

I/O & Basics of File Systems

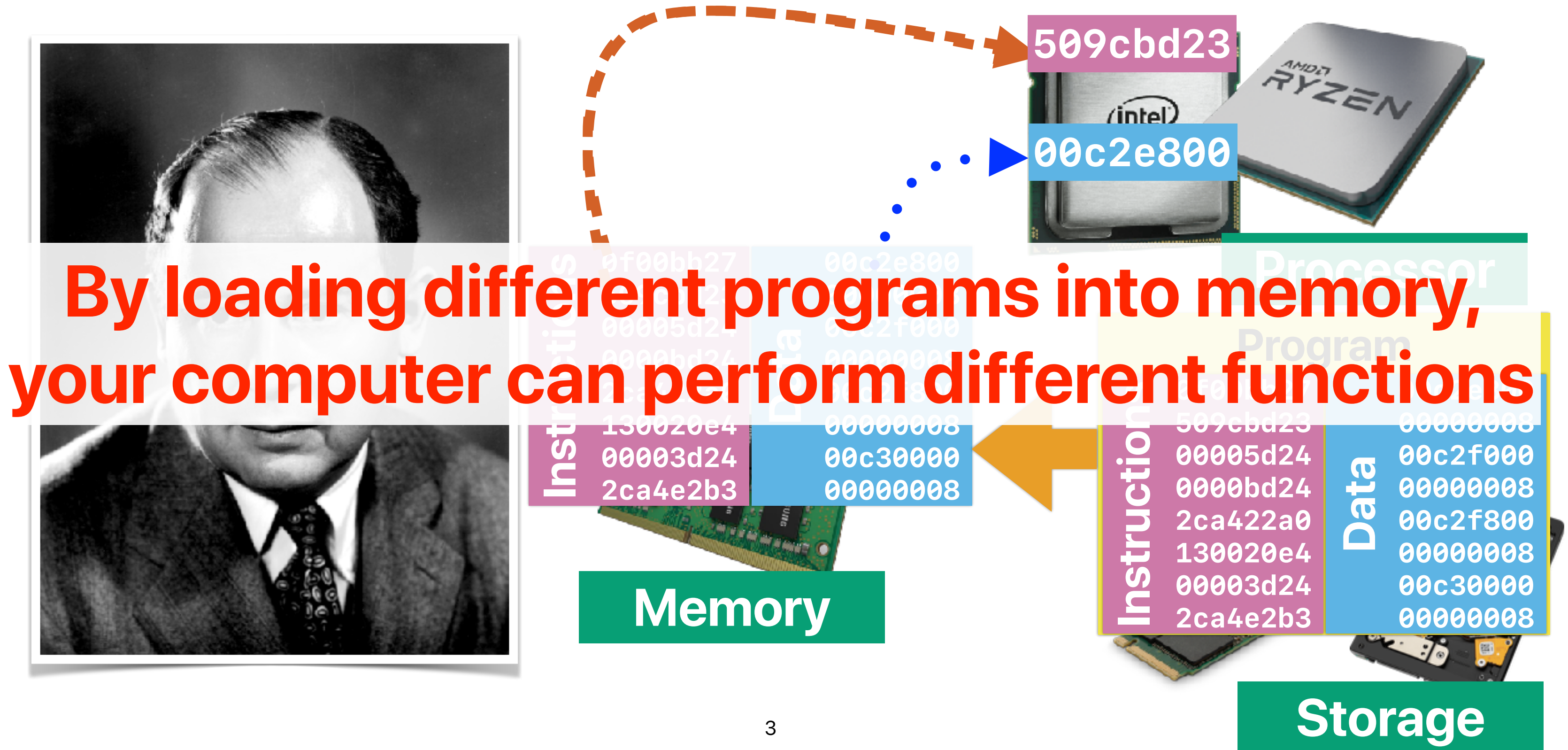
Hung-Wei Tseng

Column: Eric Weddle earned Super Bowl ring the hard way for Rams

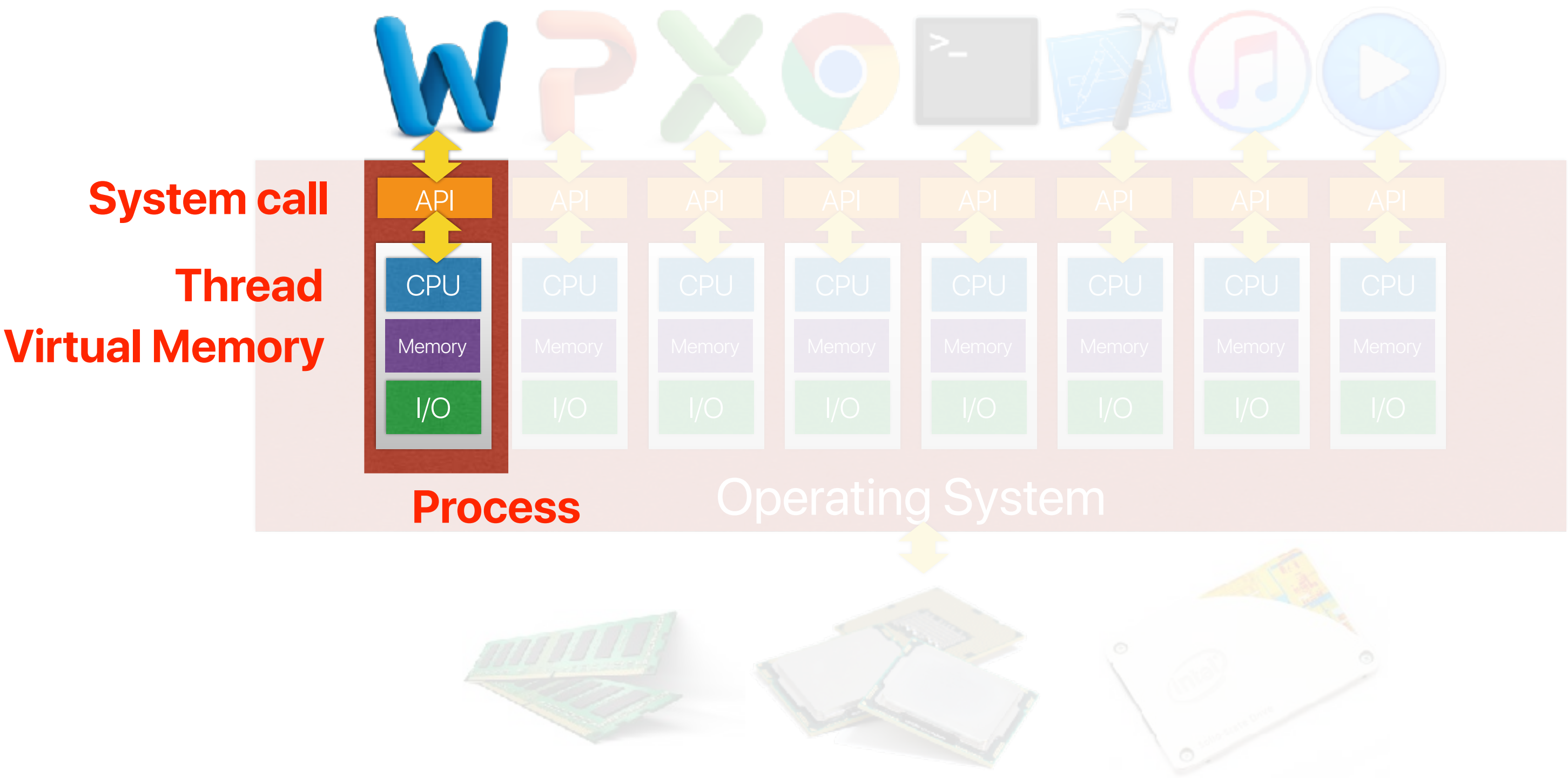


Bengals' Ja'Marr Chase is hit by Rams' Eric Weddle as Troy Reeder (51) moves in at Super Bowl LVI on Sunday at SoFi Stadium. (K.C. Alfred/The San Diego Union-Tribune)

Recap: von Neumann Architecture



Operating Systems — Virtualization, Concurrency, Persistency



Recap: abstractions in operating systems

- Process — the abstraction of a von Neumann machine
- Virtual memory — the abstraction of memory
- Thread — the abstraction of a processor
 - Threads can share virtual memory if they come from the same process
 - You don't have to create another page table when creating a thread

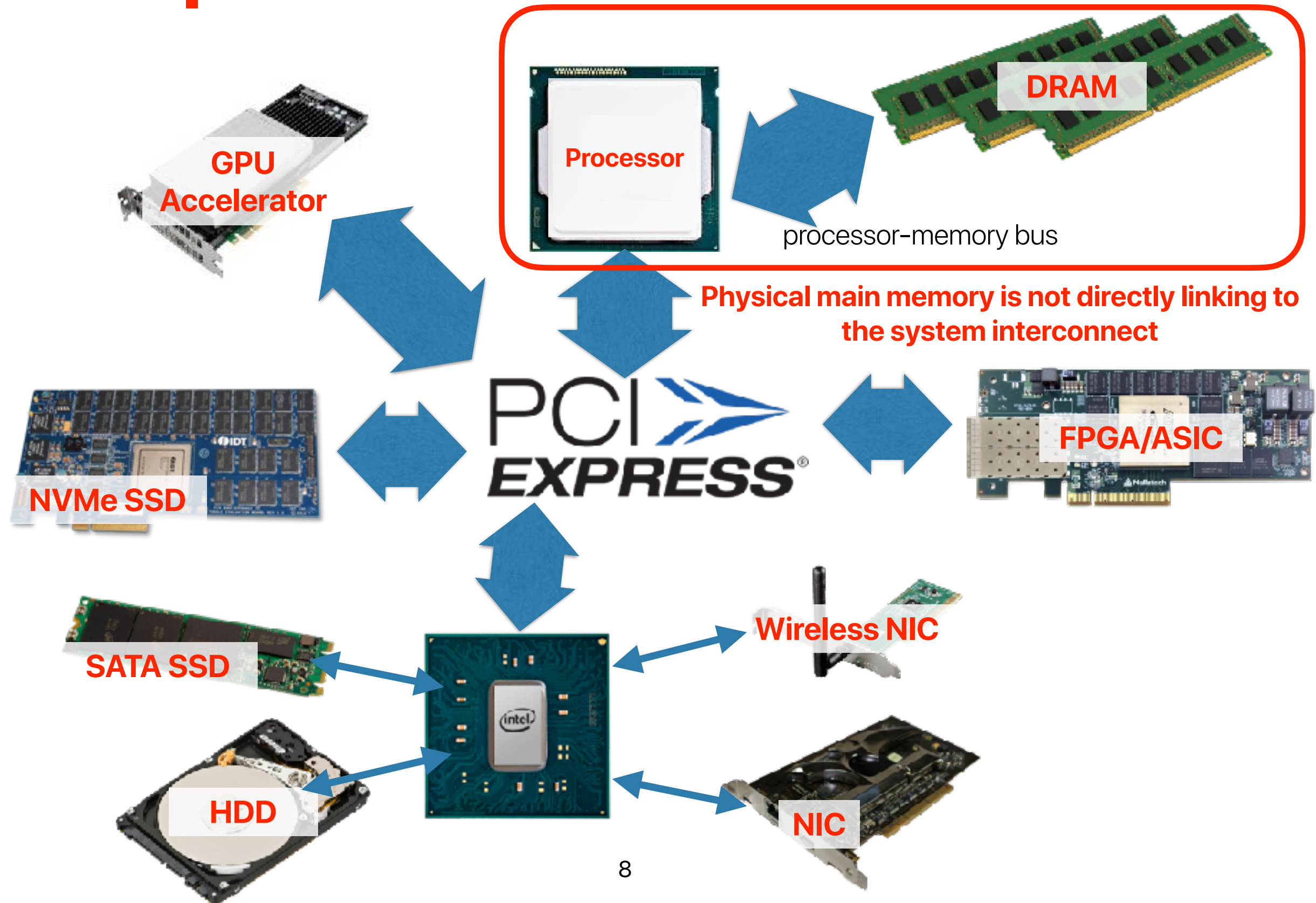
Recap: Virtual memory

- Mechanisms of maintaining the abstraction
 - Segmentation
 - Demanding page + Swapping
 - Hierarchical page table to save space overhead in mapping
 - TLB (translation look-aside buffer) to reduce the translation latency — CS203
- Policies to decide how big the space in the physical main memory each process can enjoy
 - Working set/page local replacement — VMS/UNIX/Mach
 - Global page replacement — Babaoglu's UNIX
- Policies to decide what page to stay in the physical main memory
 - FIFO + freelist — VMS/UNIX/Mach
 - Clock+ freelist — Babaoglu's UNIX
 - WS-Clock — After Carr and Hennessy

Outline

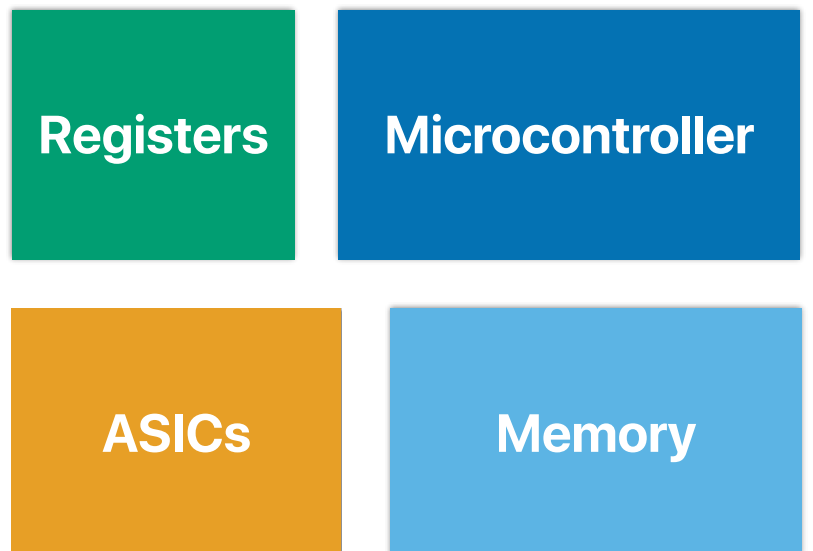
- How our systems interact with I/O
- The basics of storage devices
- File

The computer is now like a small network

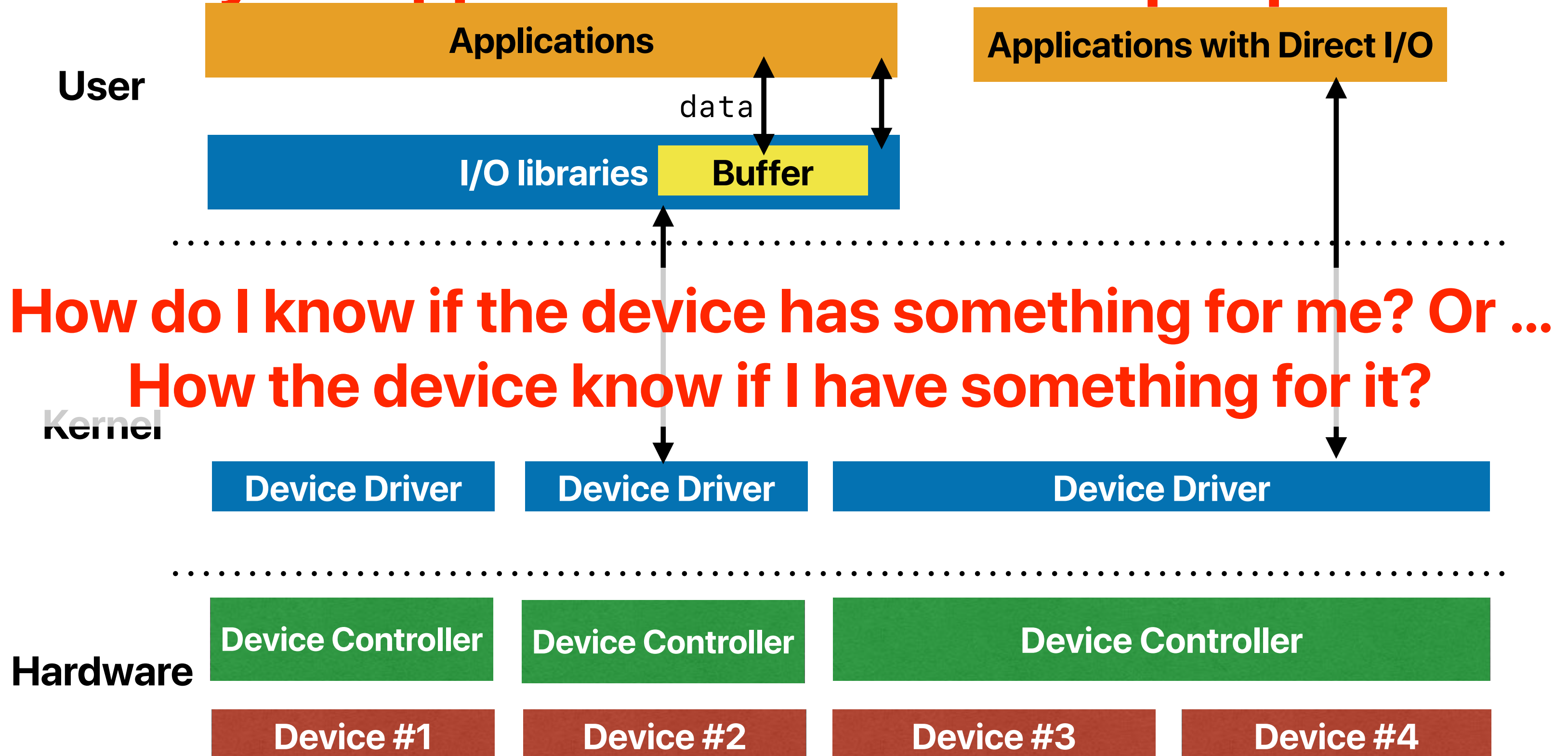


What's in each device?

- Registers
 - Command: receiving commands from host
 - Status: tell the host the status of the device
 - Data: the location of exchanging data
- Microcontroller
- Memory
- ASICs



How your application interact with peripherals



Polling v.s. Interrupt — Round 1

- Comparing polling and interrupt, how many of the following statements are true
 - ① Polling mechanism itself generally consume more CPU time than interrupt
 - ② Interrupt can improve CPU utilization if the device only needs service from the processor occasionally
 - ③ Interrupt allows asynchronous I/O in programs
 - ④ The number of instructions of handling an event after polling is higher than handling the same event after receiving an interrupt
- A. 0
B. 1
C. 2
D. 3
E. 4

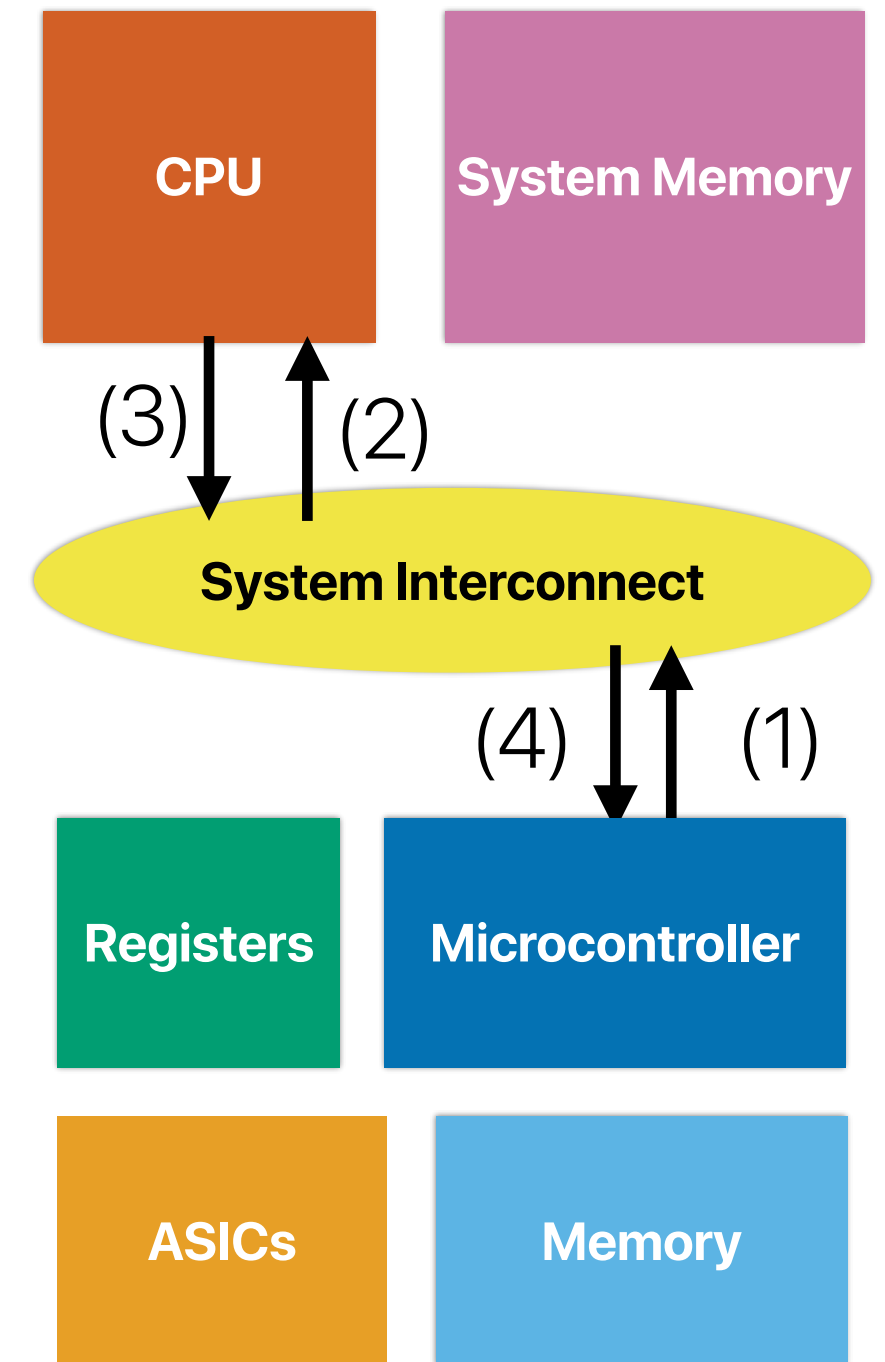
Polling v.s. IRQ	
A	<input type="text"/>
B	<input type="text"/>
C	<input type="text"/>
D	<input type="text"/>
E	<input type="text"/>

Polling v.s. Interrupt — Round 1

- Comparing polling and interrupt, how many of the following statements are true
 - ① ☒ Polling mechanism itself generally consume more CPU time than interrupt
You need to have a loop that periodically polls
 - ② ☒ Interrupt can improve CPU utilization if the device only needs service from the processor occasionally
You can context switch!
 - ③ ☒ Interrupt allows asynchronous I/O in programs
Your function can return immediately
 - ④ ☒ The number of instructions of handling an event after polling is higher than handling the same event after receiving an interrupt
Not related to polling/interrupt
- A. 0
B. 1
C. 2
D. 3
E. 4

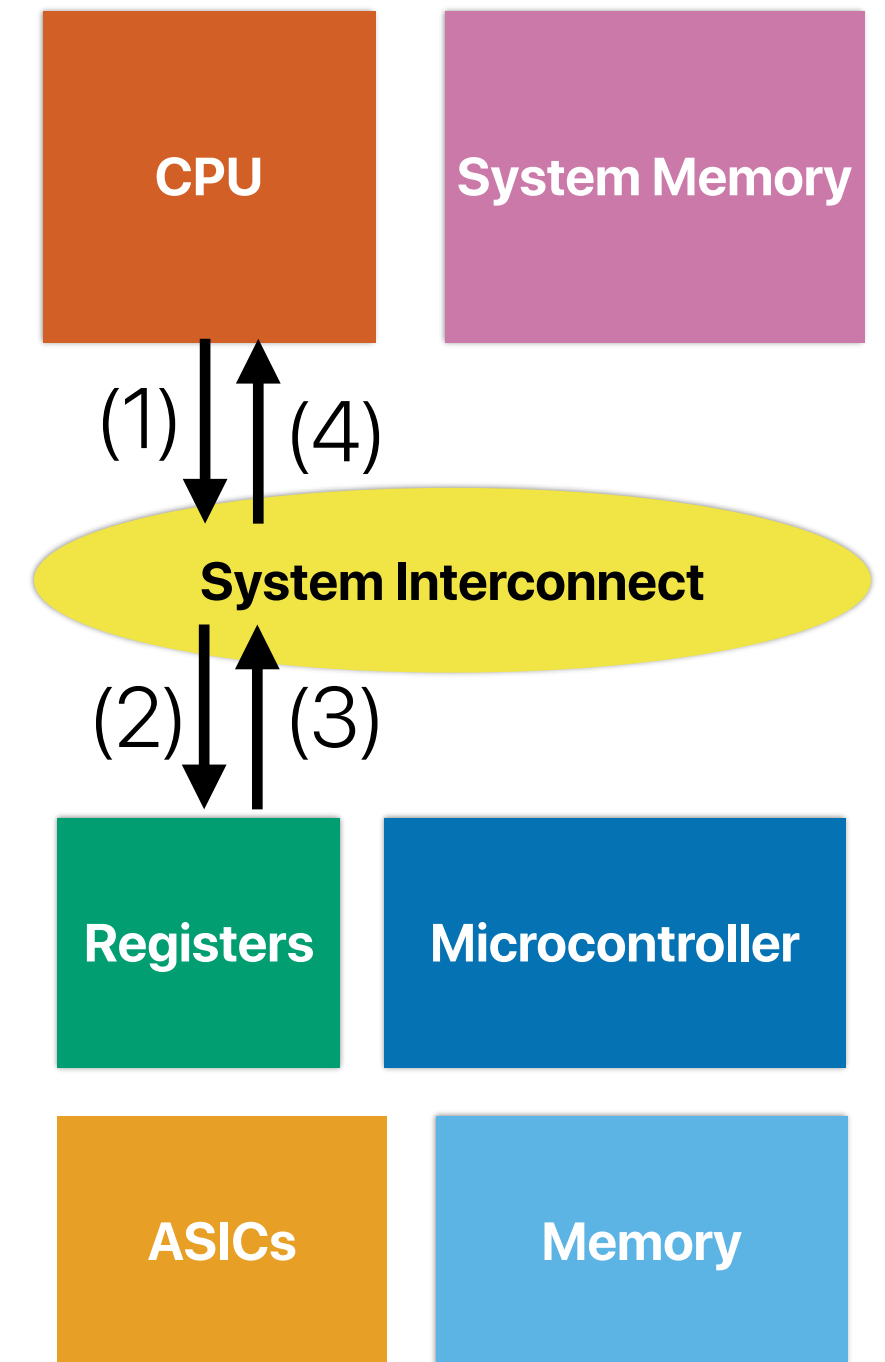
Interrupt

- The device signals the processor only when the device requires the processor/OS handle some tasks/data
- The processor only signals the device when necessary



Polling

- The processor/OS constantly asks if the device (e.g. examine the status register of the device) is ready to or requires the processor/OS handle some tasks/data
- The OS/processor executes corresponding handler if the device can handle demand tasks/data or has tasks/data ready



Interrupt v.s. Polling — Round 2

- Regarding using interrupts and polling for communicating peripheral devices, how many of the followings is/are correct?
 - Using interrupts may increase the end-to-end latency for a process comparing with polling
 - Using interrupts may increase the cache miss rates comparing with polling
 - Using interrupts for high-speed storage devices may decrease the power consumption of the processor
 - The latency of serving I/O requests using interrupts can be longer than using polling if context switches occur during the I/O

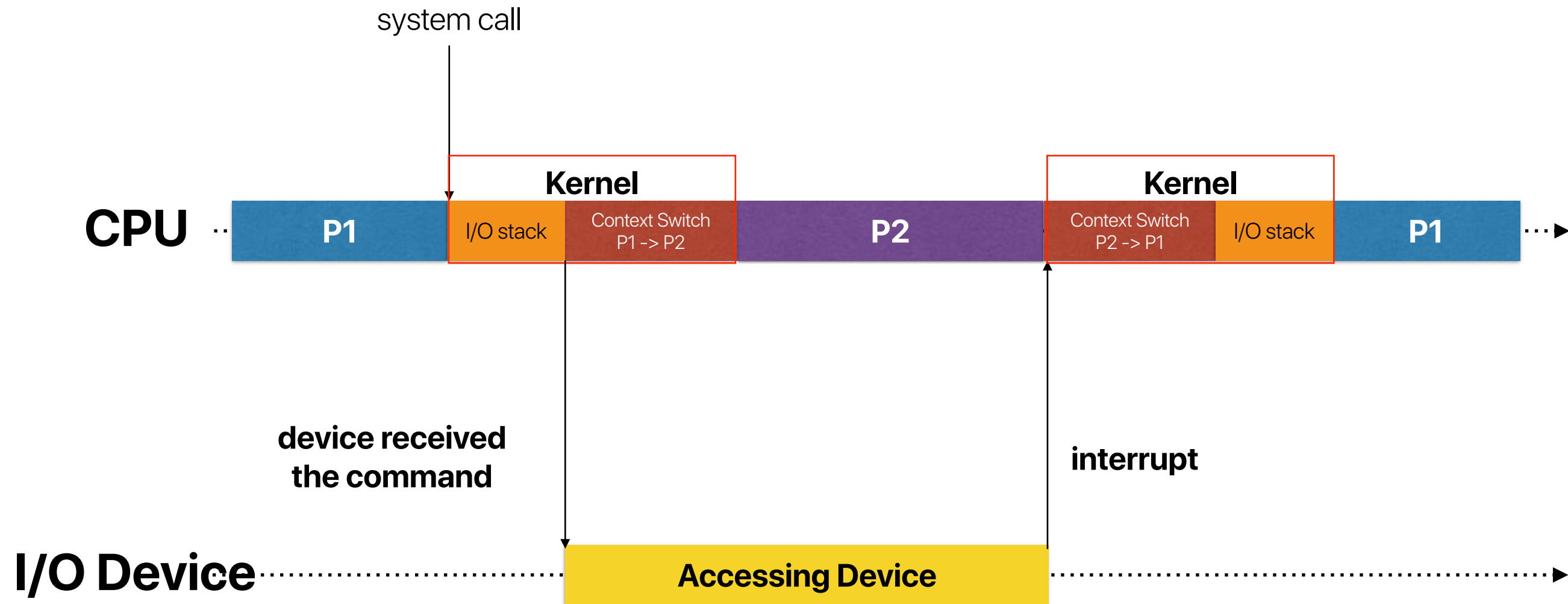
- A. 0
- B. 1
- C. 2
- D. 3
- E. 4

Polling v.s. IRQ (2)	
A	
B	
C	
D	
E	

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.

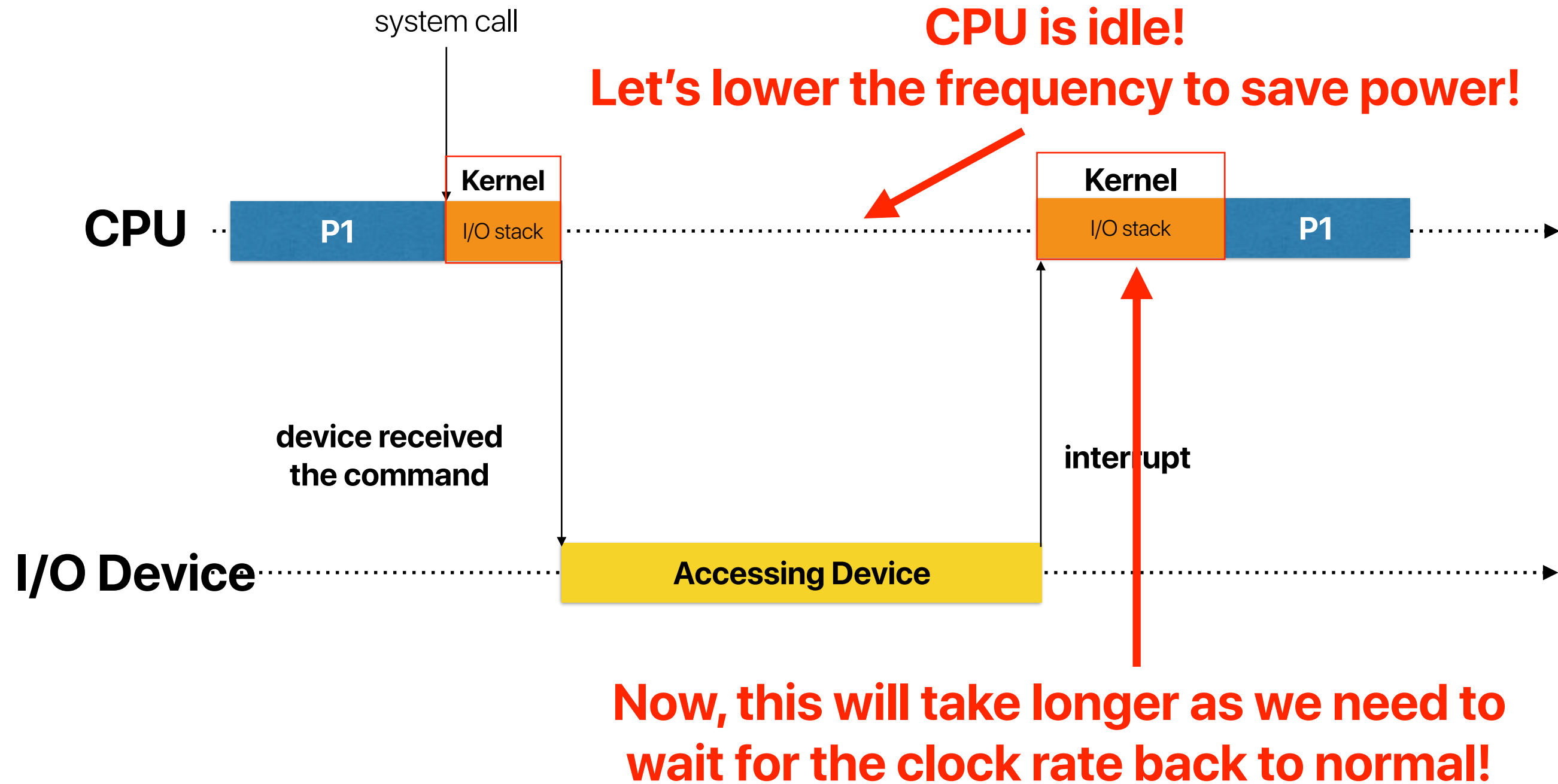


If $T_{\text{Context switch P1} \rightarrow \text{P2}} + T_{\text{Context switch P2} \rightarrow \text{P1}} < T_{\text{Accessing peripherals}}$
makes sense to context switch

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?



Interrupt v.s. Polling — Round 2

- Regarding using interrupts and polling for communicating peripheral devices, how many of the followings is/are correct?
 - ① Using interrupts may increase the end-to-end latency for a process comparing with polling
Because you context switched!
 - ② Using interrupts may increase the cache miss rates comparing with polling
Because you context switched!
 - ③ Using interrupts for high-speed storage devices may decrease the power consumption of the processor
True, because your processor is free and may be idle — allowing DVFS to lower the clock rate
 - ④ The latency of serving I/O requests using interrupts can be longer than using polling if context switches occur during the I/O
Yes. It's still because you have to switch back and warm up cache

A. 0

B. 1

C. 2

D. 3

E. 4

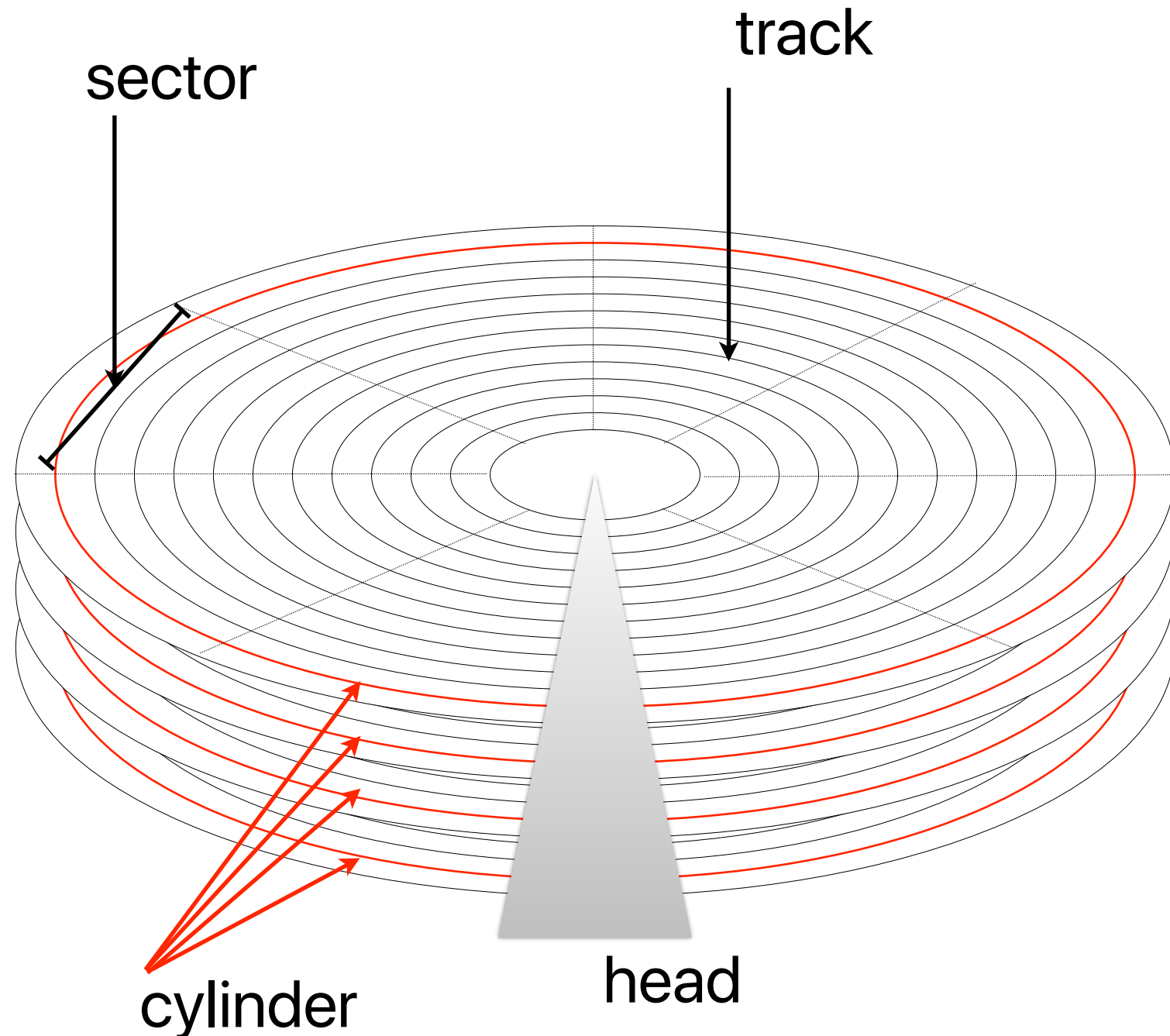
When should we poll? When should we interrupt

- 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)

Case study: interacting with 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

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

$$\begin{aligned} & 8 \text{ ms} + \frac{1}{2} \times \frac{1}{\frac{7200}{60}} + \frac{\frac{0.5}{1024}}{300} + 0.2 \text{ ms} \\ & = 8 \text{ ms} + 4.17 \text{ ms} + 0.00167 \text{ us} + 0.2 \text{ ms} = 12.36 \text{ ms} \end{aligned}$$

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

$$\begin{aligned} & 8 \text{ ms} + \frac{1}{2} \times \frac{1}{\frac{7200}{60}} + \frac{4}{300} + 0.2 \text{ ms} \\ & = 8 \text{ ms} + 4.17 \text{ ms} + 13.33 \text{ ms} + 0.2 \text{ ms} = 25.69 \text{ ms} \end{aligned}$$

Bandwidth = volume_of_data over period_of_time

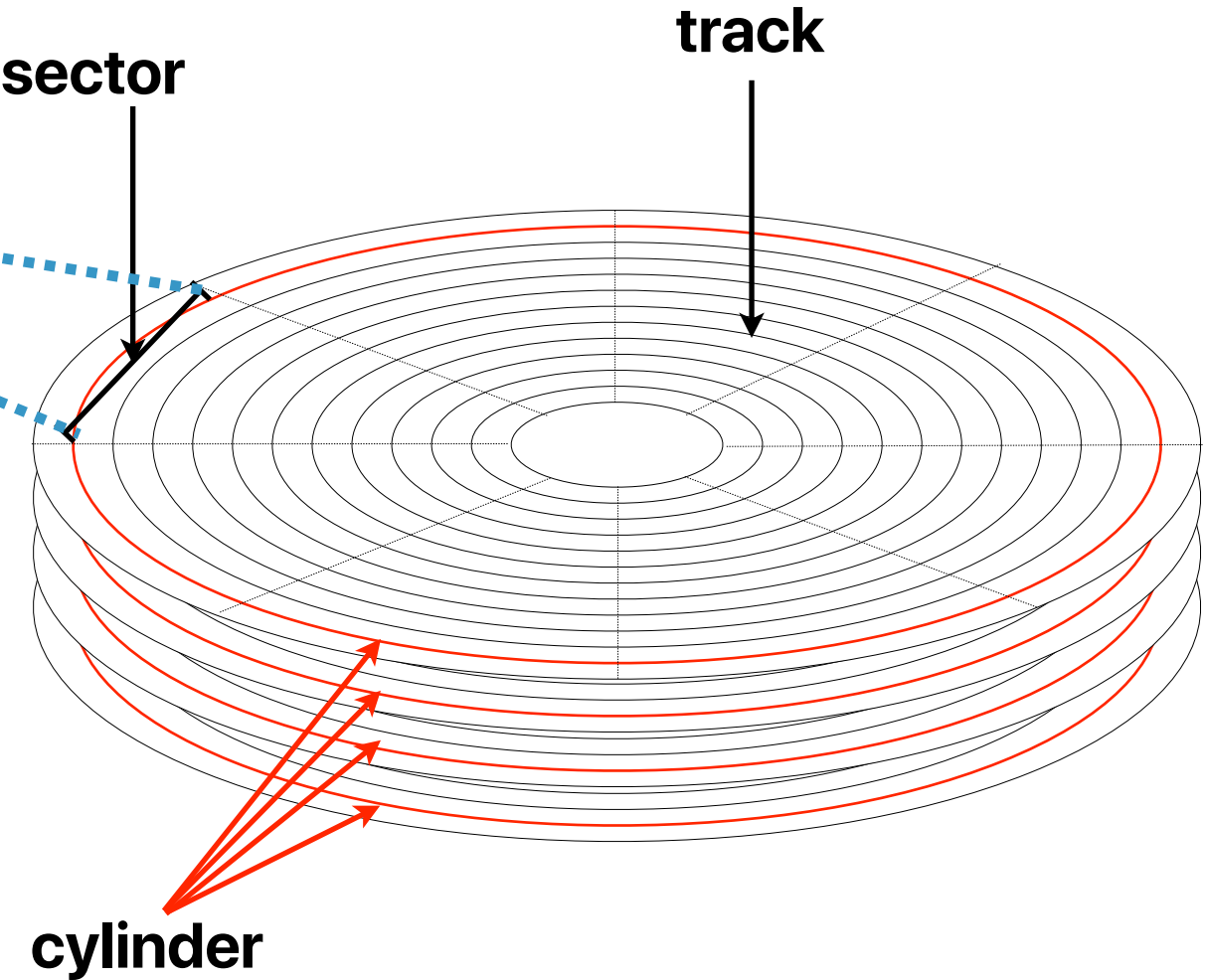
$$= \frac{4MB}{25.69ms} = 155.7 \text{ MB/sec} \quad \text{Trading latencies with bandwidth}$$

Numbering the disk space with block addresses

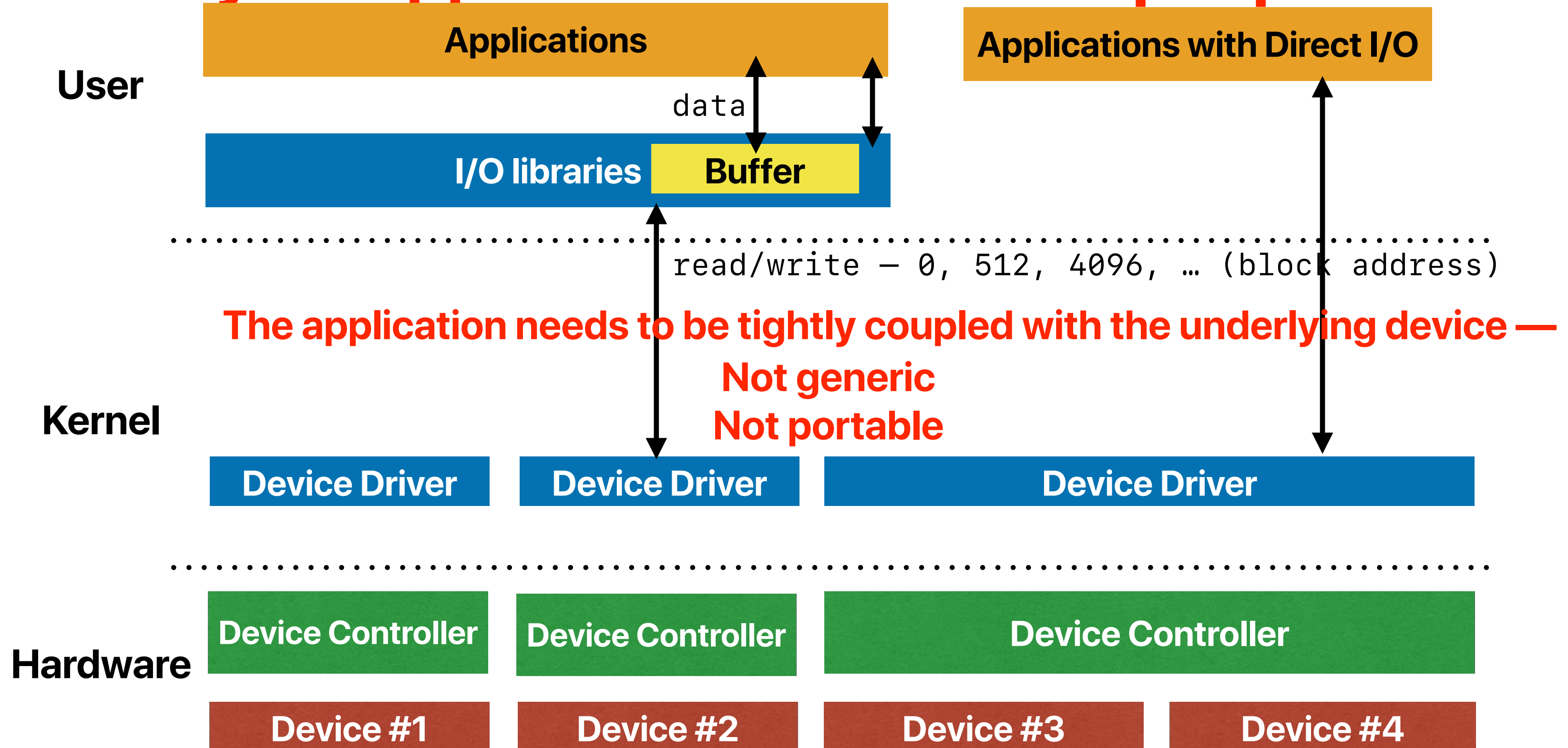
Disk blocks

0								7
8								15
16								23
24								31
32								39
40								47
48								55
56								63

...



How your application interact with peripherals



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

–David Wheeler

The file & file system abstraction

File abstraction in UNIX

- Regarding “files” in the “basic” UNIX operating system, how many of the following statements is/are correct?
 - ① Every device can be mapped to a file
 - ② The UNIX file system uses a hierarchical structure and directory is also a file in UNIX
 - ③ The UNIX file system runs in the kernel space
 - ④ The UNIX file system needs to maintain the information regarding the content type of files (e.g. image, text, C program)

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

Files in UNIX	
A	
B	
C	
D	
E	

What we've learned in the past...

The most important role of UNIX is to provide a file system. From the point of view of the user, there are three kinds of files: ordinary disk files, directories, and special files.

3.1 Ordinary Files

A file contains whatever information the user places on it, for example symbolic or binary (object) programs. No particular structuring is expected by the system. Files of text consist simply of a string of characters, with lines demarcated by the new-line character. Binary programs are sequences of words as they will appear in core memory when the program starts executing. A few user programs manipulate files with more structure: the assembler generates and the loader expects an object file in a particular format. However, the structure of files is controlled by the programs which use them, not by the system.

3.2 Directories

Directories provide the mapping between the names of files and the files themselves, and thus induce a structure on the file system as a whole. Each user has a

directory of his own files; he may also create subdirectories to contain groups of files conveniently treated together. A directory behaves exactly like an ordinary file except that it cannot be written on by unprivileged programs, so that the system controls the contents of directories. However, anyone with appropriate permission may read a directory just like any other file.


3.3 Special Files

Special files constitute the most unusual feature of the UNIX file system. Each I/O device supported by UNIX is associated with at least one such file. Special files are read and written just like ordinary disk files, but requests to read or write result in activation of the associated device. An entry for each special file resides in directory */dev*, although a link may be made to one of these files just like an ordinary file. Thus, for example, to punch paper tape, one may write on the file */dev/ppt*. Special files exist for each communication line, each disk, each tape drive, and for physical core memory. Of course, the active disks and the core special file are protected from indiscriminate access.

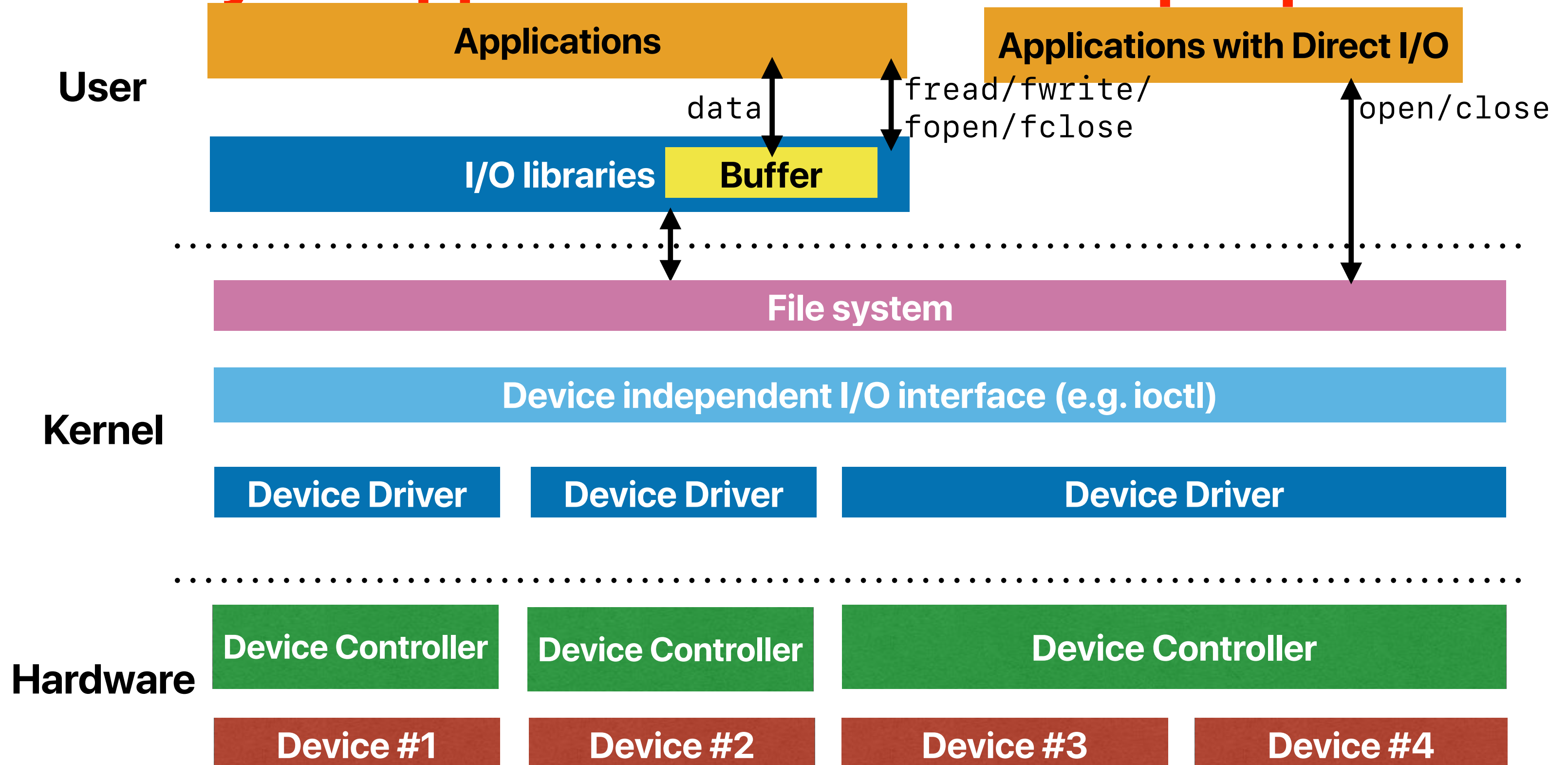
There is a threefold advantage in treating I/O devices this way: file and device I/O are as similar as possible; file and device names have the same syntax and mean-

expecting a file name as a parameter; device name; finally, special files protection mechanism as regular

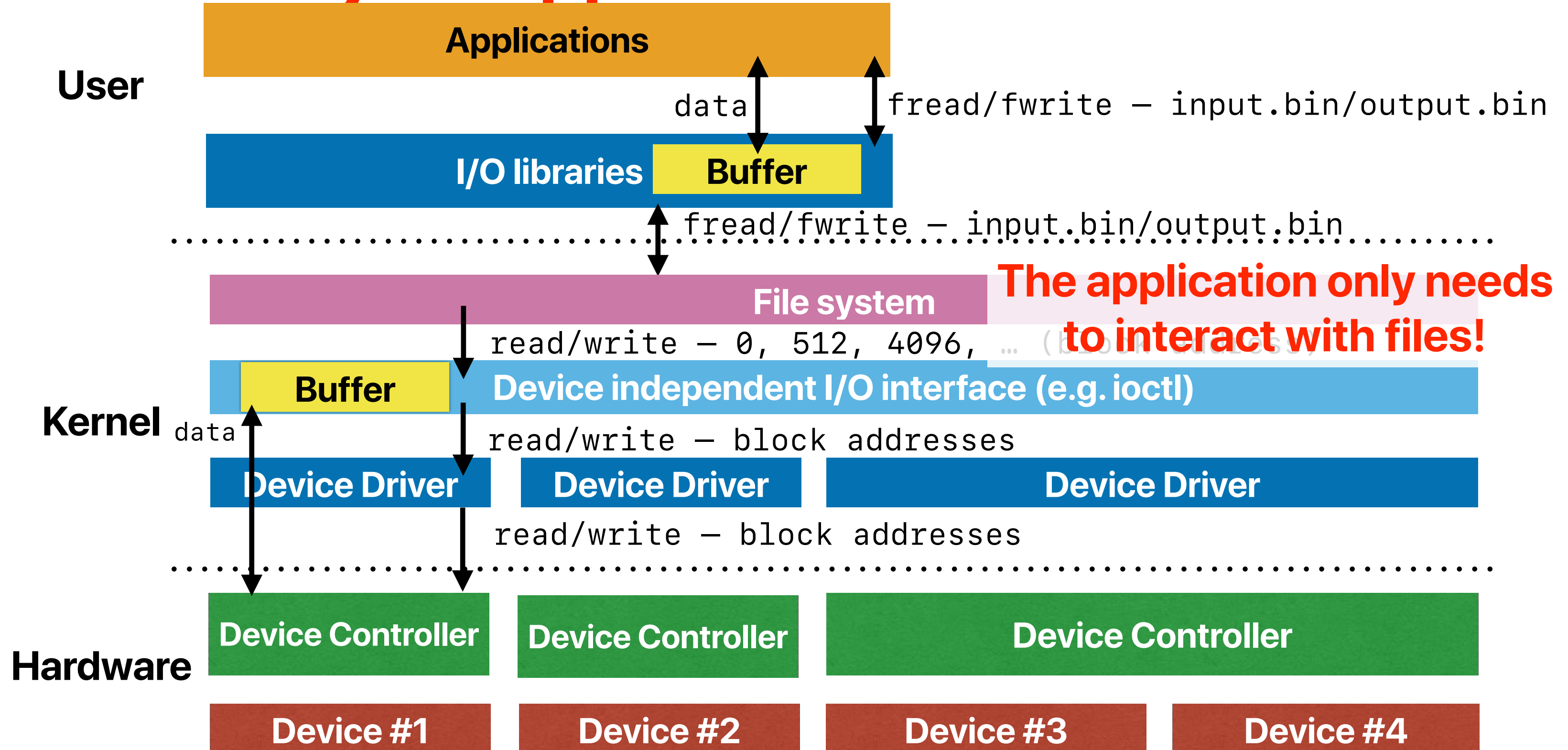
File abstraction in UNIX

- Regarding “files” in the “basic” UNIX operating system, how many of the following statements is/are correct?
 - ① Every device can be mapped to a file
 - ② The UNIX file system uses a hierarchical structure and directory is also a file in UNIX
 - ③ The UNIX file system runs in the kernel space
 -  ④ The UNIX file system needs to maintain the information regarding the content type of files (e.g. image, text, C program)
- A. 0
B. 1
C. 2
D. 3
E. 4

How your application interact with peripherals



How your application reaches H.D.D.



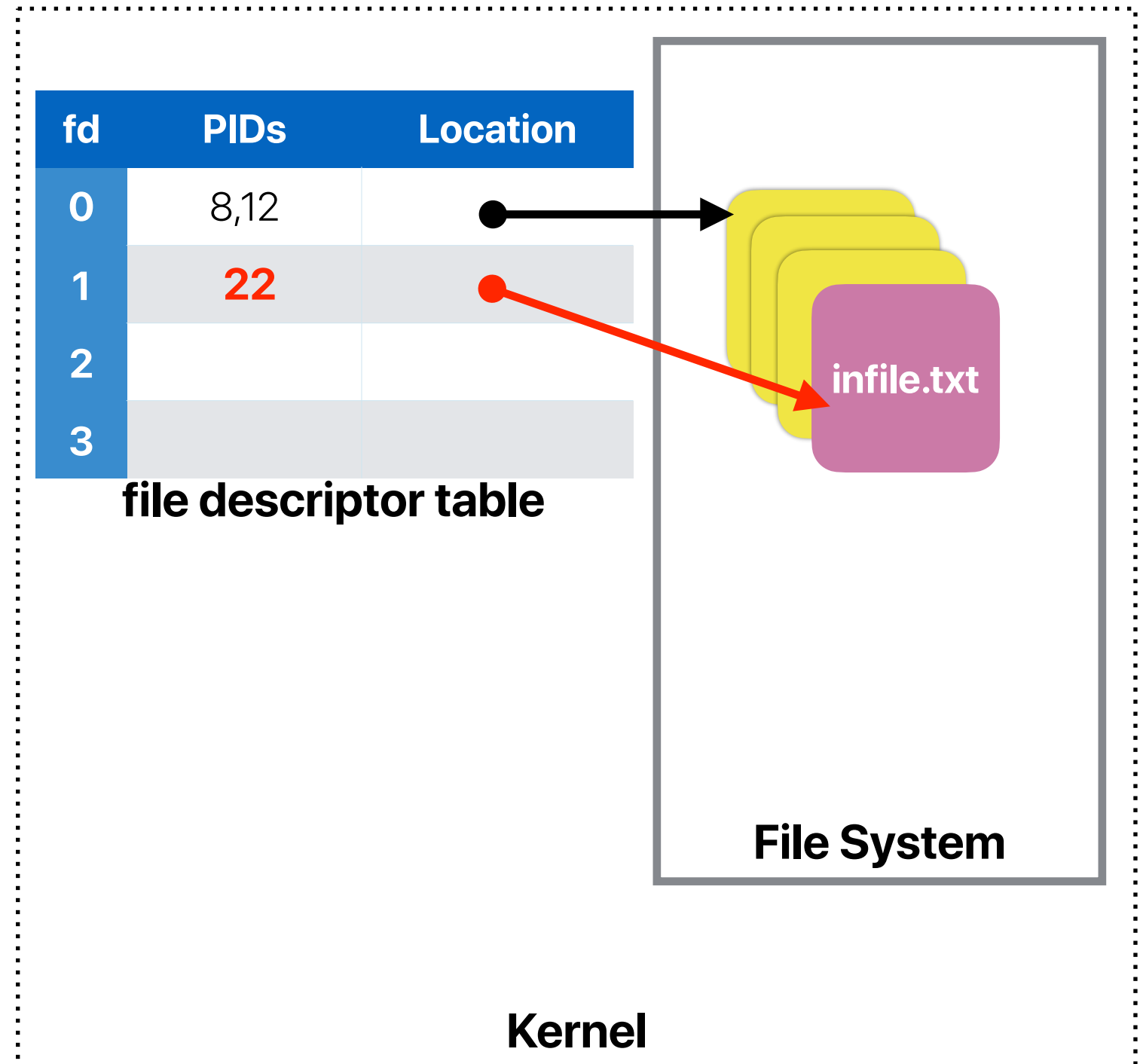
How you access files in C

```
int fd, nr, nw;
void *in_buff;
in_buff = malloc(BUFF_SIZE);

fd1 = open("infile.txt", O_RDONLY);
fd2 = open("outfile.txt", O_RDWR | O_CREAT);
nr = read(fd1, in_buff, BUFF_SIZE);
nw = write(fd2, in_buff, BUFF_SIZE);
lseek(fd1, -8, SEEK_END);
nr = read(fd1, in_buff, 8); // read last 8 bytes
// more fancy stuff here...
close(fd1);
close(fd2);
```


open

¹
fd = open("infile.txt");



Why open system call?

- Why do we have an open call, as opposed to specifying the file path for read and write?
 - A. To check file permissions
 - B. To improve performance
 - C. To lookup the file's location on disk
 - D. Exactly two of the above
 - E. All of the above (i.e. A, B, and C)

Open	
A	<input type="text"/>
B	<input type="text"/>
C	<input type="text"/>
D	<input type="text"/>
E	<input type="text"/>

Why open system call?

- Why do we have an open call, as opposed to specifying the file path for read and write?
 - A. To check file permissions
 - B. To improve performance
 - C. To lookup the file's location on disk
 - D. Exactly two of the above
 - E. All of the above (i.e. A, B, and C)

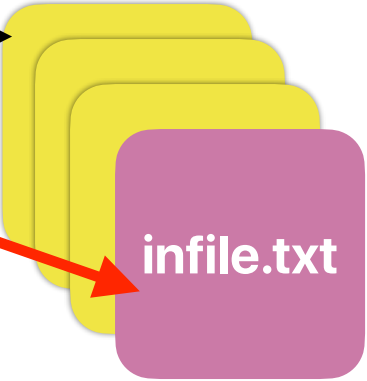
read

```
read(1fd, buff, n);
```



fd	PIDs	Location
0	8,12	
1	22	
2		
3		

file descriptor table

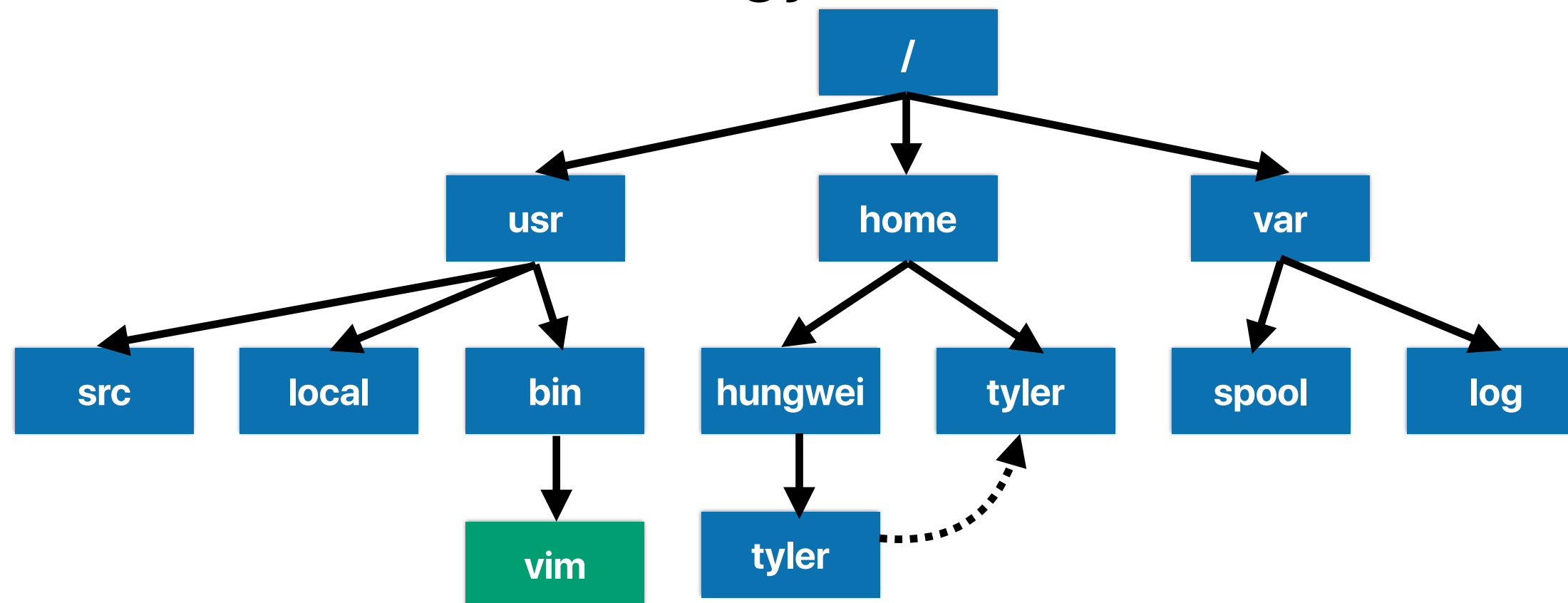


File System

Kernel

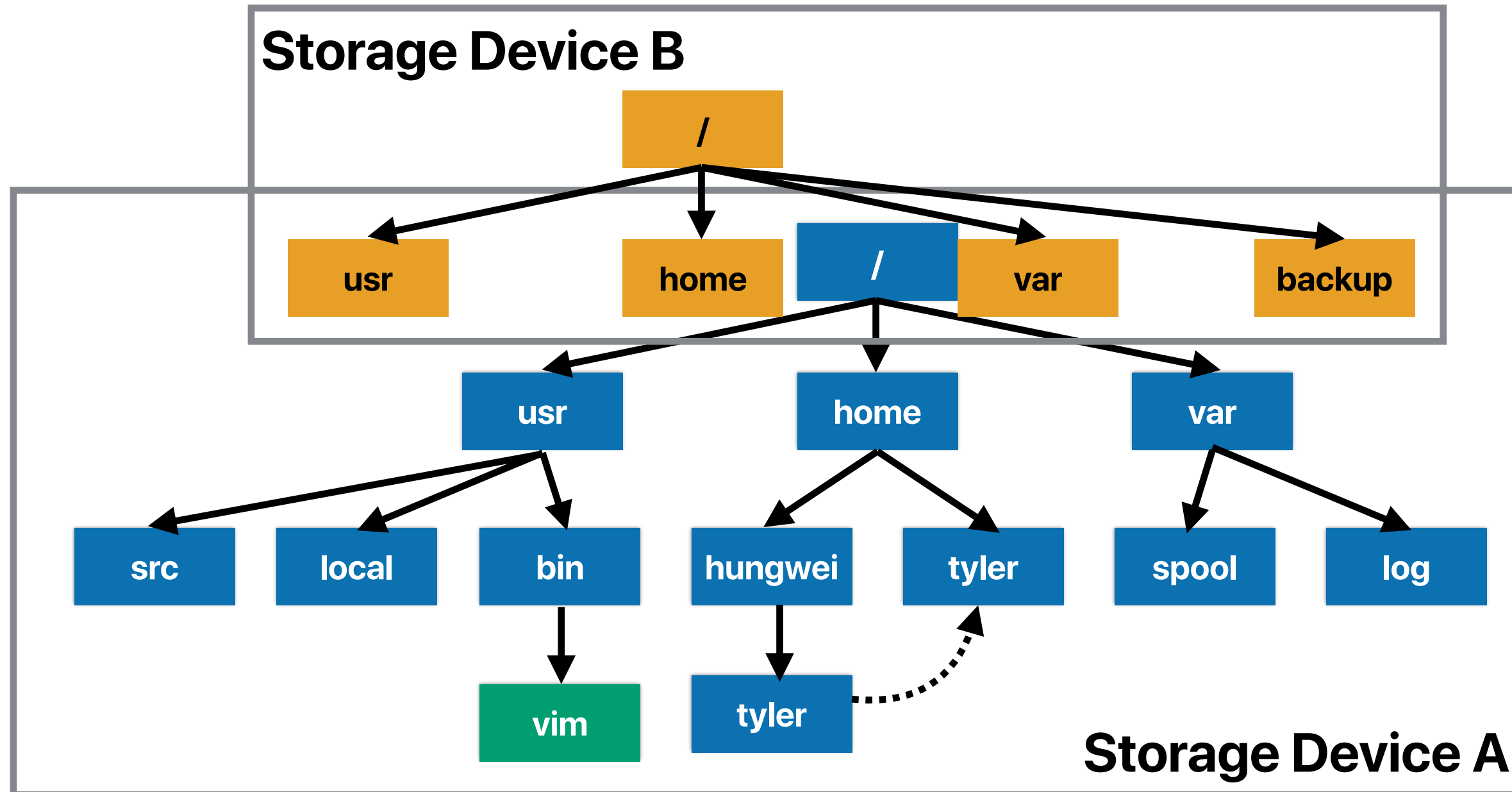
Hierarchical File System Structure

- Namespace has tree-like structure
- Root directory (/) with subdirectories, each containing its own subdirectories
- Links break the tree analogy



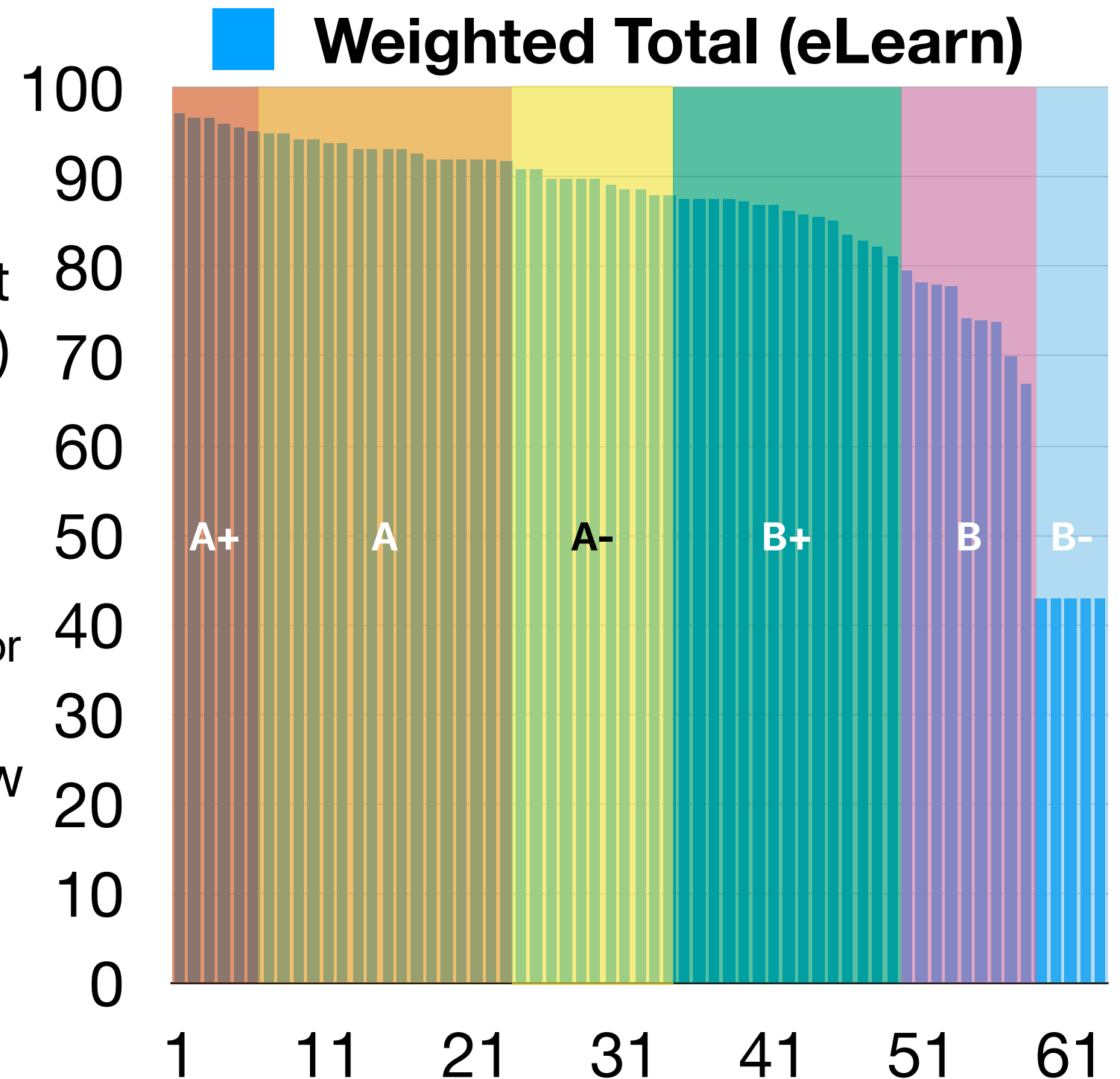
Mount

- The "/" on storage device A will become /backup now!



Midterm

- Average: ~~82~~**72**
- You may find your grade and reference solution in eLearn
- One week regrading policy, no regrading request after 2/22 (we finished the grading this morning)
 - Check the website for the protocol of regrading
 - We will regrade your **whole** exam
 - It's published on the webpage since 0th week.
 - pity points will be off (e.g., we currently give points for missing ; and mis-spelling)
- It's only 20% — more important to figure out how much you learned rather than minor points.
- The only case will fail your CS202 is cheating or being absent in final — we identified 5 cases.



Project

- Project
 - What's the high-level idea of the project?
 - Challenges?
 - What's your high-level system architecture
 - What are the system components: data structures? additional facilities?
 - What are the communication, data exchange, signaling mechanisms and scheduling policies
 - How to accomplish the goal?
 - Due 3/3 — no late submission is allowed

Announcement

- Reading quizzes due next Tuesday
- Office hour
 - Check Google Calendar
 - Use the office hour Zoom link, not the lecture one

Computer Science & Engineering

202

つづく

