[320] Welcome + First Lecture

[reproducibility]

Tyler Caraza-Harter

Introductions

Tyler Caraza-Harter

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

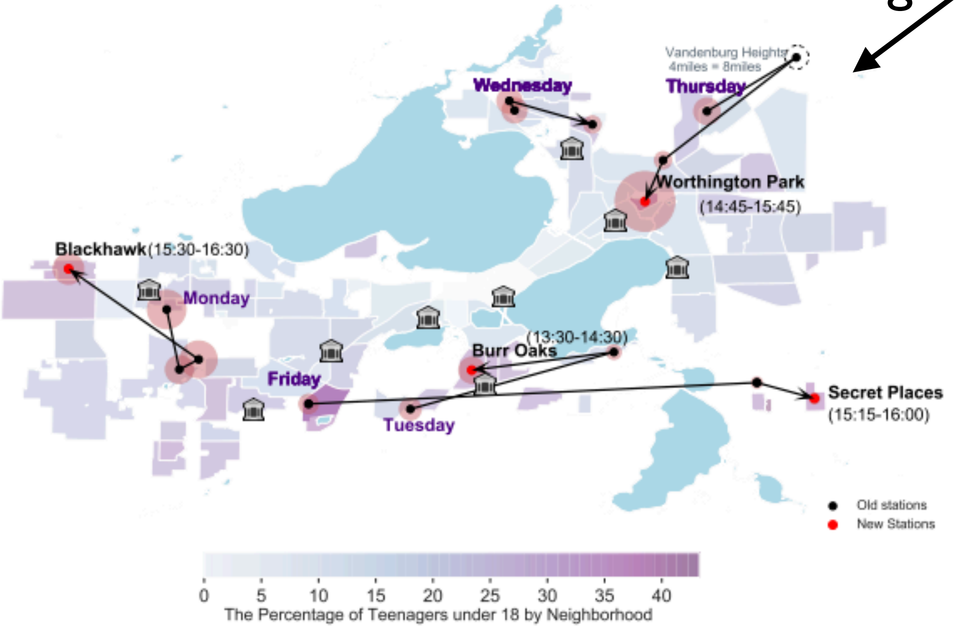
Industry experience

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

interests

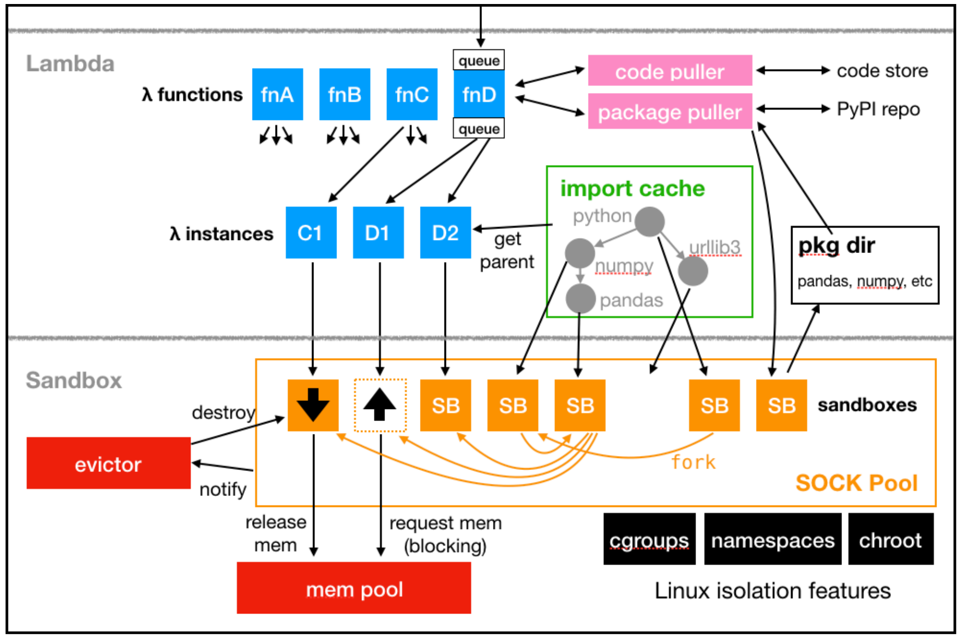
civic "hacking"

OpenLambda



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?

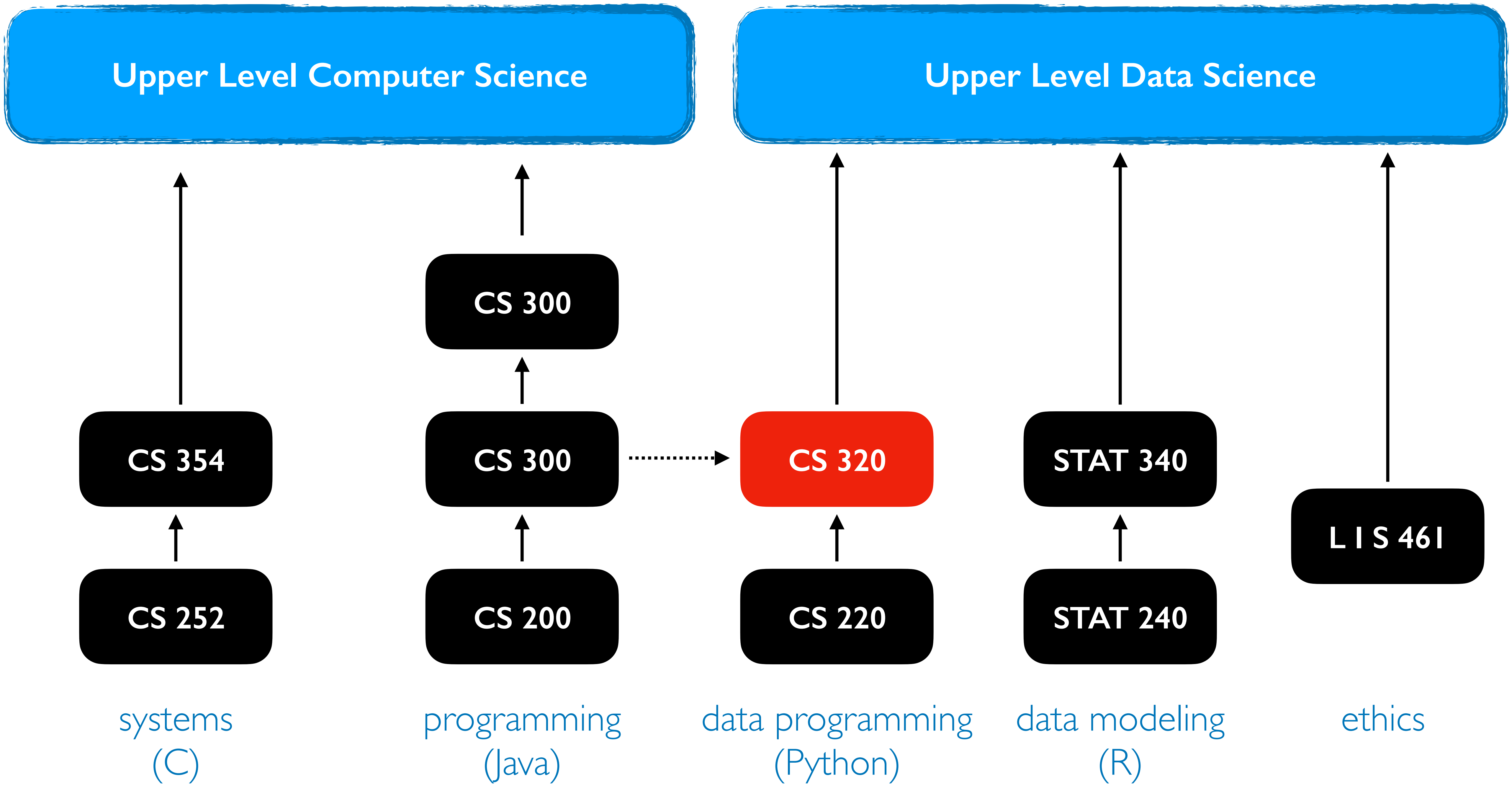
What CS courses have people taken before?

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

Please fill this form (**due today**):
https://docs.google.com/forms/d/e/1FAIpQLSfz7K0cY2-VGCtxE4TQ-zkcbcWTtzyLZQXCrgLyp6EfwU2jDg/viewform?usp=sf_link.
Why?

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

Related courses



Upper Level Computer Science

Upper Level Data Science

CS 300

CS 354

CS 300

CS 320

STAT 340

L I S 461

CS 252

CS 200

CS 220

STAT 240

systems (C)

programming (Java)

data programming (Python)

data modeling (R)

ethics

P1 (Project 1) will help 300-to-320 students pickup Python.

Welcome to Data Programming II!

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

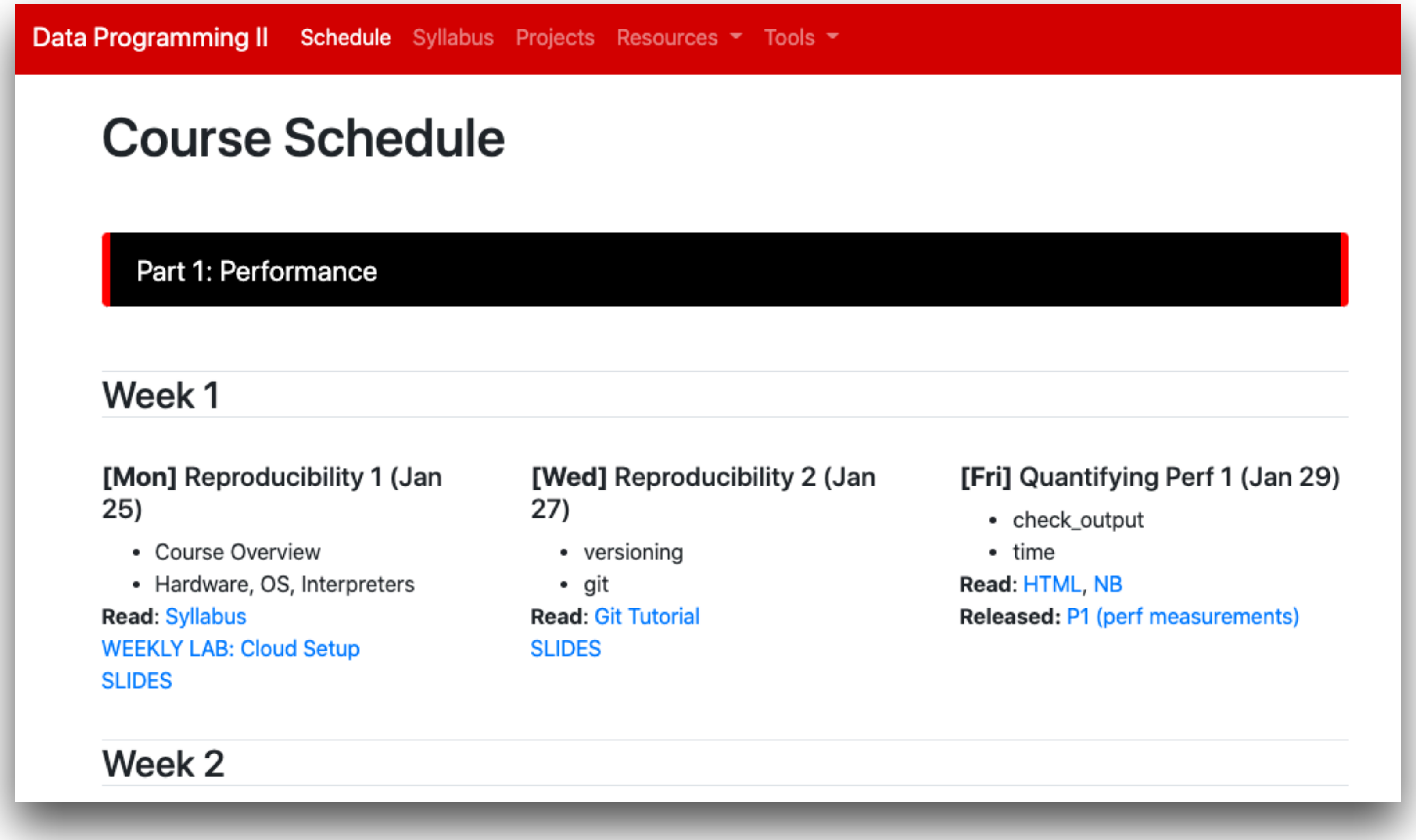
CS 220	CS 320
getting results	getting **reproducible** results
writing correct code	writing **efficient** code
using objects	designing **new types** of objects
functions: `f(obj)`	**methods**: `obj.f()`
lists+dicts	graphs+trees
analyzing datasets	**collecting**+analyzing datasets
plots	animated visualizations
tabular analysis	**simple machine learning**

CS 301 content (for review): https://tyler.caraza-harter.com/cs301/fall19/schedule.html

Course Logistics

Course Website

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



Course Schedule

Part 1: Performance

Week 1

[Mon] Reproducibility 1 (Jan 25)

- Course Overview
- Hardware, OS, Interpreters

Read: Syllabus
WEEKLY LAB: Cloud Setup
SLIDES

[Wed] Reproducibility 2 (Jan 27)

- versioning
- git

Read: Git Tutorial
SLIDES

[Fri] Quantifying Perf 1 (Jan 29)

- check_output
- time

Read: HTML, NB
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

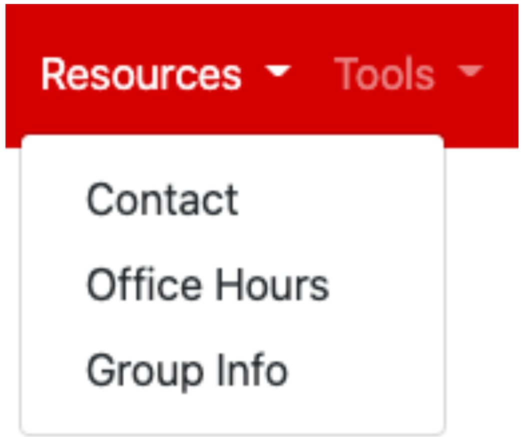
- you'll be assigned to a team of 4-7 students
- teams will last the whole semester
- some types of collaboration with team members are allowed (not required) on graded work, such as projects+quizzes
- most collaboration with non-team members in not allowed

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 in not allowed

Staff

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

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

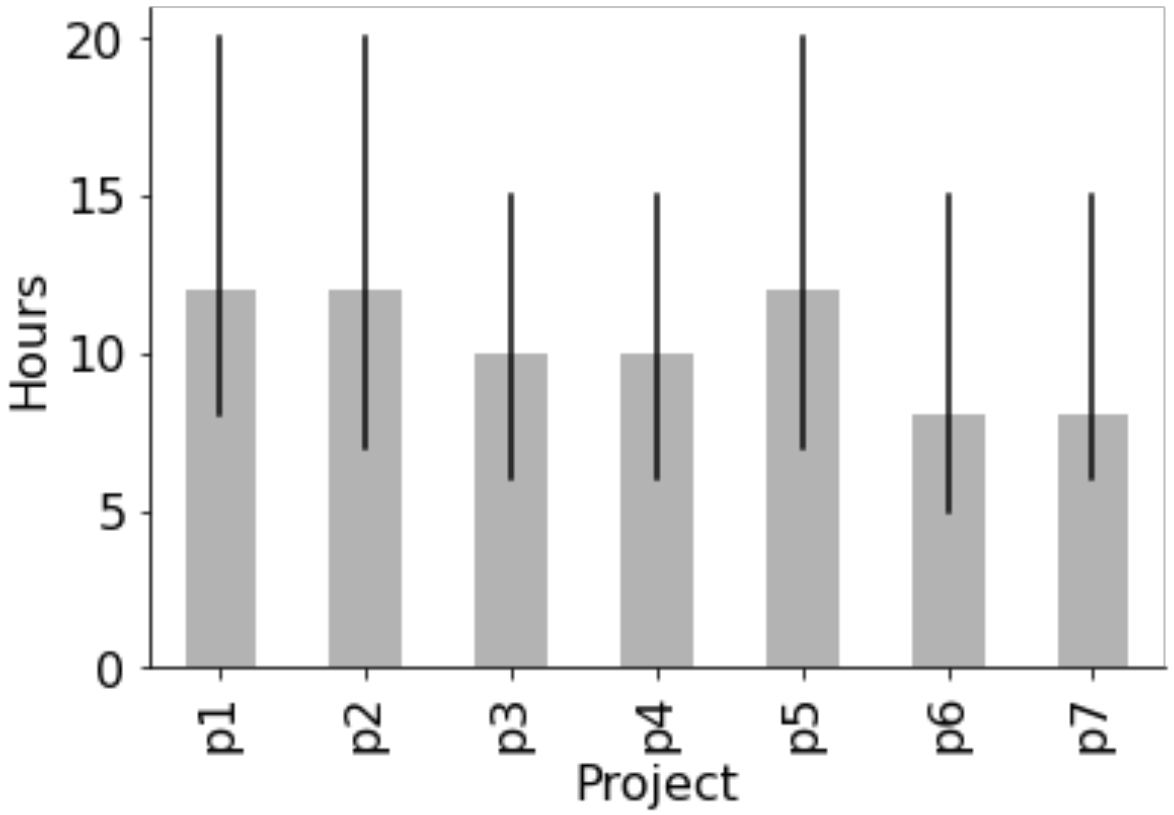
7 Projects - 6% each

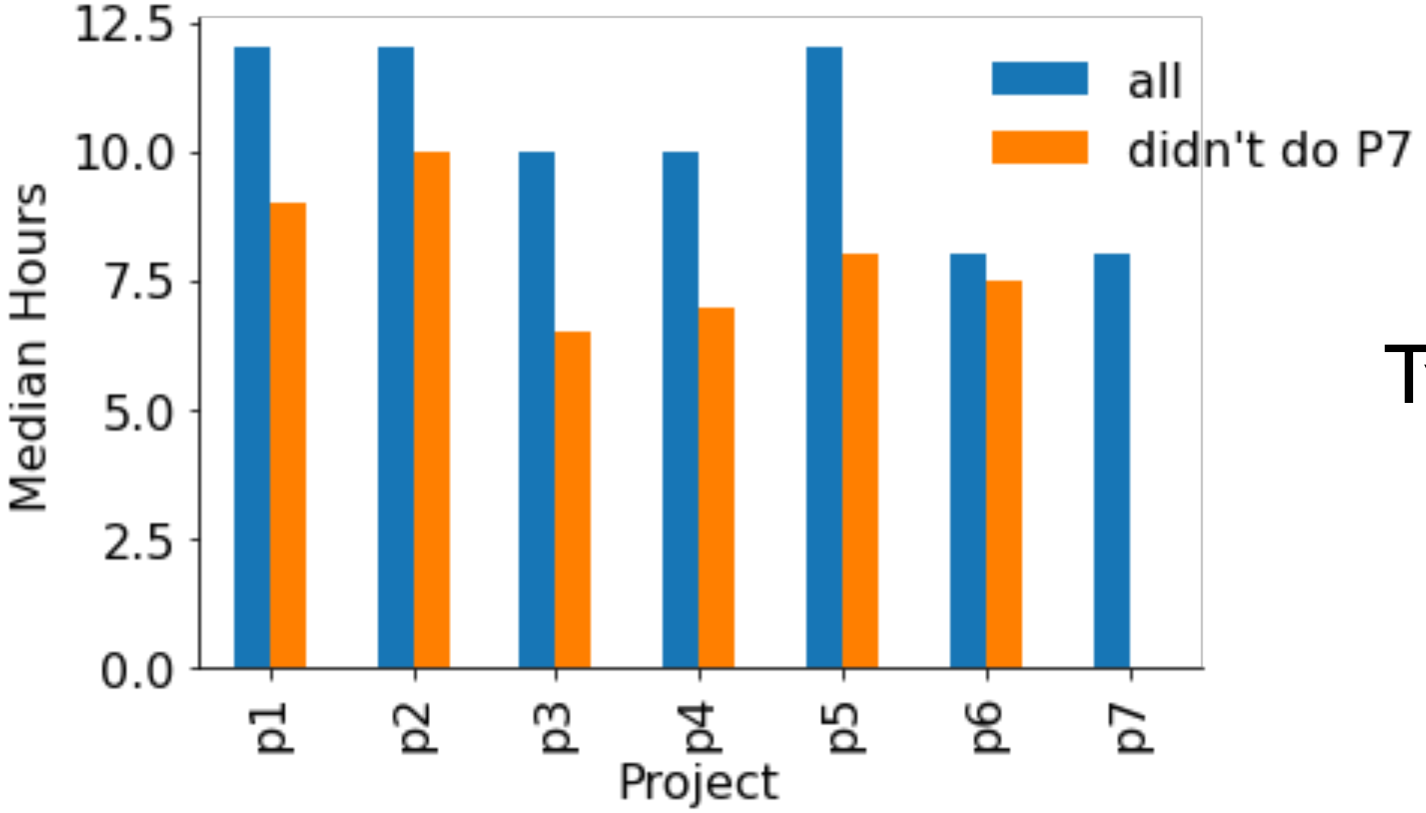
- **format**: notebook, module, or program
- part 1: you can optionally collaborate with team
- part 2: must be individually (only help from 320 staff)
- still a `tester.py`, but more depends on TA evaluation (more plots)
- ask for specific feedback
  (giving constructive criticism is a priority in CS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

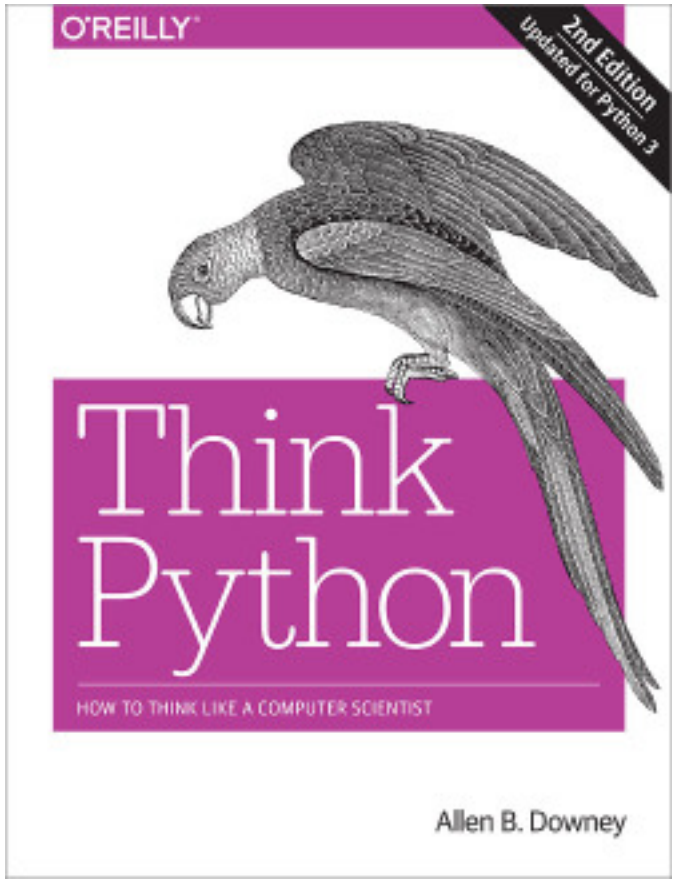
- 58 students for cheating on projects
- 2 past students for sharing solutions from past semesters
- 8 students for cheating on exams
- 1 student for faking participation

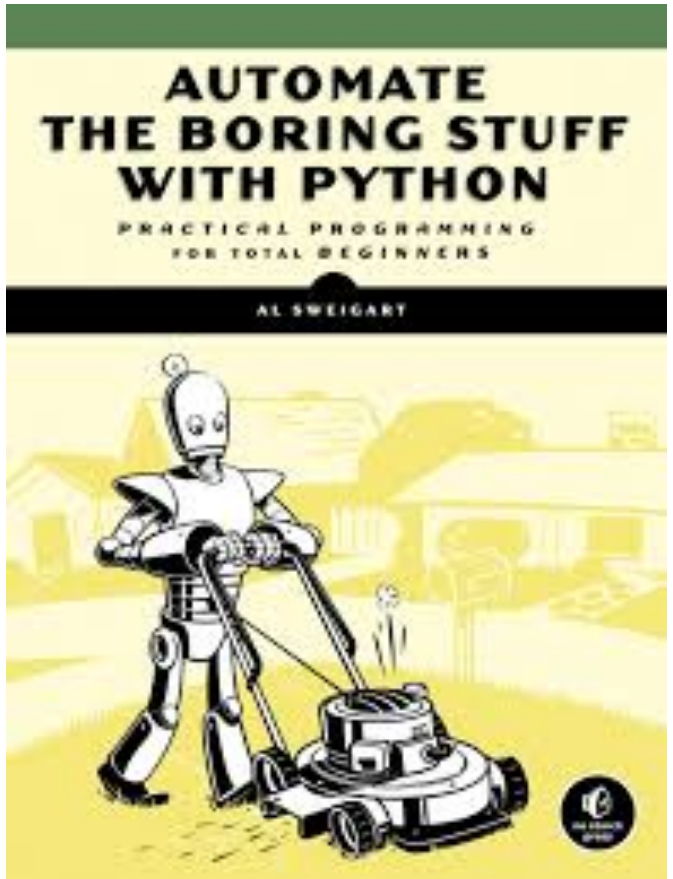
How we'll keep the class fair

- 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...




AUTOMATE THE BORING STUFF WITH PYTHON
PRACTICAL PROGRAMMING FOR TOTAL BEGINNERS
AL SWEIGART

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

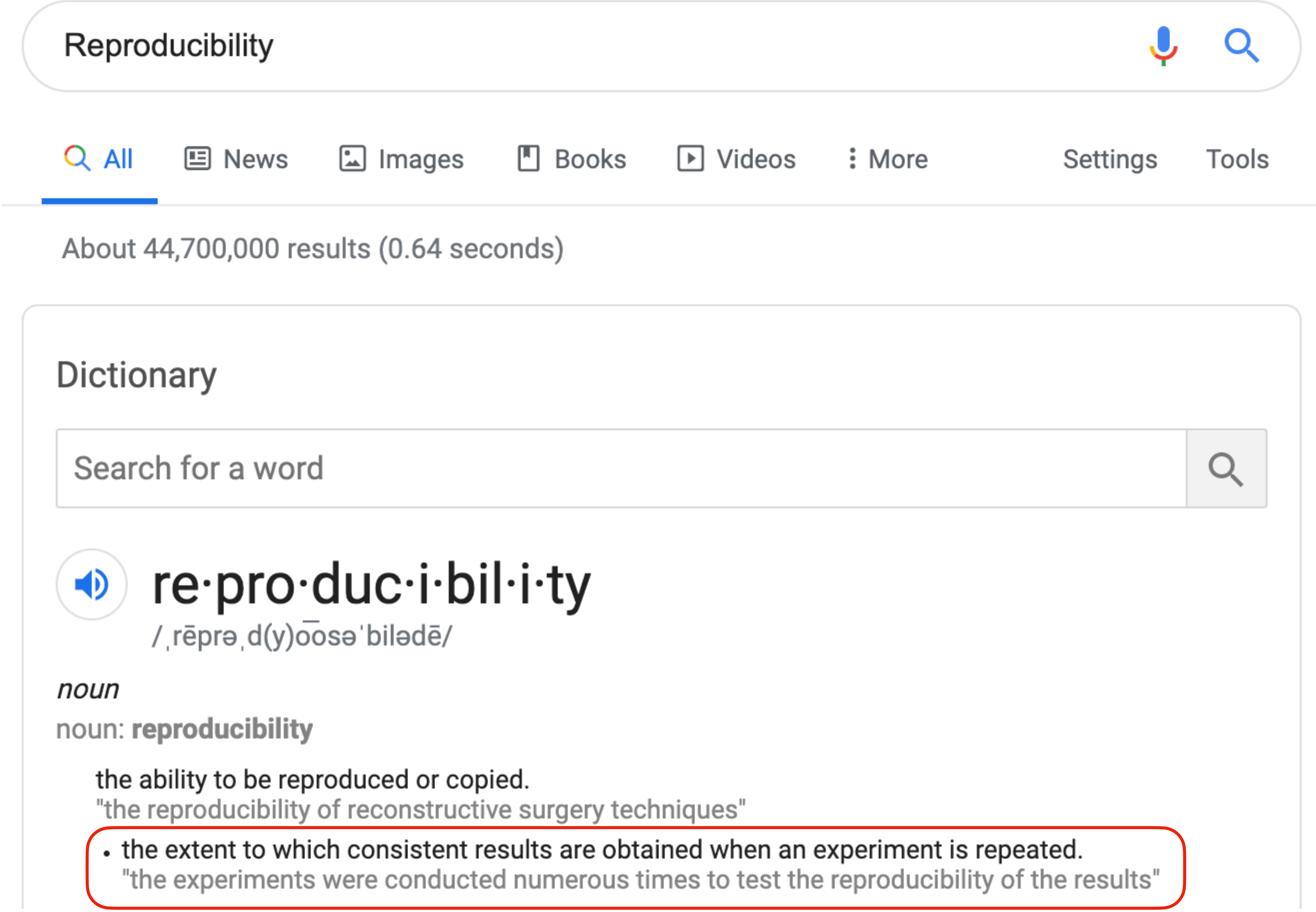
6. Run the tester often

7. If you're struggling, reach out -- the sooner, the better

Any questions?

Today's Lecture:

Reproducibility



About 44,700,000 results (0.64 seconds)

Dictionary

re·pro·duc·i·bil·i·ty

/ˌrēprəˌd(y)o͞osəˈbilədē/

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?*

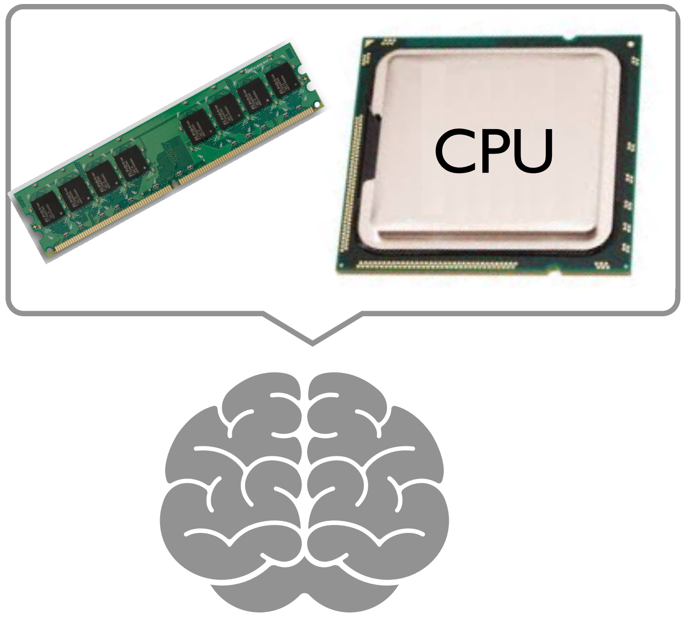
(not necessarily written in Python)

Things to match:

1 Hardware

2 Operating System

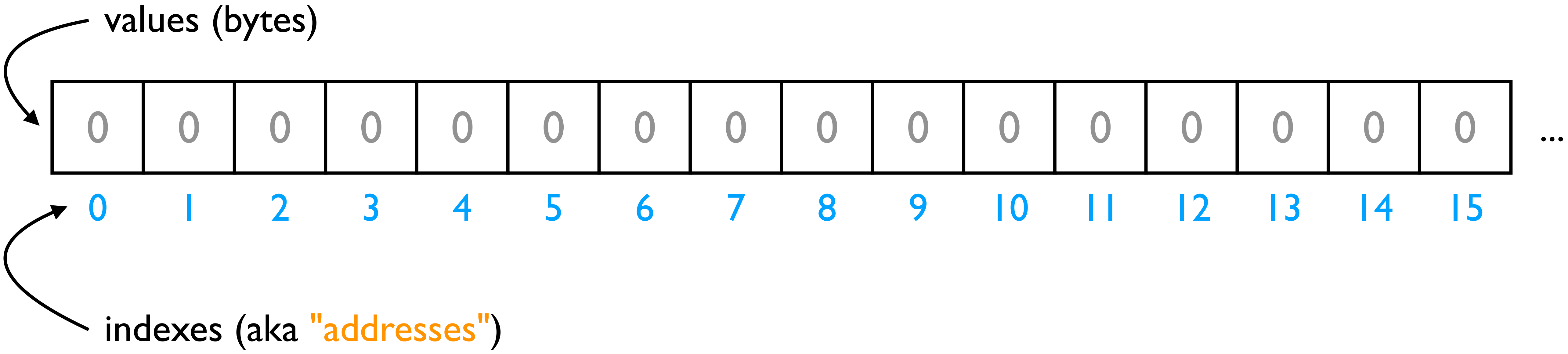
3 Dependencies ← next lecture



Hardware: Mental Model of Process Memory

Imagine...

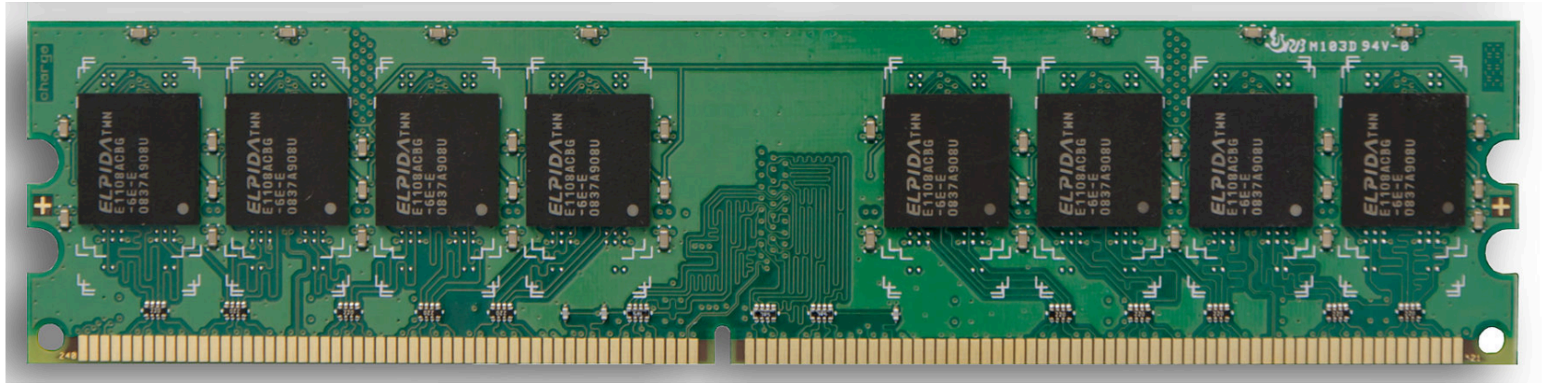
- one huge list, **per each** ~~running program~~ process, called "address space"
- every entry in the list is an integer between 0 and 255 (aka a "byte")



values (bytes)

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

indexes (aka "addresses")



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

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

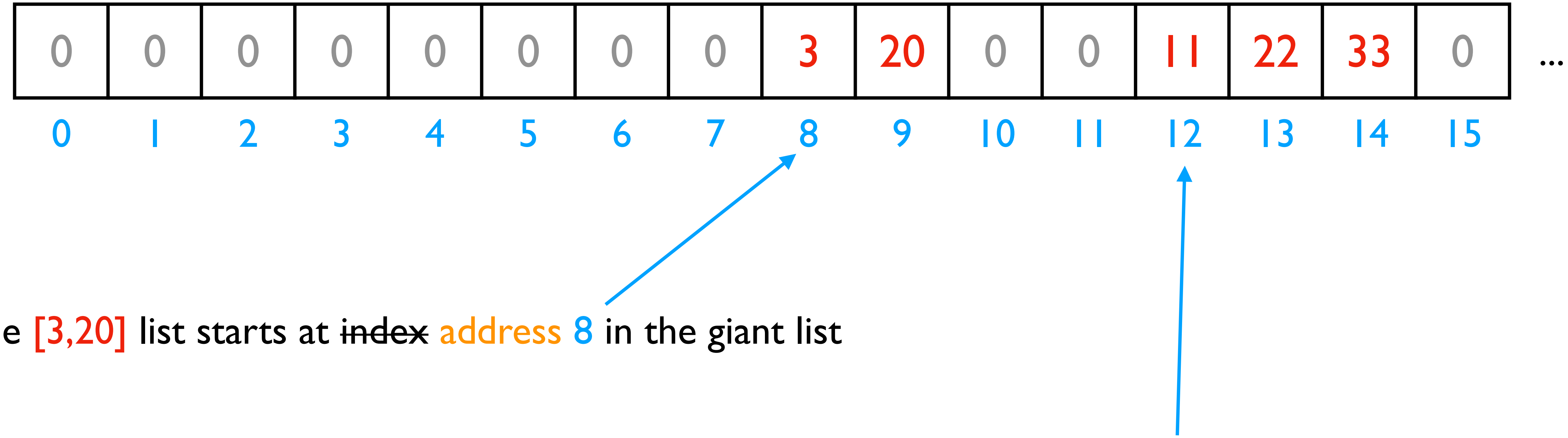
data

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

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



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

the [3,20] list starts at ~~index~~ address 8 in the giant list

the [11,22,33] list starts at address 12 in the giant list

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

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

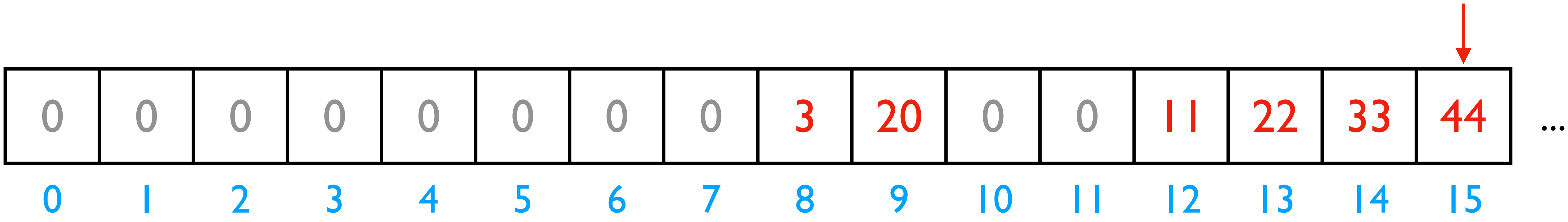
0	0	0	0	0	0	0	0	3	20	0	0	11	22	33	0	...
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	

```
# 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



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	0	0	0	0	0	3	20	0	0	11	22	33	44	...

```
# 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

0	0	0	0	0	0	0	0	3	20	0	0	11	22	33	44	...
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	

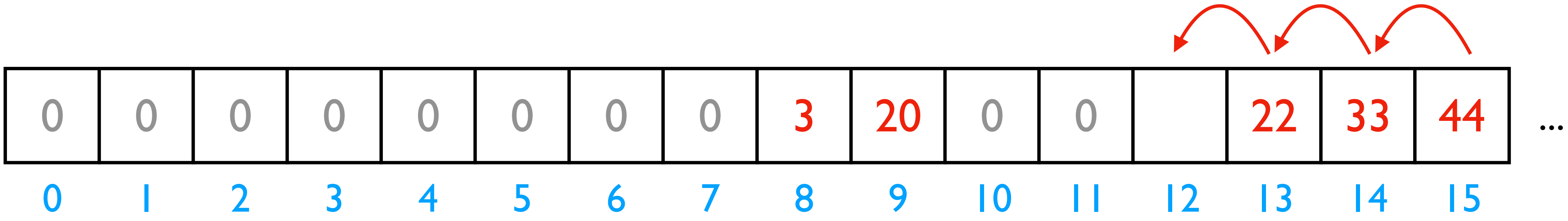
implications for performance...

```
# 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



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	3	20	0	0		22	33	44

...

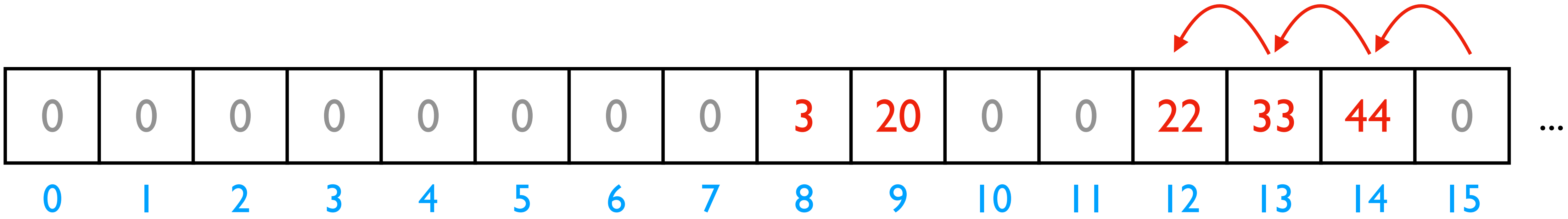
implications for performance...

```
# 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



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	3	20	0	0	22	33	44	0

...

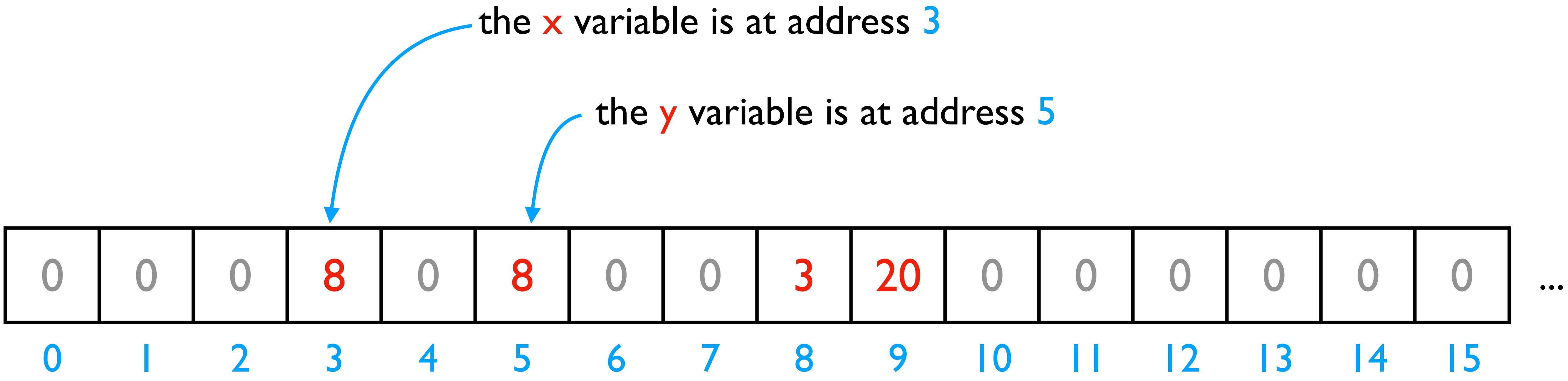
We'll think more rigorously about performance in CS 320 (big-O notation)

```
# fast
L2.append(44)

# slow
L2.pop(0)
```

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

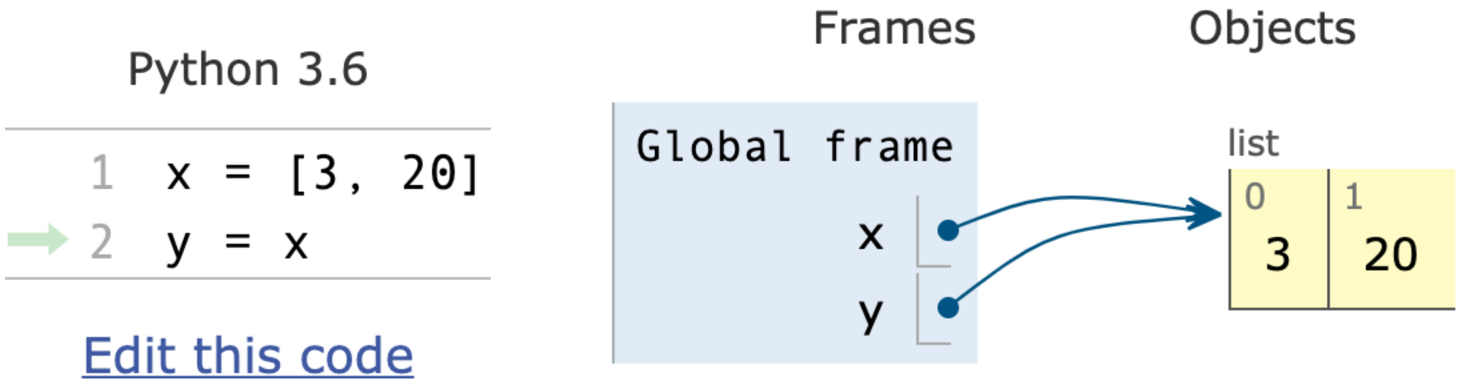
- multiple lists
- variables and other references
- strings
- code



the x variable is at address 3

the y variable is at address 5

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



```
x = [3, 20]
y = x
```

PythonTutor's visualization

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

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

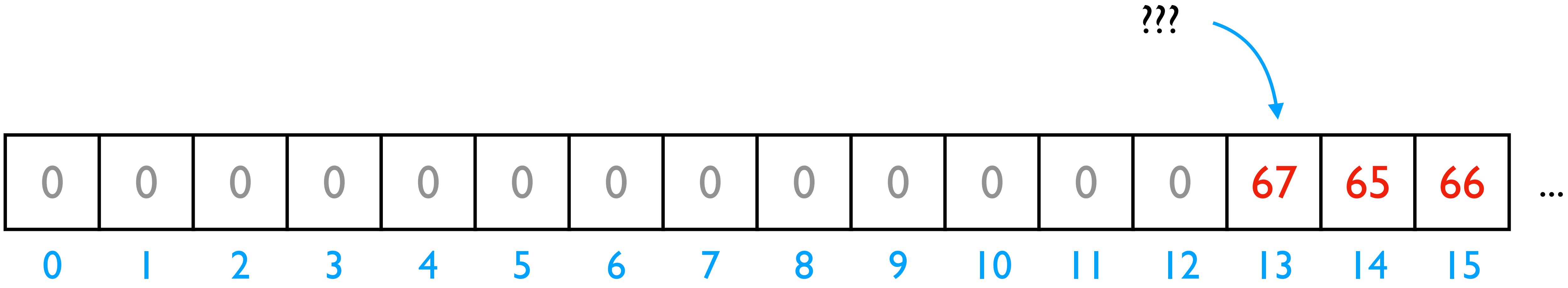
discuss: how?

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

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



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	67	65	66

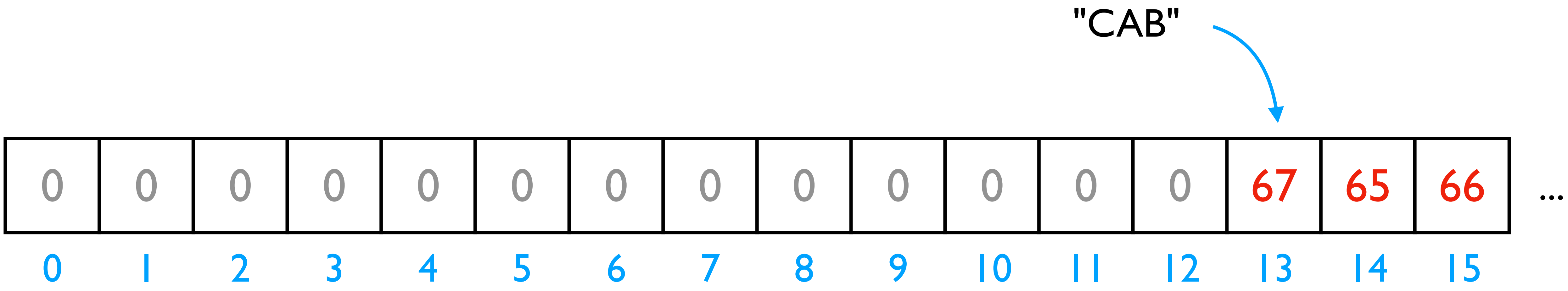
encoding:

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



"CAB"

0	0	0	0	0	0	0	0	0	0	0	0	0	67	65	66	...
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	

encoding:

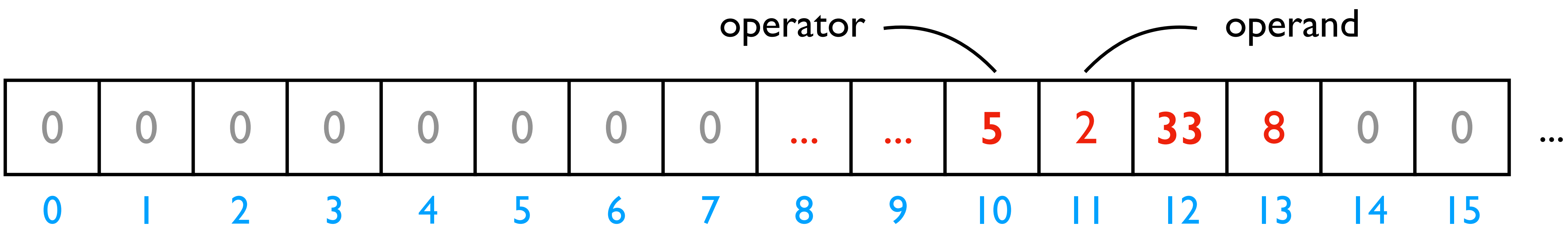
code	letter
65	A
66	B
67	C
68	D
...	...

```
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?
```



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

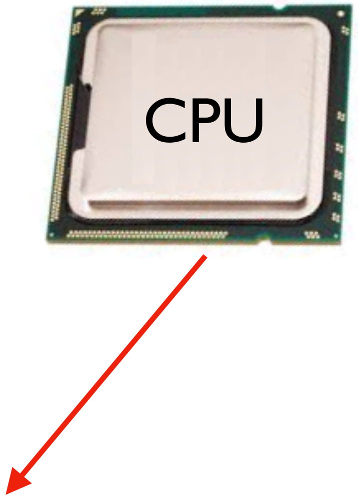
Instruction Set

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

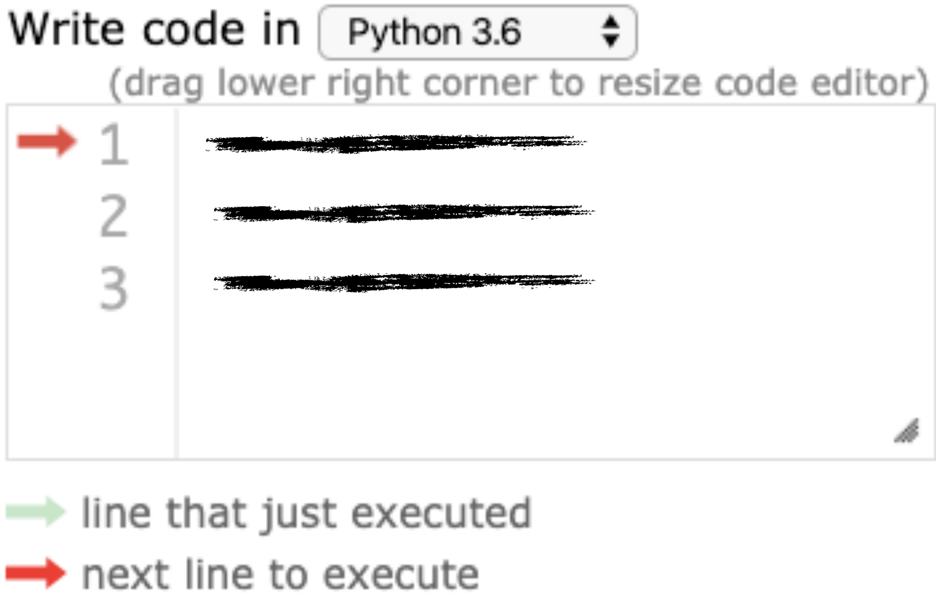
Hardware: Mental Model of CPU

CPUs interact with memory:

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



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



Write code in Python 3.6

(drag lower right corner to resize code editor)

```
[redacted]
```

line that just executed

next line to execute

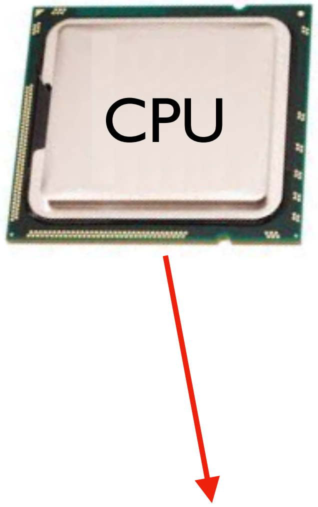
Instruction Set

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



add 2 to variable

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

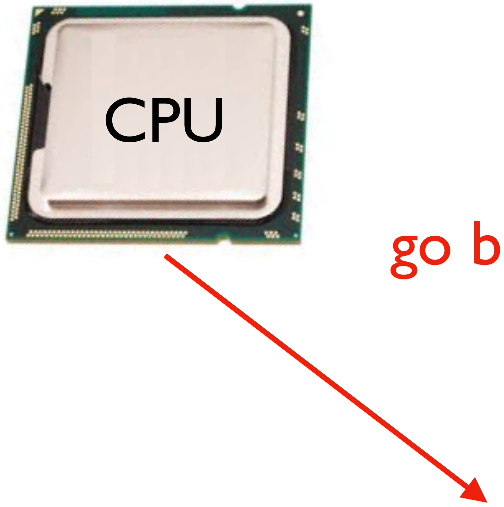
Instruction Set

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



go back to top of loop

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

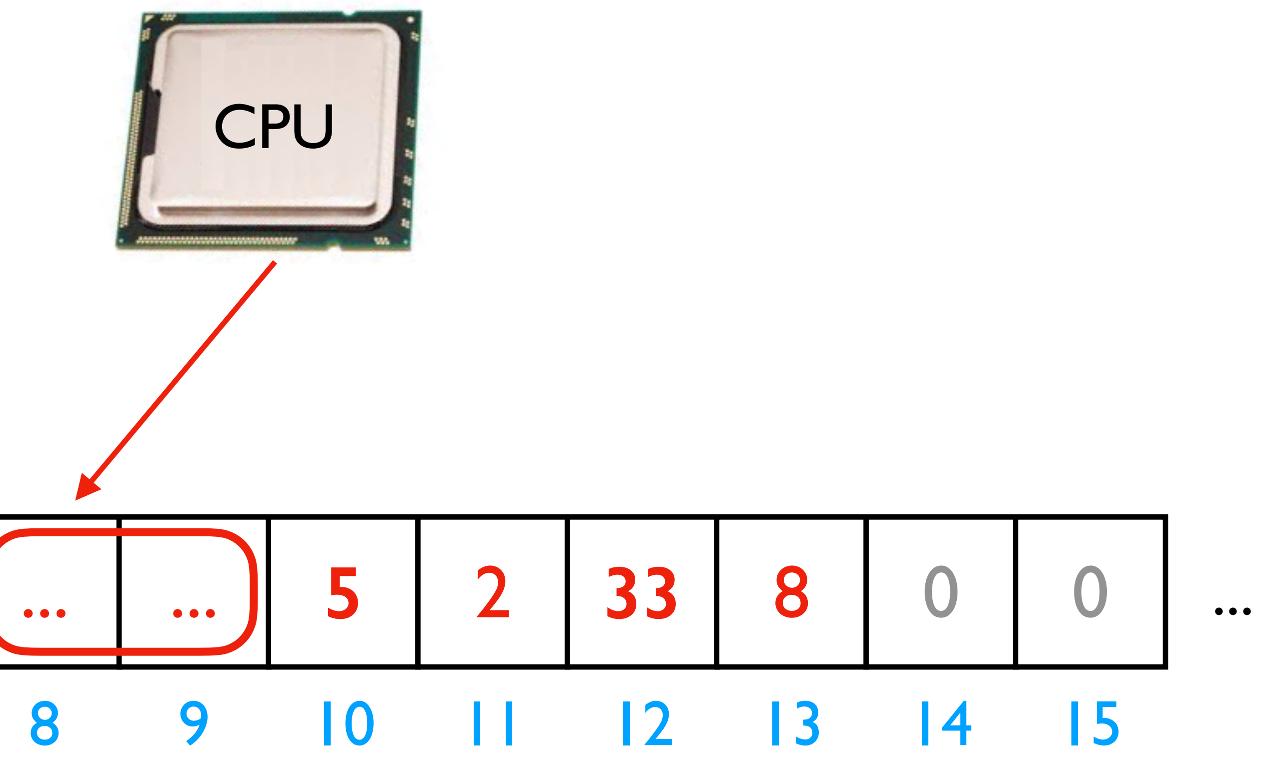
Instruction Set

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



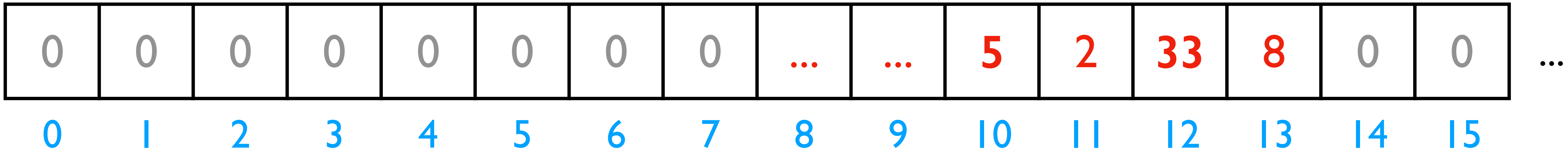
0	0	0	0	0	0	0	0	...	...	5	2	33	8	0	0	...
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	

Instruction Set

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?



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

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!

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

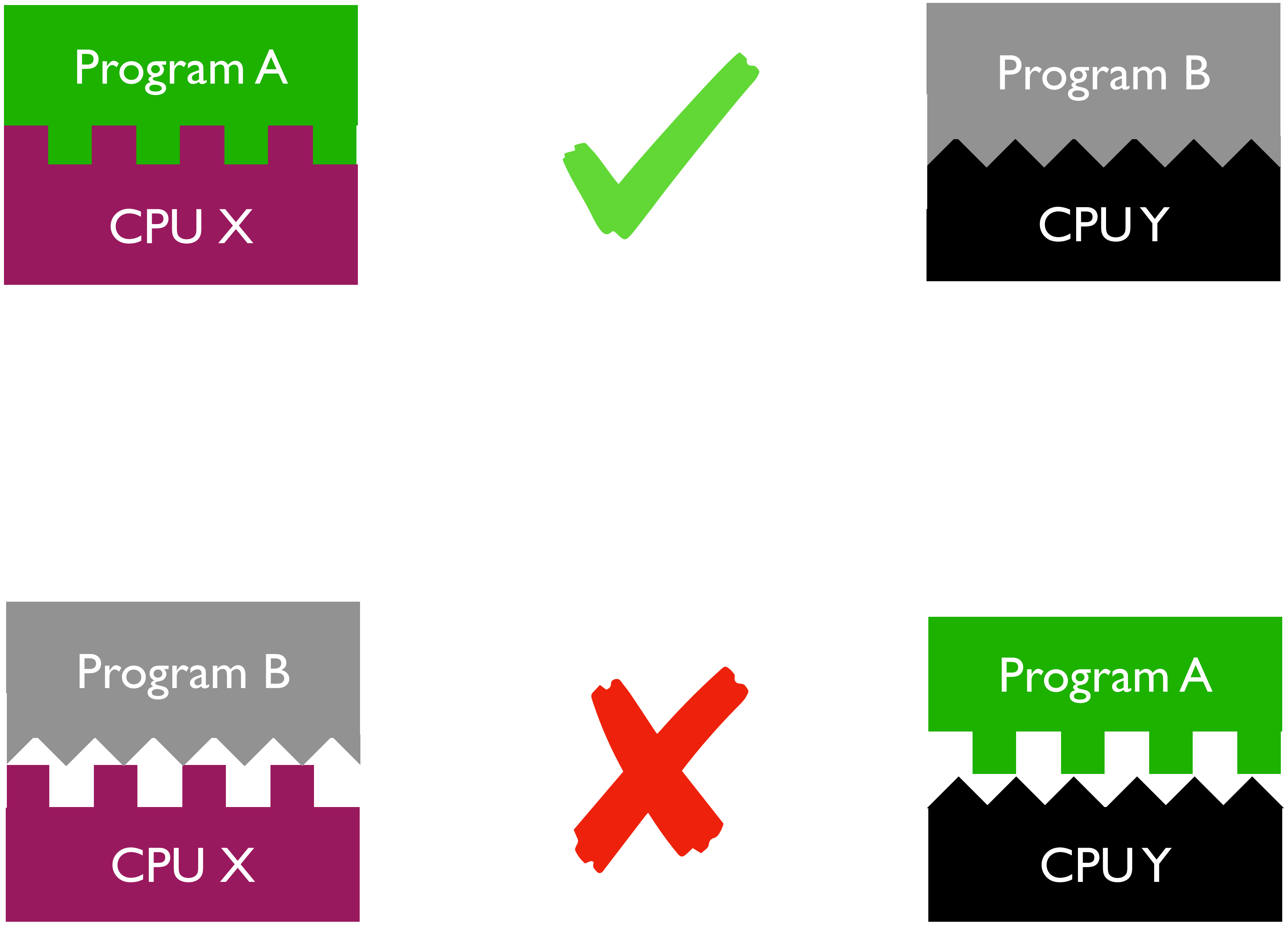
Instruction Set
for CPU X

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"



Program A

CPU X

✓

Program B

CPU Y

Program B

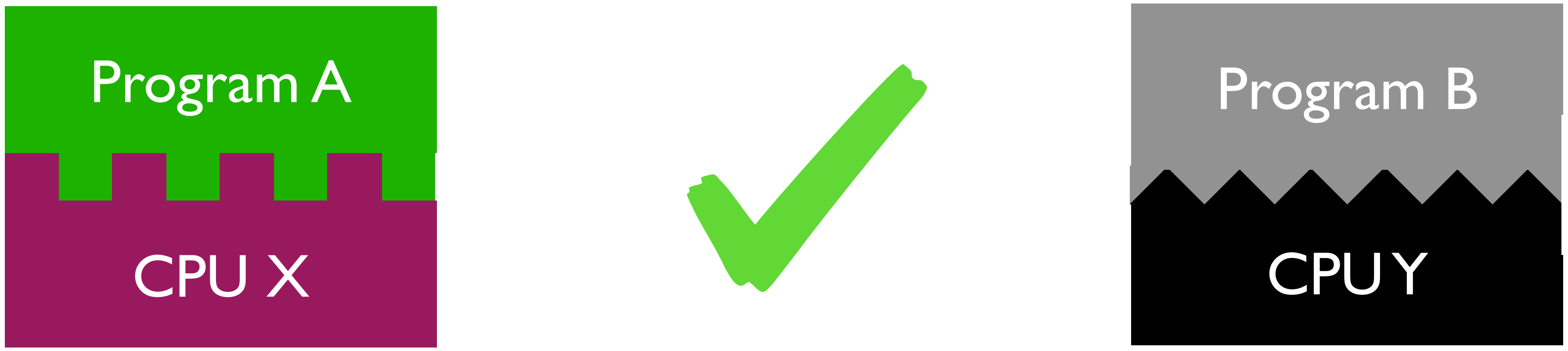
CPU X

✗

Program A

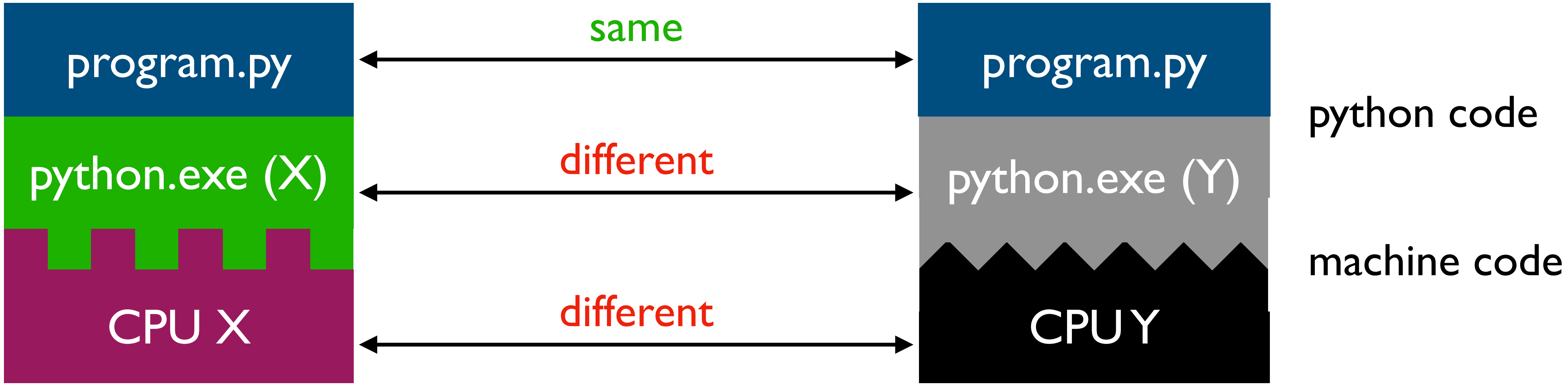
CPU Y

A Program and CPU need to "fit"



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

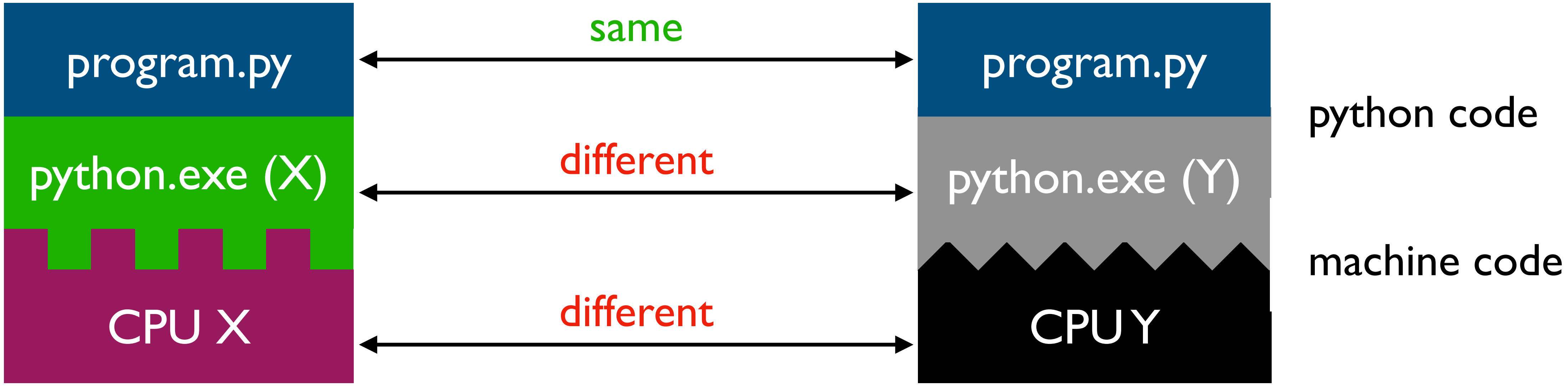
Interpreters



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?*

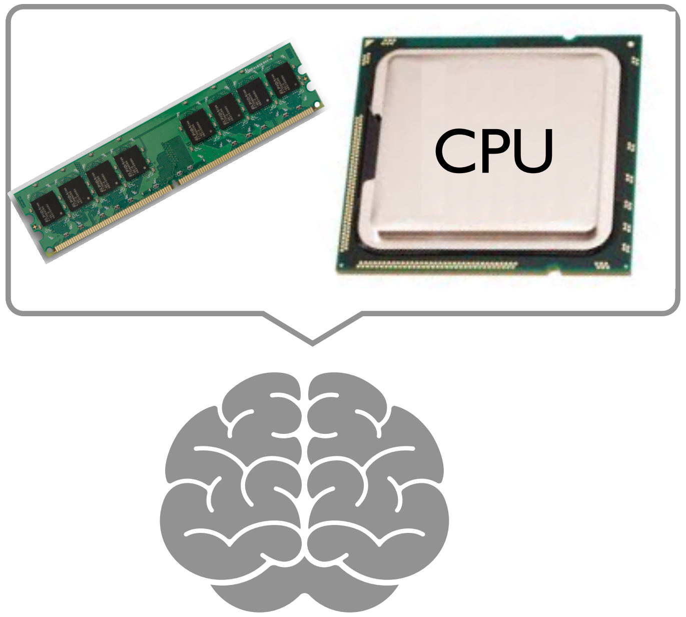
Big question: *will my program run on someone else's computer?*
(not necessarily written in Python)

Things to match:

1 Hardware

2 Operating System

3 Dependencies ← next lecture



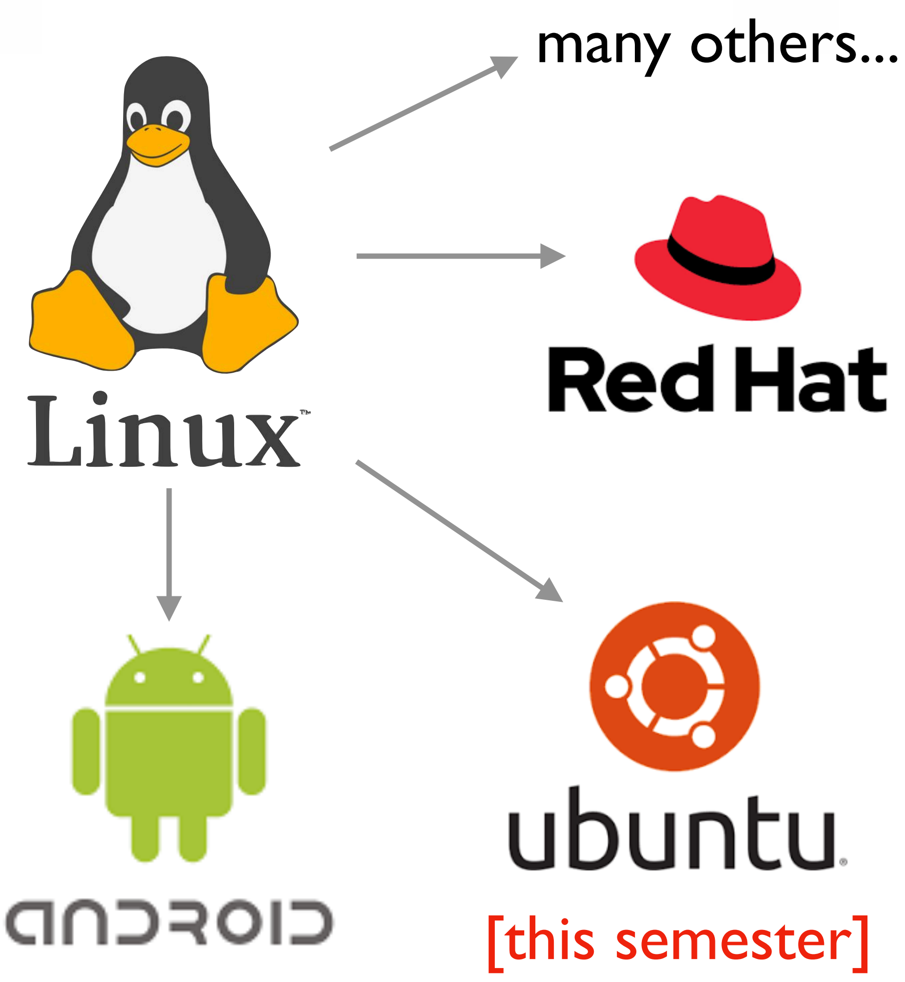
Big question: *will my program run on someone else's computer?*
(not necessarily written in Python)

Things to match:

1. Hardware
2. Operating System
3. Dependencies ← next lecture

macOS®

Windows

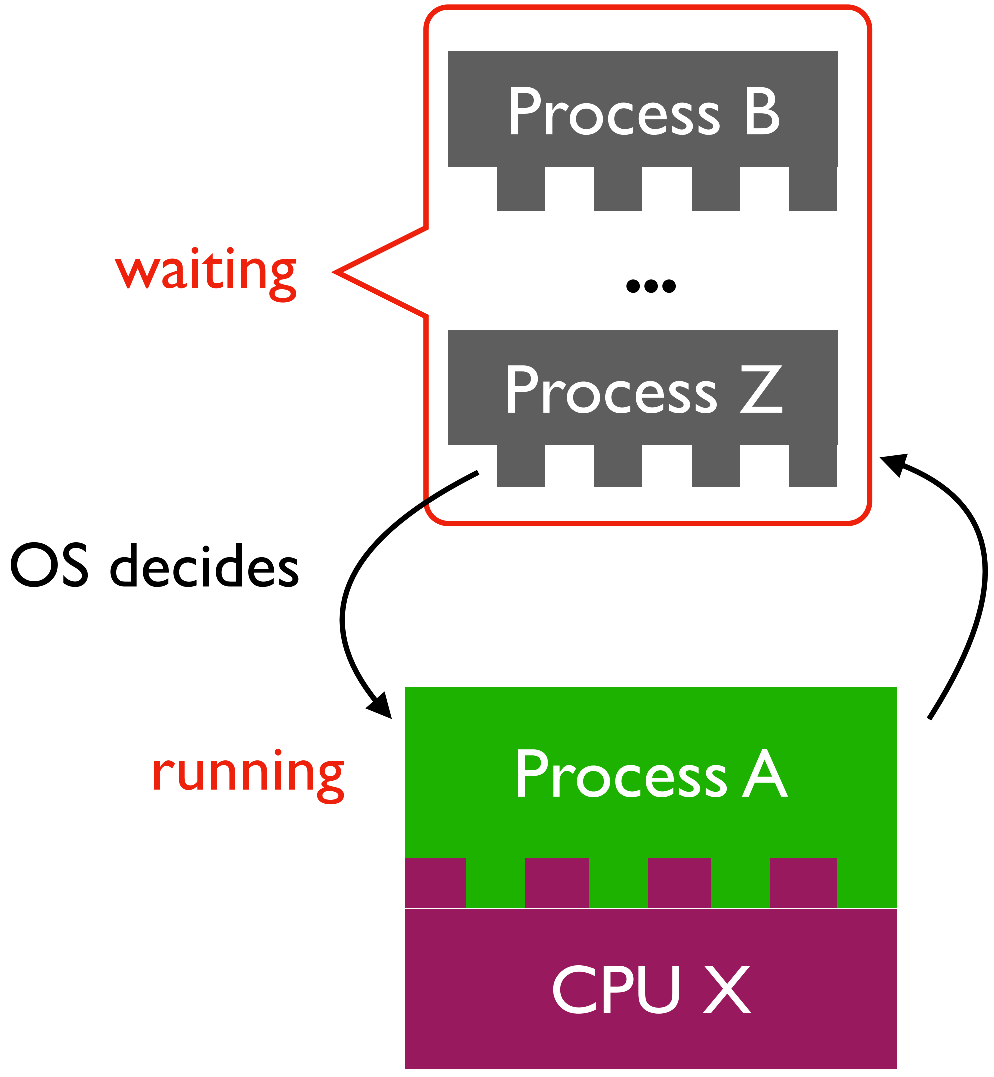


[this semester]

OS jobs: Allocate and Abstract Resources

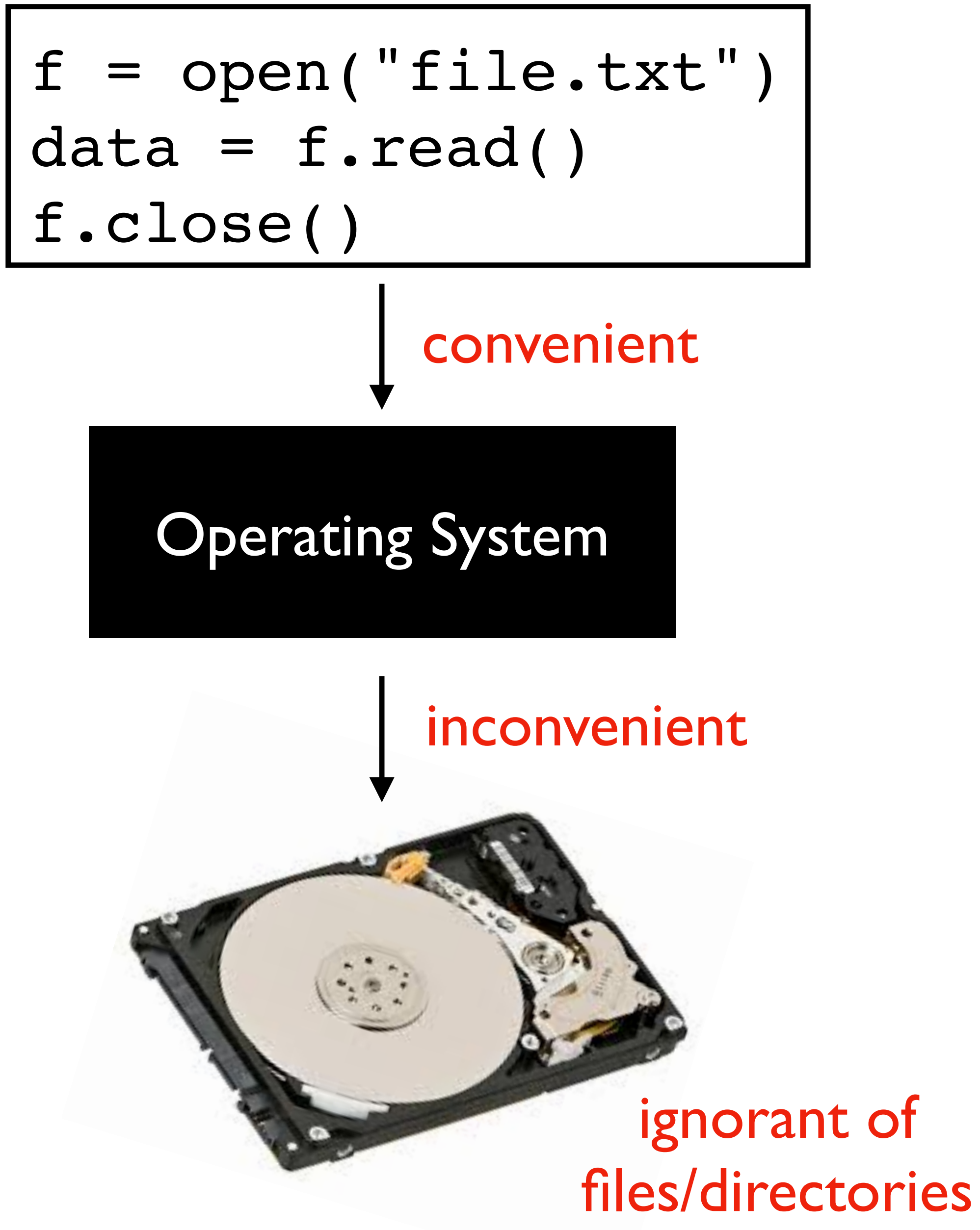
[like CPU, hard drive, etc]

1 Allocation



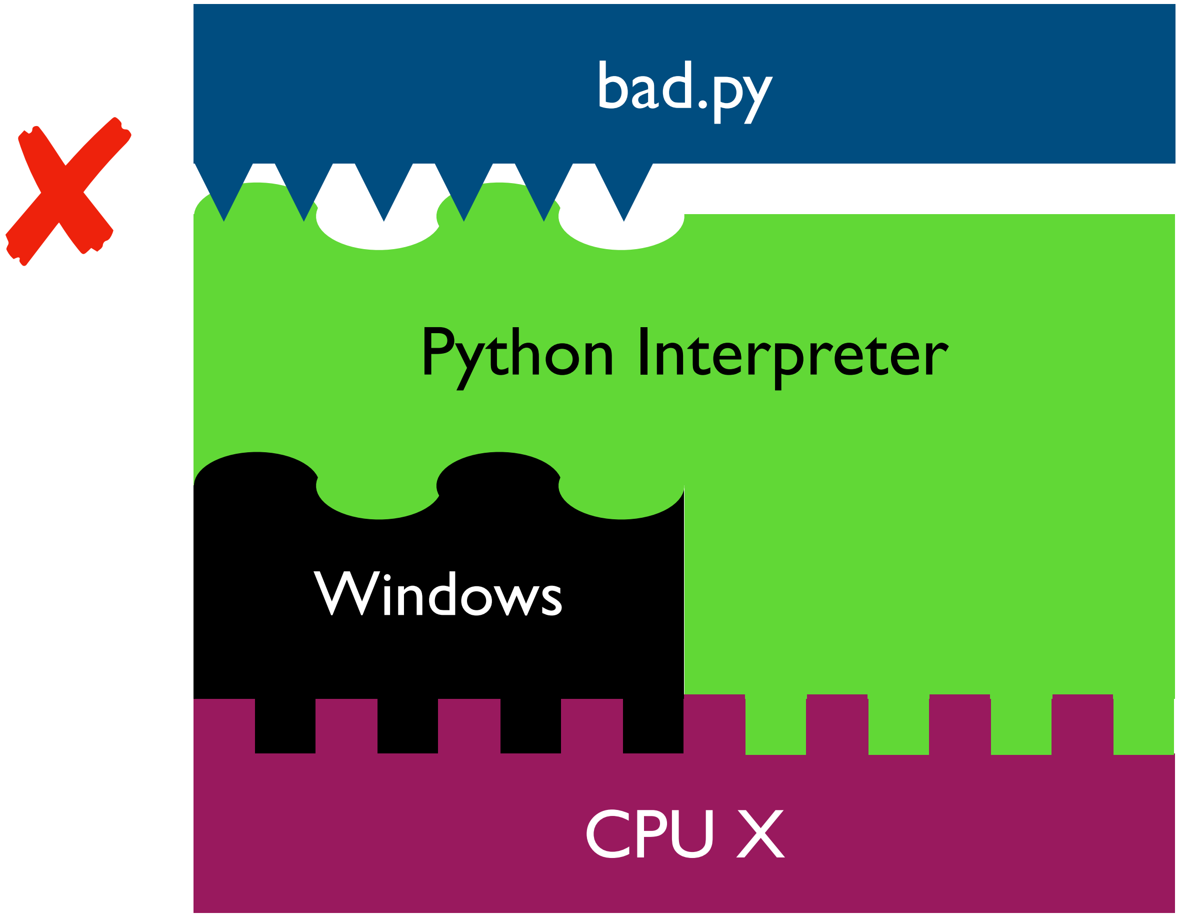
only one process can run on CPU at a time
(or a few things if the CPU has multiple "cores")

2 Abstraction



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

Harder to reproduce on different OS...



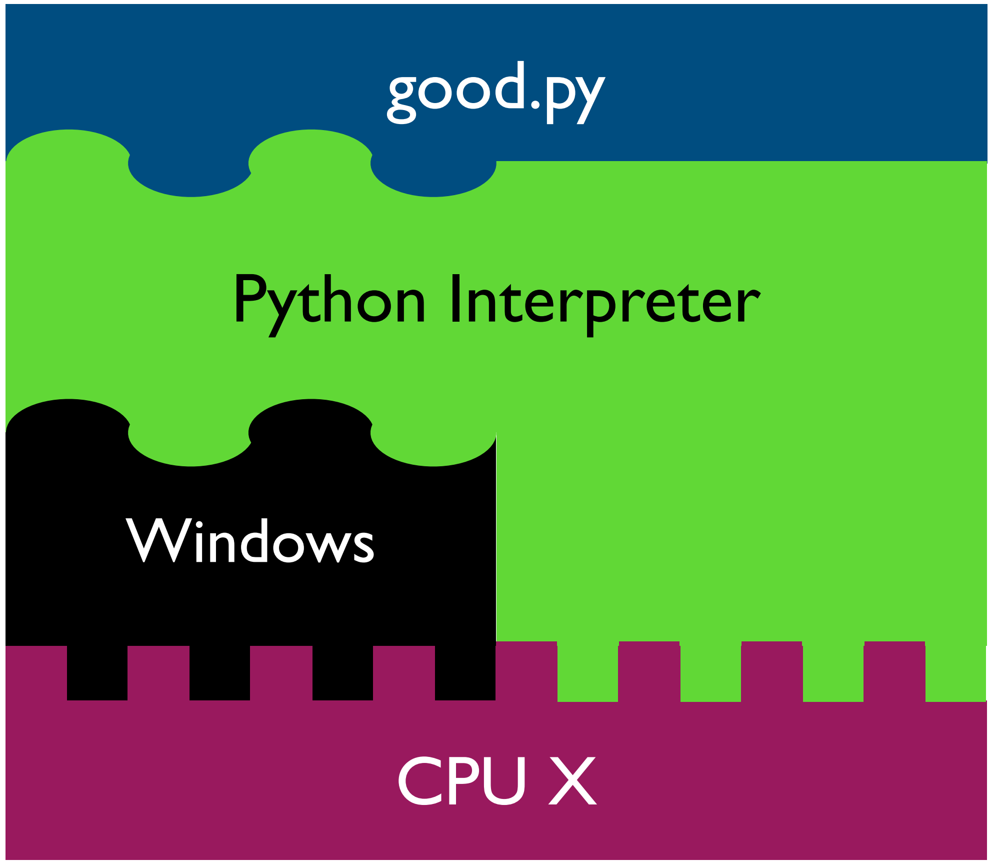
```
f = open("/data/file.txt")
...
```

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...

✓



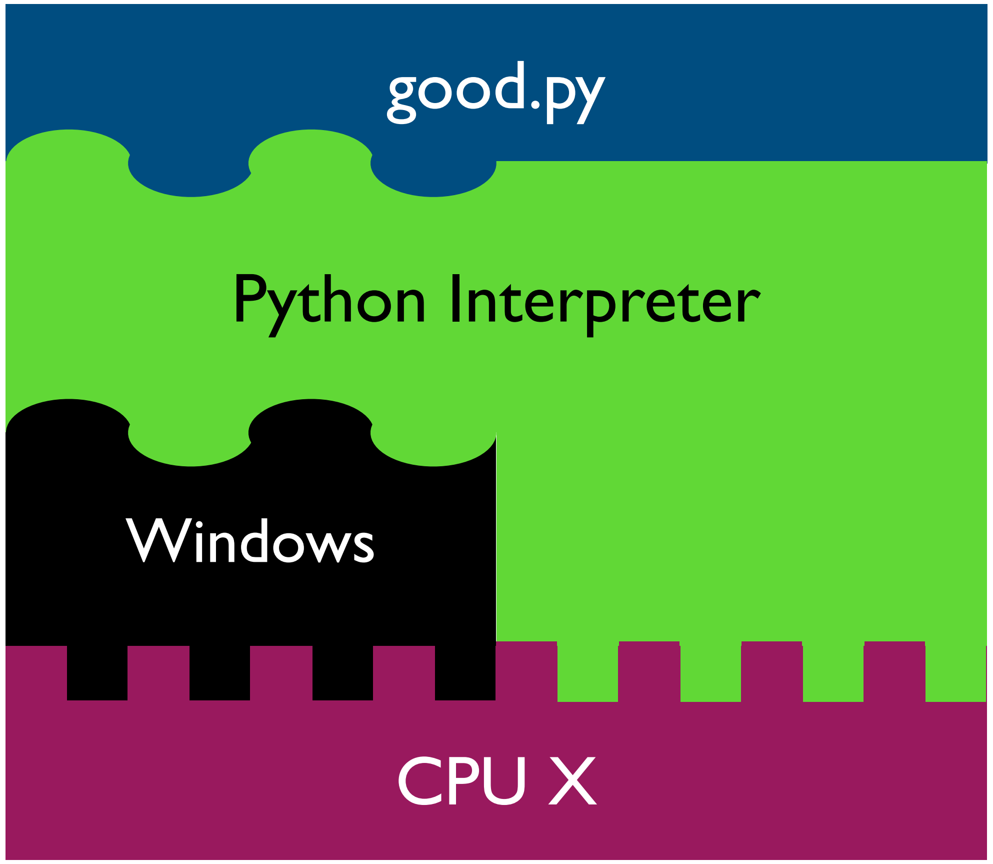
```
f = open("c:\data\file.txt")
...
```

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:

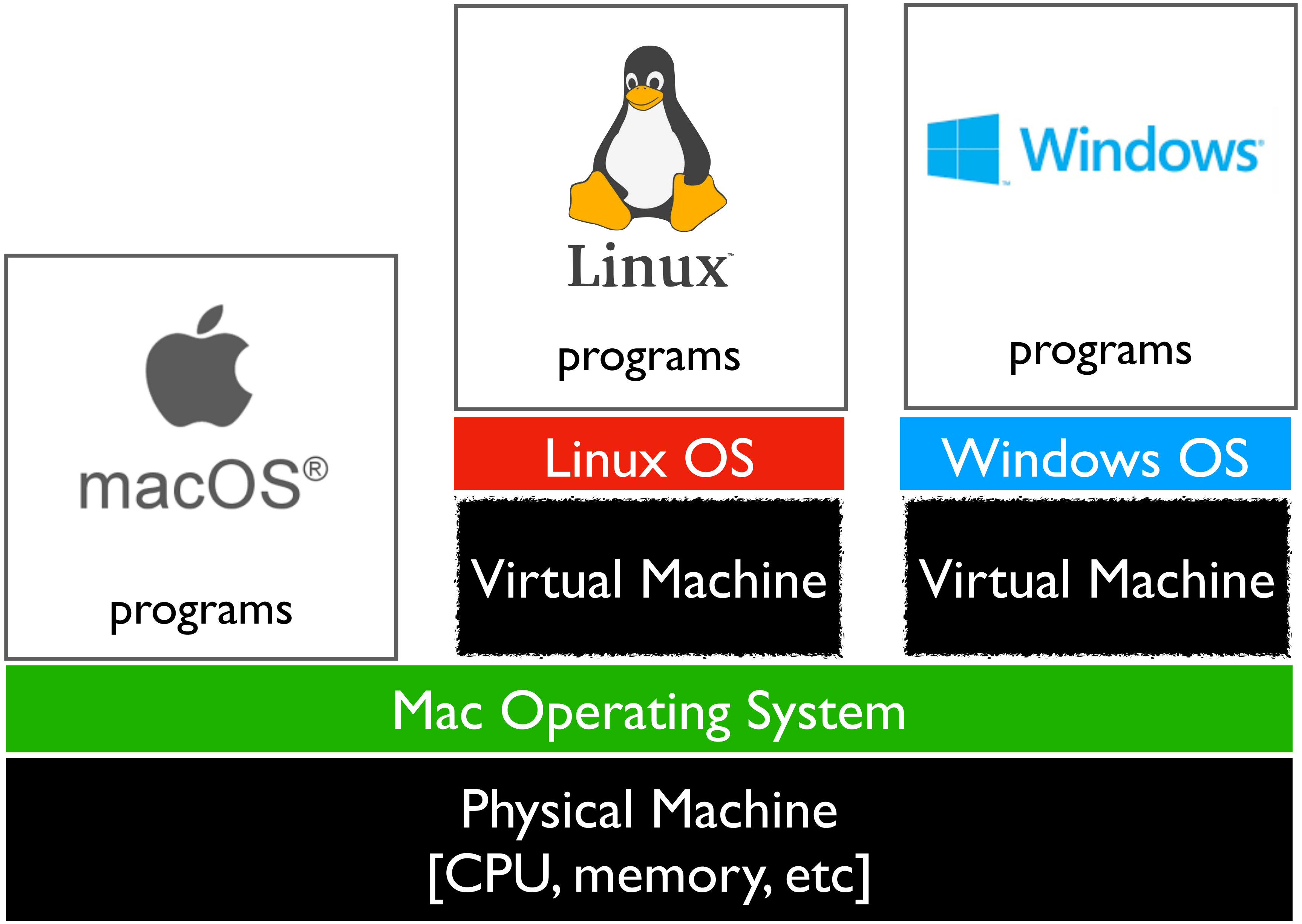
tell anybody reproducing your results to use the same OS!

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)

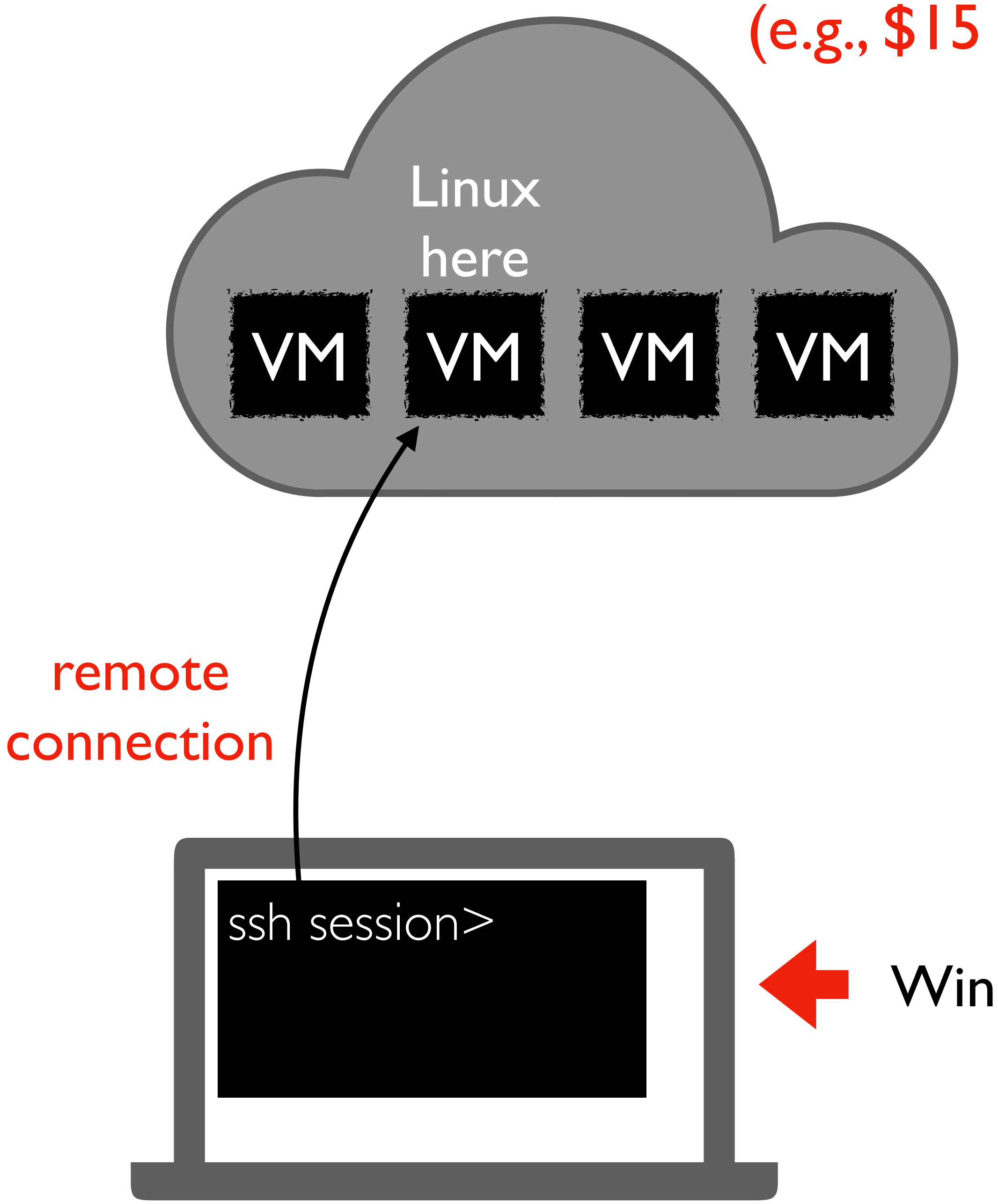


popular virtual machine software

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)



Windows, Mac, whatever

```
ssh user@best-linux.cs.wisc.edu
```

run in PowerShell/
bash to access CS lab

popular cloud providers

amazon
web services™

Microsoft
Azure

Google Cloud Platform

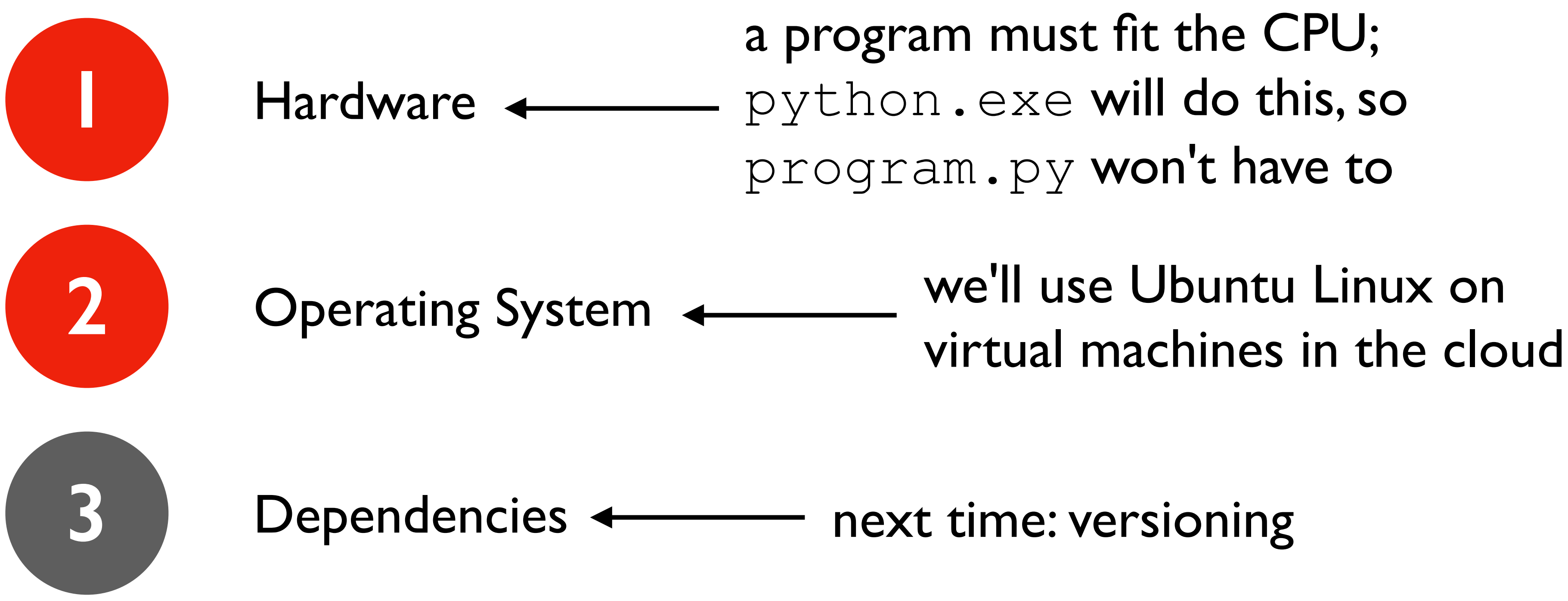
we'll use GCP virtual
machines this semester
[setup in lab]

https://docs.microsoft.com/en-us/windows-server/administration/openssh/openssh_install_firstuse

Lecture Recap: Reproducibility

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

Things to match:



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

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

3 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