File Systems & The Era of Flash-based SSD

Hung-Wei Tseng

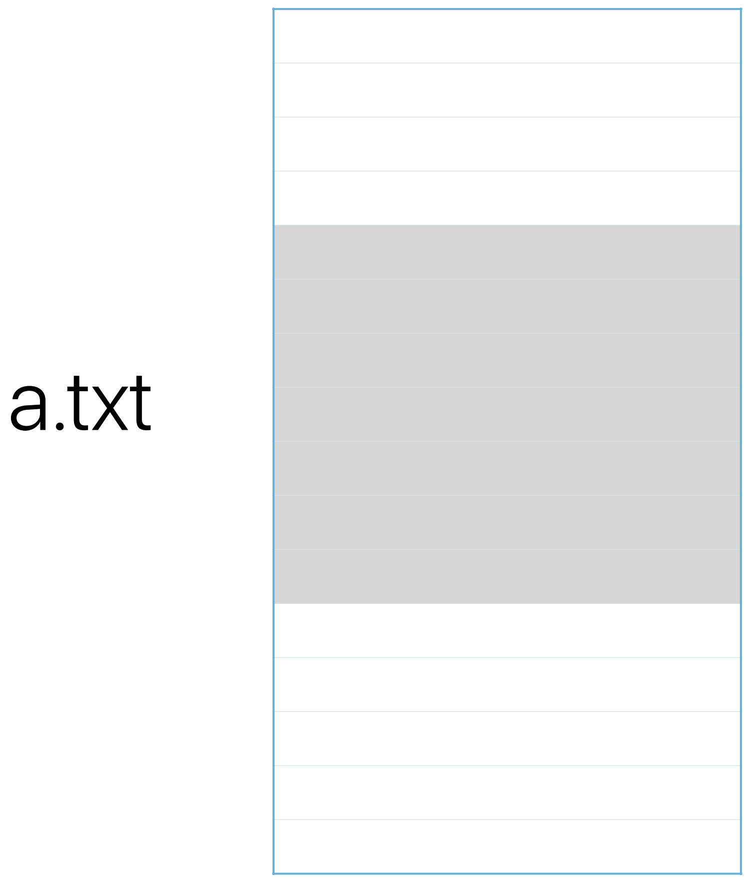
Outline

- 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

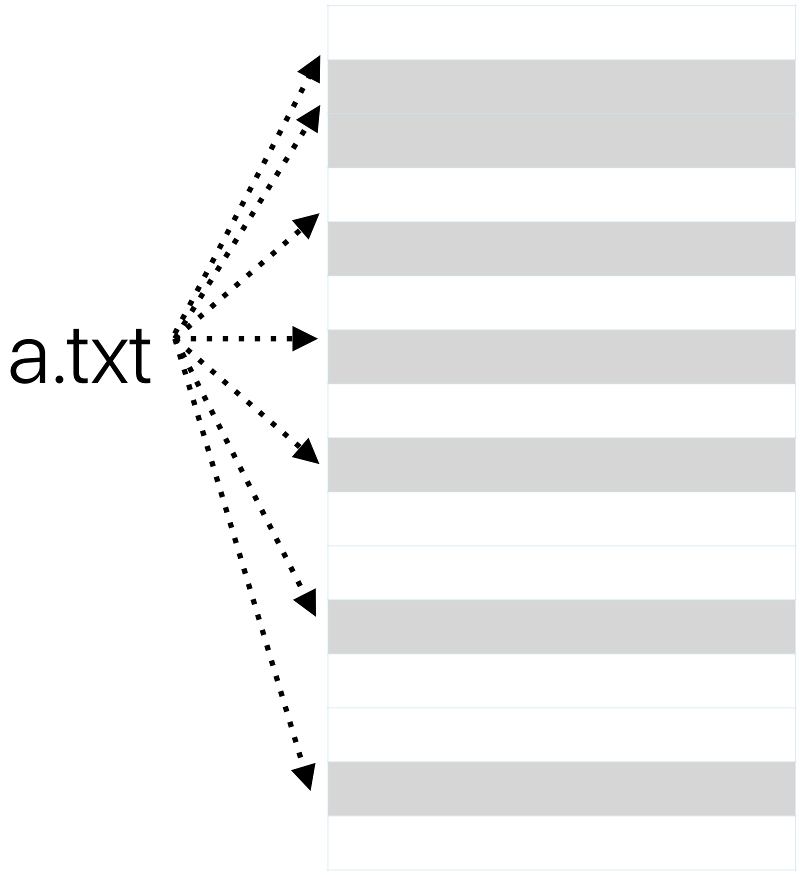
How do we allocate disk space?

- Contiguous: the file resides in continuous addresses

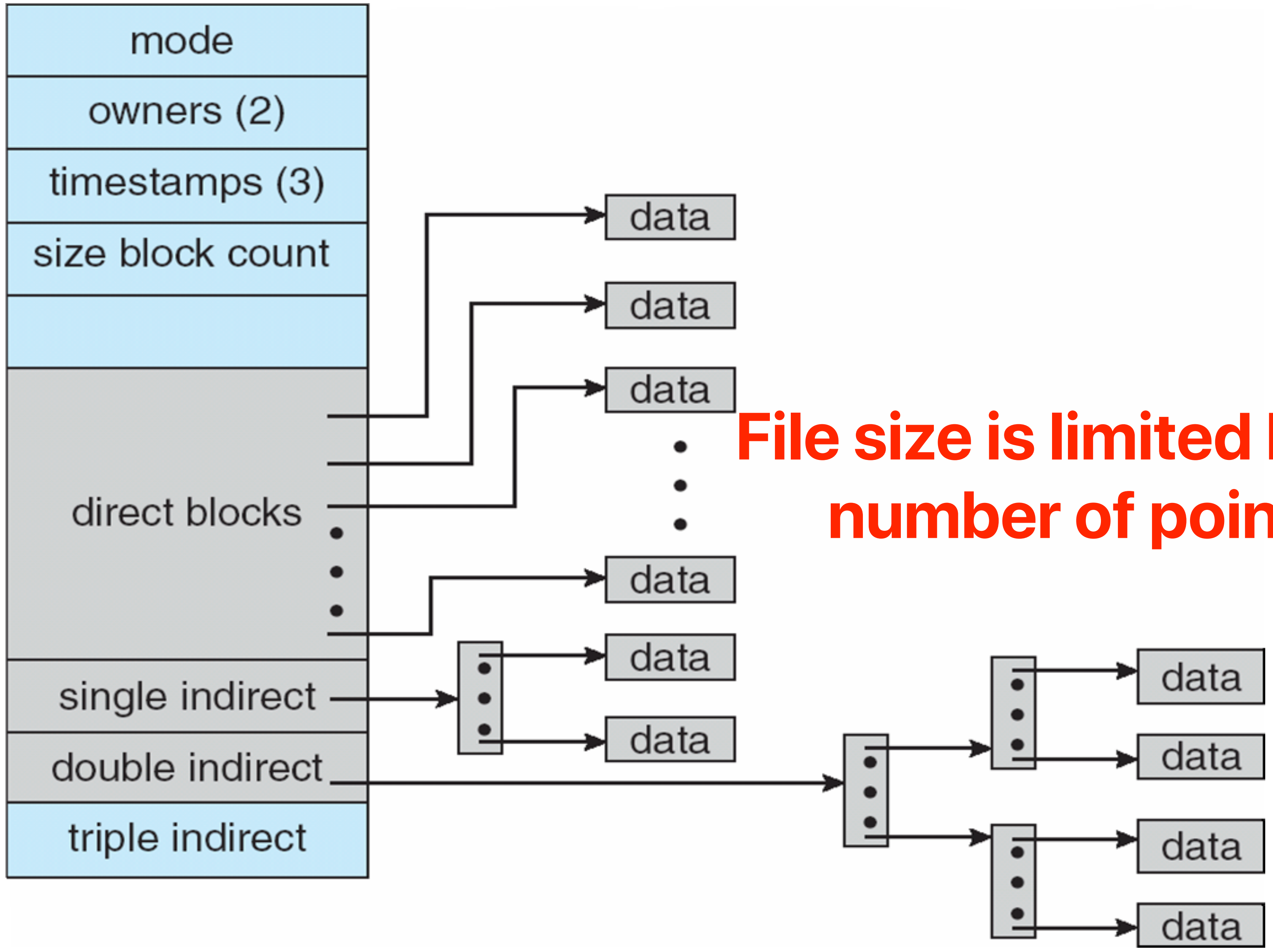


external fragment as in Segmentation

- Non-contiguous: the file can be anywhere



Conventional Unix inode

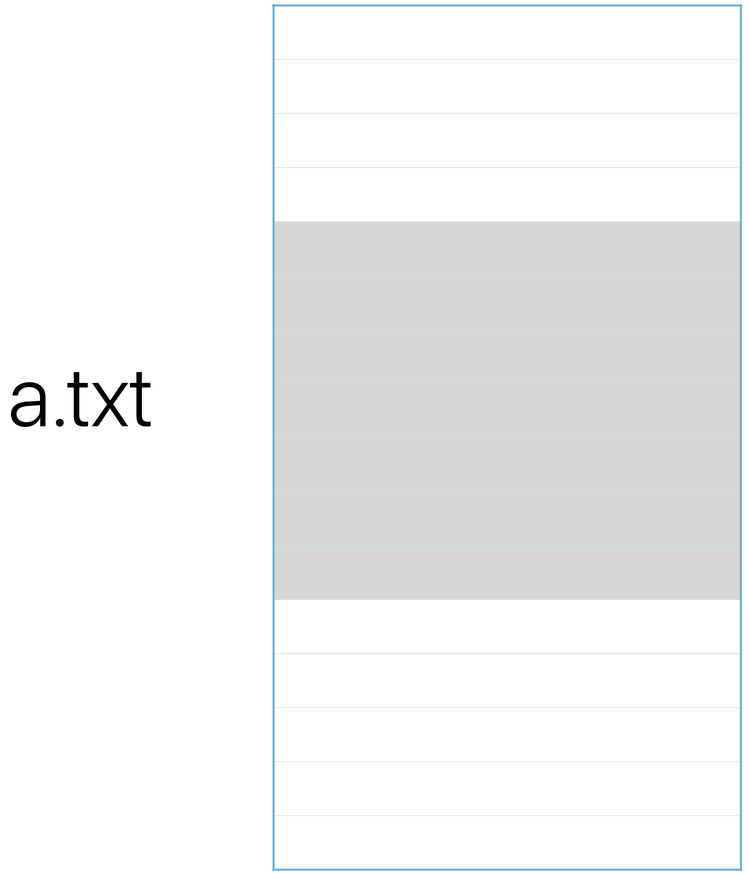


File size is limited by total number of pointers

- File types: directory, file
- File size
- Permission
- Attributes
- Types of pointers:
  - Direct: Access single data block
  - Single Indirect: Access n data blocks
  - Double indirect: Access n2 data blocks
  - Triple indirect: Access n3 data blocks
- inode has 15 pointers: 12 direct, 1 each single-, double-, and triple-indirect
- If data block size is 512B and n = 256: max file size = (12+256+2562+2563)*512 = 8GB

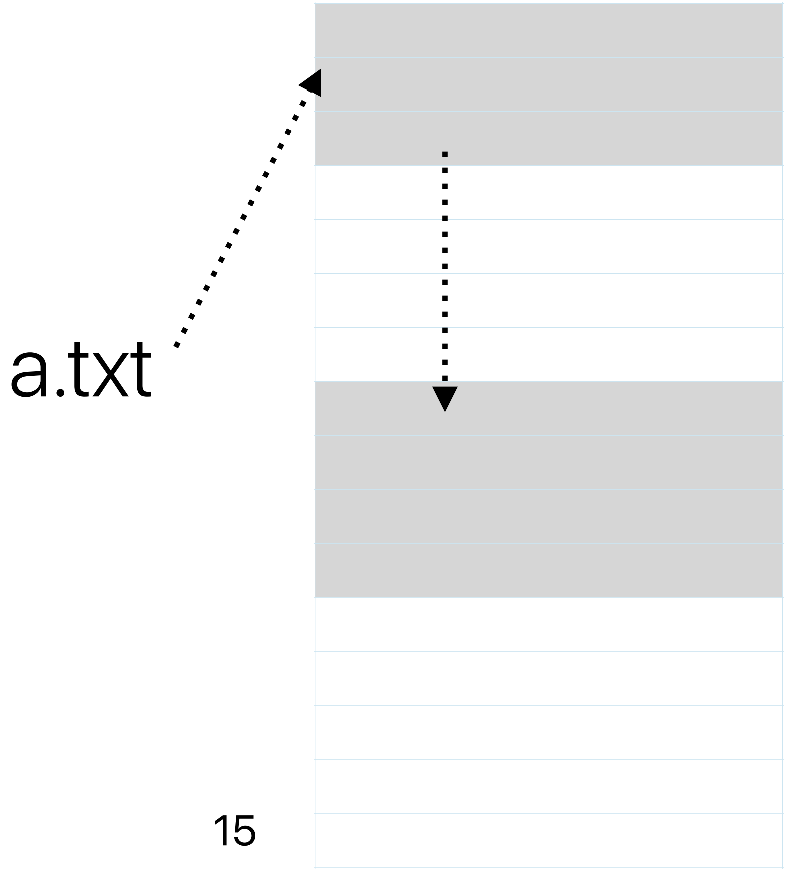
How do we allocate space?

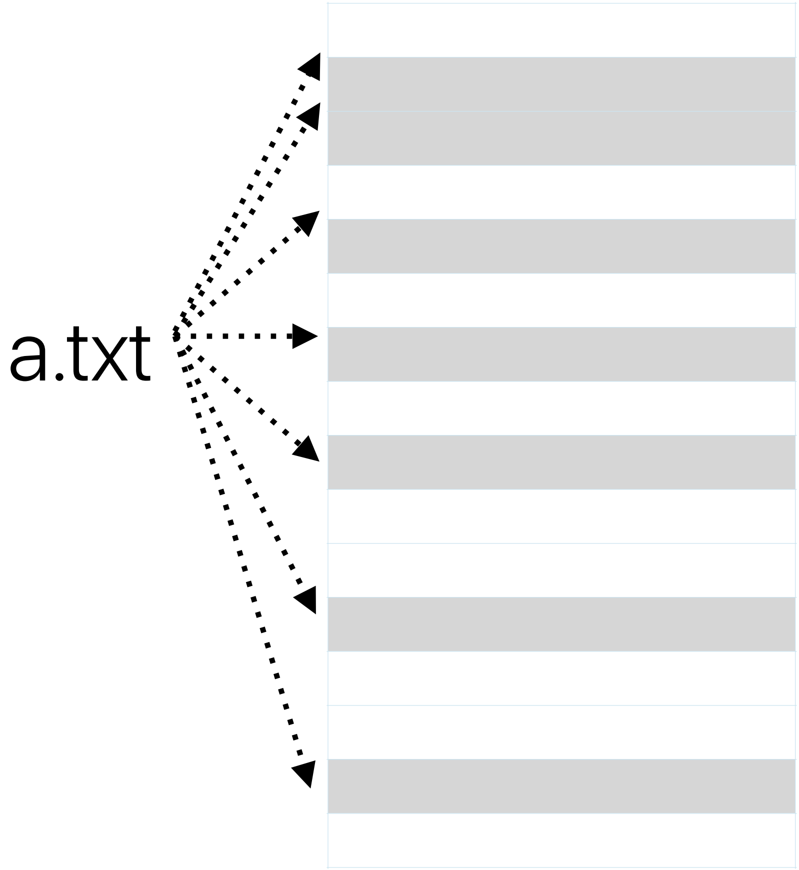
- Contiguous: the file resides in continuous addresses



- 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

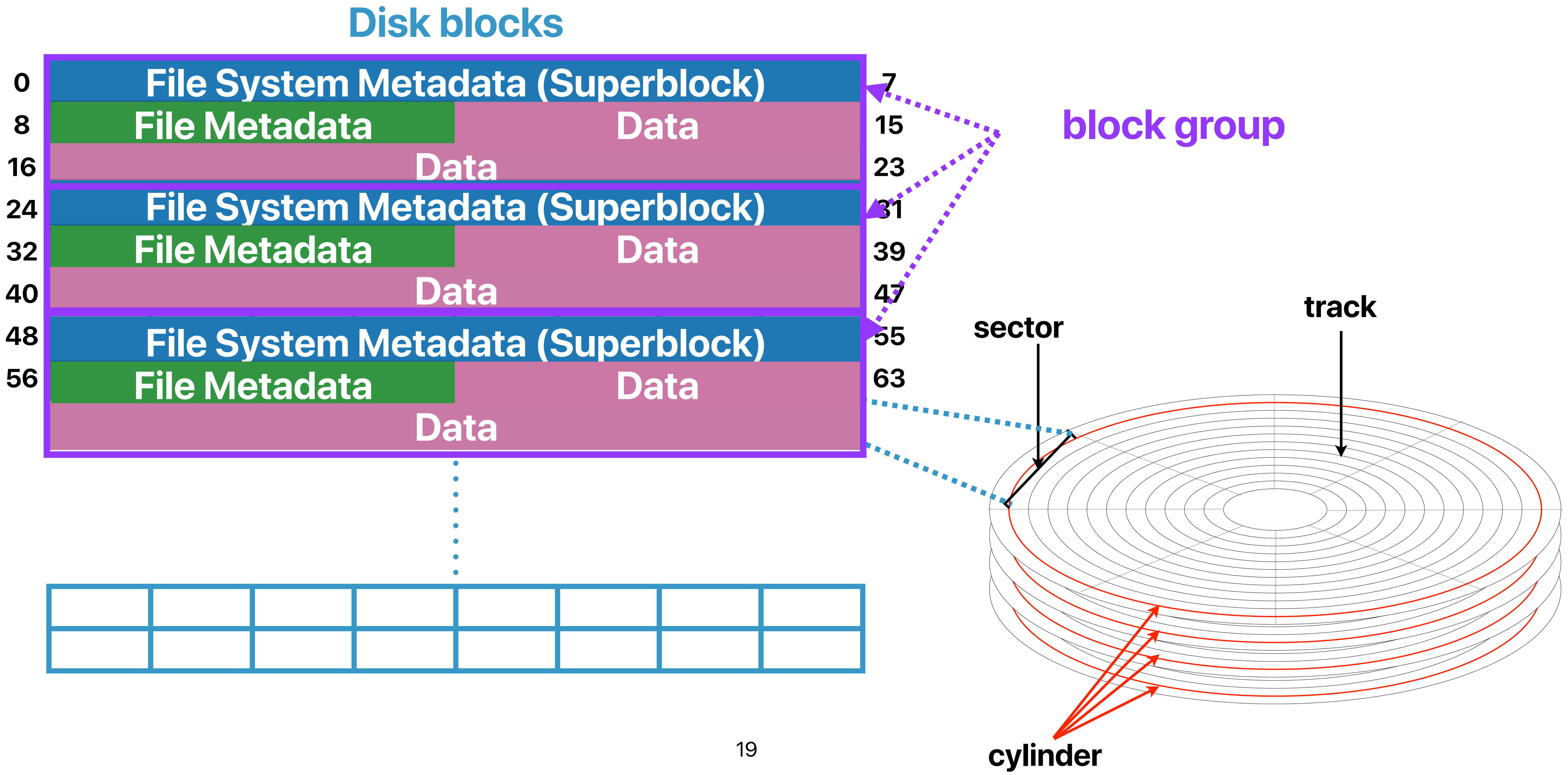
Extent file systems — ext2, ext3, ext4

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

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

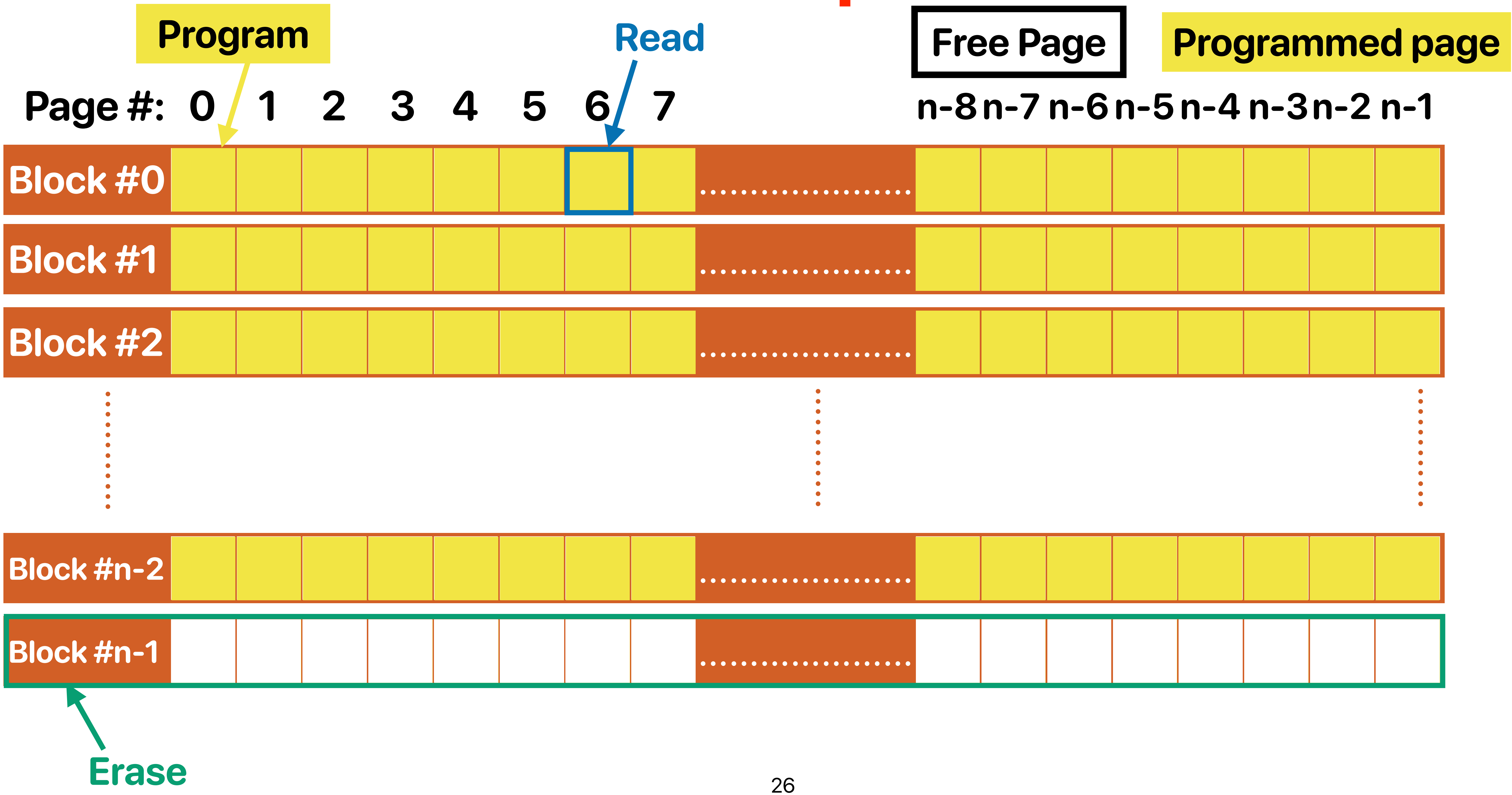
Flash-based SSDs
and
eNVy: A non-volatile, main memory storage system

Michael Wu and Willy Zwaenepoel
Rice University

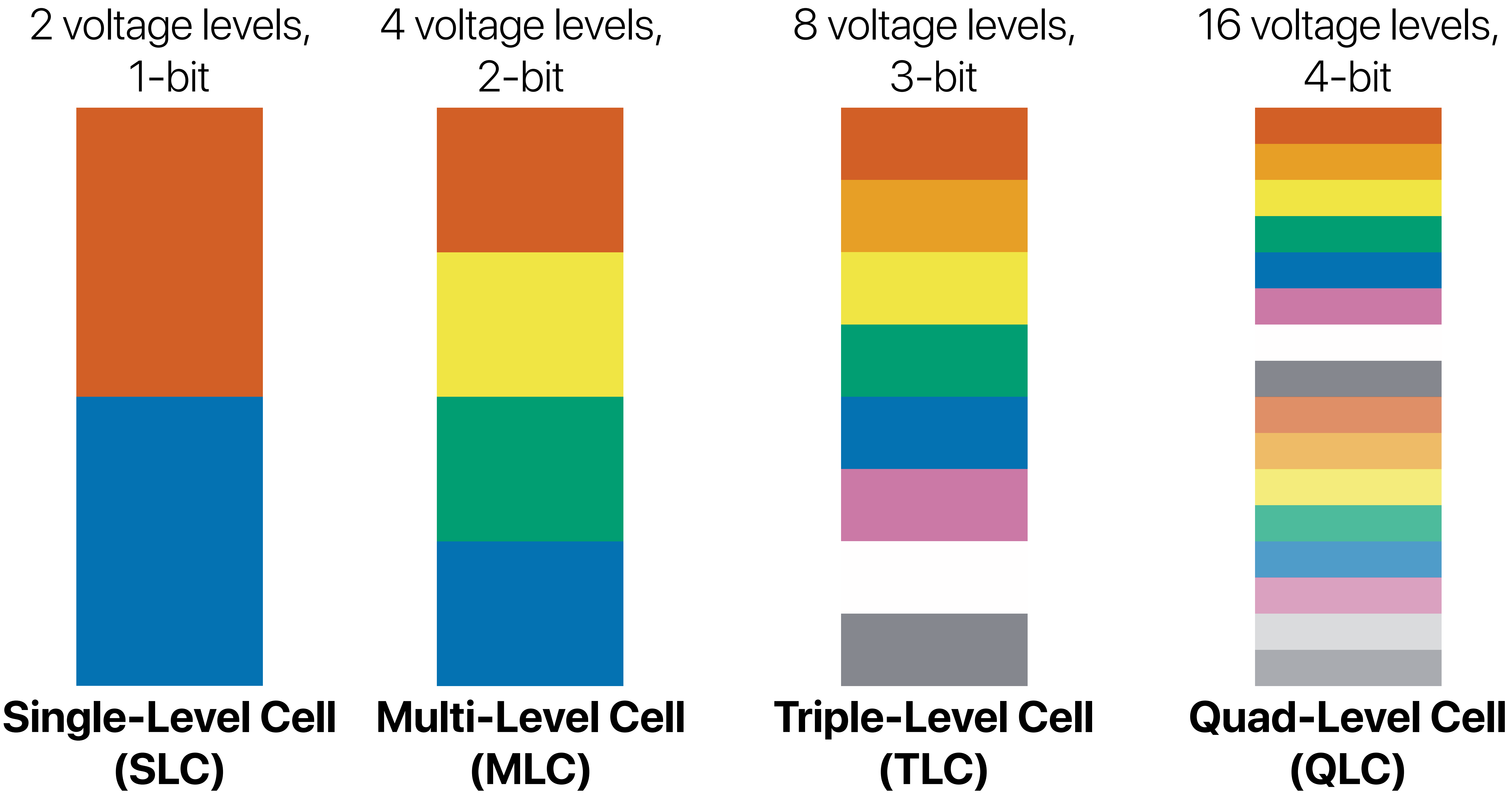
Flash memory: eNVy and now

	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



Types of Flash Chips



Programming in MLC

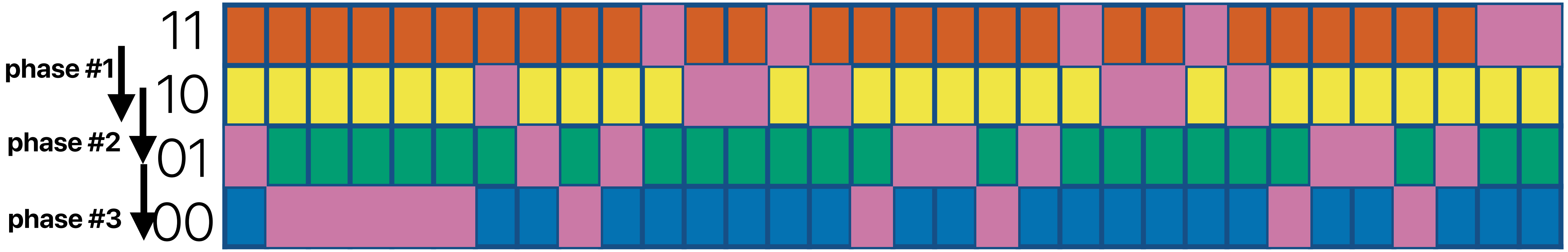
4 voltage levels,
2-bit

Multi-Level Cell (MLC)

3.14000000000000012434497875801

= 0x40091EB851EB851F

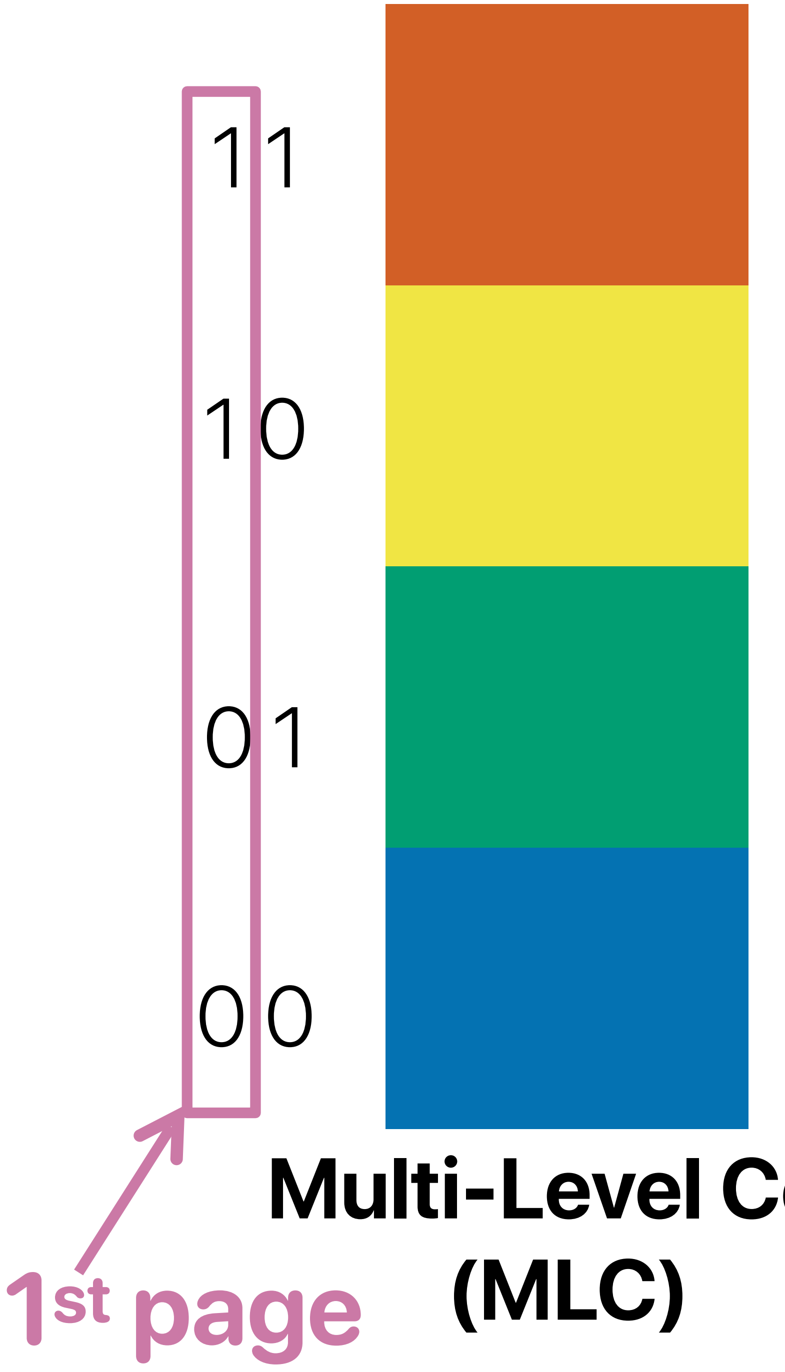
= 01000000 00001001 00011110 10111000 01010001 11101011 10000101 00011111



3 Cycles/Phases to finish programming

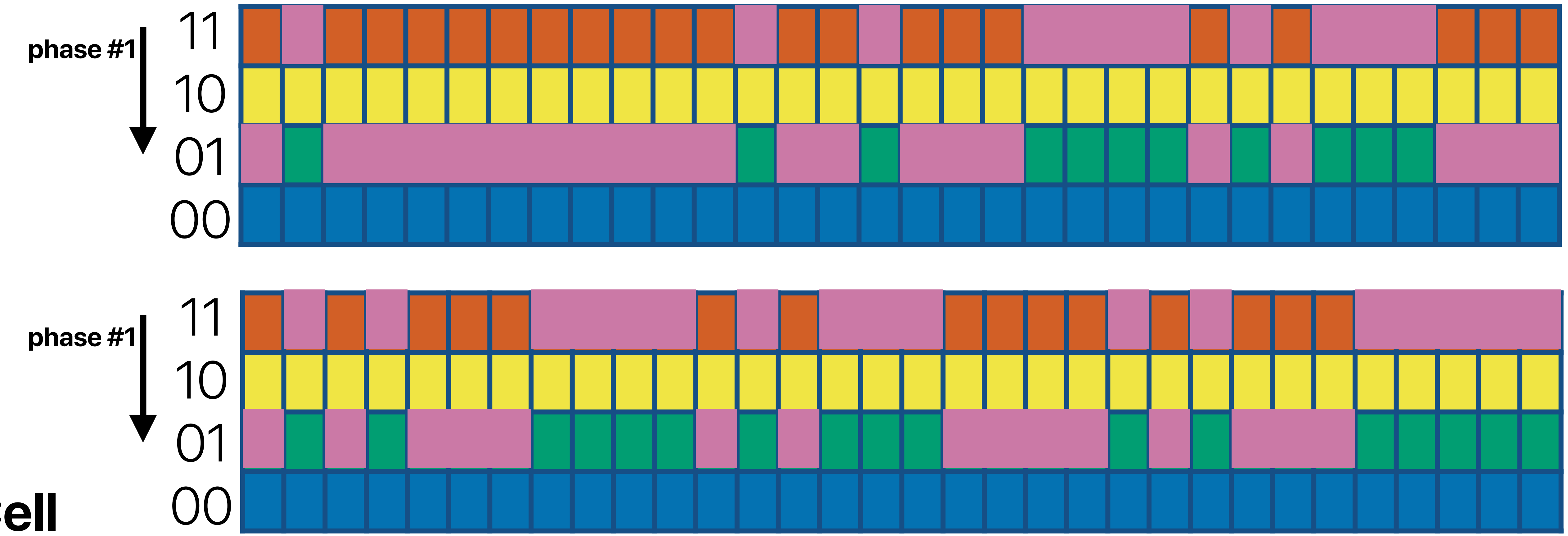
Programming in MLC

4 voltage levels,
2-bit



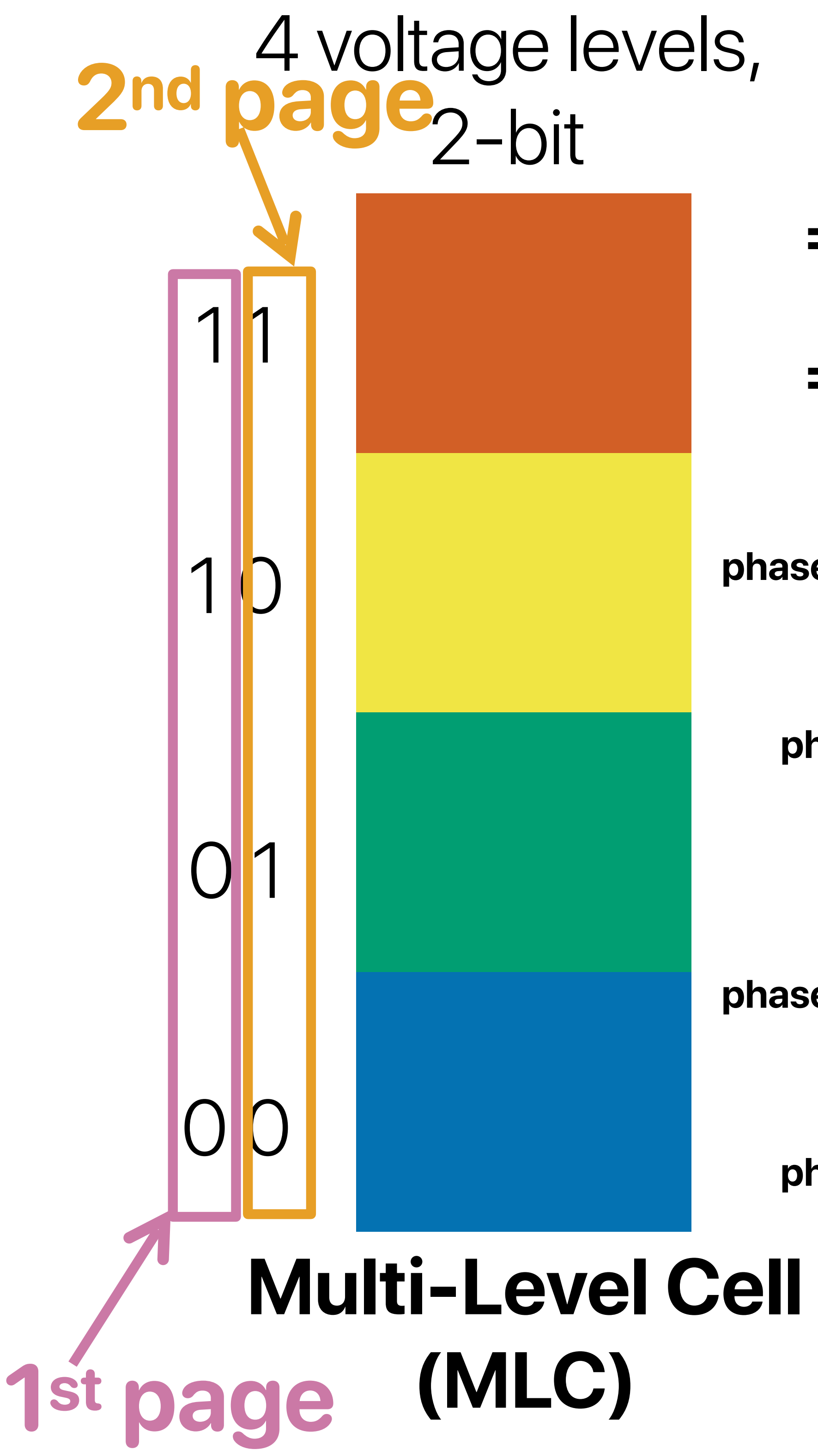
Multi-Level Cell (MLC)

3.14000000000000012434497875800
= 0x40091EB851EB851F
= 01000000 00001001 00011110 10111000 01010001 11101011 10000101 00011111



1 Phase to finish programming the first page!

Programming the 2nd page in MLC



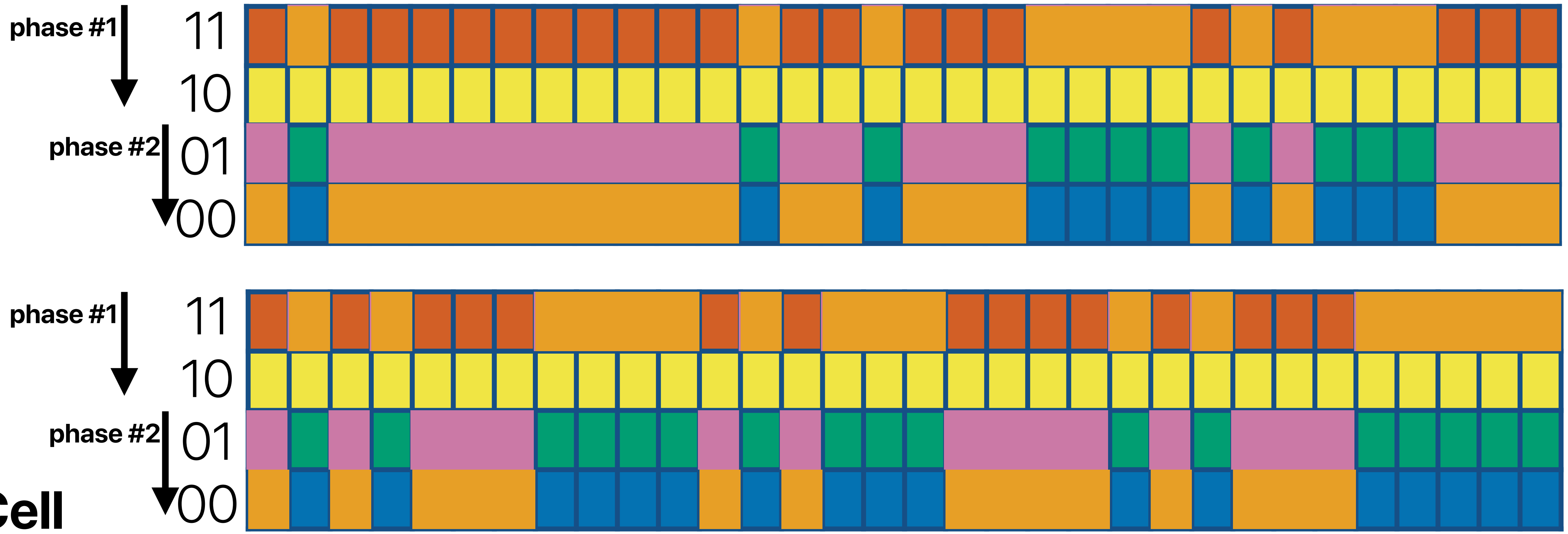
Multi-Level Cell (MLC)

3.14000000000000012434497875802

= 0x40091EB851EB851F

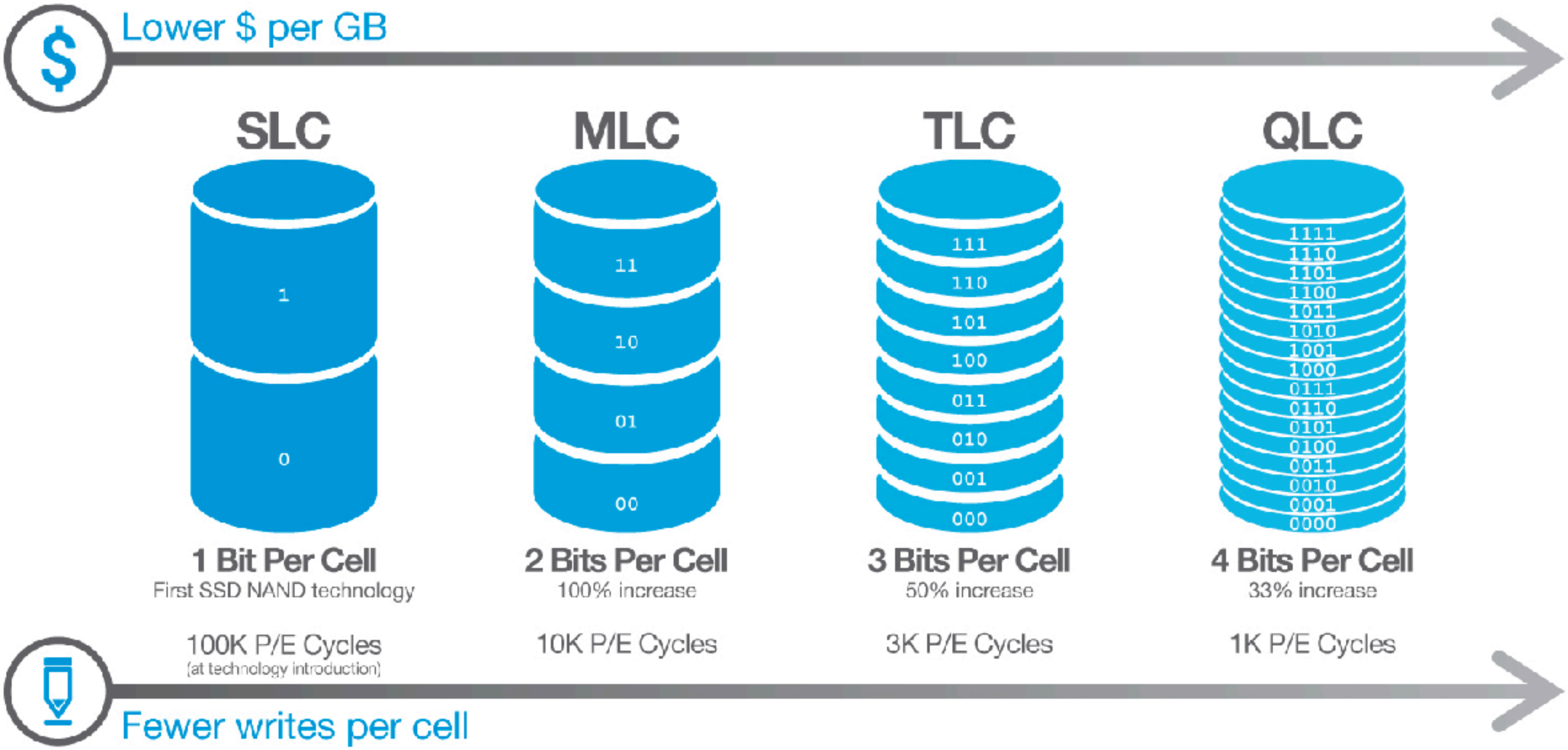
= 01000000 00001001 00011110 10111000 01010001 11101011 10000101 00011111

= 01000000 00001001 00011110 10111000 01010001 11101011 10000101 00011111



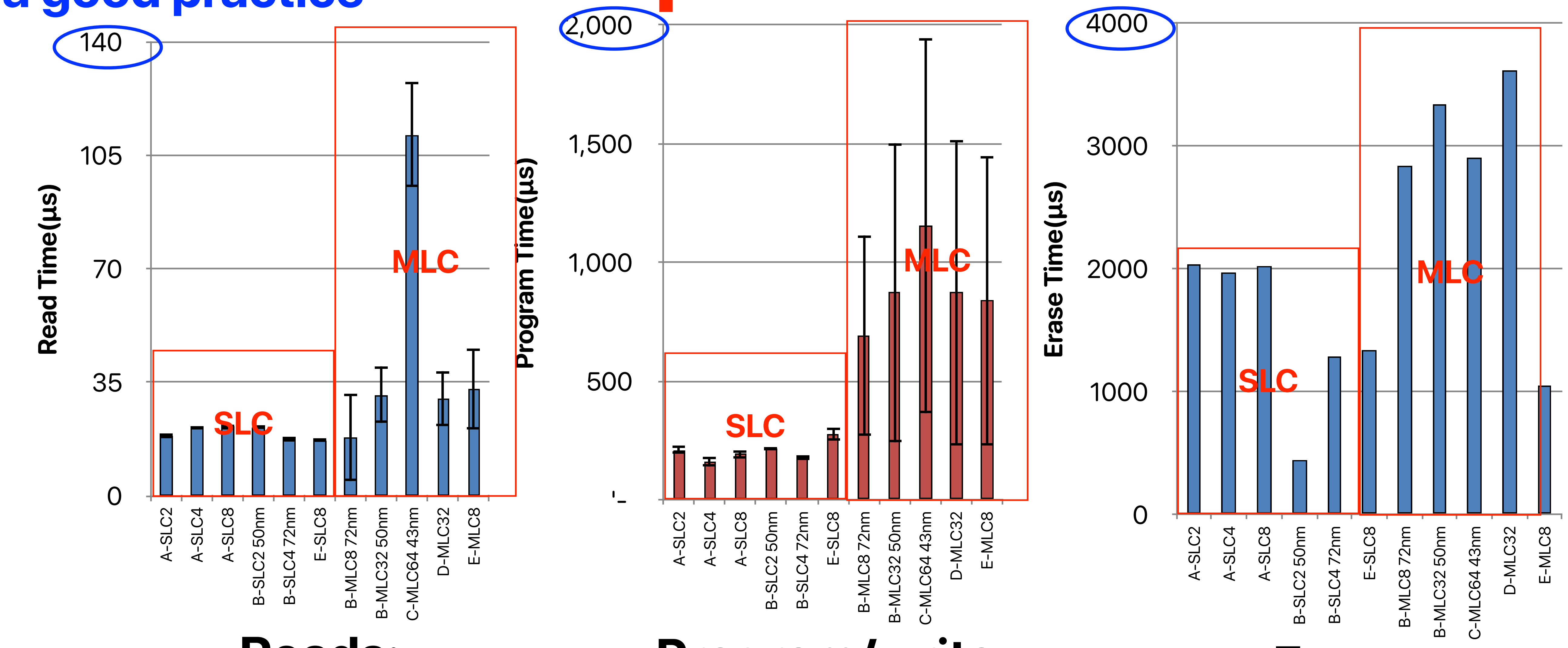
2 Phase to finish programming the second page!

QLC = More Density Per NAND Cell



Flash performance

Not a good practice



Reads: less than 150us

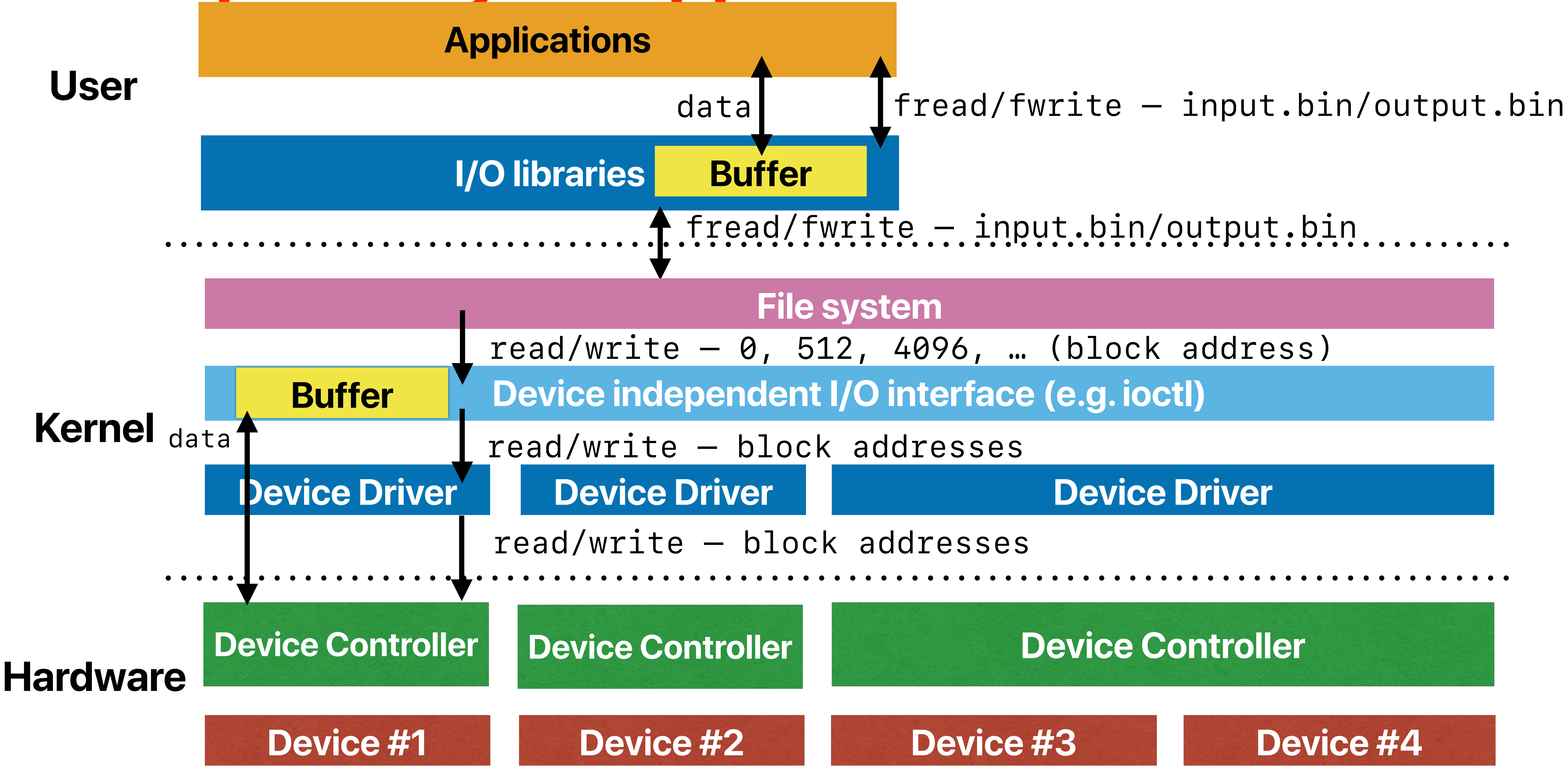
Program/write: less than 2ms

Erase: less than 3.6ms

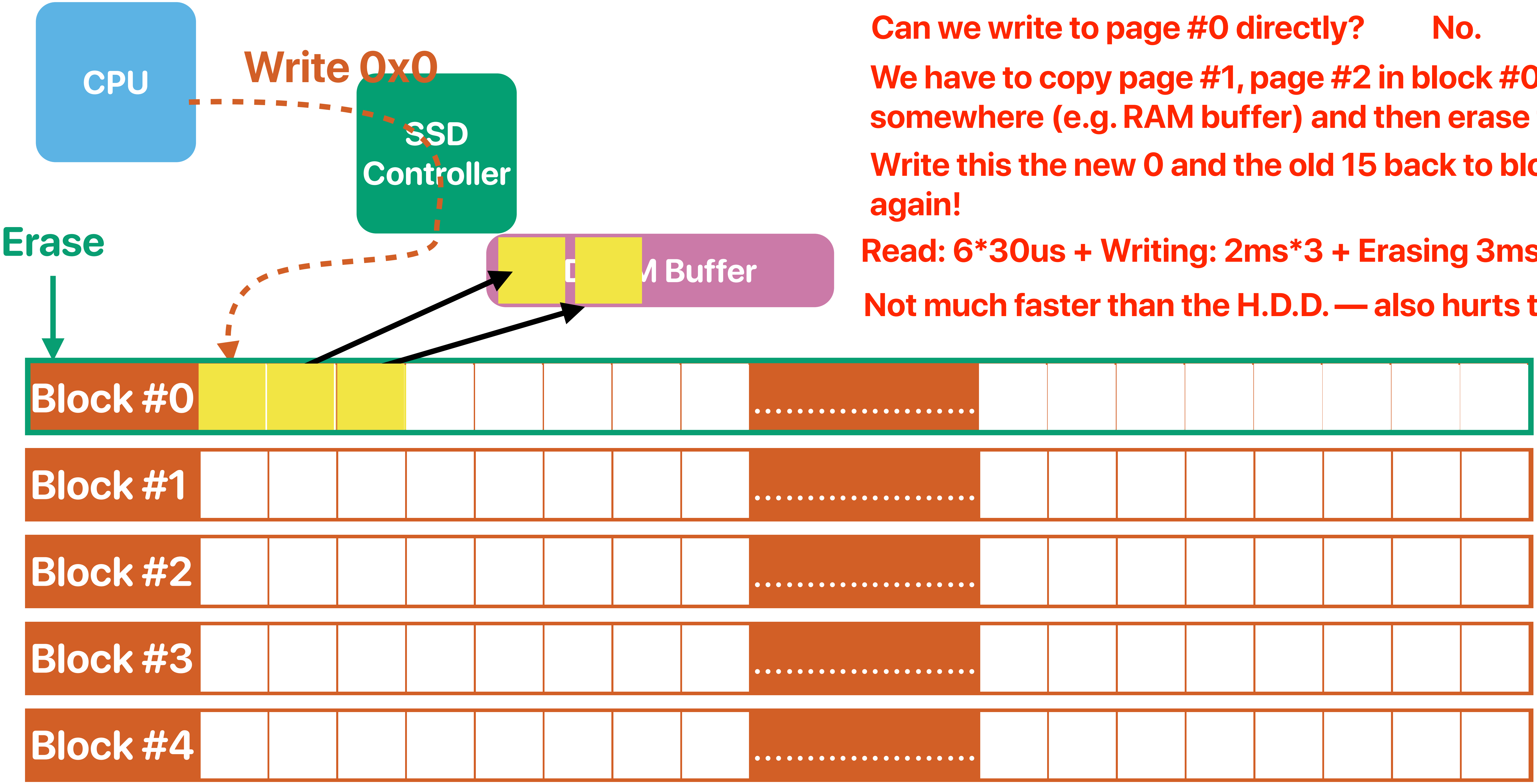
Similar relative performance for reads, writes and erases

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!

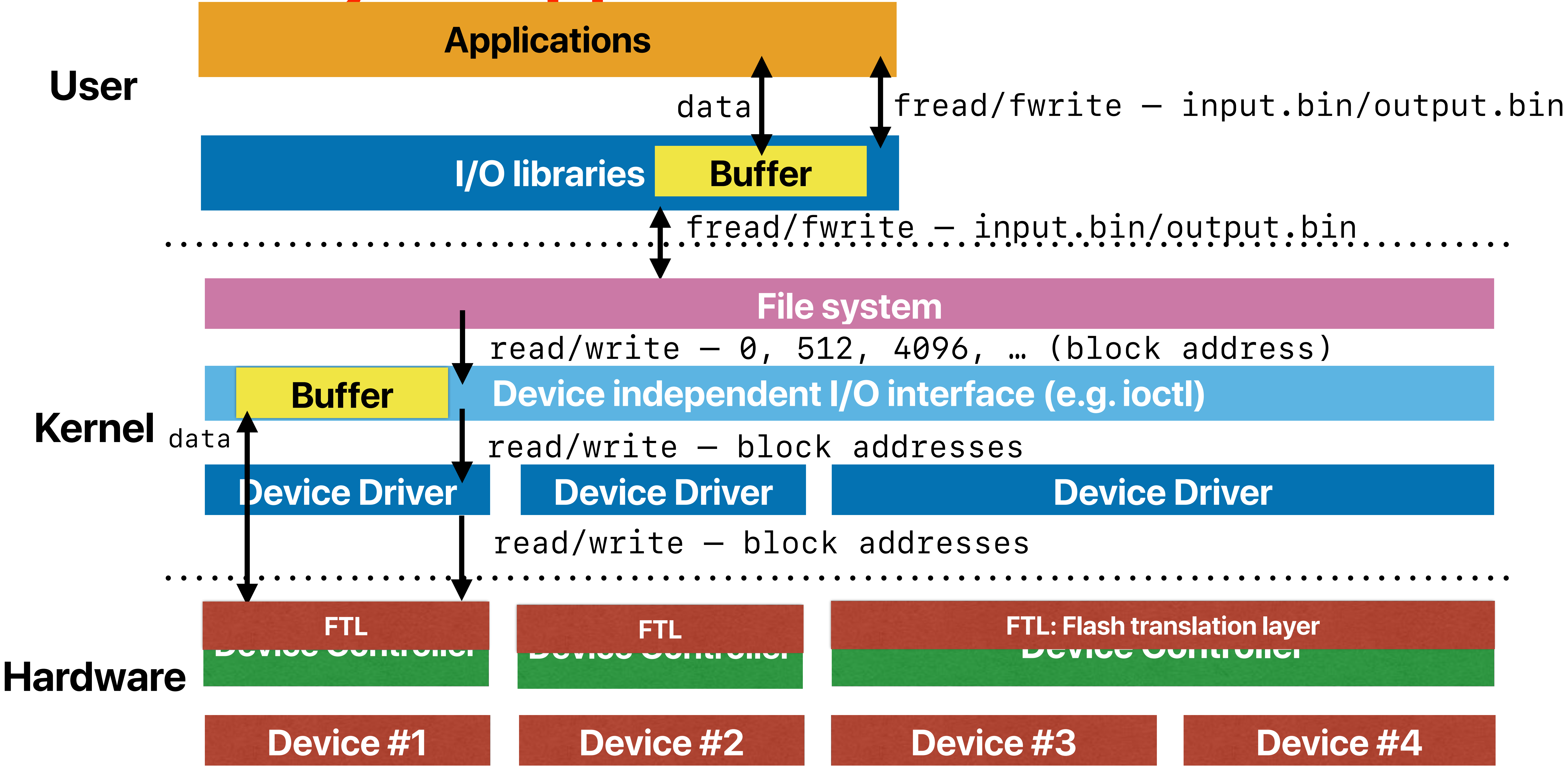
Read: 6*30us + Writing: 2ms*3 + Erasing 3ms ~ 9 ms

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

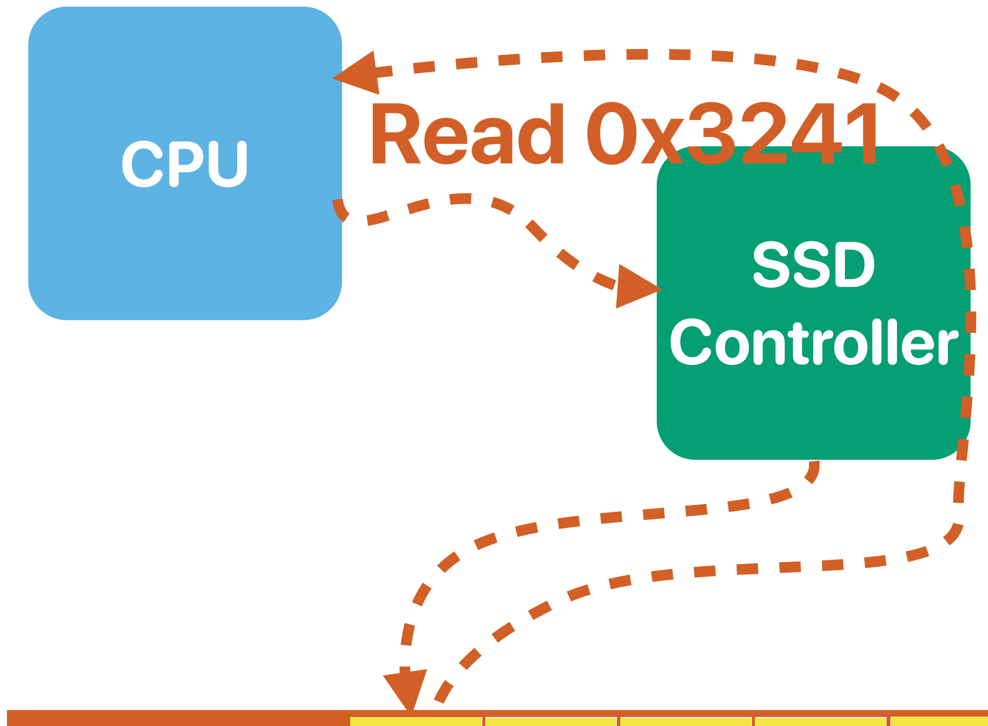
How your application reaches S.S.D.



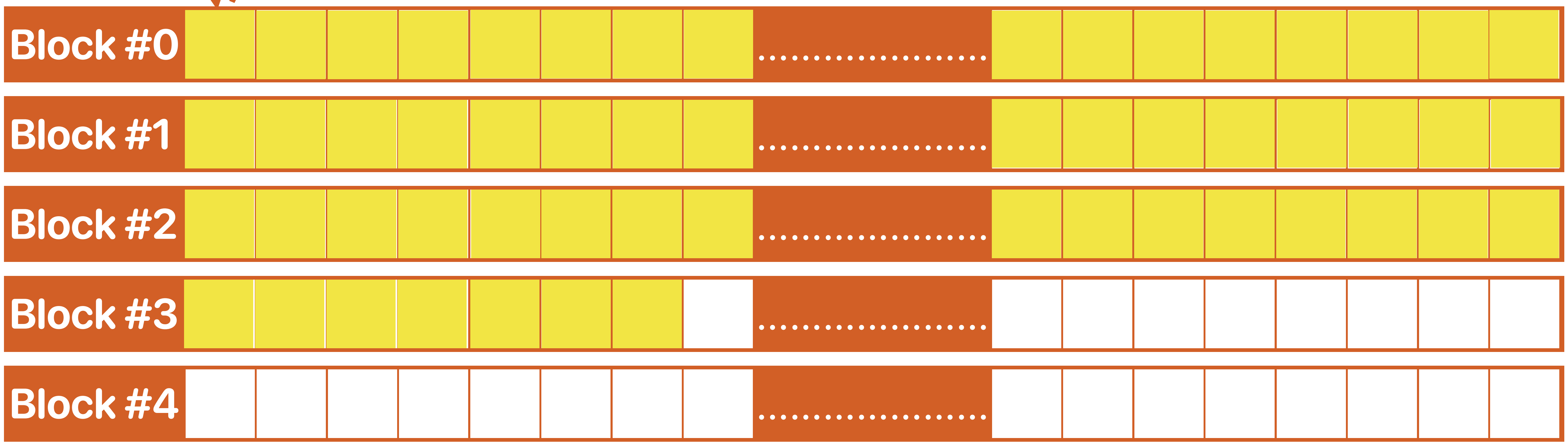
Flash Translation Layer (FTL)

- 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

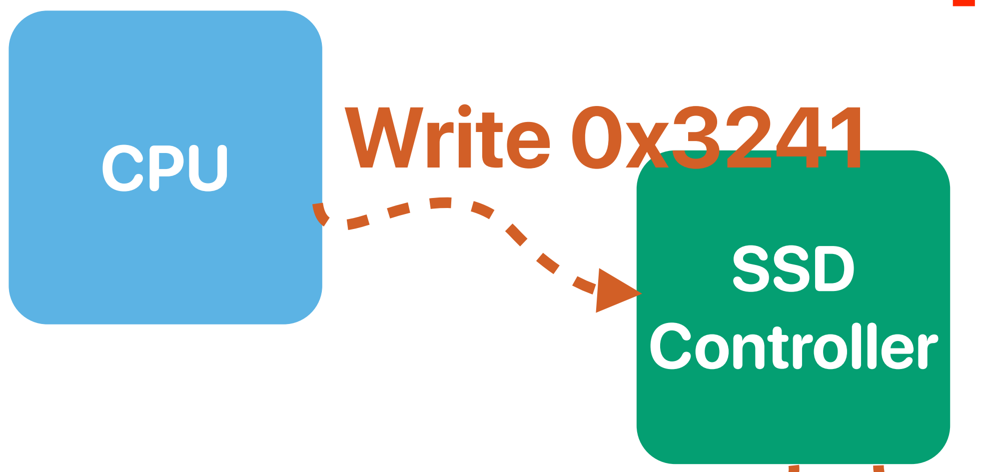
What happens on a read with FTL



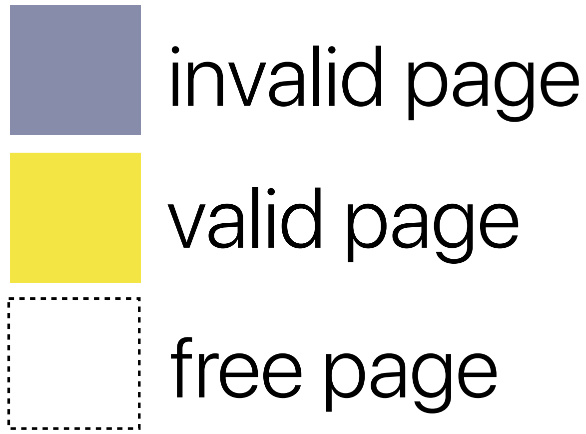
LBA	Flash Block	Flash Page
0x3241	0	0
0x3242	0	63
0x3243	1	3
0x3244	2	4
0x3245	3	6
0x3246	2	7

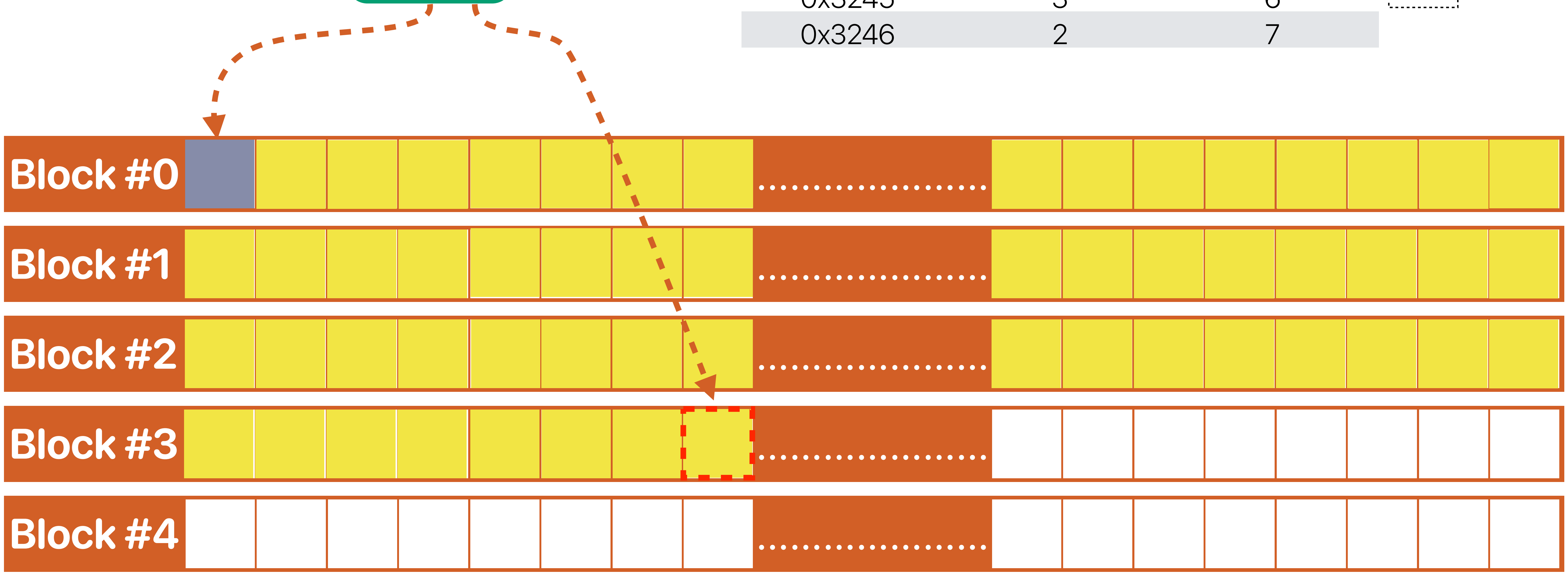


What happens on a write with FTL

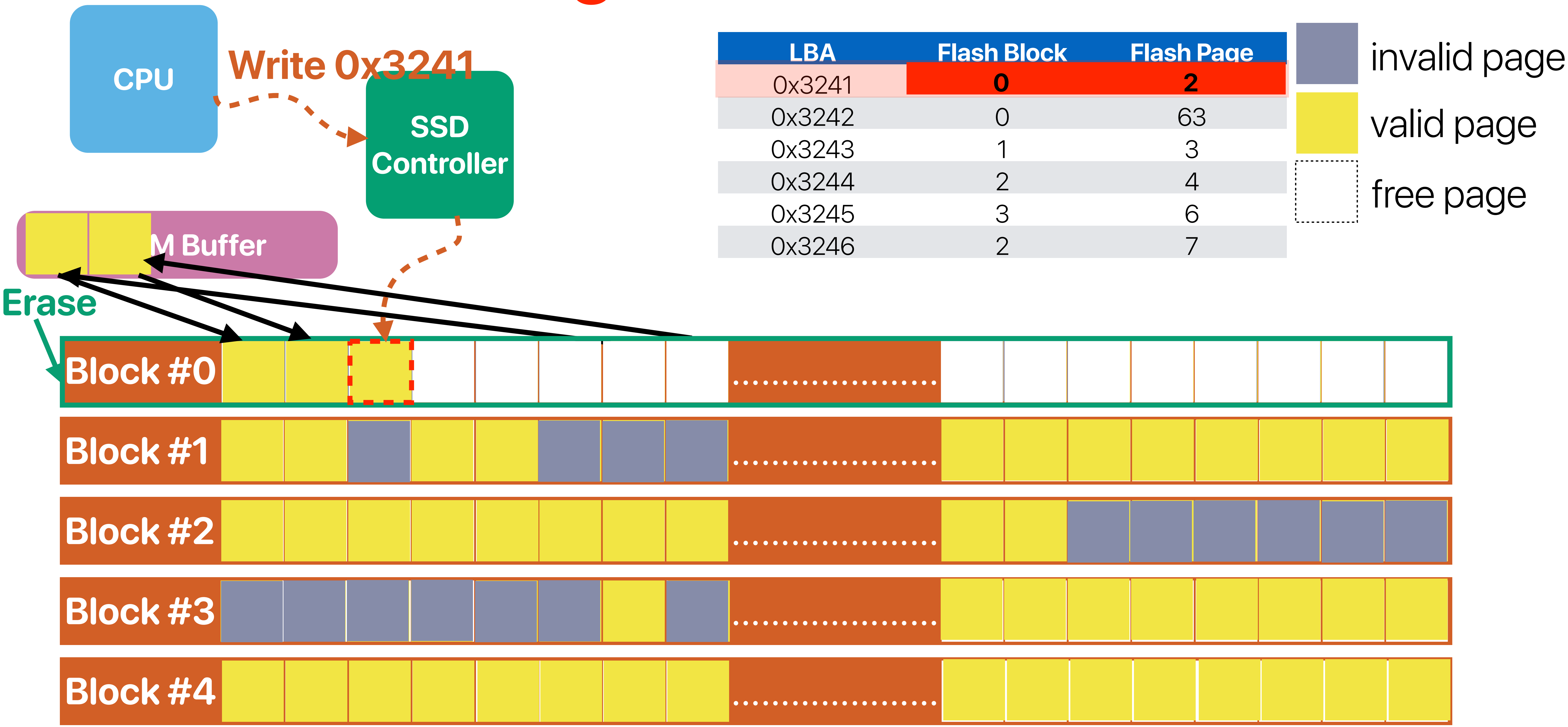


LBA	Flash Block	Flash Page
0x3241	**3**	**7**
0x3242	0	63
0x3243	1	3
0x3244	2	4
0x3245	3	6
0x3246	2	7





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

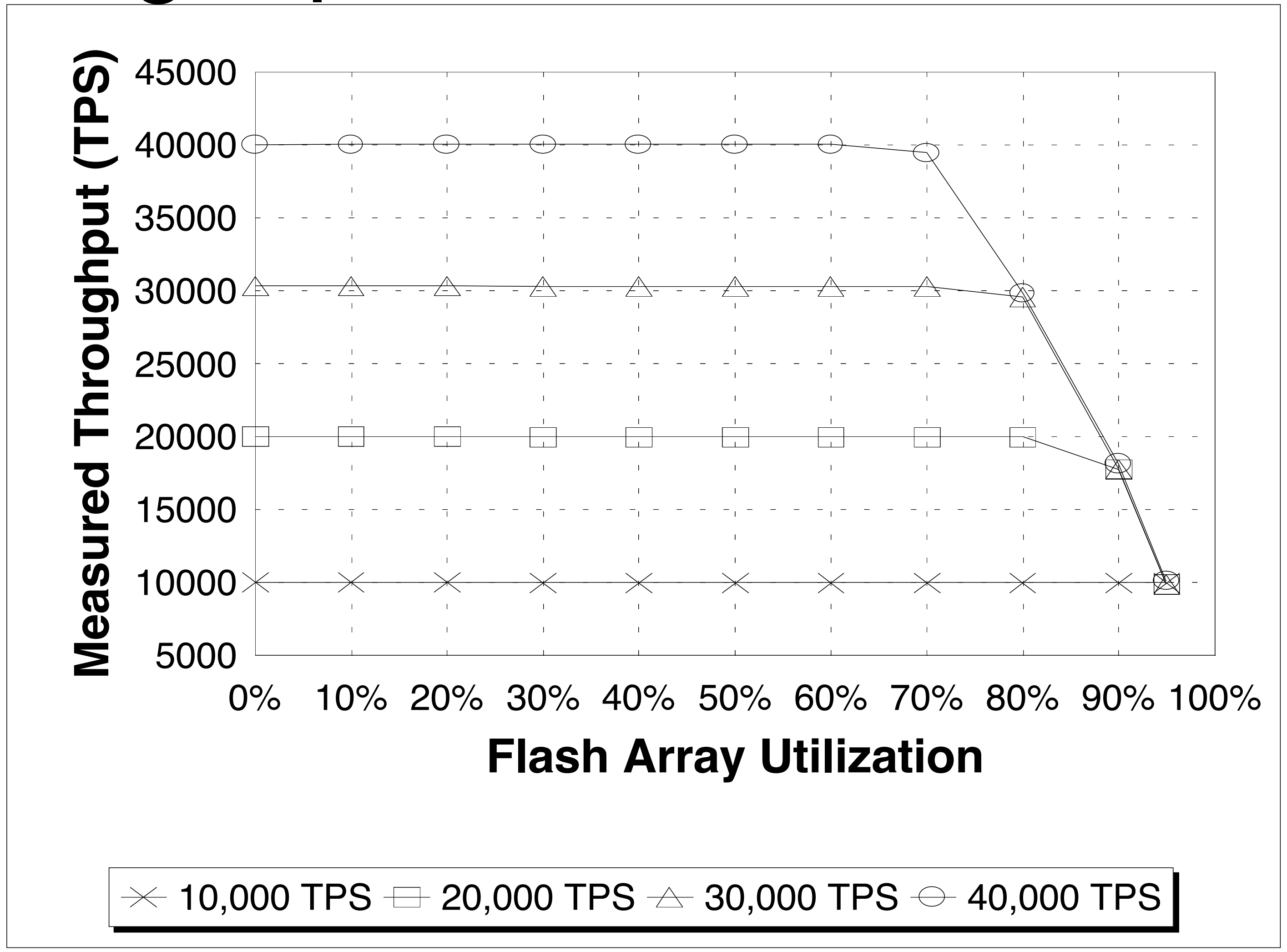
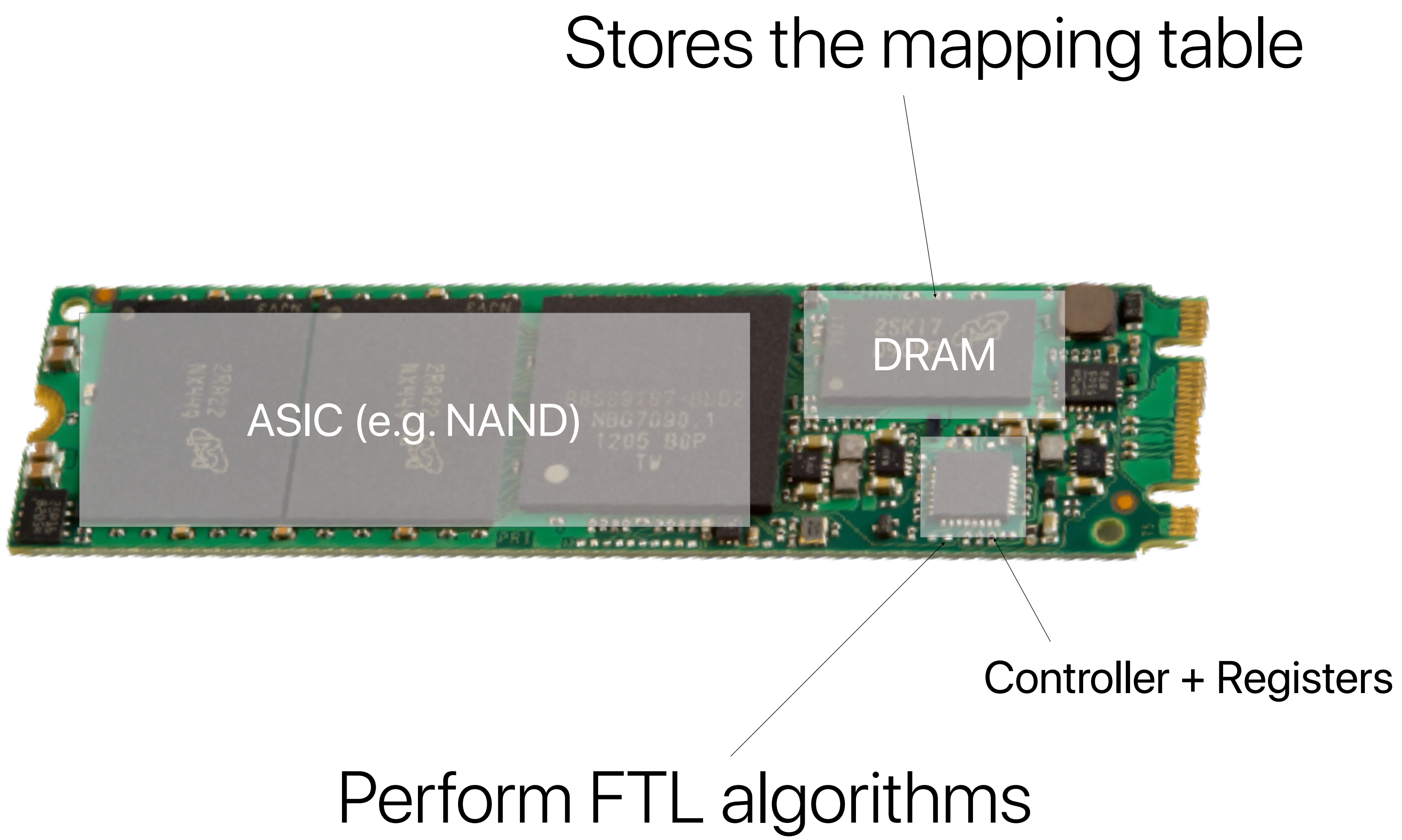


Figure 14: Throughput for Various Levels of Utilization

The impact of eNVy

- Your SSD structured exactly like this!

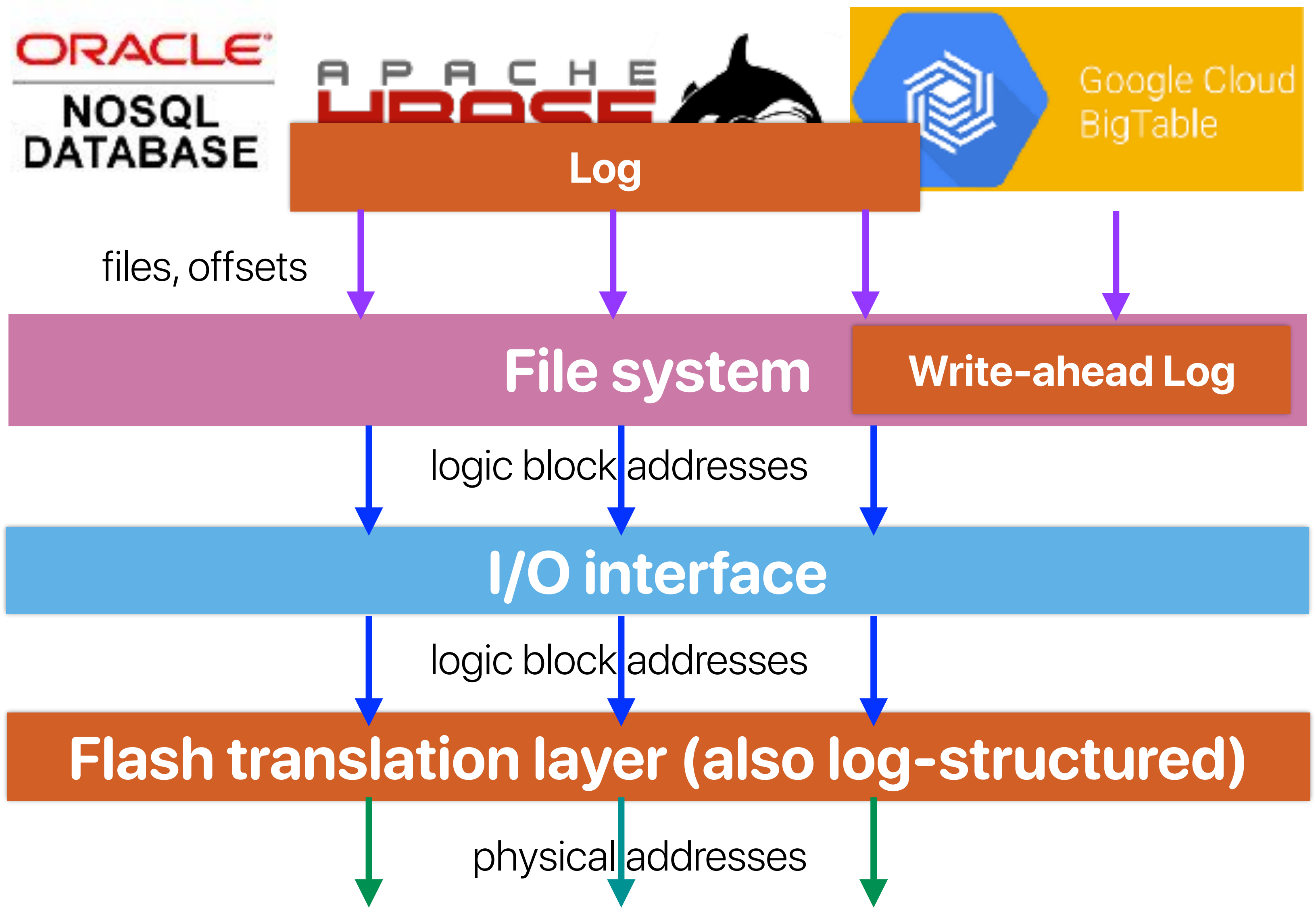


Don’t stack your log on my log

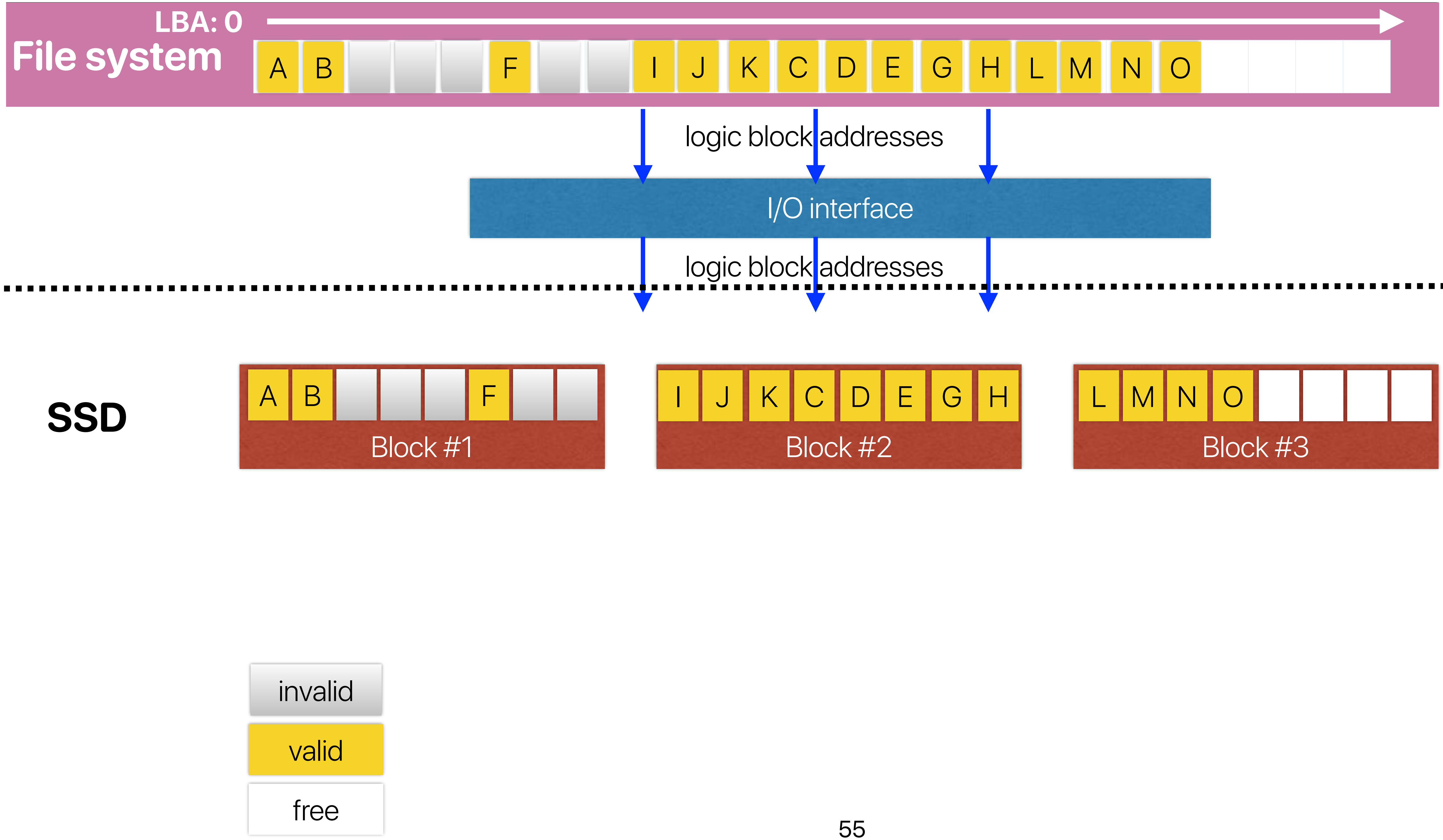
Jingpei Yang, Ned Plasson, Greg Gillis, Nisha Talagala, and Swaminathan Sundararaman
SanDisk Corporation

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



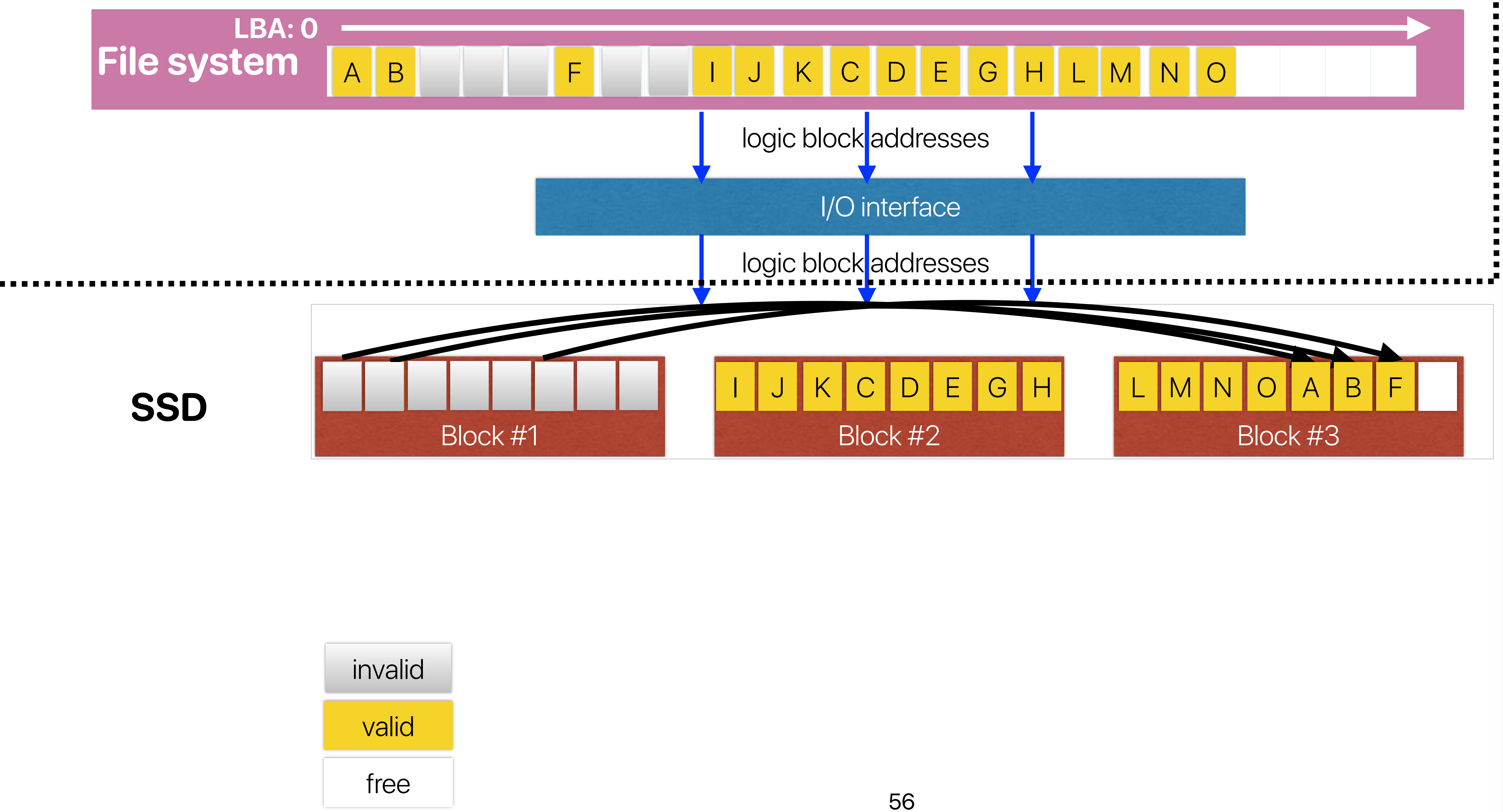
For example, garbage collection



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	-	-

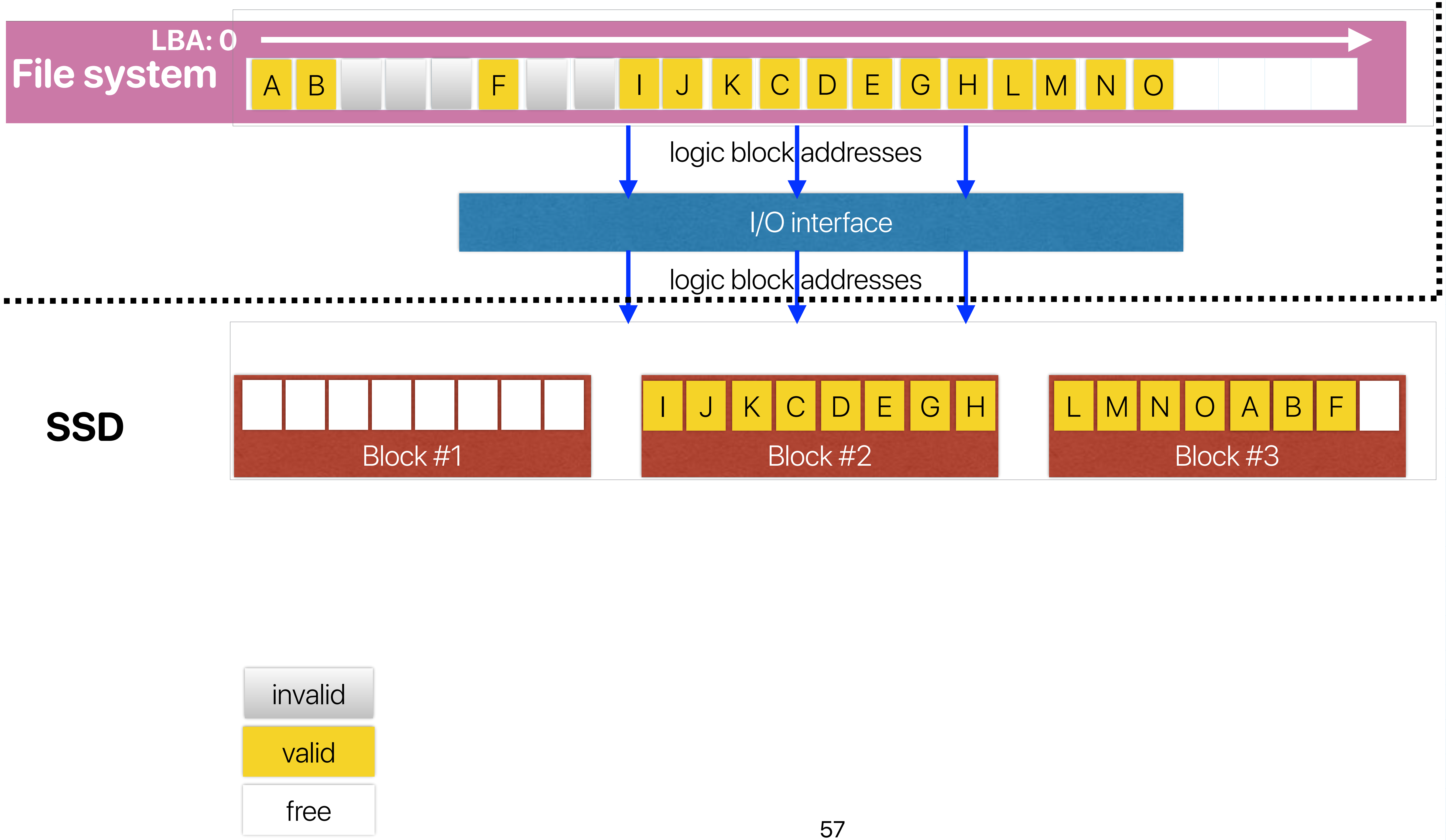
Now, SSD wants to reclaim a block



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	-	-

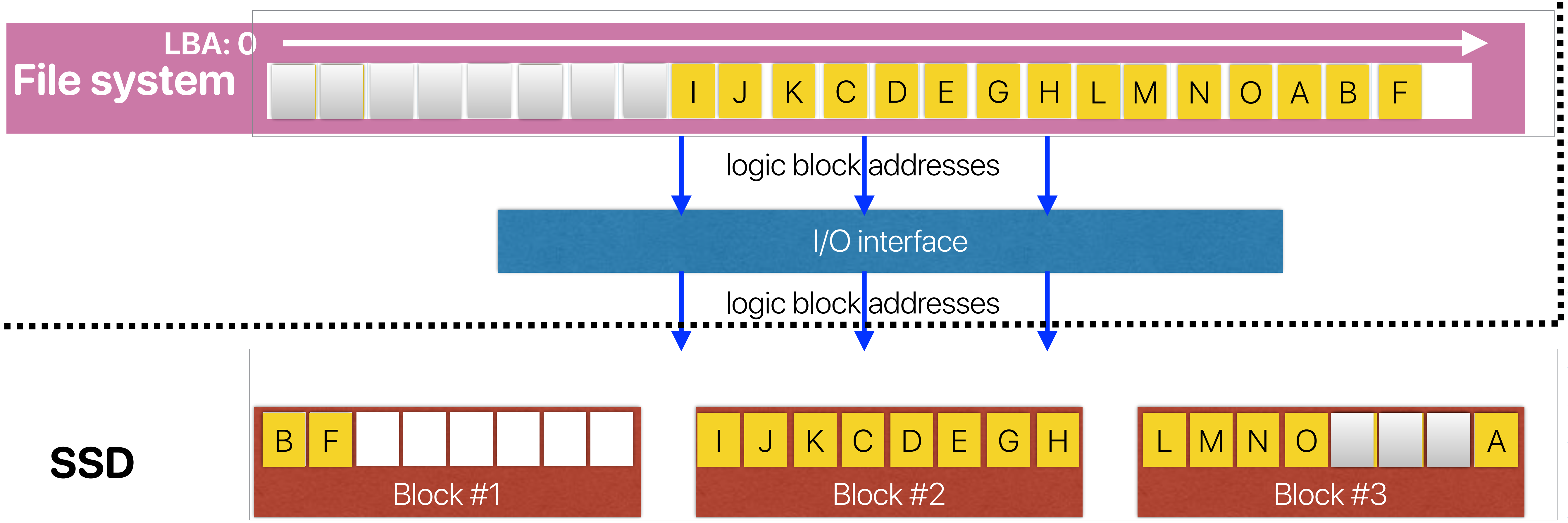
Garbage collection on the SSD done!



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?



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	-	-

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

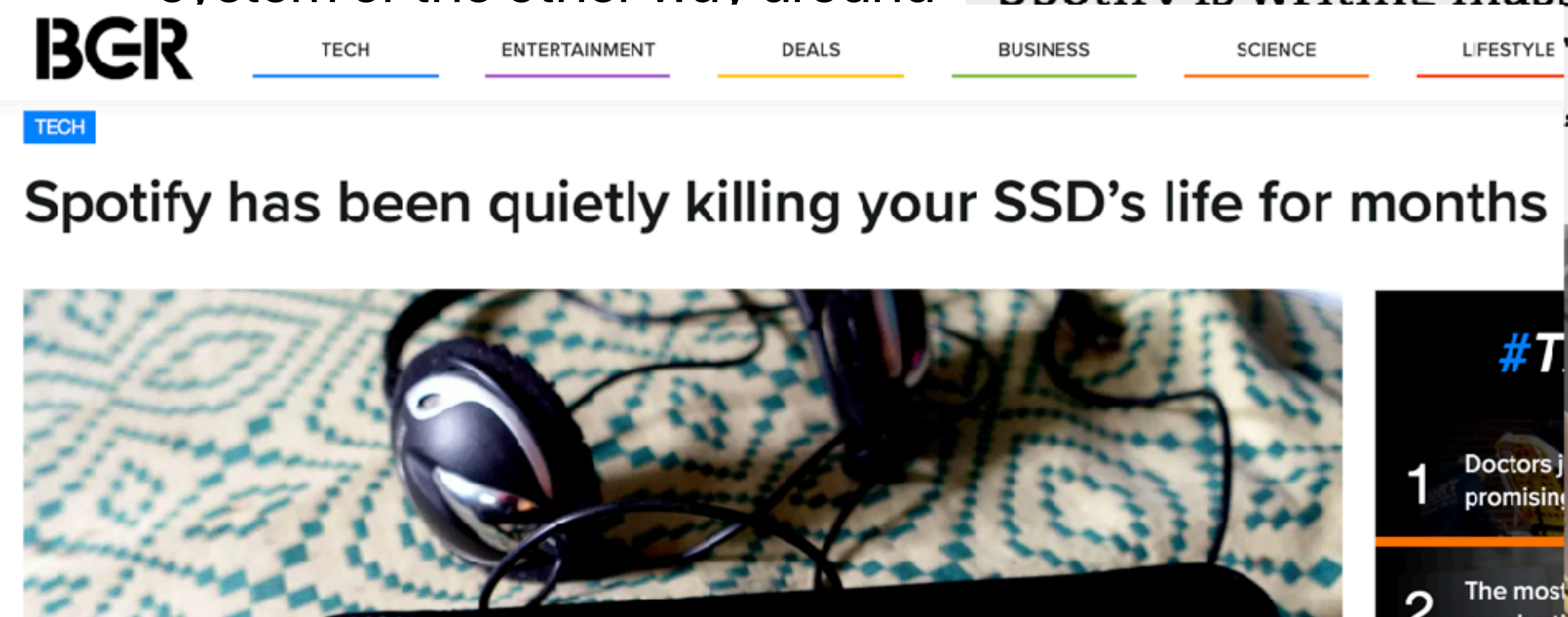
All problems in computer science can be solved by another level of indirection

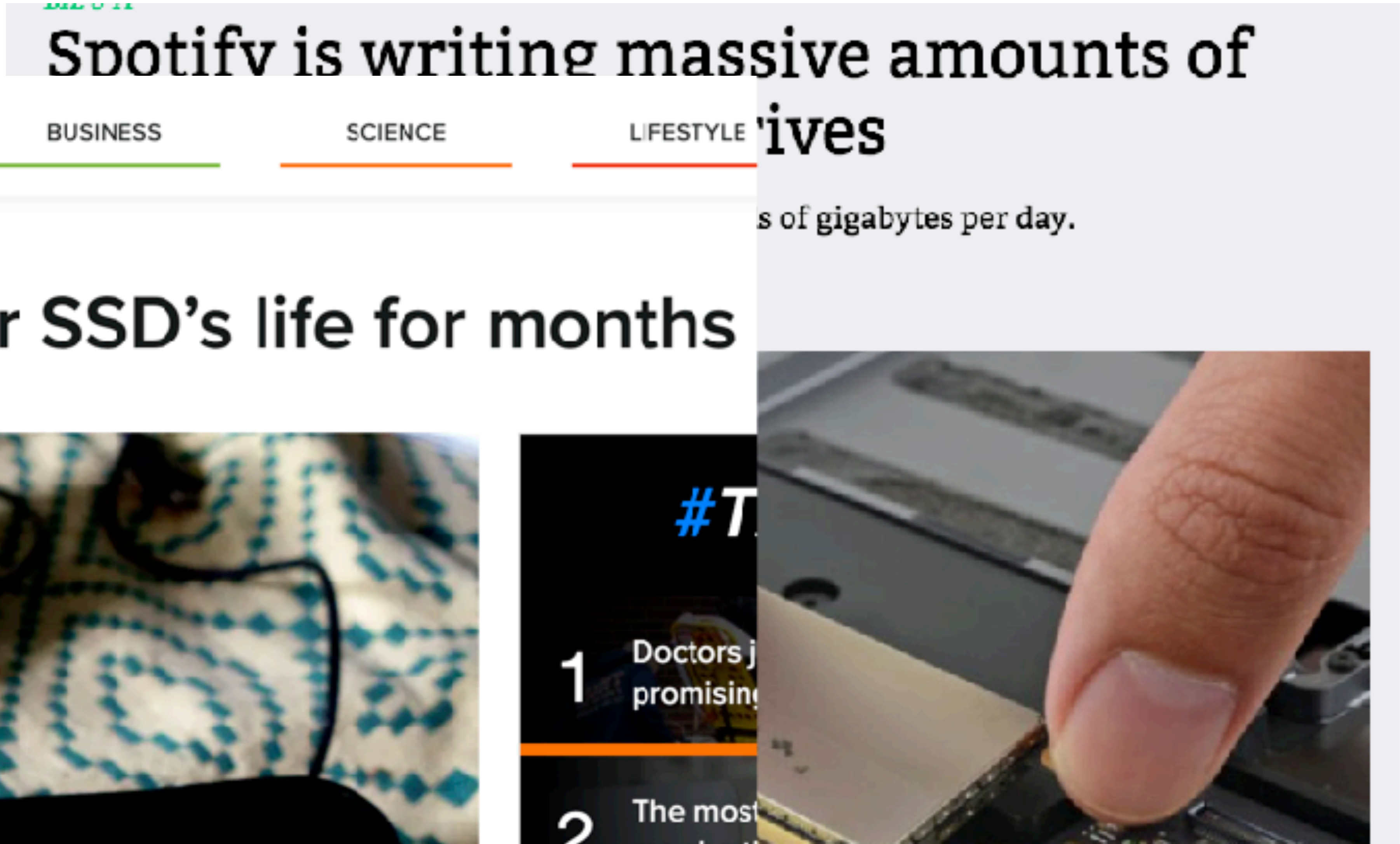
–David Wheeler

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

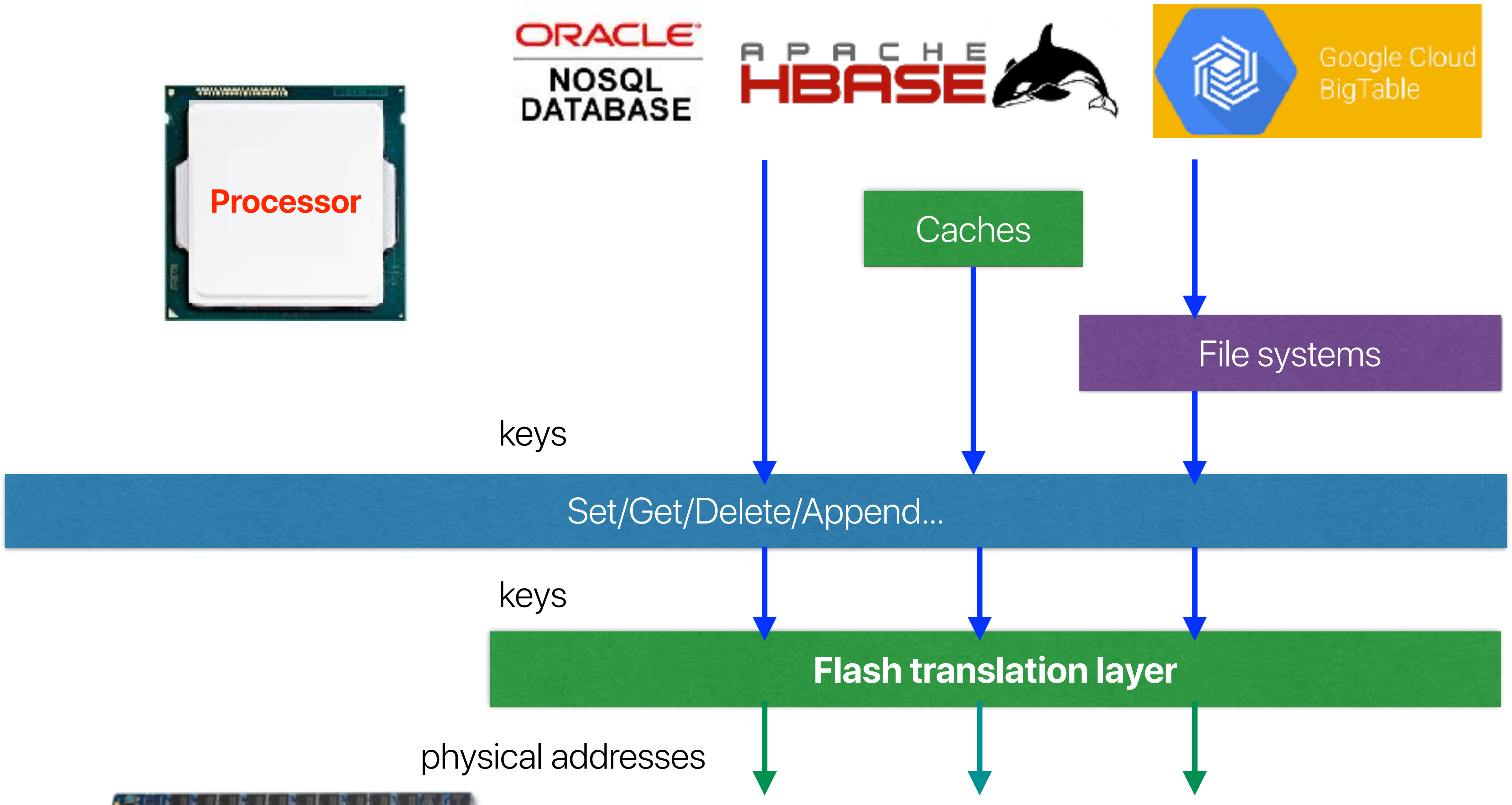
File systems for flash-based SSDs

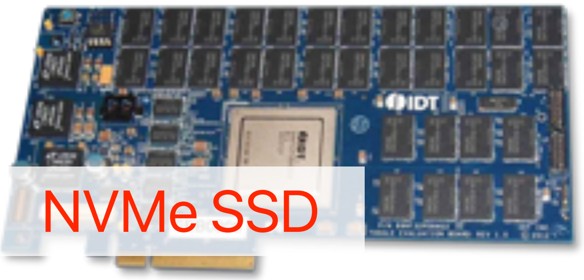
- Still an open research question
- Software designer should be aware of the characteristics of underlying hardware components
- Revising the layered design to expose more SSD information to the file system or the other way around





KAML: Modernize the storage interface❖





❖ **Yanqin Jin, Hung-Wei Tseng, Steven Swanson and Yannis Papakonstantinou. KAML: A Flexible, High-Performance Key-Value SSD. In HPCA 2017.**