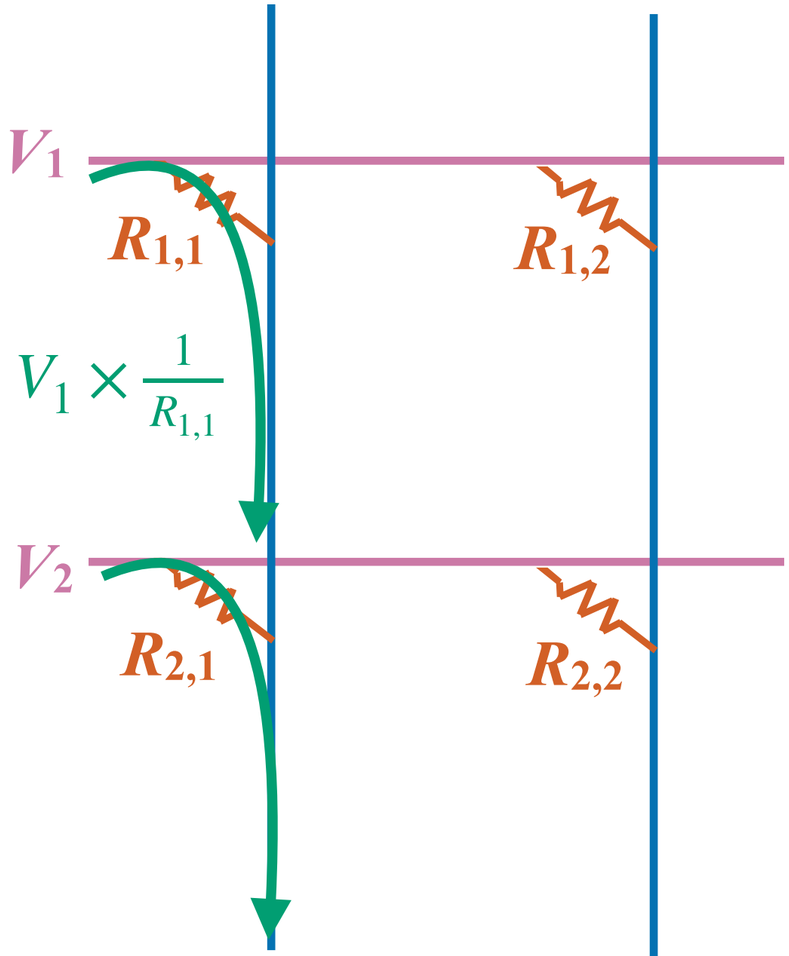
System Peripherals

Hung-Wei Tseng

ReRAM



$$I_{1,1} = V_1 \times \frac{1}{R_{1,1}}$$

$$I_1 = V_1 \times \frac{1}{R_{1,1}} + V_2 \times \frac{1}{R_{2,1}} \qquad I_2 = V_1 \times \frac{1}{R_{2,1}} + V_2 \times \frac{1}{R_{2,2}}$$

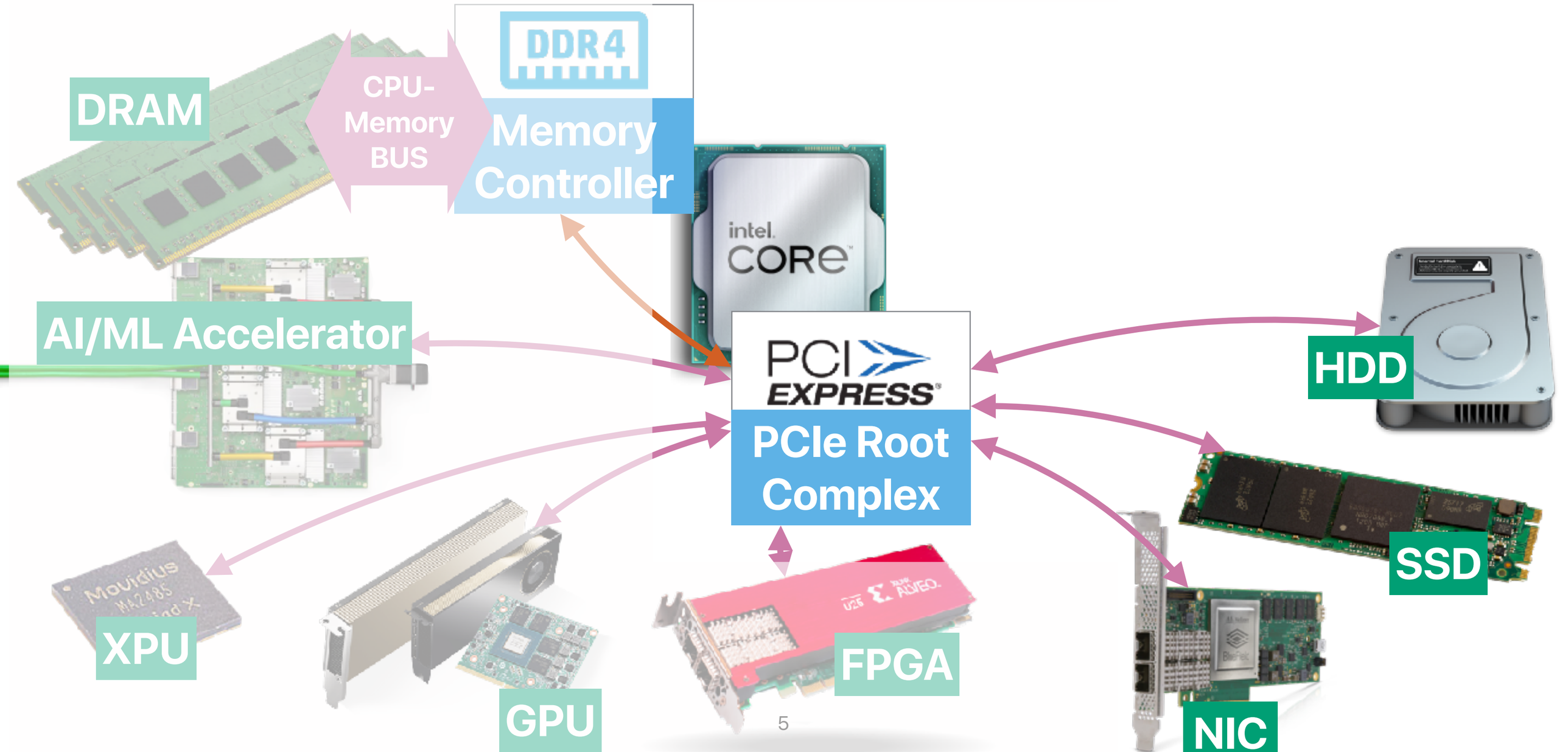
$$\begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \end{bmatrix} \times \begin{bmatrix} R_{1,1}^{-1} & R_{1,2}^{-1} & R_{1,3}^{-1} & R_{1,4}^{-1} \\ R_{2,1}^{-1} & R_{2,2}^{-1} & R_{2,3}^{-1} & R_{2,4}^{-1} \\ R_{3,1}^{-1} & R_{3,2}^{-1} & R_{3,3}^{-1} & R_{3,4}^{-1} \\ R_{4,1}^{-1} & R_{4,2}^{-1} & R_{4,3}^{-1} & R_{4,4}^{-1} \end{bmatrix}$$

3:00

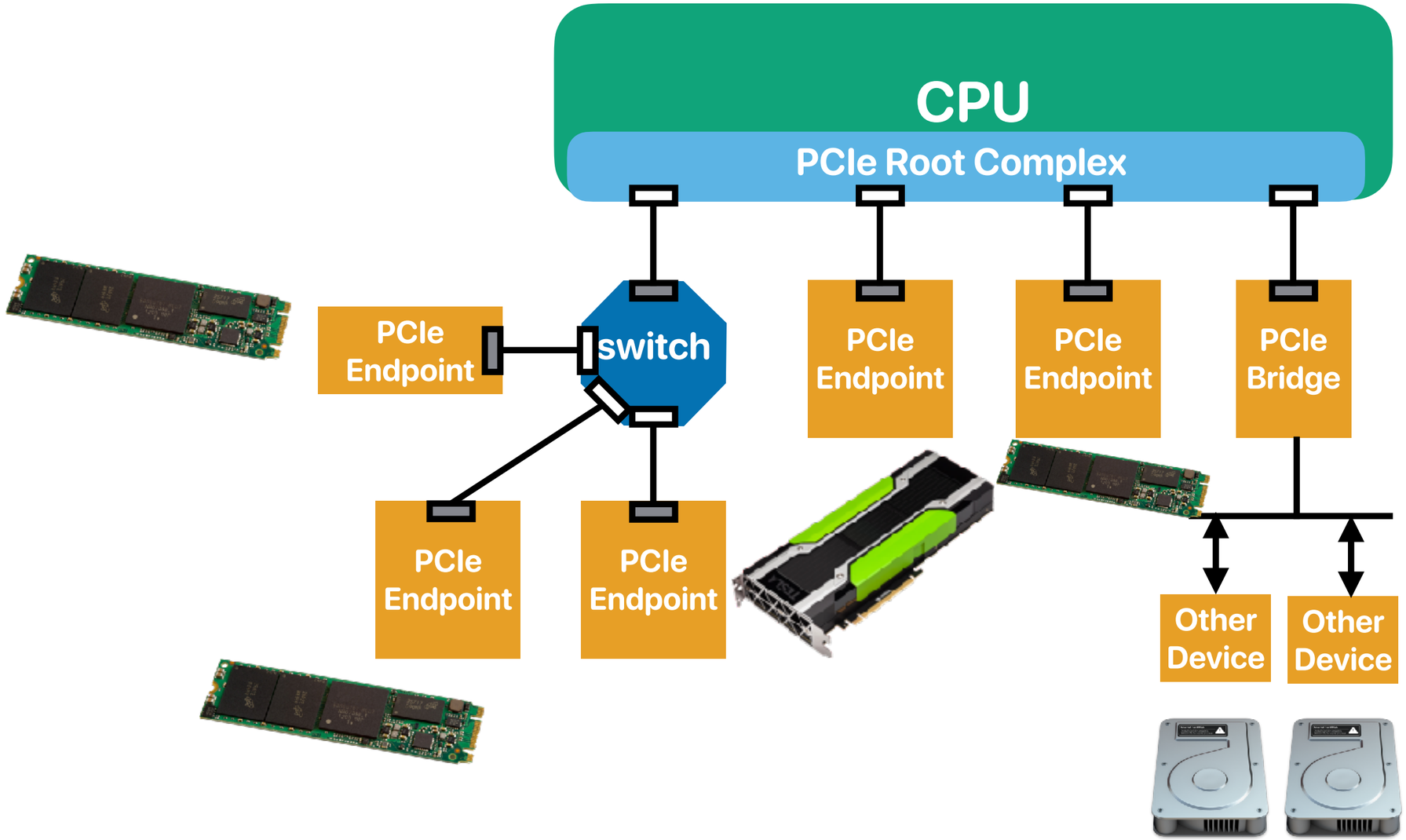
How fast is a ReRAM based accelerator?

How fast is a ReRAM accelerator?

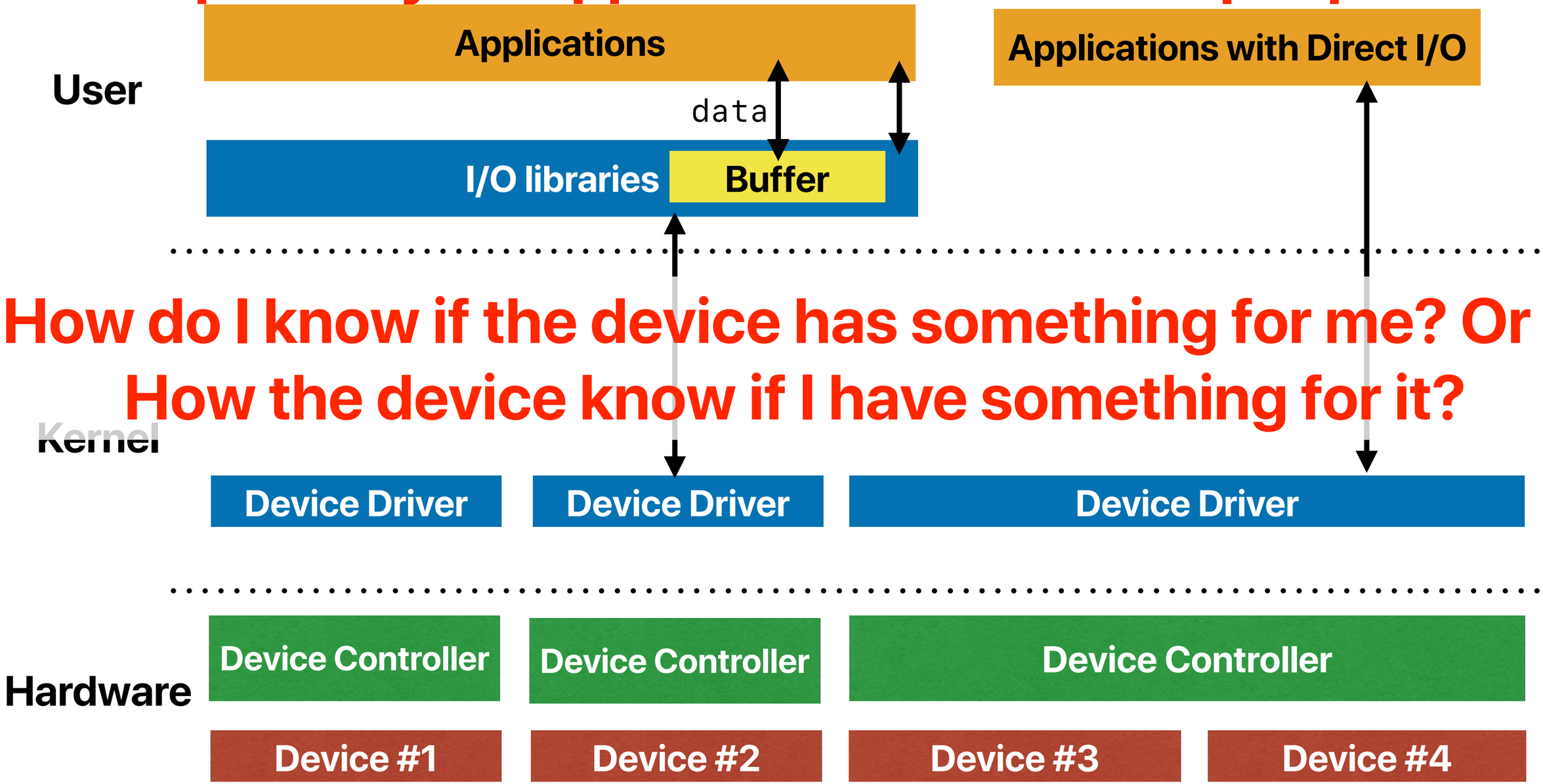
Recap: The landscape of modern computers



Recap: PCIe "Interconnect"



Recap: How your application interact with peripherals



How your CPU/OS interact with I/O devices

- The mechanism of knowing if there is an event
  - Interrupt
  - Polling
- The mechanism of handling the request
  - PIO
  - DMA

3:00

Which model is the best?

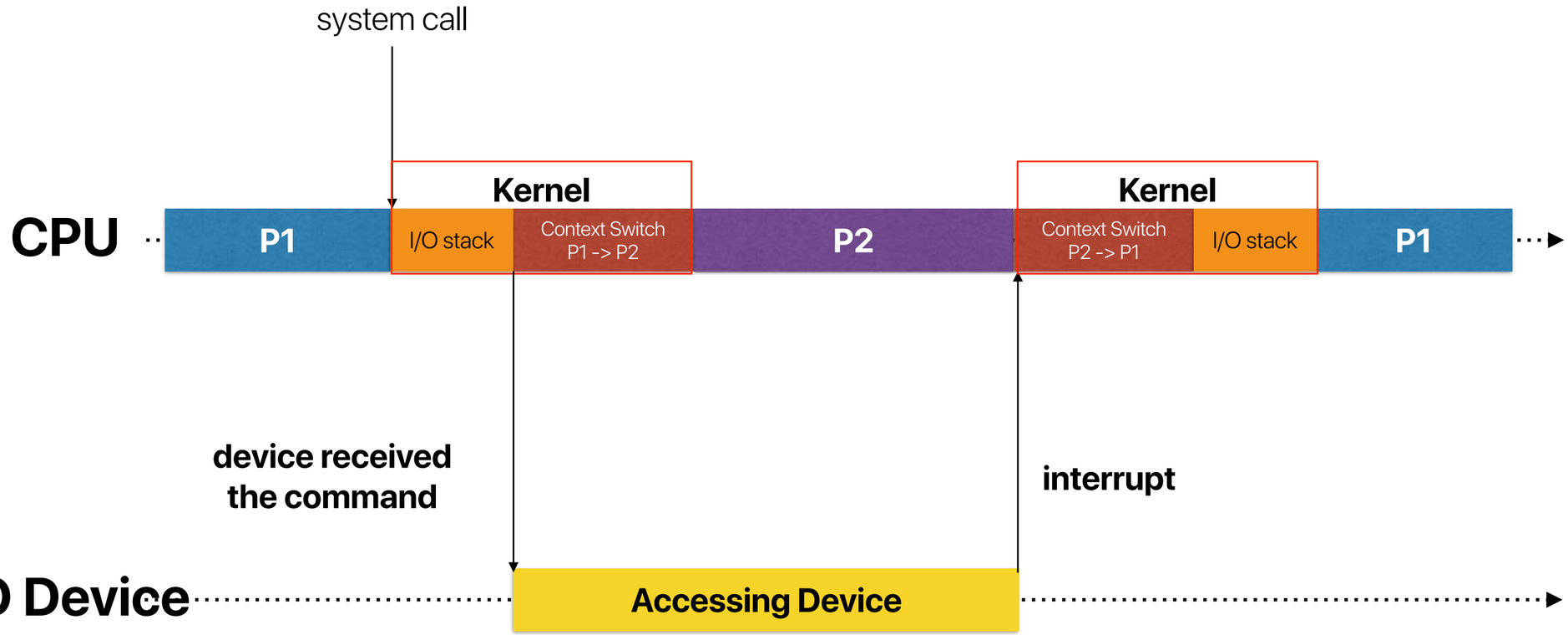
Polling/Interrupt/DMA/PIO?

	Polling	Interrupt
DMA		
PIO		

Recap: What happens during context switch

- Load architectural states from process control block (somewhere in the main memory, potentially a cache miss, TLB miss) — takes several microseconds if everything is in the physical memory
- Set processor registers according to the loaded architectural states
  - **Set the CR3 (page table base register in x86) register to identify the root page table node in the hierarchical page table**
  - Set the RIP (program counter in x86) to the previous execution
- **Restore virtual memory address**
  - **You must load the root page table node to the main memory at least.**
  - **TLB flush**
    - Invalidate all entries in the TLB
    - Most TLBs are not tagged, so you've to do this
- You DO NOT have to load every page content back from disk — remember that we have demand paging!

To switch or not to switch that's the question.



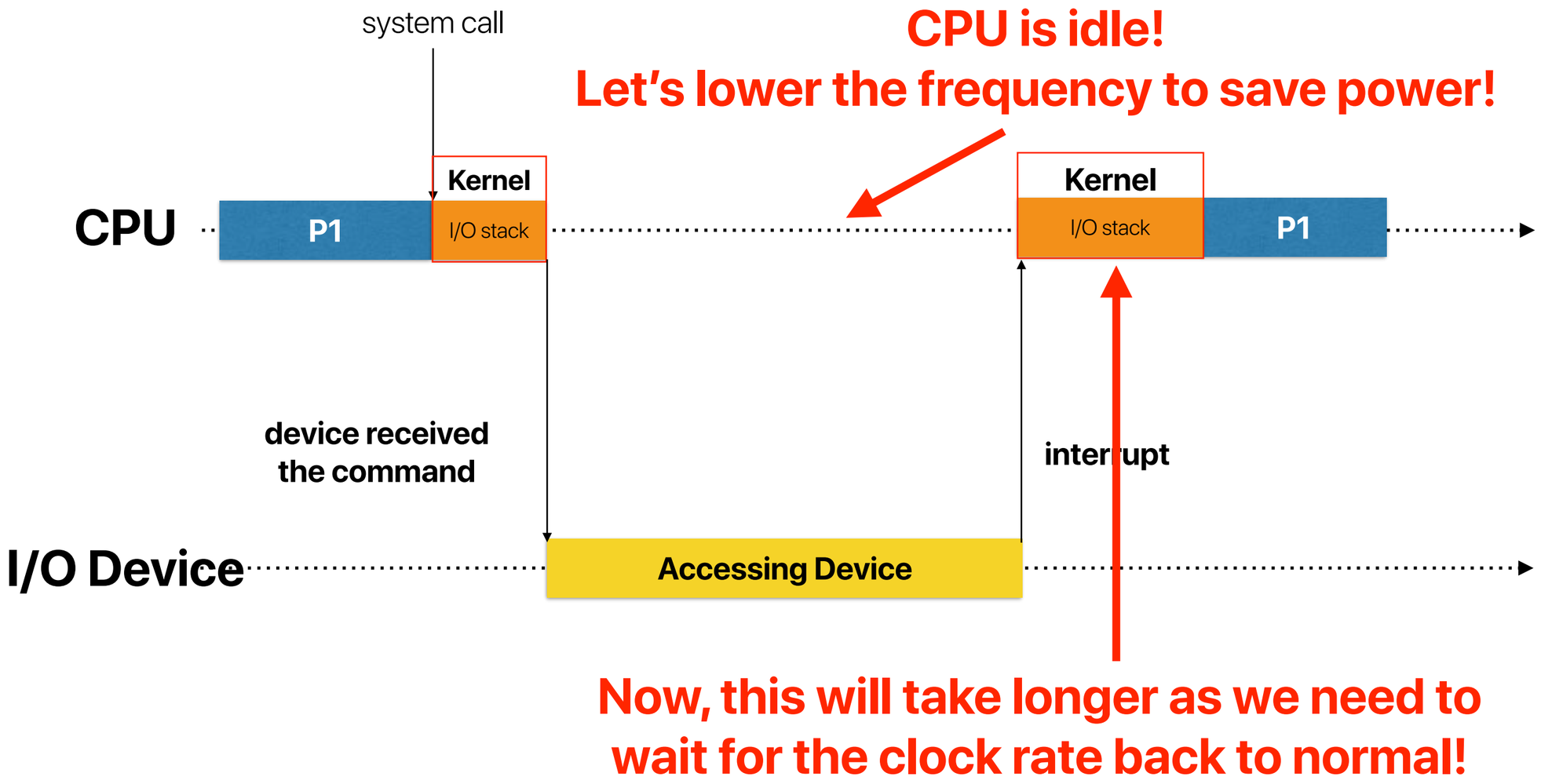
If $T_{\text{Context switch P1->P2}} + T_{\text{Context switch P2->P1}} < T_{\text{Accessing peripherals}}$

makes sense to context switch

But context switch overhead is not the only thing

- Cache warm up cost when you switch back
- TLB warm up cost

What if we don't switch?



Polling/Interrupt/DMA/PIO?

	Polling	Interrupt
DMA	CPU OPs more than IRQ Shorter latency High throughput	Lowest CPU Operations Longer latency High throughput
PIO	Most CPU operations Shorter latency Low throughput	CPU OPs more than IRQ Longer latency Low throughput

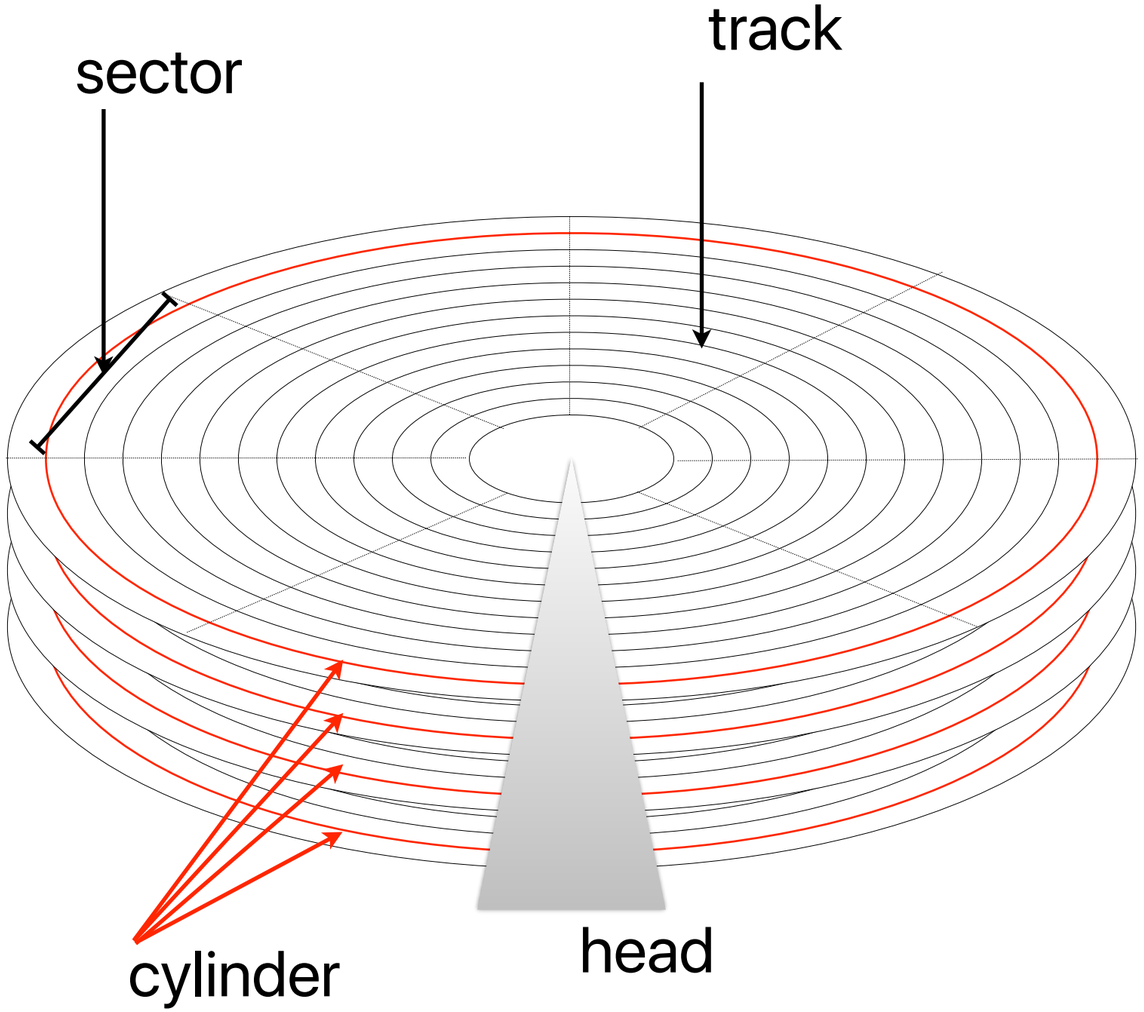
When should we poll? When should we interrupt

- Interrupt is only a good option if the benefit from context switching or energy saving is larger than waiting for the I/O to finish
- In general, applying polling on faster devices
  - DRAM
  - Non-volatile memory (e.g., flash, PCM)

Hard Disk Drives

Hard Disk Drive

Each sector is identified, locate by an "block address"



- Position the head to proper track (seek time)
- Rotate to desired sector. (rotational delay)
- Read or write data from/to disk to in the unit of sectors (e.g. 512B)
- Takes at least 5ms for each access

SATA Standard

		Speed
SATA Express	16 Gbit/s	1.97 GB/s[e]
SATA revision 3.0	6 Gbit/s	600 MB/
SATA revision 2.0	3 Gbit/s	300 MB/s
SATA revision 1.0	1.5 Gbit/s	150 MB/

Latency Numbers Every Programmer Should Know (2020 Version)

Operations	Latency (ns)	Latency (us)	Latency (ms)	
L1 cache reference	0.5 ns			~ 1 CPU cycle
Branch mispredict	3 ns			
L2 cache reference	4 ns			14x L1 cache
Mutex lock/unlock	17 ns			
Send 2K bytes over network	44 ns			
Main memory reference	100 ns			20x L2 cache, 200x L1 cache
Compress 1K bytes with Zippy	2,000 ns	2 us		
Read 1 MB sequentially from memory	3,000 ns	3 us		
Read 4K randomly from SSD*	16,000 ns	16 us		
Read 1 MB sequentially from SSD*	49,000 ns	49 us		
Round trip within same datacenter	500,000 ns	500 us		
Read 1 MB sequentially from disk	825,000 ns	825 us		
Disk seek	2,000,000 ns	2,000 us	2 ms	4x datacenter roundtrip
Send packet CA-Netherlands-CA	150,000,000 ns	150,000 us	150 ms	

https://colin-scott.github.io/personal_website/research/interactive_latency.html

Seagate Barracuda 12

- SATA II (300MB/s in theory), 7200 R.P.M., seek time around 8 ms. Assume the controller overhead is 0.2ms. What's the **latency** and **bandwidth** of accessing a **512B** sector?

Latency = seek time + rotational delay + transfer time + controller overhead

$$8\ ms \quad +\frac{1}{2}\times\frac{1}{\frac{7200}{60}} \quad +\frac{\frac{0.5}{1024}}{300} \quad +0.2\ ms$$

$$= 8\ ms + 4.17\ ms + 0.00167\ us + 0.2\ ms = 12.36\ ms$$

Bandwidth = volume_of_data over period_of_time

$$= \frac{0.5KB}{12.36ms} = 40.45KB/sec$$

Seagate Barracuda 12

- SATA II (300MB/s in theory), 7200 R.P.M., seek time around 8 ms. Assume the controller overhead is 0.2ms. What's the **latency** and **bandwidth** of accessing consecutive **4MB** data?

Latency = seek time + rotational delay + transfer time + controller overhead

$$8\ ms \quad +\frac{1}{2}\times\frac{1}{\frac{7200}{60}} \qquad +\frac{4}{300} \quad +0.2\ ms$$

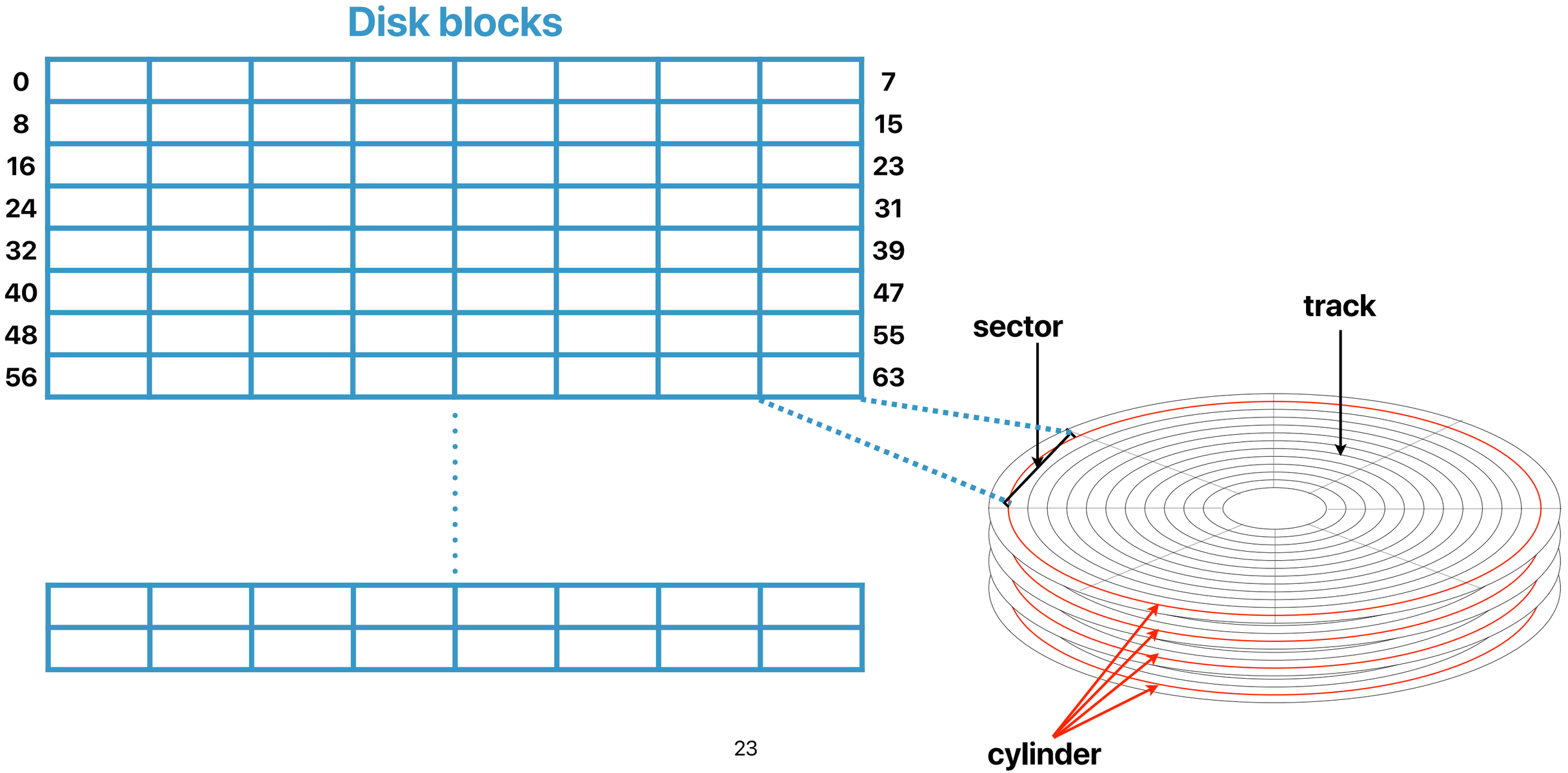
$$= 8\ ms + 4.17\ ms + 13.33\ ms + 0.2\ ms = 25.69\ ms$$

Bandwidth = volume_of_data over period_of_time

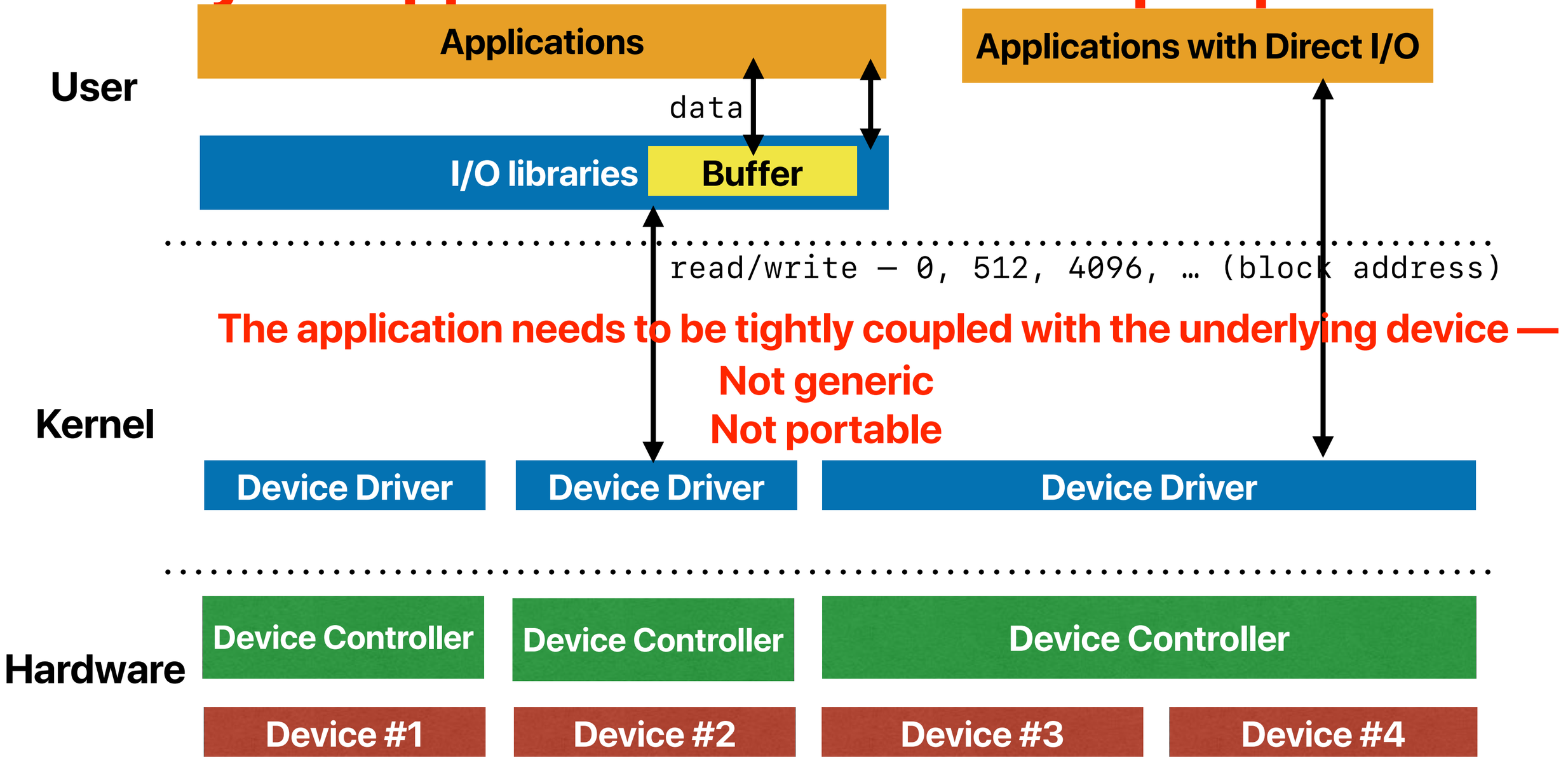
$$= \frac{4MB}{25.69ms} = 155.7\ MB/sec$$

Trading latencies with bandwidth

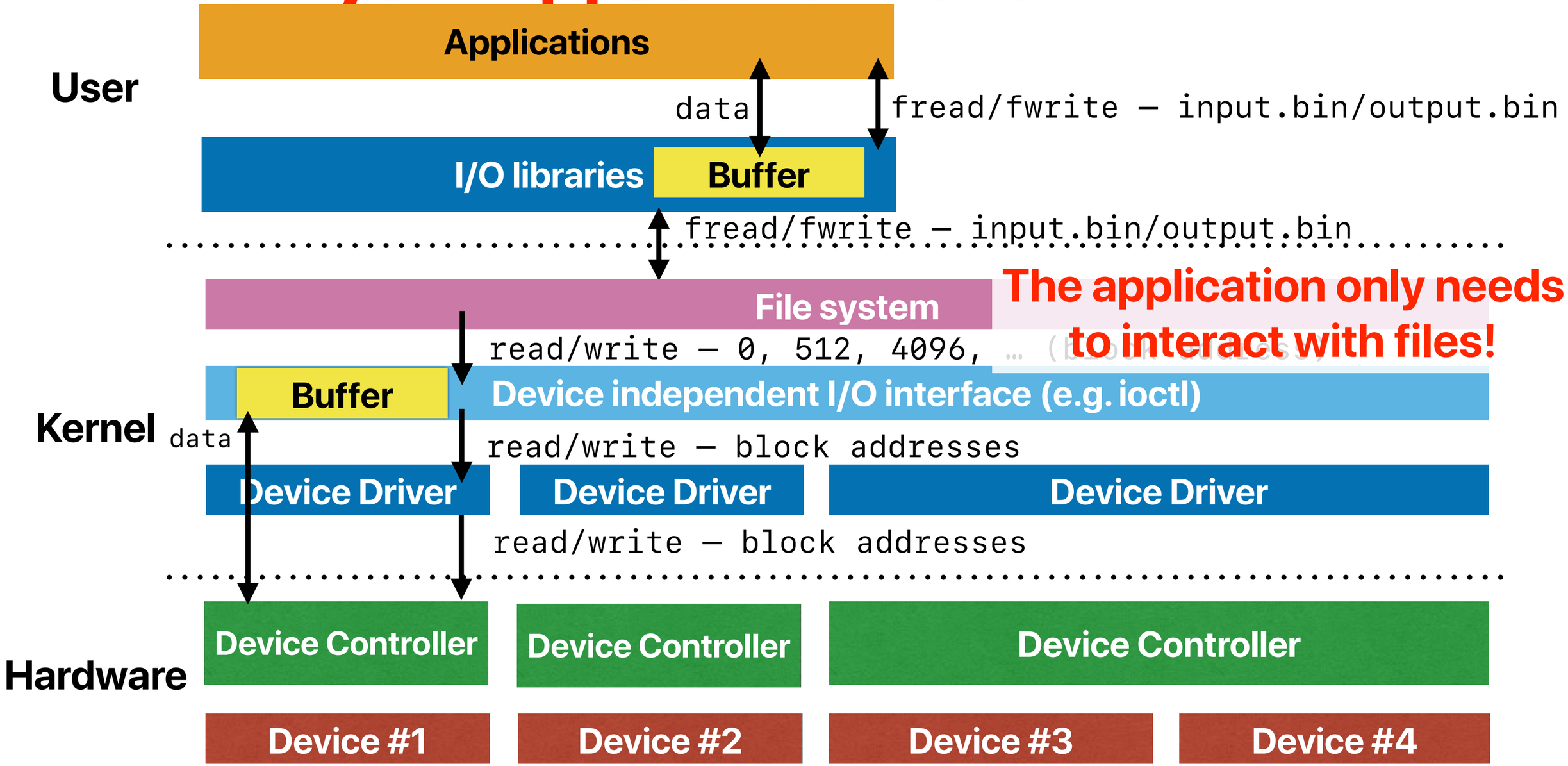
Numbering the disk space with block addresses



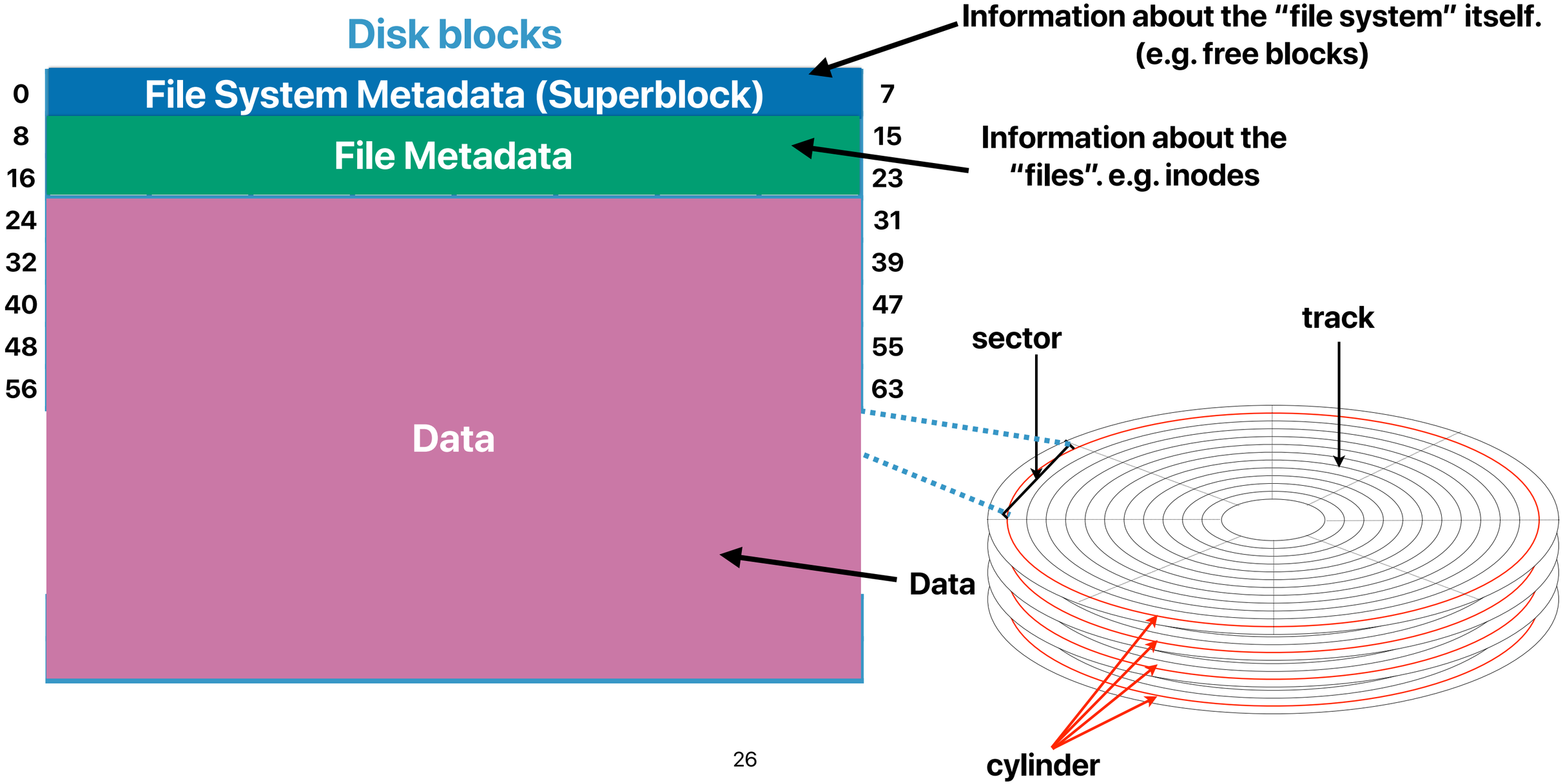
How your application interact with peripherals



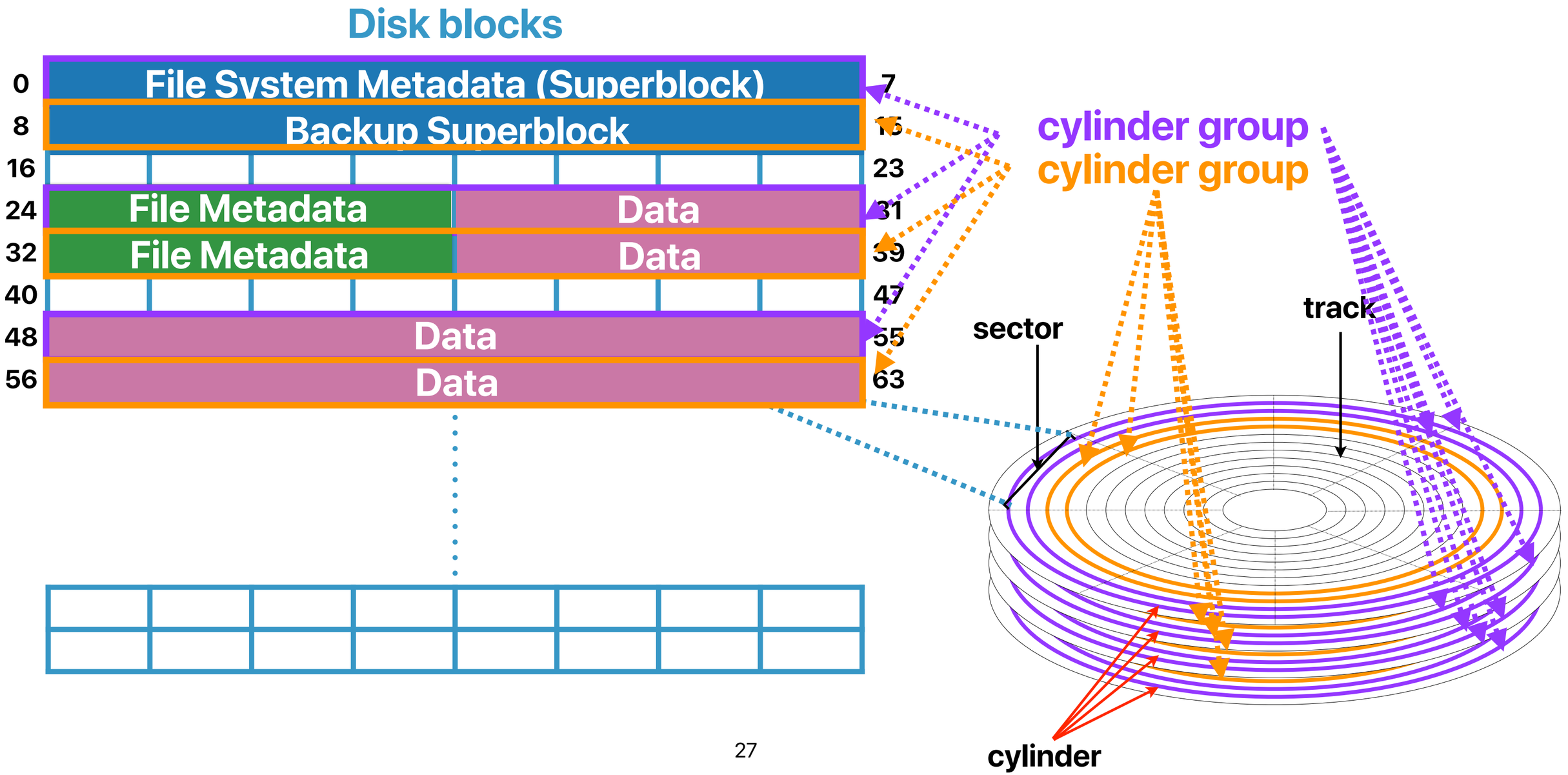
How your application reaches H.D.D.



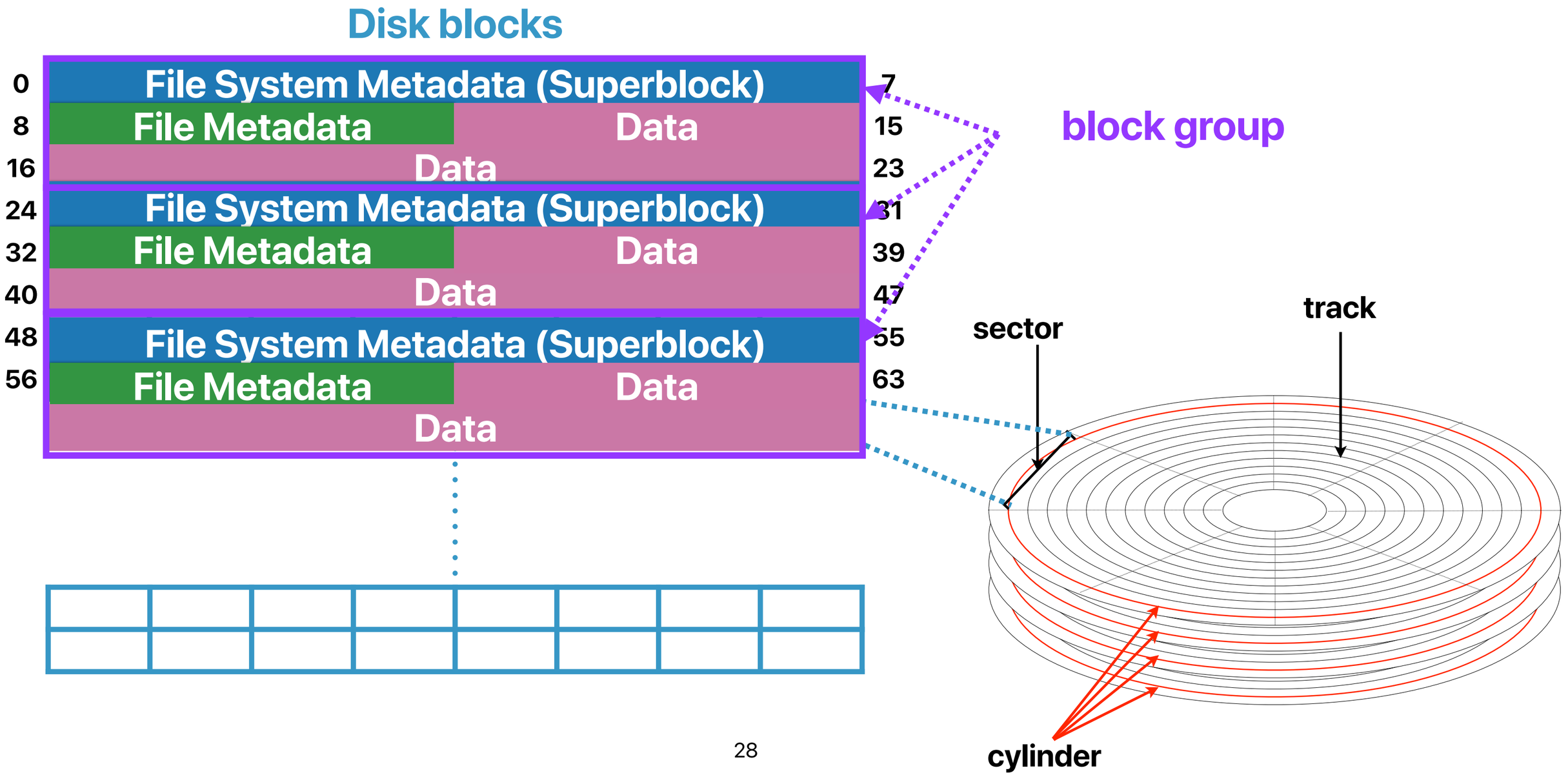
How the original UNIX file system use disk blocks



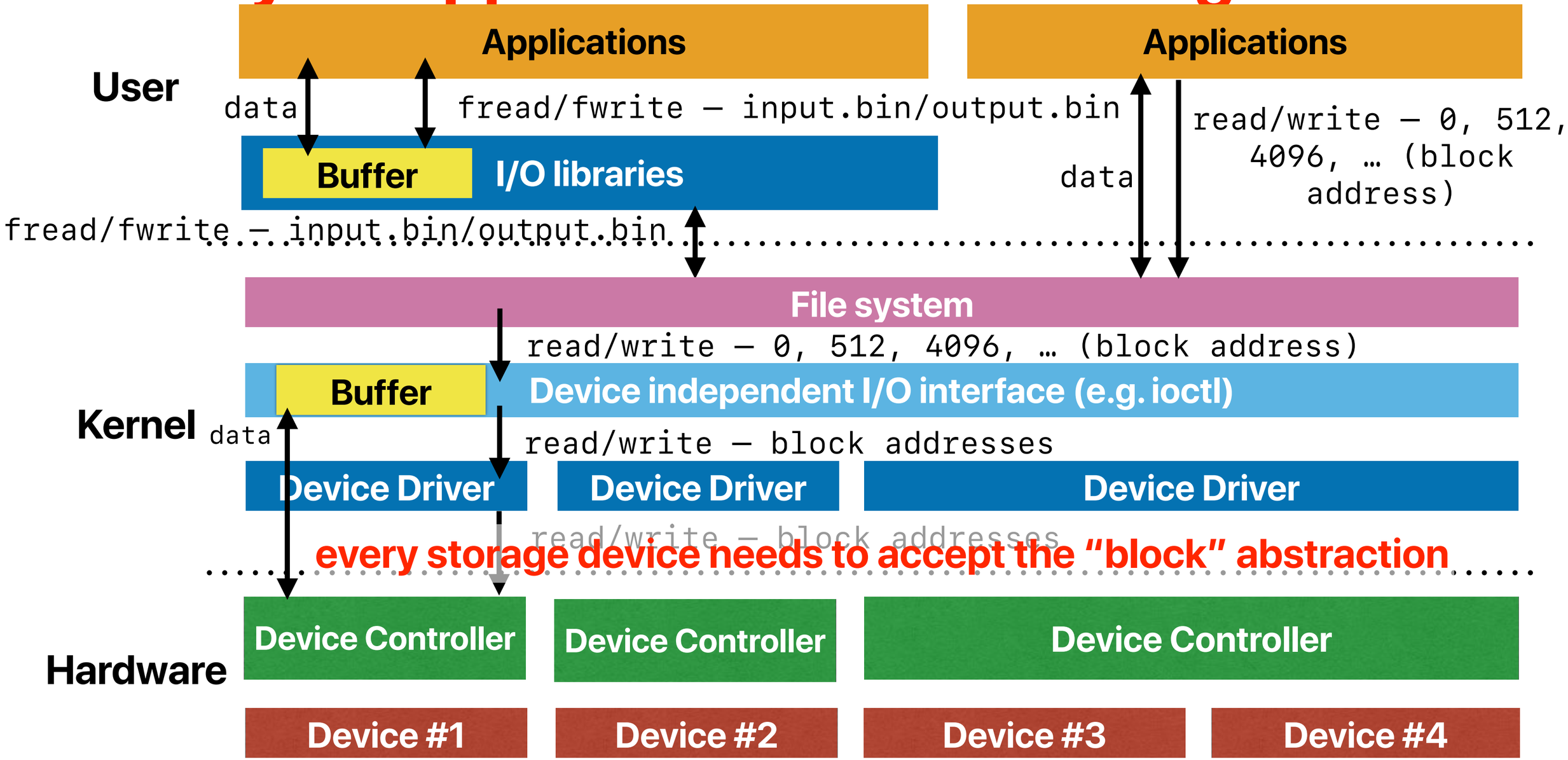
How FFS use disk blocks



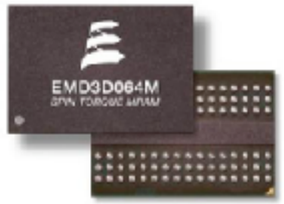
How ExtFS use disk blocks



How your application reaches storage devices



Non-volatile memory technologies



	H.D.D	Flash	Optane	STT-MRAM
Latency	~ 10-15 ms	~ 100 us (read) ~ 1 ms (write)	7 us (read) 18 us (write)	35 ns
Bandwidth	~200 MB/Sec	3.5 GB/sec (read) 2.1 GB/sec (write)	1.35 GB/sec (read) 290 MB/sec (write)	
Dollar/GB	0.0295	0.583	2.18	

Flash is still the most convincing technology for now

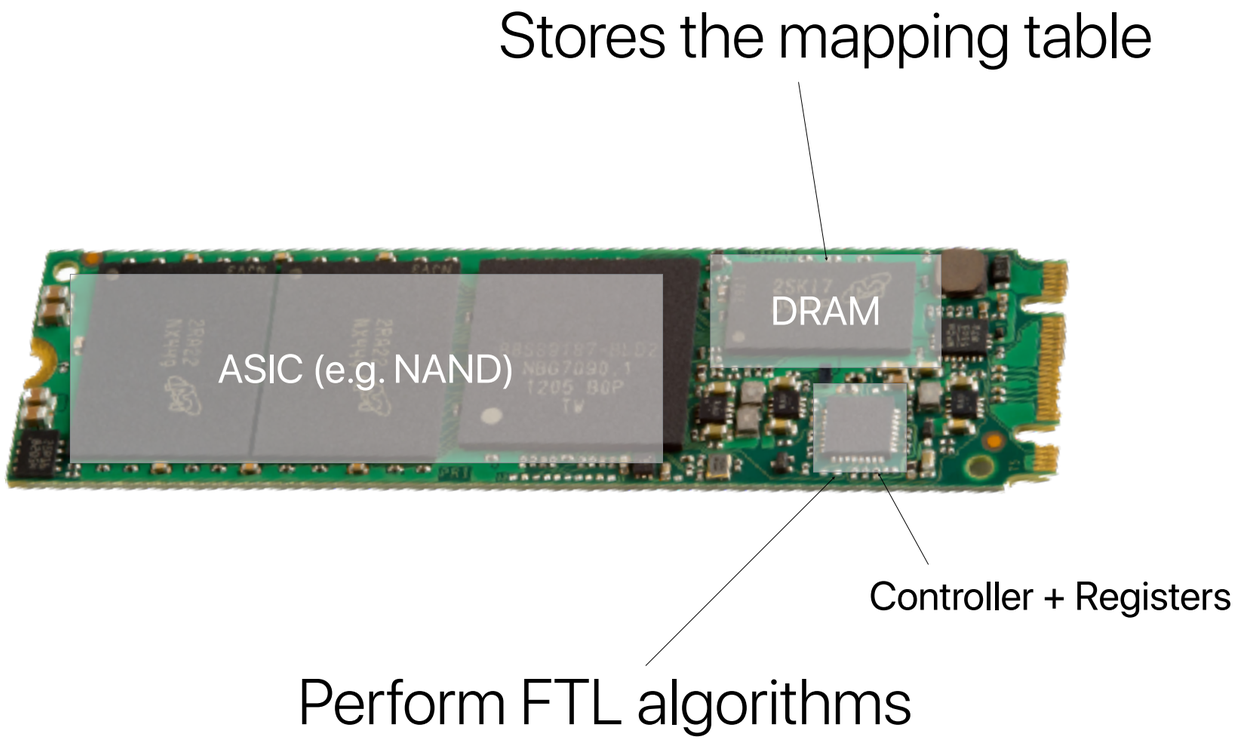
The architecture of an SSD

Controller + Registers

3:00

Why do we need a controller inside an SSD?

The architecture of an SSD



NAND flash is just odd!

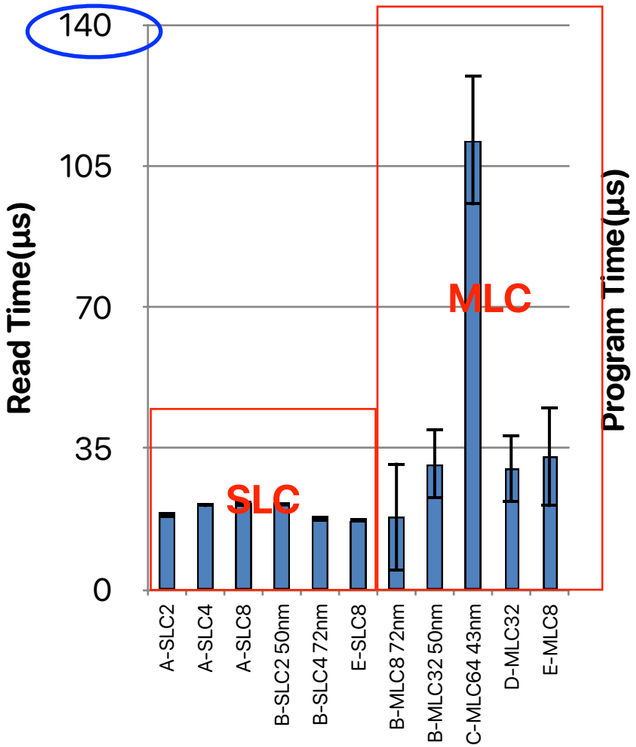
- Modern SSDs are based on NAND flash memory
- Different operation granularities
  - Read/Program in pages
  - Erase in blocks (64-384 pages)
- Performance of operation varies
  - Read — tens of us
  - Program — hundreds of us
  - Erase — ms
- Limited erase cycles
  - Only 1000 times for “QLC”

Why do we need controllers in SSDs?

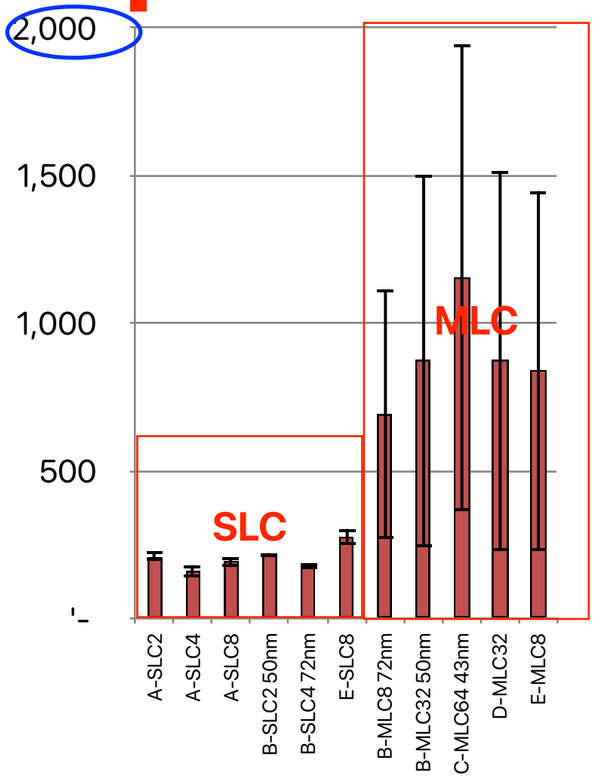
- Maintaining the block device abstraction
- Dealing with the “weird” device characteristics
- Interfacing with the interconnect

Flash performance

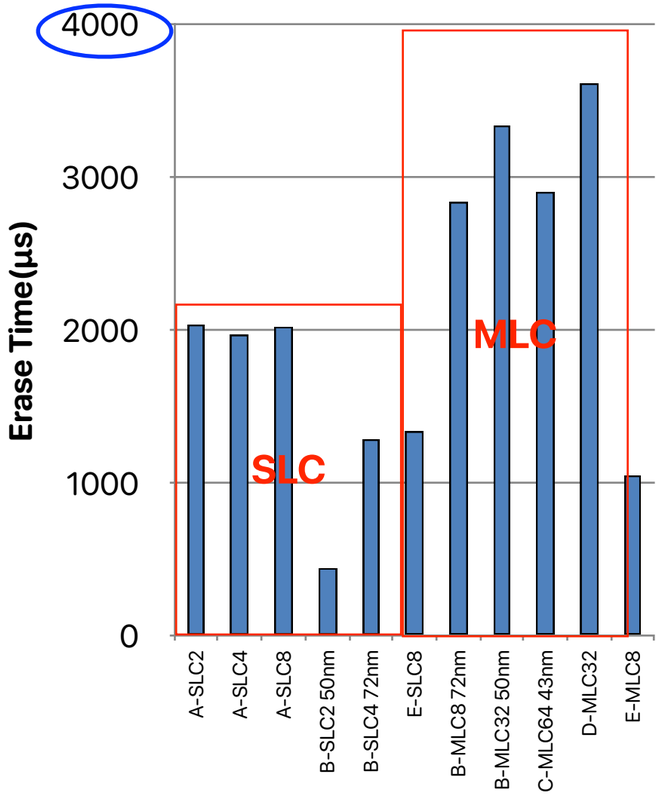
Not a good practice



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.

Firmware is …

- A software program directly interacts with hardware without an operating system
- Typically a program works in between hardware and application “software”

Why we need a processor in SSDs?

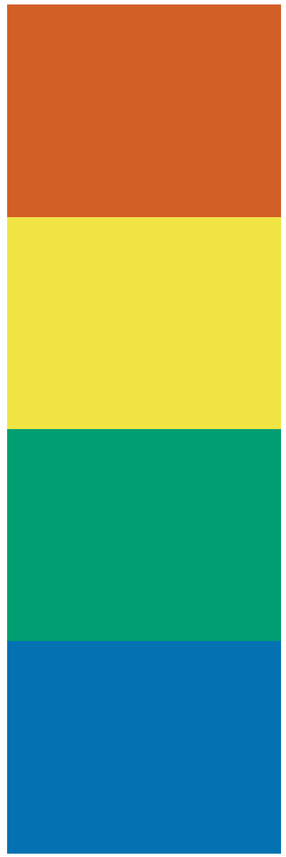
- Modern SSDs are based on NAND flash memory
- Different operation granularities
  - Read/Program in pages
  - Erase in blocks (64-384 pages)
- Performance of operation varies
  - Read — tens of us
  - Program — hundreds of us
  - Erase — ms
- Limited erase cycles
  - Only 1000 times for “QLC”

Types of Flash Chips

2 voltage levels, 1-bit

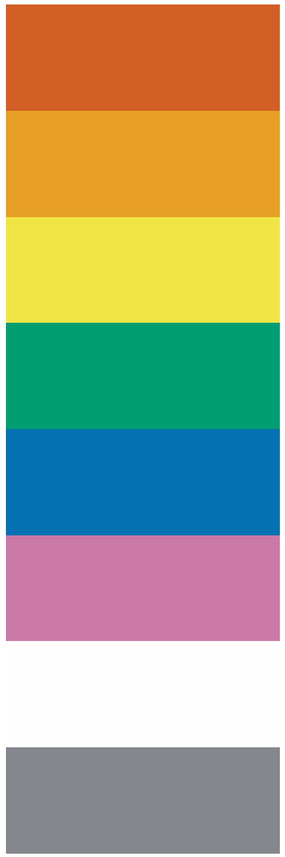
Single-Level Cell (SLC)

4 voltage levels, 2-bit



Multi-Level Cell (MLC)

8 voltage levels, 3-bit



Triple-Level Cell (TLC)

16 voltage levels, 4-bit

Quad-Level Cell (QLC)

Programming in MLC

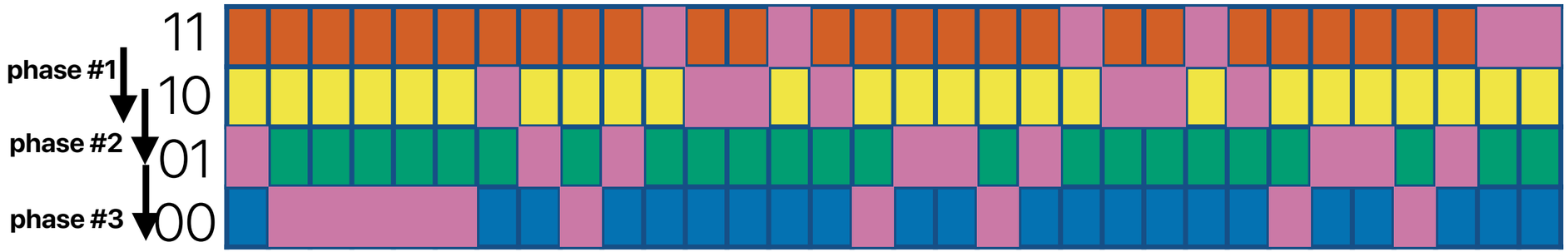
4 voltage levels,
2-bit

Multi-Level Cell (MLC)

3.14000000000000012434497875800

= 0x40091EB851EB851F

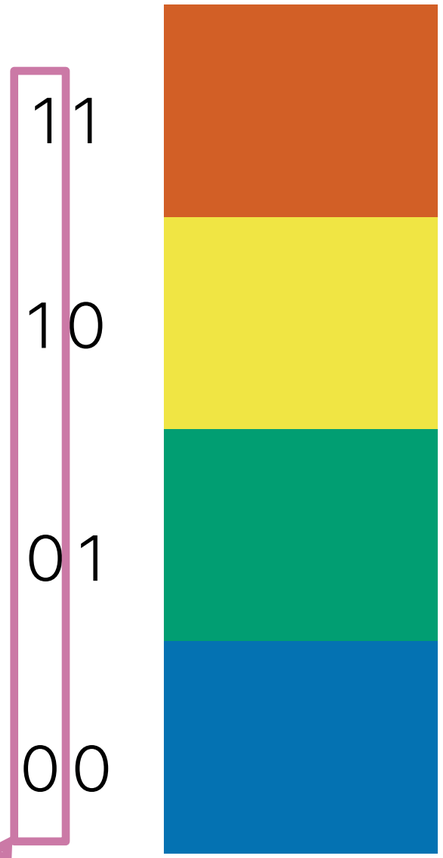
= 01000000 00001001 00011110 10111000 01010001 11101011 10000101 00011111



3 Cycles/Phases to finish programming

Programming in MLC

4 voltage levels,
2-bit



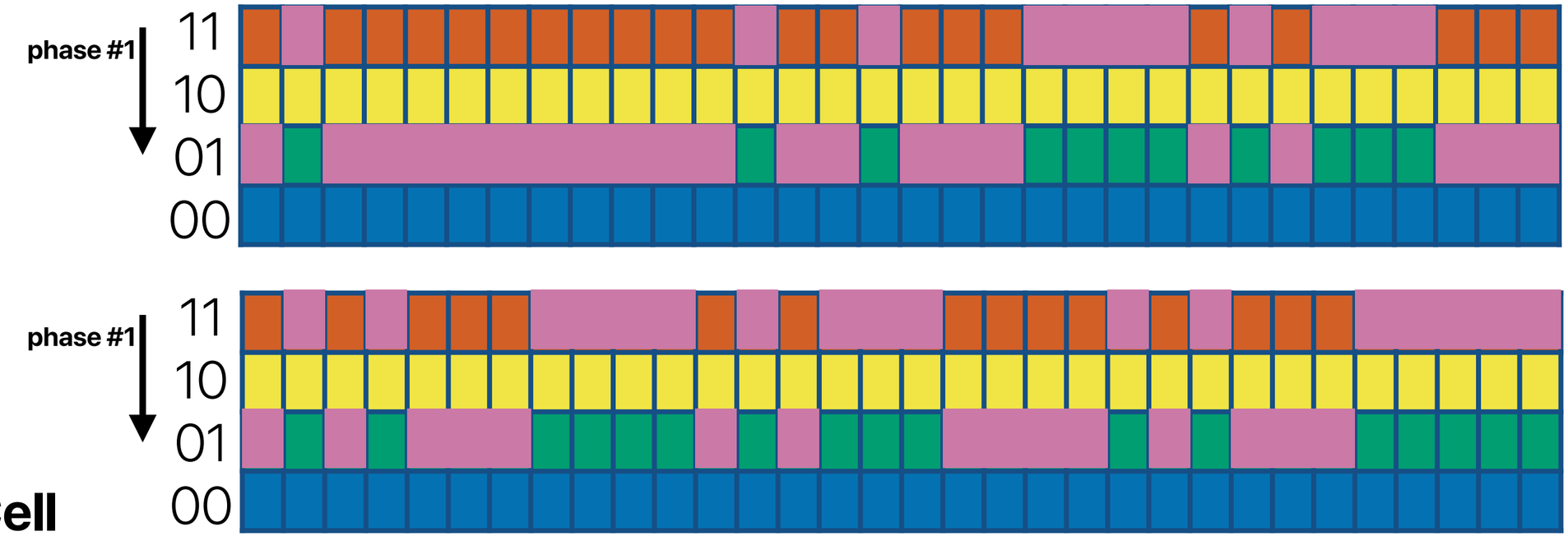
Multi-Level Cell (MLC)

1st page

3.14000000000000012434497875800

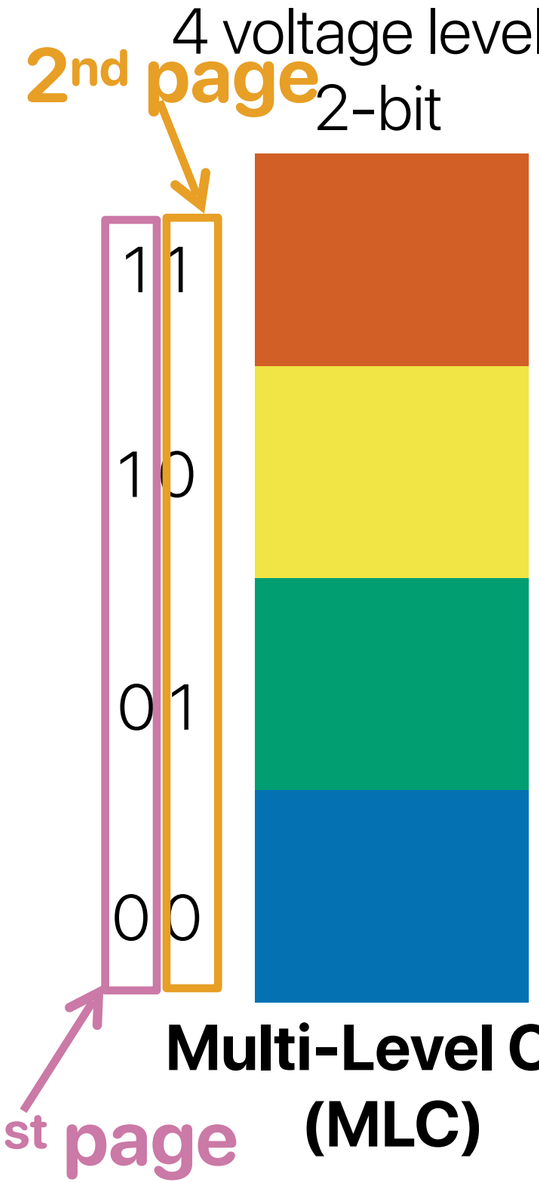
= 0x40091EB851EB851F

= 01000000 00001001 00011110 10111000 01010001 11101011 10000101 00011111



1 Phase to finish programming the first page!

Programming the 2nd page in MLC

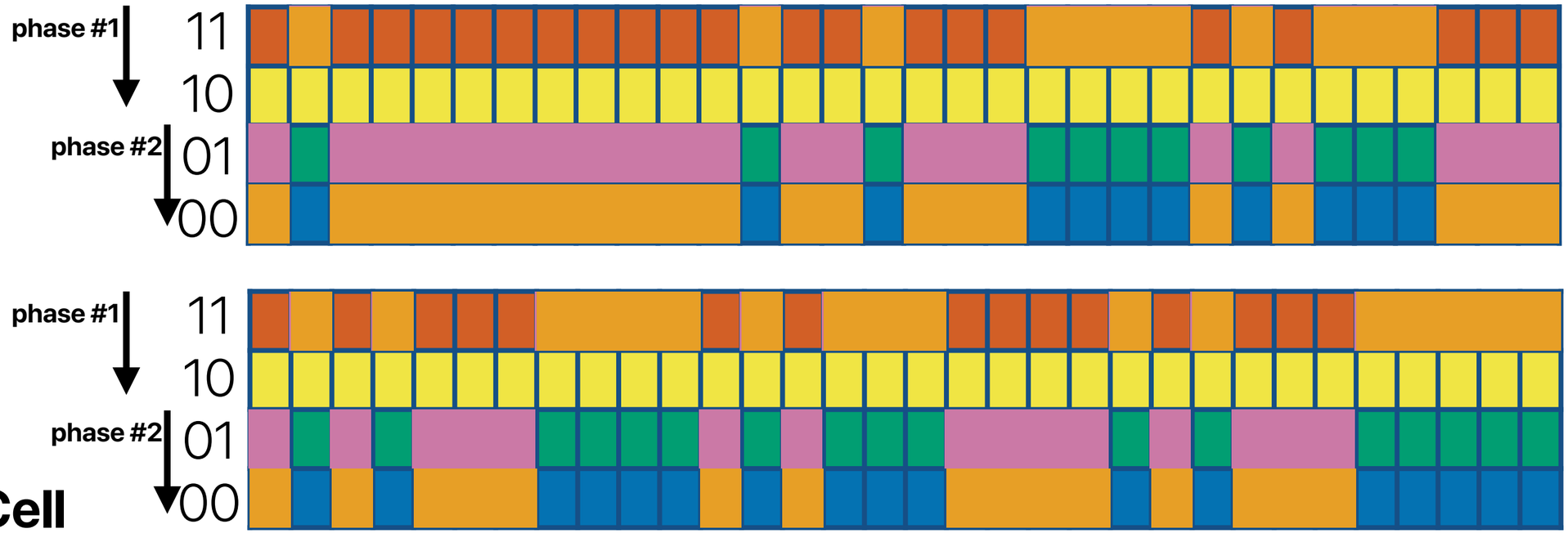


3.14000000000000012434497875800

= 0x40091EB851EB851F

= 01000000 00001001 00011110 10111000 01010001 11101011 10000101 00011111

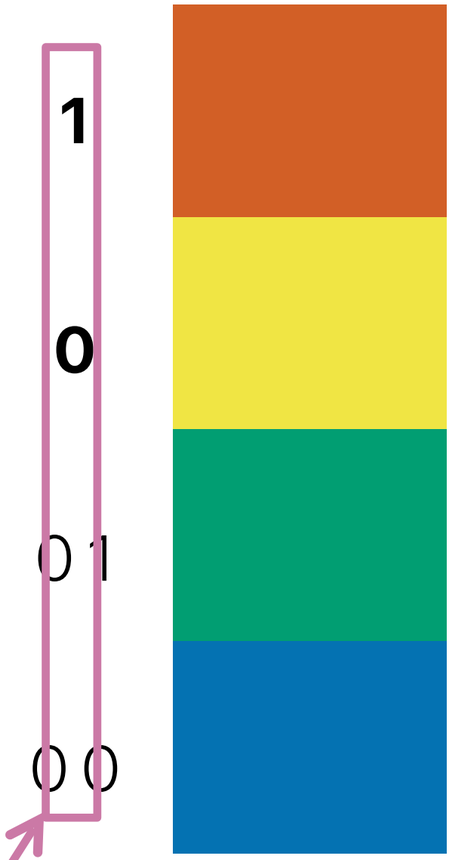
= 01000000 00001001 00011110 10111000 01010001 11101011 10000101 00011111



2 Phase to finish programming the second page!

Optimizing 1st Page Programming in MLC

4 voltage levels,
2-bit



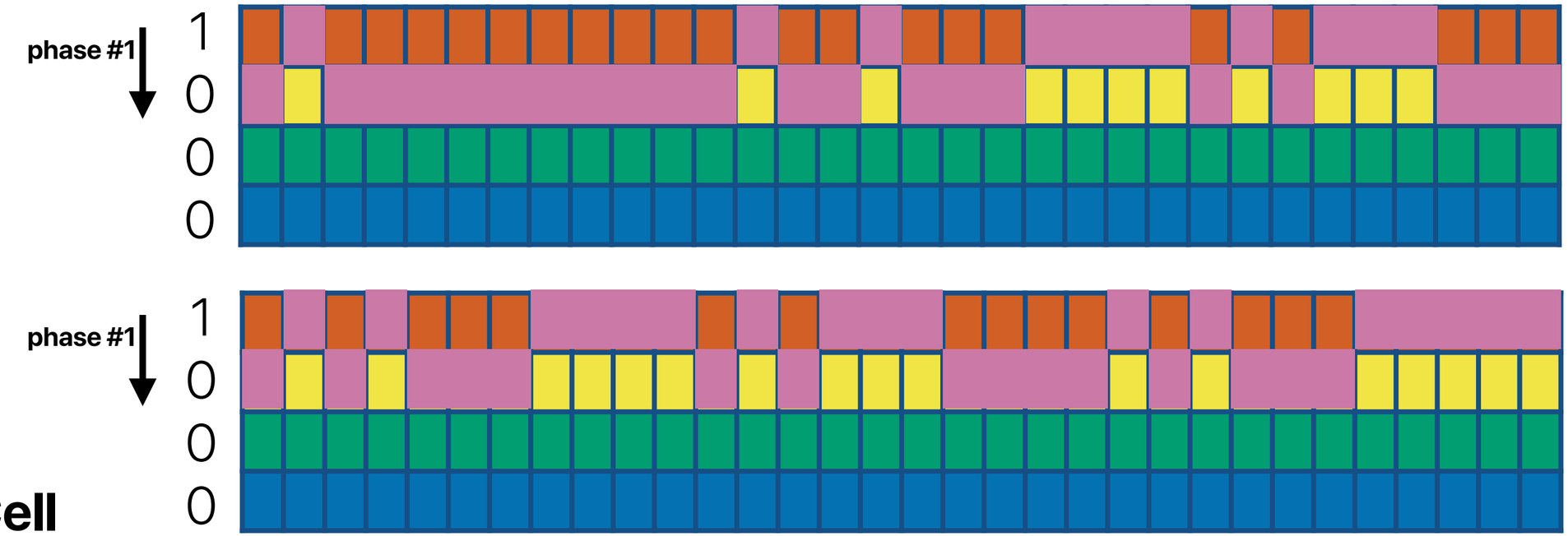
Multi-Level Cell (MLC)

1st page

3.14000000000000012434497875800

= 0x40091EB851EB851F

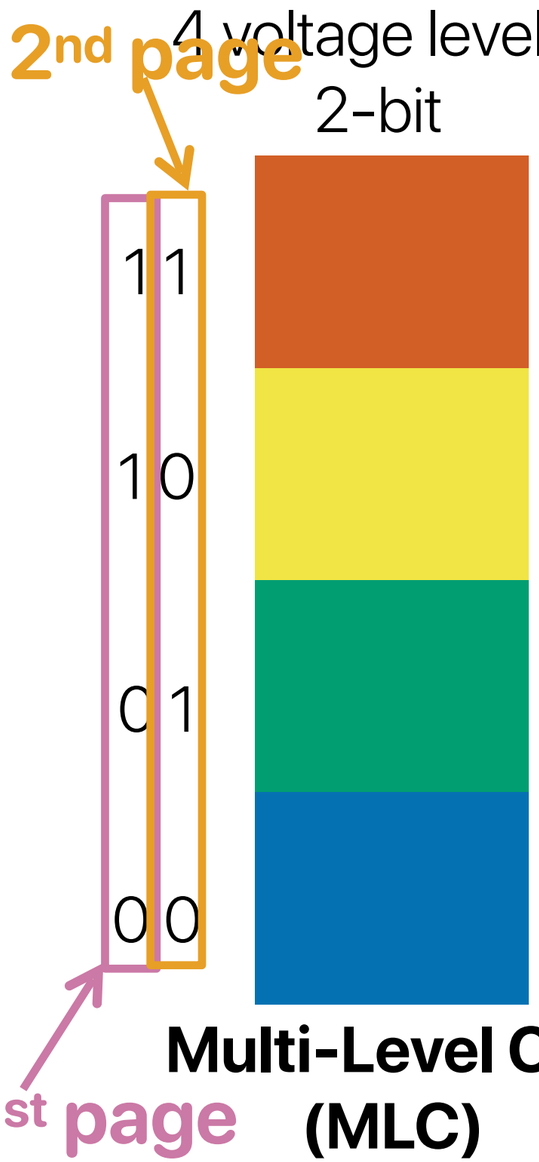
= 01000000 00001001 00011110 10111000 01010001 11101011 10000101 00011111



1 Phase to finish programming the first page!

— the phase is shorter now

2nd Page Programming in MLC

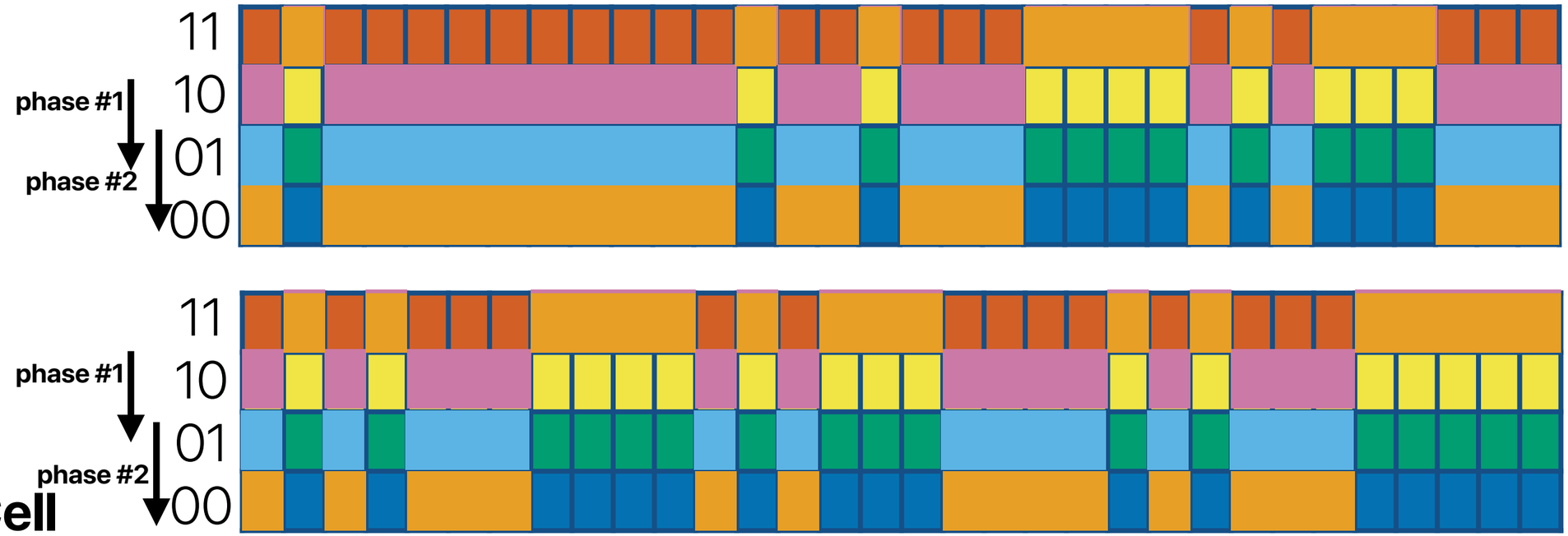


3.14000000000000012434497875800

= 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

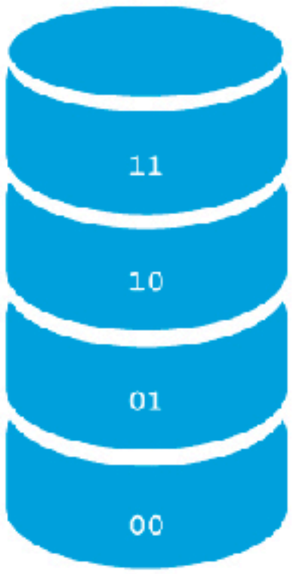
Lower $ per GB

SLC

1 Bit Per Cell

First SSD NAND technology

100K P/E Cycles
(at technology introduction)

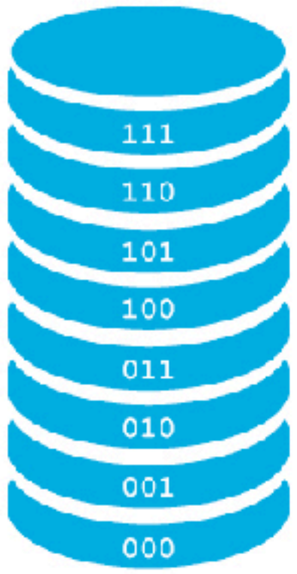


MLC

2 Bits Per Cell

100% increase

10K P/E Cycles

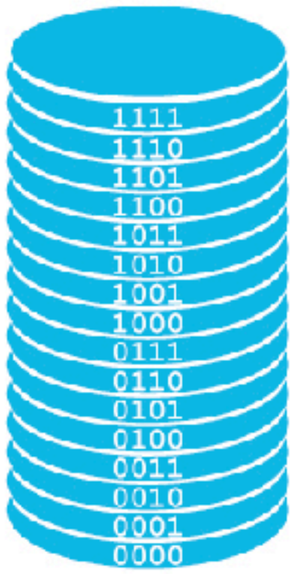


TLC

3 Bits Per Cell

50% increase

3K P/E Cycles



QLC

4 Bits Per Cell

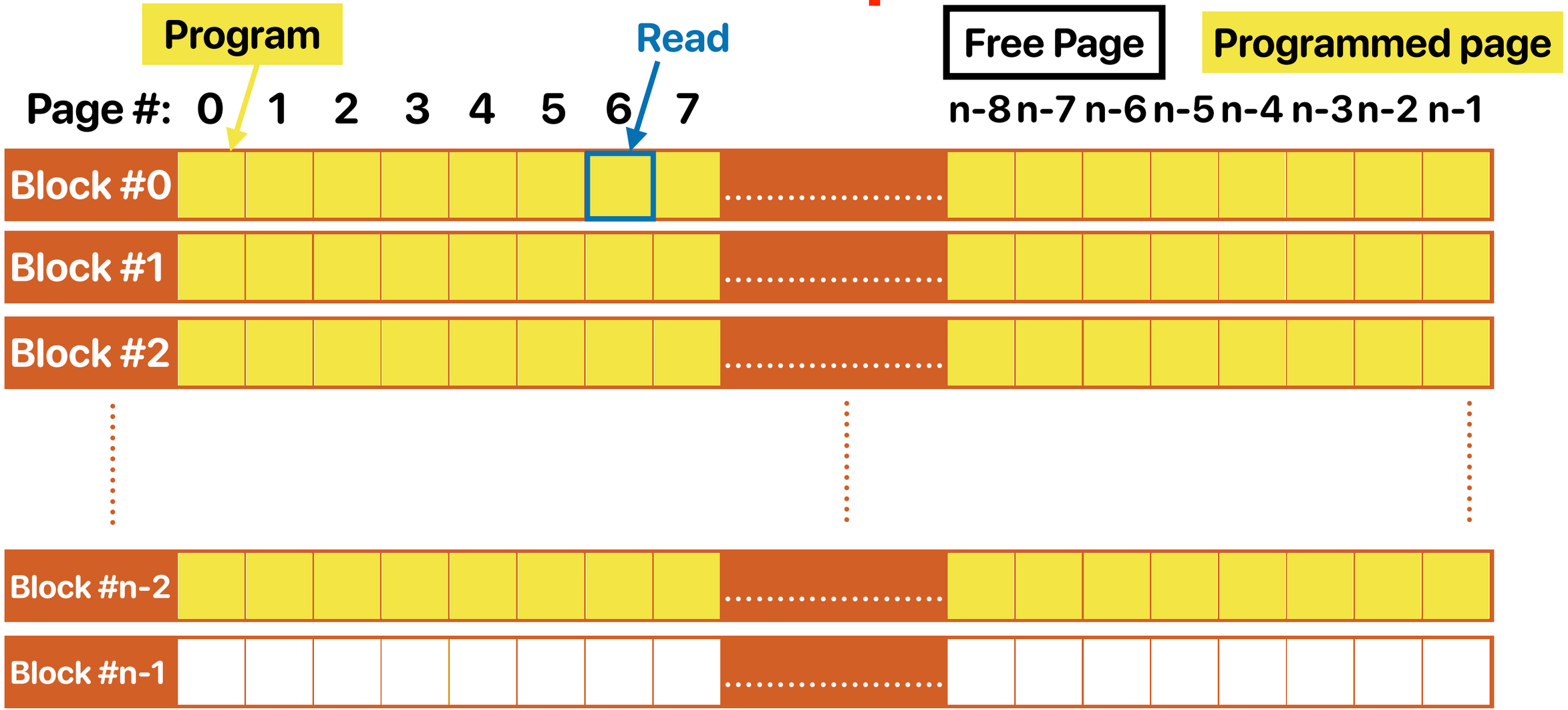
33% increase

1K P/E Cycles

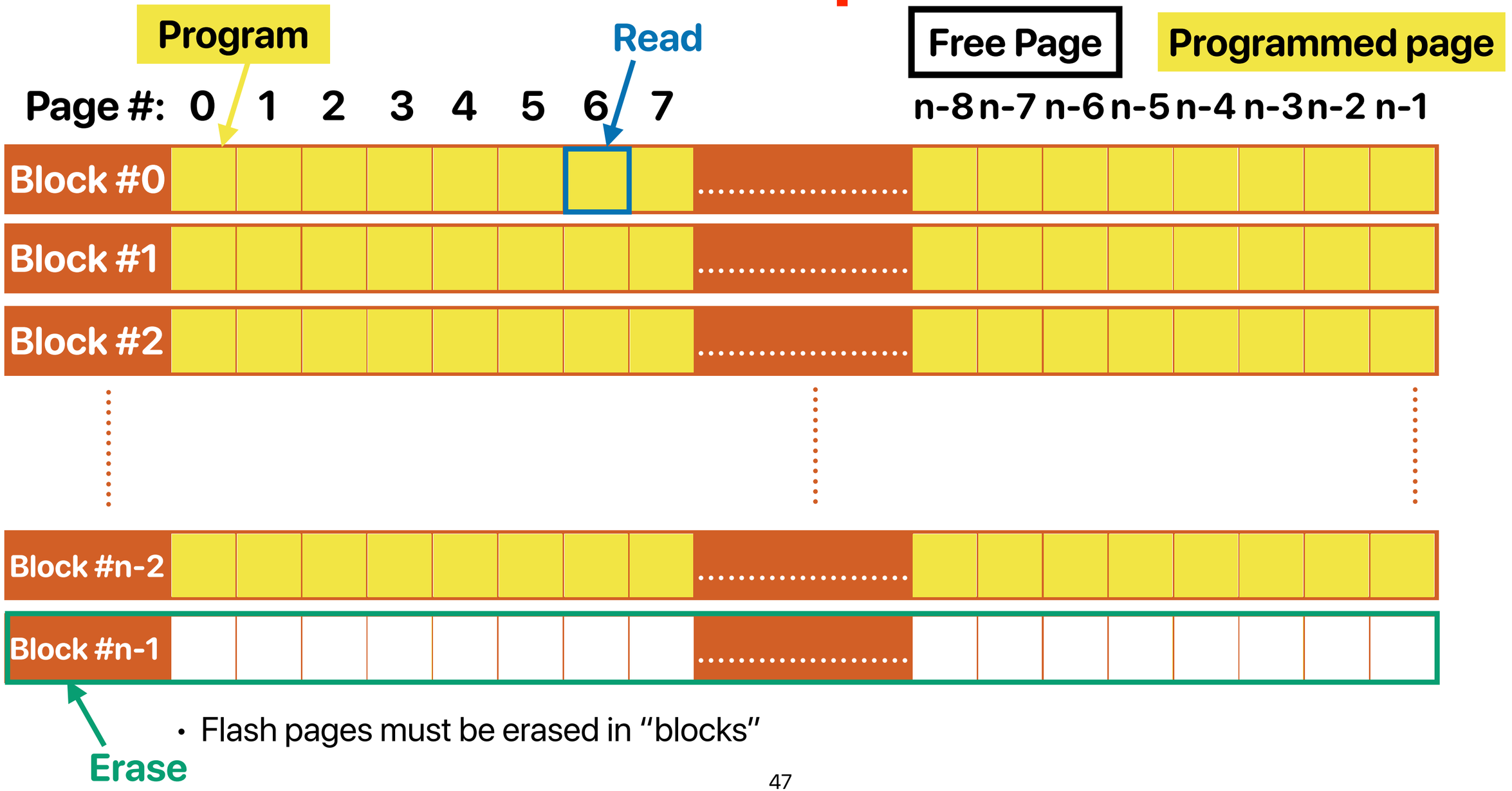
Fewer writes per cell

Micron

Basic flash operations

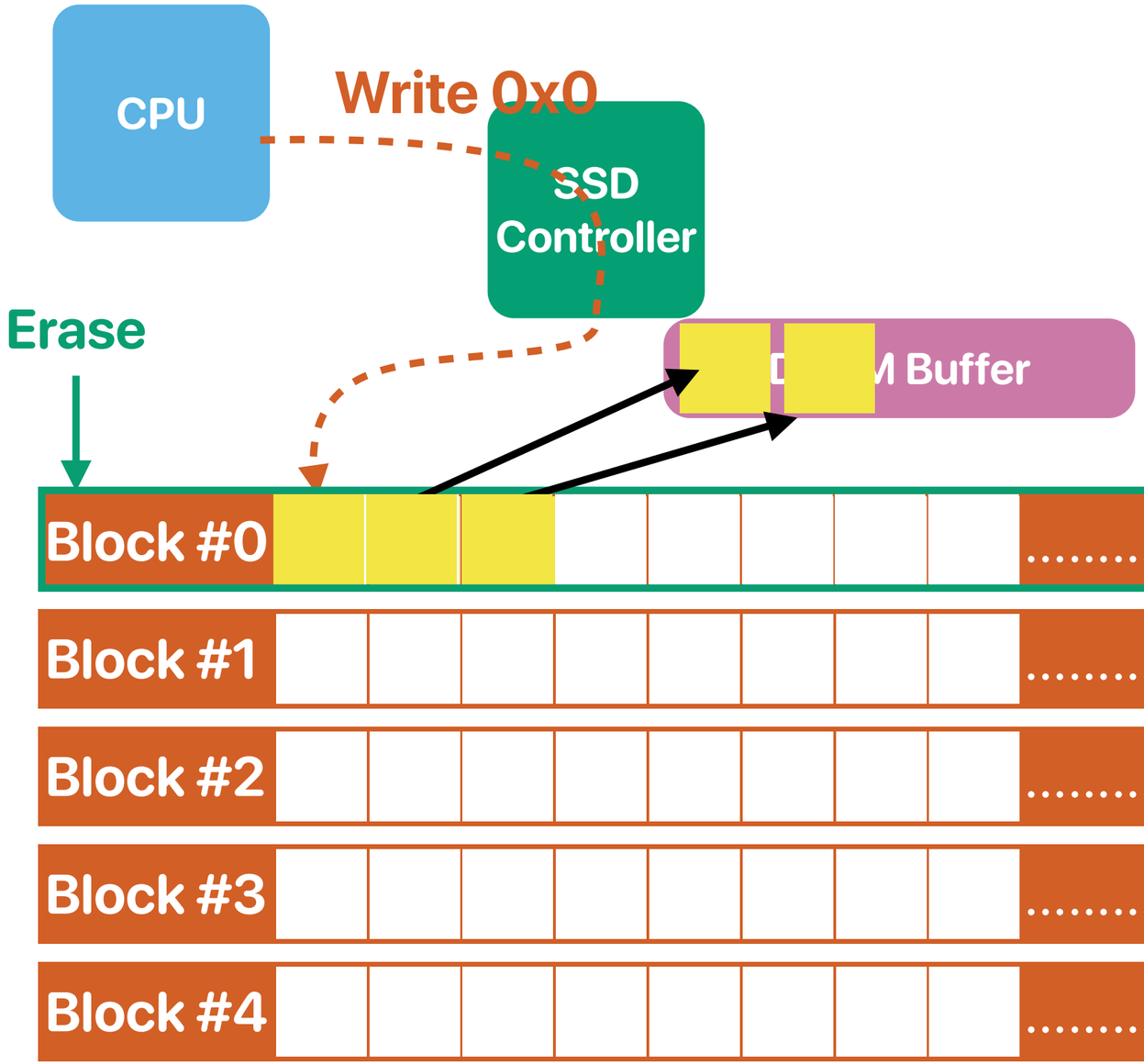


Basic flash operations



- Flash pages must be erased in "blocks"

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



Can we write to page #0 directly? No.

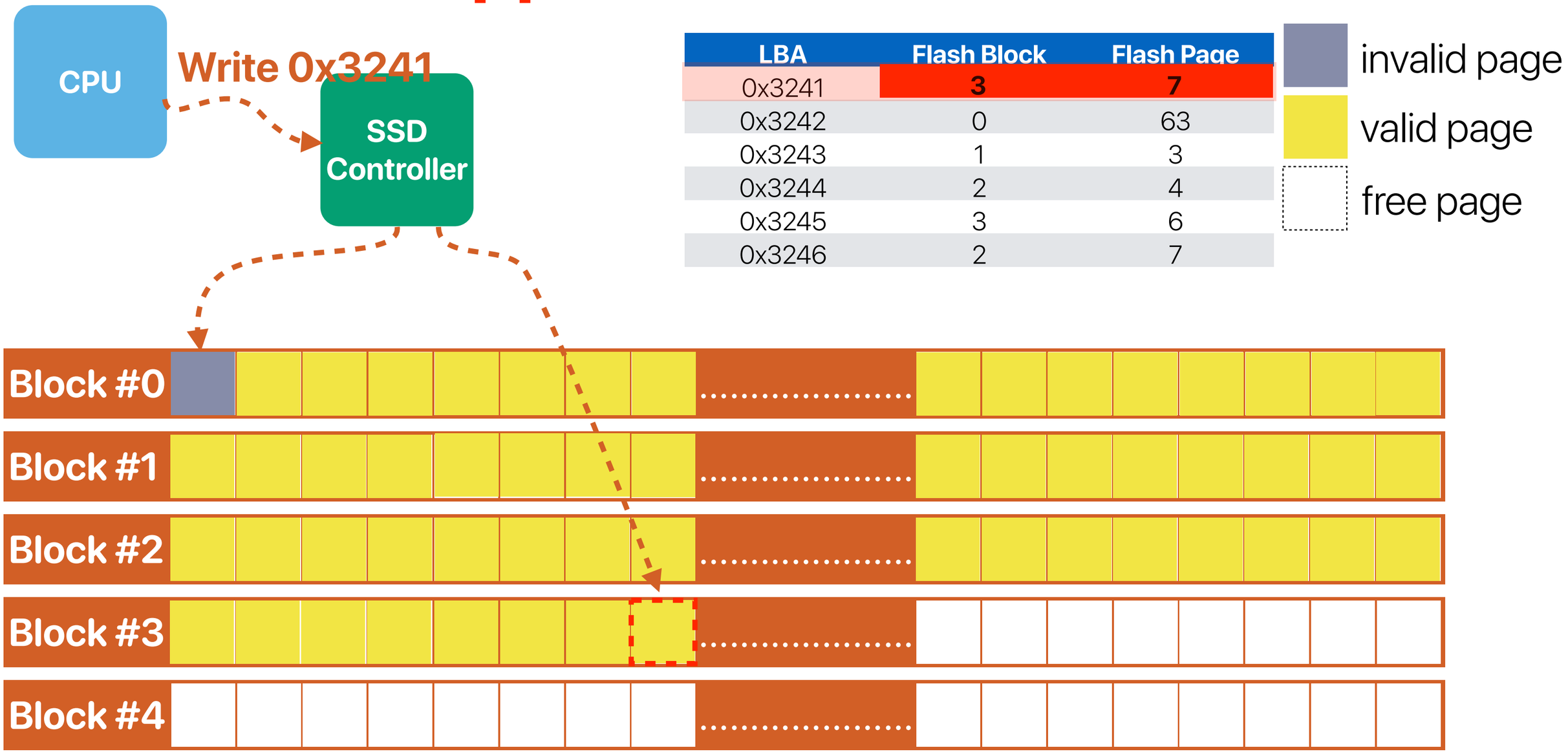
We have to copy page #1, page #2 in block #0 to somewhere (e.g. RAM buffer) and then erase the block

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

What happens on a write with FTL



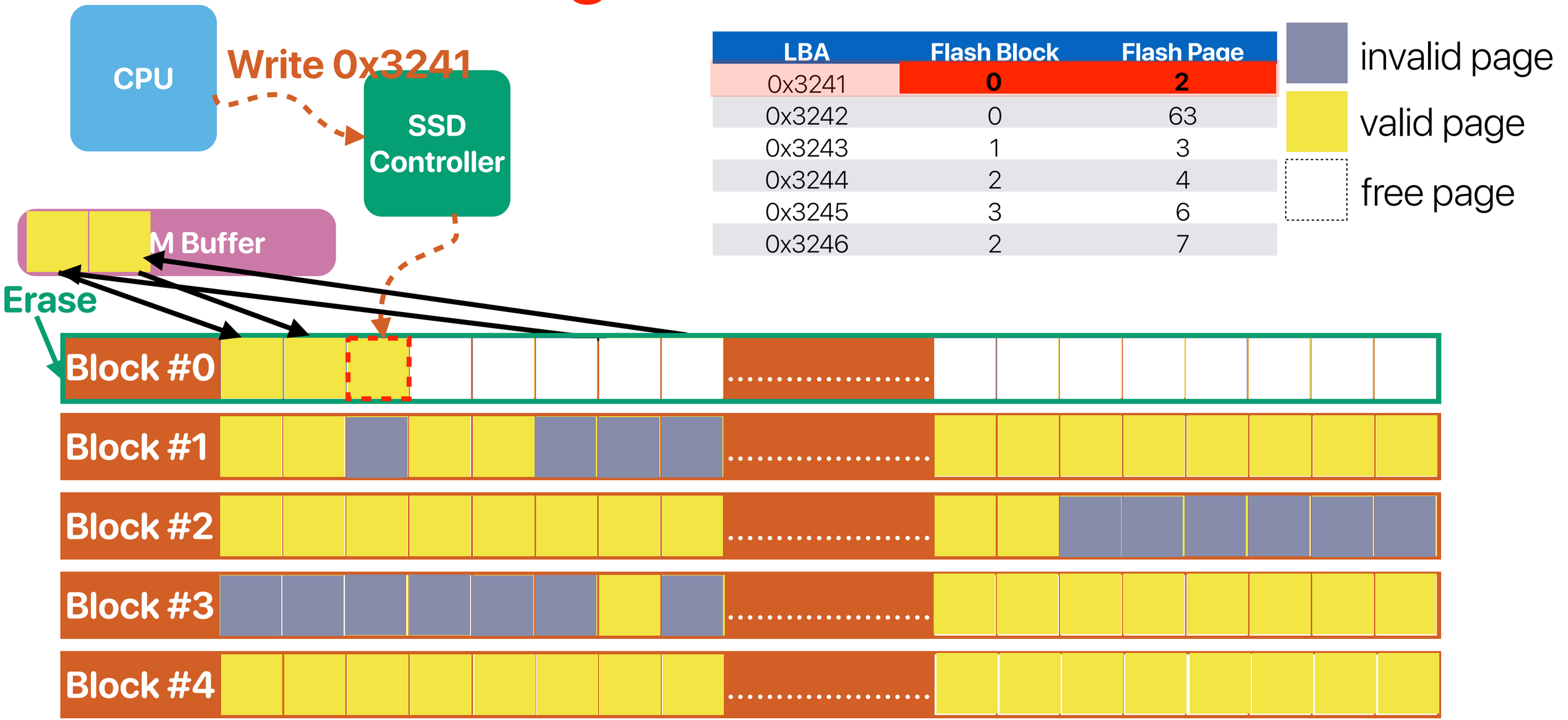
LBA	Flash Block	Flash Page
0x3241	**3**	**7**
0x3242	0	63
0x3243	1	3
0x3244	2	4
0x3245	3	6
0x3246	2	7

invalid page

valid page

free page

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

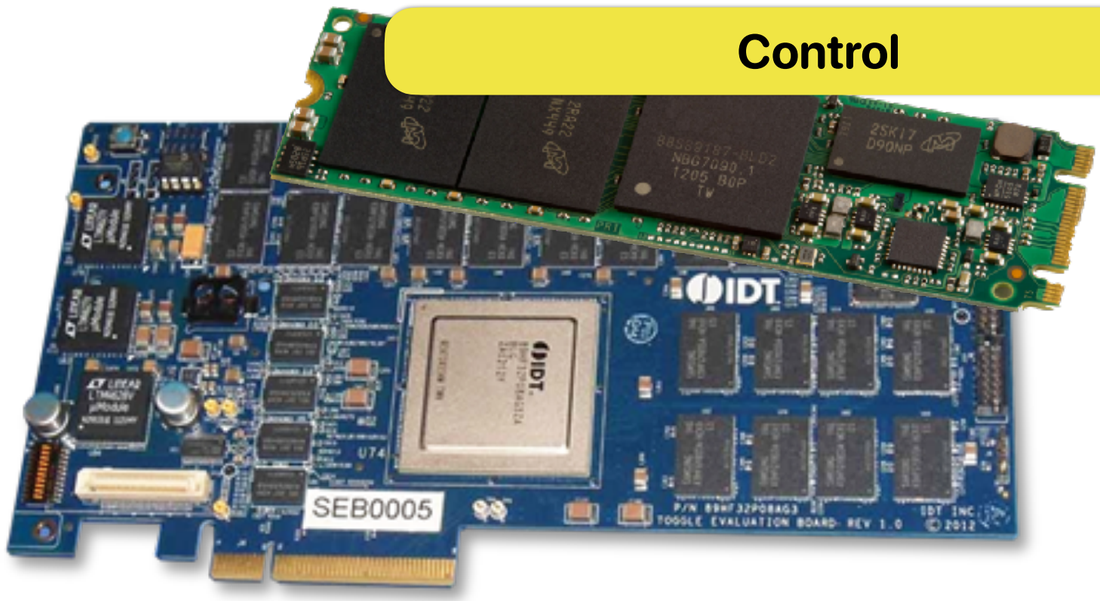
invalid page

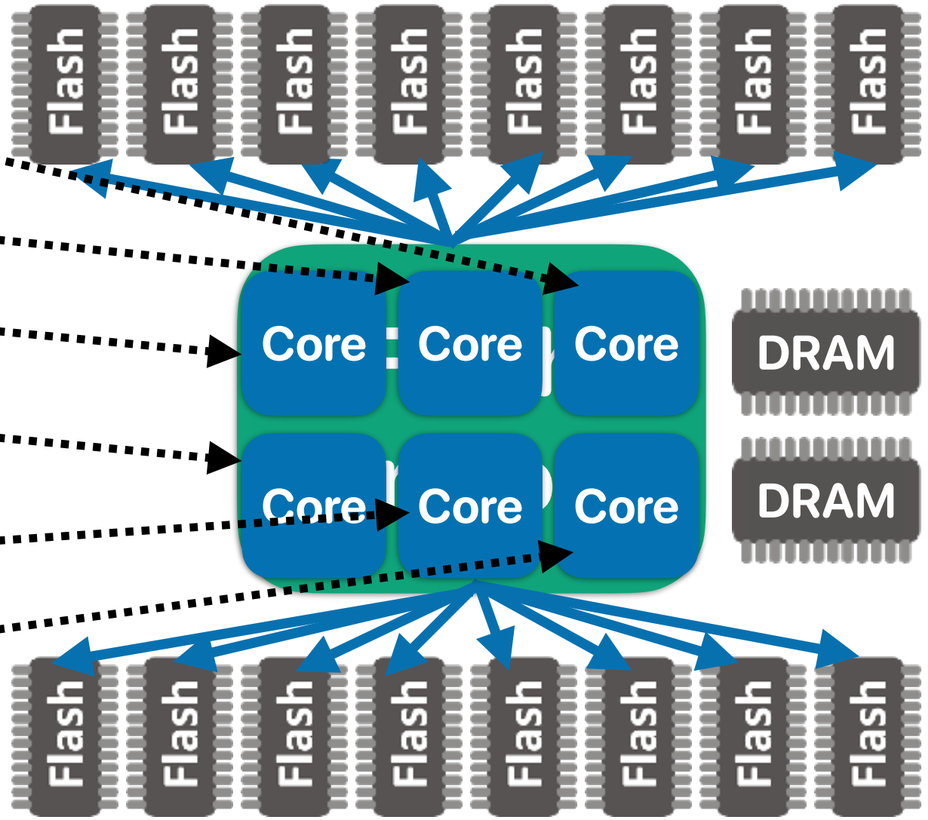
valid page

free page

The tasks of an SSD controller/firmware

- Parse NVMe/SATA/PCIe commands
- Address mapping/lookup
- Wear-leveling/Garbage Collection
- Read
- Program
- Control





Electrical
Computer Science
Engineering

277

つづく