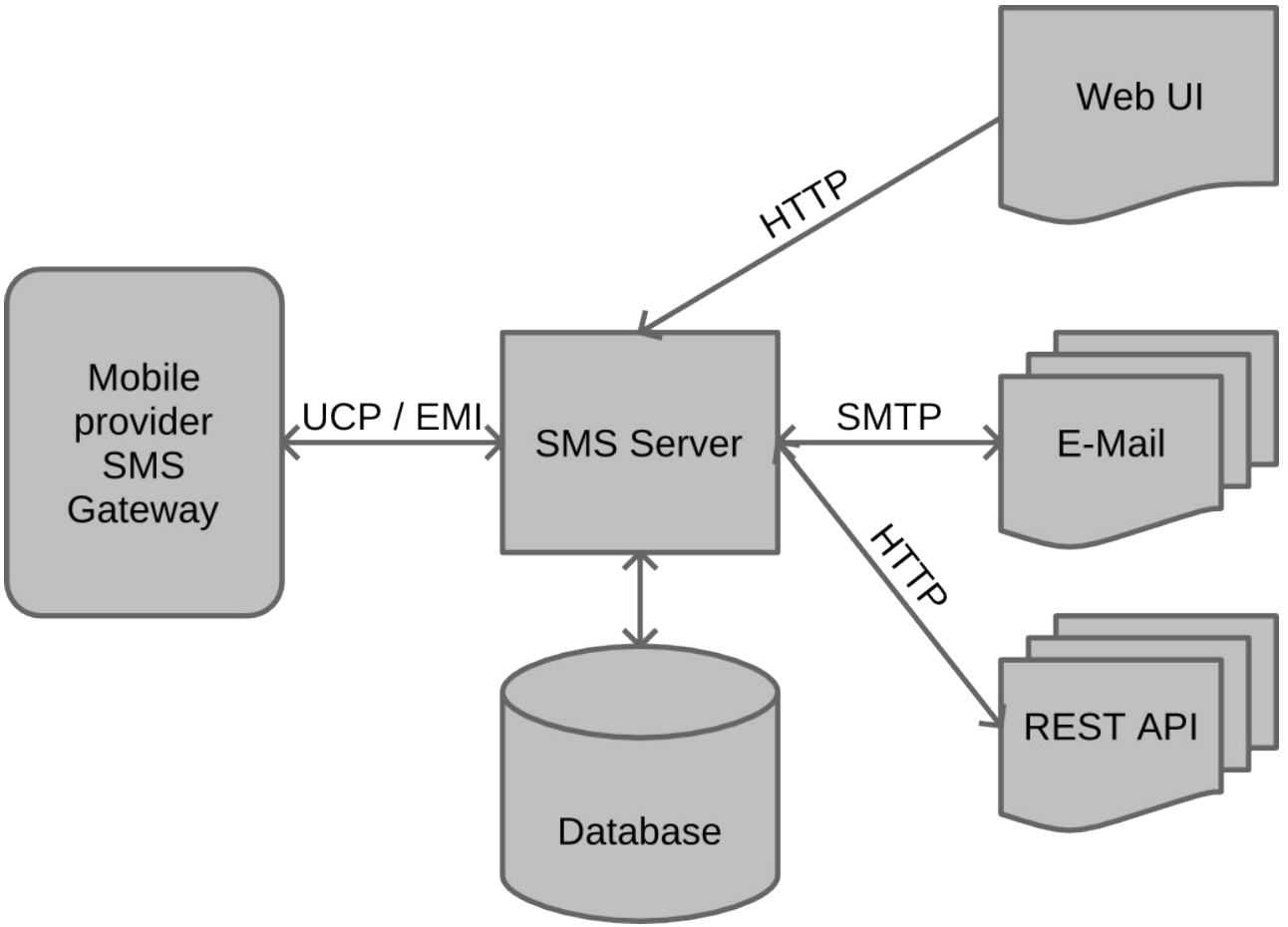
Twisted

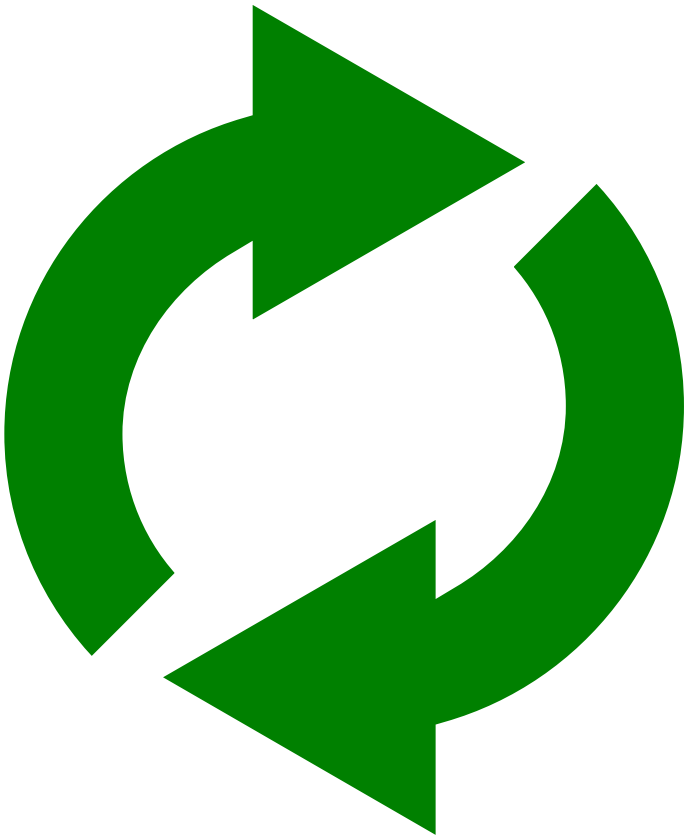
is an
event-driven
network programming framework
written in Python.

Why I dare to give this talk



Reactor

- Event-loop
- Handles IO
- Invokes callbacks on events

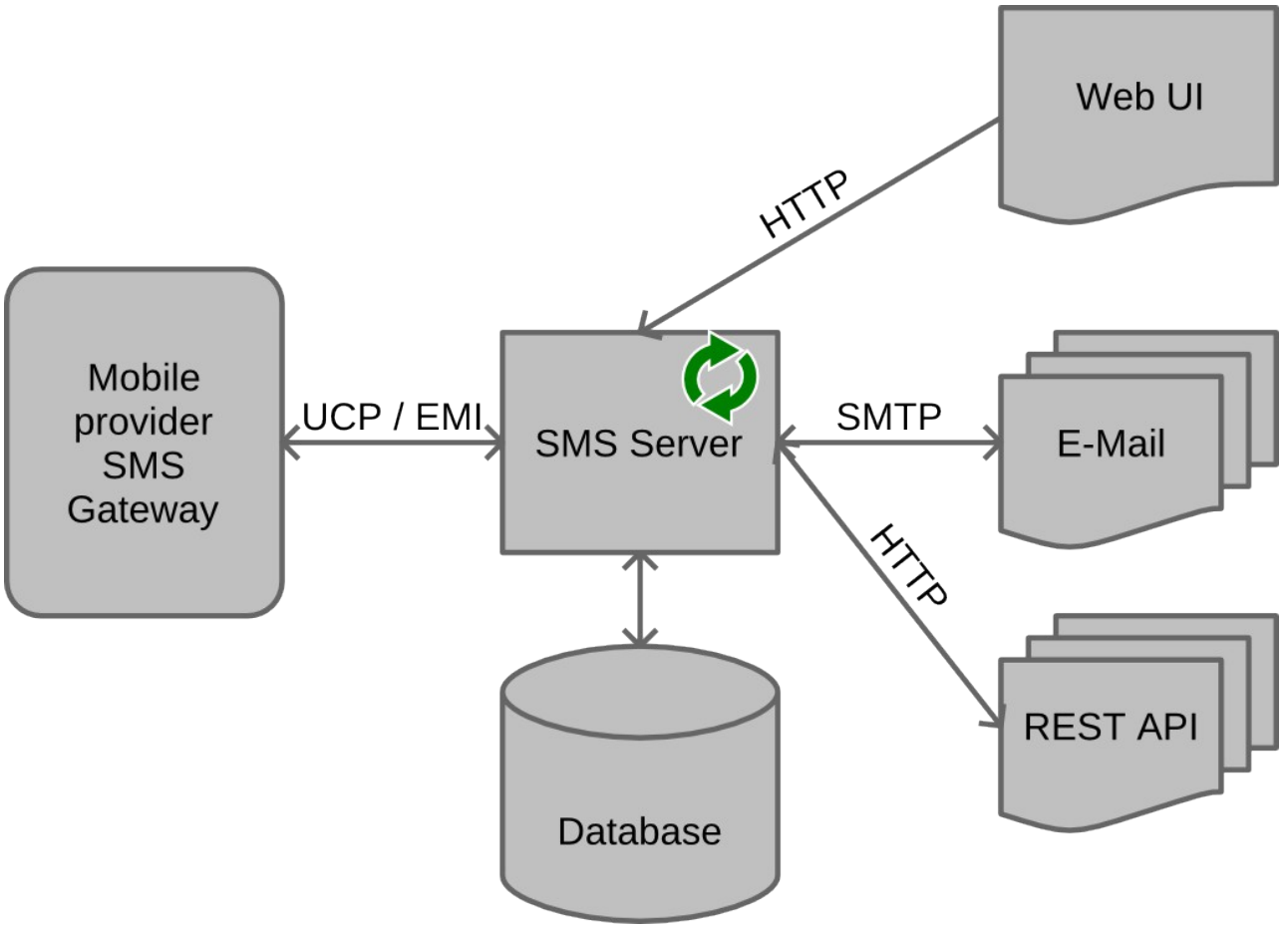


Example

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

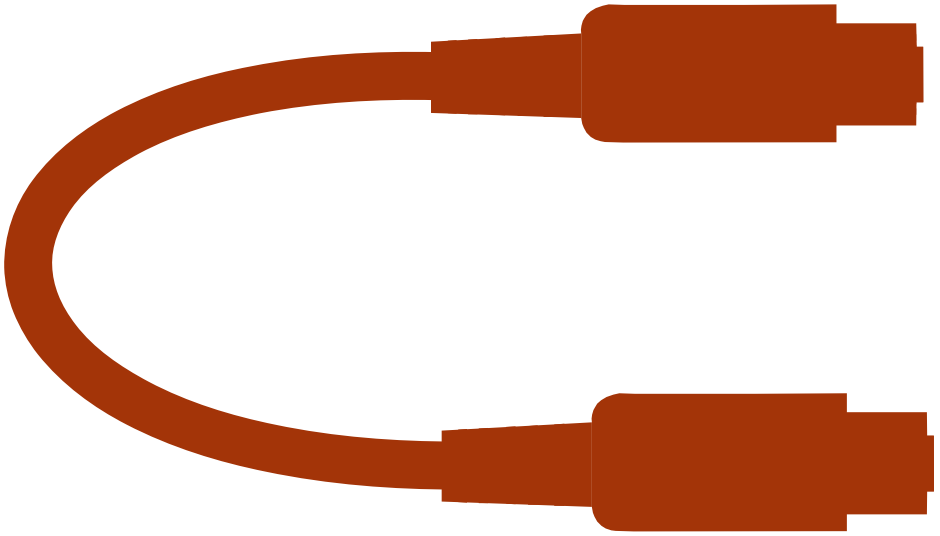
```
reactor.run()
```

The reactor and the server



Transport

- TCP, UDP, TLS, ...
- Transfers data over the wire
- Data agnostic
- Different flavors

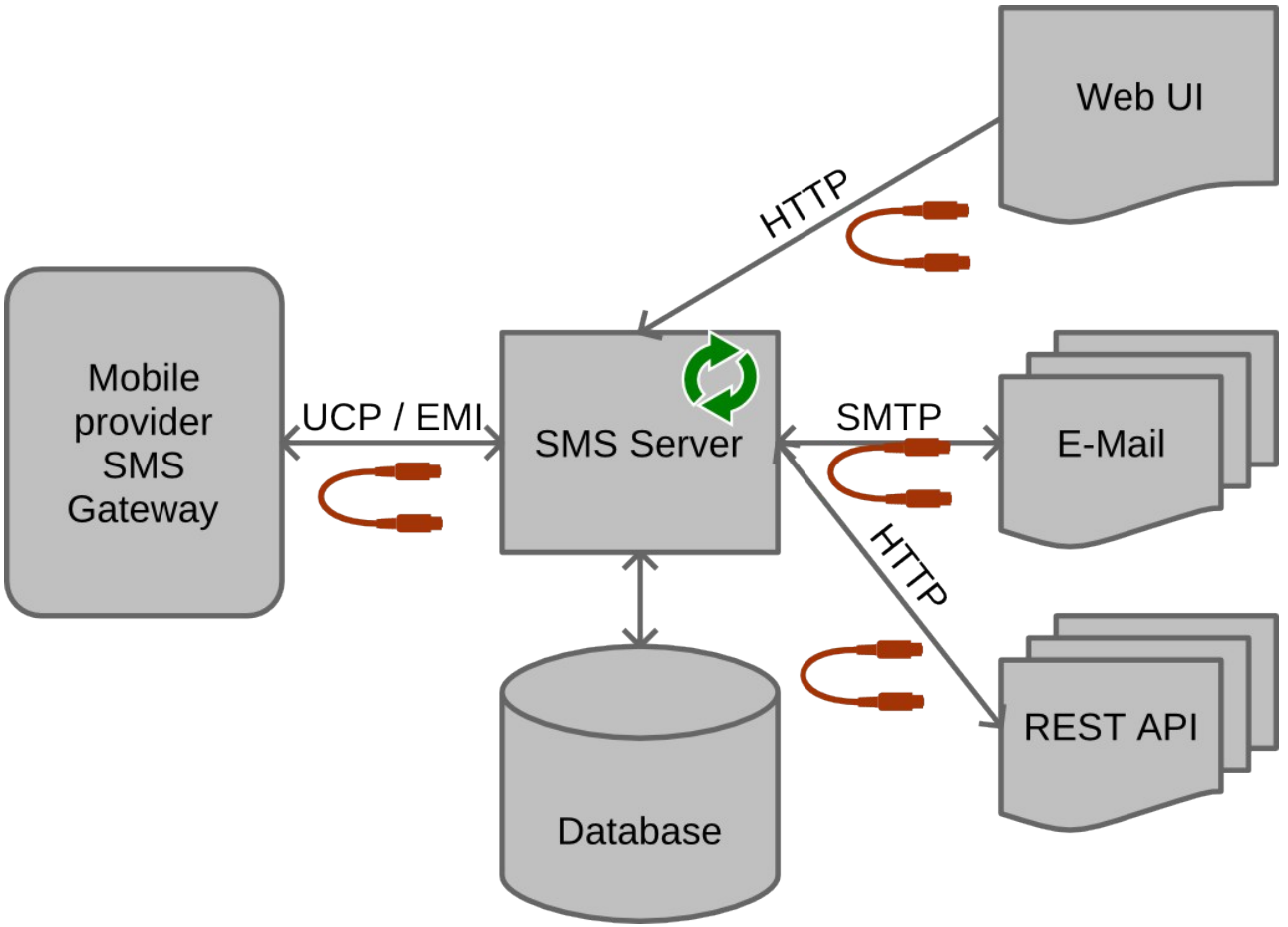


Example

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

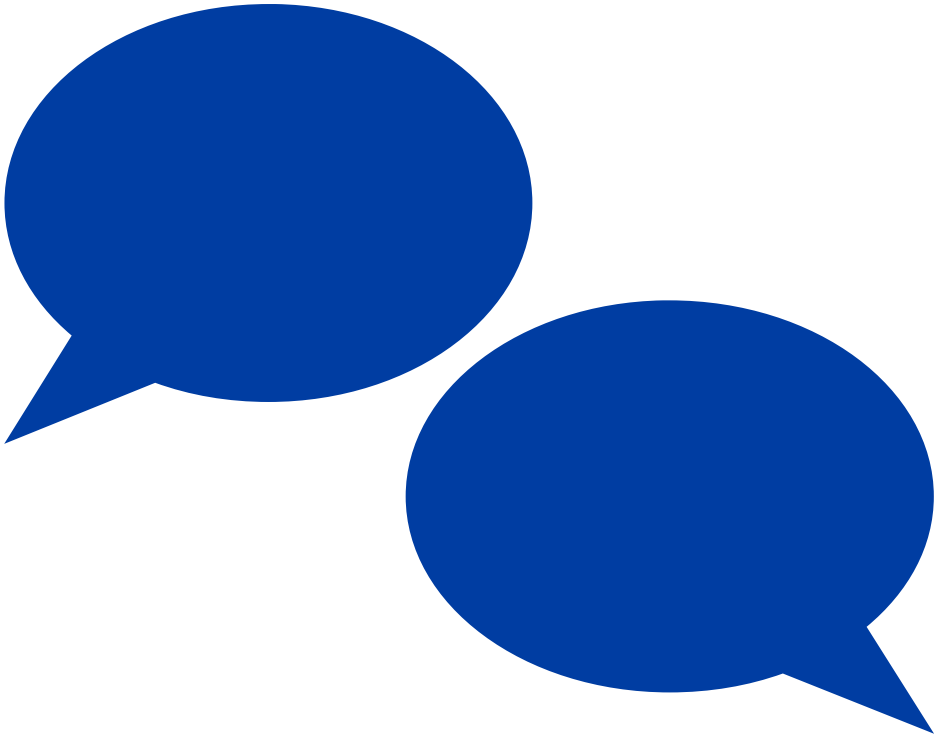
```
reactor.listenTCP(1234, ...)
reactor.run()
```

The transports and the server



Protocol

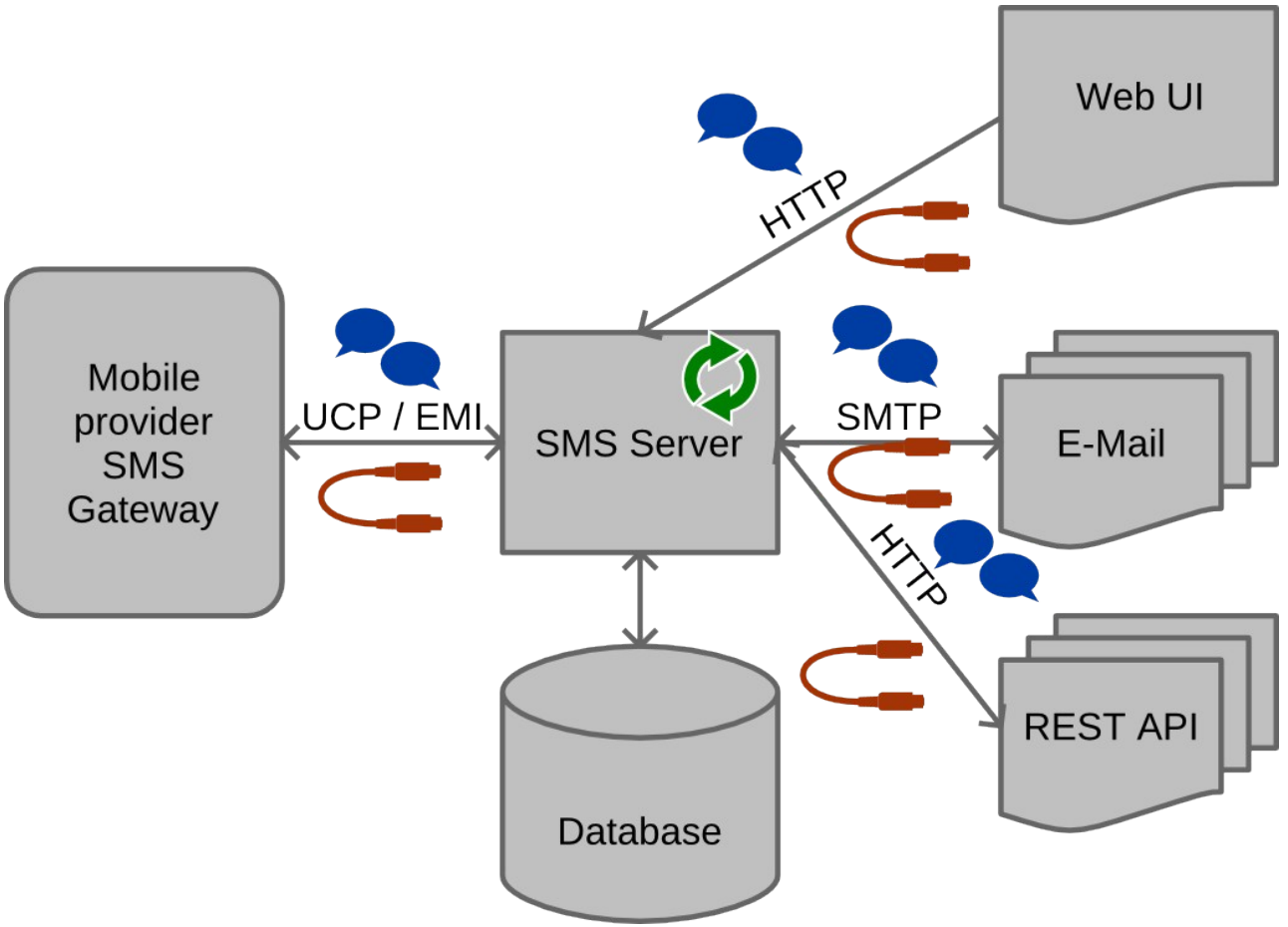
- HTTP, SMTP, SSH, ...
- Knows how to handle received data
- Knows how to prepare data for sending



Example

```
from twisted.internet import reactor, protocol

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

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

reactor.listenTCP(1234, EchoFactory())
reactor.run()
```

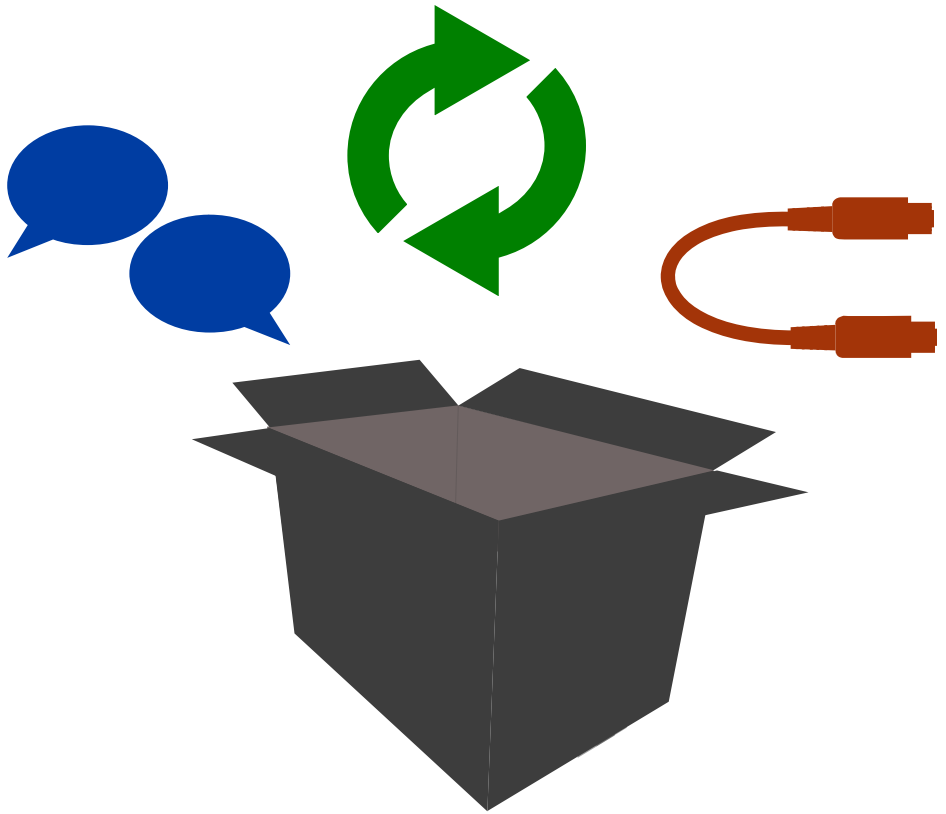
The protocols and the server



What else?

Applications

- HTTP Server,
  IRC Server,
  SSH Server, ...
- Combine protocol, transport and reactor
- Can be executed by `twistd`



A simple HTTP Server

```
twistd -n web --path . --port 8080
```

The same in code

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

resource = File('.')
factory = Site(resource)

reactor.listenTCP(8080, factory)
reactor.run()
```

... it's about web frameworks

Twisted Web Hello World

```
from twisted.internet import reactor
from twisted.web.server import Site
from twisted.web.resource import Resource

class HelloPage(Resource):
    isLeaf = True
    def render_GET(self, request):
        return "<html><body>Hello Twisted</body></html>"

resource = HelloPage()
factory = Site(resource)

reactor.listenTCP(8880, factory)
reactor.run()
```

And Now for Something Completely Different...

Asynchronous Programming

An example

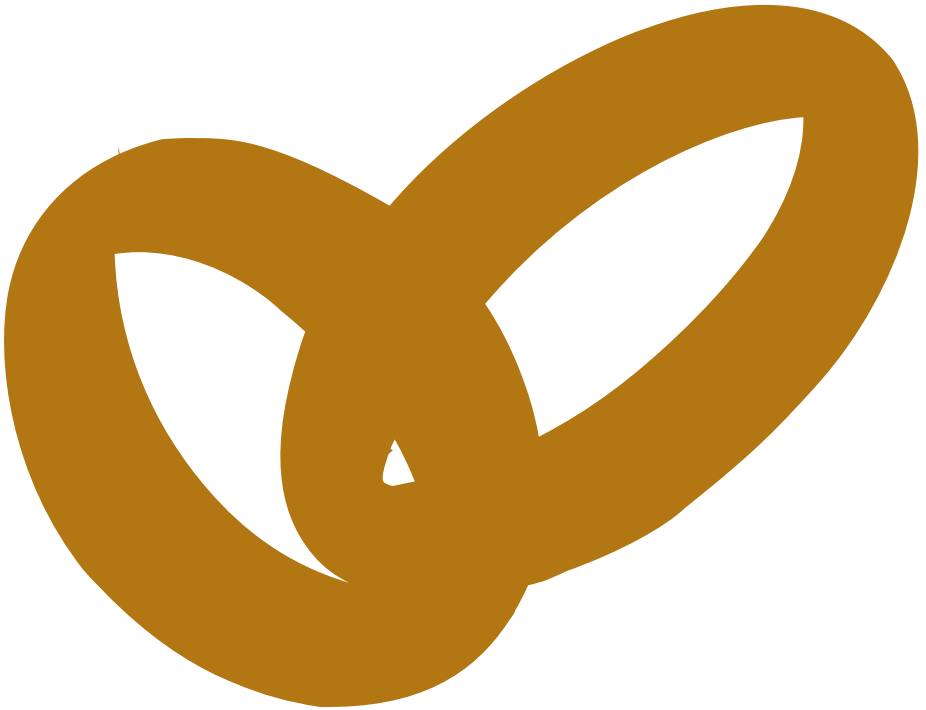
```
import urllib2


response = urllib2.urlopen('http://python.org/')
html = response.read()
print(html)
```

The async variant

```
# This code snippet is not runnable...
from fake_library import get_url, mainloop

def callback(html):
    print(html)

get_url("http://python.org", callback)

mainloop()
```

Deferred

- aka. Promise, Future
- Allow to return results that are not yet ready



The same in twisted

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

def printResult(html):
    print(html)
    reactor.stop()

d = getPage("http://python.org")
d.addCallback(printResult)

reactor.run()
```

Even better in twisted

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

@defer.inlineCallbacks
def printResult():
   html = yield getPage("http://python.org")
   print(html)
   reactor.stop()

printResult()
reactor.run()
```

Lets handle some errors

```
import urllib2

try:
   response = urllib2.urlopen('http://python.org/')
   html = response.read()
   print(html)
except Exception:
   print("Unable to fetch.")
```

Error handling using errbacks

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

def printResult(html):
   print(html)
   reactor.stop()

def printError(err):
   print("Unable to fetch")
   reactor.stop()

d = getPage("http://python.org")
d.addCallback(printResult)
d.addErrback(printError)

reactor.run()
```

Error handling usind yield

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

@defer.inlineCallbacks
def printResult():
    try:
        html = yield getPage("http://python.org")
        print(html)
    except Exception:
        print("Unable to fetch.")
    finally:
        reactor.stop()

printResult()
reactor.run()
```

Summary

- Use twisted to...
- network applications with many protocols involved
- structure your async code
- write high performance event and IO driven applications

Thank you!