AIO…WHATEVER

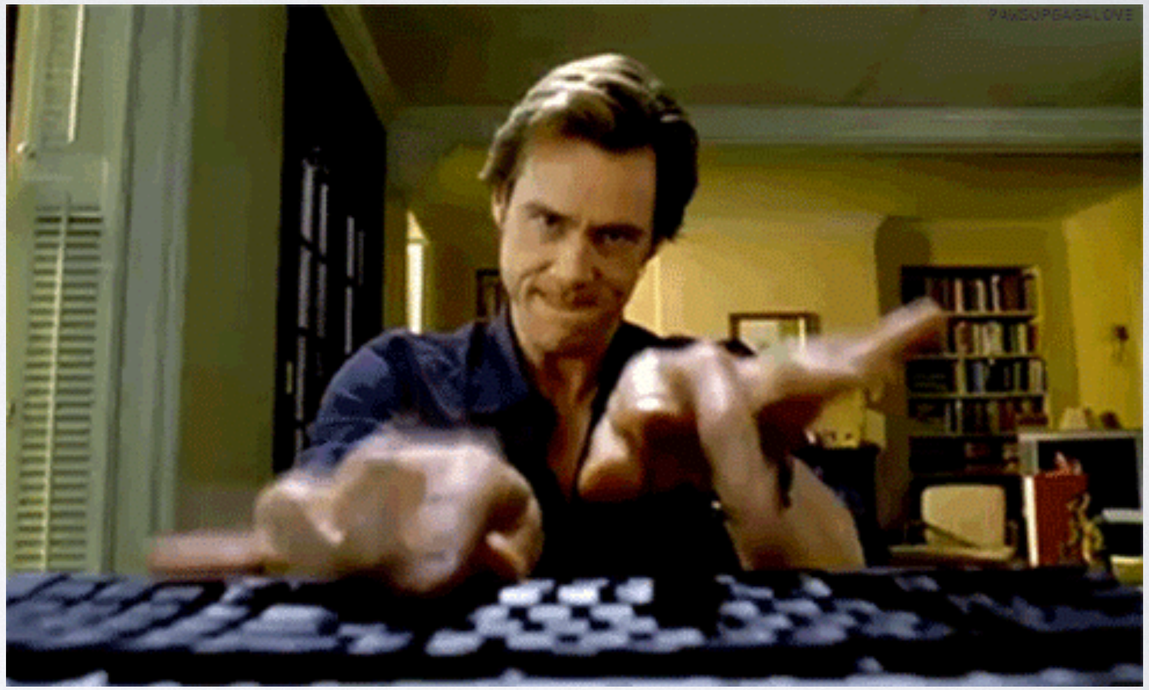
asyncio pour tout le monde

EN 2012…

Guido (et d'autres) commencent à travailler sur tulip

NODE.JS

Let's get shit done

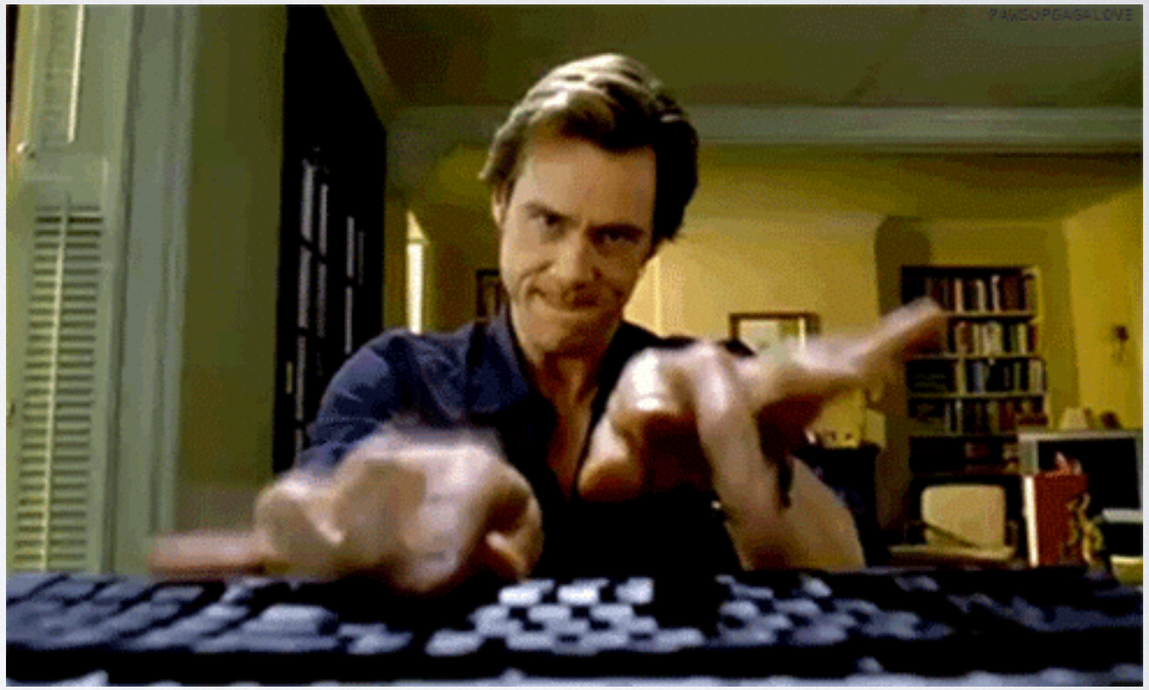


EN 2014

python 3.4 sort avec asyncio

NODE.JS

Let's get shit done



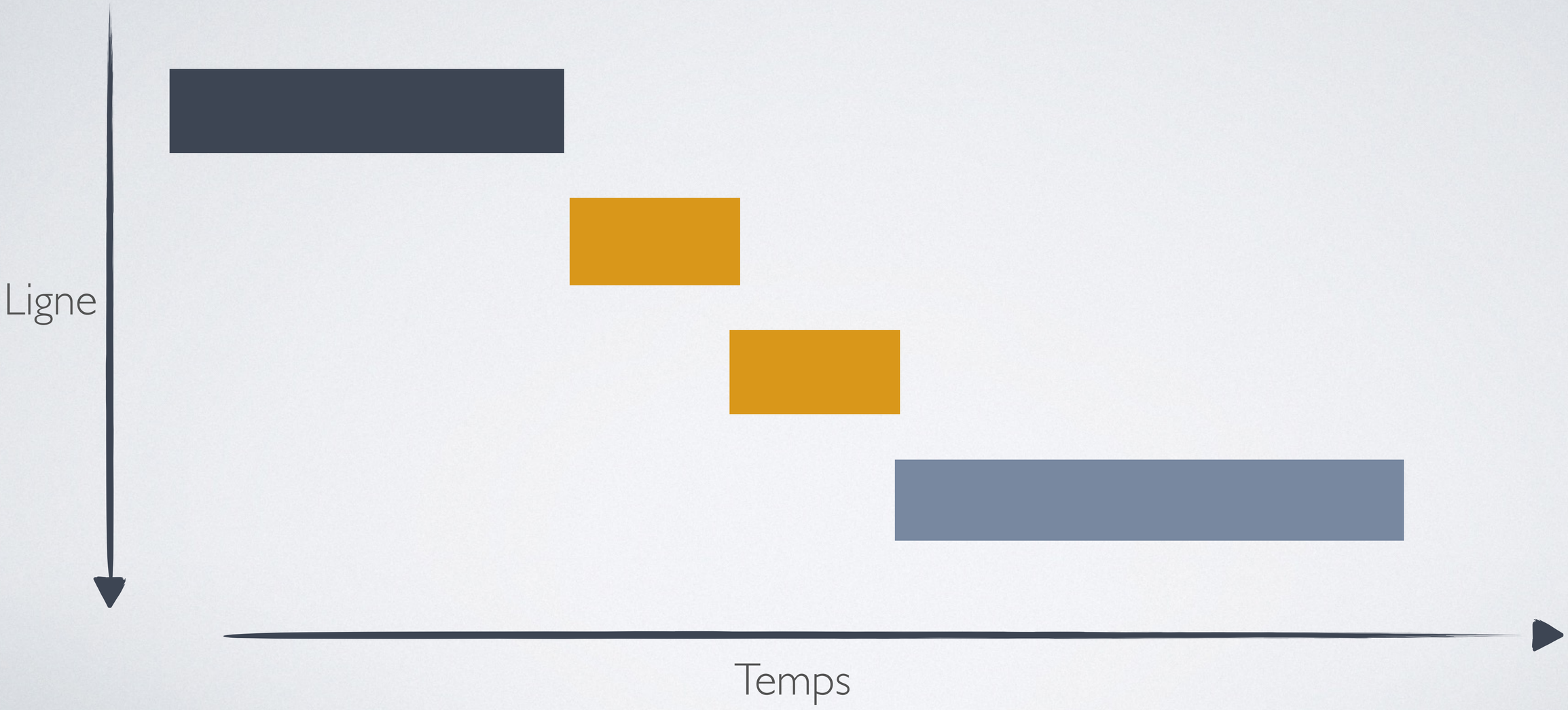
EN 2015…

sortie de python 3.5 avec async/await

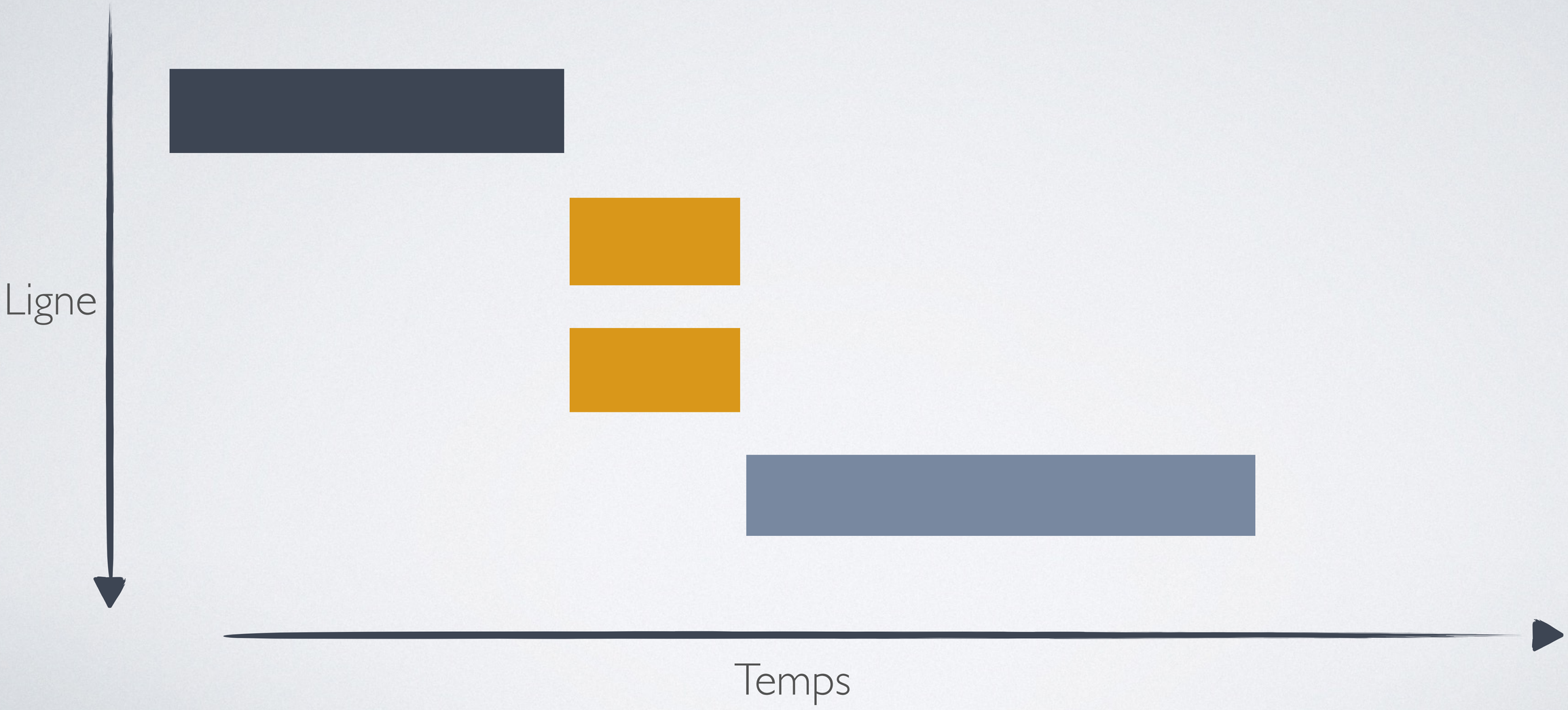
MAIS EN FAIT…

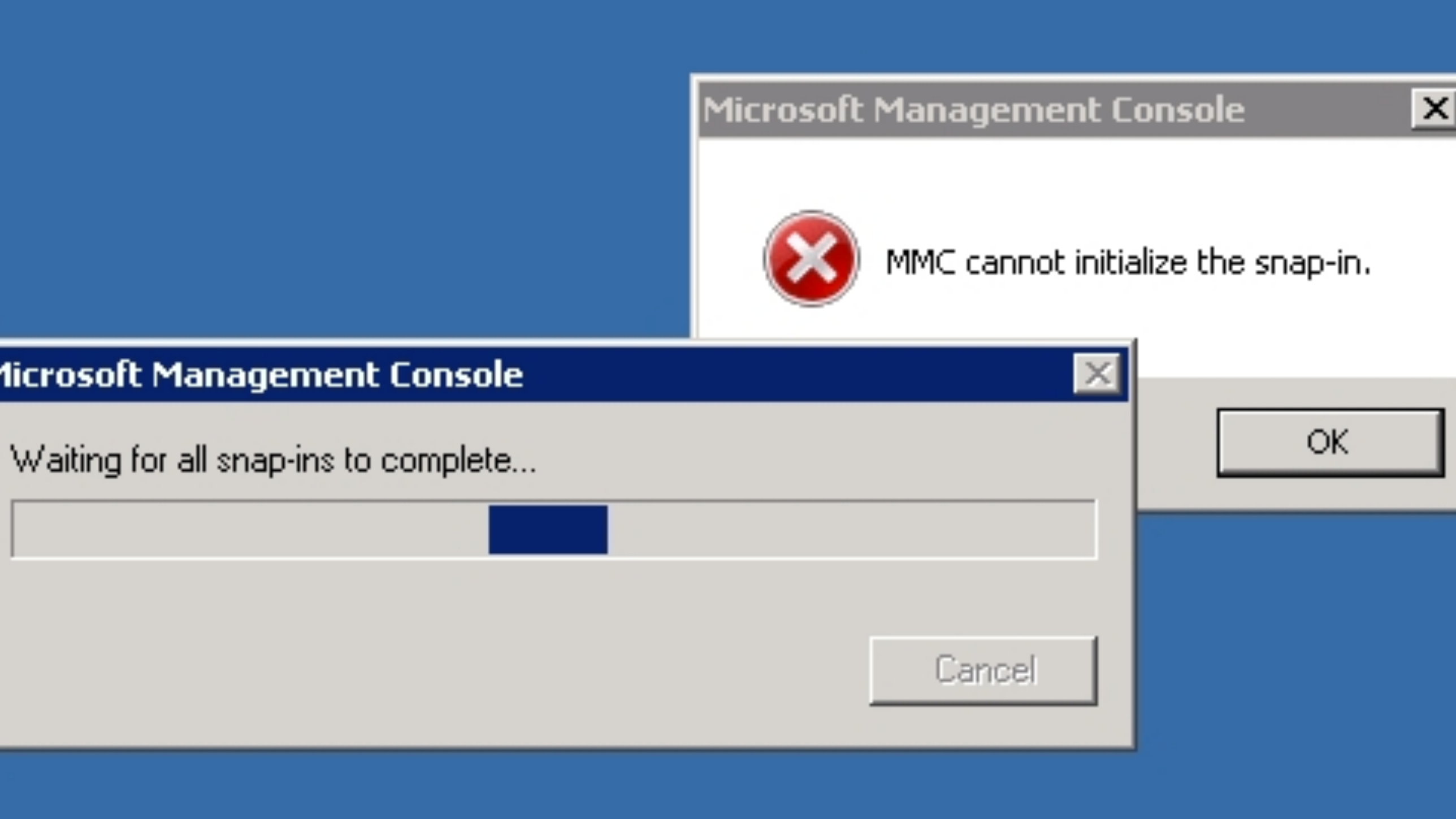
PROGRAMMATION ASYNCHRONE

PROGRAMMATION SYNCHRONE



PROGRAMMATION ASYNCHRONE




Microsoft Management Console
MMC cannot initialize the snap-in.
OK
Microsoft Management Console
Waiting for all snap-ins to complete...
Cancel

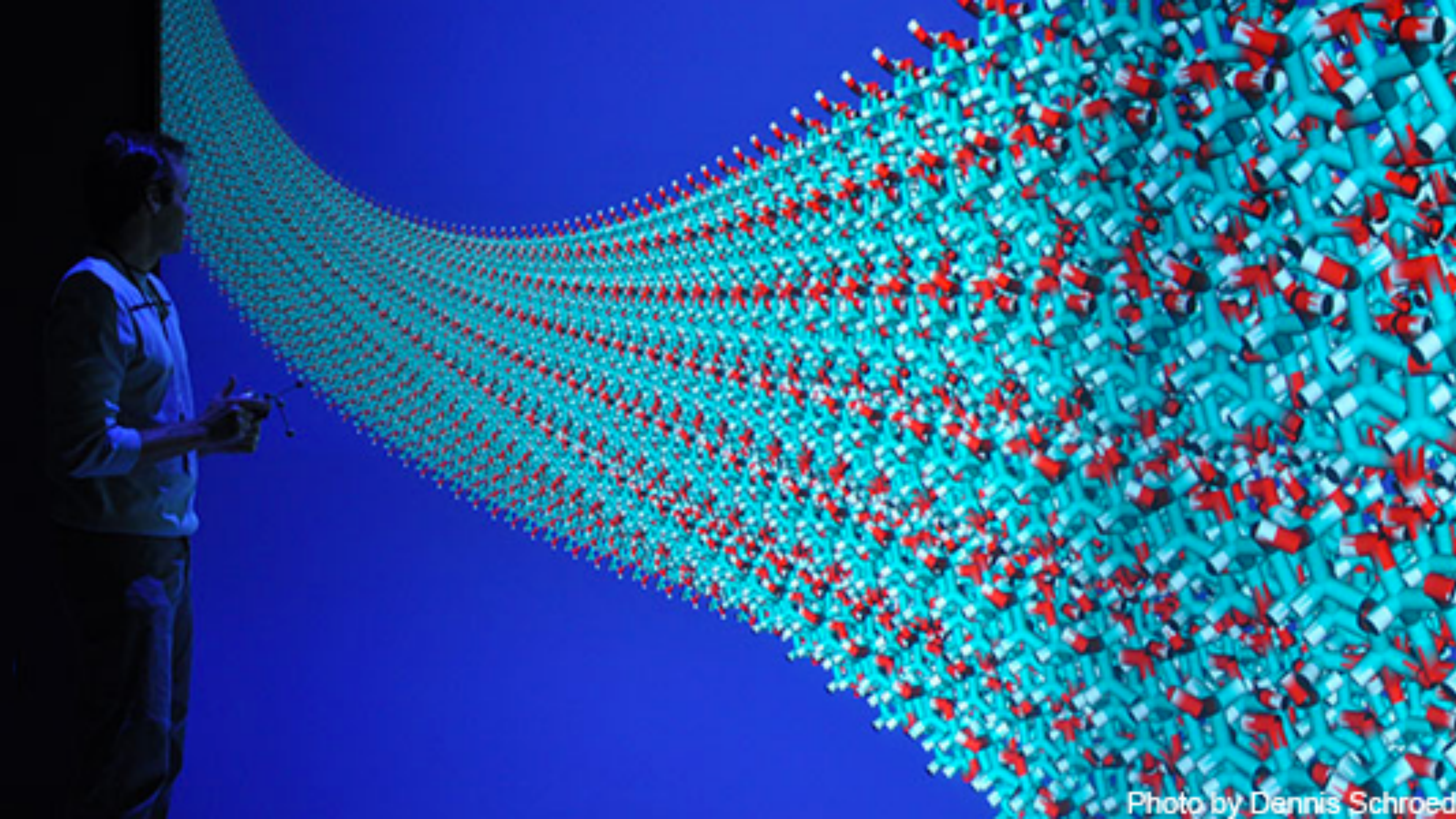


Photo by Dennis Schroed

SOLUTION?

- Threads

ET POUR LES HUMAINS?

- Thread pools
- Green threads
- Events
- Callbacks
- Futures / Promises
- Coroutines ➡ asyncio

AIO-LIBS

bibliothèques utilisant le standard

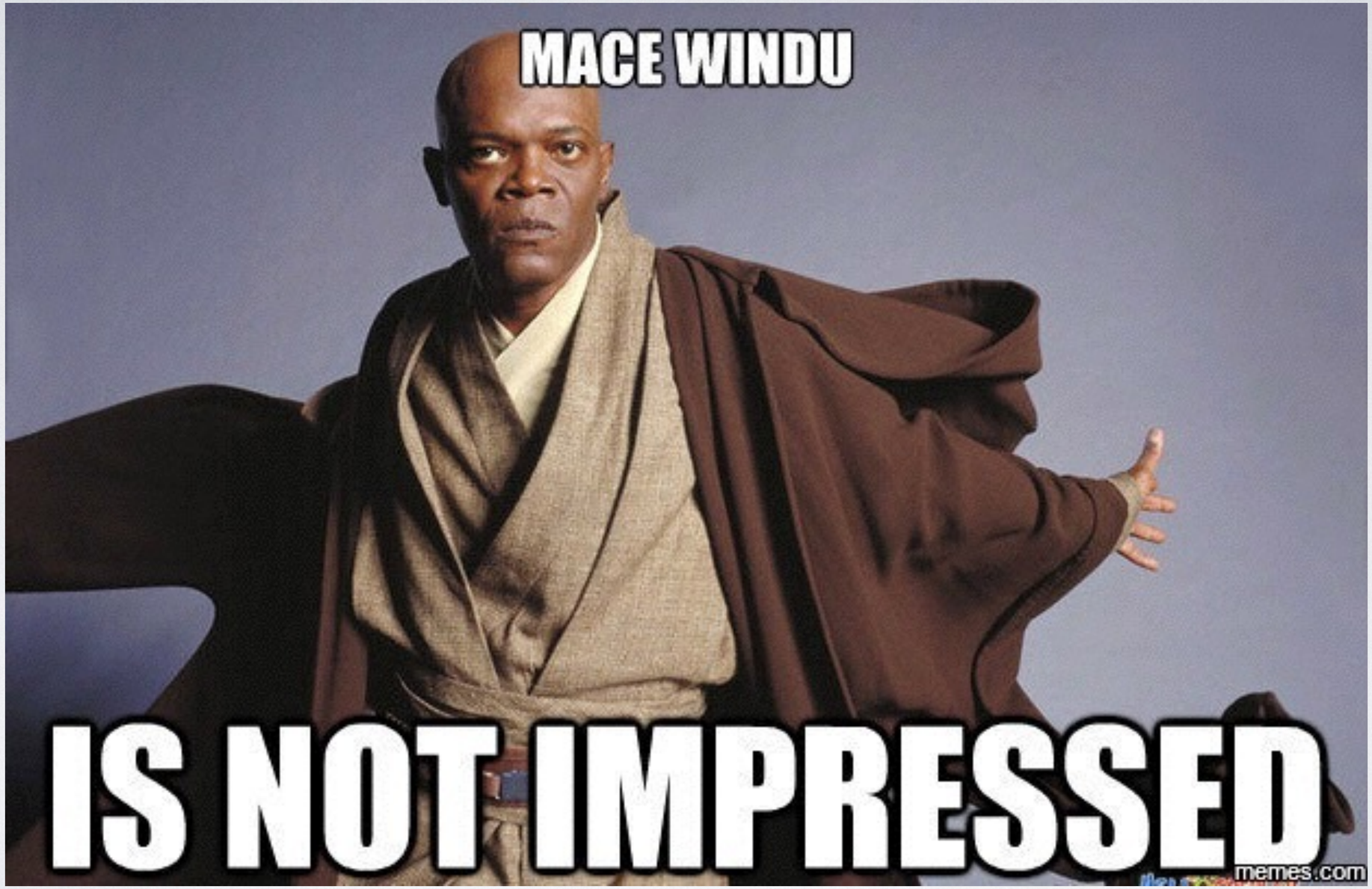
AIOHTTP CLIENT

```python
import requests

URL = 'http://echo.jsontest.com/key/value/one/two'

def call(url):
    resp = requests.get(url)
    print(resp.status_code)
    print(resp.json())

if __name__ == '__main__':
    call(URL)
```

```python
import asyncio
import aiohttp

URL = 'http://echo.jsontest.com/key/value/one/two'

async def call(url):
    async with aiohttp.get(url) as resp:
        print(resp.status)
        print(await resp.json())


if __name__ == '__main__':
    asyncio.get_event_loop().run_until_complete(call(URL))
```

```python
import requests
from concurrent.futures import ThreadPoolExecutor

URLS = [
    'http://echo.jsontest.com/key/value/test/1',
    'http://echo.jsontest.com/key/value/test/2',
    'http://echo.jsontest.com/key/value/test/3',
    'http://echo.jsontest.com/key/value/test/4',
    'http://echo.jsontest.com/key/value/test/5',
    'http://echo.jsontest.com/key/value/test/6',
]


def call(url):
    resp = requests.get(url)
    print(resp.status_code)
    print(resp.json())

if __name__ == '__main__':
    # [call(url) for url in URLS]

    executor = ThreadPoolExecutor(max_workers=10)
    executor.map(call, URLS)
```

```python
import aiohttp
import asyncio

URLS = [
    'http://echo.jsontest.com/key/value/test/1',
    'http://echo.jsontest.com/key/value/test/2',
    'http://echo.jsontest.com/key/value/test/3',
    'http://echo.jsontest.com/key/value/test/4',
    'http://echo.jsontest.com/key/value/test/5',
    'http://echo.jsontest.com/key/value/test/6',
]

async def call(url):
    async with aiohttp.get(url) as resp:
        print(resp.status)
        print(await resp.json())

if __name__ == '__main__':
    calls = asyncio.wait([call(url) for url in URLS])
    asyncio.get_event_loop().run_until_complete(calls)
```

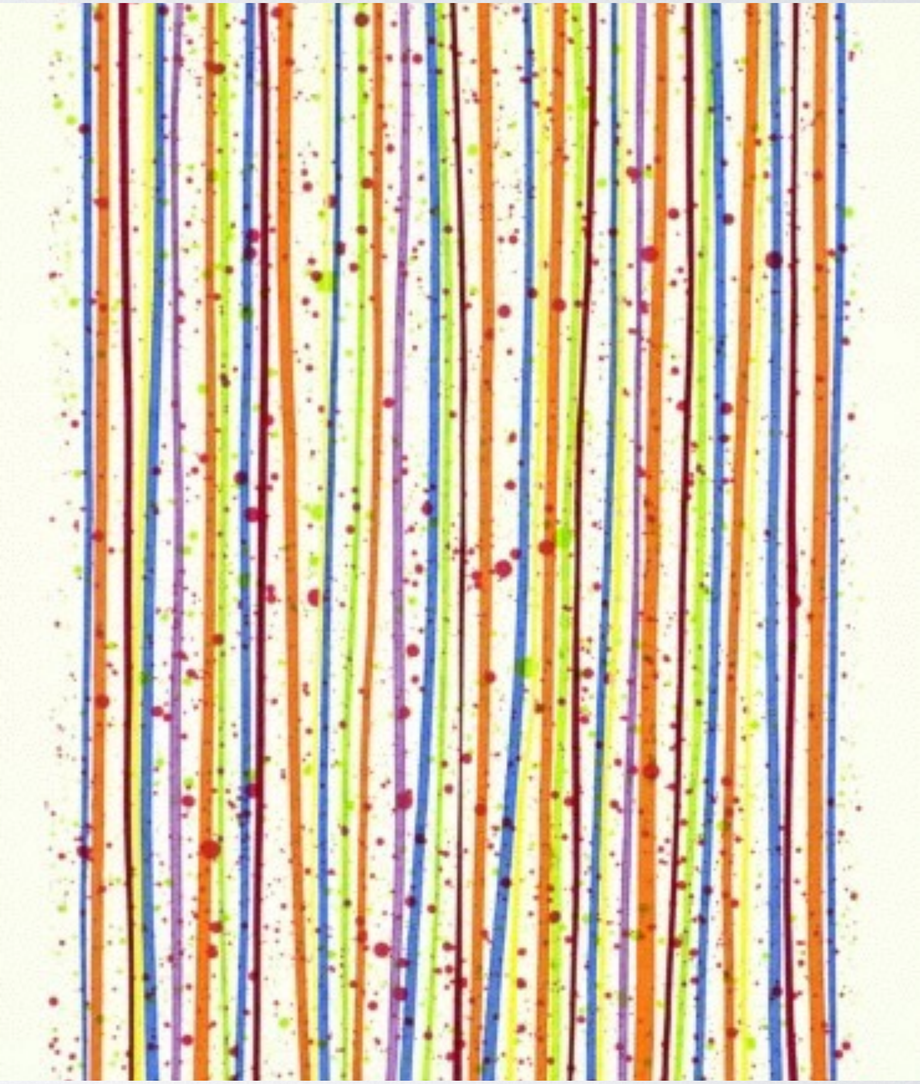

MACE WINDU
IS NOT IMPRESSED
memes.com

AIOHTTP SERVEUR

```python
from aiohttp import web
import asyncio
app = web.Application()

async def hello(request):
    name = request.match_info.get('name', "Anonymous")
    await asyncio.sleep(2)
    return web.json_response({
        'hello': name,
    })

app.router.add_route('GET', '/hello/{name}', hello)

web.run_app(app)
```

10 REQUÊTES EN PARALLÈLE

- aiohttp: 2 secondes (idem pour 100 requêtes)
- Flask: 20 secondes

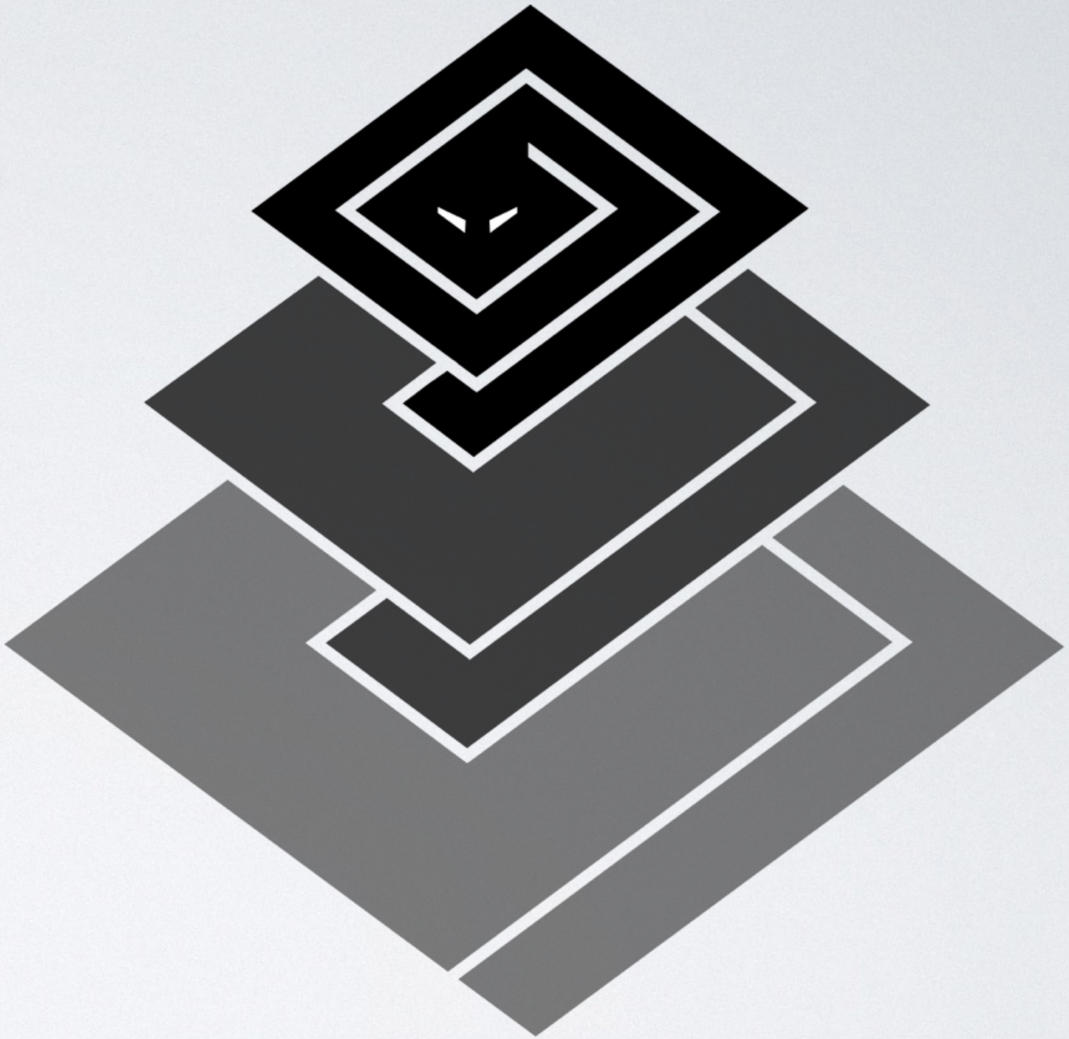


OUI MAIS

J'objecte

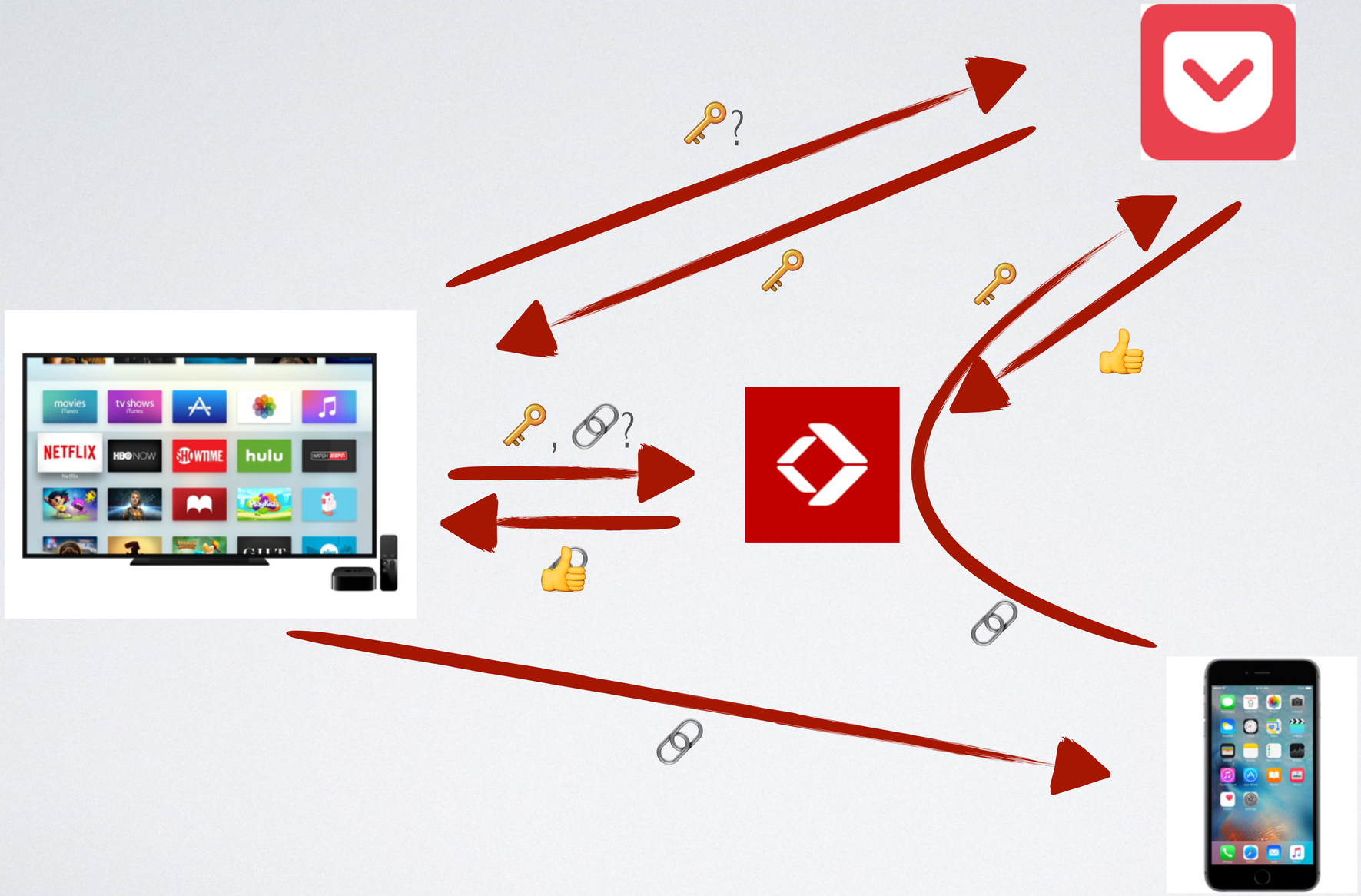
OUI MAIS 2

le retour de l’objection



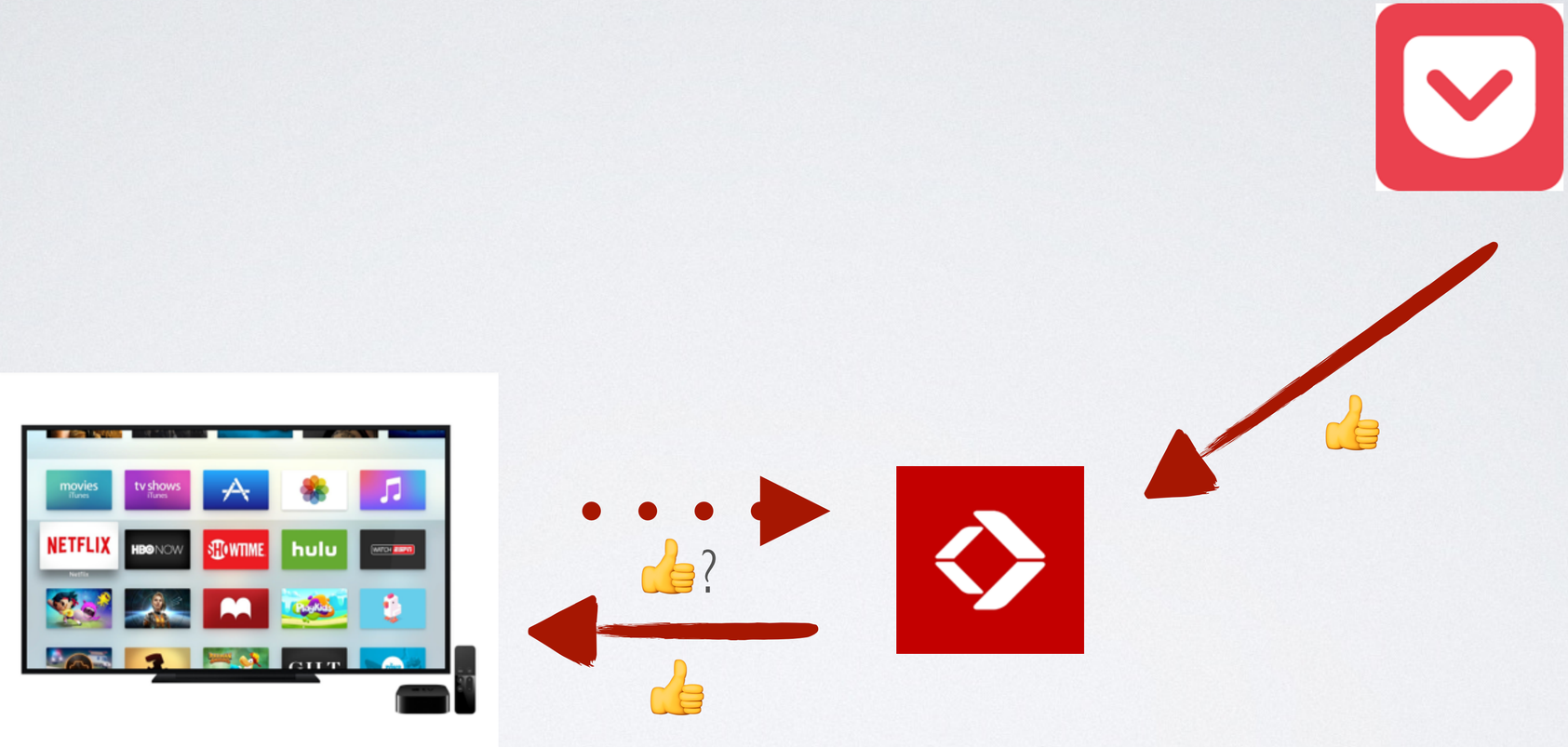
LE PROBLÈME

- WatchMeNow sur Apple TV
- Oauth sans webview
- logme.link



ARCHITECTURE

- Simple endpoints
  - Ajouter des token
  - Rediriger
- Longpolling


movies
tv shows
NETFLIX
HBO NOW
SHOWTIME
hulu

ARCHITECTURE

- aiohttp server
- redis pour garder l'état
- redis pub sub pour le longpolling

```python
async def fetch_info(request):
    secret = request.match_info.get('secret')
    redis_pool = request.app['redis_pool']
    redis_sub_pool = request.app['redis_subscribe_pool']

    # ... deal with some default cases

    async with redis_sub_pool.get() as redis_sub:
        channel, = await redis_sub.subscribe(secret)

        done, not_done = await asyncio.wait([channel.get(encoding='utf-8')], timeout=30)
        await redis_sub.unsubscribe(secret)

    async with redis_pool.get() as redis:
        info = await _info(redis, secret)

    return web.json_response(info)
```

LES AVANTAGES

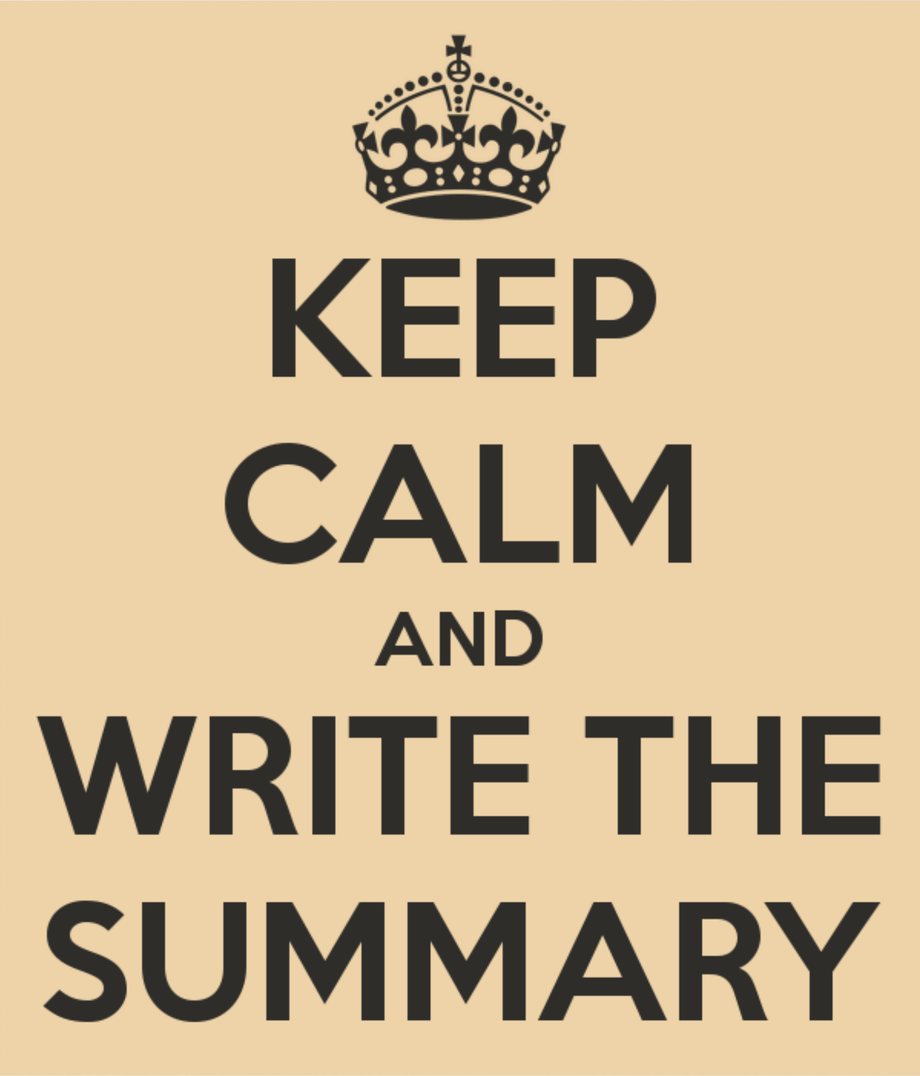
- Quelques lignes de Python
- Plusieurs clients peuvent attendre en parallèle
- Super facile à déployer

ET AUSSI

- aiohttp client
- aiopg, aiozmq, aiomysql…
- websockets

EN RÉSUMÉ

- Beaucoup de IO
- Peu de tâche intenses en CPU
- Les avantage de node.js sans les inconvénients
- Microservice, backend pour une app (JS ou native)



QUESTIONS ?