Bringing network concurrency to the sync desert

Concurrent IO & Python

Aitor Guevara • @aitorciki

Concurrency

handling

many things at once

not necessarily

doing

them simultaneously

http://youtu.be/cN_DpYBzKso

Concurrency is not Parallelism

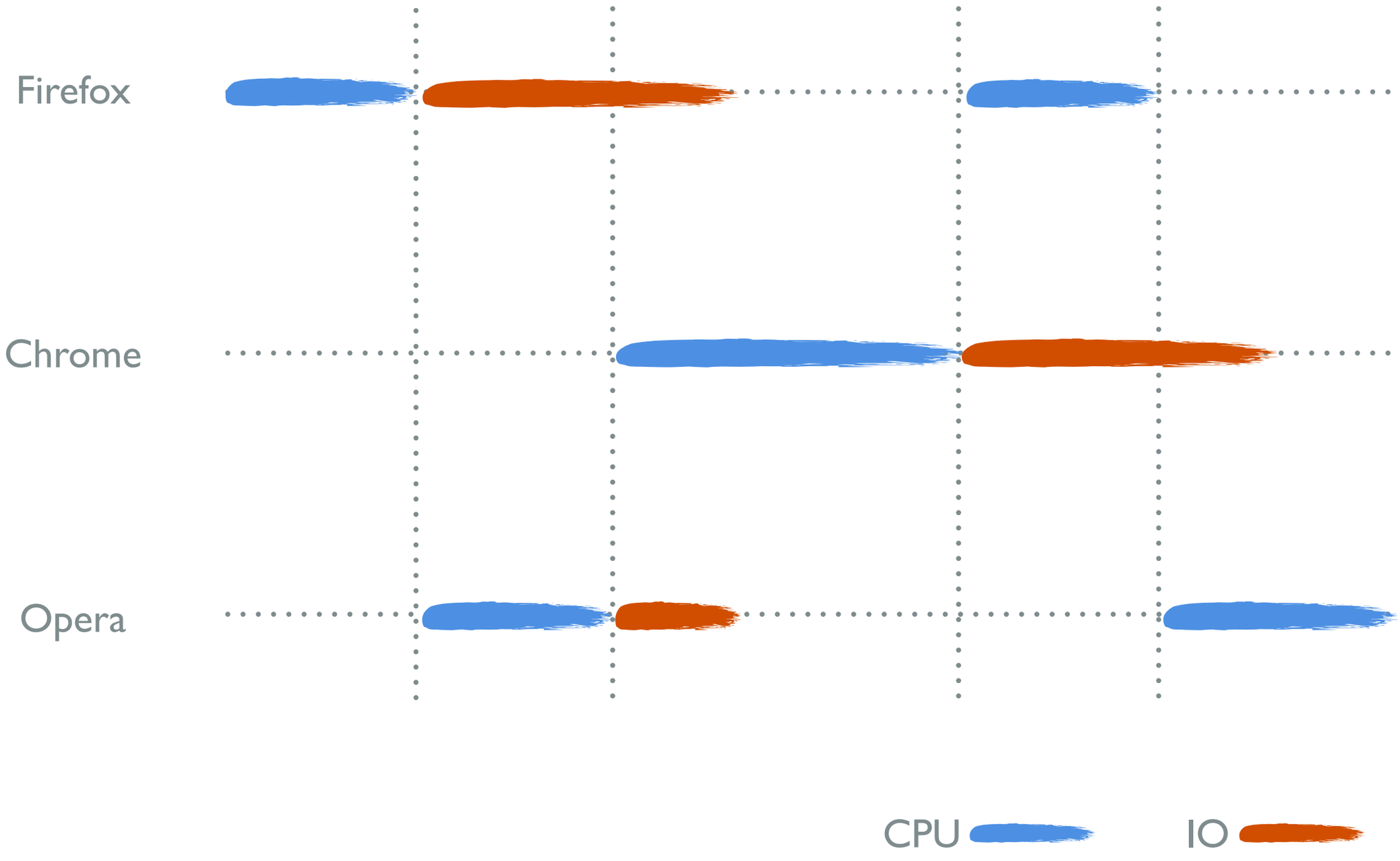
Rob Pike

more

work

done

e.g.: Single-CPU OS scheduler



Synchronous IO

```
import urllib2

urls = (
    'http://twitter.com/',
    'http://google.com/',
    'http://yahoo.com/',
    'http://facebook.com/'
)

for url in urls:
    resp = urllib2.urlopen(url)
    print url, 'OK' if resp.code == 200 else 'Bad'
```

one

thing at a time

most time

waiting

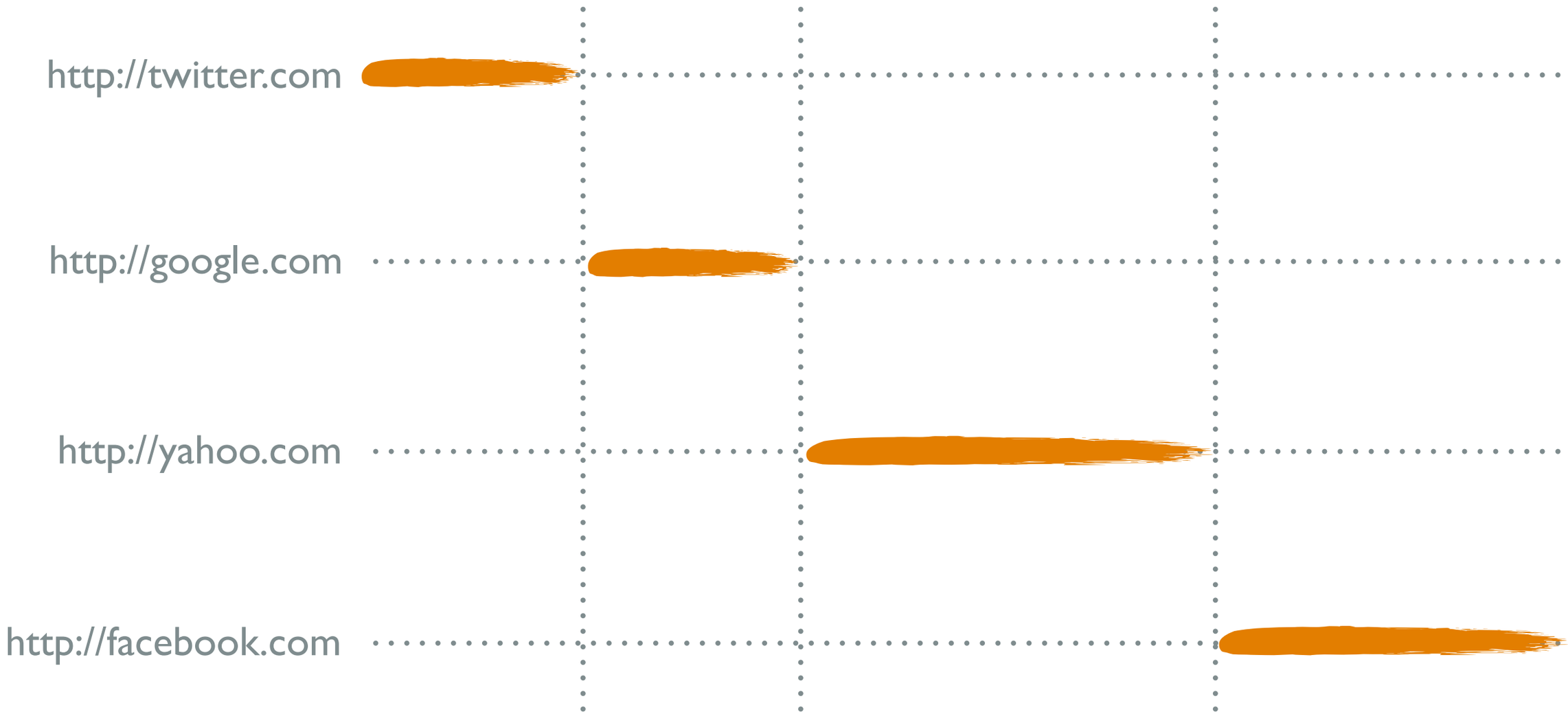
on IO

wasted

resources

usually

slow


http://twitter.com
http://google.com
http://yahoo.com
http://facebook.com

Goal: Async IO

http://twitter.com

http://google.com

http://yahoo.com

http://facebook.com

Threads

OS

threads

no

parallelism

because of GIL

multiprocessing alternative

tricky

scheduling

due to GIL

http://www.dabeaz.com/GIL/

Understanding the Python GIL

David Beazley

```python
import threading
import urllib2

urls = (
    'http://twitter.com/',
    'http://google.com/',
    'http://yahoo.com/',
    'http://facebook.com/'
)

def fetch(url):
    resp = urllib2.urlopen(url)
    print url, 'OK' if resp.code == 200 else 'Bad'

for url in urls:
    thread = threading.Thread(target=fetch, args=(url,))
    thread.start()
```

- familiar API
- easy, sync IO

- performance & resources
- complex synchronization

Twisted

event loop

callbacks

explicit event loop

the reactor

chainable

deferreds

returns a deferred

```
def do_async_stuff():
    d = async_operation()
    d.addCallback(handle_result)
    d.addCallback(further_process)
    d.addErrback(handle_exception)
    return d

d = do_async_stuff()
d.addCallback(...)
```

same deferred,
callbacks chained behind
do_async_stuff ones

chaining:
callbacks are called in order,
return of previous callback
received as param

batteries

included

```
 1 from twisted.internet.defer import DeferredList
 2 from twisted.internet.task import react
 3 from twisted.web.client import Agent, RedirectAgent
 4
 5 urls = (
 6     'http://twitter.com/',
 7     'http://google.com/',
 8     'http://yahoo.com/',
 9     'http://facebook.com/'
10 )
11
12 def print_response(resp, url):
13     print url, 'OK' if resp.code == 200 else 'Bad'
14
15 def main(reactor):
16     dl = []
17     agent = RedirectAgent(Agent(reactor))
18     for url in urls:
19         d = agent.request('GET', url)
20         d.addCallback(print_response, url)
21         dl.append(d)
22     return DeferredList(dl)
23
24 react(main)
```

- batteries included
- explicit async model

- unpythonic and “viral"
- explicit async model!

Gevent

cooperative preemptive

coroutines

create coroutine

yield as an expression

```
def echo():
    while True:
        val = (yield)
        print 'I resurrected to say {0}'.format(val)

cr = echo()
cr.next()
cr.send('hi')
cr.send('bye')
```

start coroutine

resume coroutine and
pass in a value

http://dabeaz.com/generators-uk

http://dabeaz.com/coroutines

http://dabeaz.com/finalgenerator

"Coroutines Trilogy"

David Beazley

pseudo-threads

greenlets

monkey-patches

standard lib

libev

event loop

```python
import gevent.monkey
gevent.monkey.patch_all()

import gevent
import urllib2

urls = (
    'http://twitter.com/',
    'http://google.com/',
    'http://yahoo.com/',
    'http://facebook.com/'
)

def fetch(url):
    resp = urllib2.urlopen(url)
    print url, 'OK' if resp.code == 200 else 'Bad'

gevent.joinall([gevent.spawn(fetch, url) for url in urls])
```

- sync-like API
- performance

- monkey-patching
- compatibility

asyncio

(a.k.a. tulip)

stdlib

official module

event loop

callbacks

futures

… or

coroutines

(yield from)

3rd party frameworks coexistence

```python
import asyncio
import aiohttp

urls = (
    'http://twitter.com/',
    'http://google.com/',
    'http://yahoo.com/',
    'http://facebook.com/'
)

@asyncio.coroutine
def fetch(url):
    resp = yield from aiohttp.request('GET', url)
    print(url, 'OK' if resp.status == 200 else 'Bad')

def main():
    coros = [fetch(url) for url in urls]
    yield from asyncio.wait(coros)

loop = asyncio.get_event_loop()
loop.run_until_complete(main())
```

- default, standard lib
- strategy agnostic

- missing ecosystem
- Python 3 only

Questions?

Thanks!

Aitor Guevara • @aitorciki