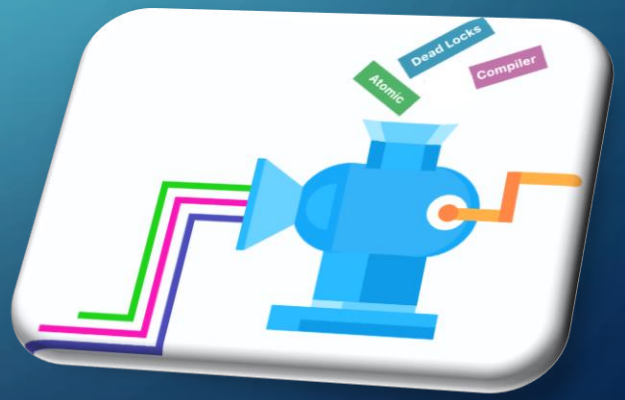
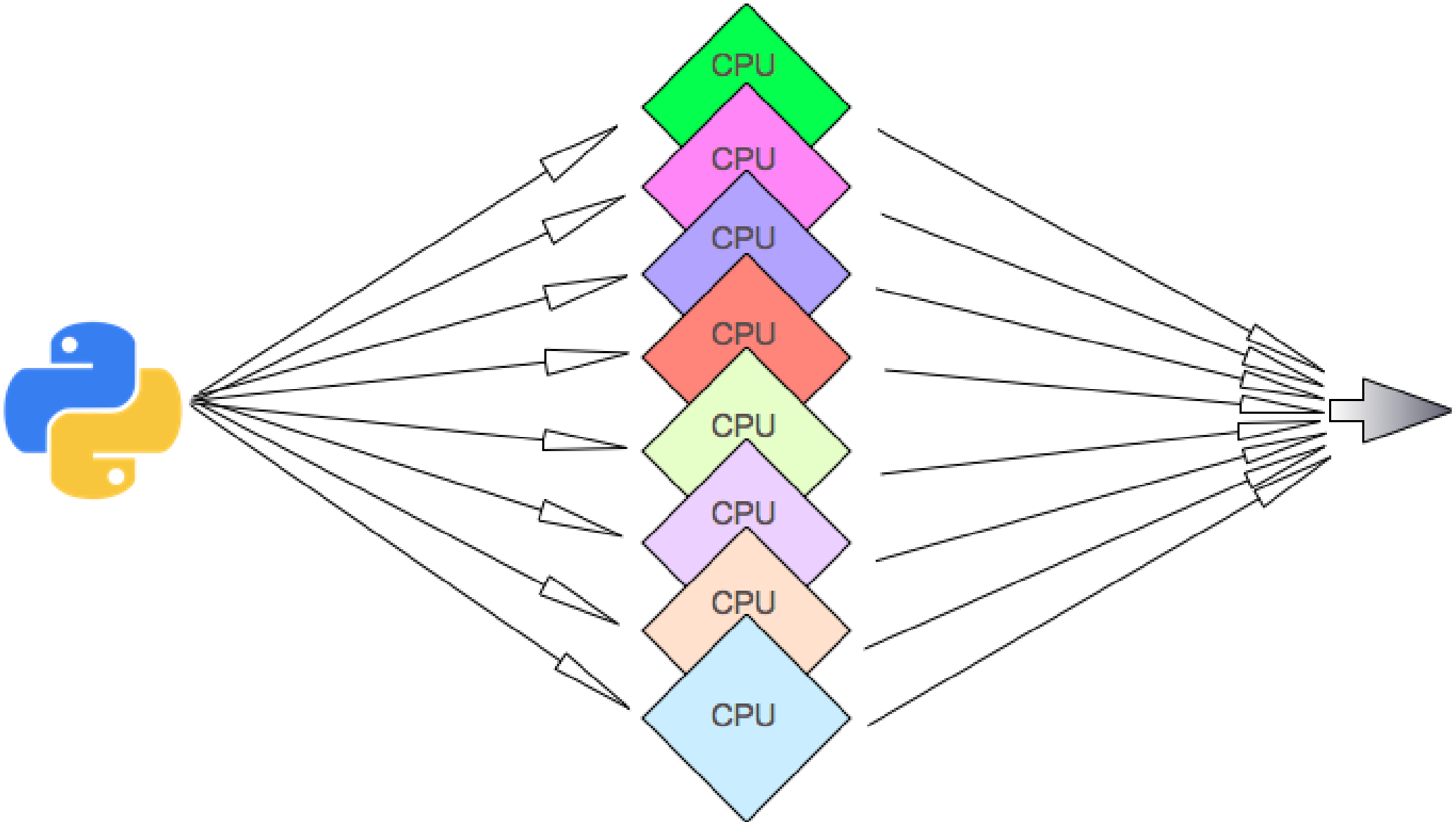
PARALLEL PROGRAMMING

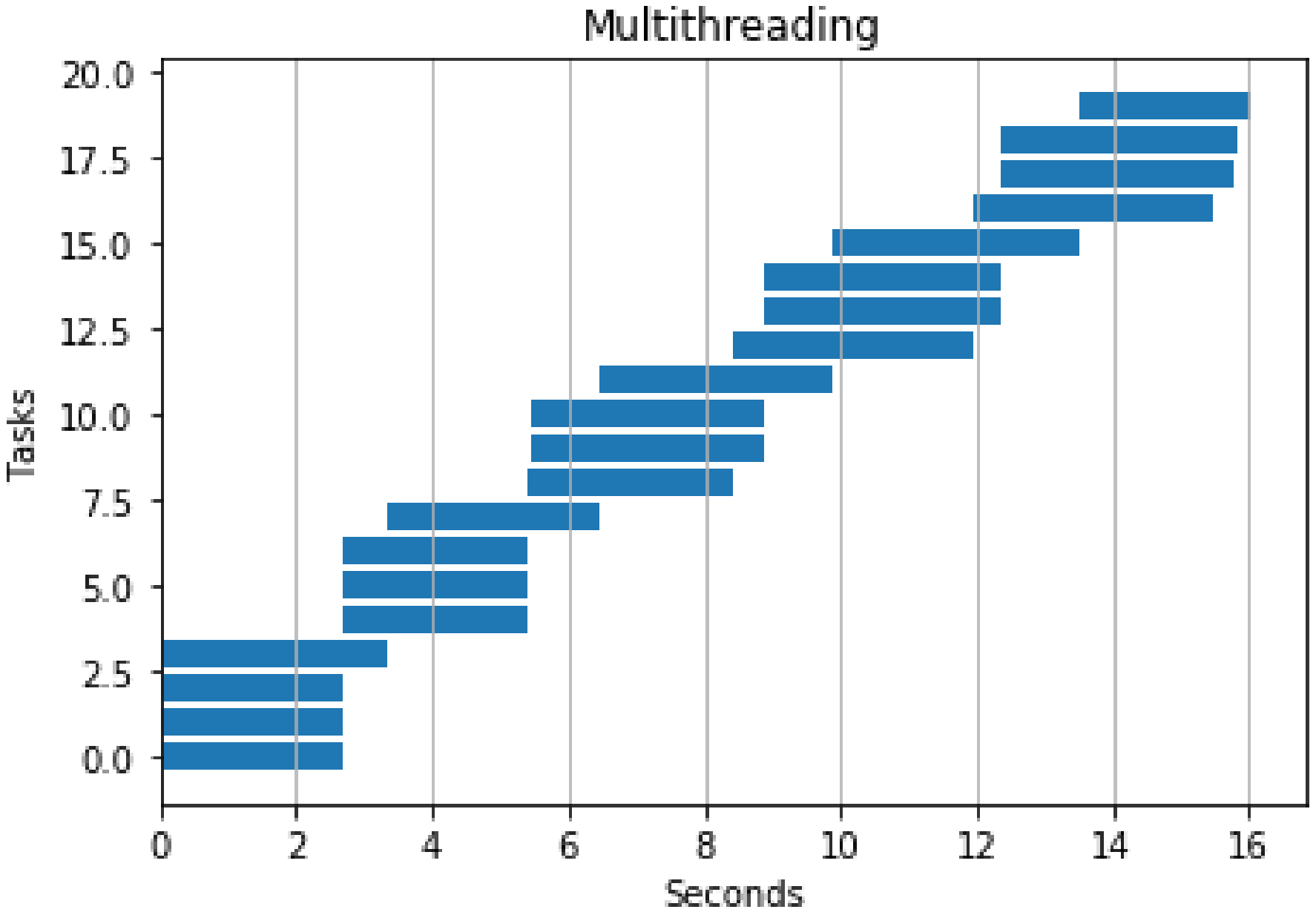
MULTIPROCESSING MODULE




CPU
CPU
CPU
CPU
CPU
CPU
CPU
CPU

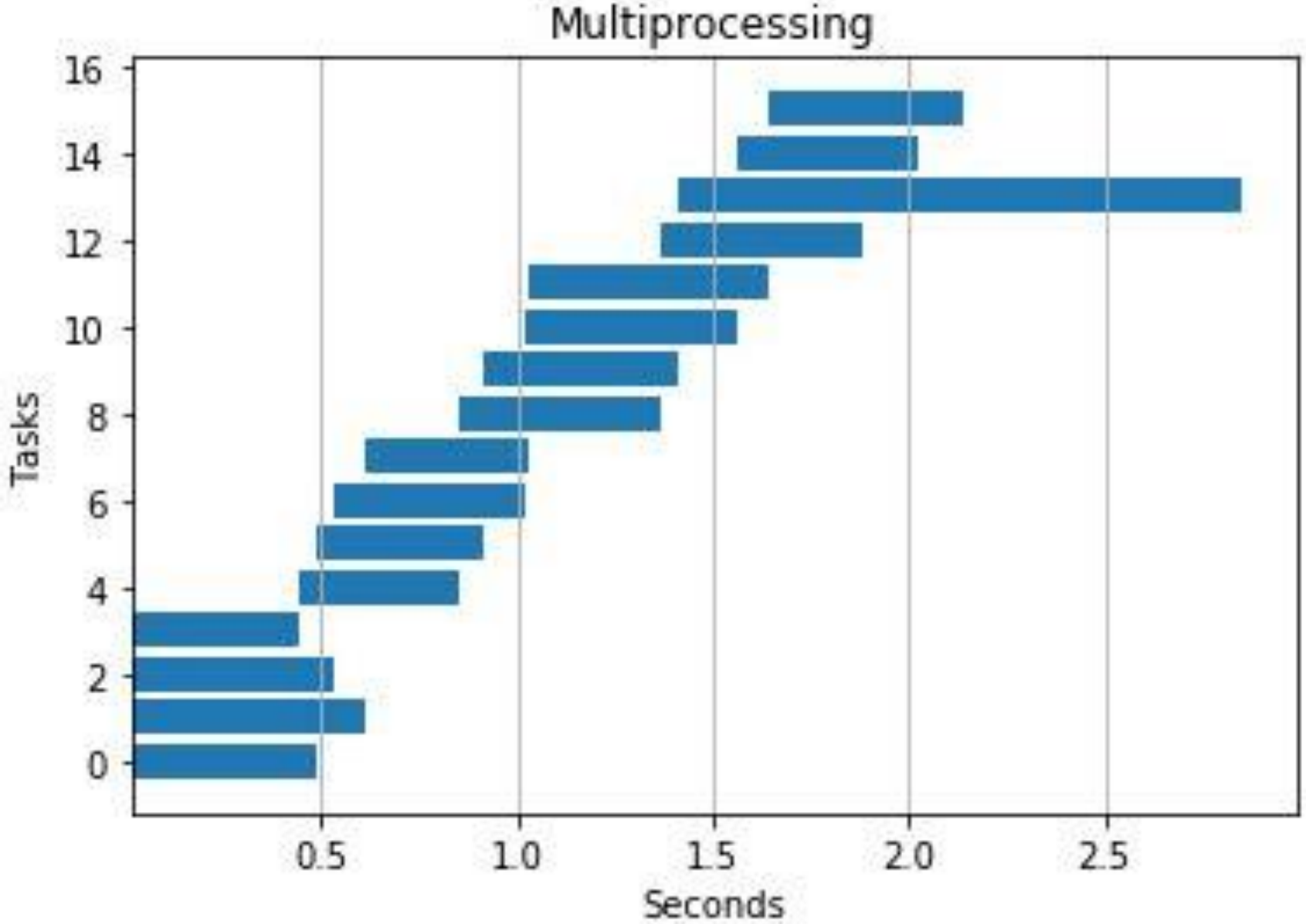
Introduction

- The Python standard library comes with "multiprocessing", a module that gives the feeling of working with threads, but that actually works with processes.
- The "multiprocessing" module is designed to look and feel like the "threading" module.
- A function can be started as a new process, with full concurrency and take advantage of multiple cores, with multiprocessing. It works very much like a thread, including the use of start and join on the Process objects you create.


Multithreading
Tasks
Seconds
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
2
4
6
8
10
12
14
16

Threading

```
import threading
import time
import random
def hello(n):
    time.sleep(random.randint(1,3))
    print("[{0}] Hello!".format(n))
threads = [ ]
for i in range(10):
    t = threading.Thread(target=hello, args=(i,))
    threads.append(t)
    t.start()
for one_thread in threads:
    one_thread.join()
print("Done!")
```


Multiprocessing
Tasks
16
14
12
10
8
6
4
2
0
0.5
1.0
1.5
2.0
2.5
Seconds

Multiprocessing

```
import multiprocessing
import time
import random
def hello(n):
    time.sleep(random.randint(1,3))
    print("[{0}] Hello!".format(n))
processes = [ ]
for i in range(10):
    t = multiprocessing.Process(target=hello, args=(i,))
    processes.append(t)
    t.start()
for one_process in processes:
    one_process.join()
print("Done!")
```

Process vs. Thread

- Perhaps the biggest difference, at least to anyone programming with threads and processes, is the fact that threads share global variables.
- By contrast, separate processes are completely separate; one process cannot affect another's variables.
- In multiprocessing, processes are spawned by creating a Process object and then calling its start() method. Process follows the API of threading.Thread.

Process Class

- Python multiprocessing Process class is an abstraction that sets up another Python process, provides it to run code and a way for the parent application to control execution.
- There are two important functions that belongs to the Process class – start() and join() function.
- Process class works better when processes are small in number and IO operations are long.

Queue Class

- Python Multiprocessing modules provides Queue class that is exactly a First-In-First-Out data structure.
- They can store any pickle Python object (though simple ones are best) and are extremely useful for sharing data between processes.
- Queues are specially useful when passed as a parameter to a Process' target function to enable the Process to consume data.

Lock Class

- The task of Lock class is quite simple. It allows code to claim lock so that no other process can execute the similar code until the lock has be released.

- So the task of Lock class is mainly two. One is to claim lock and other is to release the lock.

Pool

- Python multiprocessing Pool can be used for parallel execution of a function across multiple input values, distributing the input data across processes (data parallelism).

- The Pool class represents a pool of worker processes. It has methods which allows tasks to be offloaded to the worker processes in a few different ways.

Pipe

- The Pipe() function returns a pair of connection objects connected by a pipe which by default is duplex (two-way).
- The two connection objects returned by Pipe() represent the two ends of the pipe. Each connection object has send() and recv() methods
- Pool class works better when there are more processes and small IO wait.

Tutorial Links

https://www.tutorialspoint.com/concurrency_in_python/concurrency_in_python_introduction.htm