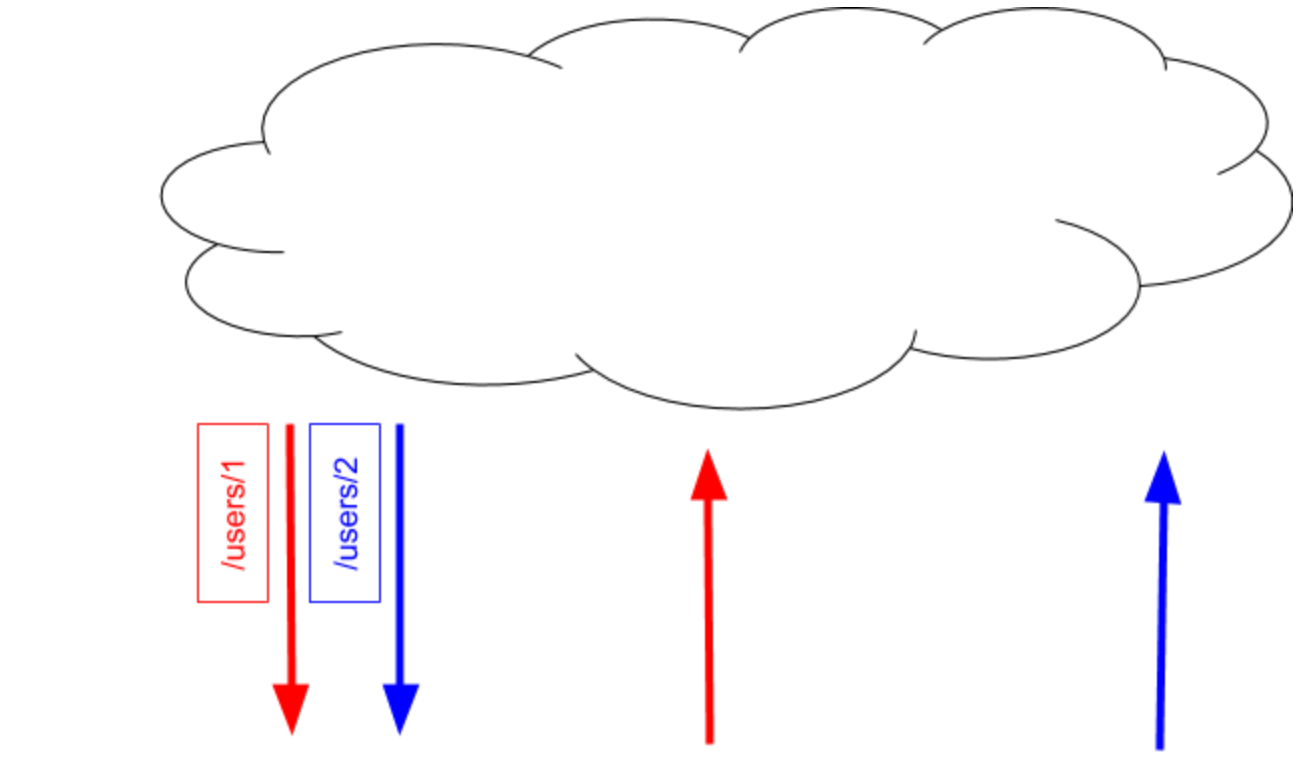
Asynchroniczność Dnia Powszedniego

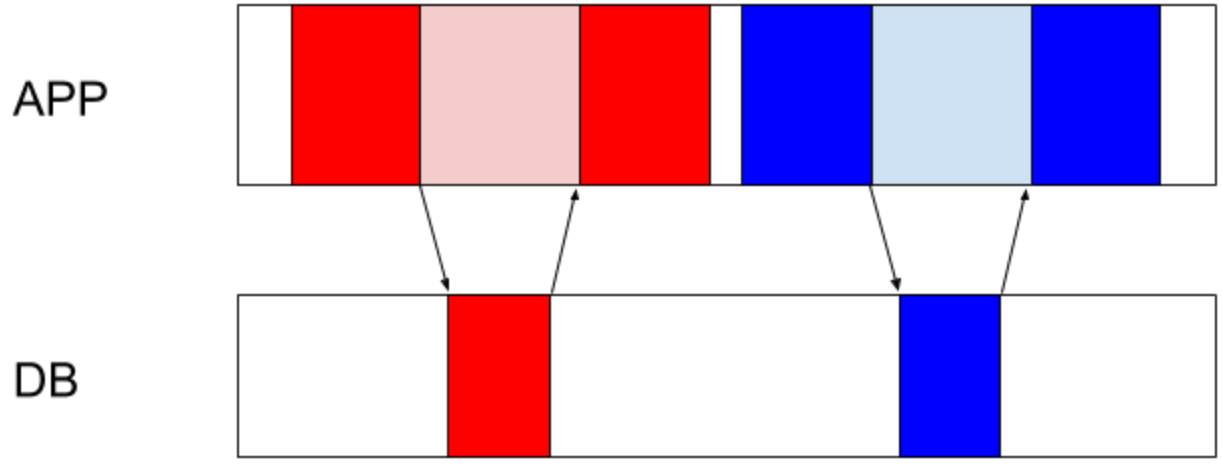
Konrad Hałas - PyWaw #67 - 08.05.2017

Agenda

- problem uzależnienia od operacji wejścia/wyjścia
- wstęp do asynchroniczności w Pythonie
- asynchroniczne serwisy webowe
- przykładowy projekt

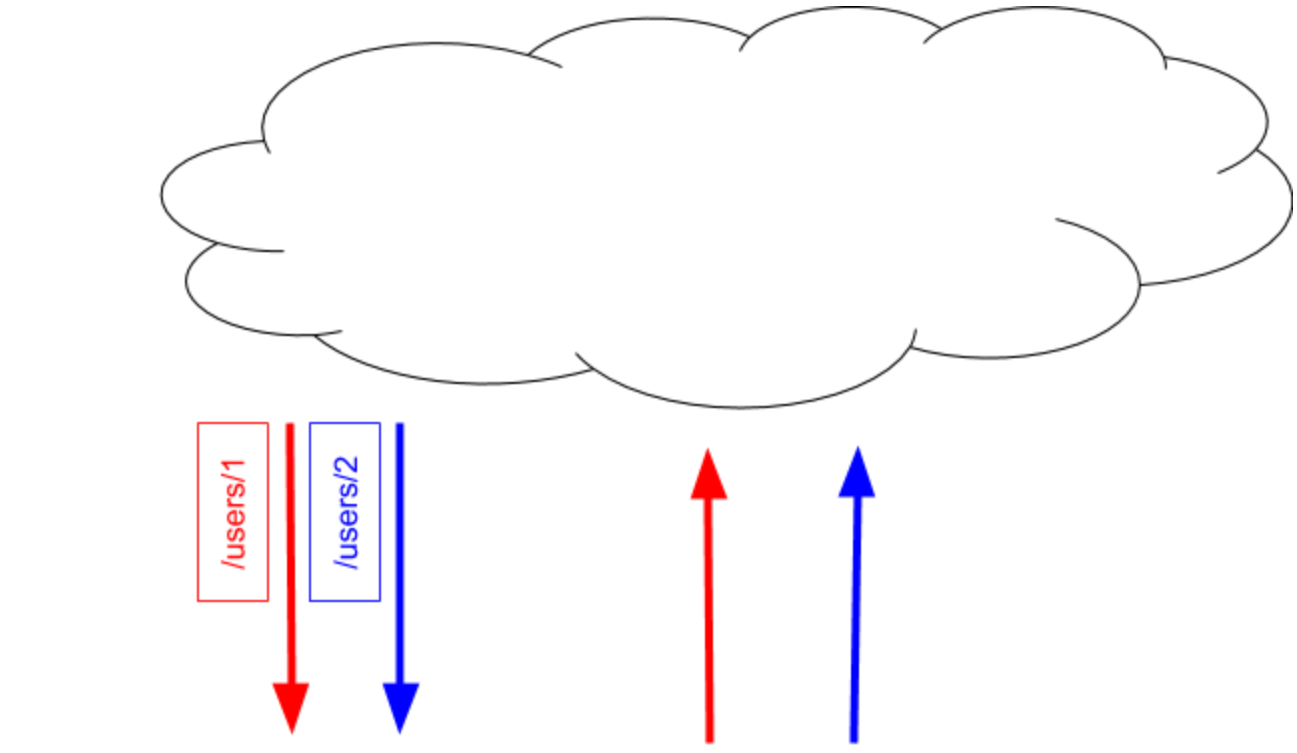
```python
@route('/users/<int:user_id>')
def user_details(user_id):
    user = UsersRepository.get_by_id(user_id=user_id)
    return Response(json=user.as_dict())
```

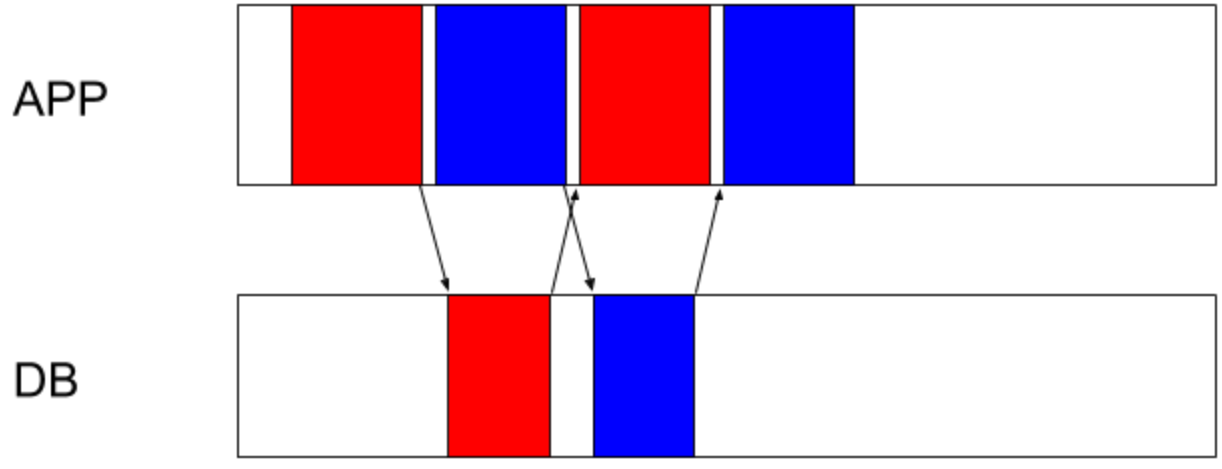

/users/1
/users/2


APP
DB

Rozwiązania

- procesy
- wątki
- asynchroniczność


/users/1
/users/2


APP
DB

Asynchroniczność w Pythonie

```python
import asyncio


async def hello_world():
    print('Hello... ')
    await asyncio.sleep(2)
    print('world!')


loop = asyncio.get_event_loop()
loop.run_until_complete(hello_world())
loop.close()
```

```python
@route('/users/<int:user_id>')
def user_details(user_id):
    user = UsersRepository.get_by_id(user_id=user_id)
    return Response(json=user.as_dict())
```

```python
@route('/users/<int:user_id>')
async def user_details(user_id):
    user = await UsersRepository.get_by_id(user_id=user_id)
    return Response(json=user.as_dict())
```

Asynchroniczne frameworki webowe

- Aiohttp
- Sanic
- Japronto

```python
from aiohttp import web


async def hello(request):
    name = request.match_info.get('name')
    return web.Response(text=f'Hello, {name}')


app = web.Application()
app.router.add_get('/{name}', hello)
web.run_app(app=app, host='localhost', port=8000)
```

Asynchroniczni klienci baz danych

- PostgreSQL - aiopg, asyncpg, aioodbc
- MySQL - aiomysql, aioodbc
- SQLite - aioodbc
- MongoDB - Motor

Asynchroniczne ORM

- SQLAlchemy - aiopg.sa, aiomysql.sa
- peewee - peewee-async

Asynchroniczne...

- ElasticSearch - aioes
- memcached - aiomcache
- Redis - aioredis
- ...

Testowanie

```python
async def test_hello(client):
    response = await client.get('/PyWaw')

    assert response.status == 200
    assert await response.text() == 'Hello, PyWaw'
```

```python
class UsersRepository:
    def __init__(self, loop):
        self.db = AsyncIOMotorClient(io_loop=loop)

    async def get_by_id(self, user_id):
        return await self.db.find_one({'id': user_id})


async def test_get_user_by_id(loop):
    users_repository = UsersRepository(loop=loop)
    user = await users_repository.get_by_id(1)

    assert user is None
```

Przykład - feed.pywaw.org

Podsumowanie

- asynchroniczność w Pythonie nie gryzie
- asynchroniczny ekosystem rozwija się bardzo szybko
- asynchroniczność pozwala lepiej wykorzystać zasoby

Dziękuję!

Dodatek

Historia asynchroniczności w Pythonie

- 2001 - Python 2.2 - generatory i słowo kluczowe `yield`
- **2002 - Twisted - obiekty `Deferred`**
- 2006 - Python 2.5 - korutyny z wykorzystaniem `yield`
- 2012 - Python 3.3 - składnia `yield from`
- **2014 - Python 3.4 - moduł `asyncio`**
- 2015 - Python 3.5 - słowa kluczowe `async` i `await`
- 2016 - Python 3.6 - składnia `async for` i `async with`