

# Tulip

## Новый стандарт на асинхронный код

Андрей Светлов

[andrew.svetlov@gmail.com](mailto:andrew.svetlov@gmail.com)

<http://asvetlov.blogspot.com>

# Существующие подходы

- Многопоточность
-

# Существующие подходы

- Многопоточность
- Обратные вызовы (twisted, tornado)
-

# Существующие подходы

- Многопоточность
- Обратные вызовы (twisted, tornado)
- Greenlet (gevent, eventlet)
-

# Существующие подходы

- Многопоточность
  - Обратные вызовы (twisted, tornado)
  - Greenlet (gevent, eventlet)
  - Yield from (tulip)
- 
- Каждый вариант несовместим с другими.
  - Универсальные библиотеки невозможны

# Tulip

- PEP 3156 — *Asynchronous IO Support Rebooted*
- Планируется попасть в Python 3.4 этой осенью
- Должен исполняться на Python 3.3
- Python 2.x остался за бортом

# yield from

- Основа построения пользовательского кода в tulip
- Место переключения контекста

# yield

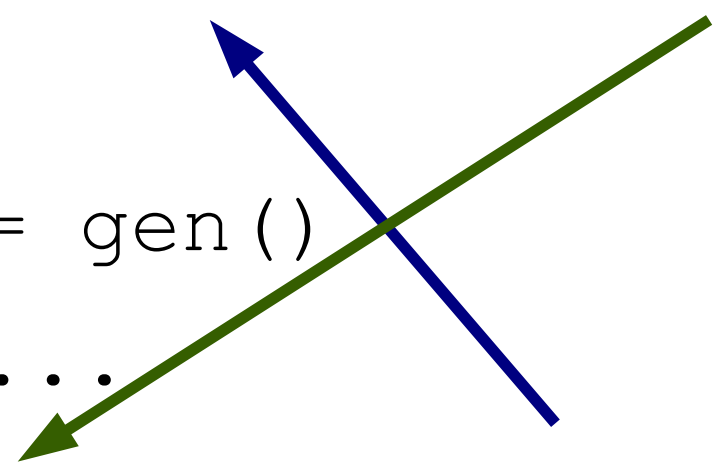
```
def gen():  
    # ...  
    val = yield 2
```

```
g = gen()  
# ...  
ret = g.send(1)
```



# yield

```
def gen():  
    # ...  
    val = yield 2  
  
g = gen()  
# ...  
ret = g.send(1)
```



# yield from

```
def a():  
    yield 1
```

```
def b():  
    yield a()
```

# yield from

```
def a():  
    yield 1
```

```
def b():  
    for i in a():  
        yield i          # <== .send() ???
```

# yield from

```
def a():  
    yield 1
```

```
def b():  
    yield from a()
```

# Эквивалент

```
_i = iter(EXPR)
try:
    _y = next(_i)
except StopIteration as _e:
    _r = _e.value
else:
    while 1:
        try:
            _s = yield _y
        except GeneratorExit as _e:
            try:
                _m = _i.close
            except AttributeError:
                pass
            else:
                _m()
            raise _e
```

```
except BaseException as _e:
    _x = sys.exc_info()
    try:
        _m = _i.throw
    except AttributeError:
        raise _e
    else:
        try:
            _y = _m(*_x)
        except StopIteration as _e:
            _r = _e.value
            break
    else:
        try:
            if _s is None:
                _y = next(_i)
            else:
                _y = _i.send(_s)
        except StopIteration as _e:
            _r = _e.value
            break
RESULT = _r
```

# Верхний слой tulip

- tulip.task
- tulip.coroutine
- scheduler
- Future

# Простейший пример

```
@tulip.task
def sleeper():
    for i in range(5):
        print(i)
        yield from tulip.sleep(1.5)
```

# HTTP запрос

```
@tulip.task
def curl(url):
    response = yield from \
        tulip.http.request('GET', url)
    print(repr(response))

    data = yield from response.read()
    print(data.decode('utf-8', 'replace'))

curl('http://example.com')
```



# Чуть глубже

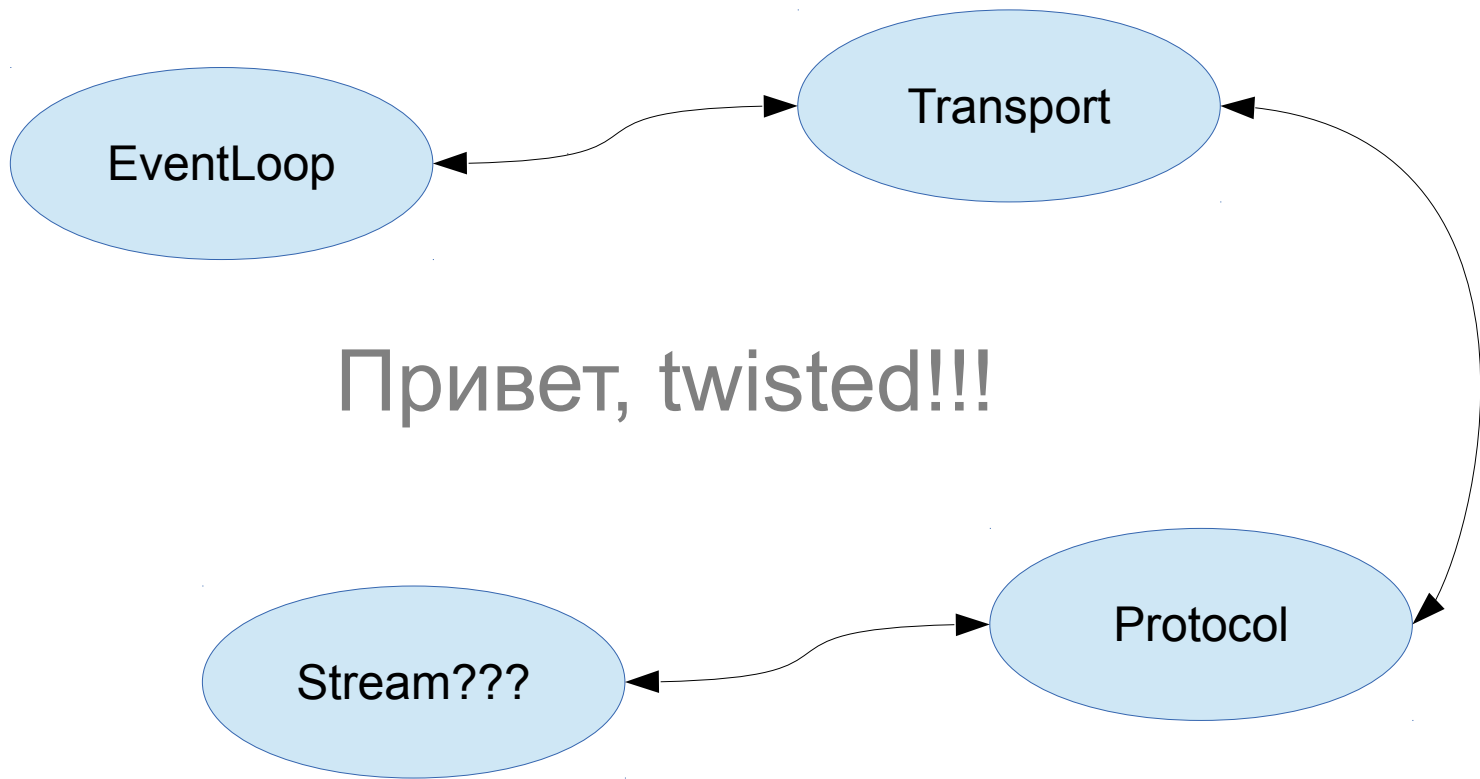
- `event_loop`
- «Сердце» системы
- Пишется один раз создателями `tulip`
- Единственный в своем контексте (по умолчанию один на поток)

```
loop = tulip.get_event_loop()  
loop.run_forever()
```

# Работа с сетью

```
loop = tulip.get_event_loop()  
loop.start_serving(MyProto,  
                   'example.com', 7777)  
loop.run_forever()
```

# Протокол и транспорт



# Транспорт

- Канал для передачи данных
- Знает, *как* пресылать байты
- Примеры: socket, ssl socket, unix pipe, subprocess, datagramm socket, serial port, inotify descriptor
- Программист использует готовые транспорты из довольно ограниченного набора, большая часть из которого уже реализована в tulip

# Протокол

- Знает, *какие* байты нужно слать
- Соответствует протоколам пользовательского уровня: HTTP, SMTP, FTP, Redis и т.д.
- Создаются авторами библиотек, не «простым» программистом
- В свою очередь предоставляют *удобный API* для работы по этому протоколу

# Пример протокола

```
class MyProto(tulip.Protocol):  
    def connection_made(self,  
                          transp):  
        self.transp = transp  
    def data_received(self, data):  
        self.transp.write(b'echo'+  
                           data)
```

# HTTP

```
resp = yield from tulip.http.request('GET',  
                                     'http://python.org/')
```

```
resp # <HttpResponse(python.org/) [200]>
```

```
resp.status # 200
```

```
resp.reason # OK
```

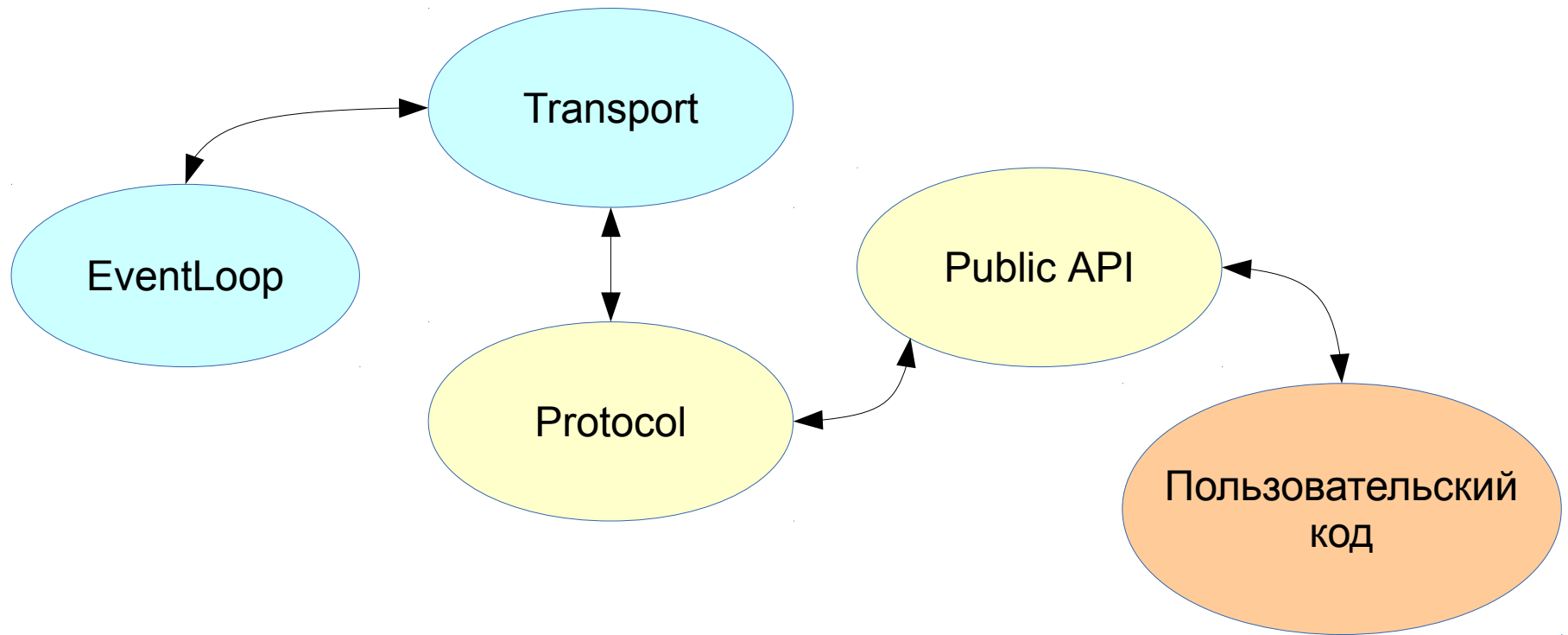
```
data = yield from resp.content.read()
```

# Под капотом у HTTP

- GET, POST, PUT и т.д.
- HTTP headers
- cookies
- загрузка файлов
- авторизация
- редиректы
- сессии
- gzip сжатие
- web сокеты
- и т.д.



# Взаимодействие частей



# Куда всё идет?

- Tulip станет частью стандартной библиотеки до ноября 2013
- Существующие части библиотеки для работы с сетью будут переписаны со временем на использование tulip: urllib, httplib и т.д.
- Twisted, tornado, gevent и т.д. могут иметь event loop совместимый с tulip

# Echo server

```
class EchoServer(tulip.Protocol):
    TIMEOUT = 5.0
    def timeout(self):
        self.transport.close()

    def connection_made(self, transport):
        self.transport = transport
        # start 5 seconds timeout timer
        self.h_timeout = \
            tulip.get_event_loop() \
                .call_later(
                    self.TIMEOUT, self.timeout)

    def data_received(self, data):
        self.transport.write(b'Re: '
                               +data)
        # restart timeout timer
        self.h_timeout.cancel()
        self.h_timeout = \
            tulip.get_event_loop() \
                .call_later(
                    self.TIMEOUT, self.timeout)

    def eof_received(self):
        pass

    def connection_lost(self, exc):
        self.h_timeout.cancel()

def start_server(loop, host, port):
    f = loop.start_serving(EchoServer, host, port)
    x = loop.run_until_complete(f)[0]
    print('serving on', x.getsockname())
```

# Echo Client

```
class EchoClient(tulip.Protocol):
    message = 'This is the message. It
will be echoed.'

    def connection_made(self,
transport):
        self.transport = transport
        self.transport.write(
            self.message.encode())

    def data_received(self, data):
        # disconnect after 10 seconds
        tulip.get_event_loop()\
            .call_later(10.0,
                self.transport.close)

def start_client(loop, host, port):
    t = tulip.Task(loop.create_connection(EchoClient,
        host, port))
    loop.run_until_complete(t)

def eof_received(self):
    pass

def connection_lost(self, exc):
    tulip.get_event_loop().stop()
```

Вопросы?

Tulip

Новый стандарт на асинхронный код

Андрей Светлов

[andrew.svetlov@gmail.com](mailto:andrew.svetlov@gmail.com)

[asvetlov.blogspot.com](http://asvetlov.blogspot.com)