INTRO TO CONCURRENCY

Created by Bahram Aghaei

COMMON I/O OPERATIONS IN WEB APPLICATIONS:

- Downloading the content of a web page
- Communicating over a network
- Running several queries against a database

But, the I/O operations tend to be **slow** for different reasons.

WHAT WE'LL COVER

- What is `asyncio`?
- Difference between CPU-bound vs. I/O-bound
- Concurrency, Parallelism, and Multitasking
- Global Interpreter Lock (GIL)
- Non-blocking I/O + Event Loop

WHAT IS ASYNCIO?

In a synchronous app, code runs sequentially, waiting for each task to finish.

Concurrency allows multiple tasks to run simultaneously, keeping the app responsive.

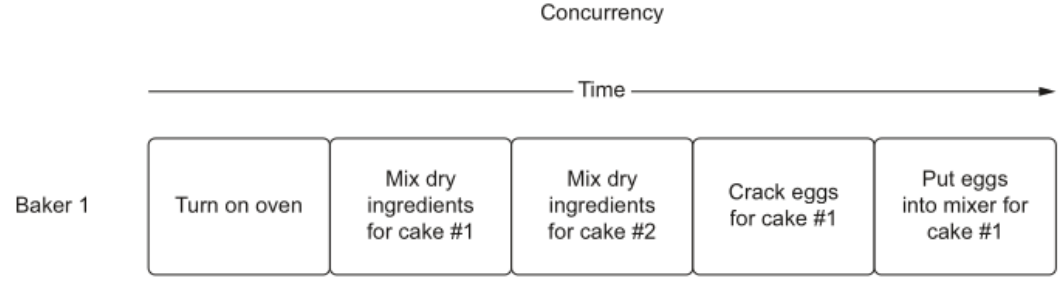
Asynchronous programming lets tasks run in the background, freeing up the system to do other work.

`asyncio` is a Python library for managing tasks asynchronously using a **single-threaded event loop**.

CPU-BOUND
VS
I/O-BOUND

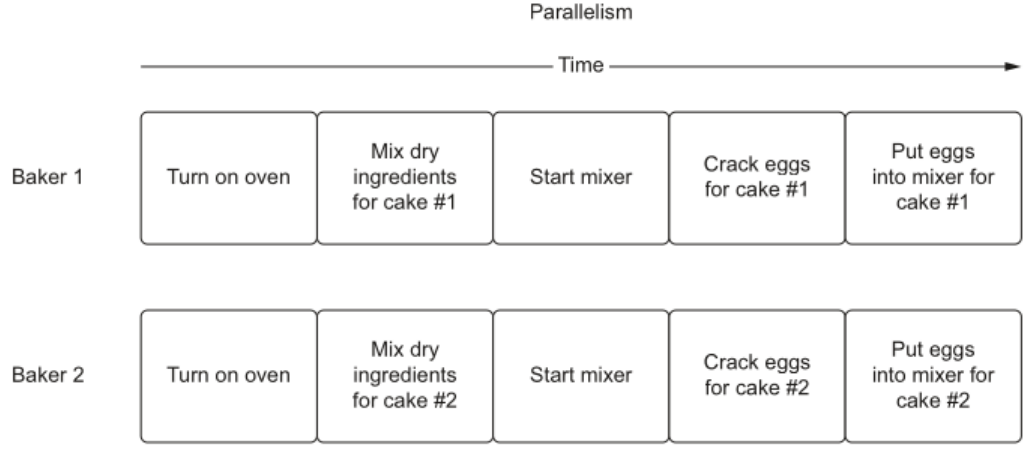
CONCURRENCY

When we say two tasks are happening concurrently, we mean those tasks are happening at the same time.



PARALLELISM

When we say something is running in **parallel**, we mean not only are there **two or more tasks** happening concurrently, but they are also **executing at the same time**.



MULTITASKING

- Preemptive Multitasking
- Cooperative Multitasking

PREEMPTIVE MULTITASKING

In this model, the operating system decides how to switch between tasks using time slicing, known as **preempting**.

COOPERATIVE MULTITASKING

In this model, the program itself defines points where other tasks can run.

“I’m pausing my task; go ahead and run other tasks.”

- **Less resource intensive**
- **More granularity**

GLOBAL INTERPRETER LOCK (GIL)

GIL prevents one Python process from executing more than one Python bytecode instruction at any given time.

Why does the GIL exist?

- due to how memory is managed in CPython.
- CPython is not thread safe.

Takeaway: For CPU-bound tasks, multithreading is not beneficial.

GIL (CONTINUED)

Is the GIL ever released?

- The GIL is released when I/O operations occur.

Why release the GIL for I/O but not CPU-bound tasks?

- I/O operations use system calls outside of the Python runtime, allowing the GIL to be released.
- CPU-bound tasks do not benefit from GIL release because they run within Python.

GIL (CONTINUED)

Using asyncio with GIL:

- asyncio uses **coroutines**, which are lightweight threads.
- **Important:** asyncio does not circumvent the GIL; it operates within its constraints.

NON-BLOCKING I/O + EVENT LOOP

Sockets are blocking by default.

```python
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Connect the socket to the server's address and port
server_address = ('localhost', 8080)
sock.connect(server_address)

# Blocking read from the socket
data = sock.recv(1024)  # Blocking call
print(f"Received: {data.decode()}")
```

SOLUTION 🤔

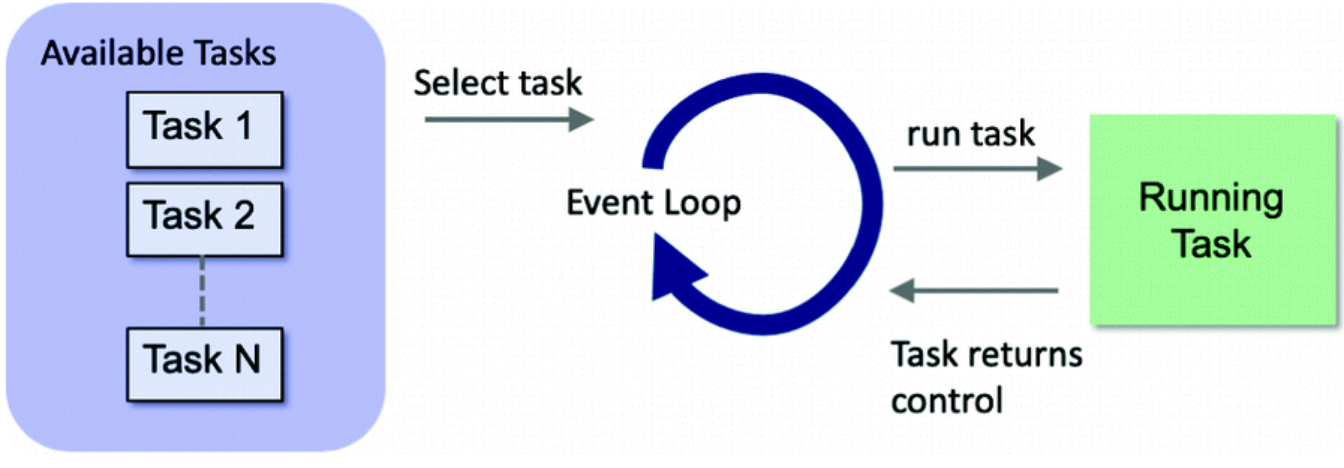
At the OS level, we can operate in non-blocking mode using **event notification systems**.

- **kqueue** — FreeBSD and MacOS
- **epoll** — Linux
- **IOCP (I/O completion port)** — Windows

THE EVENT LOOP: MANAGING I/O TASKS

How do we track which tasks are waiting for I/O versus those that are regular Python code?

an event loop



BASIC EVENT LOOP EXAMPLE

```
1  from collections import deque
2  messages = deque()
3
4  while True:
5    if messages:
6      message = messages.pop()
7      process_message(message)
8
9  def make_request():
10   cpu_bound_setup()
11   io_bound_web_request()
12   cpu_bound_postprocess()
13
14 task_one = make_request()
15 task_two = make_request()
```