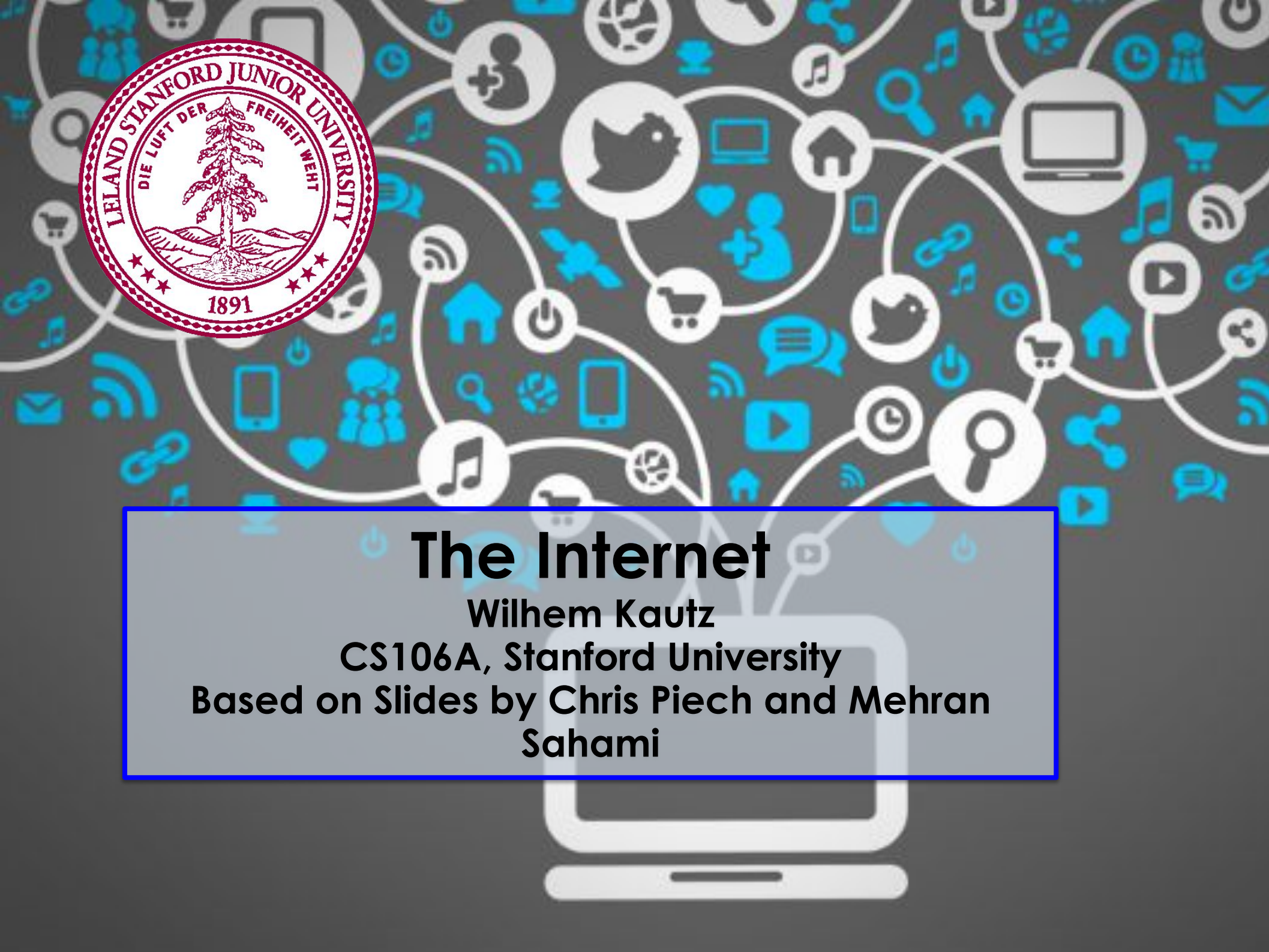
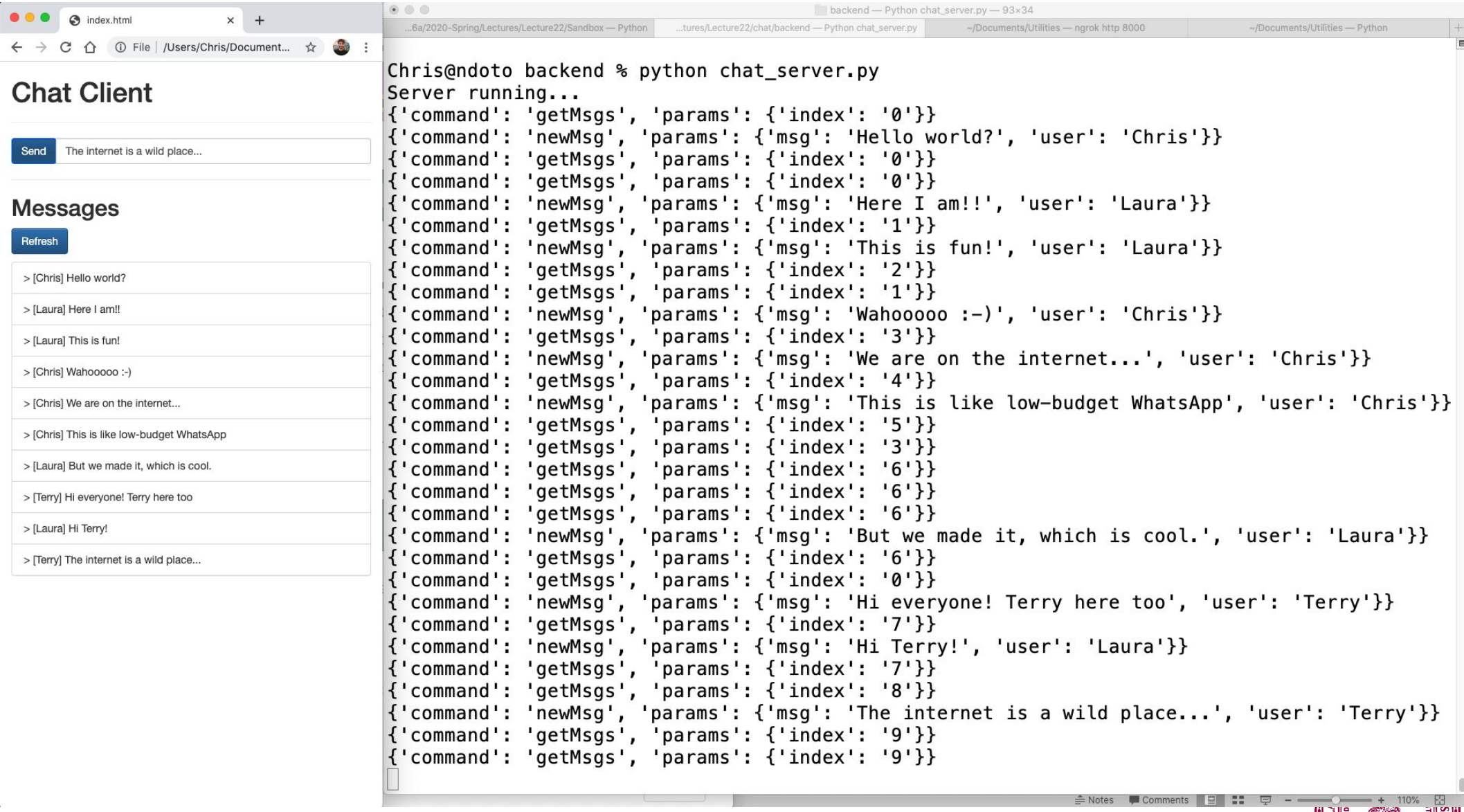


The Internet

Wilhem Kautz
CS106A, Stanford University
Based on Slides by Chris Piech and Mehran Sahami

First, a cool demo



<review>

Classes Review

Dog.py

```
class Dog:
    def __init__(self):
        self.times_barked = 0

    def bark(self):
        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(self):
        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(self):
        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(self):
        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__)
```

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

Classes Review

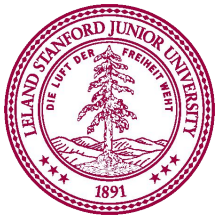
Dog.py

```
class Dog:
    def __init__(self):
        self.times_barked = 0

    def bark(self):
        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

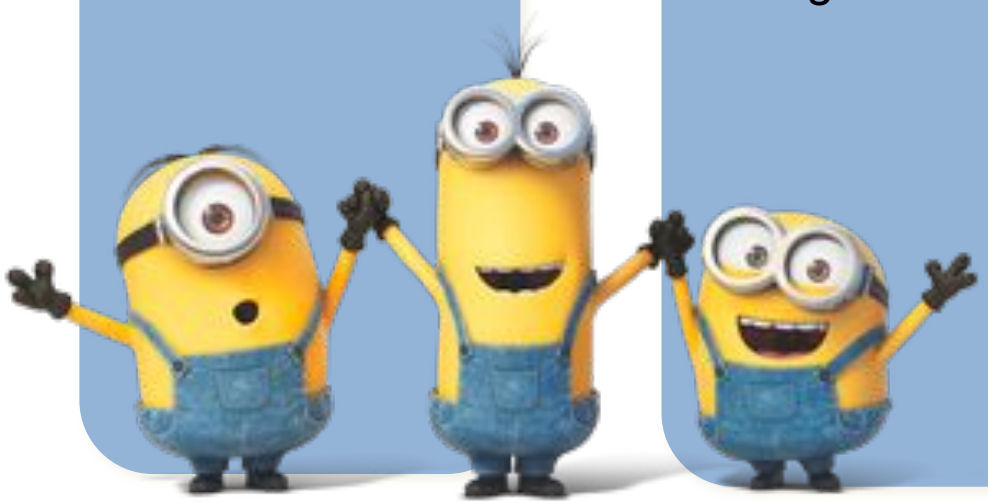
Song

Playlist

User

Song Player

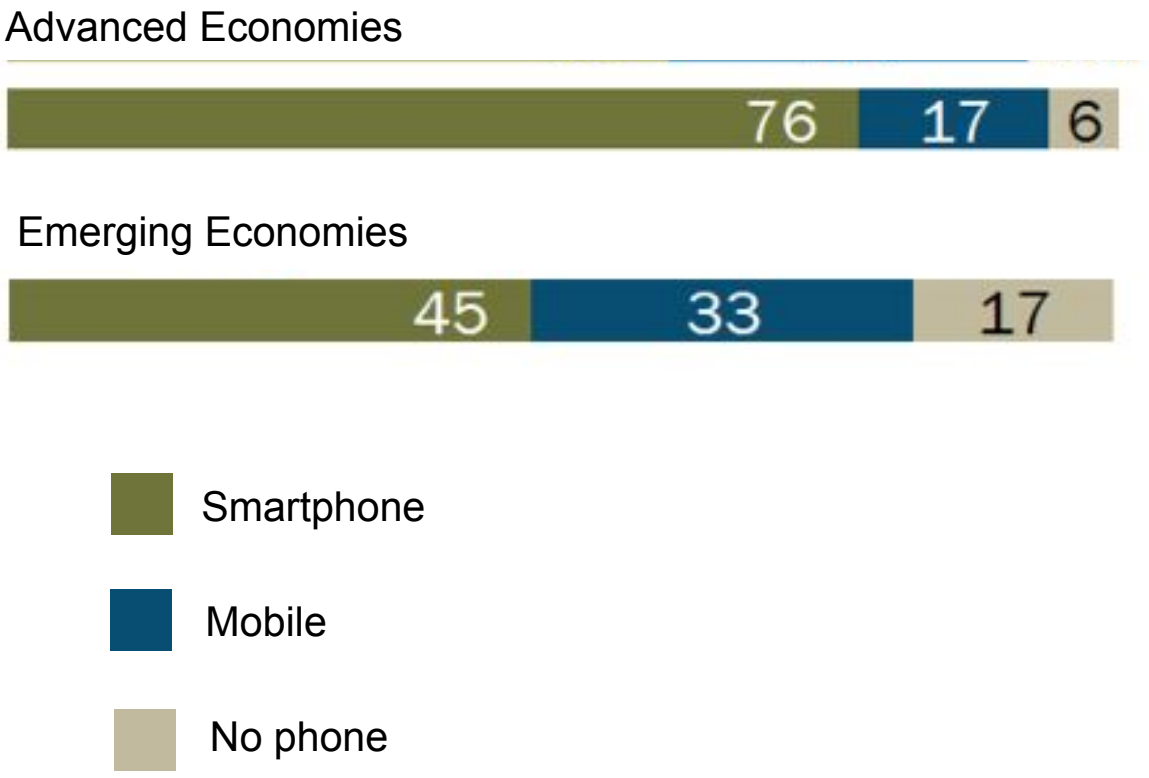
Song Retriever

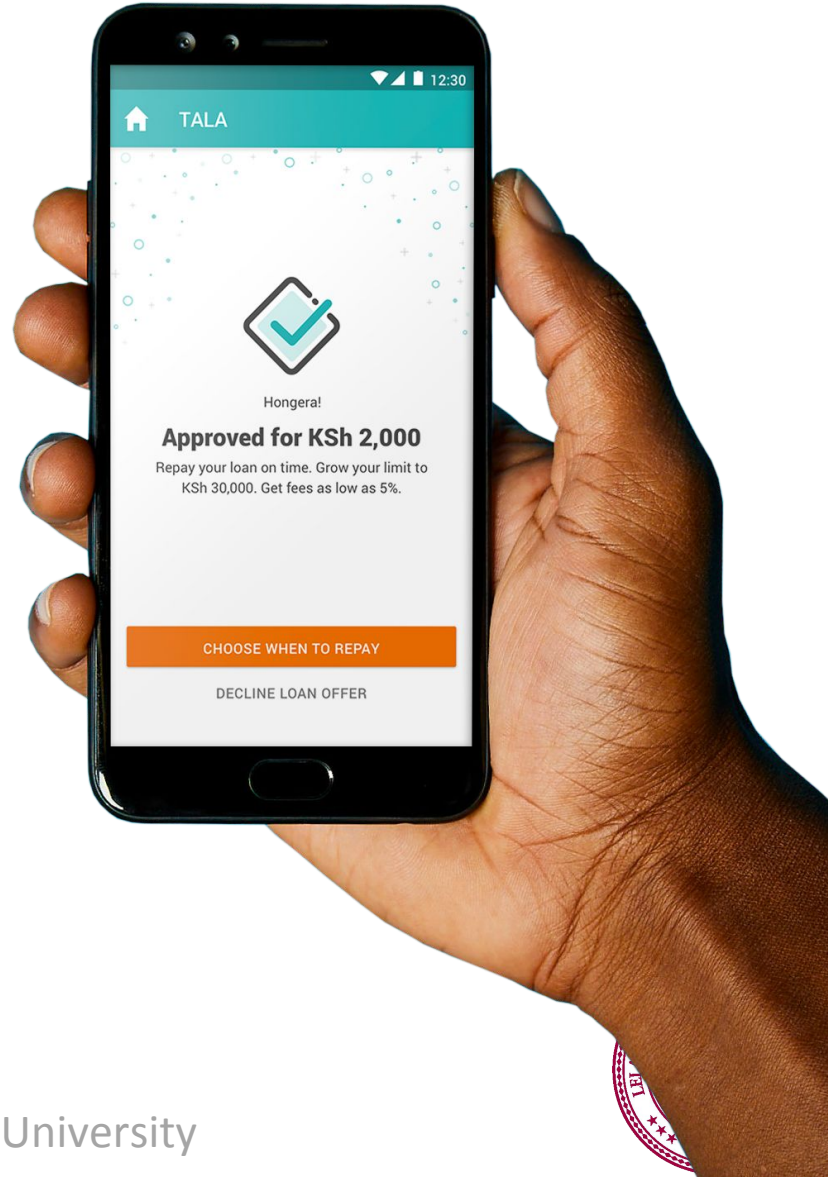


</ review>

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

Smart Phone Access

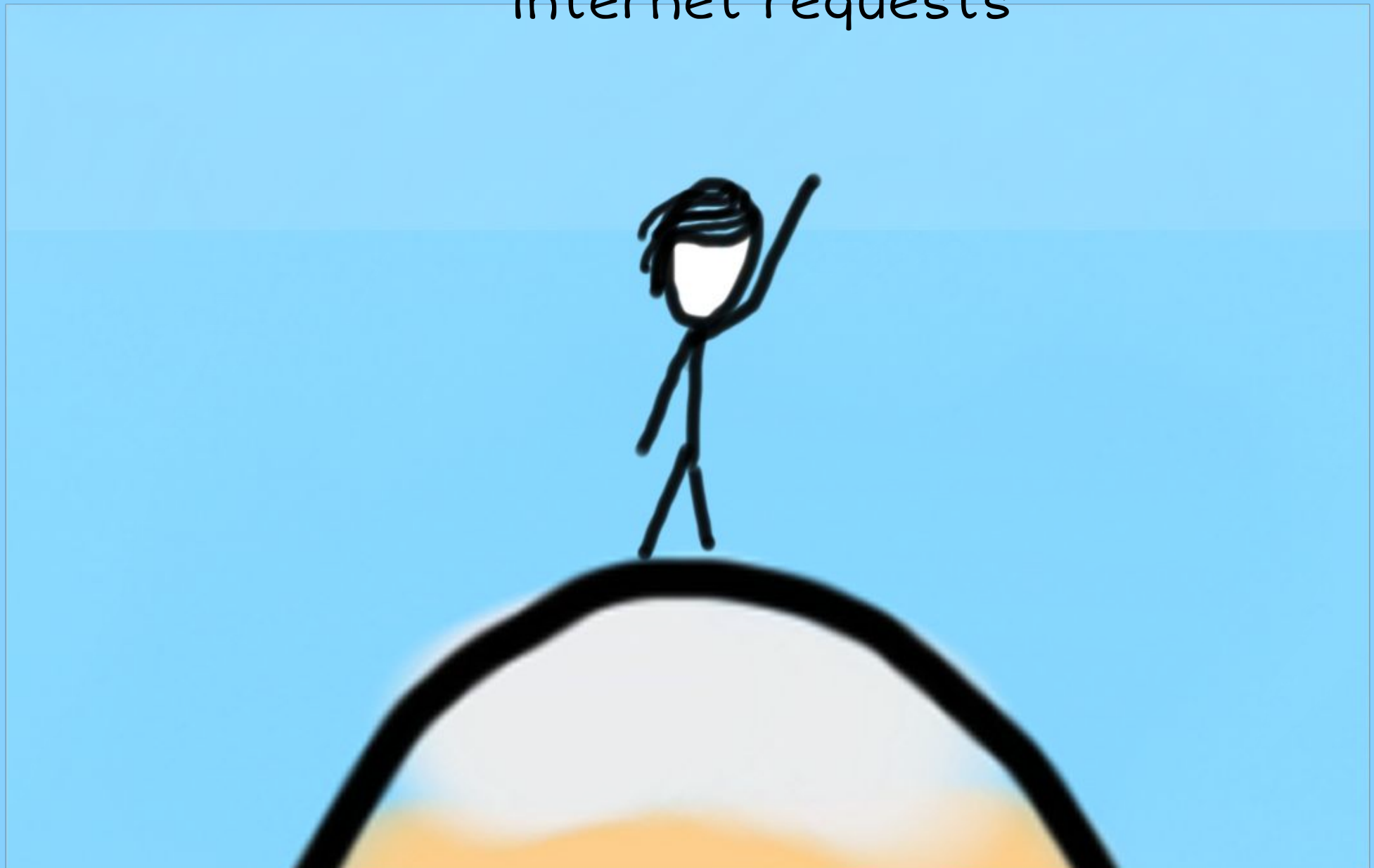




For the fourth time ever in CS106A:

Learning Goals

1. Write a program that can respond to internet requests



How does your phone communicate with facebook?

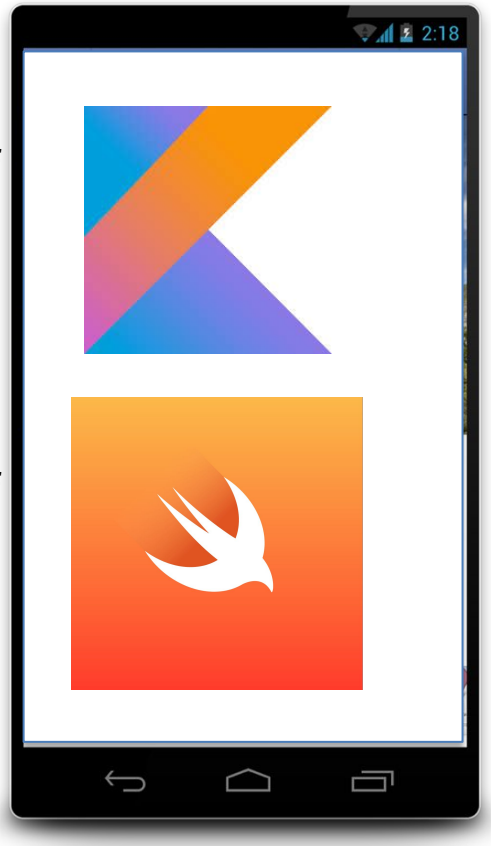
The program on your **phone**
talks to the program at
Facebook

JavaScript with HTML are the languages of websites

Face Book Server

Kotlin is the language of Android phones

Swift is the language of Apple phones



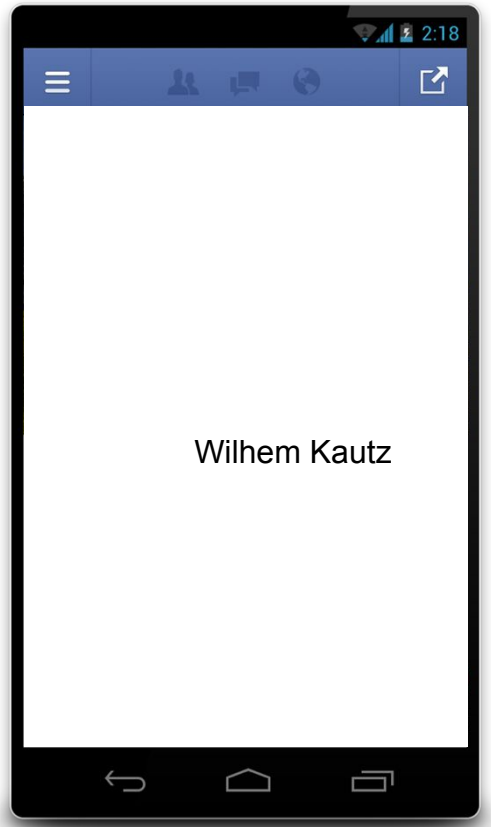
Face Book Server

Is this legit?

wkautz@stanford.edu
is now logged in

Face Book Server

Send me the full name for wkautz@stanford.edu



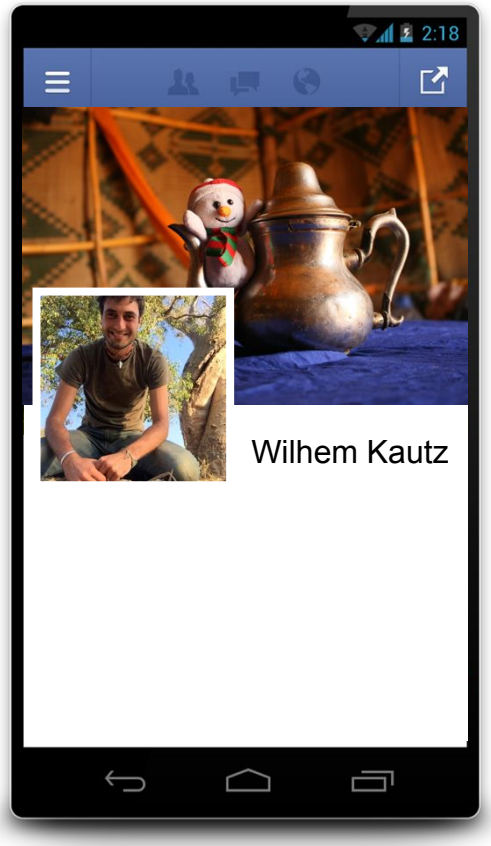
"Wilhem Kautz"

Face Book Server

```
Send me the cover photo for
    wkautz@stanford.edu
```

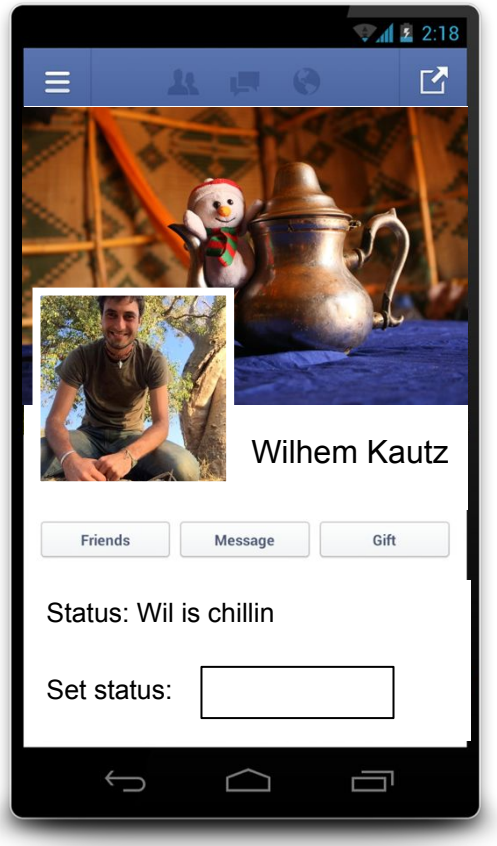
Face Book Server

Send the profile photo for wkautz@stanford.edu



Face Book Server

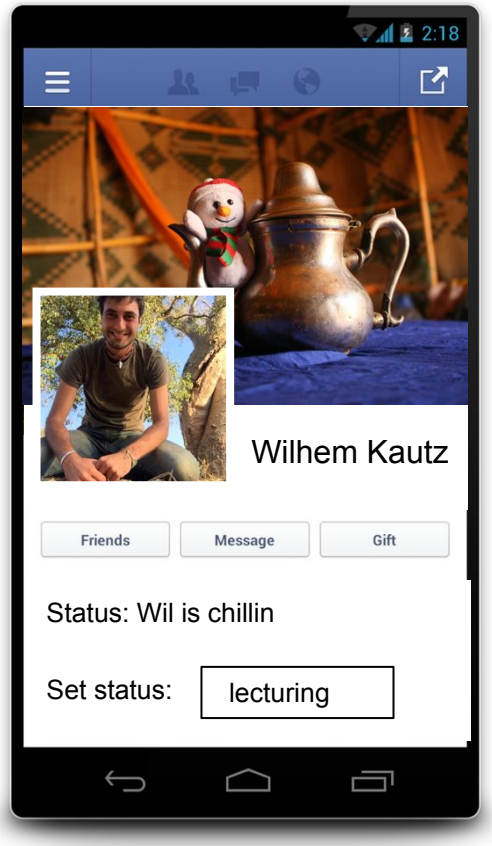
Send the status for wkautz@stanford.edu



"chillin"

Face Book Server

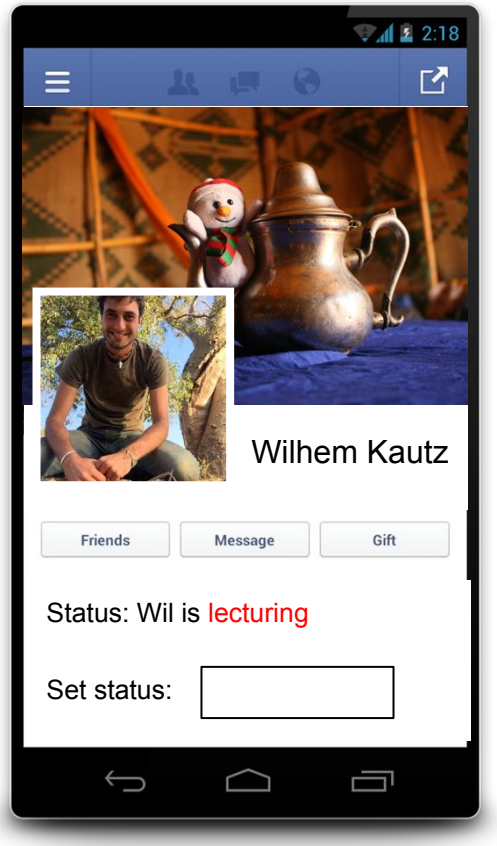
Set the status for wkautz@stanford.edu to be "lecturing"



"success"

Face Book Server

Send me the status for wkautz@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*)

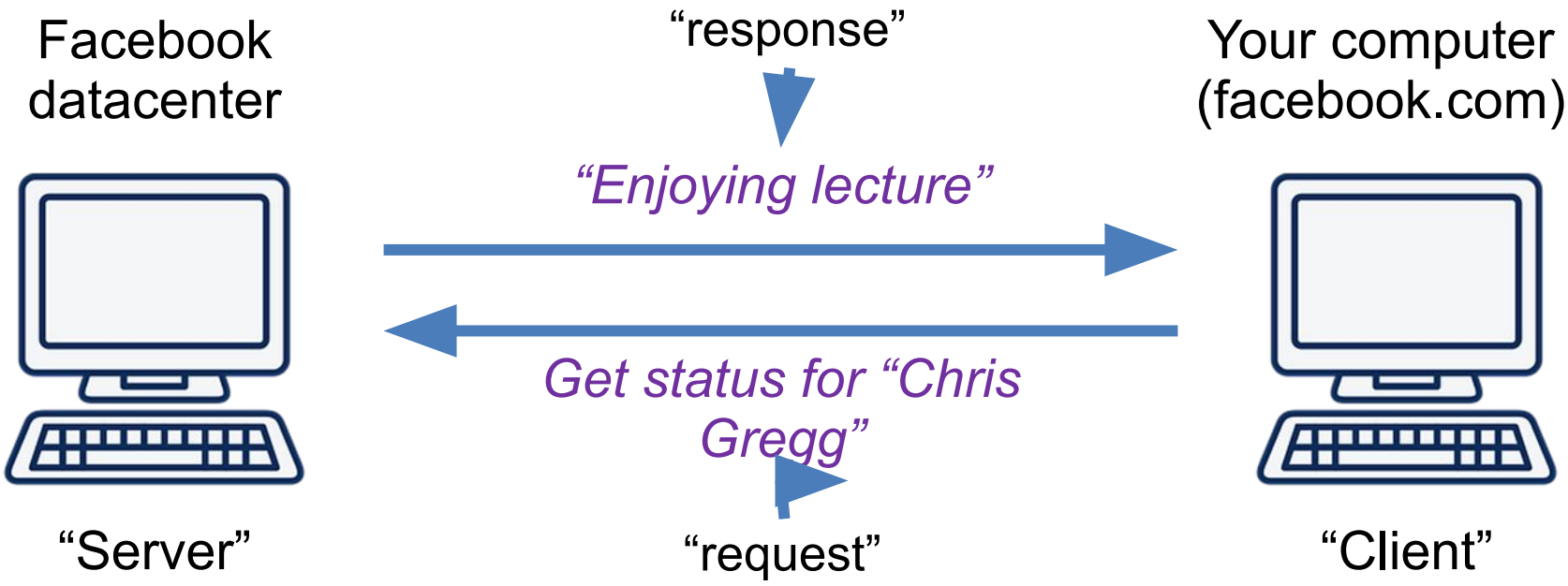
Background: The Internet

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

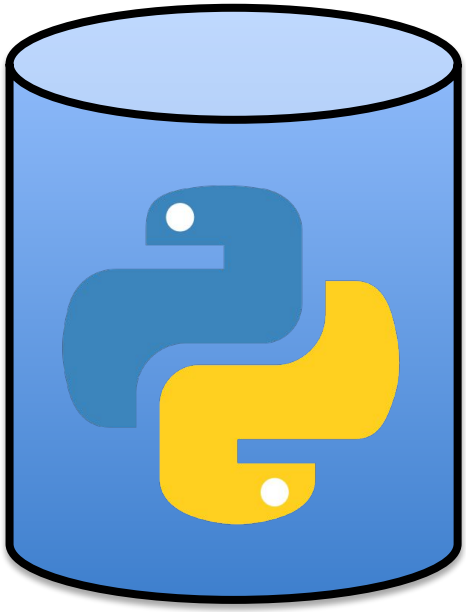


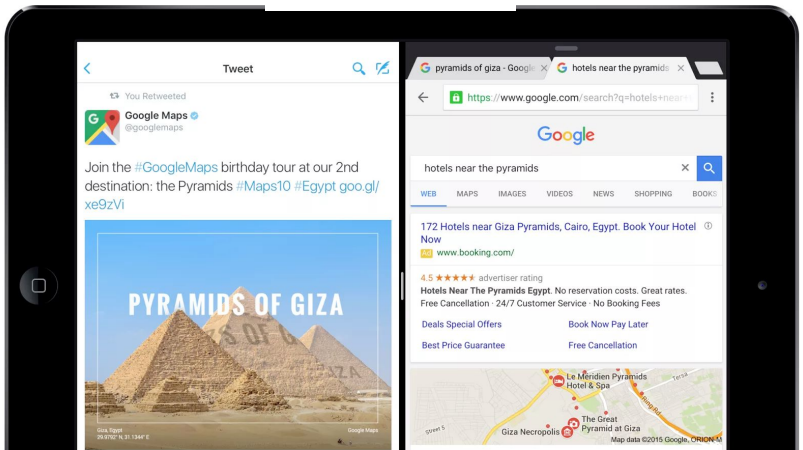
The internet is just many programs sending messages (as *Strings*)

There are two types of internet programs. Servers and Clients

Internet 101

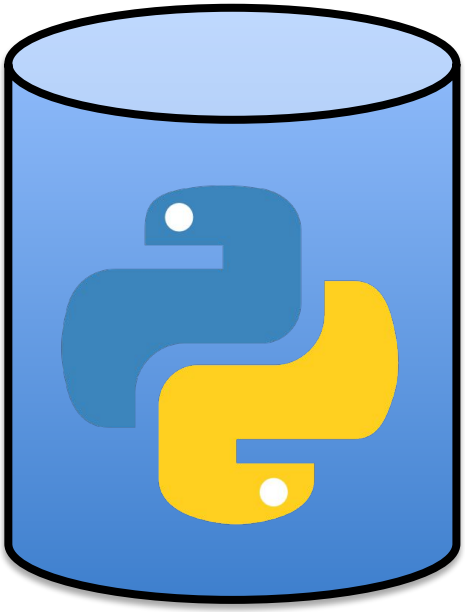
Computers on the internet



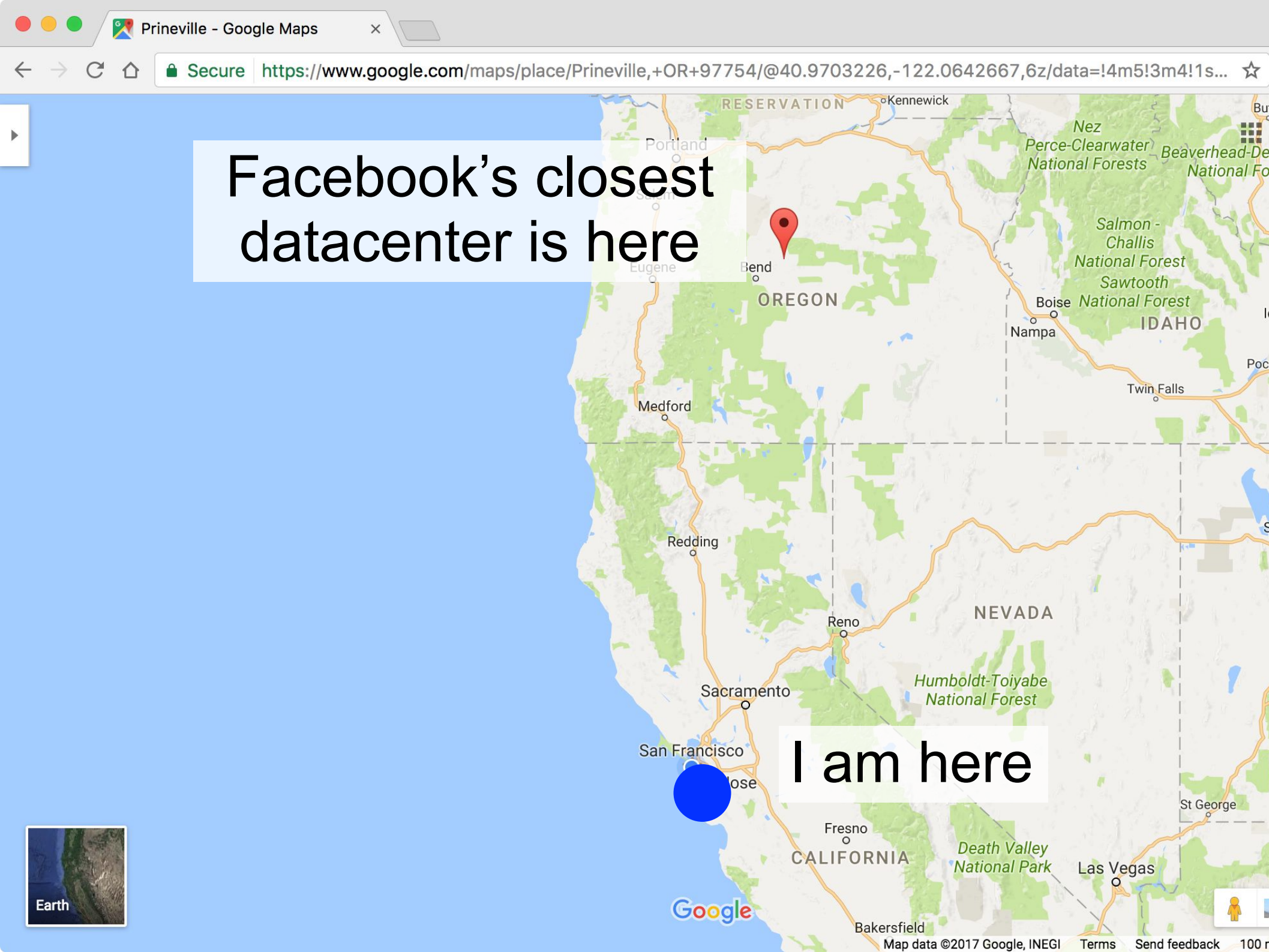


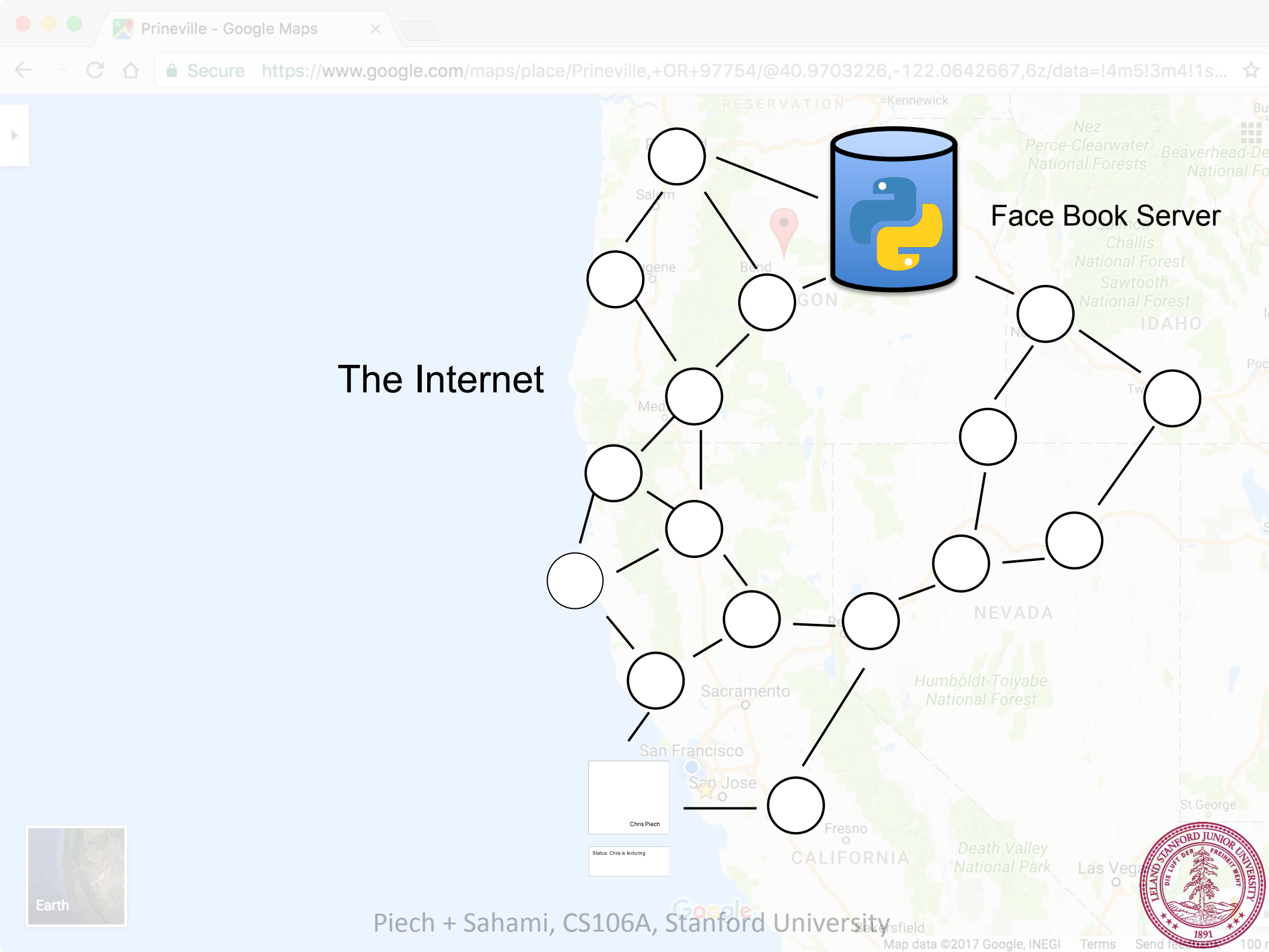
Servers are computers (running code)

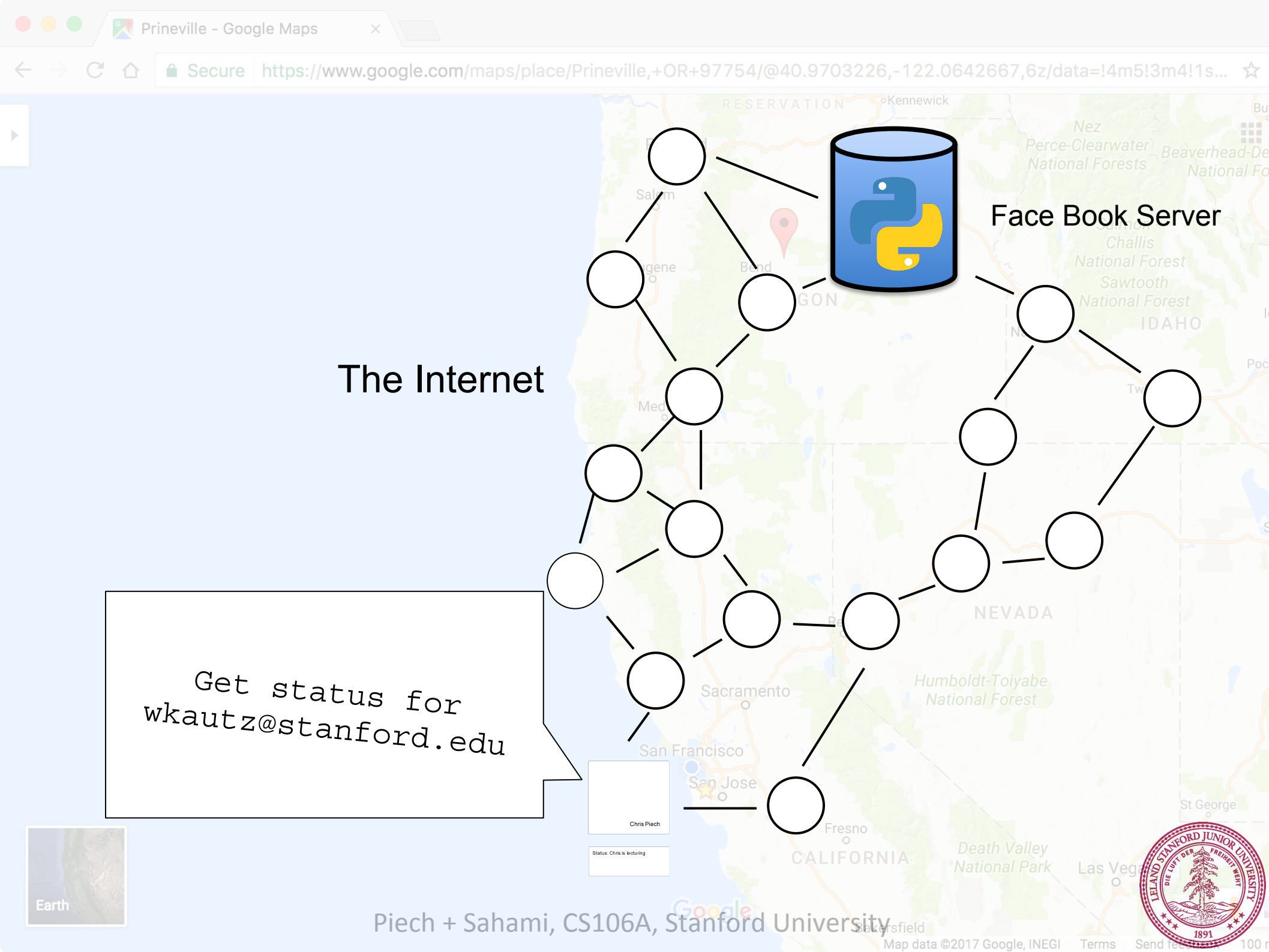
Face Book Server



=

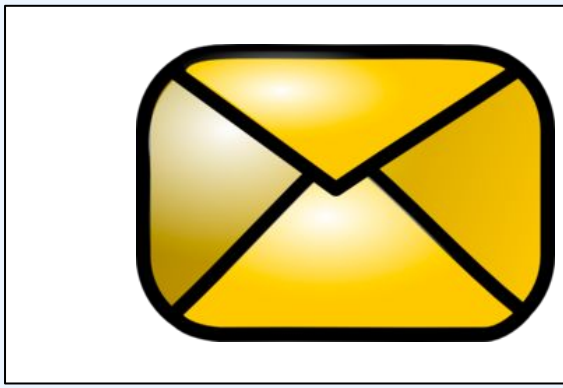

Prineville - Google Maps
Secure
https://www.google.com/maps/place/Prineville,+OR+97754/@40.9703226,-122.0642667,6z/data=!4m5!3m4!1s...
Facebook's closest datacenter is here
I am here
RESERVATION
Kennewick
Portland
Nez Perce-Clearwater National Forests
Salmon - Challis National Forest
Sawtooth National Forest
Eugene
Bend
OREGON
Boise
Nampa
IDAHO
Twin Falls
Medford
Redding
NEVADA
Reno
Humboldt-Toiyabe National Forest
Sacramento
San Francisco
St George
Fresno
CALIFORNIA
Death Valley National Park
Las Vegas
Bakersfield
Earth
Google
Map data ©2017 Google, INEGI
Terms
Send feedback


Prineville - Google Maps
Secure
https://www.google.com/maps/place/Prineville,+OR+97754/@40.9703226,-122.0642667,6z/data=!4m5!3m4!1s...
Face Book Server
The Internet
Chris Piech
Status: Chris is lecturing
Earth
Piech + Sahami, CS106A, Stanford University


Face Book Server
The Internet
Get status for wkautz@stanford.edu
Chris Piech
Status: Chris is lecturing
Earth

Prineville - Google Maps

Secure https://www.google.com/maps/place/Prineville,+OR+97754/@40.9703226,-122.0642667,6z/data=!4m5!3m4!1s...



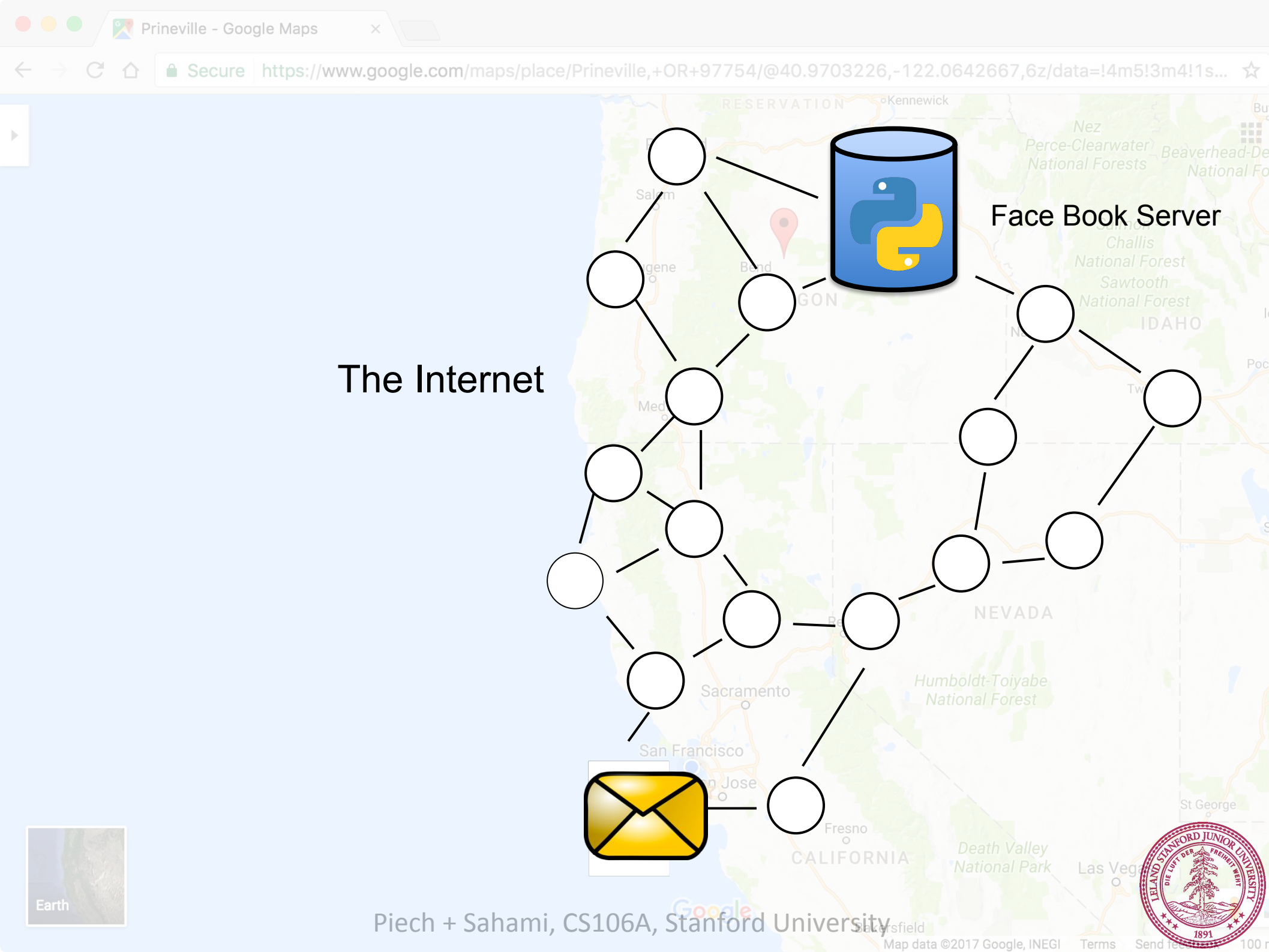
Chris Piech

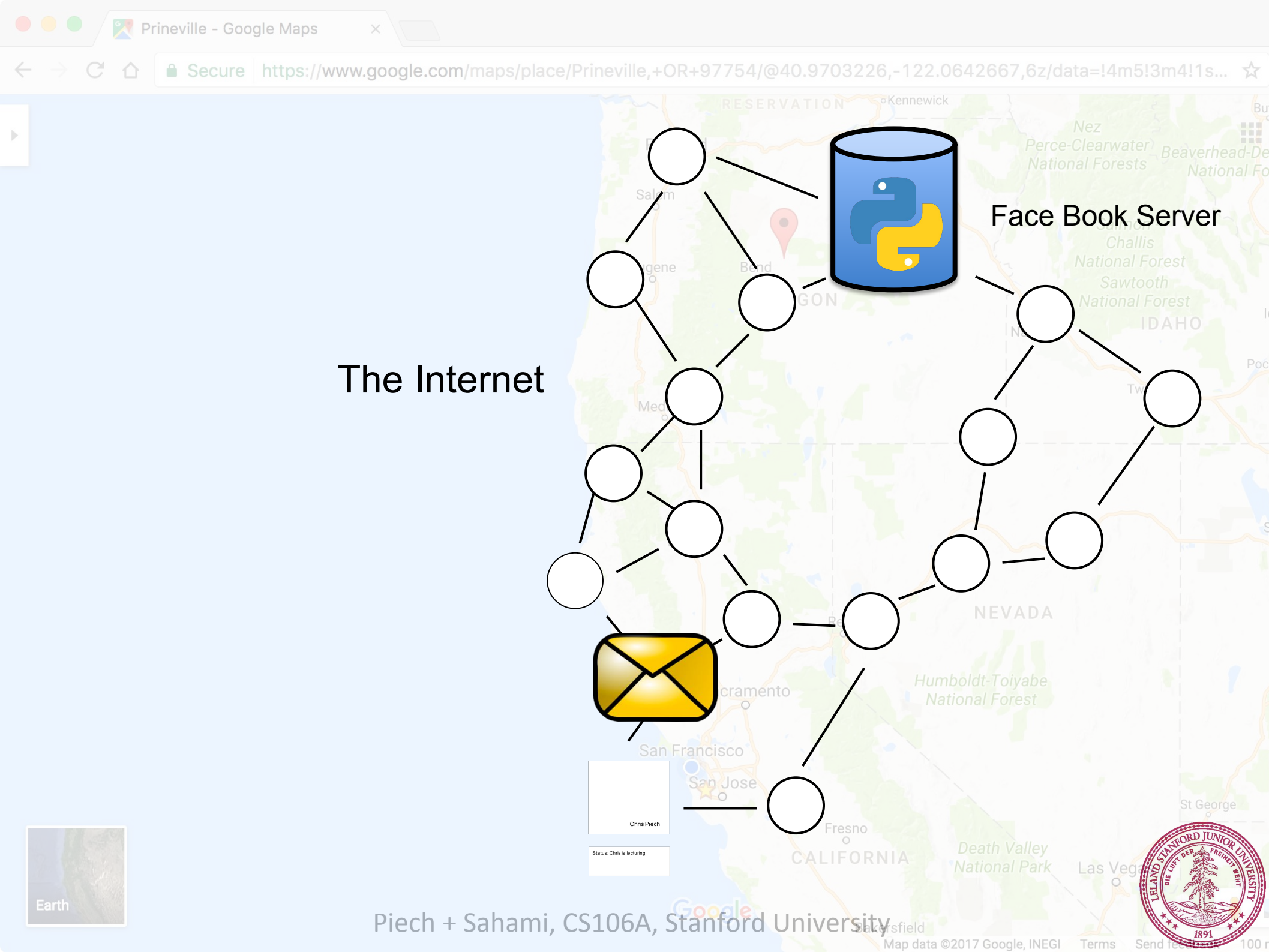
Status: Chris is lecturing

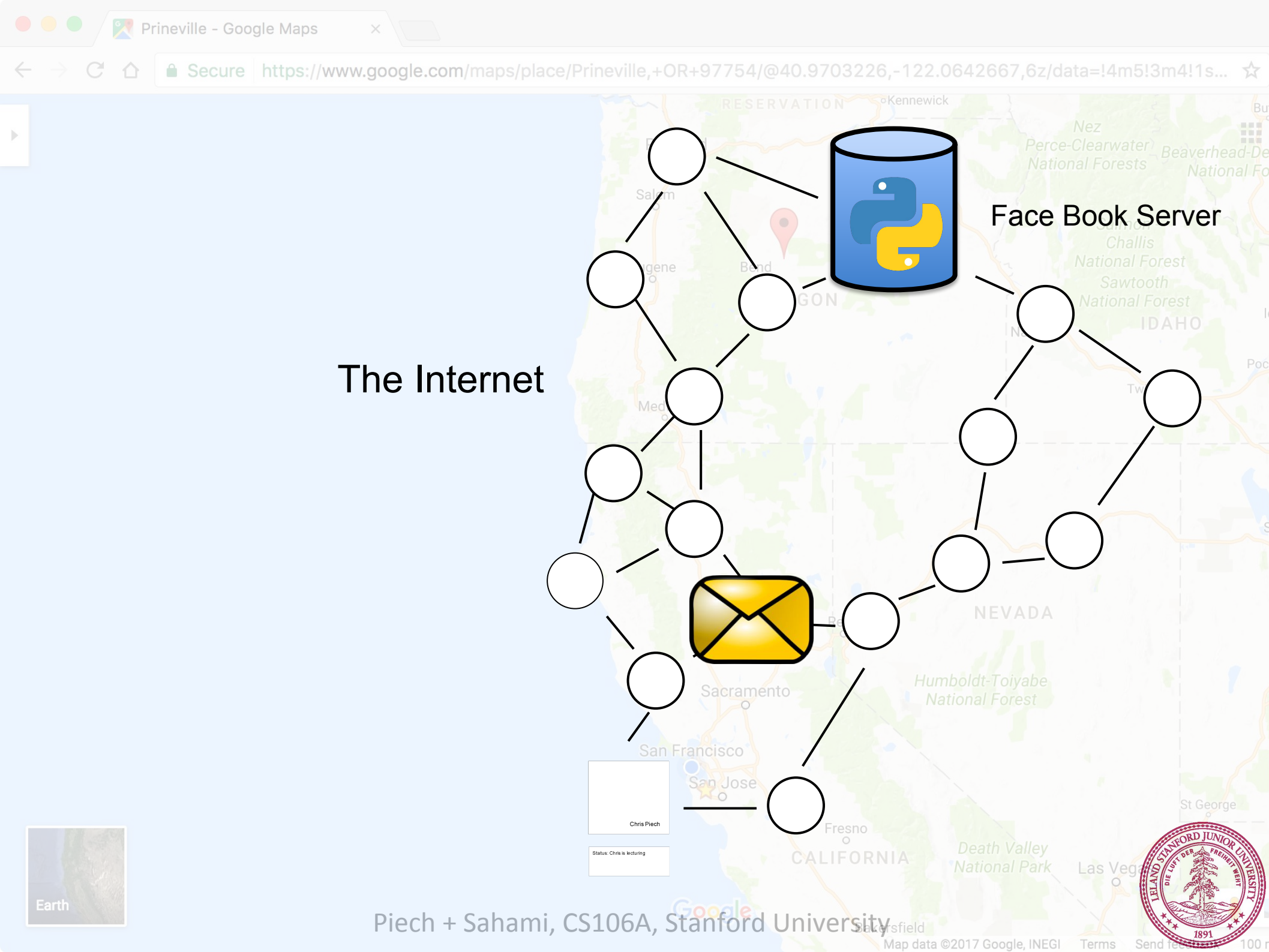
Face Book Server

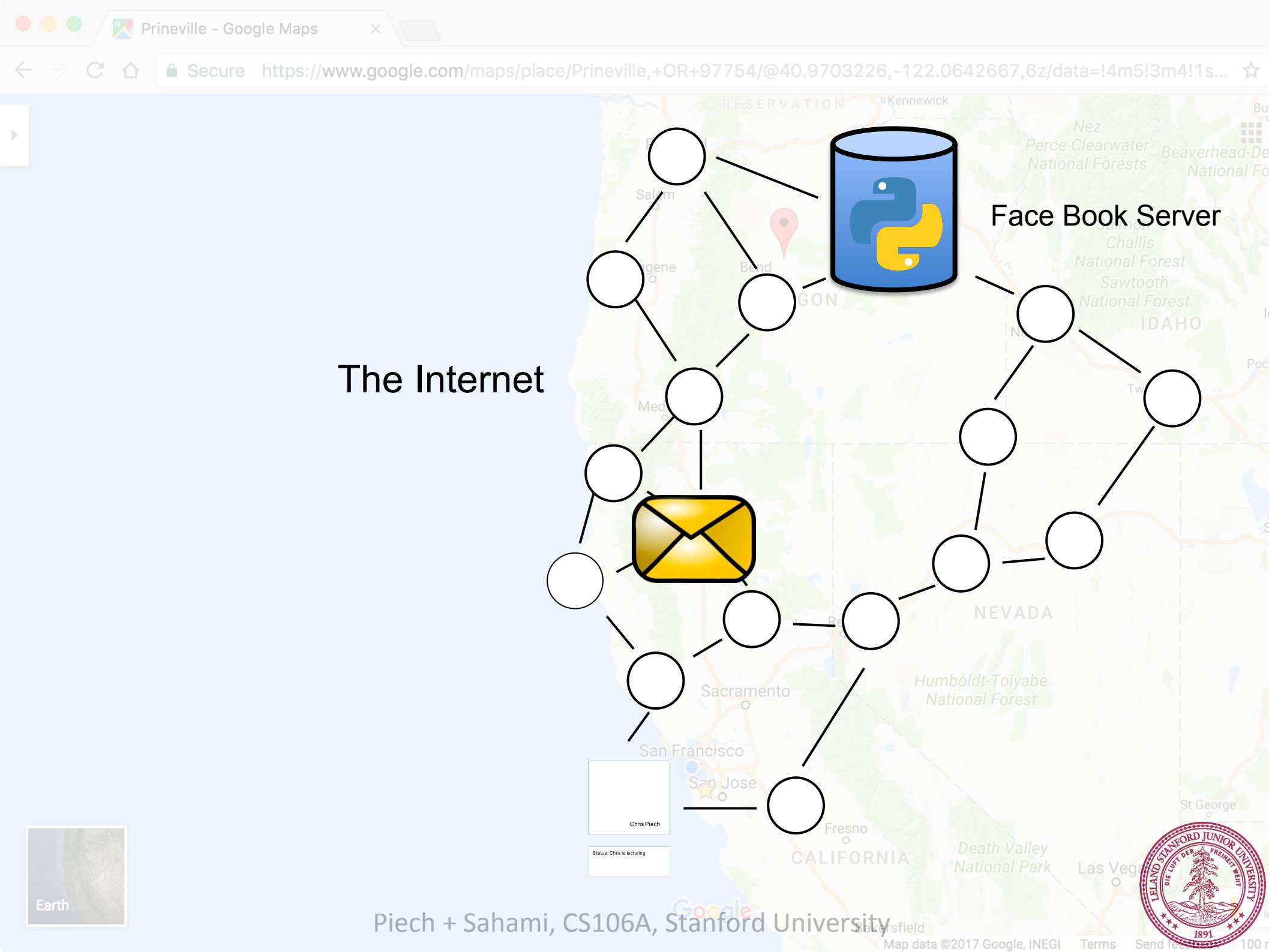
The Internet

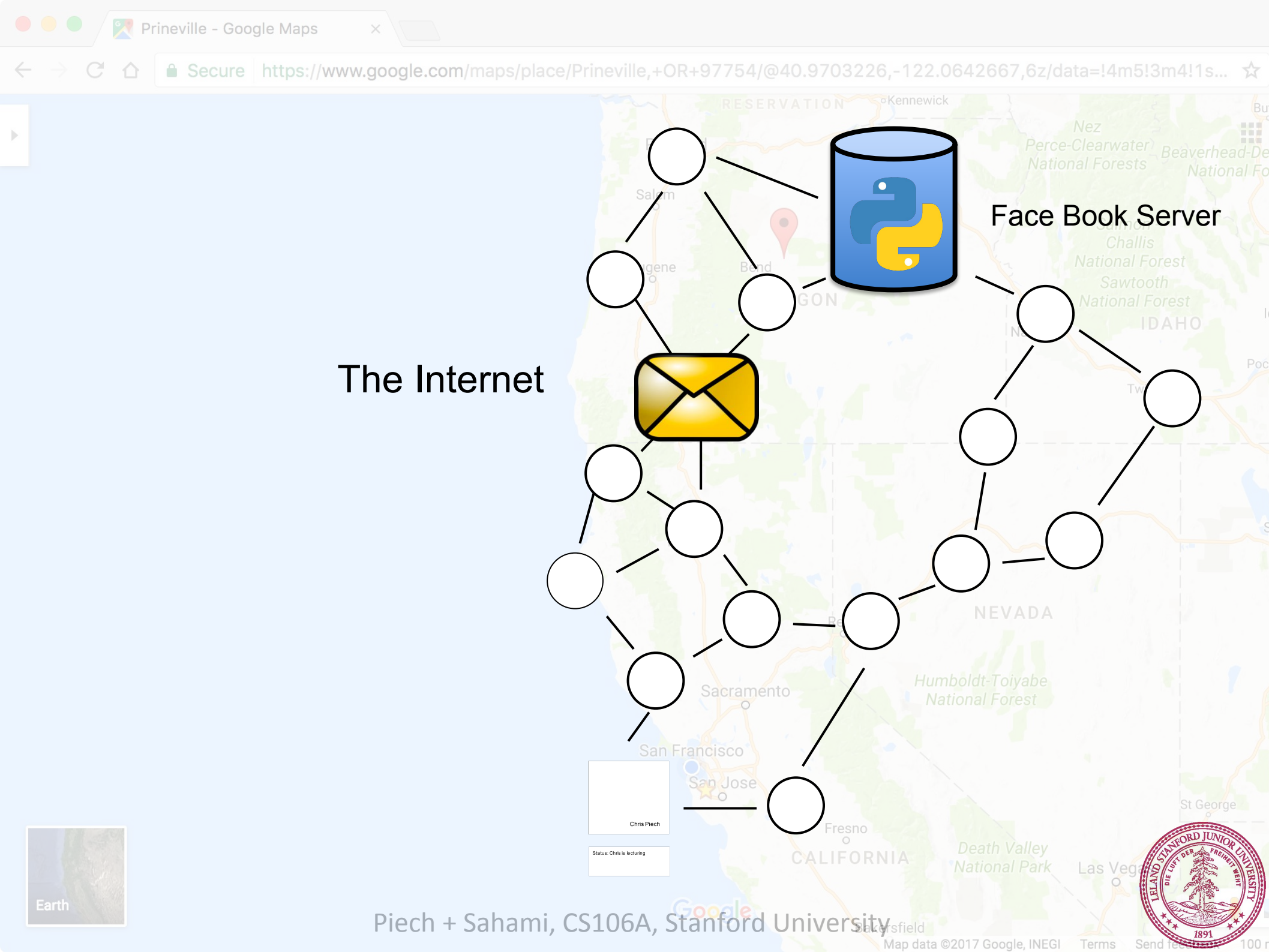
Earth

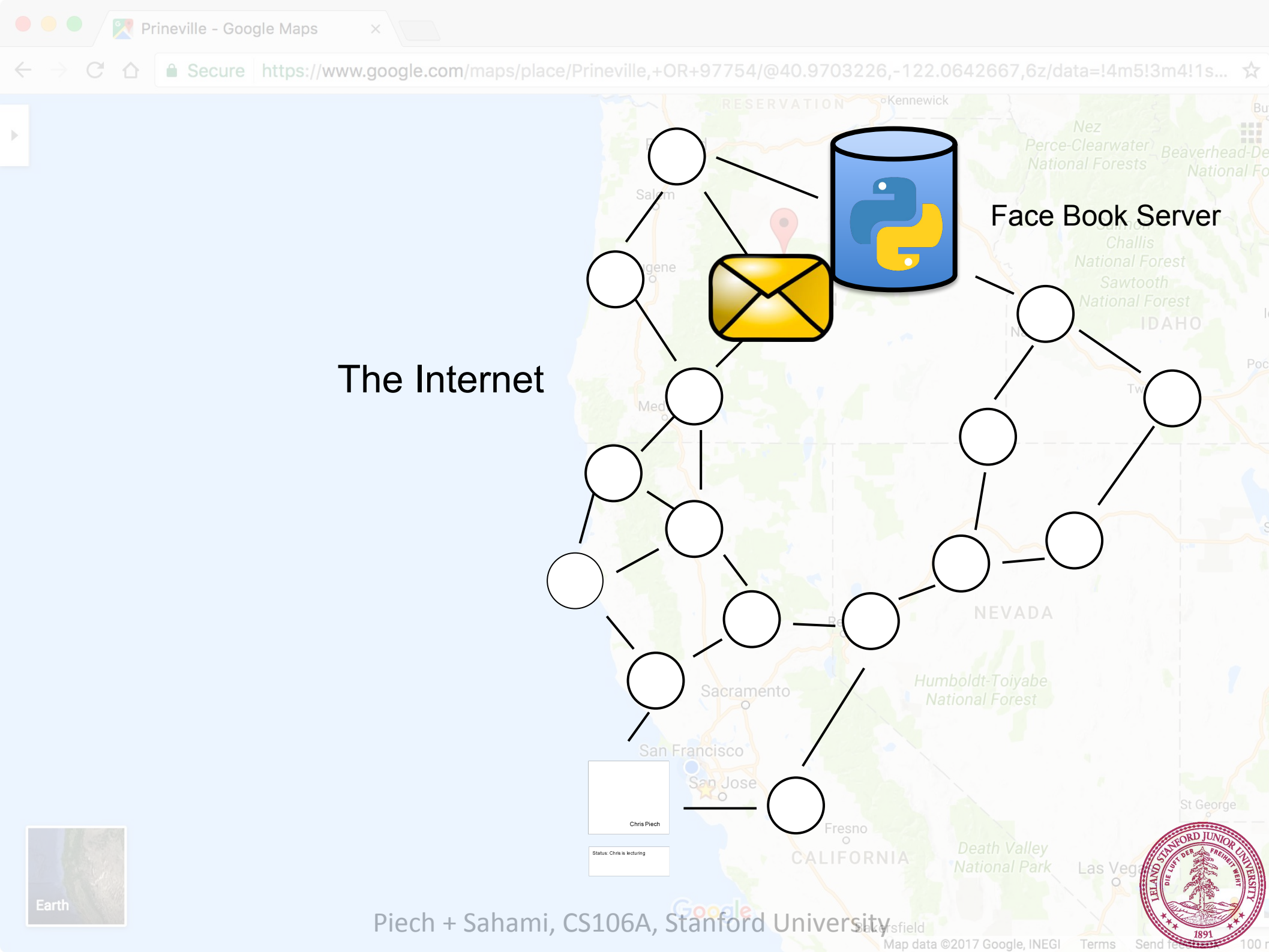

Face Book Server
The Internet

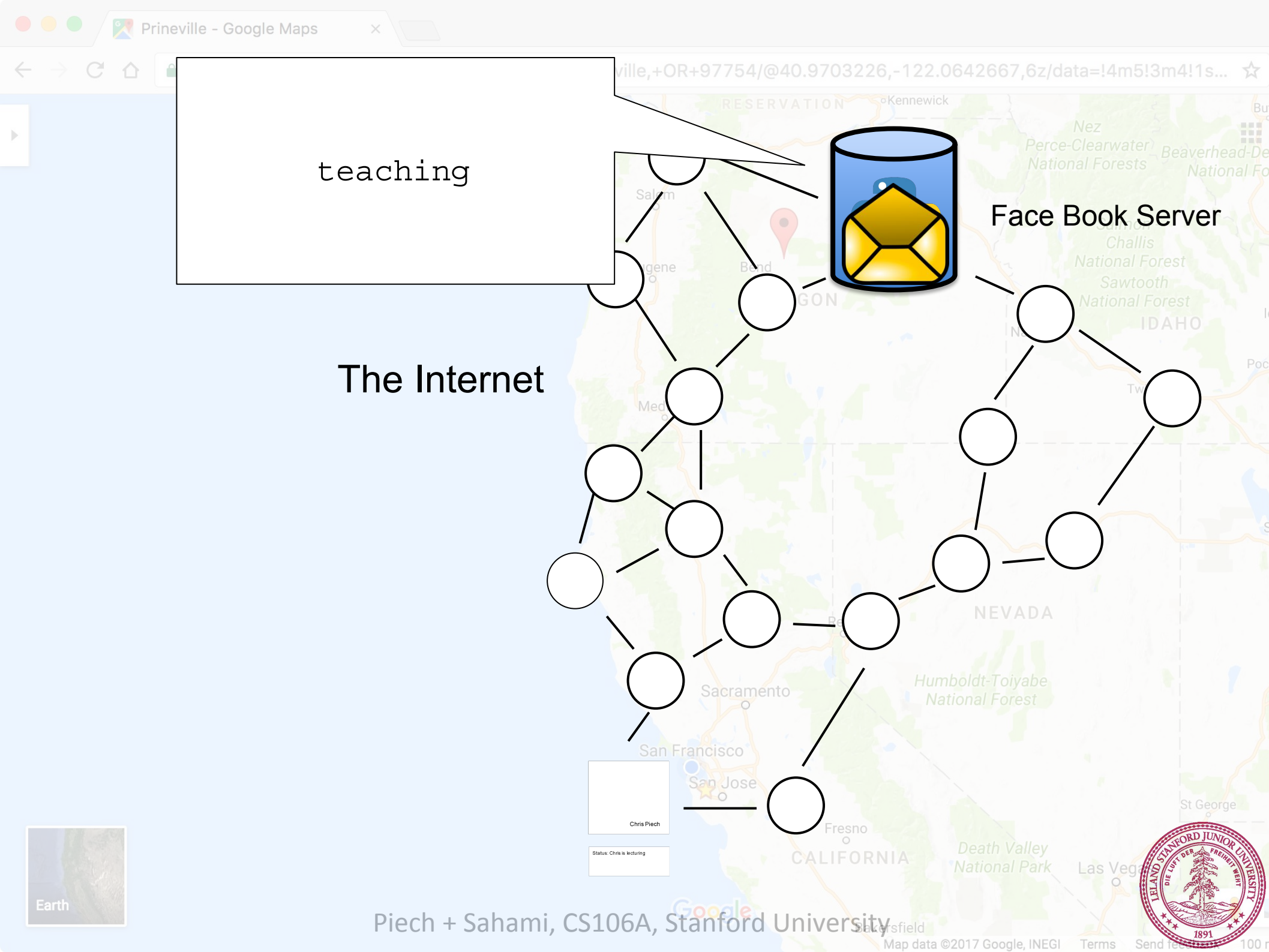

Face Book Server
The Internet
Chris Piech
Status: Chris is lecturing
Earth

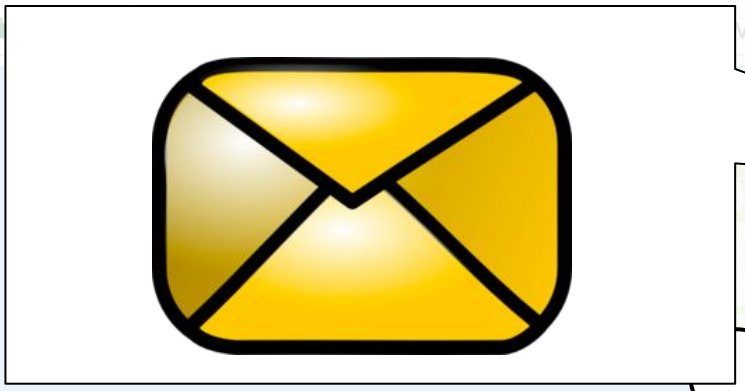

Face Book Server
The Internet
Chris Piech
Status: Chris is lecturing
Earth


Prineville - Google Maps
Secure
https://www.google.com/maps/place/Prineville,+OR+97754/@40.9703226,-122.0642667,6z/data=!4m5!3m4!1s...
Face Book Server
The Internet
Chris Piech
Status: Chris is lecturing
Earth
Piech + Sahami, CS106A, Stanford University


Face Book Server
The Internet
Chris Piech
Status: Chris is lecturing
Earth

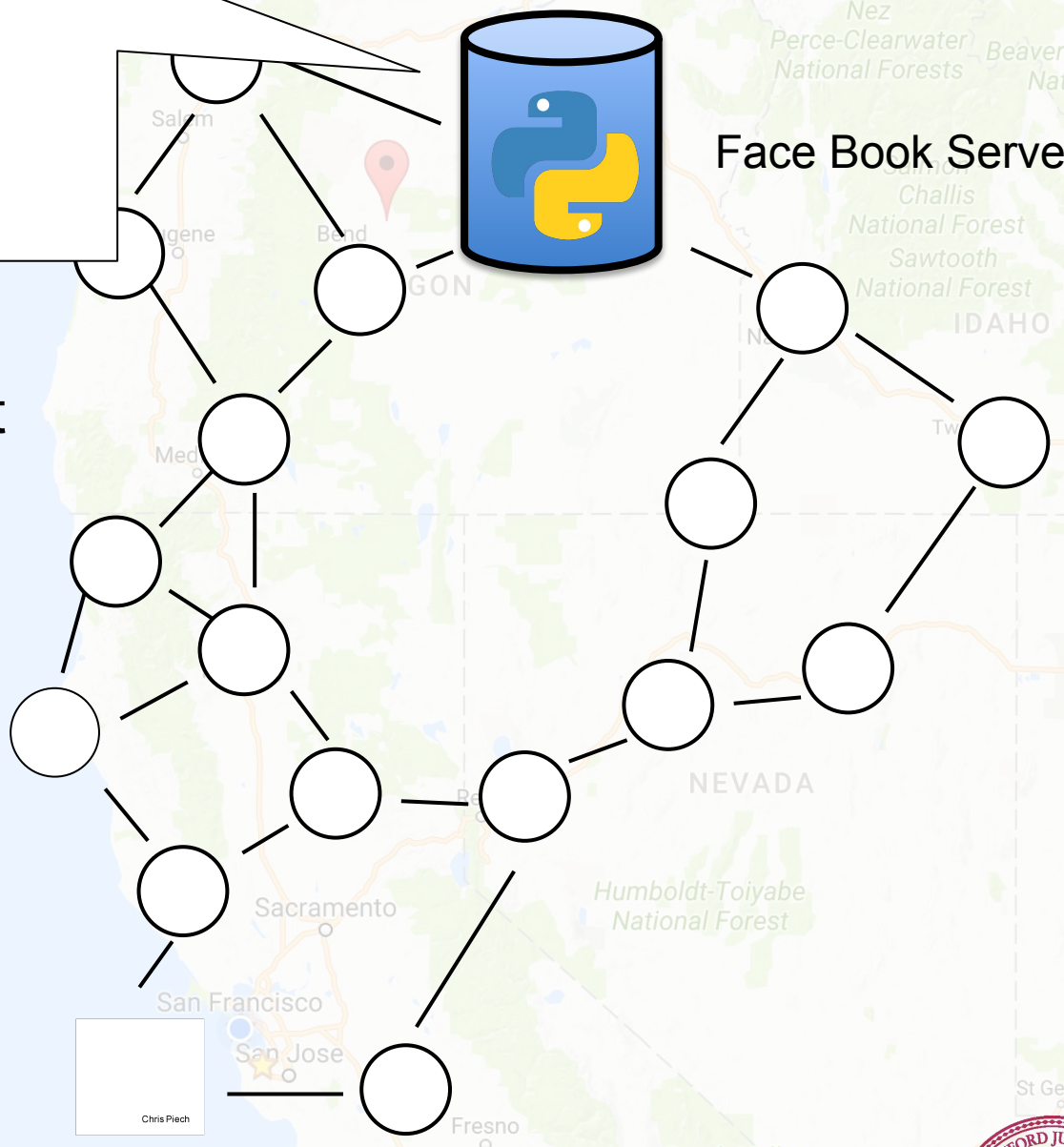

Face Book Server
The Internet
Chris Piech
Status: Chris is lecturing
Earth

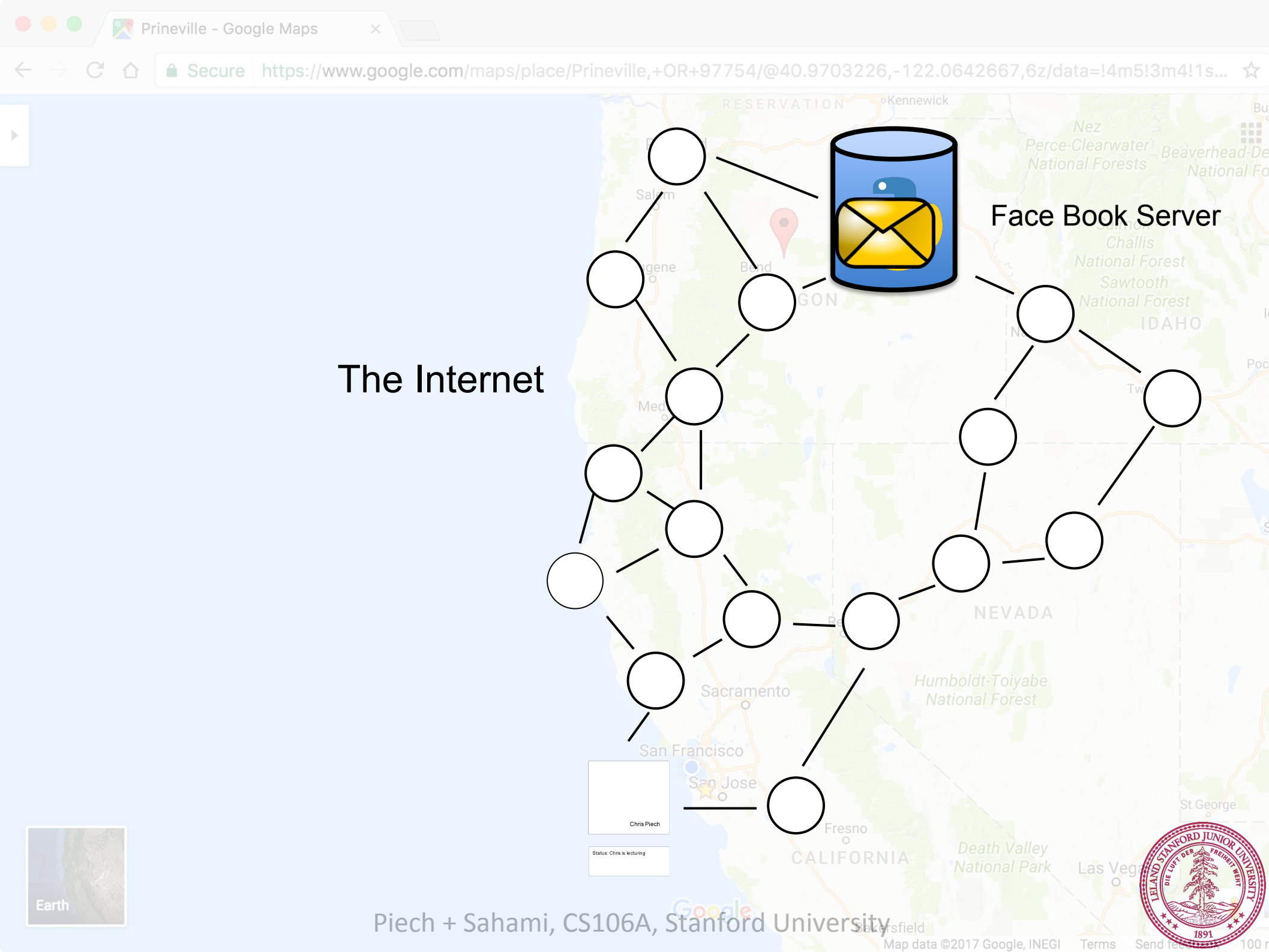

teaching
Face Book Server
The Internet
Chris Piech
Status: Chris is lecturing
Earth
Piech + Sahami, CS106A, Stanford University



Face Book Server

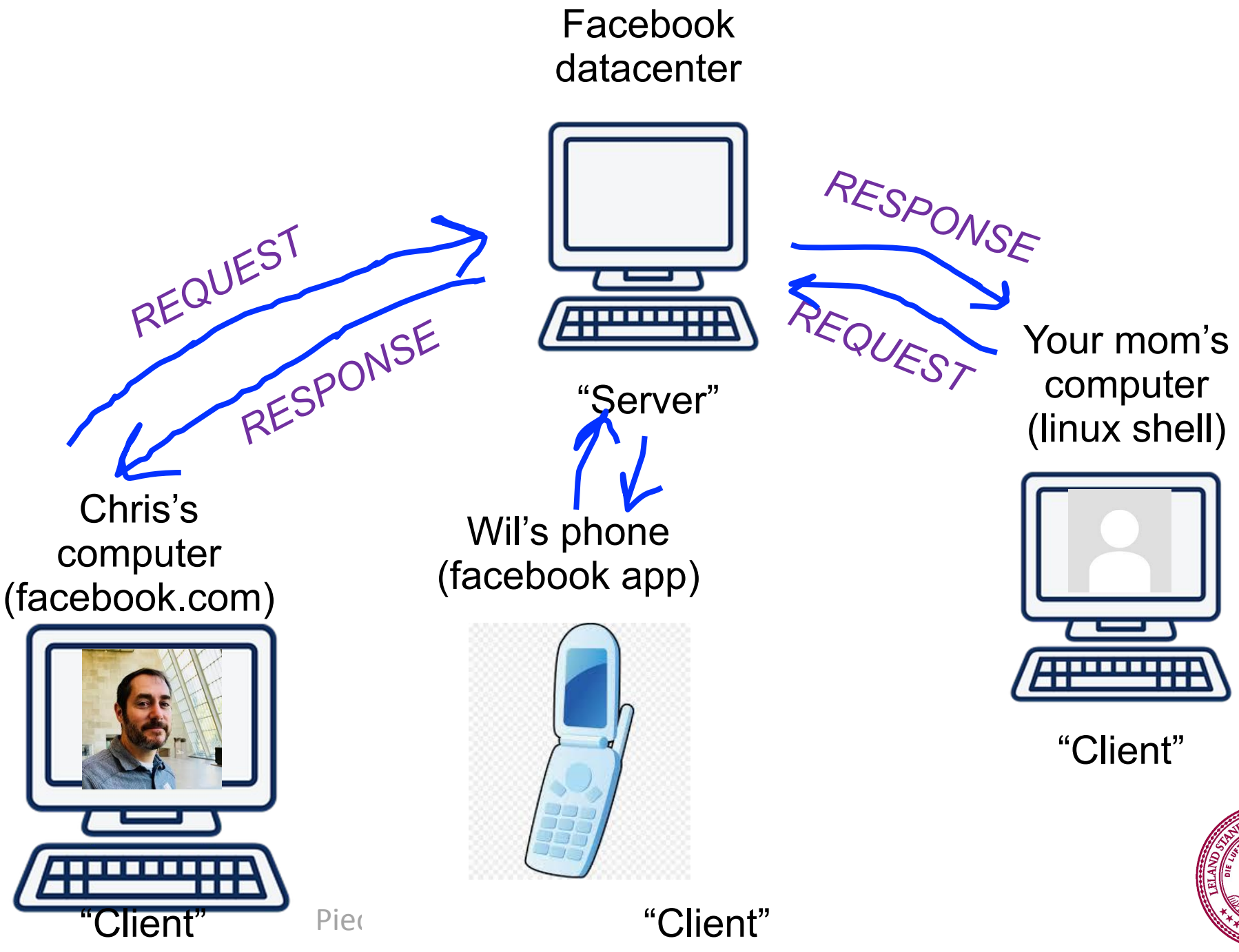
The Internet



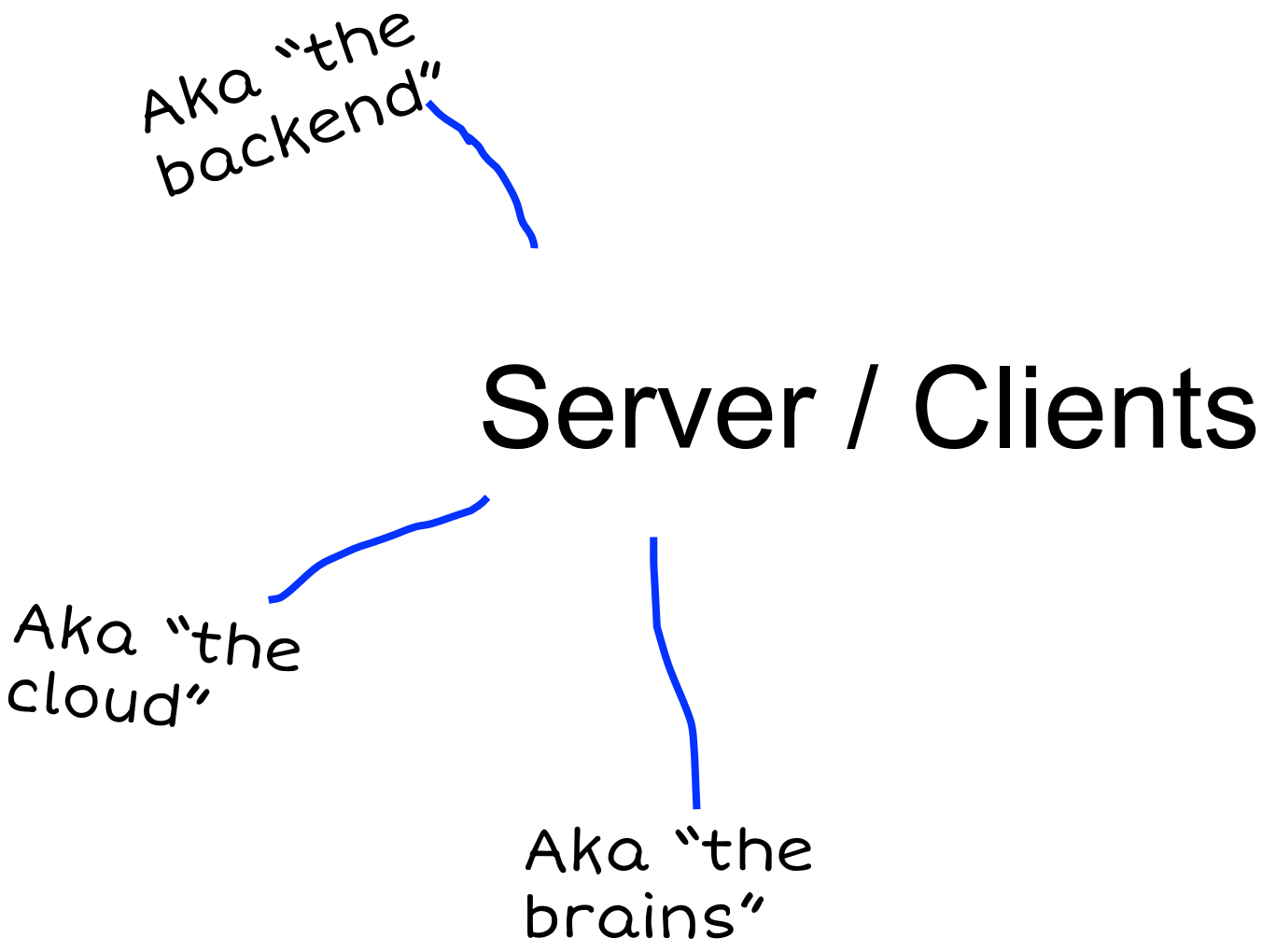

Face Book Server
The Internet
Chris Piech
Status: Chris is lecturing
Earth

Many computers can connect to the same server

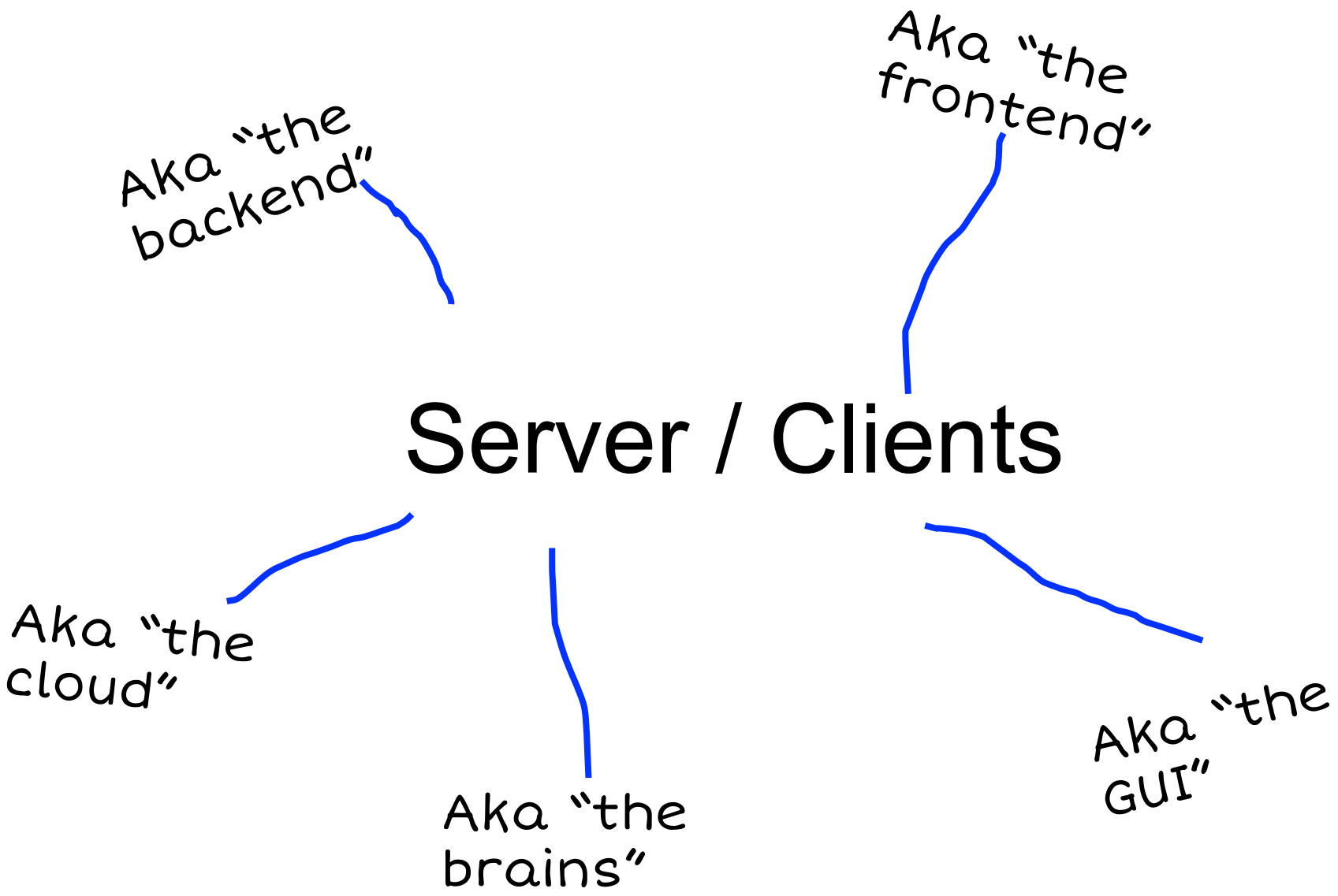
The Internet



Most of the Internet



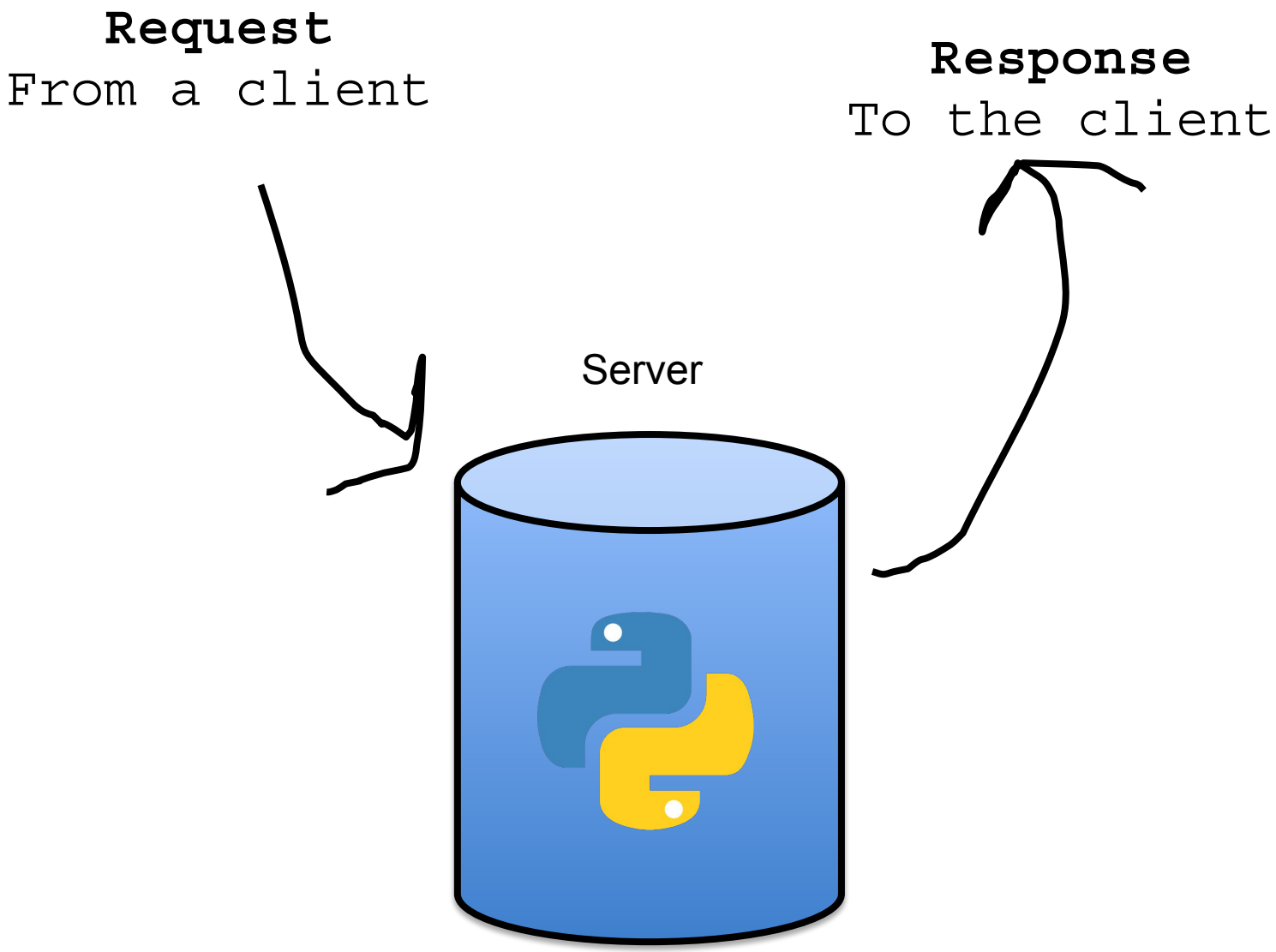
Most of the Internet



Today, the server

A server's main job is to respond to requests

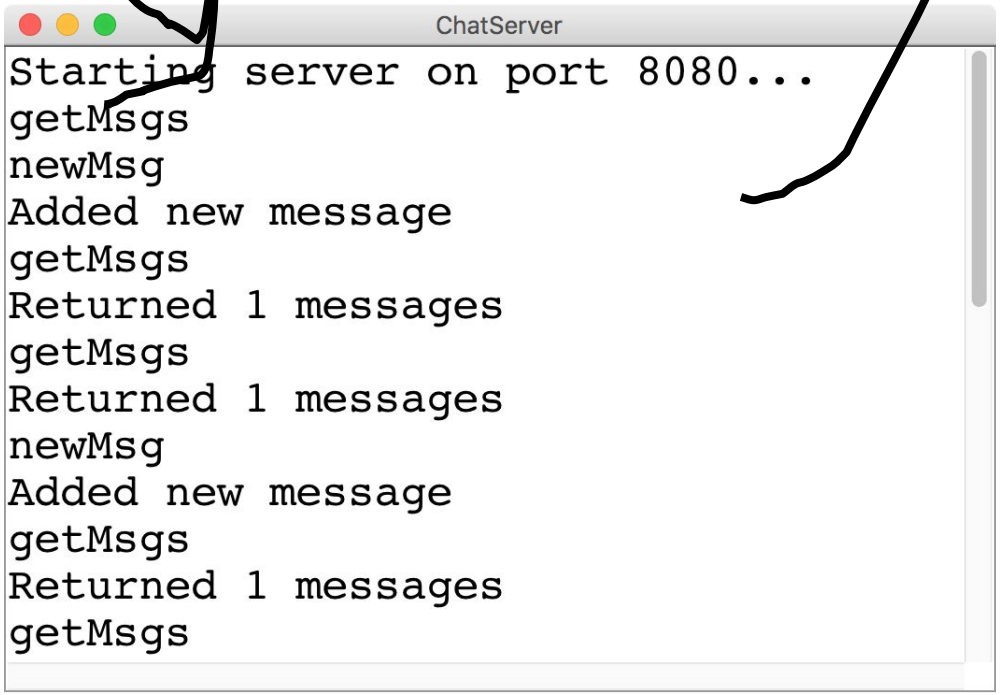
A Server's Simple Purpose



A Server's Simple Purpose

Request
`someRequest`

String
`serverResponse`



Servers on one slide

1

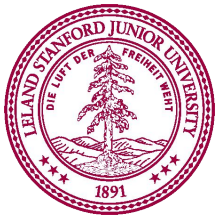
```
# handle server requests (must be in a class)
def handle_request(self, request):
    # return a string response!
```

2

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

3

```
# enjoy
```

Servers on one slide

1

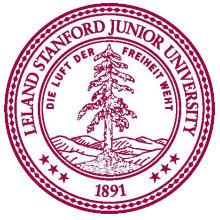
```
# handle server requests (must be in a class)
def handle_request(self, request):
    # return a string response!
```

2

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

3

```
# enjoy
```

Servers on one slide

1

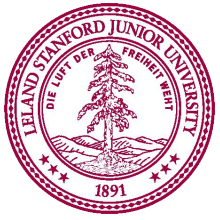
```
# handle server requests (must be in a class)
def handle_request(self, request):
    # return a string response!
```

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

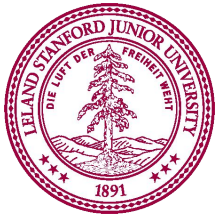
```
# handle server requests (must be in a class)
def handle_request(self, request):
    # return a string response!
```

2

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

3

```
# enjoy
```

Servers on one slide

1

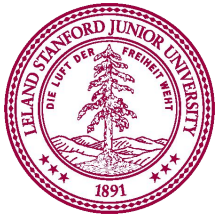
```
# handle server requests (must be in a class)
def handle_request(self, request):
    # return a string response!
```

2

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

3

```
# enjoy
```

Servers on one slide

1

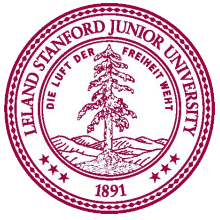
```
# handle server requests (must be in a class)
def handle_request(self, request):
    # return a string response!
```

2

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

3

```
# enjoy
```

Servers on one slide

1

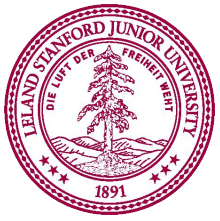
```
# handle server requests (must be in a class)
def handle_request(self, request):
    # return a string response!
```

2

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

3

```
# enjoy
```

Servers on one slide

1

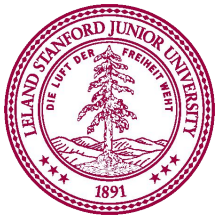
```
# handle server requests (must be in a class)
def handle_request(self, request):
    # return a string response!
```

2

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

3

```
# enjoy
```

Servers on one slide

1

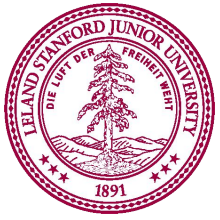
```
# handle server requests (must be in a class)
def handle_request(self, request):
    # return a string response!
```

2

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

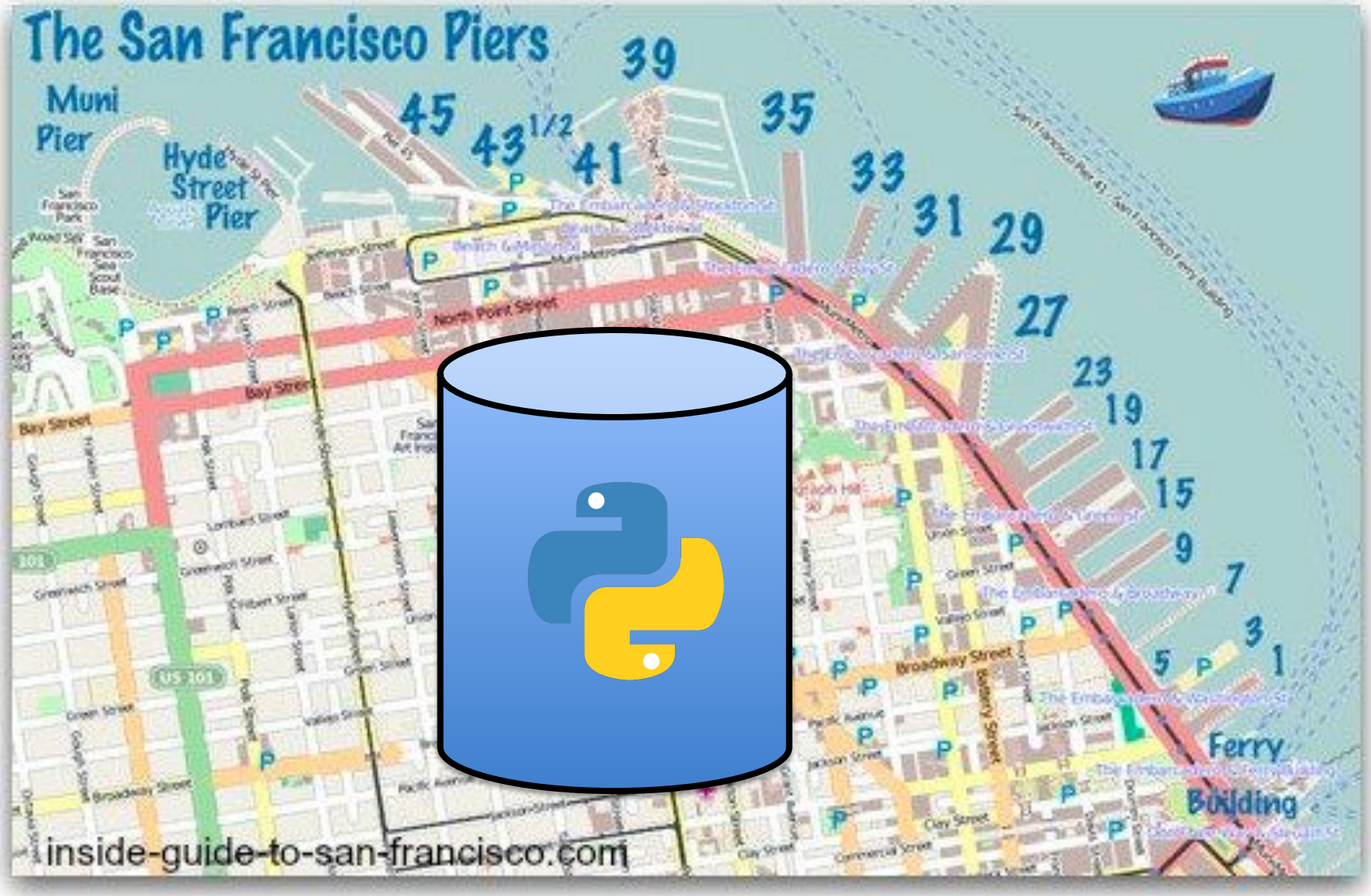
3

```
# 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

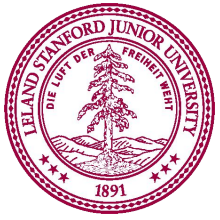
```
# handle server requests (must be in a class)
def handle_request(self, request):
    # return a string response!
```

2

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

3

```
# enjoy
```

What is a Request?

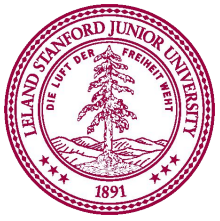
```
/* Request has a command */
command (type is string)

/* 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 requ
    An internet request has both a command and a dictionary of parameter
    This class defines a special function __str__ which means if you hav
    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 re
        return 'command=\'' + self.command + '\' params=' + str(self.par
```

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?

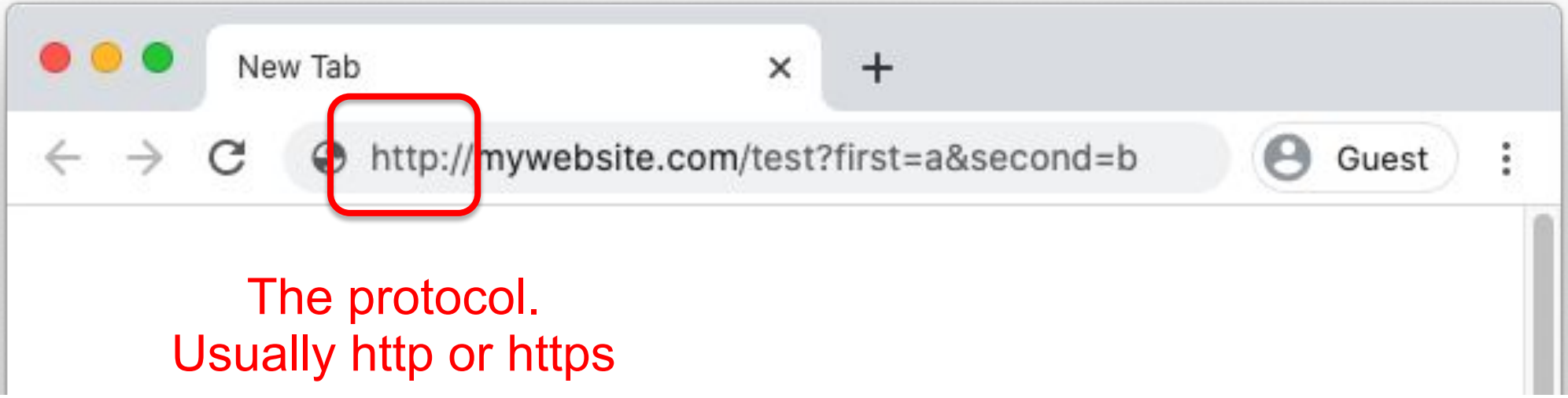
Other programs can send requests!

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

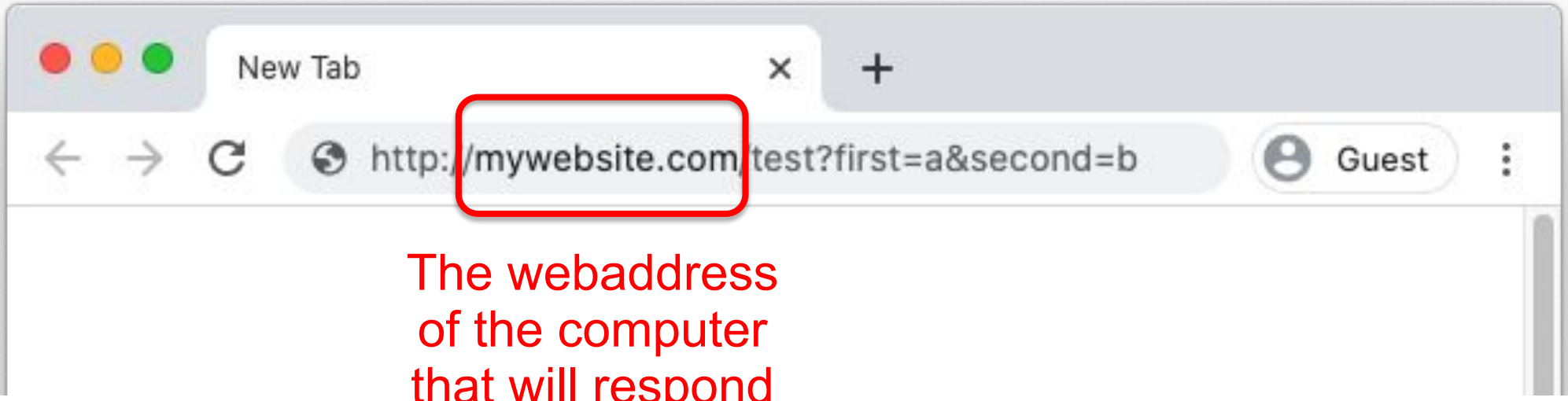
Web browsers can send requests!

Anatomy of a Browser Request

Anatomy of a Browser Request

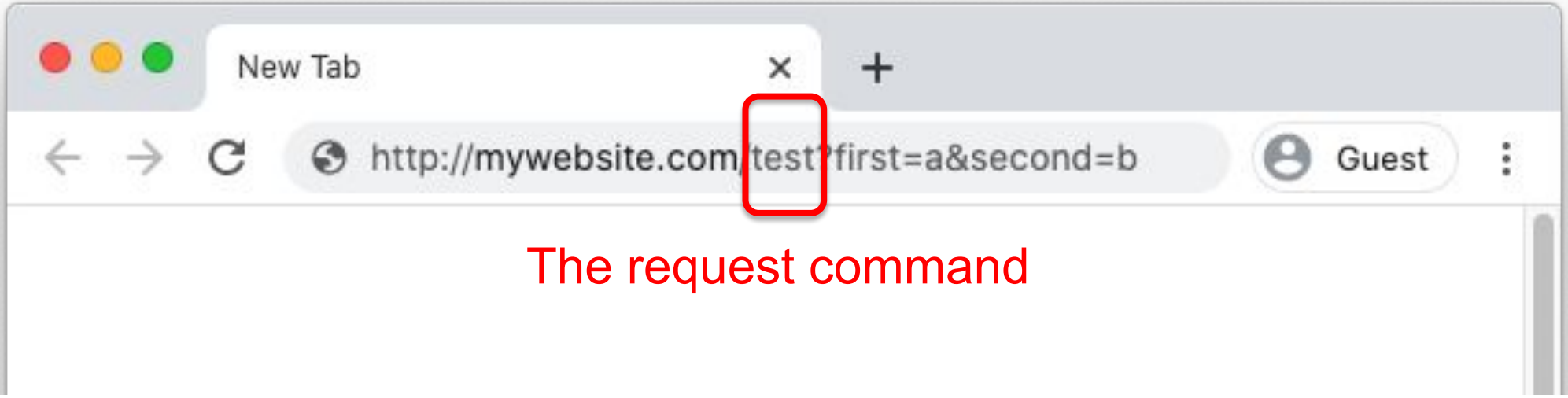


Anatomy of a Browser Request



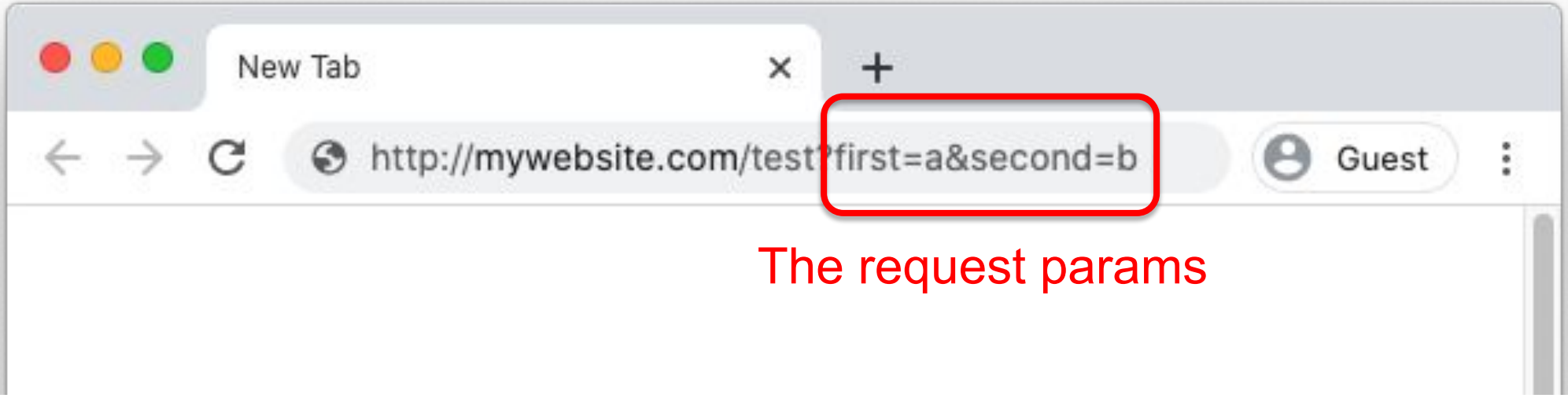
The webaddress of the computer that will respond to the request

Anatomy of a Browser Request



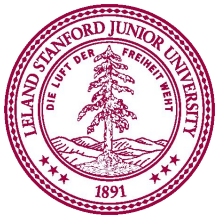
The request command

Anatomy of a Browser Request

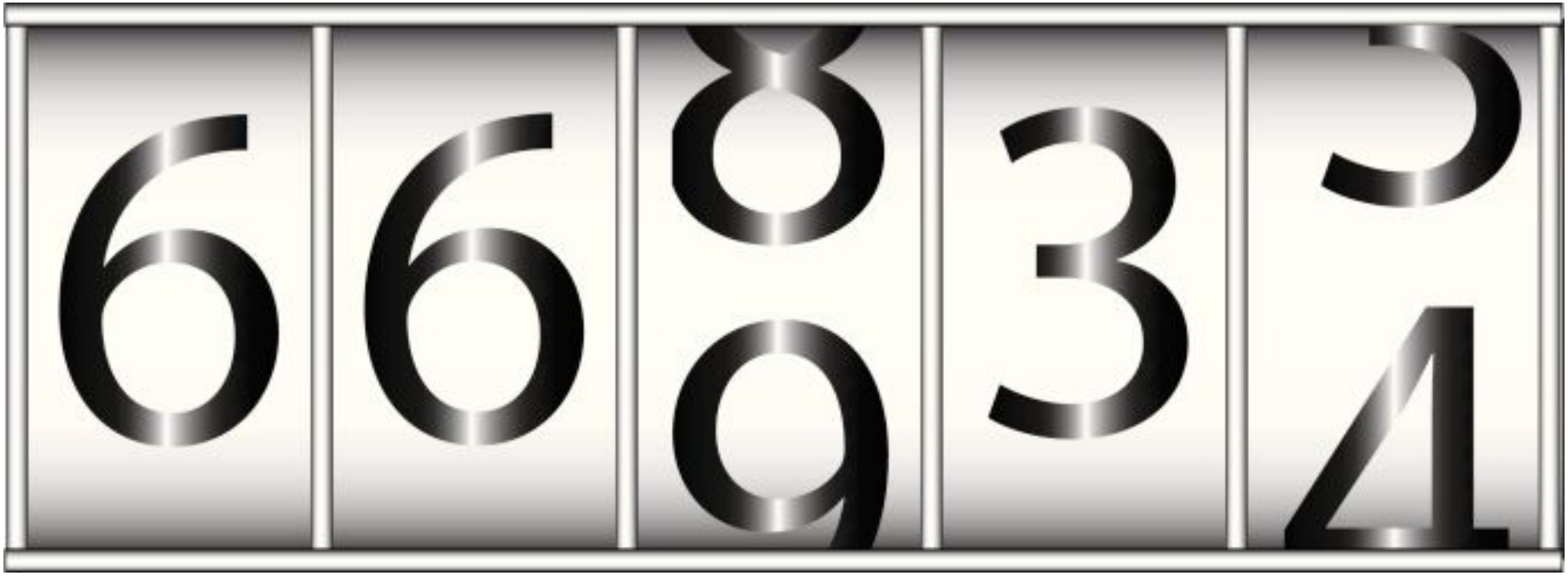


First Server Example!

```
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



Recall Requests

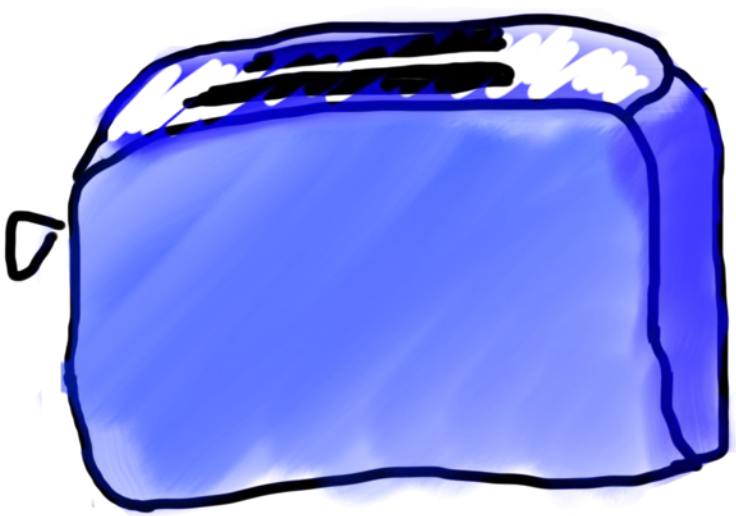
```
/* Request has a command */
command (string)

/* Request has parameters */
params (dict)
```

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

Requests are like Remote Method Calls

Server

Server has a bunch of discrete things it can do



make_toast

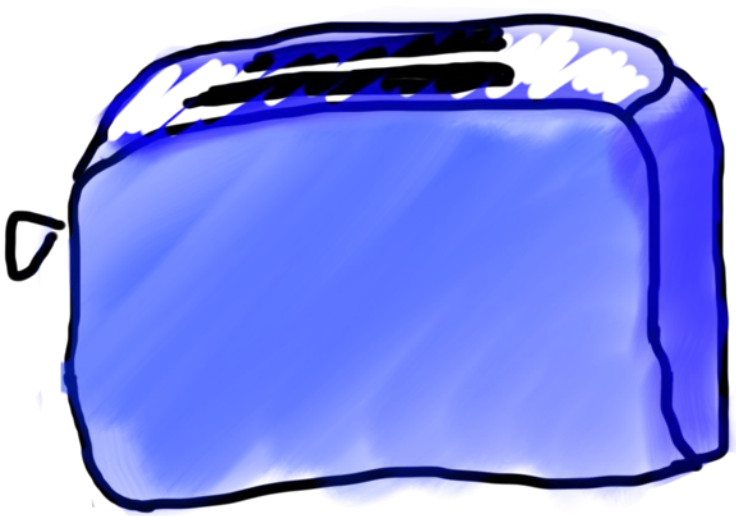
blend

ty

Requests are like Remote Method Calls

Server

Server has a bunch of discrete things it can do

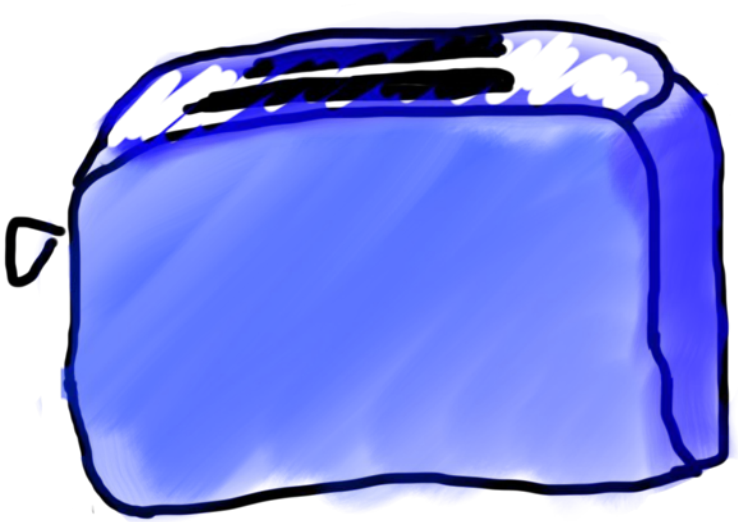


get_status

add_user

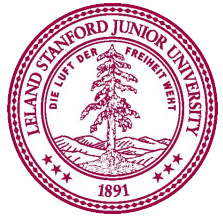
Requests are like Remote Method Calls

Server

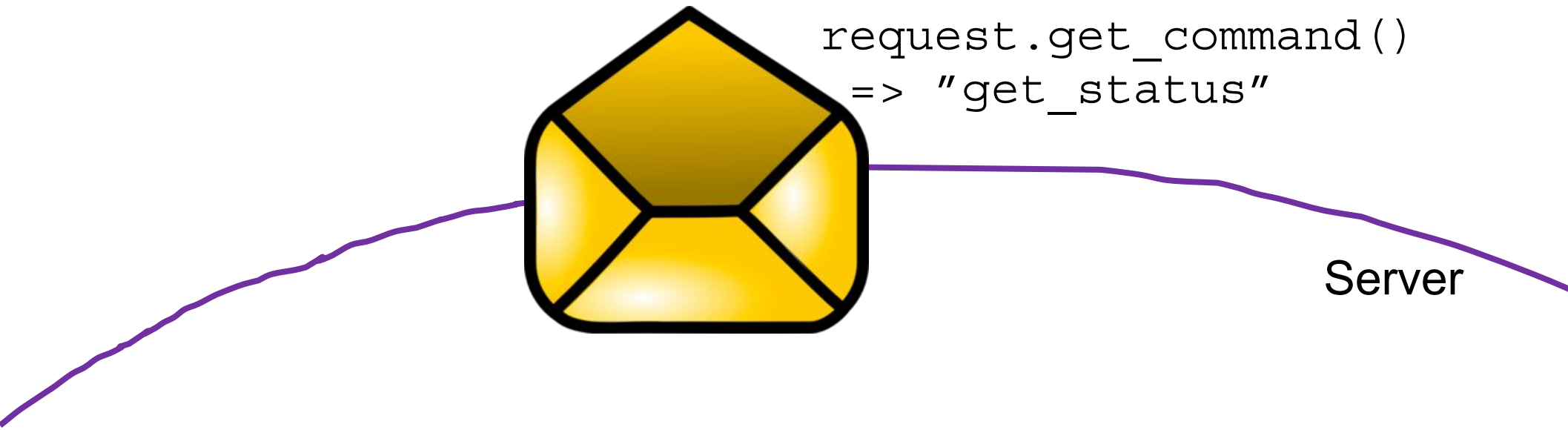


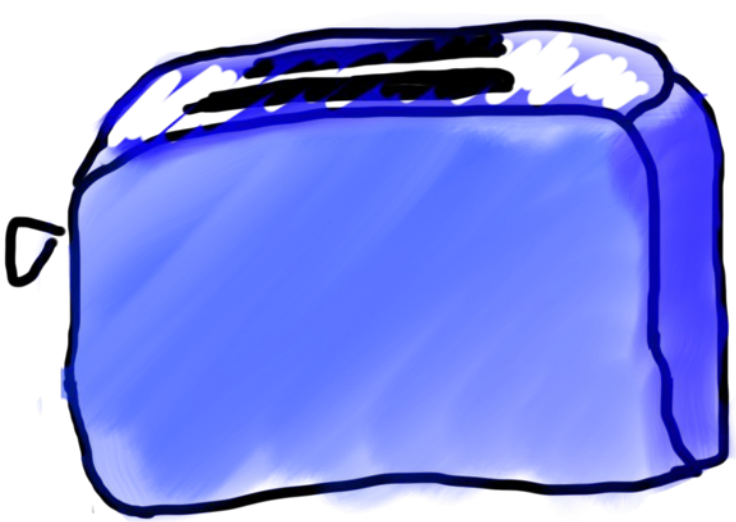
get_status

add_user



Requests are like Remote Method Calls

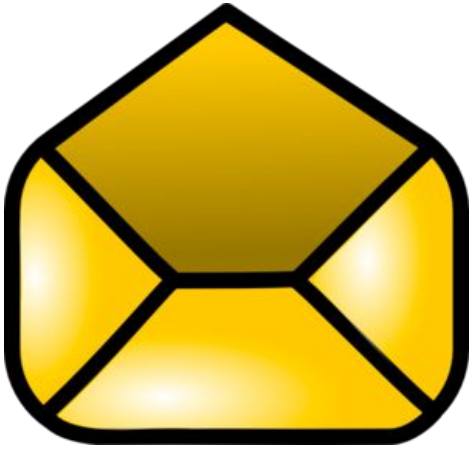




get_status

add_user

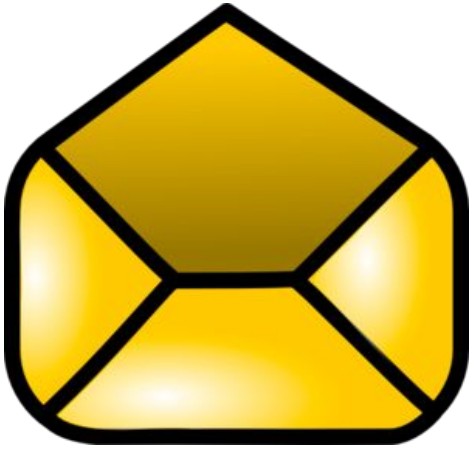
Requests are like Remote Method Calls



get_status

ty

Requests are like Remote Method Calls

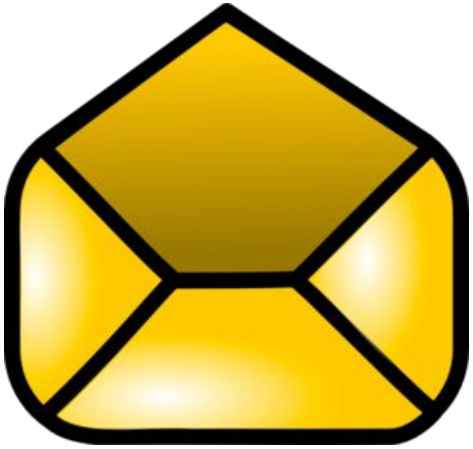


I was given a parameter!

get_status

ty

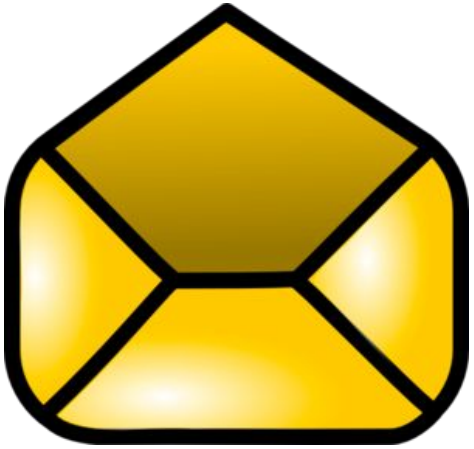
Requests are like Remote Method Calls



`request.params["userName"]`

`get_status`

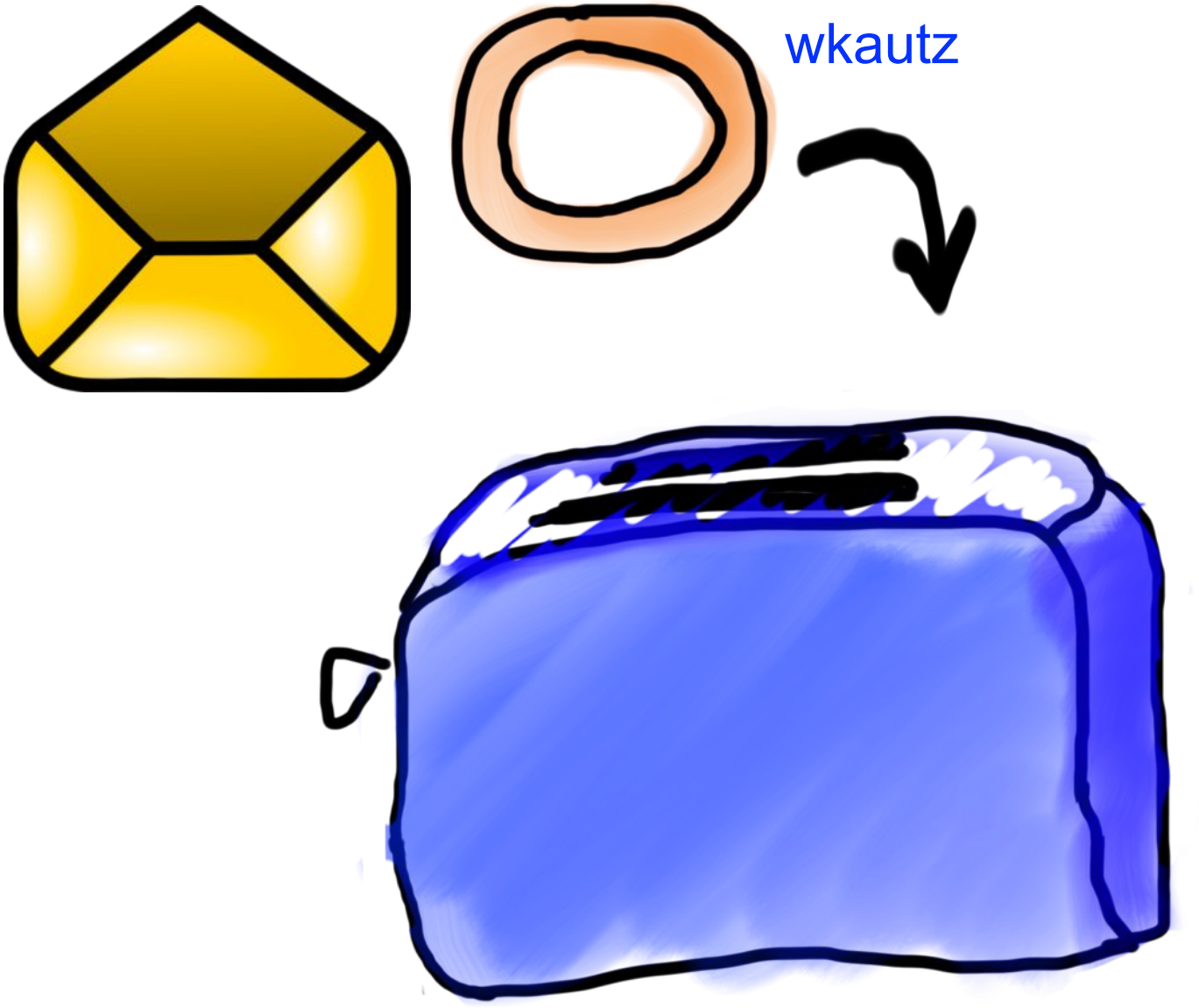
Requests are like Remote Method Calls



get_status

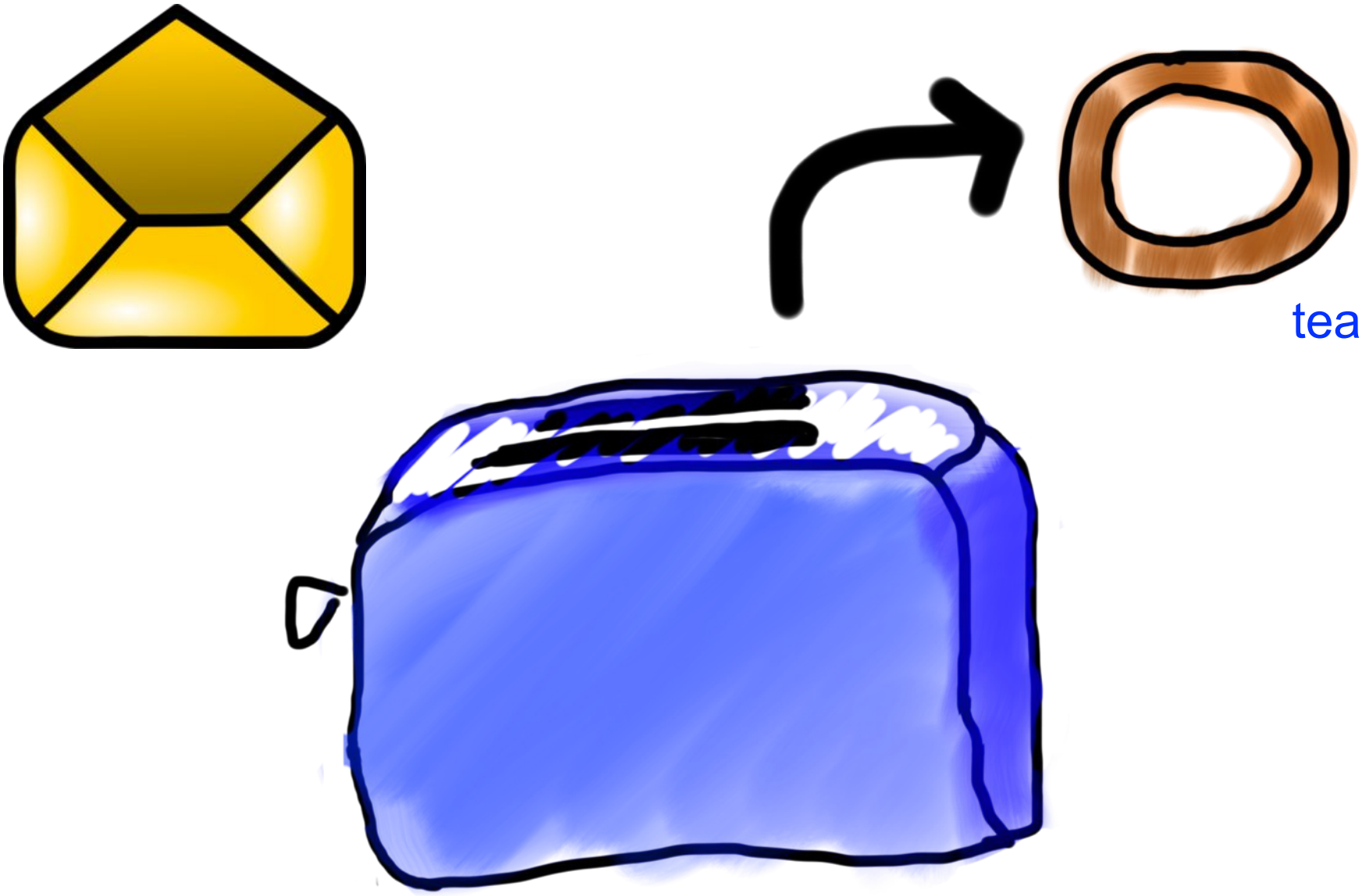
ty

Requests are like Remote Method Calls

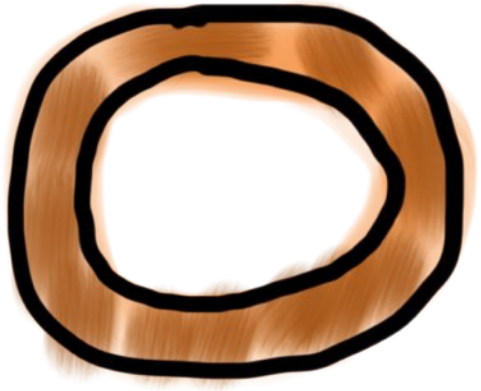


ty

Requests are like Remote Method Calls

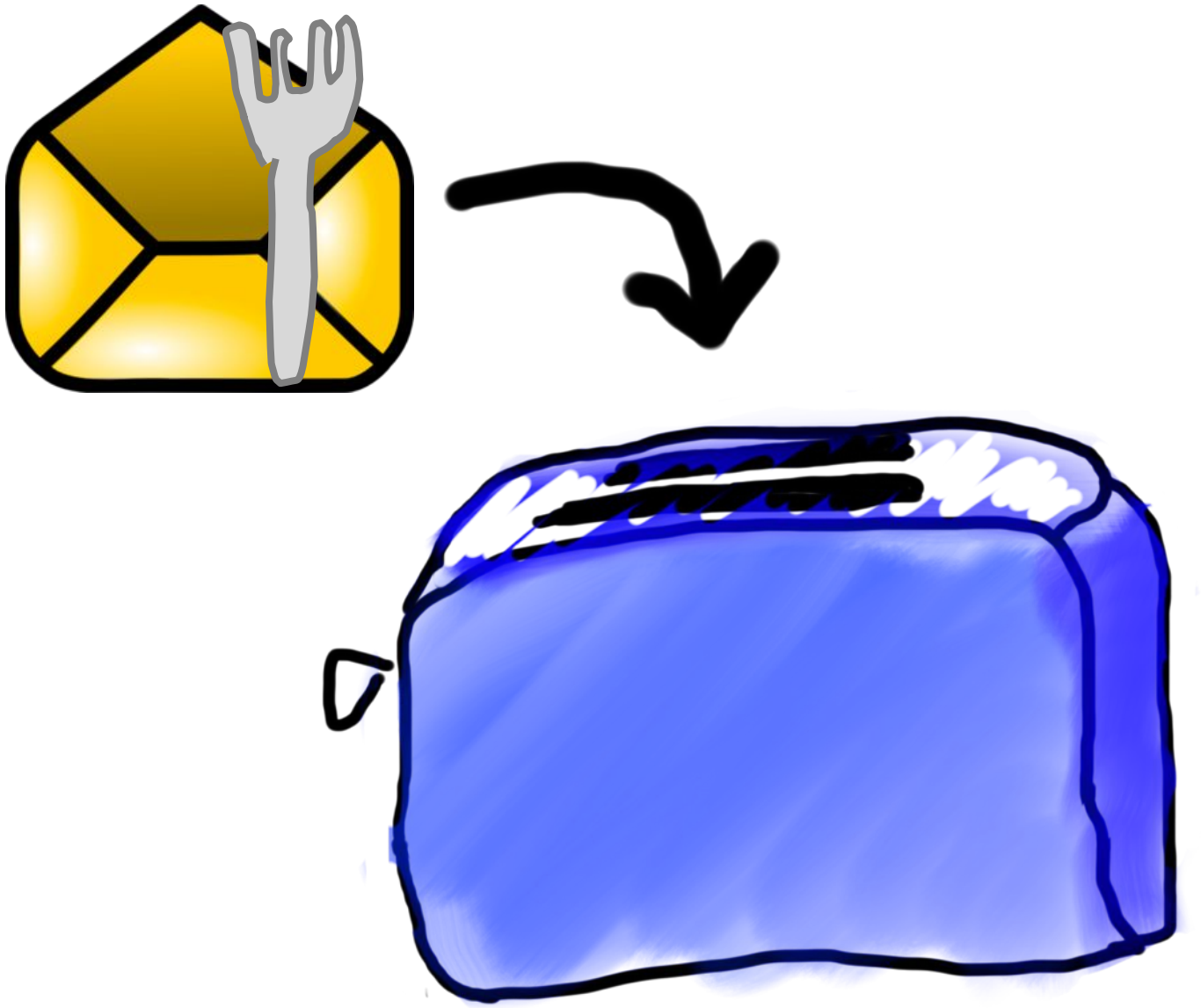


```
def handle_request(self, request):
    cmd = request.command
    if cmd == 'get_status':
        user = request.params['userName']
        status = self.get_status(user)
        return status
```



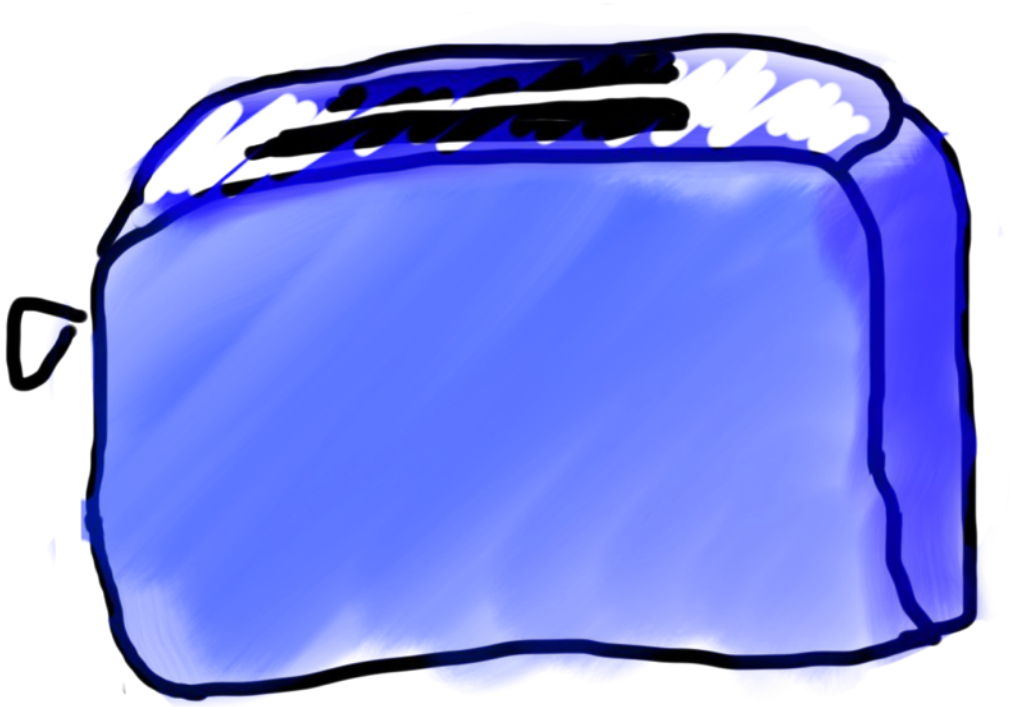
Must be a string!

Requests are like Remote Method Calls

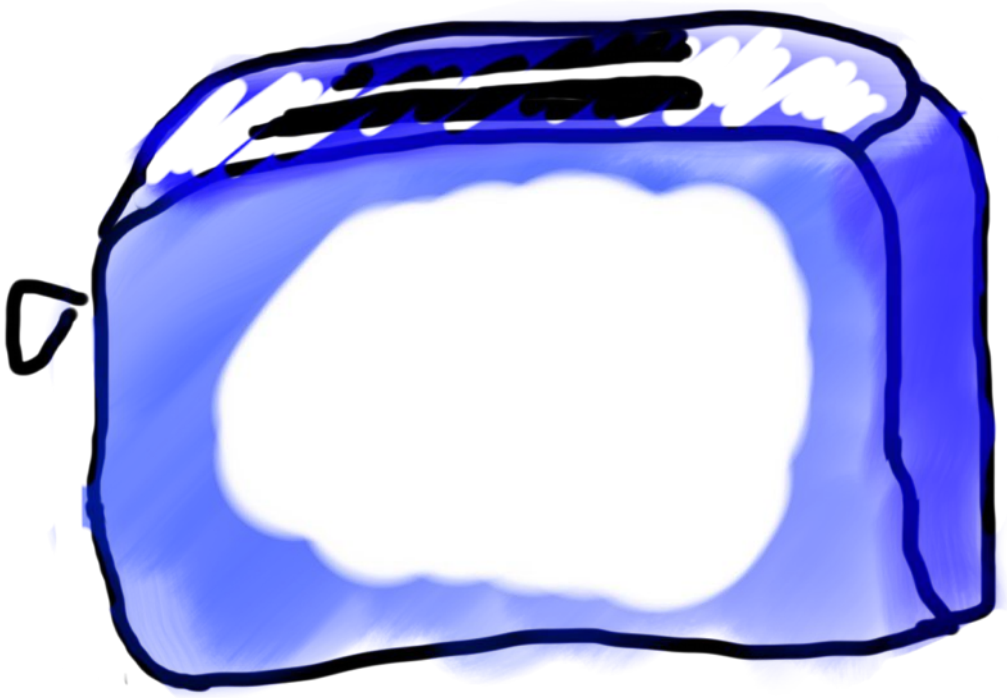


Requests are like Remote Method Calls

Requests are like Remote Method Calls

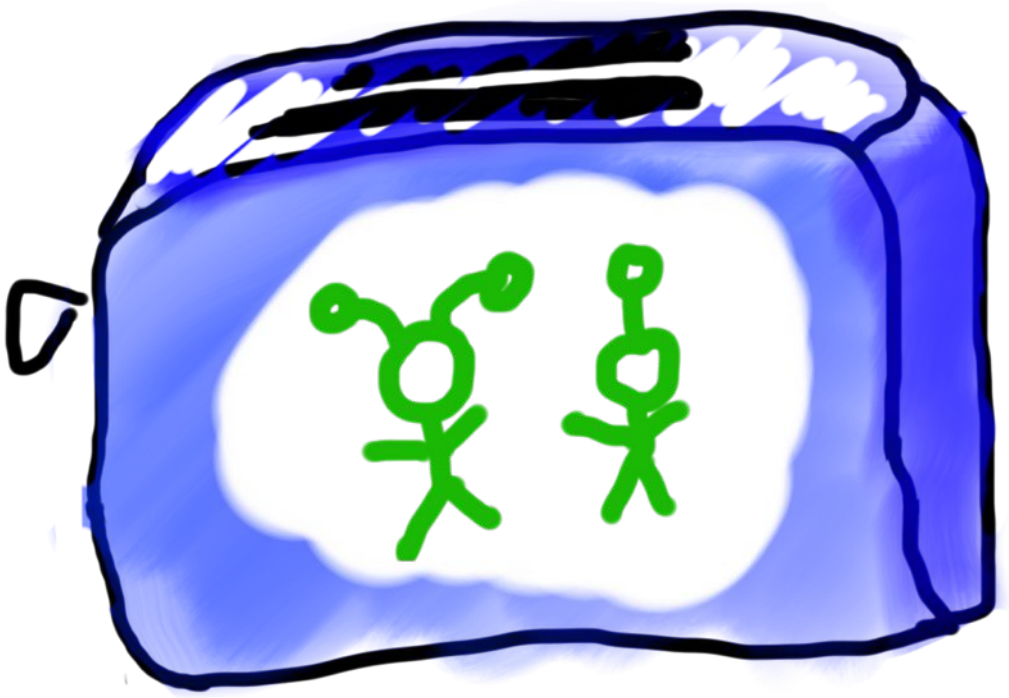


Requests are like Remote Method Calls



Requests are like Remote Method Calls

Requests are like Remote Method Calls



Requests responses are strings, often encoded using JSON

Recall JSON

ages.json

```
{
        "Chris":48,
        "Gary":70,
        "Mehran":50,
        "Wil":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":48,
        "Gary":70,
        "Mehran":50,
        "Wil":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

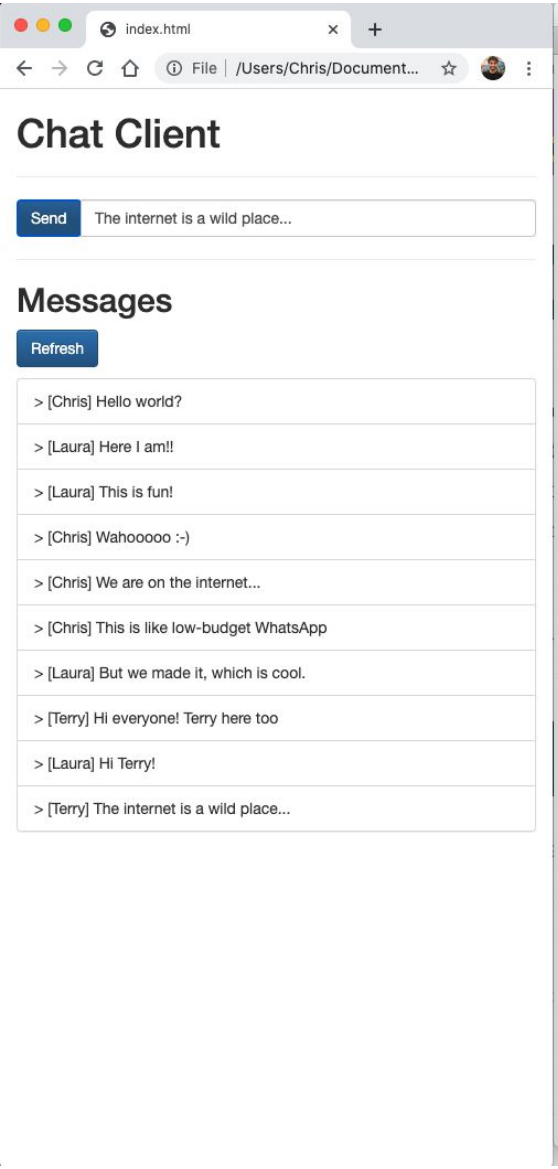
ages.json

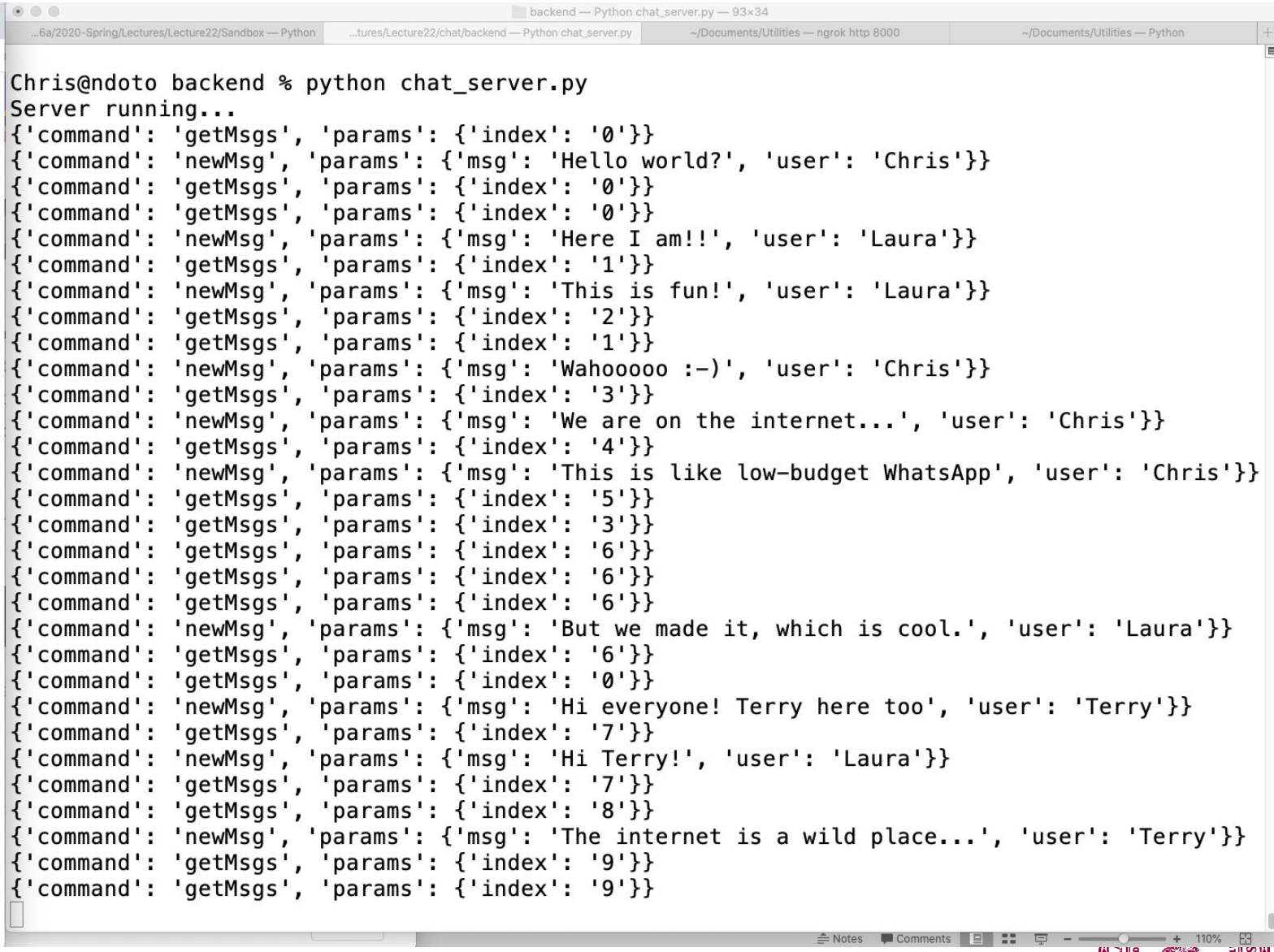
```
{
        "Chris":48,
        "Gary":70,
        "Mehran":50,
        "Wil":23,
        "Rihanna":32,
        "Adele":32
}
```

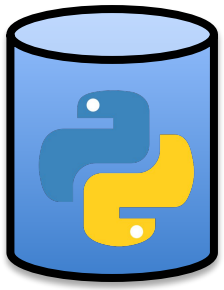
```
import json

# 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

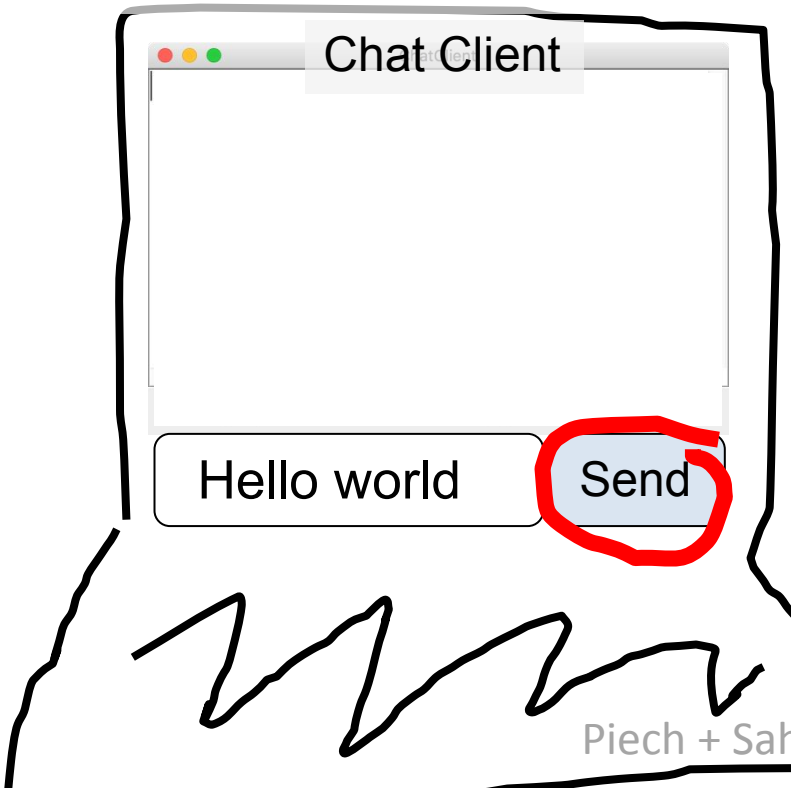


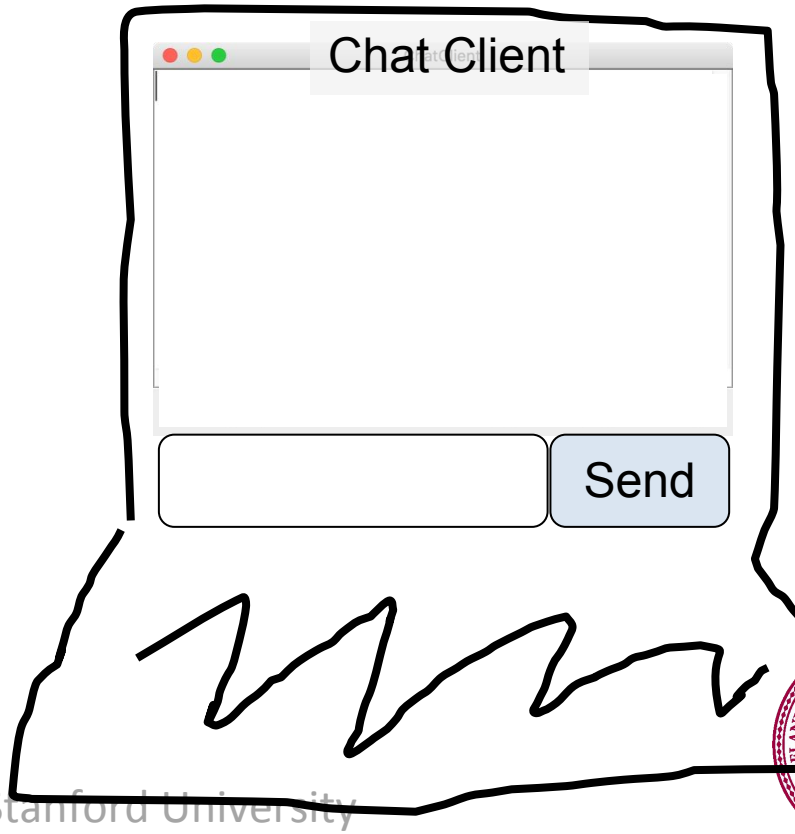


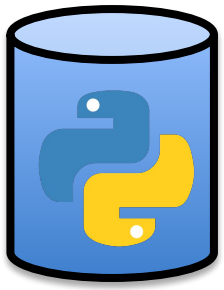


```
history = [
]
```

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

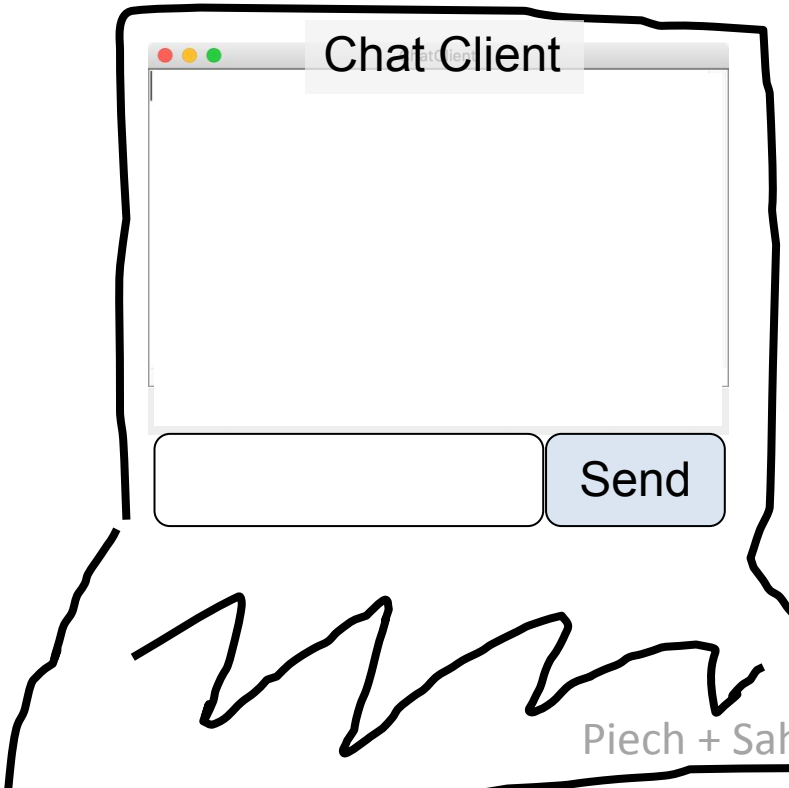


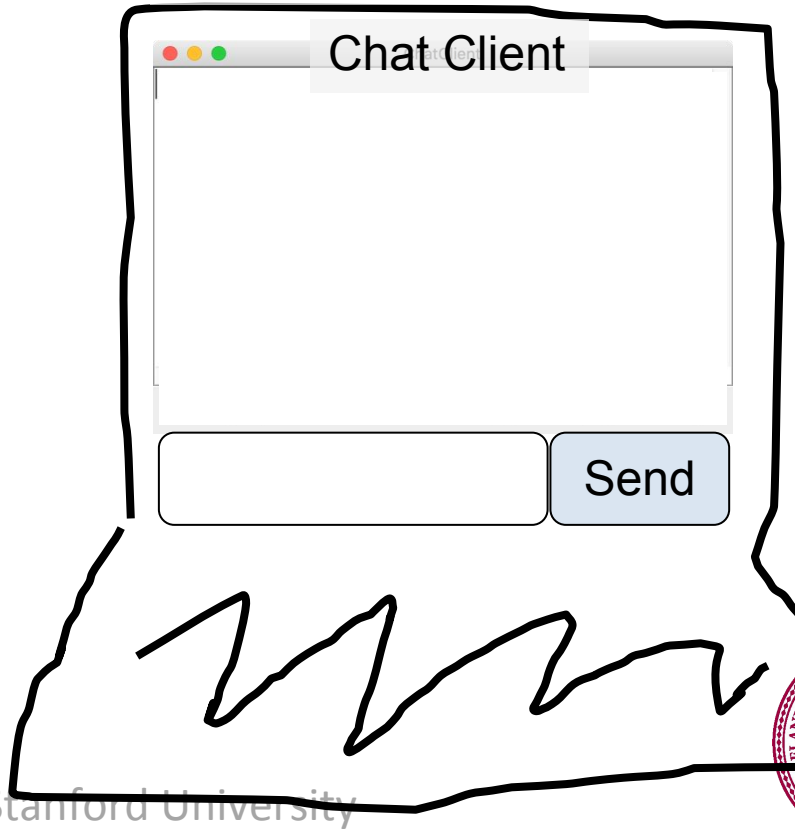


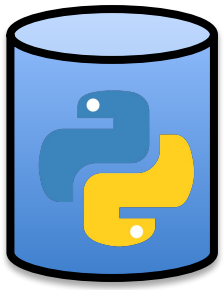


```
history = [
    '[C] Hello world'
]
```

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

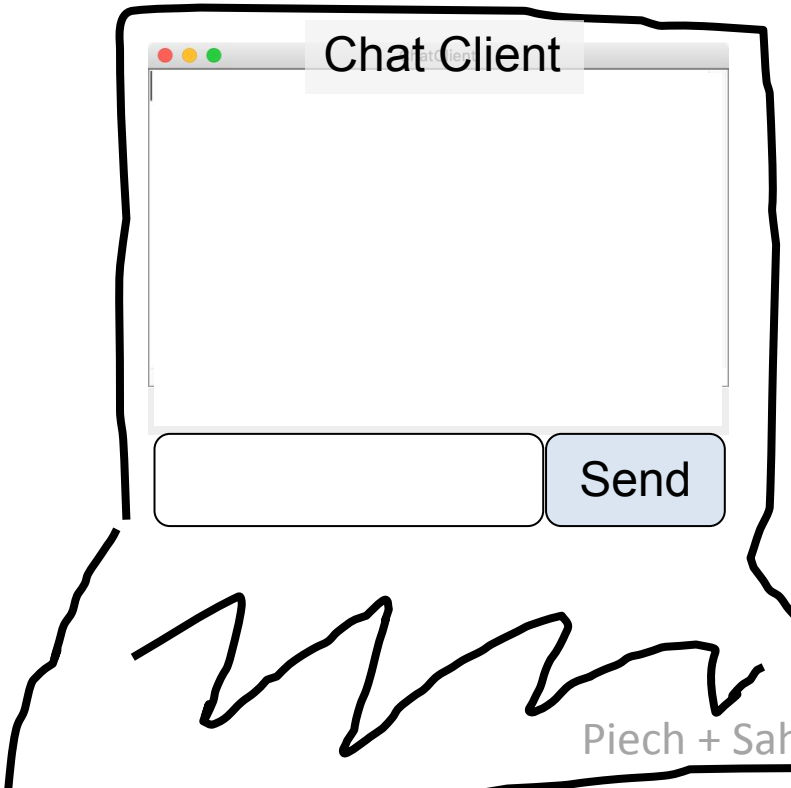


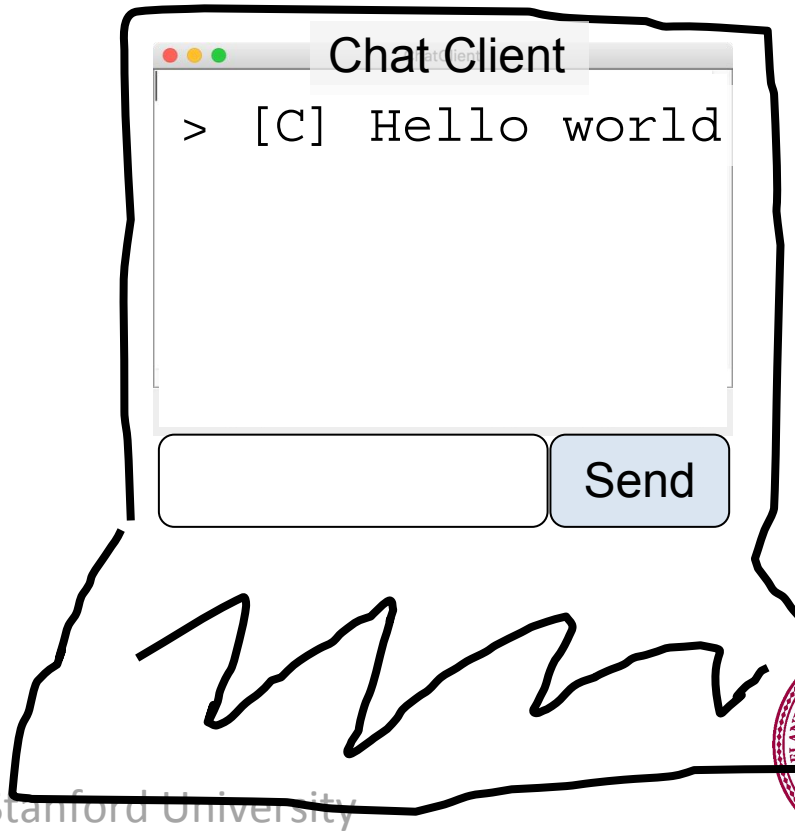




```
history = [
    '[C] Hello world'
]
```

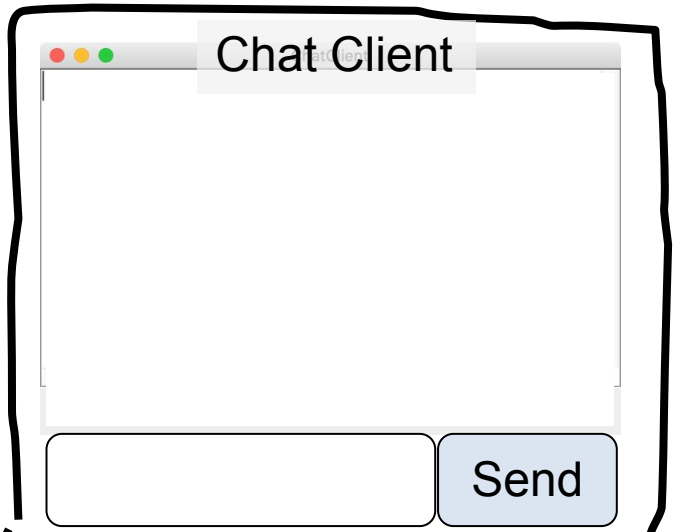
```
'["[C] Hello world"]'
```





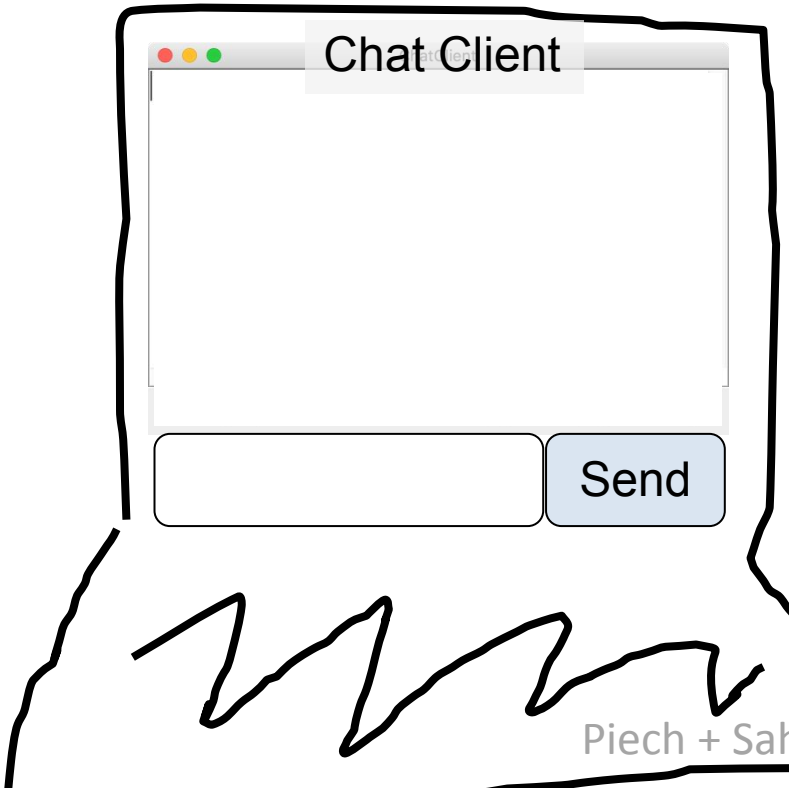
```
history = [
    '[C] Hello world'
]
```

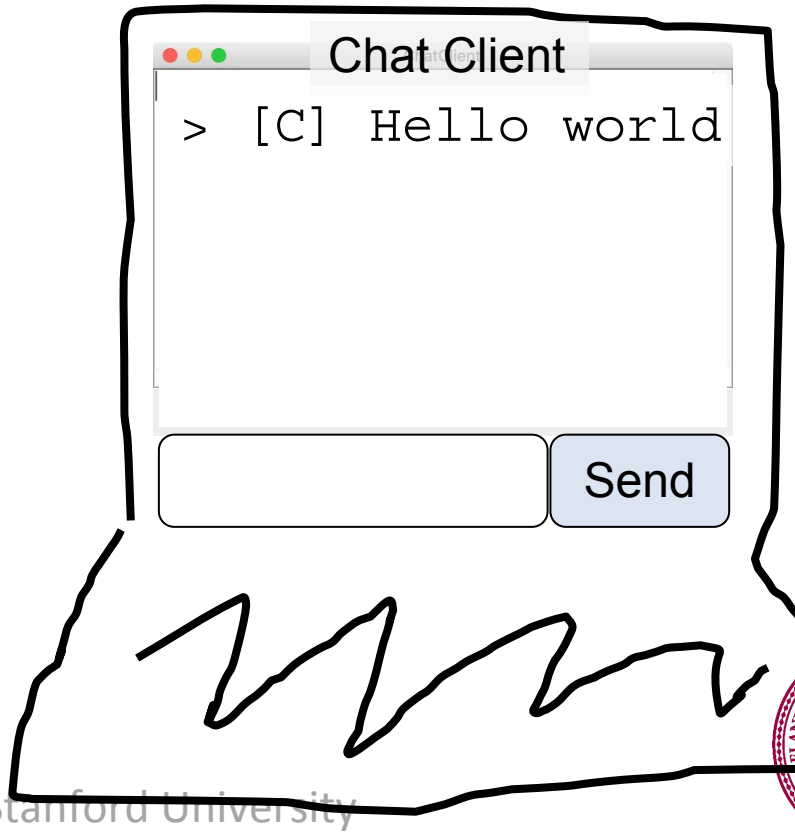
```
addMsg
{
    'msg' : 'Im here too'
    'user' : 'B'
}
```

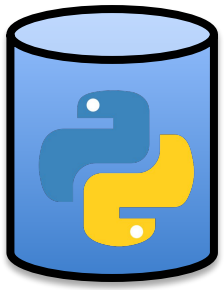


```
history = [
    '[C] Hello world',
    '[B] Im here too'
]
```

```
'Got it'
```

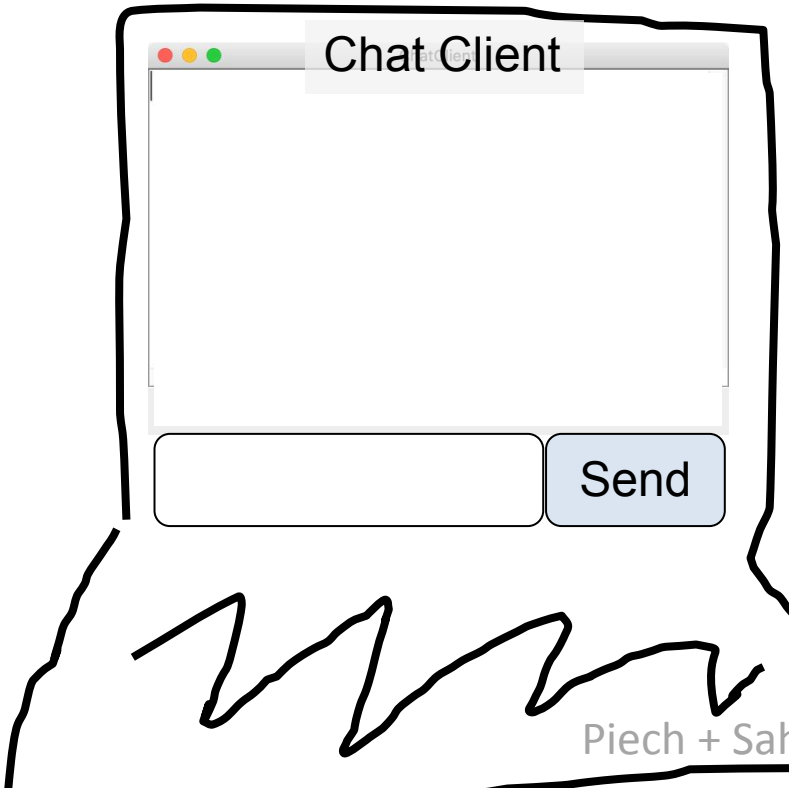


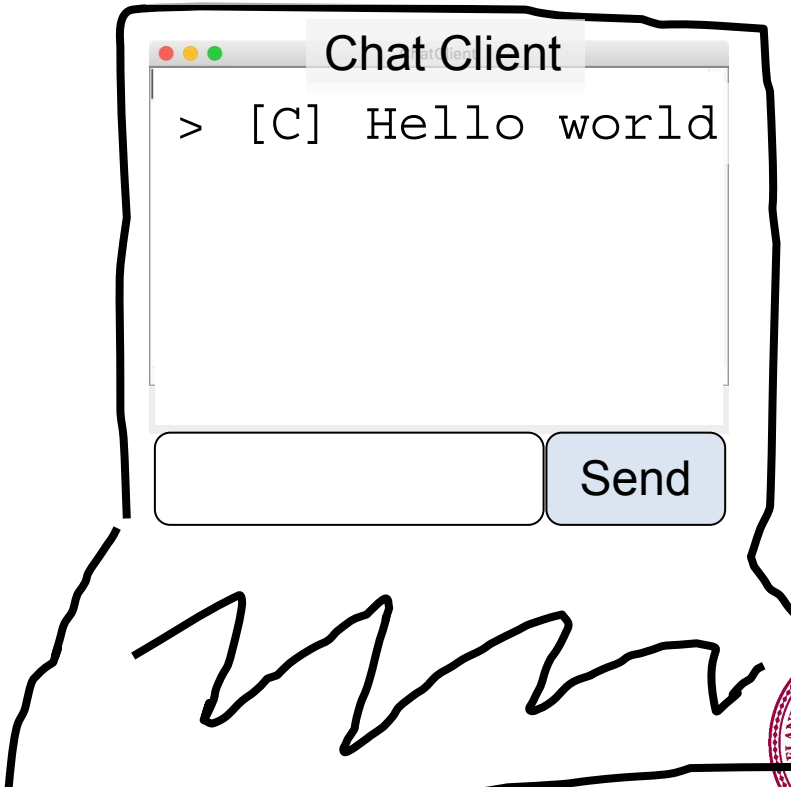




```
history = [
    '[C] Hello world',
    '[B] Im here too'
]
```

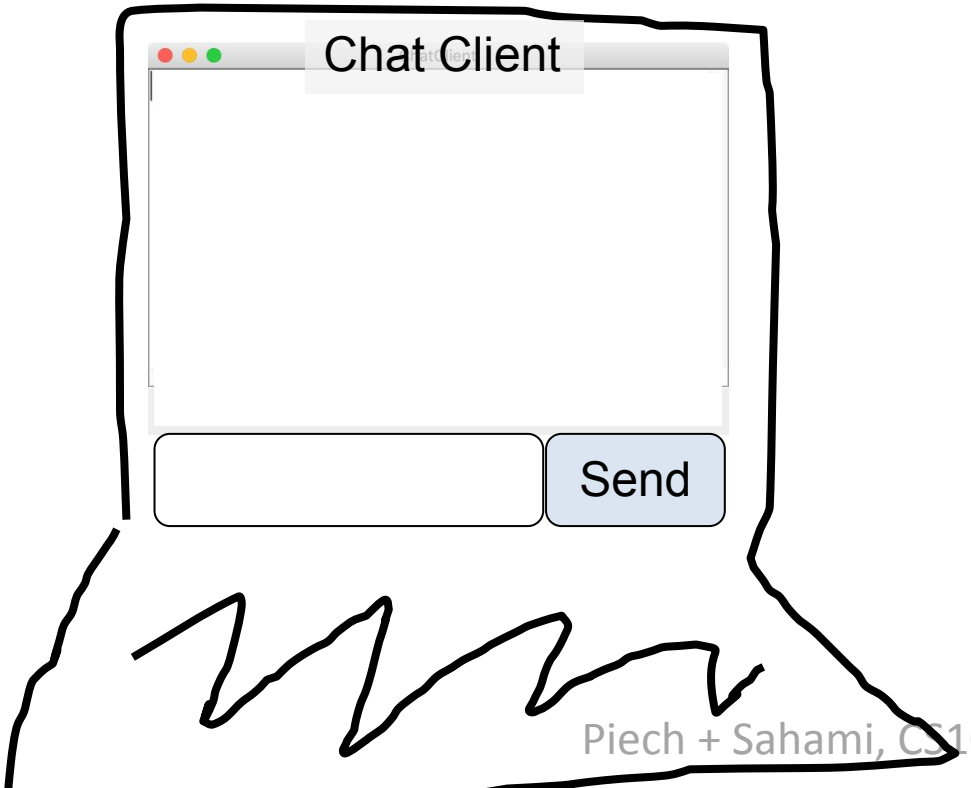
```
getMsgs
{
    'index' : 1
}
```

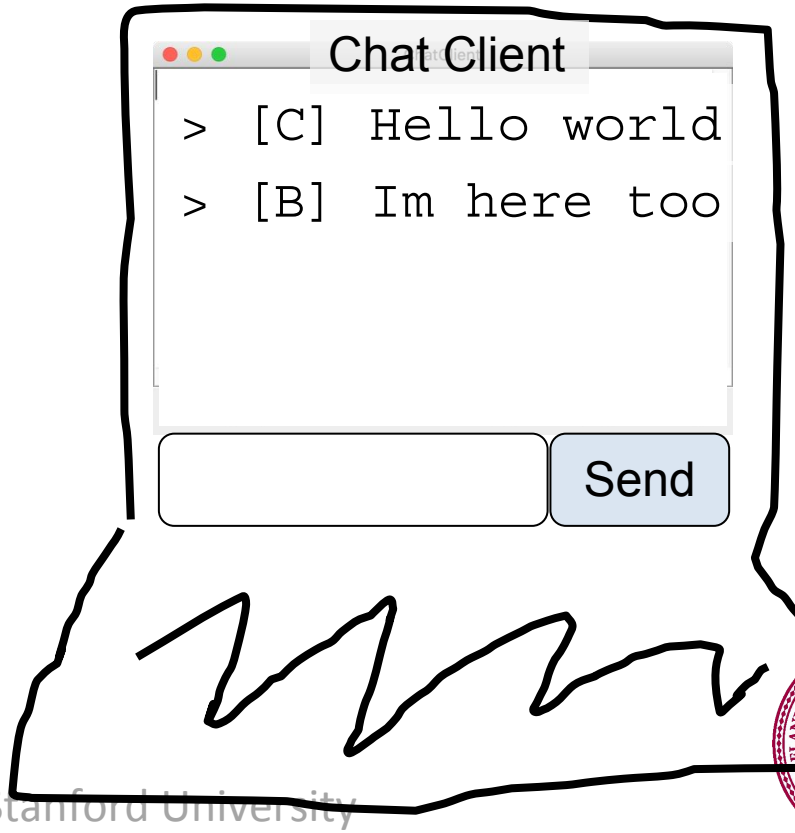


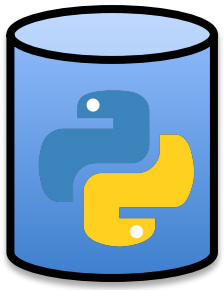


```
history = [
    '[C] Hello world',
    '[B] Im here too'
]
```

```
'["[B] Im here too"]'
```

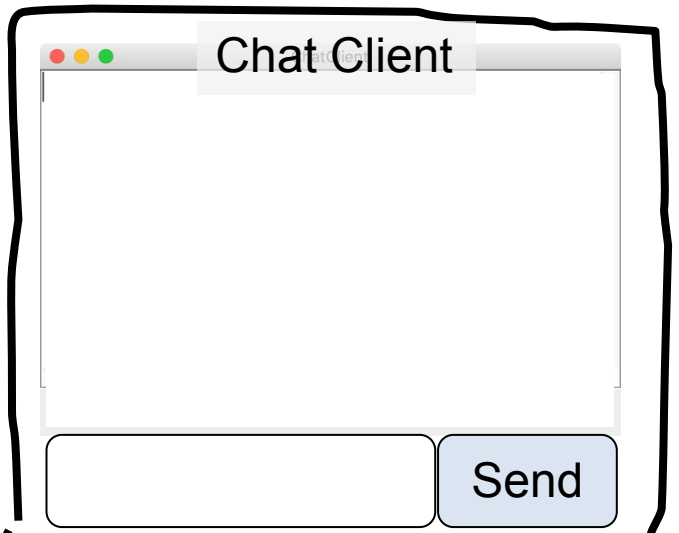


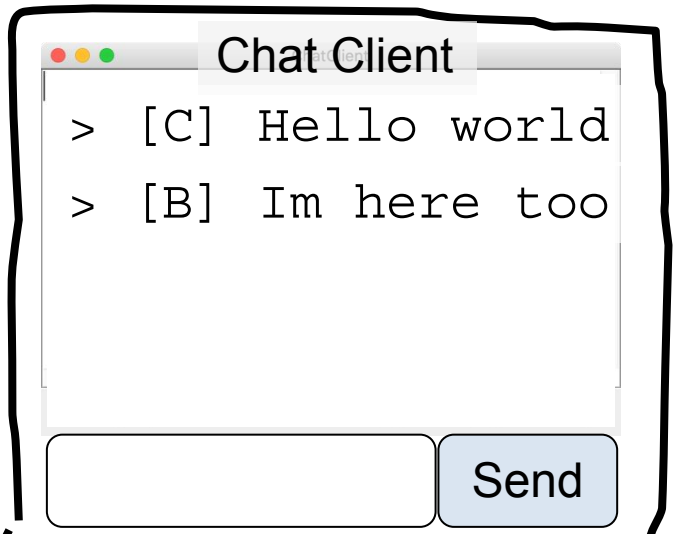


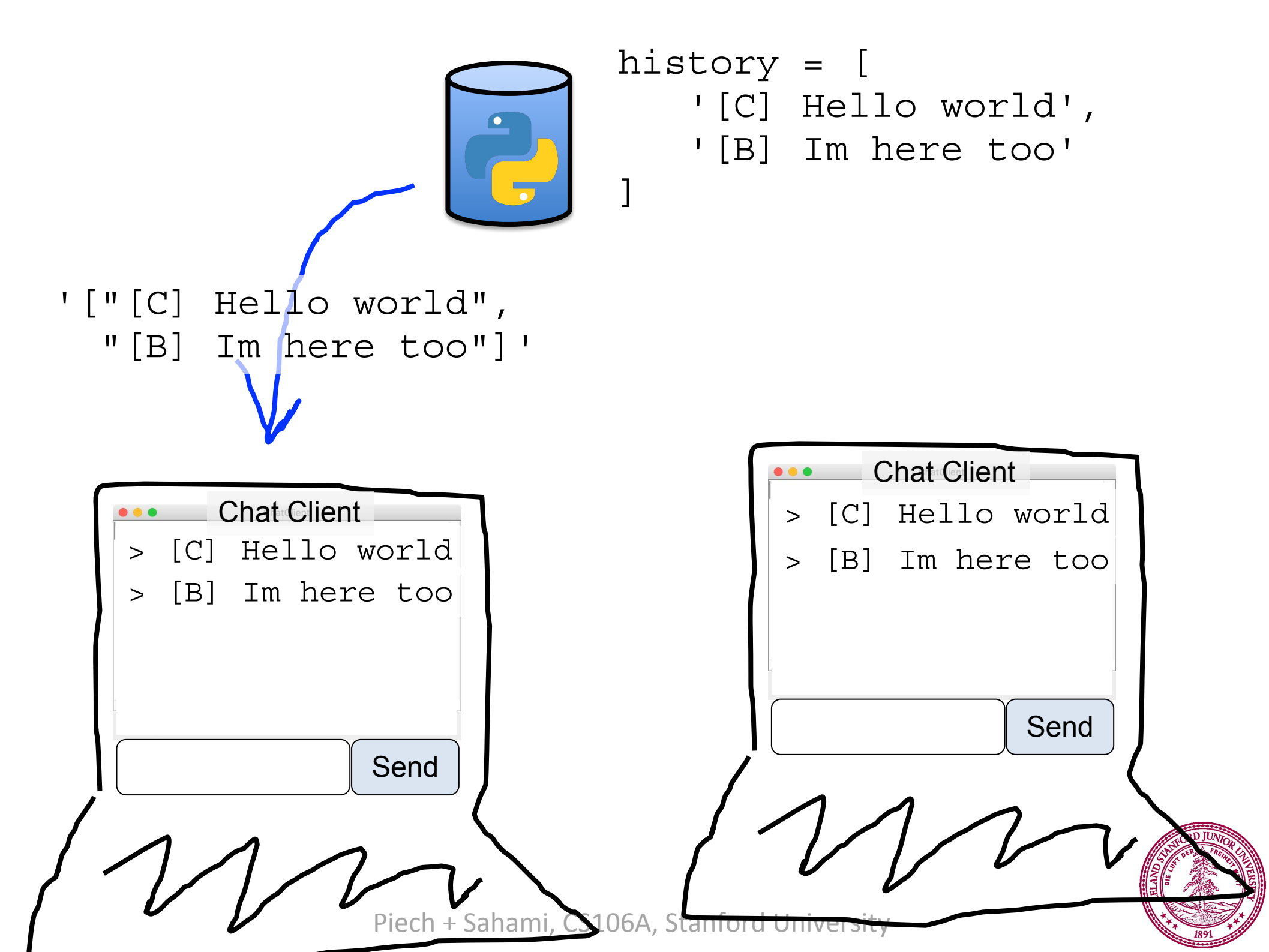


```
history = [
    '[C] Hello world',
    '[B] Im here too'
]
```

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



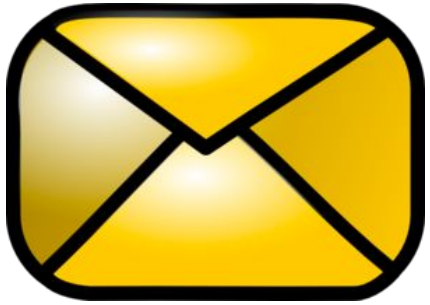



history = [
'[C] Hello world',
'[B] Im here too'
]
'["[C] Hello world",
"[B] Im here too"]'
Chat Client
> [C] Hello world
> [B] Im here too
Send
Chat Client
> [C] Hello world
> [B] Im here too
Send

Chat Server

Chat Server

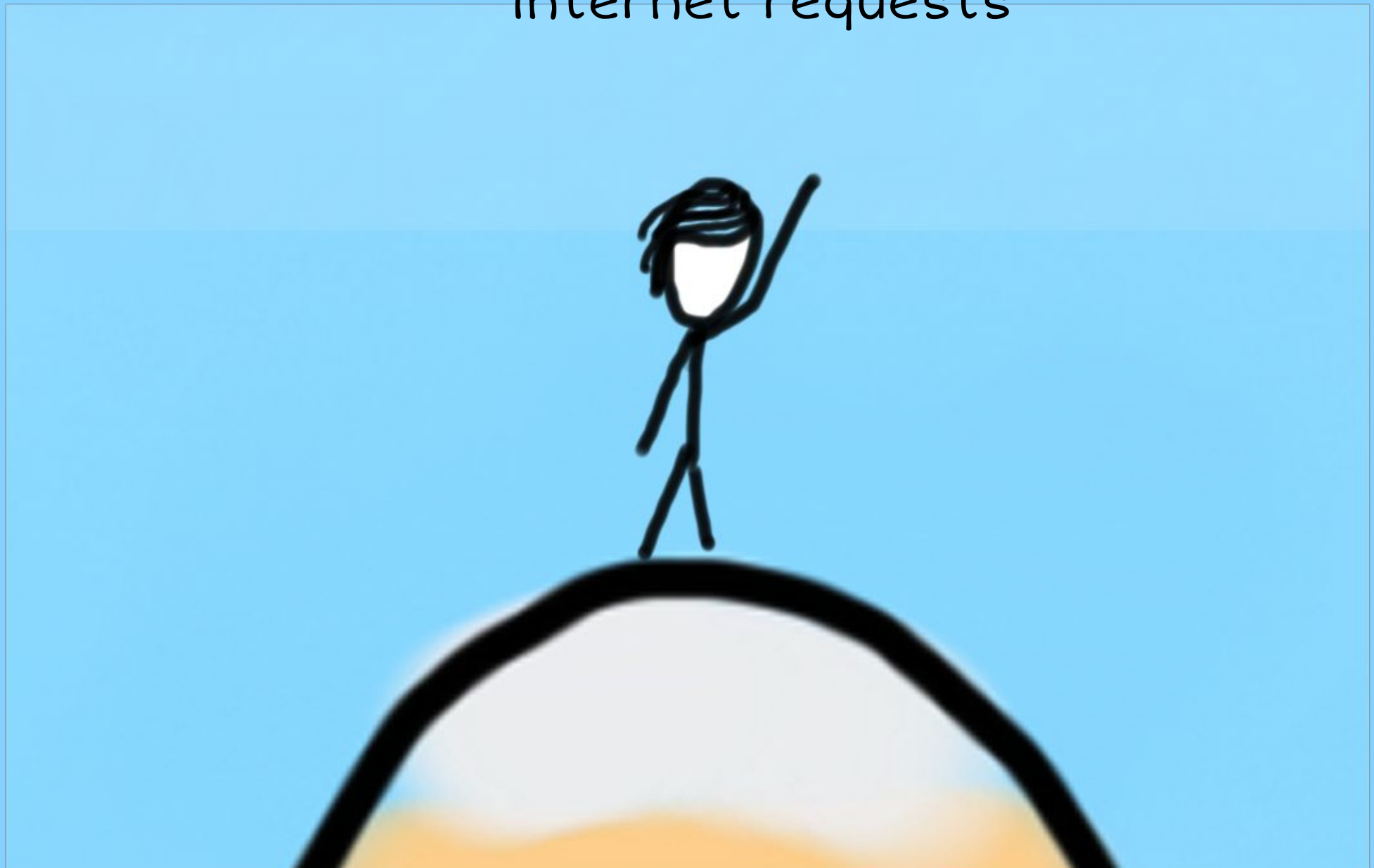
addMsg
msg = *text*
user = *user*



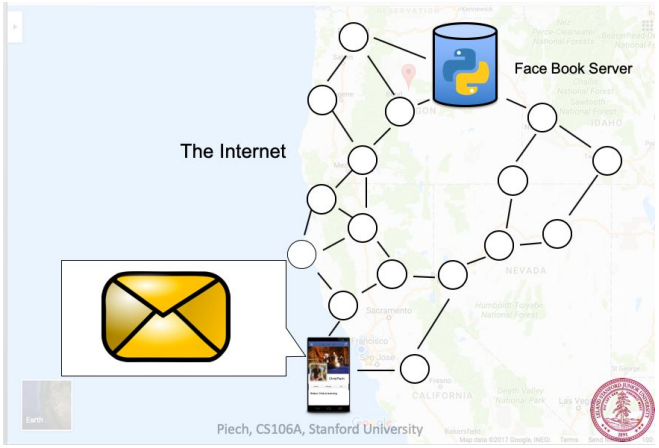
getMsgs
index = start_*index*

Learning Goals

1. Write a program that can respond to internet requests



Things we saw along the way



The request command

data_str = json.dumps(data)