现代方式使用asyncio

Kevin

3x faster Flask apps

Quart as a upgrade to Flask

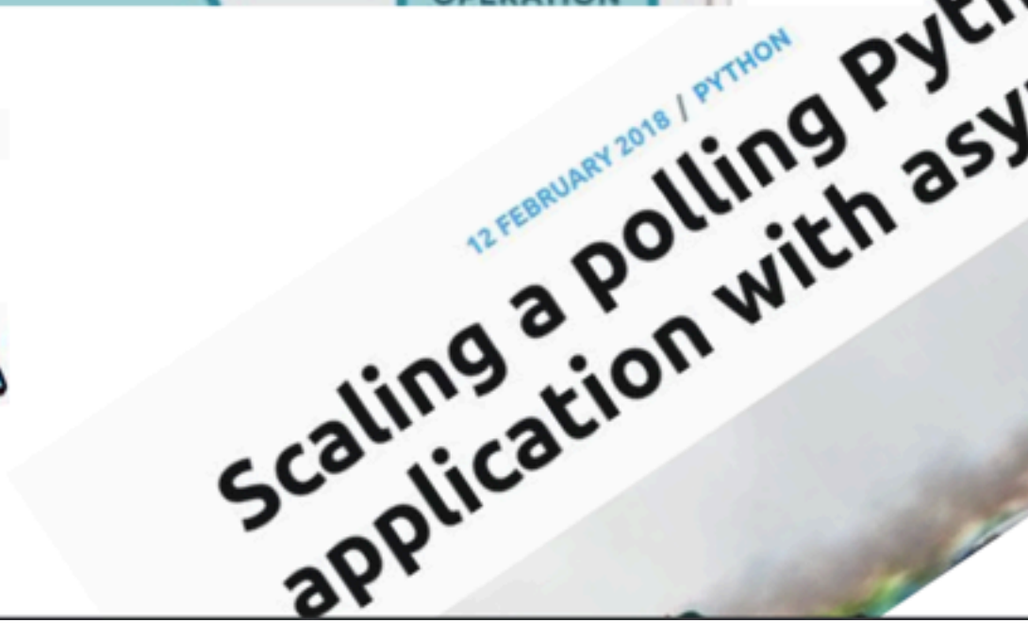
Python has evolved since Flask was

Making 1
aiohttp

Apr 22, 2016 - by Pawel

/ue

python



asyncio

- 为什么我们需要异步?
- 为什么选择asyncio?
- 如何优雅的编写asyncio程序?
- asyncio的其他

asyncio

- ~~为什么我们需要异步?~~
- 为什么选择asyncio?
- 如何优雅的编写asyncio程序?
- asyncio的其他

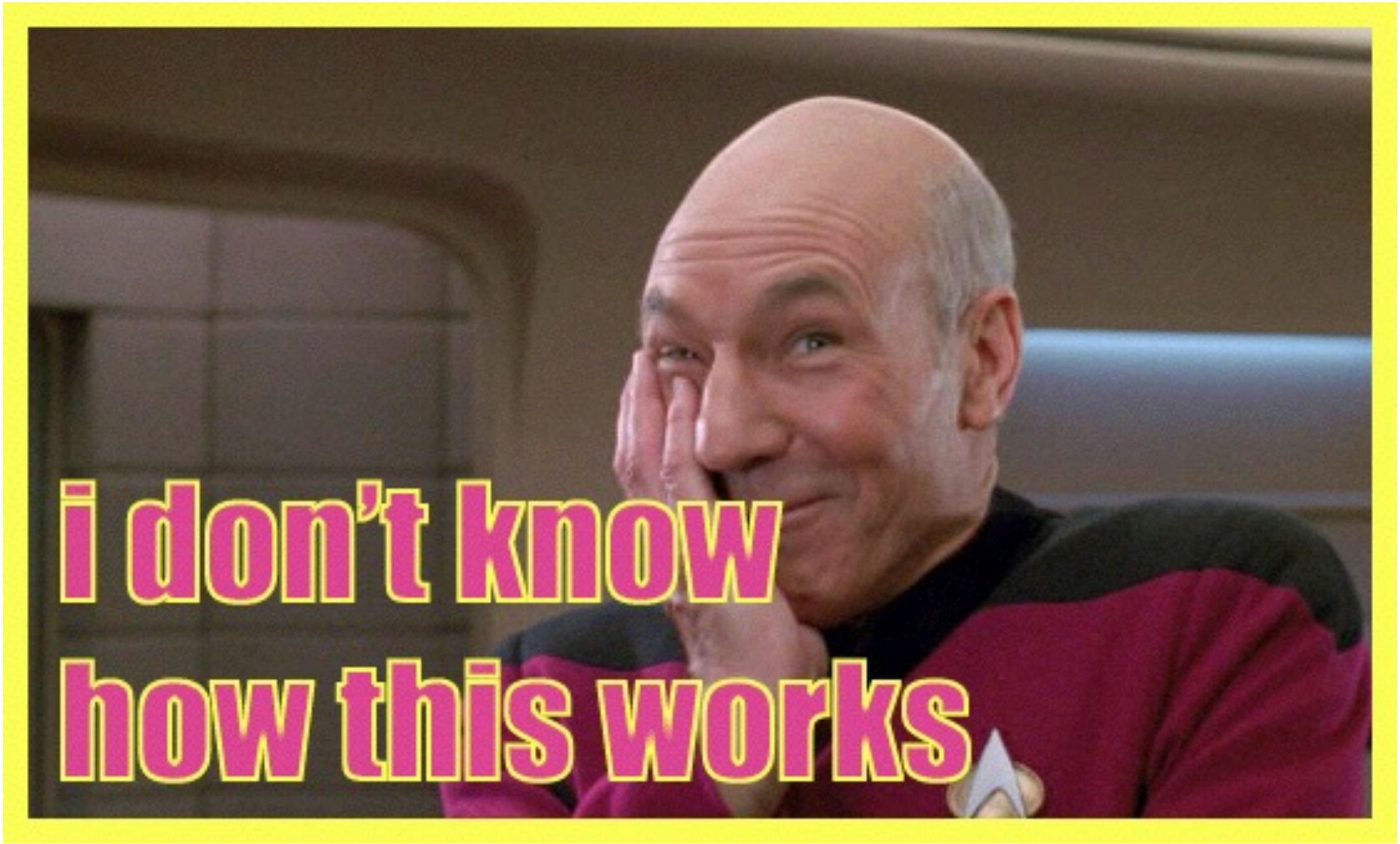
asyncio

- 为什么我们需要异步?
- 为什么选择asyncio?
- 如何优雅的编写asyncio程序?
- asyncio的其他

gevent/eventlet

```
1   import gevent
2
3   gevent.monkey.patch_all()
4
5   import urllib.request
6
7   f = urllib.request.urlopen("http://v                    )
8   print(f.read(100).decode("utf-8"))
```

gevent/eventlet



相比gevent

- 官方支持，未来的方向
- 显式切换
- 显式处理条件竞争问题更容易
- 显式控制上下文切换，不再踩坑
- 更容易兼容

asyncio

- 为什么我们需要异步?
- 为什么选择asyncio?
- 如何优雅的编写asyncio程序?
- asyncio的其他

举个例子

```
import asyncio



async def main():
    await asyncio.sleep(1)



loop = asyncio.get_event_loop()
try:
    loop.run_until_complete(main())
finally:
    loop.close()
```

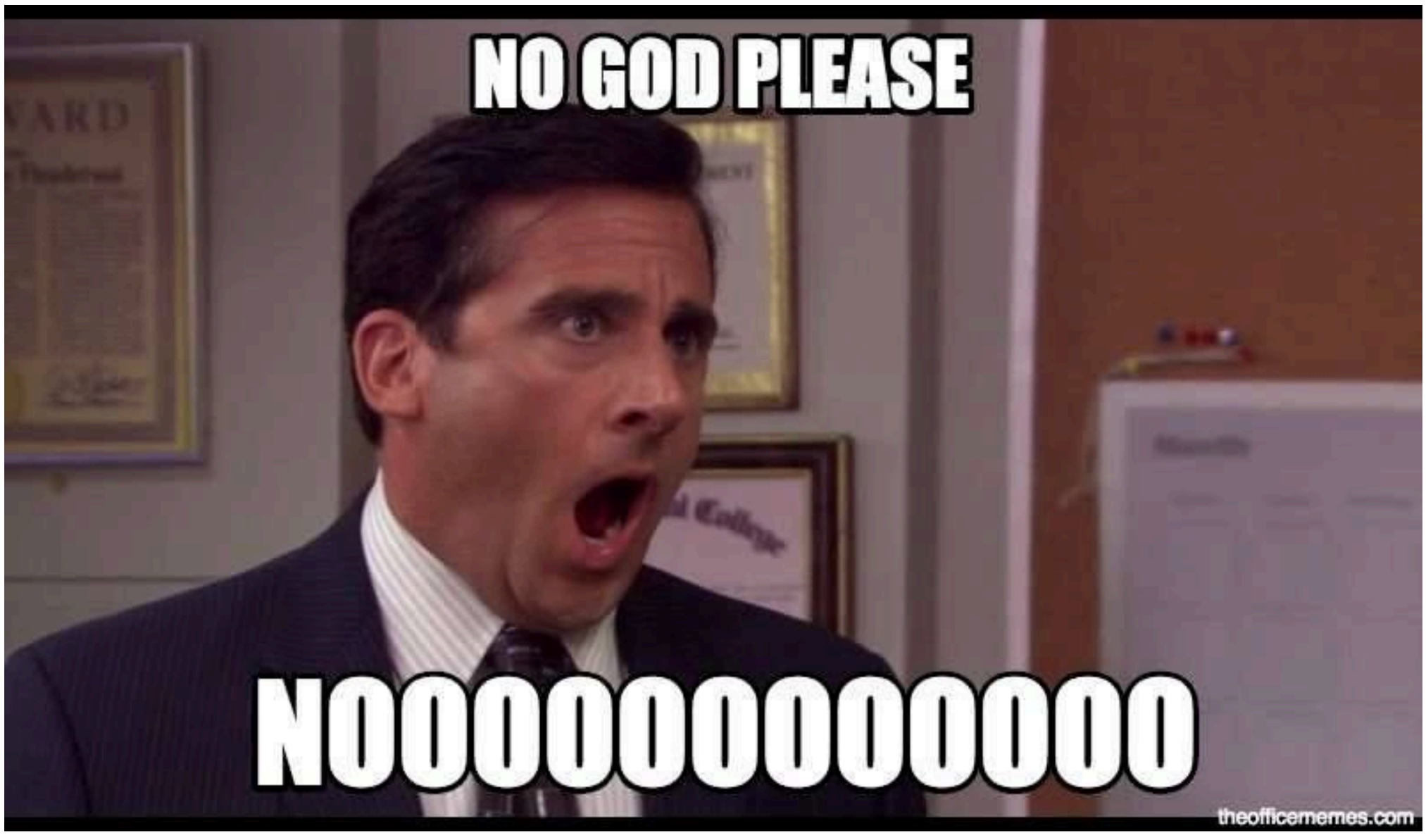
举个例子

```
import asyncio


async def echo(r, w):
    data = await r.read(100)
    w.write(data)
    await w.drain()


loop = asyncio.get_event_loop()
server = loop.run_until_complete(
    asyncio.start_server(echo, "127.0.0.1", 8000, loop=loop)
)
try:
    loop.run_forever()
except KeyboardInterrupt:
    pass
server.close()
loop.run_until_complete(server.wait_closed())
loop.close()
```

举个例子



run&serve_forever

```
import asyncio


async def echo(r, w):
    data = await r.read(100)
    w.write(data)
    await w.drain()


async def main():
    server = asyncio.start_server(echo, "127.0.0.1", 8000)
    async with server:
        server.serve_forever()


asyncio.run(main())
```

asyncio API

asyncio.run()

asyncio.gather()

asyncio.create_task()

asyncio.sleep()

streams API

normal

---

hardcore

loop.*()

protocols & transports

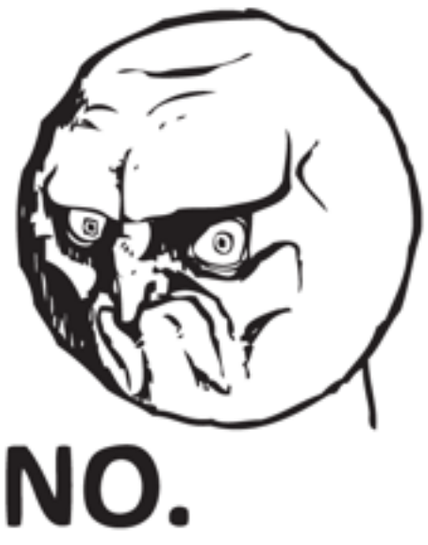
asyncio.Future

asyncio API

- 优先使用高级API，提供更简单的使用方法
- 只有在需要时，才选择使用底层API

再举个例子

```
20  async def main():
21      a, b = True, True
22      await fn1()
23      result1 = None
24      if a is True:
25          result1 = await fn2()
26      result2 = None
27      if b is True:
28          result2 = await fn3()
29      print(result1, result2)
```



gather

```python
async def main():
    a, b = True, True
    tasks = [
        fn1(),
        fn2() if a is True else None,
        fn3() if b is True else None,
    ]
    _, result1, result2 = await asyncio.gather(*tasks)
```

contextvar

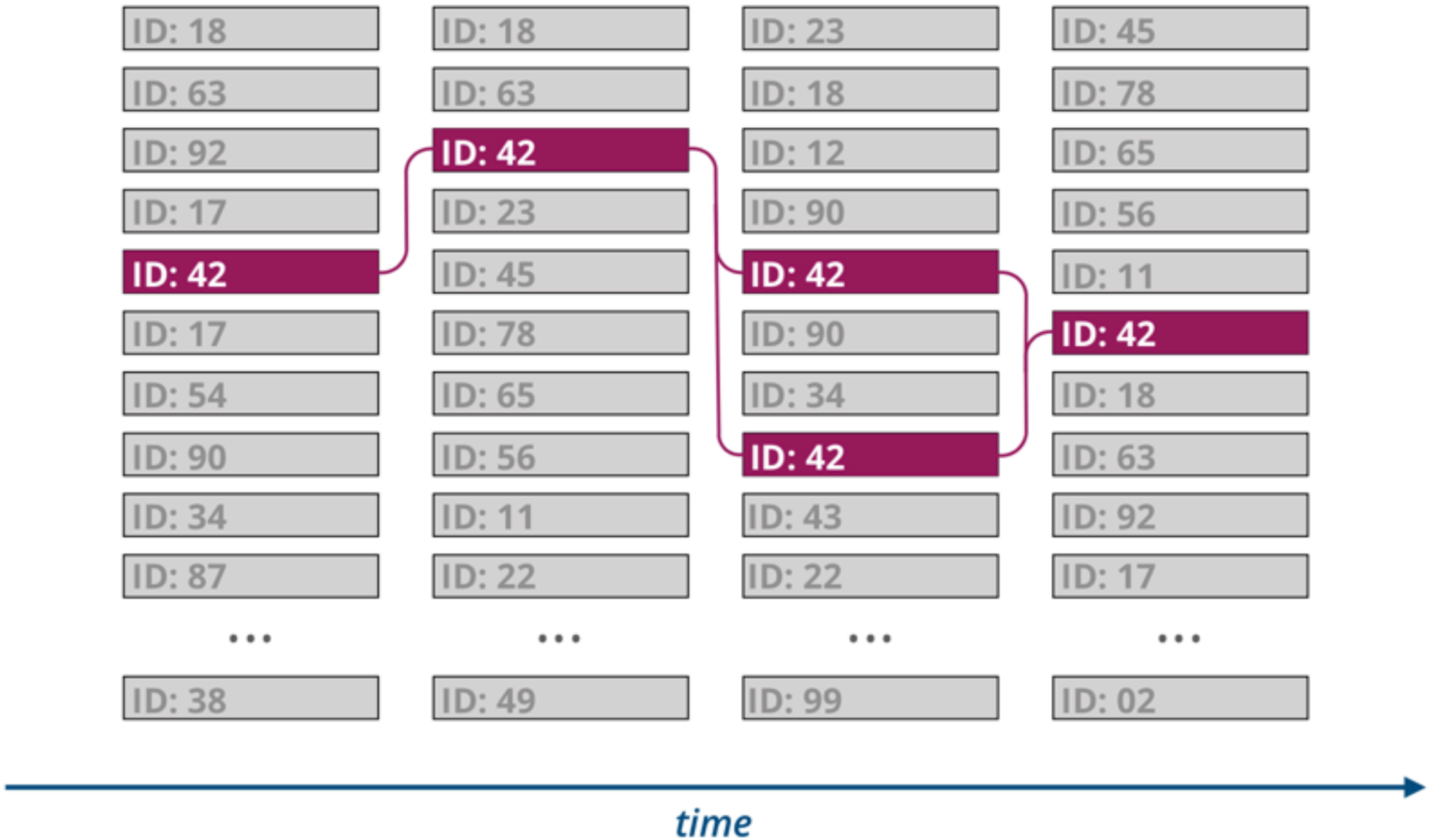
```
import asyncio
from contextvars import ContextVar

client_addr_var = ContextVar("client_addr")


def render_goodbye():
    return f"Good bye, client @ {client_addr_var.get()}\n".encode()


async def echo(r, w):
    addr = w.transport.get_extra_info("socket").getpeername()
    client_addr_var.set(addr)
    while True:
        data = await r.read(100)
        w.write(data)
    w.write(render_goodbye())
    w.close()
```

contextvar


ID: 18
ID: 63
ID: 92
ID: 17
ID: 42
ID: 17
ID: 54
ID: 90
ID: 34
ID: 87
ID: 38
ID: 18
ID: 63
ID: 42
ID: 23
ID: 45
ID: 78
ID: 65
ID: 56
ID: 11
ID: 22
ID: 49
ID: 23
ID: 18
ID: 12
ID: 90
ID: 42
ID: 90
ID: 34
ID: 42
ID: 43
ID: 22
ID: 99
ID: 45
ID: 78
ID: 65
ID: 56
ID: 11
ID: 42
ID: 18
ID: 63
ID: 92
ID: 17
ID: 02
time

async传染

```
In [1]: import asyncio

In [2]: async def my_sleep(n):
   ...:     await asyncio.sleep(n)
   ...:

In [3]: def caller():
   ...:     my_sleep(1)
   ...:

In [4]: caller()
/Users/kevin/.pyenv/versions/3.7.1/bin/ipython:2: RuntimeWarning: coroutine
'my_sleep' was never awaited
```

async传染

```
 6  def wait(coro):
 7      loop = asyncio.get_event_loop()
 8      return loop.run_until_complete(coro)
 9
10
11  def async_test(func):
12      def inner(*args, **kwargs):
13          return wait(func(*args, **kwargs))
14
15      return inner
16
17
18  class TestAsync(unittest.TestCase):
19      @async_test
20      async def test_async_method(self):
21          await asyncio.sleep(1)
```

async传染

```
In [5]: from asgiref.sync import async_to_sync

In [6]: @async_to_sync
   ...: async def my_sleep(n):
   ...:     await asyncio.sleep(n)
   ...:

In [7]: caller()
```

async传染

```python
from asgiref.sync import sync_to_async


@sync_to_async
def get_chat_id(name):
    return Chat.objects.get(name=name).id

async def main():
    result = await get_chat_id("helloworld")
```

Django Channels

```python
import aiohttp

from django.views.generic import AsyncView
from django.http import HttpResponse
from django.conf import settings


class ASGIView(AsyncView):

    async def get(self, request, *args, **kwargs):
        response_text = ''
        async with aiohttp.ClientSession() as session:
            async with session.get('https://www.google.com') as response:
                response_text = await response.text()
        return HttpResponse(response_text)

```

flask

Does werkzeug have plans to support ASGI? #1322

Open **Lynskylate** opened this issue on 8 Jun · 14 comments

Lynskylate commented on 8 Jun

Werkzeug offer many useful method, It will be much easier than starting from scratch

davidism commented on 8 Jun Member

Yes, Werkzeug and Flask will eventually support ASGI. I do not have a timeline for this, although I would be happy to help review a PR if someone started one.

👍 7

responder

```python
import responder

api = responder.API()


@api.route("/greet/{greeting}")
async def greet_world(req, resp, *, greeting):
    resp.text = f"{greeting}, world!"


@api.route("/hello/{who}")
def hello_to(req, resp, *, who):
    resp.text = f"hello, {who}!"


if __name__ == '__main__':
    api.run()
```

asyncio调试

```
$ PYTHONASYNCIODEBUG=1 python app.py
INFO: Started server process [17423]
INFO: Waiting for application startup.
INFO: <Server sockets=[<socket.socket fd=7, family=AddressFamily.AF_INET, ty
pe=SocketKind.SOCK_STREAM, proto=6, laddr=('127.0.0.1', 50102)>]> is serving
INFO: <Server sockets=[<socket.socket fd=8, family=AddressFamily.AF_INET, ty
pe=SocketKind.SOCK_STREAM, proto=6, laddr=('127.0.0.1', 5042)>]> is serving
INFO: Uvicorn running on http://127.0.0.1:5042 (Press CTRL+C to quit)
INFO: poll 999.913 ms took 1002.033 ms: timeout
INFO: poll 999.834 ms took 1004.829 ms: timeout
```

https://docs.python.org/3/library/asyncio-dev.html#debug-mode

aiomonitor

```python
import asyncio

import aiomonitor
import responder

api = responder.API()


@api.route("/{greeting}")
async def greet_world(req, resp, *, greeting):
    resp.text = f"{greeting}, world!"


if __name__ == "__main__":
    loop = asyncio.get_event_loop()
    with aiomonitor.start_monitor(loop=loop):
        api.run(loop=loop)
```

aiomonitor

```
$ nc 127.0.0.1 50101

Asyncio Monitor: 3 tasks running
Type help for commands
monitor >>> ps
+------------+---------+-----------------------------------------------------------------------------+
| Task ID    | State   | Task                                                                        |
+------------+---------+-----------------------------------------------------------------------------+
| 4513590632 | PENDING | <Task pending coro=<Lifespan.run() running at                               |
|            |         | /Users/kevin/.pyenv/versions/3.7.1/lib/python3.7/site-                      |
|            |         | packages/uvicorn/lifespan.py:28> wait_for=<Future pending                   |
|            |         | cb=[<TaskWakeupMethWrapper object at 0x10d0b0fa8>()]>>                      |
| 4513590952 | PENDING | <Task pending coro=<Server.tick() running at                                |
|            |         | /Users/kevin/.pyenv/versions/3.7.1/lib/python3.7/site-                      |
|            |         | packages/uvicorn/main.py:400> wait_for=<Future pending                      |
|            |         | cb=[<TaskWakeupMethWrapper object at 0x10d0b0f78>()]>>                      |
| 4513591112 | FINISHED | <Task finished coro=<start_interactive_server() done, defined at           |
|            |         | /Users/kevin/.pyenv/versions/3.7.1/lib/python3.7/site-                      |
|            |         | packages/aioconsole/server.py:17> result=<Server socke....1', 50102)>]>>    |
+------------+---------+-----------------------------------------------------------------------------+
monitor >>> help
Commands:
            ps                : Show task table
            where taskid      : Show stack frames for a task
            cancel taskid     : Cancel an indicated task
            signal signame    : Send a Unix signal
            stacktrace        : Print a stack trace from the event loop thread
            console           : Switch to async Python REPL
            quit              : Leave the monitor
```

asyncio

- 为什么我们需要异步?
- 为什么选择asyncio?
- 如何优雅的编写asyncio程序?
- asyncio的其他

其他：优化

- uvloop
- 新版本也是一种优化：

- Future and Task classes now have an optimized C implementation which makes asyncio code up to 30% faster. (Contributed by Yury Selivanov and INADA Naoki in bpo-26081 and bpo-28544.)

- The asyncio.get_event_loop() function has been reimplemented in C to make it up to 15 times faster. (Contributed by Yury Selivanov in bpo-32296.)
- asyncio.Future callback management has been optimized. (Contributed by Yury Selivanov in bpo-32348.)
- asyncio.gather() is now up to 15% faster. (Contributed by Yury Selivanov in bpo-32355.)
- asyncio.sleep() is now up to 2 times faster when the *delay* argument is zero or negative. (Contributed by Andrew Svetlov in bpo-32351.)
- The performance overhead of asyncio debug mode has been reduced. (Contributed by Antoine Pitrou in bpo-31970.)

其他：注意事项

- 单独使用asyncio仍旧是单线程模型
- 只有IO才可以并发
- asyncio会带来CPU压力
  - event loop和上下文切换在高并发场景下仍旧有明显消耗

Thank You!