

# [320] Welcome + First Lecture

## [reproducibility]

Department of Computer Sciences  
University of Wisconsin-Madison

# Introductions

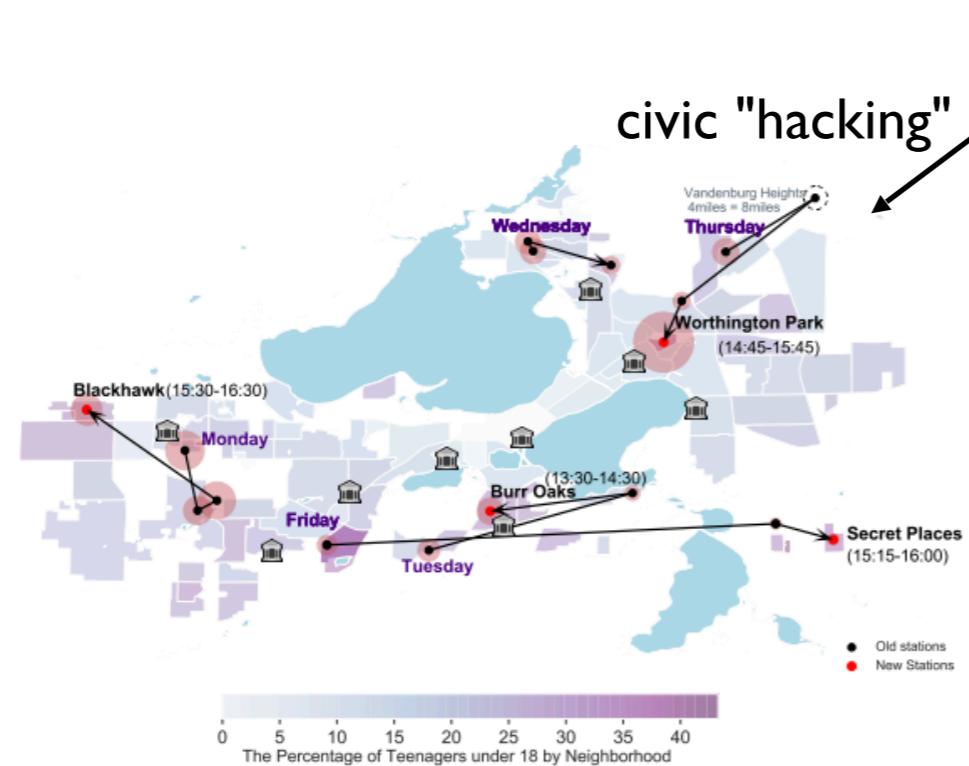
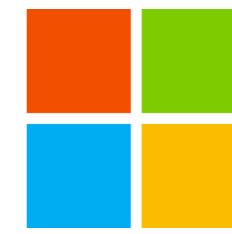
Tyler Caraza-Harter

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



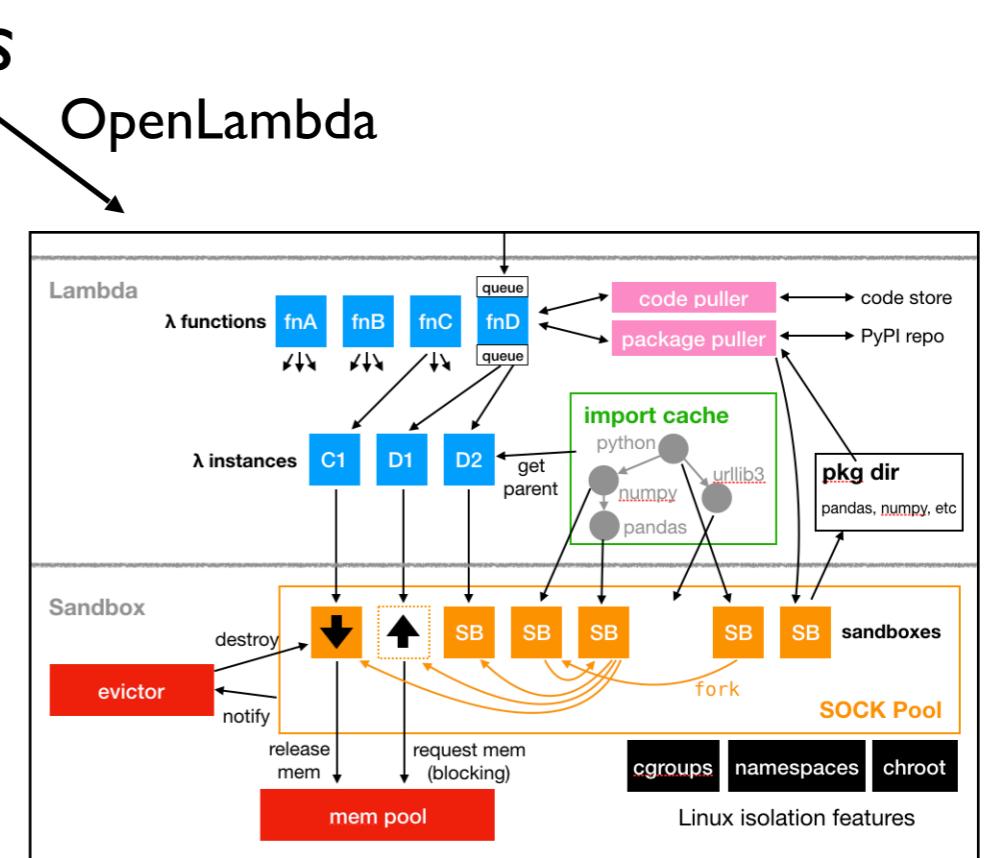
Industry experience

- 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>



# Introductions

Meenakshi (Meena) Syamkumar

- Email: [ms@cs.wisc.edu](mailto: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



# Introductions

Gurmail Singh

- Email: [Gurmail.Singh@wisc.edu](mailto:Gurmail.Singh@wisc.edu)
- Please call me “Singh” (he/him/his)

Teaching experience

- BLM Girls College, Punjab, India
- Khalsa College, Punjab, India
- University of Regina, Saskatchewan, Canada

Research Interests

- Algebra and Artificial Intelligence



# 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?

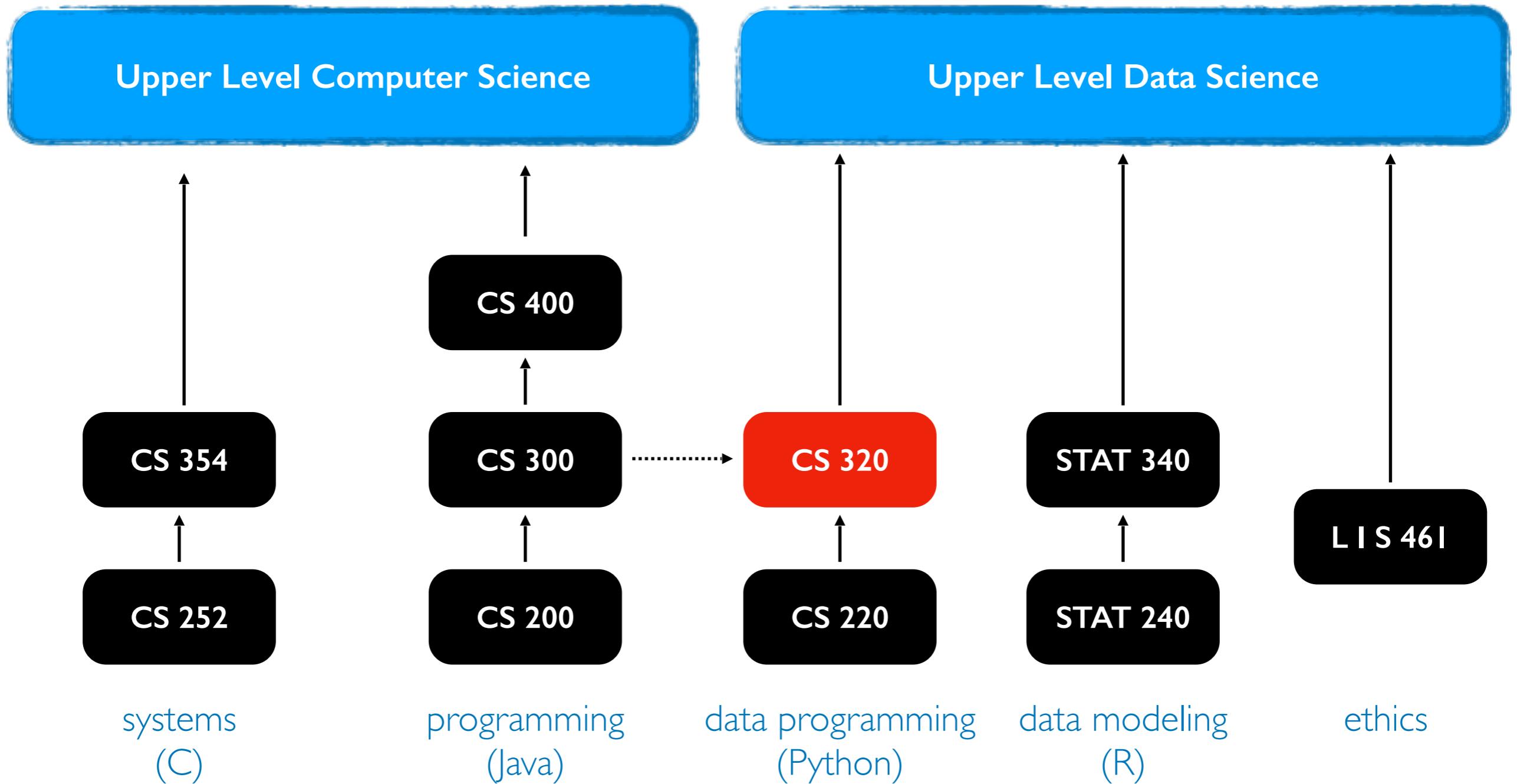
- CS 220/301? CS 200? CS 300? CS 354?

Please fill the Student Information Survey (Survey I).

## Why?

- Help me get to know you
- Get credit for completing it

# Related courses



PI (Project I) 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 <b>reproducible</b> results
writing correct code	writing <b>efficient</b> code
using objects	designing <b>new types</b> of objects
functions: <code>f(obj)</code>	<b>methods:</b> <code>obj.f()</code>
lists + dicts	graphs + trees
analyzing datasets	<b>collecting</b> + analyzing datasets
plots	animated visualizations
tabular analysis	<b>simple machine learning</b>

CS220 content (for review): <https://cs220.cs.wisc.edu/f23/schedule.html>

# Course Logistics

# Channels of Communication

- Course Website: <https://cs320.cs.wisc.edu/fa24/schedule.html>
  - Read the syllabus carefully and checkout other content on the course website
- Canvas: I will use Canvas for the following
  - General announcements
  - Quizzes/exams
  - Grade summaries
  - Online hours (Zoom links)
  - Grade Summaries
- Email: Students are supposed to check their email regularly for possible emails from me.

# Scheduled Activities

## Lectures

- 2 times weekly; recommendation: bring your laptop
- Required for participation credit! Attendance recorded via TopHat quizzes (20% score drops) as mentioned in the syllabus
- 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, and recordings may be edited.

## Lab

- Weekly on Tuesdays and Wednesdays, bring a laptop
- Work through lab exercises with group mates
- 320 staff will be available to answer questions
- Required for lab attendance credit! 3 score drops

# 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 is not allowed

## Staff

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

We all provide office hours.  
For details, please read Get Help page on  
the course website.

# Communication

## Piazza

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

## Forms

- <https://cs320.cs.wisc.edu/fa24/surveys.html>
- **Exam conflicts.** Grading Issues. Feedback form. Thank you form!

## Email (least preferred)

- me: [gurmail.Singh@wisc.edu](mailto:gurmail.Singh@wisc.edu)
- Course staff: <https://canvas.wisc.edu/courses/427084/pages/cs320-staff>
- Email is the least preferred form of communication. We will not provide debugging help via email - please attend office hours for that.

# Graded Work: Exams / Quizzes

## Ten Online Quizzes - 1% each (10% overall)- I score drops

- cumulative, two attempts, no time limit
- score will be average of both attempts
- on Canvas, open book/notes
- can take together AT SAMETIME with team members  
(no other human help allowed)

## Midterms - 10% each (20% overall)

- cumulative, individual, multi-choice, 40 minutes
- one-page two-sided note sheet allowed

## Final - 10%

- cumulative, individual, multi-choice
- 2 hours (Probably less than the scheduled time will be used for the final exam)
- one-page two-sided note sheet allowed

# Graded Work: Projects

**8 Projects - 50% overall (for details, read the syllabus)**

- **format:** notebook, module, or program
- regular deadlines are 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
- clearing auto-grader on the submission portal is mandatory
- you may not be allowed to use any late days on the last project.
- **ask for specific feedback** (constructive)

# Graded Work: Attendance + Surveys

## Lab attendance - 5% overall

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

## Lecture attendance - 4% 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 semester

## 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

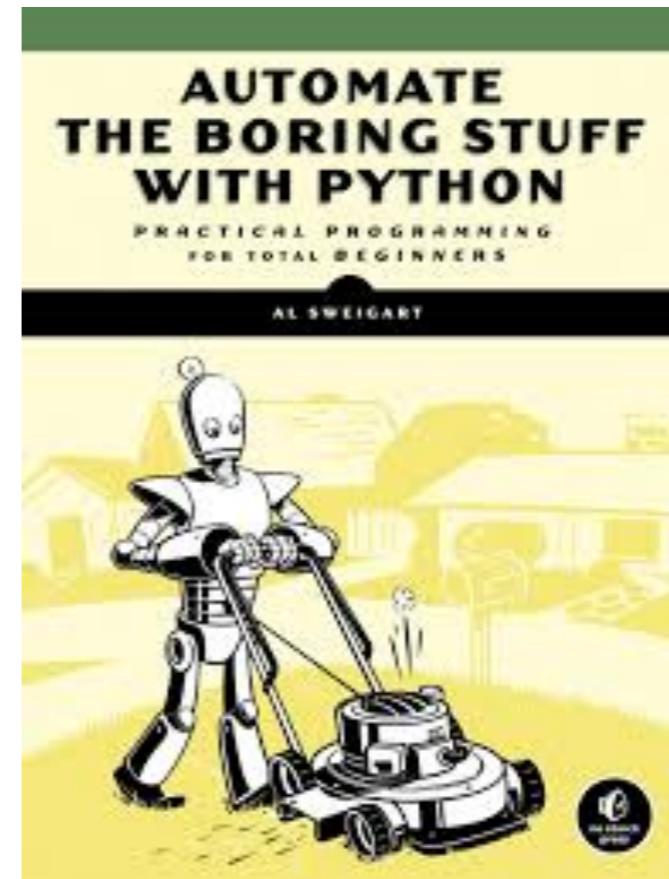
- 4 hours - lecture/lab
- 6 hours - project coding
- 2 hours - reading/quizzes/etc

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...



I may 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

[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"

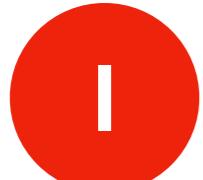
- the extent to which consistent results are obtained when an experiment is repeated.

"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:



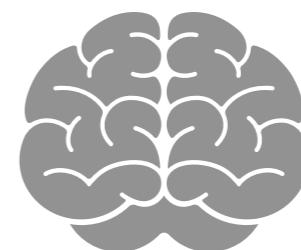
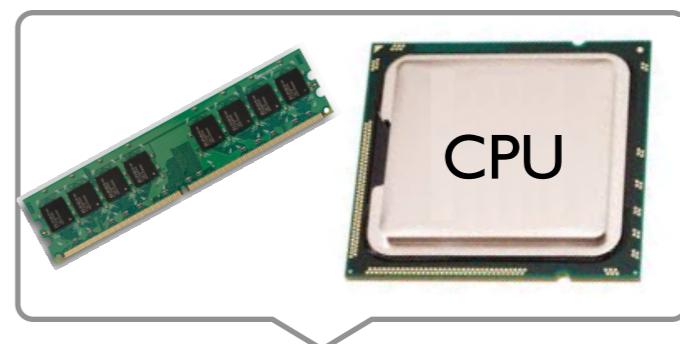
Hardware



Operating System



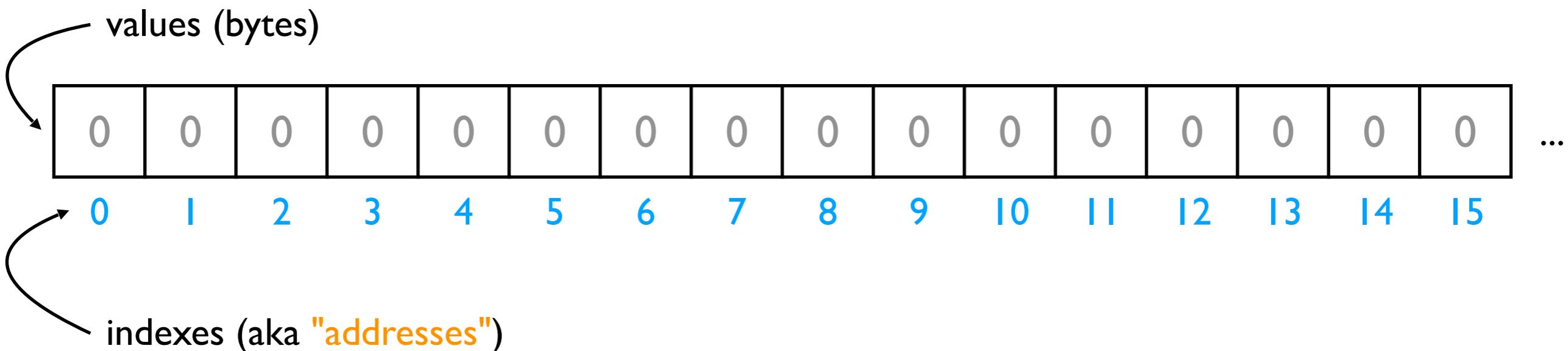
Dependencies



# Hardware: Mental Model of Process Memory

*Imagine...*

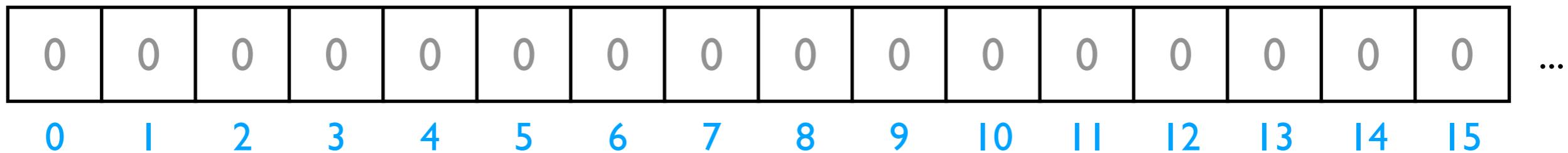
- one huge list, **per each running program process**, called "**address space**"
- every entry in the list is an integer between 0 and 255 (aka a "**byte**")



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

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

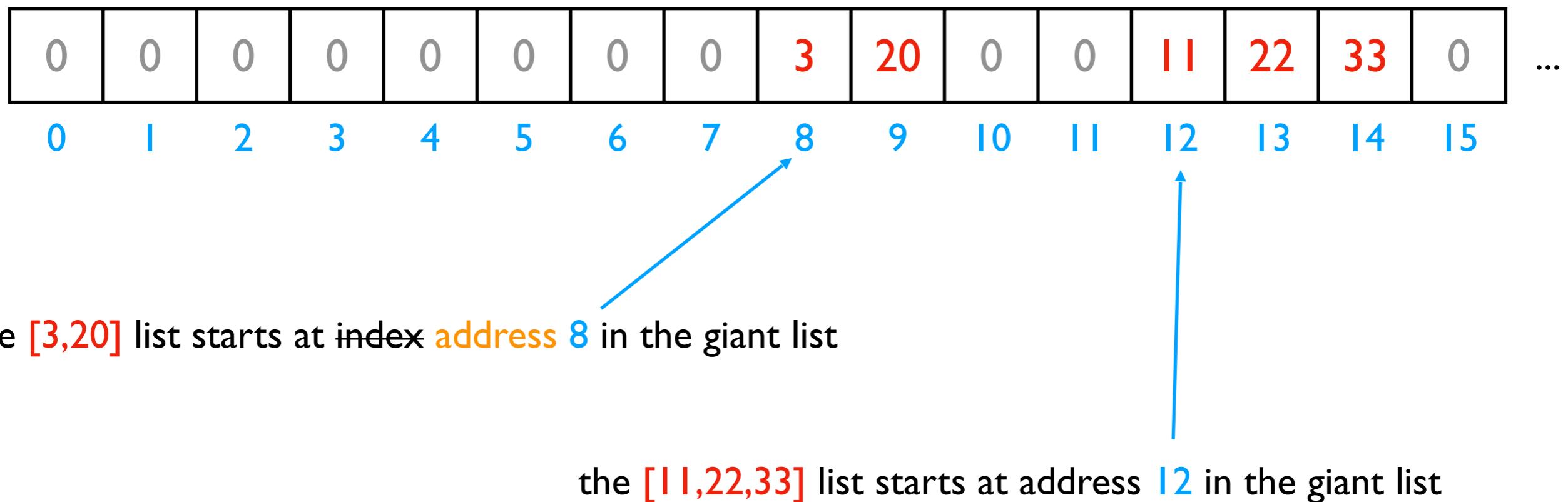
data



*Is this really all we have for state?*

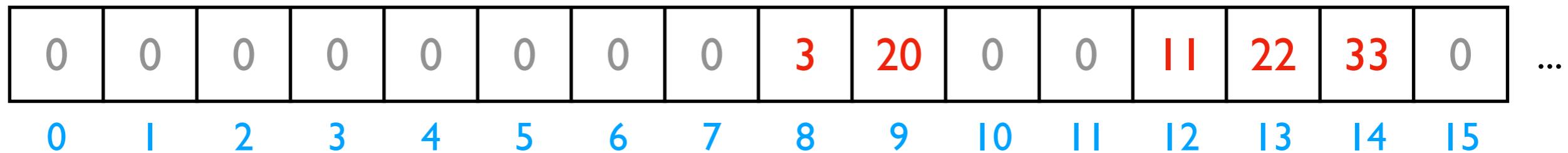
How can we use one giant list to handle the following?

- 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

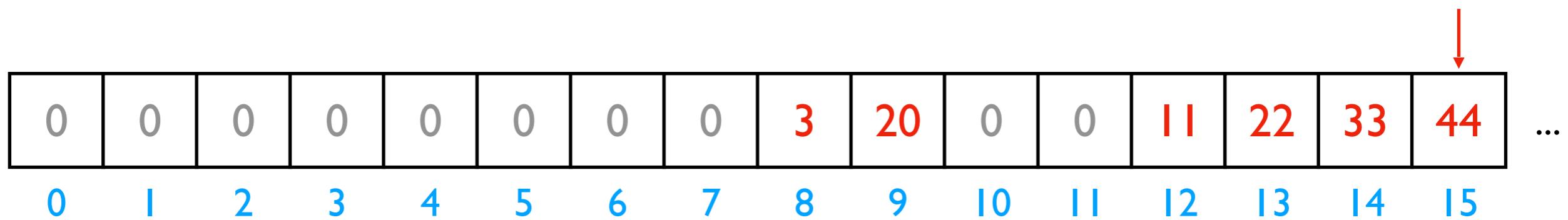


*implications for performance...*

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

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

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

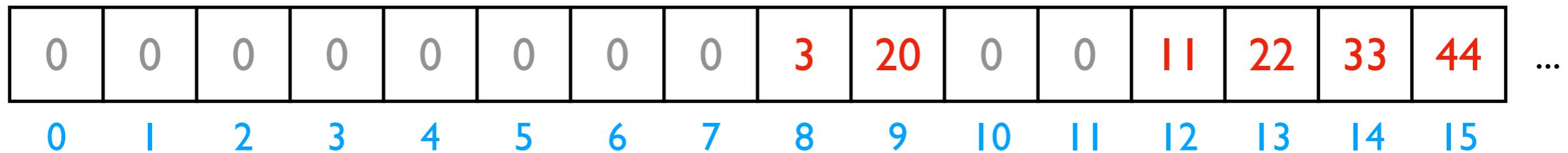


# fast  
L2.append( 44 )

*implications for performance...*

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

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



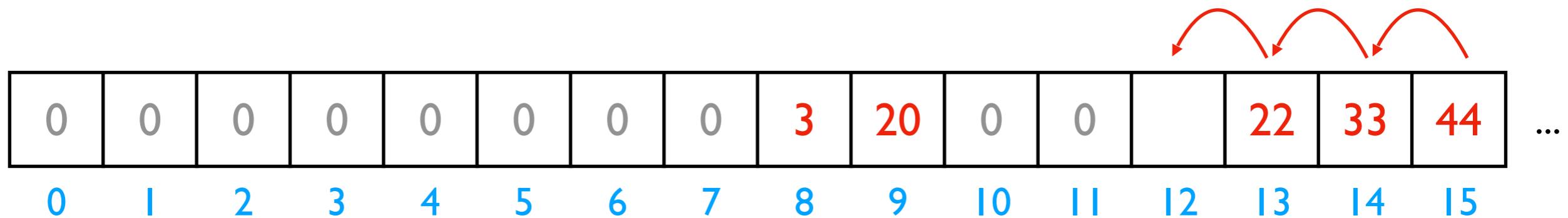
*implications for performance...*

```
# 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



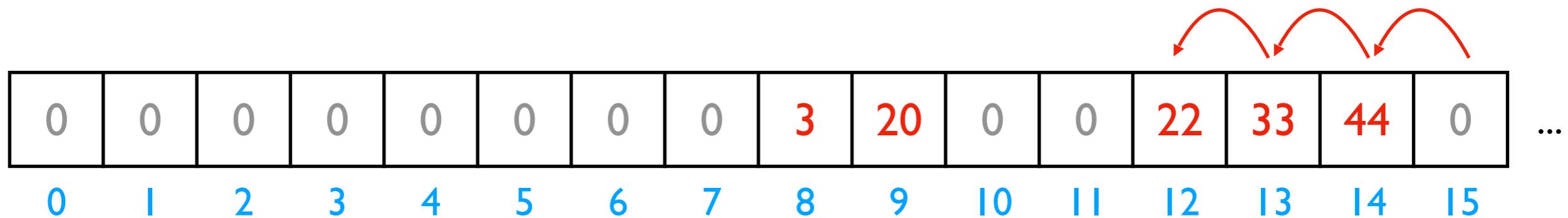
*implications for performance...*

```
# 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



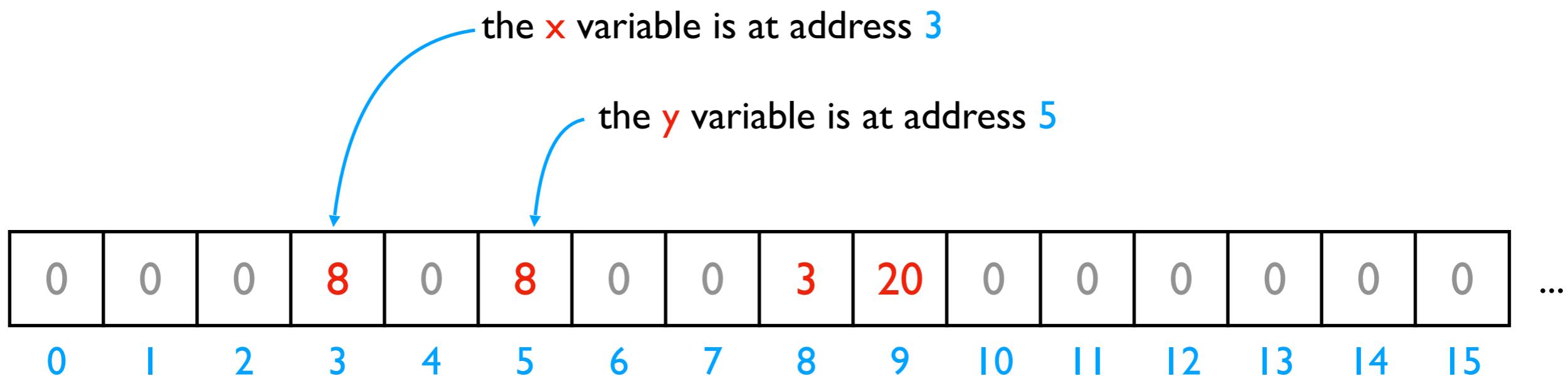
We'll think more rigorously about  
performance in CS 320 (big-O notation)

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

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

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

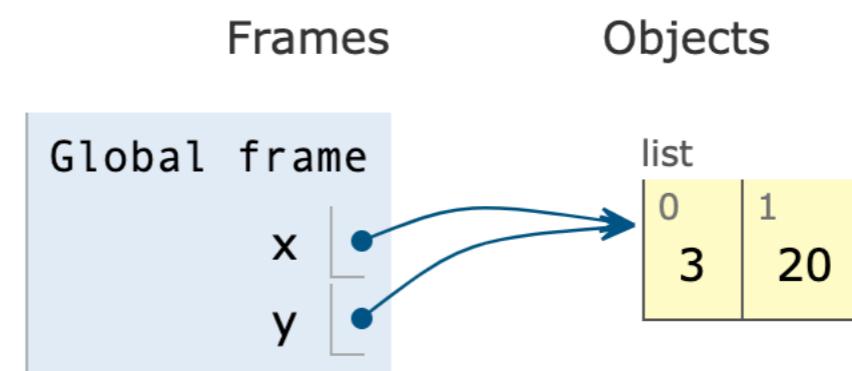
- multiple lists
- **variables and other references**
- strings
- code



Python 3.6

```
1 x = [3, 20]
→ 2 y = x
```

[Edit this code](#)

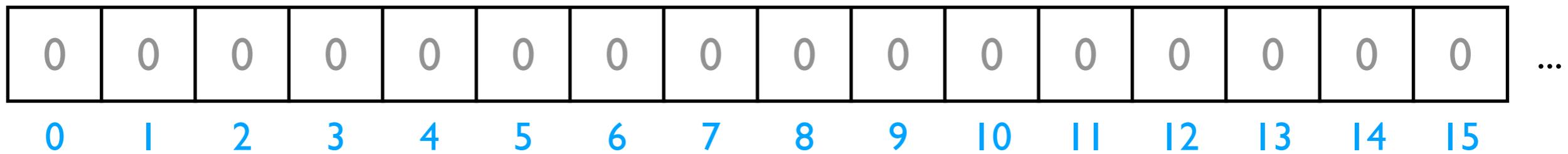


PythonTutor's visualization

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

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

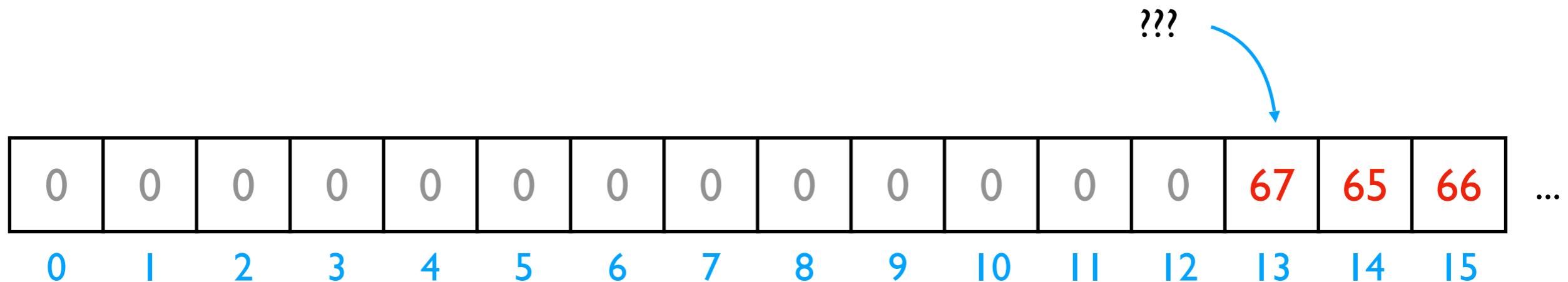
discuss: how?



*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



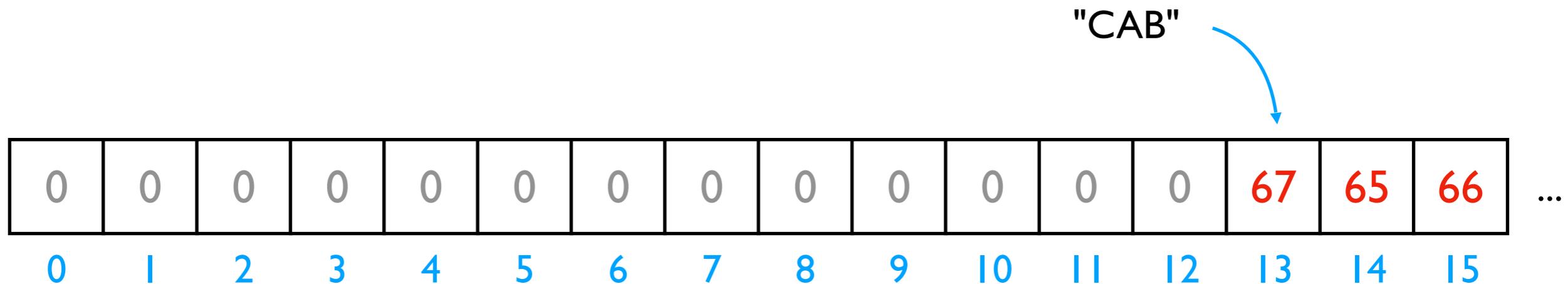
encoding:	code	letter
	65	A
	66	B
	67	C
	68	D

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



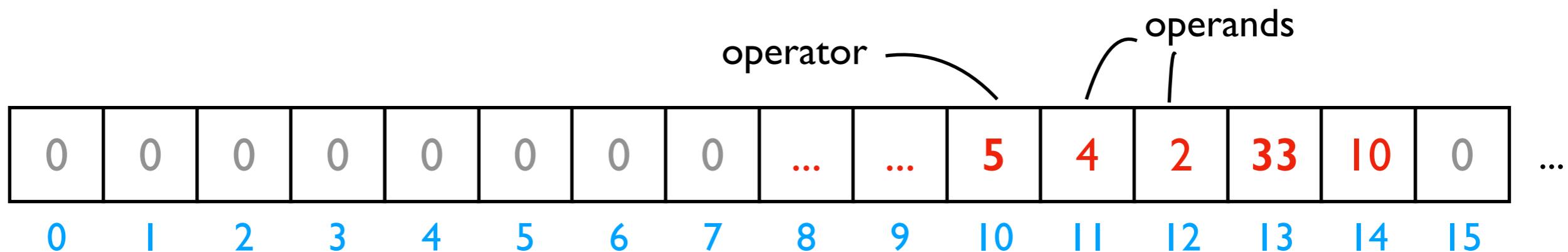
encoding:	code	letter
	65	A
	66	B
	67	C
	68	D
	...	...

```
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?
```

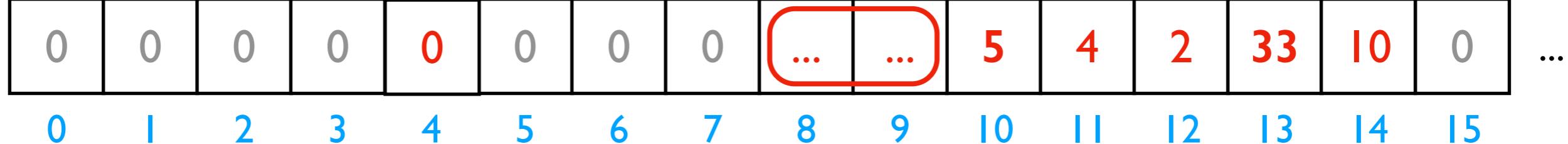


code	operation
5	ADD
8	SUB
33	JUMP
...	...

# 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
```

A code editor window showing three lines of code. The first line has a red arrow pointing to it, indicating the next line to execute. The second line has a green arrow pointing to it, indicating the line that just executed.

Instruction Set

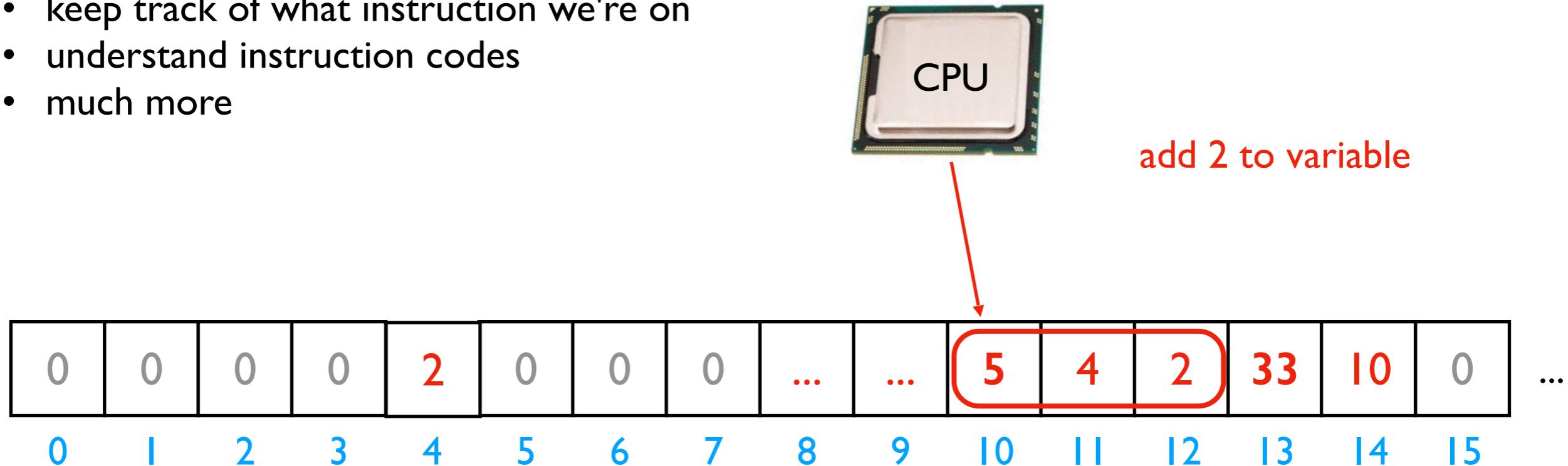
code	operation
5	ADD
8	SUB
33	JUMP
...	...

→ 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



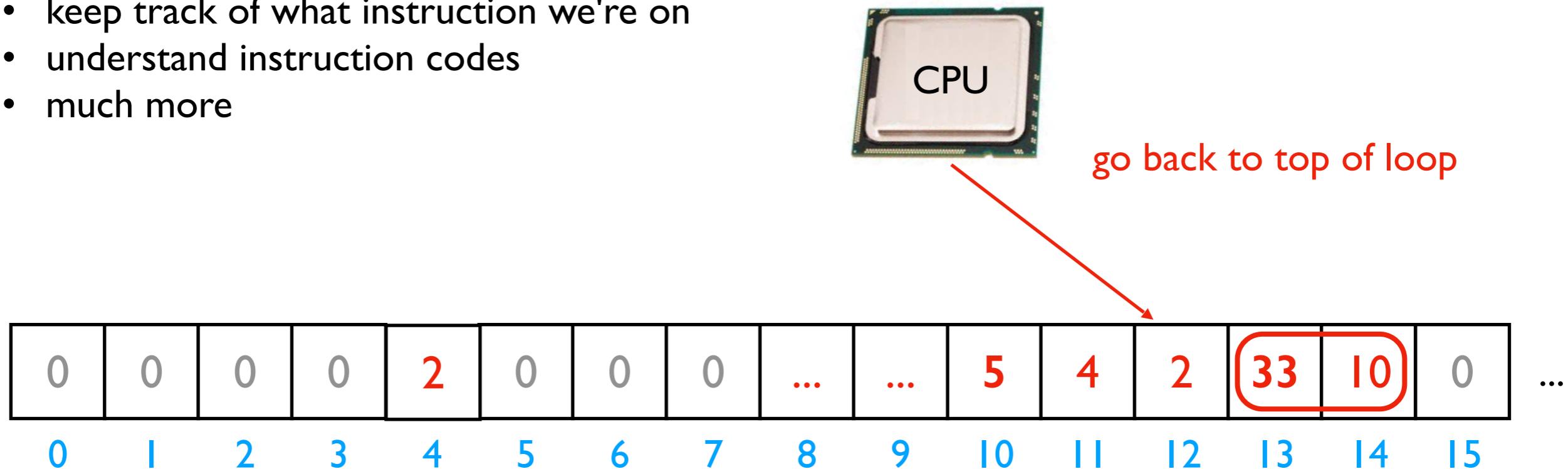
code	operation
5	ADD
8	SUB
33	JUMP
...	...

Instruction Set

# Hardware: Mental Model of CPU

CPUs interact with memory:

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

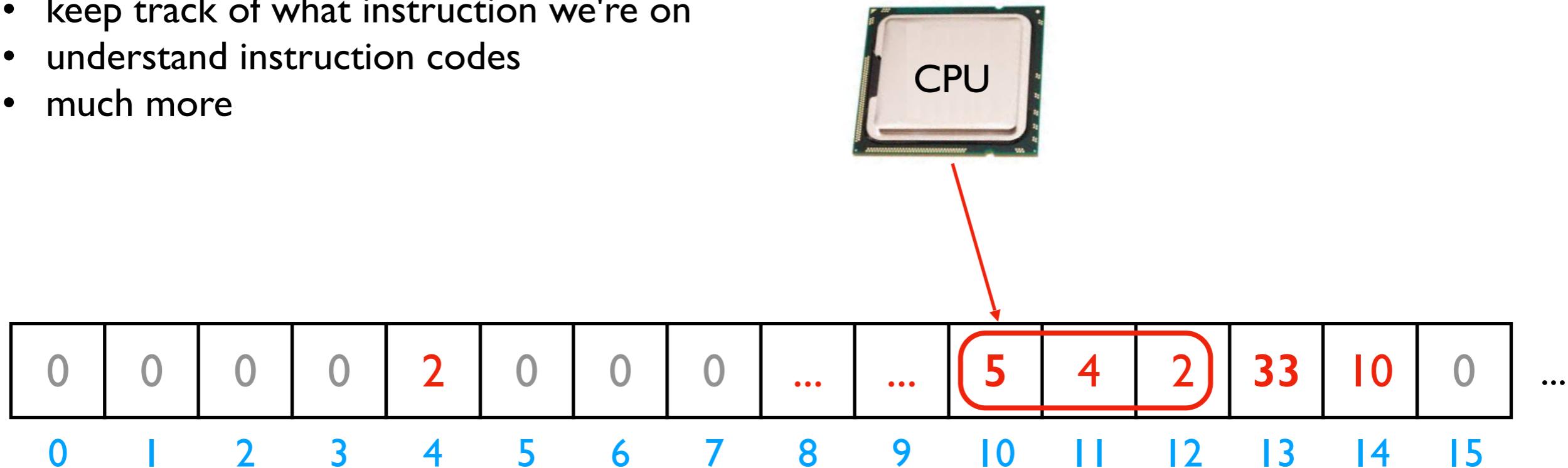


	code	operation
Instruction Set	5	ADD
	8	SUB
	33	JUMP
	...	...

# Hardware: Mental Model of CPU

CPUs interact with memory:

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

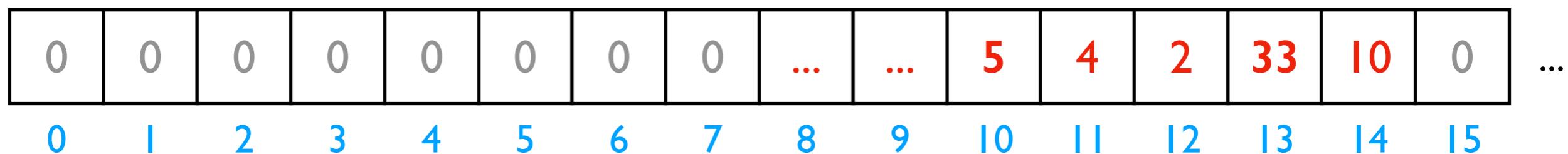


code	operation
5	ADD
8	SUB
33	JUMP
...	...

Instruction Set

# Hardware: Mental Model of CPU

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

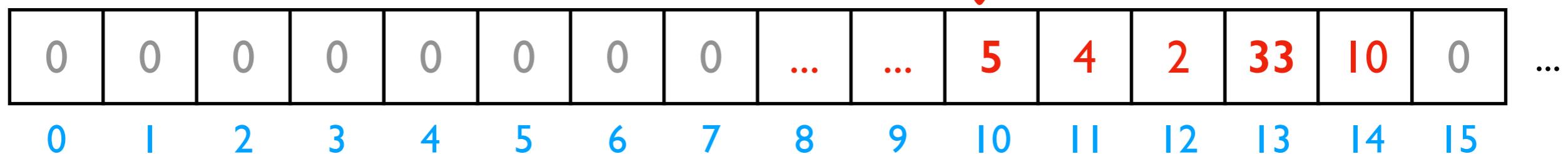


Instruction Set for CPU X	code	operation
	5	ADD
	8	SUB
	33	JUMP
	...	...

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

# Hardware: Mental Model of CPU

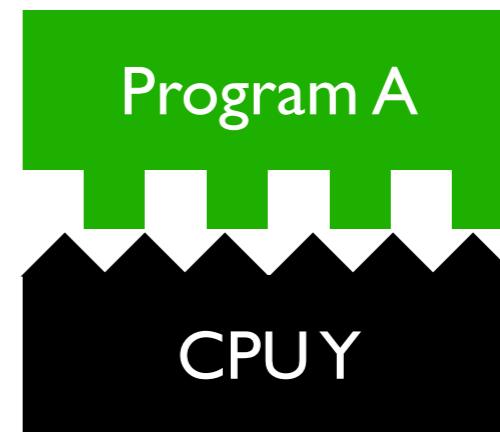
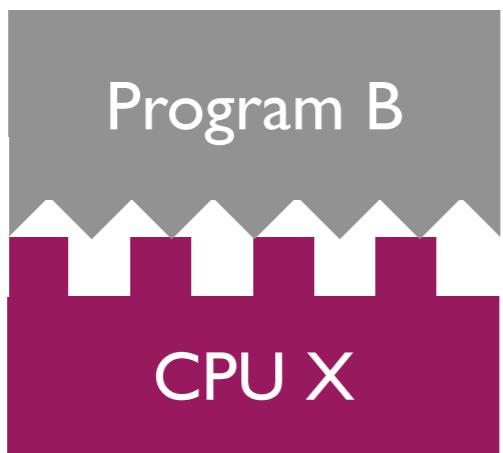
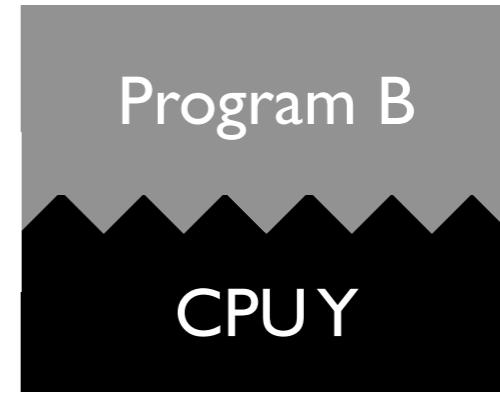
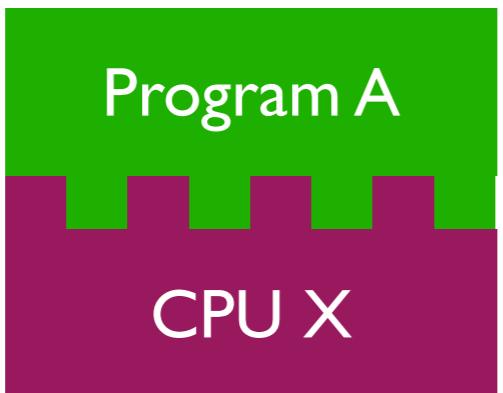
a CPU can only run programs that use instructions it understands!



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"

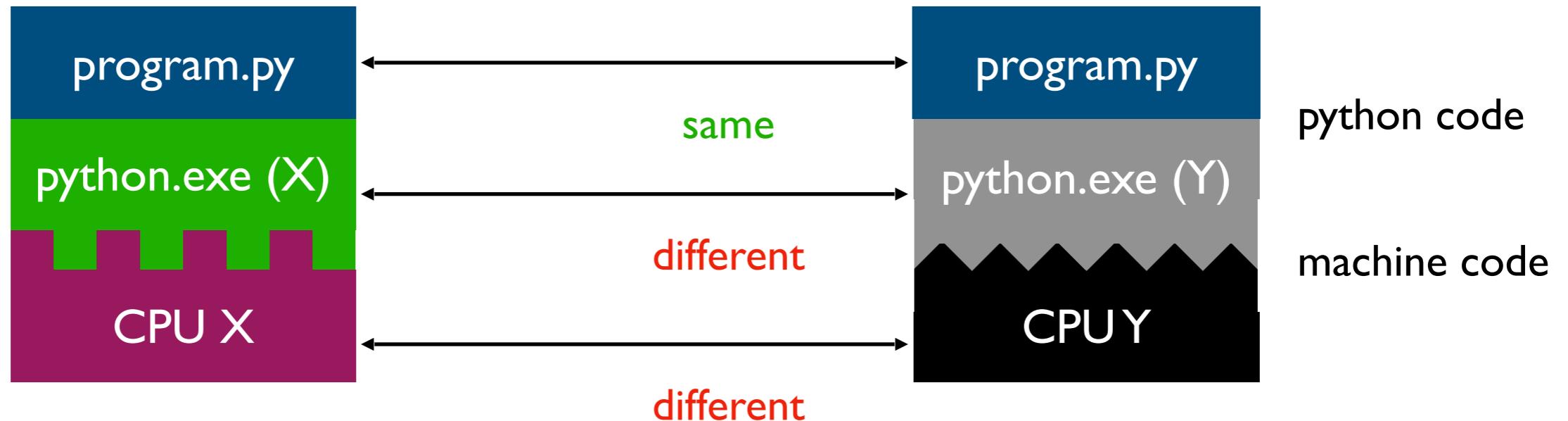


# A Program and CPU need to "fit"



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

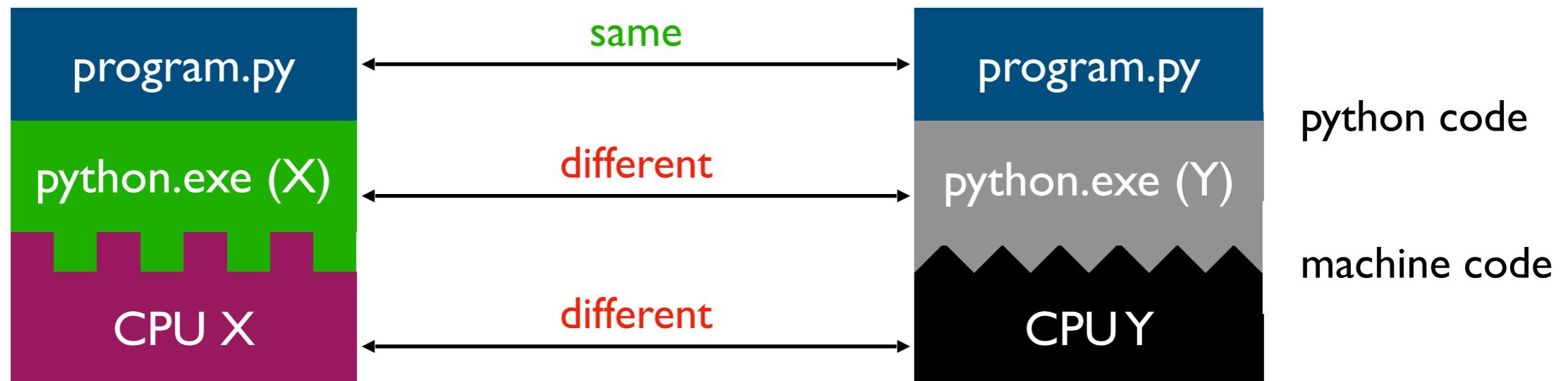
# Interpreters



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

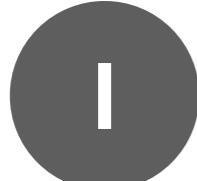


Interpreters (such as `python.exe`) make it easier to run the same code on different machines

**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:



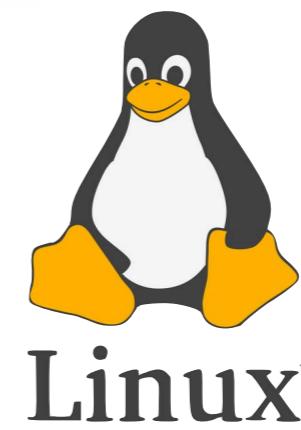
Hardware



Operating System

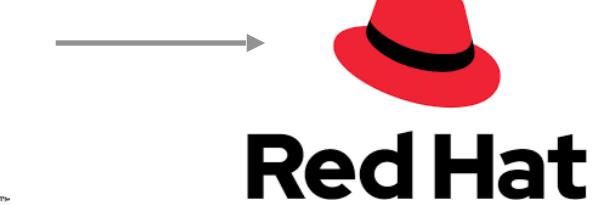


Dependencies



ANDROID

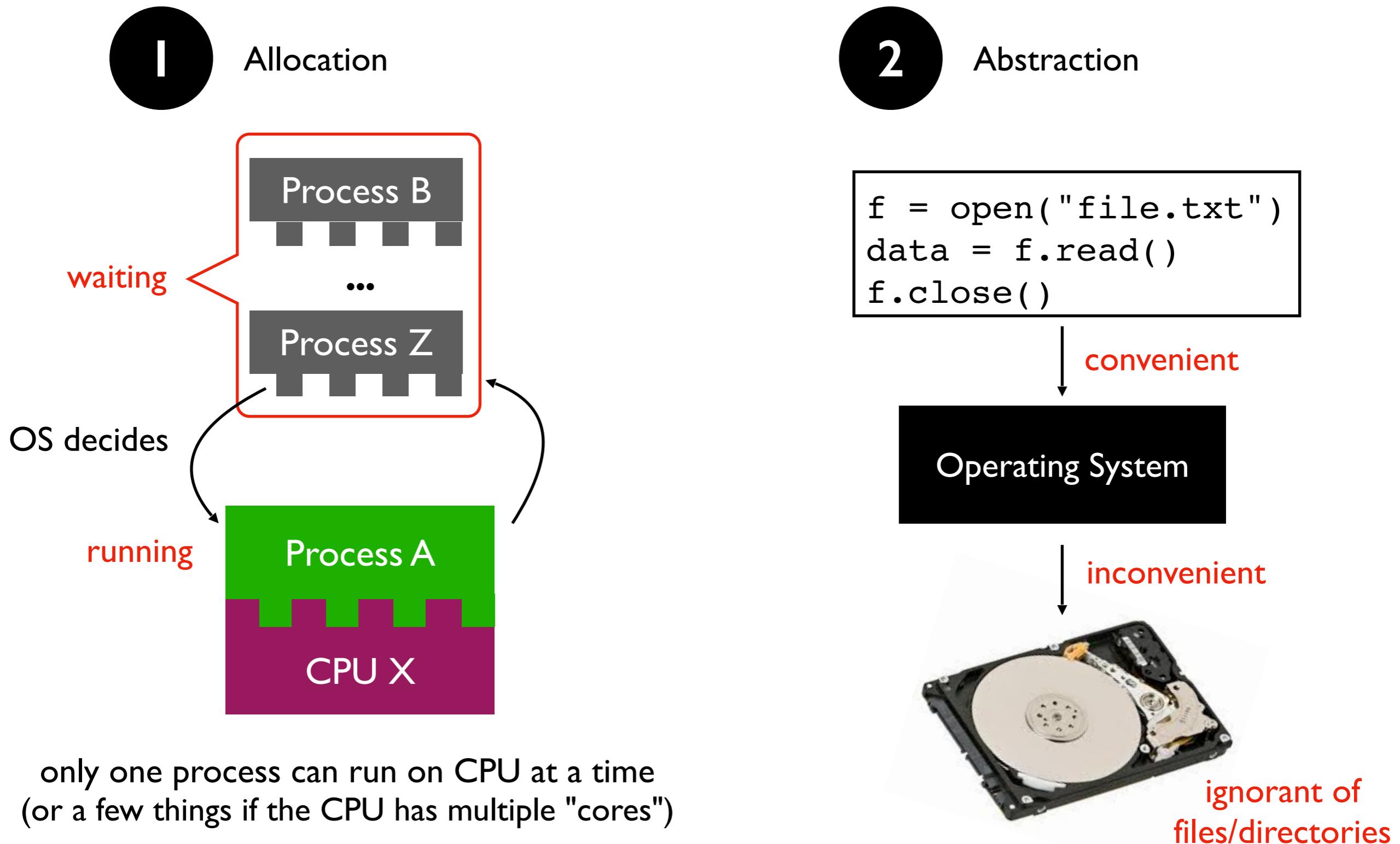
many others...



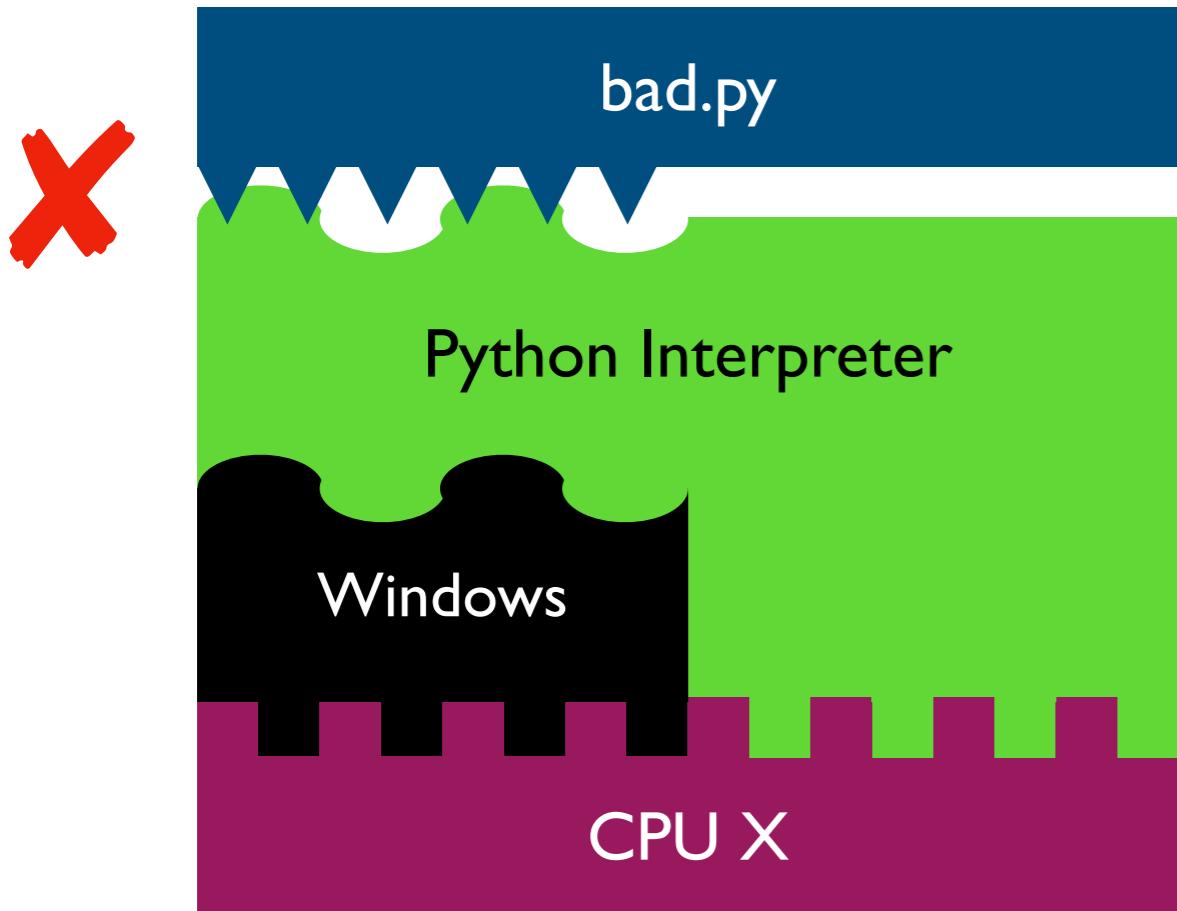
[this semester]

# OS jobs: Allocate and Abstract Resources

[like CPU, hard drive, etc]



# Harder to reproduce on different OS...

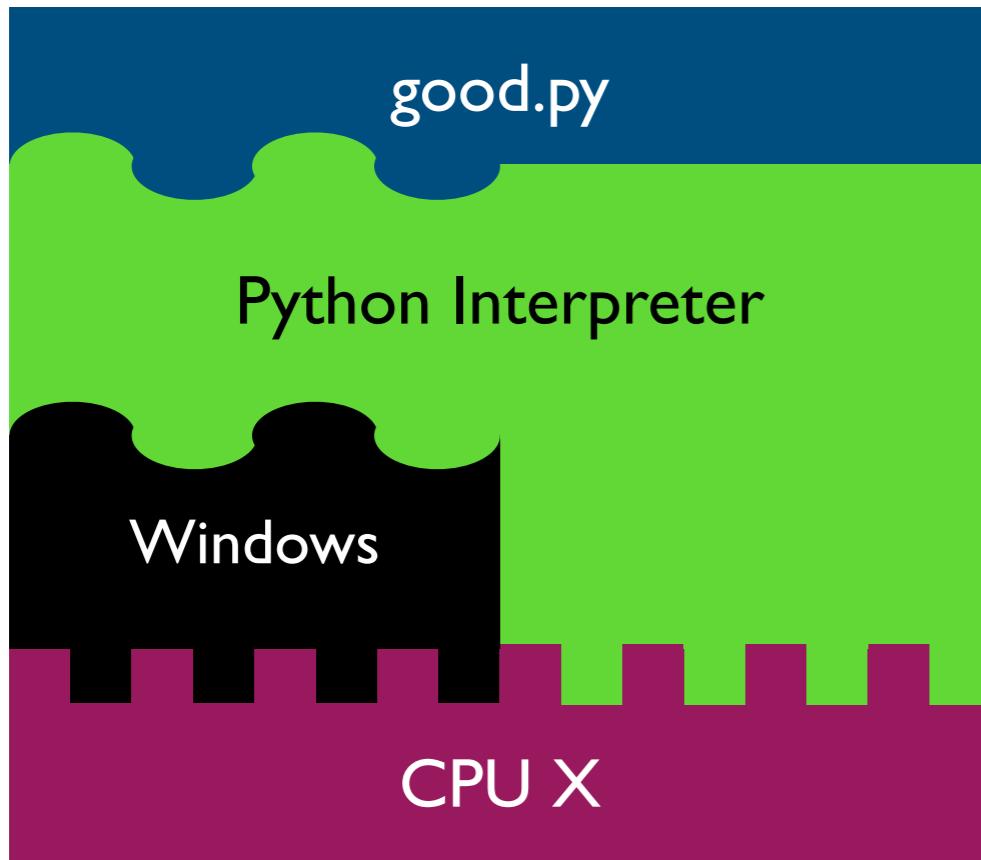


```
f = open( "/data/file.txt" )  
...  
...
```

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...

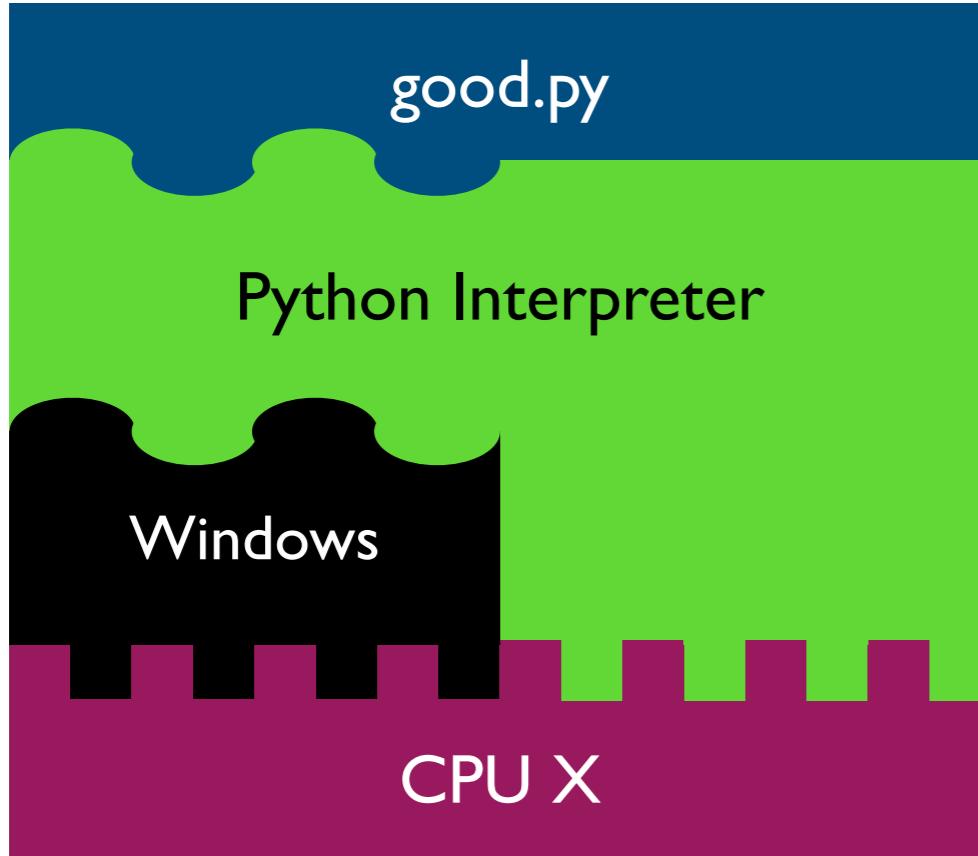


```
f = open( "c:\\data\\file.txt" )  
...
```

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!

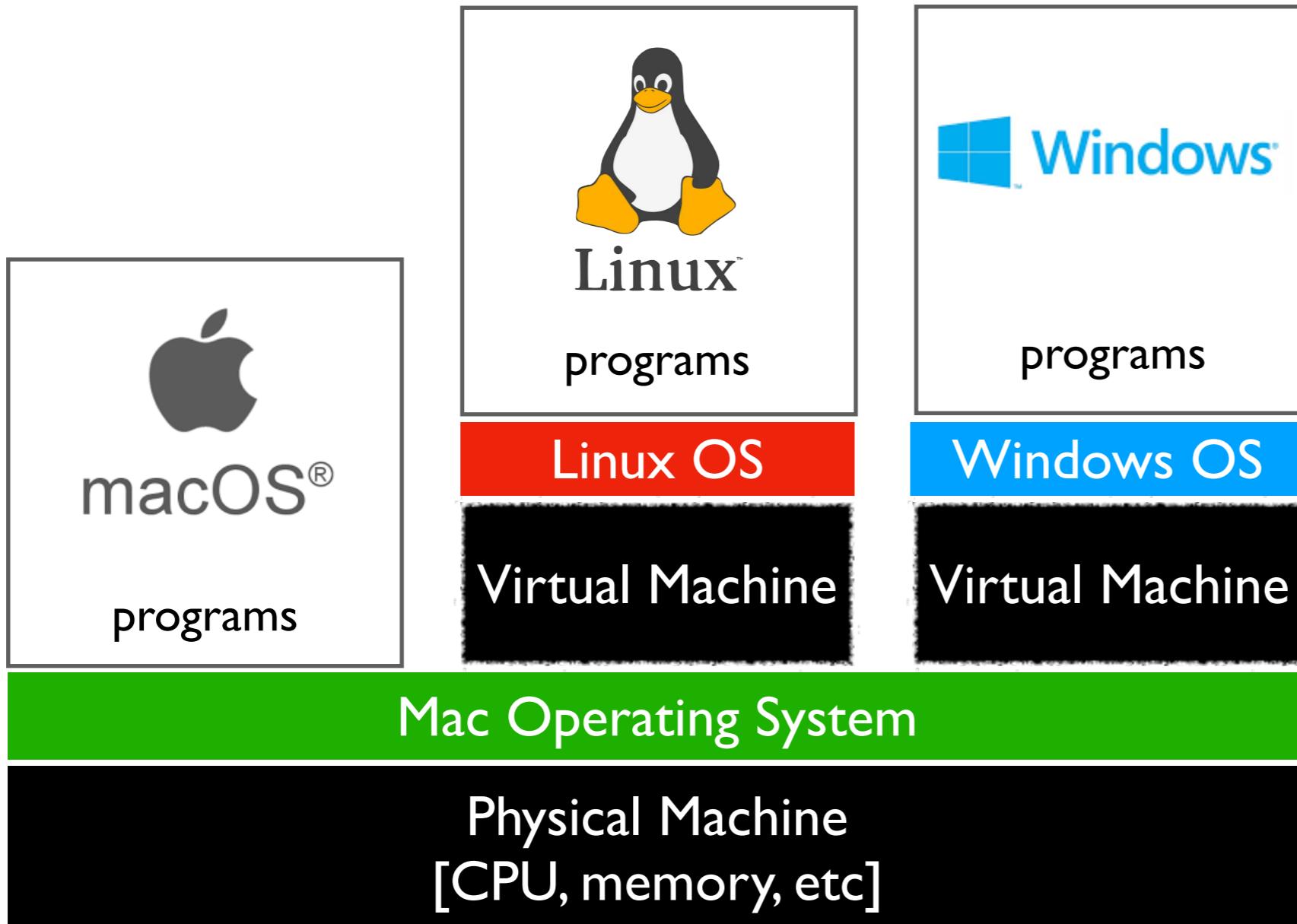
tradeoffs?

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

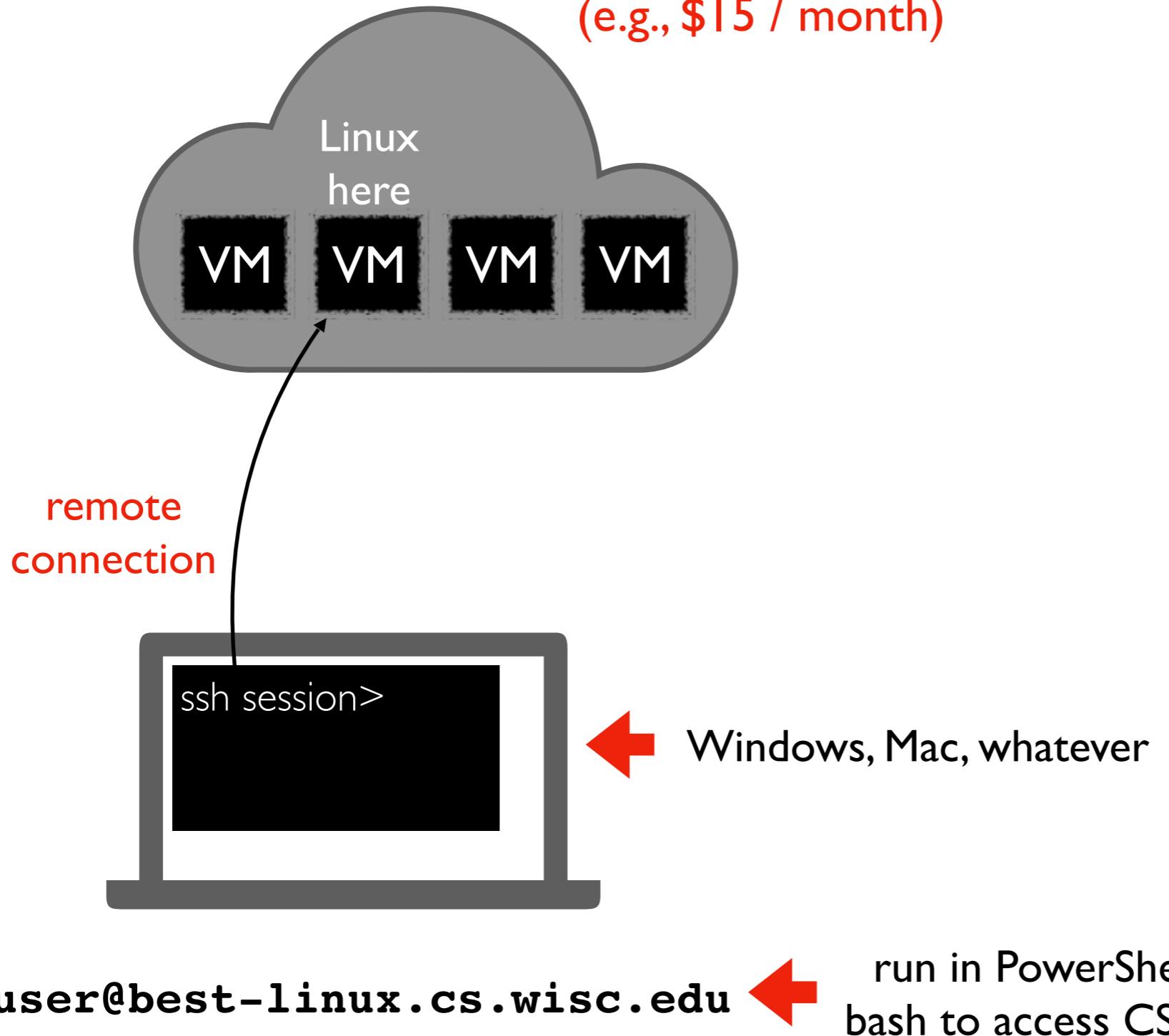


Microsoft  
Hyper-V

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)



popular cloud providers



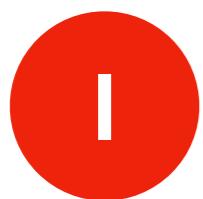
Google Cloud Platform

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