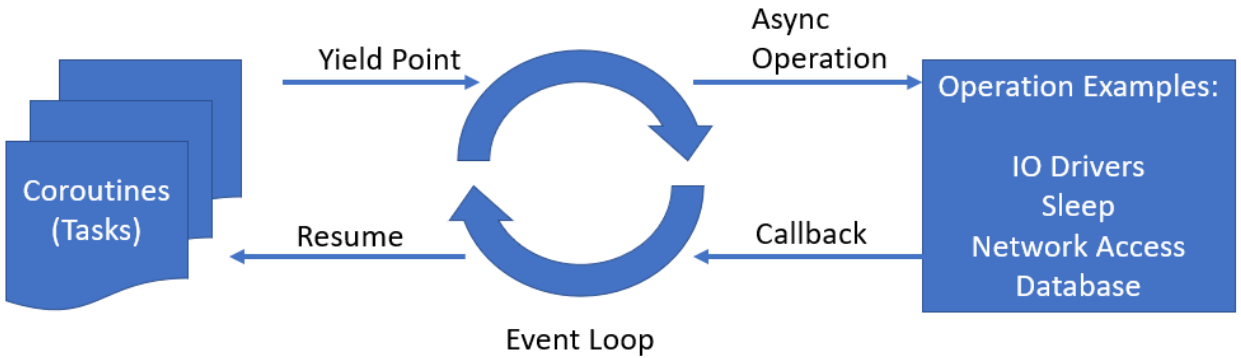
Embedded System Software

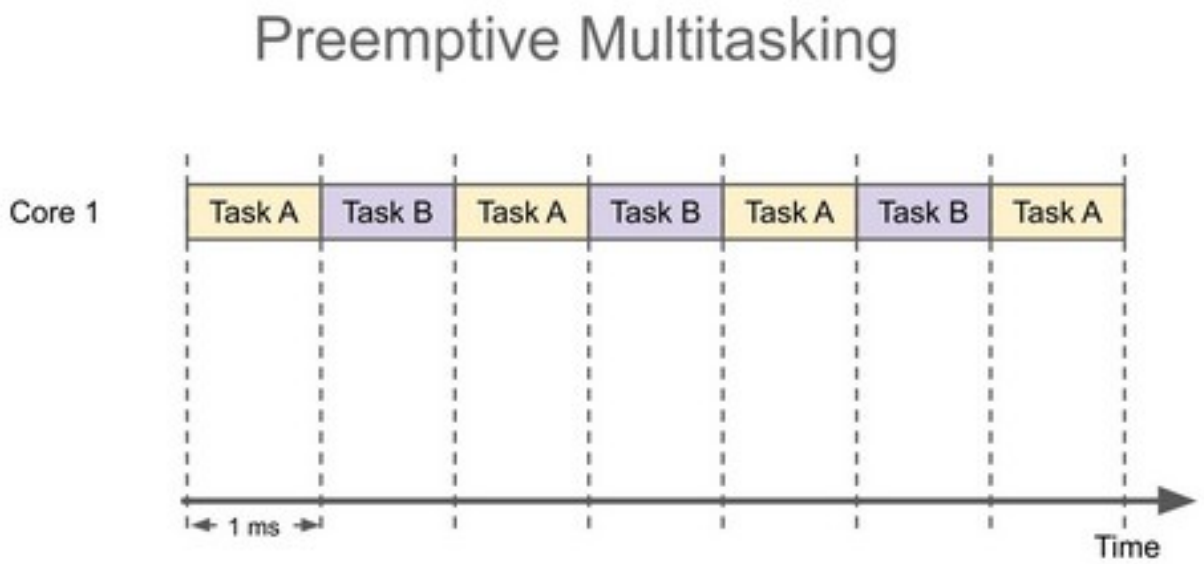
Lecture 14 : Asyncio in Micropython

Asyncio

Asyncio is a Python library that allows us to create cooperative multitasking programs

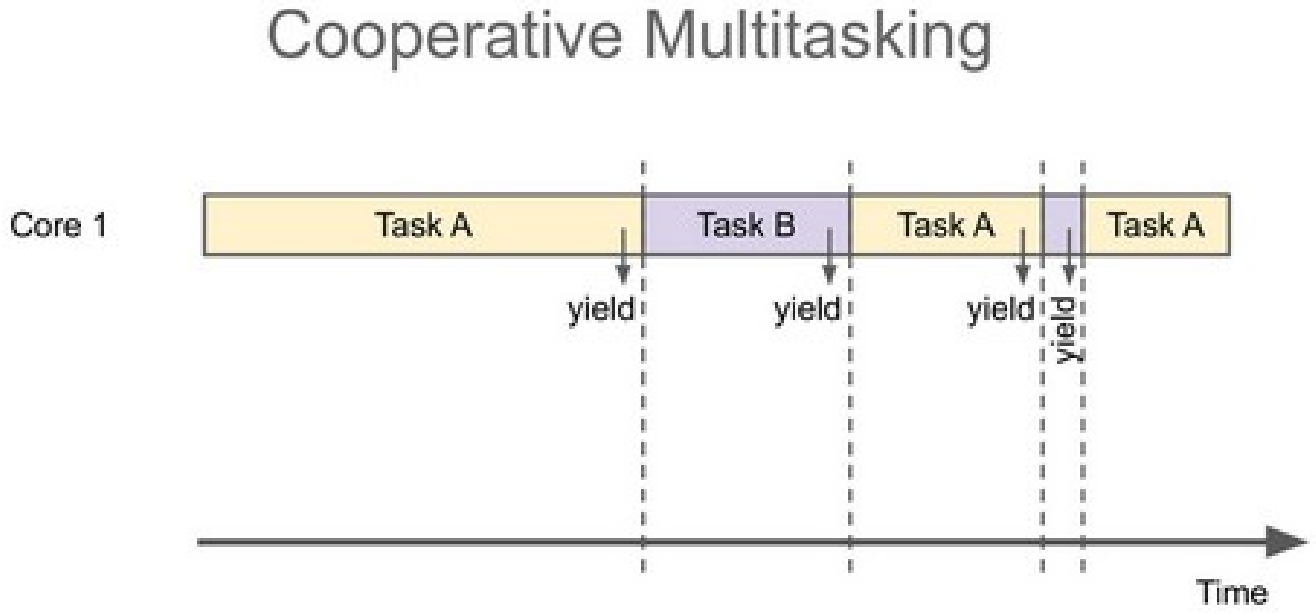


Preemptive vs Cooperative



Full preemptive multitasking is possible in Python with the **_thread** library.

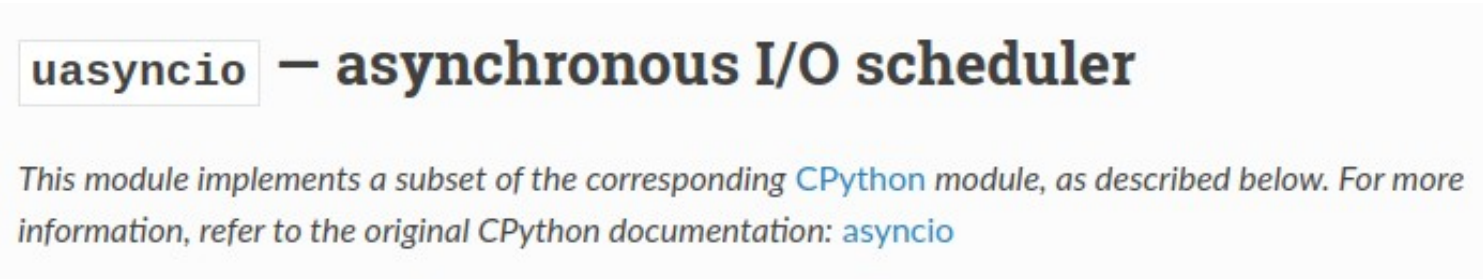
Preemptive vs Cooperative



asyncio library

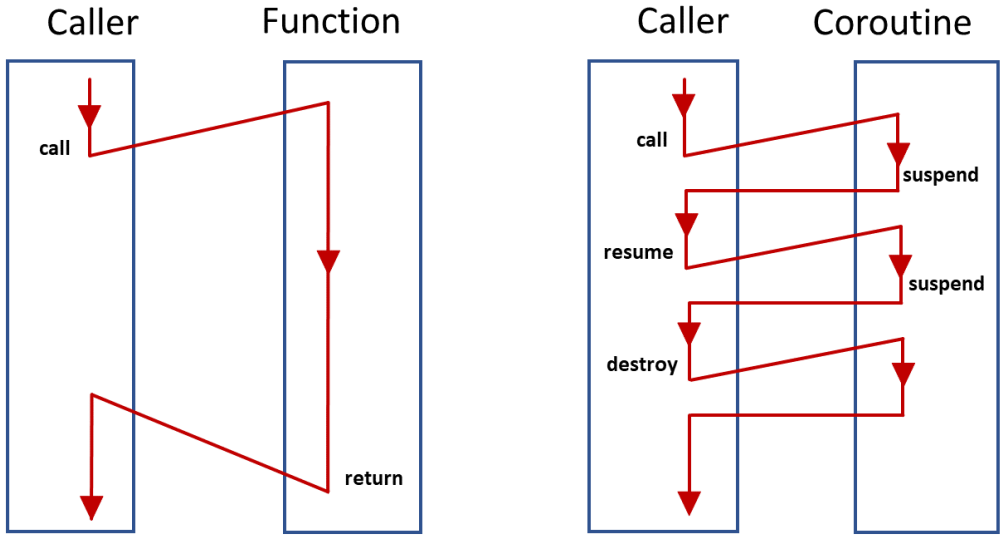
uasyncio

MicroPython implements a version of asyncio called uasyncio that contains a subset of the functions available in the full asyncio library.



Coroutine?

- A **coroutine** is a function that is capable of **pausing its own execution** to allow other code to run
- Different from a normal function.



async and await

- New keywords for python >= 3.5. For asyncio
- **async** -> For creating a coroutine function
- **await** -> For yielding

```
async def g():
    # Pause here and come back to g() when f() is ready
    r = await f()
    return r
```

Core functions

Core functions

uasyncio.create_task(*coro*)

Create a new task from the given coroutine and schedule it to run.

Returns the corresponding `Task` object.

uasyncio.current_task()

Return the `Task` object associated with the currently running task.

uasyncio.run(*coro*)

Create a new task from the given coroutine and run it until it completes.

Returns the value returned by *coro*.

uasyncio.sleep(*t*)

Sleep for *t* seconds (can be a float).

This is a coroutine.

uasyncio.sleep_ms(*t*)

Sleep for *t* milliseconds.

This is a coroutine, and a MicroPython extension.

Lab : Blinking Example

```
# Coroutine: blink
async def blink(period, led):
    while True:
        led.toggle()
        await uasyncio.sleep(period)


async def main():
    global led_red
    global led_green
    uasyncio.create_task(blink(1, led_red))
    uasyncio.create_task(blink(0.25, led_green))
    await uasyncio.sleep(10)

uasyncio.run(main())
```

Main will run until 10 seconds

Lab : Blink + Button Example

```
# Coroutine: Button
async def read_button(btn):
    while btn.value() == 1:
        await uasyncio.sleep(0.02)

# Coroutine: blink
async def blink(led, btn):
    while True:
        await read_button(btn)
        led.value(0)
        await uasyncio.sleep(0.01)

async def main():
    global led_red
    global led_green
    uasyncio.create_task(blink(led_red, btn))
    await uasyncio.sleep(10)

led_red.value(1)
uasyncio.run(main())
```

LED will turn on after button is pressed

Pro/Con

Pro

- Optimized for I/O bound tasks.
- Light-weight
- Less bug, No race condition

Con

- Hard to program, Need to understand when to yield by yourself.
- Concept is hard to understand.