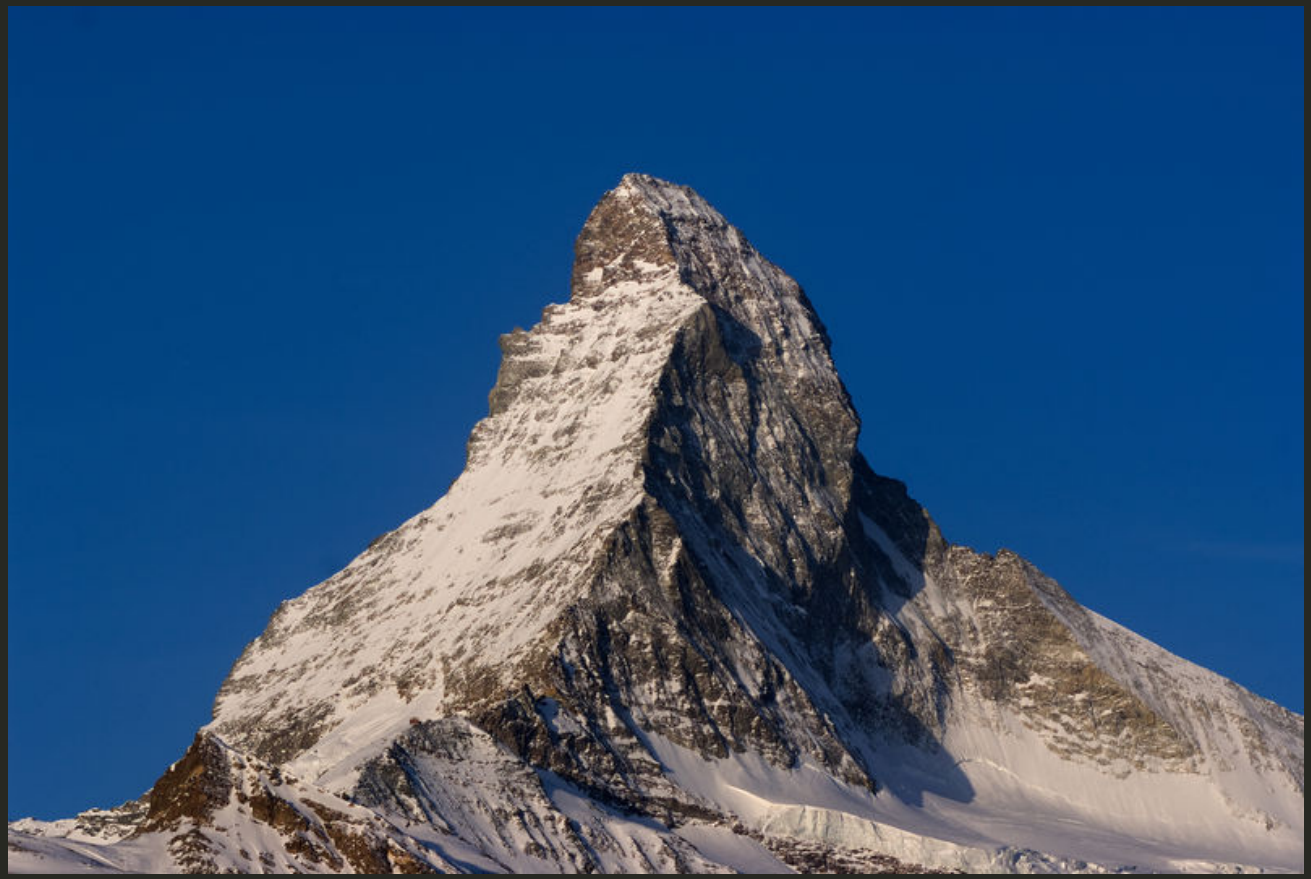
Why You Might Want To Go Async

PyCon Colombia 2018 - Jonas Obrist - HDE Inc

Hi, I'm Jonas

Hi, I'm Jonas



Hi, I'm Jonas

Hi, I'm Jonas

Quick Links

@ojiidotch

github.com/ojii

hde.co.jp/en/

Why You Might Want To Go Async

~~Why You Might Want To Go Async~~

~~Why You Might Want To Go Async~~

What Do I Mean When I Say Async

Async

Asynchronous

Asynchronous IO

Asynchronous Networking

Asynchronous Networking

Opposite of Synchronous

Examples of Synchronous (Python) Frameworks/Libraries

Examples of Synchronous (Python) Frameworks/Libraries

- `requests`

Examples of Synchronous (Python) Frameworks/Libraries

- `requests`
- `Django`

Examples of Synchronous (Python) Frameworks/Libraries

- `requests`
- `Django`
- `Flask`

Examples of Synchronous (Python) Frameworks/Libraries

- `requests`
- `Django`
- `Flask`
- `psycopg2` (PostgreSQL Library)

Examples of Synchronous (Python) Frameworks/Libraries

- `requests`
- `Django`
- `Flask`
- `psycopg2` (PostgreSQL Library)
- `boto3` (AWS Library)

Examples of Synchronous (Python) Frameworks/Libraries

- `requests`
- `Django`
- `Flask`
- `psycopg2` (PostgreSQL Library)
- `boto3` (AWS Library)
- `<most other libraries here>`

Examples of Asynchronous (Python) Frameworks/Libraries

Examples of Asynchronous (Python) Frameworks/Libraries

- `twisted`

Examples of Asynchronous (Python) Frameworks/Libraries

- `twisted`
- `tornado`

Examples of Asynchronous (Python) Frameworks/Libraries

- `twisted`
- `tornado`
- `asyncio` (stdlib)

Examples of Asynchronous (Python) Frameworks/Libraries

- `twisted`
- `tornado`
- `asyncio` (stdlib)
- `sanic` (“Flask”)

Examples of Asynchronous (Python) Frameworks/Libraries

- `twisted`
- `tornado`
- `asyncio` (stdlib)
- `sanic` (“Flask”)
- `curio`

Examples of Asynchronous (Python) Frameworks/Libraries

- `twisted`
- `tornado`
- `asyncio` (stdlib)
- `sanic` (“Flask”)
- `curio`
- `aiopg` (PostgreSQL Library)

Examples of Asynchronous (Python) Frameworks/Libraries

- `twisted`
- `tornado`
- `asyncio` (stdlib)
- `sanic` (“Flask”)
- `curio`
- `aiopg` (PostgreSQL Library)
- `aiobotocore` (AWS Library)

Examples of Asynchronous (Python) Frameworks/Libraries

- `twisted`
- `tornado`
- `asyncio` (stdlib)
- `sanic` (“Flask”)
- `curio`
- `aiopg` (PostgreSQL Library)
- `aiobotocore` (AWS Library)
- `arsenic` & `aapns` (Libraries I wrote and am shamelessly plugging here)

Asynchronous IO Core Concepts

Asynchronous IO Core Concepts

- Event Loop

Asynchronous IO Core Concepts

- Event Loop
- Coroutines (Previously: Callback Hell)

Asynchronous IO Core Concepts

- Event Loop
- Coroutines (Previously: Callback Hell)
    - A Function which allows cooperative multi-tasking (Co-operative Sub-Routine)

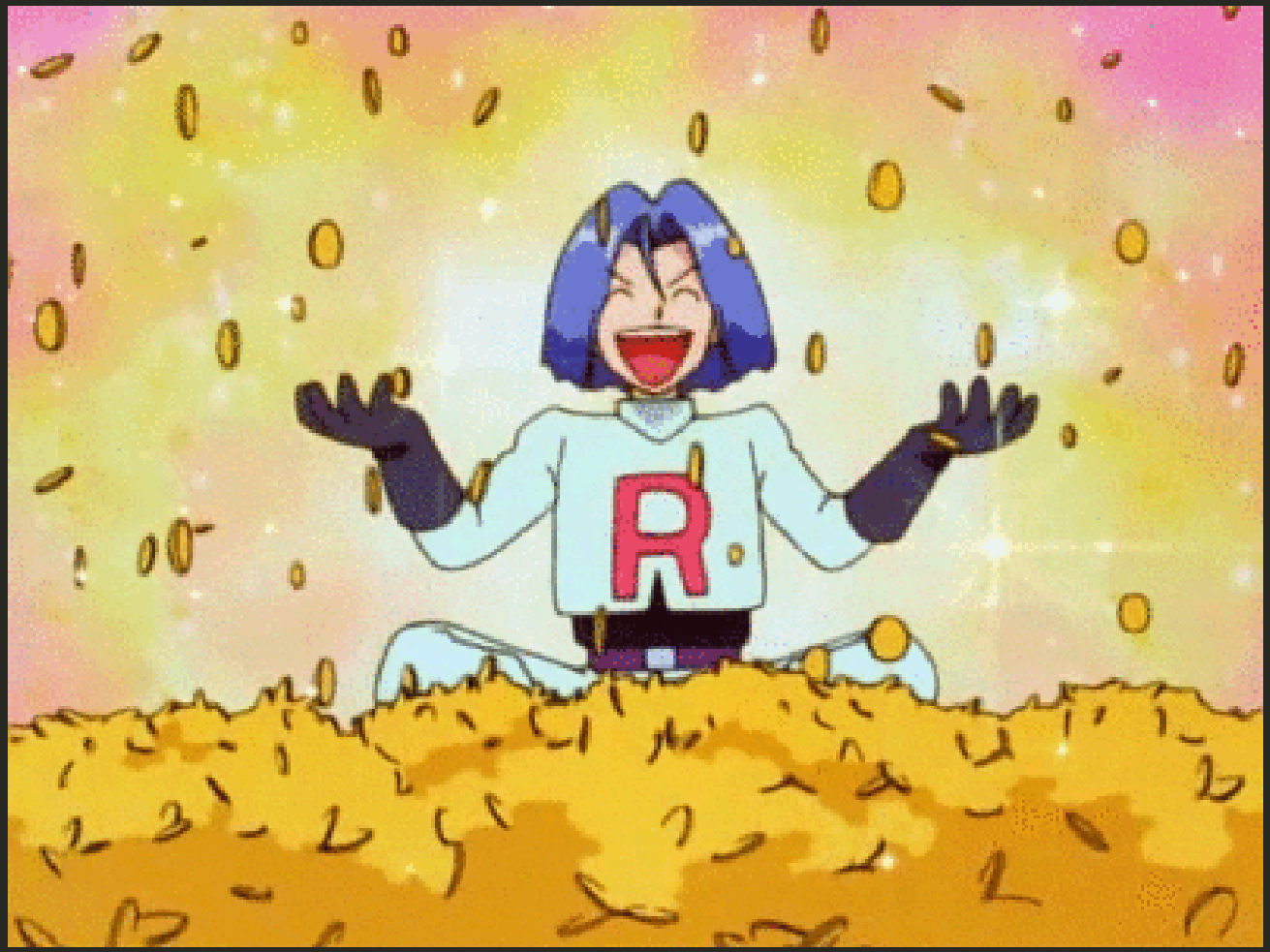
Asynchronous IO Core Concepts

- Event Loop
- Coroutines (Previously: Callback Hell)
    - A Function which allows cooperative multi-tasking (Co-operative Sub-Routine)
- IO is non-blocking

Asynchronous IO Core Concepts

- Event Loop
- Coroutines (Previously: Callback Hell)
  - A Function which allows cooperative multi-tasking (Co-operative Sub-Routine)
- IO is non-blocking
- “Logic” might be blocking

Why You Might Want To Go Async


R

Why You Might Want To Go Async

~~Why You Might Want To Go Async~~

~~Why You Might Want To Go Async~~

Why Does Going Async Save Money

Python Is “Slow”

Python Is “Slow”

- Synchronous web server can only handle **one** request per thread/process.

Python Is “Slow”

- Synchronous web server can only handle **one** request per thread/process.
- Scaling is usually done with more threads/processes/servers

Python Is “Slow”

- Synchronous web server can only handle **one** request per thread/process.
- Scaling is usually done with more threads/processes/servers
- Lots of resources (CPU) is wasted

Python Is “Slow”

- Synchronous web server can only handle **one** request per thread/process.
- Scaling is usually done with more threads/processes/servers
- Lots of resources (CPU) is wasted
    - Most of the time we're waiting for something to respond with data

Python Is “Slow”

- Synchronous web server can only handle **one** request per thread/process.
- Scaling is usually done with more threads/processes/servers
- Lots of resources (CPU) is wasted
    - Most of the time we're waiting for something to respond with data
    - Surprisingly little time is spent in our actual Python code

Python Is “Slow”

~~Python Is “Slow”~~

~~Python Is "Slow"~~

Python Is "Inefficient"

~~Python Is “Slow”~~

~~Python Is “Inefficient”~~

~~Python Is "Slow"~~

~~Python Is "Inefficient"~~

Synchronous Python Is Inefficient

Asynchronous Python To The Rescue!

Example: A Web Server

Example: A Web Server

- Handle a `POST` request

Example: A Web Server

- Handle a `POST` request
- Store the `POST` data in a database

Example: A Web Server

- Handle a `POST` request
- Store the `POST` data in a database
- Return an identifier of the database object as json

Example: Webserver (Synchronous)

Example: Webserver (Synchronous)

```
def handler(request: Request) -> Response:
    data = request.post_data
    pk = database.save_item(data)
    return Response(
        status=200,
        body=json.dumps({'pk': pk}) ,
        content_type='application/json')
    )
```

Example: Webserver (Synchronous)

```
def handler(request: Request) -> Response: # First, Second
    data = request.post_data
    pk = database.save_item(data)
    return Response(
        status=200,
        body=json.dumps({'pk': pk}) ,
        content_type='application/json')
    )
```

Example: Webserver (Synchronous)

```
def handler(request: Request) -> Response:  # Second
    data = request.post_data                # First
    pk = database.save_item(data)
    return Response(
        status=200,
        body=json.dumps({'pk': pk}) ,
        content_type='application/json')
    )
```

Example: Webserver (Synchronous)

```
def handler(request: Request) -> Response:  # Second
    data = request.post_data
    pk = database.save_item(data)           # First
    return Response(
        status=200,
        body=json.dumps({'pk': pk}) ,
        content_type='application/json')
    )
```

Example: Webserver (Synchronous)

```
def handler(request: Request) -> Response:  # Second
    data = request.post_data
    pk = database.save_item(data)
    return Response(                        # First
        status=200,
        body=json.dumps({'pk': pk}) ,
        content_type='application/json')
    )
```

Example: Webserver (Synchronous)

```
def handler(request: Request) -> Response:
    data = request.post_data                    # Second
    pk = database.save_item(data)
    return Response(
        status=200,
        body=json.dumps({'pk': pk}) ,
        content_type='application/json')
    )
    # First Done
```

Example: Webserver (Synchronous)

```
def handler(request: Request) -> Response:
    data = request.post_data
    pk = database.save_item(data)           # Second
    return Response(
        status=200,
        body=json.dumps({'pk': pk}) ,
        content_type='application/json')
    )
    # First Done
```

Example: Webserver (Synchronous)

```
def handler(request: Request) -> Response:
    data = request.post_data
    pk = database.save_item(data)
    return Response(                          # Second
        status=200,
        body=json.dumps({'pk': pk}) ,
        content_type='application/json')
    )
    # First Done
```

Example: Webserver (Synchronous)

```
def handler(request: Request) -> Response:
    data = request.post_data
    pk = database.save_item(data)
    return Response(
        status=200,
        body=json.dumps({'pk': pk}) ,
        content_type='application/json')
    )
    # First Done
    # Second Done
```

Example: Webserver (Asynchronous)

Example: Webserver (Asynchronous)

```
async def handler(request: Request) -> None:
    data = await request.get_post_data()
    pk = await database.save_item(data)
    response = request.response(
        status=200,
        content_type='application/json',
    )
    response.write(json.dumps({'pk': pk}))
    response.finish()
```

Example: Webserver (Asynchronous)

```
async def handler(request: Request) -> None:
    data = await request.get_post_data()
    pk = await database.save_item(data)
    response = request.response(
        status=200,
        content_type='application/json',
    )
    response.write(json.dumps({'pk': pk}))
    response.finish()
```

Example: Webserver (Asynchronous)

```
async def handler(request: Request) -> None:
    data = await request.get_post_data()
    pk = await database.save_item(data)
    response = request.response(
        status=200,
        content_type='application/json',
    )
    response.write(json.dumps({'pk': pk}))
    response.finish()
```

Example: Webserver (Asynchronous)

```
async def handler(request: Request) -> None:
    data = await request.get_post_data()
    pk = await database.save_item(data)
    response = request.response(
        status=200,
        content_type='application/json',
    )
    response.write(json.dumps({'pk': pk}))
    response.finish()
```

Example: Webserver (Asynchronous)

```
async def handler(request: Request) -> None:
    data = await request.get_post_data()
    pk = await database.save_item(data)
    response = request.response(
        status=200,
        content_type='application/json',
    )
    response.write(json.dumps({'pk': pk}))
    response.finish()
```

Example: Webserver (Asynchronous)

```
async def handler(request: Request) -> None:  # First, Second
    data = await request.get_post_data()
    pk = await database.save_item(data)
    response = request.response(
        status=200,
        content_type='application/json',
    )
    response.write(json.dumps({'pk': pk}))
    response.finish()
```

Example: Webserver (Asynchronous)

```
async def handler(request: Request) -> None:  # Second
    data = await request.get_post_data()      # First
    pk = await database.save_item(data)
    response = request.response(
        status=200,
        content_type='application/json',
    )
    response.write(json.dumps({'pk': pk}))
    response.finish()
```

Example: Webserver (Asynchronous)

```
async def handler(request: Request) -> None:
    data = await request.get_post_data()      # First, Second
    pk = await database.save_item(data)
    response = request.response(
        status=200,
        content_type='application/json',
    )
    response.write(json.dumps({'pk': pk}))
    response.finish()
```

Example: Webserver (Asynchronous)

```
async def handler(request: Request) -> None:
    data = await request.get_post_data()      # First
    pk = await database.save_item(data)       # Second
    response = request.response(
        status=200,
        content_type='application/json',
    )
    response.write(json.dumps({'pk': pk}))
    response.finish()
```

Example: Webserver (Asynchronous)

```
async def handler(request: Request) -> None:
    data = await request.get_post_data()
    pk = await database.save_item(data)        # First
    response = request.response(               # Second
        status=200,
        content_type='application/json',
    )
    response.write(json.dumps({'pk': pk}))
    response.finish()
```

Example: Webserver (Asynchronous)

```
async def handler(request: Request) -> None:
    data = await request.get_post_data()
    pk = await database.save_item(data)
    response = request.response(                # First
        status=200,
        content_type='application/json',
    )
    response.write(json.dumps({'pk': pk}))
    response.finish()
    # Second Done
```

Example: Webserver (Asynchronous)

```
async def handler(request: Request) -> None:
    data = await request.get_post_data()
    pk = await database.save_item(data)
    response = request.response(
        status=200,
        content_type='application/json',
    )
    response.write(json.dumps({'pk': pk}))
    response.finish()
    # Second Done
    # First Done
```

Why Does That Work?

Why Does That Work?

- Still *one* process/thread

Why Does That Work?

- Still *one* process/thread
- While one request waits for data, handle other requests

An Analogy: A Restaurant

Story Time

Story Time

- Converted the most used APIs ouf our app to use async IO

Story Time

- Converted the most used APIs ouf our app to use async IO
    - Heavy use of threads for legacy code
    - Tornado event loop
    - Still Python 2.7

Story Time

- Converted the most used APIs ouf our app to use async IO
    - Heavy use of threads for legacy code
    - Tornado event loop
    - Still Python 2.7
- Reduced number of servers by **25%**

Story Time

- Converted the most used APIs ouf our app to use async IO
    - Heavy use of threads for legacy code
    - Tornado event loop
    - Still Python 2.7
- Reduced number of servers by **25%**
    - Server bill reduced by thousands of dollars per month
    - Business logic stayed the same
    - Only moved IO code to non-blocking or threads

Why You Might Want To Go Async

~~Why You Might Want To Go Async~~

~~Why You Might Want To Go Async~~

Why You Might Want To Go Async **Now**

Why You Might Want To Go Async Now

Why You Might Want To Go Async Now

- Python 3.6
  - `async`
  - `await`
  - `async` iterators
  - `async` generators
  - `async` context managers

Why You Might Want To Go Async Now

- Python 3.6
  - `async`
  - `await`
  - `async` iterators
  - `async` generators
  - `async` context managers
- Async Revolution
  - `uvloop`
  - `aiopg`
  - `sanic`
  - `aio*`

Story Time (cont)

Story Time (cont)

- Upgraded App To Python 3.6

Story Time (cont)

- Upgraded App To Python 3.6
    - Replaced Tornado Main Loop with AsyncIO Main Loop

Story Time (cont)

- Upgraded App To Python 3.6
  - Replaced Tornado Main Loop with AsyncIO Main Loop
  - Removed ~20 dependencies

Story Time (cont)

- Upgraded App To Python 3.6
  - Replaced Tornado Main Loop with AsyncIO Main Loop
  - Removed ~20 dependencies
  - Rewrote 2 dependencies (Google API, APNS)

Story Time (cont)

- Upgraded App To Python 3.6
  - Replaced Tornado Main Loop with AsyncIO Main Loop
  - Removed ~20 dependencies
  - Rewrote 2 dependencies (Google API, APNS)
  - Used *some* asyncio libraries

Story Time (cont)

- Upgraded App To Python 3.6
  - Replaced Tornado Main Loop with AsyncIO Main Loop
  - Removed ~20 dependencies
  - Rewrote 2 dependencies (Google API, APNS)
  - Used *some* asyncio libraries
- Result

Story Time (cont)

- Upgraded App To Python 3.6
  - Replaced Tornado Main Loop with AsyncIO Main Loop
  - Removed ~20 dependencies
  - Rewrote 2 dependencies (Google API, APNS)
  - Used *some* asyncio libraries
- Result
  - **4x** speedup in request handling

Story Time (cont)

- Upgraded App To Python 3.6
  - Replaced Tornado Main Loop with AsyncIO Main Loop
  - Removed ~20 dependencies
  - Rewrote 2 dependencies (Google API, APNS)
  - Used *some* asyncio libraries
- Result
  - **4x** speedup in request handling
  - CPU usage much lower, much more stable

Story Time (cont)

- Upgraded App To Python 3.6
  - Replaced Tornado Main Loop with AsyncIO Main Loop
  - Removed ~20 dependencies
  - Rewrote 2 dependencies (Google API, APNS)
  - Used *some* asyncio libraries
- Result
  - **4x** speedup in request handling
  - CPU usage much lower, much more stable
  - Reduced number of servers by **another 30+%**

To Recap

To Recap

- Use Async To Save Money

To Recap

- Use Async To Save Money
- Use Async **Now** Because It's Ready

To Recap

- Use Async To Save Money
- Use Async **Now** Because It's Ready
- Use Python 3.6

Questions?

Thank You!

@ojiidotch

github.com/ojii