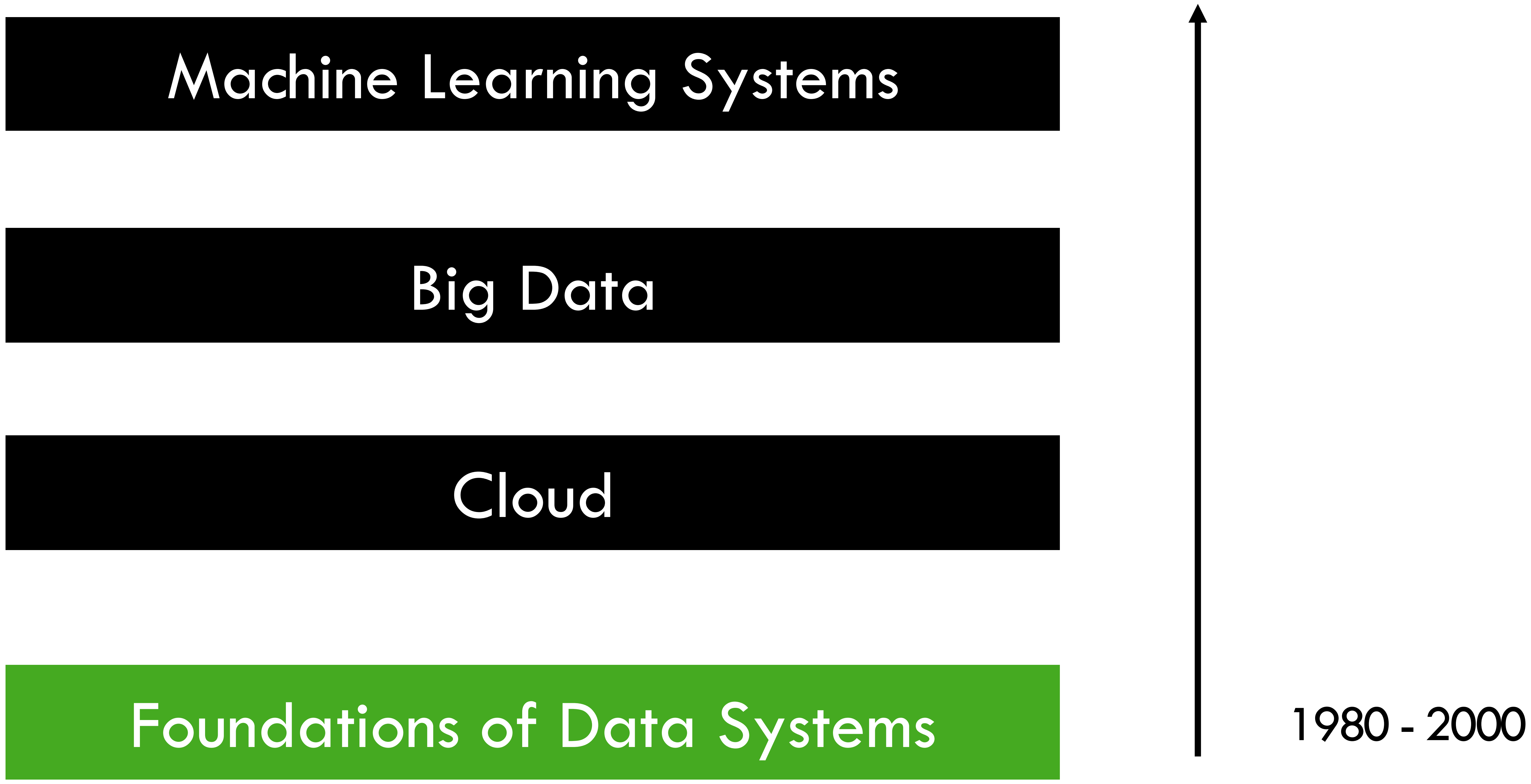
https://hao-ai-lab.github.io/dsc204a-f25/

DSC 204A: Scalable Data Systems
Fall 2025

Staff
Instructor: Hao Zhang
TAs: Mingjia Huo, Yuxuan Zhang

@haozhangml @haoailab
haozhang@ucsd.edu

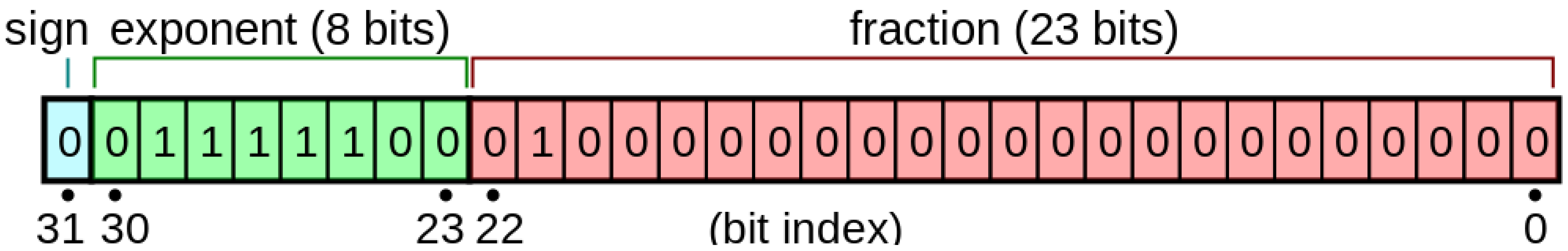
Where We Are



Machine Learning Systems
Big Data
Cloud
Foundations of Data Systems
1980 - 2000

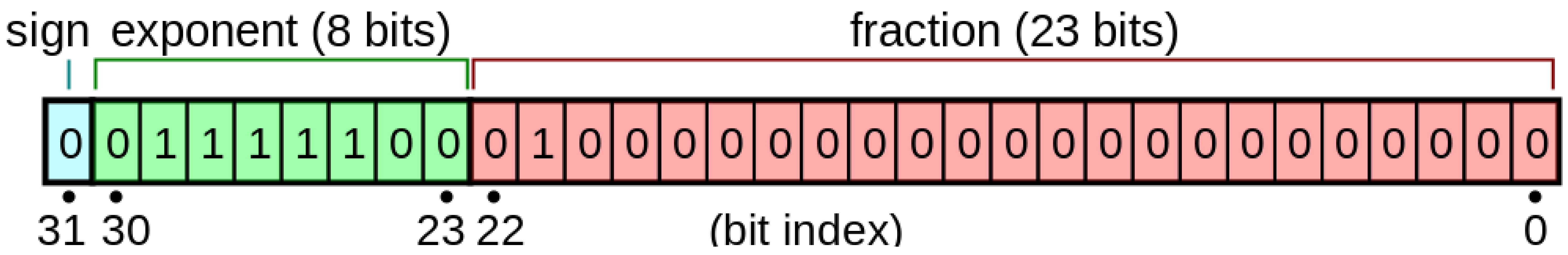
Q2: What does **exponent** and **fraction** control?

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



Digital Representation of Data: Bias

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

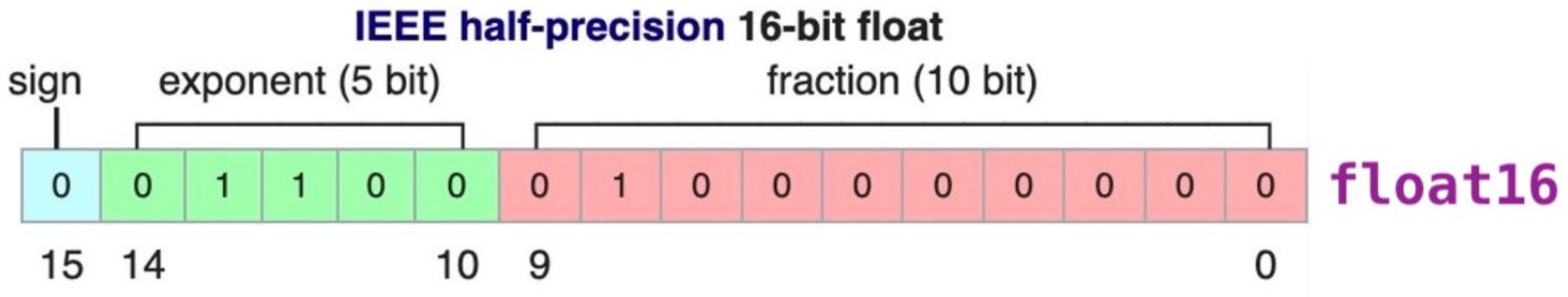


Why?

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

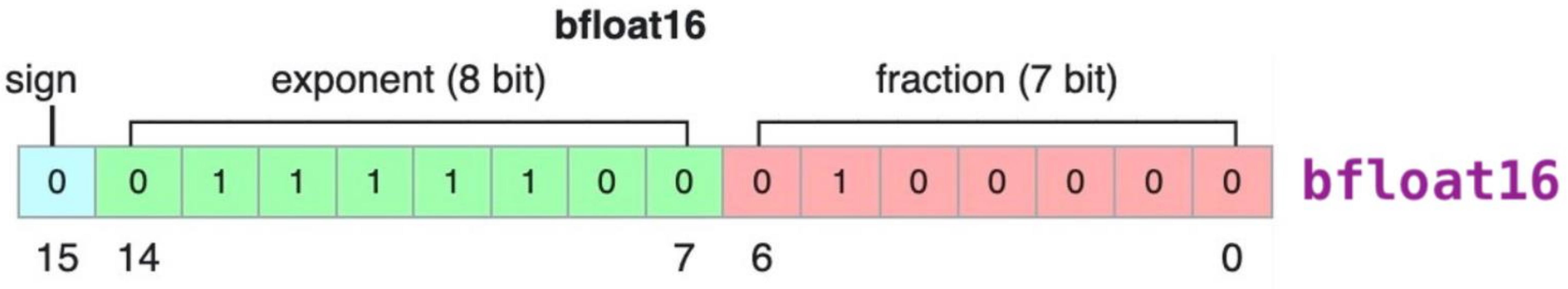
$$(-1)^0 \times 2^{124-127} \times (1 + 1 \cdot 2^{-2}) = (1/8) \times (1 + (1/4)) = 0.15625$$

Q3: What is the difference between BF16 and FP16?



Less exponent -> smaller range -> easier to overflow

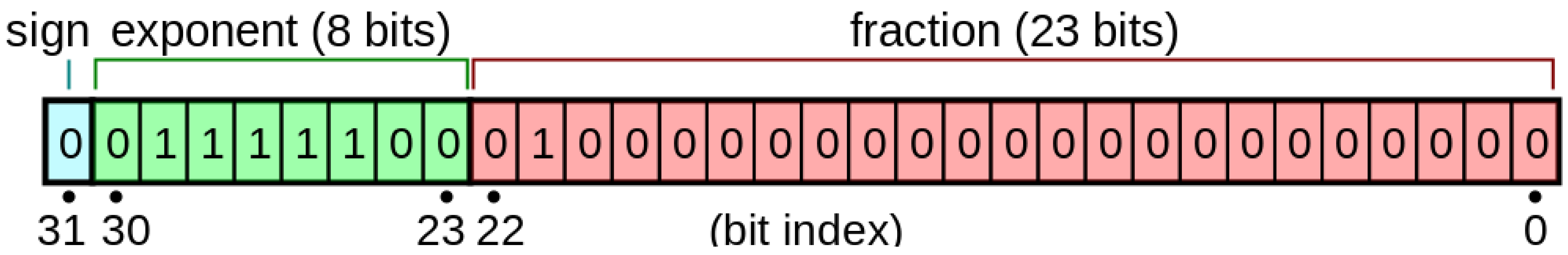
More fraction -> more precise



more exponent -> larger range -> harder to overflow

less fraction -> less precise

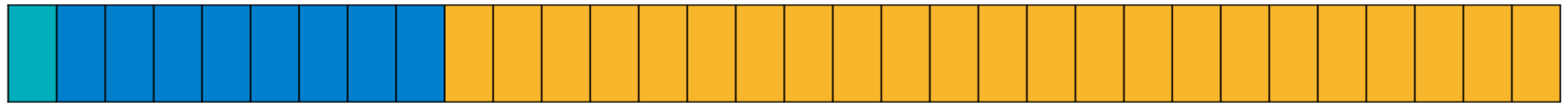
Floating-point Representation



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

Q: How to represent 0?

Floating-point Number: normal vs. subnormal



Sign **8 bit Exponent** **23 bit Fraction**

$(-1)^{sign} \times (1 + \textbf{Fraction}) \times 2^{\textbf{Exponent}-127}$

(Normal Numbers, Exponent≠0)

0 | 0 1 1 1 1 1 0 1 | 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

$\underbrace{01111101}_{125}$ $\underbrace{00010000000000000000000}_{0.0625}$

$0.265625 = 1.0625 \times 2^{-2} = (1 + \underline{0.0625}) \times 2^{\underline{125}-127}$

Should have been $(-1)^{sign} \times (1 + \textbf{Fraction}) \times 2^{0-127}$

But we force to be $(-1)^{sign} \times \textbf{Fraction} \times 2^{1-127}$

(Subnormal Numbers, Exponent=0)

0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

$\underbrace{00000000}_{0}$ $\underbrace{00000000000000000000000}_{0}$

1 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

$\underbrace{00000000}_{0}$ $\underbrace{00000000000000000000000}_{0}$

$0 = 0 \times 2^{-126}$

Q: What is the minimum positive value?

What is the minimum positive value?

Sign **8 bit Exponent** **23 bit Fraction**

$(-1)^{sign} \times (1 + \textbf{Fraction}) \times 2^{\textbf{Exponent}-127}$

(Normal Numbers, Exponent≠0)

$0\,\underbrace{00000001}_{1}\,\underbrace{00000000000000000000000}_{0}$

$2^{-126} = (1 + \underline{0}) \times 2^{\underline{1}-127}$

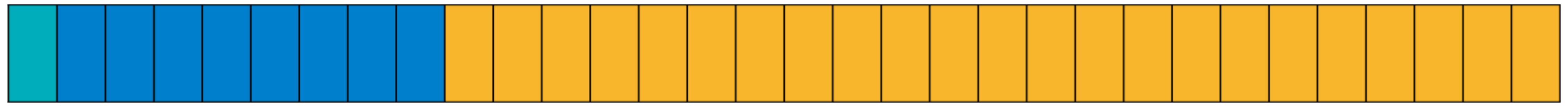
$(-1)^{sign} \times \textbf{Fraction} \times 2^{1-127}$

(Subnormal Numbers, Exponent=0)

$0\,\underbrace{00000000}_{0}\,\underbrace{00000000000000000000001}_{2^{-23}}$

$2^{-149} = 2^{-23} \times 2^{-126}$

Some Special Values



Sign 8 bit Exponent **23 bit Fraction**

$(-1)^{\text{sign}} \times (1 + \textbf{Fraction}) \times 2^{\textbf{Exponent}-127}$

(Normal Numbers, Exponent≠0)

$(-1)^{\text{sign}} \times \textbf{Fraction} \times 2^{1-127}$

(Subnormal Numbers, Exponent=0)

0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

+∞ (positive infinity)

1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

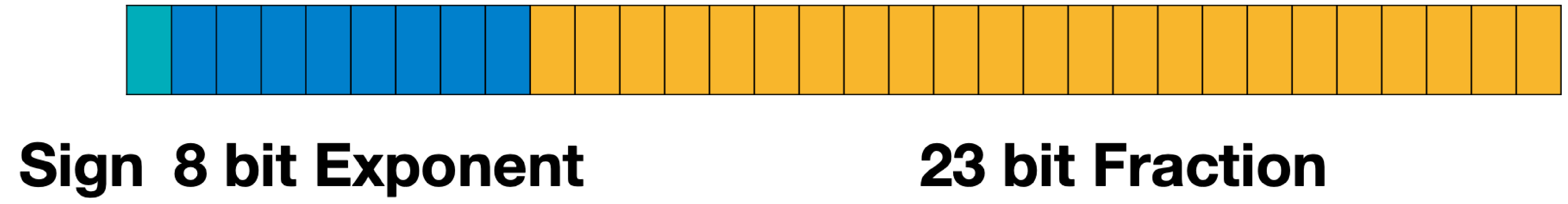
-∞ (negative infinity)

\- 1 1 1 1 1 1 1 1 -

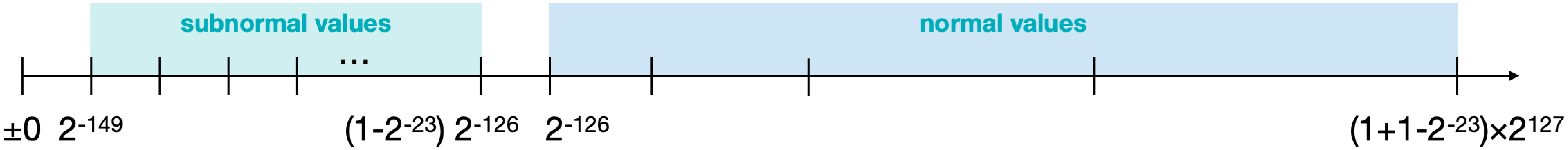
NaN (Not a Number)

much waste. Revisit in fp8.

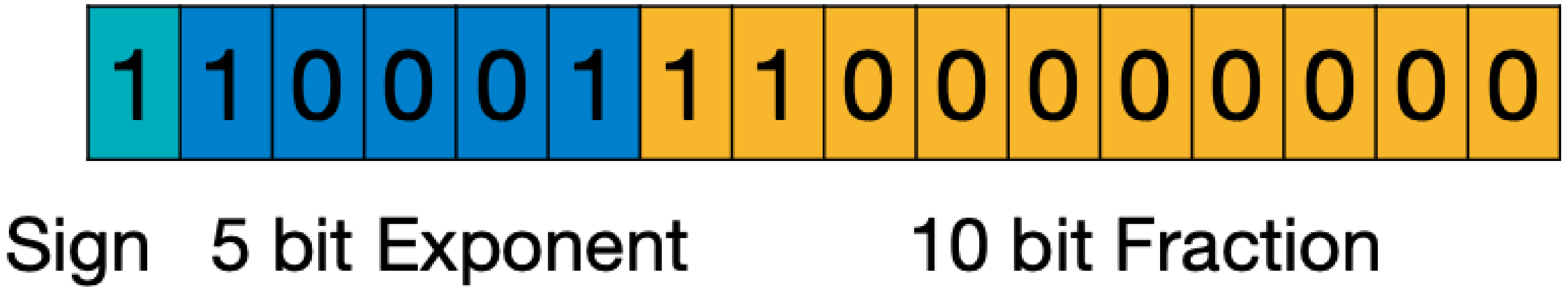
Summary of fp32



Exponent	Fraction=0	Fraction≠0	Equation
$00_H = 0$	±0	subnormal	$(-1)^{sign} \times \textbf{Fraction} \times 2^{1-127}$
$01_H \ldots FE_H = 1 \ldots 254$	normal		$(-1)^{sign} \times (1 + \textbf{Fraction}) \times 2^{\textbf{Exponent}-127}$
$FF_H = 255$	±INF	NaN	



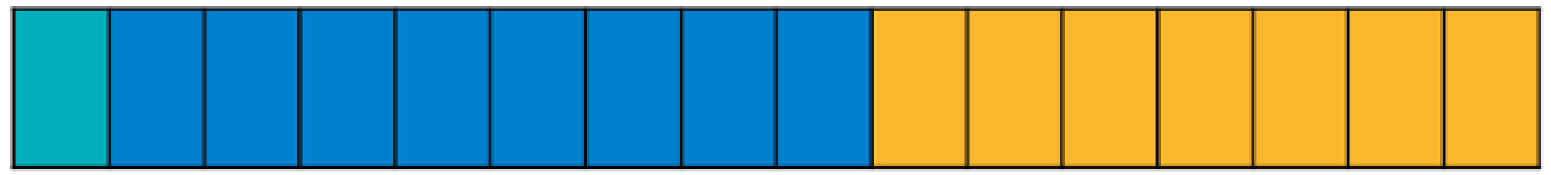
Exercise



- Sign: -
- Exponent
  - Bias:$2^4 - 1 = 15_{10}$
  - $10001_2 - 15_{10} = 17_{10} - 15_{10} = 2_{10}$
- Fraction
  - $1100000000_2 = 0.75_{10}$
- Answer: $-(1 + 0.75) \times 2^2 = -7.10_{10}$

$(-1)^{\text{sign}} \times (1 + \textbf{Fraction}) \times 2^{\textbf{Exponent}-127}$

Google Brain Float (BF16)



Exercise

What is Decimal 2.5 in BF16?

- $2.5 = 1.25 \times 2^1$
- Sign: +
- Exponent: bias is $2^7 - 1 = 127$
  - $x - 127 = 1; x = 128_{10} = 10000000_2$
- Fraction: 7-bit fraction
  - $0.25 = 0100000_2$

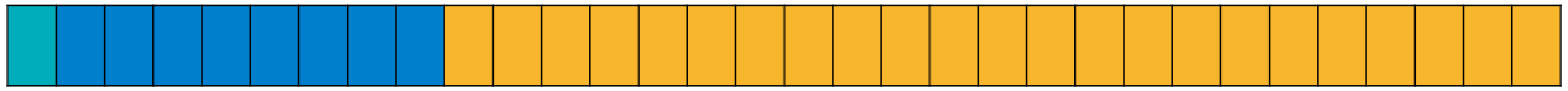
0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0

Sign 8 bit Exponent 7 bit Fraction

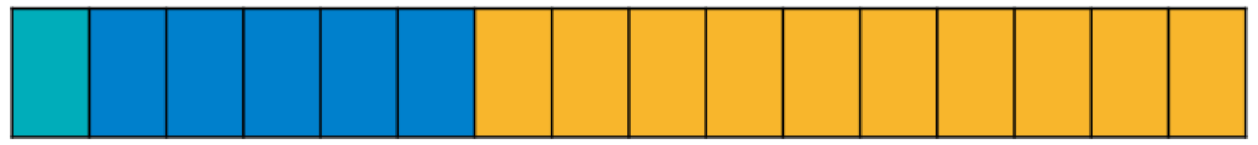
Latest FP8

- Exponent width -> Range; Fraction width -> precision

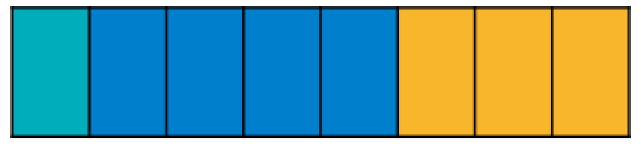
[IEEE 754](#) Single Precision 32-bit Float (IEEE FP32)



[IEEE 754](#) Half Precision 16-bit Float (IEEE FP16)

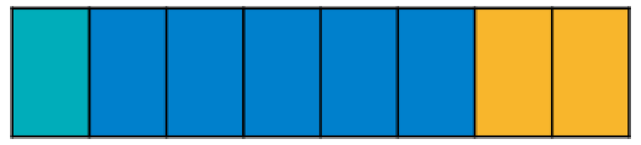


[Nvidia](#) FP8 (E4M3)



* FP8 E4M3 does not have INF, and $S.1111.111_2$ is used for NaN.
* Largest FP8 E4M3 normal value is $S.1111.110_2 = 448$.

[Nvidia](#) FP8 (E5M2) for gradient in the backward



* FP8 E5M2 have INF ($S.11111.00_2$) and NaN ($S.11111.XX_2$).
* Largest FP8 E5M2 normal value is $S.11110.11_2 = 57344$.

Format	Exponent (bits)	Fraction (bits)	Total (bits)
IEEE FP32	8	23	32
IEEE FP16	5	10	16
FP8 (E4M3)	4	3	8
FP8 (E5M2)	5	2	8

FP4

Introducing NVFP4 for Efficient and Accurate Low-Precision Inference

2025-9-30

Pretraining Large Language Models with NVFP4

NVIDIA

Abstract. Large Language Models (LLMs) today are powerful problem solvers across many domains, and they continue to get stronger as they scale in model size, training set size, and training set quality, as shown by extensive research and experimentation across the industry. Training a frontier model today requires on the order of tens to hundreds of yottaflops, which is a massive investment of time, compute, and energy. Improving pretraining efficiency is therefore essential to enable the next generation of even more capable LLMs. While 8-bit floating point (FP8) training is now widely adopted, transitioning to even narrower precision, such as 4-bit floating point (FP4), could unlock additional improvements in computational speed and resource utilization. However, quantization at this level poses challenges to training stability, convergence, and implementation, notably for large-scale models trained on long token horizons.

In this study, we introduce a novel approach for stable and accurate training of large language models (LLMs) using the NVFP4 format. Our method integrates Random Hadamard transforms (RHT) to bound block-level outliers, employs a two-dimensional quantization scheme for consistent representations across both the forward and backward passes, utilizes stochastic rounding for unbiased gradient estimation, and incorporates selective high-precision layers. We validate our approach by training a 12-billion-parameter model on 10 trillion tokens – the longest publicly documented training run in 4-bit precision to date. Our results show that the model trained with our NVFP4-based pretraining technique achieves training loss and downstream task accuracies comparable to an FP8 baseline. For instance, the model attains an MMLU-pro accuracy of 62.58%, nearly matching the 62.62% accuracy achieved through FP8 pretraining. These findings highlight that NVFP4, when combined with our training approach, represents a major step forward in narrow-precision LLM training algorithms.

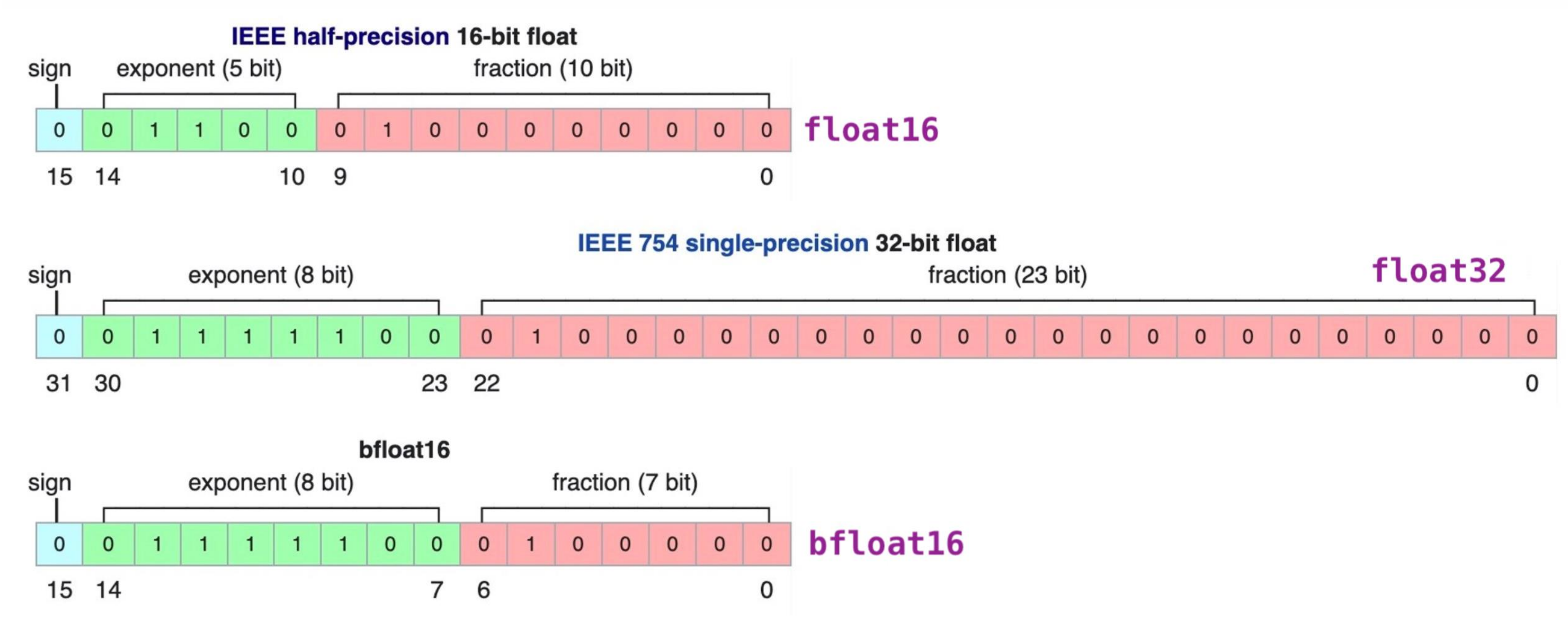
Code: Transformer Engine support for NVFP4 training.

1. Introduction

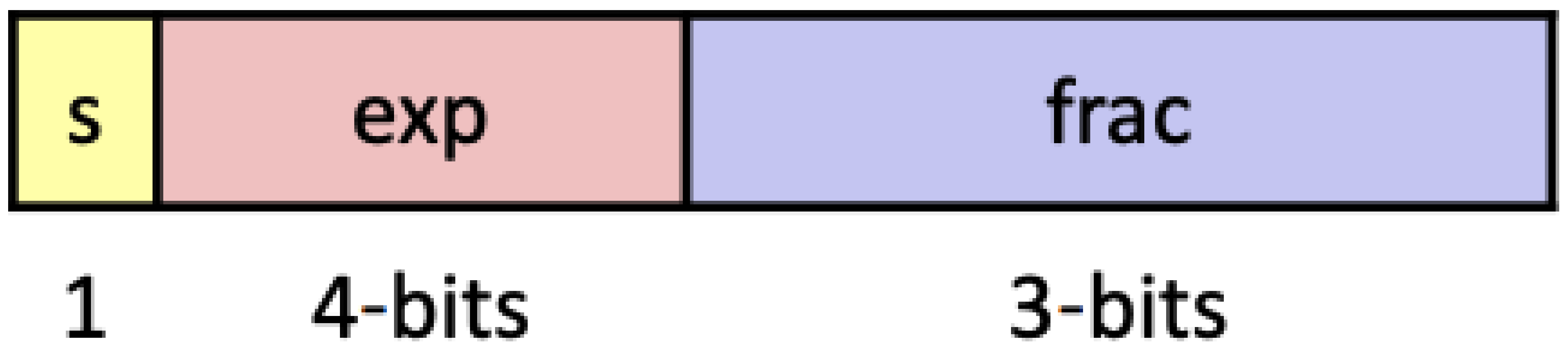
The rapid expansion of large language models (LLMs) has increased the demand for more efficient numerical formats to lower computational cost, memory demand, and energy consumption during training. 8-bit floating point (FP8 and MXFP8) has emerged as a popular data type for accelerated training of LLMs (Micikevicius et al., 2022; DeepSeek-AI et al., 2024; Mishra et al., 2025). Recent advances in narrow-precision hardware (NVIDIA Blackwell, 2024) have positioned 4-bit floating point (FP4) as the next logical step (Tseng et al., 2025b; Chmiel et al., 2025; Wang et al., 2025; Chen et al., 2025; Castro

Why BF16 is better in ML/AI?

1. Precision is enough. ML/AI is error-tolerant (why?)
2. Deep learning is easy to overflow
3. Closer range to fp32



Examples in the final exam: FP8



Demystify ChatGPT

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

str

↓

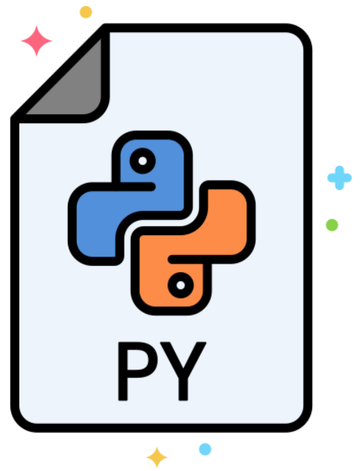
[0, 500, 32768, 1008, 922, ...]

List[integers]

↓

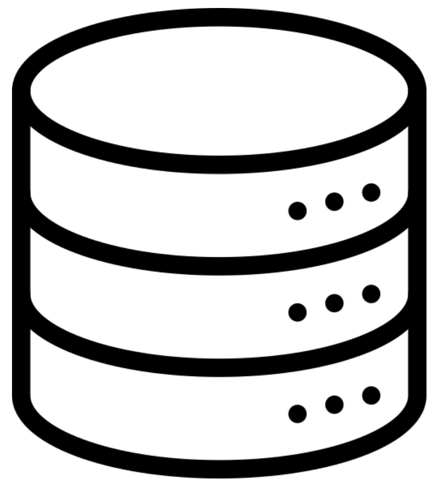
GPT =

Disk



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

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

Program in PL

↓ *Compile/Interpret*

Program in Assembly Language

↓ *Assemble*

Machine code tied to ISA

↓

Run on processor

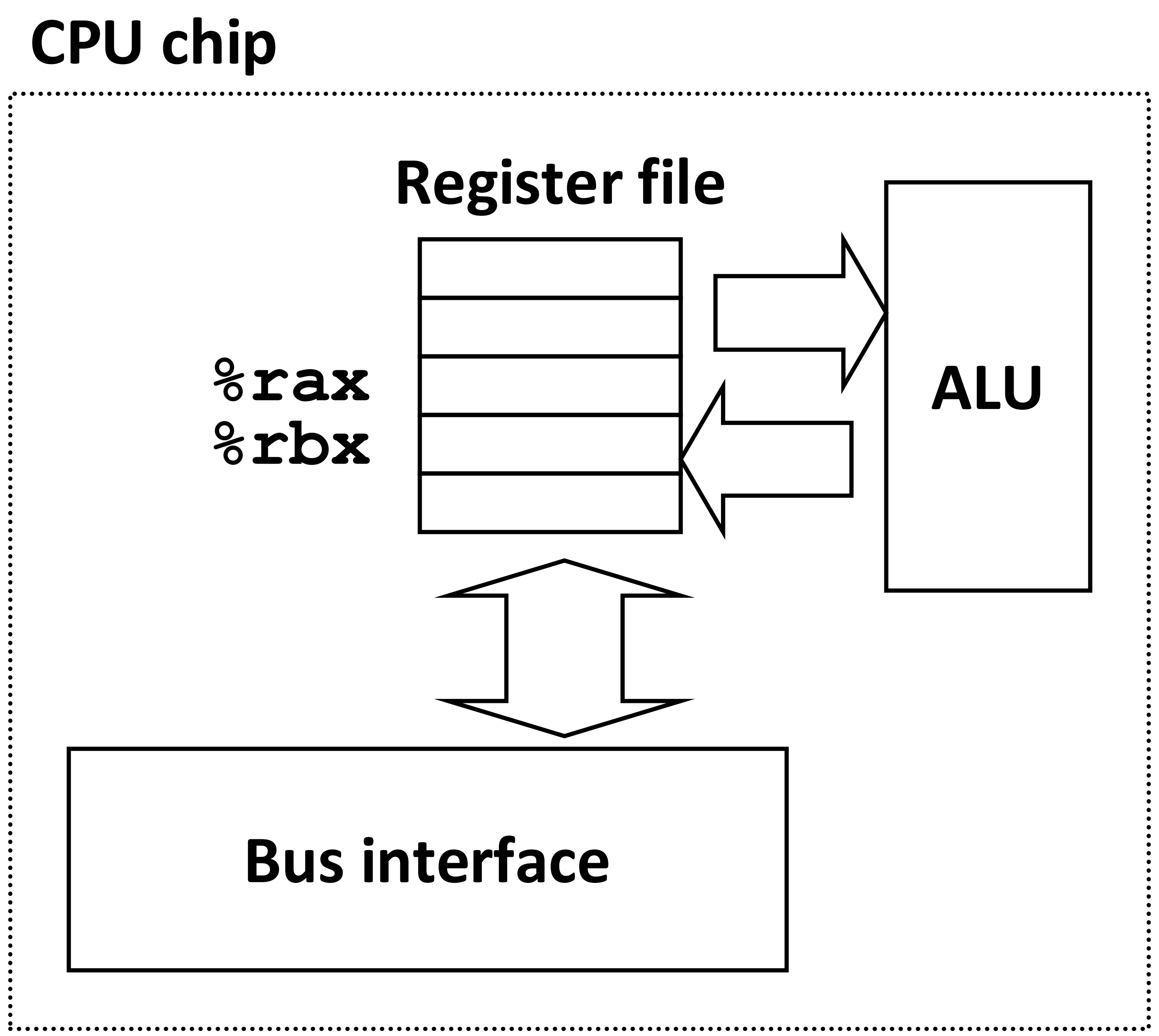
```
80483b4:       55                      push   %ebp
80483b5:       89 e5                   mov    %esp,%ebp
80483b7:       83 e4 f0                and    $0xfffffff0,%esp
80483ba:       83 ec 20                sub    $0x20,%esp
80483bd:       c7 44 24 1c 00 00 00    movl   $0x0,0x1c(%esp)
80483c4:       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
```

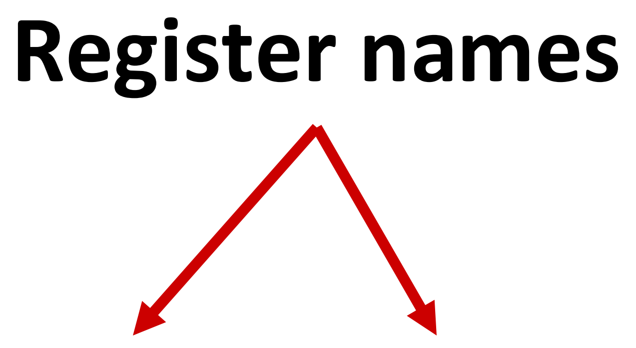
Basics of Processors

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






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

is

```
rax += rbx
```

How Fast is Processor (CPU and GPU)

- Instruction / second: number of instru

intel cpu floating point per seconds?

All Images Shopping Videos News More Tools

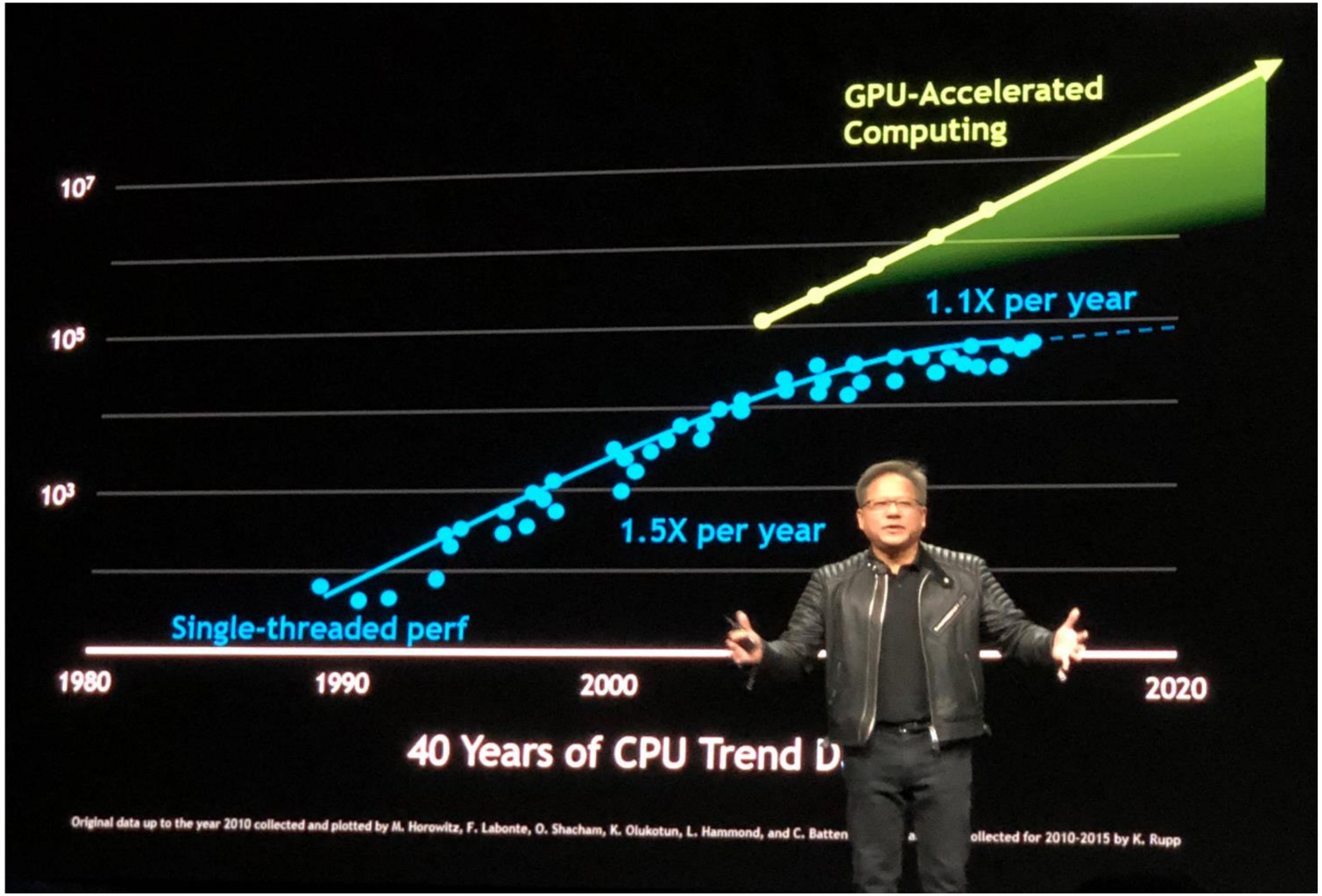
Generative AI is experimental. Learn more

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]

Moore's Law


GPU-Accelerated Computing
10^7
10^5
10^3
1.1X per year
1.5X per year
Single-threaded perf
1980
1990
2000
2020
40 Years of CPU Trend D
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
ollected for 2010-2015 by K. Rupp

The Real Problem?

100 GFLOPs/s

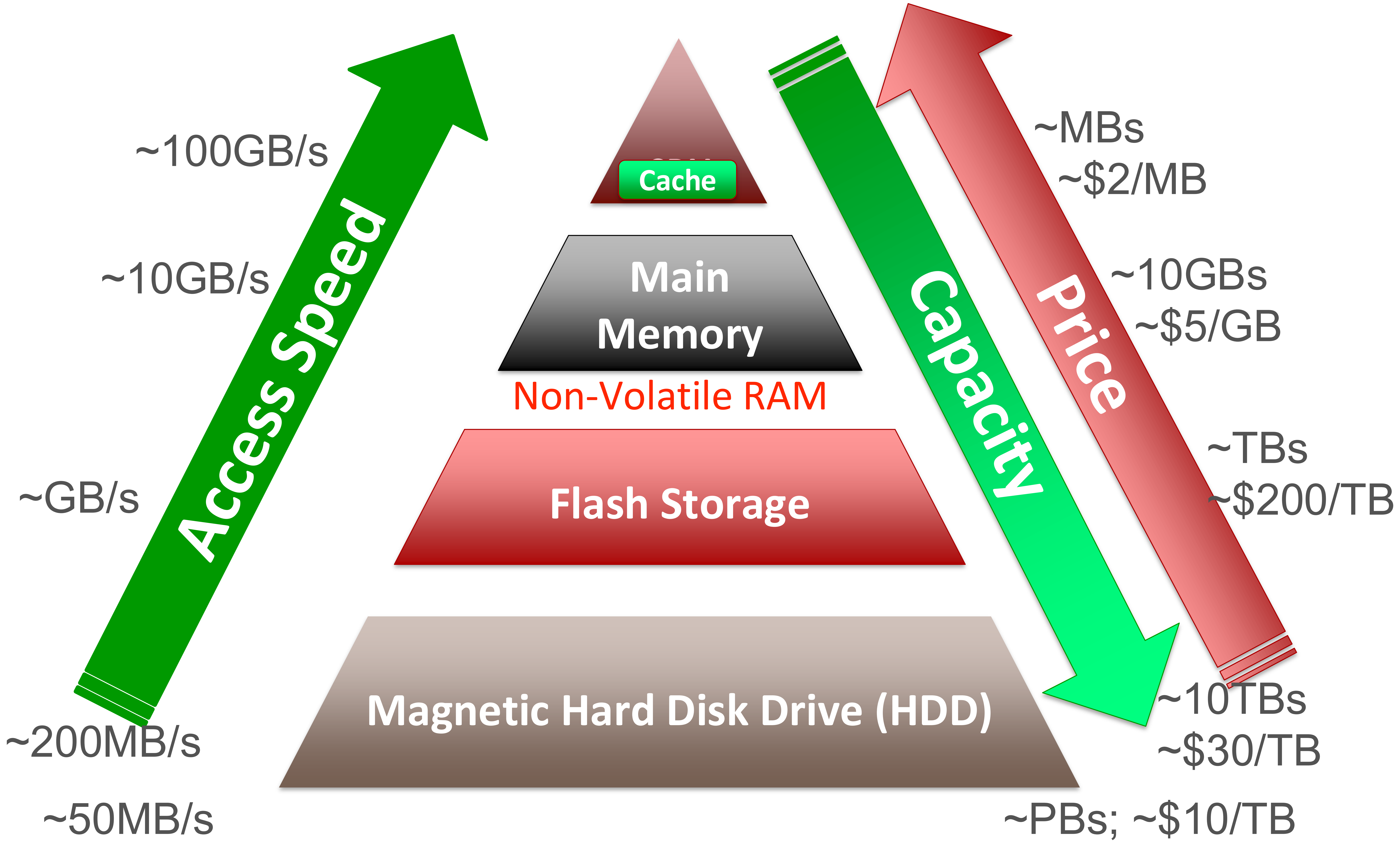
CPU

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

Magnetic Hard Disk Drive (HDD)

80 – 160 MB/s

Memory/Storage Hierarchy

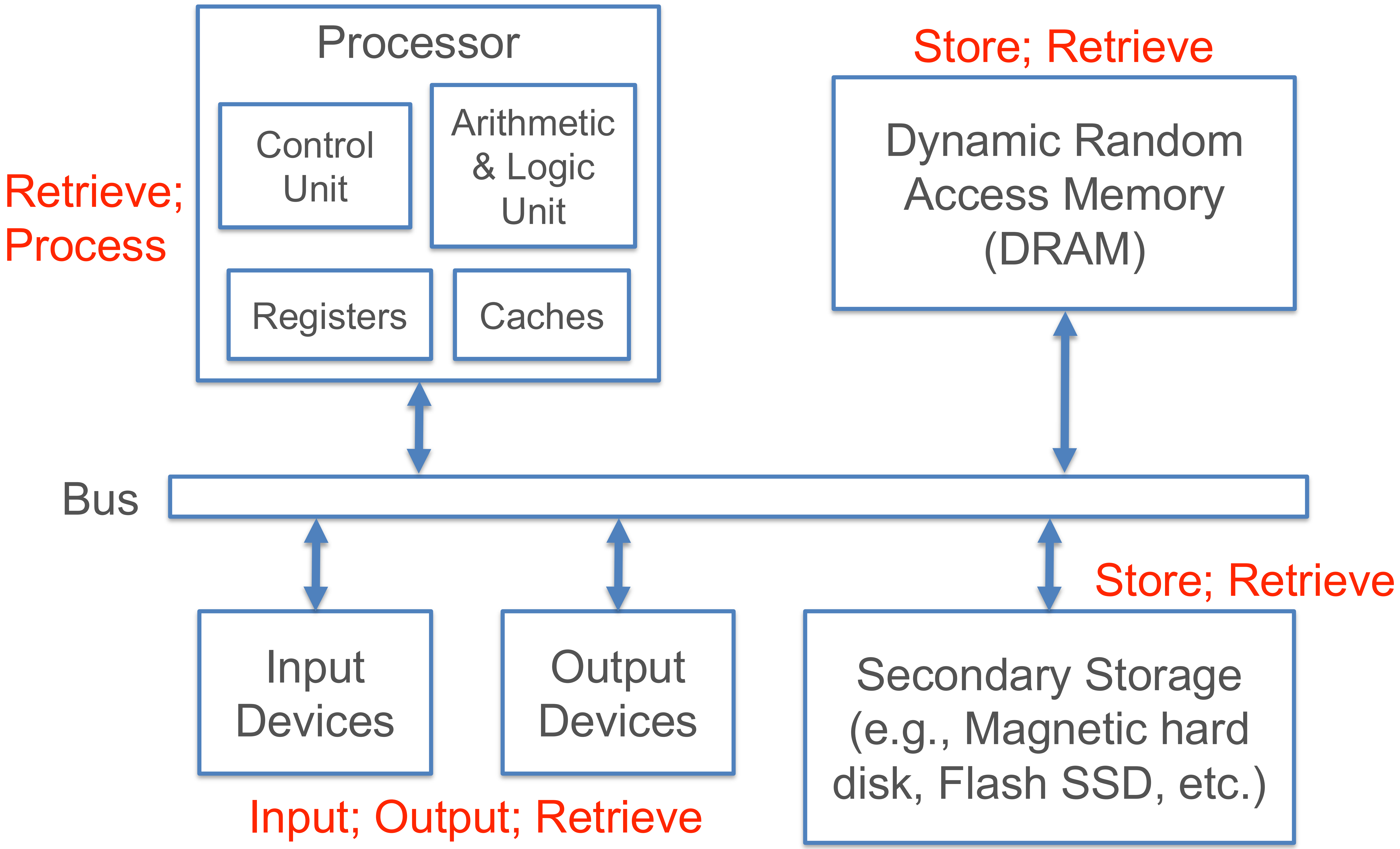


Writing & Reading Memory Instructions

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

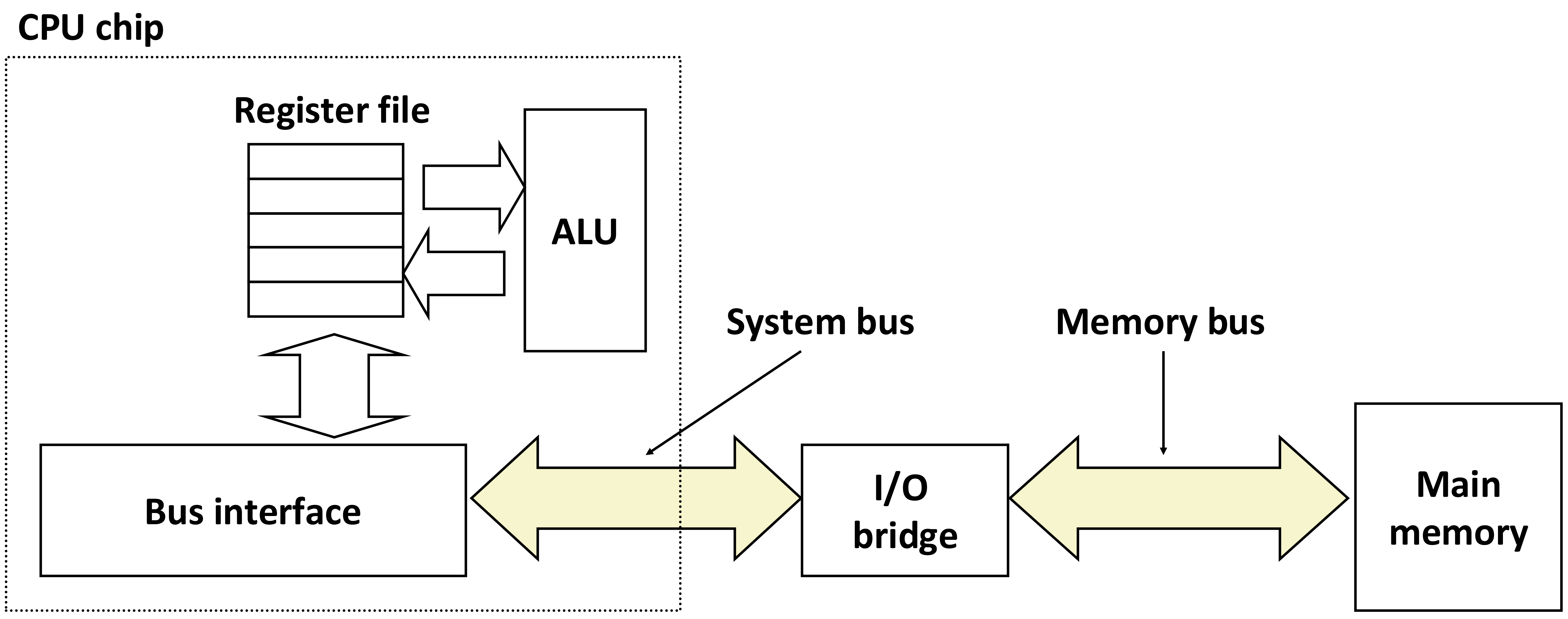
- Read
  - Transfer data from CPU to memory
    movq %rsp, %rax
  - “Load” operation

Abstract Computer Parts and Data

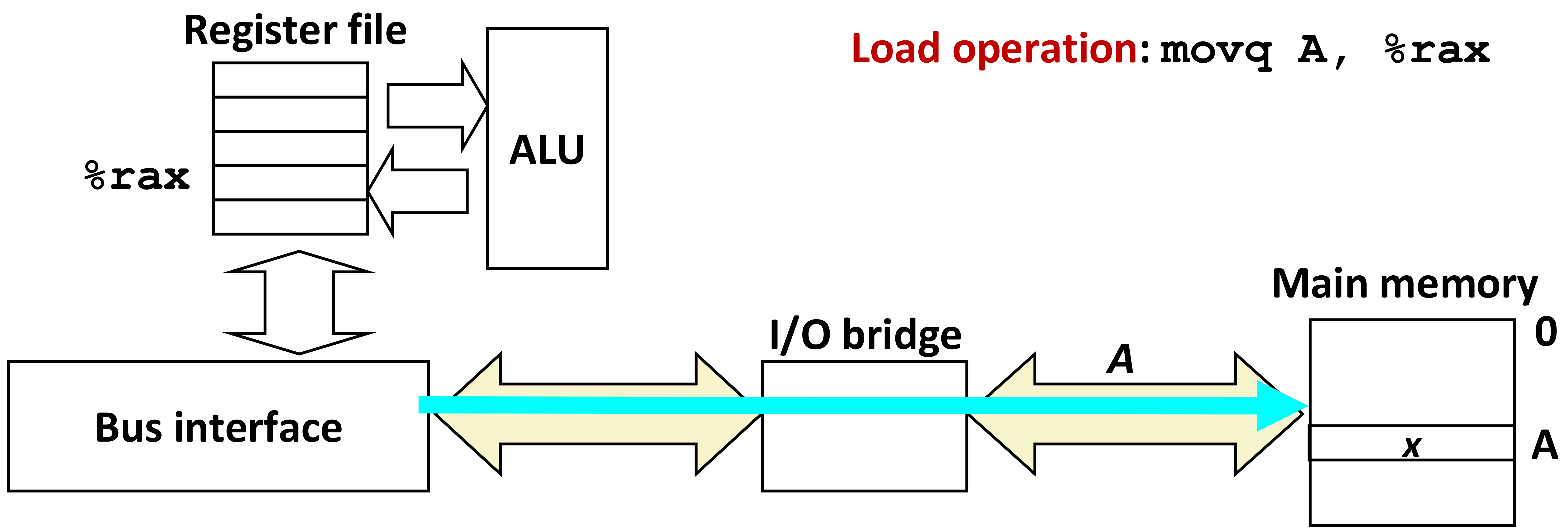


Bus Structure Connecting CPU and Memory

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

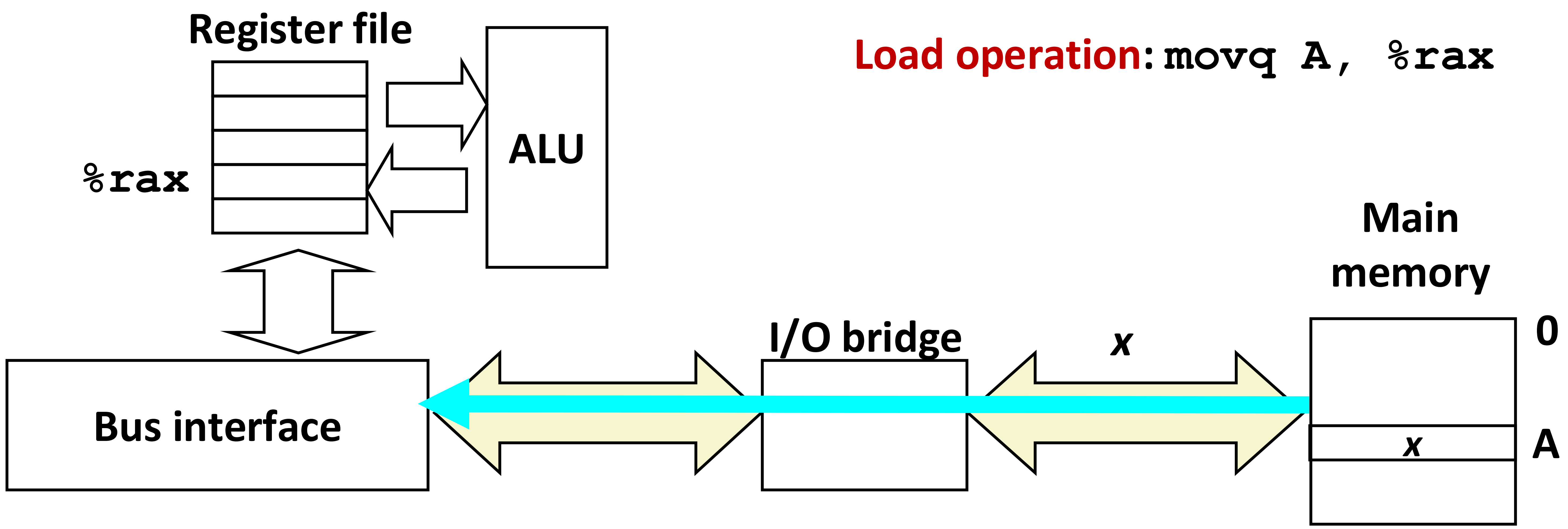


Memory Read Transaction (1)



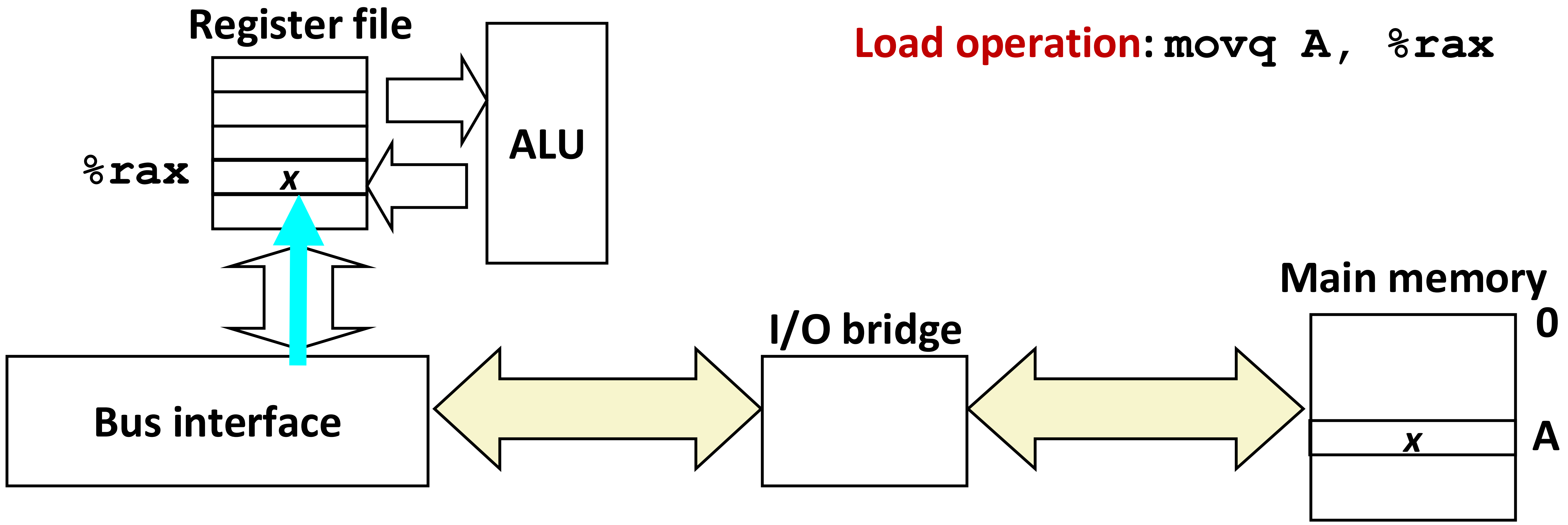
- CPU places address A on the memory bus.

Memory Read Transaction (2)



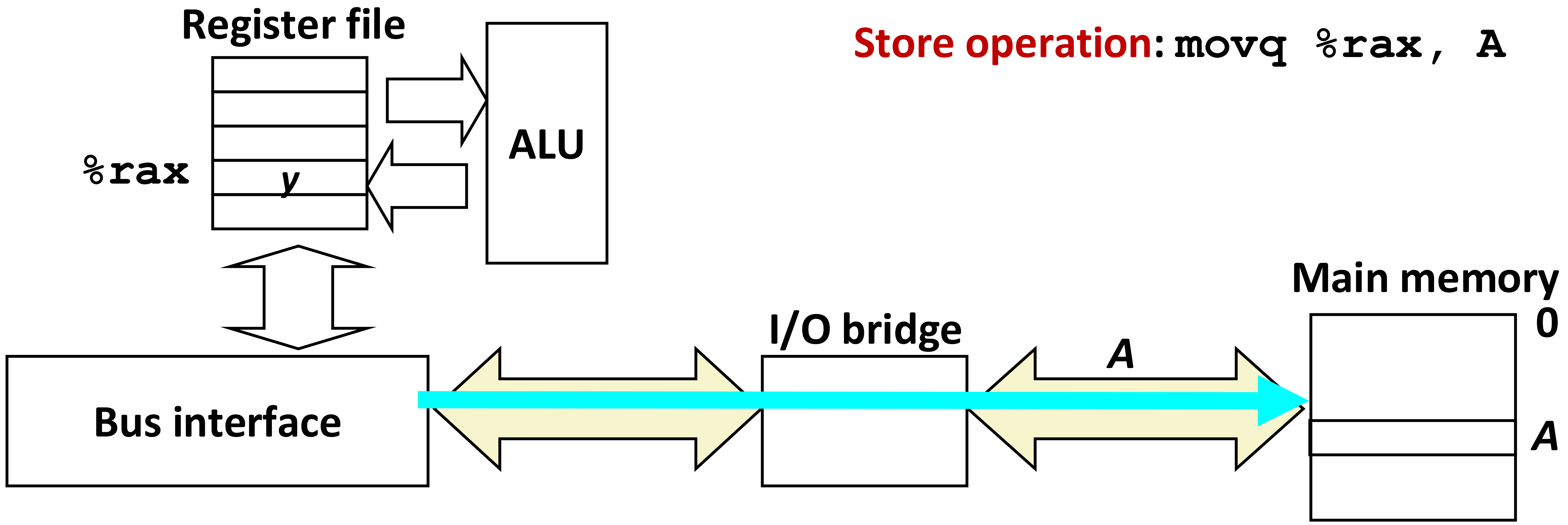
- Main memory reads A from the memory bus, retrieves word x, and places it on the bus.

Memory Read Transaction (3)



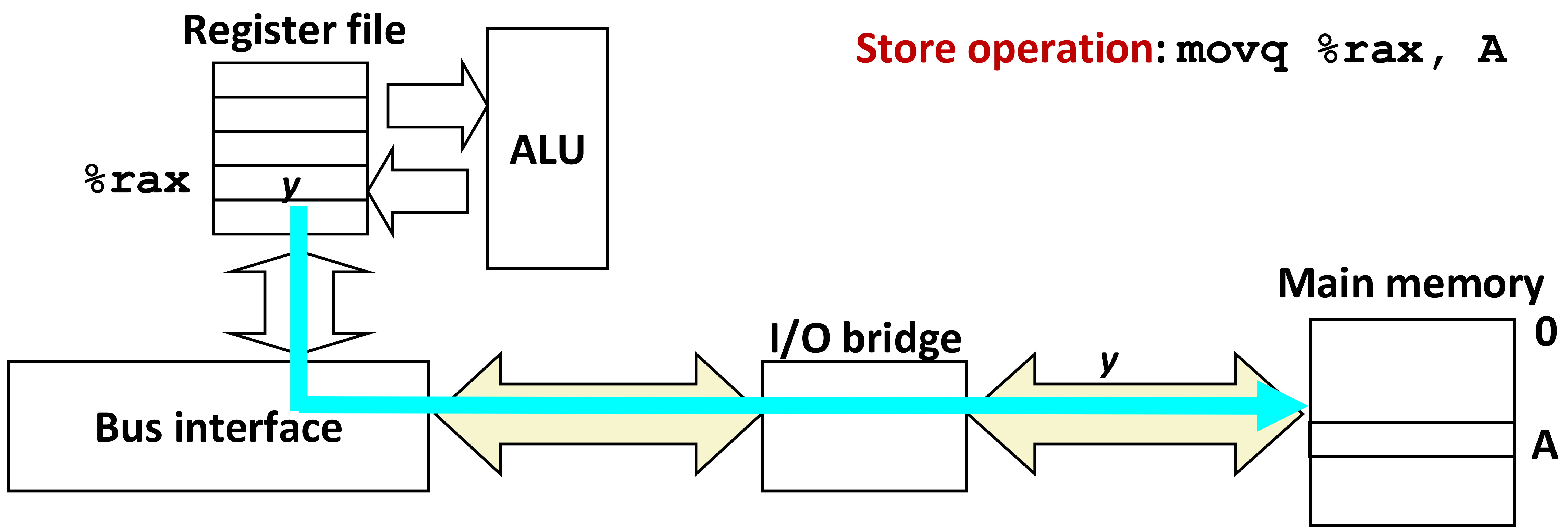
- CPU reads word x from the bus and copies it into register `%rax`.

Memory Write Transaction (1)



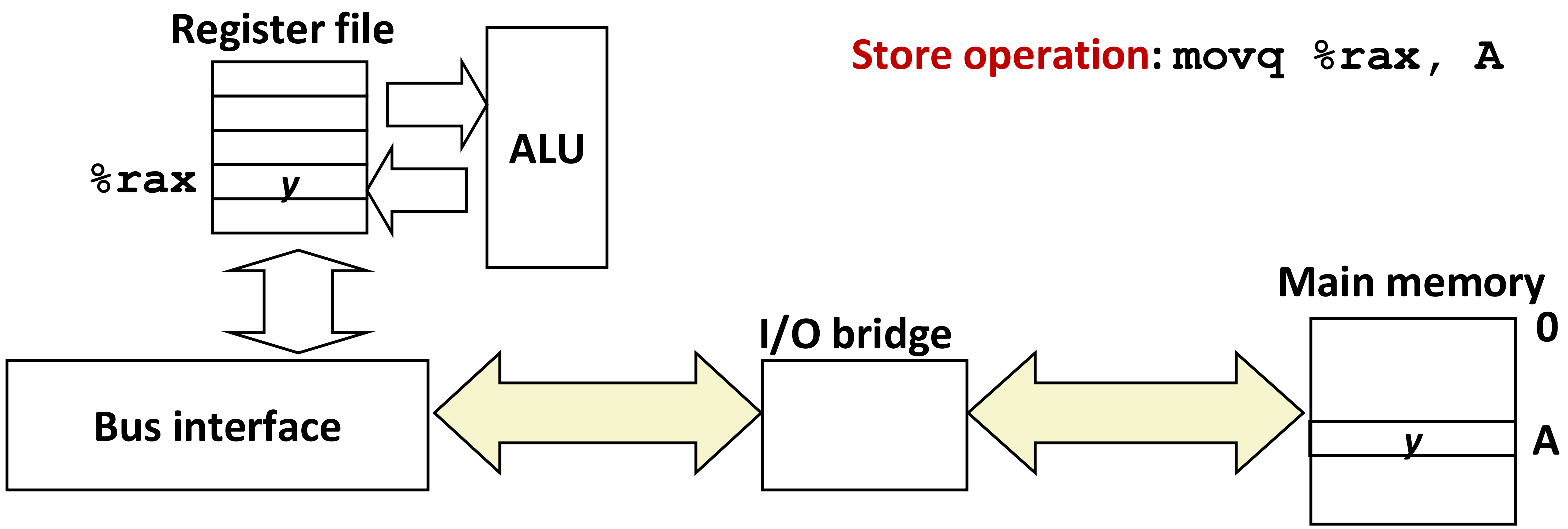
- CPU places address A on bus. Main memory reads it and waits for the corresponding data word to arrive.

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

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?

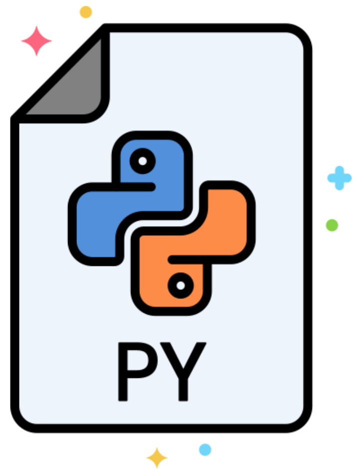
[0, 500, 32768, 1008, 922, ….]

List[integers]

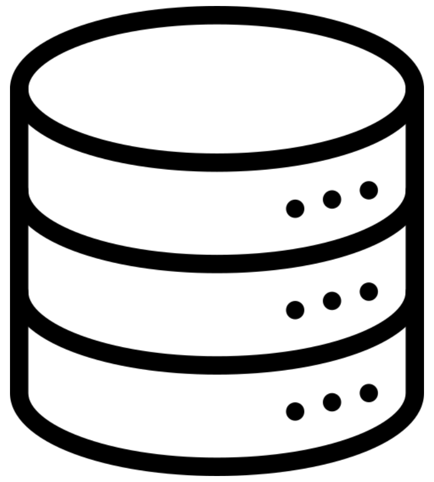
↓

GPT =

Disk

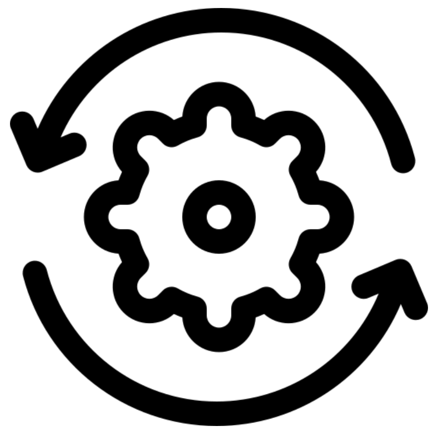


+



A few KBs

Parameters:
350 GB



↓

[0, 25116, 1234, 5984, 6, …]

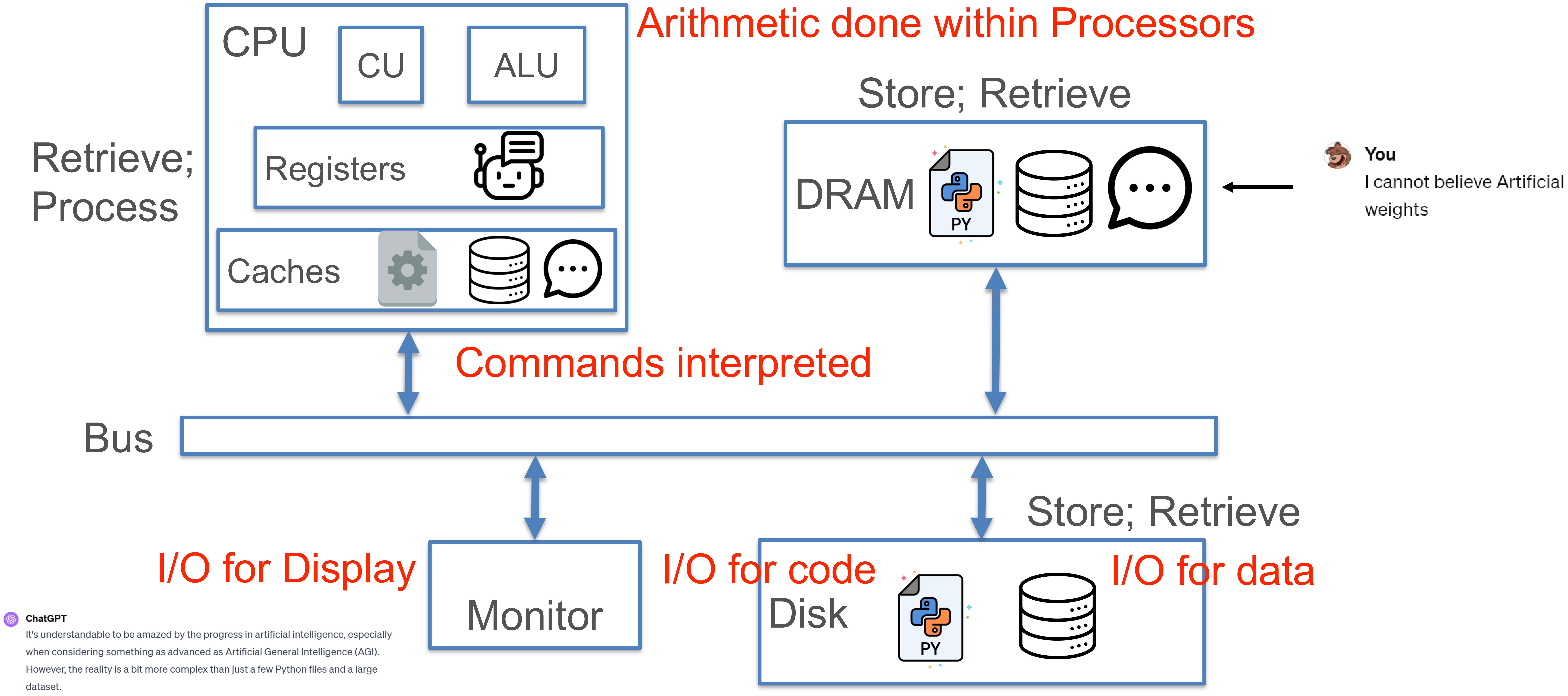
List[integers]

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.

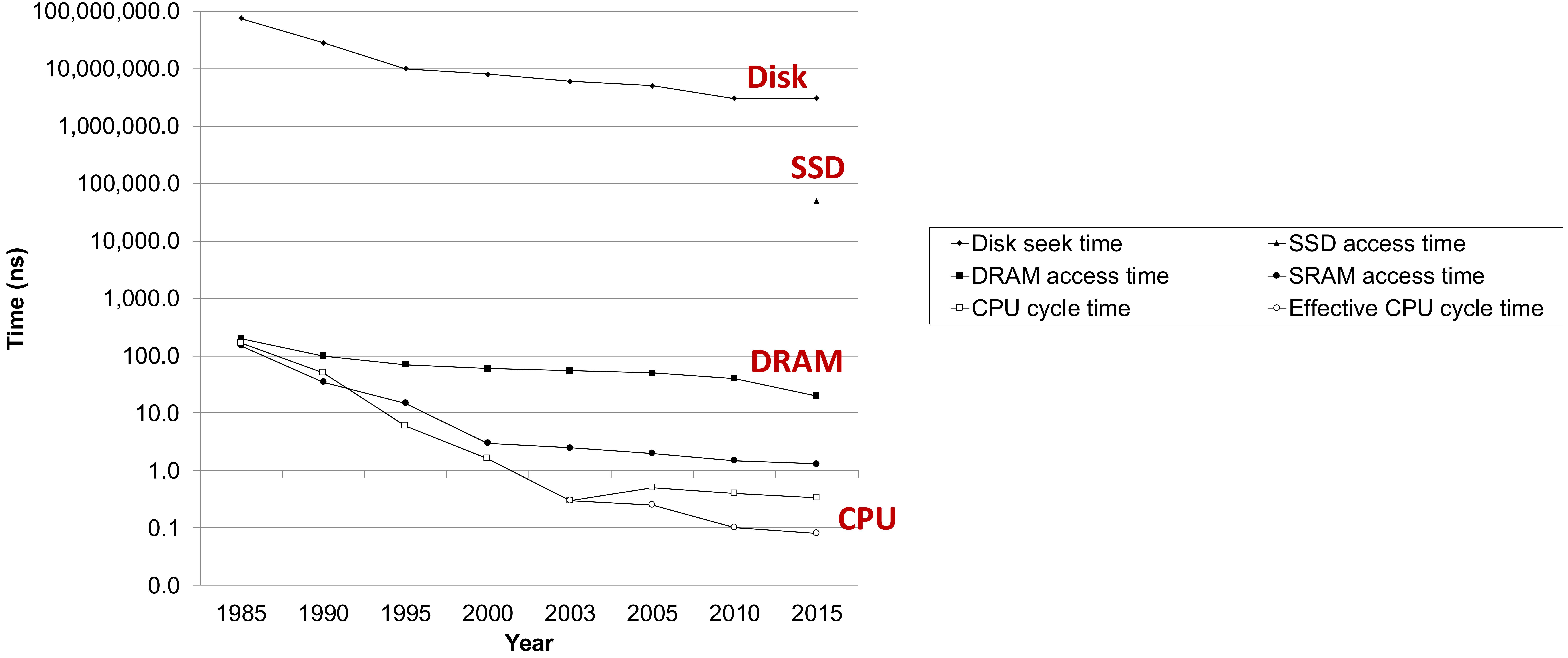
Example

But, how we can make this fast? What are potential problems here?



The CPU-Memory Gap

The gap *widens* between DRAM, disk, and CPU speeds.



Our problem, Simplified

Core Question behind Many System Research

How exactly memory hierarchy solves the gap?

Locality

- The key to bridging this CPU-Memory gap is an important property of computer programs known as locality.

copyij v.s copyji: copy a 2048 X 2048 integer array

```
void copyij(long int src[2048][2048], long int dst[2048][2048])
{
  long int i,j;
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}
```

4.3 milliseconds

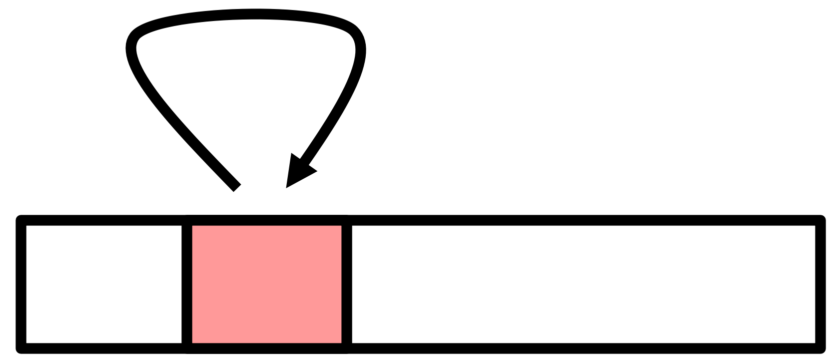
```
void copyji(long int src[2048][2048], long int dst[2048][2048])
{
  long int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

81.8 milliseconds

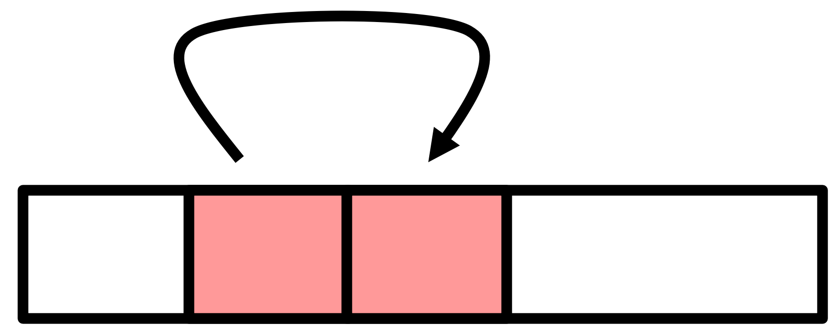
Locality

- Principle of Locality: Many Programs tend to use data and instructions with addresses near or equal to those they have used recently.

- Temporal locality:
  - Recently referenced items are likely to be referenced again in the near future



- Spatial locality:
  - Items with nearby addresses tend to be referenced close together in time



Locality Example

```
num_list = [1, 2, 3, 4, 5, 7]
sum = 0;
for (x in num_list)
    sum += x;
return sum;
```

Spatial or Temporal Locality?

- Data references
  - Reference array elements in succession (stride-1 reference pattern). — **spatial**
  - Reference variable **sum** each iteration. — **temporal**
- Instruction references
  - Reference instructions in sequence. — **spatial**
  - Cycle through loop repeatedly. — **temporal**

Qualitative Estimates of Locality

Assuming row-major array

```
int sum_array_rows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

Answer: yes

a [0] [0]	• • •	a [0] [N-1]	a [1] [0]	• • •	a [1] [N-1]	• • •	a [M-1] [0]	• • •	a [M-1] [N-1]

Question: Does this function have good locality with respect to array `a`?

Locality Example

```
int sum_array_cols(int a[M][N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```

Answer: no, unless…

M is very small

- Question: Does this function have good locality with respect to array `a`?

a [0] [0]	. . .	a [0] [N-1]	a [1] [0]	. . .	a [1] [N-1]	. . .	a [M-1] [0]	. . .	a [M-1] [N-1]

Example Exam Question

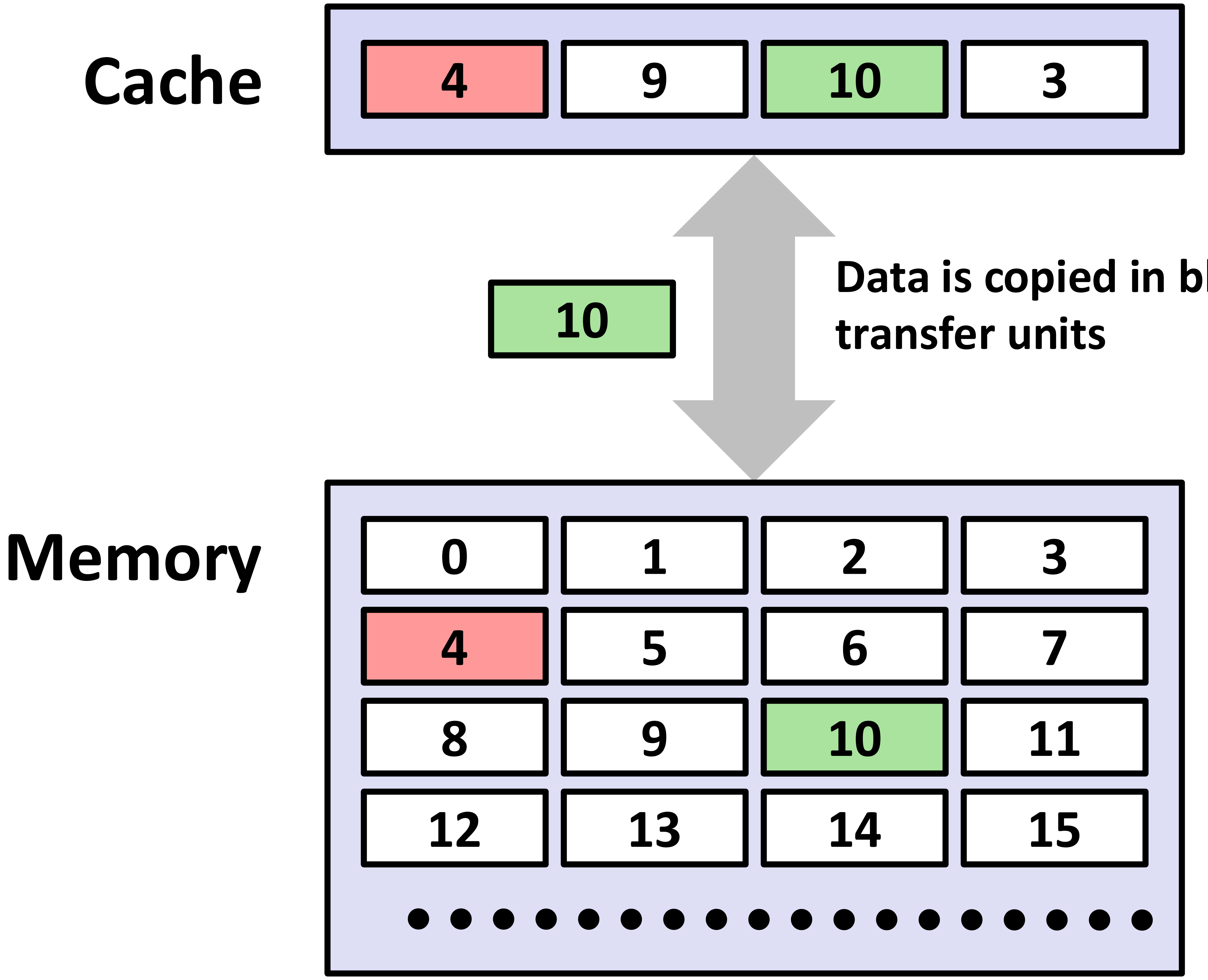
```
int sum_array_3d(int a[M][N][N])
{
    int i, j, k, sum = 0;

    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
            for (k = 0; k < M; k++)
                sum += a[k][i][j];
    return sum;
}
```

- Question: Can you permute the loops so that the function scans the 3-d array `a` with a stride-1 reference pattern (and thus has good spatial locality)?

Putting locality into practice: Caches

- Cache: A smaller, faster storage device that acts as a staging area for a subset of the data in a larger, slower device.
- Fundamental idea of a memory hierarchy:
  - For each k, the faster, smaller device at level k serves as a cache for the larger, slower device at level k+1.
- Why do memory hierarchies work?
  - Because of locality: programs tend to access the data at level k more often than they access the data at level k+1.
  - Thus, the storage at level k+1 can be slower, and thus larger and cheaper per bit.
  - Together: The memory hierarchy creates a large pool of storage that costs as much as the cheap storage near the bottom, but that serves data to programs at the rate of the fast storage near the top.

Cache in action

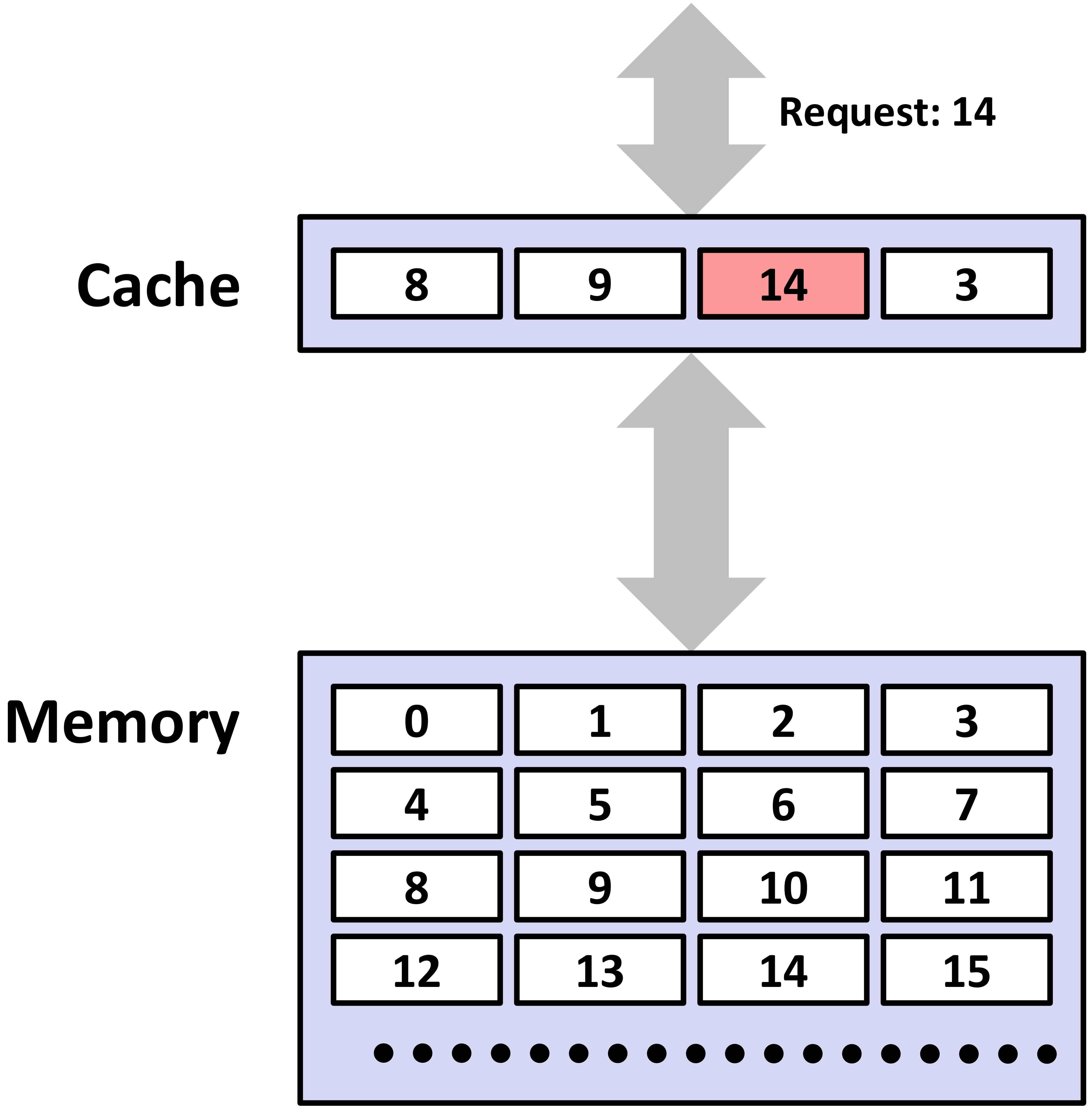


Smaller, faster, more expensive memory caches a subset of the blocks

Data is copied in block-sized transfer units

Larger, slower, cheaper memory viewed as partitioned into “blocks”

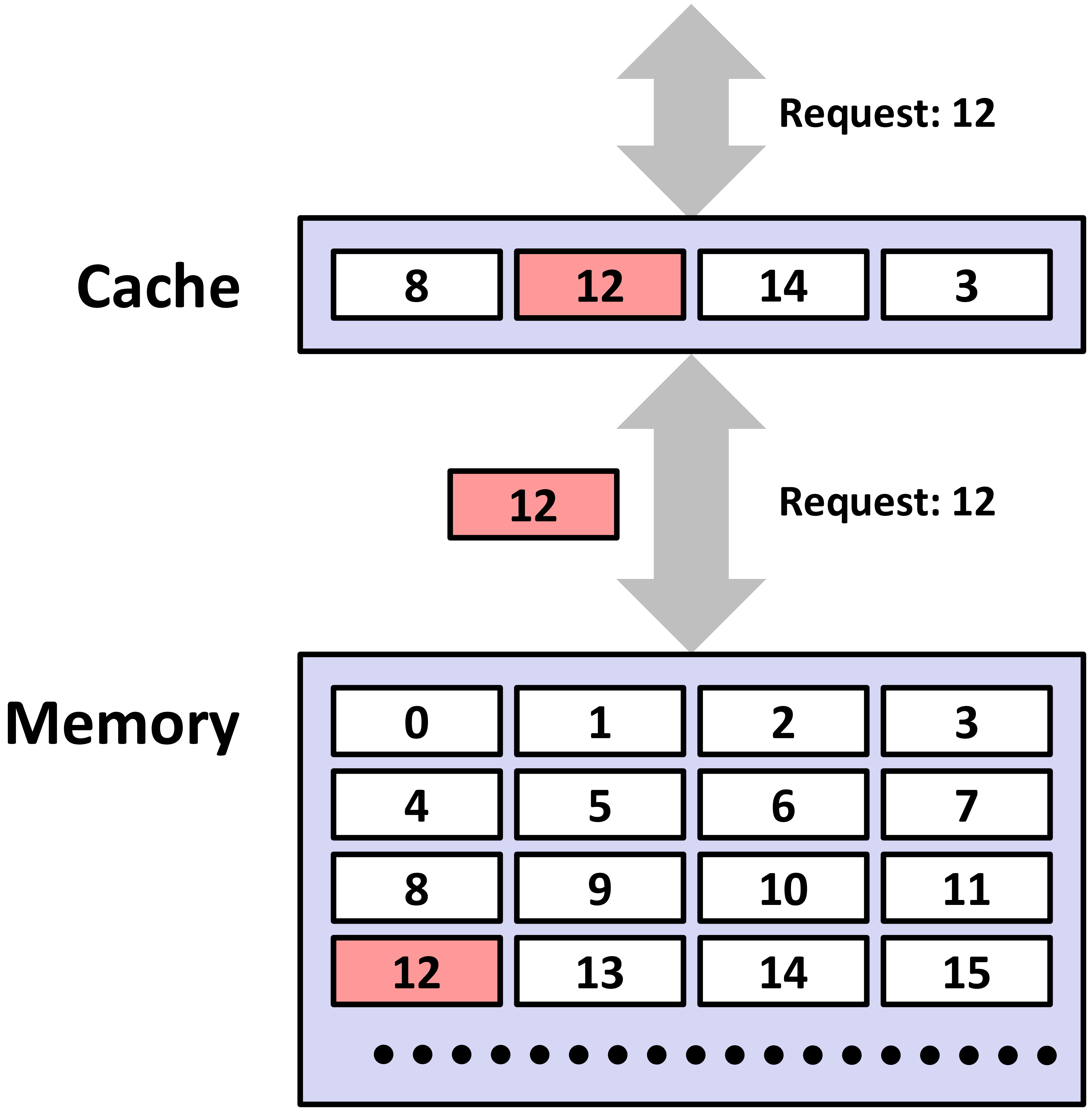
General Cache Concepts: Hit



Data in block 14 is needed

Block 14 is in cache:
Hit!

General Cache Concepts: Miss



Data in block 12 is needed

Block 12 is not in cache:
Miss!

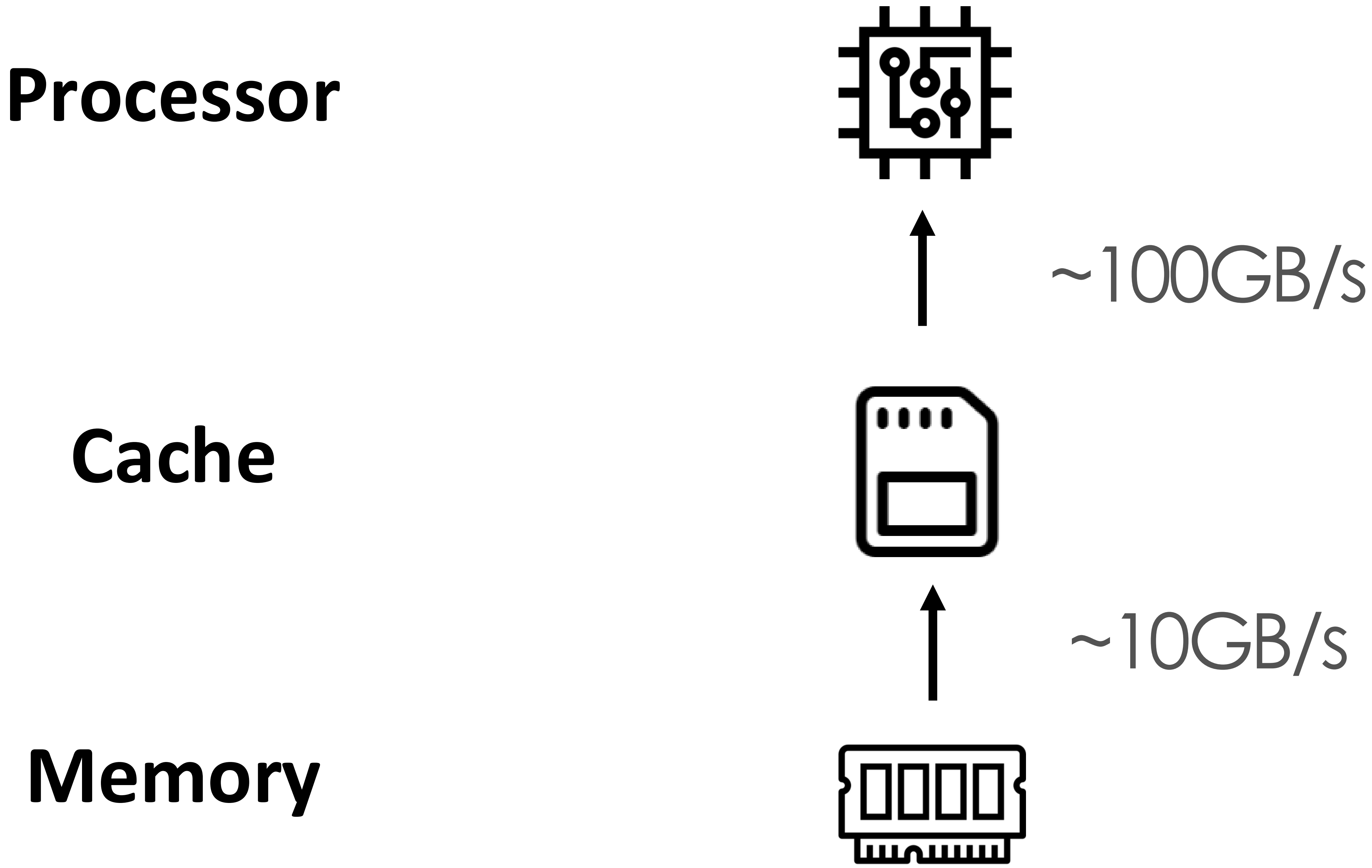
Block 12 is fetched from memory

Block 12 is stored in cache

- Placement policy: determines where b goes
- Replacement policy: determines which block gets evicted (victim)

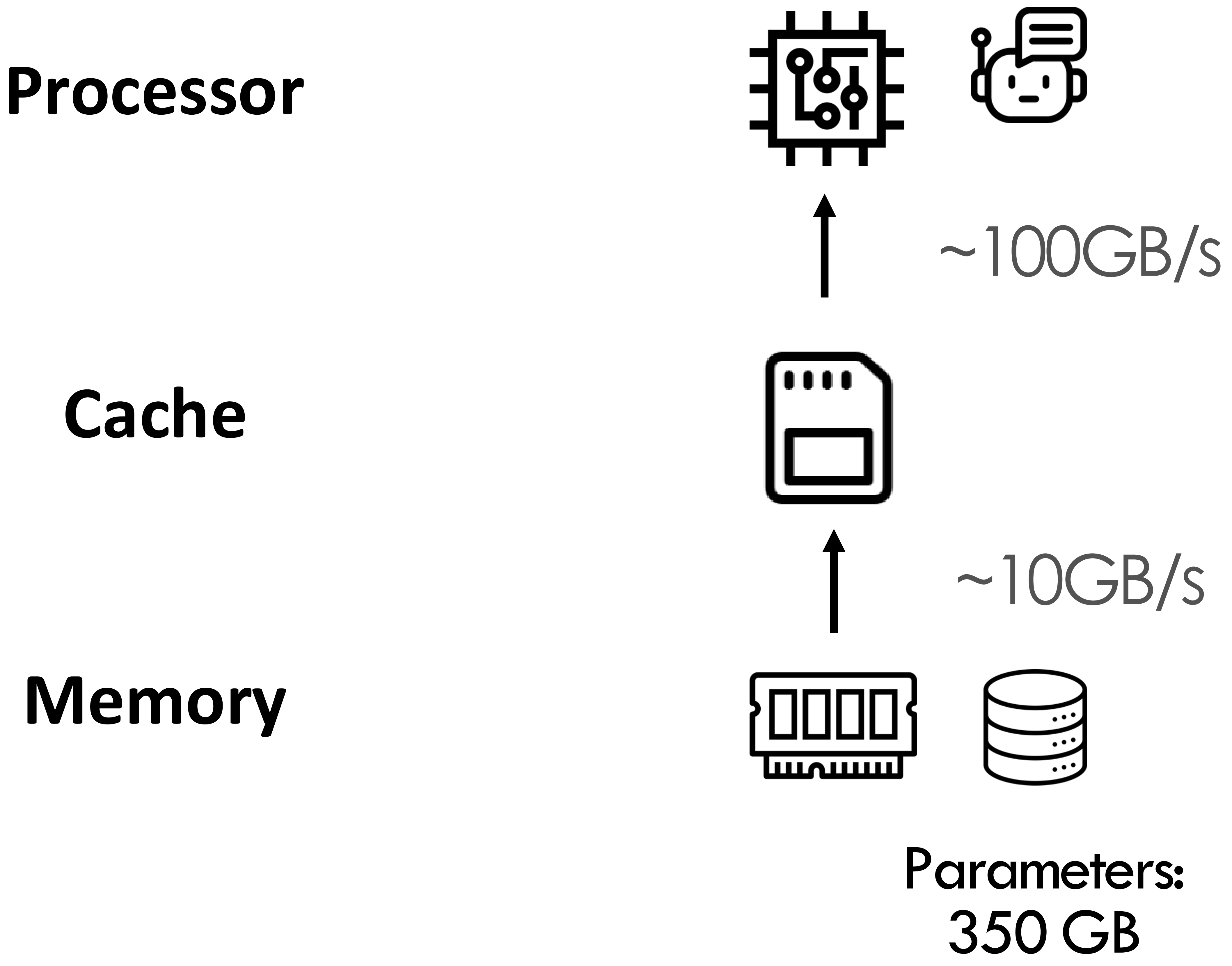
Cache in action

- If always cache hit, bandwidth?
- If always cache miss, bandwidth?



Open Question in Cache: ChatGPT

- ChatGPT: every time ChatGPT outputs token, it needs to see 350 GB parameters
- How to optimize this?



Foundation of Data Systems

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

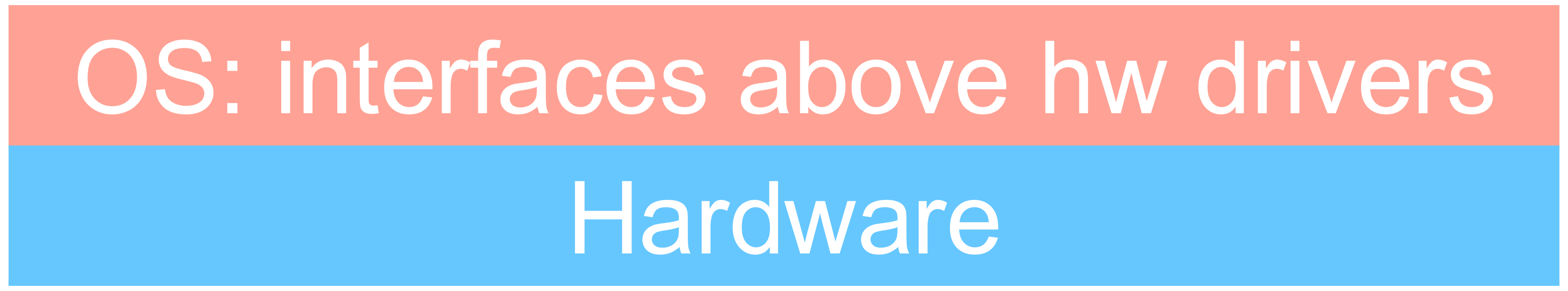
What is Operation System?

- Layers between applications and hardware

- OS makes computer hardware useful to programmers
  - Otherwise, users need to speak machine code to computer
- **[Usually**] Provides abstractions for applications
  - Manages and hides details of hardware
  - Accesses hardware through low/level interfaces unavailable to applications
- **[Often]** Provides protection
  - Prevents one app/user from clobbering another

A Primitive OS v1

- OS v1: just a library of standard services [no protection]



- Simplifying assumptions:
  - System runs one program at a time
  - No bad users or programs (?)
- Problem: poor utilization
  - - . . . of hardware (e.g., CPU idle while waiting for disk)
  - - . . . of human user (must wait for each program to finish)

OS v2: Multi-tasking

- Say: we extend the OS a bit to support many APPs
  - When one process blocks (waiting for disk, network, user input, etc.) run another process

- Problem: What can ill-behaved process do?
  - Go into infinite loop and never relinquish CPU
  - Scribble over other processes' memory to make them fail
- OS provides mechanisms **protection** to address these problems:
  - Preemption – take CPU away from looping process
  - Memory protection – protect one process' memory from one another

What is A Real OS?

- OS: manage and assign hardware resources to apps
- Goal: with N users/apps, system not N times slower
  - **Idea:** Giving resources to users who actually need them
- What can go wrong?
  - One app can interfere with other app (need **isolation**)
  - Users are gluttons, use too much CPU, etc. (need **scheduling**)
  - Total memory usage of all apps/users greater than machine's RAM (need **memory management**)
  - Disks are shared across apps / users and must be arranged propertly (need **file systems**)

Modules

- **System call:** The layer for isolation -- it abstracts the hardware and APIs for programs to use

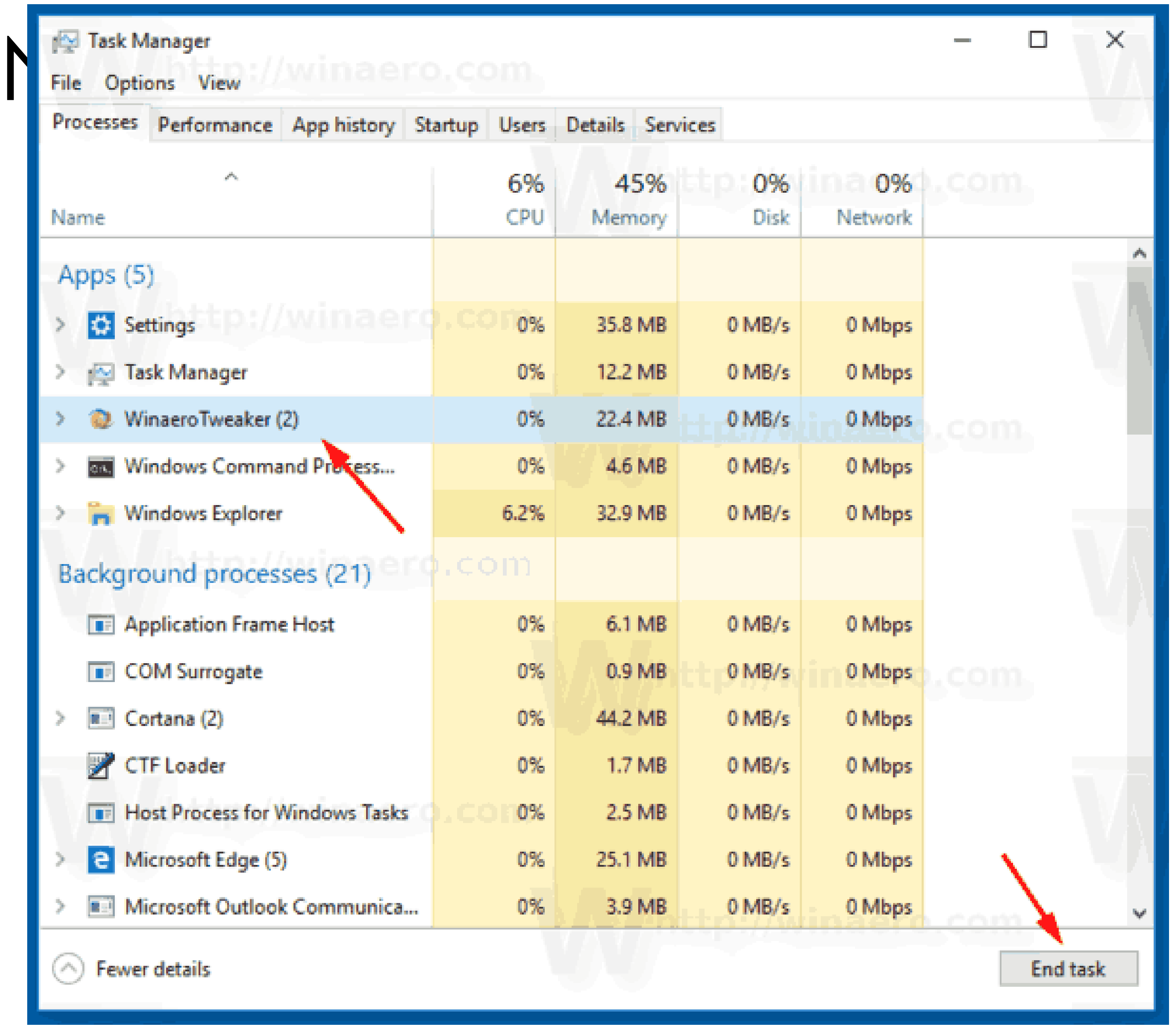
	"System Call" APIs: Isolation and protection				
Kernel Components	Process Management	Main Memory Management	Filesystems	Networking	Device Drivers
Functionality	*Virtualize* processor; "**Process**" abstraction;	*Virtualize* Main Memory	*Virtualize* disks; "**File**" abstraction;	Commun. over network	Talk to other I/O devices
	Hardware device-specific programs				
Hardware					

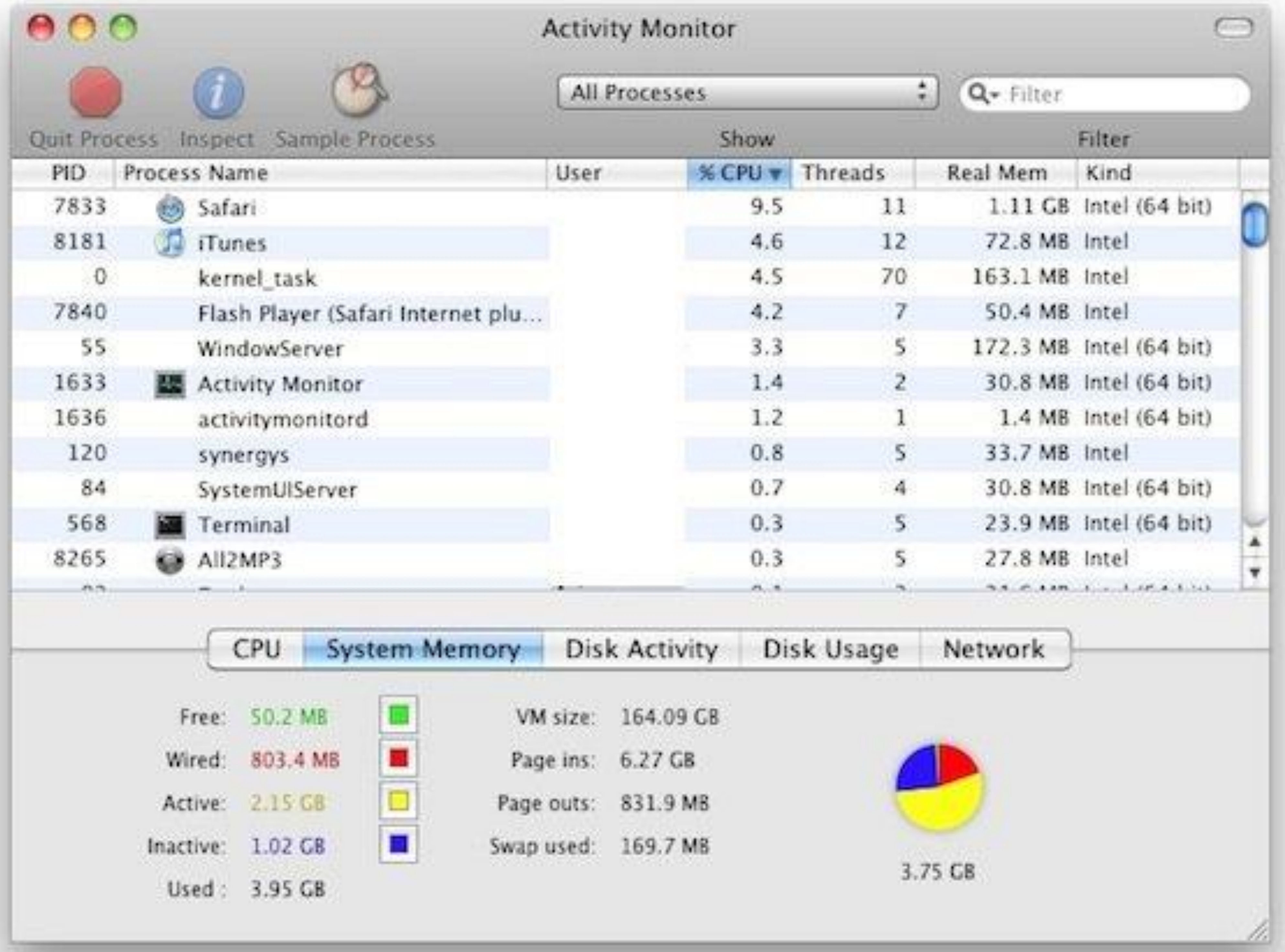
Foundation of Data Systems

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

Processes - the central abstraction in OS

- Definition: A *process* is an instance of a running program.
  - One of the most profound ideas in computer science
  - N ’ or





Main function in python

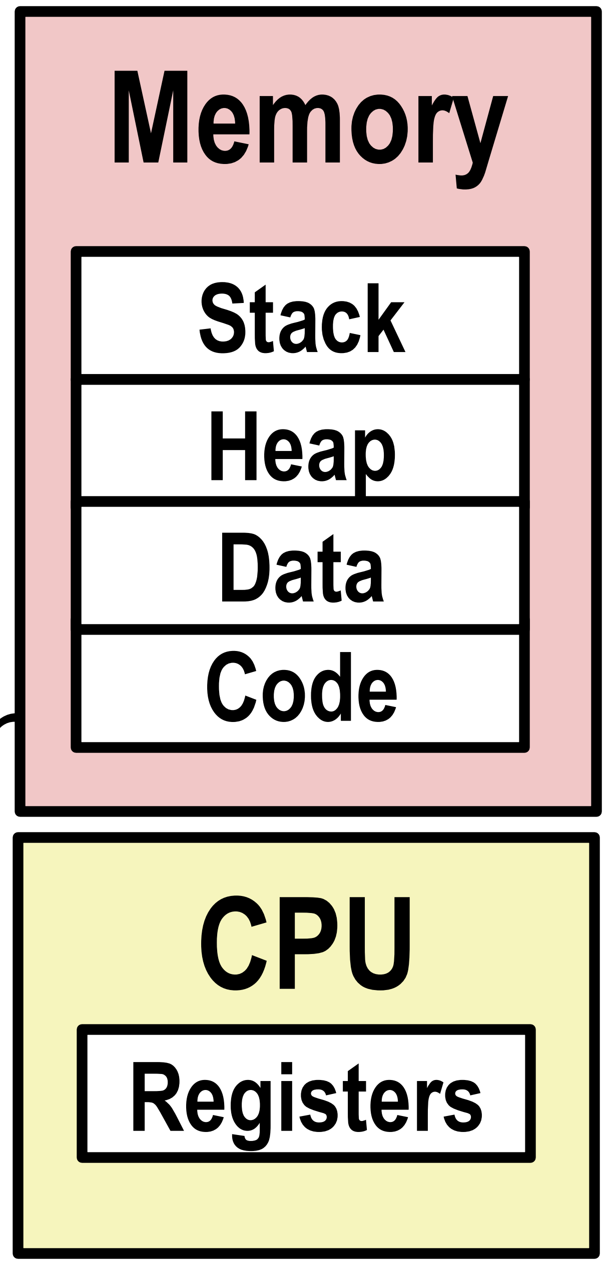
test.py

```python
print("Good Morning")


def main():
    print("Hello Python")


print("Good Evening")

if __name__ == "__main__":
    main()
```

```
Good Morning
Good Evening
Hello Python

Process finished with exit code 0
```

Processes - the central abstraction in OS

- Process provides each program with two key abstractions (for resources):
  - **Compute Resource**
    - Each program seems to have exclusive use of the CPU
    - Provided by kernel mechanism called *context switching*
  - **Memory Resource**
    - Each program seems to have exclusive use of main mem
    - Provided by kernel mechanism called virtual memory



The Abstraction of a Process

- High-level steps OS takes to get a process going:

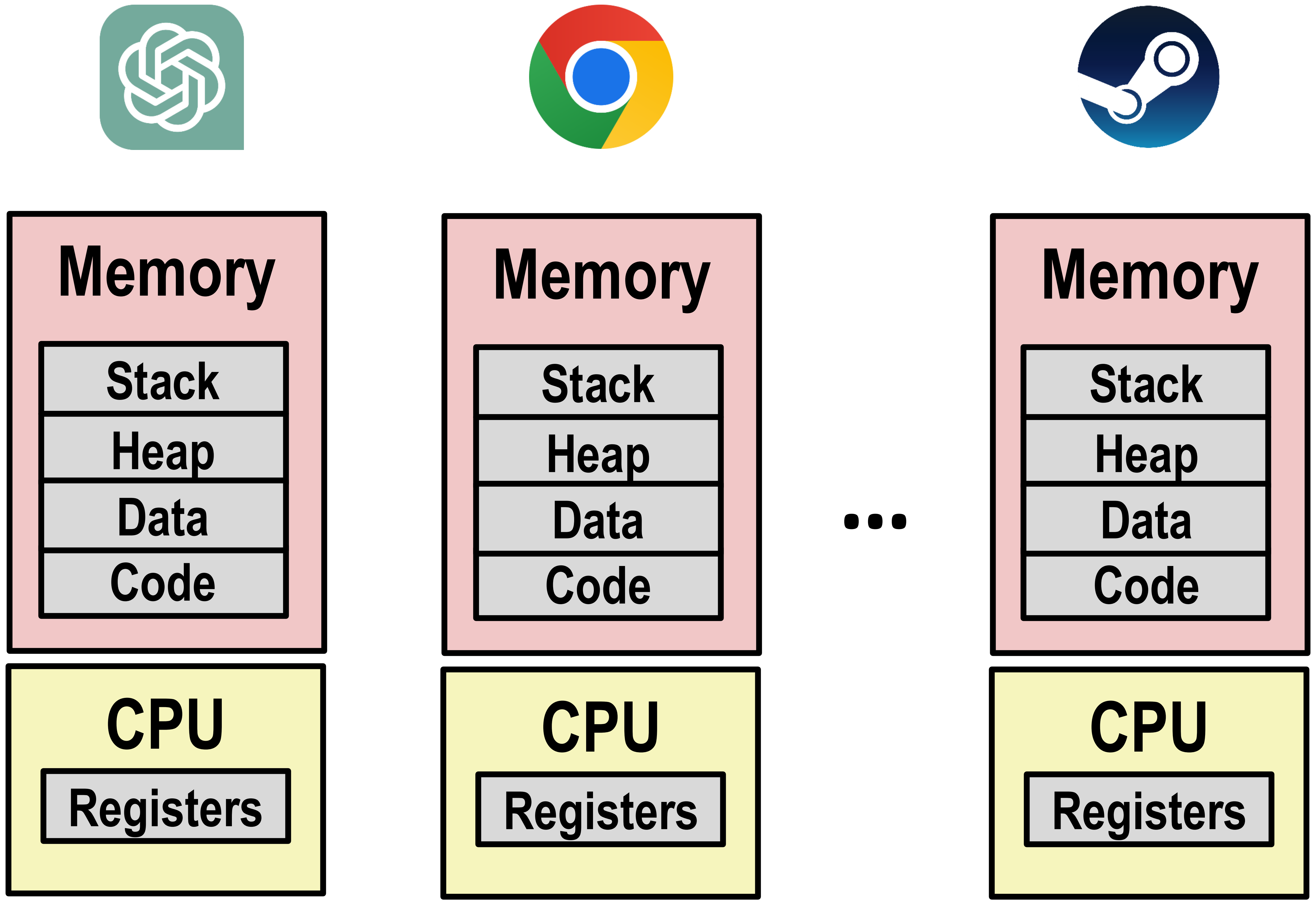
1. Create a process (get Process ID; add to Process List)
2. Assign part of DRAM to process, aka its Address Space
3. Load code and static data (if applicable) to that space
4. Set up the inputs needed to run program's *main()*
5. Update process' State to *Ready*
6. When process is scheduled (*Running*), OS temporarily hands off control to process to run the show!
7. Eventually, process finishes or run Destroy

Virtualization of Hardware Resources

***Q:** But is it not risky/foolish for OS to hand off control of hardware to a process (random user-written program)?!*

- OS has *mechanisms* and *policies* to regain control
- Virtualization:
  - Each hardware resource is treated as a virtual entity that OS can divvy up among processes in a controlled way
- Limited Direct Execution:
  - OS mechanism to time-share CPU and preempt a process to run a different one, aka "context switch"
  - A Scheduling policy tells OS what time-sharing to use
  - Processes also must transfer control to OS for "privileged" operations (e.g., I/O); System Calls API

Multiprocessing: The Illusion



- Computer runs many processes simultaneously

Multiprocessing Example

top command in terminal: many processes, Identified by Process ID (**PID**)

xterm

```
Processes: 123 total, 5 running, 9 stuck, 109 sleeping, 611 threads          11:47:07
Load Avg: 1.03, 1.13, 1.14  CPU usage: 3.27% user, 5.15% sys, 91.56% idle
SharedLibs: 576K resident, 0B data, 0B linkedit.
MemRegions: 27958 total, 1127M resident, 35M private, 494M shared.
PhysMem: 1039M wired, 1974M active, 1062M inactive, 4076M used, 18M free.
VM: 280G vsize, 1091M framework vsize, 23075213(1) pageins, 5843367(0) pageouts.
Networks: packets: 41046228/11G in, 66083096/77G out.
Disks: 17874391/349G read, 12847373/594G written.

PID    COMMAND      %CPU TIME     #TH   #WQ  #PORT #MREG RPRVT  RSHRD  RSIZE  VPRVT  VSIZE
99217- Microsoft Of 0.0  02:28.34 4     1    202   418   21M    24M    21M    66M    763M
99051  usbmuxd      0.0  00:04.10 3     1    47    66    436K   216K   480K   60M    2422M
99006  iTunesHelper 0.0  00:01.23 2     1    55    78    728K   3124K  1124K  43M    2429M
84286  bash         0.0  00:00.11 1     0    20    24    224K   732K   484K   17M    2378M
84285  xterm        0.0  00:00.83 1     0    32    73    656K   872K   692K   9728K  2382M
55939- Microsoft Ex 0.3  21:58.97 10    3    360   954   16M    65M    46M    114M   1057M
54751  sleep        0.0  00:00.00 1     0    17    20    92K    212K   360K   9632K  2370M
54739  launchdadd   0.0  00:00.00 2     1    33    50    488K   220K   1736K  48M    2409M
54737  top          6.5  00:02.53 1/1   0    30    29    1416K  216K   2124K  17M    2378M
54719  automountd   0.0  00:00.02 7     1    53    64    860K   216K   2184K  53M    2413M
54701  ocspd        0.0  00:00.05 4     1    61    54    1268K  2644K  3132K  50M    2426M
54661  Grab         0.6  00:02.75 6     3    222+  389+  15M+   26M+   40M+   75M+   2556M+
54659  cookied      0.0  00:00.15 2     1    40    61    3316K  224K   4088K  42M    2411M
53818  mdworker     0.0  00:01.67 4     1    52    91    7628K  7412K  16M    48M    2438M
50878  mdworker     0.0  00:11.17 3     1    53    91    2464K  6148K  9976K  44M    2434M
50410  xterm        0.0  00:00.13 1     0    32    73    280K   872K   532K   9700K  2382M
50078  emacs        0.0  00:06.70 1     0    20    35    52K    216K   88K    18M    2392M
```