Architecting an event-driven networking framework: Twisted Python

@jessicamckellar

I want to build

- a web server
- a mail server
- a BitTorrent client
- a DNS server
- an IRC bot
- clients and servers for a custom protocol

in Python

I want them to be

- event-driven
- cross-platform
- RFC-compliant
- testable
- deployable in a standardized fashion
- supporting code-reuse

What should I do?

The TCP/IP model (RFC 1122)

Application Layer

BGP · DHCP · DNS · FTP · Gopher · GTP · HTTP · IMAP · IRC · NNTP · NTP · POP · RIP · RPC · RTCP · RTP · RTSP · SDP · SIP · SMTP · SNMP · SOAP · SSH · STUN · Telnet · TIME · TLS/SSL ·

Transport Layer

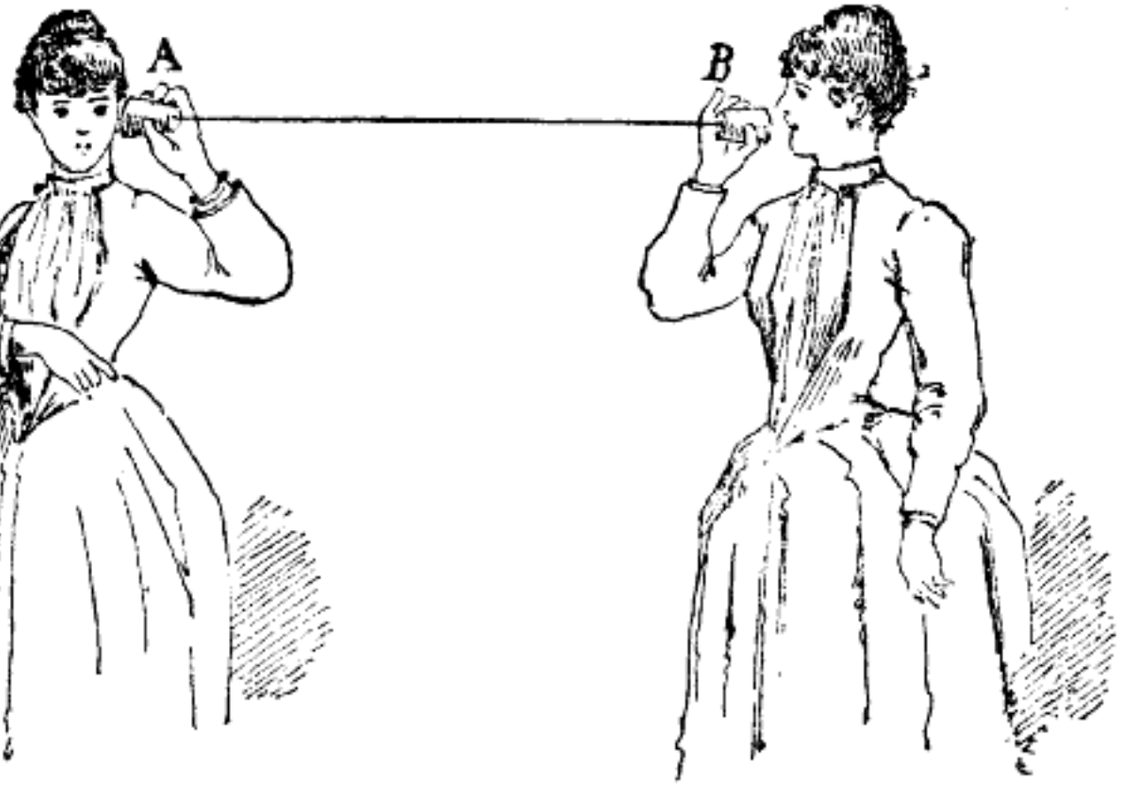
TCP · UDP · DCCP · SCTP · RSVP ·

Internet Layer

IP (IPv4, IPv6) · ICMP · ICMPv6 · IGMP ·

Link Layer

ARP · RARP · NDP · OSPF · Tunnels (L2TP) · Media Access Control (Ethernet, DSL, ISDN, FDDI) · Device Drivers



Event-driven Python

- asyncore: standard library module for writing asynchronous socket service clients and servers. (Stackless)
- gevent: coroutine-based library using greenlets to provide a synchronous API on top of the libevent event loop. (ZeroMQ)
- Tornado: asynchronous web server. (FriendFeed)

asyncore echo server

```
class EchoHandler(asyncore.dispatcher_with_send):
    def handle_read(self):
        data = self.recv(8192)
        if data:
            self.send(data)

class EchoServer(asyncore.dispatcher):
    def __init__(self, host, port):
        asyncore.dispatcher.__init__(self)
        self.create_socket(socket.AF_INET, socket.SOCK_STREAM)
        self.set_reuse_addr()
        self.bind((host, port))
        self.listen(5)

    def handle_accept(self):
        pair = self.accept()
        if pair is not None:
            sock, addr = pair
            handler = EchoHandler(sock)

server = EchoServer('localhost', 8000)
asyncore.loop()
```

Event-driven Python

- asyncore: standard library module for writing asynchronous socket service clients and servers. (Stackless)
- gevent: coroutine-based library using greenlets to provide a synchronous API on top of the libevent event loop. (ZeroMQ)
- Tornado: asynchronous web server. (FriendFeed)

Event-driven Python

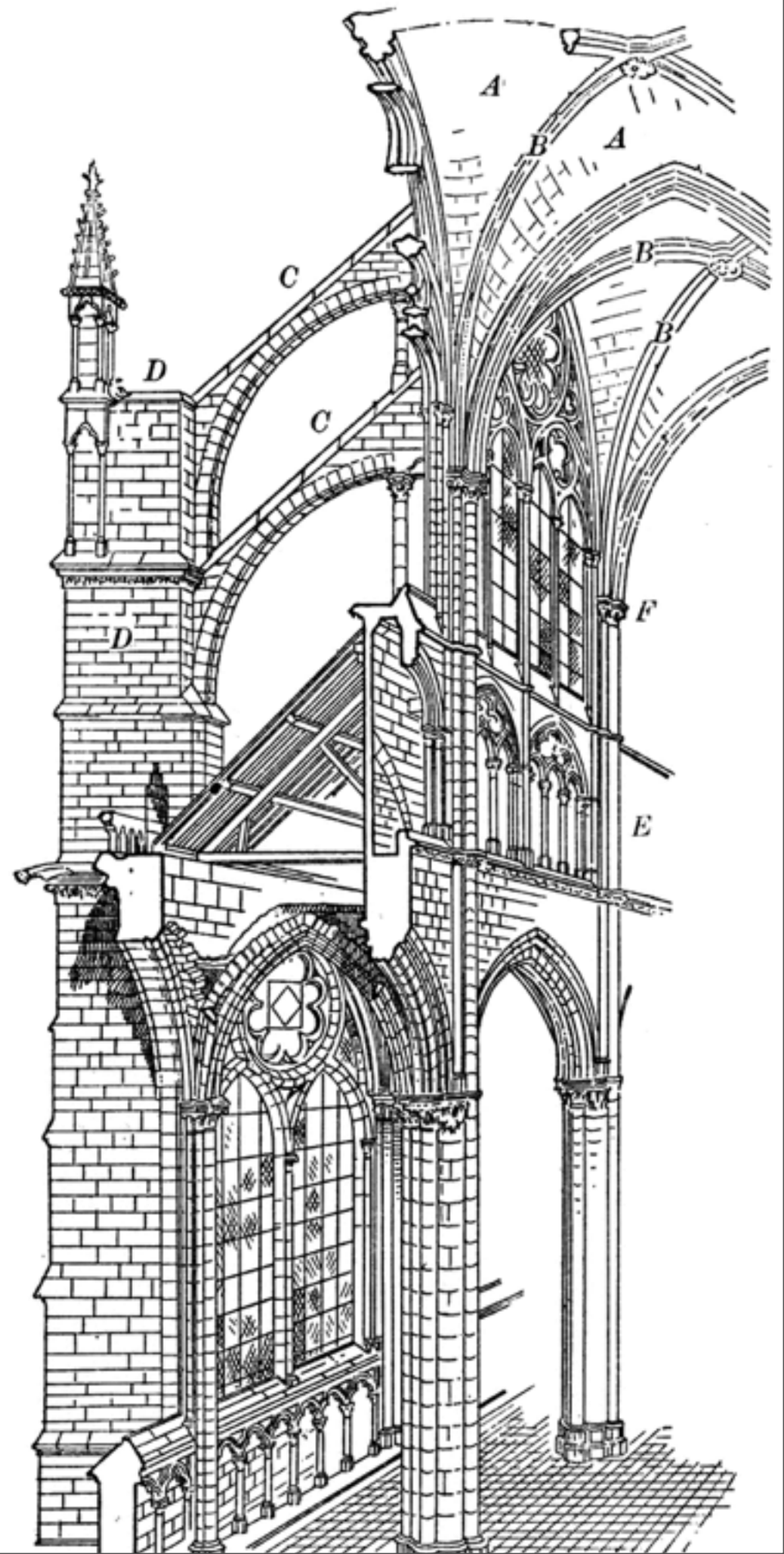
- asyncore: standard library module for writing asynchronous socket service clients and servers. (Stackless)
- gevent: coroutine-based library using greenlets to provide a synchronous API on top of the libevent event loop. (ZeroMQ)
- Tornado: asynchronous web server. (FriendFeed)

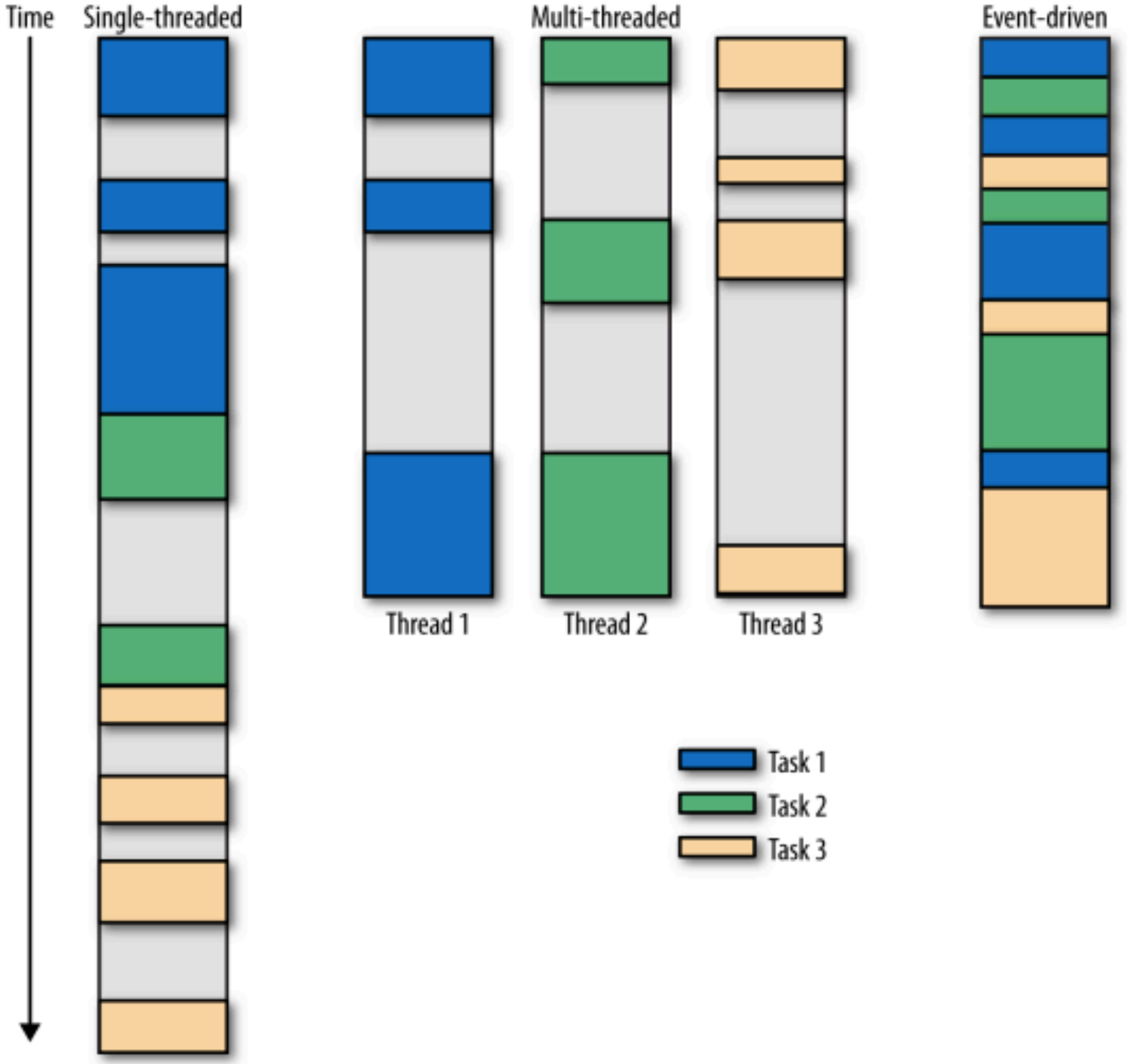
Twisted: an event-driven networking framework

- event-driven
- cross-platform
- RFC-compliant
- “batteries-included”
- general and extensible
- clients and servers

TCP, UDP, SSL,
HTTP, IMAP, IRC,
SMTP, POP3, IMAP,
DNS, FTP...

Architecture




Time
Single-threaded
Multi-threaded
Event-driven
Thread 1
Thread 2
Thread 3
Task 1
Task 2
Task 3

The reactor

```
while True:
    timeout = time_until_next_timed_event()
    events = wait_for_events(timeout)
    events += timed_events_until(now())
    for event in events:
        event.process()
```

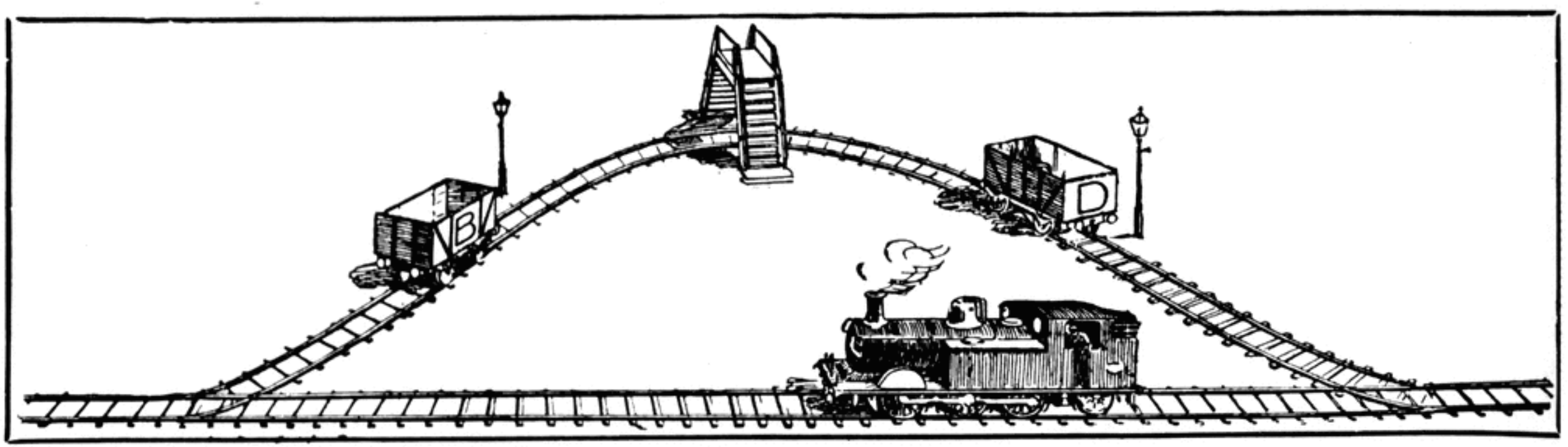
network

filesystem

timer

Transports

Represent the connection between two endpoints communicating over a network



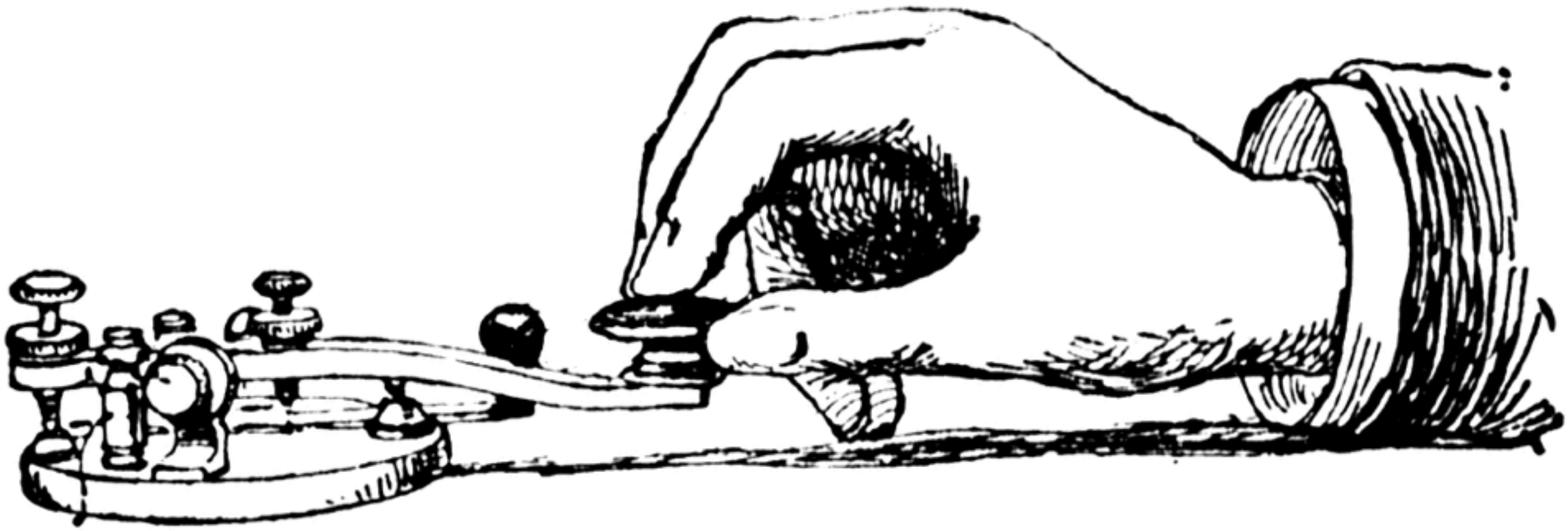
TCP, UDP, SSL, UNIX sockets

Transports

write	Write some data to the physical connection, in sequence, in a non-blocking fashion.
writeSequence	Write a list of strings to the physical connection.
loseConnection	Write all pending data and then close the connection.
getPeer	Get the remote address of this connection.
getHost	Get the address of this side of the connection.

Protocols

Describe how to process network events asynchronously.

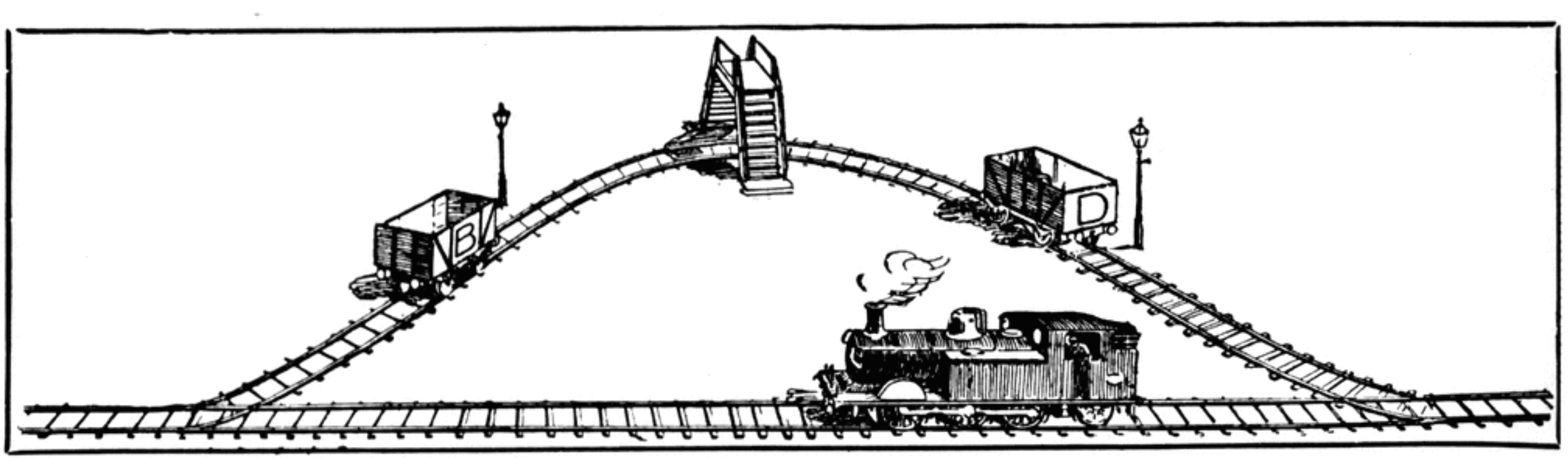


HTTP, IMAP, IRC, DNS

Protocols

makeConnection	Make a connection to a transport and a server.
connectionMade	Called when a connection is made.
dataReceived	Called whenever data is received.
connectionLost	Called when the connection is shut down.

Transports and protocols are decoupled



TCP echo server

```
class Echo(protocol.Protocol):
    def dataReceived(self, data):
        self.transport.write(data)

class EchoFactory(protocol.Factory):
    def buildProtocol(self, addr):
        return Echo()

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

TCP echo server

```
class Echo(protocol.Protocol):
    def dataReceived(self, data):
        self.transport.write(data)

class EchoFactory(protocol.Factory):
    def buildProtocol(self, addr):
        return Echo()

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

TCP echo server

```
class Echo(protocol.Protocol):
    def dataReceived(self, data):
        self.transport.write(data)

class EchoFactory(protocol.Factory):
    def buildProtocol(self, addr):
        return Echo()

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

TCP echo server

```
class Echo(protocol.Protocol):
    def dataReceived(self, data):
        self.transport.write(data)

class EchoFactory(protocol.Factory):
    def buildProtocol(self, addr):
        return Echo()

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

UDP echo server

```
class Echo(protocol.DatagramProtocol):
    def datagramReceived(self, data, addr):
        self.transport.write(data, addr)

reactor.listenUDP(8000, Echo())
reactor.run()
```

UDP echo server

```
class Echo(protocol.DatagramProtocol):
    def datagramReceived(self, data, addr):
        self.transport.write(data, addr)

reactor.listenUDP(8000, Echo())
reactor.run()
```

UDP echo server

```
class Echo(protocol.DatagramProtocol):
    def datagramReceived(self, data, addr):
        self.transport.write(data, addr)
```

```
reactor.listenUDP(8000, Echo())
reactor.run()
```

SSL echo server

```
class Echo(protocol.Protocol):
    def dataReceived(self, data):
        self.transport.write(data)

class EchoFactory(protocol.Factory):
    def buildProtocol(self, addr):
        return Echo()

context = DefaultOpenSSLContextFactory(
    "server.key", "server.crt")
reactor.listenSSL(8000, EchoFactory(),
                  context)
reactor.run()
```

SSL echo server

```
class Echo(protocol.Protocol):
    def dataReceived(self, data):
        self.transport.write(data)

class EchoFactory(protocol.Factory):
    def buildProtocol(self, addr):
        return Echo()

context = DefaultOpenSSLContextFactory(
    "server.key", "server.crt")
reactor.listenSSL(8000, EchoFactory(),
                  context)
reactor.run()
```

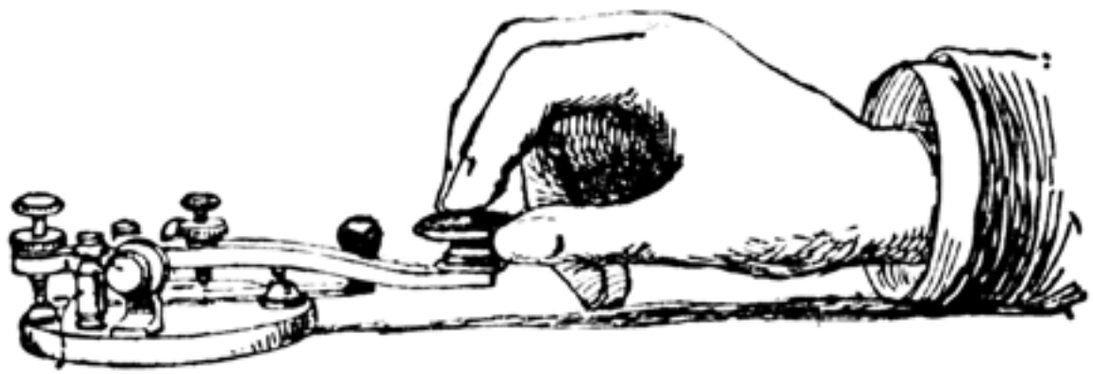
SSL echo server

```
class Echo(protocol.Protocol):
    def dataReceived(self, data):
        self.transport.write(data)

class EchoFactory(protocol.Factory):
    def buildProtocol(self, addr):
        return Echo()

context = DefaultOpenSSLContextFactory(
    "server.key", "server.crt")
reactor.listenSSL(8000, EchoFactory(),
                  context)
reactor.run()
```

Architecture

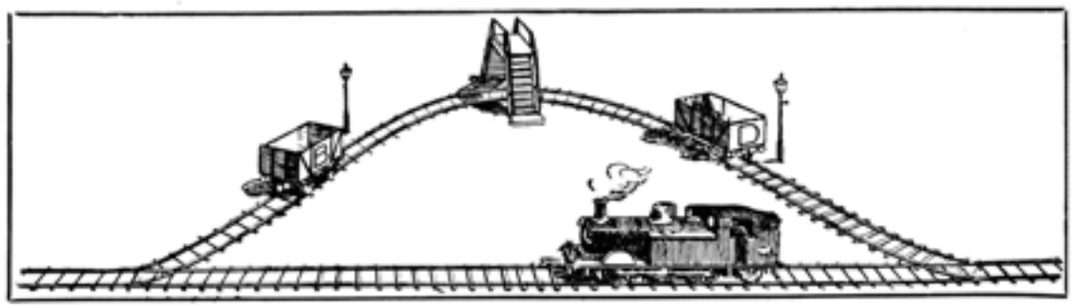
reactor

transports



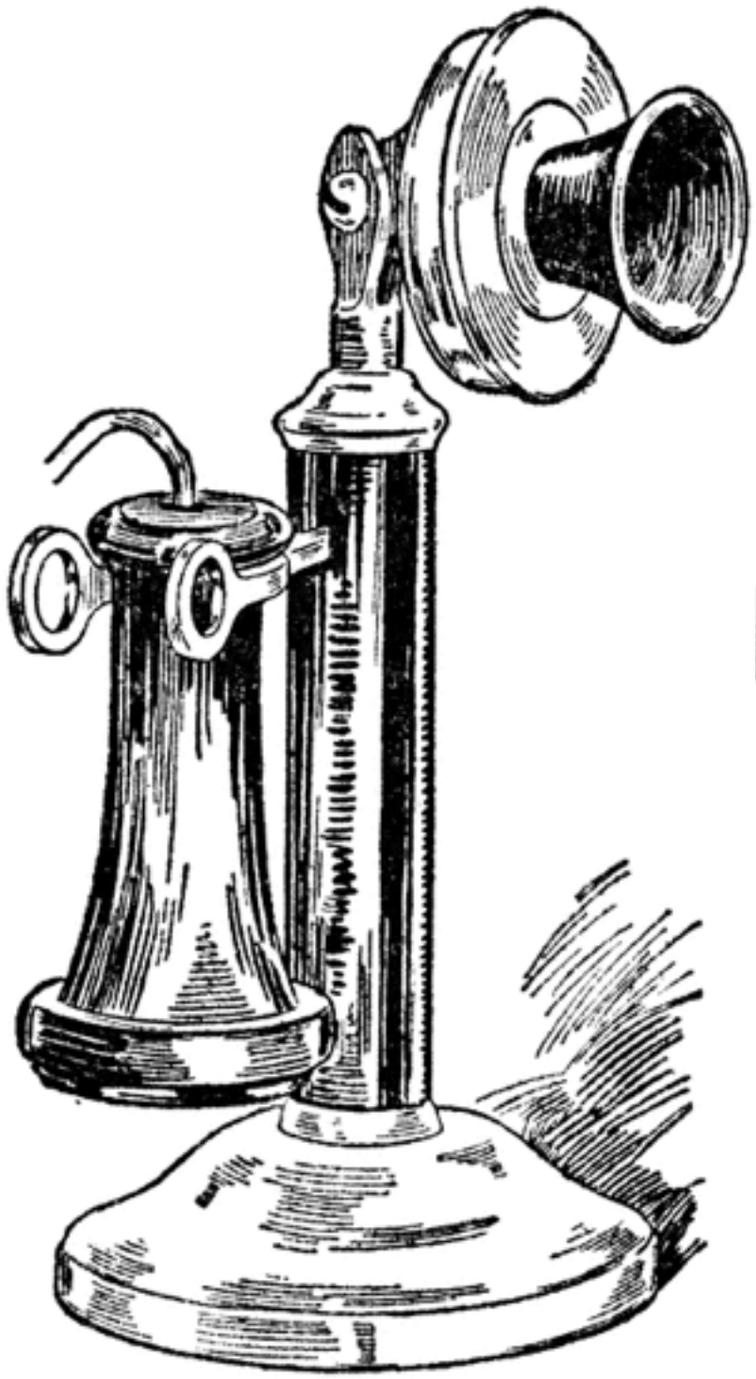
TCP, UDP, SSL,
UNIX sockets

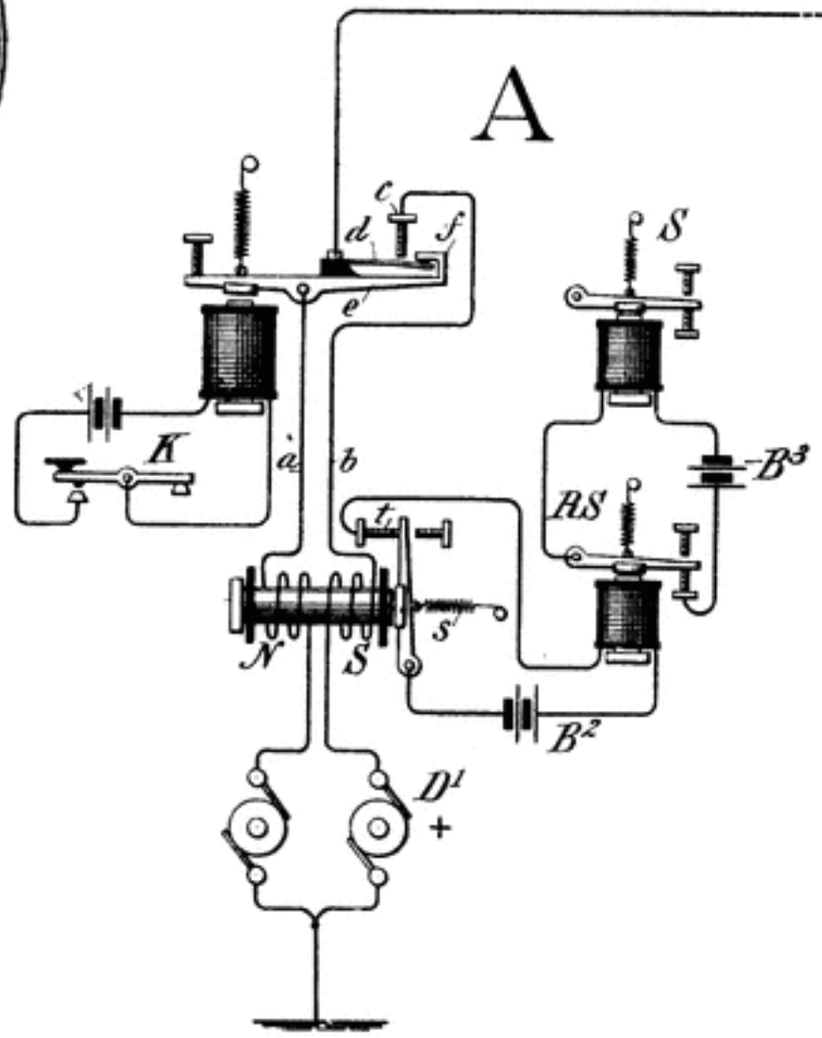
protocols

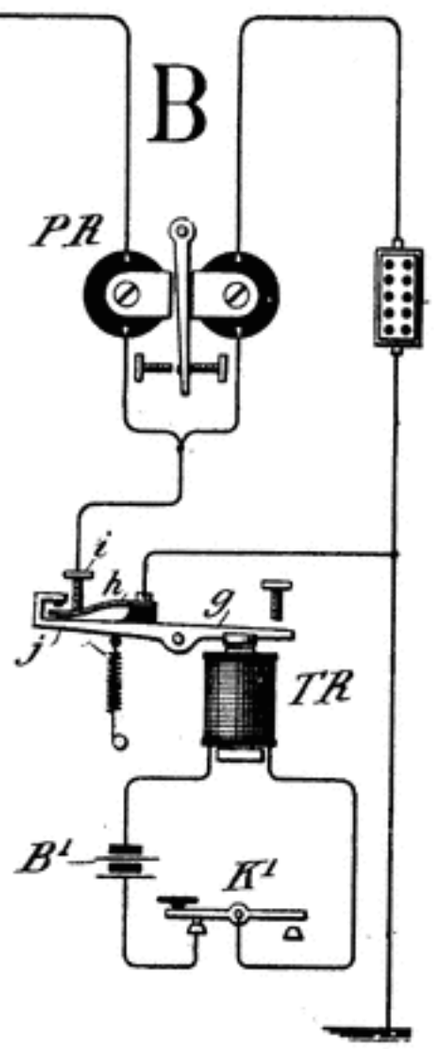


HTTP, IMAP,
IRC, DNS

Managing callbacks

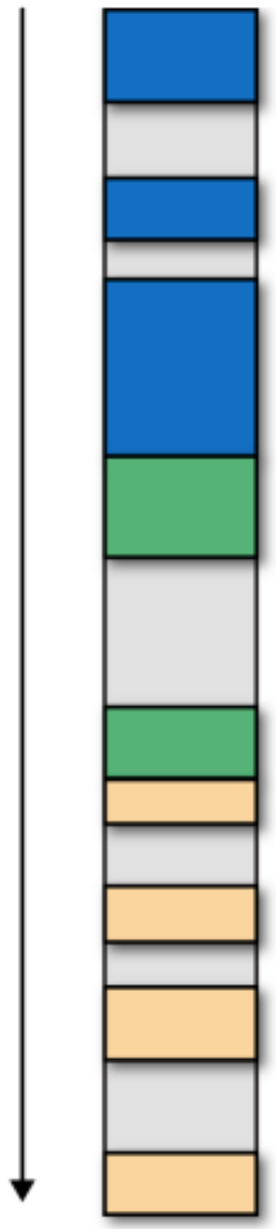






Synchronous

```
import getPage

def processPage(page):
    print page

def logError(error):
    print error

def finishProcessing():
    print "Shutting down..."
    exit(0)

url = "http://google.com"
try:
    page = getPage(url)
    processPage(page)
except Error, e:
    logError(e)
finally:
    finishProcessing()
```



```
import reactor
import getPage

def processPage(page):
    print page
    finishProcessing()

def logError(error):
    print error
    finishProcessing()

def finishProcessing():
    print "Shutting down..."
    reactor.stop()

url = "http://google.com"
getPage(url, processPage, logError)

reactor.run()
```

Event-driven,
with callbacks

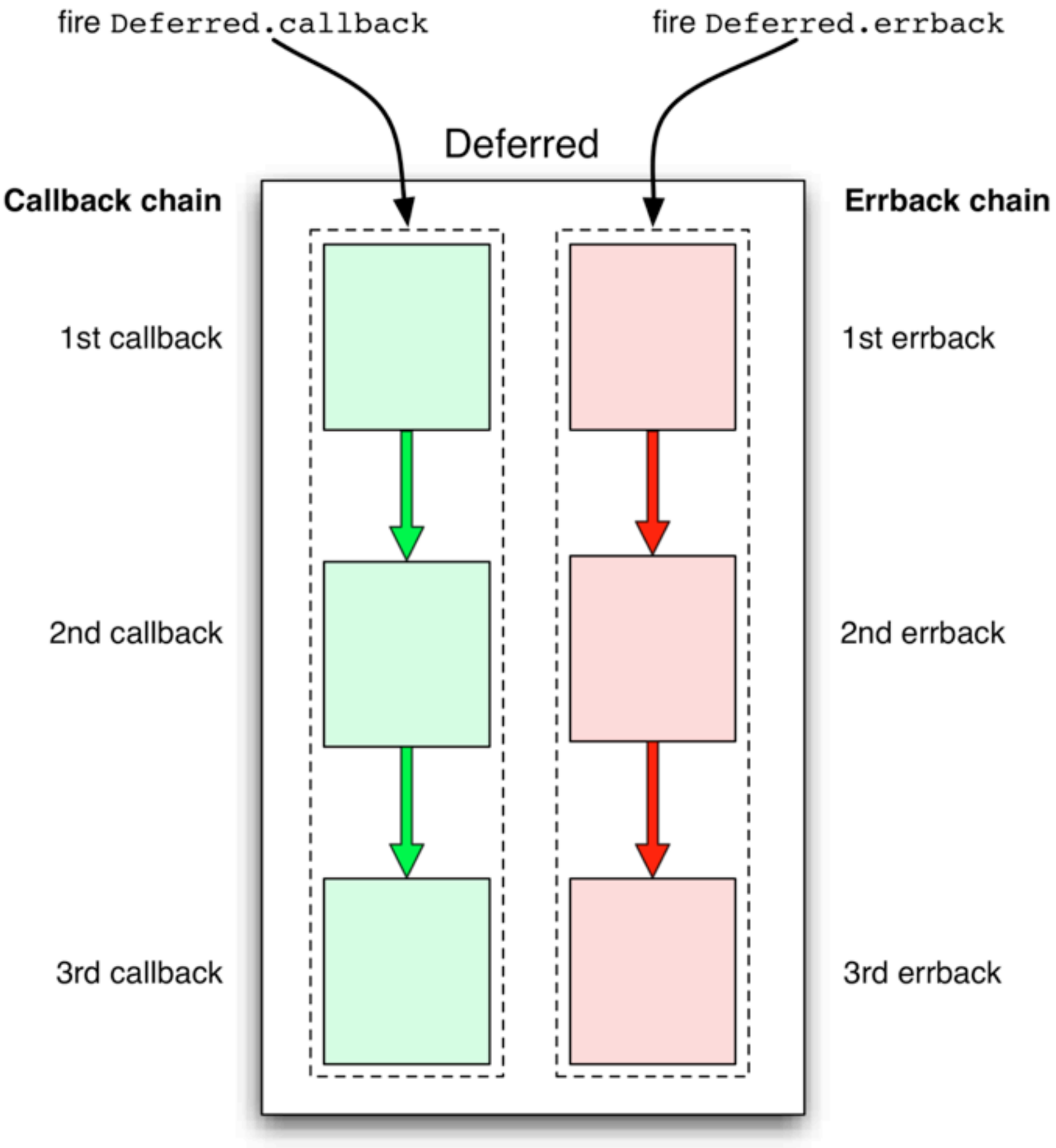
Deferred

An abstraction of the idea of a result that doesn't exist yet; a promise that a function will have a result at some point.

A `Deferred` helps manage the callback chains for processing an asynchronous result.

Event-driven, with Deferreds

```
import reactor
import getPage

def processPage(page):
    print page

def logError(error):
    print error

def finishProcessing(value):
    print "Shutting down..."
    reactor.stop()

url = "http://google.com"
deferred = getPage(url)
deferred.addCallbacks(processPage, logError)
deferred.addBoth(finishProcessing)

reactor.run()
```

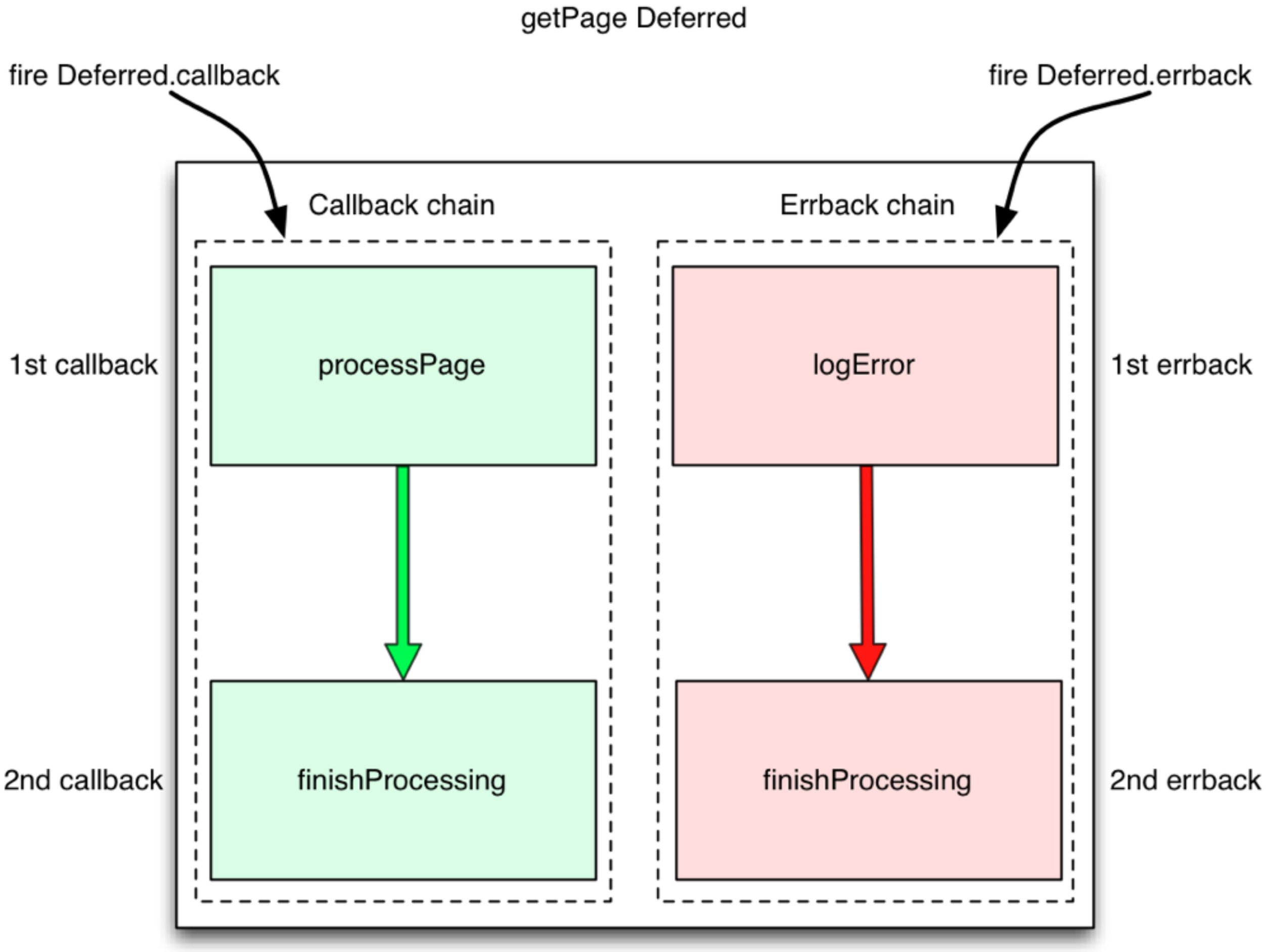

fire Deferred.callback
fire Deferred.errback
Deferred
Callback chain
Errback chain
1st callback
1st errback
2nd callback
2nd errback
3rd callback
3rd errback

Deferred API

addCallback	Register a callback with the callback chain.
addErrback	Register an errback with the errback chain. The analogous synchronous logic is the `except` part of a `try/except` block.
addCallbacks	Add a callback and errback parallel to each other in the callback chains.
addBoth	Add the same callback to both the callback and errback chains. The analogous synchronous logic is the `finally` part of a `try/except/finally` block.

Event-driven, with Deferreds

```
import reactor
import getPage

def processPage(page):
    print page

def logError(error):
    print error

def finishProcessing(value):
    print "Shutting down..."
    reactor.stop()

url = "http://google.com"
deferred = getPage(url)
deferred.addCallbacks(processPage, logError)
deferred.addBoth(finishProcessing)

reactor.run()
```

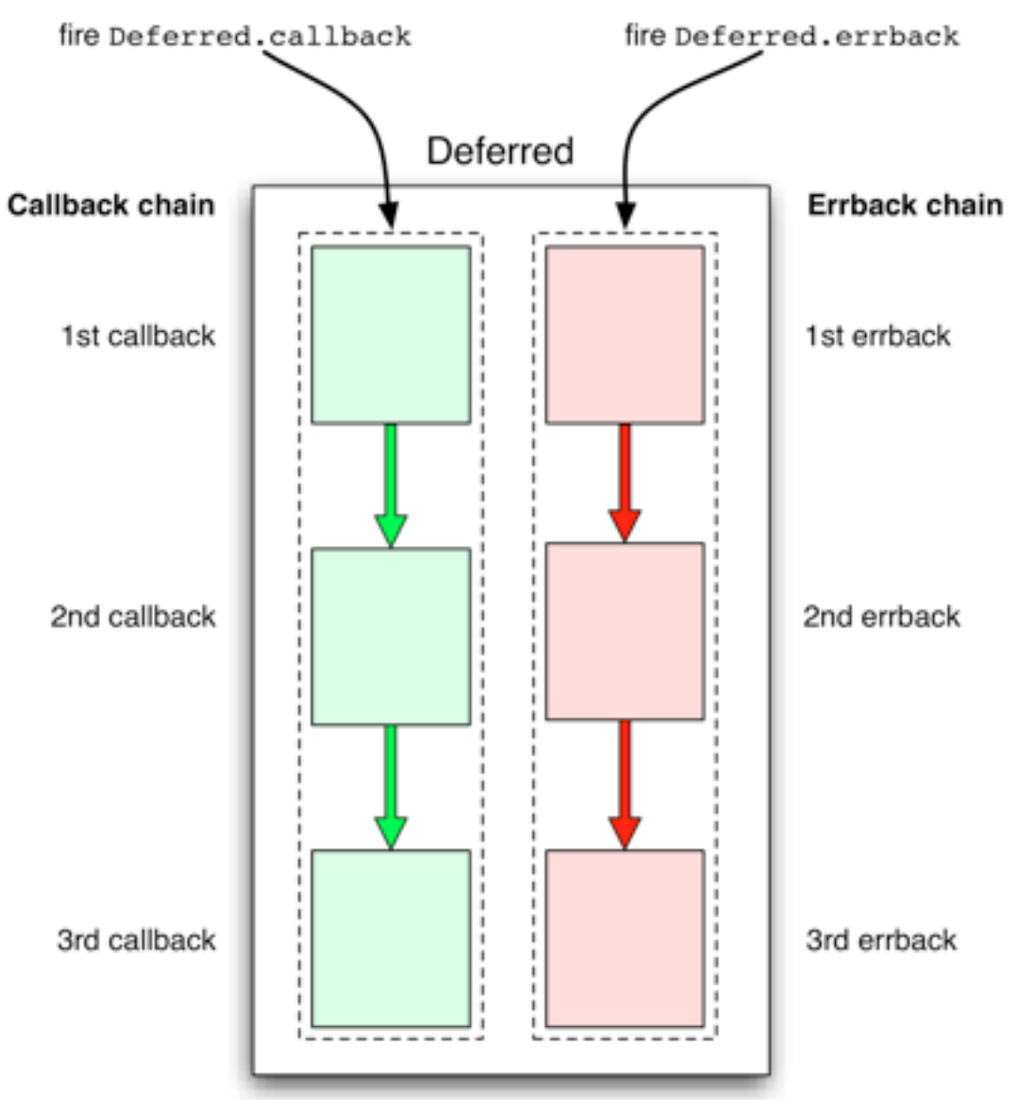

getPage Deferred
fire Deferred.callback
fire Deferred.errback
Callback chain
Errback chain
1st callback
processPage
logError
1st errback
2nd callback
finishProcessing
finishProcessing
2nd errback

Deferreds

- Can only be fired once
- Can be paused/unpaused/cancelled

try	addCallback
except	addErrback
finally	addBoth

} addCallbacks



addCallback

addErrback

addCallbacks

addBoth

MochiKit

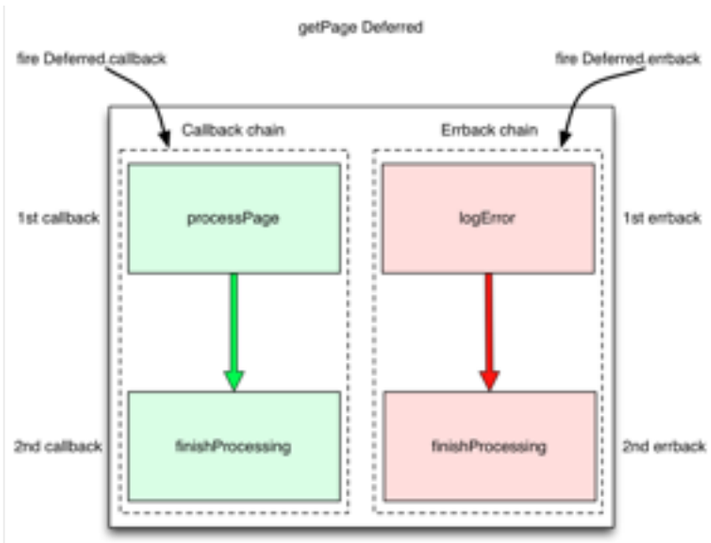
Dojo

jQuery

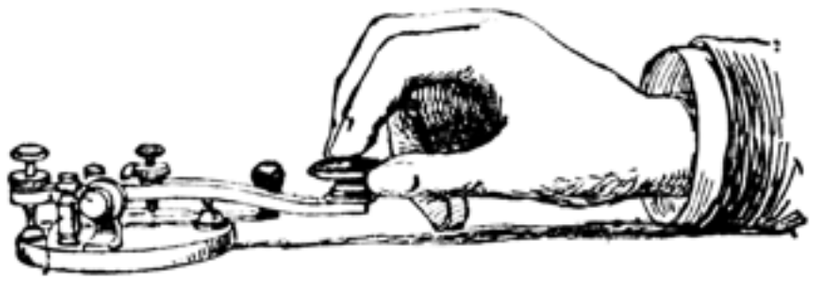
Architecture

reactor

Deferreds

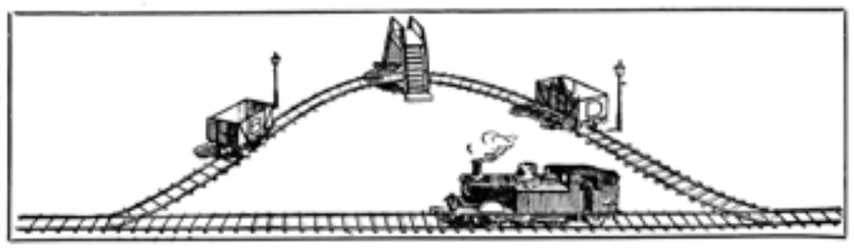


transports



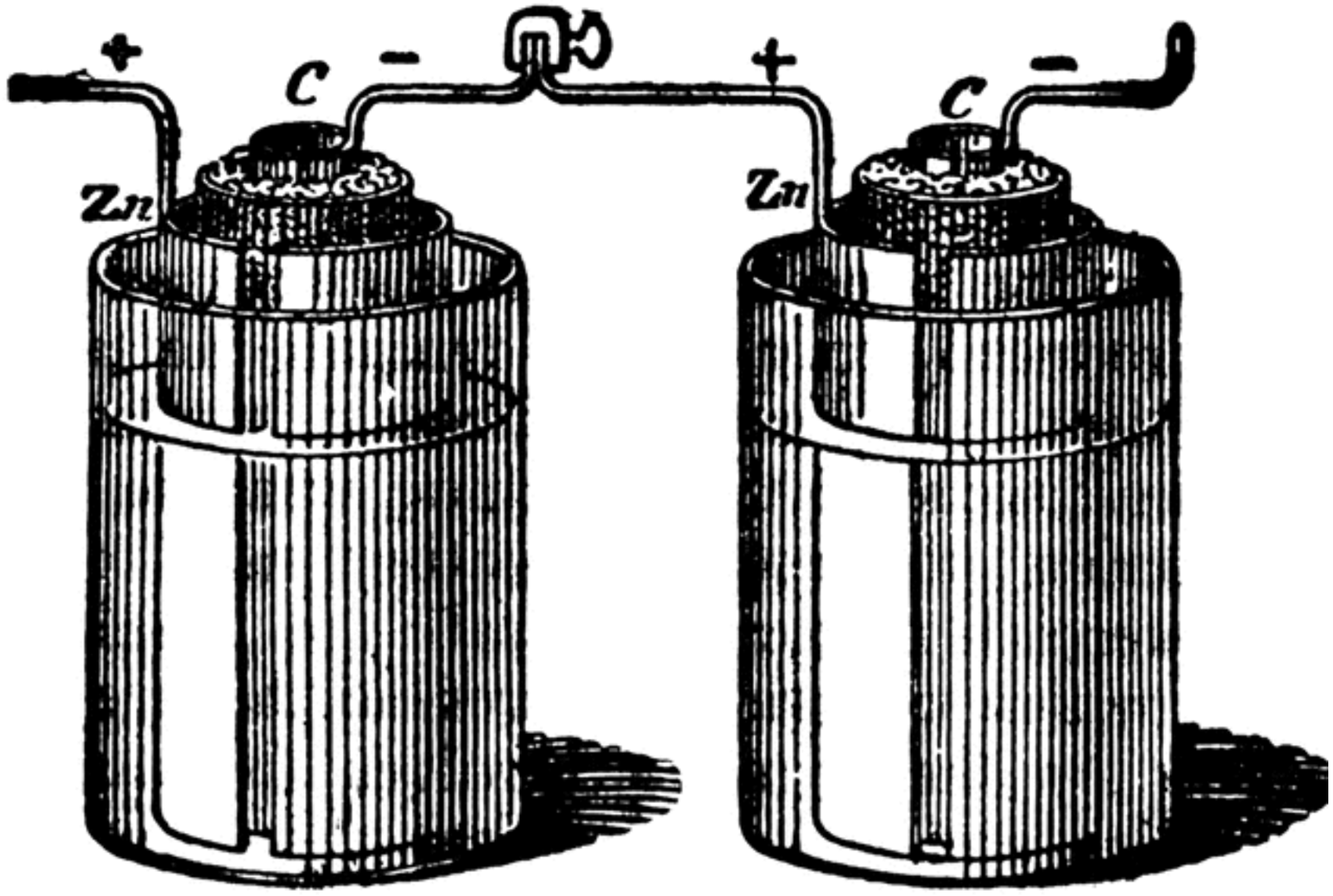
TCP, UDP, SSL, UNIX sockets

protocols



HTTP, IMAP, IRC, DNS

“Batteries-included”



HTTP server

```
from twisted.internet import reactor
from twisted.web.server import Site
from twisted.web.static import File

resource = File("/var/www/mysite")
factory = Site(resource)
reactor.listenTCP(80, factory)
reactor.run()
```

IRC server

```
from twisted.cred import checkers, portal
from twisted.internet import reactor
from twisted.words import service

wordsRealm = service.InMemoryWordsRealm(
    "example.com")
wordsRealm.createGroupOnRequest = True

checker = checkers.FilePasswordDB("passwords.txt")
portal = portal.Portal(wordsRealm, [checker])

reactor.listenTCP(6667, service.IRCFactory(
    wordsRealm, portal))
reactor.run()
```

network

filesystem

timer

- scheduling
- authentication
- interacting with databases
- using threads and processes
- testing event-driven programs

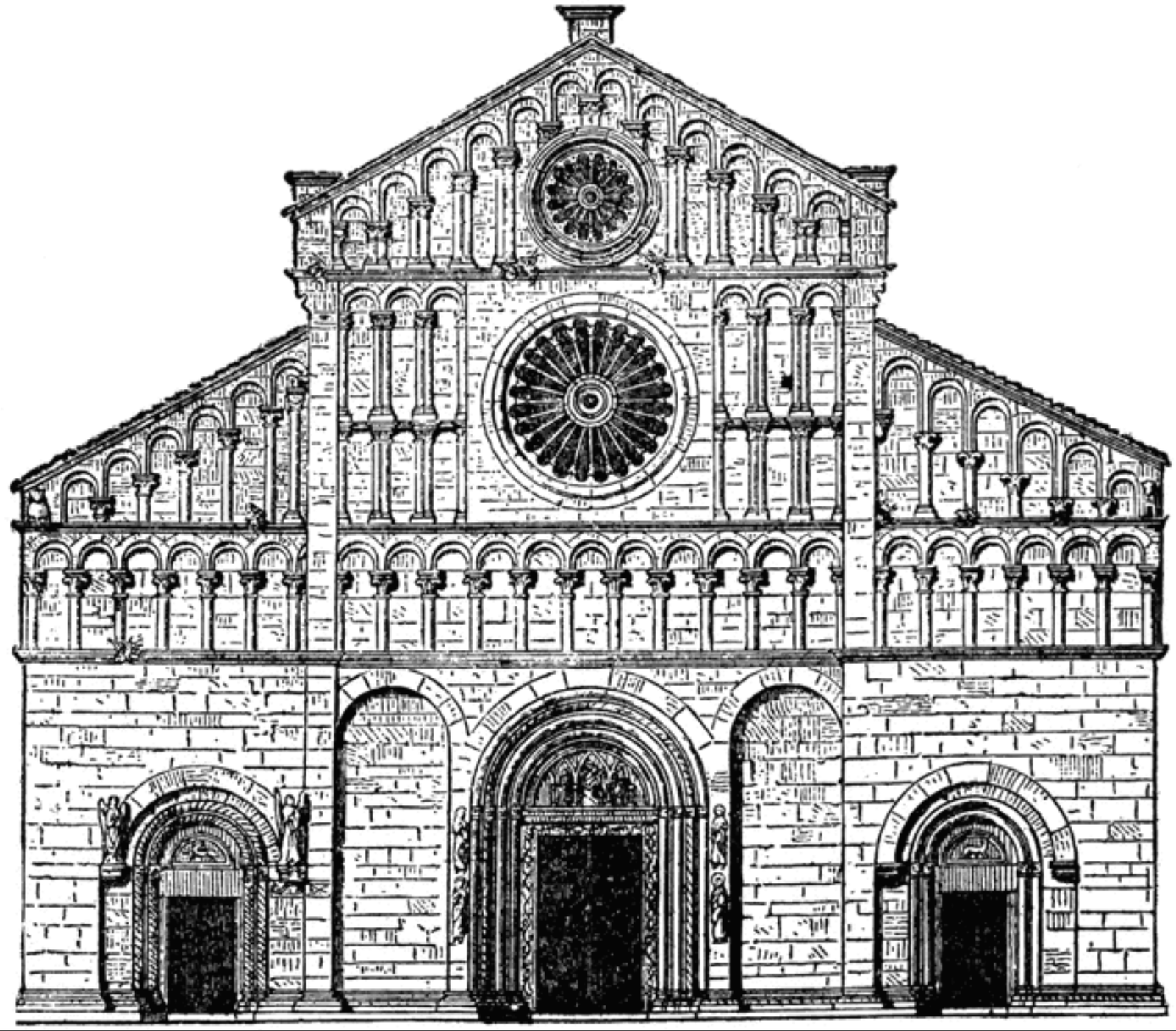
POP3 HTTP

IMAP DNS

SMTP SSH

FTP IRC

Twisted applications



TCP echo server

```
class Echo(protocol.Protocol):
    def dataReceived(self, data):
        self.transport.write(data)

class EchoFactory(protocol.Factory):
    def buildProtocol(self, addr):
        return Echo()

reactor.listenTCP(80, EchoFactory())
reactor.run()
```

echo_server.tac

```
from echo import EchoFactory

application = service.Application("echo")
echoService = internet.TCPServer(80,
                                 EchoFactory())
echoService.setServiceParent(application)
```

```
$ twistd -y echo_server.tac
2011-11-19 22:23:07 [-] Log opened.
2011-11-19 22:23:07 [-] twistd 12.1.0 (/usr/bin/python...
2011-11-19 22:23:07 [-] reactor class: selectreactor.....
2011-11-19 22:23:07 [-] echo.EchoFactory starting on 8000
2011-11-19 22:23:07 [-] Starting factory <EchoFactory>...
2011-11-19 22:23:20 [-] Received SIGTERM, shutting down.
2011-11-19 22:23:20 [-] (TCP Port 8000 Closed)
2011-11-19 22:23:20 [-] Stopping factory <EchoFactory>...
2011-11-19 22:23:20 [-] Main loop terminated.
2011-11-19 22:23:20 [-] Server Shut Down.
```

Twisted applications

- logging
- daemonization
- profiling
- authentication

twistd web --port 8000 --path .

```
twistd dns -v -p 5553 --hosts-file=hosts
```

```
sudo twistd conch -p tcp:2222
```

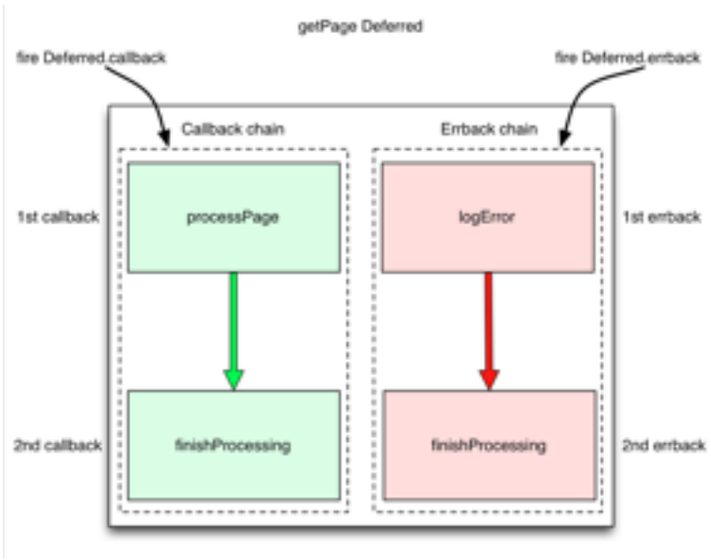
```
twistd mail -E localhost -d
example.com=/tmp/example.com
```

twistd mail --relay=/tmp/mail_queue

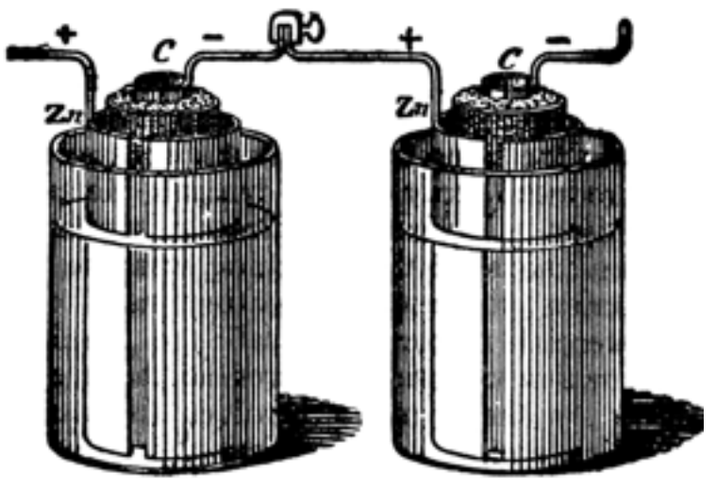
Architecture

reactor

Deferreds



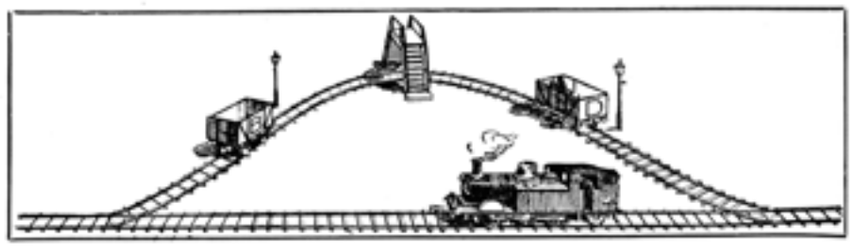
Applications



transports

TCP, UDP, SSL, UNIX sockets

protocols



HTTP, IMAP, IRC, DNS

Projects

- Ampoule
- Archive Proxy
- Bafload
- BeanREST
- Corotwine
- Downpour
- Flumotion
- Game
- Helios Certificate Authority
- M.U.T.A.N.T.
- MV3D
- POSTDoc
- Paisley
- Poetry
- PySMS
- Python Libevent
- Qt4 Reactor
- Synchronous Deferred
- Tahoe-LAFS
- Torc
- Twistar
- Twotp
- TxYoga
- Wokkel
- dAmnViper
- foolscap
- ldaprest
- merit
- pydirector
- python-tvi955
- turtl
- twisted-opm
- twisted-x11
- txACAP
- txAMQP
- txAWS
- txAuthProxy
- txCobalt
- txConvore
- txCumulus
- txEVE
- txEvolver
- txFigleafTrial
- txFluidDB
- txGenshi
- txLoadBalancer

- txM
- txMailServer
- txNetFlow
- txNetTools
- txPika
- txPostgres
- txProtoBuf
- txRDQ
- txRackspace
- txRemoteDeploy
- txRiak
- txScheduler
- txSequence
- txSimpleDB
- txSmug
- txSpore
- txStatsD
- txStomp
- txULS
- txXCP
- txZMQ
- txcomputegrid - twisted based compute grid
- txdnspython
- txmemcache
- txreCAPTCHA
- txrestapi
- txretry
- txroutes
- txsolr
- txzookeeper

launchpad

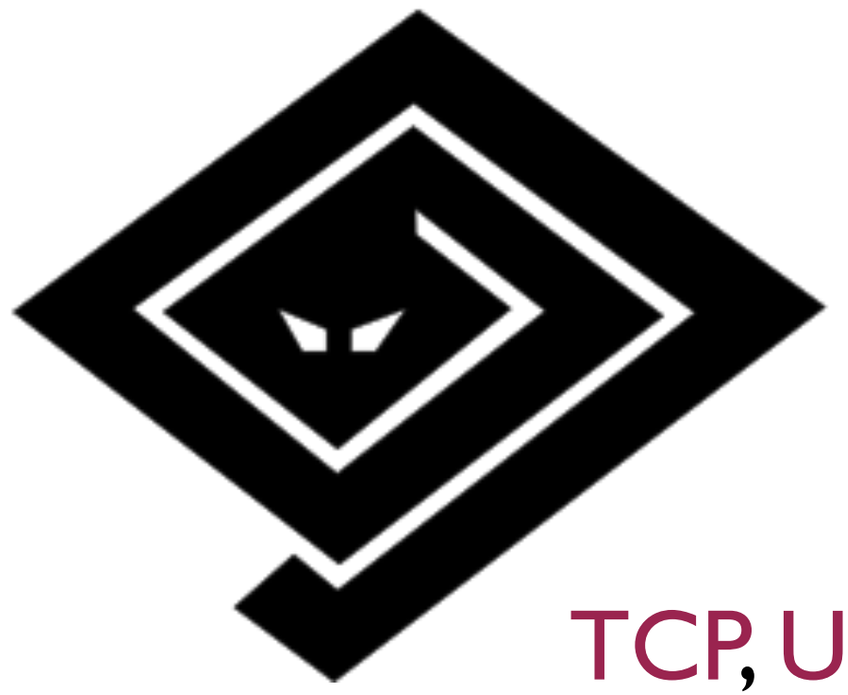
Buildbot

Scrapy

Tahoe-LAFS

Twisted: an event-driven networking framework

- event-driven
- cross-platform
- RFC-compliant
- “batteries-included”
- general and extensible
- clients and servers



TCP, UDP, SSL,
HTTP, IMAP, IRC,
SMTP, POP3, IMAP,
DNS, FTP...

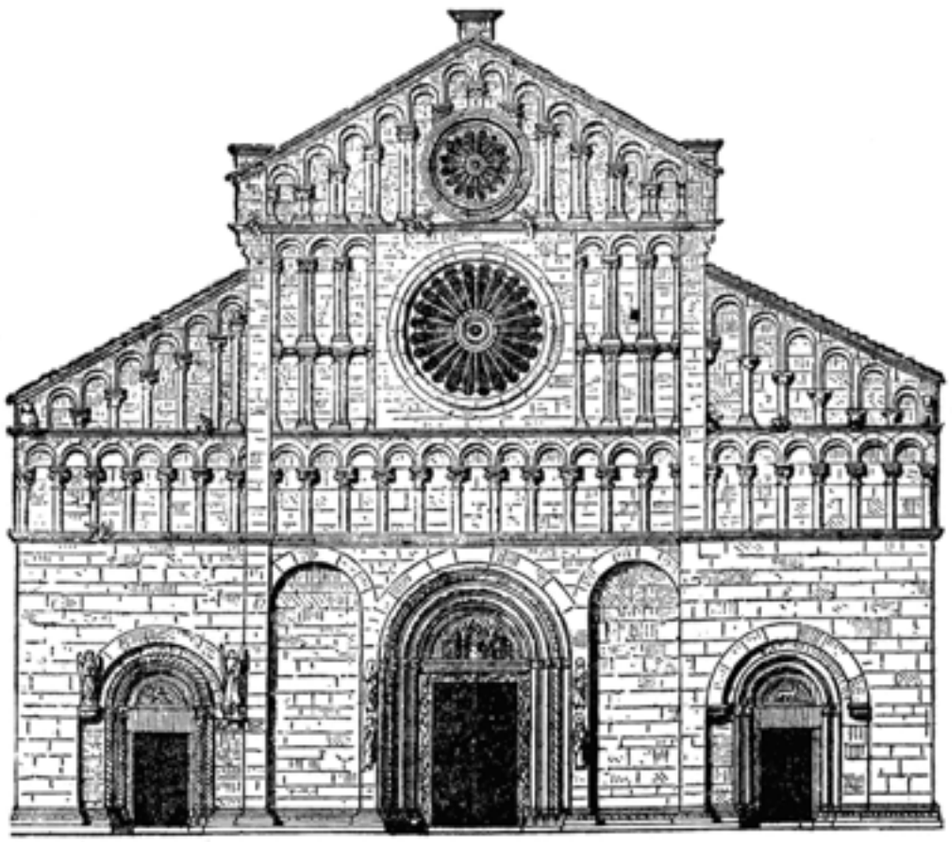
Tulip

http://www.python.org/dev/peps/pep-3156/

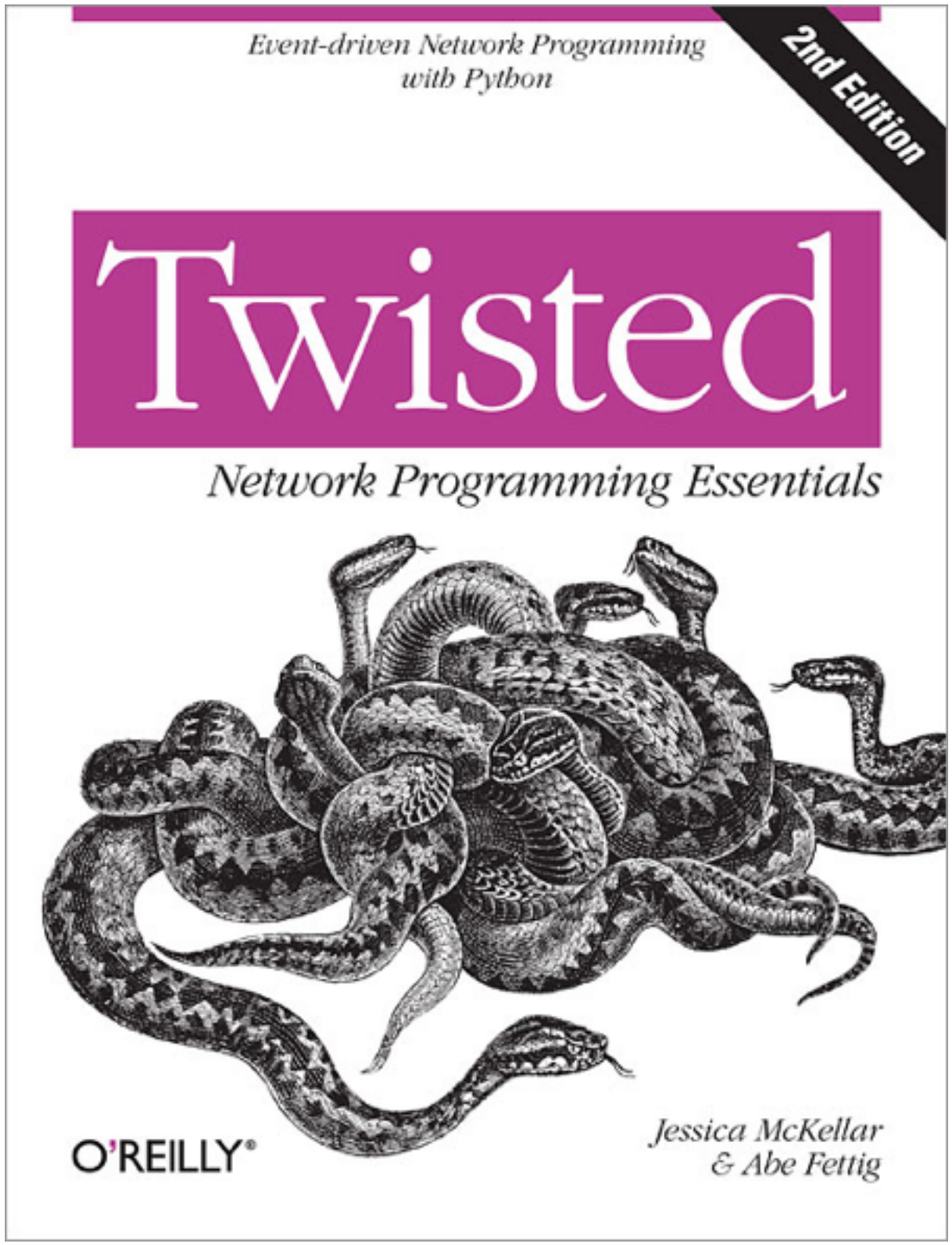
"This is a proposal for asynchronous I/O in Python 3, starting with Python 3.3...The proposal includes a pluggable event loop API, transport and protocol abstractions similar to those in Twisted, and a higher-level scheduler based on `yield from` (PEP 380)."

Food for thought

- Evolution of async primitives in Python and other languages
- Twisted as a monolithic framework



Thank you!


Event-driven Network Programming
with Python
2nd Edition
Twisted
Network Programming Essentials
O'REILLY®
Jessica McKellar
& Abe Fettig

@inlineCallbacks

```
@defer.inlineCallbacks
def download(url):
    try:
        page = yield getPage(url)
        processPage(page)
    except Error, e:
        logError(e)
    finally:
        finishProcessing(page)
```

@inlineCallbacks

- generator functions: "restartable functions" that use yield to produce iterators
- coroutines: generator functions that can accept arguments as well as `yield` them
- decorators: callables that takes a function as an argument and return a replacement function; they "wrap" functions.

@inlineCallbacks

- generator functions: "restartable functions" that use yield to produce iterators

```
def countdown(n):
    while n > 0:
        yield n
        n -= 1

x = countdown(10)
for num in x:
    print num
```

@inlineCallbacks

- coroutines: generator functions that can accept values as well as `yield` them

```
x = yield n
```

@inlineCallbacks

- decorators: callables that takes a function as an argument and return a replacement function; they "wrap" functions.

```
def decorate(func):
    print "decorating"
    return func

@decorate
def countdown(n):
    ...
```

@inlineCallbacks

- generator functions: "restartable functions" that use yield to produce iterators
- coroutines: generator functions that can accept arguments as well as `yield` them
- decorators: callables that takes a function as an argument and return a replacement function; they "wrap" functions.

@inlineCallbacks

```
@defer.inlineCallbacks
def download(url):
    try:
        page = yield getPage(url)
        processPage(page)
    except Error, e:
        logError(e)
    finally:
        finishProcessing(page)
```