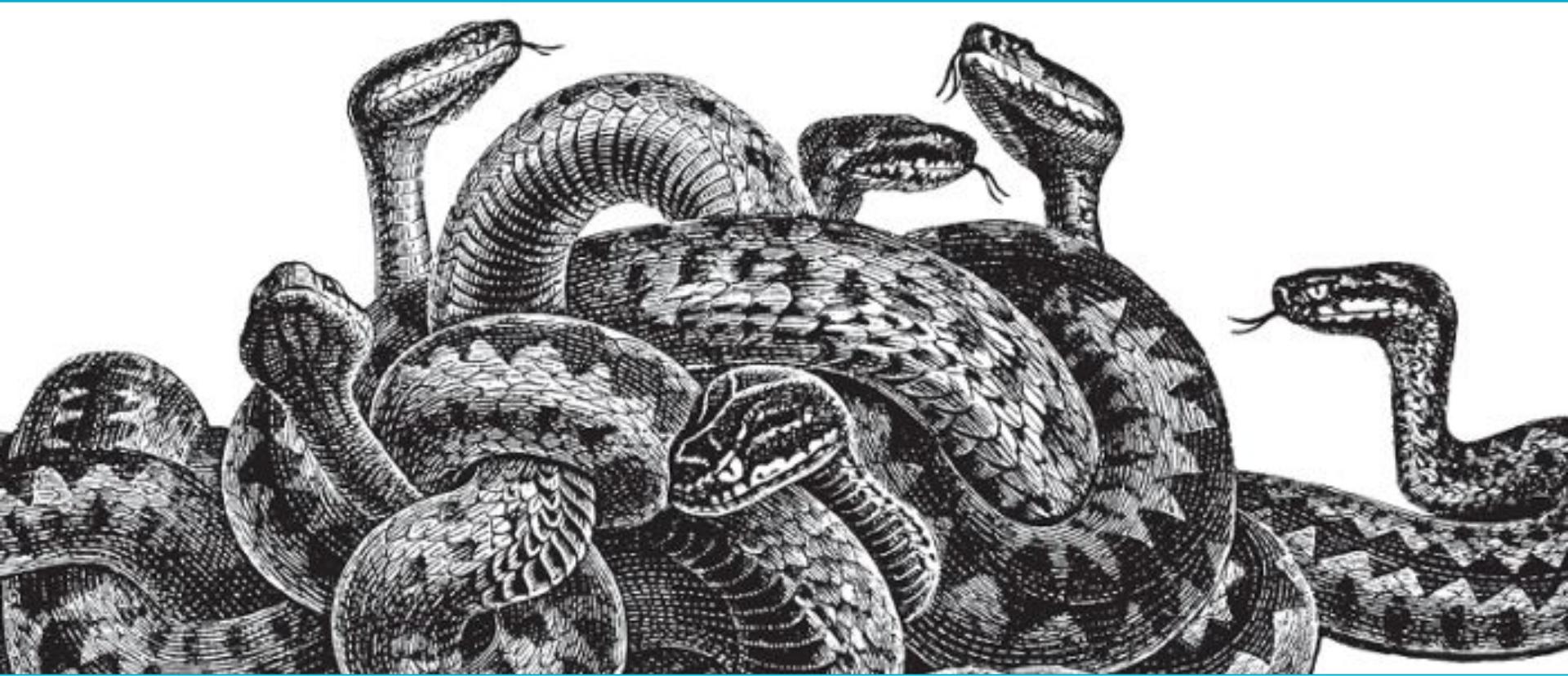
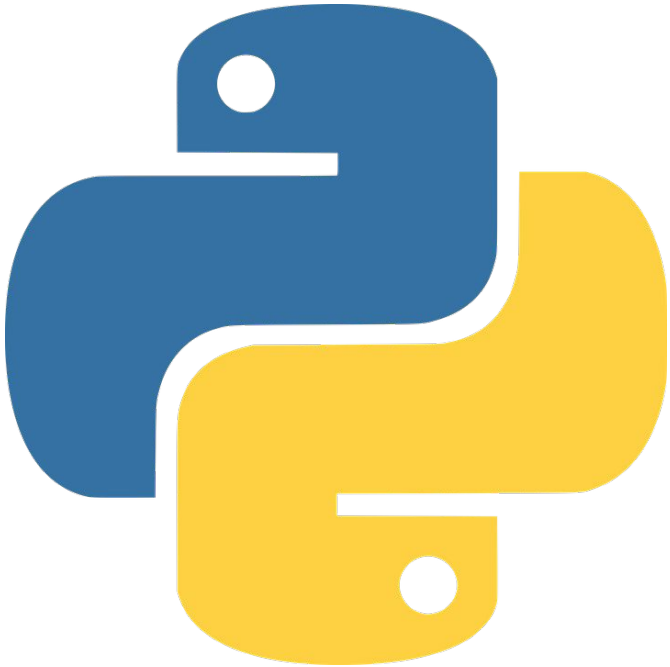
Dwa węże w cenie jednego

czyli programowanie współbieżne w Pythonie



CPython vs Anaconda vs PyPy vs ...



Dlaczego programowanie współbieżne?

Dostępność (np. serwer webowy)

Wydajność (intuicyjnie)

Świat działa w ten sposób

Concurrent vs Parallel

GIL - Global Interpreter Lock

```
// pythread.h
typedef void *PyThread_type_lock;


// ceval.c
static PyThread_type_lock interpreter_lock = 0; /* This is the GIL */

void
PyEval_InitThreads(void)
{
    if (interpreter_lock)
        return;
    interpreter_lock = PyThread_allocate_lock();
    PyThread_acquire_lock(interpreter_lock, 1);
    main_thread = PyThread_get_thread_ident();
}


// thread_pthread.h: PyThread_allocate_lock

lock = (pthread_lock *) malloc(sizeof(pthread_lock));
if (lock) {
    memset((void *)lock, '\0', sizeof(pthread_lock));
    lock->locked = 0;

    status = pthread_mutex_init(&lock->mut,
                                pthread_mutexattr_default);
```

GIL - Mity

Zło w czystej postaci

Zmniejsza wydajność Pythonowego programu

W Pythonie nie da się wykorzystać wielu rdzeni

GIL - Fakty

Nie pozwala na równoległe wykonanie kodu w obrębie jednego interpretera

Zabezpiecza wewnętrzny stan interpretera przed niespójnością (wyścigi)

Prosty koncept

A może warto usunąć GILa ?

https://github.com/larryhastings/gilectomy

Akt I - wątki

```
import threading


def do_some_work():
    print('I am worker')


print('Run some threads...')
for _ in range(10):
    threading.Thread(target=do_some_work).start()

print('I am the main thread')
```

Akt I - wątki

WSPÓŁDZIELĄ	NIE WSPÓŁDZIELĄ
Pamięć (heap)	Stos
Kod programu (instrukcje)	Rejestry (w tym wskaźnik instrukcji)
Deskryptory plików	

Akt I - wątki

Współdzielony stan - brak narzutu na komunikację

Wykonanie w obrębie jednego procesu - brak narzutu na utworzenie procesów

Współdzielony stan powoduje możliwość wystąpienia wyścigów

Wątki - model z wywłaszczeniem

Manager decyduje o przełączeniu wykonania pomiędzy wątkami

Nie nasza odpowiedzialność - wygoda użycia

Przełączenie może wystąpić w dowolnym momencie

Konieczność zabezpieczenia zasobów współdzielonych, koszt przełączenia

```python
import string
import threading

number_of_bikes = 5


def print_line_on_kiosk(text_line: str):
    print(text_line)


def rent_a_bike(renter_name: str):
    global number_of_bikes

    if number_of_bikes > 0:
        number_of_bikes -= 1
        print_line_on_kiosk(renter_name)
        print_line_on_kiosk('Just rented a bike!')
        print_line_on_kiosk('-------')
    else:
        print_line_on_kiosk(f'There is no bike for {renter_name}')
        print_line_on_kiosk('-------')


names = list(string.ascii_uppercase)[:10]

for name in names:
    threading.Thread(target=rent_a_bike, args=(name, )).start()

```

```
A
Just rented a bike!
--------
B
Just rented a bike!
--------
C
Just rented a bike!
--------
D
Just rented a bike!
--------
E
Just rented a bike!
--------
There is no bike for F
--------
There is no bike for G
--------
There is no bike for H
--------
There is no bike for I
--------
There is no bike for J
--------
```

```
D
H
There is no bike for I
C
F
Just rented a bike!
G
A
B
Just rented a bike!
J
--------
Just rented a bike!
Just rented a bike!
Just rented a bike!
--------
E
Just rented a bike!
Just rented a bike!
--------
--------
--------
Just rented a bike!
Just rented a bike!
--------
--------
--------
--------
--------
```

```python
bike_requests_queue = queue.Queue()
print_requests_queue = queue.Queue()


def bikes_manager():
    global number_of_bikes

    while True:
        renter_name = bike_requests_queue.get()
        if number_of_bikes > 0:
            number_of_bikes -= 1
            print_requests_queue.put(
                [renter_name, 'Just rented a bike!', '-------']
            )
        else:
            print_requests_queue.put(
                [f'There is no bike for {renter_name}', '-------']
            )
        bike_requests_queue.task_done()


def kiosk_manager():
    while True:
        text_lines = print_requests_queue.get()
        for line in text_lines:
            print(line)
        print_requests_queue.task_done()


def rent_a_bike(renter_name: str):
    bike_requests_queue.put(renter_name)

```

```
39      threading.Thread(target=bikes_manager, daemon=True).start()
40      threading.Thread(target=kiosk_manager, daemon=True).start()
41
42      names = list(string.ascii_uppercase)[:10]
43
44      renters = []
45      for name in names:
46          renter = threading.Thread(target=rent_a_bike, args=(name, ))
47          renters.append(renter)
48          renter.start()
49
50      for renter in renters:
51          renter.join()
52
53      bike_requests_queue.join()
54      print_requests_queue.join()
55
```

```python
import random
import threading

from exercises.utils import time_counter


def count():
    result = 0
    for value in range(100_000):
        result += random.randint(100_000_000, 999_999_999)


with time_counter():
    for _ in range(1000):
        count()

```

```
--- 141.73937582969666 seconds ---
```

```python
import random
import threading

from exercises.utils import time_counter


def count():
    result = 0
    for value in range(100_000):
        result += random.randint(100_000_000, 999_999_999)


with time_counter():
    workers = []
    for _ in range(1000):
        worker = threading.Thread(target=count)
        workers.append(worker)
        worker.start()

    for worker in workers:
        worker.join()

```

```
--- 146.32273292541504 seconds ---
```

Wątki - zwiększenie wydajności

Dla programów o dużej ilości operacji IO

Maksymalizują wykorzystanie jednego rdzenia

Narzut na przełączanie

numpy, Numba

Koniec aktu I - czas zabić wątek

Akt II - procesy

```python
import random
from multiprocessing import Pool

from exercises.utils import time_counter


def count(iteration_number: int):
    result = 0
    for value in range(iteration_number):
        result += random.randint(100_000_000, 999_999_999)


with time_counter():
    tasks = [100_000 for _ in range(1000)]
    with Pool(processes=4) as workers_pool:
        workers_pool.map(count, tasks)

```

```
--- 82.80544996261597 seconds ---
```

Akt II - procesy

Nie współdzielą pamięci - brak wyścigów

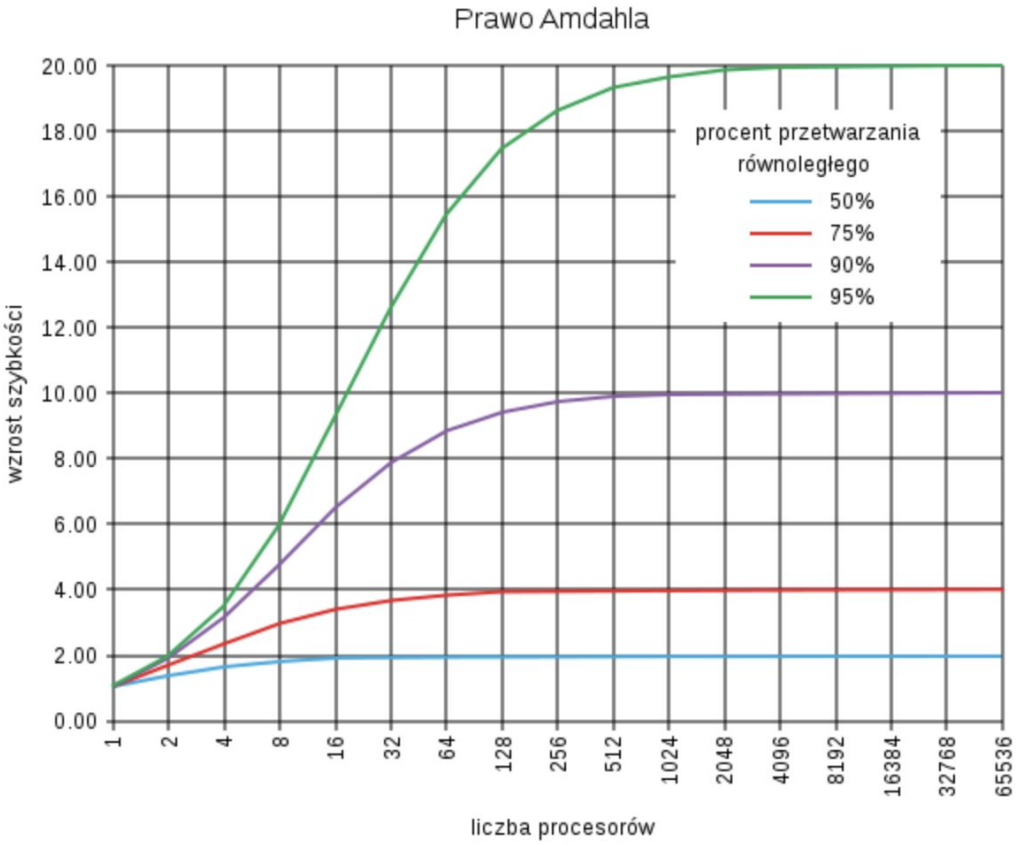
Narzut na komunikację

Wykorzystanie wielu rdzeni

Narzut na uruchomienie

Sky is the limit - prawo Amdahla

$$\frac{1}{(1-P)+\frac{P}{N}}.$$



Akt II - Proces rezerwacji i procesy

```
10      def create_station(request):
11          bike_station = BikeStation.objects.create()
12          return HttpResponse(f'Station created, id: {bike_station.pk}')
13
14
15      def rent_bike(request, station_id: int):
16          bike_station = BikeStation.objects.get(pk=station_id)
17          if bike_station.number_of_bikes > 0:
18              time.sleep(10)
19              bike_station.number_of_bikes -= 1
20              bike_station.save()
21              return HttpResponse('Rented a bike')
22          return HttpResponse('No bikes available')
23
24
25      def rent_bike_safe(request, station_id: int):
26          try:
27              with transaction.atomic():
28                  bike_station = BikeStation.objects.get(pk=station_id)
29                  if bike_station.number_of_bikes > 0:
30                      time.sleep(10)
31                      bike_station.number_of_bikes -= 1
32                      bike_station.save()
33                      return HttpResponse('Rented a bike')
34          except OperationalError:
35              return HttpResponse('Someone just took your bike!')
36
37          return HttpResponse('No bikes available')
```

Akt II - Wątki i procesy razem

https://bugs.python.org/issue27422

```python
#!/usr/bin/env python3
# coding:utf8

import sys
import multiprocessing
import subprocess
from concurrent.futures import ThreadPoolExecutor


def run(arg):
    print("starting %s" % arg)
    p = multiprocessing.Process(target=print, args=("running", arg))
    p.start()
    p.join()
    print("finished %s" % arg)


if __name__ == "__main__":
    n = 16
    tests = range(n)
    with ThreadPoolExecutor(n) as pool:
        for r in pool.map(run, tests):
            pass
```

Akt II - Gdy brakuje rdzeni

https://dask.org/

Akt III - asyncio

```python
import asyncio


async def hello():
    print("Hello")
    await asyncio.sleep(1)
    print("World!")


async def main_func():
    await asyncio.gather(hello(), hello(), hello())

asyncio.run(main_func())
```

Różne podejścia do asynchroniczności

Callbacki

Kolejka

Korutyna

Asyncio - model kooperatywny

Wykonanie w jednym wątku

Sami decydujemy kiedy się przełączyć

Brak potrzeby locków, sekcji krytycznej

Żadne wywołanie nie może być blokujące

Wykorzystanie jednego rdzenia

Przełączanie bardzo tanie

Brak narzutu na synchronizację

Potrzebny cały asynchroniczny ekosystem

native coroutine vs asynchronous generator

```
8       @asyncio.coroutine
9       def asynchronous_generator():
10          yield from task()
11
```

```
<generator object asynchronous_generator at 0x10f9e8ed0>
```

```
13      async def native_coroutine():
14          await task()
15
```

```
<coroutine object native_coroutine at 0x1100af5c8>
```

```python
6
7   async def get_bike_instruction(identifier: int):
8       print(colors[identifier] + f'Start working {identifier}')
9       result = random.randint(0, 5)
10      while result != 3:
11          print(colors[identifier] + f'{identifier} still working...')
12          await asyncio.sleep(identifier + 1)
13          result = random.randint(0, 5)
14      print(f'We have this! ({identifier})')
15      return 'Result'
16
17
18  async def main():
19      result = await asyncio.gather(
20          get_bike_instruction(0),
21          get_bike_instruction(1),
22          get_bike_instruction(2)
23      )
24      return result
25
26  result_1, result_2, result_3 = asyncio.run(main())
27  print(f"Results are: {result_1}, {result_2}, {result_3}")
28
29
```

```
Start working 0
0 still working...
Start working 1
1 still working...
Start working 2
2 still working...
0 still working...
1 still working...
We have this! (0)
2 still working...
1 still working...
2 still working...
1 still working...
1 still working...
2 still working...
1 still working...
2 still working...
1 still working...
We have this! (1)
2 still working...
2 still working...
2 still working...
2 still working...
2 still working...
2 still working...
2 still working...
```

Asynchroniczna aplikacja

- Asynchroniczny serwer + framework
- Asynchroniczny dostęp do bazy (GINO, asyncpg)
- Asynchroniczny dostęp do systemu plików
- Żadne wywołanie nie może być blokujące

Akt IV - Celery

Rozproszony system przetwarzania zadań

Zadanie -> Kolejka -> Worker -> Wykonanie

Łatwo dodać do istniejącego projektu

Nadzór nad wykonywanymi zadaniami

```
app = Celery('module_name', broker=settings.BROKER_URL)
app.config_from_object('django.conf:settings')


@app.task
def send_email(message):
    message.send()
```

```
result = send_email.delay('email_obj')
result.get(timeout=3)
```

Flower

Celery Flower

localhost:5555/workers

Celery Flower | Workers | Tasks | Monitor | Docs | About

Workers

Shut Down

	Name	Status	Concurrency	Completed Tasks	Running Tasks	Queues
	celery1.pi.local	Online	4	13902	0	images, data, video
	celery2.pi.local	Online	4	13900	0	images, data, video
	celery3.pi.local	Online	4	13826	0	images, data, video
	celery4.pi.local	Online	1	1989	0	data
	celery5.pi.local	Online	1	1983	0	data
	celery6.pi.local	Offline	3	2245	3	
	celery7.pi.local	Online	3	2283	3	celery, data
	celery8.pi.local	Online	3	2279	3	celery
	celery9.pi.local	Online	3	2287	3	celery

Akt V - podsumowanie - wątki

Wykorzystanie jednego rdzenia (GIL, wzrost wydajności gdy dużo IO)

Model z wywłaszczeniem - musimy zabezpieczyć stan systemu przed wyścigami

Wykorzystują istniejący ekosystem bibliotek

Trudno napisać poprawny kod, trudno utrzymać

Akt V - podsumowanie - procesy

Wykorzystanie wielu rdzeni ale narzut na komunikację

Dla problemów, które możemy zrównoleglić i przetwarzać części względnie niezależnie

Gdy dodatkowo potrzebna optymalizacja względem IO można łączyć z wątkami lub asyncio

Akt V - podsumowanie - asyncio

Wykorzystanie jednego rdzenia, oddanie sterowania przy operacjach IO do innego zadania

Model kooperatywny, pozwalamy na wykonanie innego zadania za pomocą await

Kod dużo prostszy w utrzymaniu niż dla programu wielowątkowego

Wymaga całego asynchronicznego ekosystemu

Akt VI - Subinterpreters

Bez narzutu procesów i bez ograniczeń GILa

Aktualnie: Draft, PEP 554

Materiały, linki

Raymond Hettinger: https://www.youtube.com/watch?v=9zinZmE3Ogk

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

Łukasz Langa: https://www.youtube.com/watch?v=I4Nn-y9ktd4

Dokumentacja (Python stdlib, PEP, Tornado, Dask, Numba, …) oraz artykuły (uwaga!)

Larry Hastings: https://www.youtube.com/watch?v=P3AyI_u66Bw

Blog: PyJazz.pl

www.linkedin.com/in/lewandowski-mikolaj