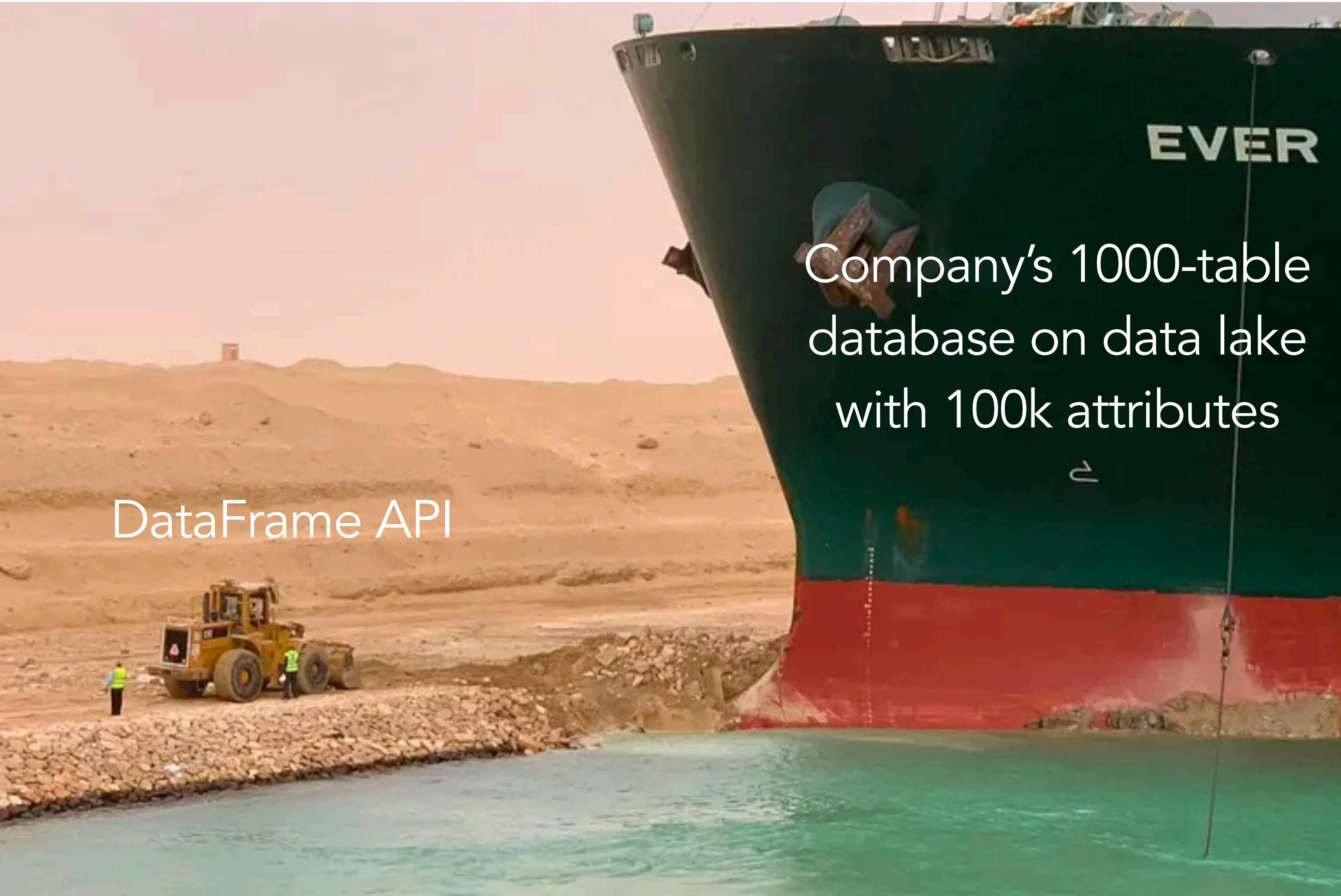


DSC 204a

Scalable Data Systems

- Haojian Jin



Where are we in the class?

Foundations of Data Systems

- Digital representation of Data → Basics of Computer Organization → Process and memory hierarchy → **Basics of Operating Systems**
- Data encoding and evolution

Scaling Distributed Systems

Data Processing and Programming model

Today

Process management: virtualization & Concurrency

Filesystems and Data Files

Main memory management

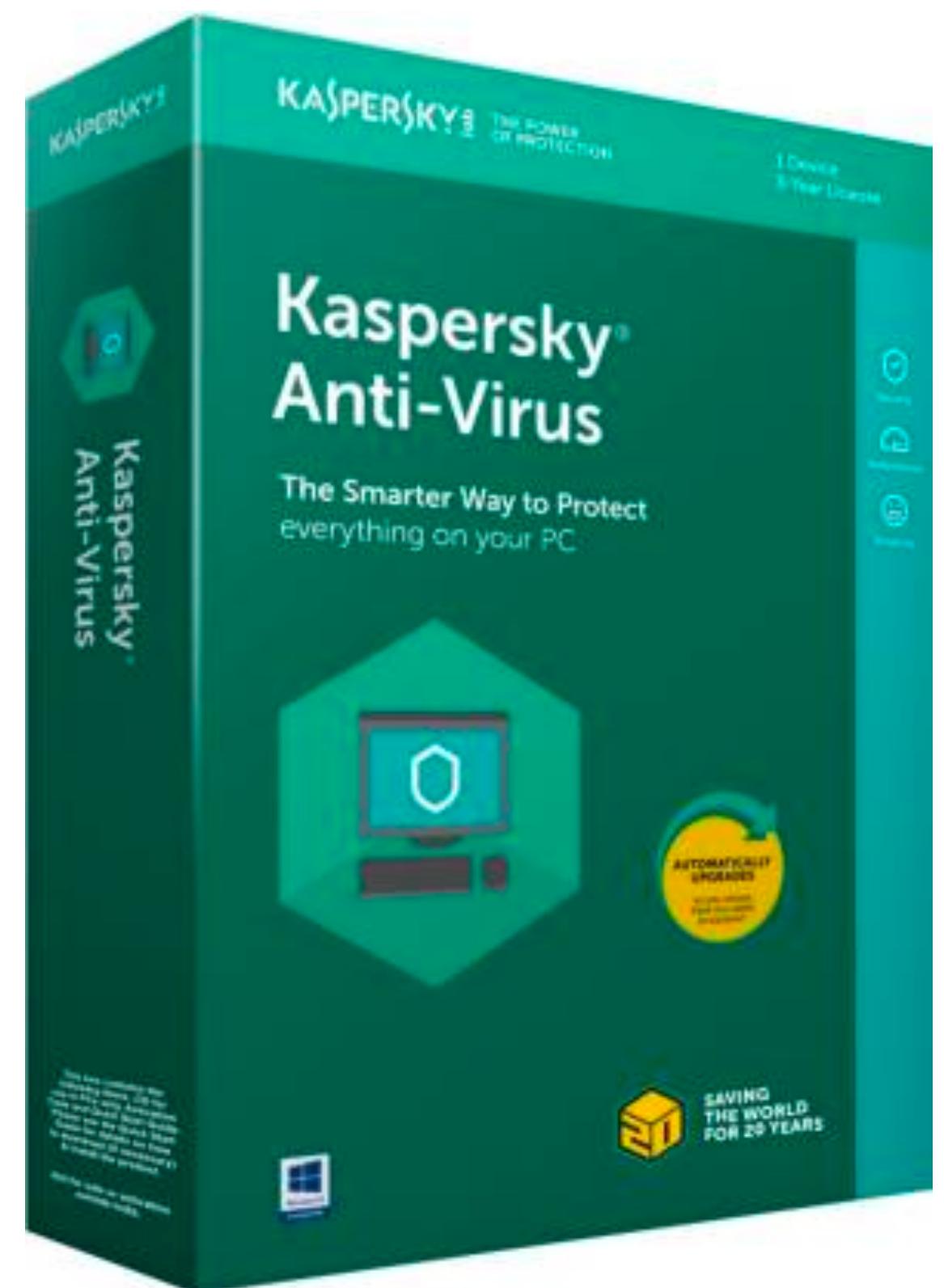
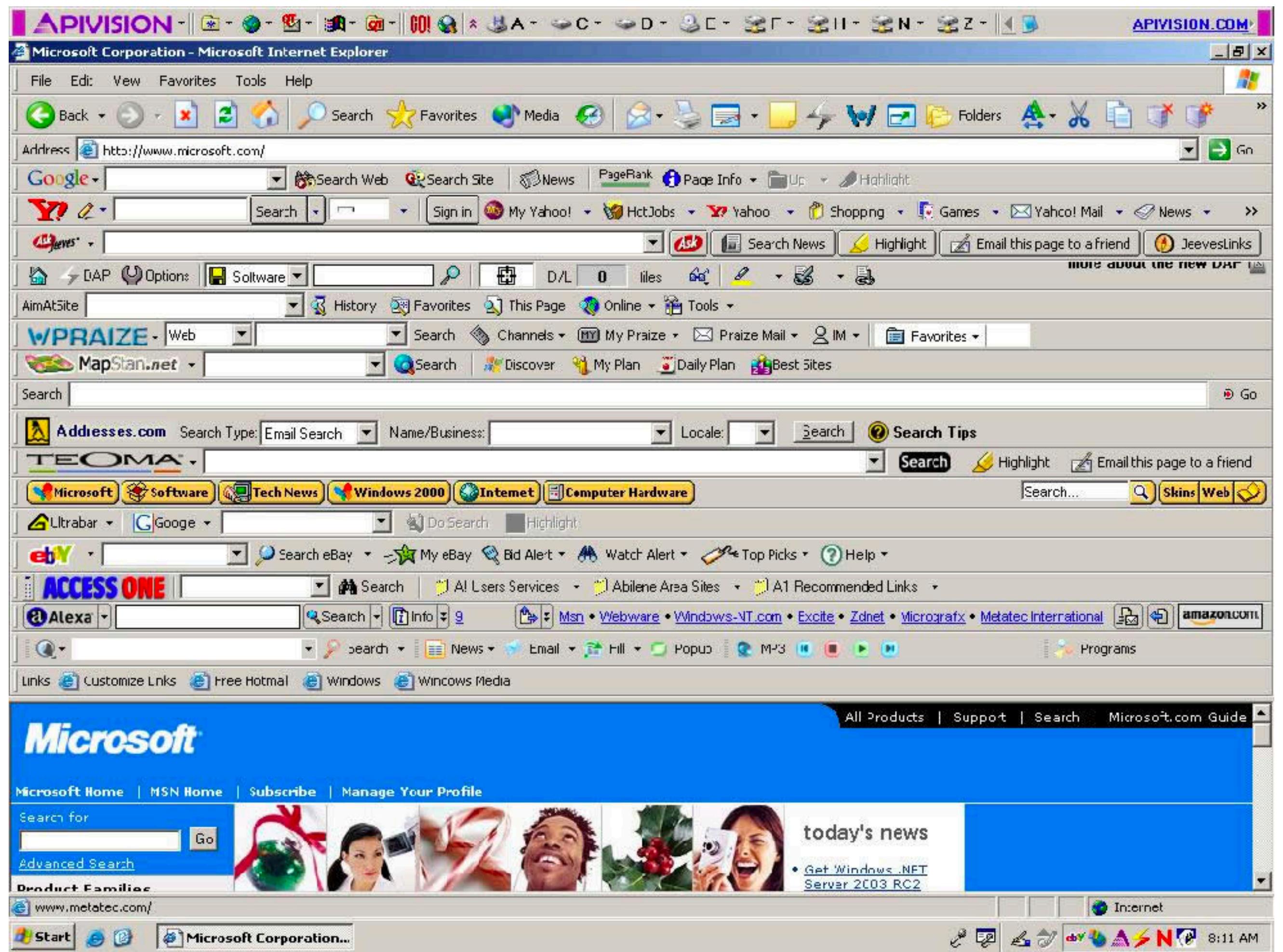
IO & Unix Pipes

Q: What is an OS? Why do we need it?



Role of an OS in a Computer

- An OS is a large set of interrelated programs that *make it easier* for applications and user-written programs to use computer hardware **effectively, efficiently, and securely**
 - Akin to a government's role in a country
 - Without OS, computer users must speak machine code!



App Stores

*“changed how software development worked,
and expanded the **number of people who could
comfortably, safely use a computer** from a few
hundred million to a few billion.”*

Technical idea #1

Putting apps in a sandbox

Apps can only do things that Apple allows and cannot ask (or persuade, or trick) the user for permission to do ‘dangerous’ things.



- Would this break my phone?
- Would this run my battery down?
- Steal my bank details?

Technical idea #2

Distributing software through a centralized app store

The screenshot shows a software download page for WinRAR (64-bit). At the top, there's a navigation bar with a 'Download' button and a search bar. Below it, a message says 'Download offers the opportunity to buy software and apps. When you buy through our links, we may get a commission.' The main content features a large 'WinRAR (64-bit)' logo and a 'FREE TO TRY' button. There are two prominent buttons: 'DOWNLOAD NOW' in green and 'PREMIUM UPGRADE'. Below these, a section titled 'Key Details of WinRAR (64-bit)' lists the following features:

- Take full control over RAR and ZIP archives, along with unpacking a dozen other archive formats
- Last updated on 11/25/21
- There has been 1 update within the past 6 months
- The current version has 1 flag on VirusTotal
- Also available on Android and Mac

At the bottom, there's a screenshot of the WinRAR application interface showing file management options like 'Archive name and parameters' and a list of files in a folder.

The screenshot shows the 'Skype for iPhone' app page on an iPhone. The top status bar indicates the time as 13:51 and signal strength. The app icon is a blue circle with a white 'S'. The title 'Skype for iPhone' and subtitle 'Social Networking' are displayed. It has a 4-star rating with 7 reviews. A 'GET' button is visible. Below the title, there are three screenshots of the app interface showing messaging and video calling. A sponsored banner for 'Verzenio' is also present.

Skype for iPhone
Social Networking
★ ★ ★ ★ 7
In-App Purchases

Click Here for More Information

Don't miss this content from our sponsor

AJ By Sponsor See More

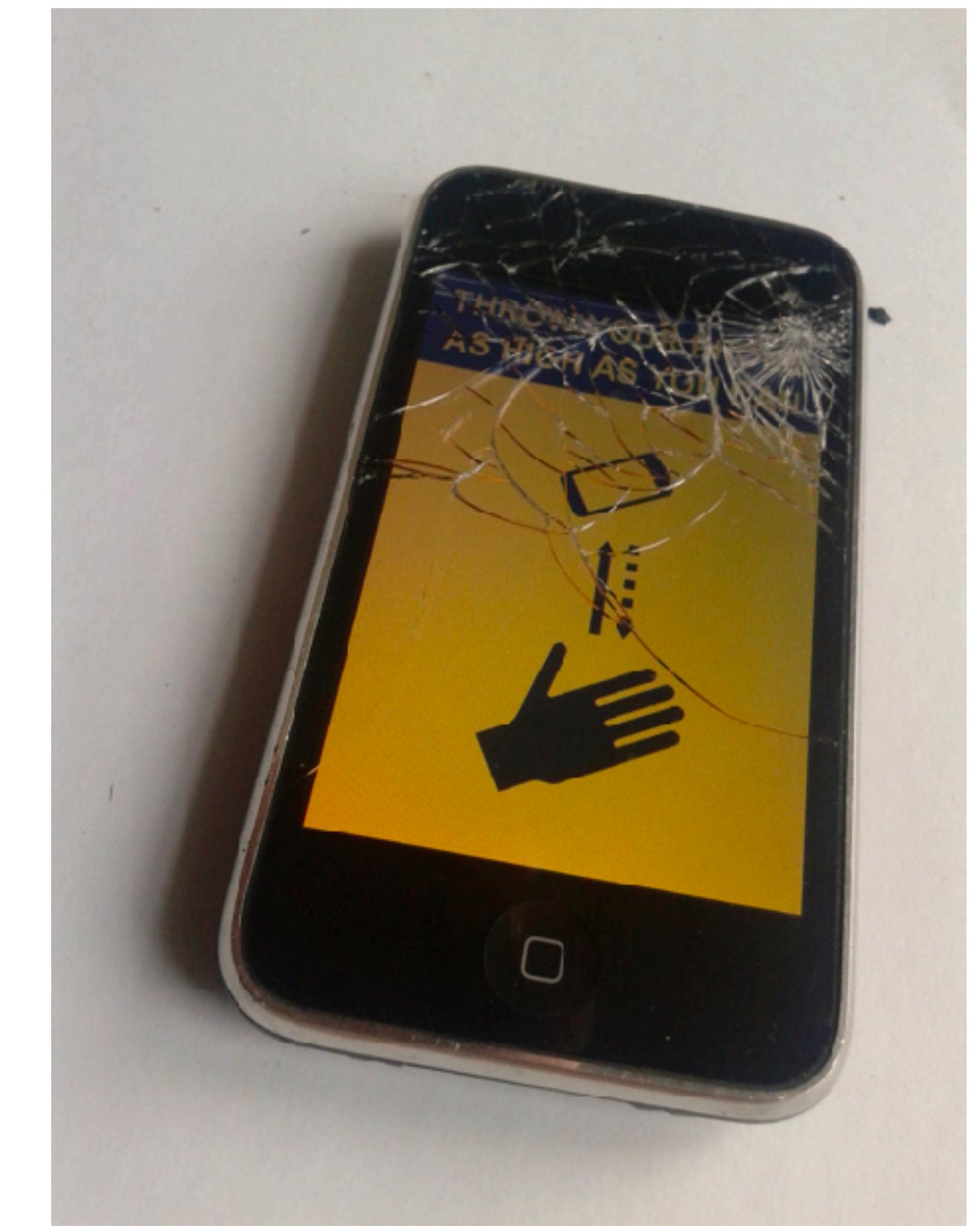
GroupMe
Social Networking
★ ★ ★ ★ 56
In-App Purchases

Make Connections That Count
Everything in One Place
Say More Than Words

Today Games Apps Updates Search

Technical idea #2

Distributing software through a centralized app store

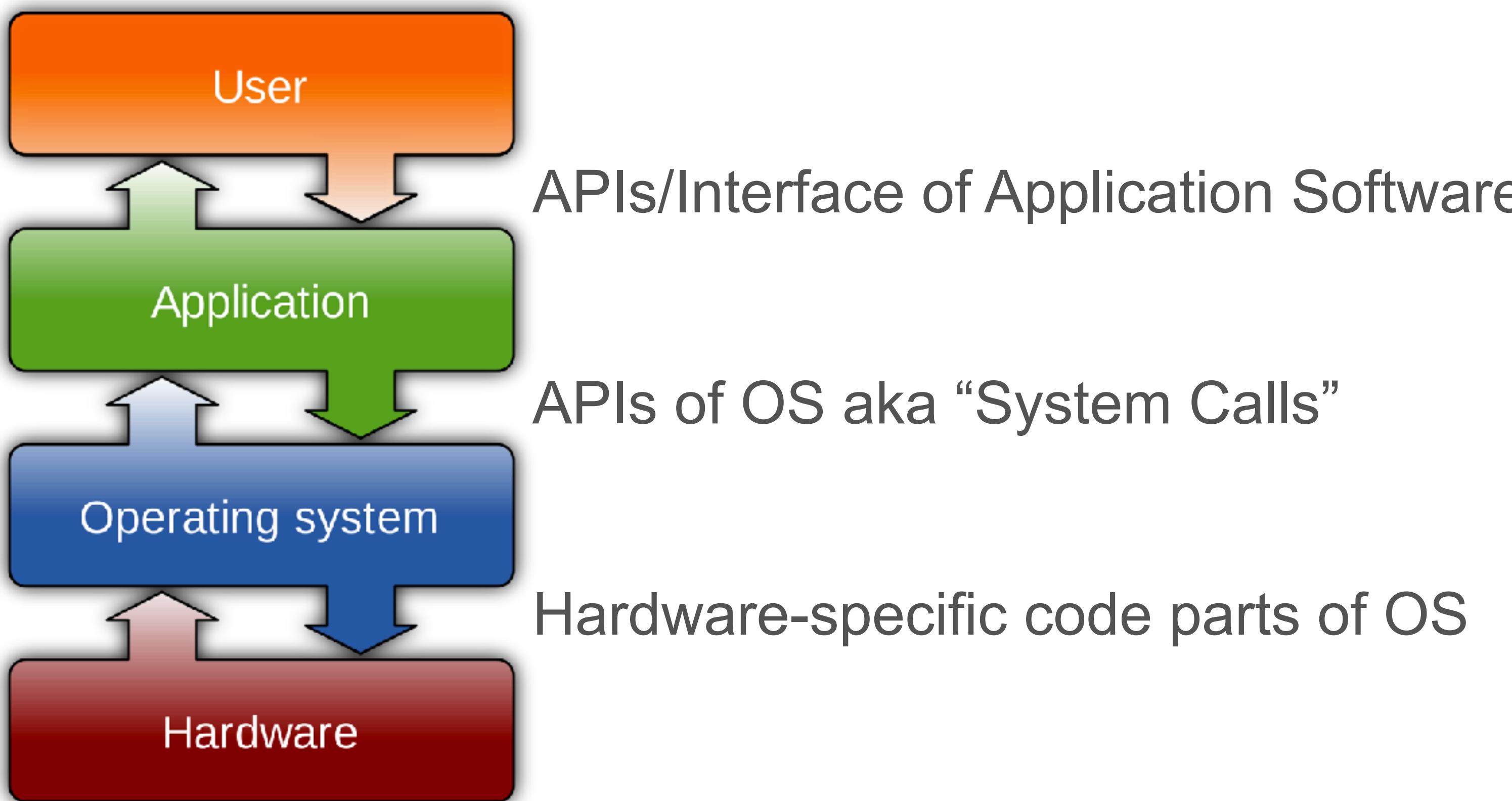


S.M.T.H.: Send Me to Heaven

Role of an OS in a Computer

- 2 key principles in OS (any system) design & impl.:
 - **Modularity:** Divide system into *functionally cohesive components* that each do their jobs well
 - Separation of concerns: https://en.wikipedia.org/wiki/Separation_of_concerns
 - Akin to executive-legislature-judiciary split
 - **Abstraction:** Layers of *functionalities* from low-level (close to hardware) to high level (close to user)
 - Principle of least privilege: https://en.wikipedia.org/wiki/Principle_of_least_privilege
 - Akin to local-city-county-state-federal levels?

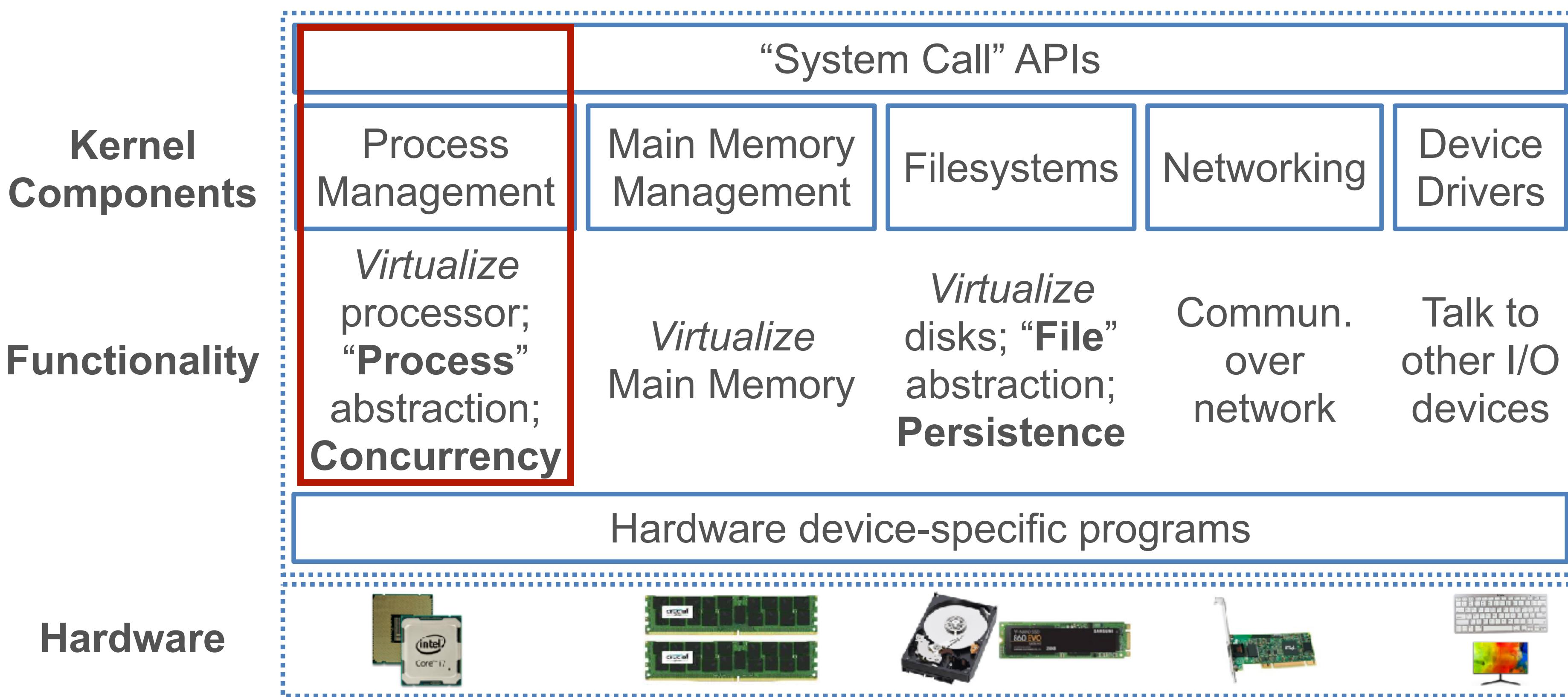
Abstraction (layers)



“Application Software” notion is now more complex due to multiple tiers of abstraction; “Platform Software” or “Software Framework” is a new tier between “Application” and OS

Modules

- **Kernel:** The core of an OS with modules to abstract the hardware and APIs for programs to use
- Auxiliary parts of OS include shell/terminal, file browser for usability, extra programs installed by I/O devices, etc.

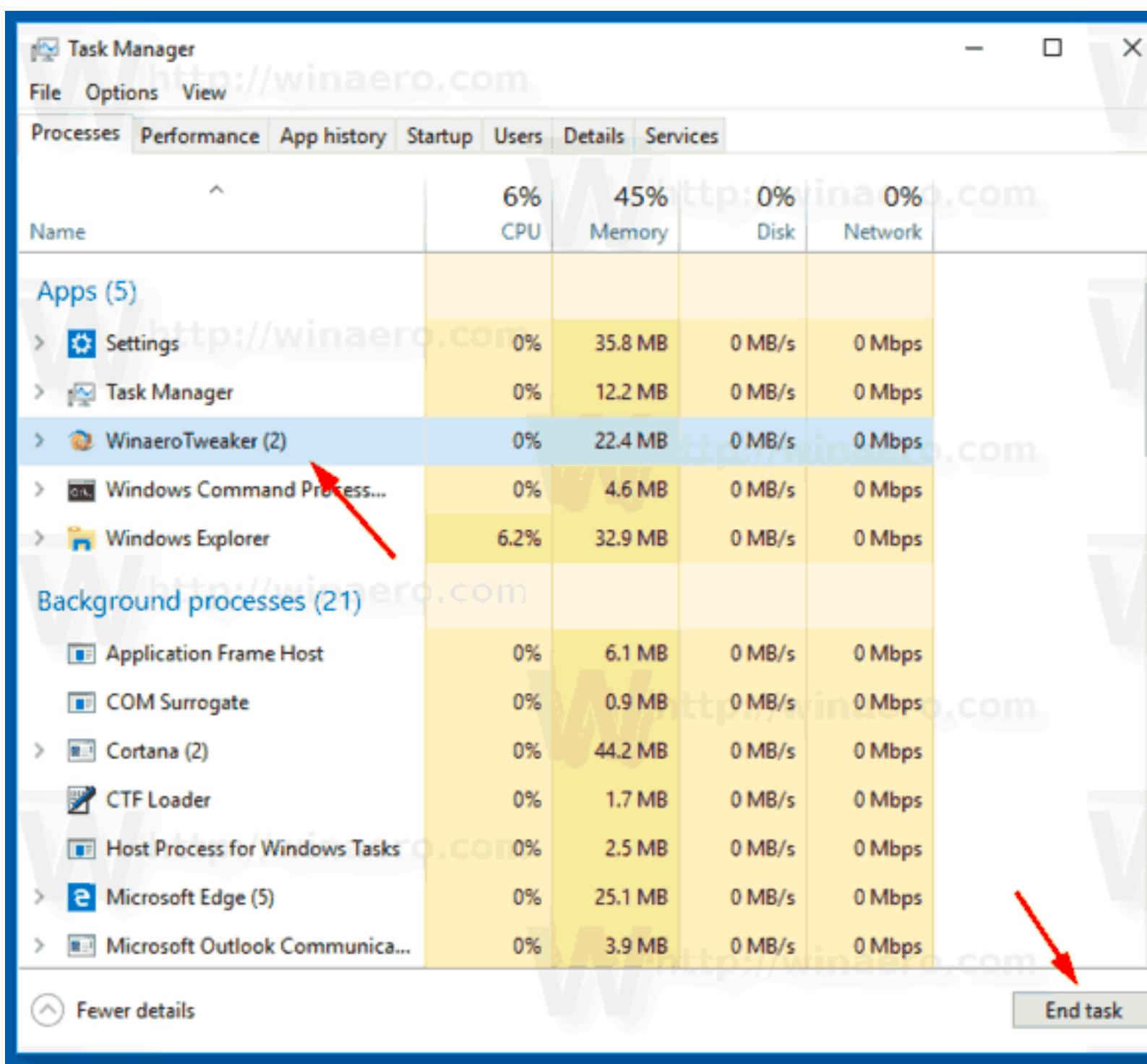


Processes - the central abstraction in OS

Definition: A **process** is an instance of a running program.

One of the most profound ideas in computer science

Not the same as “program” or “processor”



Why bother knowing process management in Data Science?

- A query is a program that becomes a process
- A data system typically *abstracts* away process management because user specifies the queries / processes in system's API



- But in the cloud era, things are up in the air! Will help to know a bit of how they handle data-intensive computations under the hood

Processes - the central abstraction in OS

Process provides each program with two key abstractions:

Logical control flow

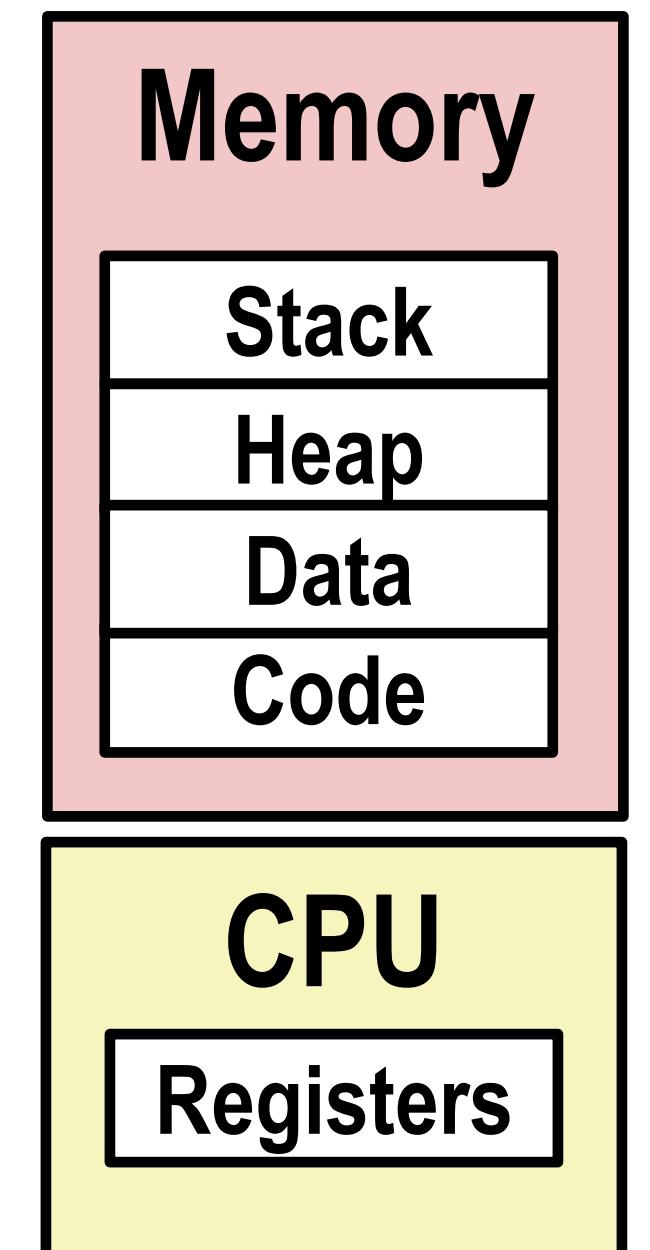
Each program seems to have exclusive use of the CPU

Provided by kernel mechanism called *context switching*

Private address space

Each program seems to have exclusive use of main memory.

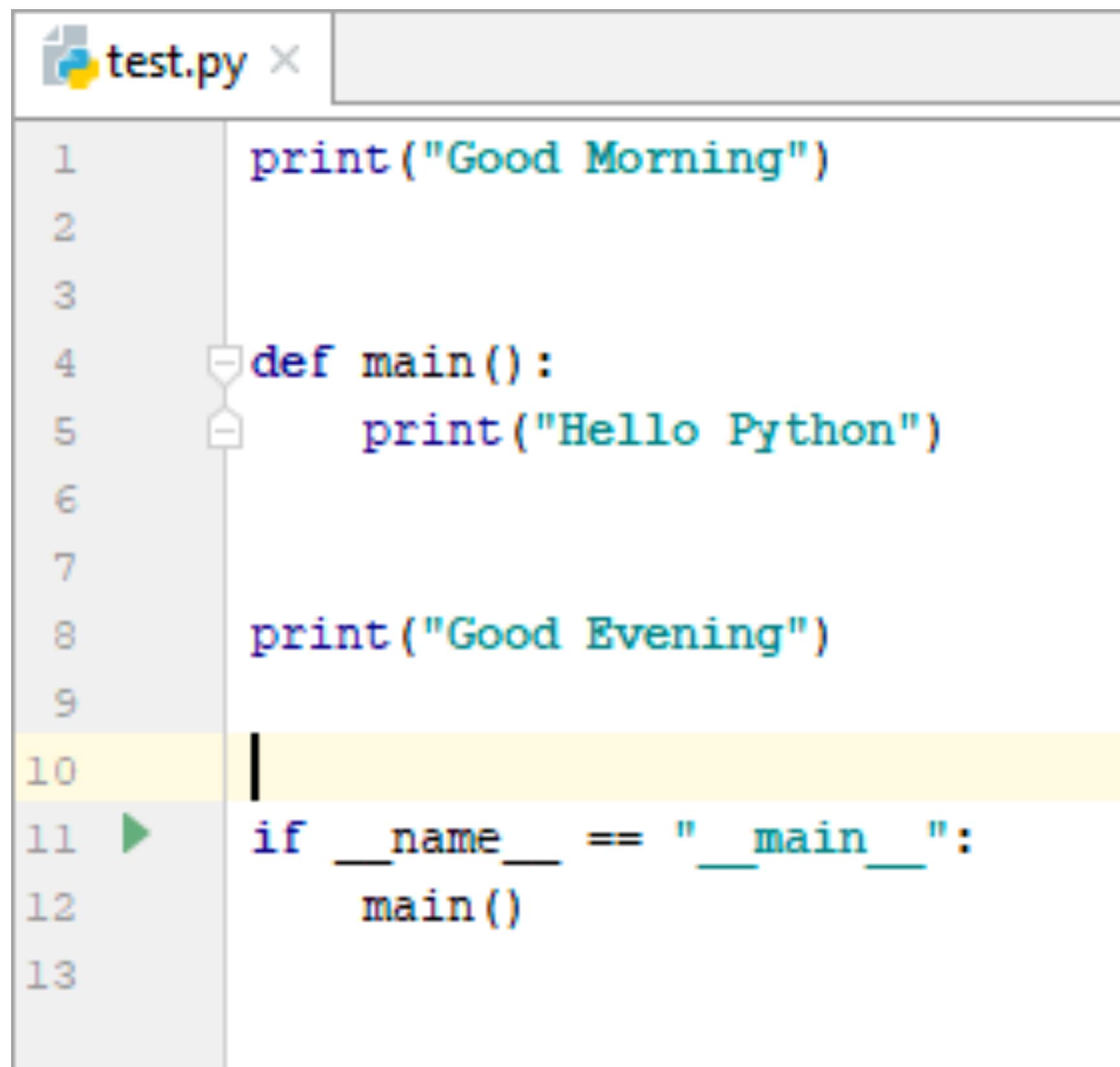
Provided by kernel mechanism called *virtual memory*



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()*

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
```

The Abstraction of a Process

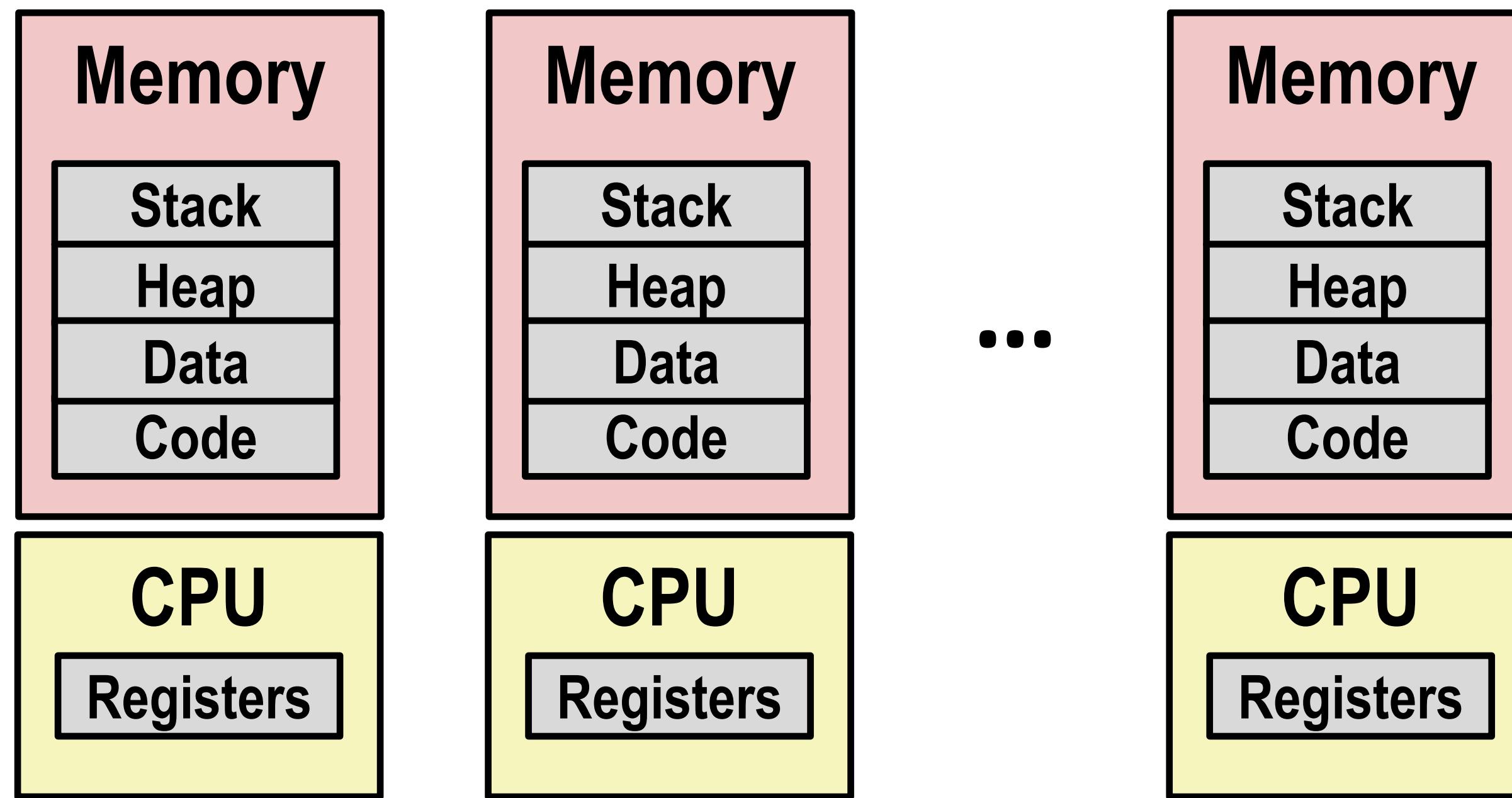
- ❖ High-level steps OS takes to get a process going:
 1. **Create** a process (get Process ID; add to Process List)
 2. Assign part of DRAM to process, aka its **Address Space**
 3. Load code and static data (if applicable) to that space
 4. Set up the inputs needed to run program's *main()*
 5. Update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

Applications for one or more users

Web browsers, email clients, editors, ...

Background tasks

Monitoring network & I/O devices

Multiprocessing Example

```
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.

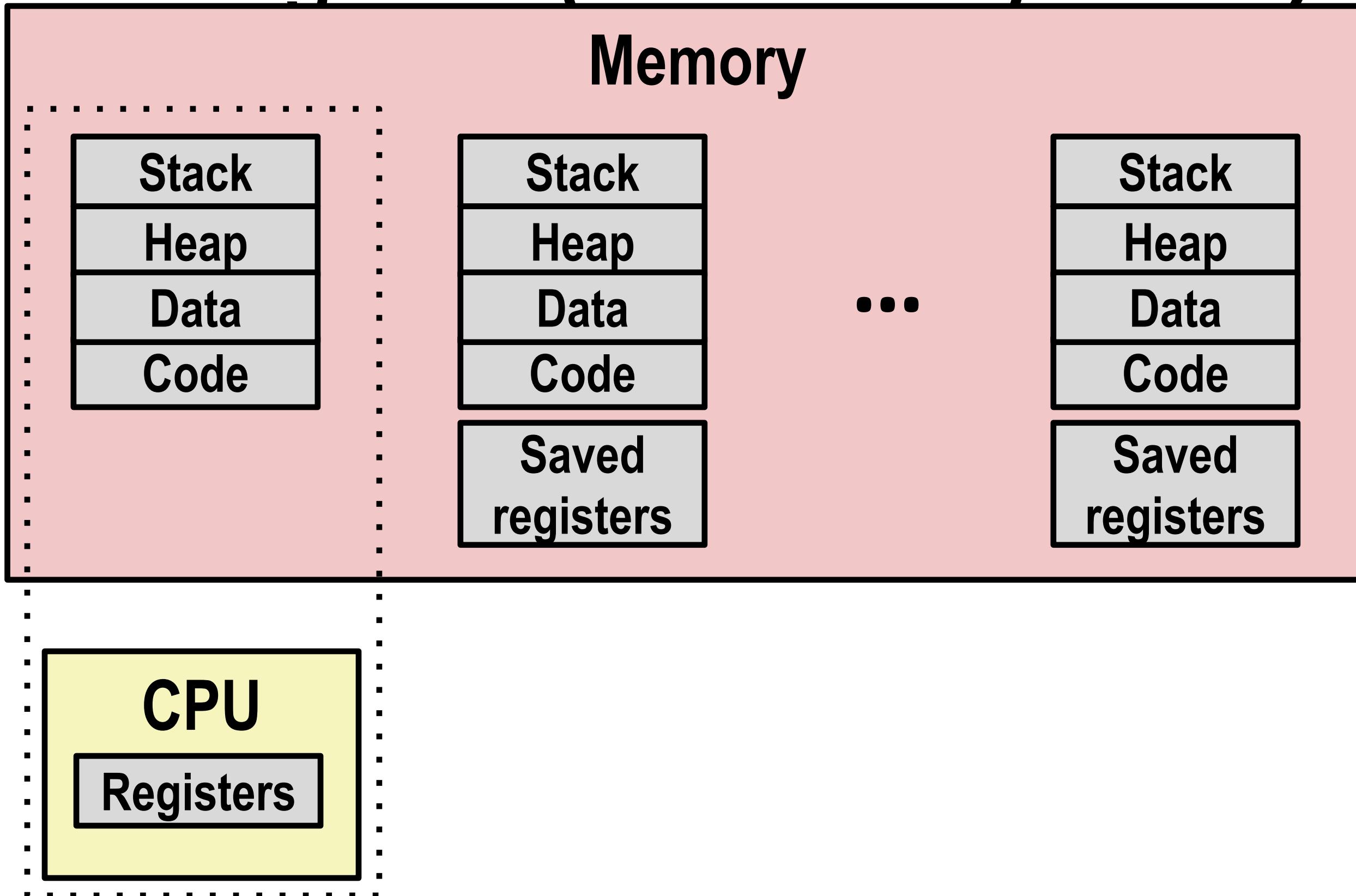
PID COMMAND %CPU TIME #TH #WQ #PORT #MREG RPRVT RSHRD RSIZE VPRVT VSIZE
99217- Microsoft Of 0.0 02:28.34 4 1 202 418 21M 24M 21M 66M 763M
99051 usbmuxd 0.0 00:04.10 3 1 47 66 436K 216K 480K 60M 2422M
99006 iTunesHelper 0.0 00:01.23 2 1 55 78 728K 3124K 1124K 43M 2429M
84286 bash 0.0 00:00.11 1 0 20 24 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 2332M
```

Running program "top" on Mac

System has 123 processes, 5 of which are active

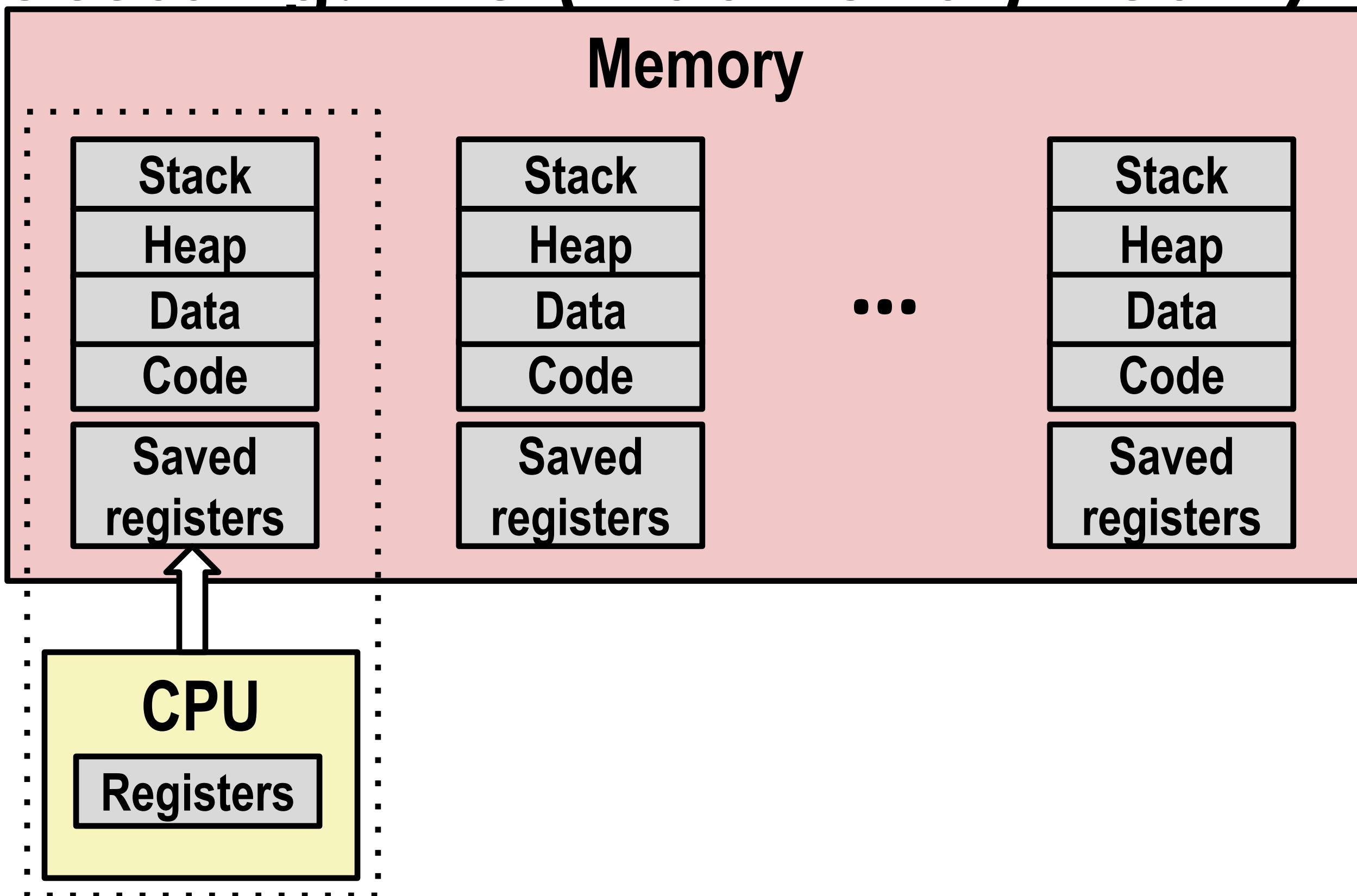
Identified by Process ID (PID)

Multiprocessing: The (Traditional) Reality



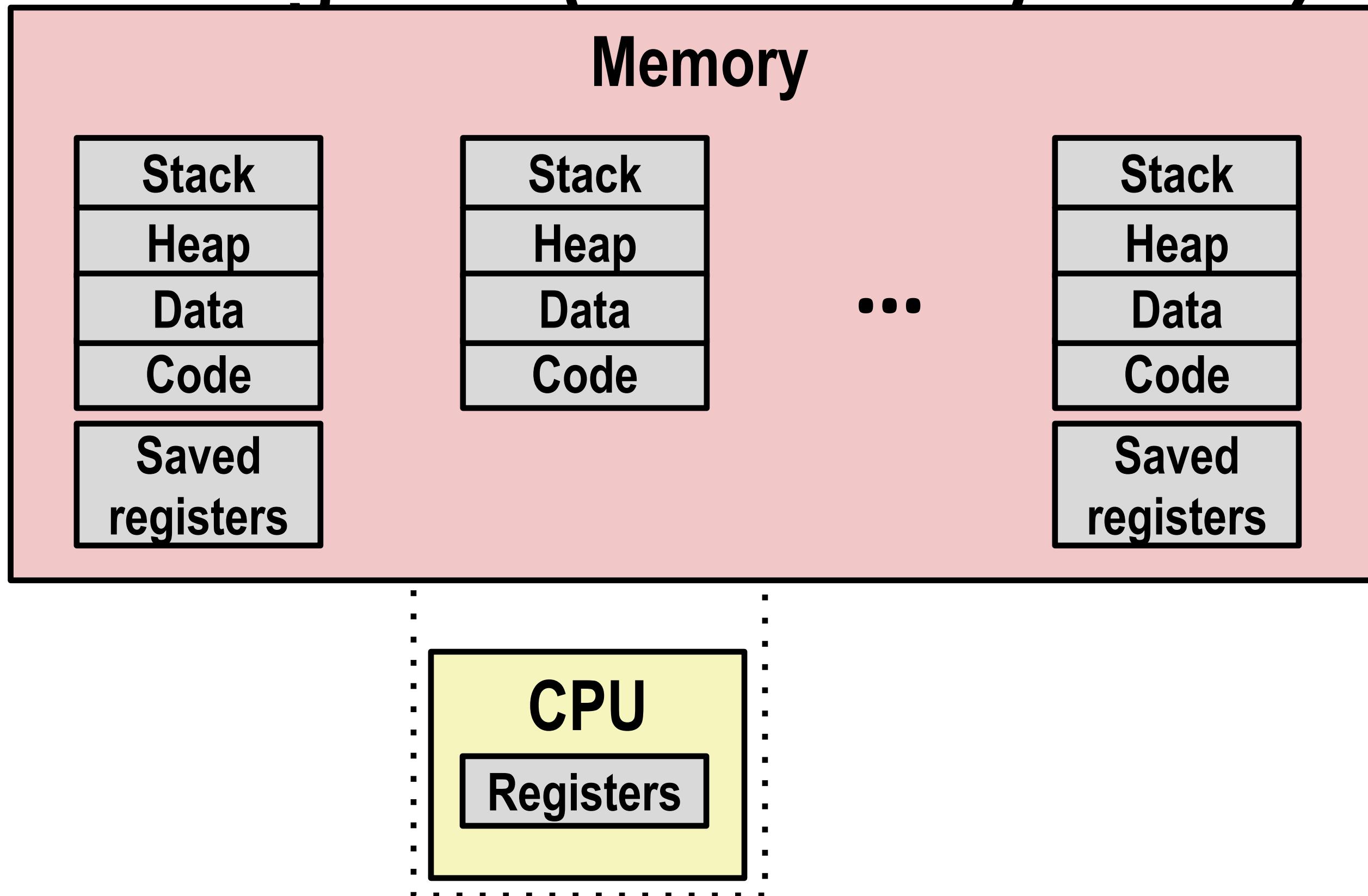
Single processor executes multiple processes concurrently
Process executions interleaved (multitasking)
Address spaces managed by virtual memory system (like last week)
Register values for nonexecuting processes saved in memory

Multiprocessing: The (Traditional) Reality



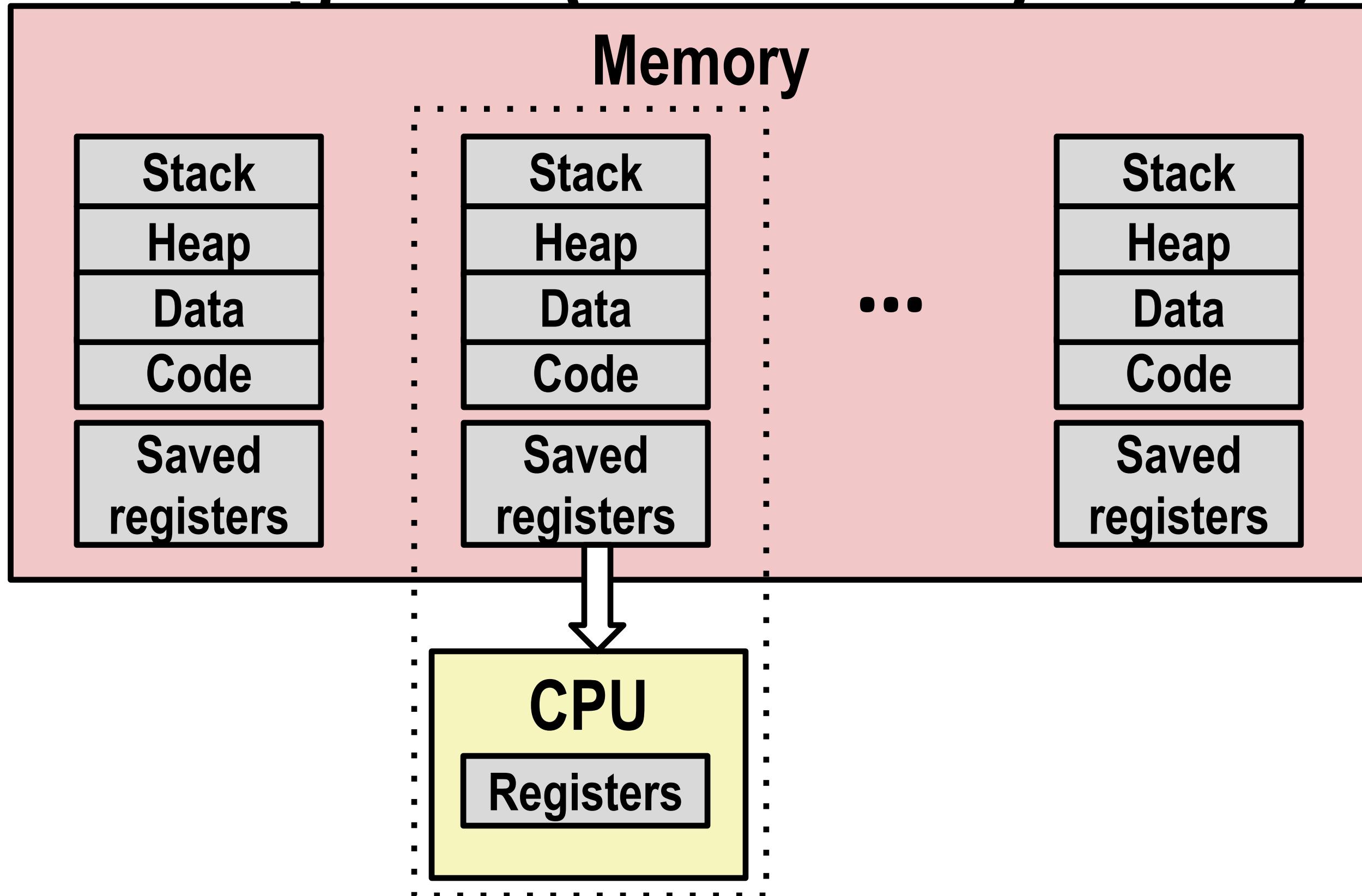
Save current registers in memory

Multiprocessing: The (Traditional) Reality



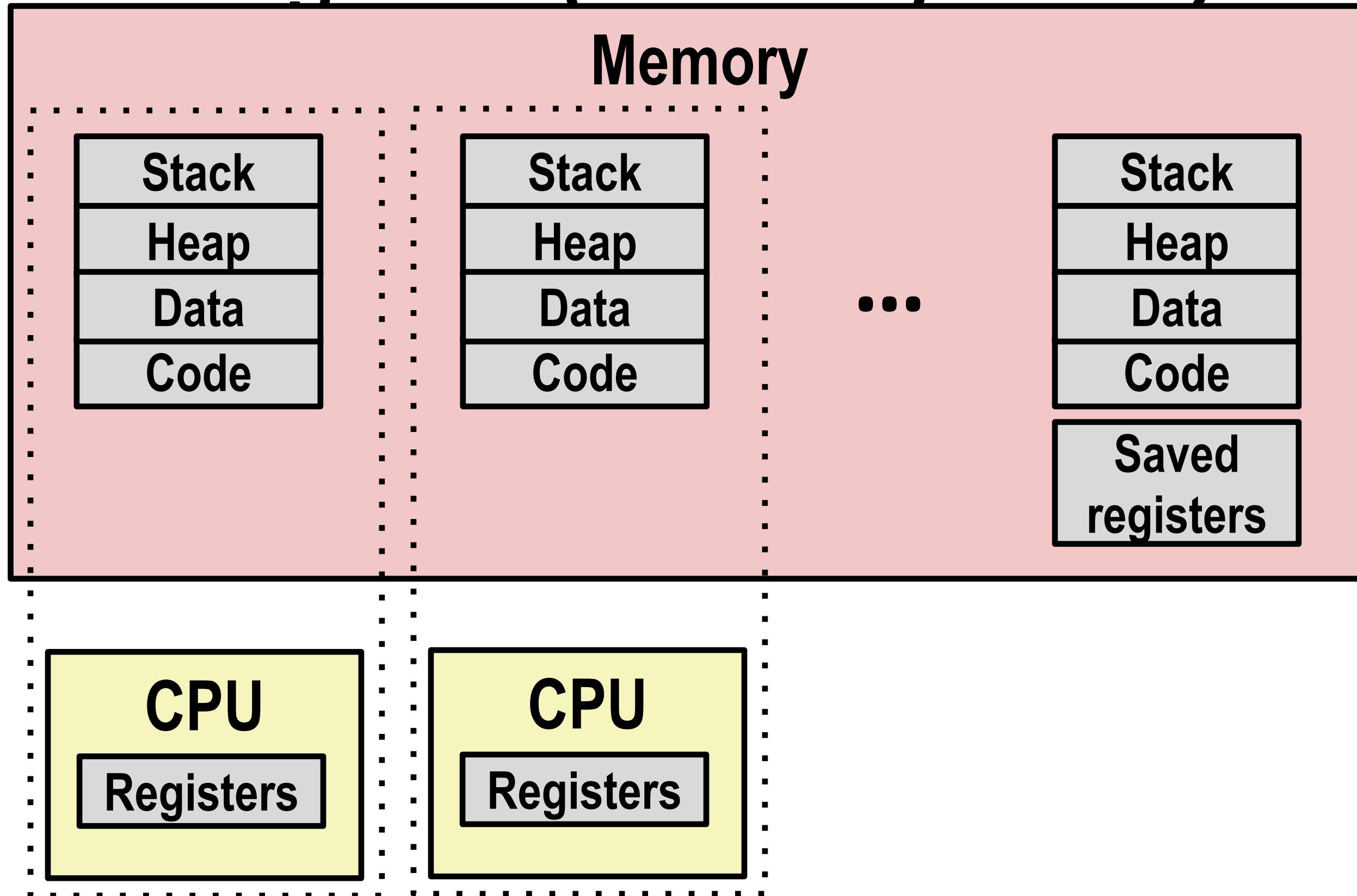
Schedule next process for execution

Multiprocessing: The (Traditional) Reality



Load saved registers and switch address space (context switch)

Multiprocessing: The (Modern) Reality



Multicore processors

Multiple CPUs on single chip

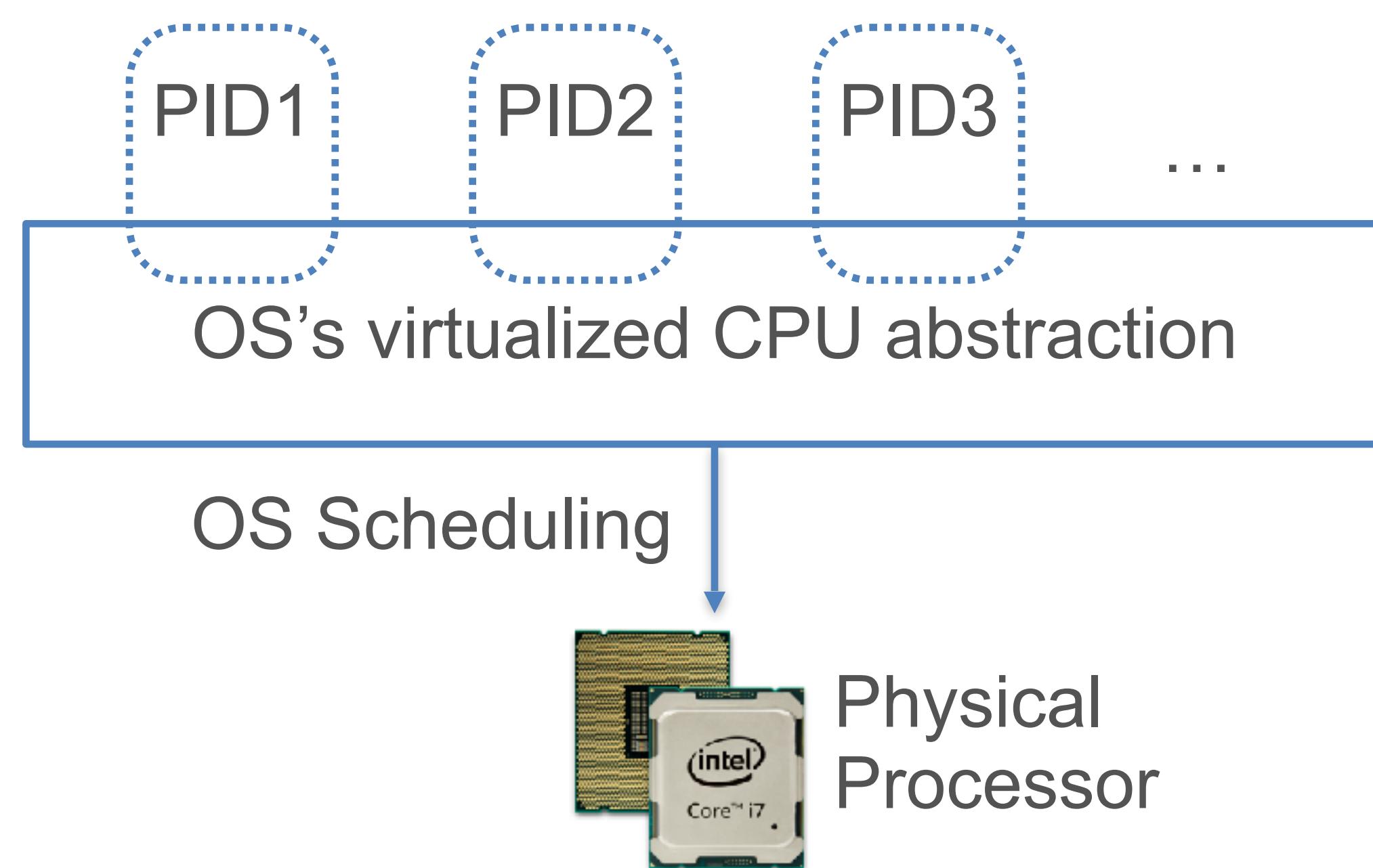
Share main memory (and some caches)

Each can execute a separate process

Scheduling of processors onto cores done by kernel

Inter-process communication

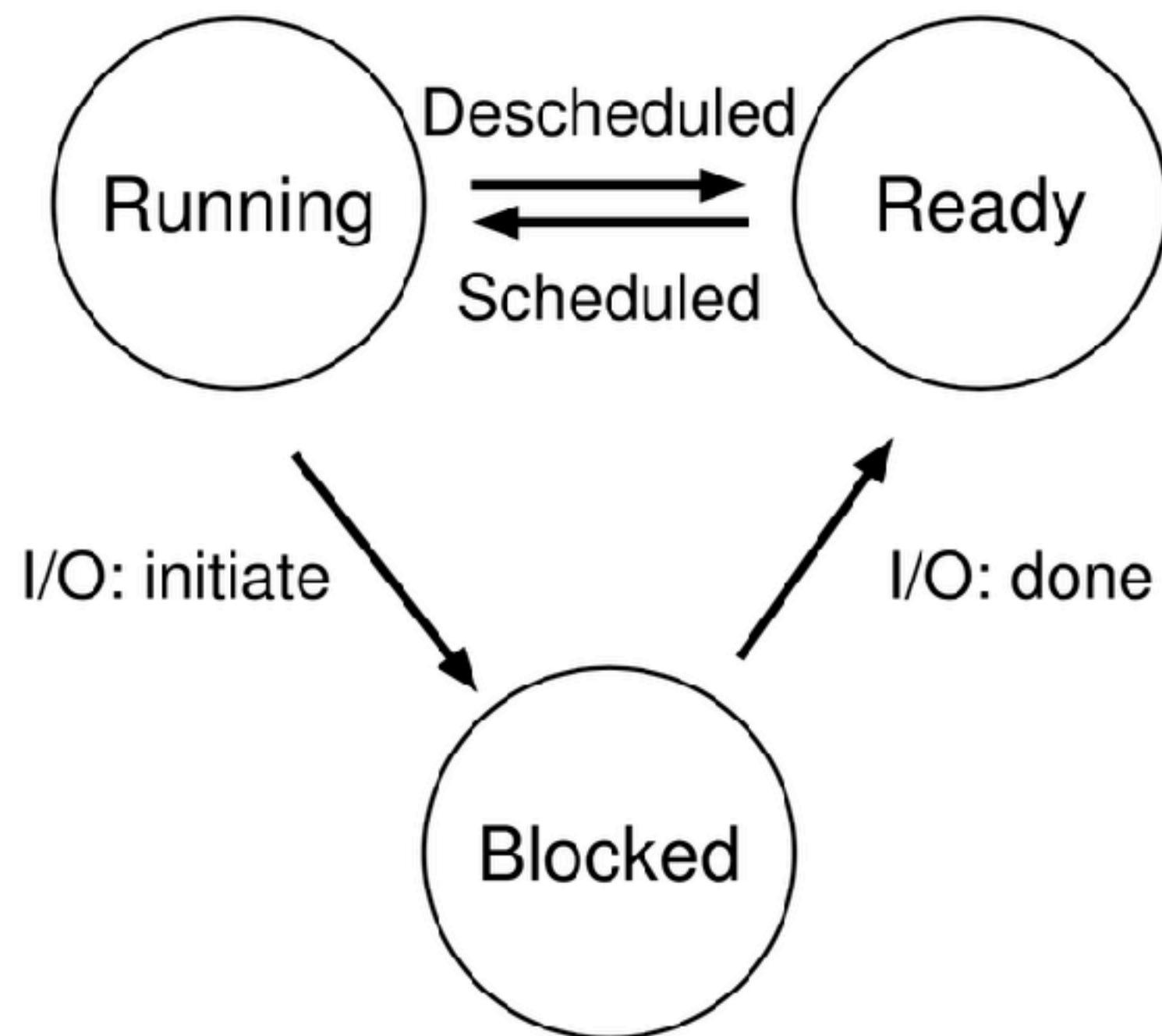
- ❖ Virtualization of processor enables process **isolation**, i.e., each process given an “illusion” that it alone runs



- ❖ 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!

Scheduling Policies/Algorithms

- **Schedule:** Record of what process runs on each CPU when
- Policy controls how OS time-shares CPUs among processes
- Key terms for a process (aka **job**):
 - **Arrival Time:** Time when process gets created
 - **Job Length:** Duration of time needed for process
 - **Start Time:** Time when process first starts on processor
 - **Completion Time:** Time when process finishes/killed
 - **Response Time** = Start Time – Arrival Time
 - **Turnaround** Time = Completion Time – Arrival Time
- **Workload:** Set of processes, arrival times, and job lengths that OS Scheduler has to handle

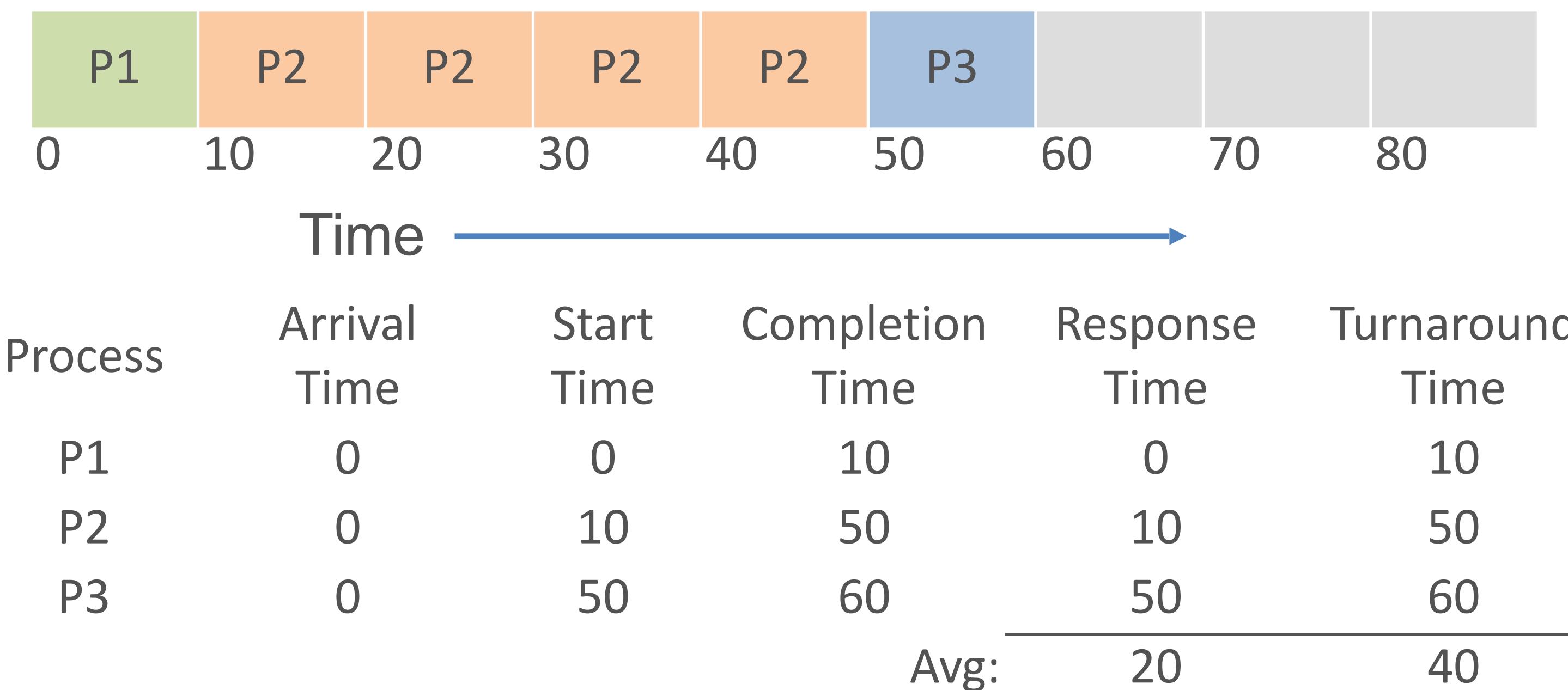
Scheduling Policies/Algorithms

- In general, not all Arrival Times and Job Lengths will be known beforehand. But **preemption** is possible.
- **Key Principle:** Inherent tension in scheduling between overall workload performance and allocation fairness
 - Performance metric is usually *Average Turnaround Time*
 - Many fairness metrics exist, e.g., Jain's fairness index
- 100s of scheduling policies studied! Well-known ones: FIFO, SJF, STCF, Round Robin, Random, etc.
 - Different criteria for ranking; preemptive vs not
 - Complex “multi-level feedback queue” schedulers
 - ML-based schedulers are “hot” nowadays!

Scheduling Policy: FIFO

- ❖ First-In-First-Out aka First-Come-First-Serve (FCFS)
- ❖ Ranking criterion: Arrival Time; no preemption allowed

Example: P1, P2, P3 of lengths 10,40,10 units arrive closely in that order

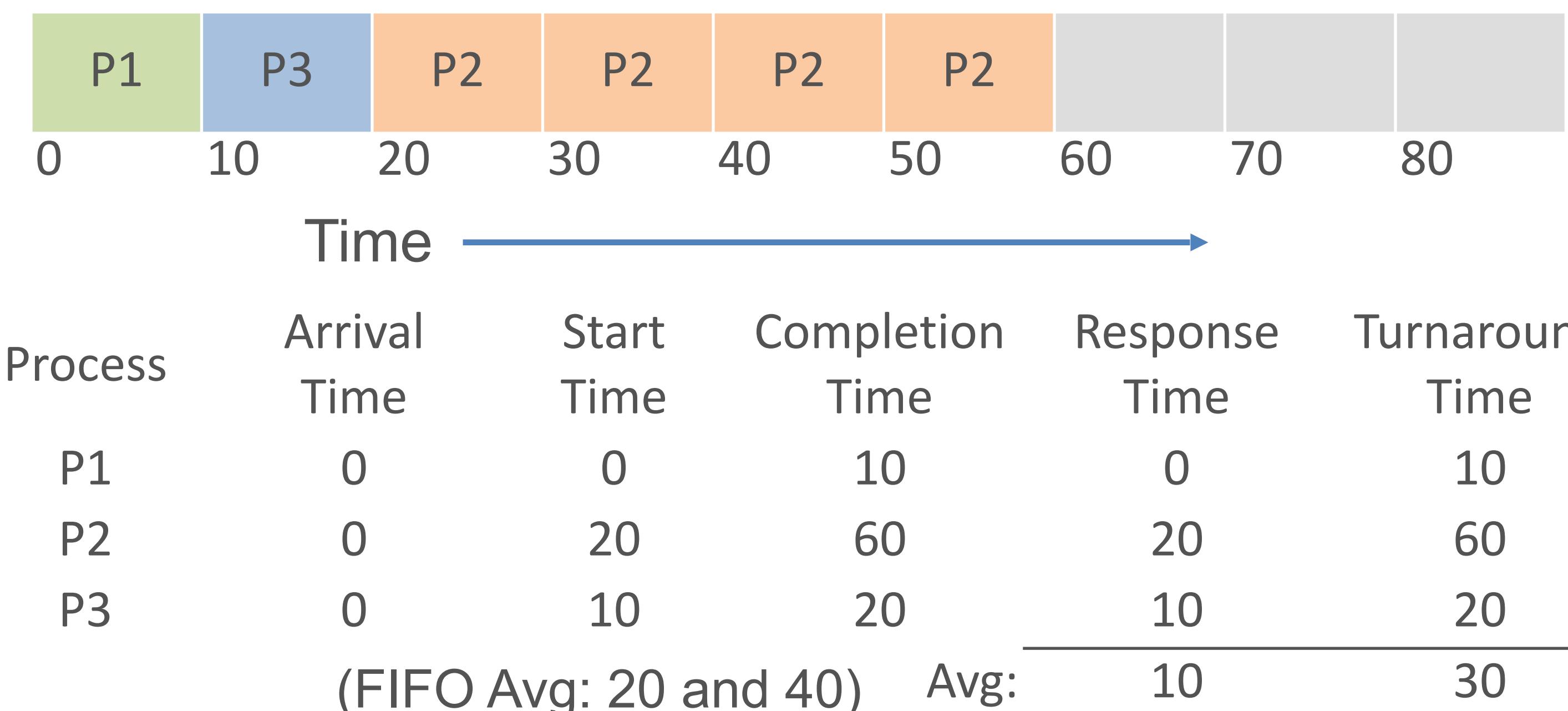


- ❖ Main con: Short jobs may wait a lot, aka “Convoy Effect”

Scheduling Policy: SJF

- ❖ Shortest Job (next) First
- ❖ Ranking criterion: Job Length; no preemption allowed

Example: P1, P2, P3 of lengths 10,40,10 units arrive closely in that order

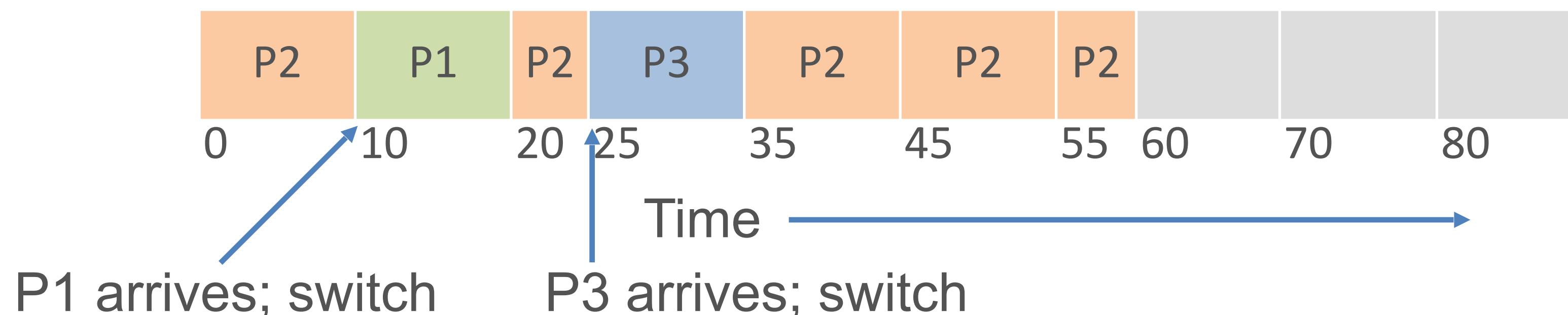


- ❖ Main con: Not all Job Lengths might be known beforehand
- ❖ Long processes may be held off indefinitely

Scheduling Policy: SCTF

- ❖ Shortest Completion Time First
- ❖ Jobs might not all arrive at same time; preemption possible

Example: P1, P2, P3 of lengths 10,40,10 units arrive at different times



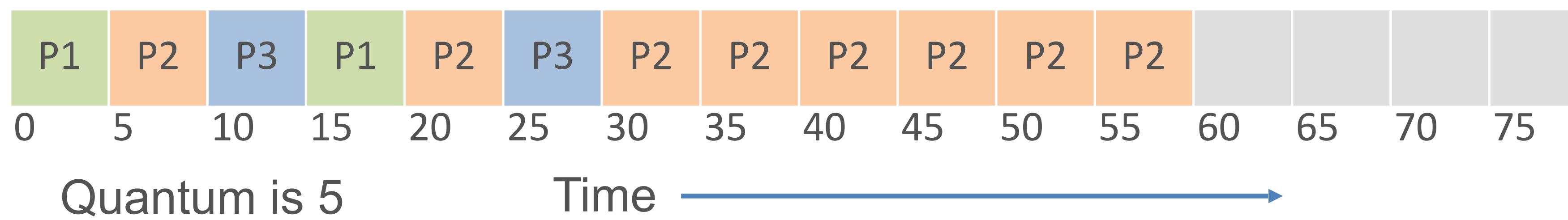
Process	Arrival Time	Start Time	Completion Time	Response Time	Turnaround Time
P1	10	10	20	0	10
P2	0	0	60	0	60
P3	25	25	35	0	10
(SJF Avg: 10 and 30)			Avg:	0	26.7

- ❖ Main con: Not all Job Lengths might be known beforehand
- ❖ Long processes may be held off indefinitely

Scheduling Policy: Round Robin

- ❖ RR does not need to know job lengths
- ❖ Fixed time *quantum* given to each job; cycle through jobs

Example: P1, P2, P3 of lengths 10,40,10 units arrive closely in that order



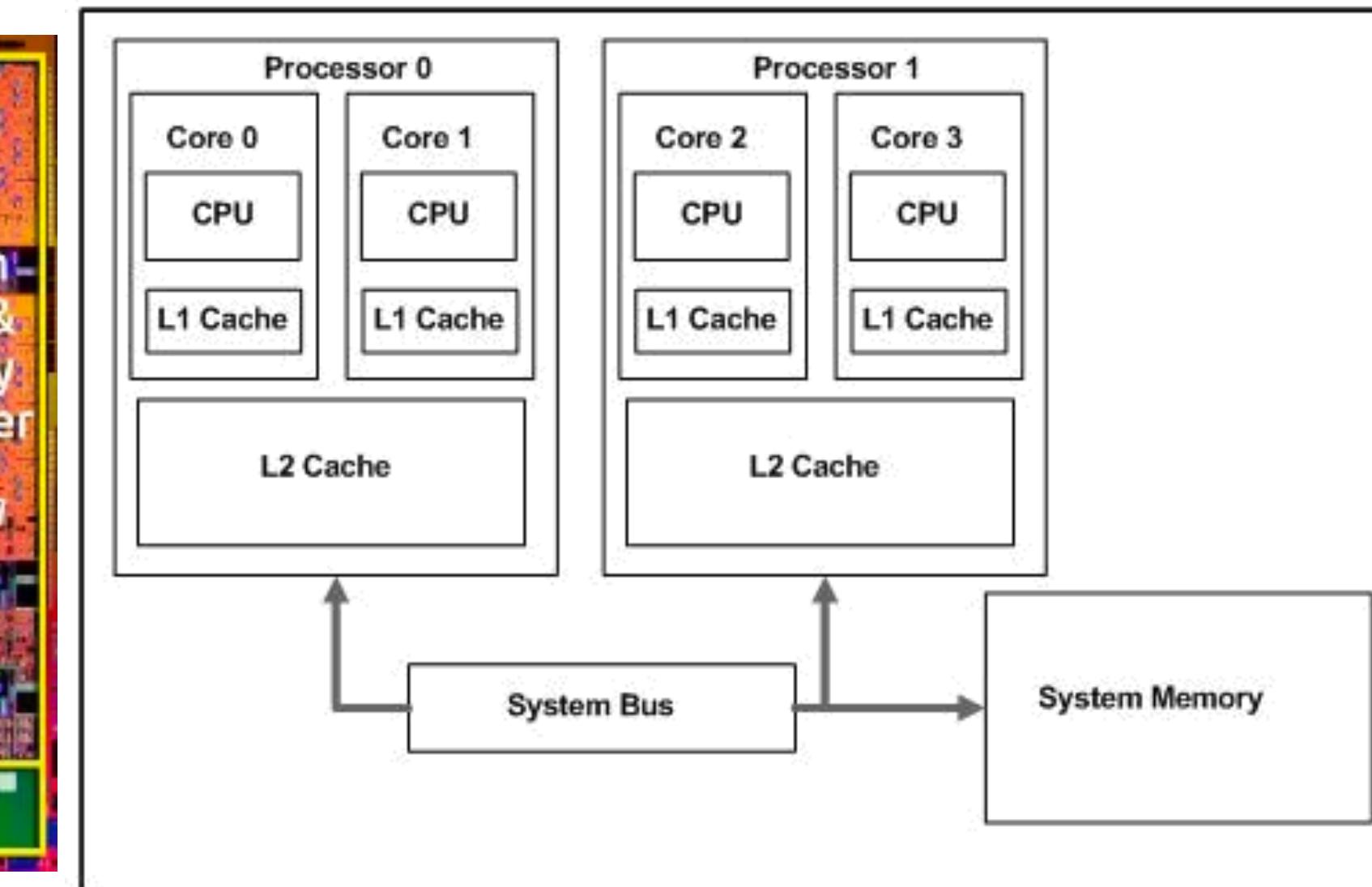
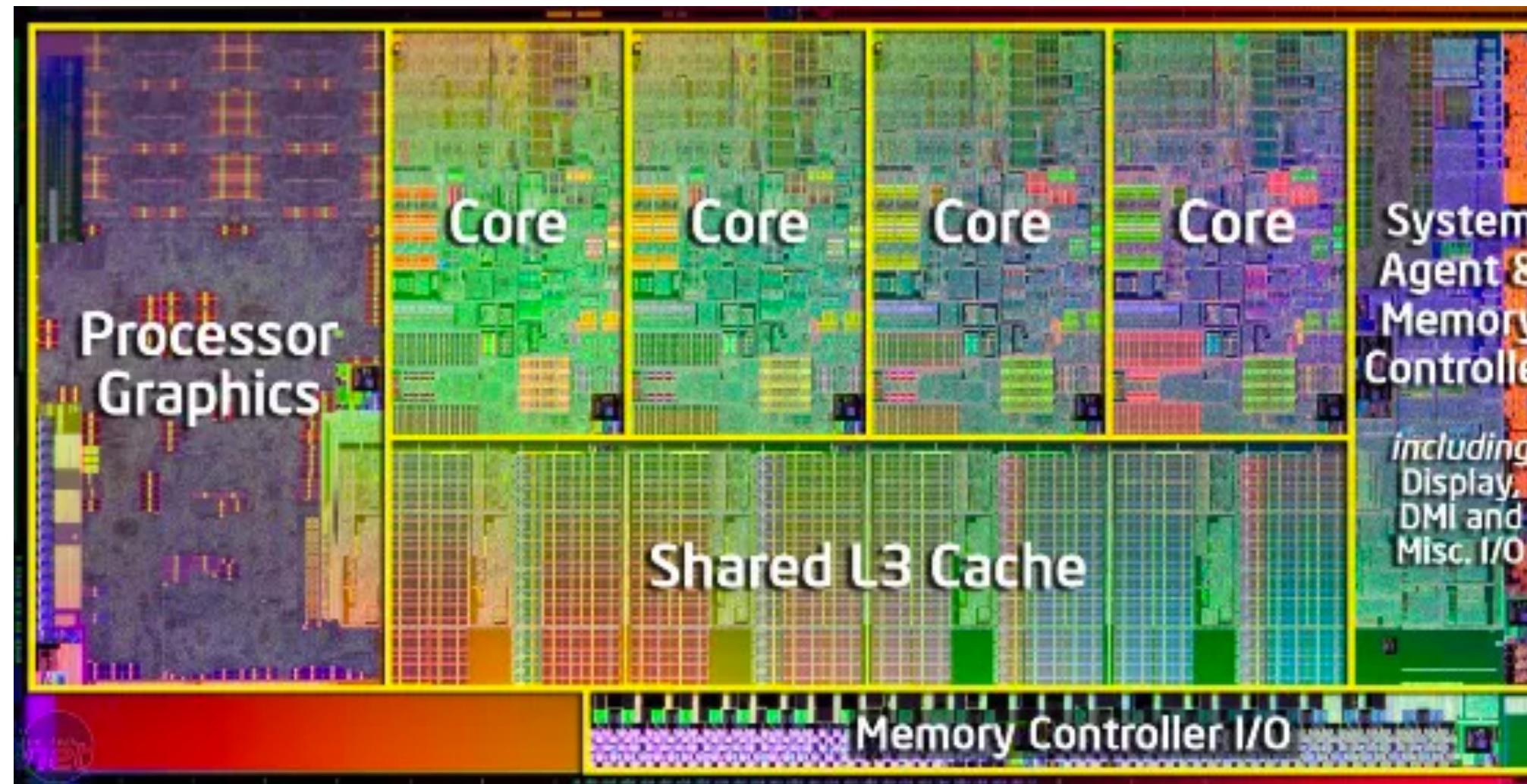
Process	Arrival Time	Start Time	Completion Time	Response Time	Turnaround Time
P1	0	0	20	0	20
P2	0	5	60	5	60
P3	0	10	30	10	30

(SJF Avg: 10 & 30; SCTF Avg: 0 & 26.7) Avg: 5 36.7

- ❖ RR is often very fair, but Avg Turnaround Time goes up!

Concurrency

- ❖ Modern computers often have multiple processors and multiple cores per processor
- ❖ **Concurrency:** Multiple processors/cores run different/same set of instructions simultaneously on different/shared data
- ❖ New levels of shared caches are added



Thread v.s. process

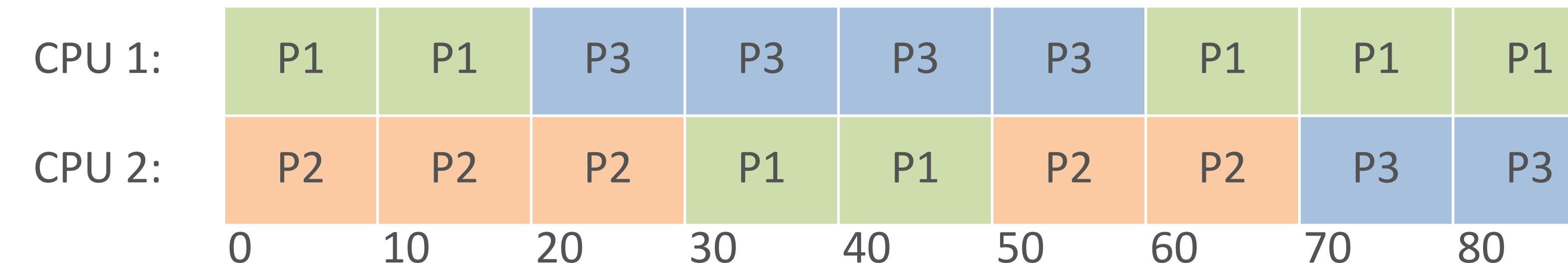
- ❖ **Multiprocessing:** Different processes run on different cores (or entire CPUs) simultaneously
- ❖ **Thread:** Generalization of OS's Process abstraction
 - ❖ A program *spawns* many threads; each run parts of the program's computations simultaneously
 - ❖ **Multithreading:** Same core used by many threads



- ❖ Issues in dealing with multithreaded programs that *write shared data*:
 - ❖ Cache coherence
 - ❖ Locking; deadlocks
 - ❖ Complex scheduling

Concurrency

- ❖ Scheduling for multiprocessing/multicore is more complex
- ❖ **Load Balancing:** Ensuring different cores/proc. are kept roughly equally busy, i.e., reduce **idle times**
- ❖ Multi-queue multiprocessor scheduling (MQMS) is common
 - ❖ Each proc./core has its own job queue
 - ❖ OS moves jobs across queues based on load
 - ❖ Example Gantt chart for MQMS:



Concurrency in Data Science

- Thankfully, most **data-intensive computations** in data science do not need concurrent writes on shared data!
 - Concurrent low-level ops abstracted away by libraries/APIs
 - **Partitioning / replication** of data simplifies concurrency
- Later topic (Parallelism Paradigms) will cover parallelism in depth:
 - Multi-core, multi-node, etc.
 - Task parallelism, Partitioned data parallelism, etc.

Peer Instruction Activity

(Switch slides)