



WARGAMING.NET

LET'S BATTLE

про асинхронное сетевое программирование

Стас Рудаков

Да что мы всё обо мне, да обо мне...

О чем будем говорить

- ▶ Что нам дает асинхронность
- ▶ Как это работает
- ▶ Twisted
- ▶ Gevent
- ▶ Tornado
- ▶ PEP 3156



`https://raw.githubusercontent.com/nott/talks/master/async.pdf`

Работа с сетью: зачем что-то изобретать?

```
1 #!/usr/bin/env python2
2 import urllib2
3 f = urllib2.urlopen('http://python.org/')
4 print f.read(10)
```

Listing 1: urllib_example.py

Работа с сетью: зачем что-то изобретать?

```
1 #!/usr/bin/env python2
2 import urllib2
3 f = urllib2.urlopen('http://python.org/')
4 print f.read(10)
```

Listing 2: urllib_example.py

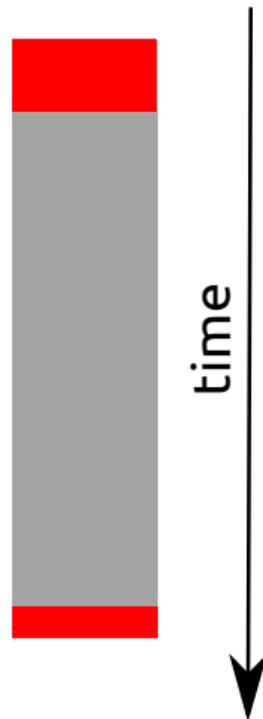
```
1 $ time python urllib_example.py
2 <!DOCTYPE
3
4 real 0m0.384s
5 user 0m0.067s
6 sys 0m0.017s
```

Работа с сетью: зачем что-то изобретать?

```
1 #!/usr/bin/env python2
2 import urllib2
3 f = urllib2.urlopen('http://python.org/')
4 print f.read(10)
```

Listing 3: urllib_example.py

```
1 $ time python urllib_example.py
2 <!DOCTYPE
3
4 real 0m0.384s
5 user 0m0.067s
6 sys 0m0.017s
```



Работа с сетью: дальше будет только хуже

```
1 #!/usr/bin/env python2
2 import sys
3 import urllib2
4 for i in xrange(5):
5     f = urllib2.urlopen('http://python.org/')
6     print f.read(10),
```

Listing 4: urllib_example2.py

Работа с сетью: дальше будет только хуже

```
1 #!/usr/bin/env python2
2 import sys
3 import urllib2
4 for i in xrange(5):
5     f = urllib2.urlopen('http://python.org/')
6     print f.read(10),
```

Listing 5: urllib_example2.py

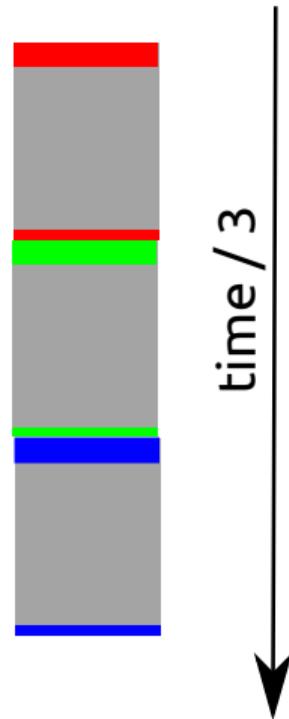
```
1 $ time python urllib_example2.py
2 <!DOCTYPE <!DOCTYPE <!DOCTYPE
3
4 real 0m1.007s
5 user 0m0.050s
6 sys 0m0.003s
```

Работа с сетью: дальше будет только хуже

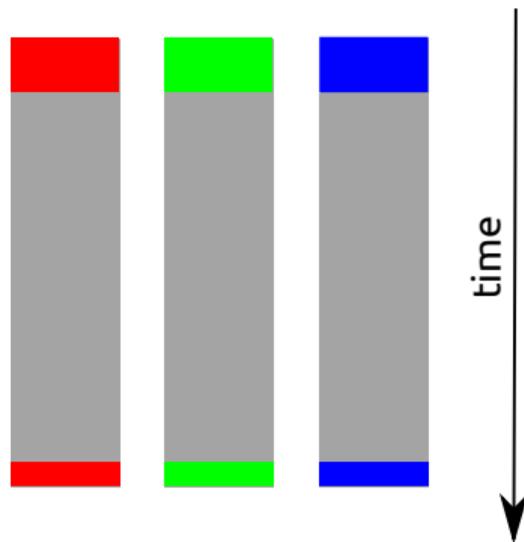
```
1 #!/usr/bin/env python2
2 import sys
3 import urllib2
4 for i in xrange(5):
5     f = urllib2.urlopen('http://python.org/')
6     print f.read(10),
```

Listing 6: urllib_example2.py

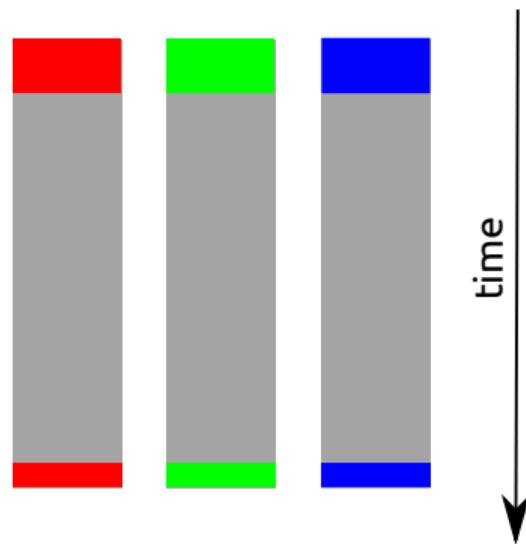
```
1 $ time python urllib_example2.py
2 <!DOCTYPE <!DOCTYPE <!DOCTYPE
3
4 real 0m1.007s
5 user 0m0.050s
6 sys 0m0.003s
```



I know, I'll use...

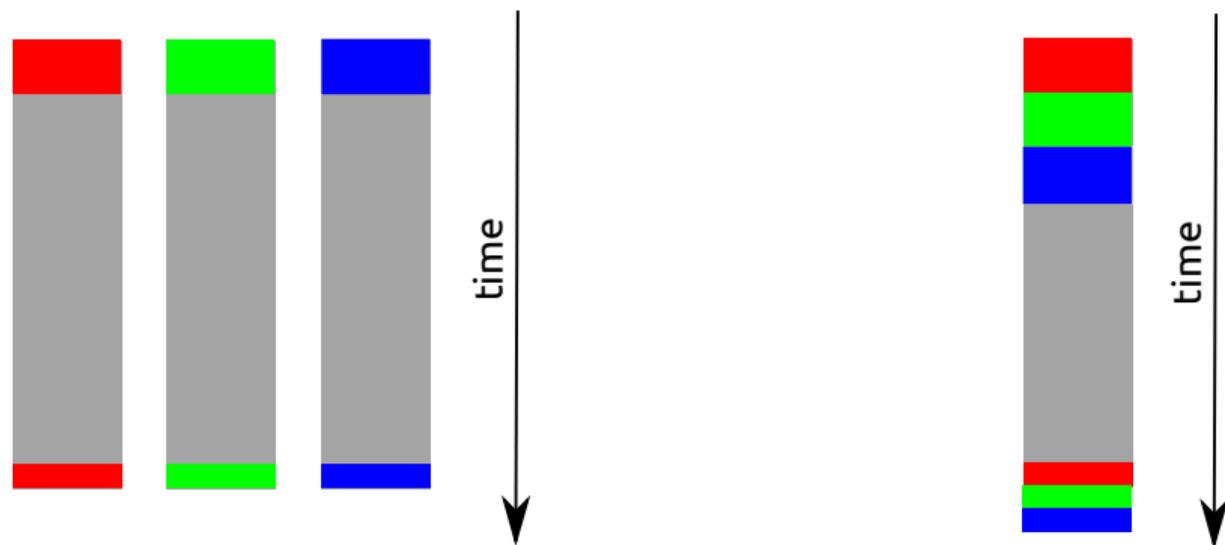


I know, I'll use...



*Some people, when confronted with a problem, think,
"I know, I'll use threads,"
and then two of them have problems. (Ned Batchelder)*

I know, I'll use...

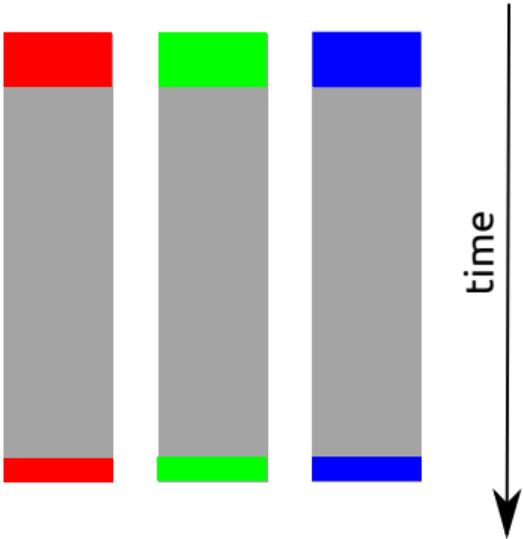


*Some people, when confronted with a problem, think,
"I know, I'll use threads,"
and then two they hav erpoblesms. (Ned Batchelder)*

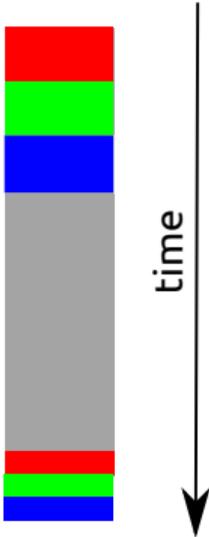
Apache vs Nginx

Apache vs Nginx

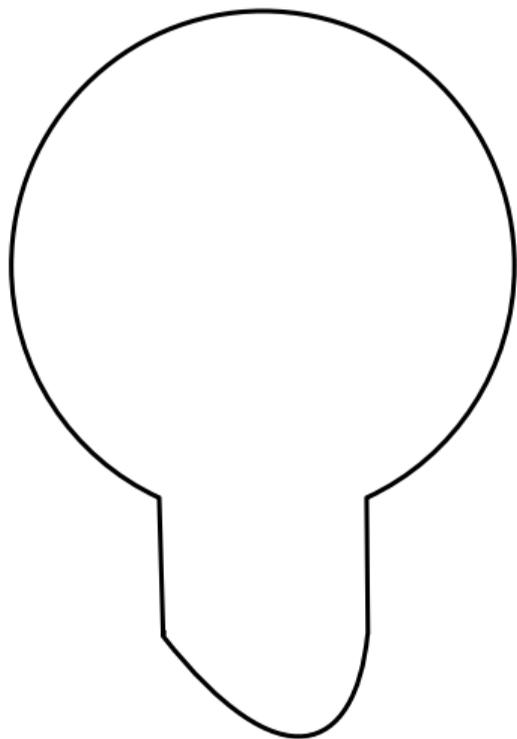
Apache



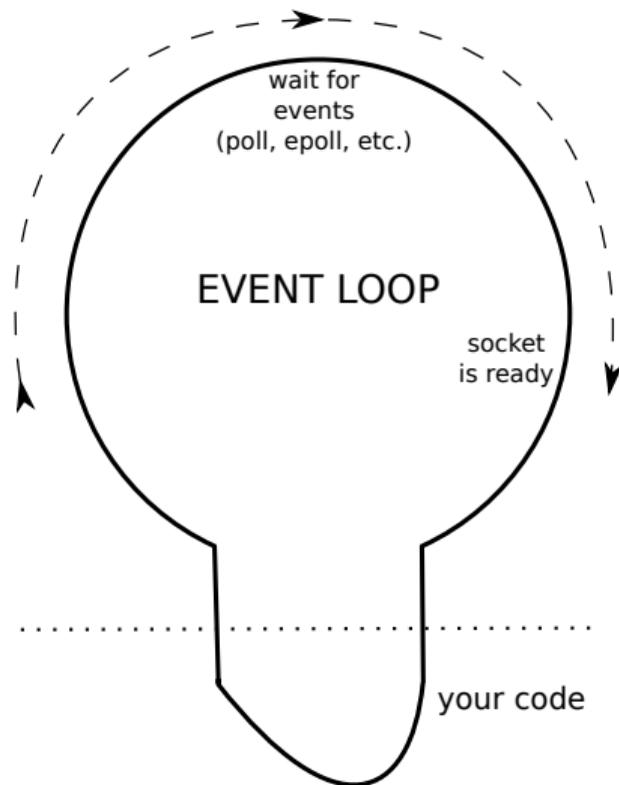
Nginx



Лампочка

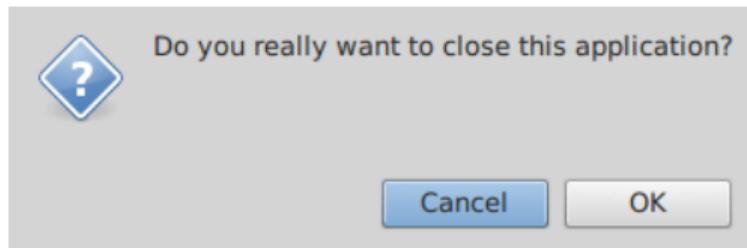


Event loop



Event loop in GUI

```
1     def OnClose(self, event):
2         dlg = wx.MessageDialog(self,
3             "Do you really want to close this application?",
4             "Confirm Exit", wx.OK|wx.CANCEL|wx.ICON_QUESTION)
5         result = dlg.ShowModal()
6         dlg.Destroy()
7         if result == wx.ID_OK:
8             self.Destroy()
```





Twisted

Twisted: transport vs protocol

Транспорт

- ▶ знает, как пересылать байты
- ▶ примеры: socket, UNIX pipe, SSL socket, etc.

Протокол

- ▶ знает, какие байты пересылать
- ▶ примеры: HTTP, SMTP, XMPP, Memcached, etc.

Twisted: transport vs protocol на примере

```
1 #!/usr/bin/env python2
2 from twisted.internet import reactor, protocol
3 class Counter(protocol.Protocol):
4     def __init__(self, factory):
5         self.factory = factory
6     def connectionMade(self):
7         num = self.factory.num = self.factory.num + 1
8         self.transport.write("open connections: %d\n" % num)
9     def connectionLost(self, reason):
10        self.factory.num -= 1
11    dataReceived = lambda self, data: None
12 class CounterFactory(protocol.Factory):
13     def __init__(self):
14         self.num = 0
15     buildProtocol = lambda self, addr: Counter(self)
16 reactor.listenTCP(8123, CounterFactory())
17 reactor.run()
```

Twisted: transport vs protocol

```
1 $ telnet 127.0.0.1 8123
2 Trying 127.0.0.1...
3 Connected to 127.0.0.1.
4 Escape character is '^]'.
5 open connections: 2
```

Twisted: Deferred

- ▶ Deferred — обещание вернуть результат.
- ▶ К нему прикрепляются callback'и и errback'и.
- ▶ Само использование Deferred еще не делает программу асинхронной.
- ▶ Важно, чтобы callback'и вызывались по событиям (в любых смыслах).

Twisted: Deferred на примере

```
1 #!/usr/bin/env python2
2 import sys
3 from twisted.internet import defer, reactor
4 from twisted.python.util import println
5 from twisted.web.client import getPage
6
7 results = []
8 for i in xrange(int(sys.argv[1])):
9     d = getPage('http://www.python.org/')
10    d.addCallback(lambda value: value[:10])
11    d.addCallback(lambda value: println(value[:: -1]))
12    d.addErrback(lambda error: println("error:", error))
13    results.append(d)
14
15 d_list = defer.DeferredList(results)
16 d_list.addBoth(lambda ignored: reactor.stop())
17 reactor.run()
```

Twisted: Deferred

```
1 $ time python twisted_example.py 1
2  EPYTCOD!<
3
4 real  0m0.833s
5 user  0m0.250s
6 sys   0m0.030s
7 $ time python twisted_example.py 5
8  EPYTCOD!<
9  EPYTCOD!<
10 EPYTCOD!<
11 EPYTCOD!<
12 EPYTCOD!<
13
14 real  0m1.116s
15 user  0m0.257s
16 sys   0m0.043s
```

Twisted: параллельная вселенная

- ▶ последовательность синхронных операторов \approx цепочка `Deferred`;
- ▶ вызов функции внутри функции \approx возврат `Deferred` из `Deferred`;
- ▶ `Deferred` \approx `stack frame`;
- ▶ блок `try...except` \approx цепочка `errback`'ов;
- ▶ `threading.join` \approx `DeferredList`;

Twisted: параллельная вселенная

- ▶ последовательность синхронных операторов \approx цепочка `Deferred`;
- ▶ вызов функции внутри функции \approx возврат `Deferred` из `Deferred`;
- ▶ `Deferred` \approx `stack frame`;
- ▶ блок `try...except` \approx цепочка `errback`'ов;
- ▶ `threading.join` \approx `DeferredList`;

- ▶ `logging`;
- ▶ `plugins`;
- ▶ несовместимость с `stdlib`;
- ▶ отлично проработанная инфраструктура;
- ▶ достаточно (но далеко не самый) быстрый;

Gevent

и прекрасный код

Gevent

```
1 #!/usr/bin/python2
2 import sys
3 from gevent import joinall, monkey, spawn
4
5 monkey.patch_all()
6
7 import urllib2
8
9 def fetch():
10     f = urllib2.urlopen('http://www.python.org/')
11     print f.read(10)
12
13 num = int(sys.argv[1])
14 # equivalent can be also implemented with threading
15 joinall([spawn(fetch) for i in xrange(num)])
```

Gevent: oh wow!

```
1 $ time python gevent_example.py 1
2 <!DOCTYPE
3
4 real 0m0.434s
5 user 0m0.077s
6 sys 0m0.020s
7
8 $ time python gevent_example.py 5
9 <!DOCTYPE
10 <!DOCTYPE
11 <!DOCTYPE
12 <!DOCTYPE
13 <!DOCTYPE
14
15 real 0m0.428s
16 user 0m0.093s
17 sys 0m0.023s
```

Greenlets inside

```
1 #!/usr/bin/env python
2 from greenlet import greenlet
3
4 def test1():
5     print(1)
6     gr2.switch()
7     print(3)
8 def test2():
9     print(2)
10    gr1.switch()
11    print(4)
12
13 gr1 = greenlet(test1)
14 gr2 = greenlet(test2)
15 gr1.switch()
```

```
$ python greenlet_example.py
1
2
3
```

Listing 8: greenlet_example.py

Gevent: есть нюанс

- ▶ Текущая версия 0.13.
- ▶ В версии 1.0 запланирован переход с `libevent` на `libev`.

Gevent: есть нюанс

- ▶ Текущая версия 0.13.
- ▶ В версии 1.0 запланирован переход с `libevent` на `libev`.
- ▶ Первая β -версия для 1.0 была выпущена около 2 лет назад.

Gevent: есть нюанс

- ▶ Текущая версия 0.13.
- ▶ В версии 1.0 запланирован переход с `libevent` на `libev`.
- ▶ Первая β -версия для 1.0 была выпущена около 2 лет назад.
- ▶ Пугают страшилки про гринлеты.

Gevent: есть нюанс

- ▶ Текущая версия 0.13.
- ▶ В версии 1.0 запланирован переход с `libevent` на `libev`.
- ▶ Первая β -версия для 1.0 была выпущена около 2 лет назад.
- ▶ Пугают страшилки про гринлеты.
- ▶ Проблемы с совместимостью с существующими библиотеками.

Tornado

и быстрые HTTP серверы

Tornado

```
1 #!/usr/bin/env python
2 import tornado.ioloop
3 import tornado.httpclient
4 import tornado.web
5
6 class Handler(tornado.web.RequestHandler):
7     @tornado.web.asynchronous
8     def get(self, path):
9         http = tornado.httpclient.AsyncHTTPClient()
10        http.fetch("http://www.python.org/" + path,
11                self._on_download)
12    def _on_download(self, response):
13        self.write(response.body)
14        self.finish()
15 application = tornado.web.Application([(r"/(.*)" , Handler),])
16 application.listen(8888)
17 tornado.ioloop.IOLoop.instance().start()
```

Tornado: we need an HTTP client!

```
1 #!/usr/bin/env python
2 import sys, tornado.ioloop, tornado.httpclient
3
4 def fetch():
5     http = tornado.httpclient.AsyncHTTPClient()
6     http.fetch("http://www.python.org/", on_download)
7 def on_download(response):
8     print(response.body[:10])
9     global num
10    num -= 1
11    if not num: ioloop.stop()
12
13 num = int(sys.argv[1])
14 ioloop = tornado.ioloop.IOLoop.instance()
15 for i in range(num):
16     ioloop.add_callback(fetch)
17 ioloop.start()
```

Tornado: the results

```
1 $ time python gevent_example.py 5
2 <!DOCTYPE
3 <!DOCTYPE
4 <!DOCTYPE
5 <!DOCTYPE
6 <!DOCTYPE
7
8 real 0m0.428s
9 user 0m0.093s
10 sys 0m0.023s
11
12 $ time python tornado_httpclient.py 1
13 <!DOCTYPE
14
15 real 0m0.469s
16 user 0m0.163s
17 sys 0m0.027s
```

PEP 3156

Asynchronous IO Support Rebooted

PEP 3156: ключевые моменты

- ▶ Разделение на транспорт и протокол — как в Twisted.
- ▶ Future — объект обещания — подсмотрен в Java Futures.
- ▶ API построено на `yield from` (до свидания, Python 2).
- ▶ Event loop — один на поток.
- ▶ Tulip — reference implementation.
- ▶ Вероятно, Tulip войдет в Python 3.4 под каким-нибудь скучным именем (осень 2013).

yield from на примере

```
1 #!/usr/bin/env python3
2
3 def a():
4     yield 1
5     yield 2
6     yield 3
7     return 'done_1'
8
9 def b():
10    done = yield from a()
11    print(done)
12    return 'done_b'
13
14 for i in b():
15    print(i)
```

Listing 9: yield_from.py

```
$ python yield_from.py
1
2
3
done_1
```

Tulip — reference implementation

```
1 #!/usr/bin/env python3
2 import sys
3 import tulip
4 import tulip.http
5
6 def curl(url):
7     response = yield from tulip.http.request('GET', url)
8     data = yield from response.read()
9     print(data.decode('utf-8', 'replace')[:10])
10
11 if __name__ == '__main__':
12     loop = tulip.get_event_loop()
13     loop.run_until_complete(curl(sys.argv[1]))
```

“Эй ты, заканчивай там!”

- ▶ Asynchronous networking — это правильный инструмент... для правильной задачи.
- ▶ Python настолько крутой, что допускает как минимум 3 возможных подхода для написания асинхронных сетевых фреймворков.
- ▶ PEP 3156 выглядит многообещающе.

“Эй ты, заканчивай там!”

- ▶ Asynchronous networking — это правильный инструмент... для правильной задачи.
- ▶ Python настолько крутой, что допускает как минимум 3 возможных подхода для написания асинхронных сетевых фреймворков.
- ▶ PEP 3156 выглядит многообещающе.
- ▶ Стас, ты забыл рассказать про совместимость на уровне event loop'ов!

СПАСИБО ЗА ВНИМАНИЕ. ВОПРОСЫ?

Стас Рудаков

<mailto:stas@garage22.net>

mailto:s_rudakou@wargaming.net

<https://raw.githubusercontent.com/nott/talks/master/async.pdf>

<http://twistedmatrix.com/>

<http://www.tornadoweb.org>

<http://www.gevent.org/>

<http://www.python.org/dev/peps/pep-3156/>

<https://code.google.com/p/tulip/>