What is wrong with Python threads

Python threads are plain old POSIX threads. This means that the threads created by Python programs are normal OS threads. As we all understand, the Python interpreter is a virtual machine (much like Java’s Virtual Machine). Unlike Java, it has no intelligence of thread management thus no thread priorities and no pre-emption. Since the threads created by Python are normal OS threads, the OS is responsible for supervising these threads and it takes care of the scheduling aspects. What does the interpreter do? Well, it does the book keeping on which thread is running when the context switch happens etc.

All this is ok, but where is GIL?

This is where the problem starts. Even though the Python interpreter starts the threads and the OS manages them, each running thread spawned by the interpreter requires exclusive access to data structures in Python source. Exclusive access is needed because Python’s memory management is not thread safe. Exclusivity points to the need for a synchronisation mechanism. Hence the rise of the Global Interpreter lock or GIL. Since only one thread can have exclusive access to Python data structures, the synchronisation is done by GIL and it allows for only one thread running at a time.

You can relate this to our code example in section 1 where even though two threads were scheduled by the OS, only one could run at a time and the other was run in a *runnable* state. When a ***runnable*** thread acquires the lock
it starts *running* and performs operations. That’s why, both the threads literally ran sequentially and it took them 6 secs to finish the complete operation.

GIL in Python source

1. A thread request in Python is a simple pthread_create() call.
2. Py_Initialize() creates the GIL and does the bookkeeping of threads.
3. As you’d have imagined by now, GIL is a mutex or a lock and implemented for synchronisation between threads.
4. static PyThread_type_lockinterpreter_lock = 0; #This is GIL

GIL Management

What happens when Python threads are created? Let’s zoom in further.

1. The Python interpreter implements a check. A check is a counter of ticks and a tick represents a Python VM byte code instruction.
2. A check dictates the CPU time slice available to a thread to execute byte code.
3. As soon as the check duration is elapsed, the current running thread releases the GIL.
4. It then signals all the ‘runnable’ threads that the GIL is free.
5. All the threads, including the one that released the GIL just now, battle for acquiring the GIL.