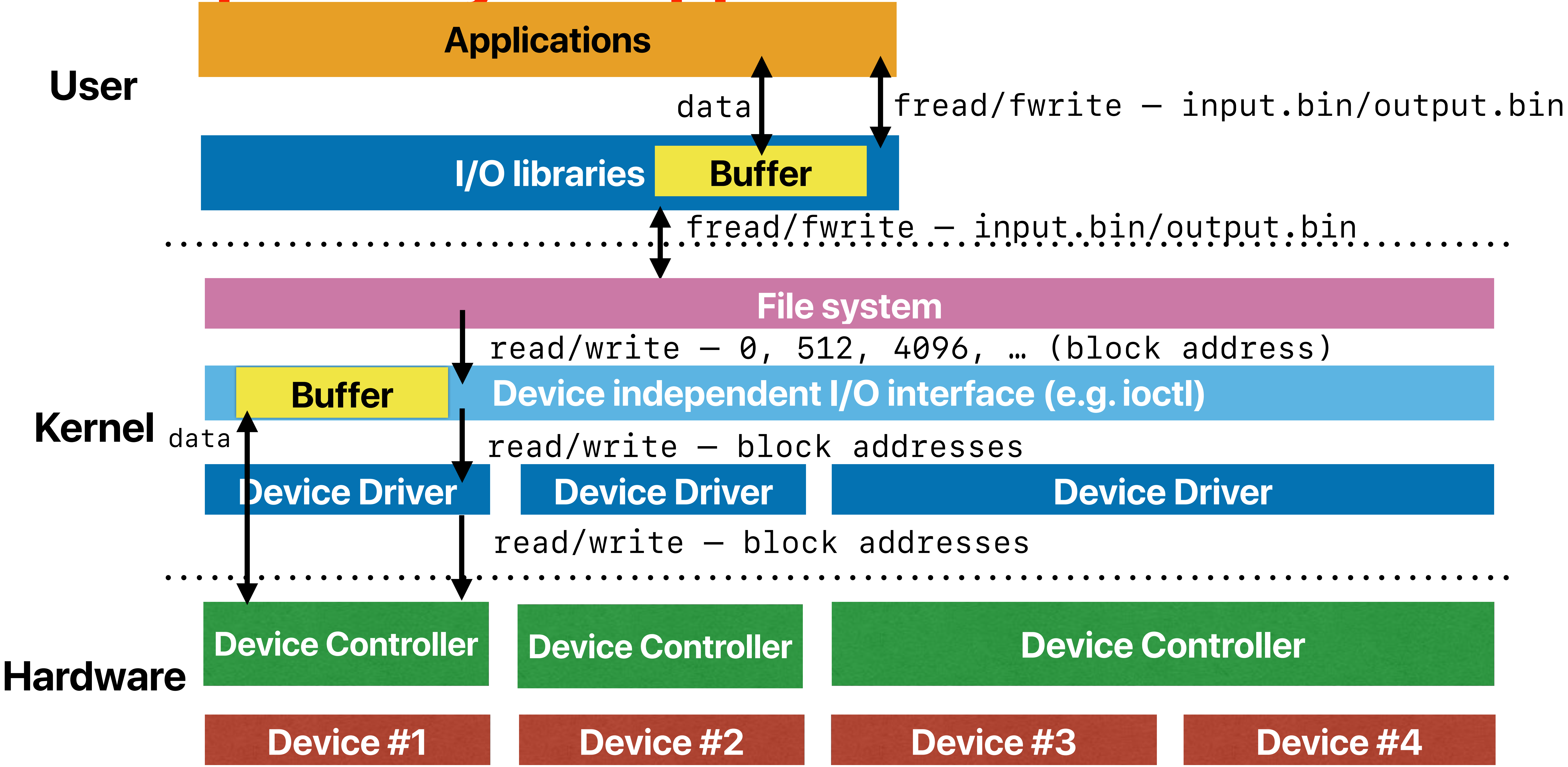
File Systems & The Era of Flash-based SSD

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!

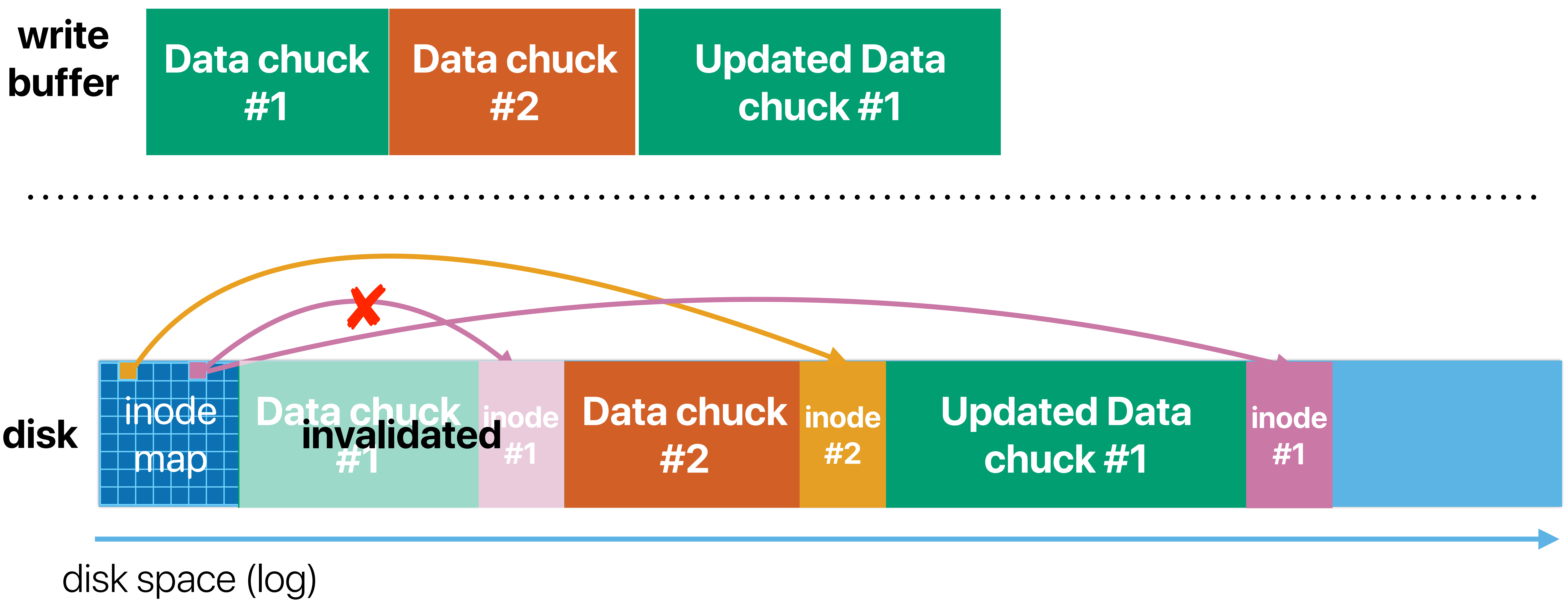
Recap: Why LFS?

- Writes will dominate the traffic between main memory and disks — Unix FFS is designed under the assumption that a majority of traffic is large files
  - 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

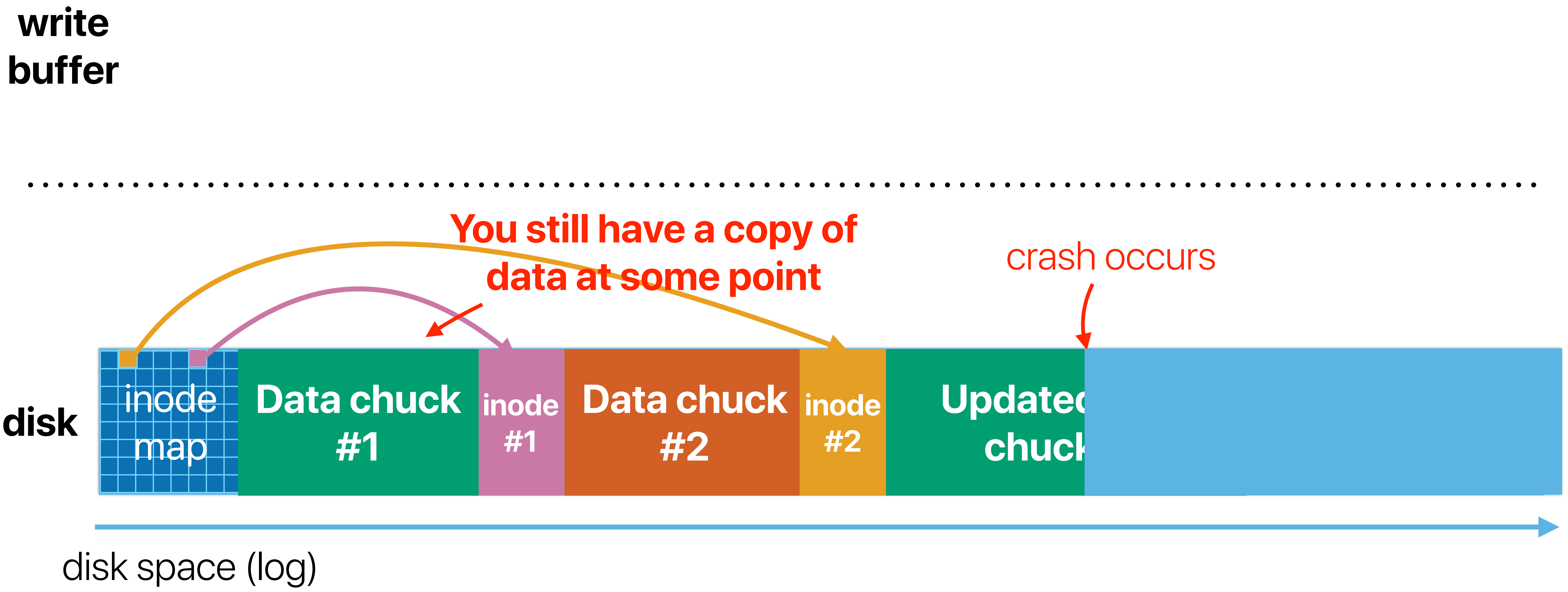
Recap: What does LFS propose?

- 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

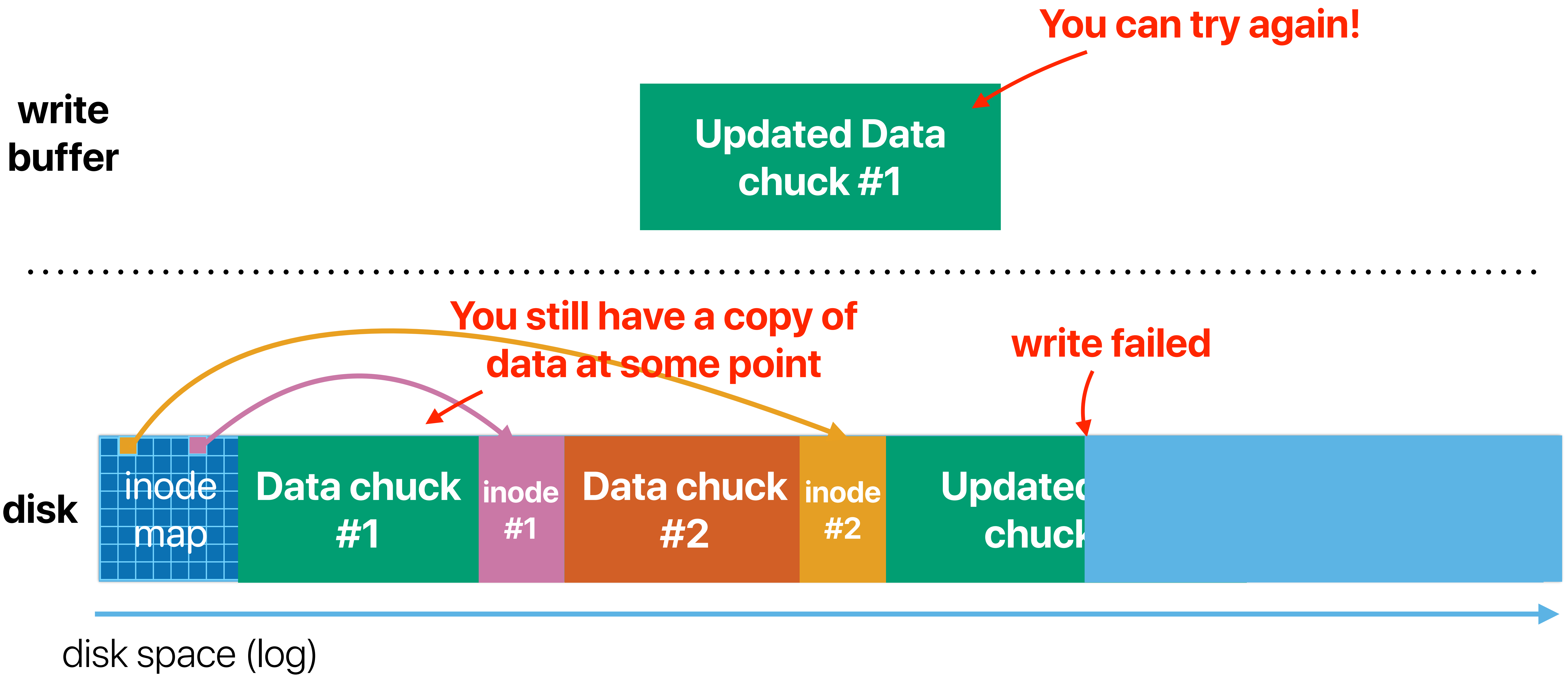


LFS v.s. crash



write buffer
disk
inode map
Data chuck #1
inode #1
Data chuck #2
inode #2
Updated chuck
You still have a copy of data at some point
crash occurs
disk space (log)

LFS v.s. write failed



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

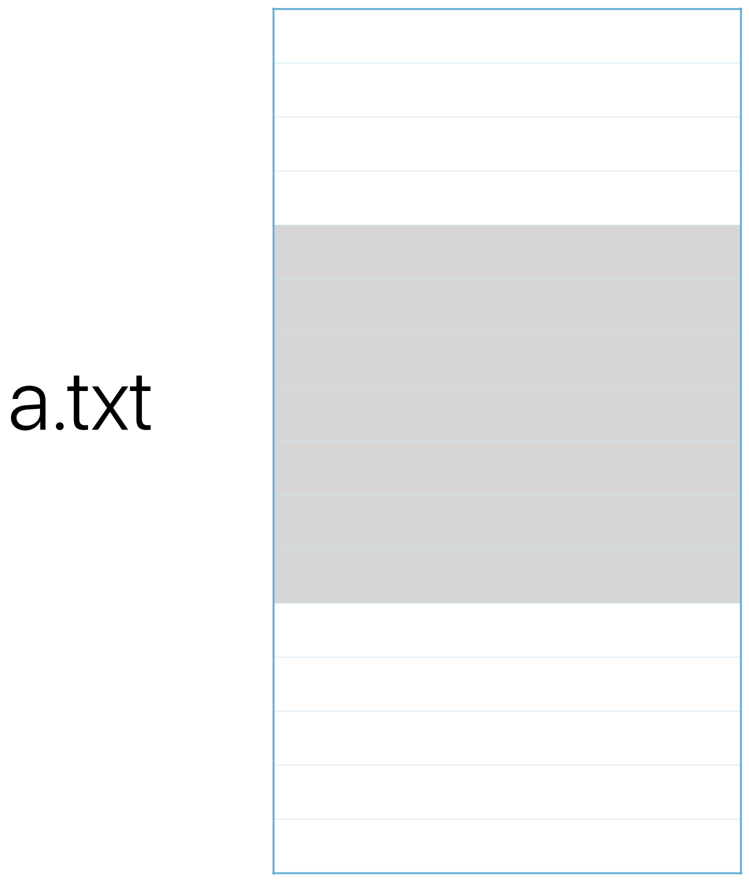
Outline

- 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

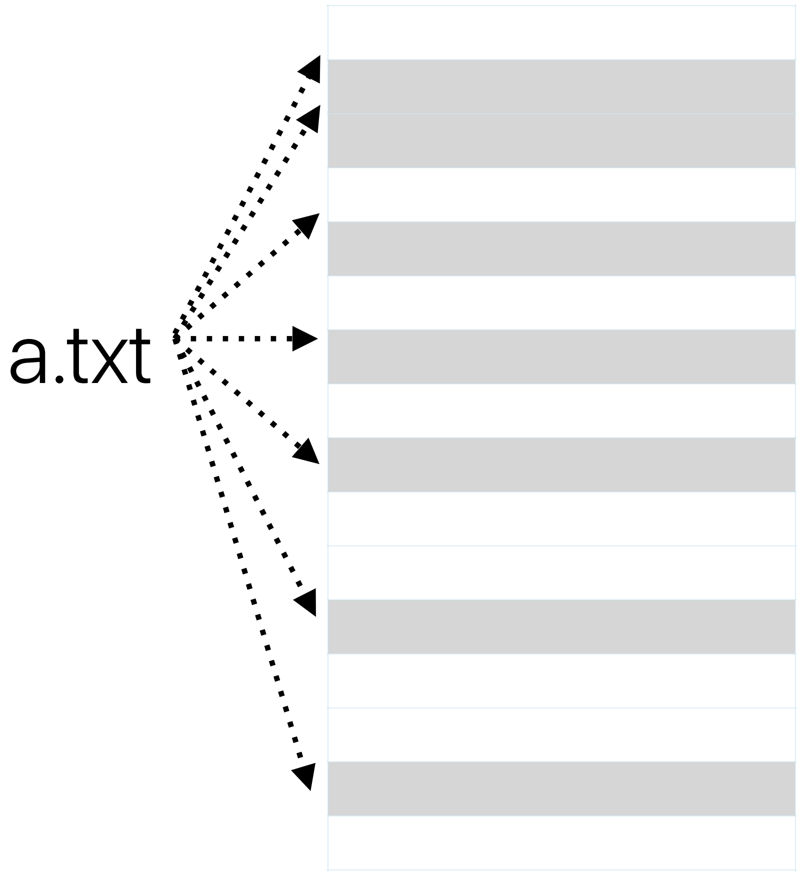
How do we allocate disk space?

- Contiguous: the file resides in continuous addresses

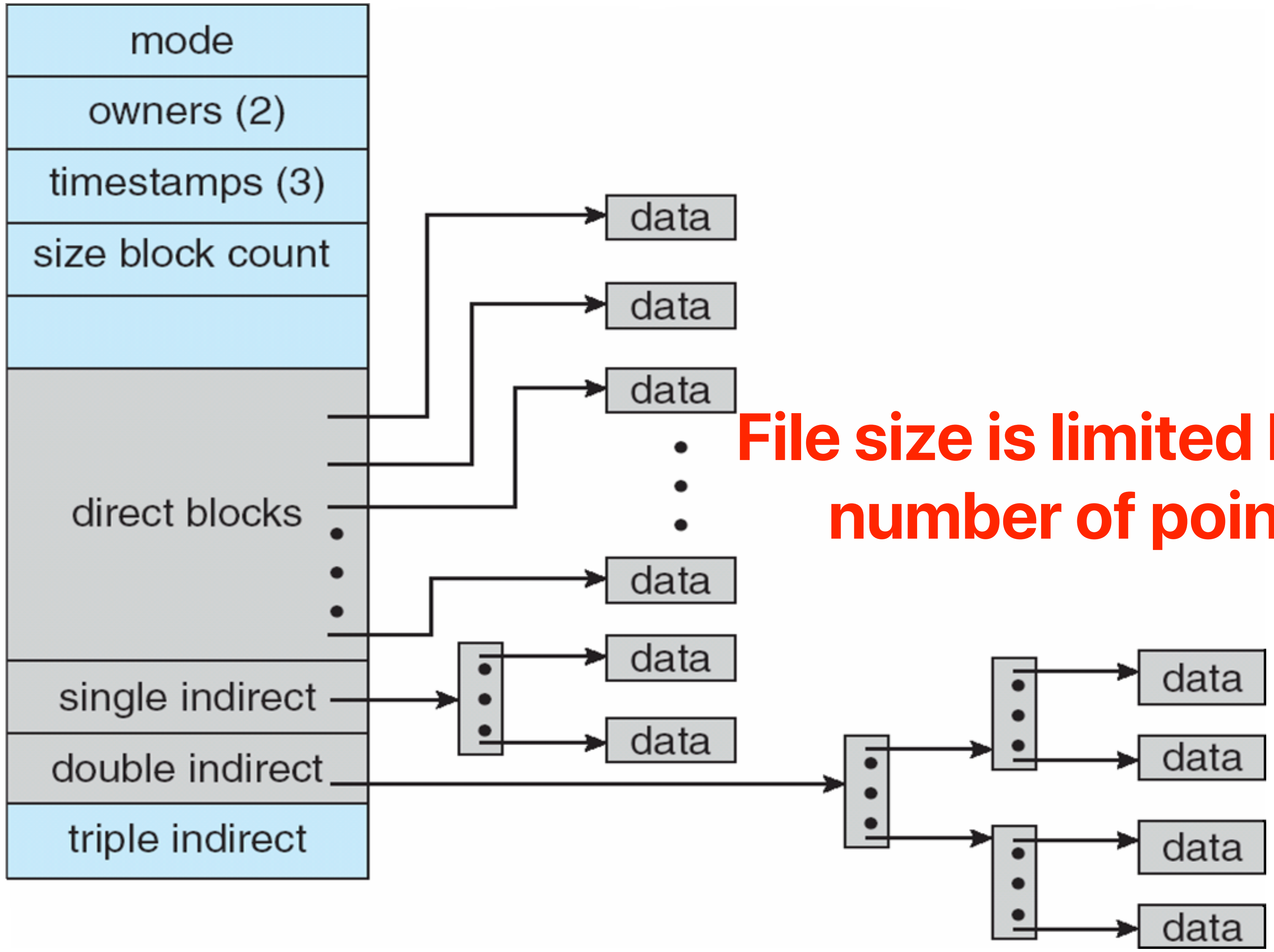


external fragment as in Segmentation

- Non-contiguous: the file can be anywhere



Conventional Unix inode

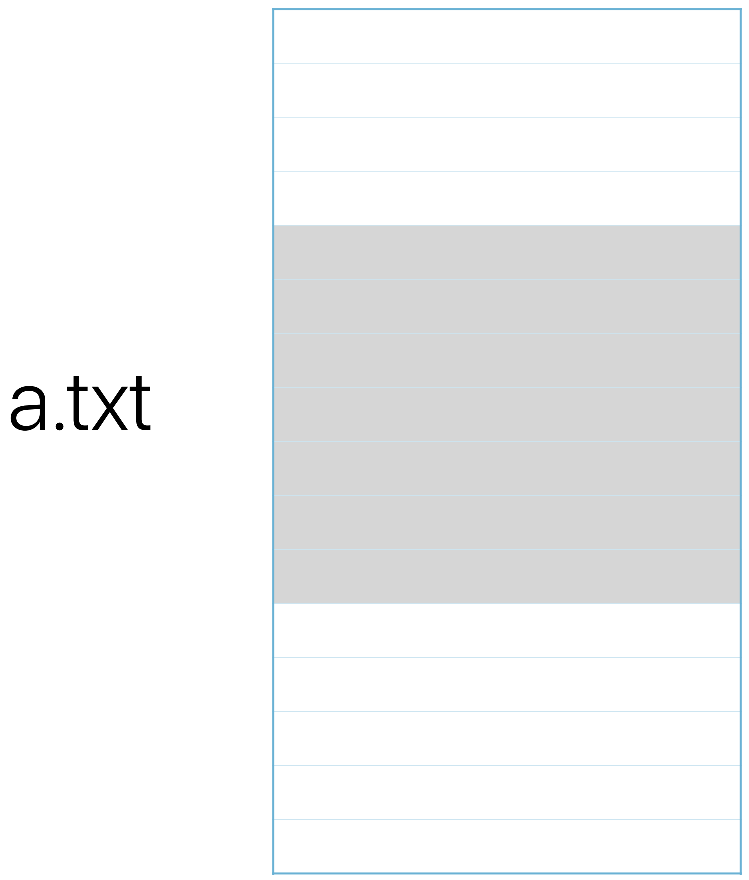


File size is limited by total number of pointers

- 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

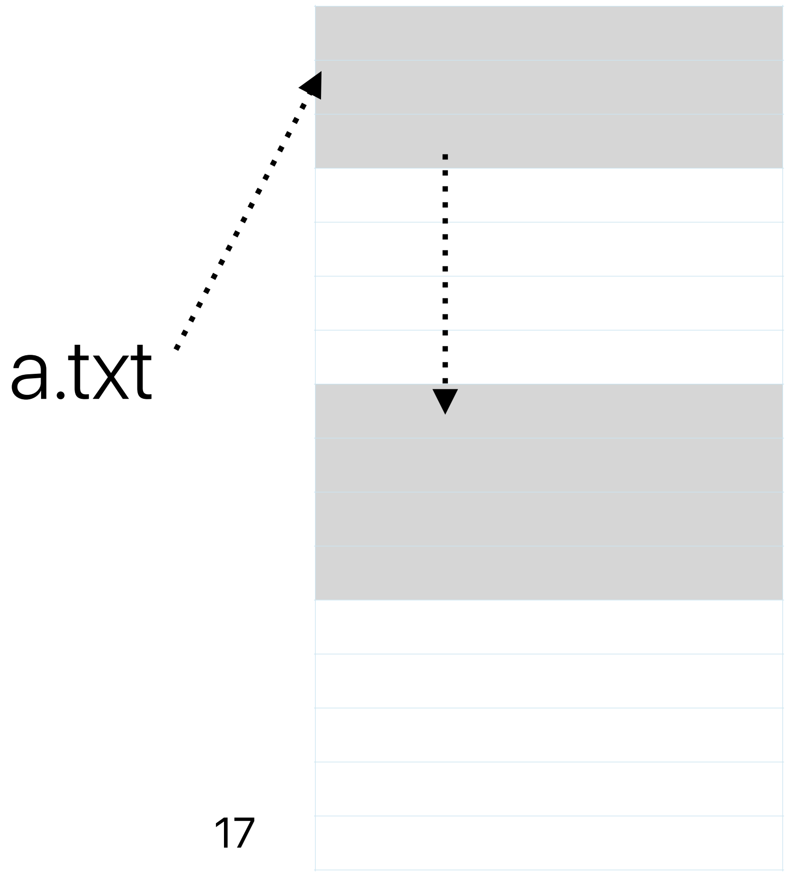
How do we allocate space?

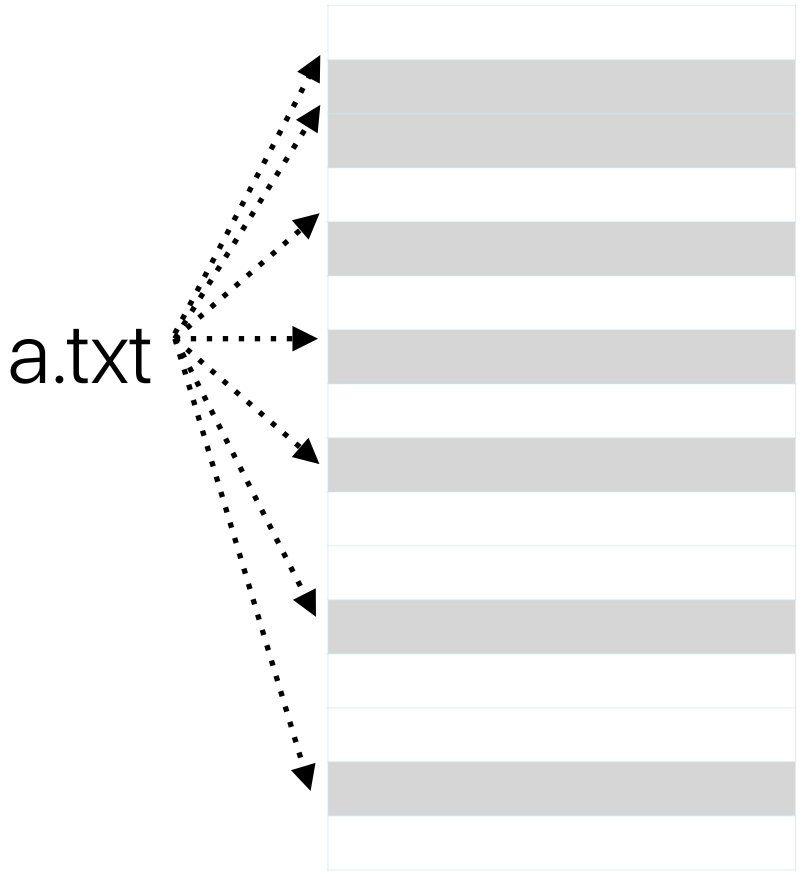
- Contiguous: the file resides in continuous addresses



- 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

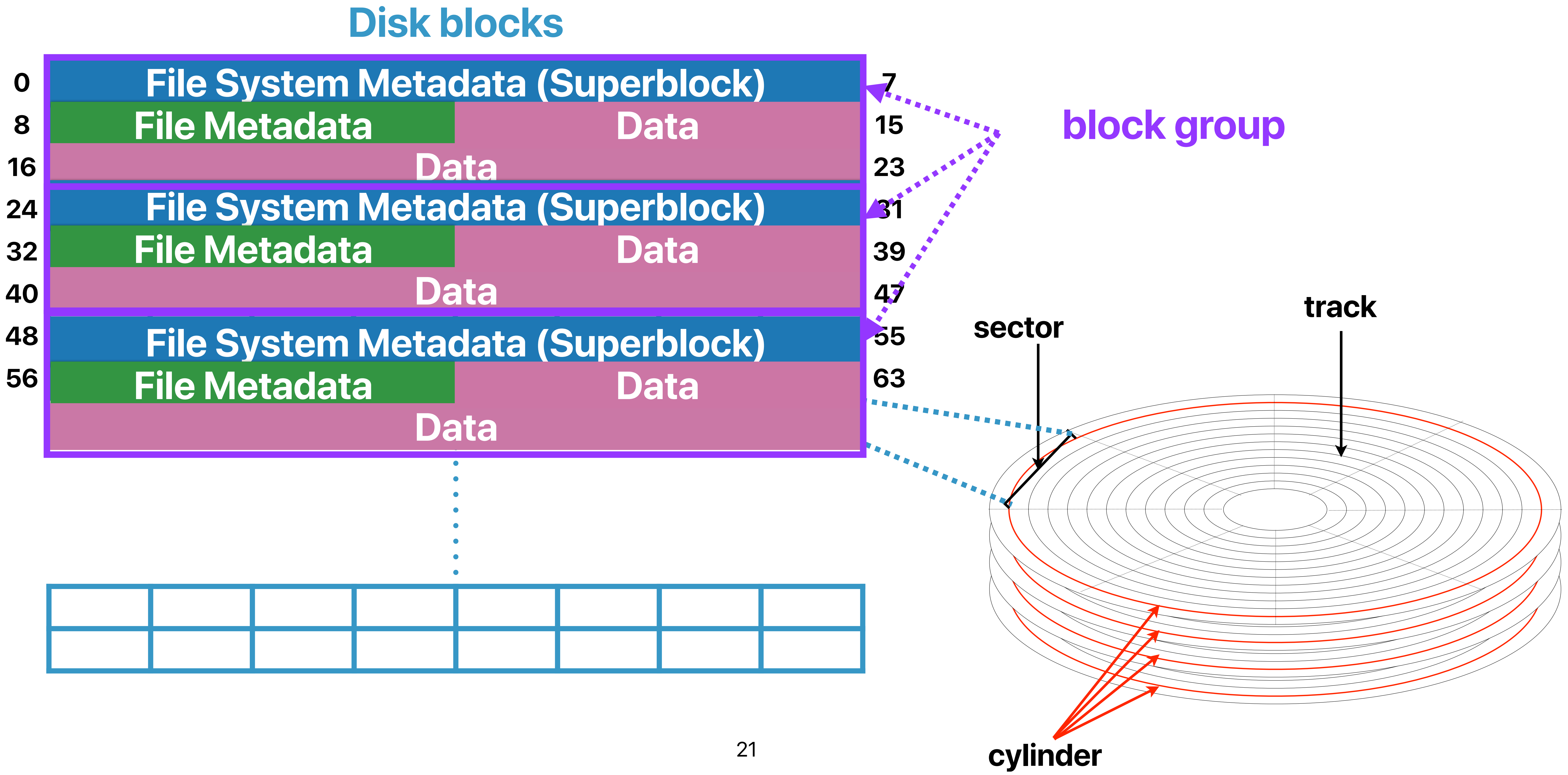
Extent file systems — ext2, ext3, ext4

- 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

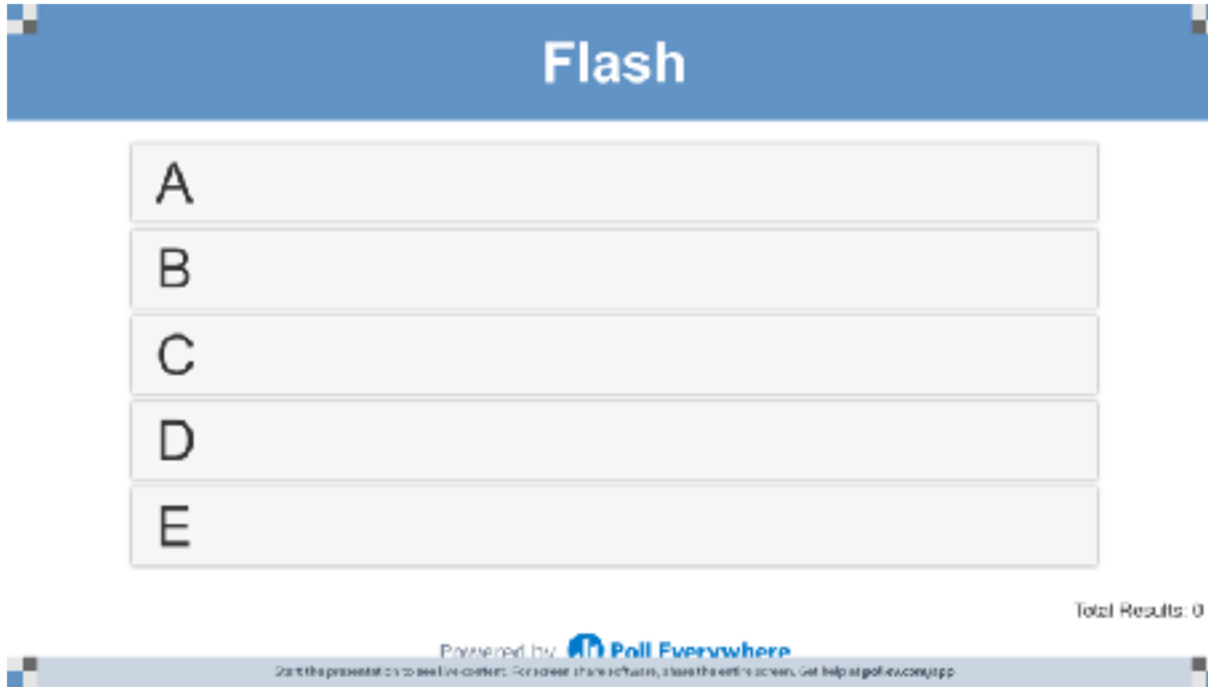
Michael Wu and Willy Zwaenepoel
Rice University

https://www.pollev.com/hungweitseng close in 1:00

The characteristics of flash memory

- Regarding the flash memory technology described in eNVy, how many of the following is/are true
  1. The write speed of flash memory can be 100x slower than reading flash
  2. The granularities of writing and erasing are different
  3. Flash memory cannot be written again without being erased
  4. The flash memory chip has limited number of erase cycles

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



The characteristics of flash memory

- Regarding the flash memory technology described in eNVy, how many of the following is/are true
  1. The write speed of flash memory can be 100x slower than reading flash

     Writes are slow
  2. The granularities of writing and erasing are different

     You can only program/write in the unit of a page (e.g. 4K), but erases must be perform by blocks (e.g. 128 pages)
  3. Flash memory cannot be written again without being erased

     In-place update needs to erase the block first
  4. The flash memory chip has limited number of erase cycles

     You cannot erase too often

  A. 0

  B. 1

  C. 2

  D. 3

  E. 4

Feature	Disk	DRAM	Low Power SRAM	Flash
Read Access	8.3ms	60ns	85ns	85ns
Write Access	8.3ms	60ns	85ns	4 – 10 microsec.
Cost/MByte	$1.00	$35.00	$120	$30.00
Data Retention Current/GByte	0A	1A	2mA	0A

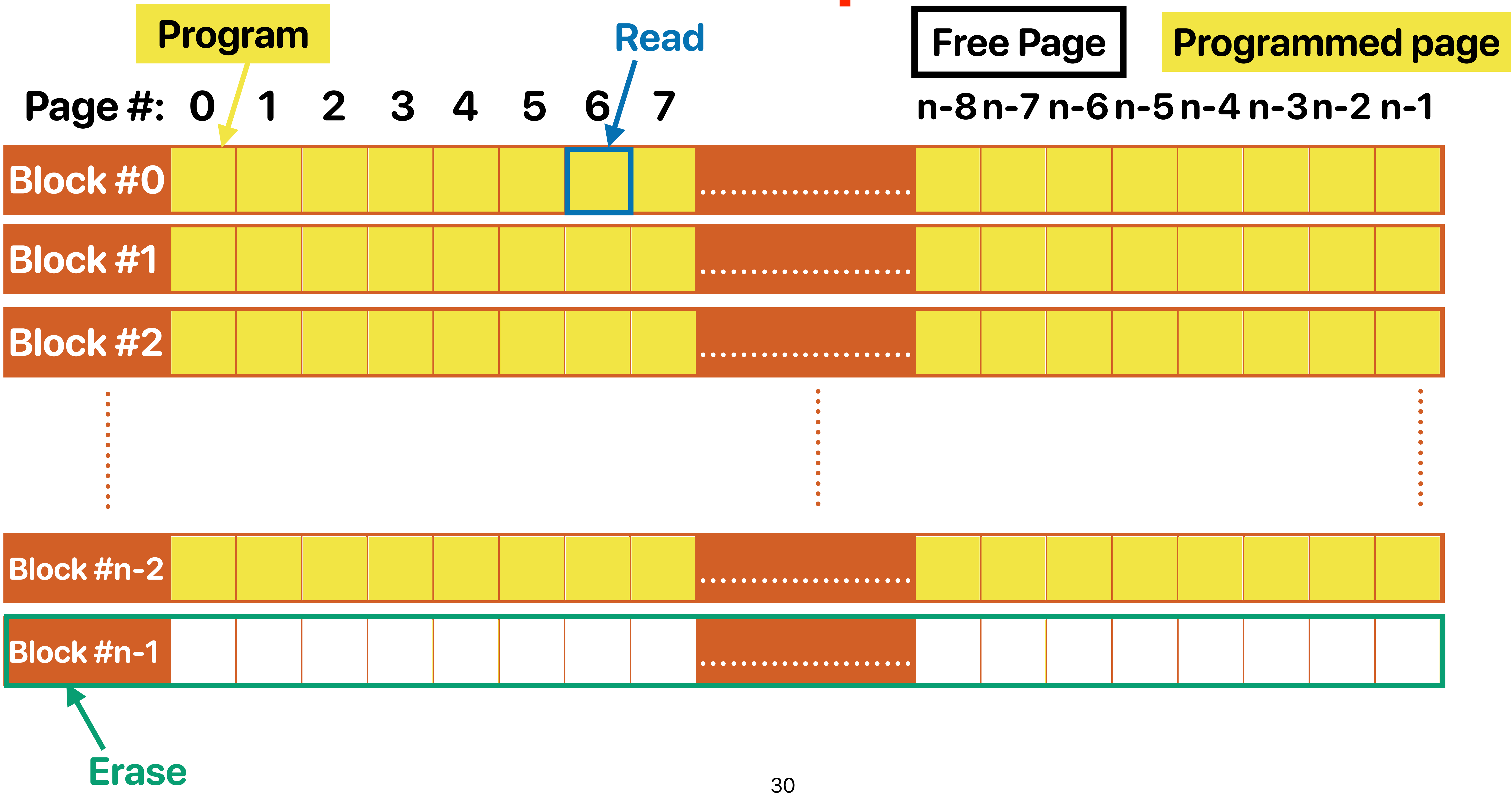
like access times (under 100ns). Individual bytes can be programmed in 4 to 10μs but cannot be arbitrarily rewritten until the entire device is erased, which takes about 50ms. Newer Flash chips allow some flexibility

ory. Furthermore, updates to Flash memory are much slower than updates to conventional memory, and the number of program-erase cycles is limited.

Flash memory: eVNy and now

	Modern SSDs	eNVy
Technologies	NAND	NOR
Read granularity	Pages (4K — 16K)	Supports byte accesses
Write/program granularity	Pages (4K — 16K)	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

2 voltage levels, 1-bit

4 voltage levels, 2-bit

8 voltage levels, 3-bit

16 voltage levels, 4-bit

Single-Level Cell (SLC)

Multi-Level Cell (MLC)

Triple-Level Cell (TLC)

Quad-Level Cell (QLC)

Programming in MLC

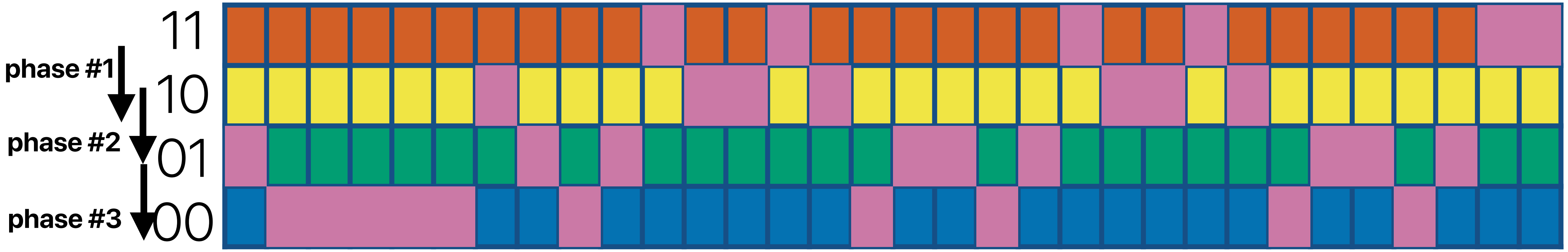
4 voltage levels, 2-bit

Multi-Level Cell (MLC)

3.14000000000000012434497875802

= 0x40091EB851EB851F

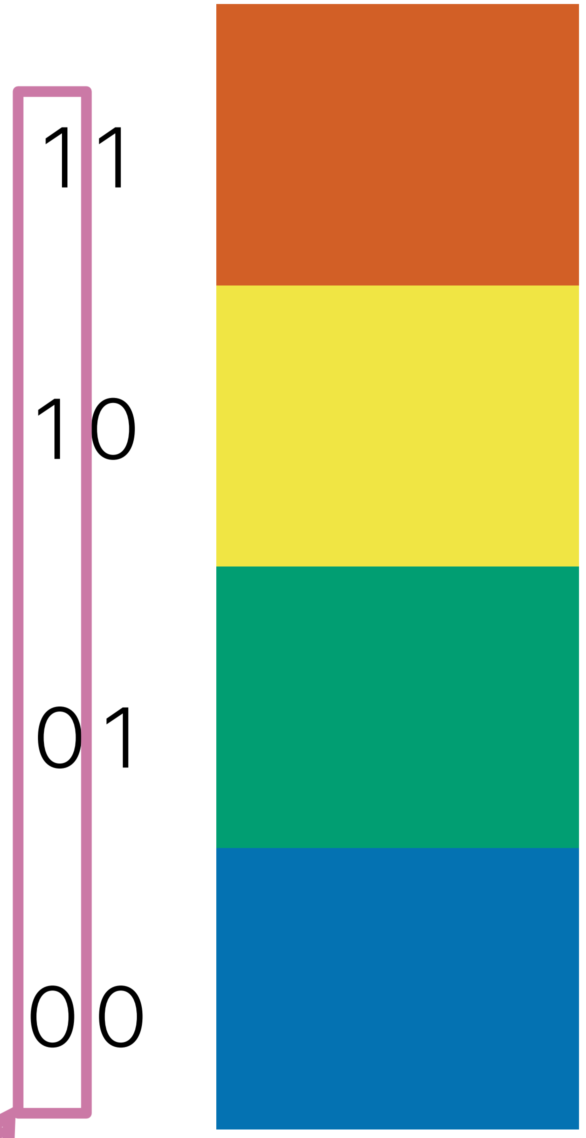
= 01000000 00001001 00011110 10111000 01010001 11101011 10000101 00011111



3 Cycles/Phases to finish programming

Programming in MLC

4 voltage levels,
2-bit



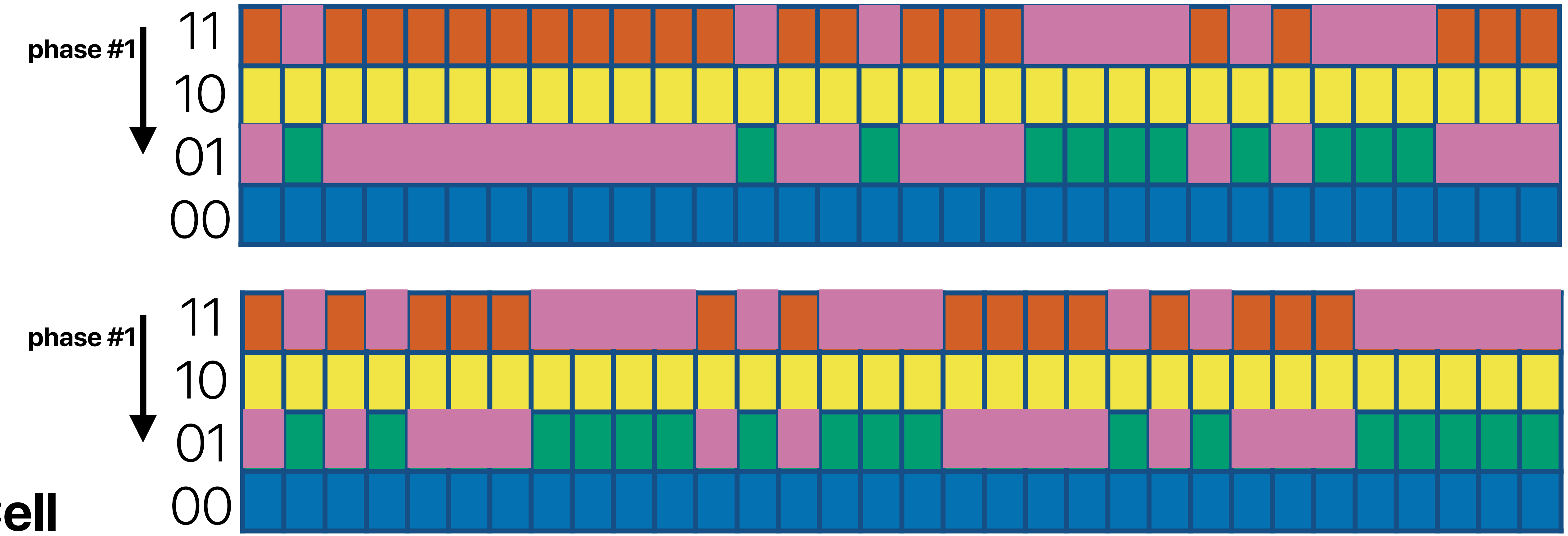
Multi-Level Cell (MLC)

1st page

3.14000000000000012434497875802

= 0x40091EB851EB851F

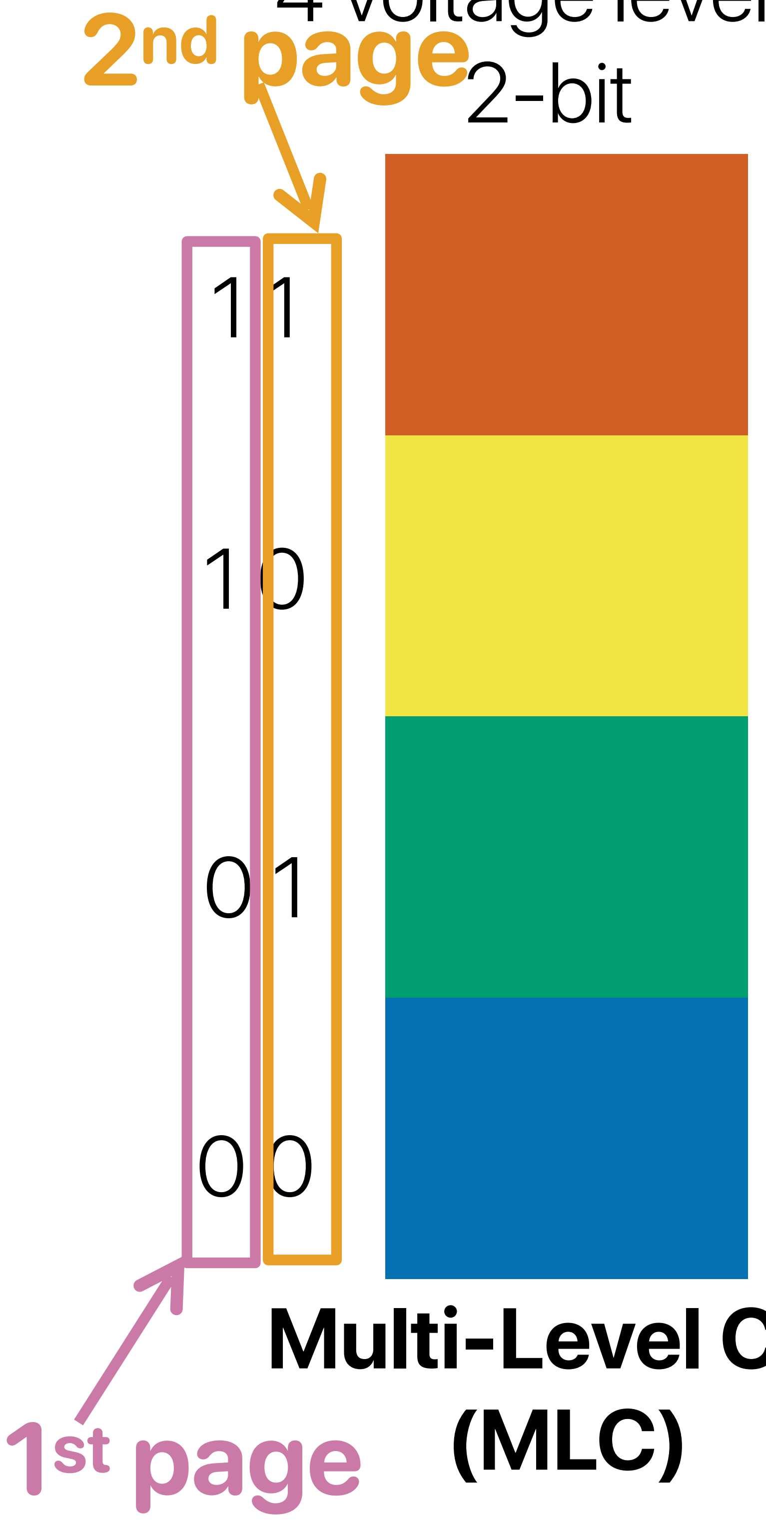
= 01000000 00001001 00011110 10111000 01010001 11101011 10000101 00011111



1 Phase to finish programming the first page!

Programming the 2nd page in MLC

4 voltage levels, 2-bit



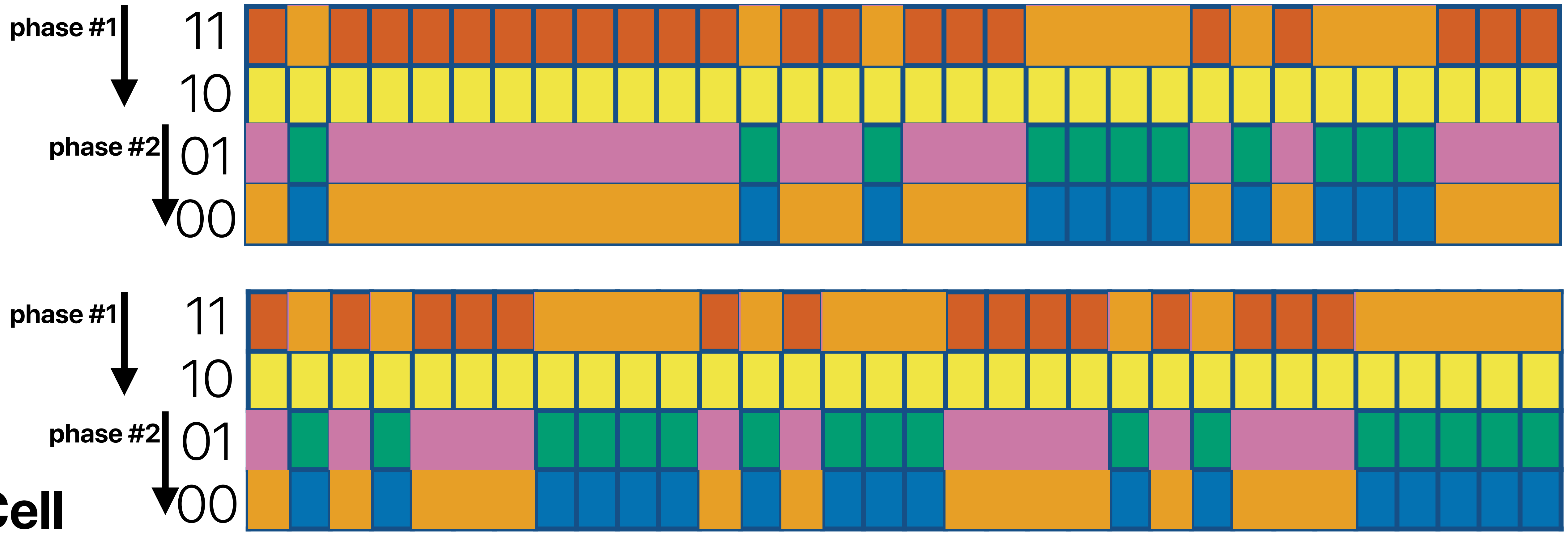
Multi-Level Cell (MLC)

3.14000000000000012434497875802

= 0x40091EB851EB851F

= 01000000 00001001 00011110 10111000 01010001 11101011 10000101 00011111

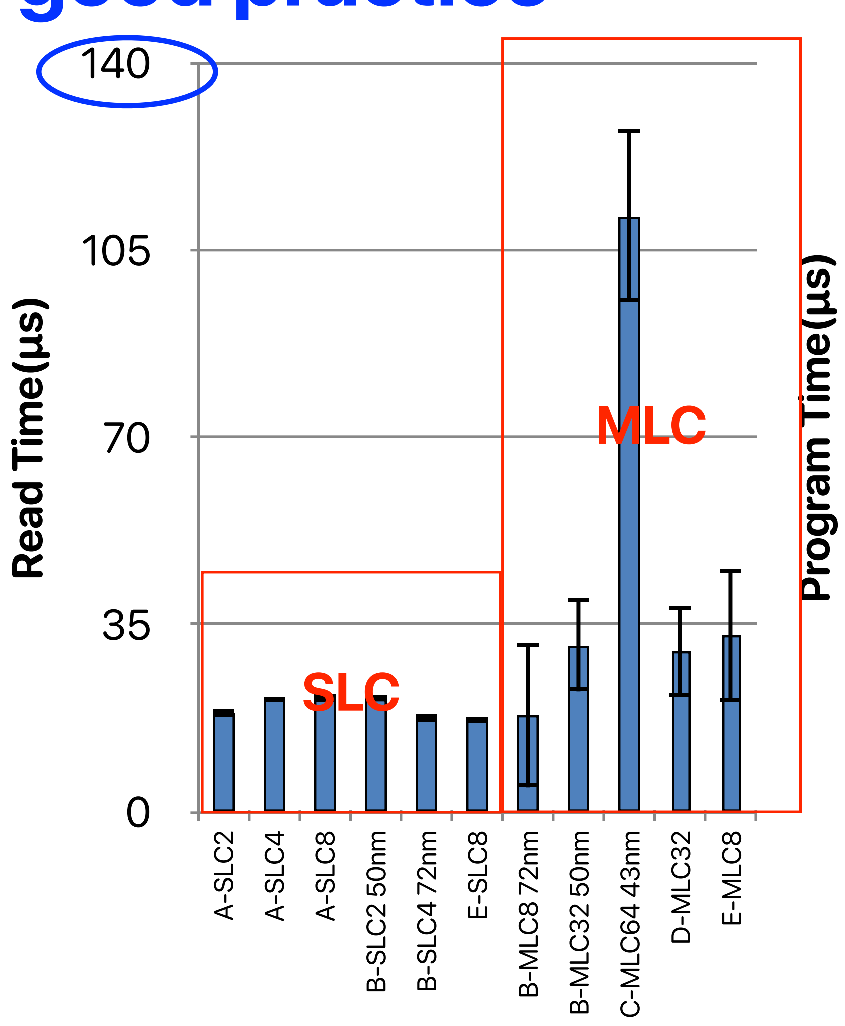
= 01000000 00001001 00011110 10111000 01010001 11101011 10000101 00011111



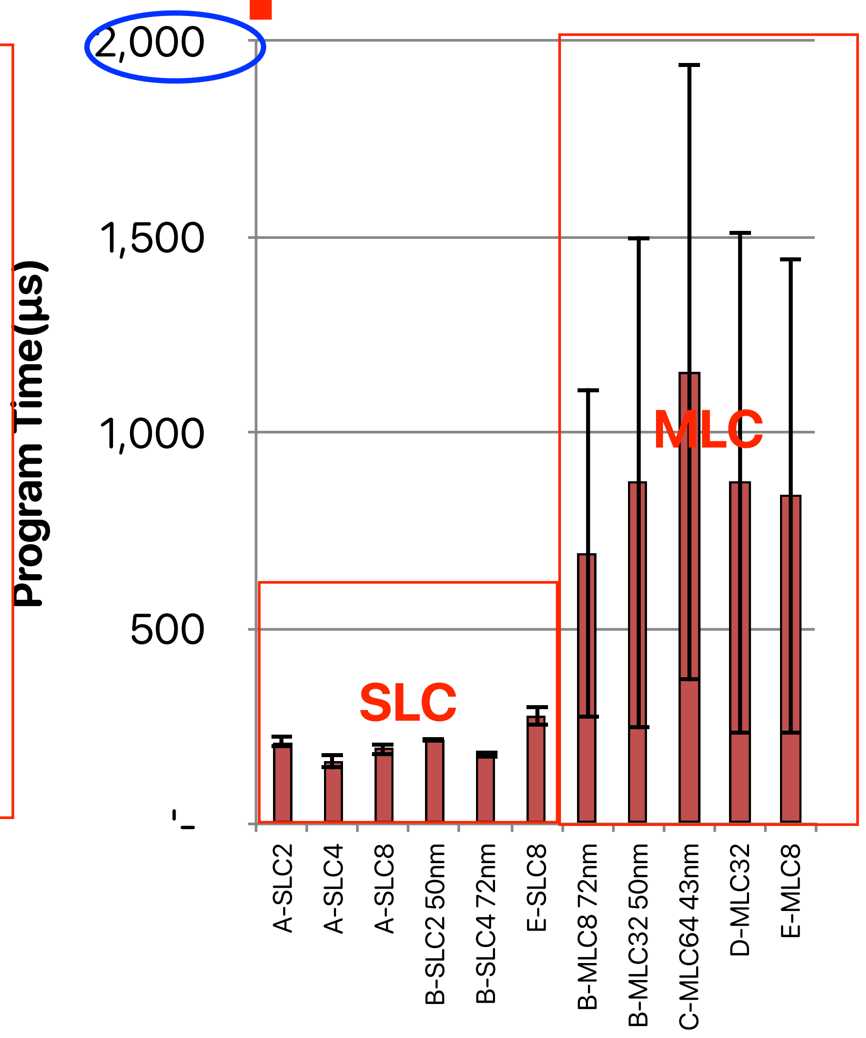
2 Phase to finish programming the second page!

Flash performance

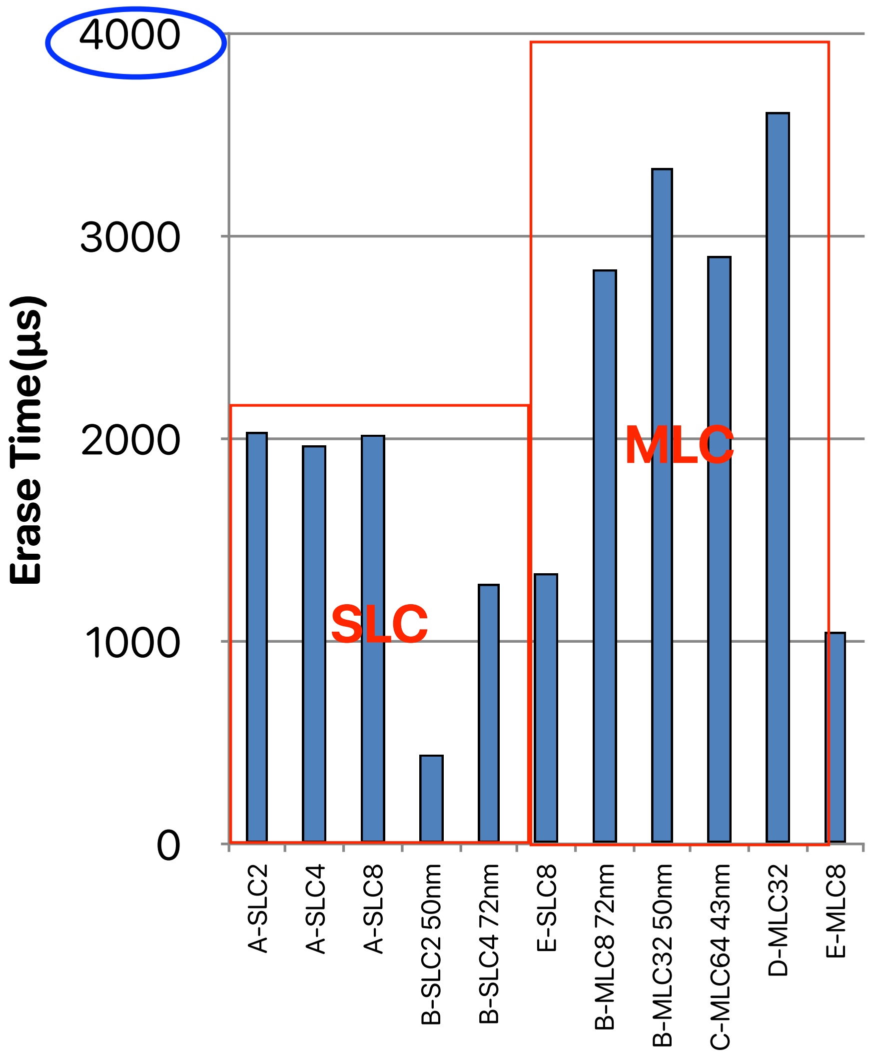
Not a good practice



Reads:
less than 150us



Program/write:
less than 2ms

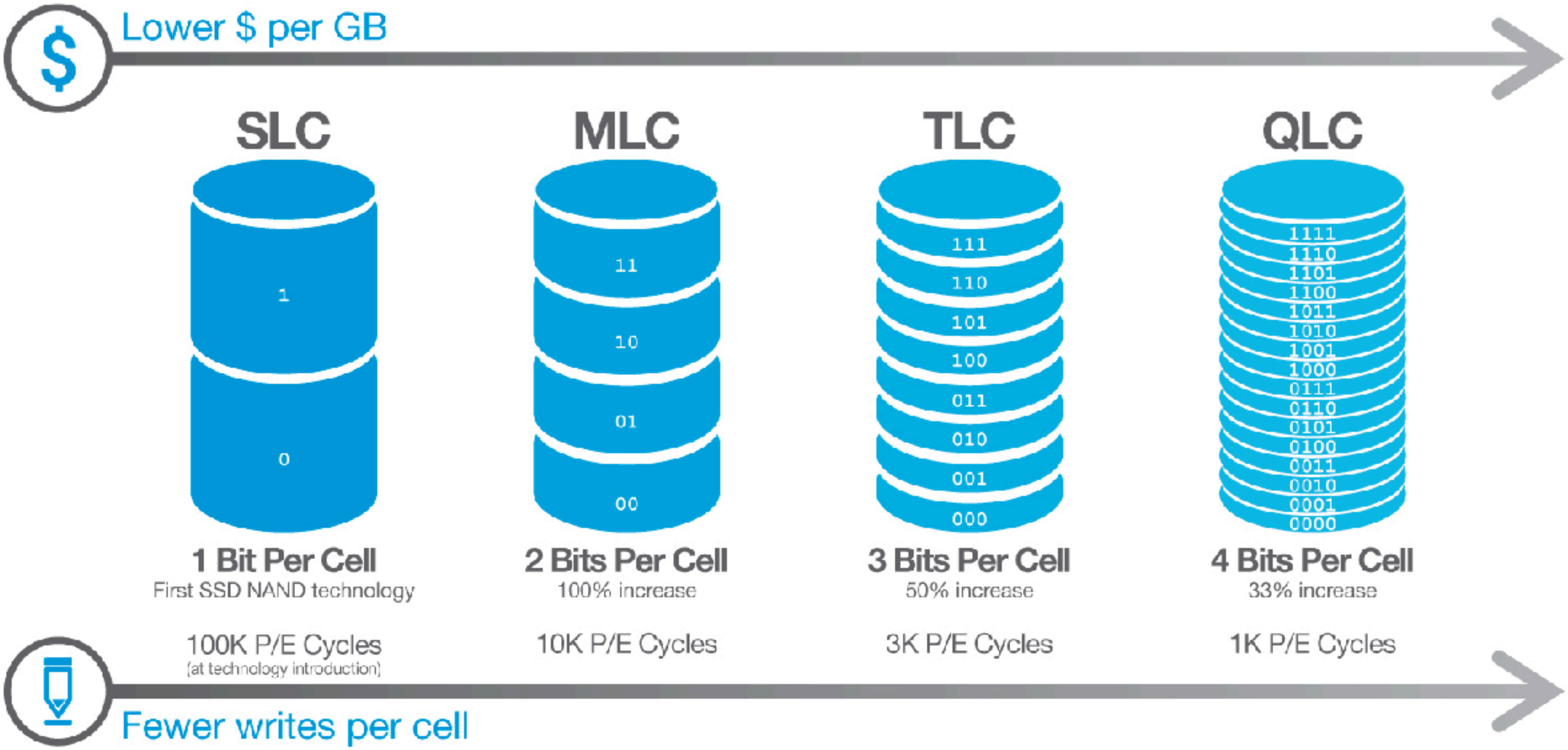


Erase:
less than 3.6ms

Similar relative performance for reads, writes and erases

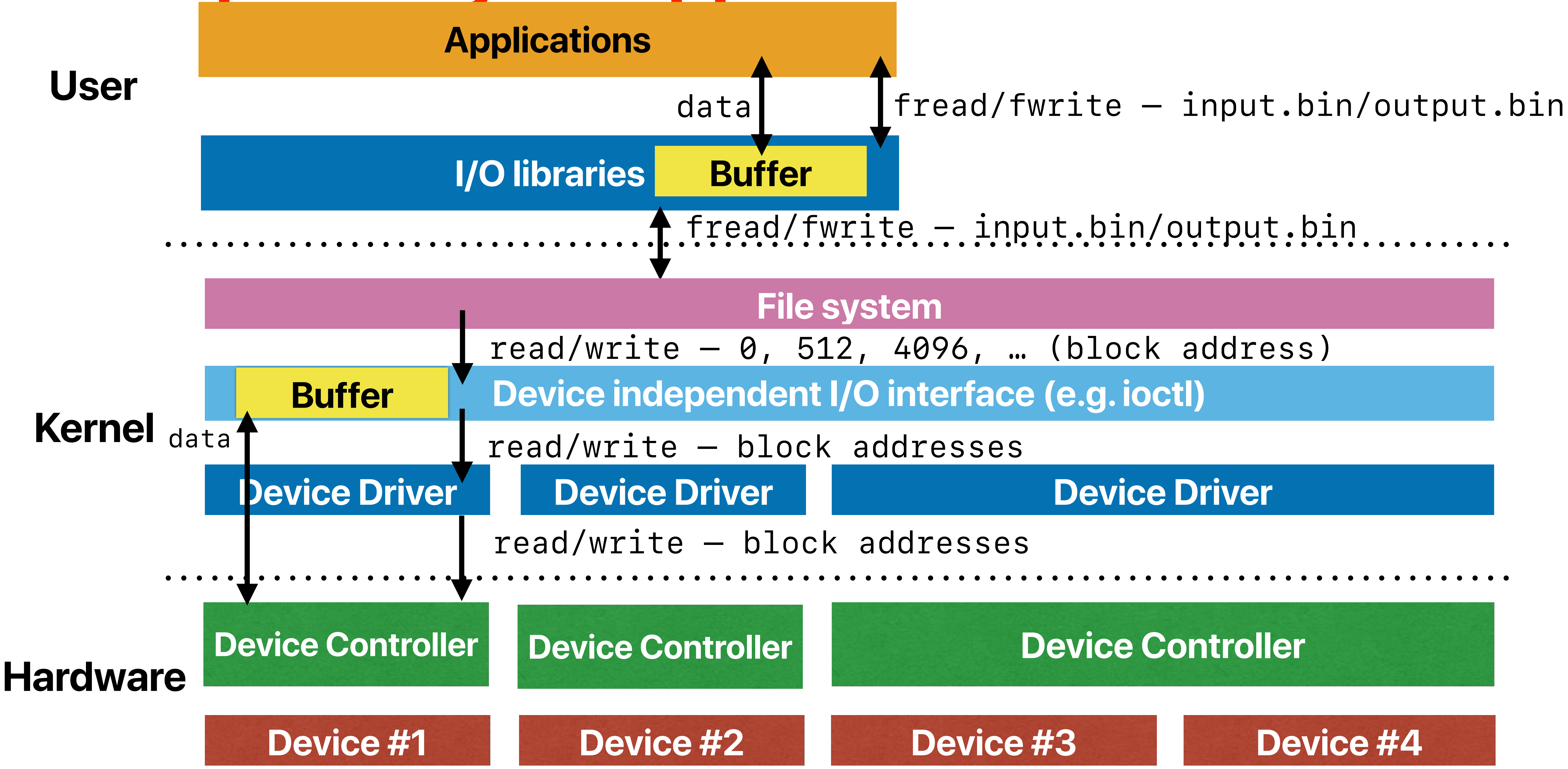
Laura M. Grupp, Adrian M. Caulfield, Joel Coburn, Steven Swanson, Eitan Yaakobi, Paul H. Siegel, and Jack K. Wolf. Characterizing flash memory: anomalies, observations, and applications. In MICRO 2009.

QLC = More Density Per NAND Cell

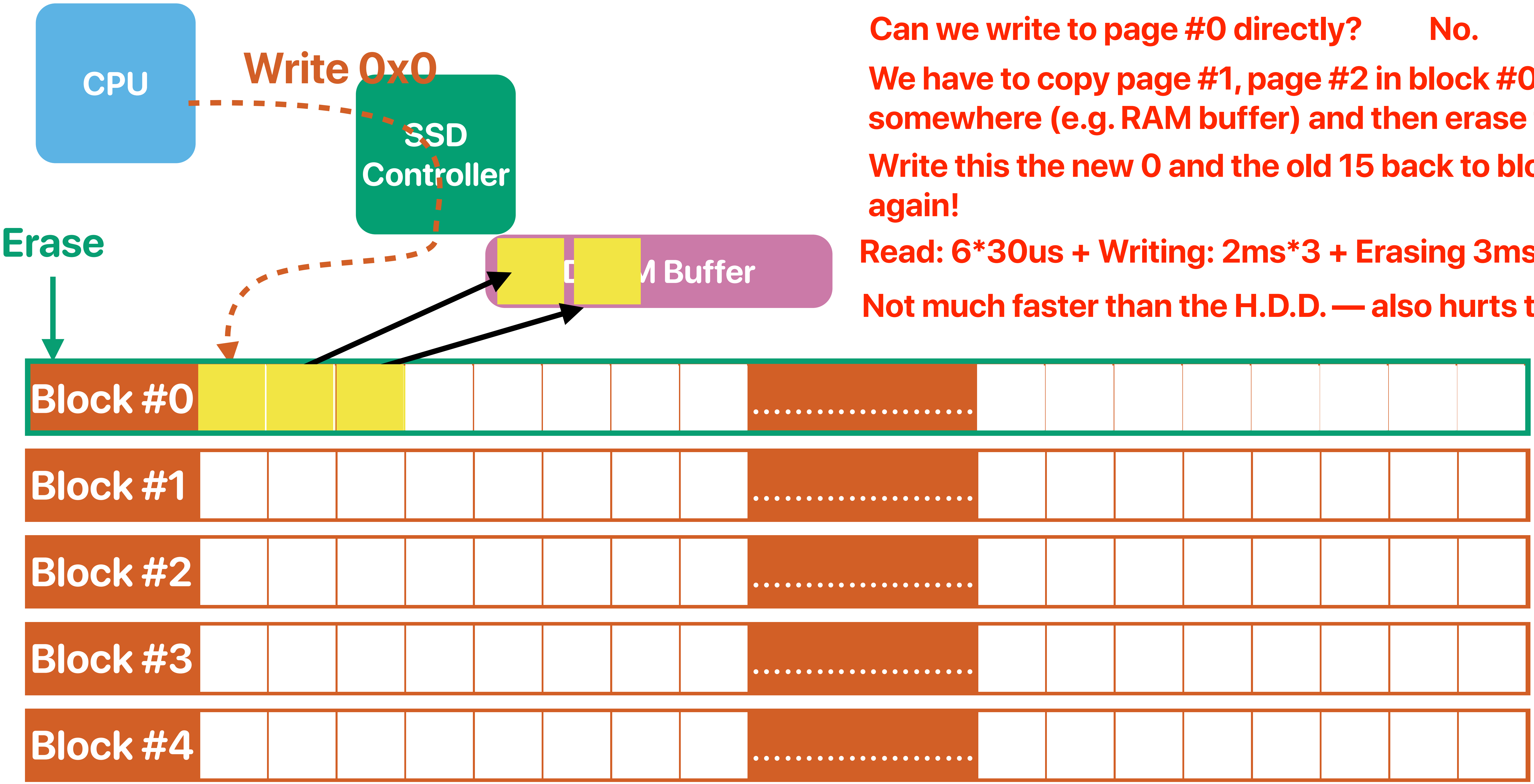


Micron

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

The characteristics of flash memory

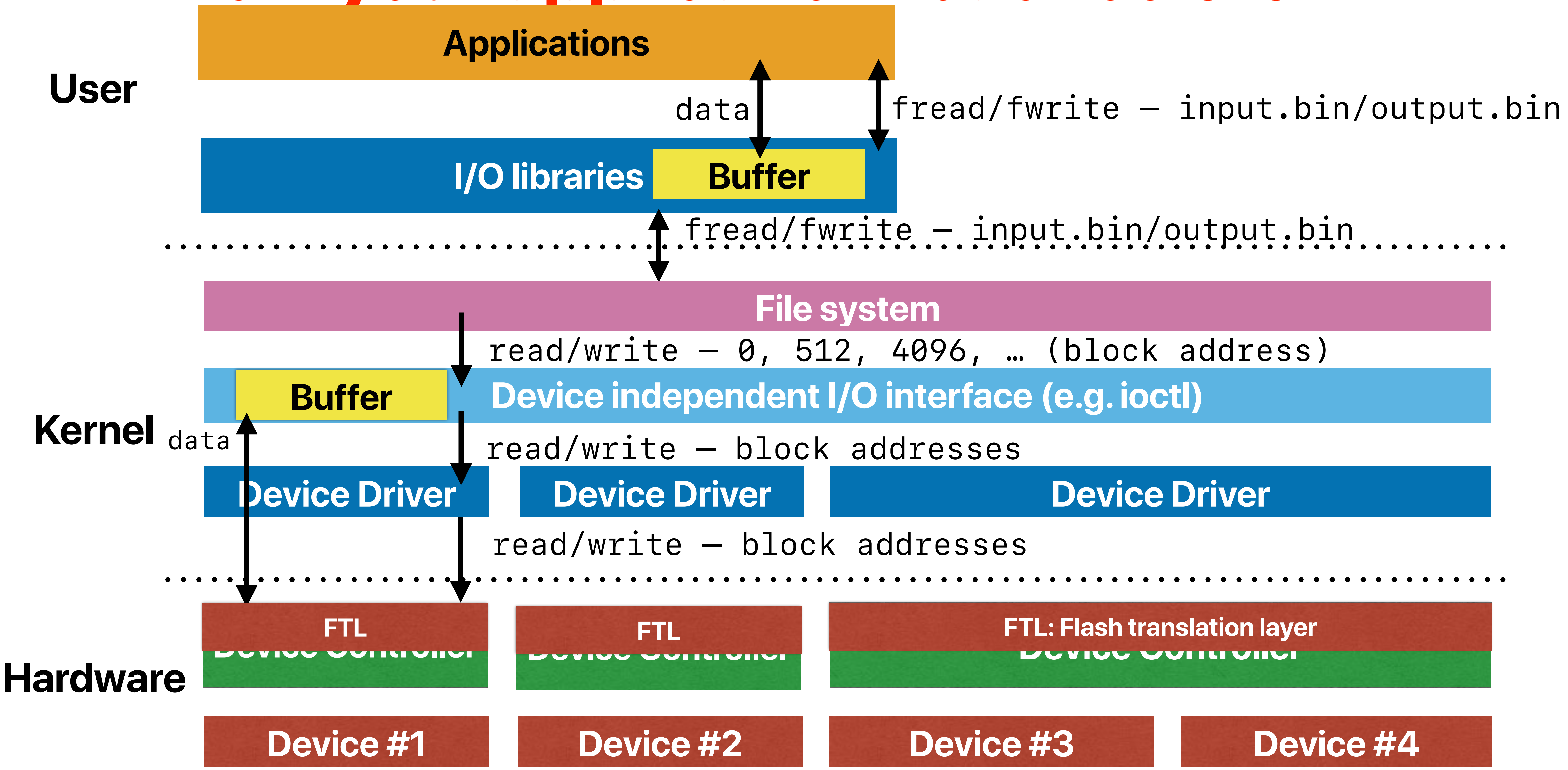
- Regarding the flash memory technology described in eNVy, how many of the following is/are true
  ① The write speed of flash memory can be 100x slower than reading flash
  Writes are slow
  ② The granularities of writing and erasing are different
  You can only program/write in the unit of a page (e.g. 4K), but erases must be perform by blocks (e.g. 128 pages)
  ③ Flash memory cannot be written again without being erased
  In-place update needs to erase the block first
  ④ The flash memory chip has limited number of erase cycles
  You cannot erase too often
  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

Writes are problematic in flash

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

–David Wheeler

How your application reaches S.S.D.



https://www.pollev.com/hungweitseng close in 1:00

How should we deal with writes?

- How many of following optimizations would help improve the write performance of flash SSDs?
  1. Write asynchronously
  2. Out-of-place update
  3. Preallocate locations for writing data
  4. Aggregate writes to the same bank/chip

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

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

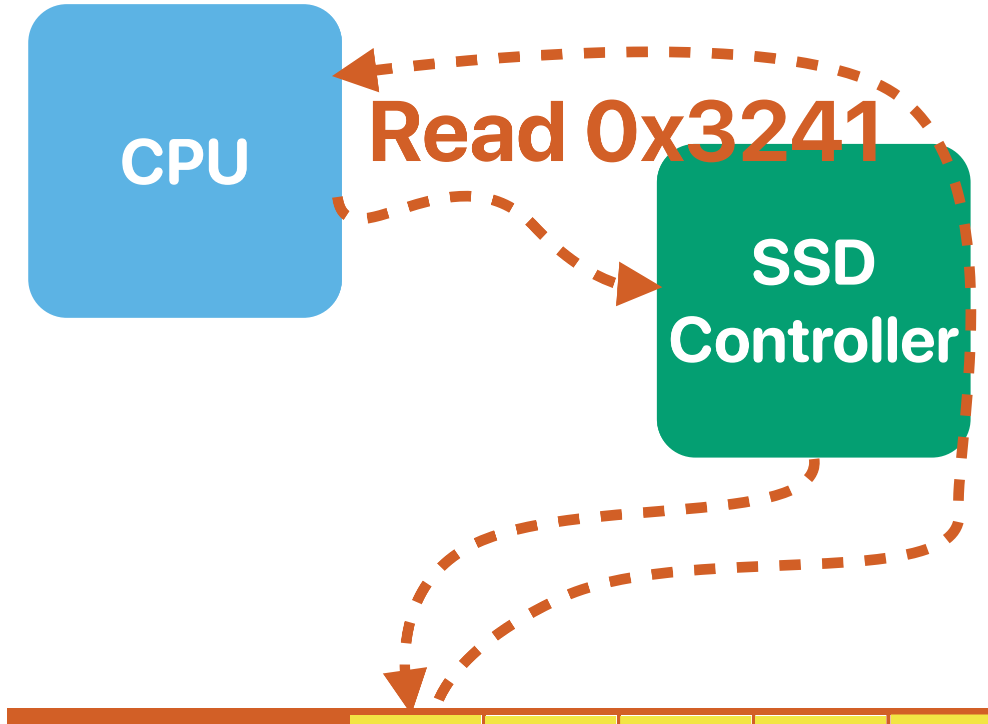
Sounds familiar …

Log-structured file system!

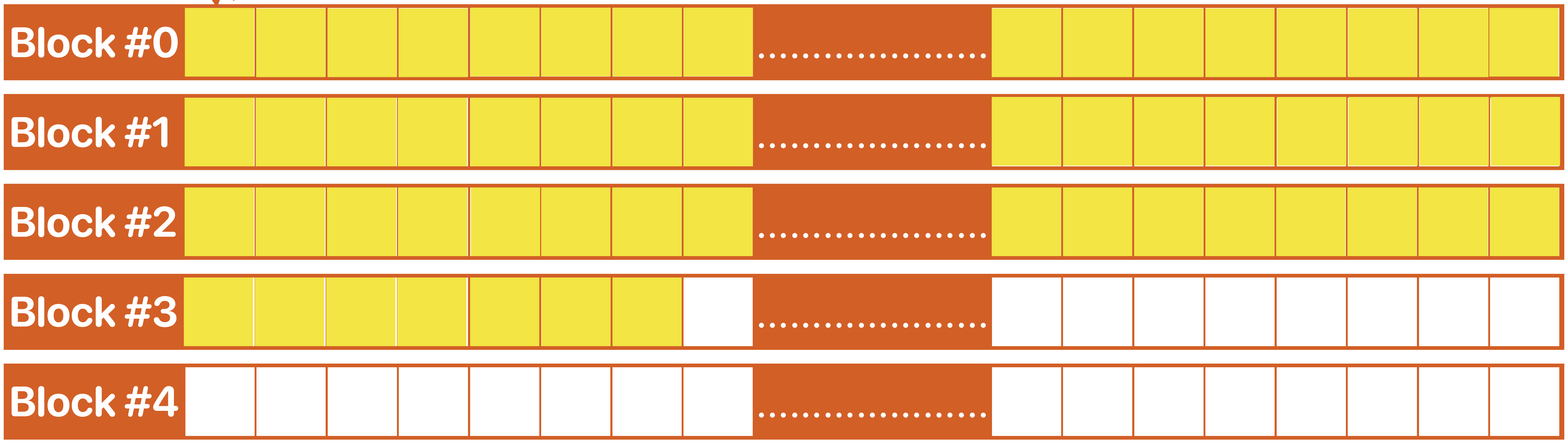
Flash Translation Layer (FTL)

- 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

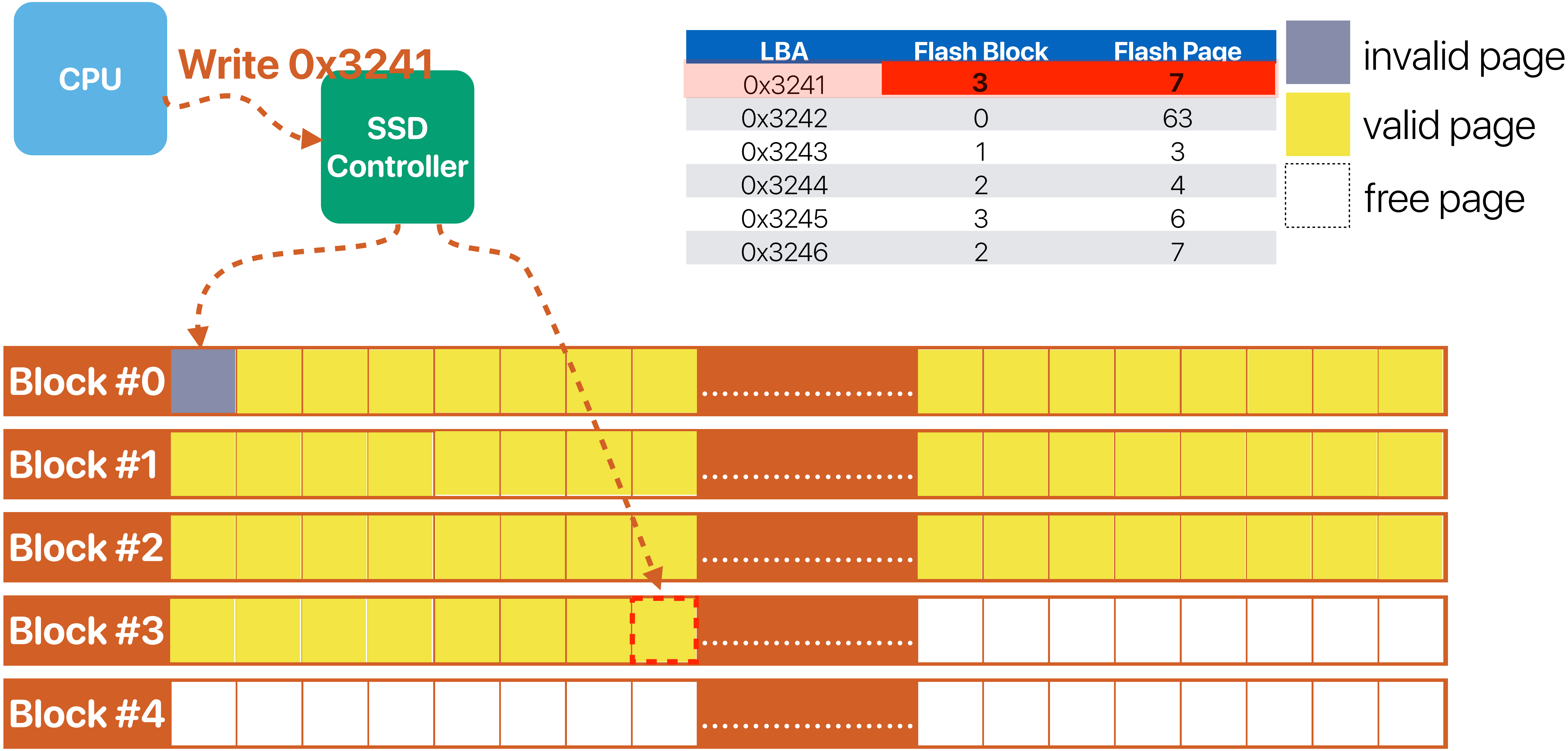
What happens on a read with FTL



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



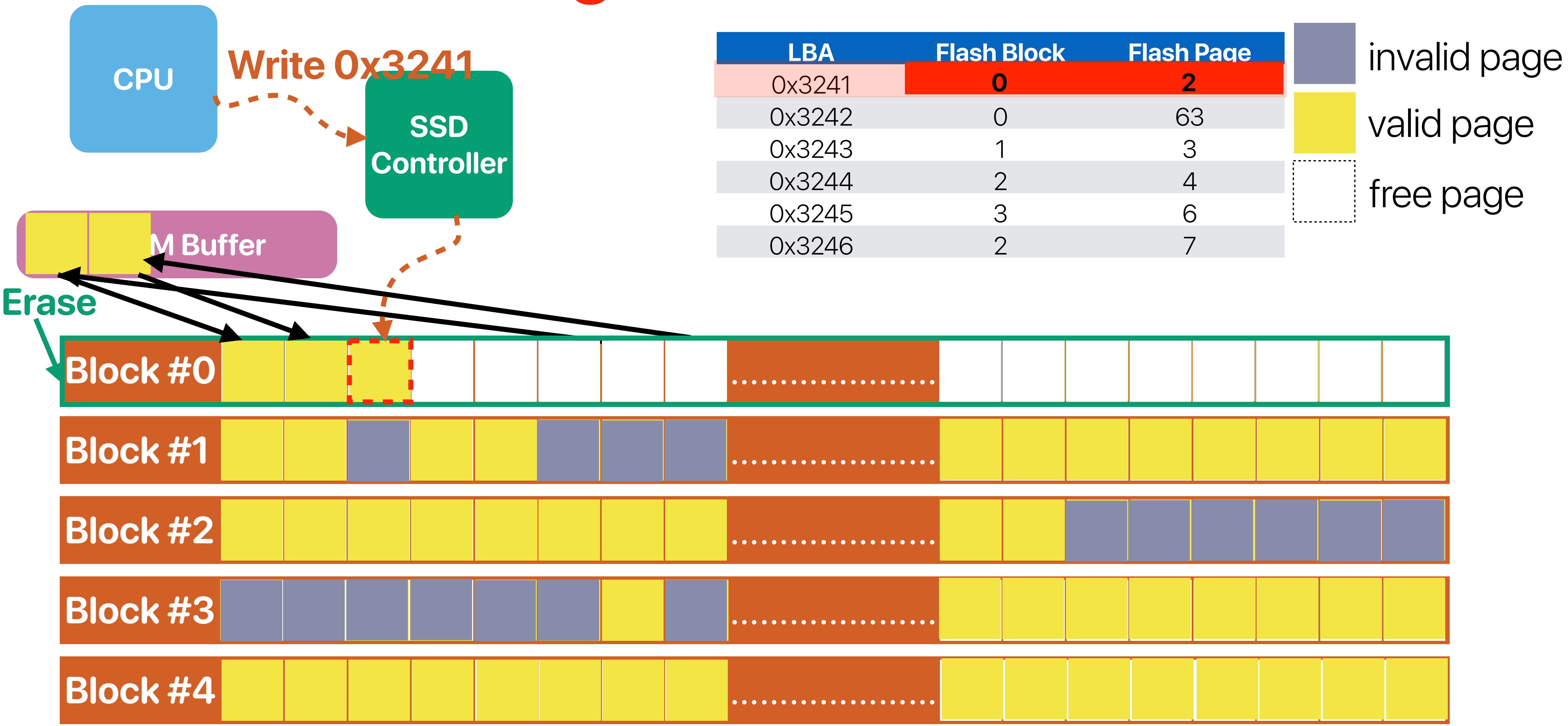
What happens on a write with FTL



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

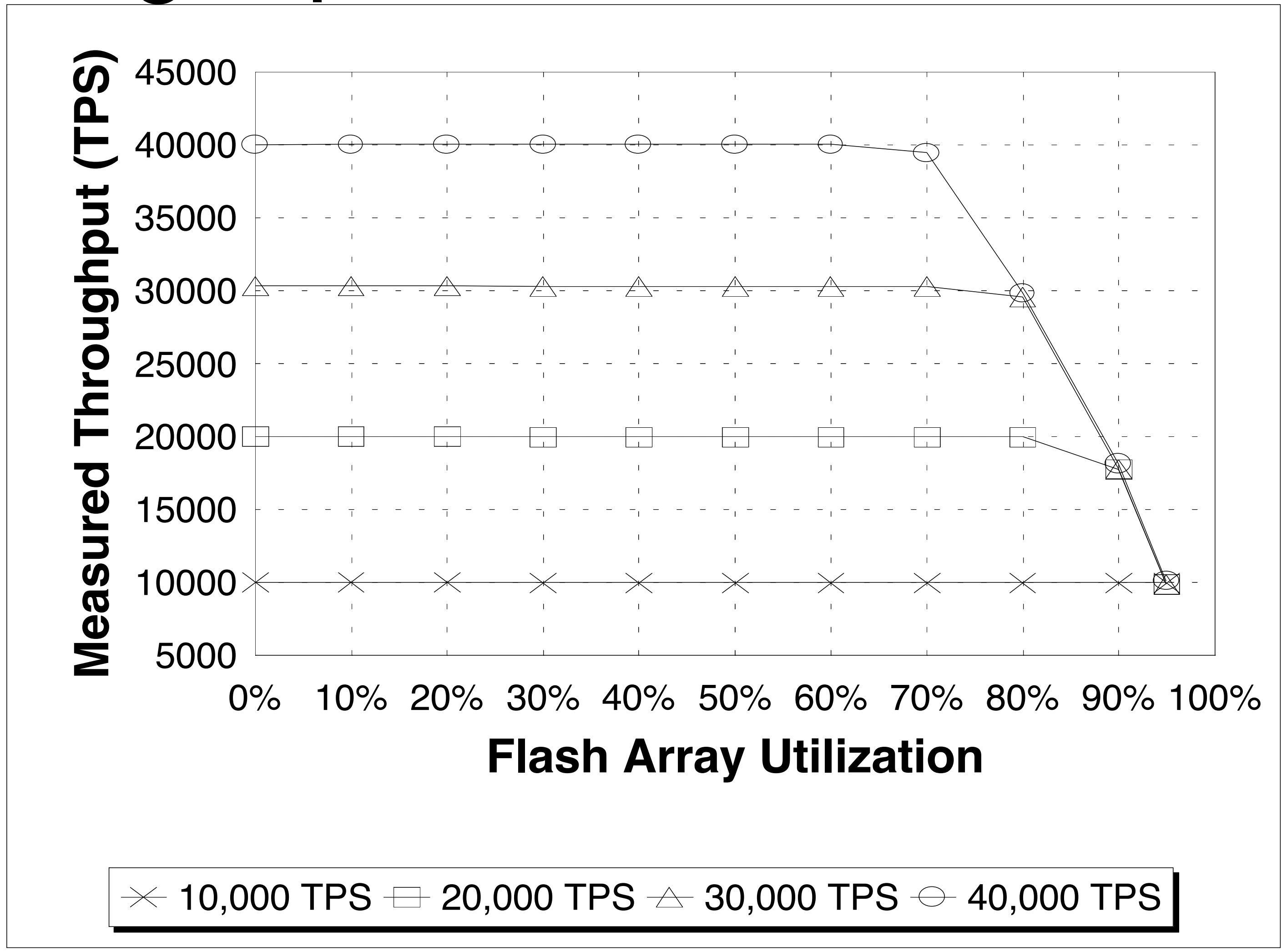
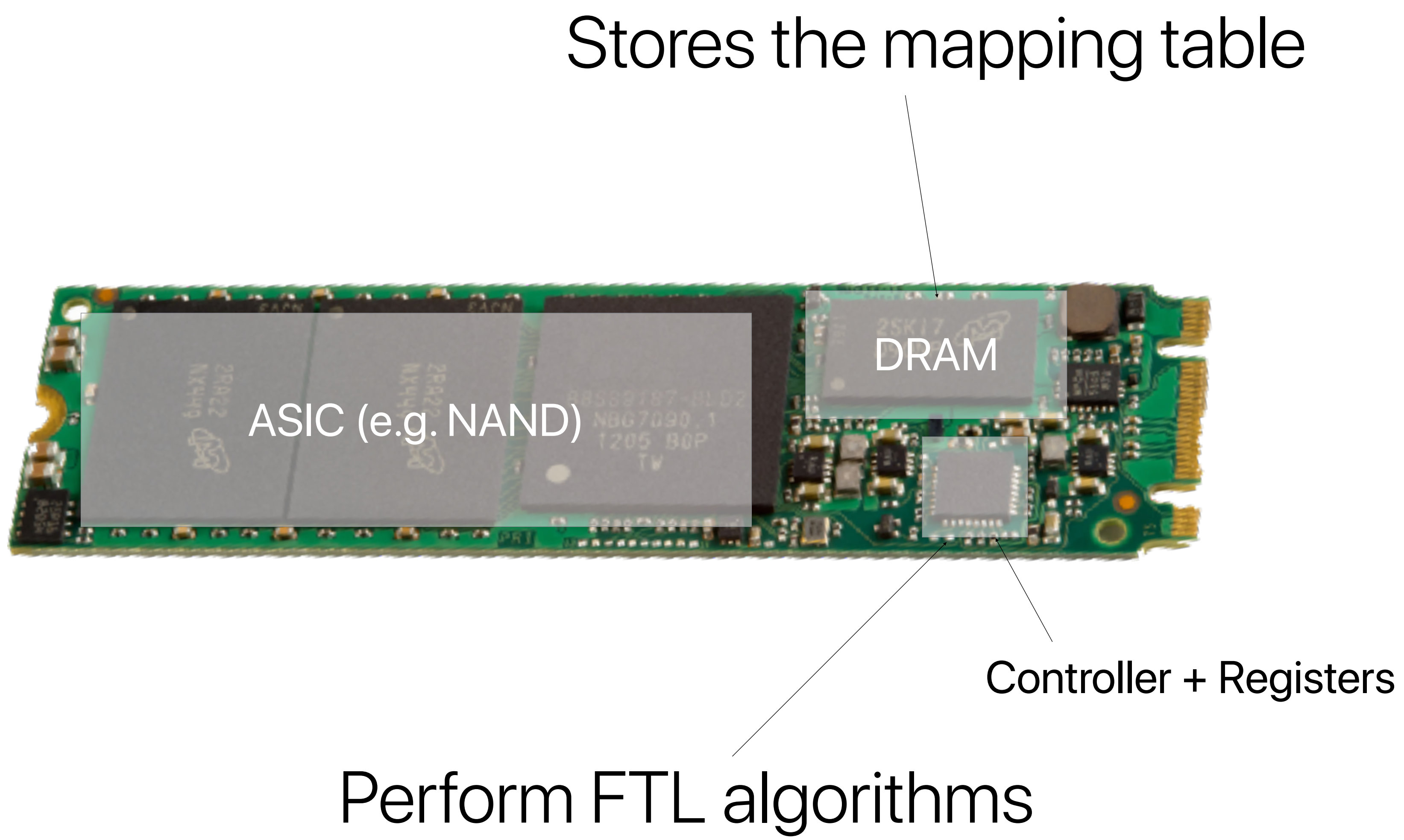


Figure 14: Throughput for Various Levels of Utilization

The impact of eNVy

- Your SSD structured exactly like this!

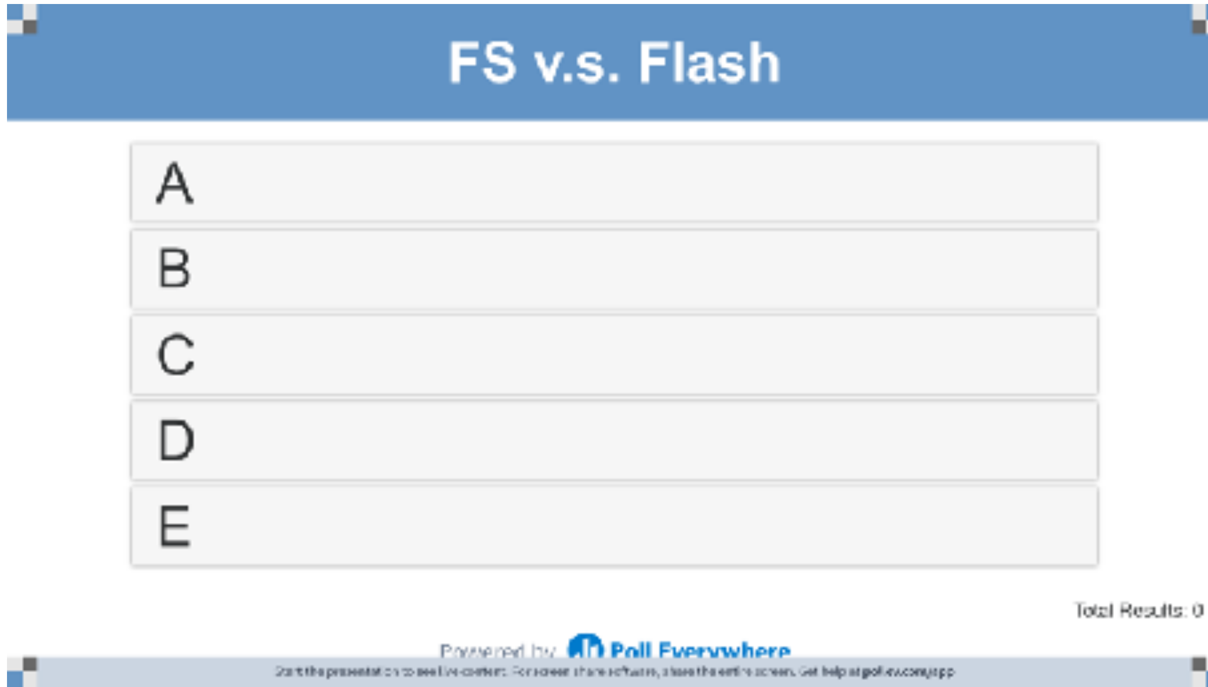


https://www.pollev.com/hungweitseng close in 1:00

File system features revisited

- How many of the following file system optimizations that we learned so far would still help improve performance if the underlying device becomes an SSD?
  1. Cylinder group
  2. Larger block size
  3. Fragments
  4. Logs

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



File system features revisited

- How many of the following file system optimizations that we learned so far would still help improve performance if the underlying device becomes an SSD?

no cylinder structure on flash. You probably want random accesses to exploit parallelism

① Cylinder group

② Larger block size maybe ... as long as the block size is larger than the page size

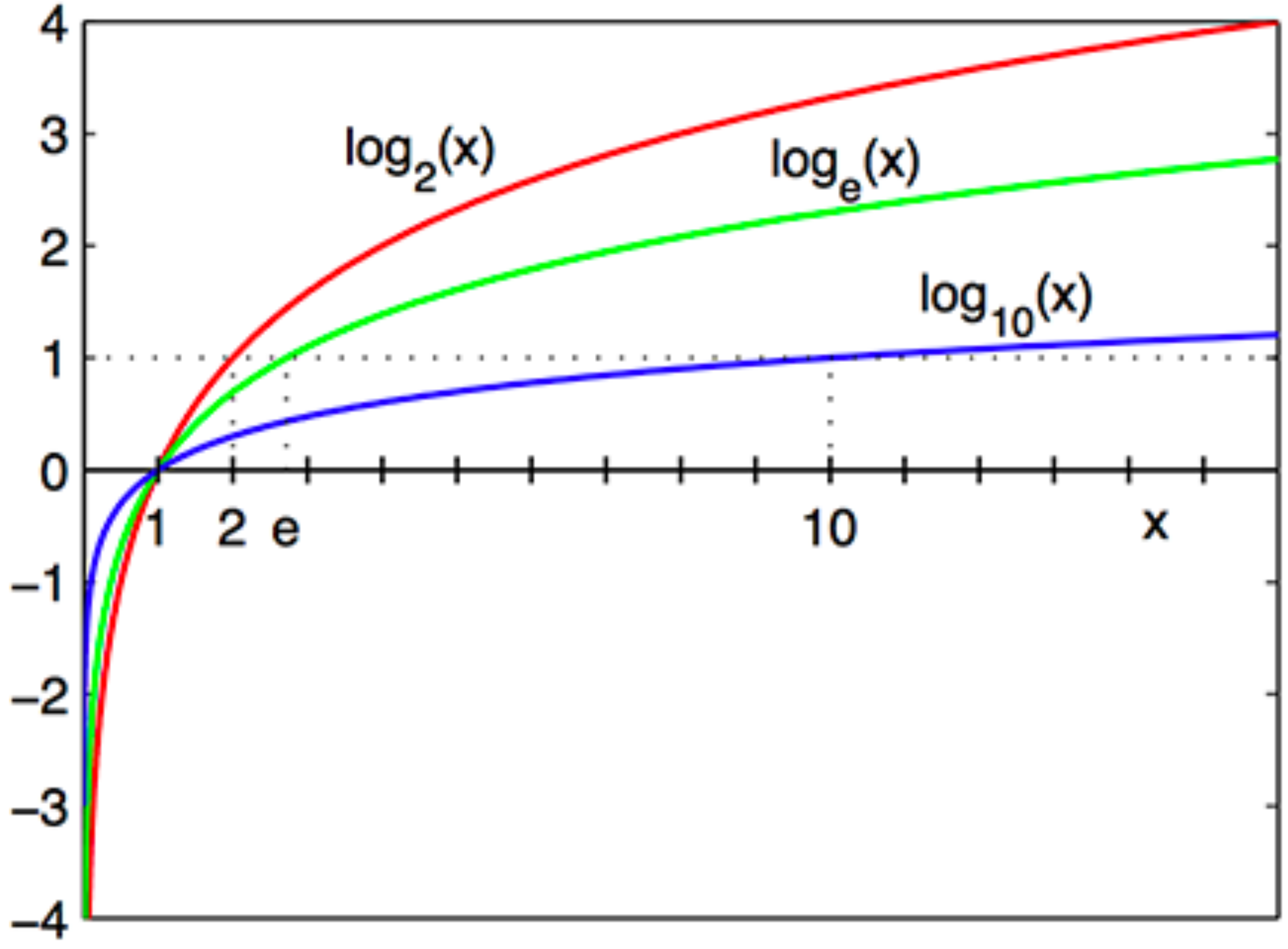
③ Fragments remember: flash can only write units of pages

④ Logs Let's discuss this with the next paper!

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

Don’t stack your log on my log

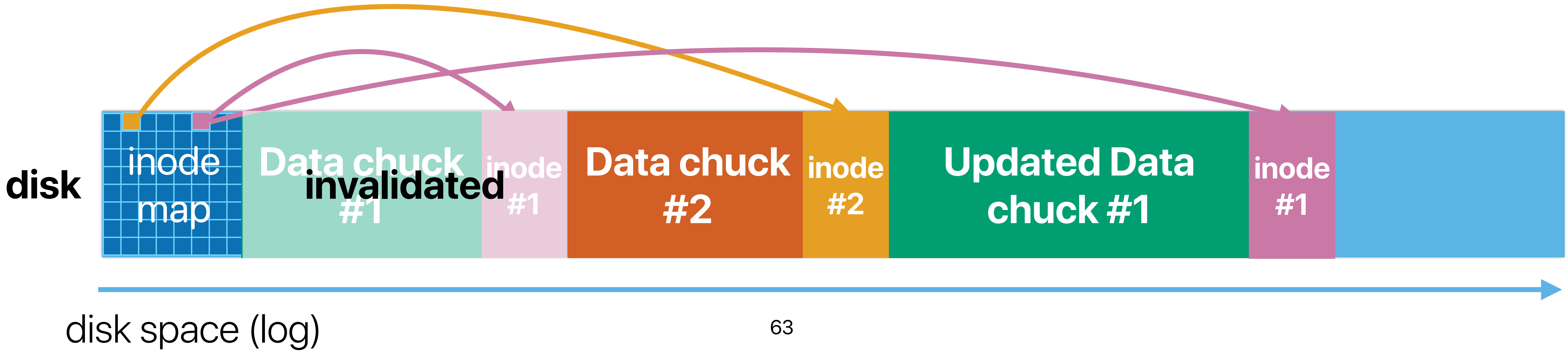
Jingpei Yang, Ned Plasson, Greg Gillis, Nisha Talagala, and Swaminathan Sundararaman
SanDisk Corporation



log$_2$(x)
log$_e$(x)
log$_{10}$(x)
4
3
2
1
0
−1
−2
−3
−4
1
2
e
10
x

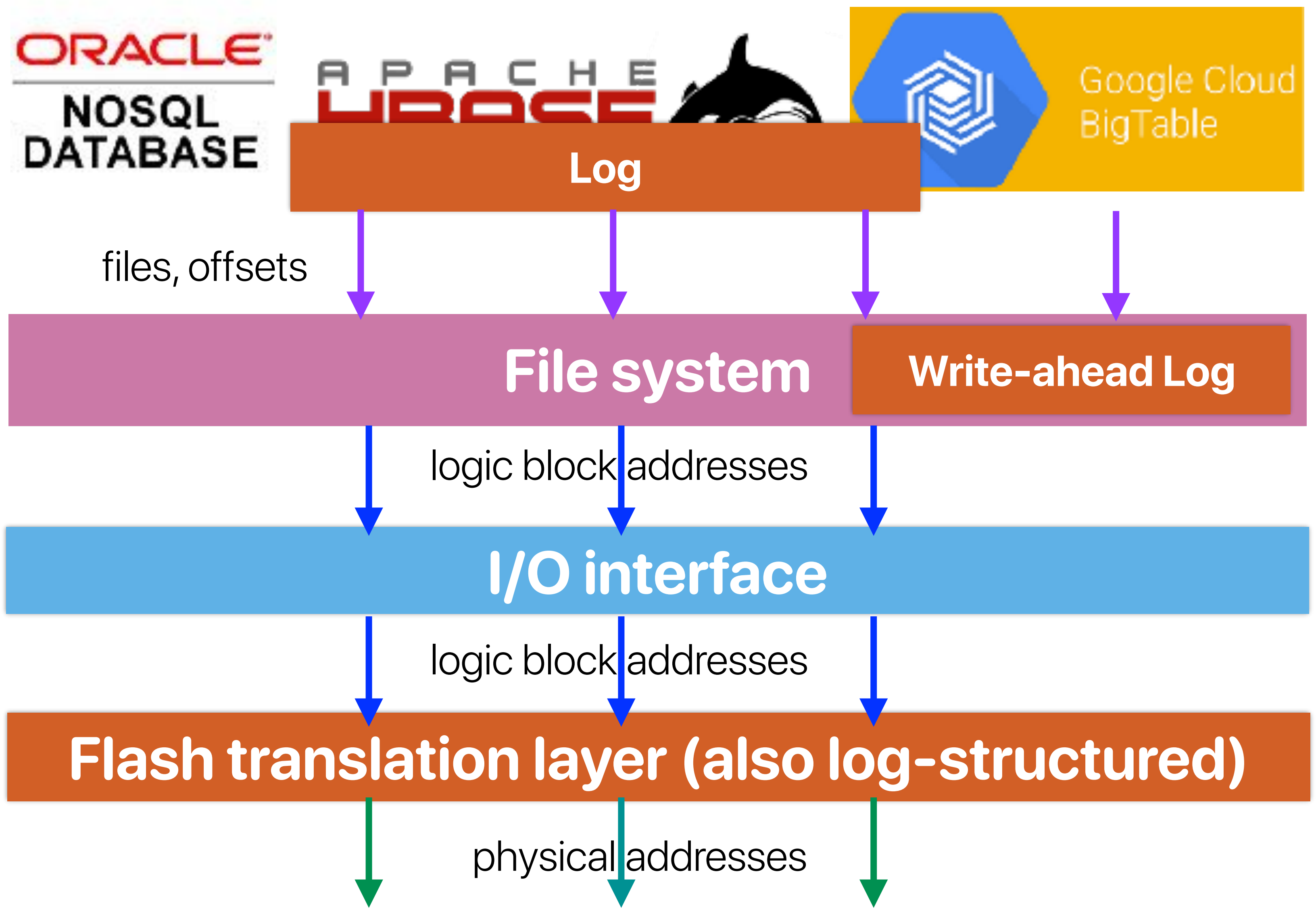
Log

- An append only data structure that records all changes
- Advantages of logs
  - Better performance — always sequential accesses
  - Faster writes — you just need to append without sanitize existing data first
  - Ease of recovery — you can find previous changes within the log
- Disadvantage of logs — you will need to explicit perform garbage collections to reclaim spaces

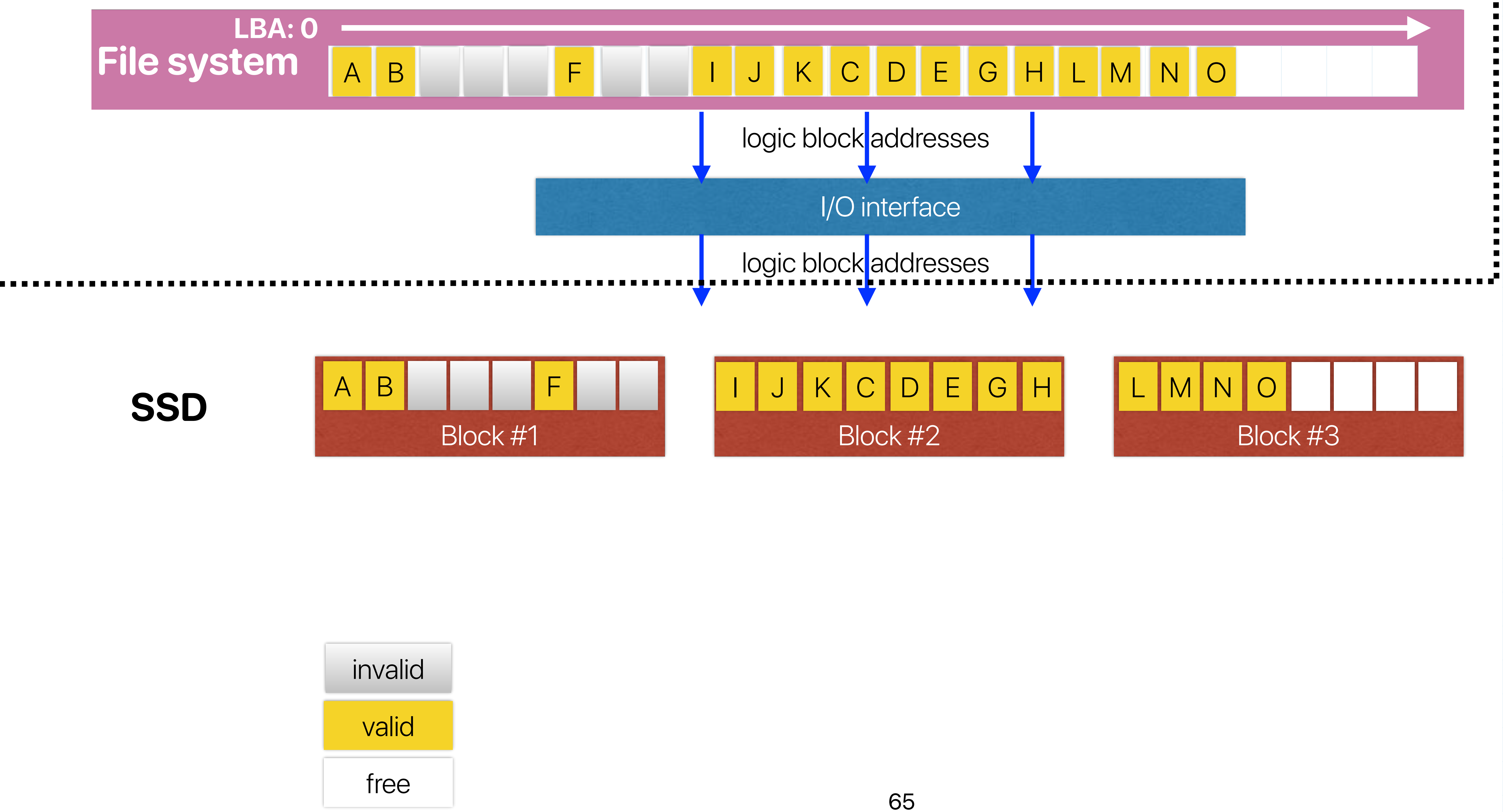


Why should we care about this paper?

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



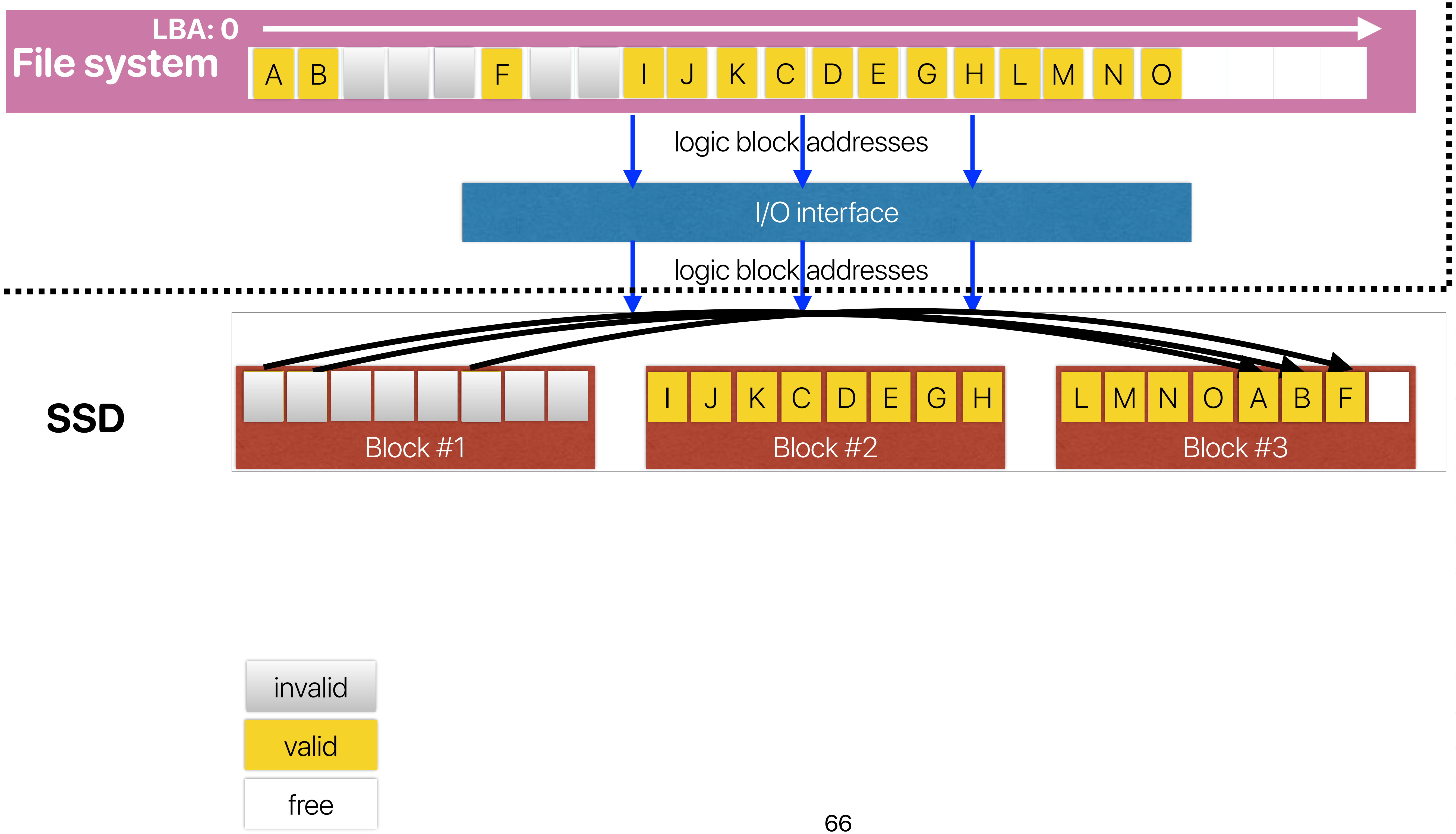
For example, garbage collection



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	-	-

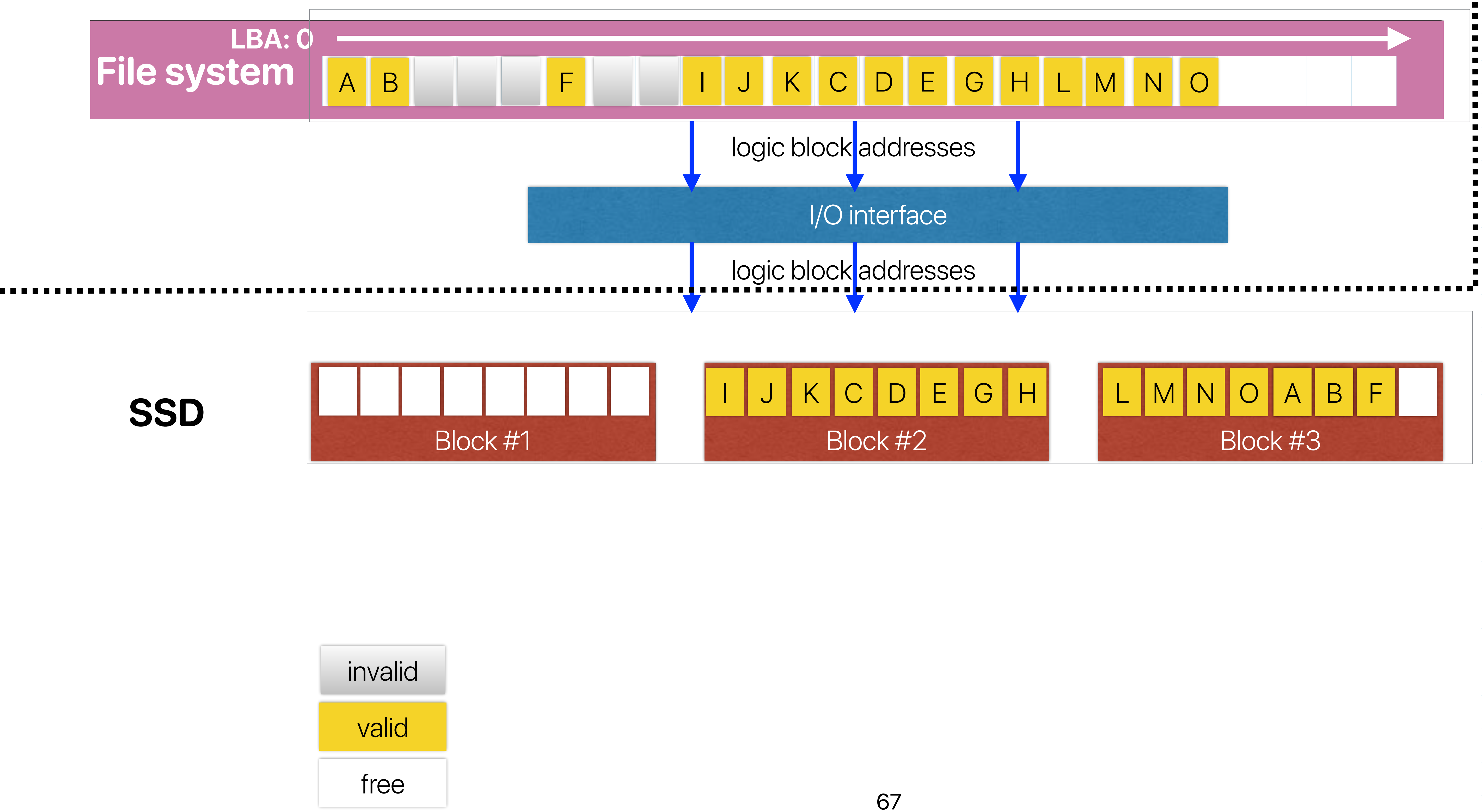
Now, SSD wants to reclaim a block



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	-	-

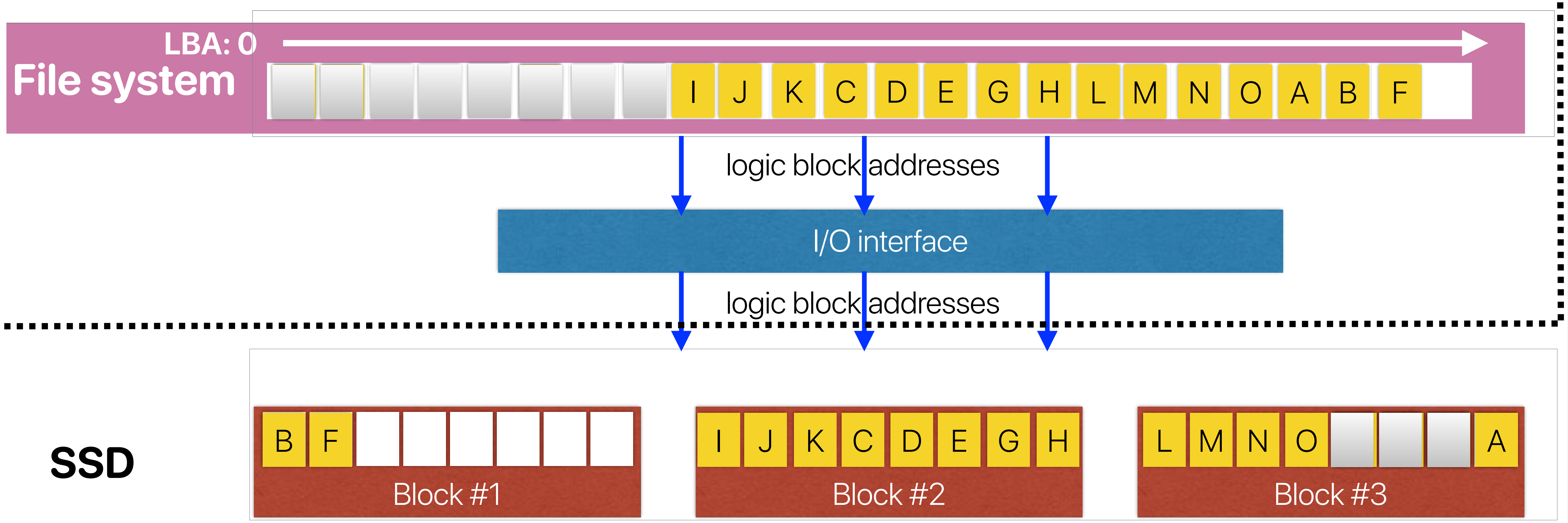
Garbage collection on the SSD done!



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 system features revisited

- How many of the following file system optimizations that we learned so far would still help improve performance if the underlying device becomes an SSD?

    no cylinder structure on flash. You probably want random accesses to exploit parallelism

    ① Cylinder group

    ② Larger block size maybe ... as long as the block size is larger than the page size

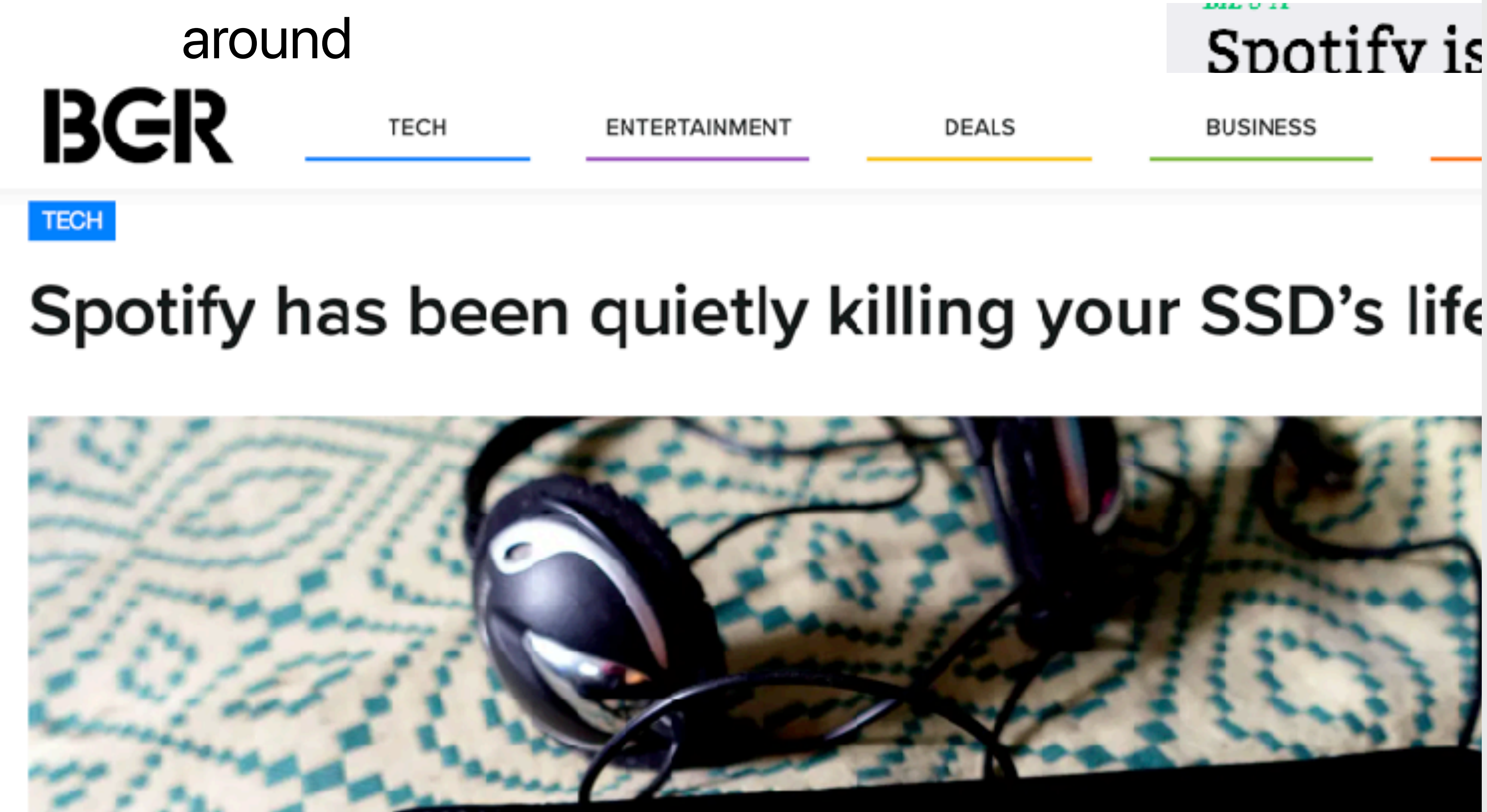
    ③ Fragments remember: flash can only write units of pages, it cannot be programmed for any smaller granularities

    ④ Logs What do you think?

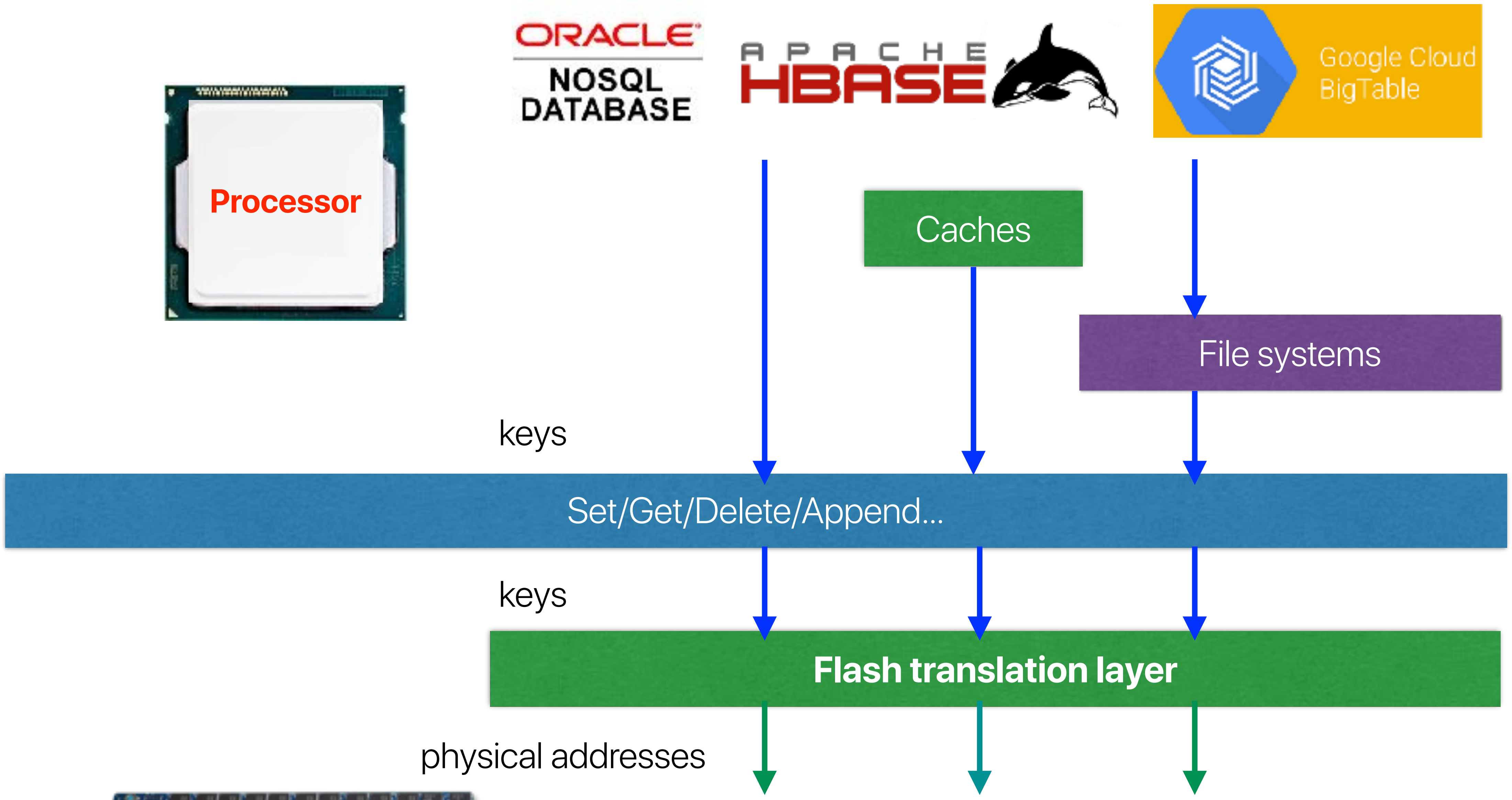
    A. 0
    B. 1
    C. 2
    D. 3
    E. 4

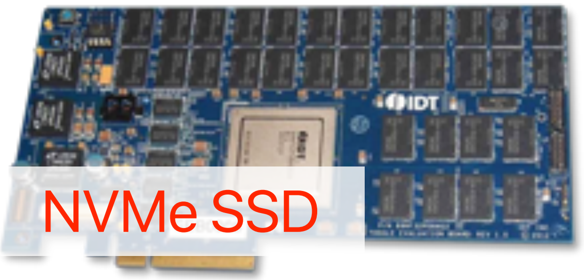
File systems for flash-based SSDs

- 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.**

Announcement

- Reading quizzes due next Tuesday
- Office hour
  - MTu 11a-12p, W 2p-3p & F 11a-12p
  - Use the office hour Zoom link, not the lecture one
- Project
  - Due 3/3
  - No late submission is allowed — to make time for grading and potential of regrading

Computer
Science &
Engineering

202

つづく