Gauchito GIL

Concurrencia en Python

Yonatan Romero

@yonatancony

import me

Concurrencia

Composición de procesos ejecutados independientemente

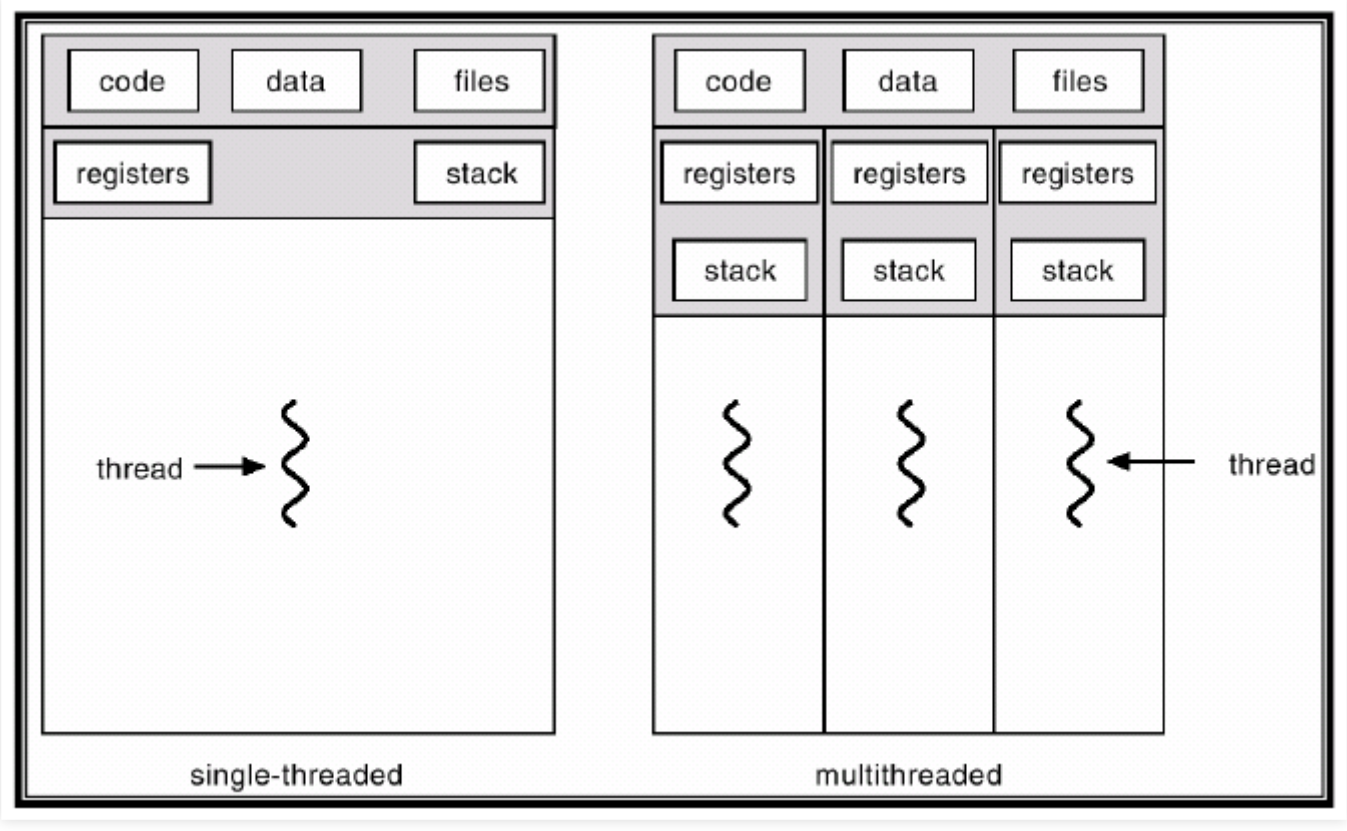
Manejar un montón de cosas que pasan al mismo tiempo

Paralelismo

Ejecución simultánea de muchas cosas

Hacer un montón de cosas al mismo tiempo

Hilos y procesos



En teoría los hilos son **más livianos** que los procesos

Tipos de procesos

CPU-bound

Uso intensivo de CPU

Ejemplo: Cálculos matemáticos, criptografía, etc

IO-bound

Uso intensivo de recursos de entrada y salida

Ejemplo

Nuestra función *CPU-bound*

```
def cpu_bound(repeticiones):
    while repeticiones > 0:
        repeticiones -= 1
```

Secuencial

```
REPETICIONES = 150000000

start = time.time()

cpu_bound(REPETICIONES)

duracion = time.time() - start
print("Tiempo transcurrido {duracion:.2f} segundos".format(
    duracion=duracion
))
```

Multiprocesos

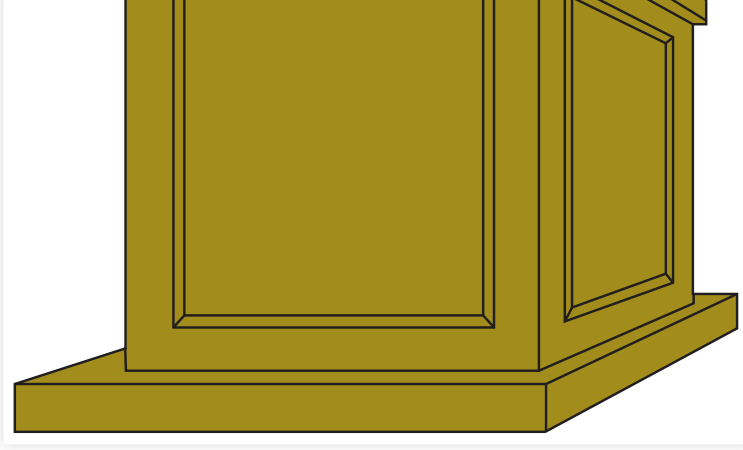
```
from multiprocessing import Process

p1 = Process(target=cpu_bound, args=(REPETICIONES//4,))
p2 = Process(target=cpu_bound, args=(REPETICIONES//4,))
p3 = Process(target=cpu_bound, args=(REPETICIONES//4,))
p4 = Process(target=cpu_bound, args=(REPETICIONES//4,))

p1.start()
p2.start()
p3.start()
p4.start()

p1.join()
p2.join()
p3.join()
p4.join()
```

Multihilos

```
from threading import Thread

t1 = Thread(target=cpu_bound, args=(REPETICIONES//4,))
t2 = Thread(target=cpu_bound, args=(REPETICIONES//4,))
t3 = Thread(target=cpu_bound, args=(REPETICIONES//4,))
t4 = Thread(target=cpu_bound, args=(REPETICIONES//4,))

t1.start()
t2.start()
t3.start()
t4.start()

t1.join()
t2.join()
t3.join()
t4.join()
```

El culpable: GIL

Global Interperter Lock



GIL

No permite ejecutar código python en forma simultánea

¿Por qué existe el GIL?

La implementación del intérprete en CPython no es *thread-safe*

Es decir que el intérprete no se diseñó para trabajar con multiples hilos

¿Y si sacamos el GIL?

No es tan fácil.

Muchas *features* del lenguaje crecieron asumiendo que existe el GIL

Special cases aren't special enough to break the rules.
Although practicality beats purity.

El GIL en otras implementaciones de Python

Pypy

Tiene GIL como CPython

Cython

Tiene GIL pero se puede liberar con un bloque `with`

Jython

No tiene GIL

IronPython

No tiene GIL


Tenemos que aprender a convivir
con el GIL

¿Cómo liberar el bloqueo?

Llamadas al sistema operativo (Por ejemplo descargar un archivo de Internet)

```
socket.recv(1024)
```

¿Cómo liberar el bloqueo?

Usando la API C

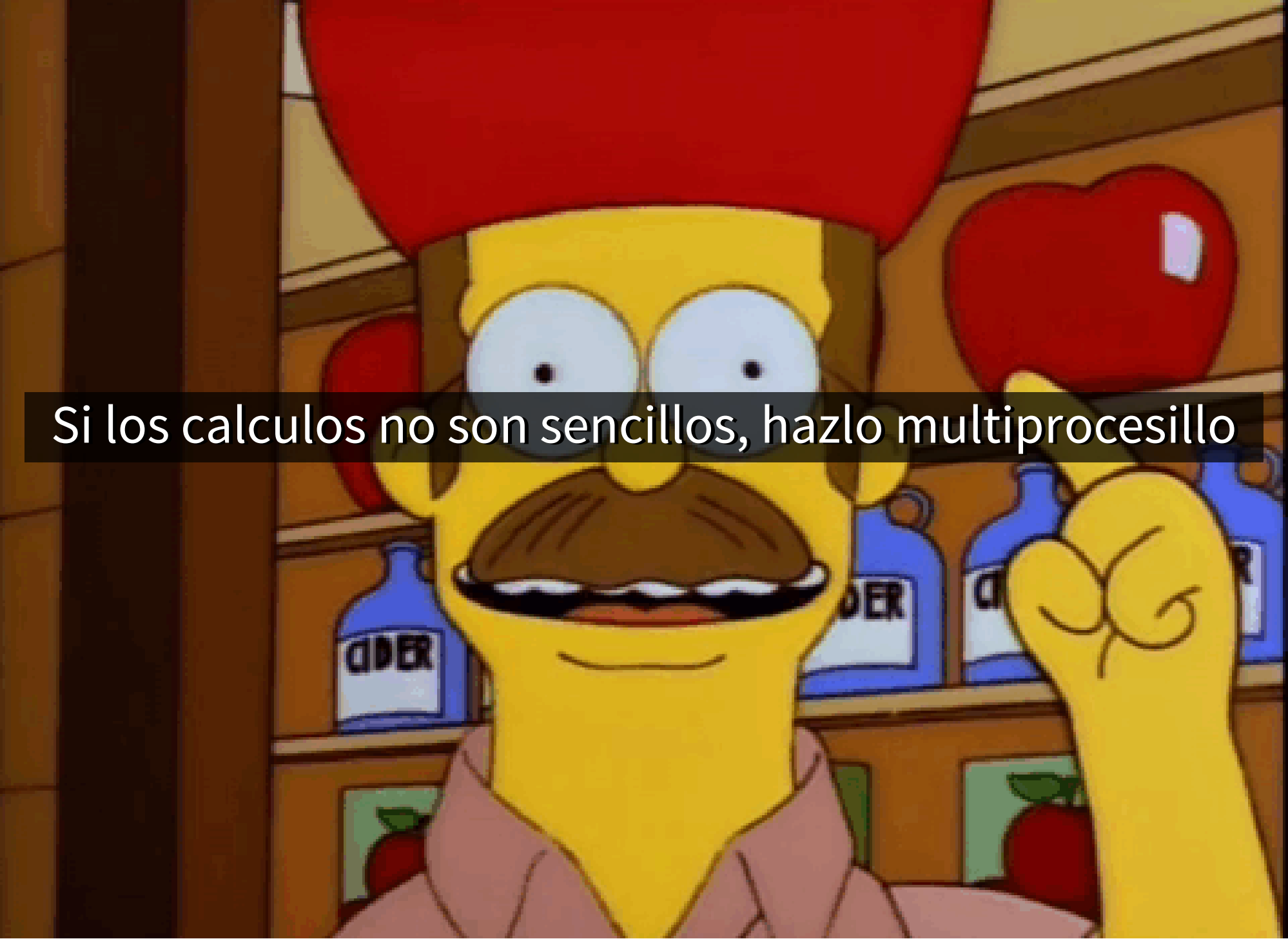
```
PyEval_InitThreads(); //initialize and aquire the GIL
//release the GIL, store thread state,
//set the current thread state to NULL
PyThreadState *mainThreadState = PyEval_SaveThread();

*main code segment*

//re-aquire the GIL (re-initialize the current thread state)
PyEval_RestoreThread(mainThreadState);

Py_Finalize();
```

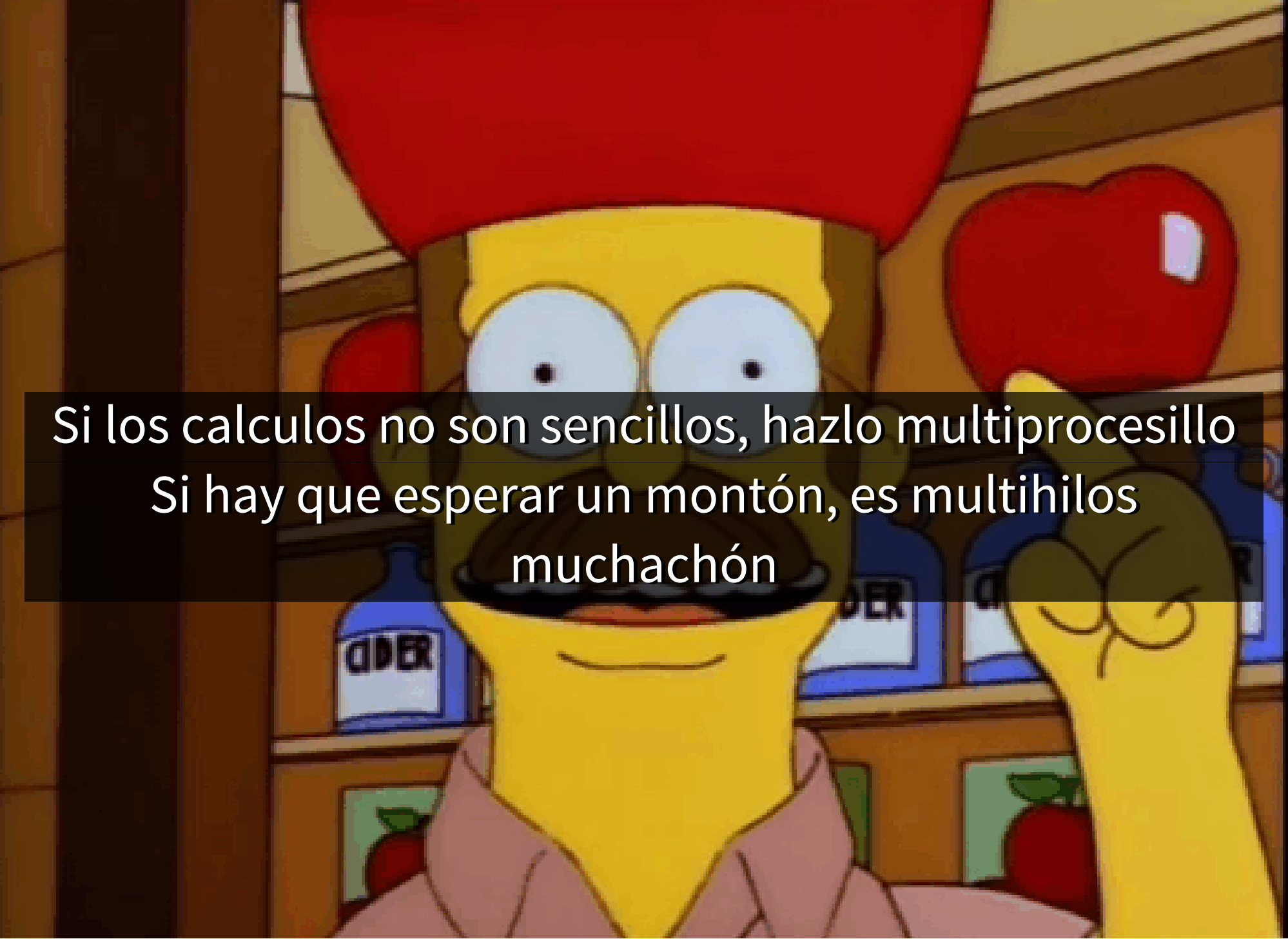
¿Cómo liberar el bloqueo?

Usando Cython

```
with nogil:
    algoritmo_paralelizable()
```


CIDER
CIDER


Si los calculos no son sencillos, hazlo multiprocesillo


Si los calculos no son sencillos, hazlo multiprocesillo
Si hay que esperar un montón, es multihilos muchachón

Asincronismo

Concurrencia cooperativa

Ejemplo IO-bound

Secuencial

```
CANTIDAD_MEMES = 10
def secuencial():
    """Descarga muchas imagenes de forma secuencial"""
    # Creo el directorio de descarga
    output_dir = pathlib.Path(OUTPUT_DIR)
    output_dir.mkdir(exist_ok=True)
    for _ in range(CANTIDAD_MEMES):
        download_one()
```

Multiprocesos

```
from concurrent.futures import ProcessPoolExecutor

from secuencial import CANTIDAD_MEMES, download_one, main

def multiproceso():
    """Descarga muchas imagenes en paralelo usando procesos"""
    with ProcessPoolExecutor(max_workers=4) as executor:
        futures = [
            executor.submit(download_one)
            for _ in range(CANTIDAD_MEMES)
        ]
        concurrent.futures.wait(futures)
```

Multihilos

```
from concurrent.futures import ThreadPoolExecutor

from secuencial import CANTIDAD_MEMES, download_one, main

def multihilo():
    """Descarga muchas imagenes en paralelo usando hilos"""
    with ThreadPoolExecutor(max_workers=CANTIDAD_MEMES) as executor:
        futures = [
            executor.submit(download_one)
            for _ in range(CANTIDAD_MEMES)
        ]
        concurrent.futures.wait(futures)
```

Asincronico

```
async def download_one():
    """Descarga una imagen y la guarda en disco"""
    url = await get_meme_url()
    print("Descargando " + url)
    await download_file(url, OUTPUT_DIR)

async def asincronico():
    """Descarga muchas imagenes de forma asincronica"""
    output_dir = pathlib.Path(OUTPUT_DIR)
    output_dir.mkdir(exist_ok=True)
    futures = [download_one() for _ in range(CANTIDAD_MEMES)]
    await asyncio.wait(futures)
```

Conclusión

El GIL limita la concurrencia en aplicaciones multihilo

Tipo de proceso	Concurrencia
CPU-Bound	Multiproceso
IO-Bound	Multihilo o asincrónico

Muchas gracias

Yonatan Romero @yonatancony

https://gitlab.com/romeroyonatan/gauchito-gil/