The GIL is irrelevant (sometimes)

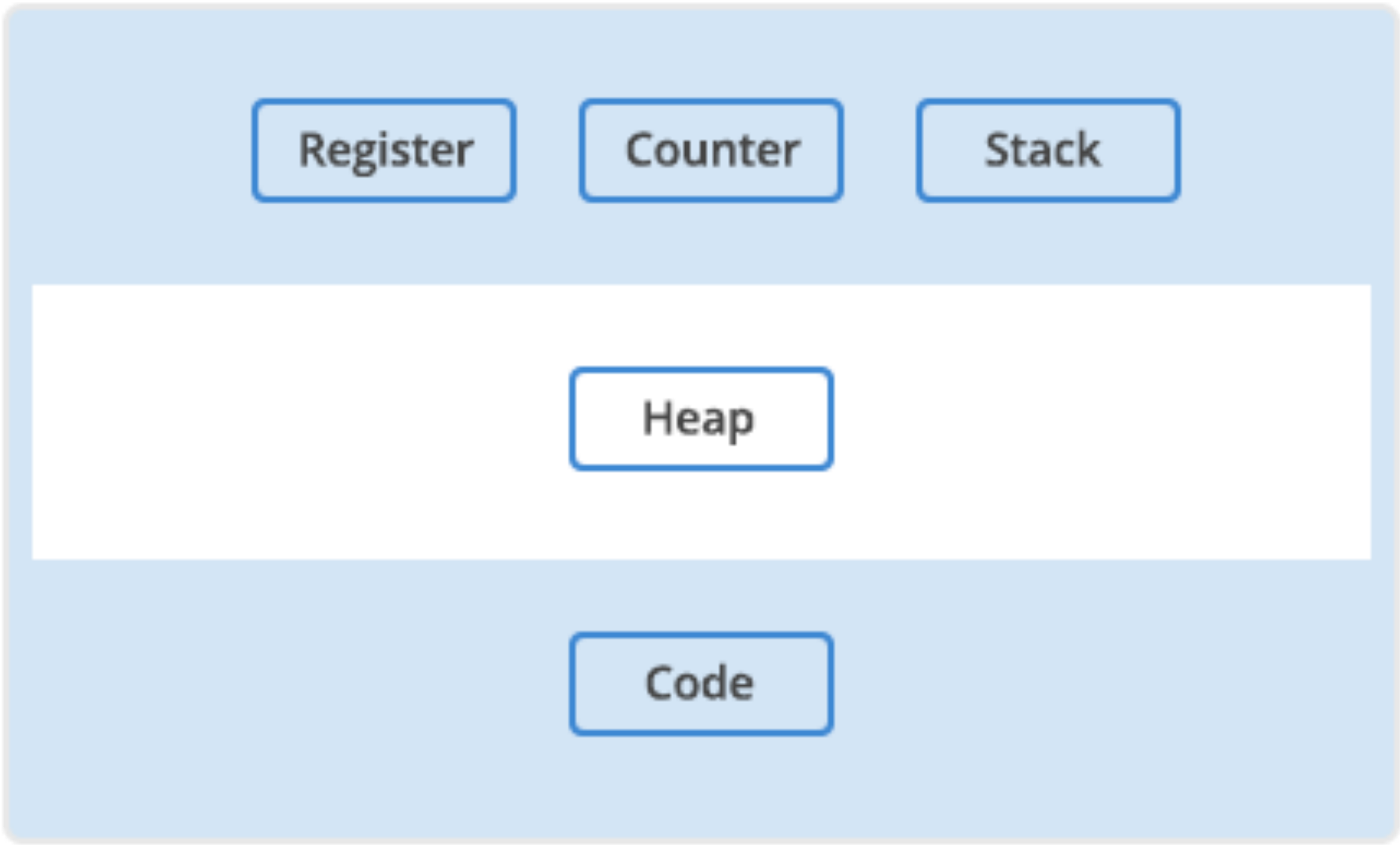
Asynchronrous IO

Ali Asad Lotia - Equal Experts
alotia+pycon@equalexperts.com Twitter: @aalotia

Overview

- Background
- Why async?
- Async in the wild
- Async in python
- asyncio
- Testing

Performance - A brief history

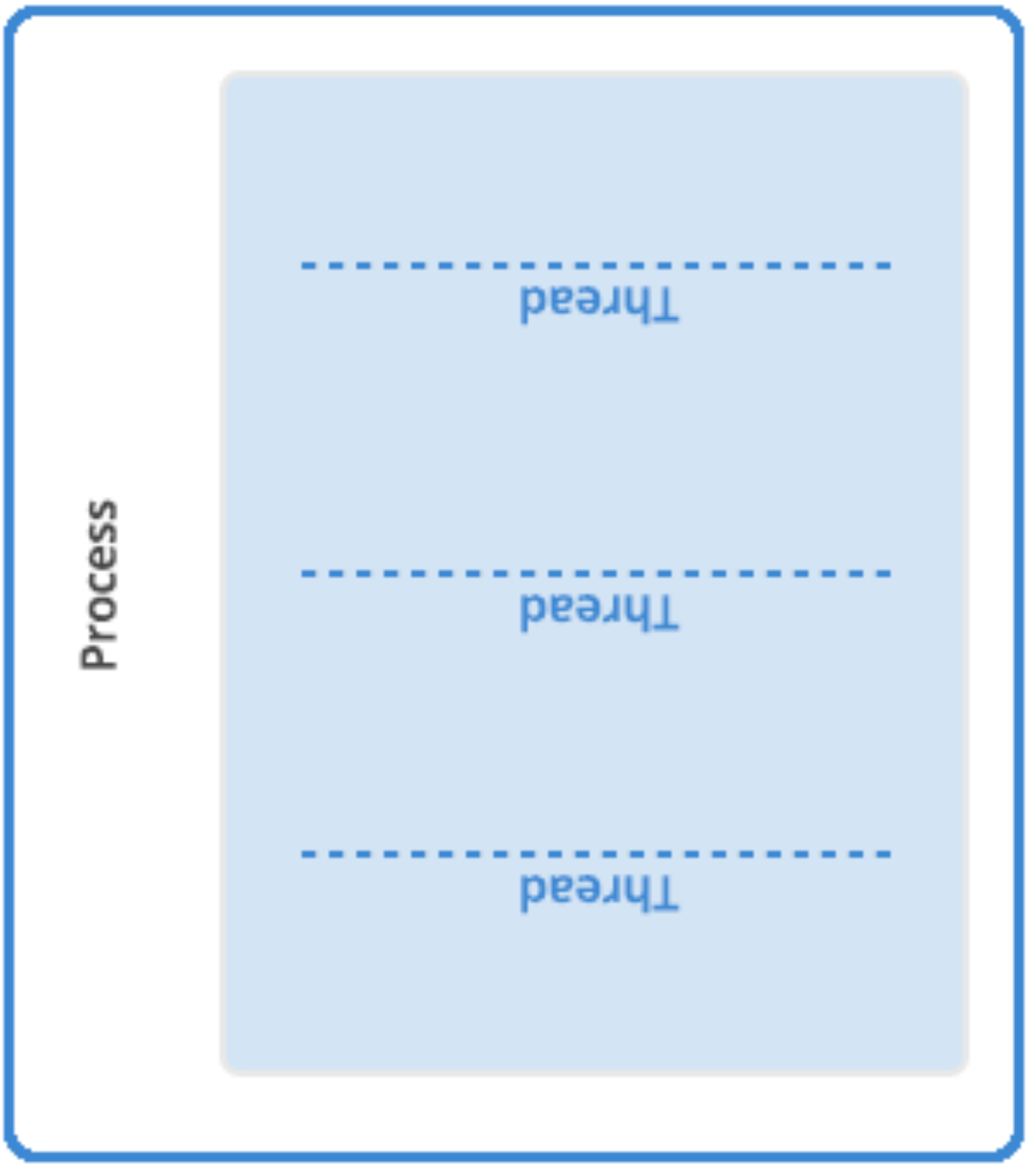

Register
Counter
Stack
Heap
Code

“Instance of a program in execution”

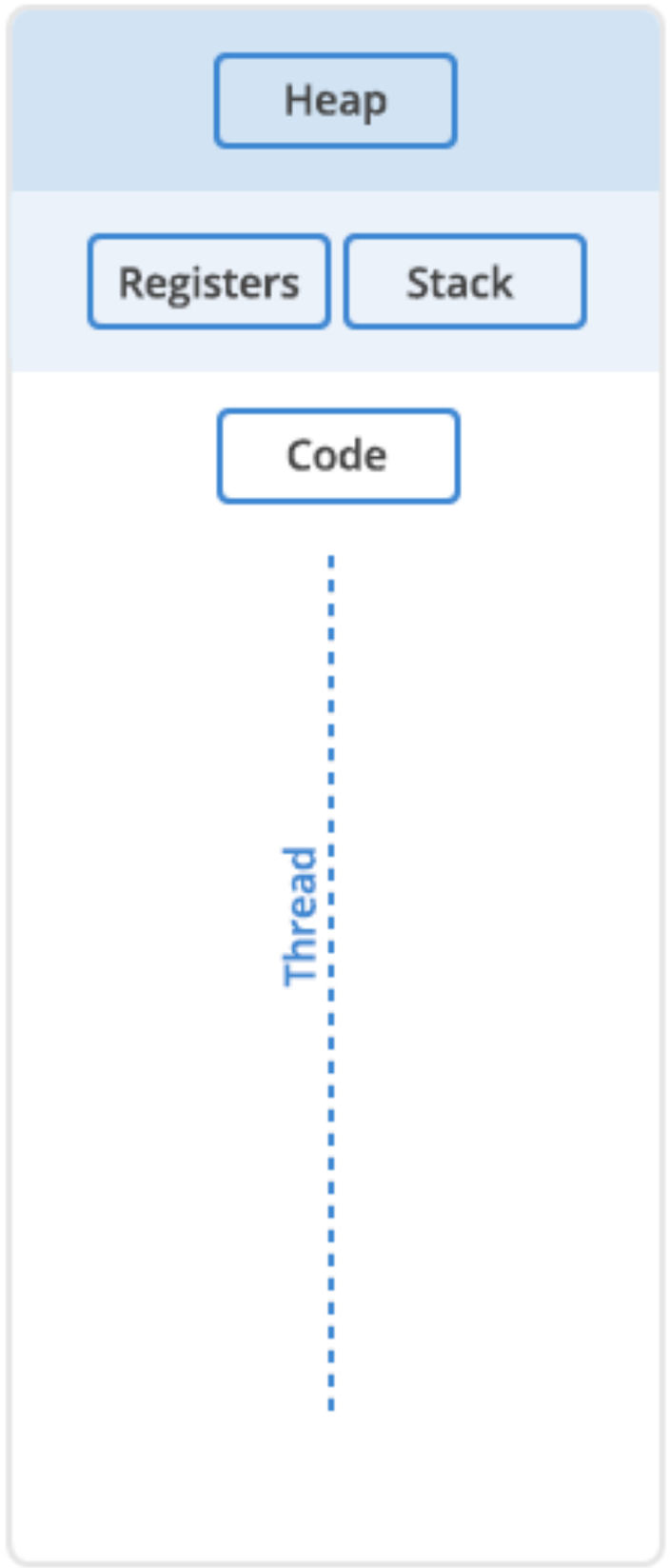
– https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/3_Processes.html

the unit of execution within a process

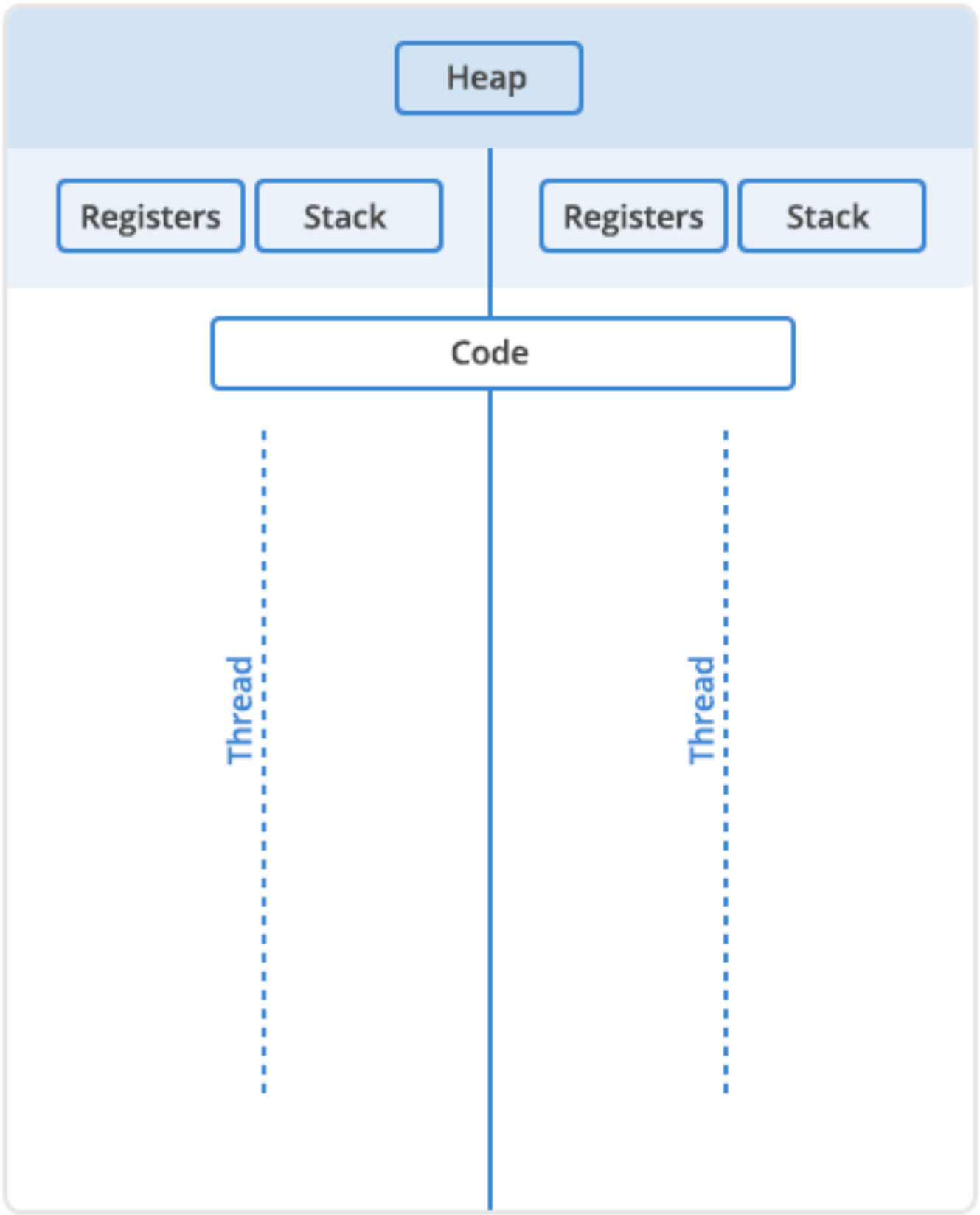
– https://www.backblaze.com/blog/whats-the-diff-programs-processes-and-threads/


Time
Process
Thread
Thread
Thread

Single Thread



Multi Threaded



“concurrency is the *composition* of independently executing processes”

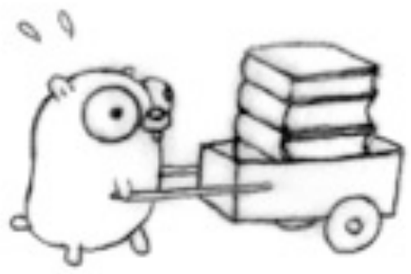
–Rob Pike, Andrew Gerrand - https://blog.golang.org/concurrency-is-not-parallelism

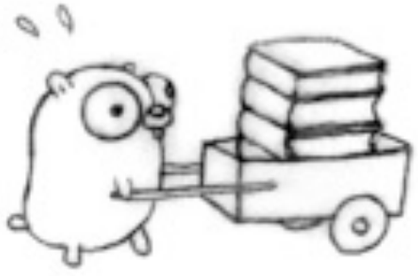
Concurrency

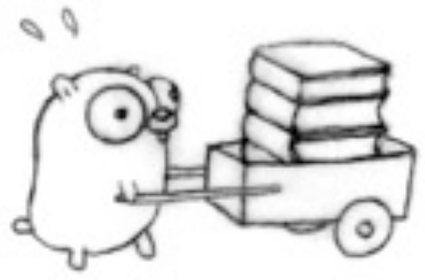
- Resembles the world
- Management of multiple tasks

“parallelism is the simultaneous *execution* of (possibly related) computations”

–Rob Pike, Andrew Gerrand - https://blog.golang.org/concurrency-is-not-parallelism







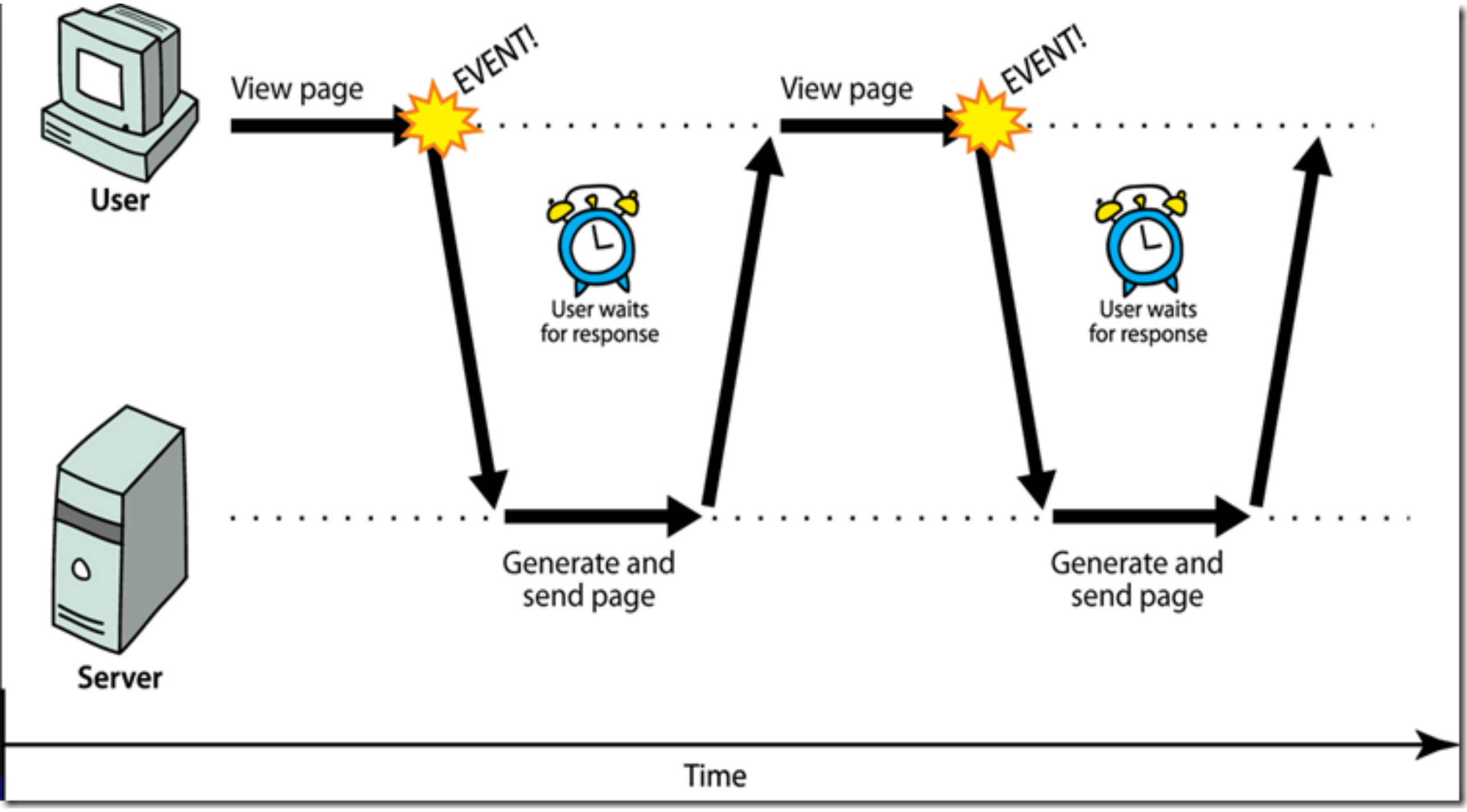
“Concurrency is about *dealing with* lots of things at once.

Parallelism is about *doing* lots of things at once.”

–Rob Pike, Andrew Gerrand - https://blog.golang.org/concurrency-is-not-parallelism

Synchronous

- Linear
- Sequential
- Easy to follow


User
View page
EVENT!
User waits
for response
View page
EVENT!
User waits
for response
Server
Generate and
send page
Generate and
send page
Time

Synchronous sleep demo

Simultaneous execution

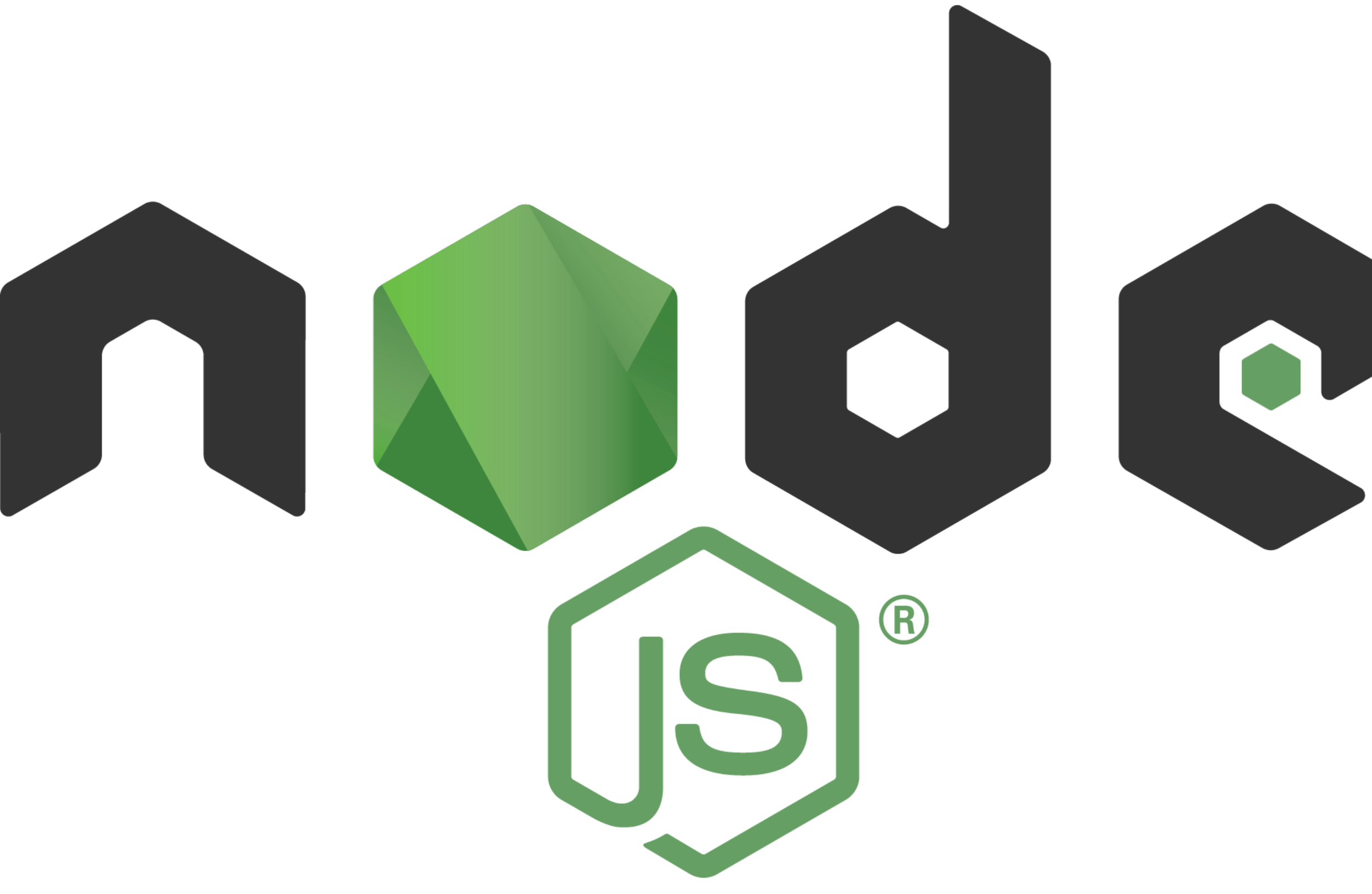
- Speedups
- Challenges
    - Race conditions
    - Debugging
- Limitations

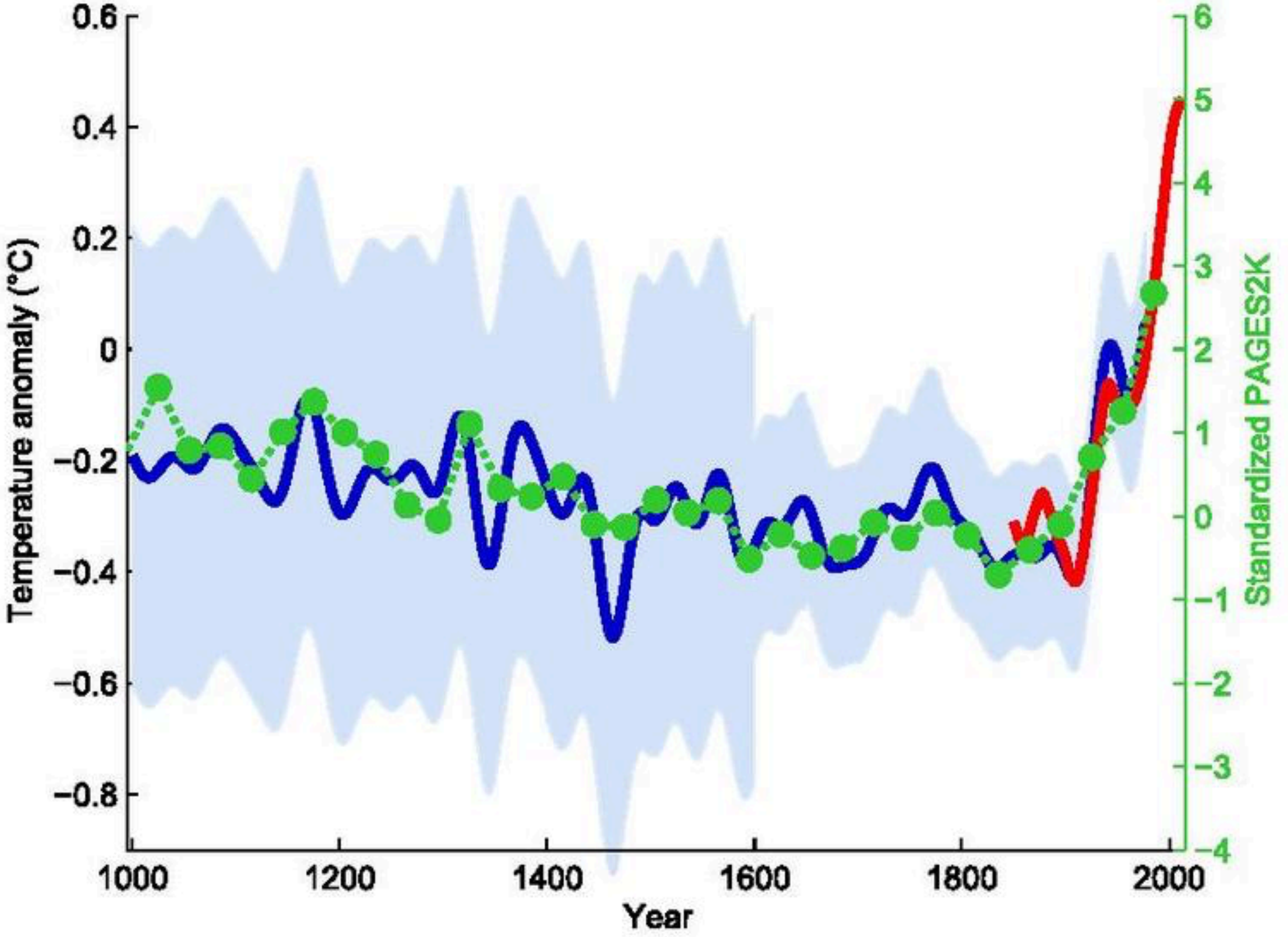
Simultaneous execution in Python

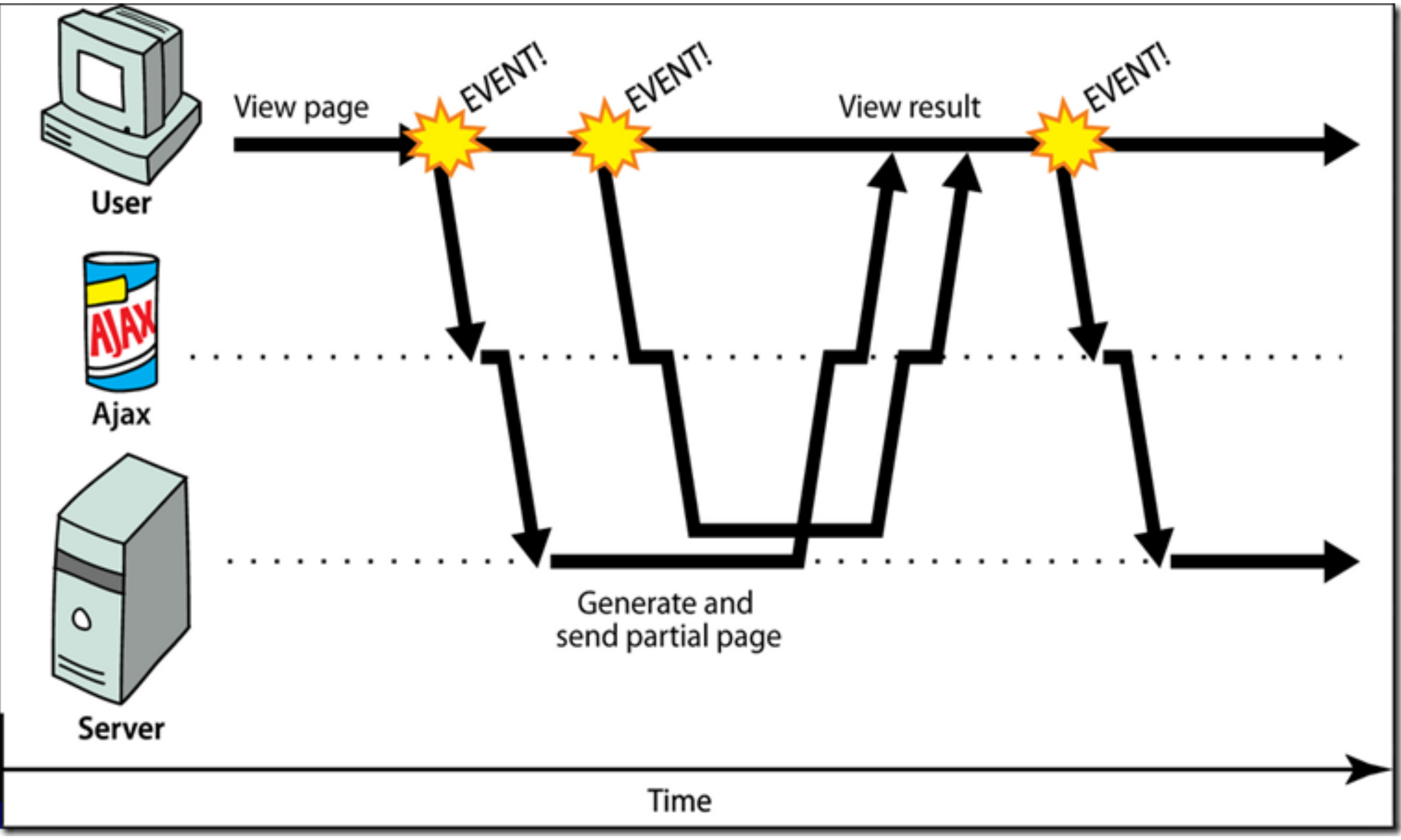
- GIL
- Workaround: Multiprocessing
  - Runaway processes

Single thread - many jobs

- What now?
- Slow upstreams
- High client counts


JS
R


0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
Temperature anomaly (°C)
6
5
4
3
2
1
0
−1
−2
−3
−4
Standardized PAGES2K
1000
1200
1400
1600
1800
2000
Year


User
Ajax
Server
View page
EVENT!
EVENT!
View result
EVENT!
Generate and
send partial page
Time

User experience

- Callback hell
- Error handling
- Debugging
- Flow control
- Maintenance

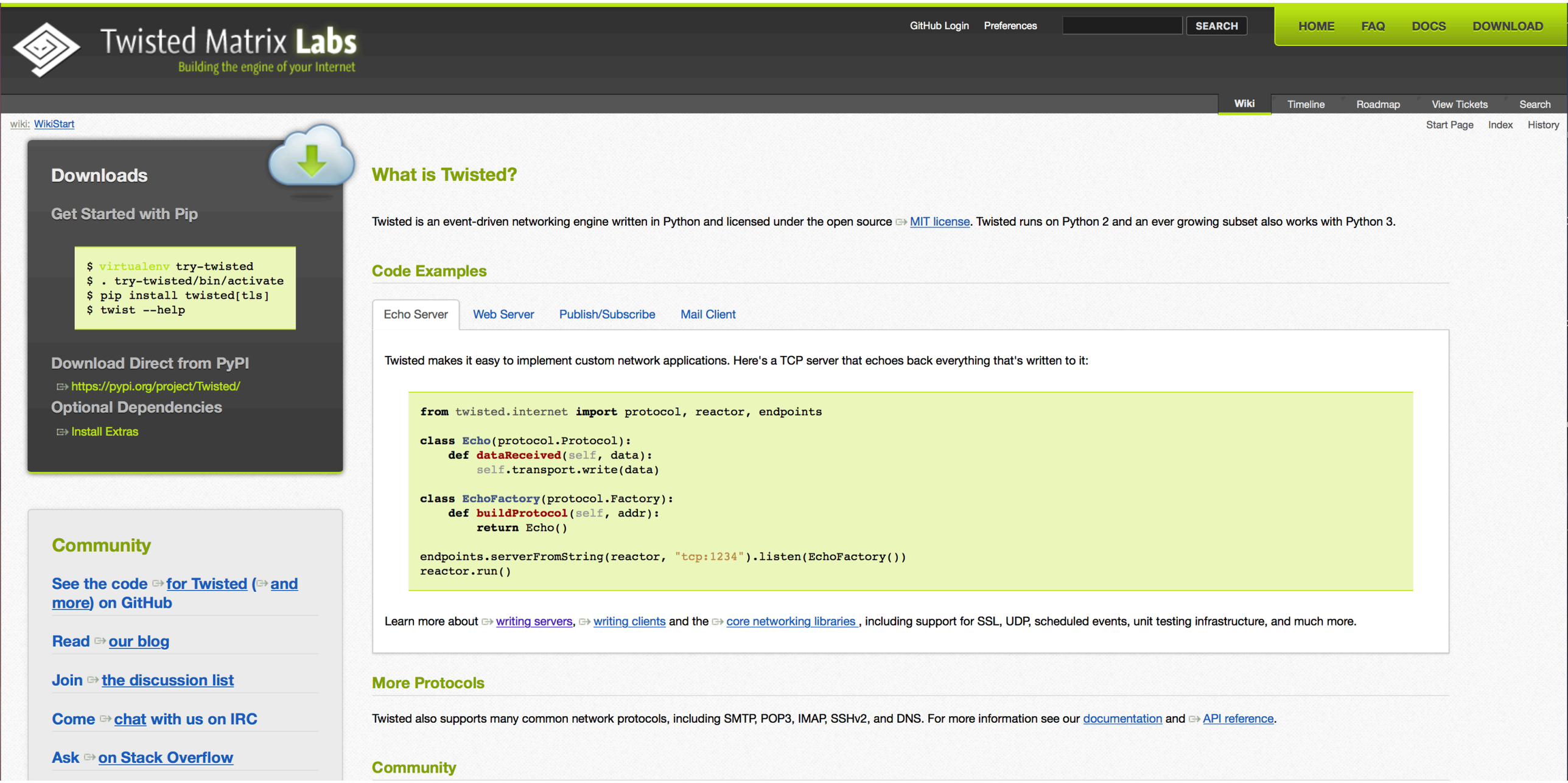
```
fs.readdir(source, function (err, files) {
  if (err) {
    console.log('Error finding files: ' + err)
  } else {
    files.forEach(function (filename, fileIndex) {
      console.log(filename)
      gm(source + filename).size(function (err, values) {
        if (err) {
          console.log('Error identifying file size: ' + err)
        } else {
          console.log(filename + ' : ' + values)
          aspect = (values.width / values.height)
          widths.forEach(function (width, widthIndex) {
            height = Math.round(width / aspect)
            console.log('resizing ' + filename + 'to ' + height + 'x' + height)
            this.resize(width, height).write(dest + 'w' + width + '_' + filename, function(err) {
              if (err) console.log('Error writing file: ' + err)
            })
          }.bind(this))
        }
      })
    })
  }
})
```

Async sleep demo

When async?

- External resources
- Databases
- Multiple client requests
- Remote caches (redis, memcache)

Twisted

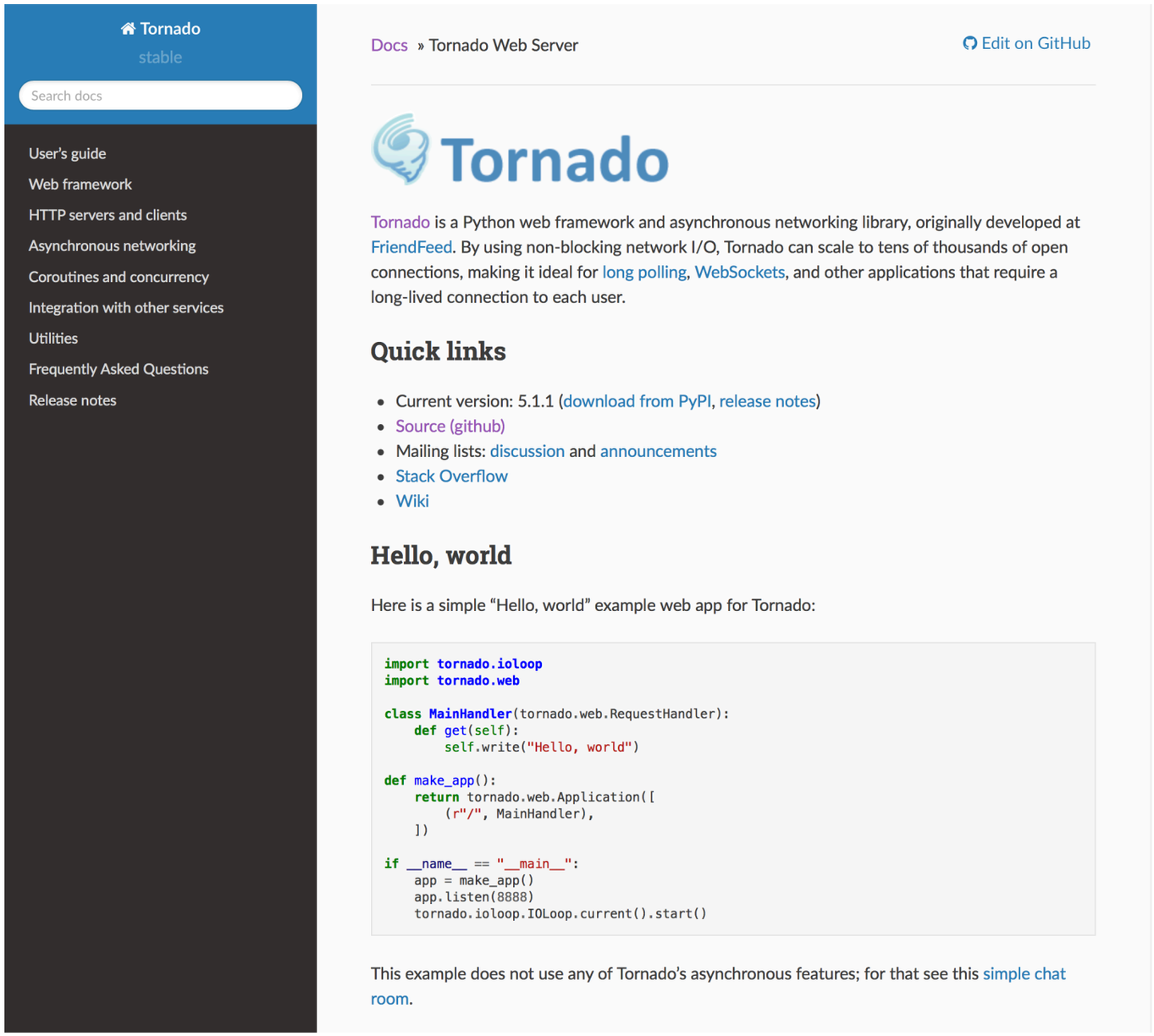


https://twistedmatrix.com/trac/

Twisted

- 2002
- Network framework
- Event-driven
- deferred / future

Tornado



http://www.tornadoweb.org/en/stable/

Tornado

- 2009
- Web focussed

asyncio

Previous topic
Networking and Interprocess Communication

Next topic
Coroutines and Tasks

This Page
Report a Bug
Show Source

asyncio — Asynchronous I/O

asyncio is a library to write **concurrent** code using the **async/await** syntax.

asyncio is used as a foundation for multiple Python asynchronous frameworks that provide high-performance network and web-servers, database connection libraries, distributed task queues, etc.

asyncio is often a perfect fit for IO-bound and high-level **structured** network code.

Hello World!

```
import asyncio

async def main():
    print('Hello ...')
    await asyncio.sleep(1)
    print('... World!')

# Python 3.7+
asyncio.run(main())
```

asyncio provides a set of **high-level** APIs to:

- run Python coroutines concurrently and have full control over their execution;
- perform network IO and IPC;
- control subprocesses;
- distribute tasks via queues;
- synchronize concurrent code;

Additionally, there are **low-level** APIs for *library and framework developers* to:

- create and manage event loops, which provide asynchronous APIs for `networking`, running `subprocesses`, handling `OS signals`, etc;
- implement efficient protocols using transports;
- bridge callback-based libraries and code with async/await syntax.

Reference

High-level APIs

- Coroutines and Tasks
- Streams
- Synchronization Primitives
- Subprocesses
- Queues
- Exceptions

https://docs.python.org/3/library/asyncio.html

asyncio

- PEP 3156 https://www.python.org/dev/peps/pep-3156/ in 2014
- Provisional 3.4.x
- Included >= 3.5.x
  - async/await
    - PEP 492 in 2015

http demo

Coroutines

- **async** - declare
- **await** - obtain result
- Context manager
    - async with
- Iterable
    - async for

Tasks

- Schedule coroutines
- asyncio.create_task(coro()) (3.7.x)
- asyncio.ensure_future(coro())
- Return **Task**

Futures

- Lower level than Tasks
- awaitable

asyncio

- complexity++
- Recall network ONLY in stdlib
- driver/library support
- Middleware and specifications

Multiple tasks

- asyncio.gather(*awaitables)

Error handling

- Callback based
  - Check for errors and handle inline
    if error…
- async/await
  - try
    catch
    except

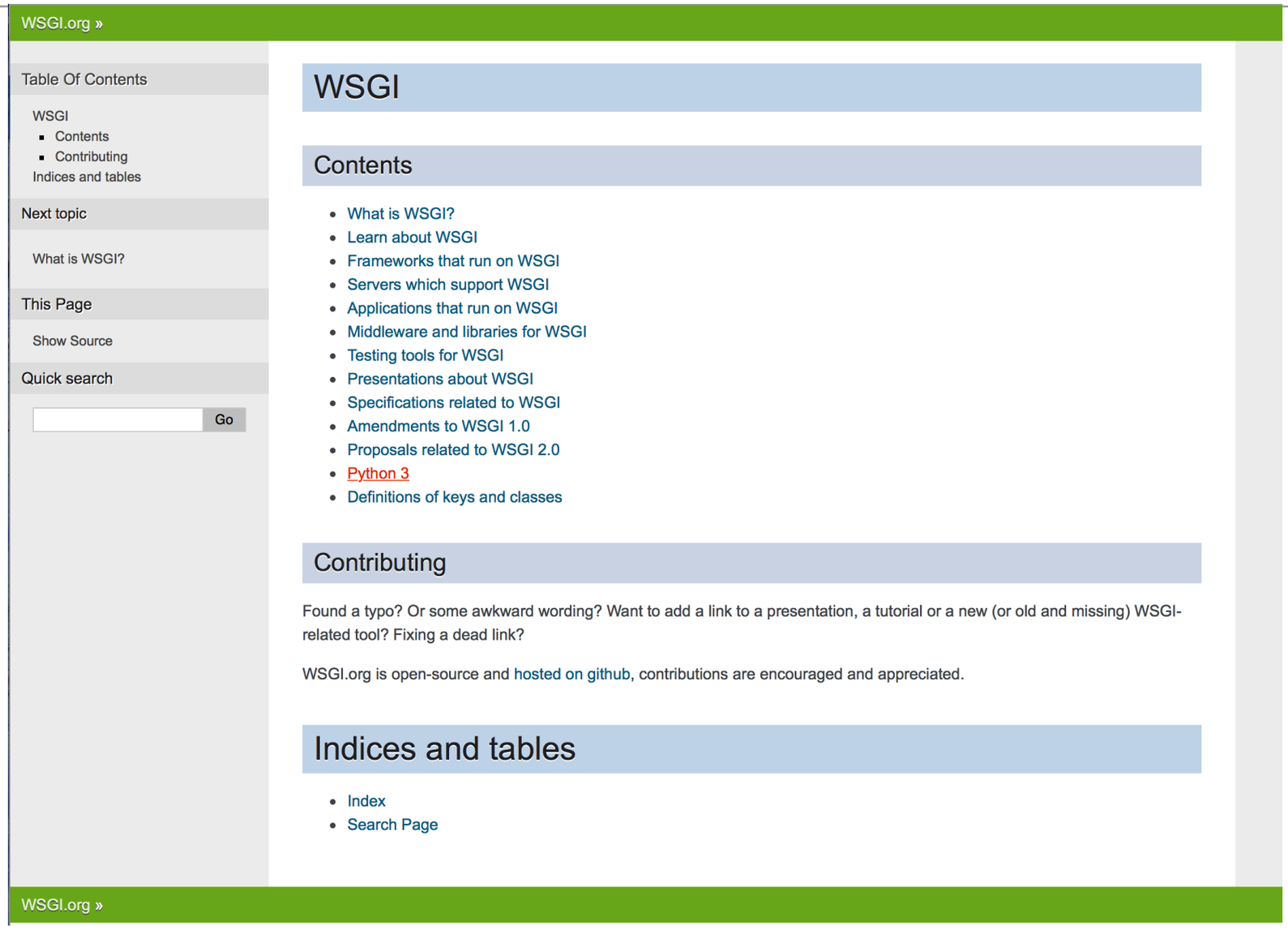
Higher level async

- Server Frameworks
  - Sanic https://sanic.readthedocs.io/en/latest/#
  - Quart http://pgjones.gitlab.io/quart/
  - aiohttp (client and server)
- DB
  - ORMs?

Web Service Interfaces

- WSGI
- ASGI

WSGI



https://wsgi.readthedocs.io/

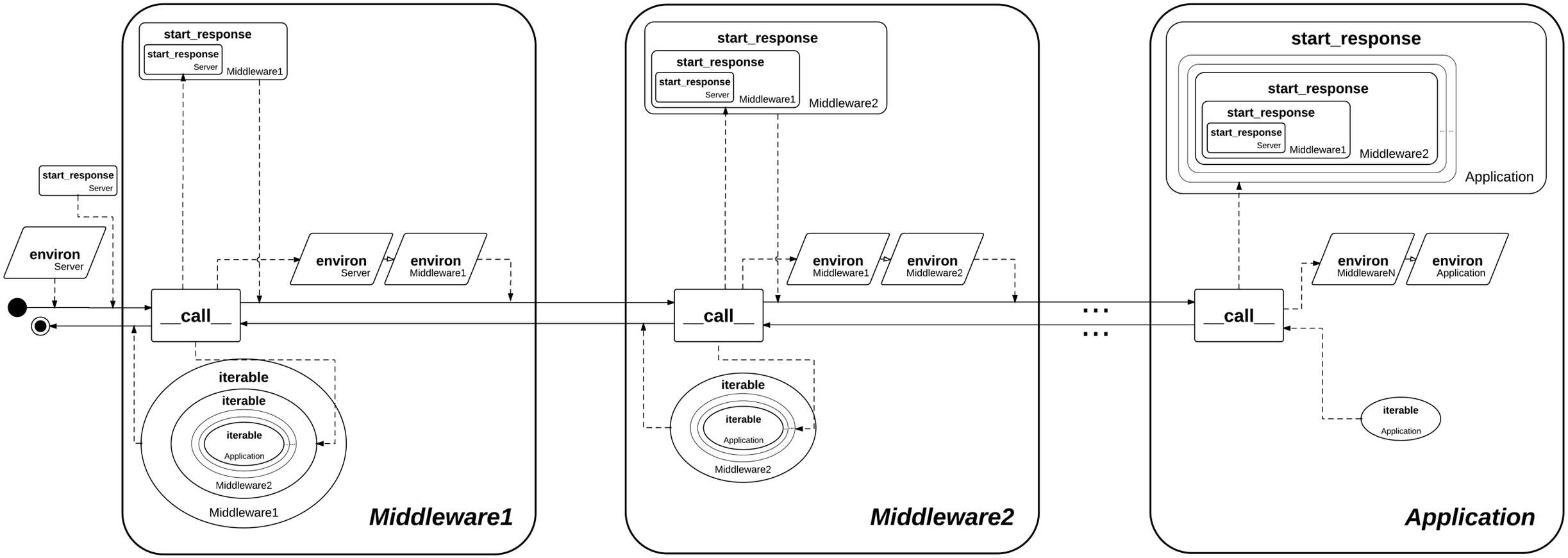
WSGI

- PEP 333
- PEP 3333 for Python 3
- RIP! Synchronous

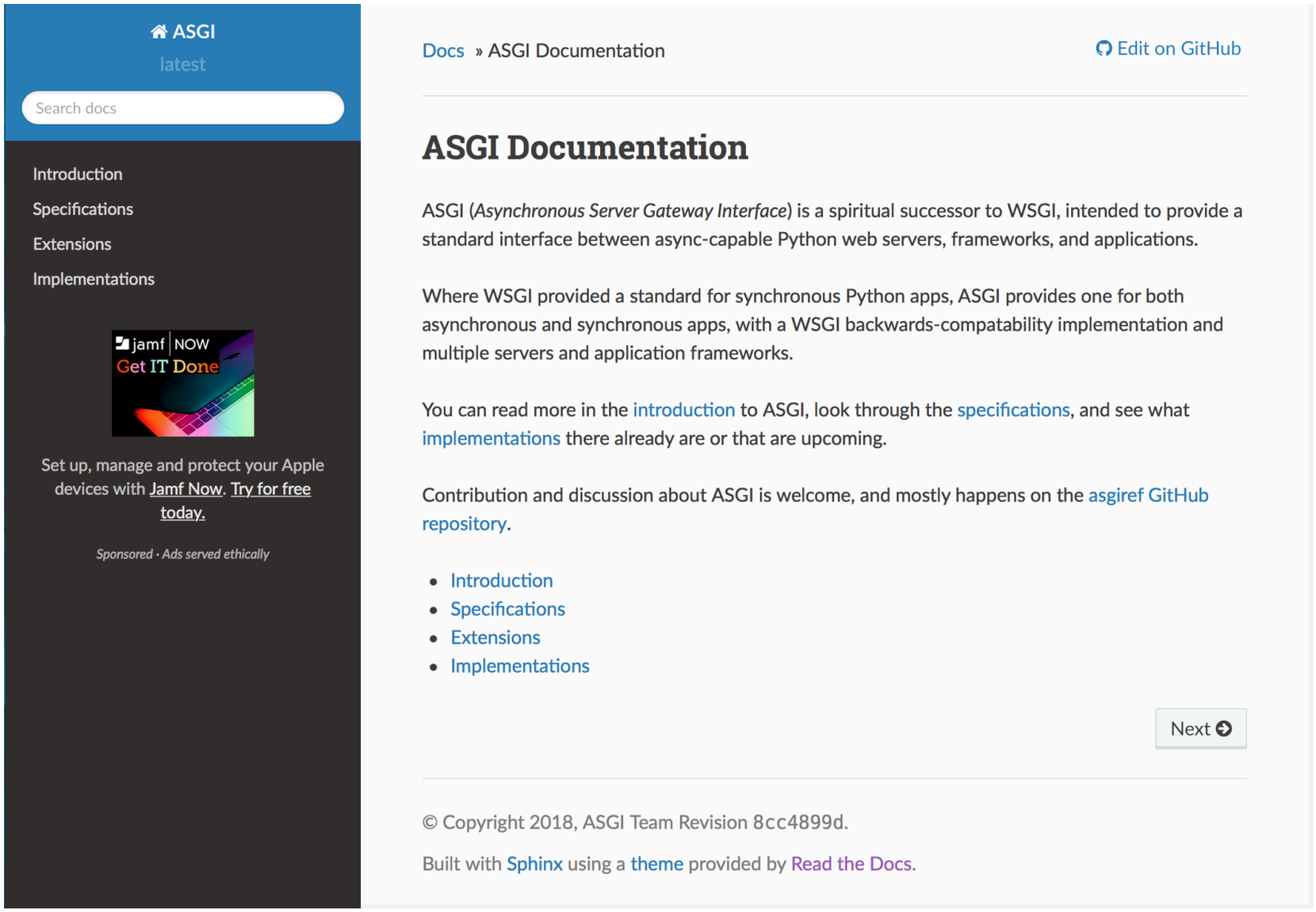
Author: Piotr Dobrogost

WSGI flow (PEP 3333)

created with LUCIDCHART.COM



ASGI



https://asgi.readthedocs.io/en/latest/

Debugging

- asyncio debug mode - https://docs.python.org/3/library/asyncio-dev.html#asyncio-debug-mode
  - Environment variable
    - PYTHONASYNCIODEBUG=1
  - Command line variable
    - python -X dev
  - Code changes
    - asyncio.run(debug=True)
    - loop.set_debug()

Testing

- unittest
  - https://github.com/Martiusweb/asynctest/
- Pytest
  - pytest asyncio https://pypi.org/project/pytest-asyncio/

Questions?

Resources

- Awesome asyncio - https://github.com/timofurrer/awesome-asyncio
- Awesome Python - https://awesome-python.com
- Concurrency is not Parallelism - Rob Pike https://blog.golang.org/concurrency-is-not-parallelism

Thanks

- YOU!
- Python community
- Pycon Pakistan
- Equal Experts

Credits

- Computer Process https://www.backblaze.com/blog/wp-content/uploads/2017/08/diagram-thread-codestack.png
- Threads https://www.backblaze.com/blog/wp-content/uploads/2017/08/diagram-thread-process-1.png
- Threads https://www.backblaze.com/blog/wp-content/uploads/2017/08/diagram-threads.png
- Gophers (single) https://talks.golang.org/2012/waza.slide#12waza.slide#12 CC-BY 3.0
- Gophers (multiple) https://talks.golang.org/2012/waza.slide#15 CC-BY 3.0
- Synchronous Execution http://lh6.ggpht.com/-L6xcGEmyc8Y/Uv2QS3Uh7gI/AAAAAAAAh6w/X_5JOjeO-z4/image_thumb%25255B24%25255D.png
- Asynchronrous Execution http://lh3.ggpht.com/-DOKIg0FjCJ4/Uv2QVjQfIlI/AAAAAAAAh7A/7tijlSlZseE/image_thumb%25255B26%25255D.png
- Hockey Stick Graph https://upload.wikimedia.org/wikipedia/commons/thumb/2/2d/T_comp_61-90.pdf/page1-795px-T_comp_61-90.pdf.jpg
- WSGI https://i.stack.imgur.com/glnJA.png
- Callback hell http://callbackhell.com