

AsynclO w praktyce

Mateusz Rogowski

O mnie

Mateusz Rogowski

Absolwent Politechniki Białostockiej

Programista: 10+ lat


Programista Python: 8+ lat



AsynclO w praktyce



Jak przyspieszyć wykonywanie zadań w Pythonie?



Funkcja (endpoint)

**Parsowanie
danych wejściowych**

**Zapytanie do
bazy danych**

**Serializacja
danych wyjściowych**

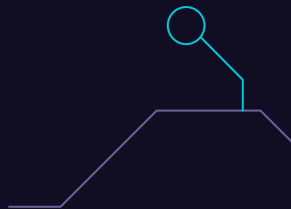


Funkcja (endpoint)

A in

A DB

A out

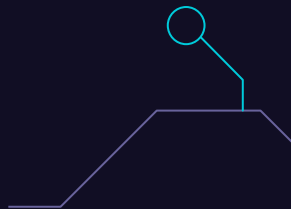




A in

A DB

A out





A in

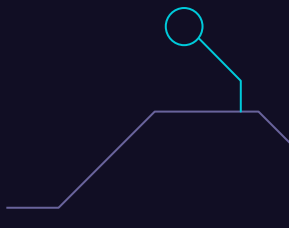
A DB

A out

B in

B DB

B out





A in

A DB

A out

B in

B DB

B out

C in

C DB

C out





A in

A DB

A out

B in

B DB

B out

C in

C DB

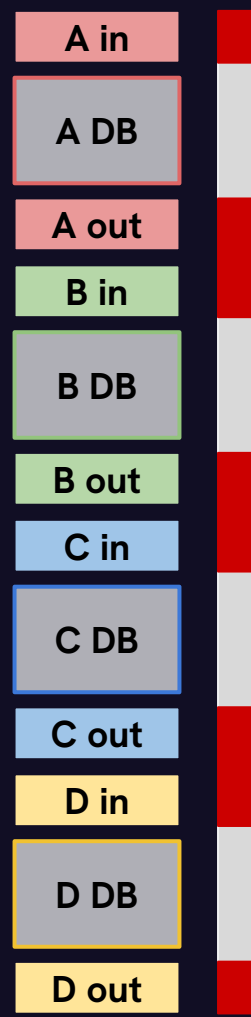
C out

D in

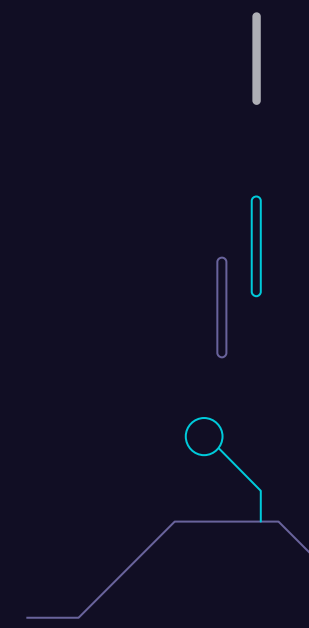
D DB

D out

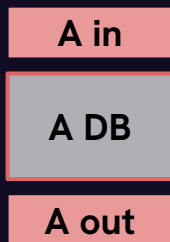




Obciążenie CPU



Równoległe wykonanie (parallelism)



Równoległe wykonanie (parallelism)

A in	B in	C in	D in
A DB	B DB	C DB	D DB
A out	B out	C out	D out

Równoległe wykonanie (parallelism)

Process 1	Process 2	Process 3	Process 4
A in	B in	C in	D in
A DB	B DB	C DB	D DB
A out	B out	C out	D out

Równoległe wykonanie (parallelism)

CPU 1	CPU 2	CPU 3	CPU 4
Process 1	Process 2	Process 3	Process 4
A in	B in	C in	D in
A DB	B DB	C DB	D DB
A out	B out	C out	D out

Asynchroniczne wykonanie

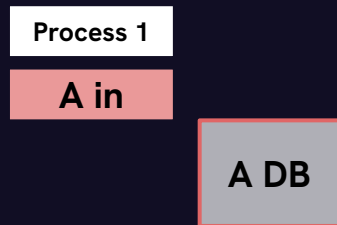
Process 1

Asynchroniczne wykonanie

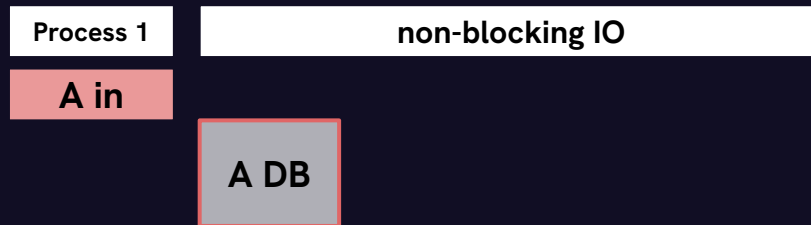
Process 1

A in

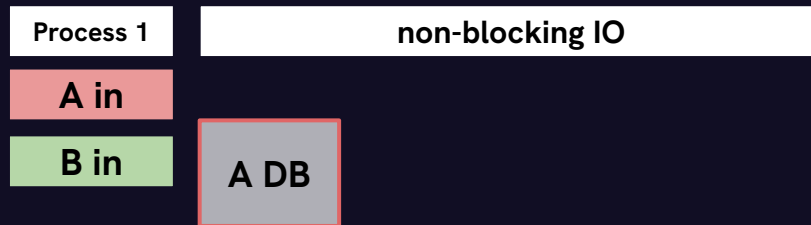
Asynchroniczne wykonanie



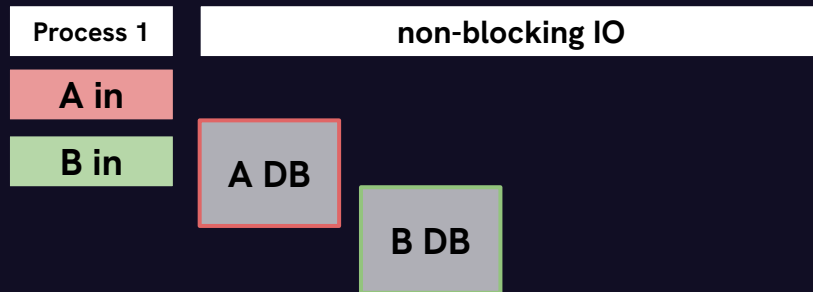
Asynchroniczne wykonanie



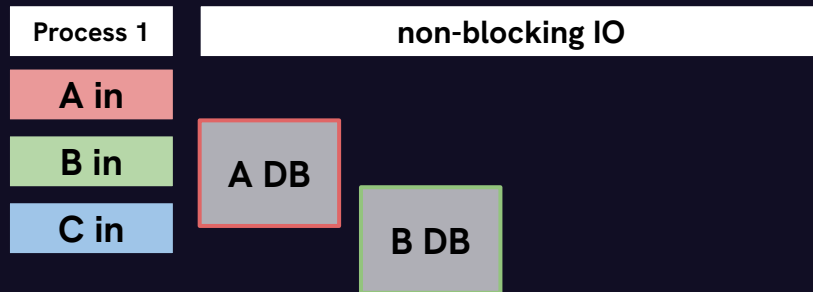
Asynchroniczne wykonanie



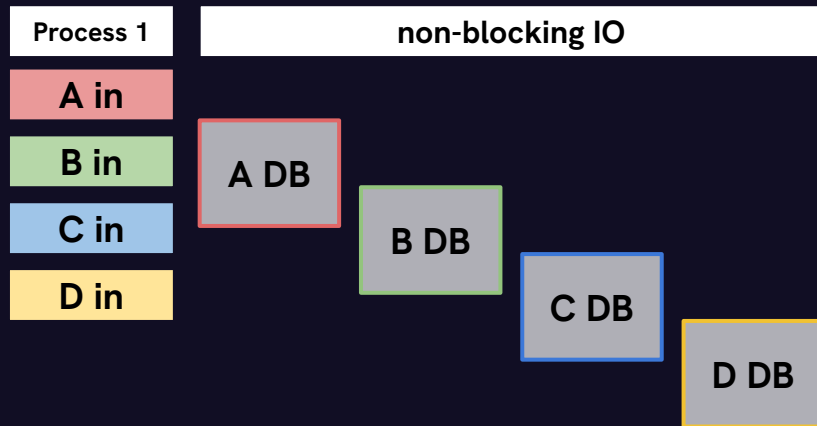
Asynchroniczne wykonanie



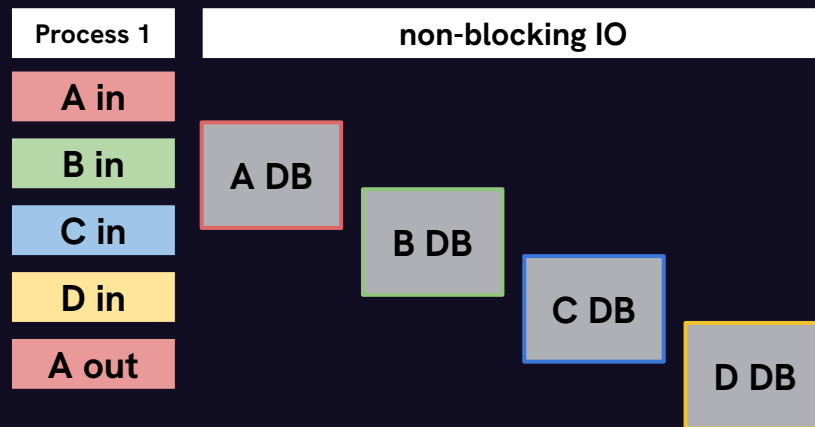
Asynchroniczne wykonanie



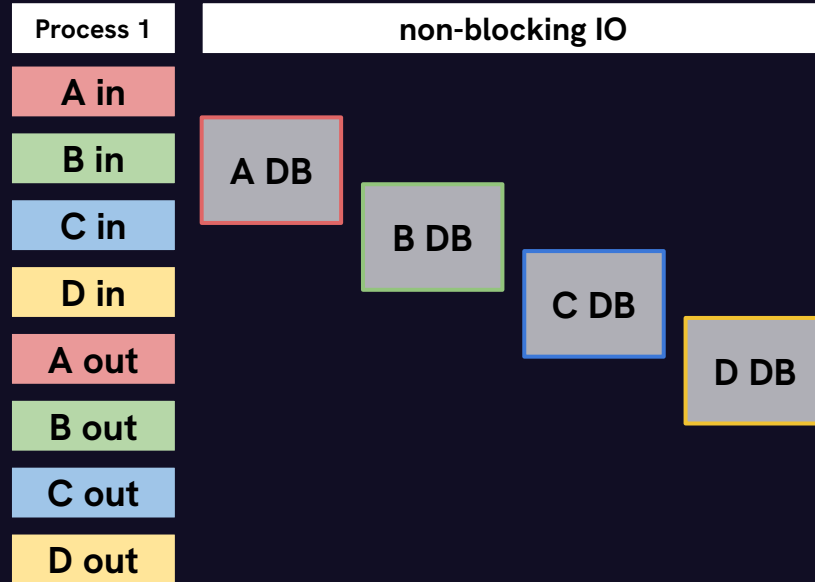
Asynchroniczne wykonanie



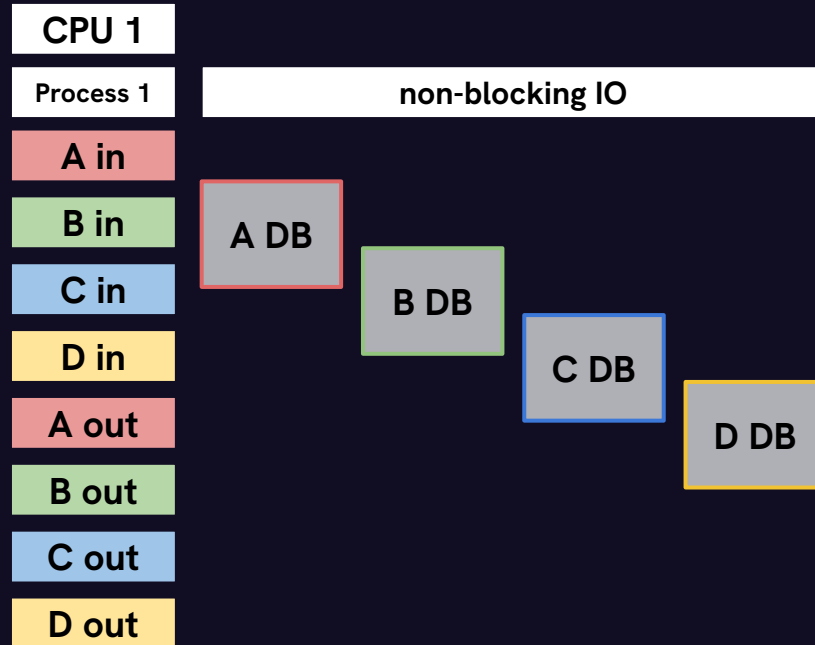
Asynchroniczne wykonanie



Asynchroniczne wykonanie



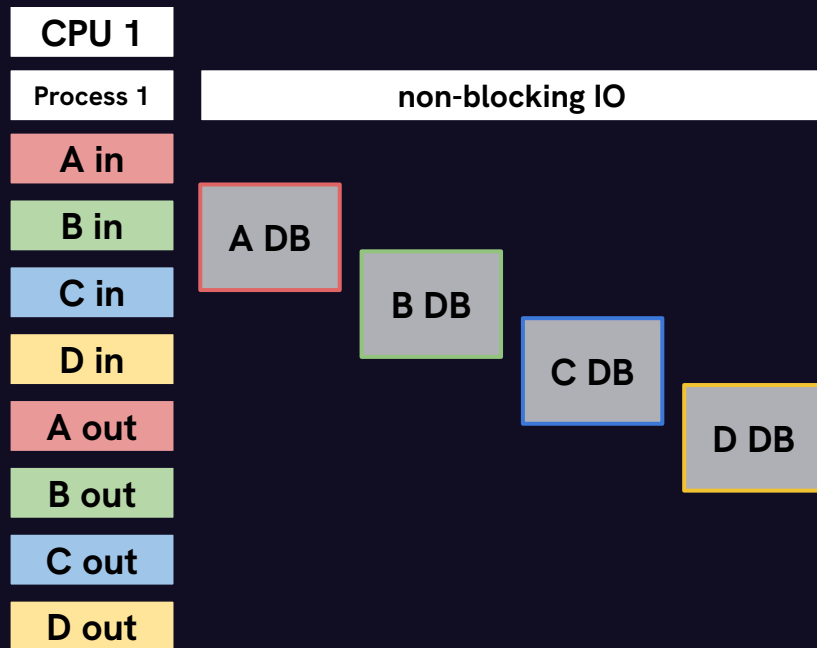
Asynchroniczne wykonanie



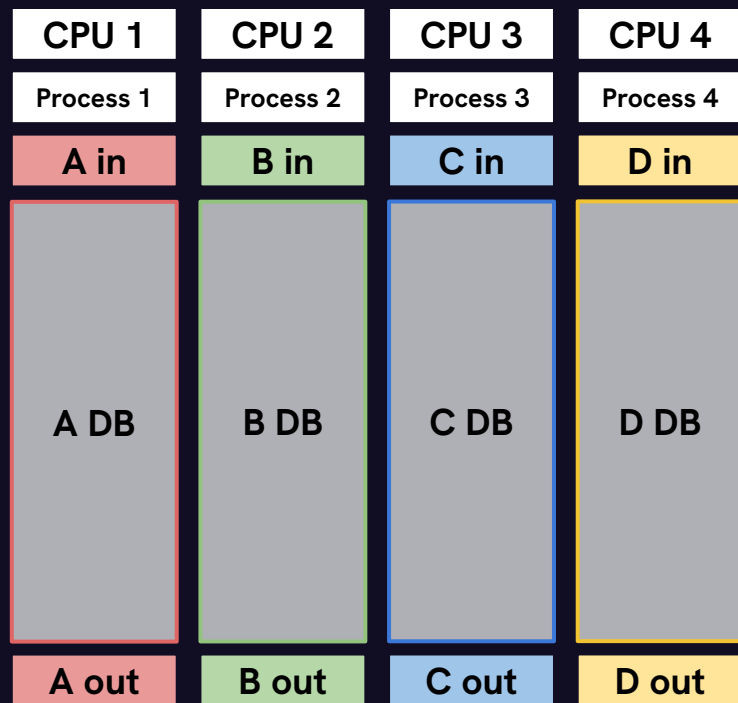
Równoległe wykonanie (parallelism)

CPU 1	CPU 2	CPU 3	CPU 4
Process 1	Process 2	Process 3	Process 4
A in	B in	C in	D in
A DB	B DB	C DB	D DB
A out	B out	C out	D out

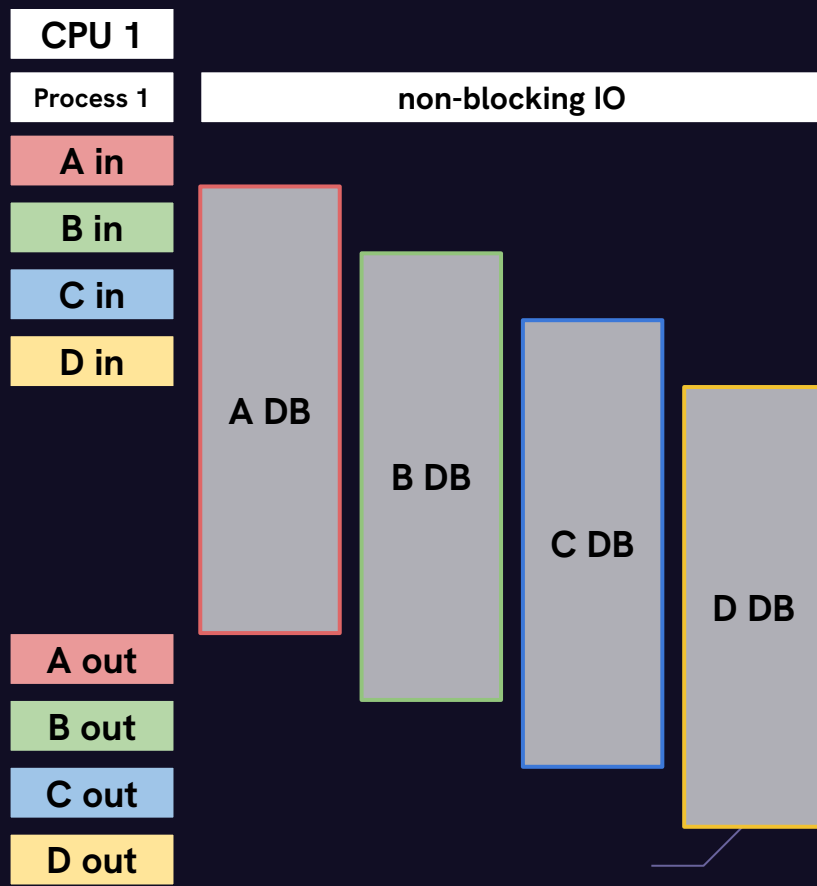
Asynchroniczne wykonanie



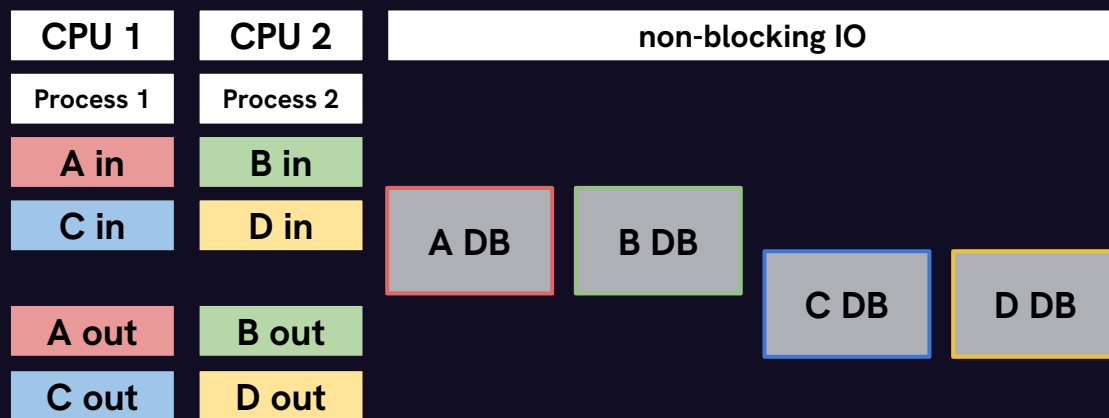
Równoległe wykonanie (parallelism)



Asynchroniczne wykonanie

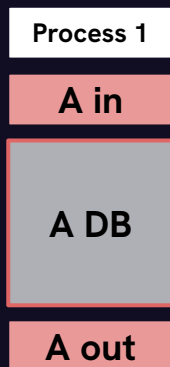


Równoległe wykonanie + asynchroniczność

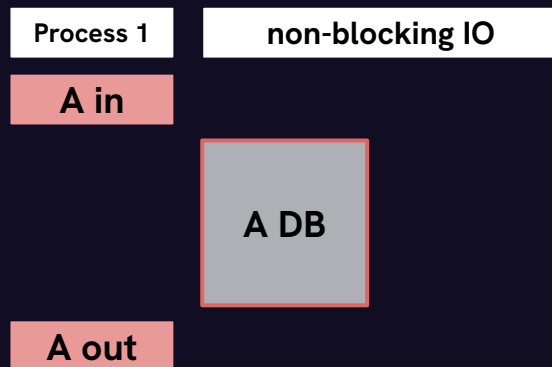
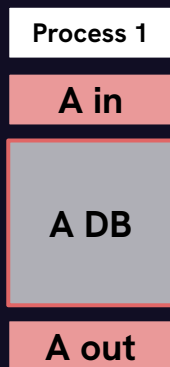


► Czas trwania IO i ilość zadań ma duże znaczenie

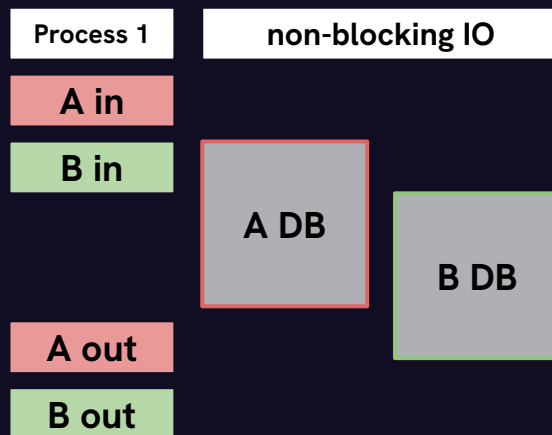
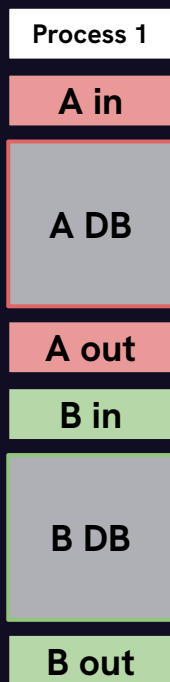
► Czas trwania IO i ilość zadań ma duże znaczenie



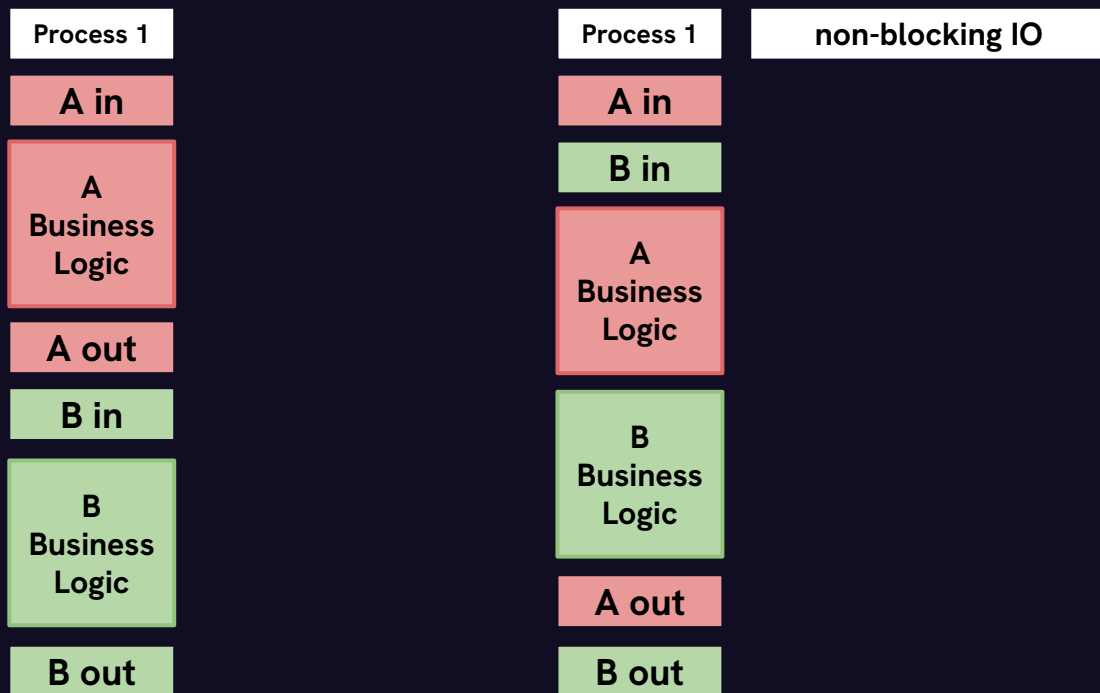
► Czas trwania IO i ilość zadań ma duże znaczenie



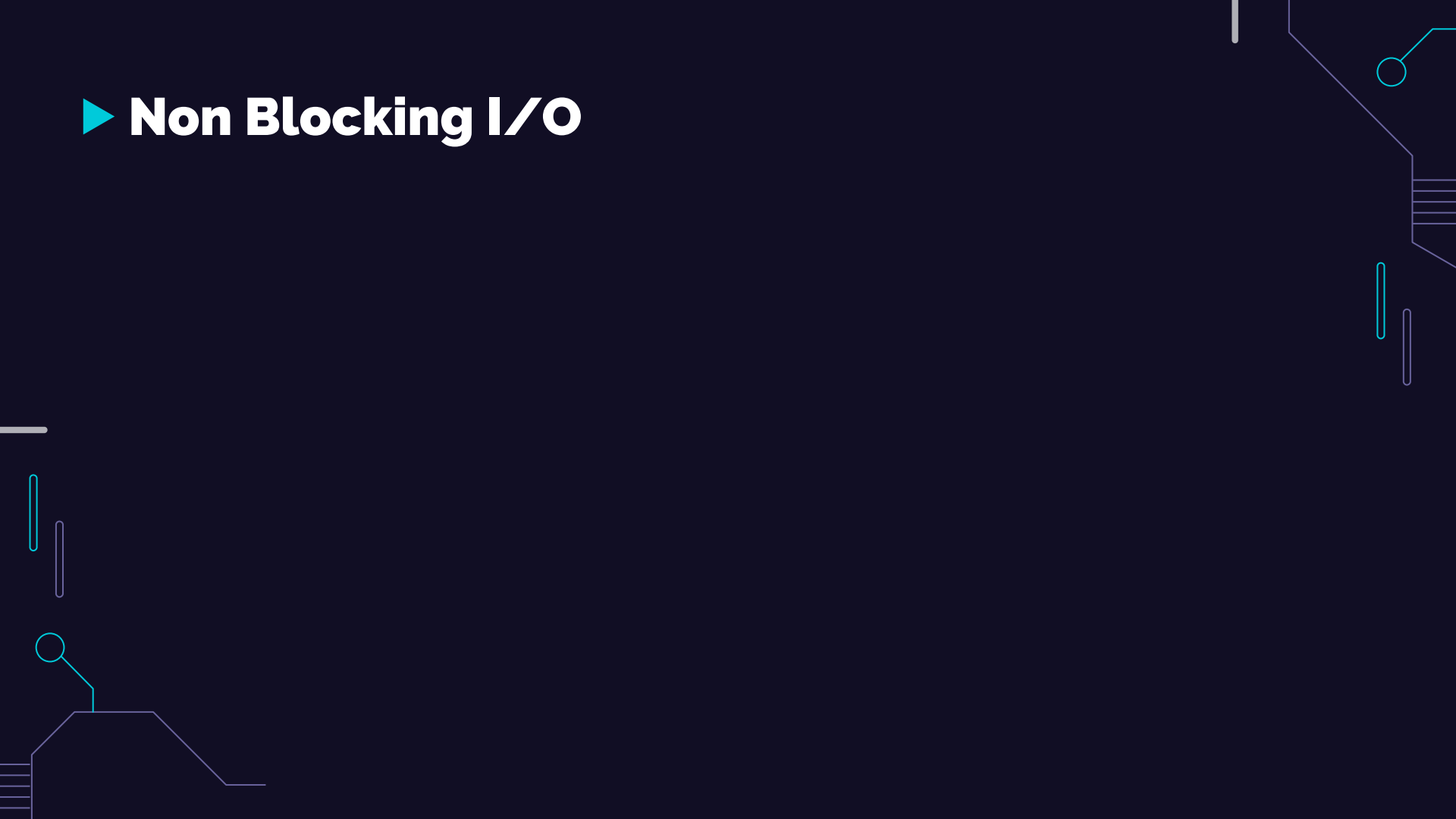
► Czas trwania IO i ilość zadań ma duże znaczenie



► Czas trwania IO i ilość zadań ma duże znaczenie



► Non Blocking I/O



► Non Blocking I/O

- Zapytania HTTP
 - httpx
 - aiohttp
 - requests-async

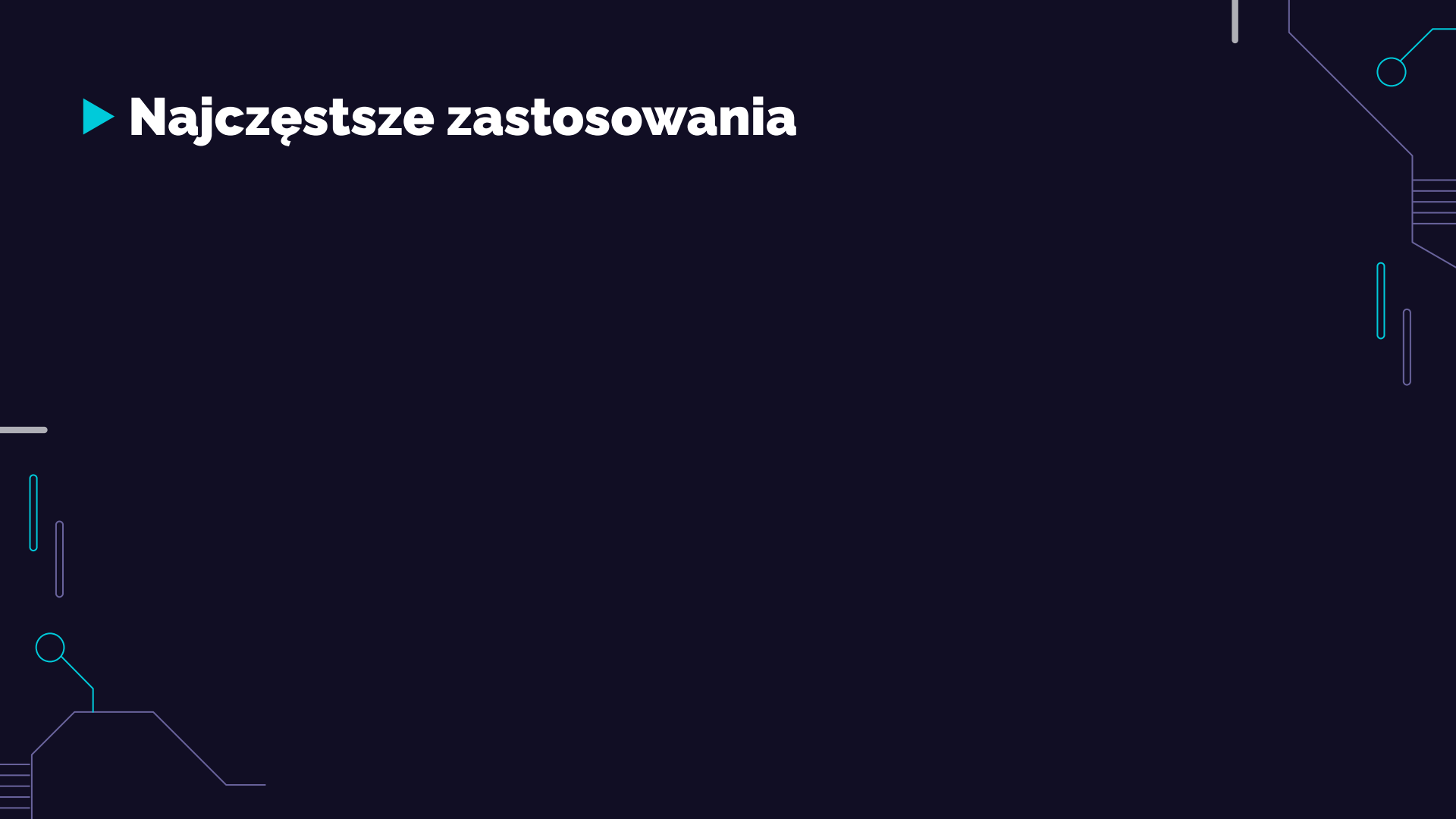
► Non Blocking I/O

- **Zapytania HTTP**
 - httpx
 - aiohttp
 - requests-async
- **Zapytania do Baz Danych**
 - asyncpg
 - aiosqlite
 - SQLAlchemy
 - Databases

► Non Blocking I/O

- **Zapytania HTTP**
 - httpx
 - aiohttp
 - requests-async
- **Zapytania do Baz Danych**
 - asyncpg
 - aiosqlite
 - SQLAlchemy
 - Databases
- **Zapis / odczyt danych na dysku**
 - aiofile
 - AnyIO

► Najczęstsze zastosowania



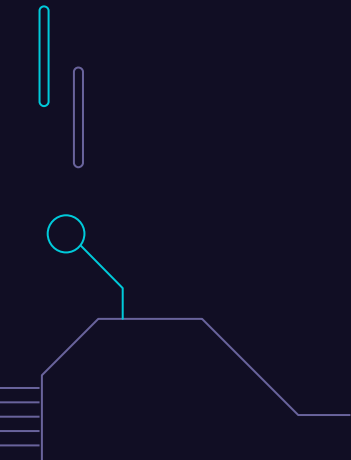
► Najczęstsze zastosowania

- REST API
 - Zapytania do bazy danych
 - Komunikacja między serwisami / mikroserwisami
 - File I/O

► Najczęstsze zastosowania

- REST API
 - Zapytania do bazy danych
 - Komunikacja między serwisami / mikroserwisami
 - File I/O
- Web crawling / scraping
 - Zapytania HTTP

► Kiedy nie stosować AsyncIO



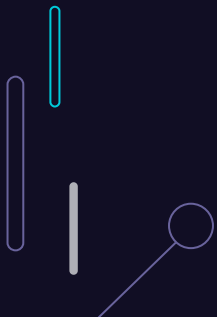
► Kiedy nie stosować AsyncIO

- Brak I/O
 - Wysokie obciążenie CPU bez czekania na I/O
 - Machine Learning (zwykle)

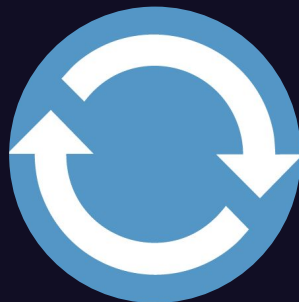
► Kiedy nie stosować AsyncIO

- Brak I/O
 - Wysokie obciążenie CPU bez czekania na I/O
 - Machine Learning (zwykle)
- Brak non blocking I/O

► Jak działa AsyncIO

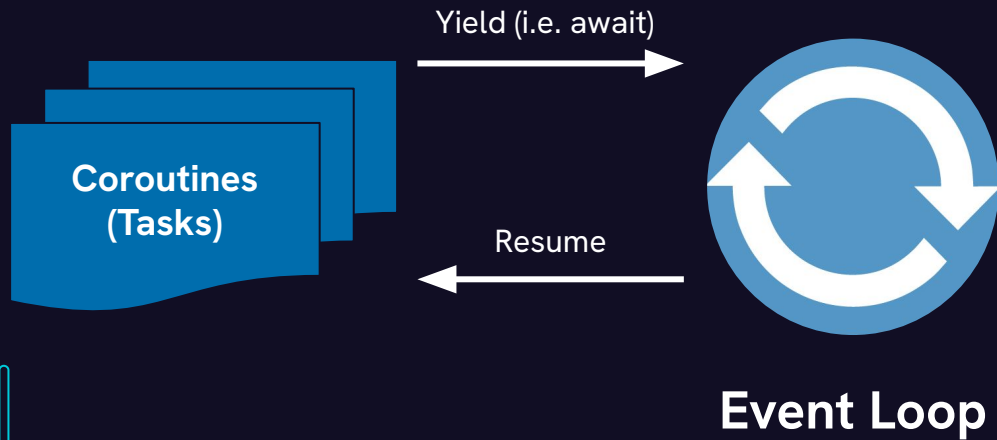


► Jak działa AsyncIO

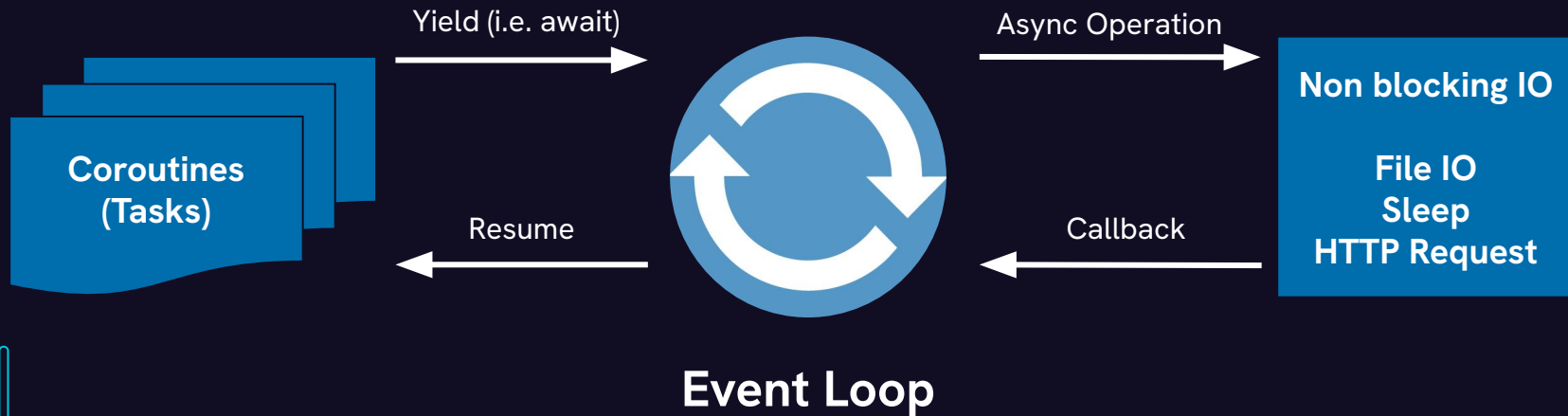


Event Loop

► Jak działa AsyncIO



► Jak działa AsyncIO



► Mity o AsyncIO

- Pozwala na obejście GIL

► Mity o AsyncIO

- Pozwala na obejście GIL
- Szybszy niż Threads

► Mity o AsyncIO

- Pozwala na obejście GIL
- Szybszy niż Threads
- Prostszy niż Threads

► Mity o AsyncIO

- Pozwala na obejście GIL
- Szybszy niż Threads
- Prostszy niż Threads
- Bezpieczny - nie ma wyścigów

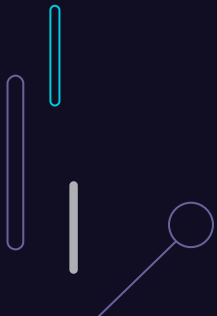
► **Bezpieczny - nie ma wyścigów**

- bezpieczny dostęp do pamięci

► Bezpieczny - nie ma wyścigów

- bezpieczny dostęp do pamięci
- możliwa modyfikacja pamięci pomiędzy wykonaniem kolejnych korutyn (coroutines)

► Mutex



► Mutex

```
lock = asyncio.Lock()

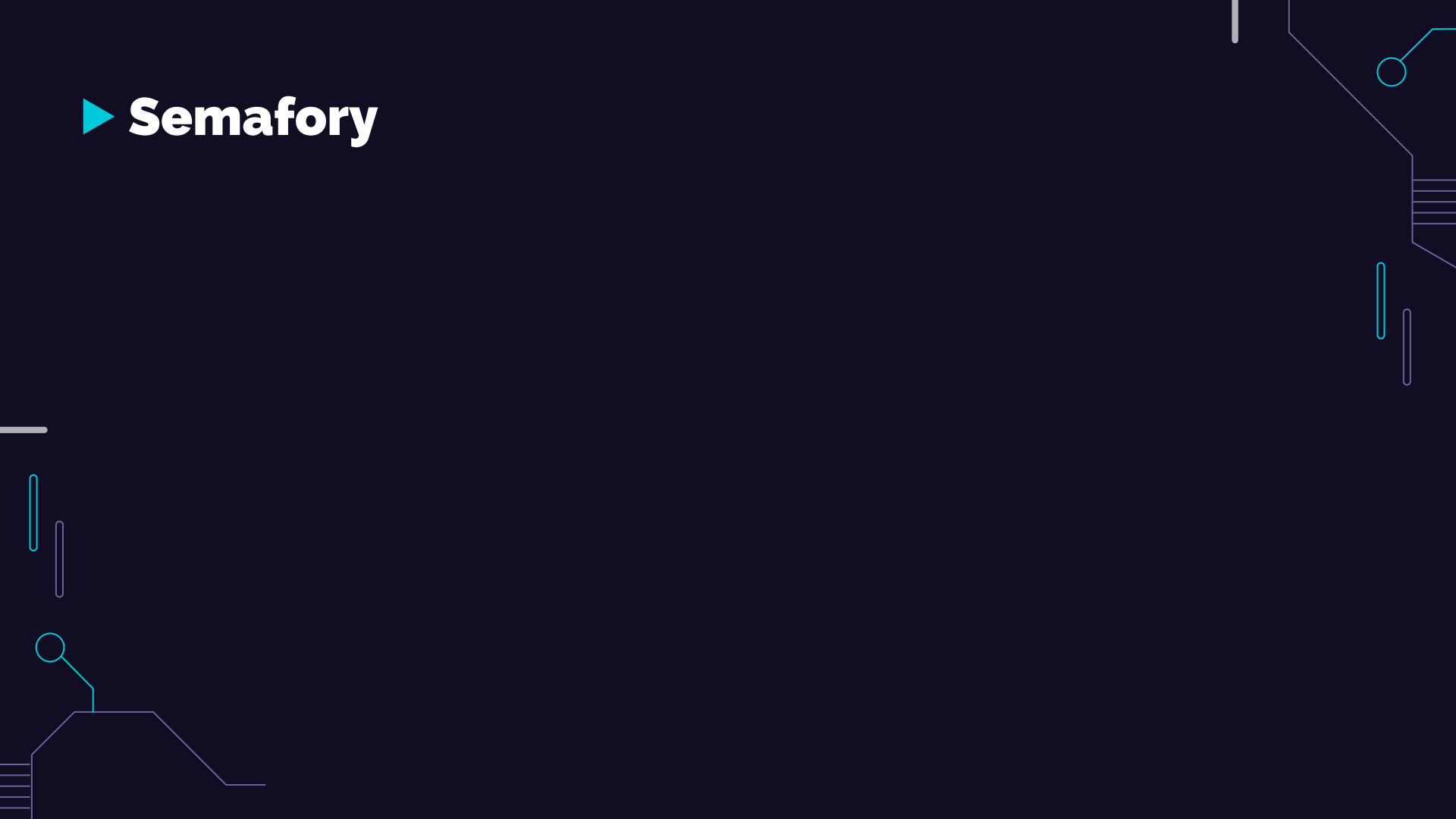
# ... later
async with lock:
    # access shared state
```

► Ograniczanie “równoczesnych” wykonań

Ograniczenie:

- używanej pamięci
- otwartych plików
- otwartych zapytań HTTP

► Semafor



► Semaphore

```
sem = asyncio.Semaphore(10)

# ... later
async with sem:
    # work with shared resource
```

► Asynchroniczne wywoływanie funkcji blokujących

```
await loop.run_in_executor
```

► Asynchroniczne wywoływanie funkcji blokujących

```
await loop.run_in_executor
```

- Wątki (Threads)

```
with concurrent.futures.ThreadPoolExecutor() as pool
```

► Asynchroniczne wywoływanie funkcji blokujących


```
await loop.run_in_executor
```

- Wątki (Threads)

```
with concurrent.futures.ThreadPoolExecutor() as pool
```

- Procesy (Multiprocessing)


```
with concurrent.futures.ProcessPoolExecutor() as pool
```



```
import asyncio
import concurrent.futures

def blocking_io():
    # File operations (such as logging) can block the
    # event loop: run them in a thread pool.
    with open('/dev/urandom', 'rb') as f:
        return f.read(100)

def cpu_bound():
    # CPU-bound operations will block the event loop:
    # in general it is preferable to run them in a
    # process pool.
    return sum(i * i for i in range(10 ** 7))
```




```
async def main():
    loop = asyncio.get_running_loop()

    ## Options:

    # 1. Run in the default loop's executor:
    result = await loop.run_in_executor(
        None, blocking_io)
    print('default thread pool', result)

    # 2. Run in a custom thread pool:
    with concurrent.futures.ThreadPoolExecutor() as pool:
        result = await loop.run_in_executor(
            pool, blocking_io)
        print('custom thread pool', result)

    # 3. Run in a custom process pool:
    with concurrent.futures.ProcessPoolExecutor() as pool:
        result = await loop.run_in_executor(
            pool, cpu_bound)
        print('custom process pool', result)
```



QA

