Operating Systems: CPU Management

DS 5110: Big Data Systems (Spring 2023)

Lecture 2b

Yue Cheng

University of Virginia

CPU processors and architecture

Basics of CPU processors

- Hardware to execute instructions
  - Other processing units: GPU, TPU, FPGA, etc.
- Instruction Set Architecture (ISA)
  - Vocabulary of instructions of a processor

Basics of CPU processors

- Hardware to execute instructions
  - Other processing units: GPU, TPU, FPGA, etc.
- Instruction Set Architecture (ISA)
  - Vocabulary of instructions of a processor

A C program

Compile ↓

Program in assembly language

Assemble ↓

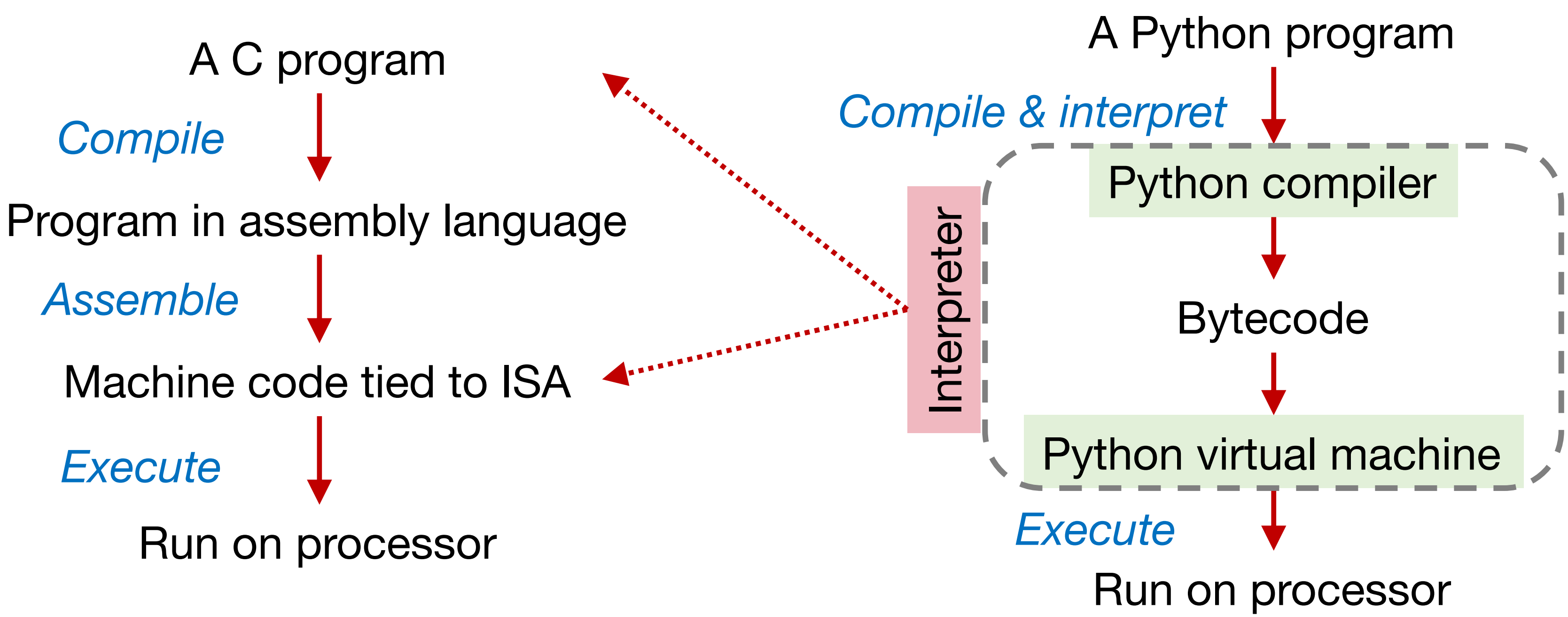
Machine code tied to ISA

Execute ↓

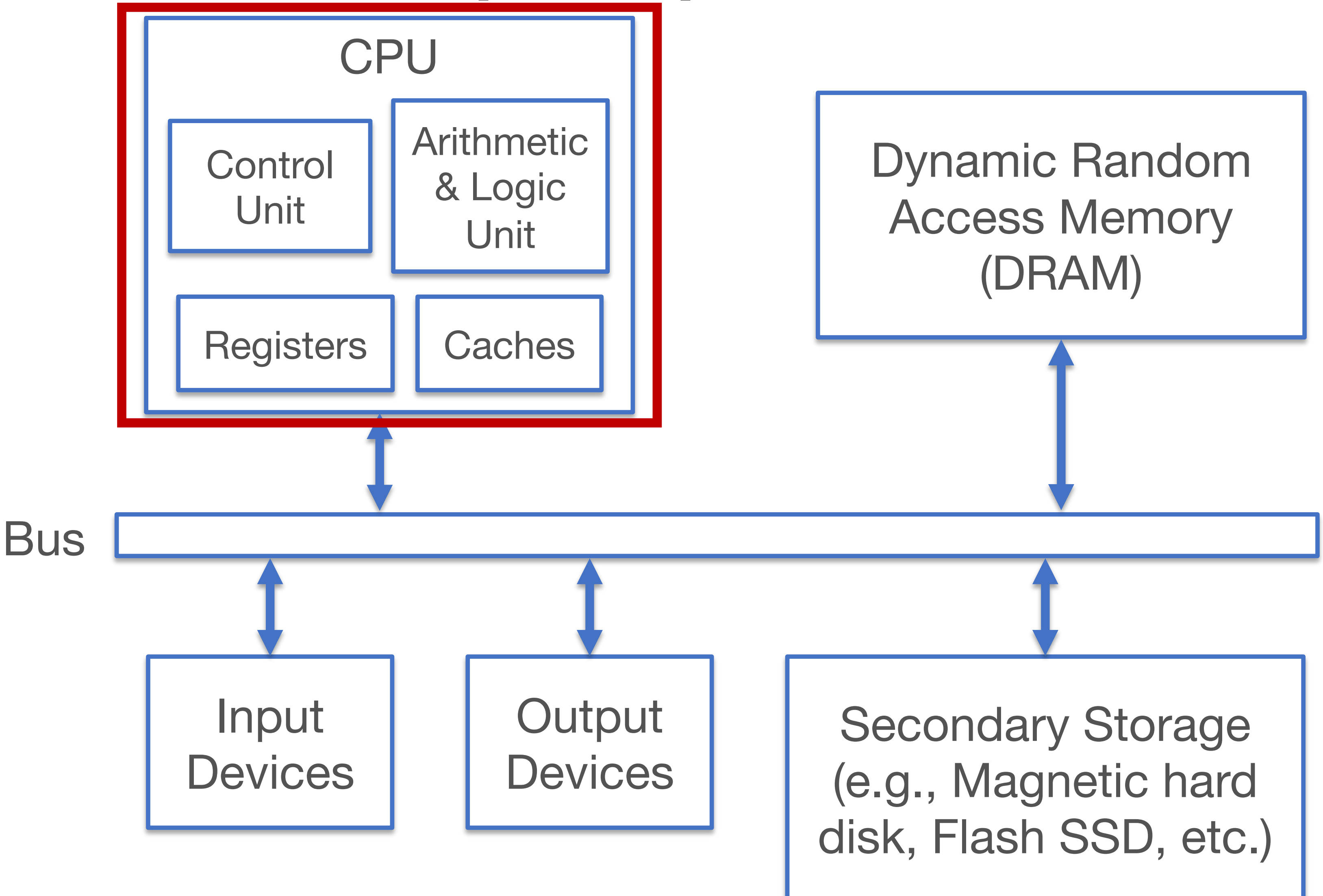
Run on processor

Basics of CPU processors

- Hardware to execute instructions
  - Other processing units: GPU, TPU, FPGA, etc.
- Instruction Set Architecture (ISA)
  - Vocabulary of instructions of a processor



Abstract computer parts



How does a CPU execute machine code?

- Most common approach: load-store architecture
- **Registers:** Tiny local memory ("scratch space") on CPU into which instructions and data are copied
- ISA specifies bit length/format of machine code instructions
- ISA has several instructions to manipulate register contents

How does a CPU execute machine code?

- Type of ISA instructions to manipulate register contents
  - Memory access: `load` (copy bytes from a DRAM address to register); `store` (reverse)
  - Arithmetic & logic on data items in registers: add/multiple/etc.; bitwise ops; compare, etc.; handled by ALU
  - Control flow (branch, call, etc.): handled by CU
- **CPU caches:** Small CPU-local memory to buffer instructions and data

CPU performance

Or, how fast can a CPU execute a program?

CPU performance

Or, how fast can a CPU execute a program?

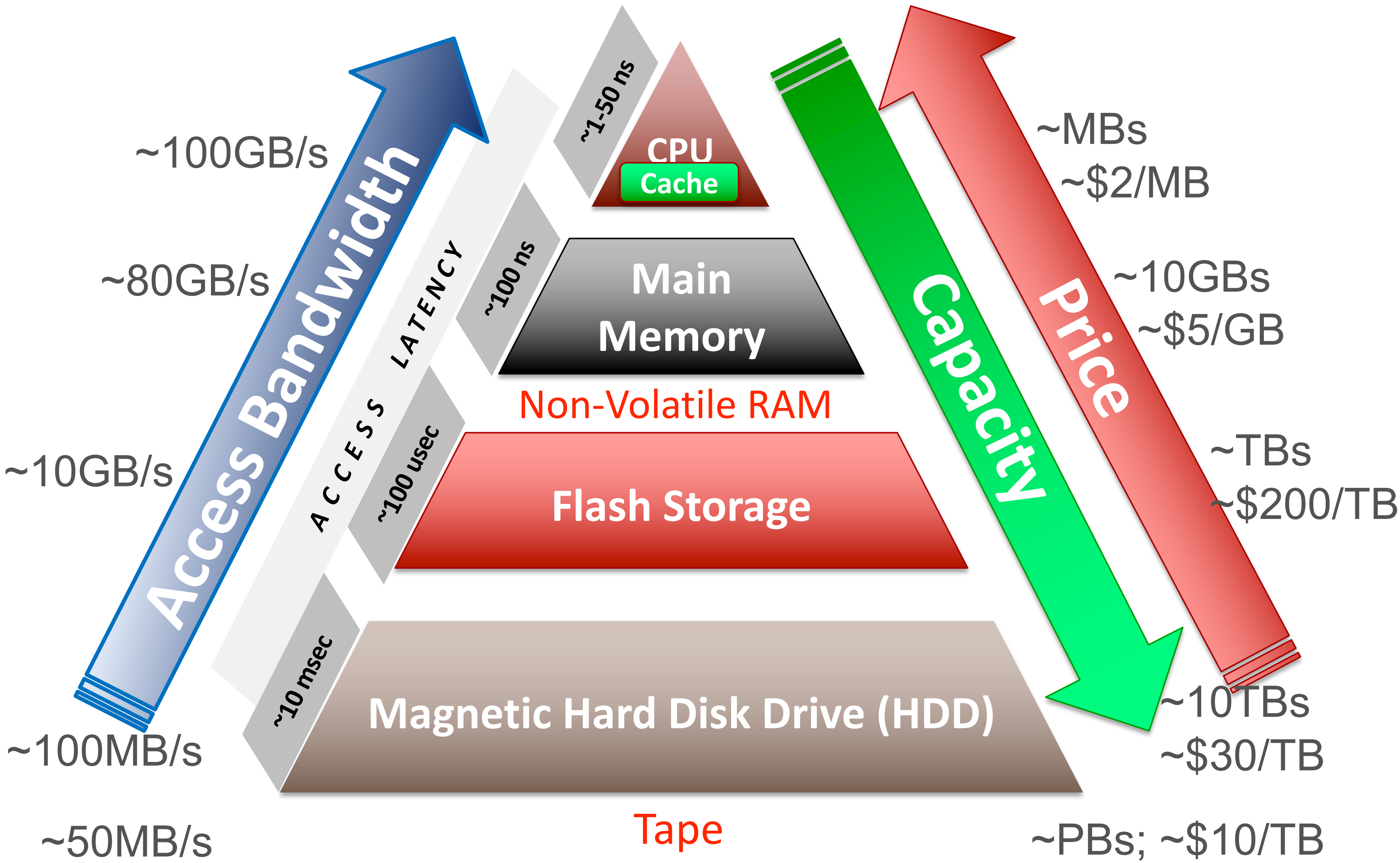
- Modern CPUs can run millions-billions of instructions per second
  - ISA tells #clock cycles per instruction
  - CPU's clock rate helps map that to runtime (ns)

CPU performance

Or, how fast can a CPU execute a program?

- Modern CPUs can run millions-billions of instructions per second
  - ISA tells #clock cycles per instruction
  - CPU's clock rate helps map that to runtime (ns)
- But most programs do not keep CPU always busy
  - Memory access instructions stall the CPU: i.e., ALU & CU idle during DRAM-register transfer
  - Worse, data may not be in DRAM – wait for disk I/O!
  - So, actual runtime of a program may be orders-of-magnitude higher than what clock rate calculation suggests

Memory-storage hierarchy



*UCSD DSC 102: Systems for scalable analysis. Arun Kumar

Key principle: Optimizing use of CPU caches is critical for processor performance!

Or, how fast can a CPU execute a program?

- Modern CPUs can run millions-billions of instructions per second
  - ISA tells #clock cycles per instruction
  - CPU's clock rate helps map that to runtime (ns)
- But most programs do not keep CPU always busy
  - Memory access instructions stall the CPU: i.e., ALU & CU idle during DRAM-register transfer
  - Worse, data may not be in DRAM – **wait for disk I/O**!
  - So, actual runtime of a program may be orders-of-magnitude higher than what clock rate calculation suggests

What is an OS?

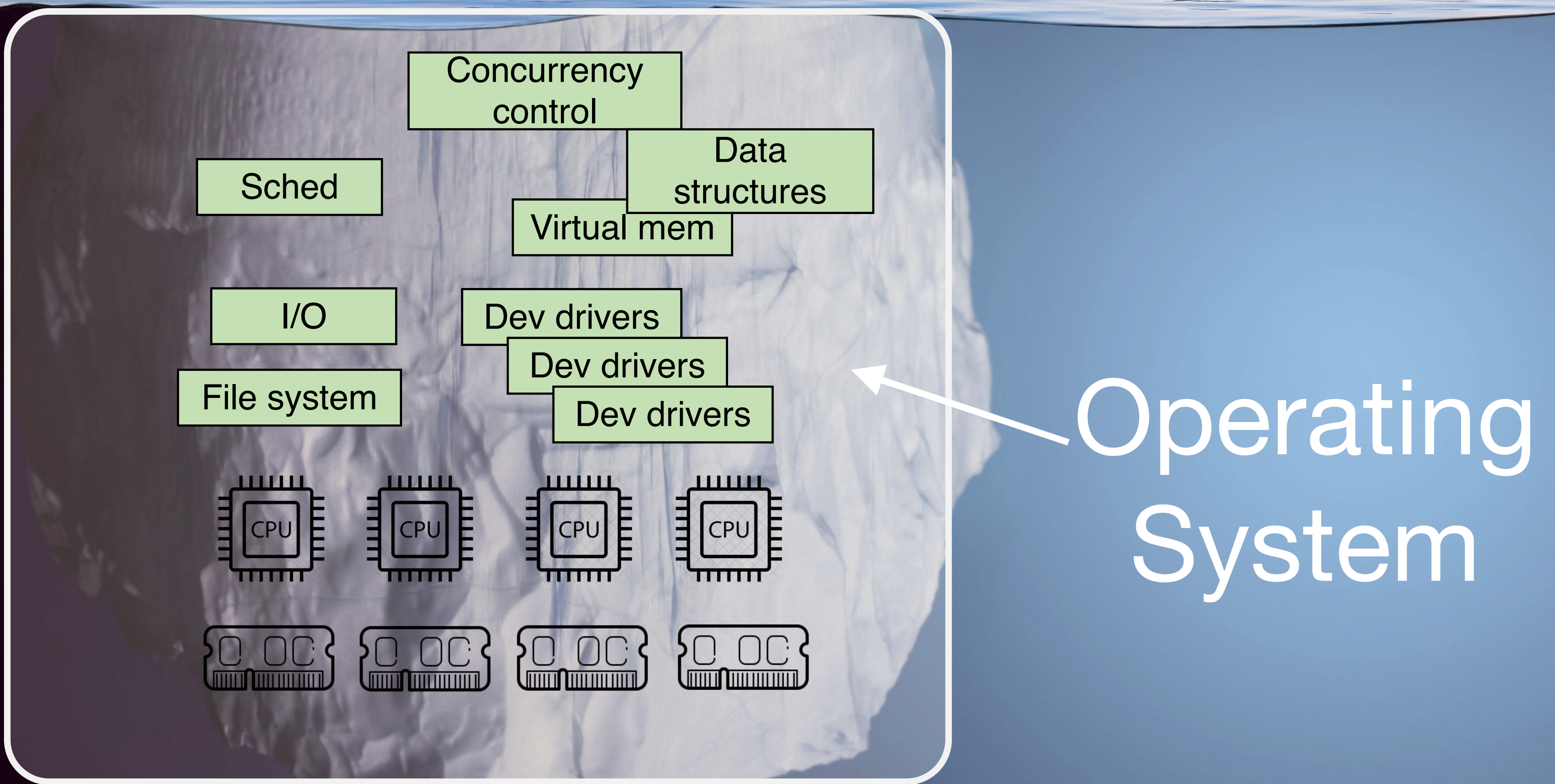
What is an OS?

- OS manages resources
  - Memory, CPU, storage, network
  - Data (file systems, I/O)

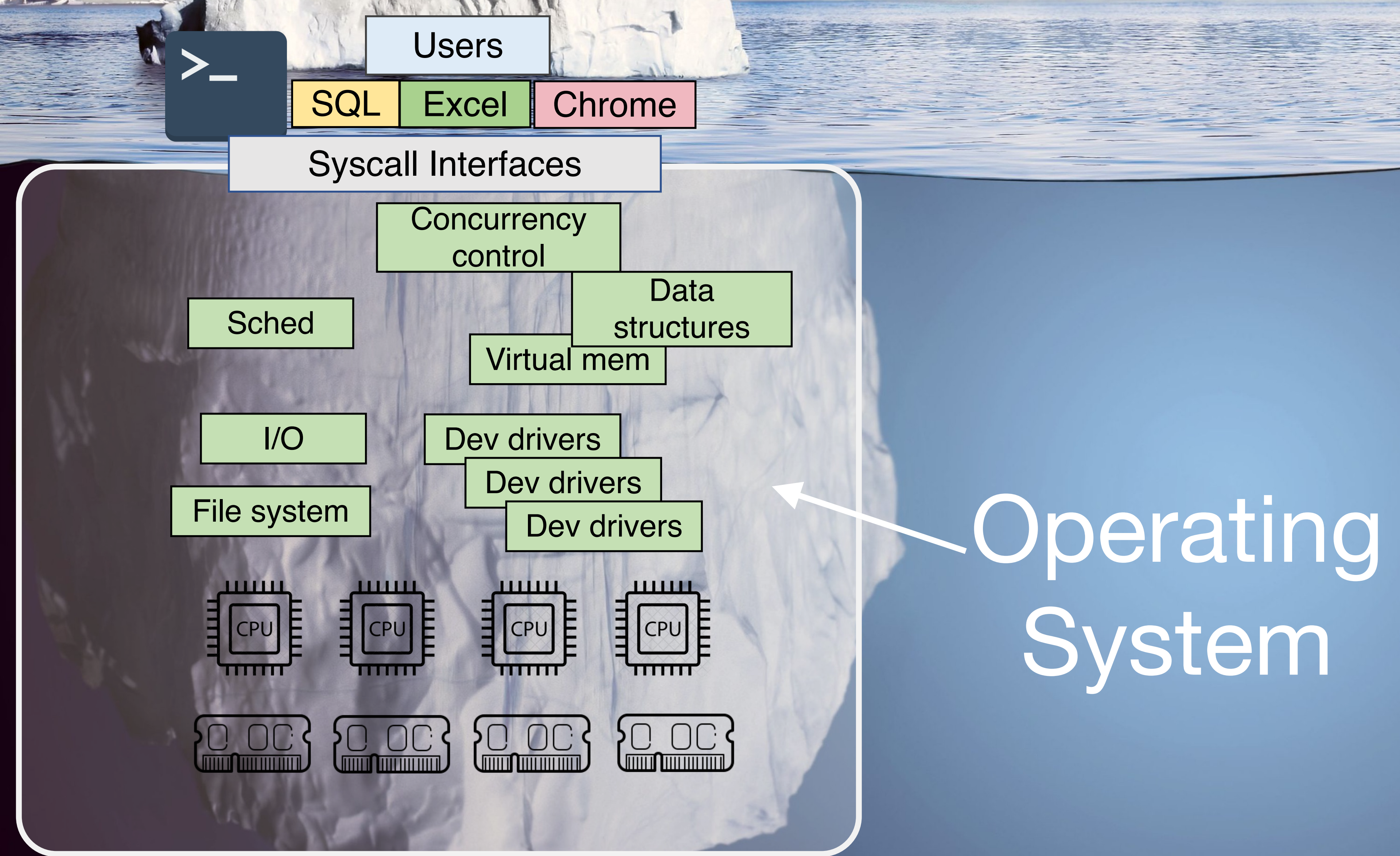
- Provides low-level abstractions to applications
  - Files
  - Processes, threads
  - Virtual machines (VMs), containers
  - …

OS abstracts away low-level details

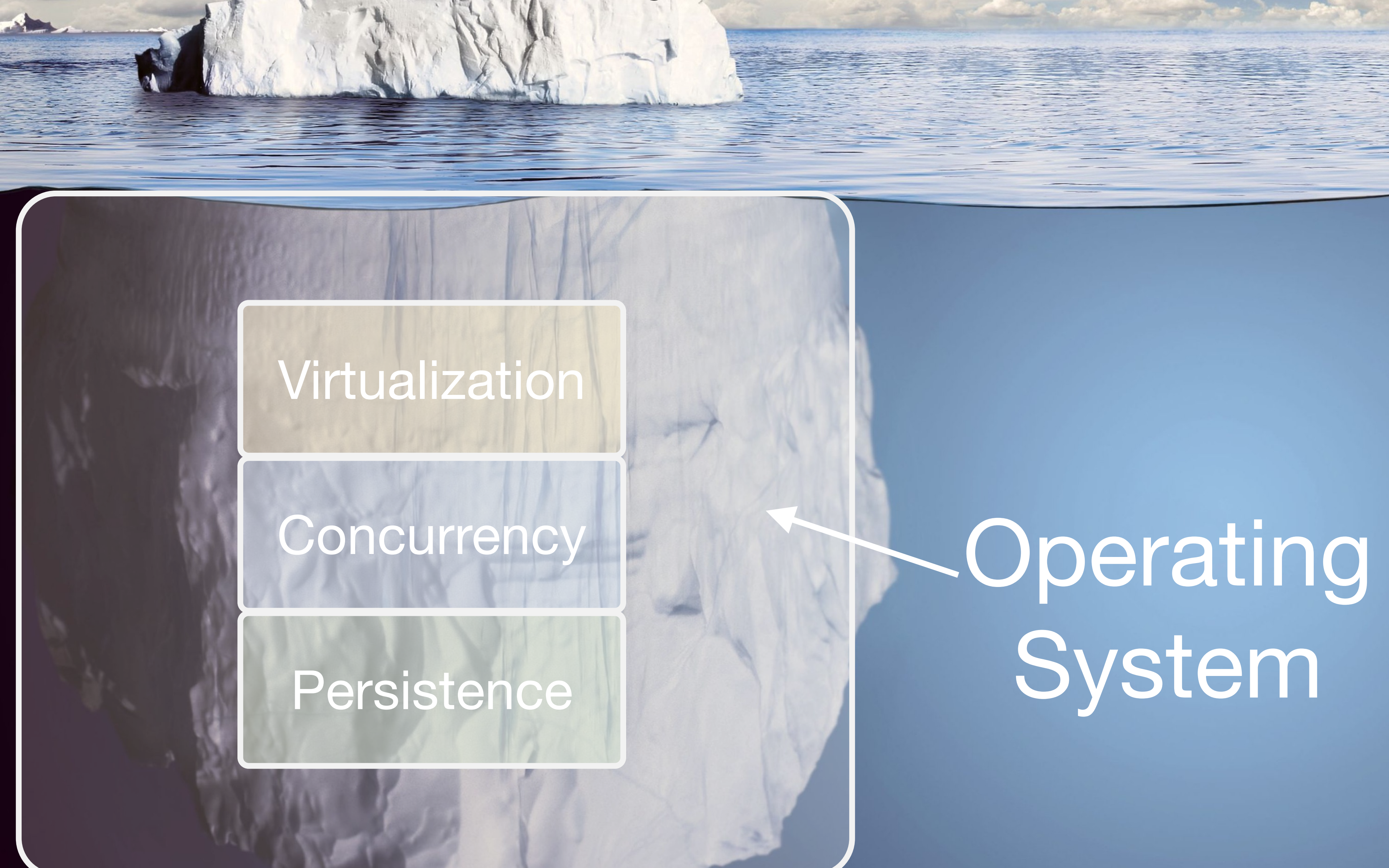
OS abstracts away low-level details



OS abstracts away low-level details



OS abstracts away low-level details



What happens when a program runs?

- A running program executes instructions
  1. The processor **fetches** an instruction from memory
  2. **Decode**: Understand which instruction it is
  3. **Execute**
  4. The processor moves on to **the next instruction** and so on

How does a running program interact with the OS?

- System calls allow a user application to tell the OS what to do
  - OS provides interfaces (APIs)
  - Hundreds of system calls (for Linux)
    - Run programs
    - Access memory
    - Access devices

Virtualization

- OS virtualizes physical resources
  - Gives illusion of private resources

Virtualizing the CPU

- OS creates and manages many virtual CPUs
  - Turning a single CPU into seemingly infinite number of CPUs
  - Allowing many programs to seemingly run at once (concurrently)

Virtualizing memory

- The physical memory is an array of bytes
- A program keeps (most of) its data in memory
  - Read memory (load): Access an address to fetch the data
  - Write memory (store): Store the data to a given address

Concurrency

- OS is juggling many things at once
  - First running one process, then another, and so forth
- Multi-threaded programs also have concurrency problem

Persistence

- Main memory (DRAM) is **volatile**
- How to persist data?
  - **Hardware**: I/O devices such as hard disk drives (HDDs)
  - **Software**: File systems

What is a process?

What is a process?

- Programs are code (static entity)
- Processes are running programs

What is a process?

- Programs are code (static entity)
- Processes are running programs

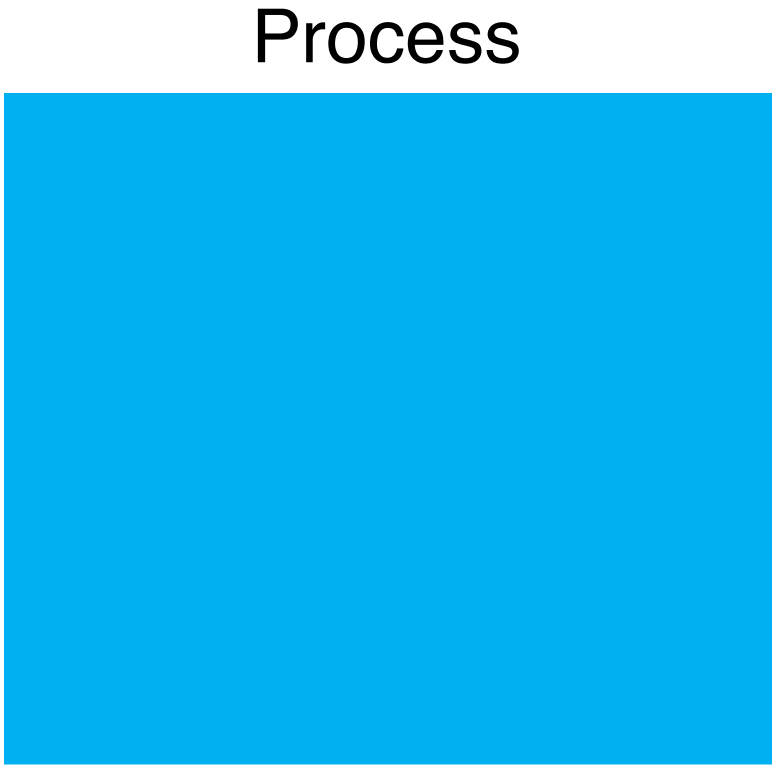
- **Q:** Why bother knowing process management in Data Science?

What is a process?

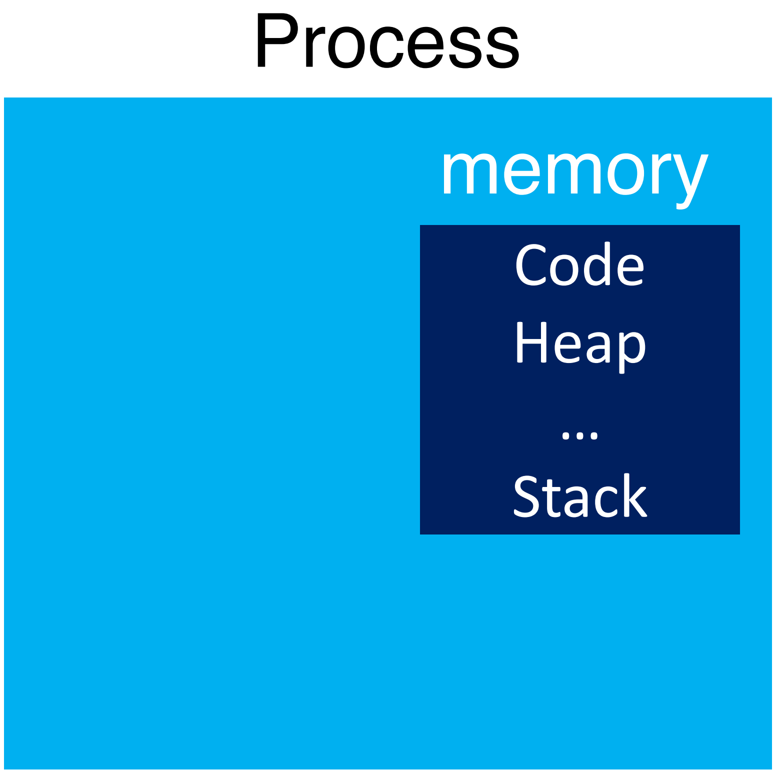
- Programs are code (static entity)
- Processes are running programs

- **Q:** Why bother knowing process management in Data Science?
  - Everything in Data Science runs in a process
  - A large data system is multiple cooperating, running processes that execute user-submitted jobs/queries

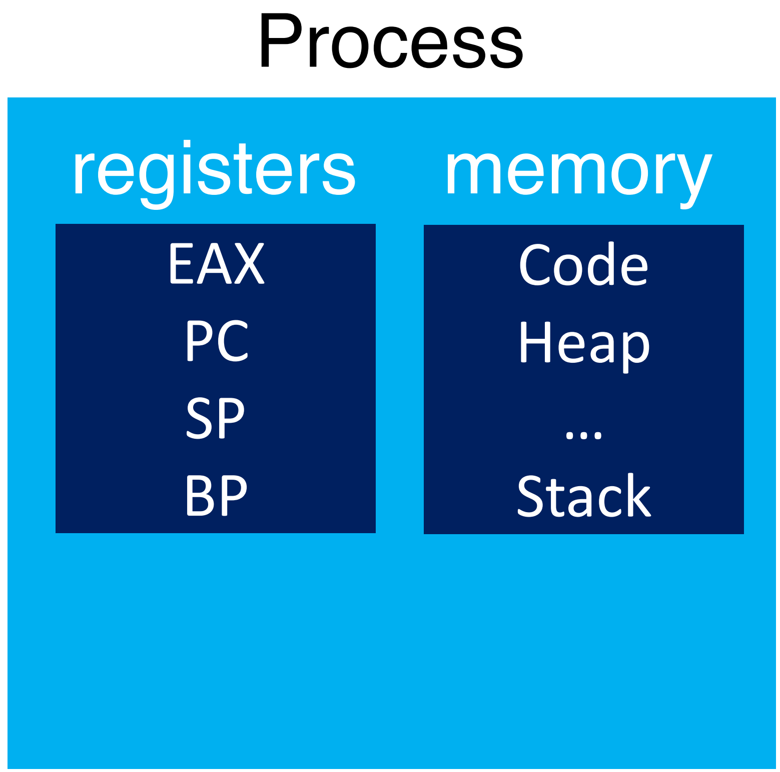
What is in a process?



What is in a process?

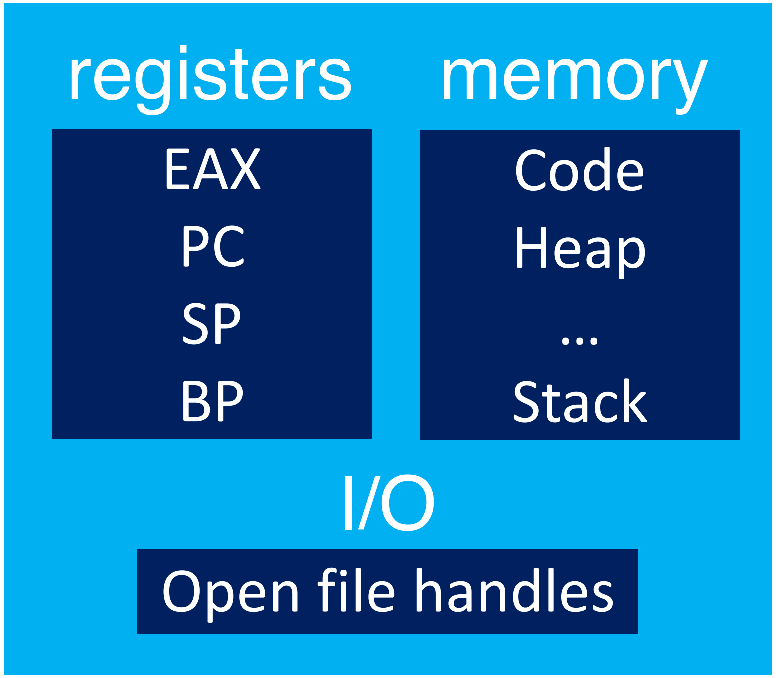


What is in a process?



What is in a process?

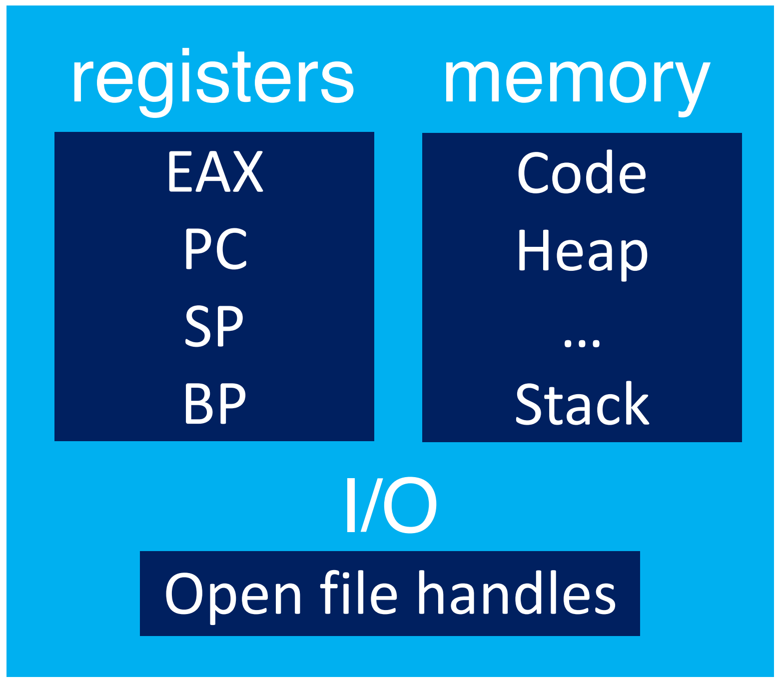
Process



What things change as a program runs?

What is in a process?

Process



What things change as a program runs?

Running program's internal state (runtime data)

Peeking inside

- Processes share code, but each has its own "context"
- CPU state
  - Instruction pointer (Program Counter)
  - Stack pointer
- Memory state
  - Set of memory addresses ("address space")
  - `cat /proc/<PID>/maps`
- Disk state
  - Set of file handles (file descriptors or fd)
  - `cat /proc/<PID>/fdinfo/*`

Is it not safe/secure for OS to hand off control of hardware to a process?

Is it not safe/secure for OS to hand off control of hardware to a process?

- **Limited direct execution (LDE):** Low-level mechanism that implements the user-kernel space separation

- Usually let processes run with no OS involvement
- Limit what processes can do
- Offer privileged operations through well-defined channels with help of OS

Limited Direct Execution (LDE)

Limited Direct Execution (LDE)

Running program

LDE
mechanism

OS

Sharing (virtualizing) the CPU

How does OS share…

- CPU?
- Memory?
- Disk?

How does OS share…

- CPU? (a: time sharing)
- Memory? (a: space sharing)
- Disk? (a: space sharing)

How does OS share…

- CPU? (a: time sharing)

Today

- Memory? (a: space sharing)

- Disk? (a: space sharing)

How does OS share...

- CPU? (a: time sharing)

Today

- Memory? (a: space sharing)
- Disk? (a: space sharing)

Goal: processes should **not** know they are sharing
(each process will get its own virtual CPU)

What to do with processes that are not running?

- A: Store context in OS structures

What to do with processes that are not running?

- A: Store context in OS structures

- Context:
  - CPU registers
  - Open file descriptors
  - State (sleeping, running, etc.)

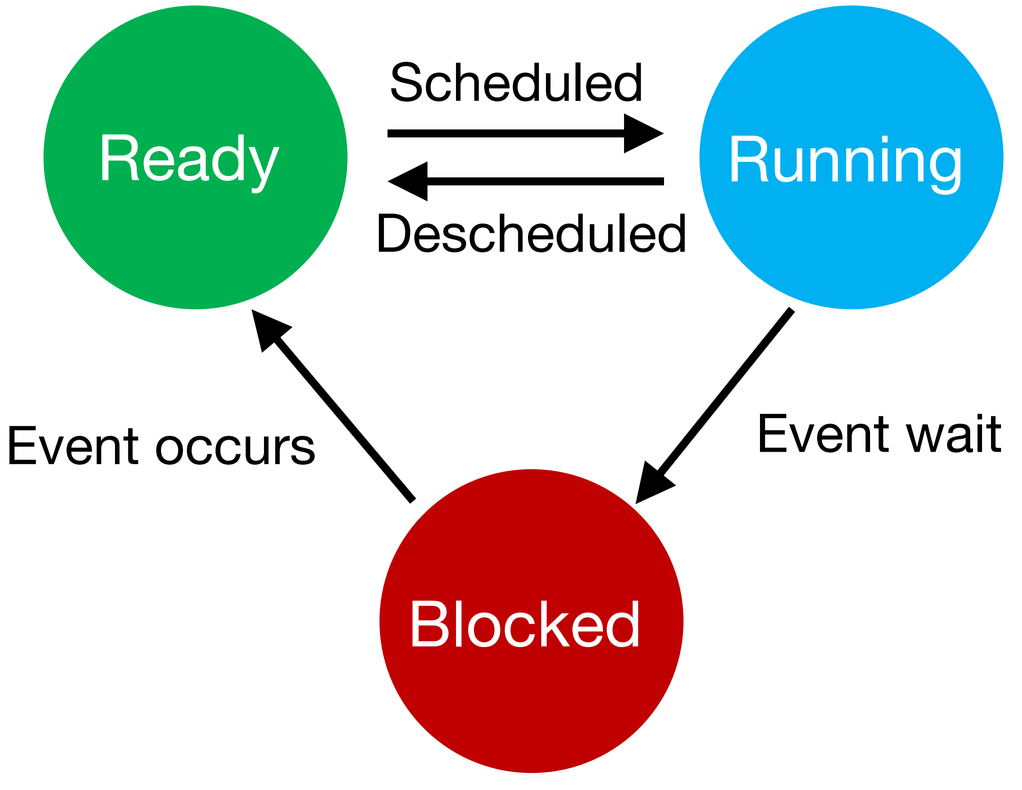
What to do with processes that are not running?

- A: Store context in OS structures

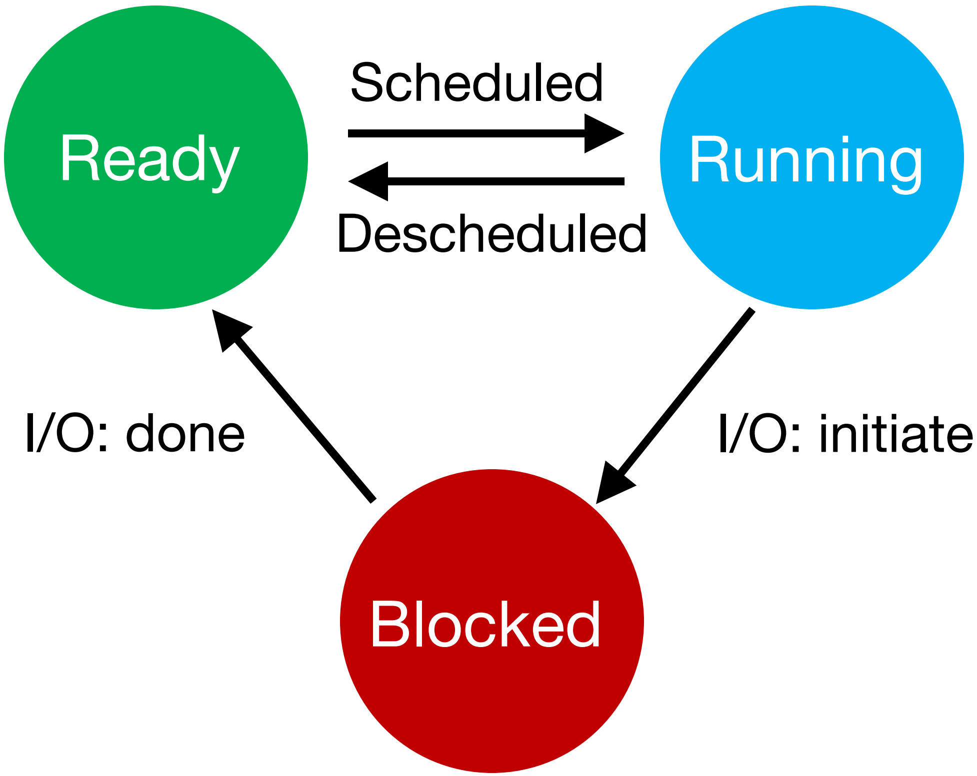
- Context:
  - CPU registers
  - Open file descriptors
  - State (sleeping, running, etc.)

Program-specific runtime data

Process state transitions

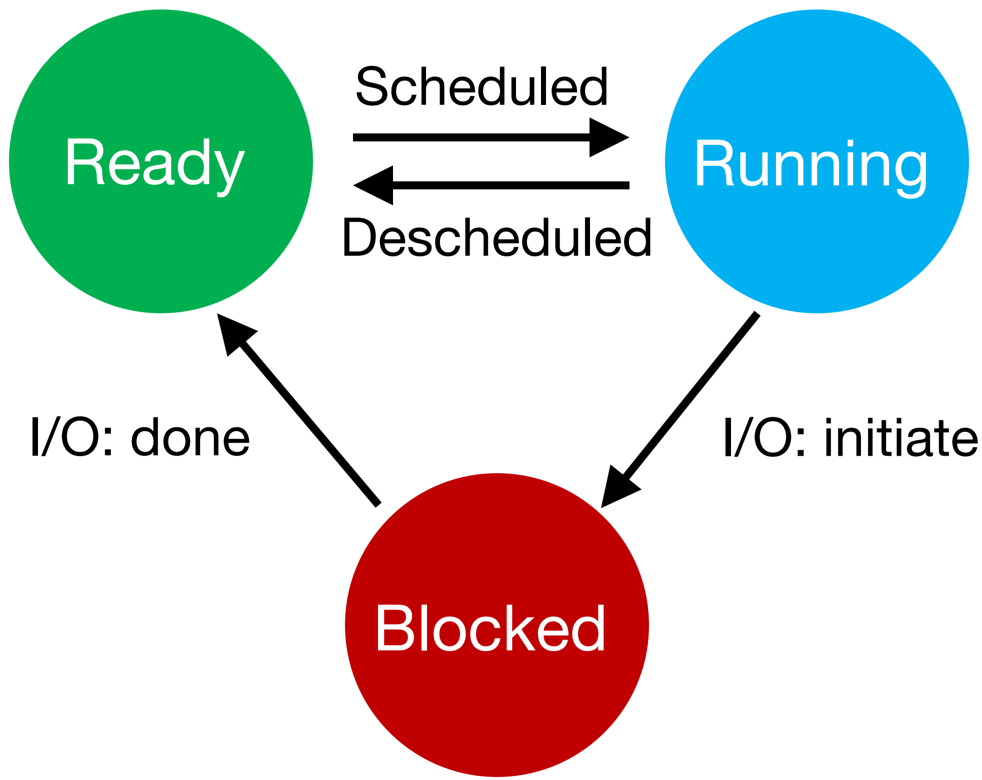


Process state transitions



On a Linux/Mac: View process state with "`ps xa`"

How to transition? (mechanism)
When to transition? (policy)



On a Linux/Mac: View process state with "`ps xa`"

CPU scheduling policies/algorithms

- **Problem to solve:** How to optimize the tradeoff b/w overall workload performance and fairness?
  - Given that the number of processes (applications) is way larger than that of the available CPU cores

- Processes get queued up and the CPU scheduler will select one in the ready queue for execution
- The scheduling policies may have tremendous effects on the system efficiency
  - Interactive systems: Responsiveness (latency)
  - General-purpose systems: Fairness in CPU usage

First-In, First-Out

Workload assumptions

1. Each job runs for the same amount of time

2. All jobs arrive at the same time

3. All jobs only use the CPU (no I/O)

4. The runtime of each job is known

FIFO

- First-In, First-Out: Run jobs in arrival order

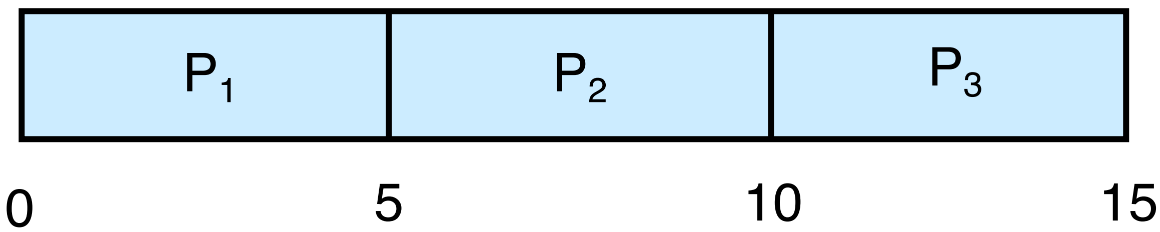
Proc	Arrival time	Runtime
P1	~0	5
P2	~0	5
P3	~0	5

FIFO

- First-In, First-Out: Run jobs in arrival order

Proc	Arrival time	Runtime
P1	~0	5
P2	~0	5
P3	~0	5

Gantt chart

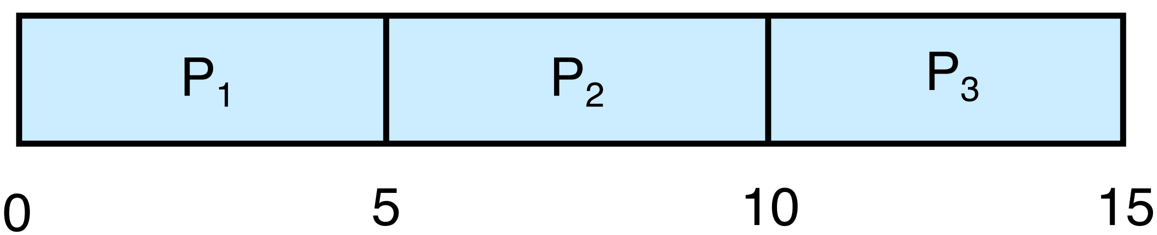


FIFO

- First-In, First-Out: Run jobs in arrival order

Proc	Arrival time	Runtime
P1	~0	5
P2	~0	5
P3	~0	5

Gantt chart



What is the average turnaround time?
Def: turnaround_time = completion_time – arrival_time

Workload assumptions

1. Each job runs for the same amount of time

2. All jobs arrive at the same time

3. All jobs only use the CPU (no I/O)

4. The runtime of each job is known

Workload assumptions

1. ~~Each job runs for the same amount of time~~

2. All jobs arrive at the same time

3. All jobs only use the CPU (no I/O)

4. The runtime of each job is known

Example: big first job

Proc	Arrival time	Runtime
P1	~0	80
P2	~0	5
P3	~0	5

What is the average turnaround time?

Example: big first job

Proc	Arrival time	Runtime
P1	~0	80
P2	~0	5
P3	~0	5

What is the average turnaround time?

Example: big first job

Proc	Arrival time	Runtime
P1	~0	80
P2	~0	5
P3	~0	5

What is the average turnaround time?

Average turnaround time: (80+85+90) / 3 = 85

Convoy effect!!

Better schedule?

Shortest Job First (SJF)

Passing the tractor

- New scheduler: SJF (Shortest Job First)

- Policy: When deciding which job to run, choose the one with the smallest runtime

Example: SJF

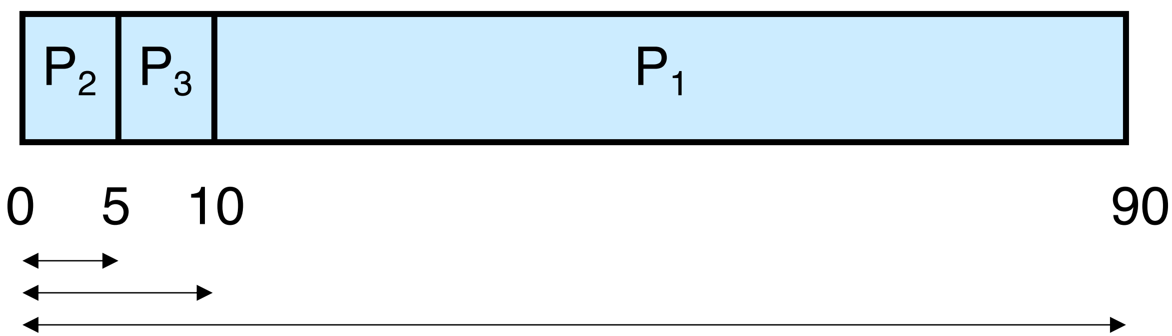
Proc	Arrival time	Runtime
P1	~0	80
P2	~0	5
P3	~0	5

What is the average turnaround time with SJF?

Example: SJF

Proc	Arrival time	Runtime
P1	~0	80
P2	~0	5
P3	~0	5

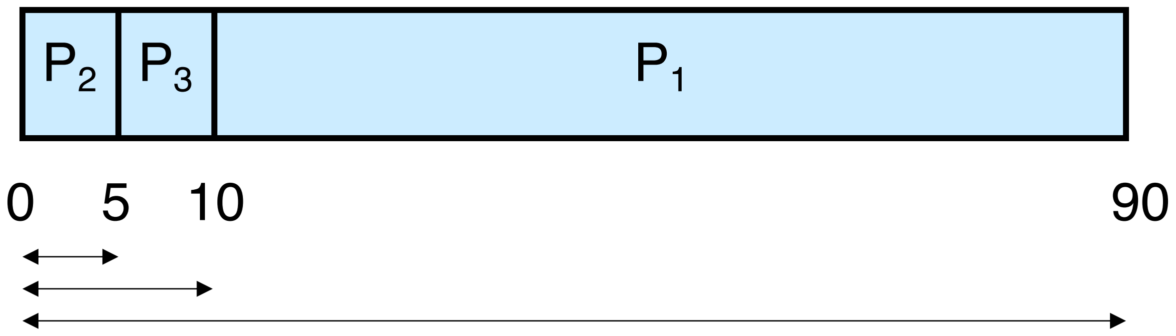
What is the average turnaround time with SJF?



Example: SJF

Proc	Arrival time	Runtime
P1	~0	80
P2	~0	5
P3	~0	5

What is the average turnaround time with SJF?



Average turnaround time: (5+10+90) / 3 = 35

Workload assumptions

1. Each job runs for the same amount of time

2. All jobs arrive at the same time

3. All jobs only use the CPU (no I/O)

4. The runtime of each job is known

Workload assumptions

1. ~~Each job runs for the same amount of time~~
2. ~~All jobs arrive at the same time~~
3. All jobs only use the CPU (no I/O)
4. The runtime of each job is known

What if jobs arrive at different time?

Shortest Job First (arrival time)

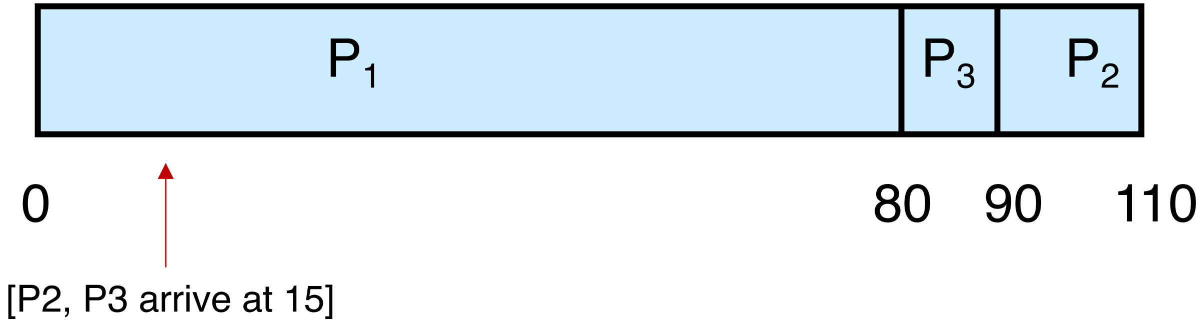
Proc	Arrival time	Runtime
P1	~0	80
P2	~15	20
P3	~15	10

What is the average turnaround time with SJF?

Shortest Job First (arrival time)

Proc	Arrival time	Runtime
P1	~0	80
P2	~15	20
P3	~15	10

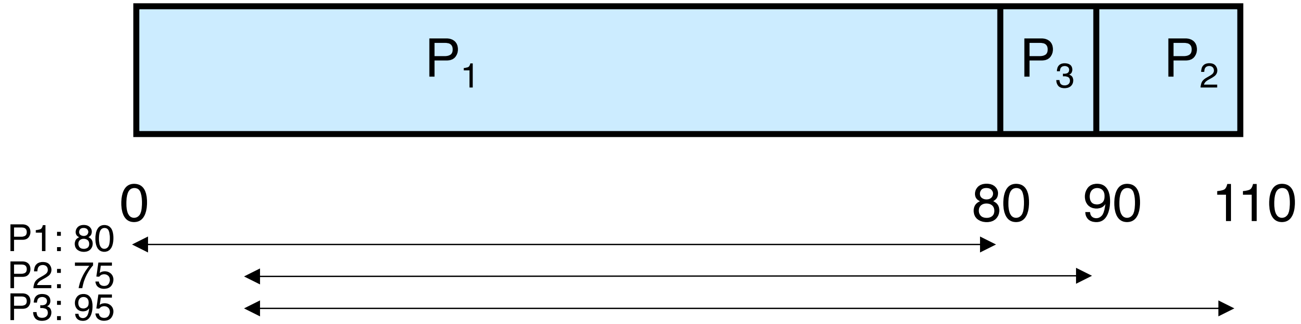
What is the average turnaround time with SJF?



Shortest Job First (arrival time)

Proc	Arrival time	Runtime
P1	~0	80
P2	~15	20
P3	~15	10

What is the average turnaround time with SJF?



Average turnaround time: (80+75+95) / 3 = ~83.3

Preemption: Job can be preempted/interrupted

A preemptive scheduler

- Previous schedulers: FIFO and SJF are non-preemptive

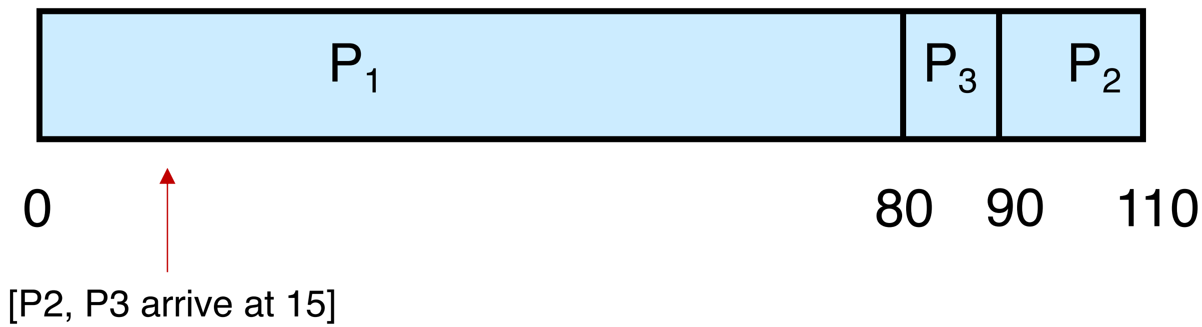
- New scheduler:

  STCF (Shortest Time-to-Completion First)

- Policy: Switch jobs so we always run the one that will complete the quickest

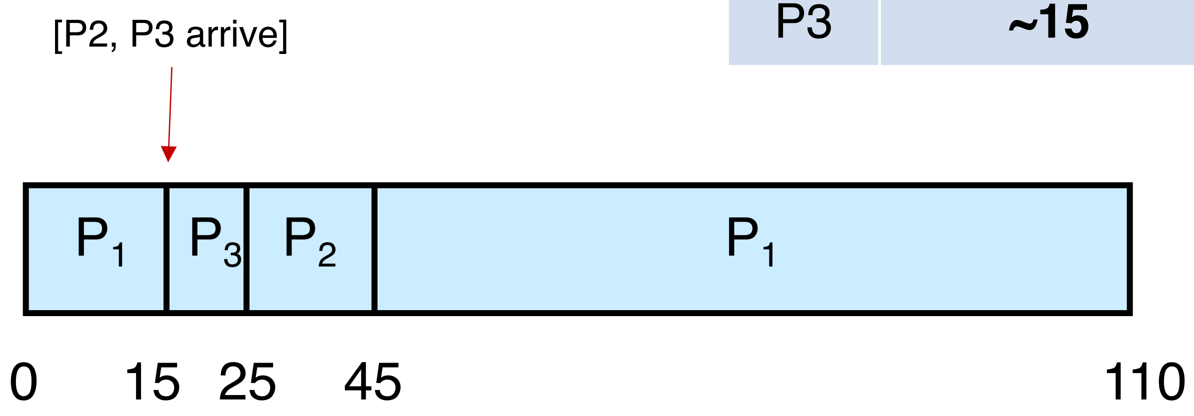
SJF

Proc	Arrival time	Runtime
P1	~0	80
P2	**~15**	20
P3	**~15**	10



STCF

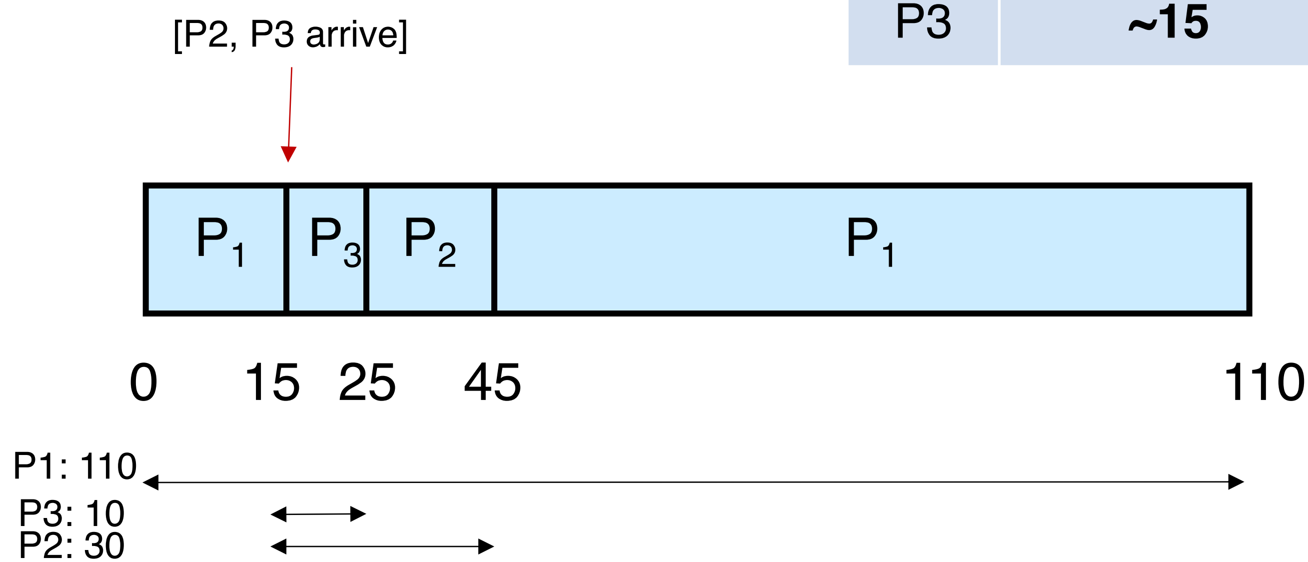
Proc	Arrival time	Runtime
P1	~0	80
P2	**~15**	20
P3	**~15**	10



What is the average turnaround time with STCF?

STCF

Proc	Arrival time	Runtime
P1	~0	80
P2	**~15**	20
P3	**~15**	10



Average turnaround time: (110+30+10) / 3 = 50

Workload assumptions

1. Each job runs for the same amount of time

2. All jobs arrive at the same time

3. All jobs only use the CPU (no I/O)

4. The runtime of each job is known

Workload assumptions

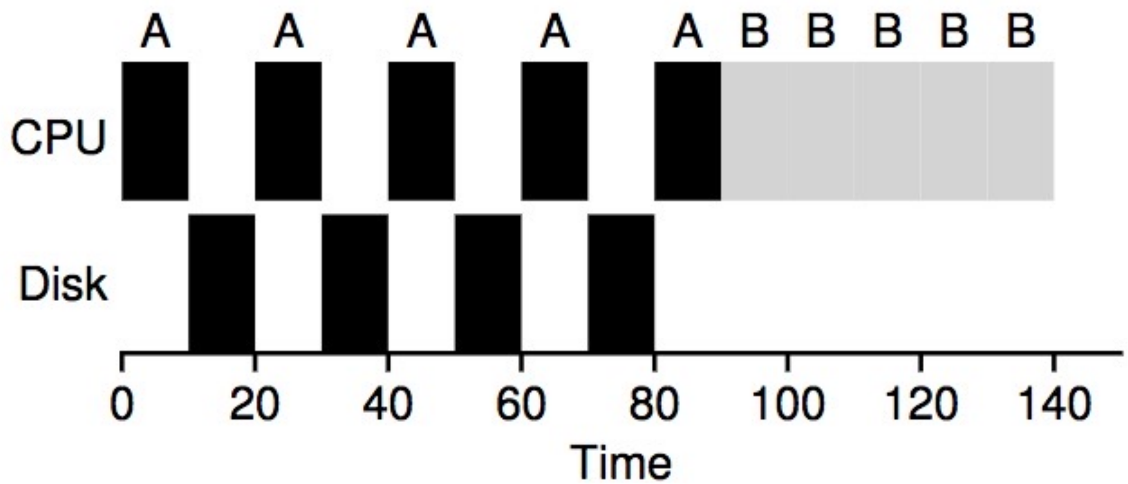
1. ~~Each job runs for the same amount of time~~
2. ~~All jobs arrive at the same time~~
3. ~~All jobs only use the CPU (no I/O)~~
4. The runtime of each job is known

What if jobs do I/Os as well?

- No good if a program can only do pure CPU-intensive compute

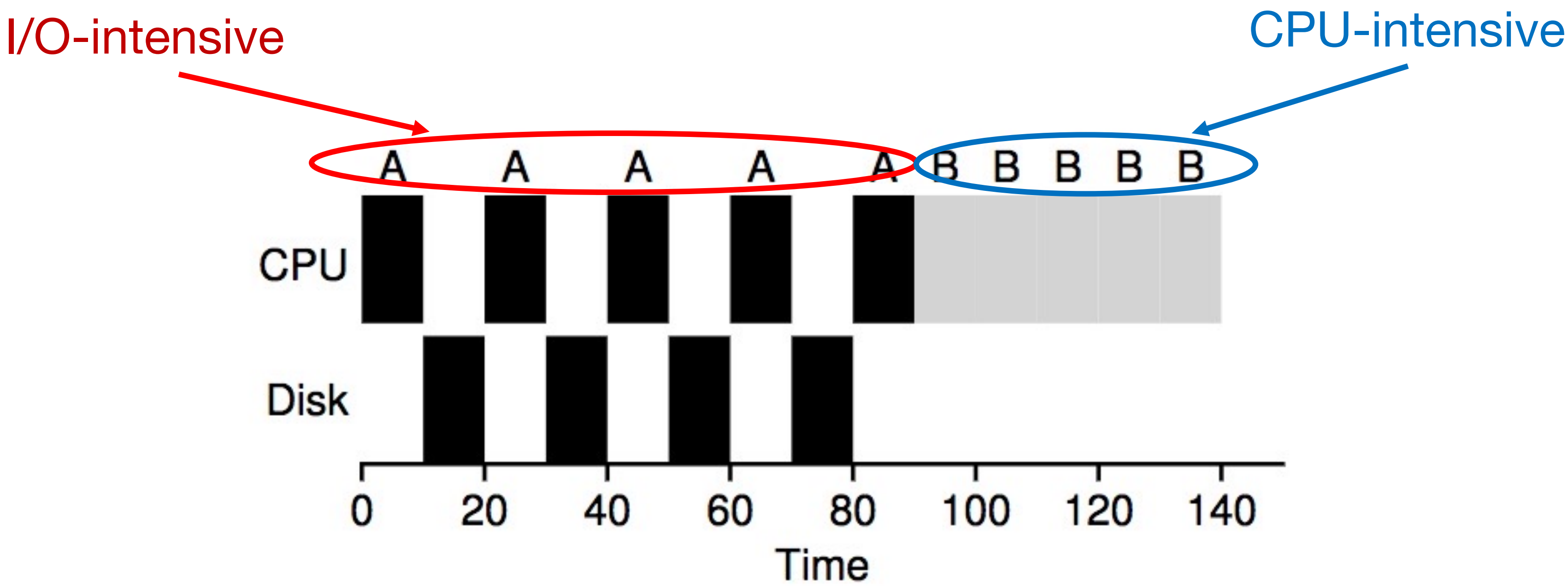
- A common execution pattern of the typical big data applications (Hadoop, Spark, Dask)
  1. completes the CPU burst
  2. performs I/O (e.g., read new CSV files from disk into DRAM)
  3. rejoins the ready queue…
  4. and completes the second CPU bursts…

Not I/O Aware



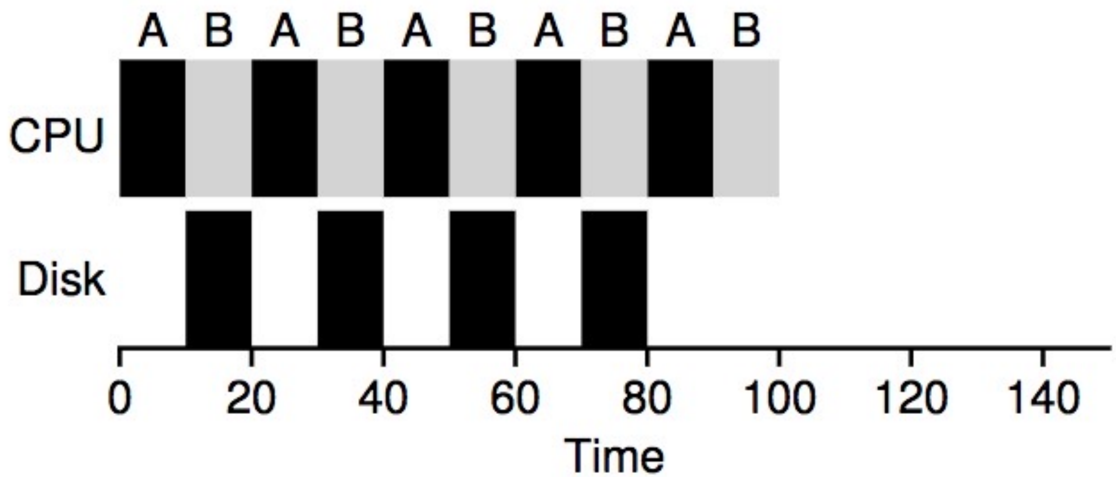
Poor use of resources

Not I/O Aware



Poor use of resources

I/O Aware (Overlap)

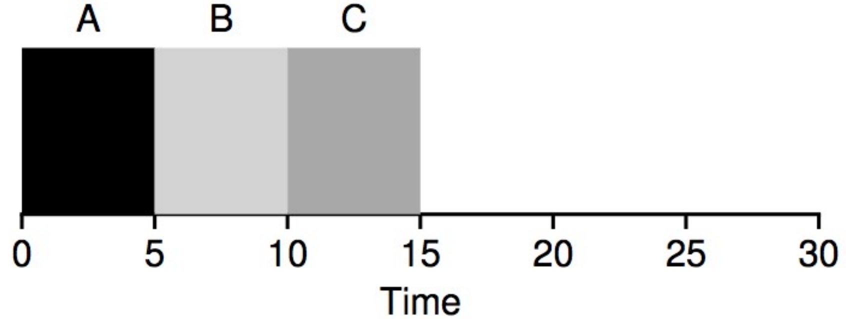


Overlap allows better use of resources!

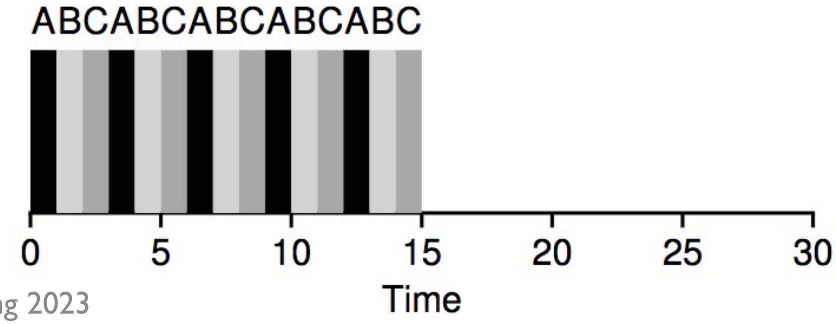
Round Robin (RR)

Process	Burst time
A	~5
B	~5
C	~5

- Each process gets a small unit of CPU time (time slice). After this time has elapsed, the process is preempted and added to the end of the ready queue
- SJF's average response time
  - (0 + 5 + 10) / 3 = 5



- RR's average response time (time slice = 1)
  - (0 + 1 + 2) / 3 = 1



Workload assumptions

1. Each job runs for the same amount of time

2. All jobs arrive at the same time

3. All jobs only use the CPU (no I/O)

4. The runtime of each job is known

Workload assumptions

1. ~~Each job runs for the same amount of time~~

2. ~~All jobs arrive at the same time~~

3. ~~All jobs only use the CPU (no I/O)~~

4. The runtime of each job is known ?

Why bother learning these low-level stuff in Data Science?

glassdoor

Search for job titles, companies, or keywords | Location

Jobs | Companies | Salaries | Careers | For Employers | Post Jobs

50 Best Jobs in America for 2022

Best Places to Work | Top CEOs | **Best Jobs** | Best Cities for Jobs | Highest Paying Jobs | Share

2022 | United States

Discover Glassdoor's Best Jobs in 2022

Using Glassdoor's unique data on jobs, salaries, and companies, we compiled a list of the 50 Best Jobs in America to help people find jobs they'll love. Each job stands out for its earning potential (median salary), job satisfaction, and job openings. Are you considering a new position? Check out this comprehensive list to see what jobs made the list this year, and view open jobs at companies across the country.

	Job Title	Median Base Salary	Job Satisfaction	Job Openings	
#1	Enterprise Architect	$144,997	4.1/5	14,021	View Jobs
#2	Full Stack Engineer	$101,794	4.3/5	11,252	View Jobs
#3	Data Scientist	$120,000	4.1/5	10,071	View Jobs
#4	Devops Engineer	$120,095	4.2/5	8,548	View Jobs
#5	Strategy Manager	$140,000	4.2/5	6,977	View Jobs
#6	Machine Learning Engineer	$130,489	4.3/5	6,801	View Jobs

Why bother learning these low-level stuff in Data Science?

- Basics of computer organization
  - Digital representation of data
  - Machine architecture (ISA)
  - CPU and memory hierarchy

- Basics of operating systems
  - CPU management
  - Memory management
  - File system and data management