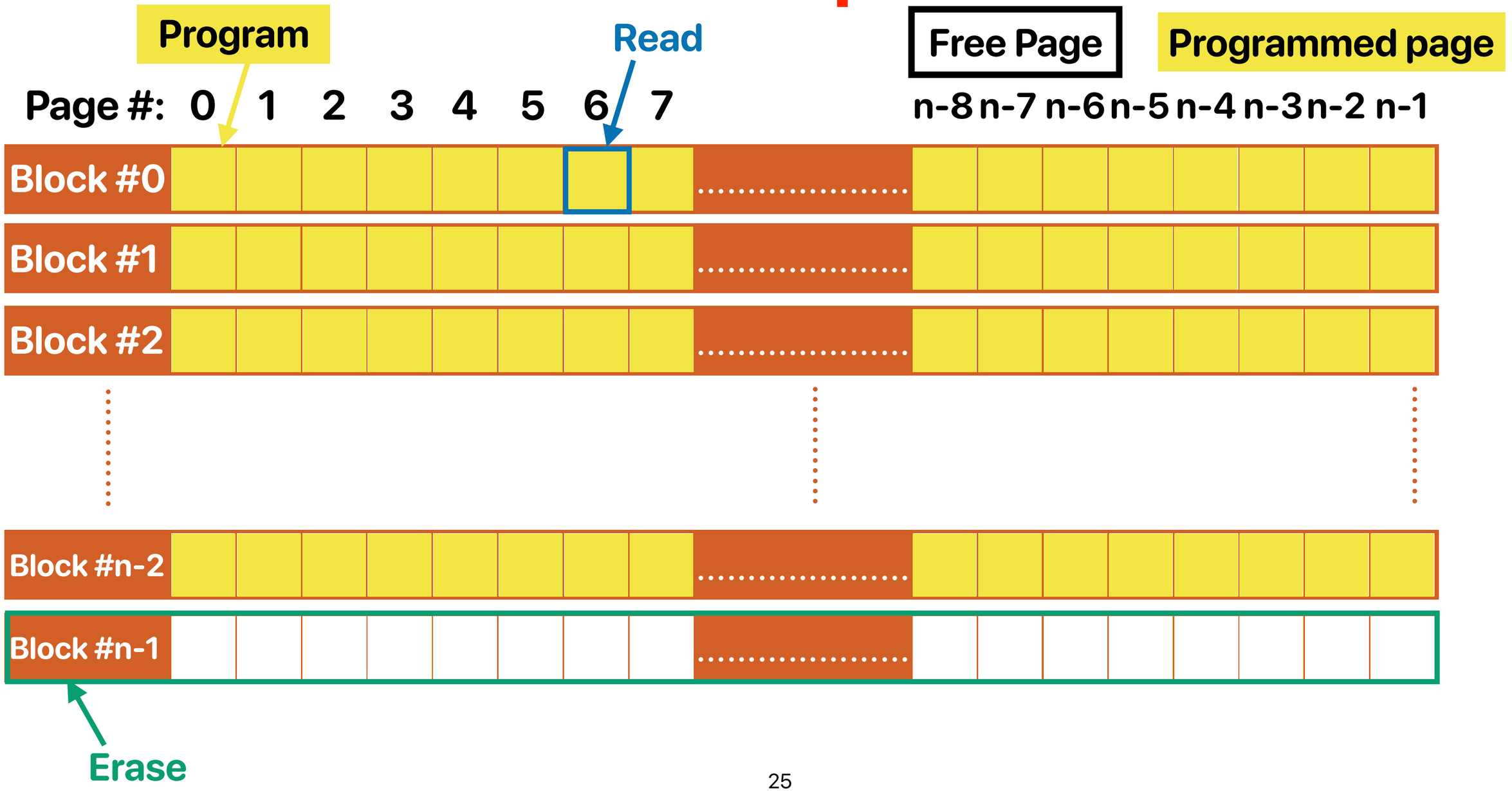
Flash-based SSDs
and
eNVy: A non-volatile, main memory storage system

Michael Wu and Willy Zwaenepoel
Rice University

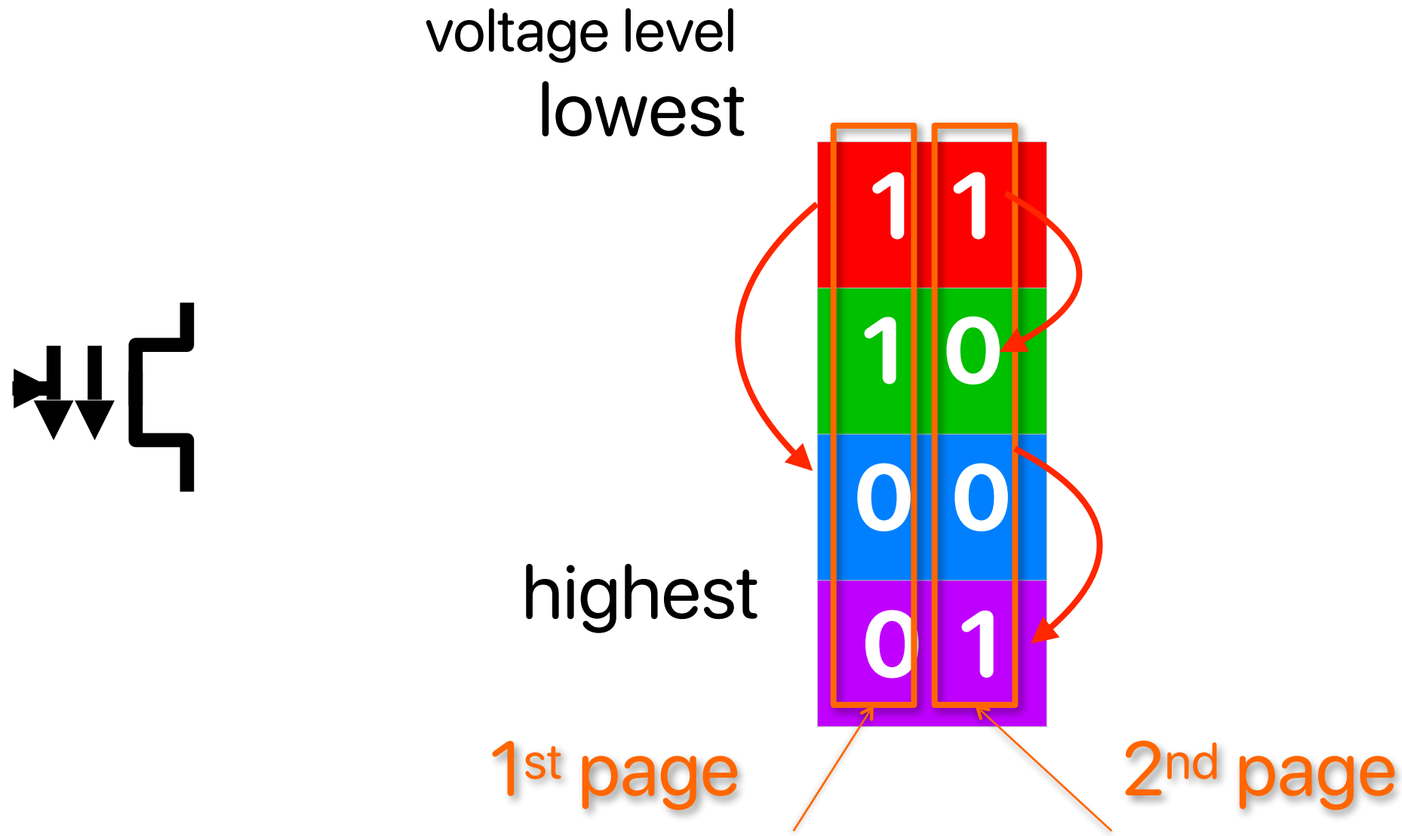
Flash memory: eVNy and now

	Modern SSDs	eNVy
Technologies	NAND	NOR
Read granularity	Pages (4K or 8K)	Supports byte accesses
Write/program granularity	Pages (4K or 8K)	Supports byte accesses
Write once?	Yes	Yes
Erase	In blocks (64 ~ 384 pages)	64 KB
Program-erase cycles	3,000 - 10,000	~ 100,000

Basic flash operations

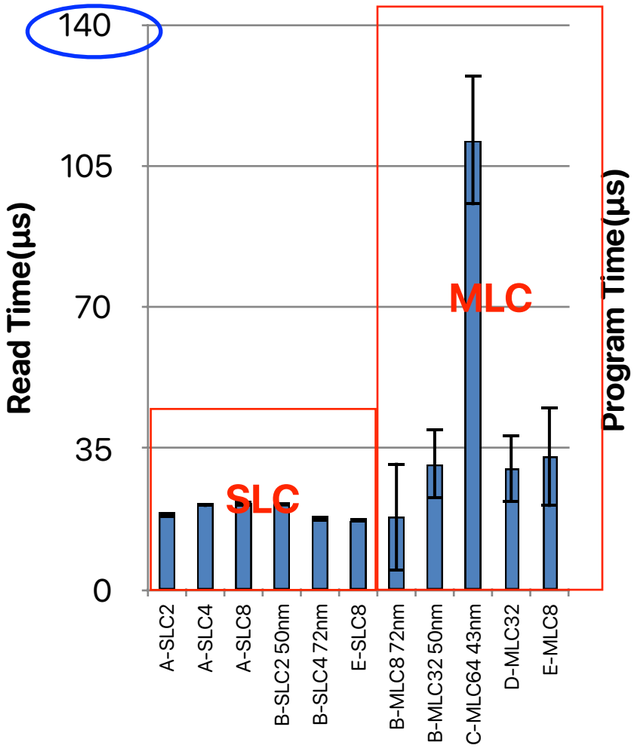


Programming MLC (multi-level cell)

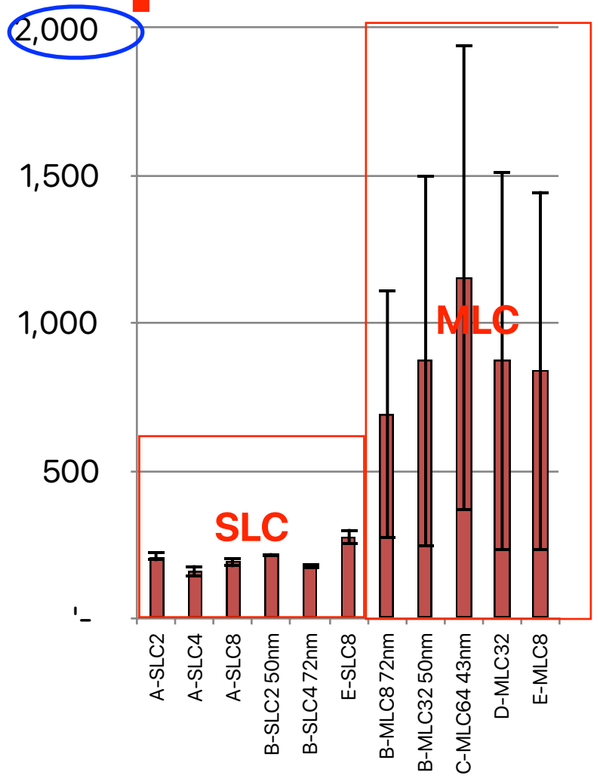


Flash performance

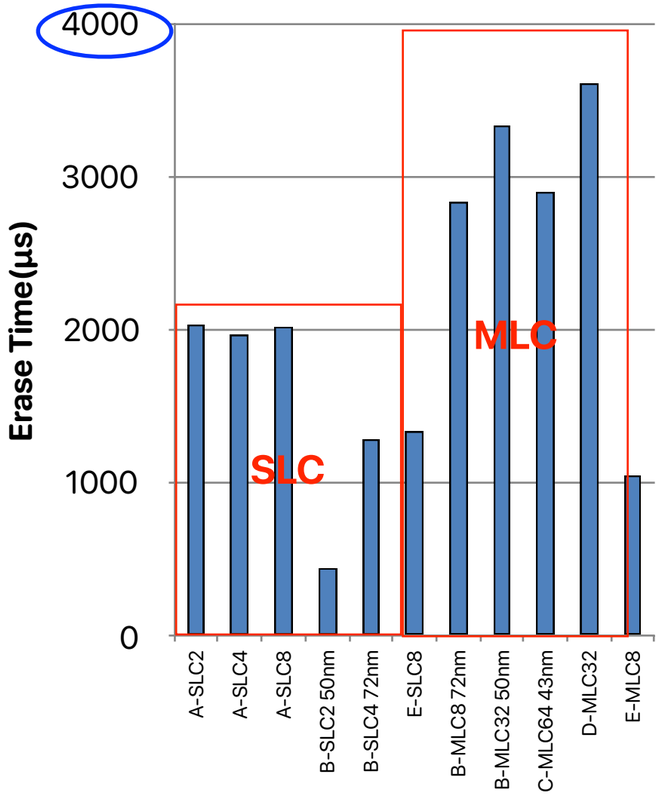
Not a good practice



Reads: less than 150us



Program/write: less than 2ms

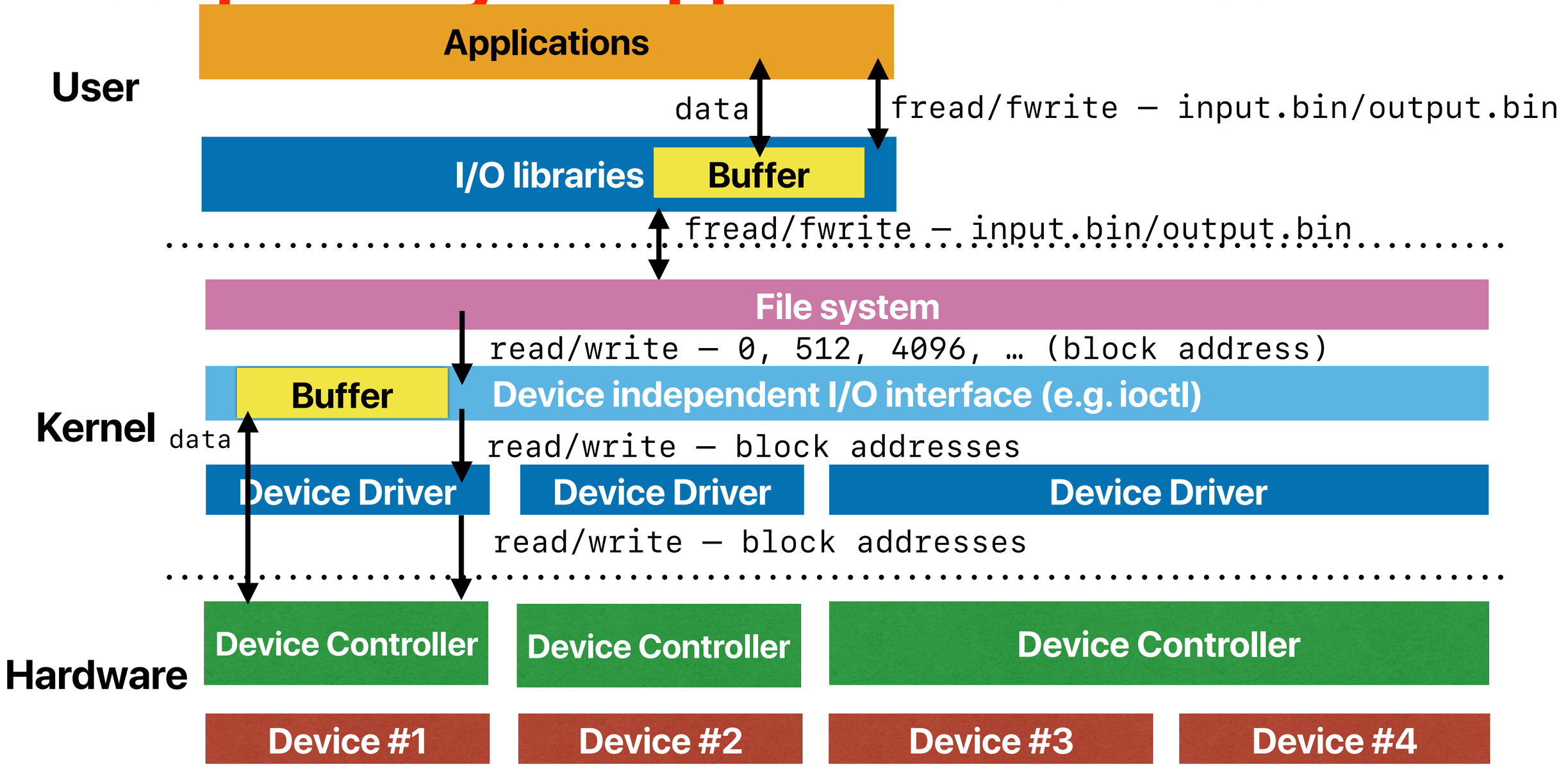


Erase: less than 3.6ms

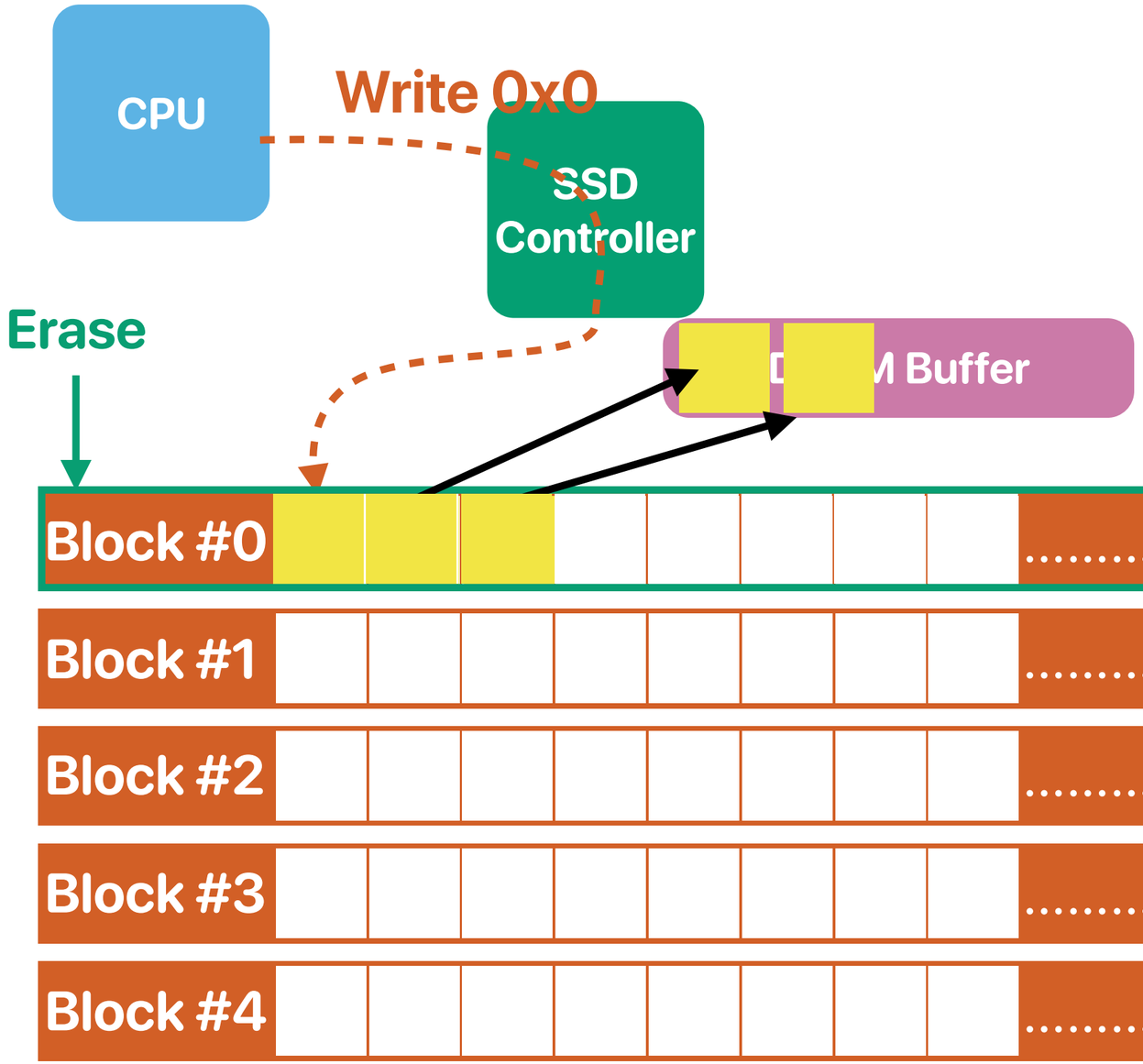
Similar relative performance for reads, writes and erases

Laura M. Grupp, Adrian M. Caulfield, Joel Coburn, Steven Swanson, Eitan Yaakobi, Paul H. Siegel, and Jack K. Wolf. Characterizing flash memory: anomalies, observations, and applications. In MICRO 2009.

Recap: How your application reaches H.D.D.



What happens on a write if we use the same abstractions as H.D.D.



Can we write to page #0 directly? No.

We have to copy page #1, page #2 in block #0 to somewhere (e.g. RAM buffer) and then erase the block

Write this the new 0 and the old 15 back to block #0 again!

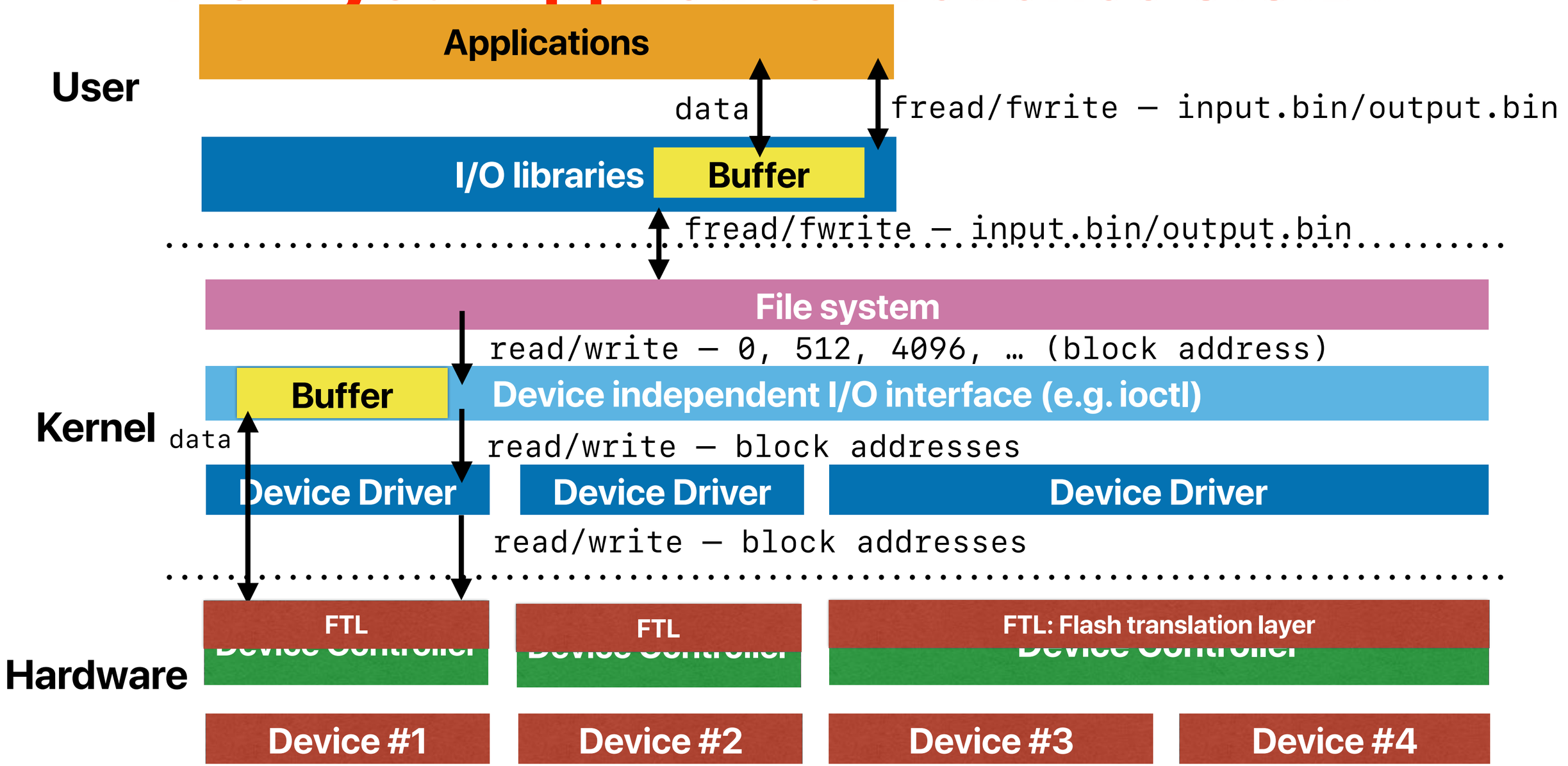
Read: 6*30us + Writing: 2ms*3 + Erasing 3ms ~ 9 ms

Not much faster than the H.D.D. — also hurts the lifetime

All problems in computer science can be solved by another level of indirection

–David Wheeler

How your application reaches S.S.D.



How should we deal with writes?

- How many of following optimizations would help improve the write performance of flash SSDs?
  - ① Write asynchronously — You need RAM buffers
  - ② Out-of-place update — Avoid time consuming read-erase-write
  - ③ Preallocate locations for writing data — You need to maintain a free-list and garbage collection when free list is low
  - ~~④~~ Aggregate writes to the same bank/chip — Probably not. You can write in parallel

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

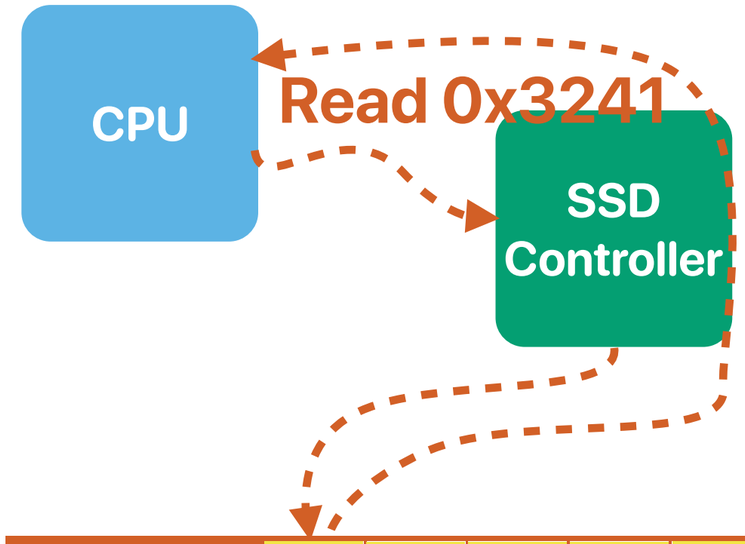
Sounds familiar …

Log-structured file system!

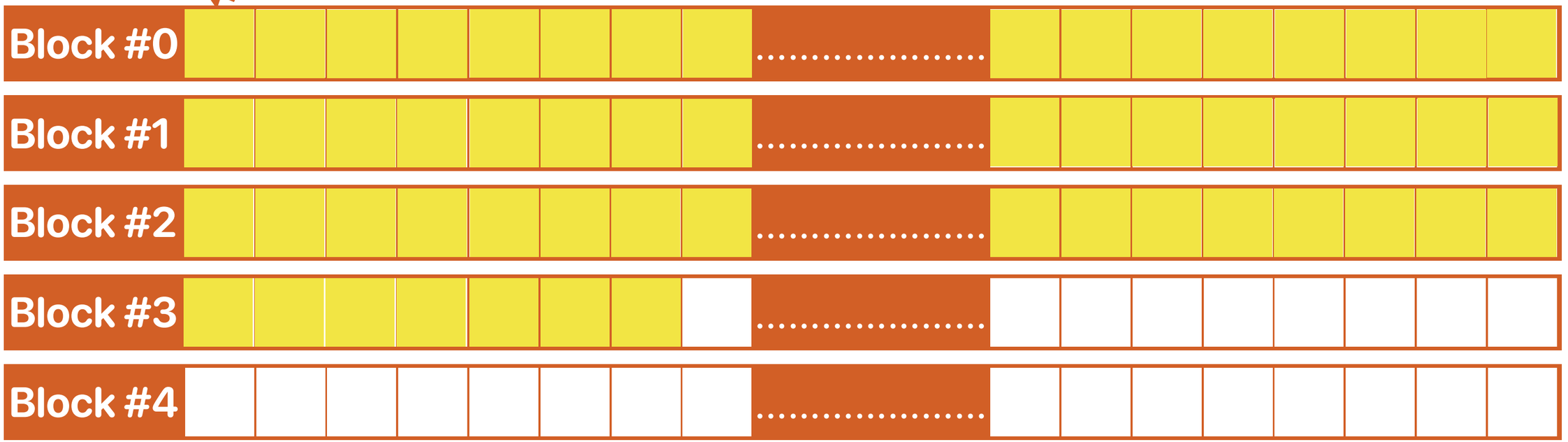
Flash Translation Layer (FTL)

- We are always lazy to modify our applications
  - FTL maintains an abstraction of LBAs (logic block addresses) used between hard disk drives and software applications
  - FTL dynamically maps your logical block addresses to physical addresses on the flash memory chip
- It needs your SSD to have a processor in it now

What happens on a read with FTL



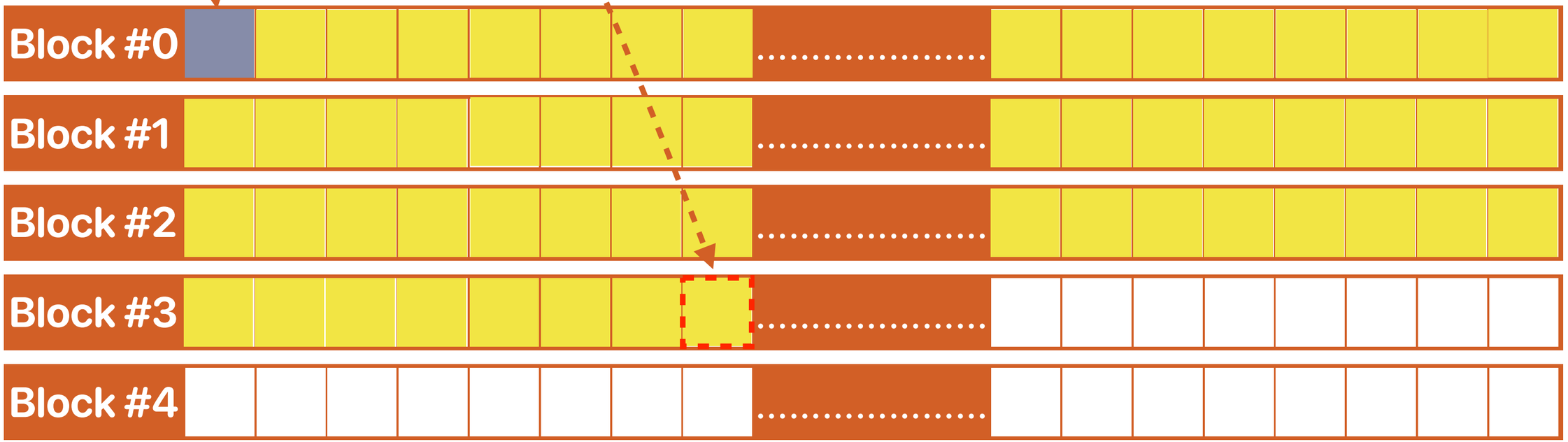
LBA	Flash Block	Flash Page
0x3241	0	0
0x3242	0	63
0x3243	1	3
0x3244	2	4
0x3245	3	6
0x3246	2	7



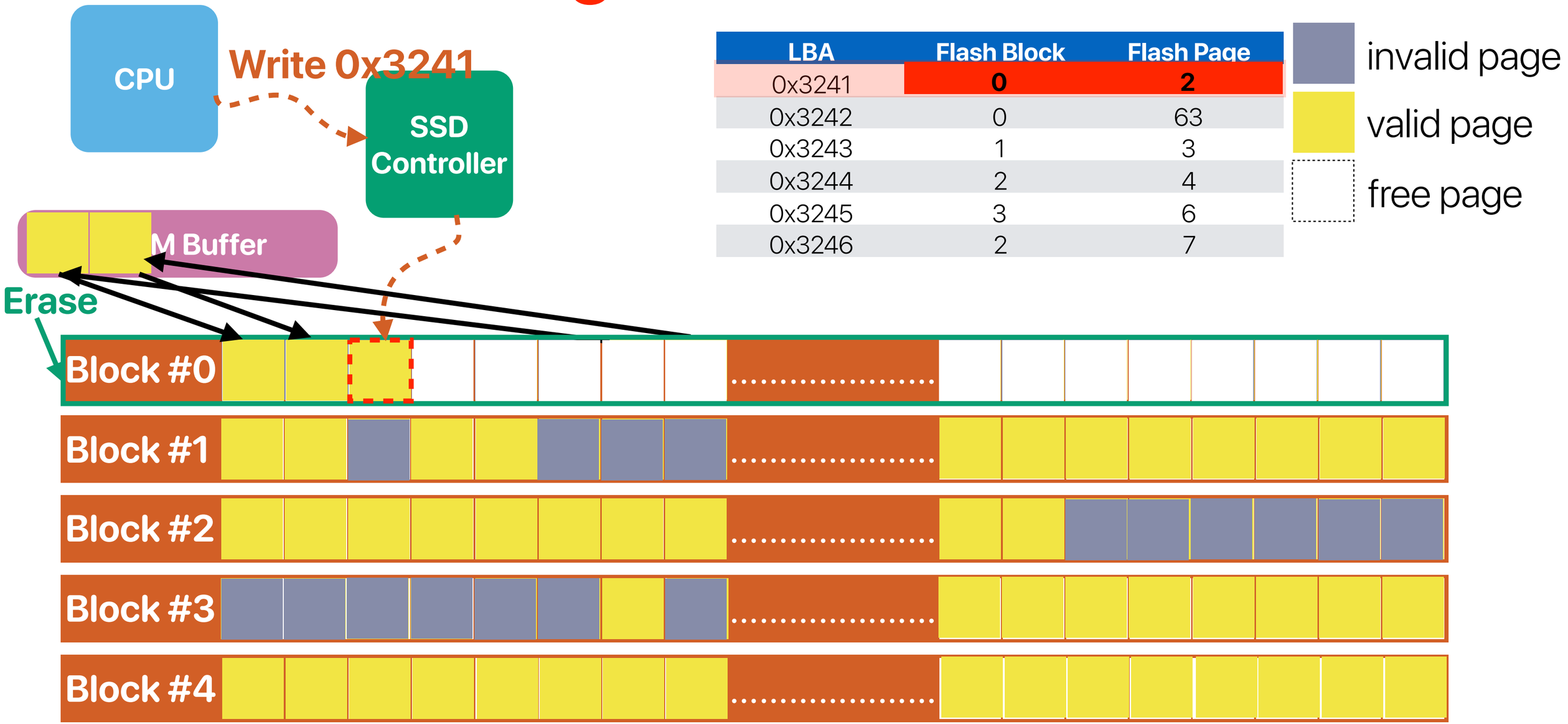
What happens on a write with FTL

Write 0x3241

LBA	Flash Block	Flash Page
0x3241	**3**	**7**
0x3242	0	63
0x3243	1	3
0x3244	2	4
0x3245	3	6
0x3246	2	7



Garbage Collection in FTL



LBA	Flash Block	Flash Page
0x3241	**0**	**2**
0x3242	0	63
0x3243	1	3
0x3244	2	4
0x3245	3	6
0x3246	2	7

Flash Translation Layer (FTL)

- We are always lazy to modify our applications
  - FTL maintains an abstraction of LBAs (logic block addresses) used between hard disk drives and software applications
  - FTL dynamically maps your logical block addresses to physical addresses on the flash memory chip
  - FTL performs copy-on-write when there is an update
  - FTL reclaims invalid data regions and data blocks to allow future updates
  - FTL executes wear-leveling to maximize the life time
- It needs your SSD to have a processor in it now

Why eNVy

- Flash memories have different characteristics than conventional storage and memory technologies
- We want to minimize the modifications in our software

What eNVy proposed

- A file system inside flash that performs
  - Transparent in-place update
  - Page remapping
  - Caching/Buffering
  - Garbage collection
- Exactly like LFS

Utilization and performance

- Performance degrades as your store more data
- Modern SSDs provision storage space to address this issue

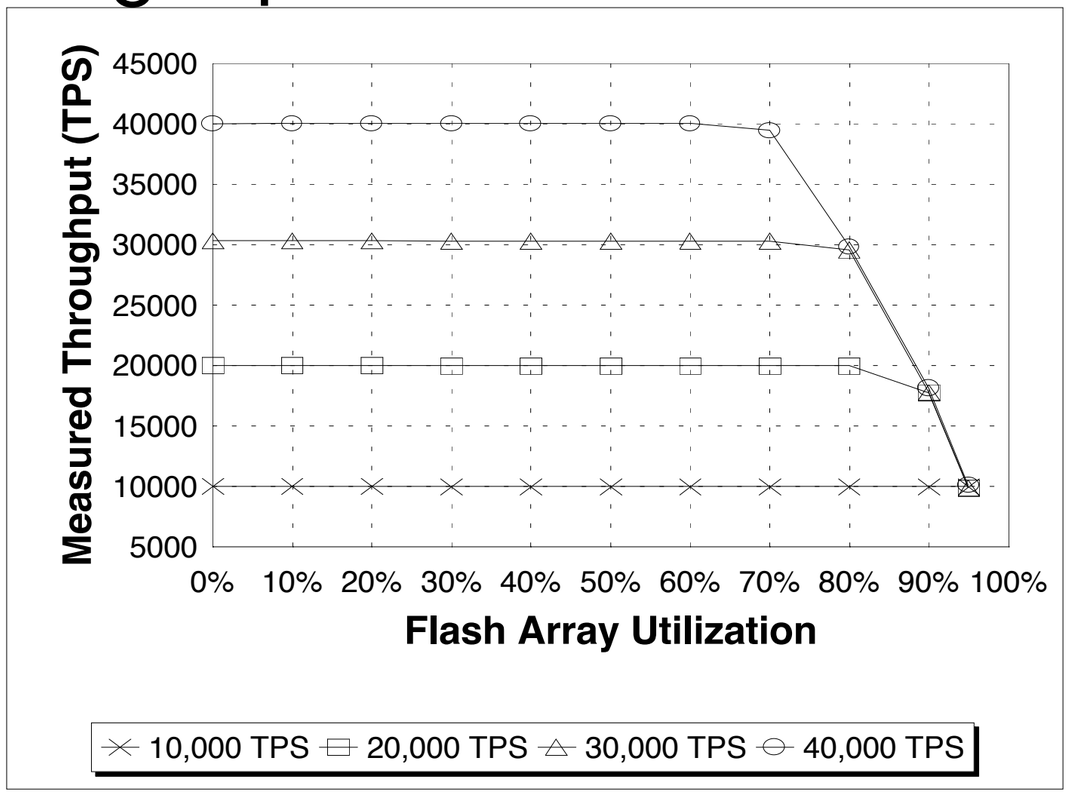
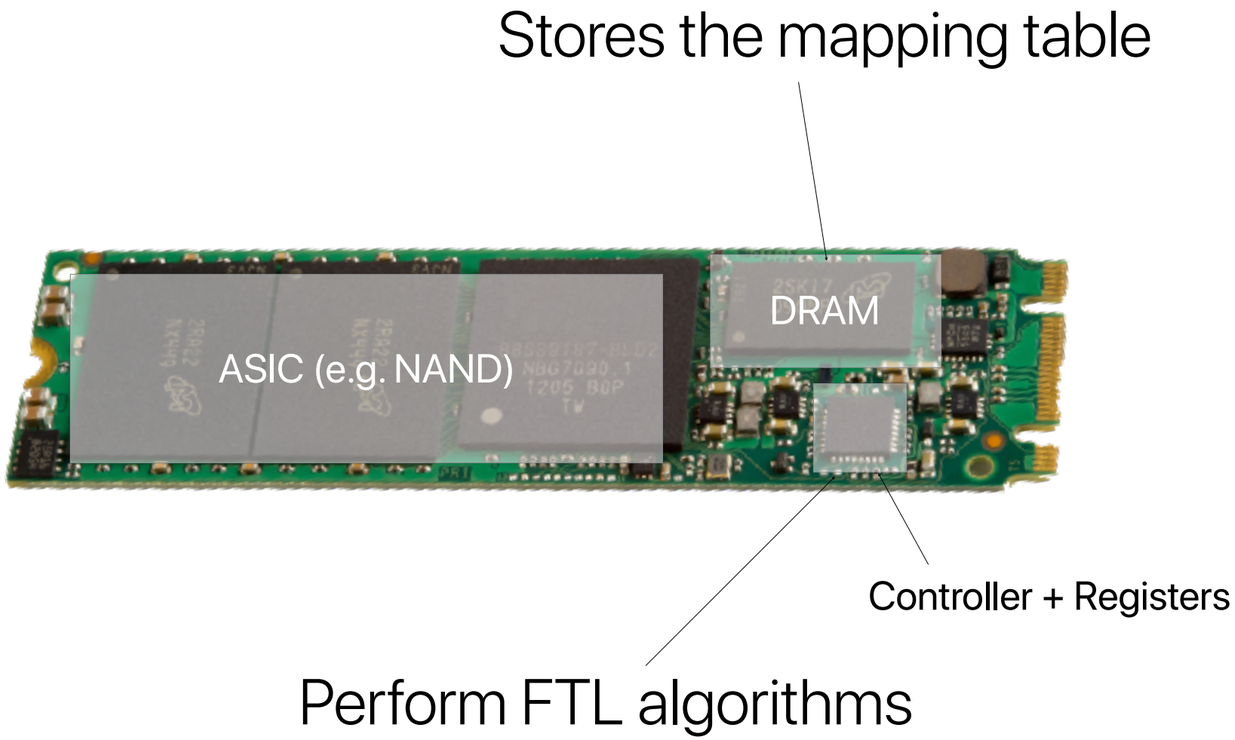


Figure 14: Throughput for Various Levels of Utilization

The impact of eNVy

- Your SSD structured exactly like this!

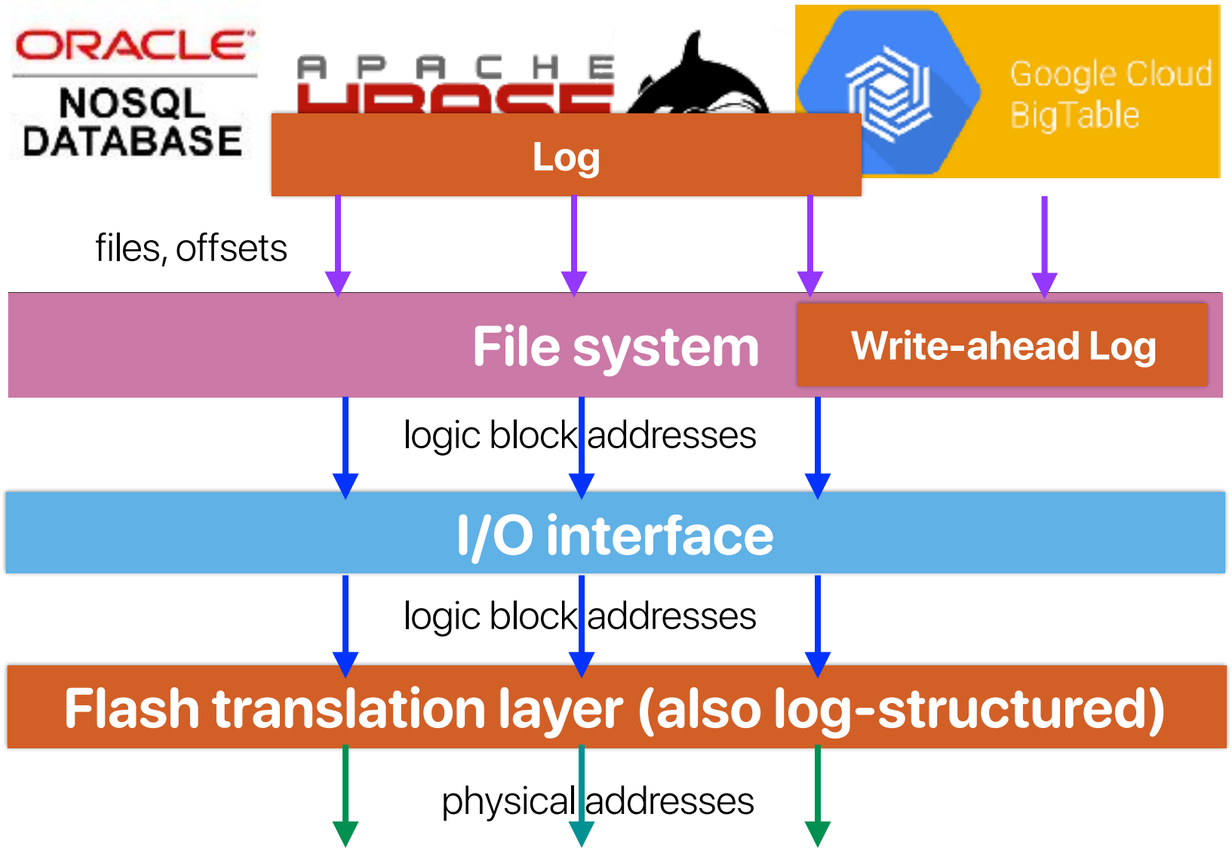


Don’t stack your log on my log

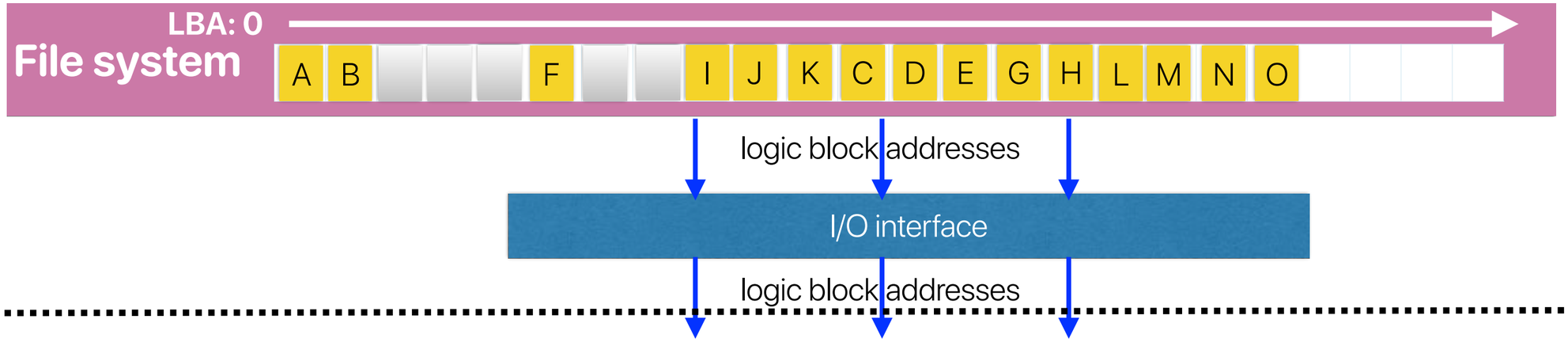
Jingpei Yang, Ned Plasson, Greg Gillis, Nisha Talagala, and Swaminathan Sundararaman
SanDisk Corporation

Why should we care about this paper?

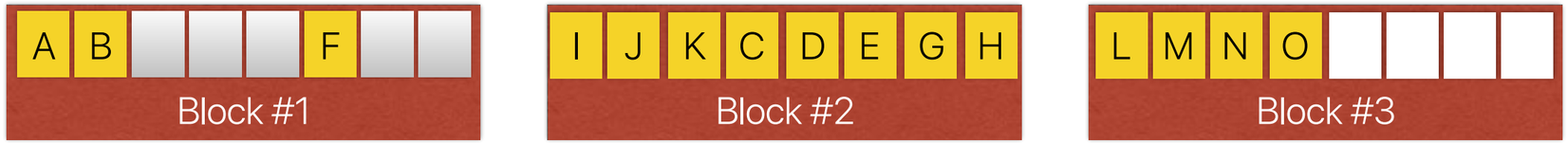
- Log is everywhere
  - Application: database
  - File system
  - Flash-based SSDs
- They can interfere with each other!
- An issue with software engineering nowadays



For example, garbage collection



SSD



invalid

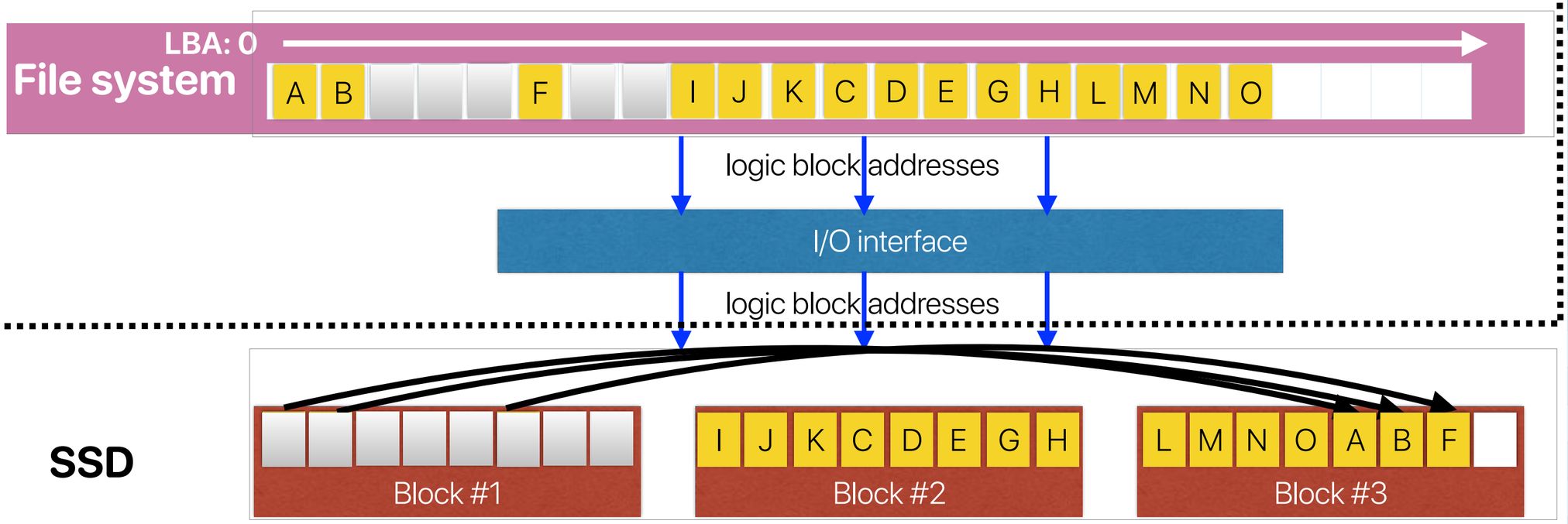
valid

free

FTL mapping table

LBA	block #	page #
0	1	0
1	1	1
2	-	-
3	-	-
4	-	-
5	1	5
6	-	-
7	-	-
8	2	0
9	2	1
10	2	2
11	2	3
12	2	4
13	2	5
14	2	6
15	2	7
16	3	0
17	3	1
18	3	2
19	3	3
20	-	-
21	-	-
22	-	-
23	-	-

Now, SSD wants to reclaim a block



FTL mapping table

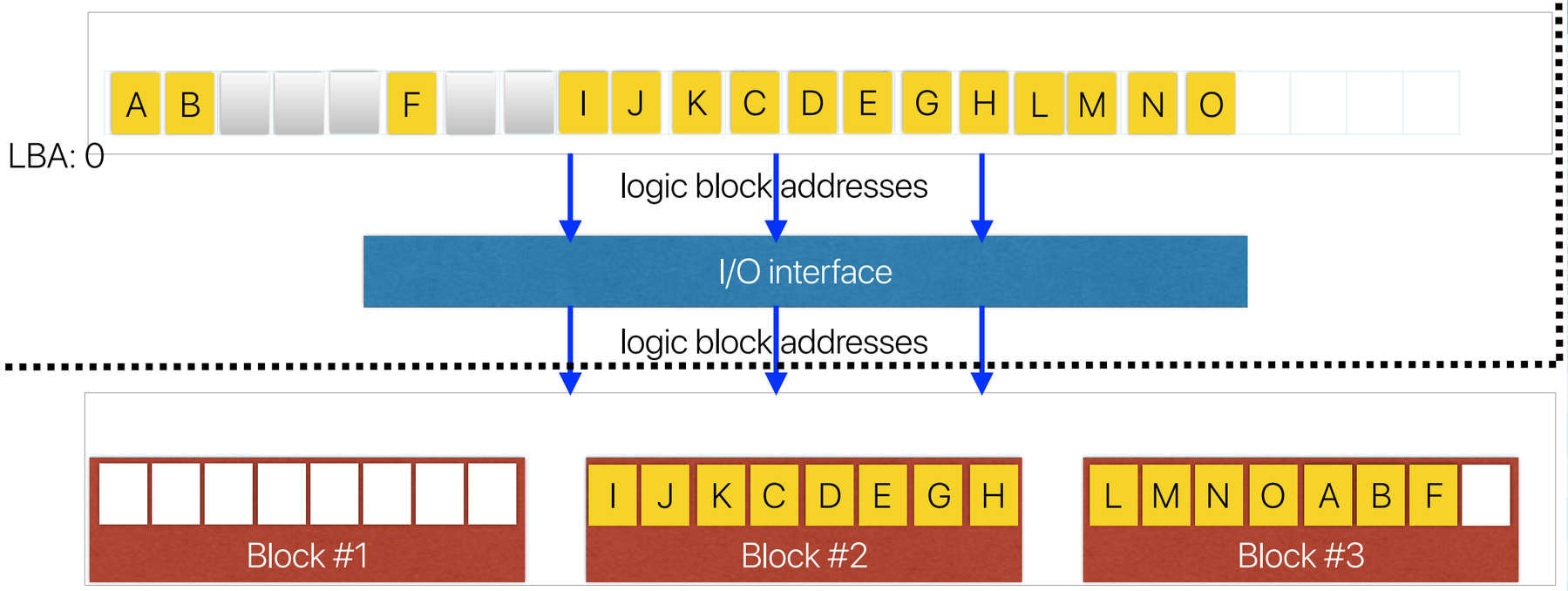
LBA	block #	page #
0	3	4
1	3	5
2	-	-
3	-	-
4	-	-
5	3	6
6	-	-
7	-	-
8	2	0
9	2	1
10	2	2
11	2	3
12	2	4
13	2	5
14	2	6
15	2	7
16	3	0
17	3	1
18	3	2
19	3	3
20	-	-
21	-	-
22	-	-
23	-	-

invalid

valid

free

Garbage collection on the SSD done!



invalid

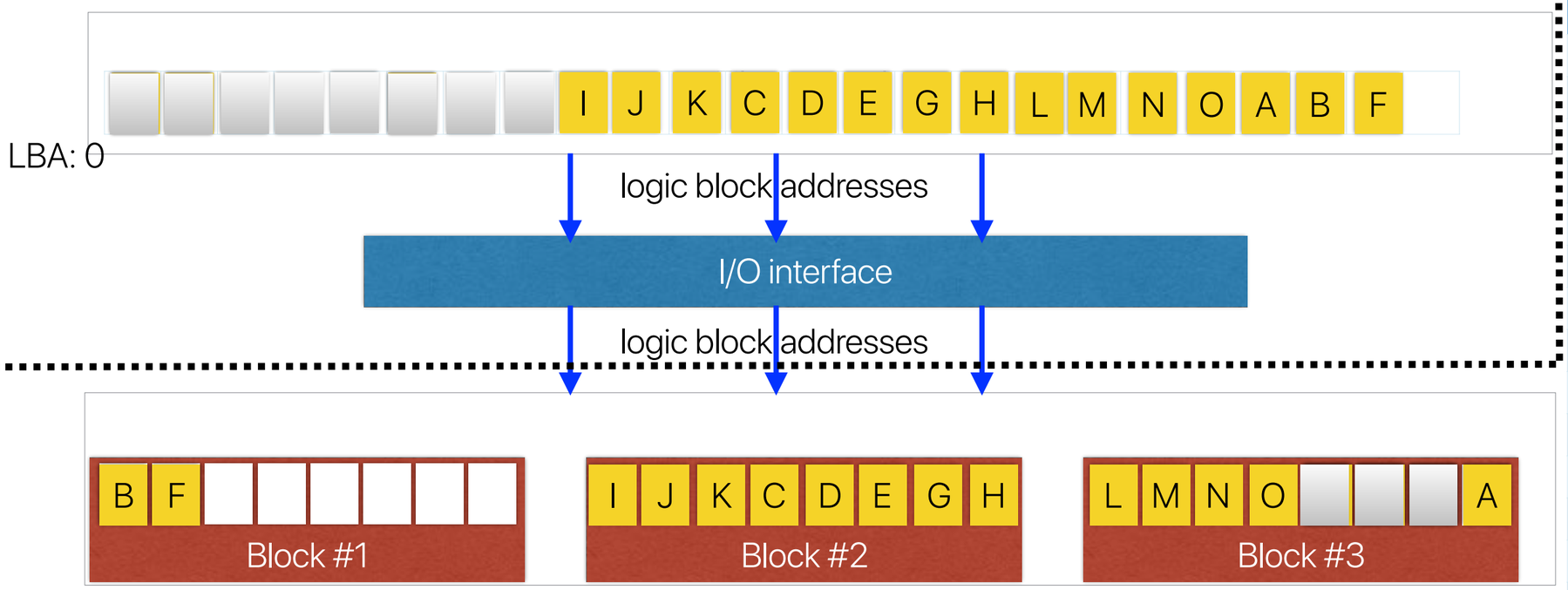
valid

free

FTL mapping table

LBA	block #	page #
0	3	4
1	3	5
2	-	-
3	-	-
4	-	-
5	3	6
6	-	-
7	-	-
8	2	0
9	2	1
10	2	2
11	2	3
12	2	4
13	2	5
14	2	6
15	2	7
16	3	0
17	3	1
18	3	2
19	3	3
20	-	-
21	-	-
22	-	-
23	-	-

What will happen if the FS wants to perform GC?



We could have avoided writing the stale A, B, F if they are coordinated!

invalid

valid

free

FTL mapping table

LBA	block #	page #
0	-	-
1	-	-
2	-	-
3	-	-
4	-	-
5	-	-
6	-	-
7	-	-
8	2	0
9	2	1
10	2	2
11	2	3
12	2	4
13	2	5
14	2	6
15	2	7
16	3	0
17	3	1
18	3	2
19	3	3
20	3	7
21	1	0
22	1	1
23	-	-

All problems in computer science can be solved by another level of indirection

–David Wheeler

...except for the problem of too many layers of indirection.

File systems for flash-based SSDs

- Still an open research question
- Software designer should be aware of the characteristics of underlying hardware components
- Revising the layered design to expose more SSD information to the file system or the other way around