[320] Welcome + First Lecture [reproducibility]

Yiyin Shen

Who am I?

Yiyin Shen

- CS PhD student
- Email: yshen82@wisc.edu

Research Interest

- CS Education
- Large Language Models

Teaching Experience

- CS320 TA => Head TA => Instructor
- CS220, CS402 Guest Lectures

Who are You?

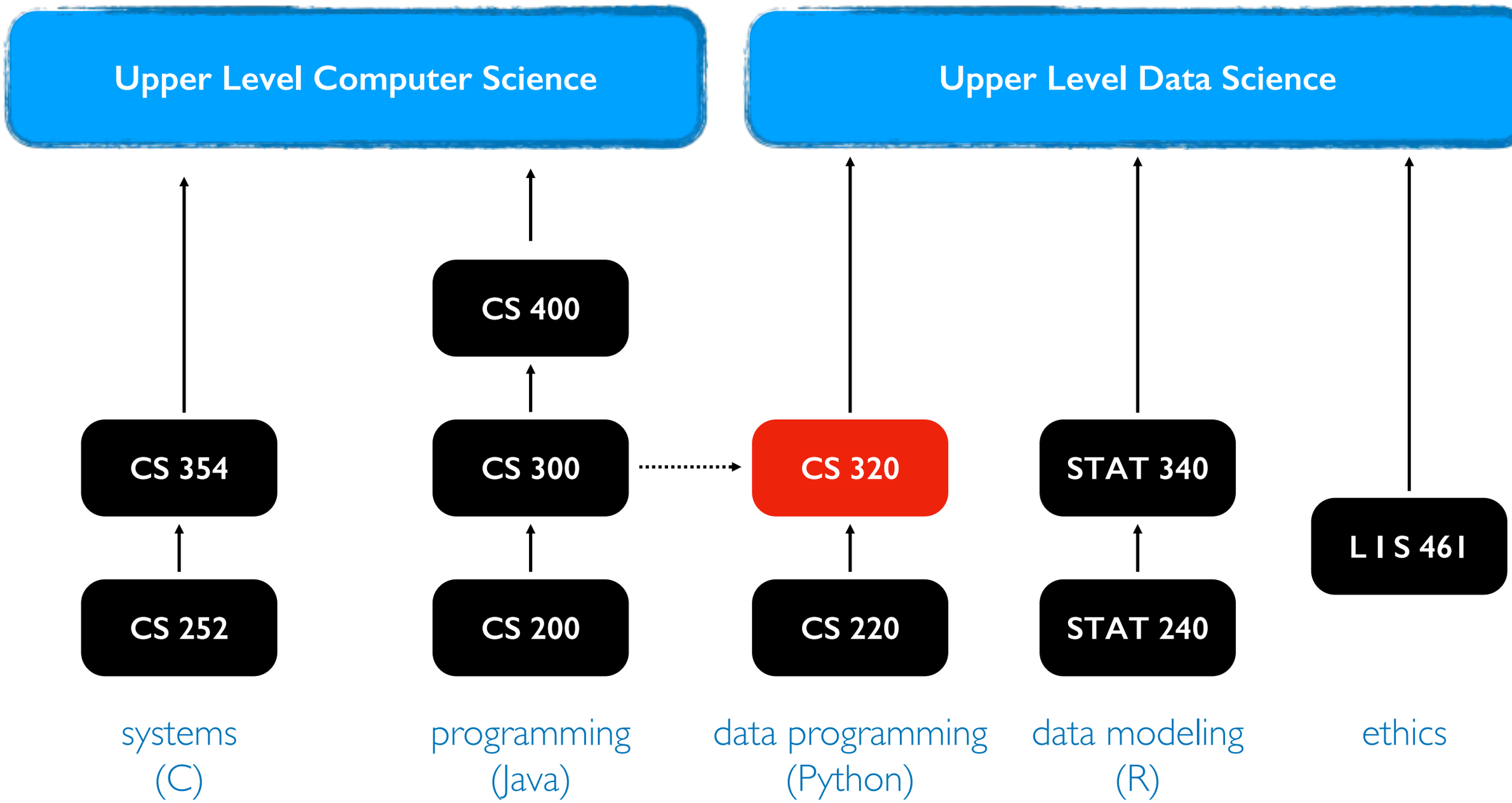
Year in school?
Major?

Please fill out the Student Information form:
(due Wed, June 7th):
https://forms.gle/bSGCkxBW7MPGGeHQ6

Why?

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

Related courses



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

Welcome to Data Science Programming II!

Builds on CS220. https://stat.wisc.edu/undergraduate-data-science-studies/

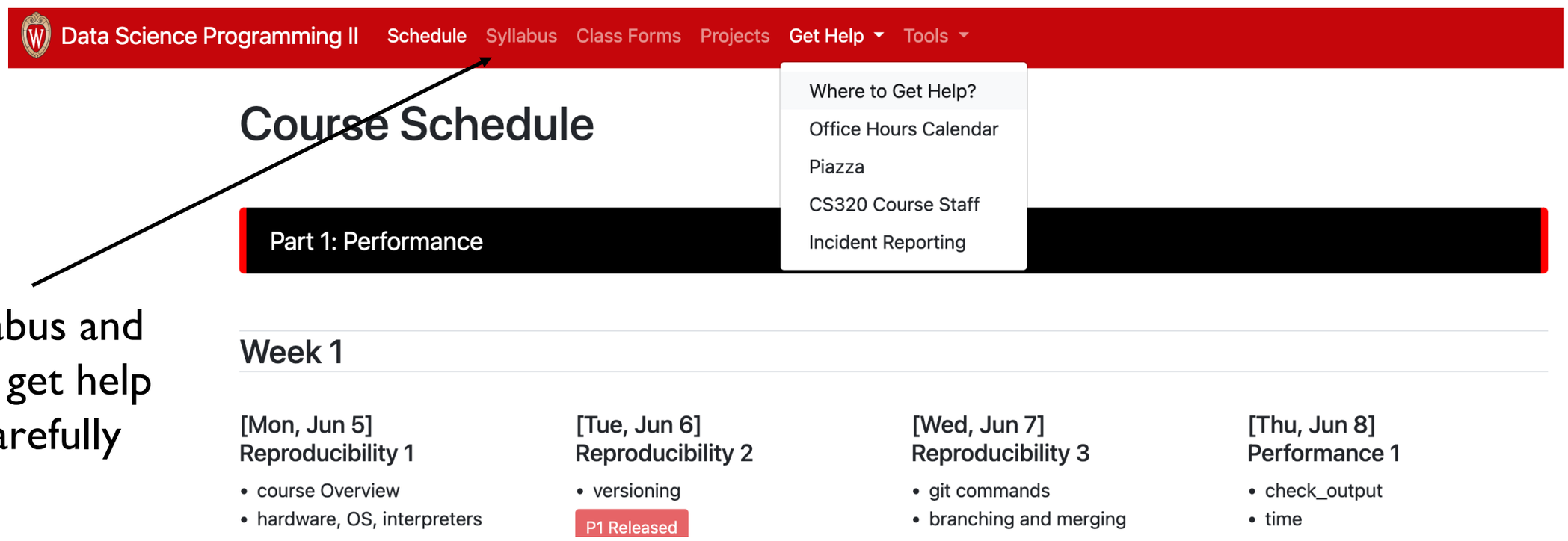
CS220	CS320
getting results	getting **reproducible** results
writing correct code	writing **efficient** code
using objects	designing **new types** of objects
functions: `f(obj)`	**methods**: `obj.f()`
lists + dicts	graphs + trees
analyzing datasets	**collecting** + analyzing datasets
plots	visualizations
tabular analysis	**simple machine learning**

CS220 content (for review): https://cs220.cs.wisc.edu/f22/schedule.html

Course Logistics

Course Website

https://tyler.caraza-harter.com/yiyin/su23/schedule.html



read syllabus and where to get help pages carefully

I'll also use Canvas for:

- Announcements
- Quizzes/exams
- Zoom: lectures, labs, office hours
- Late day summaries
- Grades

Class Organization: People

Groups

- you'll be assigned to a group of 4-7 students
- groups will last the whole semester
- collaboration with group members are allowed (not required) on labs, quizzes, and group part of the projects
- collaboration with non-group members is not allowed

Communication

Drop-in Office Hours:

- Course website – Get Help – Office Hour Calendar
- Queue: https://ohwl.herokuapp.com/

Piazza

- Don't post >5 lines of project-related code (considered cheating)
- Private posts disabled

Forms

- https://tyler.caraza-harter.com/yiyin/su23/surveys.html
- Student Information Survey, Exam Conflicts Forms, Project/Lab Grading Issue Form, Feedback Form, Thank You Form

Email (least preferred)

- me: yshen82@wisc.edu
- TA: Victor vsuciu@wisc.edu
- Course staff: https://canvas.wisc.edu/courses/355770/pages/course-staff

Scheduled Activities

Lectures (MTWR 10:00 – 10:50 AM) (2% overall)

- Recommendation: use your laptop to take notes on the provided template notebook and another screen to follow along the lecture
- Attendance is required. Attendance recorded through Google forms
- 14 drops out of 38 lectures

Labs (TR 11:00 – 11:50 AM) (4% overall)

- Work through lab activities with group mates
- 320 staff will circulate around breakout rooms to answer questions
- Attendance is required. 6 drops out of 18 labs
- 5 attendance points per lab:
- 2 for arriving no later than 5 mins after the lab starts
- 3 for showing sufficient working progress (submit code and/or running results to Canvas at the end of the lab)

Graded Work: Quizzes & Exams

Eight Online Quizzes - 1% each (1 drop, 7% overall)

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

Midterms - 11% each (22% overall)

- cumulative, individual, multi-choice, 50 minutes
- one-page two-sided note sheet
- Friday, June 30th, 7:00PM - 8:30PM
- Friday, July 21st, 7:00PM - 8:30PM

Final - 15%

- cumulative, individual, multi-choice, 2 hours
- two-page two-sided note sheet
- Thursday, August 10th, 10:00AM - 12:30PM

Graded Work: Projects & Surveys

7 Projects - 7% each (49% overall)

- format: python notebook or module
- group part: you can optionally collaborate with group
- individual part: must be done individually (only receive help from 320 staff)
- regular deadlines on course website
- late days: overall 8 late days
- hard deadline: 4 days after the regular deadline – maximum 2 late days; 10% score penalty per day after day 2
- `tester.py` with TA evaluation
- clearing auto-grader on the submission portal (course website) is mandatory

Surveys (1% overall)

Letter Grades

- Your final grade is based on sum of all points earned
- Your grade does not depend on other students' grade
- Scores will NOT be rounded up at the end of the semester
- No major score changes at the end of the semester
- No extra credits

Grade cut-offs

- 93% - 100%: A
- 88% - 92.99%: AB
- 80% - 87.99%: B
- 75% - 79.99%: BC
- 70% - 74.99%: C
- 60% - 69.99%: D

Time Commitment & Academic Conduct

Project commitment

- 10-12 hours per project is typical (2-4 hours can be done in labs)
- 20% of students sometimes spend 20+ hours on some projects
- recommendation: start early and be proactive

Typical Weekly Expectations

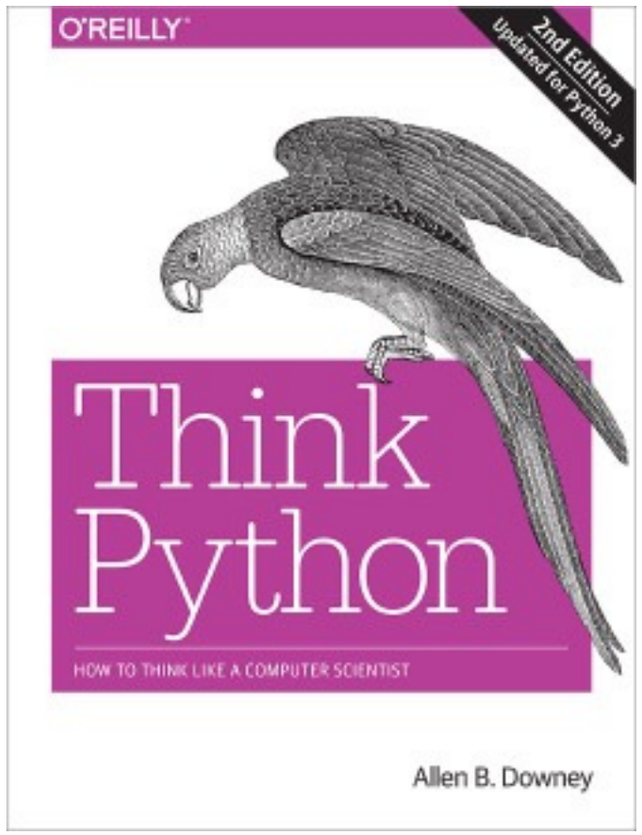
- 6 hours - lecture/lab
- 8 hours - project coding
- 2 hours - reading/quizzes/etc

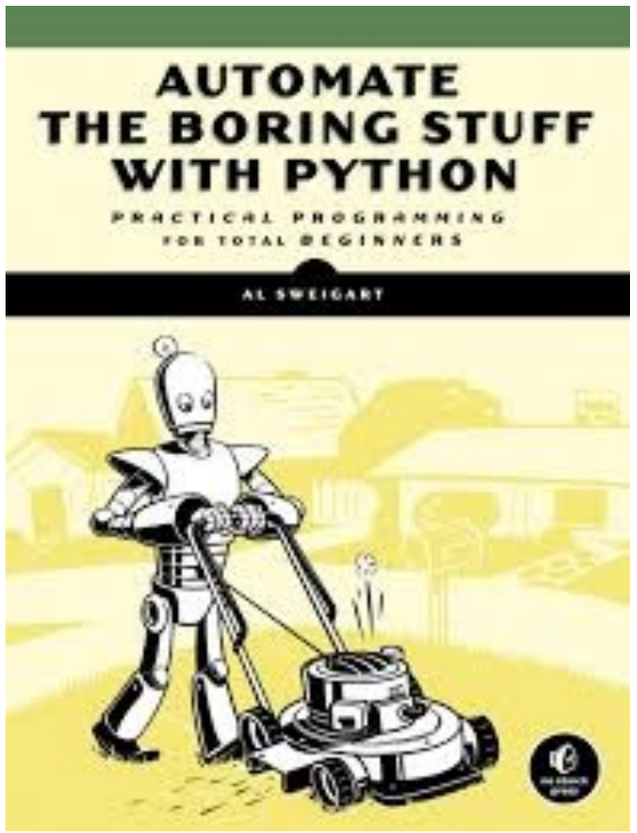
Please talk to me if you're feeling overwhelmed with 320 or your semester in general.

Academic Conduct

- Read syllabus to make sure you know what is and isn't acceptable.
- We will run plagiarism detector on project submissions.

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





I'll post links to other online articles and notes

Lectures don't assume any reading prior to class

Tips for 320 Success

1. Just show up
   Get 100% on attendance, don't miss quizzes, submit group work

2. Use office hours

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

Today's Lecture:

Reproducibility

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

/ˌrēprəˌd(y)o͞osəˈbilədē/

noun

noun: **reproducibility**

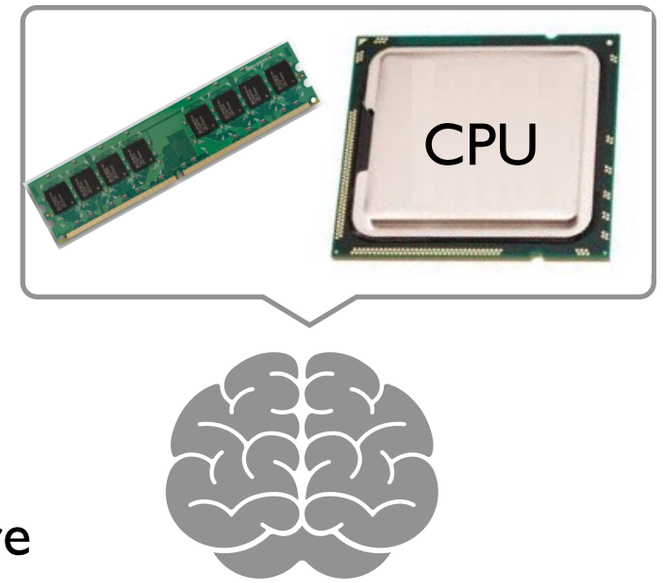
the ability to be reproduced or copied.
"the reproducibility of reconstructive surgery techniques"

- 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:

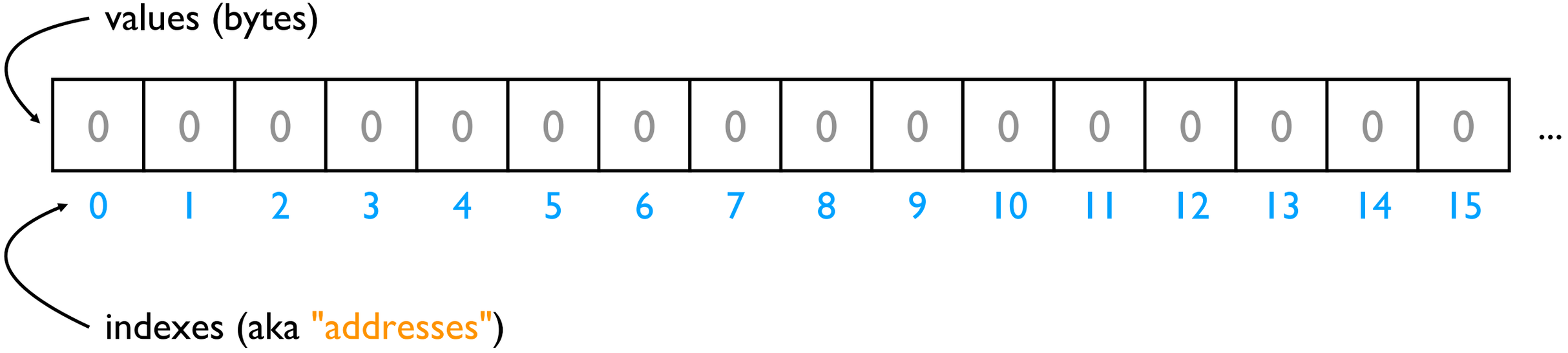


1. Hardware
2. Operating System ← next lecture
3. Dependencies ← next lecture

Hardware: Mental Model of Process Memory

Imagine...

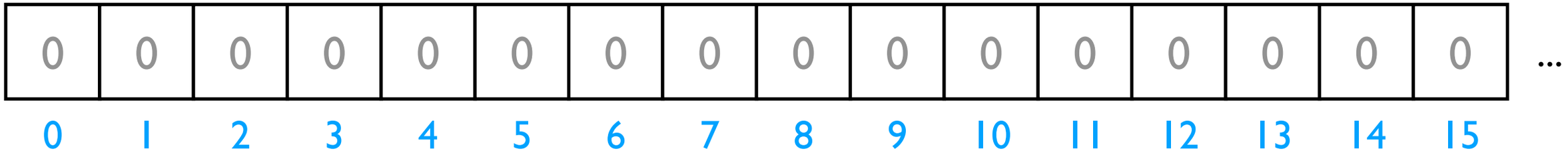
- one huge list, **per each** ~~running program~~ **process**, called **"address space"**
- every entry in the list is an integer between 0 and 255 (aka a "byte")



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

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

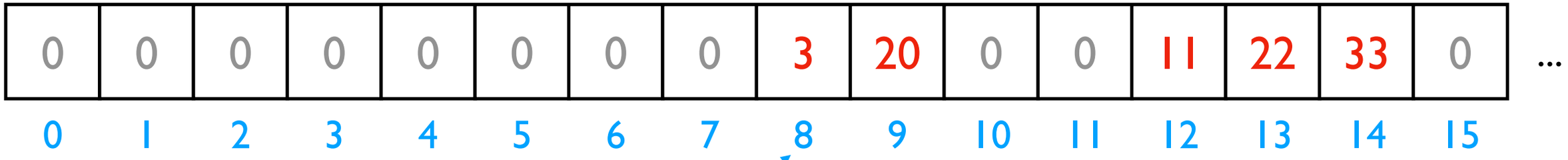
data



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

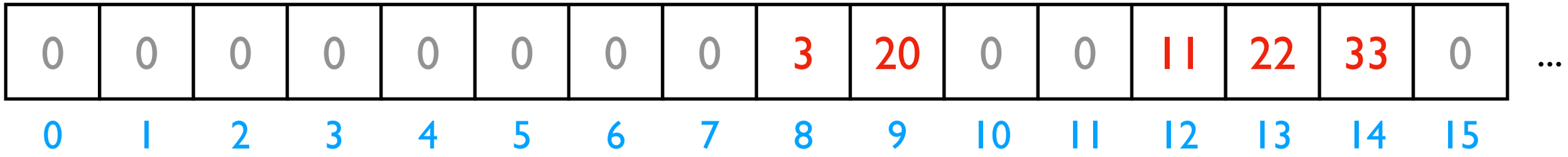


the [3,20] list starts at ~~index~~ address 8 in the giant list

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

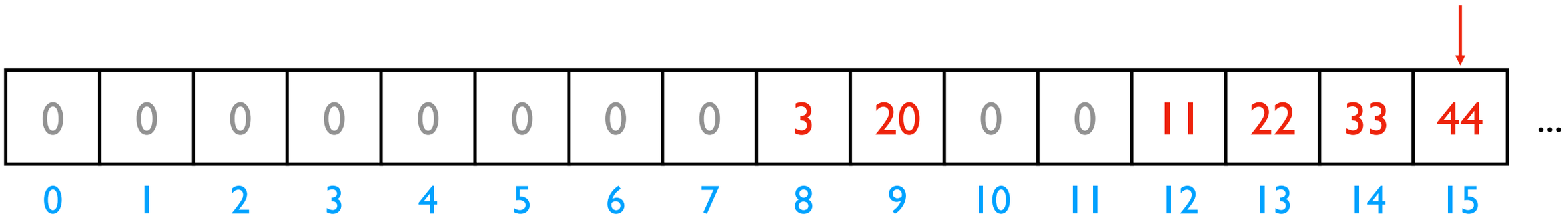


implications for performance...

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

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

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

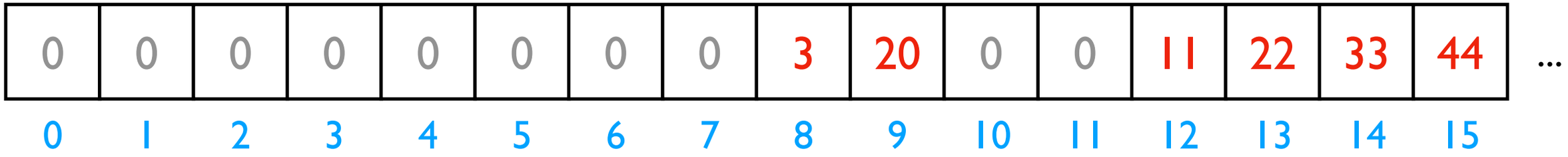


implications for performance...

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

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

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



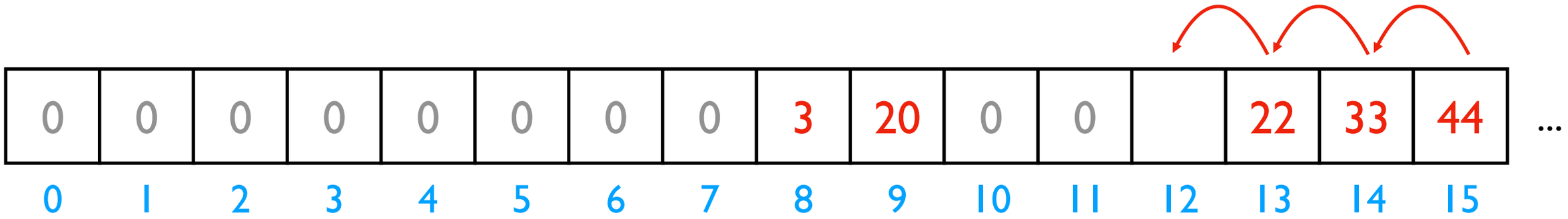
implications for performance...

```
# 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



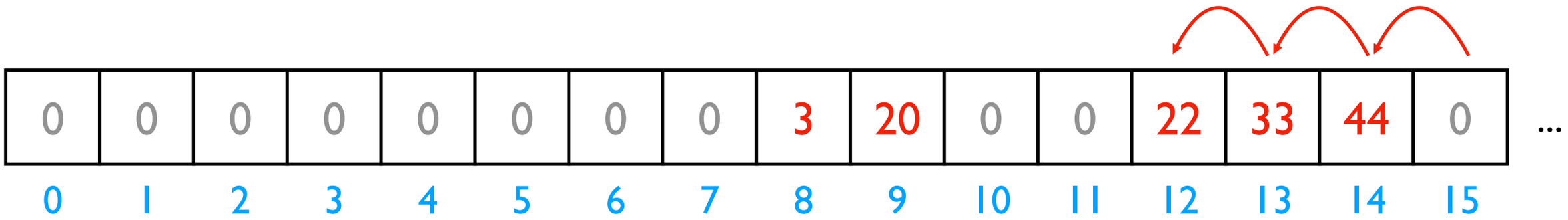
implications for performance...

```
# 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



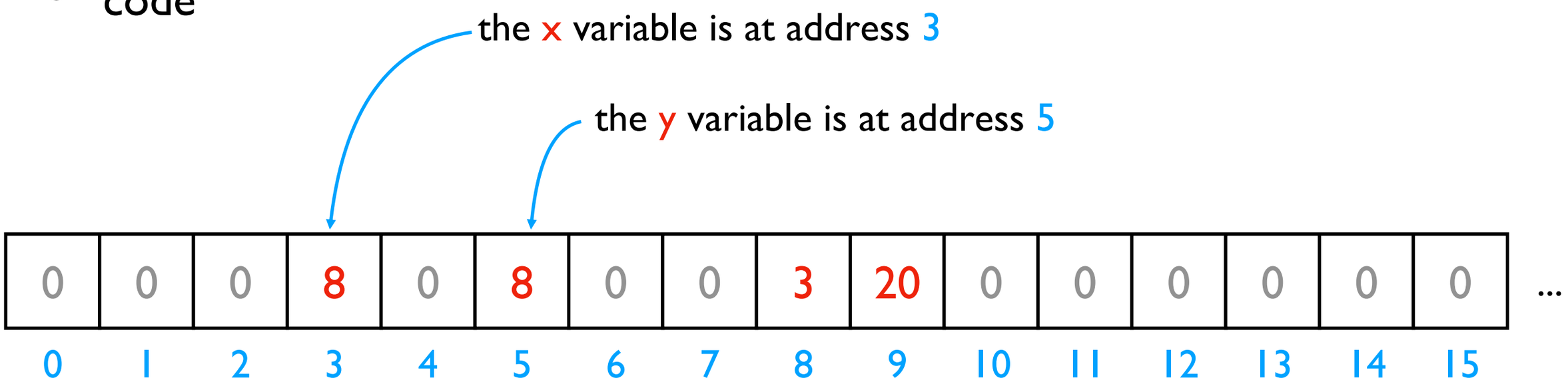
We'll think more rigorously about performance in CS 320 (big-O notation)

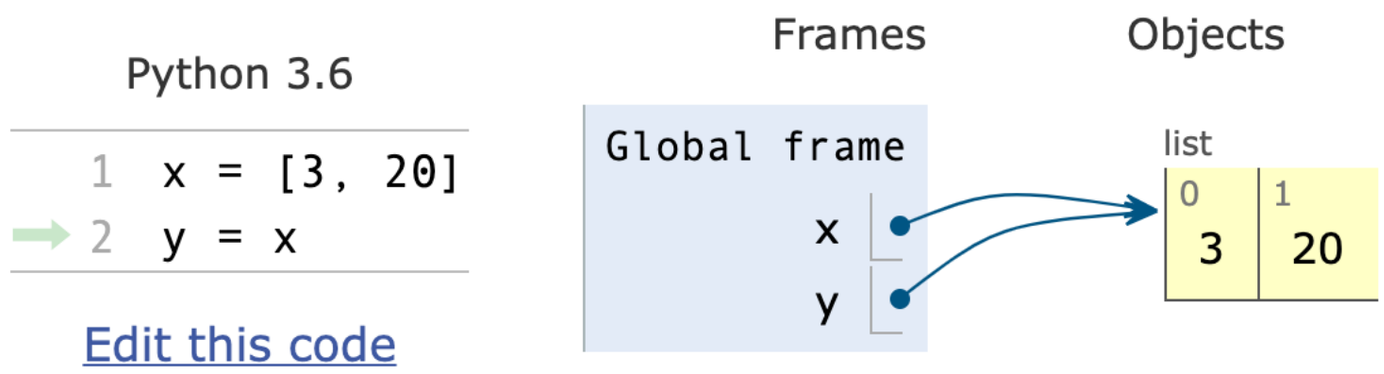
```
# fast
L2.append(44)

# slow
L2.pop(0)
```

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

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



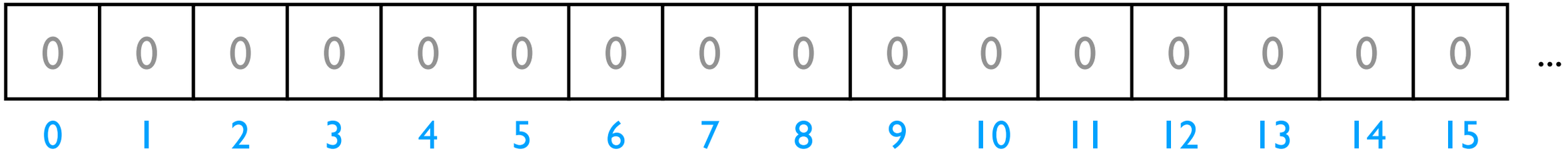


PythonTutor's visualization

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

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

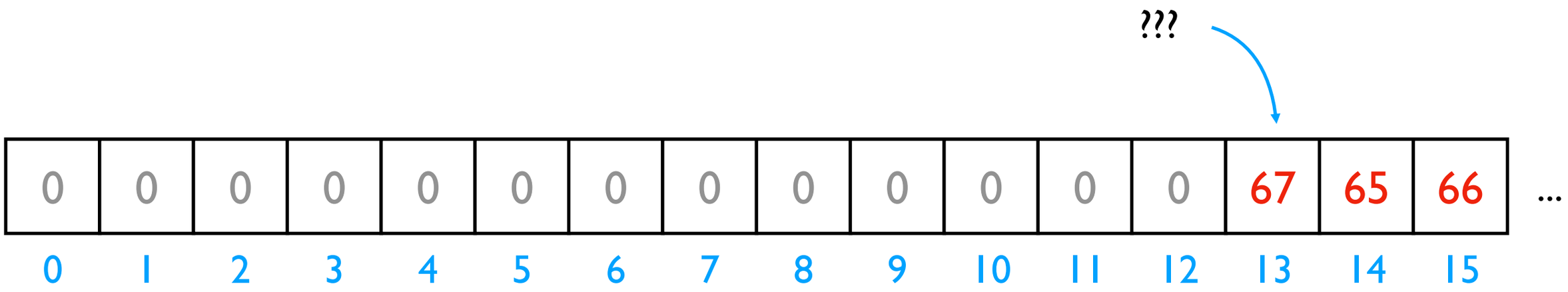
discuss: how?



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



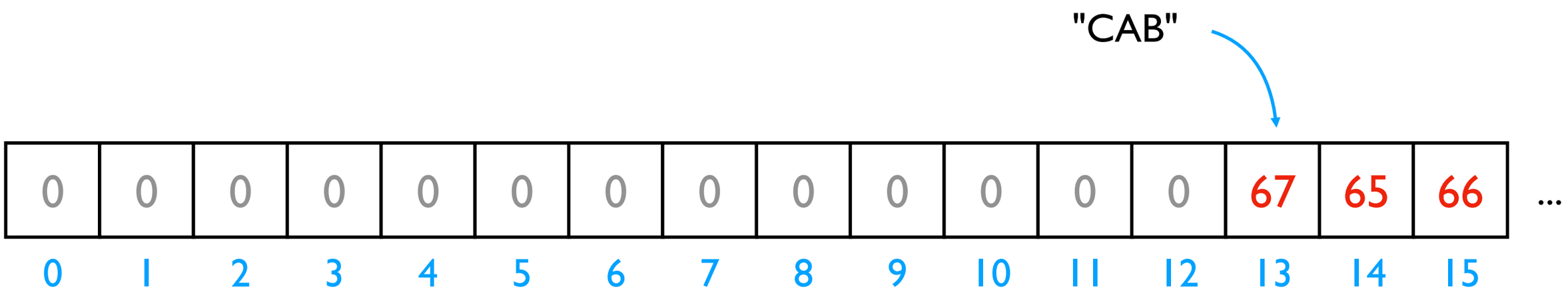
encoding:

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



encoding:

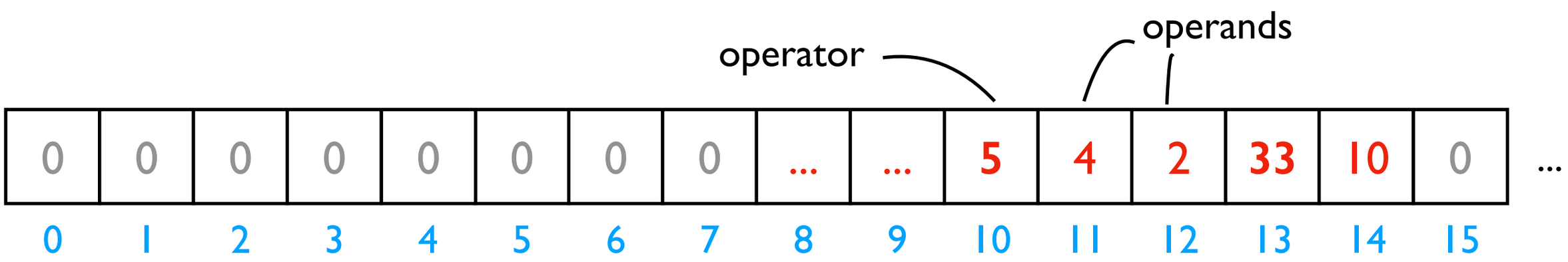
code	letter
65	A
66	B
67	C
68	D
...	...

```
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

```
i = 0
while ????:
    i += 2
    # what line next?
```



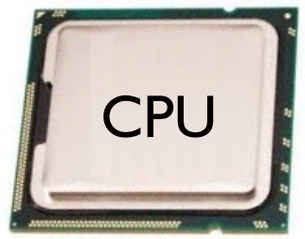
Instruction Set

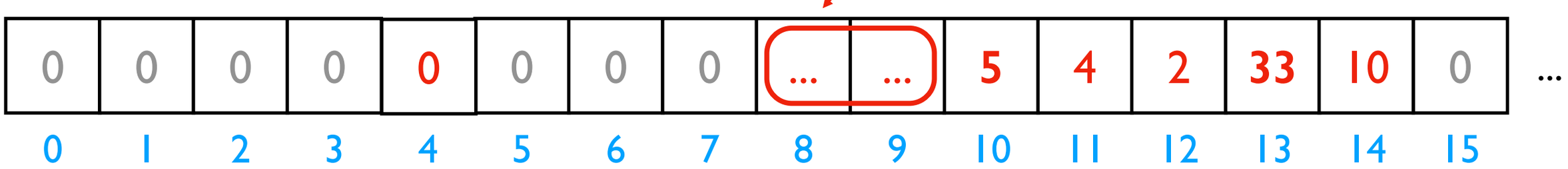
code	operation
5	ADD
8	SUB
33	JUMP
...	...

Hardware: Mental Model of CPU

CPUs interact with memory:

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





Write code in Python 3.6

(drag lower right corner to resize code editor)

```
1
2
3
```

line that just executed

next line to execute

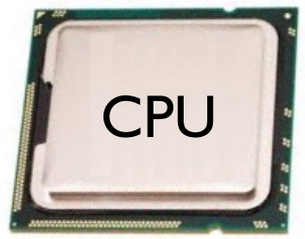
Instruction Set

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

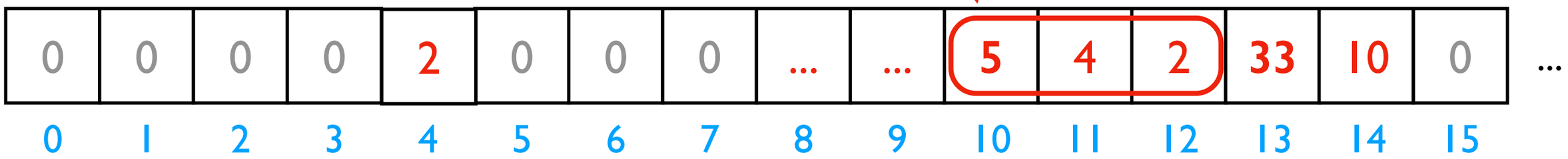
Hardware: Mental Model of CPU

CPUs interact with memory:

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



add 2 to variable



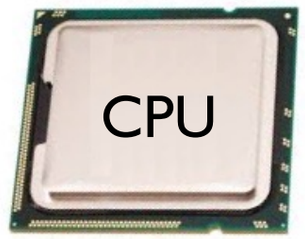
Instruction Set

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

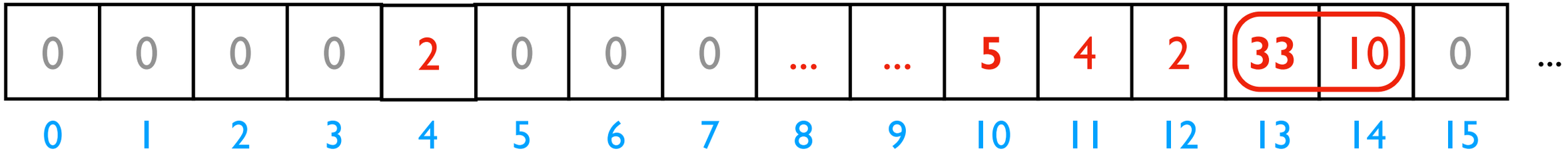
Hardware: Mental Model of CPU

CPUs interact with memory:

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



go back to top of loop



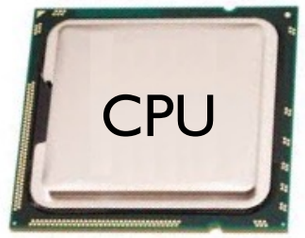
Instruction Set

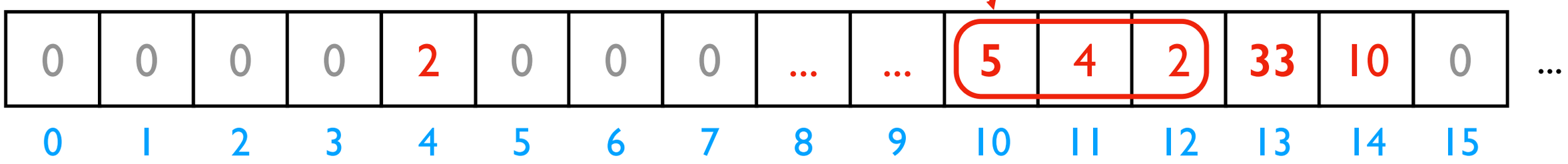
code	operation
5	ADD
8	SUB
33	JUMP
...	...

Hardware: Mental Model of CPU

CPUs interact with memory:

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



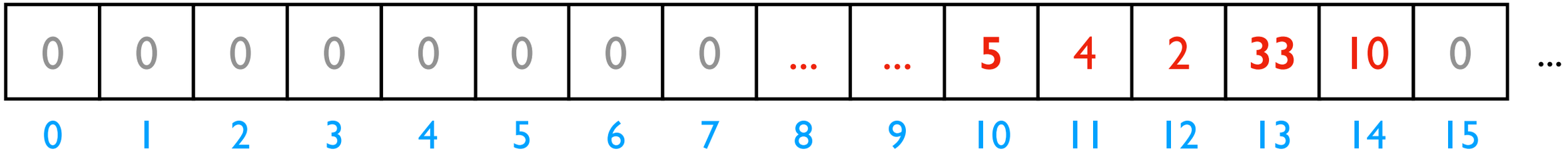


Instruction Set

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?



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**	**4**	**2**	**33**	**10**	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

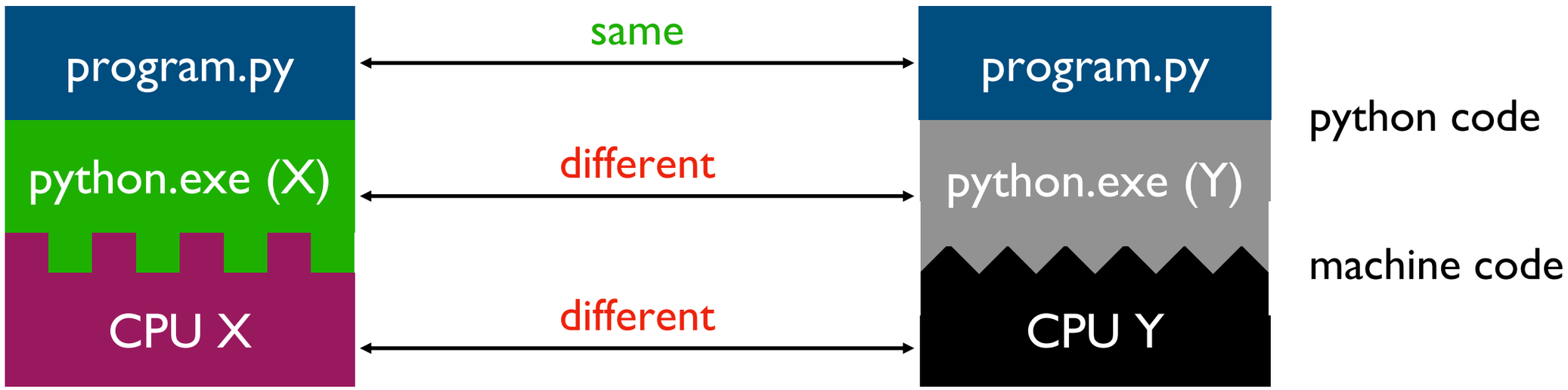
code	operation
5	SUB
8	ADD
33	undefined
...	...

A Program and CPU need to "fit"

A Program and CPU need to "fit"

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

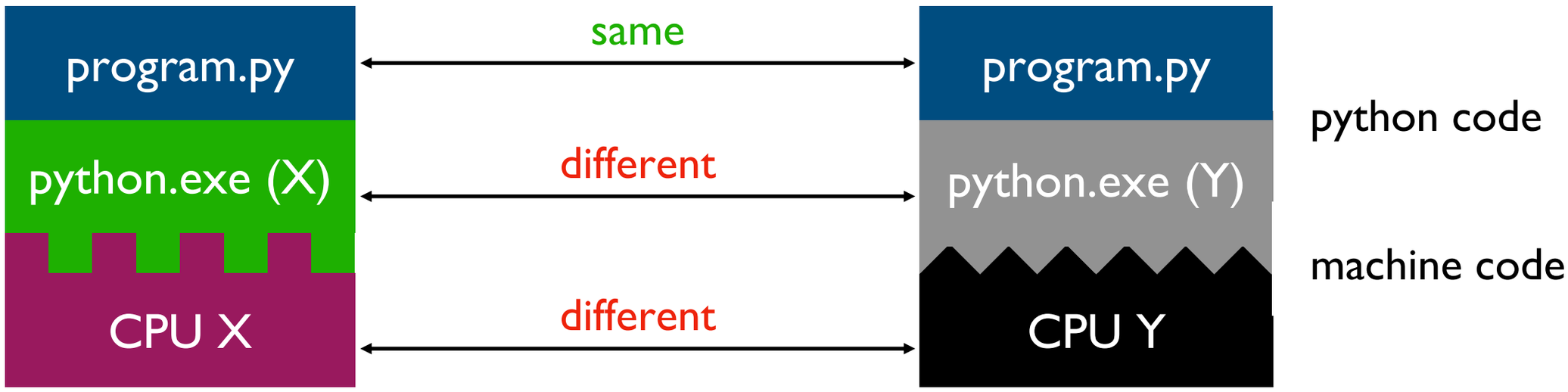
Interpreters



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



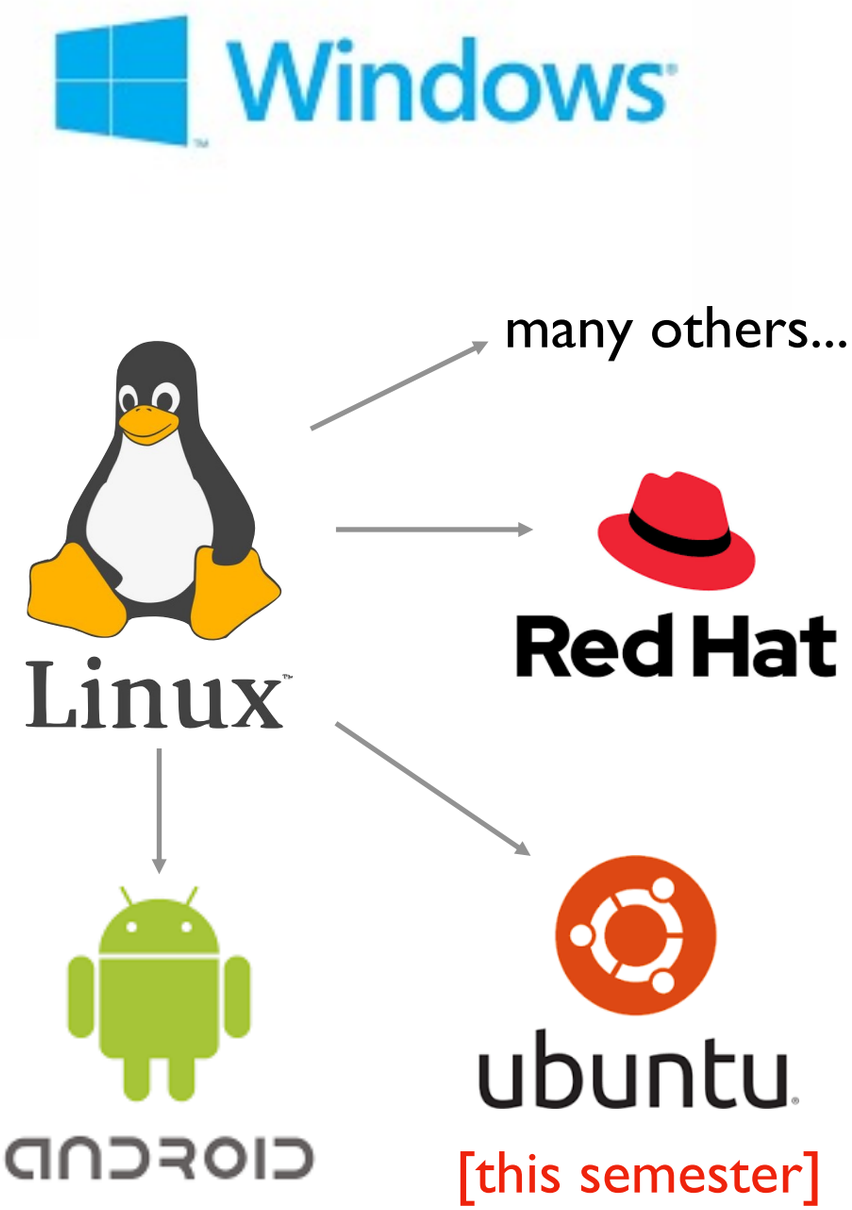
Interpreters (such as python.exe) make it easier to run the same code on different machines

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:

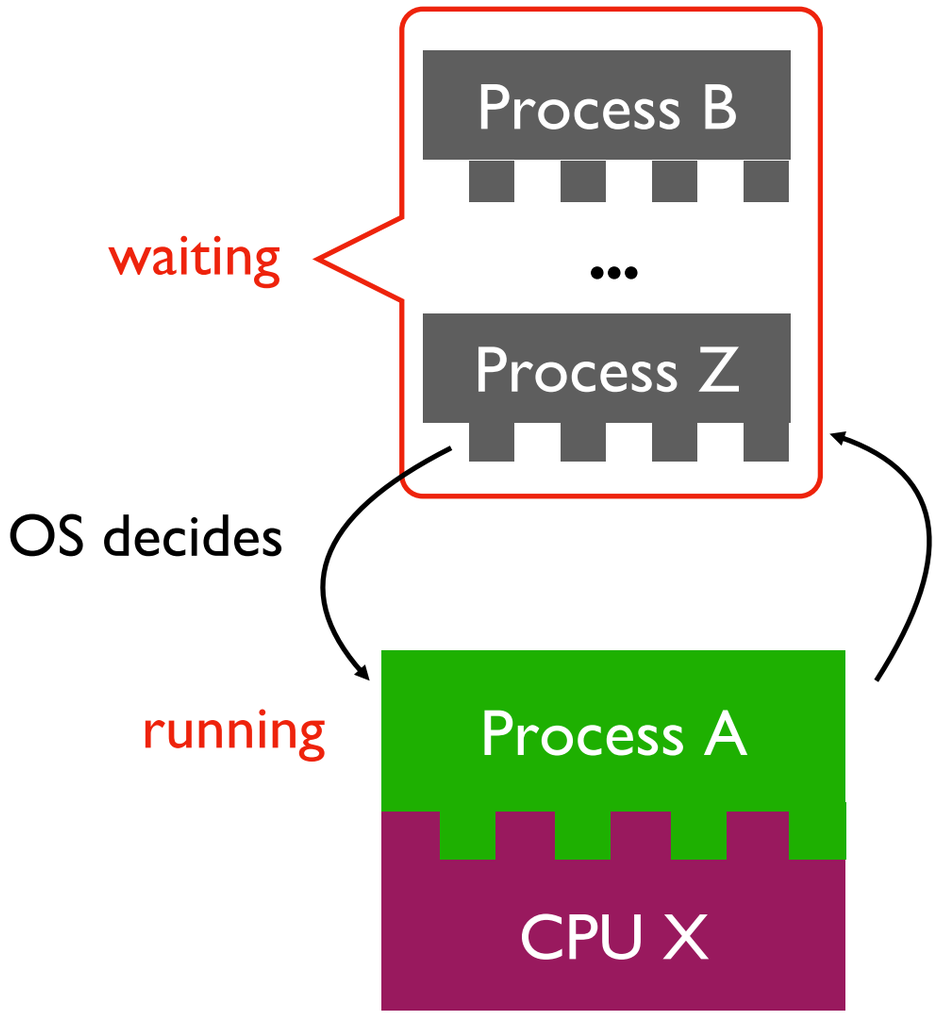
1. Hardware
2. Operating System
3. Dependencies



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()
```

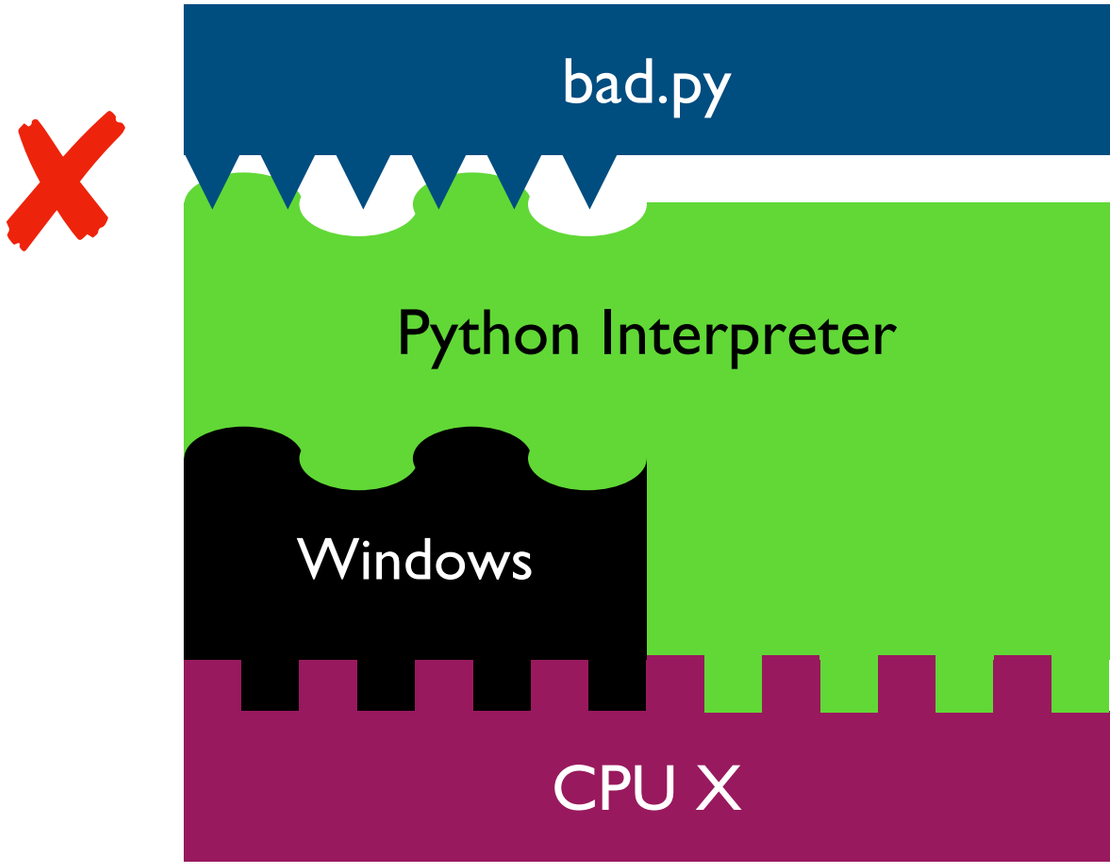
convenient

Operating System

inconvenient

ignorant of files/directories

Harder to reproduce on different OS...

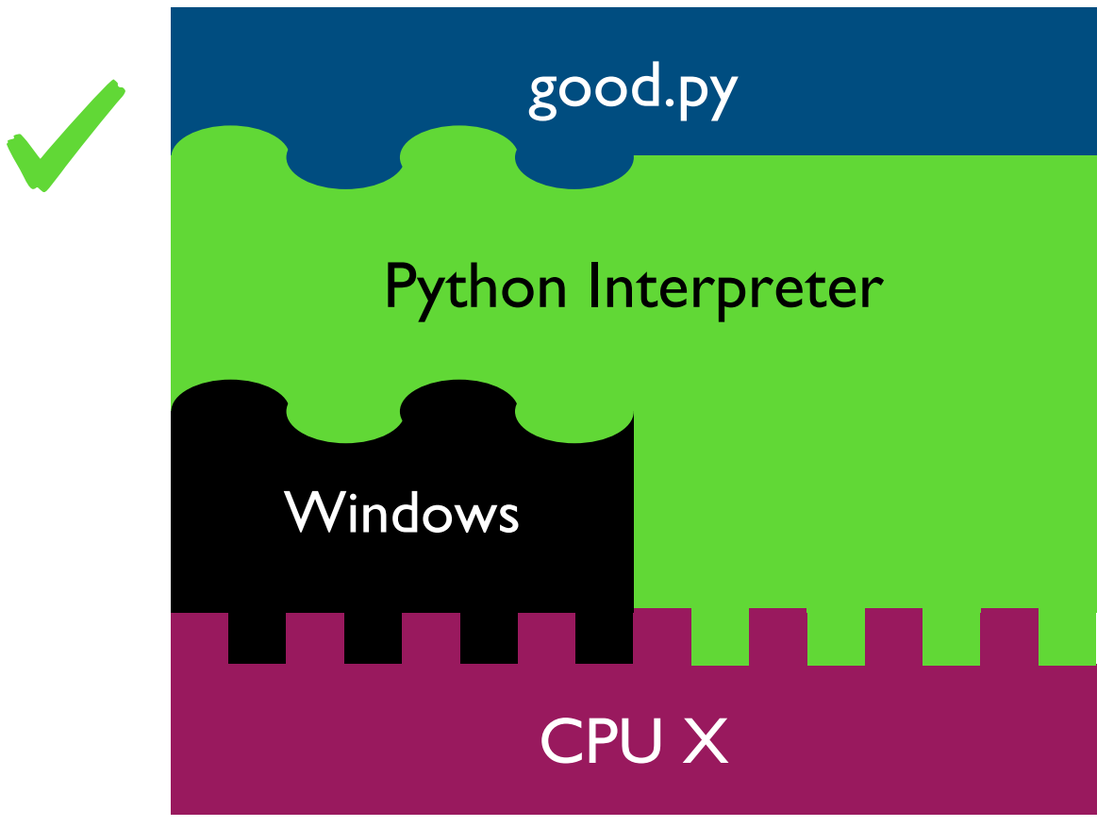



```
f = open("/data/file.txt")
...
```

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...

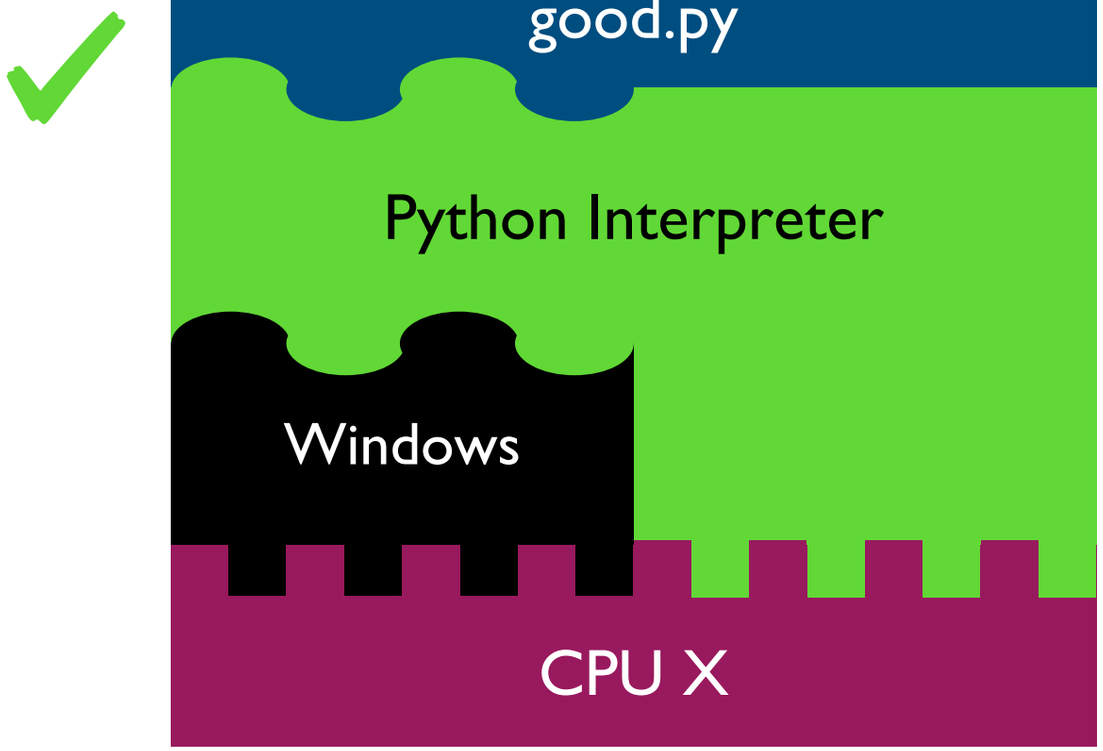



```
f = open("c:\data\file.txt")
...
```

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?

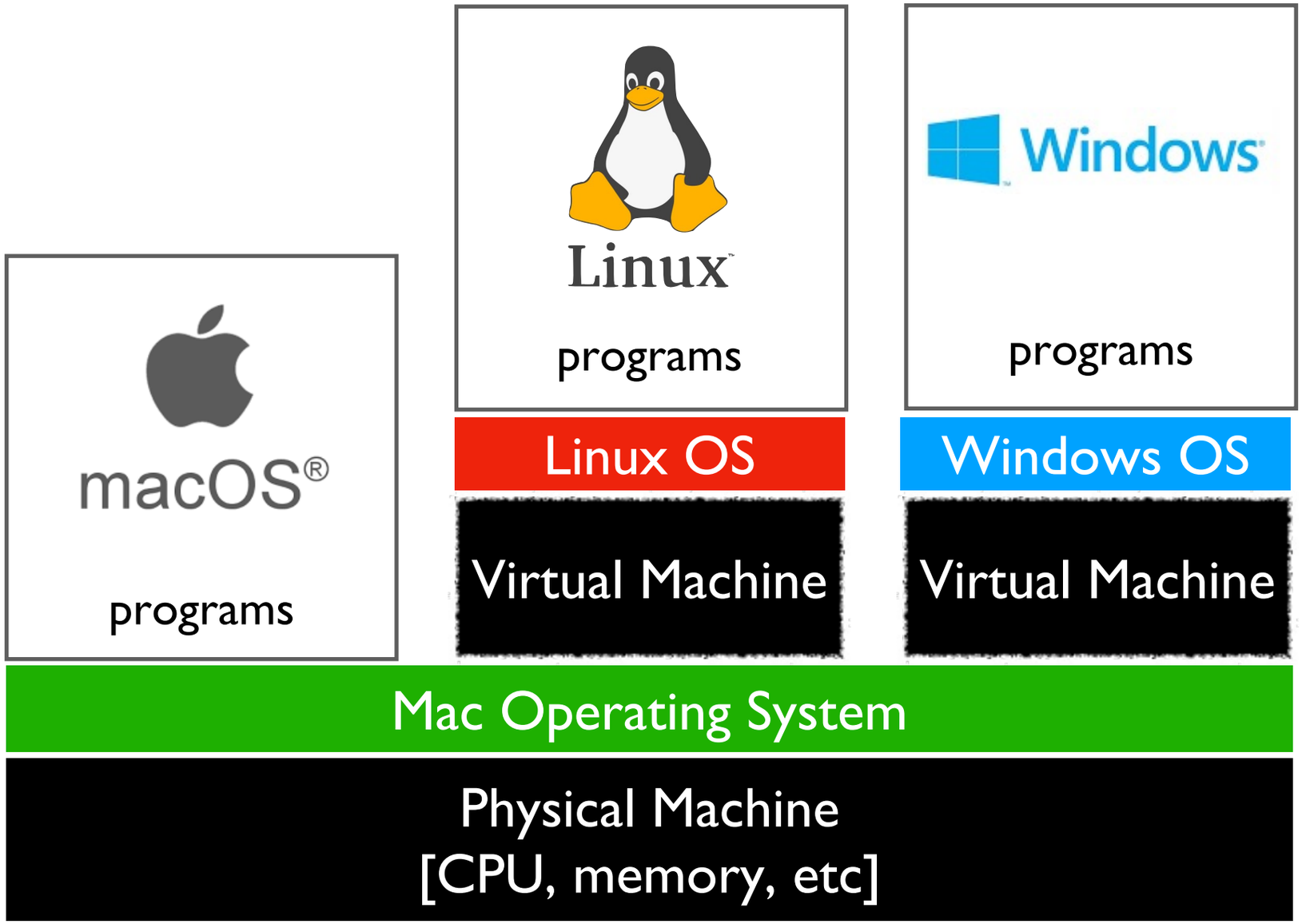
The Python interpreter mostly lets you
[Python Programmer] ignore the CPU you run on.

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

VMs (Virtual Machines)

popular virtual machine software

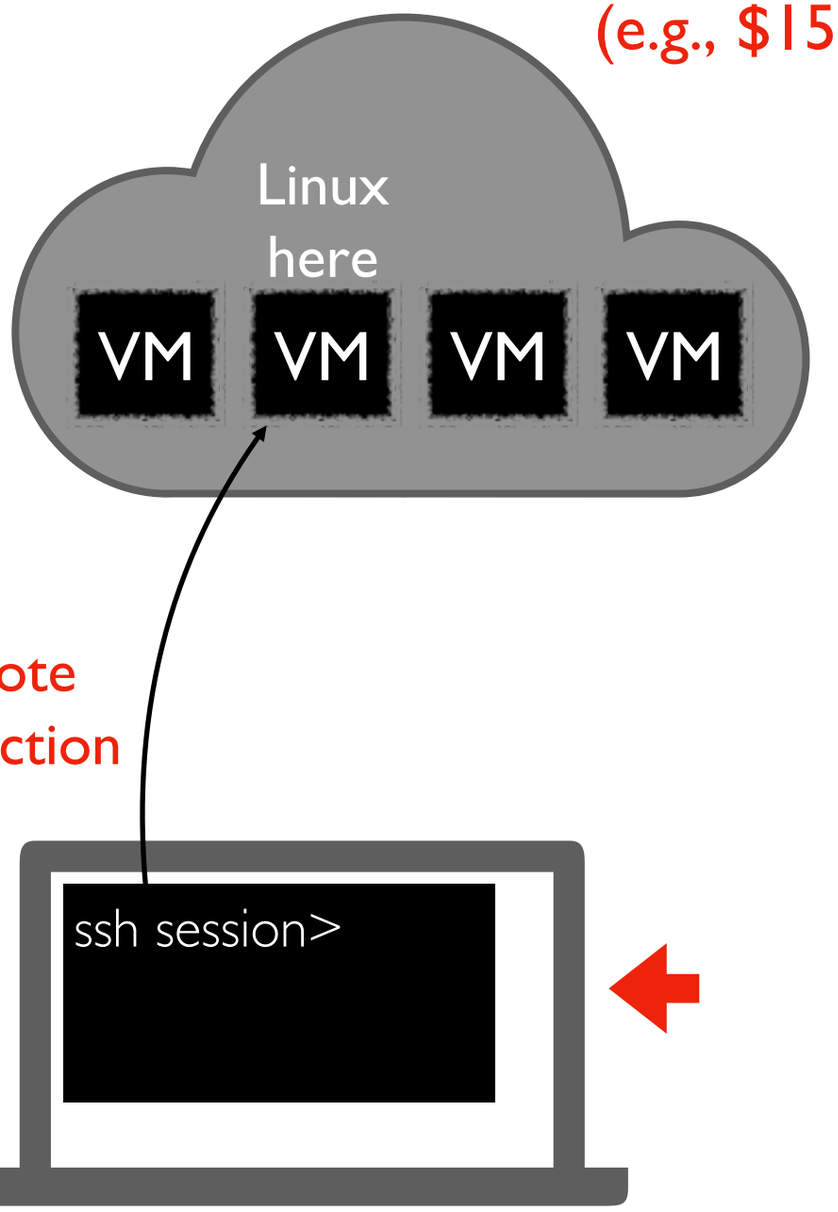
Microsoft Hyper-V



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)



Windows, Mac,
whatever

`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]

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