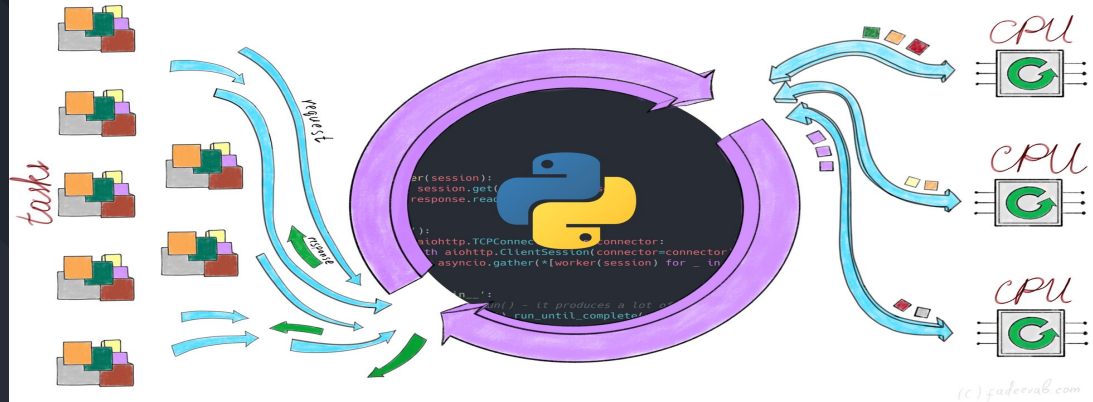


INTRODUCCIÓN A ASYNCIO



SÍNCRONO

```
1 import random
2
3 def buscarElemento(lista, elemento):
4     for i in range(0, len(lista)):
5         if(lista[i] == elemento):
6             return i
7
8 def imprimirLista(lista, nombre):
9     for i in range(0, len(lista)):
10         print nombre + "[" + str(i) + "]= " + str(lista[i])
11
12 def leerLista():
13     lista=[]
14
15     i=0
16     while i < 10:
17         lista.append(int(random.randint(0, 10)))
18         i=i+1
19     return lista
20
21 A=leerLista()
22 imprimirLista(A, "A")
23 cn=int(raw_input("Numero a buscar: "))
24 print "A[" + str(buscarElemento(A, cn)) + "]"
```

ASÍNCRONO

```
In [1]: import asyncio

In [2]: import time

In [3]: async def say_delay(msg, delay):
...:     await asyncio.sleep(delay)
...:     print(msg)
...:

In [4]: async def main():
...:     print(f"begin at {time.strftime('%H:%M:%S')}")
...:     await say_delay('Hello', 1)
...:     await say_delay('learnPython', 2)
...:     print(f"end at {time.strftime('%H:%M:%S')}")
...:

In [5]: asyncio.run(main())
begin at 21:53:51
Hello
learnPython
end at 21:53:54
```

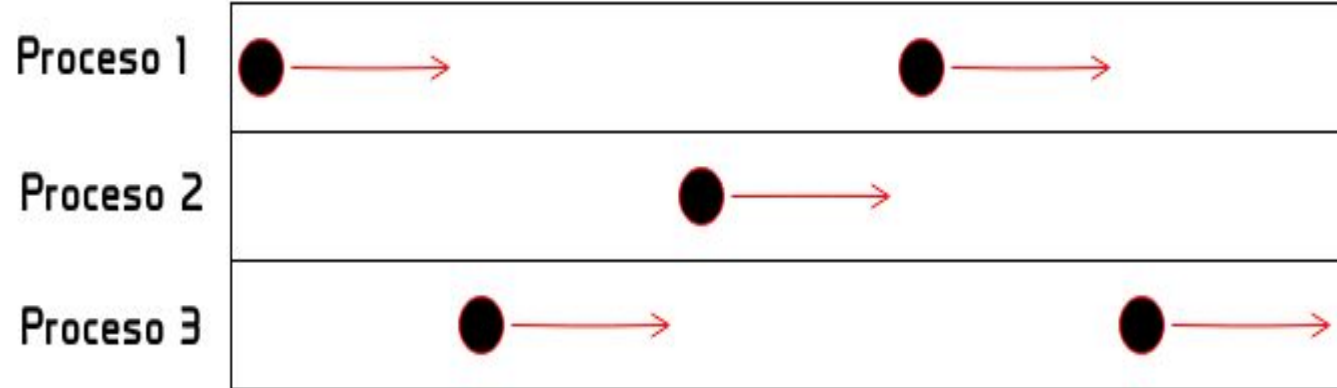
Synchronous

vs.

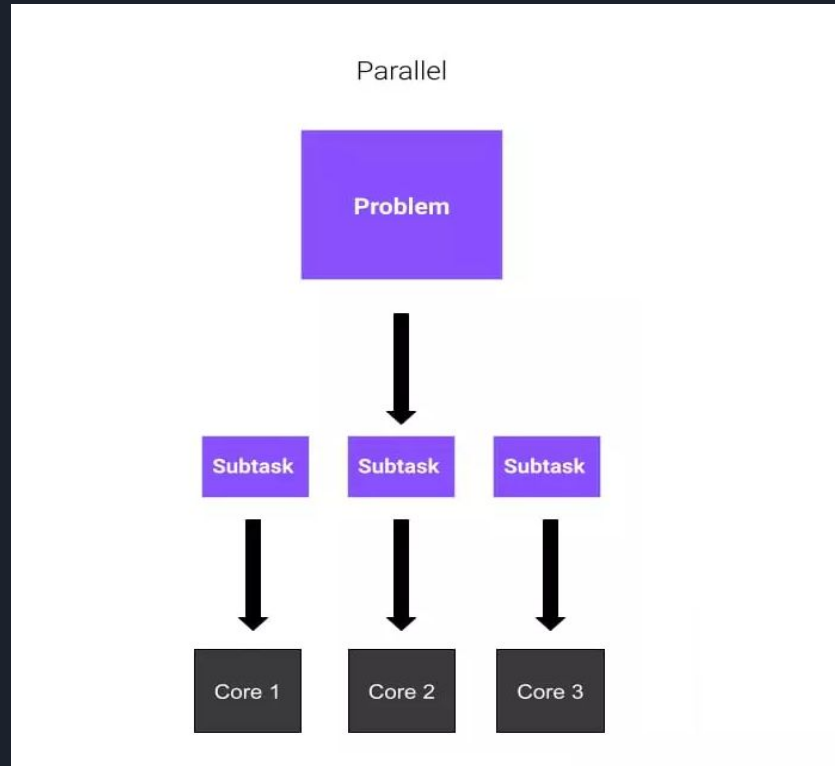
Asynchronous



CONCURRENCIA

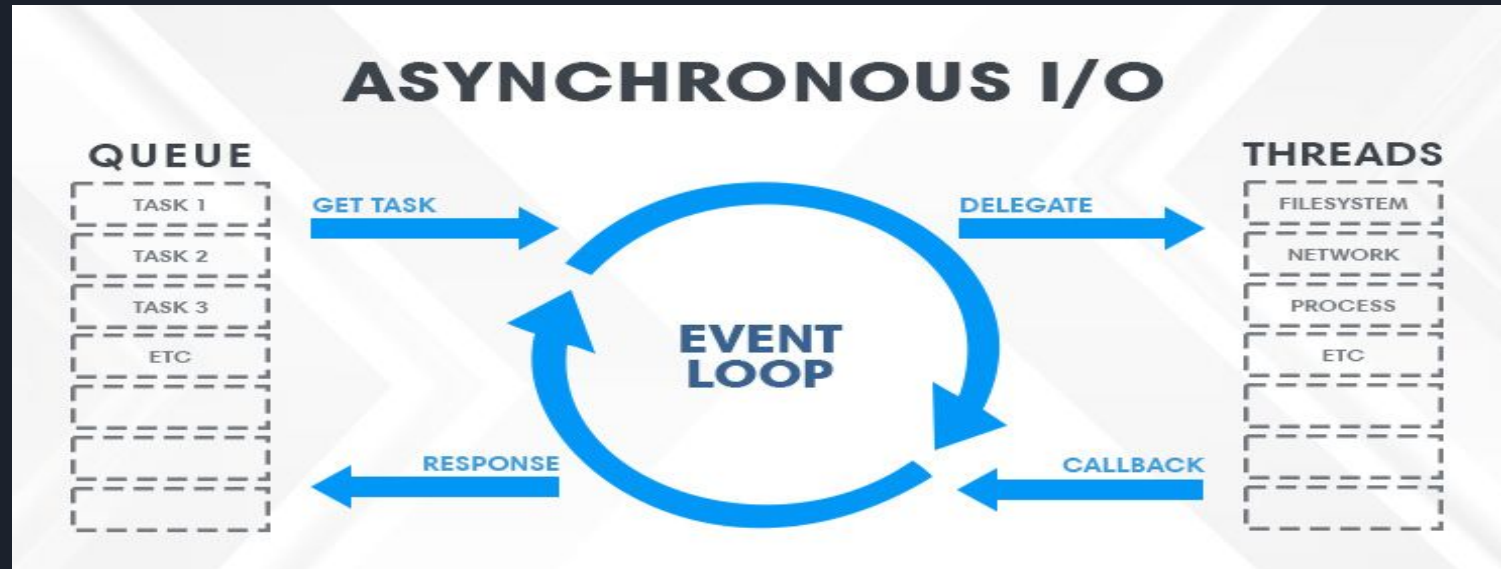



PARALELISMO



AsyncIO

AsyncIO se basa en el bucle de eventos ("event loop") de Python





Permite que las funciones sean definidas como "corutinas"

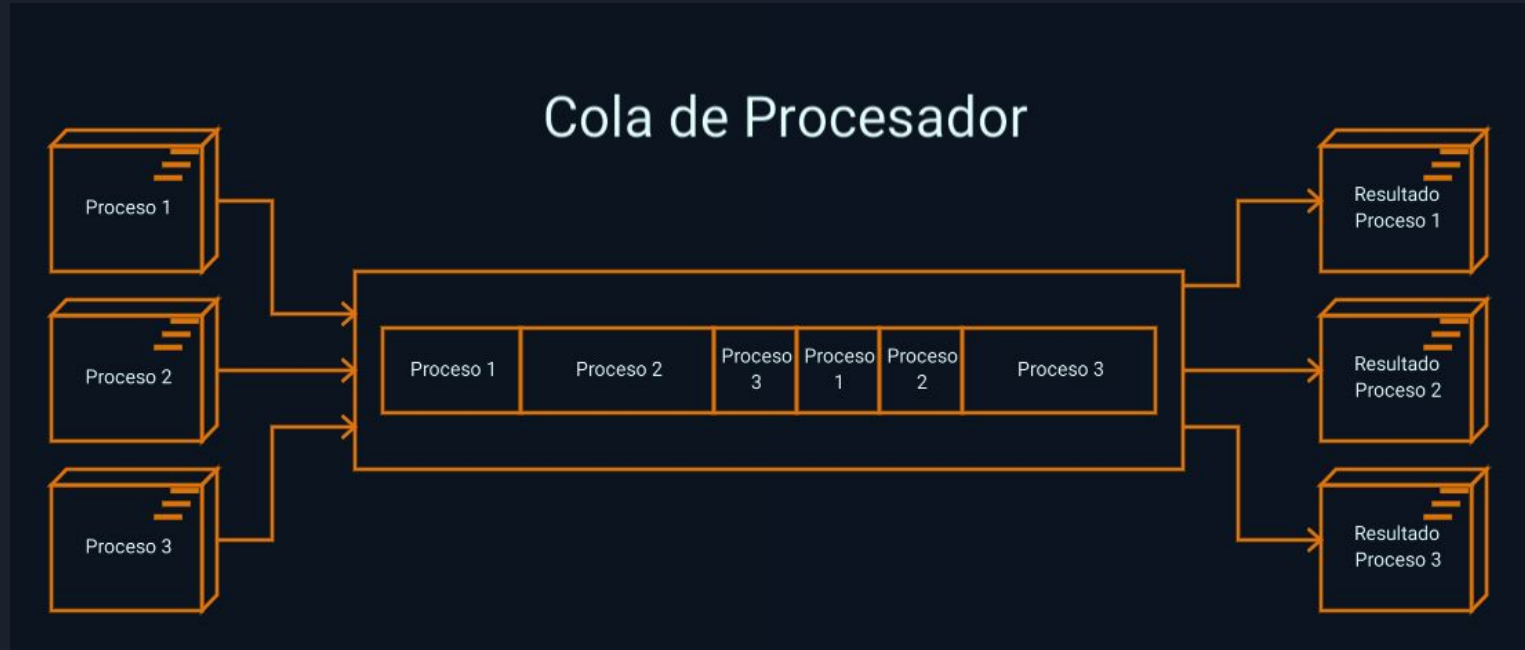
¡Hola Mundo!

```
import asyncio

async def main():
    print('Hello ...')
    await asyncio.sleep(1)
    print('... World!')

asyncio.run(main())
```


Las ventajas de AsyncIO son su facilidad para manejar tareas de manera eficiente y su capacidad para escalar de muchas conexiones concurrentes.





APLICACIONES

Programación de redes

Procesamiento de solicitudes HTTP

Acceso a bases de datos

Procesamiento de tareas por lotes



Ventajas

Programación asíncrona y no bloqueante

```
import asyncio

async def factorial(name, number):
    f = 1
    for i in range(2, number + 1):
        print(f"Task {name}: Compute factorial({number}), currently i={i}...")
        await asyncio.sleep(1)
        f *= i
    print(f"Task {name}: factorial({number}) = {f}")
    return f

async def main():
    # Schedule three calls *concurrently*:
    L = await asyncio.gather(
        factorial("A", 2),
        factorial("B", 3),
        factorial("C", 4),
    )
    print(L)

asyncio.run(main())
```

Mejora del rendimiento





Menos bloqueos y latencia reducida

Lógica más sencilla

Escalabilidad y manejo de concurrencia



GRACIAS



Miguel Santos



@jmiguel_santos



9983151987