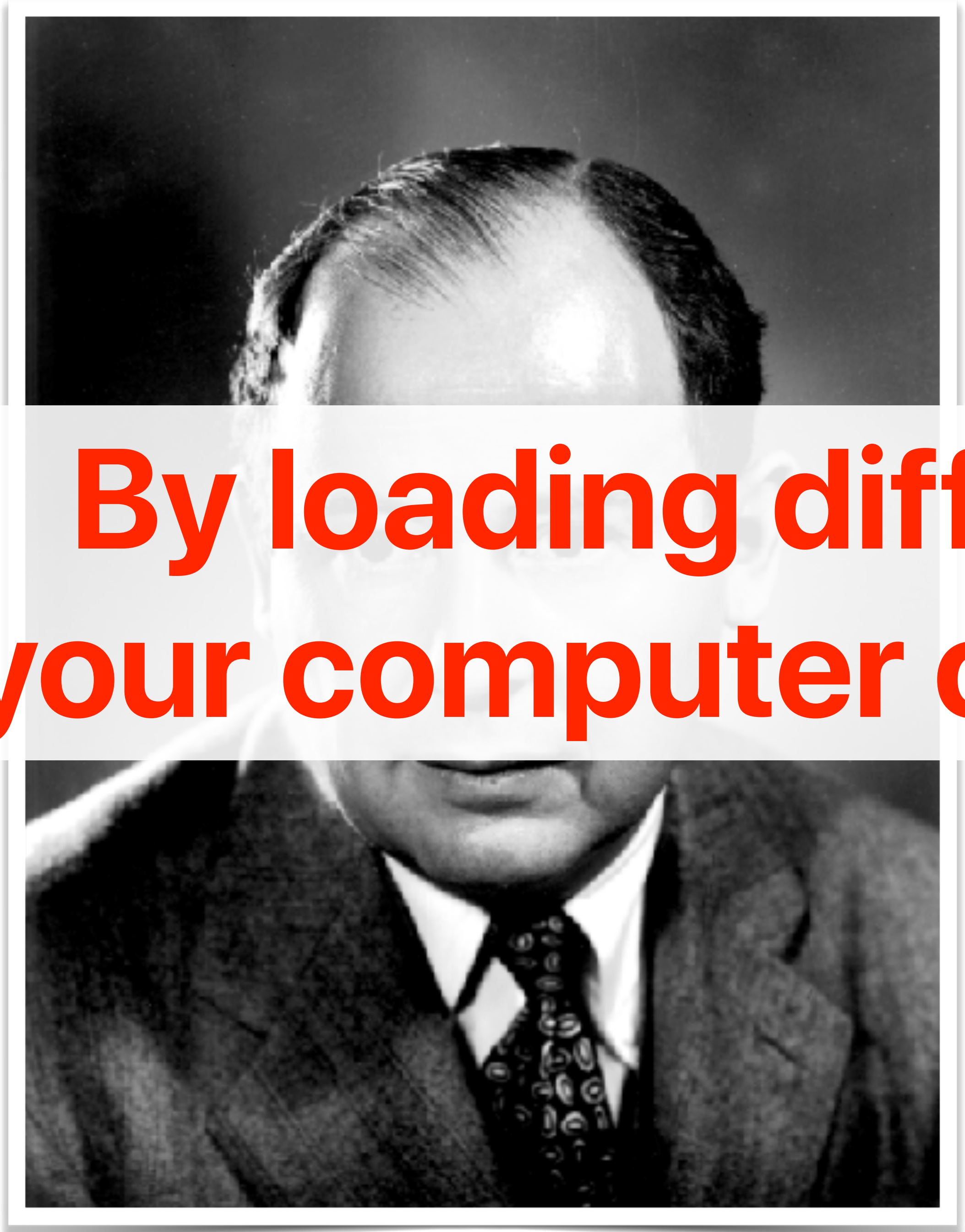
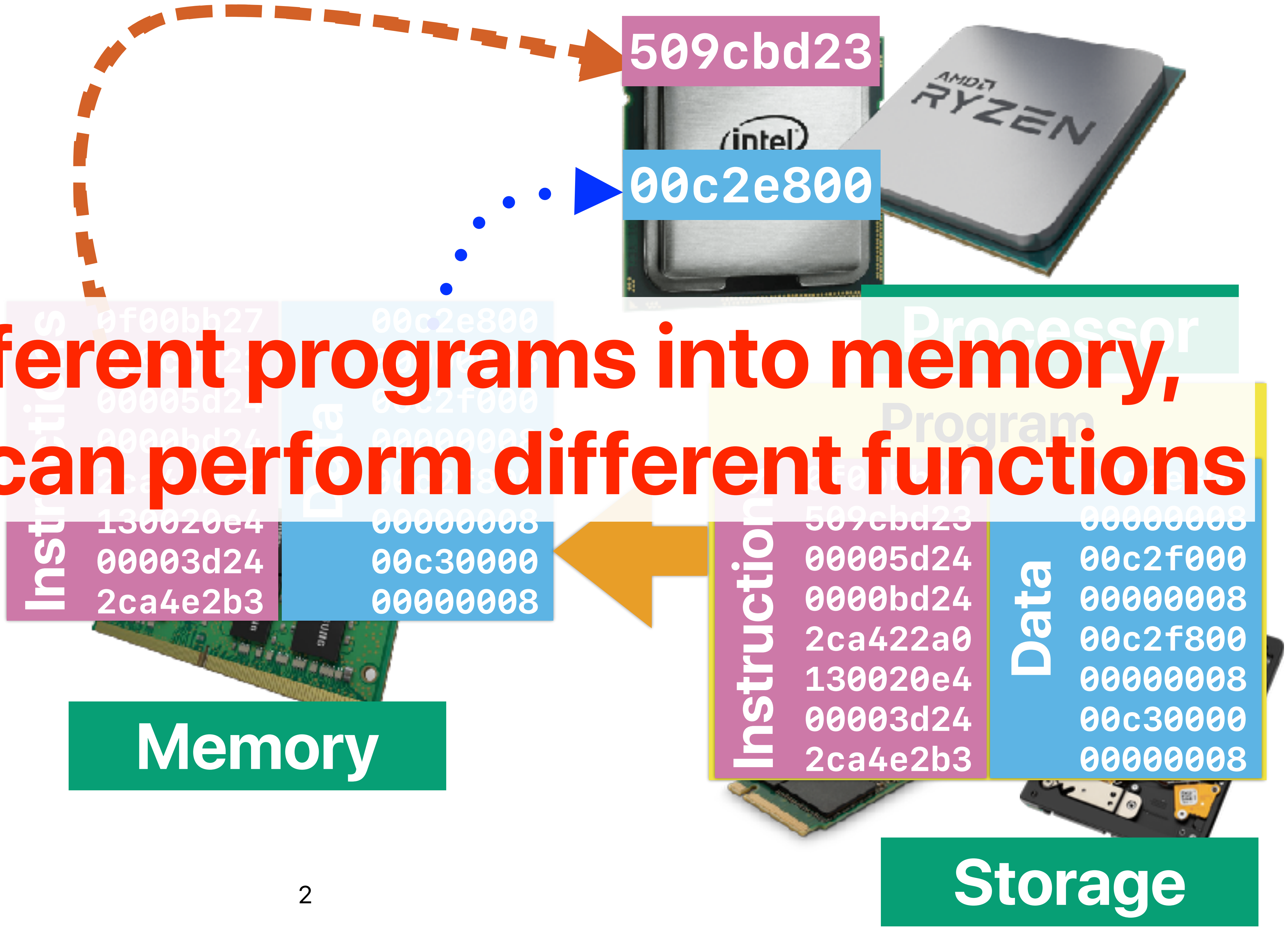
I/O & Basics of File Systems

Hung-Wei Tseng

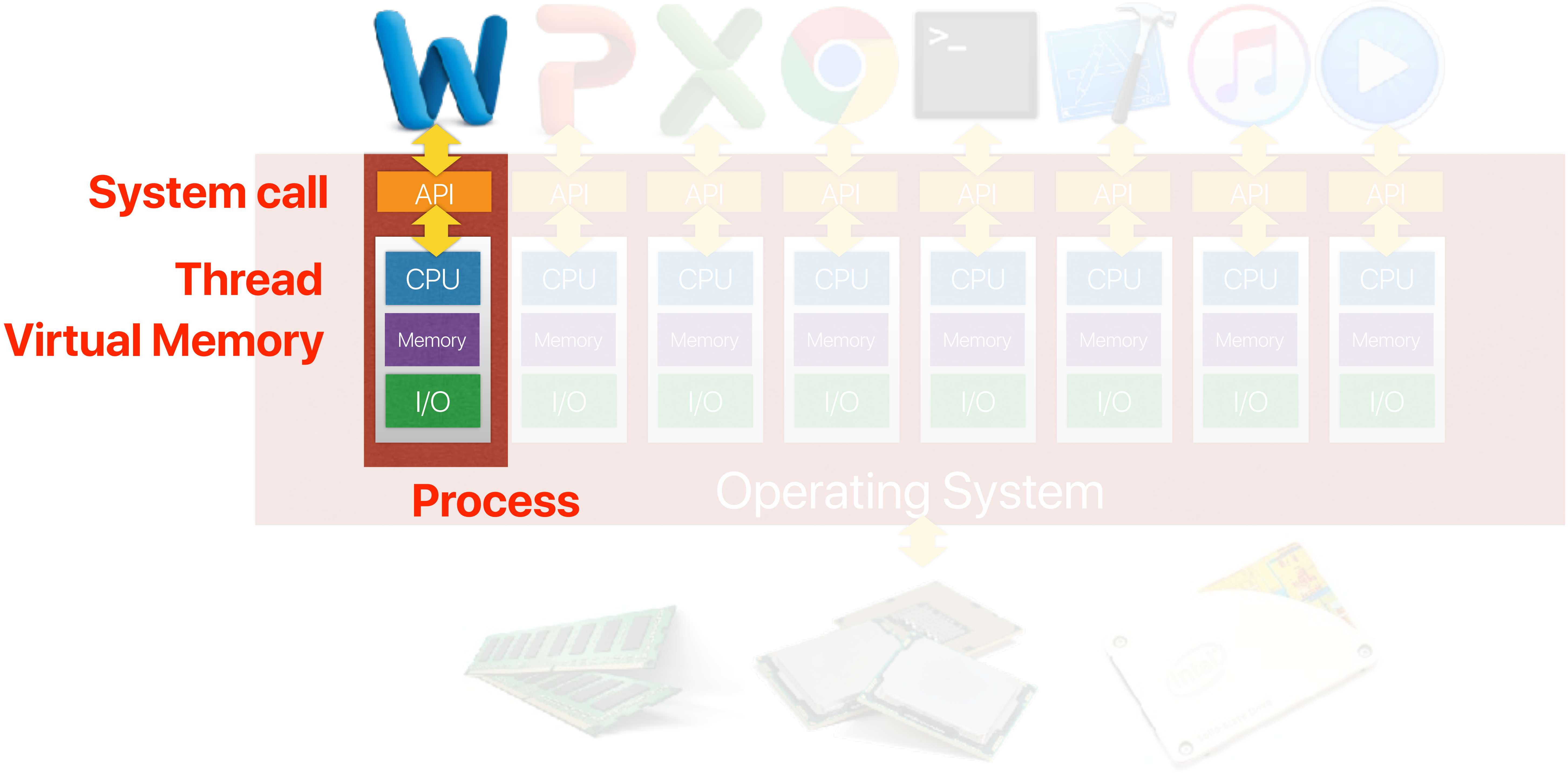
Recap: von Neumman Architecture





By loading different programs into memory, your computer can perform different functions

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

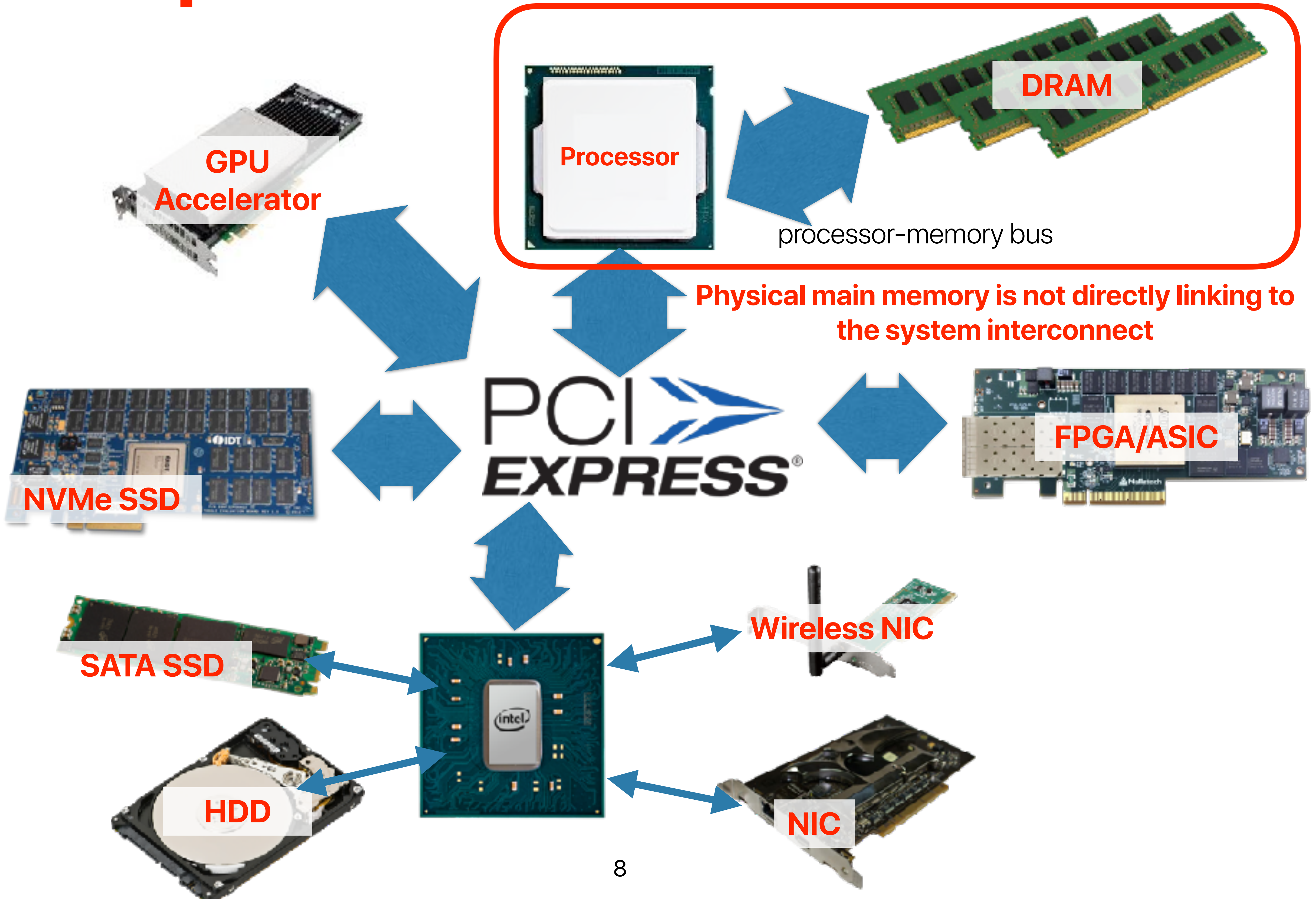
Current scoreboard

Red	Blue
16	16

Outline

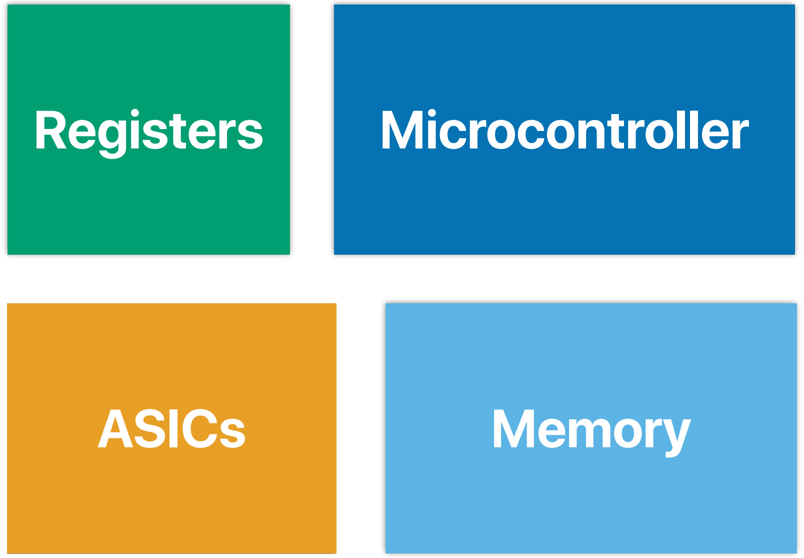
- How our systems interact with I/O
- The basics of storage devices
- File

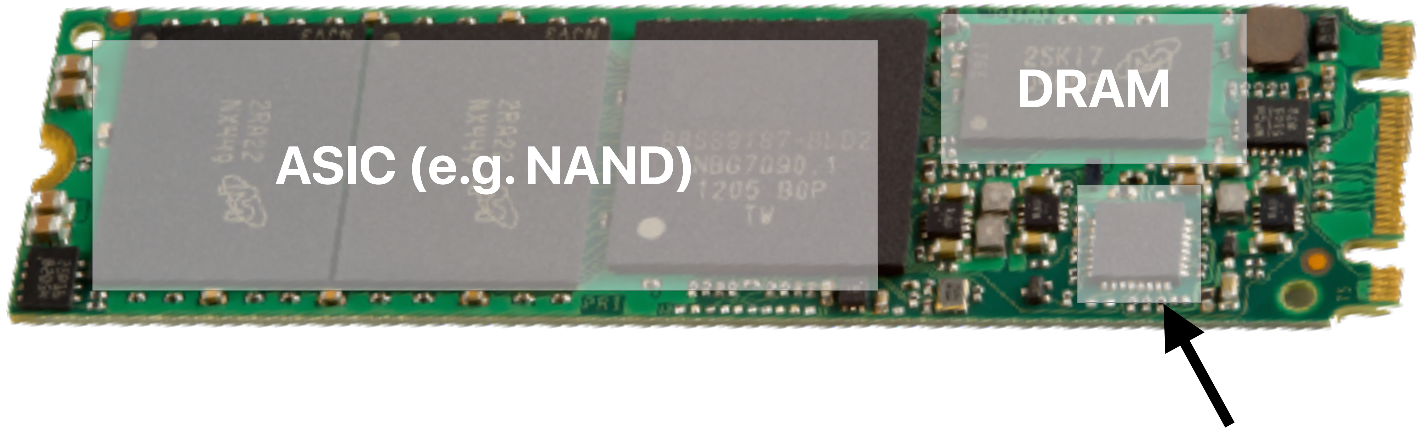
The computer is now like a small network



What's in each device?

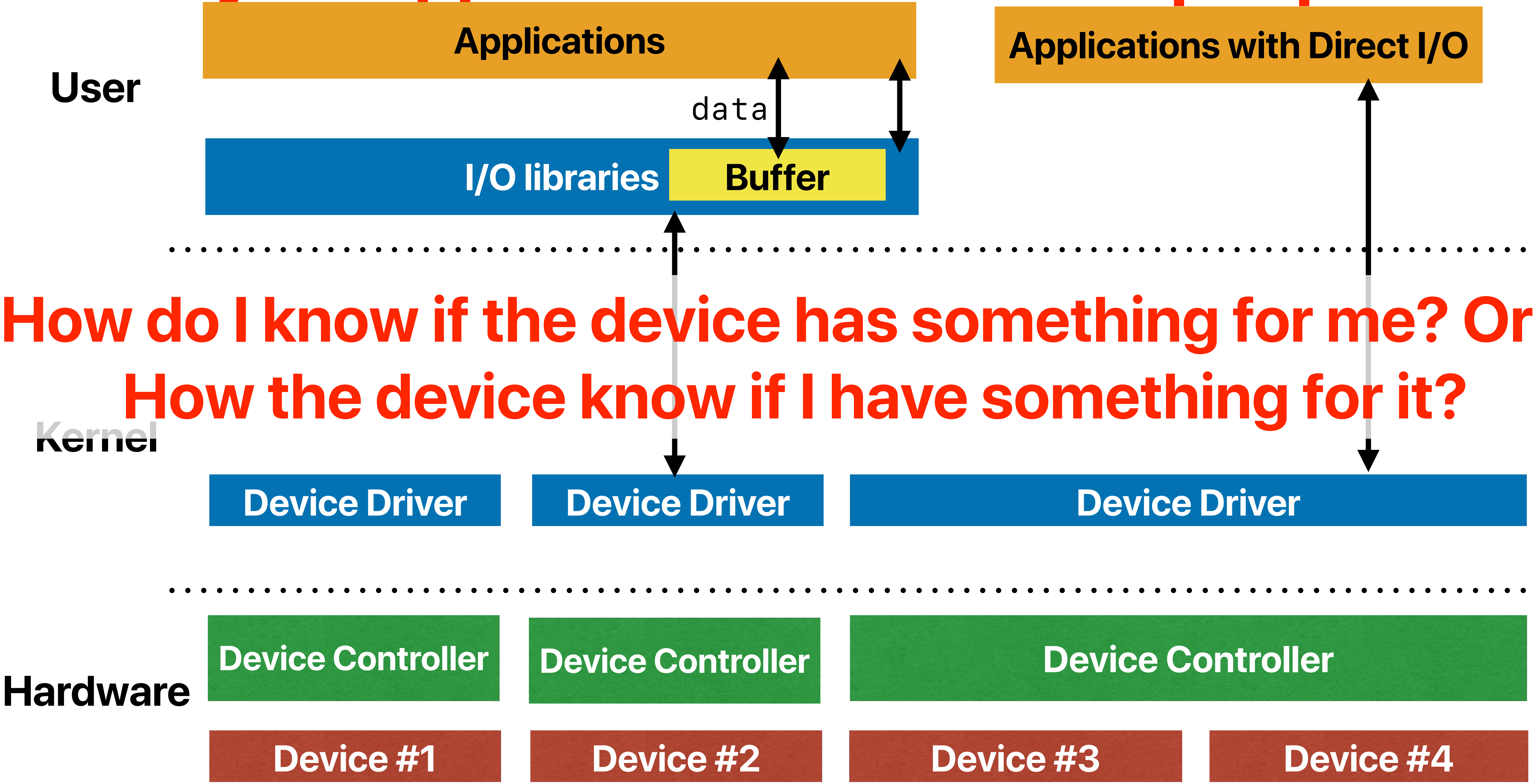
- Registers
  - Command: receiving commands from host
  - Status: tell the host the status of the device
  - Data: the location of exchanging data
- Microcontroller
- Memory
- ASICs





Controller + Registers

How your application interact with peripherals



How do I know if the device has something for me? Or …
How the device know if I have something for it?

Poll close in 1:30

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

Poll close in 1:30

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

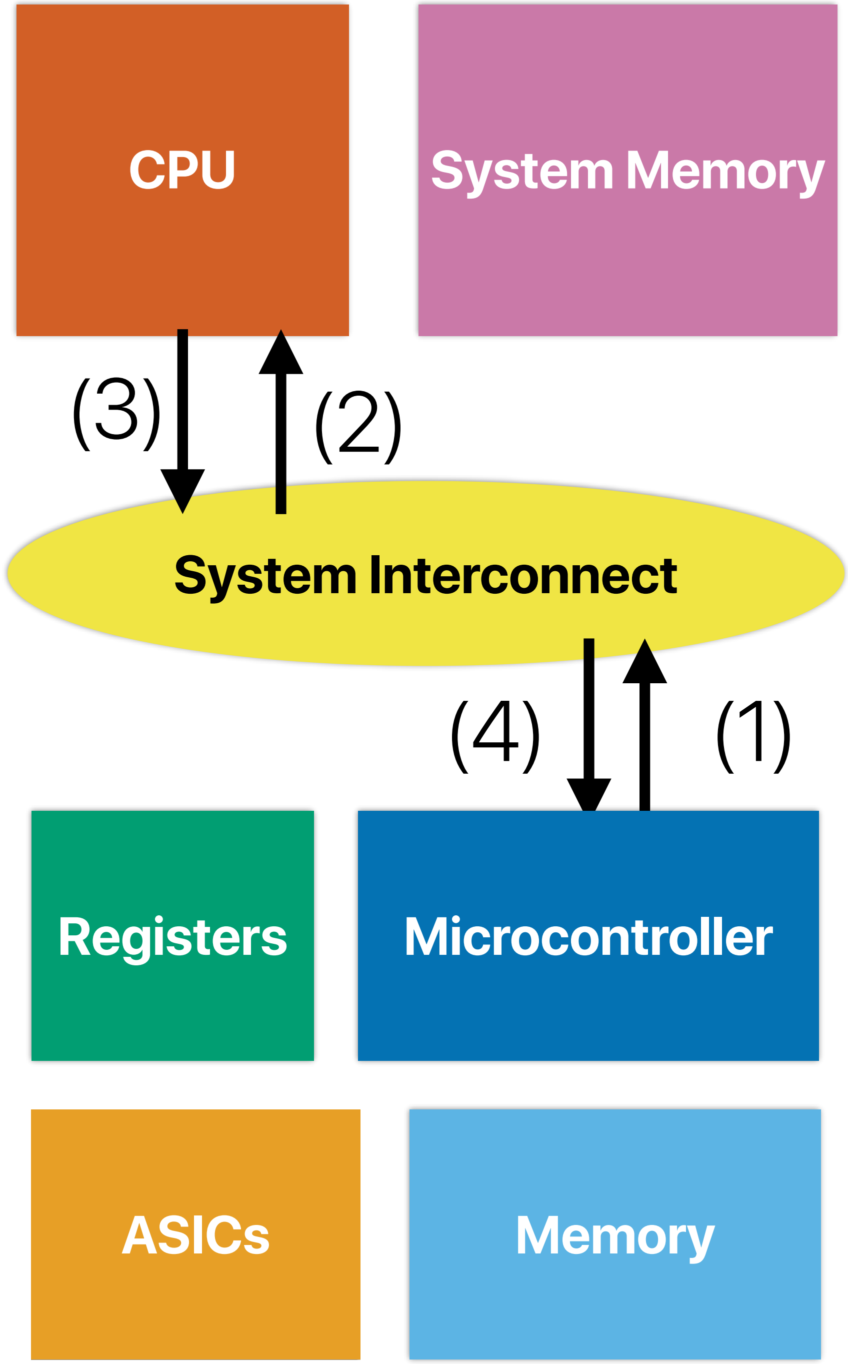
Polling v.s. Interrupt — Round 1

- 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

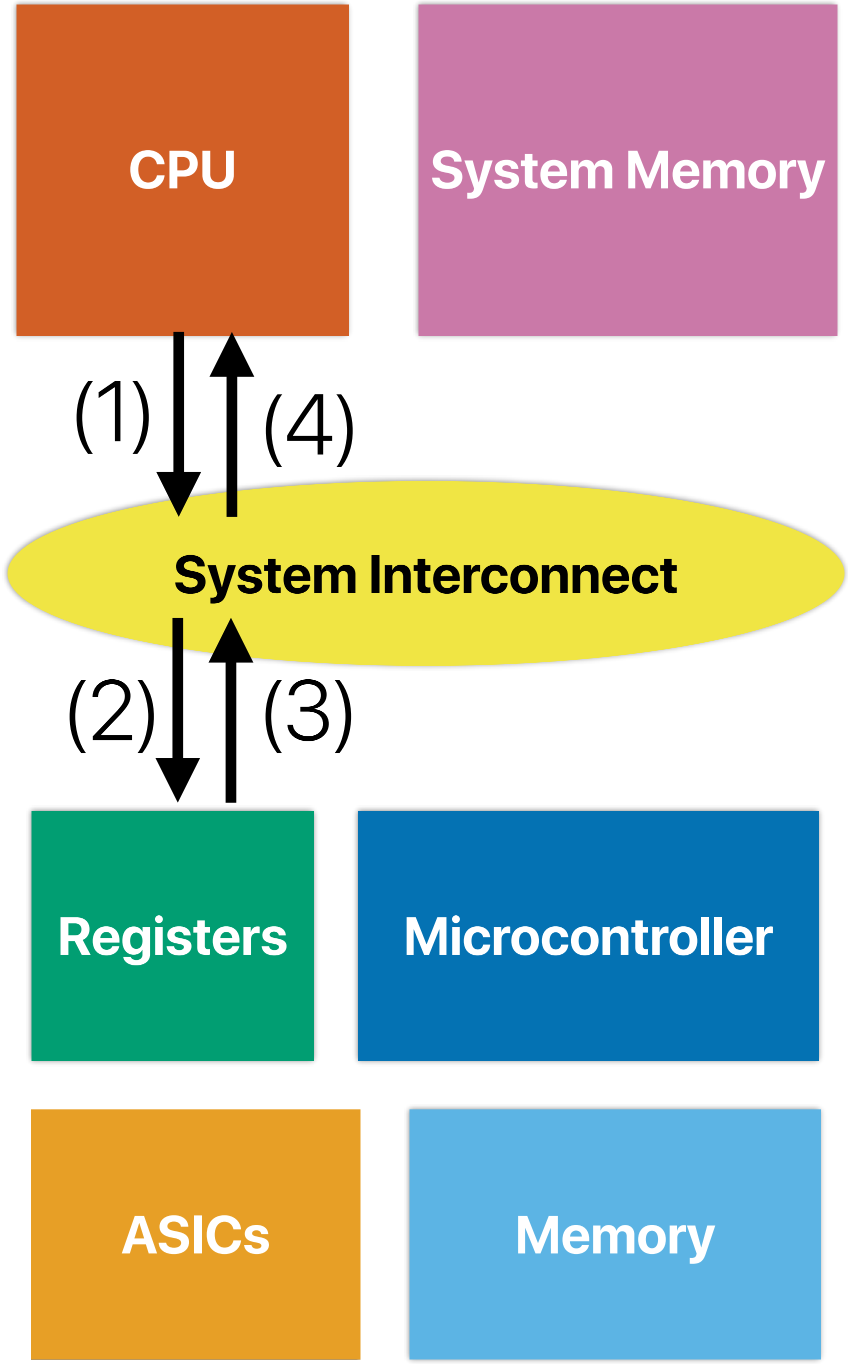
Interrupt

- 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



Poll close in 1:30

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
  ② Using interrupts may increase the cache miss rates comparing with polling
  ③ Using interrupts for high-speed storage devices may decrease the power consumption of the processor
  ④ 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

Poll close in 1:30

Interrupt v.s. Polling — Round 2

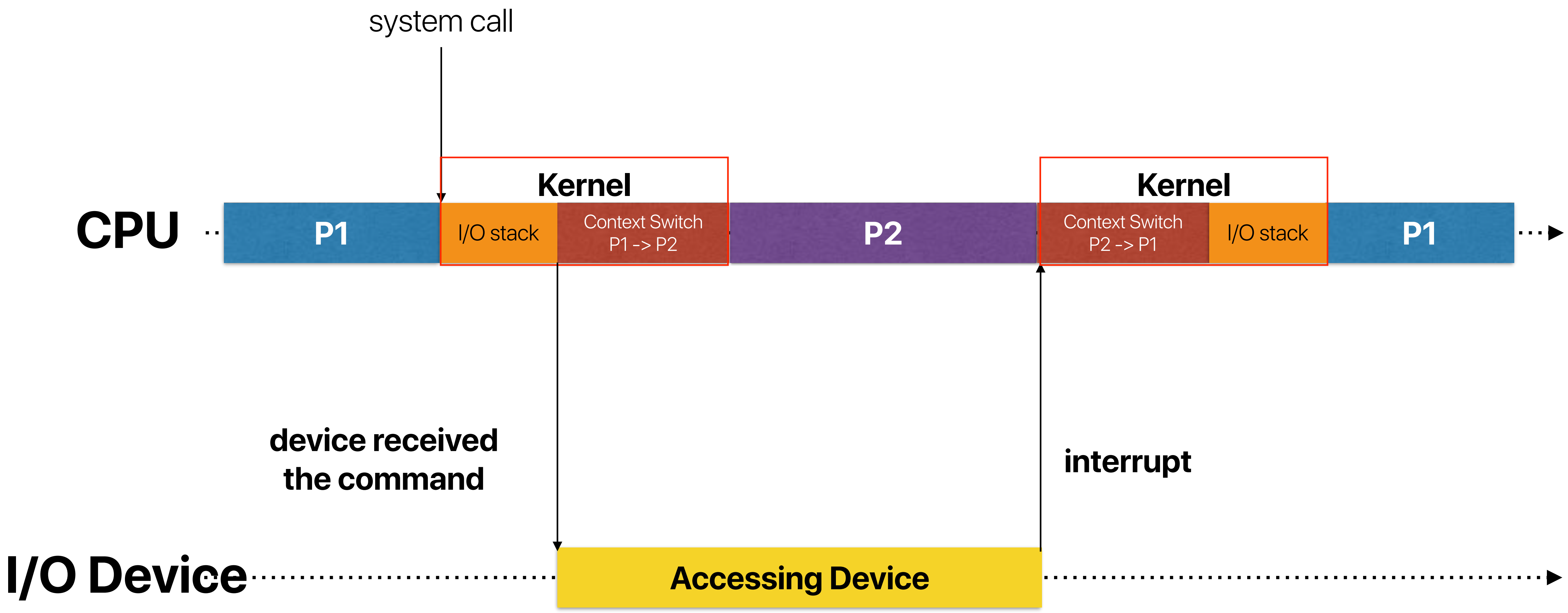
- Regarding using interrupts and polling for communicating peripheral devices, how many of the followings is/are correct?
  ① Using interrupts may increase the end-to-end latency for a process comparing with polling
  ② Using interrupts may increase the cache miss rates comparing with polling
  ③ Using interrupts for high-speed storage devices may decrease the power consumption of the processor
  ④ 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

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.



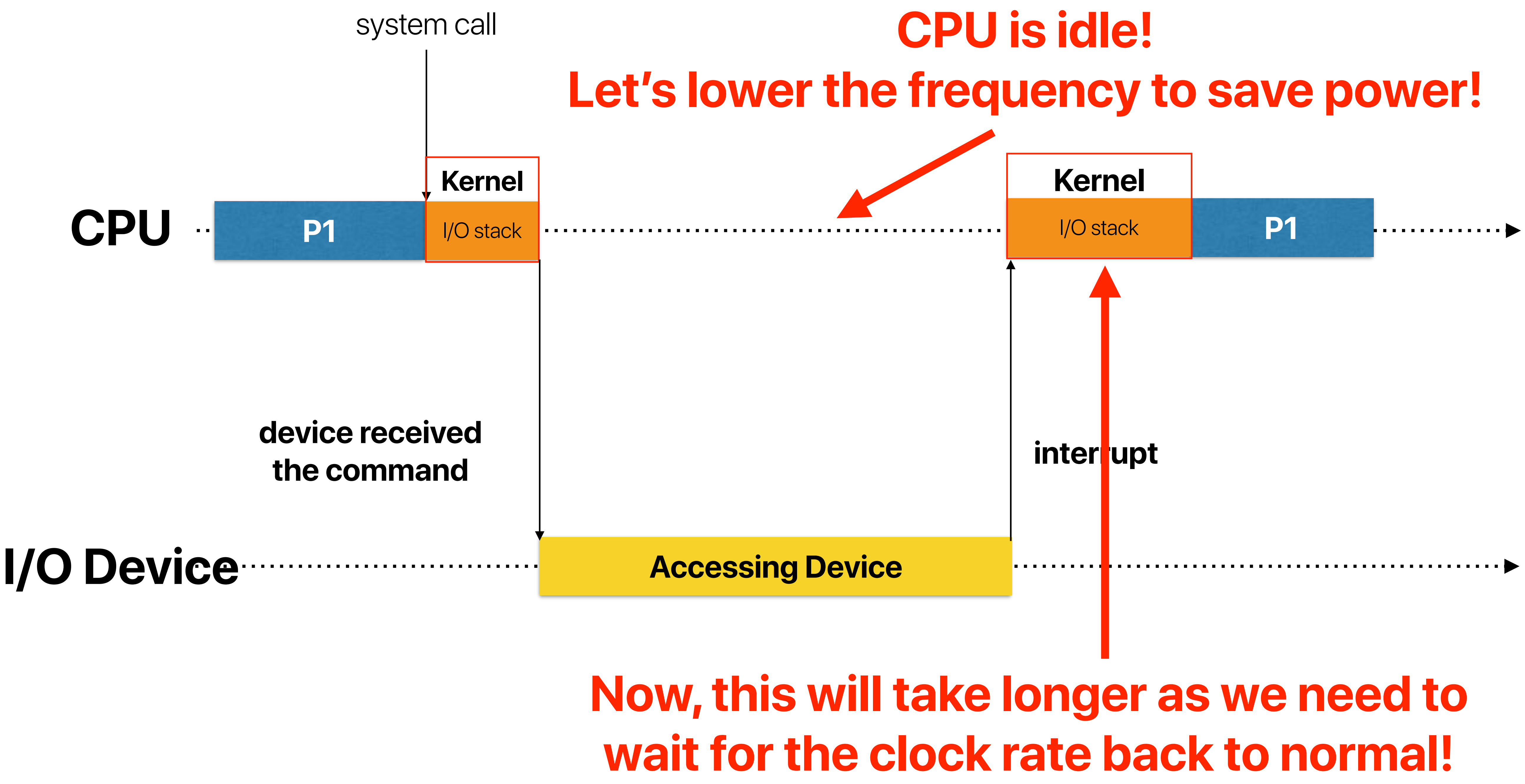
If $T_{\text{Context switch P1->P2}} + T_{\text{Context switch P2->P1}} < T_{\text{Accessing peripherals}}$

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?



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
  $\boxed{\text{E. 4}}$

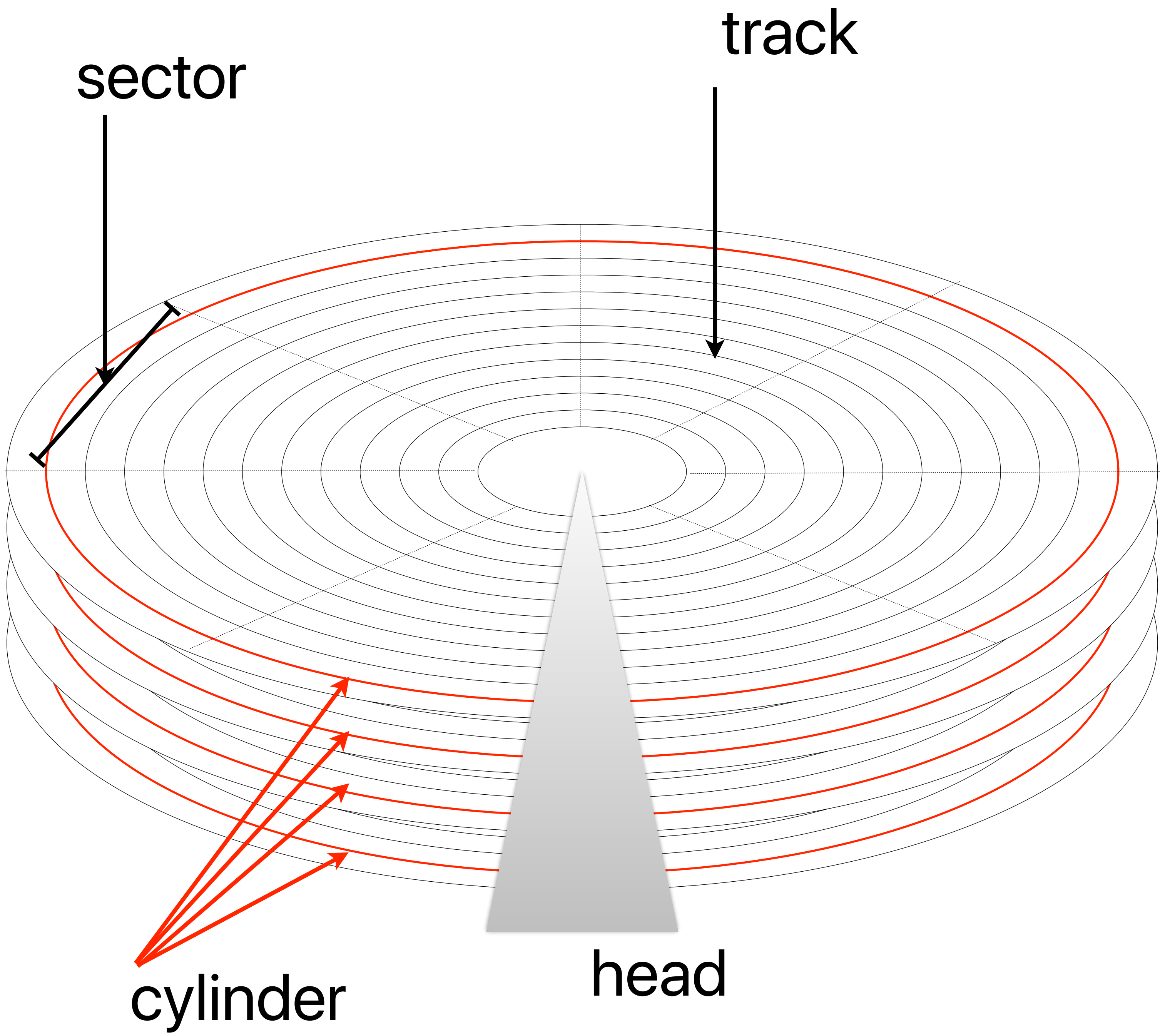
When should we poll? When should we interrupt

- 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

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

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

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

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

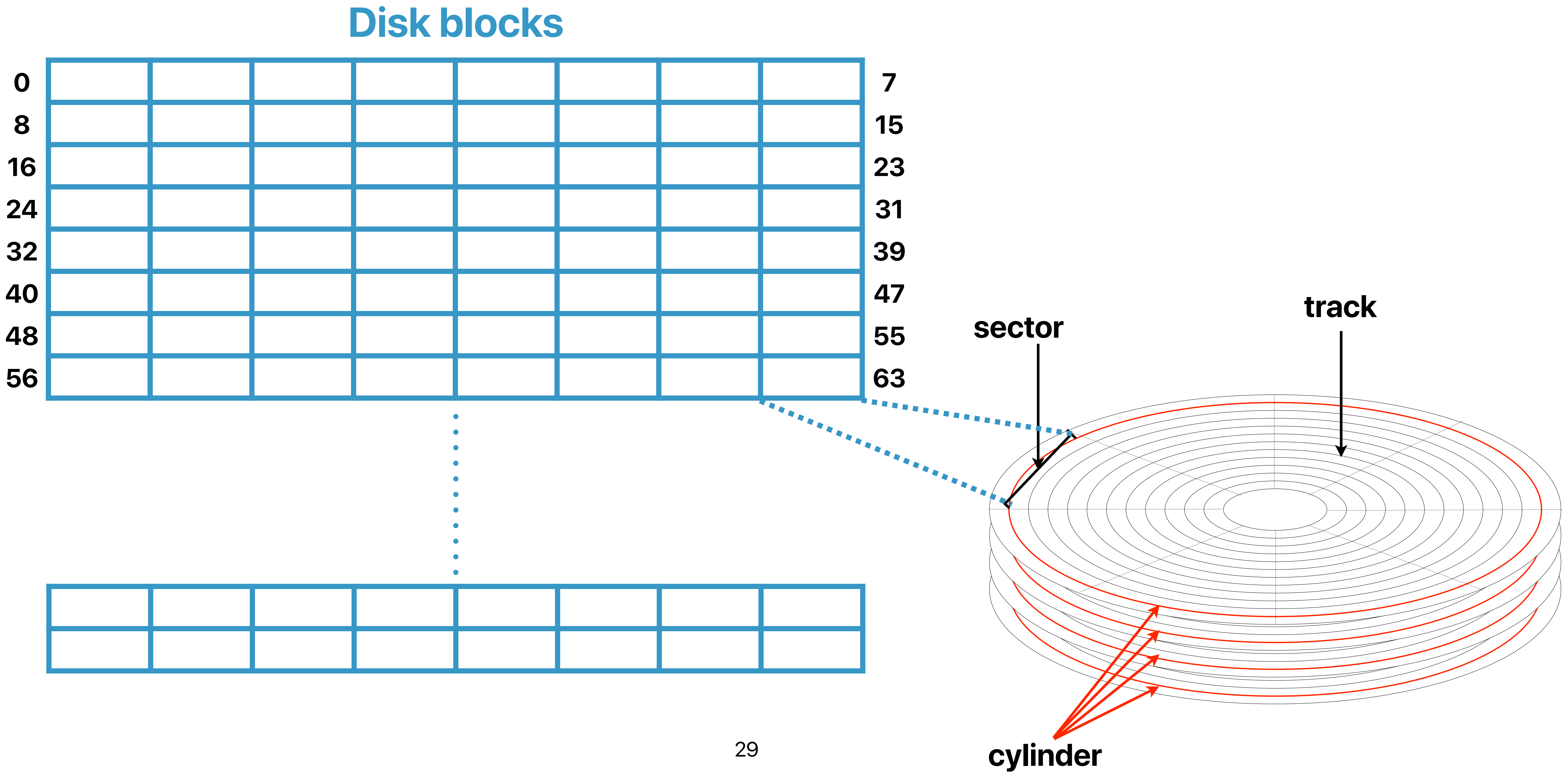
$$= 8\ ms + 4.17\ ms + 13.33\ ms + 0.2\ ms = 25.69\ ms$$

Bandwidth = volume_of_data over period_of_time

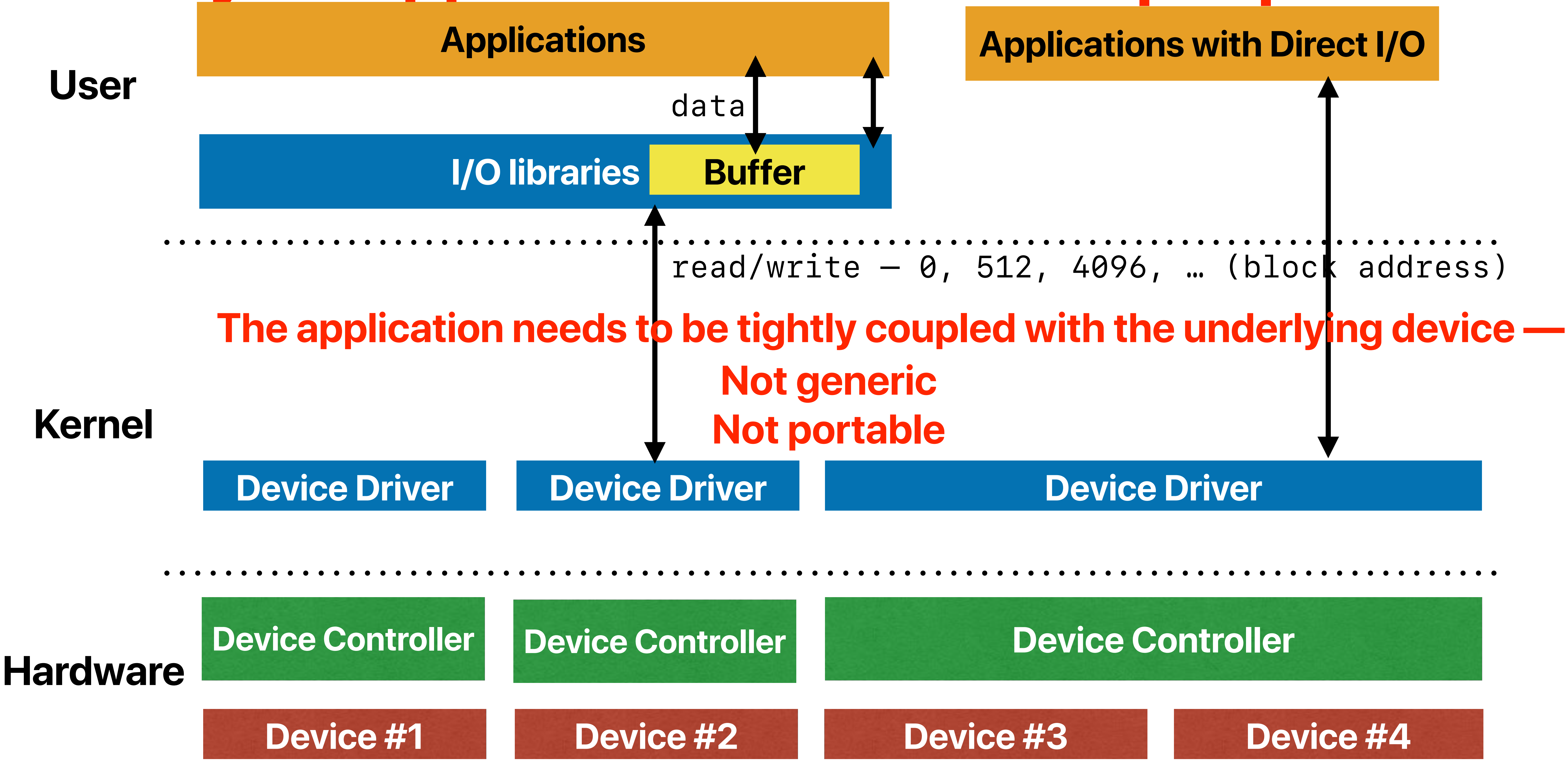
$$= \frac{4MB}{25.69ms} = 155.7\ MB/sec$$

Trading latencies with bandwidth

Numbering the disk space with block addresses



How your application interact with peripherals



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

–David Wheeler

The file & file system abstraction

Poll close in 1:30

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

Poll close in 1:30

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

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-

xpecting a file name as a param-

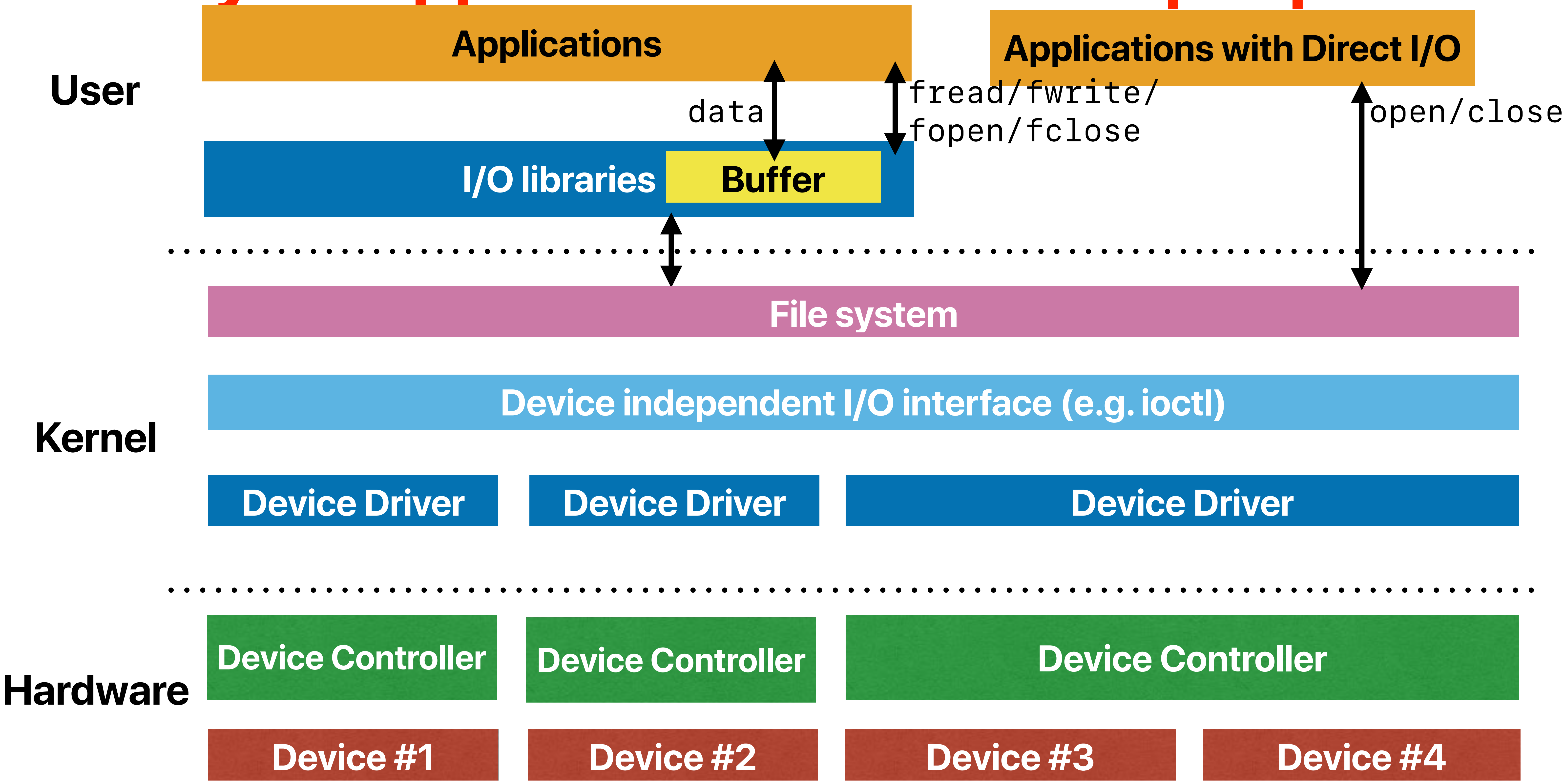
evice name; finally, special files

protection mechanism as regular

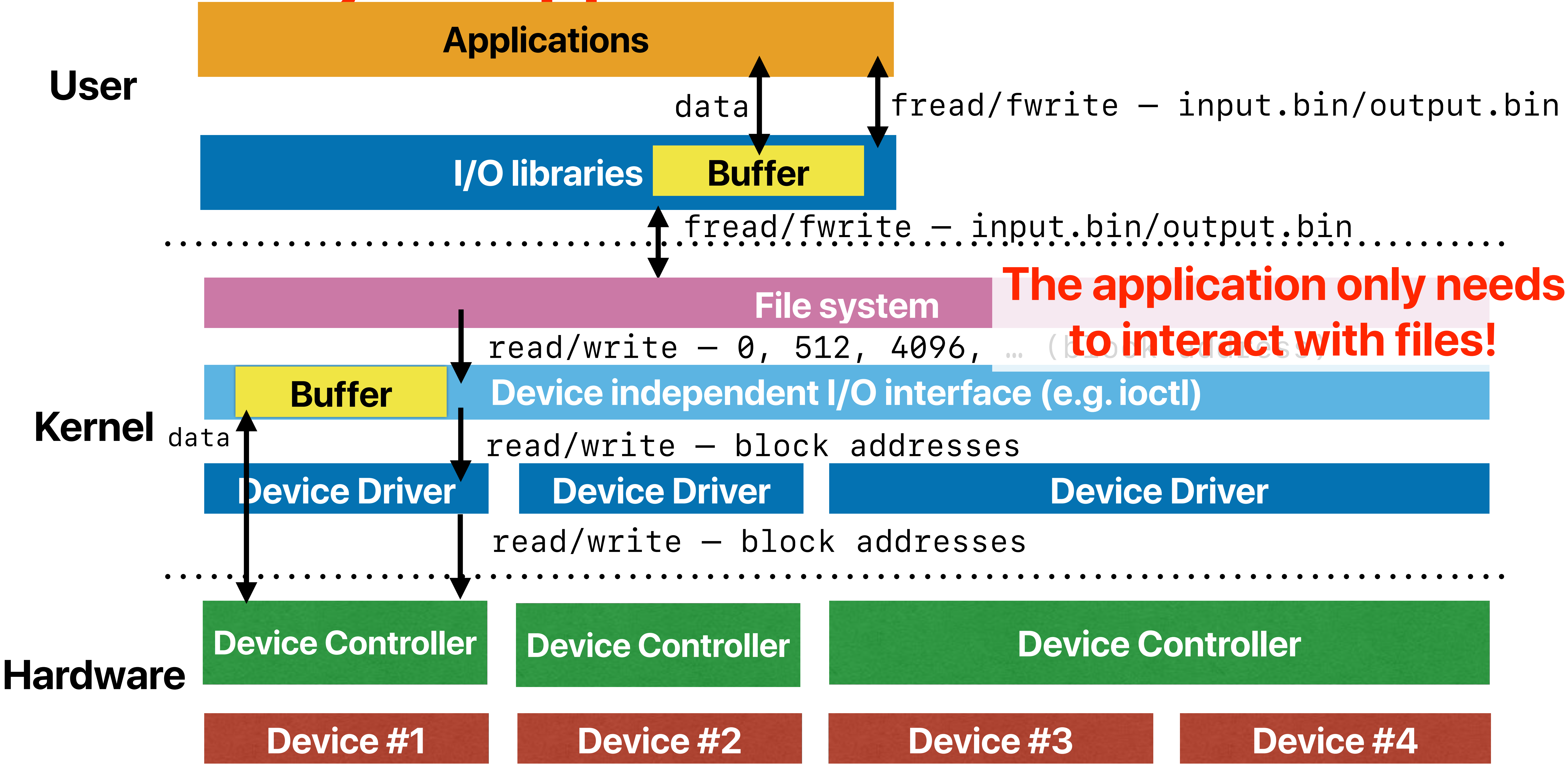
File abstraction in UNIX

- 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. $\boxed{3}$
  E. 4

How your application interact with peripherals



How your application reaches H.D.D.

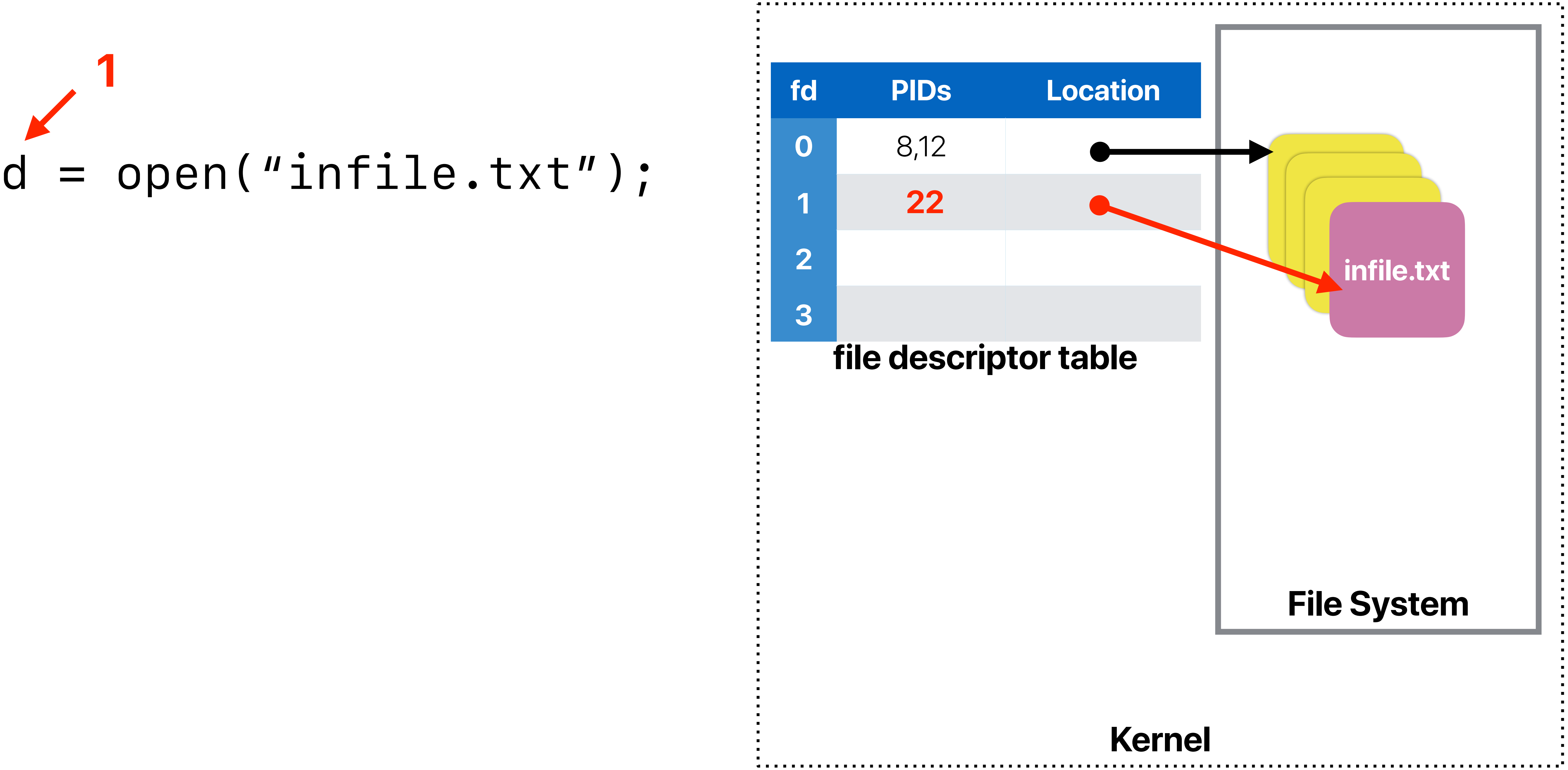


How you access files in C

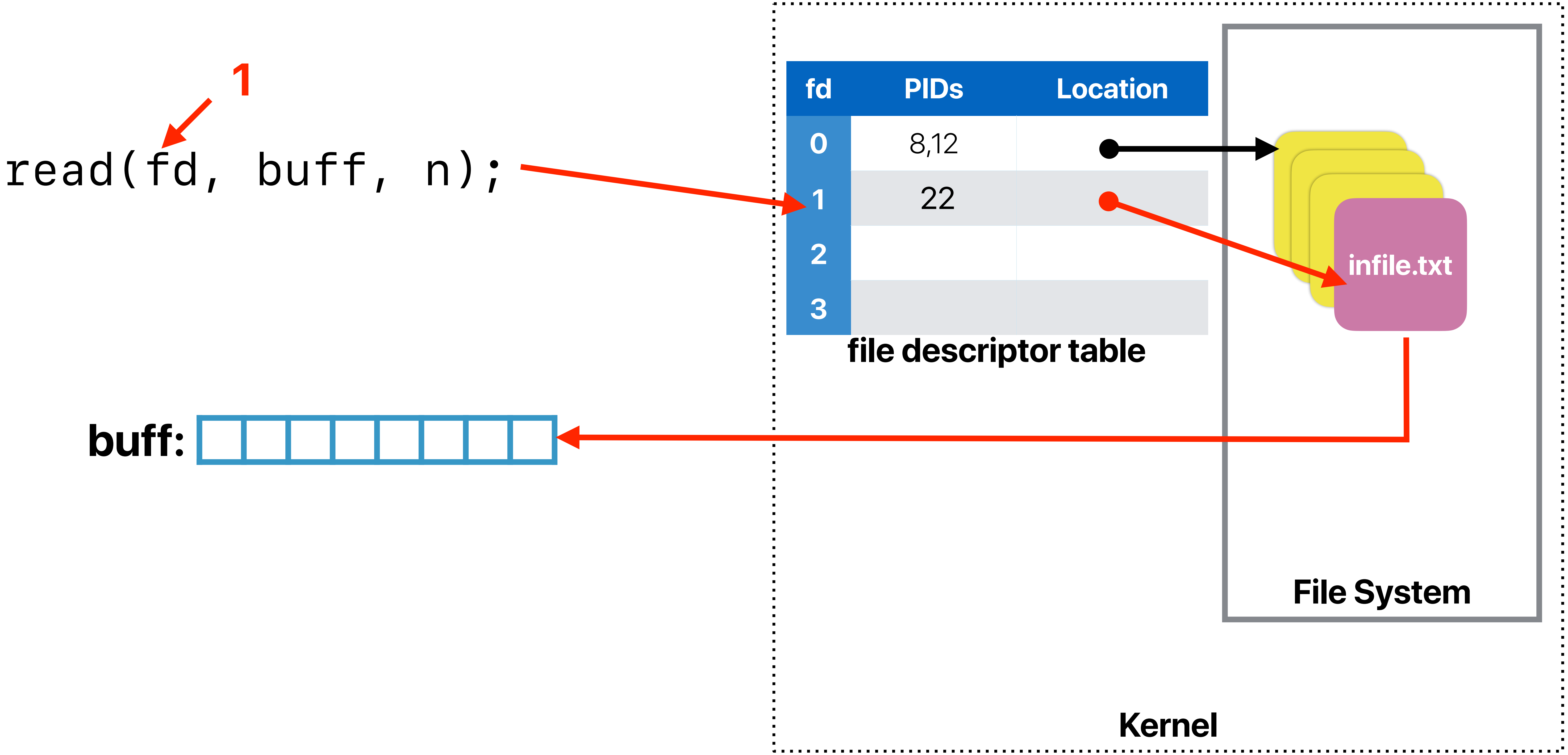
```
int fd, nr, nw;
void *in_buff;
in_buff = malloc(BUFF_SIZE);

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

```
fd = open("infile.txt");
```



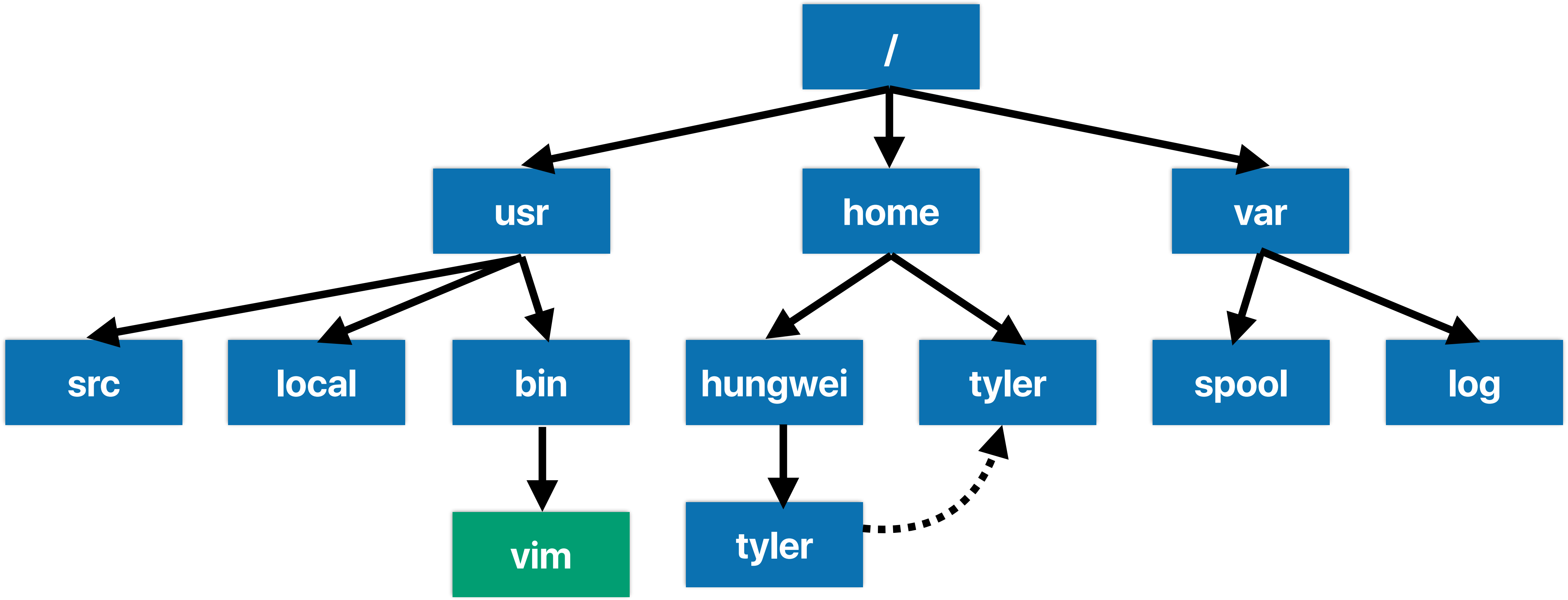
read



1
read(fd, buff, n);
fd
PIDs
Location
0
8,12
1
22
2
3
infile.txt
file descriptor table
buff:
File System
Kernel

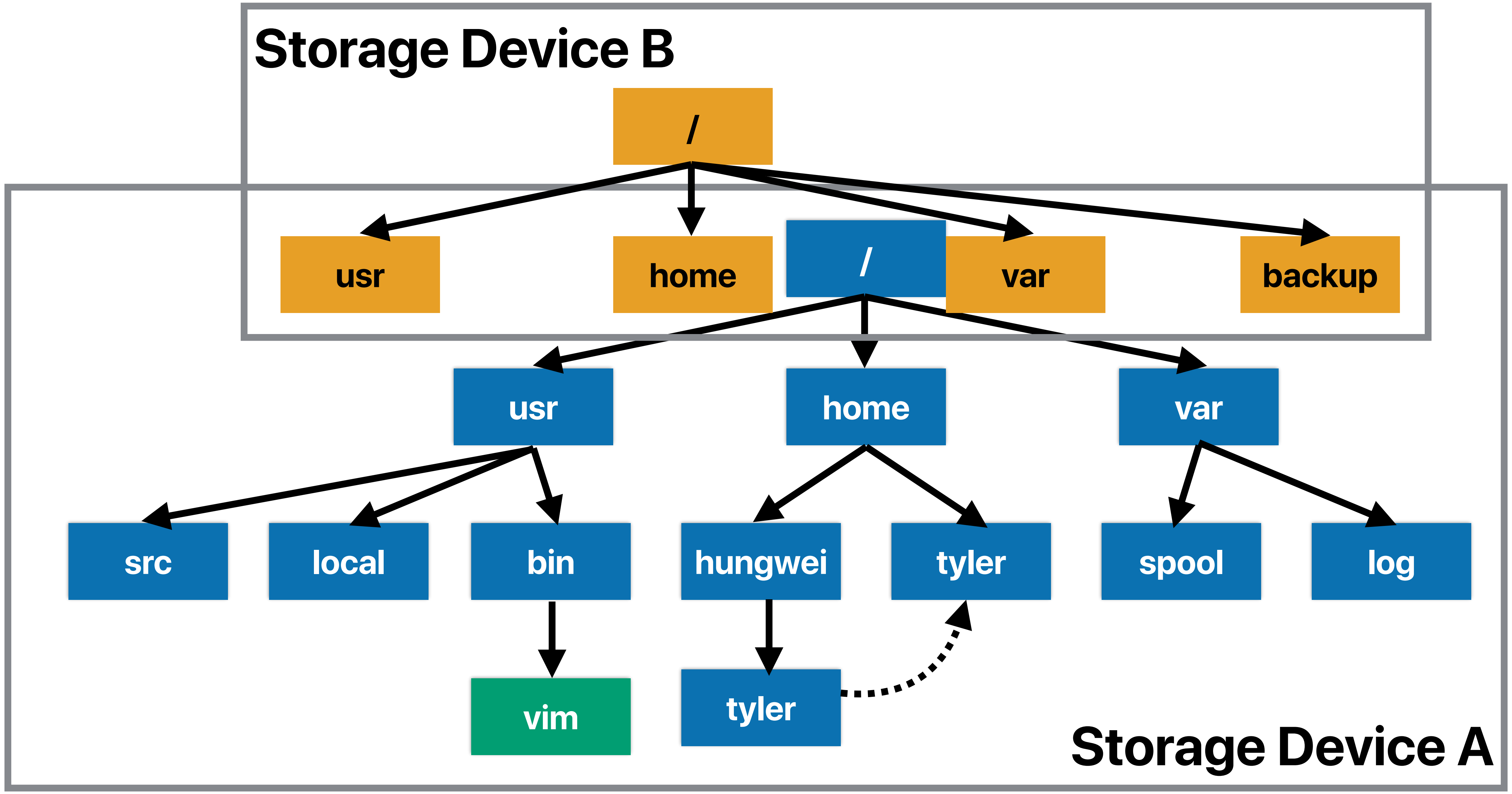
Hierarchical File System Structure

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



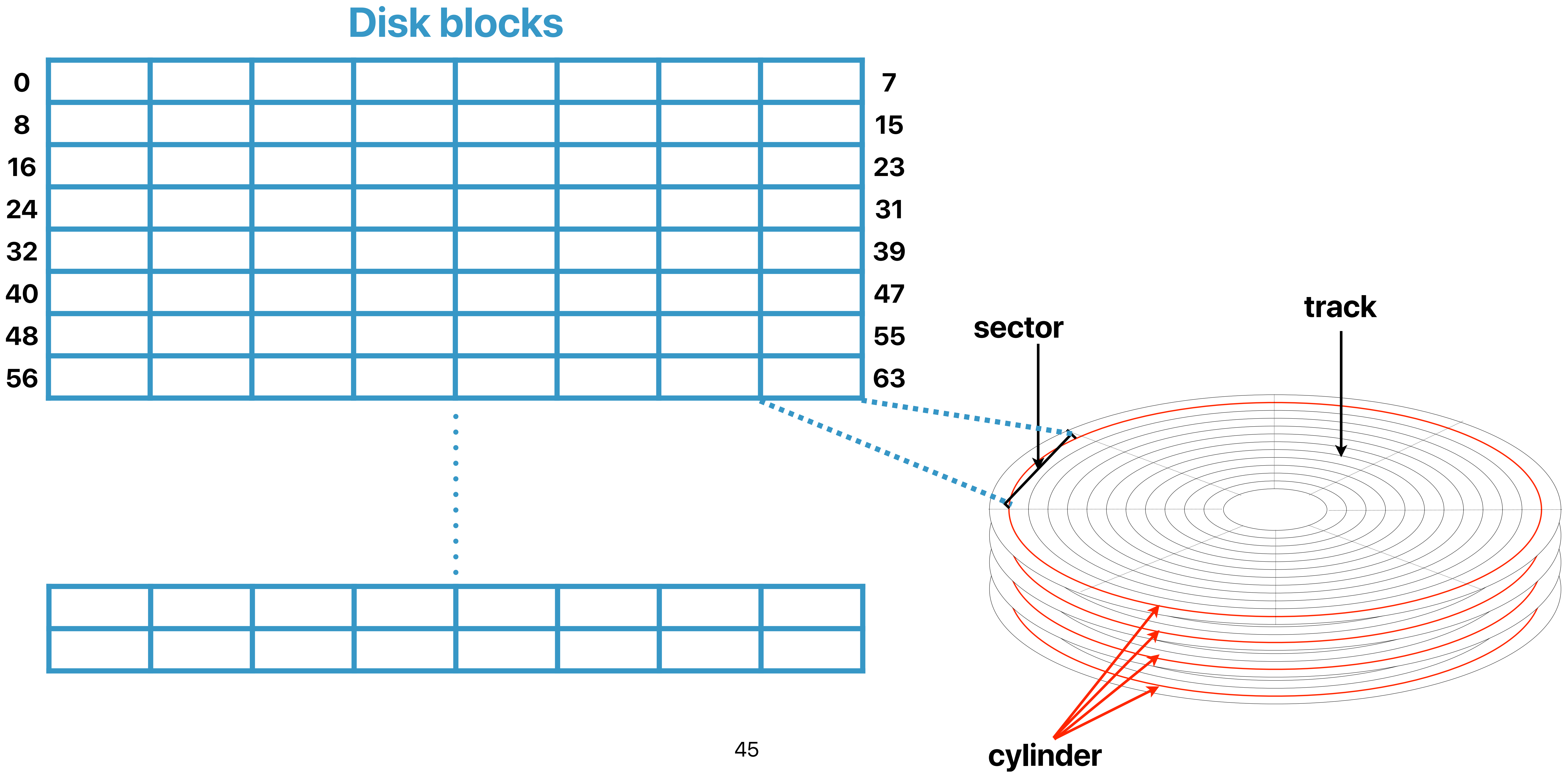
Mount

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



The design of a file system

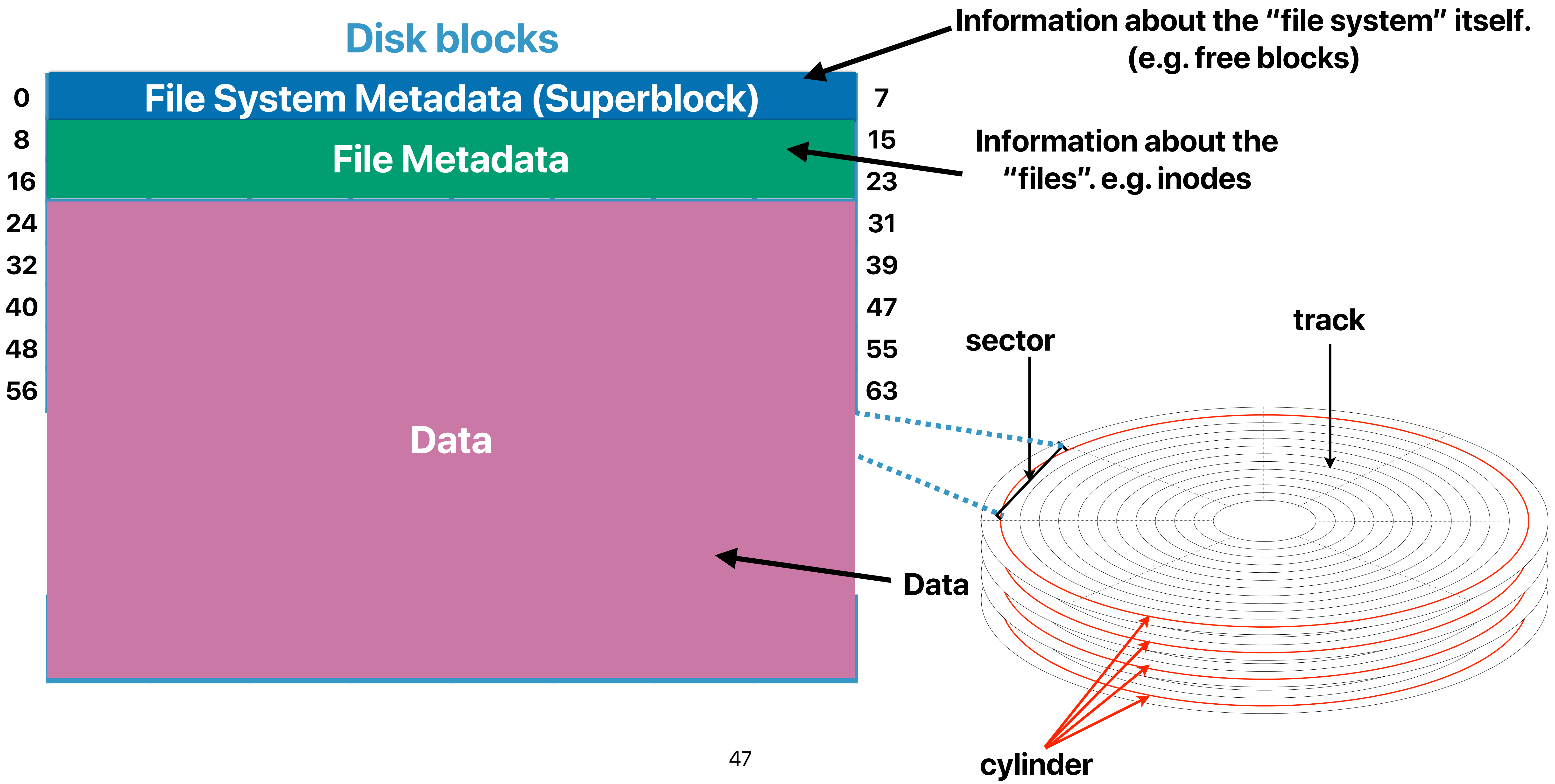
Recap: Numbering the disk space with block addresses



Questions for file systems

- How do we locate files?
  - How do we manage hierarchical namespace?
  - How do we manage file and file system metadata?
- How do we allocate storage space?
- How do we make the file system fast?
- How do we ensure file integrity?

How the original UNIX file system use disk blocks



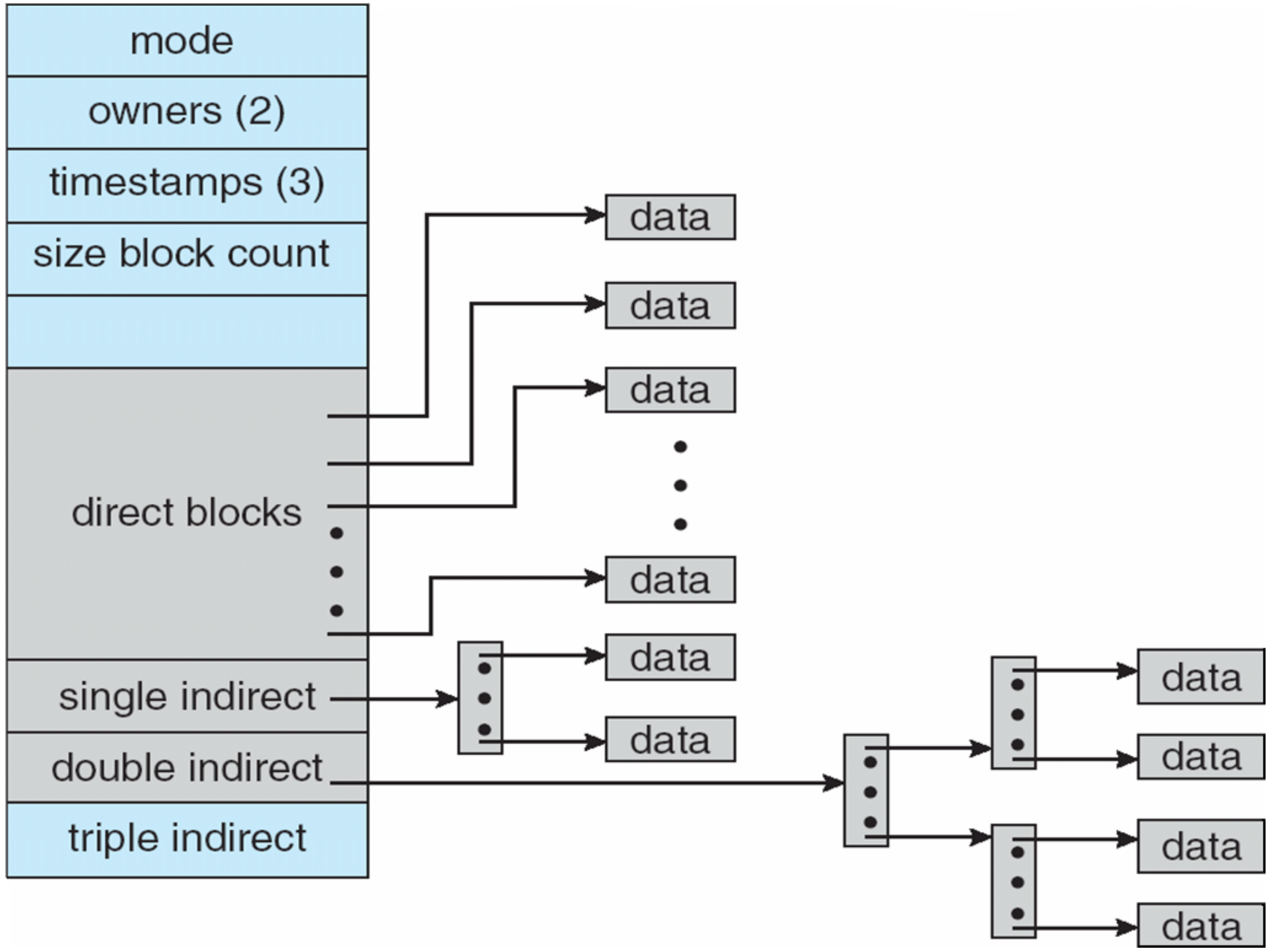
Superblock — metadata of the file system

- Contains critical file system information
  - The volume size
  - The number of nodes
  - Pointer to the head of the free list
- Located at the very beginning of the file system

inode — metadata of each file

- File types: directory, file
- File size
- Permission
- Attributes

Unix inode



- 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 n^2 data blocks
  - Triple indirect: Access n^3 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

Poll close in 1:30

Number of disk accesses

- For a file /home/hungwei/CS202/foo.c , how many accesses does the original/old, unoptimized UNIX file system need to perform to reach the actual file content in the worst case?
  A. 4
  B. 6
  C. 8
  D. 9
  E. At least 10

Poll close in 1:30

Number of disk accesses

- For a file /home/hungwei/CS202/foo.c , how many accesses does the original/old, unoptimized UNIX file system need to perform to reach the actual file content in the worst case?
  A. 4
  B. 6
  C. 8
  D. 9
  E. At least 10

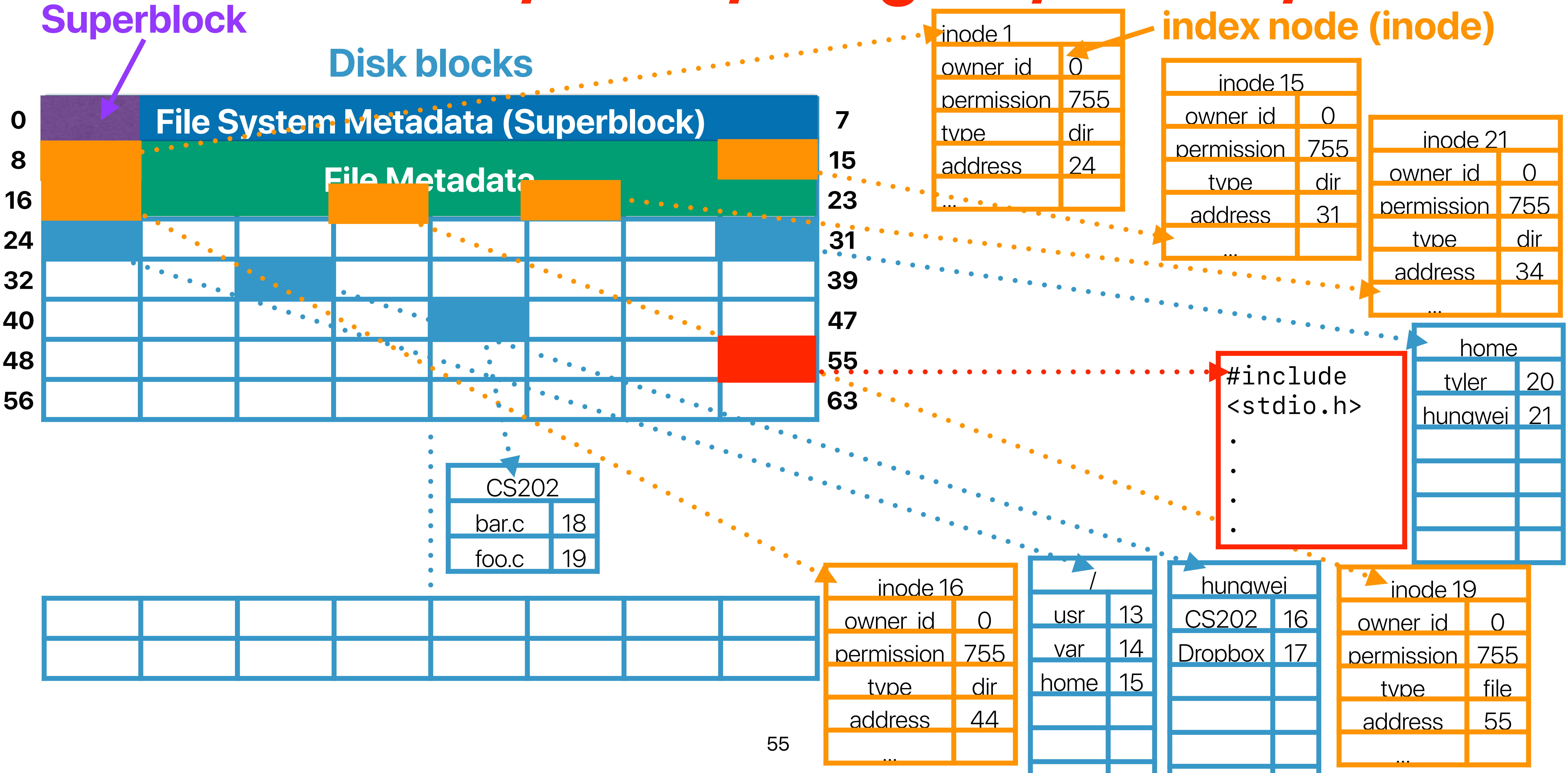
Number of disk accesses

- For a file /home/hungwei/CS202/foo.c , how many accesses does the original/old, unoptimized UNIX file system need to perform to reach the actual file content in the worst case?
  A. 4
  B. 6
  C. 8
  D. 9
  E. At least 10

What must be done to reach your files

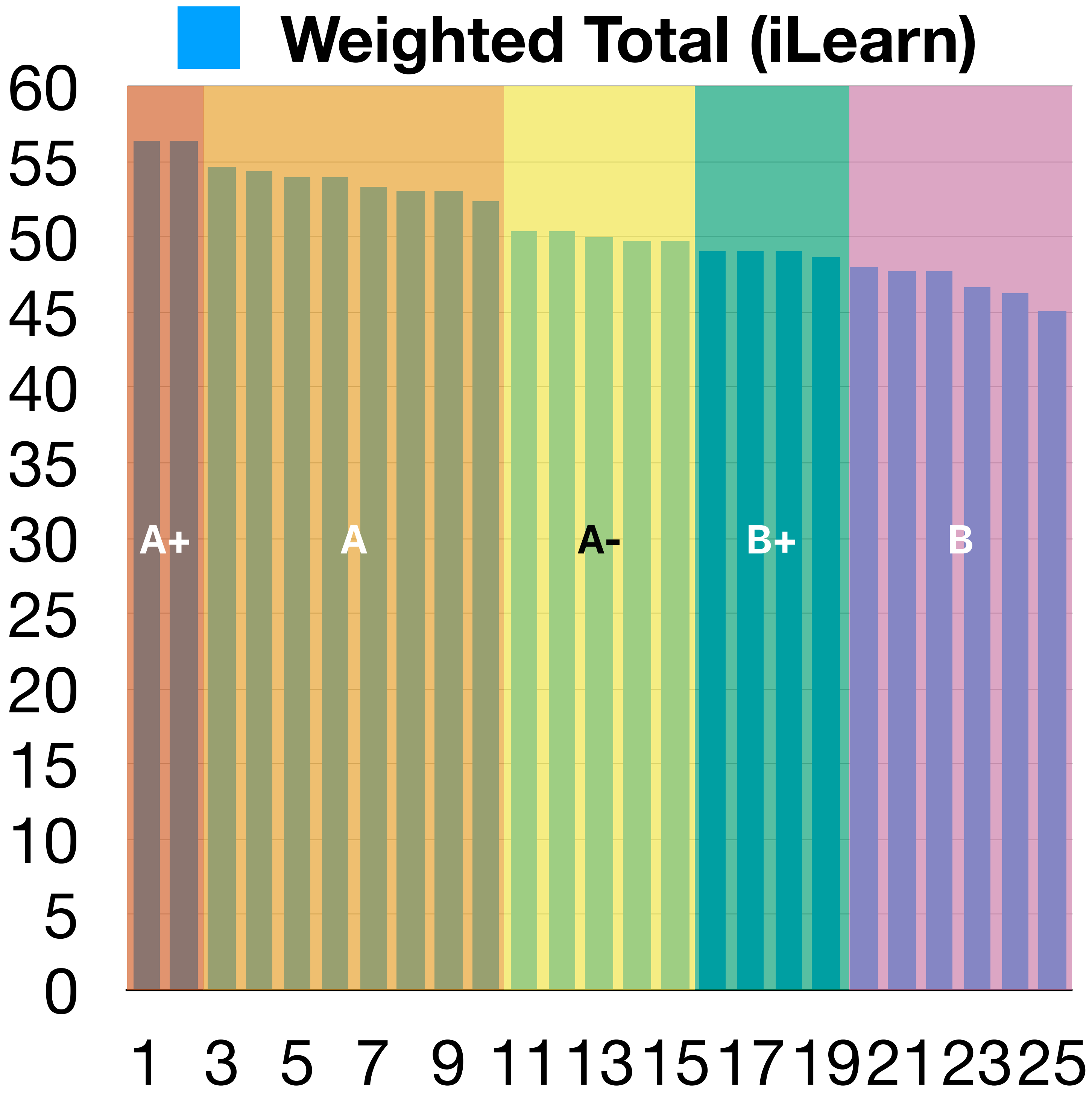
- Scenario: User wants to access /home/hungwei/CS202/foo.c
- Procedure: File system will…
  - Open "/" file (This is in known from superblock.)
  - Locate entry for "home," open that file
  - Locate entry for "hungwei", open that file
  - …
  - Locate entry for "foo.c" and open that file
- Let's use "strace" to see what happens

How to reach /home/hungwei/CS202/foo.c



Announcement

- Reading quizzes due **this Thursday**
- Office hour
  - M 3p-4p and Th 9a-10a
  - Use the office hour Zoom link, not the lecture one
- Midterm
  - Median: 76
  - You may find your grade and reference solution (midterm and final section) in iLearn
  - One week regrading policy, no regrading request after 2/23 — check the website for details
- Project
  - **Due 3/2**
  - **No late submission is allowed**



Computer
Science &
Engineering

202

つづく