

# Asyncio

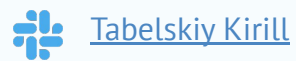


КИРИЛЛ ТАБЕЛЬСКИЙ



# КИРИЛЛ ТАБЕЛЬСКИЙ

Lightmap



[Tabelskiy Kirill](#)



# План занятия

1. [Что такое IO bound задачи?](#)
2. [Библиотека asyncio](#)
3. [Коррутины](#)



# Что такое IO bound задачи?

1. Чтение/запись в файл
2. Запись/чтение в/из stdout/stdin
3. Сетевые запросы

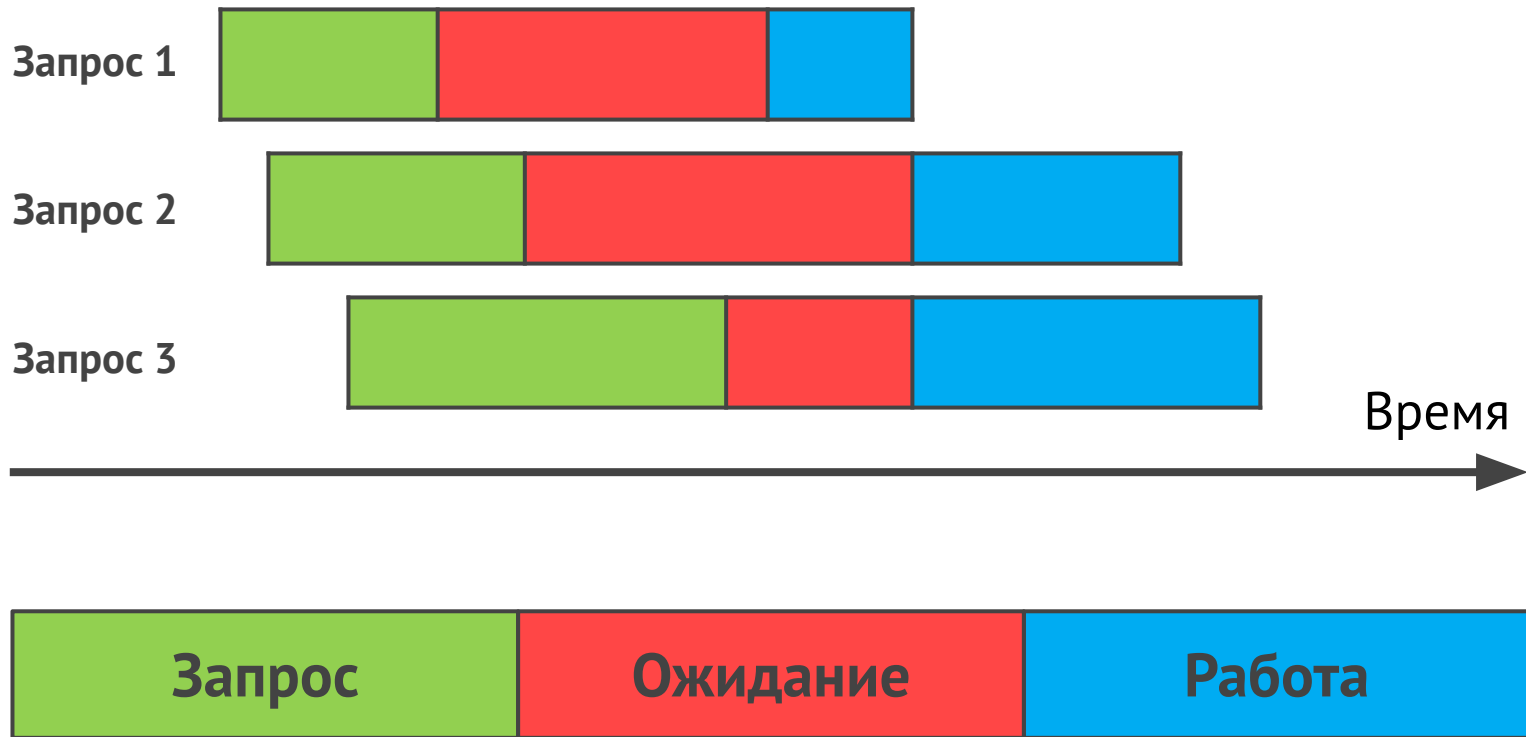
## Еще раз про IO bound

**Пример:** запросить данные из БД и обработать.



# Мультипоточный вариант

**Пример:** запросить данные из бд и обработать.

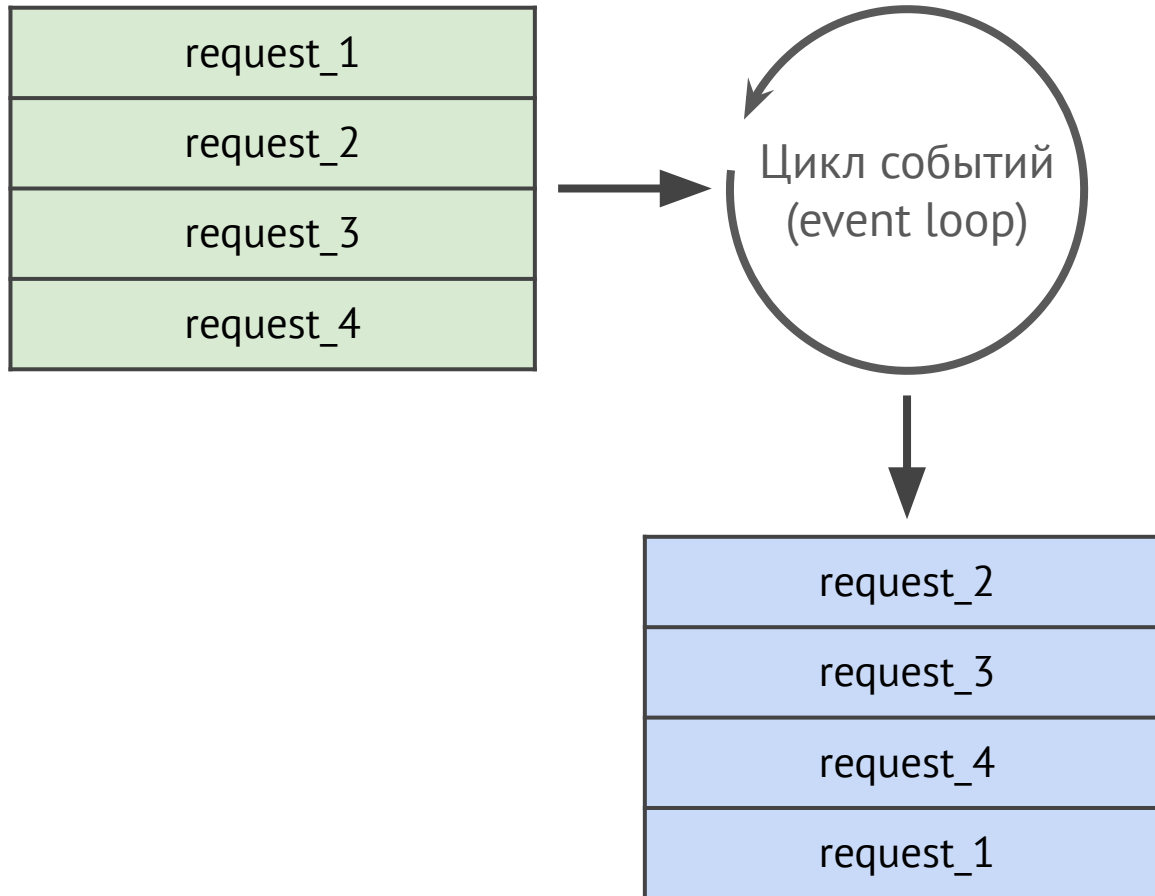




## Чем плохи потоки

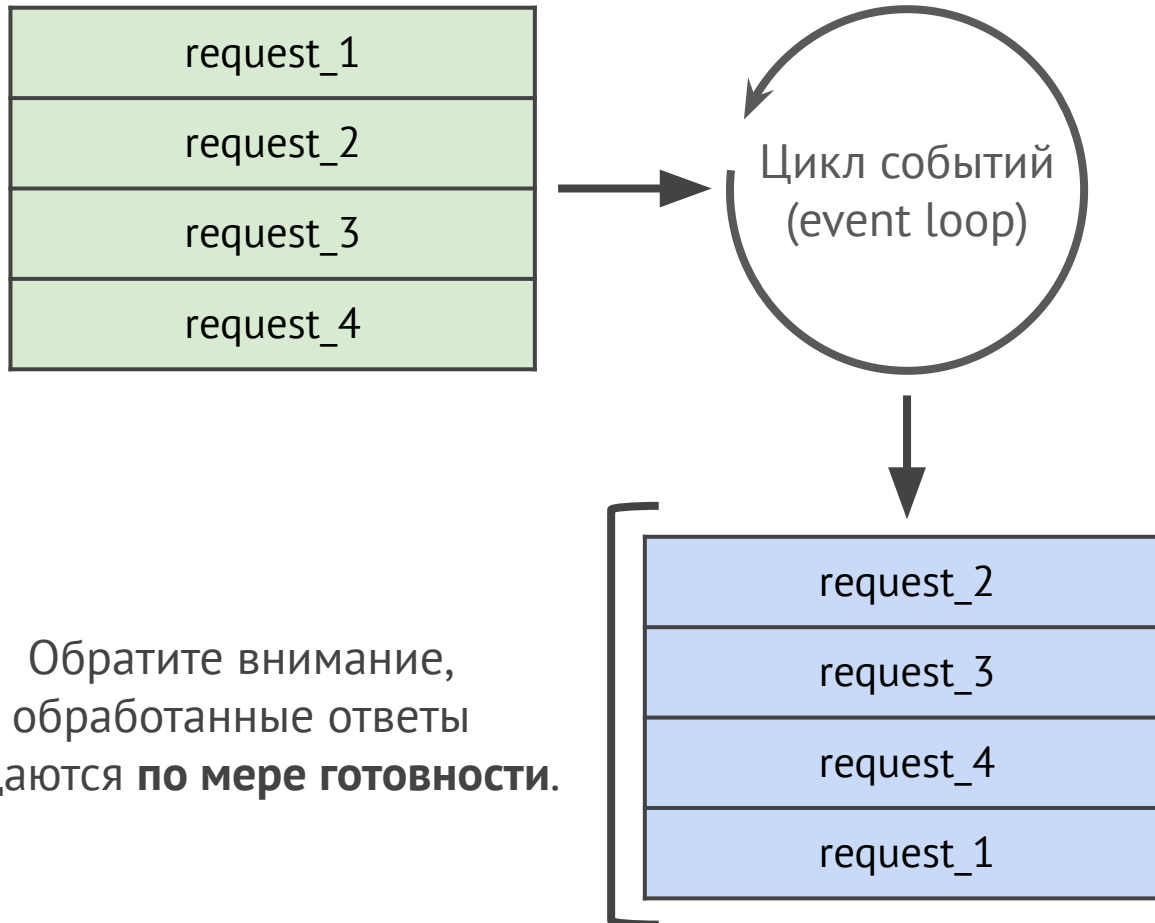
1. Максимум потоков ограничен. Его можно посмотреть/изменить здесь: `/proc/sys/kernel/threads-max`.
2. Не все объекты потокобезопасны.
3. Потоки потребляют ресурсы.

# Избавляемся от ожиданий





# Избавляемся от ожиданий



Обратите внимание,  
обработанные ответы  
выдаются **по мере готовности**.



# Бібліотека asyncio

## High-level API

Предназначен для создания приложений.

## Low-level API

Предназначен для создания библиотек.

# Пример


```
import async_vk_api
API = async_vk_api.mke_api(TOKEN)

async def request(**kwargs):
    ...
    response = await API.users.get(**kwargs)
    ...
    return response

async def main():
    coroutine1 = request(**kwargs)
    coroutine2 = request(**kwargs)
    coroutine3 = request(**kwargs)
    coroutine4 = request(**kwargs)

    responses = await asyncio.gather(coroutine1, coroutine2, coroutine3,
    coroutine4)

    ...
    asyncio.run(main())
```



Цикл событий  
(event loop)

The diagram shows a circular arrow representing an event loop. The text 'Цикл событий (event loop)' is written inside the circle. The arrow points clockwise, indicating a continuous cycle.

# Коррутины

```
async def request(url):  
    """Корутина отправляет GET запрос по HTTP  
    и возвращает ответ сервера в виде строки."""  
  
    session = aiohttp.ClientSession() #создаем сессию http  
    response = await session.get(url) #отправляем запрос и ждем ответ  
    text = await response.text() #ждем текстового представления ответа  
    await session.close() #ждем закрытия сессии  
    return text #возвращаем текстовое представление ответа  
  
async def main(): #точка входа в приложение  
    await request('http://google.com') #html  
  
if __name__ == '__main__':  
    asyncio.run(main()) #запуск асинхронного приложения
```

# Tasks

Таски – объекты, основанные на коррутинах, при их создании автоматически добавляются в event loop.

```
async def main():
    tasks = []
    task = asyncio.create_task(request('http://google.com'))
    tasks.append(task)
    task = asyncio.create_task(request('http://yandex.ru'))
    tasks.append(task)
    task = asyncio.create_task(request('http://rambler.ru'))
    tasks.append(task)
    task = asyncio.create_task(request('http://mail.ru'))
    tasks.append(task)
    for task in tasks:
        await task

if __name__ == '__main__':
    asyncio.run(main())
```

Вывод:

<http://mail.ru>

<http://google.com>

<http://rambler.ru>

<http://yandex.ru>



# Дополнительные материалы

[Документация](#)

[Как работать с aiohttp client API](#)



# Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задаём в чате Slack!
- Задачи можно сдавать по частям.
- Зачёт по домашней работе проставляется после того, как приняты **все задачи**.

**Задавайте вопросы и  
пишите отзыв о лекции!**

**Имя Фамилия**

