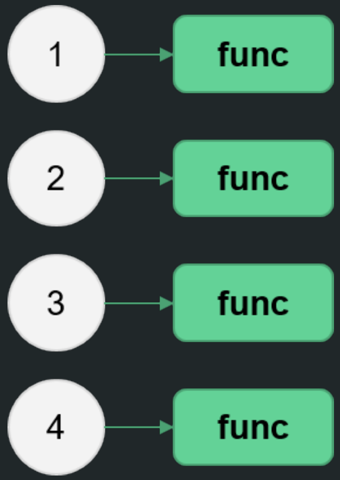
Parallel Python



Aitor Morales-Gregorio
Zbigniew Jędrzejewski-Szmek

ASPP 2025, Plovdiv

Fork/clone the repo now!

Outline

- Processes, threads and THE GIL
- Hands-on investigations of embarrassingly parallel problems
    A. Multithreading with NumPy
    B. The `multiprocessing` package
    C. Blending processes and threads
- Going further
- Wrap-up

Exercise: brainstorm

Why do we parallelize?

Talk to your partner and come up with three practical examples of where parallelization could be beneficial (in your work or another application).

Exercise: brainstorm

Why do we parallelize?

Talk to your partner and come up with three practical examples of where parallelization could be beneficial (in your work or another application).

In short, two reasons why:

- Speed up computations.
- Process “big” things.

As for the “how”...we’ll come back to that later.

Processes, threads and the *GIL*

Kitchen-2-Computer analogy

Customer
Gives orders

Large Pantry
across the street

Workstation
Includes chef, tiny countertop and tools

Transport servants
Carry stuff

Big Countertop
temporarily holds things

Kitchen-2-Computer analogy

Customer
Gives orders

Large Pantry
across the street

Big Countertop
temporarily holds things

Workstation
Includes chef, tiny countertop and tools

Transport servants
Carry stuff

Talk to your partner:
What is what?

The шопска салата program

We need to make a single dish:
<u>шопска салата</u>.

This requires:

1. Fetch vegetables from pantry
2. Wash vegetables
3. Fetch cheese from pantry
4. Grate cheese
5. Chop vegetables
6. Combine veggies and cheese
7. Place salad in the pantry

The шопска салата program

We need to make a single dish:
шопска салата.

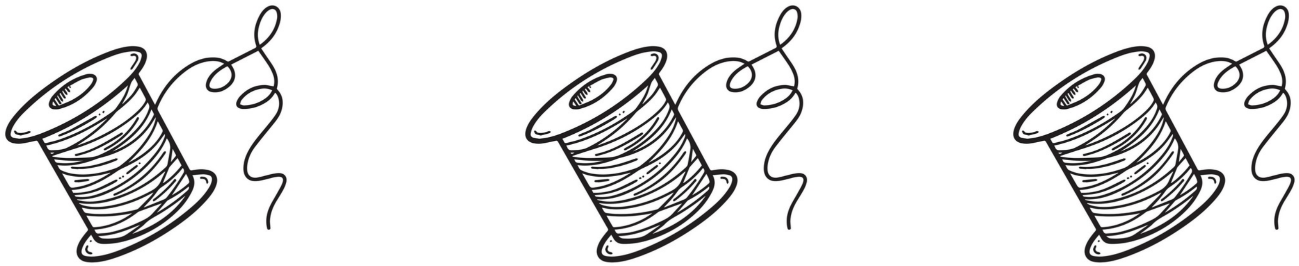
This requires:

1. Fetch vegetables from pantry
2. Wash vegetables
3. Fetch cheese from pantry
4. Grate cheese
5. Chop vegetables
6. Combine veggies and cheese
7. Place salad in the pantry

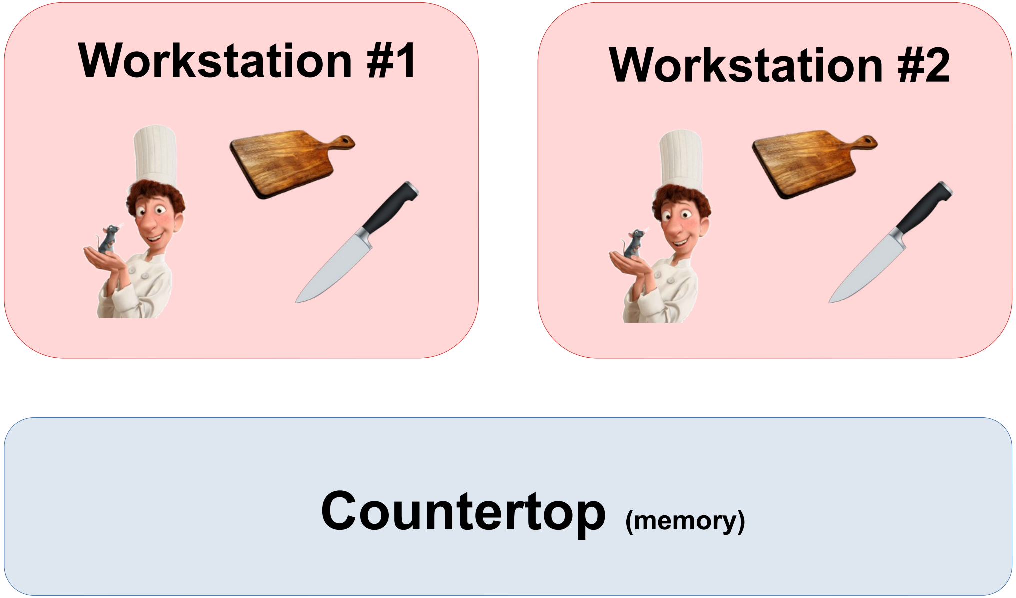
Talk to your partner:
How can we make шопска салата faster?
Without kitchen renovations

Yay! We have just designed a

multi-<u>thread</u>ed program

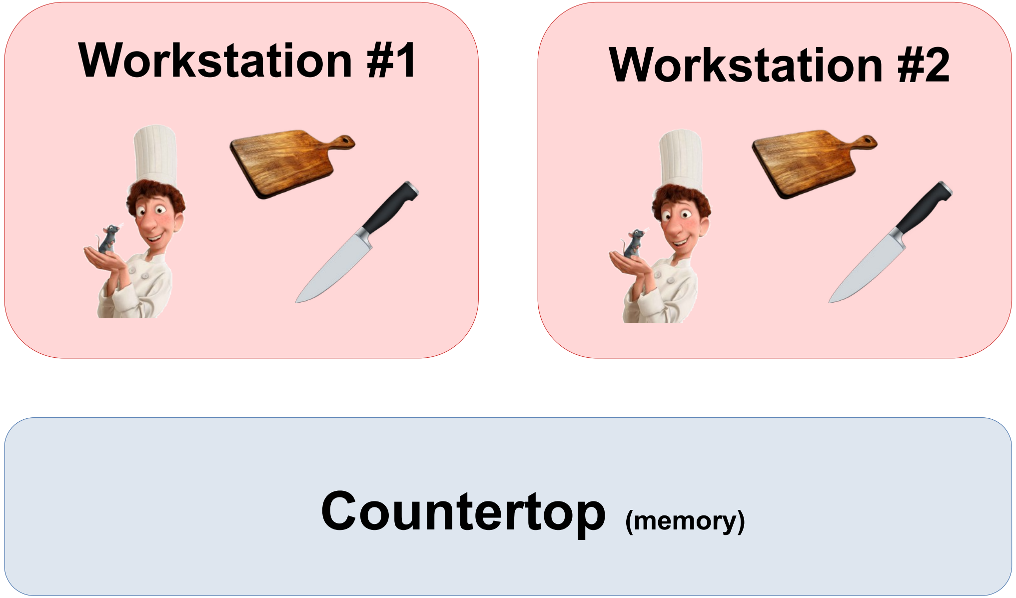


The restaurant owner made the kitchen larger!



So everything is twice as fast, **right?**

The restaurant owner made the kitchen larger!



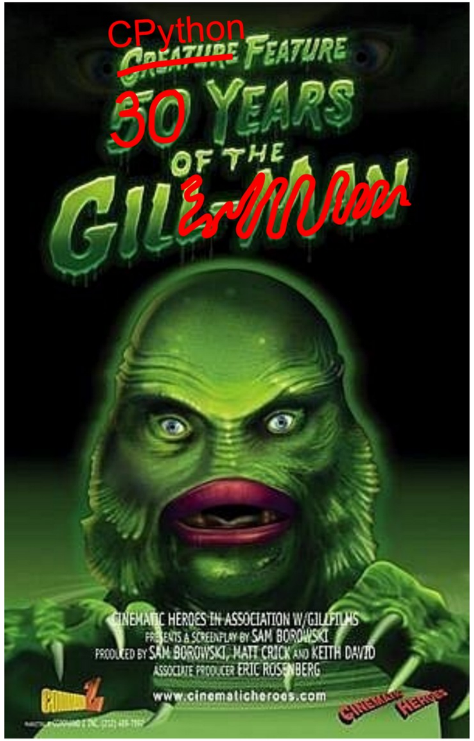
Problem #1:
Memory corruption

Problem #2:
Race conditions

Race conditions

(Live coding)

Python’s approach: The *Global Interpreter Lock* (GIL)

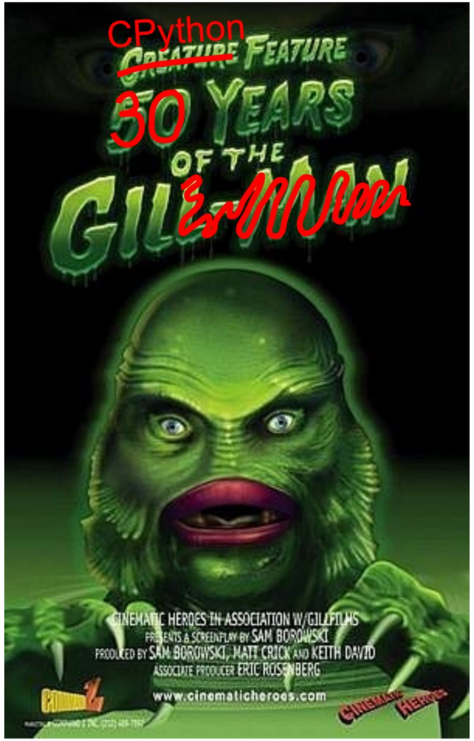


* A mutual exclusion (mutex) lock.

* Within the Python process, only 1 thread is allowed to execute pure-Python code in a given instance. Leading to **NO SPEED-UP** from multithreading.

* The lock is acquired and released by threads, approximately every 100 bytecode instructions. Also released in other cases, e.g., I/O.

Python’s approach: The *Global Interpreter Lock* (GIL)



* A mutual exclusion (mutex) lock.

* Within the Python process, only 1 thread is allowed to execute <u>pure-Python code</u> in a given instance. Leading to **<u>NO SPEED-UP</u>** from multithreading.

* The lock is acquired and released by threads, approximately every 100 bytecode instructions. Also released in other cases, e.g., I/O.

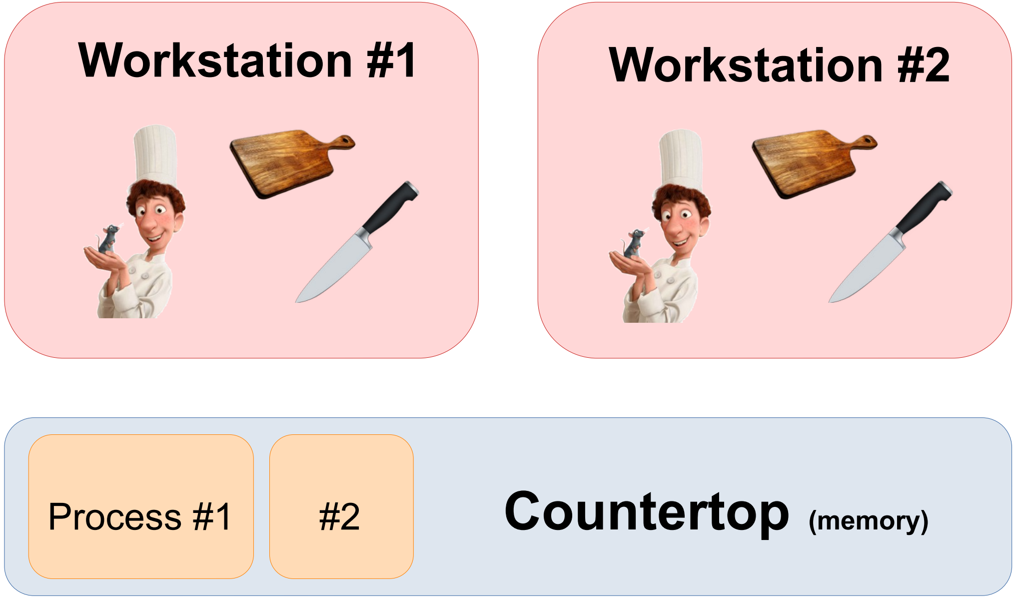
<u>Hypothesize with your partner:</u>
NumPy can achieve *massive speed-up* from multithreading. How is this possible?

NumPy's trick

NumPy runs compiled non-Python code, which operate free of the GIL

In other words, it is *many* threads disguised as one!

Faster solution: Protect the memory



<u>Multi-processing:</u>
Each process gets assigned <u>private memory</u>, other processes cannot read or write from it*, if they try there will be an error (*Segmentation Fault*)

*usually

Who does this?
The Operating System (OS)

Faster solution: Protect the memory

Great for embarrassingly parallel problems!

Workstation #1

Workstation #2

Process #1

#2

Countertop **(memory)**

<u>Multi-processing:</u>
Each process gets assigned <u>private memory</u>, other processes cannot read or write from it*, if they try there will be an error (*Segmentation Fault*)

*usually

Who does this?
The Operating System (OS)

Process
* List of instructions, i.e. an instance of some program
* Has private memory
* Can be made up of one or more threads
* Is not a physical part of the computer, it is defined by the rules of the OS

Thread
* Some instructions from the program (all instructions if single-threaded)
* Always part of a process
* Shares memory with other threads of the same process

Processes and threads do not run on a specific CPU,
the OS will allocate them and can move them mid-run

To wrap things up…a pop quiz!

On your pair computer, please navigate to **play.blooket.com** and enter game pin

Outline

- ~~Processes, threads and THE GIL~~ ✅
- Hands-on investigations of embarrassingly parallel problems
    A. Multithreading with NumPy
    B. The `multiprocessing` package
    C. Blending processes and threads
- Going further
- Wrap-up