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

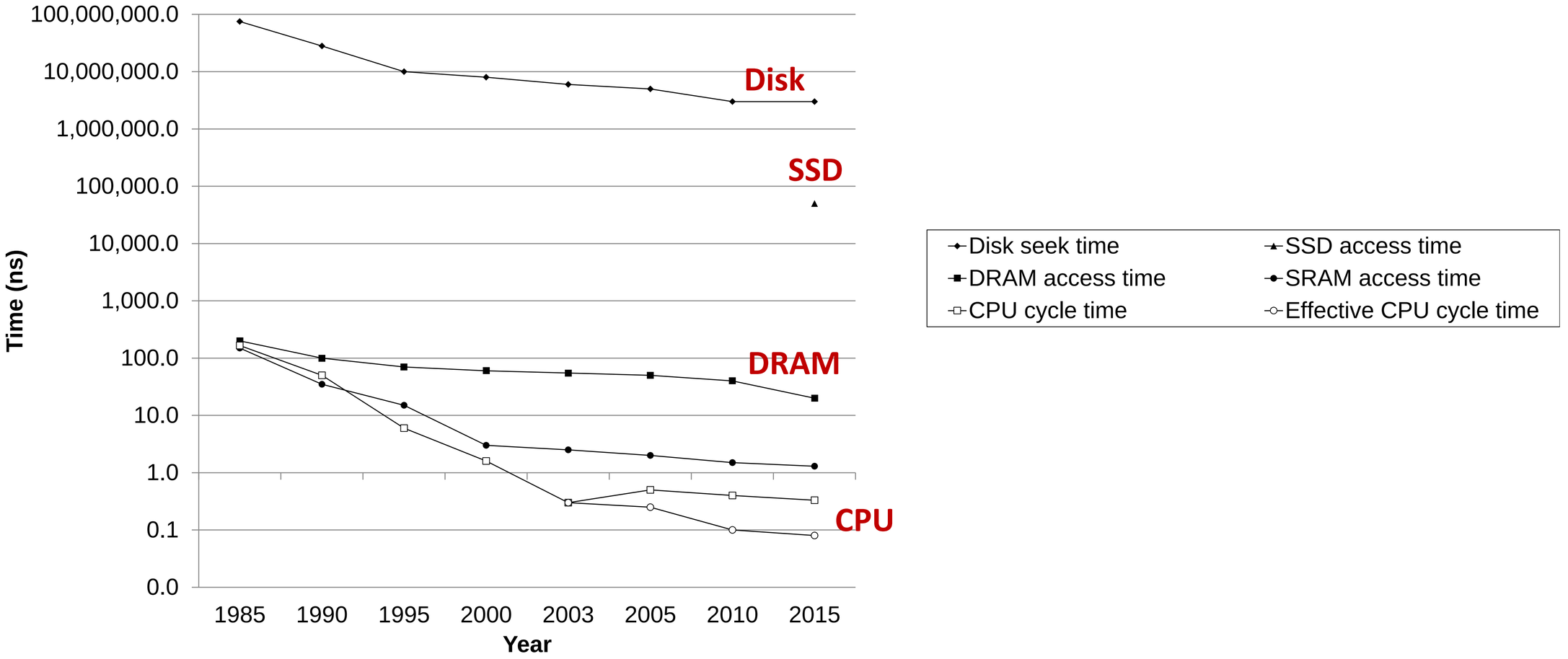
- Readings are uploaded
  - Only 1 per class, multiple optional
  - This week's readings: OS processes and memory management
- Reading summary due: Next Wednesday 1/24
  - Submit via GradeScope
  - Follow the NeurIPS template: maximum 2 pages.
- Next week reading out:
  - By this Saturday

Week 1 Recap

1. DSC204A: we see everything as data and compute
2. Computer: hardware and software
3. Data rep: bits, bytes, integer, fp16, fp32, bf16, …
4. How computer works

The CPU-Memory Gap

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



Question

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];
}
```

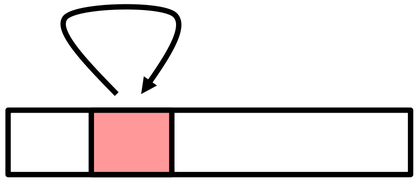
4.3 milliseconds

```
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];
}
```

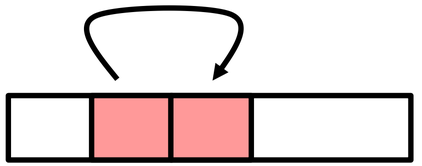
81.8 milliseconds

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;
```

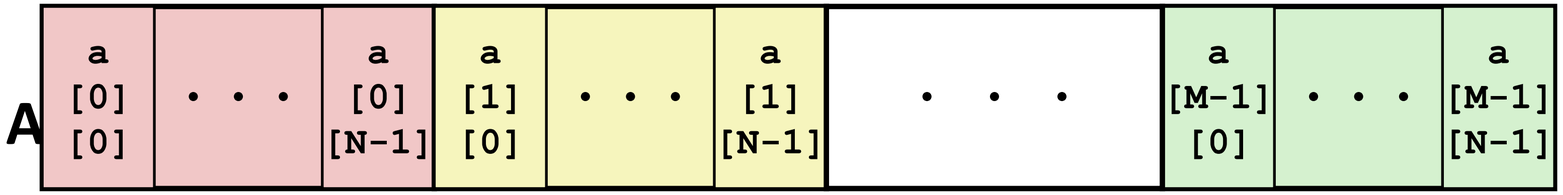
Spatial or Temporal Locality?

- 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

Hint: array layout is row-major order

```
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;
}
```



- Claim: Being able to look at code and get a qualitative sense of its locality is a key skill for a professional programmer.
- Question: Does this function have good locality with respect to array `a`?

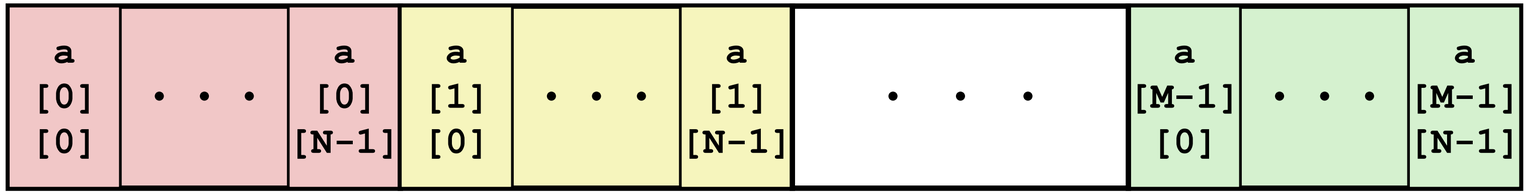
Locality Example

```
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`?



Example Exam Question

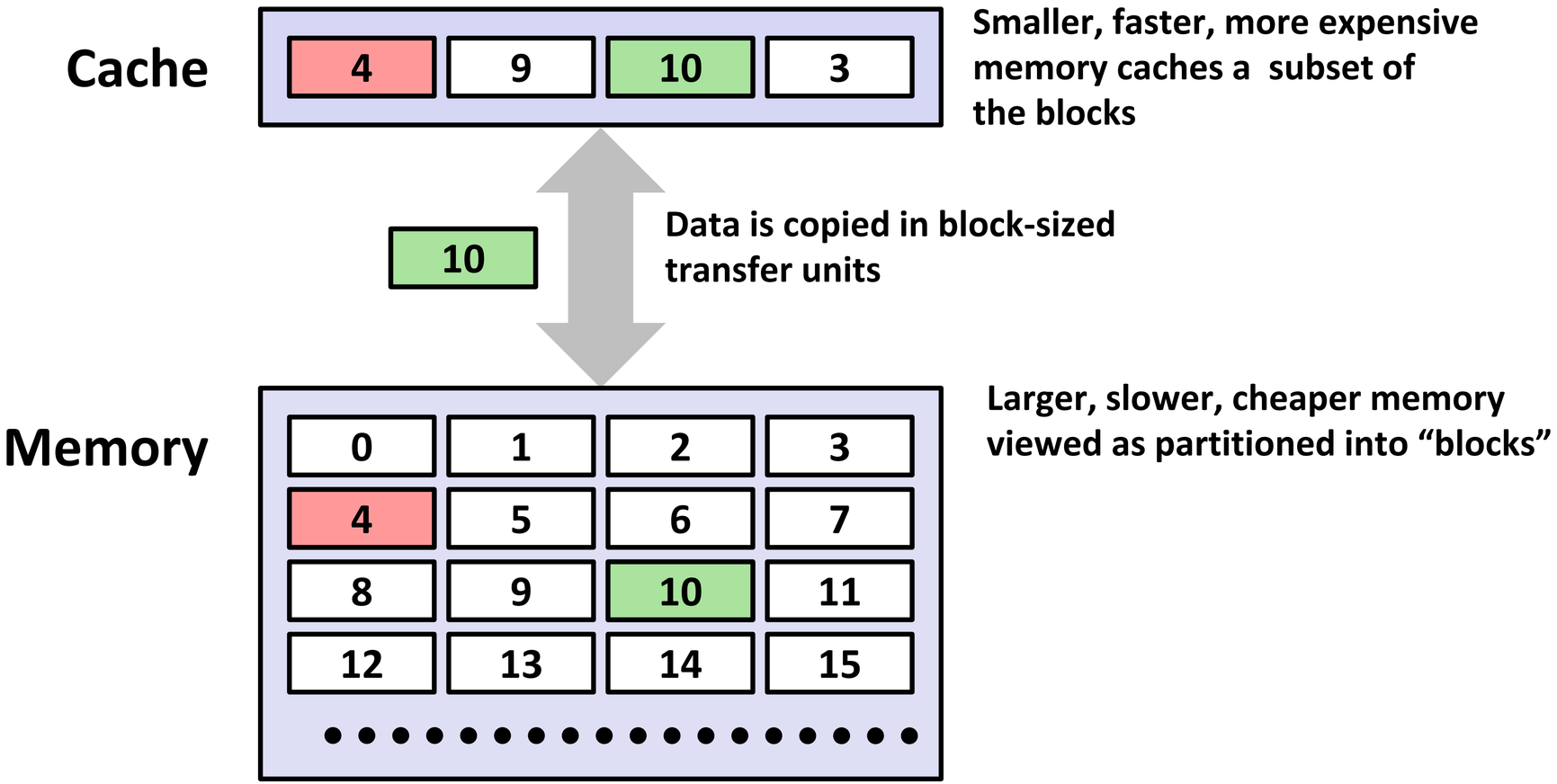
```
int sum_array_3d(int a[M][N][N])
{
    int i, j, k, sum = 0;

    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)?

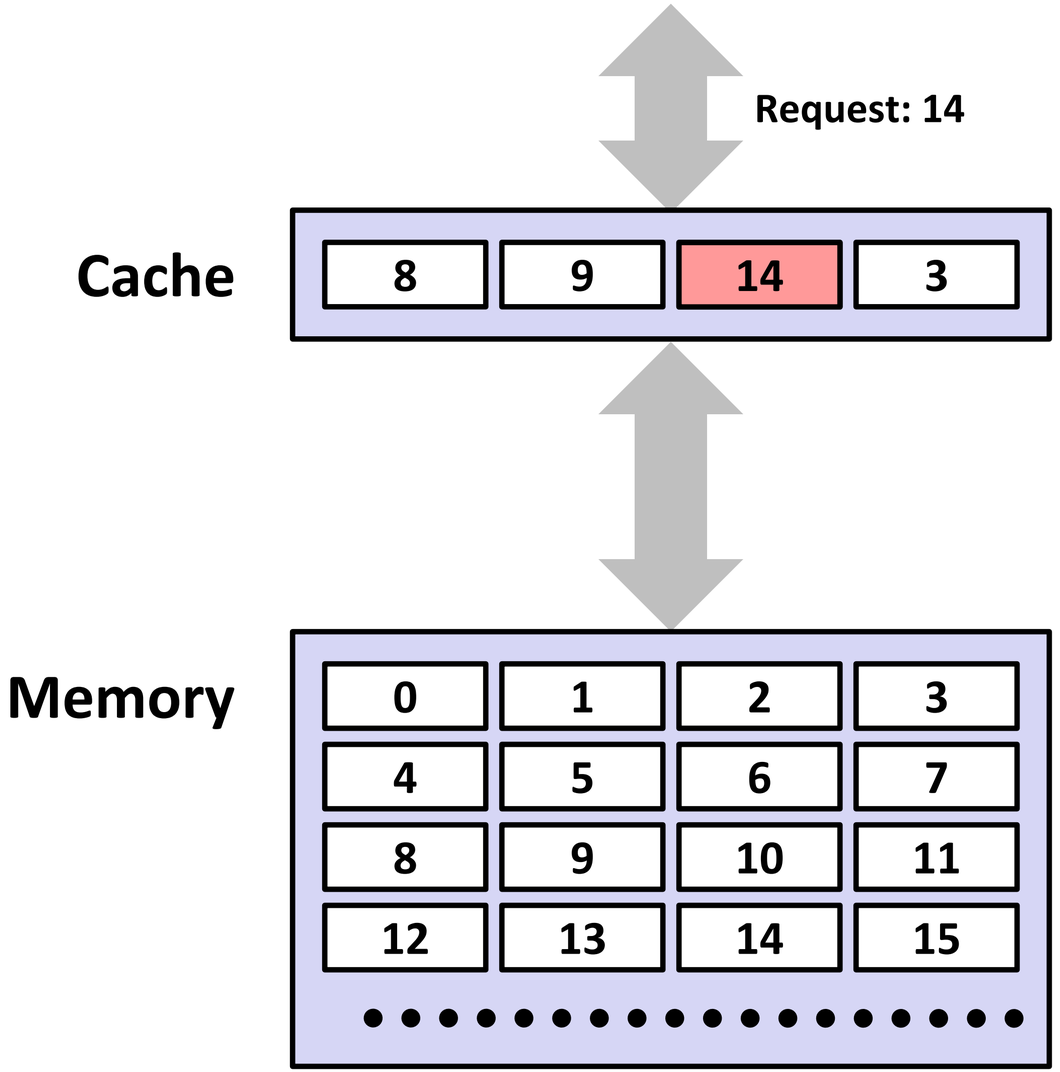
Putting locality into practice: Caches

- 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



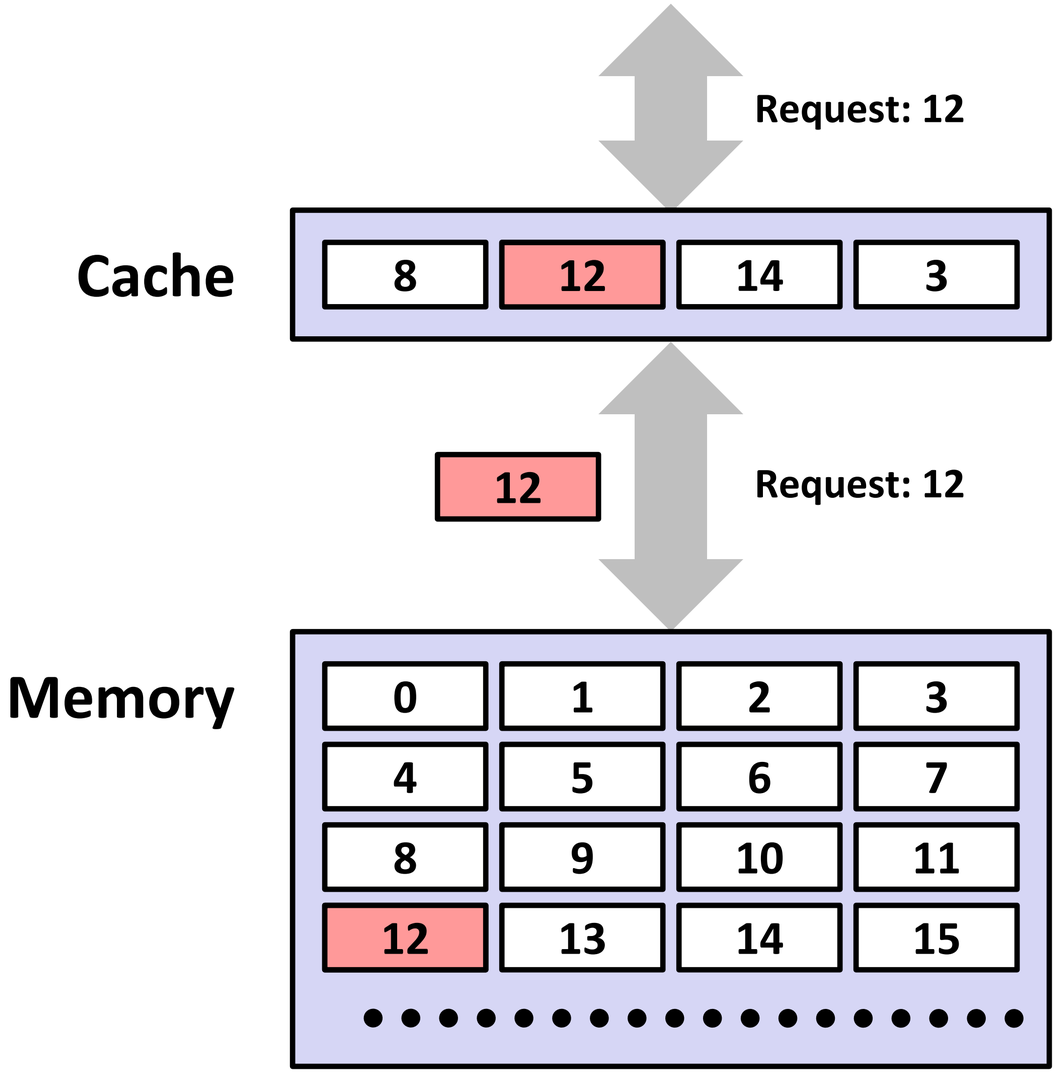
General Cache Concepts: Hit



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!

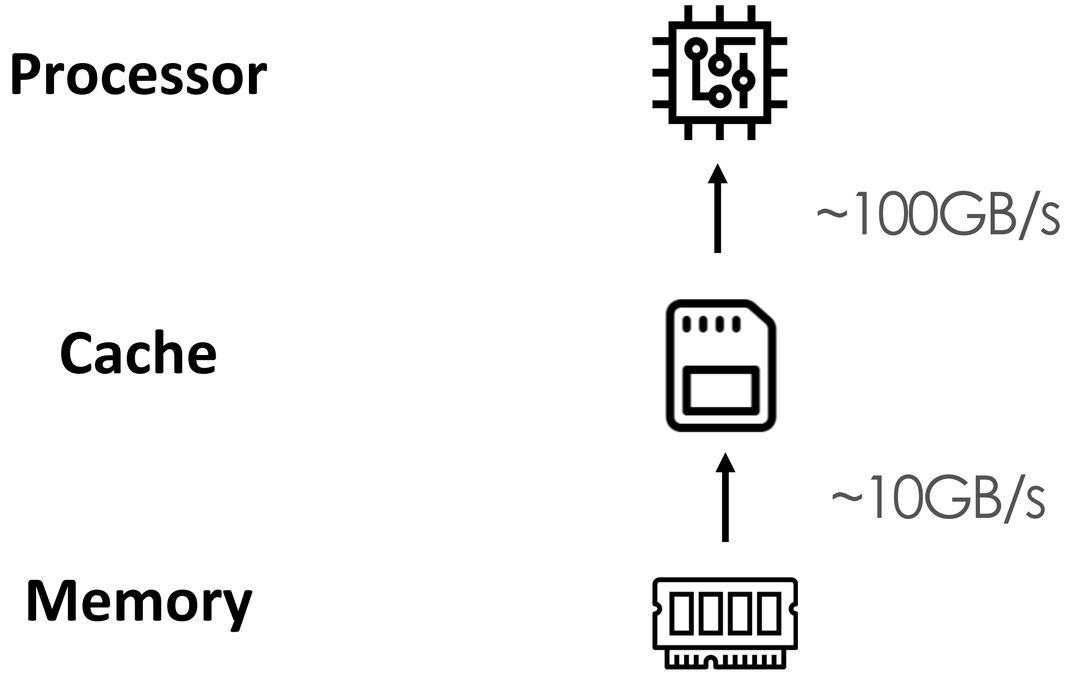
Block 12 is fetched from memory

Block 12 is stored in cache

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

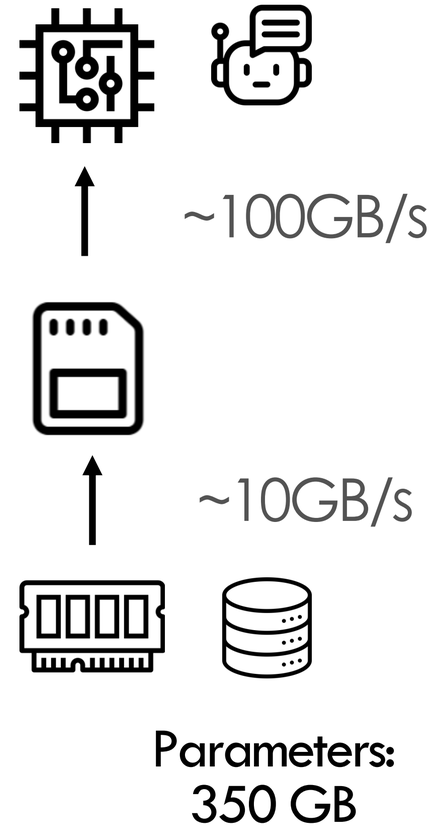
Cache in action

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



Open Question in Cache: ChatGPT

- ChatGPT: every time ChatGPT outputs token, it moves 350GB parameters from memory -> cache -> process
- How to optimize this?



Foundation of Data Systems: where we are

- Computer Organization
  - Representation of Data
  - Processors, memory, storages
- **Operating System Basics**
  - Process, scheduling, concurrency
  - Memory management
  - File systems

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

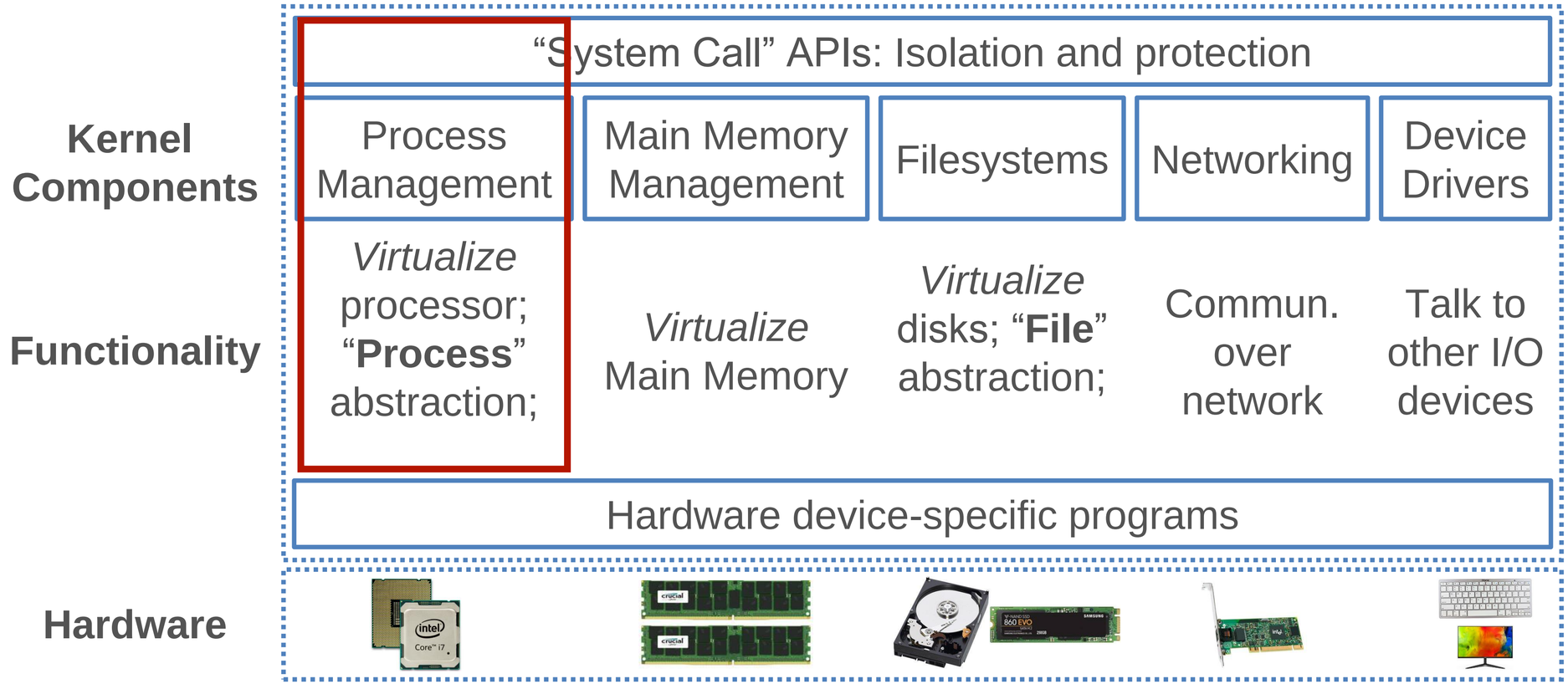
- Problem: What can ill-behaved process do?
  - Go into infinite loop and never relinquish CPU
  - Scribble over other processes' memory to make them fail
- OS provides mechanisms **protection** to address these problems:
  - Preemption – take CPU away from looping process
  - Memory protection – protect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

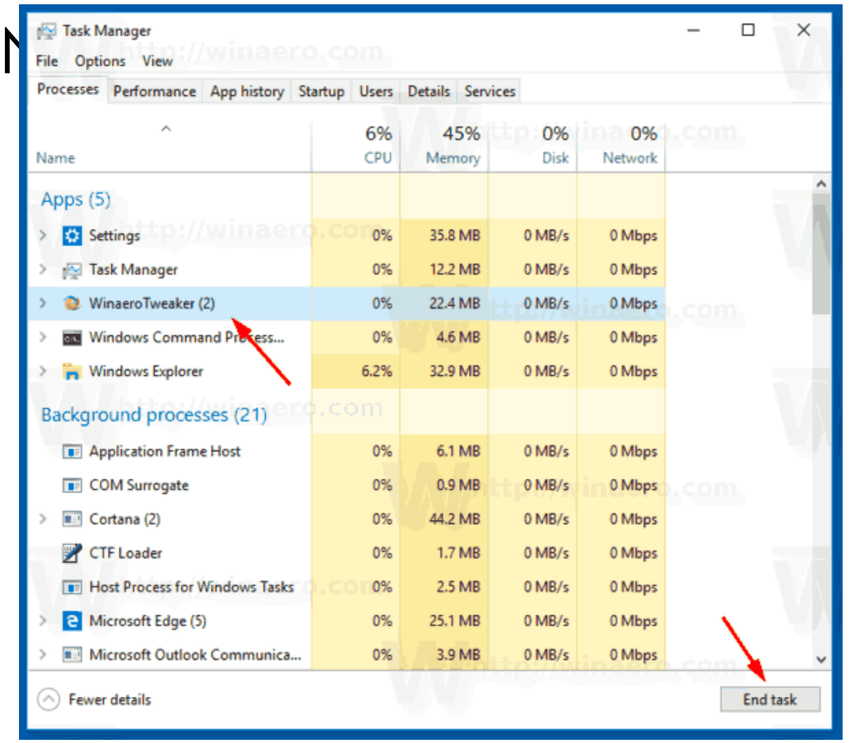


Foundation of Data Systems: where we are

- Computer Organization
  - Representation of Data
  - Processors, memory, storages
- Operating System Basics
  - **Processes, scheduling, concurrency**
  - Memory management
  - File systems

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

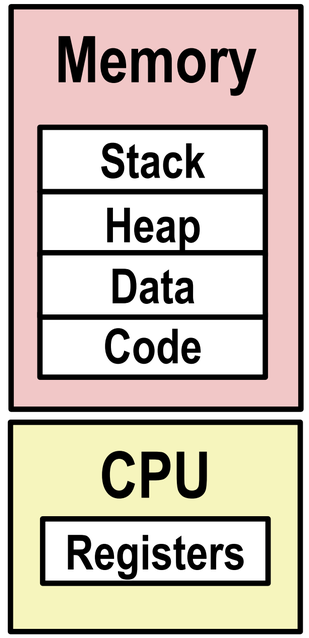
test.py

```
1  print("Good Morning")
2
3
4  def main():
5      print("Hello Python")
6
7
8  print("Good Evening")
9
10
11 if __name__ == "__main__":
12     main()
13
```

```
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 memory.
    - Provided by kernel mechanism called virtual memory



The Abstraction of a Process

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

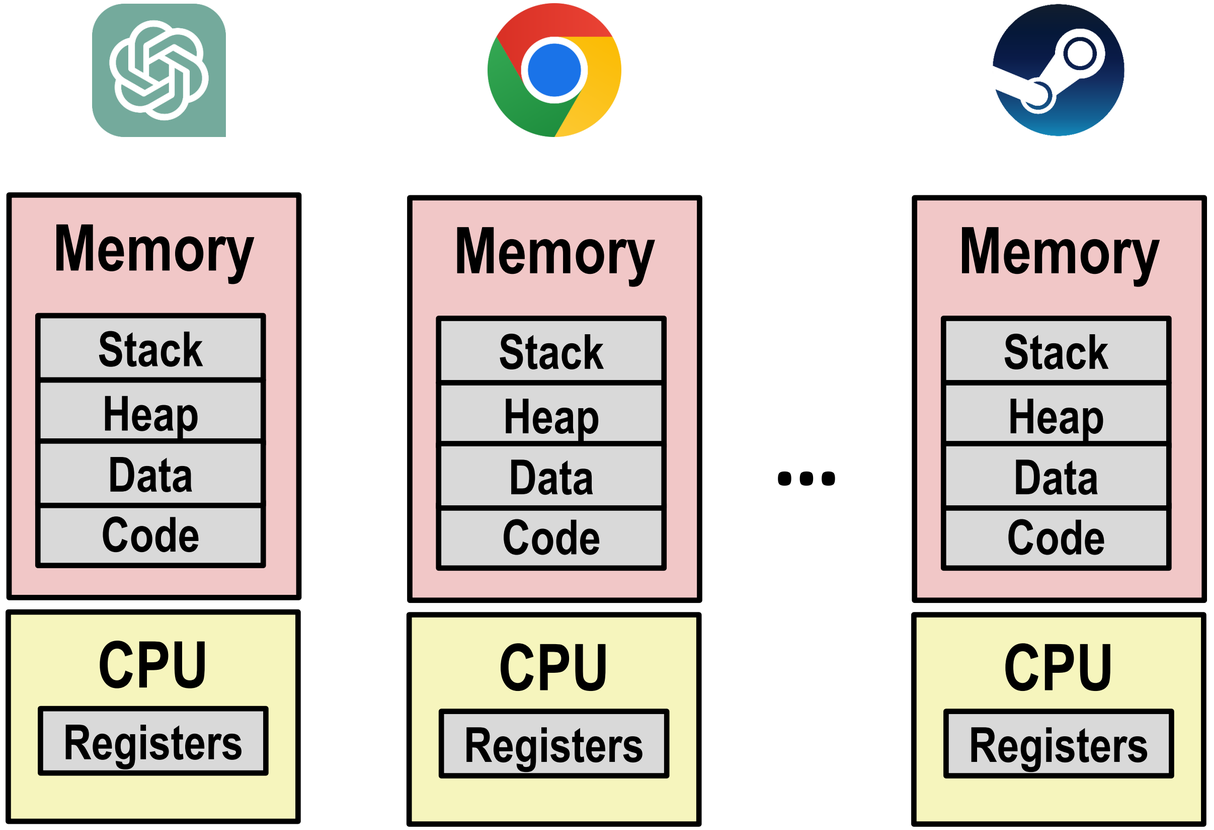
1. Create a process (get Process ID; add to Process List)
2. Assign part of DRAM to process, aka its Address Space
3. Load code and static data (if applicable) to that space
4. Set up the inputs needed to run program's *main()*
5. Update process' State to *Ready*
6. When process is scheduled (*Running*),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

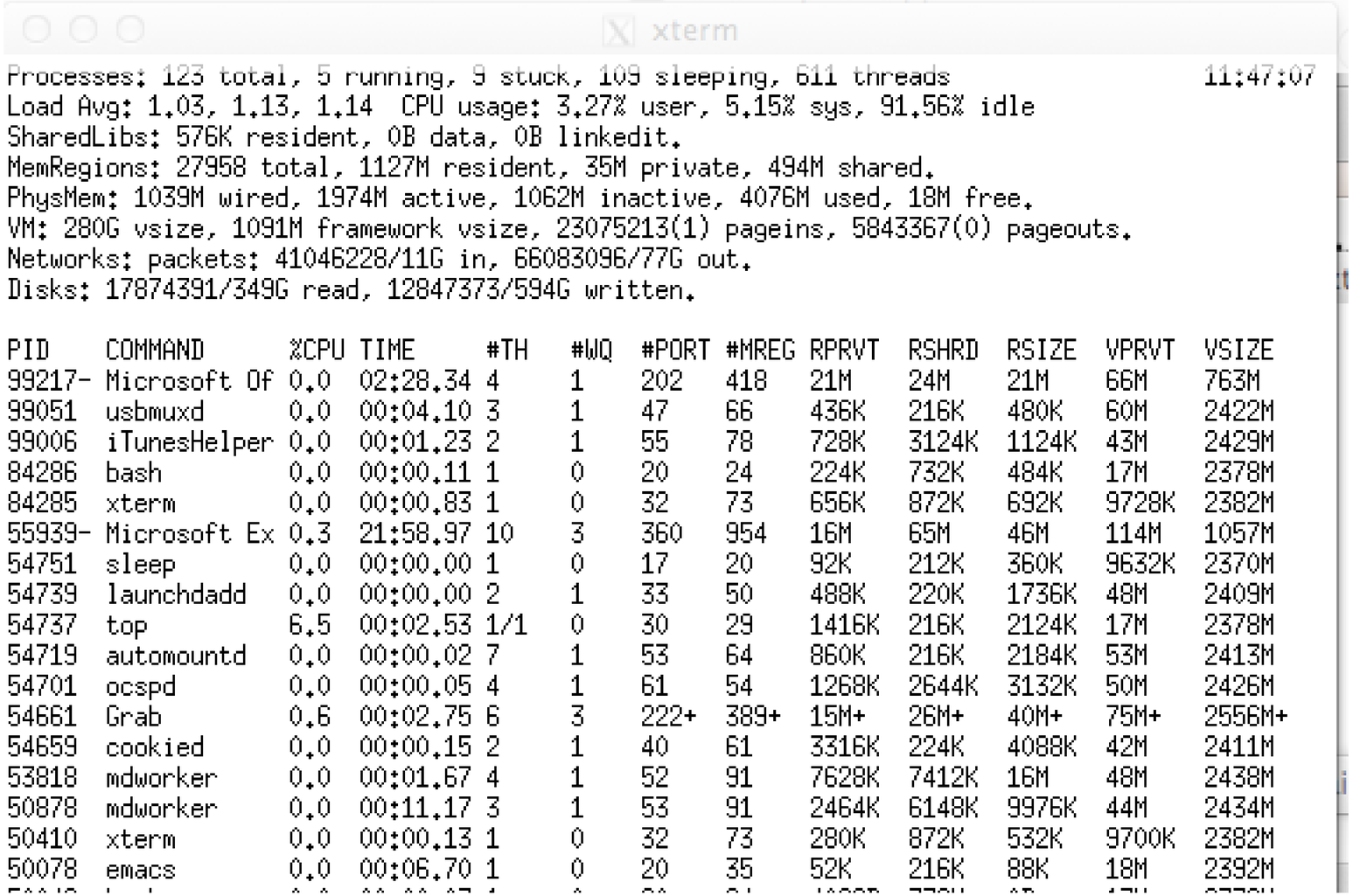
Multiprocessing: The Illusion



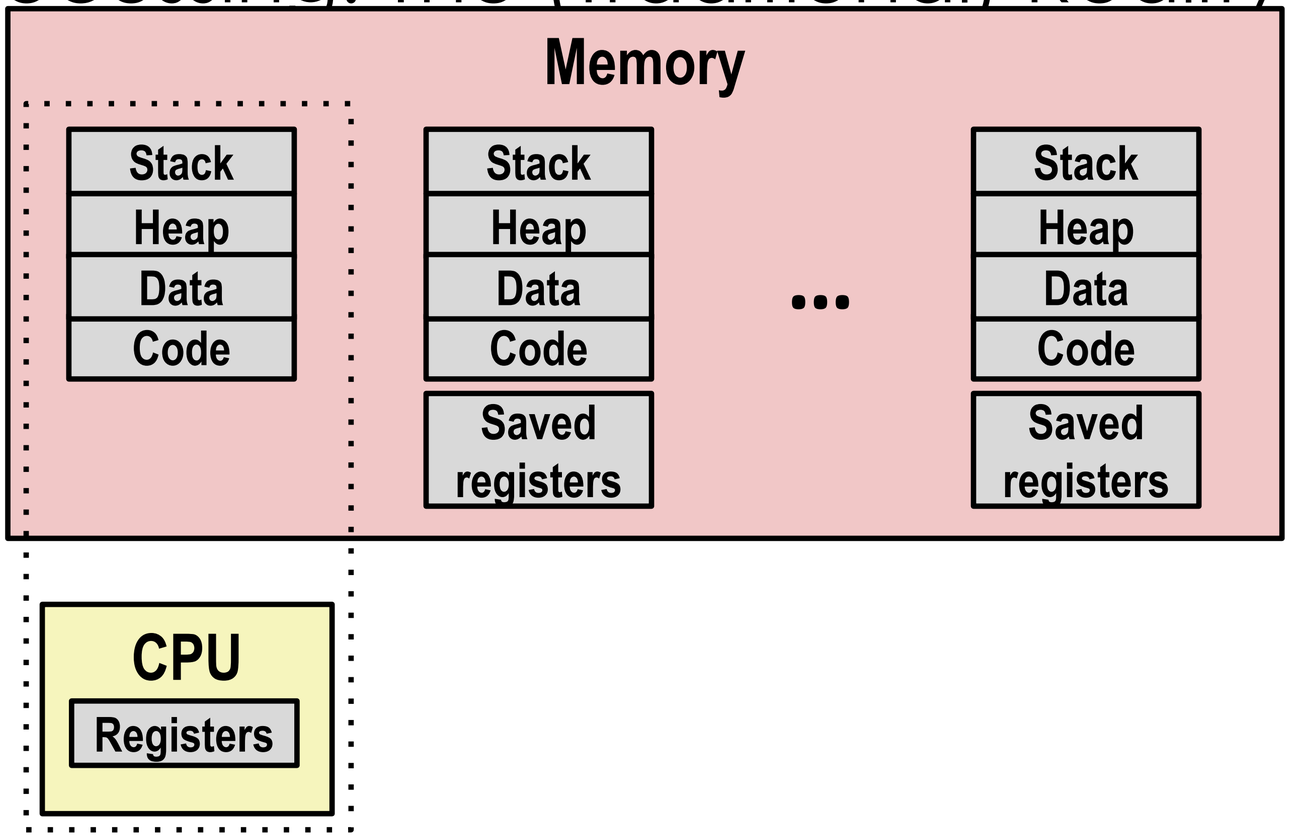
- Computer runs many processes simultaneously

Multiprocessing Example

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

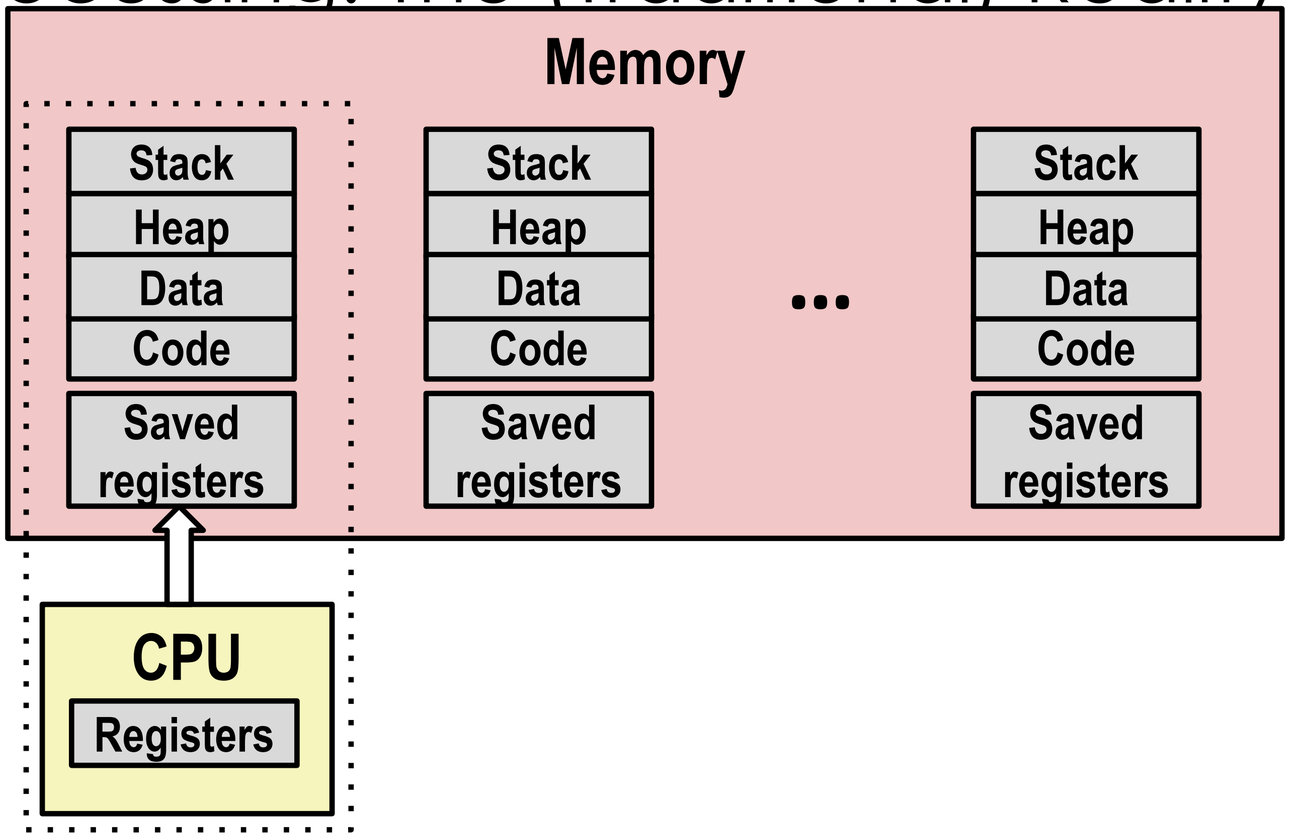


Multiprocessing: The (Traditional) Reality



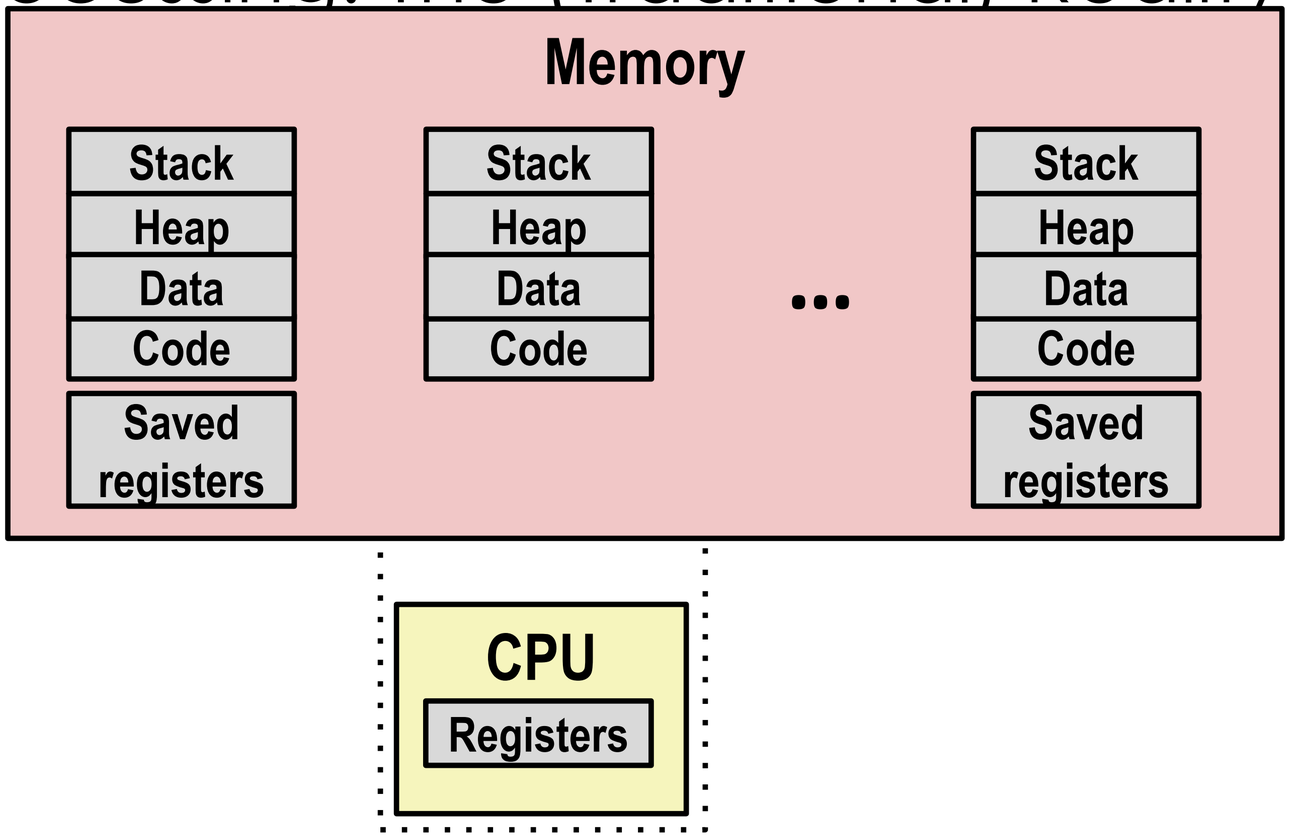
- Single processor executes multiple processes concurrently
  - Process executions interleaved (multitasking)
  - Address spaces managed by virtual memory system (like last week)
  - Register values for nonexecuting processes saved in memory

Multiprocessing: The (Traditional) Reality



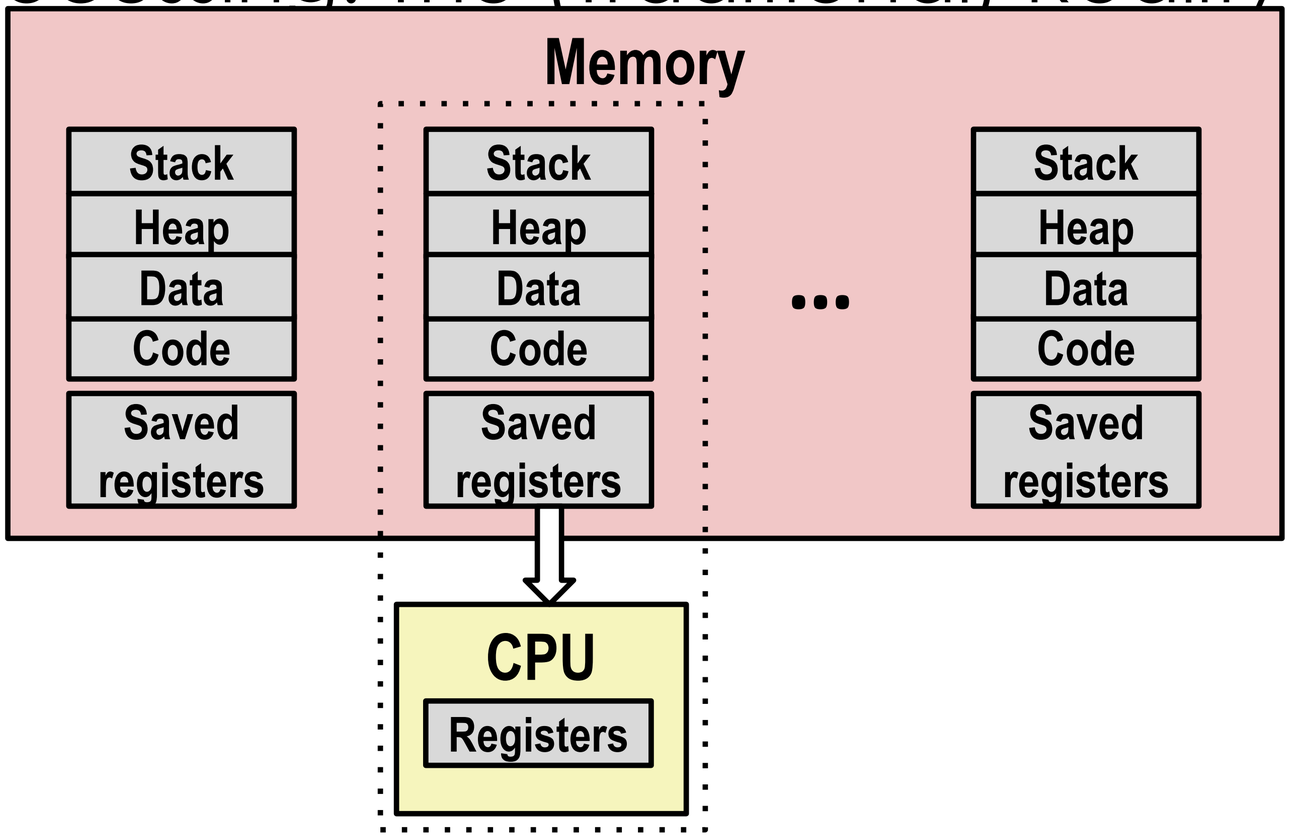
- Save current registers in memory

Multiprocessing: The (Traditional) Reality



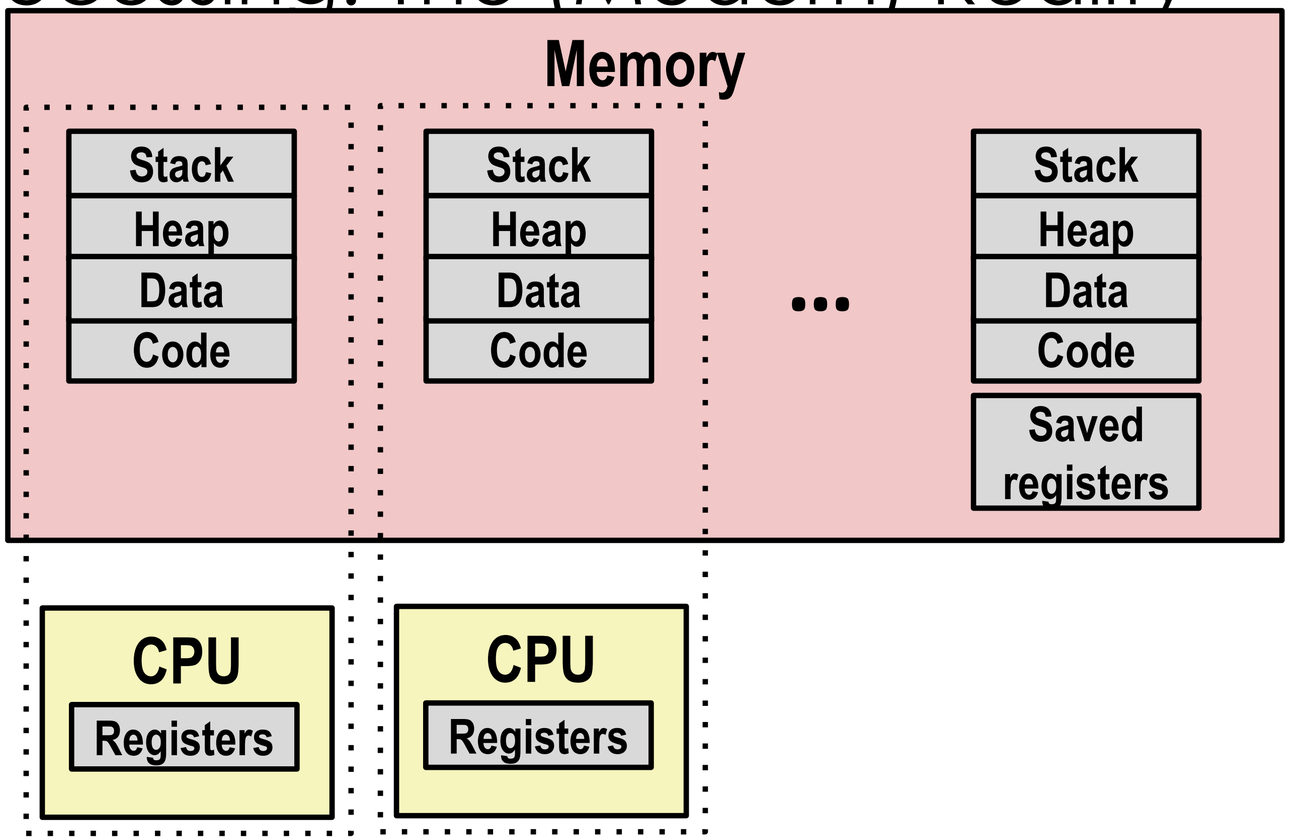
- Schedule next process for execution

Multiprocessing: The (Traditional) Reality



- Load saved registers and switch address space (context switch)

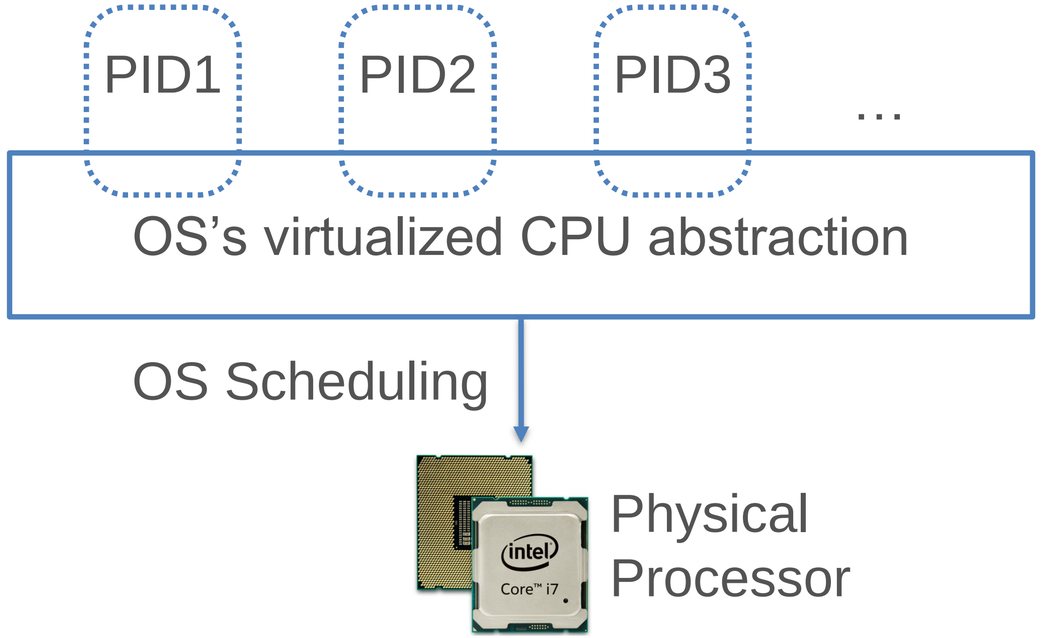
Multiprocessing: The (Modern) Reality



- Multicore processors
  - Multiple CPUs on single chip
  - Share main memory (and some caches)
  - Each can execute a separate process

Inter-process communication (IPC)

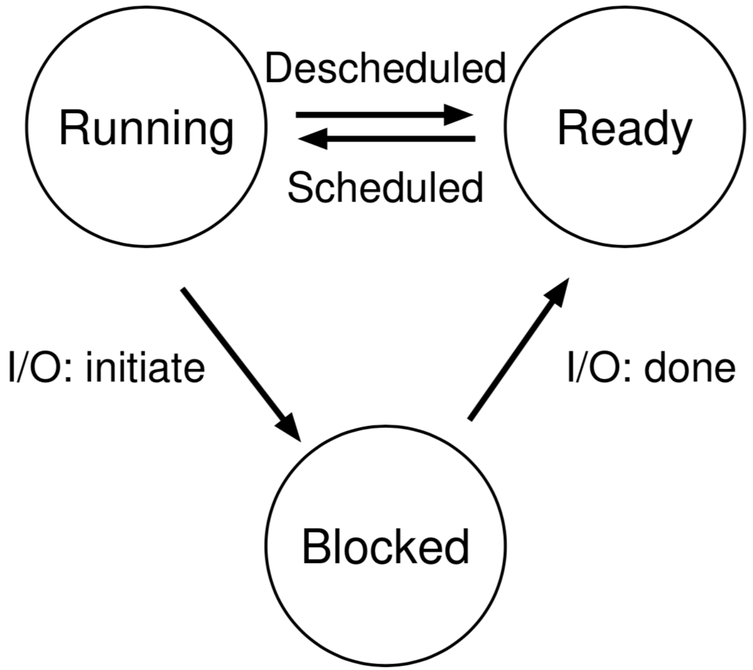
- Virtualization of processor enables process **isolation**, i.e., each process given an “illusion” that it alone runs



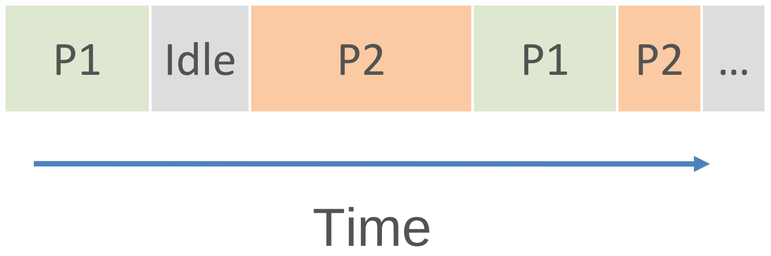
- Inter-process communication possible in System Calls API

Process Management by OS

- OS keeps moving processes between 3 states:

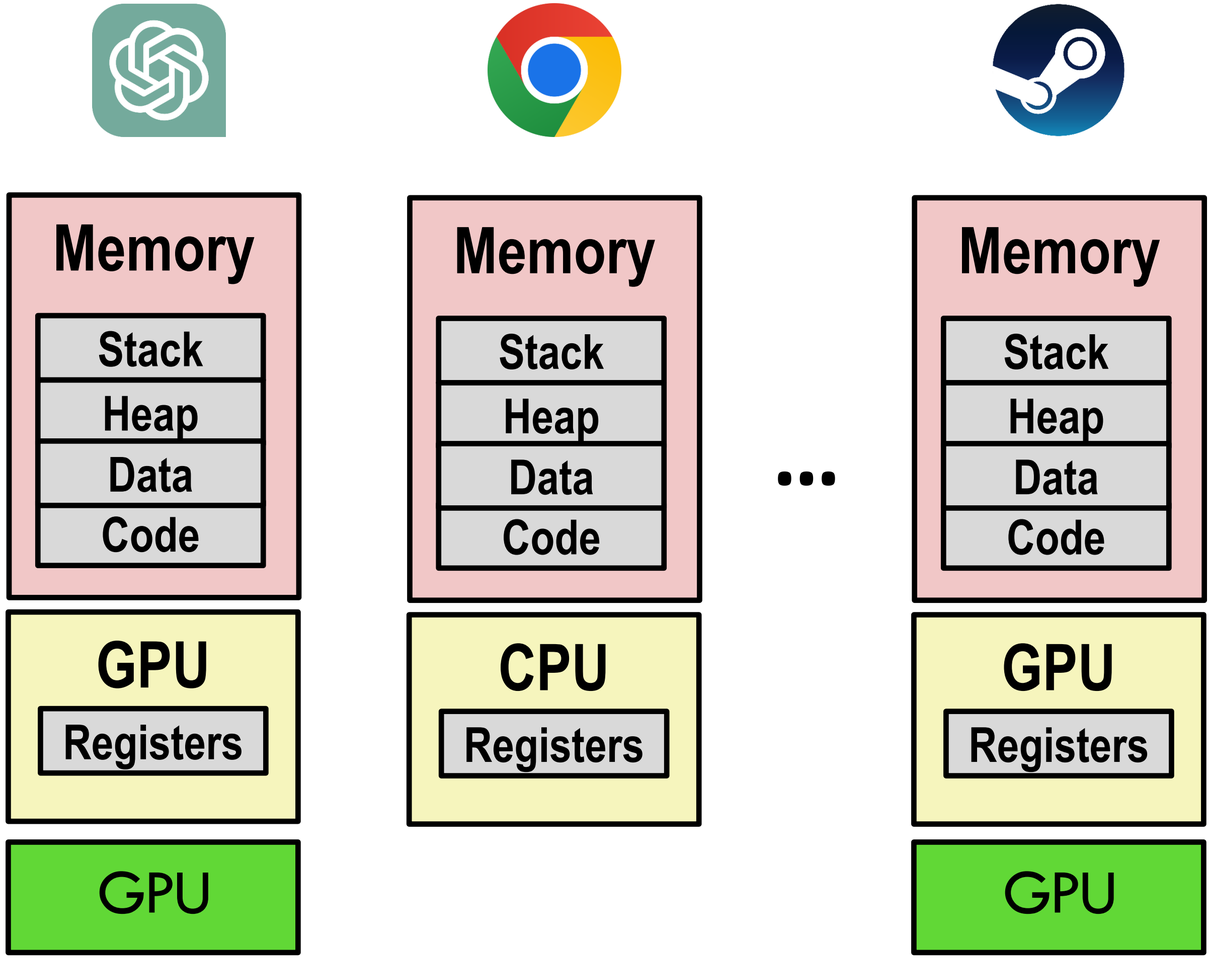


- Gantt Chart: A viz. to show what process runs when (on processor)



- Sometimes, if a process gets “stuck” and OS did not schedule something else, system **hangs**; need to reboot!

Starting Qs for next class



- Q1: What if GPUs are in?
- Q2: How GPUs differ from CPUs?