https://hao-ai-lab.github.io/dsc204a-w24/

DSC 204A: Scalable Data Systems
Winter 2024

Machine Learning Systems

Big Data

Cloud

Foundations of Data Systems

Feedback and Logistics

Request: Upload slide deck before class?

- Yes – we're catching up

Book: Design data-intensive applications

- Been in student folder

Practice Qs (review next class)

Q1: How much space do I need to store GPT-3 ?

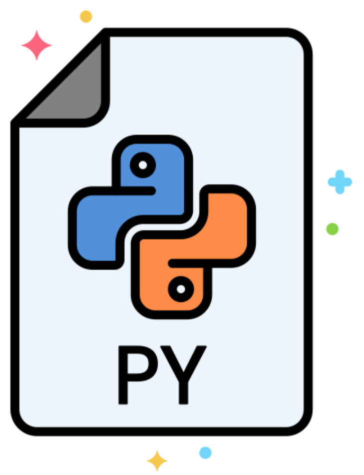
Q2: What do **exponent** and **fraction** control in float point representation?

Q3: What is the difference between BF16 and FP16?

Q1: How much space do I need to store GPT-3 ?

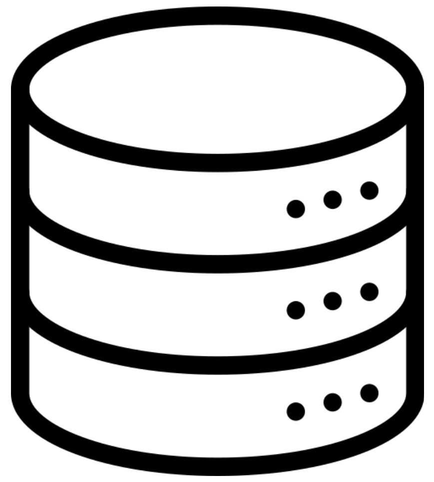
- What is GPT-3
  - An ML model with trained weights
  - = a **software** with some **built-in data**

GPT-3 =



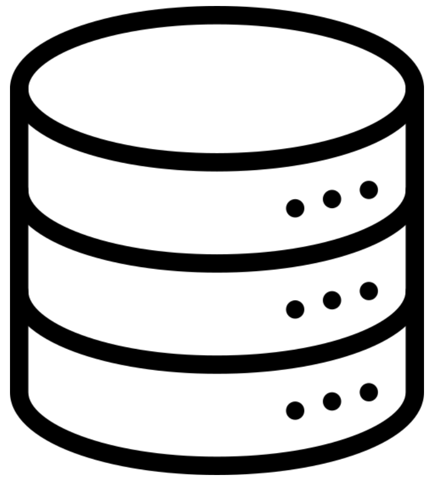
A few KBs?

+



Parameters:
How large is this?

Q1: How much space do I need to store GPT-3 ?



Parameters:
How large is this?

Data type?	# data
Bf16: 16-bit	175B

2 bytes x 175B

= 350 B bytes

= 350 GB

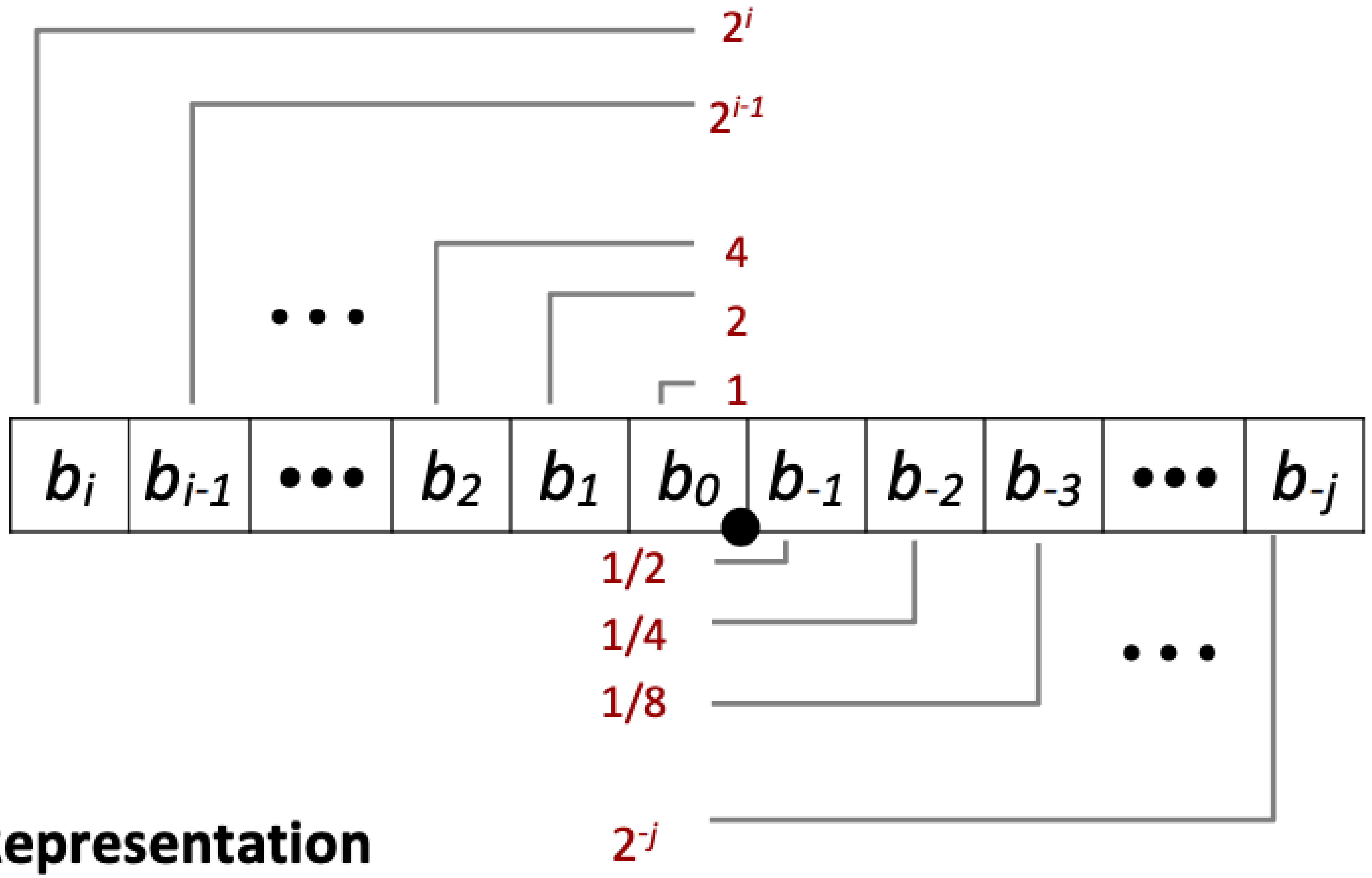
Practice Qs (review next class)

Q1: How much space do I need to store GPT-3 ?

Q2: What do exponent and fraction control in float point representation?

Q3: What is the difference between BF16 and FP16?

Fractional Binary Numbers



- **Representation**
  - Bits to right of "binary point" represent fractional powers of 2
  - Represents rational number:

$$\sum_{k=-j}^{i} b_k \times 2^k$$

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Let's design a fix-point FP6

	Sign	Integer		Fraction		
	0	0	1	0	1	0
Bit index:	5	4	3	2	1	0
	+	2	1	1/2	1/4	1/8

1+1/4= 1.25

Can represent numbers from -3.875 (111111) to 3.875 (011111).

An Example

$$0.625_{10} =$$

$$0.625_{10} = 0.101_2$$

$$0.625_{10} = 0.101_2 = 1.01 \cdot 2^{-1}$$

An Example (Cont.)

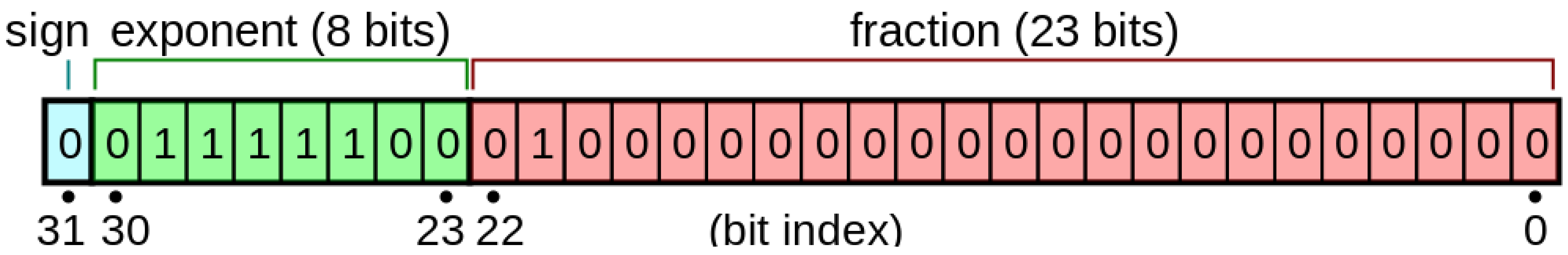
$$0.625_{10} = 0.101_2 = 1.01 \cdot 2^{-1}$$

$$(-1)^0 \cdot 2^{(1-2)} \cdot \left(1 + 0 \cdot \frac{1}{2} + 1 \cdot \frac{1}{4} + 0 \cdot \frac{1}{8}\right)$$

	sign (1 bit)	exponent (2 bits)		fraction (3 bits)		
	0	0	1	0	1	0
Bit index:	5	4	3	2	1	0

Digital Representation of Data

- Float:
  - Standard IEEE format for single (aka binary32):

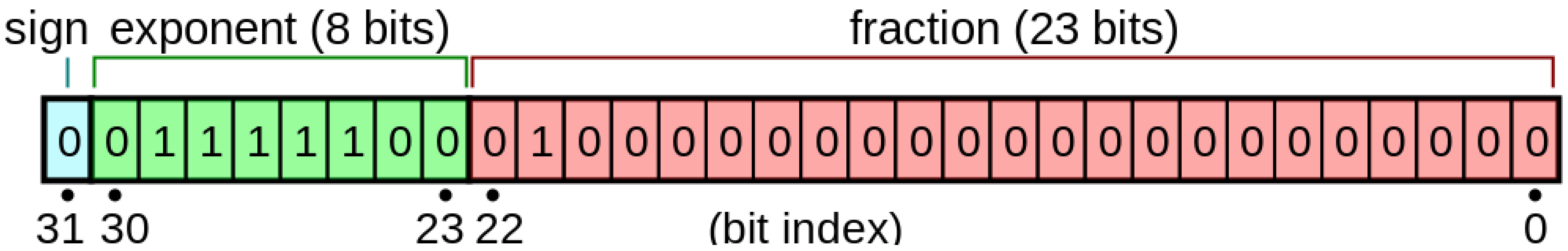


$$(-1)^{sign} \times 2^{exponent-127} \times \left(1 + \sum_{i=1}^{23} b_{23-i} 2^{-i}\right)$$

$$(-1)^0 \times 2^{124-127} \times (1 + 1 \cdot 2^{-2}) = (1/8) \times (1 + (1/4)) = 0.15625$$

Q2: What do **exponent** and **fraction** control?

- Exponent controls: range, offset
- Fraction controls: actual value, precision

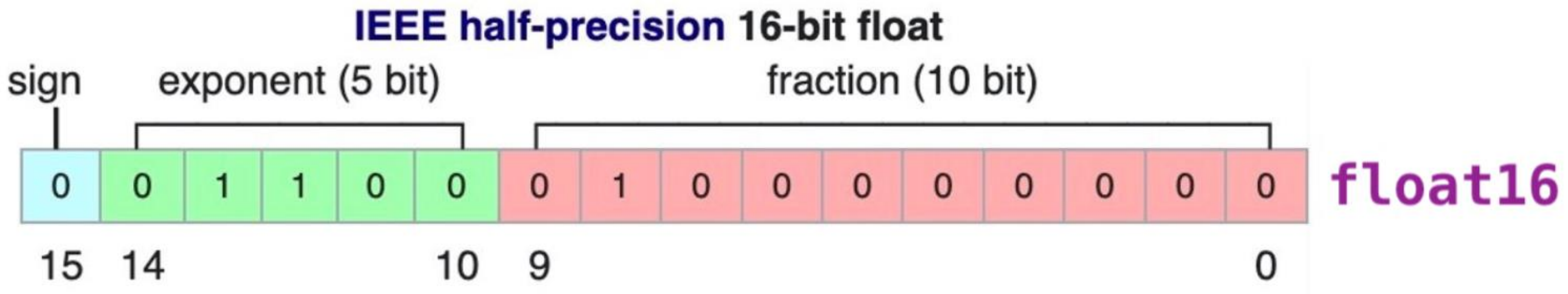


Q2: What do **exponent** and **fraction** control?

Any problem about floating point (compared to fixed point)?

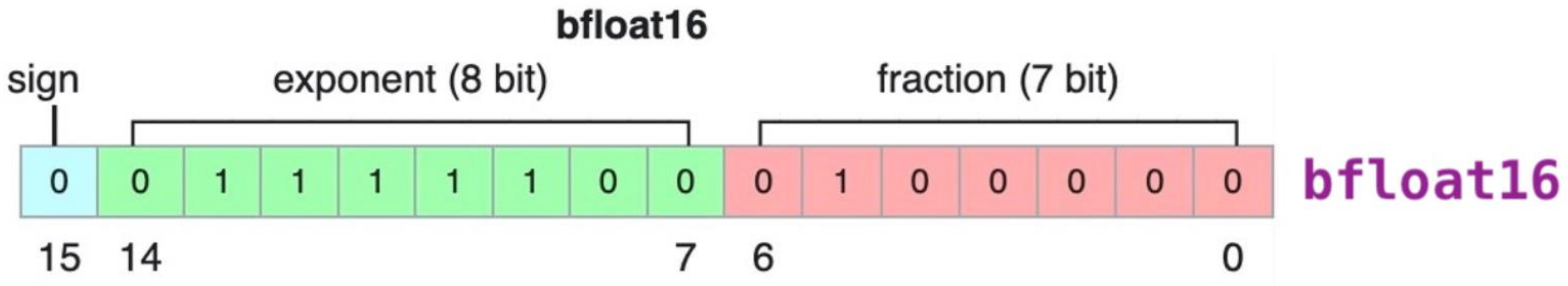
- More complex (to both human and computers)
- Inconsistent precision

Q3: What is the difference between BF16 and FP16?



Less exponent -> smaller range -> easier to overflow

More fraction -> more precise

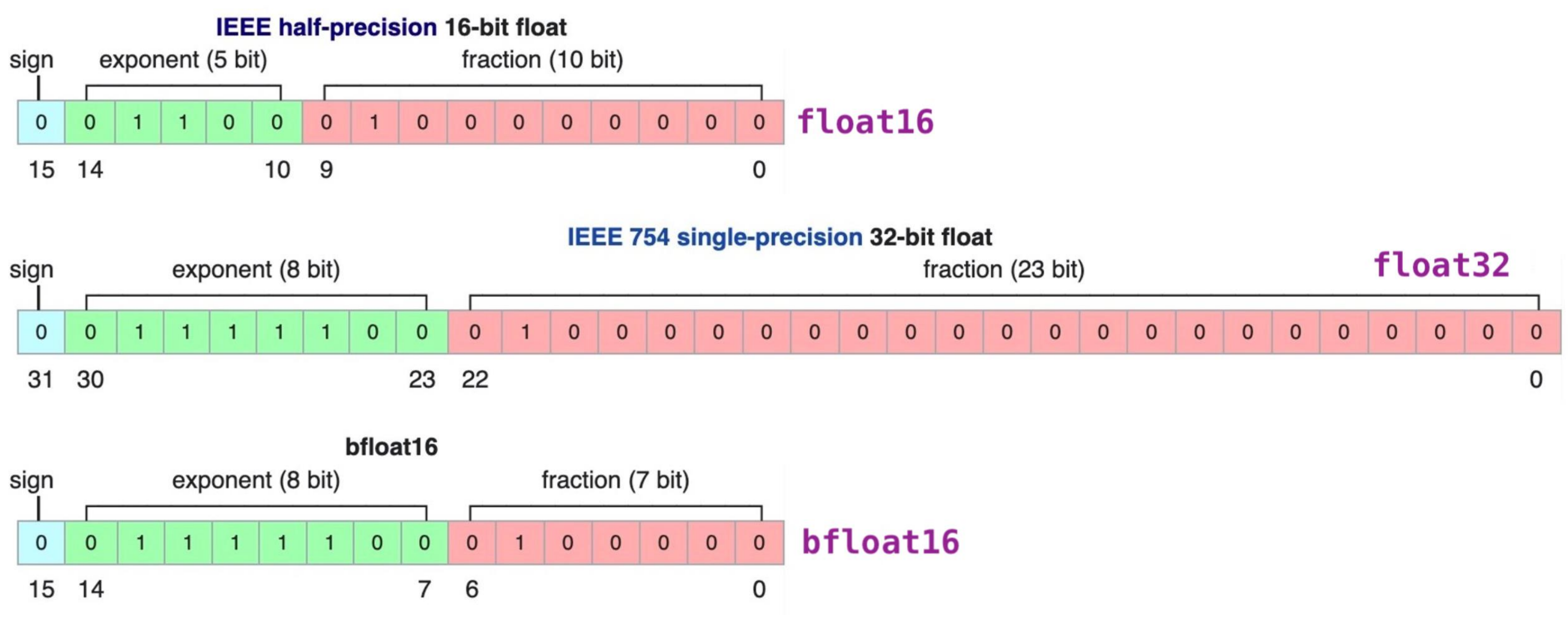


more exponent -> larger range -> harder to overflow

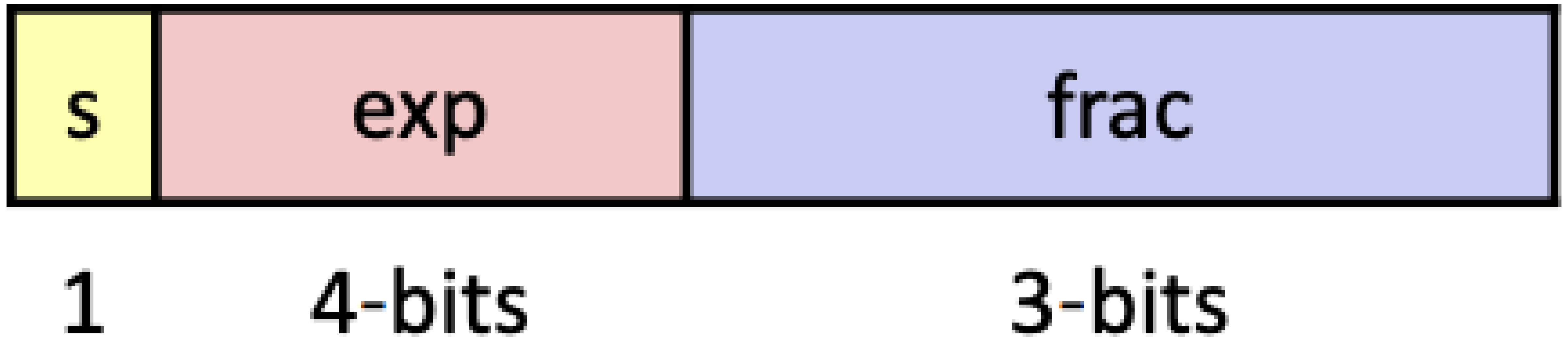
less fraction -> less precise

Why BF16 is better in ML/AI?

1. ML/AI is error-tolerant (why? what is not error-tolerant?). 7-bit precision is sufficient
2. Deep learning is easy to overflow
3. Conversion between fp32 and bf16 is less effortless



Examples in the final exam: FP8



Digital Representation of Data

- Representing **Character (char)** and **String**:
  - Letters, numerals, punctuations, etc.
  - A string is typically just a variable-sized array of char
  - C *char* is 1B; Java char is 2B; Python does not have a char type (use *str* or *bytes*)
  - American Standard Code for Information Interchange (*ASCII*) for encoding characters; initially 7-bit; later extended to 8-bit
    - Examples: 'A' is 61, 'a' is 97, '@' is 64, '!' is 33, etc.
  - *Unicode UTF-8* is now common, subsumes ASCII; 4B for ~1.1 million "code points" incl. many other language scripts, math symbols, 🤗, etc. 💻

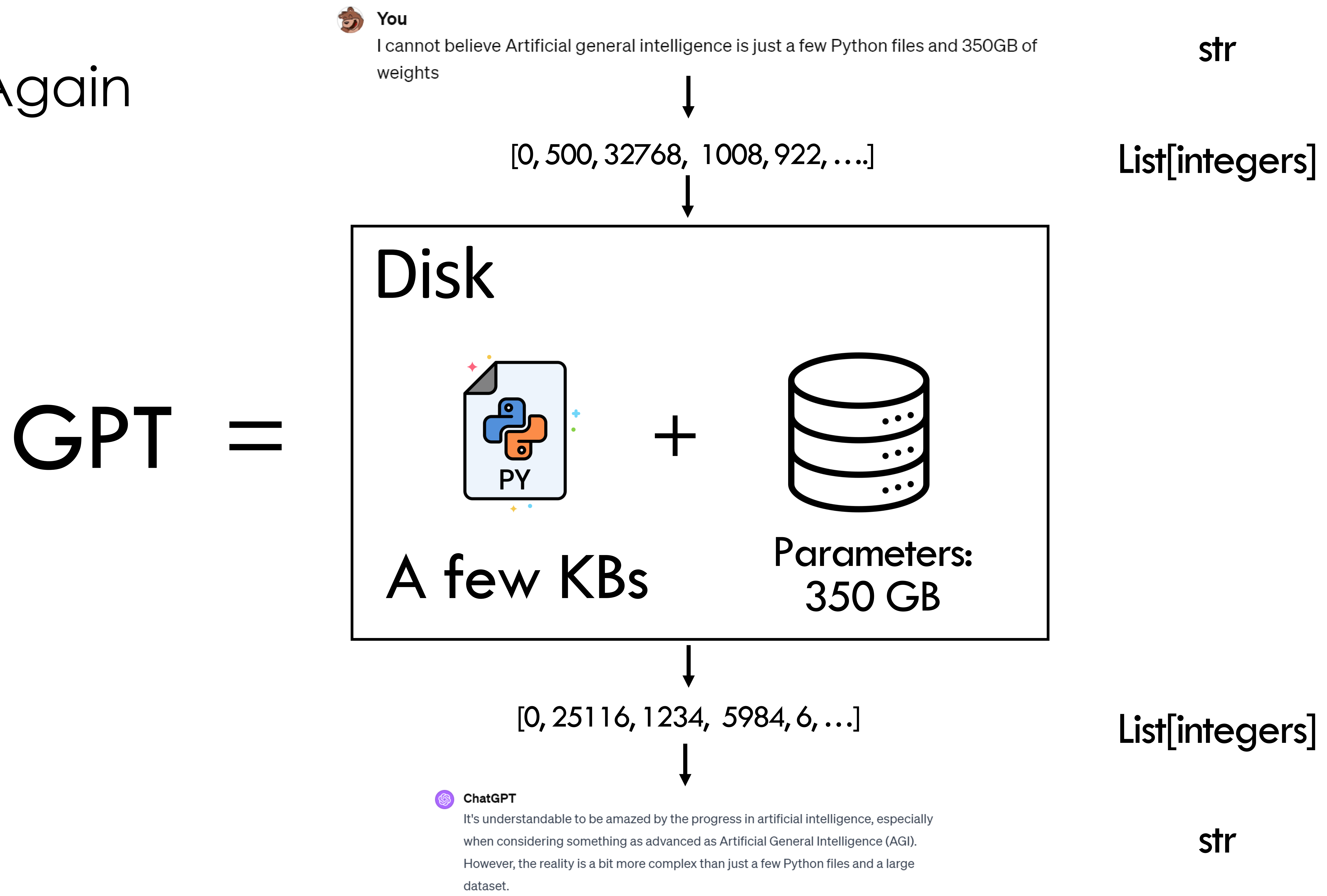
Digital Representation of Data

- All digital objects are *collections* of basic data types (bytes, integers, floats, and characters)
  - SQL dates/timestamp: string (w/ known format)
  - ML feature vector: *array* of floats (w/ known length)
  - Neural network weights: *set* of multi-dimensional *arrays* (matrices or tensors) of floats (w/ known dimensions)
  - Graph: an *abstract data type* (ADT) with *set* of vertices (say, integers) and *set* of edges (*pair* of integers)
  - Program in PL, SQL query: string (w/ grammar)
  - Other data structures or digital objects?

Foundation of Data Systems: where we are

- Computer Organization
  - Representation of Data
  - Processors, memory, storages
- Operating System Basics (next week)

GPT Again



You
I cannot believe Artificial general intelligence is just a few Python files and 350GB of weights
str
[0, 500, 32768, 1008, 922, ….]
List[integers]
GPT =
Disk
PY
A few KBs
+
Parameters:
350 GB
[0, 25116, 1234, 5984, 6, …]
List[integers]
ChatGPT
It's understandable to be amazed by the progress in artificial intelligence, especially when considering something as advanced as Artificial General Intelligence (AGI). However, the reality is a bit more complex than just a few Python files and a large dataset.
str

Disk

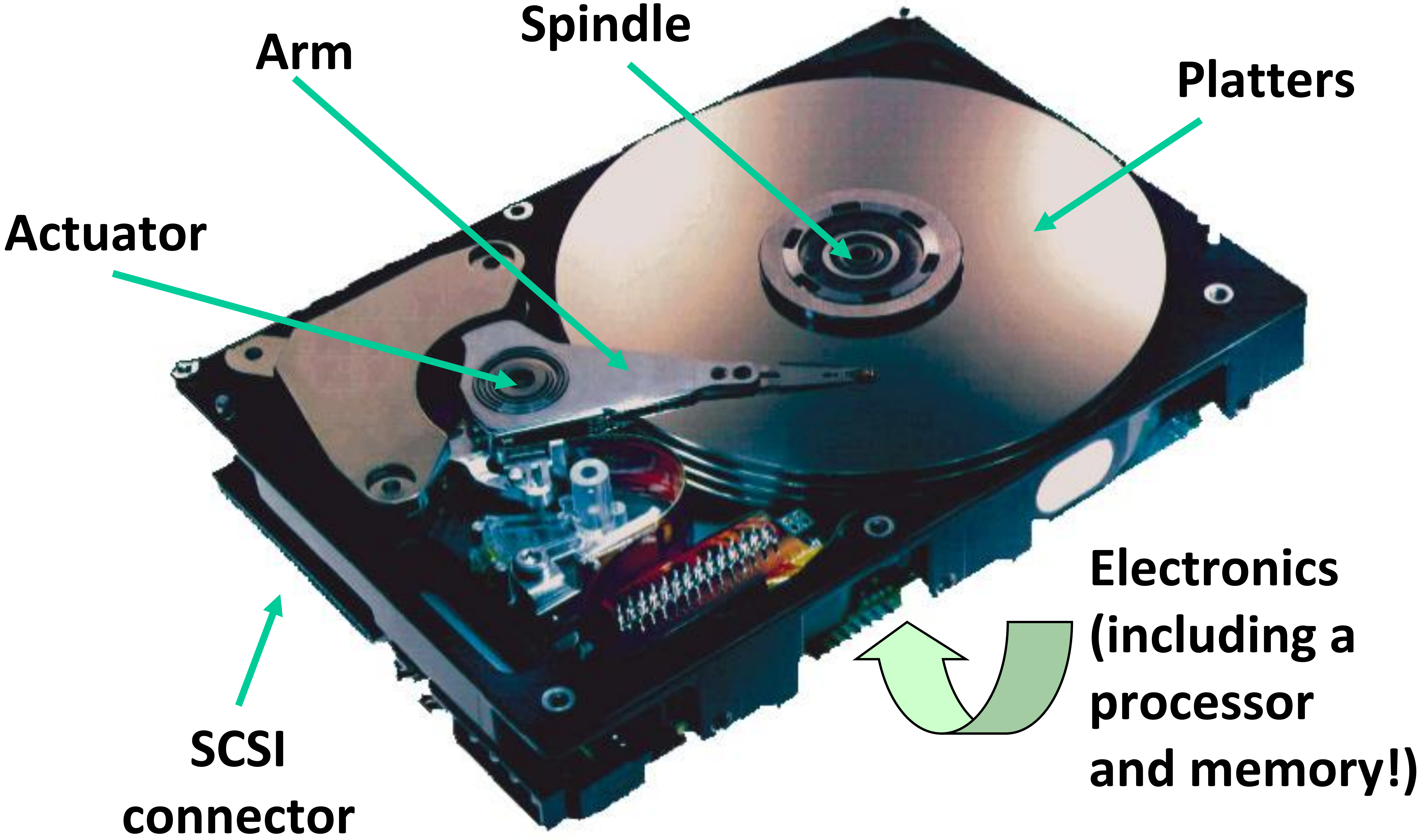
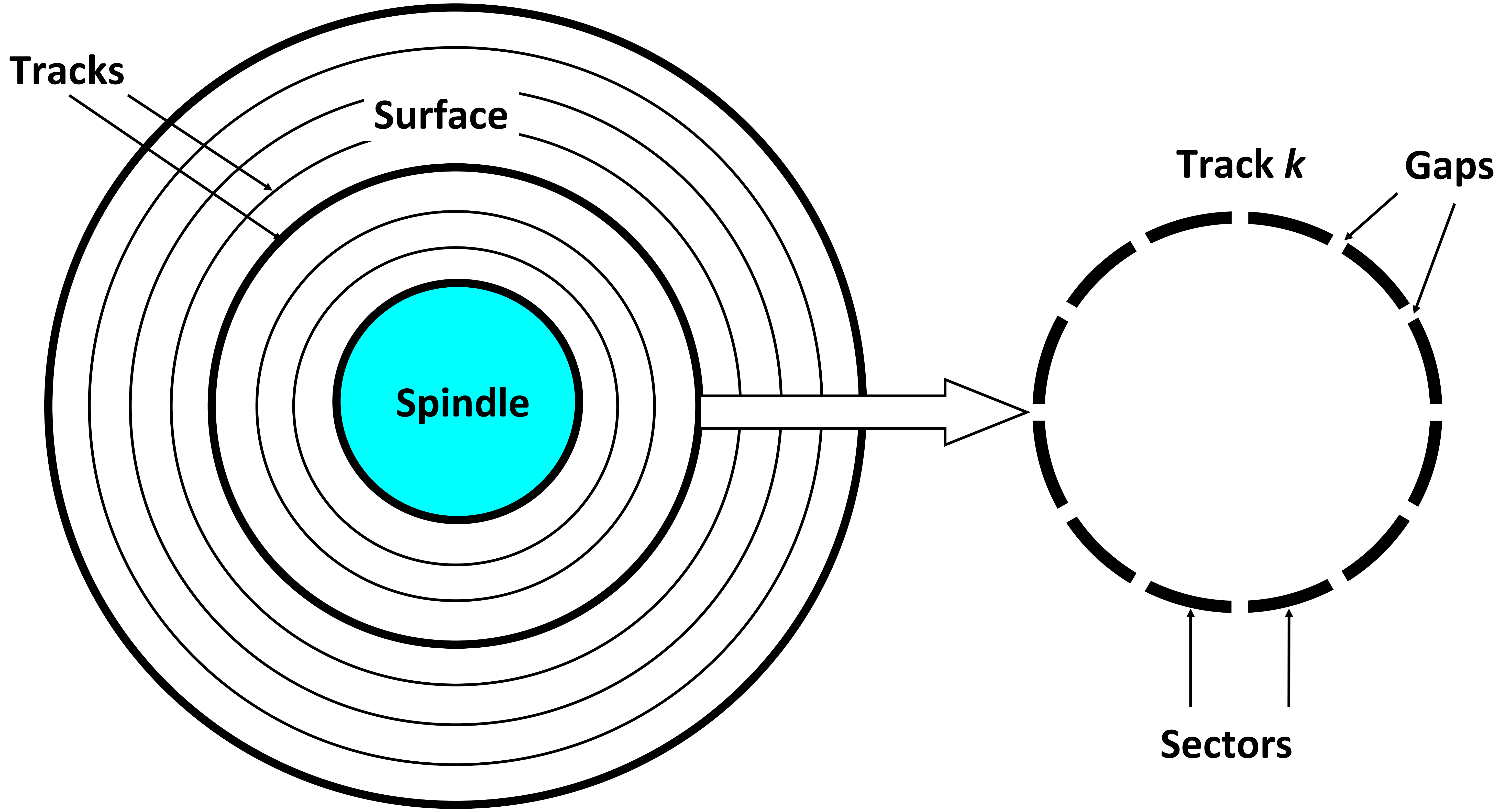


Image courtesy of Seagate Technology

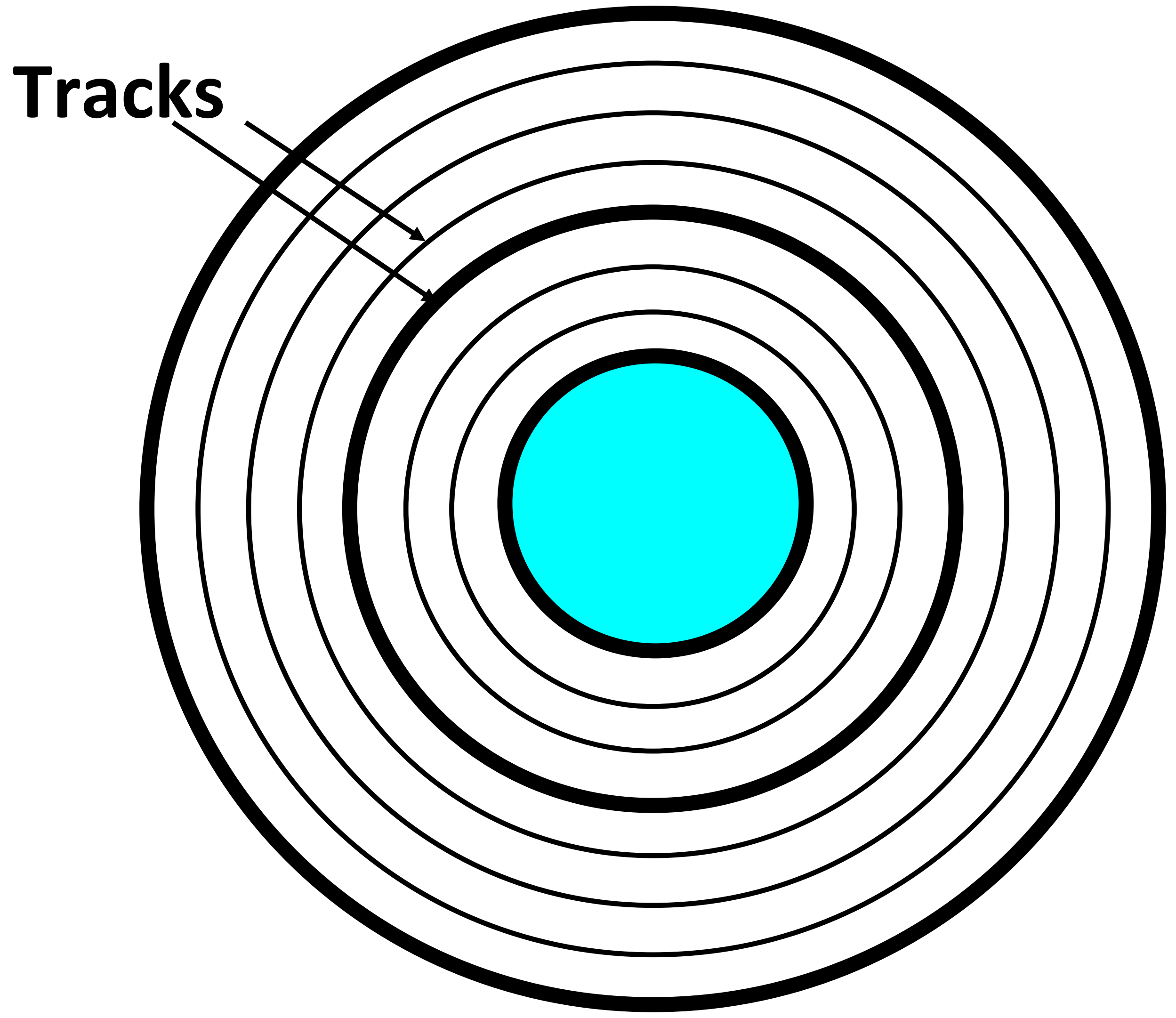
Disk Geometry

- Disks consist of platters, each with two surfaces.
- Each surface consists of concentric rings called tracks.
- Each track consists of sectors separated by gaps.



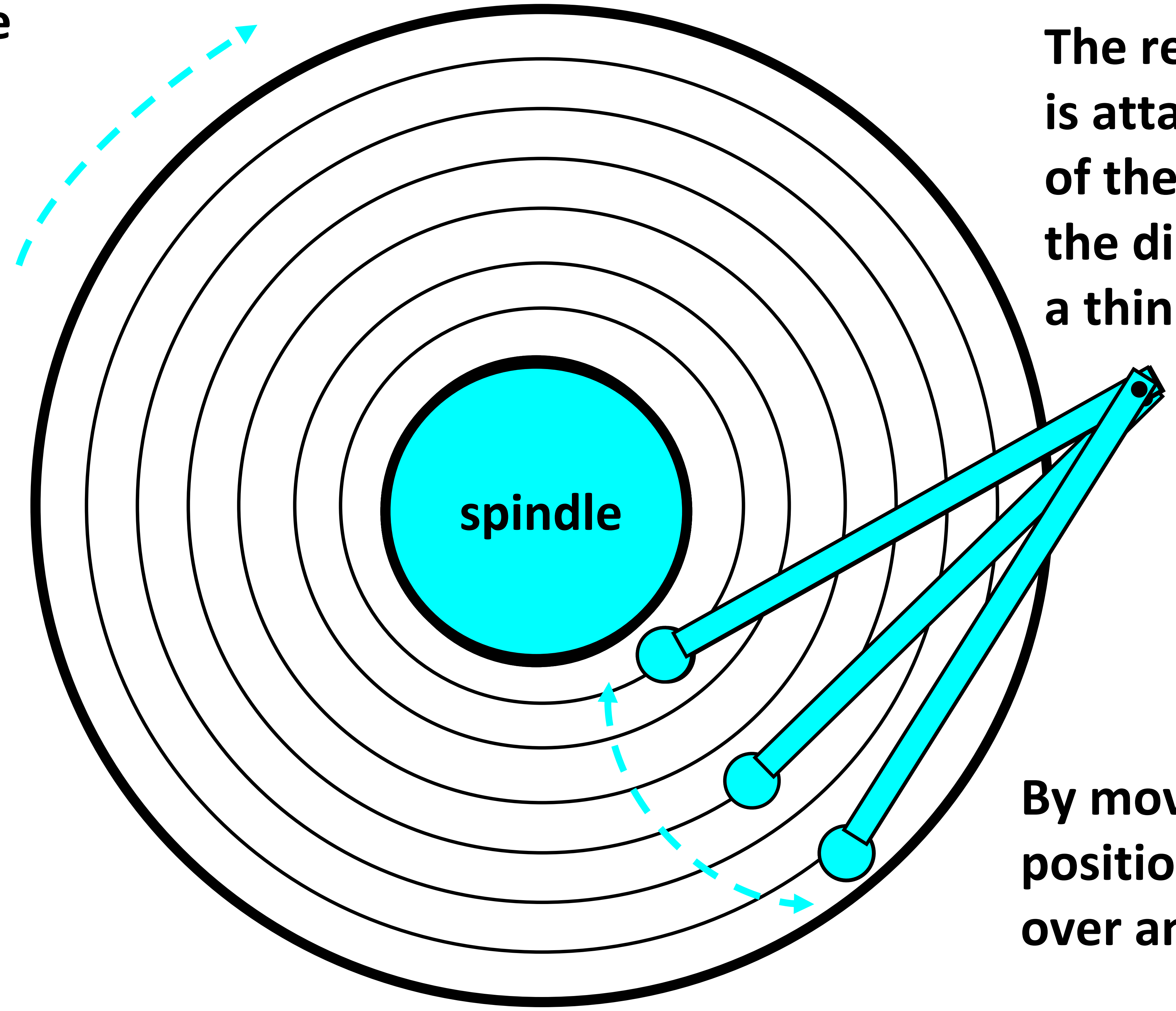
Disk Capacity

- Capacity: maximum number of bits that can be stored.
- Determined by these technology factors:
  - **Recording density (bits/in):** number of bits that can be squeezed into a 1 inch segment of a track.
  - **Track density (tracks/in):** number of tracks that can be squeezed into a 1 inch radial segment.
  - **Area density (bits/in^2):** product of recording and track density.



Disk Operation (Single-Platter View)

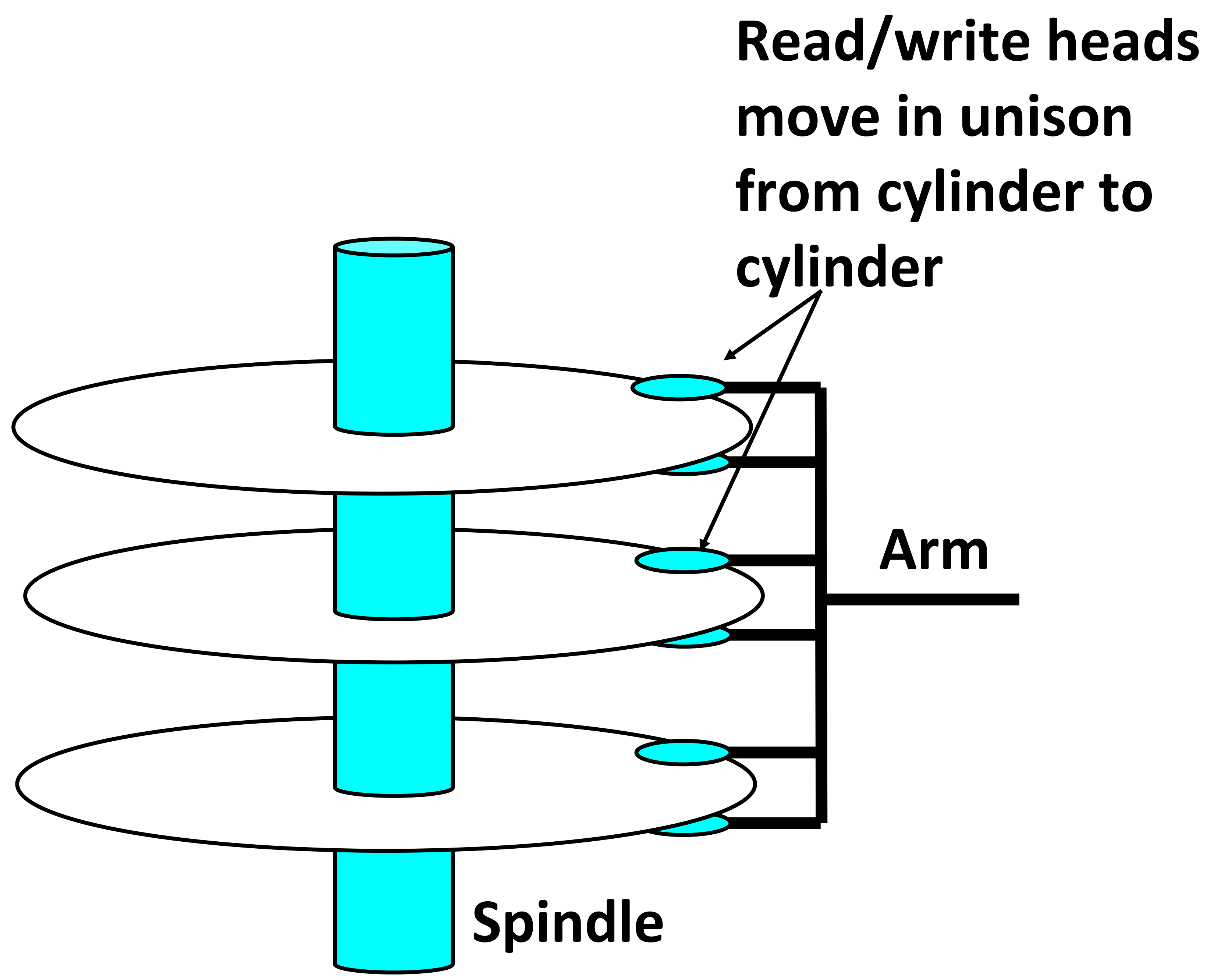
The disk surface spins at a fixed rotational rate. E.g. 7200 RPM



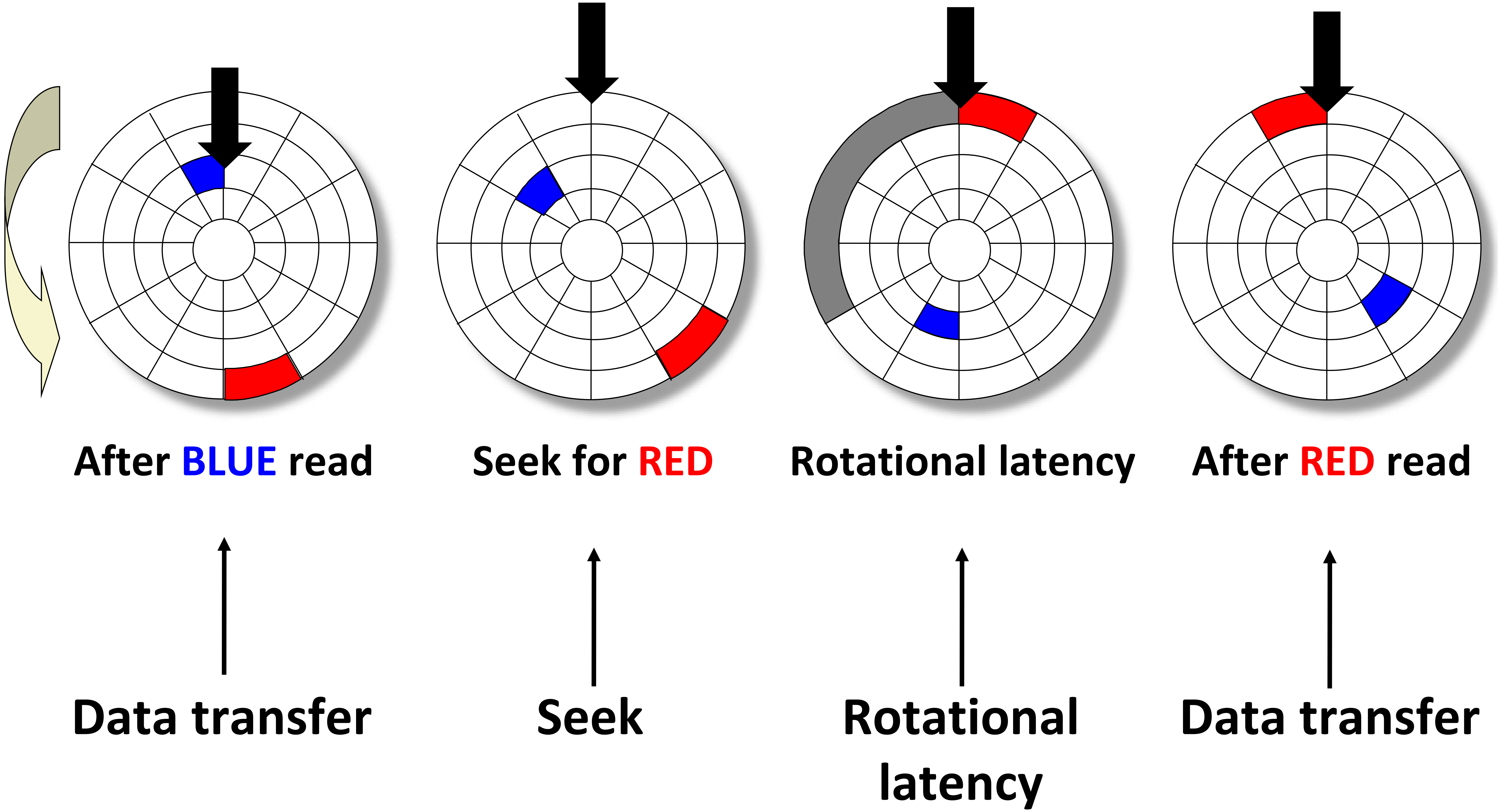
The read/write *head* is attached to the end of the *arm* and flies over the disk surface on a thin cushion of air.

By moving radially, the arm can position the read/write head over any track.

Disk Operation (Multi-Platter View)



Disk Access – Service Time Components



Disk Access Time

- Average time to access some target sector approximated by:
  - $T_{access} = T_{avg\ seek} + T_{avg\ rotation} + T_{avg\ transfer}$
- Seek time ($T_{avg\ seek}$)
  - Time to position heads over cylinder containing target sector.
  - Typical $T_{avg\ seek}$ is 3—9 ms
- Rotational latency ($T_{avg\ rotation}$)
  - Time waiting for first bit of target sector to pass under r/w head.
  - $T_{avg\ rotation}$ = 1/2 x 1/RPMs x 60 sec/1 min
  - Typical rotational rate = 7,200 RPMs
- Transfer time ($T_{avg\ transfer}$)
  - Time to read the bits in the target sector.
  - $T_{avg\ transfer}$ = 1/RPM x 1/(avg # sectors/track) x 60 secs/1 min

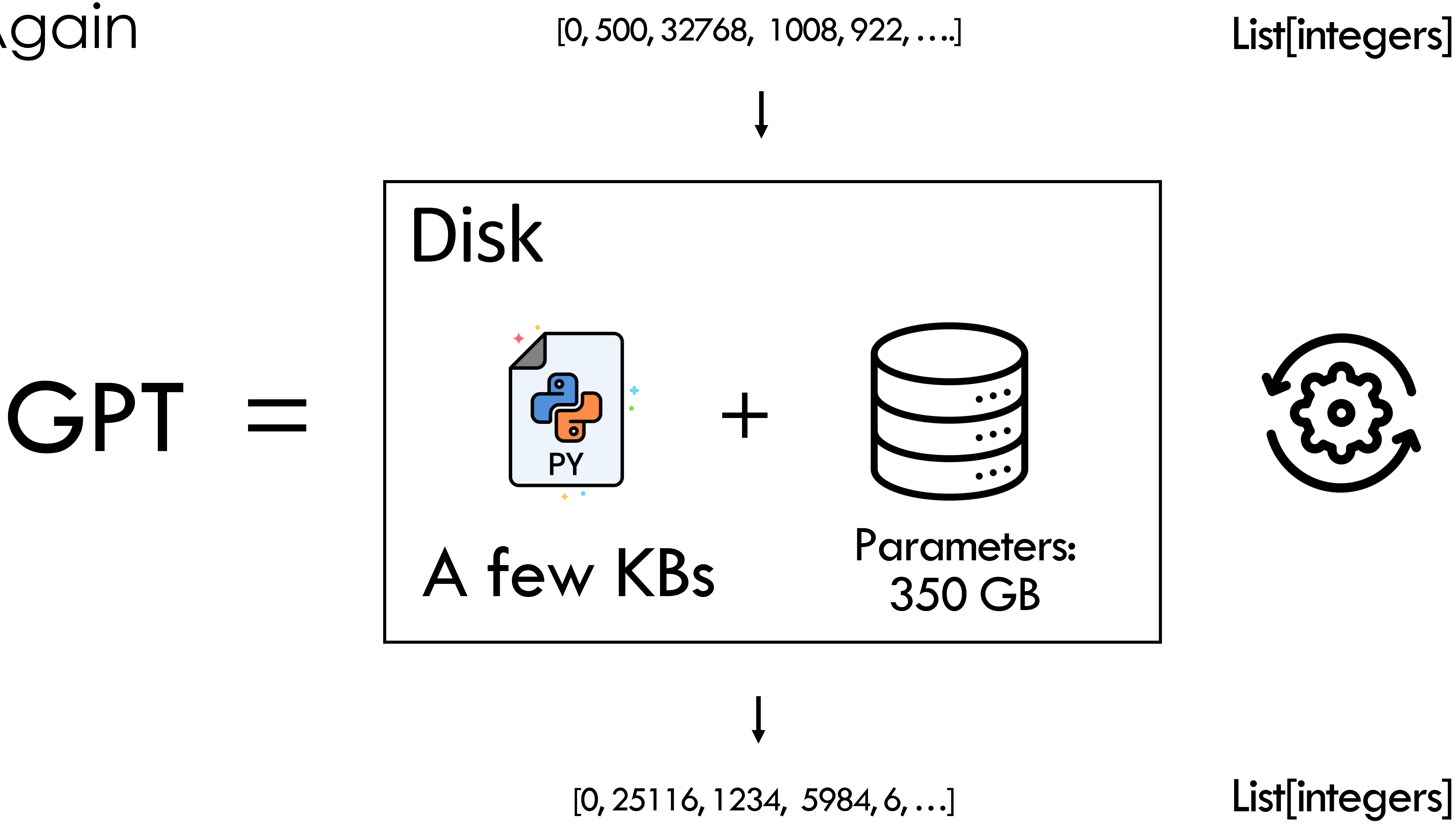
time for one rotation (in minutes)   fraction of a rotation to be read

Disk Access Time Example

- Given:
  - Rotational rate = 7,200 RPM
  - Average seek time = 9 ms
  - Avg # sectors/track = 400
- Derived:
  - $T_{avg\ rotation}$ = 1/2 x (60 secs/7200 RPM) x 1000 ms/sec = 4 ms
  - $T_{avg\ transfer}$ = 60/7200 x 1/400 x 1000 ms/sec = 0.02 ms
  - T_{access} = 9 ms + 4 ms + 0.02 ms
- Important points:
  - Access time dominated by seek time and rotational latency.
  - First bit in a sector is the most expensive, the rest are free.

HDD reading speed (7200 RPM): 80 – 160 MB/s

GPT Again



[0, 500, 32768, 1008, 922, ….]
List[integers]
GPT =
Disk
PY
A few KBs
+
Parameters:
350 GB
[0, 25116, 1234, 5984, 6, …]
List[integers]

Basics of Processors

- Processor: Hardware to orchestrate and *execute instructions* to *manipulate data* as specified by a program
  - Examples: CPU, GPU, FPGA, TPU, embedded, etc.
- ISA (Instruction Set Architecture):
  - The vocabulary of commands of a processor

Program in PL

↓ *Compile/Interpret*

Program in Assembly Language

↓ *Assemble*

Machine code tied to ISA

↓

Run on processor

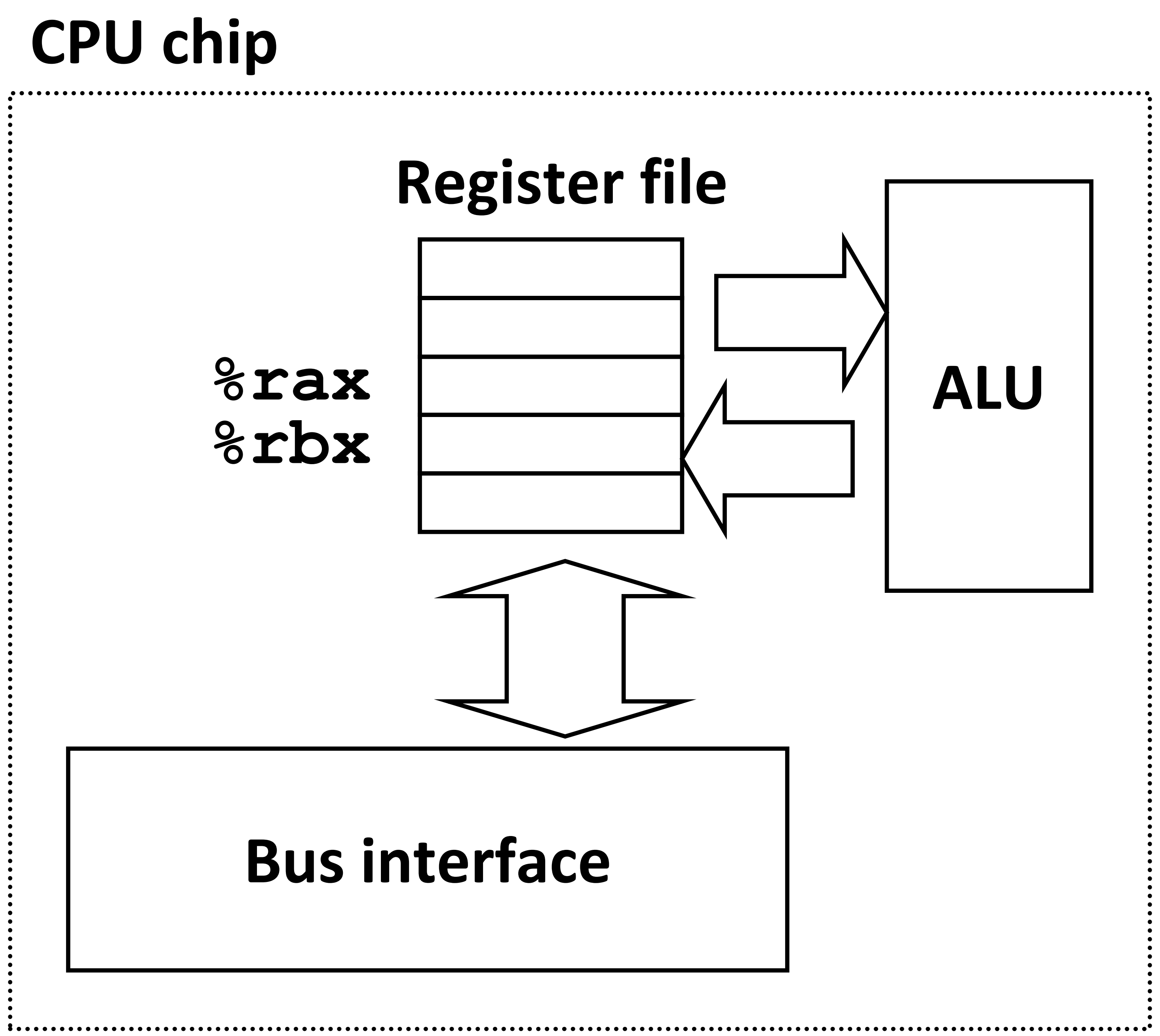
```
80483b4:       55                      push   %ebp
80483b5:       89 e5                   mov    %esp,%ebp
80483b7:       83 e4 f0                and    $0xfffffff0,%esp
80483ba:       83 ec 20                sub    $0x20,%esp
80483bd:       c7 44 24 1c 00 00 00    movl   $0x0,0x1c(%esp)
80483c4:       00
80483c5:       eb 11                   jmp    80483d8 <main+0x24>
80483c7:       c7 04 24 b0 84 04 08    movl   $0x80484b0,(%esp)
80483ce:       e8 1d ff ff ff          call   80482f0 <puts@plt>
80483d3:       83 44 24 1c 01          addl   $0x1,0x1c(%esp)
80483d8:       83 7c 24 1c 09          cmpl   $0x9,0x1c(%esp)
80483dd:       7e e8                   jle    80483c7 <main+0x13>
80483df:       b8 00 00 00 00          mov    $0x0,%eax
80483e4:       c9                      leave
80483e5:       c3                      ret
80483e6:       90                      nop
80483e7:       90                      nop
80483e8:       90                      nop
80483e9:       90                      nop
80483ea:       90                      nop
```

Basics of Processors

Q: *How does a processor execute machine code?*

- Most common approach: load-store architecture
- Registers: Tiny local memory ("scratch space") on proc. into which instructions and data are copied
- ISA specifies bit length/format of machine code commands
- ISA has several commands to manipulate register contents

Instruction



Register names

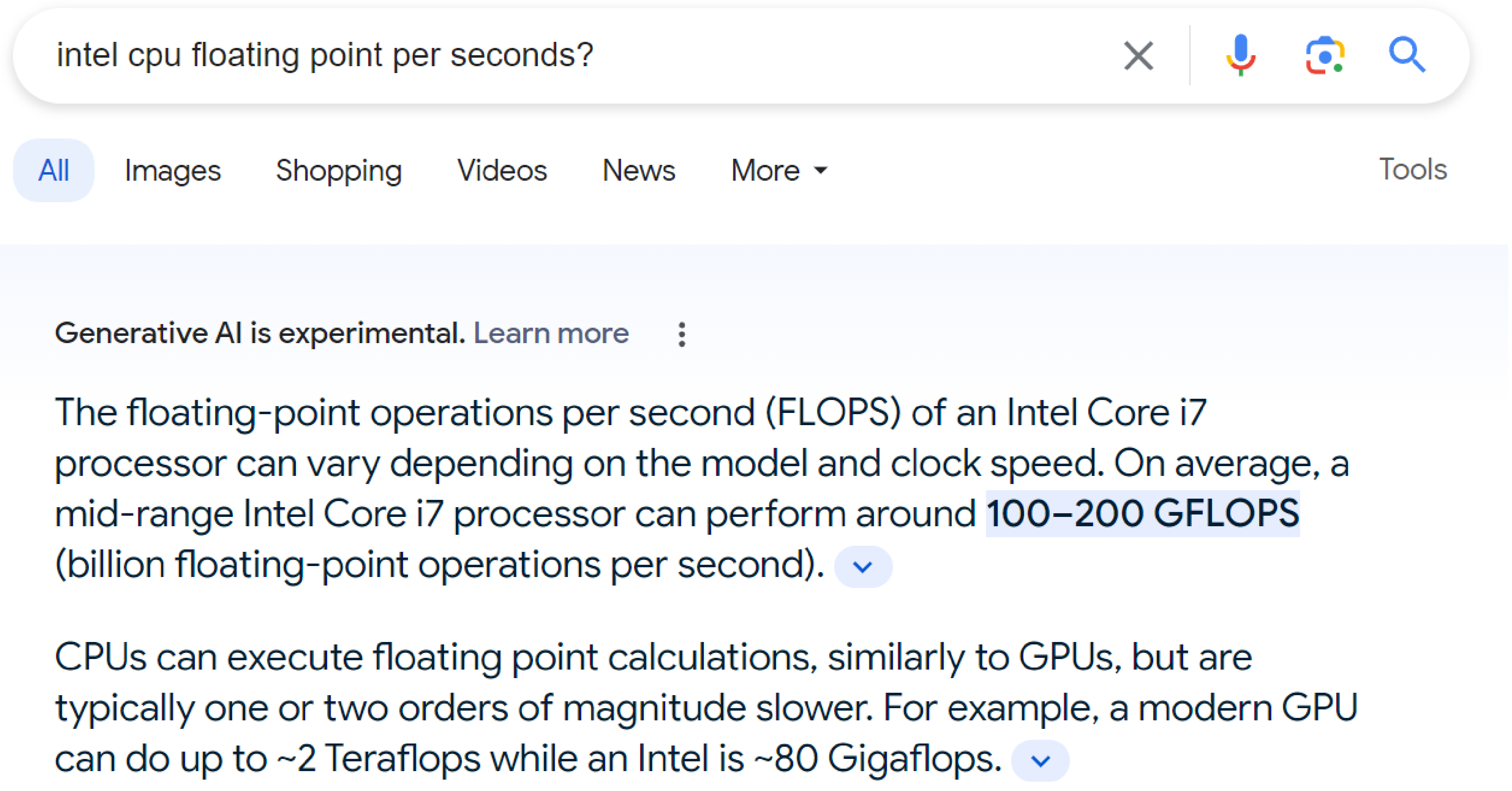
```
addq    %rbx,  %rax
```

is

```
rax += rbx
```

How Fast is Processor

- Instruction / second: number of instructions a processor can do
- Data science: We care more about computation on floating point numbers
- FLOPS: number of floating point operations a process can do


intel cpu floating point per seconds?

All Images Shopping Videos News More Tools

Generative AI is experimental. Learn more

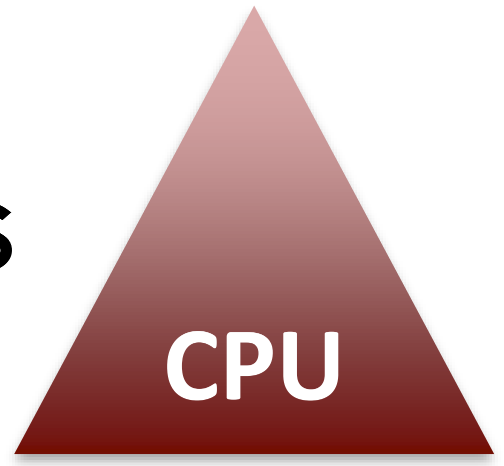
The floating-point operations per second (FLOPS) of an Intel Core i7 processor can vary depending on the model and clock speed. On average, a mid-range Intel Core i7 processor can perform around 100–200 GFLOPS (billion floating-point operations per second).

CPUs can execute floating point calculations, similarly to GPUs, but are typically one or two orders of magnitude slower. For example, a modern GPU can do up to ~2 Teraflops while an Intel is ~80 Gigaflops.

Form Factor	H100 SXM
FP64	34 teraFLOPS
FP64 Tensor Core	67 teraFLOPS
FP32	67 teraFLOPS
TF32 Tensor Core	989 teraFLOPS[2]
BFLOAT16 Tensor Core	1,979 teraFLOPS[2]
FP16 Tensor Core	1,979 teraFLOPS[2]
FP8 Tensor Core	3,958 teraFLOPS[2]

Problem?

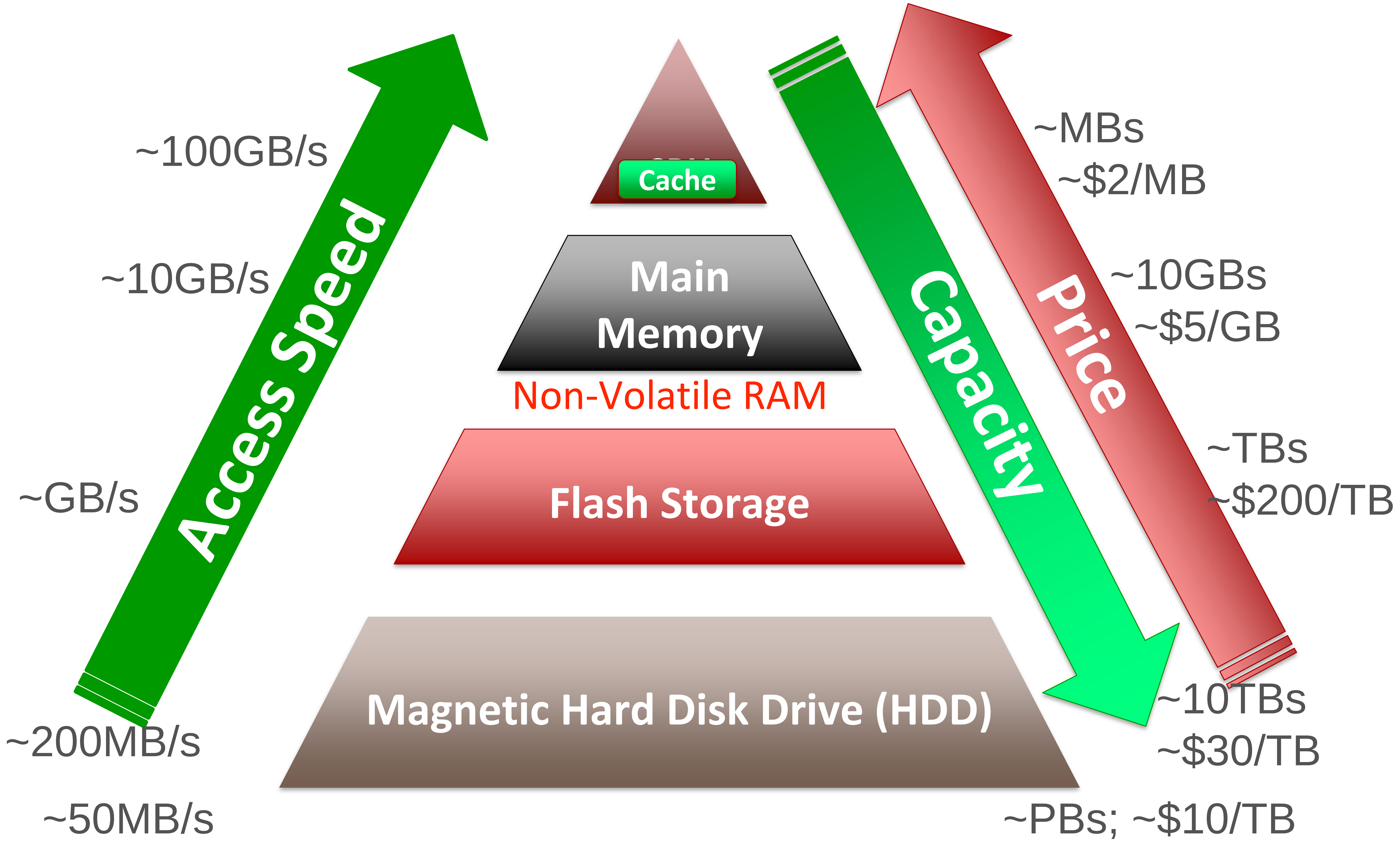
100 GFLOPs/s



1. Assume we use 0.5s to perform 50 FLOPs
2. We need to read 50x2=100 GB in the rest of 0.5s to keep the CPU busy
3. We need the CPU to read at a speed of 100GB / 0.5s = 200 GB/s

80 – 160 MB/s

Memory/Storage Hierarchy

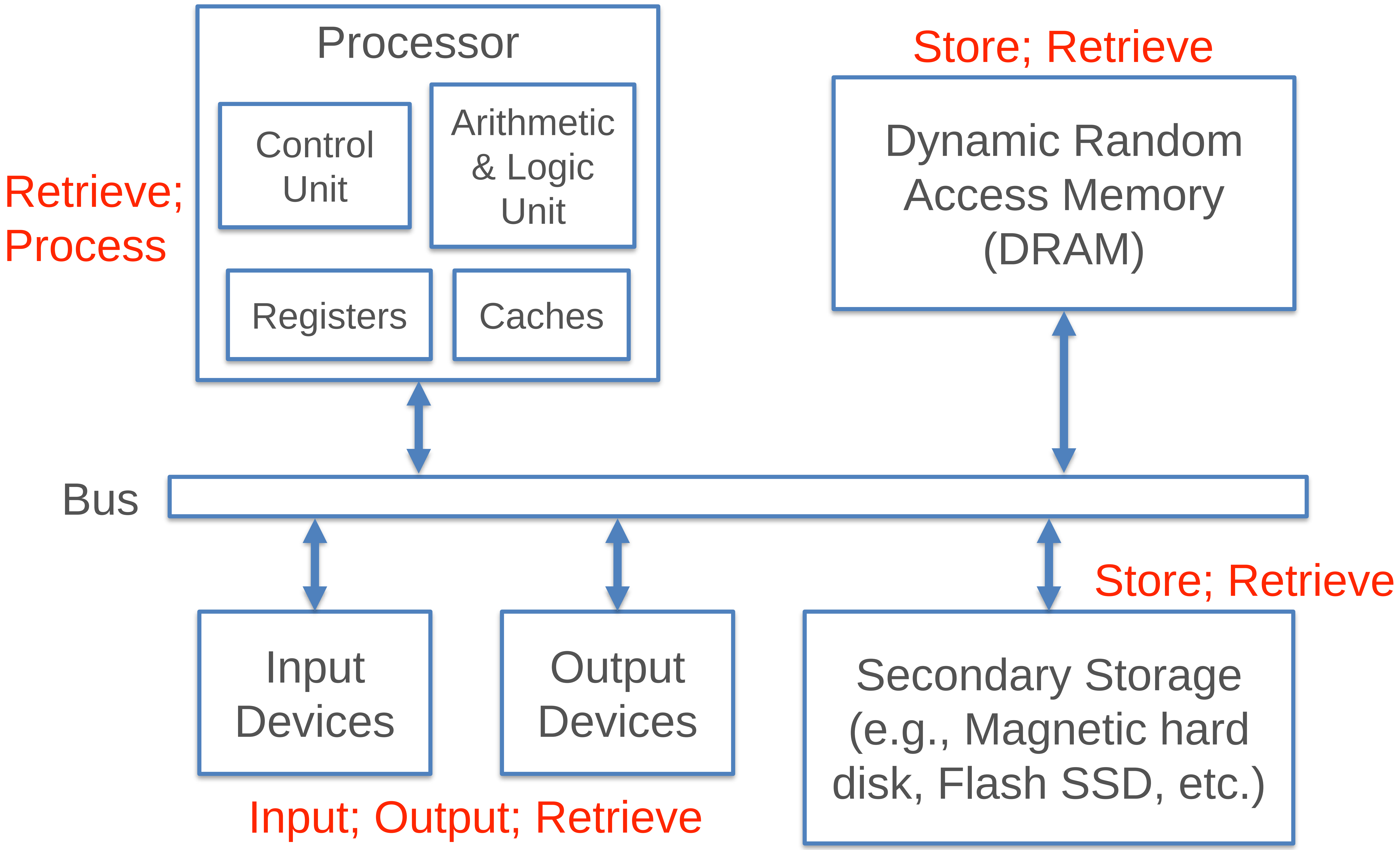


Writing & Reading Memory Instructions

- Write
  - Transfer data from memory to CPU
    movq %rax, %rsp
  - "Store" operation

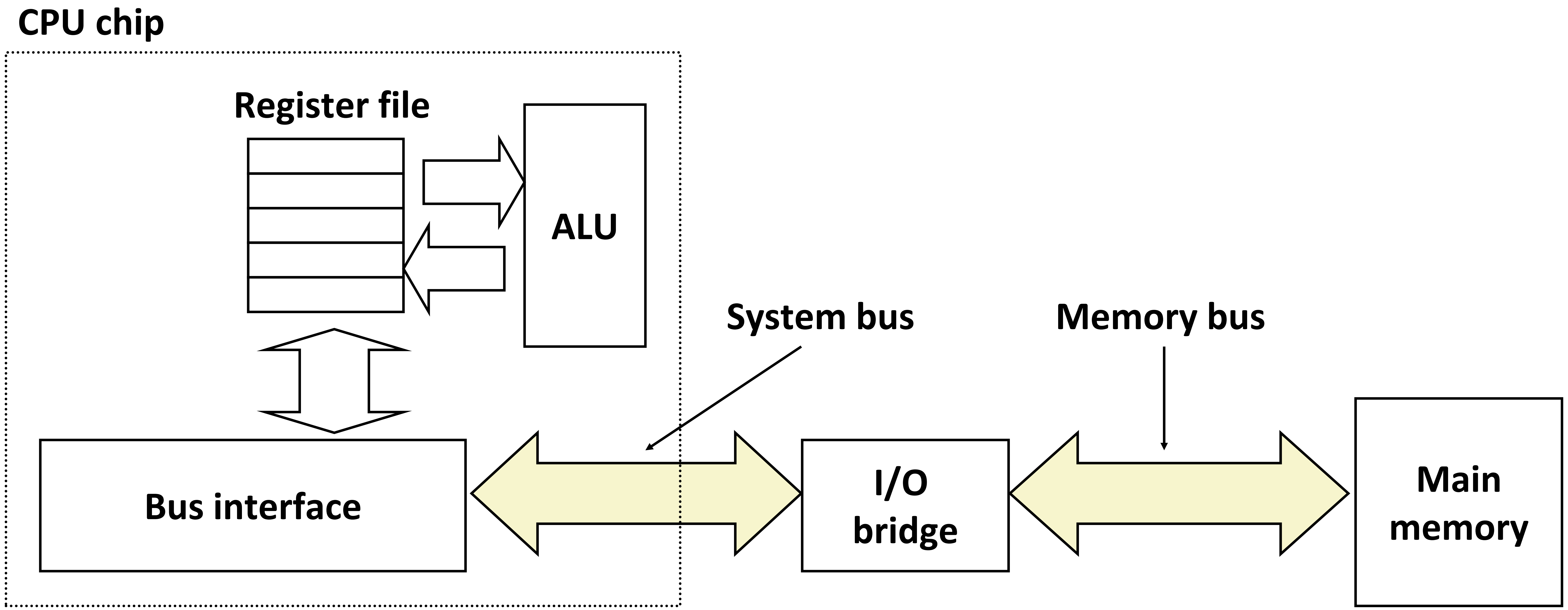
- Read
  - Transfer data from CPU to memory
    movq %rsp, %rax
  - "Load" operation

Abstract Computer Parts and Data

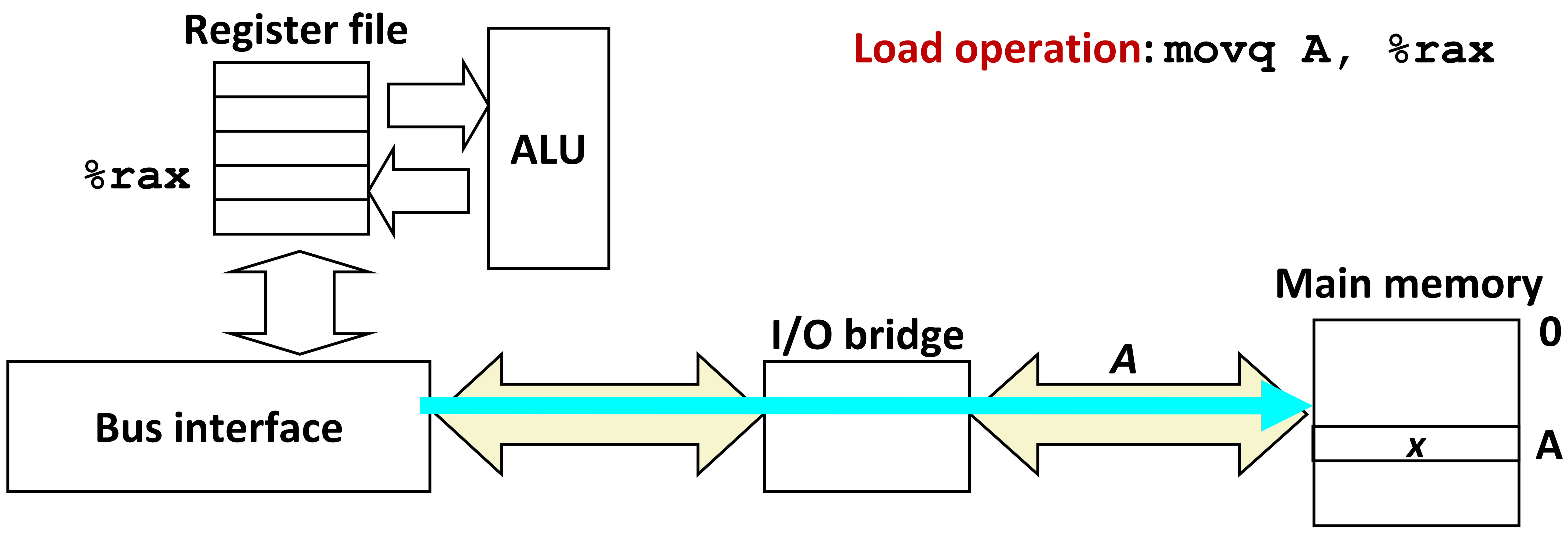


Bus Structure Connecting CPU and Memory

- A bus is a collection of parallel wires that carry address, data, and control signals.
- Buses are typically shared by multiple devices.

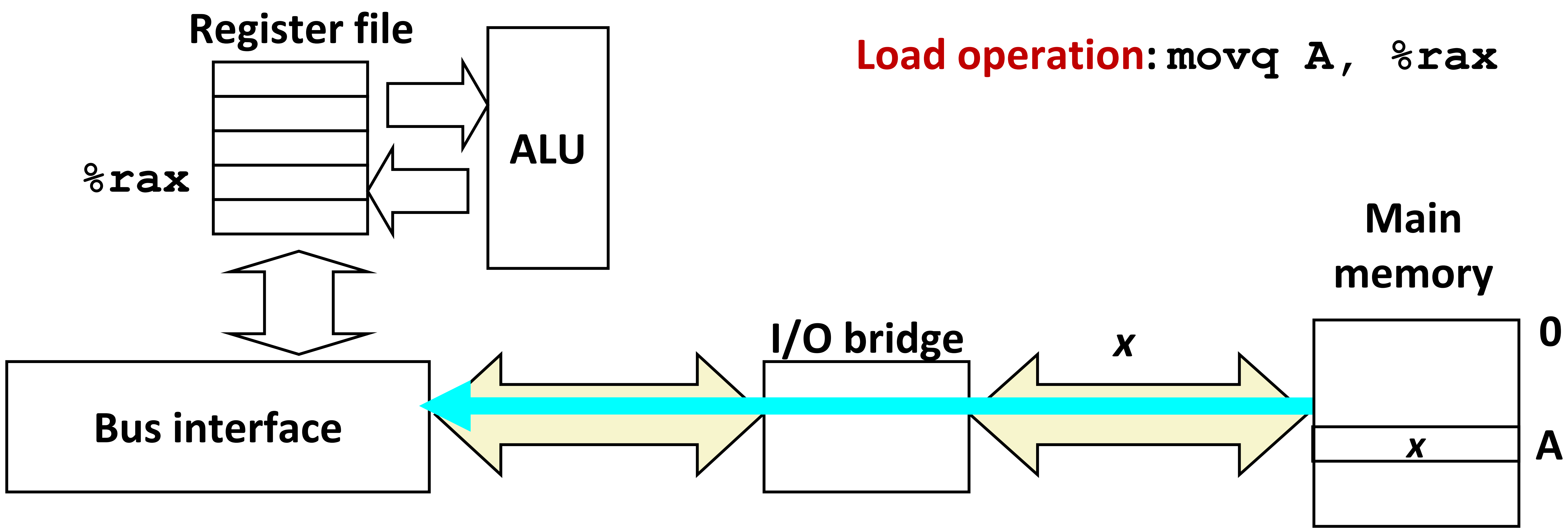


Memory Read Transaction (1)



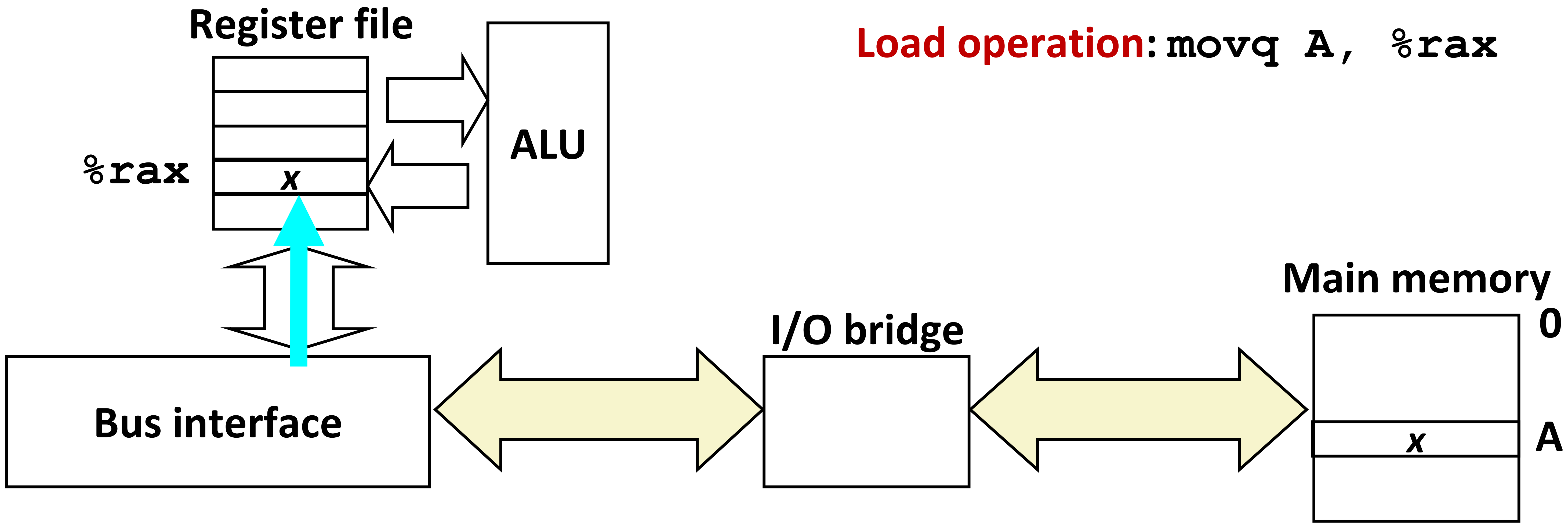
- CPU places address A on the memory bus.

Memory Read Transaction (2)



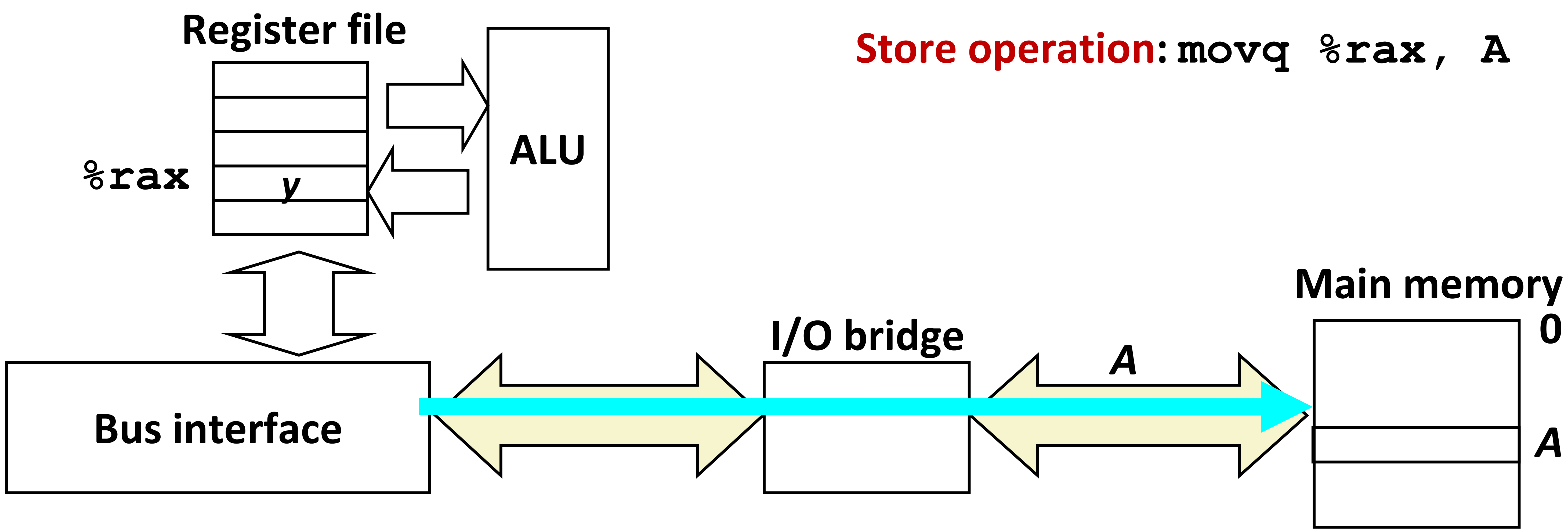
- Main memory reads A from the memory bus, retrieves word x, and places it on the bus.

Memory Read Transaction (3)



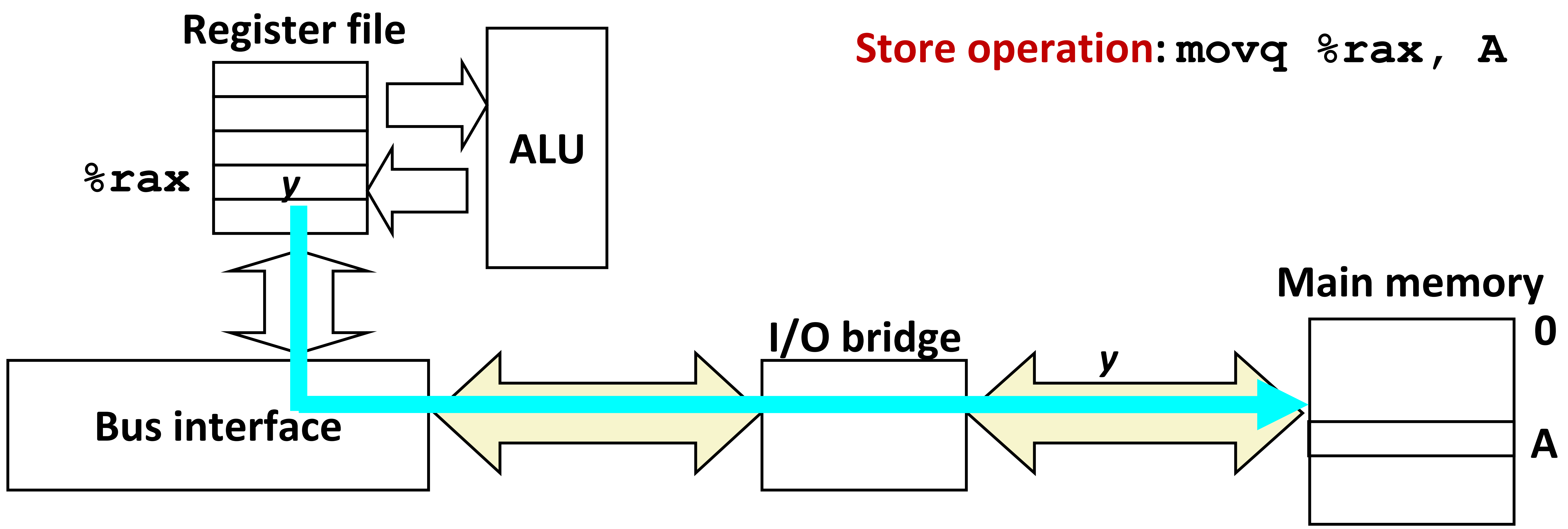
- CPU reads word x from the bus and copies it into register `%rax`.

Memory Write Transaction (1)



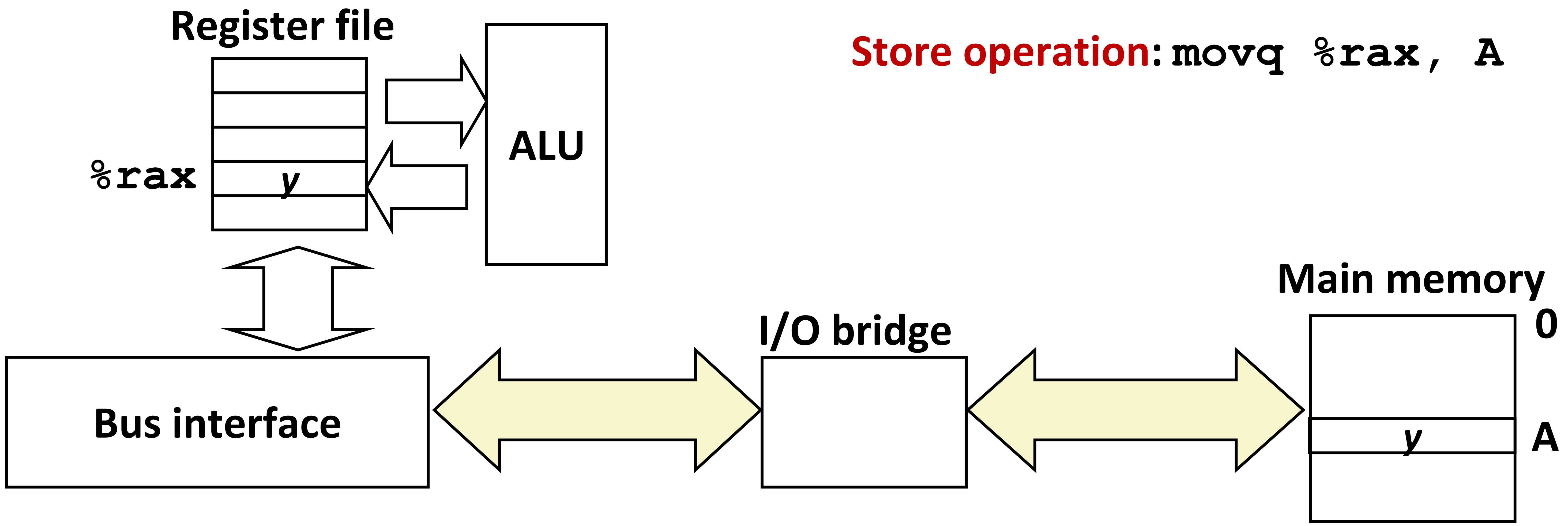
- CPU places address A on bus. Main memory reads it and waits for the corresponding data word to arrive.

Memory Write Transaction (2)



- CPU places data word y on the bus.

Memory Write Transaction (3)



- Main memory reads data word y from the bus and stores it at address A.

Basics of Processors

Q: *How does a processor execute machine code?*

- Types of ISA commands to manipulate register contents:
  - Memory access: load (copy bytes from a DRAM address to register); store (reverse); put constant
  - Arithmetic & logic on data items in registers: add/multiply/etc.; bitwise ops; compare, etc.; handled by ALU
  - Control flow (branch, call, etc.); handled by CU
- Caches: Small local memory to buffer instructions/data

If interested in more details: https://www.youtube.com/watch?v=cNN_tTXABUA

You

I cannot believe Artificial general intelligence is just a few Python files and 350GB of weights

What is GPT doing?

[0, 500, 32768, 1008, 922, ….]

List[integers]

↓

GPT =

Disk

PY

A few KBs

+

Parameters: 350 GB

↓

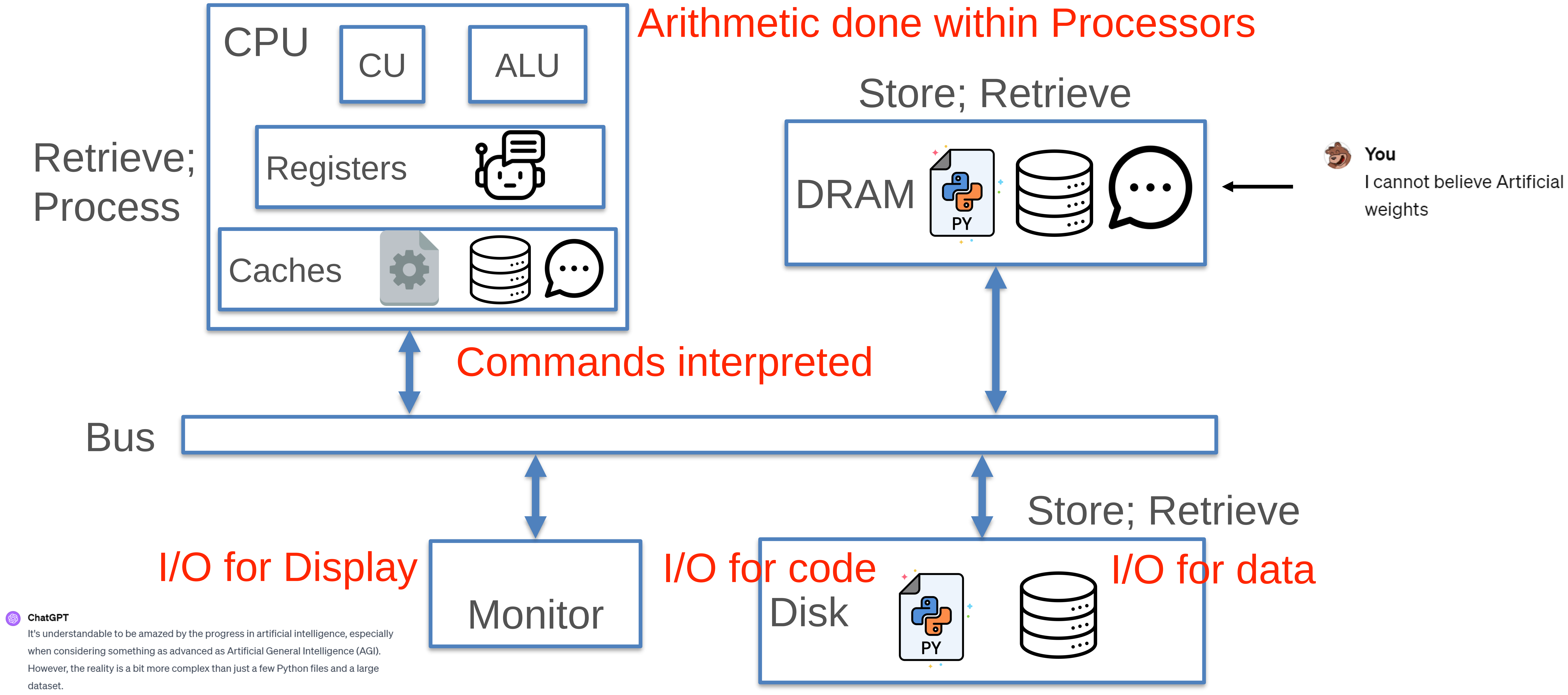
[0, 25116, 1234, 5984, 6, …]

List[integers]

ChatGPT

It's understandable to be amazed by the progress in artificial intelligence, especially when considering something as advanced as Artificial General Intelligence (AGI). However, the reality is a bit more complex than just a few Python files and a large dataset.

Example



CPU
CU
ALU
Arithmetic done within Processors
Retrieve;
Process
Registers
Caches
Store; Retrieve
DRAM
PY
You
I cannot believe Artificial
weights
Commands interpreted
Bus
Store; Retrieve
I/O for Display
Monitor
I/O for code
Disk
PY
I/O for data
ChatGPT
It's understandable to be amazed by the progress in artificial intelligence, especially when considering something as advanced as Artificial General Intelligence (AGI). However, the reality is a bit more complex than just a few Python files and a large dataset.

Foundation of Data Systems: where we are

- Computer Organization
  - Representation of Data
  - Processors, memory, storages
- **Operating System Basics (next week)**