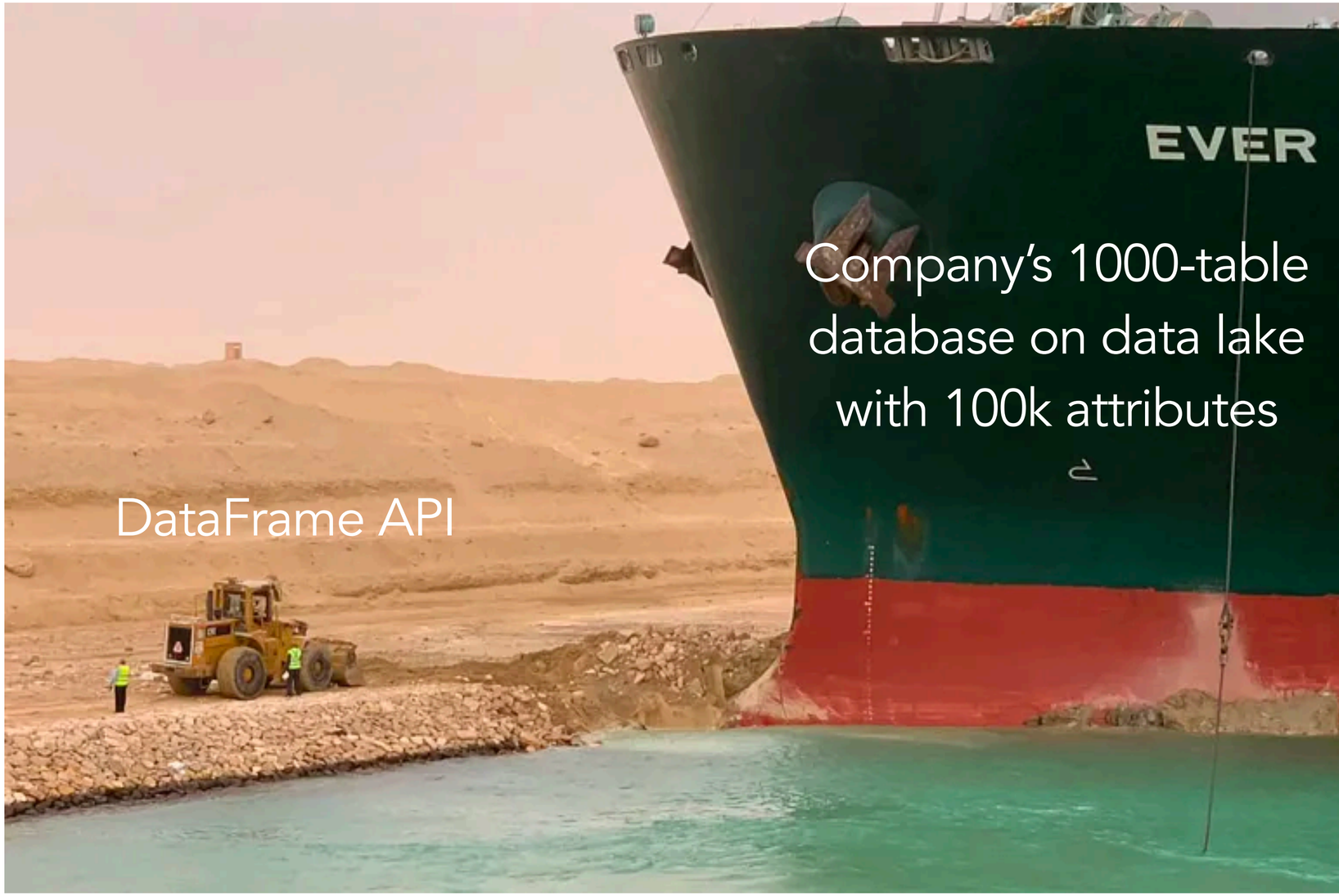
DSC 204a Scalable Data Systems

- Haojian Jin



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

Where are we in the class?

Foundations of Data Systems (2 weeks)

- Digital representation of Data → Computer Organization → Memory hierarchy → **Process** → Storage

Scaling Distributed Systems (3 weeks)

- Cloud → Distributed storage → Partition and replication (HDFS) → Distributed computation

Data Processing and Programming model (5 weeks)

- Data Models evolution → Data encoding evolution → → IO & Unix Pipes → Batch processing (MapReduce) → Stream processing (Spark)

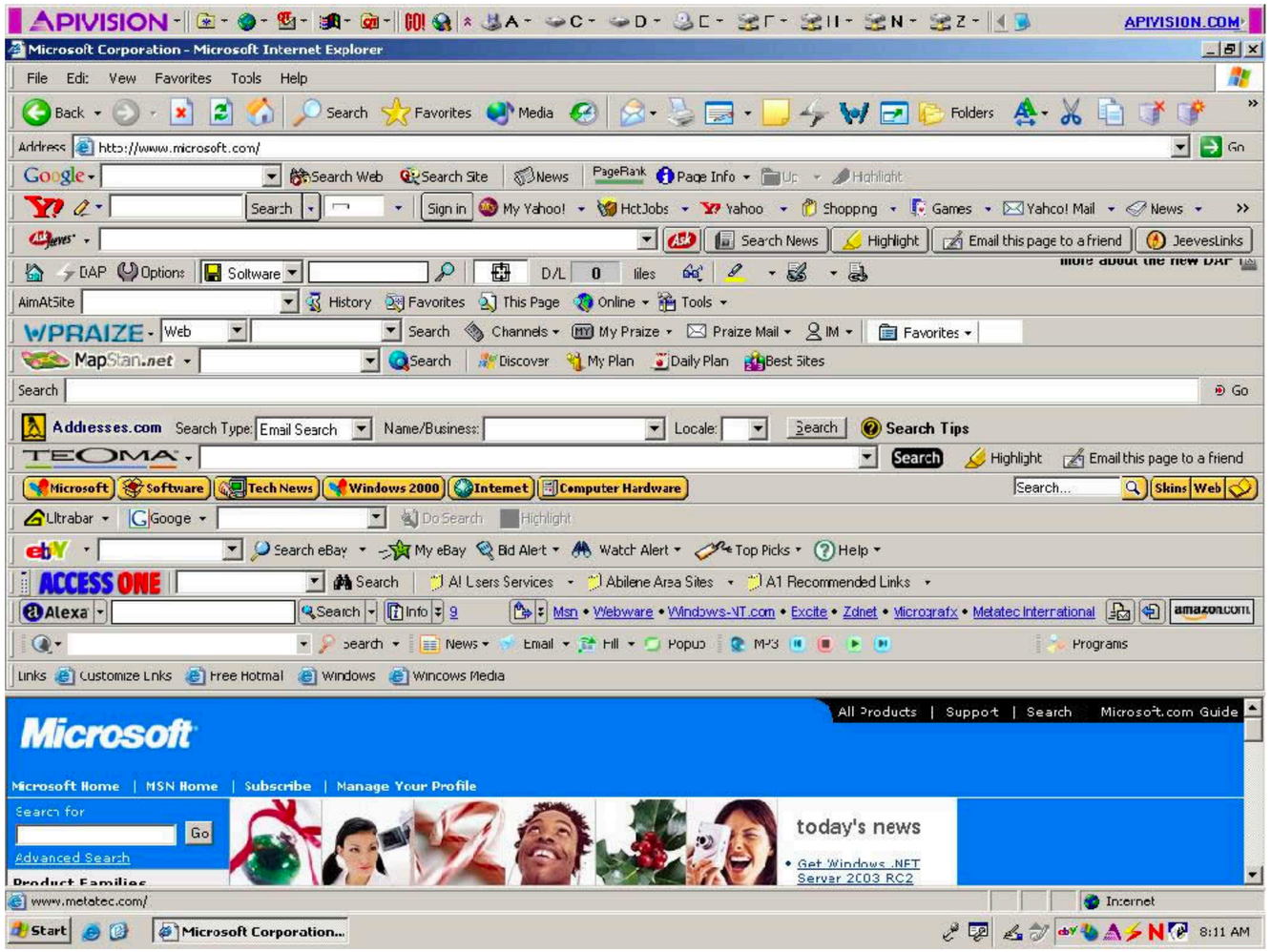
Today

Virtualization
Concurrency

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!


APIVISION
APIVISION.COM
Microsoft Corporation - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media Folders
Address http://www.microsoft.com/
Go
Google Search Web Search Site News PageRank Page Info Highlight
Search Sign in My Yahoo! HotJobs Yahoo Shopping Games Yahoo! Mail News
Search News Highlight Email this page to a friend JeevesLinks
DAP Options Software D/L 0 files
AimAtSite History Favorites This Page Online Tools
WPRAIZE Web Search Channels My Praize Praize Mail IM Favorites
MapStan.net Search Discover My Plan Daily Plan Best Sites
Search Go
Addresses.com Search Type: Email Search Name/Business: Locale: Search Search Tips
TEOMA Search Highlight Email this page to a friend
Microsoft Software Tech News Windows 2000 Internet Computer Hardware
Search... Skins Web
Ultrabar Google Do Search Highlight
eBay Search eBay My eBay Bid Alert Watch Alert Top Picks Help
ACCESS ONE Search
Alexa Search Info 9
Msn Webware Windows-NT.com Excite Zdnet Microsoft Metatec International
amazon.com
Programs
Links Customize Links Free Hotmail Windows Windows Media
Microsoft
All Products | Support | Search | Microsoft.com Guide
Microsoft Home | MSN Home | Subscribe | Manage Your Profile
Search for
Go
Advanced Search
today's news
Get Windows .NET Server 2003 RC2
www.metatec.com/
Internet
Start
Microsoft Corporation...
8:11 AM


KASPERSKY
Kaspersky Anti-Virus
The Smarter Way to Protect everything on your PC
SAVING THE WORLD FOR 20 YEARS

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."*

[1] Benedict Evans, App stores, trust and anti-trust

Technical idea #1

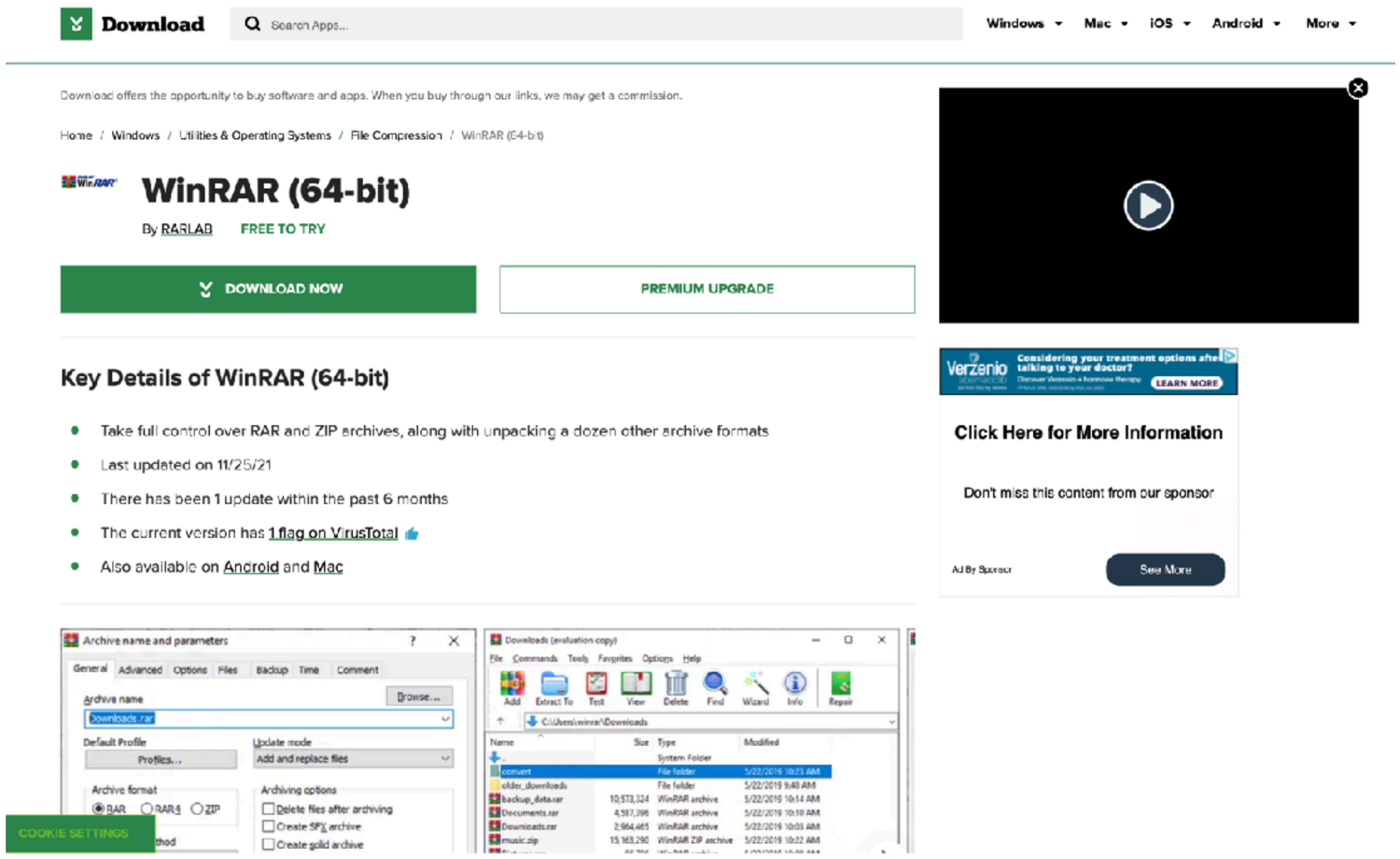
Putting apps in a sandbox

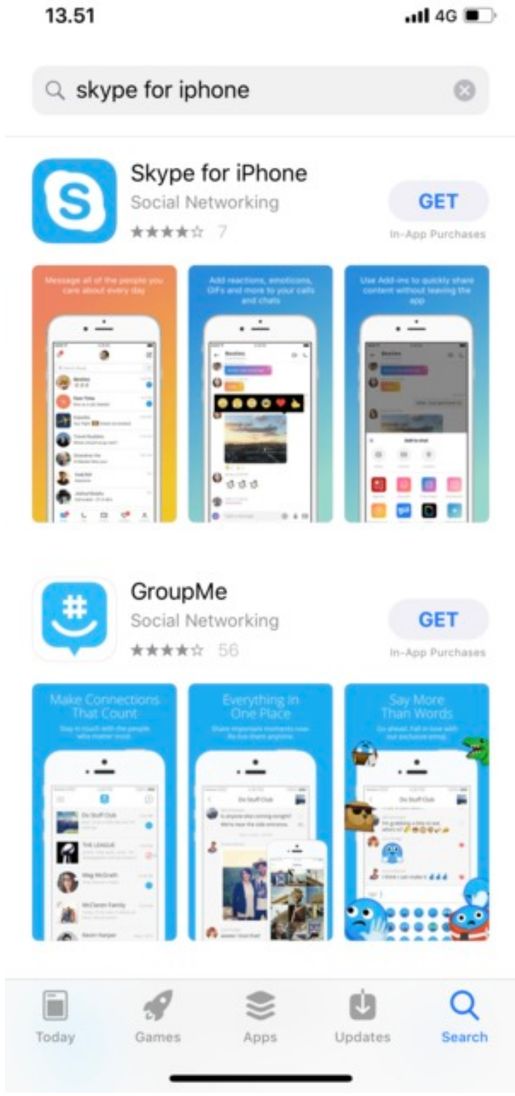
Apps can only do things that Apple allows and cannot ask (or persuade, or trick) the user for permission to do ‘dangerous’ things.

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

Technical idea #2

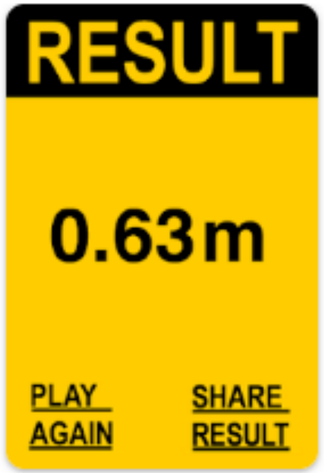
Distributing software through a centralized app store



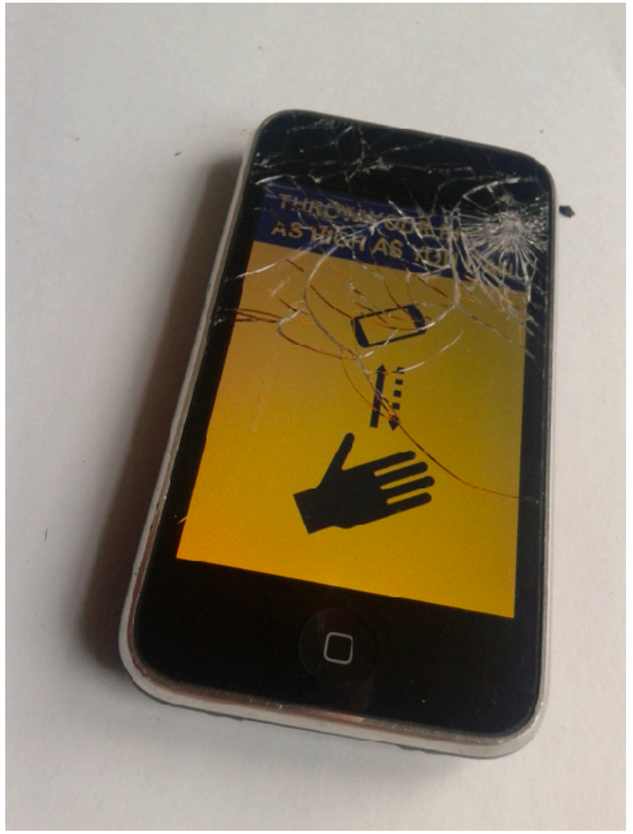


Technical idea #2

Distributing software through a centralized app store



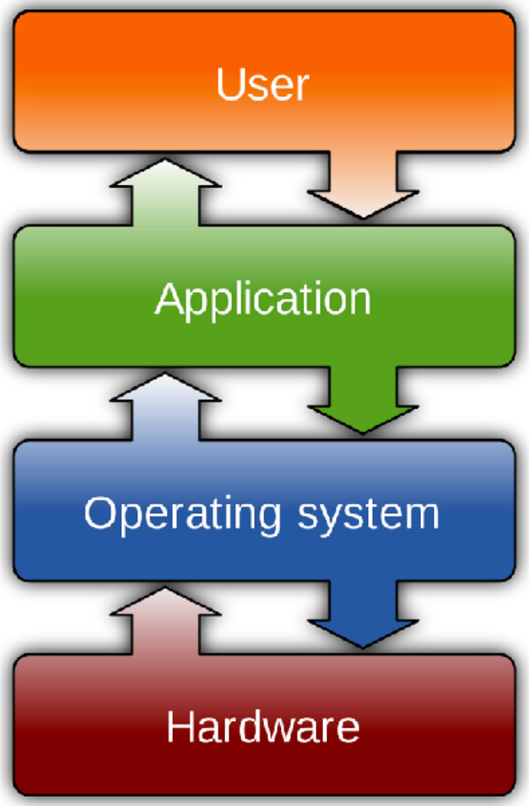
S.M.T.H.: Send Me to Heaven



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)



APIs/Interface of Application Software

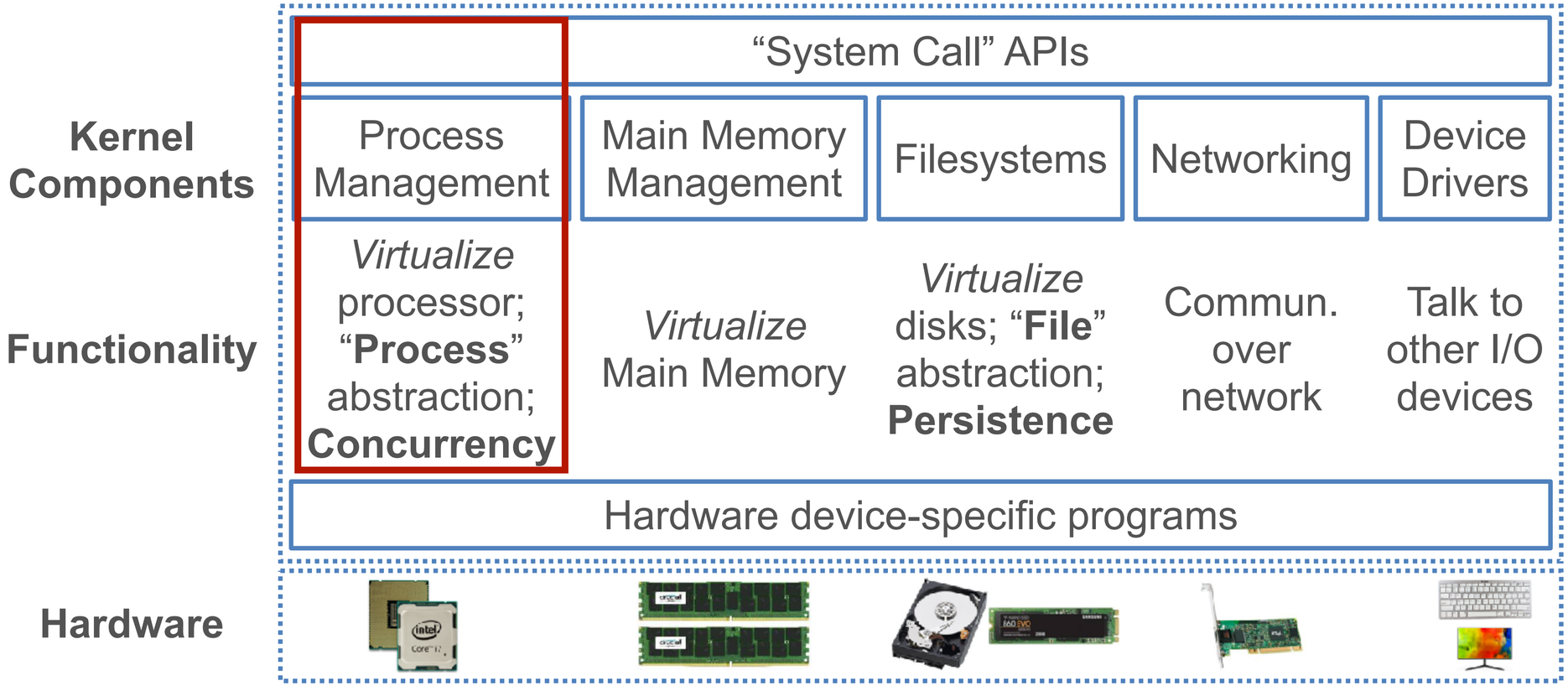
APIs of OS aka “System Calls”

Hardware-specific code parts of OS

“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.

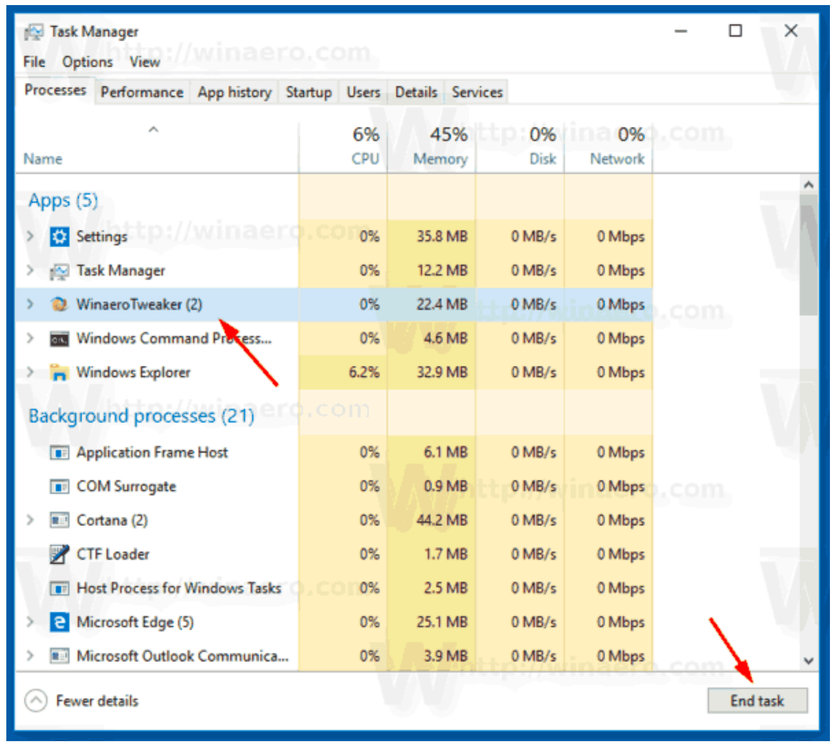


Processes - the central abstraction in OS

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

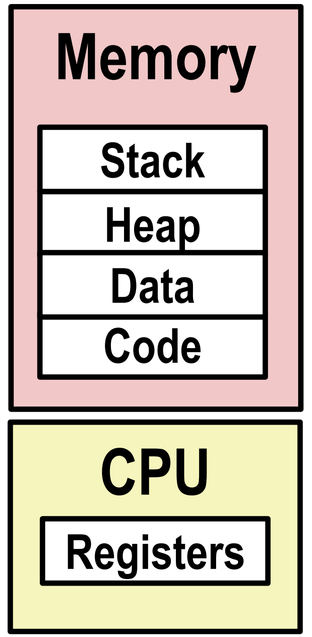
Each program seems to have exclusive use of the CPU

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:

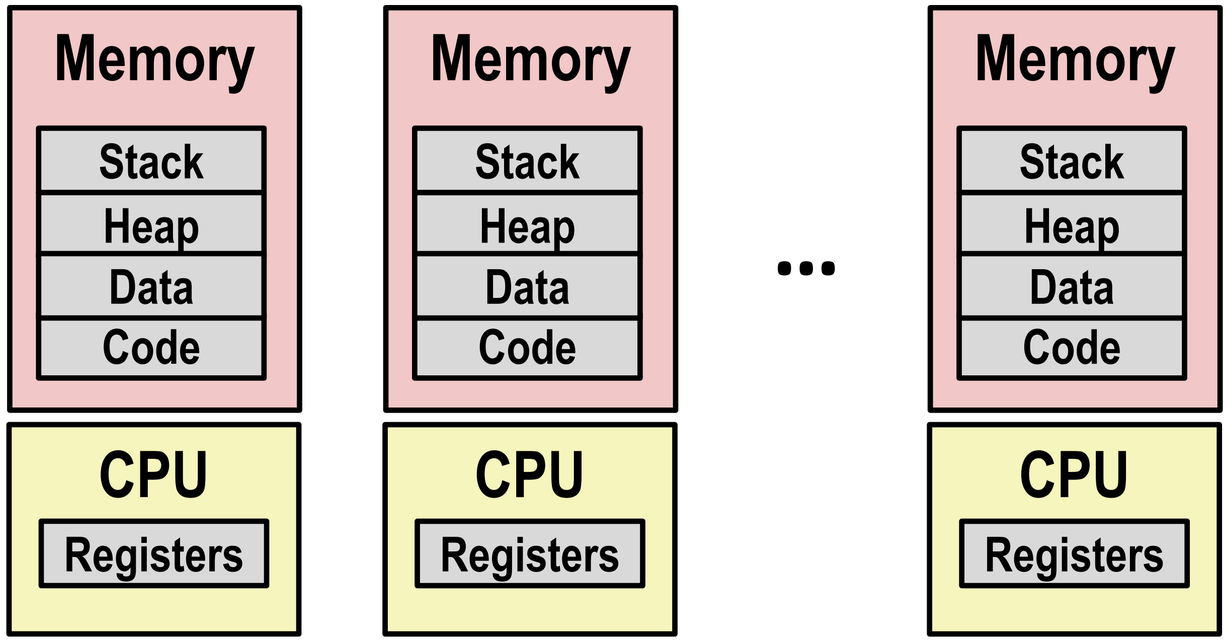
1. **Create** a process (get Process ID; add to Process List)
2. Assign part of DRAM to process, aka its **Address Space**
3. Load code and static data (if applicable) to that space
4. Set up the inputs needed to run program's *main()*
5. Update process' **State** to *Ready*
6. When process is **scheduled**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

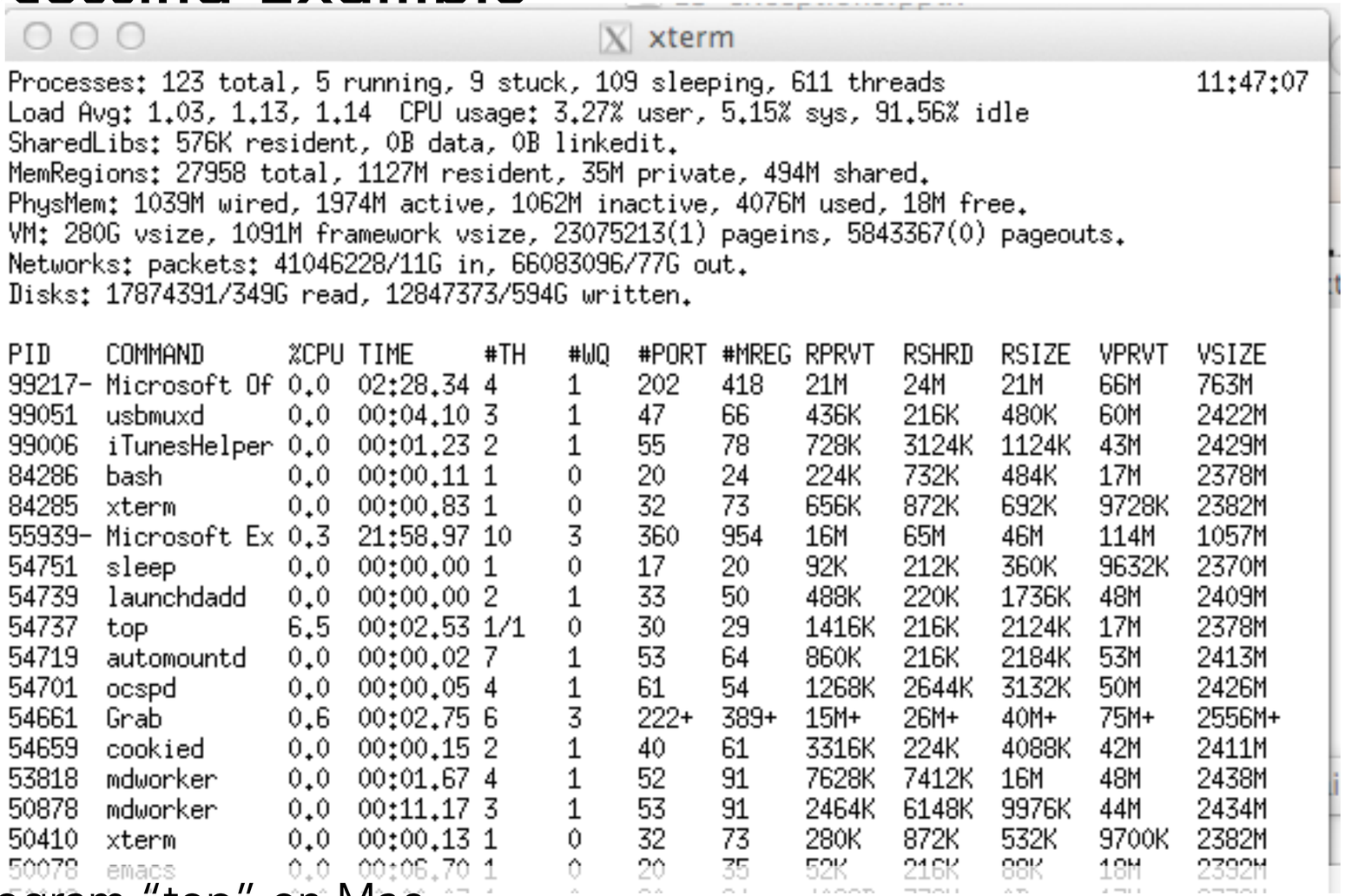
Multiprocessing: The Illusion



Computer runs many processes simultaneously

- Applications for one or more users
  - Web browsers, email clients, editors, …
- Background tasks
  - Monitoring network & I/O devices

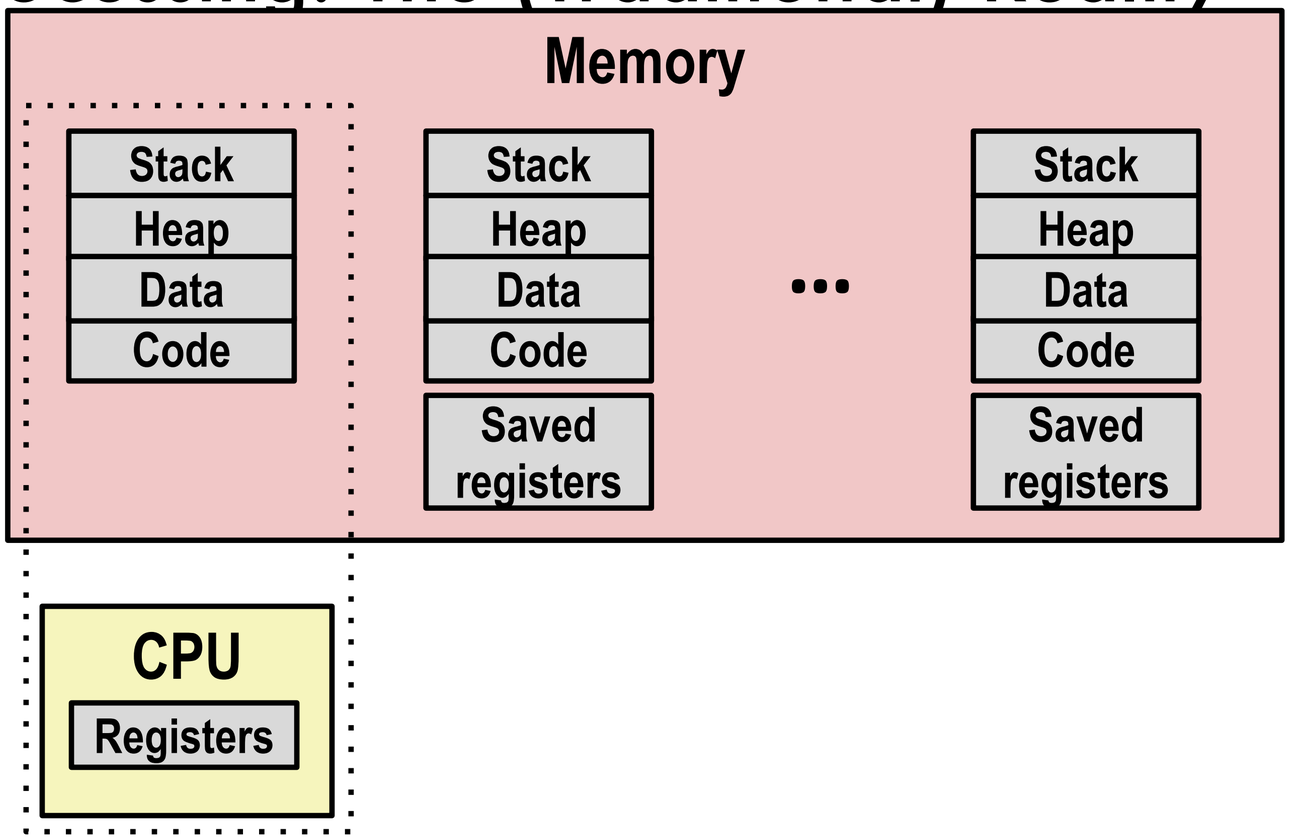
Multiprocessing Example



Running program "top" on Mac

- System has 123 processes, 5 of which are active
- Identified by Process ID (PID)

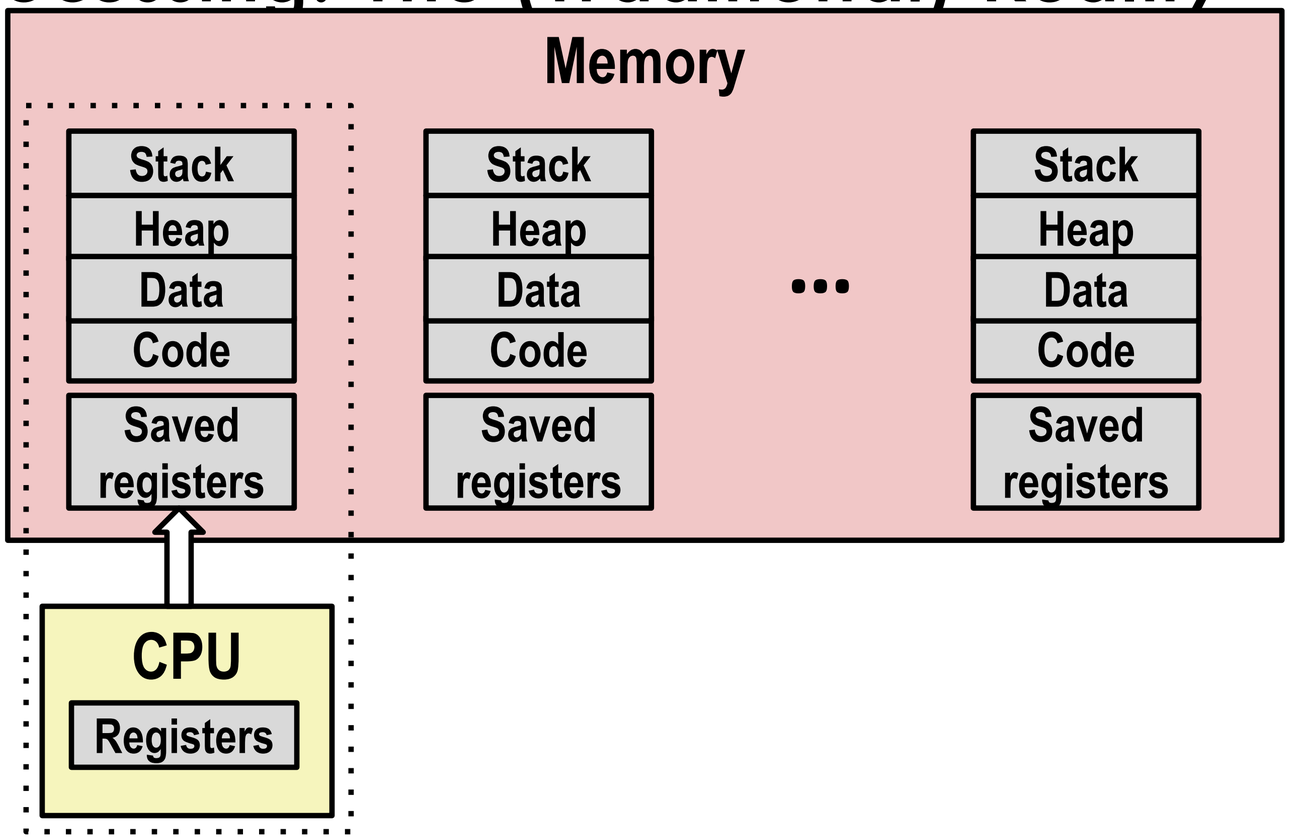
Multiprocessing: The (Traditional) Reality



Single processor executes multiple processes concurrently

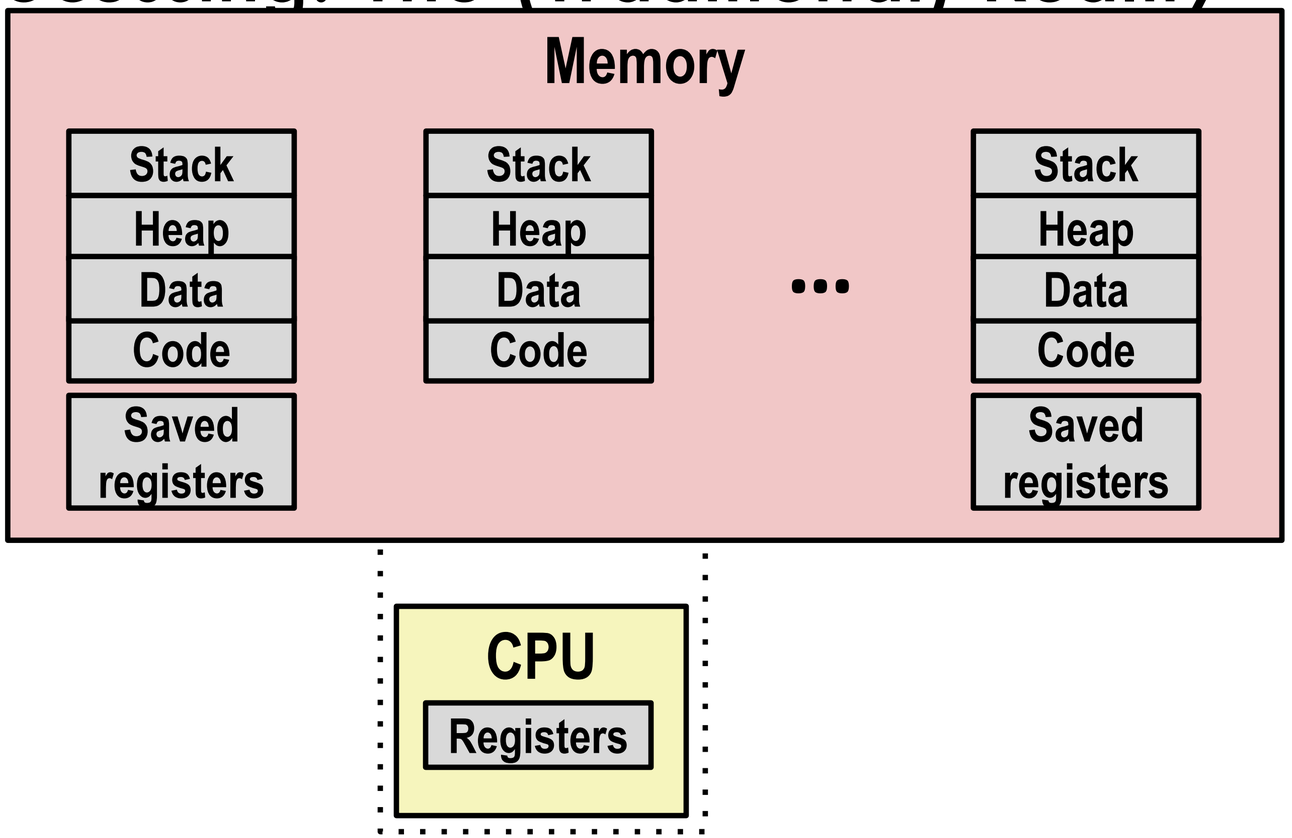
- Process executions interleaved (multitasking)
- Address spaces managed by virtual memory system (like last week)
- Register values for nonexecuting processes saved in memory

Multiprocessing: The (Traditional) Reality



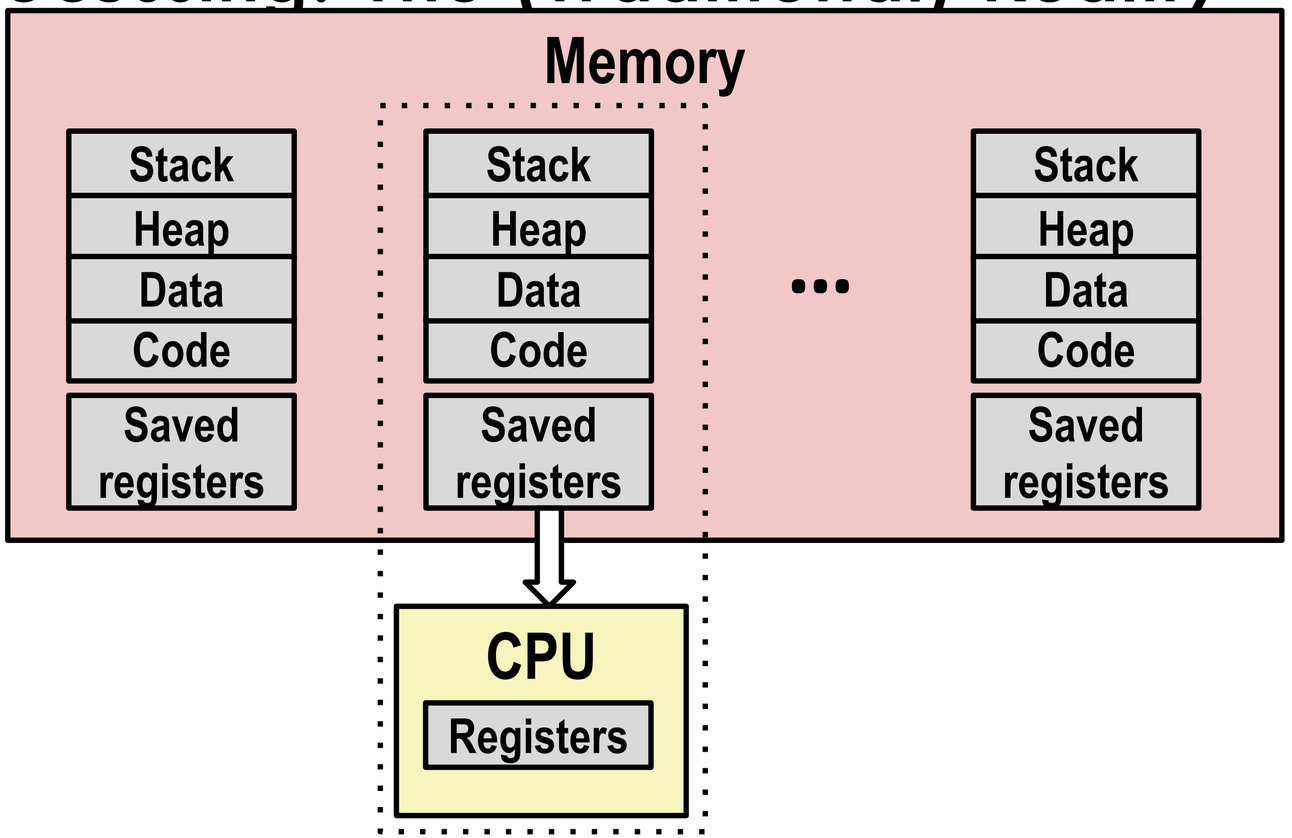
Save current registers in memory

Multiprocessing: The (Traditional) Reality



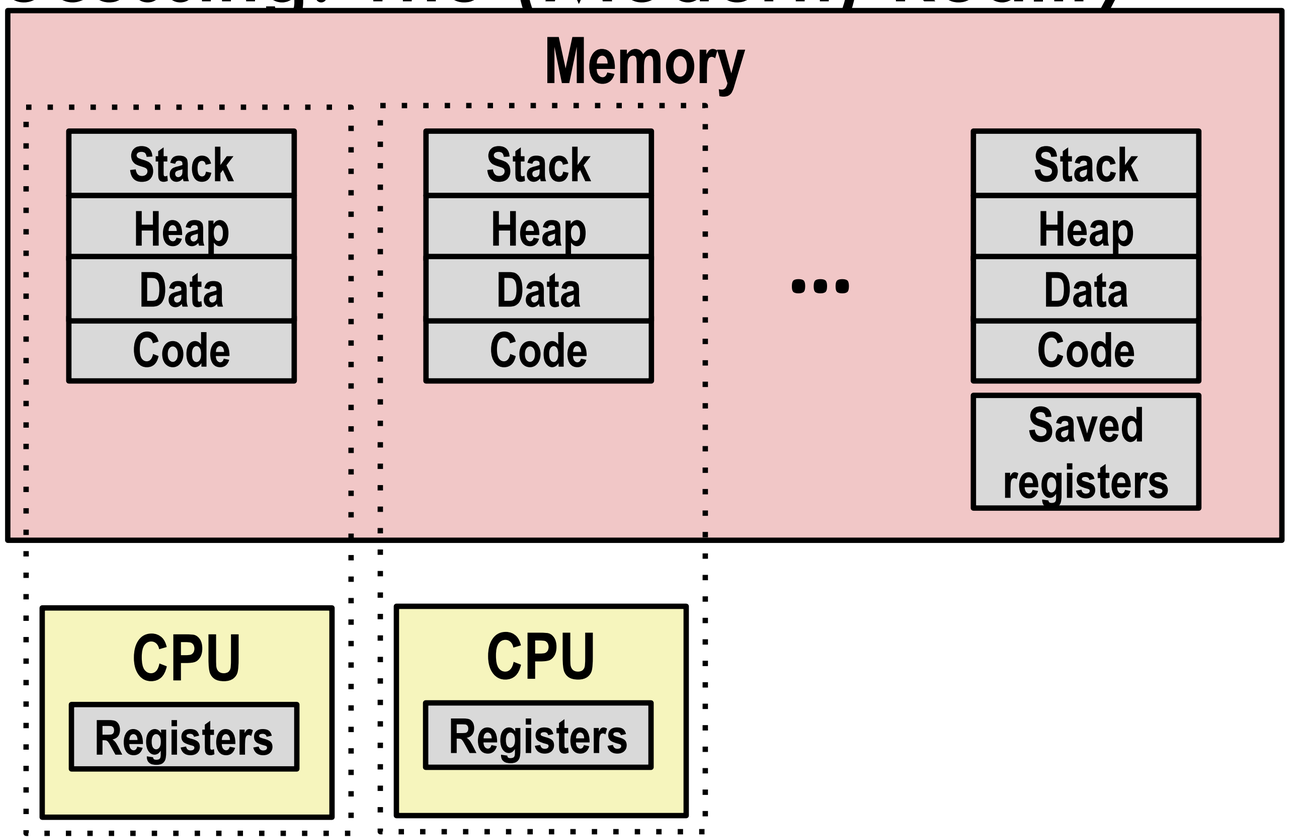
Schedule next process for execution

Multiprocessing: The (Traditional) Reality



Load saved registers and switch address space (context switch)

Multiprocessing: The (Modern) Reality

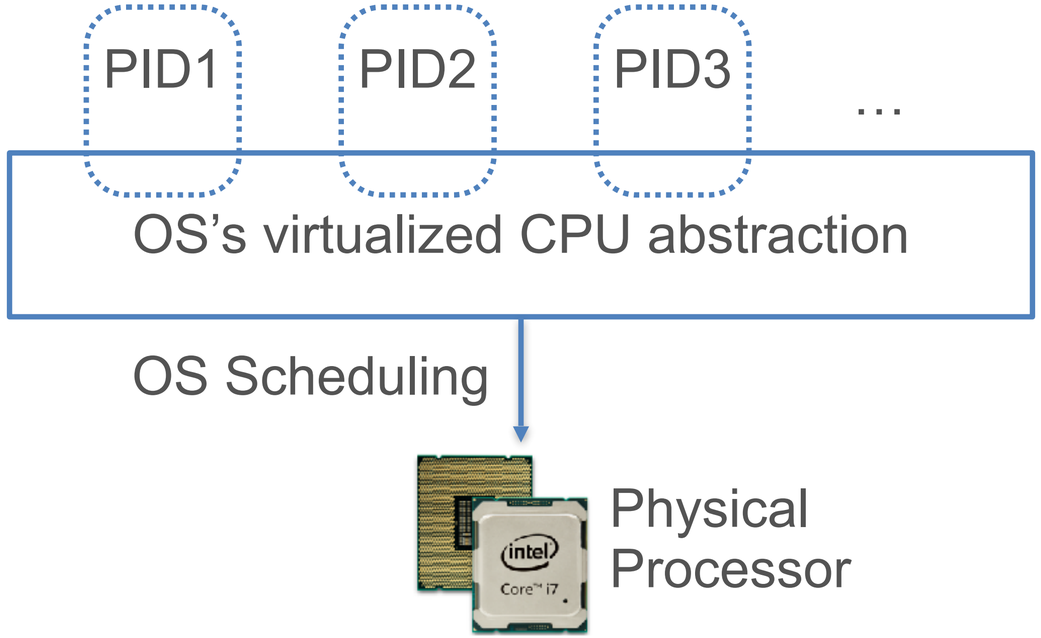


Multicore processors

- 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

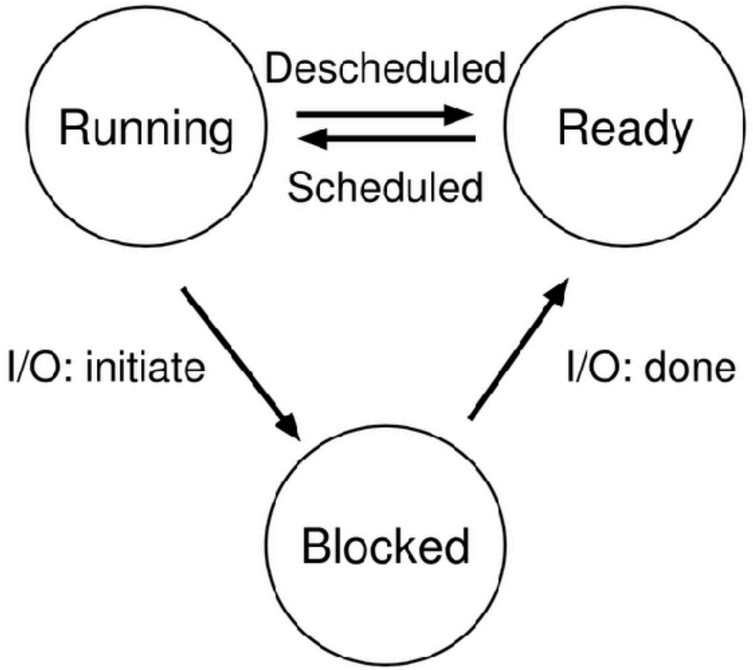
- Virtualization of processor enables process **isolation**, i.e., each process given an “illusion” that it alone runs



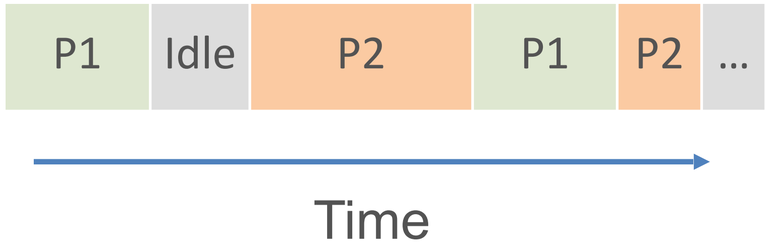
- Inter-process communication possible in System Calls API

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

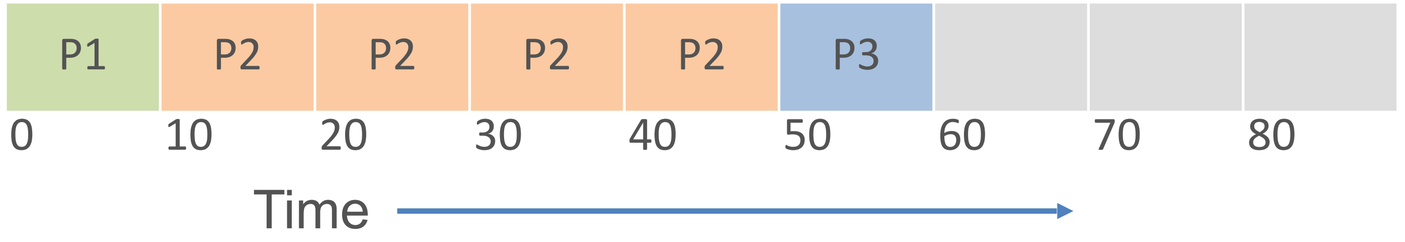
Scheduling Policies/Algorithms

- 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



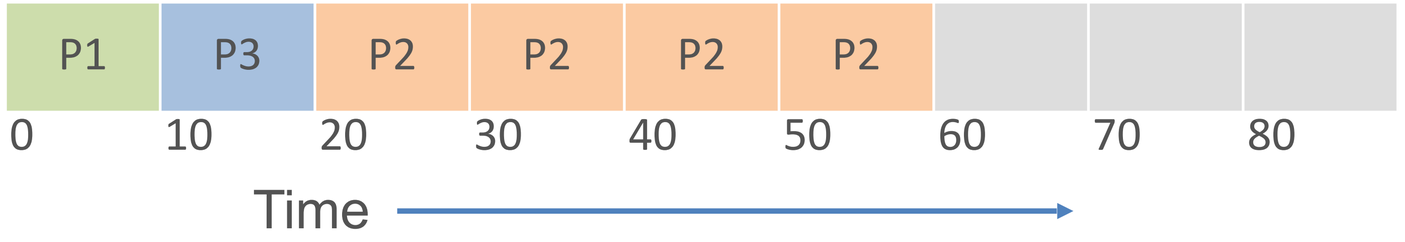
Process	Arrival Time	Start Time	Completion Time	Response Time	Turnaround Time
P1	0	0	10	0	10
P2	0	10	50	10	50
P3	0	50	60	50	60
			Avg:	20	40

- 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



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

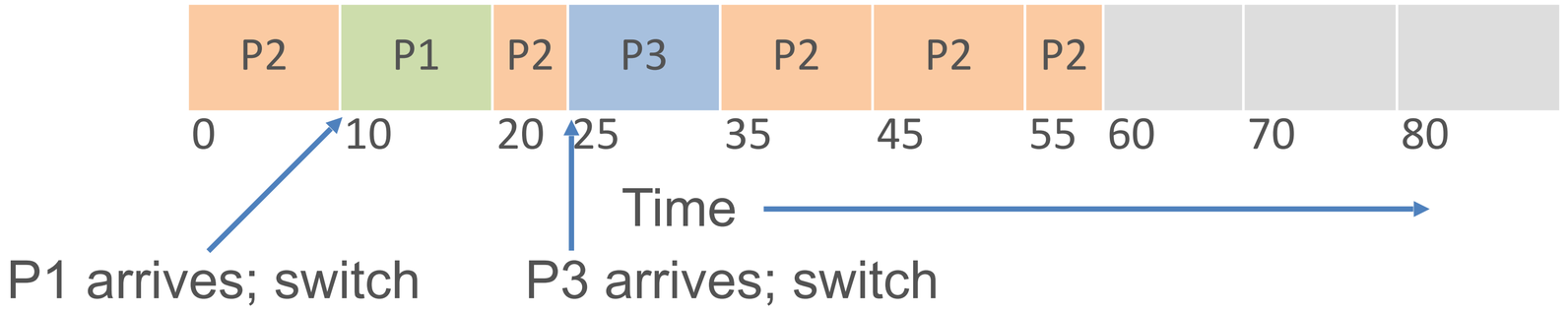
(FIFO Avg: 20 and 40)

- 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



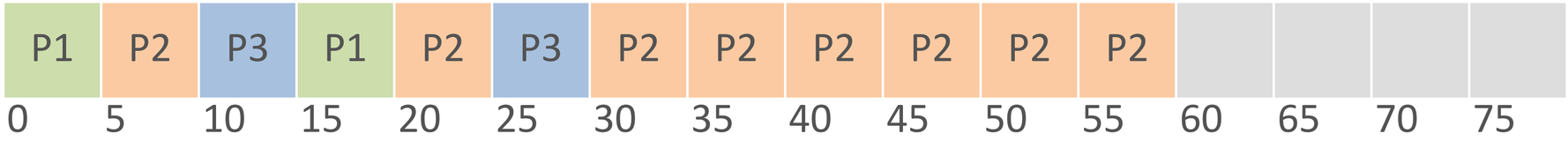
Process	Arrival Time	Start Time	Completion Time	Response Time	Turnaround Time
P1	10	10	20	0	10
P2	0	0	60	0	60
P3	25	25	35	0	10
		(SJF Avg: 10 and 30)	Avg:	0	26.7

- 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



Quantum is 5 Time →

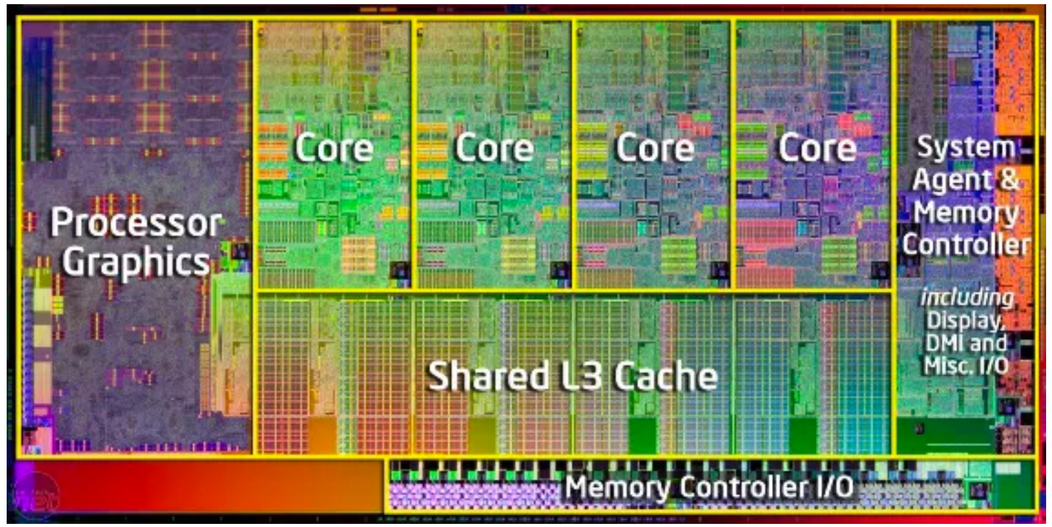
Process	Arrival Time	Start Time	Completion Time	Response Time	Turnaround Time
P1	0	0	20	0	20
P2	0	5	60	5	60
P3	0	10	30	10	30
			Avg:	5	36.7

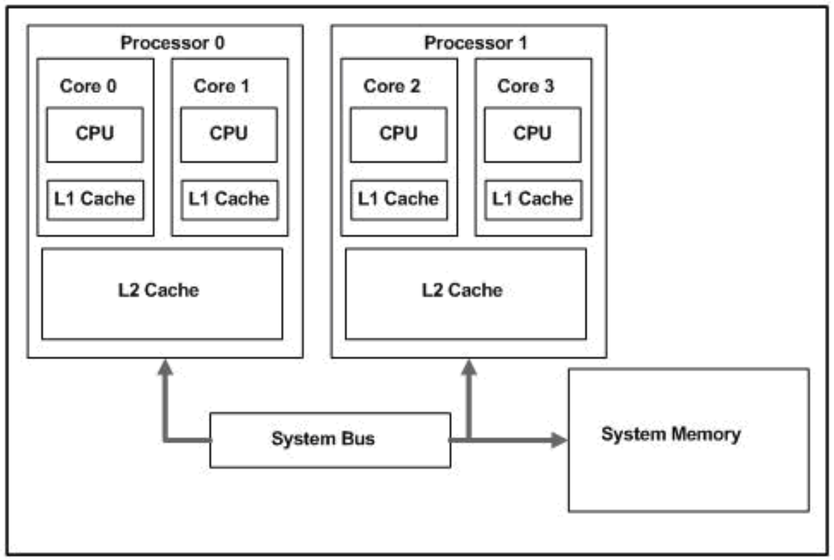
(SJF Avg: 10 & 30; SCTF Avg: 0 & 26.7)

- 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





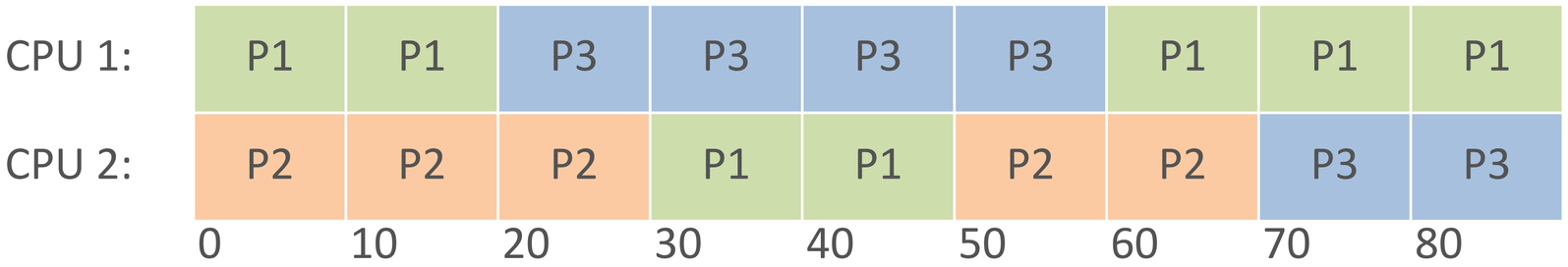
Thread v.s. process

- **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)