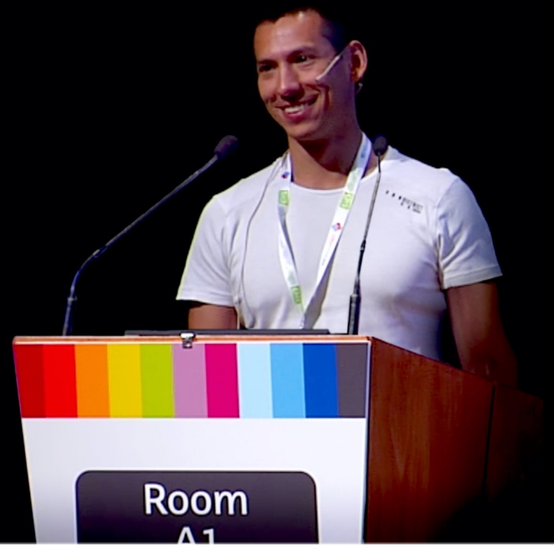
ANTON CACERES

PYTHON SUMMIT

Asynchronous Web Apps made easy with Python

I work here

TECH 5

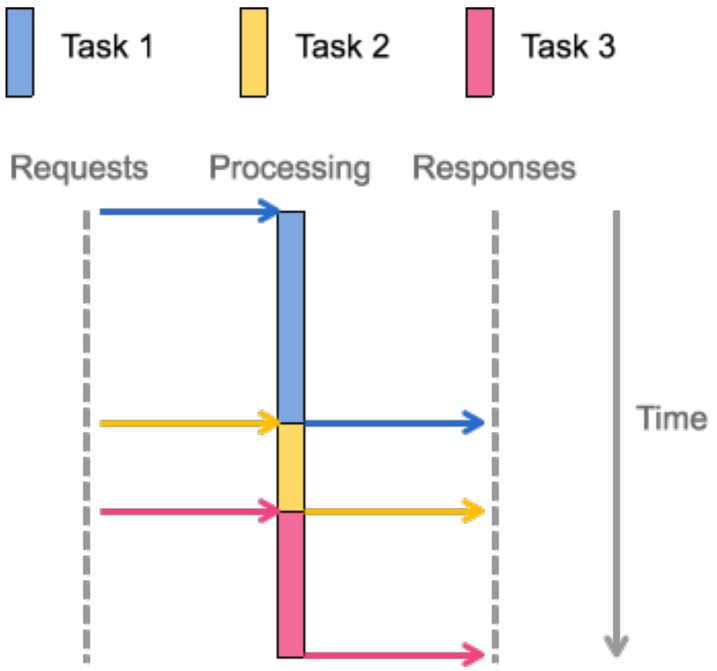


Me

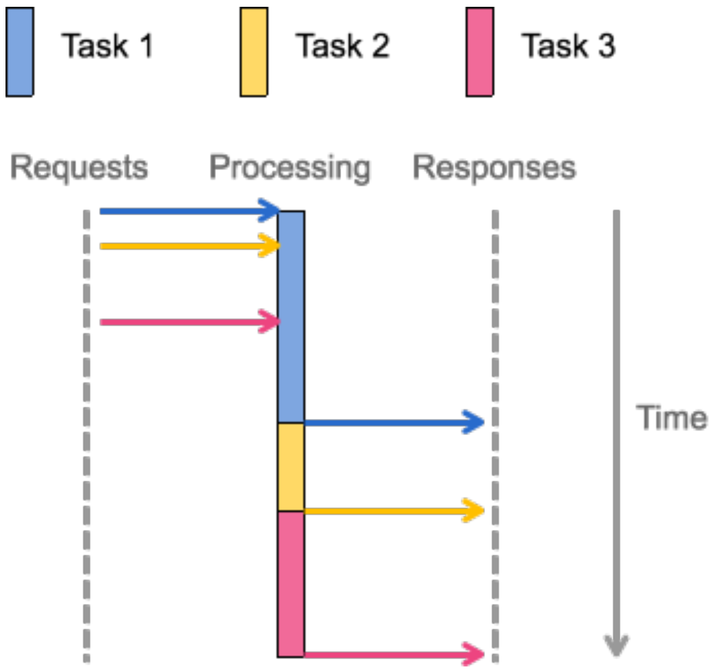
PART 1.
INTRO

REQUEST PROCESSING

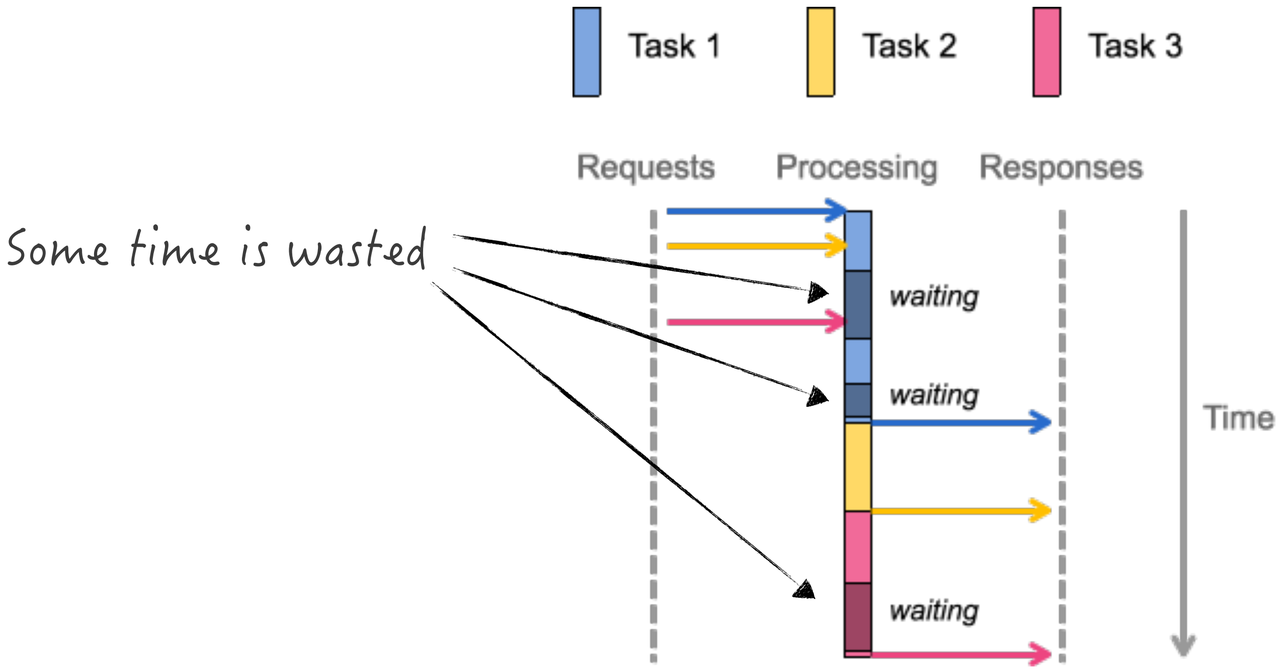
SYNCHRONOUS



SYNCHRONOUS, REALISTIC

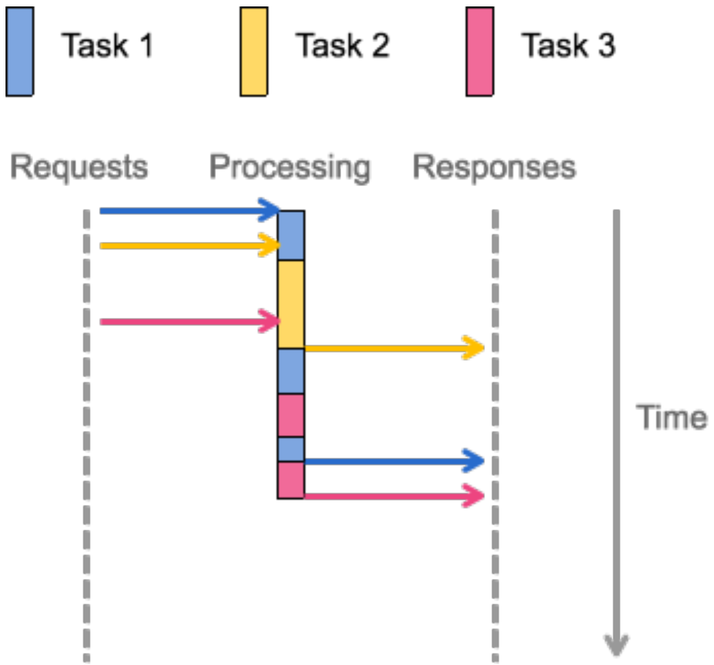


SYNCHRONOUS, REALISTIC

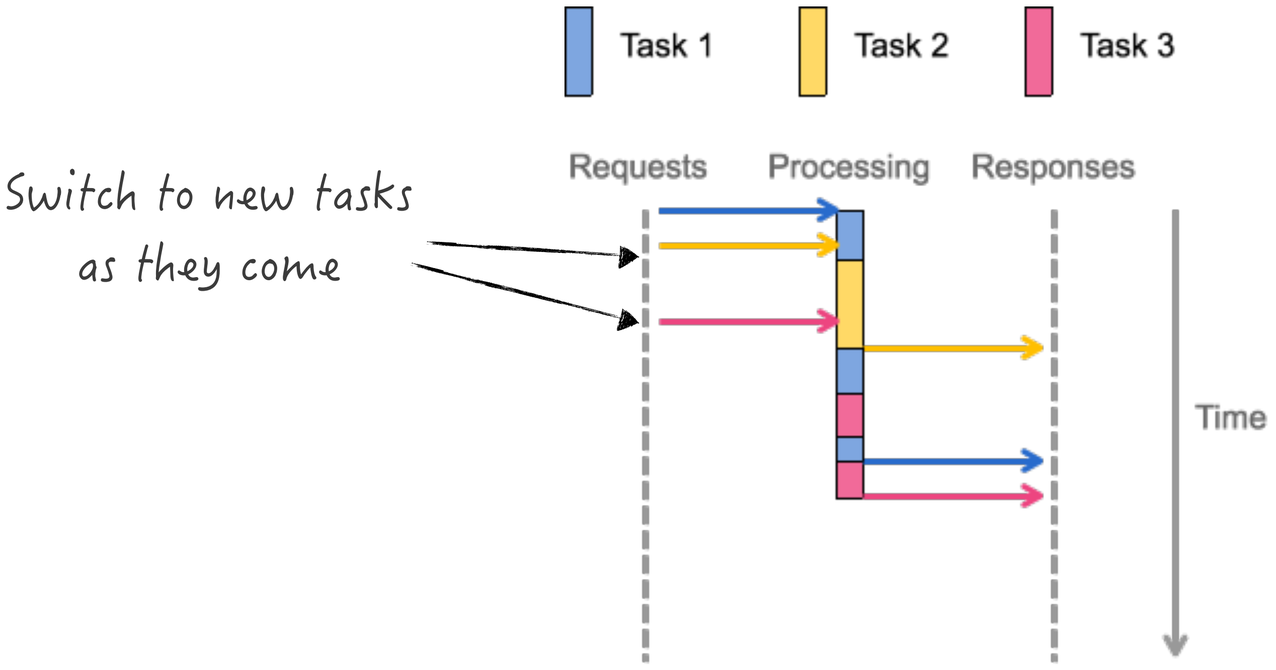


ASYNCHRONOUS

ASYNCHRONOUS

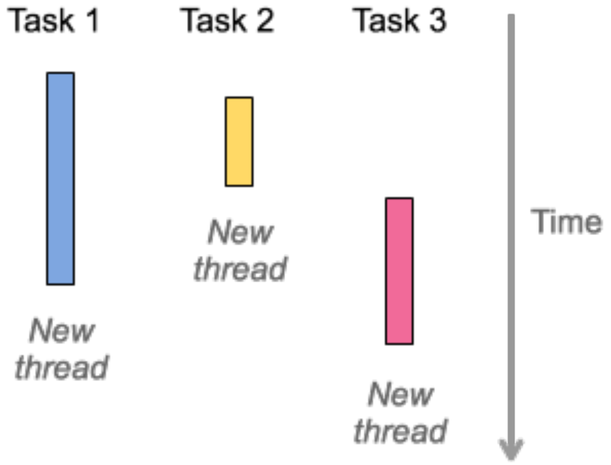


ASYNCHRONOUS



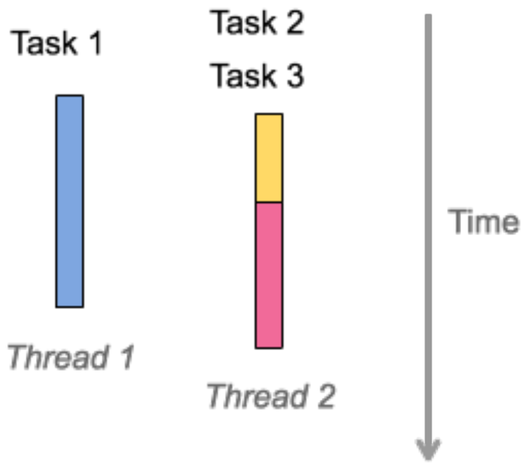
WEB SERVERS

THREADED SERVER



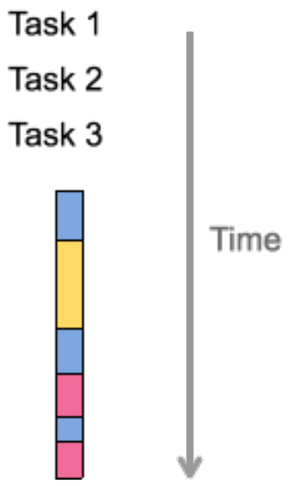
Every request in a new thread

THREADED SERVER WITH POOL



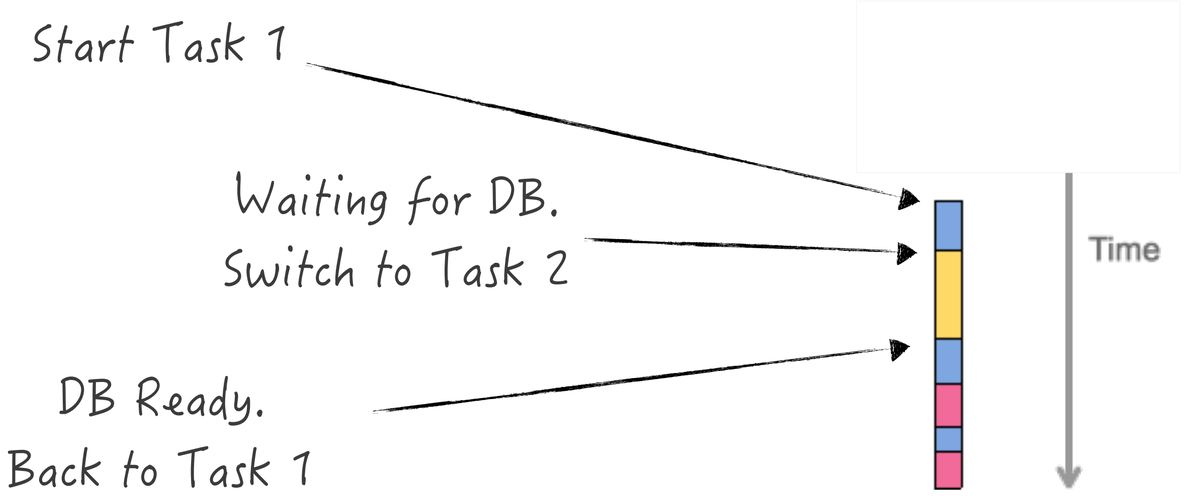
Every request in an own thread.
Thread amount fixed.

THREADED SERVER WITH POOL



One thread with event loop.
Event-driven task switching

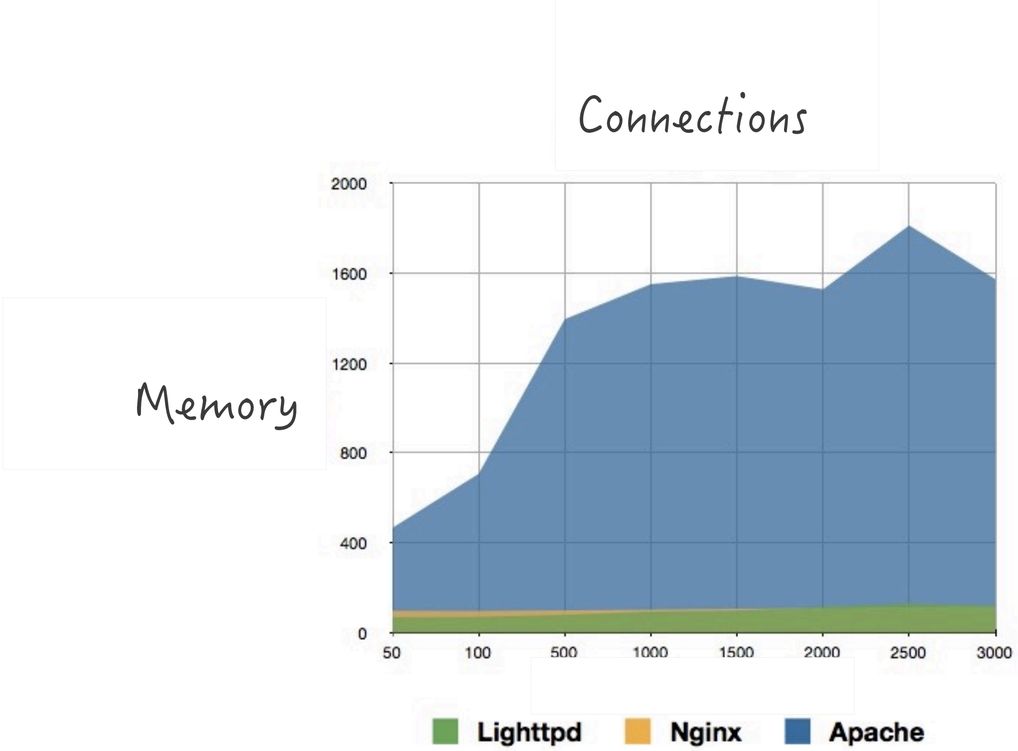
THREADED SERVER WITH POOL



ASYNC IS NOT FASTER.
IT'S SMARTER.

ASYNC IS NOT FASTER.
IT'S SMARTER.
AND LIGHTER.

WEB-SERVER MEMORY USAGE

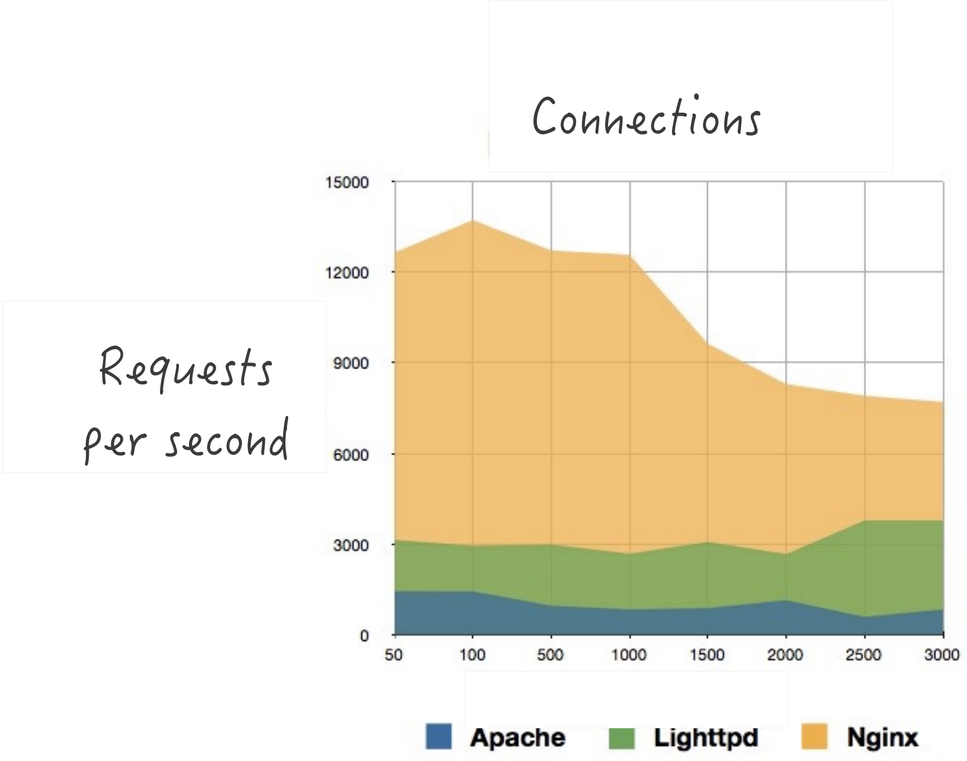


Source: http://wiki.dreamhost.com/Web_Server_Performance_Comparison

Sometimes

ASYNC IS ~~NOT~~ FASTER.
IT'S SMARTER

WEB-SERVER SPEED



Source: http://wiki.dreamhost.com/Web_Server_Performance_Comparison

ASYNC SERVERS ARE NOTHING NEW

ASYNC SERVERS ARE NOTHING NEW

Libevent in C,
Netty in Java,
Akka in Scala,
Node in Javascript,
EventMachine in Ruby

ASYNC SERVERS ARE NOTHING NEW

Libevent in C,
Netty in Java,
Akka in Scala,
Node in Javascript,
EventMachine in Ruby

And in Python... Twisted just turned 17!

WHY NOT SO POPULAR?

WHY NOT SO POPULAR?

Emoji set for async code maintainers

WHY NOT SO POPULAR?

```
before(function (done) {
  server.server(options, function (s, db, providers) {
    //clear db and add a test user - "testuser"
    db.user.remove({}, function () {
      db.notification.remove({}, function () {
        providers.provider1.insertBulk(item1, item2, item3],
          function (err, result) {
          providers.provider2.insert([item1, item2, item3]
            function (err, result) {
            providers.provider3.insert([item1, item2, item3]
              function (err, result) {
              providers.provider4.insert([item1, item2, item3],
                function (err, result) {
                s.listen();
                done();
                })
              });
            });
          });
        });
      });
  });
});
```

IN PYTHON,
IT GETS BETTER

```
async def crawl(url):
    result = await http.fetch("url")
    print(result)
```

PART 2.
ASYNC PYTHON CODE

ASYNC STACK IN PYTHON

ASYNCIO

Your essential toolset:

- futures
- coroutines
- event loop

ASYNCIO

Your essential toolset:

- futures → placeholder object
- coroutines
- event loop

ASYNCIO

Your essential toolset:

- futures → placeholder object → future = async_operation()
result = await future
- coroutines
- event loop

ASYNCIO

Your essential toolset:

- futures → placeholder object → future = async_operation()
result = await future
- coroutines → function with await
- event loop

ASYNCIO

Your essential toolset:

- futures → placeholder object → future = async_operation()
result = await future
- coroutines → function with await → async def function_name():
result = await future
- event loop

ASYNCIO

Your essential toolset:

- futures → placeholder object → future = async_operation()
result = await future
- coroutines → function with await → async def function_name():
result = await future
- event loop → engine for running it

ASYNCIO

Your essential toolset:

- futures → placeholder object → `future = async_operation()`
  `result = await future`
- coroutines → function with await → `async def function_name():`
  `result = await future`
- event loop → engine for running it → `loop = asyncio.get_event_loop()`
  `loop.run_forever()`

COMPARE IT

Sequential

```
def get():
    result = huge_db_query()
    self.write(result)
```

COMPARE IT

Sequential

```
def get():
    result = huge_db_query()
    self.write(result)
```

Async, callbacks

```
def get():
    def on_result(result):
        self.write(result)
    huge_db_query(callback=on_result)
```

COMPARE IT

Sequential

```
def get():
    result = huge_db_query()
    self.write(result)
```

Async, callbacks

```
def get():
    def on_result(result):
        self.write(result)
    huge_db_query(callback=on_result)
```

Async, coroutine

```
async def get():
    result = await huge_db_query()
    self.write(result)
```

CODE TIME

SET UP THE PROJECT

- git clone https://github.com/ma3str0/async-workshop
- cd async-workshop
- python3 -m venv venv
- pip install -r requirements.txt

SET UP THE PROJECT

- mkdir my-async-workshop
- cd my-async-workshop
- python3 -m venv venv
- source venv/bin/activate
- pip install asyncio aiohttp tornado

ASYNCIO

- Weather service is located on:
- https://caceres.me/workshop/weather/today
- Make a webapp to query that api and display results
- Use aiohttp to request async
- test with: `curl 127.0.0.1:8080/tomorrow -w '\n%{time_total}\n'`

ASYNCIO, LEVEL UP

- We also want the weather for tomorrow
- Query https://caceres.me/...../tomorrow
- Make sure not to wait for two requests sequentially
- test with: `curl 127.0.0.1:8080/tomorrow -w '\n%{time_total}\n'`

TORNADO

- Build a network discovery service.
- Keep {service: url} mapping for your services
- Proxy requests to your Weather Service

TORNADO

- Add an authentication service: caceres.me/workshop/auth with params *key=supersecret*
- Check user permissions before querying weather
- *Bonus: use your Discovery Service to query Auth Service*

WEB SOCKETS

- Websockets are super lightweight by design
- Much more suitable for microservices and real-time data
- Copy the chat app from tornado demo files

NOISE APP

- Let’s measure how noisy are our rooms using JS
- Track this data realtime with web-sockets
- Manage data streams with Tornado

THANK YOU