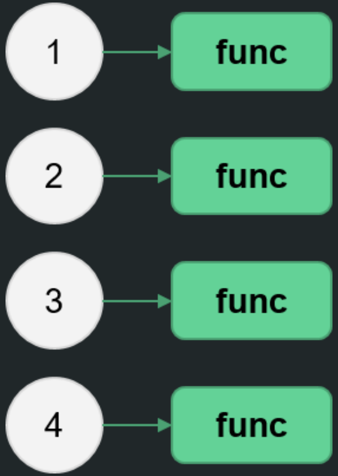
Parallel Python



Aitor Morales-Gregorio
Jenni Rinker
Zbigniew Jędrzejewski-Szmek

ASPP 2024, Heraklion

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.

Process, threads and THE GIL

The program

The dakos program.

We need to make a single dish: **DAKOS**.
This requires:

1. Fetch olive oil from the pantry
2. Fetch rusks from the pantry
3. Drizzle the rusks with water
4. Drizzle the rusks with olive oil
5. Fetch tomatoes from the pantry
6. Wash the tomatoes
7. Grate the tomatoes
8. Fetch feta from the pantry
9. Grate the feta cheese
10. Combine the softened rusks, grated tomatoes, and grated cheese
11. Place the finished dakos in the pantry

Optimize the dakos program

1. Fetch olive oil from the pantry
2. Fetch rusks from the pantry
3. Drizzle the rusks with water
4. Drizzle the rusks with olive oil
5. Fetch tomatoes from the pantry
6. Wash the tomatoes
7. Grate the tomatoes
8. Fetch feta from the pantry
9. Grate the feta cheese
10. Combine the softened rusks, grated tomatoes, and grated cheese
11. Place the finished dakos in the pantry

workstation
place for non-fetch actions

chef
a stupid box
(does actions)

countertop
temporarily holds things (recipe, intermediate ingredients, etc.)

pantry
across the street

Congratulations!

You have just designed a

multi-threaded process.

Nice explanation of processes, threads, memory in [1]

Decoding the metaphor

Key concepts:

- **Process**
  - A running program. OS assigns it space in memory for instructions/data.
- **Thread**
  - Unit of computation (i.e., set of instructions) that the OS sends to CPU for execution.
- **…and others**
  - Recall the computer architecture lecture!
  - NOTE this metaphor ignores caching.

Some* elements of our kitchen metaphor:

- Workstations
- Countertop
- ~~Stupid box~~Chef with list of tasks
- Pantry
- (extra) Restaurant owner

So which element is which?

- Workstations:
- Countertop:
- Chef w/tasks:
- Pantry:
- Restaurant owner:

**This list is non-exhaustive. ;)*

But there is a problem…

…and that problem is tomatoes.

Consider a scenario:

- The dakos program requires fetching tomatoes one-by-one.
- The "fetch tomatoes" task includes a sub-task:
  - count all the tomatoes in the pantry,
  - take 1 tomato for the dako, and
  - write on a note on the countertop how many are left in the pantry

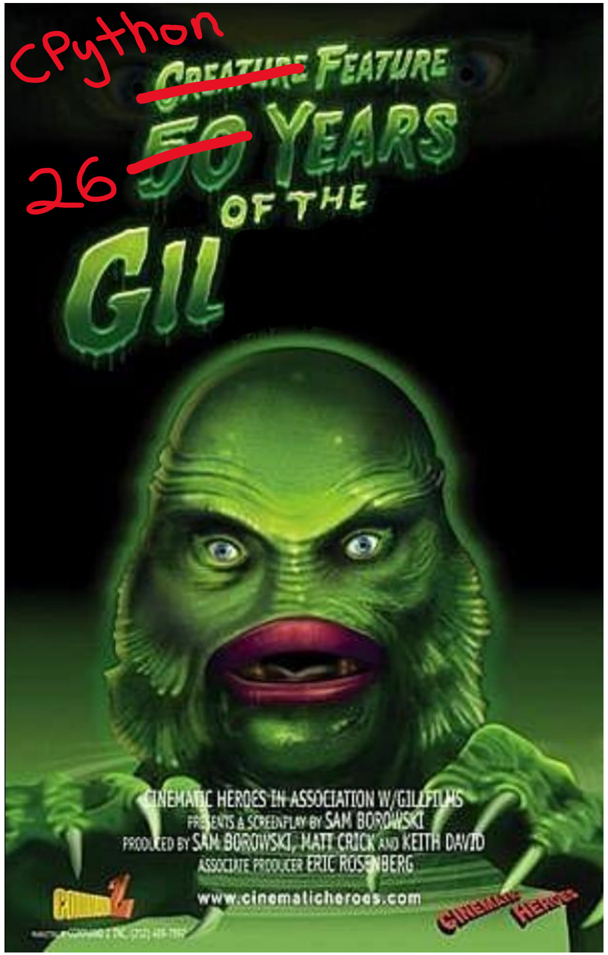
- What happens if two chefs making dakos execute their tasks at the same time?
  - The number of tomatoes will be off by 1!

This problem is called a **race condition**

Race conditions are problematic in any multithreaded code. In Python, race conditions can lead to memory corruption.

To avoid race conditions, *Python implemented something special called…

**To be specific, only CPython – Python built on the C language. Other types of Python may not have this, but you are almost certainly using CPython.*


CPython
~~Creature~~ Feature
26 ~~50~~ Years
of the
GIL
CINEMATIC HEROES IN ASSOCIATION W/GILLFILMS
PRESENTS A SCREENPLAY BY SAM BOROWSKI
PRODUCED BY SAM BOROWSKI, MATT CRICK AND KEITH DAVID
ASSOCIATE PRODUCER ERIC ROSENBERG
www.cinematicheroes.com
CINEMATIC HEROES

The “Global Interpreter Lock” (GIL)

- A “mutex” (mutual exclusion) lock.
- Within the Python process, only 1 thread is allowed to execute **pure-Python code** in a given instance.
- The lock is acquired and released by threads, approximately every 100 bytecode instructions. Also released in other cases, e.g., I/O.

Hypothesize with your partner:

NumPy can (and by default does) run code with multiple threads in parallel. How is this possible?

NumPy’s trick

NumPy interfaces with non-Python libraries that, by default, use as many threads as you have cores.

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

What does this all mean?

Computationally heavy, pure-Python code will generally have **0 speed-up** with multiple threads.

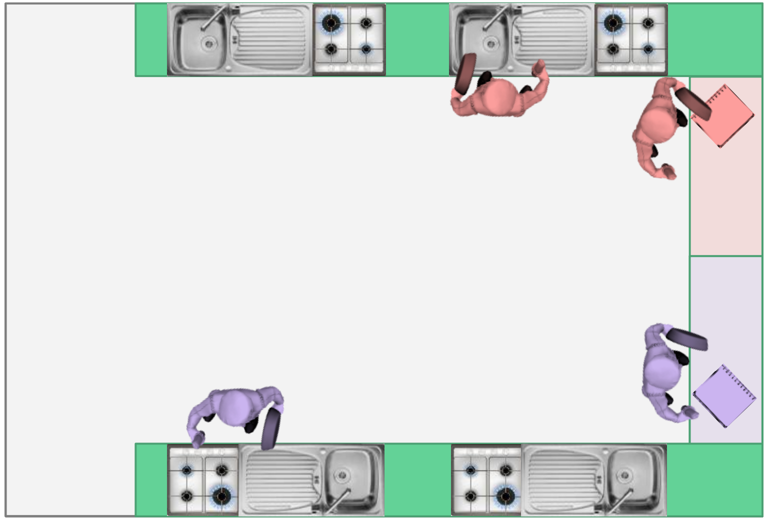
Some specific packages (NumPy) get around this by spinning up multiple threads without the Python interpreter knowing.

Note that network- and IO-bound problems release the lock and thus can be handled with multithreading.

Guess: how can we get around the GIL for non-NumPy, non-I/O code?

- Instead of multiple threads, use multiple *processes*.

Multiprocessing: multiple teams of chefs



Red team and purple team split dakos tasks.

Each team can have 1 chef working.

Each process is an instance of the Python interpreter and therefore has its own GIL!

BUT processes have separate memory, so data must be duplicated in each process.

Multiple processes therefore have additional computational overhead and memory usage.

To wrap things up…a pop quiz!

On your pair computer, please navigate to **kahoot.it** 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

Supplementary material

Further reading

1. Open textbook on Operating Systems. Chapters 4, 13 and 26 are particularly nice. pages.cs.wisc.edu