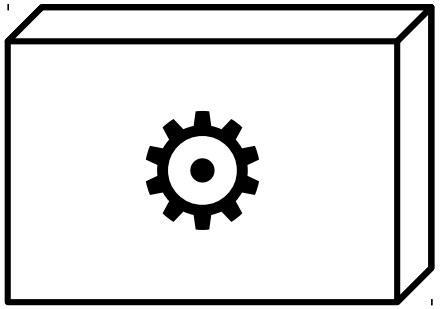
asyncio-nous

Jak i dlaczego spróbować asynchronicznego I/O z biblioteki standardowej

Python 3.4!

- 3.3 ('cause `yield from`)
- 3.4 ('cause `asyncio` in `stdlib`)



Problem

Wydajność:

- CPU bound

Problem

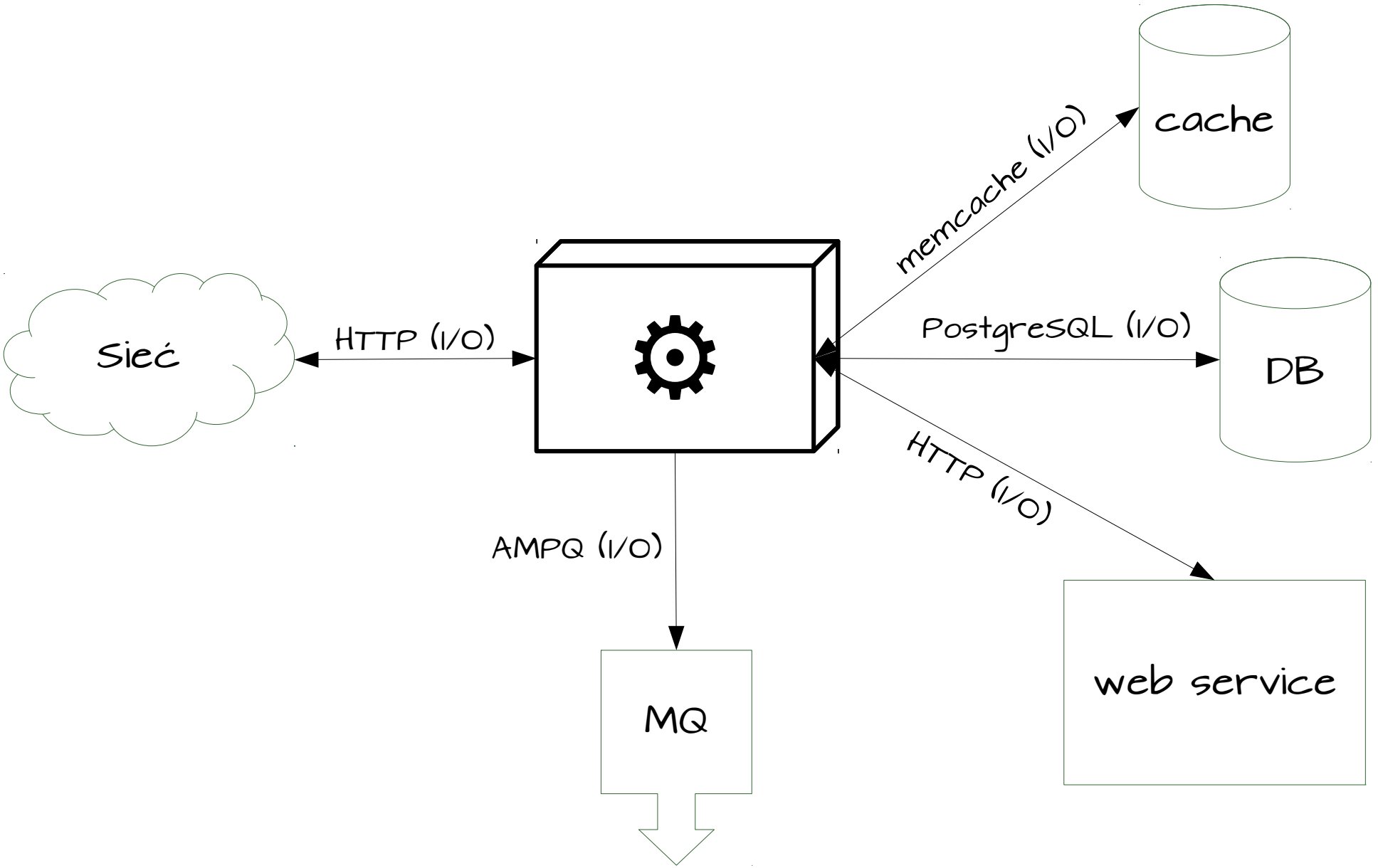
Wydajność:

- CPU bound
- Memory bound

Problem

Wydajność:

- CPU bound
- Memory bound
- I/O bound


cache
memcache (I/O)
Sieć
HTTP (I/O)
PostgreSQL (I/O)
DB
HTTP (I/O)
AMPQ (I/O)
MQ
web service

- Po co jest asynchroniczne I/O
- Czym jest moduł asyncio i dlaczego powstał
- Podstawowe koncepty
- Development kodu
- Możliwości wykorzystania

C10k

C10k

10 k concurrent clients

C10k

10 k concurrent clients

in 1999

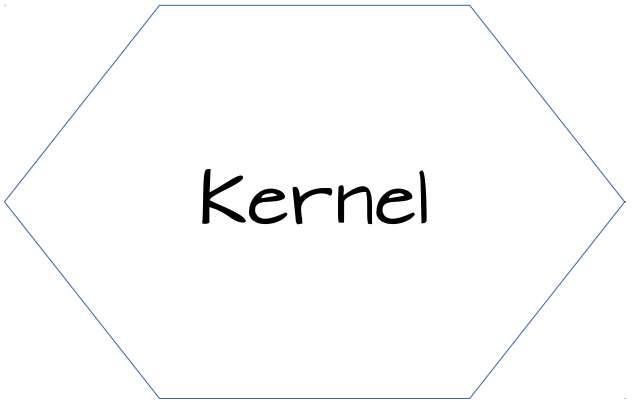
Example

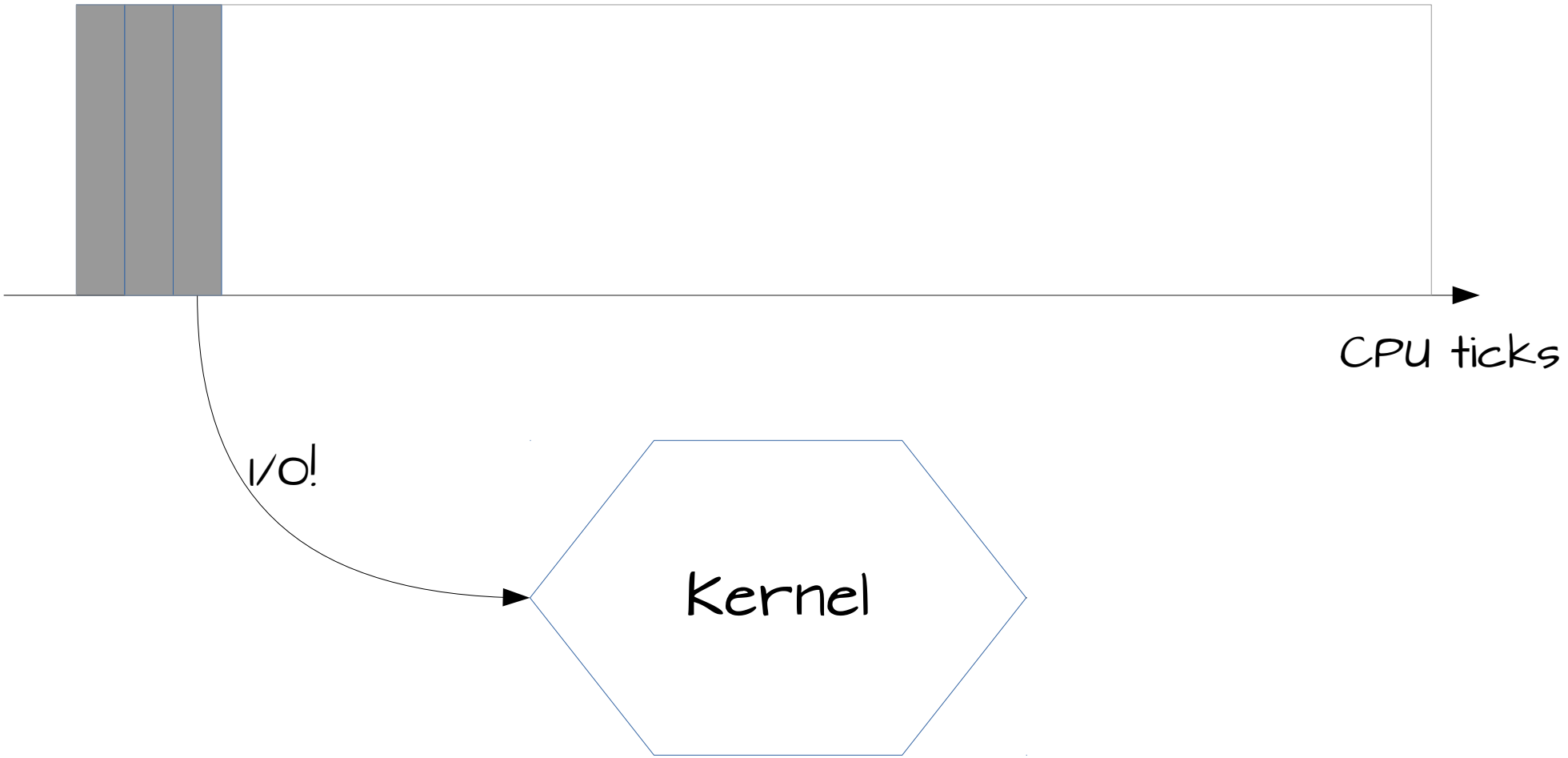
Let's make some HTTP connections.

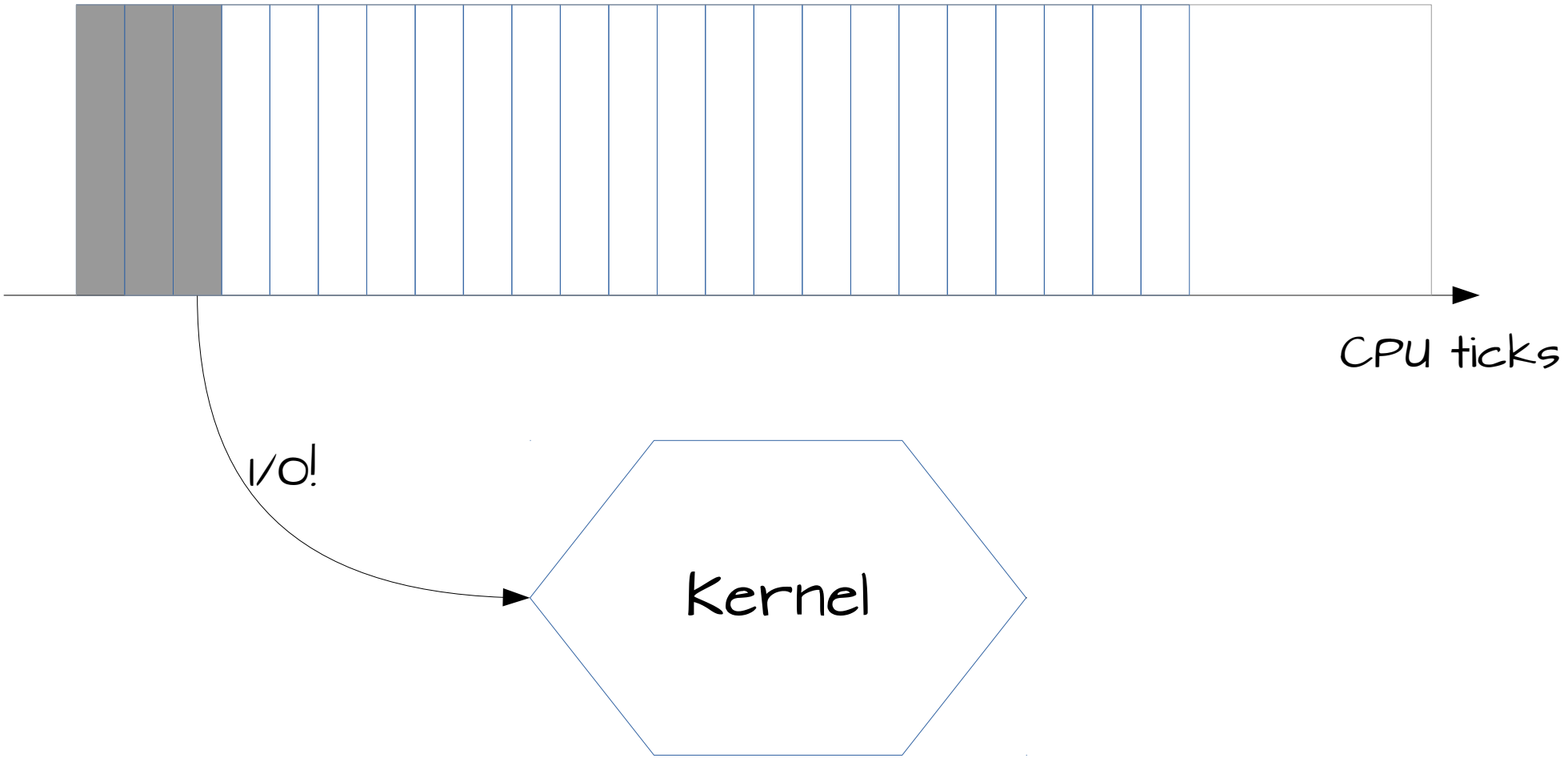
Synchronously

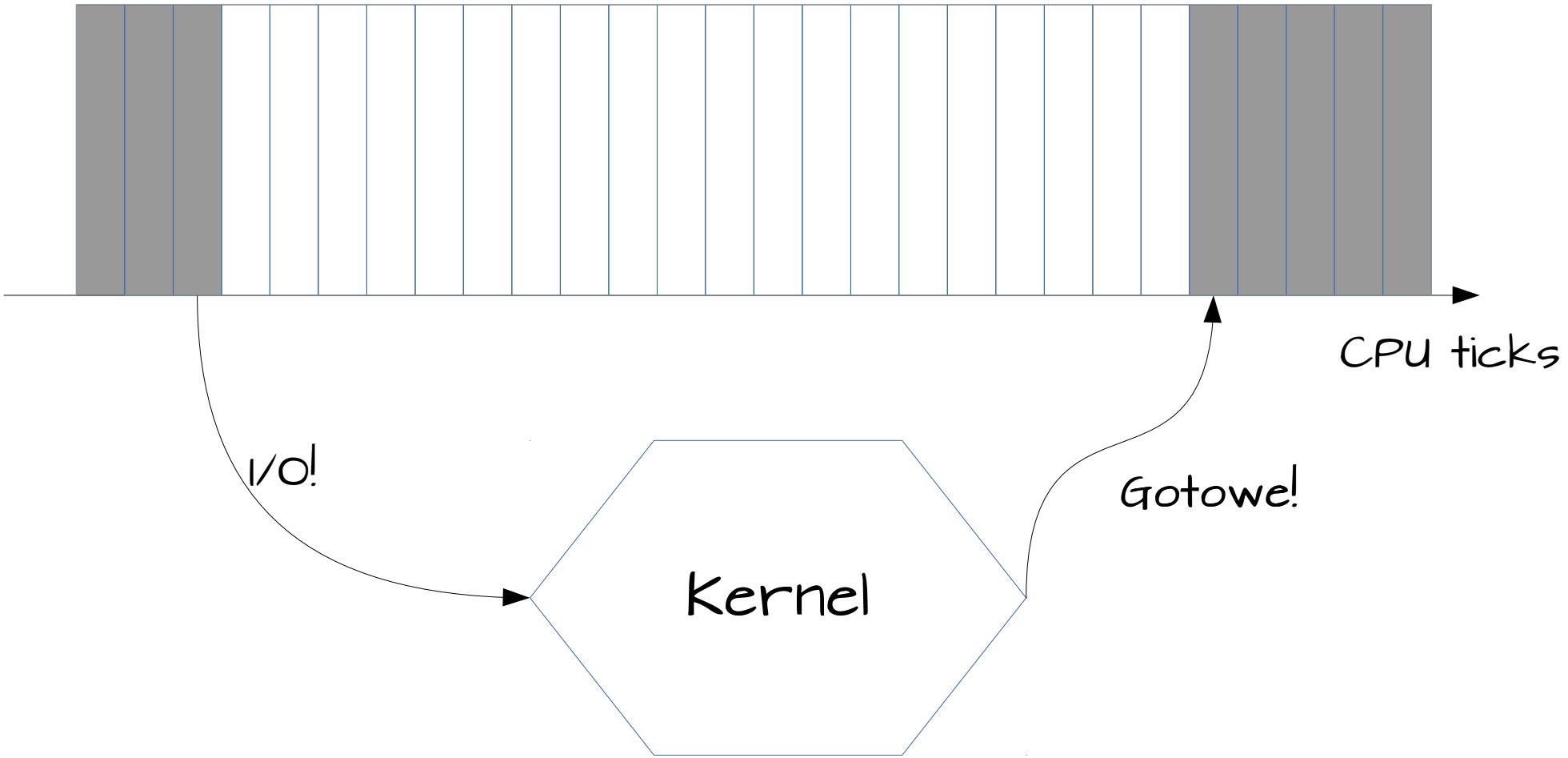
```python
# sync_example.py
import requests


def download(uri):
    response = requests.get(uri)

    return response.text


def print_beginning(uri):
    result = download(uri)

    print(result[:100])


if __name__ == '__main__':
    print_beginning('http://google.pl')
    print_beginning('http://google.fr')
    print_beginning('http://google.hu')
```

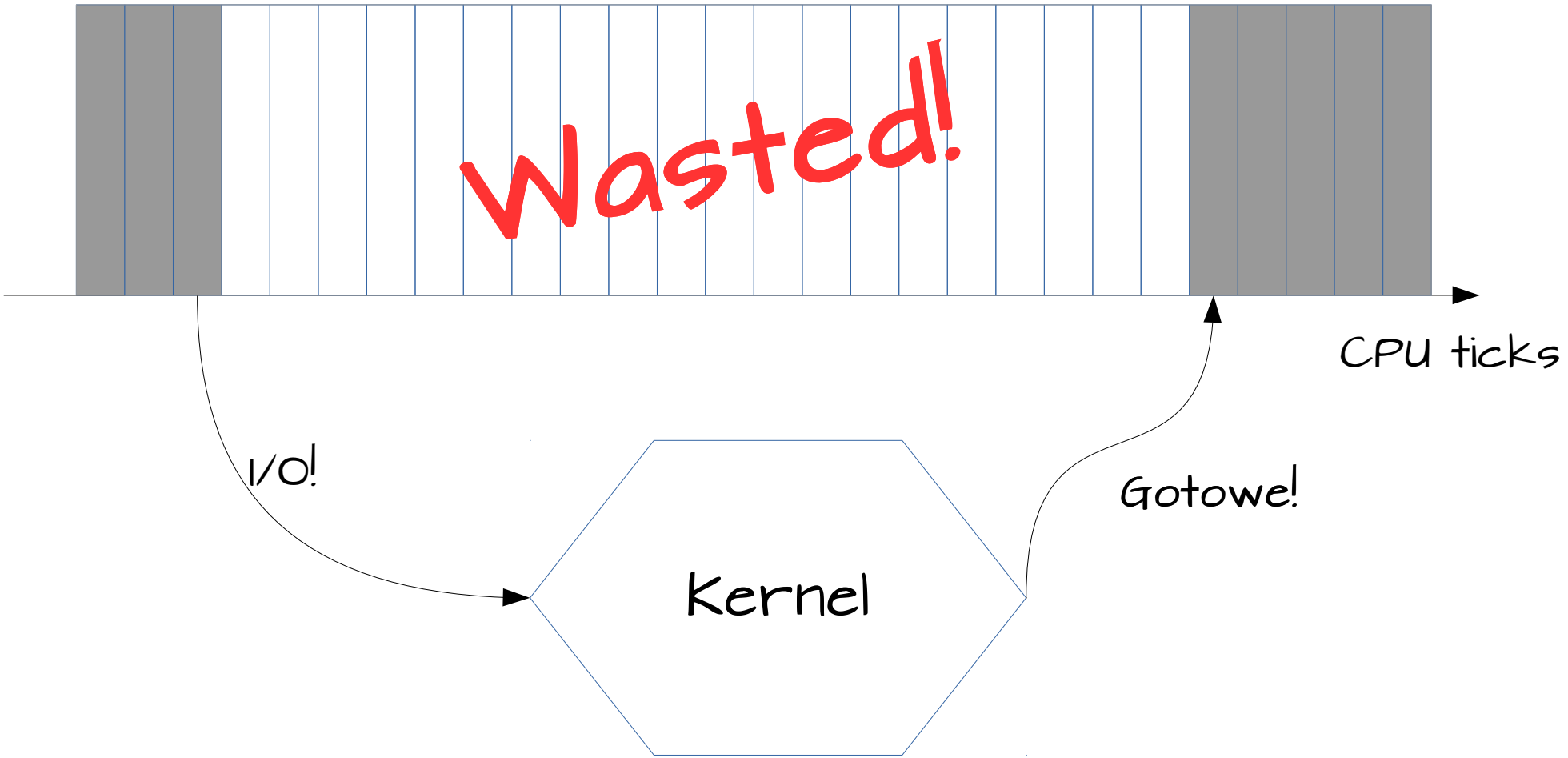

CPU ticks


Kernel


CPU ticks
I/O!
Kernel

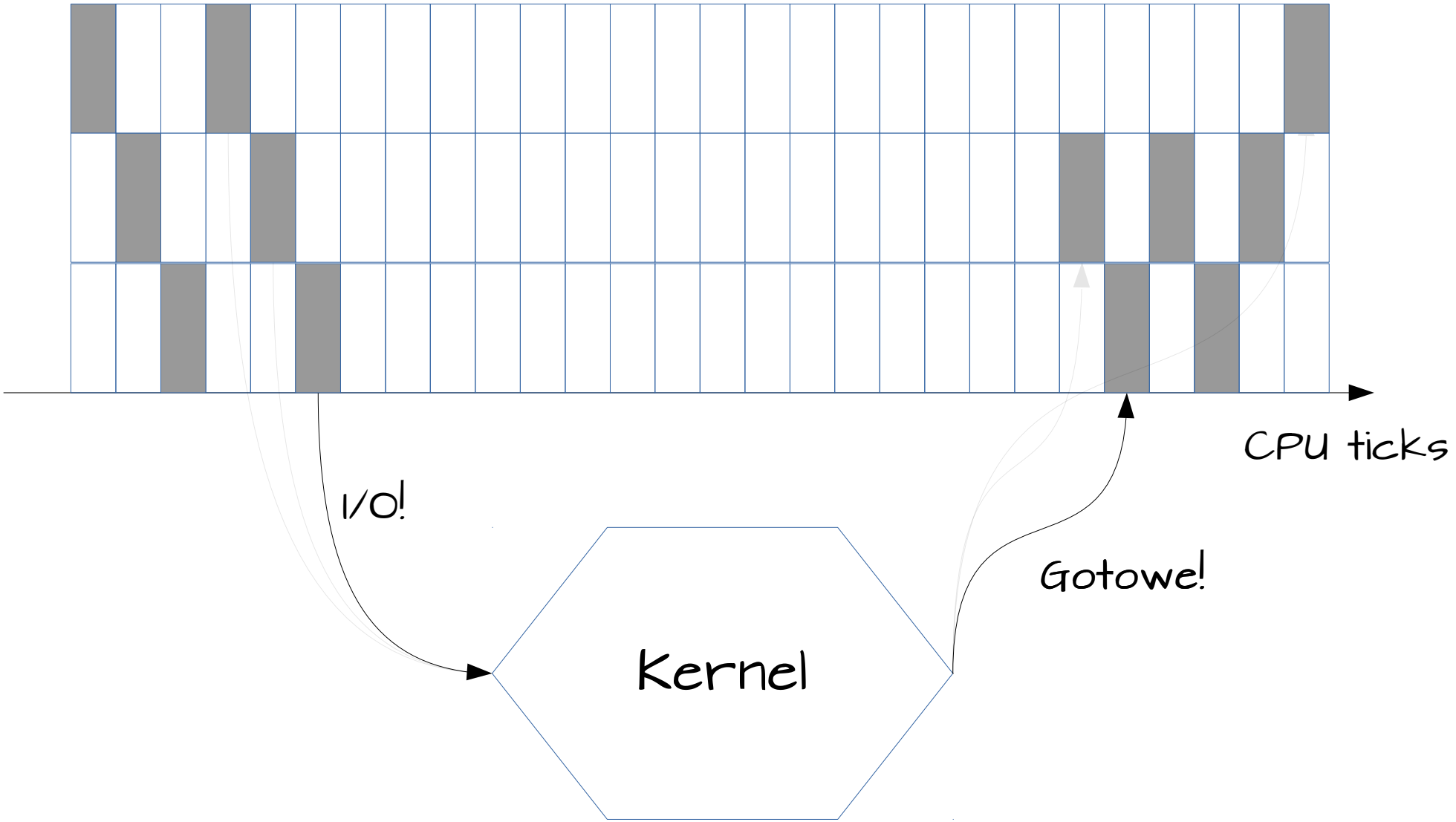

CPU ticks
I/O!
Kernel

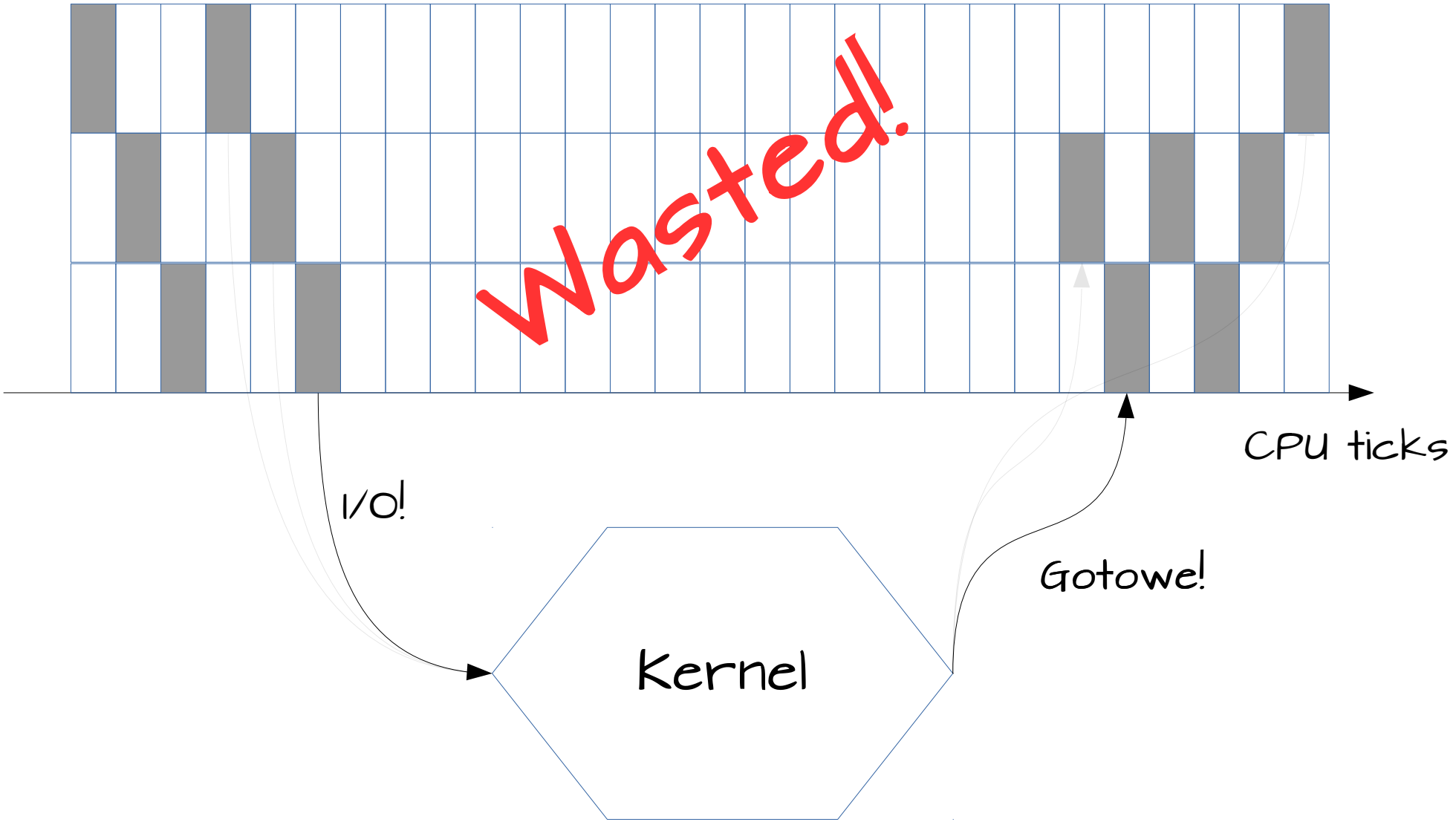

CPU ticks
I/O!
Kernel
Gotowe!


Wasted!
CPU ticks
I/O!
Kernel
Gotowe!

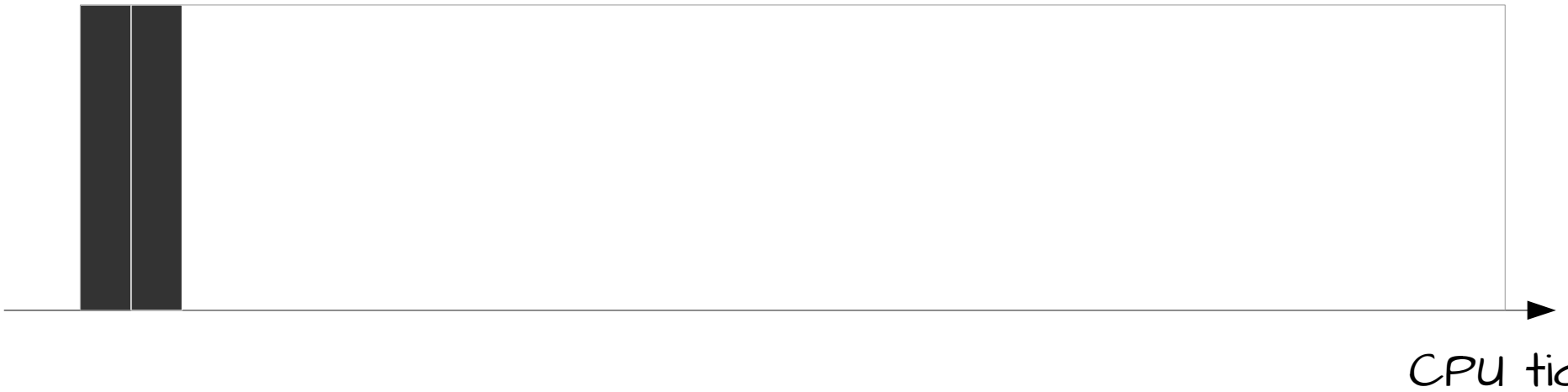
Threads, maybe?

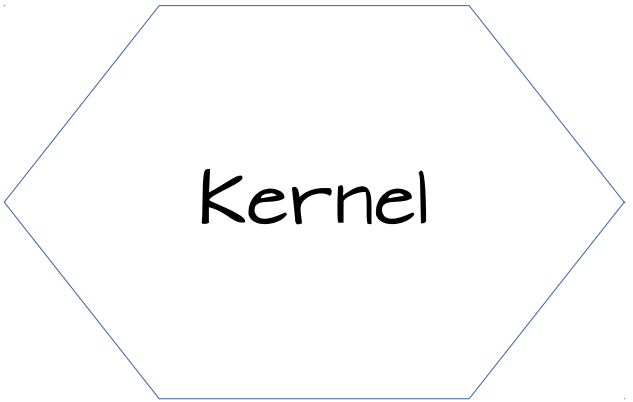
```python
# thread_example.py
import threading


# [ciach!]


if __name__ == '__main__':
    threads = []

    for uri in ('http://google.pl', 'http://google.fr', 'http://google.hu'):
        threads += [threading.Thread(target=print_beginning, args=(uri,))]

    for thread in threads:
        thread.start()

    for thread in threads:
        thread.join()
```

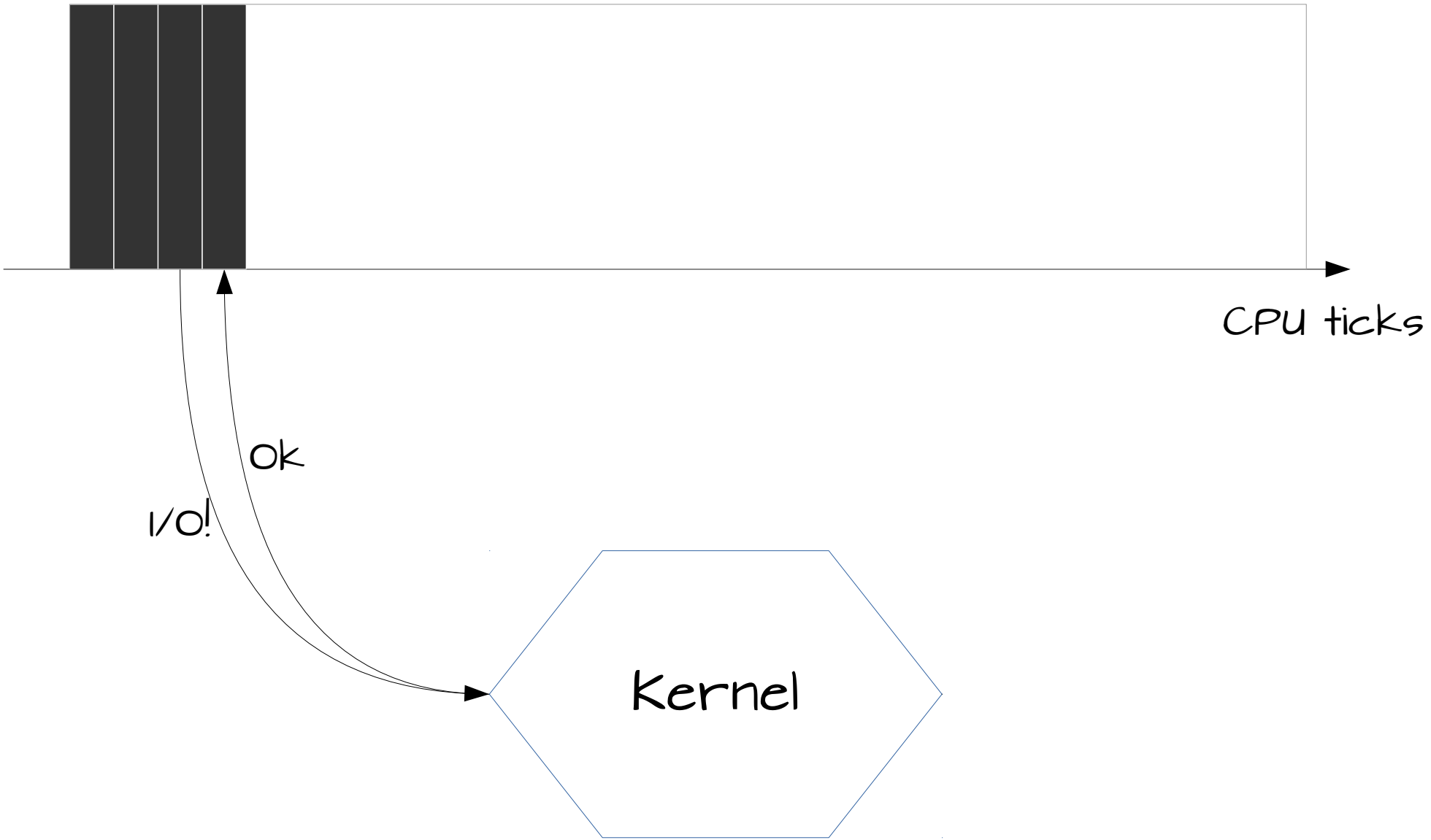

CPU ticks
I/O!
Kernel
Gotowe!

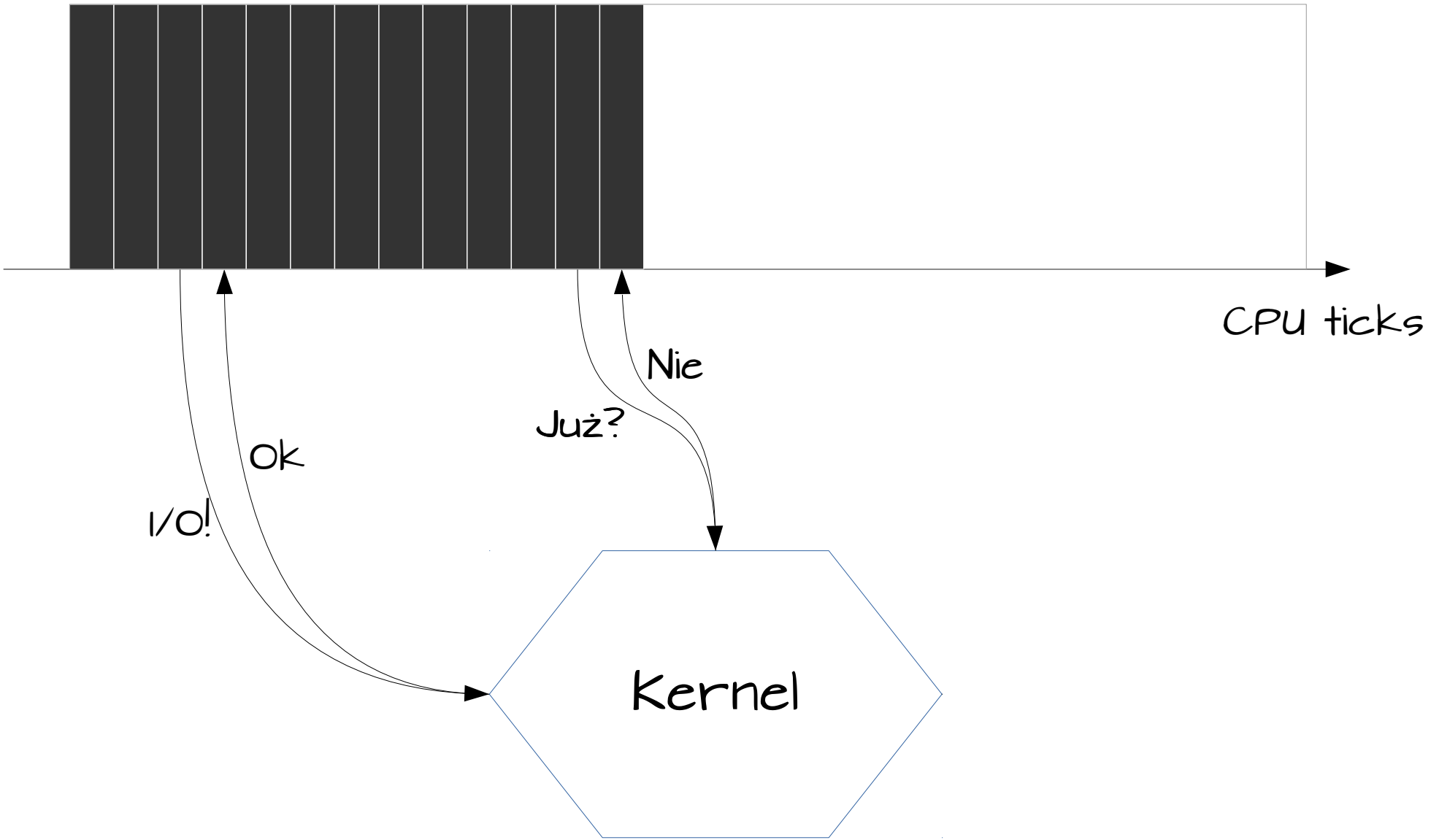

Wasted!
CPU ticks
I/O!
Kernel
Gotowe!

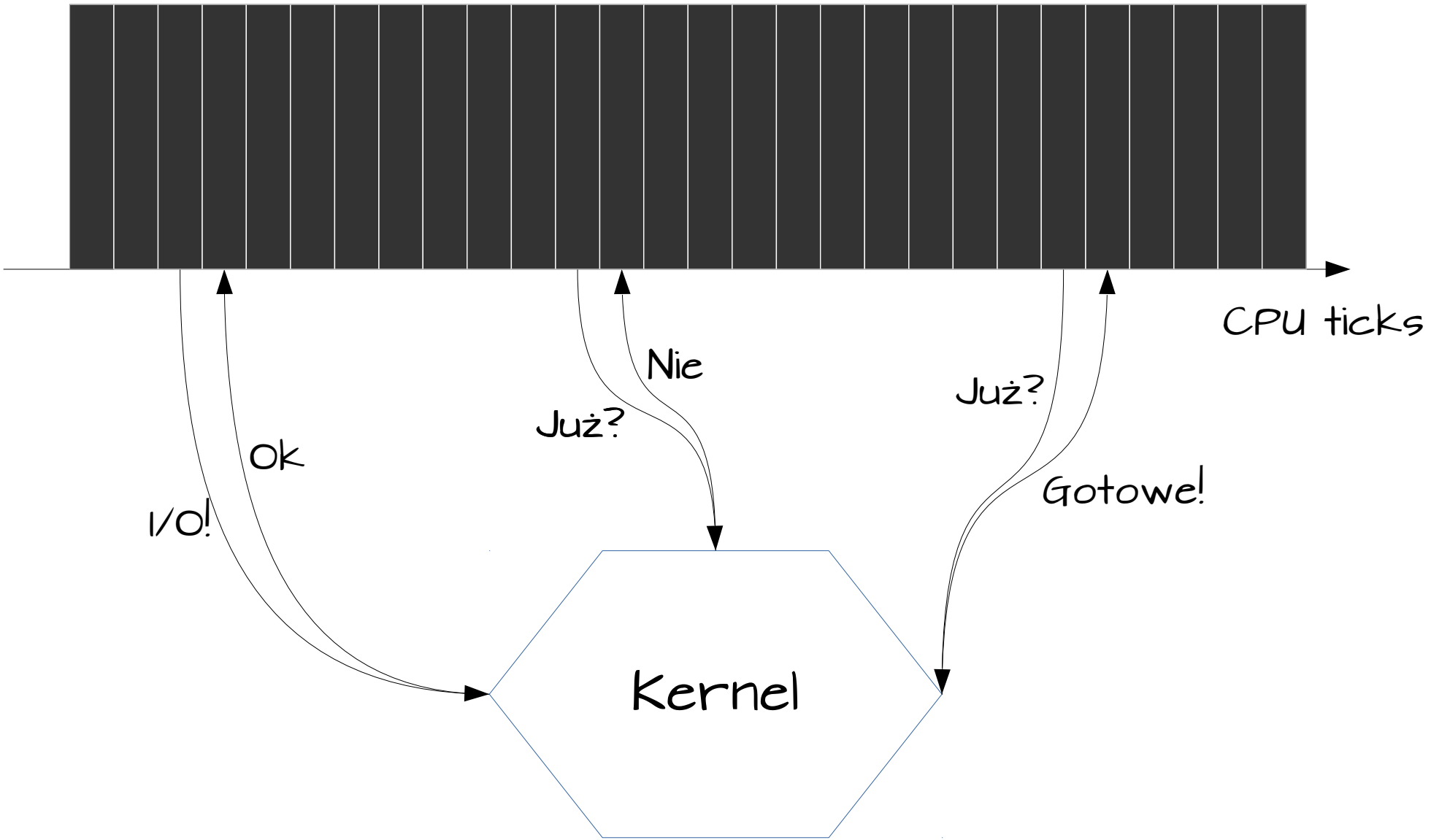
Shall we go async?

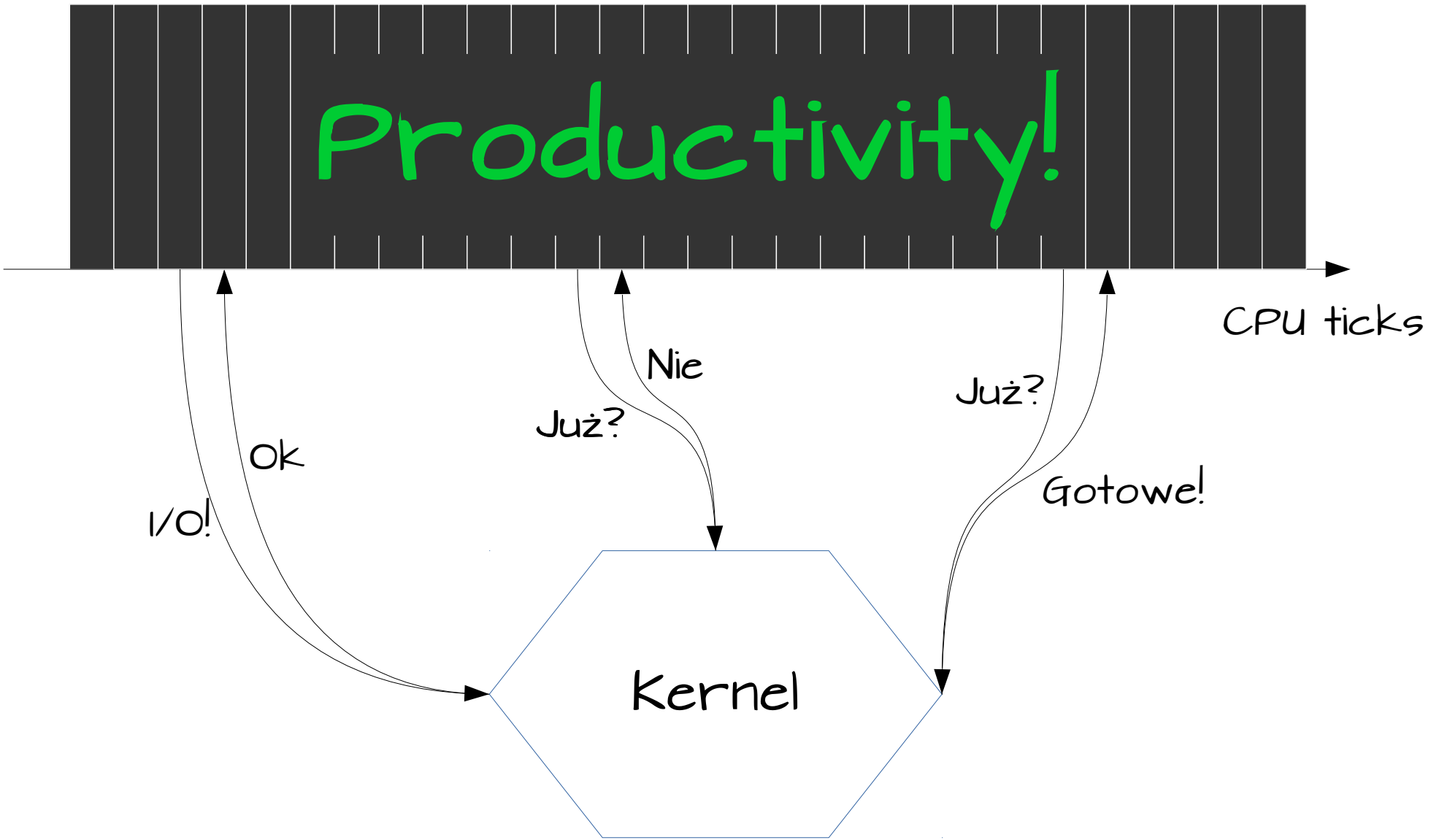

CPU ticks


Kernel


CPU ticks
ok
I/O!
Kernel


CPU ticks
Nie
Już?
ok
I/O!
Kernel


CPU ticks
I/O!
Ok
Już?
Nie
Już?
Gotowe!
Kernel


Productivity!
CPU ticks
I/O!
Ok
Już?
Nie
Już?
Gotowe!
Kernel

Q: So what's it for?

Q: So what's it for?

A: To better utilize the CPU and RAM in I/O bound applications.

Q: Why not in widespread use already?

A: 'cause it makes you change how you write the code (event-style)

Q: Why not in widespread use already?

Q: Why not in widespread use already?

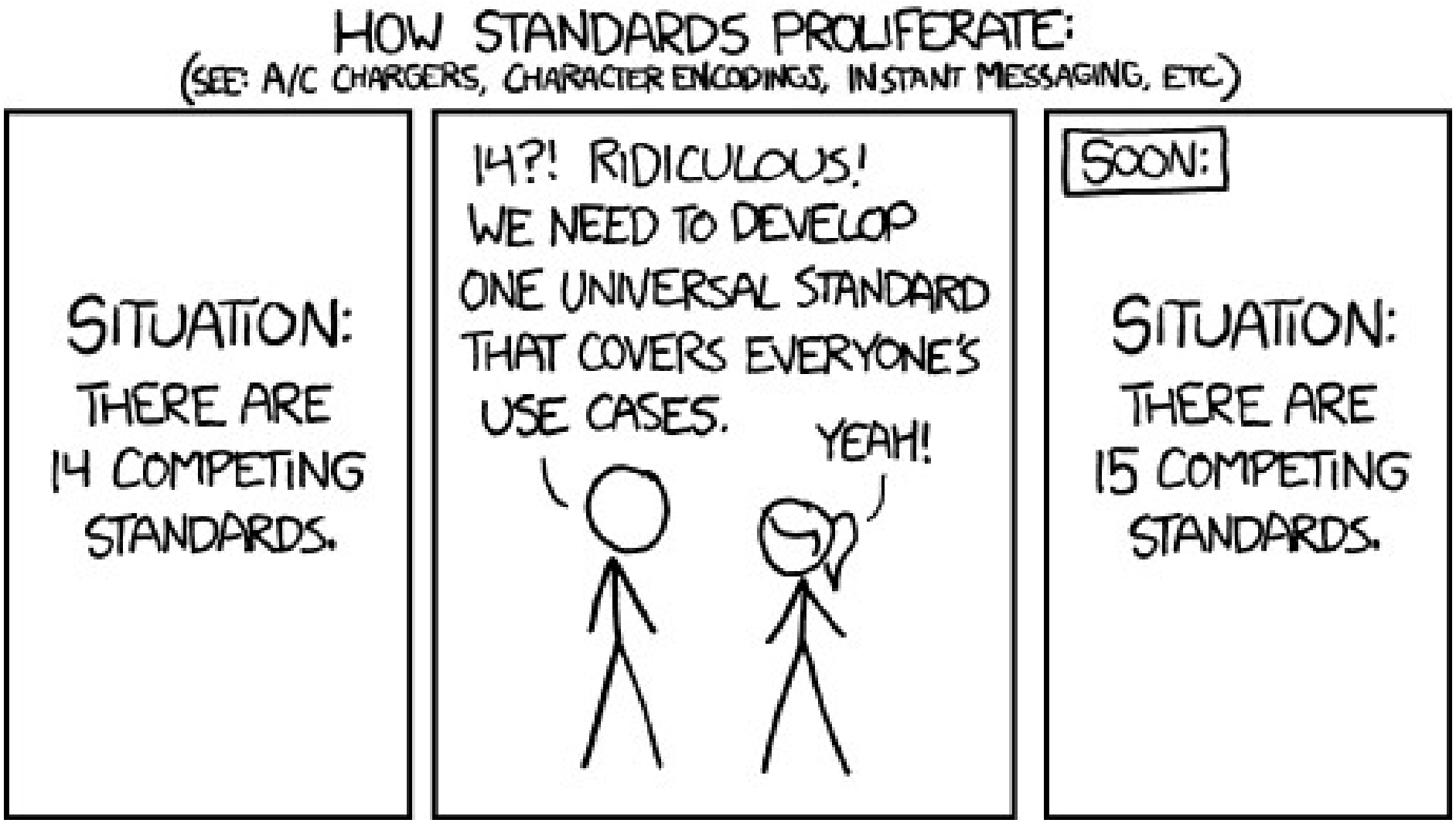
A: 'cause it makes you change how you write the code (event-style)

Q: Is it new thing in Python?

- Diesel
- ~~Orbited~~
- Twisted
- Tornado
- gevents
- Eventlet
- asyncore
- ~~Chiral~~
- ~~Friendly Flow~~
- ~~cogen~~
- weightless

"I'm not trying to reinvent the wheel.
I'm trying to build a good one."

Guido van Rossum



http://xkcd.com/927/

"Good one"?

- readability

"Good one"?

- readability
- interoperability

"Good one"?

- readability
- interoperability
- reliability

```python
# sync_example.py
import requests


def download(uri):
    response = requests.get(uri)

    return response.text


def print_beginning(uri):
    result = download(uri)

    print(str(result[:100]))


if __name__ == '__main__':
    print_beginning('http://google.pl')
    print_beginning('http://google.fr')
    print_beginning('http://google.hu')
```

(sub)generators

```
def sub_generator():
    yield 1


def generator():
    yield from sub_generator()


# Prints '1'.
print(next(generator()))
```

```python
# example.py
import aiohttp
import asyncio


@asyncio.coroutine
def download(uri):
    response = yield from aiohttp.request('GET', uri)

    return (yield from response.read_and_close())


@asyncio.coroutine
def print_beginning(uri):
    result = yield from download(uri)

    print(result.decode('utf-8')[:100])


if __name__ == '__main__':
    loop = asyncio.get_event_loop()

    loop.run_until_complete(
        asyncio.wait(
            [print_beginning('http://google.pl'),
             print_beginning('http://google.fr'),
             print_beginning('http://google.hu'),
             ]))
```

```
import aiohttp
import asyncio


@asyncio.coroutine
def download(uri):
    response = yield from aiohttp.request('GET', uri)

    return (yield from response.read_and_close())


@asyncio.coroutine
def print_beginning(uri):
    result = yield from download(uri)

    print(result.decode('utf-8')[:100])


if __name__ == '__main__':
    loop = asyncio.get_event_loop()

    loop.run_until_complete(
        asyncio.wait(
            [print_beginning('http://google.pl'),
             print_beginning('http://google.fr'),
             print_beginning('http://google.hu'),
             ]))
```

Interior

- coroutines
- futures
- tasks
- event loop

Coroutine

Just a generator

(of async magic)

Coroutine

```python
# coroutine.py
import asyncio


@asyncio.coroutine
def get_example():
    result = yield from asyncio.sleep(2)

    return 'foo'
```

Coroutine

```
import asyncio


@asyncio.coroutine
def get_example():
    result = yield from asyncio.sleep(2)

    return 'foo'
```

Coroutine

```
import asyncio


@asyncio.coroutine
def get_example():
    result = yield from asyncio.sleep(2)

    return 'foo'
```

Coroutine

```python
# coroutine.py
import asyncio


@asyncio.coroutine
def get_example():
    result = yield from asyncio.sleep(2)

    return 'foo'
```

Future

Promise of the result (or error)

Future

```
import asyncio


# Constructing.
future = asyncio.Future()

# Is future done?
assert future.done() is False

# Setting result.
future.set_result('foo')

# It is done now.
assert future.done() is True

# Will print 'foo'.
print(future.result())
```

Future (now with exception!)

```
import asyncio


# Constructing.
future = asyncio.Future()

# Is future done?
assert future.done() is False

# Setting result.
future.set_exception(Exception(':('))

# It is done now.
assert future.done() is True

# Will print str(Exception(':(')).
print(future.exception())

# Will raise Exception(':(').
future.result()
```

Task

- Coroutine, scheduled to execution, wrapped in a Future

Task

- Coroutine, scheduled to execution, wrapped in a Future
- „Future which runs a coroutine"

Task

- Coroutine, scheduled to execution, wrapped in a Future
- „Future which runs a coroutine"
- Future subclass

Task

```
import asyncio


@asyncio.coroutine
def example():
    yield from asyncio.sleep(2)

    print('foo')


if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    loop.run_until_complete(asyncio.Task(example()))
    loop.close()
```

Task

```
import asyncio


@asyncio.coroutine
def example():
    yield from asyncio.sleep(2)

    print('foo')


if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    loop.run_until_complete(asyncio.Task(example()))
    loop.close()
```

Task

```python
# task.py
import asyncio


@asyncio.coroutine
def example():
    yield from asyncio.sleep(2)

    print('foo')


if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    loop.run_until_complete(asyncio.Task(example()))
    loop.close()
```

yield from

```
r = yield from coroutine()
r = yield from future
r = yield from task
```

yield from

```
r = yield from coroutine()
r = yield from future
r = yield from task
```

yield from

```
r = yield from coroutine()
r = yield from future
r = yield from task
```

Event loop

- multiplexer

Event loop

- multiplexer
- connector

Event loop

```
import asyncio


# Get default event loop.
loop = asyncio.get_event_loop()

# Will run until future will be done, duh.
loop.run_until_complete(future)

# Close the event loop.
loop.close()
```

Event loop

```python
import asyncio


# Get default event loop.
loop = asyncio.get_event_loop()

# Will run until stopping somewhere.
loop.run_forever()

# Close the event loop.
loop.close()
```

Event loop

```
import asyncio


# Get default event loop.
loop = asyncio.get_event_loop()

# Schedule to call a callback ASAP.
loop.call_soon(callback, *arguments)

# Schedule to call a callback in...
loop.call_later(10.0, callback, *arguments)

# Schedule to call a callback at...
loop.call_later(datetime.datetime(2014, 10, 16, 19, 00), sys.exit, 41)
```

Event loop

```python
import asyncio


# Get default event loop.
loop = asyncio.get_event_loop()

# Execute in a thread.
loop.call_soon_threadsafe(callback, *arguments)
```

```python
# example.py
import aiohttp
import asyncio


@asyncio.coroutine
def download(uri):
    response = yield from aiohttp.request('GET', uri)

    return (yield from response.read_and_close())


@asyncio.coroutine
def print_beginning(uri):
    result = yield from download(uri)

    print(result.decode('utf-8')[:100])


if __name__ == '__main__':
    loop = asyncio.get_event_loop()

    loop.run_until_complete(
        asyncio.wait(
            [print_beginning('http://google.pl'),
             print_beginning('http://google.fr'),
             print_beginning('http://google.hu'),
             ]))
```

```python
import aiohttp
import asyncio


@asyncio.coroutine
def download(uri):
    response = yield from aiohttp.request('GET', uri)

    return (yield from response.read_and_close())


@asyncio.coroutine
def print_beginning(uri):
    result = yield from download(uri)

    print(result.decode('utf-8')[:100])


if __name__ == '__main__':
    loop = asyncio.get_event_loop()

    loop.run_until_complete(
        asyncio.wait(
            [print_beginning('http://google.pl'),
             print_beginning('http://google.fr'),
             print_beginning('http://google.hu'),
             ]))
```

```
import aiohttp
import asyncio


@asyncio.coroutine
def download(uri):
    response = yield from aiohttp.request('GET', uri)

    return (yield from response.read_and_close())


@asyncio.coroutine
def print_beginning(uri):
    result = yield from download(uri)

    print(result.decode('utf-8')[:100])


if __name__ == '__main__':
    loop = asyncio.get_event_loop()

    loop.run_until_complete(
        asyncio.wait(
            [print_beginning('http://google.pl'),
             print_beginning('http://google.fr'),
             print_beginning('http://google.hu'),
             ]))
```

```
import aiohttp
import asyncio


@asyncio.coroutine
def download(uri):
    response = yield from aiohttp.request('GET', uri)

    return (yield from response.read_and_close())


@asyncio.coroutine
def print_beginning(uri):
    result = yield from download(uri)

    print(result.decode('utf-8')[:100])


if __name__ == '__main__':
    loop = asyncio.get_event_loop()

    loop.run_until_complete(
        asyncio.wait(
            [print_beginning('http://google.pl'),
             print_beginning('http://google.fr'),
             print_beginning('http://google.hu'),
             ]))
```

```
import aiohttp
import asyncio


@asyncio.coroutine
def download(uri):
    response = yield from aiohttp.request('GET', uri)

    return (yield from response.read_and_close())


@asyncio.coroutine
def print_beginning(uri):
    result = yield from download(uri)

    print(result.decode('utf-8')[:100])


if __name__ == '__main__':
    loop = asyncio.get_event_loop()

    loop.run_until_complete(
        asyncio.wait(
            [print_beginning('http://google.pl'),
             print_beginning('http://google.fr'),
             print_beginning('http://google.hu'),
             ]))
```

```
import aiohttp
import asyncio


@asyncio.coroutine
def download(uri):
    response = yield from aiohttp.request('GET', uri)

    return (yield from response.read_and_close())


@asyncio.coroutine
def print_beginning(uri):
    result = yield from download(uri)

    print(result.decode('utf-8')[:100])


if __name__ == '__main__':
    loop = asyncio.get_event_loop()

    loop.run_until_complete(
        asyncio.wait(
            [print_beginning('http://google.pl'),
             print_beginning('http://google.fr'),
             print_beginning('http://google.hu'),
             ]))
```

```
import aiohttp
import asyncio


@asyncio.coroutine
def download(uri):
    response = yield from aiohttp.request('GET', uri)

    return (yield from response.read_and_close())


@asyncio.coroutine
def print_beginning(uri):
    result = yield from download(uri)

    print(result.decode('utf-8')[:100])


if __name__ == '__main__':
    loop = asyncio.get_event_loop()

    loop.run_until_complete(
        asyncio.wait(
            [print_beginning('http://google.pl'),
             print_beginning('http://google.fr'),
             print_beginning('http://google.hu'),
             ]))
```

```
import aiohttp
import asyncio


@asyncio.coroutine
def download(uri):
    response = yield from aiohttp.request('GET', uri)

    return (yield from response.read_and_close())


@asyncio.coroutine
def print_beginning(uri):
    result = yield from download(uri)

    print(result.decode('utf-8')[:100])


if __name__ == '__main__':
    loop = asyncio.get_event_loop()

    loop.run_until_complete(
        asyncio.wait(
            [print_beginning('http://google.pl'),
             print_beginning('http://google.fr'),
             print_beginning('http://google.hu'),
             ]))
```

```
import aiohttp
import asyncio


@asyncio.coroutine
def download(uri):
    response = yield from aiohttp.request('GET', uri)

    return (yield from response.read_and_close())


@asyncio.coroutine
def print_beginning(uri):
    result = yield from download(uri)

    print(result.decode('utf-8')[:100])


if __name__ == '__main__':
    loop = asyncio.get_event_loop()

    loop.run_until_complete(
        asyncio.wait(
            [print_beginning('http://google.pl'),
             print_beginning('http://google.fr'),
             print_beginning('http://google.hu'),
             ]))
```

```python
# example.py
import aiohttp
import asyncio


@asyncio.coroutine
def download(uri):
    response = yield from aiohttp.request('GET', uri)

    return (yield from response.read_and_close())


@asyncio.coroutine
def print_beginning(uri):
    result = yield from download(uri)

    print(result.decode('utf-8')[:100])


if __name__ == '__main__':
    loop = asyncio.get_event_loop()

    loop.run_until_complete(
        asyncio.wait(
            [print_beginning('http://google.pl'),
             print_beginning('http://google.fr'),
             print_beginning('http://google.hu'),
             ]))
```

No to działamy!

```
$ python example.py
...
```

No to działamy!

```
$ python example.py
```

Stop!

Testy!

async_test

```
# aiotest.py
import asyncio
import functools


# Brought from
# http://stackoverflow.com/questions/23033939/how-to-test-python-3-4-asyncio-code
def async_test(f):
    @functools.wraps(f)
    def wrapper(*args, **kwargs):
        coroutine = asyncio.coroutine(f)
        future = coroutine(*args, **kwargs)
        loop = asyncio.get_event_loop()
        loop.run_until_complete(future)

    return wrapper
```

async_test

```python
# test_example_.py
import aiotest
import unittest

import example


class MyTestCase(unittest.TestCase):
    @aiotest.async_test
    def test_download(self):
        result = yield from example.download('http://google.pl')

        result = result.decode('utf-8')[:30]

        self.assertEqual(result, '<!doctype html><html itemscope')
```

asyncio-nous TestCase

```python
# aiotest.py
import unittest


# [ciach!]


class AsyncMetaClass(type):
    def __new__(cls, name, bases, local):
        for attribute_name, attribute in local.items():
            if attribute_name.startswith('test_') and callable(attribute):
                local[attribute_name] = async_test(attribute)

        return type.__new__(cls, name, bases, local)


class TestCase(unittest.TestCase, metaclass=AsyncMetaClass):
    pass
```

ExampleTestCase

```python
# test_example.py
import aiotest

import example


class ExampleTestCase(aiotest.TestCase):
    def test_download(self):
        result = yield from example.download('http://google.pl')

        result = result.decode('utf-8')[:30]

        self.assertEqual(result, '<!doctype html><html itemscope')
```

Testy

```
$ python -m unittest
.
----------------------------------------
Ran 1 tests in 0.001s

OK
```

Only HTTP?

Of course not.

I'm network library,

not a web framework.

Interior pt. 2

- protocols
- transports

Yep, just like in Twisted.

Protocol

- HTTP
- AMPQ
- memcache
- PostreSQL
- MySQL
- ...

Protocol

```
import asyncio


class ExampleProtocol(asyncio.Protocol):
    def connection_made(self, transport):
        transport.write('Hello world'.encode())

    def data_received(self, data):
        transport.write(':)'.encode())
        print('received: {!r}'.format(data.decode()))
```

Transport

- TCP
- UDP
- SSL
- subprocess pipes

Transport

```
import asyncio


class ExampleTransport(asyncio.BaseTransport):
    def write(self, data):
        …

    def close(self):
        …
```

Interior pt. 3

- event loop policies

Interior pt. 3

- event loop policies
- synchronisation
  - locks
  - semaphores
  - queues, priority queues
  - ...

W prawdziwym świecie

HTTP

```
# third_party/aiohttp_example.py
import aiohttp
import asyncio


@asyncio.coroutine
def get_body(url):
    response = yield from aiohttp.request('GET', url)

    return (yield from response.read_and_close())
```

Redis

```
# third_party/aioredis_example.py
import aioredis
import asyncio


@asyncio.coroutine
def example():
    client = yield from aioredis.create_redis(('localhost', 6379))

    yield from client.set(b'aio-key', b'Value')

    value = yield from client.get(b'aio-key')

    yield from client.delete(b'aio-key')

    return value
```

memcached

```
# third_party/aiomcache_example.py
import aiomcache
import asyncio


@asyncio.coroutine
def example():
    client = aiomcache.Client('localhost', 11211)

    yield from client.set(b'aio-key', b'Value')

    value = yield from client.get(b'aio-key')

    yield from client.delete(b'aio-key')

    return value
```

PostgreSQL

```
# third_party/aiopg_example.py
import asyncio
import aiopg


dsn = 'dbname=foo user=bar host=127.0.0.1'


@asyncio.coroutine
def example():
    pool = yield from aiopg.create_pool(dsn)

    with (yield from pool.cursor()) as cur:
        yield from cur.execute('SELECT foo FROM bar')

        return (yield from cur.fetchone())
```

MongoDB

```
# third_party/asyncio_mongo_example.py
import asyncio
import asyncio_mongo


@asyncio.coroutine
def example():
    client = yield from asyncio_mongo.Connection.create('localhost', 27017)

    foo = client.foo  # `foo` database
    test = foo.test  # `test` collection

    return (yield from test.find(limit=10))
```

DNS

```
# third_party/aiodns_example.py
import asyncio
import aiodns


@asyncio.coroutine
def example():
    resolver = aiodns.DNSResolver()

    return (yield from resolver.query('google.com', 'A'))
```

WebSockets

```
# third_party/websockets_example.py
import asyncio
import websockets


@asyncio.coroutine
def hello():
    websocket = yield from websockets.connect('ws://localhost:8888/')

    name = input("What's your name? ")

    yield from websocket.send(name)

    print('> {}'.format(name))

    greeting = yield from websocket.recv()

    print('< {}'.format(greeting))
```

Web framework

```
# third_party/rainfall_example.py
import asyncio
from rainfall.web import Application, HTTPHandler


class RootHandler(HTTPHandler):
    @asyncio.coroutine
    def handle(self, request):
        return('Hello, world!')


app = Application({
    r'^/$': RootHandler(),
    },
)


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

Pulsar

"Concurrent framework for Python"

Python 3.3

```
$ pip install asyncio
```

Python <= 3.2

```
$ pip install trollius
```

```
import trollius
from trollius import From, Return


@trollius.coroutine
def some_coroutine():
    yield From(other_coroutine())

    raise Return('foo')
```

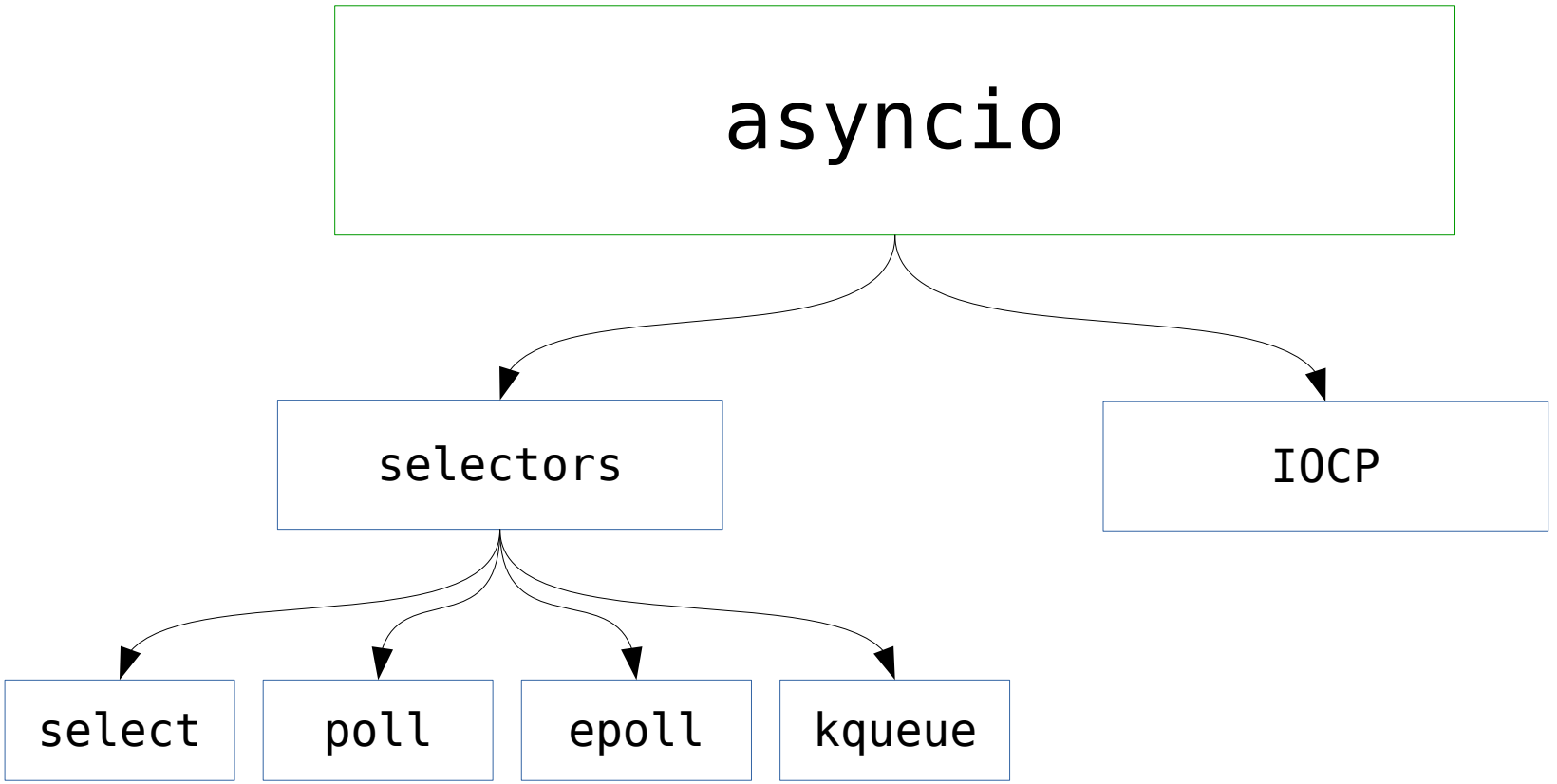
Python <= 3.2

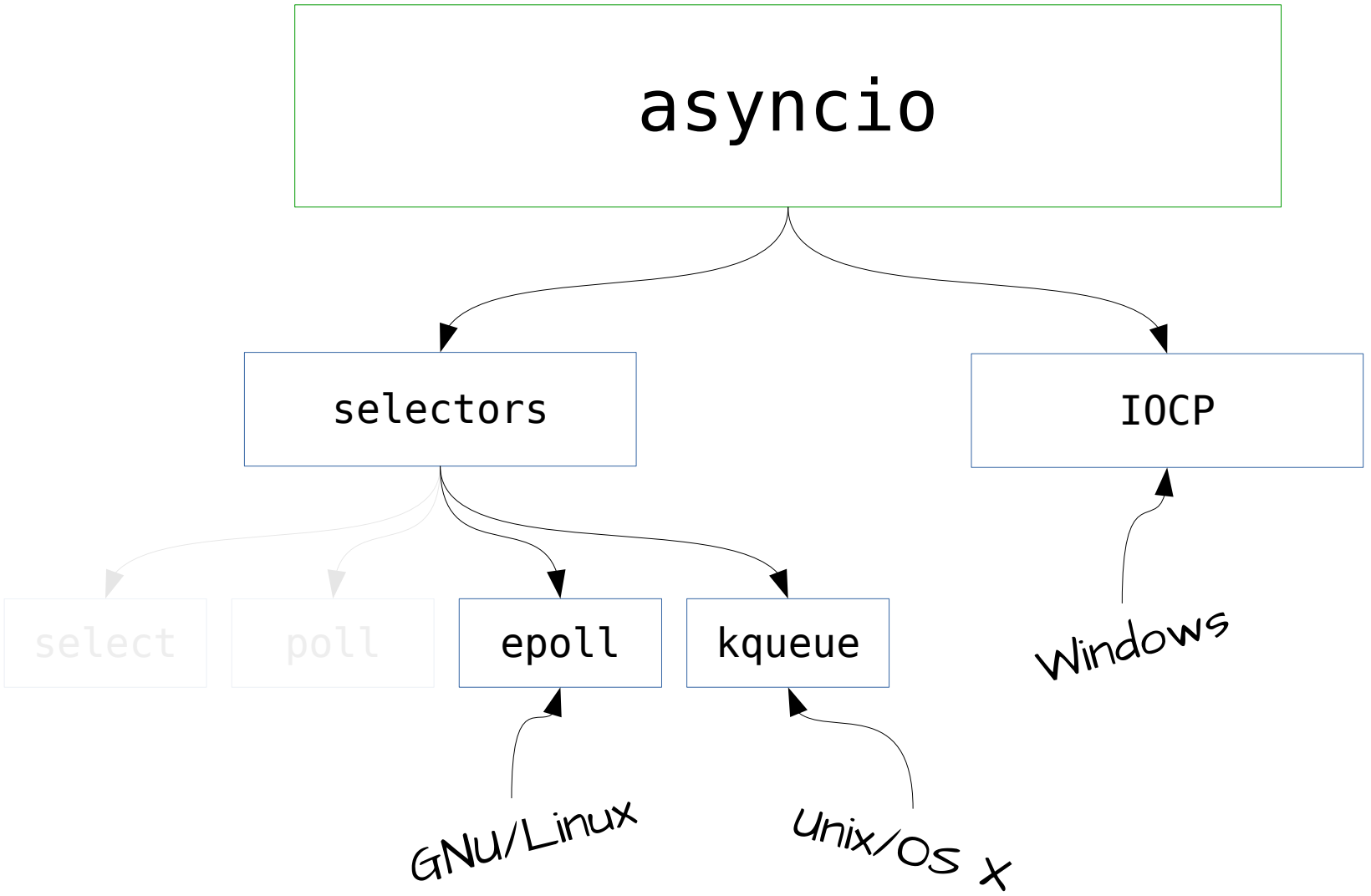
```
$ pip install trollius
```

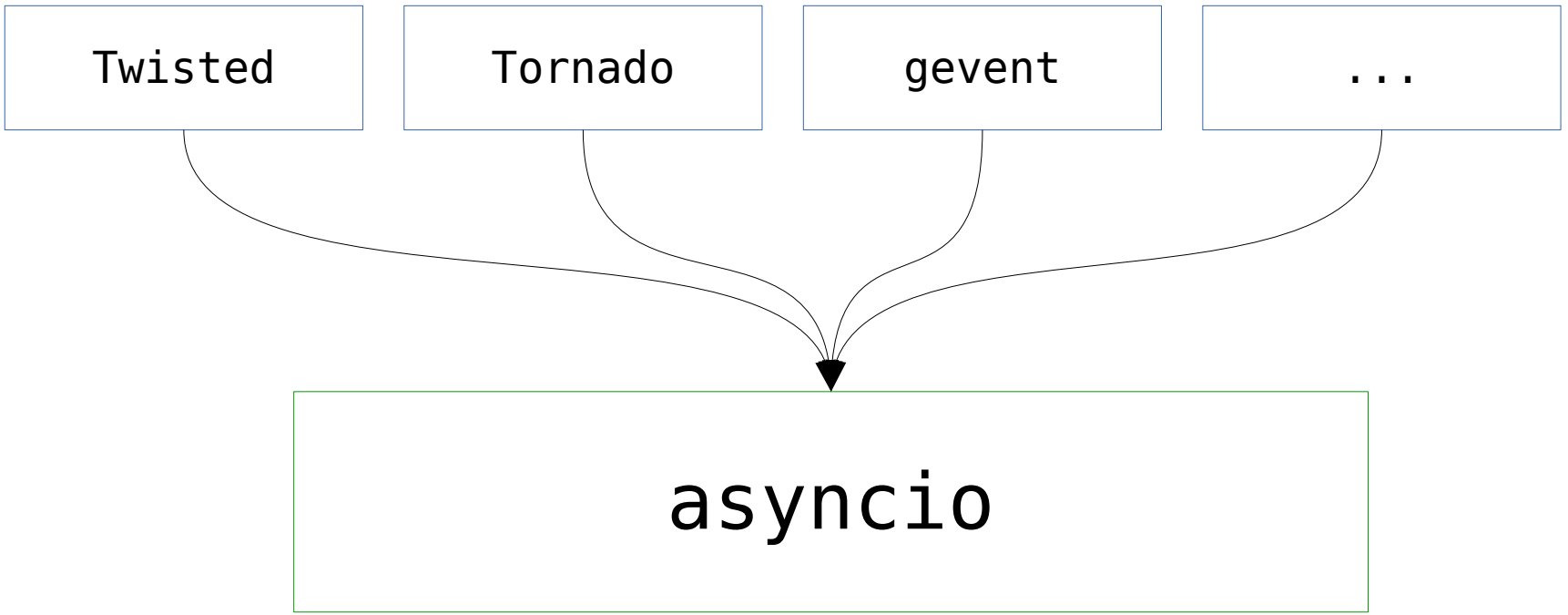
```
import trollius
from trollius import From, Return


@trollius.coroutine
def some_coroutine():
    yield From(other_coroutine())

    raise Return('foo')
```

Python >= 2.6!

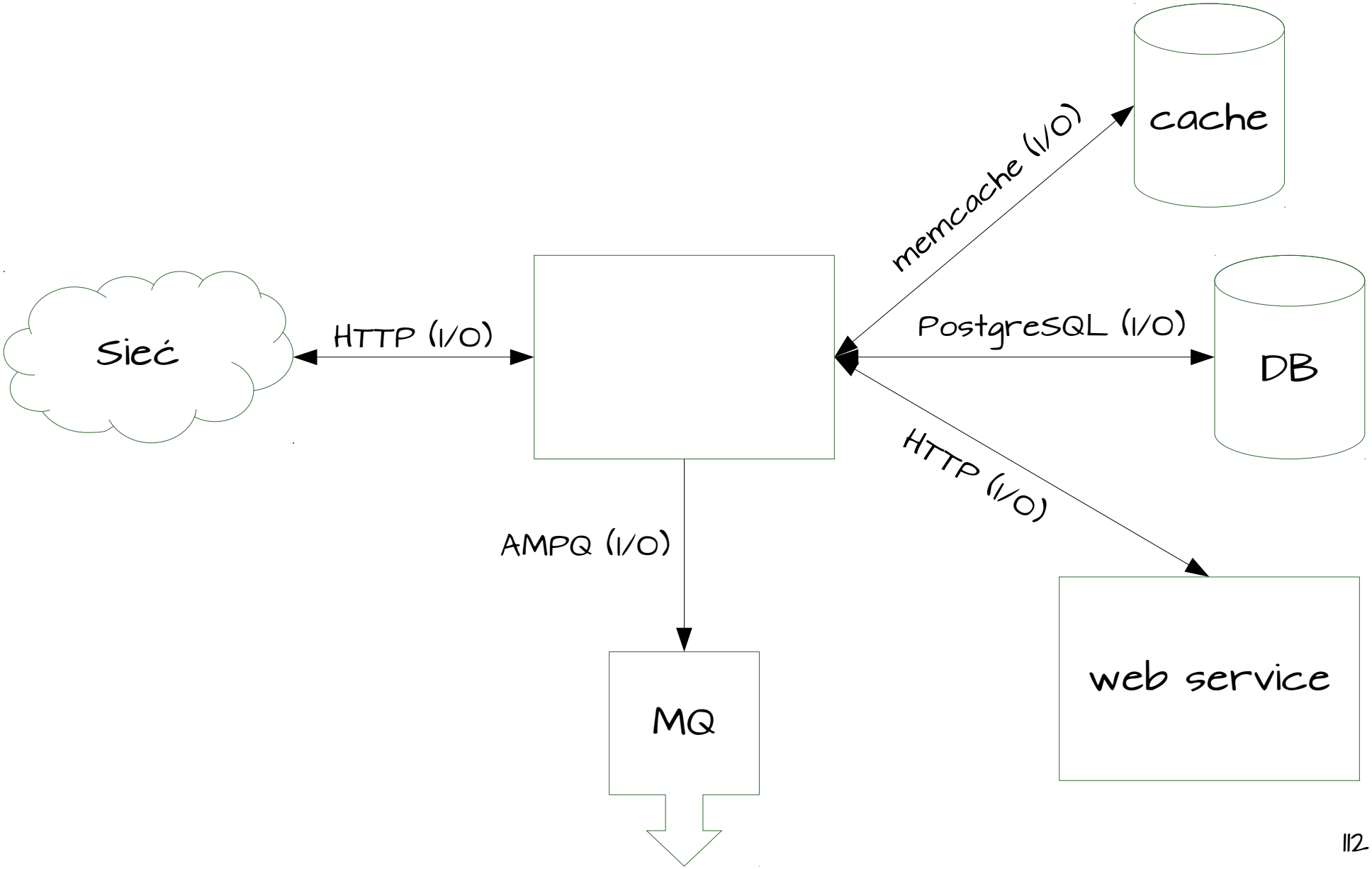
Interoperability


asyncio
selectors
IOCP
select
poll
epoll
kqueue


asyncio
selectors
IOCP
select
poll
epoll
kqueue
Windows
GNU/Linux
Unix/OS X


Twisted
Tornado
gevent
...
asyncio

Nasz use case



Nasz use case

[Insert benchmarks]

Panaceum?

Panaceum?

Nie. Ale...

Pytania?

Dzięki :)

- kolodziejj.info/asyncio/

- kegel.com/c10k.html
- legacy.python.org/dev/peps/pep-3156/
- slideshare.net/saghul/asyncio
- blog.kgriffs.com/2012/09/18/demystifying-async-io.html
- youtube.com/watch?v=1coLC-MUCJc
- kolodziejj.info/asyncio/