

[544] Intro to Big Data Systems

Tyler Caraza-Harter

Outline

Course Overview

- Introductions
- Main sites: tylercaraza-harter.com, Canvas, GitHub
- Other tools: Email, TopHat, Piazza, GitHub classroom

Resources

- Overview
- Compute
- Memory
- Storage
- Network

Deployment

Introductions

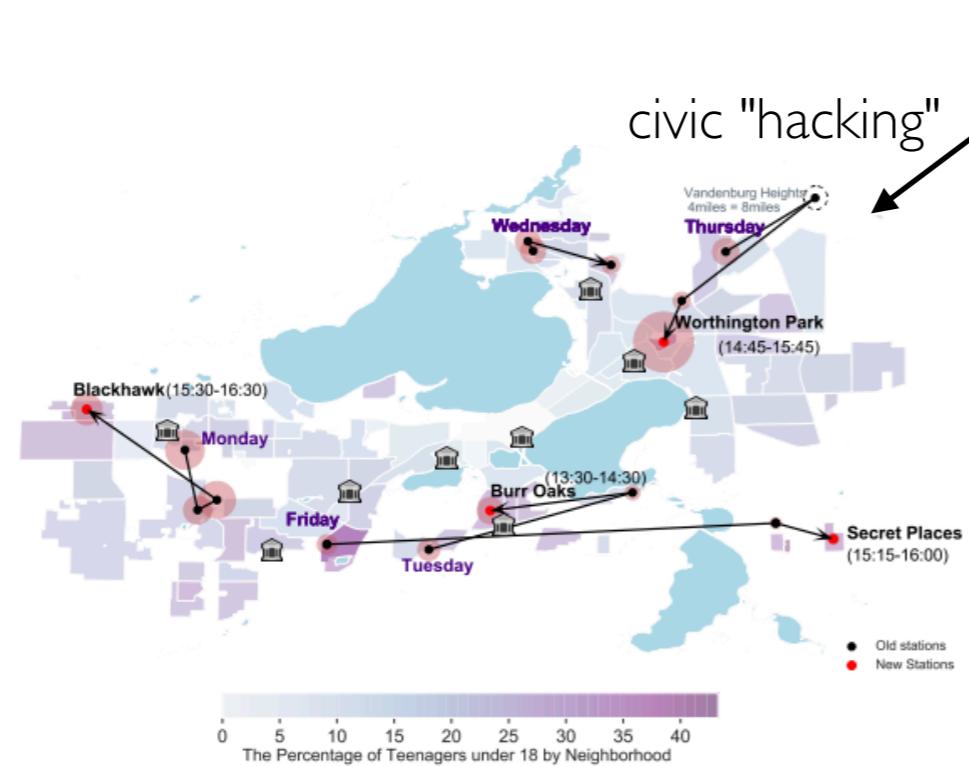
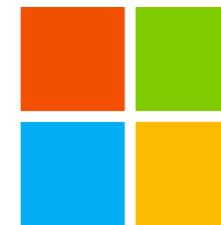
Tyler Caraza-Harter

- Long time Badger
- Email: tharter@wisc.edu
- Just call me “Tyler” (he/him)



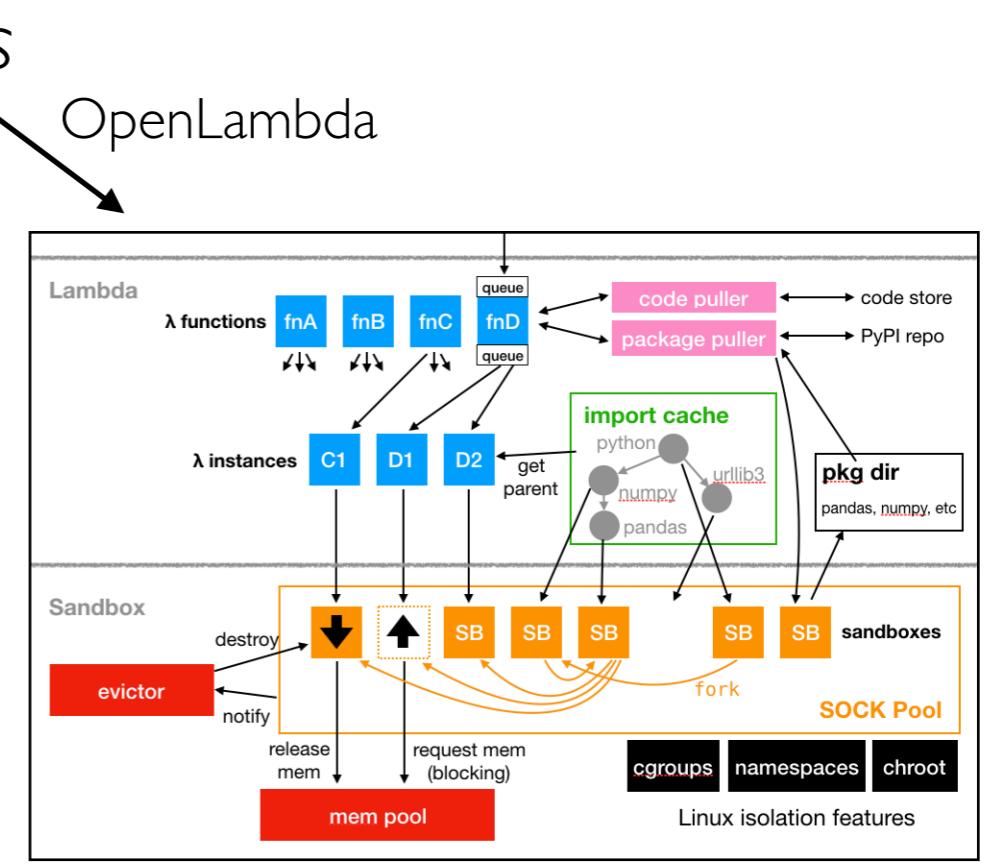
Industry experience

- Worked at Microsoft on SQL Server and Cloud
- Other internships/collaborations:
Qualcomm, Google, Facebook, Tintri



Plot by [Zishan Bai & Dingyi Zhou](#) (previous students)

More: <https://wisc-ds-projects.github.io>



Who are You?

Year in school?

- 1st year? 2nd? Junior/senior? Grad student?

Area of study

- Natural science, social science, engineering, business, statistics, data science, other?

What CS courses have people taken before?

- CS 220/301? CS 200? CS 300? CS 354?

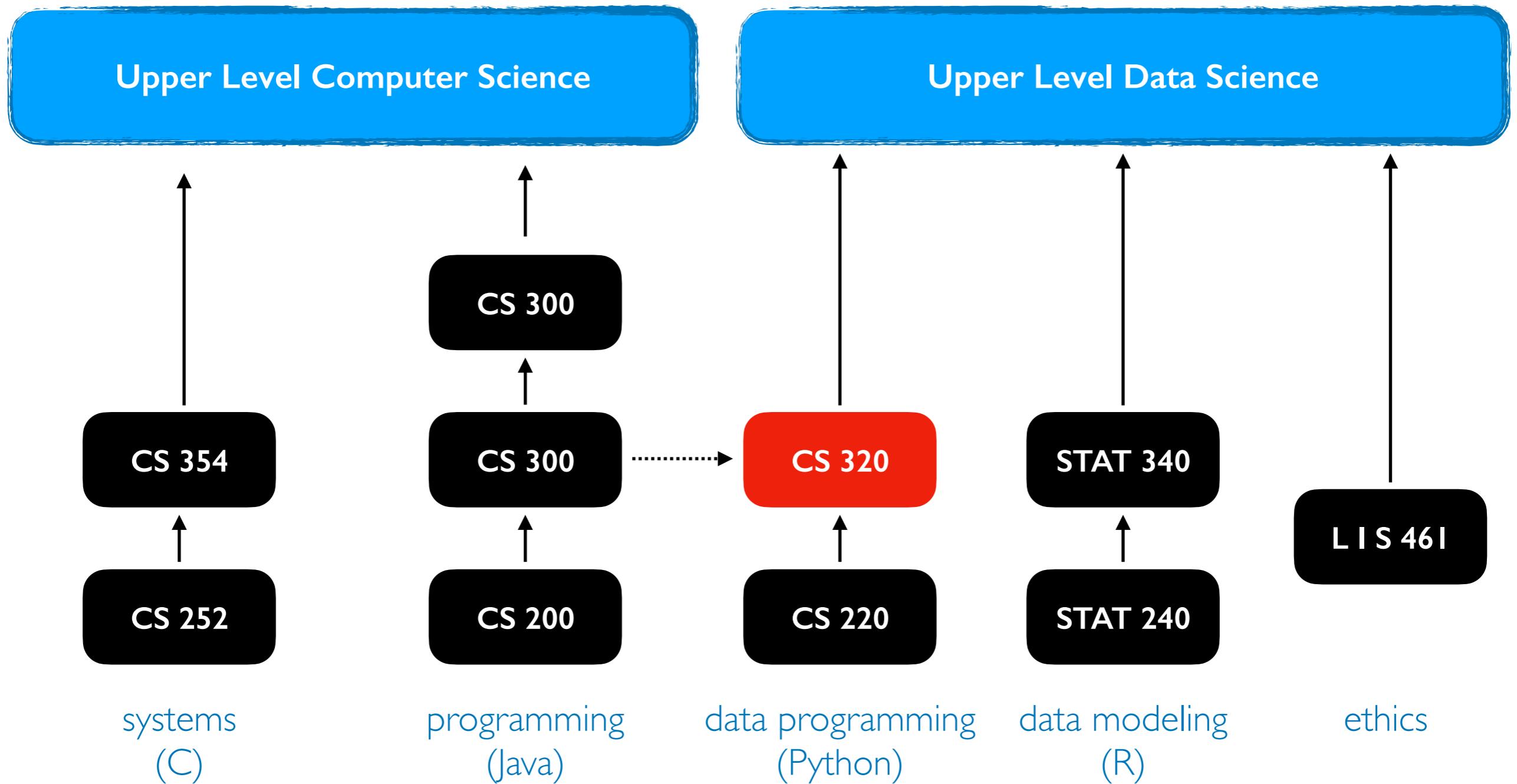
Please fill this form (**due today**):

https://docs.google.com/forms/d/e/1FAIpQLSfz7K0cY2-VGCtxE4TQ-zkcbcWTtzyLZQXCrgLyp6EfU2jDg/viewform?usp=sf_link.

Why?

- Help me get to know you
- Get participation credit
- Group formation

Related courses



PI (Project I) will help 300-to-320 students pickup Python.

Outline

Course Overview

- Introductions
- Main sites: tylercaraza-harter.com, Canvas, GitHub
- Other tools: Email, TopHat, Piazza, GitHub classroom

Resources

- [Overview](#)
- Compute
- Memory
- Storage
- Network

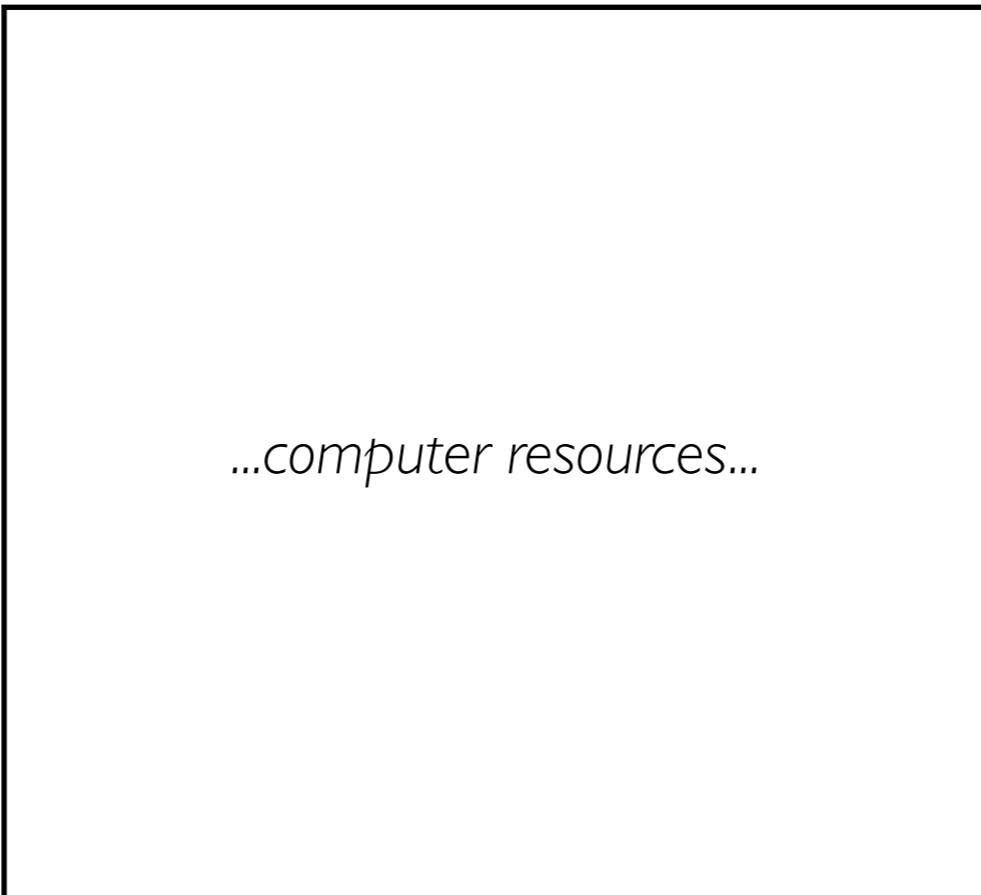
Deployment

What are "systems"?

Systems: software for managing computer **resources**

Other kinds of software (analysis code, applications) rely on systems.

a computer:



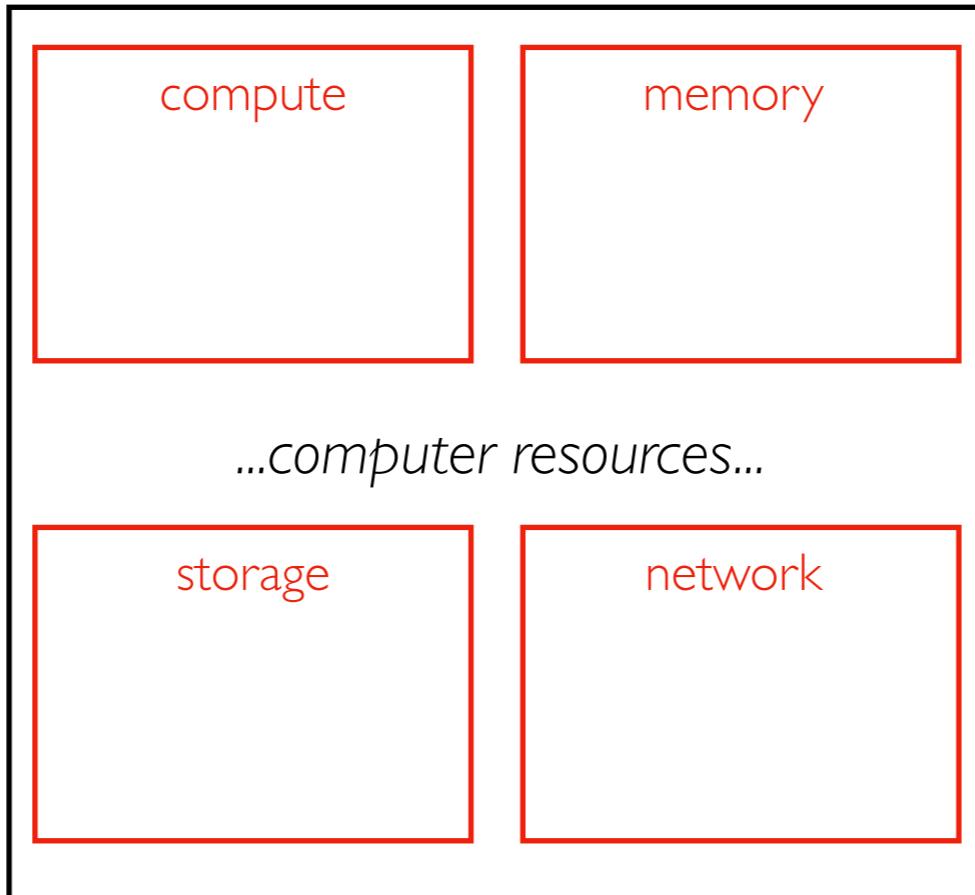
...computer resources...

Categories of resources

Systems: software for managing computer **resources**

Other kinds of software (analysis code, applications) rely on systems.

a computer:



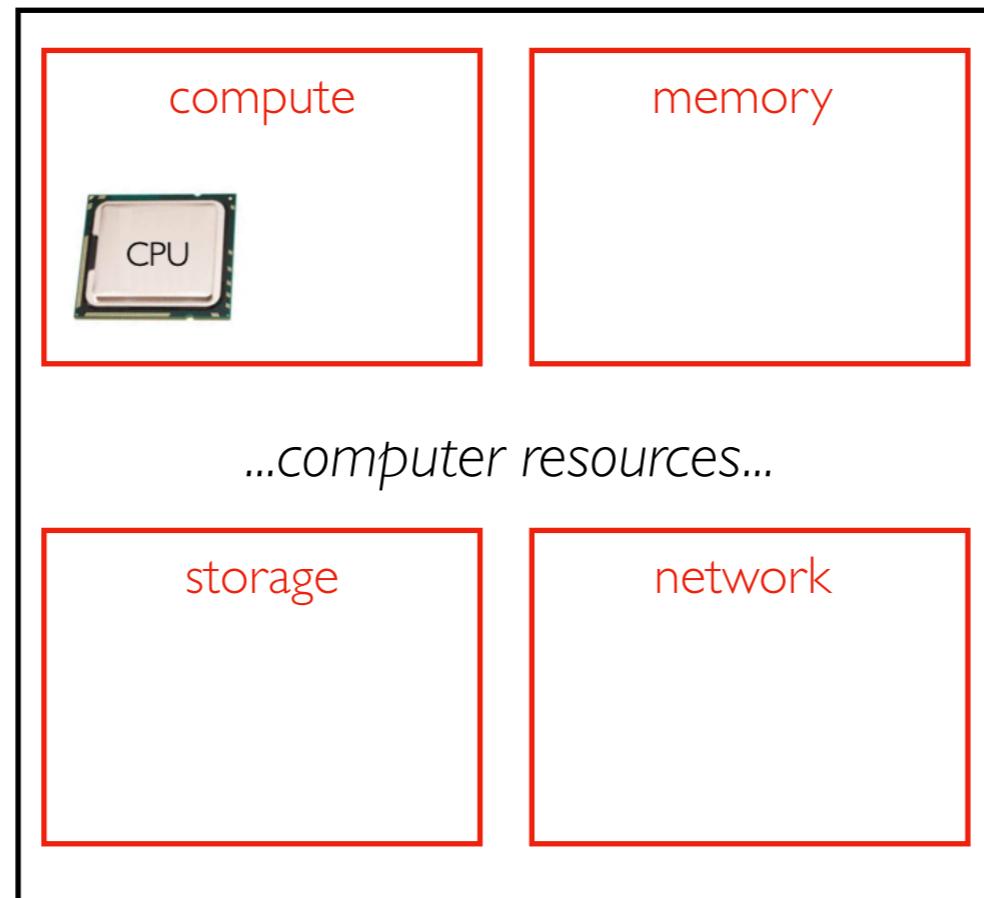
Categories of resources

Systems: software for managing computer **resources**

Other kinds of software (analysis code, applications) rely on systems.

a computer:

computational resources
execute code



central processing unit (CPU)

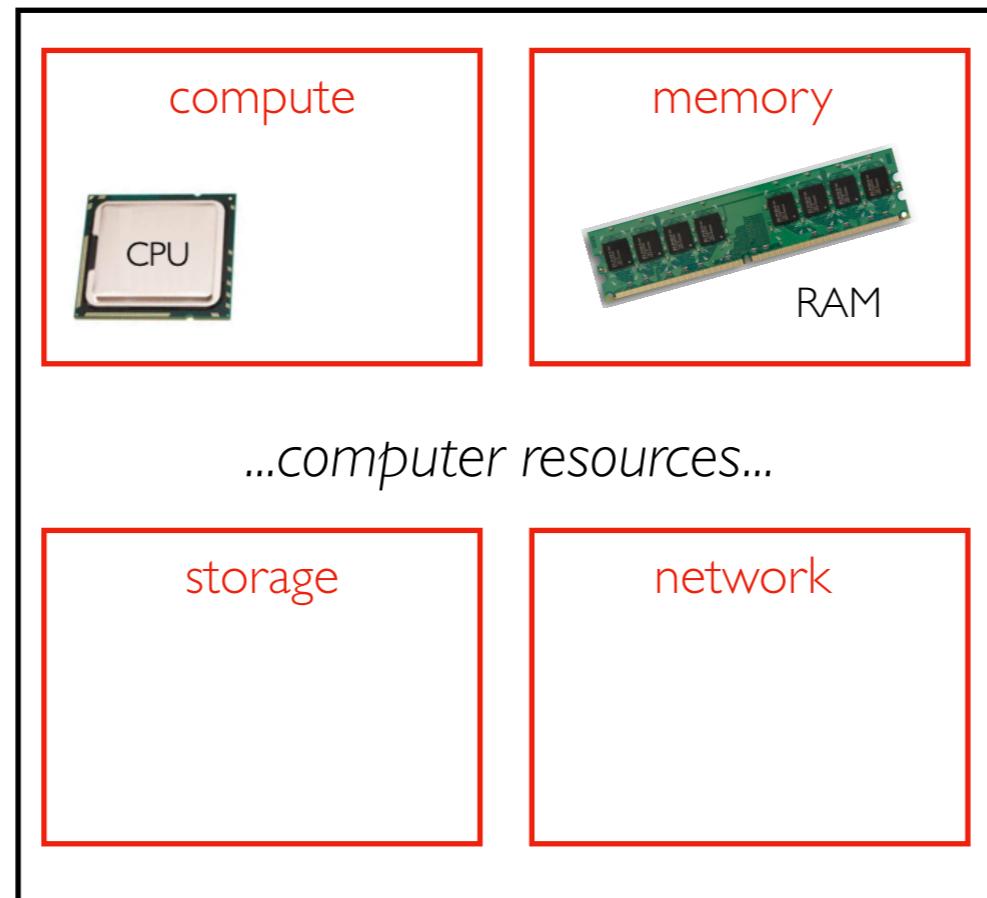
Categories of resources

Systems: software for managing computer **resources**

Other kinds of software (analysis code, applications) rely on systems.

a computer:

computational resources
execute code



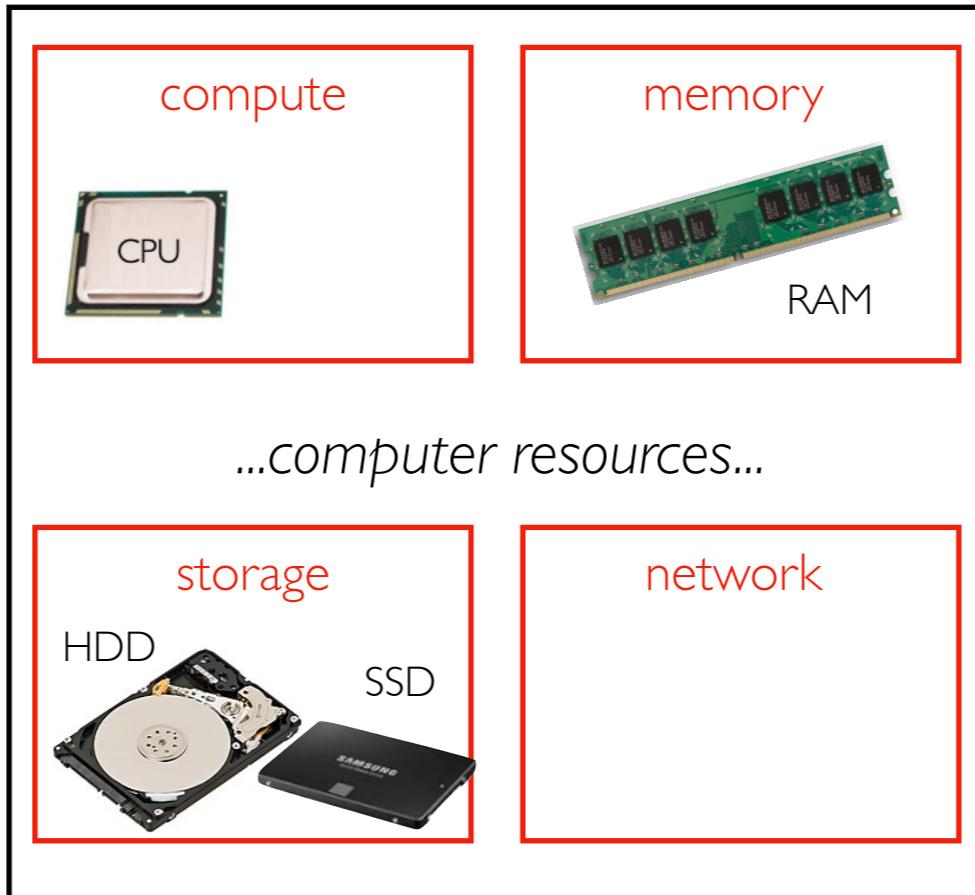
random-access memory (RAM)

Categories of resources

Systems: software for managing computer **resources**

Other kinds of software (analysis code, applications) rely on systems.

a computer:



computational resources
execute code

storage holds
long-term data

memory holds data
for active usage

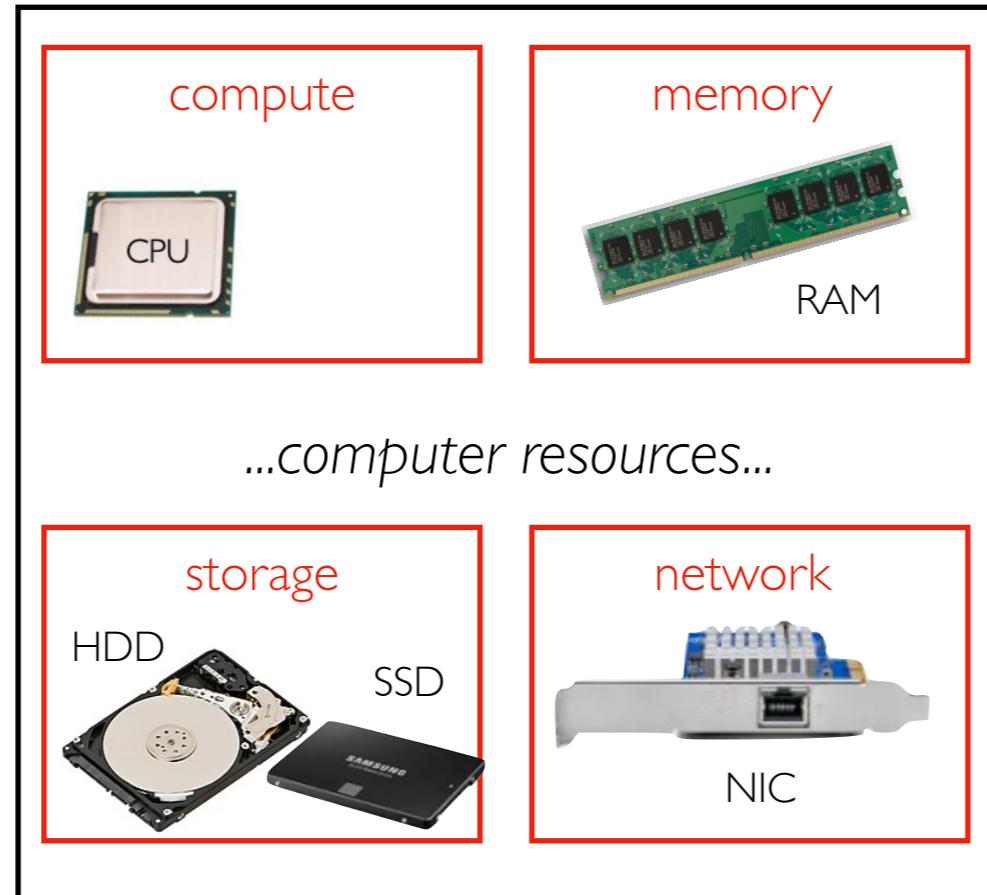
hard disk drive (HDD), solid-state disk (SSD)

Categories of resources

Systems: software for managing computer **resources**

Other kinds of software (analysis code, applications) rely on systems.

a computer:



computational resources
execute code

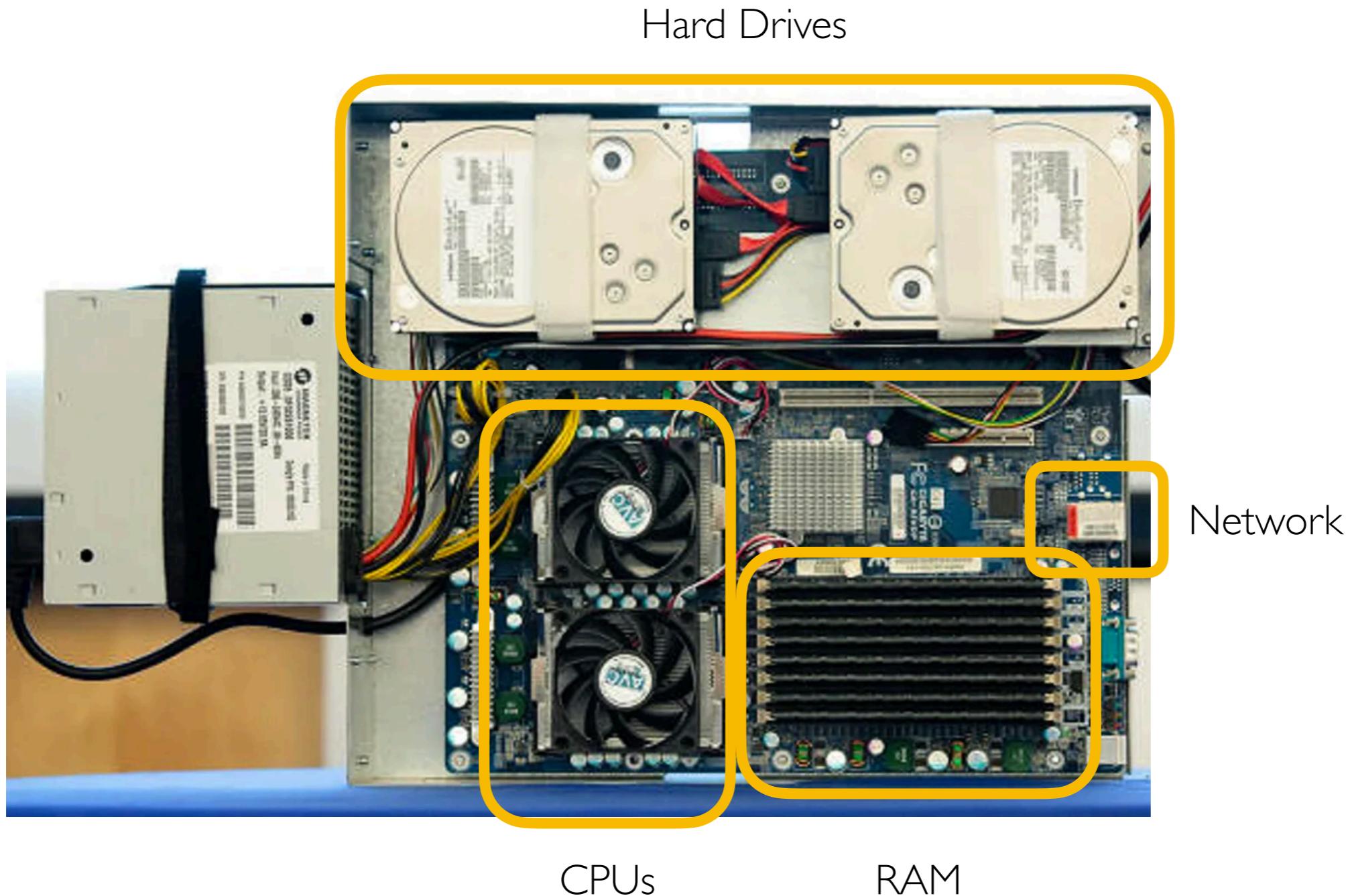
storage holds
long-term data

memory holds data
for active usage

network provides
communication between
computers

network interface card (NIC)

A real server



Big Data

Potential problems as datasets grow

- might run too slowly
- might not be able to run at all (for example, not enough memory)

Solutions:

- ????

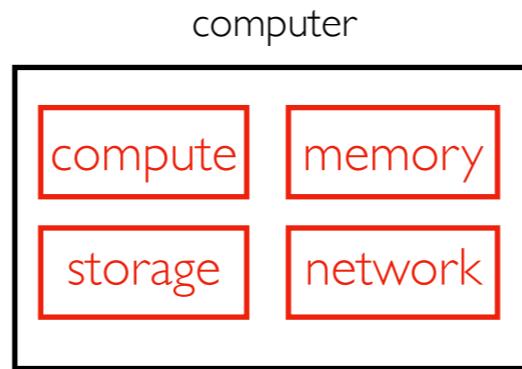
Big Data

Potential problems as datasets grow

- might run too slowly
- might not be able to run at all (for example, not enough memory)

Solutions:

- more efficient code
- use more resources



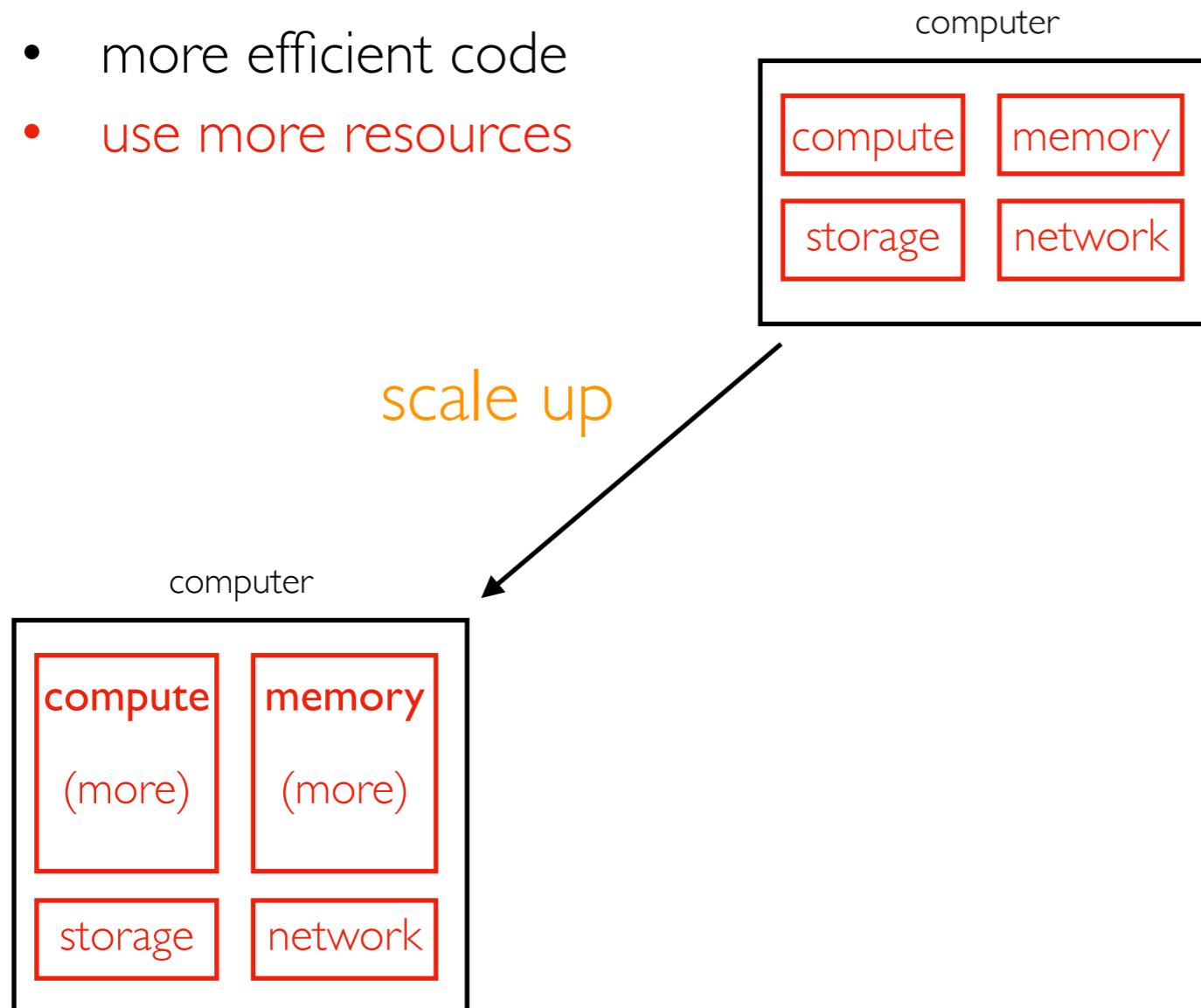
Big Data

Potential problems as datasets grow

- might run too slowly
- might not be able to run at all (for example, not enough memory)

Solutions:

- more efficient code
- use more resources



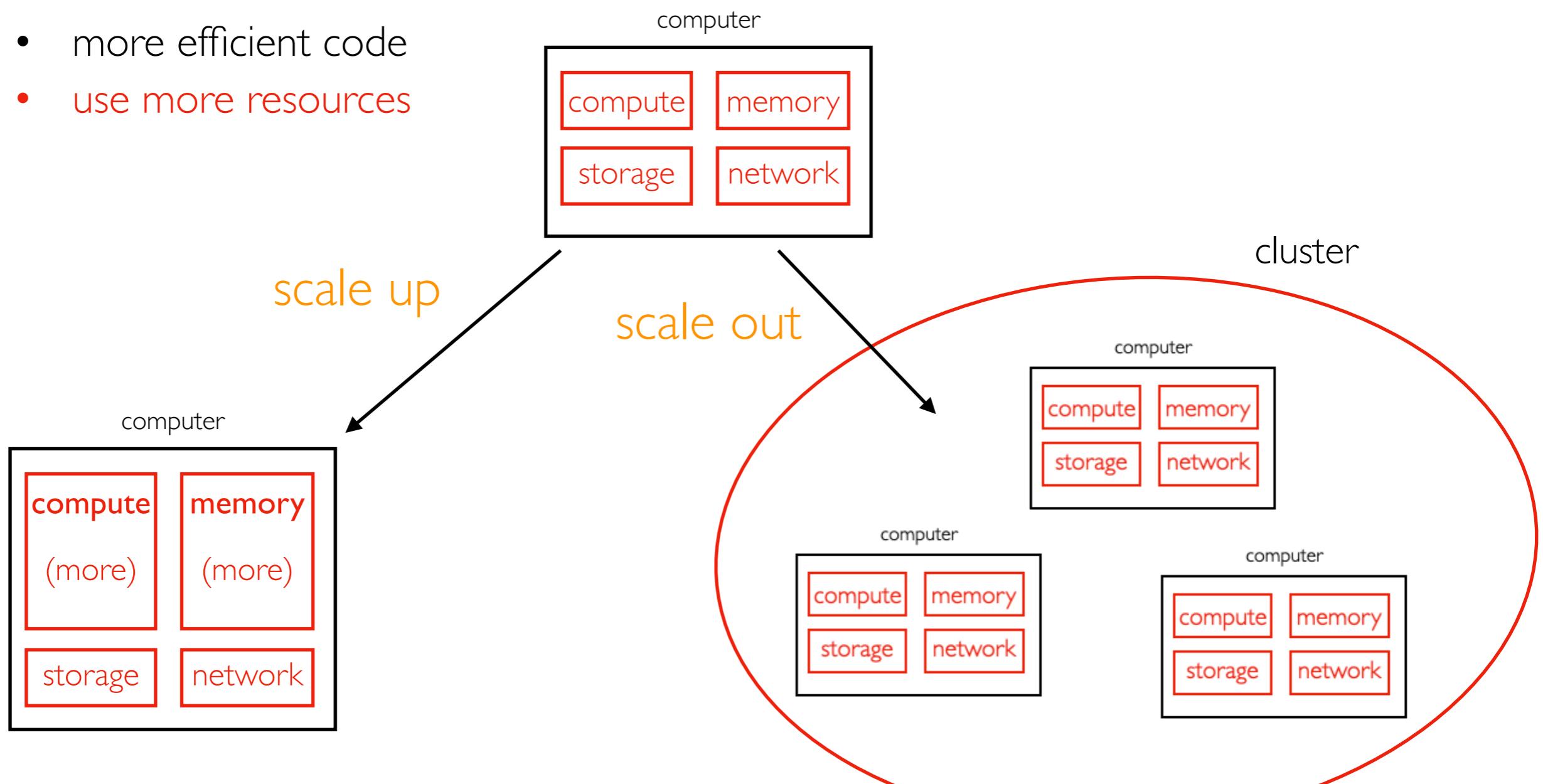
Big Data

Potential problems as datasets grow

- might run too slowly
- might not be able to run at all (for example, not enough memory)

Solutions:

- more efficient code
- use more resources



Outline

Course Overview

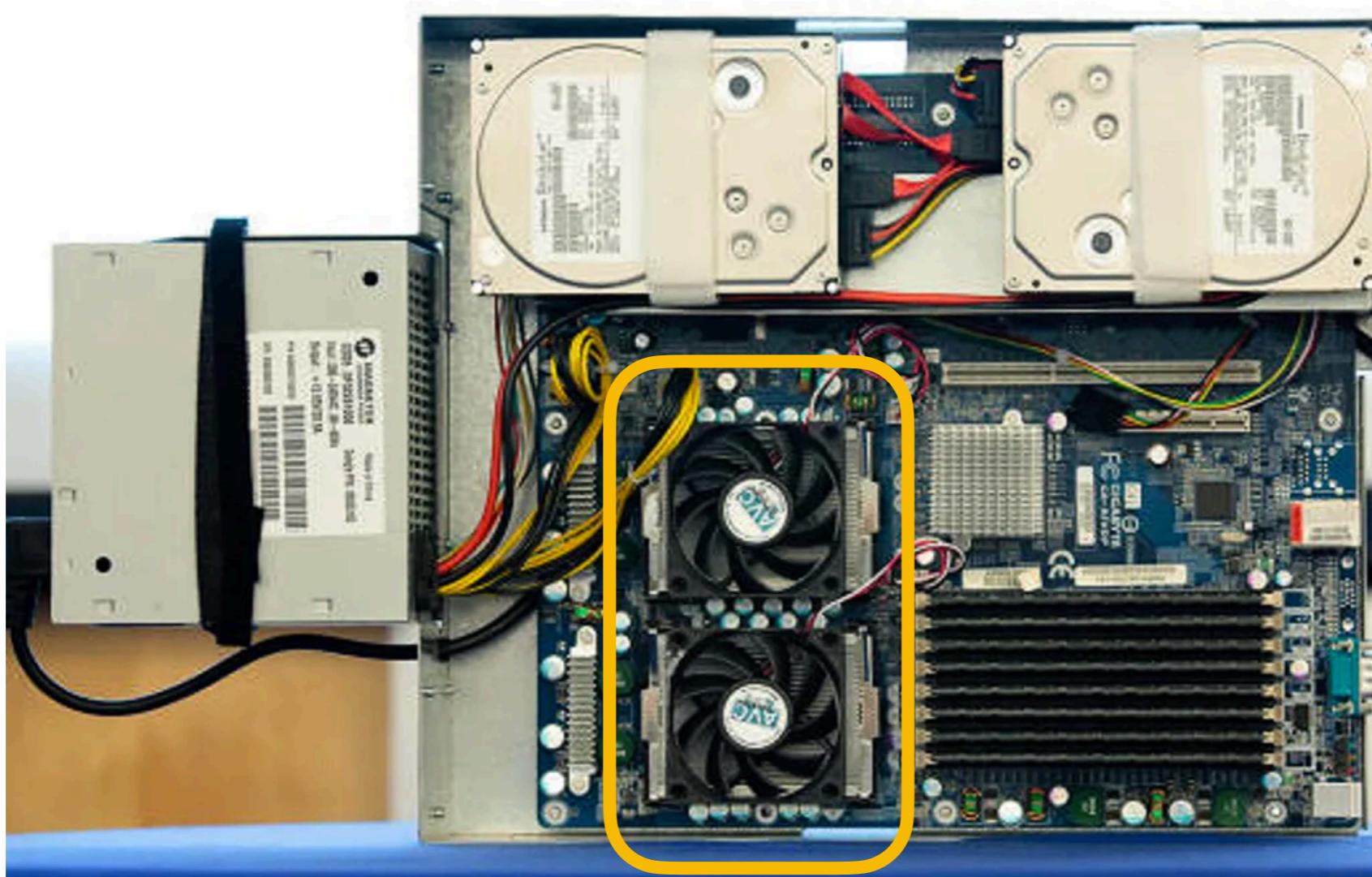
- Introductions
- Main sites: tylercaraza-harter.com, Canvas, GitHub
- Other tools: Email, TopHat, Piazza, GitHub classroom

Resources

- Overview
- Compute
- Memory
- Storage
- Network

Deployment

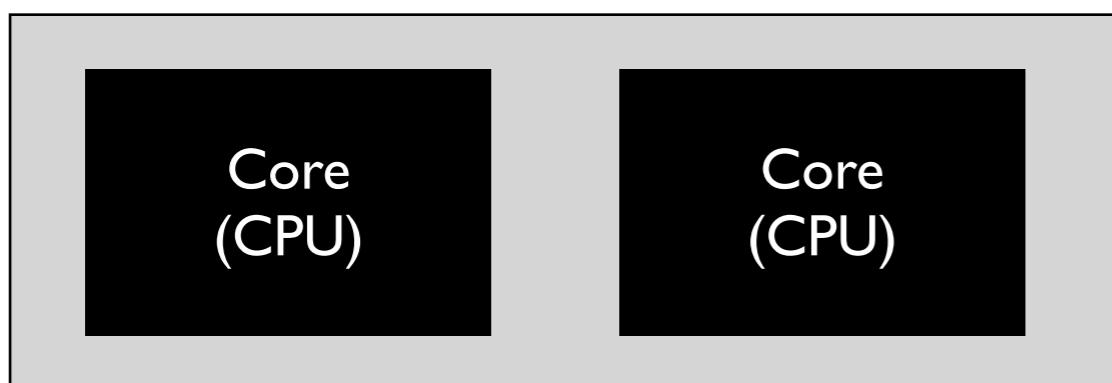
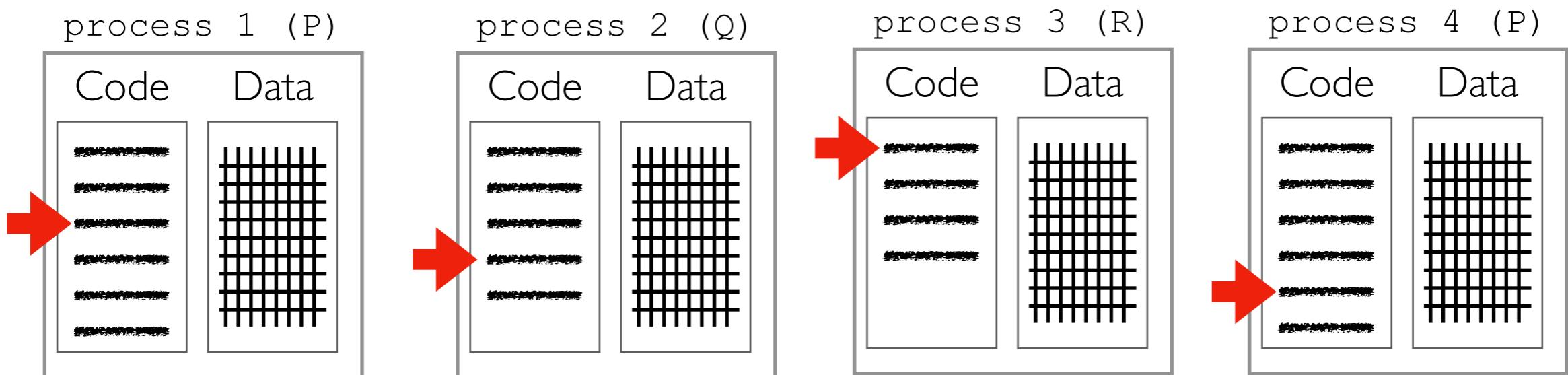
Compute



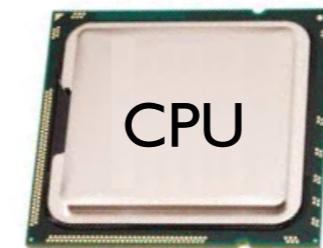
Some computers have multiple **CPUs**. Modern CPUs typically have multiple **cores**. Each core works like a CPU and runs programs by executing instructions.

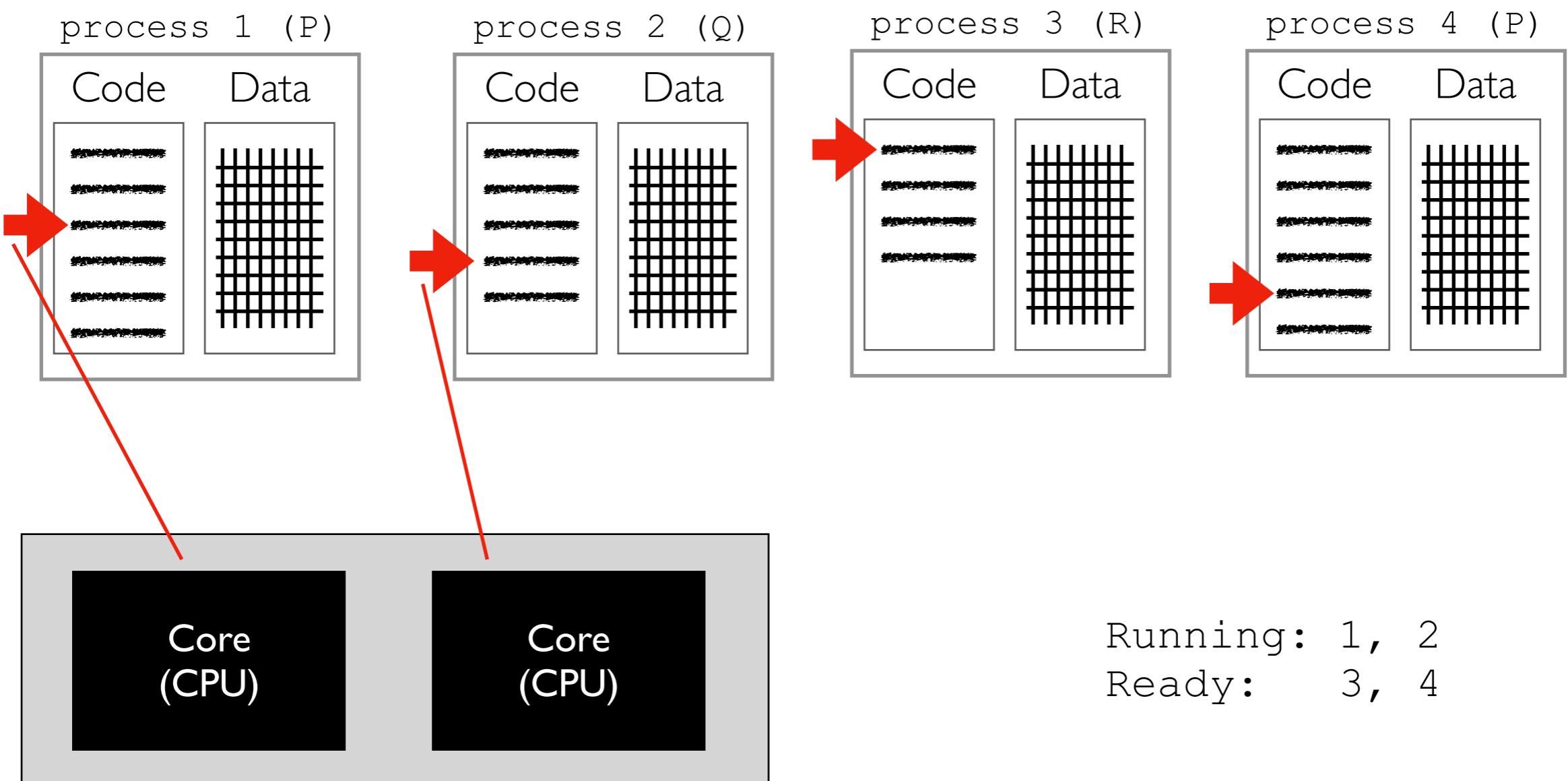
How do cores run machine code?

the operating system "schedules" processes on cores
(decides when they get to run)

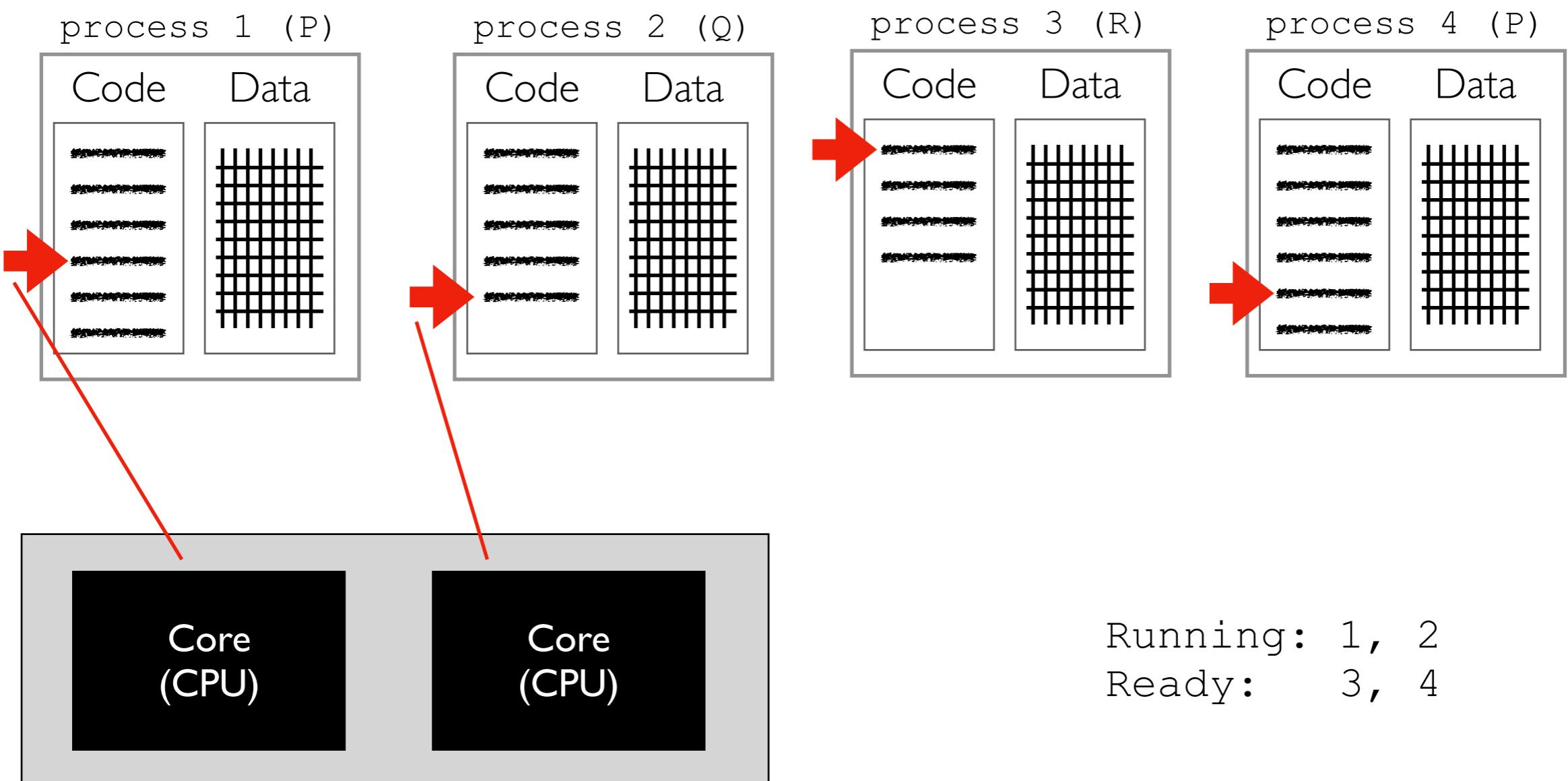


Multi-Core Processor (CPU)

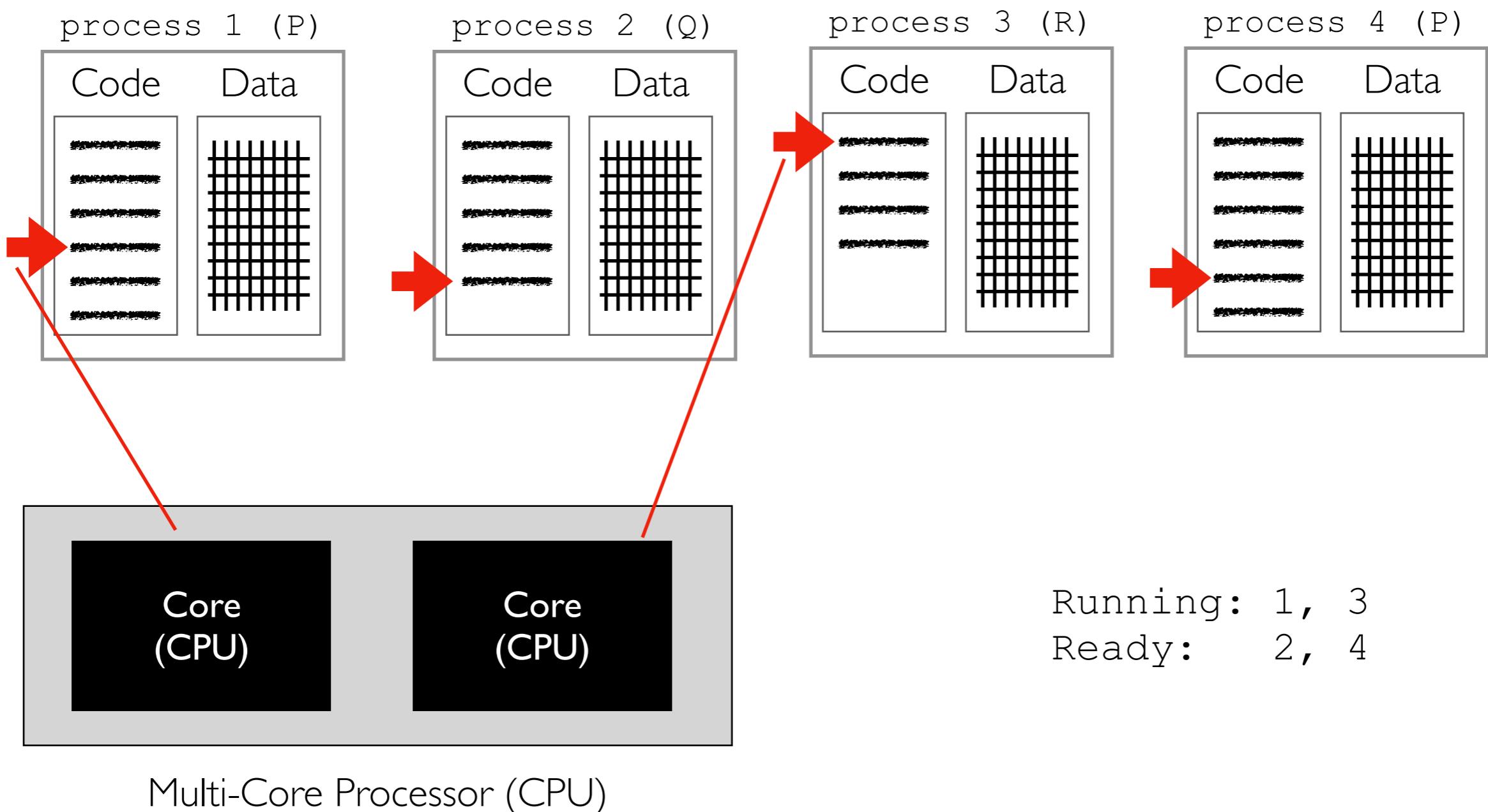


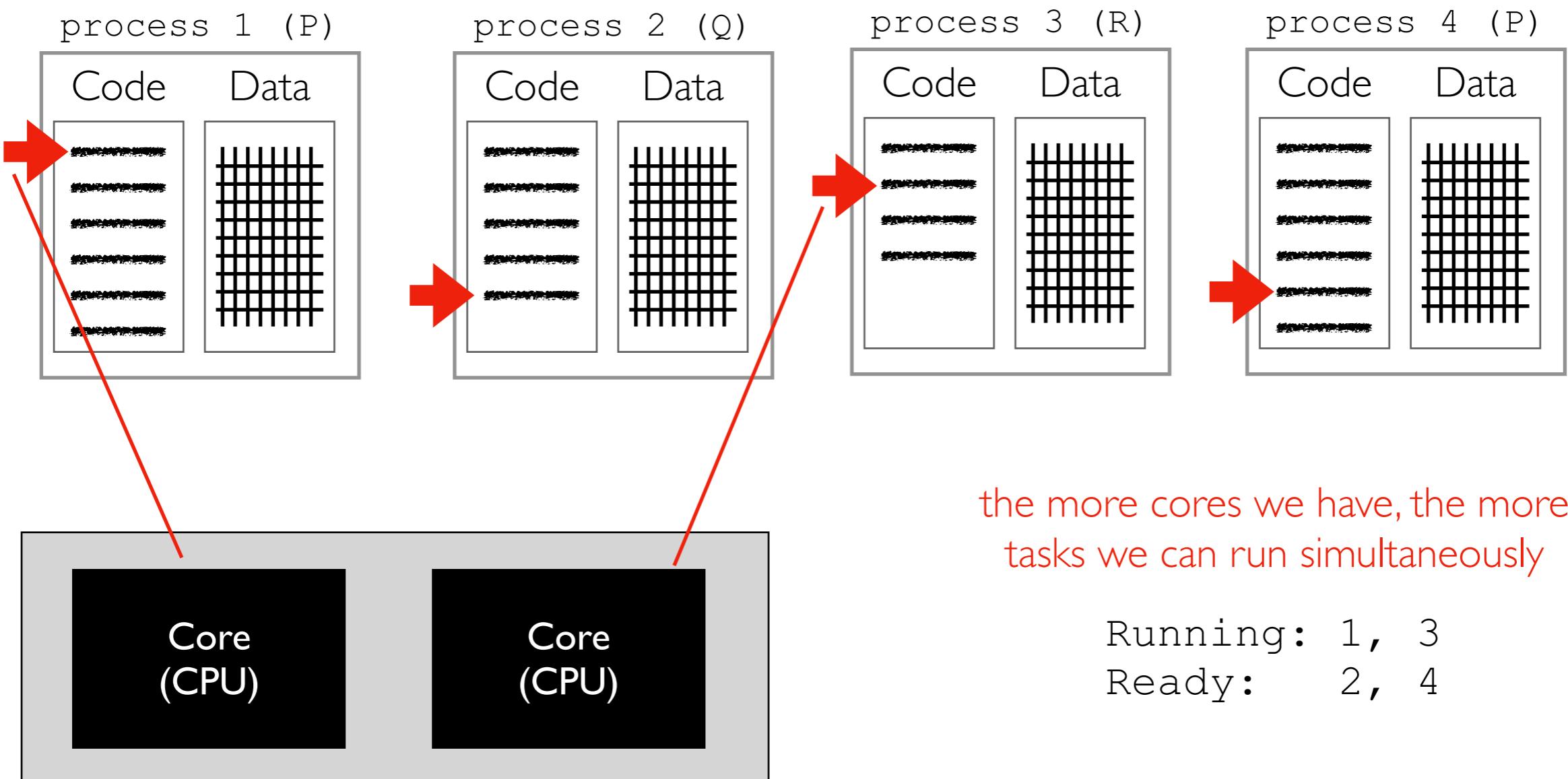


Multi-Core Processor (CPU)



Multi-Core Processor (CPU)

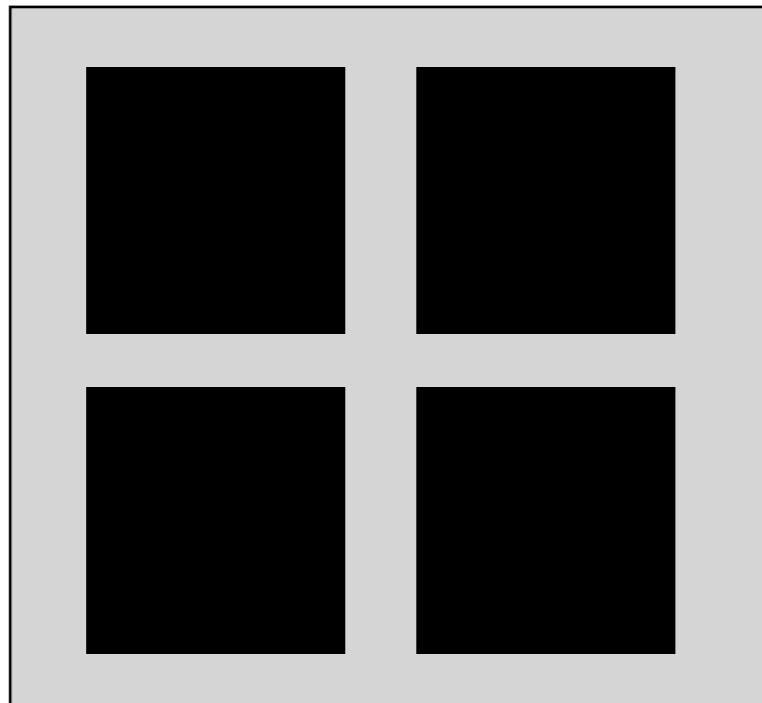
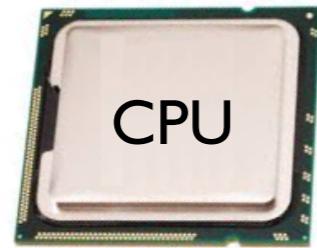




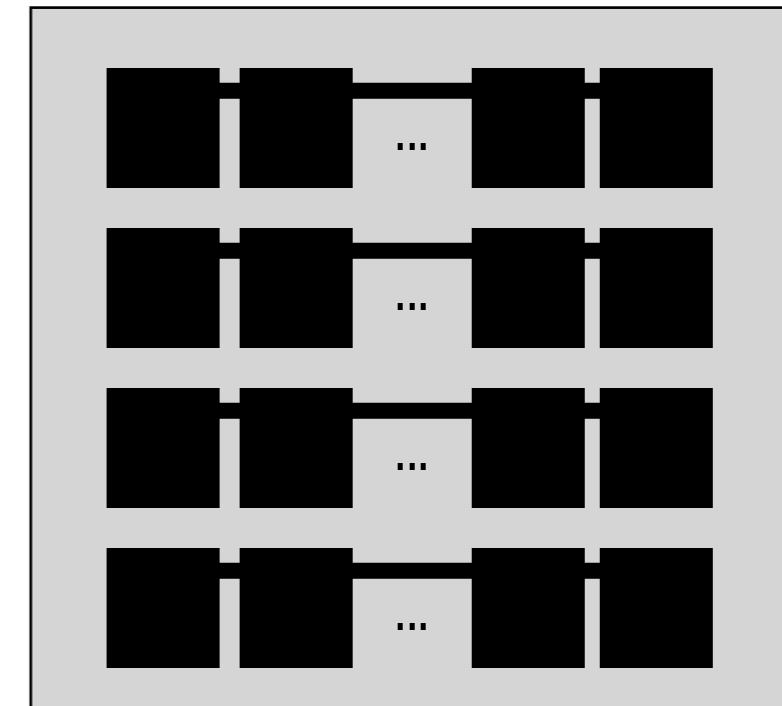
Multi-Core Processor (CPU)

What are alternatives to CPUs for compute?

GPUs (graphical processing units) are an alternative compute resource.



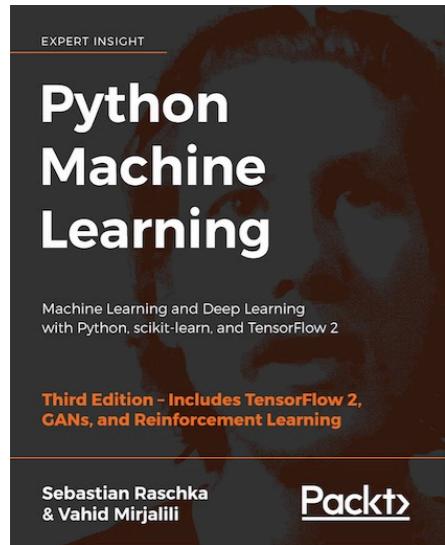
few cores that are fast,
flexible, independent



many cores that are slow,
float-optimized, coordinated

GPU vs. CPU: Cost Comparison

The GPU is 30% cheaper but 28x faster at floating-point operations!



Specifications	Intel® Core™ i7-6900K Processor Extreme Ed.	NVIDIA GeForce® GTX™ 1080 Ti
Base Clock Frequency	3.2 GHz	< 1.5 GHz
Cores	8	3584
Memory Bandwidth	64 GB/s	484 GB/s
Floating-Point Calculations	409 GFLOPS	11300 GFLOPS
Cost	~ \$1000.00	~ \$700.00

<https://sebastianraschka.com/books.html>

Resource metric: **FLOPS** (floating-point operations per second)

- floating-point ops: add, mult, etc (how to weight?)
- prefixes: K (thousand), M (million), G (billion), T (trillion)
 - how many TFLOPS does the above GPU provide?

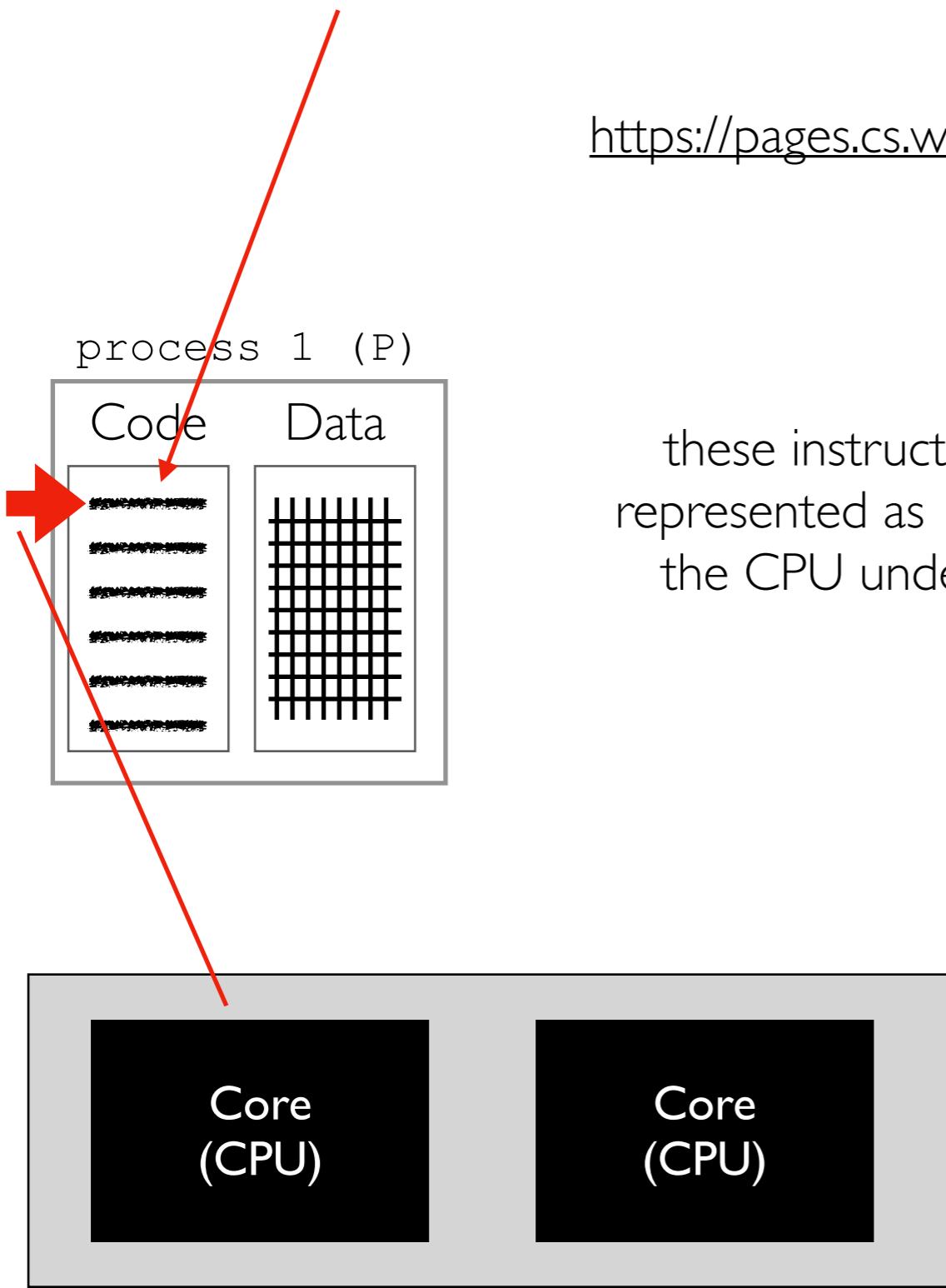
PyTorch Python package

- lets you readily use GPUs; we'll use this for project 1

how do we bridge the gap between "high level"
code (Python/Java/etc) and machine code?

Note: we'll primarily write Python this semester, but it helps to explore this in general to understand how systems like Spark work (which is written in Scala and uses the Java Virtual Machine)

these instructions are in "machine code"
that the CPU can understand



<https://pages.cs.wisc.edu/~deppeler/cs354/reference/x86-cheat-sheet.pdf>

these instructions are
represented as 1's and 0's
the CPU understands

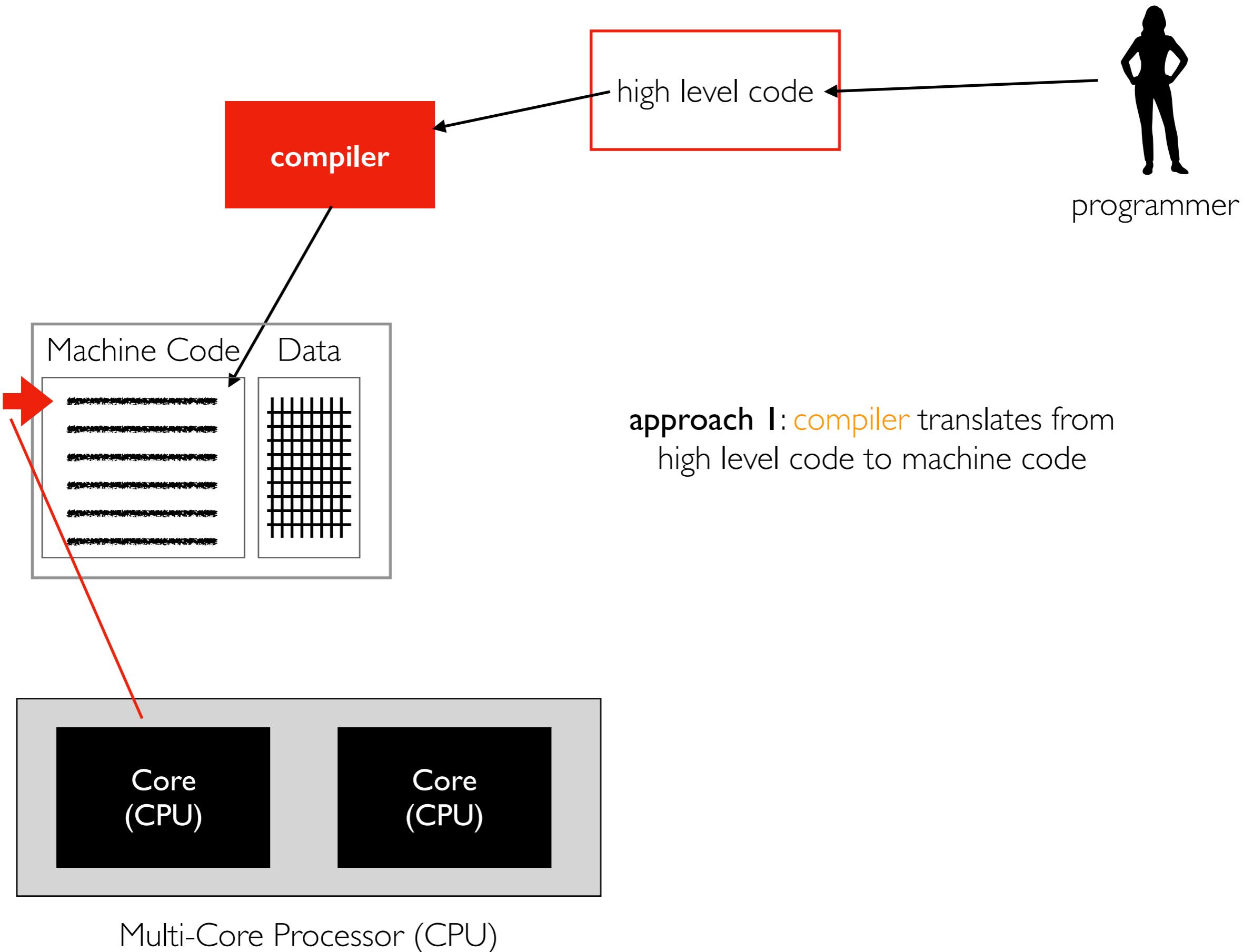
```
arithmetic
two operand instructions
addl src,dst    dst = dst + src
subl src,dst    dst = dst - src
imull src,dst   dst = dst * src
sall src,dst    dst = dst << src (aka shll)
sarl src,dst    dst = dst >> src (arith)
shrl src,dst    dst = dst >> src (logical)
xorl src,dst    dst = dst ^ src
andl src,dst    dst = dst & src
orl src,dst     dst = dst | src

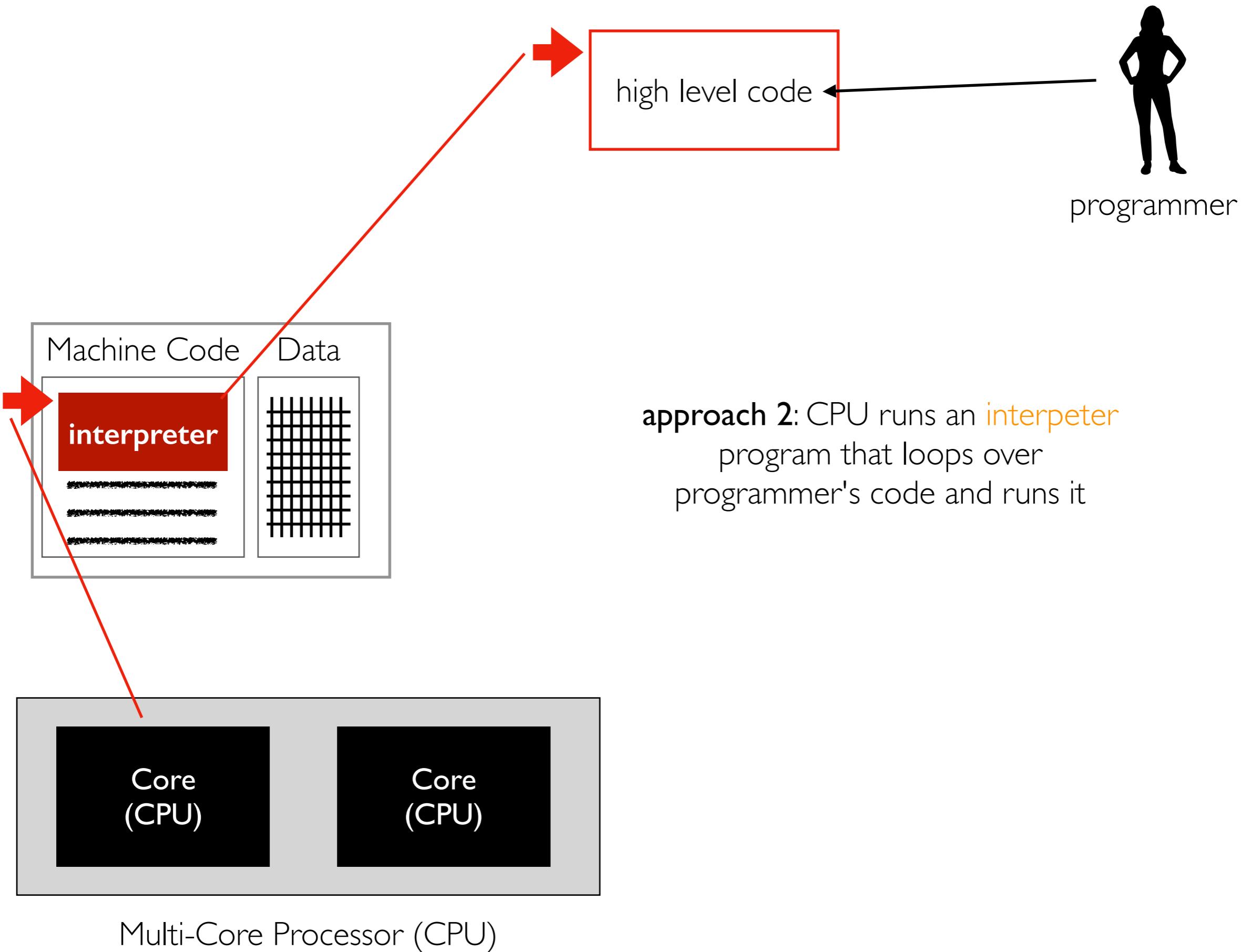
one operand instructions
incl dst        dst = dst + 1
decl dst        dst = dst - 1
negl dst        dst = -dst
notl dst        dst = ~dst

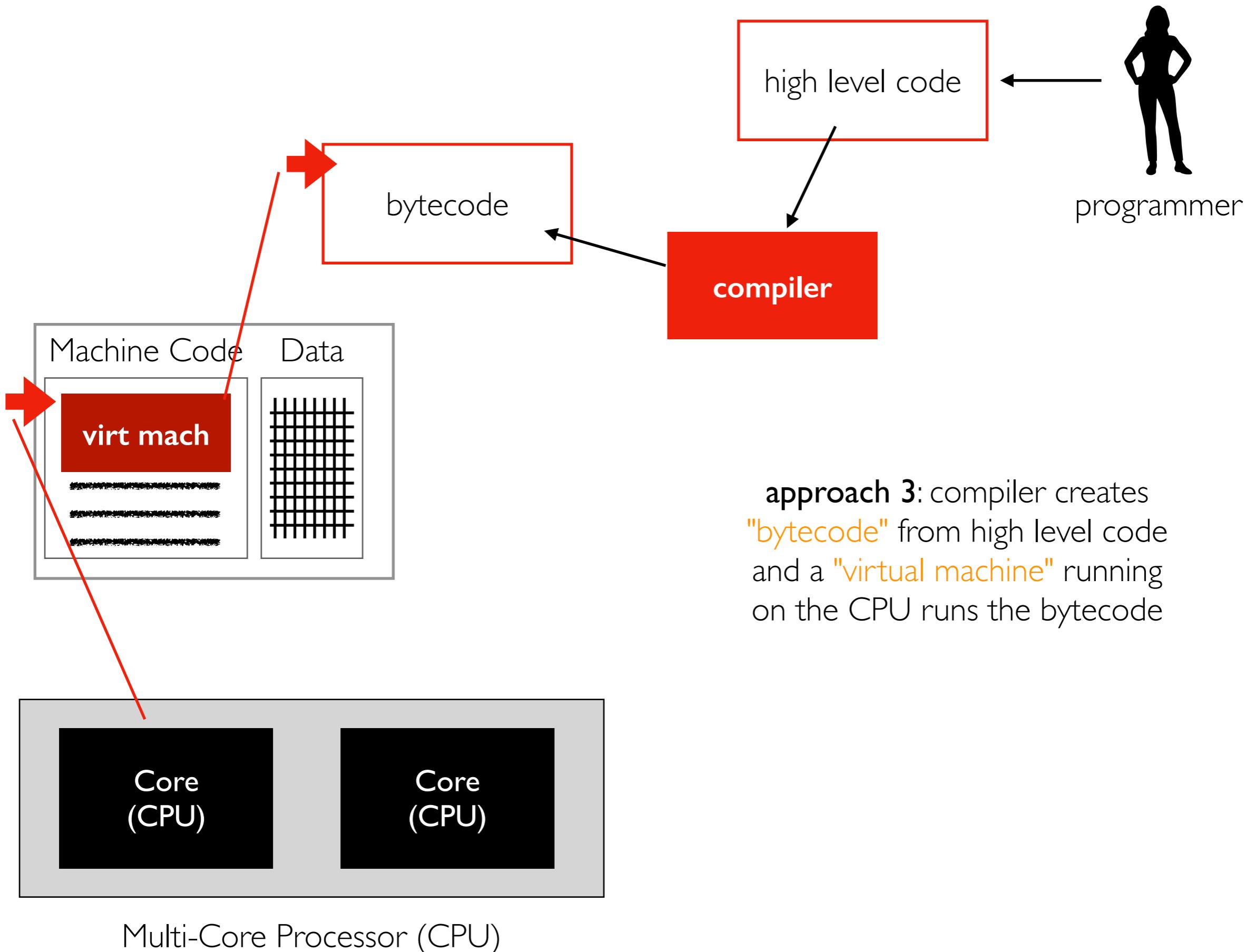
arithmetic ops set CCs implicitly
cf=1 if carry out from msb
zf=1 if dst==0,
sf=1 if dst < 0 (signed)
of=1 if two's complement
(signed) under/overflow
```

how do we bridge the gap between "high level"
code (Python/Java/etc) and machine code?

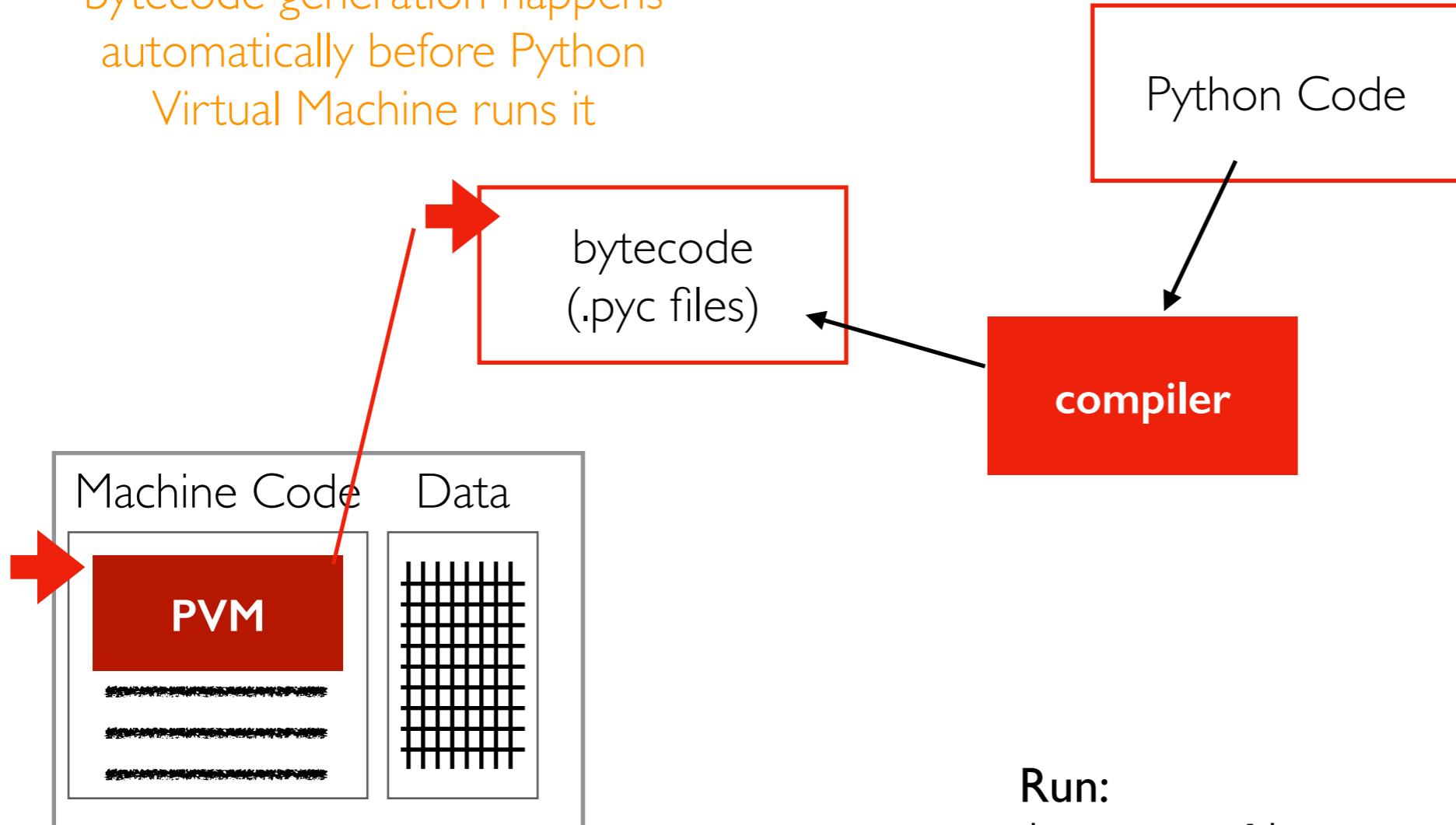
Multi-Core Processor (CPU)





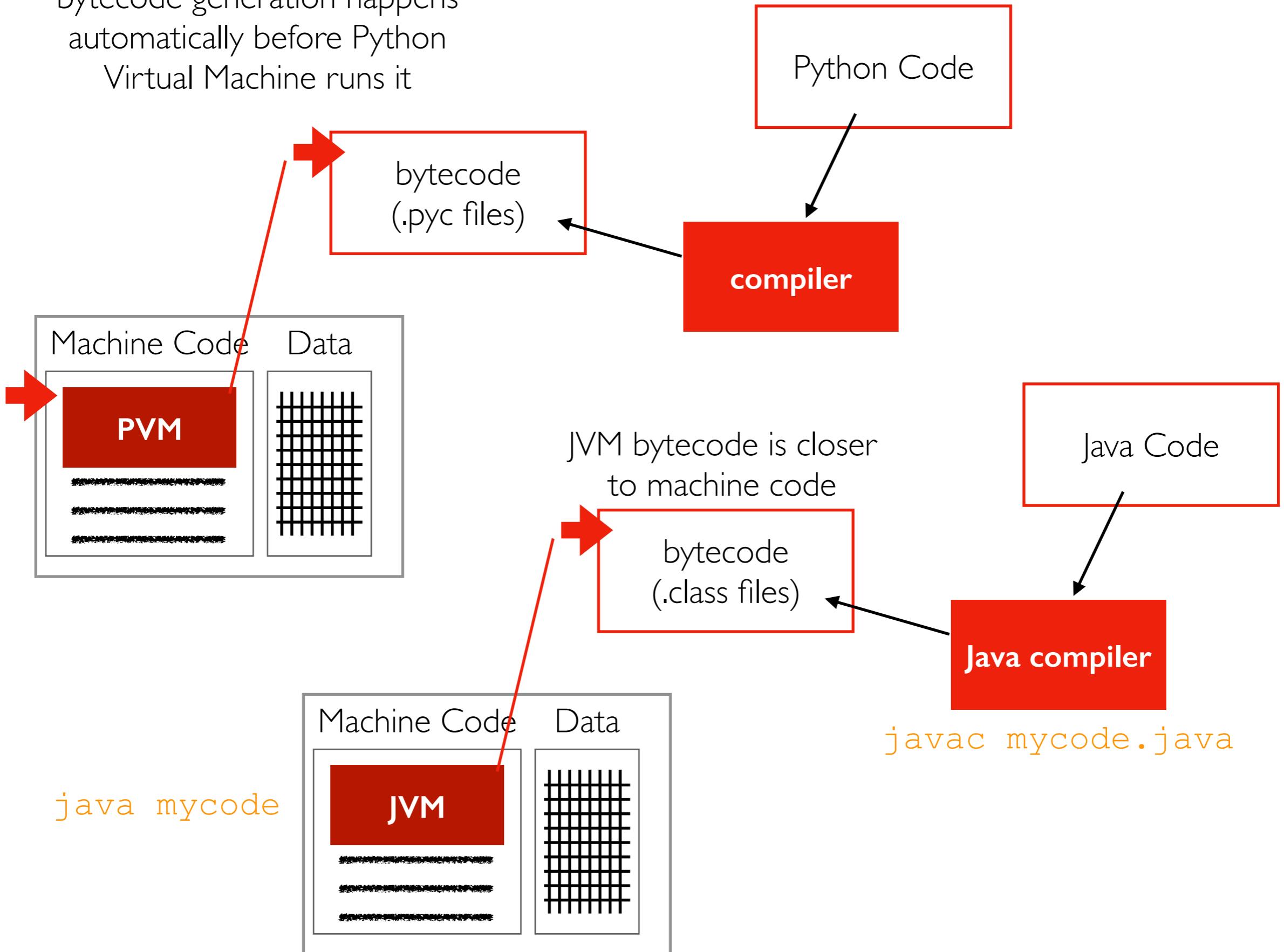


when you run "python3 ..."
bytecode generation happens
automatically before Python
Virtual Machine runs it

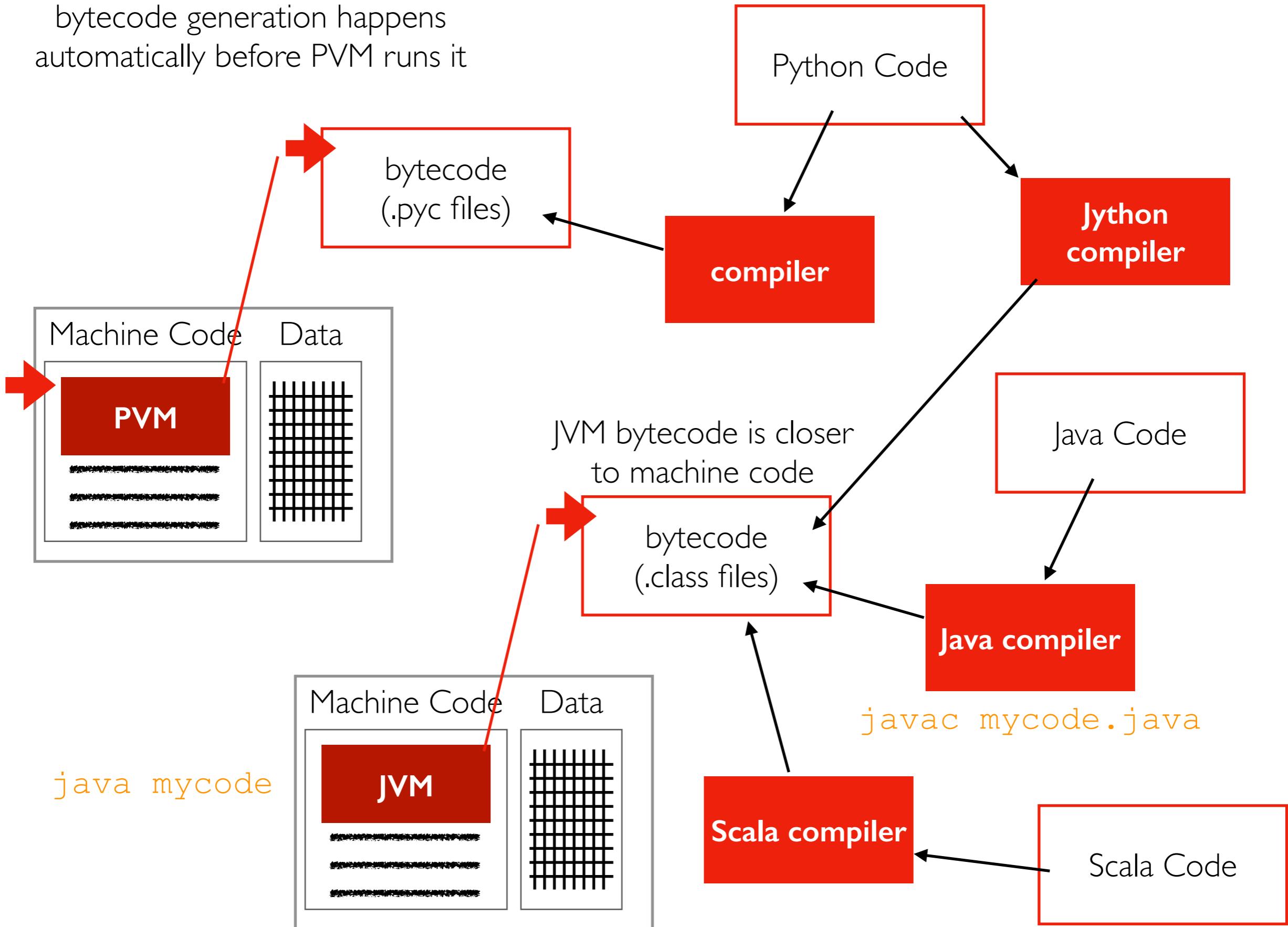


Run:
`import dis
dis.dis("y = x + 2")`

when you run "python3 ..."
bytecode generation happens
automatically before Python
Virtual Machine runs it



when you run "python3 ..."
 bytecode generation happens
 automatically before PVM runs it



Outline

Course Overview

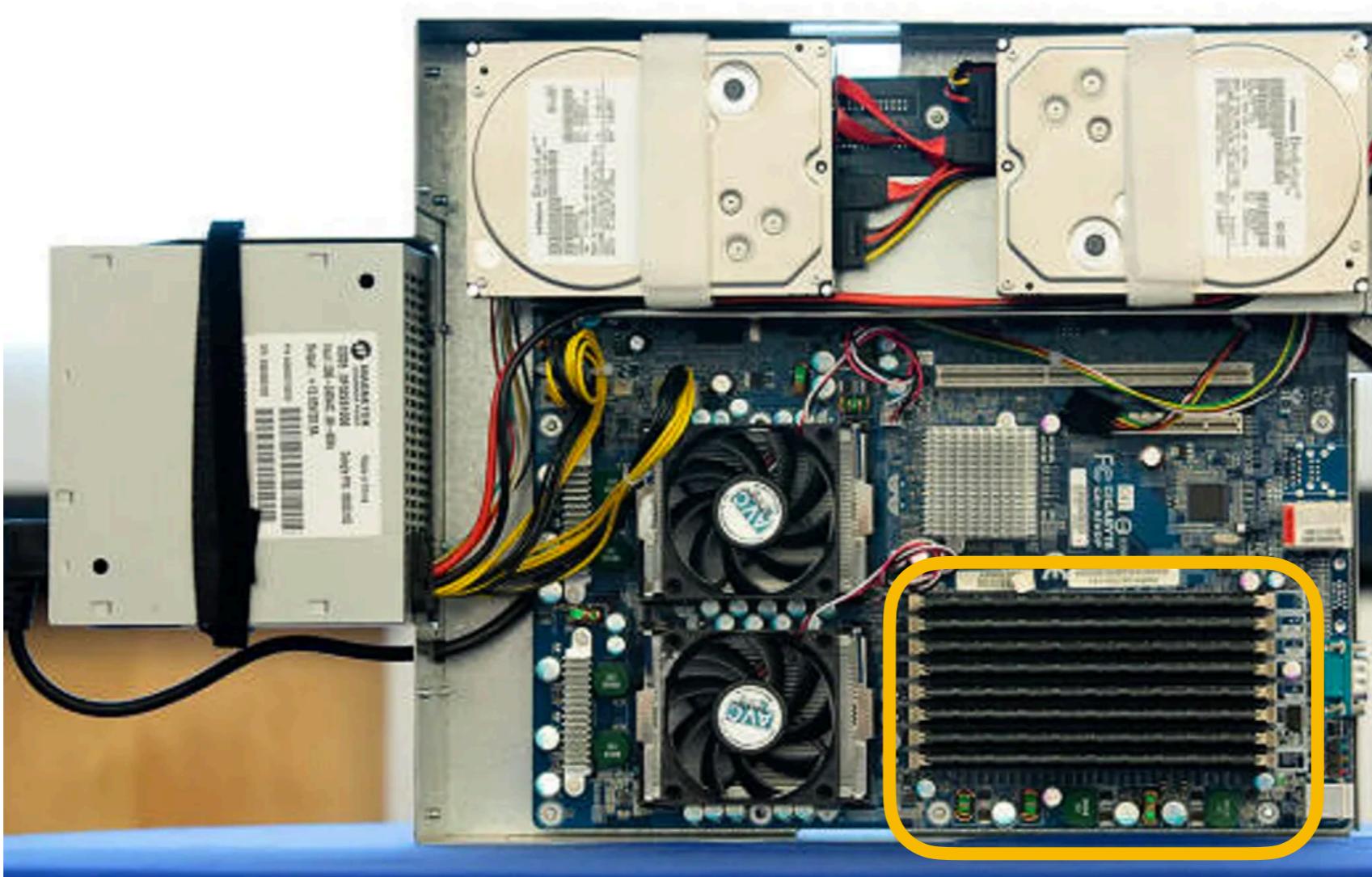
- Introductions
- Main sites: tylercaraza-harter.com, Canvas, GitHub
- Other tools: Email, TopHat, Piazza, GitHub classroom

Resources

- Overview
- Compute
- **Memory**
- Storage
- Network

Deployment

RAM: Random Access Memory



What is "Random"?

"Random" means we can jump around and access data from different locations efficiently.

In contrast, some devices that hold data are only efficient **sequentially**:



Bits

RAM holds bits.

A "bit" holds a 0 or a 1 (two possible values).

2 bits can hold 00, 01, 10, 11 (four possible values).

N bits can hold 2^N possible values.

Representation

Different encodings/representations decide what a combination of bits mean.

bits				
000	A	0	-4	
001	B	1	-3	
010	C	2	-2	colors
011	D	3	-1	images
100	E	4	0	floats
101	F	5	1	etc.
110	G	6	2	
111	H	7	3	

Bytes

A byte is 8 bits, so can hold $2^8 = 256$ possible values.

RAM is "byte addressible"

- each byte of data has it's own address the CPU can use to access it
- extracting a single bit from a byte actually involves more steps than using the whole byte

Units:

- 1 KB = 1024 bytes (or sometimes 1000 bytes)
- 1 MB = 1024 KB (or sometimes 1000 KB)
- 1 GB = 1024 MB (or sometimes 1000 MB)
- 1 TB = 1024 GB (or sometimes 1000 GB)

RAM Characteristics

Characteristics

- **small** (for example, an e2-medium VM has 4 GB)
- **volatile** (contents lost upon reboot)
- **fast** (much faster than storage devices)

Some uses

- actively used data (e.g., Python list, program code, DataFrame)
- copies of "hot" data (frequently accessed) from storage

Outline

Course Overview

- Introductions
- Main sites: tylercaraza-harter.com, Canvas, GitHub
- Other tools: Email, TopHat, Piazza, GitHub classroom

Resources

- Overview
- Compute
- Memory
- Storage
- Network

Deployment

Block devices: storing 0s and 1s

Hard Disk Drives (**HDDs**)



Solid State Disks (**SSDs**)



- 0s/1s stored on spinning magnetized platter
- moving head reads/writes data

- 0s/1s stored in charged cells
- no moving parts (faster)

Both are "**block devices**"

- data is read/written in blocks of many bytes (for example, 0.5 KB)
- reading 1 byte or 1 block takes same time

HDD and SSD Characteristics

Characteristics

- **large** (> 1 TB devices are affordable)
- **nonvolatile** (contents lost upon reboot)
- **slow** (much slower than memory)

Some uses

- large datasets
- data that needs to be preserved long term

Metrics

Capacity

- how much data can be stored?
- measured in bytes (for example, 500 GB)

Throughput

- how fast can data be read/written?
- measure in bytes/second (for example, 200 MB/s)
- throughput will depend on access pattern (for example, spinning disks have low throughput for random accesses)

Price/Terabyte of SSD & HDD are Converging Rapidly

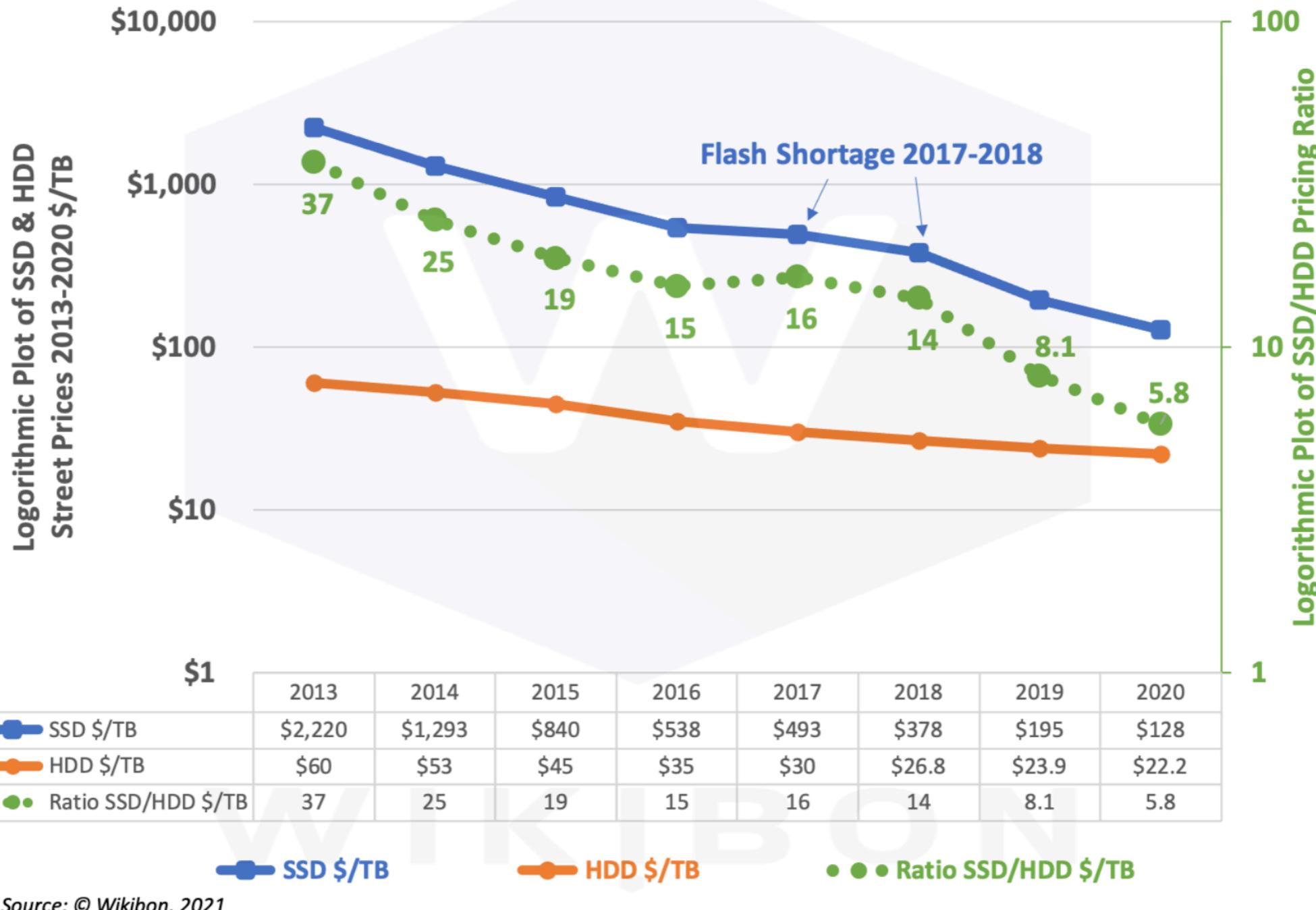


Figure 3 – Flash & HDD Pricing 2013 – 2020

Source: © Wikibon, 2021. Wikibon uses historical data is from multiple sources, including IDC, Gartner, Kitguru, Nidec, Trendfocus, and Wells Fargo LLC

Outline

Course Overview

- Introductions
- Main sites: tylercaraza-harter.com, Canvas, GitHub
- Other tools: Email, TopHat, Piazza, GitHub classroom

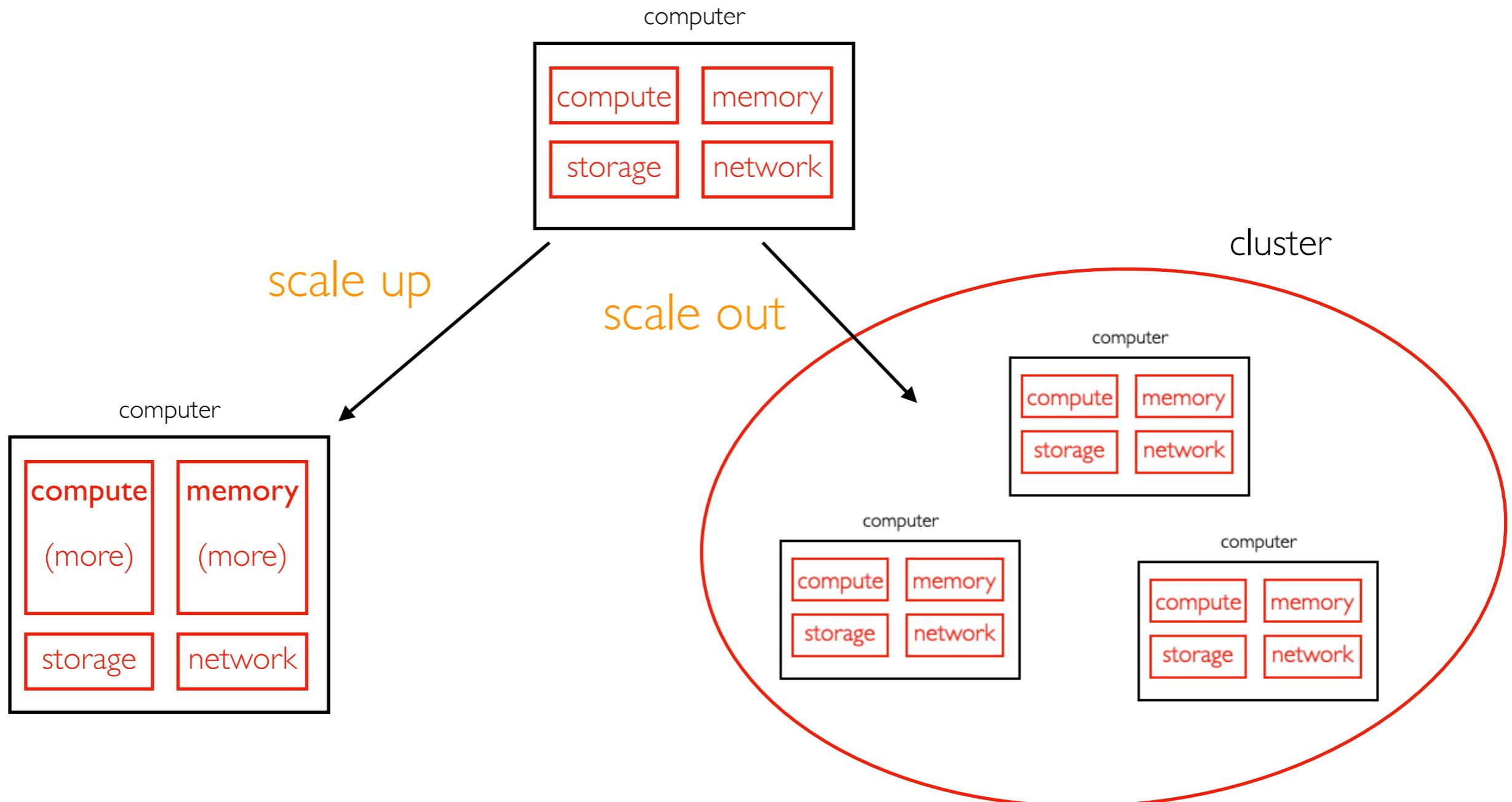
Resources

- Overview
- Compute
- Memory
- Storage
- Network

Deployment

Network

When scaling out, many **nodes** (computers) will be communicating via a network.



Network



Server



Rack

<https://www.dotmagazine.online/issues/digital-infrastructure-and-transforming-markets/data-center-models>

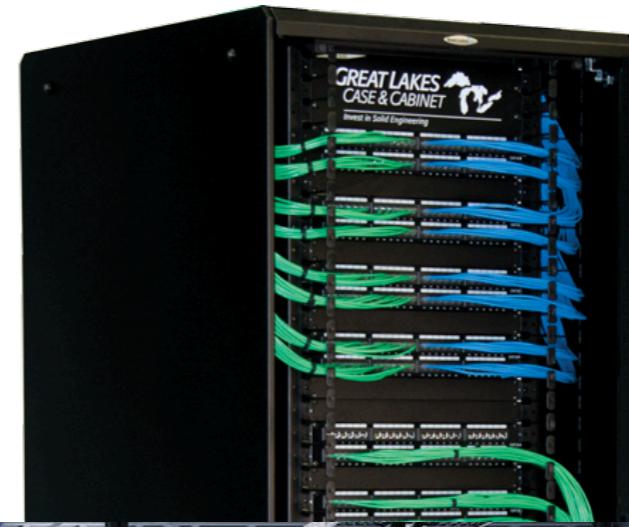
https://buy.hpe.com/us/en/servers/proliant-dl-servers/proliant-dl10-servers/proliant-dl20-server/hpe-proliant-dl20-gen10-plus-e-2336-2-9ghz-6-core-1p-16gb-u-4sff-500w-rps-server/p/p41115-b21?ef_id=Cj0KCQiAt66eBhCnARlsAKf3ZNFjsg49UV6Zm33R7lkRqi-XOd_JECmdyqNMAm2CKLSm_F-z6jTYDTQaAgMTEALw_wcB;G:s&s_kwcid=AL!13472!3!33!628972784!!g!318267171339!!1707918369!67076417419&gclid=Cj0KCQiAt66eBhCnARlsAKf3ZNFjsg49UV6Zm33R7lkRqi-XOd_JECmdyqNMAm2CKLSm_F-z6jTYDTQaAgMTEALw_wcB

https://www.server-rack-online.com/g!910ent-4048sss.html?utm_medium=shoppingengine&utm_source=googlebase&utm_source=google&utm_medium=cpc&adpos=&scid=scplpg!910ent-4048sss&sc_intid=g!910ent-4048sss&gclid=Cj0KCQiAt66eBhCnARlsAKf3ZNEMYINPAA0RFGQIF0DsieCM6oh7i3kuJvJlpnmJAlOpAJ3RWTI1QMAaAqRnEALw_wcB

Network



Server



Data Center

<https://www.dotmagazine.online/issues/digital-infrastructure-and-transforming-markets/data-center-models>

https://buy.hpe.com/us/en/servers/proliant-dl-servers/proliant-dl10-servers/proliant-dl20-server/hpe-proliant-dl20-gen10-plus-e-2336-2-9ghz-6-core-1p-16gb-u-4sff-500w-rps-server/p/p4115-b21?ef_id=Cj0KCQiAt66eBhCnARlsAKf3ZNFjsg49UV6Zm33R7lkRqi-XOd_JECmdyqNMAm2CKLSm_F-z6jTYDTQaAgMTEALw_wcB&s_kwcid=AL!13472!3!33!628972784!!g!318267171339!!1707918369!67076417419&gclid=Cj0KCQiAt66eBhCnARlsAKf3ZNFjsg49UV6Zm33R7lkRqi-XOd_JECmdyqNMAm2CKLSm_F-z6jTYDTQaAgMTEALw_wcB

https://www.server-rack-online.com/g!910ent-4048sss.html?utm_medium=shoppingengine&utm_source=googlebase&utm_source=google&utm_medium=cpc&adpos=&scid=scplpg!910ent-4048sss&sc_intid=g!910ent-4048sss&gclid=Cj0KCQiAt66eBhCnARlsAKf3ZNEMYINPAA0RFGQIF0DsieCM6oh7i3kuJvJlpnmJAIOpAJ3RWTI1QMAaAqRnEALw_wcB

Topology

Example configuring Hadoop File System (HDFS) to store data based on network topology:

python Example

```
#!/usr/bin/python3
# this script makes assumptions about the physical environment.
# 1) each rack is its own layer 3 network with a /24 subnet, which
# could be typical where each rack has its own
#     switch with uplinks to a central core router.

#
#          +-----+
#          |core router|
#          +-----+
#          /
#          \
#          +-----+      +-----+
#          |rack switch|  |rack switch|
#          +-----+      +-----+
#          | data node |  | data node |
#          +-----+      +-----+
#          | data node |  | data node |
#          +-----+      +-----+
#          "           "
```

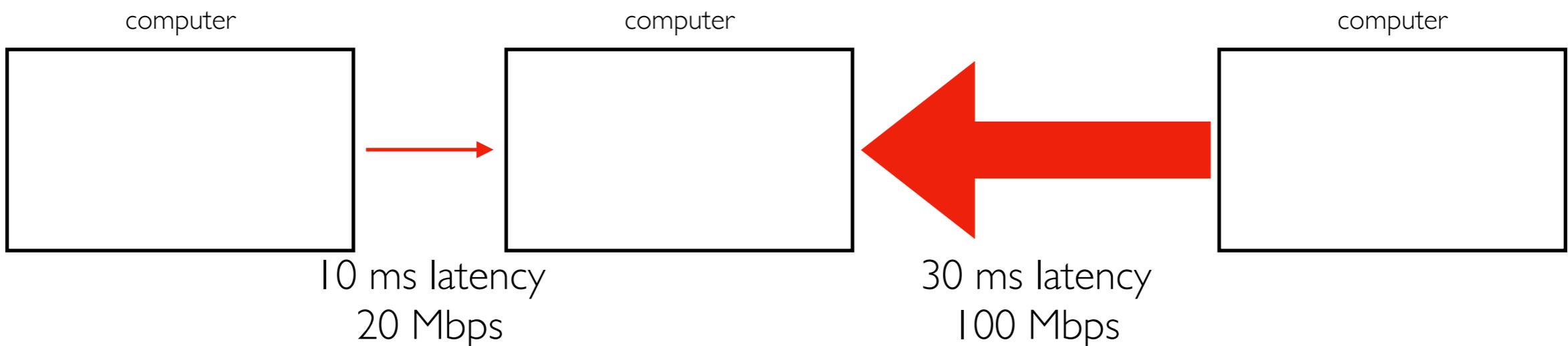
Metrics

Latency

- how long does it take to send data
- seconds, milliseconds (ms), etc

Bandwidth

- how many **bits** can be sent per second?
- Mbps (mega bits per second -- note lower case "b")
- What is faster, 10 Mbps or 10 MB/s?



Outline

Course Overview

- Introductions
- Main sites: tylercaraza-harter.com, Canvas, GitHub
- Other tools: Email, TopHat, Piazza, GitHub classroom

Resources

- Overview
- Compute
- Memory
- Storage
- Network

Deployment

Welcome to Data Programming II!

Builds on CS 301 220. <https://stat.wisc.edu/undergraduate-data-science-studies/>

CS 220

- getting results
- writing correct code
- using objects
- functions: `f(obj)`
- lists+dicts
- analyzing datasets
- plots
- tabular analysis

CS 320

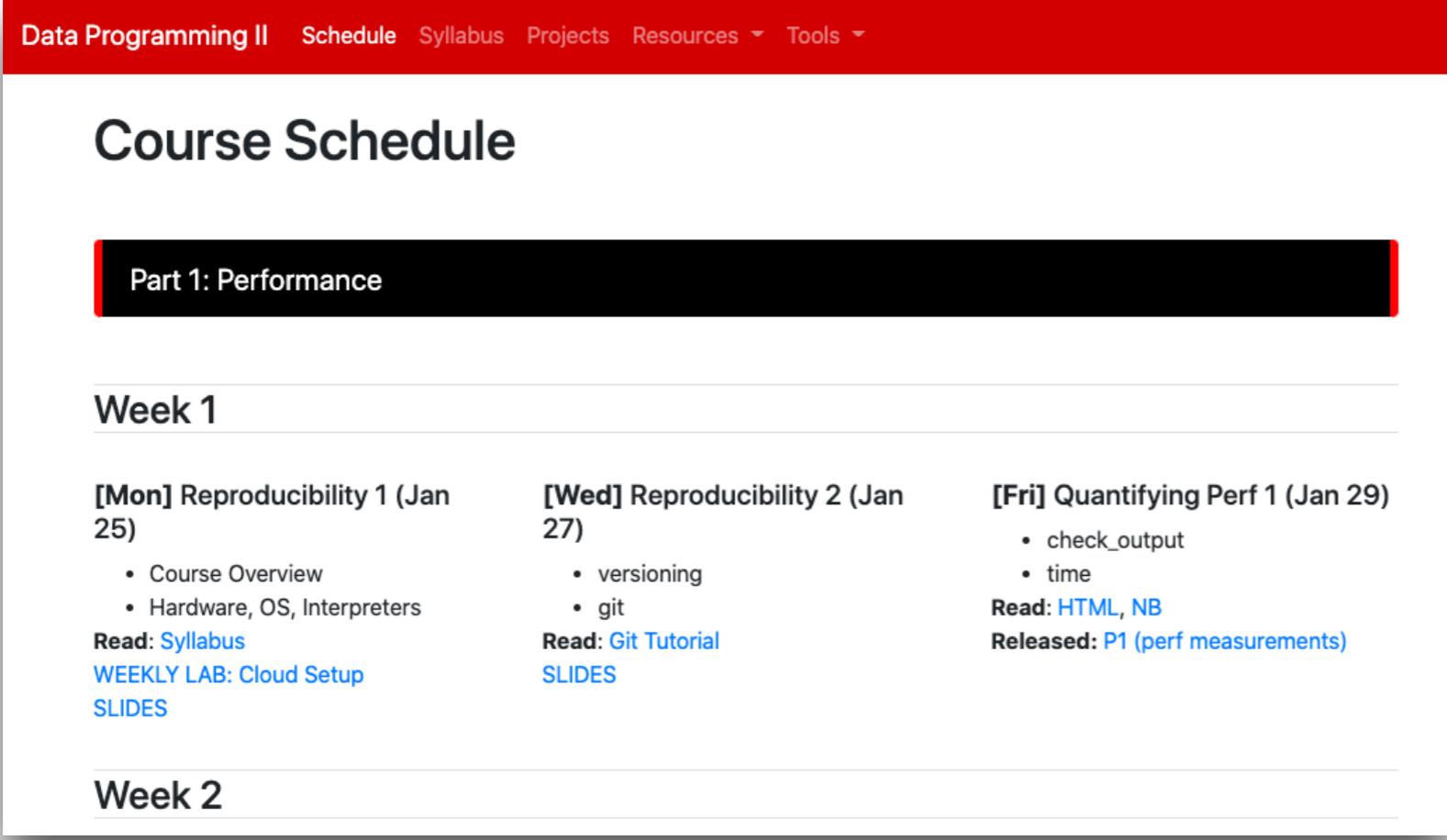
- getting **reproducible** results
- writing **efficient** code
- designing **new types** of objects
- methods**: `obj.f()`
- graphs+trees
- collecting**+analyzing datasets
- animated visualizations
- simple machine learning**



Course Logistics

Course Website

It's here: <https://tyler.caraza-harter.com/cs320/f22/schedule.html>

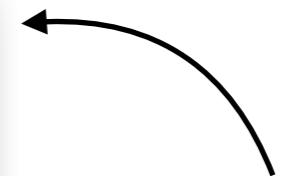


The screenshot shows a course website with a red header bar containing links for Data Programming II, Schedule, Syllabus, Projects, Resources, and Tools. Below the header, the title "Course Schedule" is displayed. A black banner labeled "Part 1: Performance" spans across the middle of the page. The main content is organized into two weeks:

Week 1

[Mon] Reproducibility 1 (Jan 25)	[Wed] Reproducibility 2 (Jan 27)	[Fri] Quantifying Perf 1 (Jan 29)
• Course Overview	• versioning	• check_output
• Hardware, OS, Interpreters	• git	• time
Read: Syllabus	Read: Git Tutorial	Read: HTML , NB
WEEKLY LAB: Cloud Setup	SLIDES	Released: P1 (perf measurements)

Week 2



read syllabus carefully
and checkout other content

I'll also use **Canvas** for four things:

- general announcements
- quizzes
- online office hours
- simple grade summaries (not feedback or exam answers)

Scheduled Activities

Lectures

- 3 times weekly
- feel free to bring a laptop
- will often be recorded+posted online (questions will be recorded -- feel free to save until after if you aren't comfortable being recorded)
- might not post if bad in-person attendance or technical issues

Lab

- Weekly on Mondays, bring a laptop
- Work through lab exercises with group mates
- 320 staff will walk around to answer questions
- Required for participation credit!
- Answer TopHat question what at lab (<https://app.tophat.com/e/594996>) or fill "Lab Absence" each week for credit: <https://tyler.caraza-harter.com/cs320/s22/surveys.html>. We'll occasionally cross-check TopHat with paper sign-in.

Class organization: People

Teams

- you'll be assigned to a team of 4-7 students
- teams will last the whole semester
- some types of collaboration with team members are allowed (not required) on graded work, such as projects+quizzes
- most collaboration with non-team members is not allowed

Staff

1. Instructor
2. Teaching Assistants (grad students)
3. Mentors (undergrads)

we all provide office hours, and you can attend any that you prefer!

Class organization: People

Teams

- you'll be assigned to a team of 4-7 students
- teams will last the whole semester
- some types of collaboration with team members are allowed (not required) on graded work, such as projects+quizzes
- most collaboration with non-team members is not allowed

Staff

1. Instructor
 2. Teaching Assistants
 3. Mentors
-  **head TA:** in charge of projects
team TA: primary contact for team, same whole semester
grader TA: reviews projects (rotates weekly)

we all provide office hours, and you can attend any that you prefer!

Communication

Piazza

- find link on site
- don't post >5 lines of project-related code (considered cheating)

Forms

- <https://tyler.caraza-harter.com/cs320/f22/surveys.html>
- Who are you? Feedback Form. Thank you! Grading Issues.

Email

- me: tharter@wisc.edu
- TAs: <https://canvas.wisc.edu/courses/322105/pages/contact-info>

Course Etiquette

Meetings

1. office hours are drop-in (no need to reserve)
2. email me about individual meeting availability if needed

Email

3. let us know your NetID (if not from netid@wisc.edu)
4. don't start new email thread if topic is the same
5. CC team members when appropriate
6. unless urgent, please give me 48 hours to respond before following up (I'll try to be faster usually)
7. use your judgement about whether to email me or TA first (if one TA doesn't know something, ask me next before others)
8. if general question, consider using piazza instead if general interest

Graded Work: Exams/Quizzes

Ten Online Quizzes - 1% each

- cumulative, no time limit
- on Canvas, open book/notes
- can take together AT SAME TIME with team members
(no other human help allowed)

Midterms - 14% each

- cumulative, individual, multi-choice, 40 minutes
- one page notes, both sides
- in class

Final - 16%

- cumulative, individual, multi-choice, 2 hours
- one page notes, both sides

Graded Work: Projects+Participation

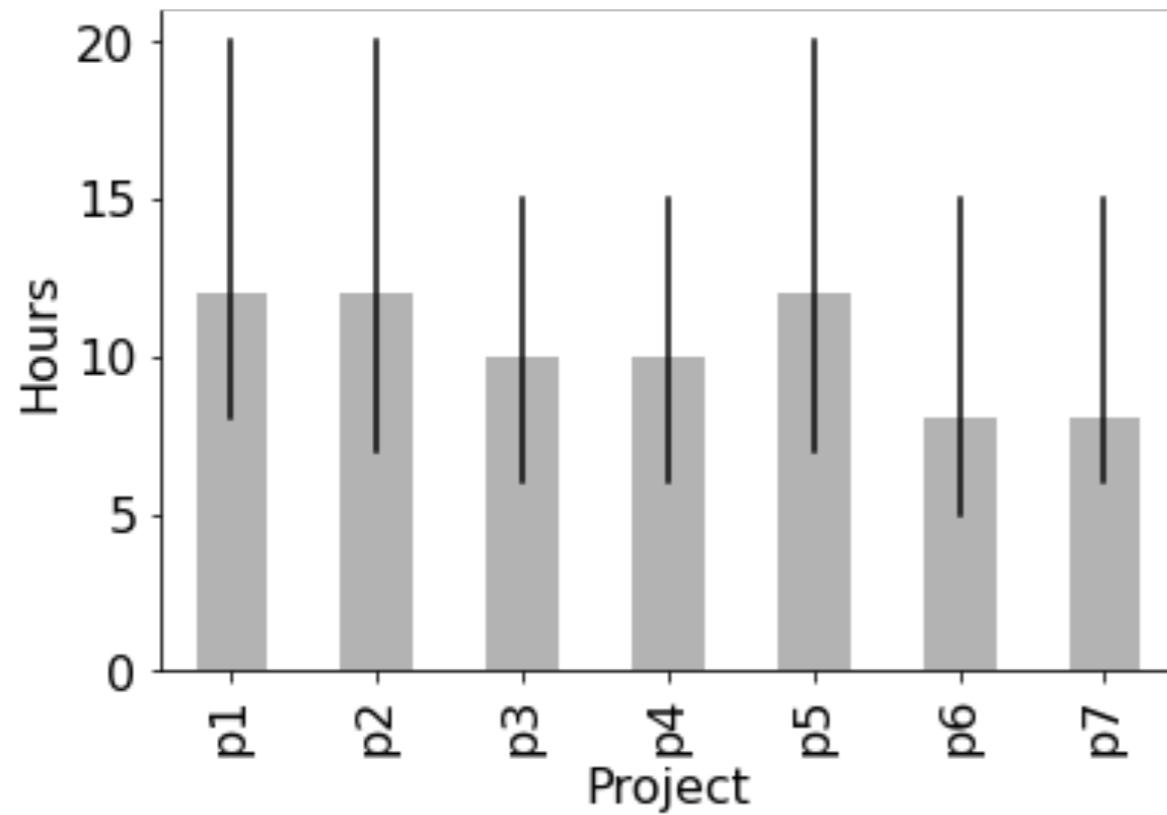
7 Projects - **6%** each

- **format:** notebook, module, or program
- part 1: you can optionally collaborate with team
- part 2: must be individually (only help from 320 staff)
- still a `tester.py`, but more depends on TA evaluation
(more plots)
- **ask for specific feedback**
(giving constructive criticism is a priority in CS 320)

Participation - **4%**

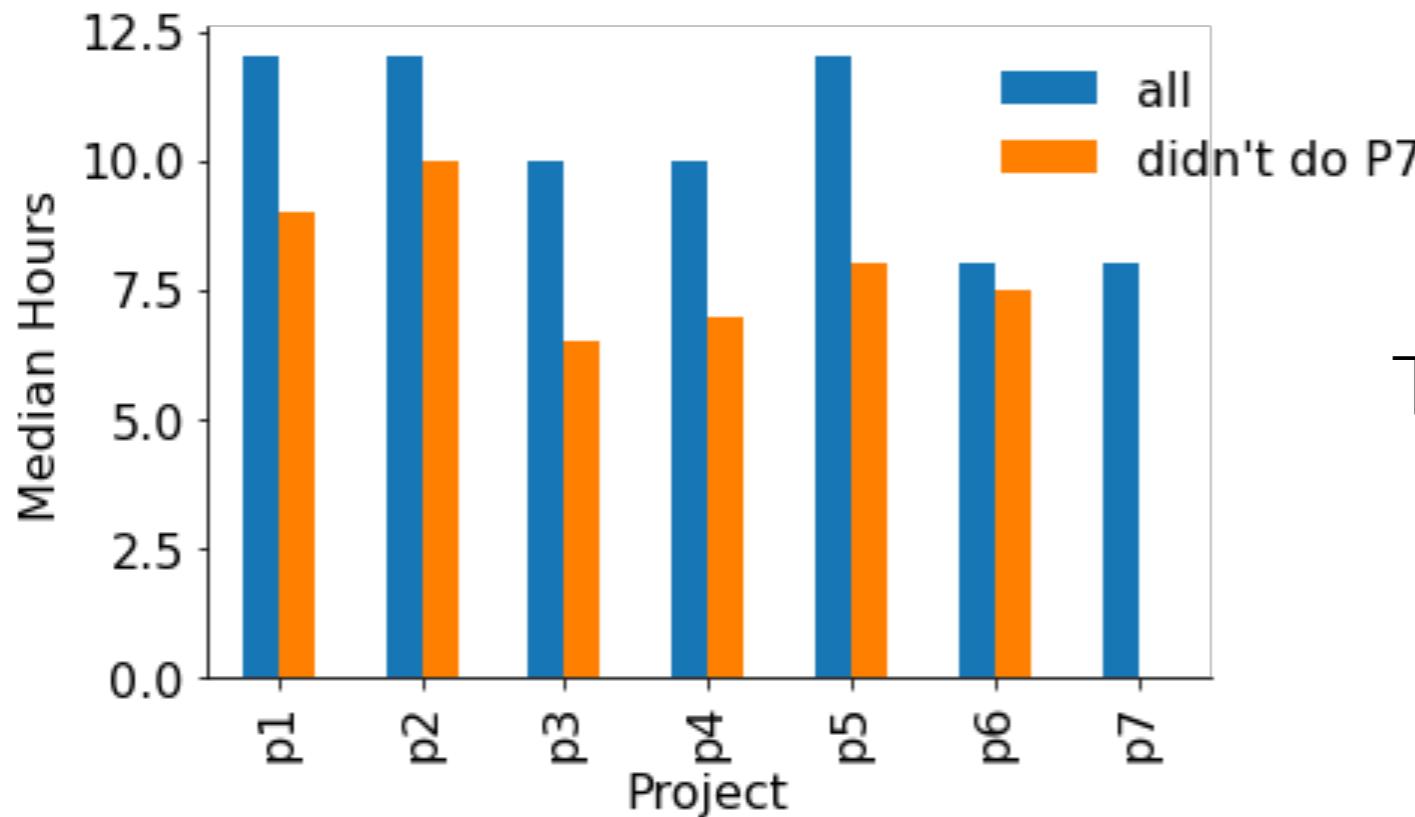
- lab attendance
- class surveys
- etc.

Time Commitment



Observations

- 10-12 hours per project is typical
- 20% of students sometimes spend 20+ hours on some projects
- students who were faster early on were less likely to complete the course



Typical Weekly Expectations

- 4 hours - lecture/lab
- 6 hours - project coding
- 2 hours - reading/quizzes/etc

Academic Misconduct

Read syllabus to make sure you know what is and isn't OK.

It's not obvious!

Since Fall 2019, I have made the following misconduct reports:

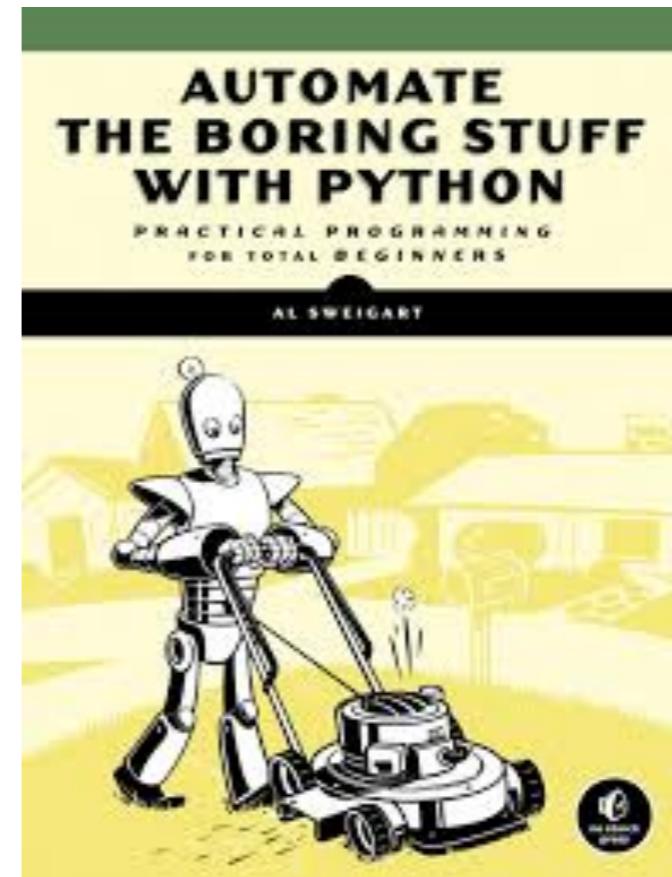
- **58** students for cheating on projects
- **2** past students for sharing solutions from past semesters
- **8** students for cheating on exams
- **1** student for faking participation

How we'll keep the class fair

- run MOSS on submissions
- randomize exam question order

Please talk to me if you're feeling overwhelmed with 320 or your semester in general!

Reading: same as 220/301 and some others...



I'll post links to other online articles and my own notes

Lectures don't assume any reading prior to class

Tips for 320 Success

1. Just show up!
 - Get 100% on participation, don't miss quizzes, submit group work
2. Use office hours
 - we're idle after a project release and swamped before a deadline
3. Do labs before projects
4. Take the lead on group collaboration
5. Learn debugging
6. Run the tester often
7. If you're struggling, reach out -- the sooner, the better

Any questions?

Today's Lecture: Reproducibility

[All](#)[News](#)[Images](#)[Books](#)[Videos](#)[More](#)[Settings](#)[Tools](#)

About 44,700,000 results (0.64 seconds)

Dictionary

Search for a word 



re·pro·duc·i·bil·i·ty

/rēprə'd(y)oosə'bilədē/

noun

noun: reproducibility

the ability to be reproduced or copied.

"the reproducibility of reconstructive surgery techniques"

- the extent to which consistent results are obtained when an experiment is repeated.

"the experiments were conducted numerous times to test the reproducibility of the results"

Discuss: how might we define "reproducibility" for a data scientist?

Big question: will my program run on someone else's computer?
(not necessarily written in Python)

Things to match:

1

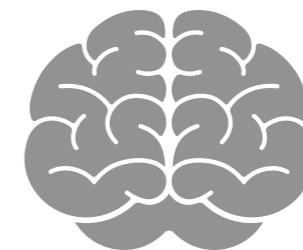
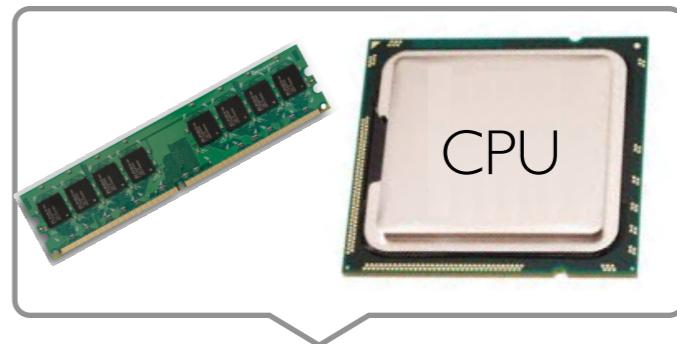
Hardware

2

Operating System

3

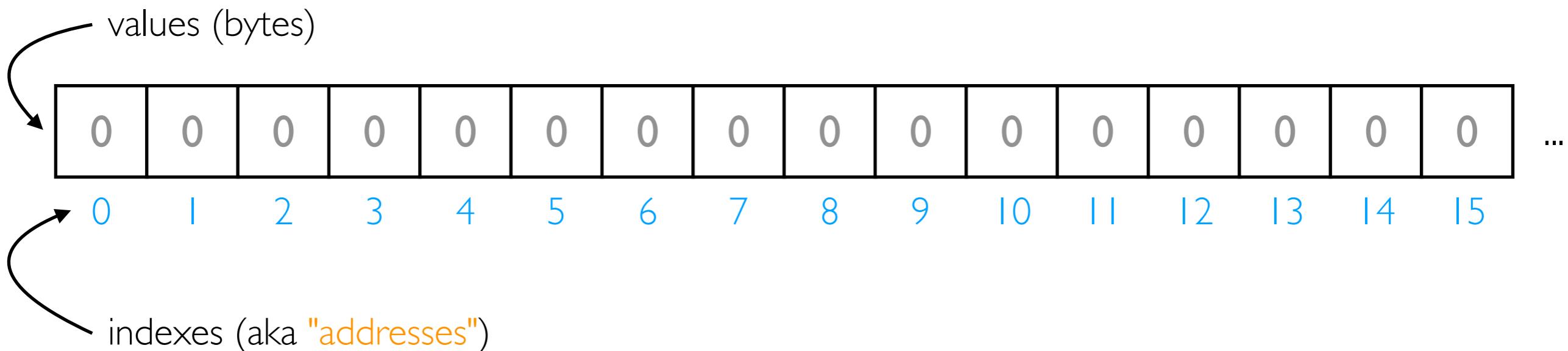
Dependencies ← next lecture



Hardware: Mental Model of Process Memory

Imagine...

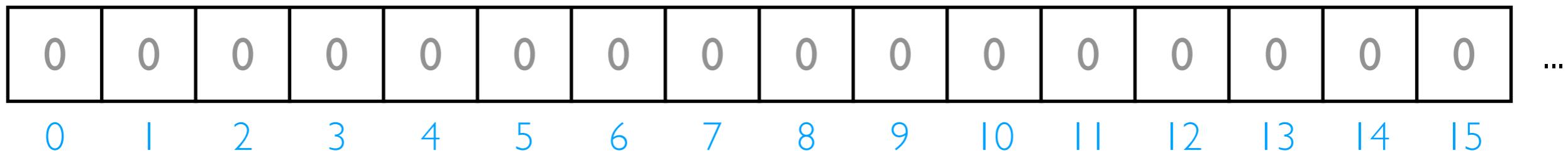
- one huge list, **per each running program process**, called "address space"
- every entry in the list is an integer between 0 and 255 (aka a "byte")



How can we use one giant list to handle the following?

- multiple lists
- variables and other references
- strings
- code

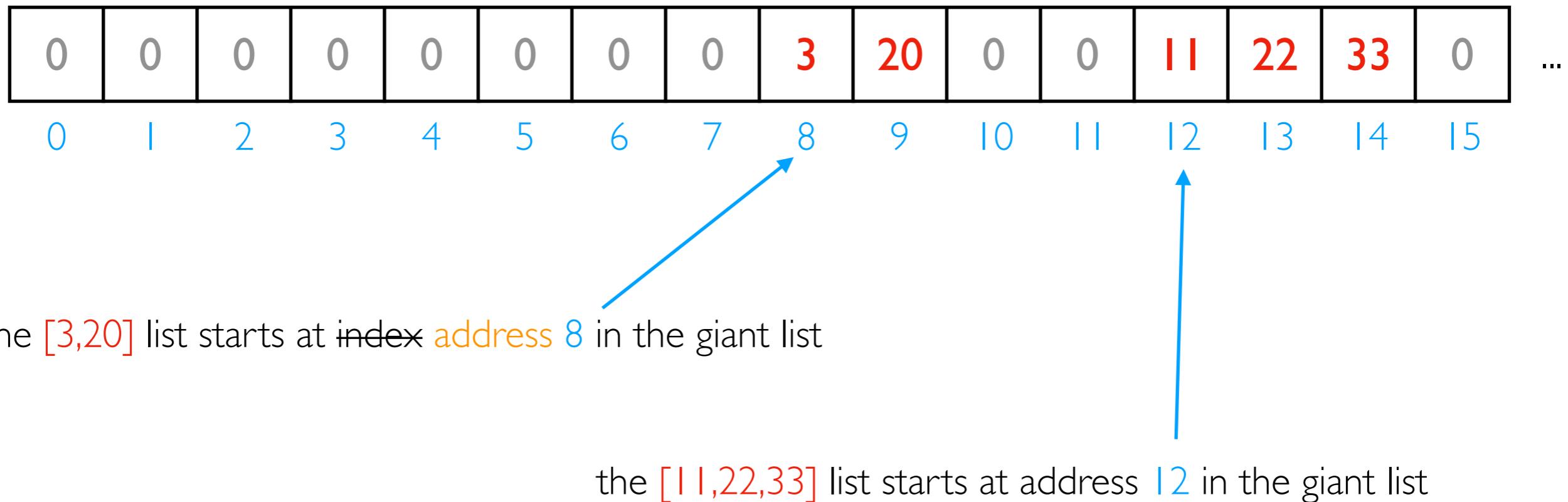
data



Is this really all we have for state?

How can we use one giant list to handle the following?

- multiple lists
- variables and other references
- strings
- code



How can we use one giant list to handle the following?

- multiple lists
- variables and other references
- strings
- code

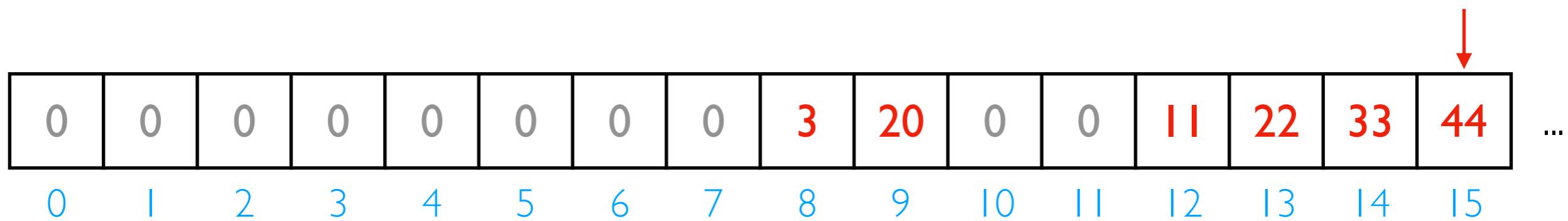
0	0	0	0	0	0	0	0	3	20	0	0	11	22	33	0	...
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	

implications for performance...

```
# fast  
L2.append( 44 )
```

How can we use one giant list to handle the following?

- multiple lists
- variables and other references
- strings
- code

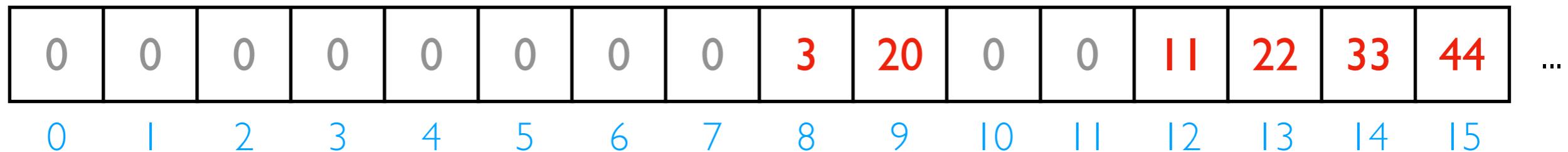


fast
L2.append(44)

implications for performance...

How can we use one giant list to handle the following?

- multiple lists
- variables and other references
- strings
- code



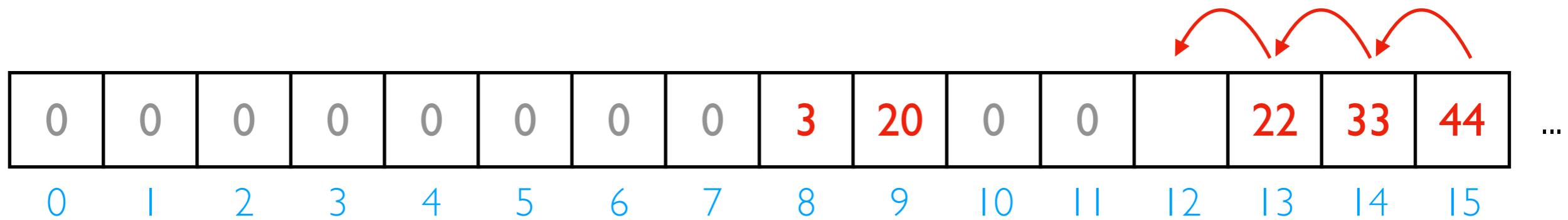
implications for performance...

```
# fast  
L2.append( 44 )
```

```
# slow  
L2.pop( 0 )
```

How can we use one giant list to handle the following?

- multiple lists
- variables and other references
- strings
- code



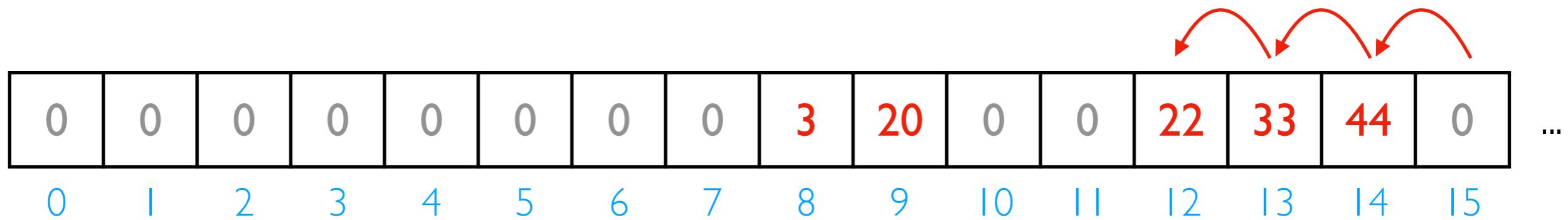
implications for performance...

```
# fast  
L2.append( 44 )
```

```
# slow  
L2.pop( 0 )
```

How can we use one giant list to handle the following?

- multiple lists
- variables and other references
- strings
- code



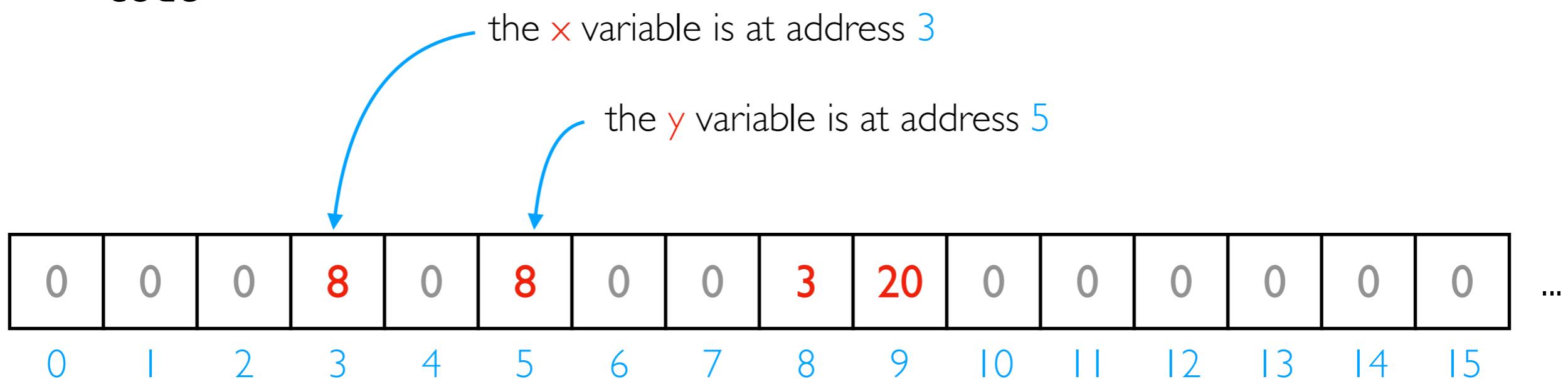
We'll think more rigorously about performance in CS 320 (big-O notation)

```
# fast  
L2.append( 44 )
```

```
# slow  
L2.pop( 0 )
```

How can we use one giant list to handle the following?

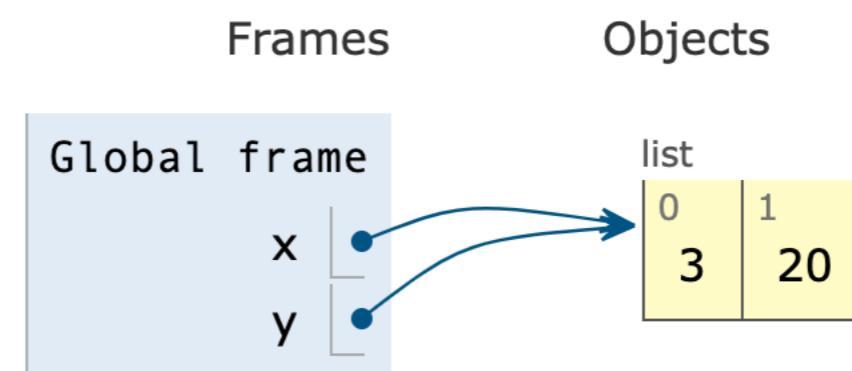
- multiple lists
- **variables and other references**
- strings
- code



Python 3.6

```
1 x = [3, 20]
→ 2 y = x
```

[Edit this code](#)

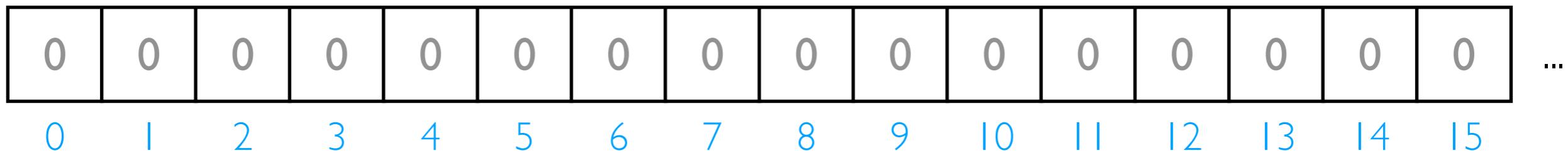


PythonTutor's visualization

How can we use one giant list to handle the following?

- multiple lists
- variables and other references
- **strings**
- code

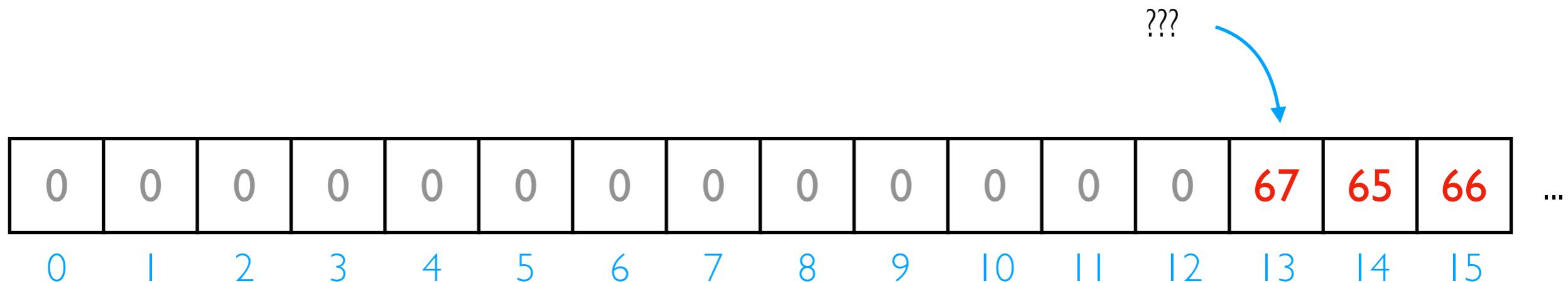
discuss: how?



Is this really all we have for state?

How can we use one giant list to handle the following?

- multiple lists
- variables and other references
- **strings**
- code



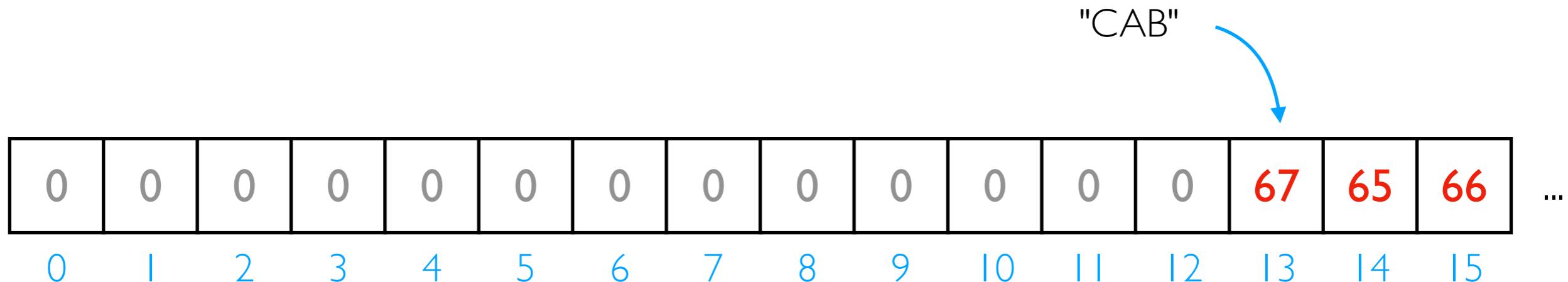
code	letter
65	A
66	B
67	C
68	D

f = open("file.txt", encoding="utf-8")

... ...

How can we use one giant list to handle the following?

- multiple lists
- variables and other references
- **strings**
- code



code	letter
65	A
66	B
67	C
68	D

f = open("file.txt", encoding="utf-8")

... ...

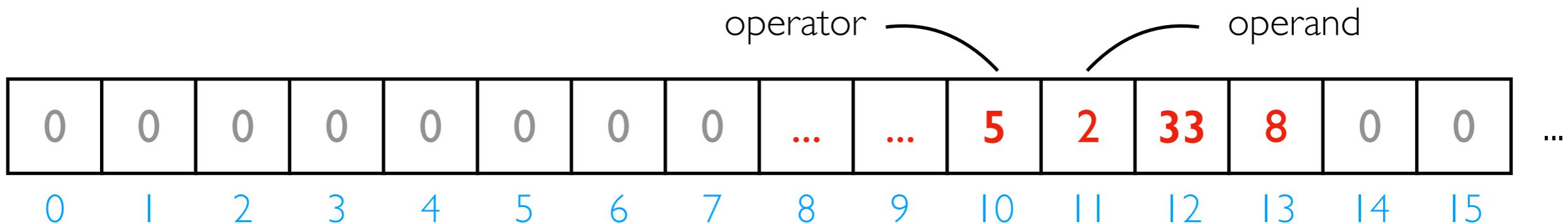
How can we use one giant list to handle the following?

- multiple lists
- variables and other references
- strings
- code

while ???:

i += 2

what line next?



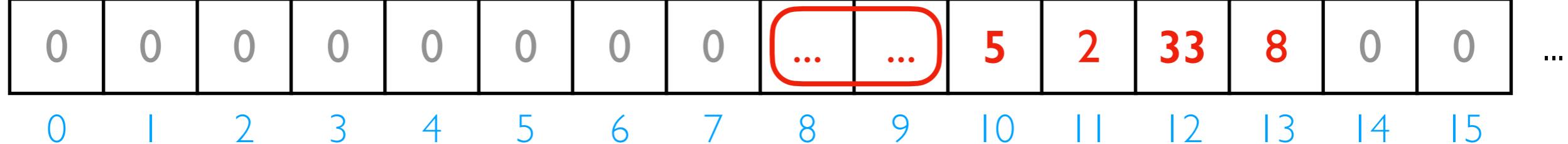
code	operation
5	ADD
8	SUB
33	JUMP
...	...

Instruction Set

Hardware: Mental Model of CPU

CPUs interact with memory:

- keep track of what instruction we're on
- understand instruction codes
- much more



Write code in Python 3.6
(drag lower right corner to resize code editor)

```
1 print("Hello, world!")
2 print("Hello, world!")
3 print("Hello, world!")
```

Instruction Set

code	operation
5	ADD
8	SUB
33	JUMP
...	...

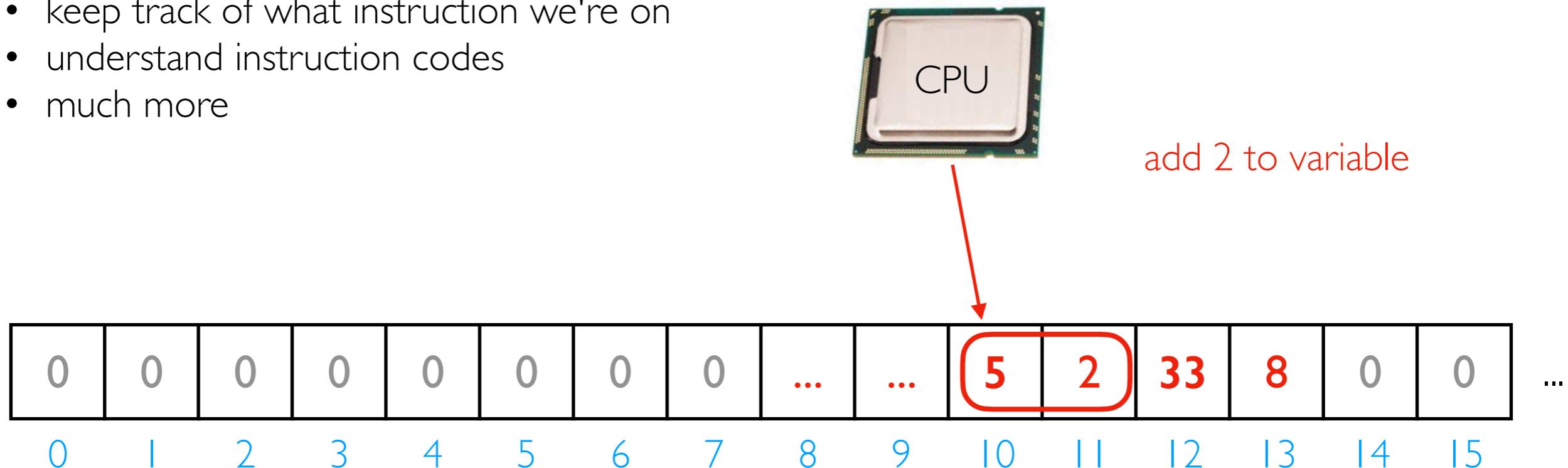
→ line that just executed

→ next line to execute

Hardware: Mental Model of CPU

CPUs interact with memory:

- keep track of what instruction we're on
- understand instruction codes
- much more

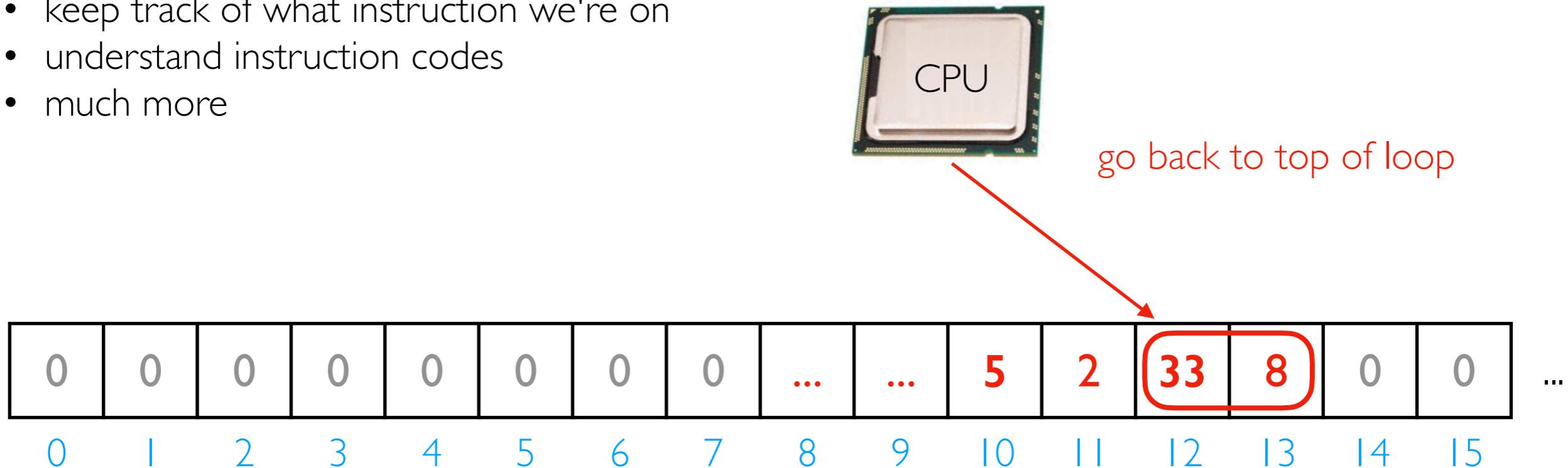


code	operation
5	ADD
8	SUB
33	JUMP
...	...

Hardware: Mental Model of CPU

CPUs interact with memory:

- keep track of what instruction we're on
- understand instruction codes
- much more



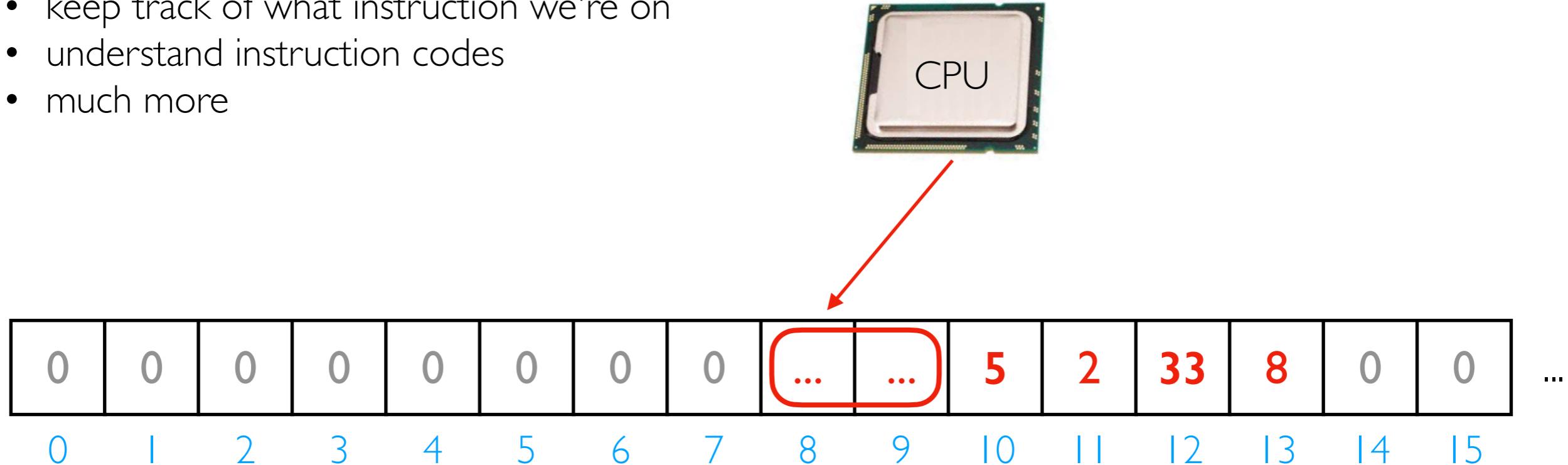
code	operation
5	ADD
8	SUB
33	JUMP
...	...

Instruction Set

Hardware: Mental Model of CPU

CPUs interact with memory:

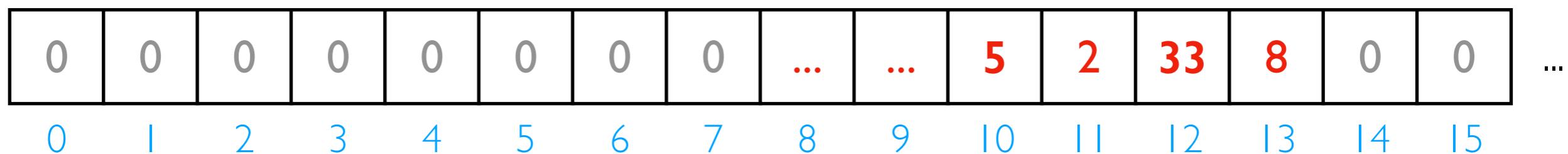
- keep track of what instruction we're on
- understand instruction codes
- much more



code	operation
5	ADD
8	SUB
33	JUMP
...	...

Hardware: Mental Model of CPU

discuss: what would happen if a
CPU tried to execute an
instruction for a different CPU?

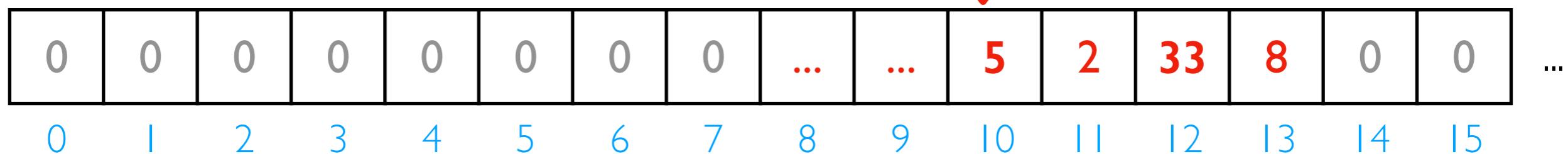


Instruction Set for CPU X	code	operation
	5	ADD
	8	SUB
	33	JUMP

Instruction Set for CPU Y	code	operation
	5	SUB
	8	ADD
	33	undefined

Hardware: Mental Model of CPU

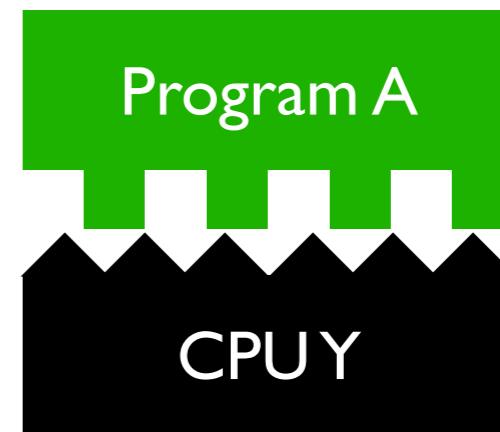
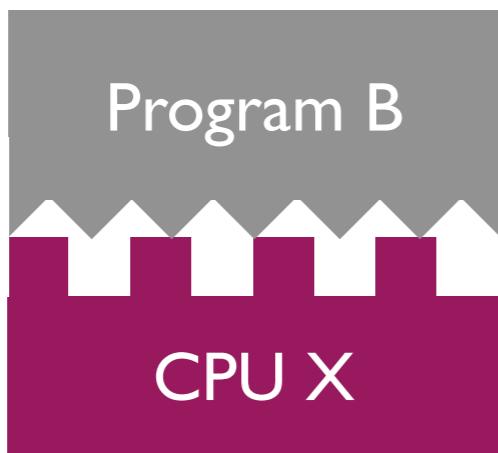
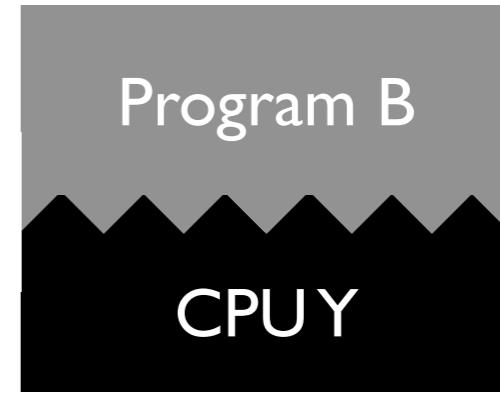
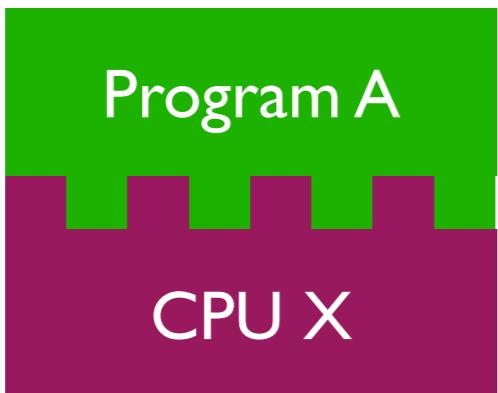
a CPU can only run programs that use instructions it understands!



Instruction Set for CPU X	code	operation
	5	ADD
	8	SUB
	33	JUMP

Instruction Set for CPU Y	code	operation
	5	SUB
	8	ADD
	33	undefined

A Program and CPU need to "fit"

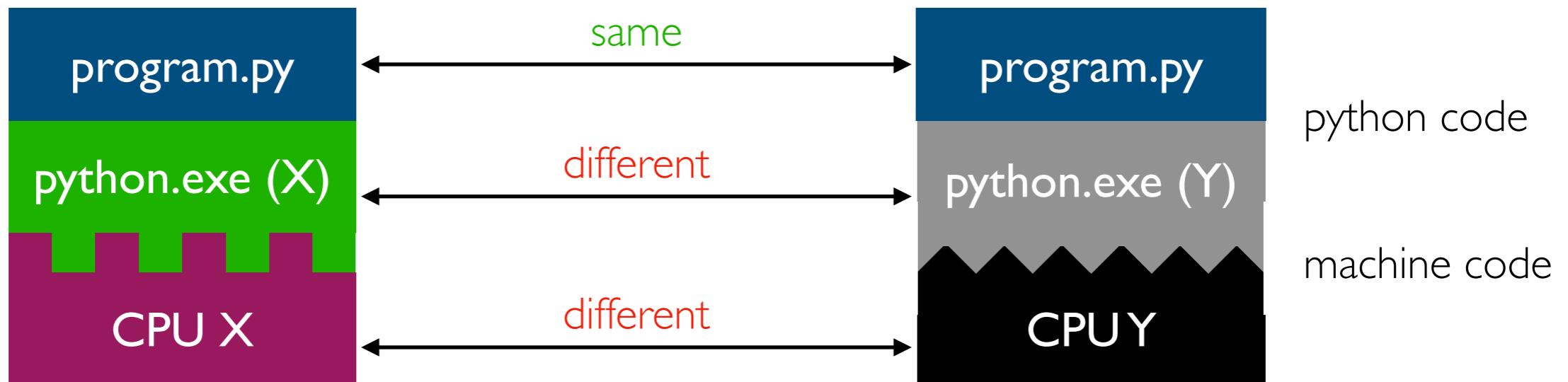


A Program and CPU need to "fit"



*why haven't we noticed this yet
for our Python programs?*

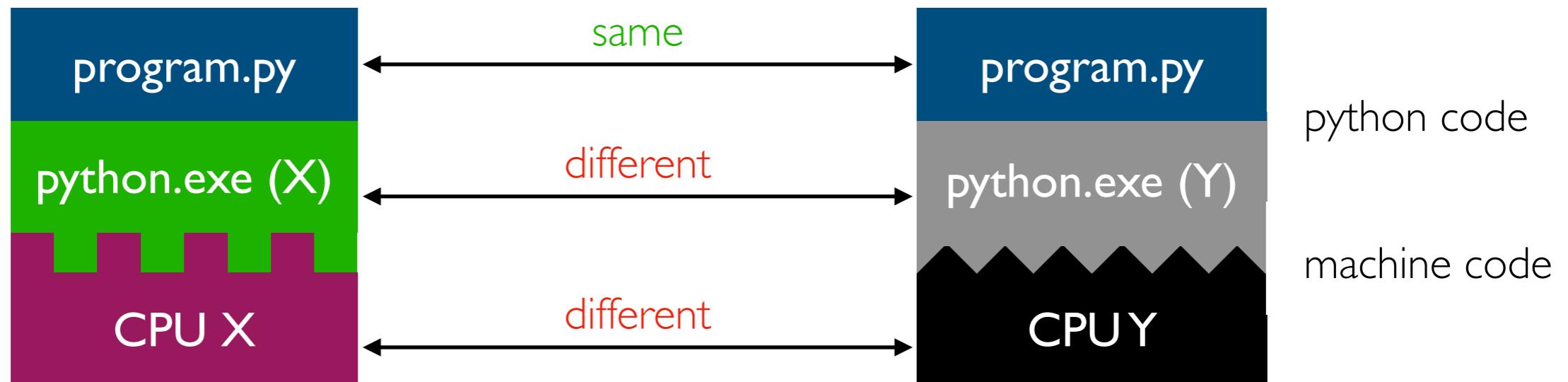
Interpreters



Interpreters (such as python.exe) make it easier to run the same code on different machines

A compiler is another tool for running the same code on different CPUs

Interpreters



Interpreters (such as python.exe) make it easier to run the same code on different machines

Discuss: if all CPUs had the instruction set,
would we still need a Python interpreter?

Big question: will my program run on someone else's computer?
(not necessarily written in Python)

Things to match:

1

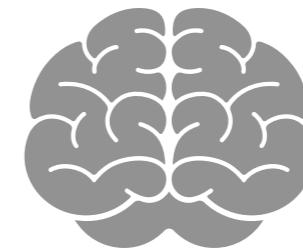
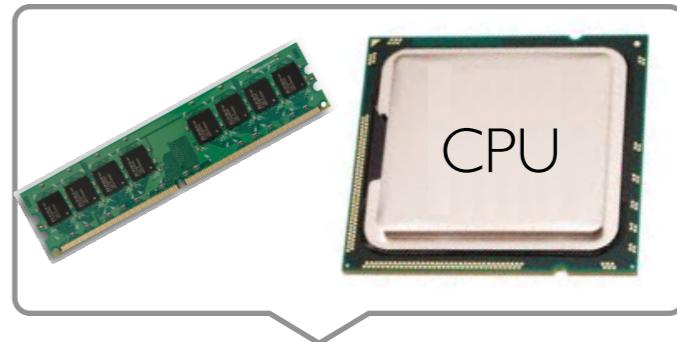
Hardware

2

Operating System

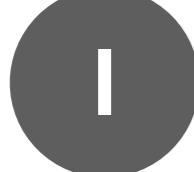
3

Dependencies ← next lecture



Big question: will my program run on someone else's computer?
(not necessarily written in Python)

Things to match:



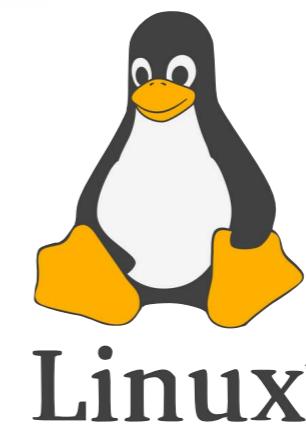
Hardware



Operating System



Dependencies ← next lecture



ANDROID

many others...



Red Hat



ubuntu

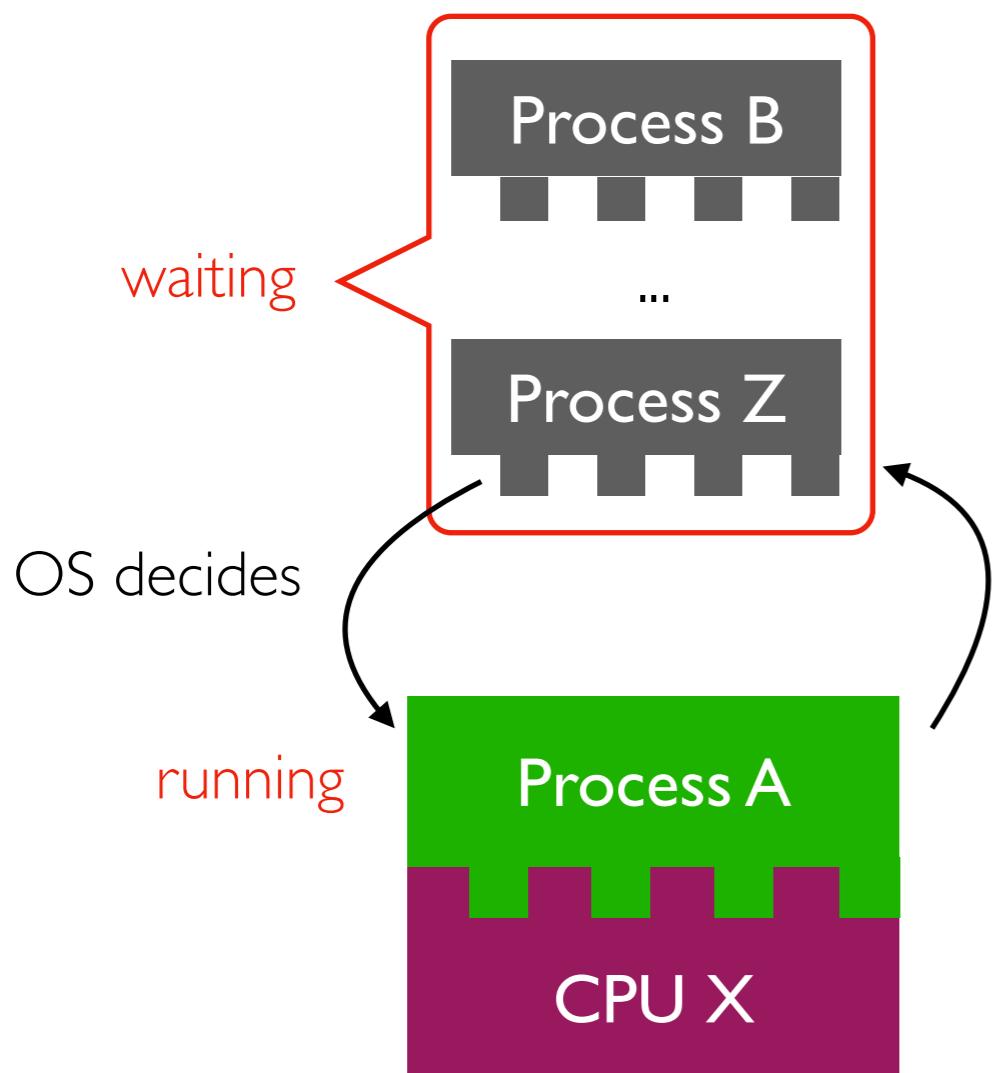
[this semester]

OS jobs: Allocate and Abstract Resources

[like CPU, hard drive, etc]

1

Allocation



2

Abstraction

```
f = open("file.txt")  
data = f.read()  
f.close()
```

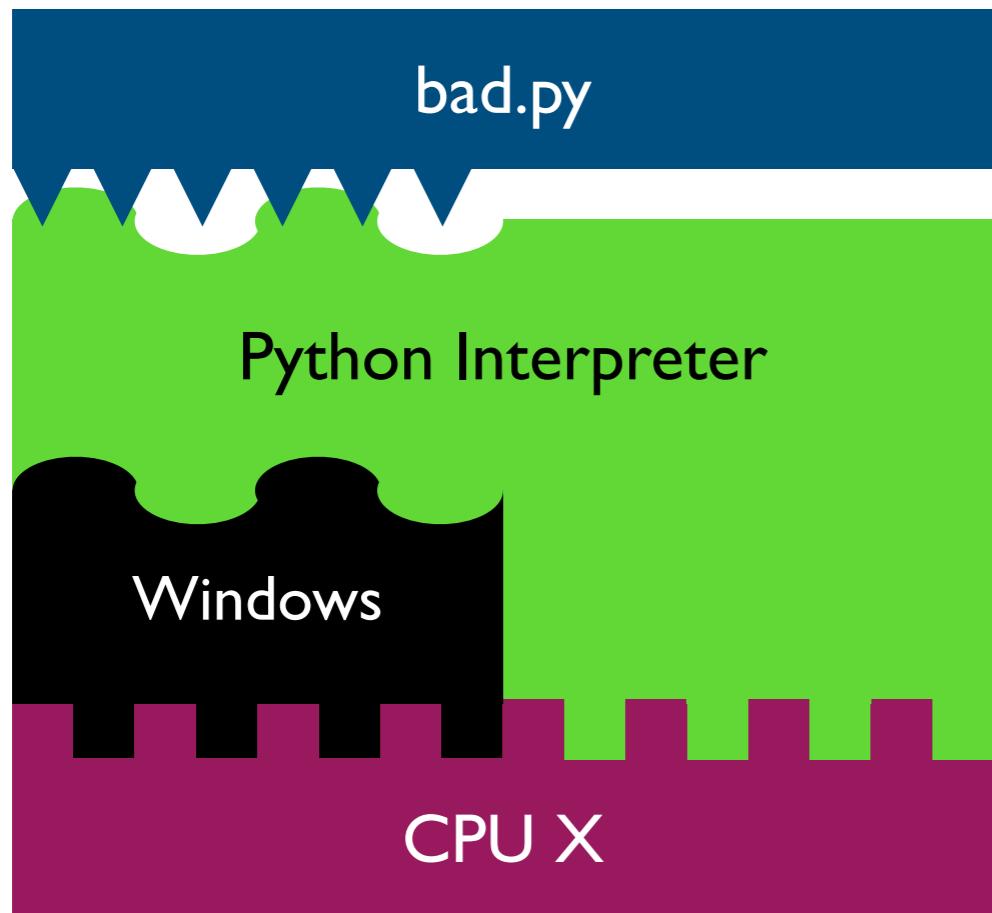
Operating System



only one process can run on CPU at a time
(or a few things if the CPU has multiple "cores")

ignorant of
files/directories

Harder to reproduce on different OS...

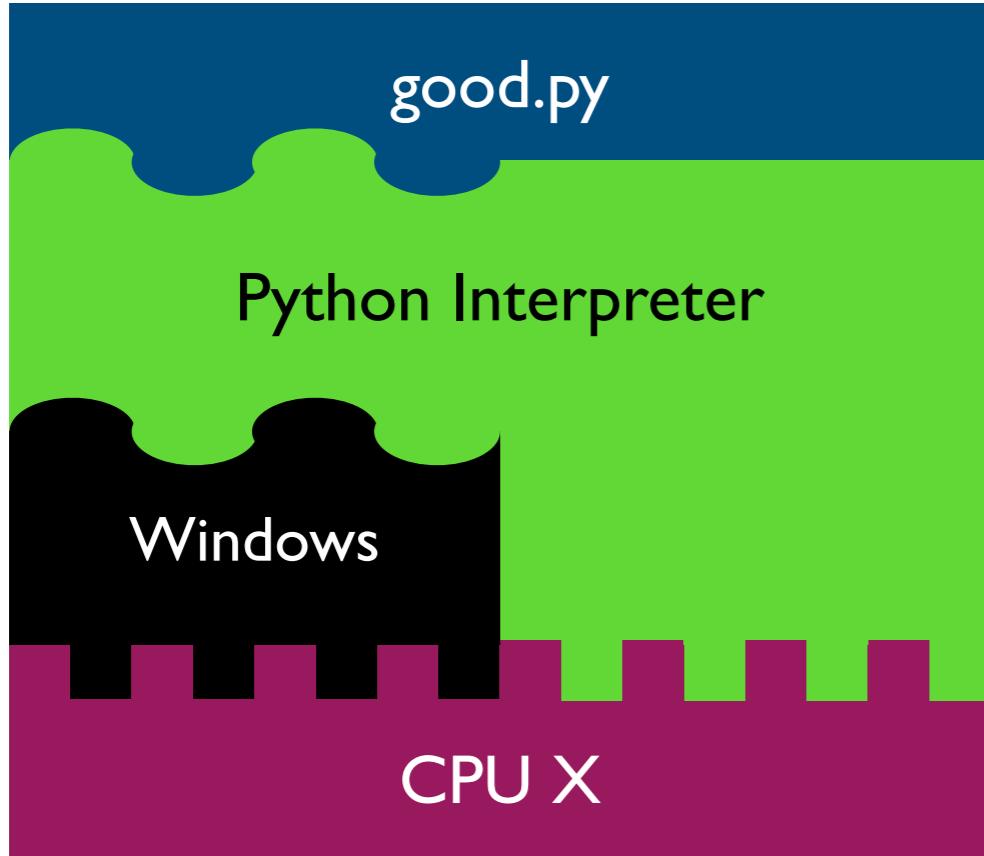


```
f = open( "/data/file.txt" )  
...  
...
```

The Python interpreter mostly lets you
[Python Programmer] ignore the CPU you run on.

But you still need to work a bit to "fit" the code to the OS.

Harder to reproduce on different OS...

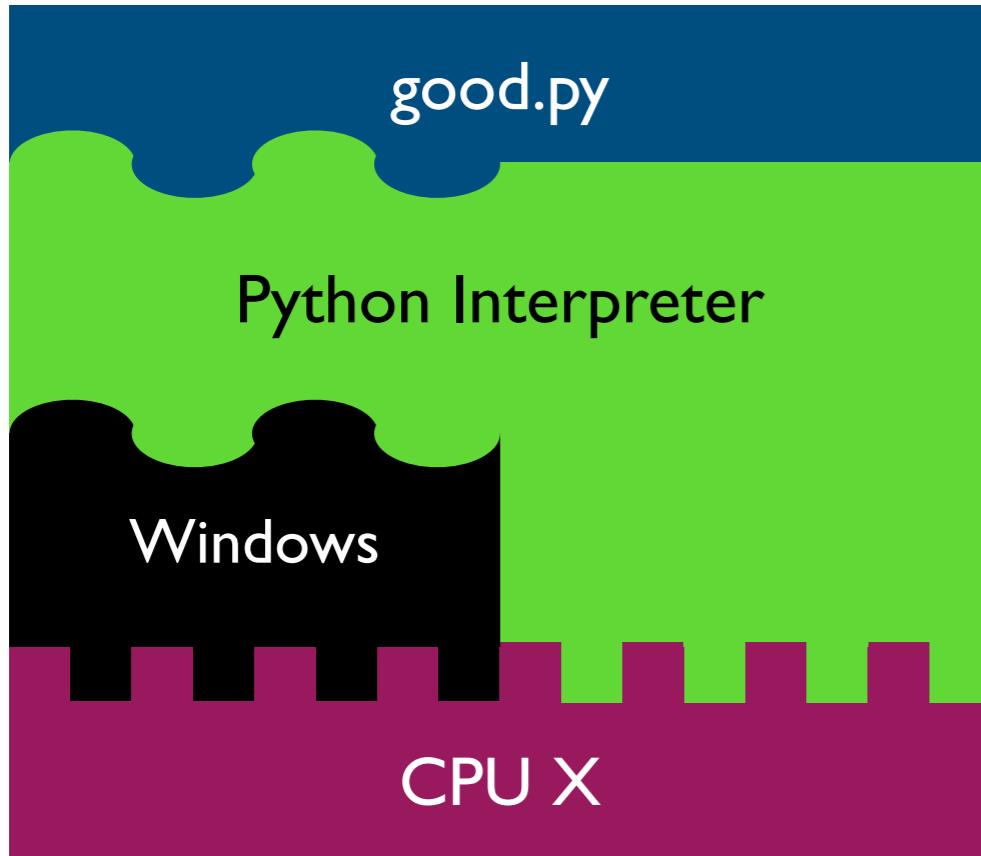


```
f = open( "c:\\data\\file.txt" )  
...
```

The Python interpreter mostly lets you
[Python Programmer] ignore the CPU you run on.

But you still need to work a bit to "fit" the code to the OS.

Harder to reproduce on different OS...



solution 1:

```
f = open(os.path.join("data", "file.txt"))  
...
```

solution 2:

tell anybody reproducing your results to use the same OS!

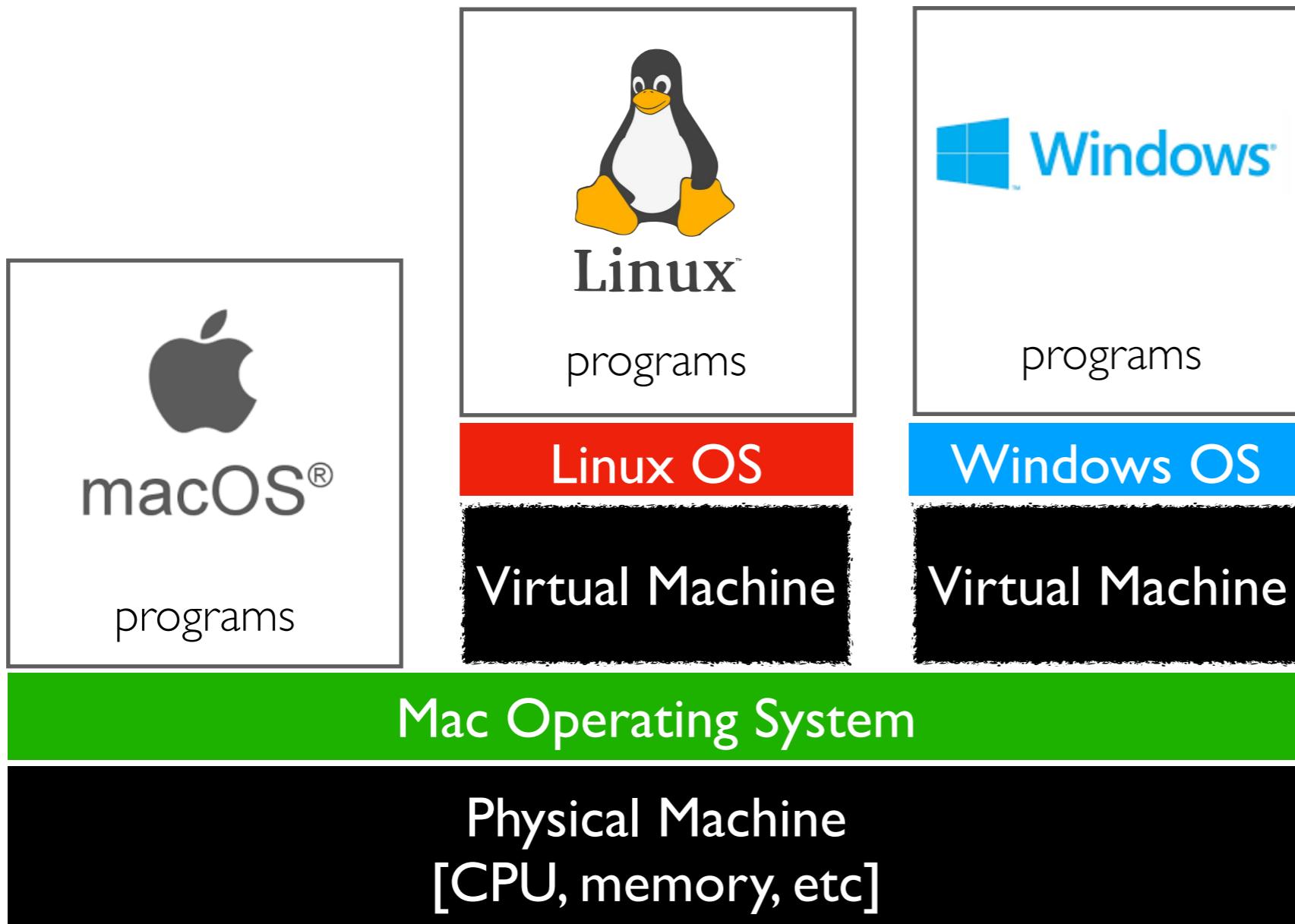
tradeoffs?

The Python interpreter mostly lets you
[Python Programmer] ignore the CPU you run on.

But you still need to work a bit to "fit" the code to the OS.

VMs (Virtual Machines)

popular virtual
machine software

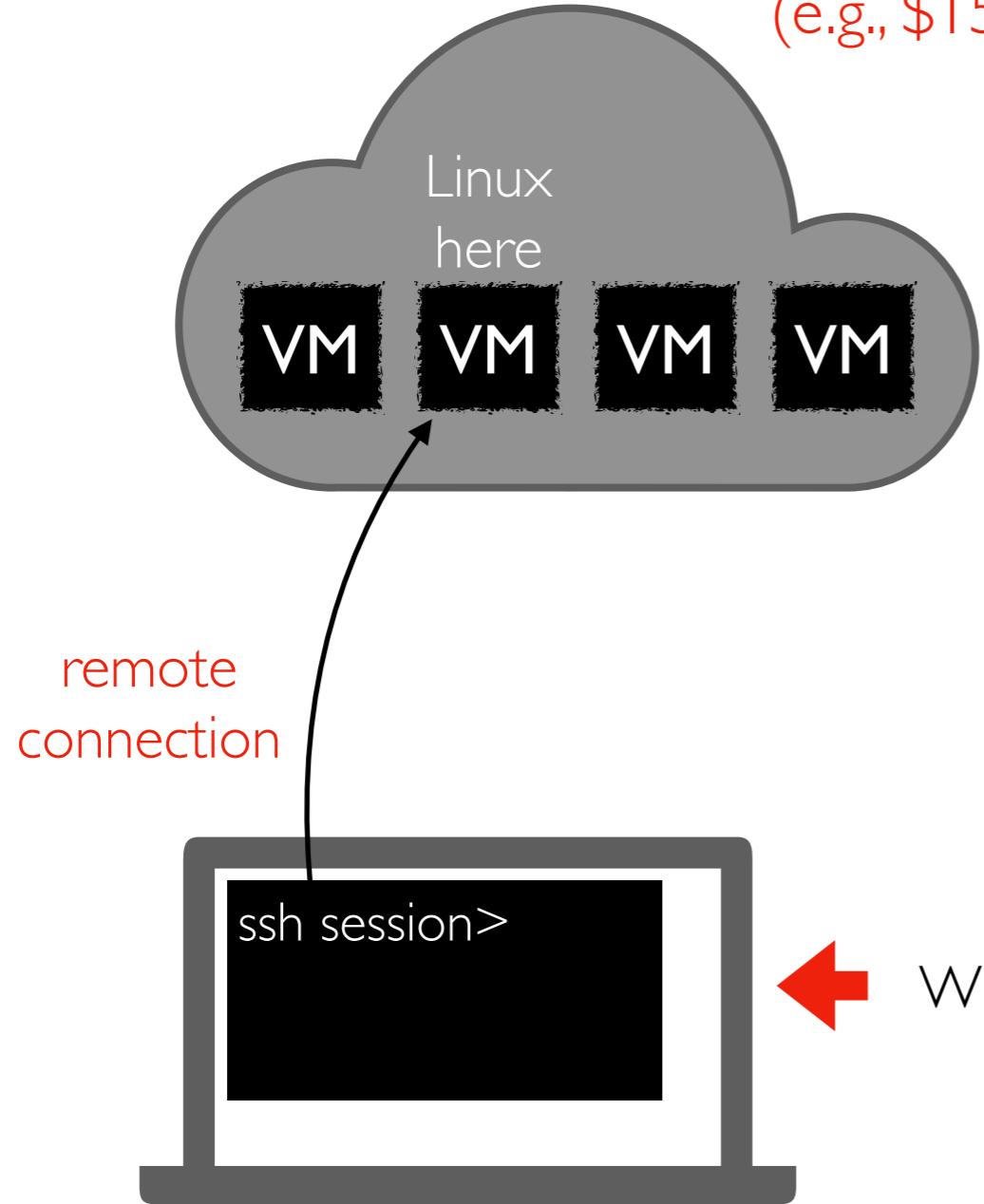


Microsoft
Hyper-V

With the right virtual machines created and operating systems installed, you could run programs for Mac, Linux, and Windows -- at the same time without rebooting!

The Cloud

cloud providers let you rent VMs
in the cloud on hourly basis
(e.g., \$15 / month)



`ssh user@best-linux.cs.wisc.edu` ← run in PowerShell/bash
to access CS lab

popular cloud providers



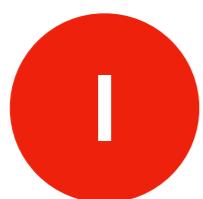
Google Cloud Platform

we'll use GCP virtual
machines this semester
[setup in lab]

Lecture Recap: Reproducibility

Big question: *will my program run on someone else's computer?*

Things to match:



Hardware ← a program must fit the CPU;
`python.exe` will do this, so
`program.py` won't have to



Operating System ← we'll use Ubuntu Linux on
virtual machines in the cloud



Dependencies ← next time: versioning

Recap of 15 new terms

reproducibility: others can run our analysis code and get same results

process: a running program

byte: integer between 0 and 255

address space: a big "list" of bytes, per process, for all state

address: index in the big list

encoding: pairing of ~~letters~~ characters with numeric codes

CPU: chip that executes instructions, tracks position in code

instruction set: pairing of CPU instructions/ops with numeric codes

operating system: software that allocates+abstracts resources

resource: time on CPU, space in memory, space on SSD, etc

allocation: the giving of a resource to a process

abstraction: hiding inconvenient details with something easier to use

virtual machine: "fake" machine running on ~~real~~ physical machine
allows us to run additional operating systems

cloud: place where you can rent virtual machines and other services

ssh: secure shell -- tool that lets you remotely access another machine