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

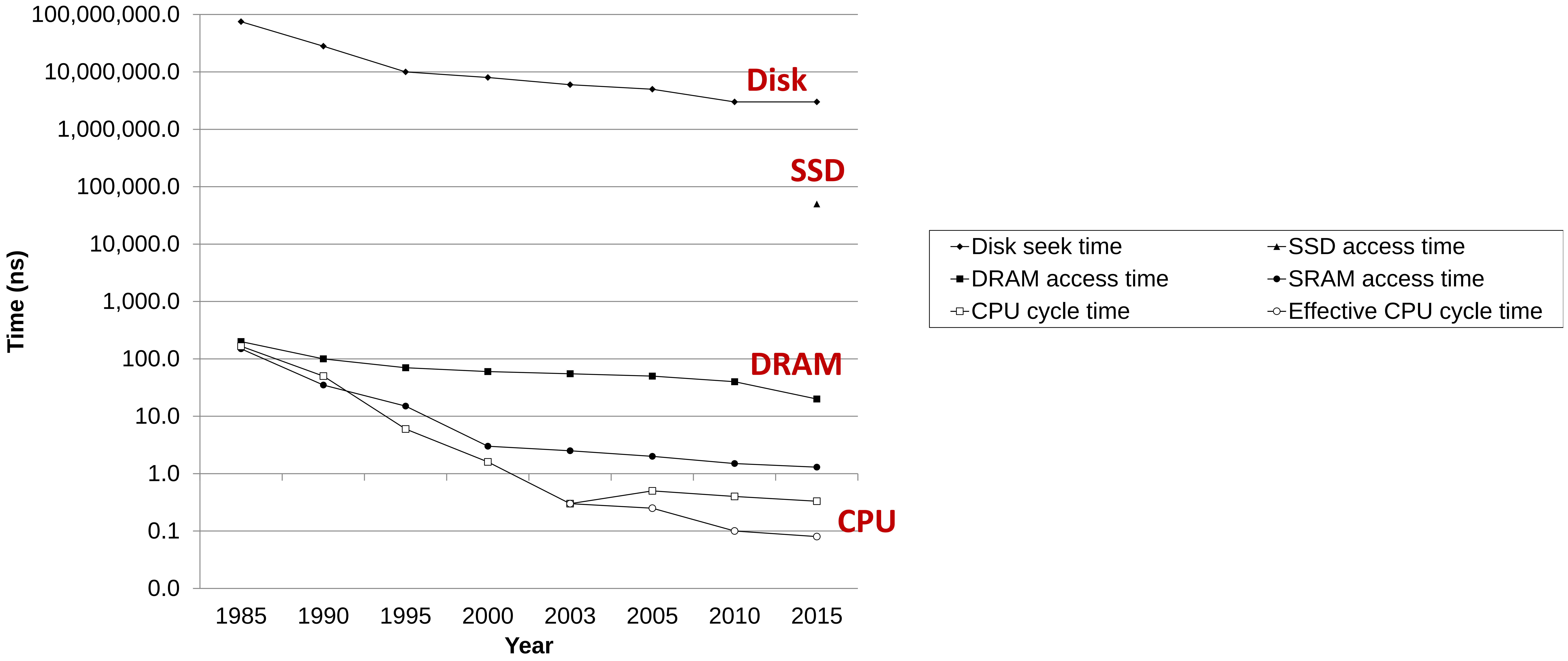
- Readings are uploaded
  - Only 1 per class, multiple optional
  - This week's readings: OS processes and memory management
- Reading summary due: Next Wednesday 1/24
  - Submit via GradeScope
  - Follow the NeurIPS template: maximum 2 pages.
- Next week readings will be out:
  - By this Saturday

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 computers work

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

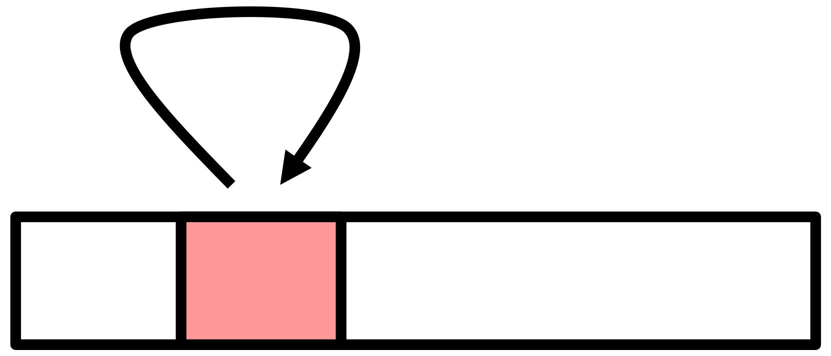
```
void copyji(long int src[2048][2048], long int dst[2048][2048])
{
  long int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

81.8 milliseconds

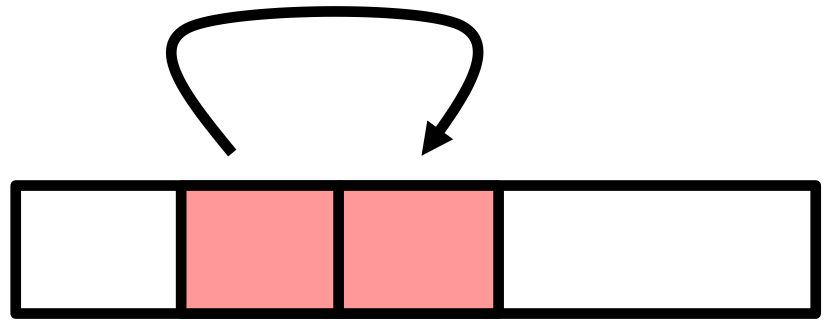
Locality

- 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

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

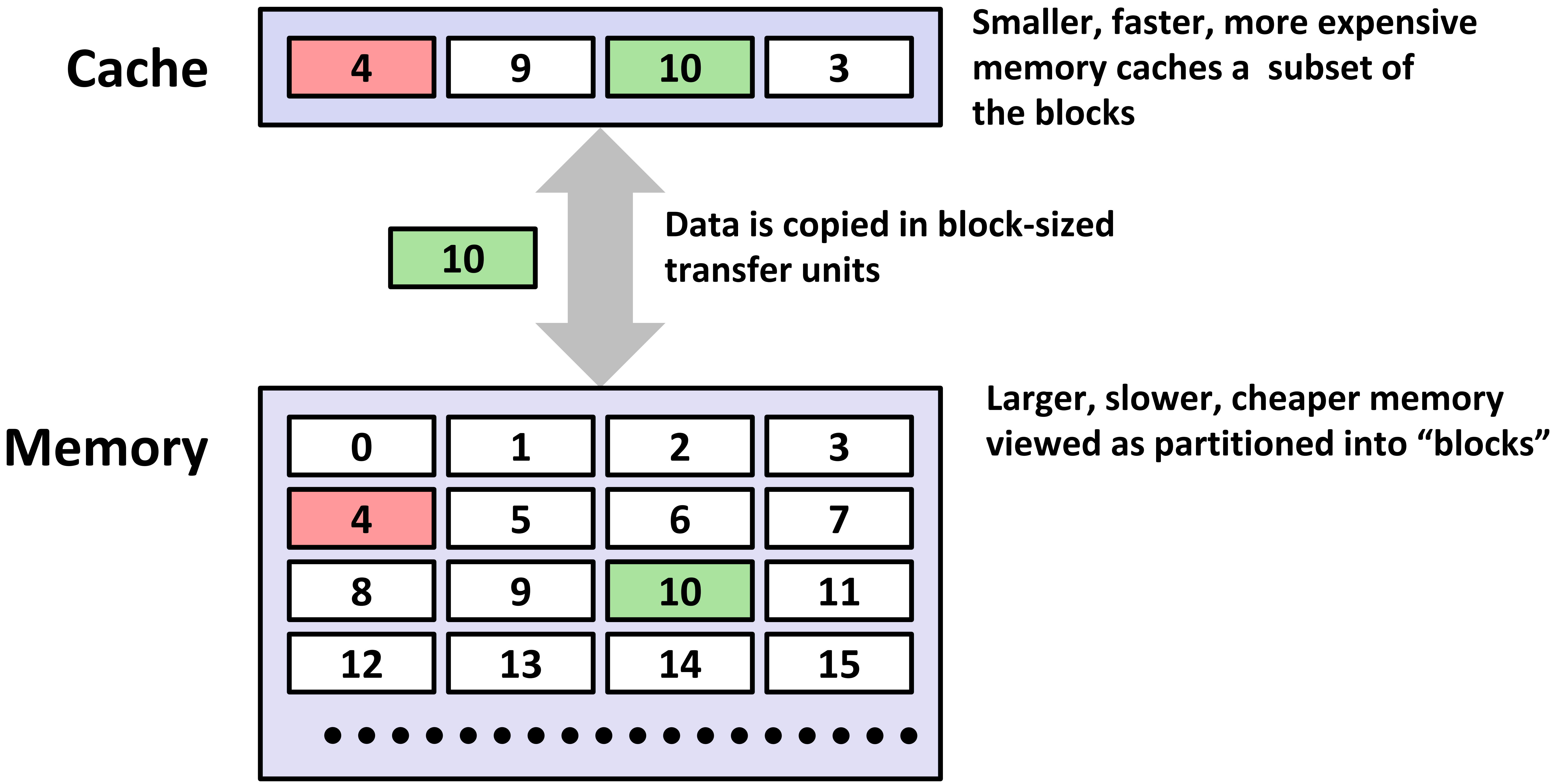
```
int sum_array_3d(int a[M][N][N])
{
    int i, j, k, sum = 0;

    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)?

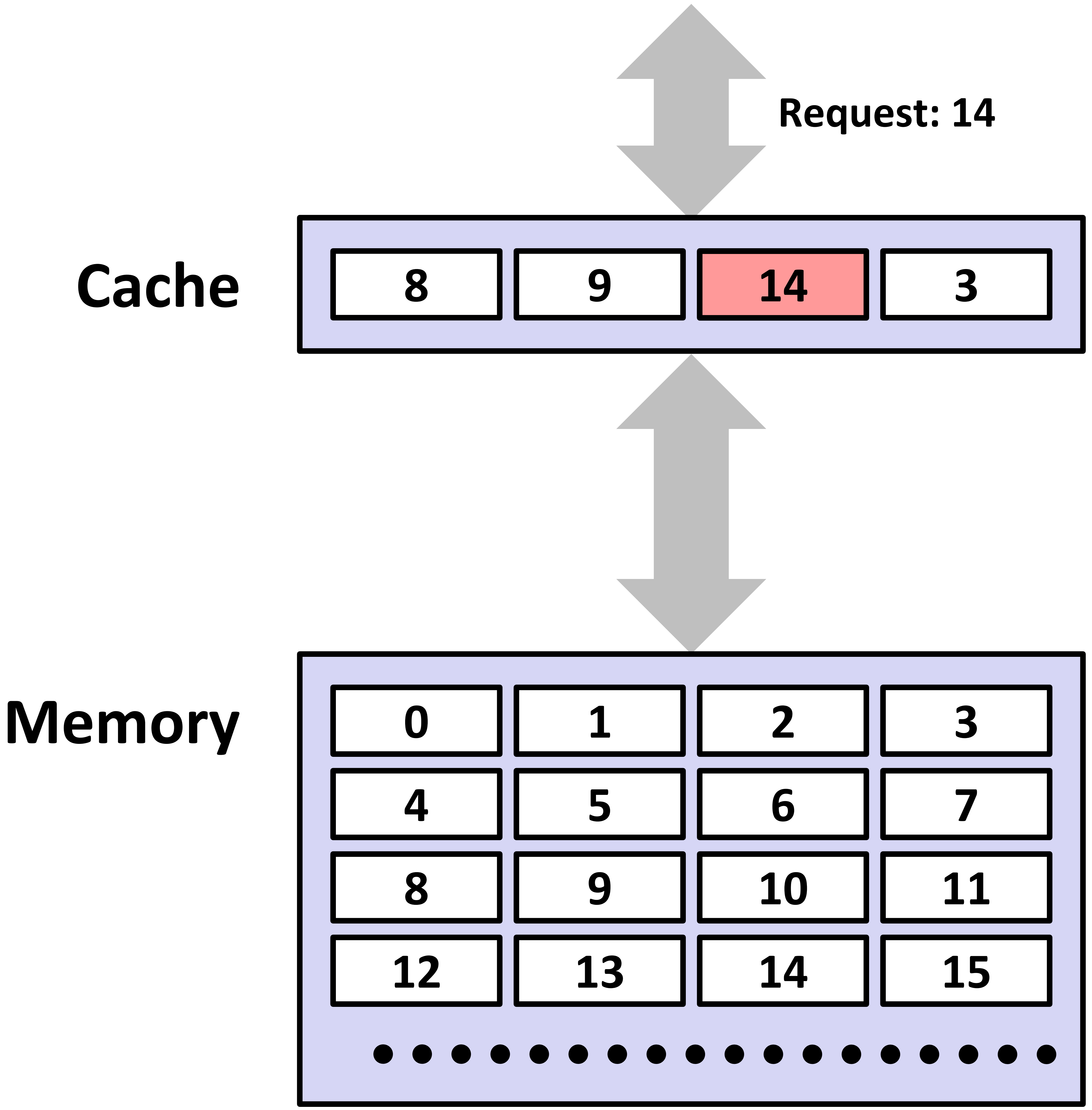
Putting locality into practice: Caches

- 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



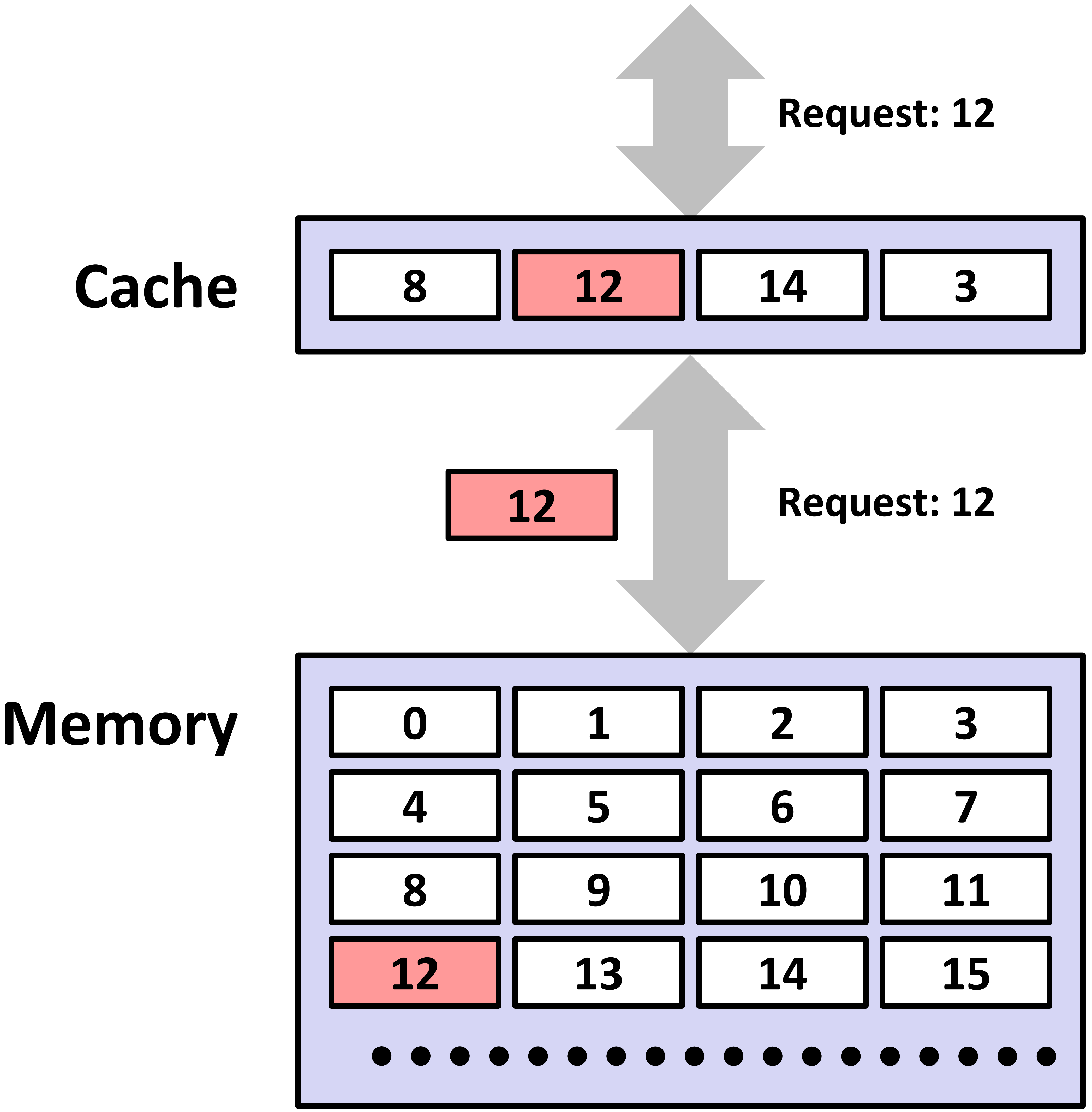
General Cache Concepts: Hit



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)

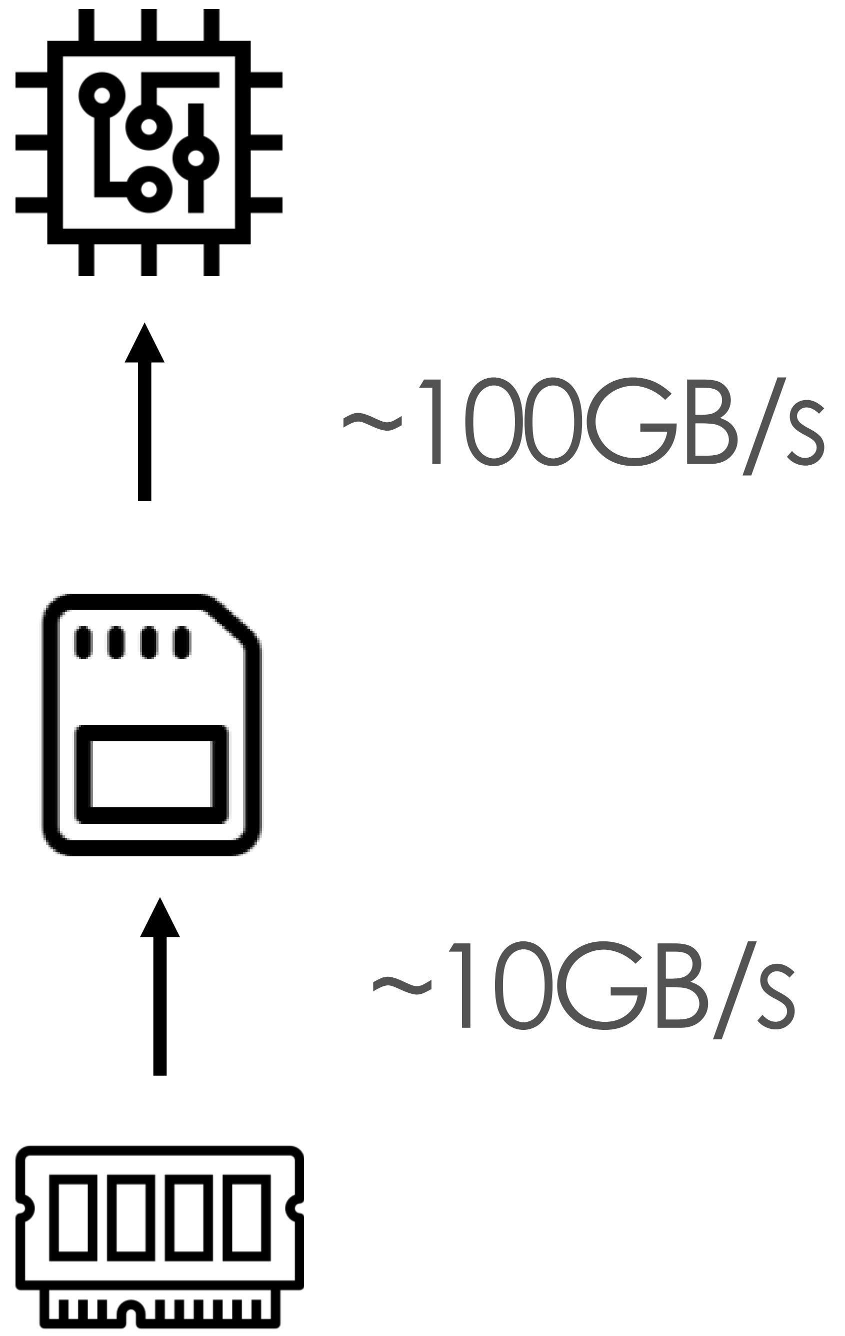
Cache in action

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

Processor

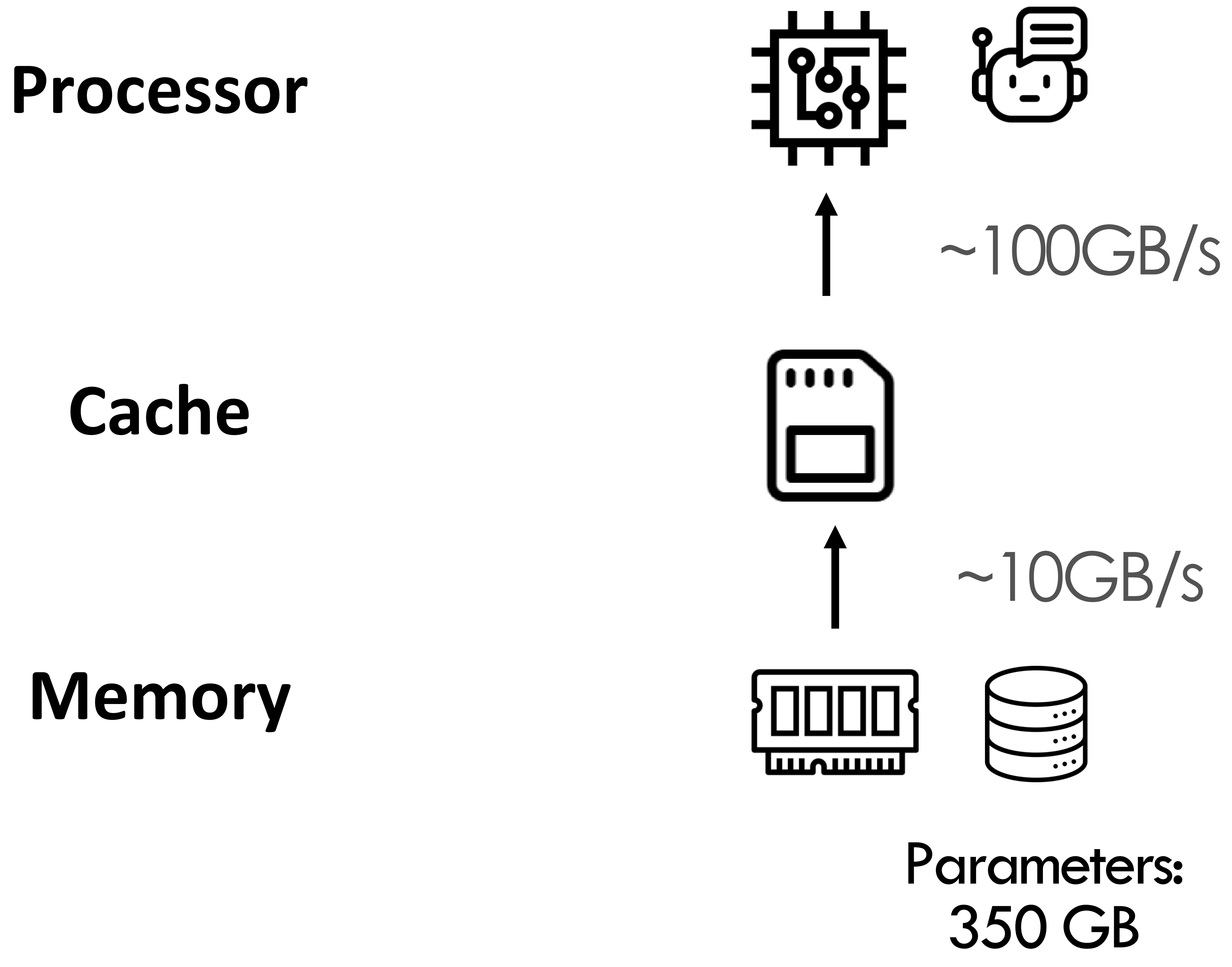
Cache

Memory



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: 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]

OS: interfaces above hw drivers

Hardware

- 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

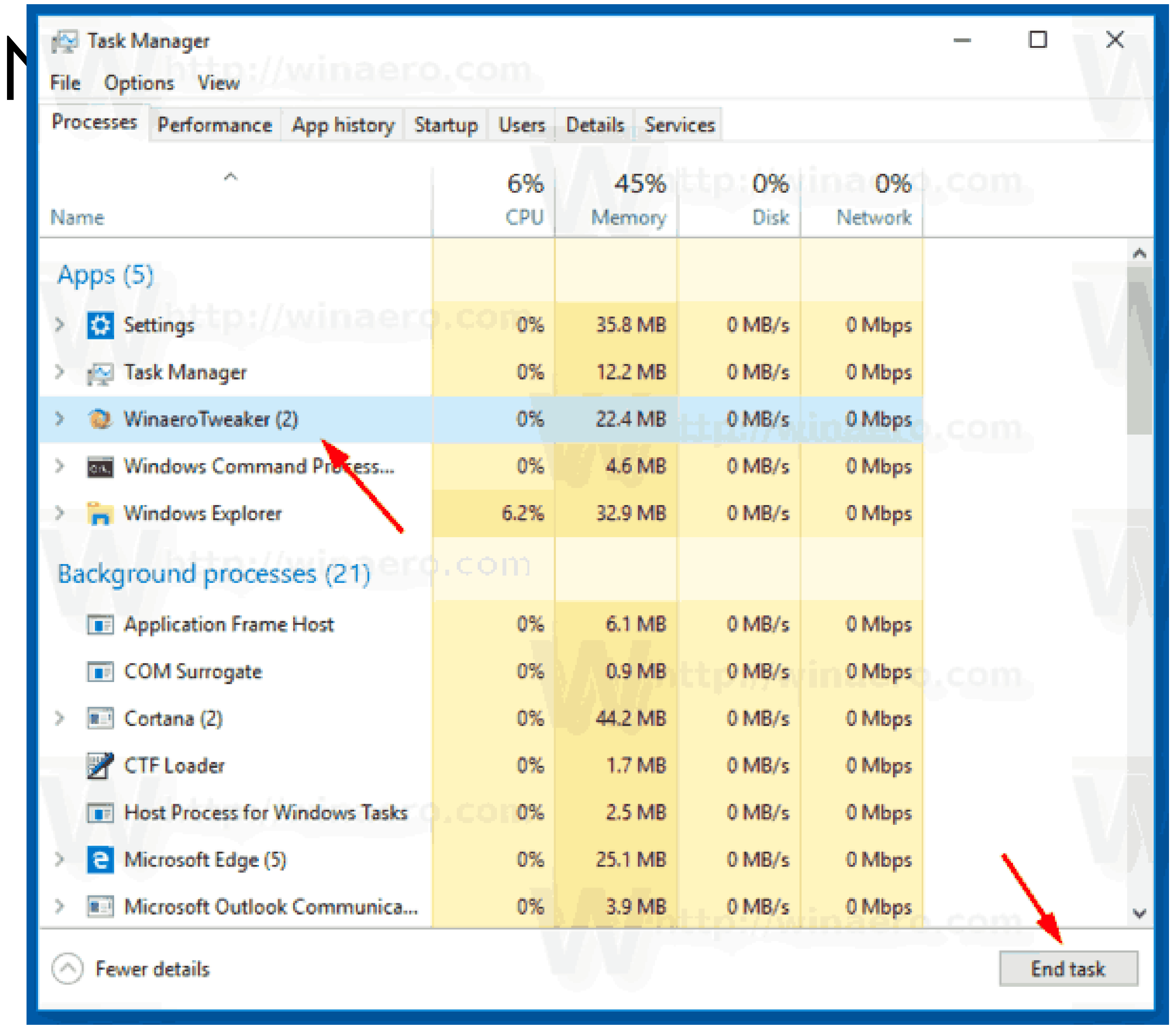
	"System Call" APIs: Isolation and protection				
Kernel Components	Process Management	Main Memory Management	Filesystems	Networking	Device Drivers
Functionality	*Virtualize* processor; "**Process**" abstraction;	*Virtualize* Main Memory	*Virtualize* disks; "**File**" abstraction;	Commun. over network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

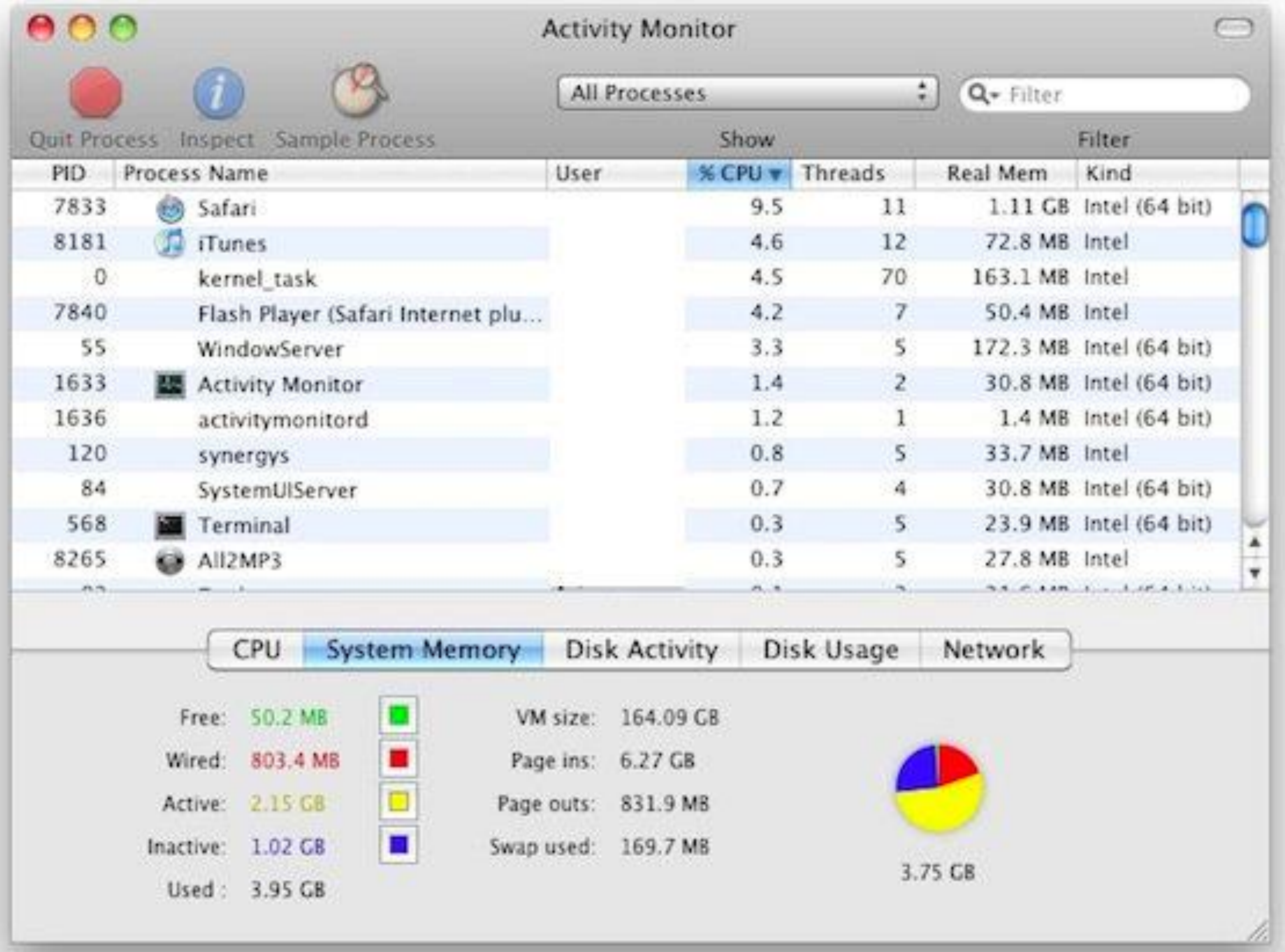
Foundation of Data Systems: where we are

- 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

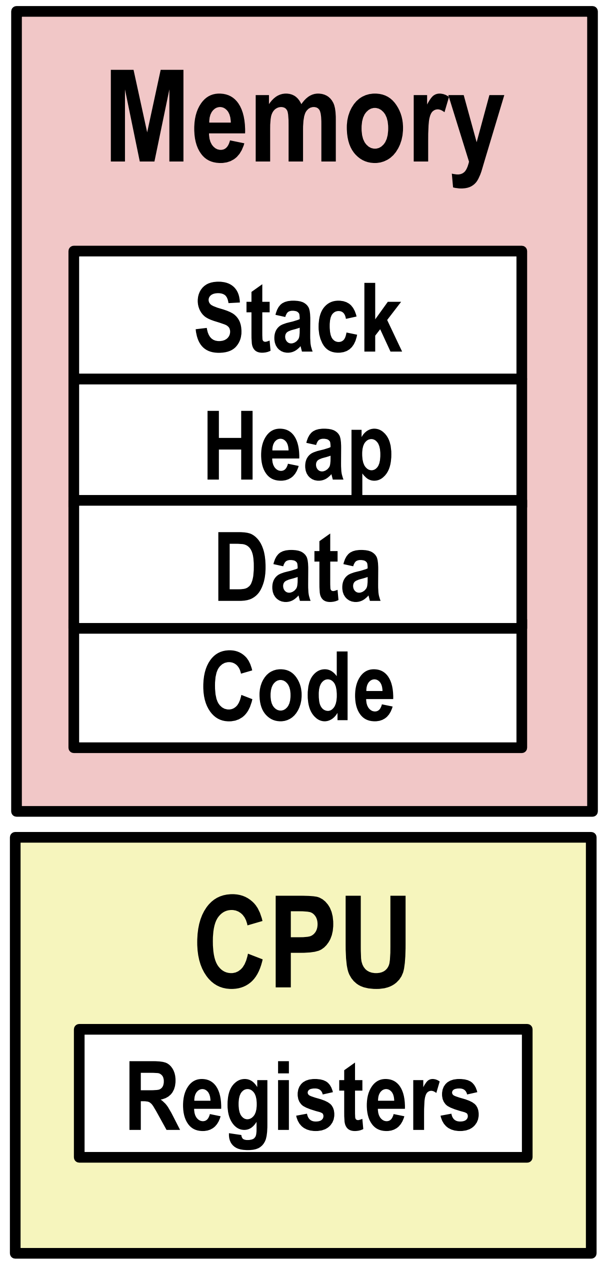
test.py

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



The Abstraction of a Process

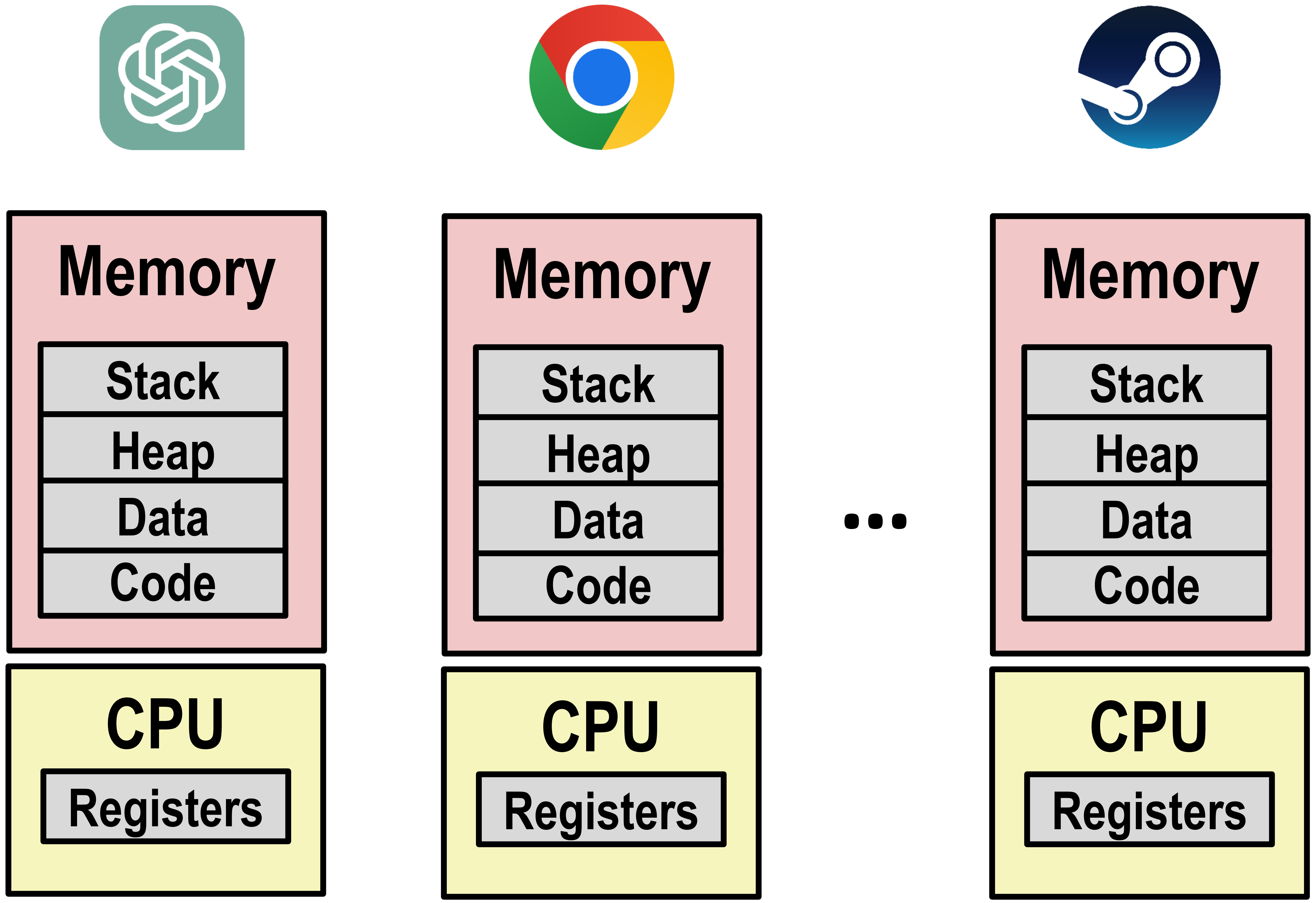
- High-level steps OS takes to get a process going:
  1. Create a process (get Process ID; add to Process List)
  2. Assign part of DRAM to process, aka its Address Space
  3. Load code and static data (if applicable) to that space
  4. Set up the inputs needed to run program's *main()*
  5. Update process' 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

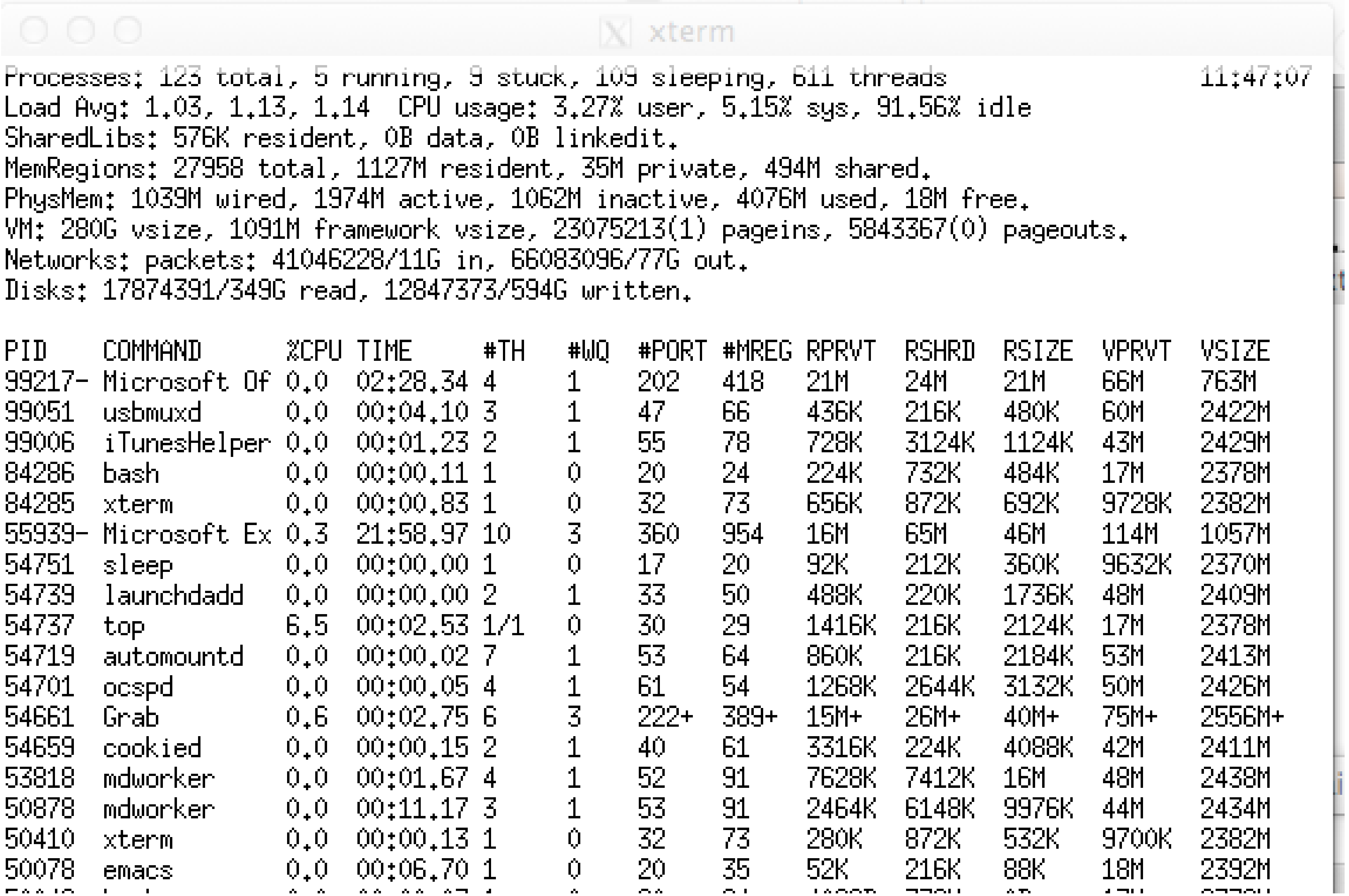
Multiprocessing: The Illusion



- Computer runs many processes simultaneously

Multiprocessing Example

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



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