
# Углубленный Python

Лекция 5

Латкин Игорь


“

Не забудьте отметиться на занятии!

*Цитата великих*

# Повестка дня


1. Python coroutines
2. asyncio
3. aiohttp,sanic


“

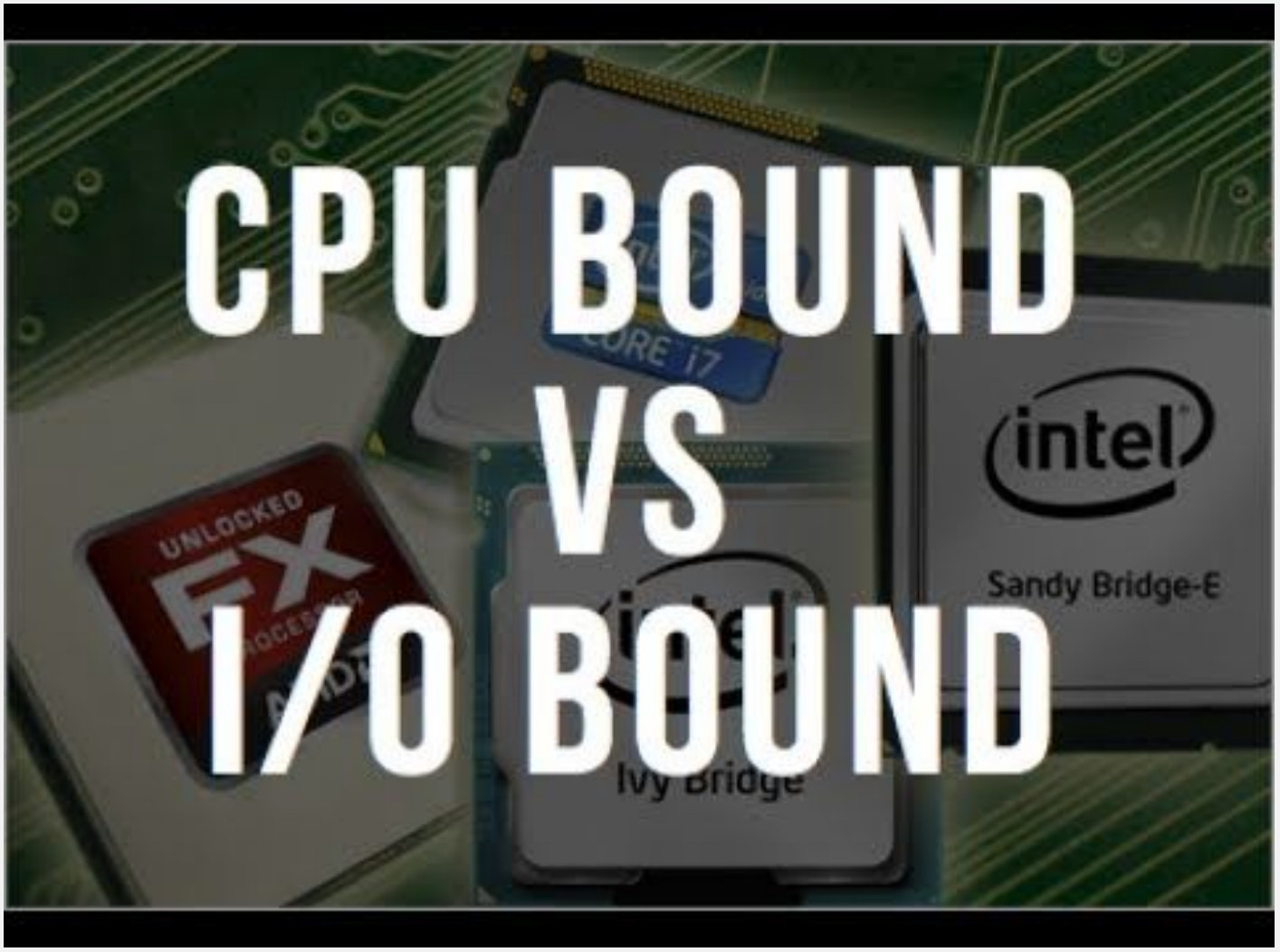
After all, the web is a big place now.


search...
Lorem ipsum

# Web Crawler. Алгоритм работы


1. Первоначальный список URL, которые необходимо скачать
2. Скачивание страниц из списка URL
3. Анализ страницы
4. Поиск связанных URL
5. Повторить с п.2.

# CPU VS I/O


CPU BOUND
VS
I/O BOUND
UNLOCKED
FX
intel
Sandy Bridge-E
Ivy Bridge

# Traditional Approach


```
def fetch(url):
    sock = socket.socket()
    sock.connect(('xkcd.com', 80))
    request = 'GET {} HTTP/1.0\r\nHost: xkcd.com\r\n\r\n'.format(url)
    sock.send(request.encode('ascii'))
    response = b''
    chunk = sock.recv(4096)
    while chunk:
        response += chunk
        chunk = sock.recv(4096)

    # Page is now downloaded.
    links = parse_links(response)
    q.add(links)
```

# Traditional Approach


- connect & recv - блокирующие операции
- C10k problem - http://www.kegel.com/c10k.html
- C100k problem
- Потоки дорогие (CPU & RAM)
- Большую часть времени потоки простаивают

# Async. Non-blocking


```
sock = socket.socket()
sock.setblocking(False)
try:
    sock.connect(('xkcd.com', 80))
except BlockingIOError:
    pass
```

# Async. Non-blocking. Bad way.


```
request = 'GET {} HTTP/1.0\r\nHost: xkcd.com\r\n\r\n'.format(url)
encoded = request.encode('ascii')

while True:
    try:
        sock.send(encoded)
        break  # Done.
    except OSError as e:
        pass

print('sent')
```

# Async. Non-blocking. Right way (1/3)


- man 2 select
- man 2 poll
- man 7 epoll
- kqueue

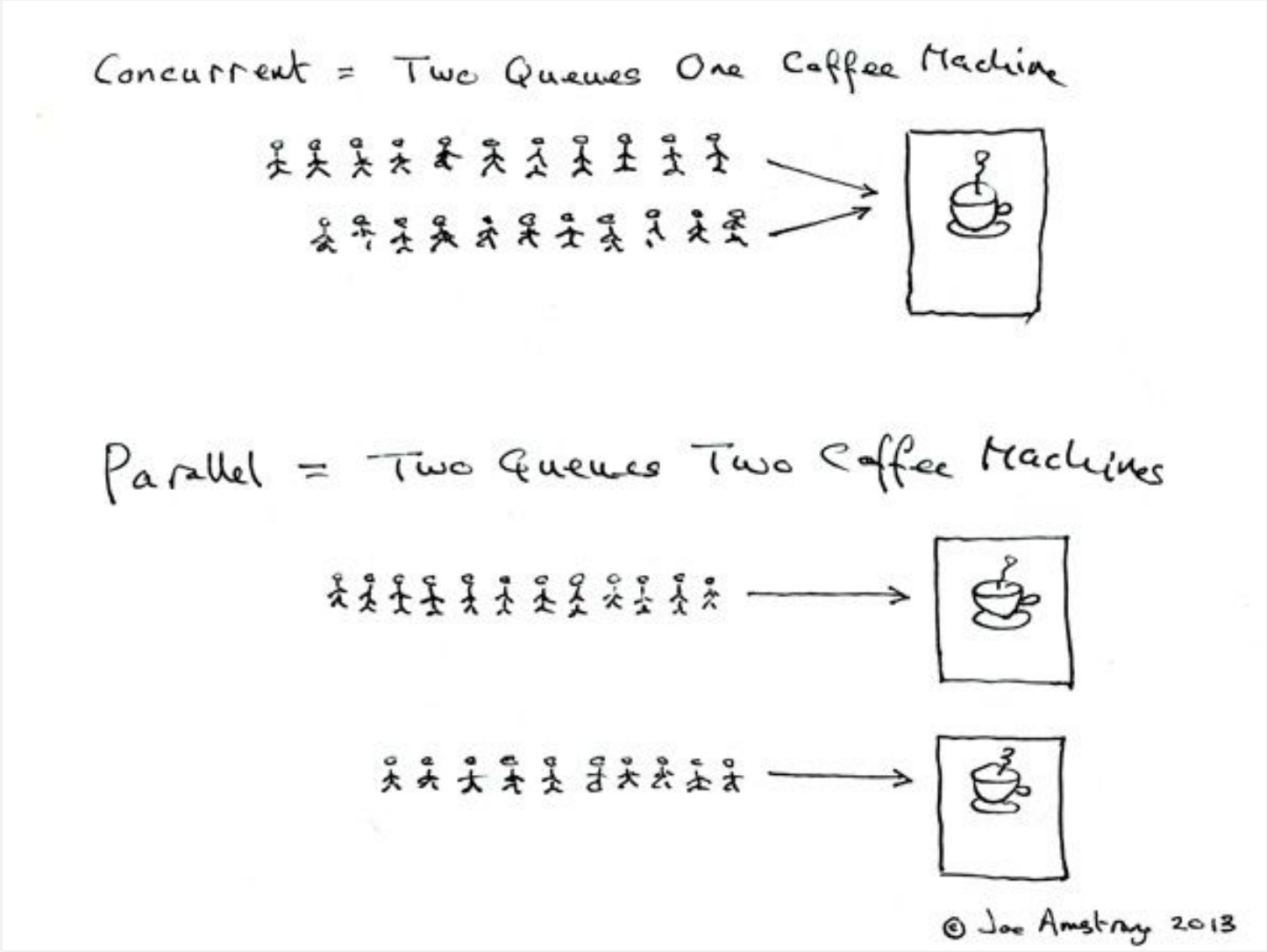
# Async. Non-blocking. Right way (2/3)


```
from selectors import DefaultSelector, EVENT_WRITE

selector = DefaultSelector()

sock = socket.socket()
sock.setblocking(False)
try:
    sock.connect(('xkcd.com', 80))
except BlockingIOError:
    pass

def connected():
    selector.unregister(sock.fileno())
    print('connected!')

selector.register(sock.fileno(), EVENT_WRITE, connected)
```

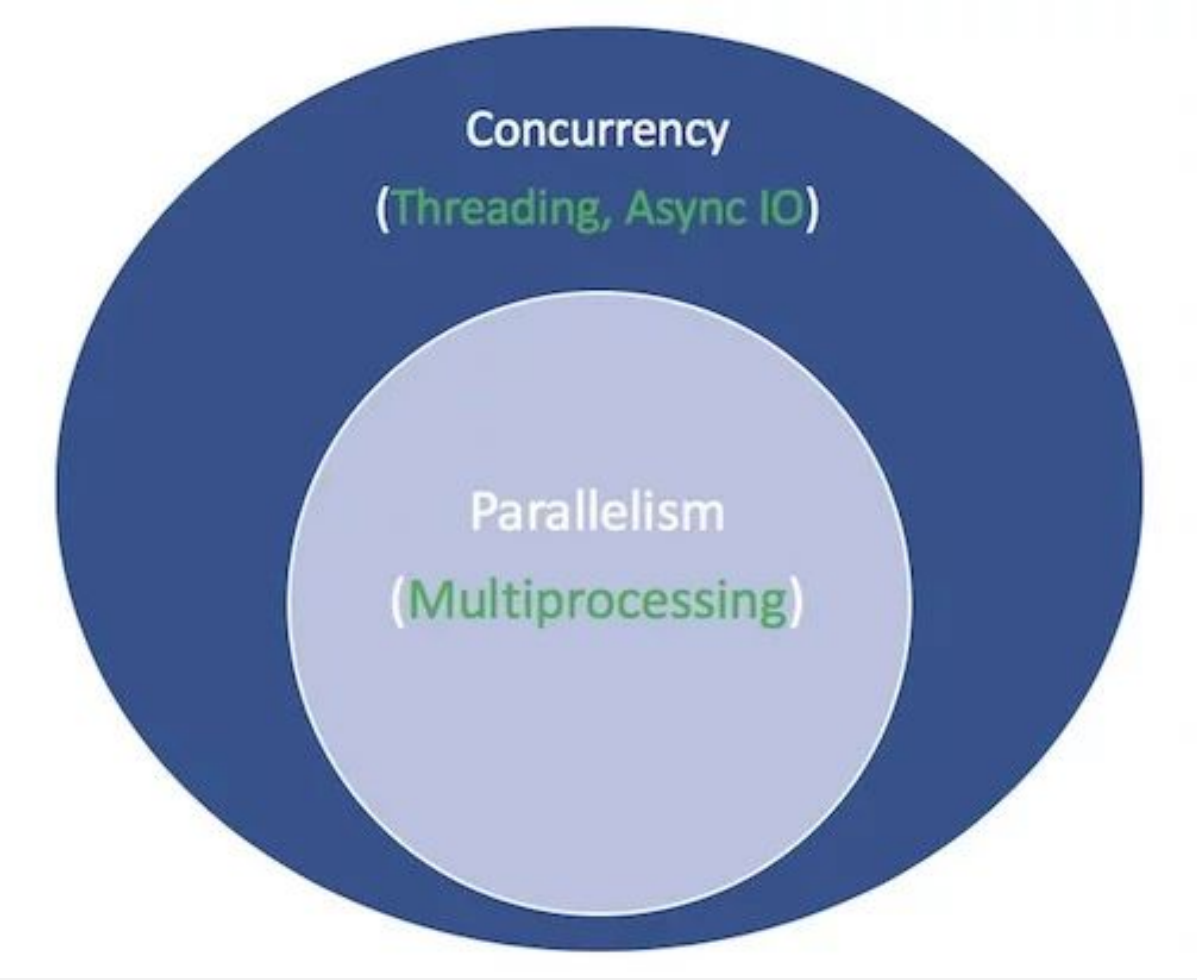
# Async. Non-blocking. Right way (3/3)


```
def loop():
    while True:
        events = selector.select()
        for event_key, event_mask in events:
            callback = event_key.data
            callback()
```

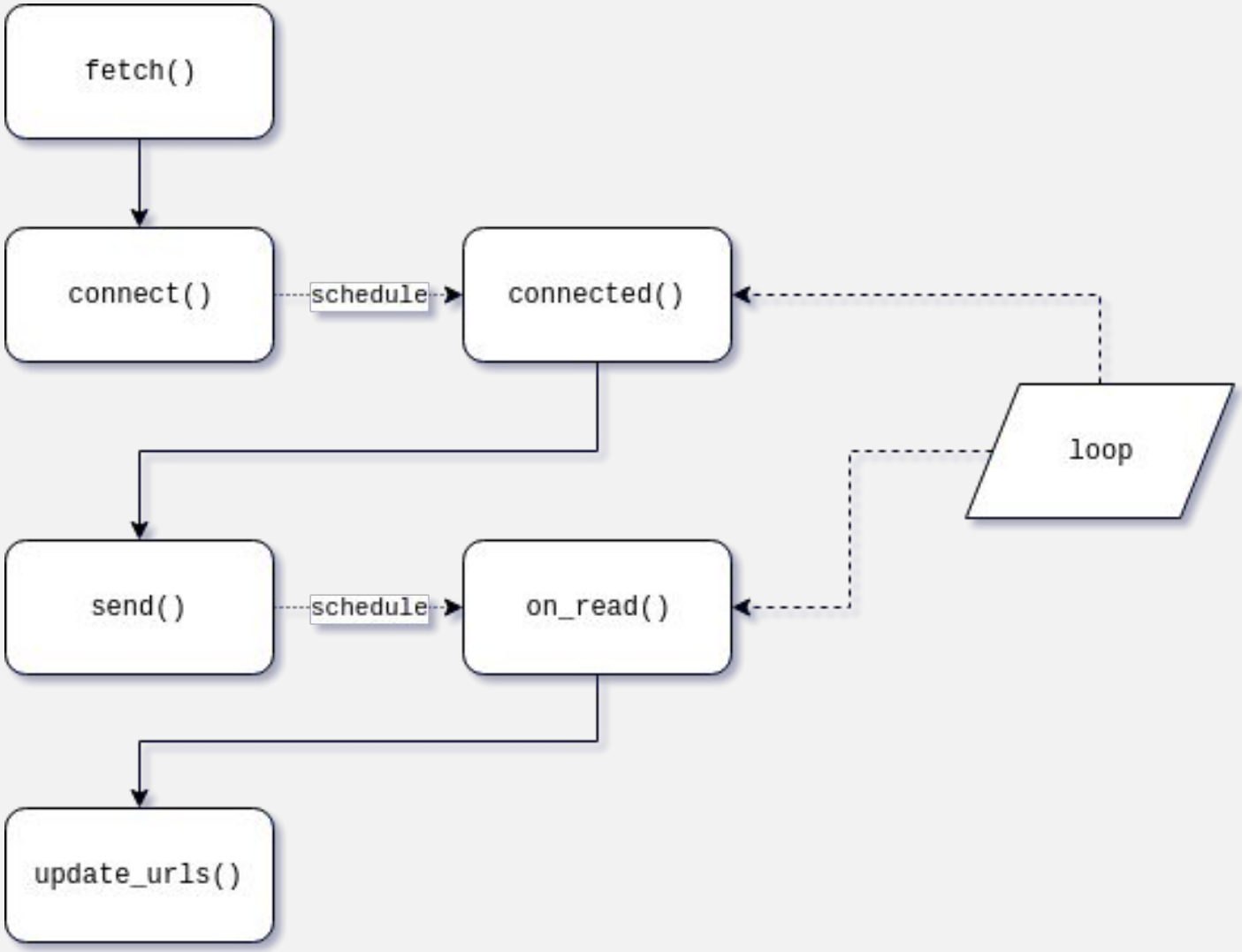
# Concurrency vs Parallelism



# Concurrency vs Parallelism



# Callback hell (1/2)



# Callback hell (2/2)

```
function hell(win) {
 // for listener purpose
 return function() {
   loadLink(win, REMOTE_SRC+'/assets/css/style.css', function() {
     loadLink(win, REMOTE_SRC+'/lib/async.js', function() {
       loadLink(win, REMOTE_SRC+'/lib/easyXDM.js', function() {
         loadLink(win, REMOTE_SRC+'/lib/json2.js', function() {
           loadLink(win, REMOTE_SRC+'/lib/underscode.min.js', function() {
             loadLink(win, REMOTE_SRC+'/lib/backbone.min.js', function() {
               loadLink(win, REMOTE_SRC+'/dev/base_dev.js', function() {
                 loadLink(win, REMOTE_SRC+'/assets/js/deps.js', function() {
                   loadLink(win, REMOTE_SRC+'/src/' + win.loader_path + '/loader.js', function() {
                     async.eachSeries(SCRIPTS, function(src, callback) {
                       loadScript(win, BASE_URL+src, callback);
                     });
                   });
                 });
               });
             });
           });
         });
       });
     });
   });
 };
}
```

# Coroutines

Don’t block
Keep moving

# Coroutines in Python 3.4


```
@asyncio.coroutine
def fetch(self, url):
    response = yield from self.session.get(url)
    body = yield from response.read()
```

# Python Generators (1/6)

```
1   def foo():
2       bar()
3
4   def bar():
5       pass
6
7   import dis
8   dis.dis(foo)
```

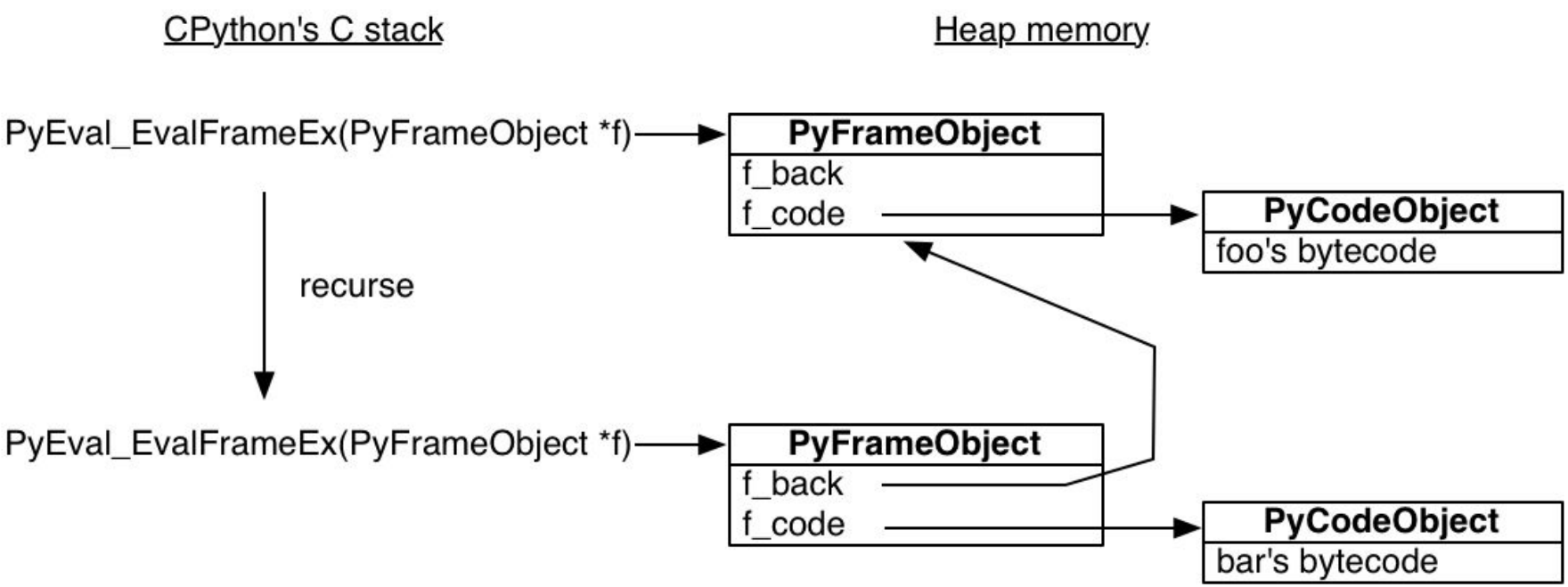
```
➜ python ./3.py
  2           0 LOAD_GLOBAL              0 (bar)
              2 CALL_FUNCTION            0
              4 POP_TOP
              6 LOAD_CONST               0 (None)
              8 RETURN_VALUE
```

# Python Generators (2/6). Frames

Стек Python-функции - тоже PyObject, а значит находится в heap!

```python
import inspect
frame = None

def foo():
    bar()

def bar():
    global frame
    frame = inspect.currentframe()

foo()
# The frame was executing the code for 'bar'.
print(frame.f_code.co_name)

# Its back pointer refers to the frame for 'foo'.
caller_frame = frame.f_back
print(caller_frame.f_code.co_name)
```

# Python Generators (3/6). Frames

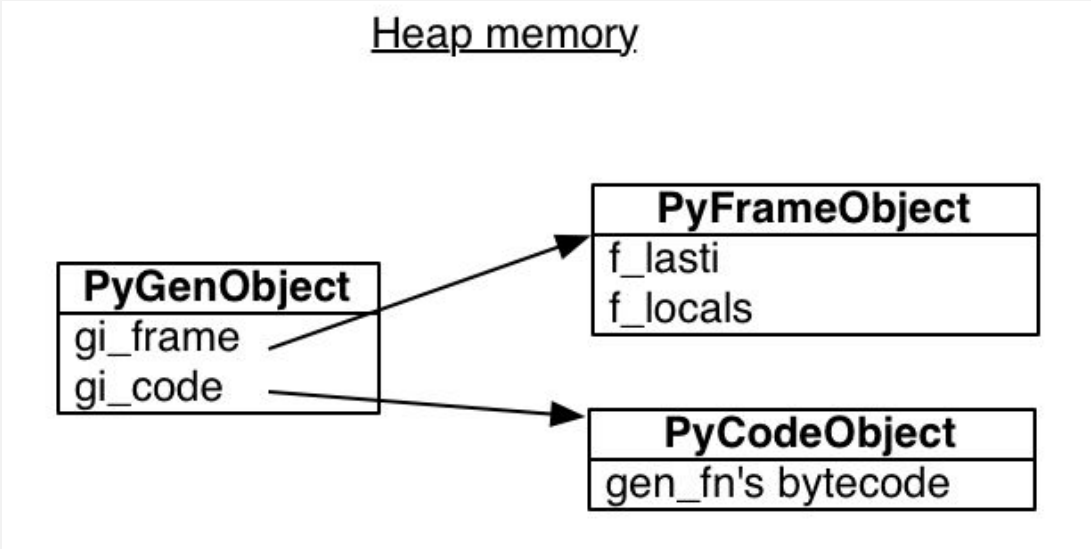


# Python Generators (4/6)

```
1   def gen_fn():
2       result = yield 1
3       print('result of yield: {}'.format(result))
4       result2 = yield 2
5       print('result of 2nd yield: {}'.format(result2))
6       return 'done'
7
8   def non_gen_fn():
9       pass
10
11  generator_bit = 1 << 5
12  print(bool(gen_fn.__code__.co_flags & generator_bit))
13  print(bool(non_gen_fn.__code__.co_flags & generator_bit))
```

# Python Generators (5/6)

```
1  def gen_fn():
2      result = yield 1
3      print('result of yield: {}'.format(result))
4      result2 = yield 2
5      print('result of 2nd yield: {}'.format(result2))
6      return 'done'
7
8  def non_gen_fn():
9      pass
10
11 generator_bit = 1 << 5
12 print(bool(gen_fn.__code__.co_flags & generator_bit))
13 print(bool(non_gen_fn.__code__.co_flags & generator_bit))
```



# Python Generators (6/6)

```
1  def gen_fn():
2      result = yield 1
3      print('result of yield: {}'.format(result))
4      result2 = yield 2
5      print('result of 2nd yield: {}'.format(result2))
6      return 'done'
7
8  def non_gen_fn():
9      pass
10
11 generator_bit = 1 << 5
12 print(bool(gen_fn.__code__.co_flags & generator_bit))
13 print(bool(non_gen_fn.__code__.co_flags & generator_bit))
```

```
16 gen = gen_fn()
17 print('total', len(gen.gi_code.co_code))
18 gen.gi_frame.f_lasti  # (-1) - последняя инструкция
19
20 print('send', gen.send(None))
21
22 gen.gi_frame.f_lasti  # (2)
```

# Future & Task

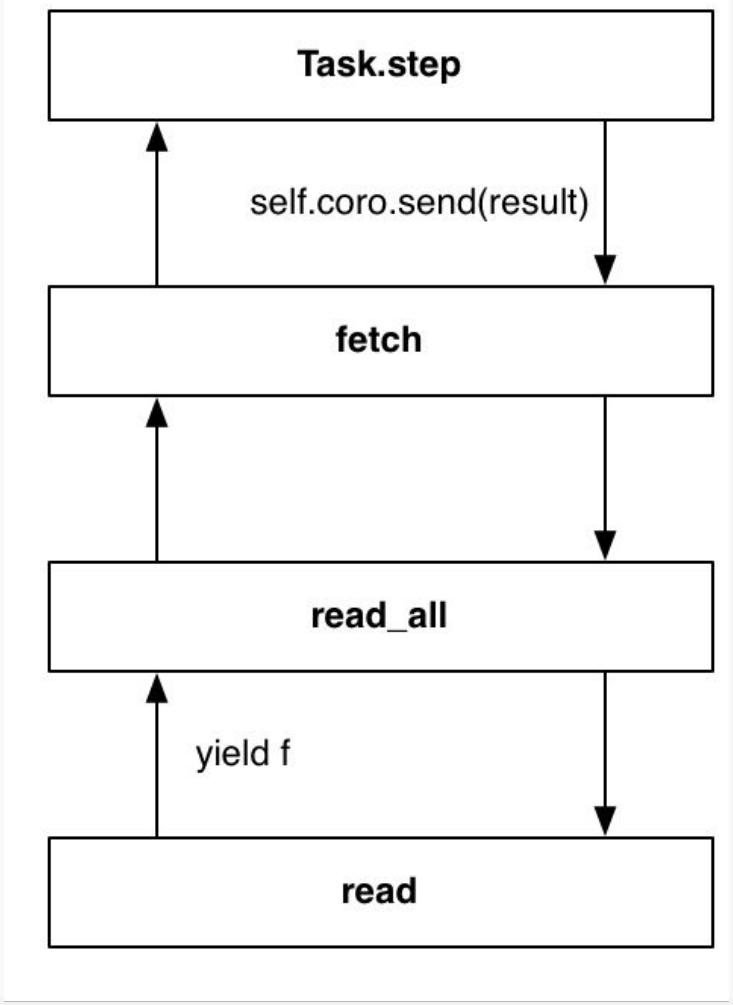

>>> fetcher.py

# yield from (1/3)

```
1   def gen_fn():
2       result = yield 1
3       print('result of yield: {}'.format(result))
4       result2 = yield 2
5       print('result of 2nd yield: {}'.format(result2))
6       return 'done'
7
8
9   def f():
10      res = yield from gen_fn()
11      print('res', res)
12
13
14  for el in f():
15      print(el)
```

# yield from (2/3)

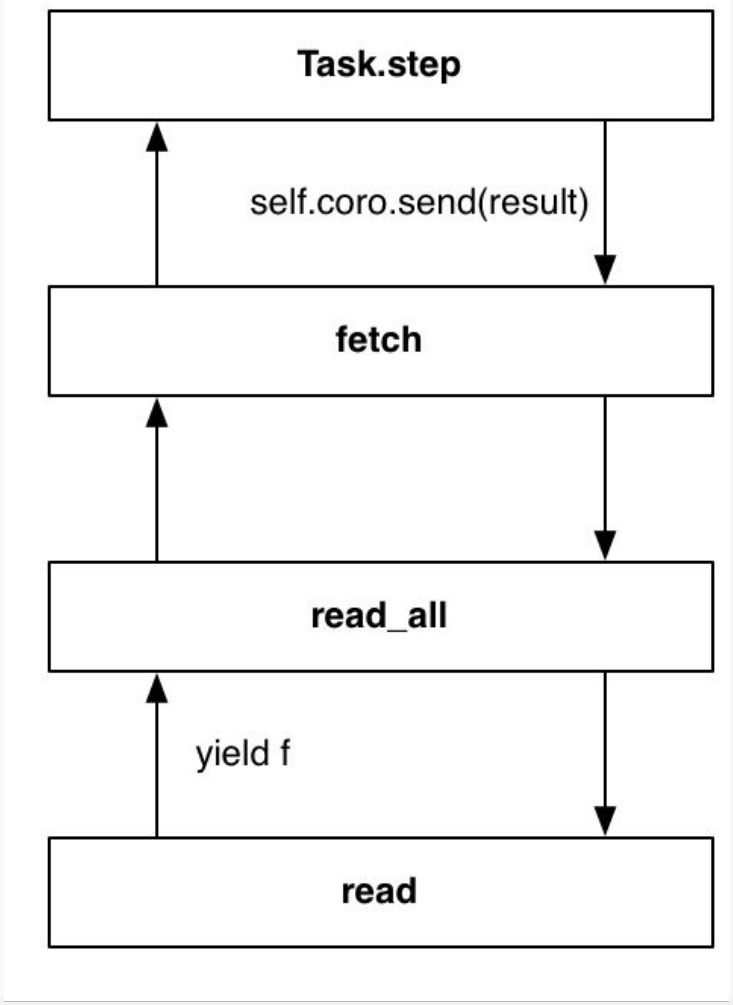
>>> fetcher2.py



# yield from (3/3)

>>> fetcher3.py

```
    def __iter__(self):
        # Tell Task to resume me here.
        yield self
        return self.result
```



# Fetcher. Wrap up.


http://www.aosabook.org/en/500L/a-web-crawler-with-asyncio-coroutines.html

# asyncio

## Hello World!

```
import asyncio

async def main():
    print('Hello ...')
    await asyncio.sleep(1)
    print('... World!')

# Python 3.7+
asyncio.run(main())
```

# asyncio


```python
import asyncio

@asyncio.coroutine
def py34_coro():
    """Generator-based coroutine, older syntax"""
    yield from stuff()

async def py35_coro():
    """Native coroutine, modern syntax"""
    await stuff()
```

**Note:** Support for generator-based coroutines is **deprecated** and is scheduled for removal in Python 3.10.

# asyncio


- 1 процесс
- 1 поток
- cooperative vs preemptive multitasking
- Передача управления в event-loop на ожидающих операциях
- async/await - это API Python, а не часть asyncio

# asyncio


```python
import asyncio

async def count():
    print("One")
    await asyncio.sleep(1)
    print("Two")

async def main():
    await asyncio.gather(count(), count(), count())

if __name__ == "__main__":
    import time
    s = time.perf_counter()
    asyncio.run(main())
    elapsed = time.perf_counter() - s
    print(f"{__file__} executed in {elapsed:0.2f} seconds.")
```

# asyncio


```
async def f(x):
    y = await z(x)  # OK - `await` and `return` allowed in coroutines
    return y

async def g(x):
    yield x  # OK - this is an async generator

async def m(x):
    yield from gen(x)  # No - SyntaxError

def m(x):
    y = await z(x)  # Still no - SyntaxError (no `async def` here)
    return y
```

# asyncio


- asyncio.ensure_future()
- asyncio.sleep()
- asyncio.wait()
- asyncio.wait_for()
- asyncio.gather()
- asyncio.Queue,
- asyncio.Lock
- asyncio.Event
- ….
- https://docs.python.org/3/library/asyncio.html

# asyncio
# web
# frameworks

# aiohttp

https://aiohttp.readthedocs.io

SANIC
FRAMEWORK

https://sanic.readthedocs.io

# aiohttp client

```python
import aiohttp
import asyncio

async def fetch(session, url):
    async with session.get(url) as response:
        return await response.text()

async def main():
    async with aiohttp.ClientSession() as session:
        html = await fetch(session, 'http://python.org')
        print(html)

loop = asyncio.get_event_loop()
loop.run_until_complete(main())
```

# aiohttp server

```python
from aiohttp import web

async def hello(request):
    return web.Response(text="Hello, world")


app = web.Application()
app.add_routes([web.get('/', hello)])

web.run_app(app)
```

# Домашнее задание №3


[10 баллов] Разработать web-crawler и поиск для сайта https://docs.python.org (или любого другого контентного)

1. Crawler должен обходить только ссылки внутри указанного домена.
2. Скачивание ресурсов должно быть реализовано в нескольких параллельных корутинах для достижения максимальной скорости обкачки.
3. Скорость обкачки должна быть параметром краулера. Например, 10 rps должно означать, что в секунду должно быть не более 10 запросов на домен.
4. Каждая страница должна быть положена в индекс elasticsearch. Можно использовать библиотеку aioelasticsearch.

[5 баллов] Разработать api, используя aiohttp или sanic, которое будет отдавать результаты поиска

/api/v1/search

1. Должен принимать следующий параметры
   a. q - текстовый запрос
   b. limit - количество результатов
   c. offset - офсет результатов
2. В ответ должен возвращать список результатов (ссылок на обкачиваемый сайт), отсортированные по релевантности

# Ссылки полезные


1. http://www.aosabook.org/en/500L/a-web-crawler-with-asyncio-coroutines.html
2. http://www.kegel.com/c10k.html
3. **https://snarky.ca/how-the-heck-does-async-await-work-in-python-3-5/**
4. https://docs.python.org/3/library/asyncio.html
5. https://realpython.com/async-io-python/


telegram: alexopryshko
email: alexopryshko@gmail.com

telegram: igorcoding
email: igor.latkin@outlook.com

Спасибо за внимание!