[320] Welcome + First Lecture

[reproducibility]

Meenakshi Syamkumar

Who am I?

Meenakshi (Meena) Syamkumar
- Email: ms@cs.wisc.edu
- Please call me "Meena"

Industry and Teaching experience
- Citrix, Cisco, and Microsoft
- CS300, CS220, CS367, guest lectures in CS640, CS740

Research
- Network measurements
- CS education

Who are You?

Canvas > Top Hat

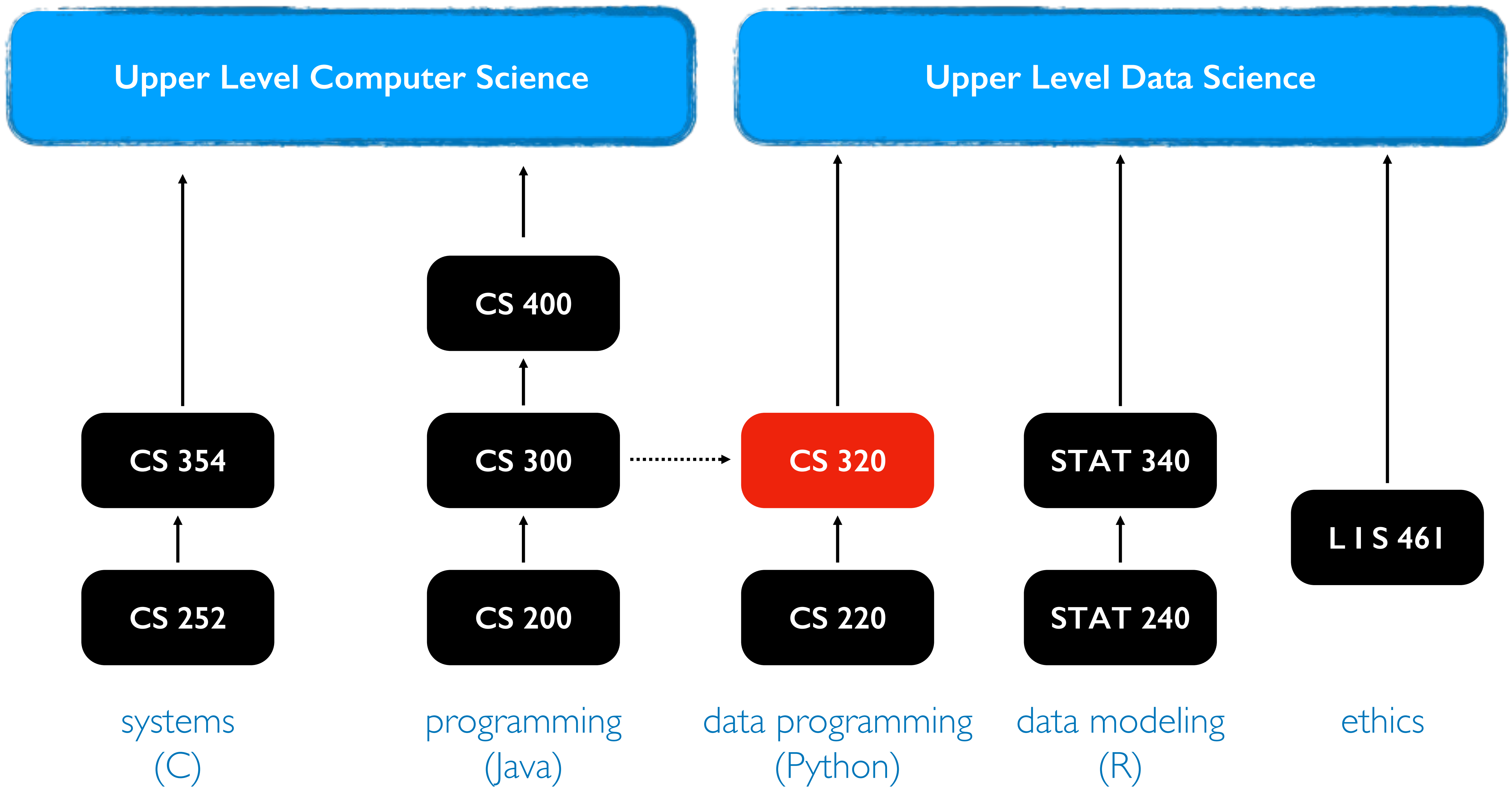
- Sign in with your wisc.edu school account

Please fill this form (due next Monday, Jan 30th):
https://forms.gle/KqvLHGrCvuP9Z7wF9

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/

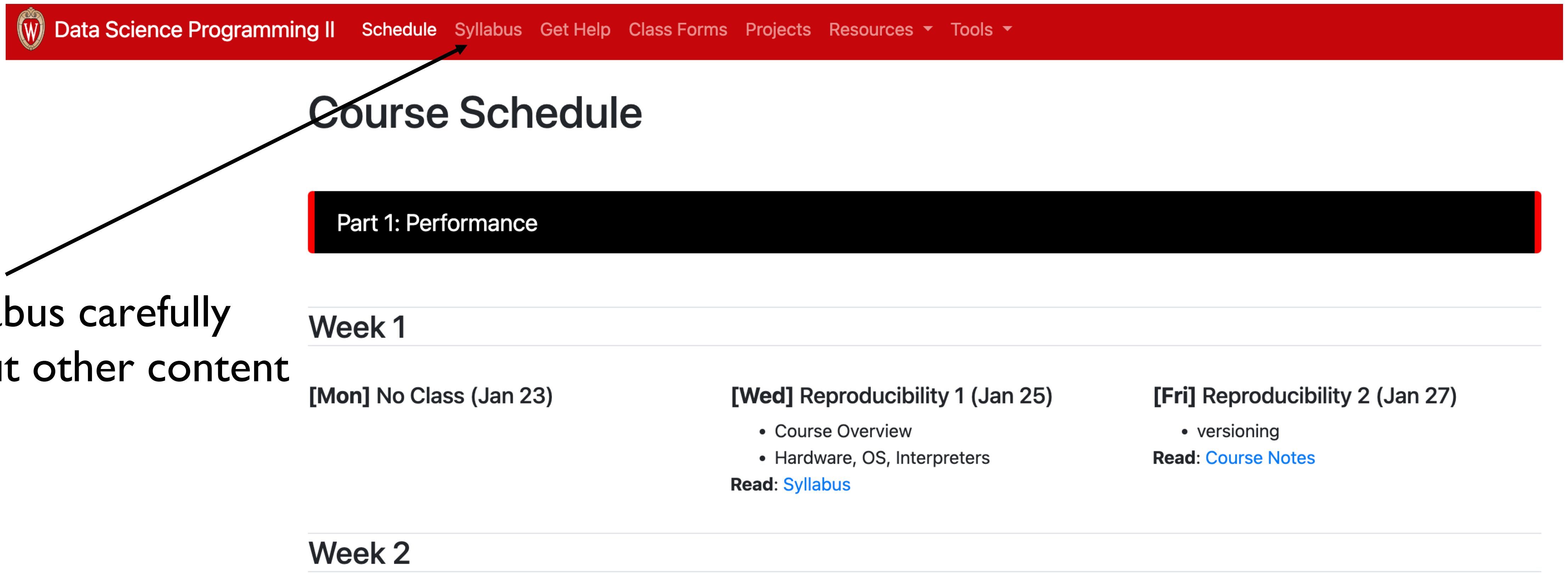
CS220	CS320
getting results	getting **reproducible** results
writing correct code	writing **efficient** code
using objects	designing **new types** of objects
functions: `f(obj)`	**methods**: `obj.f()`
lists + dicts	graphs + trees
analyzing datasets	**collecting** + analyzing datasets
plots	animated visualizations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

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

Course Logistics

Course Website

It's here: https://www.msyamkumar.com/cs320/s23/schedule.html



Data Science Programming II Schedule Syllabus Get Help Class Forms Projects Resources Tools

Course Schedule

Part 1: Performance

Week 1

[Mon] No Class (Jan 23)

[Wed] Reproducibility 1 (Jan 25)
- Course Overview
- Hardware, OS, Interpreters

Read: Syllabus

[Fri] Reproducibility 2 (Jan 27)
- versioning

Read: Course Notes

Week 2

read syllabus carefully
and checkout other content

I'll also use Canvas for four things:
- general announcements
- quizzes
- online office hours
- grade summaries & exam location / answers (individual messages)

Scheduled Activities

Lectures

- 3 times weekly; recommendation: bring your laptop
- Required for participation credit! Attendance recorded via TopHat quizzes (20% score drops)
- 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 or Tuesdays, bring a laptop
- Work through lab exercises with group mates
- 320 staff will walk around to answer questions
- Required for participation credit! Attendance recorded using name cards (3 score drops)
- 5 points per lab
  - 1 point for arriving on time, 3 points for working on the lab, 1 point for staying until end of the lab

Class organization: People

Teams

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

Staff

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

We all provide office hours.
Office hours are drop-in (no need to reserve).

Communication

Piazza

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

Forms

- https://www.msyamkumar.com/cs320/s23/surveys.html
- Student Information Survey. **Exam conflicts.** Grading Issues. Feedback form. Thank you form!

Email (least preferred)

- me: ms@cs.wisc.edu
- Head TA: Yiyin yshen82@wisc.edu
- Course staff: https://canvas.wisc.edu/courses/343506/pages/cs320-staff

Graded Work: Exams / Quizzes

Ten Online Quizzes - 1% each (10% overall)

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

Midterms - 13% each (26% overall)

- cumulative, individual, multi-choice, 40 minutes
- one-page two-sided note sheet
- **in class: March 3rd, April 7th**

Final - 15%

- cumulative, individual, multi-choice, 2 hours
- one-page two-sided note sheet
- **May 12th 10:05AM - 12:05PM**

Graded Work: Projects

7 Projects - 6% each (42% overall)

- **format**: notebook, module, or program
- part 1: you can optionally collaborate with team
- part 2: must be individually (only help from 320 staff)
- regular deadlines on course website
- late days: overall 12 late days
- hard deadline: 7 days after the regular deadline – maximum 3 late days; 5% score penalty per day after day 3
- still a `tester.py`, but more depends on TA evaluation (more plots)
- clearing auto-grader on the submission portal (course website) is mandatory
- ask for specific feedback (constructive)

Graded Work: Attendance + Surveys

Lab attendance - 4% overall

- 3 score drops:
- use these wisely – potential sickness, planned absences
- no other exceptions

Lecture attendance - 2% overall

- 20% score drops

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 off at the end of the semester
- No major score changes at the end of the the semester
- No extra credit

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
- 20% of students sometimes spend 20+ hours on some projects
- recommendation: start early and be proactive

Typical Weekly Expectations

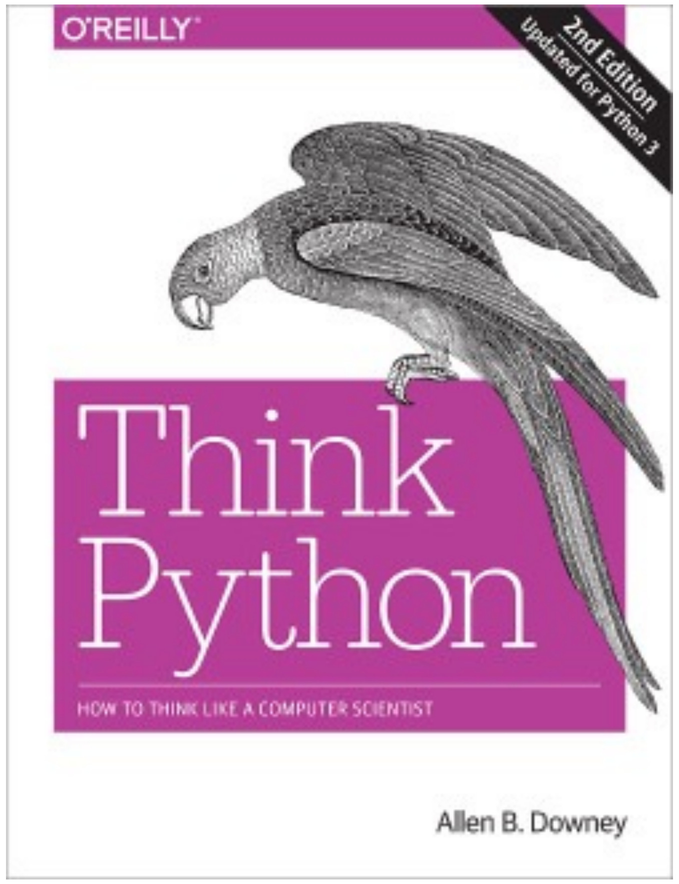
- 4 hours - lecture/lab
- 6 hours - project coding
- 2 hours - reading/quizzes/etc

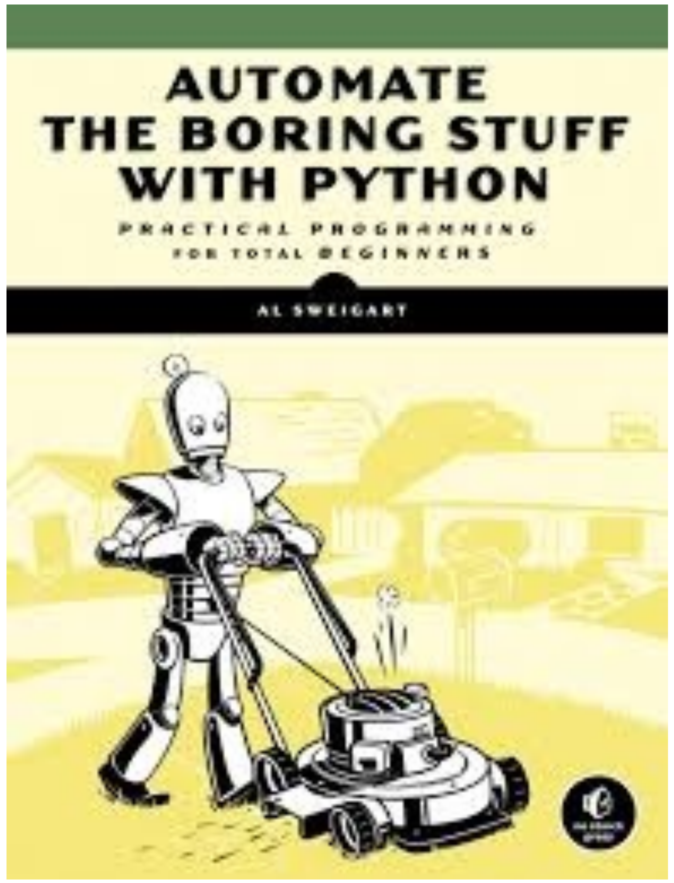
Please talk to me if you're feeling overwhelmed with 320 or your semester in general.

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...




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

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 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

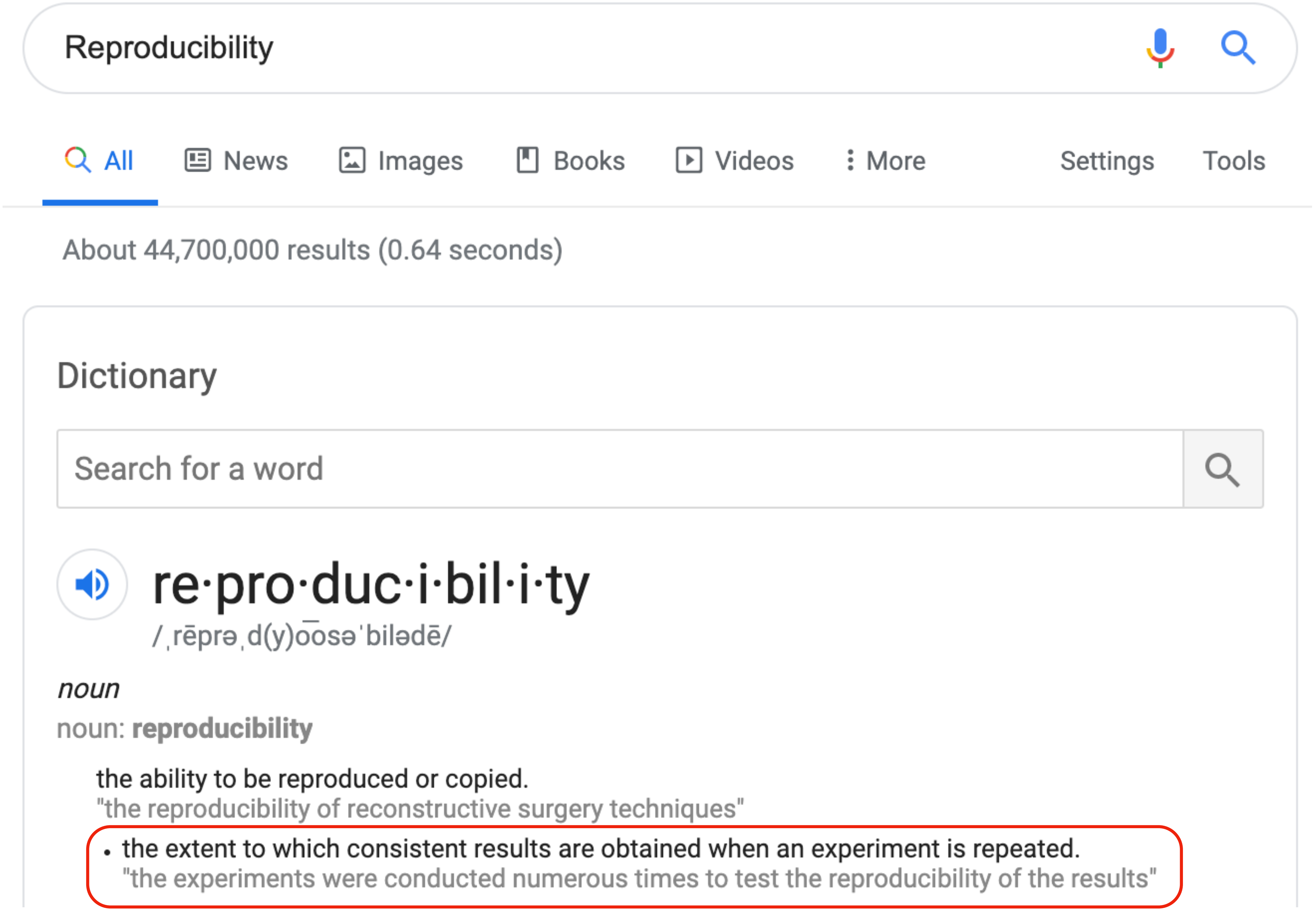
5. Learn debugging

6. Run the tester often

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

Today's Lecture:

Reproducibility



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"

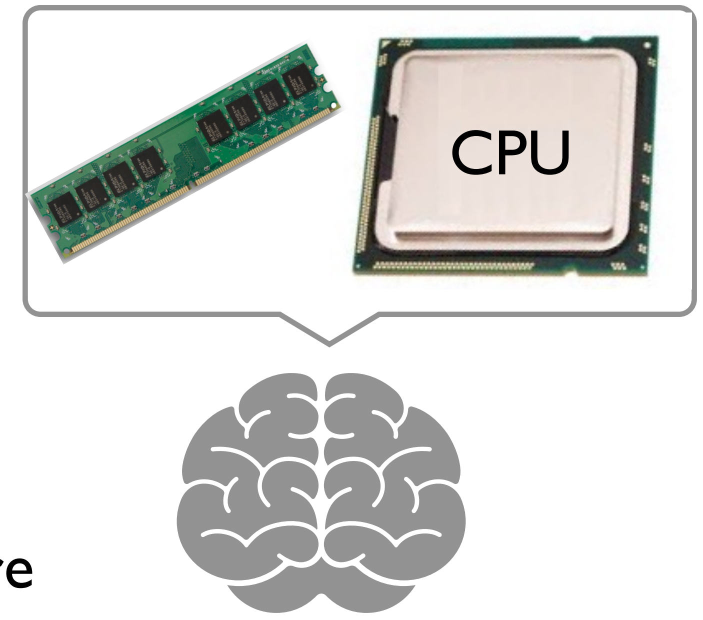
- the extent to which consistent results are obtained when an experiment is repeated.
  "the experiments were conducted numerous times to test the reproducibility of the results"

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:

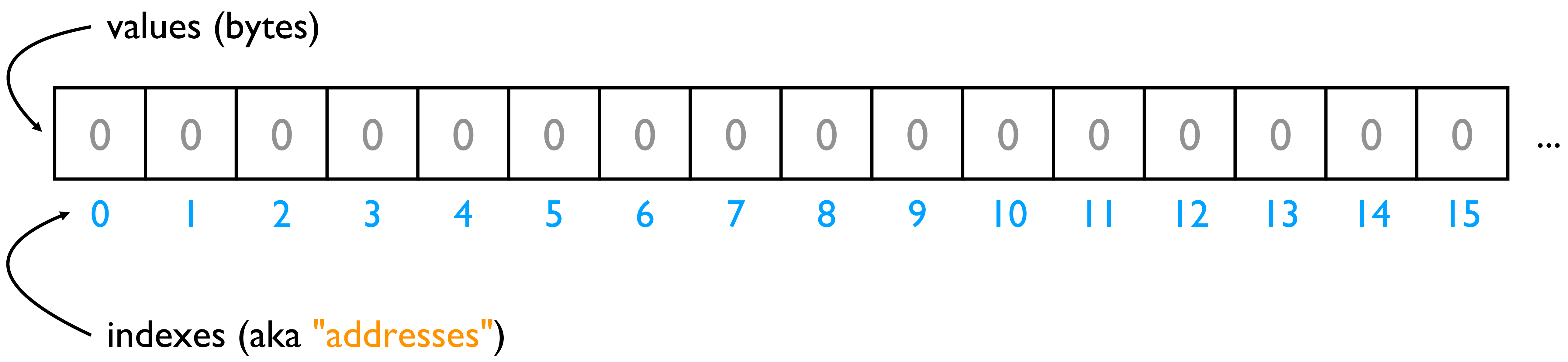
1. Hardware
2. Operating System ← next lecture
3. Dependencies ← next lecture



Hardware: Mental Model of Process Memory

Imagine...

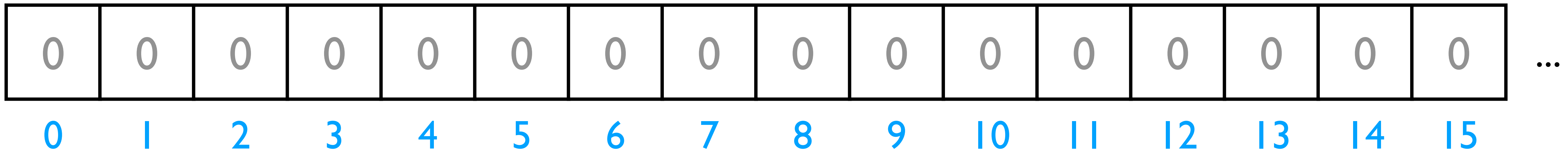
- one huge list, **per each** ~~running program~~ **process**, called **"address space"**
- every entry in the list is an integer between 0 and 255 (aka a "byte")



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

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

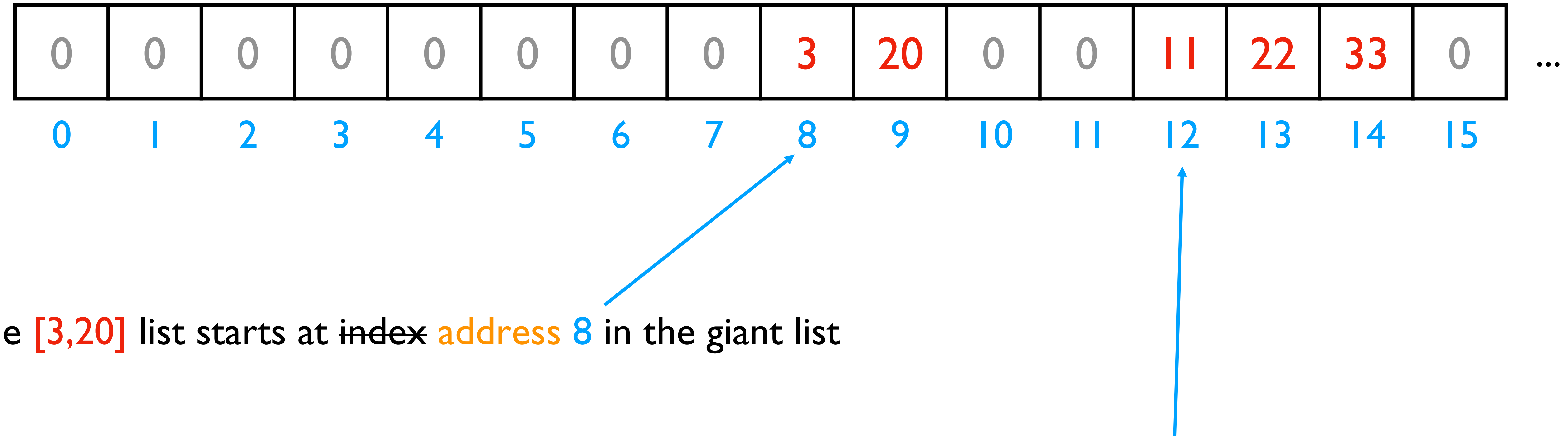
data



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

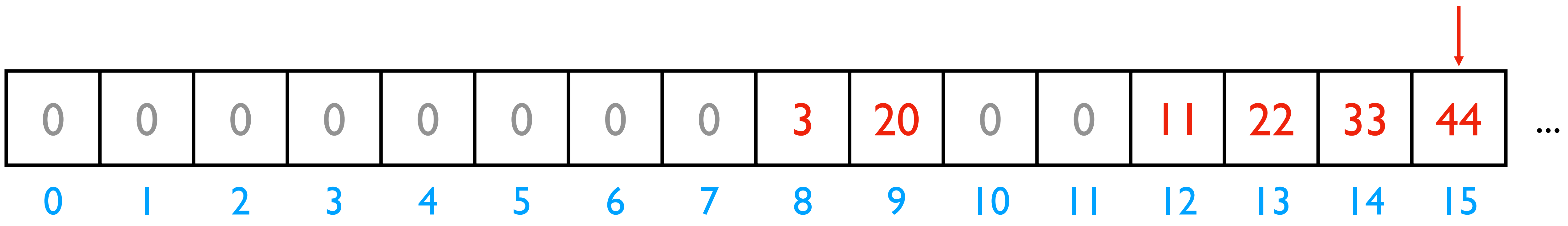
0	0	0	0	0	0	0	0	3	20	0	0	11	22	33	0	...
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	

```
# 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

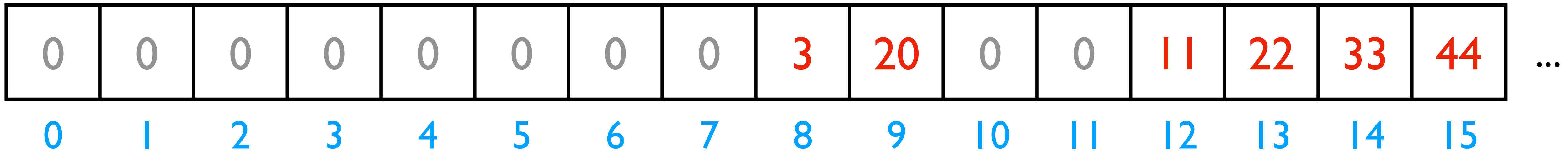


```
# fast
L2.append(44)
```

implications for performance...

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

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



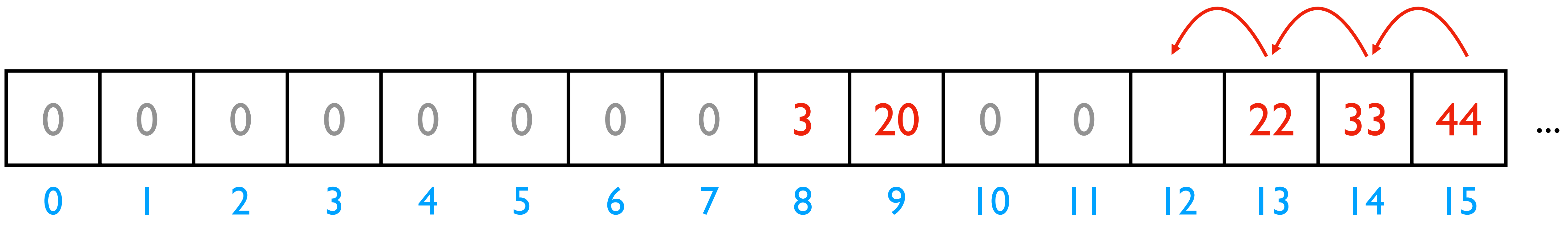
implications for performance...

```
# 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



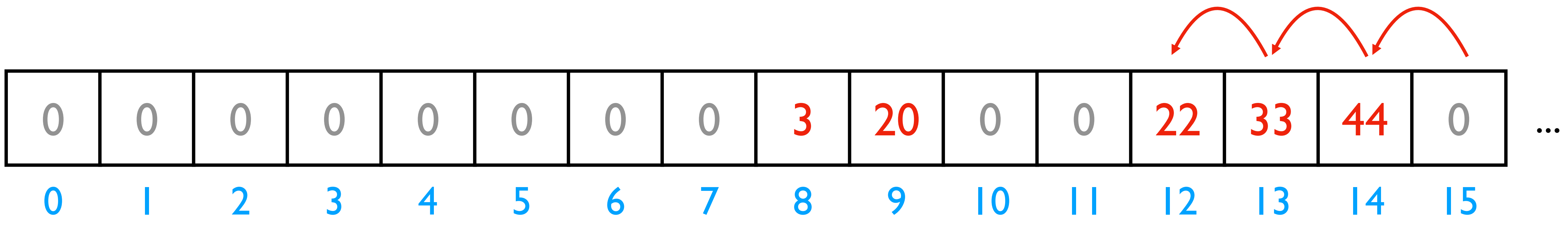
implications for performance...

```
# 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



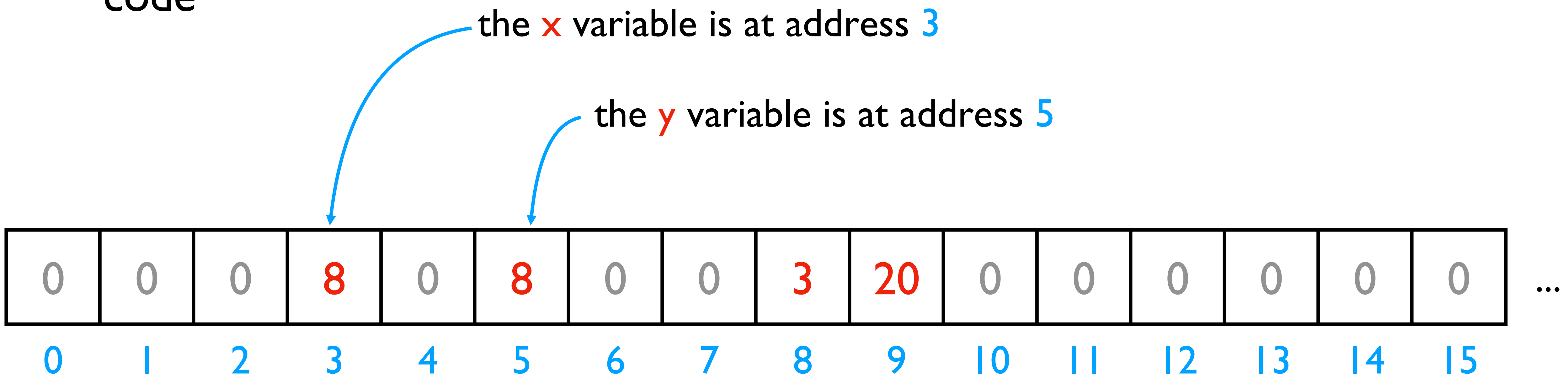
We'll think more rigorously about performance in CS 320 (big-O notation)

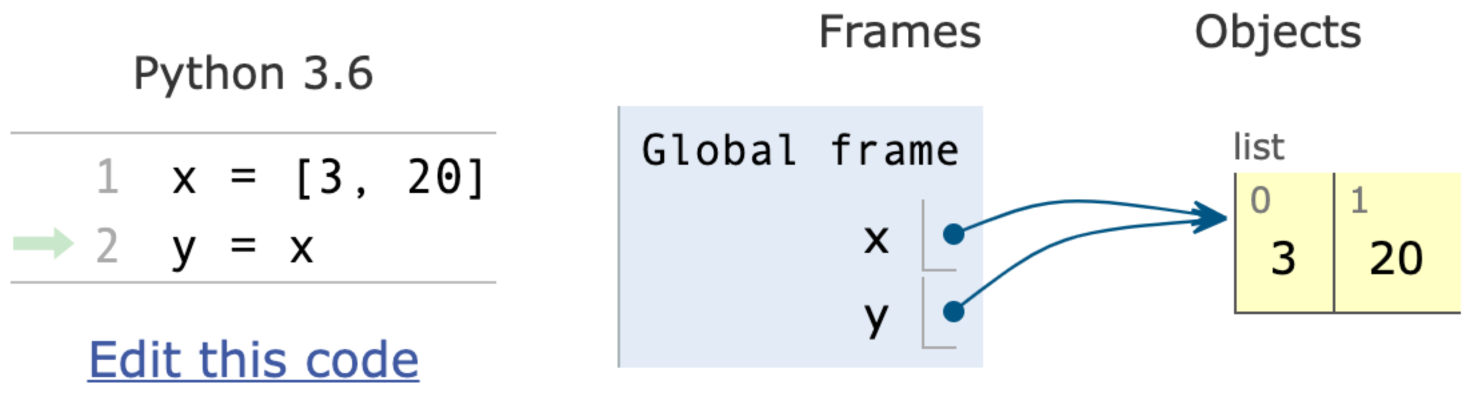
```
# fast
L2.append(44)

# slow
L2.pop(0)
```

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

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





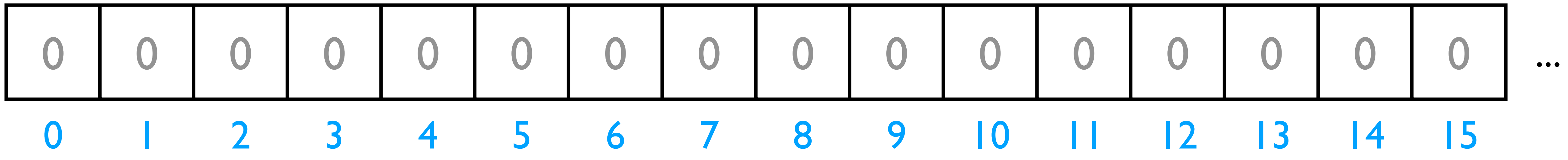
```
x = [3, 20]
y = x
```

PythonTutor's visualization

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

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

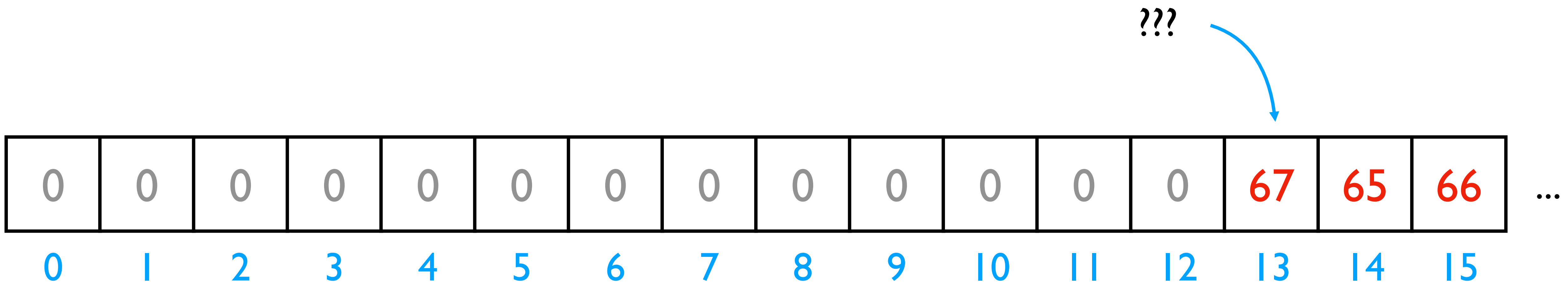
discuss: how?



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



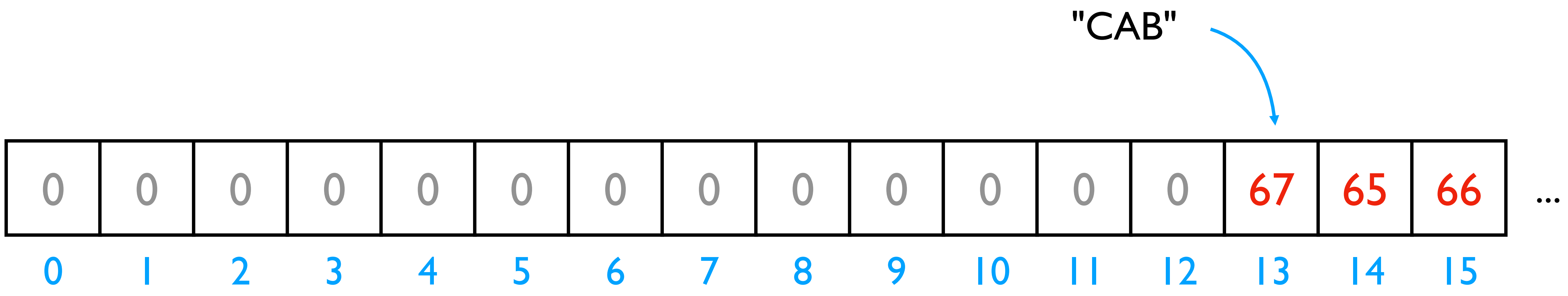
encoding:

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:

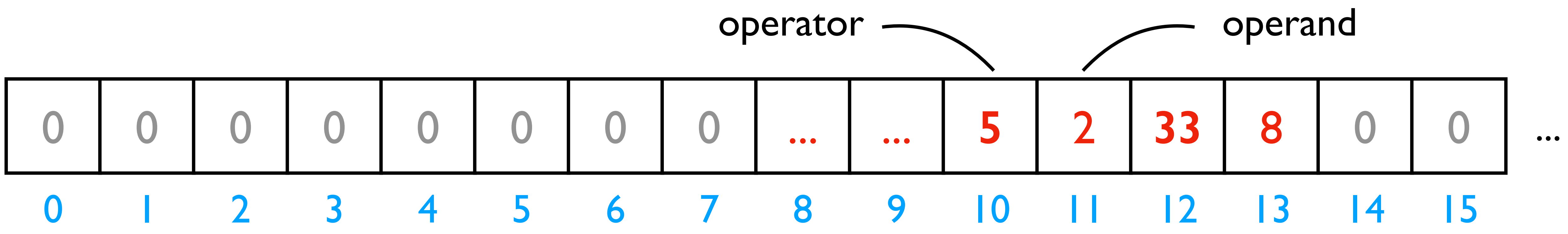
code	letter
65	A
66	B
67	C
68	D
...	...

```
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

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



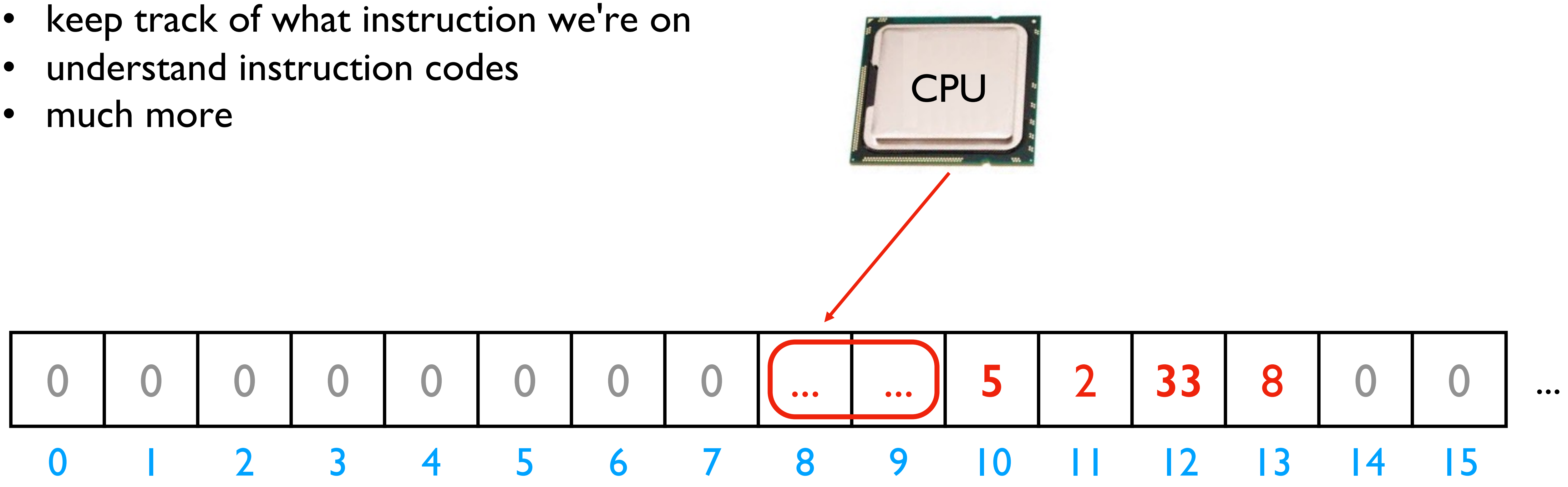
Instruction Set

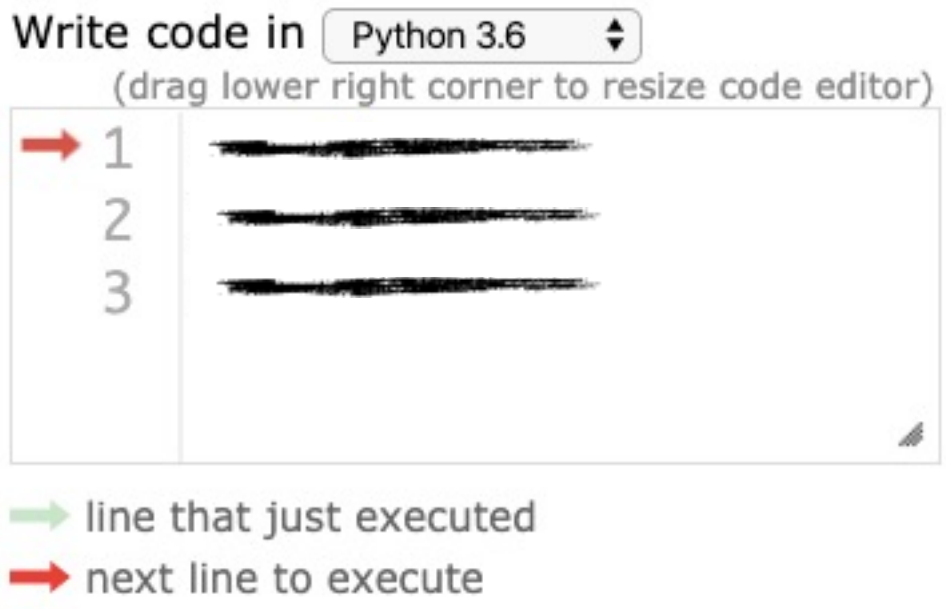
code	operation
5	ADD
8	SUB
33	JUMP
...	...

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)

[redacted]

→ line that just executed

→ next line to execute

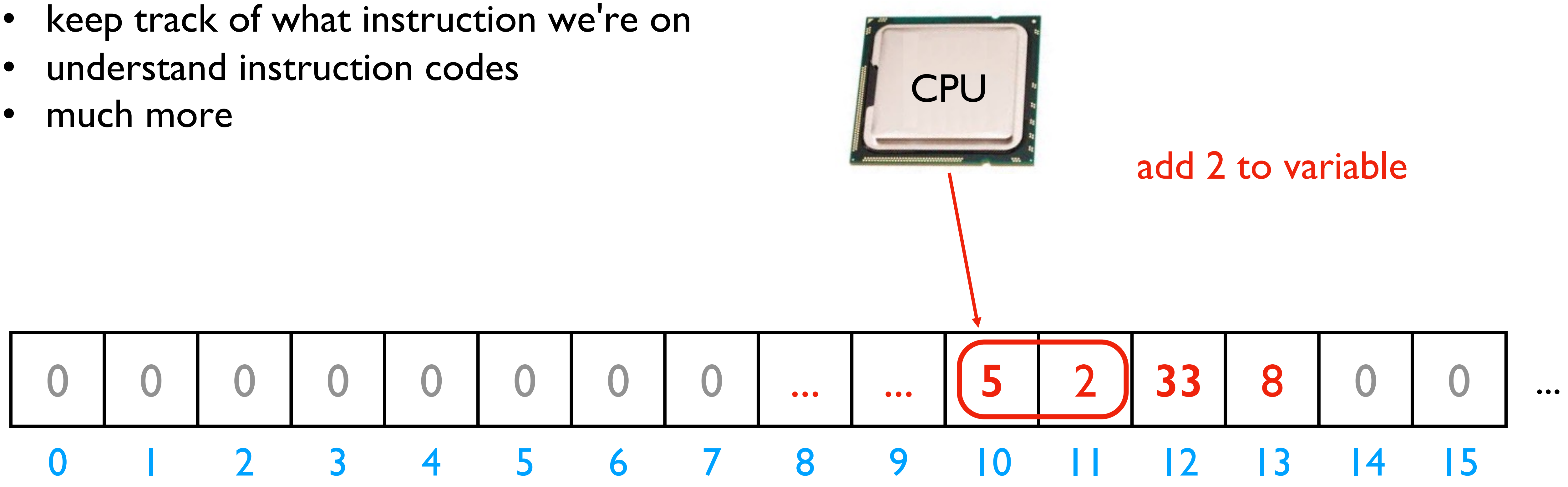
Instruction Set

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



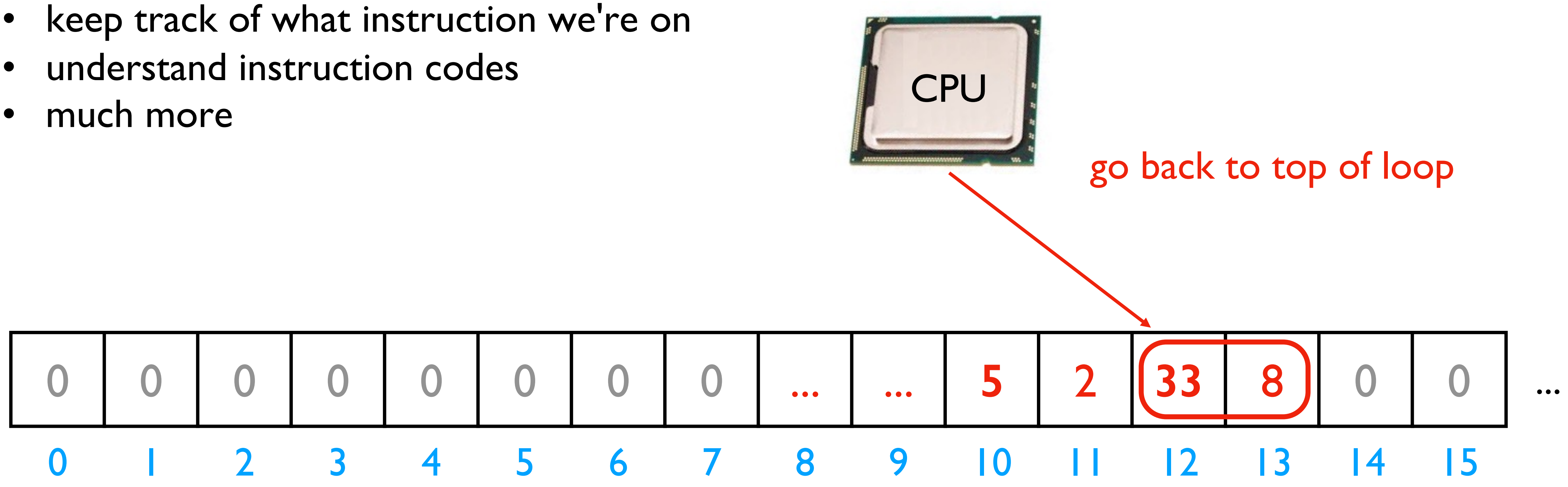
Instruction Set

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



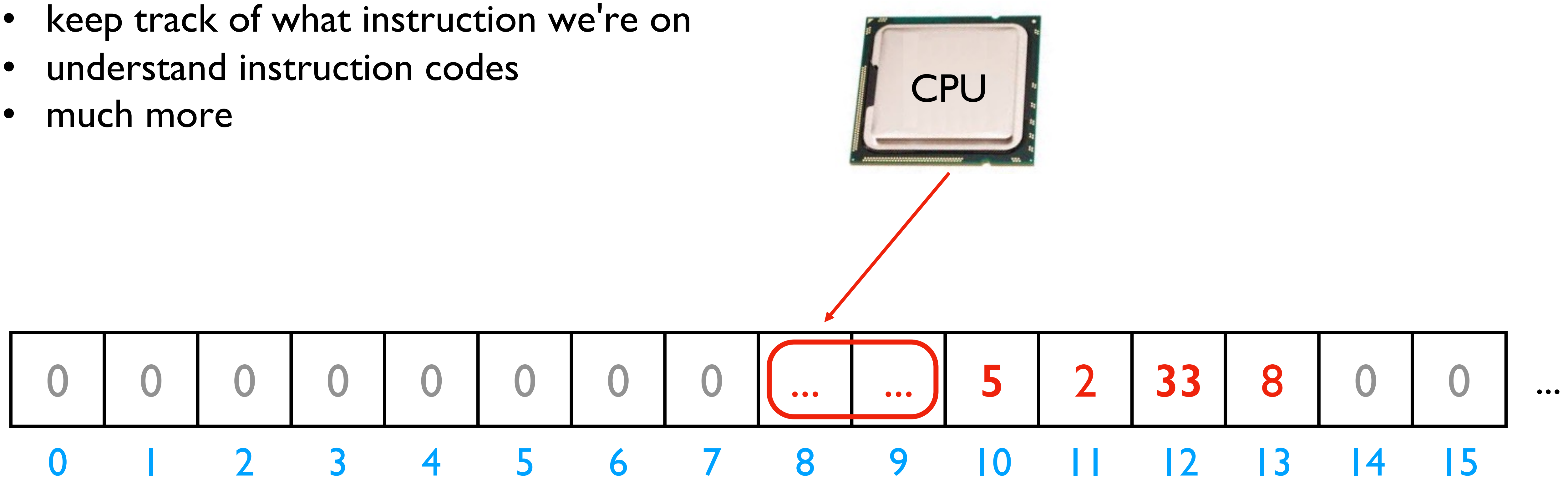
Instruction Set

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

Hardware: Mental Model of CPU

CPUs interact with memory:

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

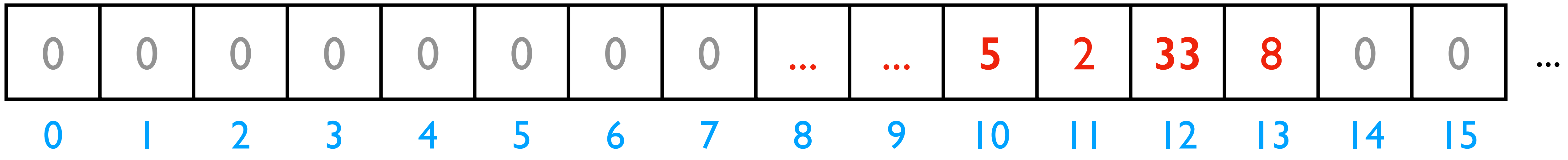


Instruction Set

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

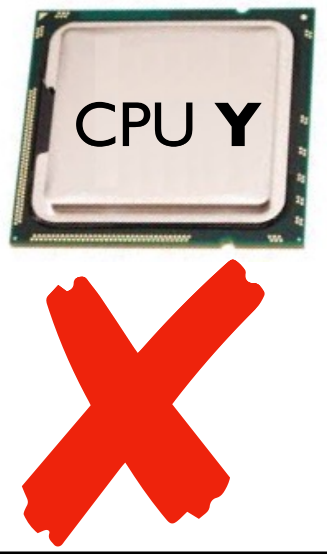
code	operation
5	ADD
8	SUB
33	JUMP
...	...

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

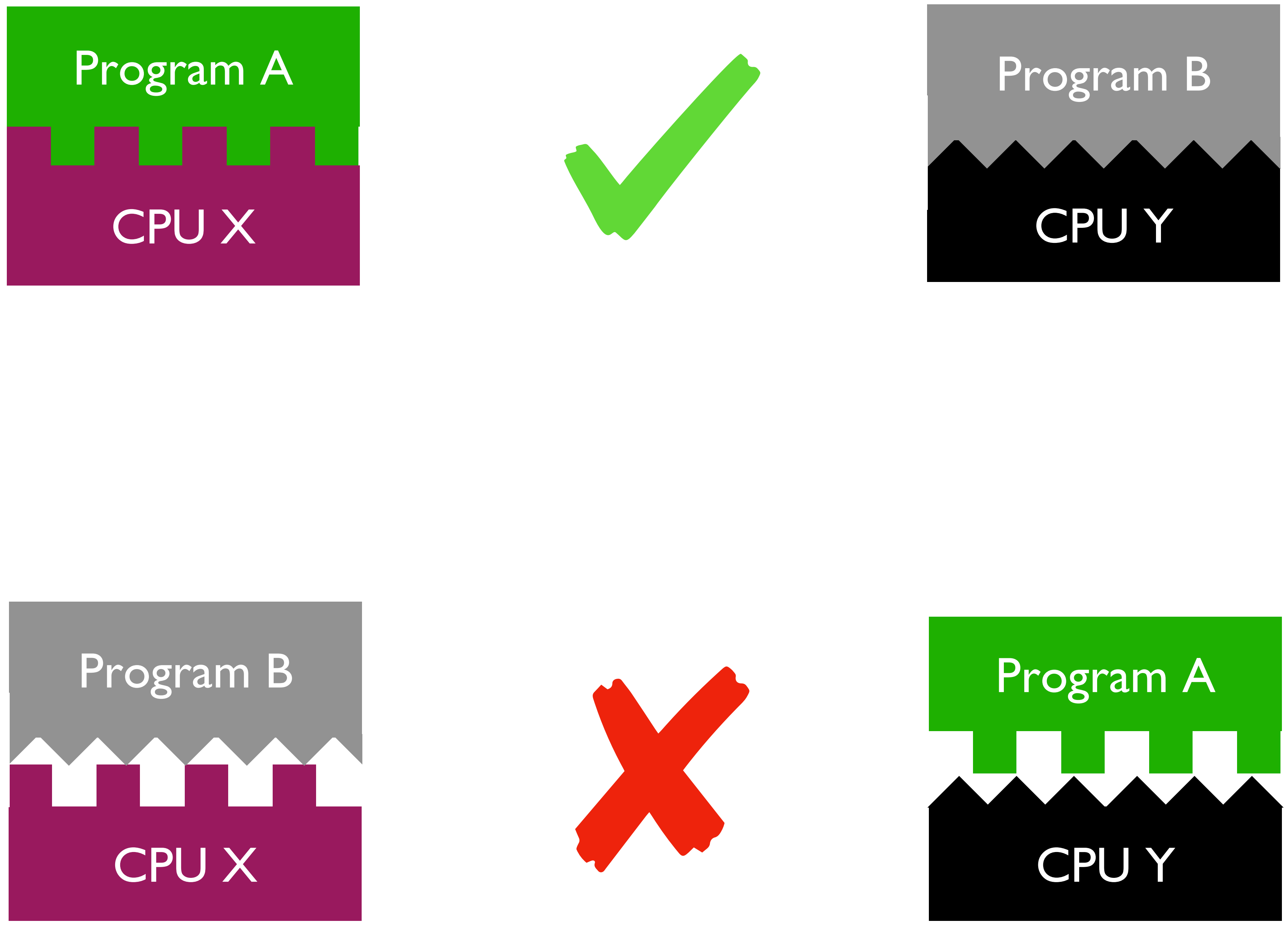
Instruction Set for CPU X

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

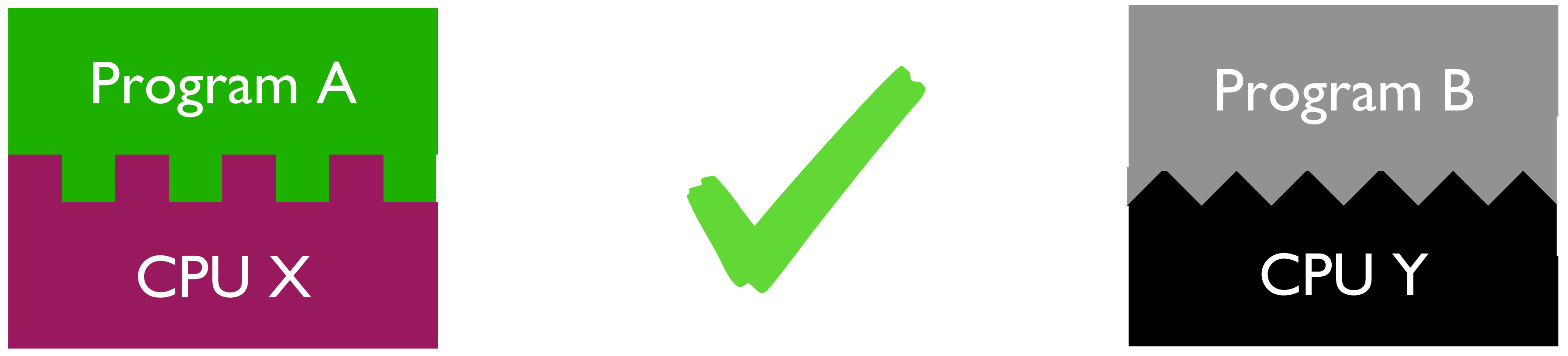
Instruction Set for CPU Y

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

A Program and CPU need to "fit"

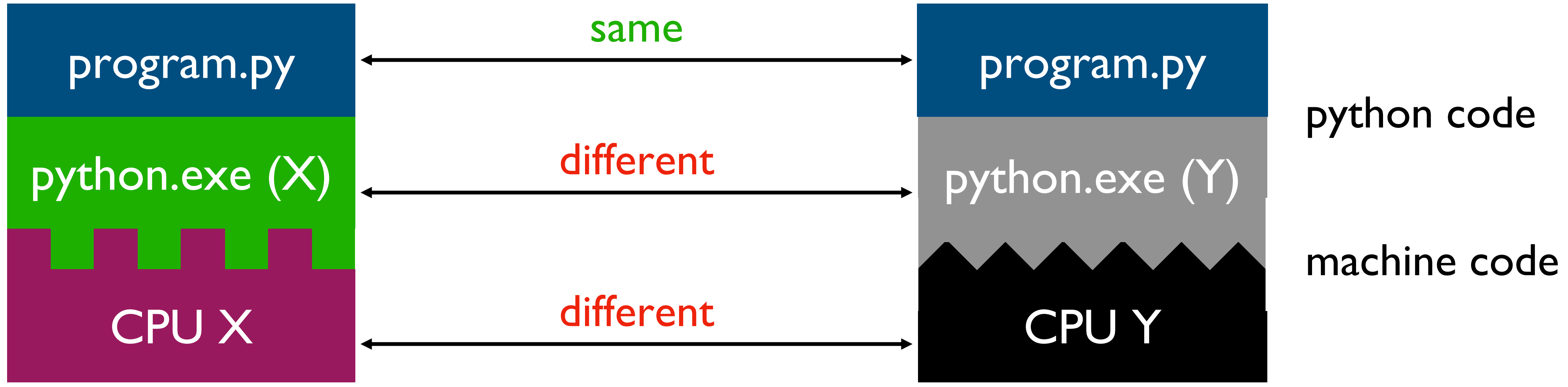


A Program and CPU need to "fit"



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

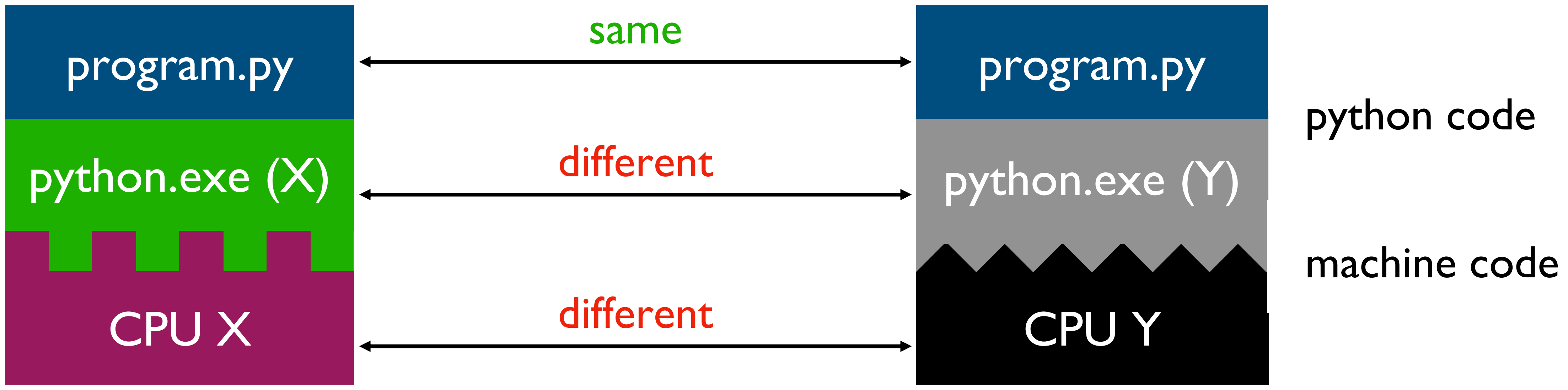
Interpreters



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?*