TWISTED
CONCEPTS & PATTERNS

INTRODUCTION

PATRICK CLOKE

LEAD ENGINEER @ PERCIPIENT NETWORKS

&

MOZILLIAN

GITHUB.COM/CLOKEP

STEPHEN DICATO

CO-FOUNDER & VP, ENGINEERING

@ PERCIPIENT NETWORKS

GITHUB.COM/DICATO

IF YOU TWITTER

@CLOKEP &
@STEPHENDICATO

STRONGARM.IO

automate your incident response

questions

WELCOME

GITHUB.COM/PERCIPIENT/TALKS

expectations

LEARN THESE 6 CONCEPTS

1. WHAT IS ASYNC PROGRAMMING

2. WHAT IS TWISTED

3. WHEN/WHY TO USE TWISTED

4. EVENT LOOP (REACTOR)

5. DEFERREDS

6. PROTOCOLS (AND MORE)

BONUS: USE TWISTED TO BUILD SYSTEMS & SERVICES

COMPOSABLE & SCALABLE SYSTEMS

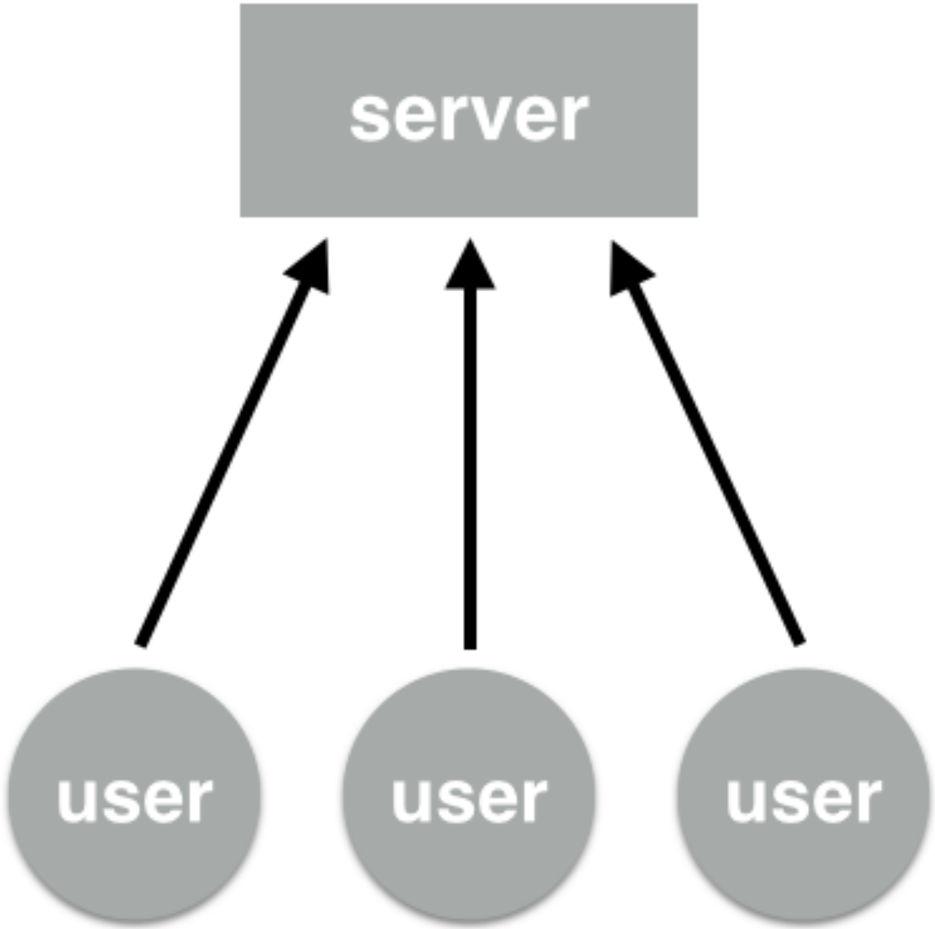
I.E., SERVICE ORIENTED ARCHITECTURE

I.E., MICROSERVICES

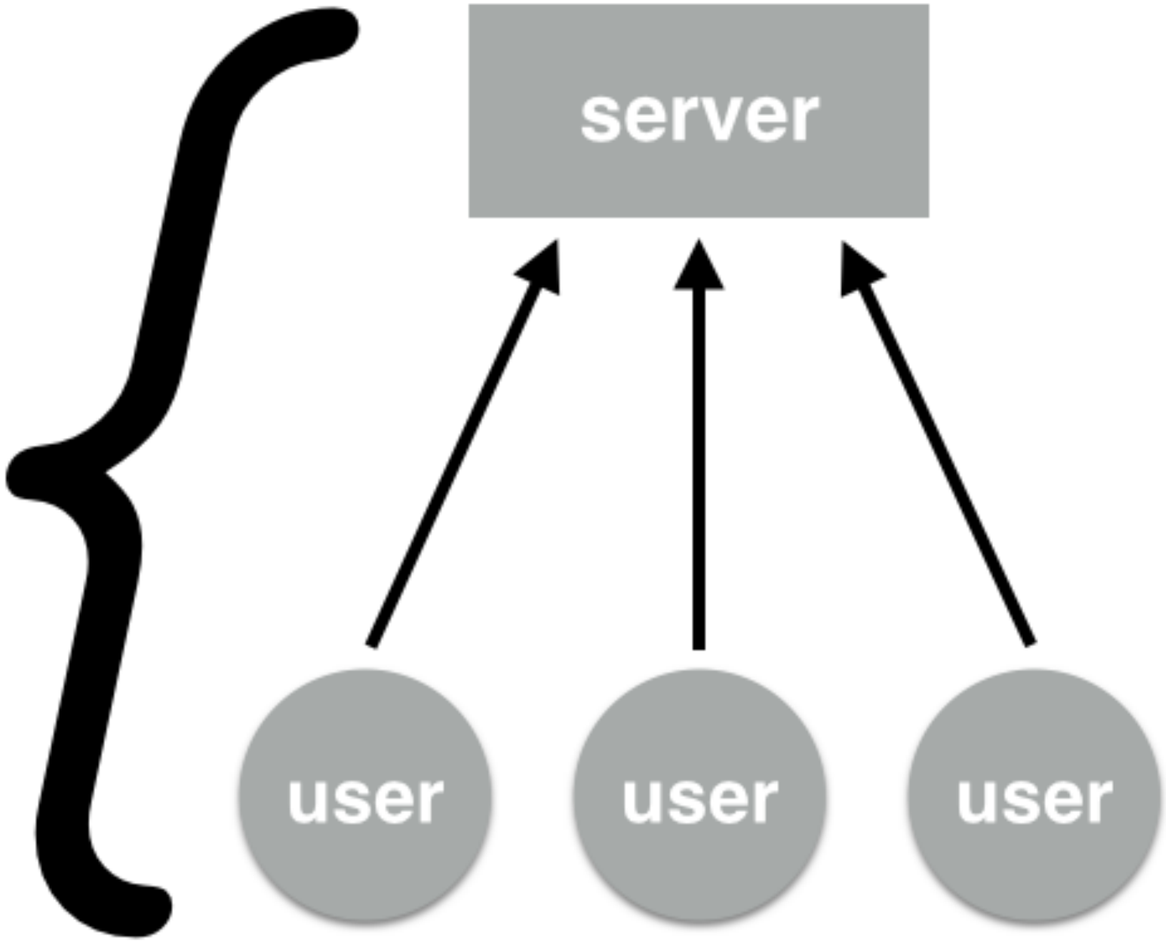
CONTEXT
AN EXAMPLE APP

chat server EXAMPLE (WITH ADMIN DASHBOARD)

- Clients use netcat
- Messages are broadcast to all users
- Clients are sent a banner on login
- Banner is configurable via an admin webpage

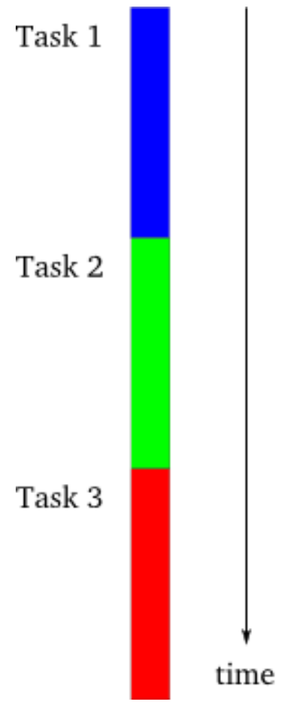

server
user
user
user

TCP
line based protocol



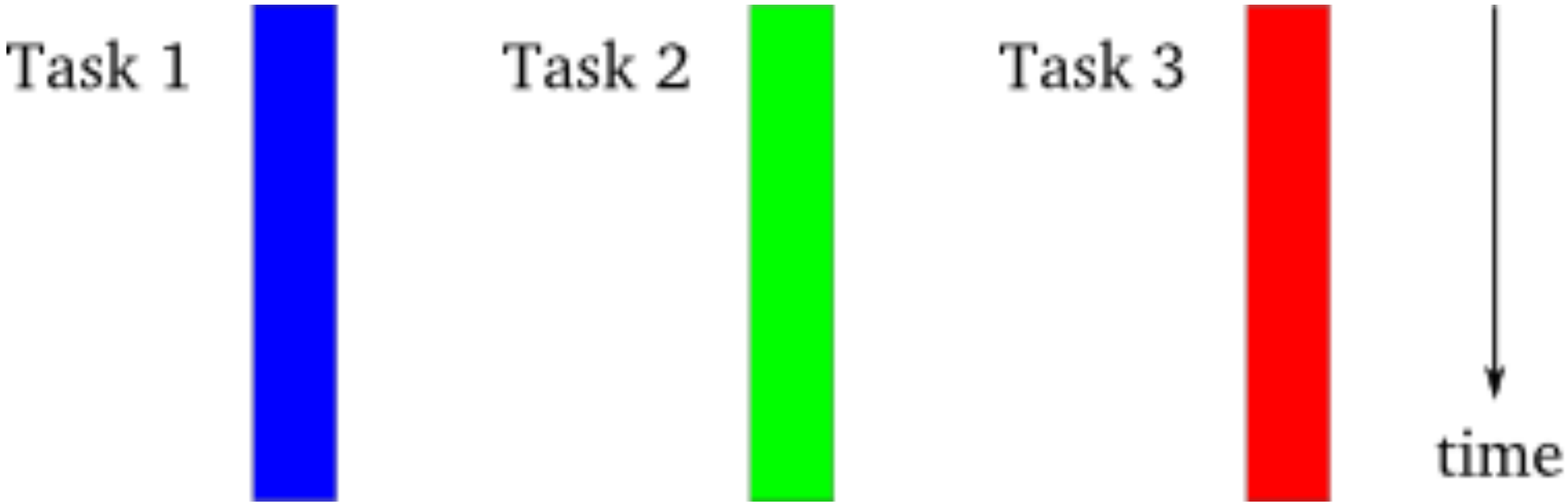
1. WHAT IS ASYNC PROGRAMMING

SYNCHRONOUS



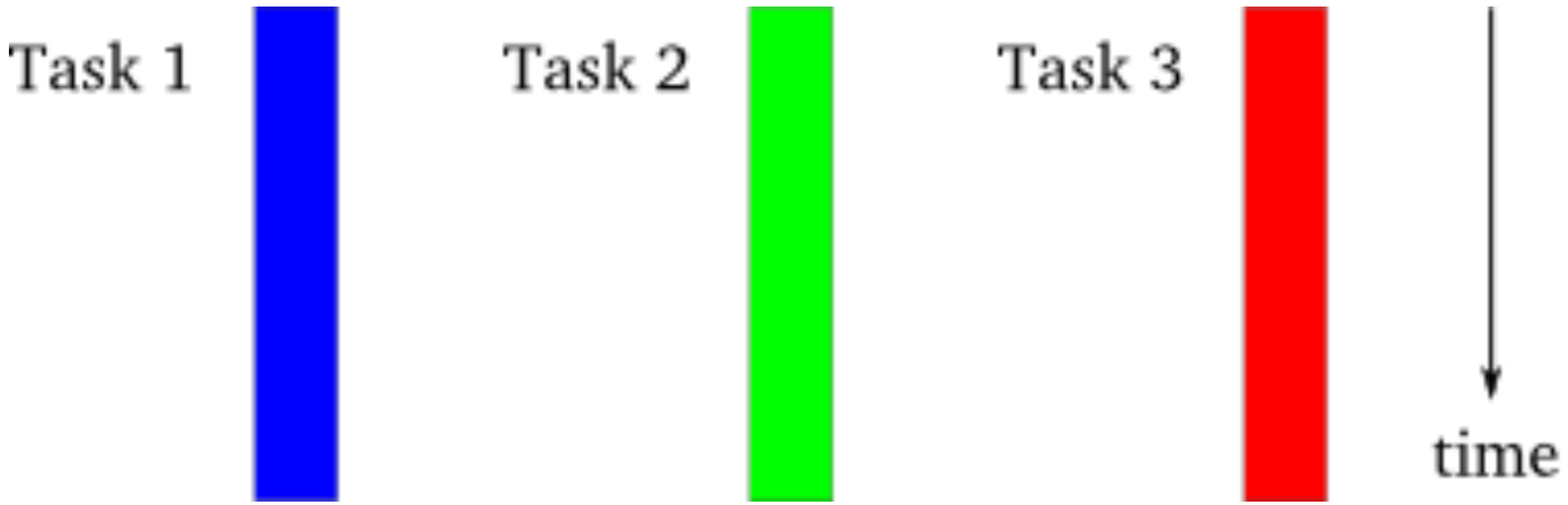
http://krondo.com/

THREADED



http://krondo.com/

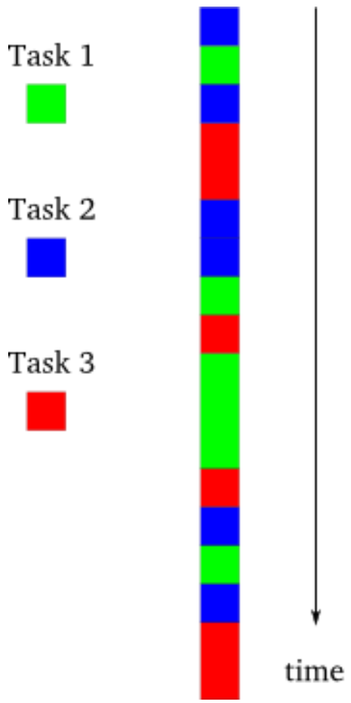
MULTIPROCESSING



multiprocessing uses same model

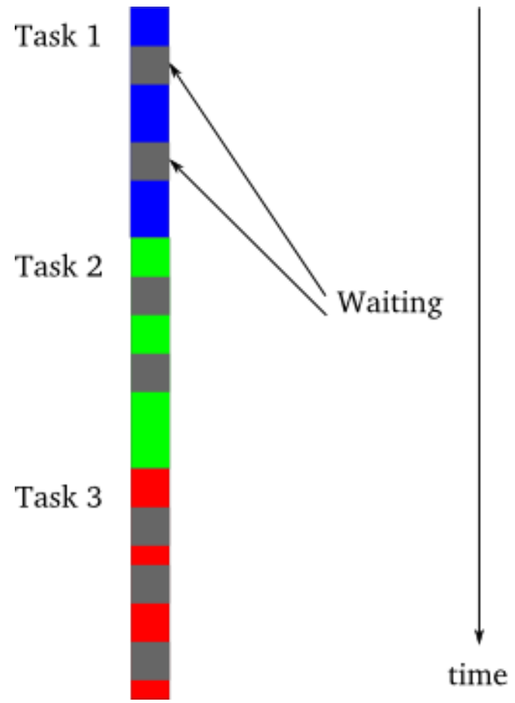
http://krondo.com/

ASYNC



http://krondo.com/

BLOCKING



http://krondo.com/

2. WHAT IS TWISTED

TWISTED:
AN EVENT-DRIVEN, *asynchronous* NETWORKING ENGINE

EVENT-DRIVEN & ASYNCRHONOUS

- user code is triggered when something it cares about happens
  - examples: new connection, data is available, HTTP request received, etc.
- non-blocking while waiting for I/O (e.g. bytes on a socket)

NETWORKING ENGINE

- supports many clients & servers out of the box
e.g. web (HTTP), chat (IRC, XMPP), mail (SMTP/IMAP/POP3)
- easily develop fully custom protocols
- easily customize behavior of built in protocols

3. WHEN/WHY TO USE TWISTED

1. high-level networking APIs
2. large protocol library
3. scalable due to asynchronous design
4. mature, with an emphasis on quality

TWISTED WILL NOT

- magically make code *non-blocking*
  i.e. help with CPU-bound tasks†
- be the simplest library to make a simple HTTP request‡

4. EVENT LOOP (REACTOR)

EVENT LOOP APIS:

NETWORKING, THREADING, EVENT DISPATCHING, TIMING, ETC.

REACTOR DEPENDS ON PLATFORM AND USAGE

(REACTOR IS A *global singleton*)

EXAMPLE (FROM `runner.py`)

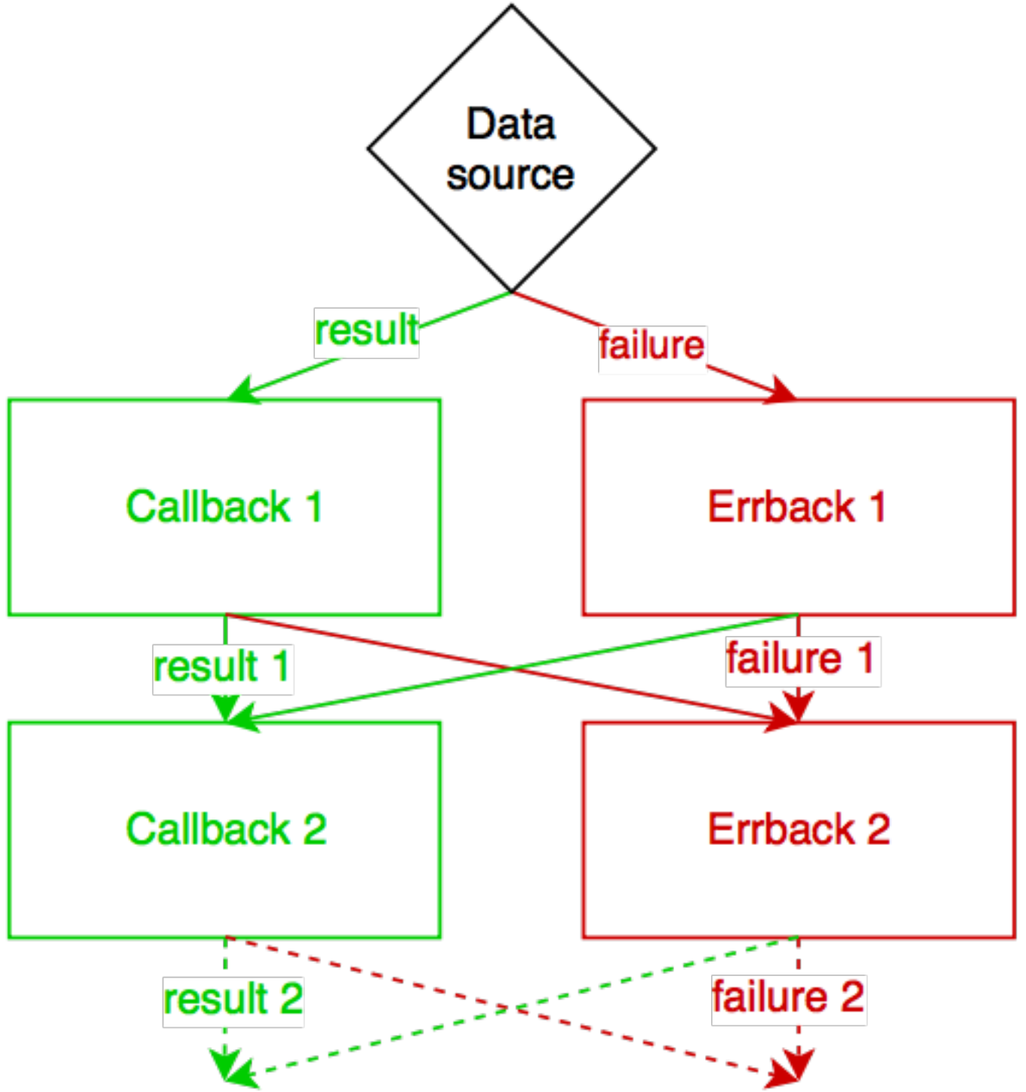
```
from twisted.internet import reactor, endpoints
from twisted.web import server

from chat import NetCatChatFactory

# Create an instance of the factory.
factory = NetCatChatFactory()

# Listen on TCP port 1400 for chat.
endpoints.serverFromString(reactor, "tcp:1400").listen(factory)

# Start listening for connections (and run the event-loop).
reactor.run()
# Note that any code after this point will *not* be executed. reactor.run enters
# an infinite loop until shutdown.
```

5. DEFERREDS

A *deferred* IS A PROMISE THAT A FUNCTION WILL EVENTUALLY HAVE A *result*

IN OTHER WORDS:
DEFERREDS ARE A PLACEHOLDER FOR A FUTURE RESULT

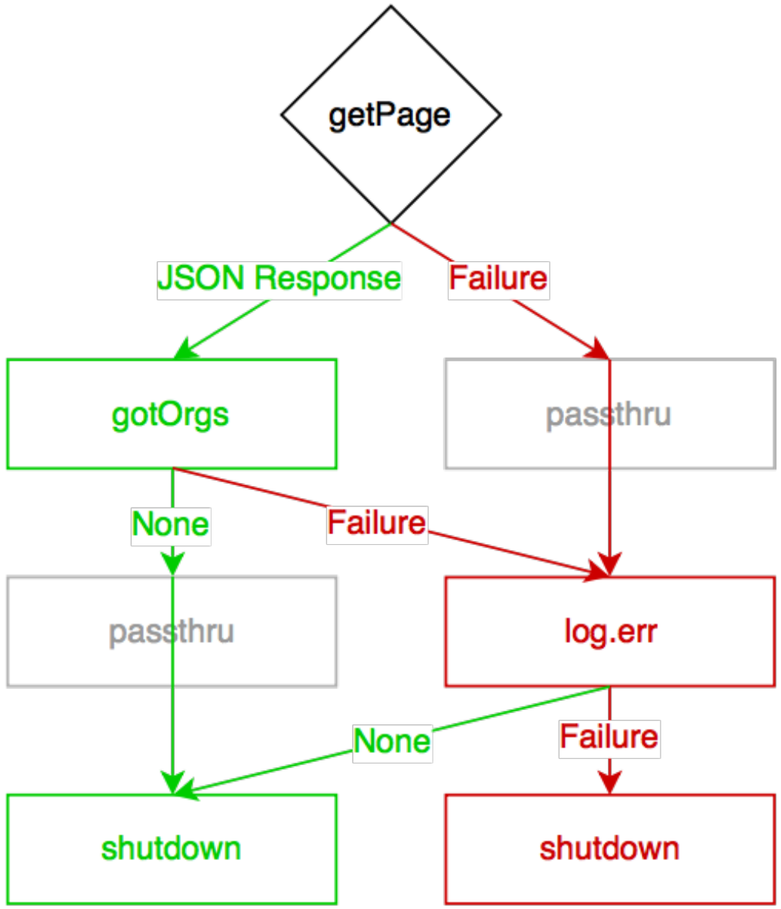
DEFERREDS MANAGE A CALLBACK CHAIN



EXAMPLE (FROM deferred_ex.py)

```
import json
from twisted.internet import reactor
from twisted.python import log
from twisted.web import client

def gotOrgs(data):
    print("Organization request done!")
    # Parse the JSON payload. TODO Error checking.
    orgs = json.loads(data)
    # Find the names of the organizations and print them.
    org_names = sorted([org['login'] for org in orgs])
    print('\n'.join(org_names))

def shutdown(ignored):
    print("Shutting down!")
    reactor.stop()  # No matter what happens, shutdown the eventloop.

# The Deferred.
d = client.getPage('https://api.github.com/users/clokep/orgs')
d.addCallback(gotOrgs)  # The callback for a successful request.
d.addErrback(log.err)  # Before shutdown, log any errors.
d.addBoth(shutdown)  # The callback/errback.

reactor.run()  # Start the eventloop.
```

EXAMPLE DEFERRED FLOW (FROM `deferred_ex.py`)



DEMO

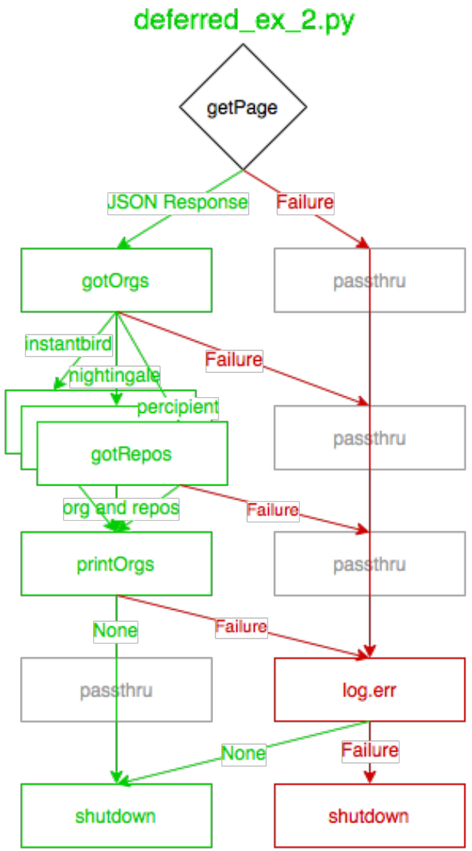
ADVANCED EXAMPLE, PART 1. (FROM deferred_ex_2.py)

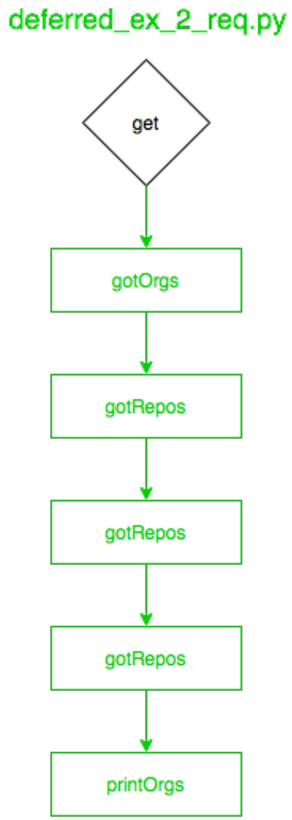
```
import json
from twisted.internet import defer, reactor
from twisted.python import log
from twisted.web import client

def gotRepos(data, org):
    pass

def gotOrgs(data):
    pass

def printOrgs(org_list):
    print("Outputting repos")
    for success, (org, repos) in org_list:
        print('\t%s: %s' % (org, ', '.join(repos) if repos else '(none)'))

def shutdown(ignored):
    print("Shutting down!")
    reactor.stop()  # No matter what happens, shutdown the eventloop.

# The Deferred.
d = client.getPage('https://api.github.com/users/clokep/orgs')
d.addCallback(gotOrgs)  # The callback for a successful request.
d.addCallback(printOrgs)
d.addErrback(log.err)  # Before shutdown, log any errors.
d.addBoth(shutdown)  # The callback/errback.

reactor.run()  # Start the eventloop.
```

ADVANCED EXAMPLE, PART 2. (FROM deferred_ex_2.py)

```
def gotRepos(data, org):
    """Got the repos for an org, return a tuple of (org name, repos)."""
    print("Got repo information for %s" % org)

    # Parse the JSON payload. TODO Error checking.
    repos = json.loads(data)
    if not repos:  # no repos, return early
        return (org, [])
    # The names of the repos in alphabetical order.
    names = sorted([repo['name'] for repo in repos])
    return (org, names)

def gotOrgs(data):
    print("Organization request done!")
    # Parse the JSON payload. TODO Error checking.
    orgs = json.loads(data)
    # The names of the organizations in alphabetical order.
    org_names = sorted([bytes(org['login']) for org in orgs])

    # Now request the repos under each org.
    ds = []
    for org in org_names:
        print("\t%s" % org)  # print out the org
        d = client.getPage('https://api.github.com/orgs/%s/repos' % org)
        d.addCallback(gotRepos, org)  # pass the org name to the callback.
        ds.append(d)

    # Returning a Deferred causes the next callback to wait.
    return defer.DeferredList(ds)
```

ADVANCED EXAMPLE DEFERRED FLOW (FROM `deferred_ex.py`)



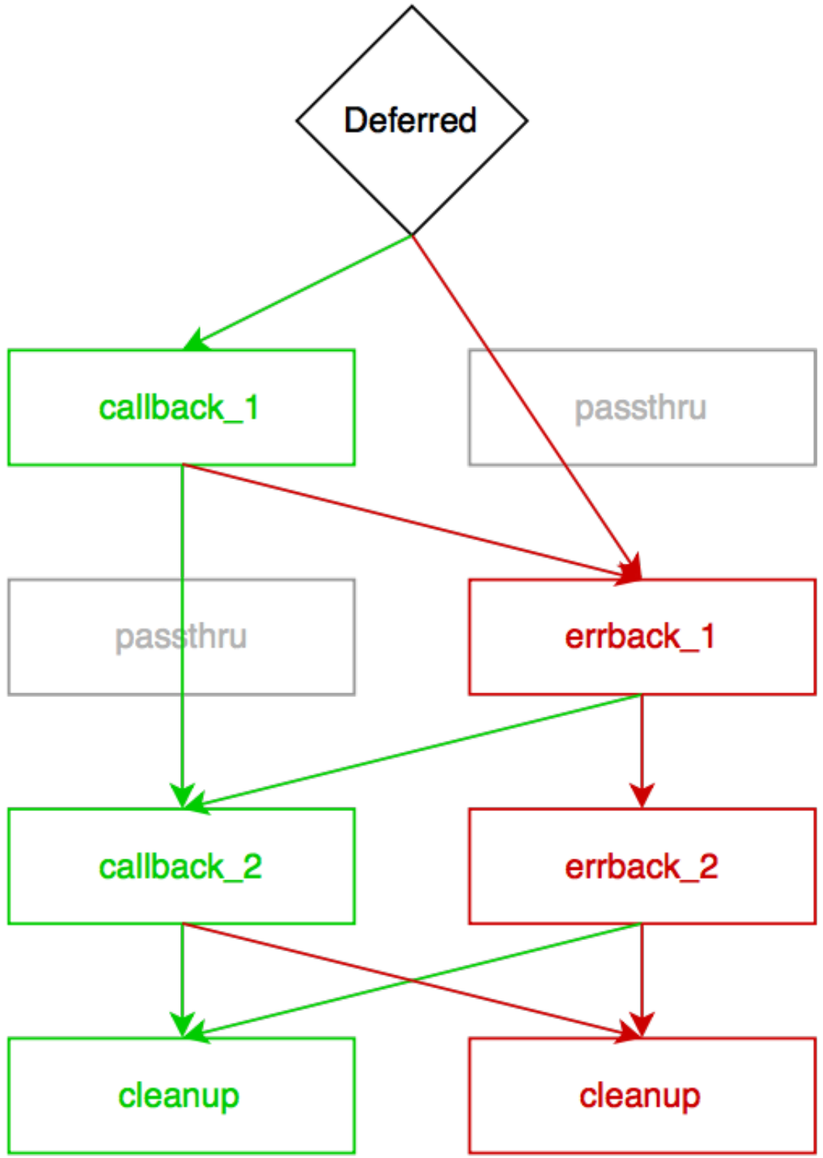


DEMO

DEFERREDS FINE PRINT

```
d = defer.Deferred()
d.addCallback(callback_1)
d.addErrback(errback_1)
d.addCallbacks(callback_2, errback_2)
d.addBoth(cleanup)
```

1. Each added callback/errback appends a new *level* to the chain
2. Errors in callbacks propagate to the next errback
3. Propagation from the errback chain to the callback chain is possible



6. PROTOCOLS (AND MORE)

`Protocols`: EVENT HANDLERS FOR A CONNECTION

- Each new connection gets a new `Protocol` *instance*
  - Basic events include:
    connection opened/closed, data available
  - Transforms wire protocol into higher level events
    (e.g. *Line* received or *HTTP request* finished)

EXAMPLE (FROM `chat.py`)

```
from twisted.internet import protocol

class NetCatChatProtocol(protocol.Protocol):
    # An instance of a Protocol exists for each established connection.

    def connectionMade(self):
        # Called when the protocol is instantiated and the connection is ready.

    def dataReceived(self, data):
        # New data (bytes) are available for consuming.

    def connectionLost(self, reason):
        # The connection is about to be terminated.

    # Has a transport property for interacting with the connection.

    # Has a factory property for interacting with the factory that build this.
```

ProtocolFactory

KEEPS *state* ACROSS Protocols

BUILDS Protocol INSTANCES

EXAMPLE (FROM `chat.py`)

```
from twisted.internet import protocol

class NetCatChatProtocol(protocol.Protocol):
    # See above.

class NetCatChatFactory(protocol.Factory):
    # By defining `protocol`, the default implementation of
    # `Factory.buildProtocol` will work fine!
    protocol = NetCatChatProtocol

    # State and other variables would be stored on the factory.
```

Transport

A WAY TO *send data*

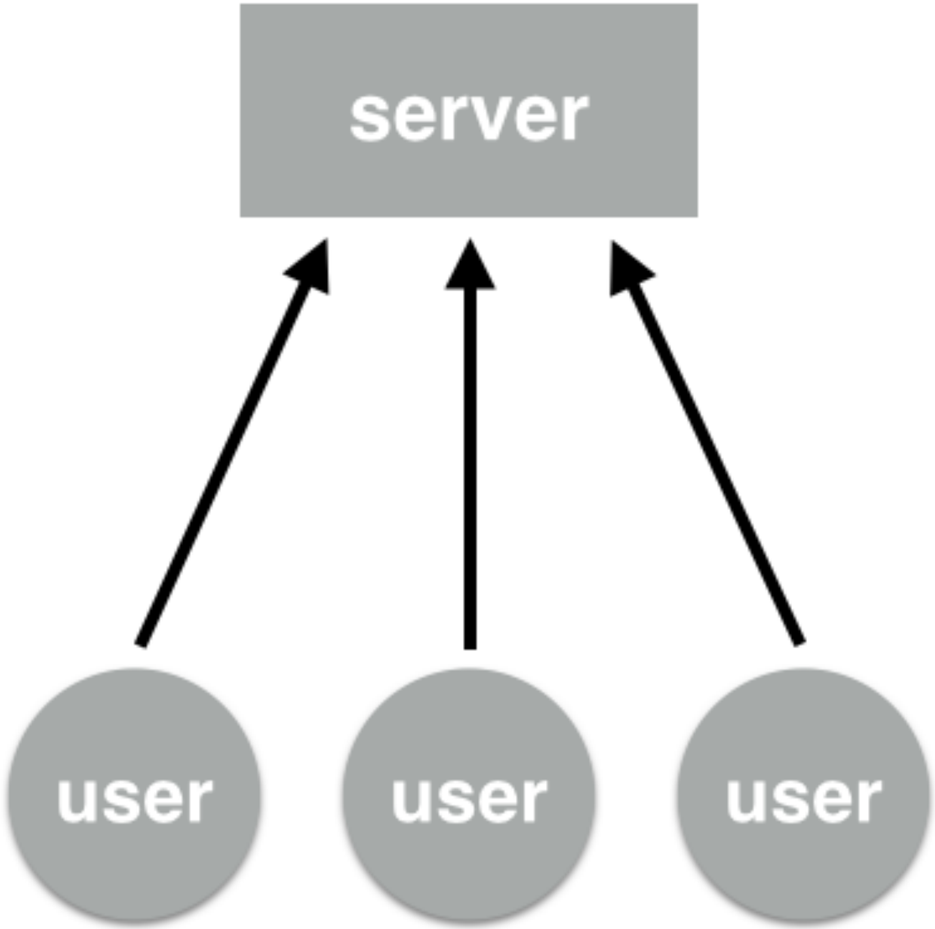
- Write *bytes* to a connection/datagram
- Close a connection
- Query local/remote addresses
- Do not assume when data will be sent

EXAMPLE (FROM `chat.py`)

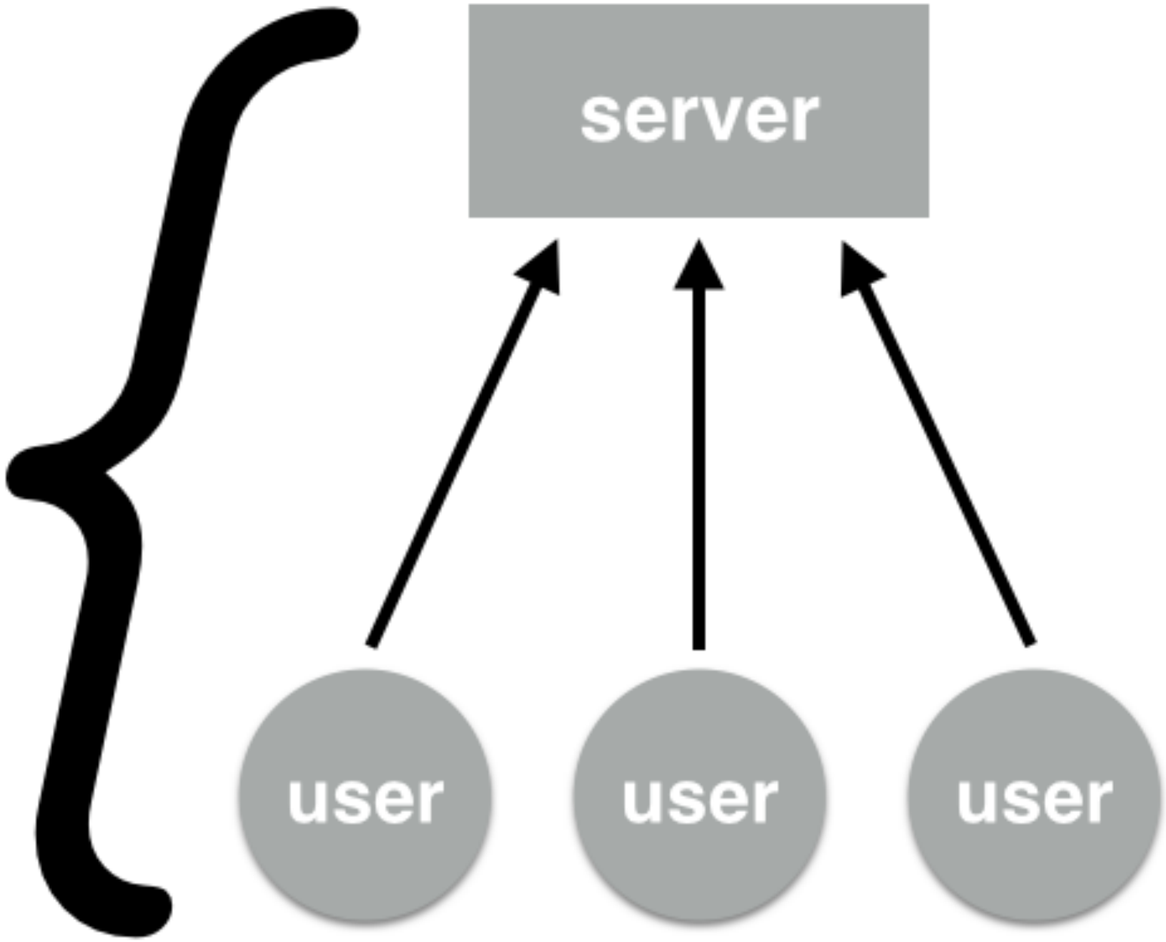
```
from twisted.internet import protocol

class NetCatChatProtocol(protocol.Protocol):
    # An instance of a Protocol exists for each established connection.

    def connectionMade(self):
        # The connection has been established, perform greetings here.
        self.transport.write(self.factory.banner)
```

DEMO

INTEGRATE TWISTED WITH OTHER SERVICES

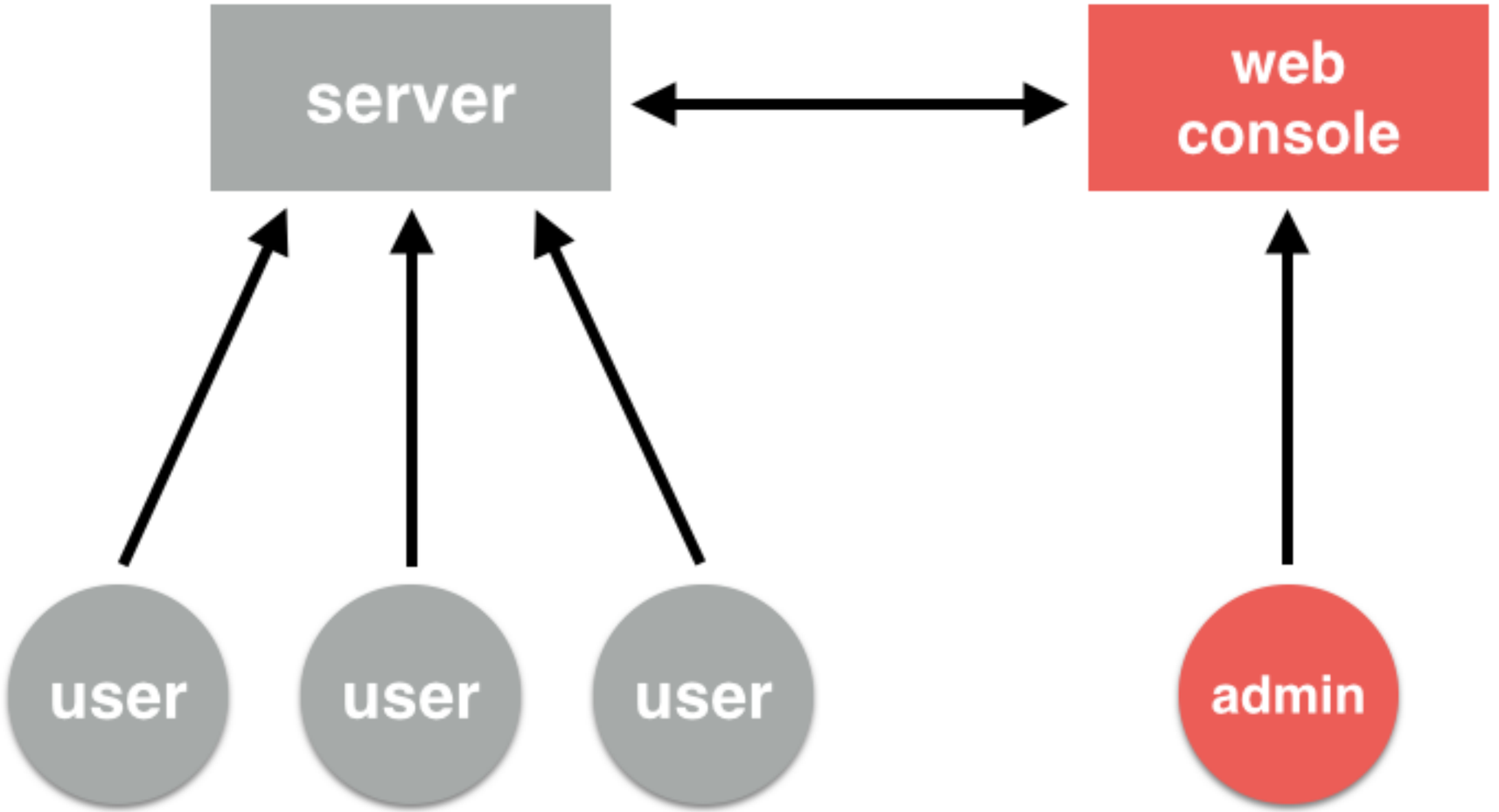

server
user
user
user

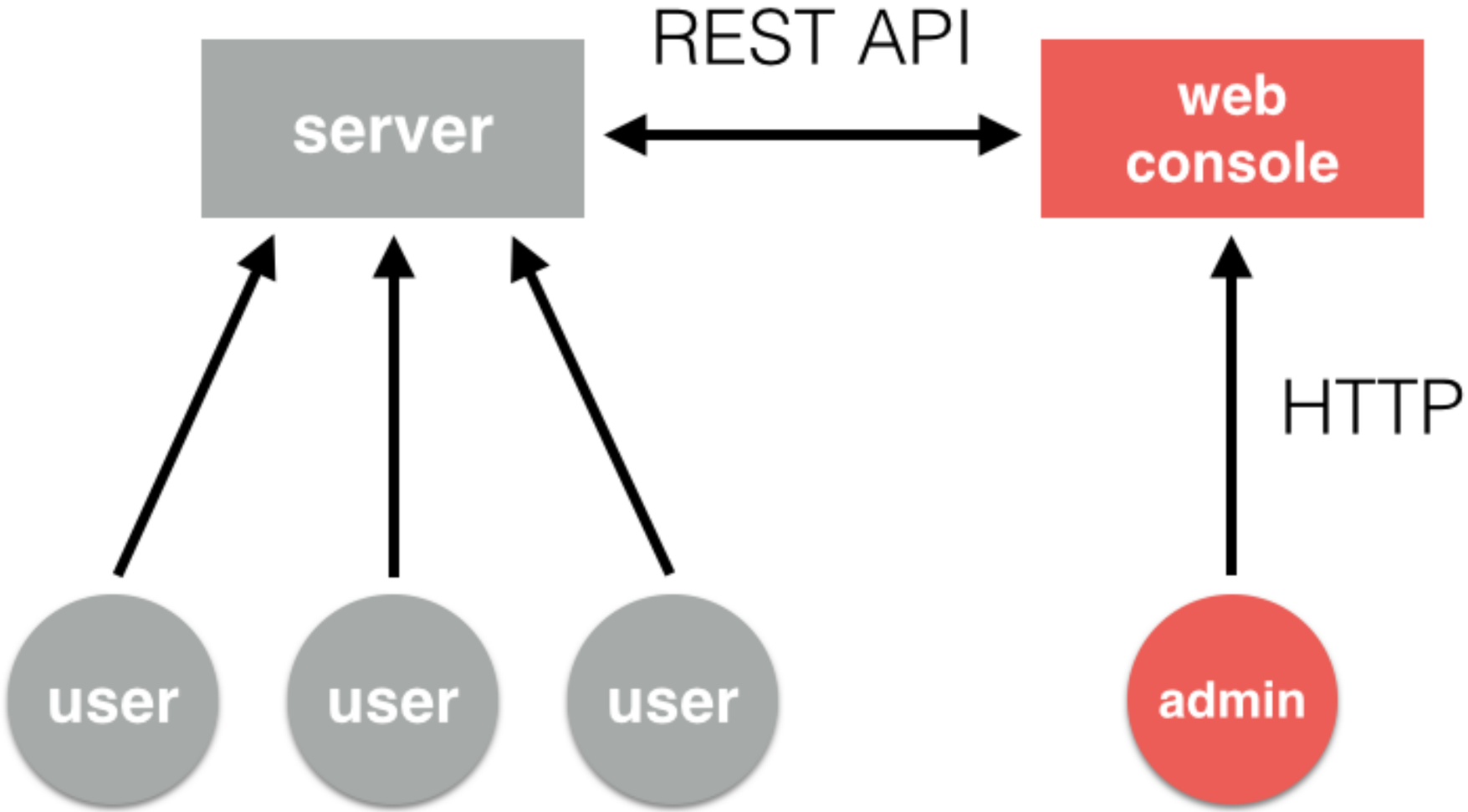
TCP
line based protocol

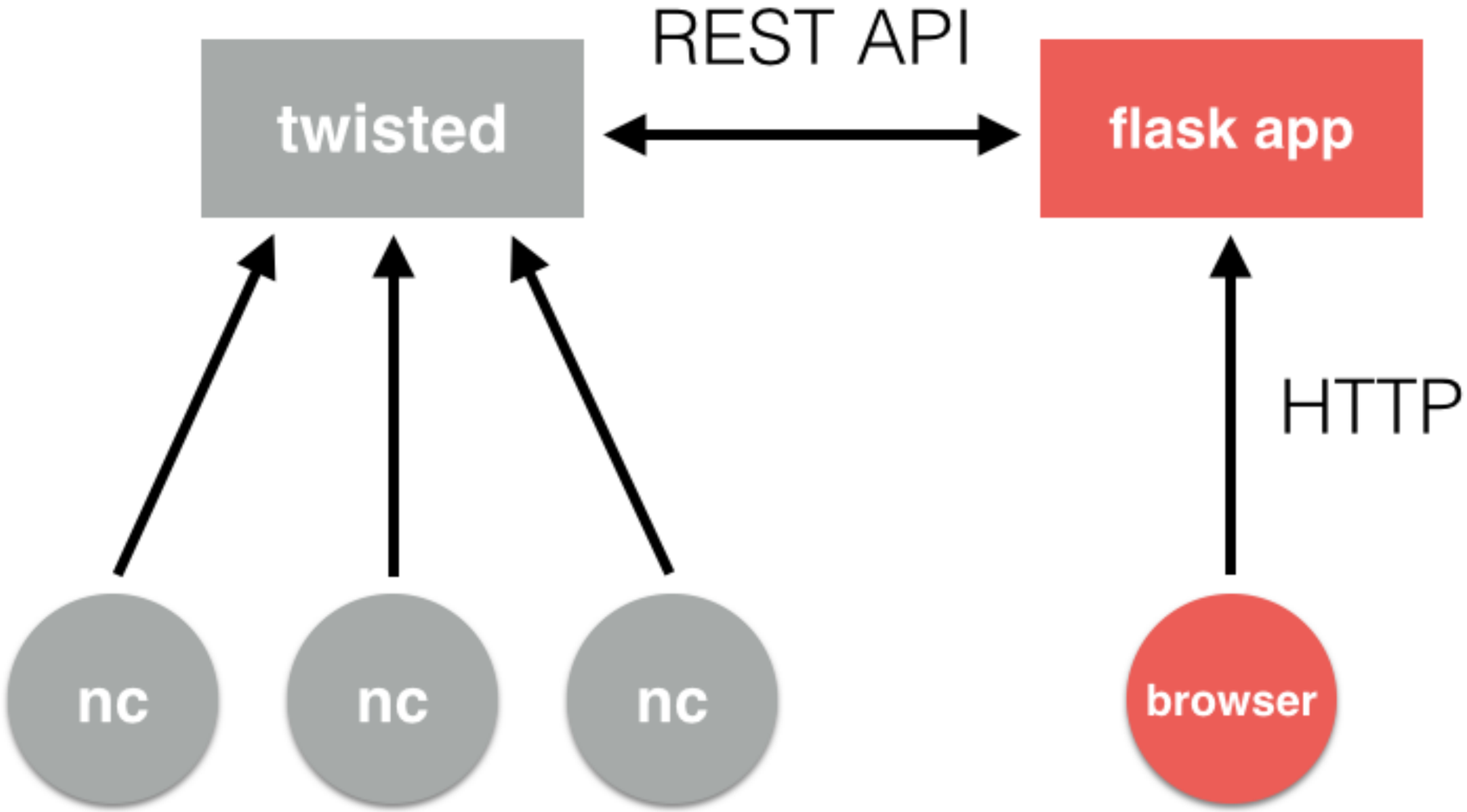


ADMIN CONSOLE

1. current user count
2. set banner


server
web
console
user
user
user
admin


REST API
server
web
console
HTTP
user
user
user
admin


REST API
twisted
flask app
HTTP
nc
nc
nc
browser

BUILD THE CHAT SERVER API

1. `GET /users/` --> returns user count
2. `GET /banner/` --> returns the current banner
3. `POST /banner/` --> sets the current banner

```
from twisted.web import resource

class ApiResource(resource.Resource):
    def __init__(self, chat_factory, *args, **kwargs):
        # This needs a reference to the NetCatChat factory object.
        self.chat_factory = chat_factory

        resource.Resource.__init__(self, *args, **kwargs)
```

USER API ENDPOINT

```
class Users(ApiResource):
    isLeaf = True

    def render_GET(self, request):
        # The user count from server.
        user_count = len(self.chat_factory.clients)
        result = {'users': user_count}

        return json.dumps(result, indent=4, separators=(',', ': ')) + "\n"
```

BANNER API ENDPOINT

```
class Banner(ApiResource):
    isLeaf = True

    def render_GET(self, request):
        # Get the banner.
        result = {'banner': self.chat_factory.banner}
        return json.dumps(result, indent=4, separators=(',', ': ')) + "\n"
```

```
def _set_banner(self, banner):
    # ... error handling ;-)
    self.chat_factory.banner = bytes(banner)  # unicode to bytes

def render_POST(self, request):
    # Set the banner.
    status = "ERROR"
    try:
        content = request.content.read()
        data = json.loads(content)['banner']

        # Make this a Deferred so the function can immediately return.
        d = task.deferLater(reactor, 0.1, self._set_banner, data)
        d.addErrback(log.err)

        status = "SUCCESS"
    except Exception:
      # ... error handling ;-)

    return json.dumps({'status': status})
```

UPDATE REACTOR TO EXPOSE API

```
from twisted.internet import reactor, endpoints
from twisted.web import server

from chat import NetCatChatFactory
from api import Root

# Create an instance of the factories.
factory = NetCatChatFactory()
site = server.Site(Root(factory))

# Listen on TCP port 1400 for chat and port 8080 for the API.
endpoints.serverFromString(reactor, "tcp:1400").listen(factory)
endpoints.serverFromString(reactor, "tcp:8080").listen(site)

# Start listening for connections (and run the event-loop).
reactor.run()
# Note that any code after this point will *not* be executed. reactor.run enters
# an infinite loop until shutdown.
```

WHAT ABOUT THE CLIENT CODE?

API CLIENT CODE / USER COUNT

```
def get_user_count():
    url = API_URL + '/users/'
    try:
        r = requests.get(url)
    except Exception as e:  # Gotta catch 'em all!
        return None

    result = r.json()
    return result['users']
```

API CLIENT CODE / GET BANNER

```
def get_banner():
    url = API_URL + '/banner/'
    try:
        r = requests.get(url)
    except Exception as e:  # Gotta catch 'em all!
        return None

    result = r.json()
    return result['banner']
```

API CLIENT CODE / SET BANNER

```
def set_banner(banner):
    url = API_URL + '/banner/'
    data = json.dumps({'banner': banner})
    try:
        r = requests.post(url, data=data)
    except Exception as e:  # Gotta catch 'em all!
        return None
```

NOW SERVE IT USING

FLASK

```
app = Flask(__name__)
API_URL = 'http://127.0.0.1:8080'

@app.route("/")
def splash():
    user_count = get_user_count()
    banner = get_banner()
    return render_template('index.html', user_count=user_count, banner=banner)


@app.route('/set_banner/', methods=['POST'])
def banner():
    banner = request.values.get('banner', "")
    set_banner(banner)
    return redirect(url_for('splash'))


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

TANGENT: `twistd`

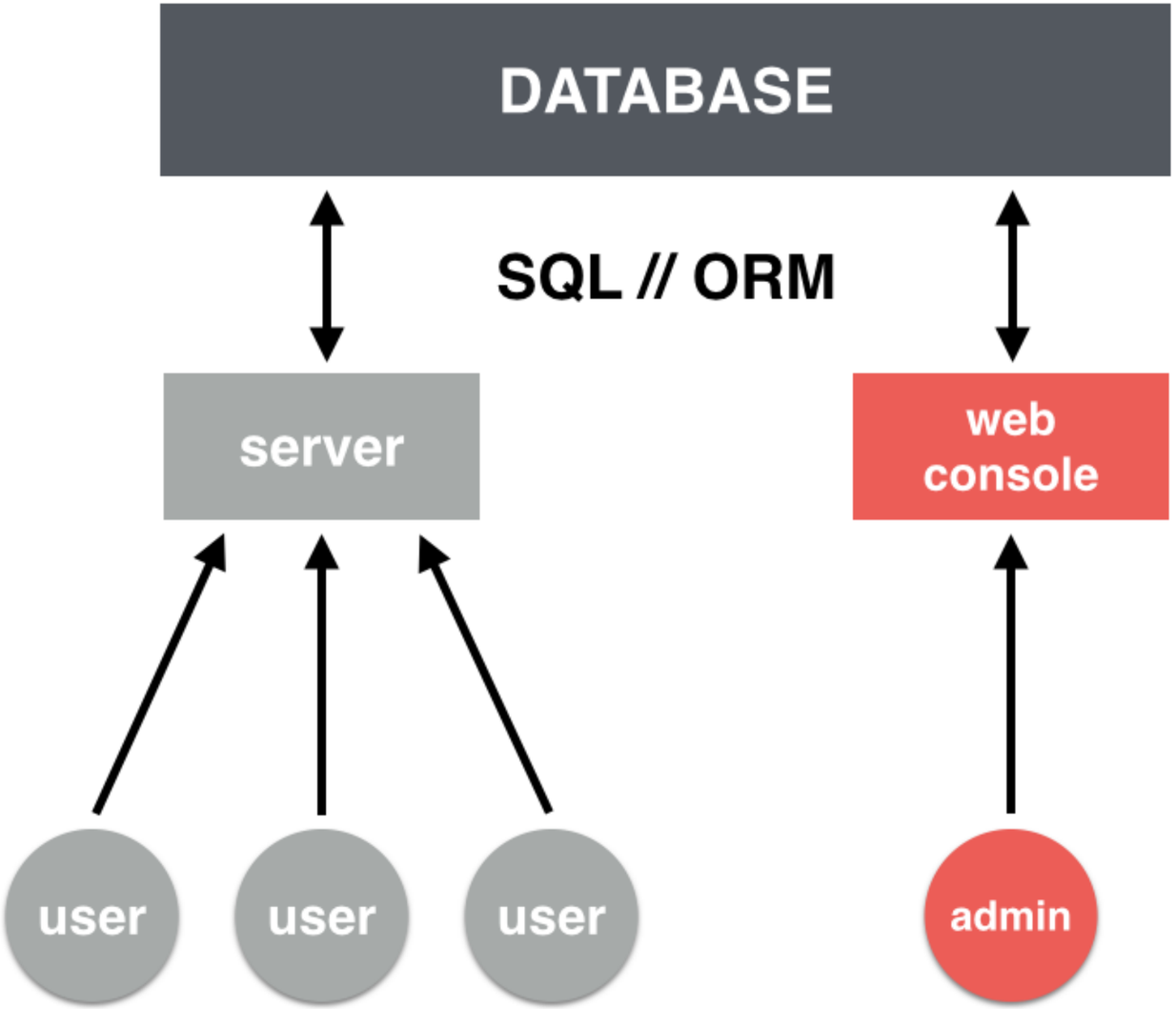
`twistd` is a daemon that helps run applications.

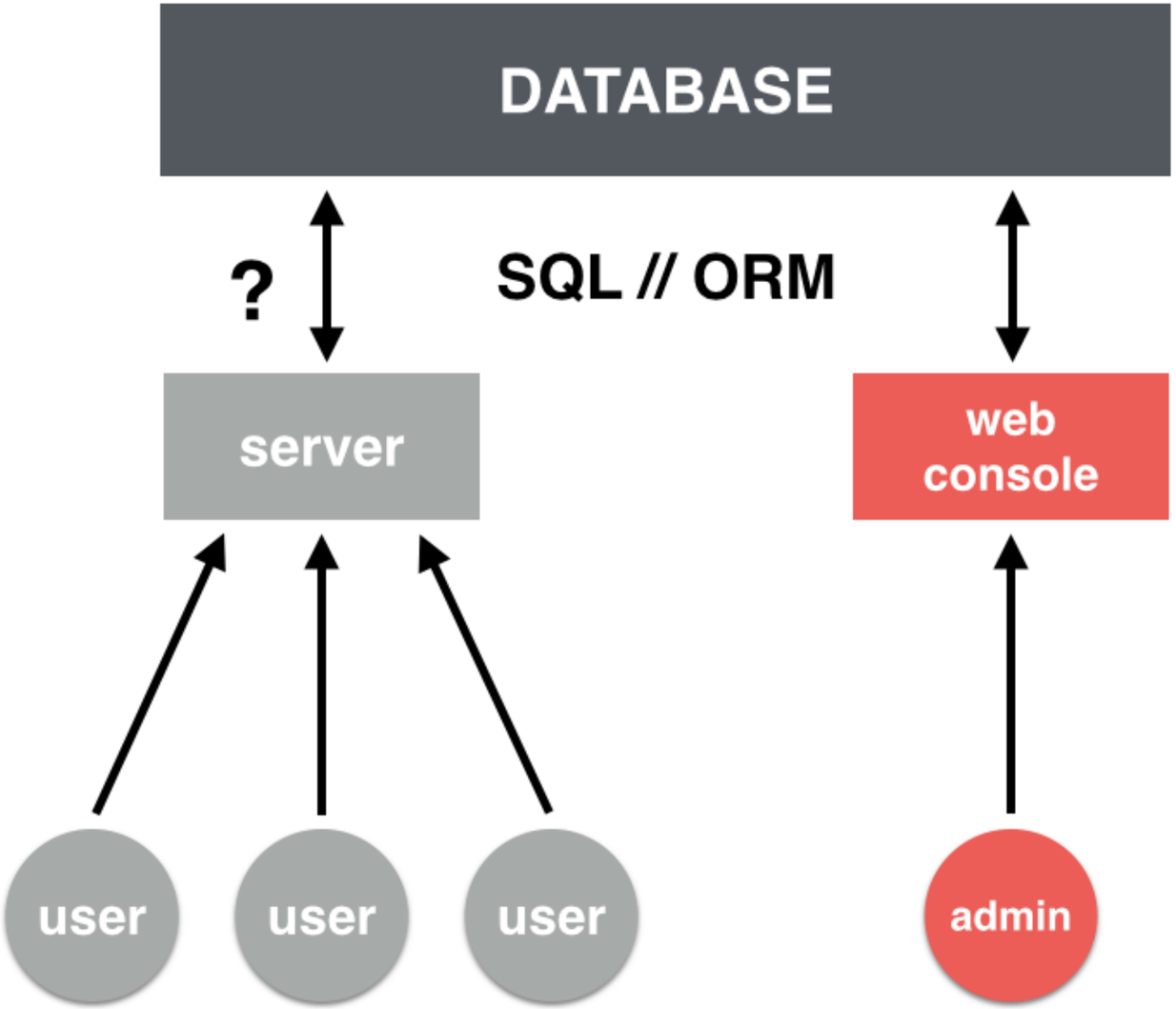
```
twistd web --wsgi=dashboard.app
```

Be careful when running as a service under upstart/systemd/init!

DEMO

WHAT ABOUT DATA PERSISTENCE?


DATABASE
SQL // ORM
server
web
console
user
user
user
admin

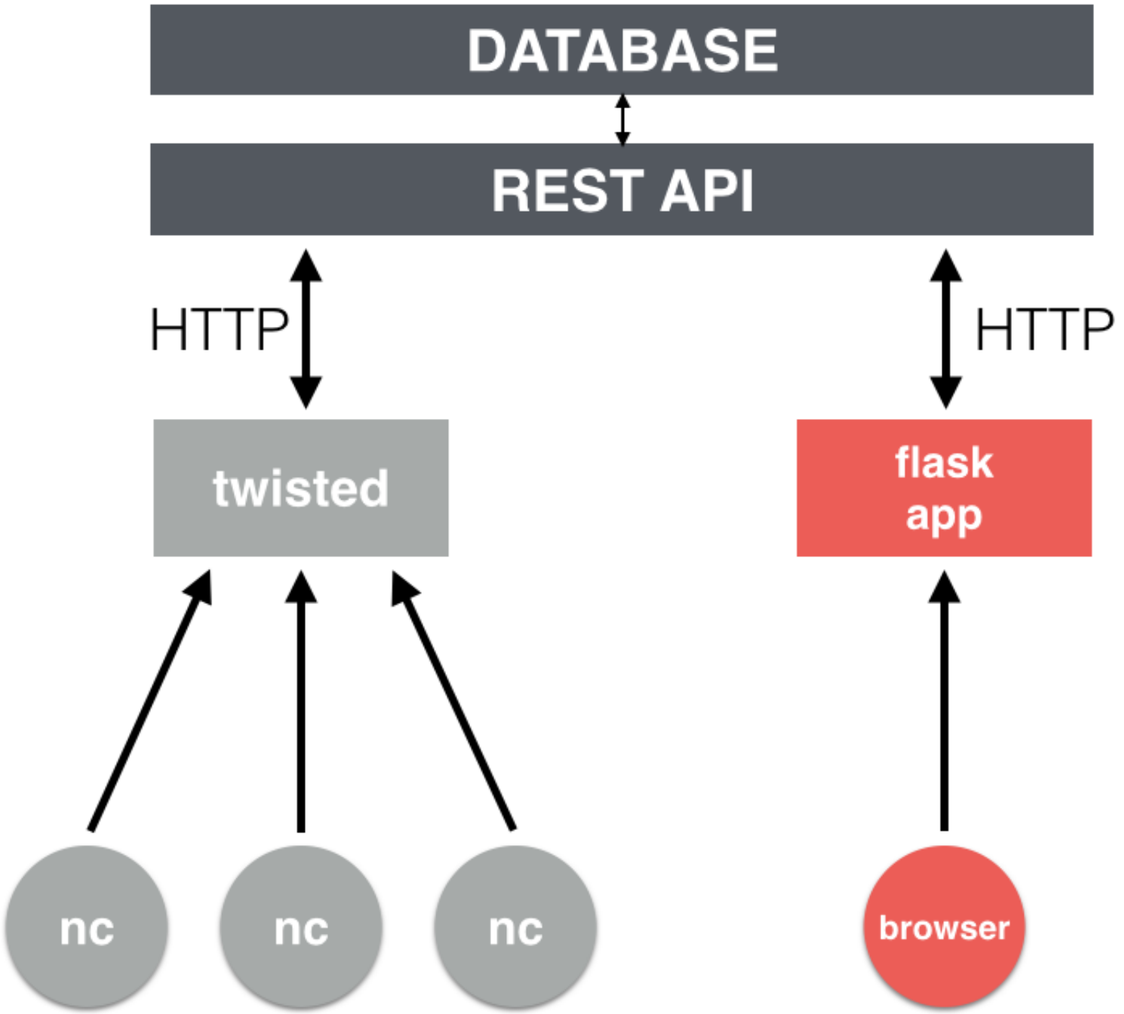

DATABASE
?
SQL // ORM
server
web
console
user
user
user
admin

1. `twisted.enterprise.adbapi`
2. SQL Alchemy
3. Django ORM

...

CAREFUL NOT TO BLOCK!

IS THERE A BETTER, MORE REUSABLE WAY?


DATABASE
REST API
HTTP
HTTP
twisted
flask
app
nc
nc
nc
browser

HOW WOULD YOU SCALE TO MANY TWISTED SERVERS?

- focus on keeping state in one place
- make Twisted services stateless
- load balance accordingly

RECAP

- what is async programming
  - what is Twisted
- when/why to use Twisted

TWISTED RECAP

- event loop (reactor)
- deferreds
- protocols (factories, transports)
- using Twisted to build larger systems/services

WHAT TO RESEARCH NEXT

- trial: testing, the twisted way
- inline callbacks: synchronous-looking deferreds in twisted
- `twistd`: running twisted code as a service
- read `http://krondo.com/?page_id=1327`

THANK YOU - QUESTIONS?

GITHUB.COM/PERCIPIENT/TALKS

patrick@percipientnetworks.com
stephen@percipientnetworks.com

(btw, we are sponsoring DjangoCon! Find us!)