GIL (Global Interpreter Lock)

Concurrency & Parallelism in Python

Piyus Gupta
Sat, 20th May 2017
BangPypers @ **vm**ware India

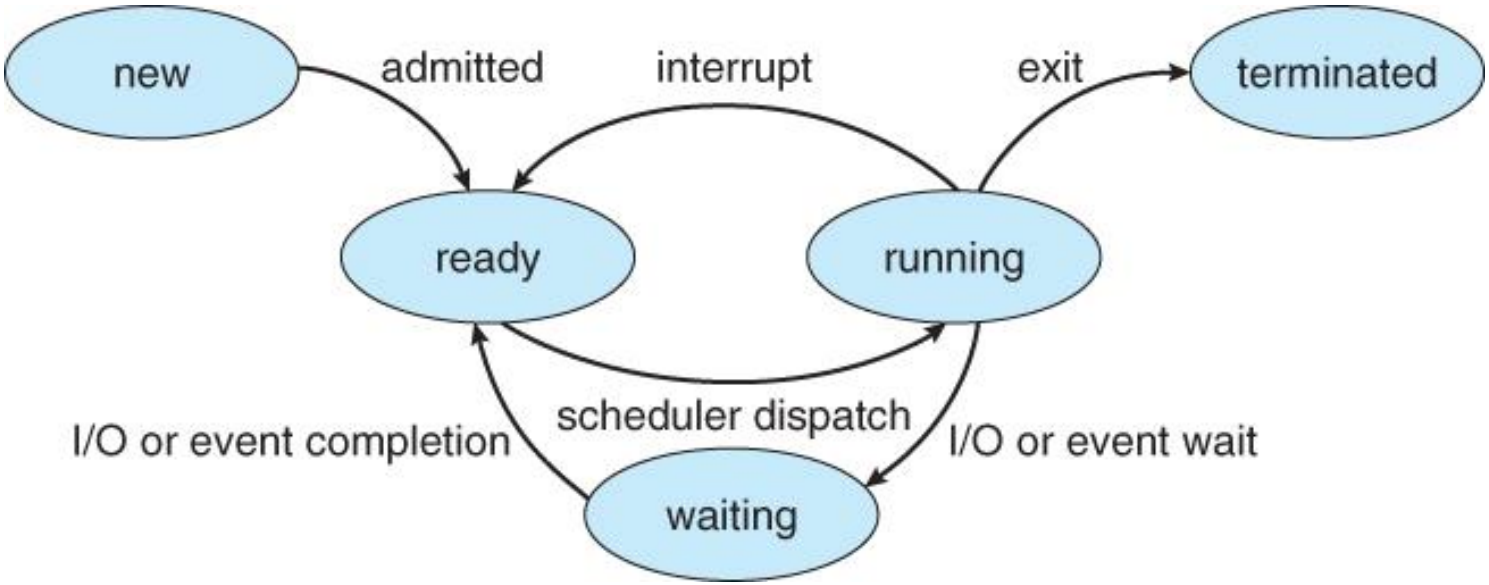
Contents:

- Some sample code before we drill GIL
- Thread Analogy
- CPU Scheduling
- Python Threads
- What is GIL ? And why is it needed ?
- High CPU usage with multithreading vs multiprocessing
- Can we really remove GIL ?

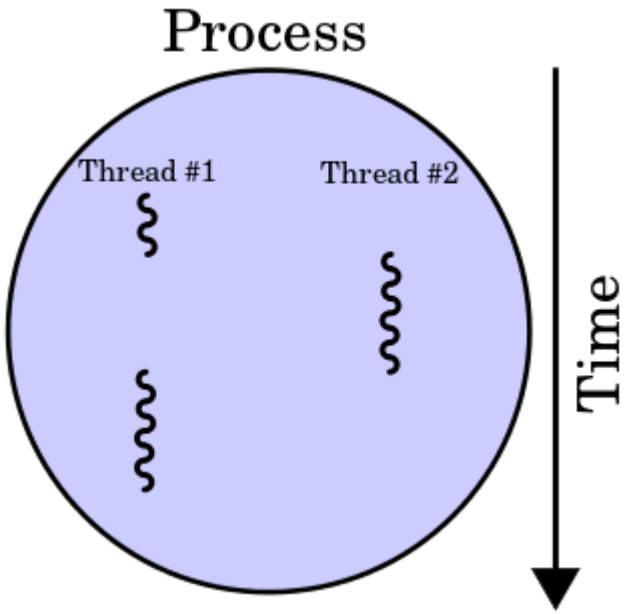
Single Threaded

Multi Threaded

CPU Scheduling



Threads vs Process



Python Threads

- ✓Python threads are real system threads
  - ✓POSIX threads (pthreads) on Unix
  - ✓Windows threads
- ✓Fully managed by host operating system
- ✓Represents threaded execution of the Python interpreter process (written in C)

What is Thread Safe ?

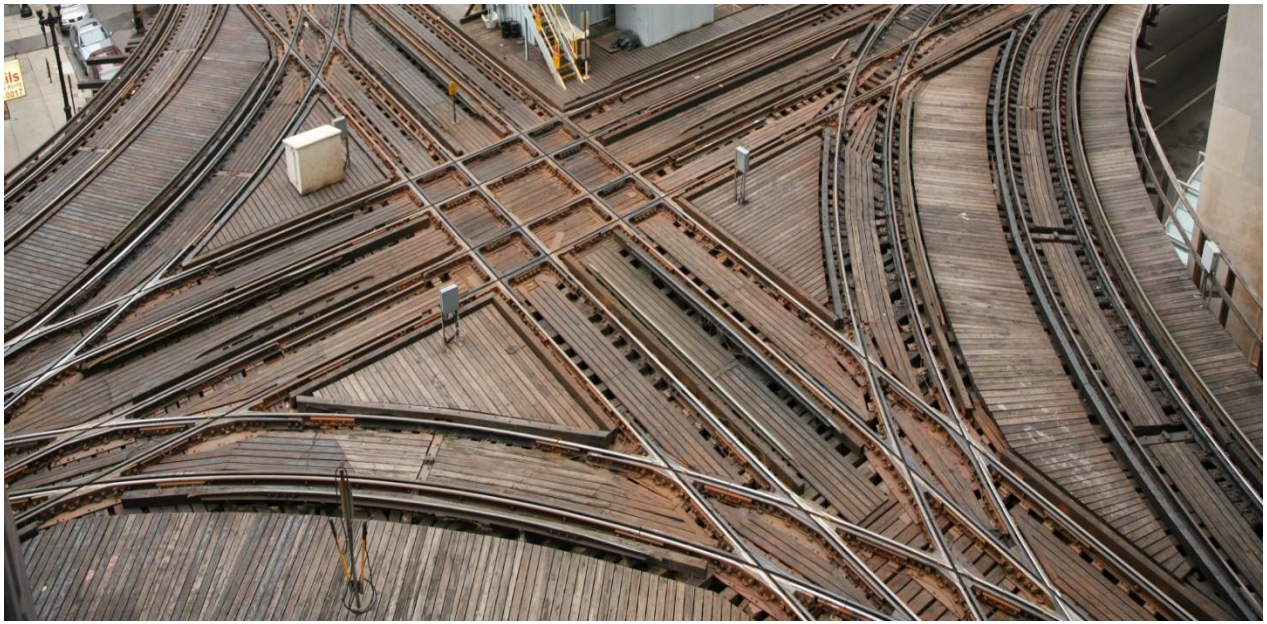
- ✓A piece of code is thread-safe if it functions correctly during simultaneous execution by multiple threads.

- ✓It must satisfy the need for multiple threads to access the same shared data, and the need for a shared piece of data to be accessed by only one thread at any given time.

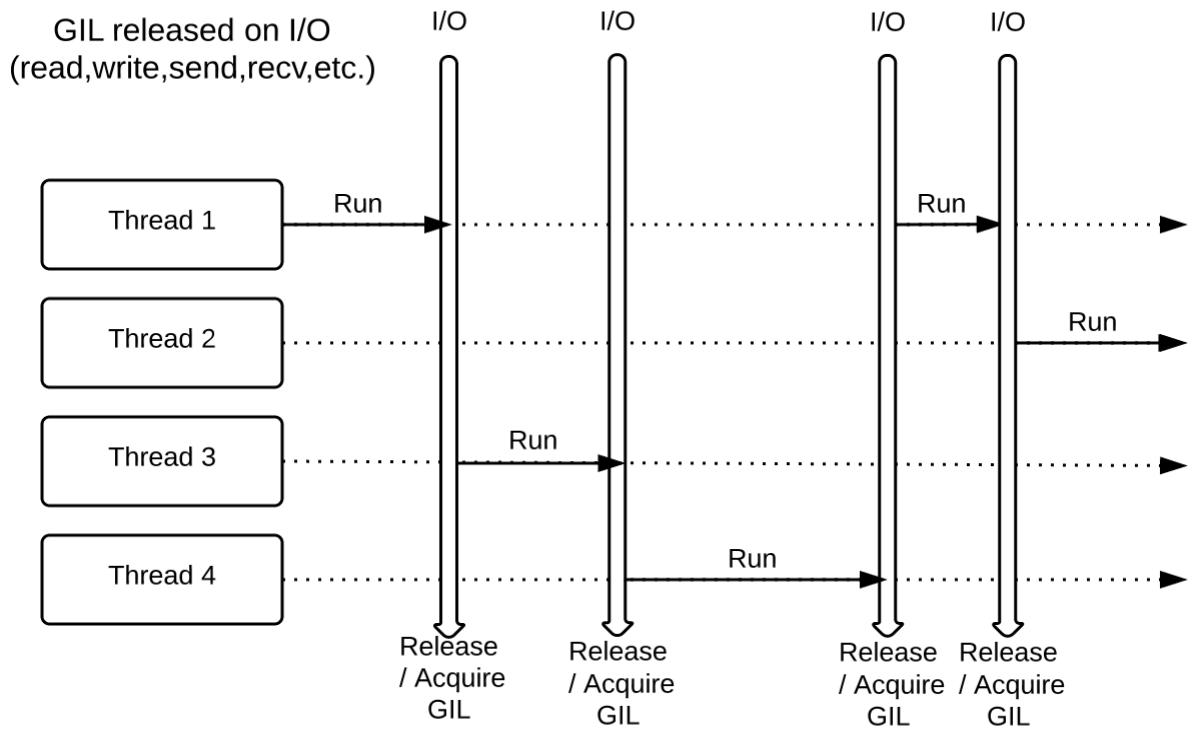
GIL Basics

- ✓Parallel execution is forbidden
- ✓There is a "global interpreter lock"
- ✓The GIL ensures that only one thread runs in the interpreter at once
- ✓Simplifies many low-level details (memory management, callouts to C extensions, etc.)

Mutual Exclusion

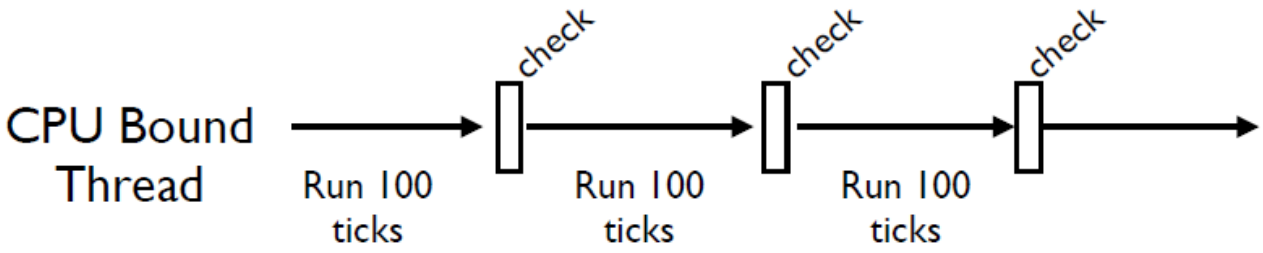


GIL working strategy



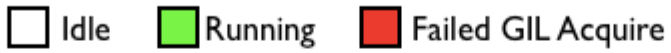
CPU Bound Task

- ✓ CPU-bound threads that never perform I/O are handled as a special case
- ✓ A "check" occurs every 100 "ticks“
- ✓ You can Change it using sys.setcheckinterval()

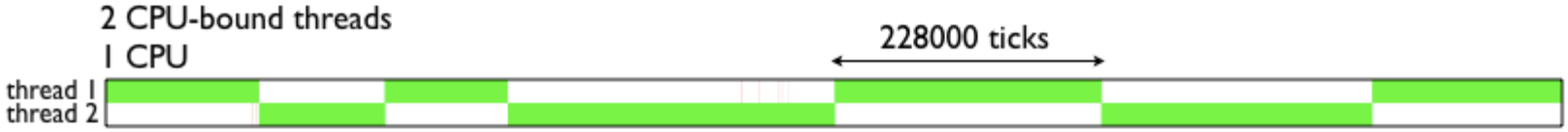



```
Python 2.7.5 (default, Jul 18 2013, 18:13:04)
[GCC 4.1.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> sys.getcheckinterval()
100
```

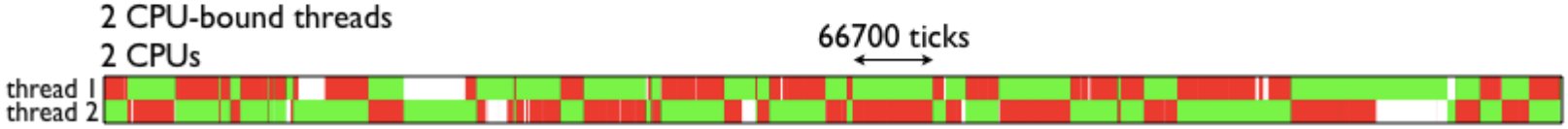
GIL - Visualized



Okay, now let's look at some threads. First, here is the behavior of running two CPU-bound threads on a single CPU system. As you will observe, the threads nicely alternate with each other after long periods of computation.



Now, let's go fire up the code on your fancy new dual-core laptop. Yow! Look at all of that GIL contention. Again, all of those red regions indicate times where the operating system has scheduled a Python thread on one of the cores, but it can't run because the thread on the other core is holding it.



Why GIL ?

- ✓Increased speed of single-threaded programs.
- ✓Easy integration of C libraries that usually are not thread-safe.
- ✓It is faster in the multi-threaded case for i/o bound programs.

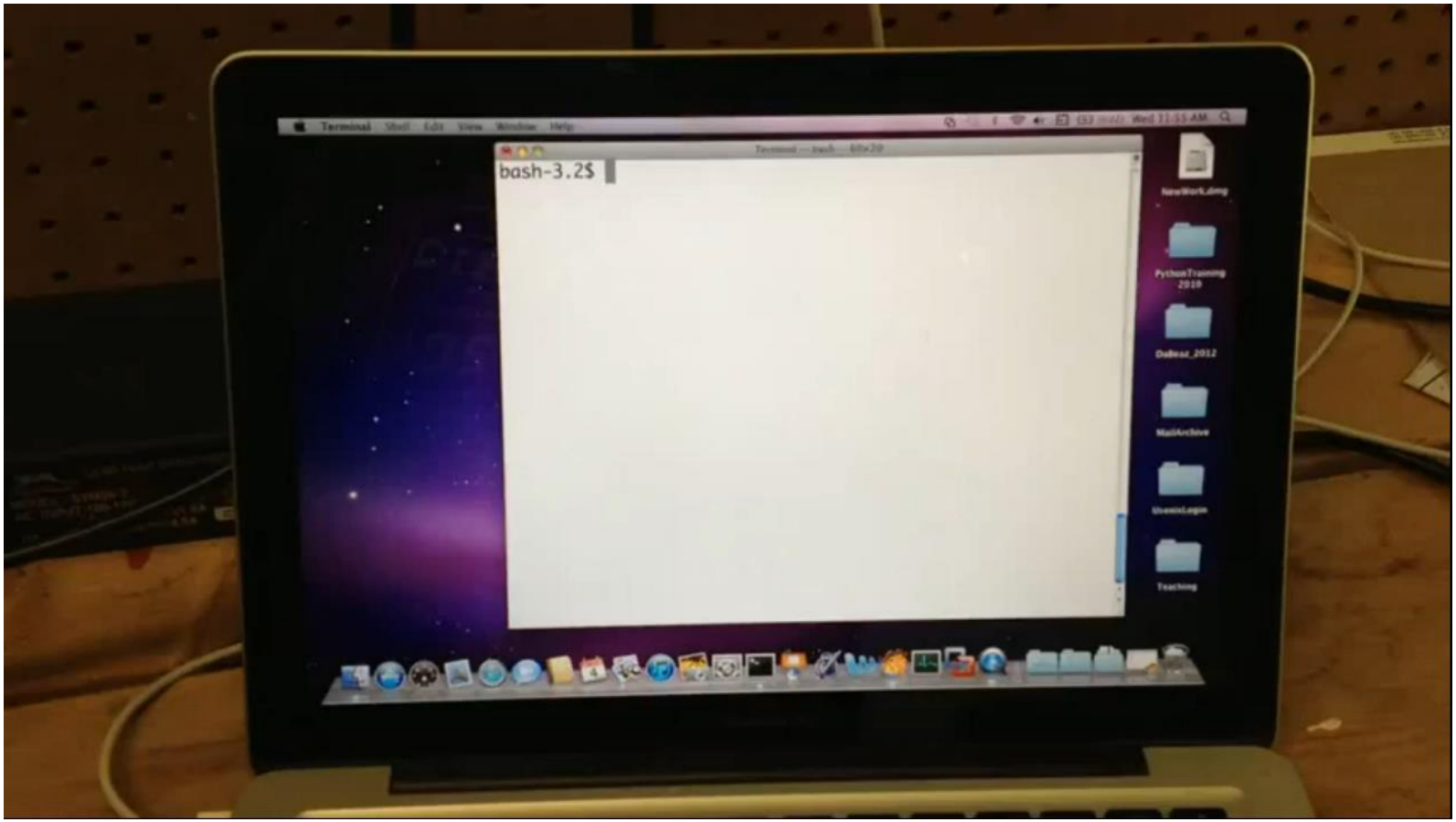
Can’t we get rid of GIL ?

- ✓Back in the days of Python 1.5, Greg Stein actually implemented a comprehensive patch that removed the GIL and replaced it with fine-grained locking.
- ✓Unfortunately, even on Windows (where locks are very efficient) this ran ordinary Python code about twice as slow as the interpreter using the GIL.
- ✓On Linux the performance loss was even worse because pthread locks aren’t as efficient.

But what about performance then ?

Conclusion

Only way to remove GIL !



References:

- http://www.dabeaz.com/GIL
- https://www.youtube.com/watch?v=Obt-vMVdM8s
- https://www.youtube.com/watch?v=SNBKWuM-Lu8
- https://randu.org/tutorials/threads/

Next Read:

- AsyncIO
- https://pymotw.com/3/concurrency.html

E-mail: piyusgupta01@gmail.com

GitHub: https://github.com/piyusgupta/juggler


Thank You