Real-time web with Django

Aymeric Augustin - @aymericaugustin

DjangoCong - Belfort - September 28th, 2013

What are we talking about?

Real-time

1. **Systems** responding within deadlines
2. **Simulations** running at wall clock time
3. **Processing** events without perceivable delay

http://en.wikipedia.org/wiki/Real-time_computing

Real-time web

"The real-time web is a set of
technologies and practices that enable users to
receive information as soon as it is published
by its authors, rather than
requiring that they or their software
check a source periodically for updates."

http://en.wikipedia.org/wiki/Real-time_web

Use cases

Chat

Games

VoIP

Notifications

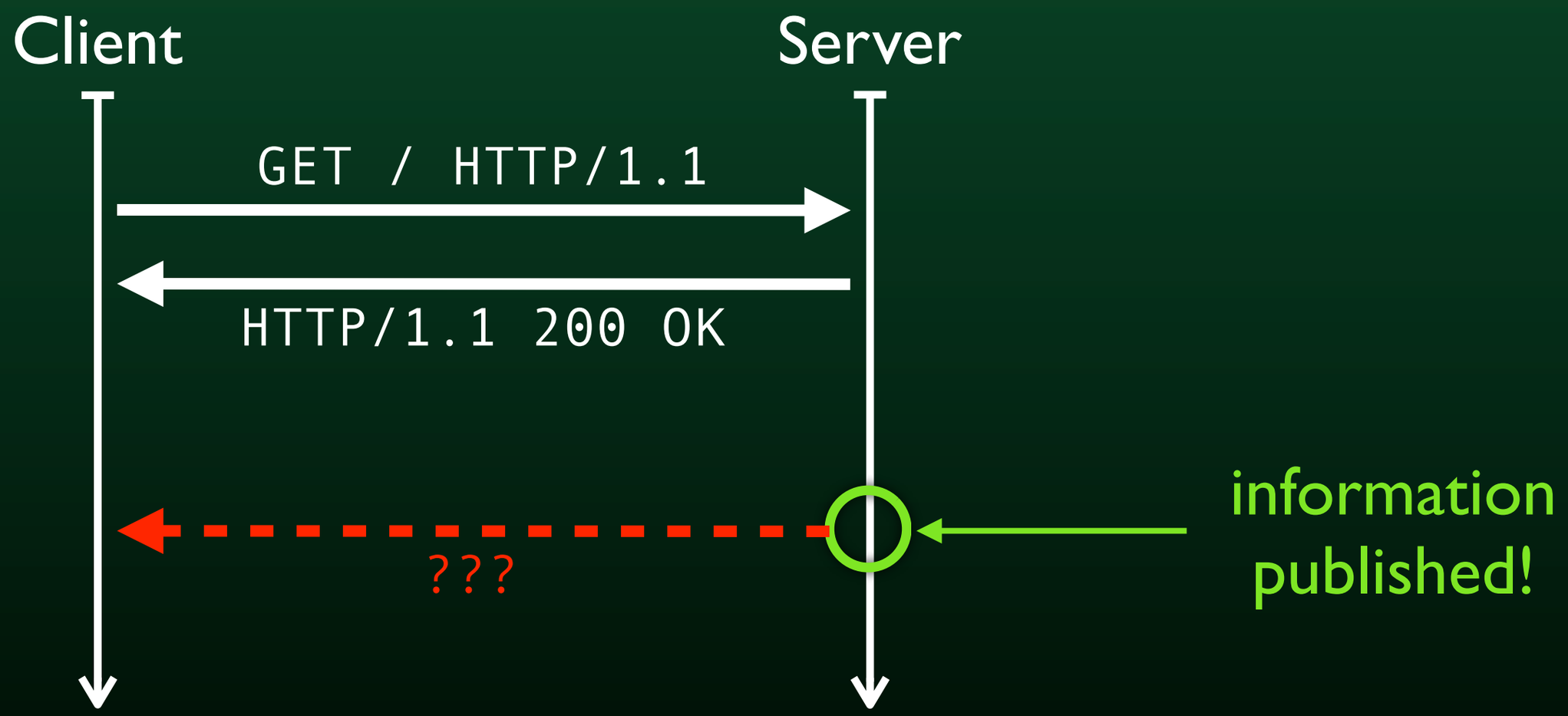
Collaboration

Live data

Social feeds

All these use cases require
servers to push events to clients.

The catch



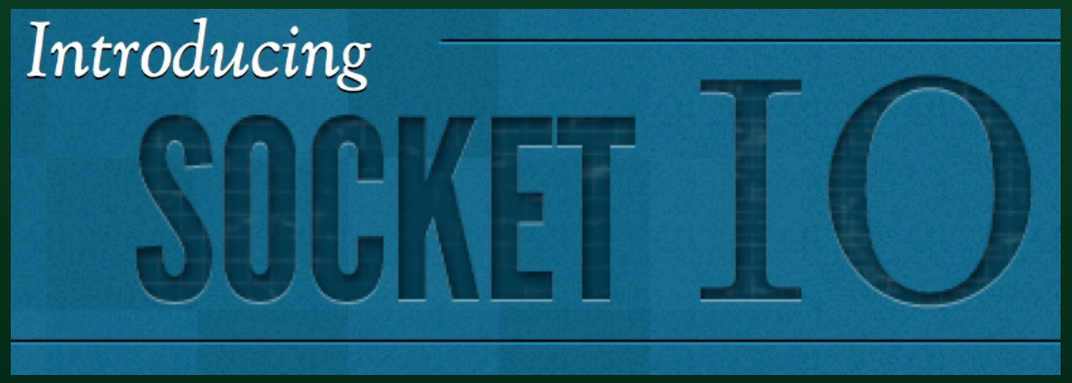
The request-response model doesn't allow servers to push events to clients.

From hacks to protocols

- 2000 : **“Pushlets”**
- 2006 : **“Comet”**
  - HTTP long polling or HTTP streaming
- 2009 : **Server-sent events**
  - HTTP stream of events and JavaScript API
- 2009 : **WebSocket**
  - Bidirectional communication and JavaScript API

Transport frameworks

SockJS



Socket.IO

LightStreamer
(proprietary)

Let's try it!

https://github.com/aaugustin/dcus13rt

An example of WebSocket

Events
Any source

Redis
PUBLISH

Pub/Sub
Redis

Redis
SUBSCRIBE

Real time
Tulip

WebSocket

Web page
JS

Helpers

```
# demo/models.py

import redis

def recv_message():
    client = redis.StrictRedis()
    pubsub = client.pubsub()

    pubsub.subscribe('demo')
    for event in pubsub.listen():
        if event['type'] == 'message':
            message = event['data'].decode('utf-8')
            break
    pubsub.unsubscribe()

    return message
```

Subscriber

```
# demo/handlers.py

import tulip

from demo.models import recv_message


@tulip.coroutine
def simple_endpoint(websocket, uri):
    # Doesn't work! recv_message isn't a coroutine.
    message = yield from recv_message()
    websocket.send(message)
```

Subscriber

```
# demo/handlers.py (1/2)

import tulip
import websockets

from demo.models import recv_message

subscribers = set()

@tulip.coroutine
def endpoint(websocket, uri):
    global subscribers
    subscribers.add(websocket)
    yield from websocket.recv()
    subscribers.remove(websocket)
```

Subscriber

```
# demo/handlers.py (2/2)

def relay_messages():
    while True:
        message = recv_message()
        for websocket in subscribers:
            if websocket.open:
                websocket.send(message)

if __name__ == '__main__':
    websockets.serve(endpoint, 'localhost', 7999)
    loop = tulip.get_event_loop()
    loop.run_in_executor(None, relay_messages)
    loop.run_forever()
```

HTML

```
# demo/templates/demo/websocket.html

<!DOCTYPE html>{% load static %}
<html>
    <head>
        <title>WebSocket demo</title>
    </head>
    <body>
        <ul><!-- messages will be inserted here --></ul>
        <script src="//ajax.googleapis.com/ajax/libs/
jquery/2.0.3/jquery.min.js"></script>
        <script src="{% static 'demo/websocket.js' %}"></
script>
    </body>
</html>
```

JavaScript

```
# demo/static/demo/websocket.js

$(function () {
    var ws = new WebSocket("ws://localhost:7999/");
    ws.onmessage = function (event) {
        $('<li>' + event.data + '</li>')
            .appendTo($('ul'));
    }
});
```

Asynchronous I/O

Execution model

- Based on an event loop
- Handle many socket connections in a single thread
  - epoll (Linux), kqueue (BSD), IOCP (Windows)
- More efficient than one thread per connection
- Suitable for network programming

http://python-notes.boredomandlaziness.org/en/latest/pep_ideas/async_programming.html

Programming model

- Based on explicit cooperative multi-threading
  - Callbacks
  - Coroutines
    - In Python: `yield (from)`
- Suitable for concurrent applications

http://python-notes.boredomandlaziness.org/en/latest/pep_ideas/async_programming.html

The sad truth

"Converting [a synchronous operation] to asynchronous requires modifying every point that calls it to yield control appropriately."

http://python-notes.boredomandlaziness.org/en/latest/pep_ideas/async_programming.html

What about gevent?

It modifies every function that performs I/O
by monkey-patching the standard library
so you don't have to change your own code.

You get the benefits of the execution model for free!
But you lose the benefits of the programming model.

Thank you!

Questions?