



*In Python 3.5+*

---

# MODERN CONCURRENCY

---

The new world of `async/await`



*San Francisco  
2017*

AN EXTENDED AND UPDATED VERSION OF...

---

The image is a black and white photograph of a man in a dark tuxedo and bow tie performing a tightrope act. He is balancing on a series of parallel white ropes that are suspended from a metal frame. He is smiling and looking towards the camera. In the background, there is a brick wall and a red banner with the word "oscon" on it. The overall theme of the image is related to concurrency and performance.

CONCURRE  
NCY WITH  
PYTHON  
3.5 ASYNC  
& AWAIT

Presented at OSCON Europe, 2015

ThoughtWorks®

# LUCIANO RAMALHO

---

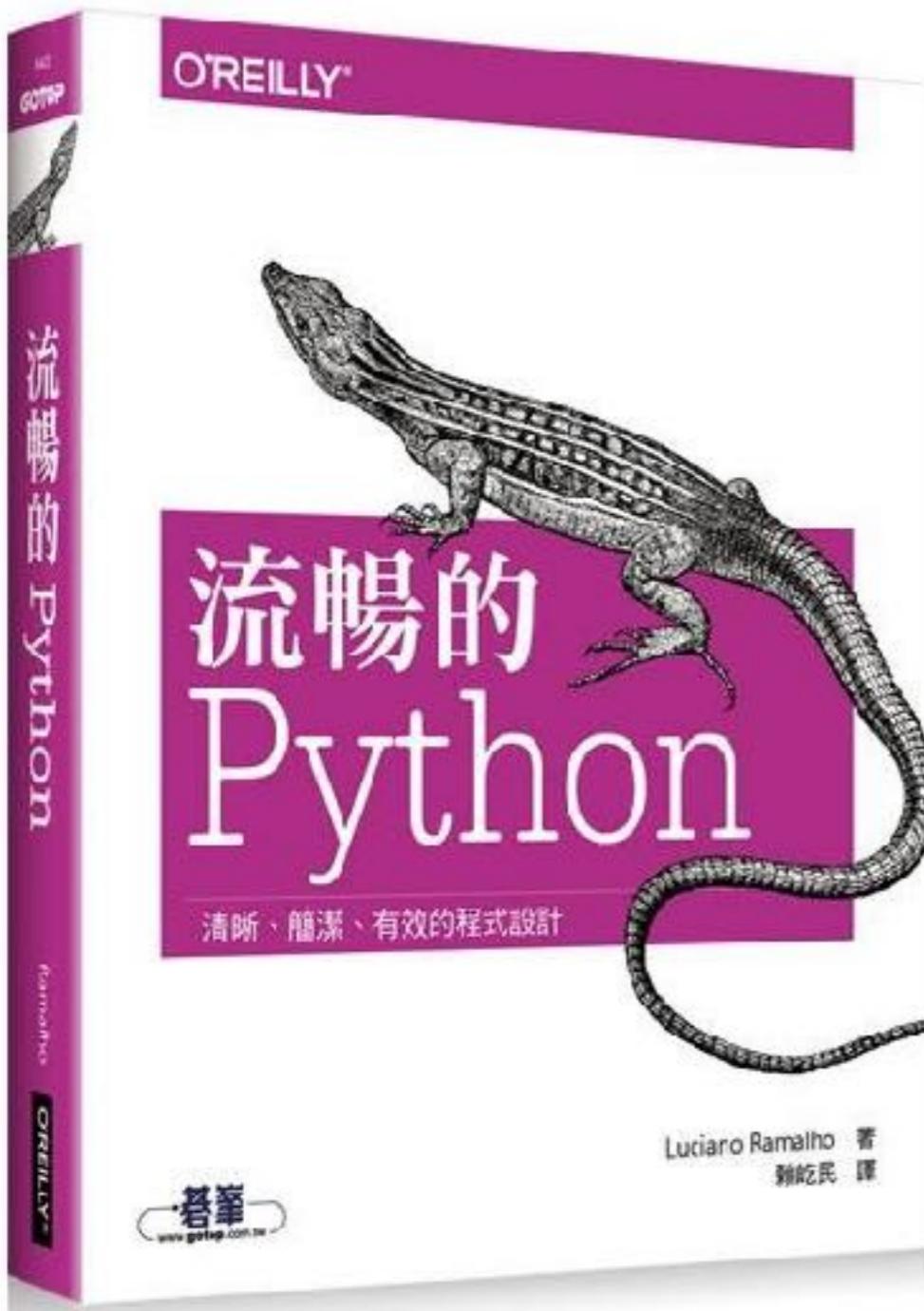
*Technical Principal*

---

@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<sup>†</sup>** (Gotop, 2016)  
also in **Polish, Korean...**

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



# 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](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  
 $24 \times 30 \text{ min} = 720 \text{ min} = 12 \text{ hours!!!}$

# 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  $24 \times 5 \text{ sec} = 120 \text{ sec} = 2 \text{ min}$
- After she completes the round, the first game is ready for her next move!
- 24 games are completed in  $2 \text{ min} \times 30 = 60 \text{ min} = 1 \text{ hour}$

# 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.<sup>4</sup>” 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\*\*](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.

# CONCURRENCY DESPITE THE GIL

---

The price of the Global Interpreter Lock

# PYTHON ALTERNATIVES

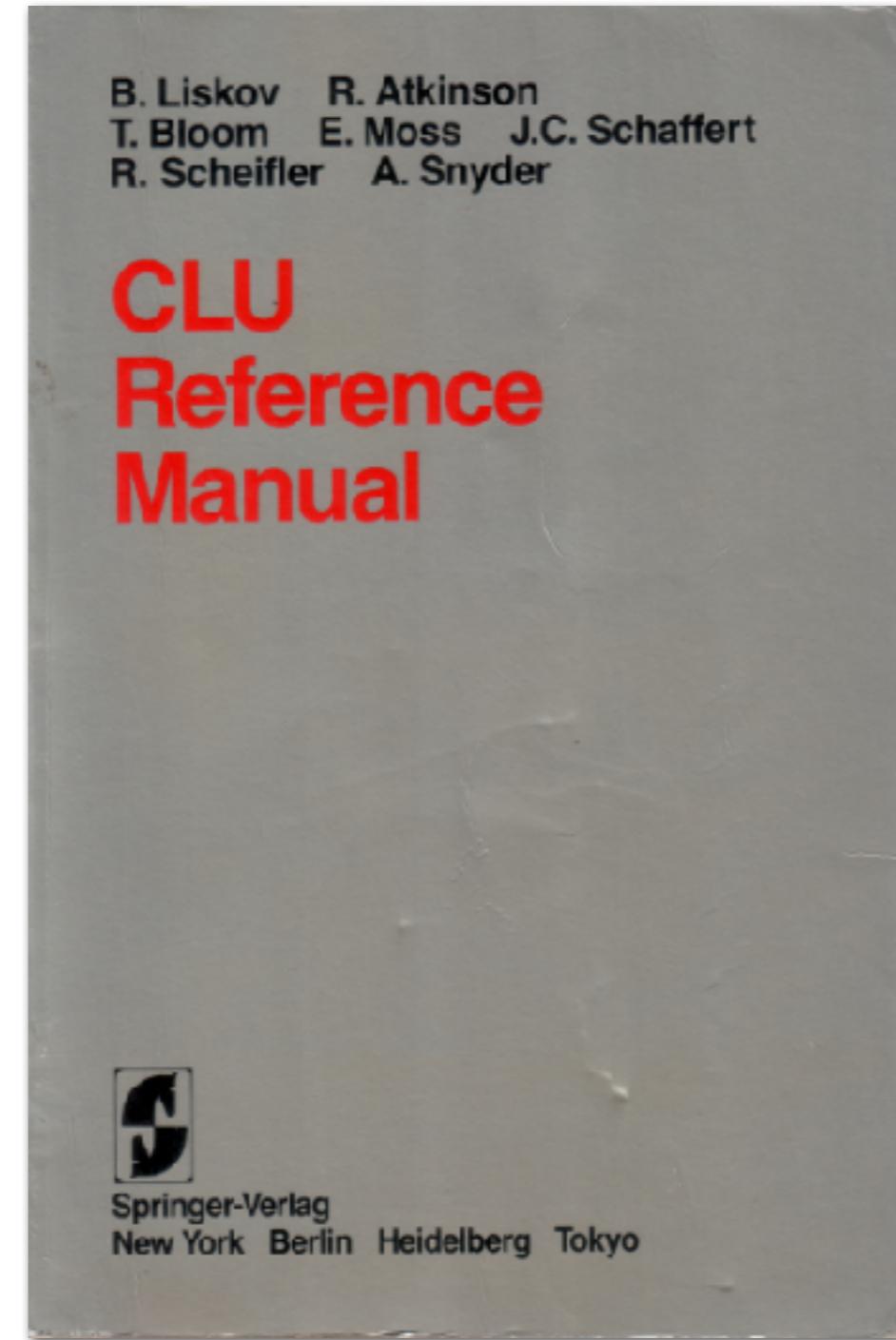
---

- **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

---

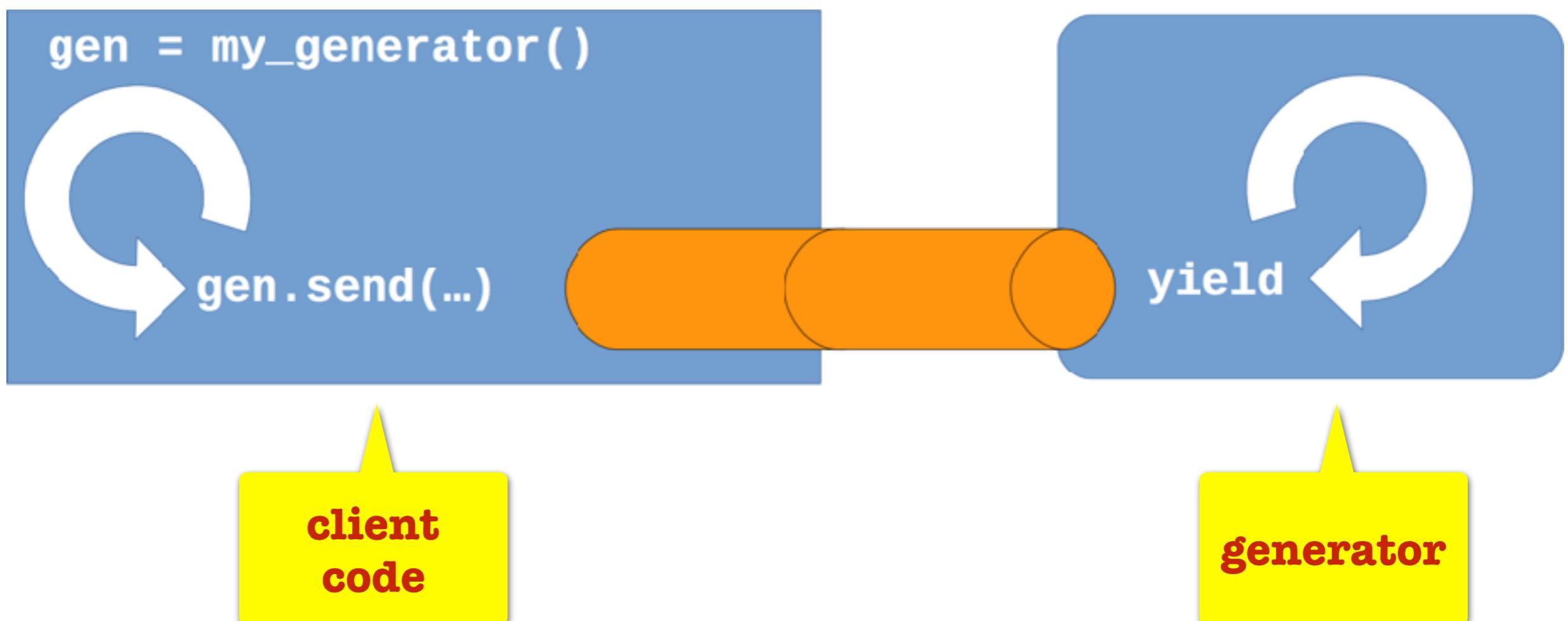
© 2010 Kenneth C. Zirkel — CC-BY-SA



**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>

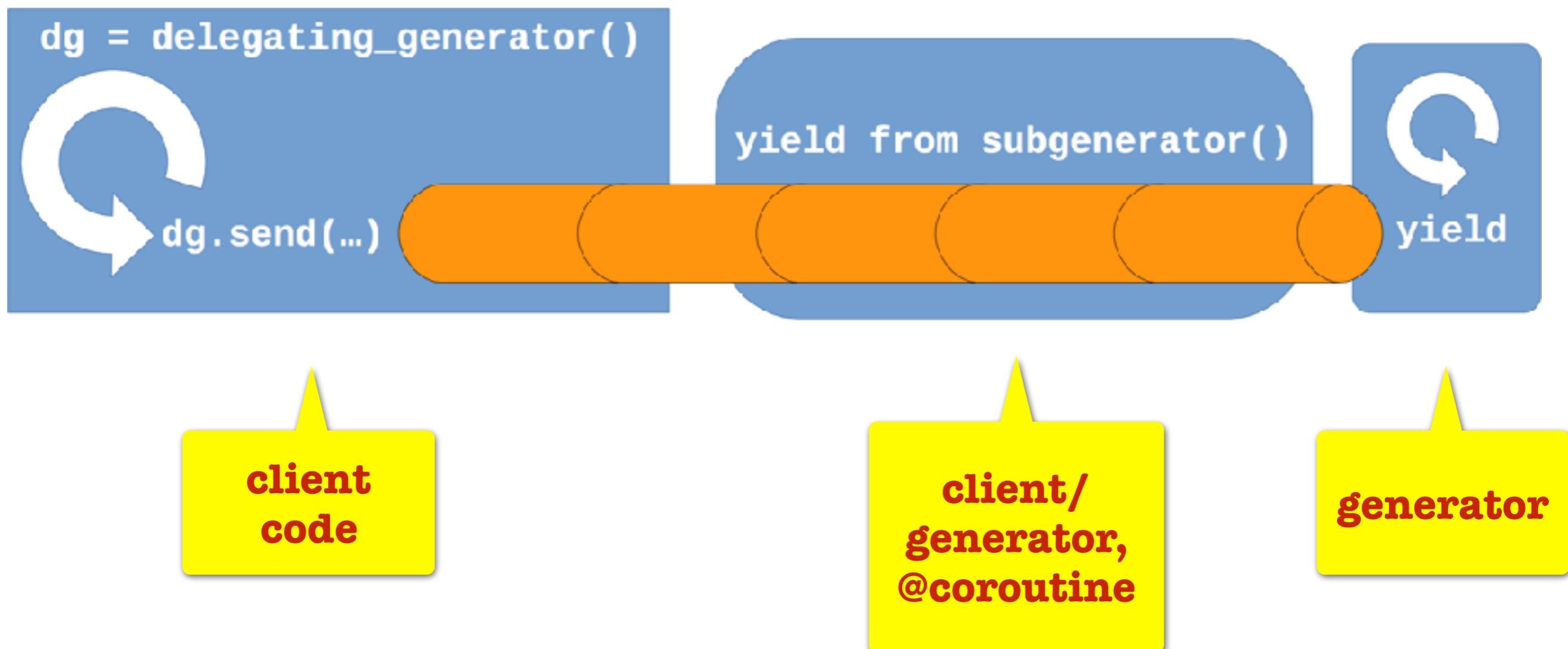
# CONCURRENCY WITH COROUTINES (1)

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



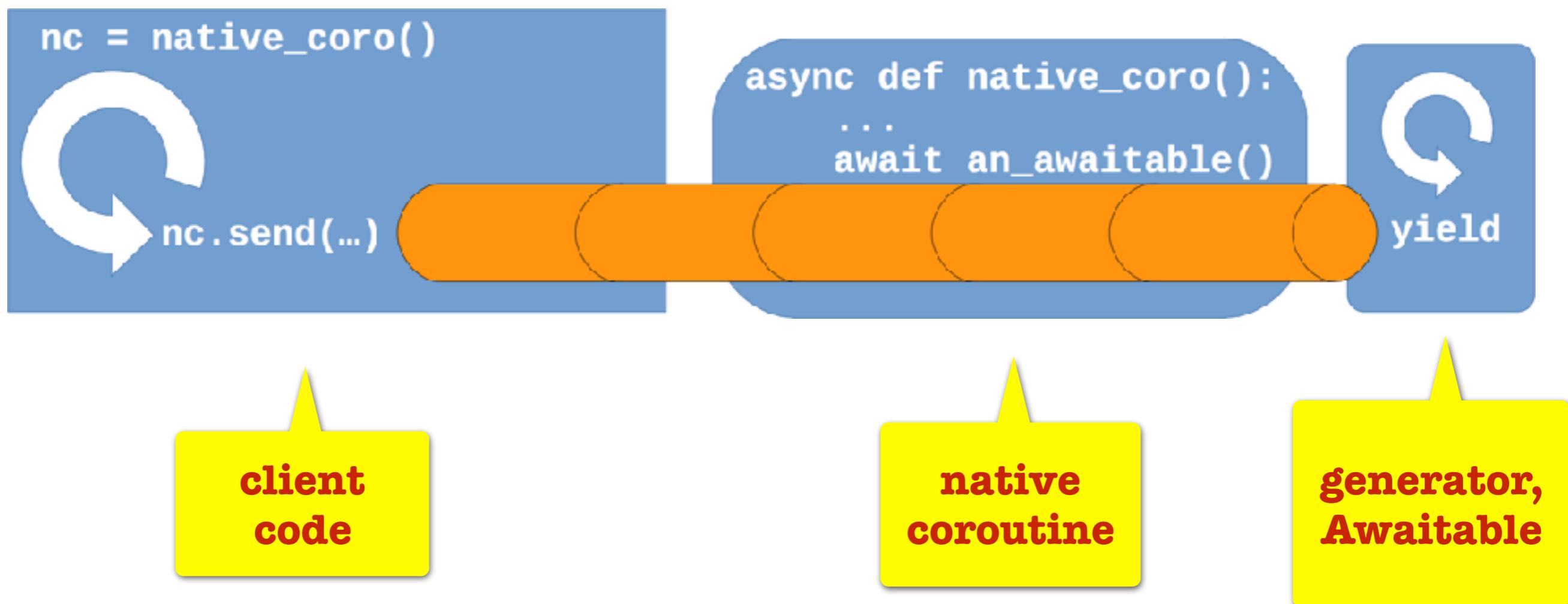
## CONCURRENCY WITH COROUTINES (2)

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



# CONCURRENCY WITH COROUTINES (3)

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



# 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

```
15 import aiohttp # ①
16
17 from flags import BASE_URL, save_flag, show, main # ②
18
19
20 @asyncio.coroutine # ③
21 def get_flag(cc):#
22     url = '{}/{cc}/{cc}.gif'.format(BASE_URL, cc=cc.lower())
23     resp = yield from aiohttp.request('GET', url) # ④
24     image = yield from resp.read() # ⑤
25     return image
26
27
28 @asyncio.coroutine
29 def download_one(cc): # ⑥
30     image = yield from get_flag(cc) # ⑦
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() # ⑧
38     to_do = [download_one(cc) for cc in sorted(cc_list)] # ⑨
39     wait_coro = asyncio.wait(to_do) # ⑩
40     res, _ = loop.run_until_complete(wait_coro) # ⑪
41     loop.close() # ⑫
42
43     return len(res)
44
```

```
15 import aiohttp # ①
16
17 from flags import BASE_URL, save_flag, show, main # ②
18
19
20 async def get_flag(cc): # ③
21     url = '{}/{cc}/{cc}.gif'.format(BASE_URL, cc=cc.lower())
22     resp = await aiohttp.request('GET', url) # ④
23     image = await resp.read() # ⑤
24     return image
25
26
27 async def download_one(cc): # ⑥
28     image = await get_flag(cc) # ⑦
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() # ⑧
36     to_do = [download_one(cc) for cc in sorted(cc_list)] # ⑨
37     wait_coro = asyncio.wait(to_do) # ⑩
38     res, _ = loop.run_until_complete(wait_coro) # ⑪
39     loop.close() # ⑫
40
41     return len(res)
42
```



# 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)

# SEQUENTIAL VS. ASYNCHRONOUS (2)

flags\_await with coroutines ... +

GitHub, Inc. (US) | https://github.com/ramalho/tudo-agora/commit/32d92b02883a0b2ada14a718ce137d1e739581b2 | C Pesquisar

37 -def get\_flag(cc):  
38 url = '{}/{cc}/{cc}.gif'.format(BASE\_URL, cc=cc.lower())  
39 - resp = requests.get(url)  
40 - return resp.content  
41  
42  
43 -def download\_one(cc):  
44 - image = get\_flag(cc)  
45 show(cc)  
46 save\_flag(image, cc.lower() + '.gif')  
47  
48  
49 def download\_many(cc\_list):  
50 - for cc in sorted(cc\_list):  
51 - download\_one(cc)  
52  
53 - return len(cc\_list)  
54  
55  
56 def main():

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)  
59  
60  
61 def main():



**0 comments on commit 32d92b0**

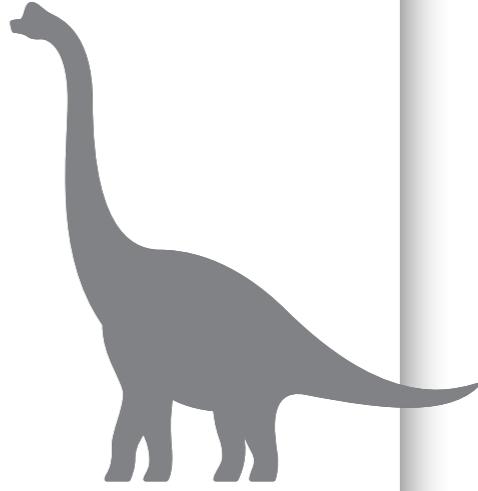


## Write

Preview

# NATIVE COROUTINES IN ACTION

```
38     async def get_flag(cc):
39         url = '{}/{}/{}/{}.gif'.format(BASE_URL, 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/>



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

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

Updated a day ago

Python ★ 197 ⚡ 29

## aiohttp\_admin

admin interface for aichttp application

Updated 2 days ago

Python ★ 24 ⚡ 4

## aiokafka

asyncio client for kafka

Updated 2 days ago

Python ★ 49 ⚡ 12

## People

8 >





## aiozmq

Python ★ 175 ⚡ 23

Asyncio (pep 3156) integration with ZeroMQ

Updated 5 days ago

## aiopg

Python ★ 332 ⚡ 48

aicpg 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

## alobotocore

Python ★ 52 ⚡ 14

asyncio support for botocore library using aiohttp

Updated 5 days ago

## aiohttp\_debugtoolbar

JavaScript ★ 54 ⚡ 16



## 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

aichhttp\_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 aichhttp (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/tudo-agora

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
7 @@ -15,6 +15,7 @@
8
9 import asyncio
10 import aiohttp
11
12 POP20_CC = ('CN IN US ID BR PK NG BD RU JP '
13             'MX PH VN ET EG DE IR TR CD FR').split()
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51 def download_many(cc_list):
52 -    loop = asyncio.get_event_loop()
53
54     task_list = [download_one(cc) for cc in cc_list]
55     super_task = asyncio.wait(task_list)
56
57     done, _ = loop.run_until_complete(super_task)
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
517
518
519
519
520
521
522
523
524
525
526
527
527
528
529
529
530
531
532
533
534
535
536
537
537
538
539
539
540
541
542
543
544
545
546
547
547
548
549
549
550
551
552
553
554
555
556
557
557
558
559
559
560
561
562
563
564
565
566
567
567
568
569
569
570
571
572
573
574
575
576
577
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
638
639
639
640
641
642
643
644
645
646
647
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
677
678
679
679
680
681
682
683
684
685
686
687
688
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
716
717
718
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
738
739
739
740
741
742
743
744
745
746
747
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
768
769
769
770
771
772
773
774
775
776
777
777
778
779
779
780
781
782
783
784
785
786
787
788
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
838
839
839
840
841
842
843
844
845
846
847
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
868
869
869
870
871
872
873
874
875
876
877
877
878
879
879
880
881
882
883
884
885
886
887
888
888
889
889
890
891
892
893
894
895
896
897
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
916
917
918
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
938
939
939
940
941
942
943
944
945
946
947
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
967
968
969
969
970
971
972
973
974
975
976
977
977
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
996
997
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1016
1017
1018
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1096
1097
1098
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1106
1107
1108
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1116
1117
1118
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1126
1127
1128
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1136
1137
1138
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1146
1147
1148
1148
1149
1149
1150
1151
1152
1153
1154
1155
1156
1156
1157
1158
1158
1159
1159
1160
1161
1162
1163
1164
1165
1165
1166
1167
1167
1168
1168
1169
1169
1170
1171
1172
1173
1174
1175
1175
1176
1177
1177
1178
1178
1179
1179
1180
1181
1182
1183
1184
1185
1185
1186
1187
1187
1188
1188
1189
1189
1190
1191
1192
1193
1194
1194
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1201
1202
1203
1204
1204
1205
1206
1206
1207
1207
1208
1209
1209
1210
1210
1211
1212
1213
1214
1215
1215
1216
1217
1217
1218
1218
1219
1219
1220
1221
1222
1223
1224
1225
1225
1226
1227
1227
1228
1228
1229
1229
1230
1231
1232
1233
1234
1235
1235
1236
1237
1237
1238
1238
1239
1239
1240
1241
1242
1243
1244
1245
1245
1246
1247
1247
1248
1248
1249
1249
1250
1251
1252
1253
1254
1255
1255
1256
1257
1257
1258
1258
1259
1259
1260
1261
1262
1263
1264
1265
1265
1266
1267
1267
1268
1268
1269
1269
1270
1271
1272
1273
1274
1275
1275
1276
1277
1277
1278
1278
1279
1279
1280
1281
1282
1283
1284
1285
1285
1286
1287
1287
1288
1288
1289
1289
1290
1291
1292
1293
1294
1294
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1301
1302
1303
1304
1304
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1311
1312
1313
1314
1314
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1321
1322
1323
1324
1324
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1331
1332
1333
1334
1334
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1341
1342
1343
1344
1344
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1351
1352
1353
1354
1354
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1361
1362
1363
1364
1364
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1371
1372
1373
1374
1374
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1381
1382
1383
1384
1384
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1391
1392
1393
1394
1394
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1401
1402
1403
1404
1404
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1411
1412
1413
1414
1414
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1421
1422
1423
1424
1424
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1431
1432
1433
1434
1434
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1441
1442
1443
1444
1444
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1451
1452
1453
1454
1454
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1461
1462
1463
1464
1464
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1471
1472
1473
1474
1474
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1481
1482
1483
1484
1484
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1491
1492
1493
1494
1494
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1501
1502
1503
1504
1504
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1511
1512
1513
1514
1514
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1521
1522
1523
1524
1524
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1531
1532
1533
1534
1534
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1541
1542
1543
1544
1544
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1551
1552
1553
1554
1554
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1561
1562
1563
1564
1564
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1571
1572
1573
1574
1574
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1581
1582
1583
1584
1584
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1591
1592
1593
1594
1594
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1601
1602
1603
1604
1604
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1611
1612
1613
1614
1614
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1621
1622
1623
1624
1624
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1631
1632
1633
1634
1634
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1641
1642
1643
1644
1644
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1651
1652
1653
1654
1654
1655
1656
1656
1657
1657
1658
1658
1659
1659
1660
1661
1662
1663
1664
1664
1665
1666
1666
1667
1667
1668
1668
1669
1669
1670
1671
1672
1673
1674
1674
1675
1676
1676
1677
1677
1678
1678
1679
1679
1680
1681
1682
1683
1684
1684
1685
1686
1686
1687
1687
1688
1688
1689
1689
1690
1691
1692
1693
1694
1694
1695
1696
1696
1697
1697
1698
1698
1699
1699
1700
1701
1702
1703
1704
1704
1705
1706
1706
1707
1707
1708
1708
1709
1709
1710
1711
1712
1713
1714
1714
1715
1716
1716
1717
1717
1718
1718
1719
1719
1720
1721
1722
1723
1724
1724
1725
1726
1726
1727
1727
1728
1728
1729
1729
1730
1731
1732
1733
1734
1734
1735
1736
1736
1737
1737
1738
1738
1739
1739
1740
1741
1742
1743
1744
1744
1745
1746
1746
1747
1747
1748
1748
1749
1749
1750
1751
1752
1753
1754
1754
1755
1756
1756
1757
1757
1758
1758
1759
1759
1760
1761
1762
1763
1764
1764
1765
1766

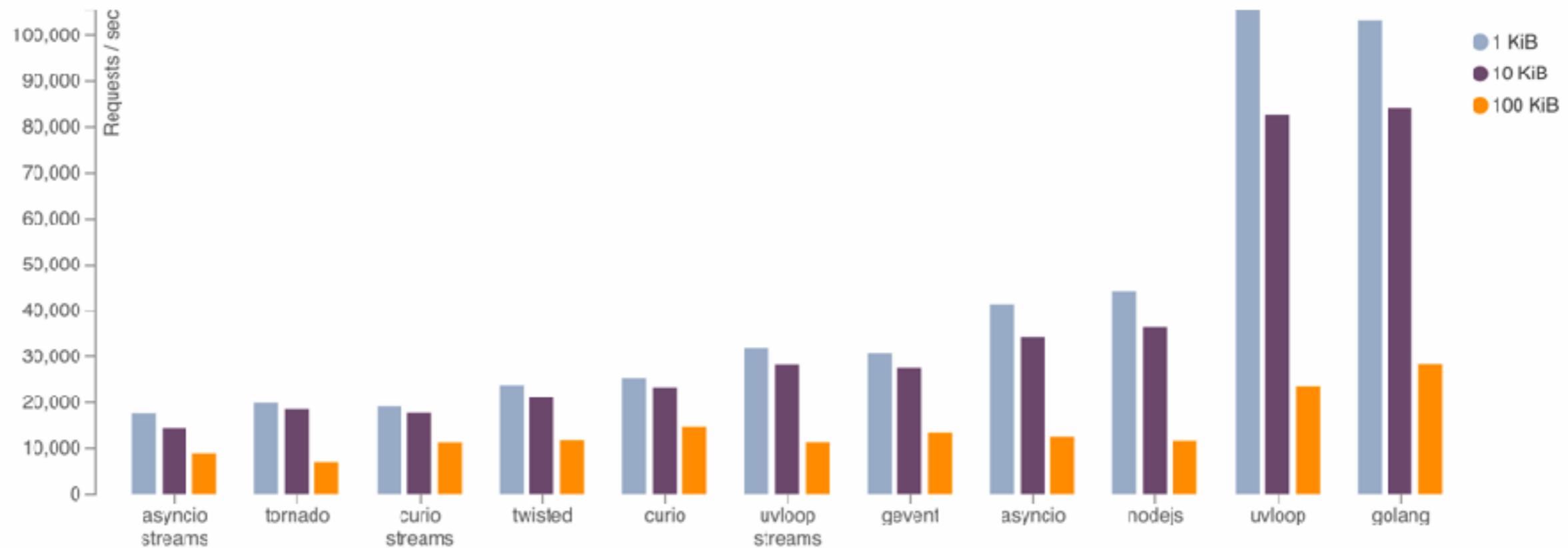
```

# 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/>



# 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.*

# 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/poets/1040181139381023/python-in-production-engineering/ vs code source repo

f 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.

### Recommended

Type A

Aggregator

Serving Facebook Multifeed: Efficiency, performance gains through redesign

Type B

Leaf

Leaf

Leaf

The diagram illustrates the transition from a monolithic architecture (Type A) to a distributed architecture (Type B) for serving the Facebook Multifeed. In Type A, a single 'Aggregator' box handles all incoming requests. In Type B, the aggregator is replaced by a 'Serving Facebook Multifeed' box, which distributes traffic to multiple 'Leaf' boxes. This redesign aims for efficiency and performance gains.

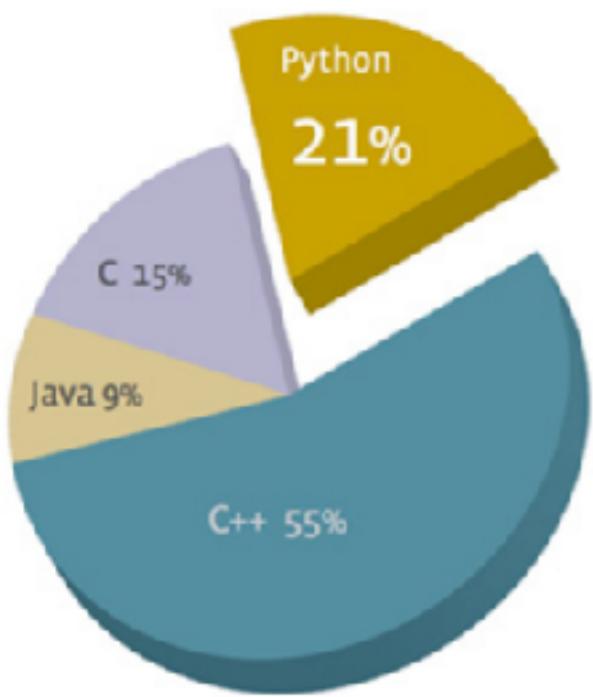
## Python at Facebook by the numbers

# PYTHON AT FACEBOOK

A screenshot of a Firefox browser window. The title bar shows 'Firefox' and various menu options. The address bar displays 'Python in production engine...' and the URL 'https://code.facebook.com/posts/1040181199381023/python-in-production-engineering/'. The main content area shows a title 'Python at Facebook by the numbers' and a pie chart illustrating the distribution of codebase by language. A sidebar on the right contains a 'SECURITY @ SCALE' section for 'Security @Scale 2015: Engineering Security'.

## 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

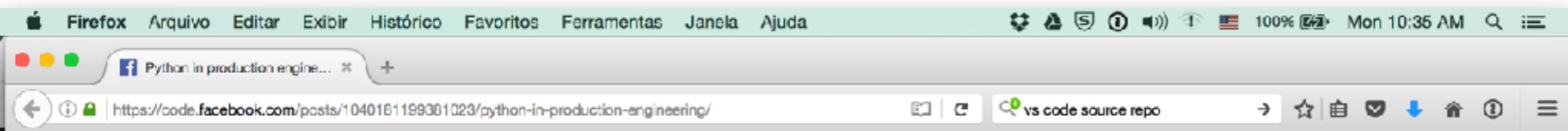
Python is heavily used by the Facebook infrastructure teams and is ubiquitous in production engineering. Teams typically maintain Python client libraries (generally Thrift) for their services,

A sidebar for 'Security @ Scale 2015: Engineering Security'. It features a large 'SSH' logo and the text 'Scalable and secure access with SSH'.

A sidebar for 'Security @ Scale 2015: Engineering Security'. It features a large 'SSH' logo and the text 'Scalable and secure access with SSH'.

A sidebar for 'Security @ Scale 2015: Engineering Security'. It features a large 'Apache Hive' logo and the text 'Join Optimization in Apache Hive'.

# PYTHON 3 + ASYNCIO AT FACEBOOK



## Python 3 deployments

Facebook's scale pushes Python's performance to its limits. Our codebase features various models and libraries (Twisted, Gevent, futures, AsyncIO, and many others). All ports and new projects use Python 3 unless Python 2 support is absolutely necessary. Currently, 5 percent of our Python services in production are running Python 3.

The following Python 3-compatible projects have already been open-sourced:

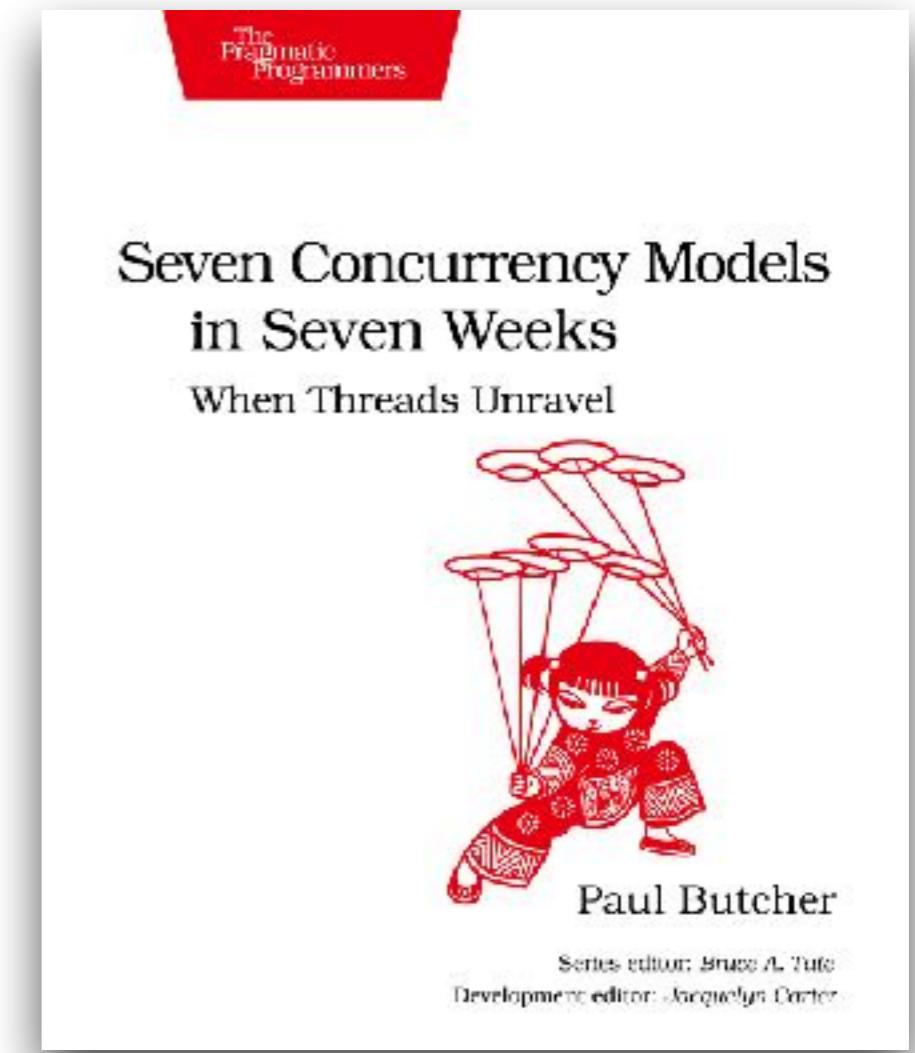
- [FBOSS CLI](#) — a Python 3.5 CLI that hits thrift APIs on Facebook in house switch agent
- [Facebook Python Ads API](#) — compatible with Python 3
- [FBTFTP](#) — a dynamic TFTP server framework written in Python 3
- [PYAIB](#) — Python Async IrcBot framework

There is a lot of exciting work to be done in expanding our Python 3 codebase. We are increasingly relying on AsyncIO, which was introduced in Python 3.4, and seeing huge performance gains as we move codebases away from Python 2. We hope to contribute more performance-enhancing fixes and features back to the Python community.

# 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](mailto:luciano.ramalho@thoughtworks.com)

ThoughtWorks®