

The GIL is irrelevant (sometimes)

Asynchronous 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

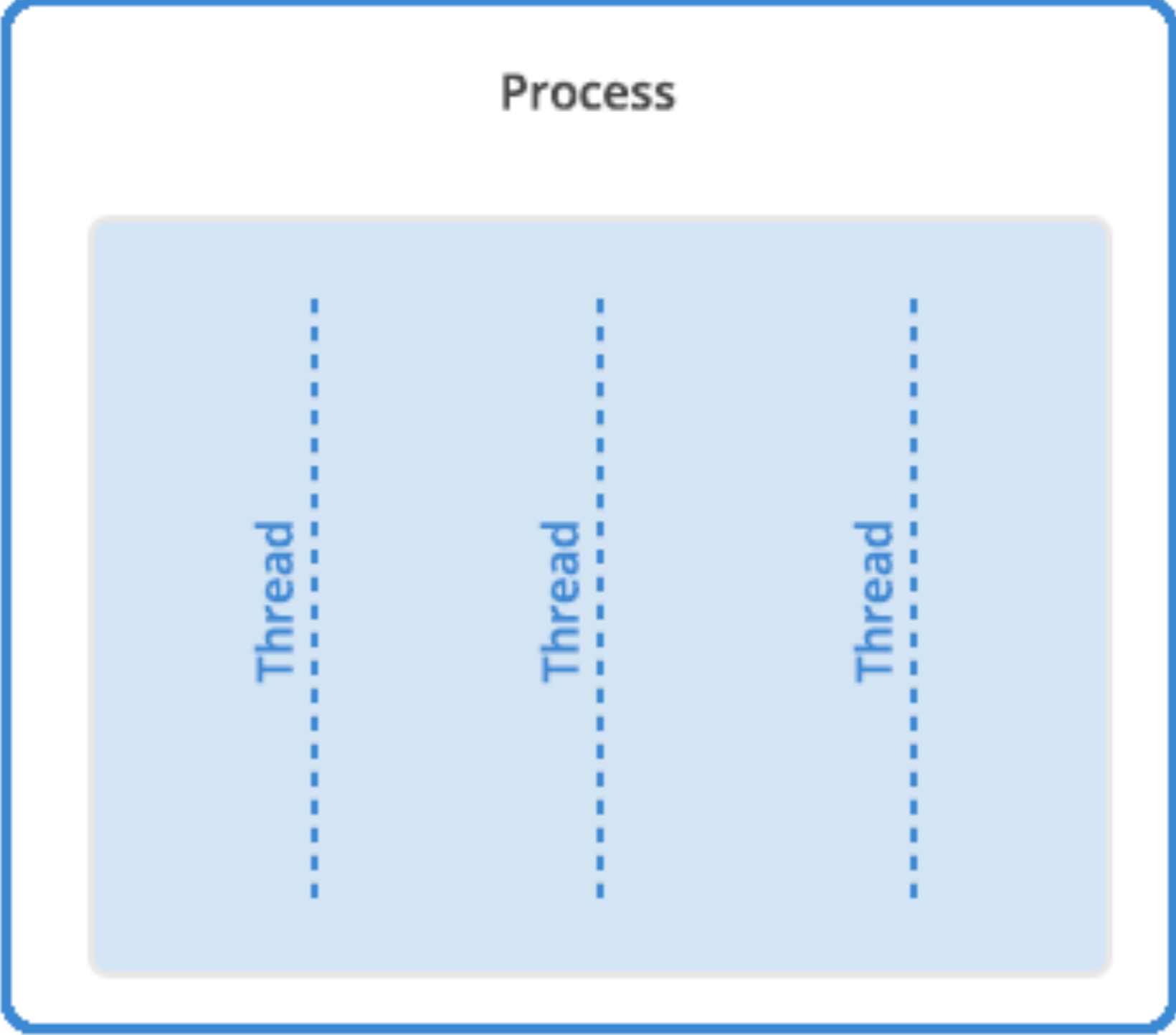
Code

“Instance of a program in execution”

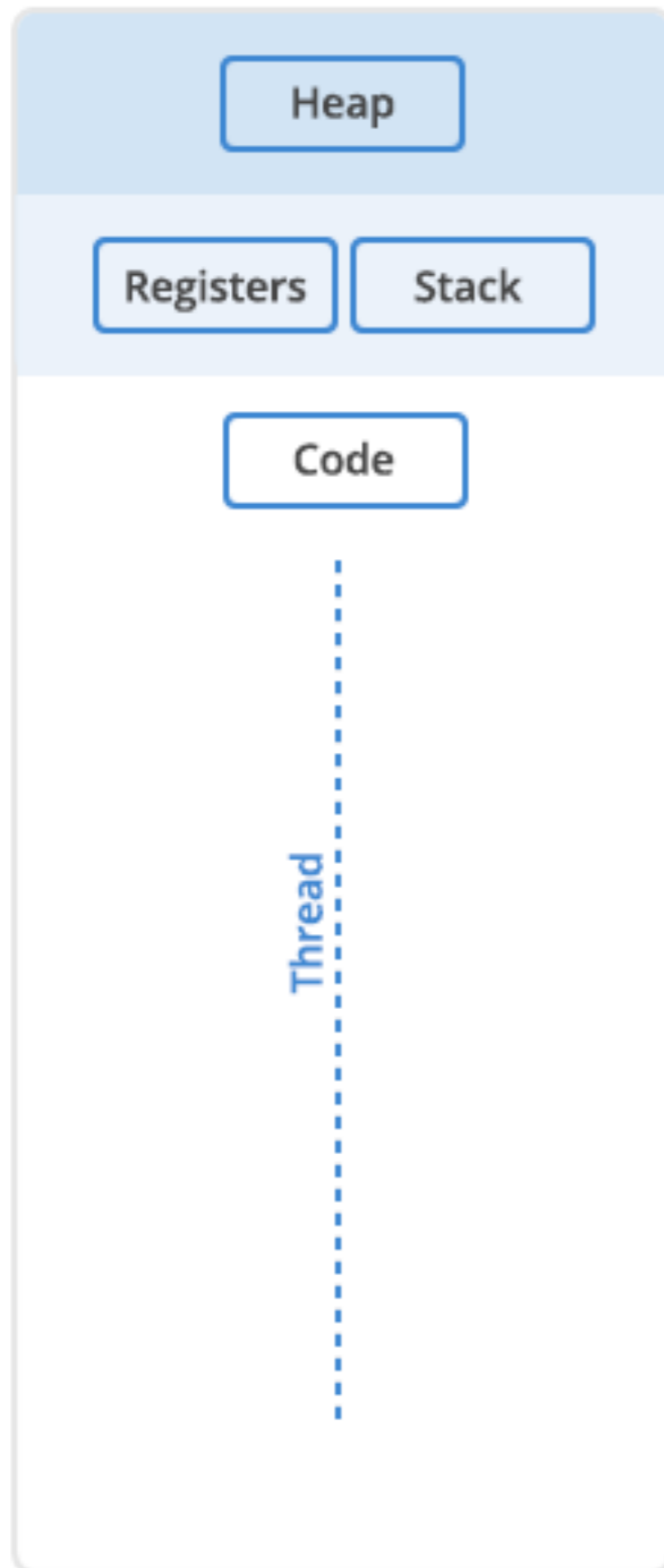
- https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/3_Processes.html

the unit of execution within a process

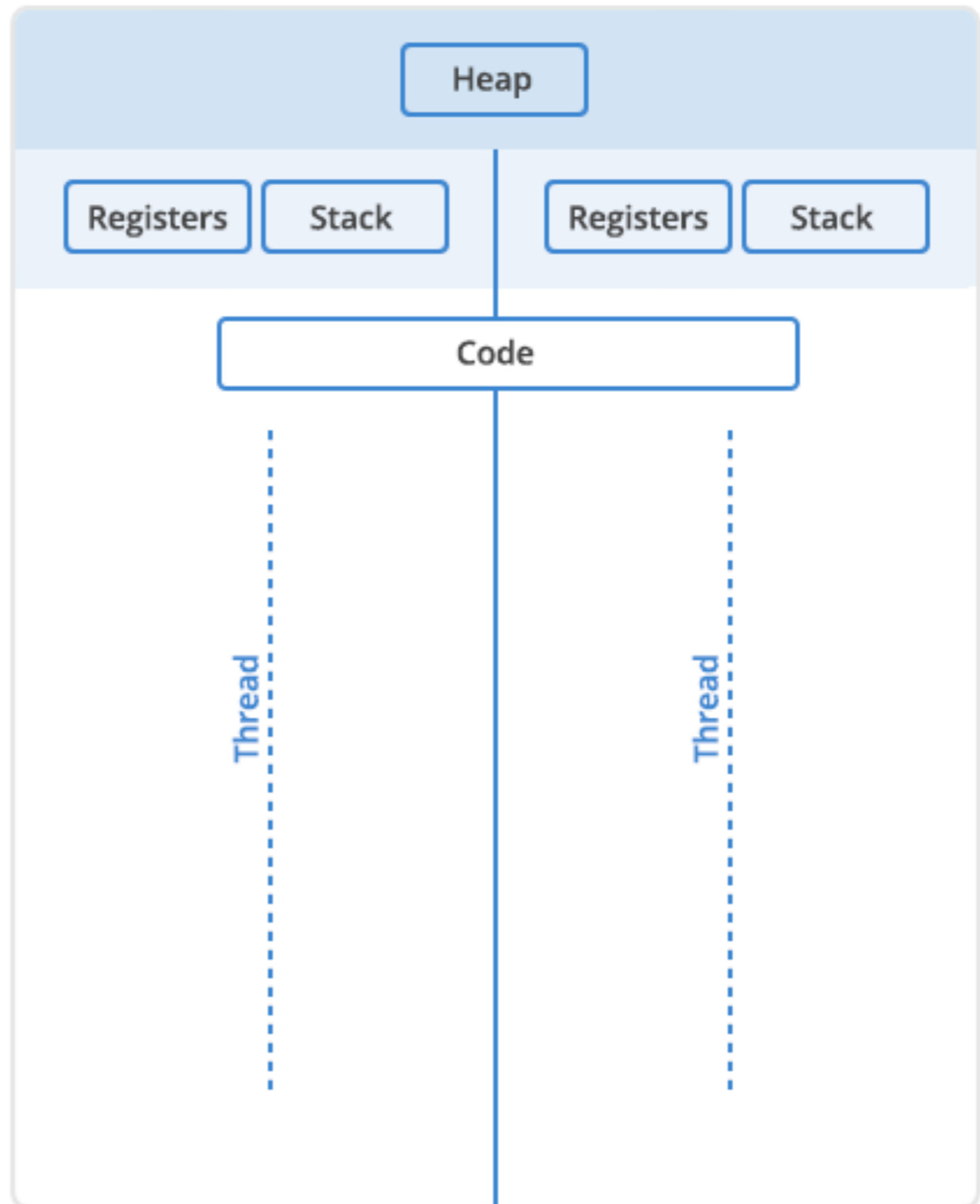
- <https://www.backblaze.com/blog/whats-the-diff-programs-processes-and-threads/>



Single Thread



Multi Threaded



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

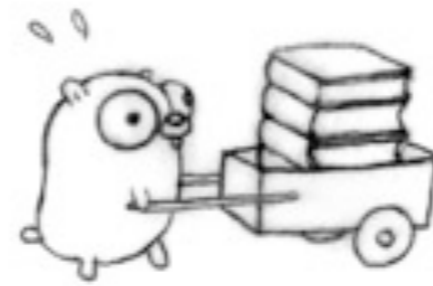
–Rob Pike, Andrew Gerrand - <https://blog.golang.org/concurrency-is-not-parallelism>

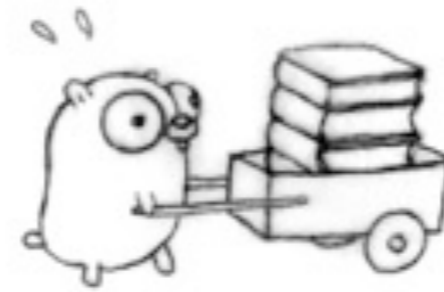
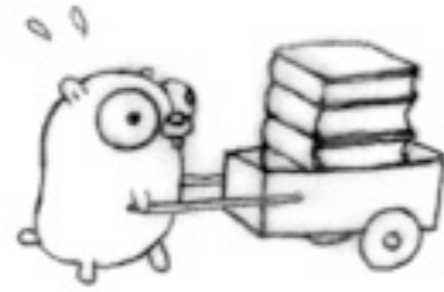
Concurrency

- 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>





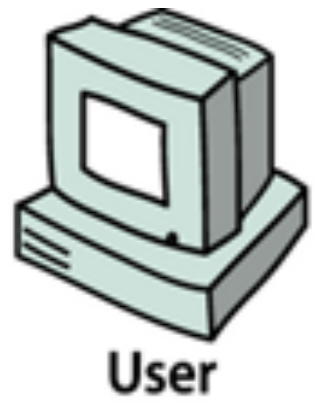
“Concurrency is about *dealing with* lots of things at once.

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

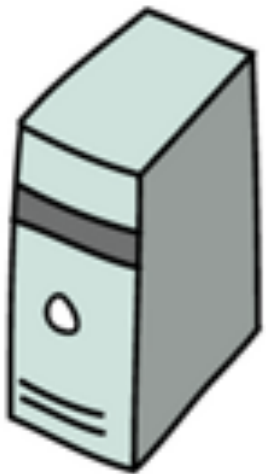
–Rob Pike, Andrew Gerrand - <https://blog.golang.org/concurrency-is-not-parallelism>

Synchronous

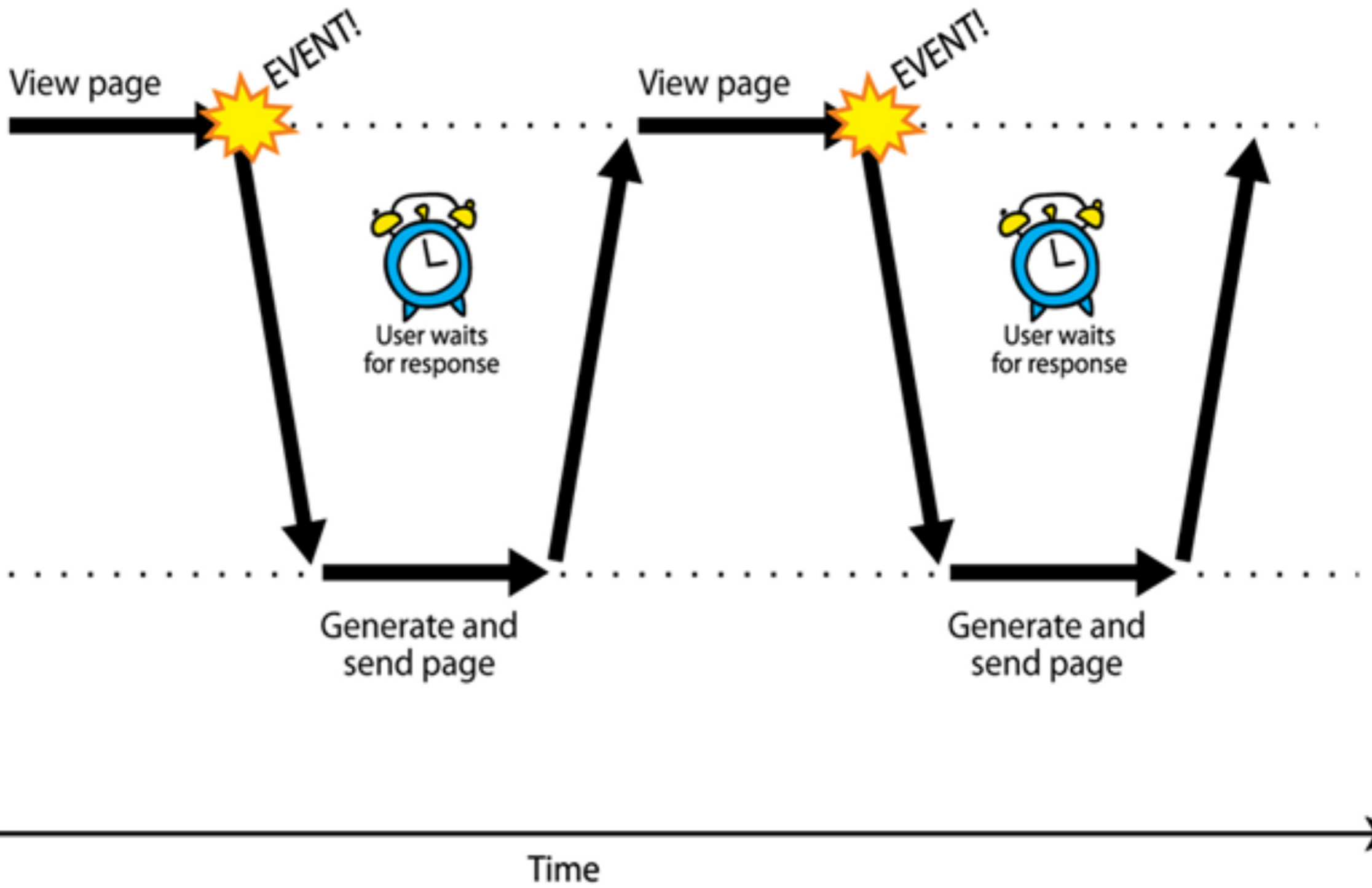
- Linear
- Sequential
- Easy to follow



User



Server



Synchronous sleep demo

Simultaneous execution

- Speedups
- Challenges
 - Race conditions
 - Debugging
- Limitations

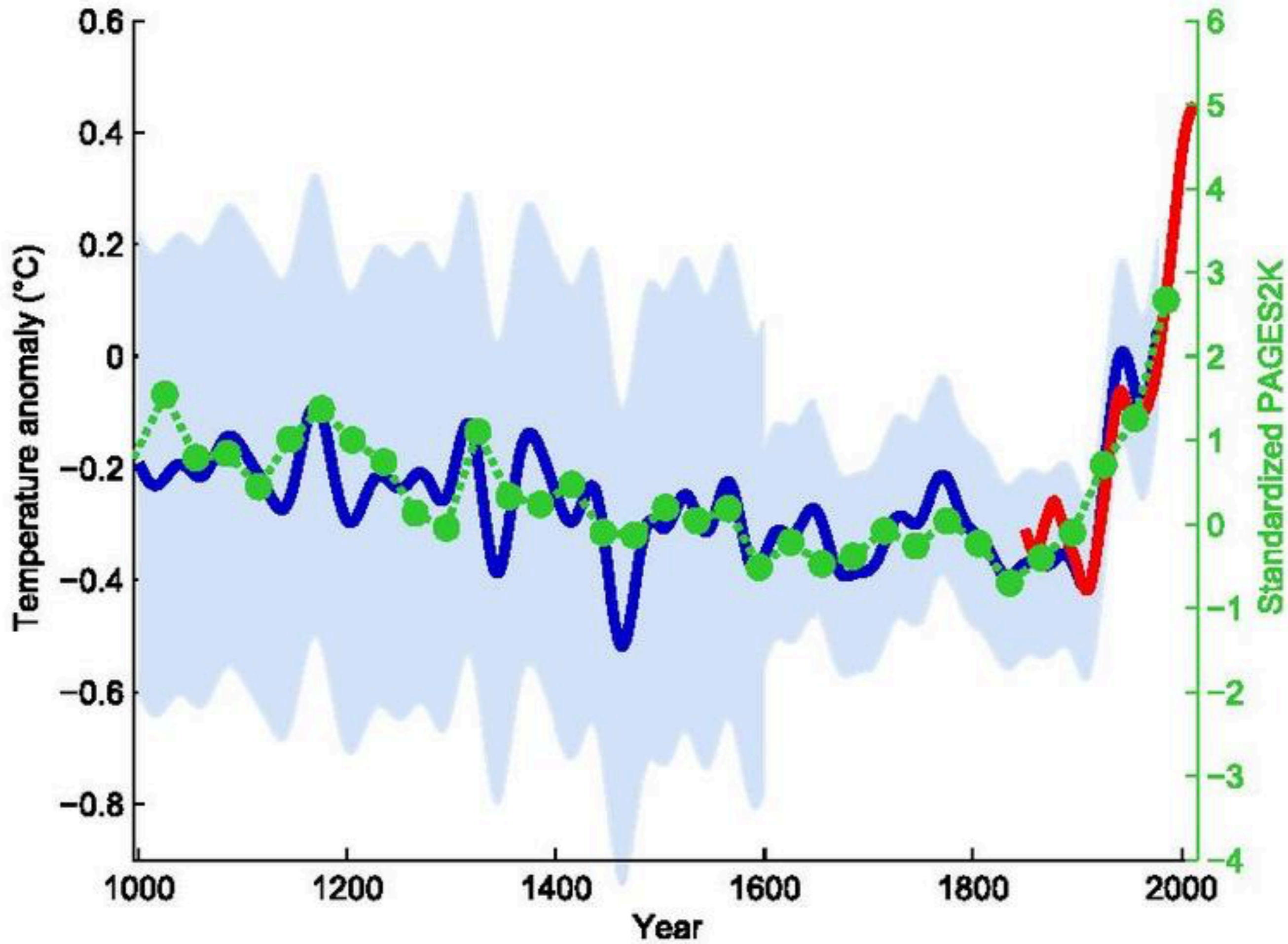
Simultaneous execution in Python

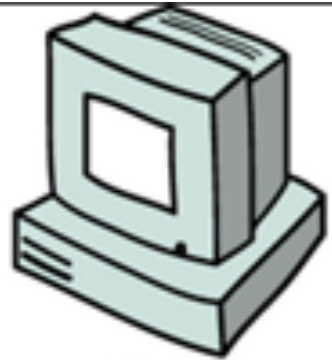
- GIL
- Workaround: Multiprocessing
 - Runaway processes

Single thread - many jobs

- What now?
- Slow upstreams
- High client counts



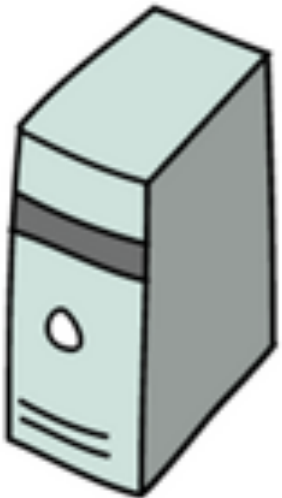




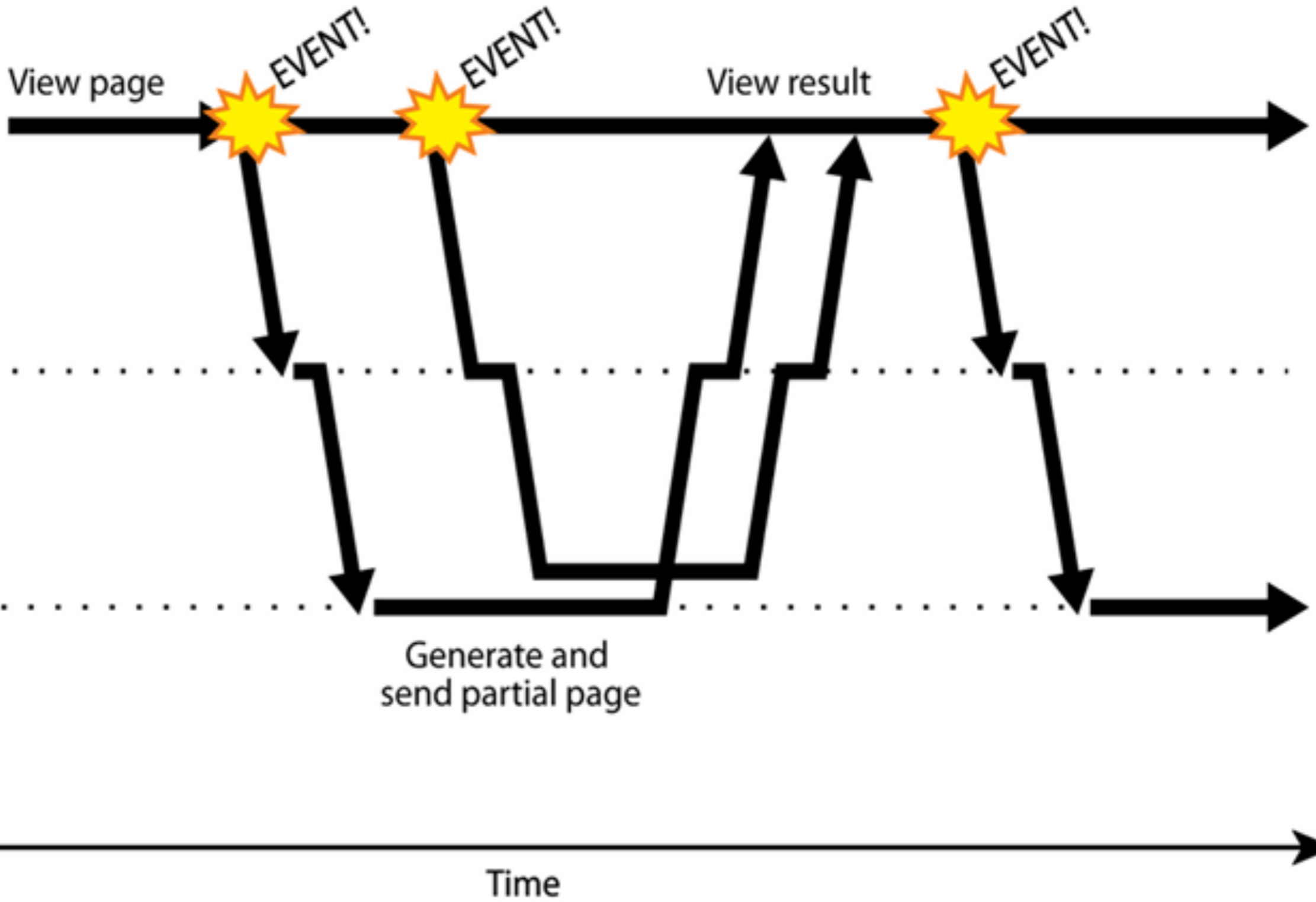
User



Ajax



Server



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

Downloads

Get Started with Pip

```
$ virtualenv try-twisted
$ . try-twisted/bin/activate
$ pip install twisted[tls]
$ twist --help
```

Download Direct from PyPI

⇒ <https://pypi.org/project/Twisted/>

Optional Dependencies

⇒ [Install Extras](#)

Community

[See the code](#) ⇒ [for Twisted](#) (⇒ [and more](#)) [on GitHub](#)

[Read](#) ⇒ [our blog](#)

[Join](#) ⇒ [the discussion list](#)

[Come](#) ⇒ [chat with us on IRC](#)

[Ask](#) ⇒ [on Stack Overflow](#)

What is Twisted?

Twisted is an event-driven networking engine written in Python and licensed under the open source ⇒ [MIT license](#). Twisted runs on Python 2 and an ever growing subset also works with Python 3.

Code Examples

[Echo Server](#) [Web Server](#) [Publish/Subscribe](#) [Mail Client](#)

Twisted makes it easy to implement custom network applications. Here's a TCP server that echoes back everything that's written to it:

```
from twisted.internet import protocol, reactor, endpoints

class Echo(protocol.Protocol):
    def dataReceived(self, data):
        self.transport.write(data)

class EchoFactory(protocol.Factory):
    def buildProtocol(self, addr):
        return Echo()

endpoints.serverFromString(reactor, "tcp:1234").listen(EchoFactory())
reactor.run()
```

Learn more about ⇒ [writing servers](#), ⇒ [writing clients](#) and the ⇒ [core networking libraries](#), including support for SSL, UDP, scheduled events, unit testing infrastructure, and much more.

More Protocols

Twisted also supports many common network protocols, including SMTP, POP3, IMAP, SSHv2, and DNS. For more information see our [documentation](#) and ⇒ [API reference](#).

Community

<https://twistedmatrix.com/trac/>

Twisted

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

Tornado

🏠 Tornado
stable

Search docs

- User's guide
- Web framework
- HTTP servers and clients
- Asynchronous networking
- Coroutines and concurrency
- Integration with other services
- Utilities
- Frequently Asked Questions
- Release notes

Docs » Tornado Web Server [Edit on GitHub](#)

Tornado

Tornado is a Python web framework and asynchronous networking library, originally developed at FriendFeed. By using non-blocking network I/O, Tornado can scale to tens of thousands of open connections, making it ideal for [long polling](#), [WebSockets](#), and other applications that require a long-lived connection to each user.

Quick links

- Current version: 5.1.1 ([download from PyPI](#), [release notes](#))
- [Source \(github\)](#)
- Mailing lists: [discussion](#) and [announcements](#)
- [Stack Overflow](#)
- [Wiki](#)

Hello, world

Here is a simple "Hello, world" example web app for Tornado:

```
import tornado.ioloop
import tornado.web

class MainHandler(tornado.web.RequestHandler):
    def get(self):
        self.write("Hello, world")

def make_app():
    return tornado.web.Application([
        (r"/", MainHandler),
    ])

if __name__ == "__main__":
    app = make_app()
    app.listen(8888)
    tornado.ioloop.IOLoop.current().start()
```

This example does not use any of Tornado's asynchronous features; for that see this [simple chat room](#).

<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.

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](#)

Hello World!

```
import asyncio

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

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

<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 \geq 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

The screenshot shows the WSGI.org website with a green header and footer. The main content area is divided into sections: 'WSGI', 'Contents', 'Contributing', and 'Indices and tables'. A sidebar on the left contains navigation links like 'Table Of Contents', 'Next topic', and a search box.

WSGI.org »

Table Of Contents

- WSGI
 - Contents
 - Contributing
- Indices and tables

Next topic

What is WSGI?

This Page

Show Source

Quick search

WSGI

Contents

- [What is WSGI?](#)
- [Learn about WSGI](#)
- [Frameworks that run on WSGI](#)
- [Servers which support WSGI](#)
- [Applications that run on WSGI](#)
- [Middleware and libraries for WSGI](#)
- [Testing tools for WSGI](#)
- [Presentations about WSGI](#)
- [Specifications related to WSGI](#)
- [Amendments to WSGI 1.0](#)
- [Proposals related to WSGI 2.0](#)
- [Python 3](#)
- [Definitions of keys and classes](#)

Contributing

Found a typo? Or some awkward wording? Want to add a link to a presentation, a tutorial or a new (or old and missing) WSGI-related tool? Fixing a dead link?

WSGI.org is open-source and [hosted on github](#), contributions are encouraged and appreciated.

Indices and tables

- [Index](#)
- [Search Page](#)

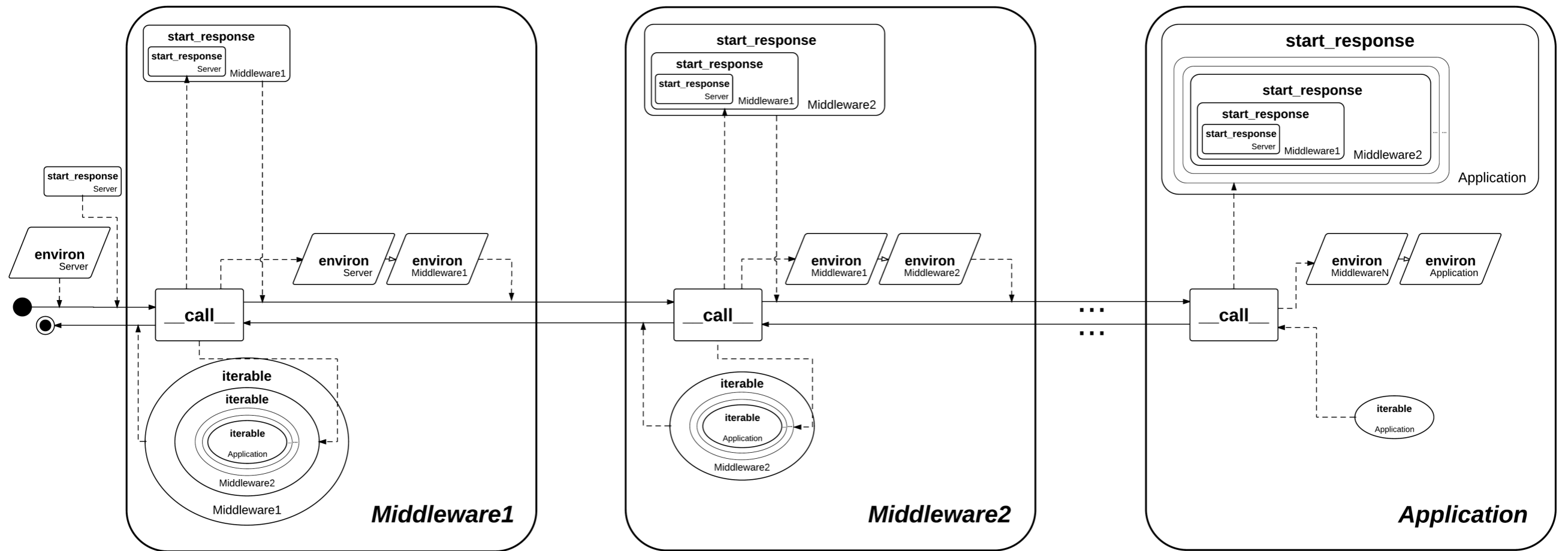
WSGI.org »

<https://wsgi.readthedocs.io/>

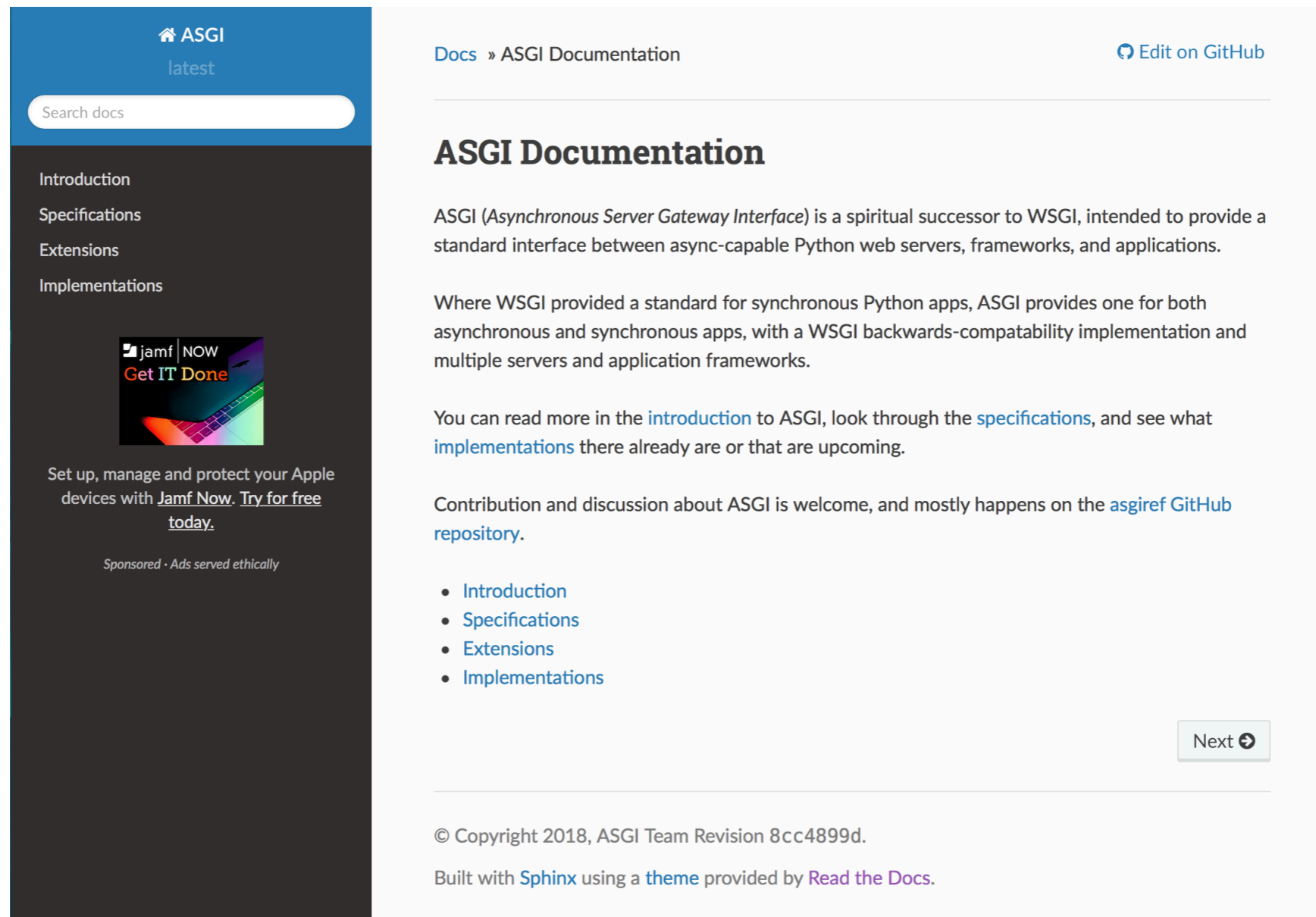
WSGI

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

WSGI flow (PEP 3333)



ASGI



The screenshot displays the ASGI documentation website. The left sidebar contains a navigation menu with the following items: Introduction, Specifications, Extensions, and Implementations. Below the menu is a sponsored advertisement for Jamf Now, which includes the text: "Set up, manage and protect your Apple devices with Jamf Now. Try for free today." and "Sponsored · Ads served ethically". The main content area features a breadcrumb trail "Docs » ASGI Documentation" and a link to "Edit on GitHub". The title "ASGI Documentation" is prominently displayed. The text explains that ASGI (Asynchronous Server Gateway Interface) is a spiritual successor to WSGI, designed to provide a standard interface between async-capable Python web servers, frameworks, and applications. It notes that ASGI provides a standard for both asynchronous and synchronous apps, with a WSGI backwards-compatibility implementation and support for multiple servers and application frameworks. Users are directed to the introduction, specifications, and implementations sections for more information. A list of links is provided: Introduction, Specifications, Extensions, and Implementations. A "Next" button with a right-pointing arrow is located at the bottom right of the main content area. The footer contains the copyright notice "© Copyright 2018, ASGI Team Revision 8cc4899d." and mentions that the site is built with Sphinx using a theme provided by Read the Docs.

<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/asyncctest/>
- 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/AAAAAAAAAh6w/X_5JOjeO-z4/image_thumb%25255B24%25255D.png
- Asynchronous Execution http://lh3.ggpht.com/-DOKlg0FjCJ4/Uv2QVjQfIII/AAAAAAAAAh7A/7tjIStZseE/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>