Tornado

A tale of love, happiness, and asynchronous Python

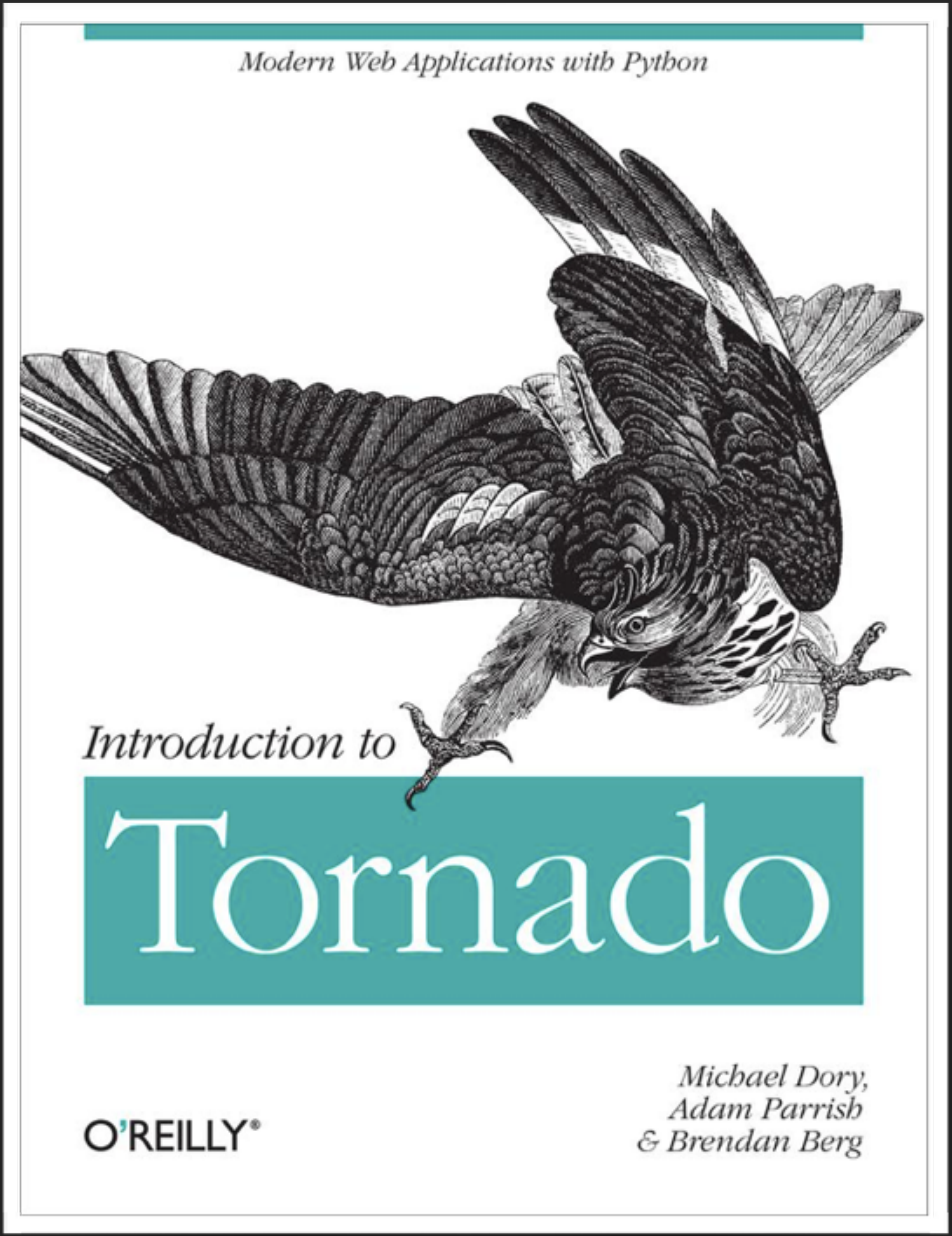
Michael Dory (Spies & Assassins)

Python User Group in Princeton, November 11, 2013

Introductions

SPIES/ASSASSINS

(WE'RE HIRING)



Modern Web Applications with Python

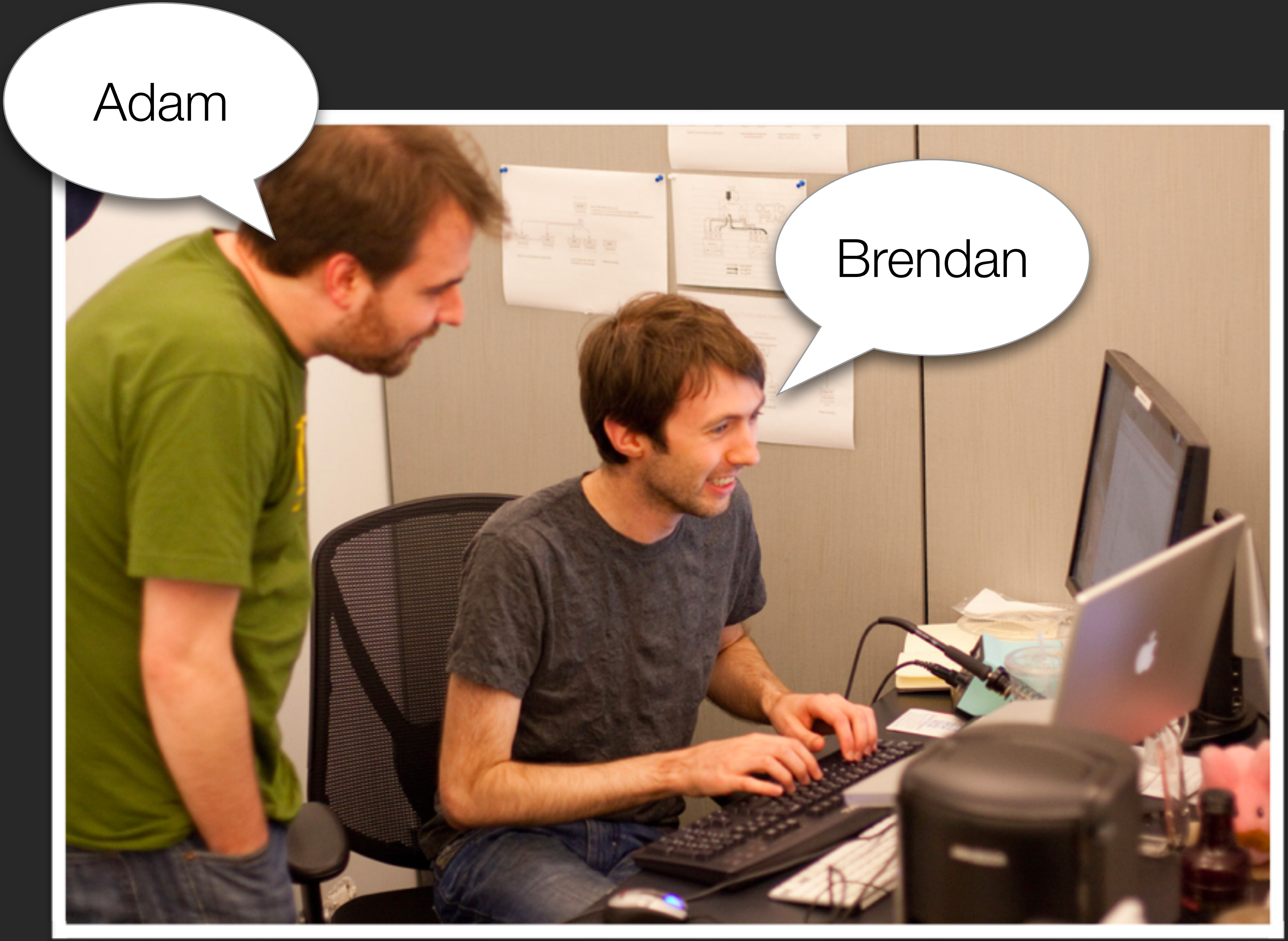
Introduction to

Tornado

Michael Dory,
Adam Parrish
& Brendan Berg

O'REILLY®

The story


Adam
Brendan

In short:
we needed an awesome framework for writing APIs

Tornado!

- Scalable, non-blocking web server
- Originally developed at Friendfeed
- Open-sourced by Facebook
- Actively maintained and community-supported
- Makes developers happy

Tornado - what it is

- Scalable and fast
- Non-blocking (and/or asynchronous)
- Lightweight
- Flexible yet robust
- Pure Python goodness

Tornado - what it is *not*

- Django/Pyramid (or Rails, etc.) replacement
- Loaded with admin tools and CMS options
- Full of dependencies on libraries and practices
- Stand-alone front-end server
- Static file server*

Python framework alternatives (then)

~~django~~

~~Bottle~~

~~web.py~~

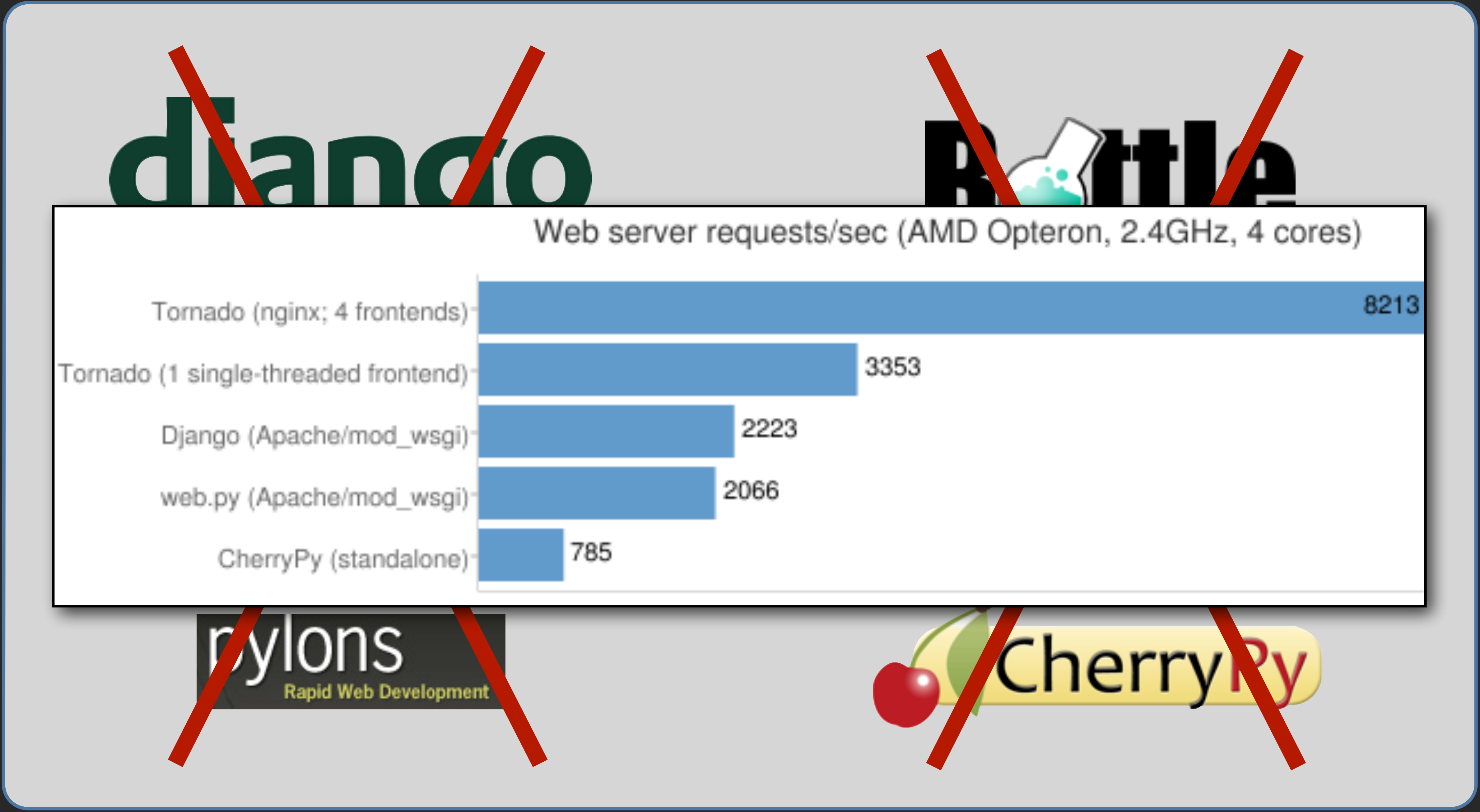
~~pylons~~
~~Rapid Web Development~~

~~CherryPy~~

Python framework alternatives (now)

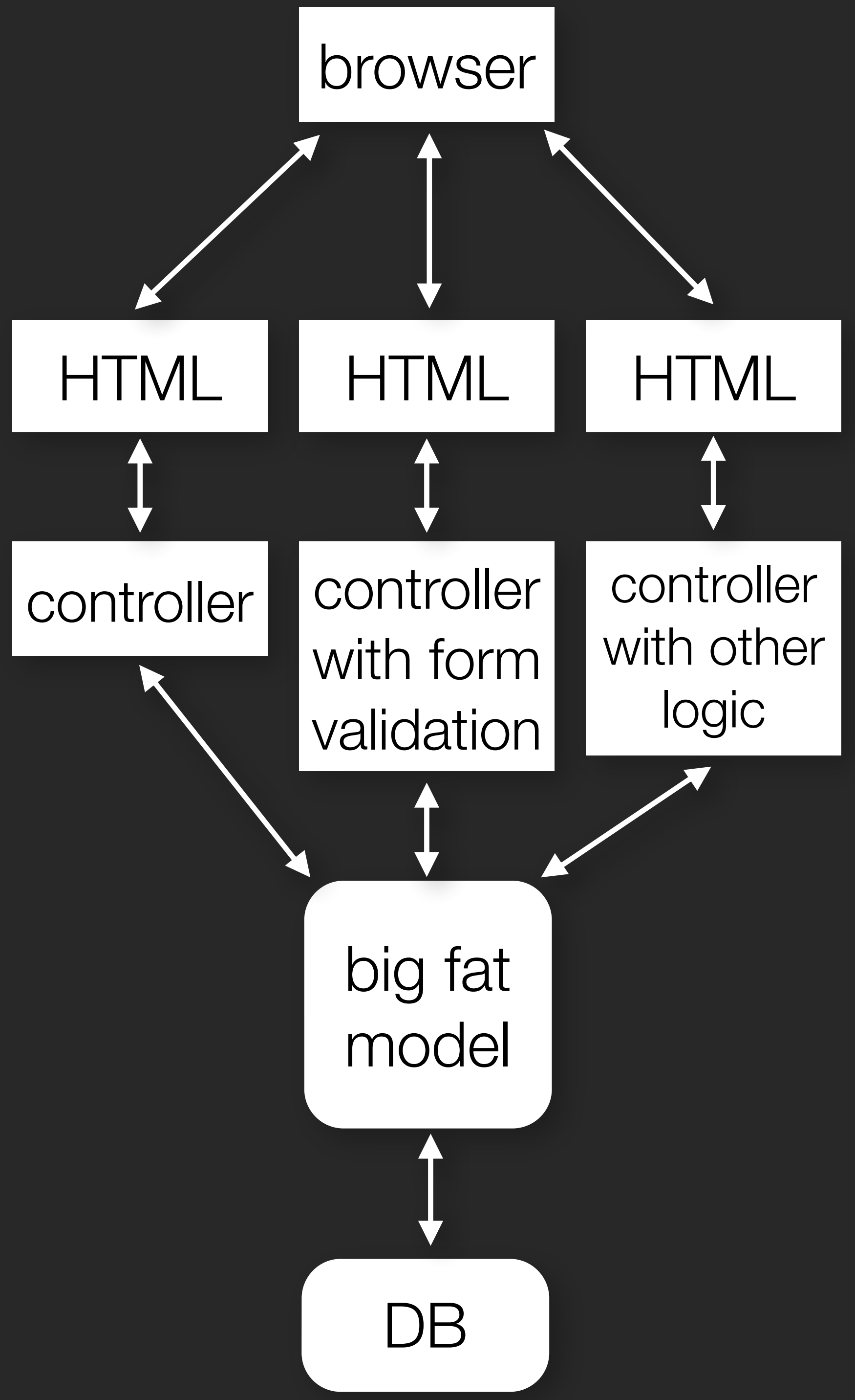
django
Flask
web development
one drop at a time

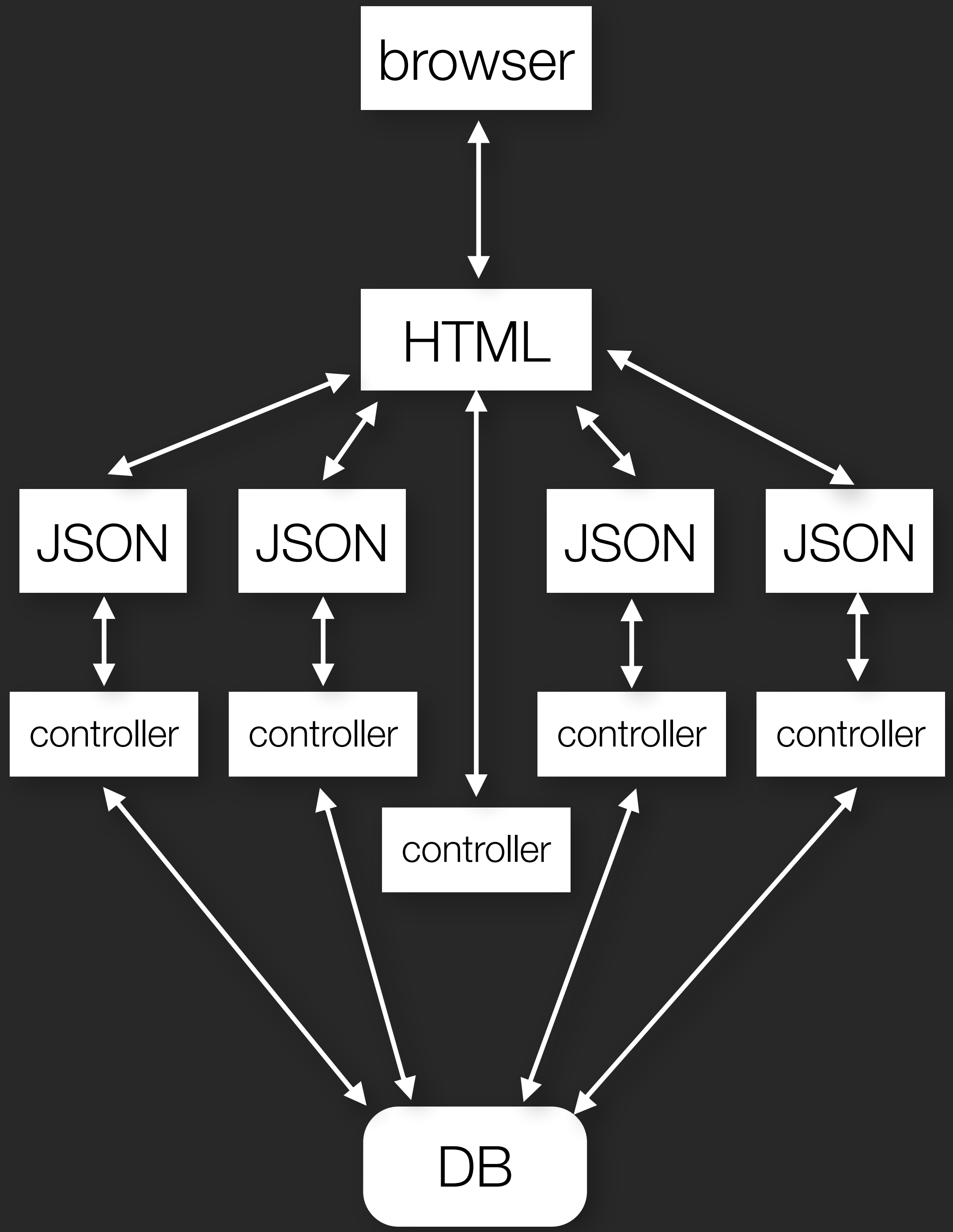
Python framework speeds (then)



Why Tornado, and not one of the others?

Different ways of thinking





Why it might be great for you

- Quick to set up and run
- Object-oriented web handlers
- Lightweight (but powerful!) template system
- No requirements for storage frameworks

What it’s perfect for

- Analytic collection (or anything real-time and write-heavy)
- Writing lightweight API’s
- Social network integration
- Web applications that are I/O bound
- Apps that make use of long polling/websockets

Who else is using it?

Quora turntable

bit.ly 10gen hipmunk

betaworks FINDINGS

So, that’s nice.
What does it look like?

```python
import os.path
import tornado.httpserver
import tornado.ioloop
import tornado.options
import tornado.web

from tornado.options import define
define("port", default=5000, help="run on the given port", type=int)

class Application(tornado.web.Application):
    def __init__(self):
        handlers = [
            (r"/([^/]+)?", MainHandler)
        ]
        settings = dict(
            template_path=os.path.join(os.path.dirname(__file__), "templates"),
            static_path=os.path.join(os.path.dirname(__file__), "static"),
            debug=True,
        )
        tornado.web.Application.__init__(self, handlers, **settings)


class MainHandler(tornado.web.RequestHandler):
    def get(self):
       self.render(
            "main.html",
            page_title="Tornado App Example"
       )


def main():
    tornado.options.parse_command_line()
    http_server = tornado.httpserver.HTTPServer(Application())
    http_server.listen(tornado.options.options.port)
    tornado.ioloop.IOLoop.instance().start()


if __name__ == "__main__":
    main()
```

```python
define("port", default=5000, help="run on the given port", type=int)

class Application(tornado.web.Application):
    def __init__(self):
        handlers = [
            (r"/([^/]+)?", MainHandler)
        ]
        settings = dict(
            template_path=os.path.join(os.path.dirname(__file__), "templates"),
            static_path=os.path.join(os.path.dirname(__file__), "static"),
            debug=True,
        )
        tornado.web.Application.__init__(self, handlers, **settings)
```

```python
# main.py
class MainHandler(tornado.web.RequestHandler):
    def get(self):
        self.render(
            'main.html',
            page_title='Tornado App Example',
            page_heading='Showing a template example',
            page_items=['One', 'Two', 'Three']
        )
```

```html
<!-- main.html -->
    <div id="main">
        <div id="container">
            <h1>{{ page_heading }}</h1>
            <ul>
                {% for item in page_items %}
                <li>{{ item }}</li>
                {% end %}
            </ul>
        </div><!-- end container -->
    </div><!-- end main -->
```

```python
# main.py
class MainHandler(tornado.web.RequestHandler):
    def get(self):
        self.render(
            'main.html',
            page_title='Tornado App Example',
            page_heading='Showing a template example',
            page_items=['One', 'Two', 'Three']
        )
```

```html
<!-- main.html -->
<head>
    <meta charset="utf-8">
    <meta http-equiv=“X-UA-Compatible" content="IE=edge">
    <title>{{ page_title }}</title>
    <link rel="stylesheet" href="{{ static_url("css/style.css")
}}" />
</head>
```

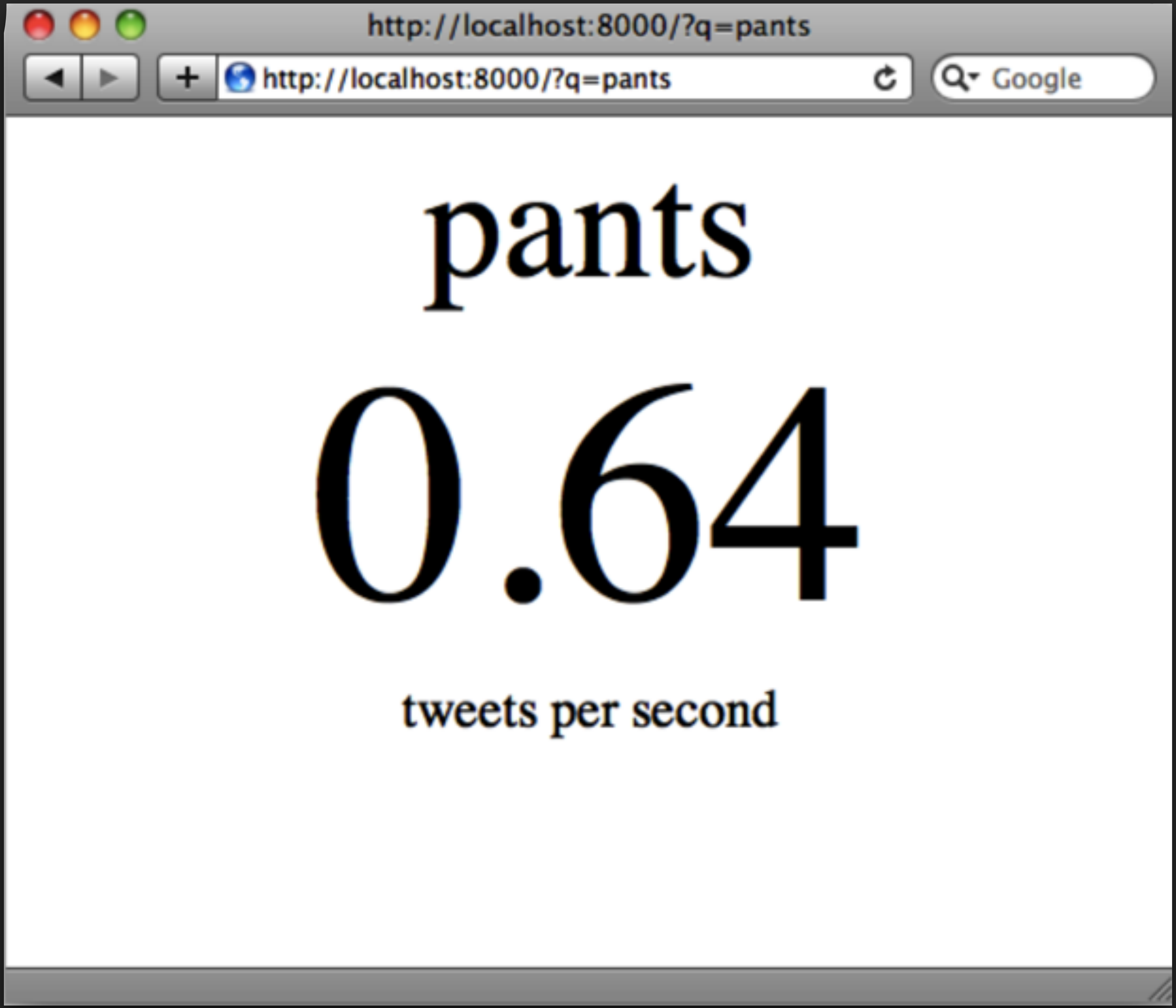
```python
# main.py
class MainHandler(tornado.web.RequestHandler):
    def get(self):
        self.render(
            'main.html',
            page_title='Tornado App Example',
            page_heading='Showing a template example',
            page_items=['One', 'Two', 'Three']
        )
```

```html
<!-- list-page.html –>
{% extends “main.html" %}

{% block main %}
<div id="container">
    <h1>{{ page_heading }}</h1>
    <ul>
        {% for item in page_items %}
        <li>{{ item }}</li>
        {% end %}
    </ul>
</div><!-- end container -->
{% end %}
```

What about the asynchronous stuff?

Okay. Here’s a Twitter API example.
About pants.



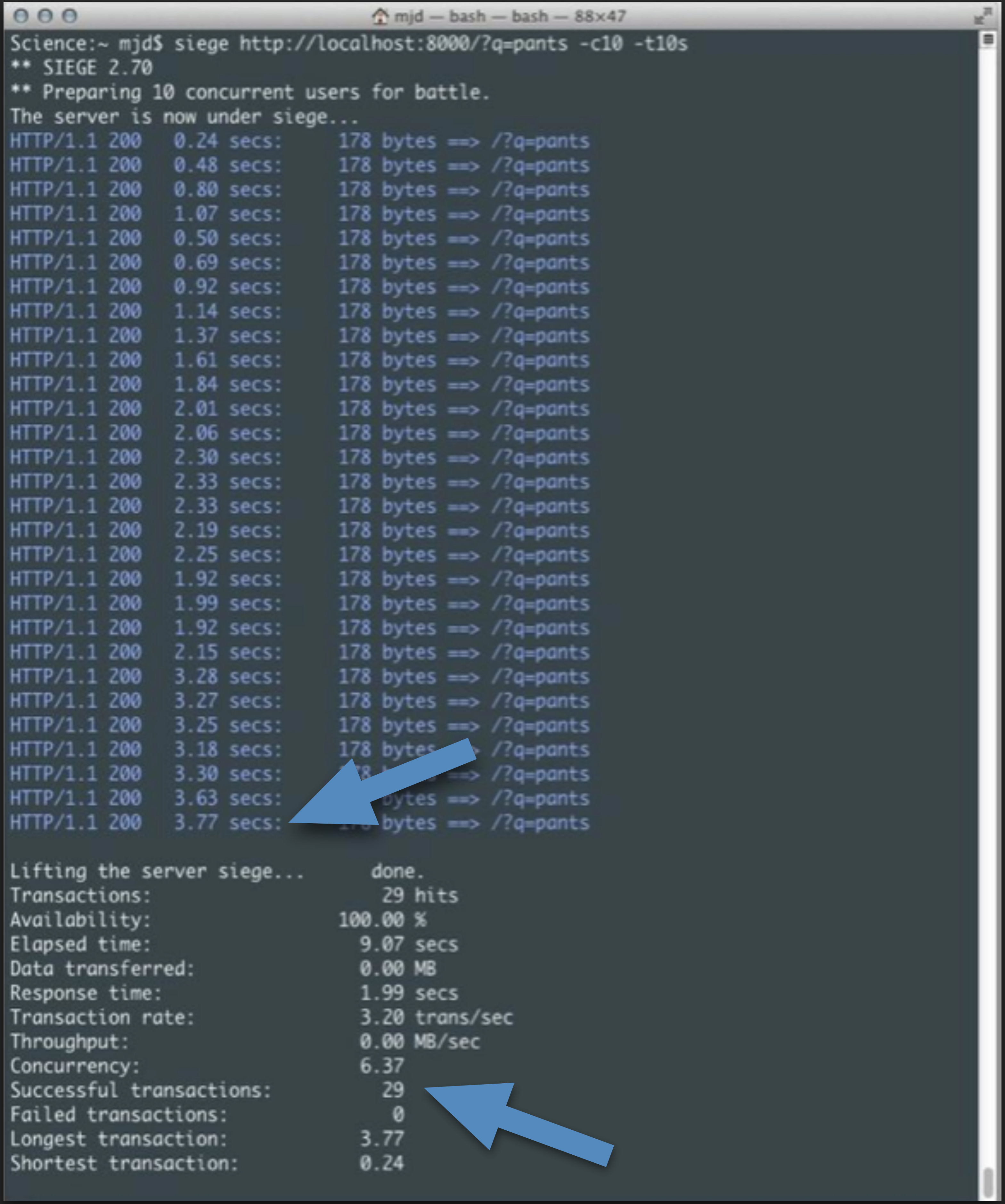
pants

0.64

tweets per second

```python
class IndexHandler(tornado.web.RequestHandler):
        def get(self):
                query = self.get_argument('q')
                client = tornado.httpclient.HTTPClient()
                response = client.fetch("http://search.twitter.com/search.json?" + \
                                urllib.urlencode(
                                     {"q": query, "result_type": "recent", "rpp": 100}))
                body = json.loads(response.body)
                result_count = len(body['results'])
                now = datetime.datetime.utcnow()
                raw_oldest_tweet_at = body['results'][-1]['created_at']
                oldest_tweet_at = datetime.datetime.strptime(raw_oldest_tweet_at,
                                "%a, %d %b %Y %H:%M:%S +0000")
                seconds_diff = time.mktime(now.timetuple()) - \
                                time.mktime(oldest_tweet_at.timetuple())
                tweets_per_second = float(result_count) / seconds_diff

                self.write("""
<div style="text-align: center">
        <div style="font-size: 72px">%s</div>
        <div style="font-size: 144px">%.02f</div>
        <div style="font-size: 24px">tweets per second</div>
</div>""" % (query, tweets_per_second))
```

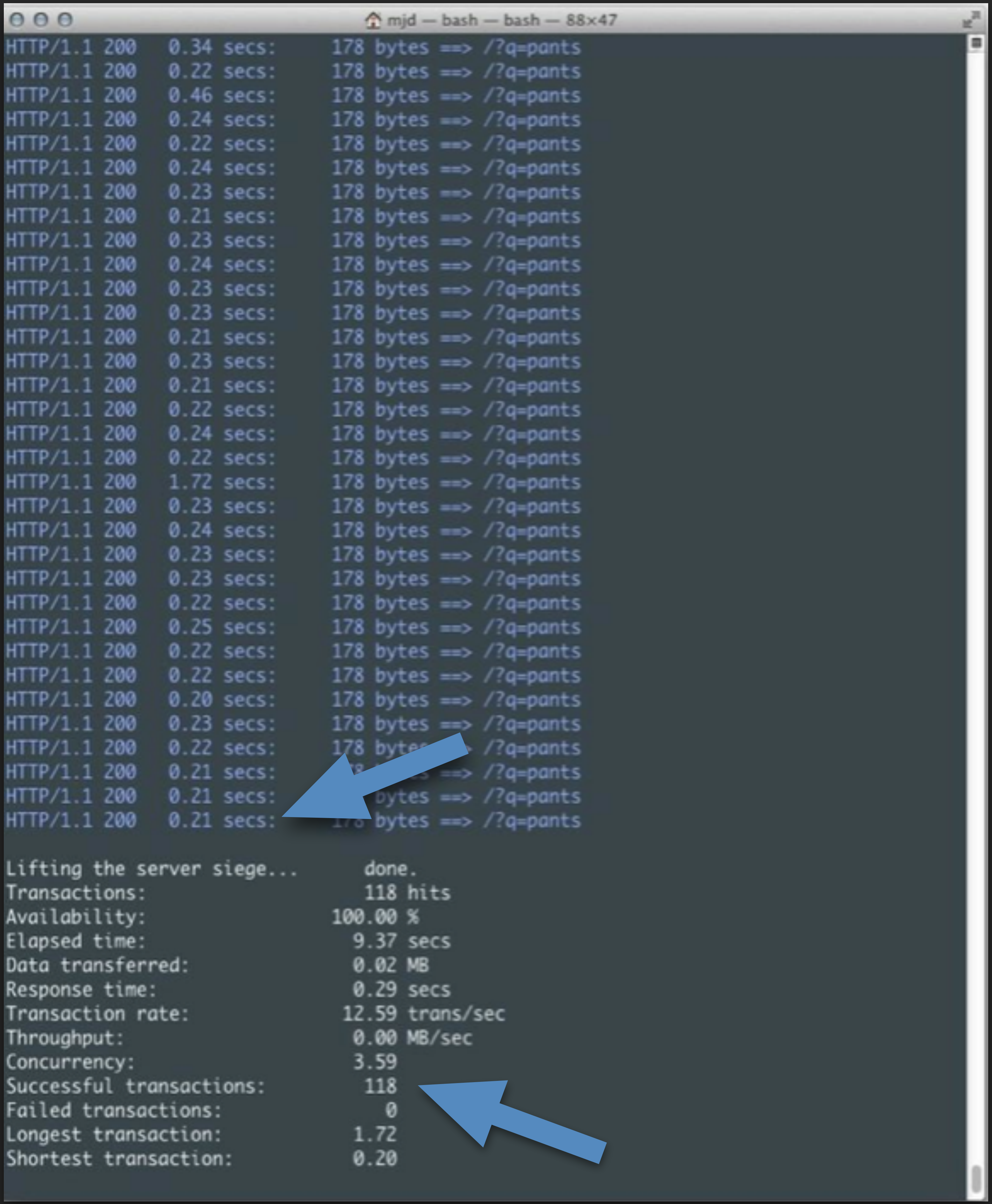


```
Science:~ mjd$ siege http://localhost:8000/?q=pants -c10 -t10s
** SIEGE 2.70
** Preparing 10 concurrent users for battle.
The server is now under siege...
HTTP/1.1 200   0.24 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.48 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.80 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   1.07 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.50 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.69 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.92 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   1.14 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   1.37 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   1.61 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   1.84 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   2.01 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   2.06 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   2.30 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   2.33 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   2.33 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   2.19 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   2.25 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   1.92 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   1.99 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   1.92 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   2.15 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   3.28 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   3.27 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   3.25 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   3.18 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   3.30 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   3.63 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   3.77 secs:     178 bytes ==> /?q=pants

Lifting the server siege...      done.
Transactions:                      29 hits
Availability:                  100.00 %
Elapsed time:                    9.07 secs
Data transferred:                0.00 MB
Response time:                   1.99 secs
Transaction rate:                3.20 trans/sec
Throughput:                      0.00 MB/sec
Concurrency:                     6.37
Successful transactions:           29
Failed transactions:                0
Longest transaction:             3.77
Shortest transaction:            0.24
```

```python
class IndexHandler(tornado.web.RequestHandler):
        @tornado.web.asynchronous
        @tornado.gen.engine
        def get(self):
                query = self.get_argument('q')
                client = tornado.httpclient.AsyncHTTPClient()
                response = yield tornado.gen.Task(client.fetch,
                                "http://search.twitter.com/search.json?" + \
                                urllib.urlencode(
                                     {"q": query, "result_type": "recent", "rpp": 100}))
                body = json.loads(response.body)
                result_count = len(body['results'])
                now = datetime.datetime.utcnow()
                raw_oldest_tweet_at = body['results'][-1]['created_at']
                oldest_tweet_at = datetime.datetime.strptime(raw_oldest_tweet_at,
                                "%a, %d %b %Y %H:%M:%S +0000")
                seconds_diff = time.mktime(now.timetuple()) - \
                                time.mktime(oldest_tweet_at.timetuple())
                tweets_per_second = float(result_count) / seconds_diff

                self.write("""
<div style="text-align: center">
        <div style="font-size: 72px">%s</div>
        <div style="font-size: 144px">%.02f</div>
        <div style="font-size: 24px">tweets per second</div>
</div>""" % (query, tweets_per_second))
                self.finish()
```



```
HTTP/1.1 200   0.34 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.22 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.46 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.24 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.22 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.24 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.23 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.21 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.23 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.24 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.23 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.23 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.21 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.23 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.21 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.22 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.24 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.22 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   1.72 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.23 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.24 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.23 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.23 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.22 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.25 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.22 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.22 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.20 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.23 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.22 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.21 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.21 secs:     178 bytes ==> /?q=pants
HTTP/1.1 200   0.21 secs:     178 bytes ==> /?q=pants

Lifting the server siege...      done.
Transactions:                    118 hits
Availability:                 100.00 %
Elapsed time:                   9.37 secs
Data transferred:               0.02 MB
Response time:                  0.29 secs
Transaction rate:              12.59 trans/sec
Throughput:                     0.00 MB/sec
Concurrency:                    3.59
Successful transactions:         118
Failed transactions:               0
Longest transaction:            1.72
Shortest transaction:           0.20
```

How can I use this in production?

NGINX

Supervisor

hang on to your processes

heroku

Example Nginx Configuration

```
user                    nginx;
worker_processes        1;
error_log               /var/log/nginx/error.log;
pid                     /var/run/nginx.pid;

events {
    worker_connections  1024;
}

http {
    upstream tornado_pool {
        server localhost:8000;
        server localhost:8001;
        server localhost:8002;
        server localhost:8003;
    }
    server {
        listen       80;
        location / {
            proxy_set_header Host $http_host;
            proxy_redirect false;
            proxy_pass http://tornado_pool;
        }
    }
}
```

Example Supervisord Configuration

```
[unix_http_server]
file=/tmp/supervisor.sock    ; (the path to the socket file)

[supervisord]
logfile=/tmp/supervisord.log ; (main log file;default $CWD/supervisord.log)
logfile_maxbytes=50MB        ; (max main logfile bytes b4 rotation;default 50MB)
logfile_backups=10           ; (num of main logfile rotation backups;default 10)
loglevel=info                ; (log level;default info; others: debug,warn,trace)
pidfile=/tmp/supervisord.pid ; (supervisord pidfile;default supervisord.pid)
nodaemon=false               ; (start in foreground if true;default false)

[program:tornado-app]
command=/var/www/tornado-app/current/venv/bin/python app/main.py --port=8000
directory=/var/www/tornado-app/current
autostart=true
autorestart=true
```

Example Heroku Procfile

```
web: python app/main.py --port=$PORT
# yes, that's really it.
```

Example Motor integration

```python
from tornado import gen

class NewMessageHandler(tornado.web.RequestHandler):
    @tornado.web.asynchronous
    @gen.coroutine
    def post(self):
        """Insert a message."""
        msg = self.get_argument('msg')
        db = self.settings['db']

        # insert() returns a Future. Yield the Future to get the result.
        result = yield db.messages.insert({'msg': msg})

        # Success
        self.redirect('/')
```

In short:

It’s quick to set up,
simple to deploy,
and easy to maintain

and makes for happy developers =)

Links!

http://tornadoweb.org

https://github.com/facebook/tornado

http://shop.oreilly.com/product/0636920021292.do

https://github.com/introduction-to-tornado

Thanks!

https://github.com/mikedory
@mike_dory