Concurrency and Parallelism in Python.

Not as complicated as you think.

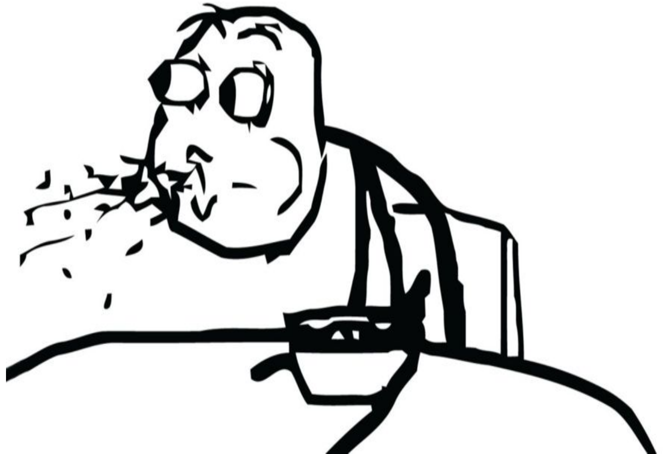
By Ibukun oluwayomi.
Software Development Engineer @ Amadeus Services London.
https://github.com/ioluwayo

Agenda

- What really is concurrency?
  - Is it the same as Parallelism?
- The Global interpreter lock (GIL).
- What tools for concurrency are available to python developers.
  - What are the limitations of these tools.
- Code samples and funny pictures.

Disclaimer: The opinions expressed in this presentation and on the following slides are mine and not that of my employer.

Concurrency is not Parallelism.



So what is concurrency?

“Two events are concurrent if neither can caussally affect the other.” Leslie *Lamport 1976.*

A way to design applications, programs, algorithms such that they comprise of components that cannot causally affect each other.

Think independence and decomposition into separate components. Makes it easy to understand and scale (cue parallelism).

So, multiple *things* can be **dealt** with at the same time …….

What is parallelism then?

Actually **doing** multiple things at the same time.

It is a consequence of concurrency. When an application has been designed such that components and processes are independent of each other, then multiple cpu cores can be recruited to handle them simultaneously.

They are different but related.

Concurrency permits parallelism

Concurrency

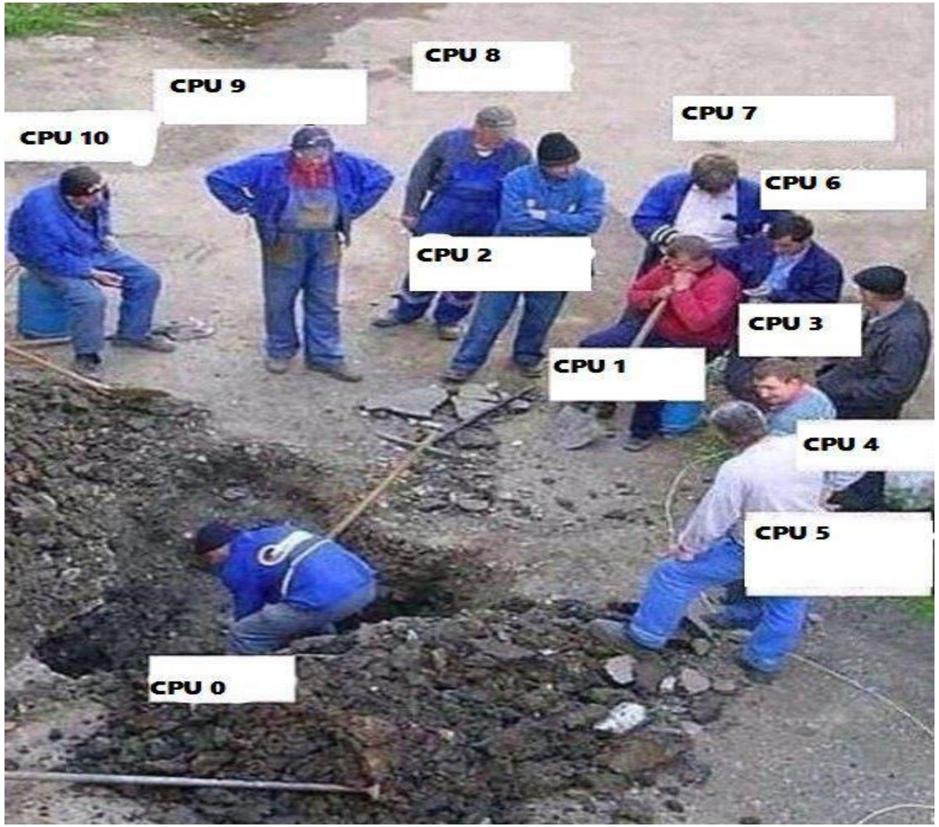
- Application design and structure.
- Decoupled and independent processes. Multiple things can be dealt with at the same time.
- Facilitates techniques like interleaved processes and parallelism.
- More general than parallelism.

Parallelism

- Doing multiple things at the same time (execution).
- Without concurrent design, it’s impossible to implement parallelism successfully.
- At least two tasks execute literally at the same time.
- Requires hardware with multiple processing units.

You've probably heard that Python does not support multithreading.

Well that is not true but you need to know a few things to prevent this.



The Global Interpreter Lock.

(AKA The GIL)

Multithreading and the GIL.

- The GIL prevents multiple python threads from executing the same python bytecode at once.
- Only one thread can make forward progress at any given time.
- Execution time is split among multiple threads. Python tries to be fair by allocating equal amount of time to each thread.
- There is some over head due to thread management. (Acquiring and releasing the GIL between threads.)

Use threads for IO intensive tasks.

- The GIL is released when waiting for responses from 3rd party resources. (System calls, TCP sockets, database transactions)
- The GIL is released when executing thread safe C libraries.
- The number of threads created should be controlled.

Multiprocessing and the GIL

- This is python’s solution to the GIL problem.
- True parallelism can be achieved using this module.
- Multiple processes are run simultaneously on different CPU ‘cores’.
- Memory is not shared across processes.
- The module abstracts away details of communication between python processes.

Use Multiprocessing for CPU intensive tasks to improve performance.

- Spinning up new processes is costly.
- Use process pool to control the amount of processes created.
- Idle processes will take up new tasks from a Queue of tasks.

Does the GIL protect against data races?

NO

- Threads can be interrupted at any time between operations.
- Atomic operations are safe.
- Non atomic operations are not.
- Use the Threading.Lock class to prevent race conditions.

Thank you. Any ~~Easy~~ Questions?

Sources

- Awesome blog post on how parallelism is not concurrency. By Lindsey Kuper
- Awesome video on Concurrency. By Rob Pike.
- In depth analysis of the GIL by David Beazley.
- Effective python by Brett Slatkin.
- Expert python programming by Michal Jaworski and Tarek Ziade.