

---

---

# Concurrency / Parallelism

— Лекция 1 —

---

---

# Что сегодня обсуждаем

- Постановка проблемы
  - CPU- и IO- bound задачи
- Как люди научились решать такие проблемы
  - Примитивы на уровне ОС
- Как Python экосистема укладывается в эти примитивы
- В каких случаях что использовать

# Постановка проблемы

- Хотим делать много вещей одновременно
- МНОГО вещей.
- ЕЩЕ БОЛЬШЕ ВЕЩЕЙ
- и ПОБЫСТРЕЕ
- хотим делать



# Много каких вещей?

- Нагружаем CPU
  - Числомолотилка
  - Енкодинг - декодинг
  - Заталкивание байтов в GPU память
- Такие задачи называются CPU bound

# Много каких вещей?

- Блокирующие IO задачи
  - Дисковая подсистема
  - Сетевое взаимодействие
- В каждой задаче ЖДЕМ какого-то внешнего события
- Такие задачи называются IO bound

**Другими словами,  
“параллельность” бывает разная**

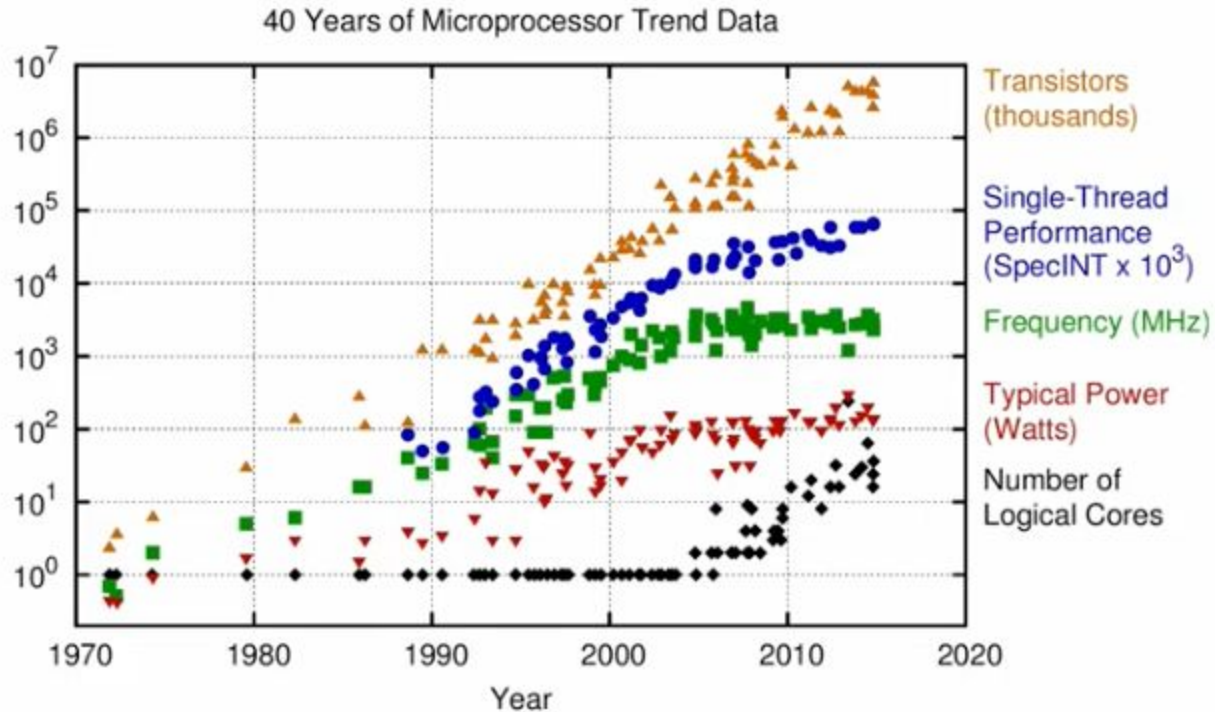
**Подходы для ее достижения  
– тоже разные**



# Concurrency VS Parallelism

- Youtube – Rob Pike – concurrency is not parallelism
  - <https://www.youtube.com/watch?v=oV9rvDlKEg>
- <https://www.geeksforgeeks.org/difference-between-concurrency-and-parallelism/>
- Операционная система – это про что?
- Железо – это про что?

# Отступление про железо и “железный” параллелизм



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten  
New plot and data collected for 2010-2015 by K. Rupp

Украдено у Ашот Варданян – [Peta-Scale software in 2021](#)

# Levels of Parallelism

Available in 2021

- **Multi-Processing** = separate address space
- **Multi-Threading** = separate stack-frames
  - 256 threads in a server
  - 8 threads in a laptop

# Levels of Parallelism

Available in 2021

- **Multi-Processing** = separate address space
- **Multi-Threading** = separate stack-frames
- Multi-Data (SIMD) = separate operation arguments
  - 512 bits/argument on a server
  - 256 bits/argument on a server

# Levels of Parallelism

Available in 2021

- Multi-Device = separate memory pool
  - GPUs, FPGAs, ASICs
  - up-to  $128/16 = 8$  extension cards/server
- **Multi-Processing** = separate address space
- **Multi-Threading** = separate stack-frames
- Multi-Data (SIMD) = separate operation arguments

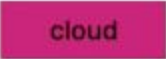


# Levels of Parallelism

Available in 2021

- Multi-Node = separate machines:
  - Simultaneous Parallelism **Supercomputers**
  - Stream Processing **Datacenters**
- Multi-Device = separate memory pool
- **Multi-Processing** = separate address space
- **Multi-Threading** = separate stack-frames
- Multi-Data (SIMD) = separate operation arguments

# Exploiting Parallelism

In 2021

- Multi-Node @ any language
- Multi-Device @ CUDA, OpenCL, SyCL C++, GLSL
- Multi-Processing @ any language 
- Multi-Threading @ most languages 
- Multi-Data (SIMD) @ in Assemblers & in C 



# **Multi-processing**

# **Multi-threading**

# Что придумали на уровне ОС – процессы

- Главный кирпичик изоляции
  - Изоляция стеита процессора (регистров)
  - Изоляция адресного пространства (виртуальная память, пара слов)
  - Изоляция окружения
- Процессы шедулятся системным планировщиком
- Демонстрация на примере Linux
  - Pid, ppid, gid – иерархическая структура
    - Каждый процесс имеет родителя – как это происходит?
  - /proc/<pid>
  - <https://en.wikipedia.org/wiki/Cgroups> – почитайте сами:)

# Создание новых процессов – fork

- Родитель “форкается” – fork
  - Делается независимая копия, 1 в 1
    - **У копии все свое**
      - Память
      - Стейт процессора
      - ...
  - И родитель, и копия продолжают работу с момента форка
    - <https://www.geeksforgeeks.org/fork-system-call/>
- Пример
  - Интерпретатор python делает ребенка через fork
    - Ребенок – полная копия родителя – модули импортированы, ...

# Создание новых процессов – spawn

- spawn = fork + exec
  - Семейство системных вызовов `exec*` заменяет образ текущего процесса образом какого-то другого процесса из файла
    - При этом теряется **все состояние**
- Пример
  - Интерпретатор python создает ребенка через spawn
    - Ребенок – свежий интерпретатор, который будет импортировать модули **заново**
    - Это очень важное различие, которое проявляется на linux vs windows + mac
      - Пример на семинаре

# Оверхед создания процесса

- Fork – тяжелая операция
  - Нужно
    - создать “хендл” процесса
    - скопировать стейт процессора
    - скопировать дескрипторы
    - **скопировать всю память**
- Умные люди придумали Copy-On-Write (COW)
  - <https://en.wikipedia.org/wiki/Copy-on-write>
  - Превосходно используется в “железных” языках
    - константная память
  - Позволяет форкать детей общим объемом **аллоцированной** памяти больше, чем RAM
  - В питоне страницы быстро вымываются из-за счетчиков ссылок
- Windows привет :)

# Оверхед создания процесса

- `spawn` – сверх дорогая операция

# Потоки

- Все тоже самое, только
  - Общее пространство памяти

# потоки VS процессы

- Потоки

- дешевле
- Используют общую память
  - Больше синхронизации
- “Взрыв” потока → “взрыв” всего процесса

- Процессы

- дороже
- Используют изолированную память
  - IPC только через файлы / примитивы ОС (файлы, сокет, pipes, named shared memory, сигналы)
  - Меньше необходимости в синхронизации
- Процесс имеет независимый жизненный цикл



# Potential performance improvements vs single-threaded app on AMD Epyc 7742

- Multi-Node (cluster size)x
- Multi-Device 100x faster GEMM on Nvidia A100
- Multi-Processing **50x faster simulation on 128 threads**
- Multi-Threading **100x faster simulation on 128 threads**
- Multi-Data (SIMD) 4x faster encoding/decoding

Why different?

**Процессы**

**concurrency**

**ИЛИ**

**parallelism**

**?**

# Причем здесь Питон?

- Питон имеет примитивы для создания **честных системных потоков**
- Питон имеет примитивы для создания **честных процессов**

Примеры – на семинаре

**КОНЕЦ**



**Знакомьтесь, GIL**

```
>>> import that
```

The Unwritten Rules of Python

1. You do not talk about the GIL.
  2. You do NOT talk about the GIL.
  3. Don't even mention the GIL. No seriously.
- ...

# “Так сложилось исторически...”

– цитаты великих людей

- Python – язык программирования
- CPython – референсная имплементация
- Спецификация и имплементация взаимосвязаны
  - CPython влияет на Python (`__slots__`)
- CPython *интересно* реализовал подход к многопоточности
  - Unfortunately, since the GIL exists, other features have grown to depend on the guarantees that it enforces. This makes it hard to remove the GIL without breaking many official and unofficial Python packages and modules.



# Cpython – виртуальная машина

- Import dis
- Опкоды
- ceval.c

# GIL

- <https://wiki.python.org/moin/GlobalInterpreterLock>
- David Beazley
  - человек -легенда в мире python)
  - <https://ephraim.net/wp-content/uploads/2014/12/UnderstandingGIL.pdf>
- Что важно понимать
  - Каждая итерация eval-лупа – под GIL-ом
  - GIL “снимается”
    - при блокирующих C-вызовах
    - Бинарные расширения могут снимать GIL (numpy)

# Мультипроцессинг и Python

- Необходимость сериализации объектов при общении между процессами
- Способ запуска
  - Linux – fork
  - Windows, mac – spawn
- У каждого процесса – **свой** GIL
  - Позволяет утилизировать процессор

# IO vs CPU bound и Python

- IO bound
  - Потоки, если не нужно иметь отдельный жизненный цикл
  - Процессы, если нужно
- CPU bound
  - Pure-python
    - процессы
  - Бинарные расширения
    - Скорее всего, отпускают GIL сами, поэтому делаем треды
      - Пример: numpy

**КОНЕЦ**

