Process and Threads in Python

Do you know how many CPU(s) are there in your computer? What is the model name?

```
>>> import platform
>>> platform.processor()
'arm'

>>> platform.system()
'Darwin'

>>> platform.architecture()
('64bit', '')

>>> import multiprocessing
>>> multiprocessing.cpu_count()
8
```

Process

- A process is basically a program in execution.
- For example, we write a program to make an entry into database table. When we execute this program CPU allocates a processor to execute this program .
- The process keeps running until program has finished off with the task.
- Processes do NOT share memory.
- Processes are not lightweight and take bit of more time to start and terminate (as compared to threads).
- **Process life cycle:** Start → Ready → Running → Waiting → Exit

Thread

- A thread is an execution unit that is part of a process.
- Every thread is part of a process and NOT vice versa.
- A process can have multiple threads.
- Threads can share memory.
- Threads are lightweight and takes less time to create and terminate.
- At any given time, there is at least one thread per process.

IO operations

- There are three main jobs of a computer - input, output and processing.
- Most of the times it's input and output and processing is simply incidental.
- Data-in and data-out operations are termed as `IO operations`.
- Following are the examples of IO operations -
  - Reading and writing data from and to the file
  - Reading and writing data into database
  - Network communication (accessing third party APIs)

How does my computer do several things at once?

- On a single core machine, it doesn't.
- Computers can only do one task (or *process*) at a time. But a computer can change tasks very rapidly, and fool slow human beings into thinking it's doing several things at once. This is called *timesharing*.
- One of the kernel's jobs is to manage timesharing. It has a part called the *scheduler* which keeps information inside itself about all the other (non-kernel) processes.

`Concurrency` vs `Parallelism`

- `Concurrency` is when two or more tasks can start, run, and complete in overlapping time **periods.** It does NOT necessarily mean they'll ever both be running **at the same instant.** For example, *multi-tasking* on a single-core machine.
- `Parallelism` is when tasks *literally* run at the same time, e.g., on a multicore processor.

GIL 🔒

(Global Interpreter Lock)

- GIL is a limitation in Python.
- Due to GIL, Python threads are restricted to an execution model that only one thread can execute in the interpreter at any given time. This blocks us from taking advantage of multiple CPU cores.
- For this reason, Python threads are generally not used for compute intensive tasks.
- Python threads are much suited for IO operations.
- It is mutex that protects to access Python objects, preventing multiple threads from executing Python byte code at once.
- This lock is necessary because underlying `CPython` implementation is not thread-safe.
- It acts as a gate keeper on the object allowing one thread in and blocking access to all others.

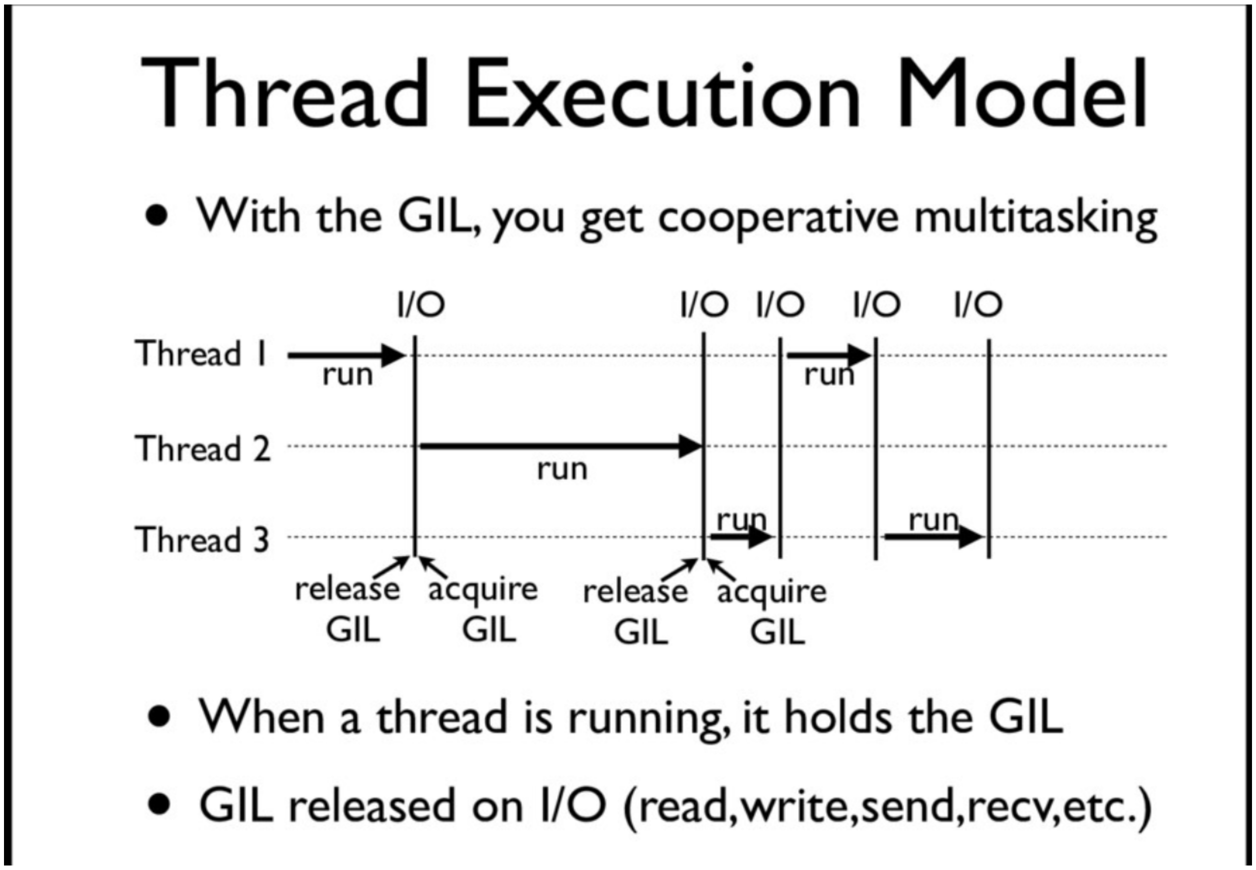
Simple analogy -

```
Suppose you're part of a big hitted discussion.

You establish a rule with a "ball" being passed around.

RULE - "whoever is holding the ball can speak. Other should shut their mouth."

This rule ensures people do not speak over each other.

Here ball is a "mutex" and person holding the ball is a "thread"
```

```
Word "mutex" means "Mutually Exclusive Object"
```

Python is a linear language

To being with, Python was designed to run on single core machine (that gets to tell you how old Python is!).



Thread Execution Model

- With the GIL, you get cooperative multitasking
- When a thread is running, it holds the GIL
- GIL released on I/O (read,write,send,recv,etc.)