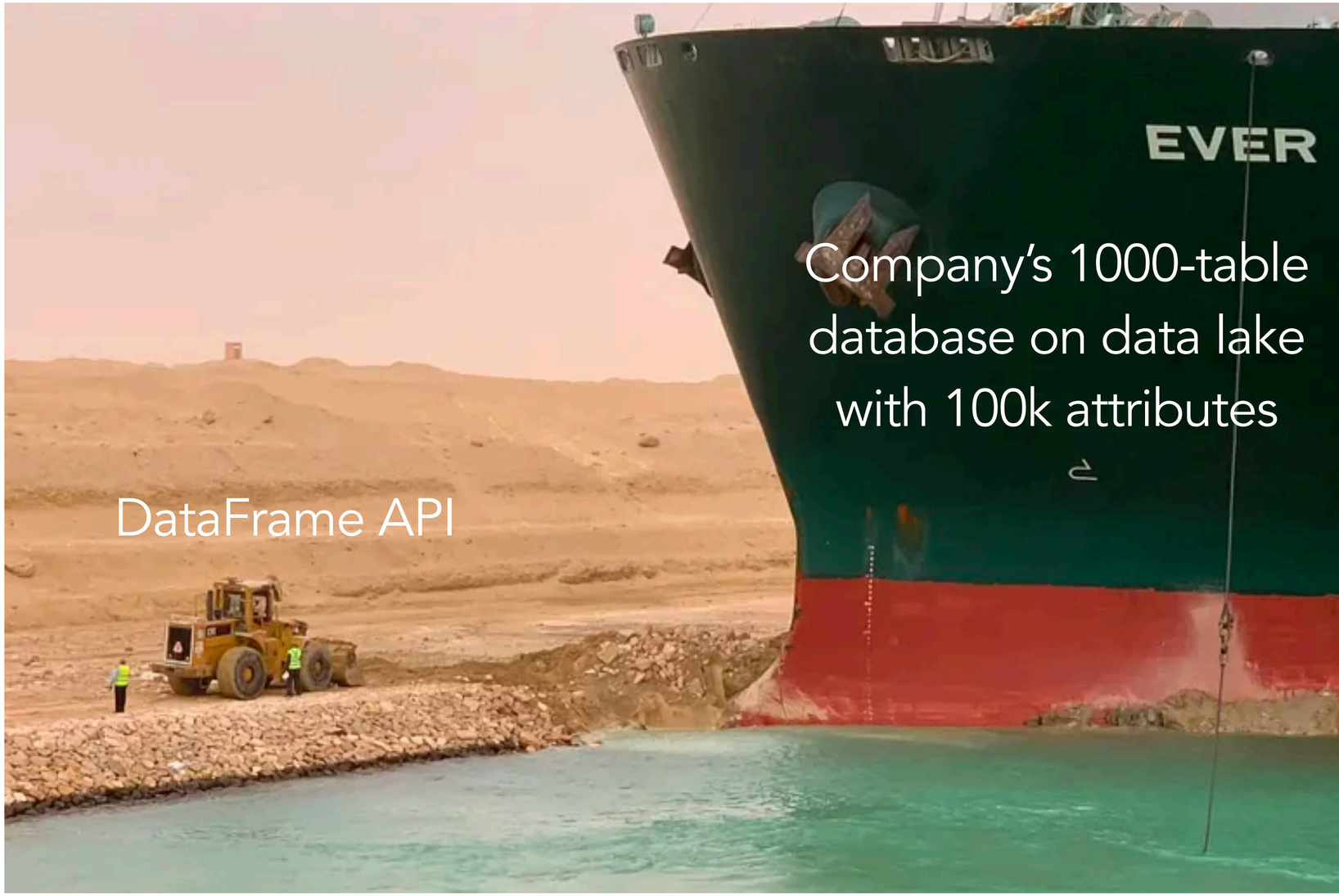
DSC 204a Scalable Data Systems

- Haojian Jin

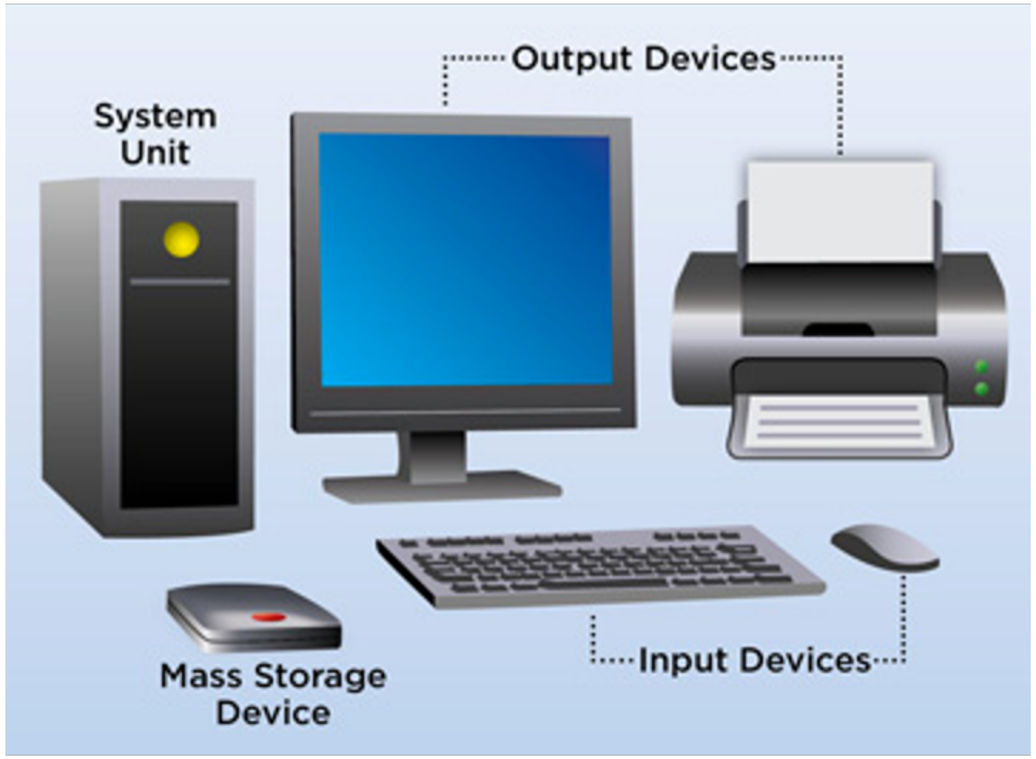


Meme idea credit: https://datasystemsfun.tumblr.com/

What is a computer?

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

Basics of Computer Organization

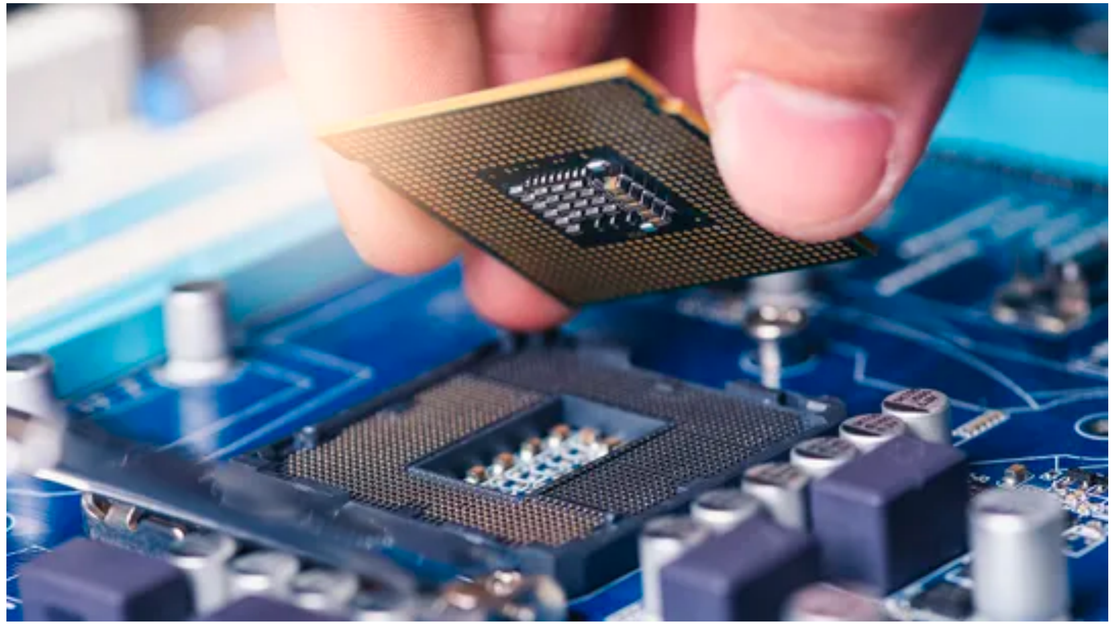


- Hardware: The electronic machinery (wires, circuits, transistors, capacitors, devices, etc.)
- Software: Programs (instructions) and data

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

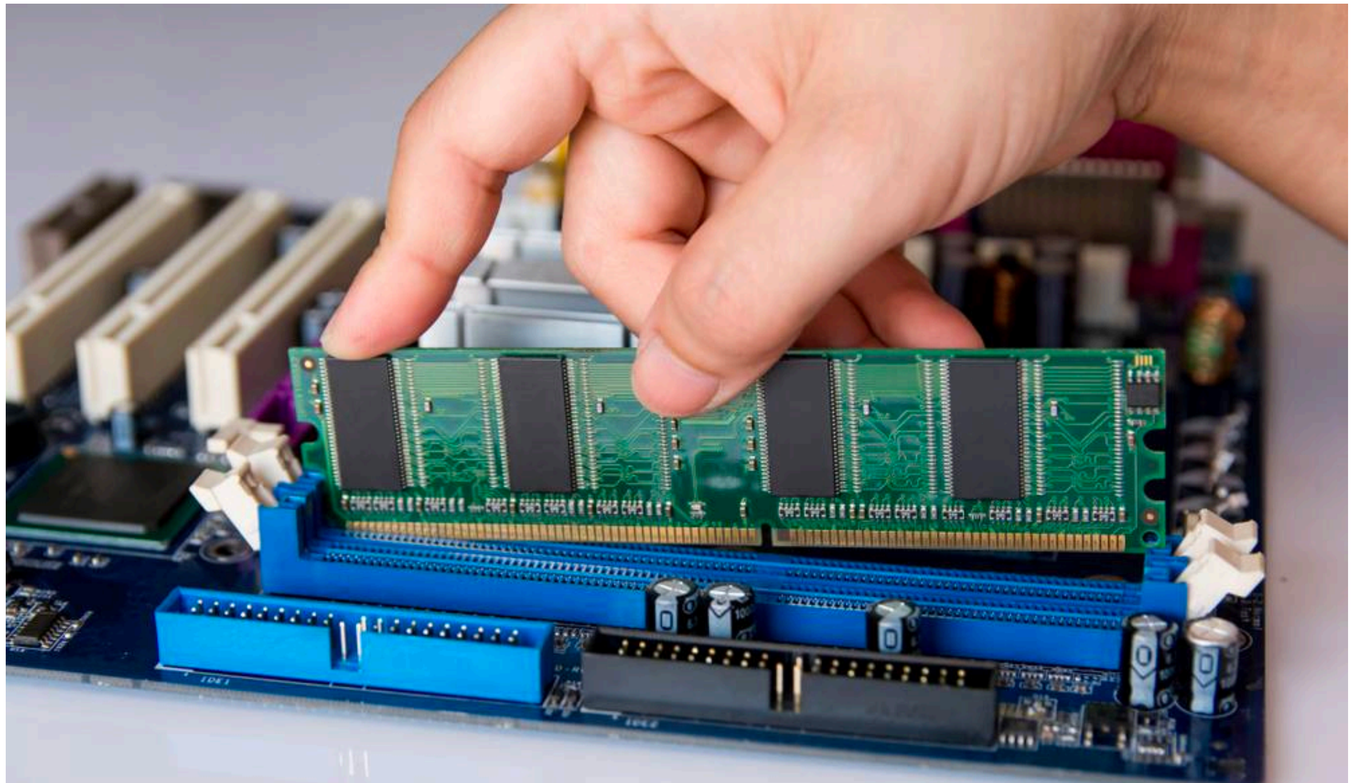
Key Parts of Computer Hardware

- 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

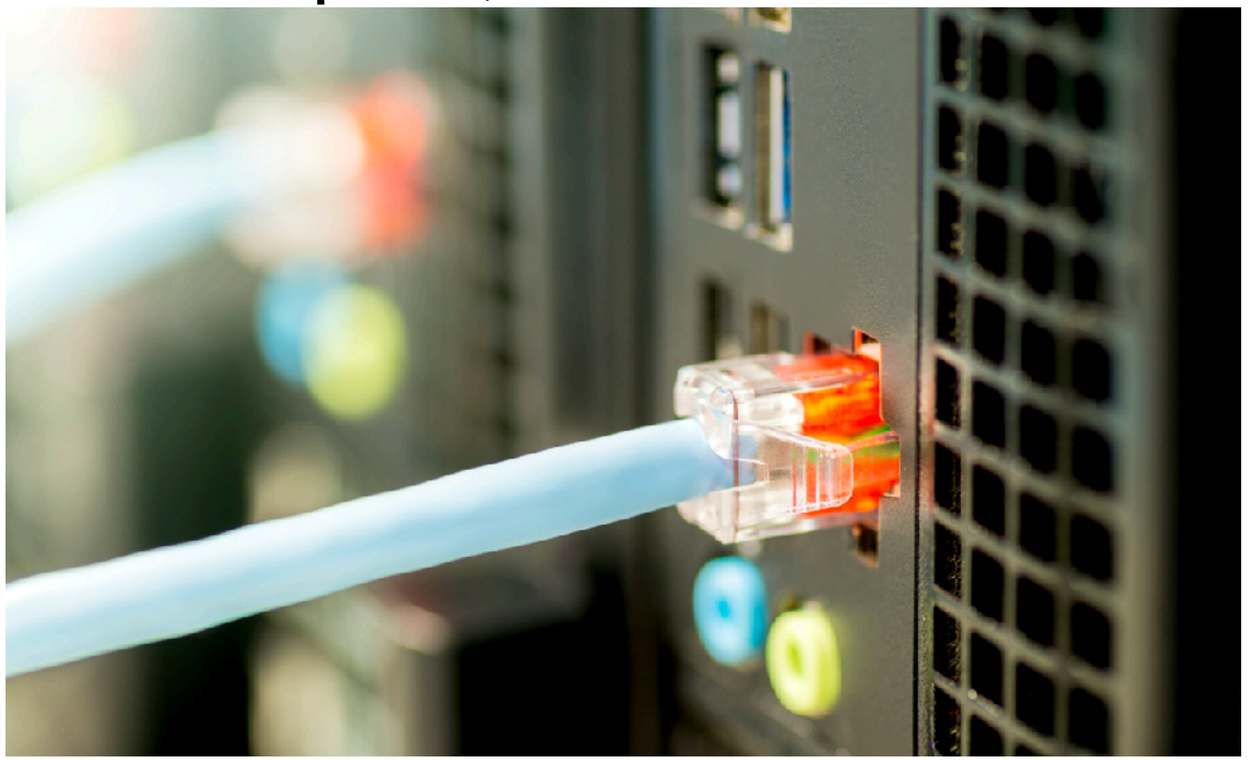


Key Parts of Computer Hardware

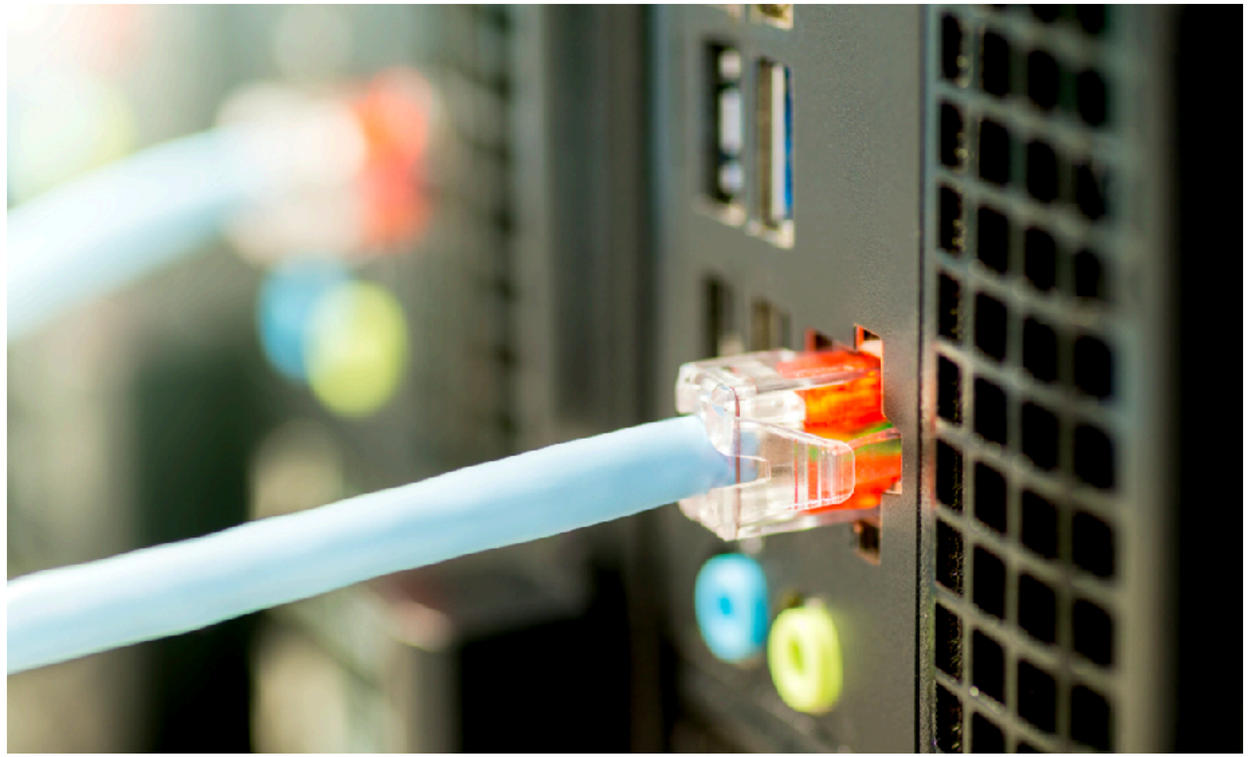
- Disk (aka secondary/persistent storage)
  - Similar to memory but persistent, slower, and higher capacity / cost ratio; various addressing schemes

Key Parts of Computer Hardware

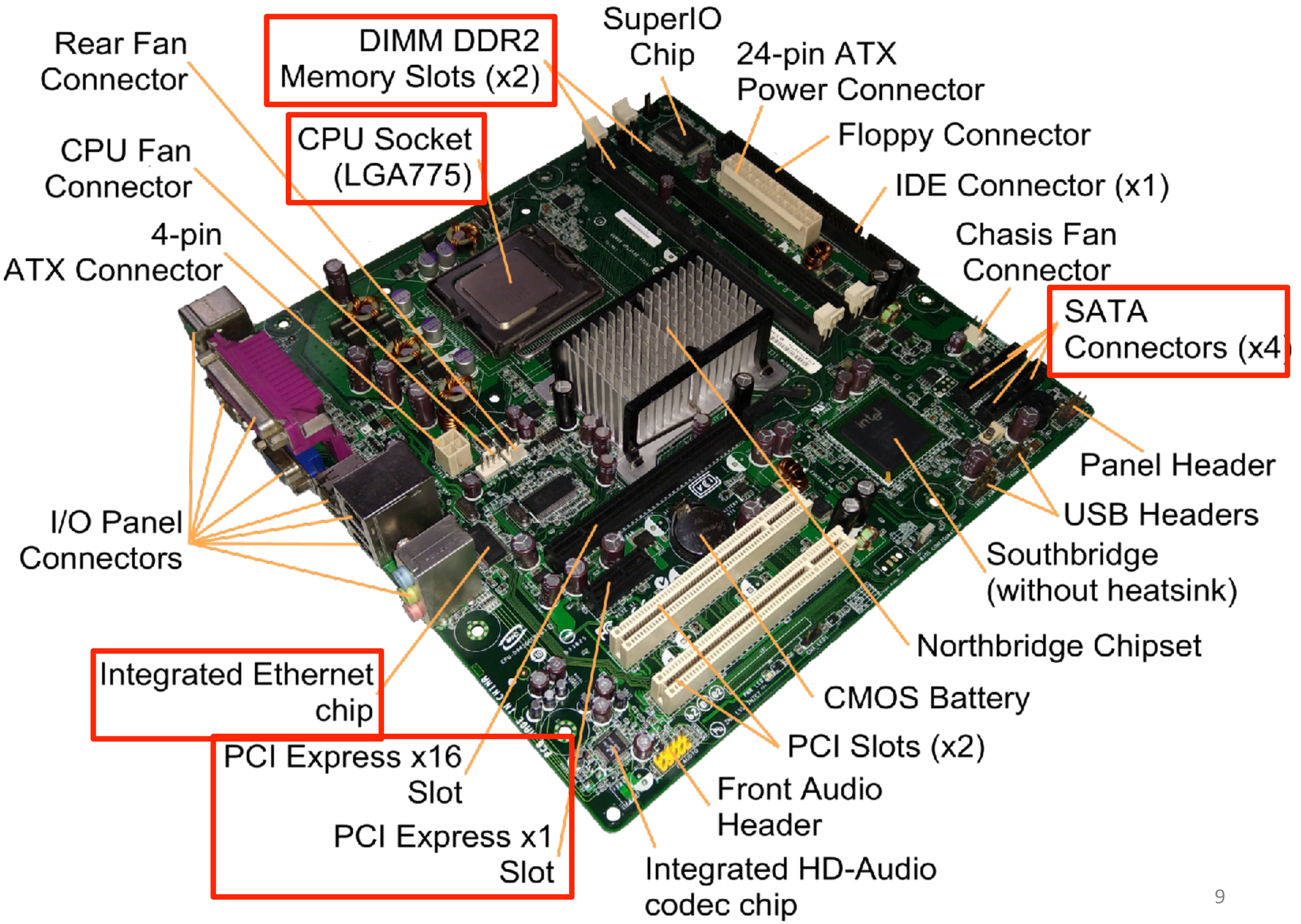
- Network interface controller (NIC)
  - Hardware to send data to / retrieve data over network of interconnected computers/devices



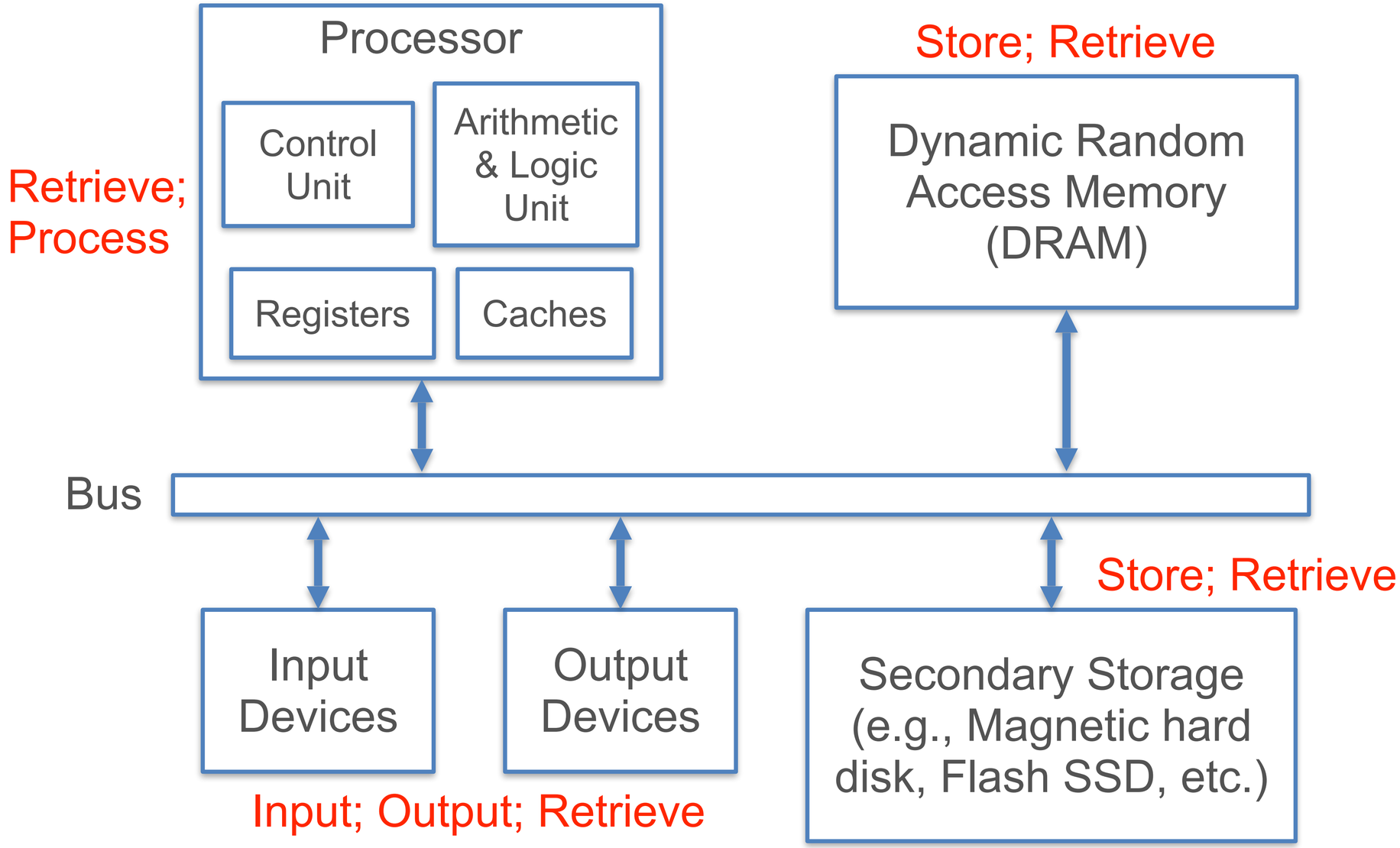
Bonus: Debugging your network



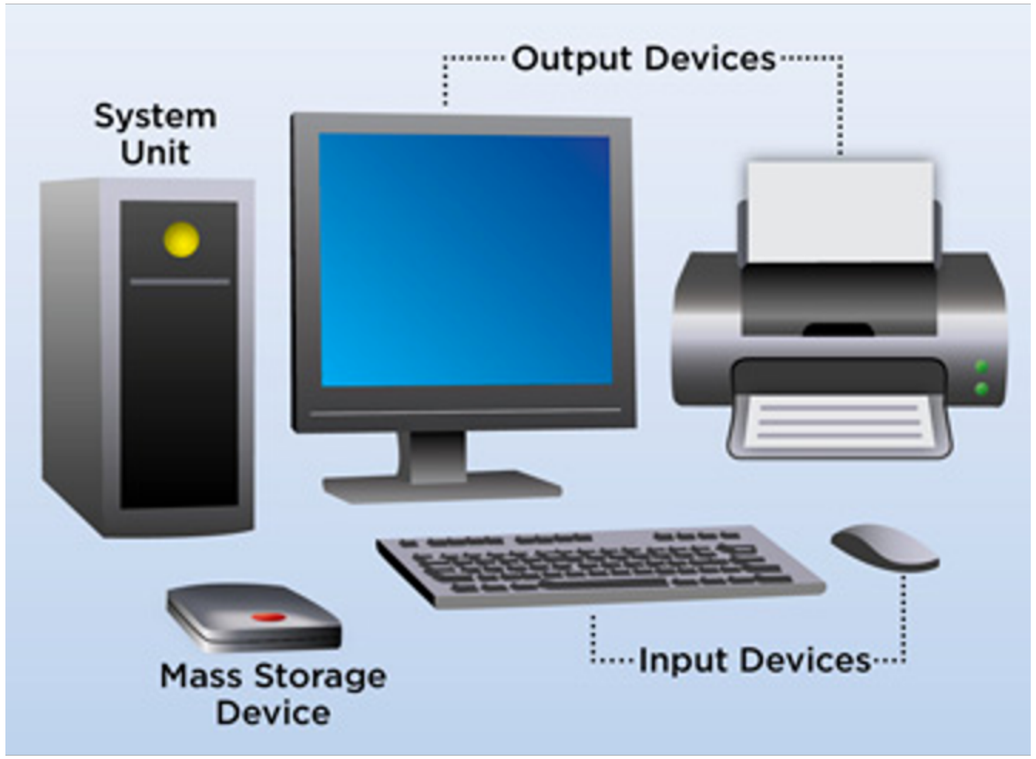
- Green
  - on: 100 Mbps (fast)
  - off: 10 Mbps (slow)
- Orange
  - on: Connected. No data
  - off: Disconnected
  - blinking: being transferred.


Rear Fan Connector
DIMM DDR2 Memory Slots (x2)
SuperIO Chip
24-pin ATX Power Connector
CPU Fan Connector
CPU Socket (LGA775)
Floppy Connector
IDE Connector (x1)
4-pin ATX Connector
Chasis Fan Connector
SATA Connectors (x4)
Panel Header
USB Headers
I/O Panel Connectors
Southbridge (without heatsink)
Northbridge Chipset
Integrated Ethernet chip
CMOS Battery
PCI Express x16 Slot
PCI Slots (x2)
Front Audio Header
PCI Express x1 Slot
Integrated HD-Audio codec chip

Abstract Computer Parts and Data



Parts of a Computer



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

https://www.webopedia.com/TERM/C/computer.html

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.

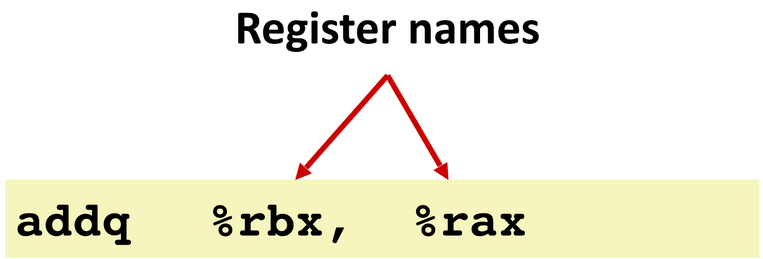
Main kinds of Software

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

Recap: Digital Representation of Data

- The *size* and *interpretation* of a data type depends on PL
- A **Byte** (B; 8 bits) is typically the basic unit of data types
  - CPU can't address anything smaller than a byte.
- **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! (**Q:** *Why?*)
- **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 $(2^{32} - 1)$; Python3 *int* is effectively unlimited length (PL magic!)

Instruction



These are 64-bit registers, so we know this is a 64-bit add

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 ↓

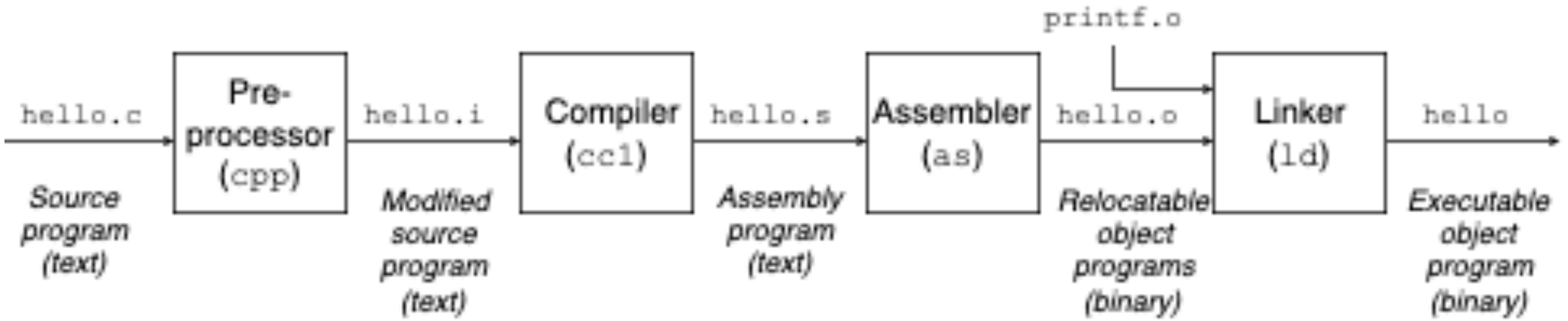
Machine code tied to ISA

↓

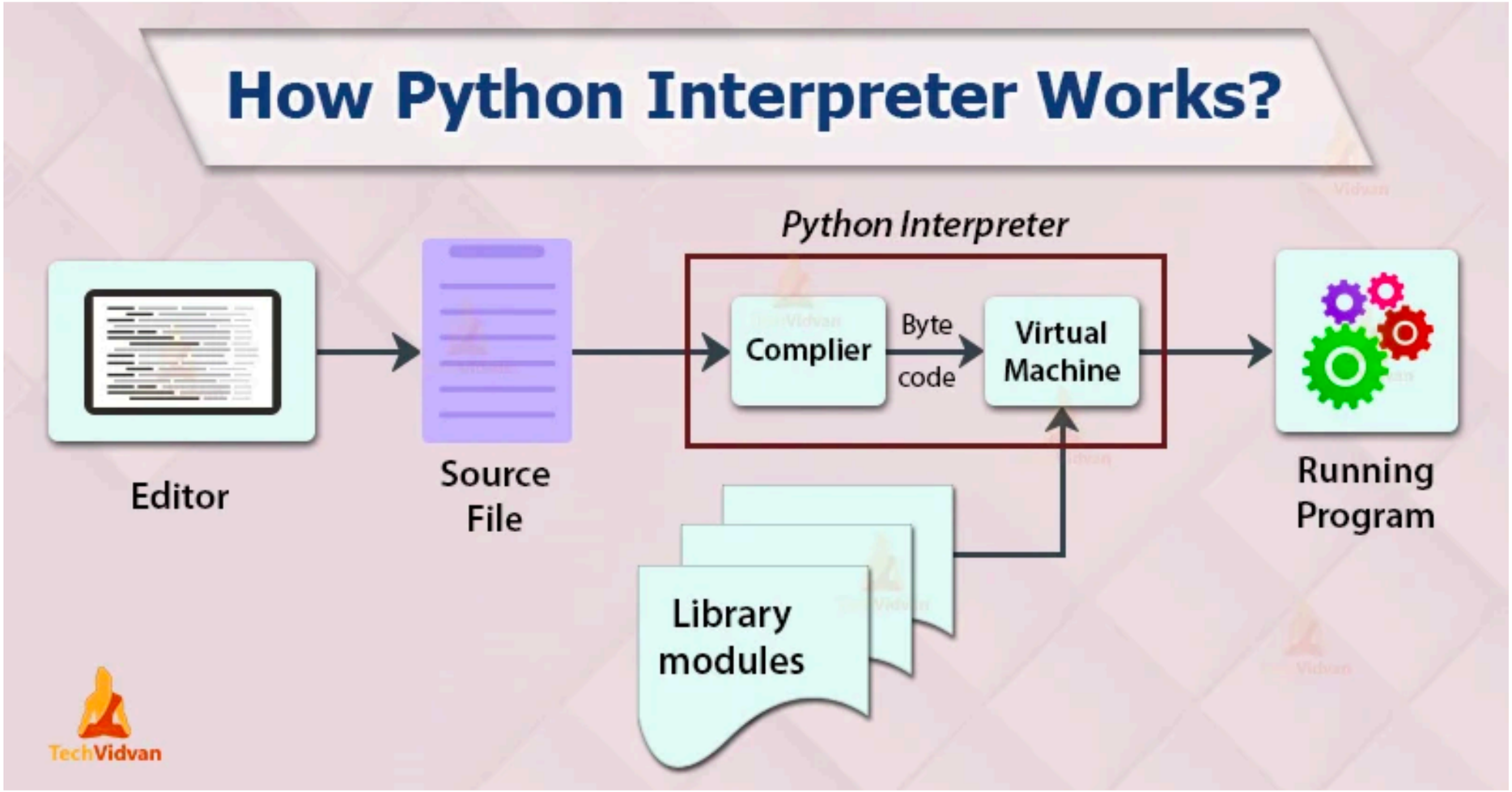
Run on processor

```
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
```

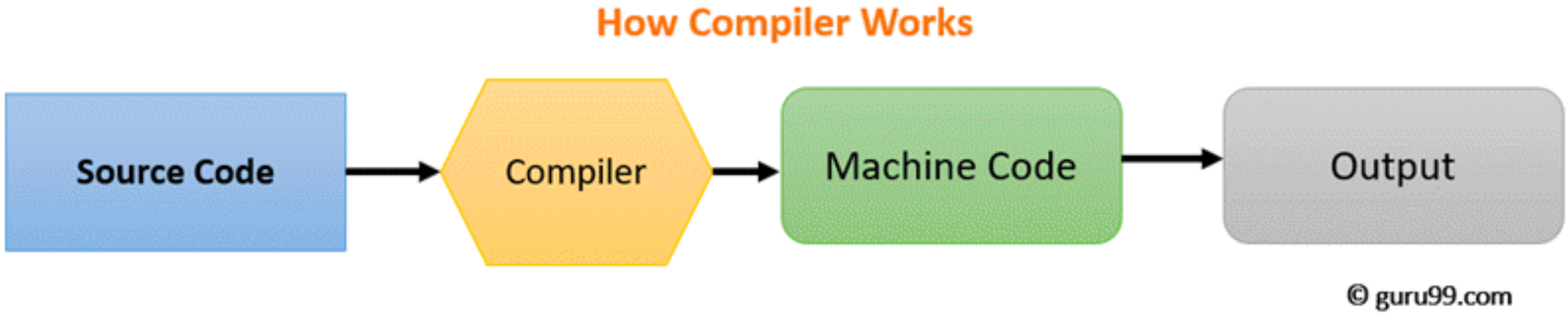
Compilation



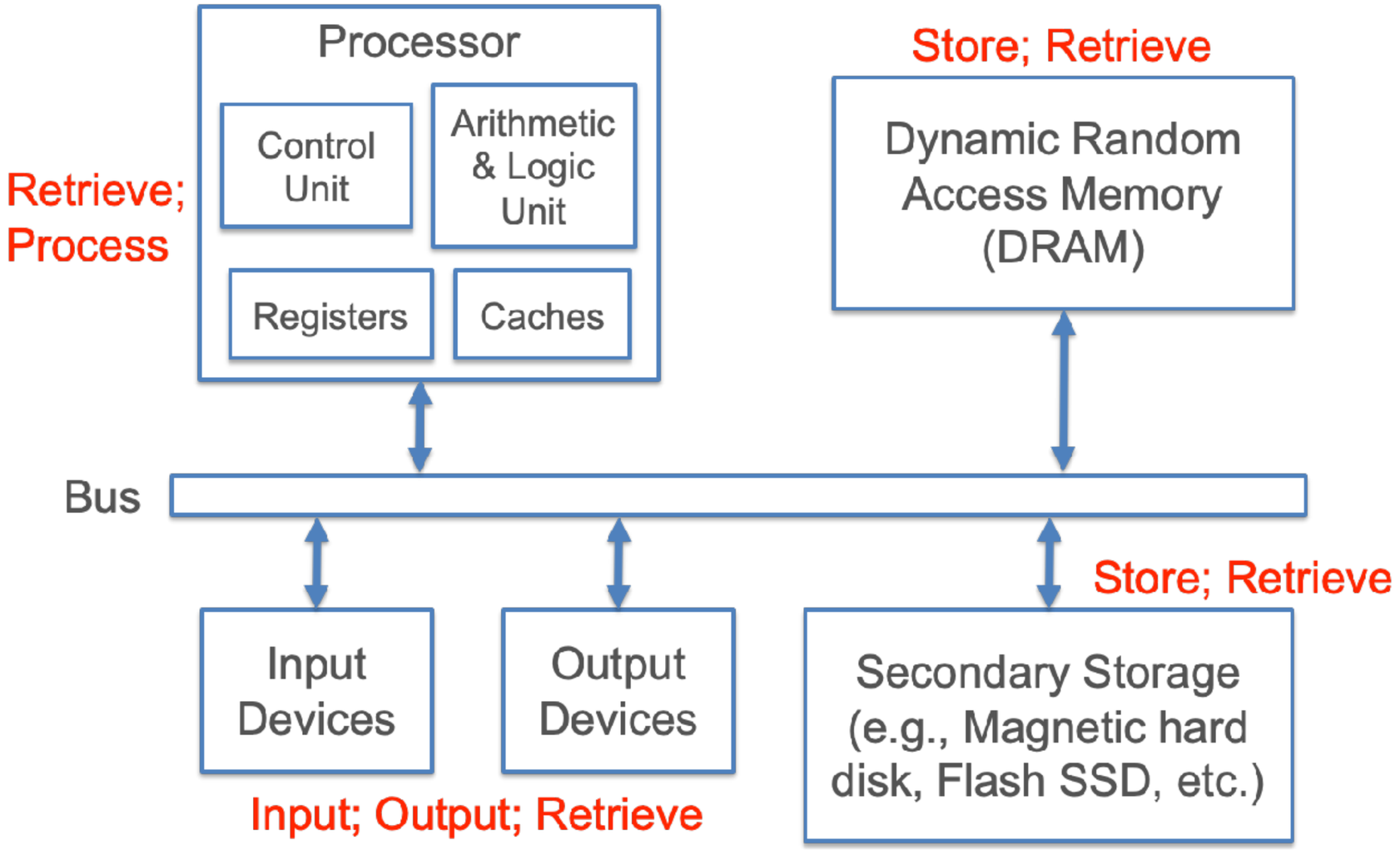
Interpretation



Compiler v.s. Interpreter



Abstract Computer Parts and Data



Writing & Reading Memory

Write

Transfer data from memory to CPU

movq %rax, 8(%rsp)

"Store" operation

Read

Transfer data from CPU to memory

movq 8(%rsp), %rax

"Load" operation

From 5th lecture

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

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

Example

Q: *What does this program do when run with 'python'?*
(Assume tmp.csv is in current working directory)

tmp.py

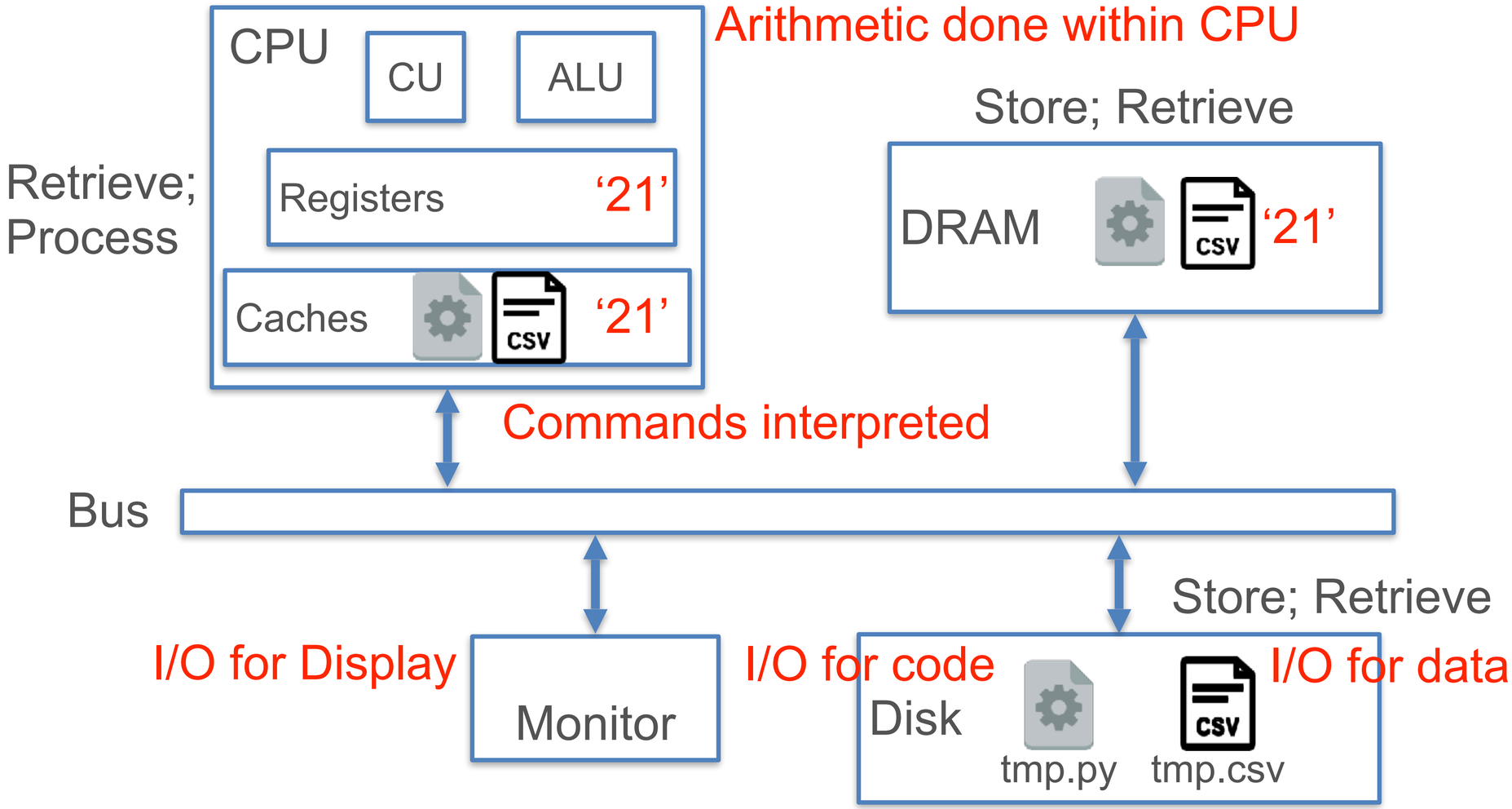
```
import pandas as p
m = p.read_csv('tmp.csv',header=None)
s = m.sum().sum()
print(s)
```

tmp.csv

```
1,2,3
4,5,6
```

Example

Rough sequence of events when program is executed



Processor Performance

Q: *How fast can a processor process a program?*

- Modern CPUs can run millions of instructions per second!
  - ISA tells #**clock cycles** per instruction
  - CPU's **clock rate** helps map that to runtime (ns)
- Alas, most programs do not keep CPU always busy:
  - Memory access commands **stall** the processor; ALU and CU are *idle* during DRAM-register transfer
  - Worse, data may not be in DRAM—wait for (disk) I/O!
  - So, actual *runtime* of program may be OOM higher than what clock rate calculation suggests

Key Principle: Optimizing access to DRAM and use of processor caches is critical for processor performance!

Review Questions

- What is an ISA?
- What are the 3 main kinds of commands in an ISA?
- Why do CPUs have both registers and caches?
- Why is it typically impossible for data processing programs to achieve 100% processor utilization?

Review Questions

- What is an ISA?
- What are the 3 main kinds of commands in an ISA?
- Why do CPUs have both registers and caches?
- Why is it typically impossible for data processing programs to achieve 100% processor utilization?
- Which of these memory hierarchy layers is the most expensive: CPU cache, DRAM, flash disks, or magnetic hard disks?
- Which of the above layers is the slowest for data access?
- Which library helps ML users avoid need for writing GPU cache-aware computations?