



<https://hao-ai-lab.github.io/dsc204a-f25/>

# DSC 204A: Scalable Data Systems

## Fall 2025

---

Staff

Instructor: Hao Zhang

TAs: Mingjia Huo, Yuxuan Zhang



[@haozhangml](https://twitter.com/haozhangml)



[@haoailab](https://twitter.com/haoailab)



[haozhang@ucsd.edu](mailto:haozhang@ucsd.edu)

# Logistics

- Please finish Beginning of Quarter survey by 10/10
  - If  $\geq 80\%$  finish it, all of you get 1 point
  - TA will update completion percentage
- Enrollment approval and waitlist:
  - I approve anyone in EASy
  - If you are on enrollment list, wait until end of this week
    - Normally we will have new slots because some students will drop after seeing PA1
- PA1 will be posted by EoW

# Where We Are

Machine Learning Systems

Big Data

Cloud

Foundations of Data Systems

1980 - 2000

# Logistics

- Please finish Beginning of Quarter survey by 10/10
  - If  $\geq 80\%$  finish it, all of you get 1 point
  - TA will update completion percentage
- Enrollment approval and waitlist:
  - I approve anyone in EASy
  - If you are on enrollment list, wait until end of this week
    - Normally we will have new slots because some students will drop after seeing PA1
- PA1 will be posted by EoW

# Foundation of Data Systems

- Computer Organization
  - Representation of data
  - processors, memory, storage
- OS basics
  - Process, scheduling
  - Memory

Q: What is a computer?

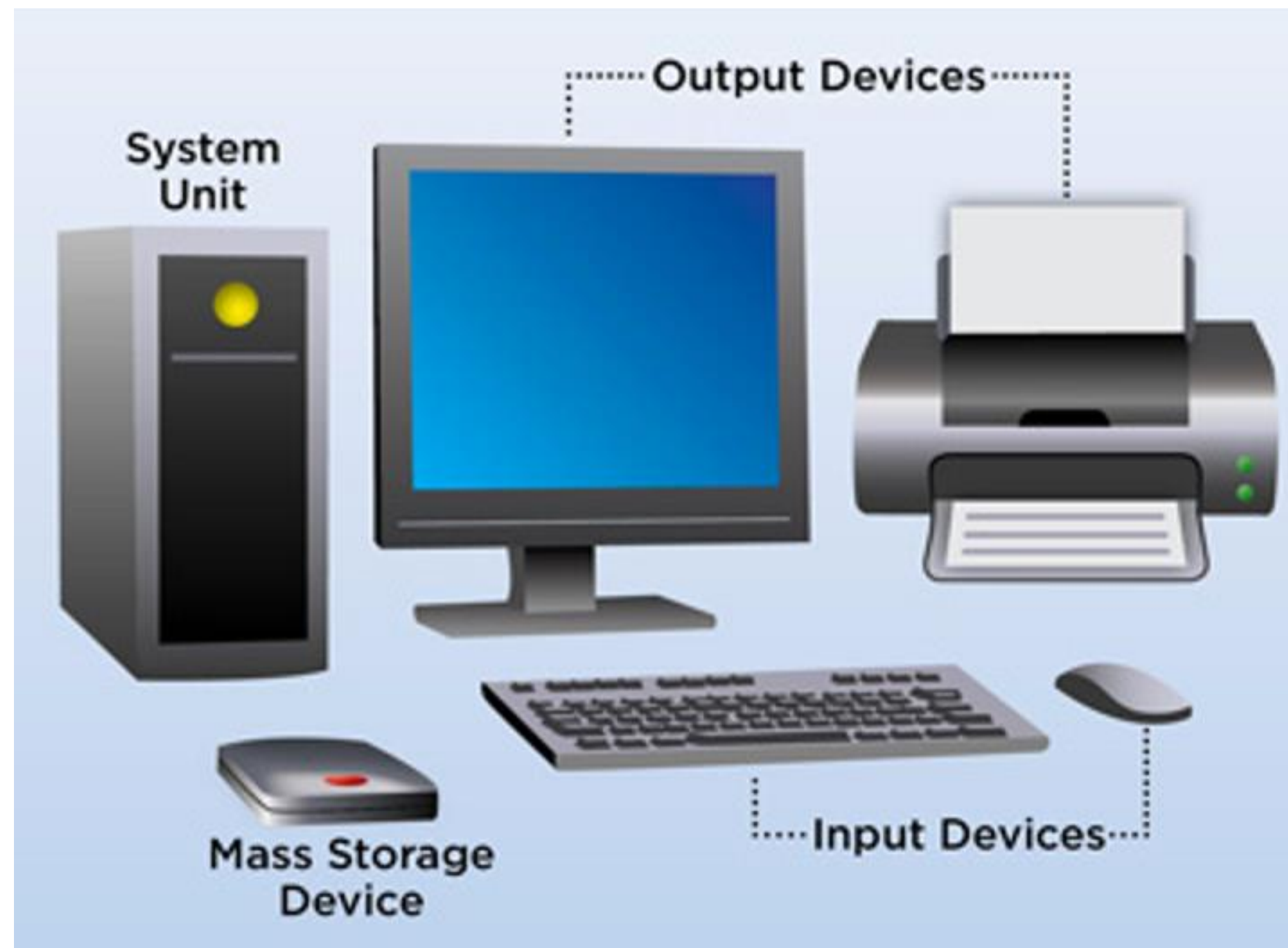
# What is a computer?



Peter Naur

A **programmable** electronic device that can **store, retrieve, and process** digital **data**.

# Basics of Computer Organization

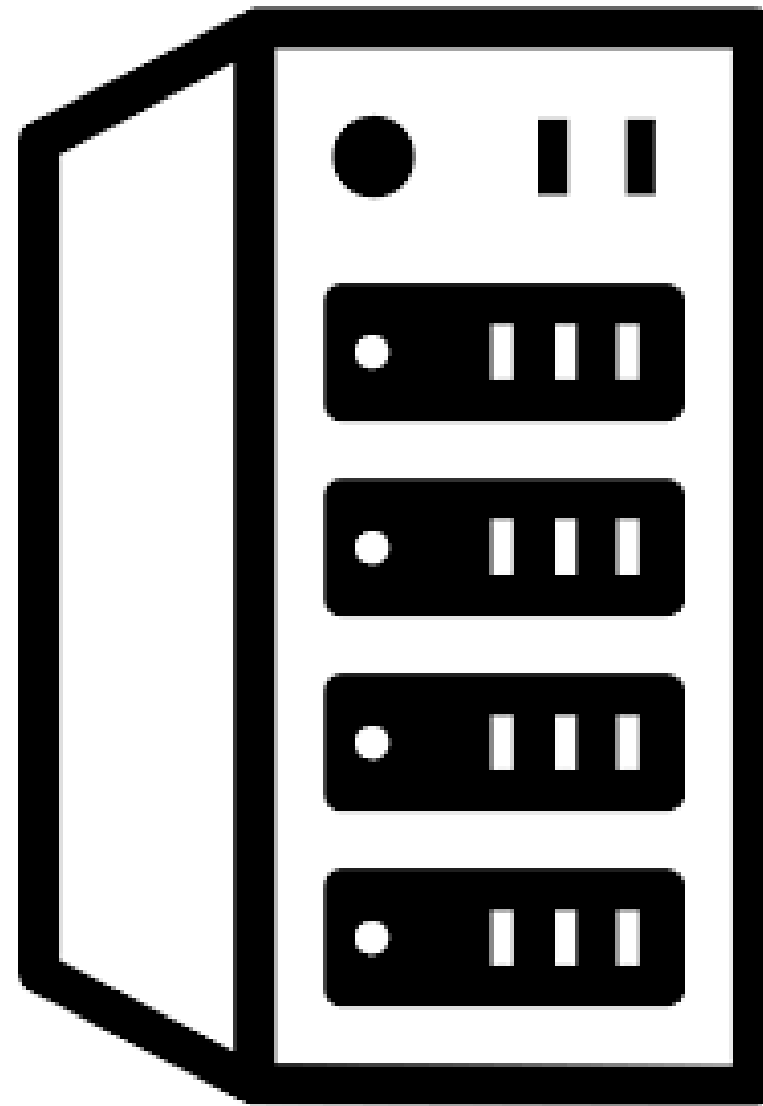


- Hardware: The electronic machinery (wires, circuits, transistors, capacitors, devices, etc.)
- Software: Programs (instructions) and data

Ch. 1, 2.1-2.3, 2.12, 4.1, and 5.1-5.5 of CompOrg Book



# Basics of Computer Organization



To store and retrieve data, we need:

- Disks
- Memory
- Why we need both? (we'll come back in near future)

To process data:

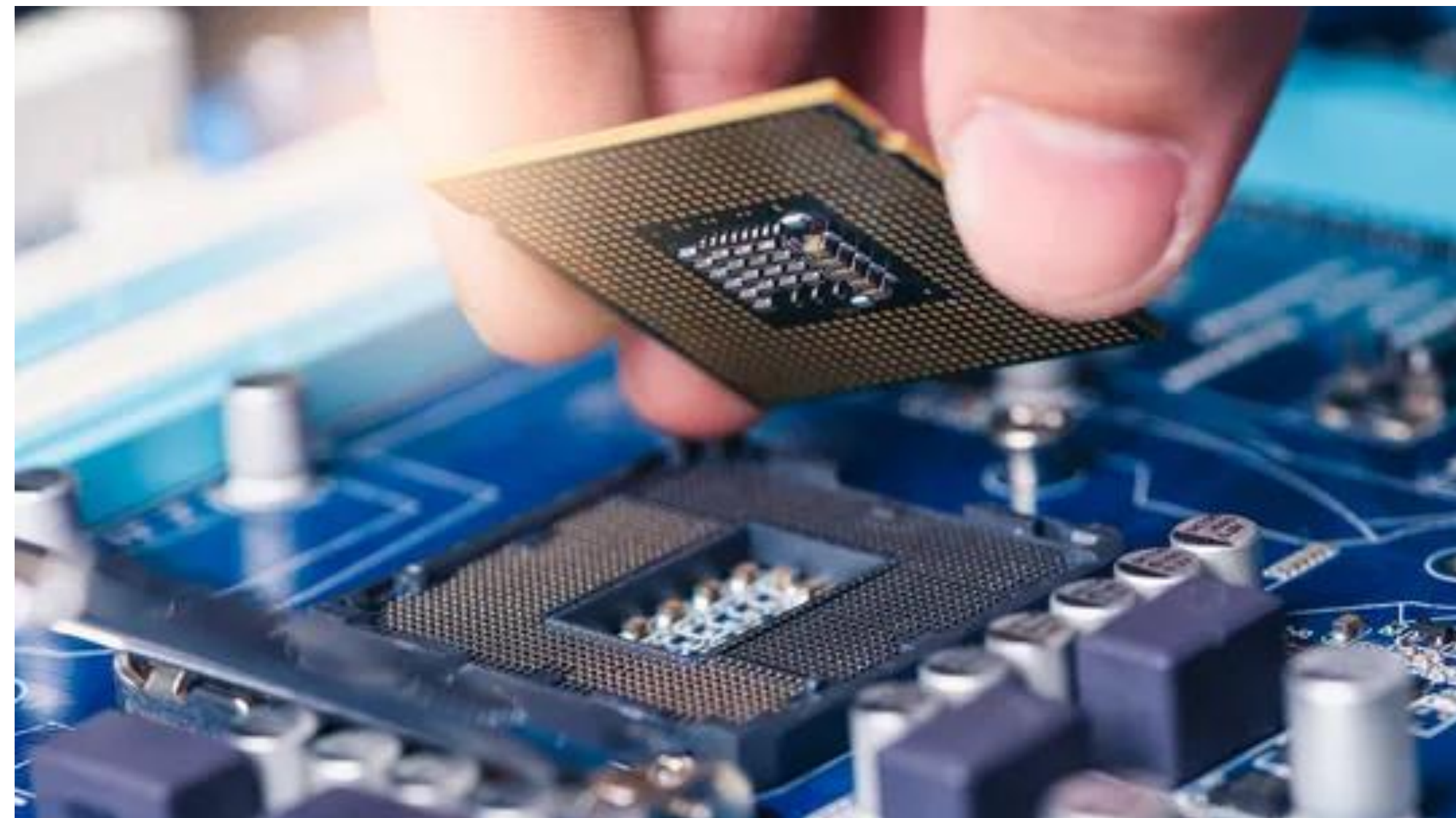
- Processors: CPU and GPU

To retrieve data from remote

- Networks

# Key Parts of Computer Hardware

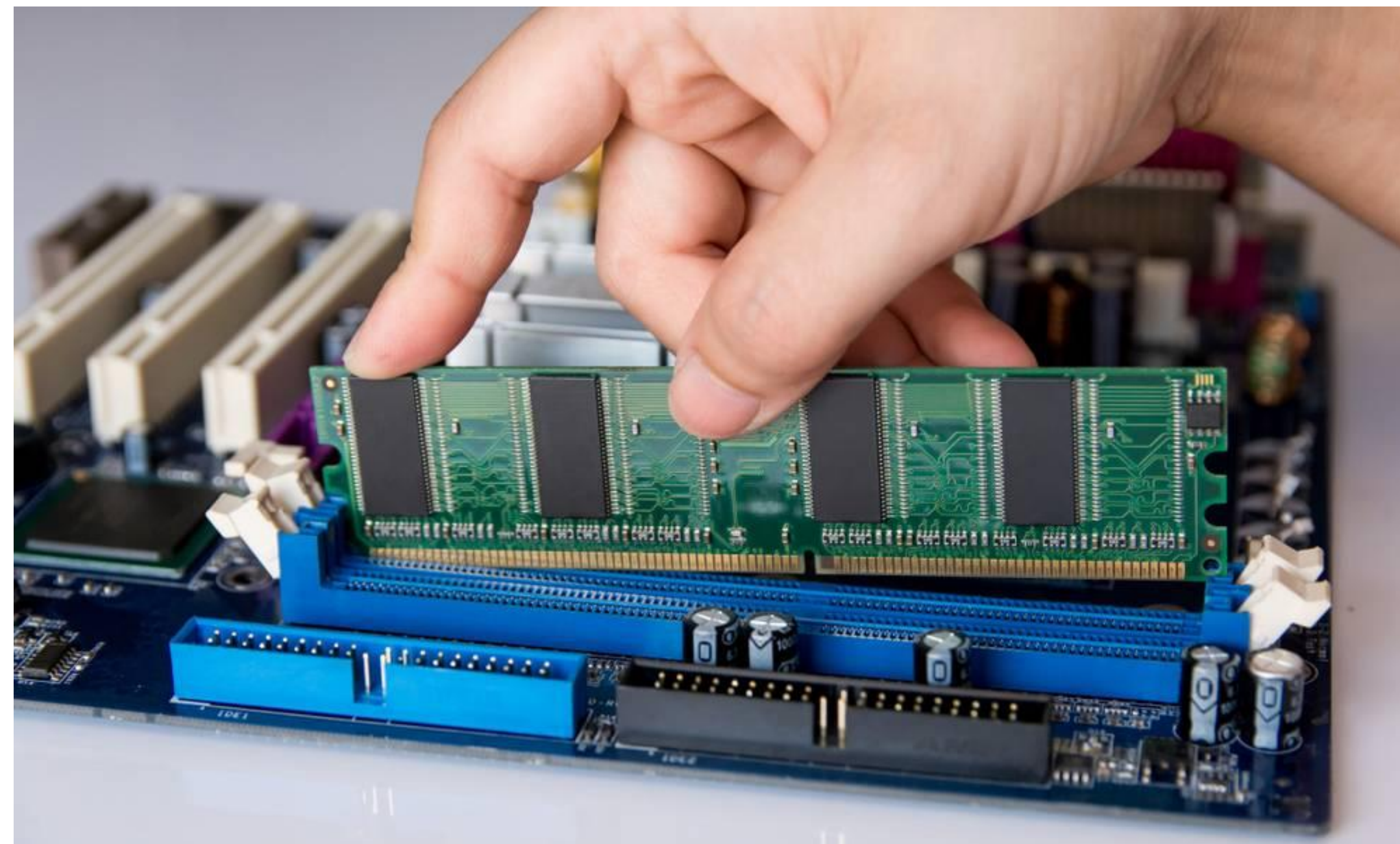
- Processor (CPU, GPU, etc.)
  - Hardware to orchestrate and execute instructions to manipulate data as specified by a program





# Key Parts of Computer Hardware

- Main Memory (aka Dynamic Random Access Memory)
  - Hardware to store data and programs that allows very fast location/retrieval; byte-level addressing scheme





# Key Parts of Computer Hardware

- Disk (aka secondary/persistent storage)
  - Similar to memory but persistent, slower, and higher capacity / cost ratio; various addressing schemes



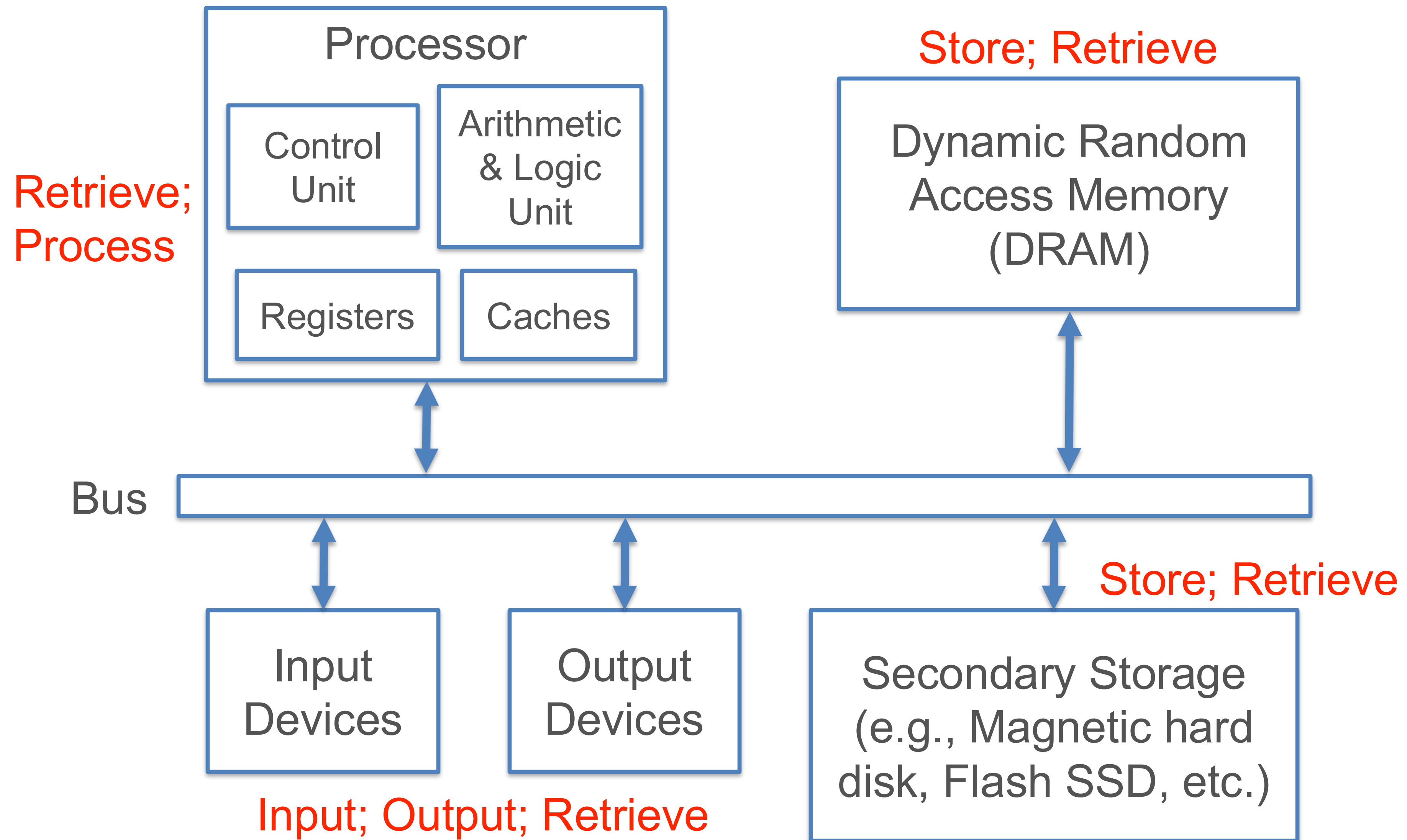


# Key Parts of Computer Hardware

- Network interface controller (NIC)
  - Hardware to send data to / retrieve data over network of interconnected computers/devices

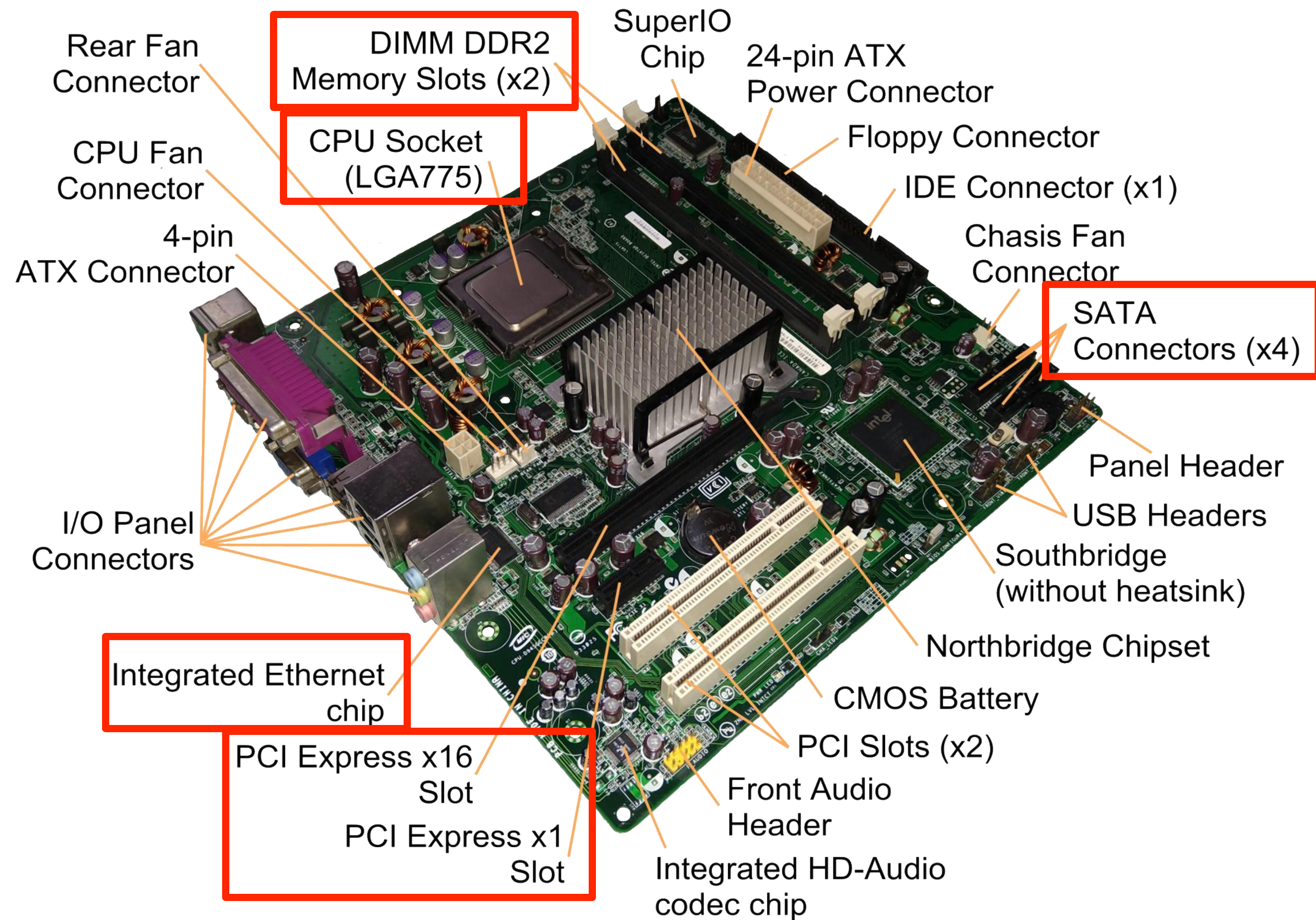


# Abstract Computer Parts and Data



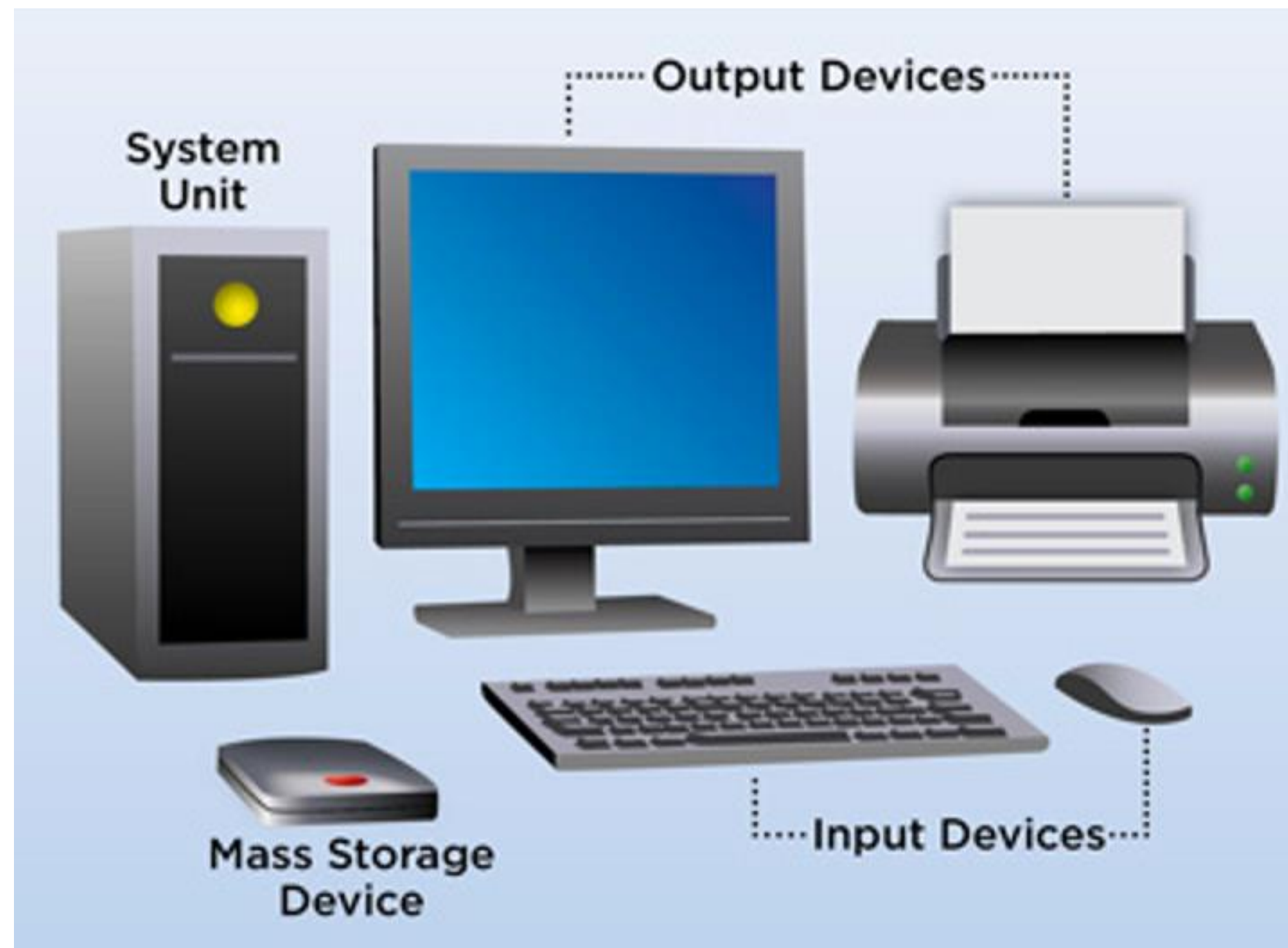


# In Reality





# Parts of a Computer



- Hardware: The electronic machinery (wires, circuits, transistors, capacitors, devices, etc.)
- Software: Programs (instructions) and data



# Key Aspects of Software

- Instruction
  - A command understood by hardware; finite vocabulary for a processor: Instruction Set Architecture (ISA); bridge between hardware and software
- Program (aka code)
  - A collection of instructions for hardware to execute

# Key Aspects of Software

- Programming Language (PL)
  - A human-readable formal language to write programs; at a much higher level of abstraction than ISA
- Application Programming Interface (API)
  - A set of functions (“interface”) exposed by a program/set of programs for use by humans/other programs
- Data
  - Digital representation of information that is stored, processed, displayed, retrieved, or sent by a program

# Main kinds of Software

- Firmware
  - Read-only programs “baked into” a device to offer basic hardware control functionalities
- Operating System (OS)
  - Collection of interrelated programs that work as an intermediary platform/service to enable application software to use hardware more effectively/easily
  - Examples: Linux, Windows, MacOS, etc.

# Main kinds of Software

- Application Software
  - A program or a collection of interrelated programs to manipulate data, typically designed for human use
  - Examples: Excel, Chrome, PostgreSQL, etc.

# Foundation of Data Systems

- Computer Organization
  - **Representation of data**
  - Processors, memory, storage
- OS basics
  - Process, scheduling
  - Memory

Q: How is data represented in computers?

[illegible][illegible]

# Digital Representation of Data

- **Bits: All digital data are sequences of 0 & 1 (binary digits)**
  - **high-low/off-on electromagnetism on disk.**
- Data type: First layer of abstraction to interpret a bit sequence with a human-understandable category of information; interpretation fixed by the PL
  - Example common datatypes: Boolean, Byte, Integer, “floating point” number (Float), Character, and String
- Data structure: A second layer of abstraction to *organize* multiple instances of same or varied data types as a more complex object with specified properties
  - Examples: Array, Linked list, Tuple, Graph, etc.

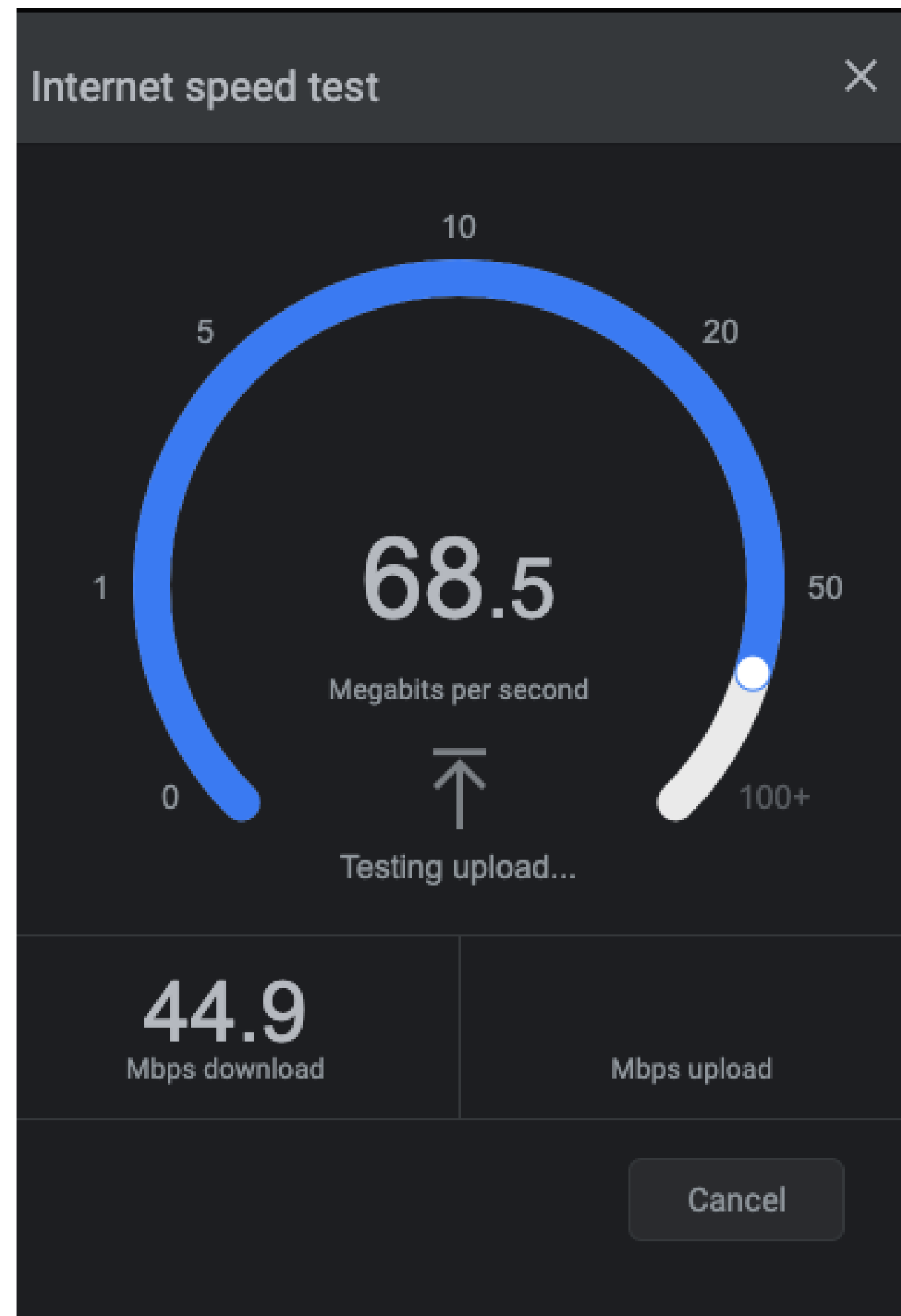








# Digital Representation of Data

- **Bits:** All digital data are sequences of 0 & 1 (binary digits)
  - high-low/off-on electromagnetism on disk.
- **Data type:** First layer of abstraction to interpret a bit sequence with a human-understandable category of information; interpretation fixed by the PL
  - e.g.: Boolean, Byte, Integer, “floating point” number (Float), Character, and String
- **Data structure:** A second layer of abstraction to organize multiple instances of same or varied data types as a more complex object with specified properties
  - Examples: Array, Linked list, Tuple, Graph, etc.

# Count everything in binary

- Use Base 2 to represent Number
  - 0, 1, 10, 11, 100, 101, ...
  - Represent  $15213_{10}$  as  $0011\ 1011\ 0110\ 1101_2$
  - Represent  $1.20_{10}$  as  $1.0011\ 0011\ 0011\ 0011\ [0011]..._2$
- Represent negative numbers as ...?
  - (we'll come back to this)



Name	Size ▾	Kind
 HB50 cupcakes.JPG	2 MB	JPEG image
 Roller Skating.JPG	1.3 MB	JPEG image
 50HBJukebox2.jpg	720 KB	JPEG image
 Facebook.tiff	399 KB	TIFF image
 7_days_to_enrol.png	173 KB	PNG image
 JoggingShoes.jpg	71 KB	JPEG image

(Capital) B (bytes) vs. (lower case) b (bits)

# Encoding Byte Values

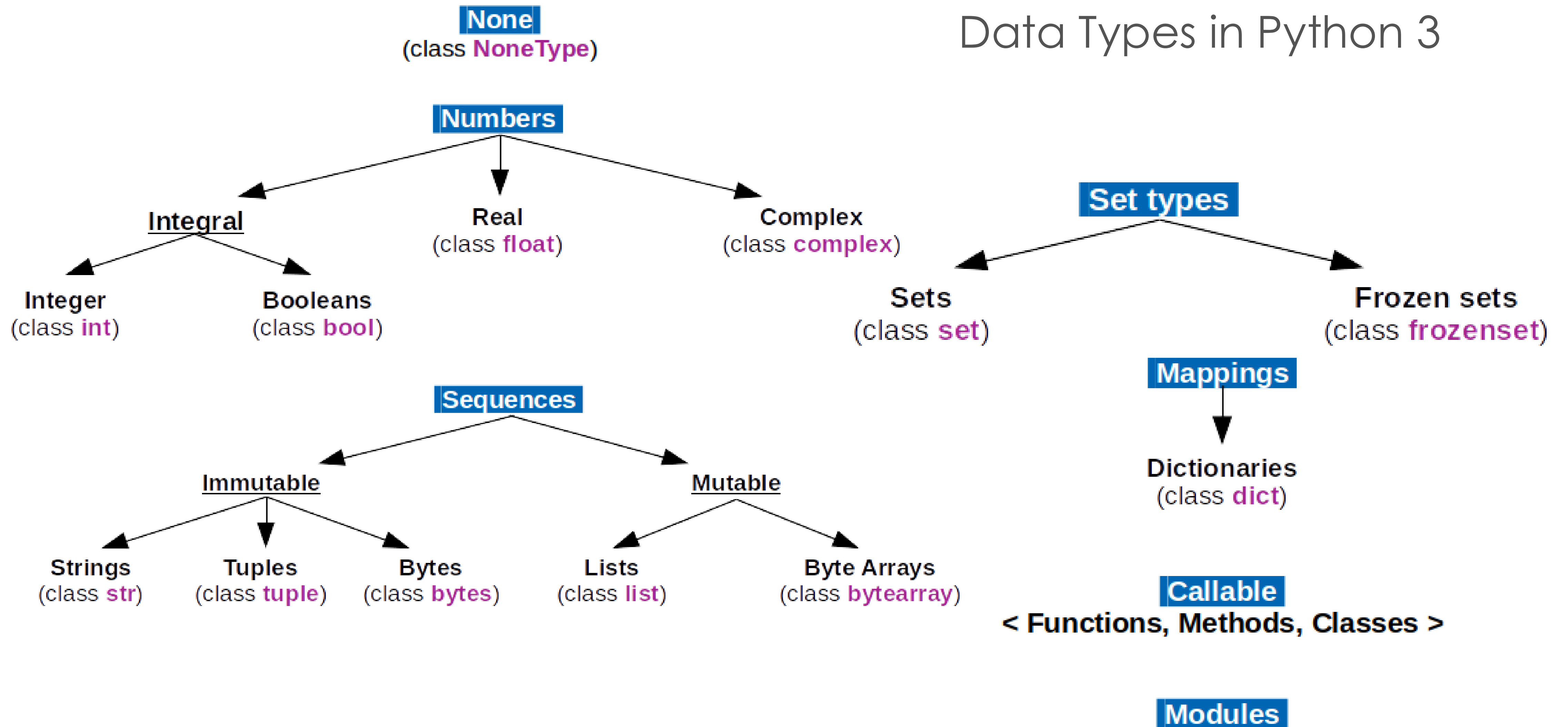
- Byte = 8 bits
- Why?
  - Historical Development
  - Practicality and Standardization
- A Byte (B; 8 bits) is typically the basic unit of data types
  - CPU can't address anything smaller than a byte.

## Bytes -> Data types: bool, int, float, string, ...

- The *size* and *interpretation* of a data type depends on PL
- Boolean:
  - Examples in data sci.: Y/N or T/F responses
  - Just 1 bit needed but actual size is almost always 1B, i.e., 7 bits are wasted!
- Integer:
  - Examples in data science: #friends, age, #likes
  - Typically 4 bytes; many variants (short, unsigned, etc.)
  - Java *int* can represent  $-2^{31}$  to  $(2^{31} - 1)$ ; C *unsigned int* can represent 0 to  $(2^{32} - 1)$ ;

# Digital Representation of Data

## Data Types in Python 3



# Digital Representation of Data

*Q: How many unique data items can be represented by 3 bytes?*

- Given  $k$  bits, we can represent  $2^k$  unique data items
- 3 bytes = 24 bits  $\Rightarrow 2^{24}$  items, i.e., 16,777,216 items
- Common approximation:  $2^{10}$  (i.e., 1024)  $\sim 10^3$  (i.e., 1000); recall kibibyte (KiB = 1024 B) vs kilobyte (KB = 1000 B) and so on

*Q: How many bits are needed to distinguish 97 data items?*

- For  $k$  unique items, invert the exponent to get  $\log_2(k)$
- But #bits is an integer! So, we only need  $\lceil \log_2(k) \rceil$
- So, we only need the next higher power of 2
- $97 \rightarrow 128 = 2^7$ ; so, 7 bits

# Digital Representation of Data

Q: How to convert from decimal to binary representation?

- Given decimal  $n$ , if power of 2 (say,  $2^k$ ), put 1 at bit position  $k$ ; if  $k=0$ , stop; else pad with trailing 0s till position 0
- If  $n$  is not power of 2, identify the power of 2 just below  $n$  (say,  $2^k$ ); #bits is then  $k$ ; put 1 at position  $k$
- Reset  $n$  as  $n - 2^k$ ; return to Steps 1-2
- Fill remaining positions in between with 0s

	7	6	5	4	3	2	1	0	Position/Exponent of 2
Decimal	128	64	32	16	8	4	2	1	Power of 2
$5_{10}$						1	0	1	
$47_{10}$			1	0	1	1	1	1	
$163_{10}$	1	0	1	0	0	0	1	1	
$16_{10}$				1	0	0	0	0	

Q: Binary to decimal?



# Digital Representation of Data

```
void show_squares()
{
    int x;
    for (x = 5; x <= 5000000; x*=10)
        printf("x = %d x^2 = %d\n", x, x*x);
}
```

$x = 5 \quad x^2 = 25$

$x = 50 \quad x^2 = 2500$

$x = 500 \quad x^2 = 250000$

$x = 5000 \quad x^2 = 25000000$

$x = 50000 \quad x^2 = -1794967296$

$x = 500000 \quad x^2 = 891896832$

$x = 5000000 \quad x^2 = -1004630016$



# Two-complement: Simple Example

	-16	8	4	2	1	
10 =	0	1	0	1	0	8+2 = 10

	-16	8	4	2	1	
-10 =	1	0	1	1	0	-16+4+2 = -10

# Encoding Integers

## Unsigned

$$B2U(X) = \sum_{i=0}^{w-1} x_i \cdot 2^i$$

## Two's Complement

$$B2T(X) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$$

**Sign Bit**



```
short int x = 15213;  
short int y = -15213;
```

# Two-complement Encoding Example (Cont.)

**x =**        15213: 00111011 01101101  
**y =**        -15213: 11000100 10010011

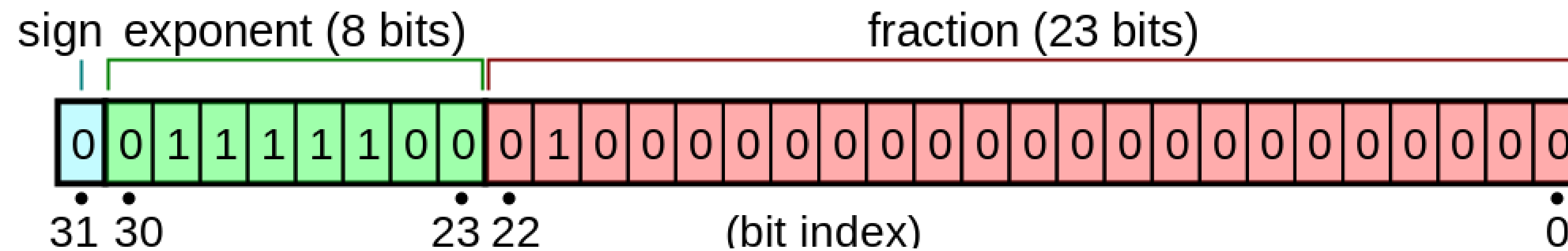
Weight	15213		-15213	
1	1	1	1	1
2	0	0	1	2
4	1	4	0	0
8	1	8	0	0
16	0	0	1	16
32	1	32	0	0
64	1	64	0	0
128	0	0	1	128
256	1	256	0	0
512	1	512	0	0
1024	0	0	1	1024
2048	1	2048	0	0
4096	1	4096	0	0
8192	1	8192	0	0
16384	0	0	1	16384
-32768	0	0	1	-32768
<b>Sum</b>	<b>15213</b>		<b>-15213</b>	

# Digital Representation of Data

- **Float:**
  - Examples in data sci.: salary, scores, model weights
  - IEEE-754 single-precision format is 4B long; double-precision format is 8B long
  - Java and C *float* is single; Python *float* is double!

# Digital Representation of Data

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



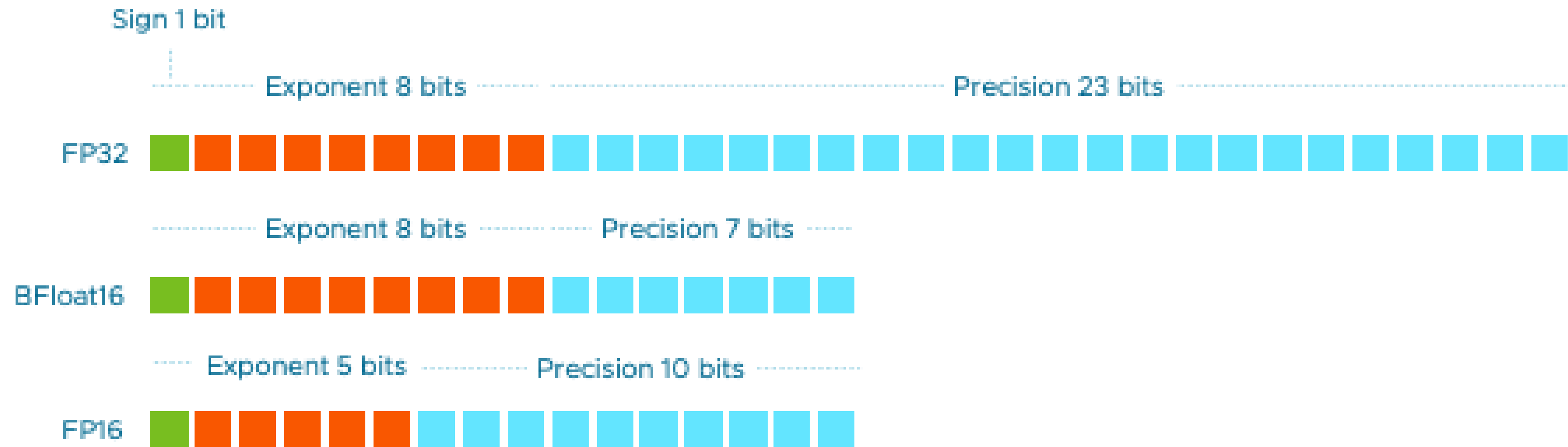
$$(-1)^{sign} \times 2^{exponent-127} \times \left(1 + \sum_{i=1}^{23} b_{23-i} 2^{-i}\right)$$

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

# Digital Representation of Data

- More float standards: double-precision (float64; 8B) and half-precision (float16; 2B); different #bits for exponent, fraction
- Float16 is now common for **deep learning** parameters:
  - Native support in PyTorch, TensorFlow, etc.; APIs also exist for weight quantization/rounding post training

# New magical float standards



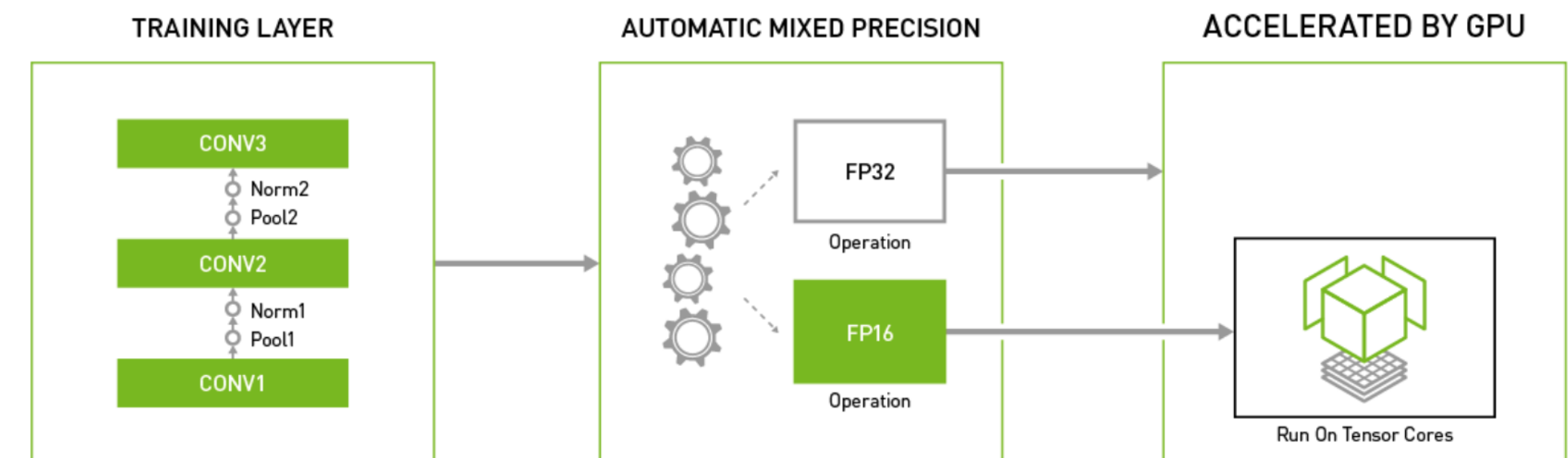
What's the difference between bf16 and fp16?



# Fp16 vs. Fp32

NVIDIA Deep Learning SDK support mixed-precision training; 2-3x speedup with similar accuracy!

Form Factor	H100 SXM
FP64	34 teraFLOPS
FP64 Tensor Core	67 teraFLOPS
FP32	67 teraFLOPS
TF32 Tensor Core	989 teraFLOPS <sup>2</sup>
BFLOAT16 Tensor Core	1,979 teraFLOPS <sup>2</sup>
FP16 Tensor Core	1,979 teraFLOPS <sup>2</sup>
FP8 Tensor Core	3,958 teraFLOPS <sup>2</sup>



Using Automatic Mixed Precision for Major Deep Learning Frameworks

# 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 65, '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?

## Practice Qs (will appear in Final)

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

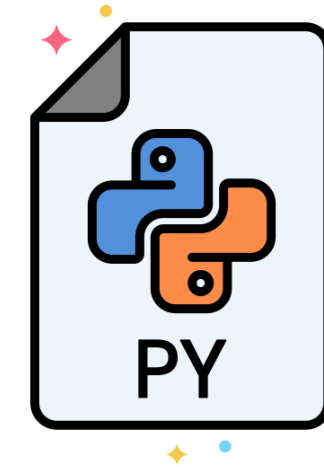
Q2: Deep Dive: what does **exponent** and **fraction** control in float point representation?

Q3: What is the difference between BF16 and FP16?

# Q1: How many 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 many 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	

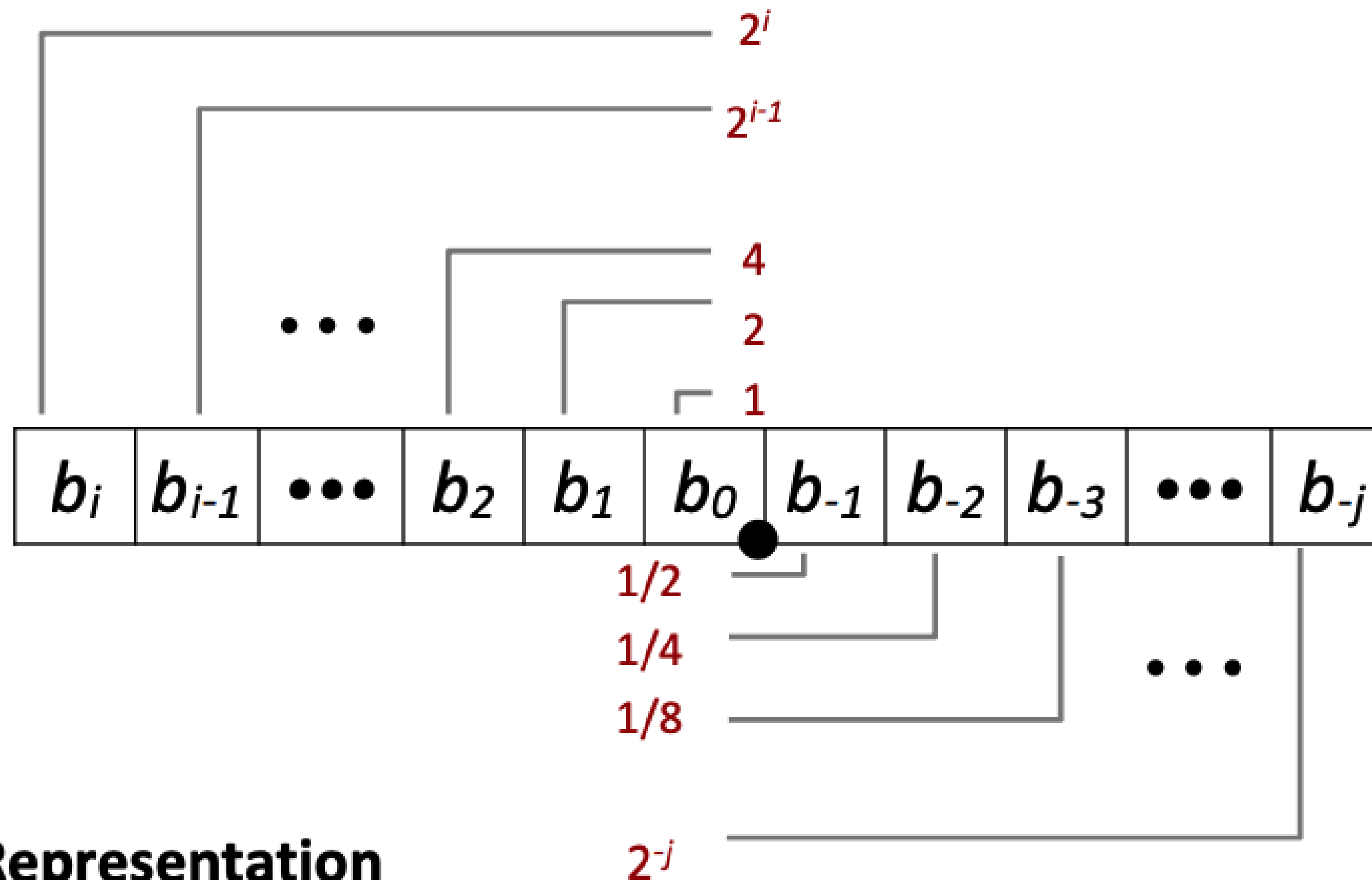
## Practice Qs (review next class)

Q1: How many 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



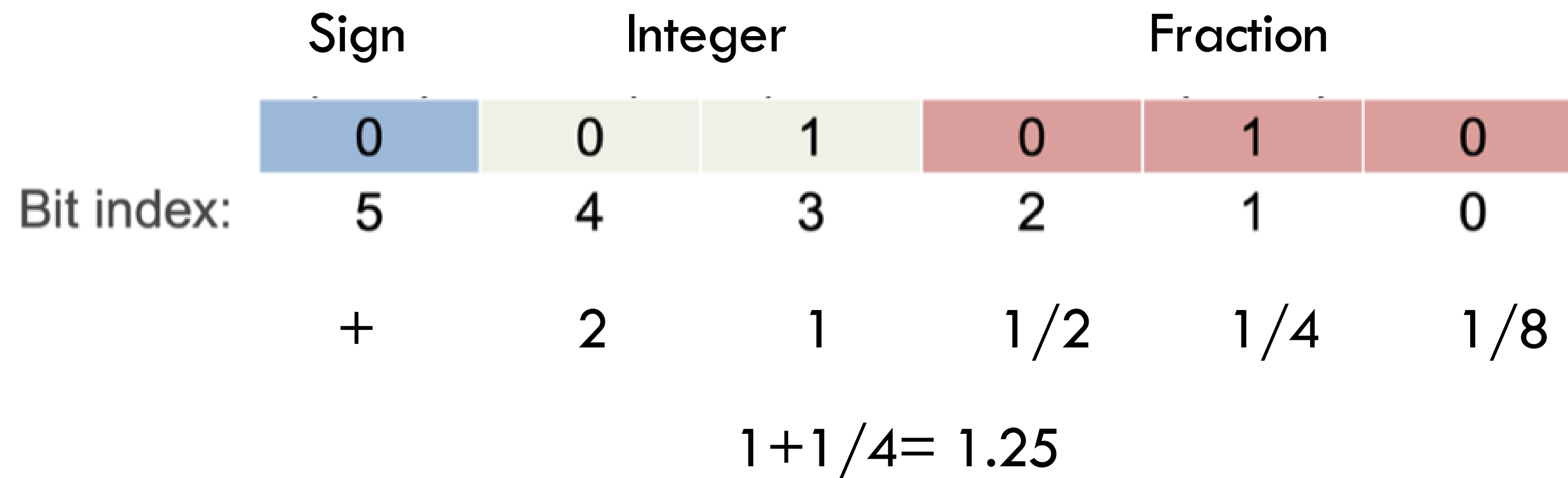
## ■ Representation

- Bits to right of “binary point” represent fractional powers of 2
- Represents rational number:

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



# Let's design a fix-point FP6



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

## An Example

$$0.625_{10} =$$

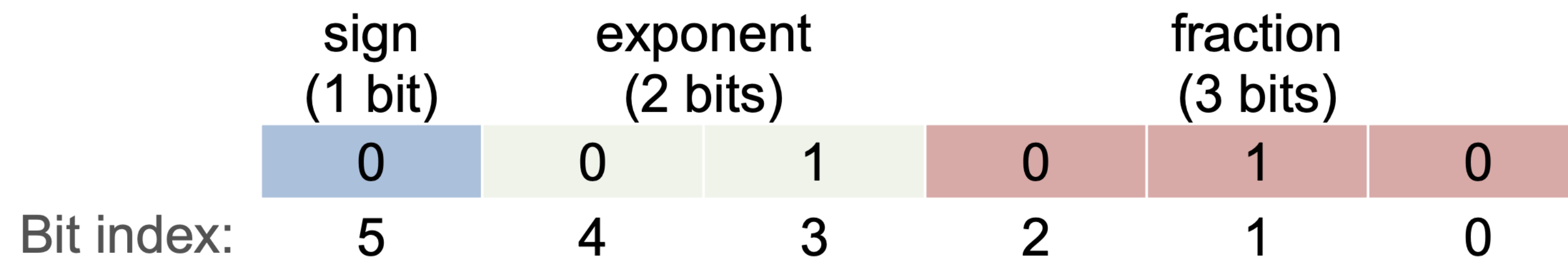
$$0.625_{10} = 0.101_2$$

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

## An Example (Cont.)

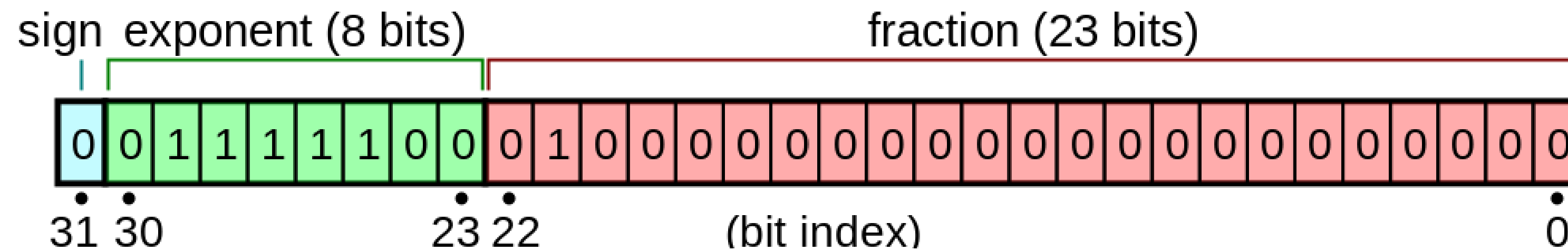
$$0.625_{10} = 0.101_2 = 1.01 \cdot 2^{-1}$$

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



# Digital Representation of Data

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

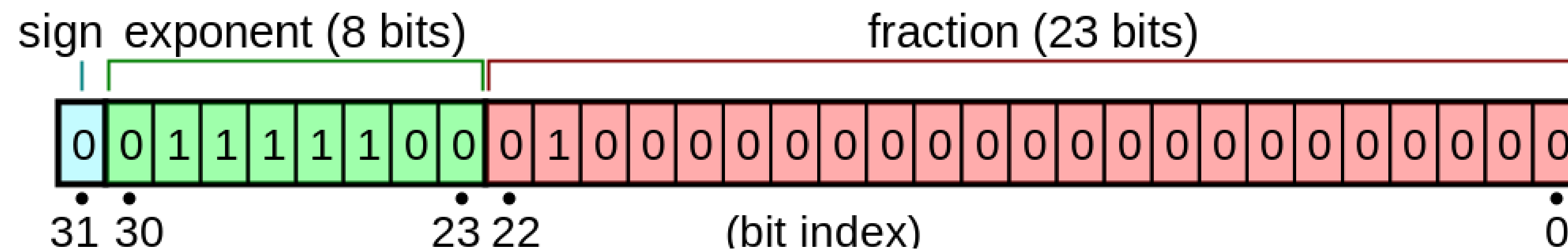


$$(-1)^{sign} \times 2^{exponent-127} \times \left(1 + \sum_{i=1}^{23} b_{23-i} 2^{-i}\right)$$

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

## Q2: What does **exponent** and **fraction** control?

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

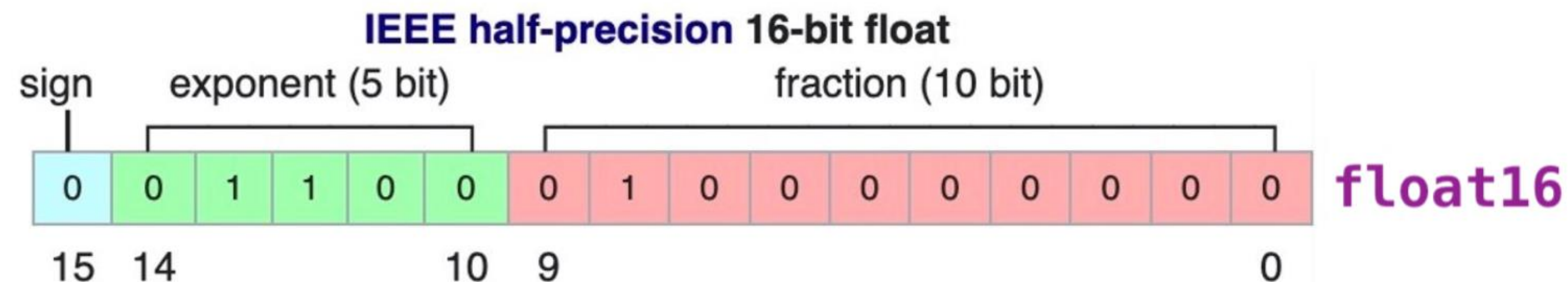


Q2: What does **exponent** and **fraction** control?

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

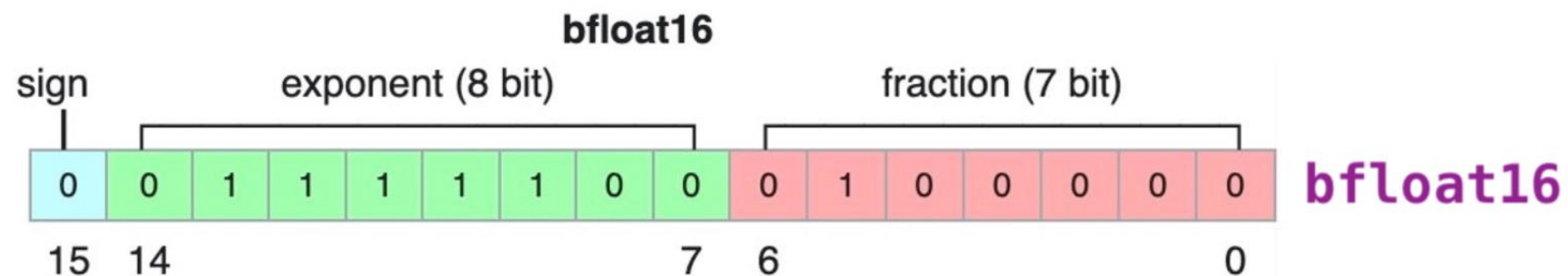
- More complex (to both human and computers)
- Inconsistent precision

Q3: What is the difference between BF16 and FP16?



Less exponent -> smaller range -> easier to overflow

More fraction -> more precise

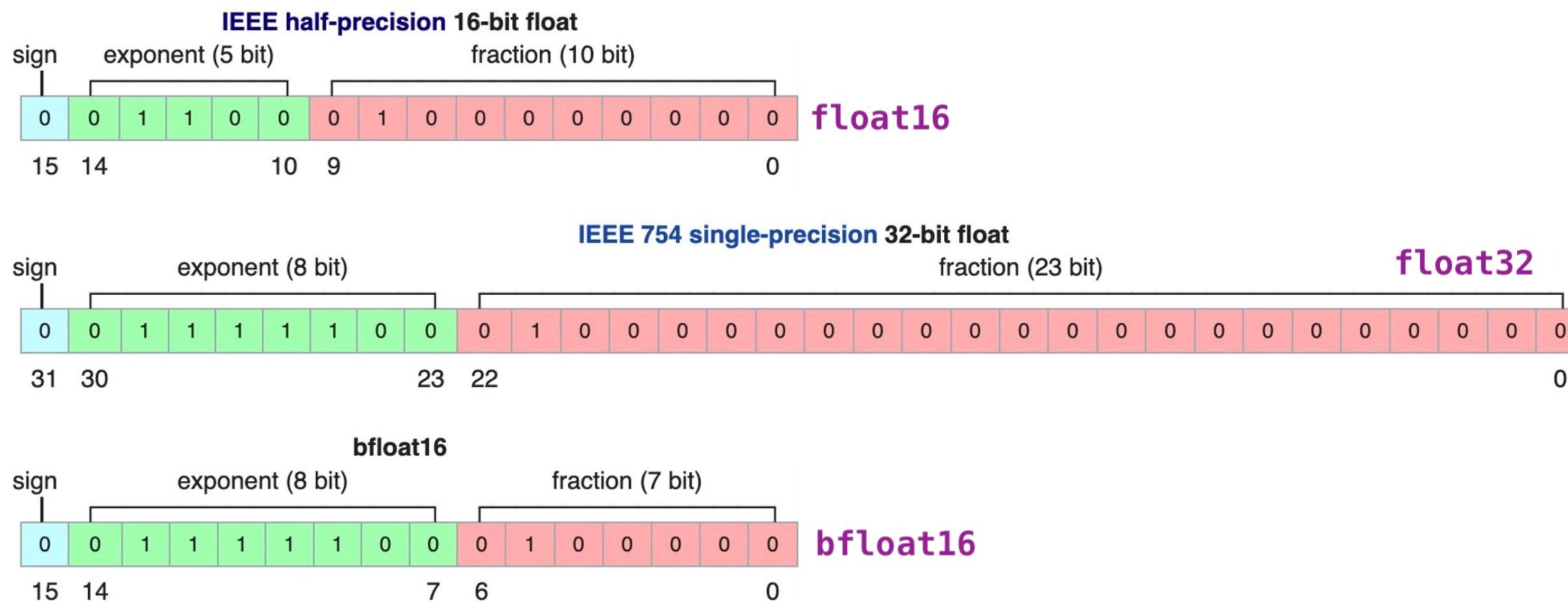


more exponent -> larger range -> harder to overflow

less fraction -> less precise

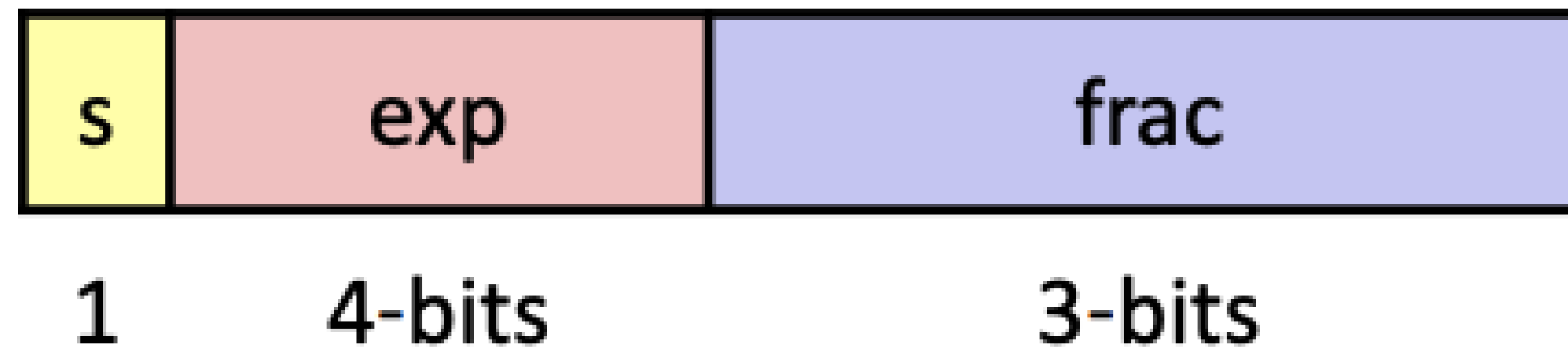
# Why BF16 is better in ML/AI?

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



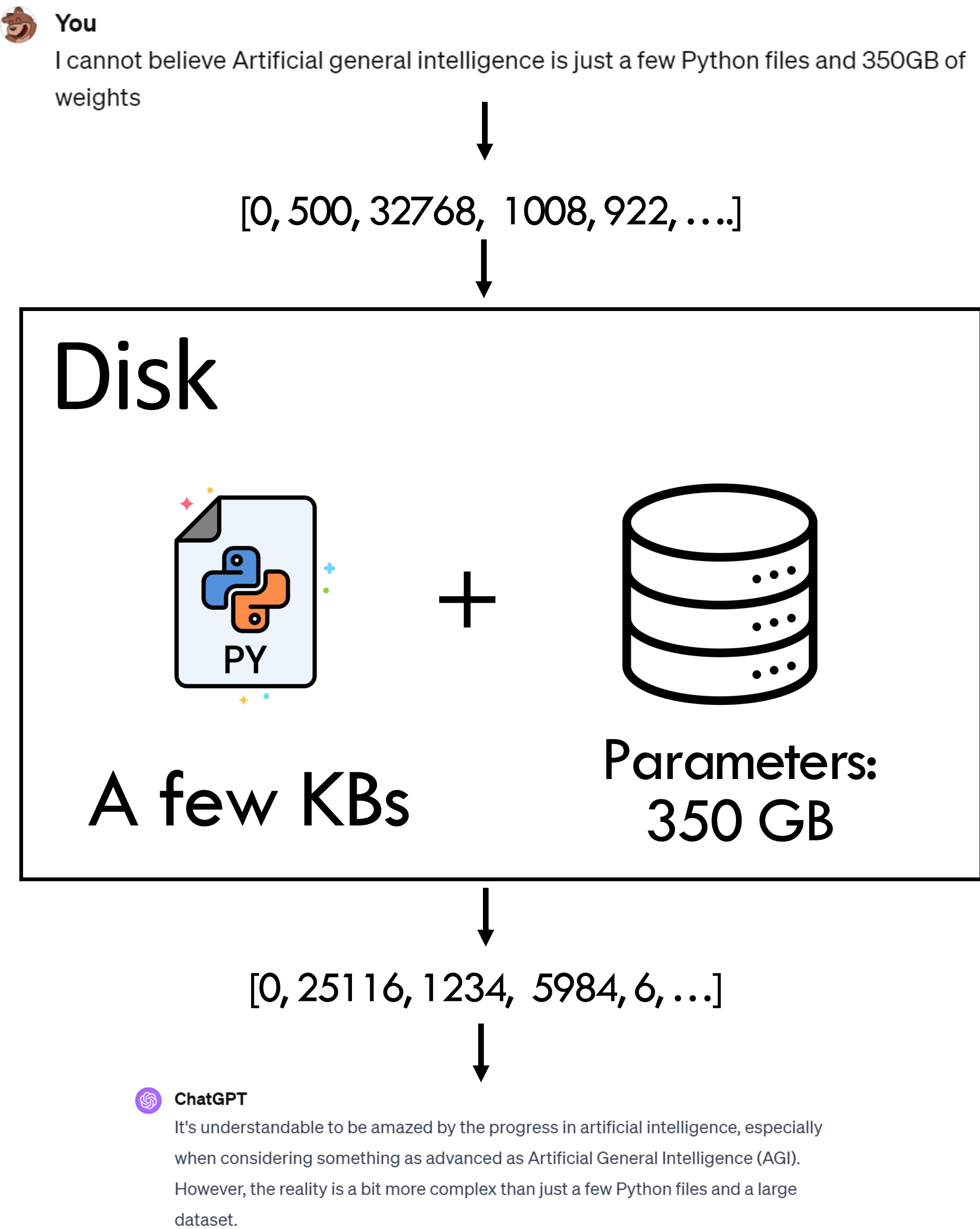


Examples in the final exam: FP8



# GPT Again

GPT =



`str`

`List[integers]`

`List[integers]`

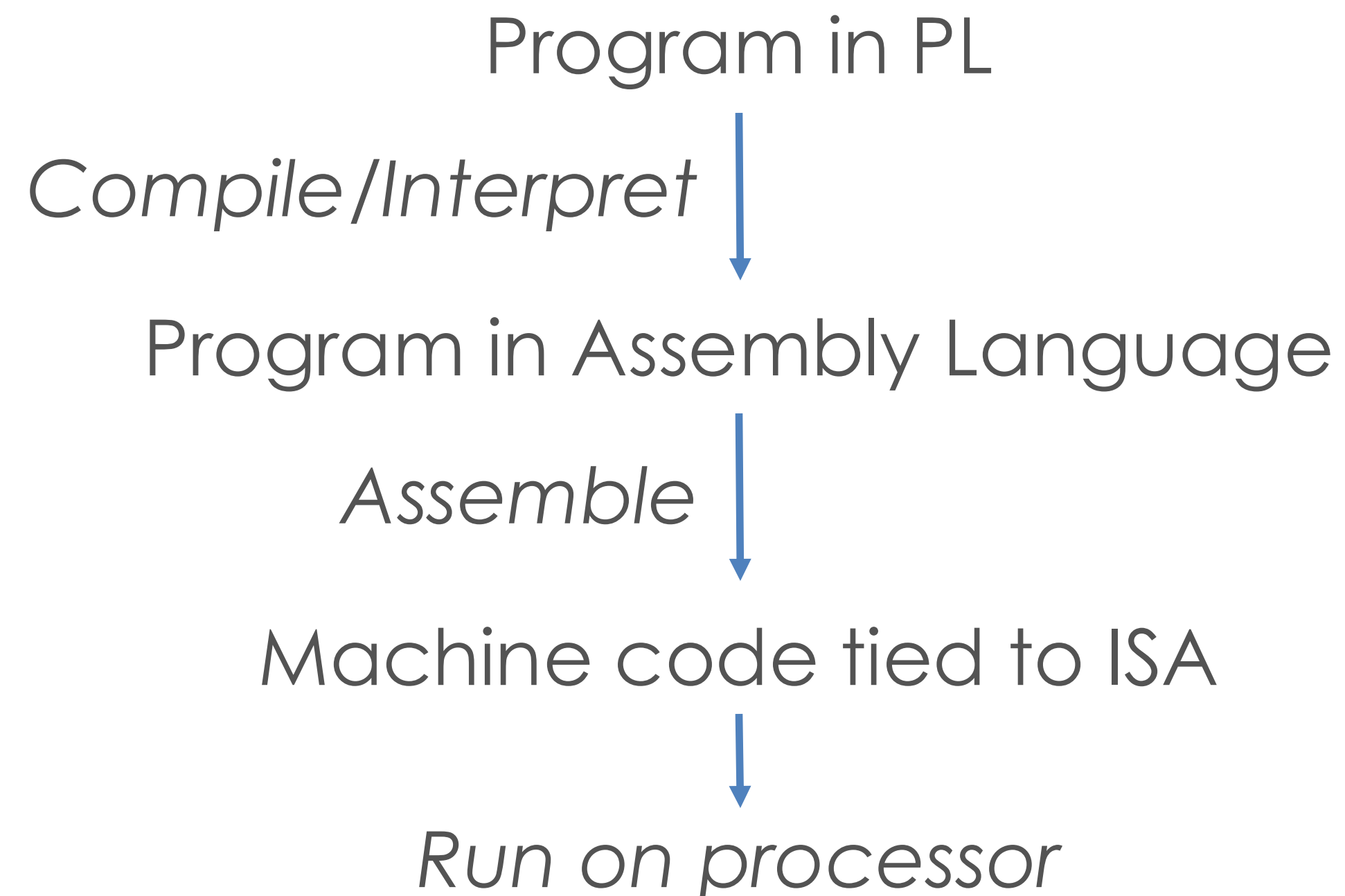
`str`

# Foundation of Data Systems

- Computer Organization
  - Representation of data
  - **processors, memory, storage**
- OS basics
  - Process, scheduling
  - Memory

# 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



```
80483b4: 55          push    %ebp
80483b5: 89 e5       mov     %esp,%ebp
80483b7: 83 e4 f0    and     $0xffffffff0,%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
```

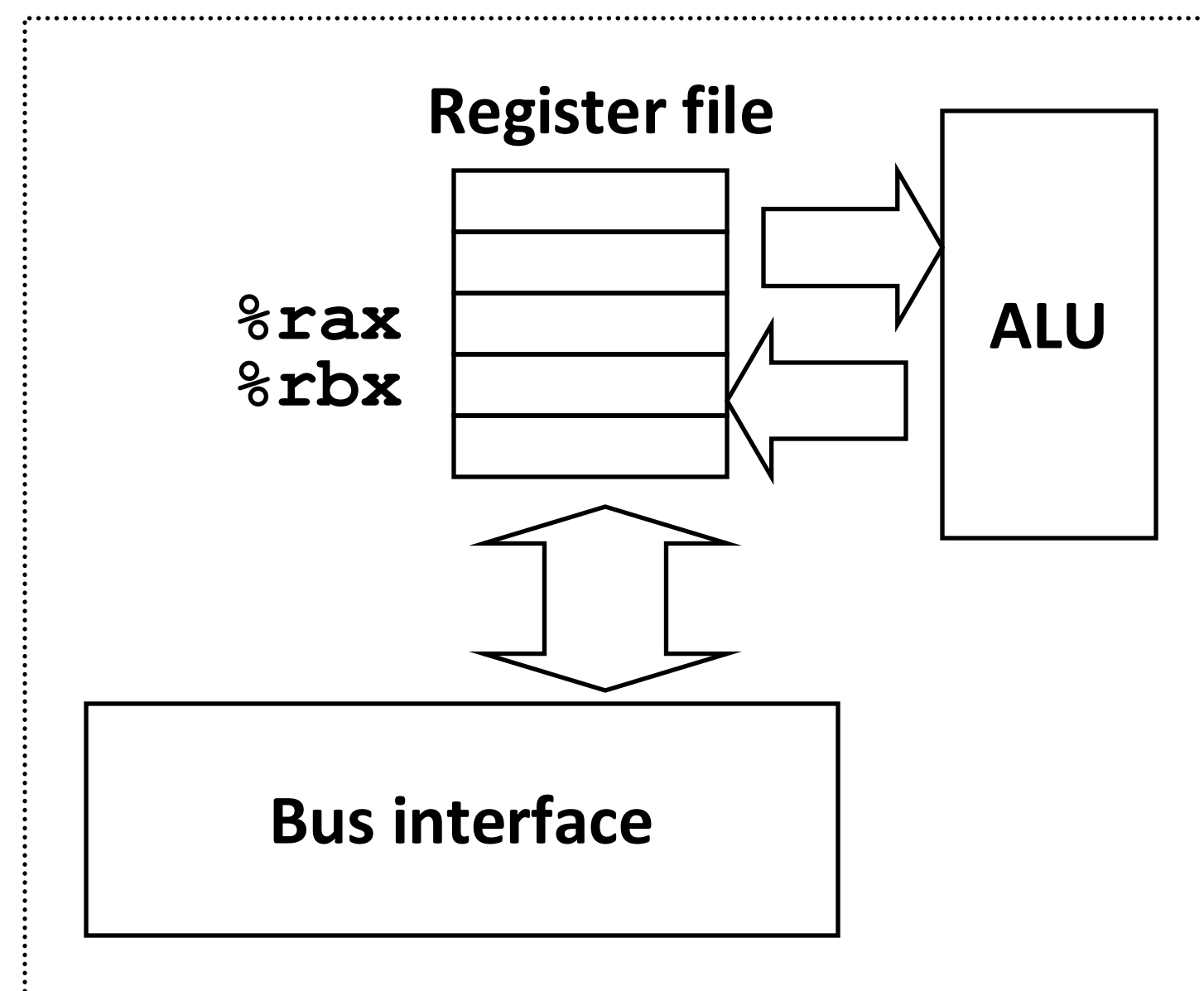
# Basics of Processors

*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

## CPU chip



## Register names

**addq    %rbx,    %rax**

is

**rax += rbx**

# How Fast is Processor

Instruction / second: number of instructions per second

intel cpu floating point per seconds?

All

Images

Shopping

Videos

News

More

Tools

Generative AI is experimental. [Learn more](#)

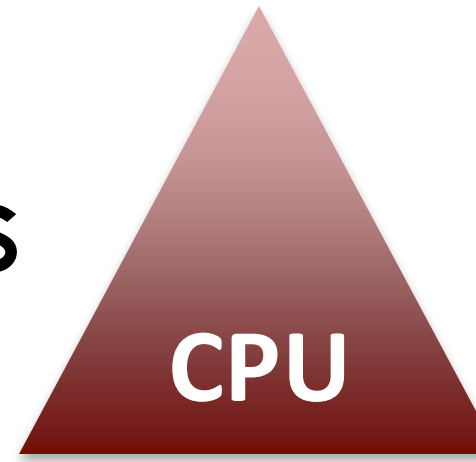
The floating-point operations per second (FLOPS) of an Intel Core i7 processor can vary depending on the model and clock speed. On average, a mid-range Intel Core i7 processor can perform around 100–200 GFLOPS (billion floating-point operations per second).

CPU's 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 <sup>2</sup>
BFLOAT16 Tensor Core	1,979 teraFLOPS <sup>2</sup>
FP16 Tensor Core	1,979 teraFLOPS <sup>2</sup>
FP8 Tensor Core	3,958 teraFLOPS <sup>2</sup>

# Problem?

100 GFLOPs/s



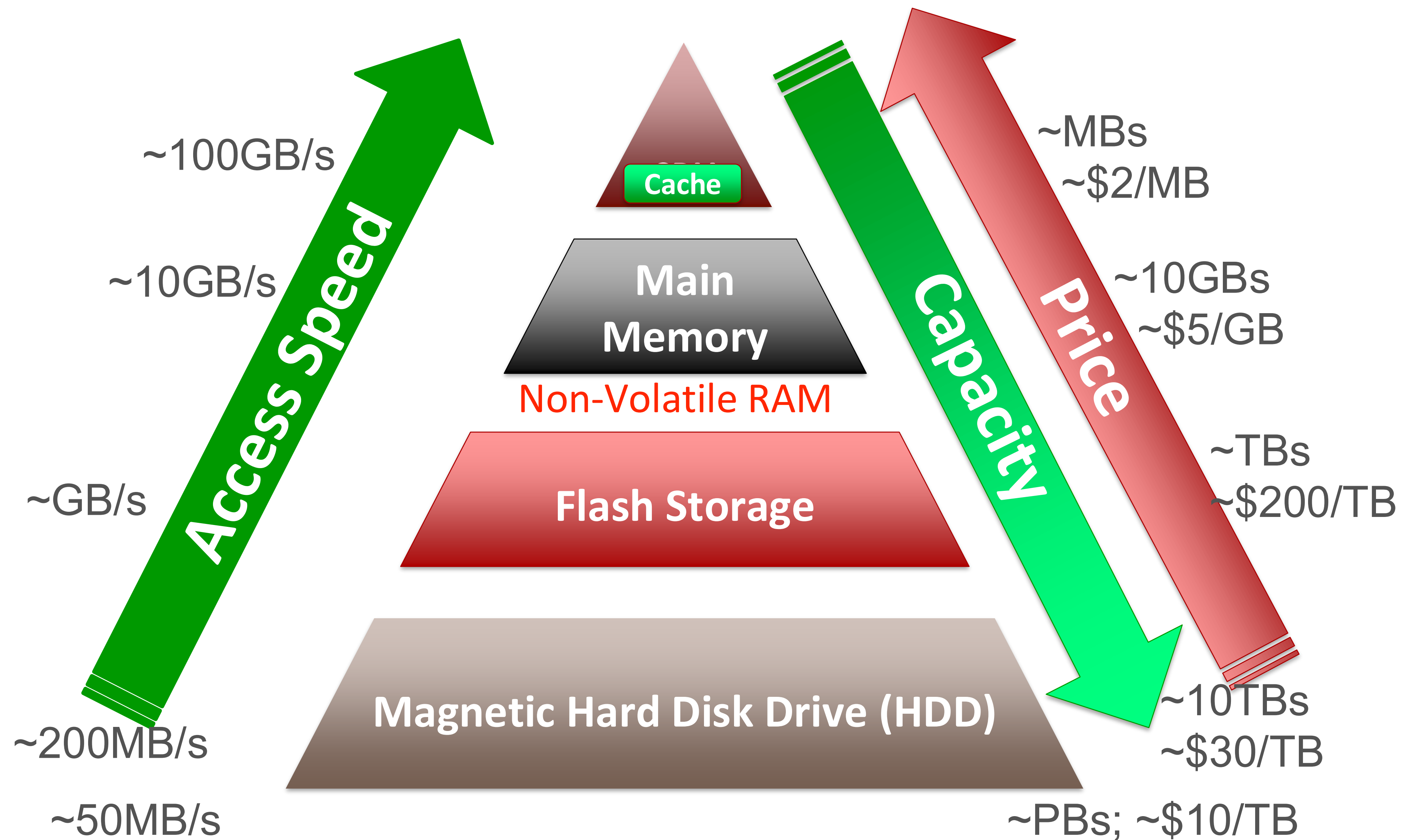
1. Assume we use 0.5s to perform 50 FLOPs
2. We need to read  $50 \times 2 = 100$  GB in the rest of 0.5s to keep the CPU busy
3. We need the CPU to read at a speed of  $100\text{GB} / 0.5\text{s} = 200 \text{ GB/s}$

Magnetic Hard Disk Drive (HDD)

80 – 160 MB/s



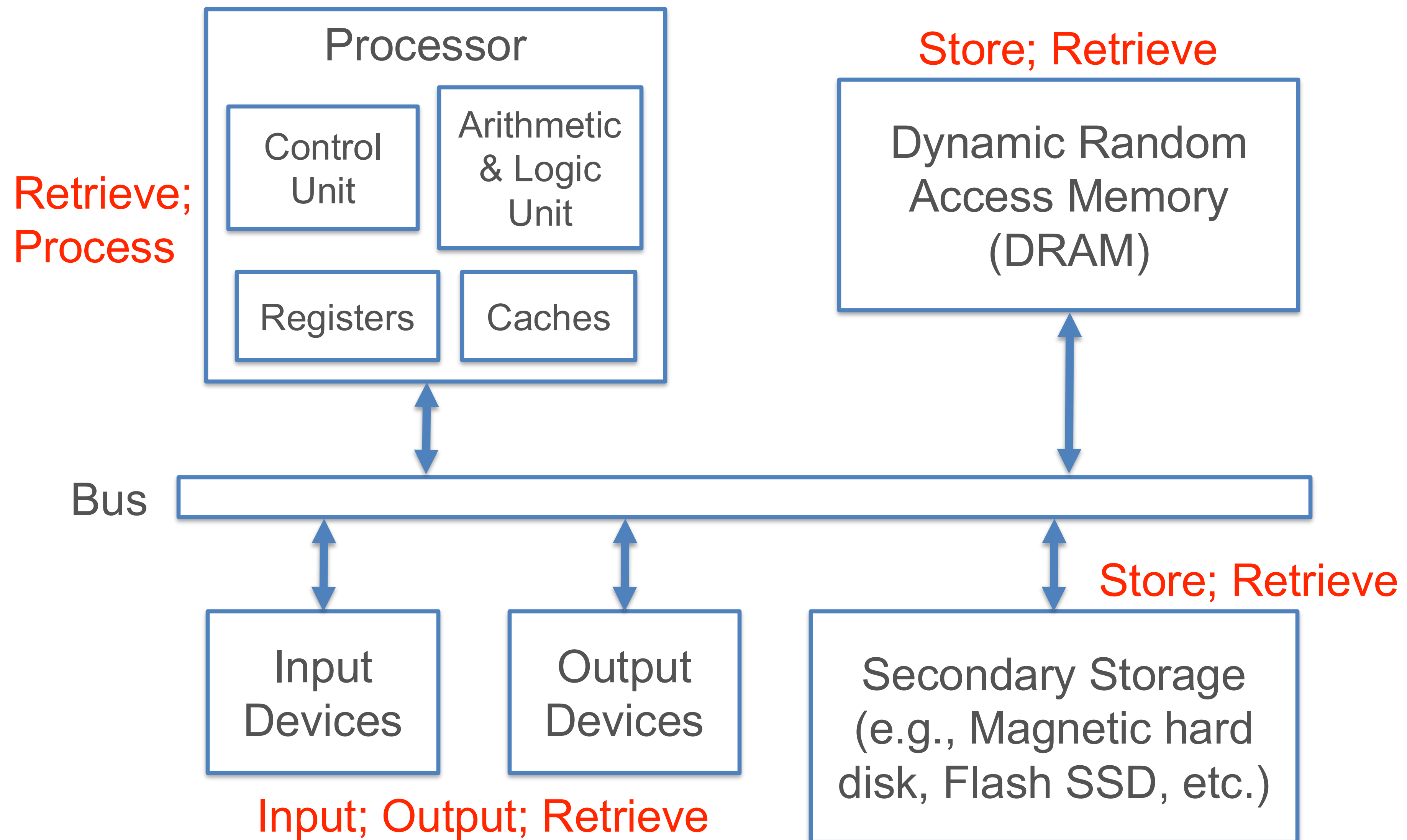
# Memory/Storage Hierarchy



# Writing & Reading Memory Instructions

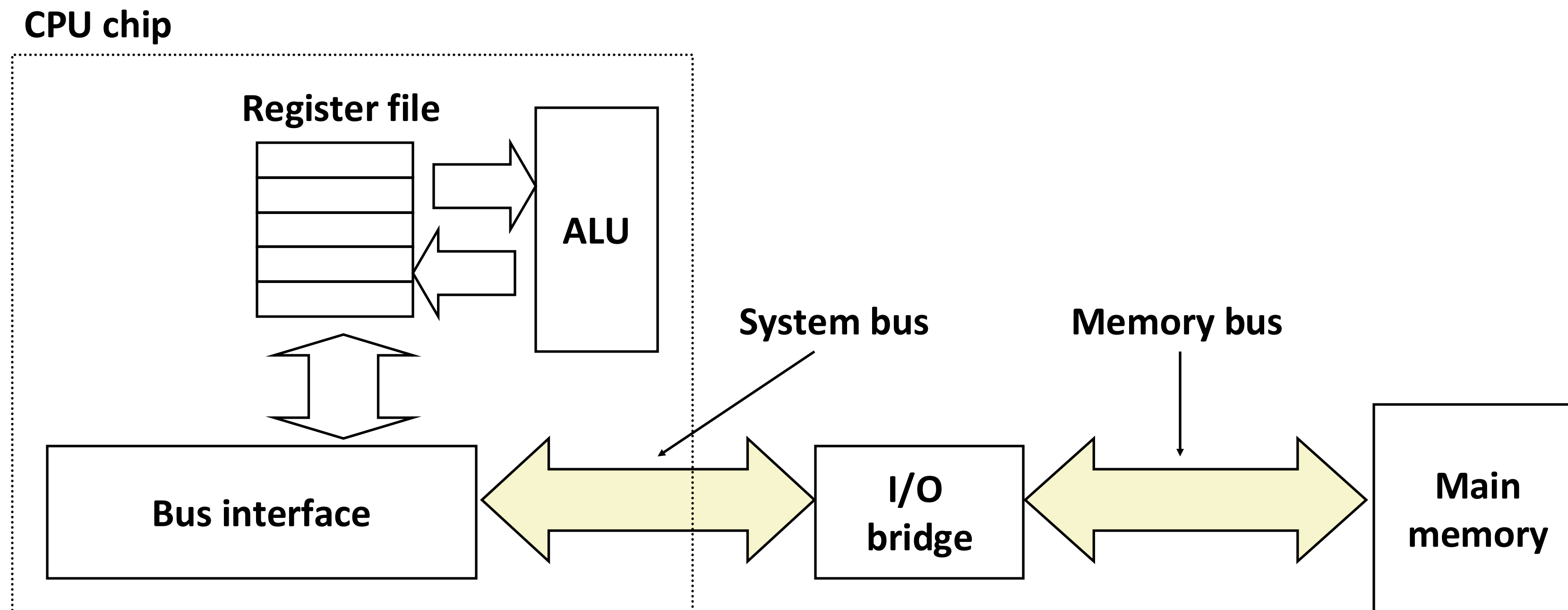
- Write
  - Transfer data from memory to CPU  
`movq %rax, %rsp`
  - “Store” operation
- Read
  - Transfer data from CPU to memory  
`movq %rsp, %rax`
  - “Load” operation

# Abstract Computer Parts and Data

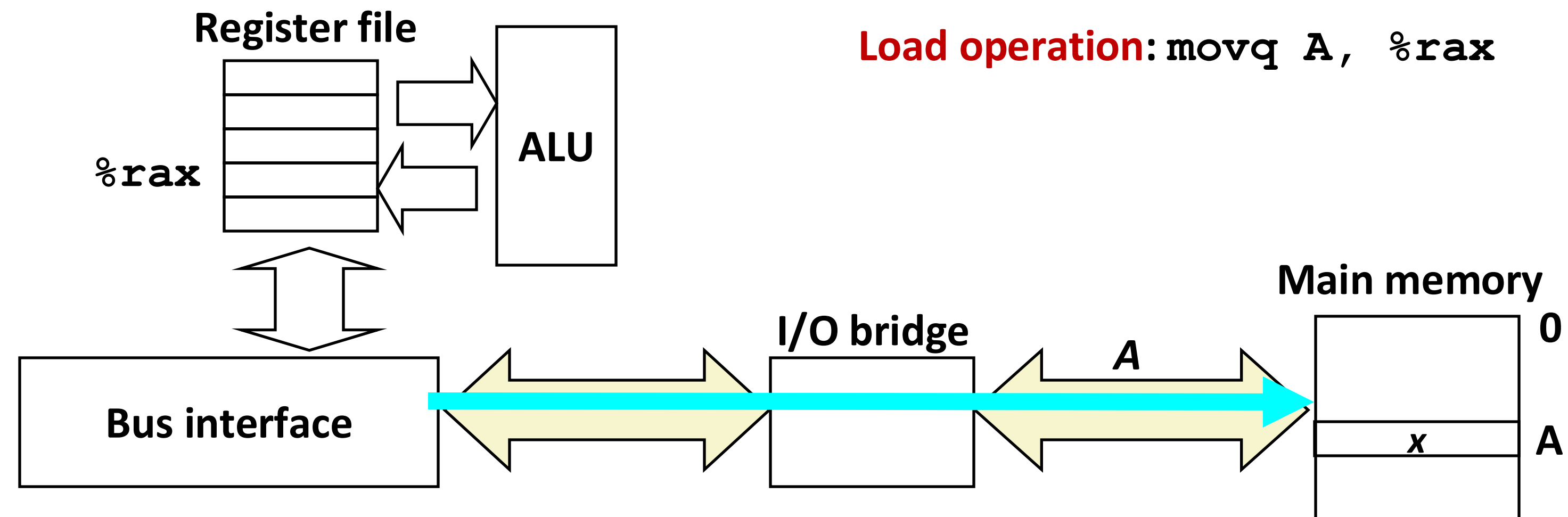


# Bus Structure Connecting CPU and Memory

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

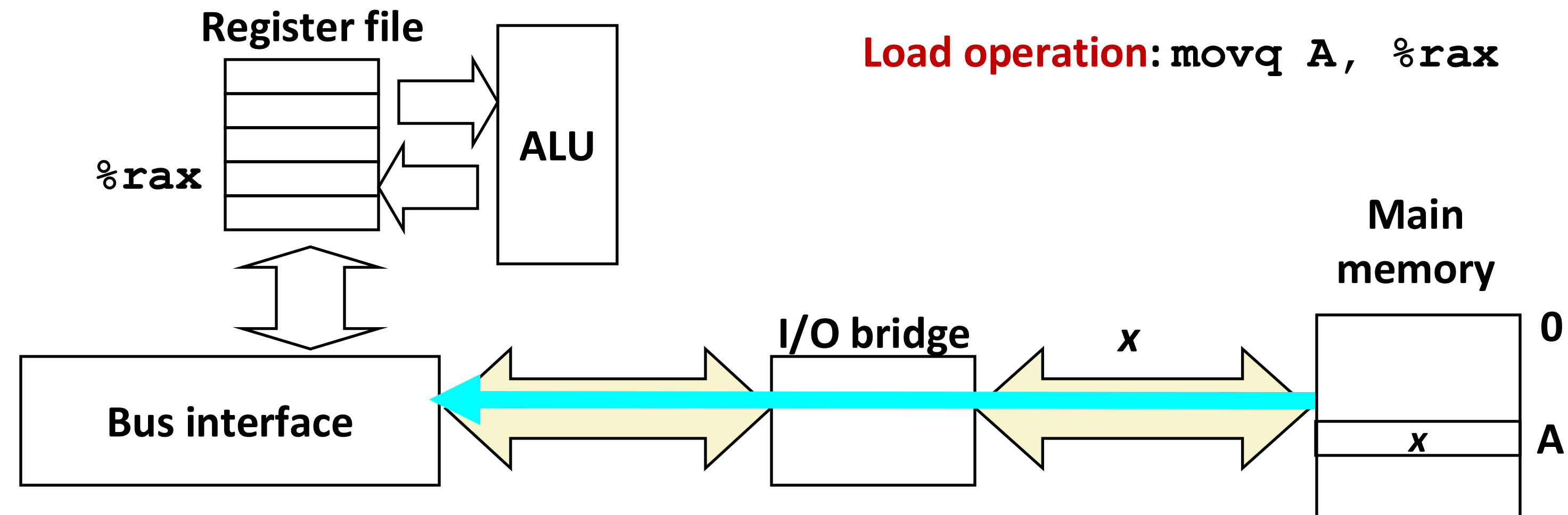


# Memory Read Transaction (1)



- CPU places address **A** on the memory bus.

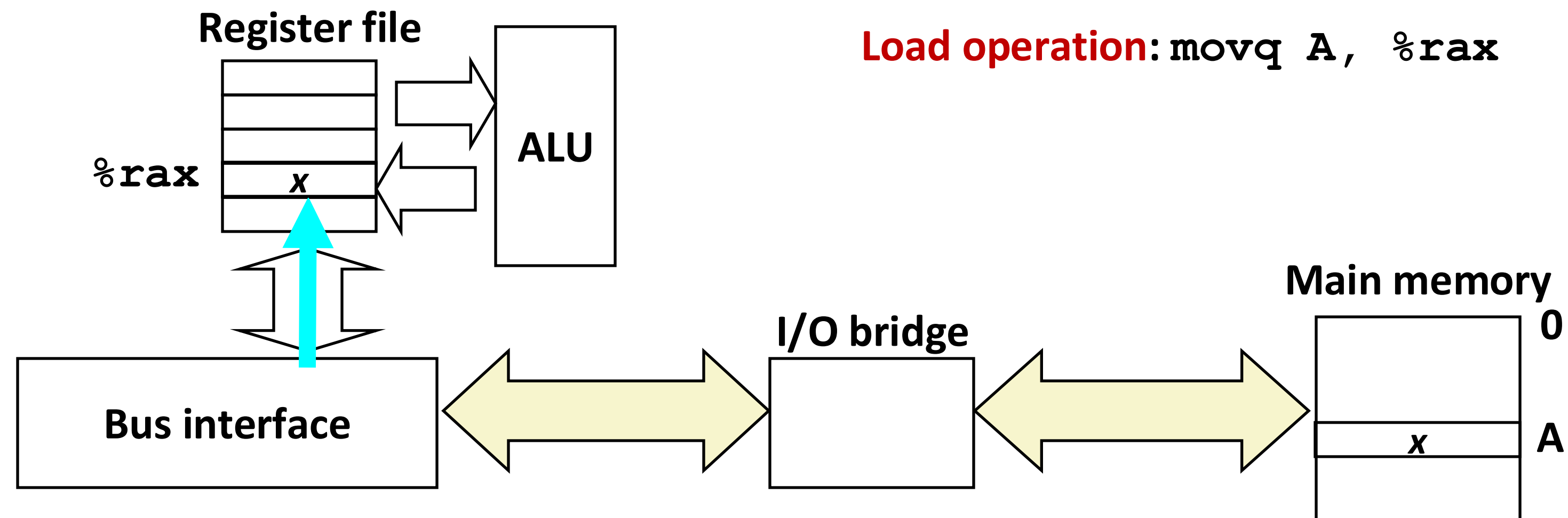
# Memory Read Transaction (2)



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

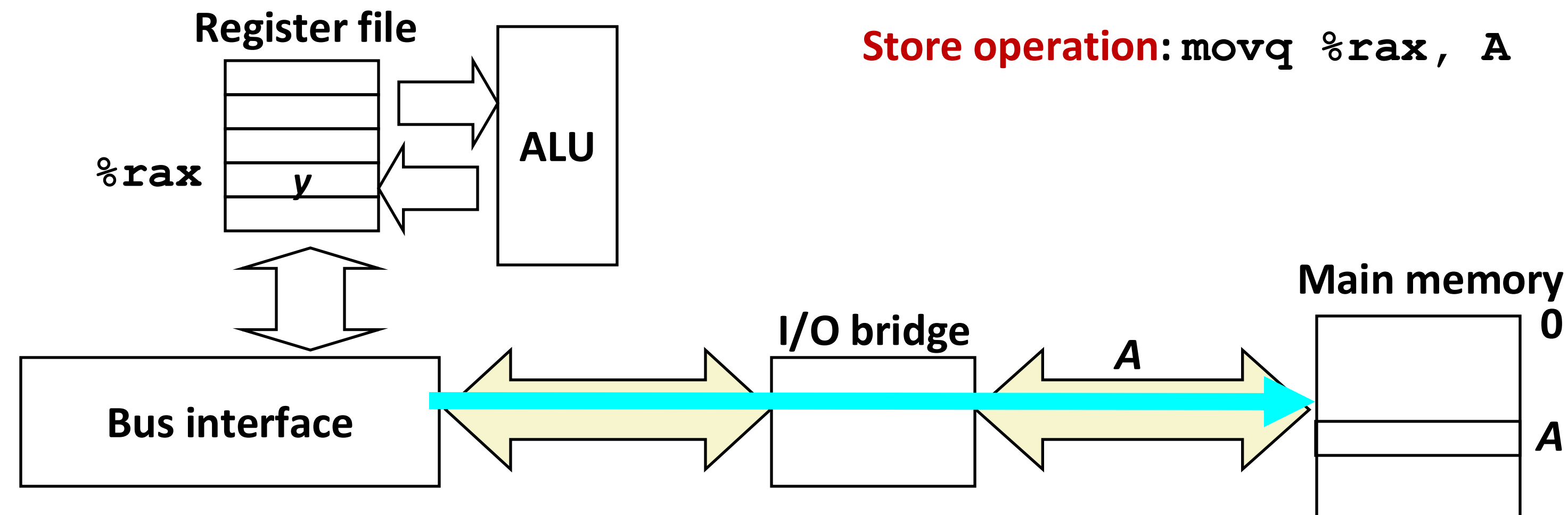


# Memory Read Transaction (3)



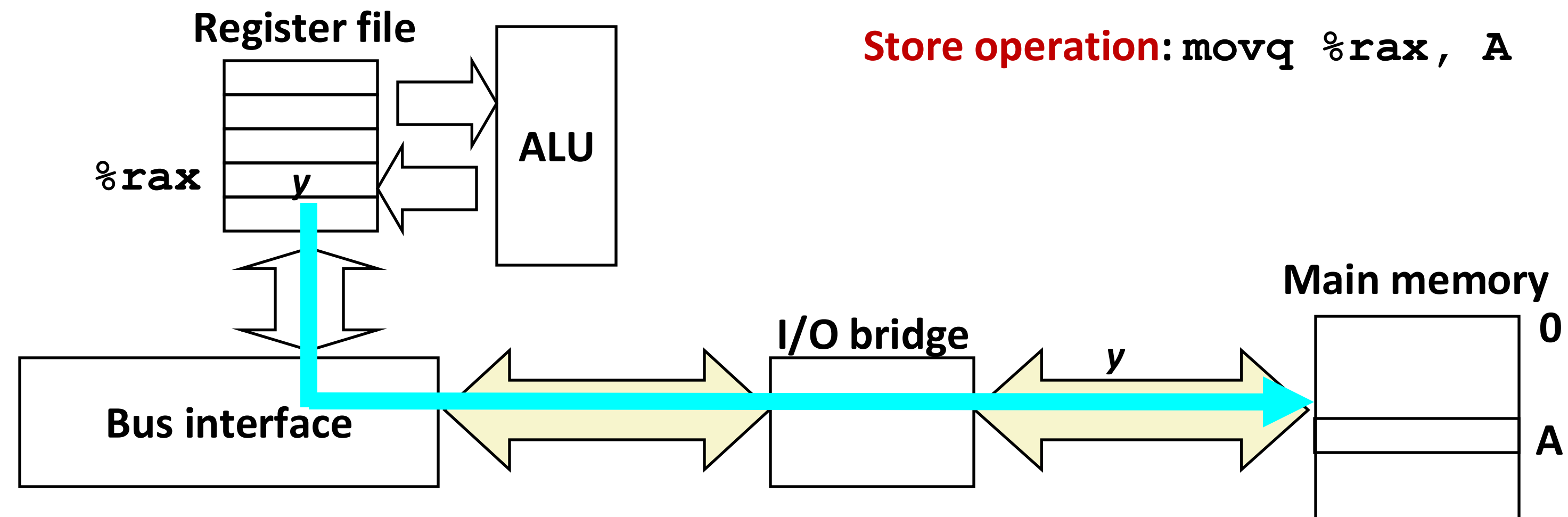
- CPU reads word `x` from the bus and copies it into register `%rax`.

# Memory Write Transaction (1)



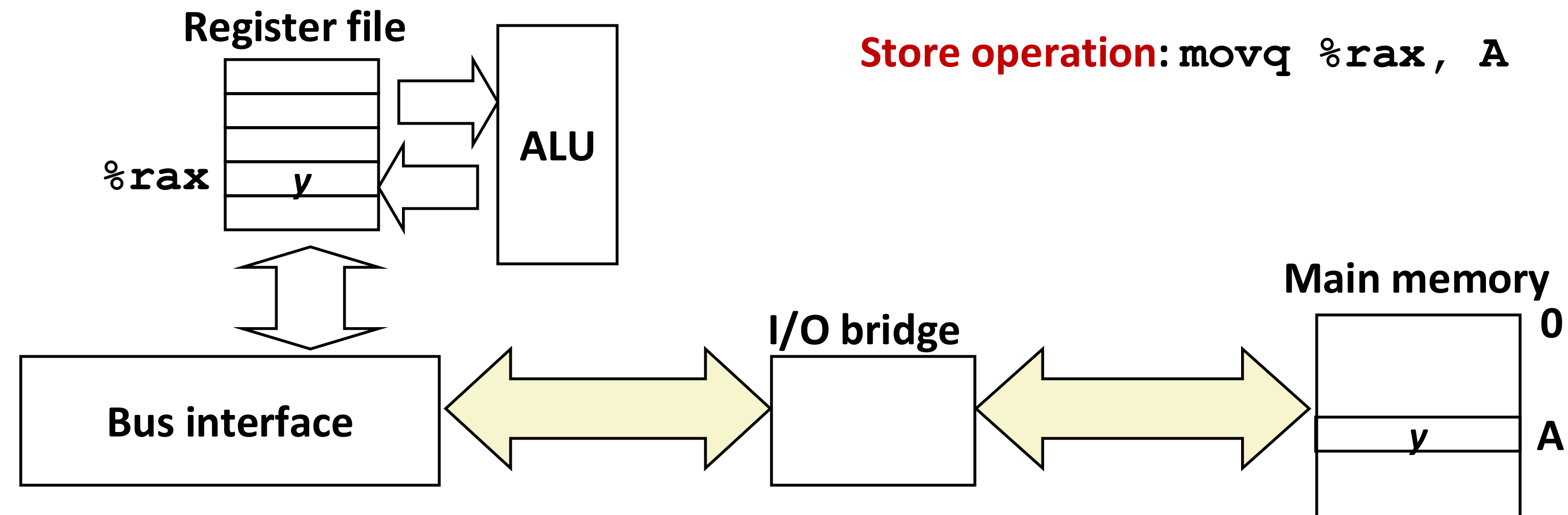
- CPU places address `A` on bus. Main memory reads it and waits for the corresponding data word to arrive.

# 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



You

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

# What is GPT doing?

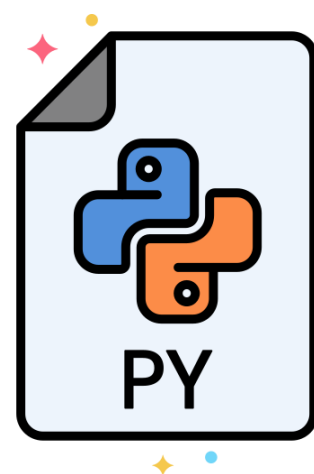
[0, 500, 32768, 1008, 922, ...]

List[integers]

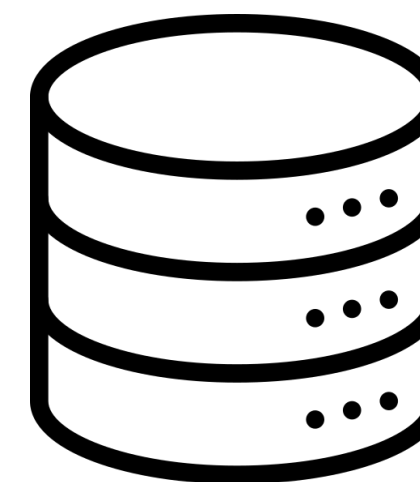


## GPT =

### Disk

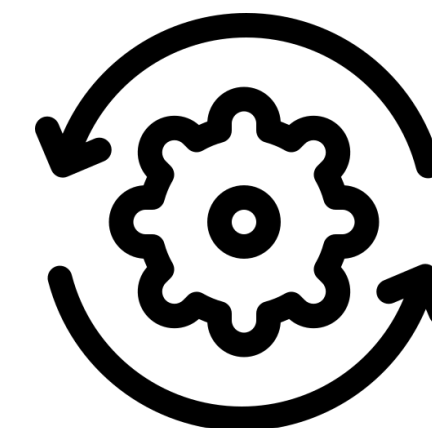


+



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

