Alta Performance com Python

Paralelismo e Concorrência

Python Cerrado
Brasília 2018

Alta Performance com Python

Paralelismo e Concorrência

Bruno Barbosa

Desenvolvedor de Software na Globo.com

Brasiliense se aventurando pelo errejota :)

Programador por diversão há 14 anos

Há 7 anos escrevendo software profissionalmente

Apaixonado por Python e todo seu ecossistema

Iniciou sua carreira trabalhando com Python, Zope e Plone

Trabalhando atualmente na maior parte do tempo com Python, Go e JavaScript.

globo.com

Mais de

50 milhões

de visitas por dia

Homes da **Globo.com**

Processos e Threads

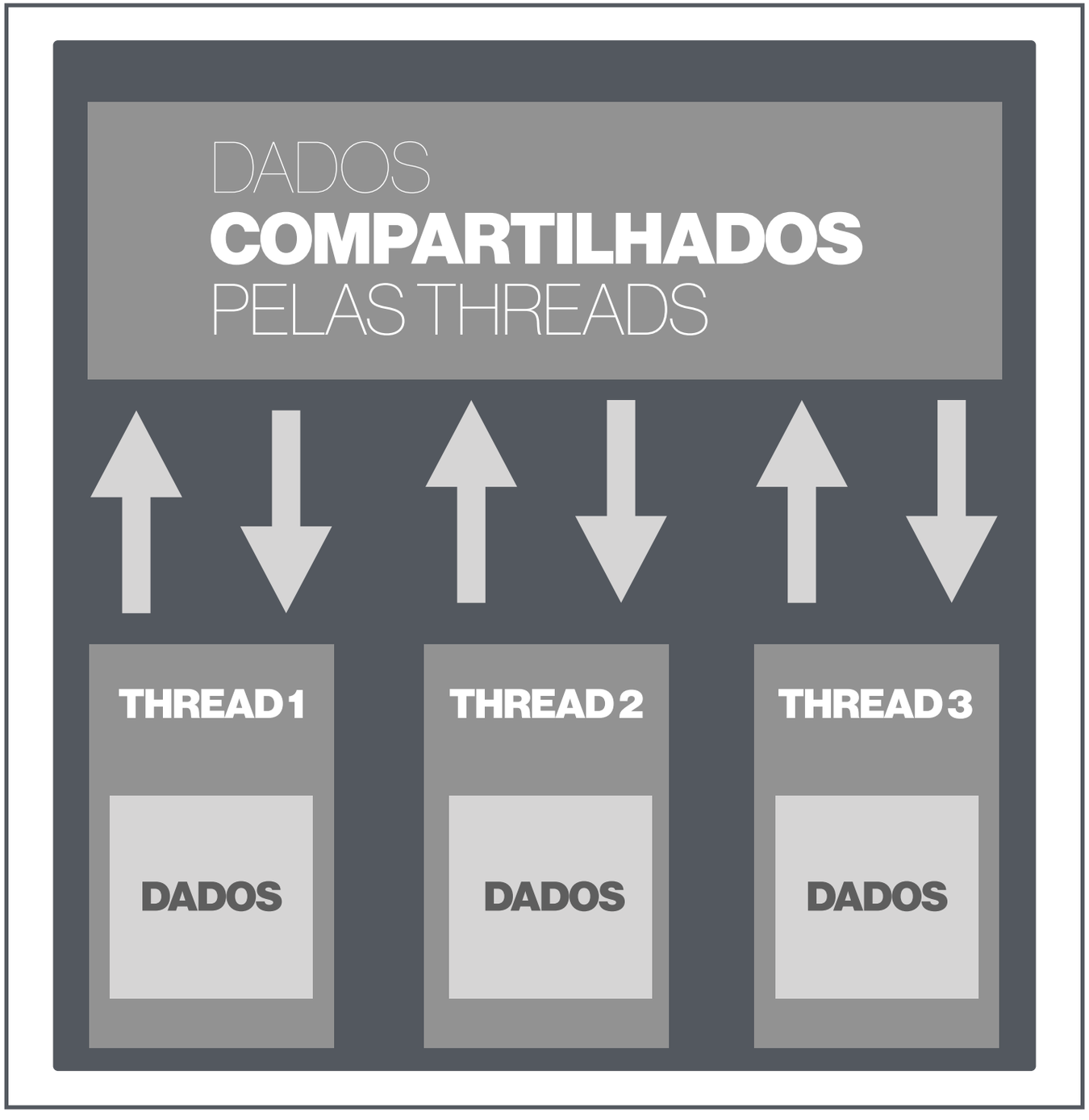
Processos e Threads

Comparativo

	Processos	Threads
Compartilham memória	✗	✓
Custo de spawn/switch	>	<
Necessidade de recursos	>	<
Mecanismos de sincronização	✗	✓

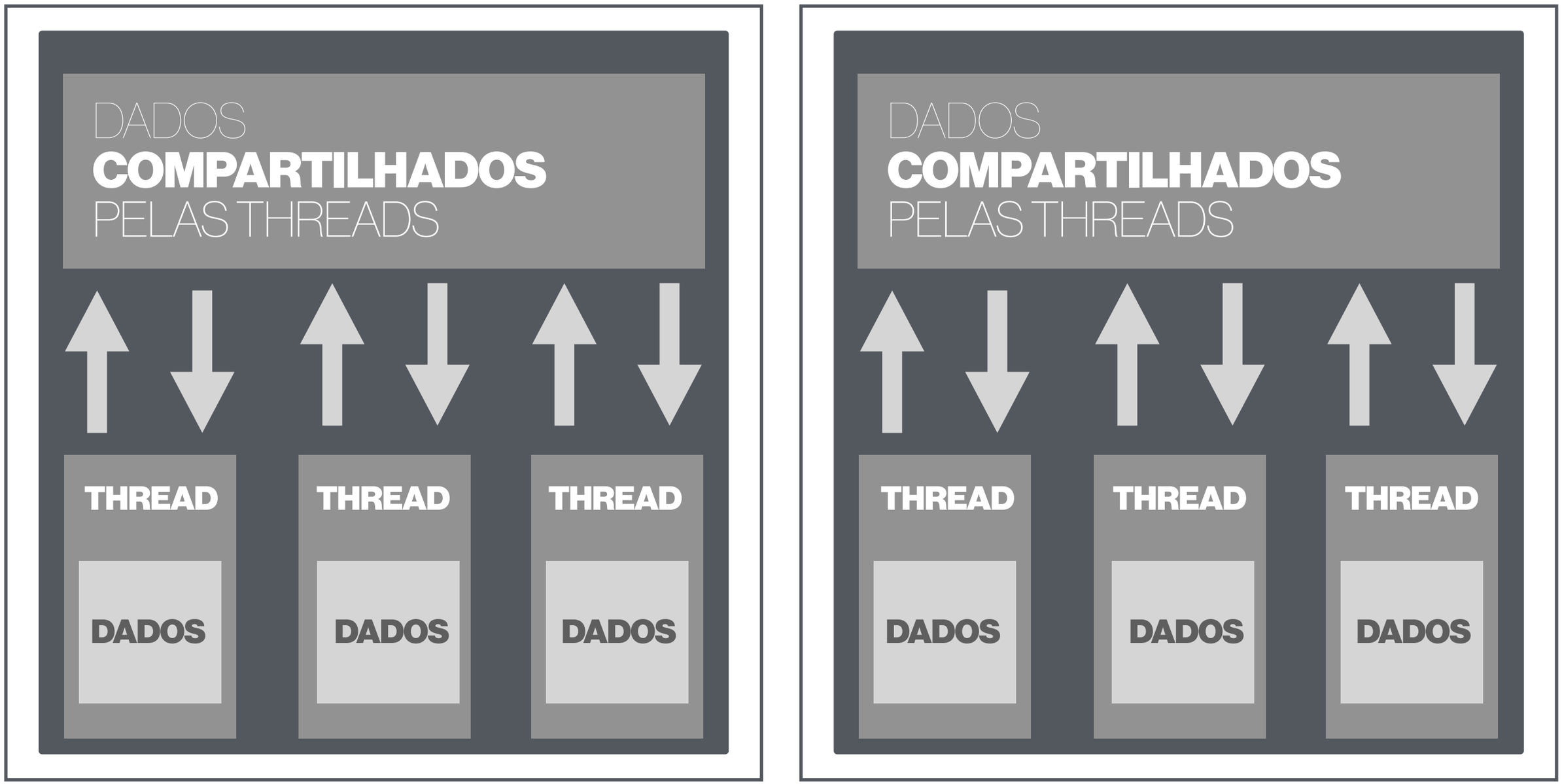
Processos e Threads

Anatomia de um processo



Processos e Threads

Anatomia de um processo



Processos e Threads

Tipos de processos

CPU BOUND

e

I/O BOUND

Processos e Threads

Tipos de processos

CPU BOUND

Computações matemáticas intensas

Algoritmos de busca e ordenação em memória

Processamento e reconhecimento de imagem

✓ **Paralelismo**

I/O BOUND

Transferência de dados pela rede

Escrita de arquivo em disco

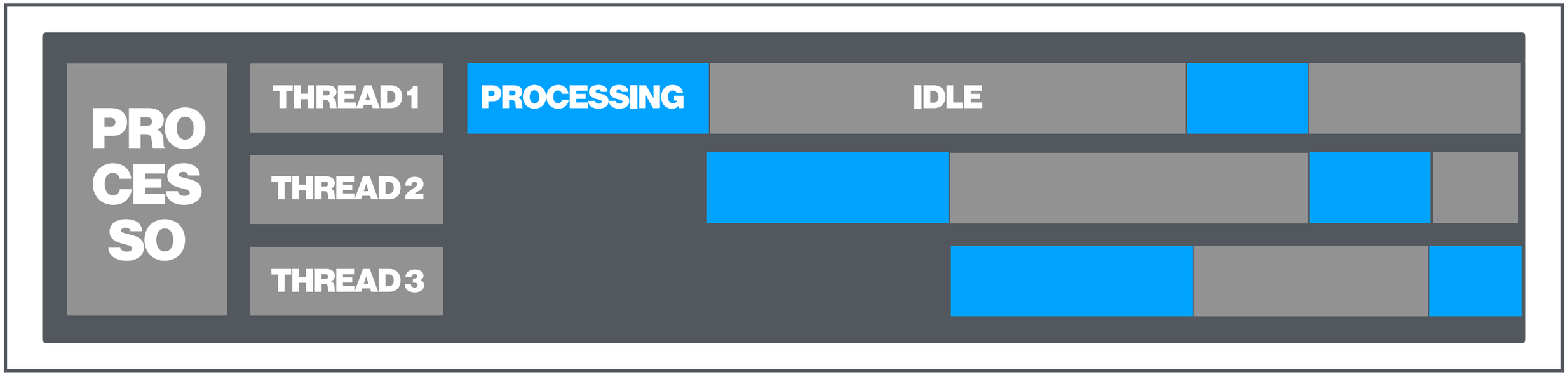
Consulta a uma API HTTP

✓ **Concorrência**

Concorrência e Paralelismo

Como funciona

CONCORRÊNCIA

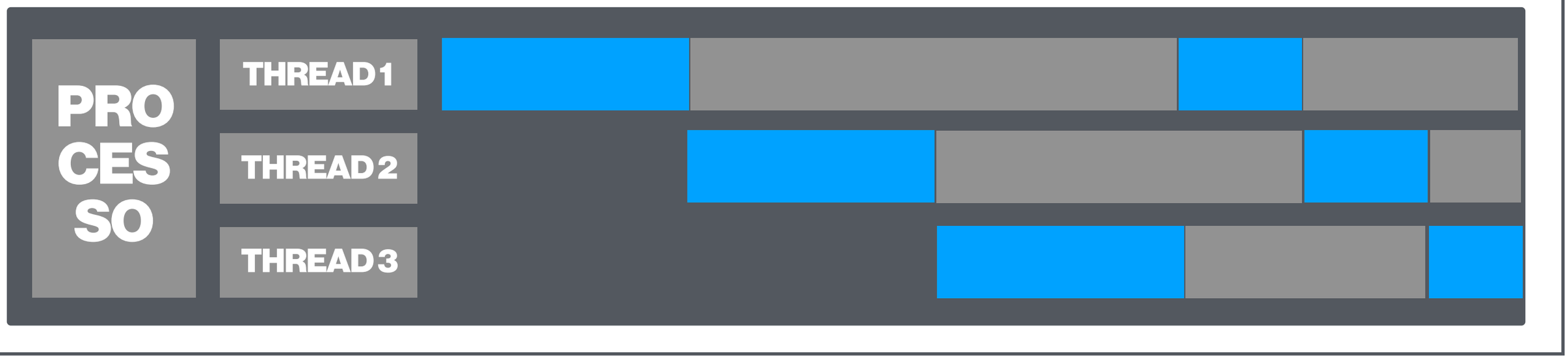


Concorrência e Paralelismo

Como funciona

PARALELISMO

CPU



CPU

VAMOS FALAR DE

PYTHON

O famigerado **GIL**

(Global Interpreter Lock)

SHOW ME

{ the Code }

I/O Bound

Simulando uma atividade de rede

```
import multiprocessing
import os
import threading
import time

MAX_WORKERS = 4
```

```
def io_expensive(sleep_time=1):
  """ Do nothing, wait for a timer to expire """
    print("PID: {}, Process Name: {}, Thread Name: {}".format(
        os.getpid(),
        multiprocessing.current_process().name,
        threading.current_thread().name)
    )
    time.sleep(sleep_time)
```

I/O Bound

Simulando uma atividade de rede

1º Teste: Sequencial

```
import time
from main_example import MAX_WORKERS, io_expensive

start_time = time.time()
for i in range(MAX_WORKERS):
    io_expensive(i)
end_time = time.time()

print("\n✔ Serial time=", end_time - start_time)
```

I/O Bound

Simulando uma atividade de rede

2º Teste: Threads

```python
import time
from concurrent import futures

from main_example import MAX_WORKERS, io_expensive

start_time = time.time()
with futures.ThreadPoolExecutor(MAX_WORKERS) as executor:
    executor.map(io_expensive, range(MAX_WORKERS))
end_time = time.time()

print("\n✔ Threads time=", end_time - start_time)
```

I/O Bound

Simulando uma atividade de rede

2º Teste: Threads

```
import time
from concurrent import futures

from main_example import MAX_WORKERS, io_expensive

start_time = time.time()

with futures.ThreadPoolExecutor(MAX_WORKERS) as executor:
    executor.map(io_expensive, range(MAX_WORKERS))

end_time = time.time()

print("\n✔ Threads time=", end_time - start_time)
```

I/O Bound

Simulando uma atividade de rede

3º Teste: Processos

```python
import time
from concurrent import futures

from main_example import MAX_WORKERS, io_expensive

start_time = time.time()
with futures.ProcessPoolExecutor() as executor:
    executor.map(io_expensive, range(MAX_WORKERS))
end_time = time.time()

print("\n✔ Parallel time=", end_time - start_time)
```

I/O Bound

Simulando uma atividade de rede

3º Teste: Processos

```
import time
from concurrent import futures

from main_example import MAX_WORKERS, io_expensive

start_time = time.time()

with futures.ProcessPoolExecutor() as executor:
    executor.map(io_expensive, range(MAX_WORKERS))

end_time = time.time()

print("\n✔ Parallel time=", end_time - start_time)
```

I/O Bound

Simulando uma atividade de rede

```
Running I/O expensive tasks...

✔ Running tasks serially...

PID: 10326, Process Name: MainProcess, Thread Name: MainThread
PID: 10326, Process Name: MainProcess, Thread Name: MainThread
PID: 10326, Process Name: MainProcess, Thread Name: MainThread
PID: 10326, Process Name: MainProcess, Thread Name: MainThread

✔ Serial time= 6.005445957183838
------------------------------------------------------------

✔ Running tasks using threads...

PID: 10326, Process Name: MainProcess, Thread Name: ThreadPoolExecutor-0_0
PID: 10326, Process Name: MainProcess, Thread Name: ThreadPoolExecutor-0_0
PID: 10326, Process Name: MainProcess, Thread Name: ThreadPoolExecutor-0_1
PID: 10326, Process Name: MainProcess, Thread Name: ThreadPoolExecutor-0_2

✔ Threads time= 3.0051229000091553
------------------------------------------------------------

✔ Running tasks using processes...

PID: 10329, Process Name: ForkProcess-1, Thread Name: MainThread
PID: 10330, Process Name: ForkProcess-2, Thread Name: MainThread
PID: 10329, Process Name: ForkProcess-1, Thread Name: MainThread
PID: 10331, Process Name: ForkProcess-3, Thread Name: MainThread

✔ Parallel time= 3.050796031951904
------------------------------------------------------------
```

CPU Bound

Simulando uma atividade de alto processamento

```
import multiprocessing
import os
import threading
import time

MAX_WORKERS = 4

def cpu_expensive():
    """ Do some computations """
    print("PID: {}, Process Name: {}, Thread Name: {}".format(
        os.getpid(),
        multiprocessing.current_process().name,
        threading.current_thread().name)
    )
    x = 0
    while x < 10000000:
        x += 1
```

CPU Bound

Simulando uma atividade de alto processamento

1º Teste: Sequencial

```
import time
from main_example import MAX_WORKERS, cpu_expensive

start_time = time.time()
for i in range(MAX_WORKERS):
    cpu_expensive()
end_time = time.time()

print("\n✔ Serial time=", end_time - start_time)
```

CPU Bound

Simulando uma atividade de alto processamento

2º Teste: Threads

```python
import time
from concurrent import futures

from main_example import MAX_WORKERS, cpu_expensive

start_time = time.time()
with futures.ThreadPoolExecutor(MAX_WORKERS) as executor:
    for _ in range(MAX_WORKERS):
        executor.submit(cpu_expensive)
end_time = time.time()

print("\n✔ Threads time=", end_time - start_time)
```

I/O Bound

Simulando uma atividade de rede

2º Teste: Threads

```
import time
from concurrent import futures

from main_example import MAX_WORKERS, cpu_expensive

start_time = time.time()

with futures.ThreadPoolExecutor(MAX_WORKERS) as executor:
    for _ in range(MAX_WORKERS):
        executor.submit(cpu_expensive)

end_time = time.time()

print("\n✔ Threads time=", end_time - start_time)
```

CPU Bound

Simulando uma atividade de alto processamento

3º Teste: Processos

```python
import time
from concurrent import futures

from main_example import MAX_WORKERS, io_expensive
```

```python
start_time = time.time()
with futures.ProcessPoolExecutor() as executor:
    for _ in range(MAX_WORKERS):
        executor.submit(cpu_expensive)
end_time = time.time()

print("\n✔ Parallel time=", end_time - start_time)
```

I/O Bound

Simulando uma atividade de rede

3º Teste: Processos

```
import time
from concurrent import futures

from main_example import MAX_WORKERS, io_expensive

start_time = time.time()

with futures.ProcessPoolExecutor() as executor:
    for _ in range(MAX_WORKERS):
        executor.submit(cpu_expensive)

end_time = time.time()

print("\n✔ Parallel time=", end_time - start_time)
```

CPU Bound

Simulando uma atividade de alto processamento

```
Running CPU expensive tasks...

✔ Running tasks serially...

PID: 11001, Process Name: MainProcess, Thread Name: MainThread
PID: 11001, Process Name: MainProcess, Thread Name: MainThread
PID: 11001, Process Name: MainProcess, Thread Name: MainThread
PID: 11001, Process Name: MainProcess, Thread Name: MainThread

✔ Serial time= 3.126549005508423
------------------------------------------------------------

✔ Running tasks using threads...

PID: 11001, Process Name: MainProcess, Thread Name: ThreadPoolExecutor-0_0
PID: 11001, Process Name: MainProcess, Thread Name: ThreadPoolExecutor-0_1
PID: 11001, Process Name: MainProcess, Thread Name: ThreadPoolExecutor-0_2
PID: 11001, Process Name: MainProcess, Thread Name: ThreadPoolExecutor-0_3

✔ Threads time= 3.0347580909729004
------------------------------------------------------------

✔ Running tasks using processes...

PID: 11004, Process Name: ForkProcess-2, Thread Name: MainThread
PID: 11003, Process Name: ForkProcess-1, Thread Name: MainThread
PID: 11005, Process Name: ForkProcess-3, Thread Name: MainThread
PID: 11006, Process Name: ForkProcess-4, Thread Name: MainThread

✔ Parallel time= 1.7643580436706543
------------------------------------------------------------
```

PARA CASOS MAIS COMPLEXOS USE

OS MÓDULOS **THREADING** e **MULTIPROCESSING**

E O TAL DO

ASYNC / AWAIT ?


POLICE BOX

GERADORES!!!

Generators

Produzindo valores através de yield

```
def gen_integer():
    print("Calling for fist time ... ")
    yield 1

    print("Calling for second time ... ")
    yield 2

    print("Calling for third time ... ")
    yield 3
```

```
>>> gen = gen_integer()

>>> next(gen)
Calling for fist time…
1

>>> next(gen)
Calling for second time…
2

>>> next(gen)
Calling for third time…
3

>>> next(gen)
Traceback (most recent
call last):
  …
StopIteration
```

Generators

Produzindo valores através de yield

```
def gen_fruit():
    fruits = ['apple', 'banana', 'grape']
    for fruit in fruits:
        yield fruit
```

```
>>> basket = gen_fruit()
>>> for fruit in basket:
        print(f"Got: {fruit}")

Got: apple
Got: banana
Got: grape
```

Generators

Usando yield from

```
def gen_fruit():
    fruits = ['apple', 'banana', 'grappes']
    yield from fruits s:
        yield fruit
```

```
>>> basket = gen_fruit()
>>> for fruit in basket:
        print(f"Got: {fruit}")

Got: apple
Got: banana
Got: grape
```

Coroutines

Coroutines

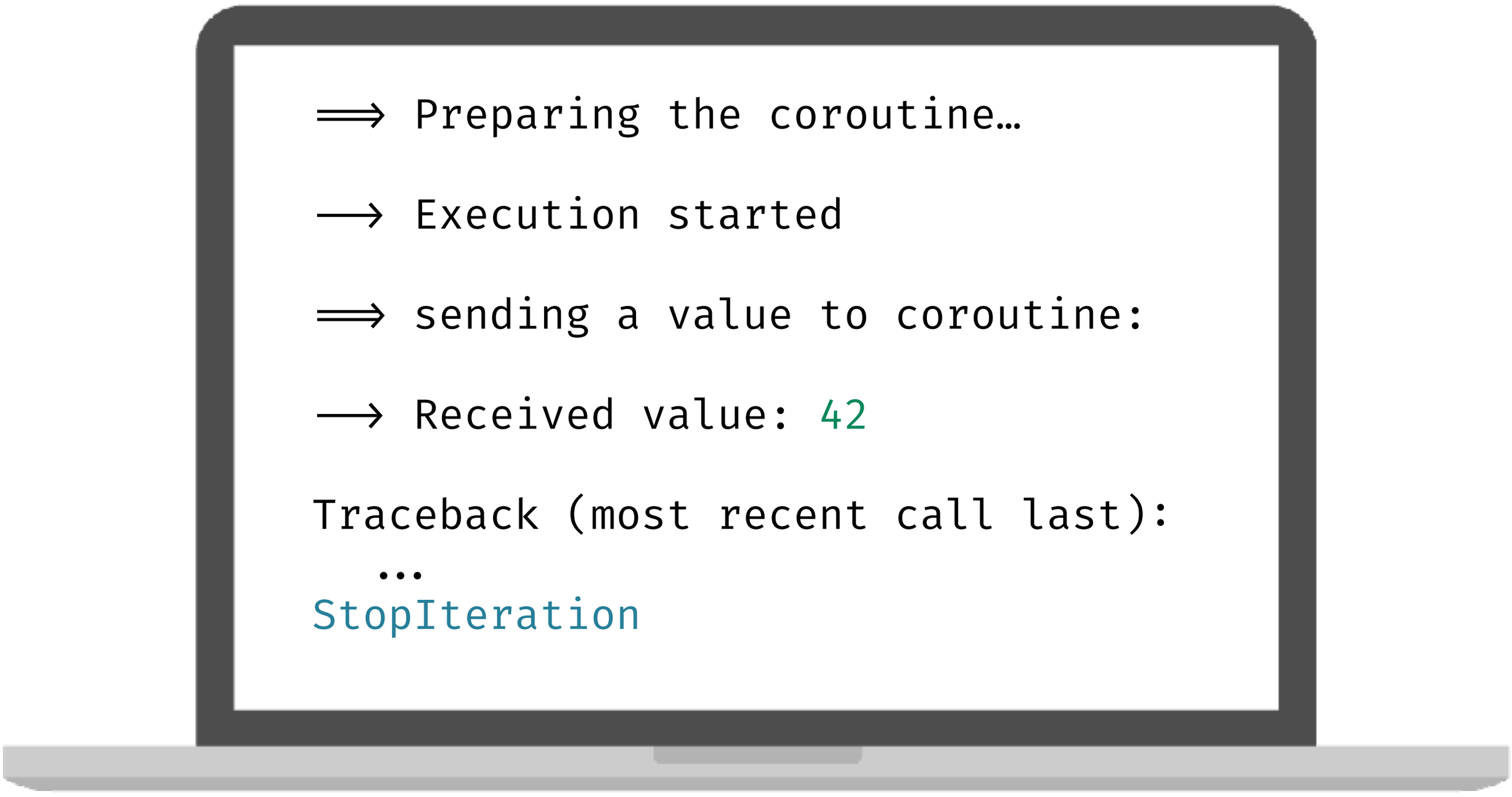
Um gerador melhorado

```
def example_coro():
    print("⟶ Execution started")
    x = yield
    print(f"⟶ Received value: {x}")
```

```
def main():
    coro = example_coro()
    print("⟹ Preparing the coroutine ... ")
    next(coro)
    print("⟹ sending a value to coroutine: ")
    coro.send(42)
```

Coroutines

Um gerador melhorado



Coroutines

Um gerador melhorado

```
def example_coro_2(start_num):
    print("⟶ Example started!")
    value = yield start_num

    print(f"⟶ Received: {value}")
    yield start_num + value
```

```
def example_2():
    coro = example_coro_2(20)

    initial_value = next(coro)
    print(f"⟹ Initial value: {initial_value}")

    v = coro.send(22)
    print(f"⟹ sum: {v}")

    next(coro)
```

Coroutines

Um gerador melhorado

```
⟶ Example started!

⟹ Initial value: 20

⟶ Received: 22

⟹ sum: 42
Traceback (most recent call last):
   ...
StopIteration
```

ASYNC / AWAIT

ASYNC / AWAIT

Código assíncrono através de corrotinas

```
import asyncio
import time

@asyncio.coroutine
def say_after(delay, what):
    yield from asyncio.sleep(delay)
    print(what)

@asyncio.coroutine
def main():
    print('started at', time.strftime('%X'))

    yield from say_after(1, 'hello')
    yield from say_after(2, 'world')

    print('finished at', time.strftime('%X'))


asyncio.run(main())
```

ASYNC / AWAIT

Código assíncrono através de corrotinas

```
import asyncio
import time

async def say_after(delay, what):
    await asyncio.sleep(delay)
    print(what)

async def main():
    print('started at', time.strftime('%X'))

    await say_after(1, 'hello')
    await say_after(2, 'world')

    print('finished at', time.strftime('%X'))

asyncio.run(main())
```

ASYNC / AWAIT

Fazendo requisições HTTP com o método aiohttp

```
import aiohttp
import asyncio

async def fetch(session, url):
    async with session.get(url) as response:
        return await response.text()

async def main():
    async with aiohttp.ClientSession() as session:
        html = await fetch(session, 'http://python.org')
        print(html)

loop = asyncio.get_event_loop()
loop.run_until_complete(main())
```

We're Hiring!!!

TEMOS VAGAS!!!

TALENTOS.GLOBO.COM

Contato

Onde me encontrar

@brunobbbs