ASYNCIO PITFALLS

Andrew Svetlov

andrew.svetlov@gmail.com

https://asvetlov.github.io/piter-py-2017/

BIO

- Use Python since 1999
- Python Core Developer
- asyncio developer
- aiohttp co-author
- Leader of https://github.com/aio-libs team

INTRO

ASYNCIO STRONG POINTS

- 6 years old
- Super popular
- Very fast growing ecosystem

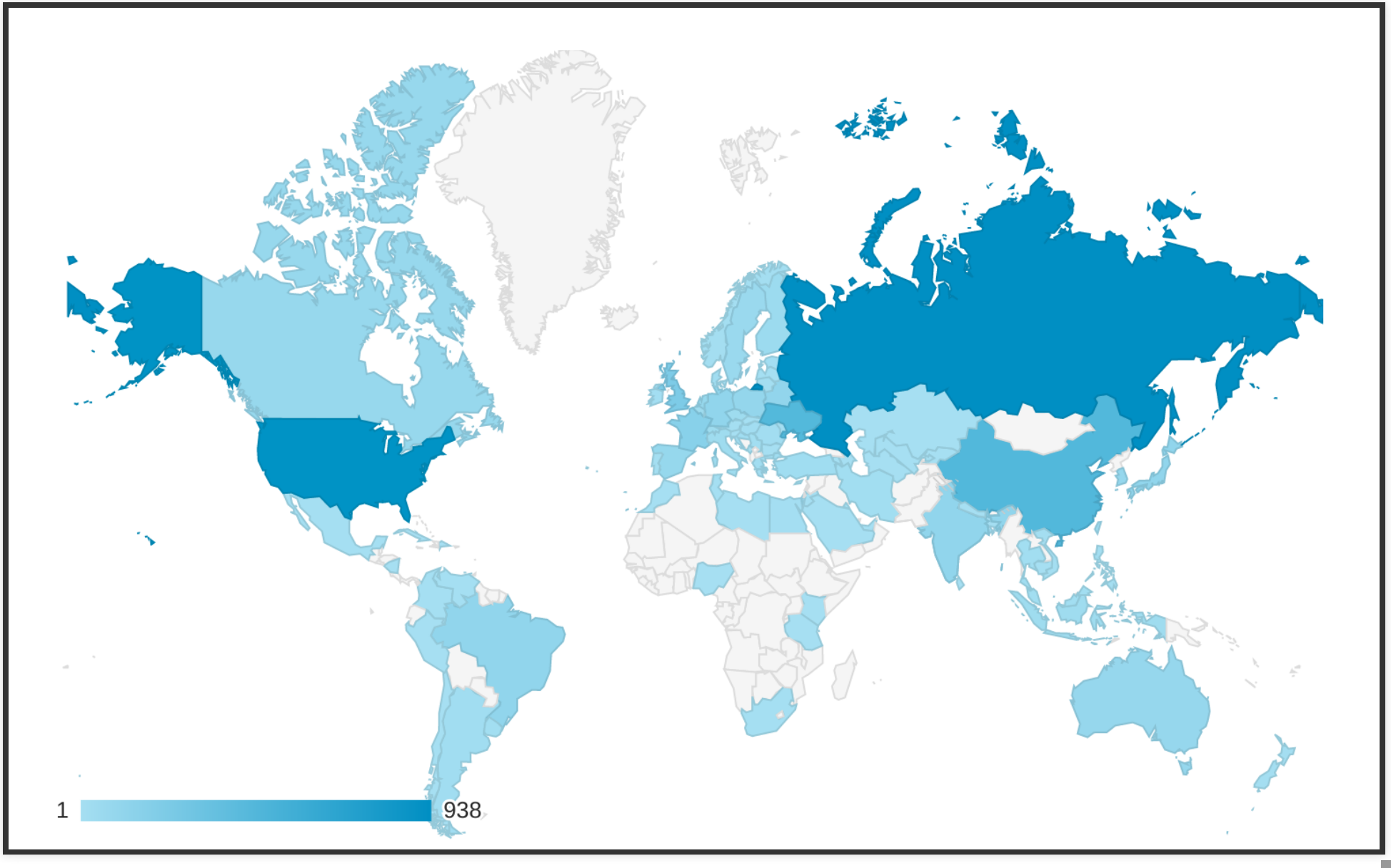
ASYNCIO KEY FEATURES

- Async Networking
- Performance
- Stability

AIOHTTP

- Async web client an server
- Developed since 2013
- Proven High load

AIOHTTP DOCUMENTATION USAGE



BASICS

ASYNC STYLE

```
async def f():
    return await g()
```

PAGE FETCHING

```
async def fetch(client):
    async with client.get('http://python.org') as resp:
        return await resp.text()
```

YIELD POINT

```
async def f():
    lst = []          # 1
    a = await g()     # 2
    lst.append(a)     # 3
    b = await h()     # 4
    lst.append(b)     # 5
    return lst        # 6
```

BLOCKING CALLS

```
async def f():
    return calc_pi(1000)
```

THREAD POOL

```
async def f():
    loop = asyncio.get_event_loop()
    return await loop.run_in_executor(None, calc_pi, 1000)
```

LIGHTWEIGHT TASKS

```
async def fetch(client, url):
    ...

async def main(client, loop):
    tasks = []
    for url in urls:
        task = loop.create_task(fetch(client, url))
        tasks.append(task)
    await asyncio.gather(*tasks)
```

DEBUG MODE

```
$ PYTHONASYNCIODEBUG=x python x.py
```

TIMEOUTS

STABILITY

- Network is not reliable
- ... newer
- ... ever
- But it's pretty stable on developer's machine

EXPLICIT TIMEOUTS

```
async with async_timeout.timeout(10):
    await gather(url)
```

REALITY

- Why explicit timeouts never work?
- ... and what to do?

Answer: reasonable **timeout defaults** in *every library*.

SPAWN TASKS

```
async def fetcher(client, url):
    async with timeout(10):
        async with client.get(url) as resp:
            return await resp.text()

async def main(client, loop):
    tasks = []
    for url in urls:
        task = loop.create_task(fetch(client, url))
        tasks.append(task)
    await asyncio.gather(*tasks)
```

SPAWN TASKS 2

```
async def fetcher(client, url):
    async with client.get(url) as resp:
        return await resp.text()

async def main(client, loop):
    tasks = []
    for url in urls:
        task = loop.create_task(fetch(client, url))
        tasks.append(task)
    async with timeout(10):
        await asyncio.gather(*tasks)
```

TASK CANCELLATION

VERY IMPLICIT AND DEADLY UNEXPECTED

WEB-HANDLER

```
async def handler(request):
    db = request.app['db']
    async with db.acquire() as conn:
        await conn.execute("INSERT ... INTO ...")
```

Hint: `asyncio.shield()`

AIOJOBS

FIRE AND FORGET

```
loop.create_task(worker(param))
```

TASK WAITING

```
task = loop.create_task(worker(param))
...
await task
```

SAFE CODE

- Control amount of spawned tasks
- Process their exceptions
- Close spawned but not finished yet tasks properly

```python
async def coro(timeout):
    await asyncio.sleep(timeout)

async def main():
    scheduler = await aiojobs.create_scheduler()
    for i in range(100):
        # spawn jobs
        await scheduler.spawn(coro(i/10))

    await asyncio.sleep(5.0)
    # not all scheduled jobs are finished at the moment

    # gracefuly close spawned jobs
    await scheduler.close()
```

AIOJOBS AND AIOHTTP

```python
from aiohttp import web
from aiojobs.aiohttp import atomic, setup

@atomic
async def handler(request):
    await save_to_db()
    return web.Response()

app = web.Application()
app.router.add_post('/post', handler)
setup(app)
```

ASYNC DESIGN HINTS

HIDDEN CONSTRUCTORS

```
class A:
    client = aiohttp.ClientSession()
```

VS

```
client = await aiohttp.create_session()
```

SPLIT BRAIN

```
reader, writer = await asyncio.open_connection(host, port)
```

VS

```
stream = await open_connection(host, port)
```

ASYNC FUNCTIONS EVERYWHERE

```
writer.write(b'data')
await writer.drain()
```

VS

```
await stream.write(b'data')
```

CLOSE() SHOULD BE AN ASYNC

```
writer.close()
```

VS

```
server.close()
await server.wait_closed()
```

VS

```
await stream.close()
```

QUESTIONS?

Andrew Svetlov

http://asvetlov.blogspot.com

andrew.svetlov@gmail.com

https://asvetlov.github.io/piter-py-2017/