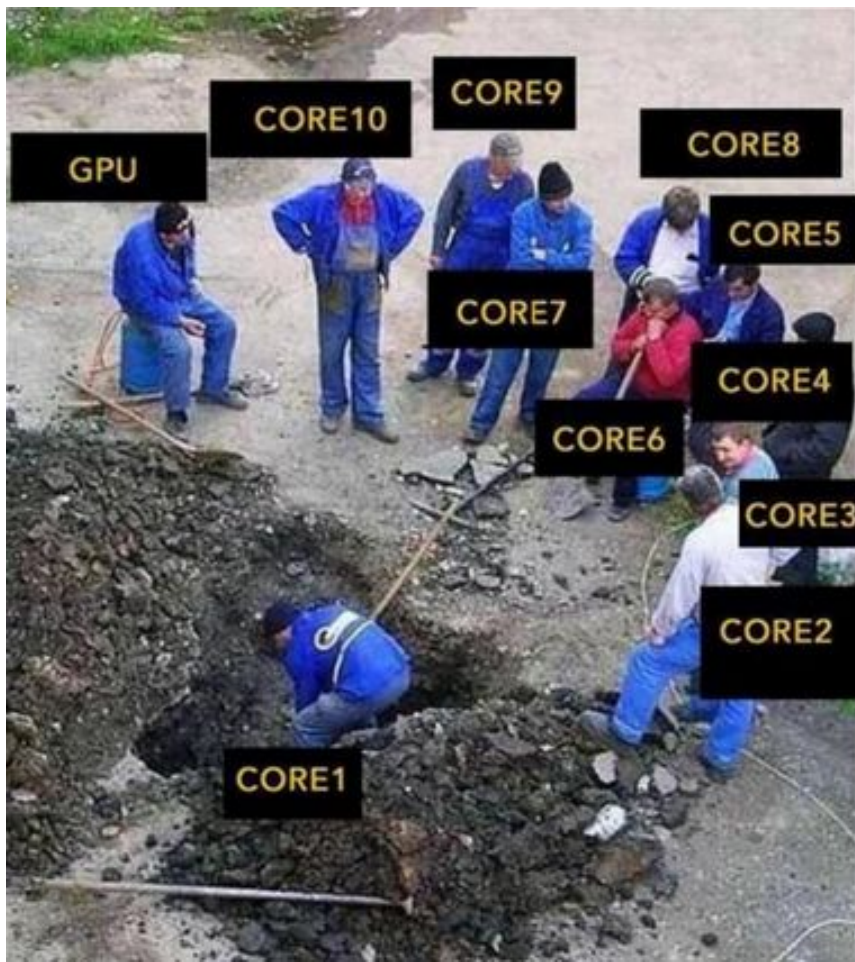


# Углубленный Python: Потоки async Воркшоп

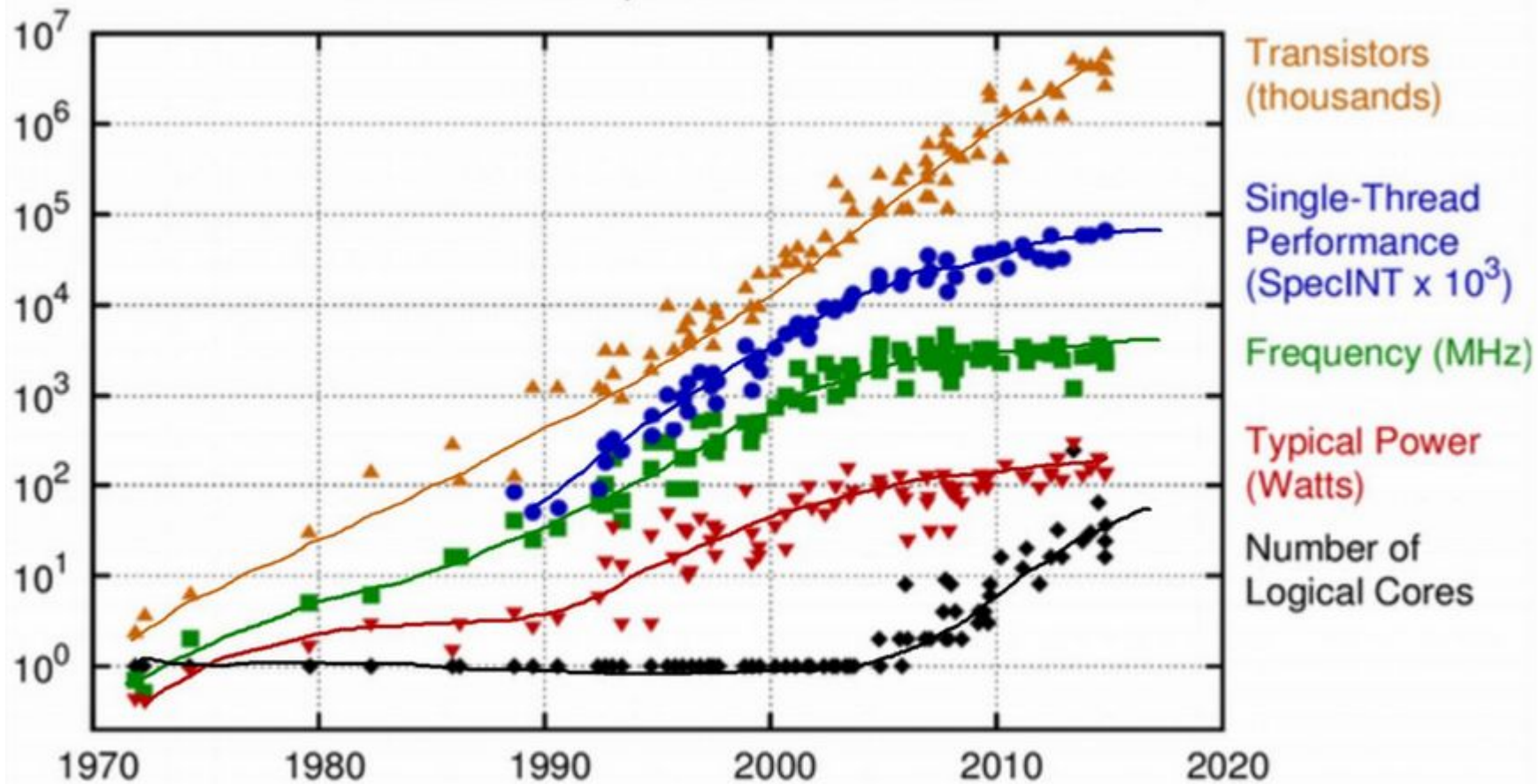
Тарасов Артём



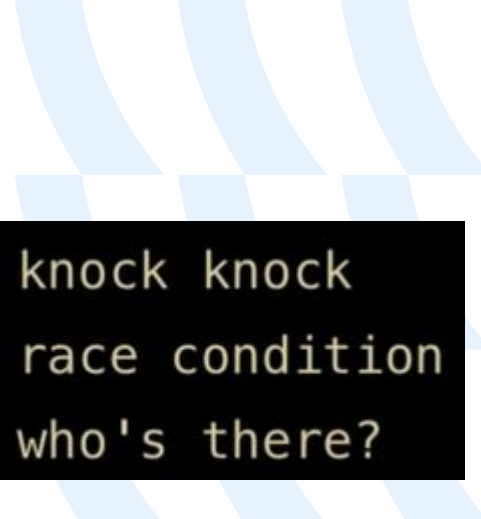
# Синхронный код



40 Years of Microprocessor Trend Data



# Race condition



```
A: knock knock  
A: race condition  
B: who's there?
```

# Асинхронность и фреймворки

Django, Flask и FastAPI

Ни один из них не имеет встроенной поддержки асинхронности, но ее можно добавить с помощью внешних библиотек.

Для реализации асинхронных вызовов FastAPI использует стандартный модуль `asyncio` и поддерживает асинхронные функции нативно, на уровне ядра. У Django и Flask такой поддержки нет, но они тоже умеют работать с асинхронными библиотеками `asyncio` или `aiohttp`. Так что разогнать свои веб-приложения с помощью асинхронности использование этих фреймворков не помешает.



Flask



FastAPI



# Flask

```
import aiohttp # импортируем библиотеку для работы с асинхронным примером
from flask import Flask # импортируем библиотеку для работы с фреймворком
app = Flask(__name__) # создаем приложение
@app.route("/") # объявляем страницу
async def main(): # создаем асинхронную функцию
    async with aiohttp.ClientSession() as session: # открываем асинхронную
клиентскую сессию
        async with session.get("https://www.example.com") as response: #
используя сессию делаем асинхронный запрос
            return response.text

if __name__ == "__main__": # объявляем секцию main
    app.run() # запускаем фреймворк по с настройками по умолчанию
```

# FastAPI

```
from fastapi import FastAPI # импортируем библиотеку для работы с  
фреймворком
```

```
import asyncio # импортируем библиотеку для работы с асинхронным примером  
app = FastAPI() # создаем приложение
```

```
@app.get("/") # объявляем страницу  
async def main(): # создаем асинхронную функцию  
    await asyncio.sleep(10) # с помощью асинхронной библиотеки запускаем  
наш пример  
    return {"message": "Hello World"}
```

```
if __name__ == "__main__": # объявляем секцию main  
    uvicorn.run(app) # запускаем фреймворк по с настройками по умолчанию
```

# Django channels

```
from channels.generic.websocket import AsyncWebsocketConsumer #библиотека  
для работы с асинхронностью
```

```
class MyConsumer (AsyncWebsocketConsumer):
```

```
    async def connect(self): # создаем асинхронный метод  
        await self.accept() # ожидаем  
        await asyncio.sleep (10) # выполняем действия  
        await self.send(text data="Hello World") # отправляем данные  
        await self.close() # закрываем
```



# Вопросики

- 1) Что такое event loop (цикл событий) в контексте асинхронного программирования?
- 2) Что такое корутина (coroutine) и как она отличается от обычной функции?
- 3) Какие методы управления асинхронными задачами предоставляет asyncio, и как они различаются?

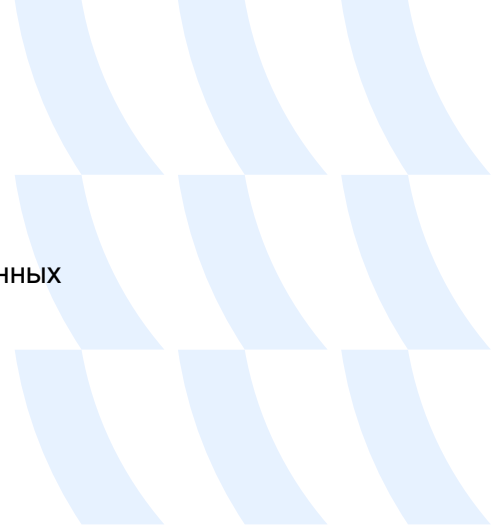


# Чаще всего применяется

**Сетевые операции.** При выполнении операций ввода-вывода, таких как запросы к базам данных или сетевые запросы, асинхронность позволяет выполнять множество таких операций параллельно, минимизируя время ожидания.

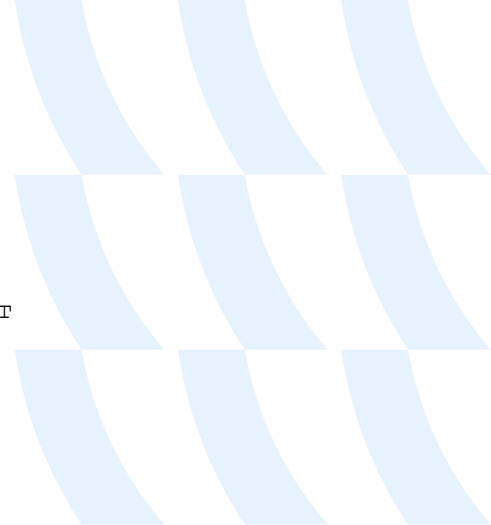
**Параллельные вычисления.** Асинхронность позволяет выполнять различные вычисления одновременно, что особенно полезно при обработке больших объемов данных или выполнении сложных вычислений.

**Пользовательский интерфейс.** В программировании с графическим интерфейсом асинхронность позволяет выполнять длительные операции (например, загрузку данных из сети) без блокирования пользовательского интерфейса.



# Асинхронщина используется в

- Сетевые приложения (`Twisted`, `aiohttp`, `requests`)
- Базы данных (с использованием библиотеки `SQLAlchemy Core`): Позволяет выполнять асинхронные запросы к базам данных.
- Асинхронная обработка данных: (`Celery`: Библиотека для асинхронной обработки задач, таких как отправка электронных писем, обработка изображений и другие асинхронные операции. `aiofiles`: Позволяет асинхронно работать с файлами. )
- Микросервисы:
- Многозадачные приложения:
- Разработка чат-приложений, ботов и веб-скраперов:



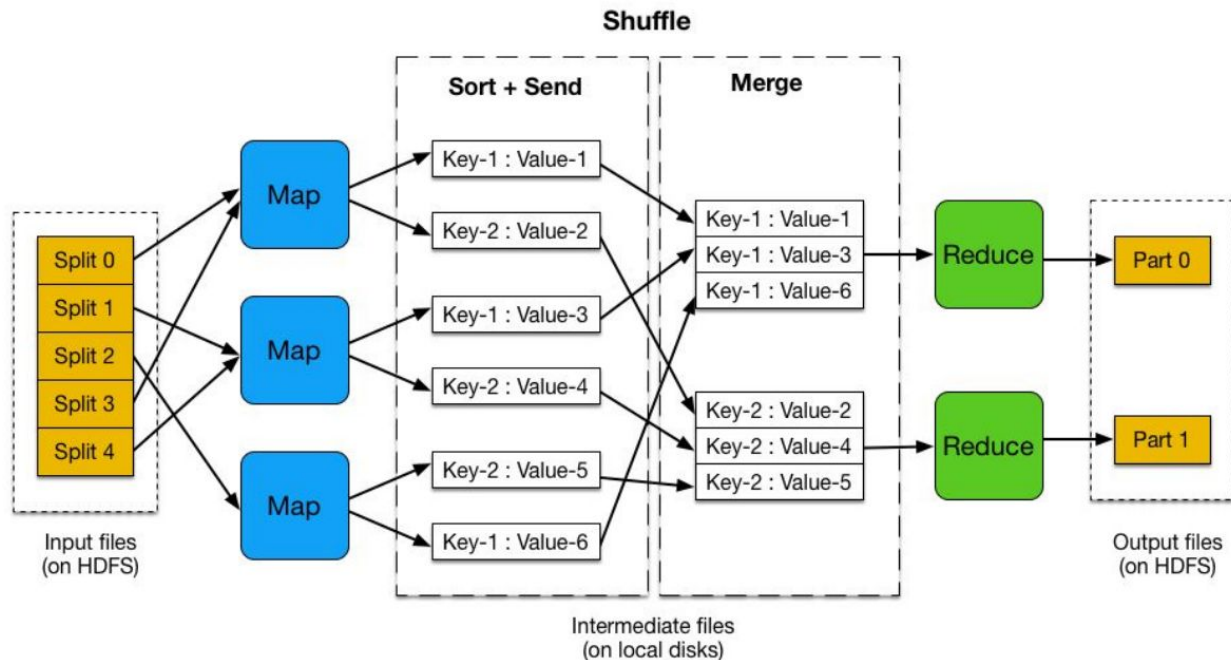
# Разминка

Напишите `thread_guard`



# MapReduce

- Модель распределенной обработки данных от Google
- Обрабатываем данные параллельно!
- Сложный процесс обработки можно декомпозировать на несколько простых



# Пример MapReduce в Python

```
from functools import reduce
from typing import Tuple
```

```
def find_longest_word(text: str) -> Tuple[str, int]:
    if not text: return "", 0
    longest word = max(text.split(), key=len)
    return longest word, len(longest word)
```

```
def merge_words_len(first: Tuple[str, int], second: Tuple[str, int]) ->
Tuple[str, int]:
    return first if first[1] > second[1] else second
```

```
# тексты уже разбиты на части (партиции)
texts = [
    "Hello world", "This is VK workshop", "The longest word in here is
abracadabra",
]
mapped_results = [find_longest_word(text) for text in texts]
print(*mapped_results)
result = reduce(merge_words_len, mapped_results)
# reduce Уменьшает массив до накопленного значения,
# применяя ЛЯМБДА к каждому значению и возвращая общее значение в
аккумулятор.
print(result)
```

```
>>> ('Hello', 5) ('workshop', 8) ('abracadabra', 11)
>>> ('abracadabra', 11)
```

Решаем задачу

<https://colab.research.google.com/drive/1hqgE577DtS7BXevKaq1xdtQhSpj4Hj3z?usp=sharing>





Q&A.  
С радостью  
отвечу на ваши  
вопросы.