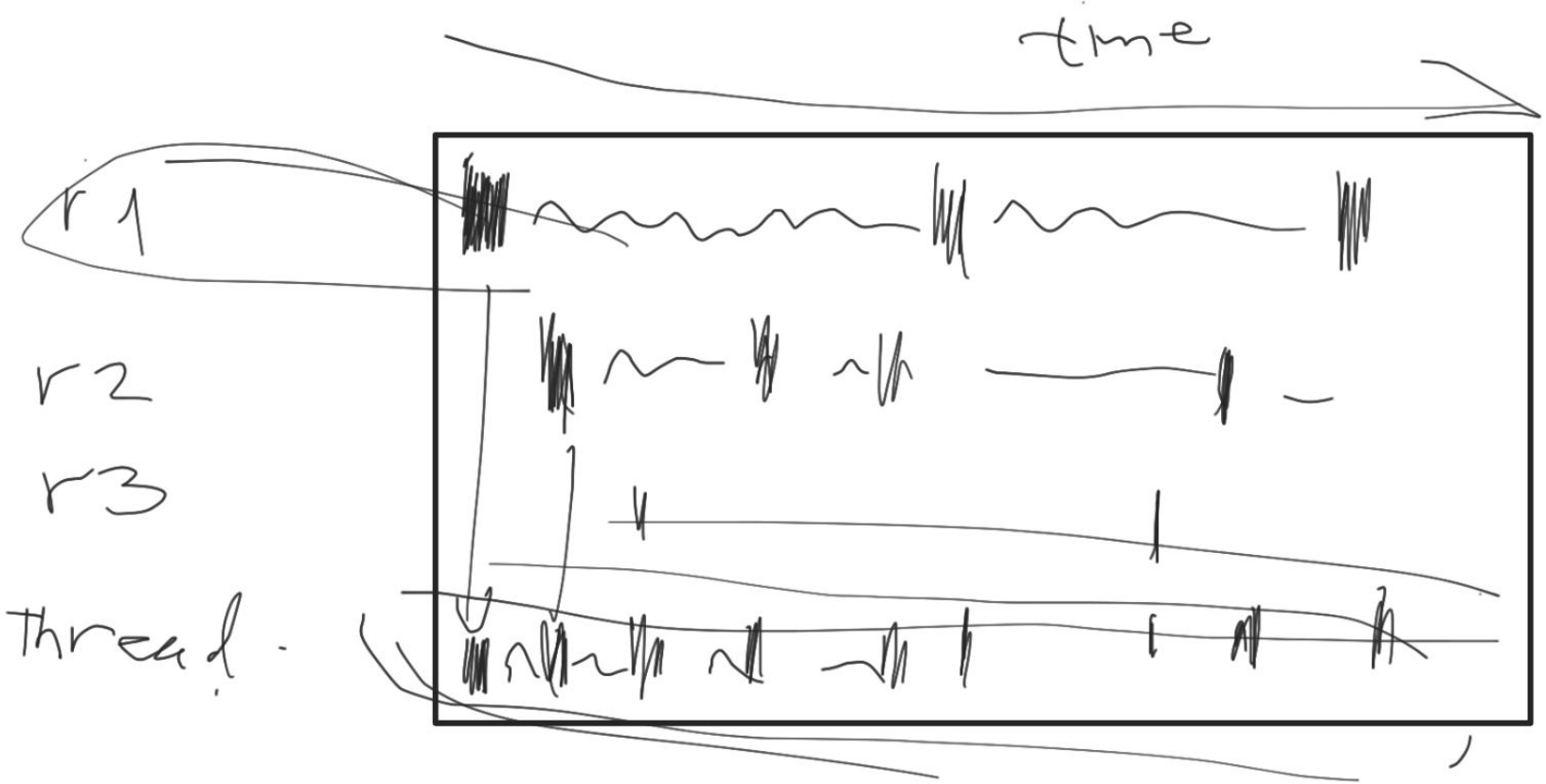
Async / Await / AsyncIO

Лекция 2

Recap: 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


time
r1
r2
r3
thread

Можно ли обойтись без потоков и процессов?

- Гуглите презентации David Beazley
- Демонстрация коллбеки
- Countup, Countdown
  - threading
  - custom scheduler
    - Разбиваем на микроюниты
  - Выносим sleep в шедулер
- Producer / Consumer
  - Threading
  - Custom scheduler
  - Проблема остановки очереди

Takeaways

- IoC (inversion of control)
  - Не мы сами вызываем функции
  - Это делает за нас шедулер!
- Но это происходит не магически
  - Мы должно помочь шедулеру
    - Разбить на мы должны сами разбить функции на маленькие куски
  - Это называется https://en.wikipedia.org/wiki/Cooperative_multitasking
- Гуглите callback hell
- Можно сделать, но тяжело поддерживать / понимать

Генераторные функции!

- Позволяют “перезайти” в функции
-

Материалы

https://peps.python.org/pep-0492/

https://stackoverflow.com/questions/49005651/how-does-asyncio-actually-work

David Beazley!

Async / Await

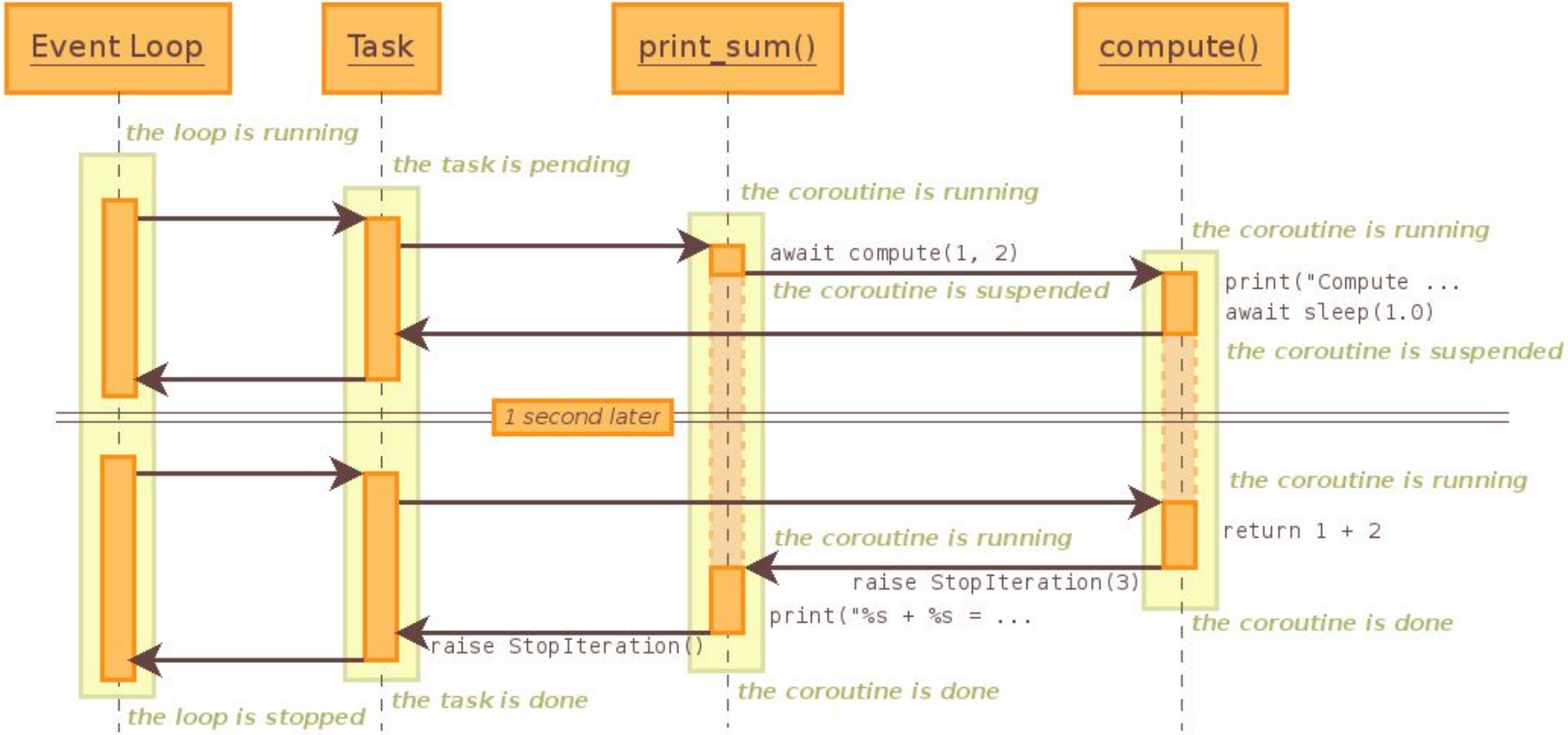
- Линейный код
  - 99% блокировок, которые нужны в multithreading, НЕ НУЖНЫ
-

Сложности

- Что делать, если используем async, но event-loop / Не всегда есть asyncio совместимые либы
- Как обойти?
- Как запустить синхронную функцию асинхронно?
- Как запустить асинхронную функцию синхронно?
- Два разных синтаксиса – можно ли писать один раз код, который будет запускаться везде?

Проблемы асинхронного подхода

- Нельзя мешать CPU и IO bound задачи
- IoC подразумевает, что IoC контейнер умеет хендлить все необходимые типы асинхронных событий (например, поддержка select на уровне ОС), поддержка фс, ...


Event Loop
Task
print_sum()
compute()
the loop is running
the task is pending
the coroutine is running
the coroutine is running
await compute(1, 2)
the coroutine is suspended
print("Compute ...
await sleep(1.0)
the coroutine is suspended
1 second later
the coroutine is running
return 1 + 2
the coroutine is running
raise StopIteration(3)
print("%s + %s = ...
the coroutine is done
raise StopIteration()
the coroutine is done
the task is done
the loop is stopped