



The Internet

Chris Piech and Mehran Sahami
CS106A, Stanford University

First, a cool demo

The screenshot shows a Mac OS X desktop with two windows open. On the left is a "Chat Client" application window titled "index.html". It has a "Send" button and a text input field containing "The internet is a wild place...". Below it is a "Messages" section with a "Refresh" button and a list of messages from users Chris, Laura, and Terry. On the right is a terminal window titled "backend — Python chat_server.py — 93x34" showing a log of JSON command objects being processed by the server. The log includes various "getMsgs" and "newMsg" commands with parameters like user names and message content.

```
Chris@ndoto backend % python chat_server.py
Server running...
[{"command": "getMsgs", "params": {"index": "0"}}, {"command": "newMsg", "params": {"msg": "Hello world?", "user": "Chris"}}, {"command": "getMsgs", "params": {"index": "0"}}, {"command": "getMsgs", "params": {"index": "0"}}, {"command": "newMsg", "params": {"msg": "Here I am!!!", "user": "Laura"}}, {"command": "getMsgs", "params": {"index": "1"}}, {"command": "newMsg", "params": {"msg": "This is fun!", "user": "Laura"}}, {"command": "getMsgs", "params": {"index": "2"}}, {"command": "getMsgs", "params": {"index": "1"}}, {"command": "newMsg", "params": {"msg": "Wahooooo :-)", "user": "Chris"}}, {"command": "getMsgs", "params": {"index": "3"}}, {"command": "newMsg", "params": {"msg": "We are on the internet...", "user": "Chris"}}, {"command": "getMsgs", "params": {"index": "4"}}, {"command": "newMsg", "params": {"msg": "This is like low-budget WhatsApp", "user": "Chris"}}, {"command": "getMsgs", "params": {"index": "5"}}, {"command": "getMsgs", "params": {"index": "3"}}, {"command": "getMsgs", "params": {"index": "6"}}, {"command": "getMsgs", "params": {"index": "6"}}, {"command": "getMsgs", "params": {"index": "6"}}, {"command": "newMsg", "params": {"msg": "But we made it, which is cool.", "user": "Laura"}}, {"command": "getMsgs", "params": {"index": "6"}}, {"command": "getMsgs", "params": {"index": "0"}}, {"command": "newMsg", "params": {"msg": "Hi everyone! Terry here too", "user": "Terry"}}, {"command": "getMsgs", "params": {"index": "7"}}, {"command": "newMsg", "params": {"msg": "Hi Terry!", "user": "Laura"}}, {"command": "getMsgs", "params": {"index": "7"}}, {"command": "getMsgs", "params": {"index": "8"}}, {"command": "newMsg", "params": {"msg": "The internet is a wild place...", "user": "Terry"}}, {"command": "getMsgs", "params": {"index": "9"}}, {"command": "getMsgs", "params": {"index": "9"}}
```



Search 

- scientists confirm chinese horseshoe bats responsible sars
- scientists confirm chinese bats responsible sars outbreak
- sixth person dies sars like virus
- new sars like virus shows person person transmission
- issues alert sars type infection
- fuel prices bigger problem sep 11 sars
- sars expert dies
- royal society inducts sars expert
- sars probably originated bats
- bats may source sars

Search Engine



<review>

Classes Review

Dog.py

```
class Dog:  
    def __init__(self):  
        self.times_barked = 0  
  
    def bark():  
        print('woof')  
        self.times_barked += 1
```

life.py

```
def main():  
    simba = Dog()  
    juno = Dog()  
  
    simba.bark()  
    juno.bark()  
    simba.bark()  
  
    print(simba.__dict__)  
    print(juno.__dict__)
```



Classes Review

Dog.py

```
class Dog:  
    def __init__(self):  
        self.times_barked = 0  
  
    def bark():  
        print('woof')  
        self.times_barked += 1
```

life.py

```
def main():  
    simba = Dog()  
    juno = Dog()  
  
    simba.bark()  
    juno.bark()  
    simba.bark()  
  
    print(simba.__dict__)  
    print(juno.__dict__)
```

1. What happens when you make a **new** one?



Classes Review

Dog.py

```
class Dog:  
    def __init__(self):  
        self.times_barked = 0  
  
    def bark():  
        print('woof')  
        self.times_barked += 1
```

life.py

```
def main():  
    simba = Dog()  
    juno = Dog()  
  
    simba.bark()  
    juno.bark()  
    simba.bark()  
  
    print(simba.__dict__)  
    print(juno.__dict__)
```

2. What **variables** does each instance store?



Classes Review

Dog.py

```
class Dog:  
    def __init__(self):  
        self.times_barked = 0  
  
    def bark():  
        print('woof')  
        self.times_barked += 1
```

life.py

```
def main():  
    simba = Dog()  
    juno = Dog()  
  
    simba.bark()  
    juno.bark()  
    simba.bark()  
  
    print(simba.__dict__)  
    print(juno.__dict__)
```

2. What **methods** can you call on an instance?



Classes Review

Dog.py

```
class Dog:  
    def __init__(self):  
        self.times_barked = 0  
  
    def bark():  
        print('woof')  
        self.times_barked += 1
```

life.py

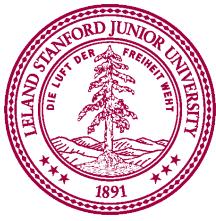
```
def main():  
    simba = Dog()  
    juno = Dog()  
  
    simba.bark()  
    juno.bark()  
    simba.bark()  
  
    print(simba.__dict__)  
    print(juno.__dict__)
```

Did I mention that a class is like a fancy dictionary?



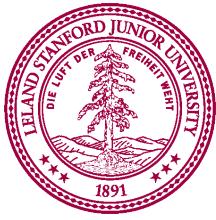


Classes define new variable
types





Classes decompose your
program across files



Define New Variable Types

Song

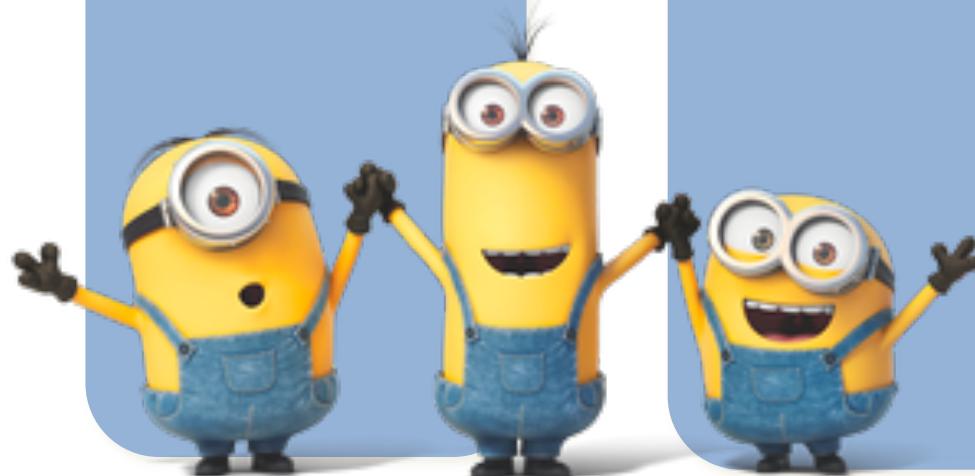
Playlist

User



Song Player

Song Retriever



</ review>

One reason programming is
fun is because of the
internet...

Smart Phone Access

Advanced Economies



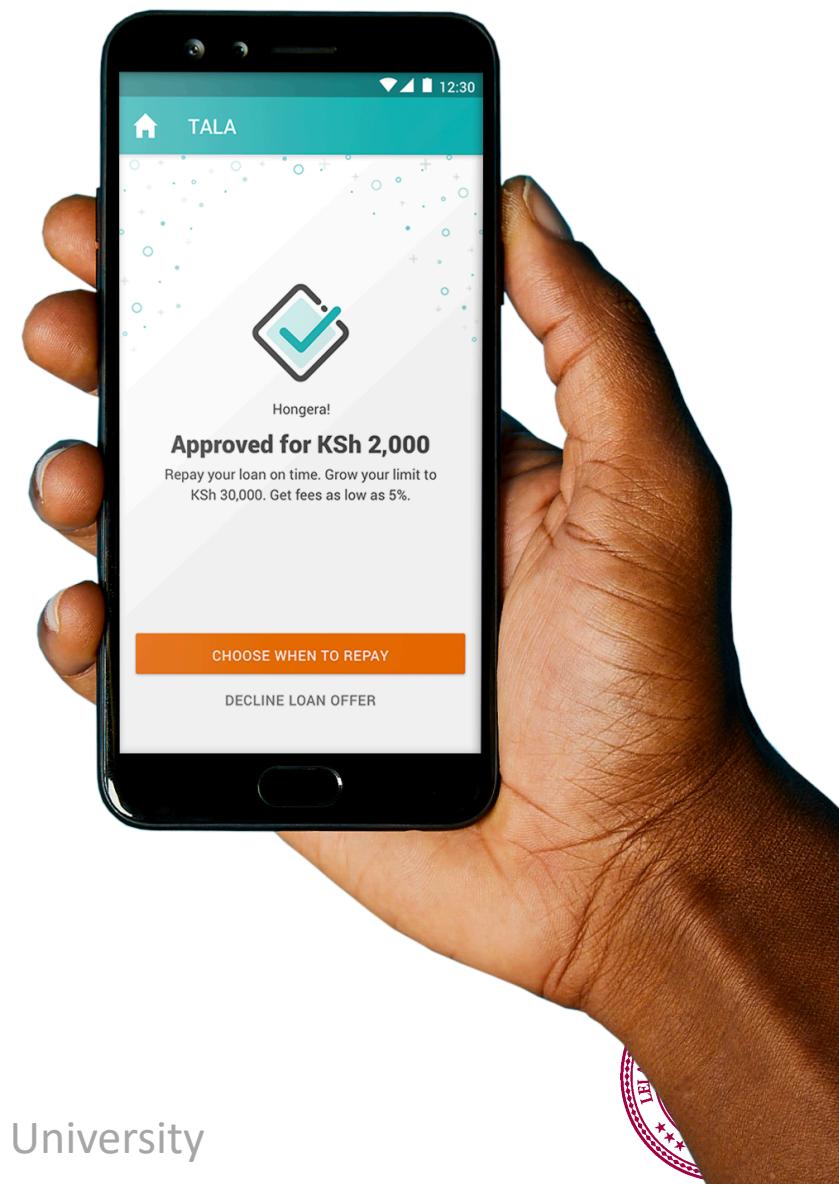
Emerging Economies



Smartphone

Mobile

No phone



For the third time ever in
CS106A:

Learning Goals

1. Write a program that can respond to internet requests

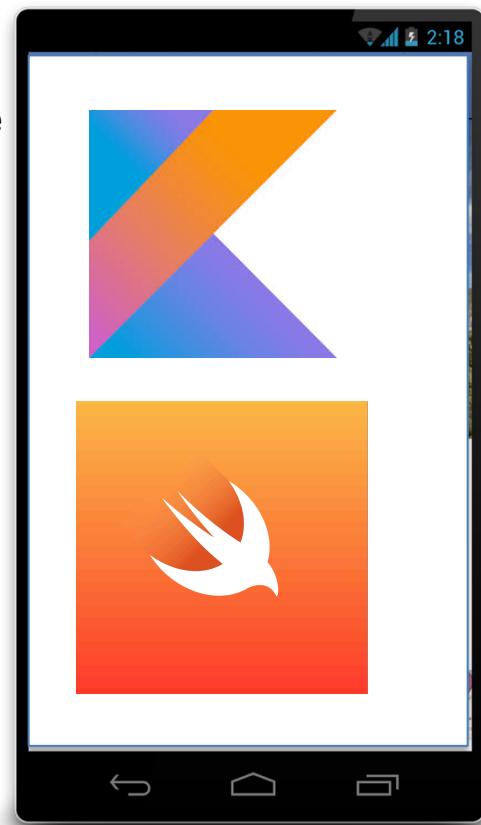


How does your phone
communicate with facebook?

The program on your phone
talks to the program at
Facebook

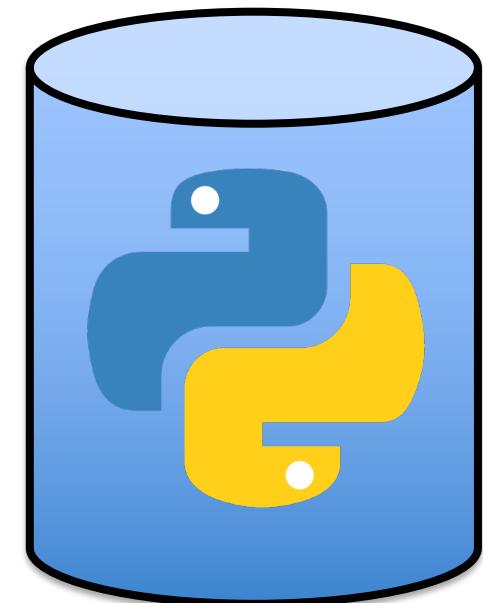
Face Book Server

JavaScript
with HTML
are the
languages
of websites



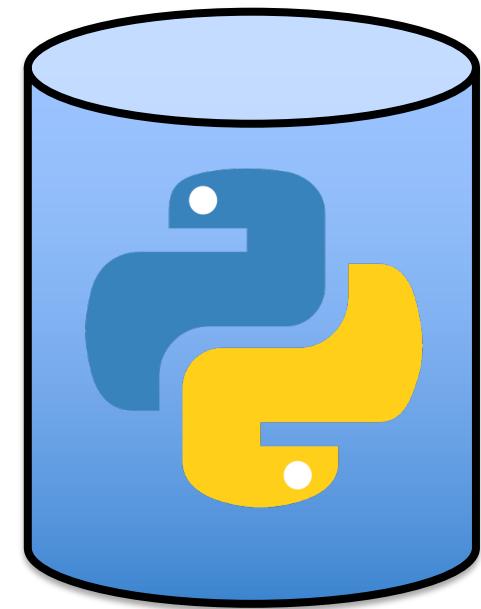
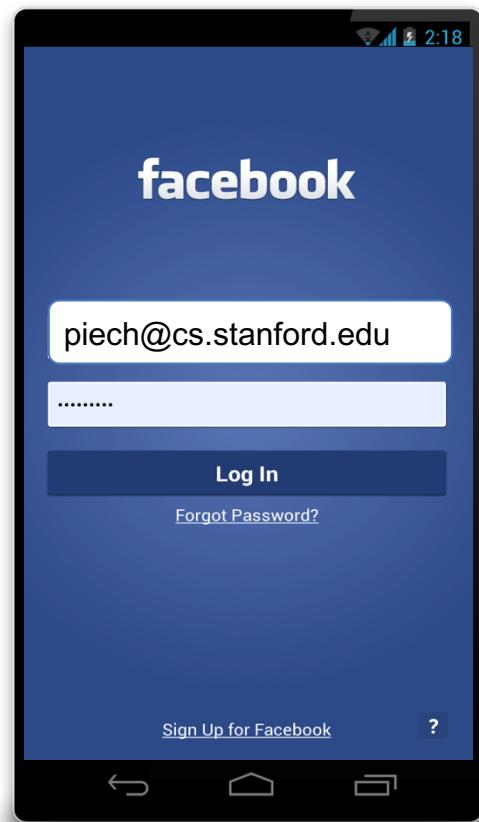
Kotlin is the
language
of Android
phones

Swift is the
language
of Apple
phones



Face Book Server

Is this legit?

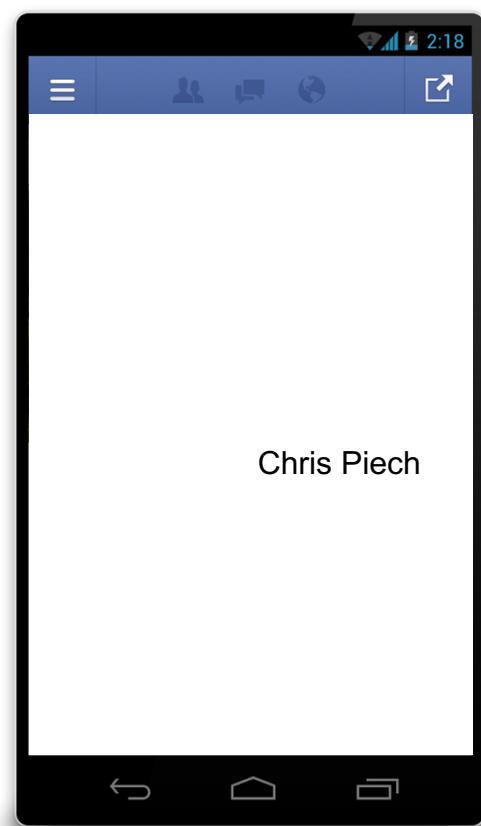


piech@cs.stanford.edu
is now logged in



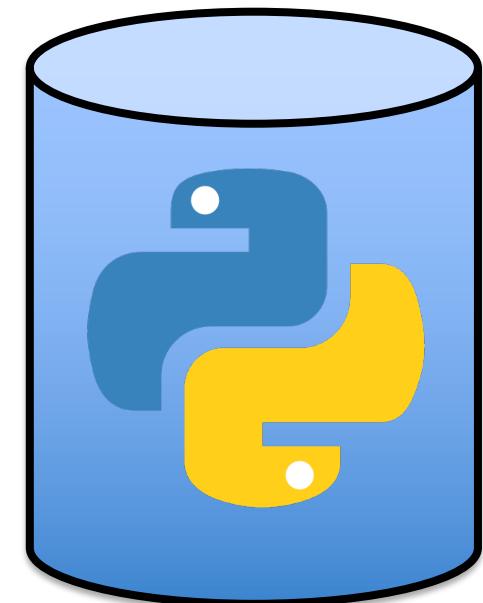
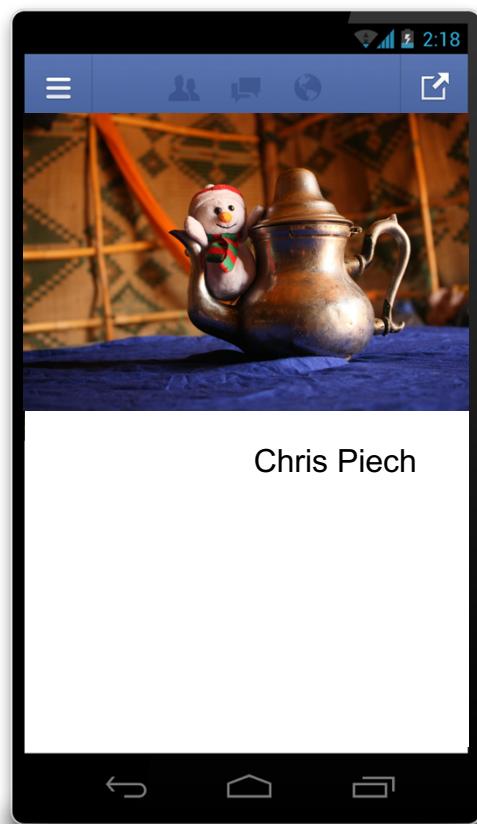
Face Book Server

Send me the **full name** for
piech@cs.stanford.edu



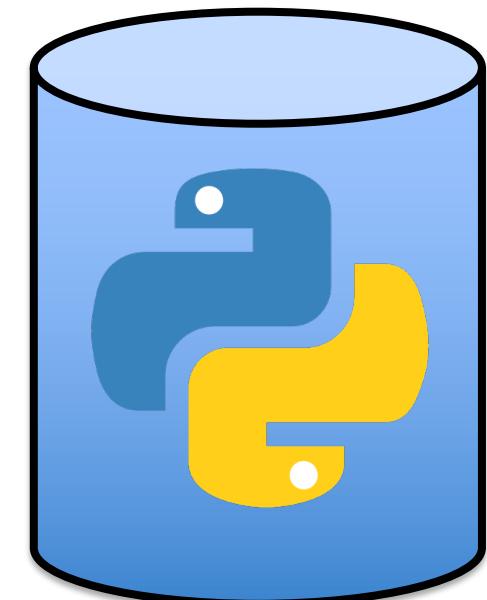
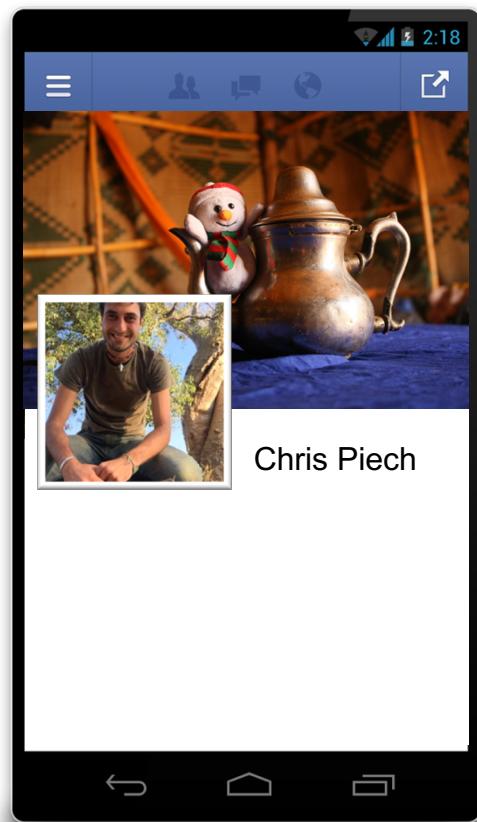
Face Book Server

Send me the **cover photo** for
piech@cs.stanford.edu



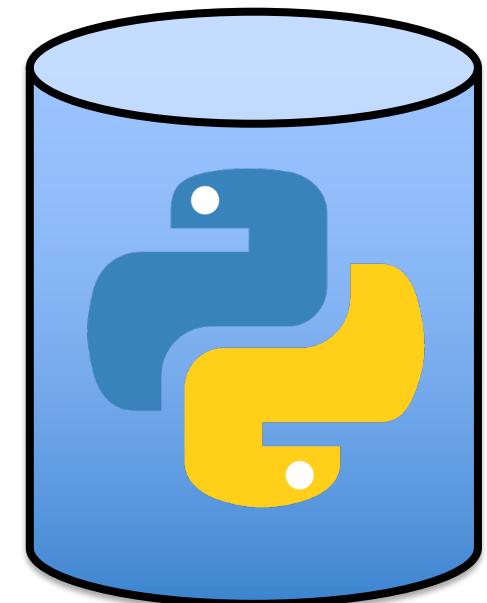
Face Book Server

Send the **profile photo** for
piech@cs.stanford.edu

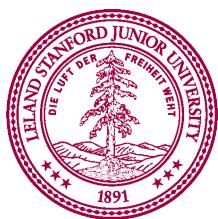


Face Book Server

Send the **status** for
piech@cs.stanford.edu

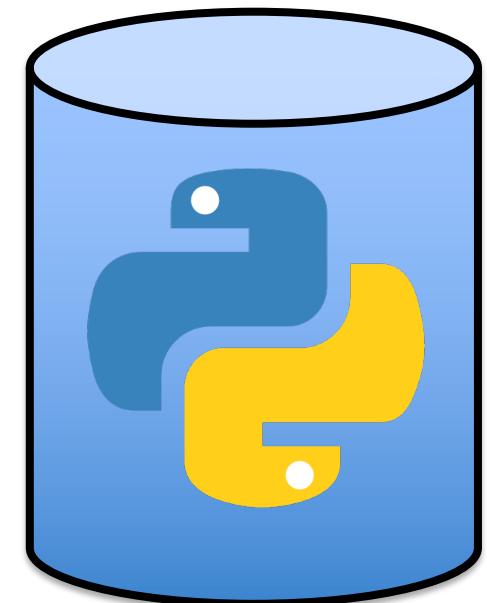
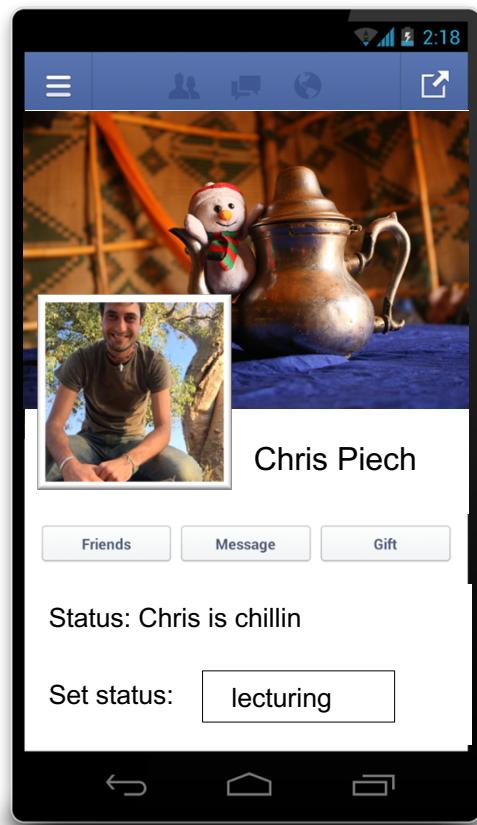


“chillin”



Face Book Server

Set the **status** for
piech@cs.stanford.edu
to be "**lecturing**"

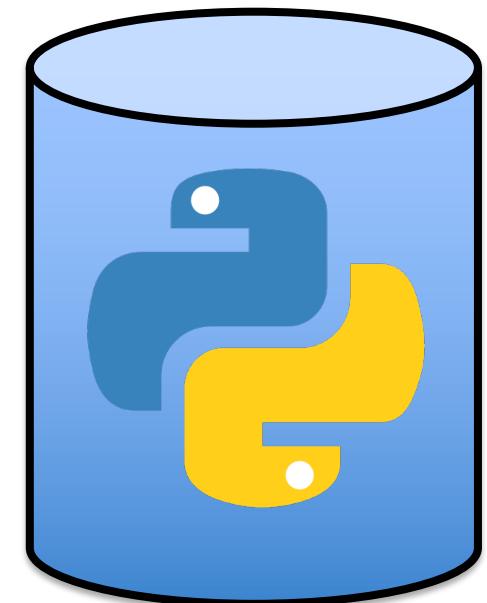


"success"

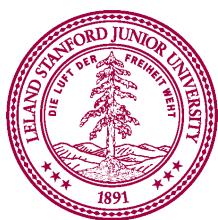


Face Book Server

Send me the **status** for
piech@cs.stanford.edu



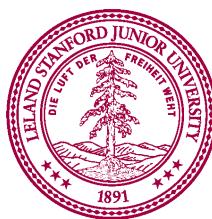
“lecturing”



Background: The Internet



The internet is just many programs sending messages (as *Strings*)



Background: The Internet



The internet is just many programs sending messages (as *Strings*)



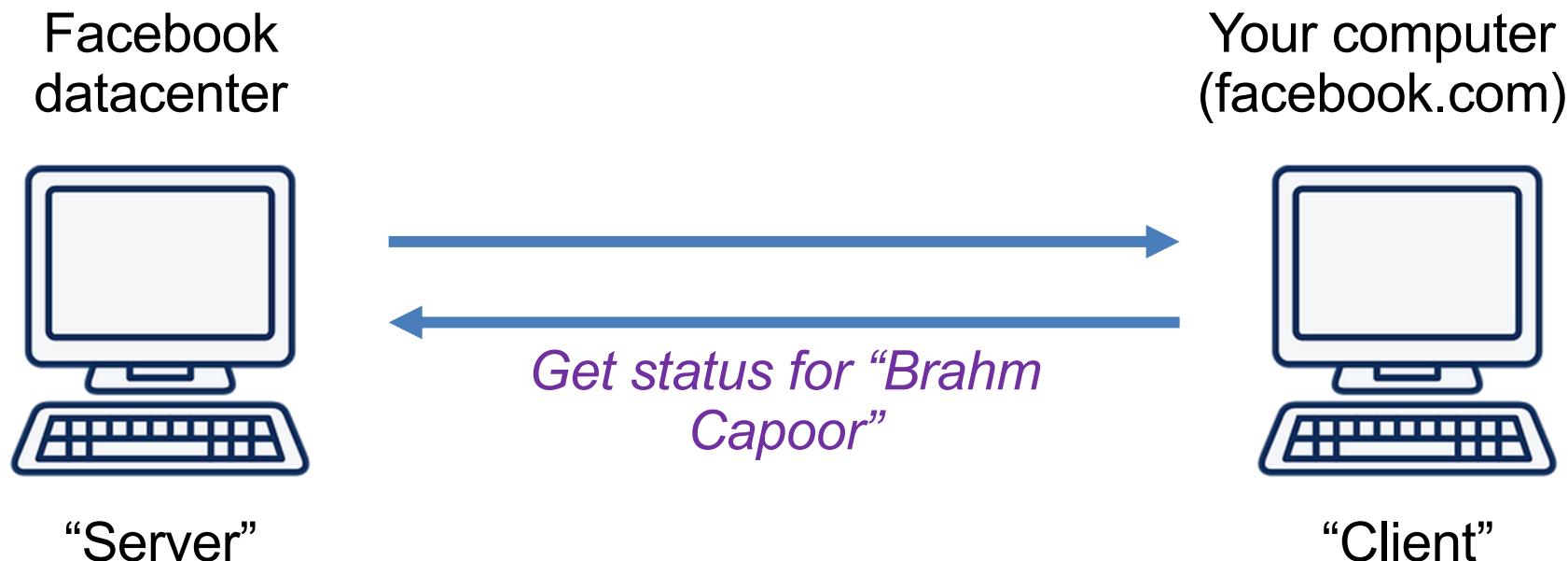
Background: The Internet



The internet is just many programs sending messages (as *Strings*)



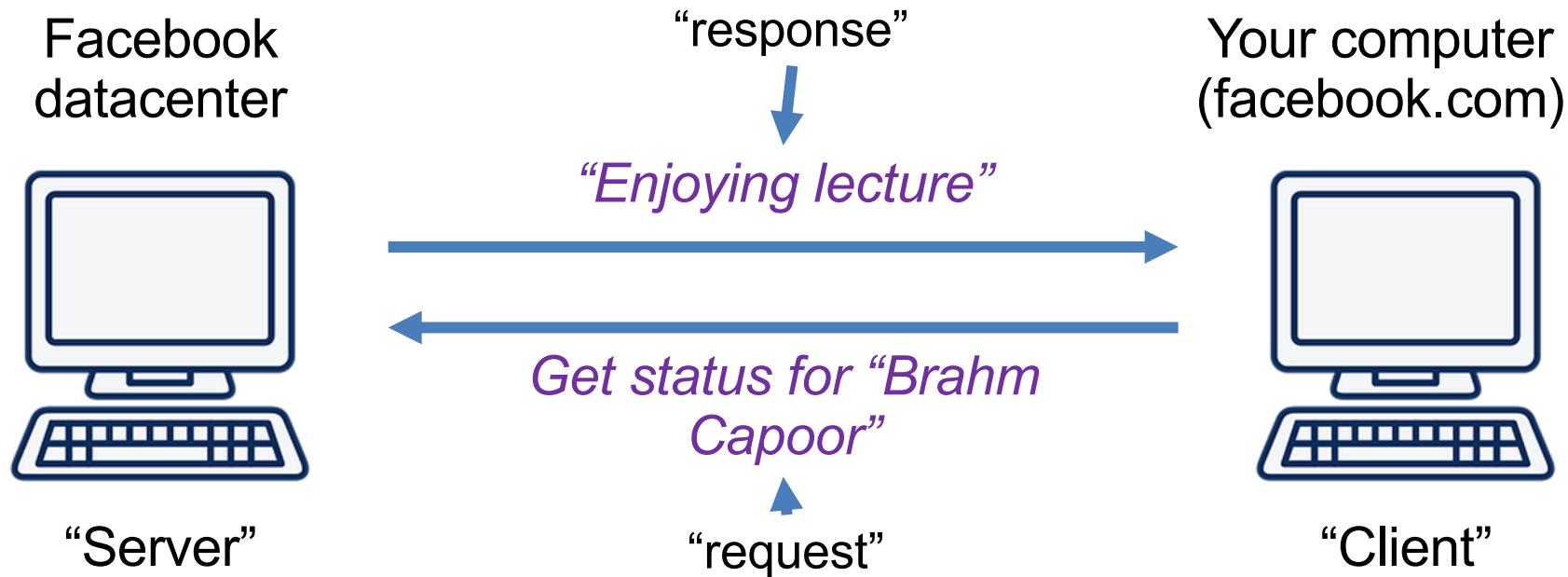
Background: The Internet



The internet is just many programs sending messages (as *Strings*)



Background: The Internet



The internet is just many programs sending messages (as *Strings*)



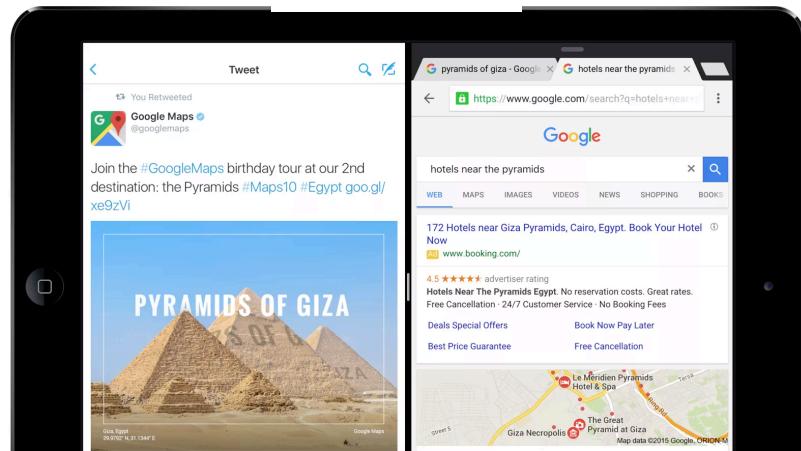


There are two types of
internet programs. Servers
and Clients



Internet 101

Computers on the internet

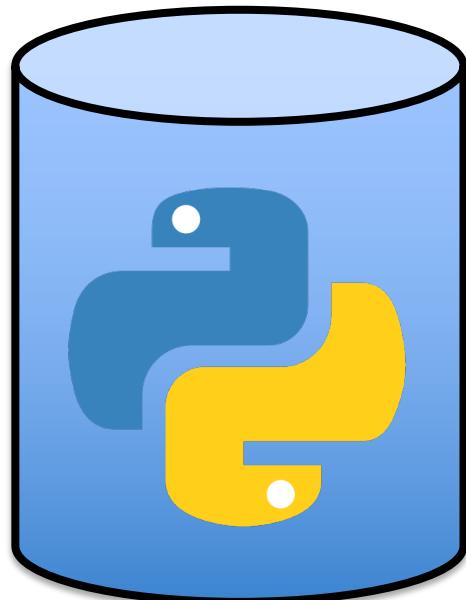


, CS106A, Stanford University



Servers are computers (running code)

Face Book Server



=



Facebook's closest datacenter is here

I am here



Google



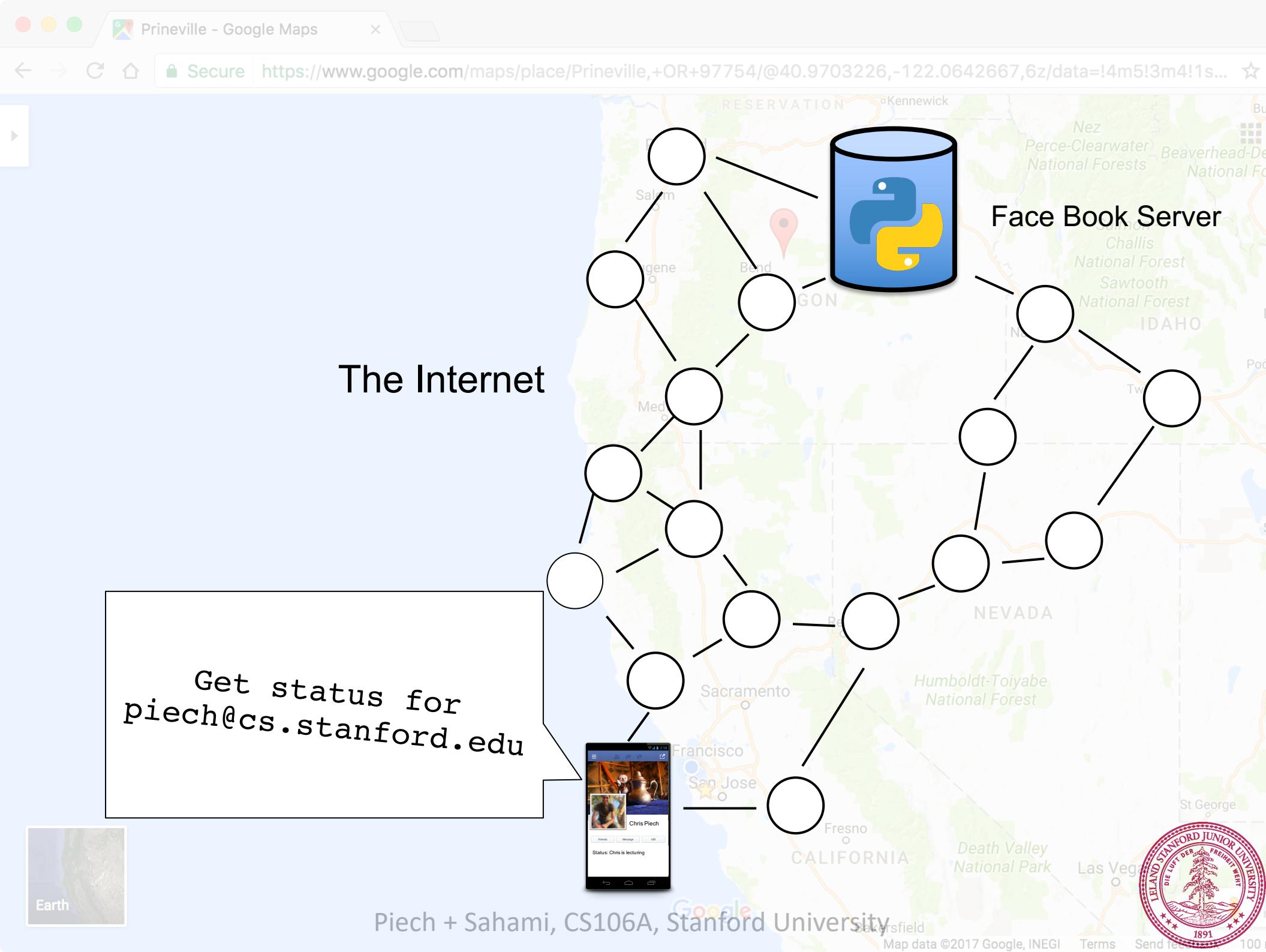
The Internet



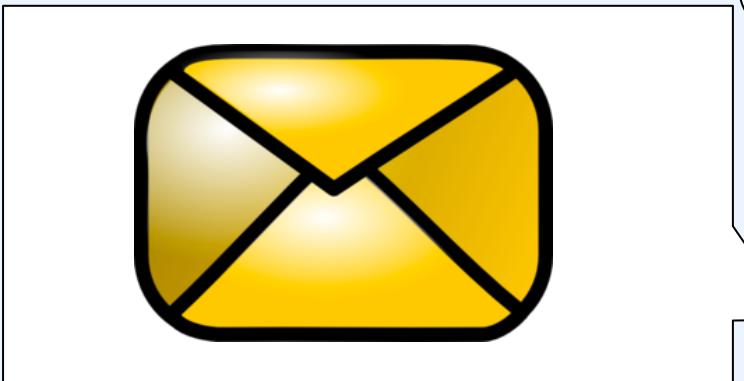
Piech + Sahami, CS106A, Stanford University

Map data ©2017 Google, INEGI

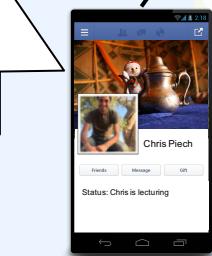




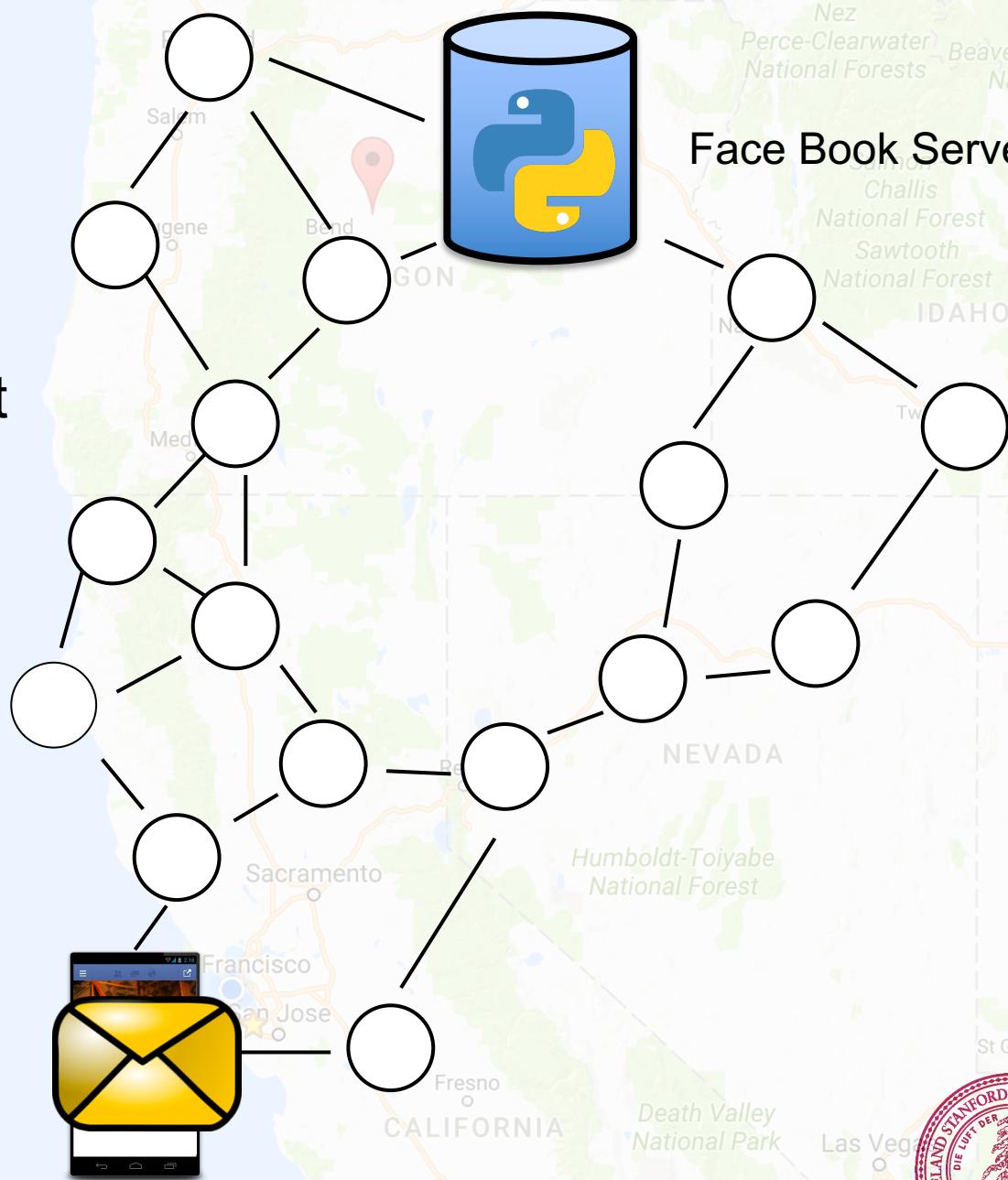
The Internet



Piech + Sahami, CS106A, Stanford University



The Internet



teaching

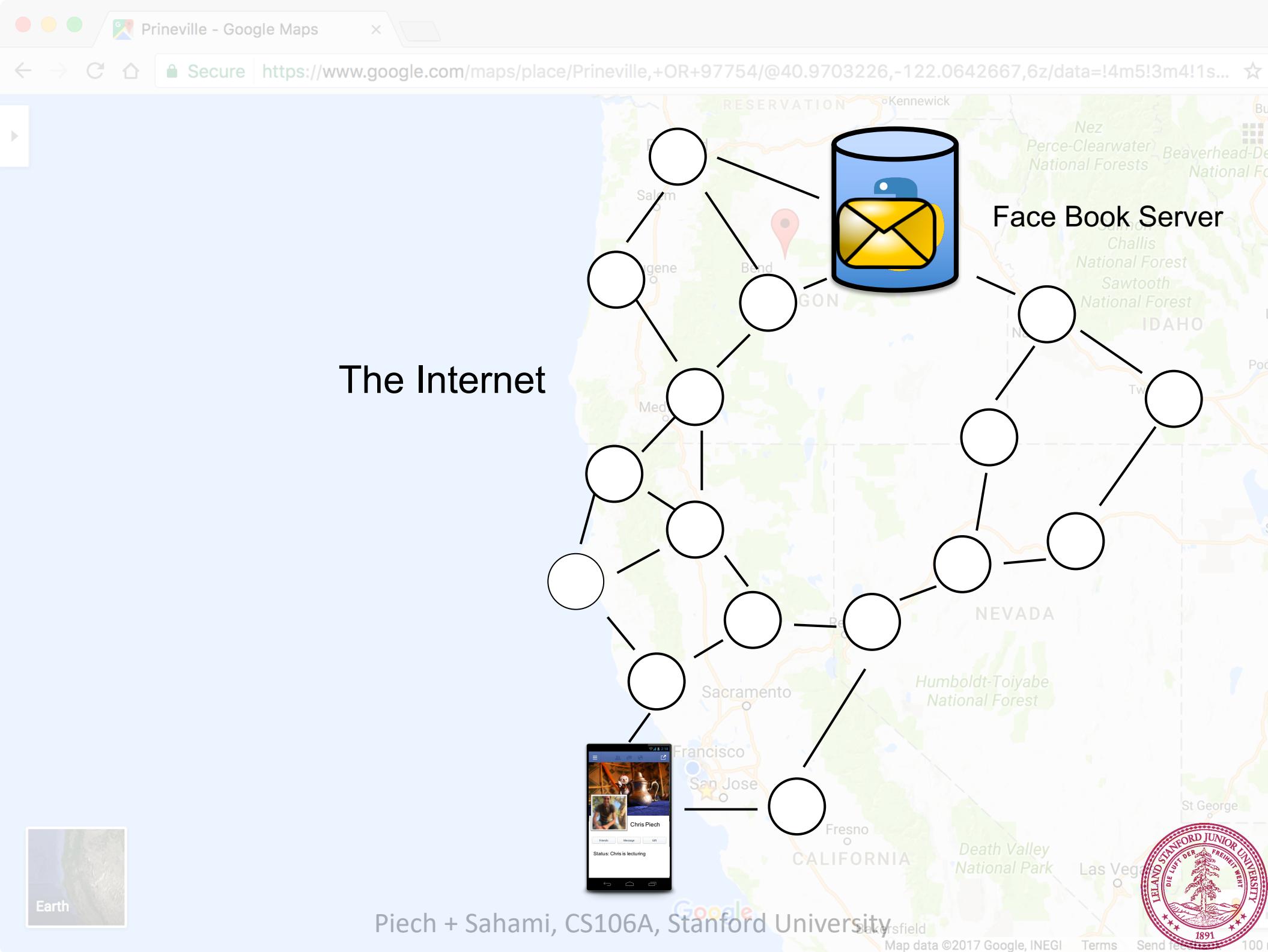
The Internet





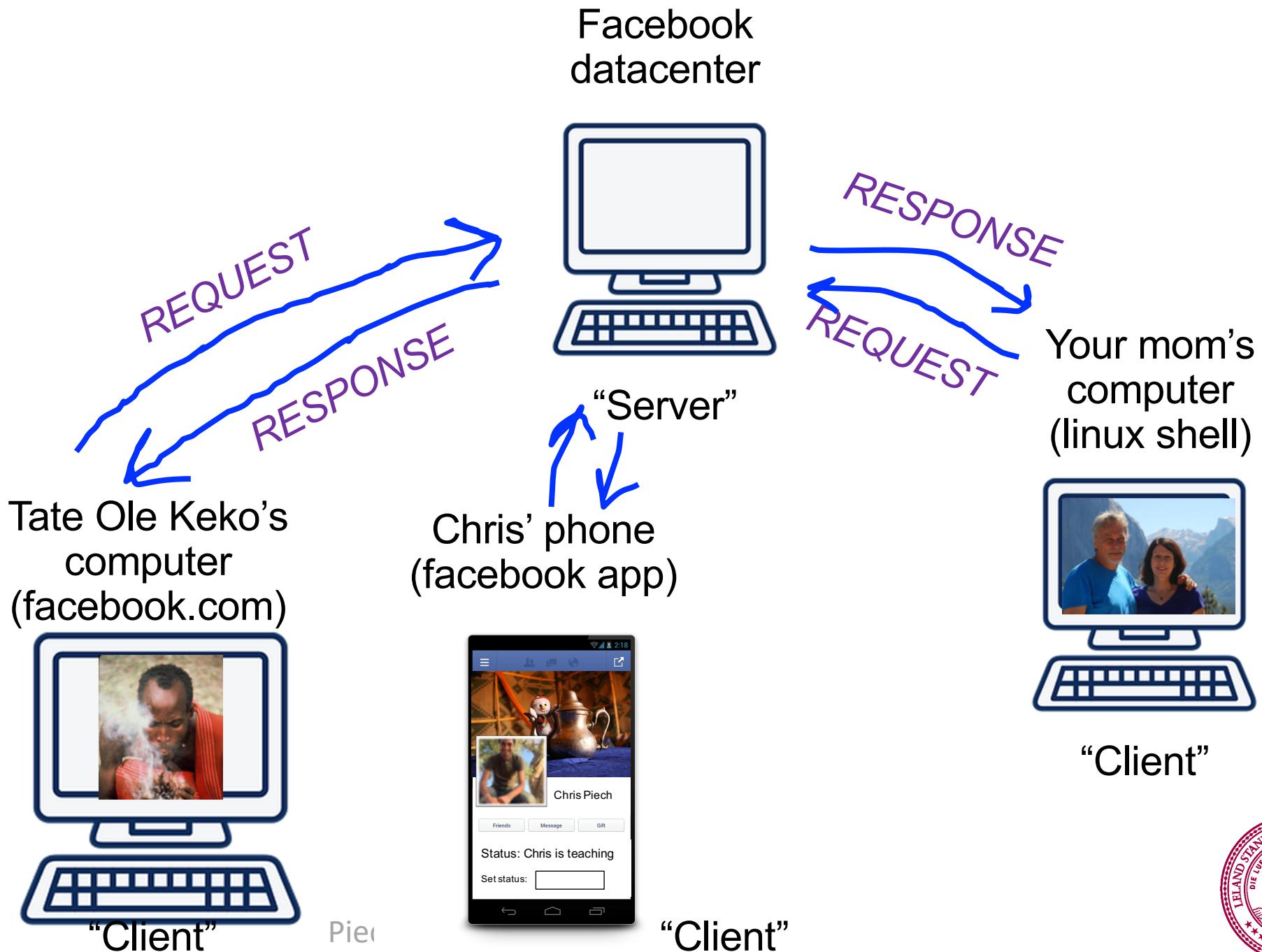
The Internet



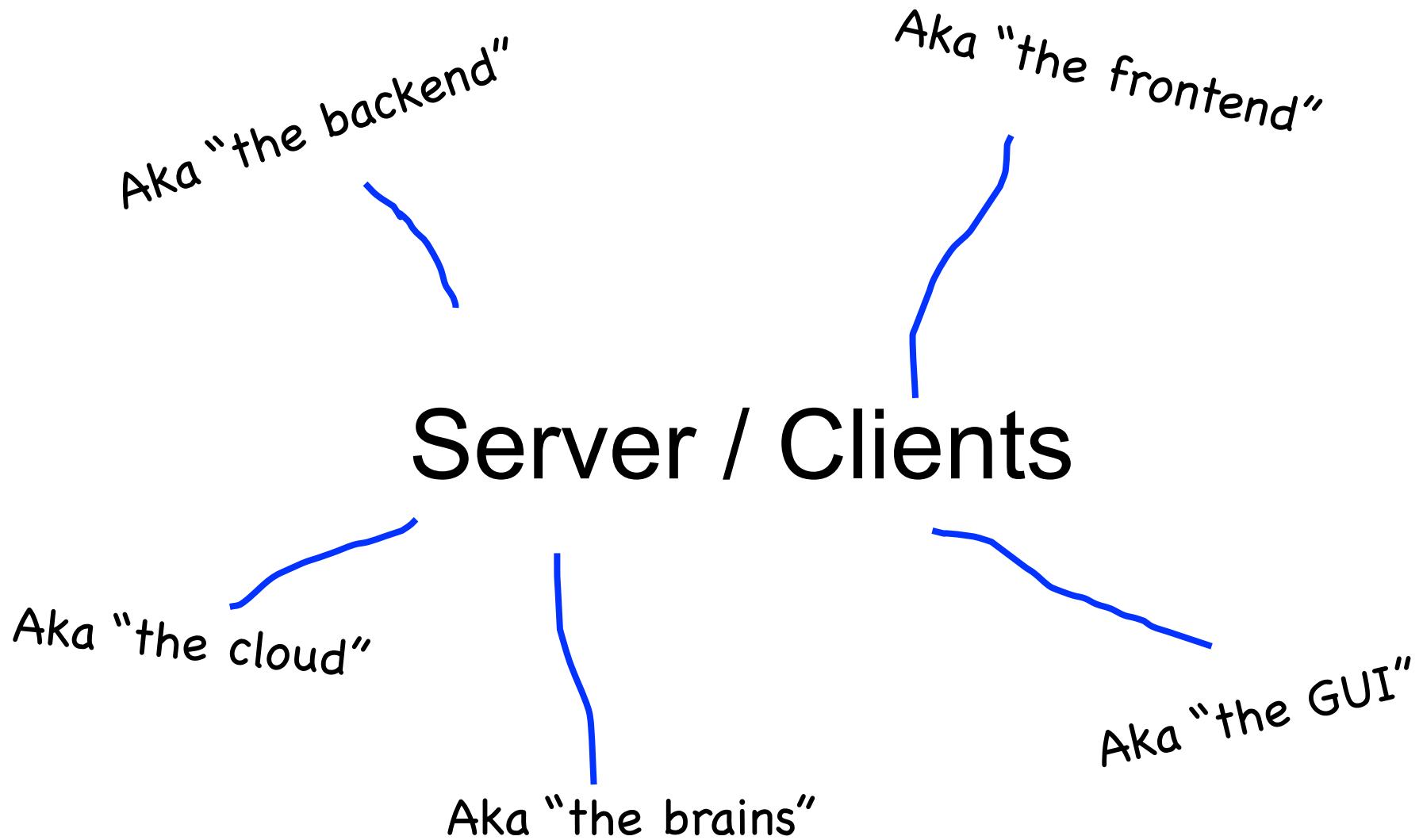


Many computers can connect
to the same server

The Internet



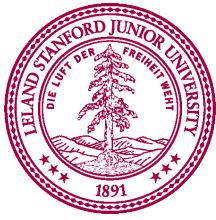
Most of the Internet



Today, the server



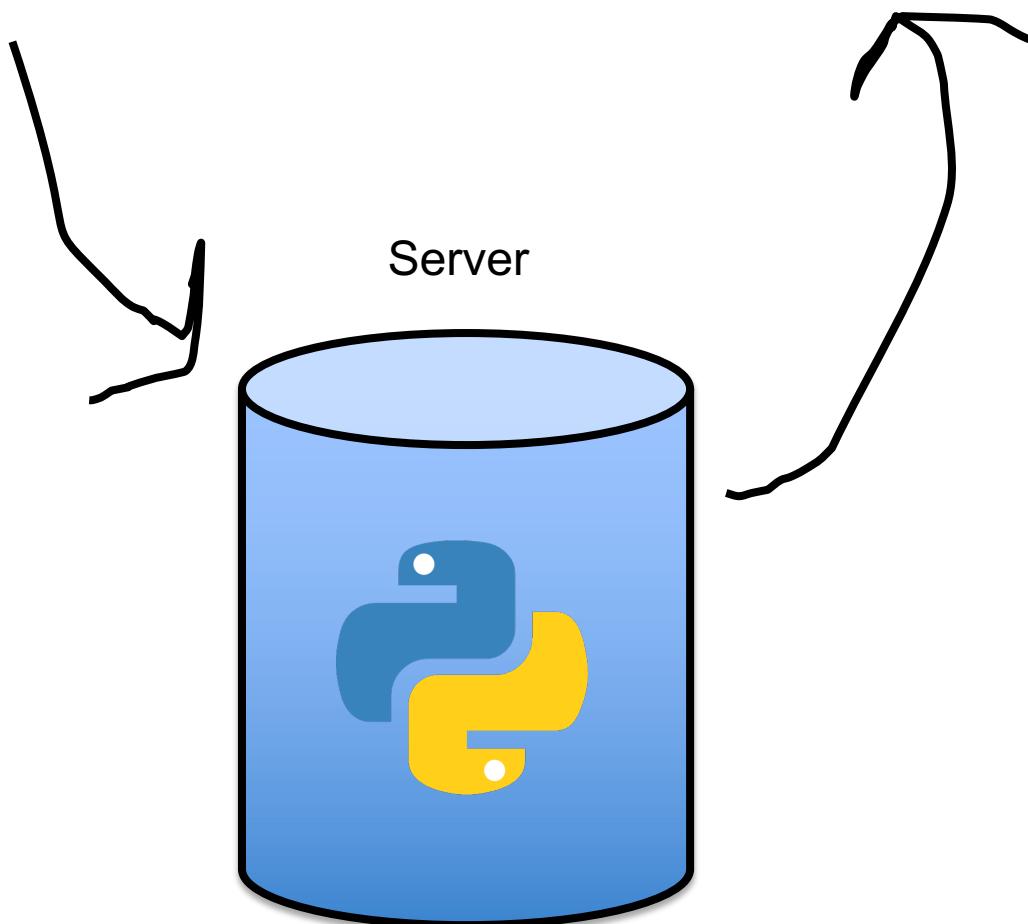
A server's main job is to respond to requests



A Server's Simple Purpose

Request
From a client

Response
To the client



A Server's Simple Purpose

Request
someRequest

String
serverResponse

```
ChatServer
Starting server on port 8080...
getMsgs
newMsg
Added new message
getMsgs
Returned 1 messages
getMsgs
Returned 1 messages
newMsg
Added new message
getMsgs
Returned 1 messages
getMsgs
```



Servers on one slide

1

```
# handle server requests (must be in a class)
def handle_request(self, request):
    # return a string response!
```

2

```
# turn on the server
def main():
    # make an instance of your server class
    handler = MyServer()
    # start the server!
    run_server(handler, 8000)
```

3

```
# enjoy
```



Servers on one slide

1

```
# handle server requests (must be in a class)
def handle_request(self, request):
    # return a string response!
```

2

```
# turn on the server
def main():
    # make an instance of your server class
    handler = HitServer()
    # start the server!
    run_server(handler, 8000)
```

3

```
# enjoy
```



Servers on one slide

1

```
# handle server requests (must be in a class)
def handle_request(self, request):
    # return a string response!
```

2

```
# turn on the server
def main():
    # make an instance of your server class
    handler = HitServer()
    # start the server!
    run_server(handler, 8000)
```

3

```
# enjoy
```



Servers on one slide

1

```
# handle server requests (must be in a class)
def handle_request(self, request):
    # return a string response!
```

2

```
# turn on the server
def main():
    # make an instance of your server class
    handler = HitServer()
    # start the server!
    run_server(handler, 8000)
```

3

```
# enjoy
```



Servers on one slide

1

```
# handle server requests (must be in a class)
def handle_request(self, request):
    # return a string response!
```

2

```
# turn on the server
def main():
    # make an instance of your server class
    handler = HitServer()
    # start the server!
    run_server(handler, 8000)
```

3

```
# enjoy
```



Servers on one slide

1

```
# handle server requests (must be in a class)
def handle_request(self, request):
    # return a string response!
```

2

```
# turn on the server
def main():
    # make an instance of your server class
    handler = HitServer()
    # start the server!
    run_server(handler, 8000)
```

3

```
# enjoy
```



Servers on one slide

1

```
# handle server requests (must be in a class)
def handle_request(self, request):
    # return a string response!
```

2

```
# turn on the server
def main():
    # make an instance of your server class
    handler = MyServer()
    # start the server!
    run_server(handler, 8000)
```

3

```
# enjoy
```



Servers on one slide

1

```
# handle server requests (must be in a class)
def handle_request(self, request):
    # return a string response!
```

2

```
# turn on the server
def main():
    # make an instance of your server class
    handler = MyServer()
    # start the server!
    run_server(handler, 8000)
```

3

```
# enjoy
```



Servers on one slide

1

```
# handle server requests (must be in a class)
def handle_request(self, request):
    # return a string response!
```

2

```
# turn on the server
def main():
    # make an instance of your server class
    handler = HitServer()
    # start the server!
    run_server(handler, 8000)
```

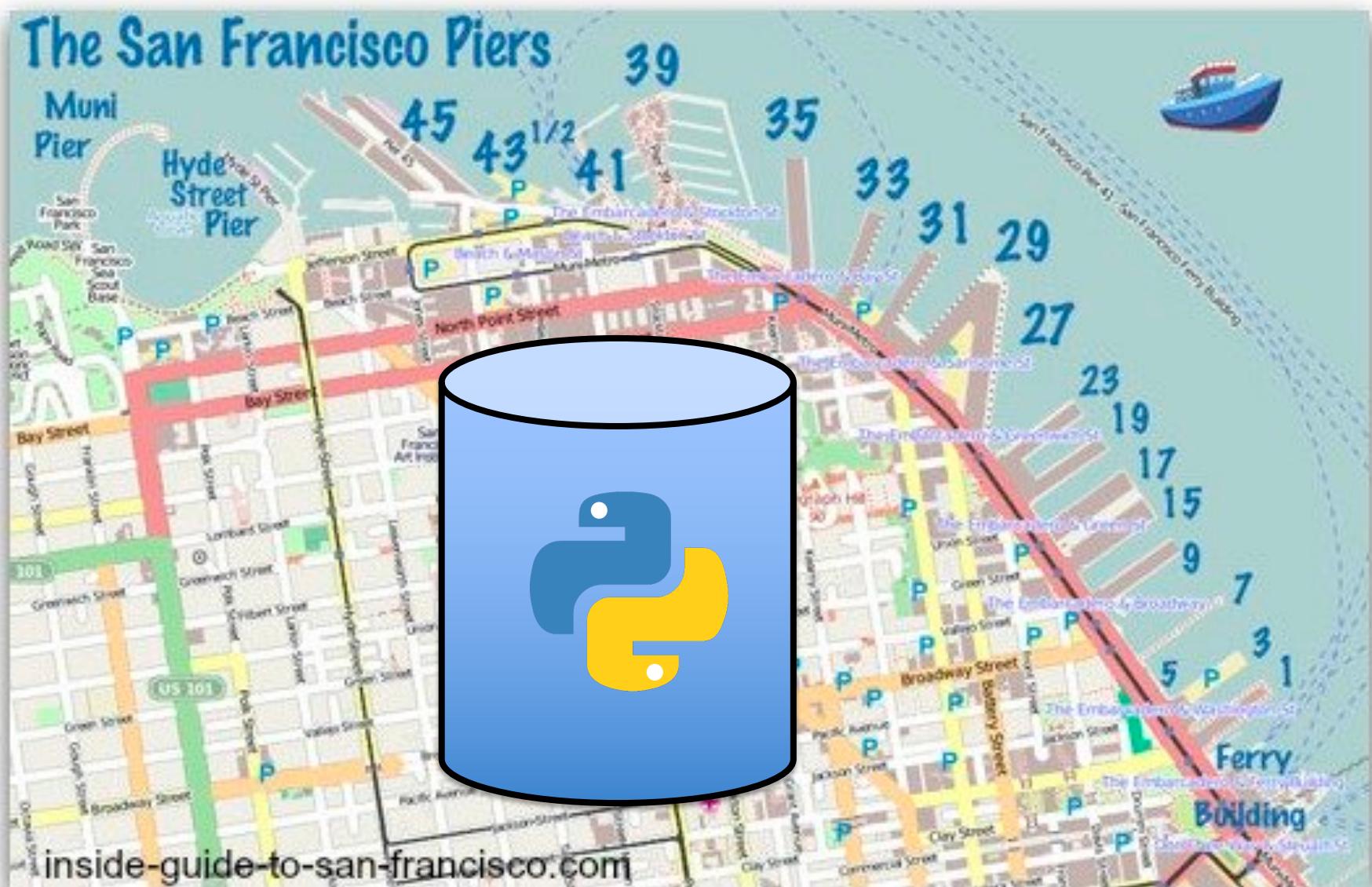
3

enjoy



What is a Port?

80 is a **special port** for a server that wants to talk to **web browsers**



Servers on one slide

1

```
# handle server requests (must be in a class)
def handle_request(self, request):
    # return a string response!
```

2

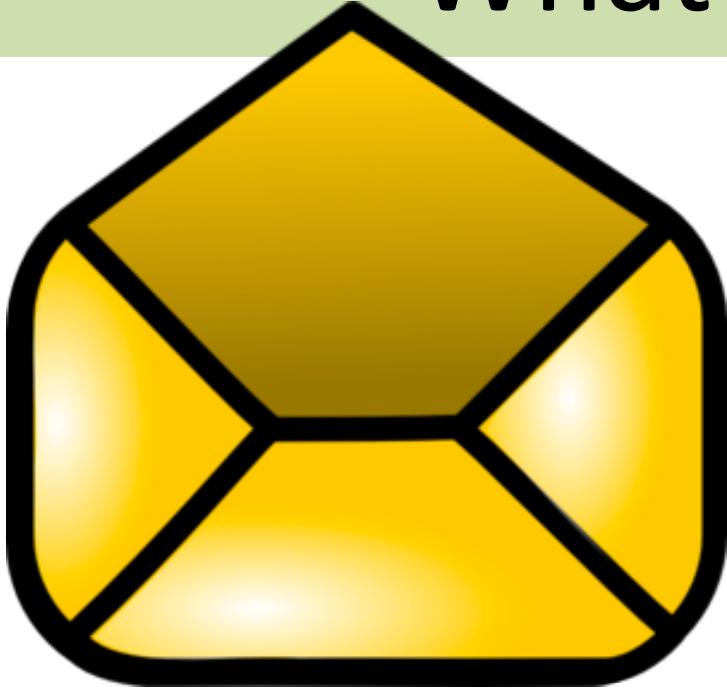
```
# turn on the server
def main():
    # make an instance of your server class
    handler = HitServer()
    # start the server!
    run_server(handler, 8000)
```

3

```
# enjoy
```



What is a Request?



```
/* Request has a command */  
command (type is string)  
  
/* Request has parameters */  
params (type is dict)
```

```
// methods that the server calls on requests  
request.command  
request.params
```



```
class Request:
```

```
    '''
```

The request class packages the key information from an internet request. An internet request has both a command and a dictionary of parameters. This class defines a special function `__str__` which means if you have an instance of a request you can put it in a print function.

```
    '''
```

```
def __init__(self, request_command, request_params):
```

```
    # every request has a command (string)
```

```
    self.command = request_command
```

```
    # every request has params (dictionary). Can be {}
```

```
    self.params = request_params
```

```
def get_params(self):
```

```
    # a 'getter' method to get the params
```

```
    return self.params
```

```
def get_command(self):
```

```
    # a 'getter' method to get the command
```

```
    return self.command
```

```
def __str__(self):
```

```
    # a special method which says what happens when you 'print' a request
```

```
    return 'command=' + self.command + '\n' 'params=' + str(self.params)
```

First Server Example!

```
from SimpleInternet import run_server
import json

class MyServer:
    def __init__(self):
        ''' You can store data in your server! '''
        pass

    # this is the server request callback function.
    def handle_request(self, request):
        ''' This function gets called every time someone makes a
        request to our server.'''
        return 'hello world'

def main():
    # make an instance of your server class
    handler = MyServer()
    # start the server to handle internet requests!
    run_server(handler, 8000)
```



Who makes requests?

Who makes requests?

Other programs can send requests!

```
response = requests.get('https://xkcd.com/353/')
```

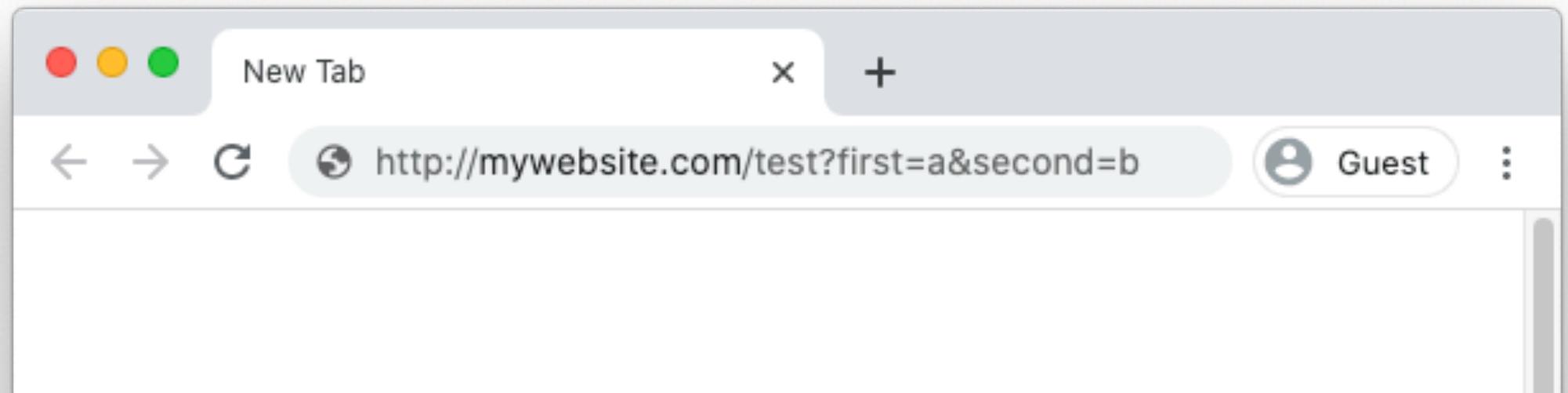
Who makes requests?

Other programs can send requests!

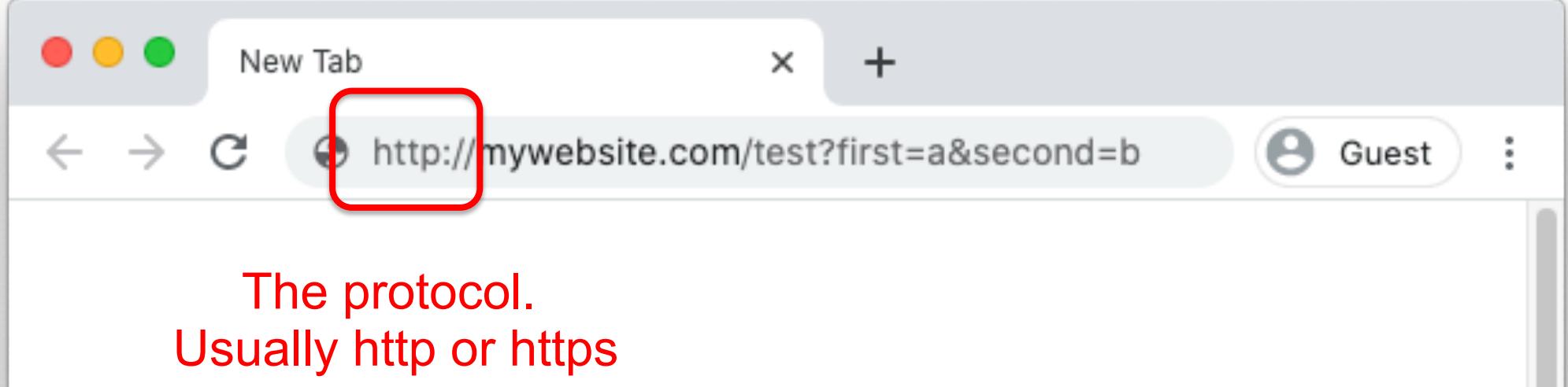
```
response = requests.get('https://xkcd.com/353/')
```

Web browsers can send requests!

Anatomy of a Browser Request



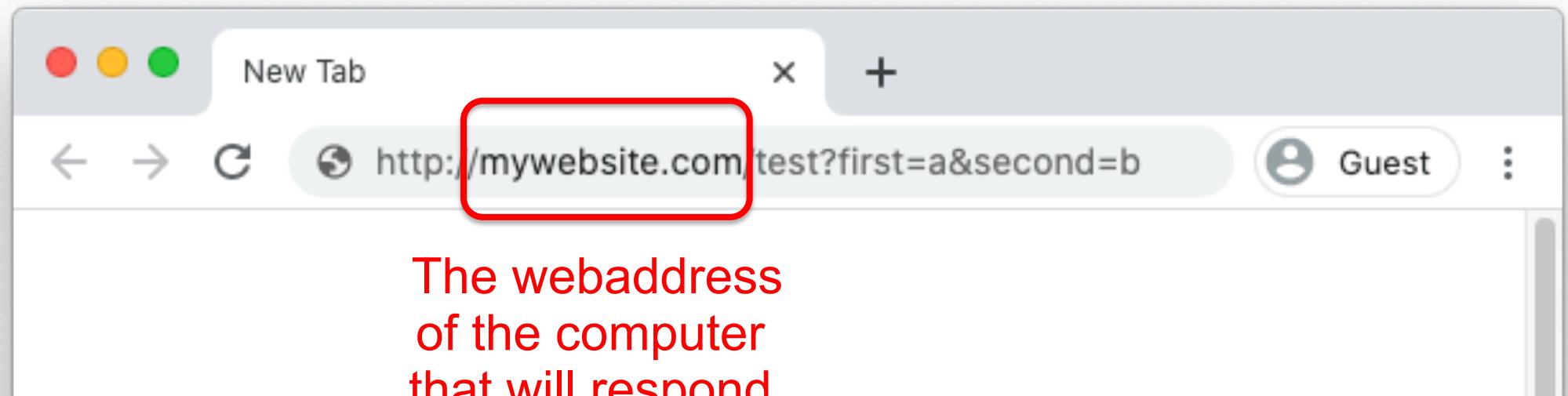
Anatomy of a Browser Request



The protocol.
Usually http or https



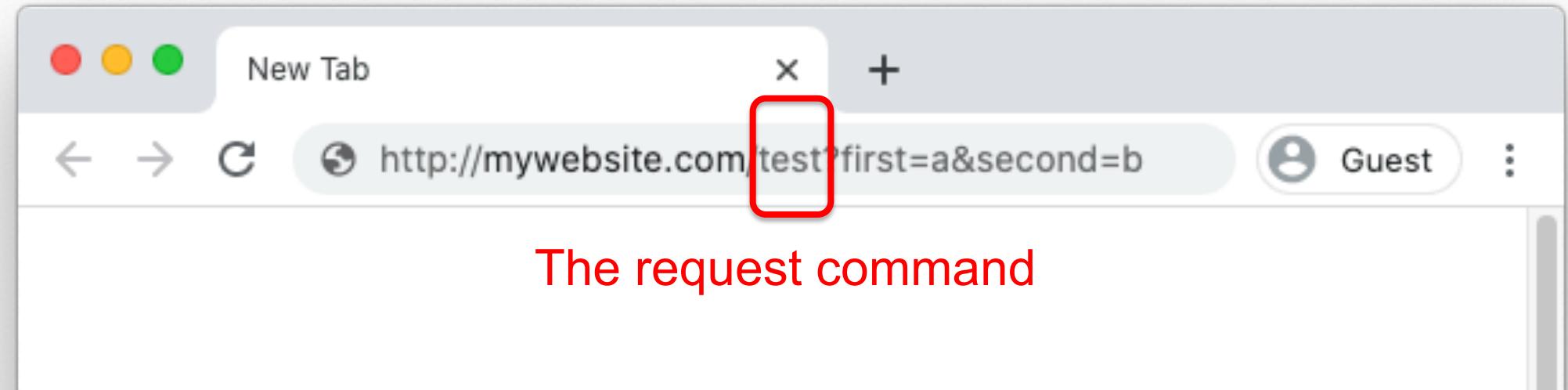
Anatomy of a Browser Request



The webaddress
of the computer
that will respond
to the request



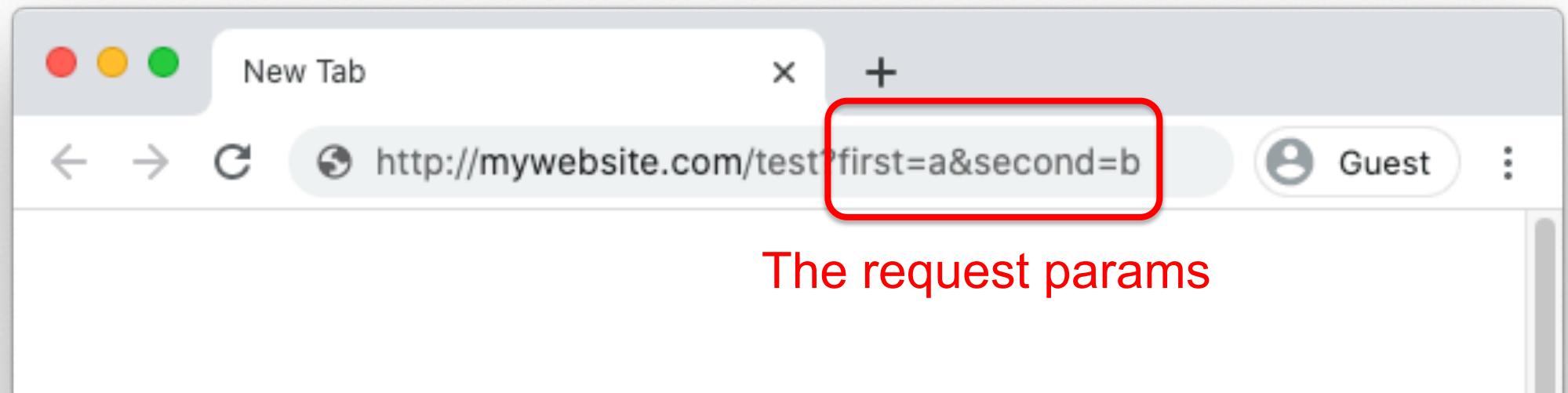
Anatomy of a Browser Request



The request command



Anatomy of a Browser Request



First Server Example!

```
from SimpleInternet import run_server
import json

class MyServer:
    def __init__(self):
        ''' You can store data in your server! '''
        pass

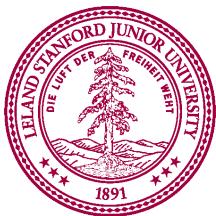
    # this is the server request callback function.
    def handle_request(self, request):
        ''' This function gets called every time someone makes a
        request to our server.'''
        return 'hello world'

def main():
    # make an instance of your server class
    handler = MyServer()
    # start the server to handle internet requests!
    run_server(handler, 8000)
```



Hit Counter

668332



Recall Requests

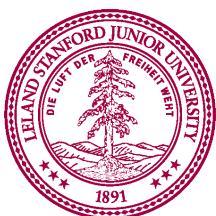
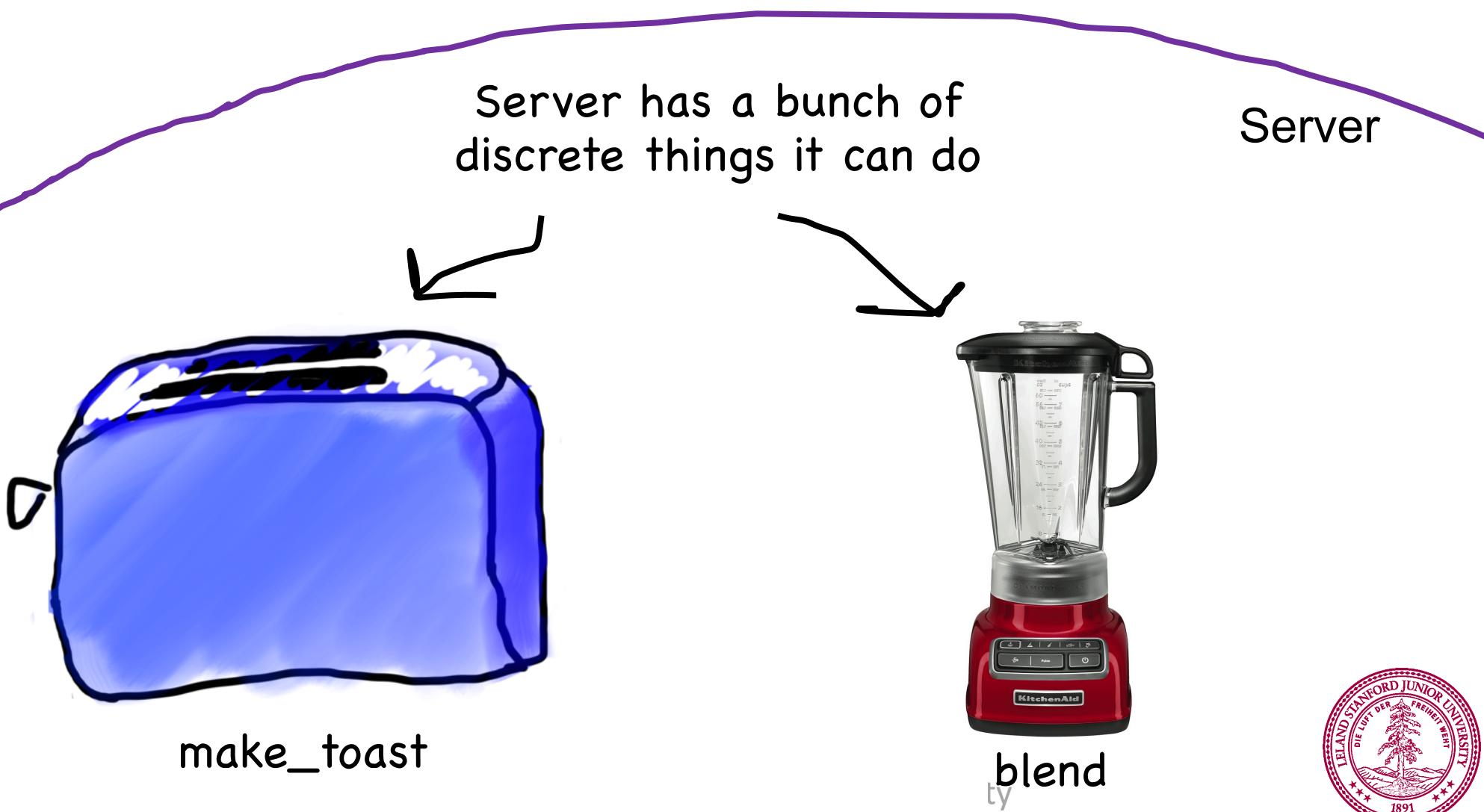


```
/* Request has a command */  
command (string)  
  
/* Request has parameters */  
params (dict)
```

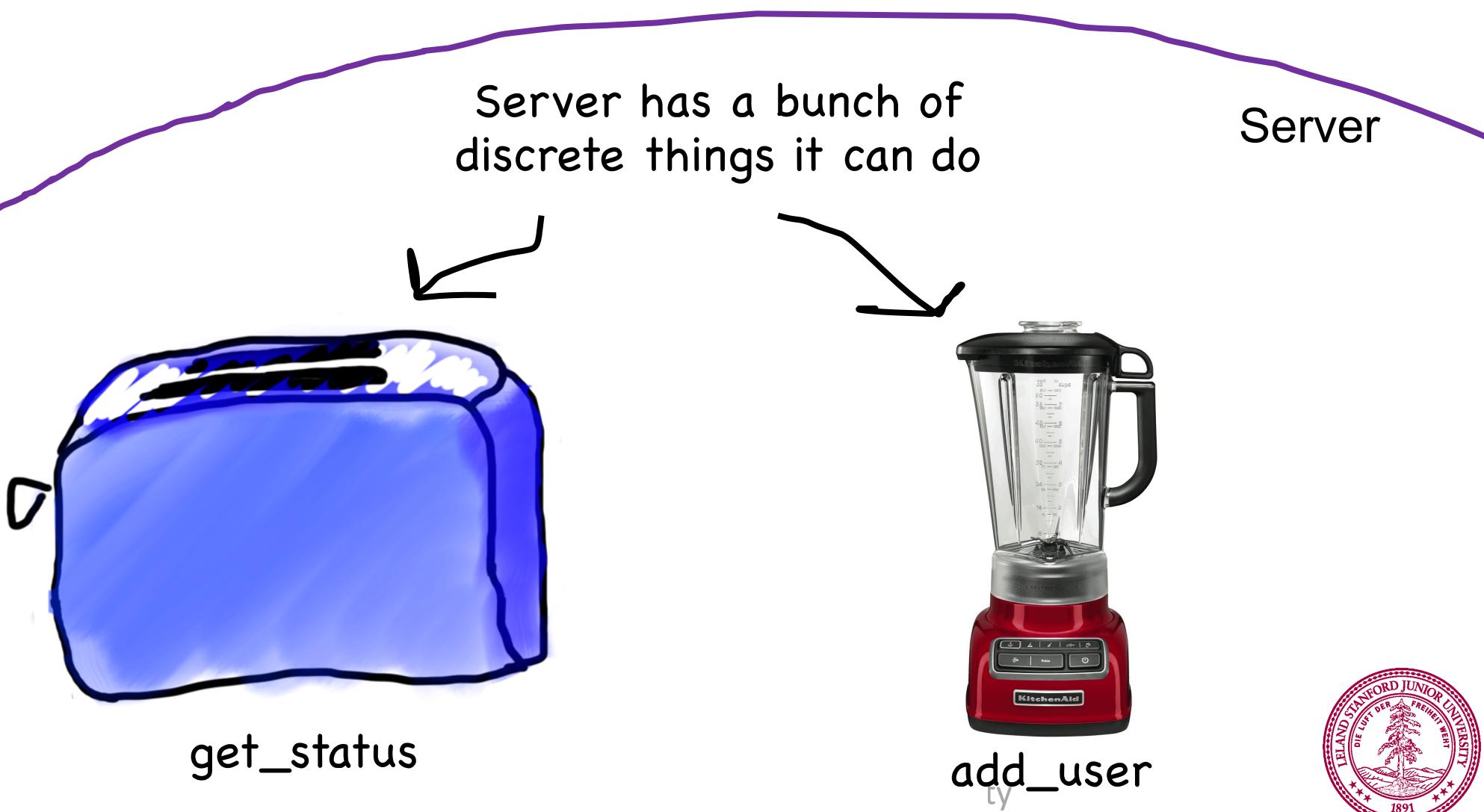
```
// methods that the server calls on requests  
request.command  
request.params
```



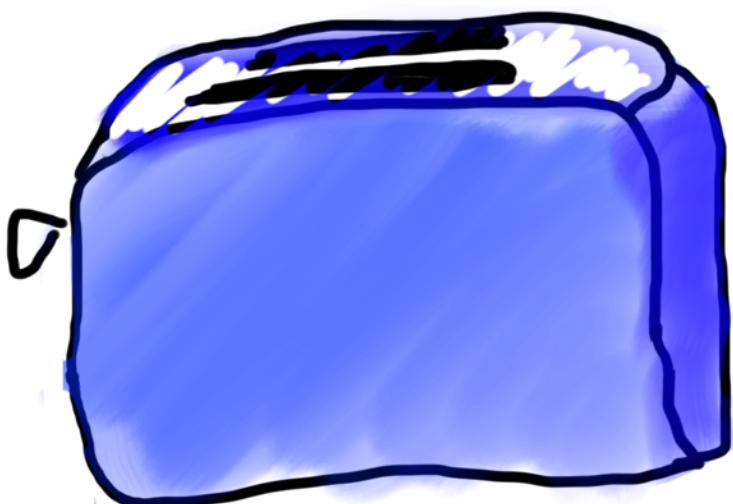
Requests are like Remote Method Calls



Requests are like Remote Method Calls



Requests are like Remote Method Calls



get_status

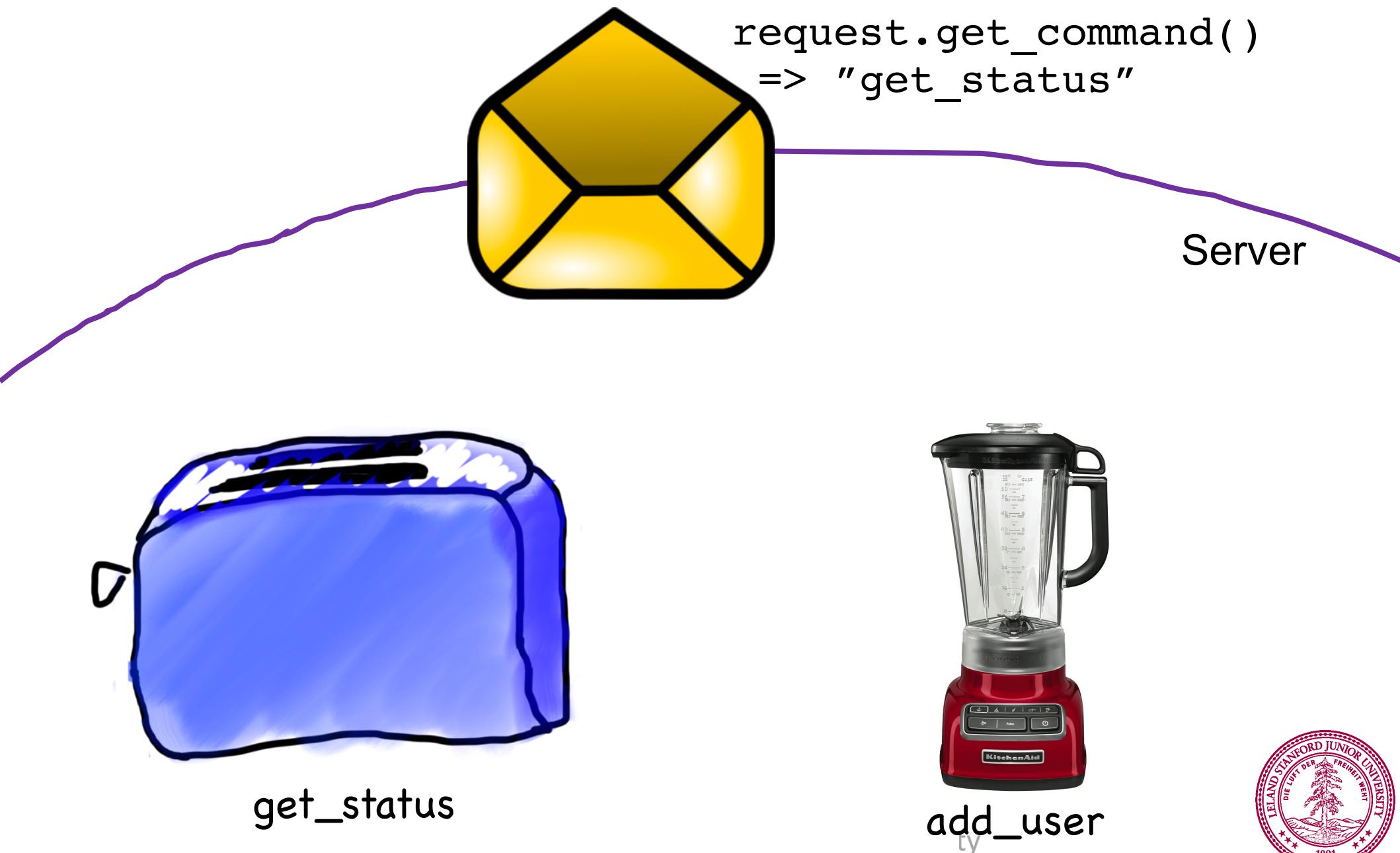


add_user

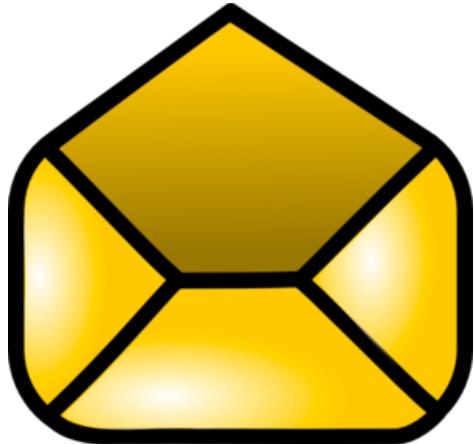
Server



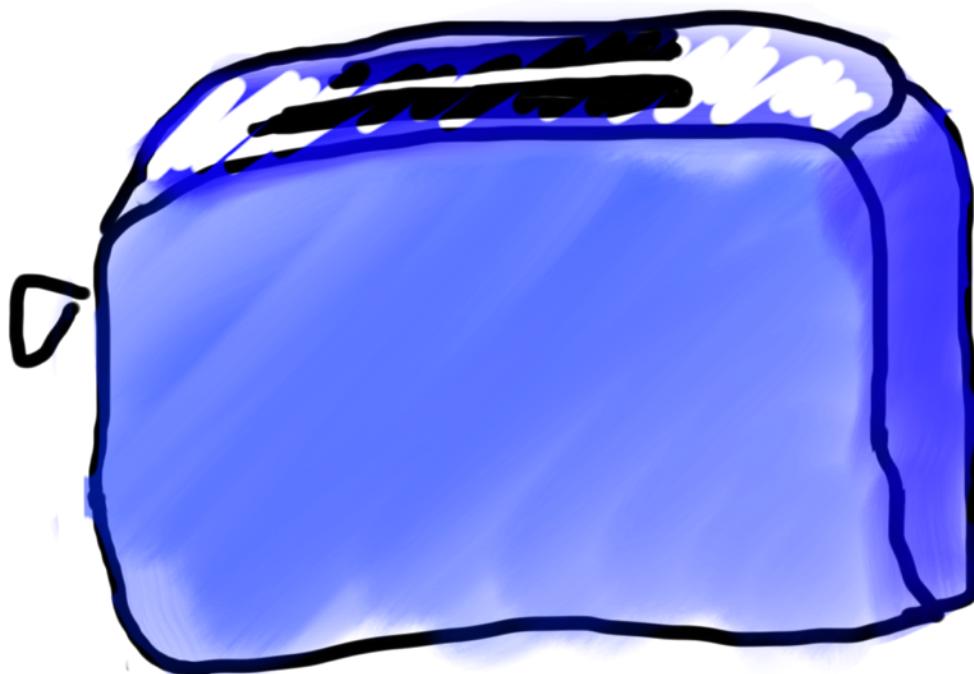
Requests are like Remote Method Calls



Requests are like Remote Method Calls



To make toast, I need a parameter which is the kind of bread

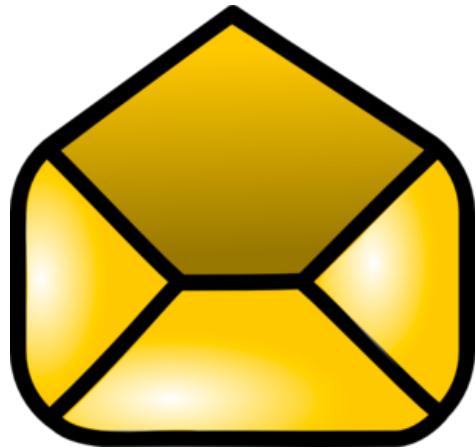


get_status

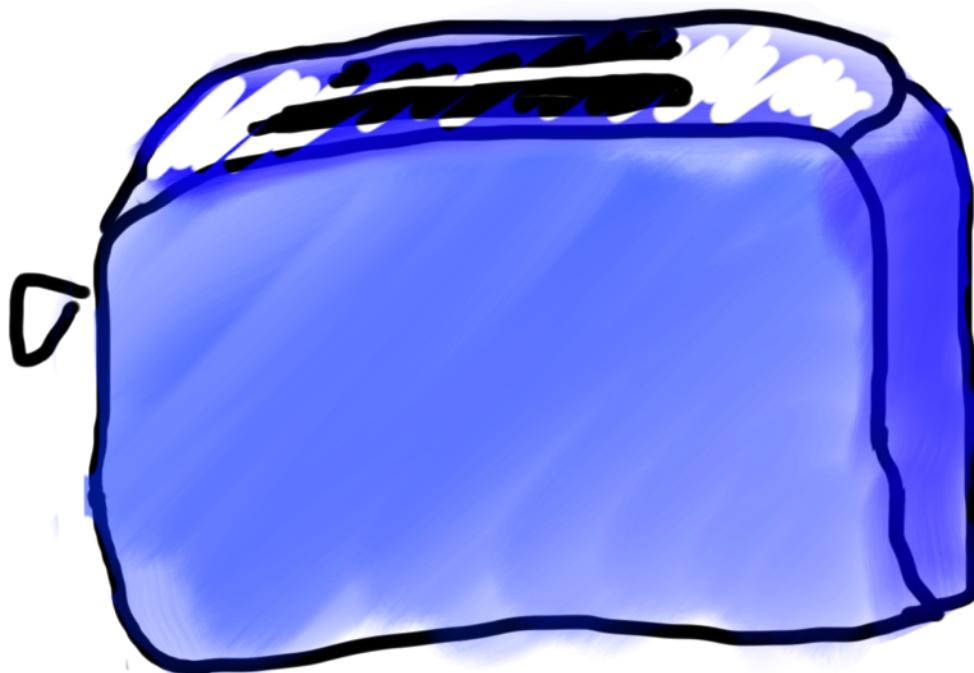
ty



Requests are like Remote Method Calls



I was given a parameter!

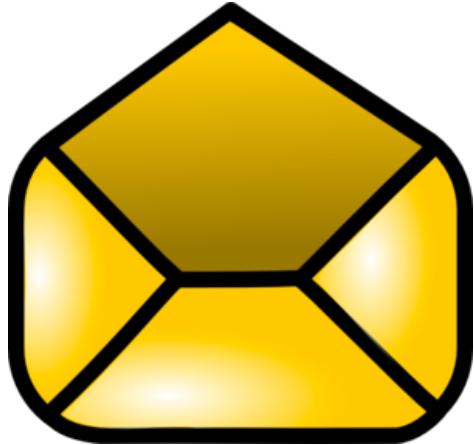


get_status

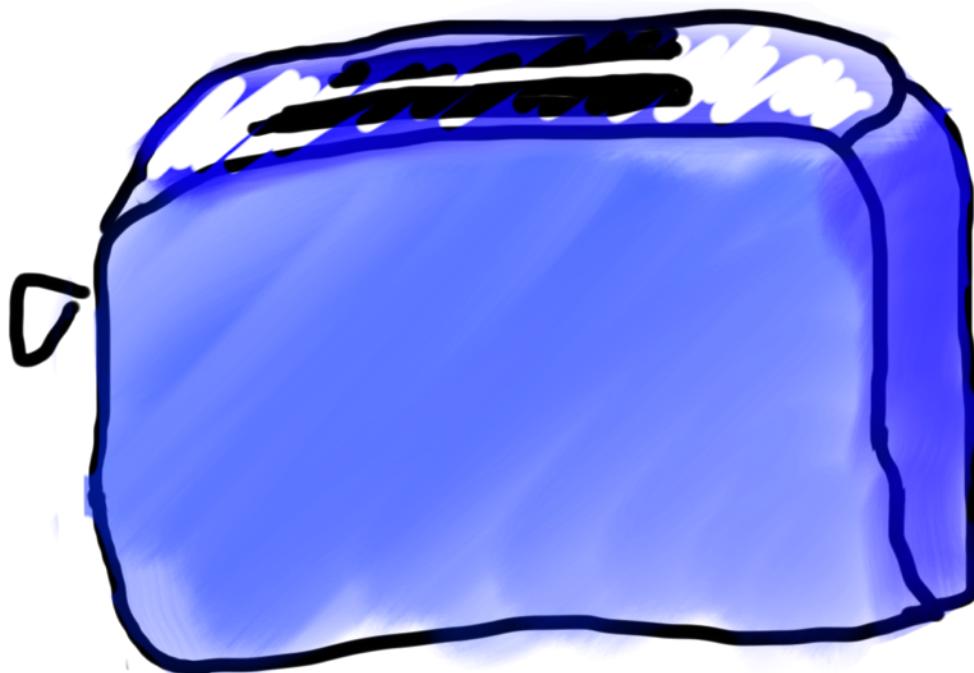
ty



Requests are like Remote Method Calls



`request.params["userName"]`

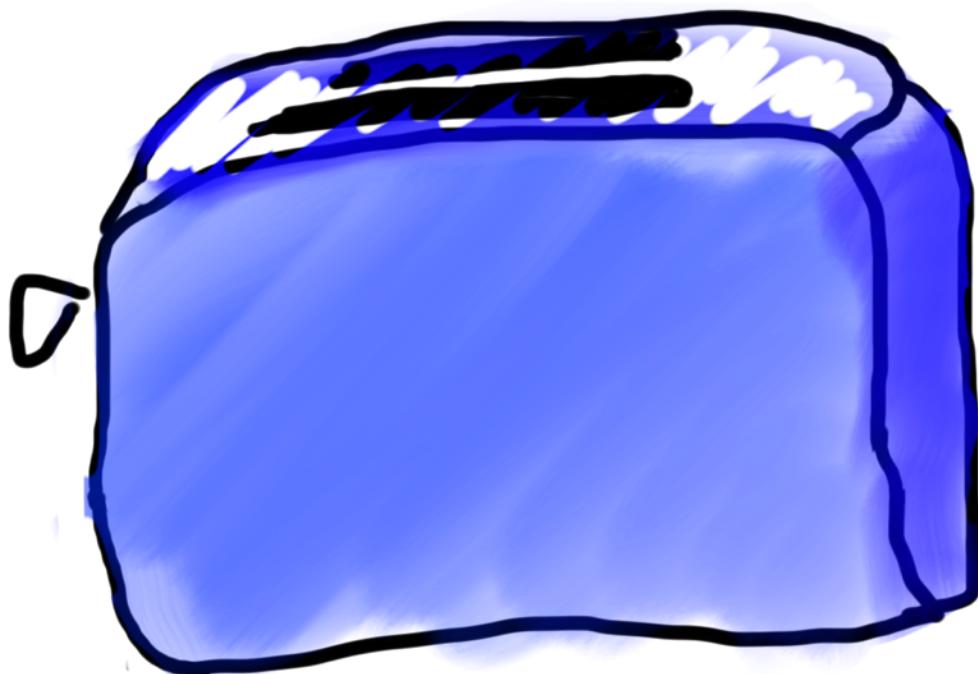
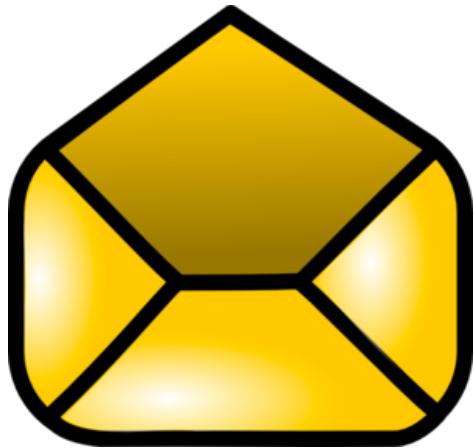


`get_status`

ty



Requests are like Remote Method Calls

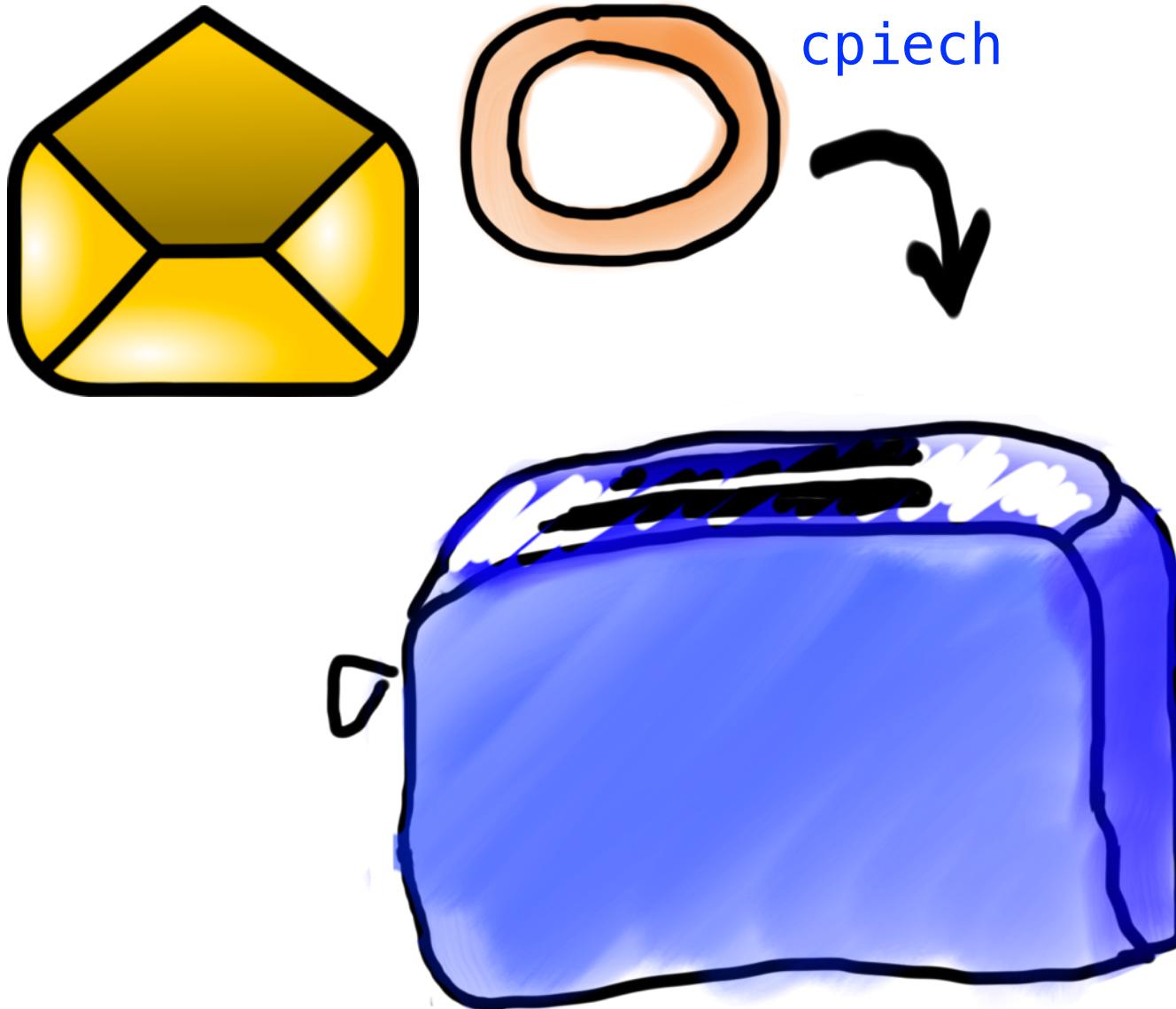


get_status

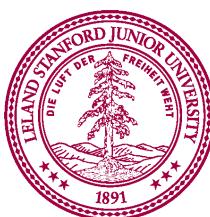
ty



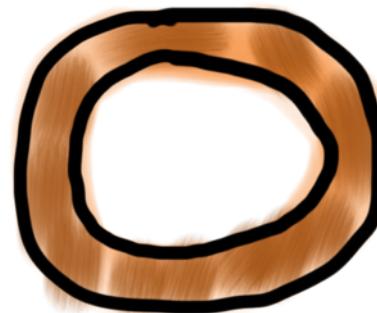
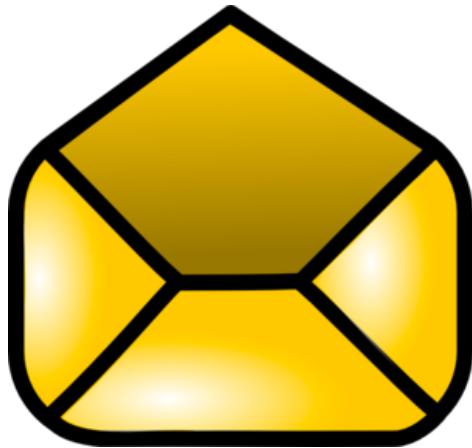
Requests are like Remote Method Calls



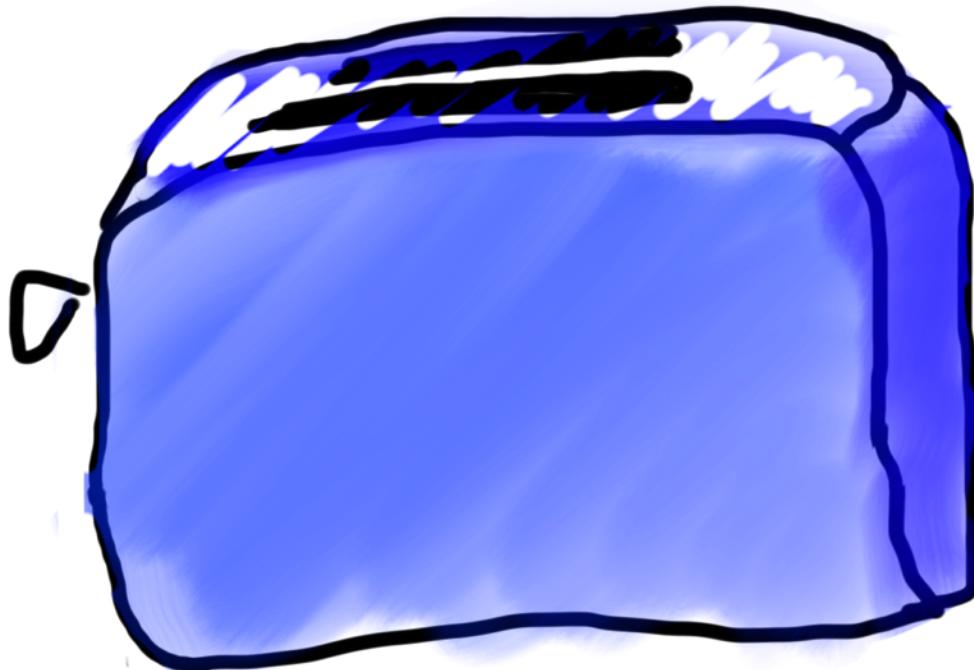
ty



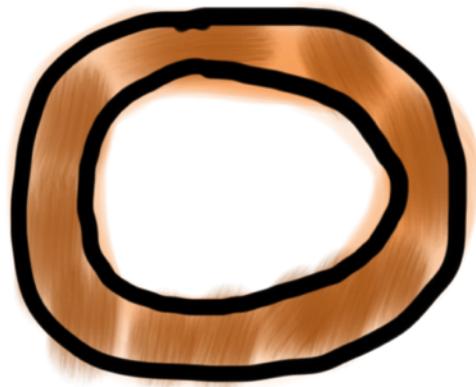
Requests are like Remote Method Calls



teaching



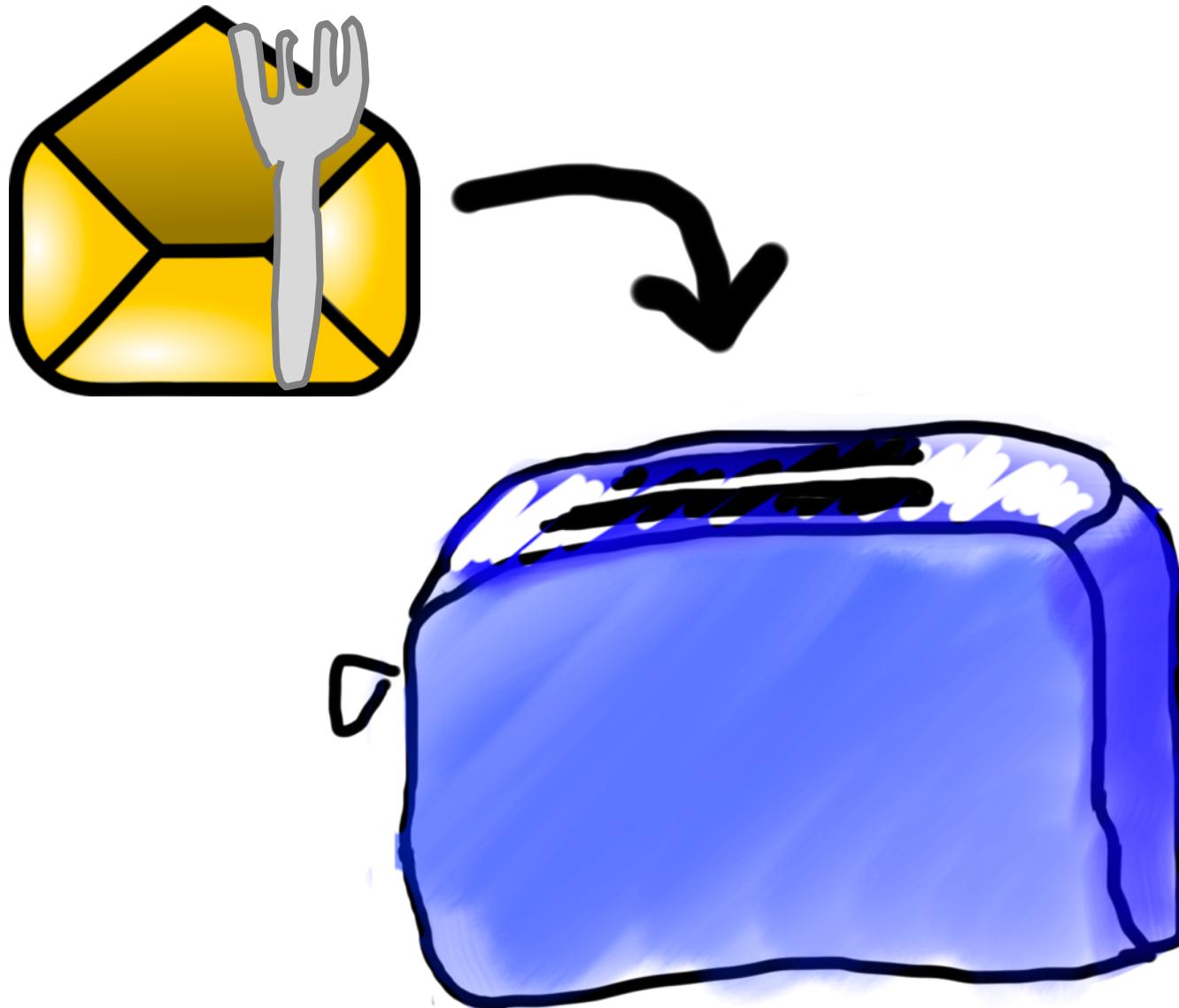
```
def handle_request(self, request):
    cmd = request.command
    if cmd == 'get_status':
        user = request.params['userName']
        status = self.get_status(user)
    return status
```



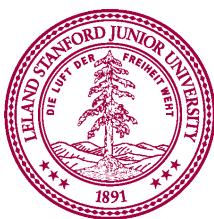
Must be a string!



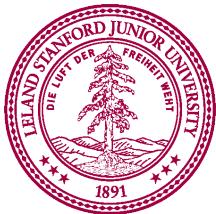
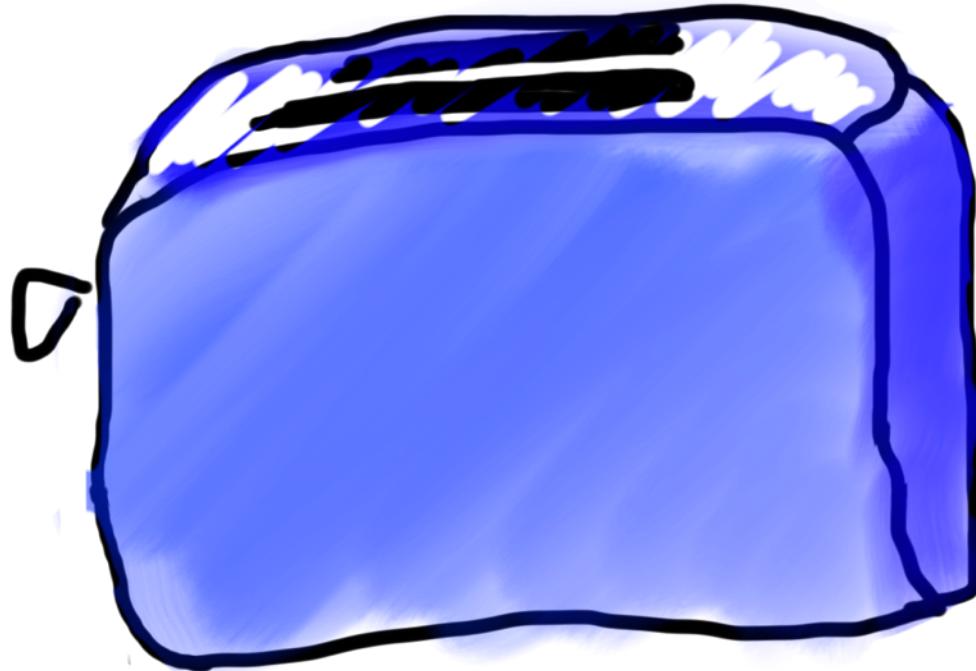
Requests are like Remote Method Calls



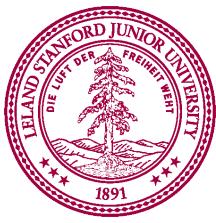
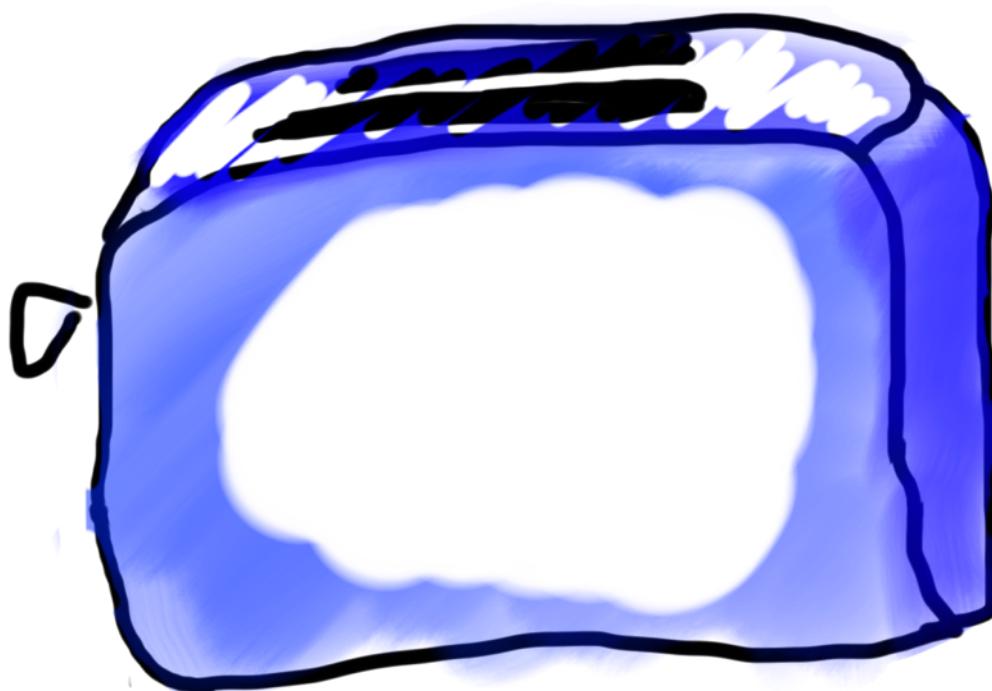
Requests are like Remote Method Calls



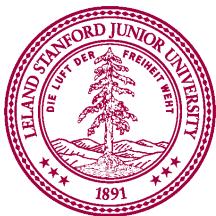
Requests are like Remote Method Calls



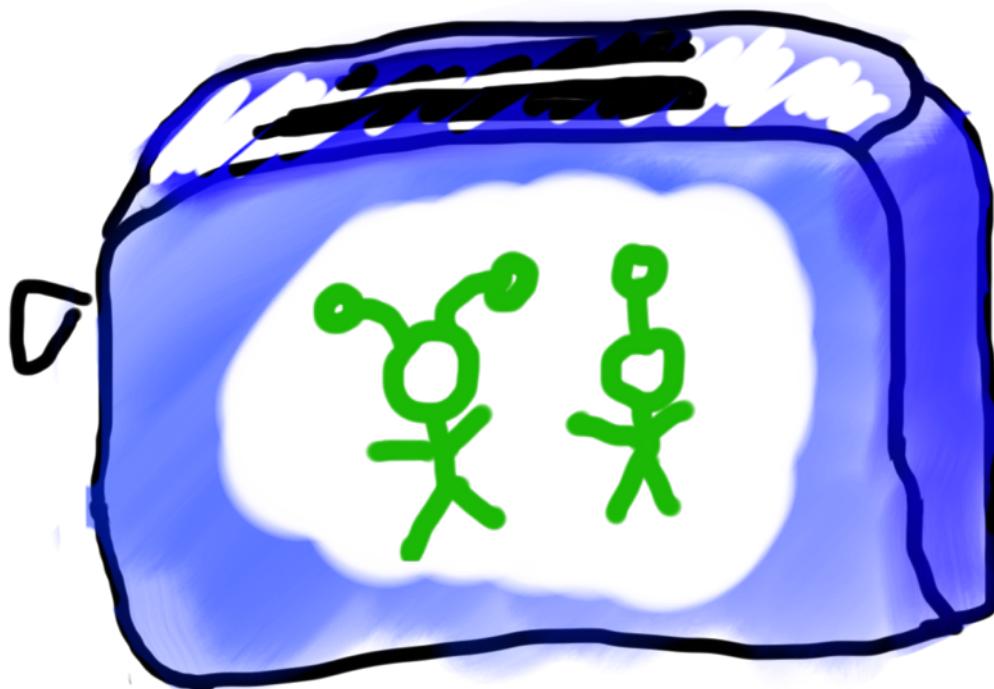
Requests are like Remote Method Calls



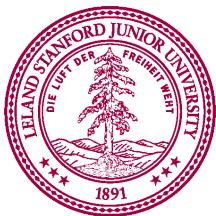
Requests are like Remote Method Calls



Requests are like Remote Method Calls

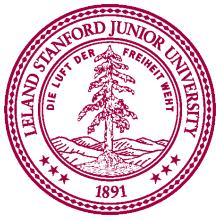


Piech + Sahami, CS106A, Stanford University





Requests responses are
strings, often encoded
using JSON



Recall JSON

ages.json

```
{  
    "Chris":32,  
    "Gary":70,  
    "Mehran":50,  
    "Brahm":23,  
    "Rihanna":32,  
    "Adele":32  
}
```

```
import json  
  
# load data  
data = json.load(open('ages.json'))  
  
# save data  
json.dump(data, open('ages.json'))
```



Recall JSON

ages.json

```
{  
    "Chris":32,  
    "Gary":70,  
    "Mehran":50,  
    "Brahm":23,  
    "Rihanna":32,  
    "Adele":32  
}
```

```
import json  
  
# load data  
data = json.load(open('ages.json'))  
  
# save data  
json.dump(data, open('ages.json'))
```



Recall JSON

ages.json

```
{  
    "Chris":32,  
    "Gary":70,  
    "Mehran":50,  
    "Brahm":23,  
    "Rihanna":32,  
    "Adele":32  
}
```

```
import json  
  
# load data  
data = json.load(open('ages.json'))  
  
# save data  
json.dump(data, open('ages.json'))  
  
# write a variable to a string  
data_str = json.dumps(data)
```



Time for a little chat

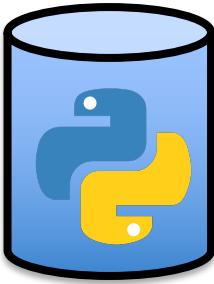
Chat Server and Client

The screenshot shows a Mac OS X desktop with two windows open. On the left is a "Chat Client" window with a text input field containing "The internet is a wild place..." and a "Send" button. Below it is a "Messages" section showing a list of messages from users Chris, Laura, and Terry. On the right is a terminal window titled "backend — Python chat_server.py — 93x34" displaying a log of message exchange between the client and the server. The log shows various commands like "getMsgs" and "newMsg" being sent and received.

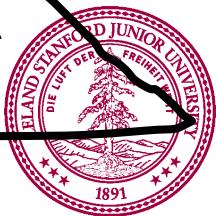
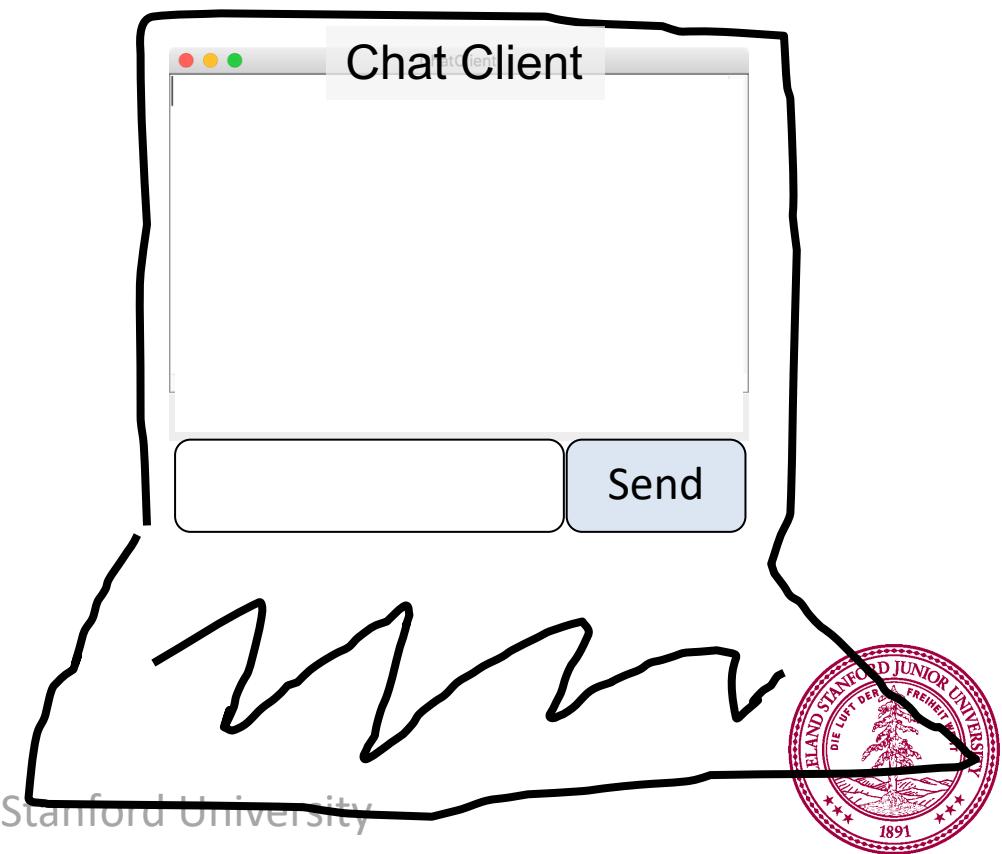
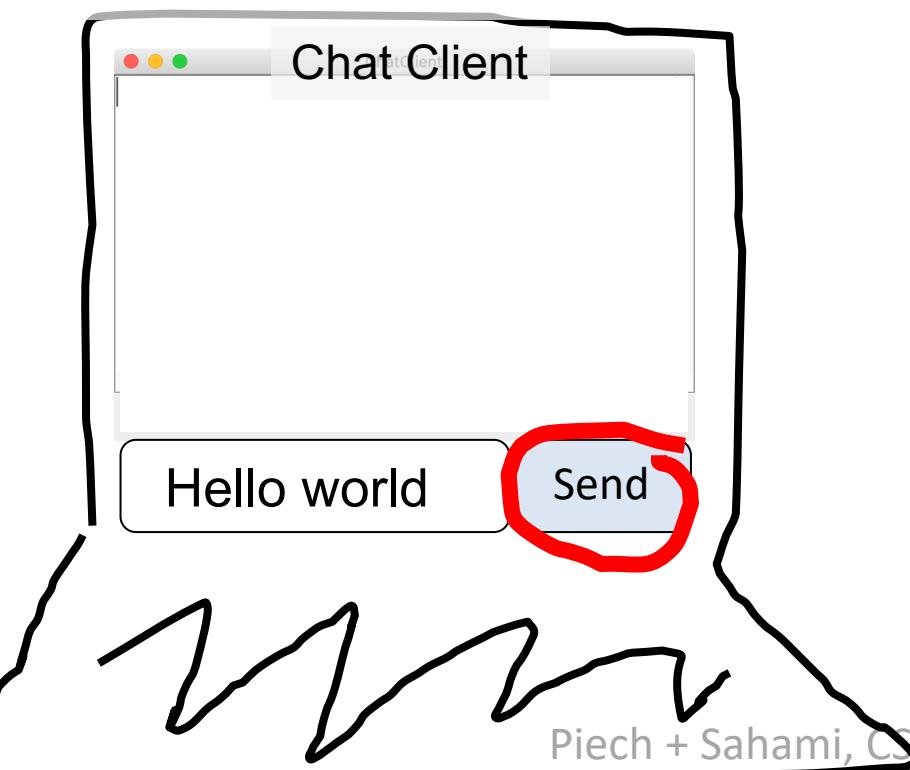
```
Chris@ndoto backend % python chat_server.py
Server running...
[{'command': 'getMsgs', 'params': {'index': '0'}},
 {'command': 'newMsg', 'params': {'msg': 'Hello world?', 'user': 'Chris'}},
 {'command': 'getMsgs', 'params': {'index': '0'}},
 {'command': 'getMsgs', 'params': {'index': '0'}},
 {'command': 'newMsg', 'params': {'msg': 'Here I am!!!', 'user': 'Laura'}},
 {'command': 'getMsgs', 'params': {'index': '1'}},
 {'command': 'newMsg', 'params': {'msg': 'This is fun!', 'user': 'Laura'}},
 {'command': 'getMsgs', 'params': {'index': '2'}},
 {'command': 'getMsgs', 'params': {'index': '1'}},
 {'command': 'newMsg', 'params': {'msg': 'Wahooooo :-)', 'user': 'Chris'}},
 {'command': 'getMsgs', 'params': {'index': '3'}},
 {'command': 'newMsg', 'params': {'msg': 'We are on the internet...', 'user': 'Chris'}},
 {'command': 'getMsgs', 'params': {'index': '4'}},
 {'command': 'newMsg', 'params': {'msg': 'This is like low-budget WhatsApp', 'user': 'Chris'}},
 {'command': 'getMsgs', 'params': {'index': '5'}},
 {'command': 'getMsgs', 'params': {'index': '3'}},
 {'command': 'getMsgs', 'params': {'index': '6'}},
 {'command': 'getMsgs', 'params': {'index': '6'}},
 {'command': 'getMsgs', 'params': {'index': '6'}},
 {"command": "newMsg", "params": {"msg": "But we made it, which is cool.", "user": "Laura"}},
 {"command": "getMsgs", "params": {"index": "6"}}, {"command": "getMsgs", "params": {"index": "0"}}, {"command": "newMsg", "params": {"msg": "Hi everyone! Terry here too", "user": "Terry"}}, {"command": "getMsgs", "params": {"index": "7"}}, {"command": "newMsg", "params": {"msg": "Hi Terry!", "user": "Laura"}}, {"command": "getMsgs", "params": {"index": "7"}}, {"command": "getMsgs", "params": {"index": "8"}}, {"command": "newMsg", "params": {"msg": "The internet is a wild place...", "user": "Terry"}}, {"command": "getMsgs", "params": {"index": "9"}}, {"command": "getMsgs", "params": {"index": "9"}}
```

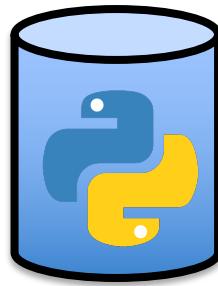


```
history = [  
]
```



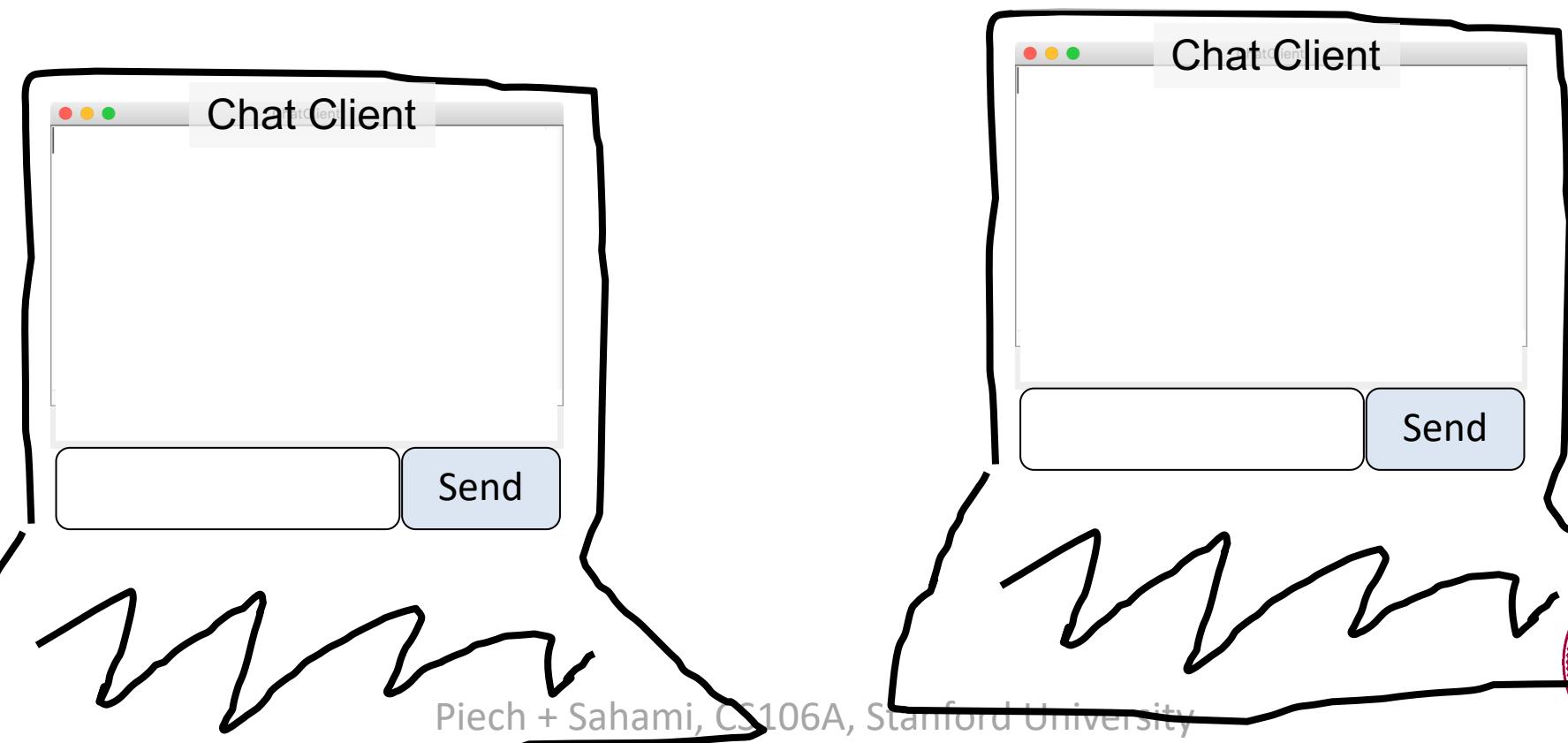
```
addMsg  
{  
    'msg' : Hello world,  
    'user' : 'C'  
}
```





```
history = [  
    '[C] Hello world'  
]
```

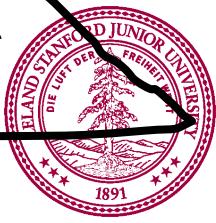
```
getMsgs  
{  
    'index' : 0  
}
```

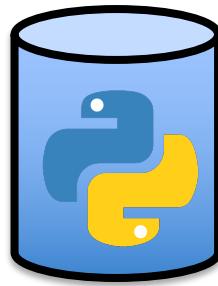




```
history = [  
    '[C] Hello world'  
]
```

' [" [C] Hello world"] '





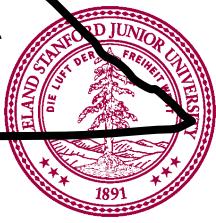
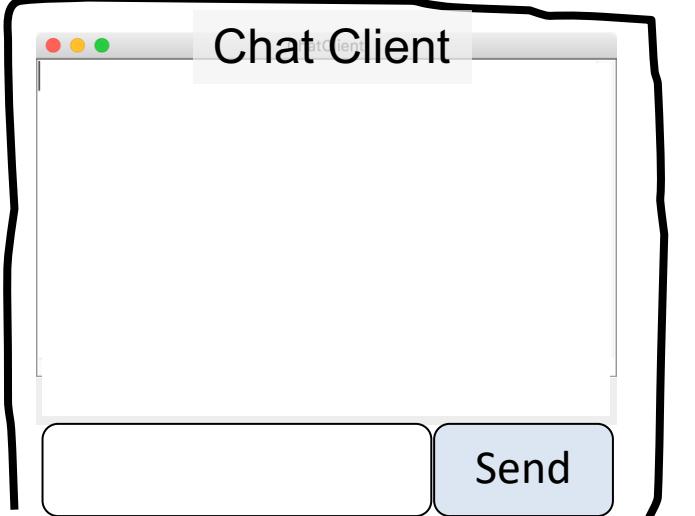
```
history = [  
    '[C] Hello world'  
]  
  
addMsg  
{  
    'msg' : 'Im here too'  
    'user' : 'B'  
}
```

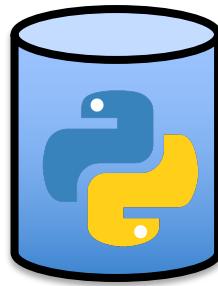




```
history = [  
    '[C] Hello world',  
    '[B] Im here too'  
]
```

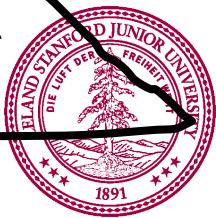
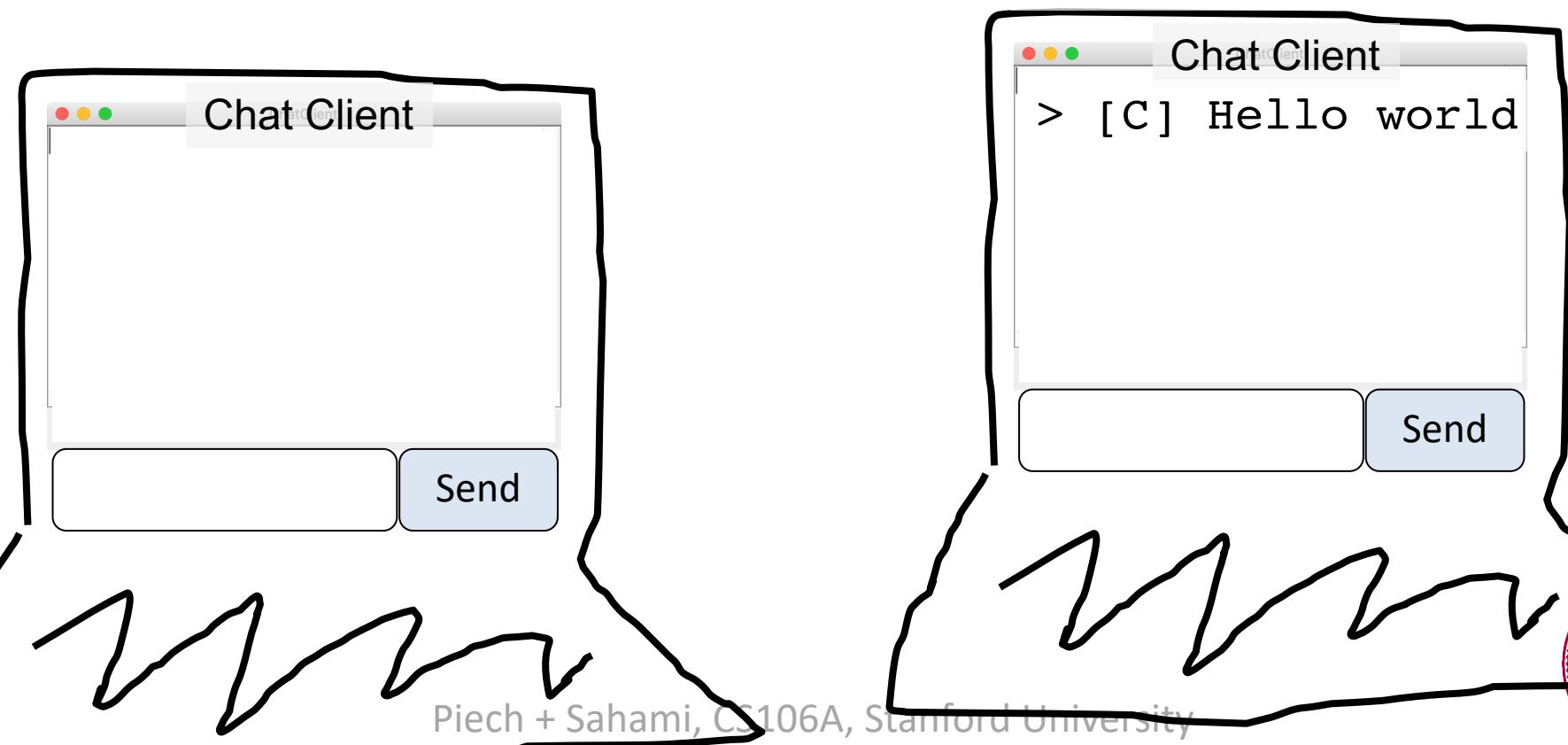
'Got it'





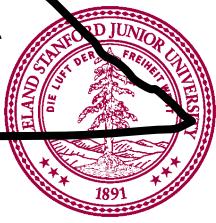
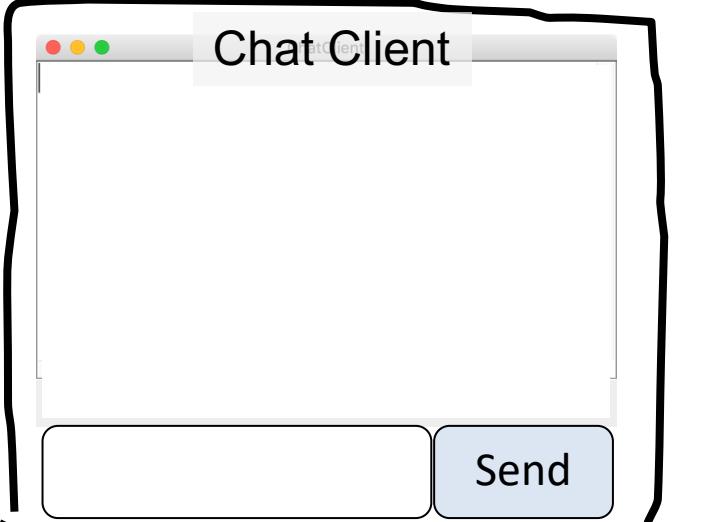
```
history = [  
    '[C] Hello world',  
    '[B] Im here too'  
]
```

```
getMsgs  
{  
    'index' : 1  
}
```

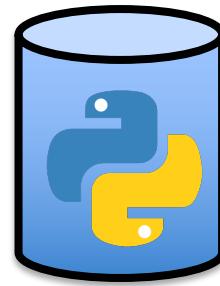




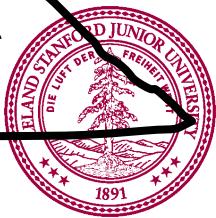
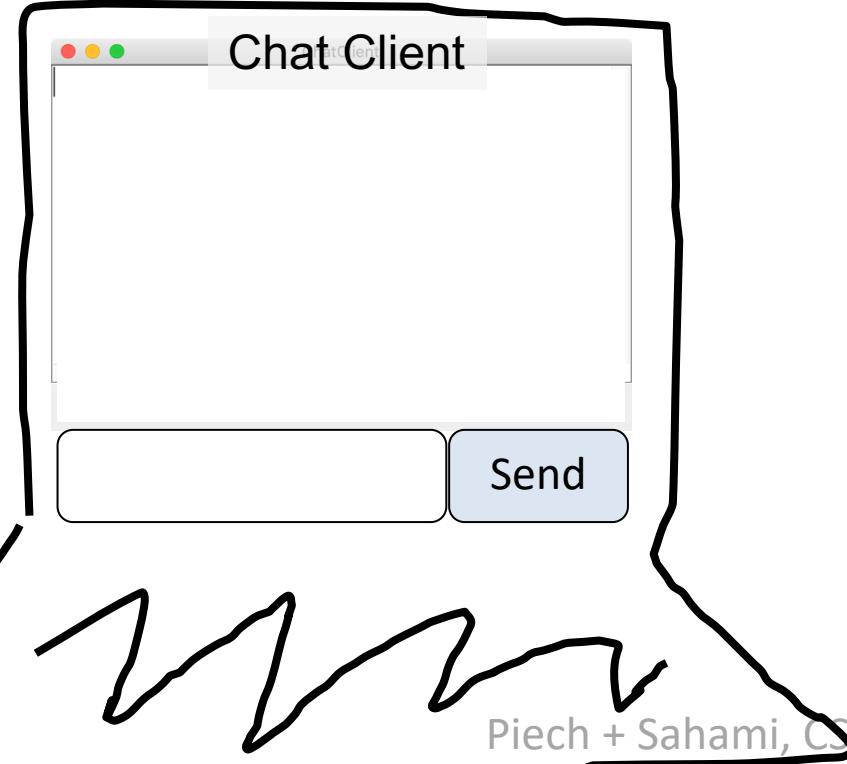
```
history = [  
    '[C] Hello world',  
    '[B] Im here too'  
]  
  
'["[B] Im here too"]'
```

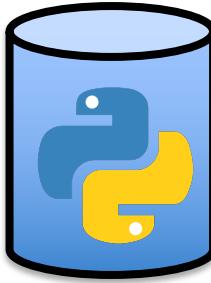


```
getMsgs  
{  
    'index' : 0  
}
```



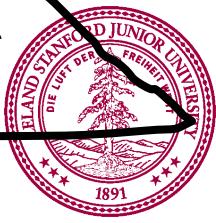
```
history = [  
    '[C] Hello world',  
    '[B] Im here too'  
]
```





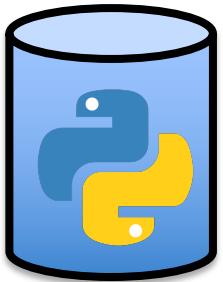
```
history = [
    '[C] Hello world',
    '[B] Im here too'
]
```

```
'[C] Hello world',
'[B] Im here too']'
```



Chat Server

Chat Server



addMsg
msg = *text*
user = *user*



getMsgs
index = *start_index*

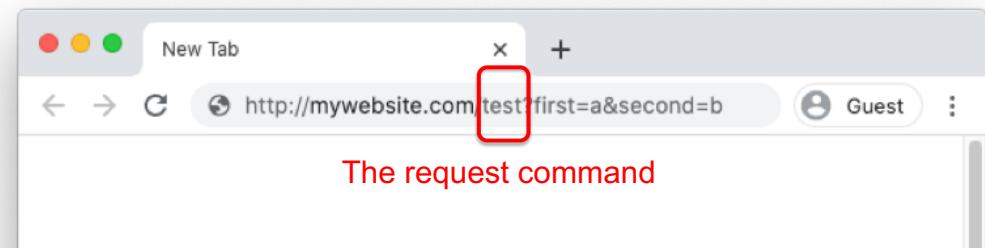
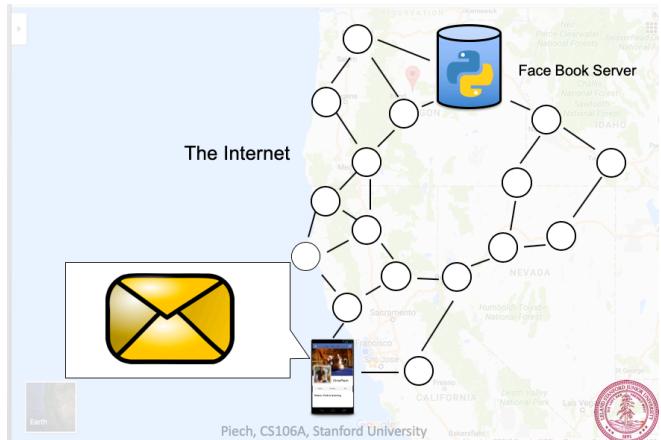


Learning Goals

1. Write a program that can respond to internet requests



Things we saw along the way



```
data_str = json.dumps(data)
```



Search 

- [scientists confirm chinese horseshoe bats responsible sars](#)
- [scientists confirm chinese bats responsible sars outbreak](#)
- [sixth person dies sars like virus](#)
- [new sars like virus shows person person transmission](#)
- [issues alert sars type infection](#)
- [fuel prices bigger problem sep 11 sars](#)
- [sars expert dies](#)
- [royal society inducts sars expert](#)
- [sars probably originated bats](#)
- [bats may source sars](#)

Search Engine

