ThoughtWorks®

In Python 3.5+

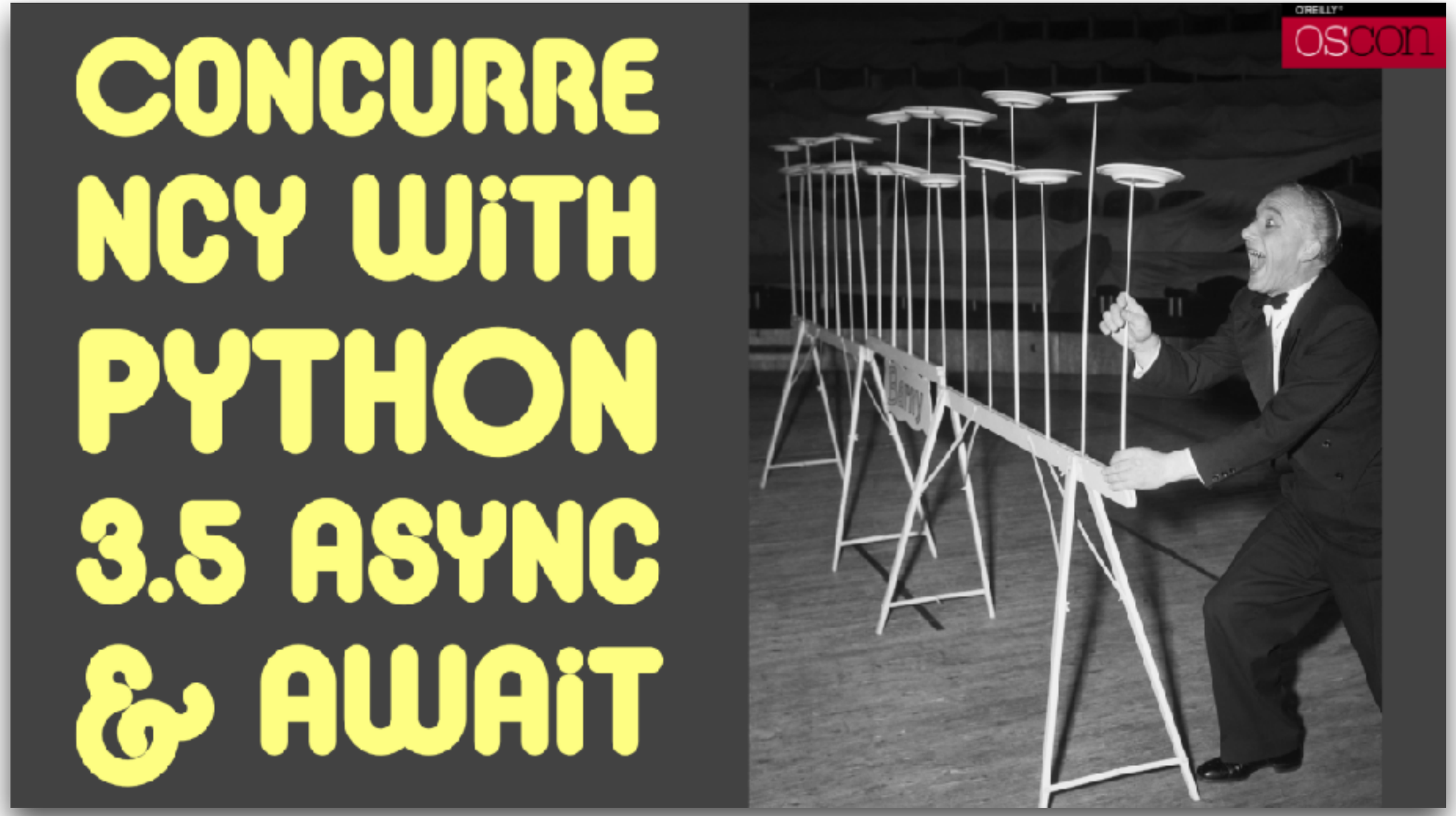
MODERN CONCURRENCY

The new world of async/await

PyBay

San Francisco
2017

AN EXTENDED AND UPDATED VERSION OF...



Presented at OSCON Europe, 2015

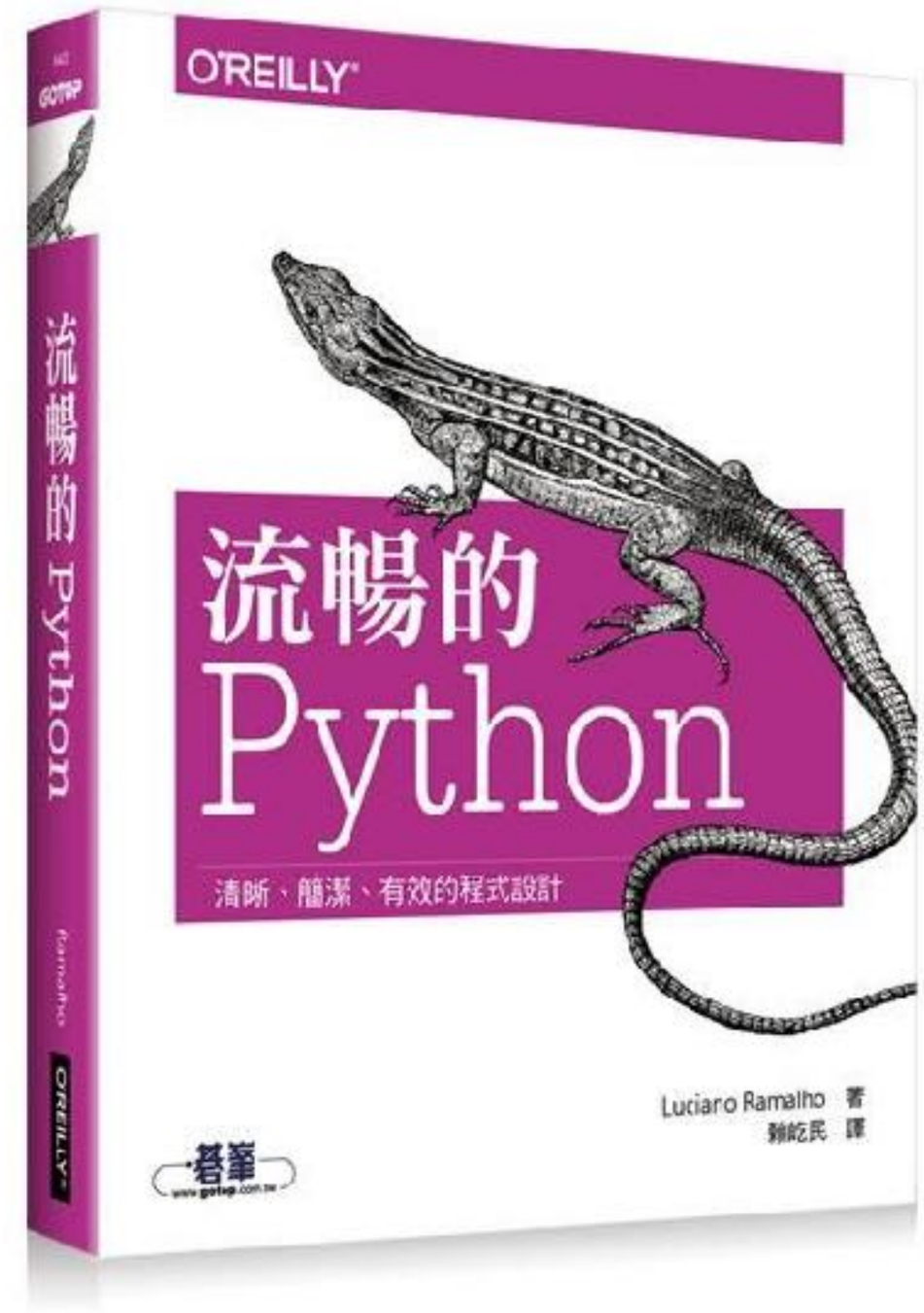
ThoughtWorks®

LUCIANO RAMALHO

Technical Principal

@ramalhoorg
luciano.ramalho@thoughtworks.com

FLUENT PYTHON, MY FIRST BOOK



4.7 stars at Amazon.com

Fluent Python (O'Reilly, 2015)

Python Fluente (Novatec, 2015)

Python к вершинам мастерства* (DMK, 2015)

流暢的 Python† (Gotop, 2016)

also in **Polish, Korean...**

** Python. To the heights of excellence*
† Smooth Python

ThoughtWorks®

CONCURRENCY

Not the same as parallelism

CONCURRENCY VS. PARALLELISM

Concurrency vs. parallelism

Concurrency is about dealing with lots of things at once.

Parallelism is about doing lots of things at once.

Not the same, but related.

Concurrency is about structure, parallelism is about execution.

Concurrency provides a way to structure a solution to solve a problem that may (but not necessarily) be parallelizable.

Rob Pike - 'Concurrency Is Not Parallelism'
https://www.youtube.com/watch?v=cN_DpYBzKso

PLATE SPINNING

The essential idea of concurrency: spinning 18 plates does not require 18 hands.

You can do it with 2 hands, if you know when each plate needs an intervention to keep spinning.

CHESS EXHIBITION ANALOGY

By Michael Grinberg in
“Asynchronous Python for the Complete Beginner” (PyCon 2017)

Real World Analogy: Chess Exhibition

Simultaneous chess exhibition by Judit Polgár, 1992 Photo by Ed Yourdon

- Assumptions:
  - 24 opponents
  - Polgár moves in 5 seconds
  - Opponents move in 55 seconds
  - Games average 30 move pairs
- Each game runs for 30 minutes
- 24 sequential games would take

24 x 30 min = 720 min = 12 hours!!!

CHESS EXHIBITION ANALOGY (2)

By Michael Grinberg in
“Asynchronous Python for the Complete Beginner” (PyCon 2017)

Real World Analogy: Chess Exhibition

Simultaneous chess exhibition by Judit Polgár, 1992 Photo by Ed Yourdon

- Polgár moves on first game
- While opponent thinks, she moves on second game, then third, fourth...
- A move on all 24 games takes her
  24 x 5 sec = 120 sec = 2 min
- After she completes the round, the first game is ready for her next move!
- 24 games are completed in
  2 min x 30 = 60 min = 1 hour

RUNNING CIRCLES AROUND BLOCKING CALLS

From *Fluent Python*:

Ryan Dahl, the inventor of Node.js, introduces the philosophy of his project by saying "We're doing I/O completely wrong.[4]" He defines a *blocking function* as one that does disk or network I/O, and argues that we can't treat them as we treat nonblocking functions. To explain why, he presents the numbers in the first two columns of Table 18-1.

Table 18-1. Modern computer latency for reading data from different devices; third column shows proportional times in a scale easier to understand for us slow humans

Device	CPU cycles	Proportional "human" scale
L1 cache	3	3 seconds
L2 cache	14	14 seconds
RAM	250	250 seconds
disk	41,000,000	1.3 years
network	240,000,000	7.6 years

4. Video: Introduction to Node.js at 4:55.

ThoughtWorks®

FLAG LAB

A simple HTTP client

RUN THE FLAG DOWNLOAD EXAMPLES

1. Clone: **https://github.com/fluentpython/concurrency**

2. Find the flags*.py examples in the **`lab-flags/`** directory

3. Run:
   - **`flags.py`**
   - **`flags_threadpool.py`**
   - **`flags_await.py`**

We will study the **`flags_await.py`** example in detail later.

ThoughtWorks®

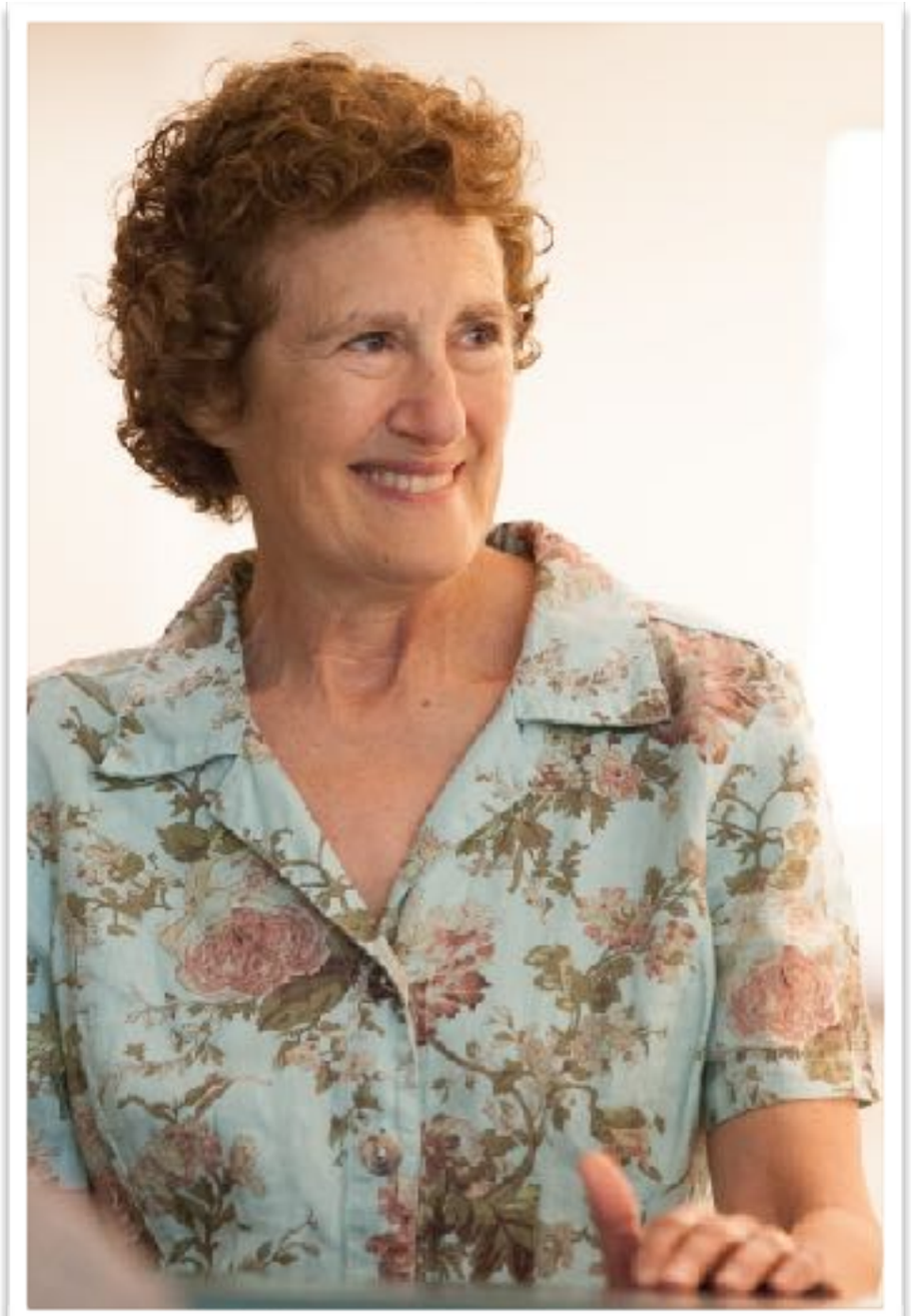
CONCURRENCY DESPITE THE GIL

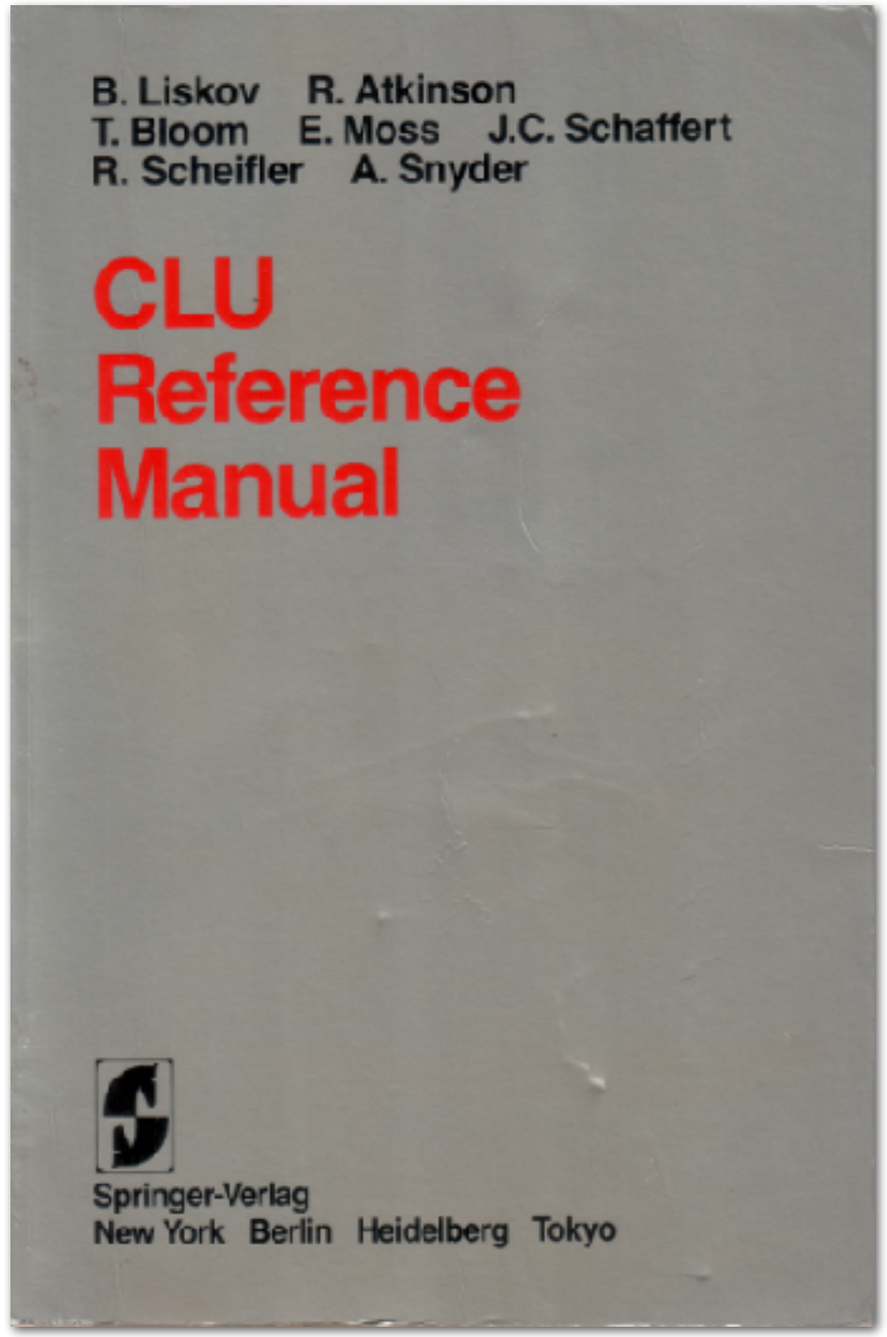
The price of the Global Interpreter Lock

PYTHON ALTERNATIVES

- **Threads**:
  OK for high performance I/O on constrained settings
  - See: Motor 0.7 Beta With Pymongo 2.9 And A Threaded Core
    — A. Jesse Jiryu Davis — https://emptysqua.re/blog/motor-0-7-beta/
- **GIL-releasing threads**:
  some external libraries in Cython, C, C++, FORTRAN...
- **Multiprocessing**: multiple instances of Python
- **Celery** and other distributed task queues
- Callbacks & deferreds in **Twisted**
- **gevent**: greenlets with monkey-patched libraries
- Generators and coroutines in **Tornado** e **Asyncio**

GENERATORS WITH YIELD: WORK BY BARBARA LISKOV

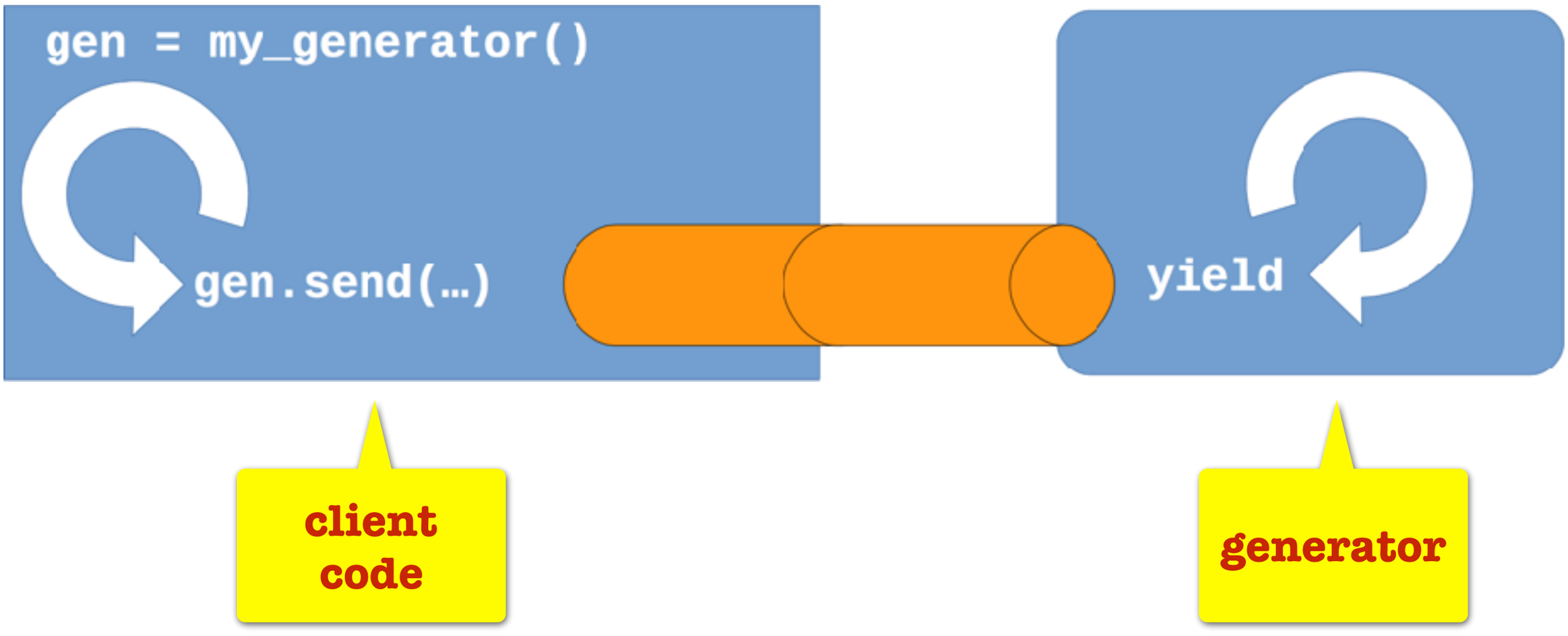




CLU Reference Manual — B. Liskov et. al. — © 1981 Springer-Verlag — also available online from MIT: http://publications.csail.mit.edu/lcs/pubs/pdf/MIT-LCS-TR-225.pdf

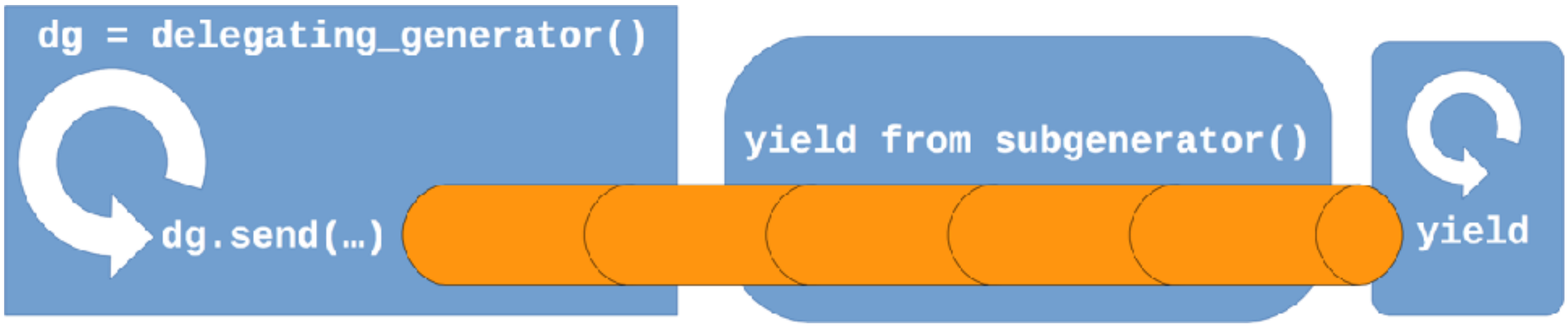
CONCURRENCY WITH COROUTINES (1)

- In Python 2.5 (2006), the modest **generator** was enhanced with a **.send()** method



CONCURRENCY WITH COROUTINES (2)

- In Python 3.3 (2012), the **yield from** syntax allowed a generator to delegate to another generator…



client code

client/ generator, @coroutine

generator

CONCURRENCY WITH COROUTINES (3)

- Finally, in Python 3.5 (2015), **native coroutines** were born

client code

native coroutine

generator, Awaitable

ThoughtWorks®

NATIVE COROUTINES

Better syntax for asynchronous programming

ThoughtWorks®

SPINNER LAB

New takes on an old example

RUN THE SPINNER EXAMPLES

1. Find the examples in the **`lab-spinner/`** directory

2. Run:

- **`spinner_thread.py`**
- **`spinner_asyncio.py`**
- **`spinner_curio.py`**

3. Let's study those examples side by side.

ThoughtWorks®

COUNTDOWN LAB

The countdown sleep experiment

BLOCKING AND THE EVENT LOOP

---

1. Go to the **lab-countdown/** directory

2. Run the **countdown.py** example a few times, noting the interleaving of the counts.

3. Edit the example as described at the top of the source file.

4. Run it again. Discuss the result with your neighbor.

GENERATOR-COROUTINES VS. NATIVE COROUTINES

Kaleidoscope

flags_yield.py | flags_await.py

```
15 import aiohttp  # Ⓐ
16
17 from flags import BASE_URL, save_flag, show, main  # Ⓑ
18
19
20 @asyncio.coroutine  # Ⓒ
21 def get_flag(cc):
22     url = '{}/{cc}/{cc}.gif'.format(BASE_URL, cc=cc.lower())
23     resp = yield from aiohttp.request('GET', url)  # Ⓓ
24     image = yield from resp.read()  # Ⓔ
25     return image
26
27
28 @asyncio.coroutine
29 def download_one(cc):  # Ⓕ
30     image = yield from get_flag(cc)  # Ⓖ
31     show(cc)
32     save_flag(image, cc.lower() + '.gif')
33     return cc
34
35
36 def download_many(cc_list):
37     loop = asyncio.get_event_loop()  # Ⓗ
38     to_do = [download_one(cc) for cc in sorted(cc_list)]  # Ⓘ
39     wait_coro = asyncio.wait(to_do)  # Ⓙ
40     res, _ = loop.run_until_complete(wait_coro)  # Ⓚ
41     loop.close() # Ⓛ
42
43     return len(res)
44
```

```
15 import aiohttp  # Ⓐ
16
17 from flags import BASE_URL, save_flag, show, main  # Ⓑ
18
19
20 async def get_flag(cc):  # Ⓒ
21     url = '{}/{cc}/{cc}.gif'.format(BASE_URL, cc=cc.lower())
22     resp = await aiohttp.request('GET', url)  # Ⓓ
23     image = await resp.read()  # Ⓔ
24     return image
25
26
27 async def download_one(cc):  # Ⓕ
28     image = await get_flag(cc)  # Ⓖ
29     show(cc)
30     save_flag(image, cc.lower() + '.gif')
31     return cc
32
33
34 def download_many(cc_list):
35     loop = asyncio.get_event_loop()  # Ⓗ
36     to_do = [download_one(cc) for cc in sorted(cc_list)]  # Ⓘ
37     wait_coro = asyncio.wait(to_do)  # Ⓙ
38     res, _ = loop.run_until_complete(wait_coro)  # Ⓚ
39     loop.close() # Ⓛ
40
41     return len(res)
42
```

ThoughtWorks®

ASYNC/AWAIT

Where the action is today

THE NEW ASYNC DEF SYNTAX

- **PEP 492**: New keywords introduced in Python 3.5
- **async def** to define *native coroutines*
- **await** to delegate processing to **Awaitable** objects
    - can only be used in native coroutines

- **Awaitable** or *"Future-like"*:
    - Instances of **asyncio.Future** (or **Task**, a subclass of **Future)**
    - native coroutines (**async def...**)
    - generator-coroutines decorated with **@types.coroutine**
    - objects implementing **__await__** (which returns an iterator)

SEQUENTIAL VS. ASYNCHRONOUS (1)

flags_await with coroutines ...

GitHub, Inc. (US) https://github.com/ramalho/tudo-agora/commit/32d92b02883a0b2ade14a716ce137d1e739581b2

Pesquisar

27 countries/flags_await.py

View

```
...  @@ -1,10 +1,10 @@
 1   """Download flags of top 20 countries by population
 2
 3  -Sequential version
 4
 5   Sample run::
 6
 7  -    $ python3 flags_seq.py
 8       BD BR CD CN DE EG ET FR ID IN IR JP MX NG PH PK RU TR US VN
 9       20 flags downloaded in 10.16s
10
     @@ -13,7 +13,8 @@
13   import time
14   import sys
15
16  -import requests
17
18   POP20_CC = ('CN IN US ID BR PK NG BD RU JP '
19               'MX PH VN ET EG DE IR TR CD FR').split()
     @@ -34,23 +35,27 @@ def save_flag(img, filename):
34           fp.write(img)
35
36
37  -def get_flag(cc):
38       url = '{}/{cc}/{cc}.gif'.format(BASE_URL, cc=cc.lower())
39  -    resp = requests.get(url)
```

```
 1   """Download flags of top 20 countries by population
 2
 3  +Asynchronous version
 4
 5   Sample run::
 6
 7  +    $ python3 flags_await.py
 8       BD BR CD CN DE EG ET FR ID IN IR JP MX NG PH PK RU TR US VN
 9       20 flags downloaded in 10.16s
10
13   import time
14   import sys
15
16  +import asyncio
17  +import aiohttp
18
19   POP20_CC = ('CN IN US ID BR PK NG BD RU JP '
20               'MX PH VN ET EG DE IR TR CD FR').split()
35           fp.write(img)
36
37
38  +async def get_flag(cc):
39       url = '{}/{cc}/{cc}.gif'.format(BASE_URL, cc=cc.lower())
40  +    resp = await aiohttp.request('GET', url)
```

SEQUENTIAL VS. ASYNCHRONOUS (2)

flags_await with coroutines ...

GitHub, Inc. (US) https://github.com/ramalho/tudo-agora/commit/32d92b02883a0b2ada14a715ce137d1e739581b2

Pesquisar

```
-def get_flag(cc):
-    url = '{}/{cc}/{cc}.gif'.format(BASE_URL, cc=cc.lower())
-    resp = requests.get(url)
-    return resp.content


-def download_one(cc):
-    image = get_flag(cc)
     show(cc)
     save_flag(image, cc.lower() + '.gif')


 def download_many(cc_list):
-    for cc in sorted(cc_list):
-        download_one(cc)


-    return len(cc_list)


 def main():
```

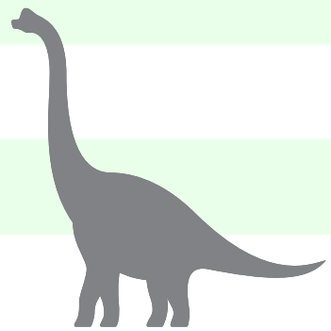
```
+async def get_flag(cc):
+    url = '{}/{cc}/{cc}.gif'.format(BASE_URL, cc=cc.lower())
+    resp = await aiohttp.request('GET', url)
+    image = await resp.read()
+    return image


+async def download_one(cc):
+    image = await get_flag(cc)
     show(cc)
     save_flag(image, cc.lower() + '.gif')


 def download_many(cc_list):
+    loop = asyncio.get_event_loop()
+    task_list = [download_one(cc) for cc in cc_list]
+    super_task = asyncio.wait(task_list)
+    done, _ = loop.run_until_complete(super_task)
+    loop.close()


+    return len(done)


 def main():
```



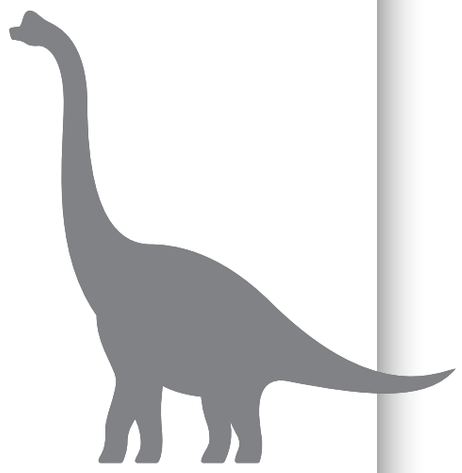
0 comments on commit `32d92b0`

Lock conversation

Write Preview

NATIVE COROUTINES IN ACTION

```
38  async def get_flag(cc):
39      url = '{}/{cc}/{cc}.gif'.format(BASE_URL, cc=cc.lower())
40      resp = await aiohttp.request('GET', url)
41      image = await resp.read()
42      return image
43
44
45  async def download_one(cc):
46      image = await get_flag(cc)
47      show(cc)
48      save_flag(image, cc.lower() + '.gif')
49
50
51  def download_many(cc_list):
52      loop = asyncio.get_event_loop()
53      task_list = [download_one(cc) for cc in cc_list]
54      super_task = asyncio.wait(task_list)
55      done, _ = loop.run_until_complete(super_task)
56      loop.close()
57
58      return len(done)
```



MORE SYNTACTIC SUPPORT

- **PEP 492** also introduced:
  - **async with**:
    invokes asynchronous special methods **__aenter__*** and **__aexit__***
    - ***: coroutines (return **Awaitable** objects)
  - **async for**:
    invokes special methods **__aiter__** e **__anext__***
    - **__aiter__**: not a coroutine, but returns an asynchronous iterator
    - asynchronous integrator implements **__anext__*** as a coroutine

EXAMPLE USING ASYNC WITH AND ASYNC FOR

```
1  import asyncio
2  import aiopg
3
4  dsn = 'dbname=aiopg user=aiopg password=passwd host=127.0.0.1'
5
6  async def go():
7      async with aiopg.create_pool(dsn) as pool:
8          async with pool.acquire() as conn:
9              async with conn.cursor() as cur:
10                 await cur.execute("SELECT 1")
11                 ret = []
12                 async for row in cur:
13                     ret.append(row)
14                 assert ret == [(1,)]
15
16 loop = asyncio.get_event_loop()
17 loop.run_until_complete(go())
```

STILL MORE SYNTACTIC SUPPORT

- New features in **Python 3.6**:
  - **PEP 525:** Asynchronous Generators (!)
  - **PEP 530**: Asynchronous Comprehensions

ASYNC/AWAIT IS NOT JUST FOR ASYNCIO

- In addition to **asyncio**, there are (at least) **curio** and **trio** leveraging native coroutines for asynchronous I/O with very different APIs.
- Brett Cannon's launchpad.py example: native coroutines with a toy event loop in 120 lines using only the packages **time**, **datetime**, **heapq** and **types**.

ThoughtWorks®

ASYNCIO

First use case for yield from

ASYNCIO: FIRST PACKAGE TO LEVERAGE ASYNC/AWAIT

- Package designed by Guido van Rossum (originally: Tulip)
  - added to Python 3.4, provisional status up to Python 3.5: significant API changes
- **asyncio** is no longer provisional in *Python 3.6*
  - most of the API is rather low-level: support for library writers
  - no support for HTTP in the standard library: aiohttp is the most cited add-on
- Very active eco-system
  - see: https://github.com/aio-libs/

GitHub, Inc. [US] https://github.com/aio-libs/

This organization Search Pull requests Issues Gist

aio-libs

The set of asyncio-based libraries built with high quality for humans

https://groups.google.com/forum/#!forum/aio-libs

Repositories People 8

Filters Find a repository...

aiomysql

Python ★ 197 29

aiomysql is a library for accessing a MySQL database from the asyncio

Updated a day ago

aiohttp_admin

Python ★ 24 4

admin interface for aiohttp application

Updated 2 days ago

aiokafka

Python ★ 49 12

asyncio client for kafka

Updated 2 days ago

People 8 >

aiozmq

Python ★ 175 23

Asyncio (pep 3156) integration with ZeroMQ

Updated 5 days ago

aiopg

Python ★ 332 48

aiopg is a library for accessing a PostgreSQL database from the asyncio

Updated 5 days ago

sockjs

Python ★ 53 10

SockJS Server

Updated 5 days ago

multidict

Python ★ 13 3

multidict implementation

Updated 5 days ago

aiobotocore

Python ★ 52 14

asyncio support for botocore library using aiohttp

Updated 5 days ago

aiohttp_debugtoolbar

aioodbc

Python ★ 39 0

aicodbc - is a library for accessing a ODBC databases from the asyncio

Updated 2 days ago

aioredis

Python ★ 196 41

asyncio (PEP 3156) Redis support

Updated 2 days ago

aiohttp_session

Python ★ 39 24

Provide sessions for aiohttp.web

Updated 3 days ago

aiohttp_jinja2

Python ★ 49 19

jinja2 template renderer for aiohttp.web

Updated 3 days ago

yarl

Python ★ 33 4

Yet another URL library

Updated 4 days ago

aiozmq

Python ★ 175 23

aiohttp_debugtoolbar

JavaScript ★ 54 16

aichttp_debugtoolbar is library for debugtoolbar support for aiohttp

Updated 7 days ago

aiohttp_cors

Python ★ 21 6

CORS support for aiohttp

Updated 7 days ago

janus

Python ★ 31 1

Thread-safe asyncio-aware queue

Updated 7 days ago

aiomcache

Python ★ 35 7

Minimal asyncio memcached client

Updated 7 days ago

aiorwlock

Python ★ 17 1

Synchronization primitive RWLock for asyncio (PEP 3156)

Updated 7 days ago

Minimal asyncio memcached client

Updated 7 days ago

aiorwlock

Python ★ 17 1

Synchronization primitive RWLock for asyncio (PEP 3156)

Updated 7 days ago

aiosmtpd

Python ★ 20 5

A reimplementation of the Python stdlib smtpd.py based on asyncio.

Updated 8 days ago

aiohttp_mako

Python ★ 9 2

mako template renderer for aiohttp.web

Updated 8 days ago

Previous 1 2 Next

aiocouchdb

Python ★ 32 8

CouchDB client built on top of aichttp (asyncio)

Updated 25 days ago

async_timeout

Python ★ 5 1

asyncio-compatible timeout class

Updated 28 days ago

sphinxcontrib-asyncio

Python ★ 4 1

Sphinx extension to add asyncio-specific markups

Updated on Apr 15

aioppspp

Python ★ 9 2

IETF PPSP RFC7574 in Python/asyncio

Updated on Feb 3

aiorest

Python ★ 76 9

REST interface for server based on aiohttp (abandoned)

Updated on Apr 30, 2015

PLUGGABLE EVENT LOOP

- **asyncio** includes an event loop
- The **AbstractEventLoopPolicy** API lets us replace the default loop with another implementing **AbstractEventLoop**
  - **AsyncIOMainLoop** implemented by the **Tornado** project
  - An event loop for GUI programming: **Quamash** (PyQt4, PyQt5, PySide)
  - Event loops wrapping the **libuv** library, the highly efficient asynchronous core of Node.js

UVLOOP

- Implemented as Cython bindings for **libuv**
- Written by Yuri Selivanov, who proposed the **async/await** syntax
  - PEP 492 — Coroutines with async and await syntax

USING UVLOOP

uvloop example · ramalho/t...

GitHub, Inc. (US) https://github.com/ramalho/tudo-agora/commit/e1abc22e4e8de8178e2d0e45d3c70c13071301e6

vs code source repo

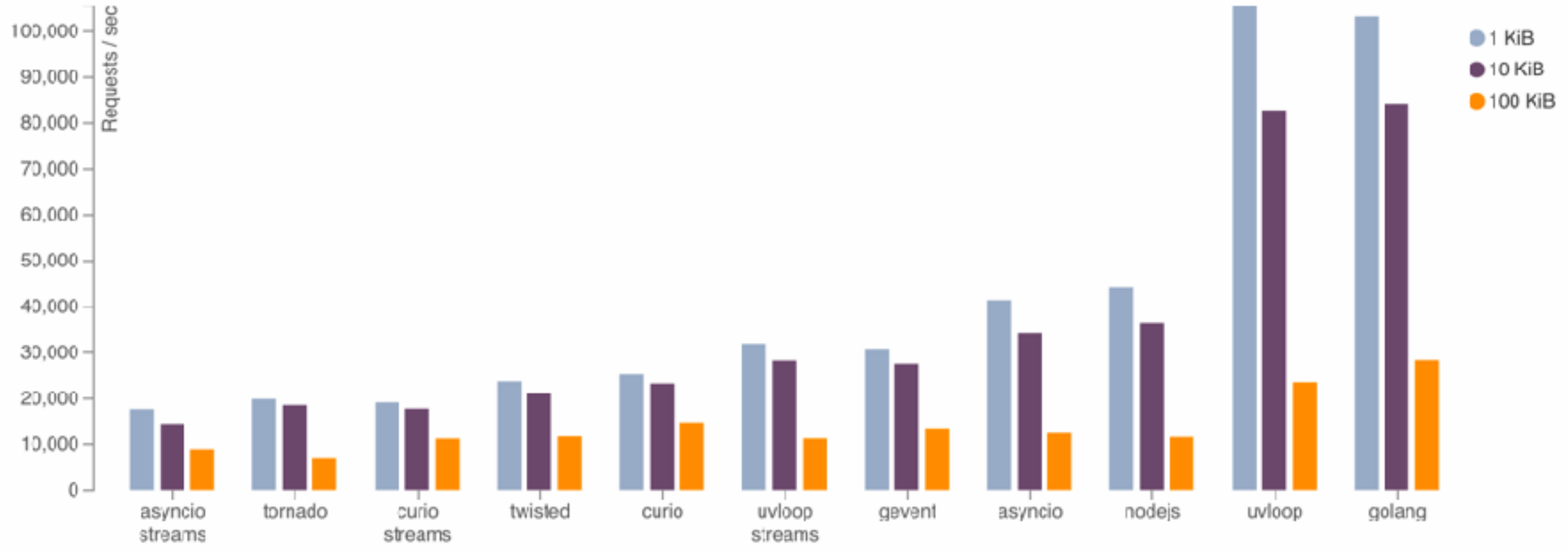
```
...  @@ -1,6 +1,6 @@
1    """Download flags of top 20 countries by population
2
3   -Asynchronous version
4
5    Sample run::
6
...  @@ -15,6 +15,7 @@
15
16   import asyncio
17   import aiohttp
18  +import uvloop
19
20   POP20_CC = ('CN IN US ID BR PK NG BD RU JP '
21               'MX PH VN ET EG DE IR TR CD FR').split()
...  @@ -49,7 +50,8 @@ def save_flag(img, filename):
50
51   def download_many(cc_list):
52  -    loop = asyncio.get_event_loop()
53  +    loop = uvloop.new_event_loop()
54  +    asyncio.set_event_loop(loop)
55       task_list = [download_one(cc) for cc in cc_list]
56       super_task = asyncio.wait(task_list)
57       done, _ = loop.run_until_complete(super_task)
```

UVLOOP PERFORMANCE

TCP

This benchmark tests the performance of a simple echo server with different message sizes. We use 1, 10, and 100 KiB packages. The concurrency level is 10. Each benchmark was run for 30 seconds.

See also the full TCP benchmarks report.



Fonte: **uvloop: Blazing fast Python networking** — Yury Selivanov — 2016-05-03

https://magic.io/blog/uvloop-make-python-networking-great-again/

ThoughtWorks®

WRAPPING UP

The end is near

MY TAKE ON ASYNCIO

- Young ecosystem: libraries evolving fast
  - even trivial examples in Fluent Python now issue warnings or are broken
- **asyncio** with is open for better implementation thanks to its pluggable event loop policy
  - alternative event loops available for a while:
    - Tornado: **AsyncIOMainLoop**
    - QT: **Quamash**
    - libuv: **uvloop** and **pyuv**
- Give **Python 3.6** a try before jumping to Go, Elixir or Node

THE ONE (ABSTRACT) LOOP

One Loop to rule them all, One Loop to find them,
One Loop to bring them all and in liveness bind them.

Fluent Python / Ch. 18: Concurrency with asyncio / Soapbox

FACEBOOK: PYTHON IN PRODUCTION ENGINEERING

Firefox Arquivo Editar Exibir Histórico Favoritos Ferramentas Janela Ajuda 100% Mon 10:34 AM

Python in production engine...

https://code.facebook.com/posts/1040131139381023/python-in-production-engineering/

vs code source repo

Code

Search

Android iOS Web Backend Hardware

27 de maio PRODUCTION ENGINEERING · PYTHON

Python in production engineering

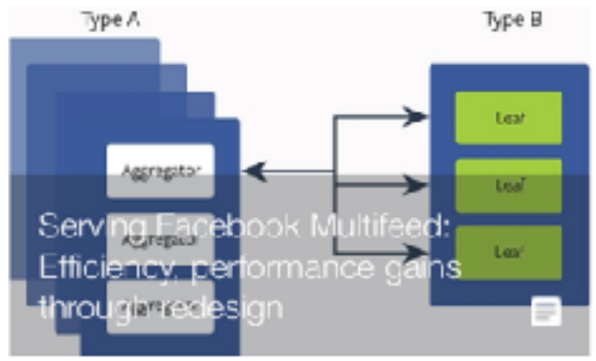
Romain Komorn

Python aficionados are often surprised to learn that Python has long been the language most commonly used by **production engineers** at Facebook and is the third most popular language at Facebook, behind Hack (our in-house dialect of PHP) and C++. Our engineers build and maintain thousands of Python libraries and binaries deployed across our entire infrastructure.

Every day, dozens of Facebook engineers commit code to Python utilities and services with a wide variety of purposes including binary distribution, hardware imaging, operational automation, and infrastructure management.

Python at Facebook by the numbers

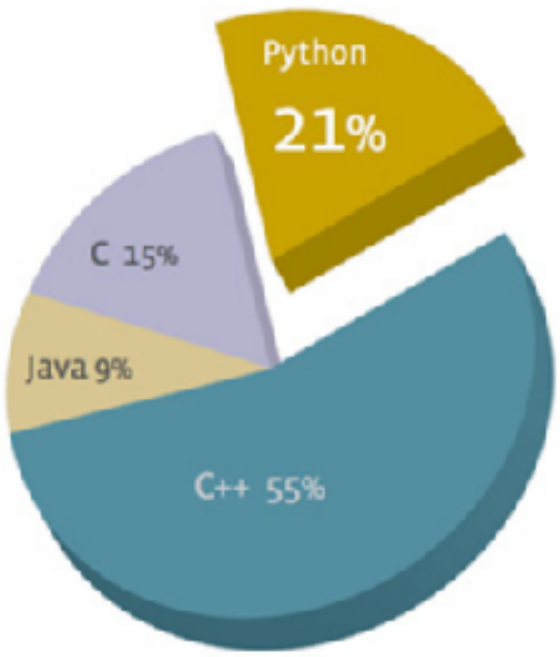
Recommended



PYTHON AT FACEBOOK

Python at Facebook by the numbers

- 21 percent of Facebook Infrastructure's codebase

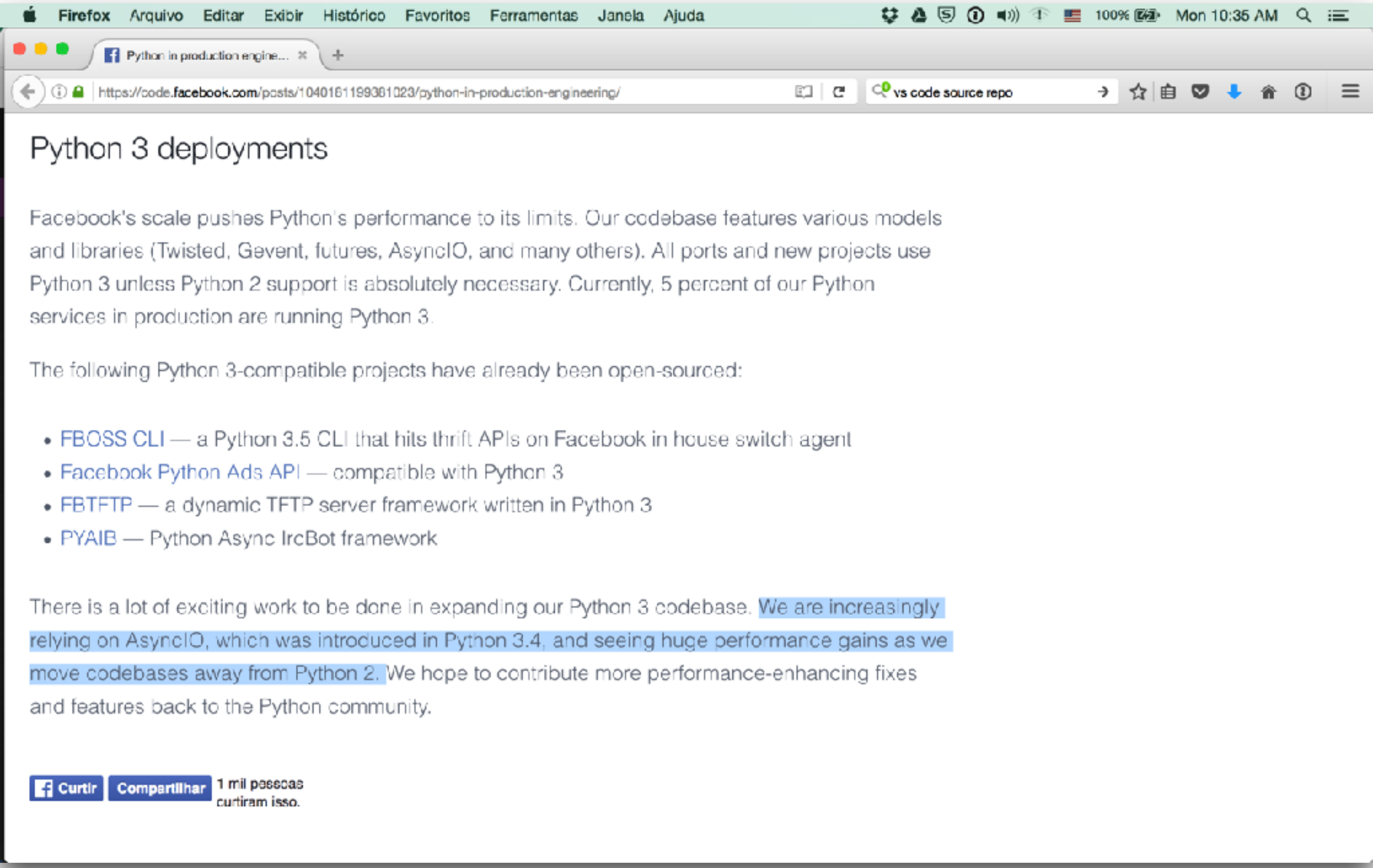


- Millions of lines of code, thousands of libraries and binaries
- 2016 to date: average 5,000 commits per month, 1,000+ committers
- 5 percent Py3 (as of May 2016)

Python in production engineering

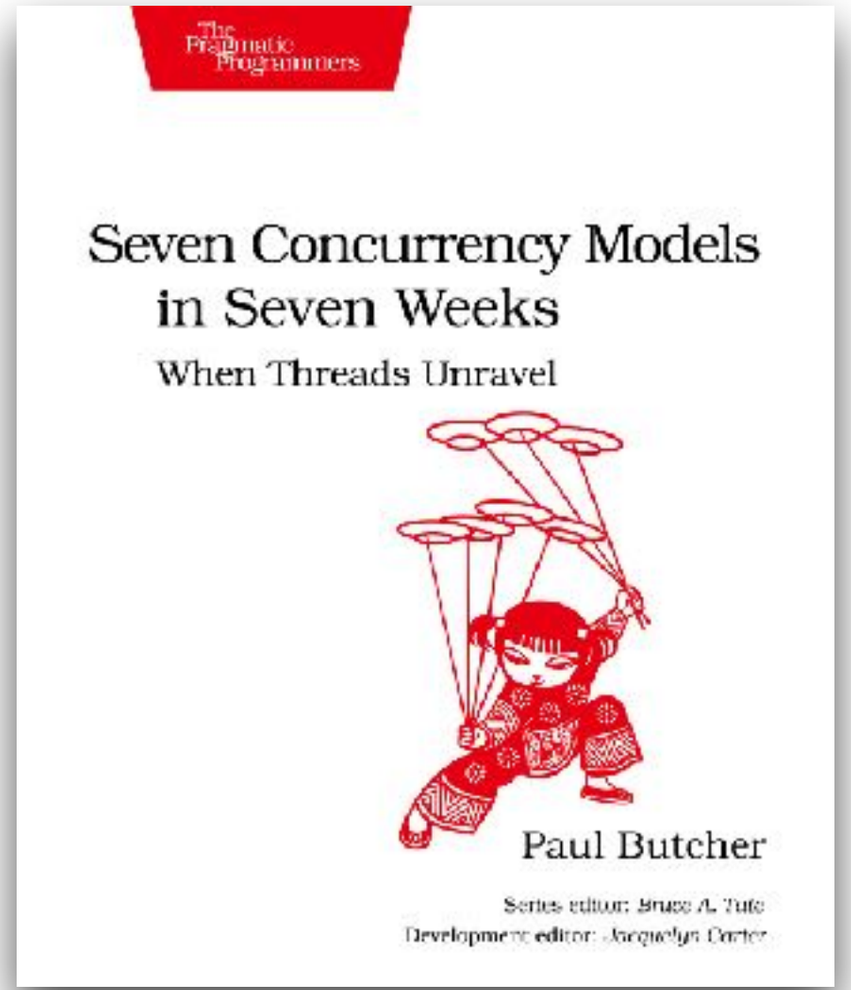
Python is heavily used by the Facebook infrastructure teams and is ubiquitous in production engineering. Teams typically maintain Python client libraries (generally Thrift) for their services,

PYTHON 3 + ASYNCIO AT FACEBOOK



UNRAVELLING THREADS

- *Seven Concurrency Models in Seven Weeks — When Threads Unravel* (Paul Butcher)
- Callbacks are not covered
- Chap. 1: the problem with threads
- Remaining 6 chapters: more powerful abstractions
  - Actors, CSP, STM, data parallelism…
- Native support in languages
  - Erlang, Elixir, Clojure, Go, Cilk, Haskell…



OTHER USES FOR ASYNC/AWAIT

Python's loose coupled introduction of new syntax with semantics based on __special_methods__ allows even more experimentation than new asyncio event loops.

Libraries using async/await with very different APIs:

- David Beazley's **curio**
- Nathaniel…'s **trio**

¿QUESTIONS?

ThoughtWorks®

LUCIANO RAMALHO

Technical Principal

@ramalhoorg
luciano.ramalho@thoughtworks.com

ThoughtWorks®