Fintech Python

Лекция 7

Пара слов обо мне

- Занимаюсь чат-ботами
- Учусь на матмехе и в ШАДе

Шибаев Александр

Процессы

Процесс — программа, которая исполняется в данный момент и обладает набором ресурсов:

- образом исполняемого машинного кода
- памятью
- открытые файлы, сокеты

Процессы изолированы друг от друга операционной системой

Создание новых процессов

python script.py - один процесс

```
In [86]: import os

         print('Before fork')

         os.fork()

         print("After fork")
```

```
Before fork
After fork
After fork
```

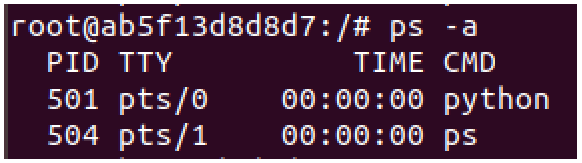
```
In [101]: import os

          os.fork()
          os.fork()
          os.fork()

          print("Hello, world")
```

```
Hello, world
Hello, world
Hello, world
Hello, world
Hello, world
Hello, world
Hello, world
Hello, world
```

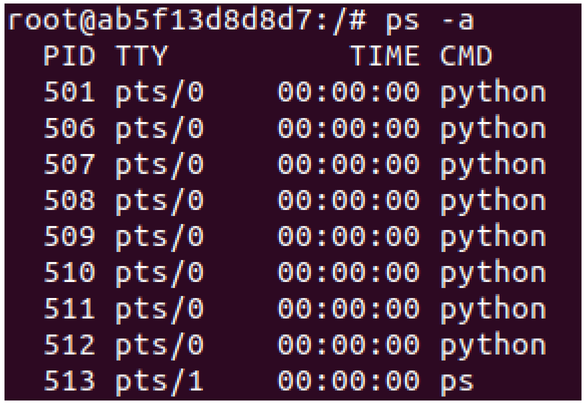
In [ ]:
```python
# script.py
import os
import time

time.sleep(10)
os.fork()
os.fork()
os.fork()
time.sleep(60)
```

`python script.py` - в первом терминале

`ps -a` - во втором



```
root@ab5f13d8d8d7:/# ps -a
  PID TTY          TIME CMD
  501 pts/0    00:00:00 python
  504 pts/1    00:00:00 ps
```



```
root@ab5f13d8d8d7:/# ps -a
  PID TTY          TIME CMD
  501 pts/0    00:00:00 python
  506 pts/0    00:00:00 python
  507 pts/0    00:00:00 python
  508 pts/0    00:00:00 python
  509 pts/0    00:00:00 python
  510 pts/0    00:00:00 python
  511 pts/0    00:00:00 python
  512 pts/0    00:00:00 python
  513 pts/1    00:00:00 ps
```

In [102]:
```
import os
import time

x = 1

if os.fork():  # Возвращает 0 в дочернем процессе и pid ребенка в родительском
    x += 1
    time.sleep(1)
    print(f'Parent: {x}')
else:
    x += 2
    time.sleep(1)
    print(f'Child: {x}')
```

```
Parent: 2
Child: 3
```

Copy on write

- Пока читаем - используем старые данные
- При записи копируем

Есть один нюанс…

multiprocessing

https://docs.python.org/3/library/multiprocessing.html (https://docs.python.org/3/library/multiprocessing.html)

In [ ]:
```
from multiprocessing import Process
import os

def info(title):
    print(title)
    print('parent process:', os.getppid())
    print('process id:', os.getpid())

def f(name):
    info('function f')
    print('hello', name)

if __name__ == '__main__':
    info('main line')
    p = Process(target=f, args=('bob',))
    p.start()
    p.join()
```

```
In [5]: from multiprocessing import Process, Queue


        def worker(job: int, queue: Queue):
            queue.put(job)


        queue = Queue()
        processes = [Process(target=worker, args=(i, queue)) for i in range(30)]
        for p in processes:
            p.start()
        for p in processes:
            p.join()

        result = [queue.get() for i in range(30)]
        print(result)
```

```
[1, 0, 3, 2, 4, 5, 6, 10, 7, 8, 11, 12, 9, 14, 13, 15, 17, 18, 20, 19, 16, 21,
22, 23, 24, 26, 25, 28, 27, 29]
```

In [117]:
```python
import time
import numpy as np
from multiprocessing import Process
from multiprocessing.sharedctypes import Value, RawArray

def worker(array, idx, value):
    time.sleep(0.1)
    array[idx] = value

def main():
    array = RawArray('i', [0] * 10)
    processes = [
        Process(target=worker, args=(array, i, i * 2)) for i in range(5)
    ]
    for p in processes:
        p.start()
    for p in processes:
        p.join()

    print(list(array))

main()
```

```
[0, 2, 4, 6, 8, 0, 0, 0, 0, 0]
```

```
In [6]: size = 100_000_000
        arr = [1] * size
```

```
In [7]: %%time
        sum(arr)
```

```
CPU times: user 1.26 s, sys: 4 ms, total: 1.26 s
Wall time: 2.15 s
```

```
Out[7]: 100000000
```

```
In [8]: from multiprocessing import Pool

        process_count = os.cpu_count()
        part_size = size // process_count

        process_count

Out[8]: 8
```

```
In [9]: %%time
        with Pool(process_count) as p:
            p.map(
                sum,
                (arr[i * part_size: (i+1) * part_size] for i in range(process_count))
            )

        CPU times: user 3.3 s, sys: 762 ms, total: 4.07 s
        Wall time: 7.62 s
```

Проблема в большом объеме данных

```
In [10]: def get_sum(size):
             return sum([1] * size)
```

```
In [11]: %%time
         with Pool(process_count) as p:
             p.map(get_sum, [part_size] * process_count)
```

```
CPU times: user 5.72 ms, sys: 195 ms, total: 201 ms
Wall time: 1.21 s
```

```
In [12]: with Pool(process_count) as p:
             %timeit p.map(get_sum, [part_size] * process_count)
```

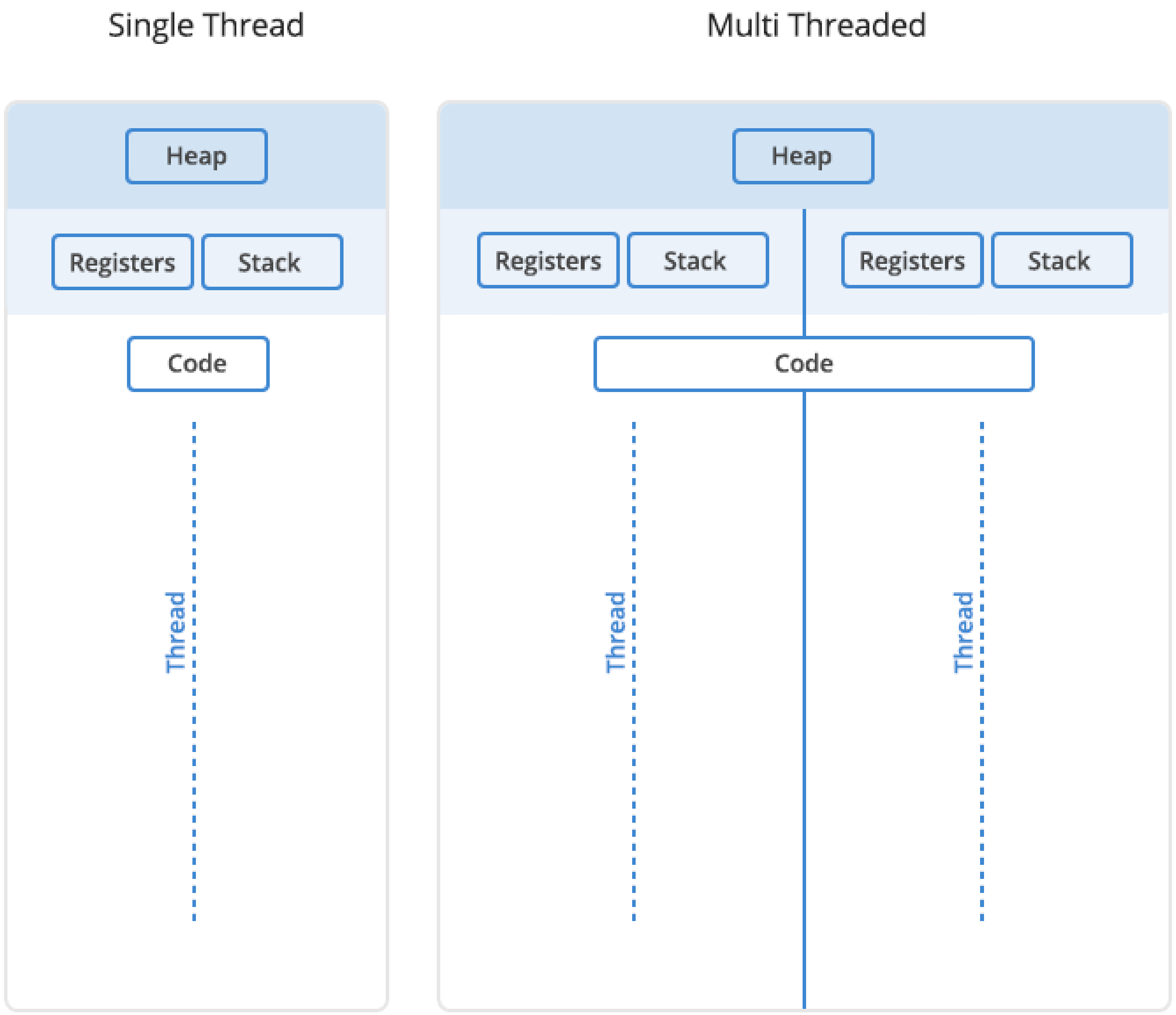
```
677 ms ± 28.9 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

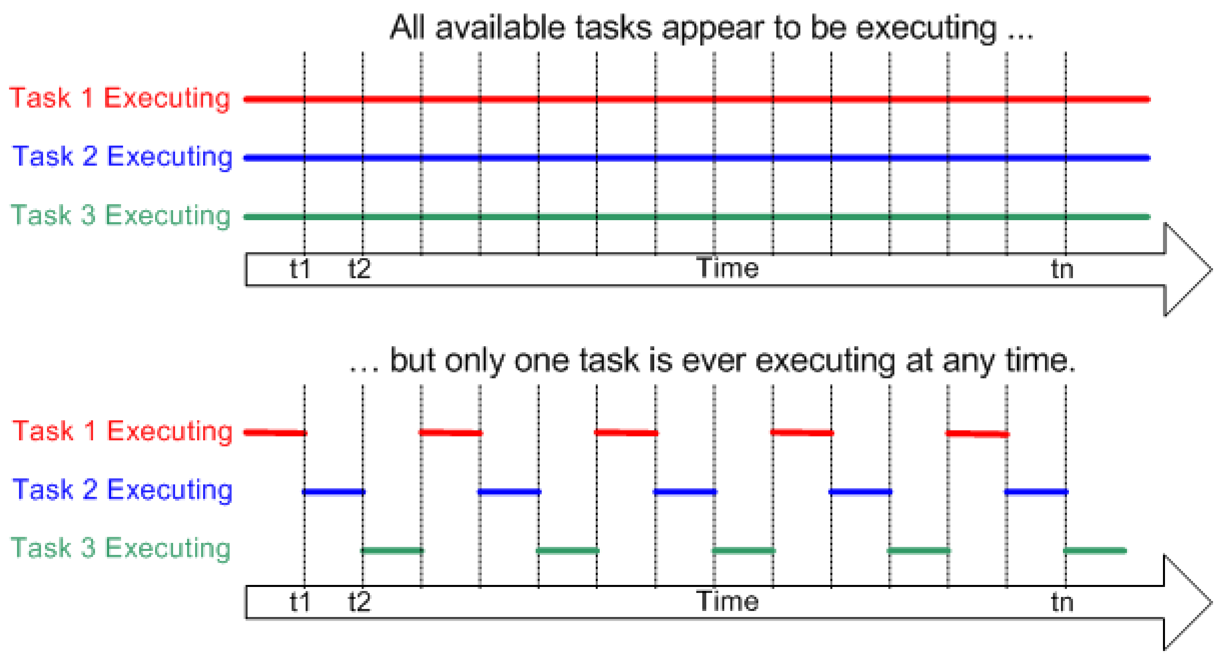
Таки слишком дорого

Резюме

- Создавать процессы — это дорого
- Передавать данные между процессами — тоже дорого. Поэтому иногда меньше процессов — лучше
- Если данных для обмена много, и задача не слишком тяжелая, лучше обойтись без multiprocessing'a

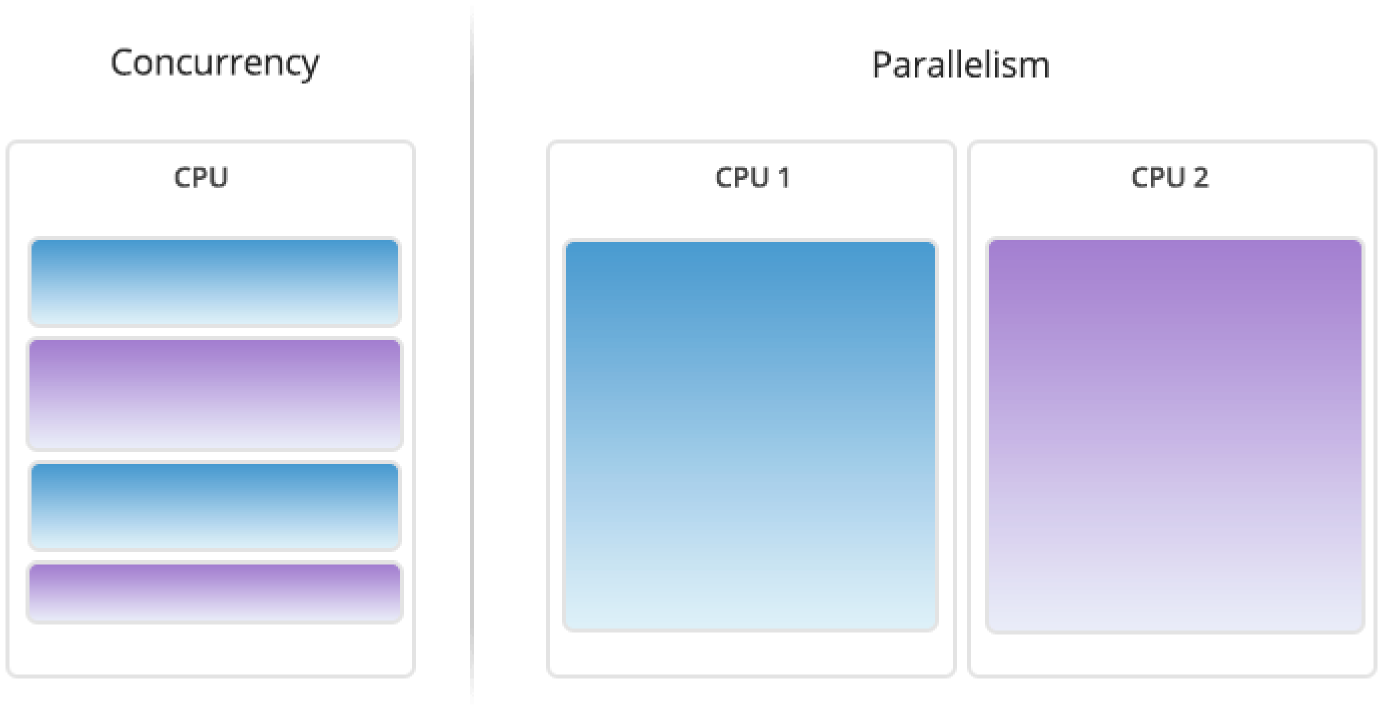
Потоки


Single Thread
Multi Threaded
Heap
Registers
Stack
Code
Thread
Heap
Registers
Stack
Registers
Stack
Code
Thread
Thread



В один момент времени одно ядро процессора исполняет ровно один поток

Несколько ядер могут выполнять несколько потоков буквально одновременно



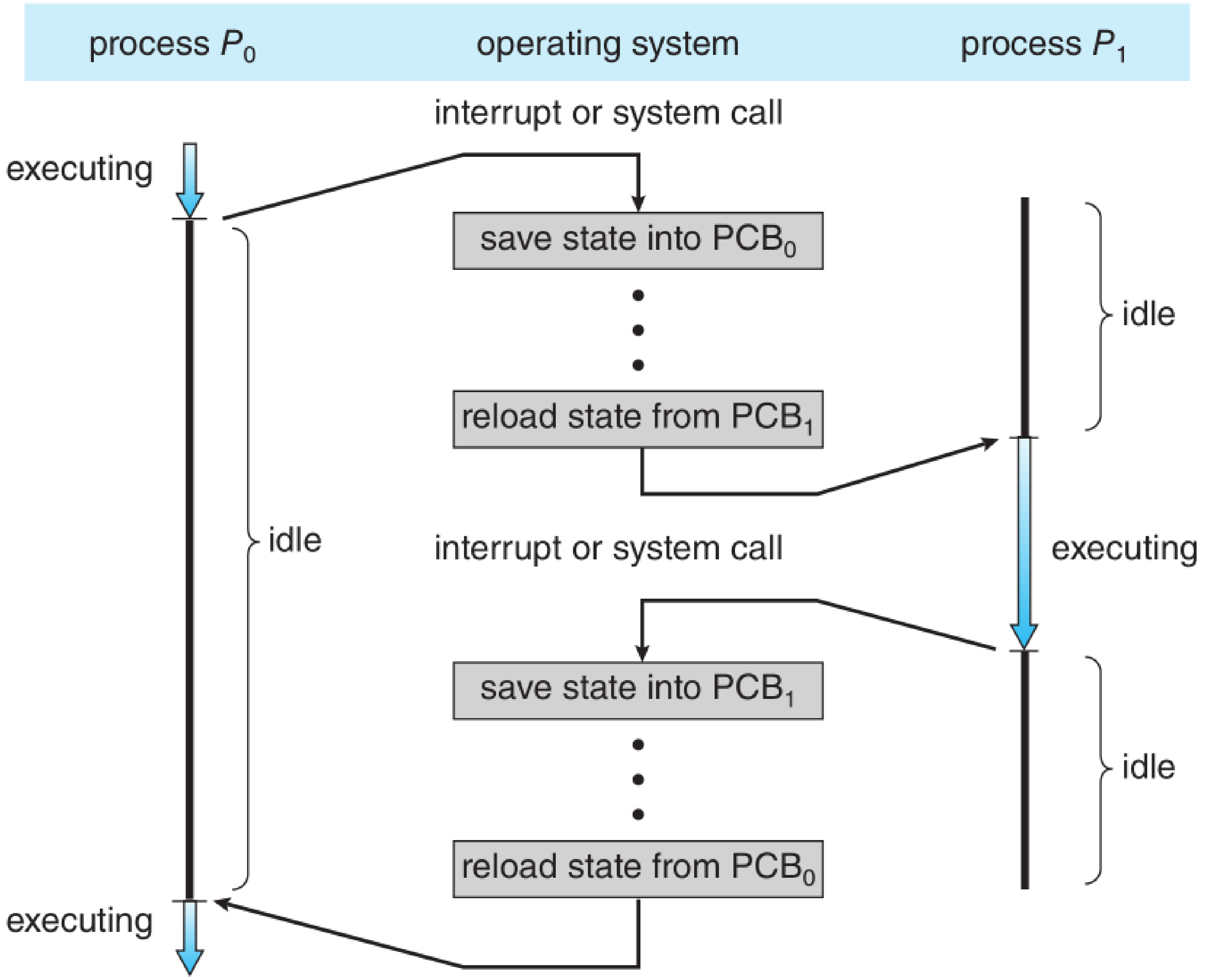
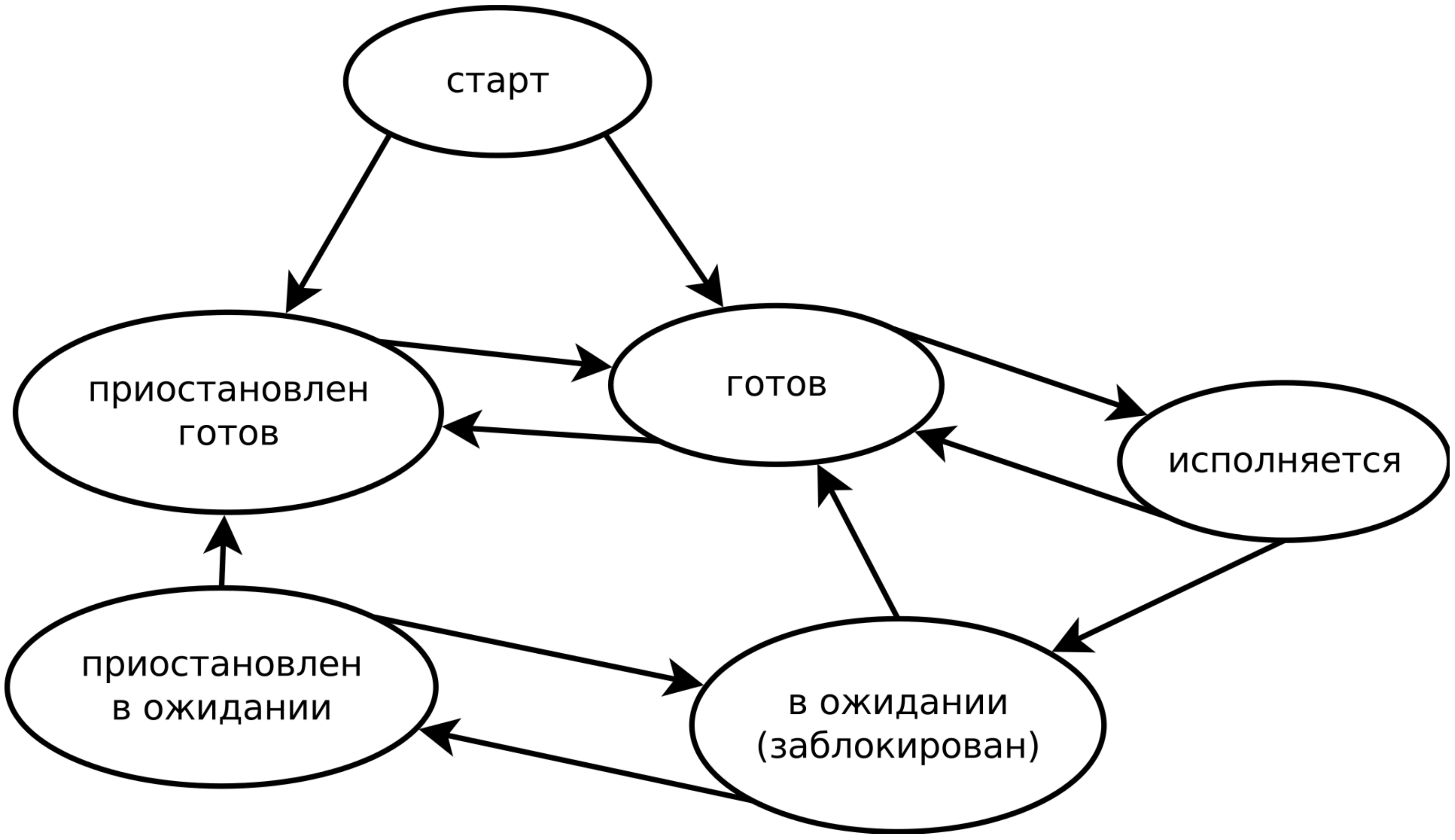


Diagram showing context switch from process to process.

https://github.com/mit-pdos/xv6-public/blob/master/proc.c#L323 (https://github.com/mit-pdos/xv6-public/blob/master/proc.c#L323)

```c
for(;;){

    // Loop over process table looking for process to run.
    acquire(&ptable.lock);
    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
      if(p->state != RUNNABLE)
        continue;

      c->proc = p;
      switchuvm(p);
      p->state = RUNNING;

      swtch(&(c->scheduler), p->context);
      switchkvm();

      c->proc = 0;
    }
    release(&ptable.lock);
  }
```

https://github.com/mit-pdos/xv6-public/blob/master/proc.h#L35
(https://github.com/mit-pdos/xv6-public/blob/master/proc.h#L35)



https://docs.python.org/3/library/threading.html
(https://docs.python.org/3/library/threading.html)

In [41]:
```python
from threading import Thread


def worker(num):
    print(f'Worker: {num}')


threads = [Thread(target=worker, args=(i,)) for i in range(5)]
for t in threads:
    t.start()
for t in threads:
    t.join()
```

```
Worker: 0
Worker: 1Worker: 2

Worker: 3
Worker: 4
```

In [ ]:

```python
from threading import Thread

x = 0

def worker(num):
    global x
    x += 1


threads = [Thread(target=worker, args=(i,)) for i in range(10)]
for t in threads:
    t.start()
for t in threads:
    t.join()

x
```

```
In [14]: from threading import Thread

         x = 0

         def worker(num):
             global x
             x += 1


         threads = [Thread(target=worker, args=(i,)) for i in range(10)]
         for t in threads:
             t.start()
         for t in threads:
             t.join()

         x

Out[14]: 10
```

Нам просто повезло

Давайте усугубим ситуацию

In [42]:
```python
import time
from threading import Thread

x = 0

def worker(num: int) -> None:
    global x
    old_x = x
    time.sleep(0.00001)
    new_x = old_x + 1
    x = new_x


threads = [Thread(target=worker, args=(i,)) for i in range(1000)]
for t in threads:
    t.start()
for t in threads:
    t.join()

x
```

Out[42]:
```
916
```

Race condition

Решение первое: в лоб

```
In [16]: import time
         from threading import Thread, Lock

         x = 0

         def worker(num: int, lock: Lock) -> None:
             global x
             lock.acquire()
             old_x = x
             time.sleep(0.00001)
             new_x = old_x + 1
             x = new_x
             lock.release()


         lock = Lock()
         threads = [Thread(target=worker, args=(i, lock)) for i in range(1000)]
         for t in threads:
             t.start()
         for t in threads:
             t.join()

         x
Out[16]: 1000
```

Чуть более правильное решение

In [17]:
```python
import time
from threading import Thread, Lock

x = 0

def worker(num: int, lock: Lock) -> None:
    global x
    with lock:
        old_x = x
        time.sleep(0.00001)
        new_x = old_x + 1
        x = new_x


lock = Lock()
threads = [Thread(target=worker, args=(i, lock)) for i in range(1000)]
for t in threads:
    t.start()
for t in threads:
    t.join()

x
```

Out[17]:
```
1000
```

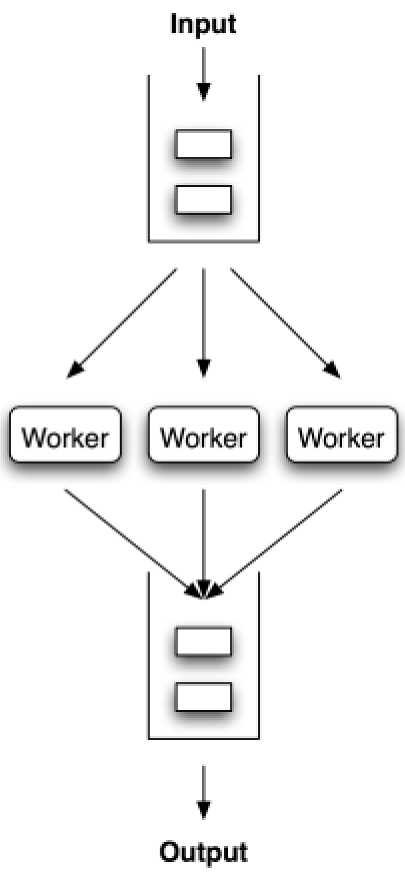
С локами приходится думать

```
In [18]:
from threading import Lock

lock_a = Lock()
lock_b = Lock()
data = {}

def func_a():
    with lock_a:
        with lock_b:
            pass


def func_b():
    with lock_b:
        with lock_a:
            pass
```

In [81]:
```python
from threading import Lock, Thread

def func(lock: Lock) -> None:
    acquired = lock.acquire(timeout=0.5)
    if not acquired:
        print('Without lock')
        return
    try:
        time.sleep(1)
        print('Hello')
    finally:
        lock.release()

lock = Lock()
threads = [Thread(target=func, args=(lock,)) for i in range(4)]
for t in threads:
    t.start()
for t in threads:
    t.join()
```

```
Without lockWithout lock

Without lock
Hello
```

Вариант с queue.Queue (https://docs.python.org/3/library/queue.html)



А что если вызывать форк в многопоточной программе?

In [92]:
```python
import os
import time
from threading import Lock, Thread
from multiprocessing.sharedctypes import RawArray

def worker(lock: Lock, array, idx, value):
    with lock:
        time.sleep(0.001)
        array[idx] += value

def bad_worker(lock: Lock, array, idx, value):
    time.sleep(0.0001)
    os.fork()
    worker(lock, array, idx, value)
```

```
In [94]:
lock = Lock()

def main():
    array = RawArray('i', [0] * 10)
    threads = [
        Thread(target=worker, args=(lock, array, 0, 1))
        for i in range(1000)
    ]
    bad_workers = [
        Thread(target=bad_worker, args=(lock, array, 1, 1))
        for i in range(10)
    ]
    threads.extend(bad_workers)
    for t in threads:
        t.start()
    for t in threads:
        t.join()

    time.sleep(1)
    print(list(array))

main()
```

```
[1000, 10, 0, 0, 0, 0, 0, 0, 0, 0]
```

In [3]:

```
def before_fork():
    lock.acquire()

def after_fork():
    lock.release()

os.register_at_fork(before=before_fork)
os.register_at_fork(after_in_parent=after_fork)
os.register_at_fork(after_in_child=after_fork)

main()
```

```
[1000, 20, 0, 0, 0, 0, 0, 0, 0, 0]
```

In [85]:
```
import os
import time
from threading import Thread
from multiprocessing import Lock
from multiprocessing.sharedctypes import RawArray

lock = Lock()

def main():
    array = RawArray('i', [0] * 10)
    threads = [
        Thread(target=worker, args=(lock, array, 0, 1))
        for i in range(1000)
    ]
    bad_workers = [
        Thread(target=bad_worker, args=(lock, array, 1, 1))
        for i in range(10)
    ]
    for t in threads:
        t.start()
    for t in threads:
        t.join()

    time.sleep(1)
    print(list(array))

main()
```

```
[1000, 20, 0, 0, 0, 0, 0, 0, 0, 0]
```

Задача с суммой массива

In [100]:
```python
def adder(arr, part_id, thread_count, results_queue):
    results_queue.put(sum(arr[part_id::thread_count]))


def sum_using_threads(arr, thread_count):
    res_queue = queue.Queue()
    threads = [
        Thread(target=adder, args=(arr, i, thread_count, res_queue))
        for i in range(thread_count)
    ]
    for thread in threads:
        thread.start()

    results = []
    for thread in threads:
        results.append(res_queue.get())
        thread.join()

    return sum(results)
```

```
In [101]: size = 10 ** 7
          arr = [1 for _ in range(size)]
```

```
In [102]: %%timeit
          sum(arr[:])
```

```
144 ms ± 6.69 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
In [103]: %%timeit
          sum_using_threads(arr, 4)
```

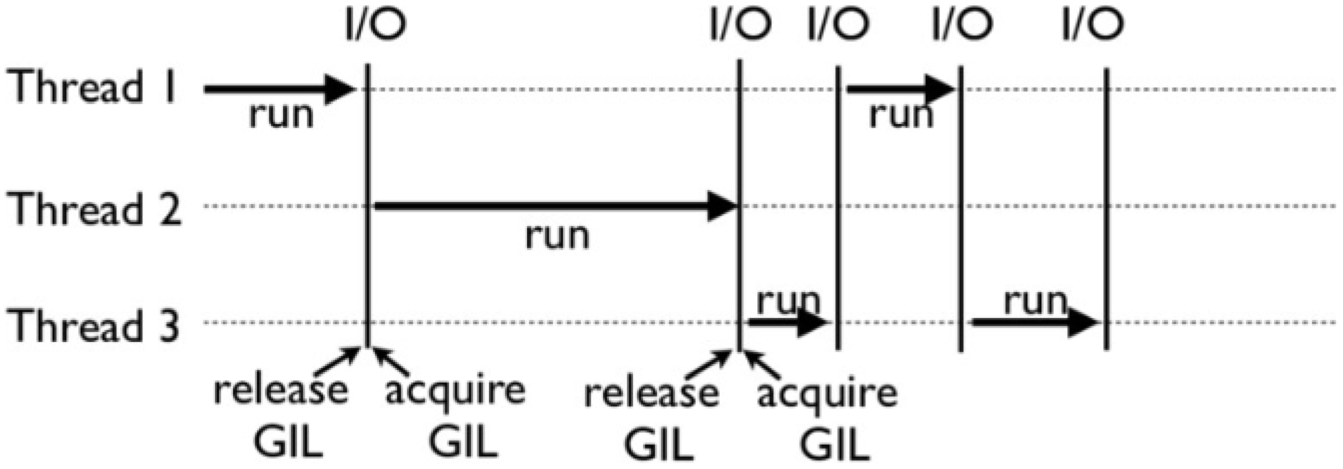
```
215 ms ± 45.9 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

- ХОСПАДИ, а разговоров то было...

GIL* - Global Interpreter Lock

https://asvetlov.blogspot.com/2011/07/gil.html
(https://asvetlov.blogspot.com/2011/07/gil.html)

****Запрещенная в России преступная организация***


I/O
I/O
I/O
I/O
I/O
Thread 1
run
run
Thread 2
run
Thread 3
run
run
release GIL
acquire GIL
release GIL
acquire GIL

Для тех, кто любит почитать исходники на ночь

```c
PyObject *
PyEval_EvalFrameEx(PyFrameObject *f, int throwflag)
{
    PyThreadState *tstate = PyThreadState_GET();
    /* ... */
    for (;;) {
        /* ... */
        if (_Py_atomic_load_relaxed(&eval_breaker)) {
            /* ... */
            if (_Py_atomic_load_relaxed(&gil_drop_request)) {
                /* Give another thread a chance */
                if (PyThreadState_Swap(NULL) != tstate)
                    Py_FatalError("ceval: tstate mix-up");
                drop_gil(tstate);

                /* Other threads may run now */

                take_gil(tstate);
                if (PyThreadState_Swap(tstate) != NULL)
                    Py_FatalError("ceval: orphan tstate");
            }
        }
    /* instruction processing */
    }
}
```

Как же выглядит сам GIL?

```c
struct _gil_runtime_state {
    unsigned long interval;
    _Py_atomic_address last_holder;
    _Py_atomic_int locked;
    unsigned long switch_number;
    PyCOND_T cond;
    PyMUTEX_T mutex;
};
```

Зачем нужен GIL? Почему же его не убрали?

Отпускаем GIL:

- Если есть те, кто его ждет
- Отдаем добровольно перед системным вызовом

Забираем GIL:

- Если мы его отдали по просьбе, то не просим сразу
- Если не получилось захватить GIL, то ждем 5 милисекунд и отправляем запрос на переключение

In [105]:
```
import requests

urls = [
    'https://www.yandex.ru', 'https://www.google.com',
    'https://www.python.org', 'https://github.com'
]
```

In [106]:
```
%%timeit
for url in urls:
    requests.get(url).text
```

```
1.49 s ± 70.9 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [107]:
```
def read_url(url):
    return requests.get(url).text
```

In [108]:
```
%%timeit
readers = [
    Thread(target=read_url, args=(url,)) for url in urls
]
for reader in readers:
    reader.start()
for reader in readers:
    reader.join()
```

```
464 ms ± 17.7 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [112]:

```
from concurrent.futures import as_completed, ThreadPoolExecutor


def fetch_all(executor):
    future_to_url = {executor.submit(read_url, url): url for url in urls}
    for future in as_completed(future_to_url):
        url = future_to_url[future]
        try:
            data = future.result()
        except Exception as exc:
            pass
        else:
            pass


with ThreadPoolExecutor(max_workers=5) as executor:
    %timeit fetch_all(executor)
```

```
457 ms ± 48.9 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

Вывод

- Для IO bound (web, crawlers) приложений потоки отлично работают
- Для CPU bound (math, image processing) используем процессы или специальные C extension (numpy), которые умеют параллелиться без GIL
- В вебе почти всегда комбинация обоих вариантов, т.е. N процессов и в каждом M тредов

Питонячая магия

In [140]:
```
class A:
    def __init__(self, x):
        self.x = x

    def f(self):
        print('f')

    @staticmethod
    def g(self):
        print('g')

a = A(1)
print(a.__dict__)
```

```
{'x': 1}
```

In [141]:
```
A.__dict__
```

Out[141]:
```
mappingproxy({'__module__': '__main__',
              '__init__': <function __main__.A.__init__(self, x)>,
              'f': <function __main__.A.f(self)>,
              'g': <staticmethod at 0x7ffb50251b90>,
              '__dict__': <attribute '__dict__' of 'A' objects>,
              '__weakref__': <attribute '__weakref__' of 'A' objects>,
              '__doc__': None})
```

In [43]:
```
class Cow:
    def __init__(self, name: str) -> None:
        self._name = name

    # Проверяем имя коровы
    def set_name(self, name: str) -> None:
        if not isinstance(name, str):
            raise ValueError()
        if name == "":
            raise ValueError()
        self._name = name
```

In [44]:
```
class Sheep:
    def __init__(self, name: str):
        self._name = name

    def set_name(self, name: str) -> None:
        if not isinstance(name, str):
            raise ValueError()
        if name == "":
            raise ValueError()
        self._name = name
```

Решения?

Наследование

```
In [45]: class Animal:
             def __init__(self, name: str) -> None:
                 self._name = name

             def set_name(self, name: str) -> None:
                 if not isinstance(name, str):
                     raise ValueError()
                 if name == "":
                     raise ValueError()
                 self._name = name

         class Cow(Animal):
             pass

         class Sheep(Animal):
             pass
```

Проблема: появился фермер

In [46]:
```
class Farmer:
    def __init__(self, name: str, surname: str) -> None:
        self._name = name
        self._surname = surname

    def set_name(self, name: str) -> None:
        if not isinstance(name, str):
            raise ValueError()
        if name == "":
            raise ValueError()
        self._name = name

    def set_surname(self, surname: str) -> None:
        if not isinstance(surname, str):
            raise ValueError()
        if surname == "":
            raise ValueError()
        self._surname = surname
```

In [47]:
```
class Farmer(Animal):
    def __init__(self, name: str, surname: str) -> None:
        super().__init__(name)
        self._surname = surname

    def set_surname(self, surname: str) -> None:
        if not isinstance(surname, str):
            raise ValueError()
        if surname == "":
            raise ValueError()
        self._surname = surname
```

Дескрипторы

https://docs.python.org/3/howto/descriptor.html (https://docs.python.org/3/howto/descriptor.html)

In [ ]:
```
a.x  # a - объект
# type(a).__dict__['x'].__get__(a, type(a))
```

In [114]:

```
class NonEmptyString:
    def __init__(self, attr_name: str):
        self._attr_name = attr_name

    def __get__(self, obj: Optional[Any], objtype: Optional[type] = None) -> Any:
        if obj is None:
            return self
        return getattr(obj, self._attr_name)

    def __set__(self, obj: Any, value: str) -> None:
        if not isinstance(value, str):
            raise ValueError()
        if value == "":
            raise ValueError()
        setattr(obj, self._attr_name, value)

    def __delete__(self, obj: Any) -> None:
        raise ValueError()
```

In [115]:
```
class Farmer:
    name = NonEmptyString("_name")
    surname = NonEmptyString("_surname")

    def __init__(self, name: str, surname: str) -> None:
        self.name = name
        self.surname = surname


farmer = Farmer("Grzegorz", "Brzęczyszczykiewicz")
print(farmer.name, farmer.surname)
farmer.name = "Boris"
print(farmer.name, farmer.surname)
try:
    farmer.name = ''
except ValueError:
    print('Error!')
```

```
Grzegorz Brzęczyszczykiewicz
Boris Brzęczyszczykiewicz
Error!
```

```
In [116]: try:
              del farmer.name
          except ValueError:
              print('Error!')
```

```
Error!
```

```
In [17]: print('name' in farmer.__dict__)
         print('name' in Farmer.__dict__)
         attr = Farmer.__dict__['name']
         print(attr)
         print(attr.__get__(farmer, type(farmer)))
```

```
False
True
<__main__.NonEmptyString object at 0x7ffb680a3650>
Boris
```

Примеры:

- property

Non-data дескрипторы

Не определены `__set__` и `__delete__`

```
In [ ]: C.x
        # C.__dict__['x'].__get__(None, C)
```

```
In [25]: from typing import Callable, Any

         class StaticMethod:
             def __init__(self, func: Callable[..., Any]):
                 self._func = func

             def __get__(self, instance: Optional[Any], owner: type) -> Callable[..., Any]:
                 return self._func

         def mystaticmethod(func: Callable[..., Any]) -> StaticMethod:
             return StaticMethod(func)

         class A:
             @mystaticmethod
             def f():
                 print('1')


         A.f()
         a = A()
         a.f()
```

```
1
1
```

```
In [28]: print('f' in A.__dict__)
         print(type(A.__dict__['f']))

         attr = A.__dict__['f']

         print(type(attr.__get__(None, A)))
```

```
True
<class '__main__.StaticMethod'>
<class 'function'>
```

Примеры non-data дескрипторов:

- staticmethod
- classmethod

Метаклассы

In [ ]:
```python
class A:
    def __init__(self):
        print('A.__init__')
        self.x = 1


a = A()
print(type(a))
print(a.x)
```

In [58]:

```
class A:
    def __init__(self):
        print('A.__init__')
        self.x = 1


a = A()
print(type(a))
print(a.x)
```

```
A.__init__
<class '__main__.A'>
1
```

Всё есть объект

```
In [70]: print(type(1))
         print(type('a'))
         print(type(type))
         print(type(str))
```

```
<class 'int'>
<class 'str'>
<class 'type'>
<class 'type'>
```

In [65]:
```python
class B(A):
    def method(self):
        print(1)

b = B()
b.method()
```

```
A.__init__
1
```

In [66]:
```python
B = type('B', (A,), {'method': lambda self: print(1)})

b = B()
b.method()
```

```
A.__init__
1
```

In [ ]:
```
class B:
    def __new__(cls, *args):
        print("B.__new__")
        print(args)
        return object.__new__(cls)

    def __init__(self, *args):
        print("B.__init__")
        print(args)
        self.x = "arg"

b = B(2)
```

In [118]:
```
class B:
    def __new__(cls, *args):
        print("B.__new__")
        print(args)
        return object.__new__(cls)

    def __init__(self, *args):
        print("B.__init__")
        print(args)
        self.x = "arg"

b = B(2)
```

```
B.__new__
(2,)
B.__init__
(2,)
```

In [ ]:
```
class B:
    def __new__(cls, *args):
        print("B.__new__")
        print(args)
        type_ = type("C", tuple(), {})
        obj = type_()
        cls.__init__(obj)
        return obj

    def __init__(self, *args):
        print("B.__init__")
        print(args)
        self.x = "arg"


b = B(2)
print(type(b))
print(b.x)
```

In [1]:
```
class B:
    def __new__(cls, *args):
        print("B.__new__")
        print(args)
        type_ = type("C", tuple(), {})
        obj = type_()
        cls.__init__(obj)
        return obj

    def __init__(self, *args):
        print("B.__init__")
        print(args)
        self.x = "arg"
```

In [2]:
```
b = B(2)
```

```
B.__new__
(2,)
B.__init__
()
```

In [3]:
```
print(type(b))
```

```
<class '__main__.C'>
```

In [4]:
```
print(b.x)
```

```
arg
```

In [ ]:
```
class C:
    def __new__(cls, *args) -> int:
        print("C.__new__")
        print(args)
        return 1

    def __init__(self, x: int) -> None:
        print("C.__init__")
        print(args)
        self.x = x

b = C(3)
print(type(b))
print(b.x)
```

In [119]:
```
class C:
    def __new__(cls, *args) -> int:
        print("C.__new__")
        print(args)
        return 1

    def __init__(self, x: int) -> None:
        print("C.__init__")
        print(args)
        self.x = x
```

In [120]:
```
b = C(3)
```

```
C.__new__
(3,)
```

In [121]:
```
print(type(b))
```

```
<class 'int'>
```

In [122]:
```
print(b.x)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-122-a5a627eb78d5> in <module>
----> 1 print(b.x)

AttributeError: 'int' object has no attribute 'x'
```

In [34]:
```python
class Metaclass(type):
    def __new__(
        cls: type,
        name: str,
        bases: Tuple[type, ...],
        dct: Dict[str, Any]
    ) -> "Metaclass":
        print("Metaclass.__new__")
        print(cls, name, bases, dct)
        obj = type.__new__(cls, name, bases, dct)
        return obj
```

In [35]:
```python
class Example(metaclass=Metaclass):
    pass
```

```
Metaclass.__new__
<class '__main__.Metaclass'> Example () {'__module__': '__main__', '__qualname__': 'Example'}
```

In [87]:
```python
class Metaclass(type):
    def __new__(cls, name, bases, dct: Dict[str, Any]) -> "Metaclass":
        print("Metaclass.__new__")
        print(cls, name, bases, dct)
        obj = type.__new__(cls, name, bases, dct)
        return obj

    def __init__(
        cls:   type,
        name:  str,
        bases: Tuple[type],
        dct:   Dict[str, Any]
    ) -> None:
        print("Metaclass.__init__")
        print(cls, name, bases, dct)


class Exmaple(metaclass=Metaclass):
    def f():
        pass
```

```
Metaclass.__new__
<class '__main__.Metaclass'> Exmaple () {'__module__': '__main__', '__qualname__': 'Exmaple', 'f': <functi
on Exmaple.f at 0x7f4984313710>}
Metaclass.__init__
<class '__main__.Exmaple'> Exmaple () {'__module__': '__main__', '__qualname__': 'Exmaple', 'f': <function
Exmaple.f at 0x7f4984313710>}
```

In [88]:
```
class Metaclass(type):
    def __new__(meta, name, bases, dct) -> "Metaclass":
        print("Metaclass.__new__")
        return super().__new__(meta, name, bases, dct)

    def __init__(cls, name, bases, dct) -> None:
        print("Metaclass.__init__")

    def __call__(cls, *args, **kwargs):
        print("Metaclass.__call__")
        print(cls, args, kwargs)
        return type.__call__(cls, *args, **kwargs)
        # return cls(*args, **kwargs)  бесконечная рекурсия

class Exmaple(metaclass=Metaclass):
    def __init__(self, *args, **kwargs):
        print("Exmaple.__init__")
        print(args, kwargs)
```

```
Metaclass.__new__
Metaclass.__init__
```

In [89]:
```
obj = Exmaple(1, x=2)
```

```
Metaclass.__call__
<class '__main__.Exmaple'> (1,) {'x': 2}
Exmaple.__init__
(1,) {'x': 2}
```

Связь класса с метаклассом:

1. `__new__` вызывается до создания класса, возвращает класс
2. `__init__` после создания класса
3. `__call__` вызывается перед созданием объекта класса

ABC + abstractmethod

Что это? Зачем нужен?

In [130]:
```python
class AbstractMethod:
    def __call__(self) -> None:
        raise NotImplementedError("Method not implemented")

def abstractmethod(method: Callable[..., Any]) -> AbstractMethod:
    return AbstractMethod()

class Animal():
    @abstractmethod
    def hello(self) -> None:
        pass
```

In [ ]:

```
animal = Animal()
animal.hello()
```

```
In [131]: animal = Animal()
          animal.hello()
```

```
---------------------------------------------------------------------------
NotImplementedError                       Traceback (most recent call last)
<ipython-input-131-edd9b096089d> in <module>
      1 animal = Animal()
----> 2 animal.hello()

<ipython-input-130-12aa2b6e3e8c> in __call__(self)
      1 class AbstractMethod:
      2     def __call__(self) -> None:
----> 3         raise NotImplementedError("Method not implemented")
      4 
      5 def abstractmethod(method: Callable[..., Any]) -> AbstractMethod:

NotImplementedError: Method not implemented
```

Настоящий ABC скорее всего сложнее, и нижеследующий наверное не работает в каких-то случаях

In [64]:
```
from copy import deepcopy

import inspect


class MyABCMeta(type):
    def __init__(
        cls: type, name: str, bases: Tuple[type, ...], dct: Dict[str, Any]
    ) -> None:
        # Собираем все AbstractMethod из класса, который создаём
        abstract_methods = {
            name for name, value in dct.items() if isinstance(value, AbstractMethod)
        }
        # Собираем все AbstractMethod из родителей класса, который создаём
        for base in bases:
            new_methods = inspect.getmembers(
                base, predicate=lambda x: isinstance(x, AbstractMethod)
            )
            abstract_methods.update({k for k, v in new_methods})
        # Теперь в abstract_methods собрали все методы, которые нужно переписать
        # Собираем все функции, которые есть в классе, который создаём
        concrete_methods = {
            name for name, value in dct.items() if inspect.isfunction(value)
        }
        # Записываем все непереопределённые методы в __abstract_methods__
        cls._abstract_methods = abstract_methods - concrete_methods

    def __call__(cls: type, *args: Any, **kwargs: Any) -> Any:
        # Если на момент создания объекта в классе остаются абстрактные методы кидаем ошибку
        if cls._abstract_methods:
            methods = ", ".join(cls._abstract_methods)
            raise NotImplementedError("Methods not implemented: {}".format(methods))
        return type.__call__(cls, *args, **kwargs)

class MyABC(metaclass=MyABCMeta):
    pass
```

In [65]:

```
 class Animal(MyABC):
    @abstractmethod
    def hello(self) -> None:
        pass


class Cow(Animal):
    def hello(self) -> None:
        print("Moo")

class Sheep(Animal):
    pass
```

```
In [66]: try:
             l = Animal()
         except NotImplementedError as e:
             print(e)
         try:
             s = Sheep()
         except NotImplementedError as e:
             print(e)

         c = Cow()
         c.hello()
```

```
Methods not implemented: hello
Methods not implemented: hello
Moo
```

Как же тут определился метакласс?

Tenacity

Библиотека для ретраев

```
In [41]:
import random
from tenacity import retry

@retry
def do_something_unreliable():
    if random.randint(0, 10) > 1:
        print('Fail')
        raise IOError("Broken sauce, everything is hosed!!!111one")
    else:
        return "Awesome sauce!"

print(do_something_unreliable())
```

```
Fail
Fail
Fail
Fail
Fail
Awesome sauce!
```

Когда останавливаться?

In [43]:
```
from tenacity import stop_after_delay, stop_after_attempt
```

In [53]:
```
@retry(stop=(stop_after_delay(10) | stop_after_attempt(5)))
def stop_after_10_s_or_5_retries():
    print("Stopping after 10 seconds or 5 retries")
    raise Exception

try:
    stop_after_10_s_or_5_retries()
except Exception as exc:
    print(exc)
```

```
Stopping after 10 seconds or 5 retries
Stopping after 10 seconds or 5 retries
Stopping after 10 seconds or 5 retries
Stopping after 10 seconds or 5 retries
Stopping after 10 seconds or 5 retries
RetryError[<Future at 0x7ffb523c4a90 state=finished raised Exception>]
```

Сколько ждать?

```
In [ ]: @retry(wait=wait_exponential(multiplier=1, min=4, max=10))
        def wait_exponential_1():
            raise Exception
```

In [ ]:

```
В каком случае ретраить?
```

In [ ]:

```
@retry(retry=retry_if_exception_type(IOError))
def might_io_error():
    print("Retry forever with no wait if an IOError occurs, raise any other errors")
    raise Exception
```

```
In [ ]: Ретрай блока кода?
```

```
In [ ]: from tenacity import Retrying, RetryError, stop_after_attempt

        try:
            for attempt in Retrying(stop=stop_after_attempt(3)):
                with attempt:
                    raise Exception('My code is failing!')
        except RetryError:
            pass
```

И ещё много всего прочего

https://tenacity.readthedocs.io/en/latest/ (https://tenacity.readthedocs.io/en/latest/)

BeautifulSoup

In [68]:
```
data = """
<html><head><title>The Dormouse's story</title></head>
<body>
<p class="title"><b>The Dormouse's story</b></p>

<p class="story">Once upon a time there were three little sisters; and their names were
<a href="http://example.com/elsie" class="sister" id="link1">Elsie</a>,
<a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and
<a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
and they lived at the bottom of a well.</p>

<p class="story">...</p>
"""
```

In [69]:
```python
from bs4 import BeautifulSoup

soup = BeautifulSoup(data)
```

In [70]:
```python
for link in soup.find_all('a'):
    print(link.get('href'))
```

```
http://example.com/elsie
http://example.com/lacie
http://example.com/tillie
```

```
In [71]: soup.body
```

```
Out[71]: <body>
         <p class="title"><b>The Dormouse's story</b></p>
         <p class="story">Once upon a time there were three little sisters; and their names were
         <a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
         <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a> and
         <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>;
         and they lived at the bottom of a well.</p>
         <p class="story">...</p>
         </body>
```

In [72]:
```python
for string in soup.strings:
    print(repr(string))
```

```
"The Dormouse's story"
'\n'
'\n'
"The Dormouse's story"
'\n'
'Once upon a time there were three little sisters; and their names were\n'
'Elsie'
',\n'
'Lacie'
' and\n'
'Tillie'
';\nand they lived at the bottom of a well.'
'\n'
'...'
'\n'
```

In [73]:
```
tag = soup.body.p
```

In [79]:
```
for parent in tag.parents:
    if parent is None:
        print(parent)
    else:
        print(parent.name)
```

```
body
html
[document]
```

А дальше сами

https://www.crummy.com/software/BeautifulSoup/bs4/doc/
(https://www.crummy.com/software/BeautifulSoup/bs4/doc/)

Вопросы?