Twisted

an introductory training

Aim of this training

- Ascend from clueless to aspiring newbie
- Consider Twisted for new applications
- Be able to ask the right questions
- Be able to dive in the source

About me

http://orestis.gr - @orestis

About you

Raise your hand if...

Raise your hand if...

- You have done web development
- You have done GUI development
- You know something about networking

Some setup

- Install Twisted 11.0
- Grab the Twisted source & apidocs from the USB keys

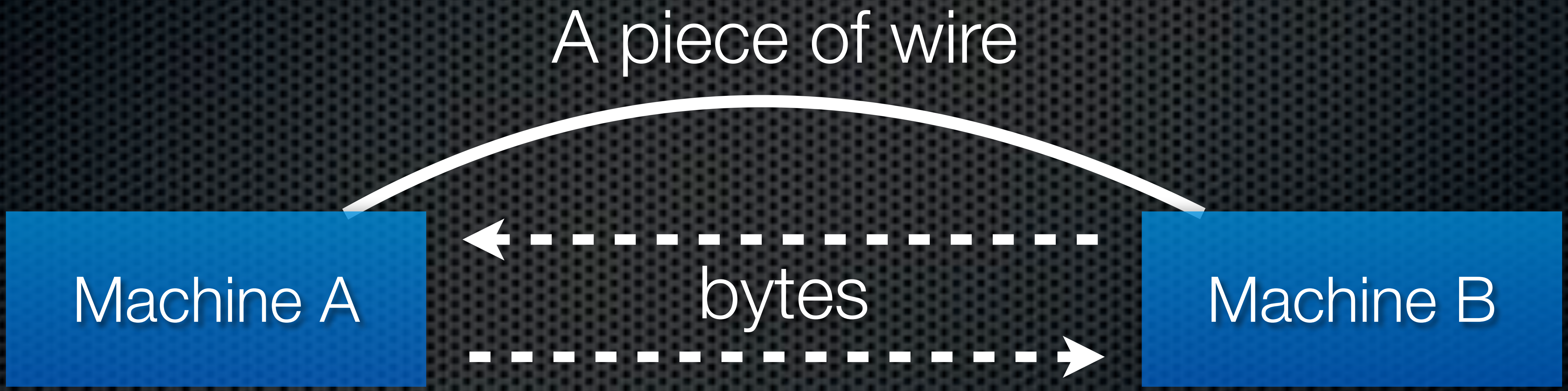
Test

```
>>> from twisted.internet import reactor
>>> from twisted import version
>>> version
Version('twisted', 11, 0, 0)
>>>
```

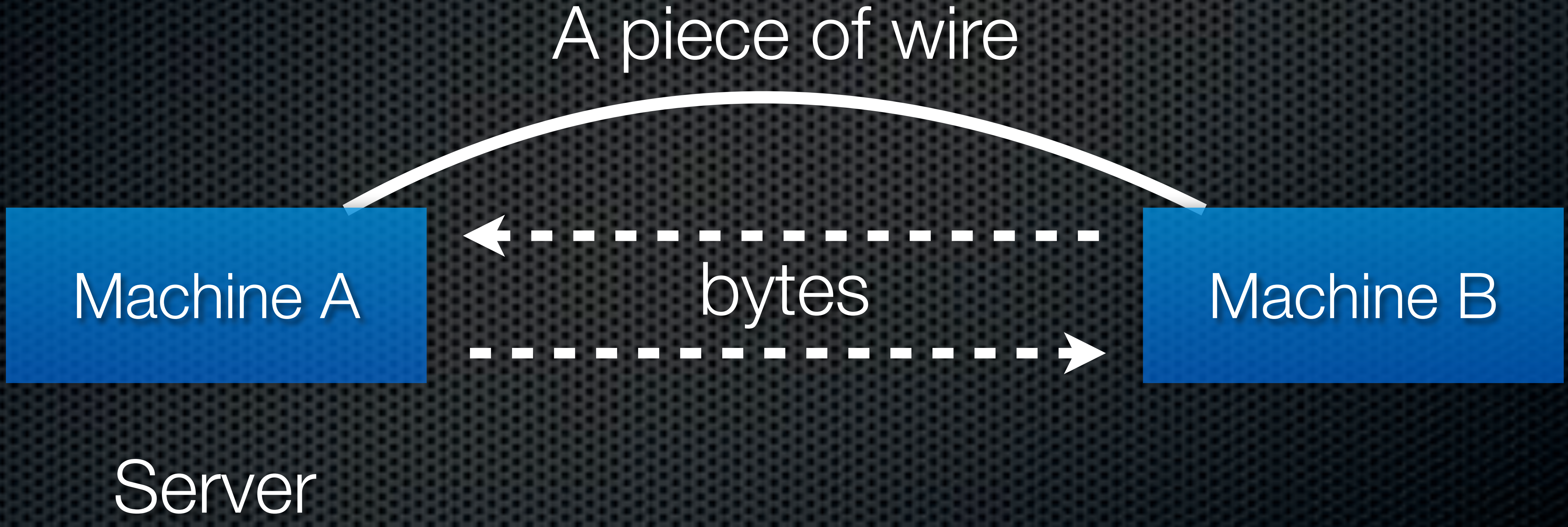
Network programming

Machine A
Machine B

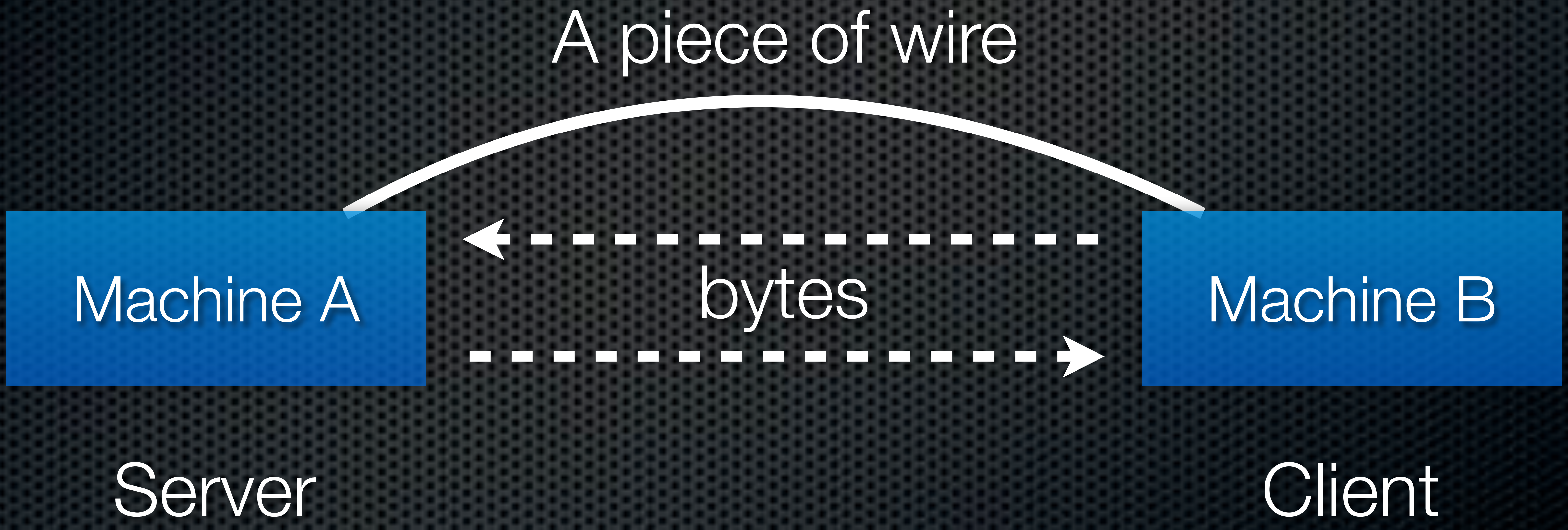
Network programming



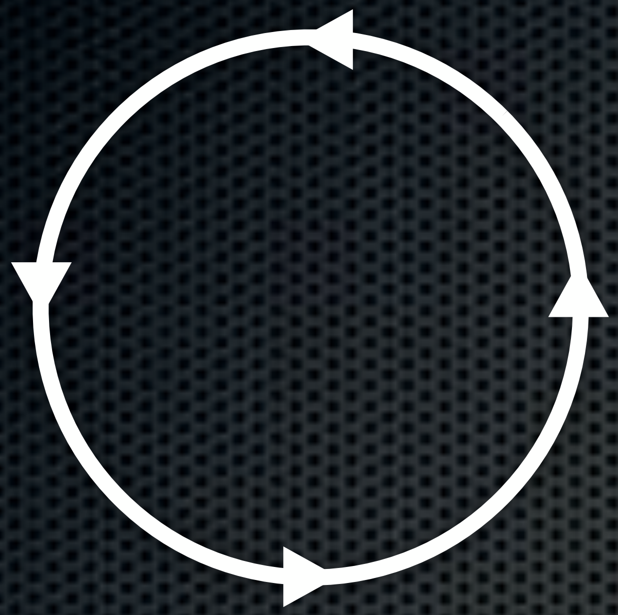
Network programming



Network programming

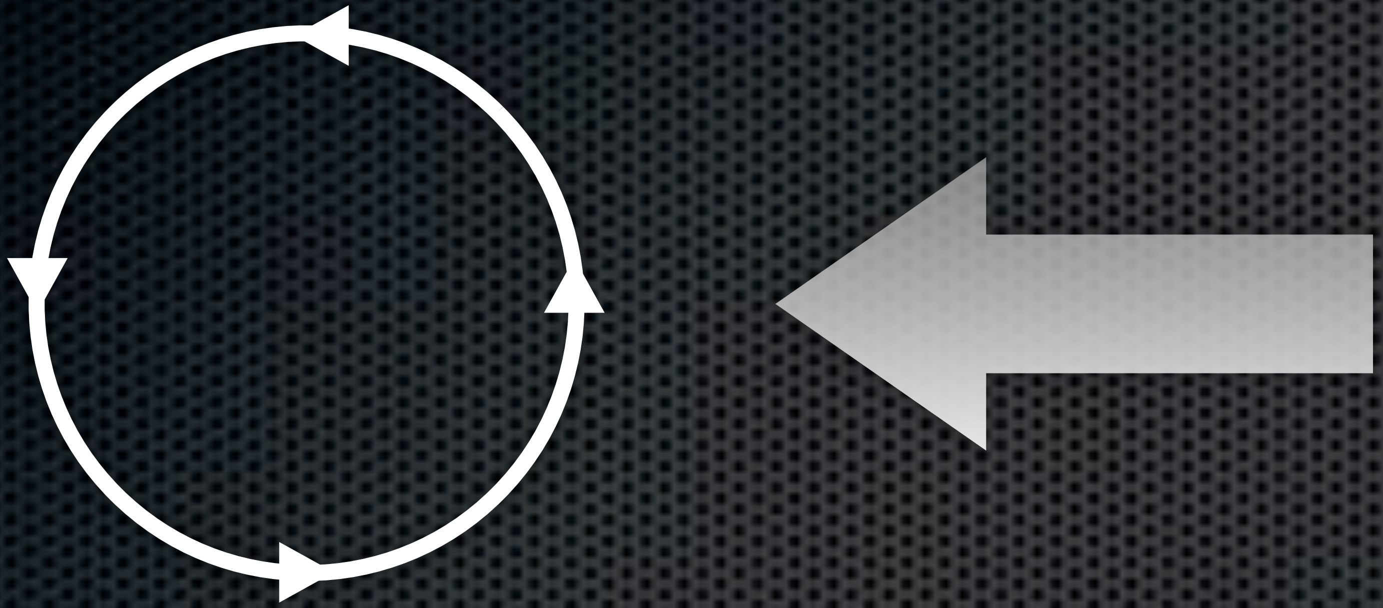


Network programming



Server listens on a port

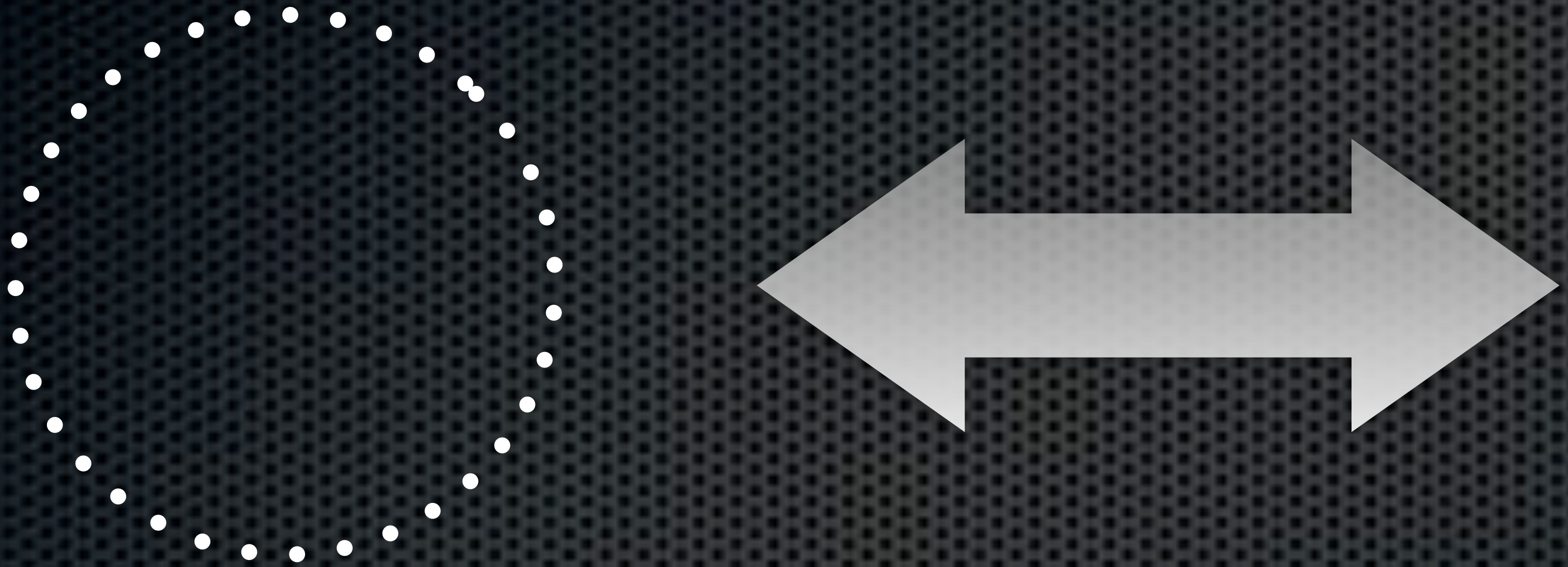
Network programming



Client connects to port

Server listens on a port

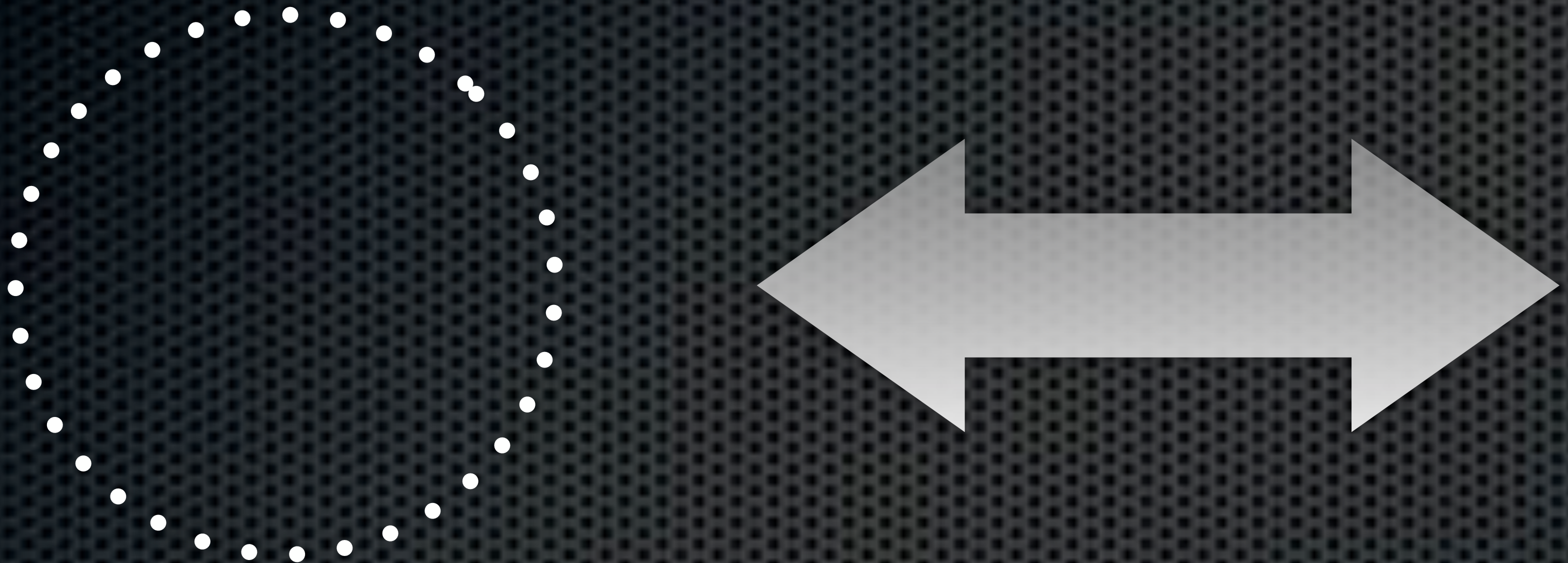
Service request



Client reads/
writes data

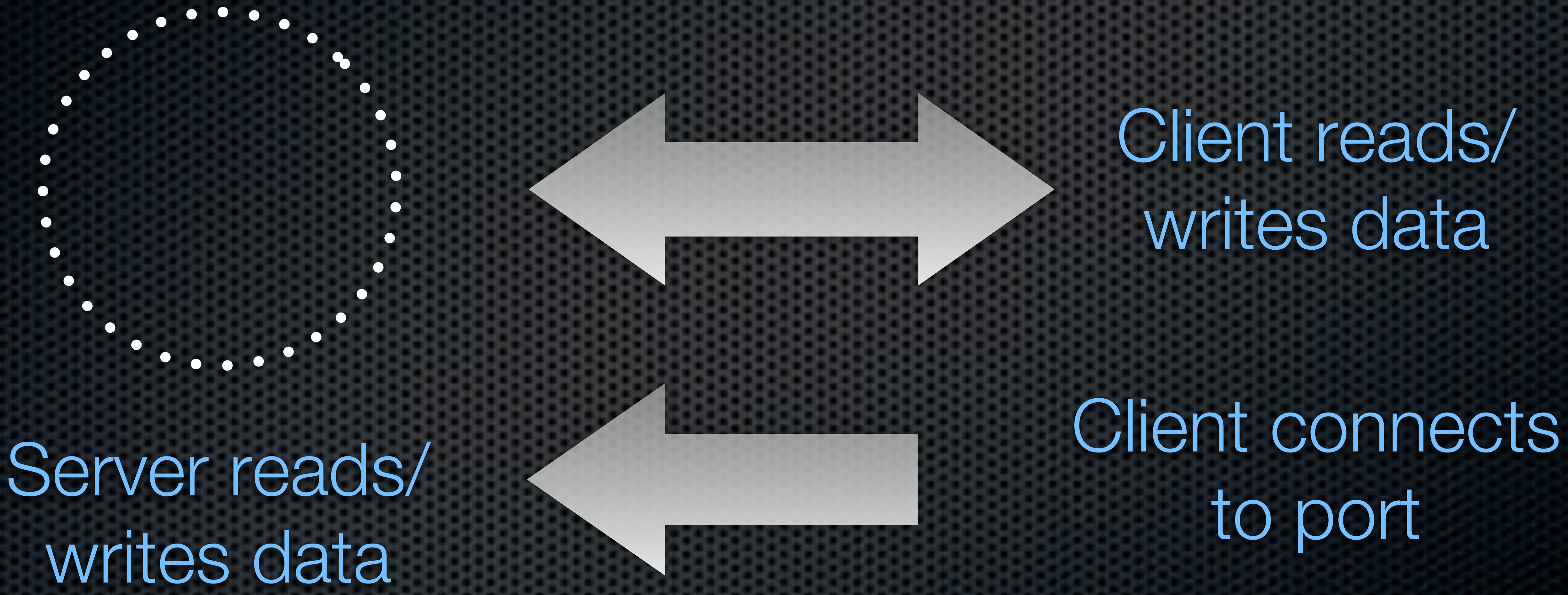
Server reads/
writes data

While servicing...

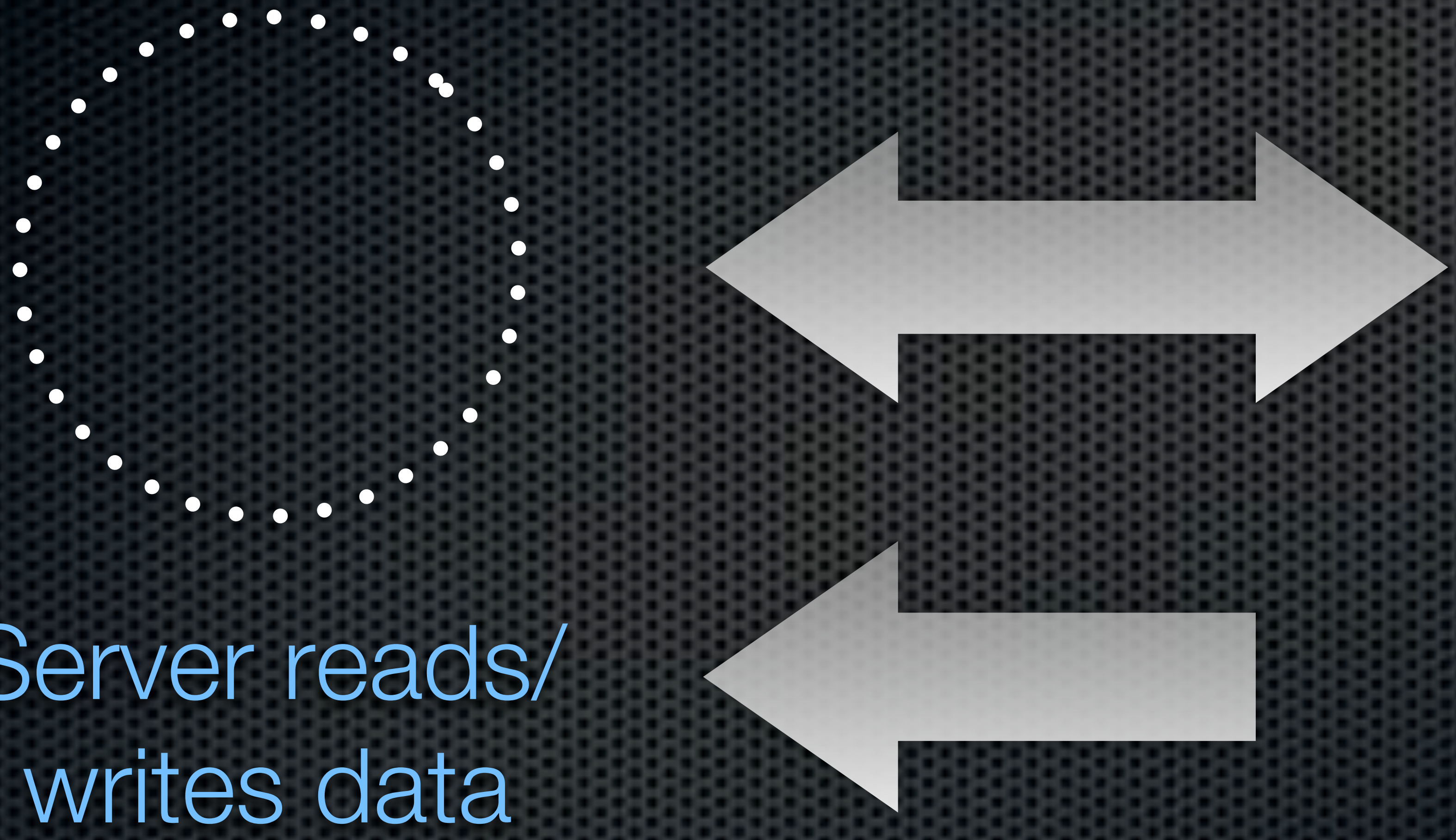


Server reads/
writes data

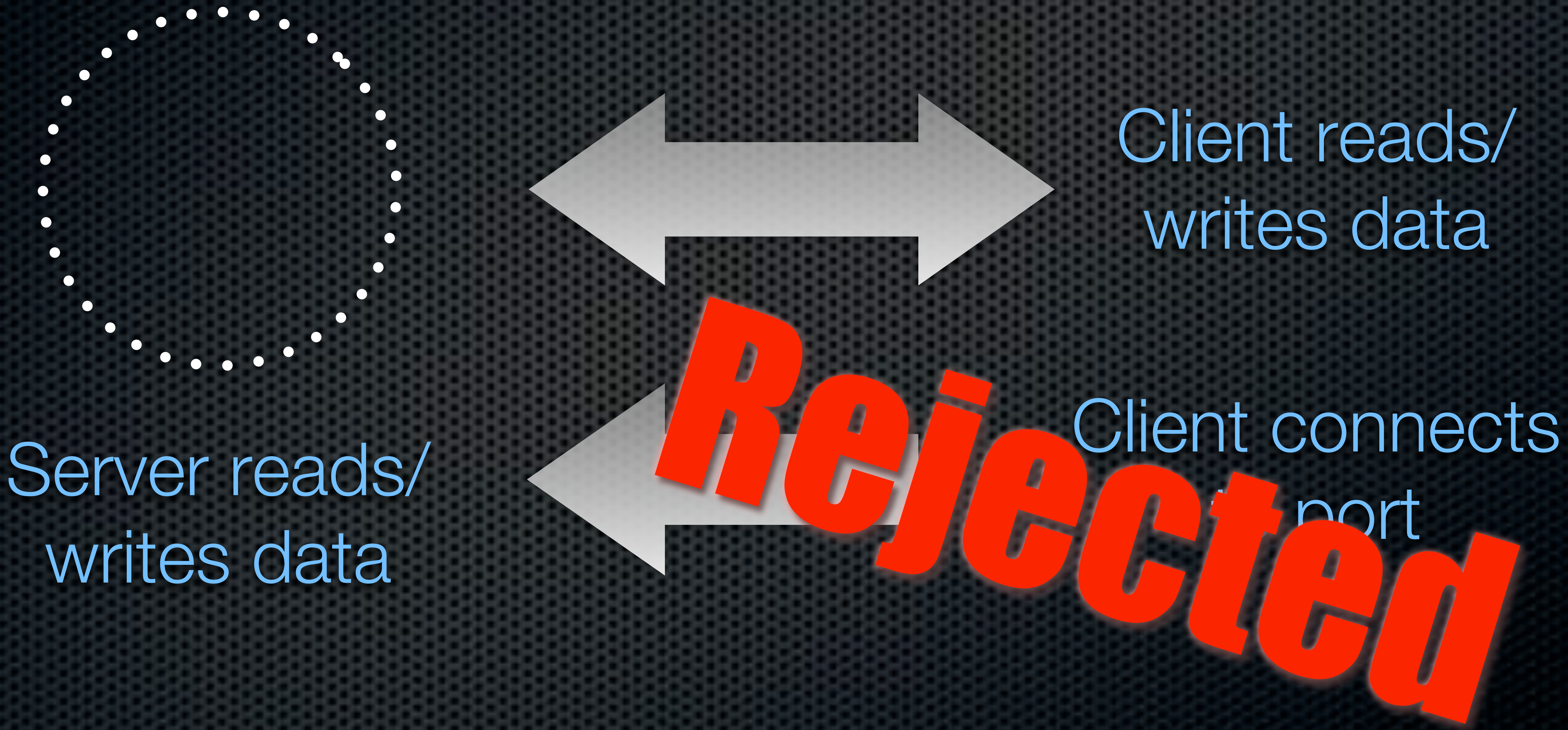
While servicing...



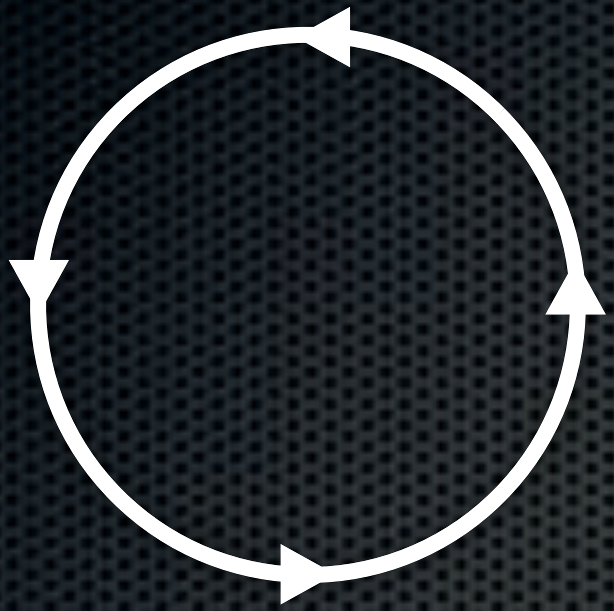
Timeout!


Client reads/
writes data
Client connects
to port
Server reads/
writes data

Timeout!

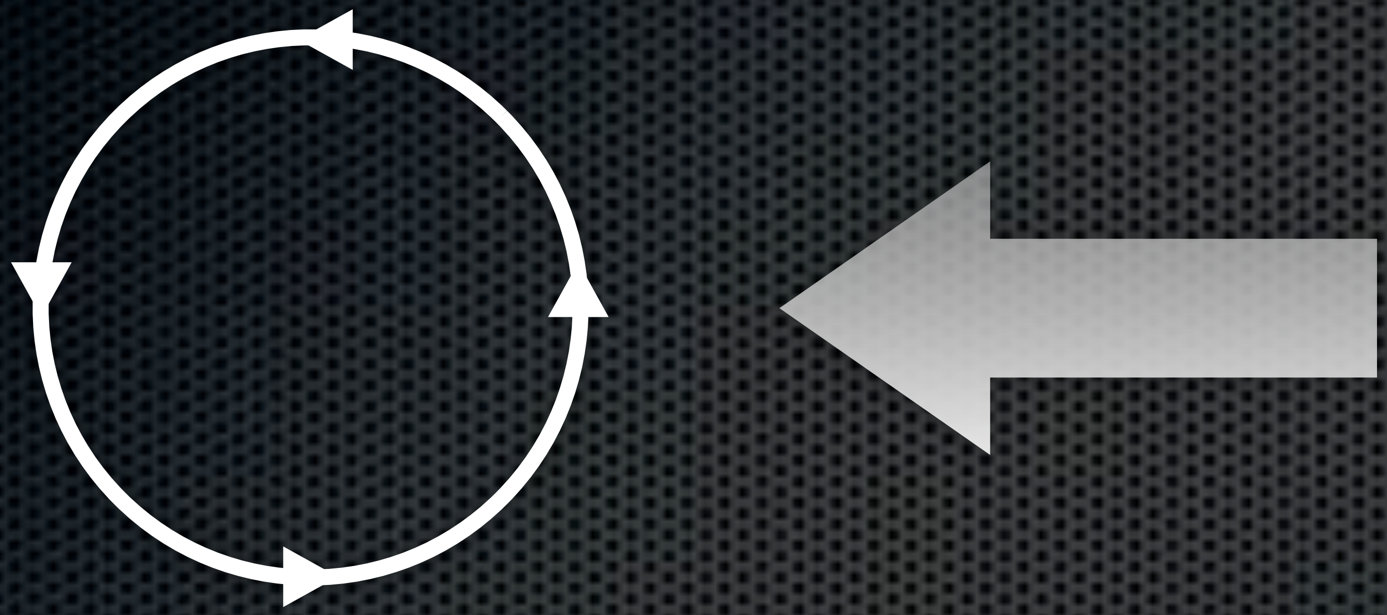

Client reads/ writes data
Client connects to port
Server reads/ writes data
Rejected

Must not block!



Server listens
on a port

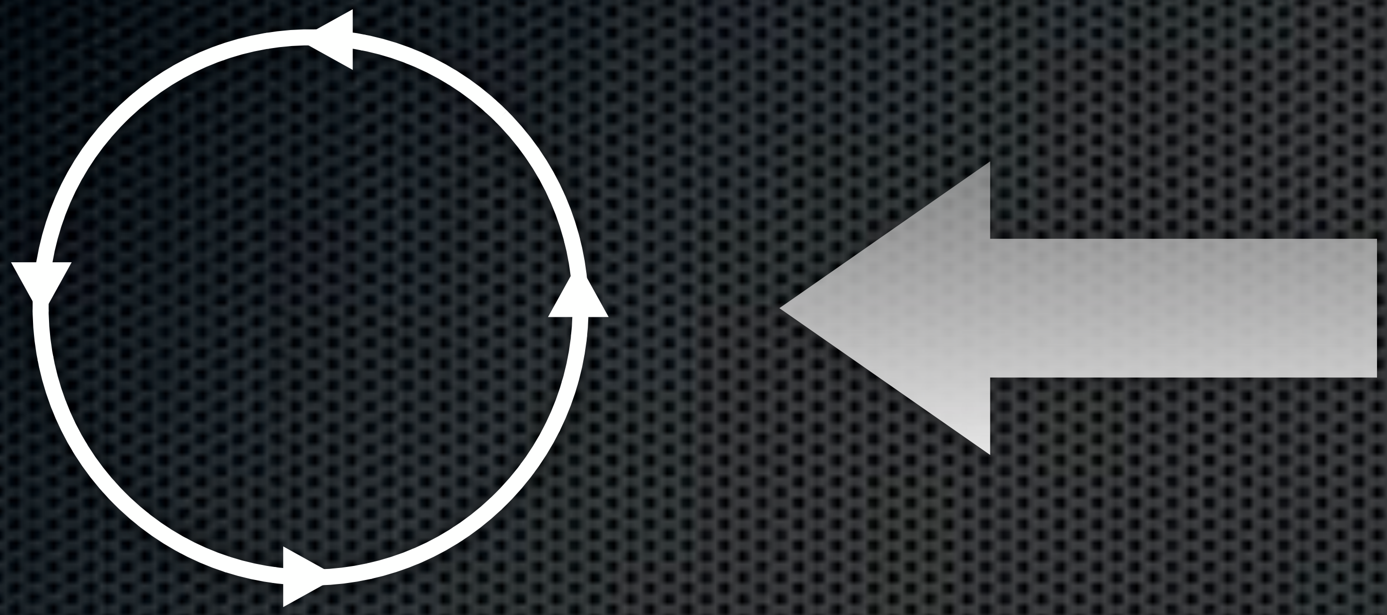
Must not block!



Client connects
to port

Server listens
on a port

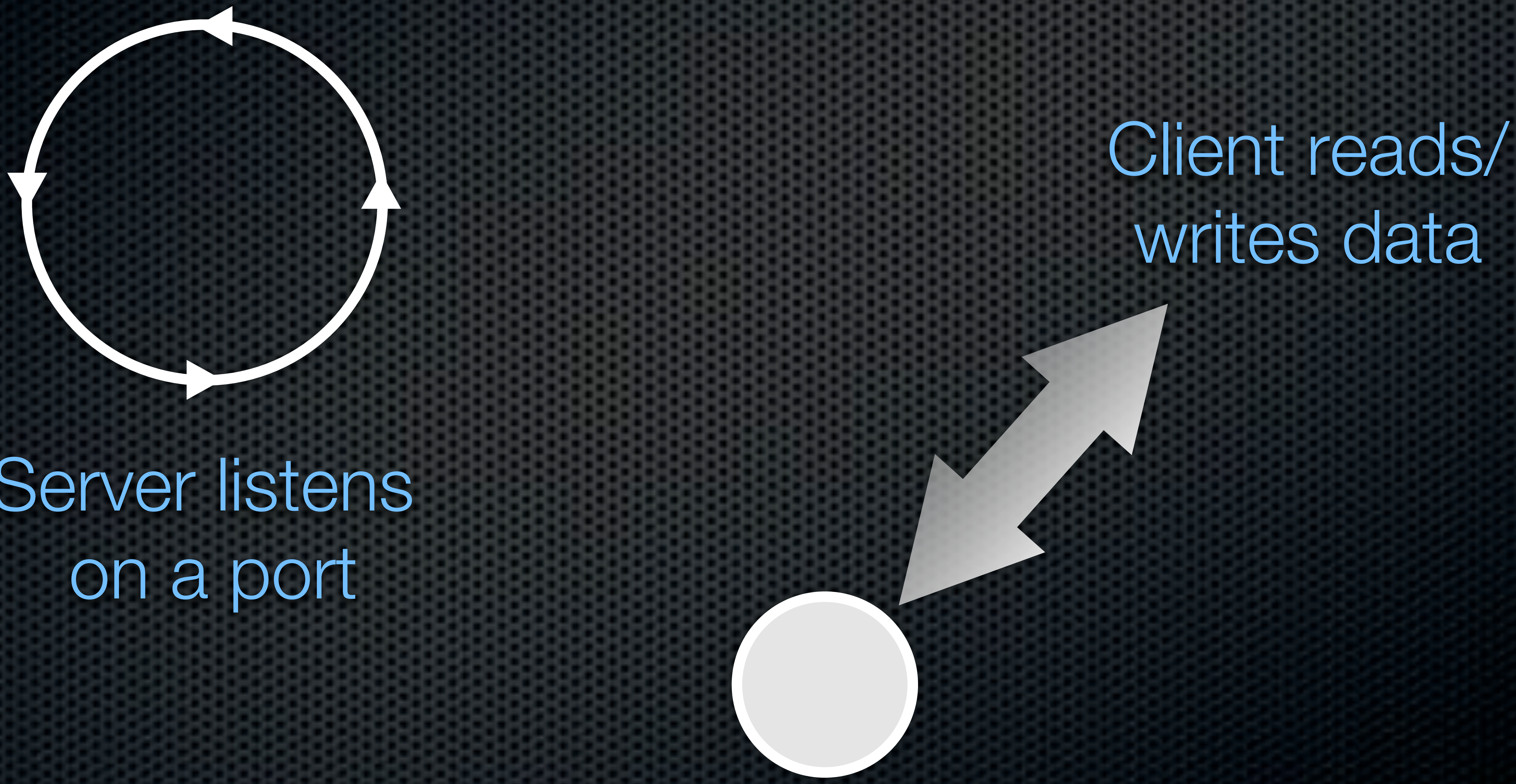
Handle request elsewhere



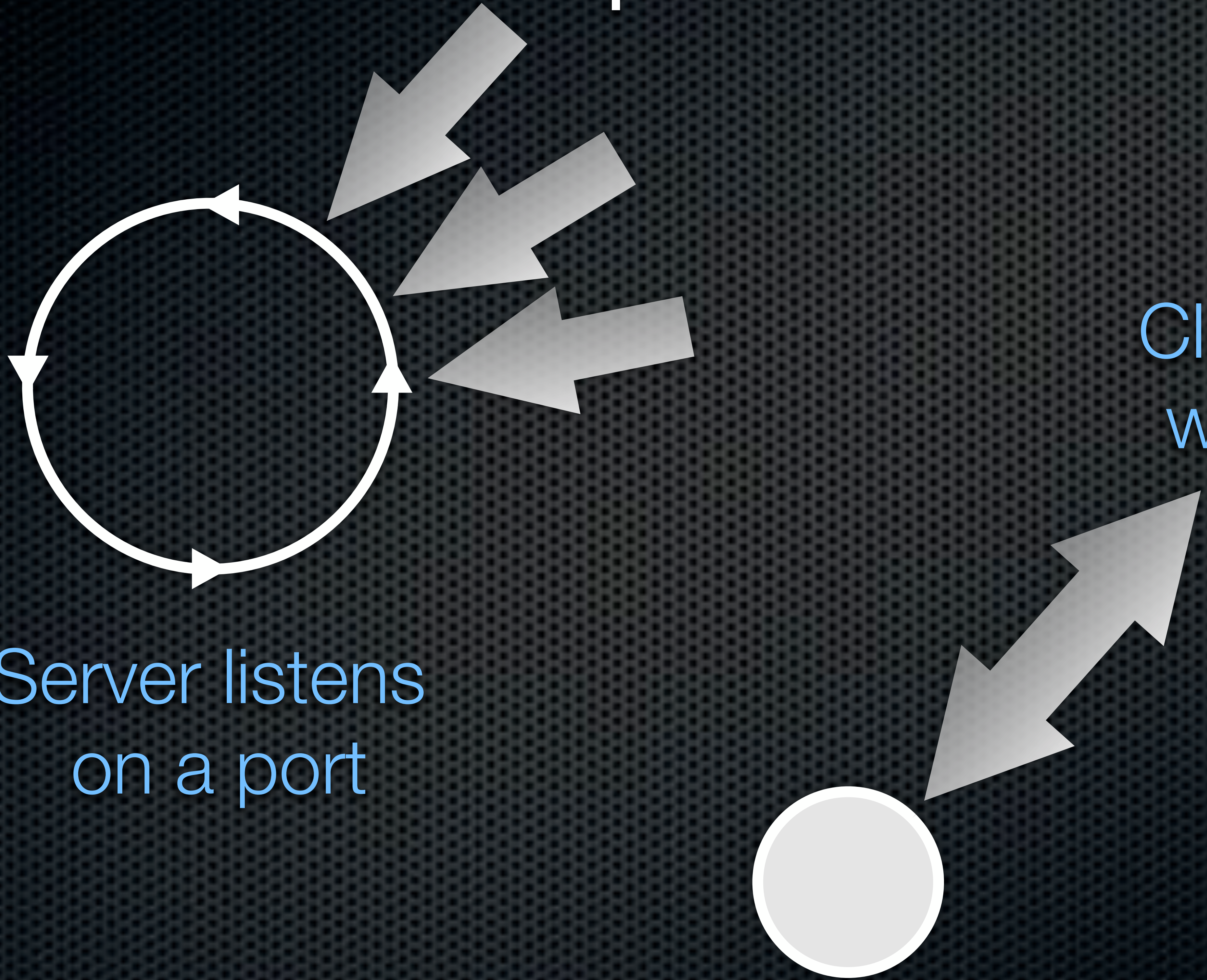
Client connects
to port

Server listens
on a port

Handle request elsewhere

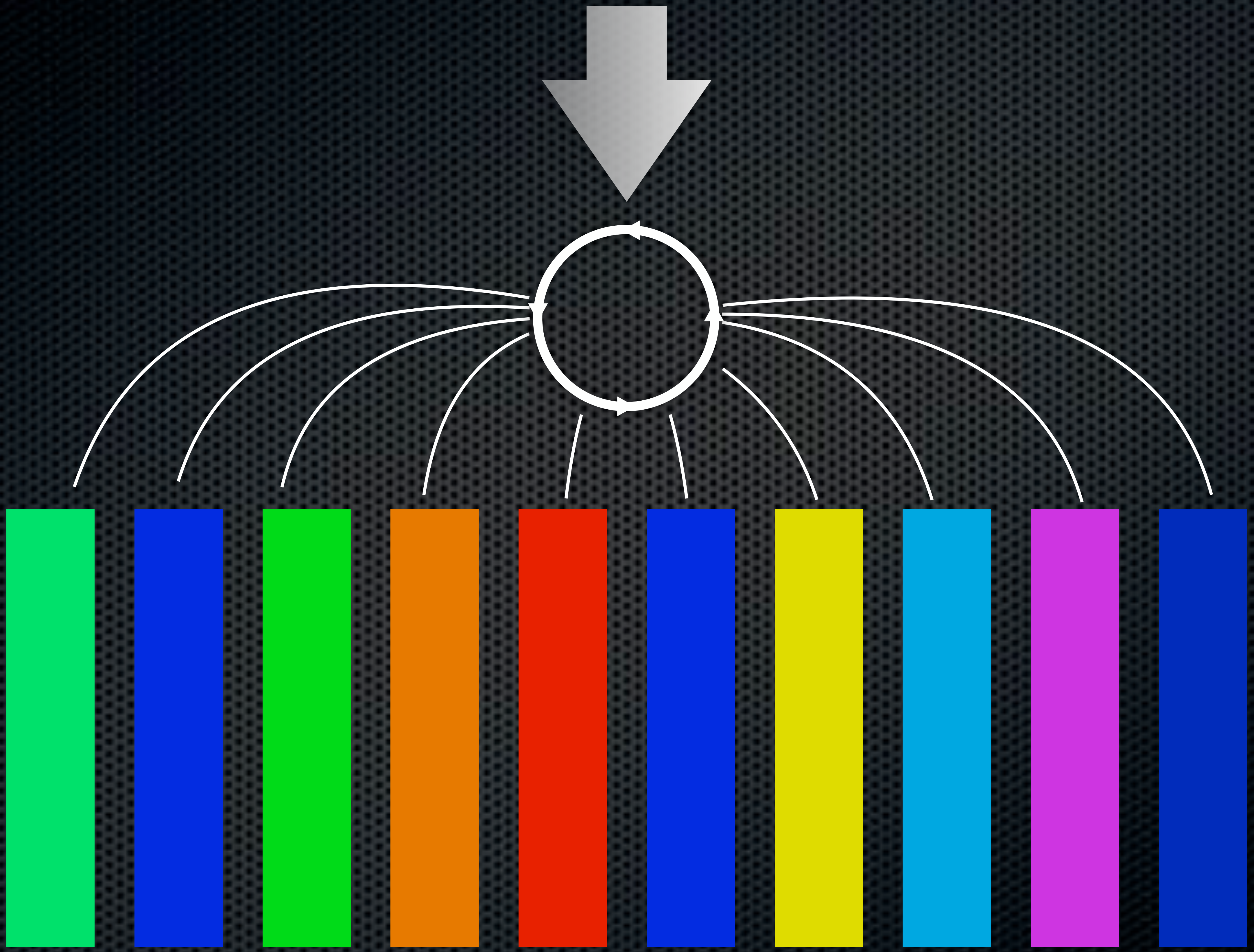


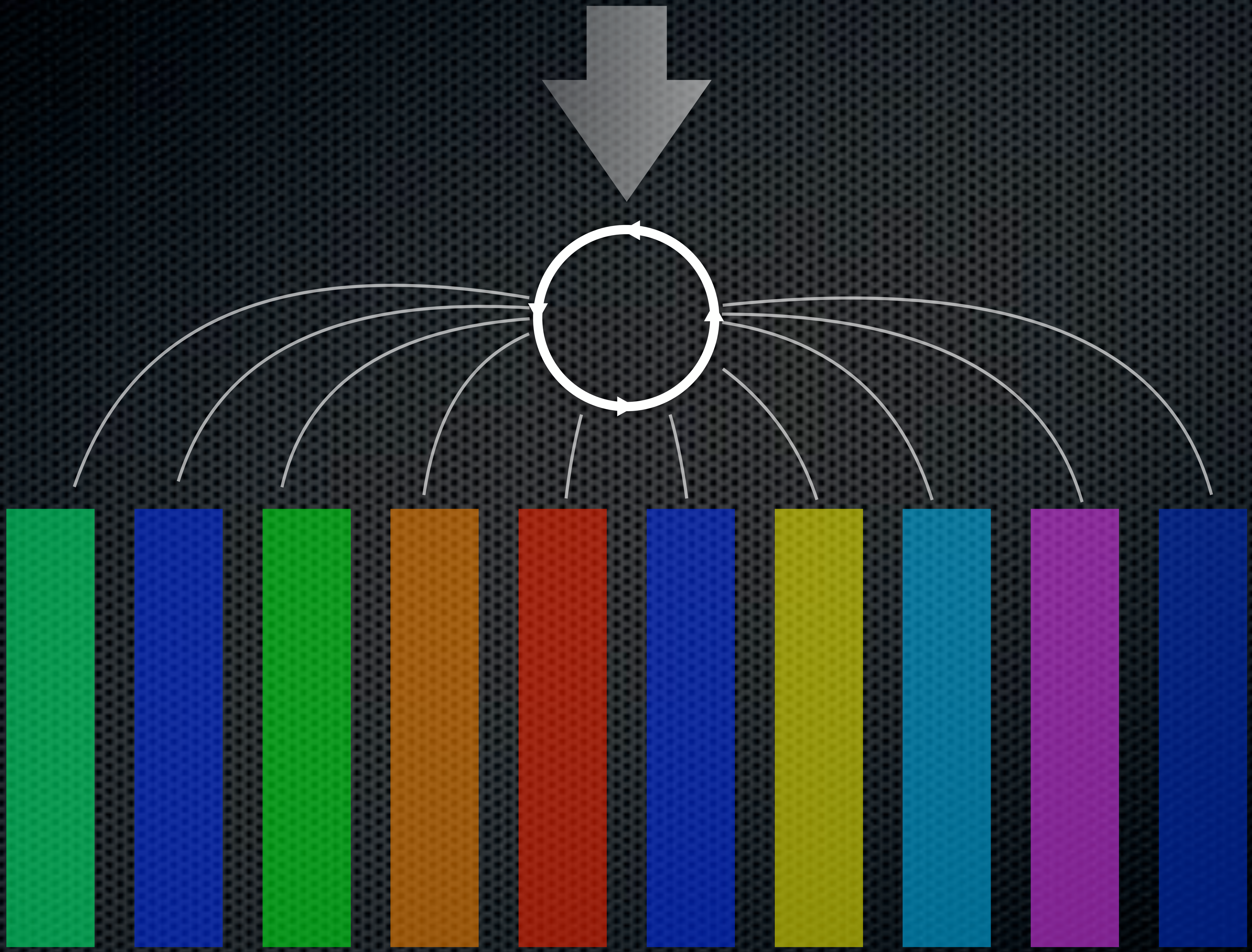
Handle request elsewhere

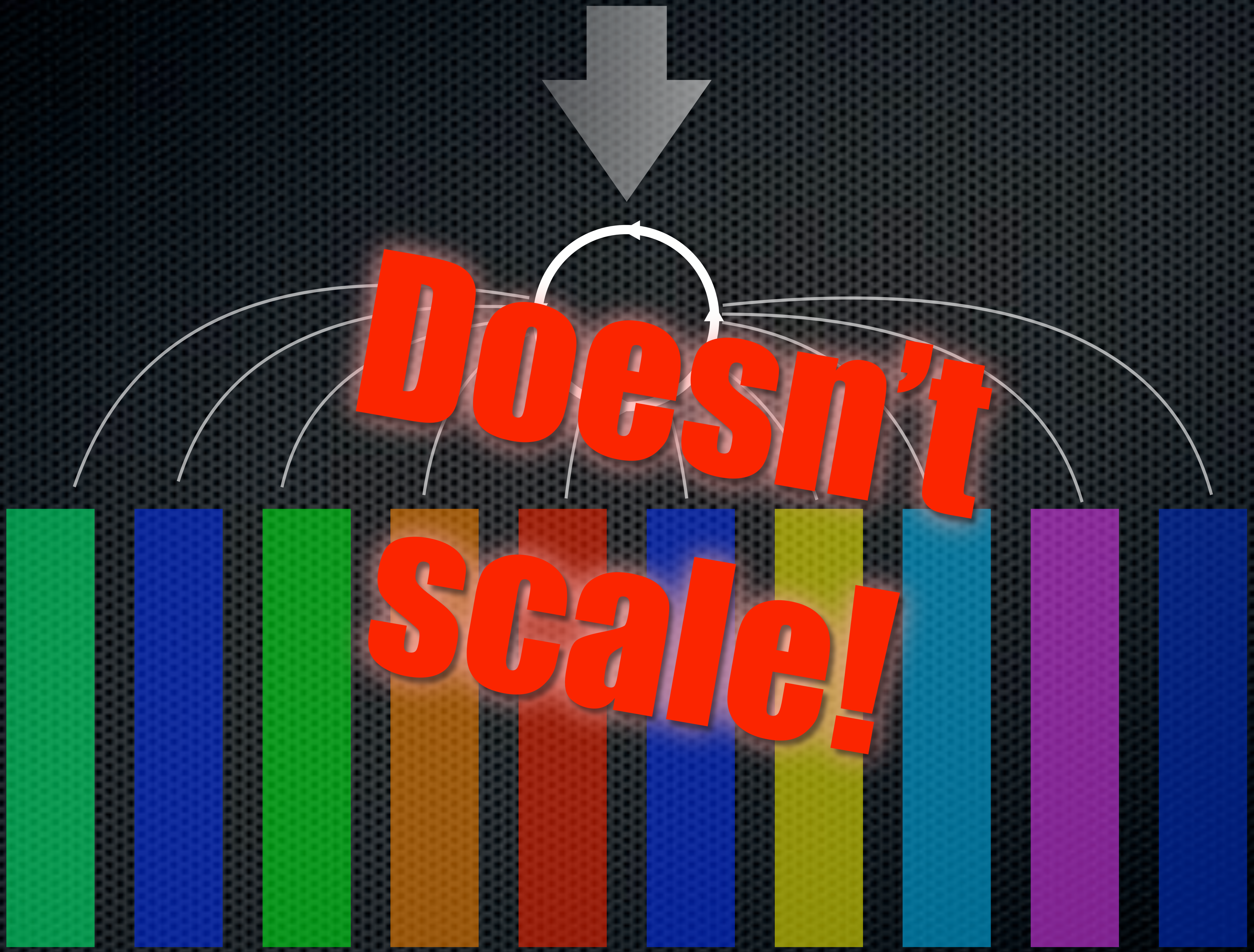


Handle request elsewhere

- Fork a new process
- Have worker threads
- Asynchronous I/O



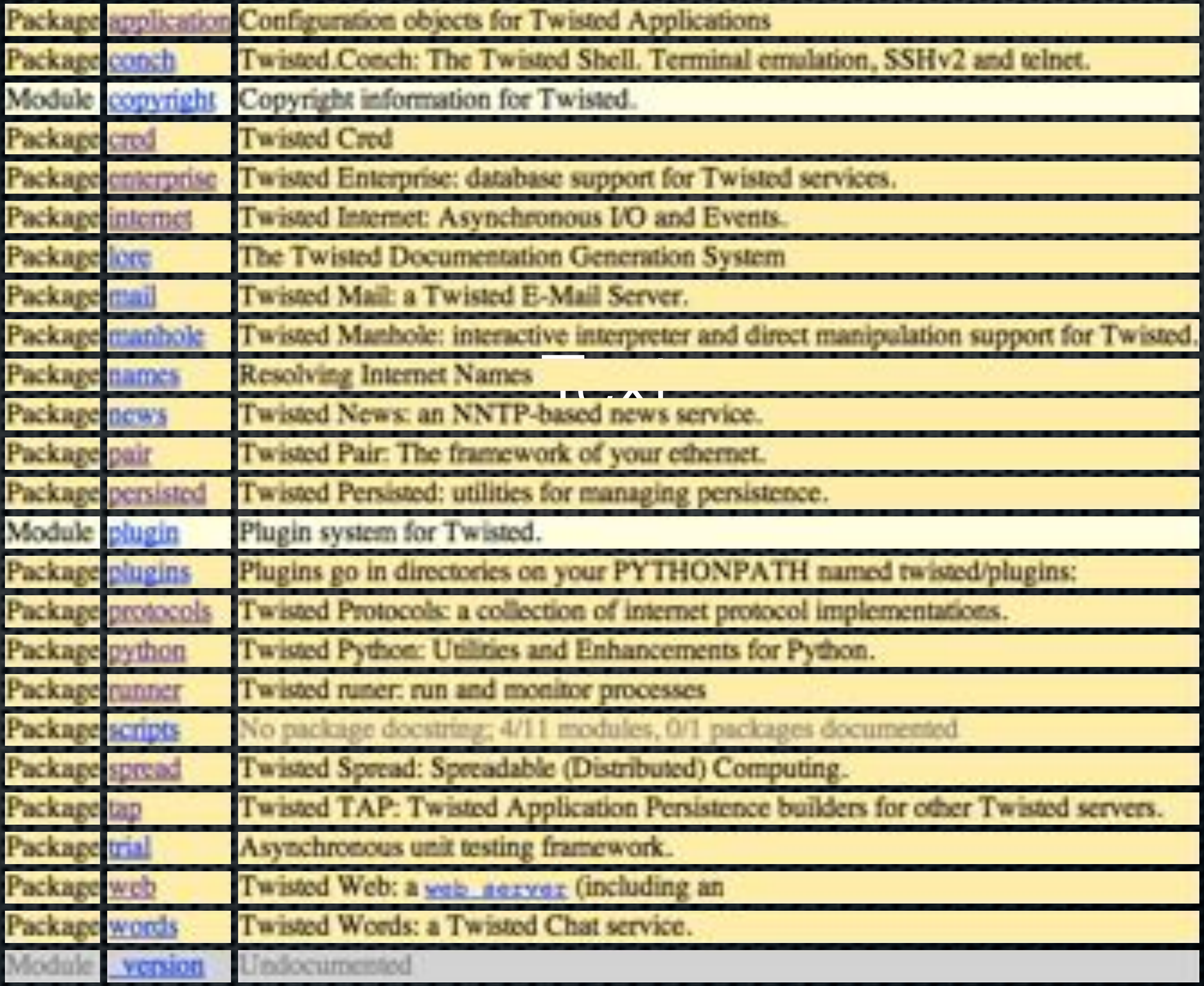



Doesn't
scale!

Twisted?

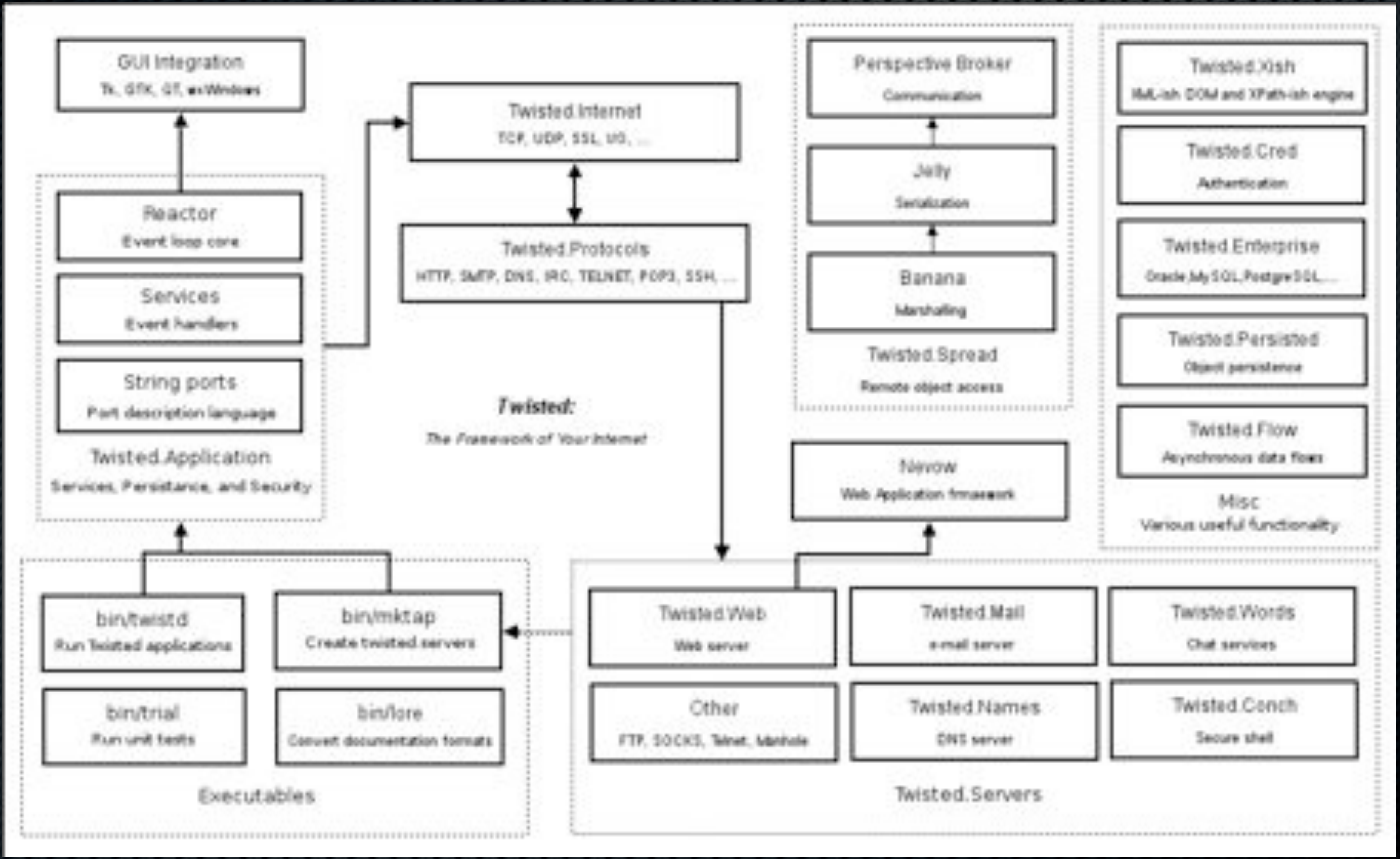
Twisted is a networking engine written in Python, supporting numerous protocols. It contains a web server, numerous chat clients, chat servers, mail servers, and more.

Twisted!



Package	application	Configuration objects for Twisted Applications
Package	conch	Twisted.Conch: The Twisted Shell. Terminal emulation, SSHv2 and telnet.
Module	copyright	Copyright information for Twisted.
Package	cred	Twisted Cred
Package	enterprise	Twisted Enterprise: database support for Twisted services.
Package	internet	Twisted Internet: Asynchronous I/O and Events.
Package	lore	The Twisted Documentation Generation System
Package	mail	Twisted Mail: a Twisted E-Mail Server.
Package	manhole	Twisted Manhole: interactive interpreter and direct manipulation support for Twisted.
Package	names	Resolving Internet Names
Package	news	Twisted News: an NNTP-based news service.
Package	pair	Twisted Pair: The framework of your ethernet.
Package	persisted	Twisted Persisted: utilities for managing persistence.
Module	plugin	Plugin system for Twisted.
Package	plugins	Plugins go in directories on your PYTHONPATH named twisted/plugins:
Package	protocols	Twisted Protocols: a collection of internet protocol implementations.
Package	python	Twisted Python: Utilities and Enhancements for Python.
Package	runner	Twisted runer: run and monitor processes
Package	scripts	No package docstring; 4/11 modules, 0/1 packages documented
Package	spread	Twisted Spread: Spreadable (Distributed) Computing.
Package	tap	Twisted TAP: Twisted Application Persistence builders for other Twisted servers.
Package	trial	Asynchronous unit testing framework.
Package	web	Twisted Web: a web server (including an
Package	words	Twisted Words: a Twisted Chat service.
Module	_version	Undocumented

Twisted!



GUI Integration
Tk, GTK, QT, wxWindows
Twisted.Internet
TCP, UDP, SSL, UG, ...
Reactor
Event loop core
Services
Event handlers
String ports
Port description language
Twisted.Application
Services, Persistance, and Security
Twisted.Protocols
HTTP, SMTP, DNS, IRC, TELNET, POP3, SSH, ...
Twisted:
The Framework of Your Internet
Perspective Broker
Communication
Jelly
Serialization
Banana
Marshalling
Twisted.Spread
Remote object access
Nevow
Web Application framework
Twisted.Xish
XML-ish DOM and XPath-ish engine
Twisted.Cred
Authentication
Twisted.Enterprise
Oracle,MySQL,PostgreSQL,...
Twisted.Persisted
Object persistence
Twisted.Flow
Asynchronous data flows
Misc
Various useful functionality
bin/twistd
Run Twisted applications
bin/mktap
Create twisted servers
bin/trial
Run unit tests
bin/lore
Convert documentation formats
Executables
Twisted.Web
Web server
Twisted.Mail
e-mail server
Twisted.Words
Chat services
Other
FTP, SOCKS, Telnet, Manhole
Twisted.Names
DNS server
Twisted.Conch
Secure shell
Twisted.Servers

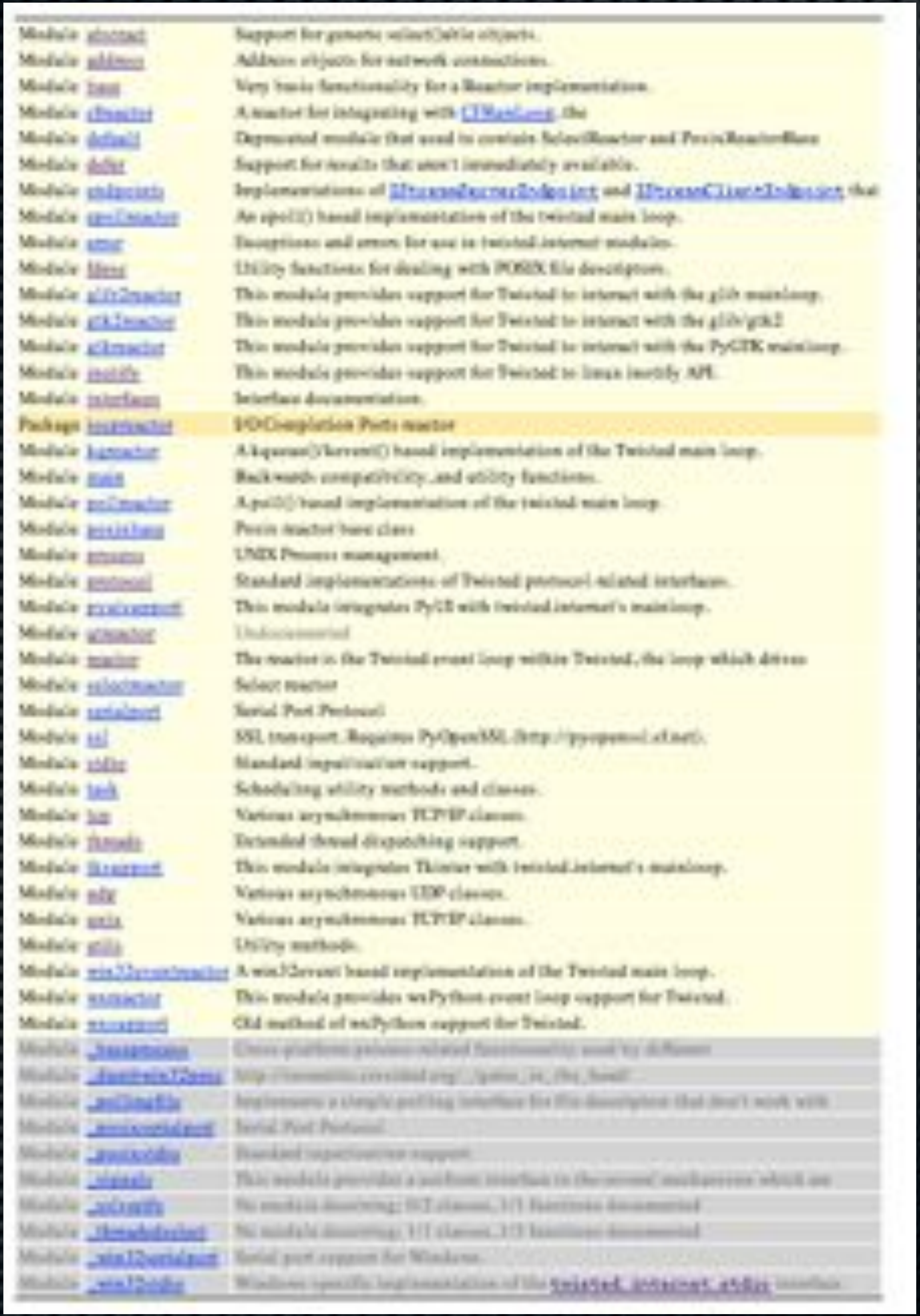
Twisted

Twisted is a **networking engine** written in Python, supporting numerous protocols. It contains a web server, numerous chat clients, chat servers, mail servers, and more.

twisted.internet

Asynchronous I/O and Events.

twisted.internet !!!



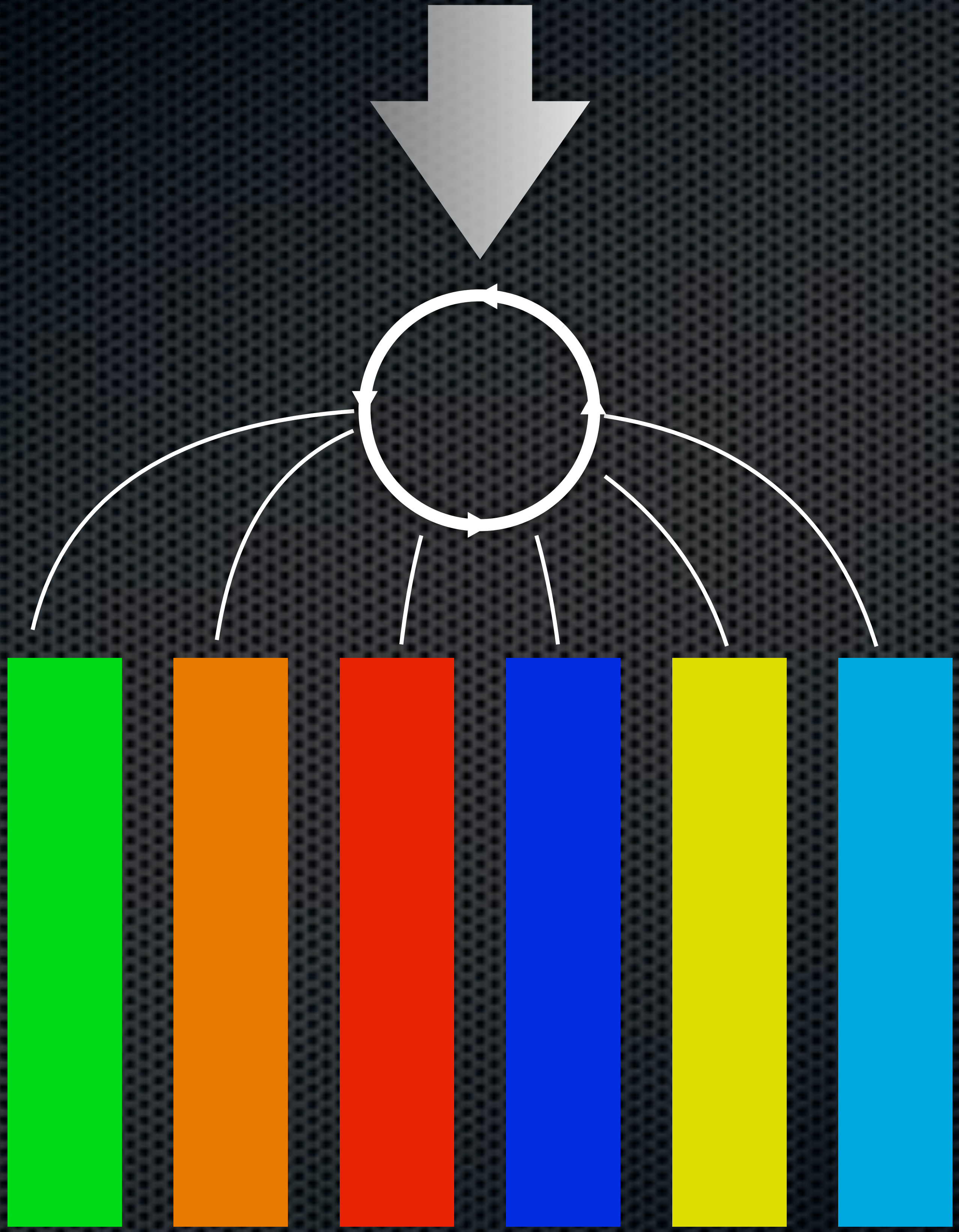
Module	abstract	Support for generic select()able objects.
Module	address	Address objects for network connections.
Module	base	Very basic functionality for a Reactor implementation.
Module	cfreactor	A reactor for integrating with CFRunLoop, the
Module	default	Deprecated module that used to contain SelectReactor and PosixReactorBase
Module	defer	Support for results that aren't immediately available.
Module	endpoints	Implementations of IStreamServerEndpoint and IStreamClientEndpoint that
Module	epollreactor	An epoll() based implementation of the twisted main loop.
Module	error	Exceptions and errors for use in twisted.internet modules.
Module	fdesc	Utility functions for dealing with POSIX file descriptors.
Module	glib2reactor	This module provides support for Twisted to interact with the glib mainloop.
Module	gtk2reactor	This module provides support for Twisted to interact with the glib/gtk2
Module	gtkreactor	This module provides support for Twisted to interact with the PyGTK mainloop.
Module	inotify	This module provides support for Twisted to linux inotify API.
Module	interfaces	Interface documentation.
Package	iocpreactor	I/O Completion Ports reactor
Module	kqreactor	A kqueue()/kevent() based implementation of the Twisted main loop.
Module	main	Backwards compatibility, and utility functions.
Module	pollreactor	A poll() based implementation of the twisted main loop.
Module	posixbase	Posix reactor base class
Module	process	UNIX Process management.
Module	protocol	Standard implementations of Twisted protocol-related interfaces.
Module	pyuisupport	This module integrates PyUI with twisted.internet's mainloop.
Module	qtreactor	Undocumented
Module	reactor	The reactor is the Twisted event loop within Twisted, the loop which drives
Module	selectreactor	Select reactor
Module	serialport	Serial Port Protocol
Module	ssl	SSL transport. Requires PyOpenSSL (http://pyopenssl.sf.net).
Module	stdio	Standard input/out/err support.
Module	task	Scheduling utility methods and classes.
Module	tcp	Various asynchronous TCP/IP classes.
Module	threads	Extended thread dispatching support.
Module	tksupport	This module integrates Tkinter with twisted.internet's mainloop.
Module	udp	Various asynchronous UDP classes.
Module	unix	Various asynchronous TCP/IP classes.
Module	utils	Utility methods.
Module	win32eventreactor	A win32event based implementation of the Twisted main loop.
Module	wxreactor	This module provides wxPython event loop support for Twisted.
Module	wxsupport	Old method of wxPython support for Twisted.
Module	_baseprocess	Cross-platform process-related functionality used by different
Module	_dumbwin32proc	http://isometric.sixsided.org/_/gates_in_the_head/
Module	_pollingfile	Implements a simple polling interface for file descriptors that don't work with
Module	_posixserialport	Serial Port Protocol
Module	_posixstdio	Standard input/out/err support.
Module	_signals	This module provides a uniform interface to the several mechanisms which are
Module	_sslverify	No module docstring; 0/2 classes, 1/1 functions documented
Module	_threadedselect	No module docstring; 1/1 classes, 1/3 functions documented
Module	_win32serialport	Serial port support for Windows.
Module	_win32stdio	Windows-specific implementation of the twisted.internet.stdio interface

twisted.internet !!!

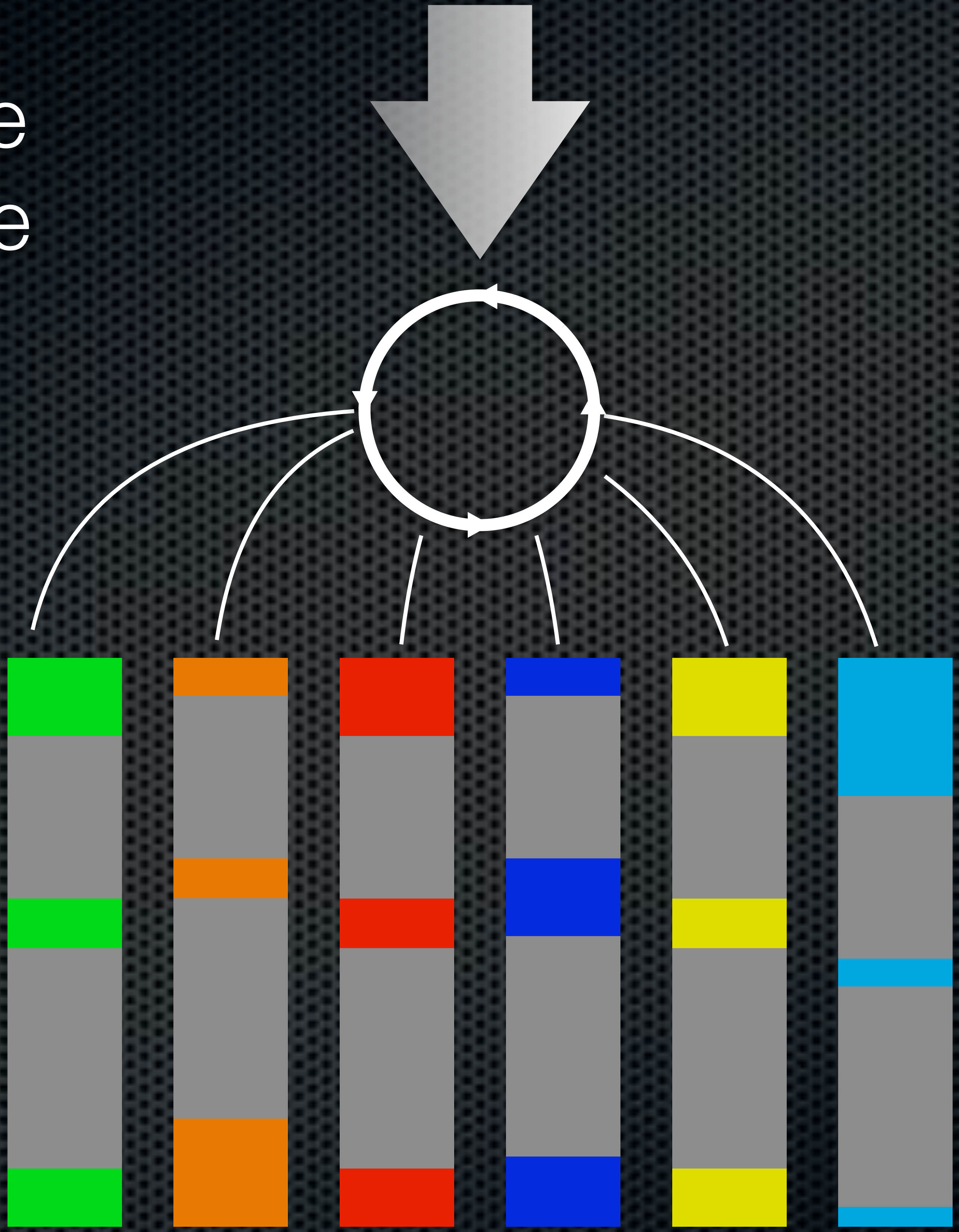
- defer
- endpoints
- error
- protocol
- reactor
- task

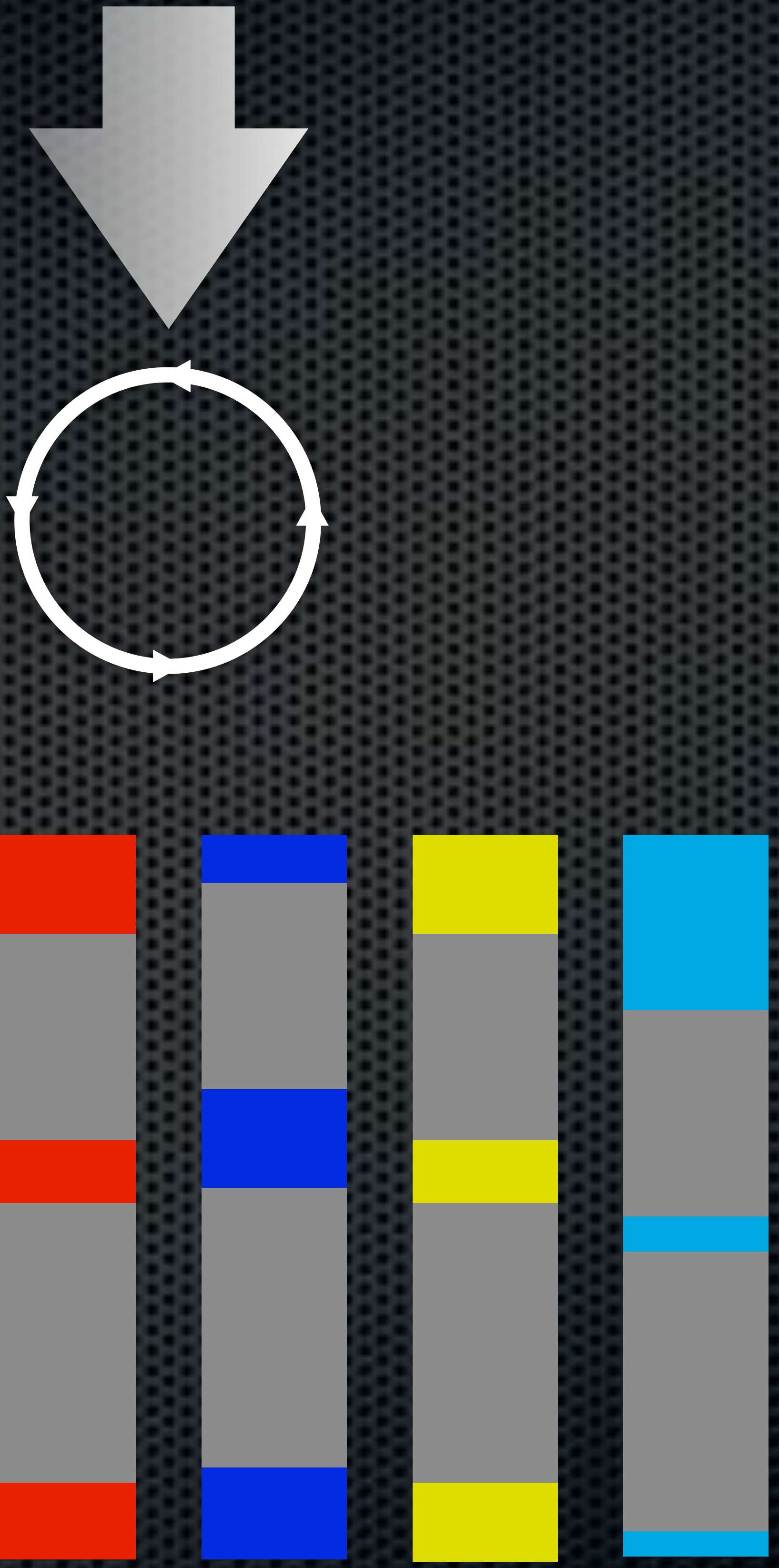
twisted.internet reactor

"the loop which drives applications using Twisted"

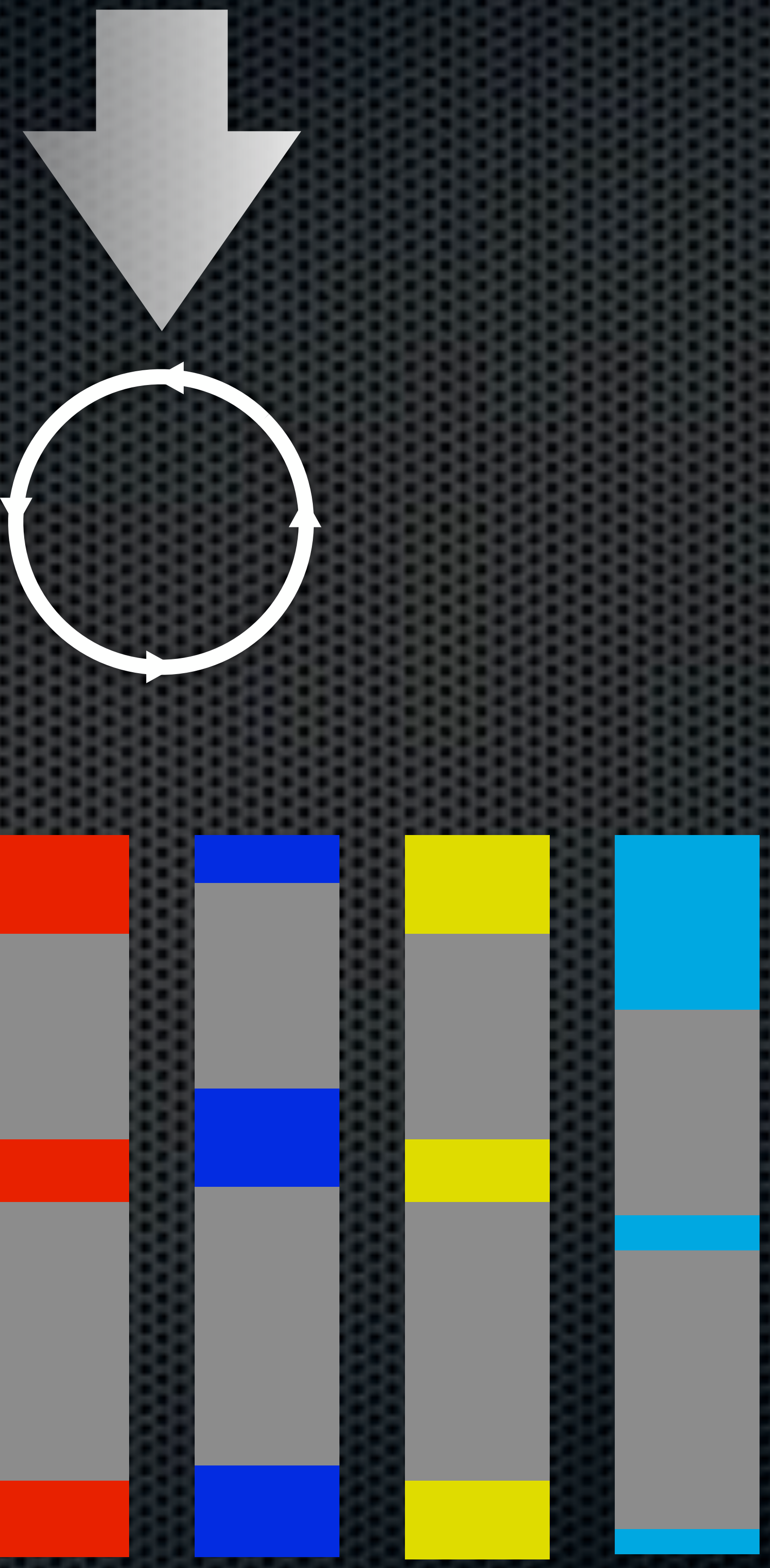


Most of the time we are waiting

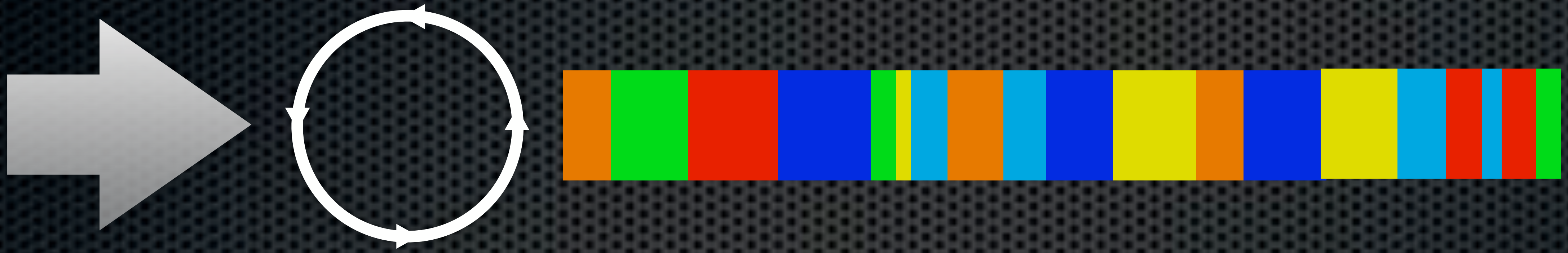




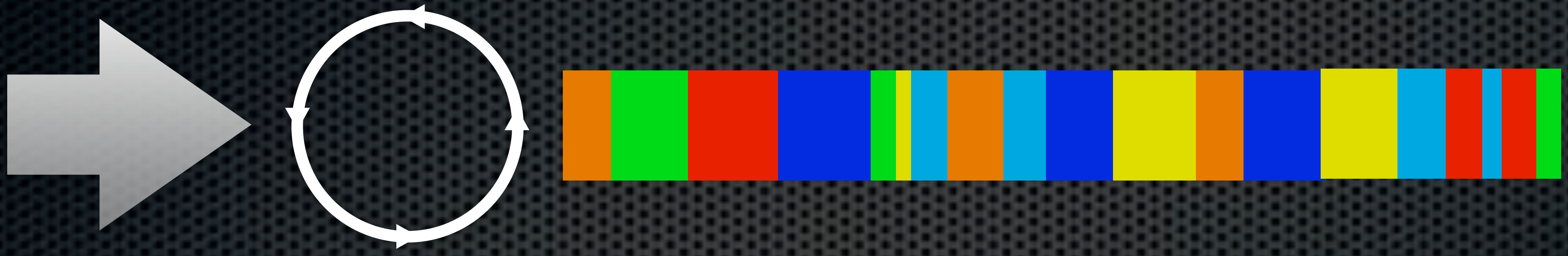
What if we could...



Eliminate Blocking?

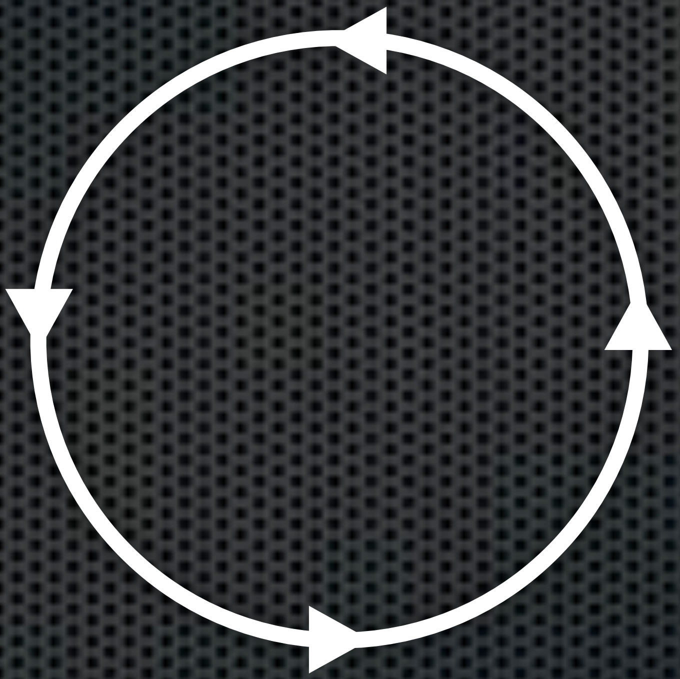


Eliminate Blocking?



Reactor loop

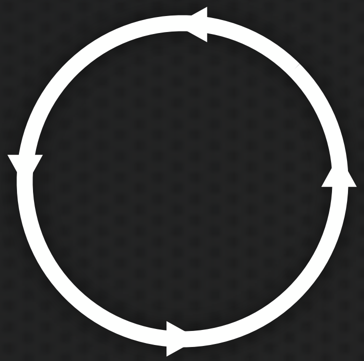
Event happens



Callback is called

Twisted reactor loop

```python
from twisted.internet import reactor
```



```python
reactor.run()
```

Run this!

Listen on a port, then do something

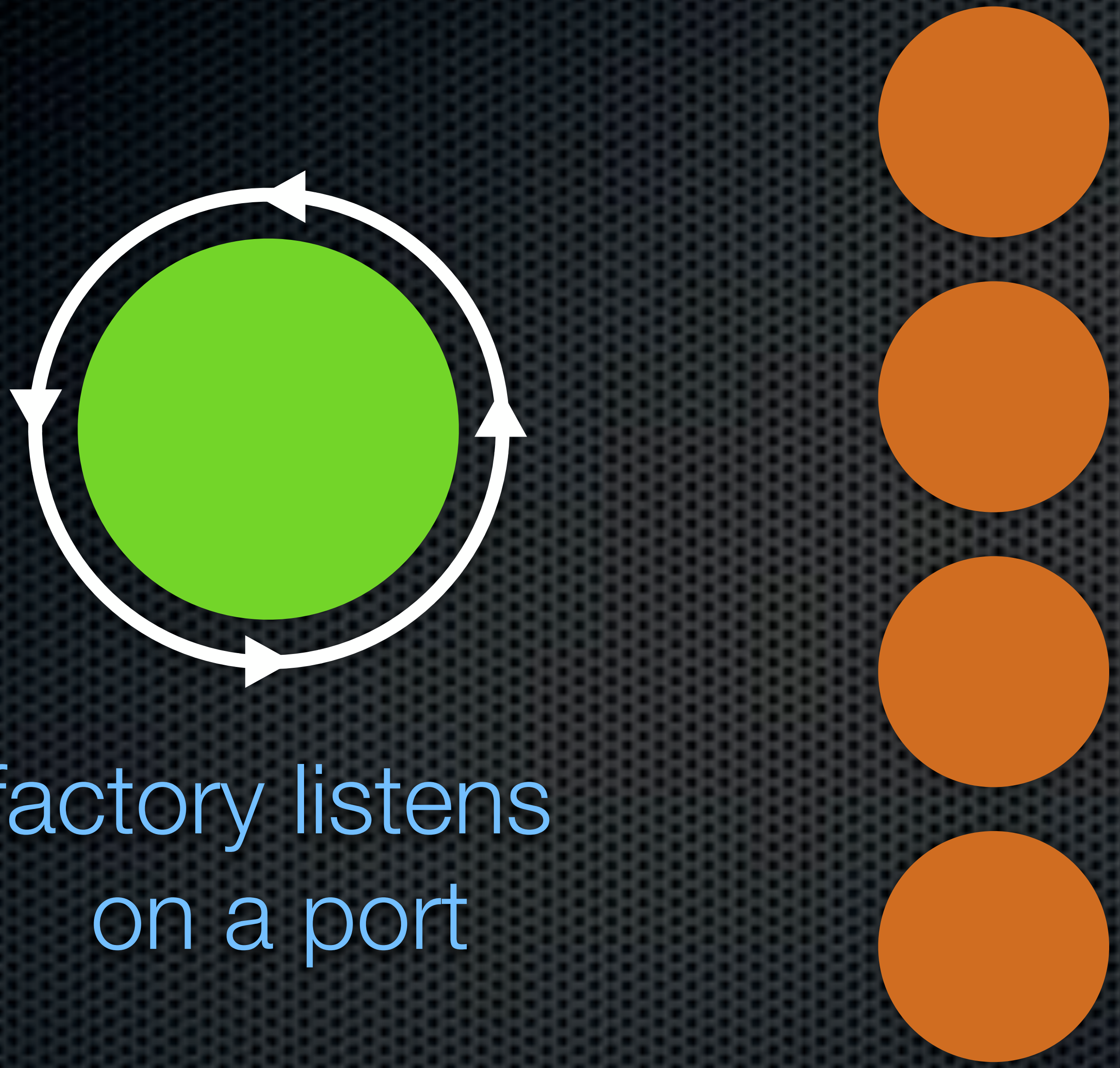
```
from twisted.internet import reactor




reactor.listenTCP(8000, ?????????)
reactor.run()
```

Listen on a port, then do something

```
from twisted.internet import reactor, protocol


factory = protocol.ServerFactory()
factory.protocol = protocol.Protocol


reactor.listenTCP(8000, factory)
reactor.run()
```

Run this!

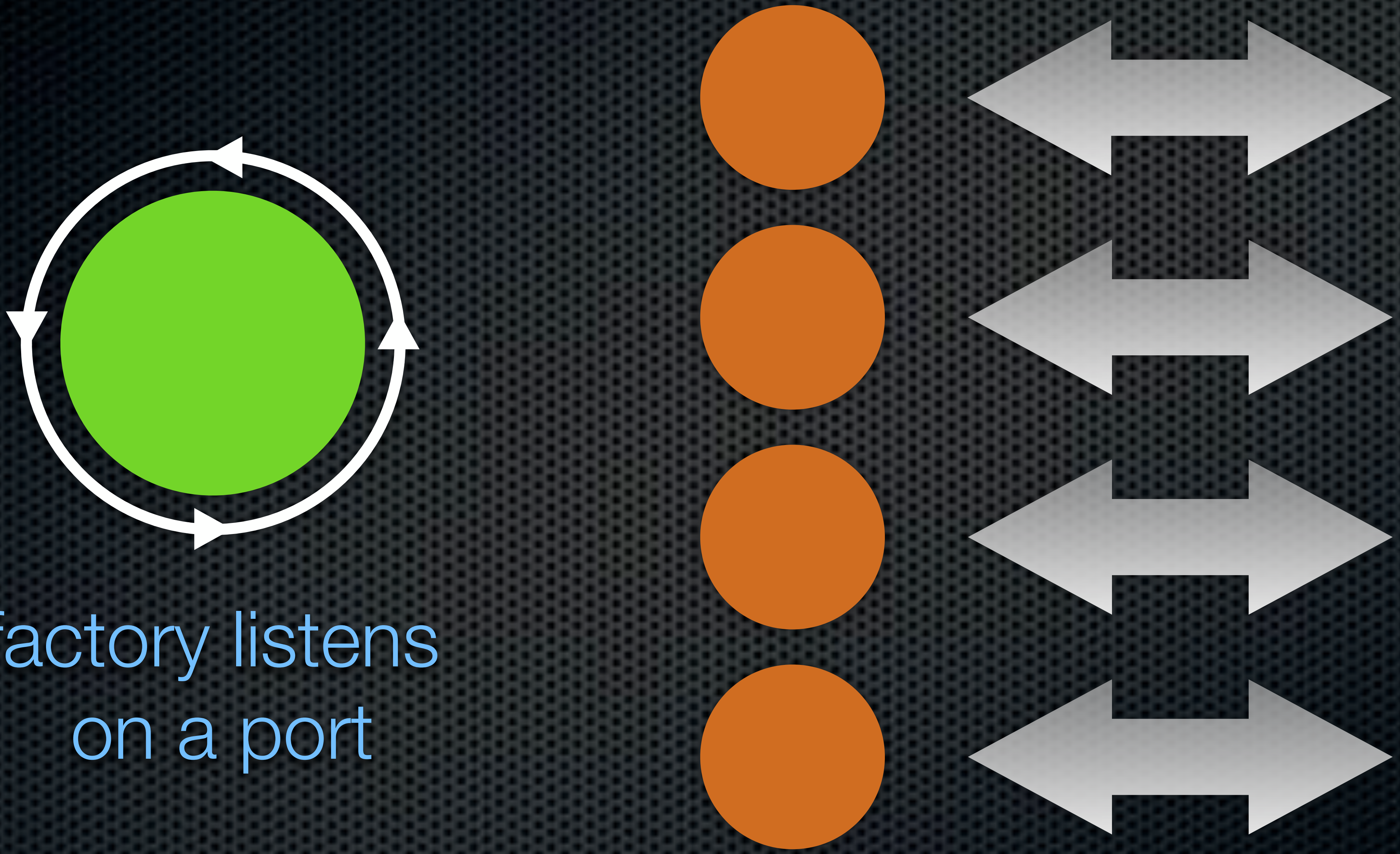
Handle request elsewhere



factory listens on a port

for each connection, a protocol instance is created

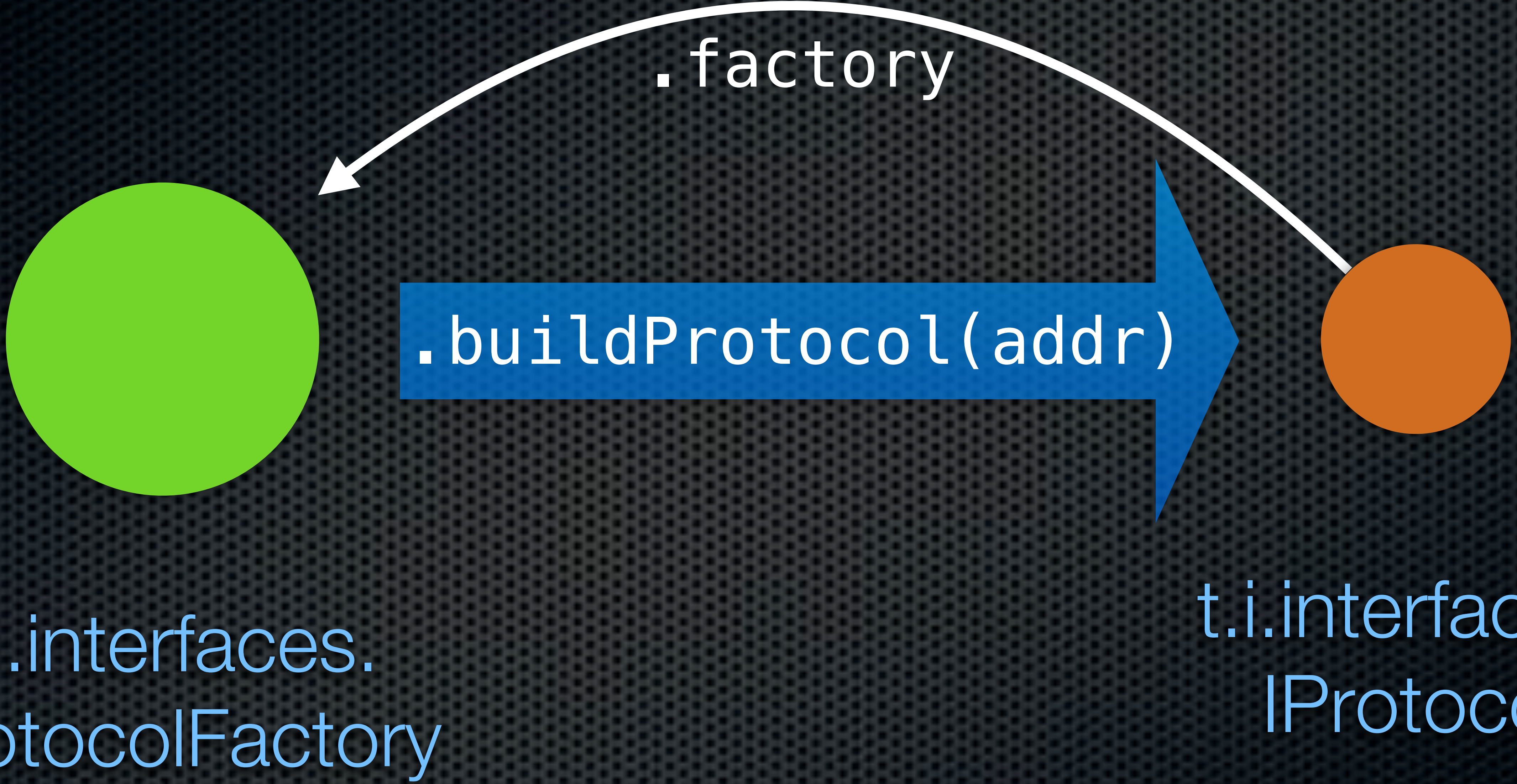
Handle request elsewhere



factory listens on a port

for each connection, a protocol instance is created

Handle request elsewhere



Twisted Uppercase Server

Server that returns the data, uppercased

Twisted Uppercase Server

```python
from twisted.internet import reactor, protocol


class UpperProtocol(protocol.Protocol):

    def connectionMade(self):
        self.transport.write('Hi! Send me text to convert to uppercase\n')

    def connectionLost(self, reason):
        pass

    def dataReceived(self, data):
        self.transport.write(data.upper())
        self.transport.loseConnection()

factory = protocol.ServerFactory()
factory.protocol = UpperProtocol

reactor.listenTCP(8000, factory)
reactor.run()
```

upperserver.py

Run this!

Twisted Uppercase Server

```python
from twisted.internet import reactor, protocol


class UpperProtocol(protocol.Protocol):

    def connectionMade(self):
        self.transport.write('Hi! Send me text to convert to uppercase\n')

    def connectionLost(self, reason):
        pass

    def dataReceived(self, data):
        self.transport.write(data.upper())
        self.transport.loseConnection()


factory = protocol.ServerFactory()
factory.protocol = UpperProtocol

reactor.listenTCP(8000, factory)
reactor.run()
```

Run this!

Twisted Uppercase Server

```
$ telnet localhost 8000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Hi! Send me text to convert to uppercase
twisted is cool
TWISTED IS COOL
Connection closed by foreign host.
$
```

A better client

multiclient.py

```
import socket

def make_connection(host, port, data_to_send):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((host, port))
    s.send(data_to_send)
    s.send('\r\n')
    b = []
    while True:
        data = s.recv(1024)
        if data:
            b.append(data)
        else:
            break

    return ''.join(b)


if __name__ == '__main__':
    import sys
    host, port = sys.argv[1].split(':')
    data_to_send = sys.argv[2:]

    print “sending”, d
    for d in data_to_send:
        print make_connection(host, int(port), d)
```

Run this!

A better client (output)

```
$ python multiclient.py 127.0.0.1:8000 a b c d
sending a
Hi! Send me text to convert to uppercase
A

sending b
Hi! Send me text to convert to uppercase
B

sending c
Hi! Send me text to convert to uppercase
C

sending d
Hi! Send me text to convert to uppercase
D
```

Questions so far?

Exercise coming up!

Exercise 1

- Count connected clients
- Announce number of connected clients when connecting
- **HINT**: Protocols have a “factory” instance attribute

Counting uppercase server

```
from twisted.internet import reactor, protocol


class UpperProtocol(protocol.Protocol):

    def connectionMade(self):
        self.factory.count += 1
        self.transport.write('Hi! There are %d clients\n' % self.factory.count)

    def connectionLost(self, reason):
        self.factory.count -= 1

    def dataReceived(self, data):
        self.transport.write(data.upper())
        self.transport.loseConnection()

class CountingFactory(protocol.ServerFactory):
    protocol = UpperProtocol
    count = 0

reactor.listenTCP(8000, CountingFactory())
reactor.run()
```

upperserver_ex.py

Run this!

An even better client

```
import threading
from multiclient import make_connection

def t_connection(host, port, d):
    print 'sending', d
    print make_connection(host, port, d)



if __name__ == '__main__':

    import sys
    host, port = sys.argv[1].split(':')
    data_to_send = sys.argv[2:]

    threads = []
    for d in data_to_send:
        t = threading.Thread(target=t_connection, args=(host, int(port), d))
        t.start()
        threads.append(t)

    for t in threads:
        t.join()

    print 'finished'
```

threadedclient.py

Run this!

A brief recap

- reactor runs forever
- a factory instance is tied to a specific port
- protocol instances are created for each client
- implement specific methods on the protocol to add functionality

Twisted Proxy Server (v1)

- Client sends an URL followed by a telnet newline \r\n
- The server returns the contents of that URL
- Connection is closed

“followed by a newline”

```python
from twisted.internet import protocol
class MyProtocol(protocol.Protocol):
    def connectionMade(self):
        self.buffer = []

    def dataReceived(self, data):
        self.buffer.append(data)
        if '\r\n' in data:
            line, rest = ''.join(self.buffer).split('\n')
            self.buffer = [rest]
        print line
```

```python
from twisted.protocols import basic
class MyProtocol(basic.LineReceiver):
    def lineReceived(self, line):
        print line
```

twisted.protocols

amp basic dict finger ftp
gps htb ident loopback
memcache mice pcp
policies portforward postfix
shoutcast sip socks
stateful telnet tls wire

twisted.protocols.basic

NetstringReceiver LineOnlyReceiver LineReceiver IntNStringReceiver Int32StringReceiver Int16StringReceiver Int8StringReceiver StatefulStringProtocol FileSender

twisted.protocols

Don’t reinvent the wheel!

Twisted Proxy Server (v1)

```
from twisted.internet import reactor, protocol
from twisted.protocols import basic

import urllib2
import time

class ProxyProtocol(basic.LineReceiver):

    def lineReceived(self, line):
        if not line.startswith('http://'):
            return
        start = time.time()
        print 'fetching', line
        data = urllib2.urlopen(line).read()
        print 'fetched', line
        self.transport.write(data)
        self.transport.loseConnection()
        print 'took', time.time() - start

factory = protocol.ServerFactory()
factory.protocol = ProxyProtocol

reactor.listenTCP(8000, factory)
reactor.run()
```

proxy1.py

Run this!

Let’s time it!

```
import threading
from multiclient import make_connection
import time

def t_connection(host, port, d):
    start = time.time()
    make_connection(host, port, d)
    print d, 'took', time.time() - start


if __name__ == '__main__':
    import sys
    host, port = sys.argv[1].split(':')
    data_to_send = sys.argv[2:]

    threads = []
    overallstart = time.time()
    for d in data_to_send:
        t = threading.Thread(target=t_connection, args=(host, int(port),
d))
        t.start()
        threads.append(t)

    for t in threads:
        t.join()

    print 'finished in', time.time() - overallstart
```

timingclient.py

Run this!

Let’s time it! (output)

```
$ python timingclient.py 127.0.0.1:8000 [...]
http://orestis.gr took 2.02559900284
http://amazon.com took 2.02640080452
http://apple.com took 3.81526899338
http://google.com took 3.81563591957
finished in 3.81683182716
```

Client

Server

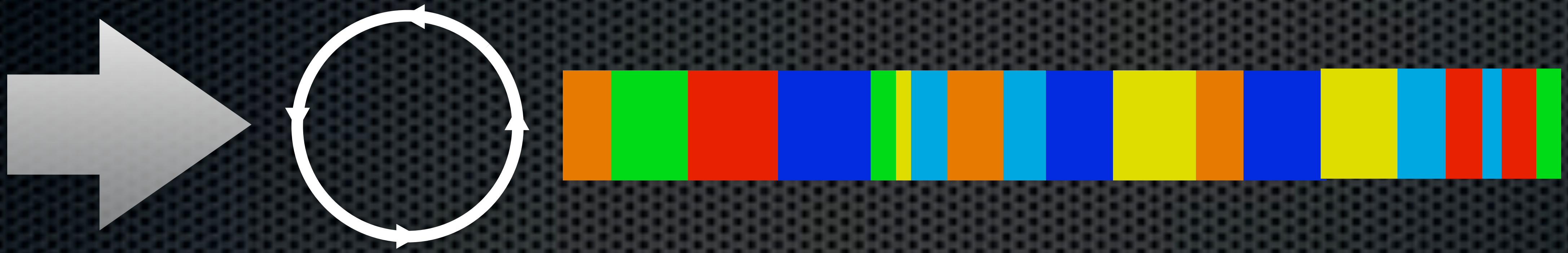
```
$ python proxy1.py

fetched http://orestis.gr
took 0.566583156586

fetched http://amazon.com
took 1.45738196373

fetched http://apple.com
took 1.01898193359

fetched http://google.com
took 0.770982027054
```

Something's wrong!

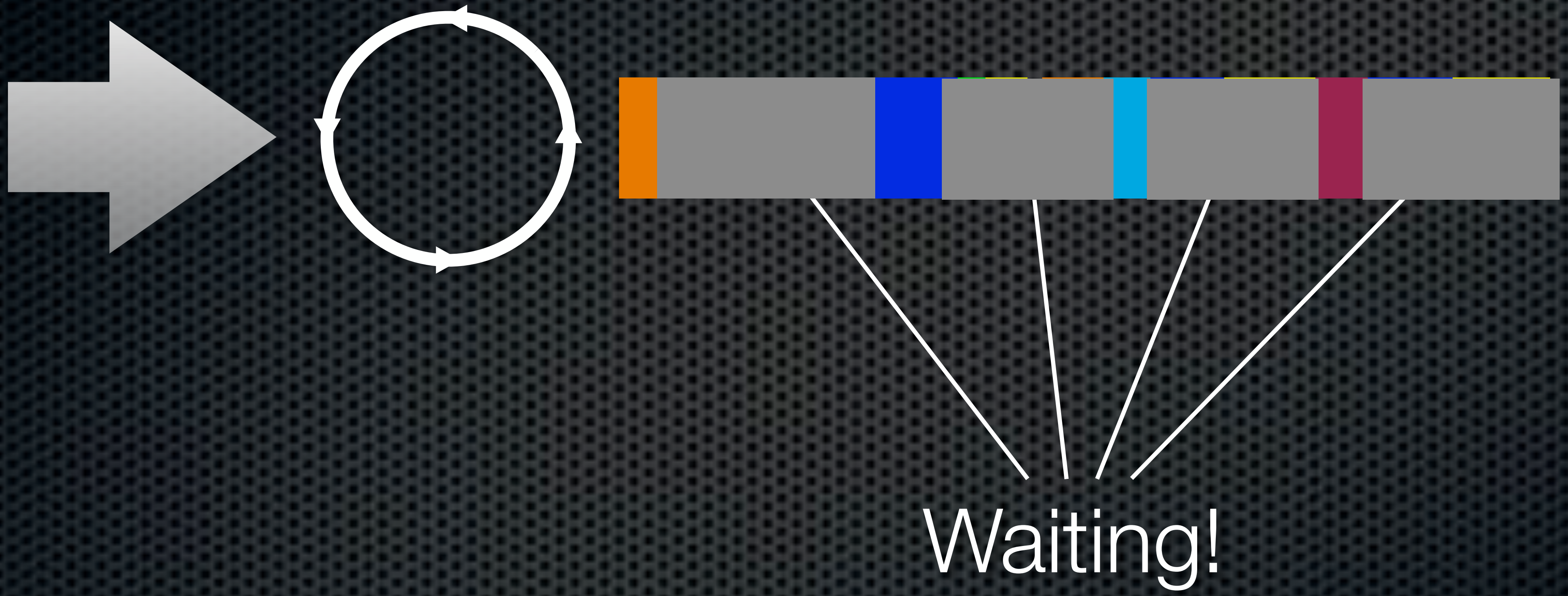
	Individual requests
	0.770982027054
	0.566583156586
	1.45738196373
	1.01898193359
SUM:	**3.81392908096**

Threaded client
3.81683182716

Eliminate Blocking?



But in this case...



The culprit

```
print 'fetching', line
data = urllib2.urlopen(line).read()
print 'fetched', line
```

The culprit

```
print '[illegible]tching', line

data = u[illegible]en(line).read()

print 'fetched', [illegible]
```

Forbidden

You didn’t think it’d be that easy, right?

The callbacks must be **cooperative**

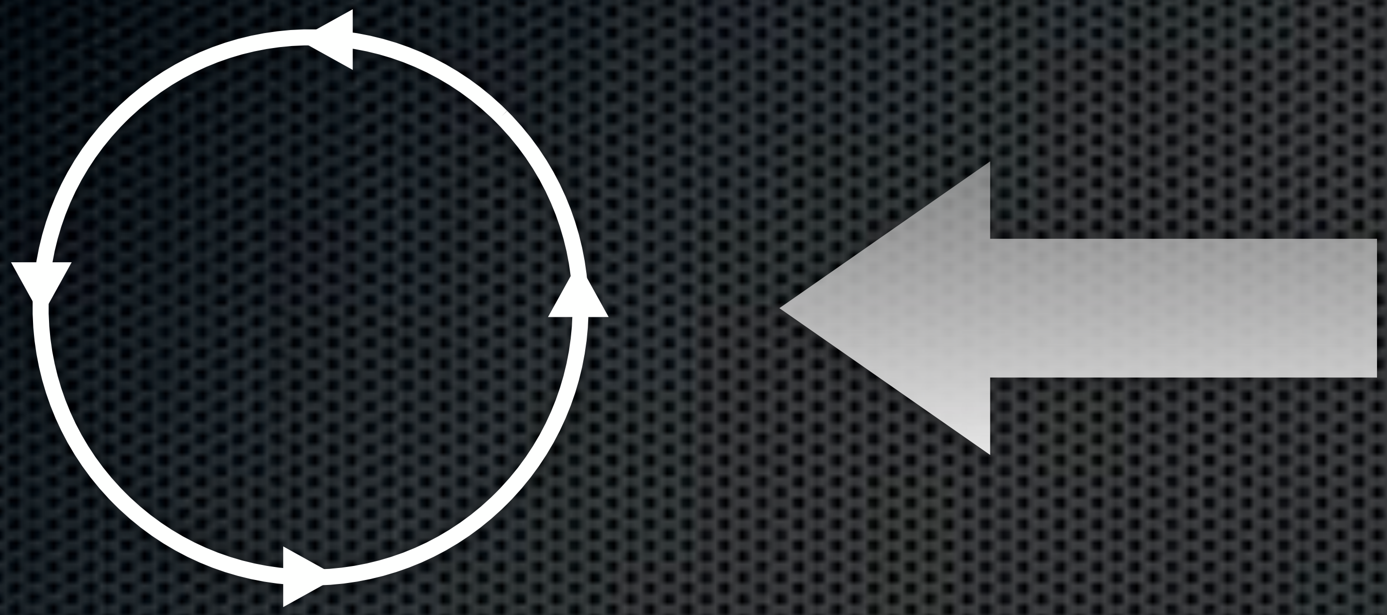
The callbacks must be **cooperative**

- When accessing the network, **return control back to the loop**
- The loop will call your code when the network is ready

Network programming

Server listens
on a port

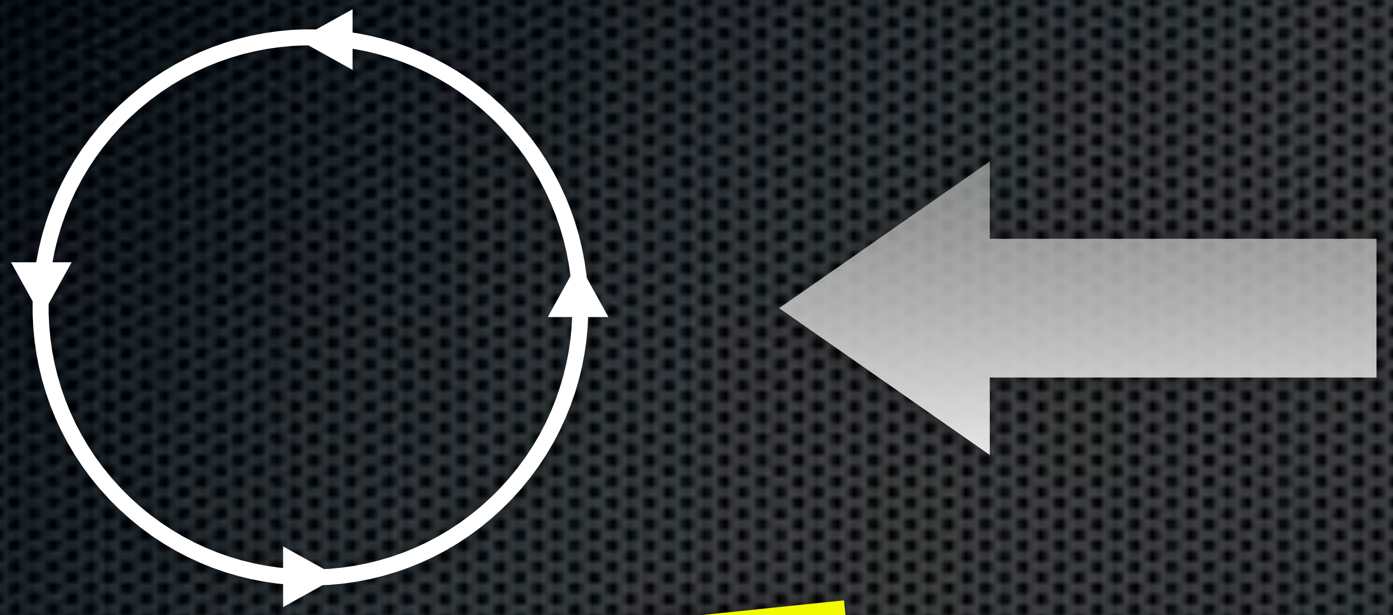
Network programming



Server listens
on a port

Client connects
to port

Network programming



Customer asks for a toasted panini

Cook waits at the stall

High-level panini stall

```
import stall

panini = stall.order_panini(spec)
eat(panini)
```

```
data = urllib2.urlopen(line).read()
```

Panini stall

- Wait my turn
- Place order
- Wait for panini
- Eat panini

Network

- Make connection
- Send request
- Read data
- Use data

Low-level panini stall

```
import stall
import time

stall.enter_queue()
while not stall.is_my_turn():
    time.sleep(0.1)
stall.place_order(spec)
while not stall.order_is_ready():
    time.sleep(0.1)
panini = stall.get_panini()
eat(panini)
```

Low-level panini stall

```
import stall
import time

stall.enter_queue()
while not stall.is_my_turn():
    time.sleep(0.1)
stall.place_order(spec)
while not stall.order_is_ready():
    time.sleep(0.1)
panini = stall.get_panini()
eat(panini)
```

Waste of time!

- We are idling the CPU!
- Nothing else can run!
- How selfish of us!

Solution: Callbacks!

Callbacks, you know...

```
$.ajax({
   type: "POST",
   url: "some.php",
   data: "name=John&location=Boston",
   success: function(msg){
      alert( "Data Saved: " + msg );
   }
});
```

Callbacks, you know...

```
$.ajax({
   type: "POST",
   url: "some.php",
   data: "name=John&location=Boston",
   success: function(msg){
      alert( "Data Saved: " + msg );
   }
});
```

High-level panini stall

```
import stall

panini = stall.order_panini(spec)
eat(panini)
```

```
import stall

stall.order_panini(spec, when_ready=eat)
```

Callbacks can be messy

- Add error handling?
- Pass the result around?
- Cancel the original request?
- Consistent API?

Introducing Deferred

twisted.internet.defer

A Deferred is...

- A promise of a result...
- A result that will appear in the future...
- A result you can pass around...
- Something you can attach callbacks to.
- A (mostly) standalone module!

Deferred Panini

```
import stall

def eat(panini):
    print ”YUM! I’ve just eated a”, panini
deferred = stall.order_panini(spec)
deferred.addCallback(eat)
```

Deferreds are Everywhere

Get used to them!

Twisted has re-implementations of most of the stdlib.

They had to do it - not a case of NIH!

So....

```
import urllib2
data = urllib2.urlopen(url).read()
print data
```

```
from twisted.web.client import getPage

def got_page(data):
    print data

deferred = getPage(url)
deferred.addCallback(got_page)
```

In context...

```
def lineReceived(self, line):
    if not line.startswith('http://'):
        return
    start = time.time()
    print 'fetching', line
    def gotData(data):
        print 'fetched', line
        self.transport.write(data)
        self.transport.loseConnection()
        print 'took', time.time() - start
    deferredData = getPage(line)
    deferredData.addCallback(gotData)
```

Asynchronous

```
def lineReceived(self, line):
    if not line.startswith('http://'):
        return
    start = time.time()
    print 'fetching', line
    data = urllib2.urlopen(line).read()
    print 'fetched', line
    self.transport.write(data)
    self.transport.loseConnection()
    print 'took', time.time() - start
```

Synchronous

Python reminder:

```python
def lineReceived(self, line):
    if not line.startswith('http://'):
        return
    start = time.time()
    print 'fetching', line
    def gotData(data):
        print 'fetched', line
        self.transport.write(data)
        self.transport.loseConnection()
        print 'took', time.time() - start
    deferredData = getPage(line)
    deferredData.addCallback(gotData)
```

Python reminder:

```
def lineReceived(self, line):
    if not line.startswith('http://'):
        return
    start = time.time()
    print 'fetching', line
    def gotData(data):
        print 'fetched', line
        self.transport.write(data)
        self.transport.loseConnection()
        print 'took', time.time() - start
    deferredData = getPage(line)
    deferredData.addCallback(gotData)
```

A tidier way

```
class ProxyProtocol(basic.LineReceiver):
    def writeDataAndLoseConnection(self, data, url, starttime):
        print 'fetched', url
        self.transport.write(data)
        self.transport.loseConnection()
        print 'took', time.time() - starttime

    def lineReceived(self, line):
        if not line.startswith('http://'):
            return
        start = time.time()
        print 'fetching', line
        deferredData = getPage(line)
        deferredData.addCallback(self.writeDataAndLoseConnection,
                                 line, start)
```

Twisted Proxy Server (v2)

```
from twisted.web.client import getPage
from twisted.internet import reactor, protocol
from twisted.protocols import basic

import time

class ProxyProtocol(basic.LineReceiver):
    def writeDataAndLoseConnection(self, data, url, starttime):
        print 'fetched', url,
        self.transport.write(data)
        self.transport.loseConnection()
        print 'took', time.time() - starttime

    def lineReceived(self, line):
        if not line.startswith('http://'):
            return
        start = time.time()
        print 'fetching', line
        deferredData = getPage(line)
        deferredData.addCallback(writeDataAndLoseConnection,
                                 line, self.transport, start)

factory = protocol.ServerFactory()
factory.protocol = ProxyProtocol

reactor.listenTCP(8000, factory)
reactor.run()
```

proxy2.py

Run this!

Let’s time it! (output)

```
$ python timingclient.py 127.0.0.1:8000 [...]
http://orestis.gr took 0.487771987915
http://apple.com took 0.853055000305
http://google.com took 1.00087499619
http://amazon.com took 1.58436584473
finished in 1.5850892067
```

Client

Server

```
$ python proxy2.py

fetching http://orestis.gr
fetching http://amazon.com
fetching http://google.com
fetching http://apple.com
fetched http://orestis.gr took 0.486361026764
fetched http://apple.com took 0.850247859955
fetched http://google.com took 0.998661994934
fetched http://amazon.com took 1.58235692978
```

Much better!

	Individual requests
	0.486361026764
	0.850247859955
	0.998661994934
	1.58235692978
SUM:	**3.917627811433**

Threaded client
1.5850892067

The callbacks must be **cooperative**

- When accessing the network, **return control back to the loop**
- The loop will call your code when the network is ready

never call blocking functions

return control to the loop

Exercise 2a

- Implement a caching proxy server!
- Save response data in plain dict
- Lookup response data
- **QUESTION:** Where should you store the dict?

Caching Proxy Server (v1)

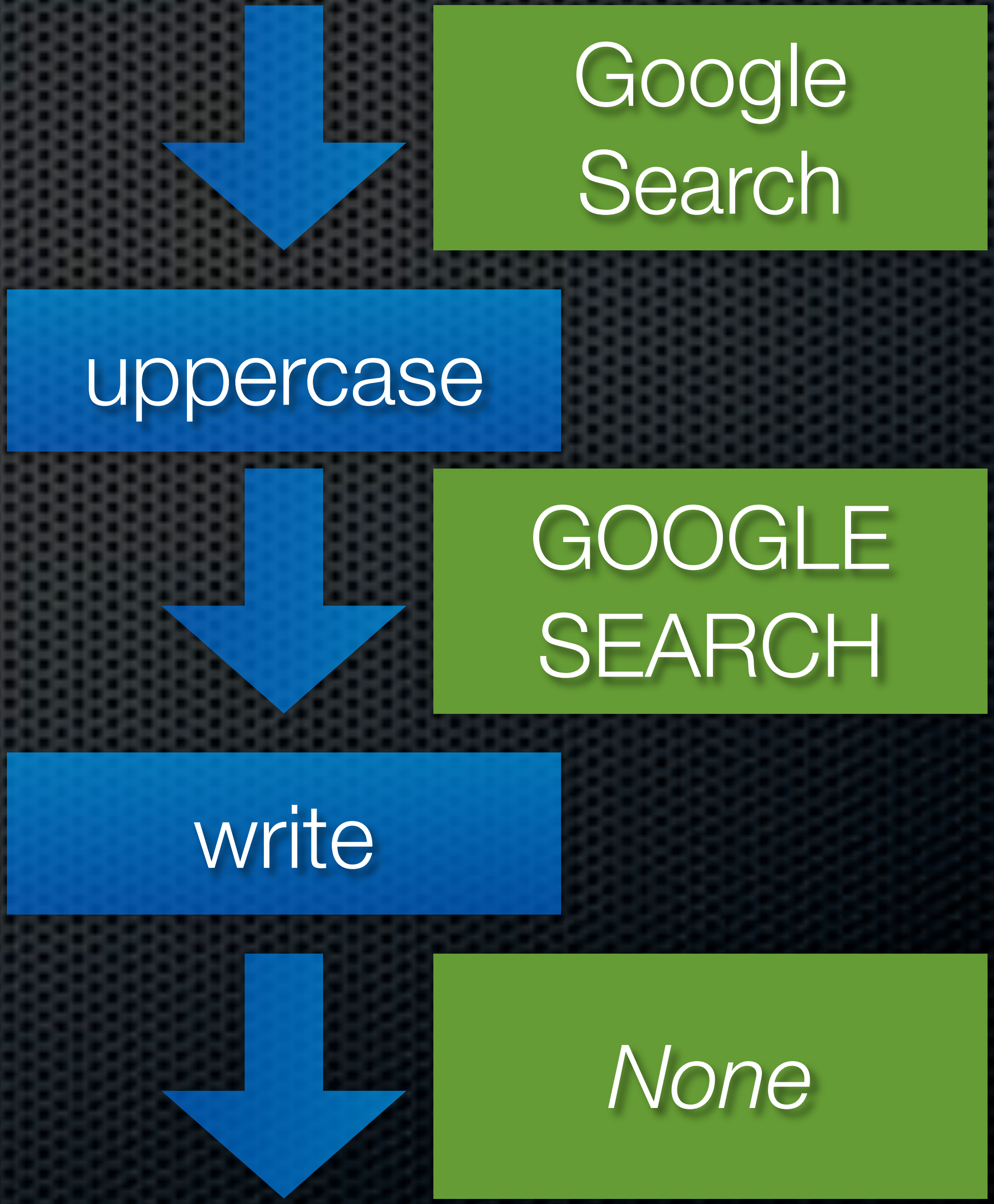
```
from twisted.web.client import getPage
from twisted.internet import reactor, protocol
from twisted.protocols import basic

class CachingProxyProtocol(basic.LineReceiver):

    def lineReceived(self, line):
        if not line.startswith('http://'):
            return
        try:
            data = self.factory.cache[line]
            self.transport.write(data)
            self.transport.loseConnection()
        except KeyError:
            def gotData(data):
                self.factory.cache[line] = data
                self.transport.write(data)
                self.transport.loseConnection()
            deferredData = getPage(line)
            deferredData.addCallback(gotData)

class CachingProxyFactory(protocol.ServerFactory):
    protocol = CachingProxyProtocol
    cache = {}

reactor.listenTCP(8000, CachingProxyFactory())
reactor.run()
```

proxy2_ex1.py

Run this!

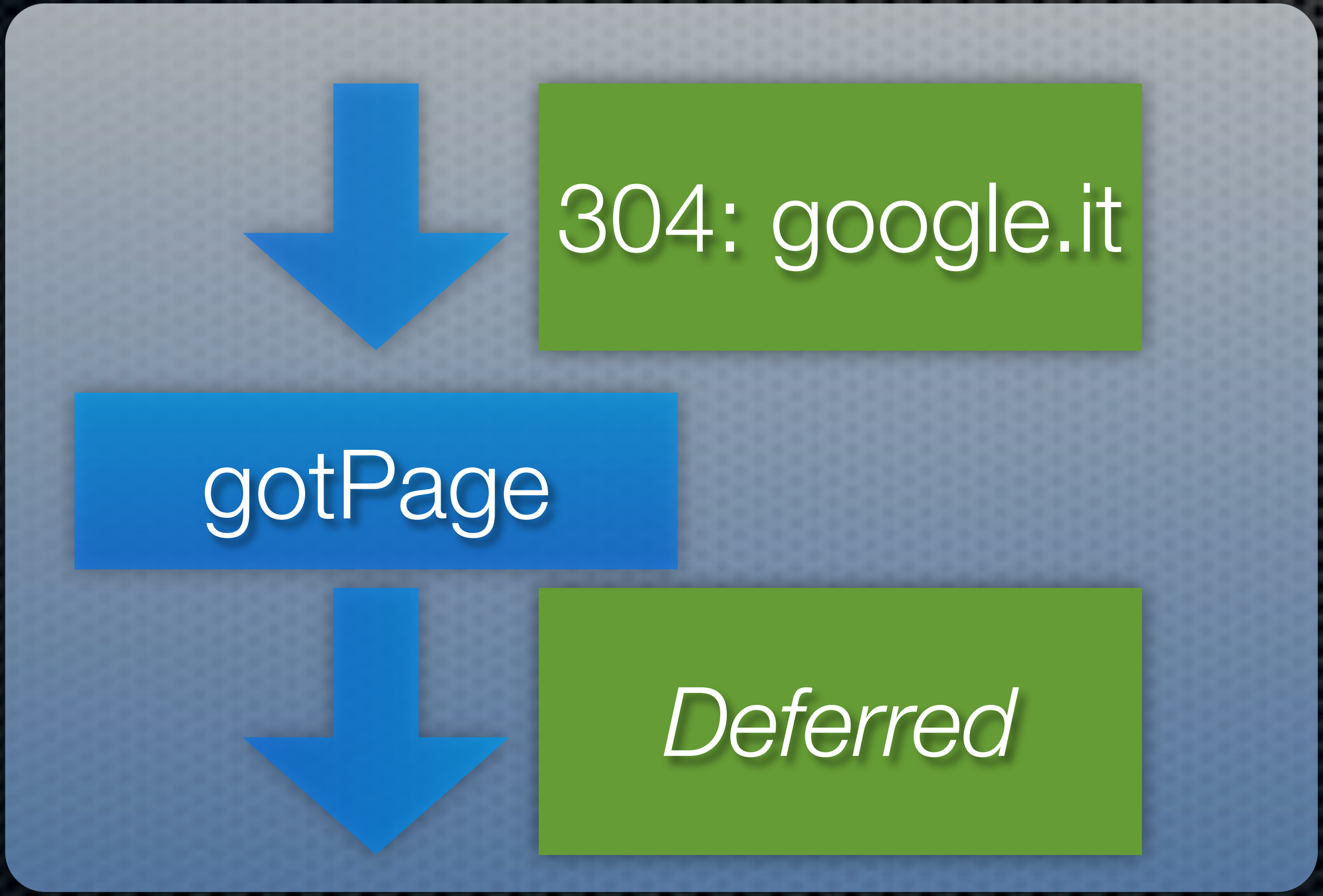
Cool Deferred Features: Chaining

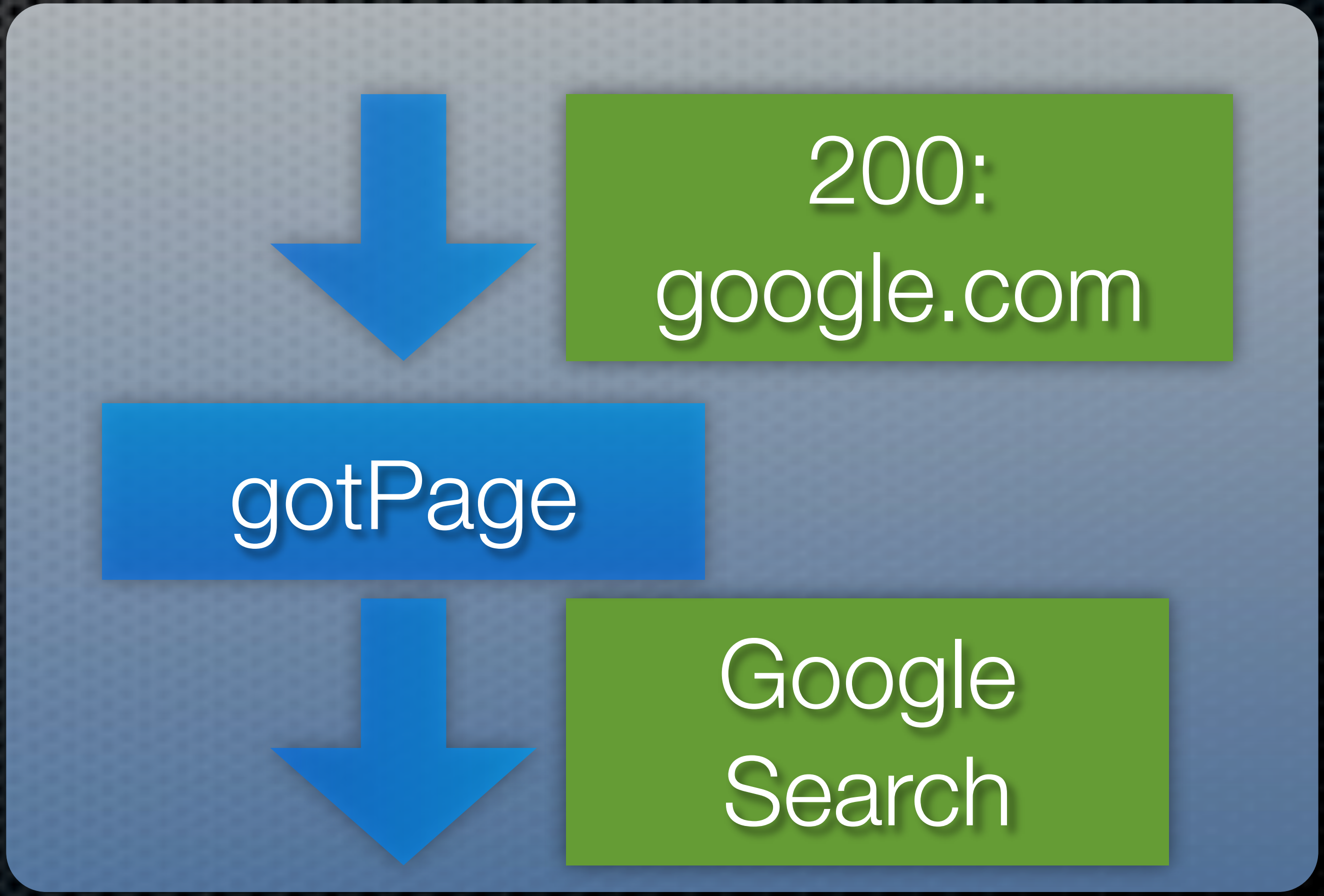
```python
from twisted.web.client import getPage

def uppercase(s):
    return s.upper()

def write(s):
    print s

d = getPage("http://www.google.com")
d.addCallback(uppercase)
d.addCallback(write)
```



Cool Deferred Features: Deferring

```python
from twisted.web.client import getPage

def gotPage(data):
    newURL = getRedirect(data)
    if newURL:
        return getPage(newURL)
    else:
        return data
```

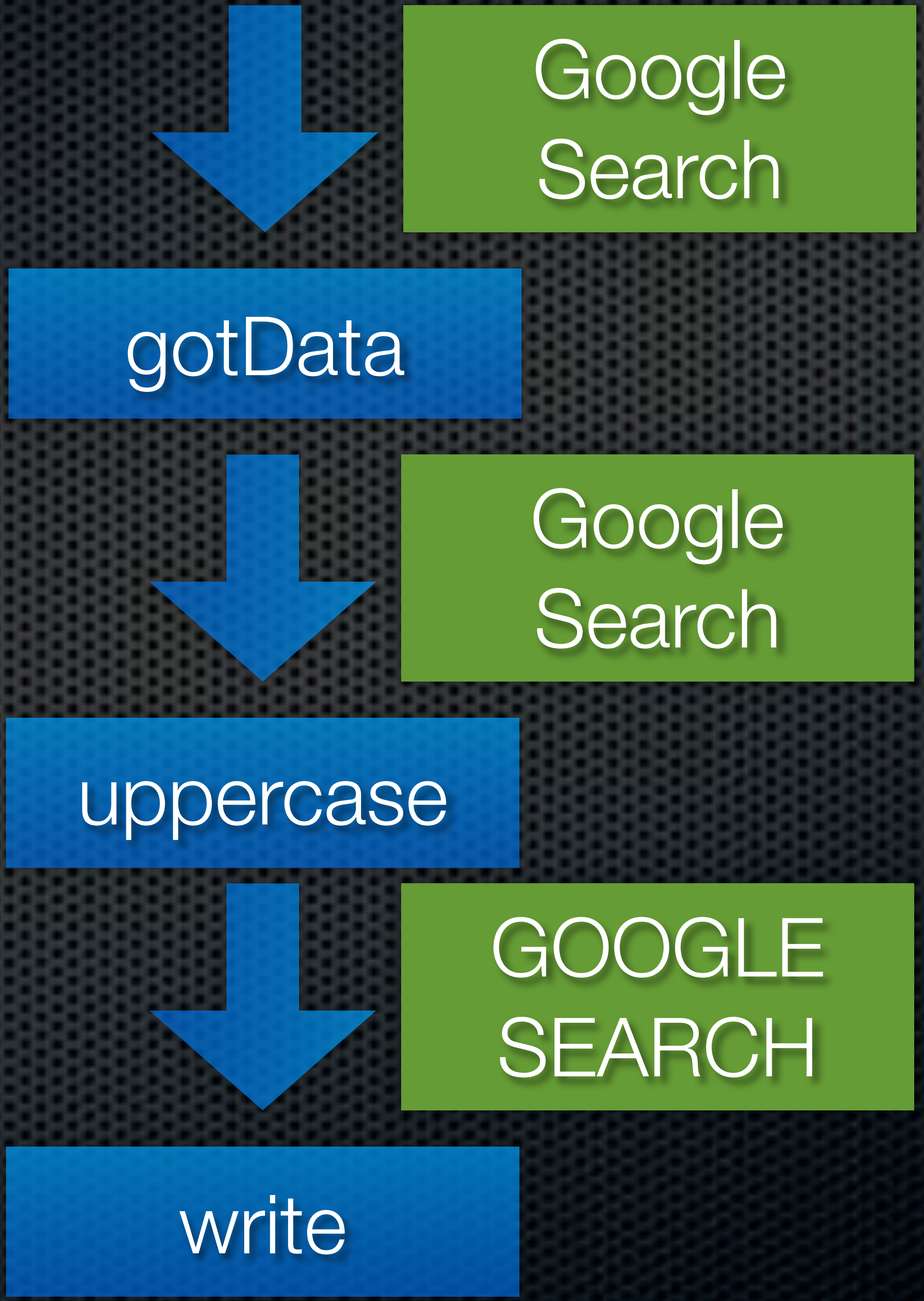




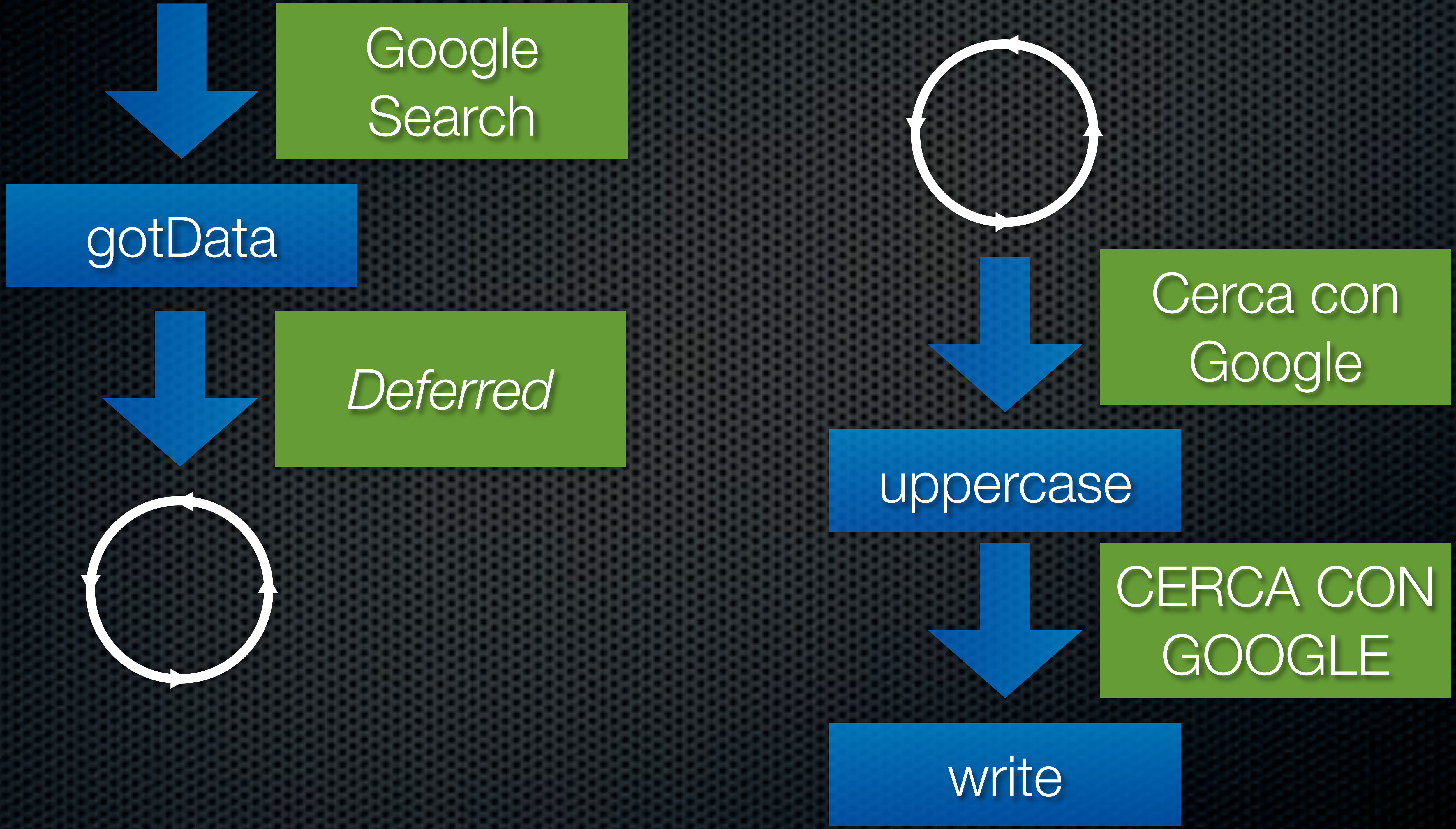
Chaining and Deferring

```
from twisted.web.client import getPage

def gotPage(data):
    newURL = getRedirect(data)
    if newURL:
        return getPage(newURL)
    else:
        return data

def uppercase(s):
    return s.upper()

def write(s):
    print s

d = getPage("http://www.google.com")
d.addCallback(gotPage)
d.addCallback(uppercase)
d.addCallback(write)
```

Chaining and Deferring



Chaining and Deferring



Exercise 2b

- Put those cool features into use!
- One method to get a page, returning a deferred
- One callback to store it to cache
- One callback to write to transport
- **HINT:** Use defer.succeed(data) to return a “primed” Deferred

Caching Proxy Server (v2)

```python
from twisted.web.client import getPage
from twisted.internet import reactor, protocol, defer
from twisted.protocols import basic

class CachingProxyProtocol(basic.LineReceiver):

    def _getPage(self, url):
        try:
            data = self.factory.cache[url]
            return defer.succeed(data)
        except KeyError:
            d = getPage(url)
            d.addCallback(self._storeInCache, url, self.factory.cache)
            return d

    def _storeInCache(self, data, url, cache):
        cache[url] = data
        return data

    def writeDataToTransport(self, data):
        self.transport.write(data)
        self.transport.loseConnection()

    def lineReceived(self, line):
        if not line.startswith('http://'):
            return
        deferredData = self._getPage(line)
        deferredData.addCallback(self.writeDataToTransport)

class CachingProxyFactory(protocol.ServerFactory):
    protocol = CachingProxyProtocol
    cache = {}

reactor.listenTCP(8000, CachingProxyFactory())
reactor.run()
```

proxy2_ex2.py

Run this!

BREAK

Write questions on whiteboard

Writing clients

the Twisted way

Remember this?

```
import socket

def make_connection(host, port, data_to_send):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((host, port))
    s.send(data_to_send)
    s.send('\r\n')
    b = []
    while True:
        data = s.recv(1024)
        if data:
            b.append(data)
        else:
            break

    return ''.join(b)


if __name__ == '__main__':
    import sys
    host, port = sys.argv[1].split(':')
    data_to_send = sys.argv[2:]

    for d in data_to_send:
        print make_connection(host, int(port), d)
```

Remember this?

```python
import socket

def make_connection(host, port, data_to_send):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((host, port))
    s.send(data_to_send)
    s.send('\r\n')
    b = []
    while True:
        data = s.recv(1024)
        if data:
            b.append(data)
        else:
            break

    return ''.join(b)


if __name__ == '__main__':
    import sys
    host, port = sys.argv[1].split(':')
    data_to_send = sys.argv[2:]

    for d in data_to_send:
        print make_connection(host, int(port), d)
```

Remember this?

```
import socket

def make_connection(host, port, data_to_send):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((host, port))
    s.send(data_to_send)
    s.send('\r\n'
    b = []
    while True:
        data =   rec  1  4
        if data
            b.append
        else:
            break

    return ''.join(b)


if __name__ == '__main__':
    import sys
    host, port = sys.argv[1].split(':')
    data_to_send = sys.argv[2:]

    for d in data_to_send:
        print make_connection(host, int(port), d)
```

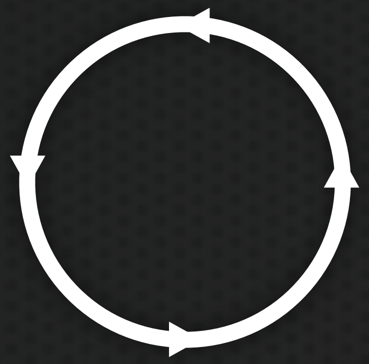
Forbidden

Return control to the loop

In clients, too

Twisted reactor loop

```
from twisted.internet import reactor
```



```
reactor.run()
```

Connect to a host, then do something

```
from twisted.internet import reactor, protocol





reactor.connectTCP(“127.0.0.1”, 8000, ?????)
reactor.run()
```

Connect to a host, then do something

```
from twisted.internet import reactor, protocol


factory = protocol.ClientFactory()
factory.protocol = protocol.Protocol


reactor.connectTCP(“127.0.0.1”, 8000, factory)
reactor.run()
```

Run this!

Let’s write a twisted client

Run the uppercase server

Twisted Simple Client (v1)

```
from twisted.internet import reactor, protocol


class UppercaseClientProtocol(protocol.Protocol):
    def connectionMade(self):
        self.transport.write(self.factory.text)
        self.transport.write('\r\n')

    def dataReceived(self, data):
        print data


if __name__ == '__main__':
    import sys
    host, port = sys.argv[1].split(':')
    data_to_send = sys.argv[2:]

    for d in data_to_send:
        print 'sending', d
        factory = protocol.ClientFactory()
        factory.protocol = UppercaseClientProtocol
        factory.text = d
        reactor.connectTCP(host, int(port), factory)

    reactor.run()
```

simpleclient.py

Run this!

Simple Client v1 (output)

```
$ python simpleclient.py 127.0.0.1:8000 a b c d
sending a
sending b
sending c
sending d
Hi! Send me text to convert to uppercase
D
Hi! Send me text to convert to uppercase
A
Hi! Send me text to convert to uppercase
B
Hi! Send me text to convert to uppercase
C

^C
$
```

Observations

- Data returns with random order
- We cannot access the returned data
- Loop never stops
- Performance?

We need to gather results

hmm, what should we use?

Twisted Simple Client (v2)

```
from twisted.internet import reactor, protocol, defer


class UppercaseClientProtocol(protocol.Protocol):
    def connectionMade(self):
        self.transport.write(self.factory.text)
        self.transport.write('\r\n')
        self.buffer = []

    def dataReceived(self, data):
        self.buffer.append(data)

    def connectionLost(self, reason):
        alldata = ''.join(self.buffer)
        self.factory.deferred.callback(alldata)


def gotData(data, request):
    print 'received response for', request
    print data


if __name__ == '__main__':
    import sys
    host, port = sys.argv[1].split(':')
    data_to_send = sys.argv[2:]

    for data in data_to_send:
        print 'sending', data
        d = defer.Deferred()
        d.addCallback(gotData, data)
        factory = protocol.ClientFactory()
        factory.protocol = UppercaseClientProtocol
        factory.text = data
        factory.deferred = d
        reactor.connectTCP(host, int(port), factory)

    reactor.run()
```

simpleclient2.py

Run this!

We created our own Defered instance

create one...

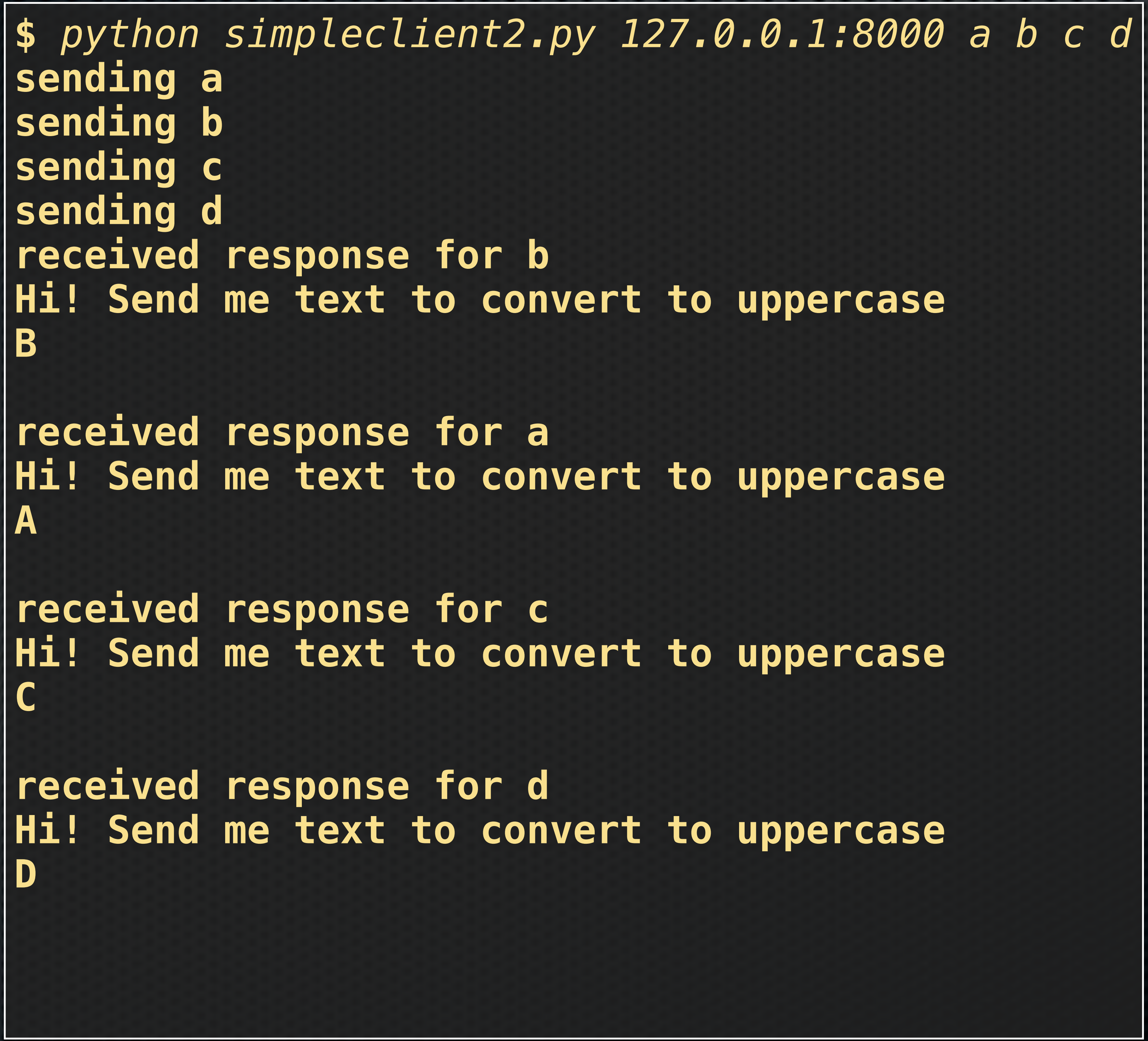
```
d = defer.Deferred()
```

...pass it around...

...then in the future:

```
d.callback(result)
```

Simple Client v2 (output)



```
$ python simpleclient2.py 127.0.0.1:8000 a b c d
sending a
sending b
sending c
sending d
received response for b
Hi! Send me text to convert to uppercase
B

received response for a
Hi! Send me text to convert to uppercase
A

received response for c
Hi! Send me text to convert to uppercase
C

received response for d
Hi! Send me text to convert to uppercase
D
```

Observations

- ~~Data returns with random order~~
- ~~We cannot access the returned data~~
- Loop never stops
- Performance?

Stop the loop when everything is finished

wait until all Deferreds have fired

Introducing DeferredList

- A list of Deferreds!
- You create it with a list of Deferreds
- .addCallback
- When all the Deferreds have finished, its callback fires.

Introducing DeferredList

```
from twisted.internet import defer
from twisted.web.client import getPage

pages = ['http://www.google.com', 'http://www.orestis.gr', ...]

all_deferreds = []
for page in pages:
    d = getPage(page)
    d.addCallback(gotPage)
    all_deferreds.append(d)

deferredList = defer.DeferredList(all_deferreds)
def all_finished(results):
    print "ALL PAGES FINISHED"
deferredList.addCallback(all_finished)
```

Twisted Simple Client (v3)

```python
from twisted.internet import reactor, protocol, defer

from simpleclient2 import gotData, UppercaseClientProtocol


if __name__ == '__main__':
    import sys
    host, port = sys.argv[1].split(':')
    data_to_send = sys.argv[2:]

    all_deferreds = []
    for data in data_to_send:
        print 'sending', data
        d = defer.Deferred()
        d.addCallback(gotData, data)
        factory = protocol.ClientFactory()
        factory.protocol = UppercaseClientProtocol
        factory.text = data
        factory.deferred = d
        all_deferreds.append(d)
        reactor.connectTCP(host, int(port), factory)

    deferredList = defer.DeferredList(all_deferreds)
    def all_done(results):
        reactor.stop()
    deferredList.addCallback(all_done)


    reactor.run()
```

simpleclient3.py

Run this!

Simple Client v3 (output)

```
$ python simpleclient2.py 127.0.0.1:8000 a b c d
sending a
sending b
sending c
sending d
received response for b
Hi! Send me text to convert to uppercase
B

received response for a
Hi! Send me text to convert to uppercase
A

received response for c
Hi! Send me text to convert to uppercase
C

received response for d
Hi! Send me text to convert to uppercase
D


$
```

Observations

- ~~Data returns with random order~~
- ~~We cannot access the returned data~~
- ~~Loop never stops~~
- Performance?

Twisted Simple Client (v4)

```python
from twisted.internet import reactor, protocol, defer

import time

from simpleclient2 import UppercaseClientProtocol

def gotData(data, request, starttime):
    print 'request', request, 'took', time.time() - starttime

if __name__ == '__main__':
    import sys
    host, port = sys.argv[1].split(':')
    data_to_send = sys.argv[2:]

    overallstart = time.time()
    all_deferreds = []
    for data in data_to_send:
        print 'sending', data
        d = defer.Deferred()
        d.addCallback(gotData, data, time.time())
        factory = protocol.ClientFactory()
        factory.protocol = UppercaseClientProtocol
        factory.text = data
        factory.deferred = d
        all_deferreds.append(d)
        reactor.connectTCP(host, int(port), factory)

    deferredList = defer.DeferredList(all_deferreds)
    def all_done(results):
        reactor.stop()
    deferredList.addCallback(all_done)


    reactor.run()
    print 'finished, took', time.time() - overallstart
```

simpleclient4.py

Run this!

Simple Client v4 (output)

```
$ python simpleclient4.py 127.0.0.1:8000 [...]
sending http://orestis.gr
sending http://amazon.com
sending http://google.com
sending http://apple.com
request http://google.com took 0.596433877945
request http://amazon.com took 1.22381305695
request http://apple.com took 1.58467292786
request http://orestis.gr took 2.00057697296
finished, took 2.00096821785
```

against the proxy server

```
fetched http://google.com took 0.593424081802
fetched http://amazon.com took 1.22066617012
fetched http://apple.com took 1.57956194878
fetched http://orestis.gr took 1.99814605713
```

Single threaded performance!

As good as threaded performance, without the complexity

Exercise 3

- Write an HTTP GET command-line script

Send:

```
GET <path> HTTP/1.1\r\n
Host: <host>\r\n
User-Agent: <UA-string>\r\n
Connection: close\r\n
\r\n
```

Receive:

```
<data>
```

Usage:

```
$ python httpget.py host:port <path>
```

Exercise 3

```python
from twisted.internet import reactor, protocol, defer

class HTTPGETProtocol(protocol.Protocol):
    def connectionMade(self):
        self.buffer = []
        self.transport.write('GET %s HTTP/1.1\r\n' % self.factory.path)
        self.transport.write('User-Agent: europython/2011\r\n')
        self.transport.write('Host: %s\r\n' % self.factory.host)
        self.transport.write('Connection: close\r\n')
        self.transport.write('\r\n')

    def dataReceived(self, data):
        self.buffer.append(data)


    def connectionLost(self, reason):
        self.factory.deferred.callback(''.join(self.buffer))


def get(address, host, path):
    f = protocol.ClientFactory()
    f.protocol = HTTPGETProtocol
    f.path = path
    f.host = host
    f.deferred = defer.Deferred()
    reactor.connectTCP(address, 80, f)
    return f.deferred
```

httpget.py

A brief recap on writing clients

- Figure out the protocol
- Write the protocol
- Create a factory, set instance variables
- Access the variables from the protocol
- Connect the factory to a host & socket

twisted.web

“the early days”

twisted.web NOP

```
from twisted.web.resource import Resource
from twisted.web.server import Site
from twisted.internet import reactor
from twisted.python import log
import sys
log.startLogging(sys.stdout)


root = Resource()
factory = Site(root)
reactor.listenTCP(8000, factory)
reactor.run()
```

web1.py

Run this!

twisted.web

```python
from twisted.web.resource import Resource
from twisted.web.server import Site
from twisted.internet import reactor
from twisted.python import log
import sys
log.startLogging(sys.stdout)

class Index(Resource):
    def render_GET(self, request):
        return "HELLO"


class Page(Resource):
    def render_GET(self, request):
        return 'A PAGE'


root = Resource()
root.putChild('', Index())
root.putChild('page', Page())
factory = Site(root)
reactor.listenTCP(8000, factory)
reactor.run()
```

web2.py

Run this!

twisted.web persistent

```python
from twisted.web.server import Site, NOT_DONE_YET


class LongRunning(Resource):
    def render_GET(self, request):

        request.write('A')
        reactor.callLater(1, request.write, 'B')
        reactor.callLater(2, request.write, 'C')
        reactor.callLater(3, request.finish)
        return NOT_DONE_YET

root.putChild('long', LongRunning())
```

```
curl -N localhost:8000/long
```

web3.py

twisted.web deferreds

```python
class LongRunning(Resource):
    def render_GET(self, request):
        url = request.args['url'][0]
        d = getPage(url)
        d.addCallback(request.write)
        d.addCallback(lambda _: request.finish())
        return NOT_DONE_YET
```

```
curl -N localhost:8000/long?url=http://...
```

web3b.py

Let’s write a key-value store

...and expose it to over the network

First, the store

```
class KeyValueStore(object):
    def __init__(self):
        self.store = {}

    def get(self, key):
        return self.store[key]

    def set(self, key, value):
        self.store[key] = value
        return key

    def delete(self, key):
        del self.store[key]
```

Too simple

- Let’s make it keep stuff for only 15 seconds

Scheduling

```python
from twisted.internet import reactor

def print_it(p):
    print p

reactor.callLater(5, print_it, 'HI')
reactor.callLater(6, reactor.stop)

reactor.run()
```

sched1.py

Run this!

Scheduling

```python
from twisted.internet import reactor

def print_it(p):
    print p

delayedCall = reactor.callLater(5, print_it, 'HI')
def abort():
    if delayedCall.active():
        print 'CANCELLING'
        delayedCall.cancel()
reactor.callLater(4, abort)
reactor.callLater(6, reactor.stop)

reactor.run()
```

sched2.py

Run this!

A better store

```python
from twisted.internet import reactor

class KeyValueStore(object):
    def __init__(self):
        self.store = {}
        self.timeouts = {}

    def get(self, key):
        return self.store[key]

    def set(self, key, value):
        self._cancelTimeout(key)
        self.store[key] = value
        self.timeouts[key] = reactor.callLater(15, self.delete, key)
        return key

    def delete(self, key):
        self._cancelTimeout(key)
        del self.store[key]

    def _cancelTimeout(self, key):
        delayedCall = self.timeouts.pop(key, None)
        if delayedCall and delayedCall.active():
            delayedCall.cancel()
```

keyvalue.py

How can we expose that?

- Custom protocol
- “Standard” protocol (memcached, other?)
- HTTP (REST, web)
- RPC
- Other?

Custom protocol

- Implement both a server and a client with this spec:

	Send	Receive

get:	`get <key>\r\n`	`VALUE <key> <value>\r\n`
		`GET_NOT_FOUND <key>\r\n`

set:	`set <key> <value>\r\n`	`STORED\r\n`

delete:	`delete <key>\r\n`	`DELETED <key>\r\n`
		`DEL_NOT_FOUND <key>\r\n`

```python
from keyvalue import KeyValueStore

from twisted.internet import reactor, protocol
from twisted.protocols import basic


class KeyValueStoreProtocol(basic.LineReceiver):
    def lineReceived(self, line):
        command, args = line.split()[0], line.split()[1:]
        if command == 'get':
            try:
                value = self.factory.store.get(args[0])
                self.sendLine('VALUE %s %s' % (args[0], value))
            except KeyError:
                self.sendLine('GET_NOT_FOUND %s' % args[0])
        elif command == 'set':
            self.factory.store.set(args[0], args[1])
            self.sendLine('STORED')
        elif command == 'delete':
            try:
                self.factory.store.delete(args[0])
                self.sendLine('DELETED %s' % args[0])
            except KeyError:
                self.sendLine('DEL_NOT_FOUND %s' % args[0])

factory = protocol.ServerFactory()
factory.protocol = KeyValueStoreProtocol
factory.store = KeyValueStore()

reactor.listenTCP(11211, factory)
reactor.run()
```

keyvalue_server.py

```python
from twisted.internet import defer
from twisted.protocols import basic

class KeyValueClientProtocol(basic.LineReceiver):
    def __init__(self):
        self.get_deferreds = []
        self.set_deferreds = []
        self.delete_deferreds = []
    def get(self, key):
        self.sendLine('get %s' % key)
        d = defer.Deferred()
        self.get_deferreds.append(d)
        return d
    def set(self, key, value):
        self.sendLine('set %s %s' % (key, value))
        d = defer.Deferred()
        self.set_deferreds.append(d)
        return d
    def delete(self, key):
        self.sendLine('delete %s' % key)
        d = defer.Deferred()
        self.delete_deferreds.append(d)
        return d
```

```python
    def lineReceived(self, line):
        if line.startswith('VALUE'):
            value = line.split()[-1]
            d = self.get_deferreds.pop(0)
            d.callback(value)
        elif line.startswith('STORED'):
            key = line.split()[-1]
            d = self.set_deferreds.pop(0)
            d.callback(key)
        elif line.startswith('DELETED'):
            key = line.split()[-1]
            d = self.delete_deferreds.pop(0)
            d.callback(key)
        elif line.startswith('DEL_NOT_FOUND'):
            pass #???
        elif line.startswith('GET_NOT_FOUND'):
            pass #???
```

keyvalue_client.py

How to deal with errors?

Asynchronous exception handlers?

Introducing errbacks

- Like callbacks, but for error conditions
- Create a chain using .addErrback
- Called explicitly - d.errback(reason)
- Called implicitly, when a callback function raises

Errback example (v1)

```
from twisted.internet import reactor, defer


def on_success(msg):
    print 'SUCCESS', msg

def on_error(f):
    print 'ERROR', f.getErrorMessage()


d1 = defer.Deferred()
d1.addCallback(on_success)
d1.addErrback(on_error)
reactor.callLater(1, d1.callback, 'NEAT')

d2 = defer.Deferred()
d2.addCallback(on_success)
d2.addErrback(on_error)
reactor.callLater(2, d2.errback, Exception('BUMMER'))


reactor.run()
```

errback.py

Run this!

Errback example (v2)

```
from twisted.internet import reactor, defer


def on_success(msg):
    print 'SUCCESS', msg

def on_error(f):
    print 'ERROR', f.getErrorMessage()


d1 = defer.Deferred()
d1.addCallback(on_success)
d1.addErrback(on_error)
reactor.callLater(1, d1.callback, 'NEAT')
reactor.callLater(2, d1.errback, Exception('BUMMER'))


reactor.run()
```

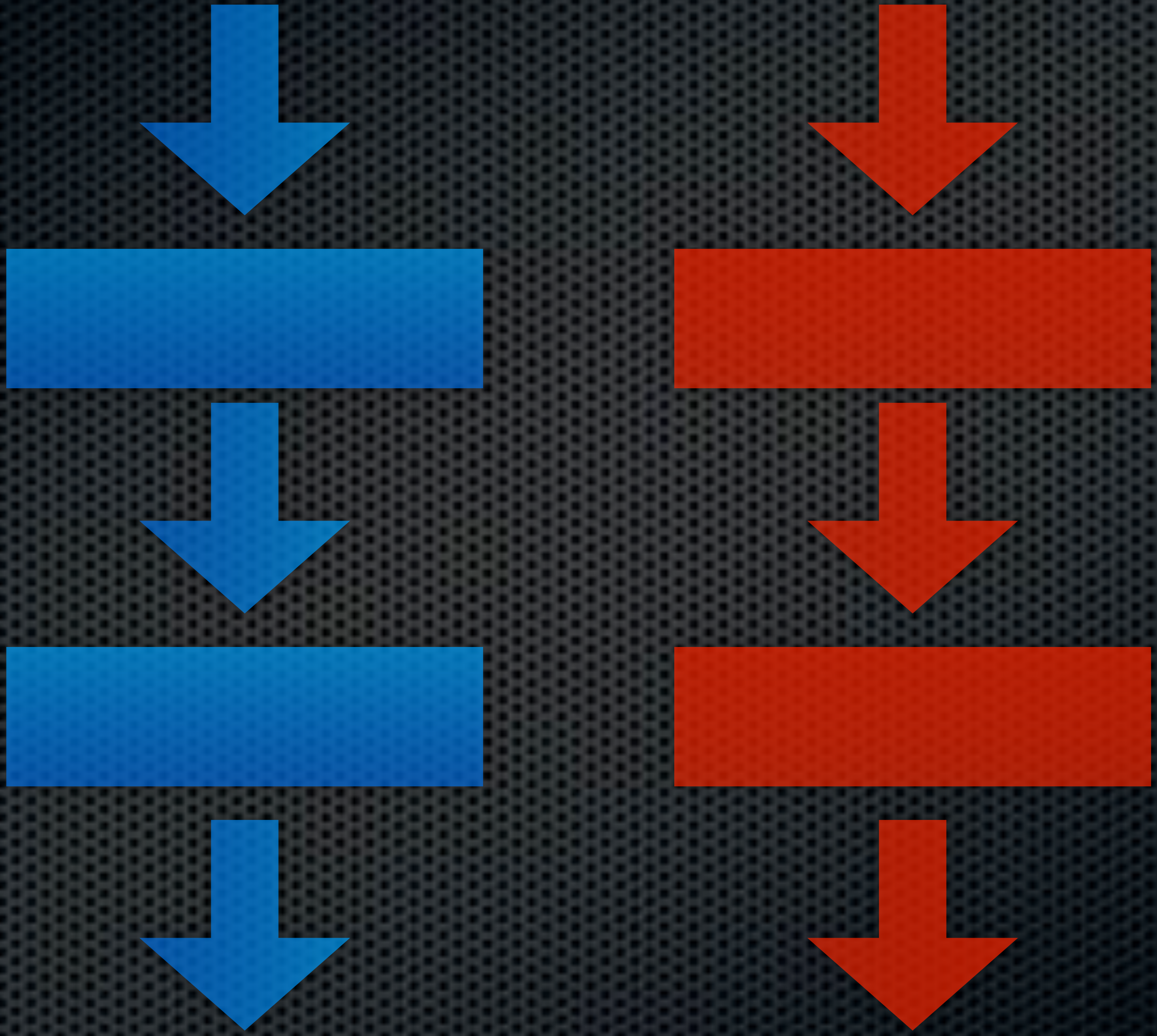
Run this!

errback_wrong.py

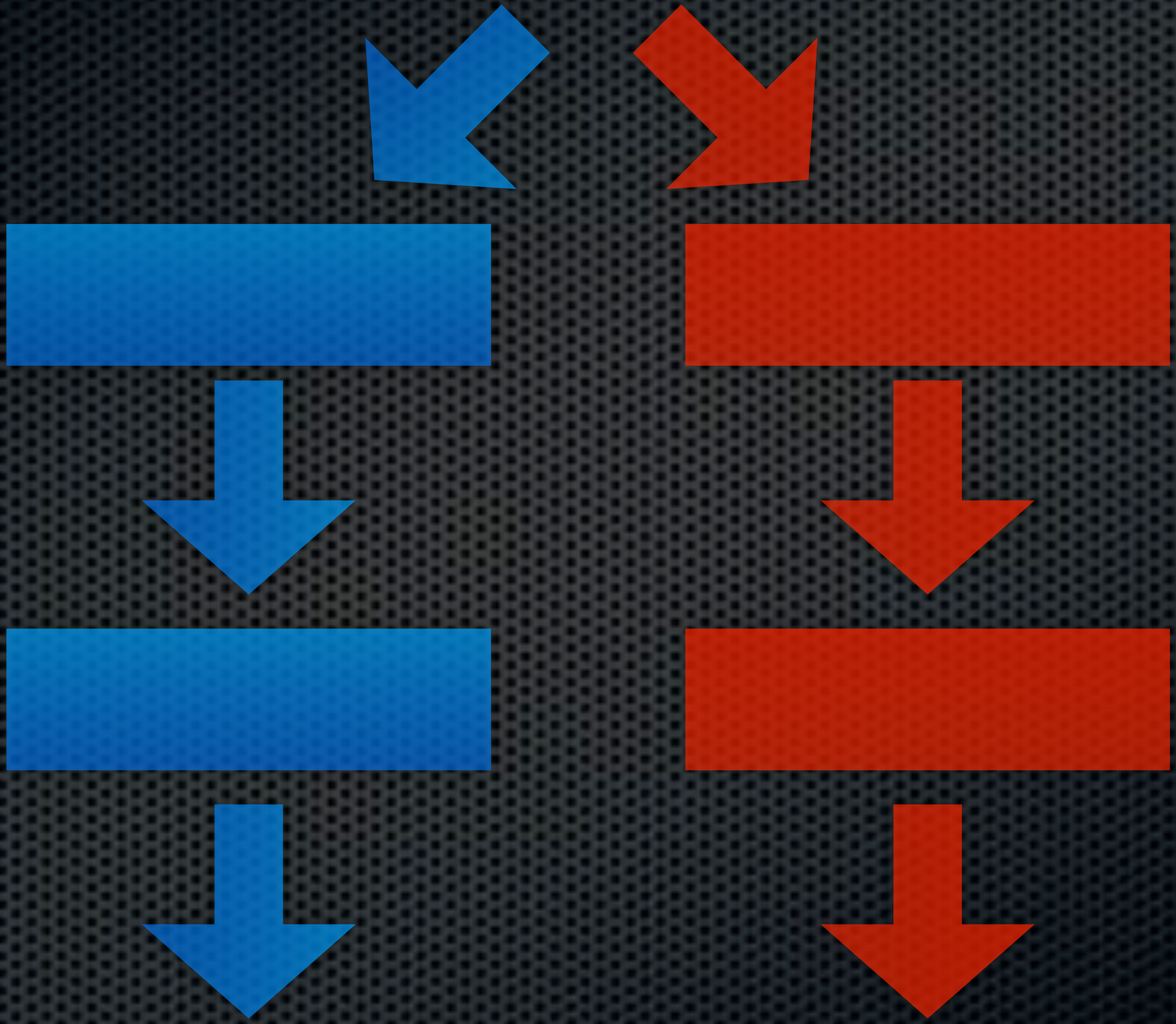
```
defer.setDebugging(True)
```

Deferreds are one-shot

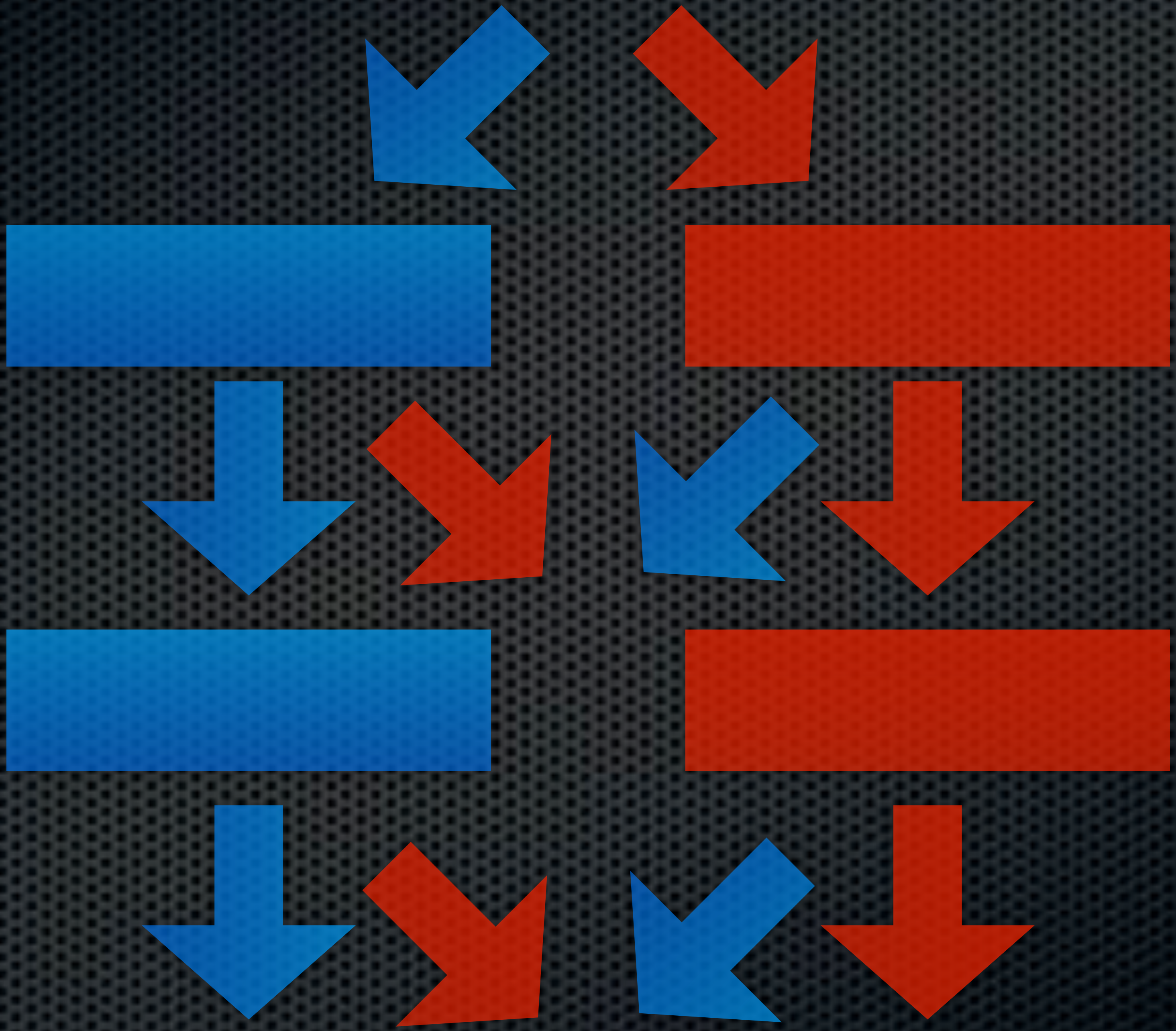
either **one** callback or **one** errback



Calling code

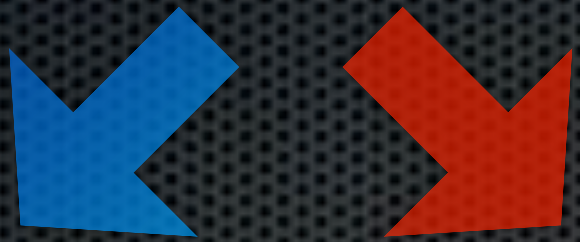


Calling code



Relax, it's easier than it looks

Calling code



Synchronous

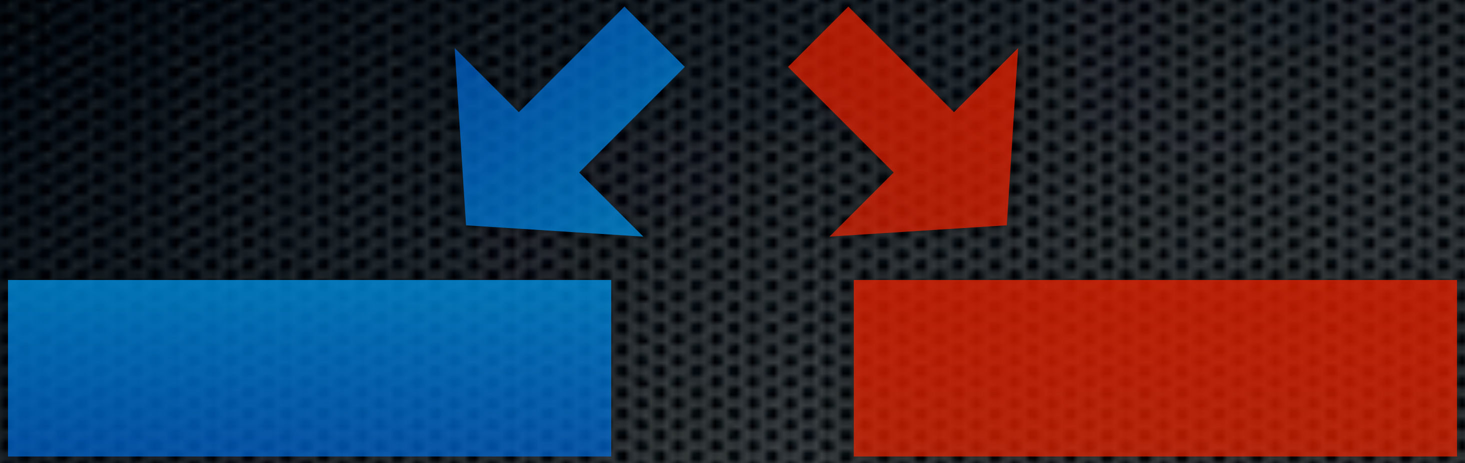
```
def s_function(result):
    if result == "NO":
        raise Exception(result)
    else:
        return result.upper()
```

```
try:
    result = s_function(something)
    print result
except:
    print "OH NOES"
```

Asynchronous

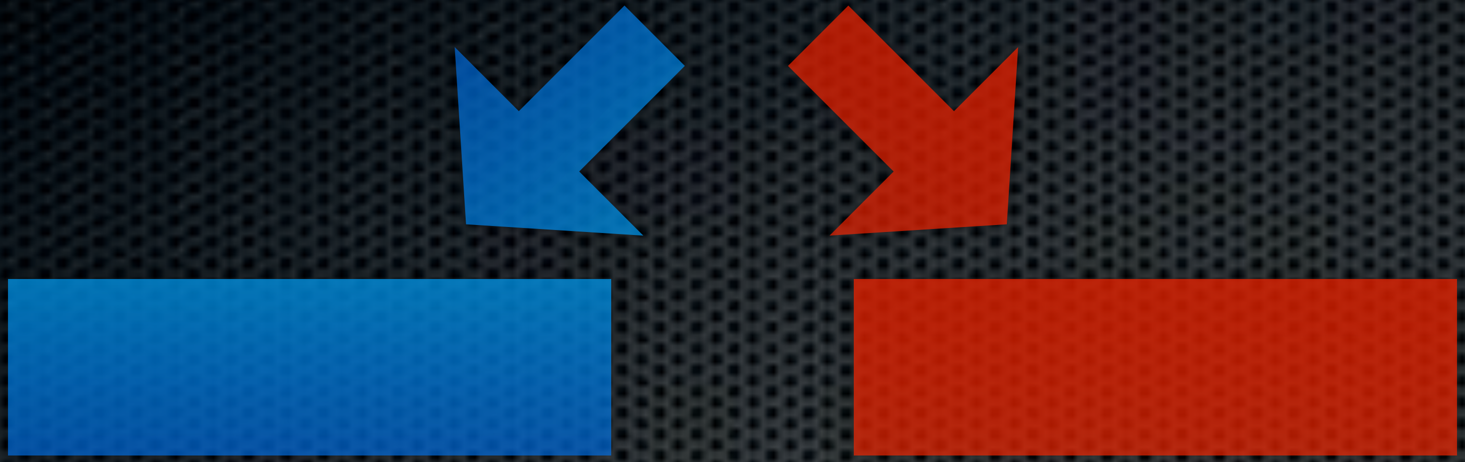
```
def a_function(result, d):
    if result == "NO":
        d.errback(Exception(result))
    else:
        d.callback(result)
```

```
def on_success(r):
    print r
def on_error(_):
    print "OH NOES"
d = defer.Deferred()
d.addCallbacks(on_success, on_error)
a_function(something, d)
```



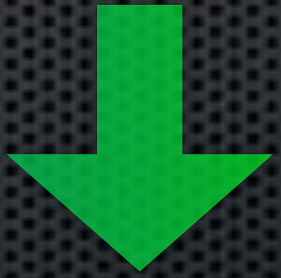
addCallbacks?

```
d = defer.Deferred()

d.addCallback(on_success)
d.addErrback(on_error)

a_function(something, d)
```



```python
def NOP(x):
    return x
```

```python
d.addCallback(on_success)
d.addErrback(on_error)
```



```python
d.addCallbacks(on_success, NOP)
d.addCallbacks(NOP, on_error)
```

Google Doodle Alt Text

- getPage("http://www.google.com")
- On error, print "ERROR: Google down"
- Try to find doodle text, raise Exception if not found
- On error, print "ERROR: No doodle found"
- Finally, print doodle text

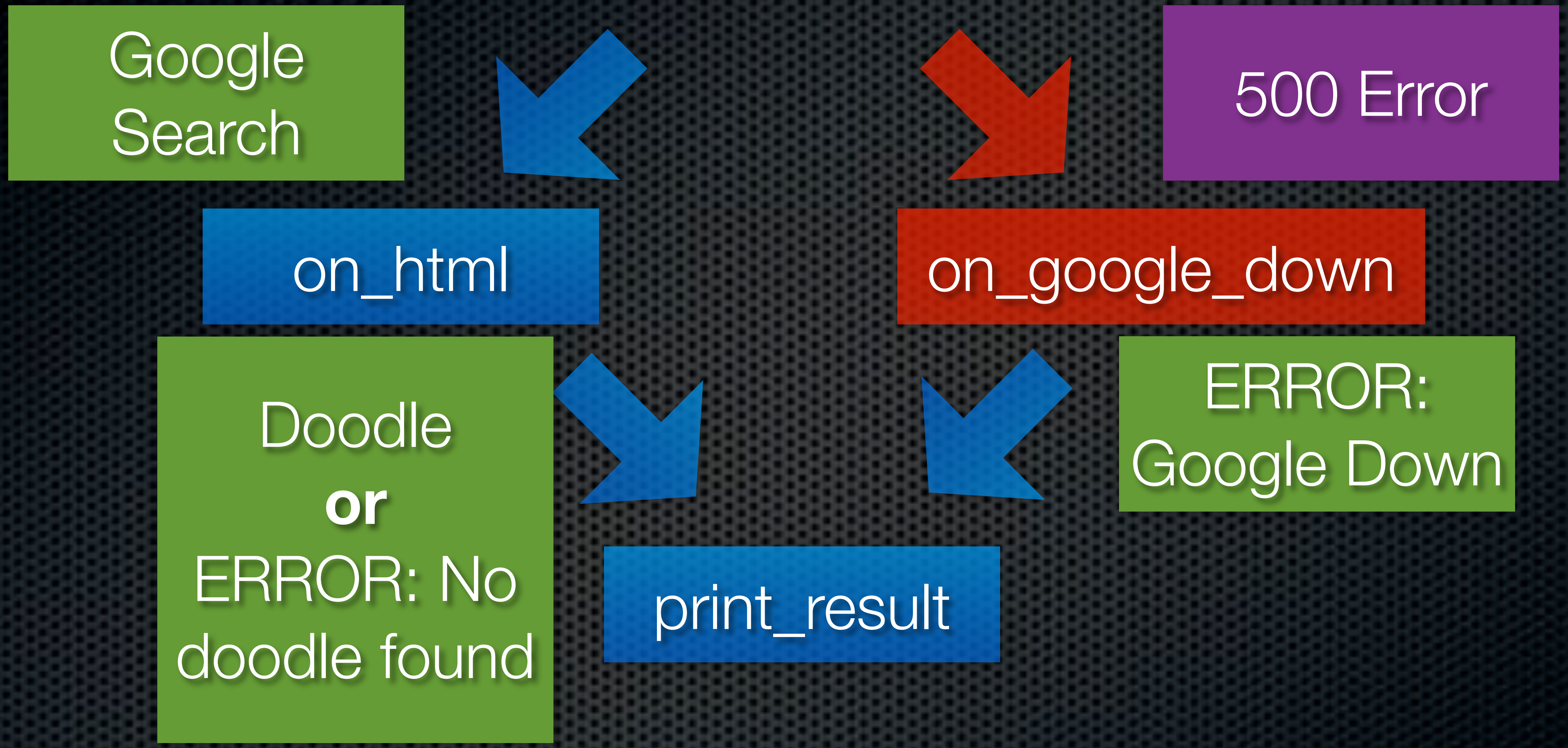
Google Doodle Alt Text

```python
try:
    html = urllib2.openurl('http://www.google.com').read()
    try:
        doodle_text = find_doodle_text(html)
        print doodle_text
    except:
        print "ERROR: No doodle found"
except:
    print "ERROR: Google down"
```

Google Doodle Alt Text

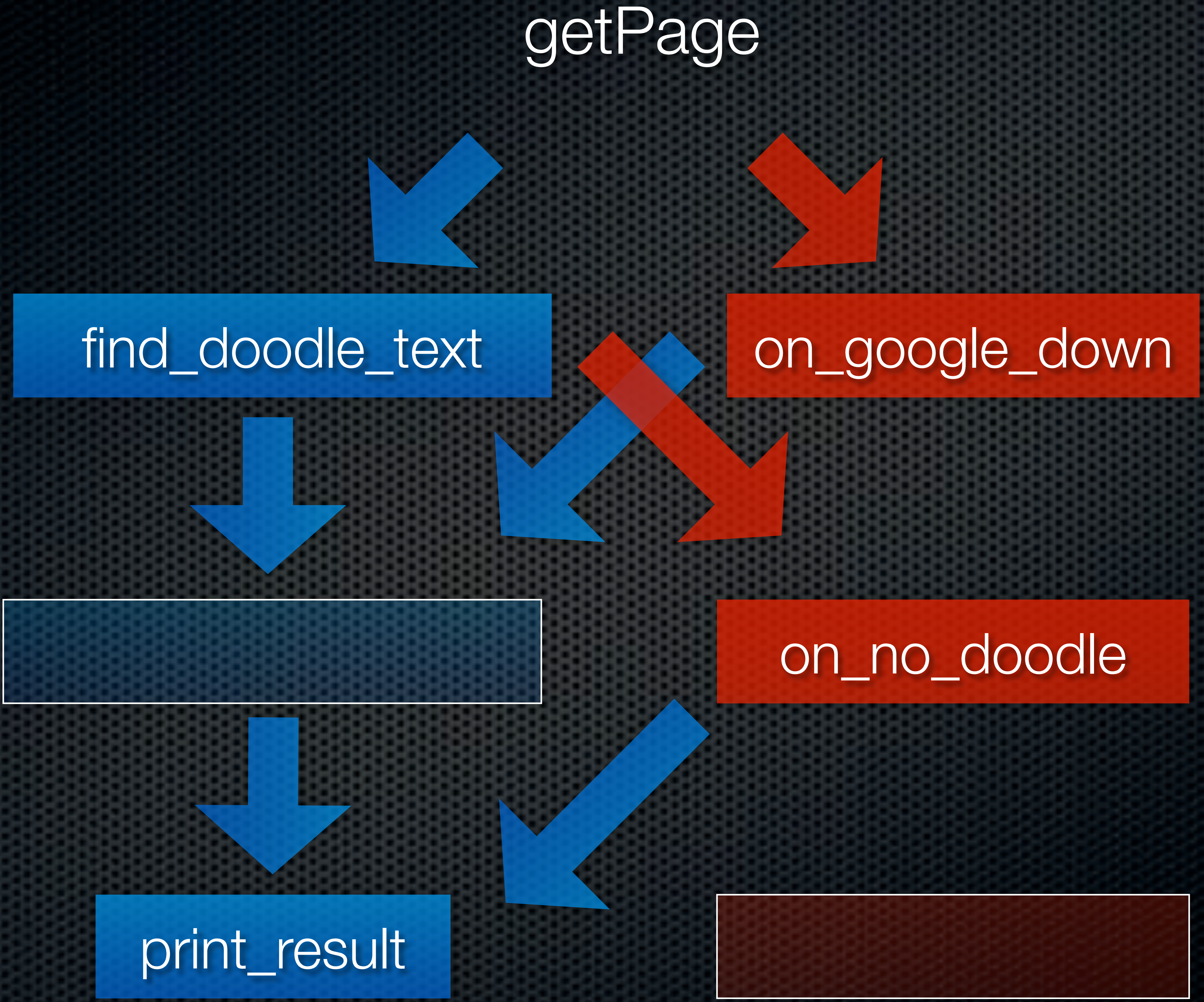
```python
d = getPage('http://www.google.com')

def on_google_down(e):
    return "ERROR: Google down"

def on_page(html):
    try:
        return find_doodle_text(html)
    except:
        return "ERROR: No doodle found"

def print_result(r):
    print r

d.addCallbacks(on_page, on_google_down)
d.addCallback(print_result)
```

getPage

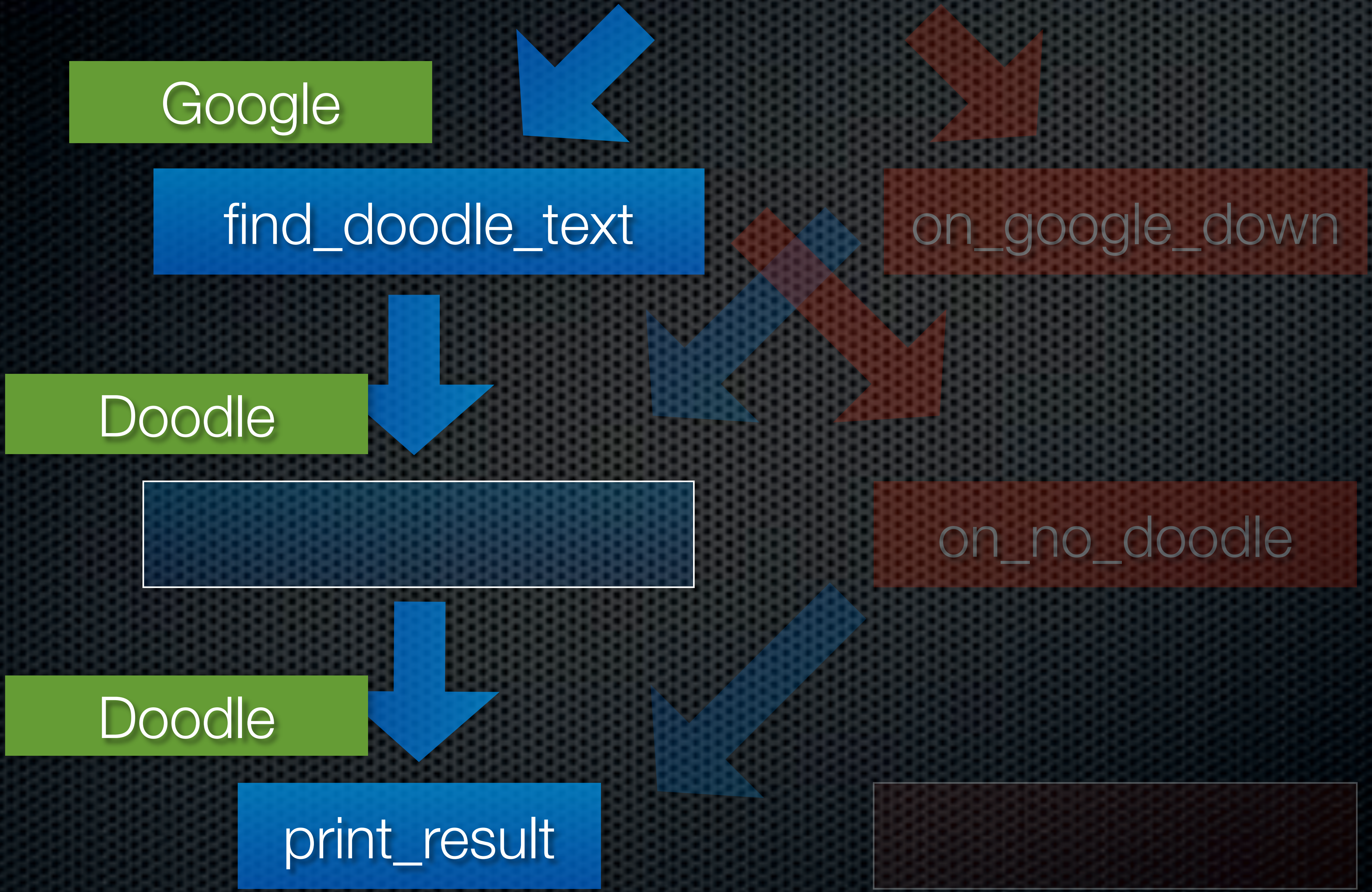

Google Search
500 Error
on_html
on_google_down
Doodle **or** ERROR: No doodle found
ERROR: Google Down
print_result

Google Doodle Alt Text

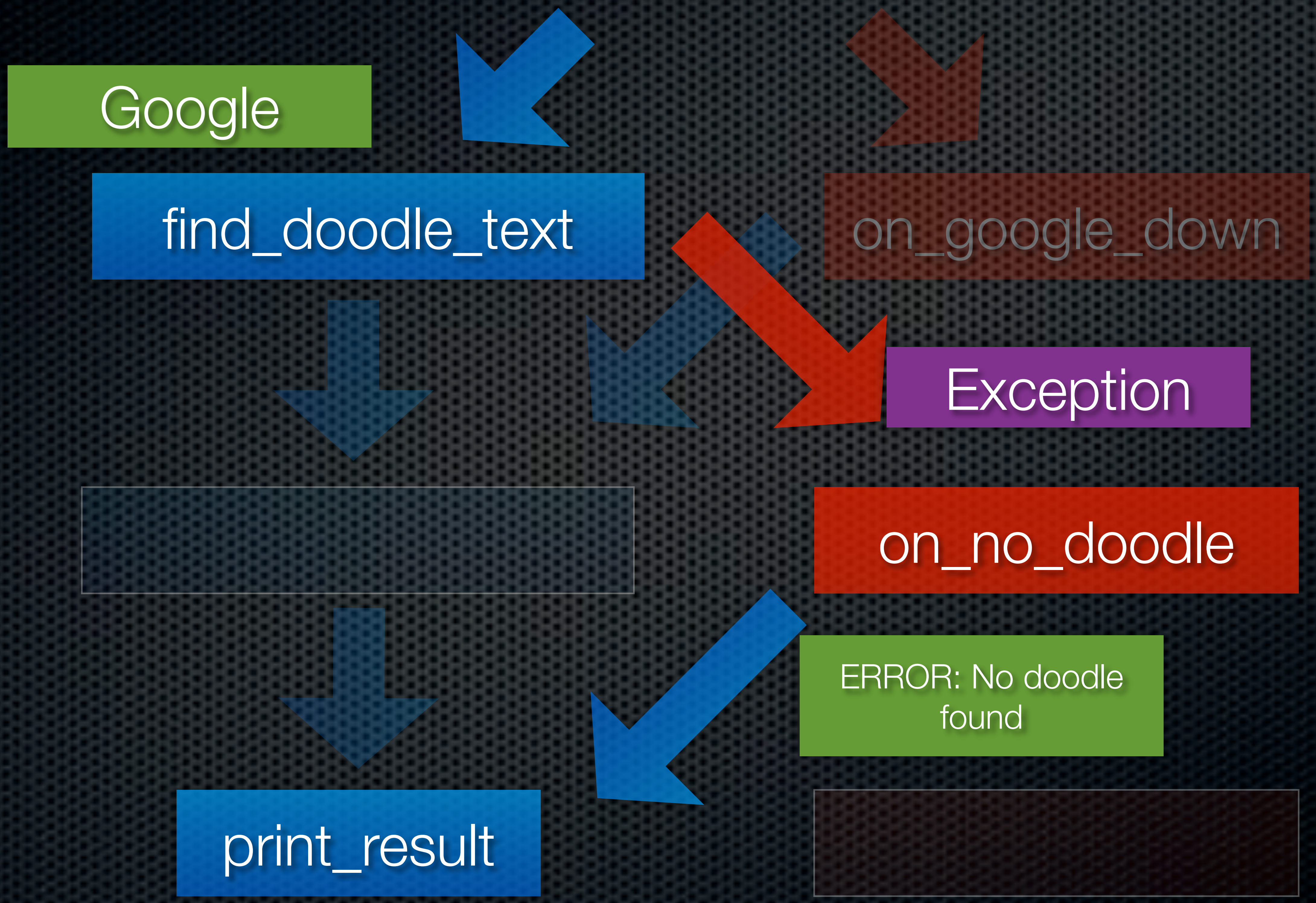
```
d = getPage('http://www.google.com')

def on_google_down(e):
    return "ERROR: Google down"
def on_no_doodle(e):
    return "ERROR: No doodle found"
def print_result(r):
    print r

d.addCallbacks(find_doodle_text,
on_google_down)
d.addErrback(on_no_doodle)
d.addCallback(print_result)
```


getPage
find_doodle_text
on_google_down
on_no_doodle
print_result

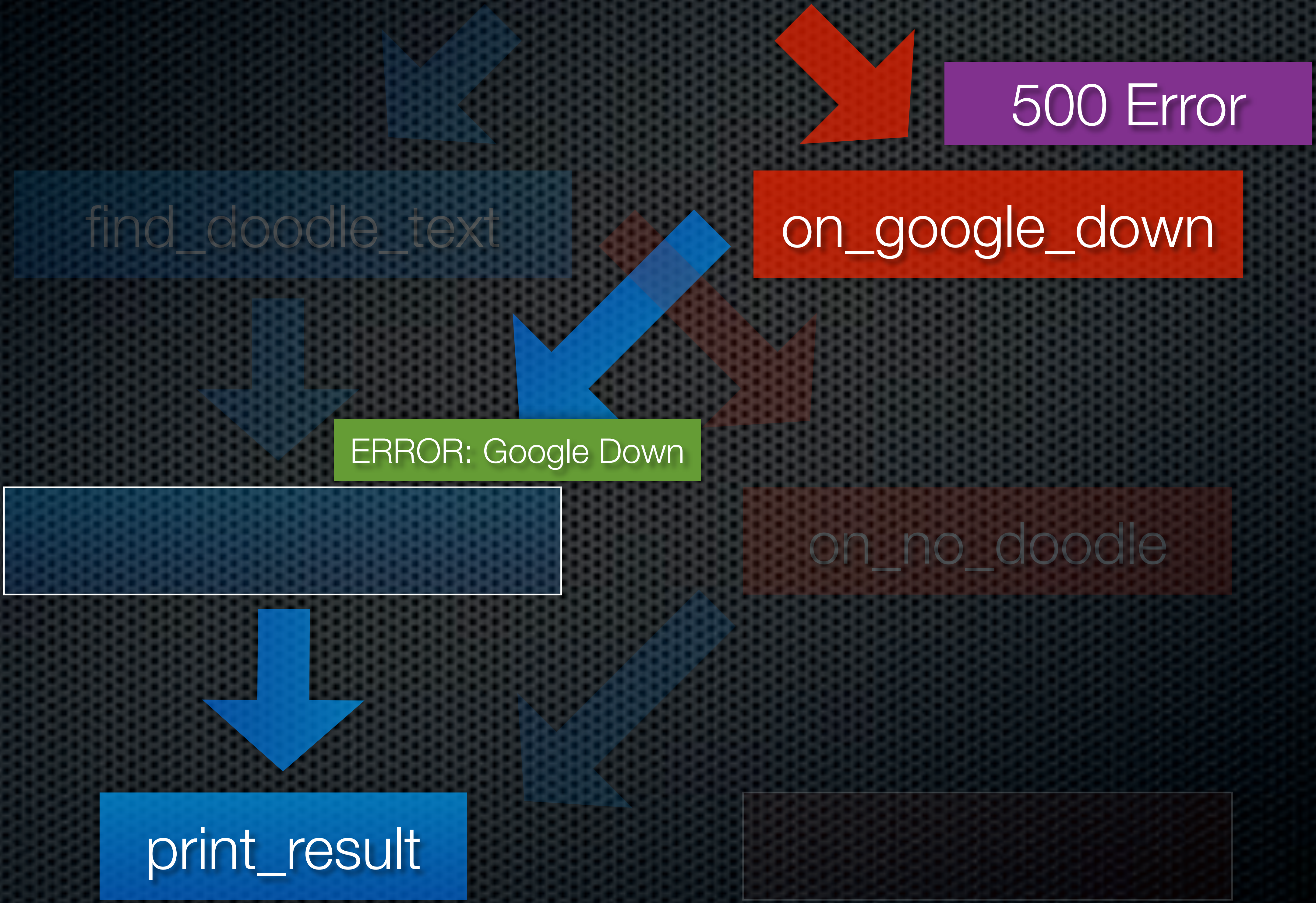
getPage


Google
find_doodle_text
on_google_down
Doodle
on_no_doodle
Doodle
print_result

getPage


Google
find_doodle_text
on_google_down
Exception
on_no_doodle
ERROR: No doodle found
print_result

getPage


500 Error
find_doodle_text
on_google_down
ERROR: Google Down
on_no_doodle
print_result

Why did we start this excursion?

oh right, the Key-Value store client

So, with errbacks:

```
        elif line.startswith('DEL_NOT_FOUND'):
            key = line.split()[-1]
            d = self.delete_deferreds.pop(0)
            d.errback(KeyError(key))
        elif line.startswith('GET_NOT_FOUND'):
            key = line.split()[-1]
            d = self.get_deferreds.pop(0)
            d.errback(KeyError(key))
```

keyvalue_client2.py

Let’s use it

- We’ll write a few web pages that will expose the store to the web
- Run the store server in a different process (or machine)
- Use the store client in the web server

keyvalue_web.py

Index page

```
class Index(Resource):
    def render_GET(self, request):
        return """<html><body>
        <form action="/get">
            <h2>Get</h2>
            Key:<input type="text" name="key">
            <input type="submit">
        </form>
        <form action="/set">
            <h2>Set</h2>
            Key:<input type="text" name="key">
            Value:<input type="text" name="value">
            <input type="submit">
        </form>
        <form action="/delete">
            <h2>Delete</h2>
            Key:<input type="text" name="key">
            <input type="submit">
        </form>
        </body></html>"""
```

Get page

```
class GetPage(Resource):
    def __init__(self, kv):
        Resource.__init__(self)
        self.kv = kv
    def render_GET(self, request):
        key = request.args['key'][0]
        d = self.kv.get(key)
        d.addErrback(lambda f: 'NOT FOUND: %s' % f.getErrorMessage
())
        d.addCallback(request.write)
        d.addCallback(lambda _: request.finish())
        return NOT_DONE_YET
```

Initialization

```
from keyvalue_client2 import KeyValueClientProtocol
def got_protocol(kv):
    root = Resource()
    root.putChild('', Index())
    root.putChild('get', GetPage(kv))
    root.putChild('set', SetPage(kv))
    root.putChild('delete', DeletePage(kv))
    factory = Site(root)
    reactor.listenTCP(8000, factory)

client = protocol.ClientCreator(reactor, KeyValueClientProtocol)
d = client.connectTCP('localhost', 11211)
d.addCallback(got_protocol)

reactor.run()
```

t.i.protocol.ClientCreator

- Change your protocol to have __init__
- Create it with a protocol class and args
- Connect it to a host:port
- Attach a callback
- When the protocol is instantiated, callback is fired

Have a play

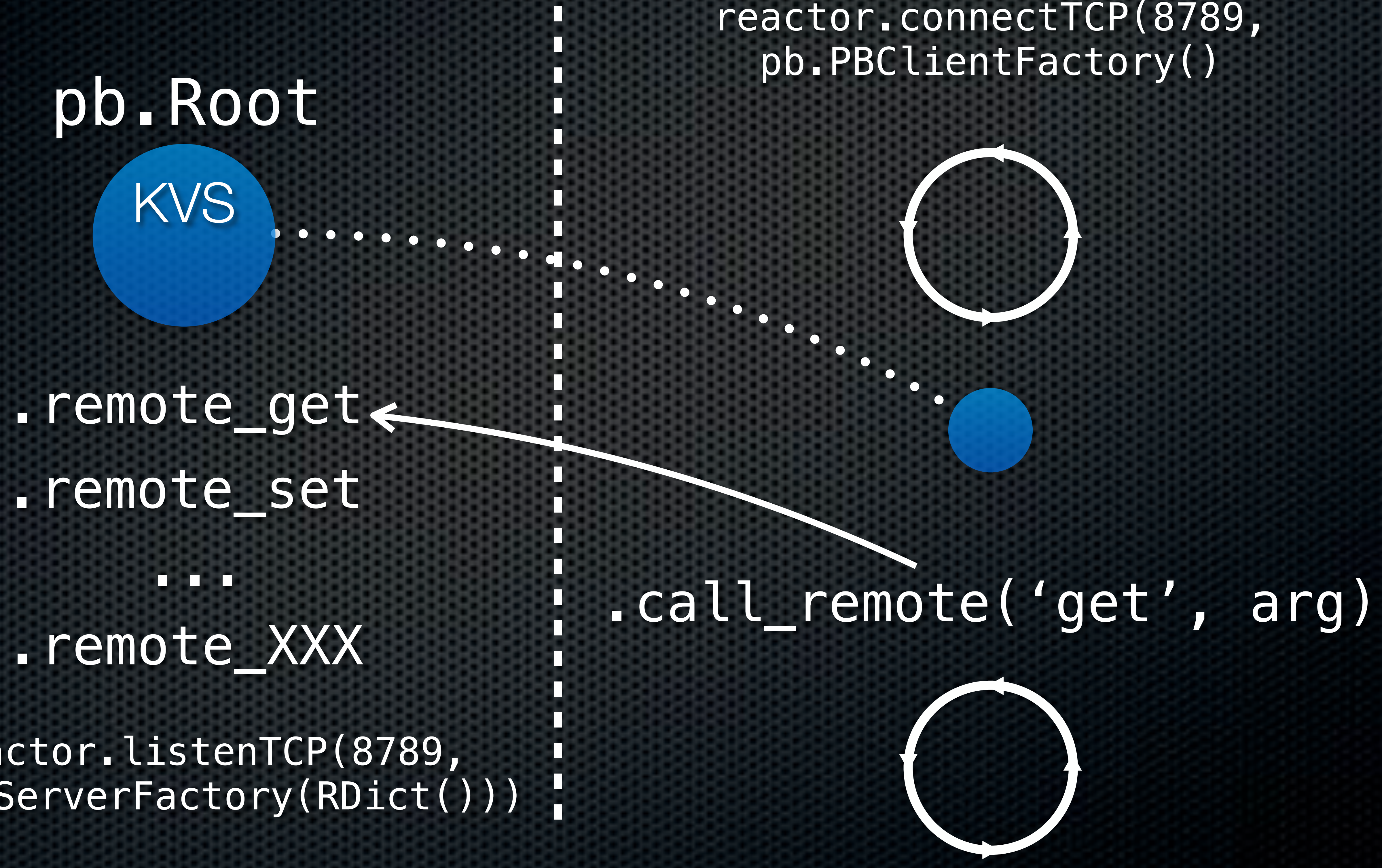
cool, yes?

from twisted.spread import pb

Perspective Broker: Easy and flexible IPC

Also, Foolscap: http://foolscap.lothar.com/trac

Perspective Broker



```
reactor.connectTCP(8789,
  pb.PBClientFactory()
```

```
pb.Root
```

```
.remote_get
.remote_set
    ...
.remote_XXX
```

```
.call_remote('get', arg)
```

```
  reactor.listenTCP(8789,
pb.PBServerFactory(RDict()))
```

PB Server

keyvalue_pb.py

```
from keyvalue import KeyValueStore
from twisted.internet import reactor
from twisted.spread import pb

class KeyValuePB(pb.Root):
    def __init__(self, store):
        self.store = store

    def remote_get(self, key):
        return self.store.get(key)

    def remote_set(self, key, value):
        return self.store.get(key, value)

    def remote_delete(self, key):
        return self.store.delete(key)

if __name__ == '__main__':
    from twisted.python import log
    import sys
    log.startLogging(sys.stdout)
    store = KeyValueStore()
    factory = pb.PBServerFactory(KeyValuePB(store))

    reactor.listenTCP(8789, factory)
    reactor.run()
```

PB Client

keyvalue_web2.py

```
from keyvalue_web import makeSite
from twisted.internet import reactor
from twisted.spread import pb

class RemoteKeyValue(object):
    def __init__(self, rkv):
        self.rkv = rkv

    def get(self, key):
        return self.rkv.callRemote('get', key)

    def set(self, key, value):
        return self.rkv.callRemote('set', key, value)

    def delete(self, key):
        return self.rkv.callRemote('delete', key)


if __name__ == '__main__':
    from twisted.python import log
    import sys
    log.startLogging(sys.stdout)
    factory = pb.PBClientFactory()
    reactor.connectTCP('localhost', 8789, factory)
    d = factory.getRootObject()
    def got_root(rkv):
        kv = RemoteKeyValue(rkv)
        factory = makeSite(kv)
        print 'listening'
        reactor.listenTCP(8000, factory)
    d.addCallback(got_root)
    reactor.run()
```

Multi-server process

```python
from keyvalue import KeyValueStore
from keyvalue_server import KeyValueStoreProtocol
from keyvalue_pb import KeyValuePB
from twisted.spread import pb

from twisted.internet import reactor, protocol

if __name__ == '__main__':
    from twisted.python import log
    import sys
    log.startLogging(sys.stdout)
    store = KeyValueStore()

    factory = protocol.ServerFactory()
    factory.protocol = KeyValueStoreProtocol
    factory.store = store
    reactor.listenTCP(11211, factory)

    pb_factory = pb.PBServerFactory(KeyValuePB(store))
    reactor.listenTCP(8789, pb_factory)

    reactor.run()
```

keyvalue_server2.py

Questions?

Grab-bag of topics

Long-running operations

```python
def fib(target):
    first = 0
    second = 1

    for i in xrange(target - 1):
        new = first + second
        first = second
        second = new
    return second


class Fibonacci(Resource):
    def render_GET(self, request):
        num = int(request.args['num'][0])

        request.write('Result is: ')
        d = defer.Deferred()
        result = fib(num)
        d.callback(result)
        d.addCallback(lambda n: request.write('%s digits long\r\n' % len(str
(n))))
        d.addCallback(lambda _: request.finish())
        return NOT_DONE_YET
```

```
curl -N localhost:8000/?num=100000
```

Long-running operations

```
def fib(target):
    first = 0
    second = 1

    for i in xrange(target - 1):
        new = first + second
        first = second
        second = new
    return second


class Fibonacci(Resource):
    def render_GET(self, request):
        num = int(request.args['num'][0])

        request.write('Result is: ')
        d = defer.Deferred()
        result = fib(num)
        d.callback(result)
        d.addCallback(lambda n: request.write('%s digits long\r\n' % len(str
(n))))
        d.addCallback(lambda _: request.finish())
        return NOT_DONE_YET
```

Long-running operations

```
def fib(target):
    first = 0
    second = 1

    for i in xrange(target - 1):
        new = first + second
        first = second
        second = new
    return second


class Fibonacci(Resource):
    def render_GET(self, request):
        num = int(request.args['num'][0])

        request.write('Result is: ')
        d = defer.Deferred()
        result = fib(num)
        d.callback(result)
        d.addCallback(lambda n: request.write('%s digits long\r\n' % len(str
(n))))
        d.addCallback(lambda _: request.finish())
        return NOT_DONE_YET
```

Forbidden

Reactor is single threaded

- Do not hog the CPU in your callback
- Data will not move in or out
- Scheduled calls will be delayed
- Connections may time out

The callbacks must be **cooperative**

- When accessing the network, **return control back to the loop**
- The loop will call your code when the network is ready
- Must do as little work as possible
- **Doesn't eliminate CPU-bound delays!**

Long-running operations

```
class Fibonacci(Resource):
    def render_GET(self, request):
        num = int(request.args['num'][0])

        request.write('Result is: ')
        d = defer.Deferred()
        d = threads.deferToThread(fib, num)
        d.addCallback(lambda n: request.write('%s digits long\r
\n' % len(str(n))))
        d.addCallback(lambda _: request.finish())

        return NOT_DONE_YET
```

curl -N localhost:8000/?num=100000

Testing Twisted

- Built-in twisted.trial TestRunner with support for Deferreds etc.
- A bit clunky for my taste
- Prefer nose.twistedtools (built-in!)

nose.twistedtools

```
from nose.twistedtools import deferred, reactor


@deferred(timeout=5.0)
def test_async():
    d = getPage(“http://www.google.com”)
    def got_page(p):
        assert 'Google’ in p
    d.addCallback(got_page)
    return d
```

Run this!

Deploying twisted services

“it’s WebScale”

- https://bitbucket.org/jerub/twisted-plugin-example

More cool stuff

- spawnProcess
- manhole
- enterprise.dbapi
- GUI integration
- Third Party libraries (tx____)
- https://launchpad.net/tx

Less cool stuff

- Cruft (constantly improving though!)
- Debugging is harder than async

More resources

- http://twistedmatrix.com
- https://github.com/orestis/EuroPython-2011-Twisted-Training
- https://bitbucket.org/jerub/twisted-plugin-example
- http://krondo.com/?page_id=1327
- http://as.ynchrono.us/ (JP Calderone)
- Slides will be available on the europython.eu site

THE END

Thank you