
Async Python. Who's there? Knock knock.

Parallelism, concurrency, multi-processing, threading, oh my! 

Parallelism: performing multiple operations at the same time.

Concurrency: performing multiple operations in an overlapping manner.

As David said earlier today...

"Nobody uses threads to make their applications less buggy or more stable."

-- @DavidOstrovsky #confoo

Async I/O: a style of concurrent programming in which tasks release the CPU during idling periods, so that other tasks can use it.

Which model is the best?

It depends...

1. heavy computations or maths (CPU bound)
   - multi-processing
2. network, servers, HTTP, sockets (I/O bound)
   - multi-threading
   - asyncio

How does asyncio work?

Marc Aubé @maaube 12 m

Hey @miguelgrinberg 👋 I was wondering if I could borrow your Judit Polgár example, for a talk on asyncio. I think it beautifully illustrates cooperative concurrency. Obviously, I'd give credit for the analogy.

Miguel Grinberg
@miguelgrinberg

@maaube Yes, I have no problem with that! Send me a link to your talk if it makes it to youtube!

@maaube

Chess Exhibition

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

Synchronous Chess Exhibition

- Judit plays one game at a time
- Each game runs for 30 minutes
- There are 24 games to play
- 24 sequential games would take:

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

@maaube

Asynchronous Chess Exhibition

- Polgár makes a move
- While the opponent thinks, she moves on the second game, then third, ...
- A move on all 24 games takes her:

  24 x 5 sec = 120 sec = *2 minutes*
- 24 games are completed in:

  2 min x 30 = 60 min = *1 hour!*

@maaube

Asynchronous Chess Exhibition

- Polgár makes a move
- While the opponent thinks, she moves on the second game, then third, ...
- A move on all 24 games takes her:

  24 x 5 sec = 120 sec = *2 minutes*
- 24 games are completed in:

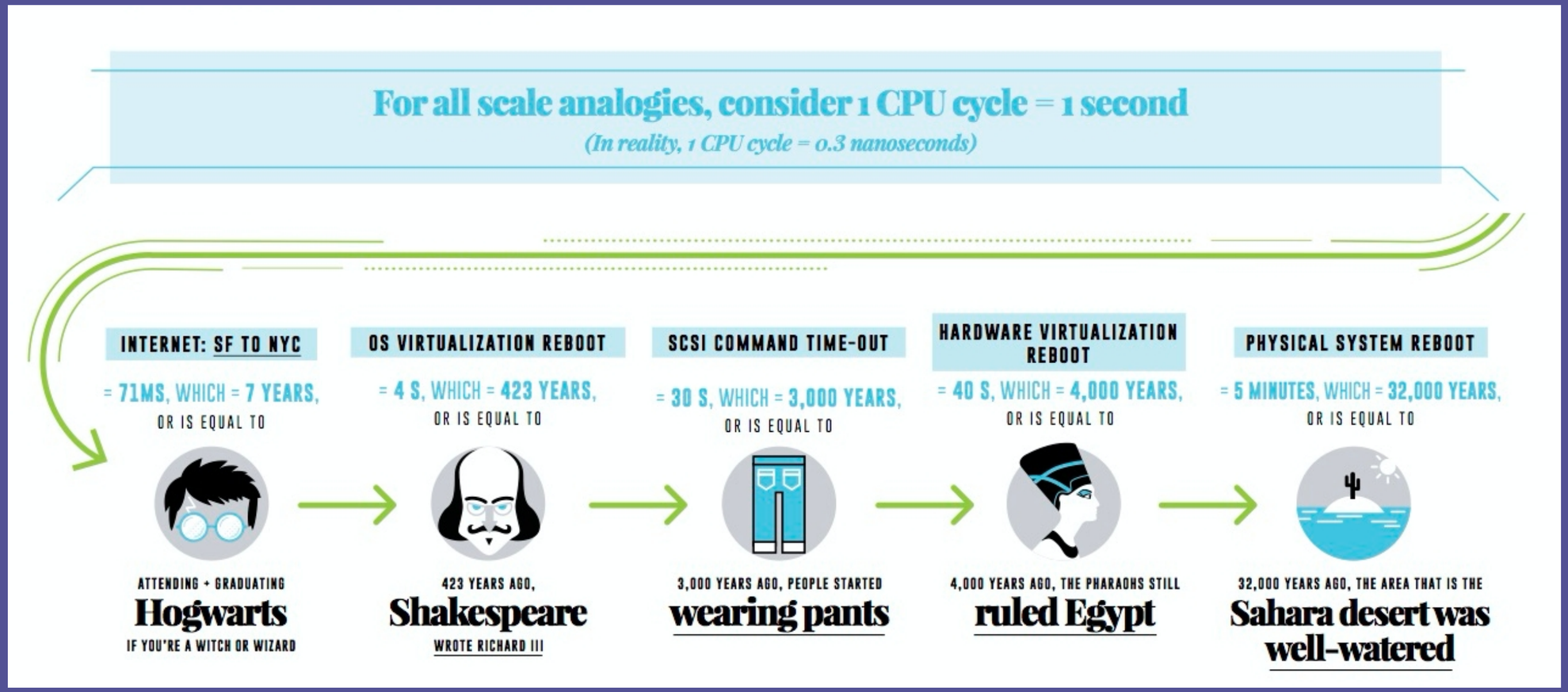
  2 min x 30 = 60 min = *1 hour!*

@maaube

Cooperative concurrency

A fancy way of saying that your async functions can give back control to the execution loop, *so that each function can run at the optimal time*.

Understanding computer time



async and await

```python
import asyncio

async def hello():
    print("Hello")
    await asyncio.sleep(1)
    print("World!")
```

```python
import asyncio

async def hello():
    print("Hello")
    await asyncio.sleep(1)
    print("World!")
```

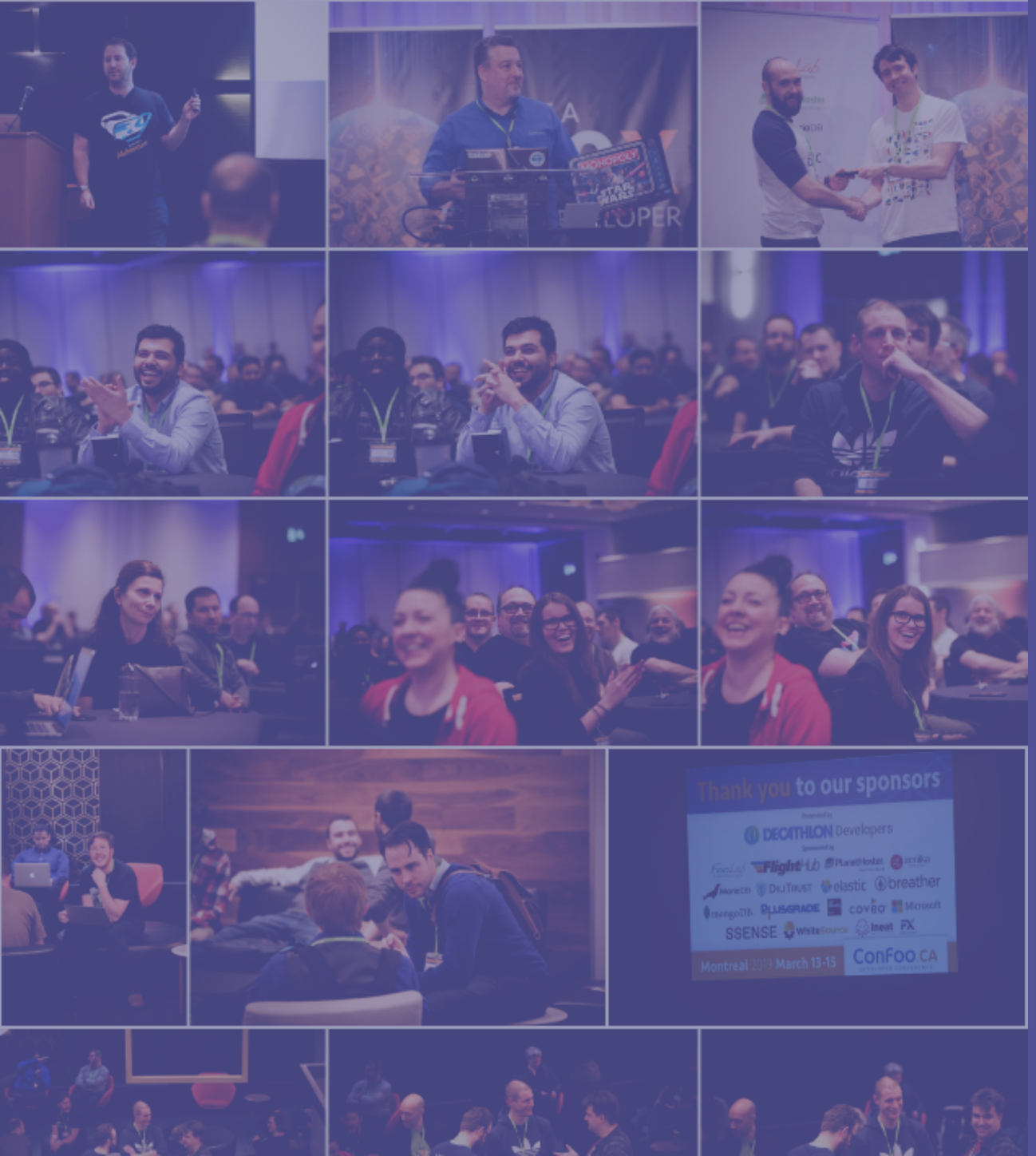
```python
import asyncio

async def hello():
    print("Hello")
    await asyncio.sleep(1)
    print("World!")

async def main():
    await asyncio.gather(hello(), hello(), hello())

if __name__ == "__main__":
    asyncio.run(main())
```

```
1 Hello
2 Hello
3 Hello
4 World!
5 World!
6 World!
```

```python
import asyncio

async def hello():
    print("Hello")
    await asyncio.sleep(1)
    print("World!")

async def main():
    await asyncio.gather(hello(), hello(), hello())

if __name__ == "__main__":
    asyncio.run(main())
```

```
async_hello.py executed in 1.01 seconds.
```

Made-up problem time!



Goals

1. Crawl Confoo's Flickr account
2. Download all pictures from photo stream
3. Make a 120x120 thumbnail for each of them
4. Convert it to black & white
5. Be fast! 🏎

```python
async def main():
    pages = [
        f'https://www.flickr.com/photos/confoo/page{i}'
        for i in range(1, 100)
    ]

    # TODO: crawl pages, find images, download and process them.
```

```python
import aiohttp

async def fetch_html(url):
    async with aiohttp.ClientSession() as session:
        response = await session.get(url, allow_redirects=False)

        if response.status != 200:
            return ''

        return await response.text()
```

```python
import aiohttp

IMG_PATTERN = re.compile(r'url\(\/\/(live.staticflickr\.com.+\.jpg)\)')

async def fetch_html(url, session):
    # ...

async def find_images_in_page(url, session):
    async with aiohttp.ClientSession() as session:
        html = await fetch_html(url, session)

        images = set()
        for img_url in IMG_PATTERN.findall(html):
            images.add(f'https://{img_url}')

        return images
```

```python
async def fetch_html(url, session):
    # ...

async def find_images_in_page(url, session):
    # ...

async def find_all_images(urls):
    async with aiohttp.ClientSession() as session:
        tasks = []
        for url in urls:
            tasks.append(find_images_in_page(url, session))

        return await asyncio.gather(*tasks)
```

```python
import itertools

def flatten(iterable):
    return list(itertools.chain.from_iterable(iterable))

async def find_all_images(urls):
    async with aiohttp.ClientSession() as session:
        # ...

        return flatten(await asyncio.gather(*tasks))
```

```python
async def main():
    pages = [
        f'https://www.flickr.com/photos/confoo/page{i}'
        for i in range(1, 100)
    ]

    images = await find_all_images(pages)
    print(f'[+] Found {len(images)} images.')

    # TODO: download images and process them.

if __name__ == '__main__':
    asyncio.run(main())
```

```python
from io import BytesIO
from PIL import Image

def process_image(bytes, filename):
    img = Image.open(BytesIO(bytes))
    img = img.resize((120, 120))
    img = img.convert("L")
    img.save(f'images/{filename}')
```

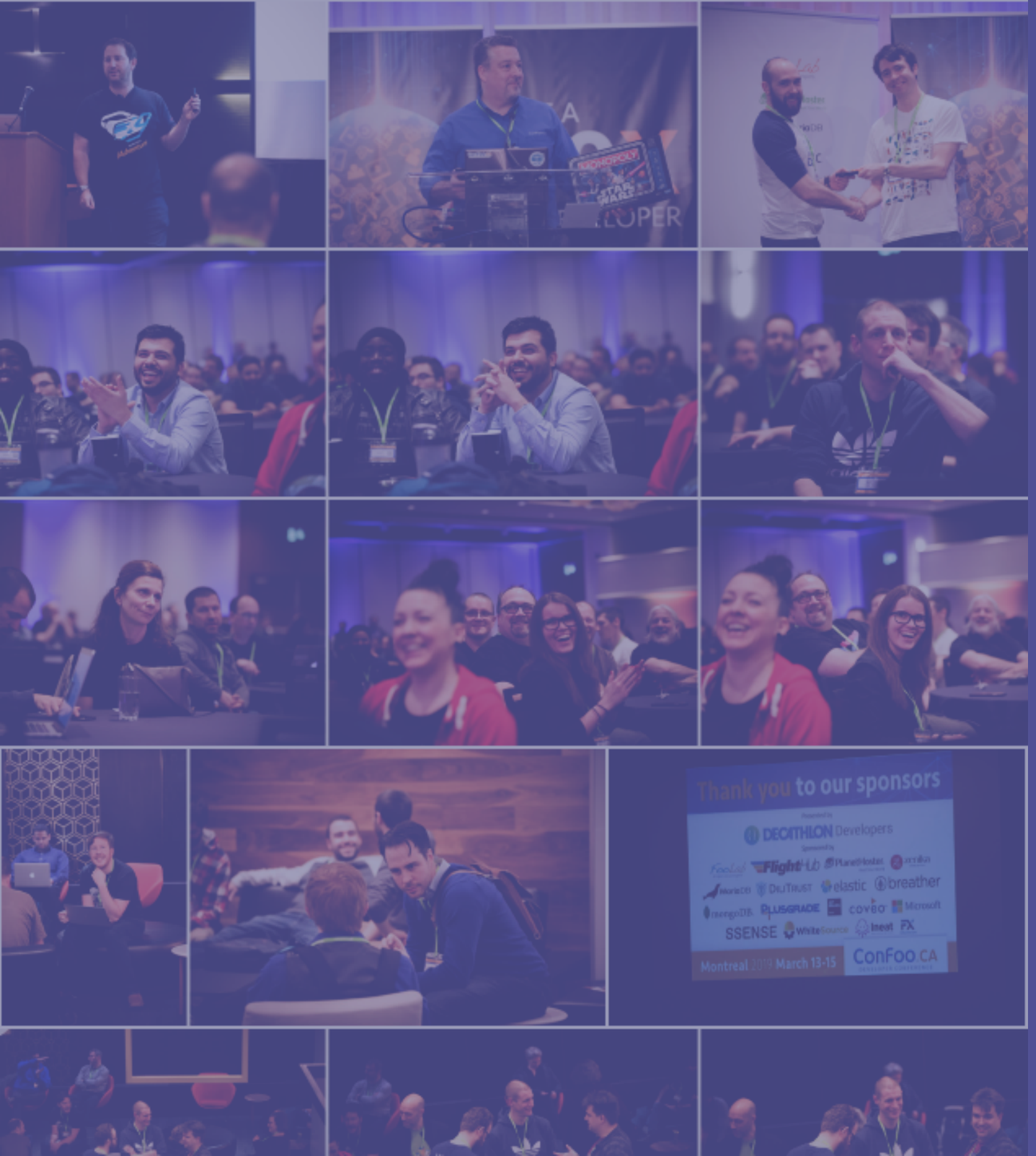
This function is blocking...

```python
import asyncio
from concurrent.futures.thread import ThreadPoolExecutor
from functools import partial

thread_pool = ThreadPoolExecutor()

def process_image(bytes, filename):
    # ...

async def async_process_image(bytes, filename):
    loop = asyncio.get_event_loop()

    await loop.run_in_executor(
        thread_pool,
        partial(process_image, bytes, filename)
    )
```

```python
def process_image(bytes, filename):
    # ...

async def async_process_image(bytes, filename):
    # ...

async def download_image(url):
    filename = url.split('/')[-1]

    async with aiohttp.ClientSession() as session:
        async with session.get(url) as resp:
            if resp.status == 200:
                await async_process_image(await resp.read(), filename)
```

```python
def process_image(bytes, filename):
    # ...

async def async_process_image(bytes, filename):
    # ...

async def download_image(url, session):
    # ...

async def download_images(urls):
    async with aiohttp.ClientSession() as session:
        tasks = []
        for url in urls:
            tasks.append(download_image(url, session))

        await asyncio.gather(*tasks)
```

```python
async def main():
    pages = [
        f'https://www.flickr.com/photos/confoo/page{i}'
        for i in range(1, 100)
    ]

    images = await find_all_images(pages)
    print(f'[+] Found {len(images)} images.')

    await download_images(images)
    print(f'[+] Processed all {len(images)} images.')

if __name__ == '__main__':
    asyncio.run(main())
```



Goals

1. Crawl Confoo's Flickr account ✅
2. Download all pictures from photo stream ✅
3. Make a 120x120 thumbnail for each of them ✅
4. Convert it to black & white ✅
5. Be fast! ❓

```
[-] Found 481 images.
[-] Processed all 481 images.

[!] Script executed in 6.52 seconds.
```

Going even faster! 

```python
async def main():
    pages = [
        f'https://www.flickr.com/photos/confoo/page{i}'
        for i in range(1, 100)
    ]

    q = asyncio.Queue()

    # Create one "producer" for every page, that will add image URLs to the Queue
    producers = [
        asyncio.create_task(find_images_in_page(url, q))
        for url in pages
    ]

    # Create 10 consumers, that will process images as soon as they're available
    consumers = [
        asyncio.create_task(download_image(q))
        for _ in range(10)
    ]

    await asyncio.gather(*producers)
    await q.join()  # Implicitly awaits consumers, too
    for c in consumers:
        c.cancel()

if __name__ == '__main__':
    asyncio.run(main())
```

GitHub repo

Pitfalls

- long CPU-intensive tasks
  - execute in a background thread
  - routinely release CPU with `await asyncio.sleep(0)`
- blocking library functions

Resources

README.md

Awesome asyncio 

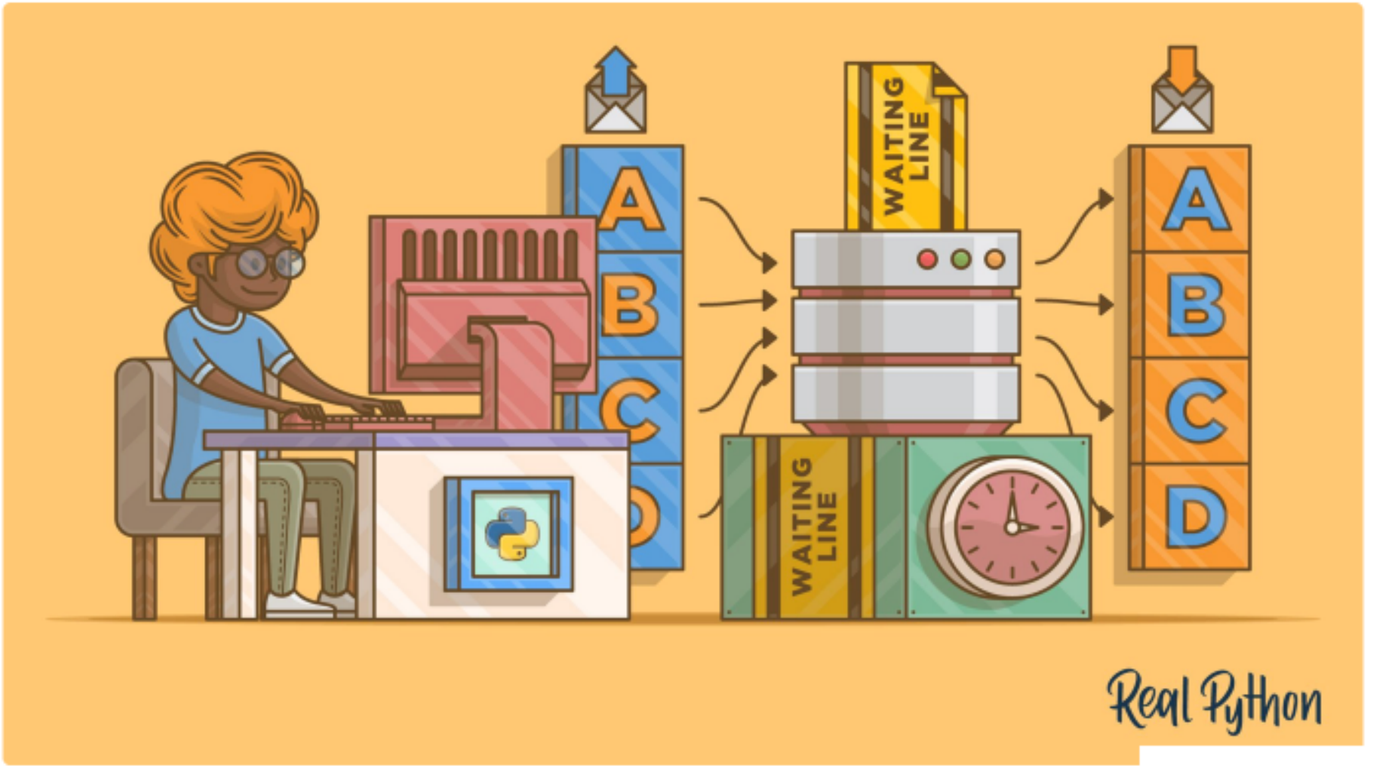
> A carefully curated list of awesome Python asyncio frameworks, libraries, software and resources.

The Python asyncio module introduced to the standard library with Python 3.4 provides infrastructure for writing single-threaded concurrent code using coroutines, multiplexing I/O access over sockets and other resources, running network clients and servers, and other related primitives.

Asyncio is not really a brand-new technology however it appears to be very trending since a few years - especially in the Python community and with the release of Python 3.4 in March 2014. Thus, it's pretty hard to keep yourself up-to-date with the most awesome packages out there. Find some of those *awesome* packages here and if you are missing one we count on you to create an Issue or a Pull Request with your suggestion.

Contents

Real Python Start Here Learn Python Store More Searc



Async IO in Python: A Complete Walkthrough

by Brad Solomon 36 Comments intermediate python

@maaube

Questions ?