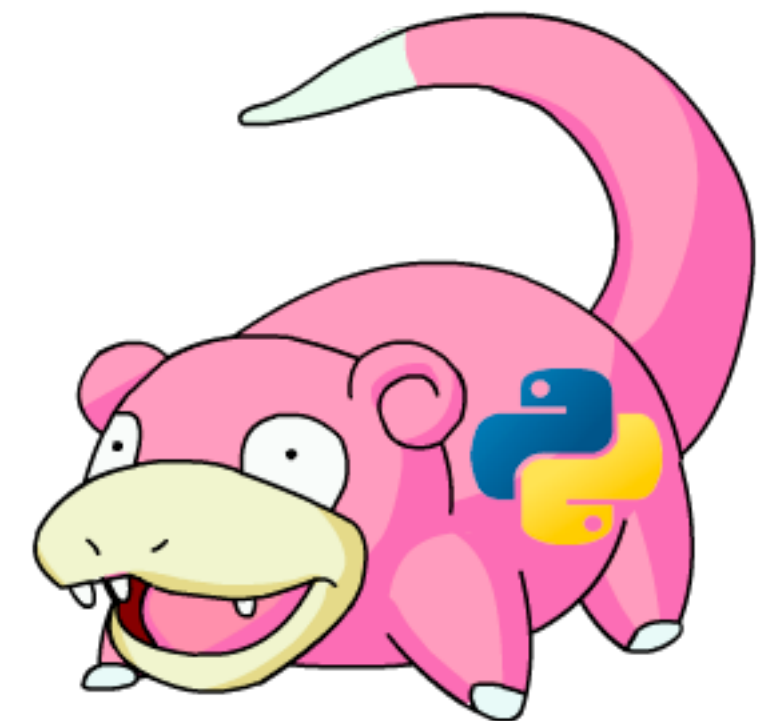


# Лекция №3.

Часть 2.

Распараллеливание процессов

01.12.2020



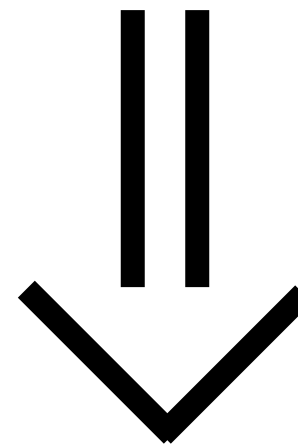
# План

- Что такое параллелизм. Синхронные и асинхронные процессы
- Поток в Python
- Что такое GIL. Ограничения Python при реализации параллельных вычислений
- Инструменты для ускорения программ на Python

# Параллелизм

**Синхронное** выполнение программы подразумевает последовательное выполнение операций

**Асинхронное** – предполагает возможность независимого выполнения задач.



**Параллельное выполнение программ** позволяет разбить код программы на блоки, работающие параллельно (одновременно!)

# Зачем нужен параллелизм?

**Параллельное программирование** служит для создания программ, эффективно использующих вычислительные ресурсы за счет одновременного исполнения кода на нескольких вычислительных узлах

Если ваши скрипты на питоне работают медленнее, чем хотелось бы, их можно ускорить, запуская параллельные задачи.


В общем, параллельные вычисления позволяют выполнять множество вычислений одновременно таким образом сокращая время выполнения программы.

# Кратко о многопоточности

**Поток** — это самая маленькая единица выполнения, которая может быть выполнена в операционной системе. Потоки относятся к наивысшему уровню кода, который может выполнять ваш процессор.

Технология **Hyper-Threading** («гиперпоточность», «многопоточность») позволяет одному ядру ЦП выступать в качестве двух ядер, ускоряя выполнение конкретной программы или приложения. Основная идея многопоточности заключается в достижении параллелизма путем разделения процесса на несколько потоков.



 = логический процессор

# Ограничения Python при реализации параллельных вычислений

в Python используется **глобальная блокировка интерпретатора (Global Interpreter Lock — GIL)**, накладывающая некоторые ограничения на потоки. А именно, нельзя использовать несколько процессоров одновременно.

GIL не позволяет в одном интерпретаторе Python эффективно использовать больше одного потока. Защитники GIL утверждают, что однопоточные программы при наличии GIL работают намного эффективнее. Но наличие GIL означает, что параллельные вычисления с использованием множества потоков и общей памяти невозможны. А это достаточно сильное ограничение в современном многоядерном мире.

**Как ускорить?**

# JIT компиляция

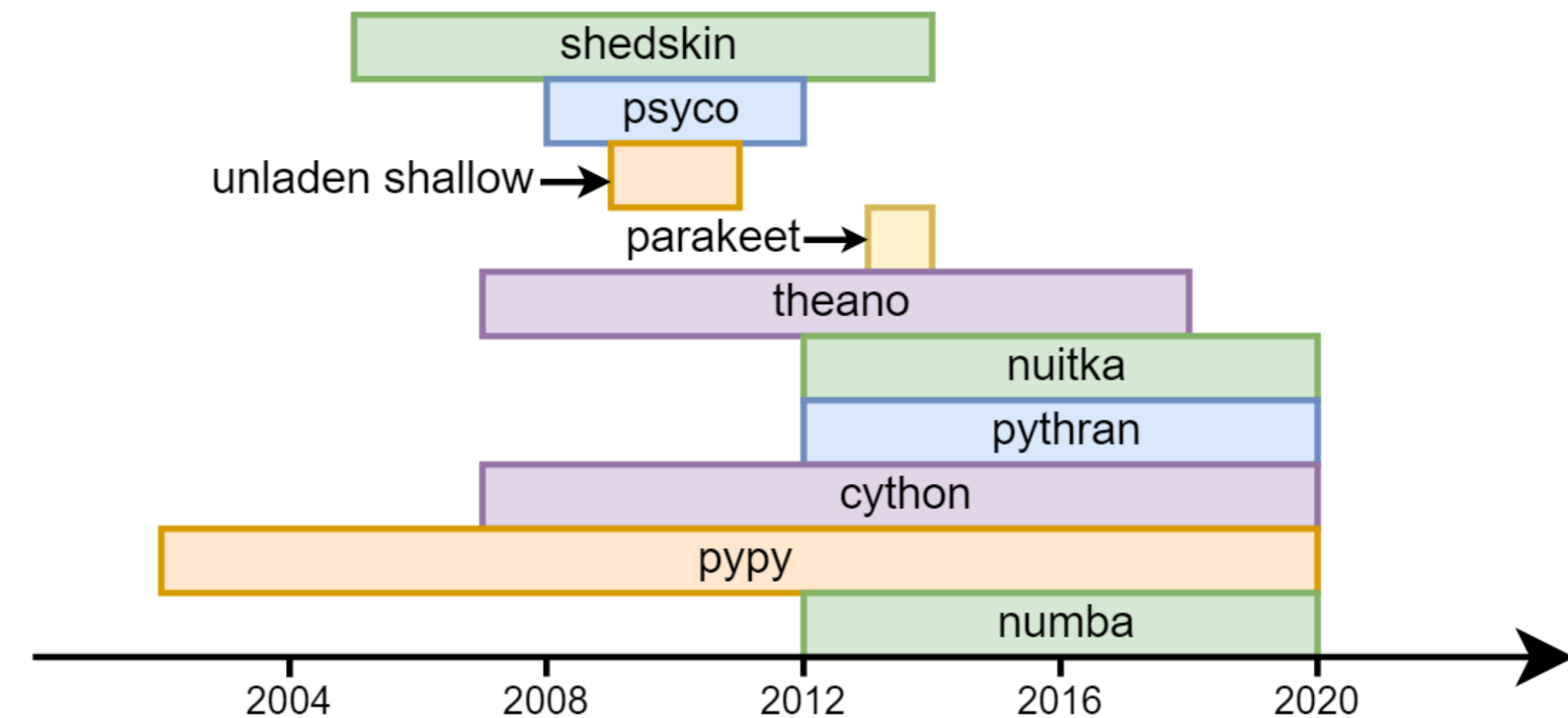
**JIT (Just in Time) компиляция** - это вид динамической компиляции, созданный для увеличения производительности программных систем, использующих байт-код, путем компиляции байт-кода в машинный код непосредственно во время работы программы.

JIT можно рассматривать как способ оптимизации



# Инструменты для ускорения работы Python

- PyPy - альтернативный интерпретатор
- Pyston - альтернативный интерпретатор
- Nuitka - транслирует код в C++
- Cython - гибрид Си + Python
- **Numba** - использует jit-компиляцию
- **Joblib** - использует мемоизацию(кэширование)



В практической части занятия разберем на примере использование **Numba** и **Joblib**

Более продвинутые методы оптимизации — в духе CPython, — а также альтернативные интерпретаторы — в духе PyPy, Jython и Iron Python — не будут рассматриваться в практике сознательно, т.к. чаще всего не поддерживают основные библиотеки для data science

# Инструменты для ускорения работы Python

Сравнение скорости перемножения матриц в Python:

method	time(ms)*	version
numba	9.77	0.16.0
np.outer	9.79	1.9.1
cython	10.1	0.21.2
parakeet	11.6	0.23.2
pyru	16.36	2.4.0
np.einsum	16.6	1.9.1
theano	17.4	0.6.0

\* less time = faster

# Numba

Все операторы, функции и классы делятся в отношении нумбы на две части: те, которые нумба «понимает» и те, которые она «не понимает».

В документации по numba есть два таких списка (с примерами):

- подмножество функционала [питона](#), знакомое нумбе и
- подмножество функционала [numpy](#), знакомое нумбе.

**Понимает:** list, dict, tuple, str, bytes и numpy массивы

**Не понимает:** никакие другие библиотеки, в том числе Pandas, Scipy

# Numba

Чтобы ускорить функцию, надо перед её определением вписать декоратор **@njit**:

```
from numba import njit

@njit
def f(n):
    s = 0.
    for i in range(n):
        s += sqrt(i)
    return s
```

Ускорение в **40** раз.

Подробнее разберем этот и другие примеры в практической части занятия

# Полезные ссылки

[Как устроен GIL в Python](#)

[Работа с потоками в Python](#)

[Python + Numba быстрее Си? серьезно?](#)