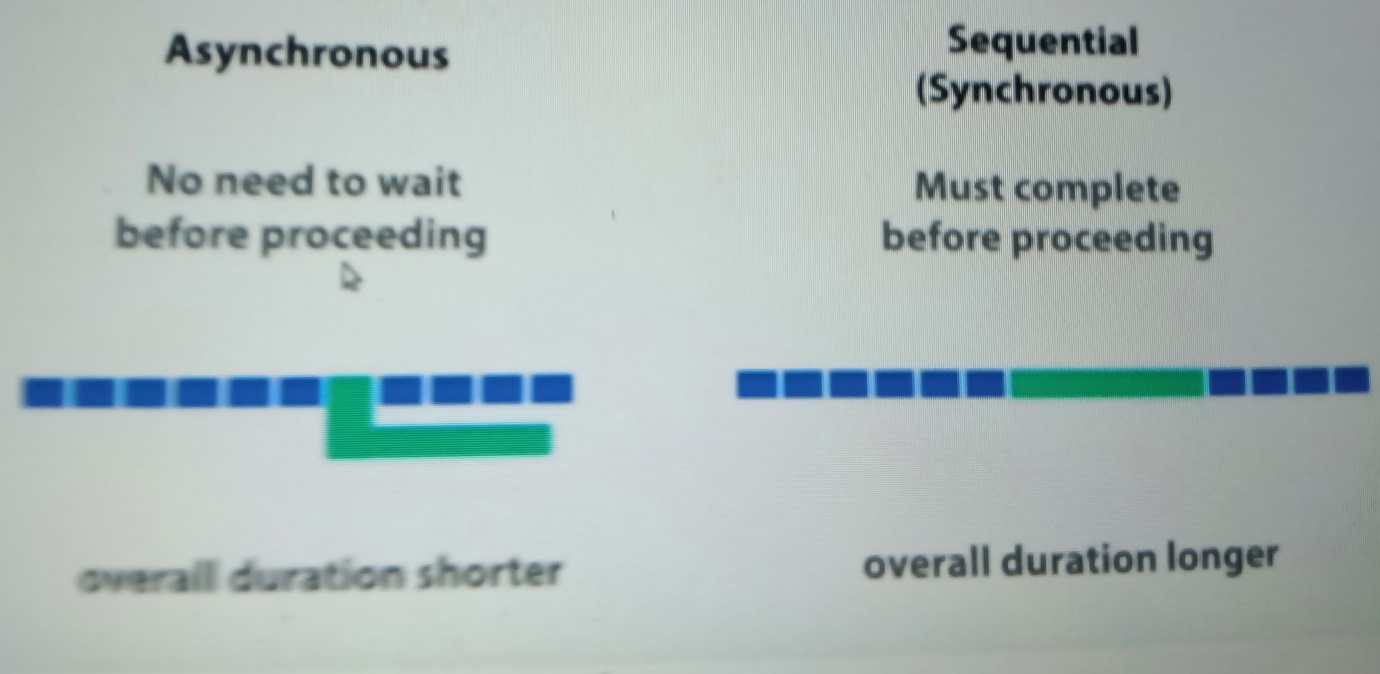
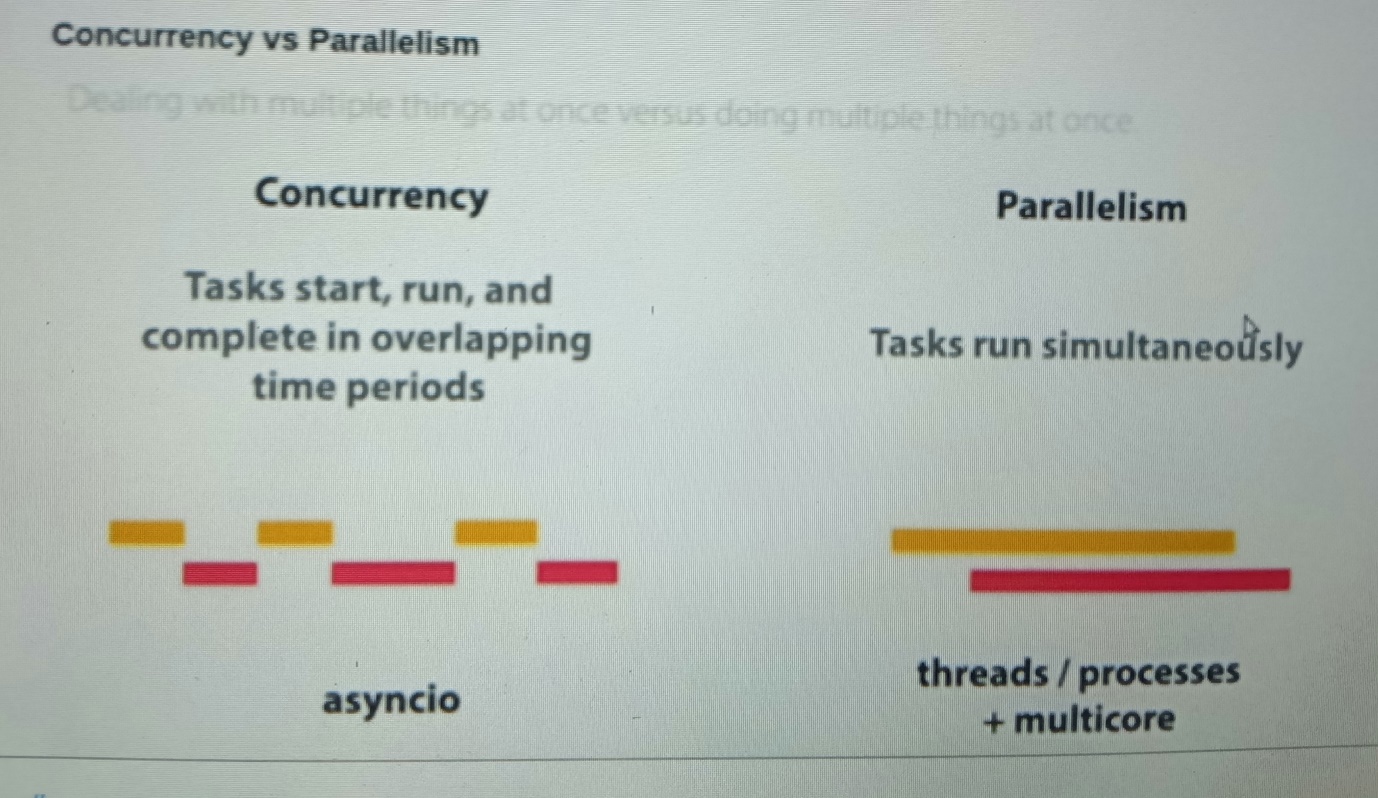
Async:

* Concurrency is like having two threads running on a single core CPU.
* Instructions from each thread could be interleaved, but at any given time,

only one of the two threads is actively making progress.

* Parallelism is like having two threads running simultaneously on different cores of a multi-core CPU.





**Synchronous:**

- Runs functions one after another

**Asynchronous: asyncio**

* runs multiple functions in parallel without threads.
* functions that regularly suspended by awaiting something
* should not use blocking functions like time.sleep()
* Fire off some task, and decide that while we don’t have the result of that task means do someother work instead of waiting.

**Asynchronous programming?**

* Improves performance & utilization of machine resources effectively.

**The Event Loop**

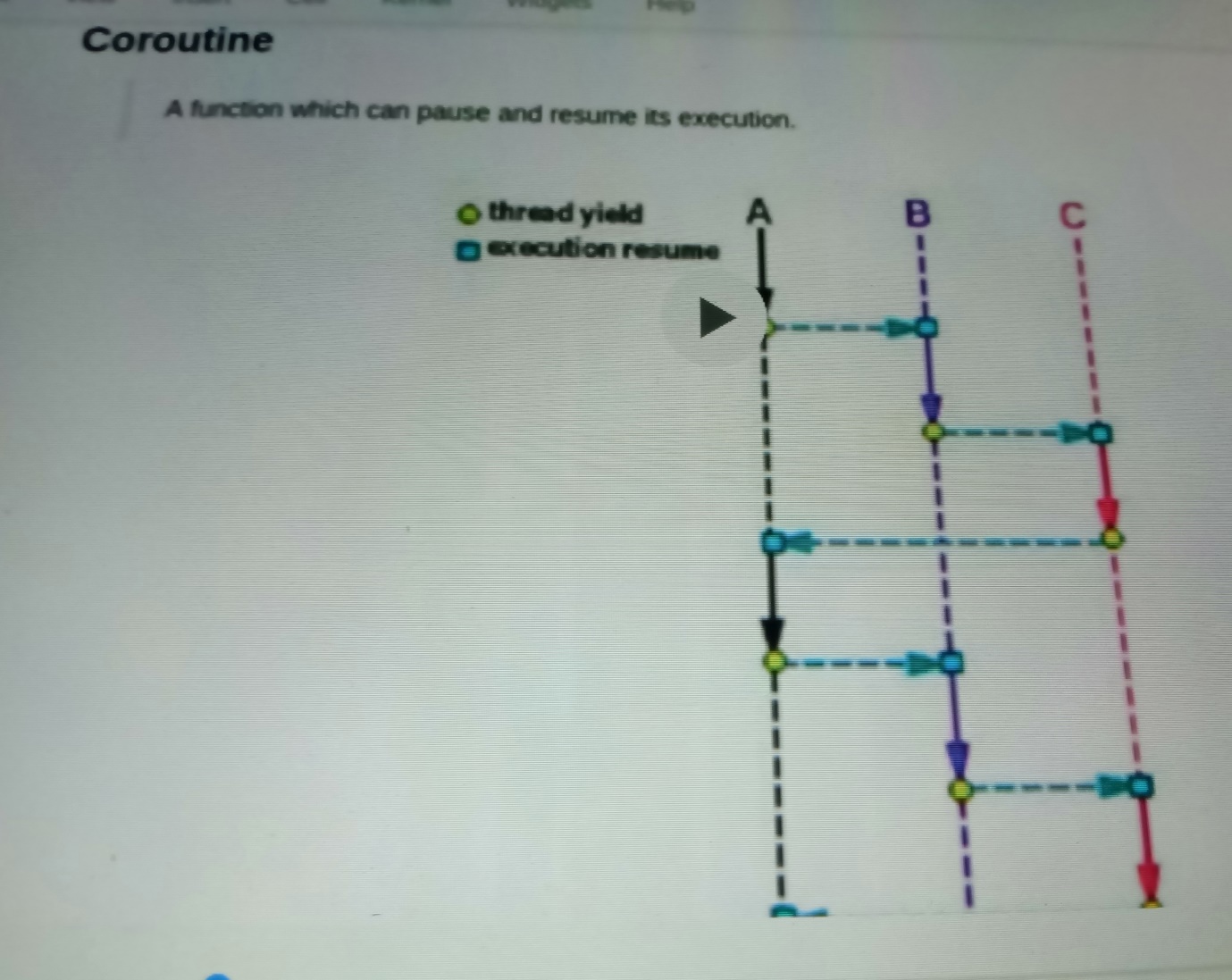
* The event loop is the orchestrator of the symphony.
* It runs tasks one after the other.
* At any given time, only one of the tasks is running.
* The event loop time is precious.
* If you are not making progress, you should step off the loop, so that someone else can.
* Event loop is the measure of progress.

**The Coroutine & Task**

* Coroutines are a key element of the symphony which are basically functions.
* It is the coroutines, and their co-operative nature, that enables giving up

**Asyncio :** Asyncio is all about writing asynchronous programs in Python.

Asyncio is a beautiful symphony between an Event loop, Tasks and Coroutines all coming together so perfectly .



* A coroutine is a stateful generalisation of the concept of subroutine(Method)
* A coroutine, on the other hand, is a cute little stateful widget.
* It looks like a subroutine, but it maintains state in between executions.
* when a coroutine “returns” (yields control) it simply means that it has paused its execution (with some saved state).
* Coroutines look like a normal function, but in their behaviour they are stateful objects with resume() and pause() — like methods.

async and await is supported only from python 3.5

* In Python 3.5+, the way a coroutine pauses itself is using the await keyword.
* Inside a coroutine, when await on another coroutine, we step off the
* event loop and schedule the awaited coroutine to run immediately.
* That is, an await other\_coroutine inside a coroutine will pause it,

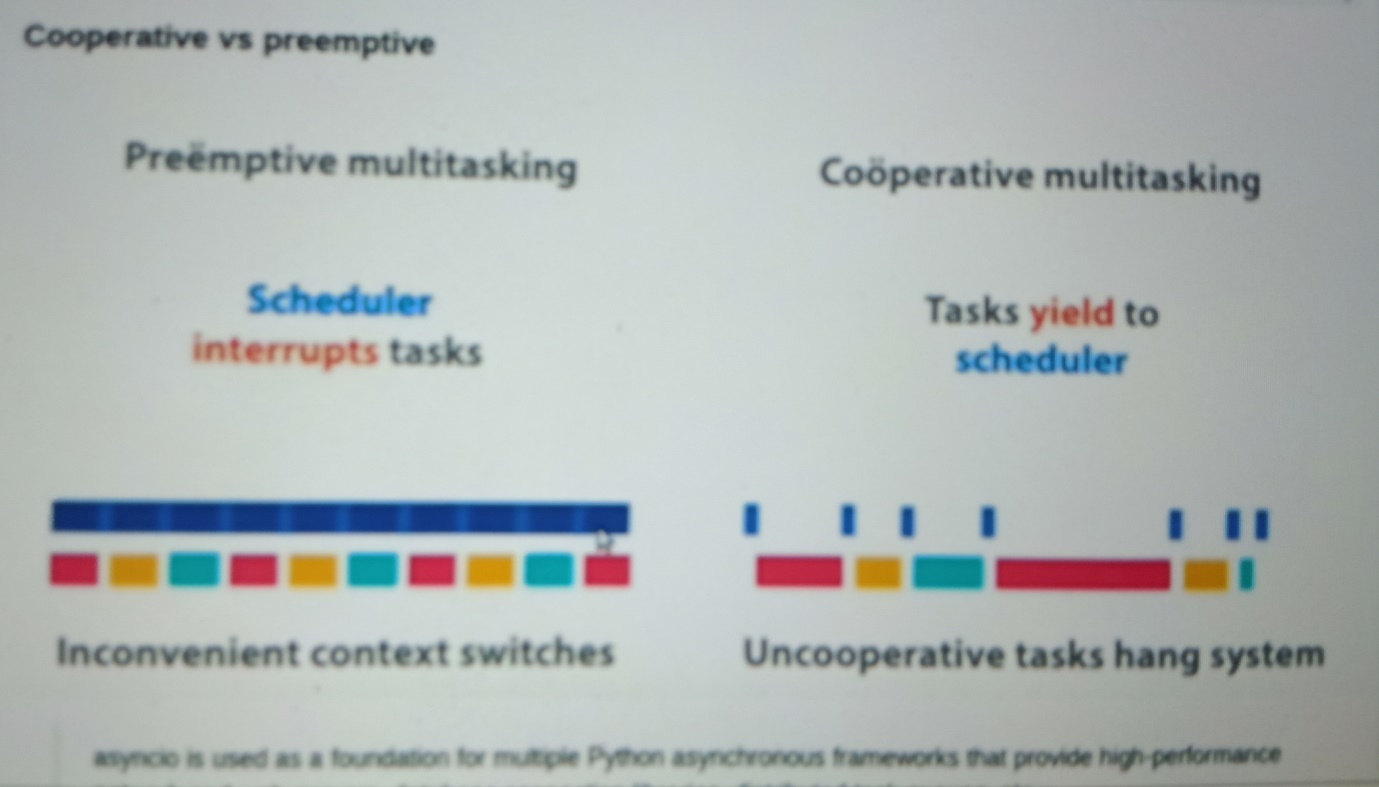
and schedule the coroutine other\_coroutine to run immediately.

* Note that the code runs in a single thread and yet, the output will

have interleaved print statements.

* This happens because when a coroutine gets blocked,

it steps off the loop, so that the other one can run



-------------------------------------------------------