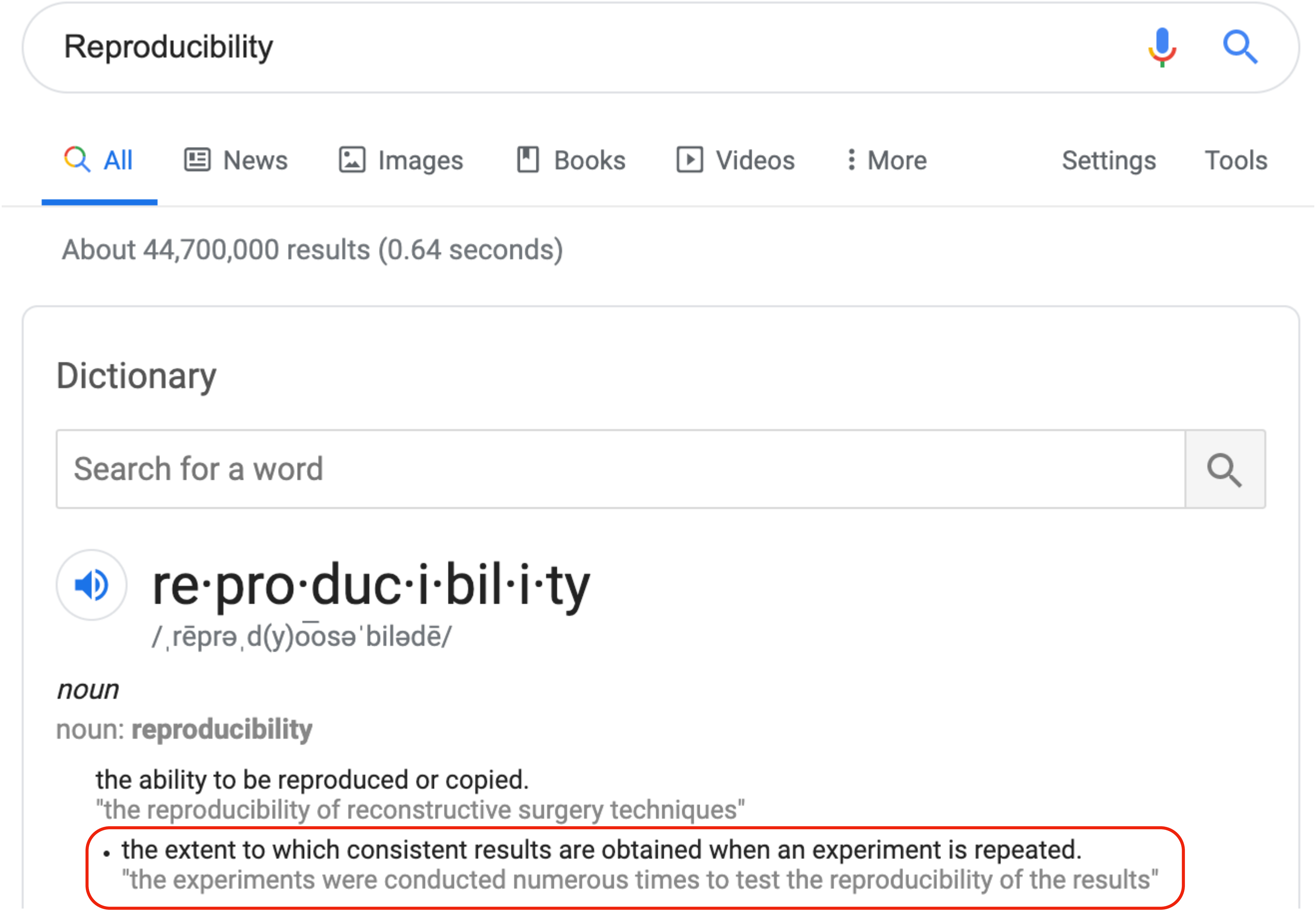
[320] Reproducibility 2

Yiyin Shen



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"

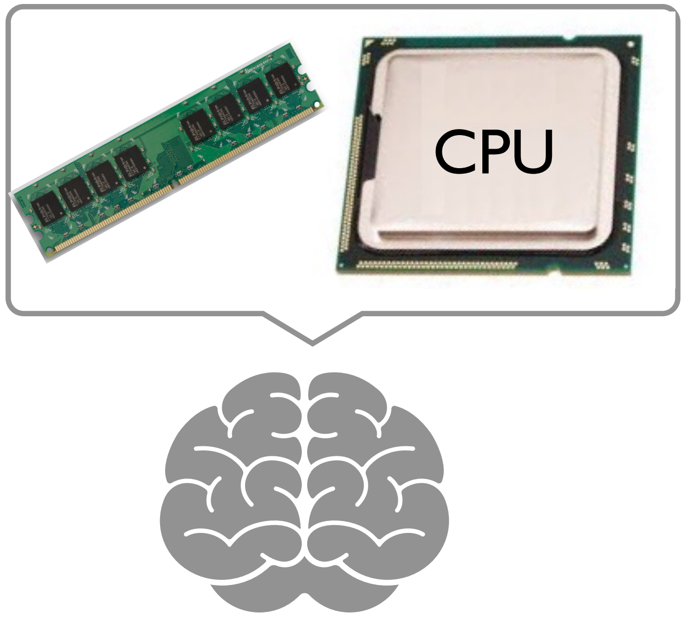
- the extent to which consistent results are obtained when an experiment is repeated.
  "the experiments were conducted numerous times to test the reproducibility of the results"

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:

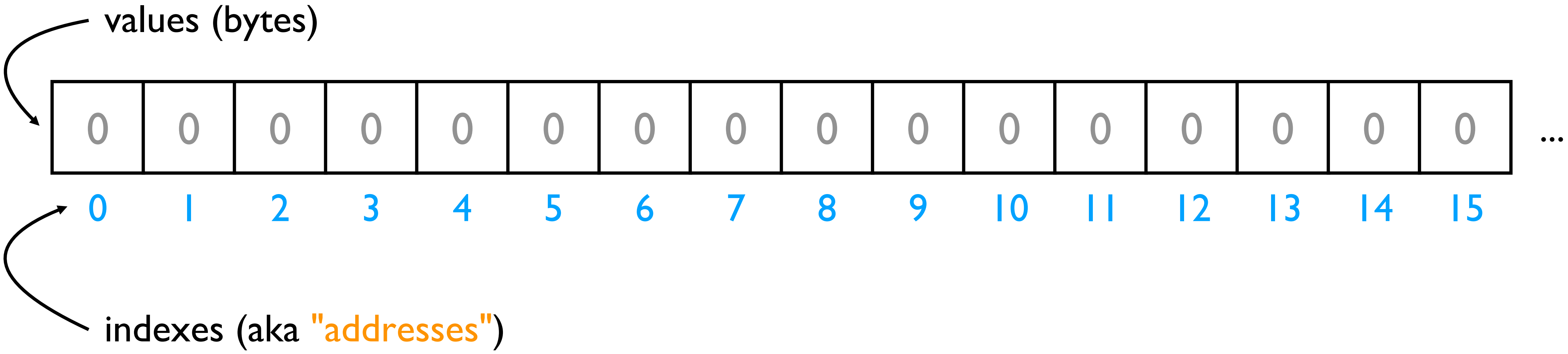
1. Hardware
2. Operating System
3. Dependencies



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")



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	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	

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

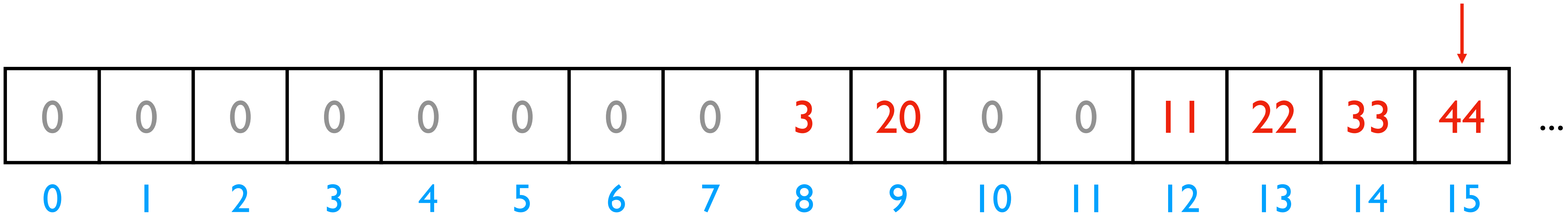
0	0	0	0	0	0	0	0	3	20	0	0	11	22	33	0	...
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	

```
# 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	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	

```
# 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	

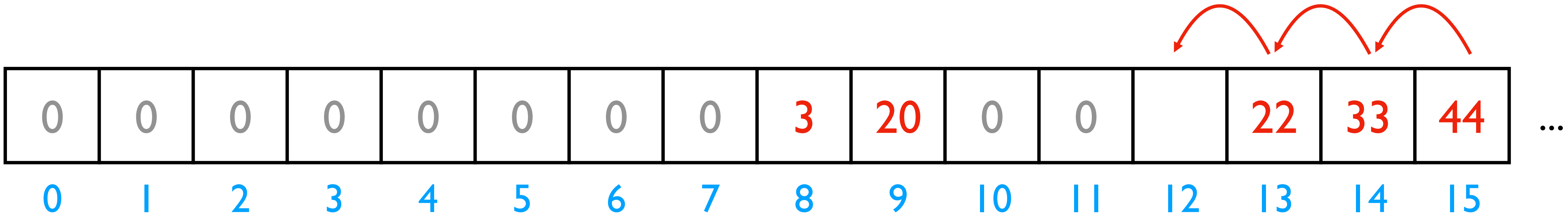
implications for performance...

```
# 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

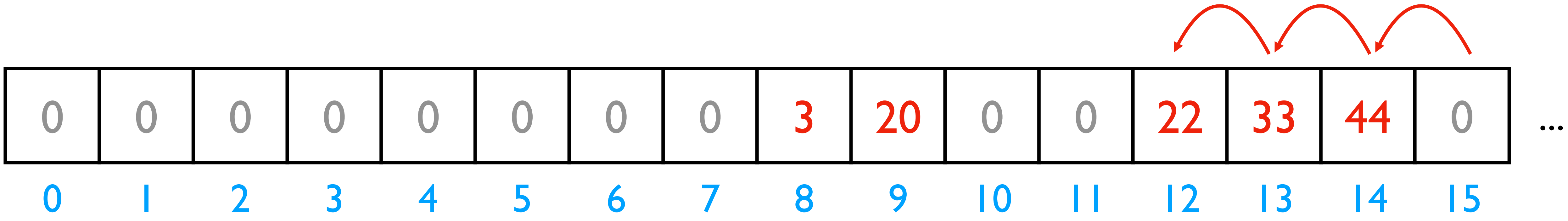
implications for performance...

```
# 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	...

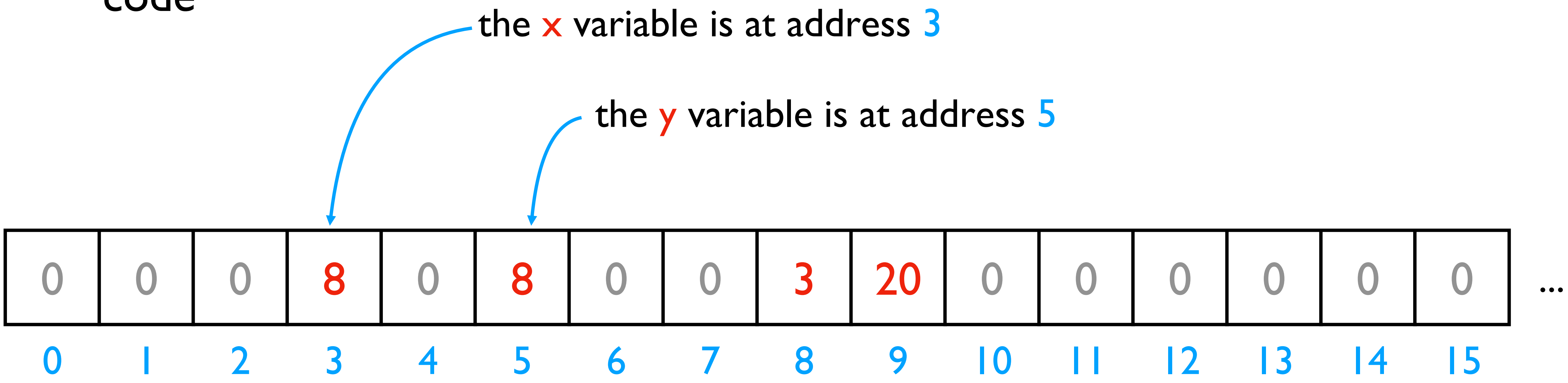
We'll think more rigorously about performance in CS 320 (big-O notation)

```
# fast
L2.append(44)

# slow
L2.pop(0)
```

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

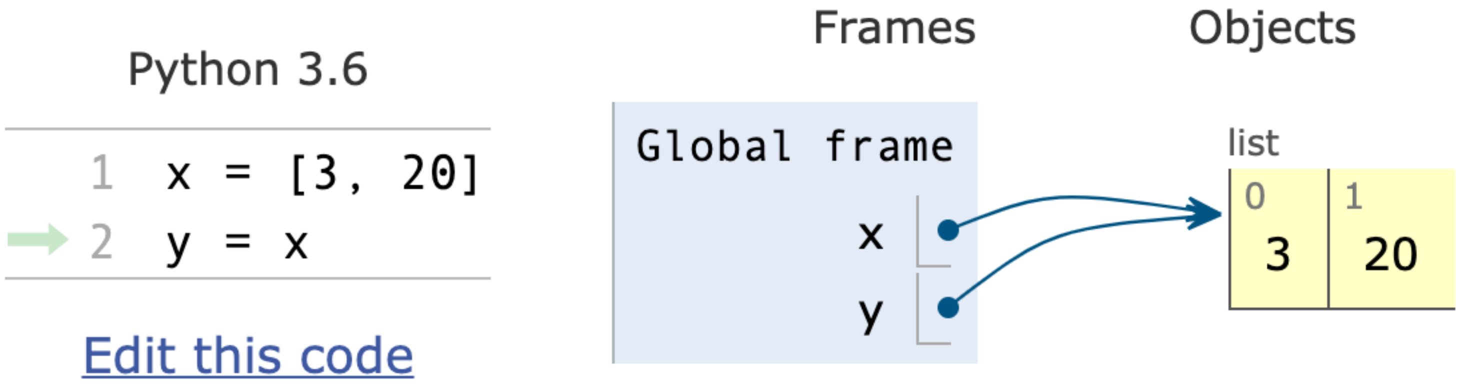
- multiple lists
- variables and other references
- strings
- code



the x variable is at address 3

the y variable is at address 5

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



Python 3.6

```
x = [3, 20]
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

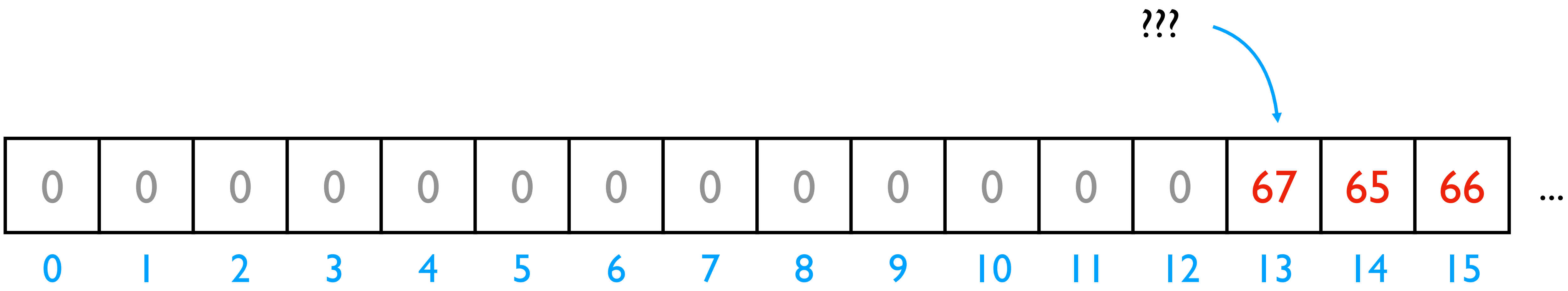
discuss: how?

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	...

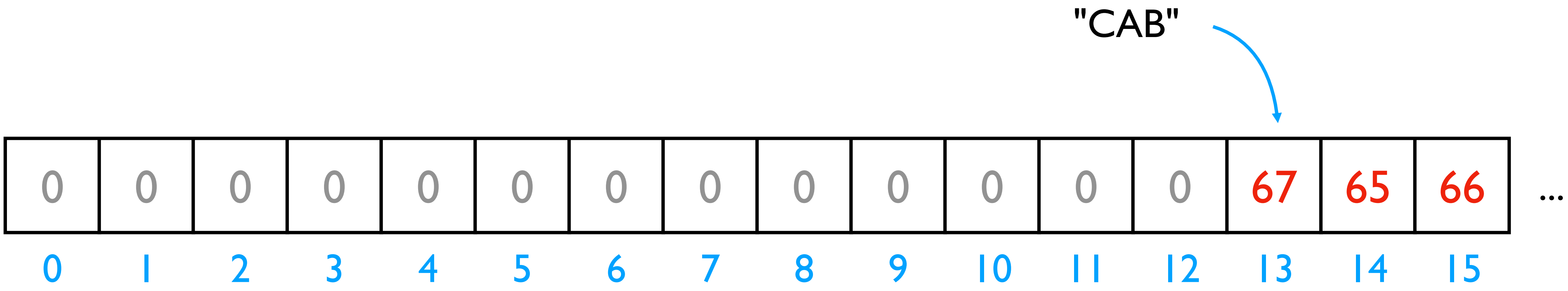
encoding:

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:

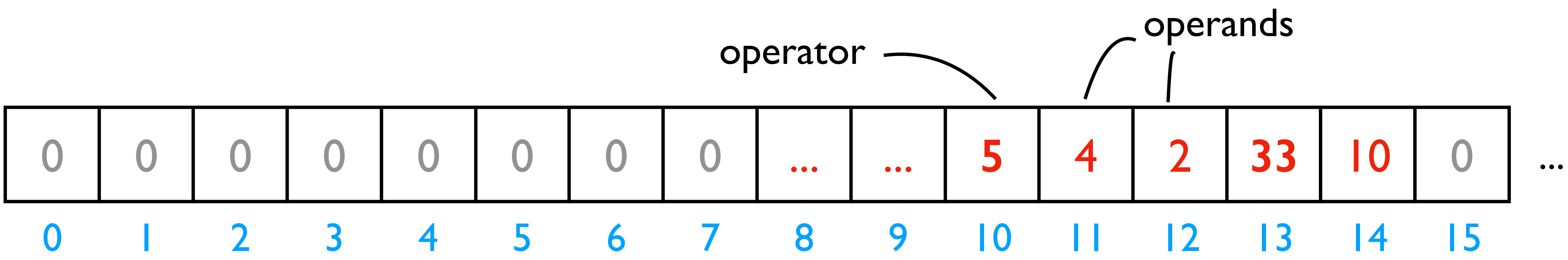
code	letter
65	A
66	B
67	C
68	D
...	...

```
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

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



operator

operands

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	4	2	33	10	0	...

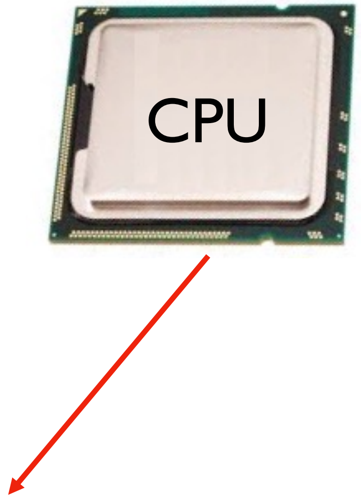
Instruction Set

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	4	2	33	10	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

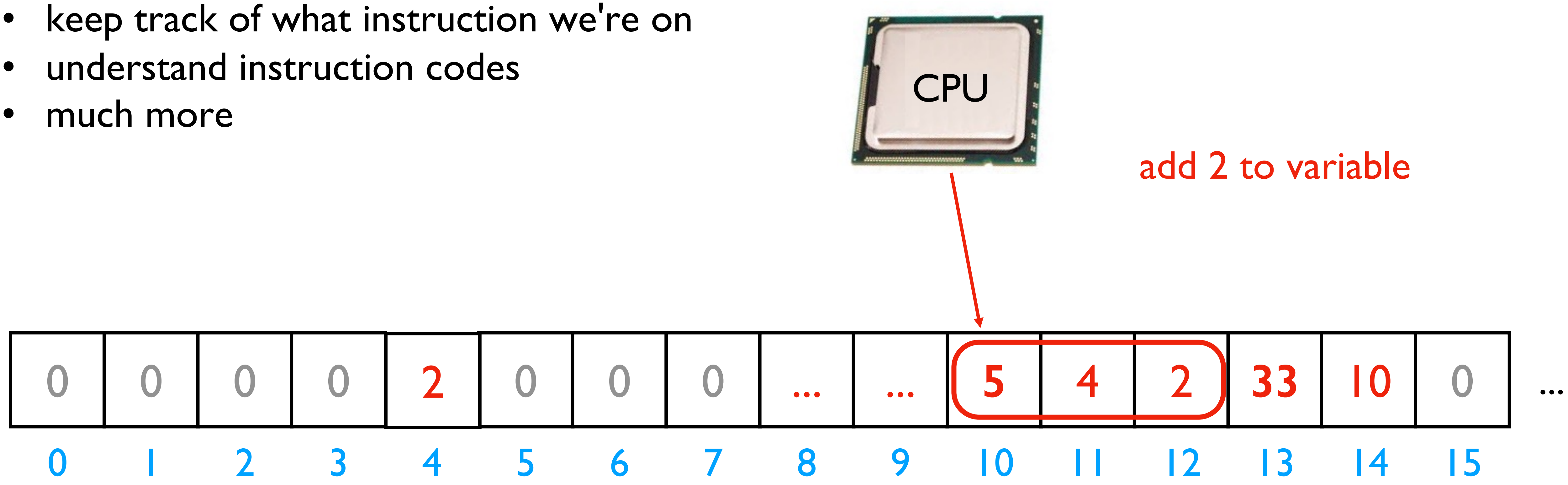
Instruction Set

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	2	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	

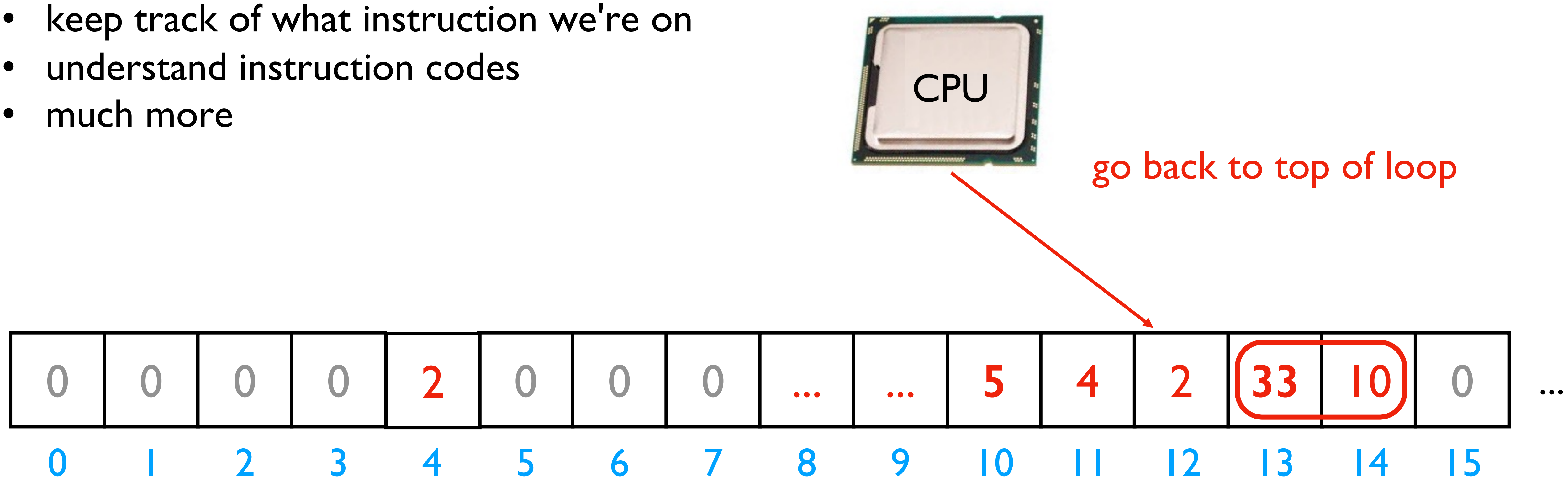
Instruction Set

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	2	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	

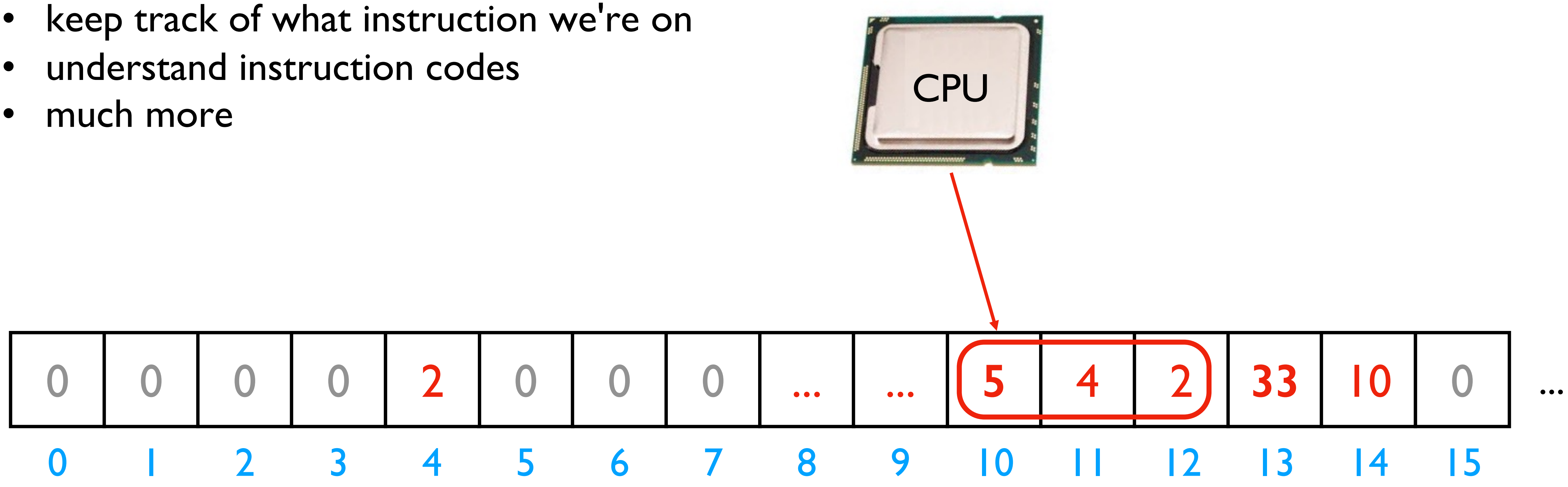
Instruction Set

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	2	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

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	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
...	...

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	

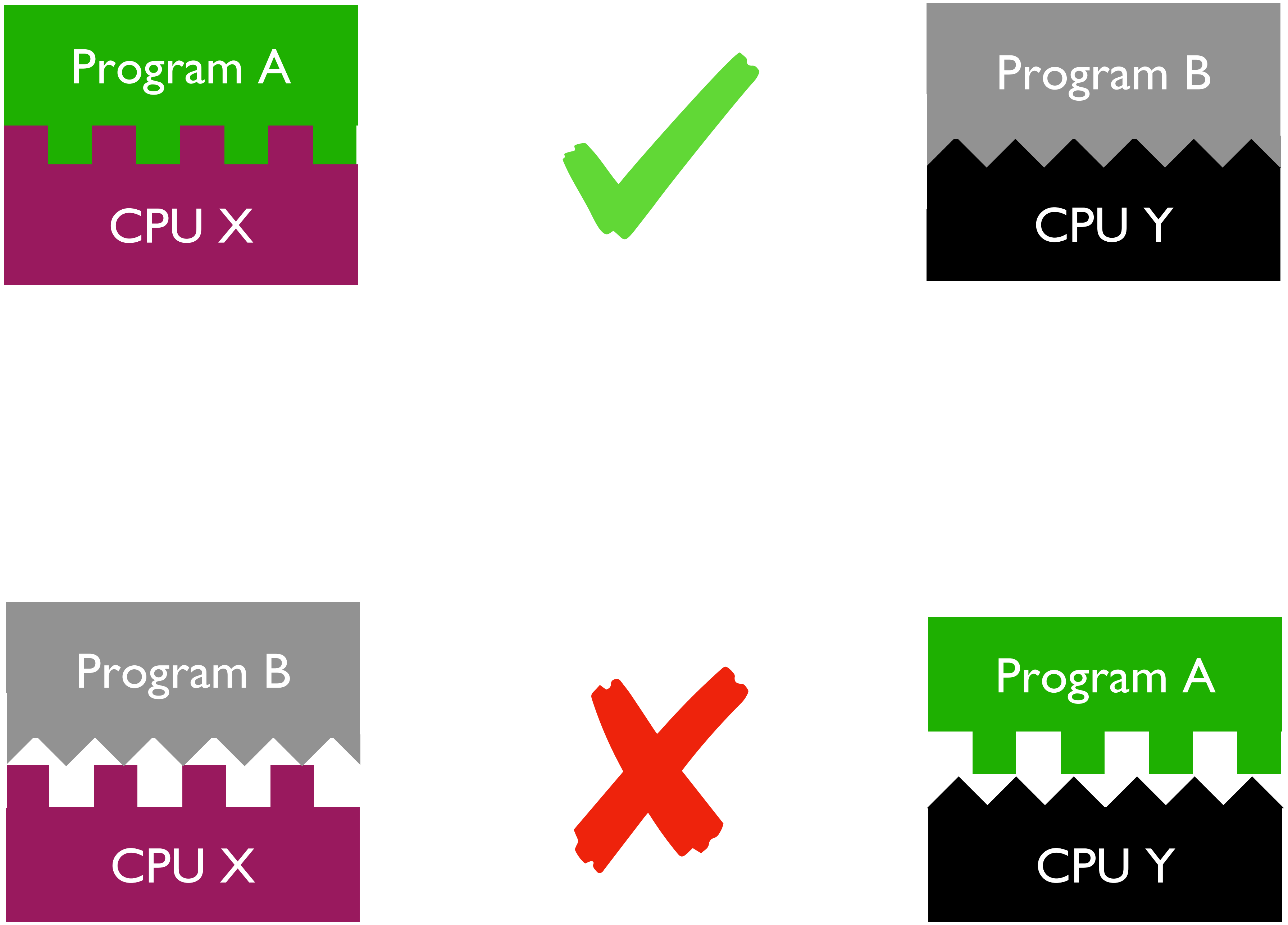
Instruction Set for CPU X

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

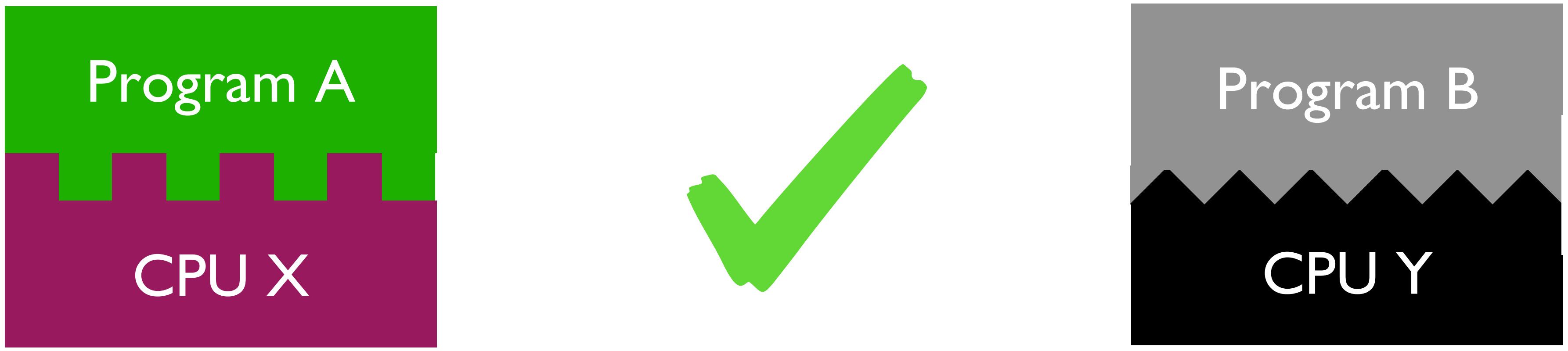
Instruction Set for CPU Y

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

A Program and CPU need to "fit"

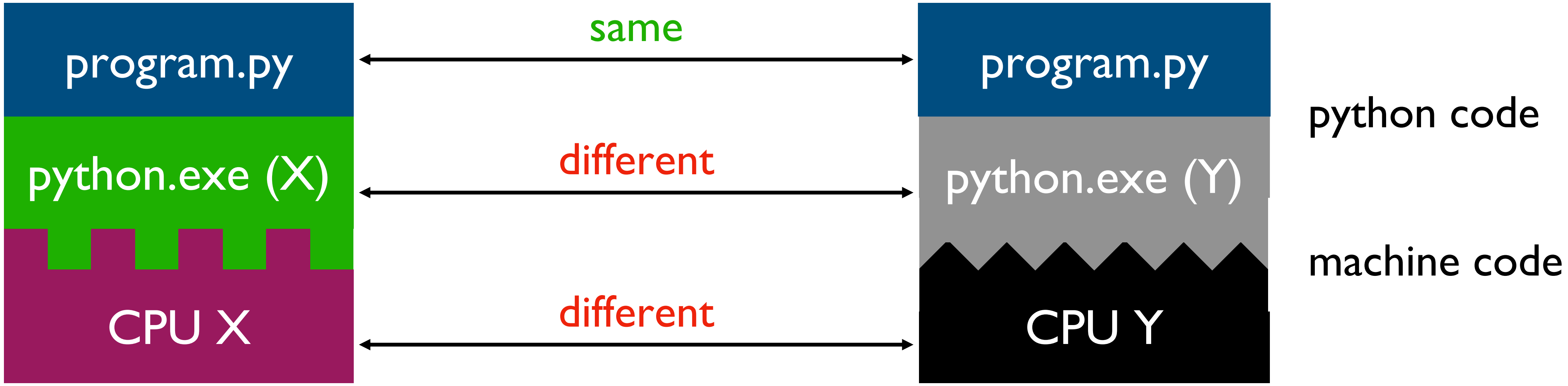


A Program and CPU need to "fit"



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

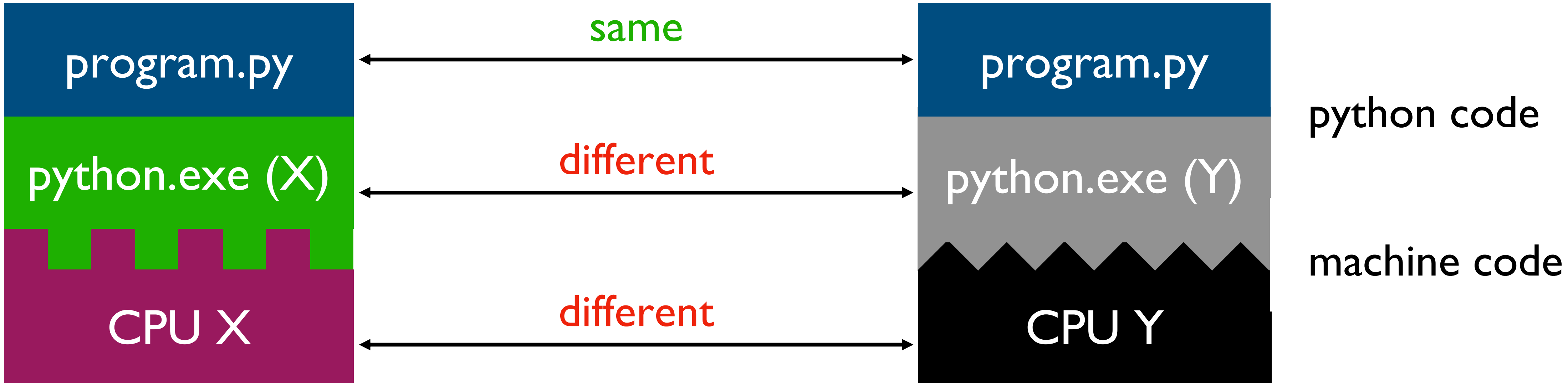
Interpreters



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



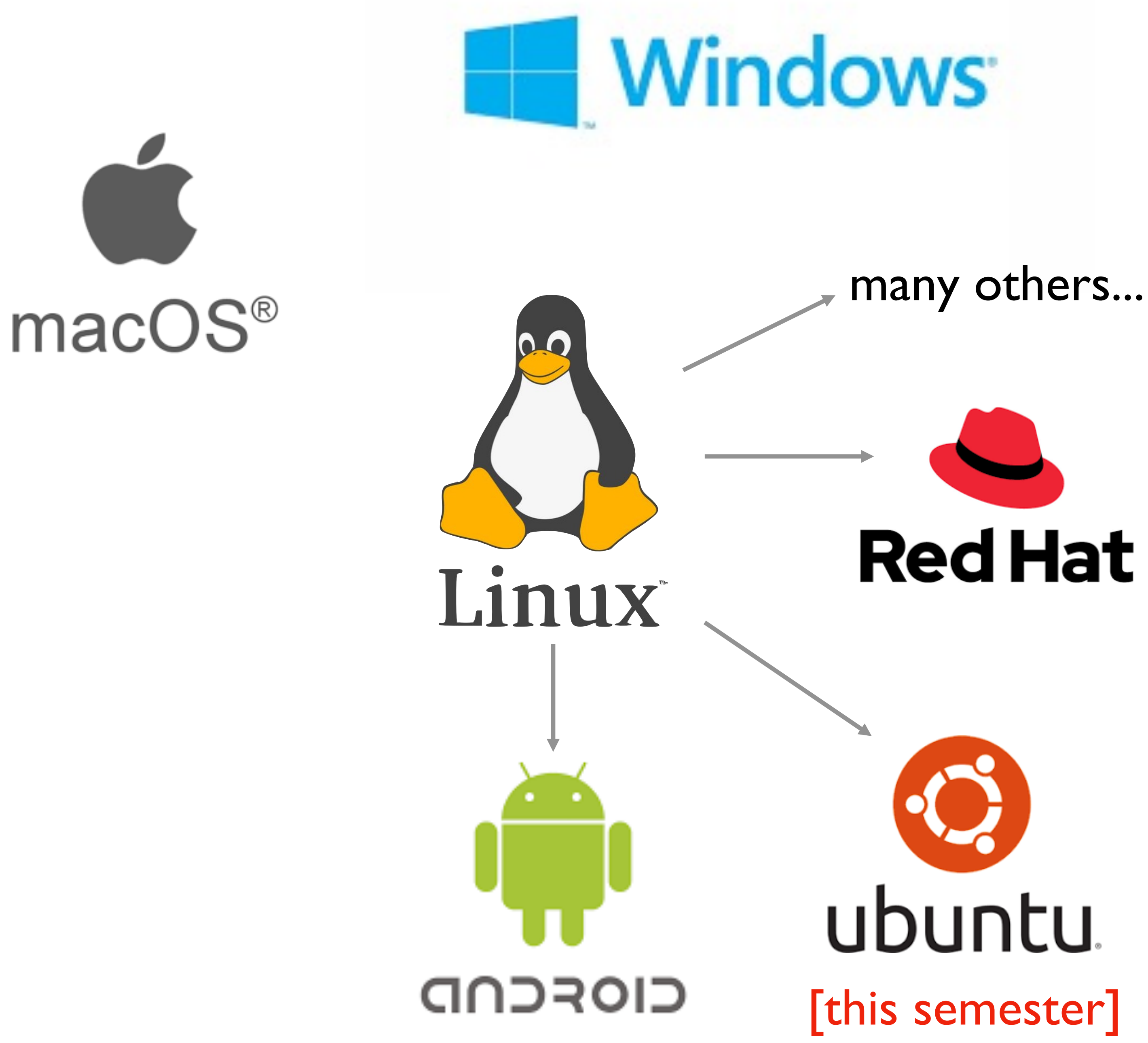
Interpreters (such as python.exe) make it easier to run the same code on different machines

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



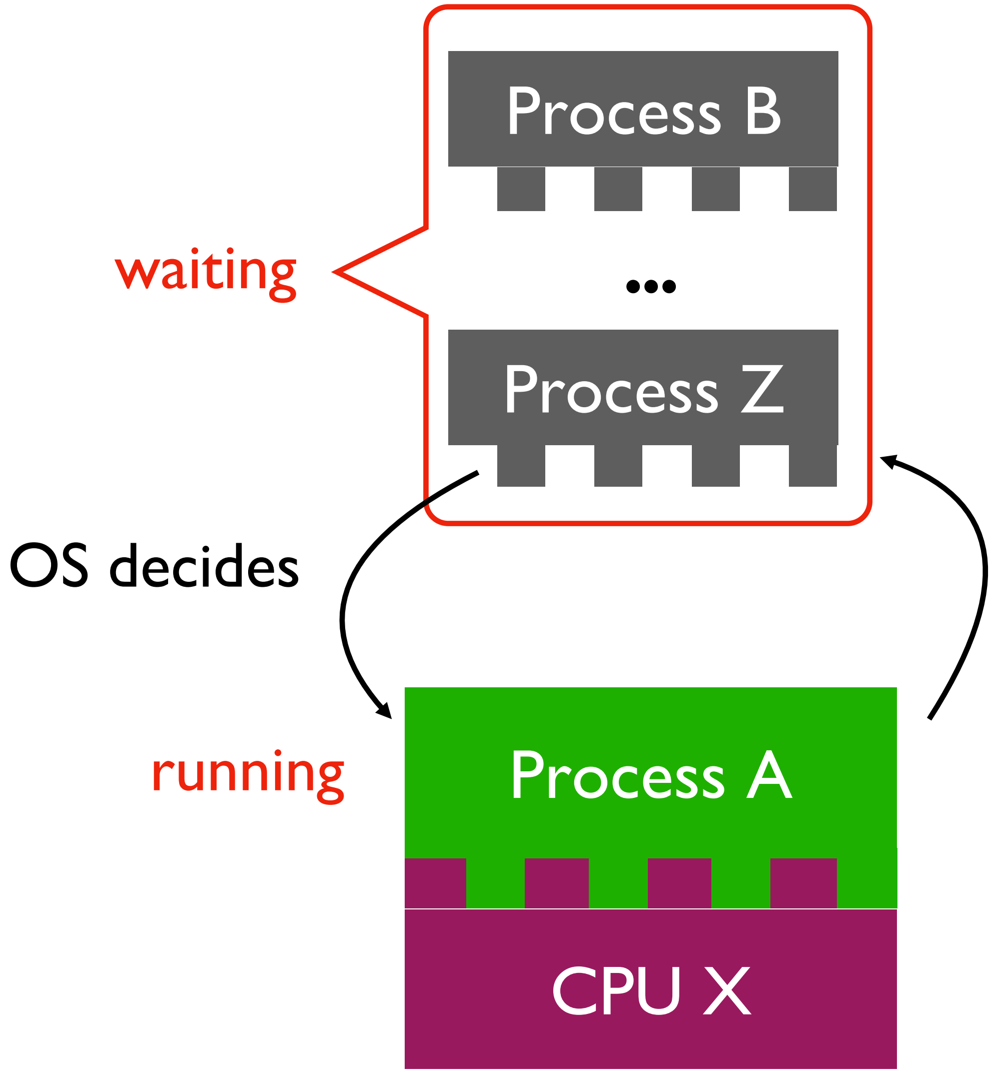
many others...

[this semester]

OS jobs: Allocate and Abstract Resources

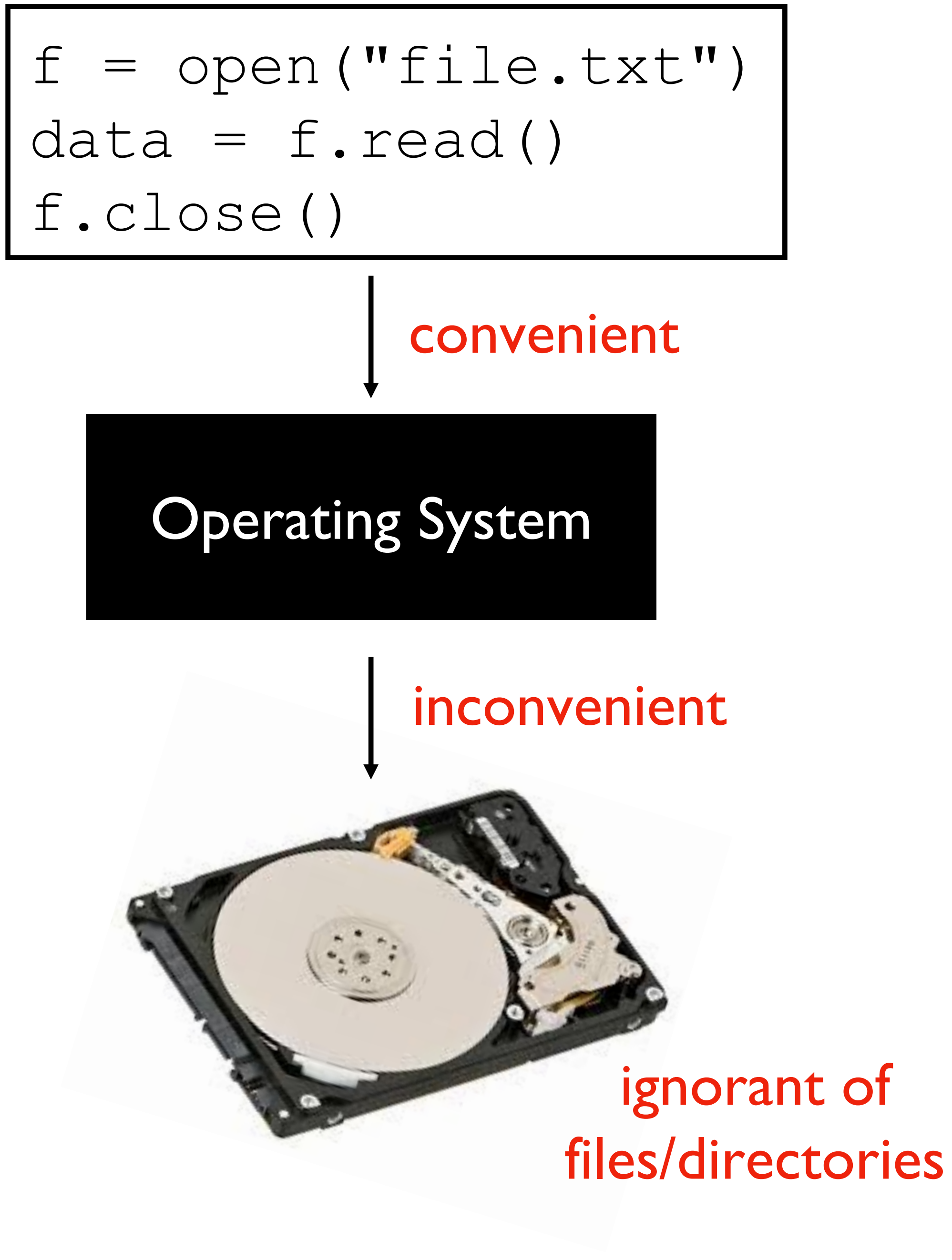
[like CPU, hard drive, etc]

1 Allocation



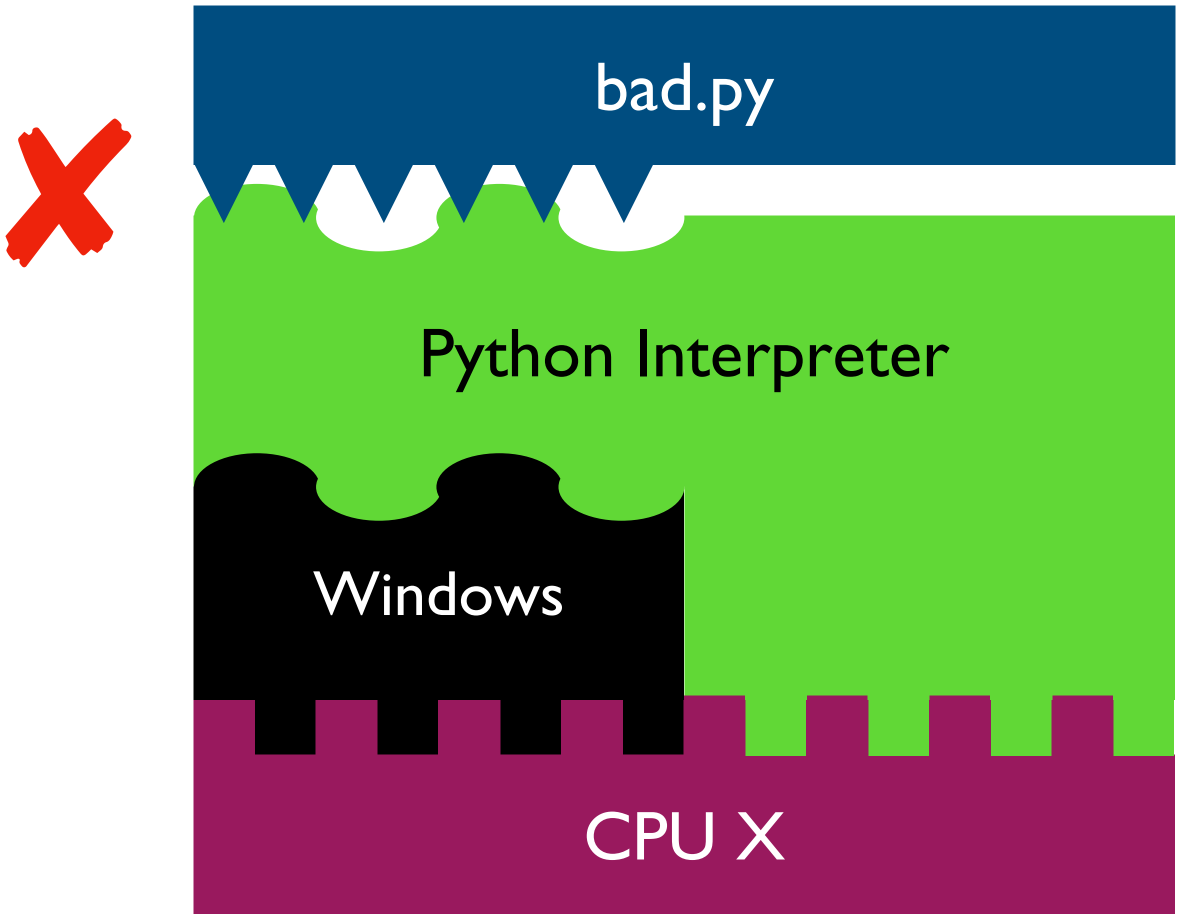
only one process can run on CPU at a time
(or a few things if the CPU has multiple "cores")

2 Abstraction



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

Harder to reproduce on different OS...

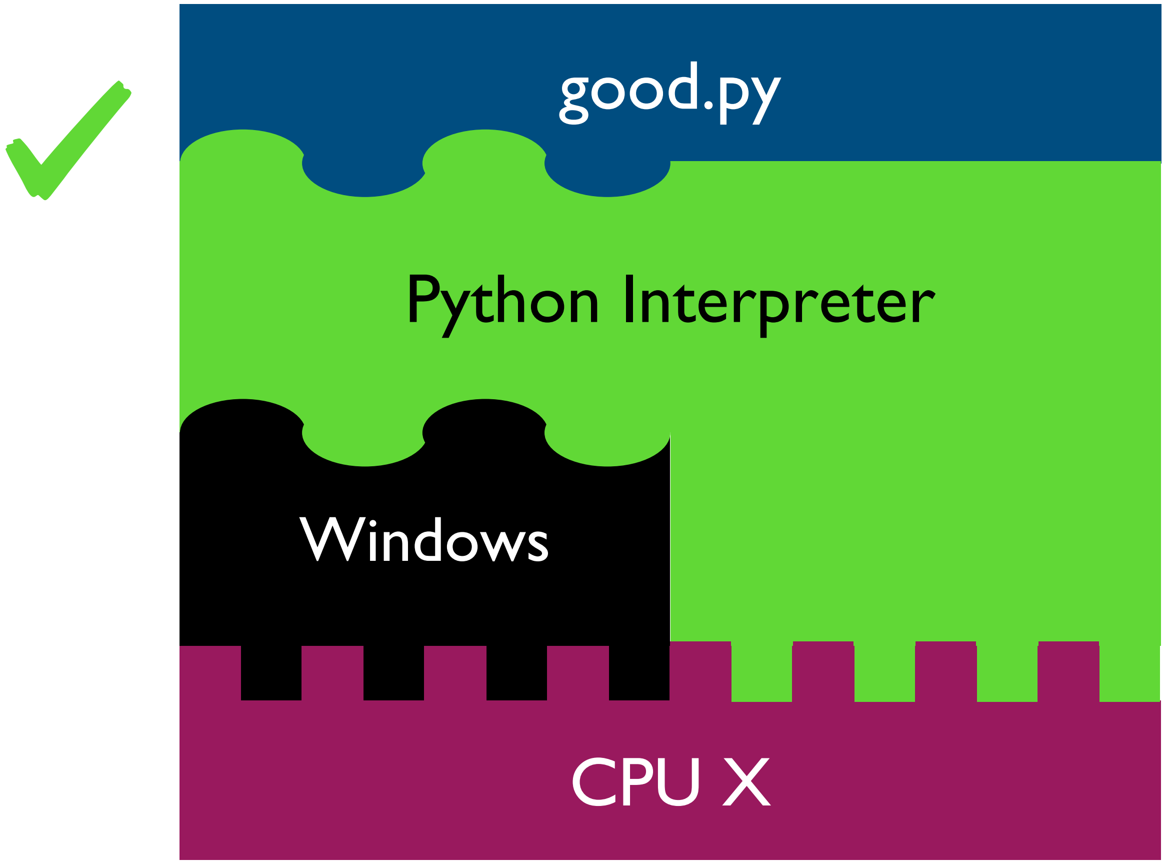


```
f = open("/data/file.txt")
...
```

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...

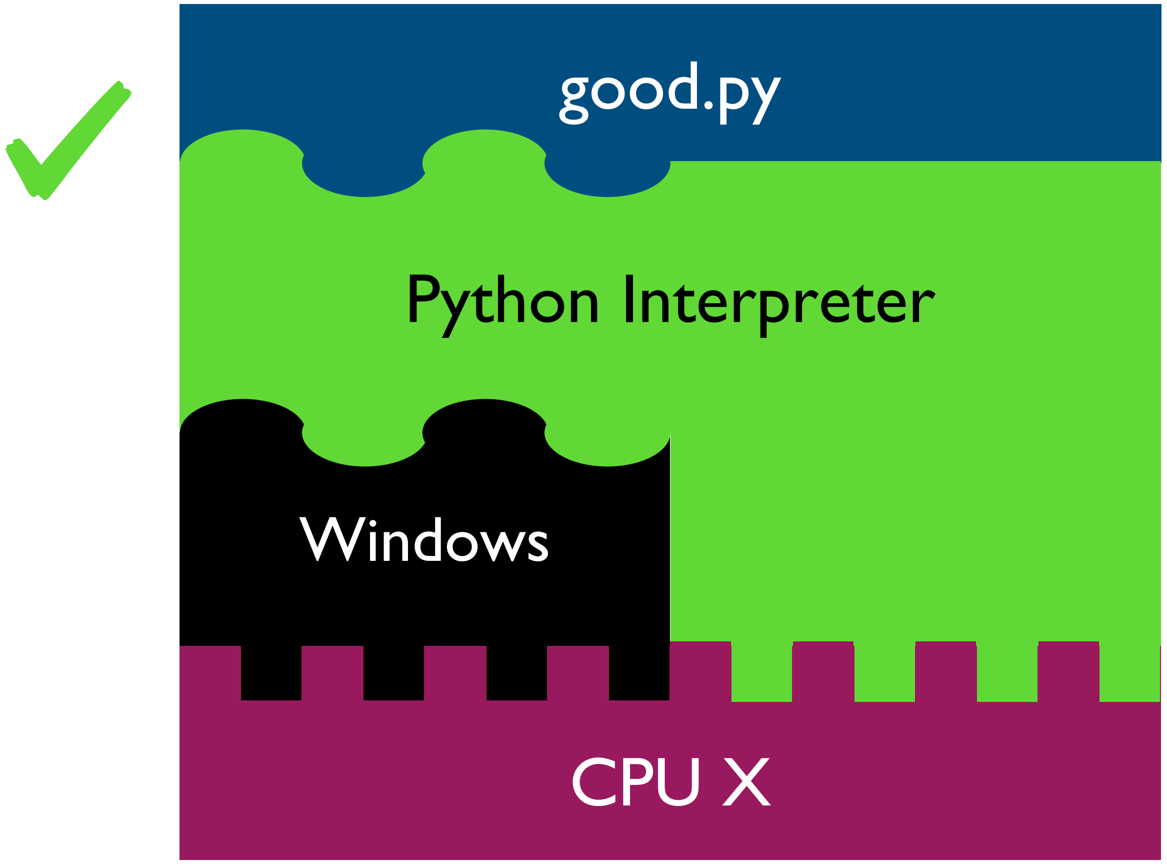


```
f = open("c:\data\file.txt")
...
```

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:

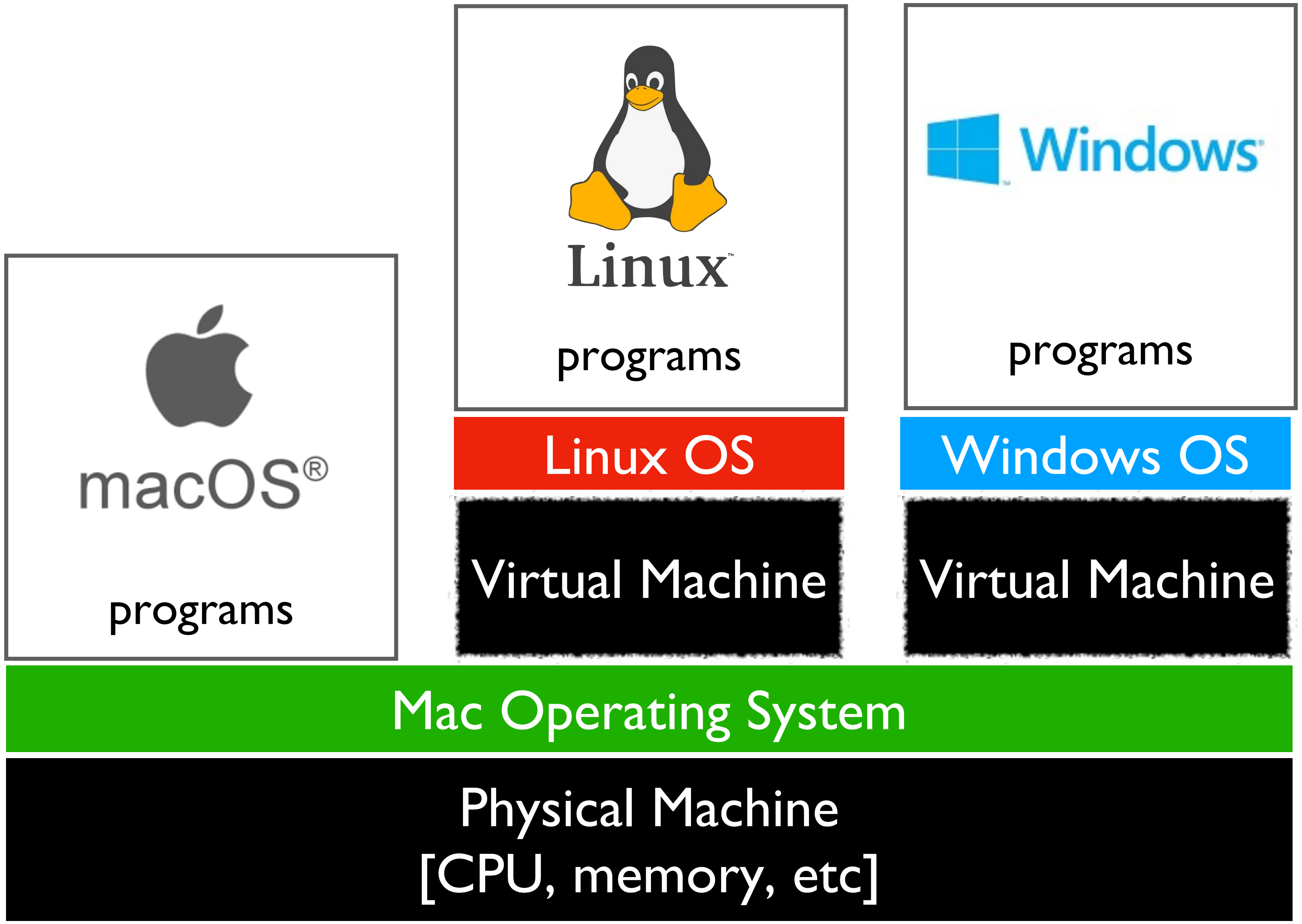
tell anybody reproducing your results to use the same OS!

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)

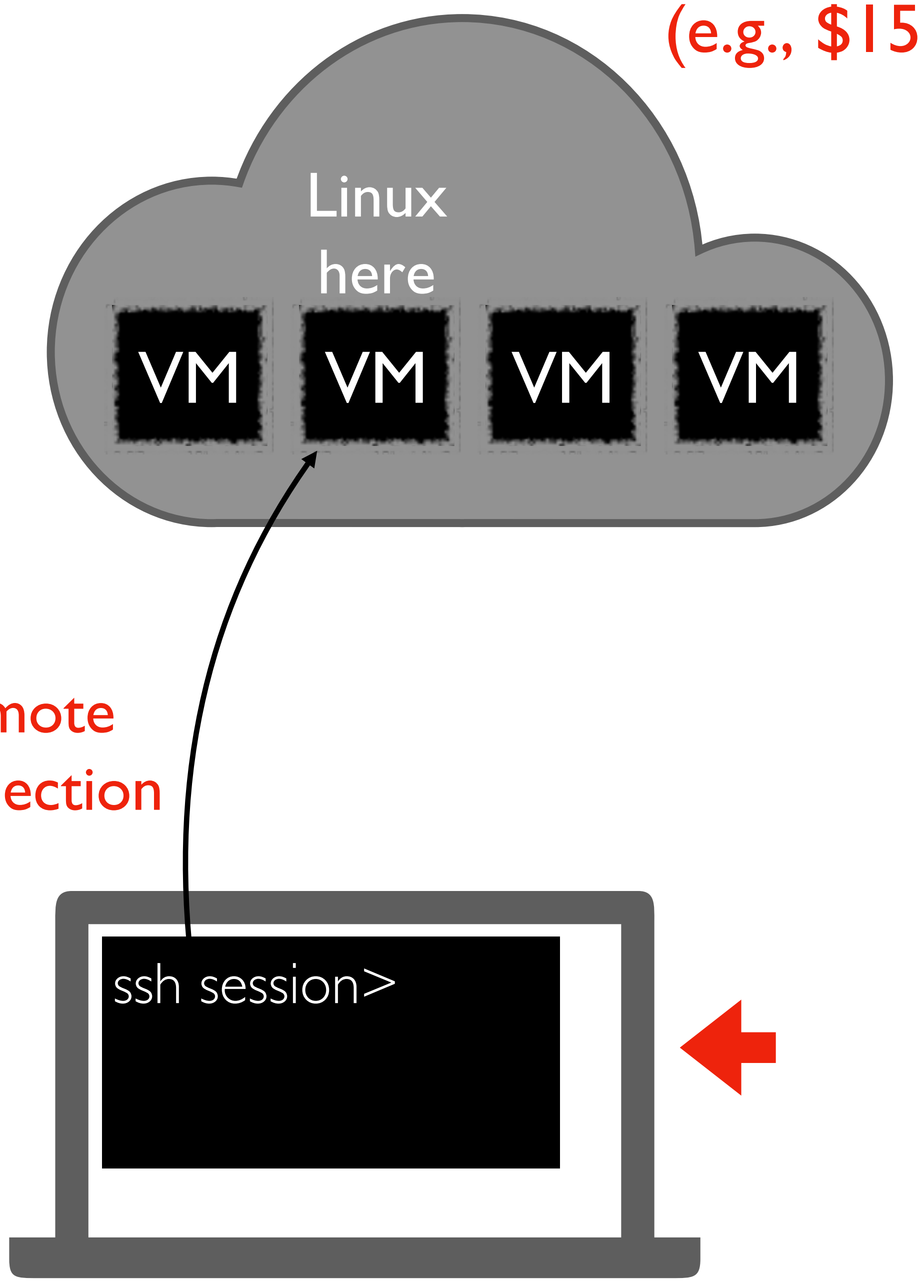


popular virtual machine software

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 ← today: 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

[320] Version Control (git)

Yiyin Shen

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 ← today: versioning

Dependency Versions

this program "depends" on pandas

program.py

```
import os, sys, json
import pandas

import pandas

print("Pandas Version:", pandas.__version__)

# code that uses pandas
```

you can check a module version

behavior depends on which release was installed

```
pip install pandas
```

or

```
pip install pandas==0.25.1
```

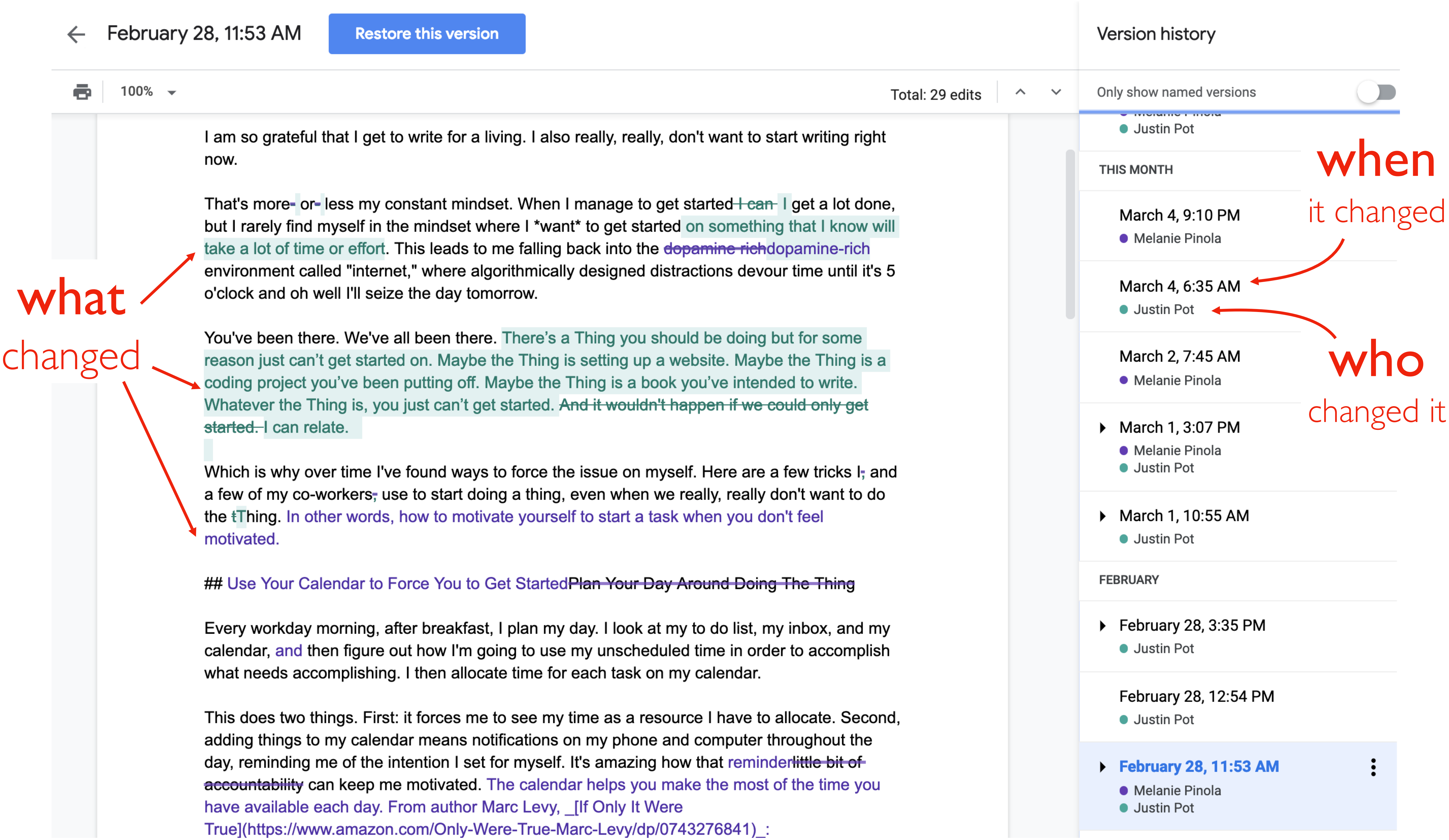
or

```
pip install pandas==0.24.0
```

or...

Versioning: motivation and basic concepts

Many tools auto-track history (e.g., Google Docs)



I am so grateful that I get to write for a living. I also really, really, don't want to start writing right now.

That's more~~-~~ or~~-~~ less my constant mindset. When I manage to get started~~ I can~~ I get a lot done, but I rarely find myself in the mindset where I *want* to get started on something that I know will take a lot of time or effort. This leads to me falling back into the ~~dopamine-rich~~dopamine-rich environment called "internet," where algorithmically designed distractions devour time until it's 5 o'clock and oh well I'll seize the day tomorrow.

You've been there. We've all been there. There's a Thing you should be doing but for some reason just can't get started on. Maybe the Thing is setting up a website. Maybe the Thing is a coding project you've been putting off. Maybe the Thing is a book you've intended to write. Whatever the Thing is, you just can't get started. ~~And it wouldn't happen if we could only get started.~~ I can relate.

Which is why over time I've found ways to force the issue on myself. Here are a few tricks I~~,~~ and a few of my co-workers~~,~~ use to start doing a thing, even when we really, really don't want to do the ~~t~~Thing. In other words, how to motivate yourself to start a task when you don't feel motivated.

Use Your Calendar to Force You to Get Started~~Plan Your Day Around Doing The Thing~~

Every workday morning, after breakfast, I plan my day. I look at my to do list, my inbox, and my calendar, and then figure out how I'm going to use my unscheduled time in order to accomplish what needs accomplishing. I then allocate time for each task on my calendar.

This does two things. First: it forces me to see my time as a resource I have to allocate. Second, adding things to my calendar means notifications on my phone and computer throughout the day, reminding me of the intention I set for myself. It's amazing how that reminder~~little bit of accountability~~ can keep me motivated. The calendar helps you make the most of the time you have available each day. From author Marc Levy, _[If Only It Were True](https://www.amazon.com/Only-Were-True-Marc-Levy/dp/0743276841)_:

Melanie Pinola
Justin Pot

THIS MONTH

March 4, 9:10 PM
Melanie Pinola

March 4, 6:35 AM
Justin Pot

March 2, 7:45 AM
Melanie Pinola

March 1, 3:07 PM
Melanie Pinola
Justin Pot

March 1, 10:55 AM
Justin Pot

FEBRUARY

February 28, 3:35 PM
Justin Pot

February 28, 12:54 PM
Justin Pot

February 28, 11:53 AM
Melanie Pinola
Justin Pot

https://zapier.com/apps/google-docs/tutorials/google-docs-revision-history

Version Control Systems (VCS)

Useful for many kinds of projects

- code, papers, websites, etc
- manages all files for same project (maybe thousands) in a repository

Explicit snapshots/checkpoints, called *commits*

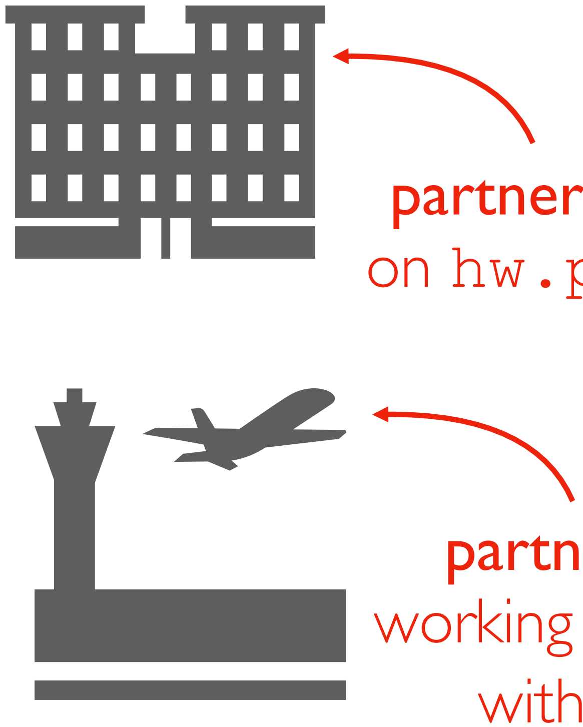
- users **manually** run commands to preserve good versions

Explicit commit messages

- who, what, when, **why**

Work can *branch* out and be *merged* back

- people can work offline
- can get feedback before merging
- humans need to resolve *conflicts* when versions being merged are too different

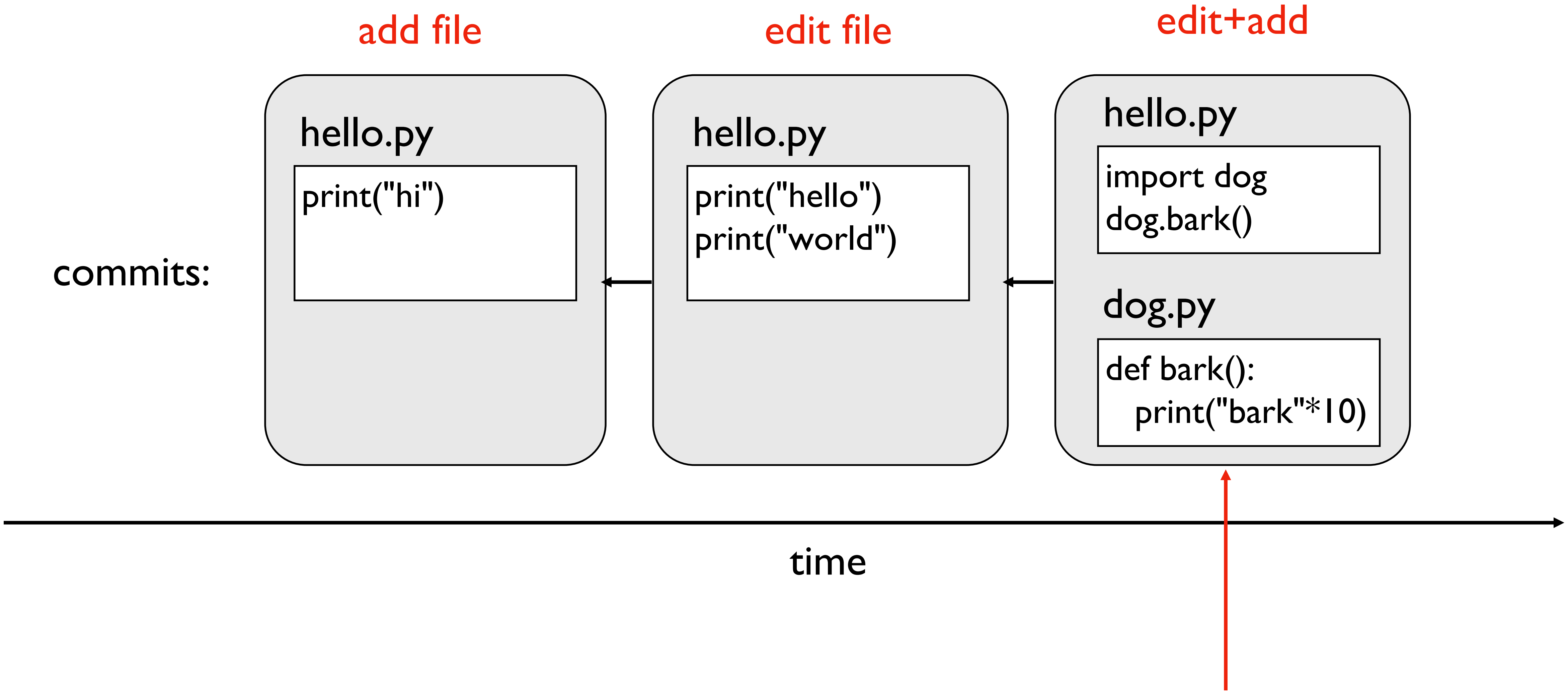


partner A working on `hw.py` at school

partner B also working on `hw.py`, without wifi

what happens when the plane lands?

Example



commits:

add file

hello.py

```
print("hi")
```

edit file

hello.py

```
print("hello")
print("world")
```

edit+add

hello.py

```
import dog
dog.bark()
```

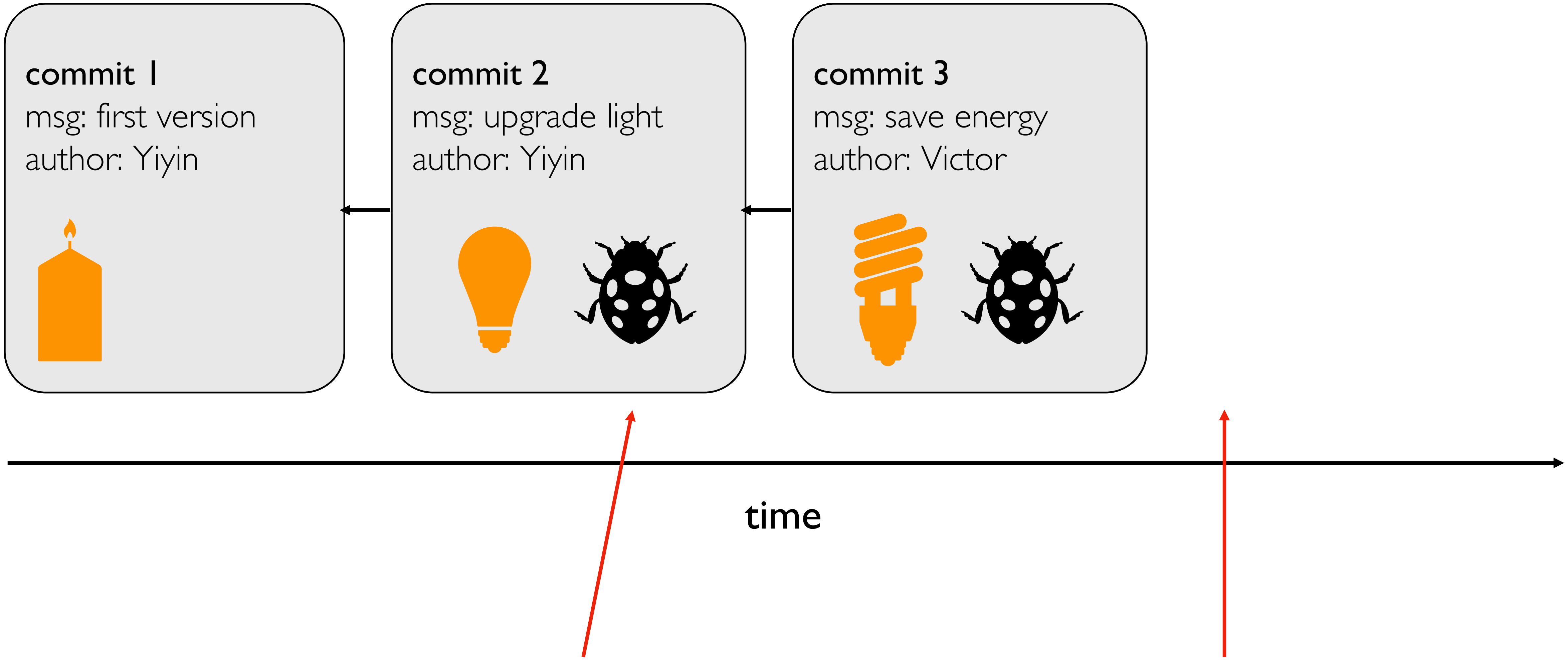
dog.py

```
def bark():
   print("bark"*10)
```

time

at any point in time,
you just see one version
of the files on your computer

Use case 1: troubleshooting discovered bug



commit 1
msg: first version
author: Yiyin

commit 2
msg: upgrade light
author: Yiyin

commit 3
msg: save energy
author: Victor

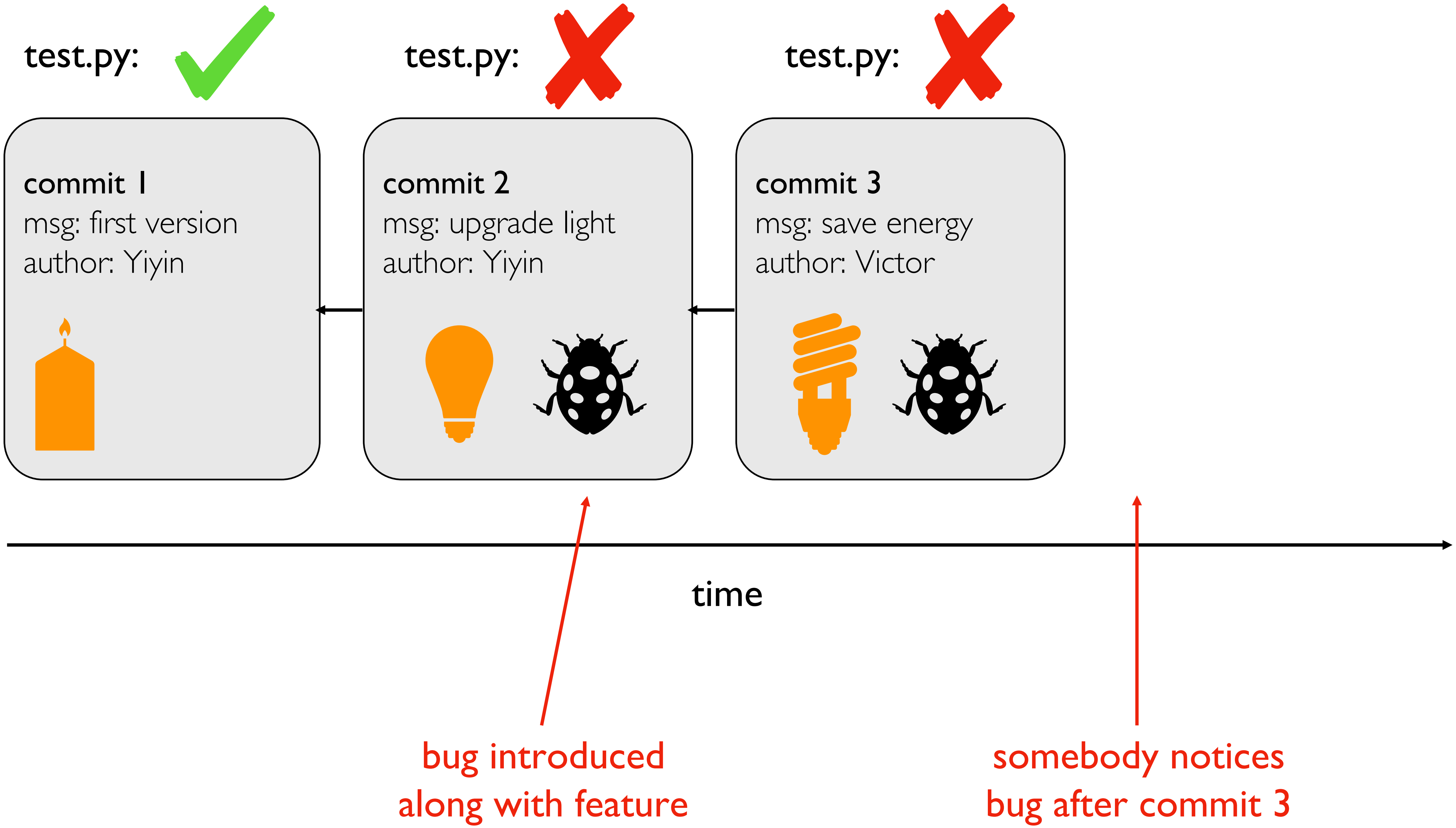
time

bug introduced
along with feature

somebody notices
bug after commit 3

who will get blamed?

Use case 1: troubleshooting discovered bug



test.py: ✓

commit 1
msg: first version
author: Yiyin

test.py: ✗

commit 2
msg: upgrade light
author: Yiyin

test.py: ✗

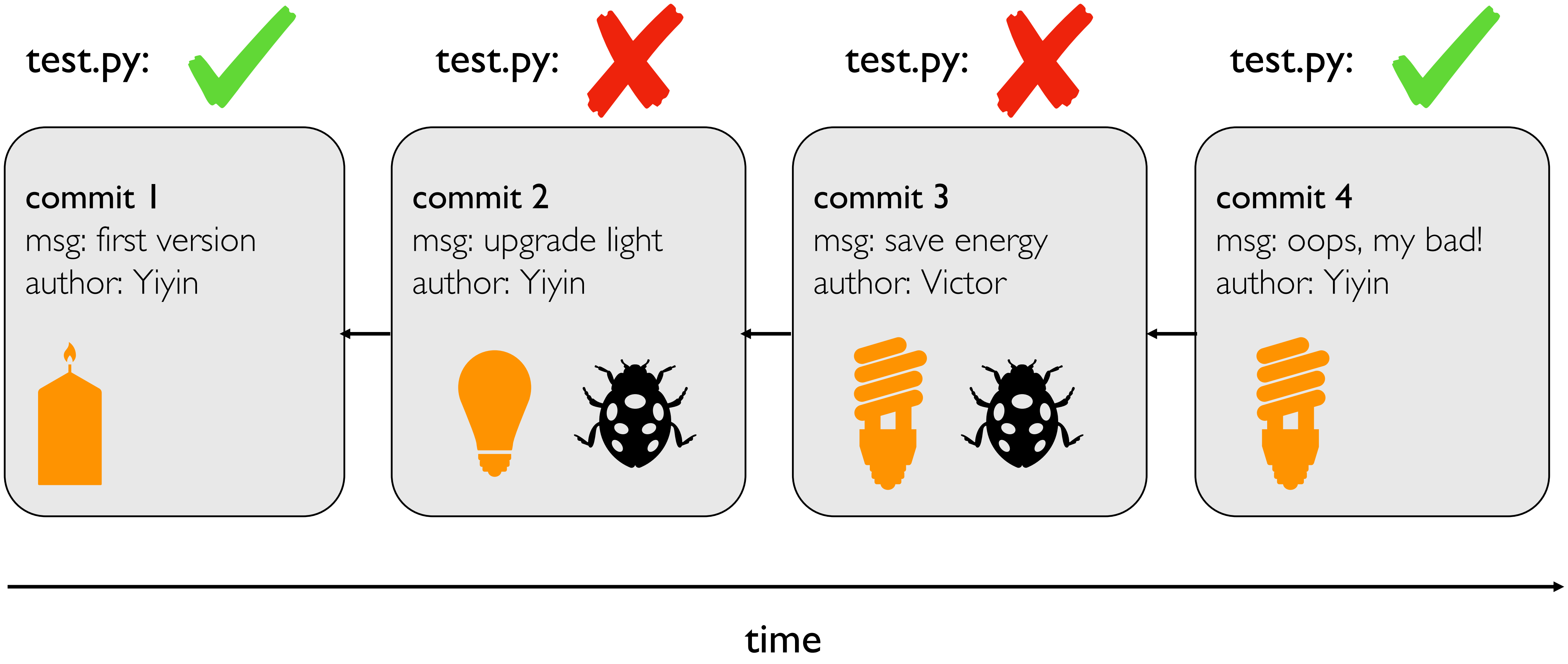
commit 3
msg: save energy
author: Victor

time

bug introduced
along with feature

somebody notices
bug after commit 3

Use case 1: troubleshooting discovered bug



test.py: ✓

commit 1
msg: first version
author: Yiyin

test.py: ✗

commit 2
msg: upgrade light
author: Yiyin

test.py: ✗

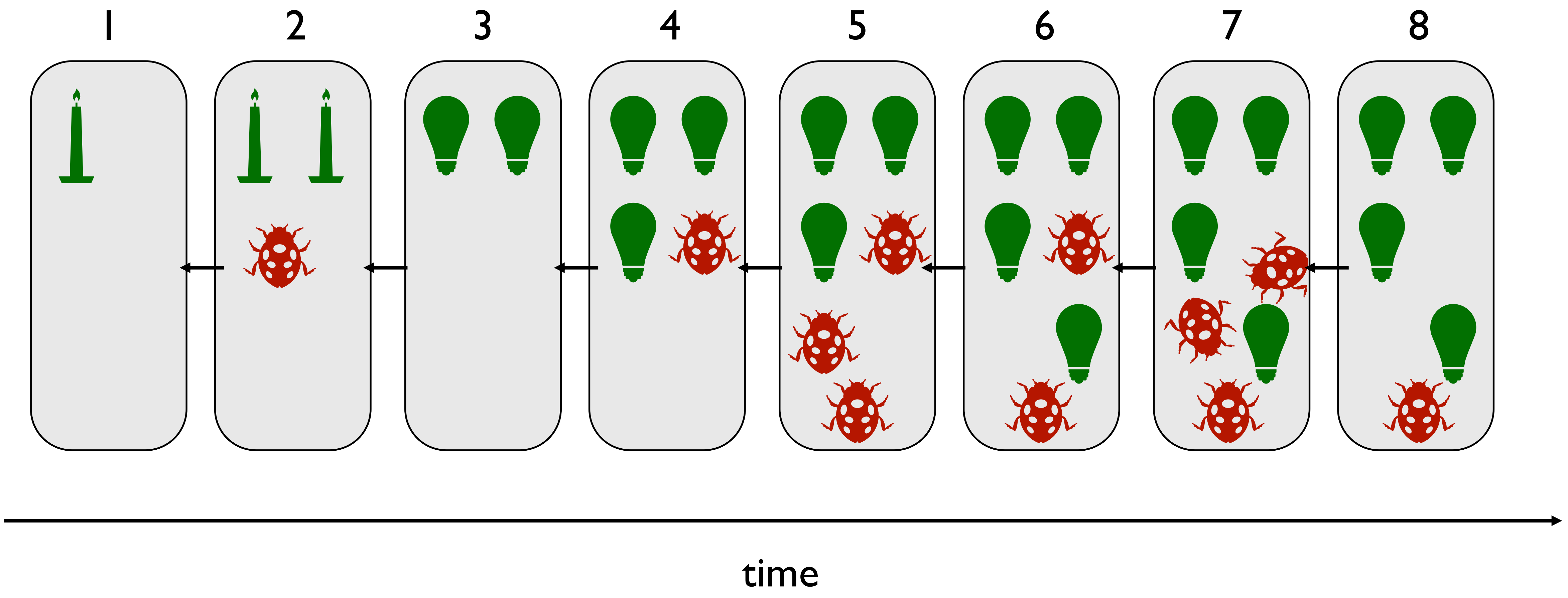
commit 3
msg: save energy
author: Victor

test.py: ✓

commit 4
msg: oops, my bad!
author: Yiyin

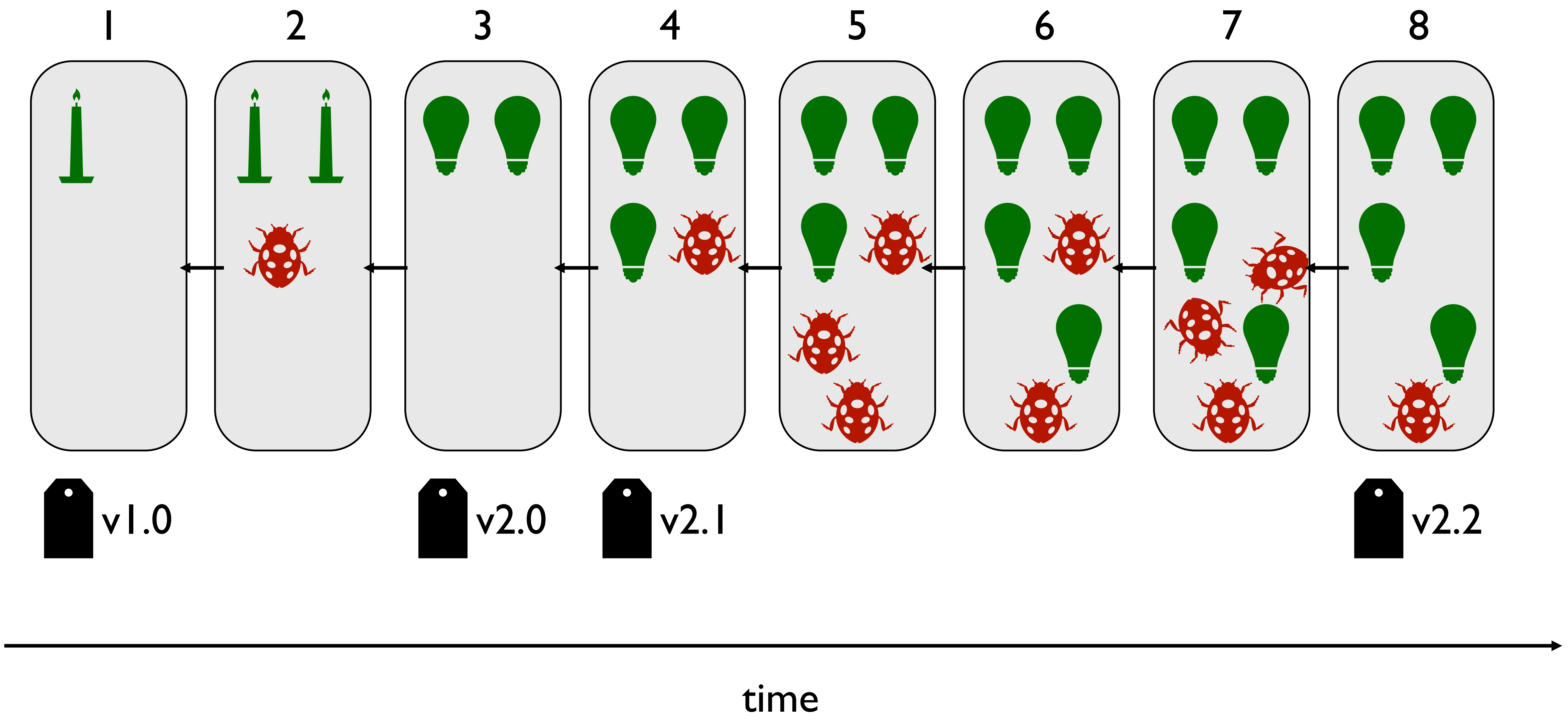
time

Use case 2: versioned releases



which version would you use?

Use case 2: versioned releases

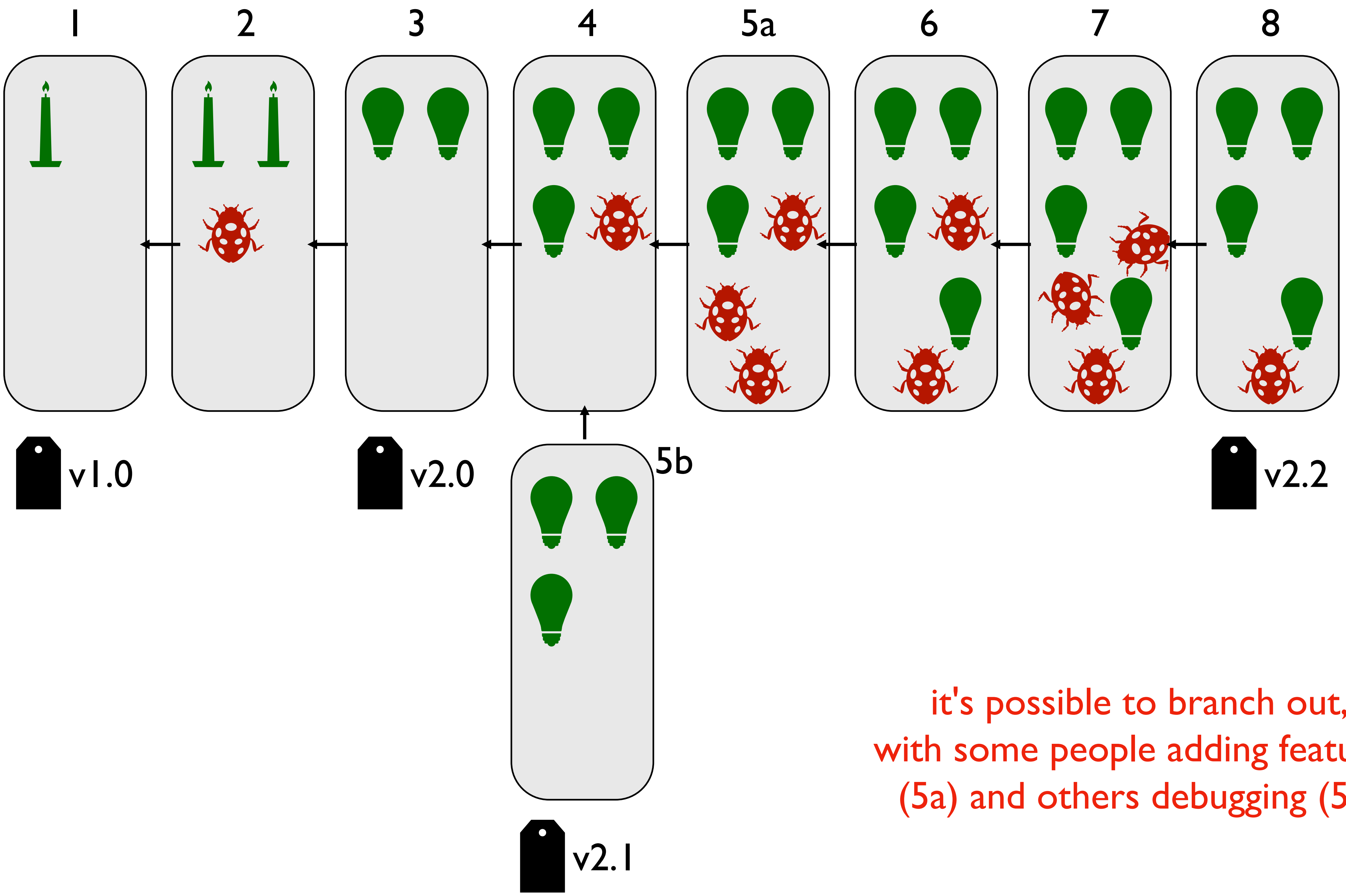


tag "good" commits to create releases

https://pypi.org/project/pandas/#history

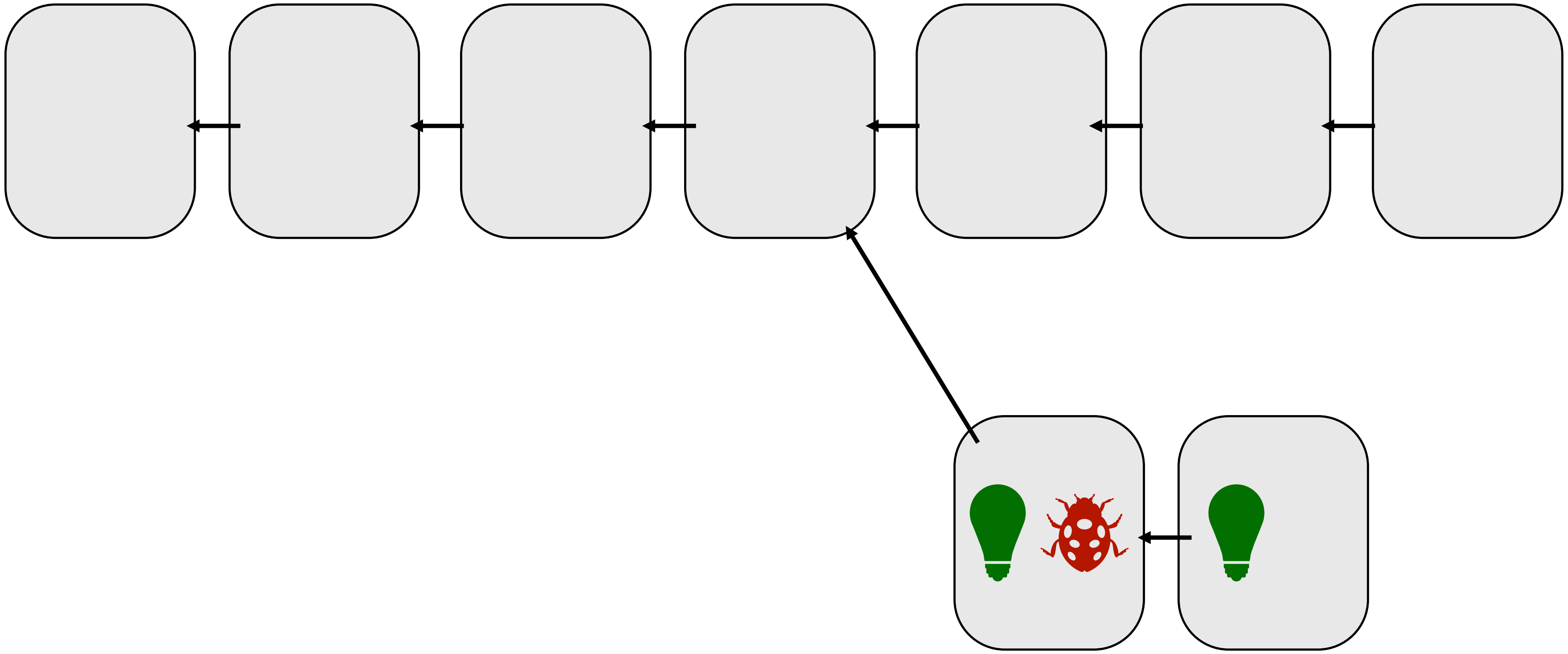
https://github.com/pandas-dev/pandas/releases

Use case 2: versioned releases



it's possible to branch out,
with some people adding features
(5a) and others debugging (5b)

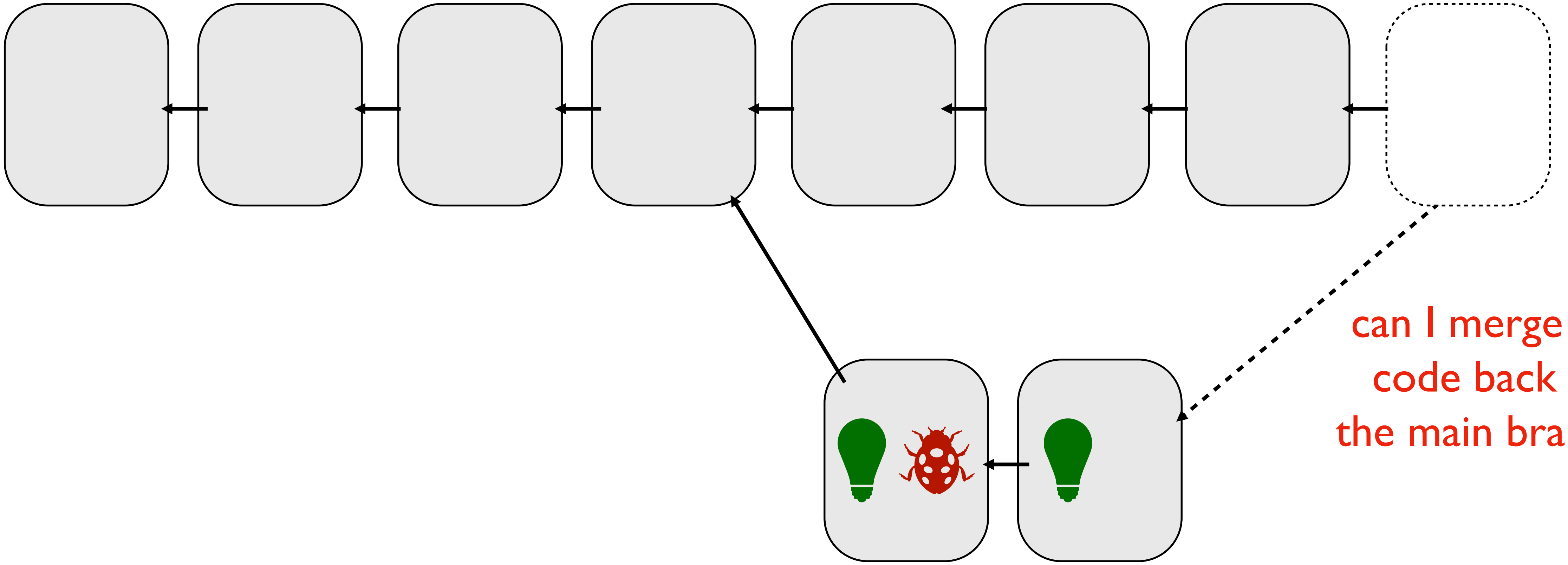
Use case 3: feedback



main branch
of code

developer's personal branch
with experimental feature

Use case 3: feedback



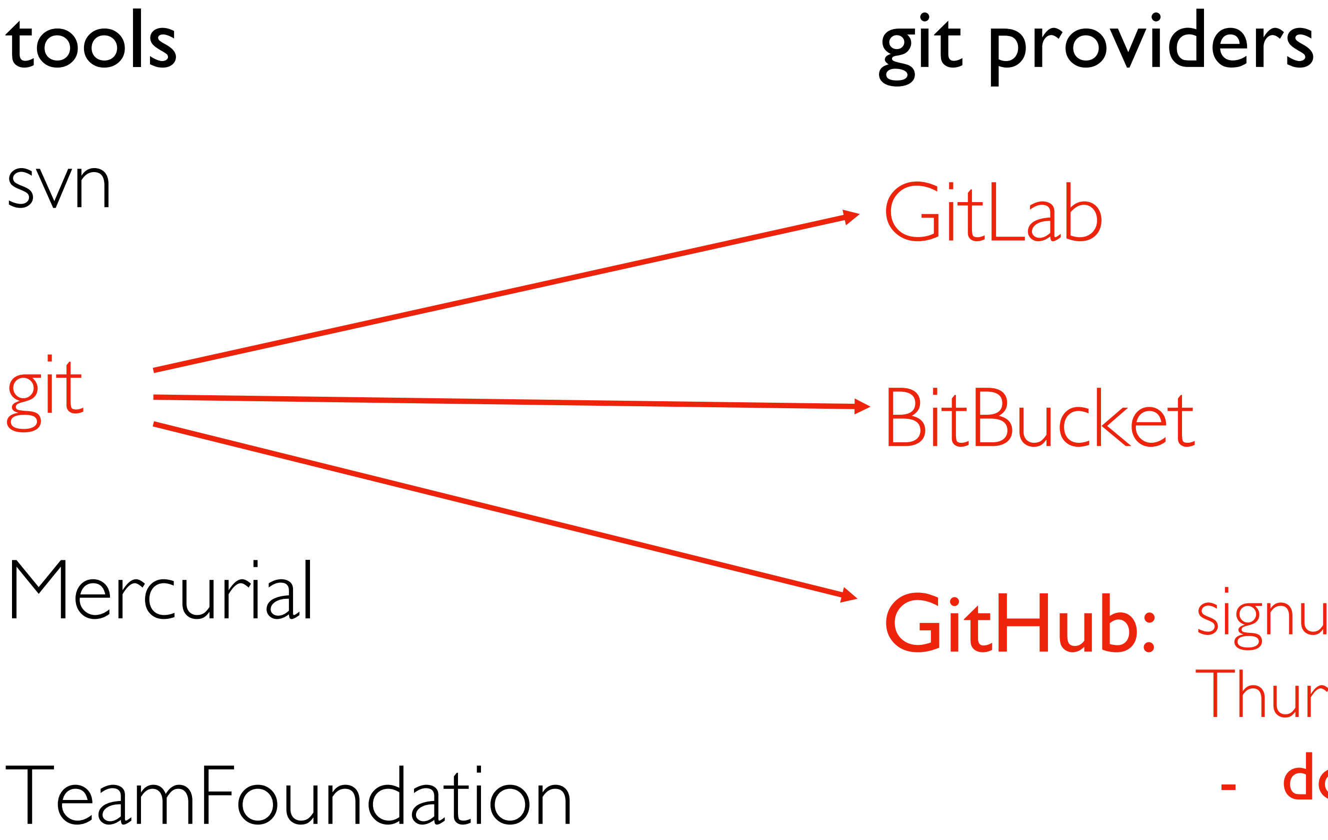
main branch
of code

can I merge my
code back to
the main branch?

developer's personal branch
with experimental feature

git

Version Control System Tools



tools

svn

git

Mercurial

TeamFoundation

git providers

GitLab

BitBucket

GitHub: signup for a free account for Thursday's lab

- **do** choose a name that won't embarrass you on a resume
- **do not** post course work

Linus Torvalds
developed git to manage
Linux as a
BitKeeper replacement

https://www.linuxjournal.com/content/25-years-later-interview-linus-torvalds

Git Demos

https://github.com/yiyins2/CS320-SU23/tree/main
https://github.com/yiyins2/CS320-SU23-lecture-notes

Connect to VM:

- Mac: terminal; Windows: powershell
- ssh username@computer: connect to a VM via SSH

Shortcuts:

^D	exit connection
^C	terminate the current command
^R	search history

pwd	display current working directly
cd	go down a directory
cd ..	go up a directory
ls	list all files in the directory
cat	display the files

Git Demos

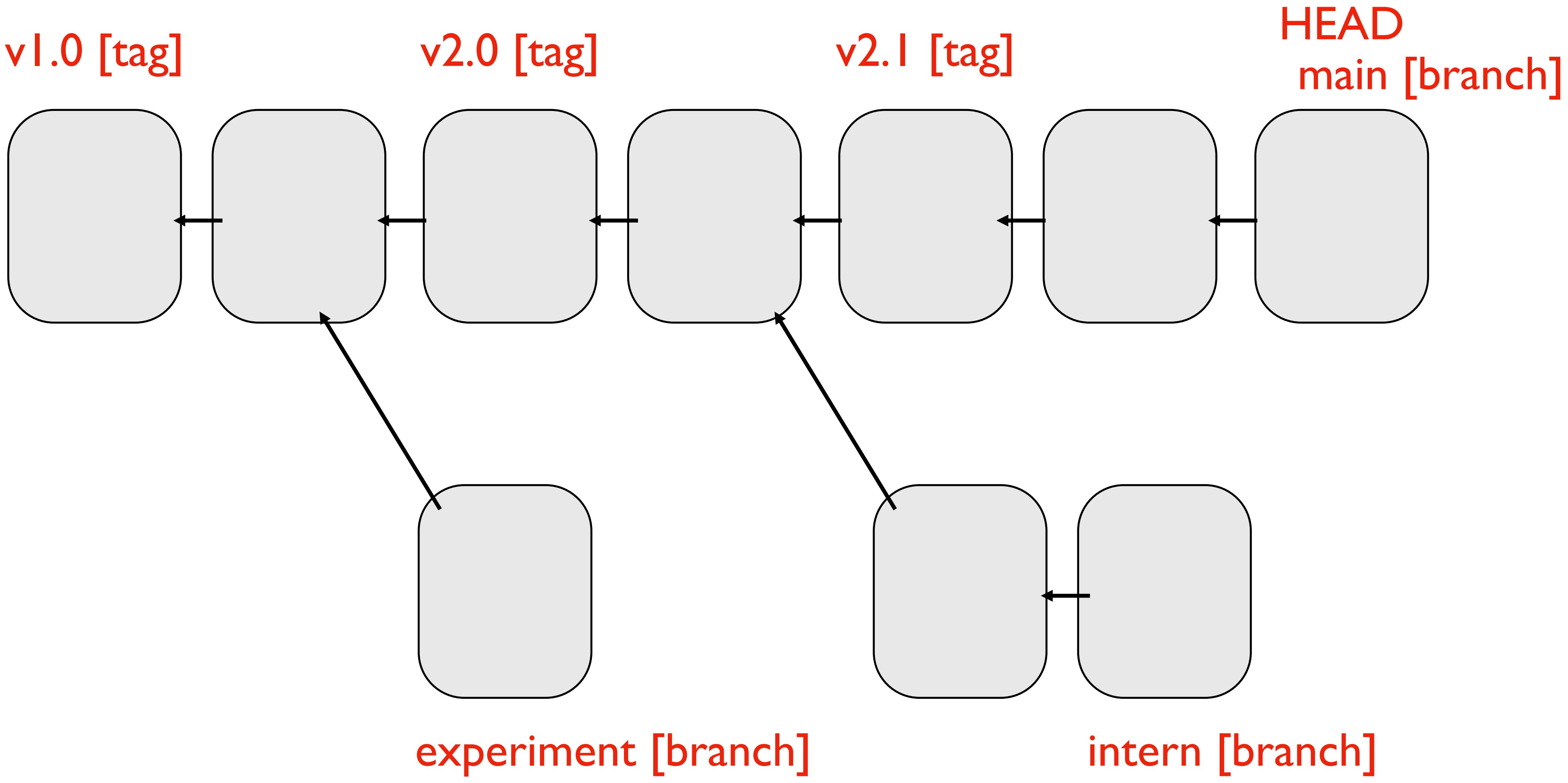
https://github.com/yiyins2/CS320-SU23/tree/main
https://github.com/yiyins2/CS320-SU23-lecture-notes

Git Commands:

- git clone: retrieve an entire repository from a hosted location via URL
- git log: show all commits in the current branch's history
- git status: show modified files in working directory, staged for your next commit
- git pull: fetch and merge any commits from the tracking remote branch
- git add: add a file as it looks now to your next commit (stage)
- git commit: commit your staged content as a new commit snapshot
- git push: transmit local branch commits to the remote repository branch
- git branch: list your branches. a * will appear next to the currently active branch
- git checkout: switch to another branch and check it out into your working directory

HEAD, Branches, and Tags

Remembering commit numbers is a pain! Various kinds of labels can serve as easy-to-remember aliases



HEAD: wherever you currently are (only one of these)
tag: label tied to a specific commit number
branch: label tied to end of chain (moves upon new commits)