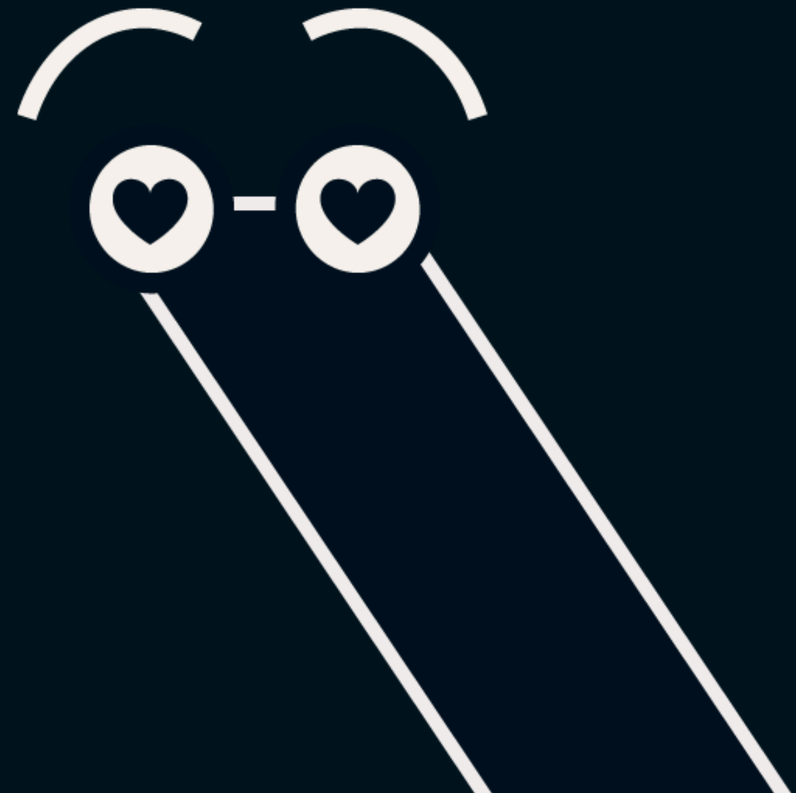
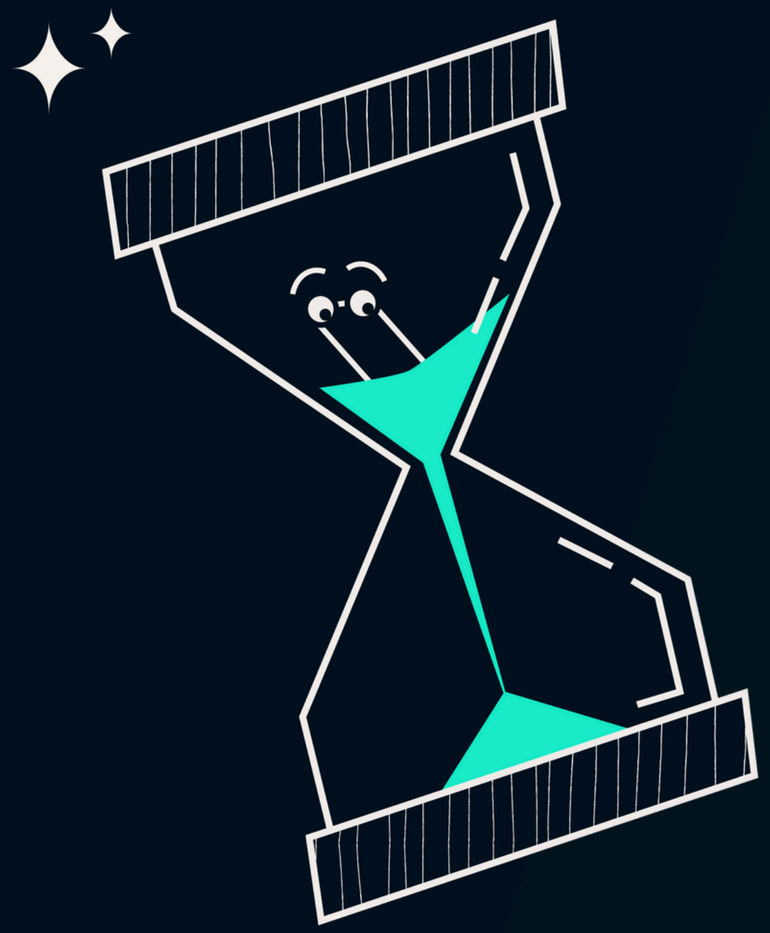
xtream

The Hitchhiker's Guide to Asyncio

Emanuele Fabbiani

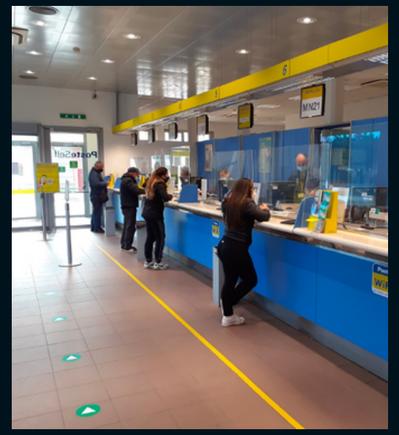




CPU Cycle	1s
L1 Cache	3s
RAM	4 mins
Network	7.6 years


McDonald's
McDonald's
DRIVETHRU


6
5
MN21



Preemptive multitasking

`threading`

Single Core *

Cooperative multitasking

`asyncio`

Single Core

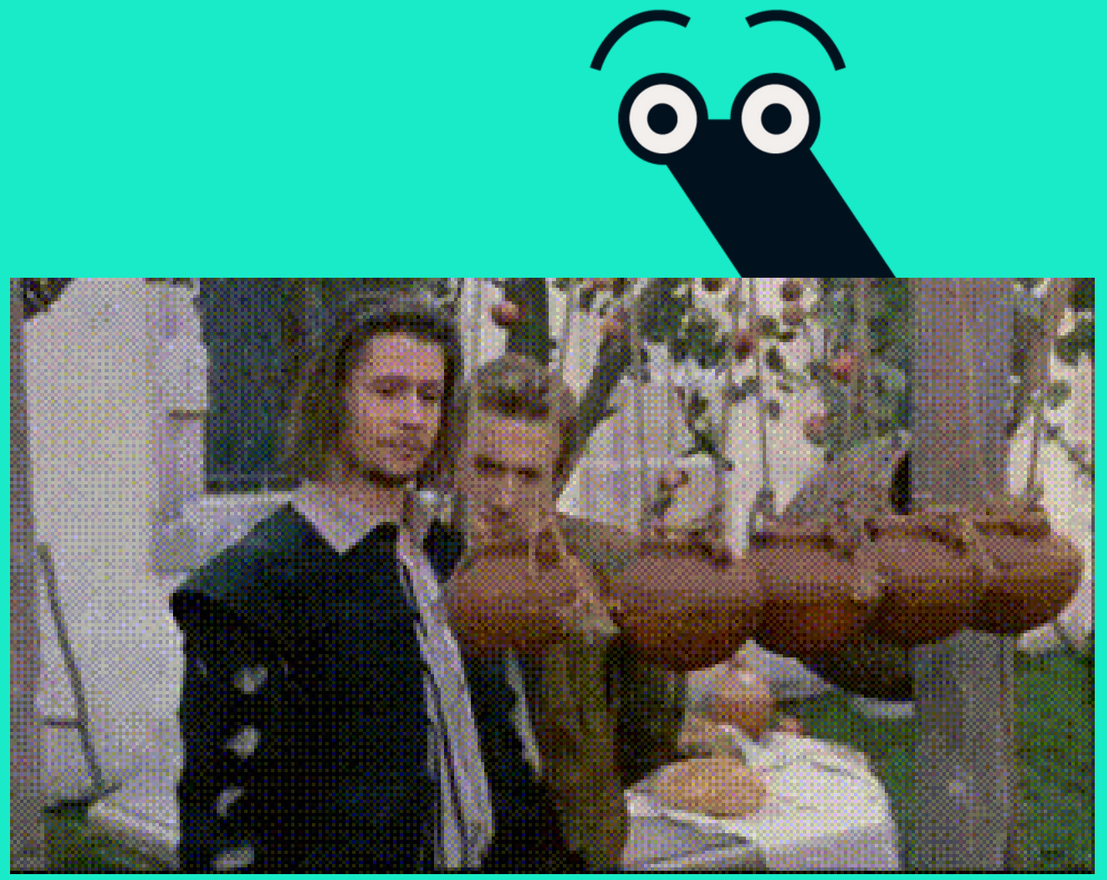
Multiprocessing

`multiprocessing`

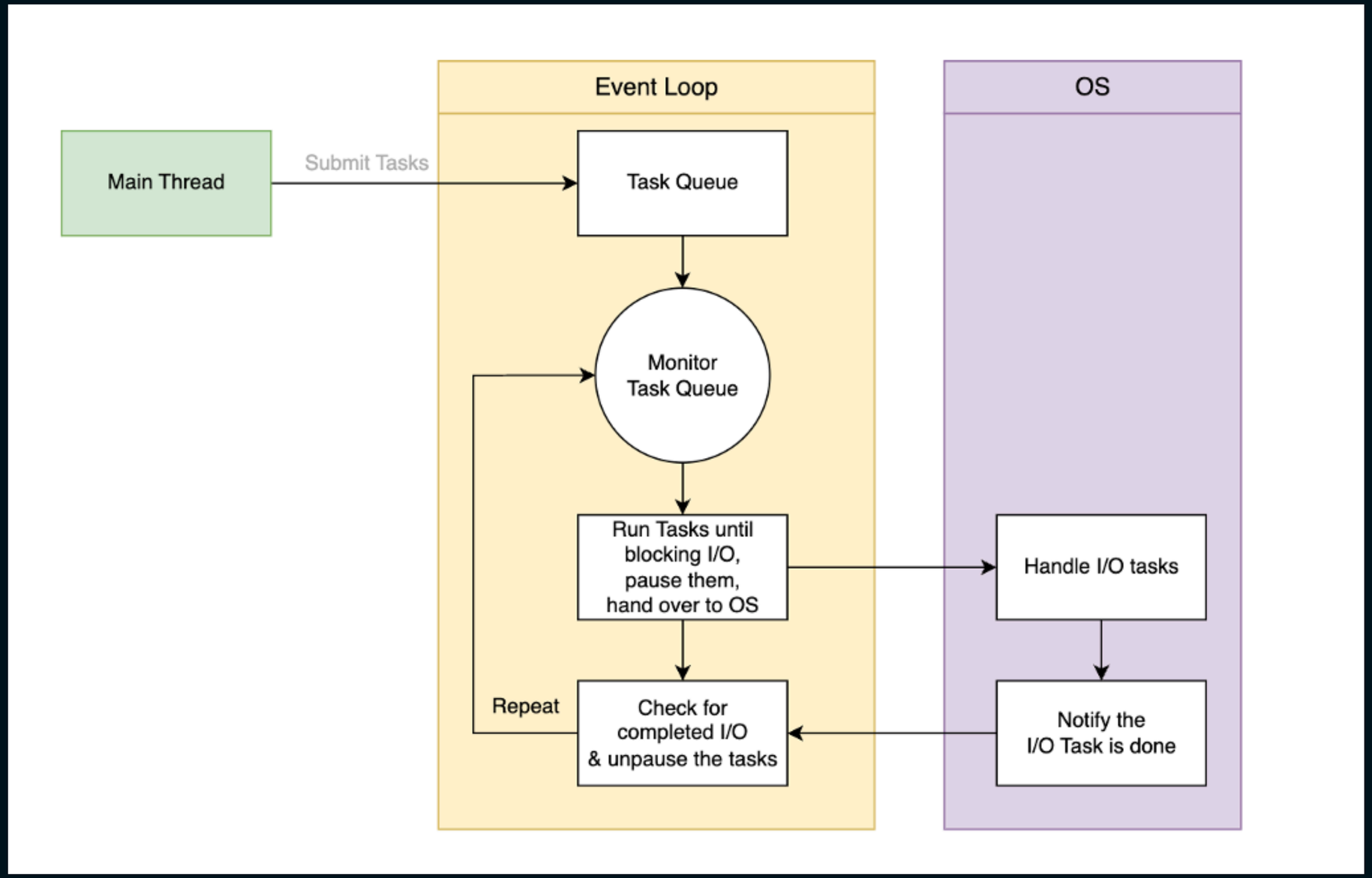
Many Cores

*Optional free threading was introduced in Python 3.13

Live Coding



How Does it Work ?


Event Loop
OS
Main Thread
Submit Tasks
Task Queue
Monitor
Task Queue
Run Tasks until
blocking I/O,
pause them,
hand over to OS
Handle I/O tasks
Repeat
Check for
completed I/O
& unpause the tasks
Notify the
I/O Task is done

- Python 2.2 -> Generators and yield
- Python 2.5 -> Generators support .send()
- Python 3.3 -> yield from
- Python 3.4 -> asyncio with @asyncio.coroutine
- Python 3.5 -> async / await

...

```python
from typing import Generator

def jumping_range(up_to: int) -> Generator[int, int, None]:
    index, jump = 0, 0
    while index < up_to:
        jump = yield index
        if jump is None:
            jump = 1
        index += jump

if __name__ == '__main__':
    iterator = jumping_range(5)
    print(next(iterator))    # 0
    print(iterator.send(2))  # 2
    print(next(iterator))    # 3
    print(iterator.send(-1)) # 2
```

```python
# Python 3.4
import asyncio

@asyncio.coroutine
def countdown(number, n):
    while n > 0:
        print('T-minus', n, '({})'.format(number))
        yield from asyncio.sleep(1)
        n -= 1

if __name__ == '__main__':
 loop = asyncio.get_event_loop()
 tasks = [
     asyncio.ensure_future(countdown("A", 2)),
     asyncio.ensure_future(countdown("B", 3))
 ]
 loop.run_until_complete(asyncio.wait(tasks))
 loop.close()
```

```python
@asyncio.coroutine
def py34_coro():
    yield from stuff()
```

```
>>> dis.dis(py34_coro)
  2           0 LOAD_GLOBAL
              3 CALL_FUNCTION
              6 GET_YIELD_FROM_ITER
              7 LOAD_CONST
             10 YIELD_FROM
             11 POP_TOP
             12 LOAD_CONST
             15 RETURN_VALUE
```

```python
async def py35_coro():
    await stuff()
```

```
>>> dis.dis(py35_coro)
  1           0 LOAD_GLOBAL
              3 CALL_FUNCTION
              6 GET_AWAITABLE
              7 LOAD_CONST
             10 YIELD_FROM
             11 POP_TOP
             12 LOAD_CONST
             15 RETURN_VALUE
```

https://docs.python.org/3/library/concurrency.html

https://docs.python.org/3/library/asyncio.html

https://realpython.com/python-concurrency

https://realpython.com/python-gil

https://realpython.com/async-io-python

https://snarky.ca/how-the-heck-does-async-await-work-in-python-3-5

https://github.com/donlelef/the-hitchhikers-guide-to-asyncio

The End

Ego Slide

Emanuele Fabbiani

AI Engineer @ xtream

Professor in AI at Catholic University of Milan

PhD in Applied AI

Speaker at AMLD Lausanne, ODSC London, PyData Berlin, PyCon Florence, PyData Milan, PyData Paris

Lecturer at UniPV, PoliMI, UniCatt, HSLU, Politechnika Wroclawska.