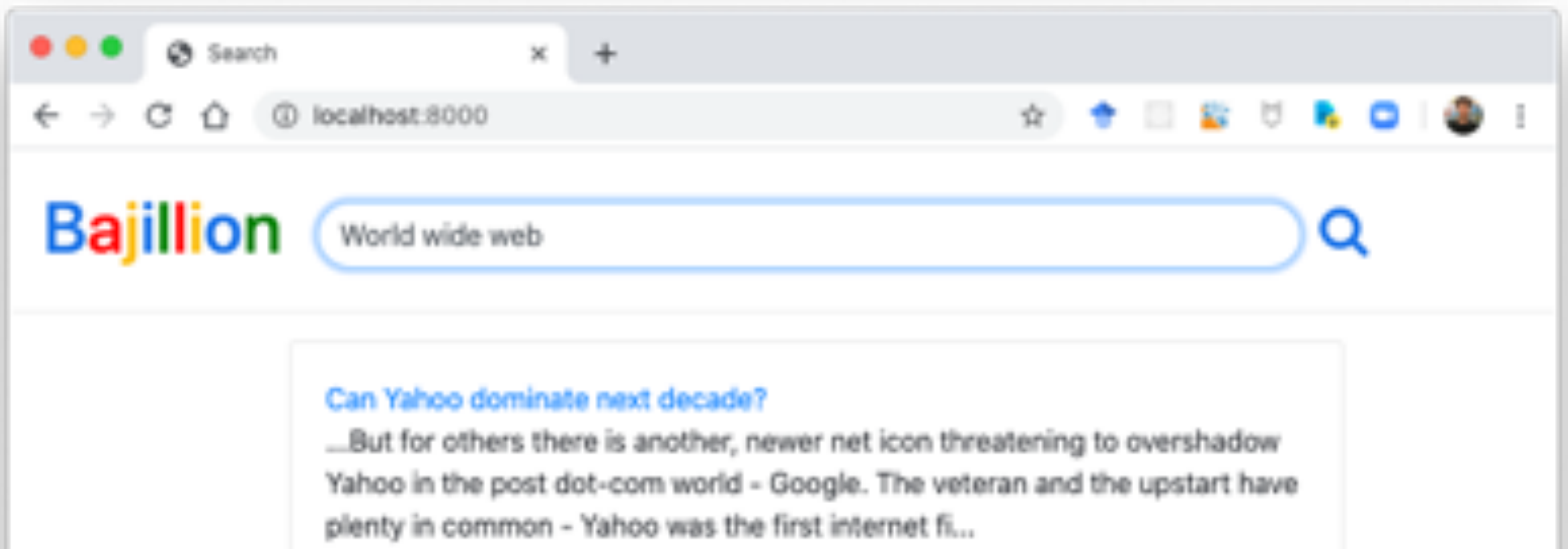
The Internet

Chris Piech and Mehran Sahami

CS106A, Stanford University

Housekeeping

- Last assignment, Bajillion, is due next Wednesday
- Today, easiest, most fun extra credit in cs106a ☺

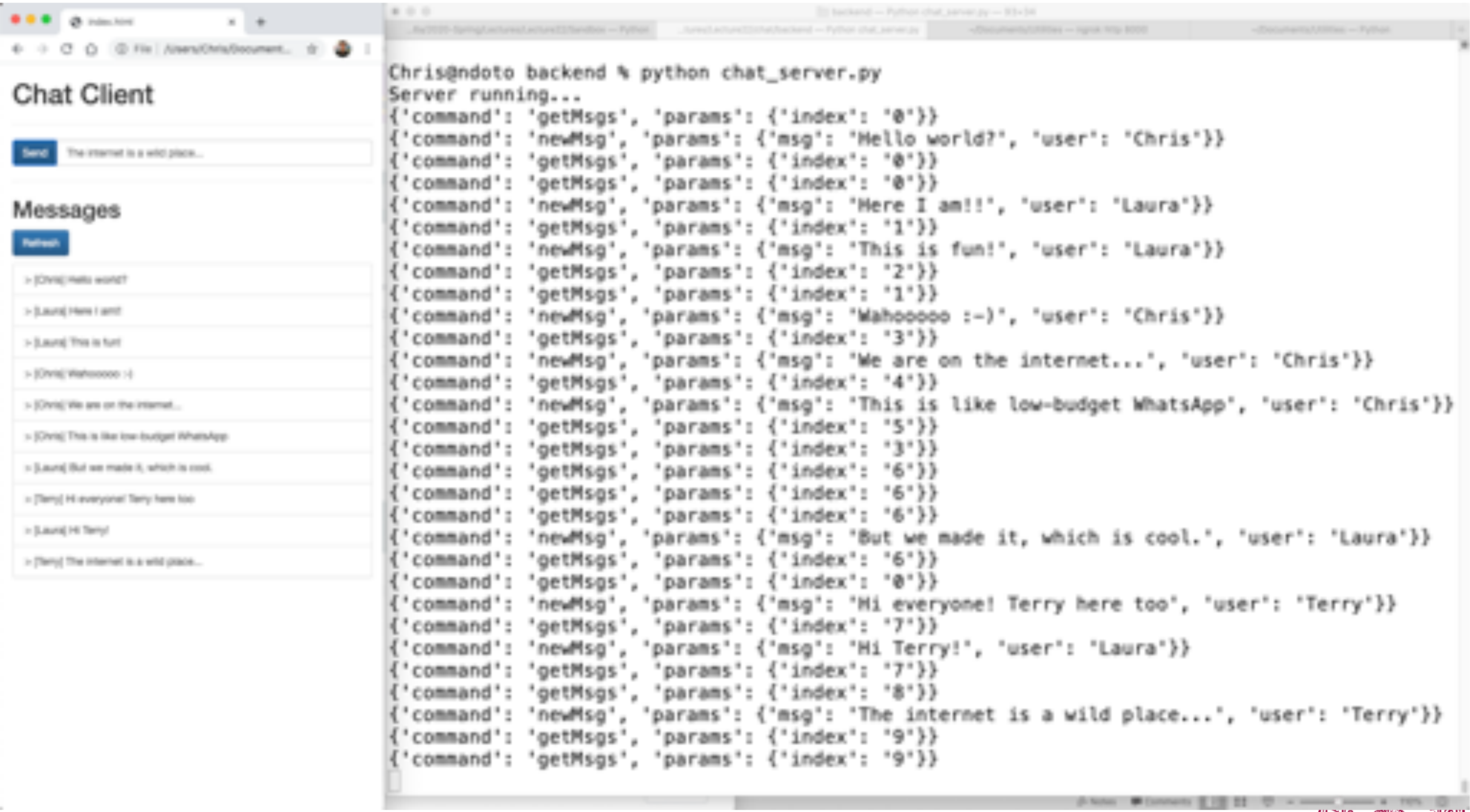


World wide web

Can Yahoo dominate next decade?

...But for others there is another, newer net icon threatening to overshadow Yahoo in the post dot-com world - Google. The veteran and the upstart have plenty in common - Yahoo was the first internet fi...

First, a cool demo



Chat Client

The internet is a wild place...

Messages

> [Chris] Hello world?
> [Laura] Here I am!!
> [Laura] This is fun!
> [Chris] Wahoooooo :-)
> [Chris] We are on the internet...
> [Chris] This is like low-budget WhatsApp
> [Laura] But we made it, which is cool.
> [Terry] Hi everyone! Terry here too
> [Laura] Hi Terry!
> [Terry] The internet is a wild place...

```
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': 'Wahoooooo :-)', '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'}}
```

Guiding question for today:

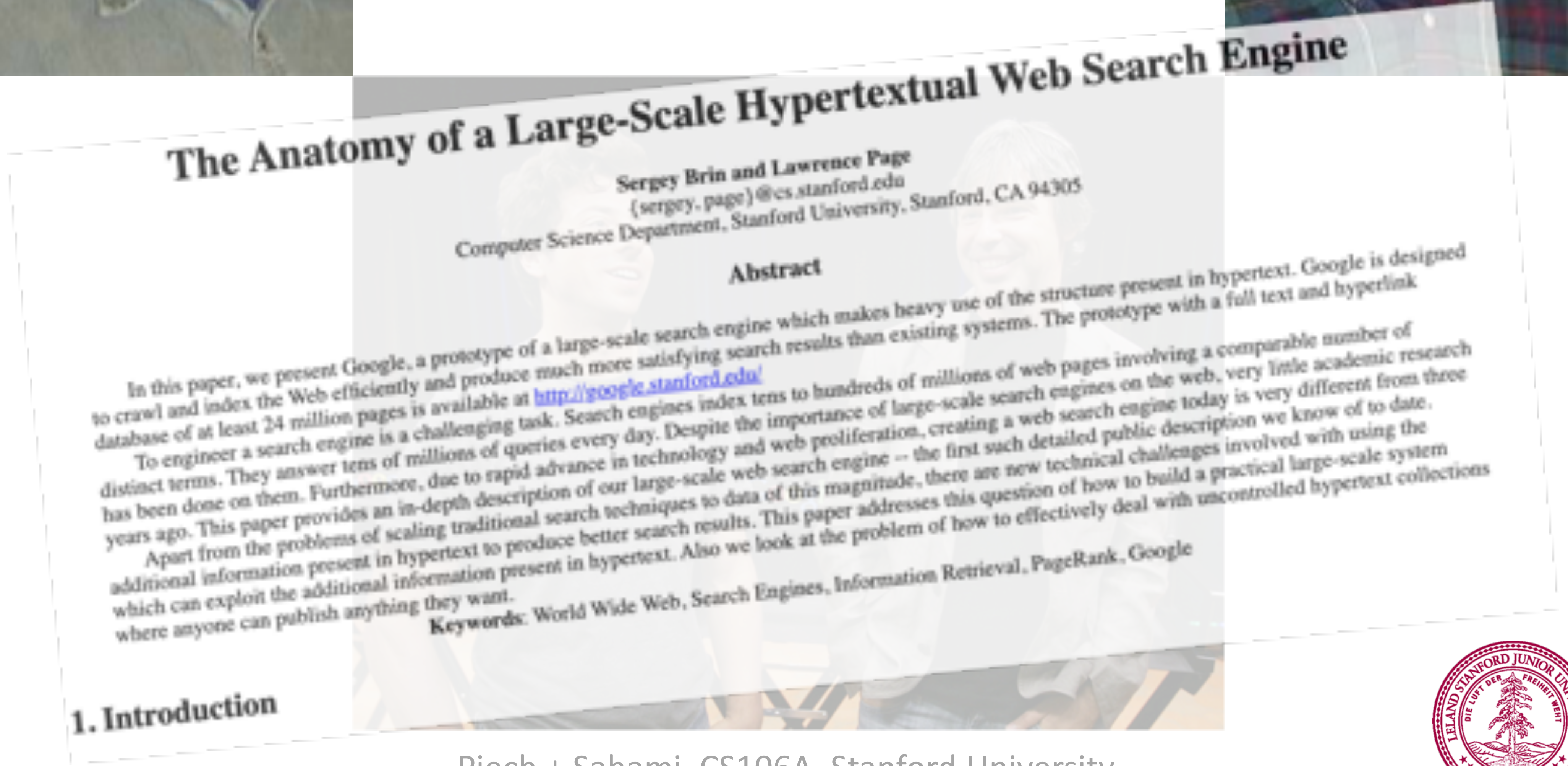
How do you write programs that interact with the **internet**?

Ethics of Search

Ethics of Search

Ethics of Search

Ethics of Search



The Anatomy of a Large-Scale Hypertextual Web Search Engine

Sergey Brin and Lawrence Page
{sergey, page}@cs.stanford.edu
Computer Science Department, Stanford University, Stanford, CA 94305

Abstract

In this paper, we present Google, a prototype of a large-scale search engine which makes heavy use of the structure present in hypertext. Google is designed to crawl and index the Web efficiently and produce much more satisfying search results than existing systems. The prototype with a full text and hyperlink database of at least 24 million pages is available at http://google.stanford.edu/

To engineer a search engine is a challenging task. Search engines index tens to hundreds of millions of web pages involving a comparable number of distinct terms. They answer tens of millions of queries every day. Despite the importance of large-scale search engines on the web, very little academic research has been done on them. Furthermore, due to rapid advance in technology and web proliferation, creating a web search engine today is very different from three years ago. This paper provides an in-depth description of our large-scale web search engine -- the first such detailed public description we know of to date.

Apart from the problems of scaling traditional search techniques to data of this magnitude, there are new technical challenges involved with using the additional information present in hypertext to produce better search results. This paper addresses this question of how to build a practical large-scale system which can exploit the additional information present in hypertext. Also we look at the problem of how to effectively deal with uncontrolled hypertext collections where anyone can publish anything they want.

Keywords: World Wide Web, Search Engines, Information Retrieval, PageRank, Google

1. Introduction

Ethics of Search

"we expect that advertising funded search engines will be inherently biased towards the advertisers and away from the needs of the consumers ...

Since it is very difficult even for experts to evaluate search engines, search engine bias is particularly insidious. ...

a search engine could add a small factor to search results from "friendly" companies, and subtract a factor from results from competitors. ...

[W]e believe the issue of advertising causes enough mixed incentives that it is crucial to have a competitive search engine that is transparent and in the academic realm."

Brin & Page 1998

What is Bias in Search?

Possible Concerns about Bias in Search:

(1) "search-engine technology is not neutral, but instead has embedded features in its design that favor some values over others"?

Possible Concerns about Bias in Search:

(1) "search-engine technology is not neutral, but instead has embedded features in its design that favor some values over others"?

- "relevance" to user
- "quality" of results

Possible Concerns about Bias in Search:

(1) "search-engine technology is not neutral, but instead has embedded features in its design that favor some values over others"?

- "relevance" to user
- "quality" of results

Thick normative terms that encode values!

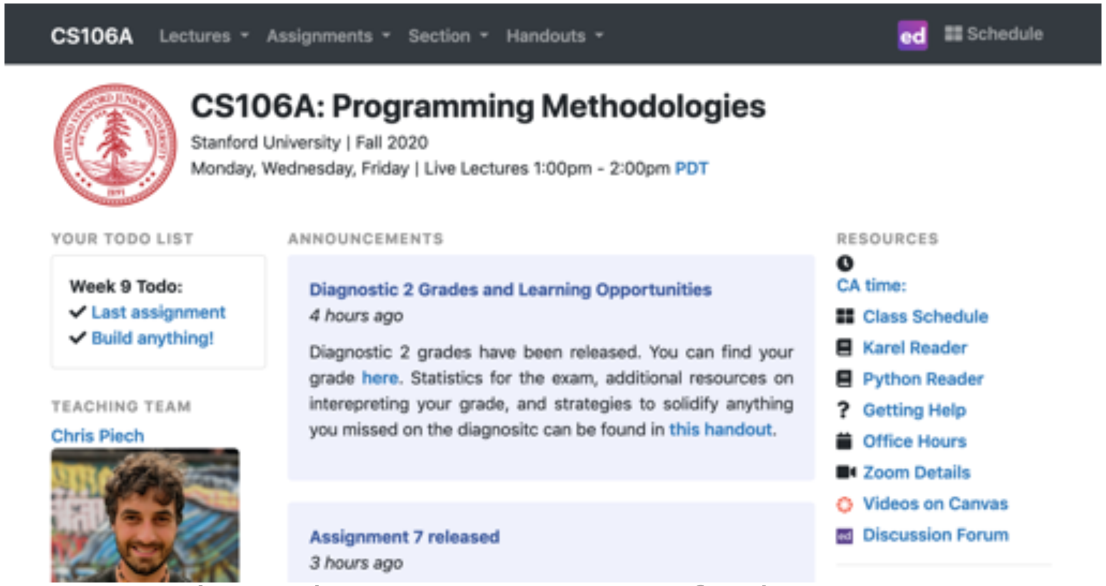
Possible Concerns about Bias in Search:

(2) "major search engines systematically favor some sites (and some kinds of sites) over others in the lists of results they return in response to user search queries"?

Concerns about Bias in Search:

(2) "major search engines systematically favor some sites (and some kinds of sites) over others in the lists of results they return in response to user search queries"?

"High quality website"



CS106A: Programming Methodologies

Stanford University | Fall 2020

Monday, Wednesday, Friday | Live Lectures 1:00pm - 2:00pm PDT

YOUR TODO LIST

Week 9 Todo:
- ✓ Last assignment
- ✓ Build anything!

TEACHING TEAM

Chris Piech

ANNOUNCEMENTS

Diagnostic 2 Grades and Learning Opportunities

4 hours ago

Diagnostic 2 grades have been released. You can find your grade here. Statistics for the exam, additional resources on interepreting your grade, and strategies to solidify anything you missed on the diagnositc can be found in this handout.

Assignment 7 released

3 hours ago

RESOURCES

CA time:
- Class Schedule
- Karel Reader
- Python Reader
- Getting Help
- Office Hours
- Zoom Details
- Videos on Canvas
- Discussion Forum

Concerns about Bias in Search:

(2) "major search engines systematically favor some sites (and some kinds of sites) over others in the lists of results they return in response to user search queries"?

"low quality website"

mehranandchrisburritos.com

Nothing links to it,

doesn't actually help you find burritos

Concerns about Bias in Search:

(3) "search algorithms do not use **objective criteria** in generating their lists of results for search queries"?

Our criteria are

- "relevance" to user
- "quality" of results

Relevance is a subjective metric.

It can't be determined without asking, relevant to whom?

Relevance and Advertising

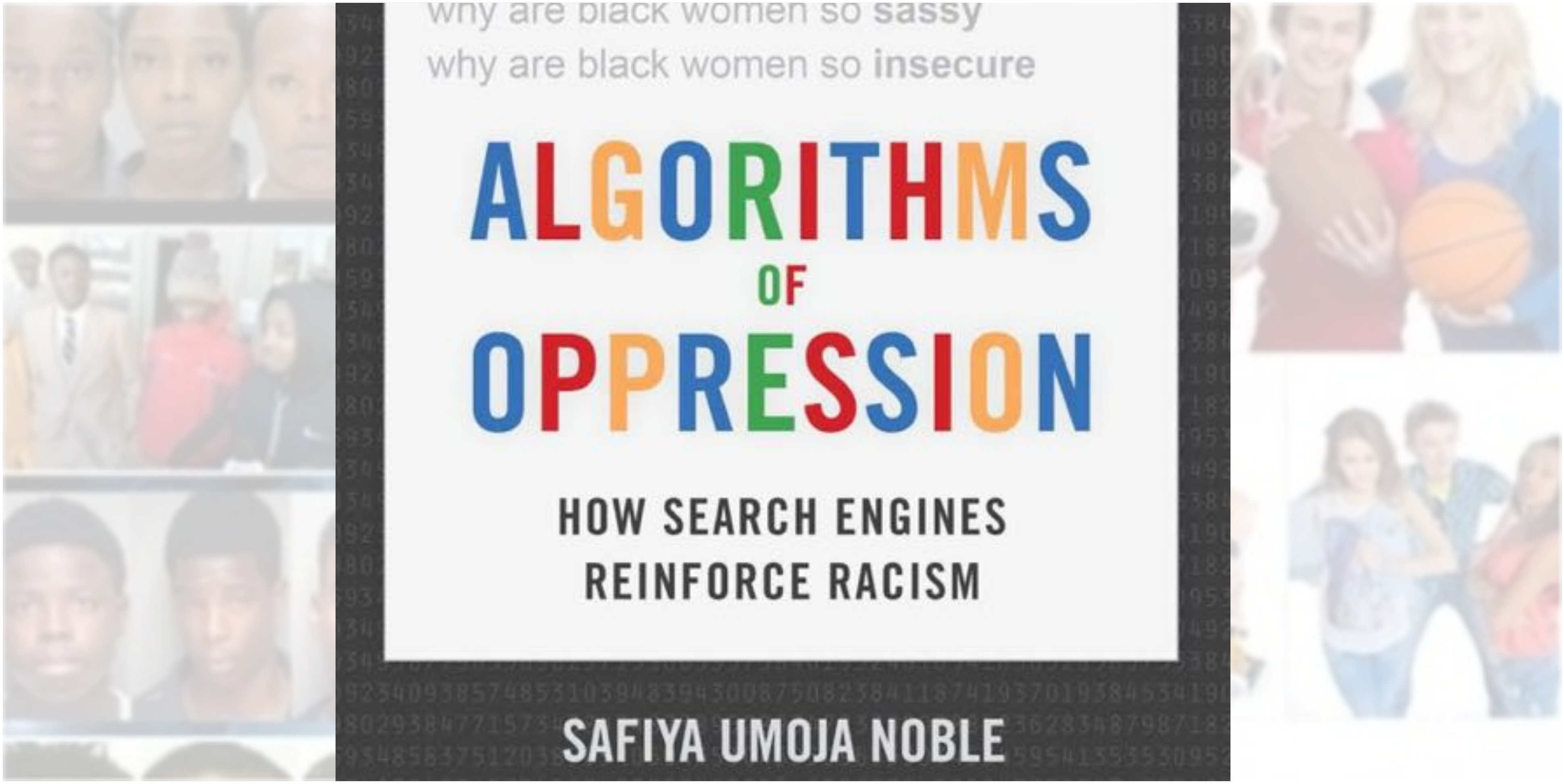
How might advertising affect relevance?

Advertising bias might lead to delivering what the advertiser wants the user to see, rather than what the user considers relevant.

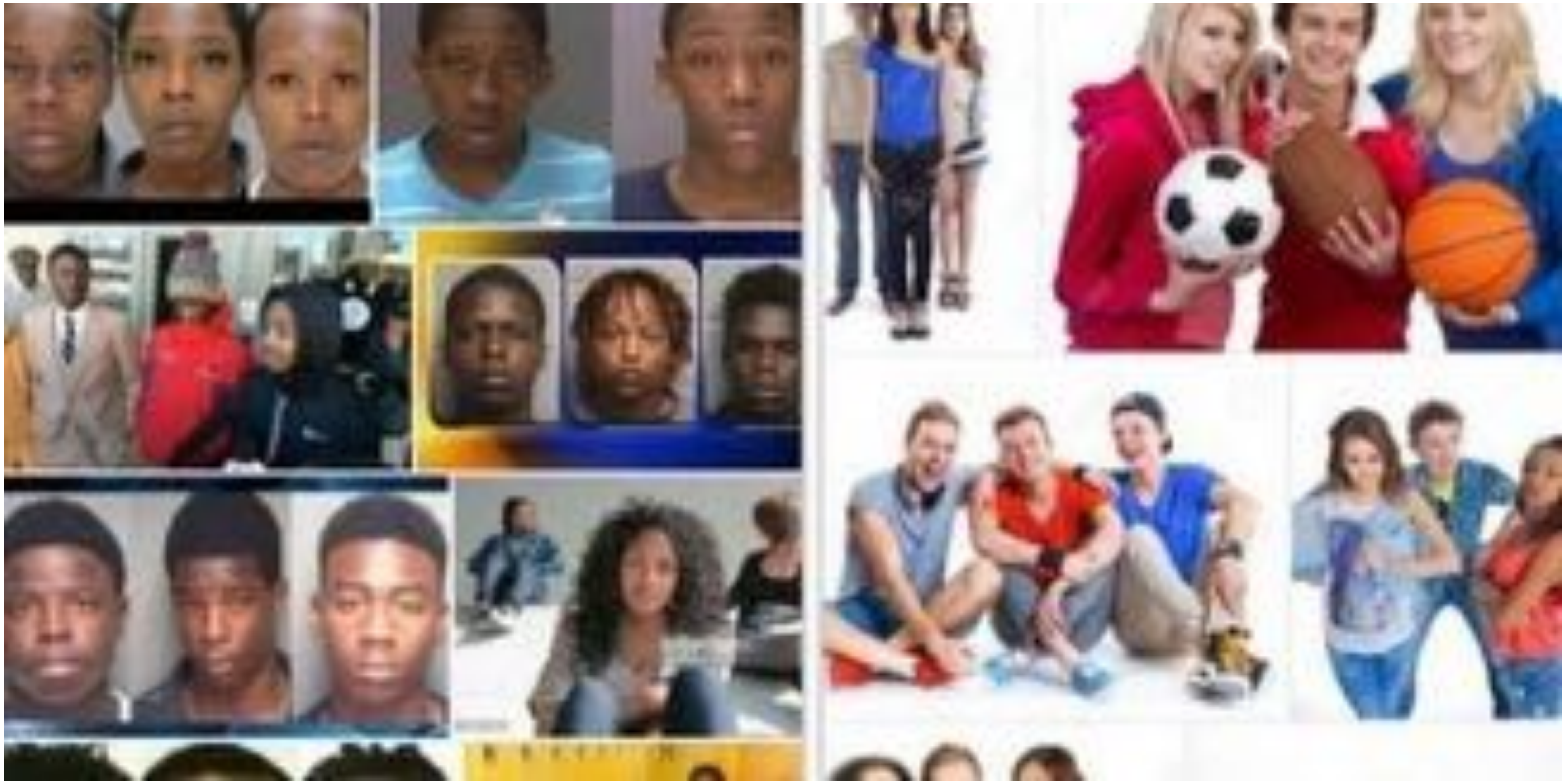
Hence advertising bias would move search results further from the subjective success of delivering the user's desired results.

What about quality? Is that an objective metric?

Bias, Quality, and Lack of Objectivity


why are black women so sassy
why are black women so insecure
ALGORITHMS OF OPPRESSION
HOW SEARCH ENGINES REINFORCE RACISM
SAFIYA UMOJA NOBLE

Bias, Quality, and Lack of Objectivity



Search for "three black teenagers" vs "three white teenagers"

Objectivity: Definitions

Objectivity as freedom from bias?

Objectivity as faithfulness to facts?

Objectivity as absence of normative commitments?

Objectivity as integration of multiple perspectives & reflexivity about values in the methodology?

Objectivity: Definitions

Objectivity as integration of multiple perspectives & reflexivity about values in the methodology

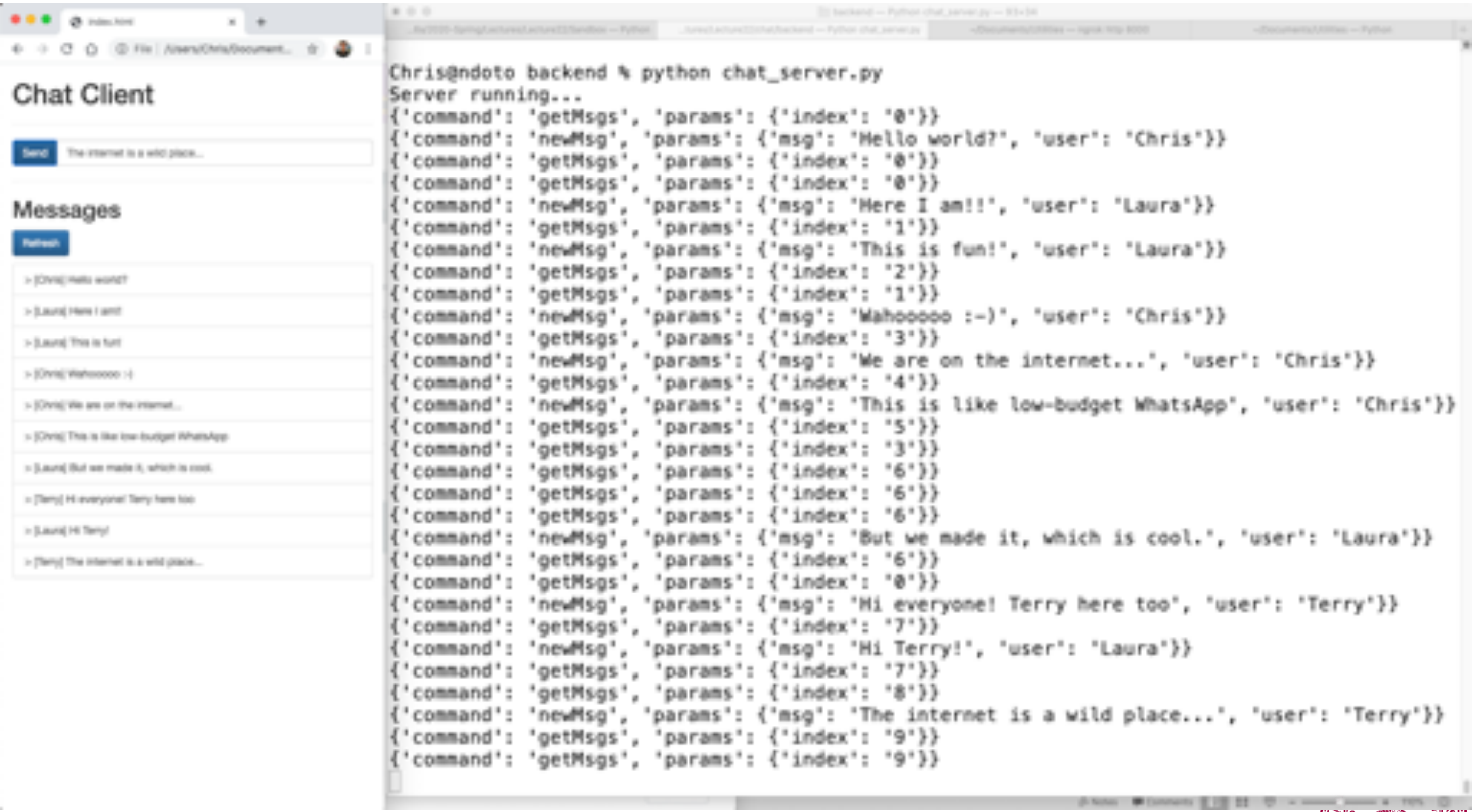
Personalization & Democracy

Search engines have been called the "gatekeepers of the web," tools that "structure knowledge" for internet users & "contribute to the public use of reason." Does this confer obligation?

Guiding question for today:

How do you write programs that interact with the **internet**?

Recall



Chat Client

The internet is a wild place...

Messages

- > [Chris] Hello world?
- > [Laura] Here I am!!
- > [Laura] This is fun!
- > [Chris] Wahooooo :-)
- > [Chris] We are on the internet...
- > [Chris] This is like low-budget WhatsApp
- > [Laura] But we made it, which is cool.
- > [Terry] Hi everyone! Terry here too
- > [Laura] Hi Terry!
- > [Terry] The internet is a wild place...

```
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'}}
```

<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*

Define New Variable Types

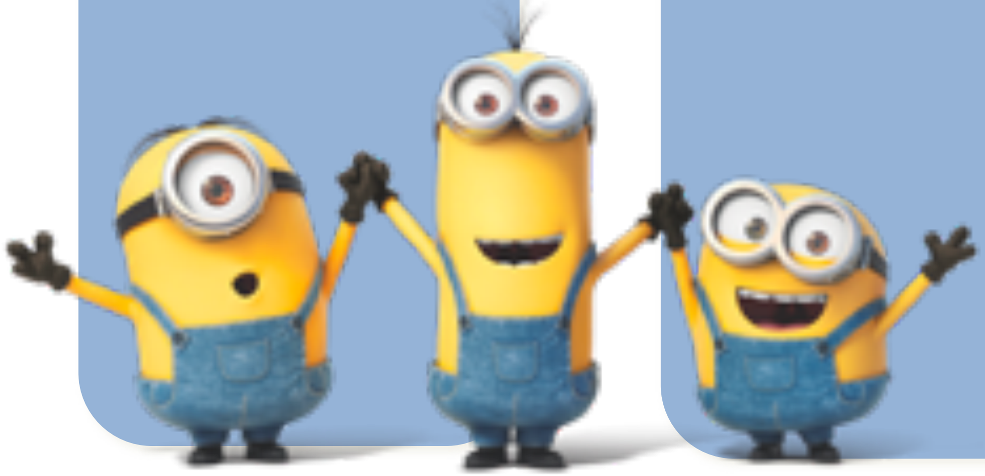
Song

Playlist

User

Song Player

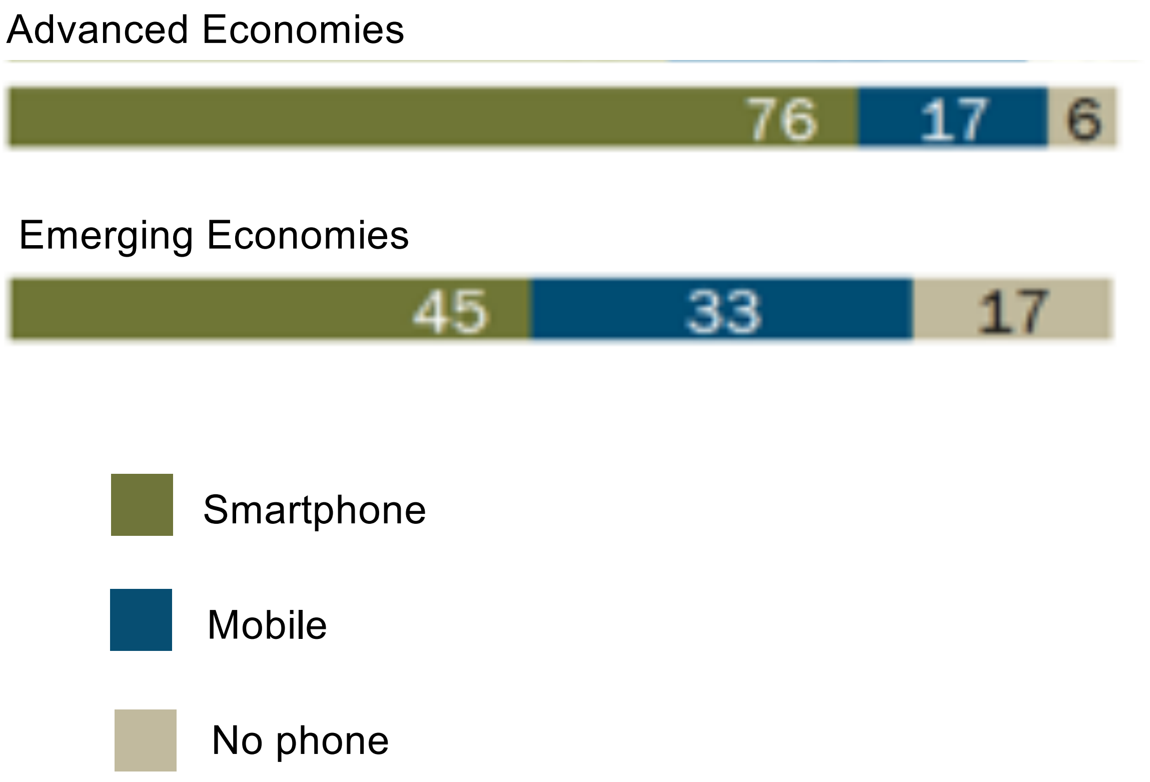
Song Retriever

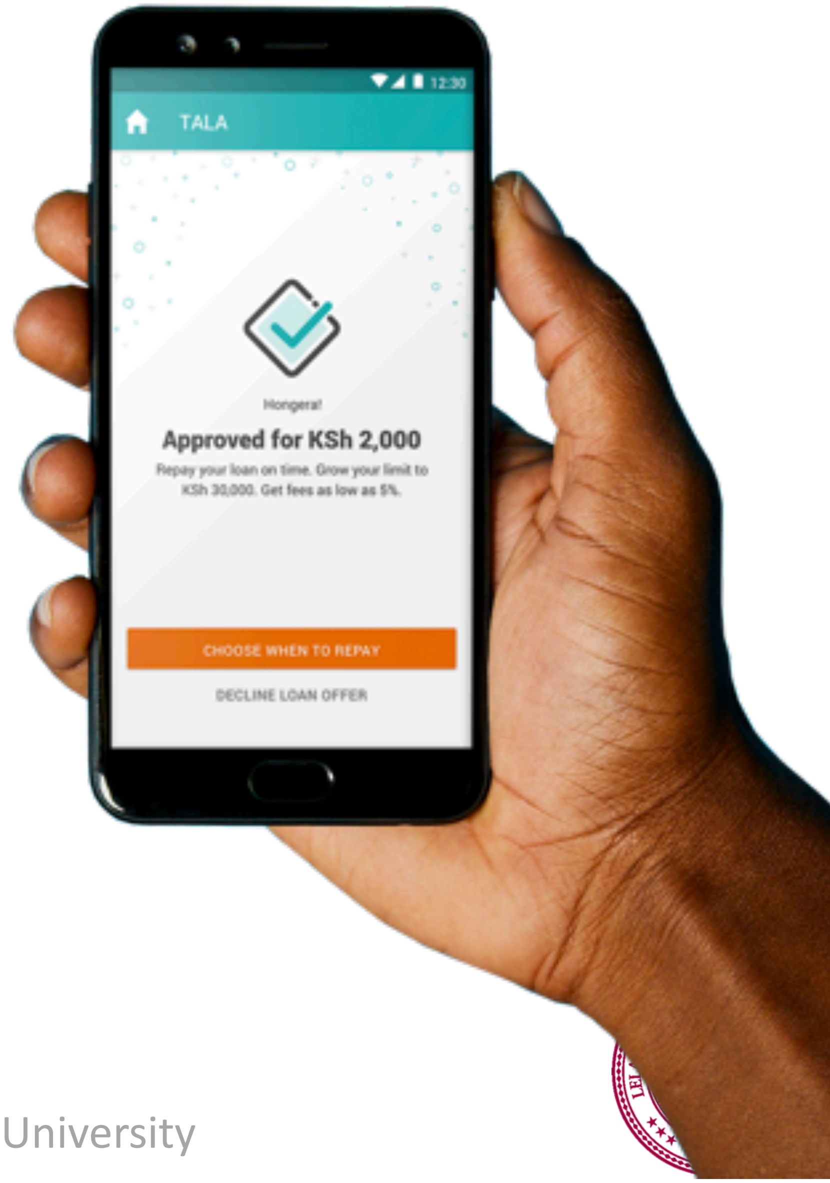


</ review>

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

Smart Phone Access





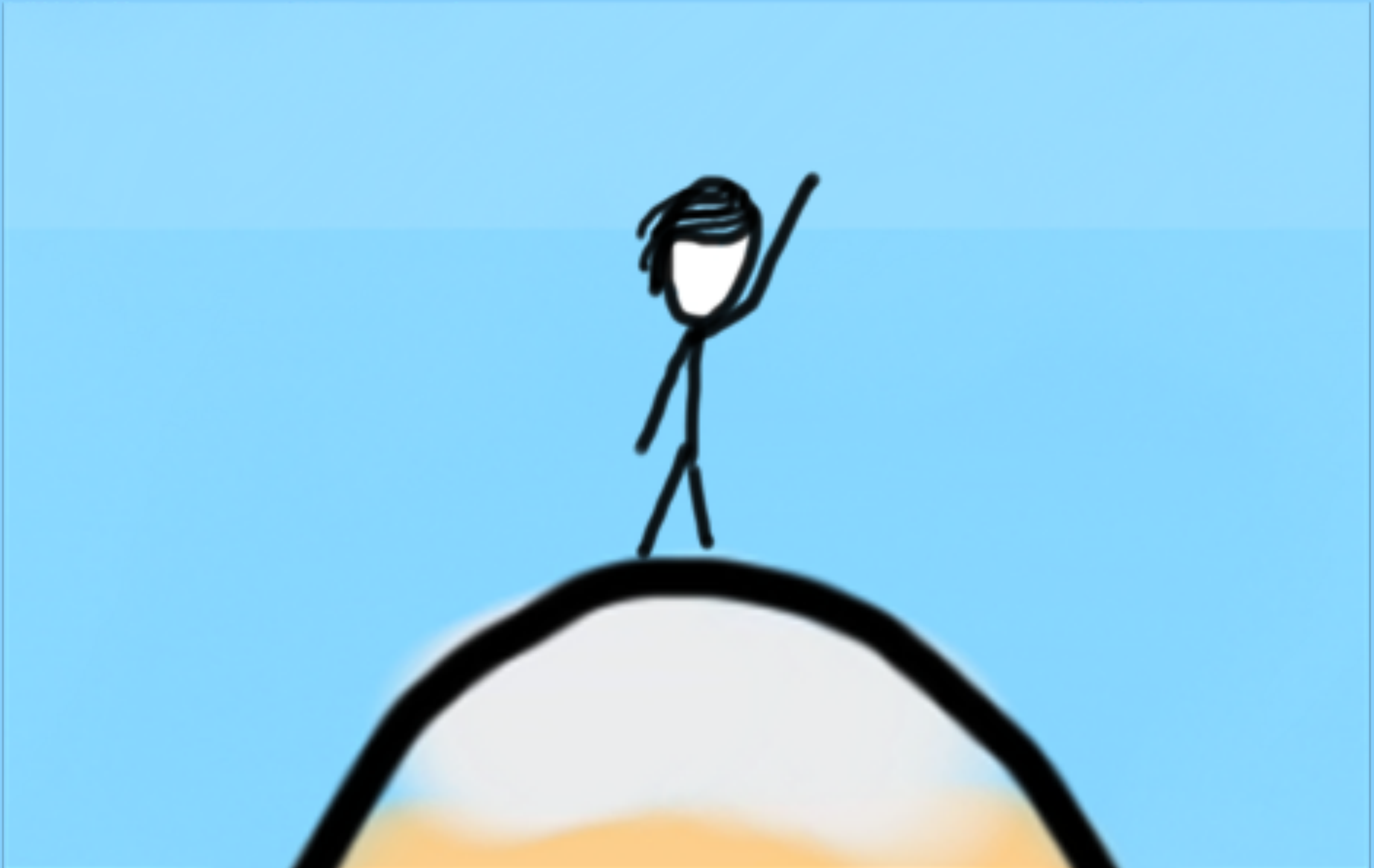
Hongera!

Approved for KSh 2,000

Repay your loan on time. Grow your limit to KSh 30,000. Get fees as low as 5%.

Learning Goals

1. Write a program that can respond to internet requests

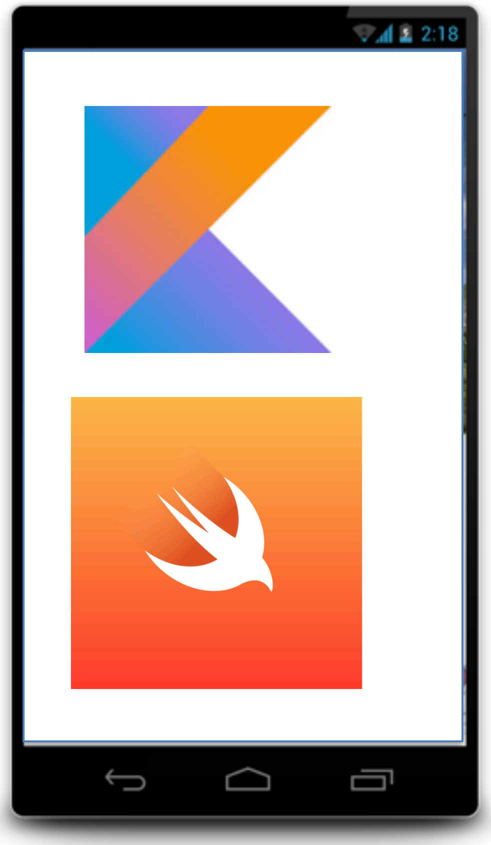


How does your phone communicate with facebook?

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

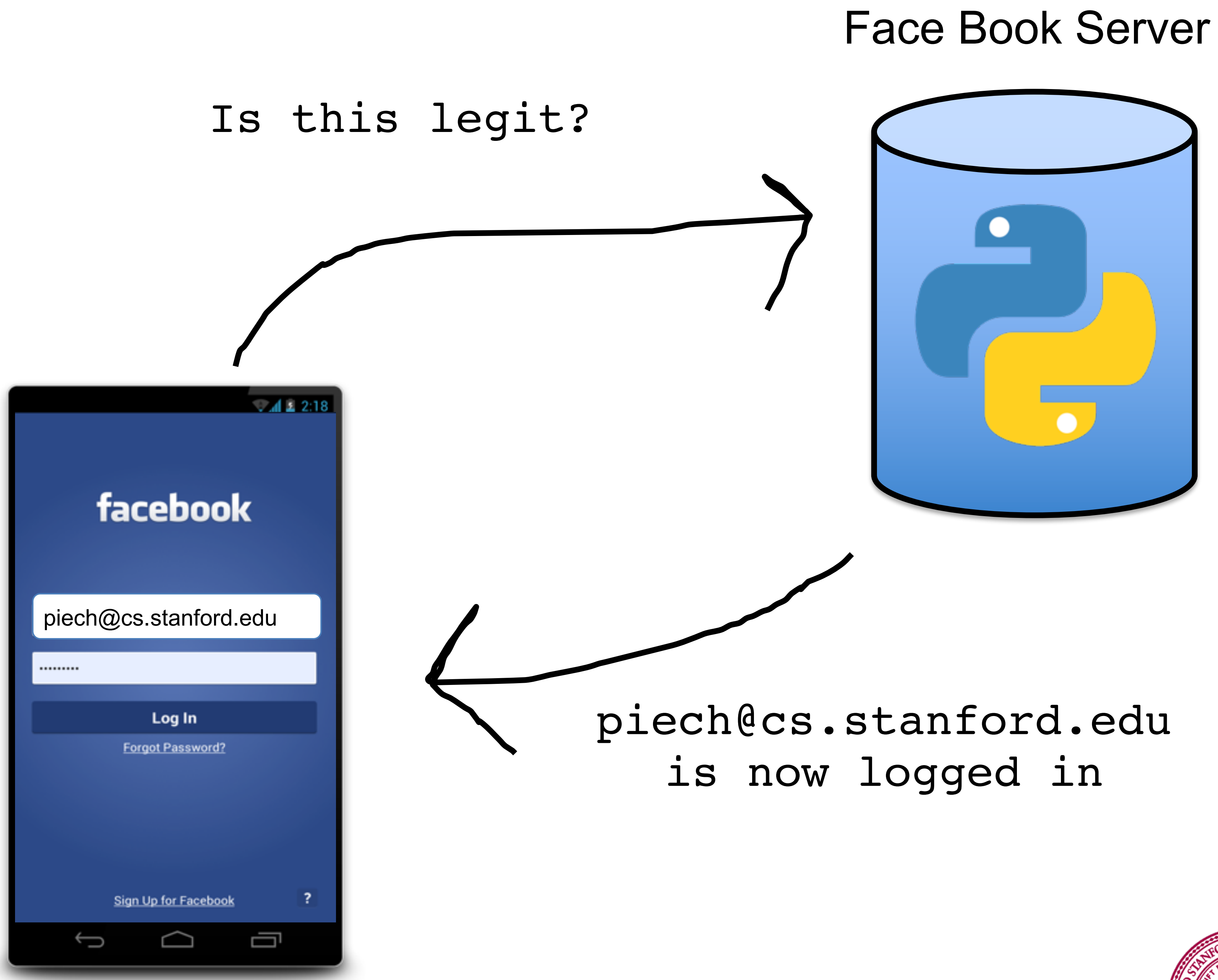
Face Book Server

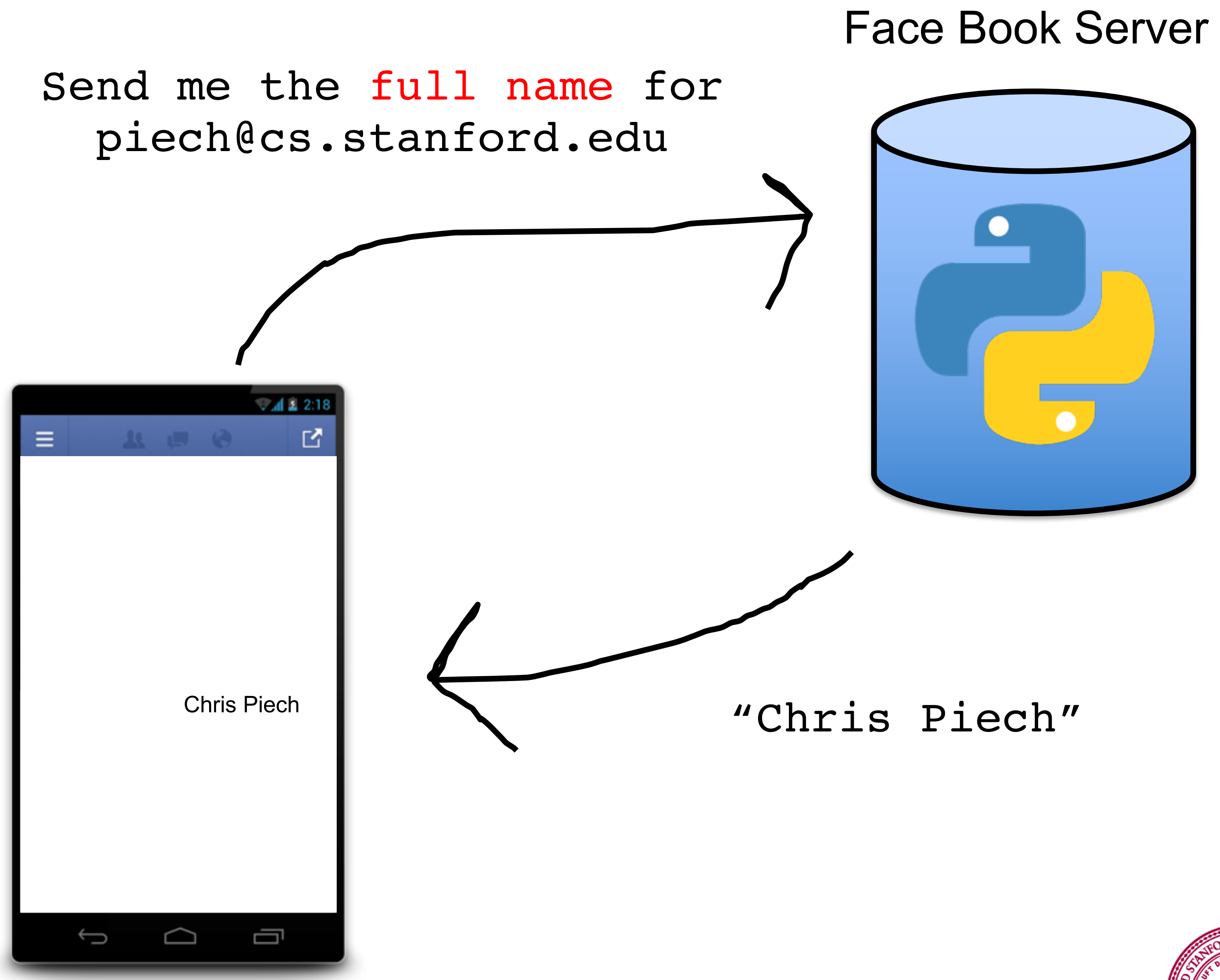
JavaScript with HTML are the languages of websites



Kotlin is the language of Android phones

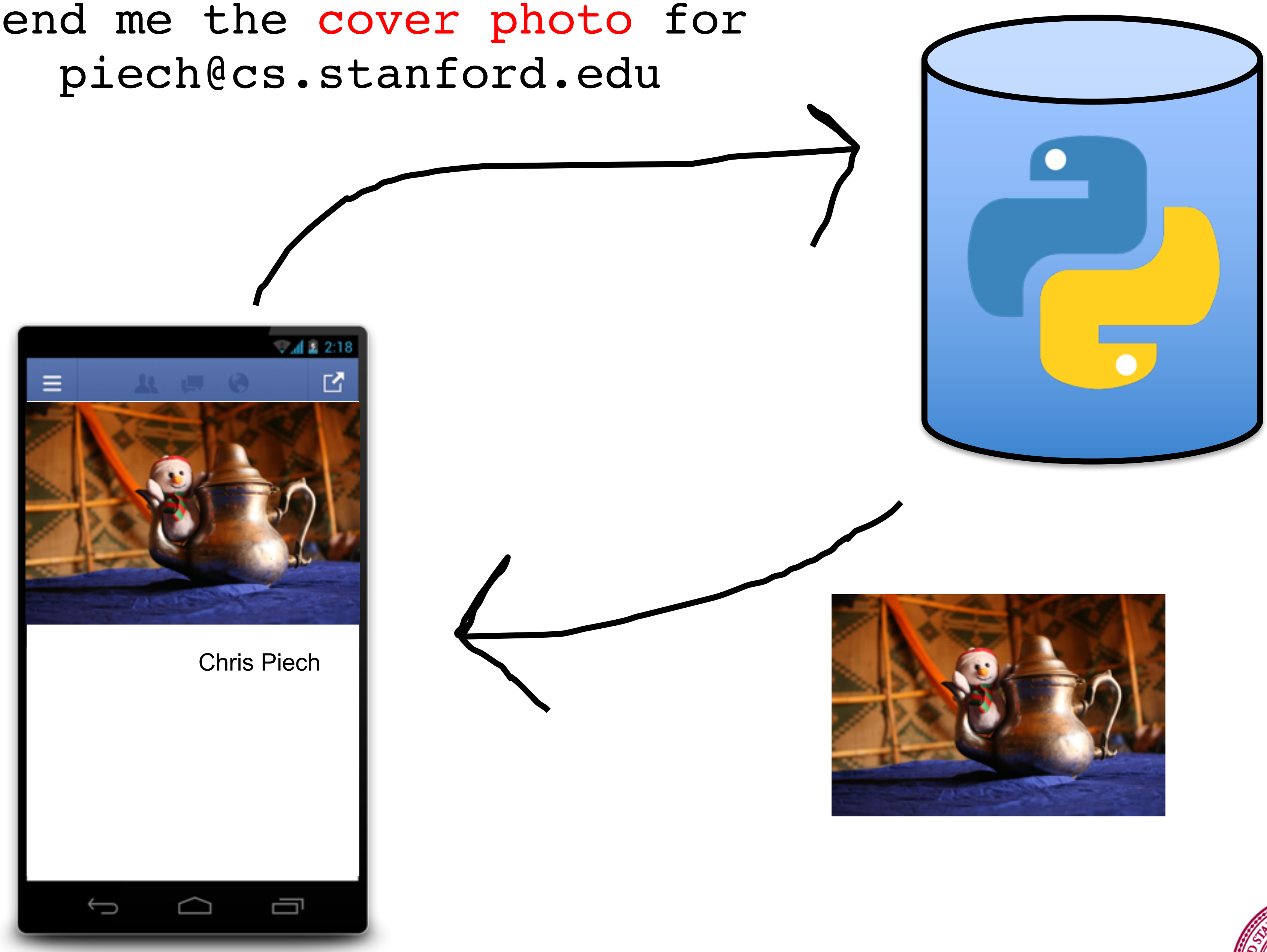
Swift is the language of Apple phones


Face Book Server
Is this legit?
facebook
piech@cs.stanford.edu
Log In
Forgot Password?
Sign Up for Facebook
piech@cs.stanford.edu
is now logged in

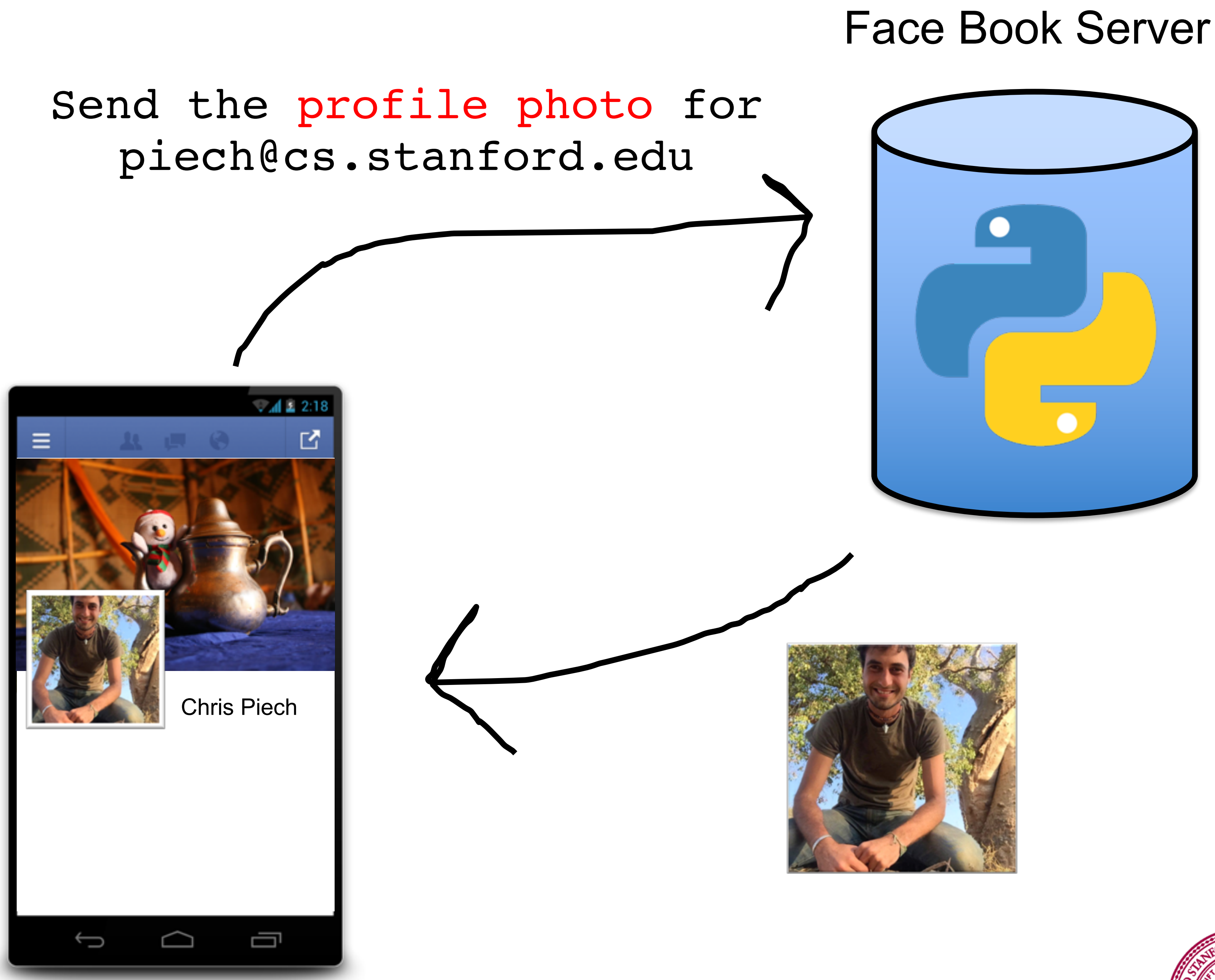

Face Book Server
Send me the full name for piech@cs.stanford.edu
"Chris Piech"
2:18
Chris Piech

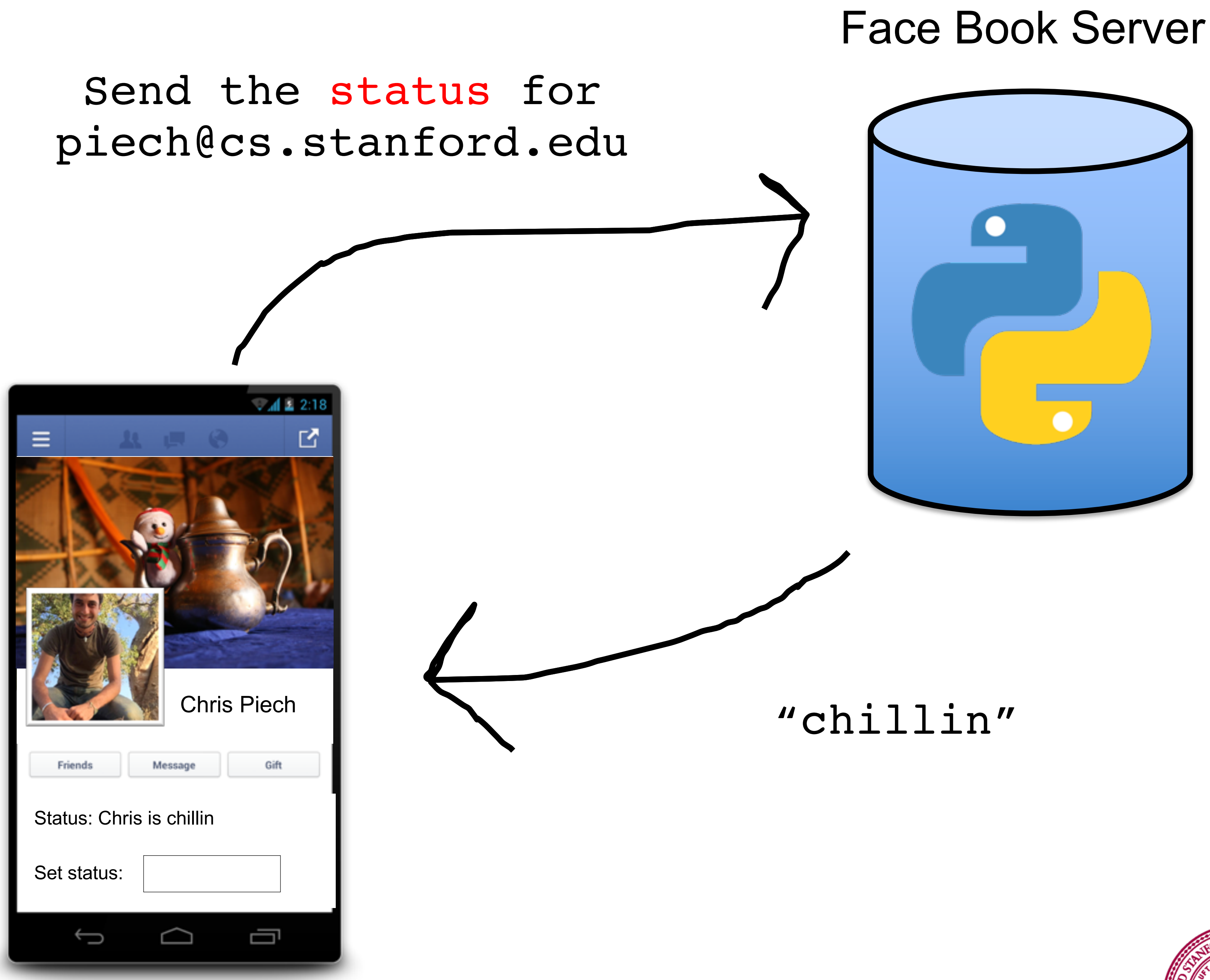
Face Book Server

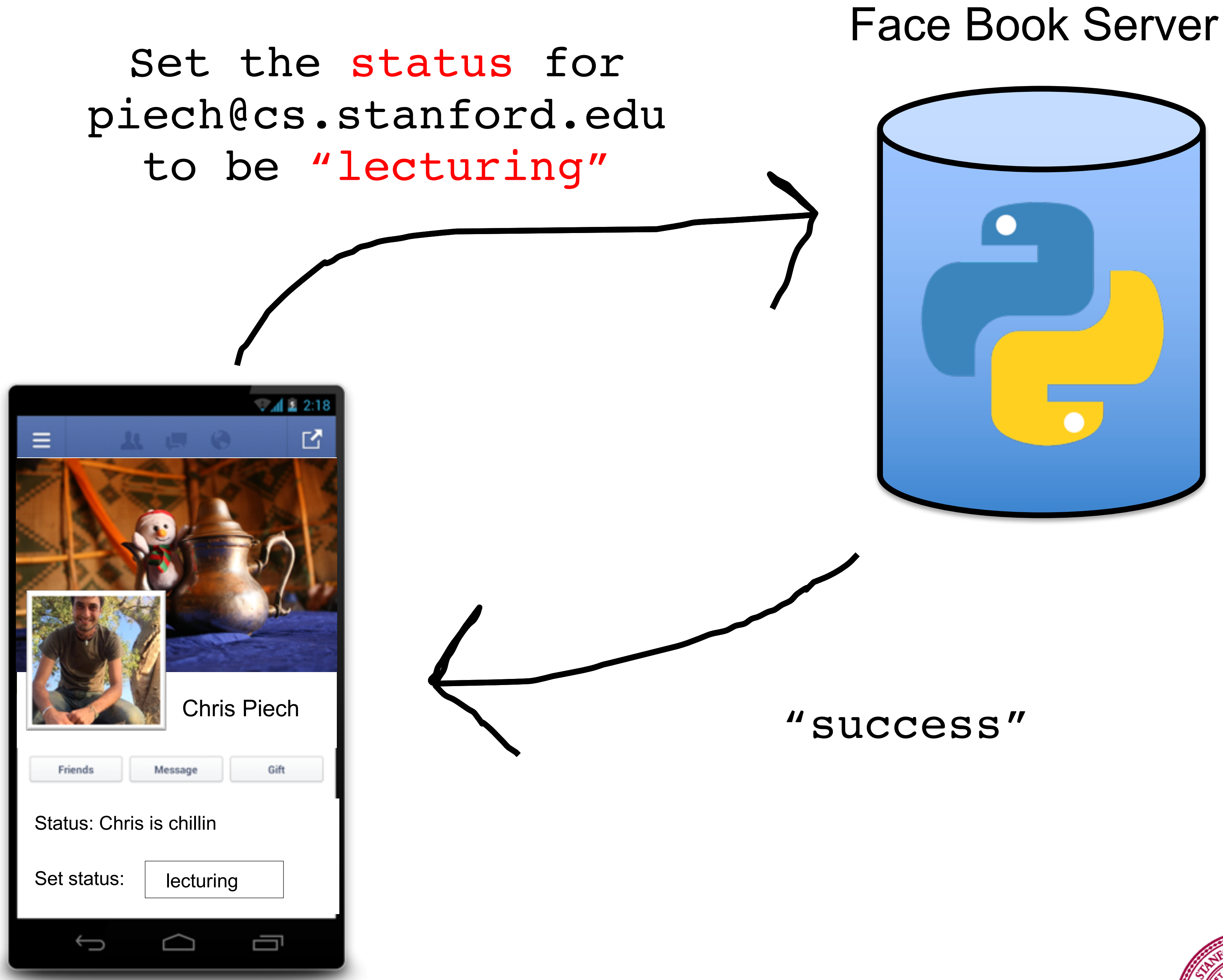
Send me the cover photo for piech@cs.stanford.edu



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




Face Book Server
Send the status for piech@cs.stanford.edu
"chillin"
2:18
Chris Piech
Friends
Message
Gift
Status: Chris is chillin
Set status:



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

"success"

Chris Piech

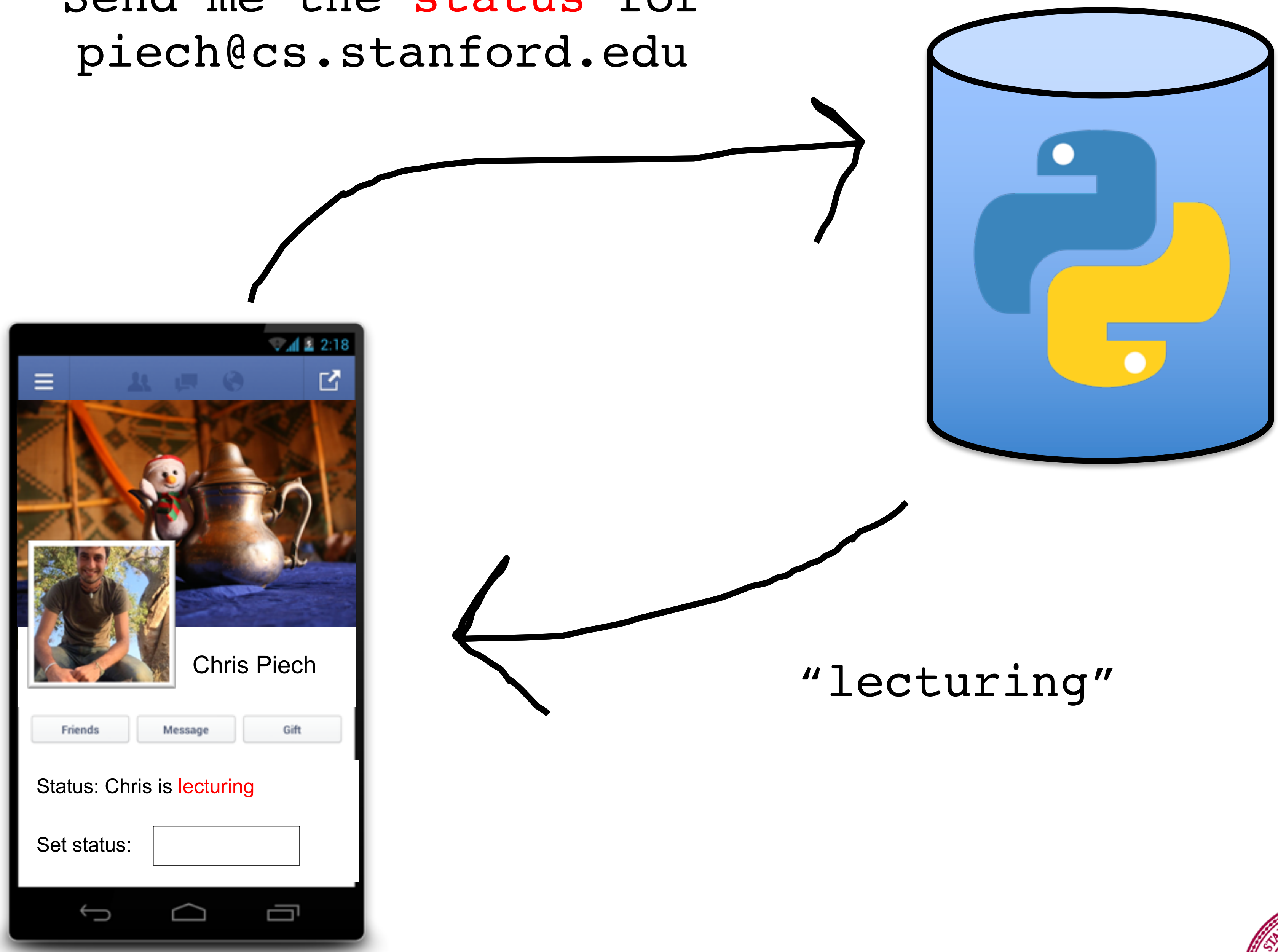
Status: Chris is chillin

Set status: lecturing

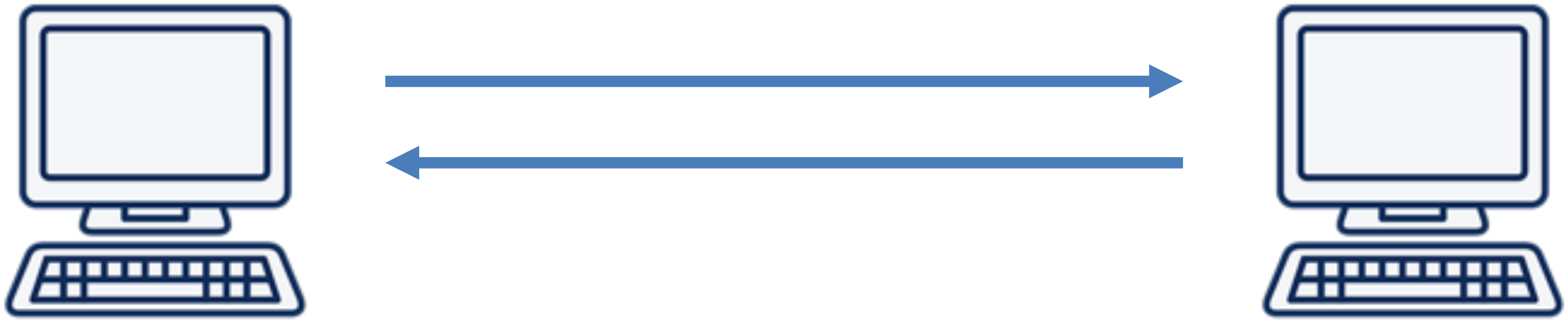
Face Book Server

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

“lecturing”

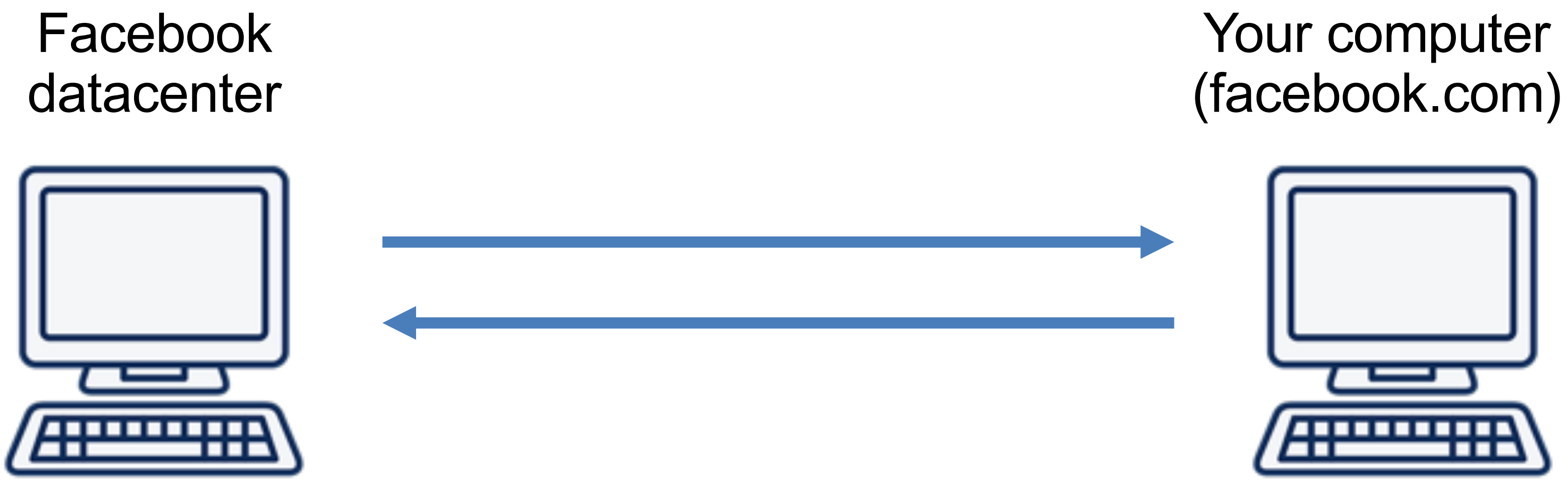


Background: The Internet



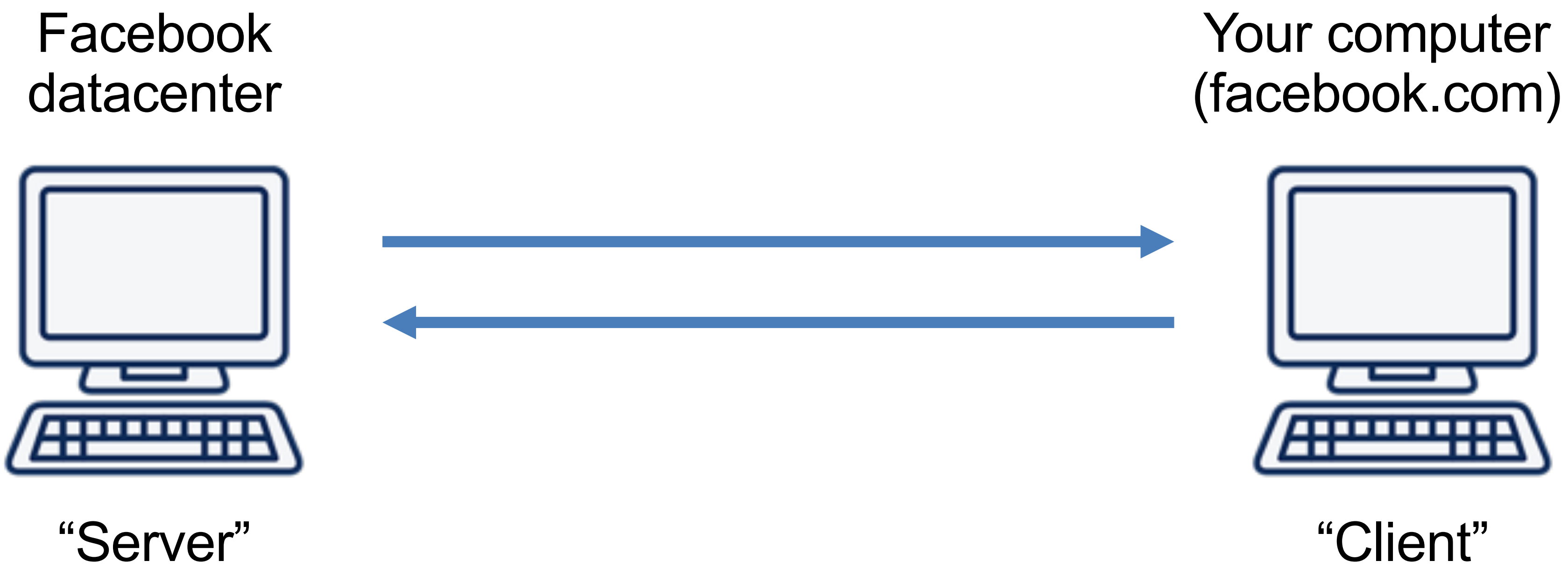
The internet is just many programs sending messages (as *Strings*)

Background: The Internet



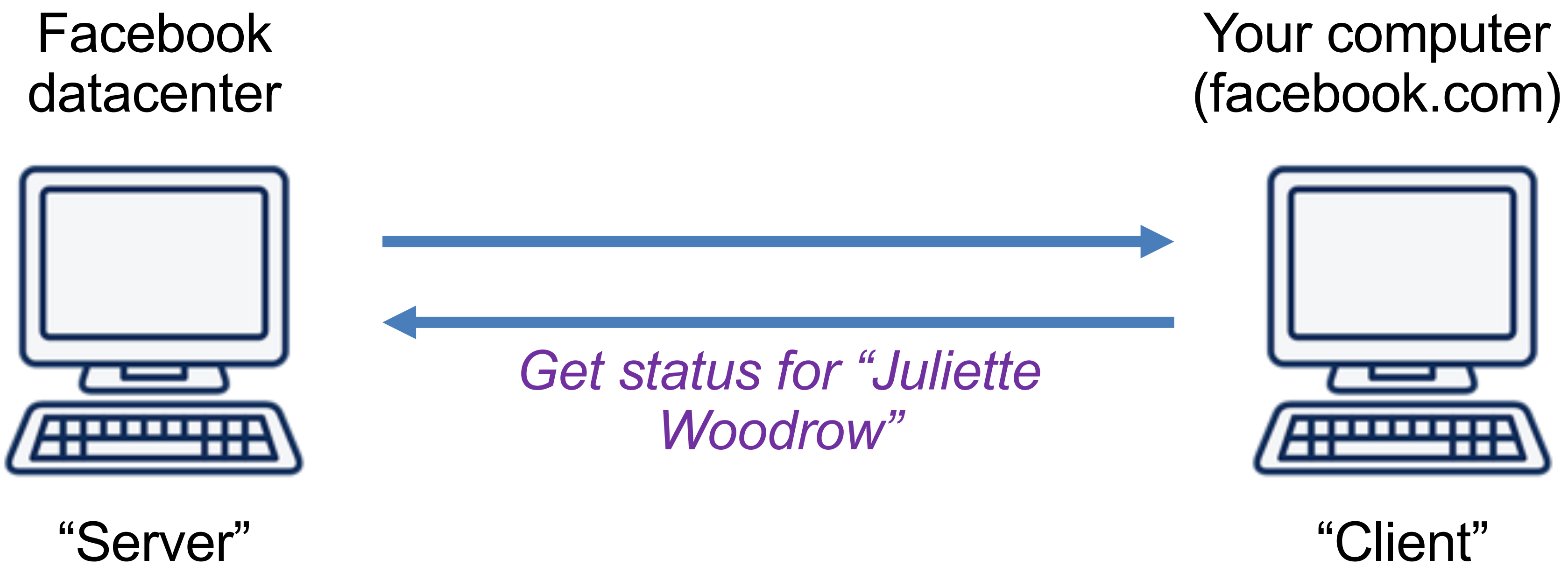
The internet is just many programs sending messages (as *Strings*)

Background: The Internet



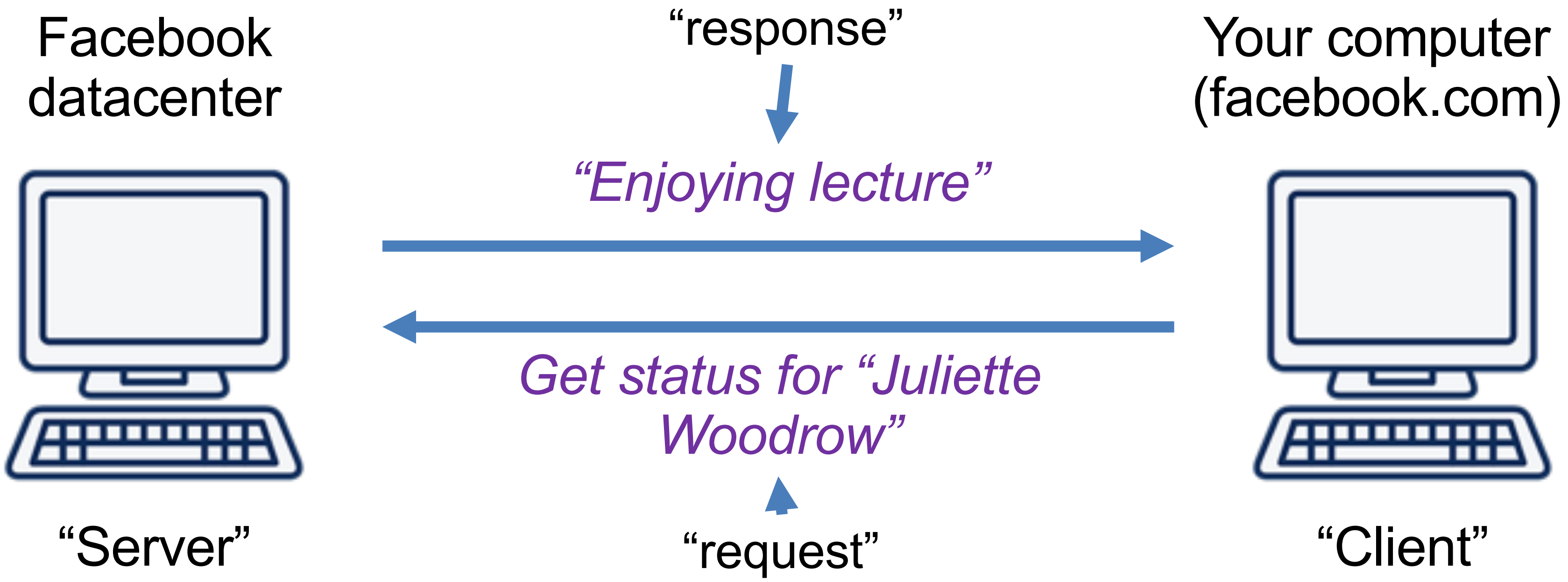
The internet is just many programs sending messages (as *Strings*)

Background: The Internet


Facebook datacenter
Your computer (facebook.com)
Get status for "Juliette Woodrow"
"Server"
"Client"

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

Background: The Internet



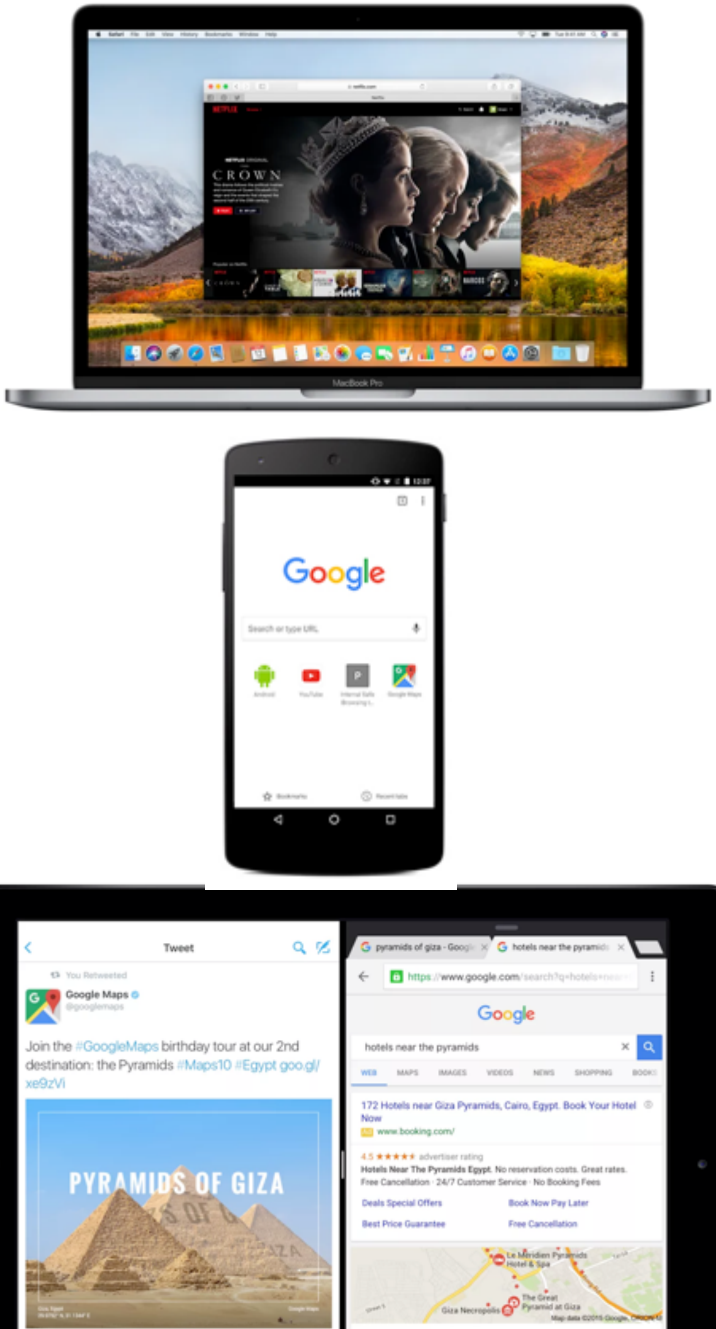
The internet is just many programs sending messages (as *Strings*)

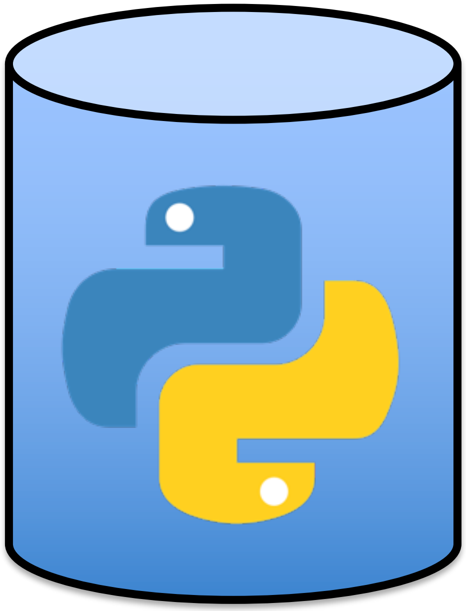
The internet is just many programs sending messages (as strings) to one another

There are two types of internet programs. Servers and Clients

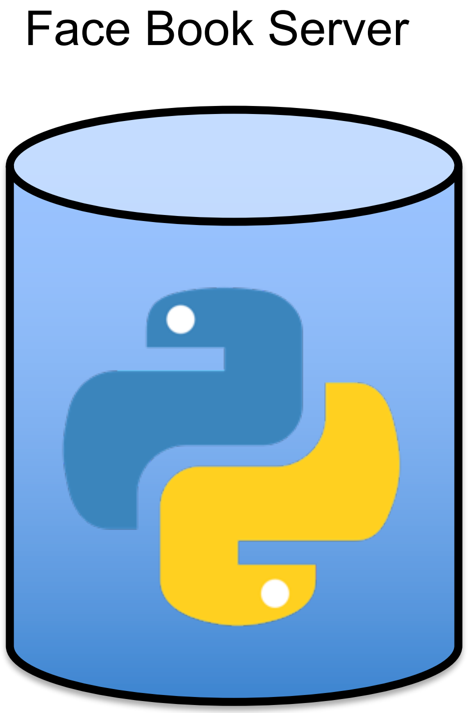
Internet 101

Computers on the internet

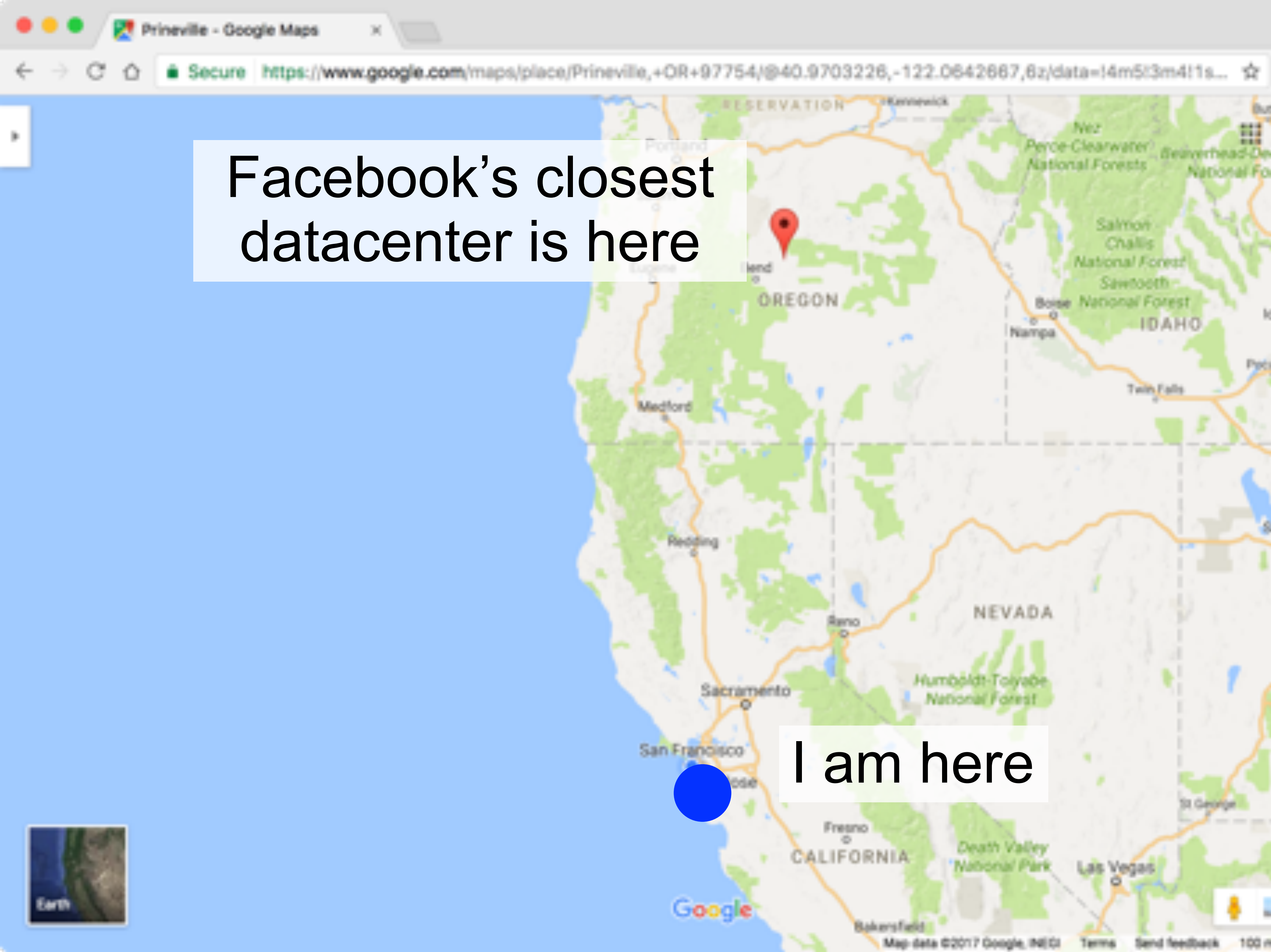


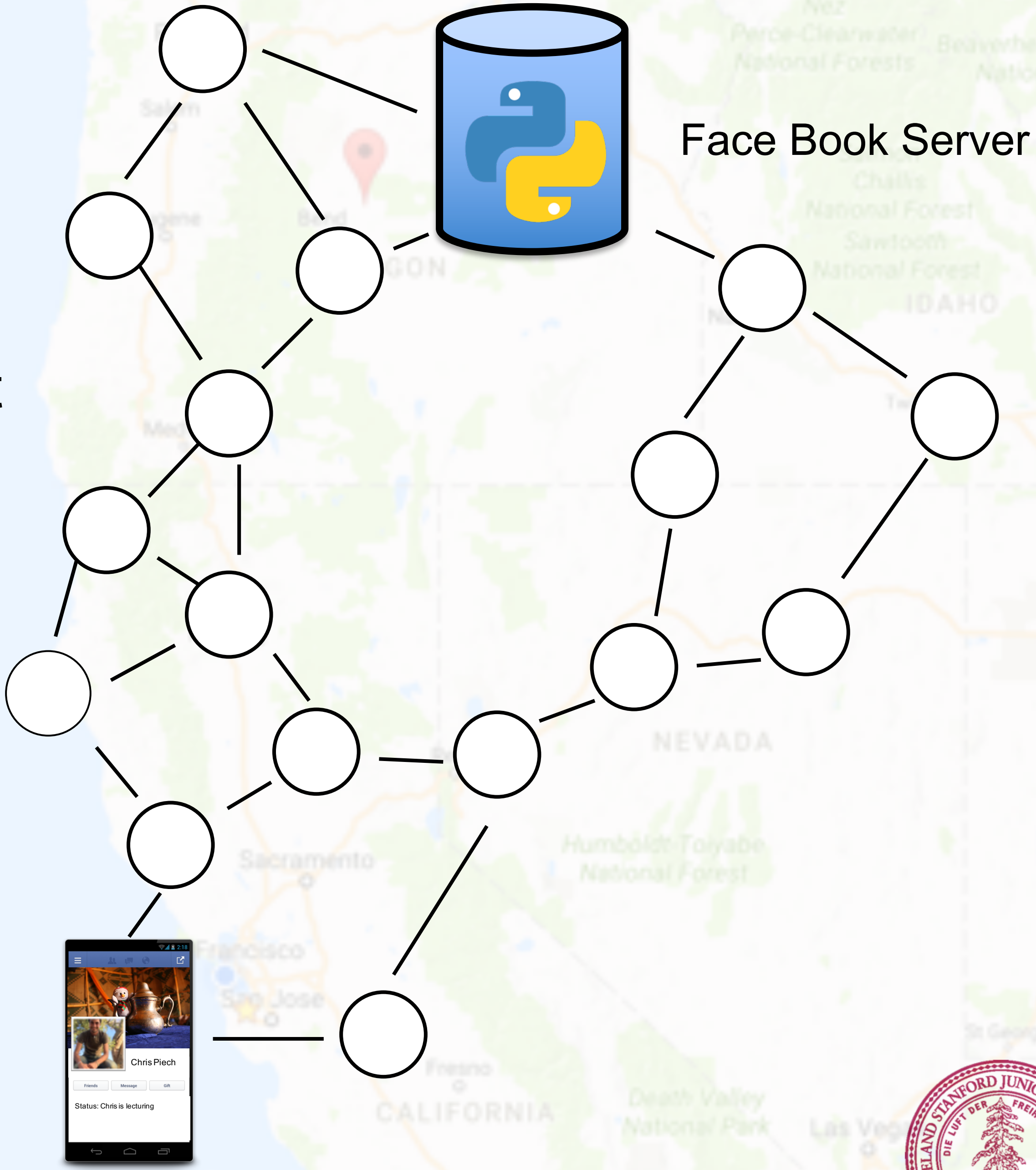


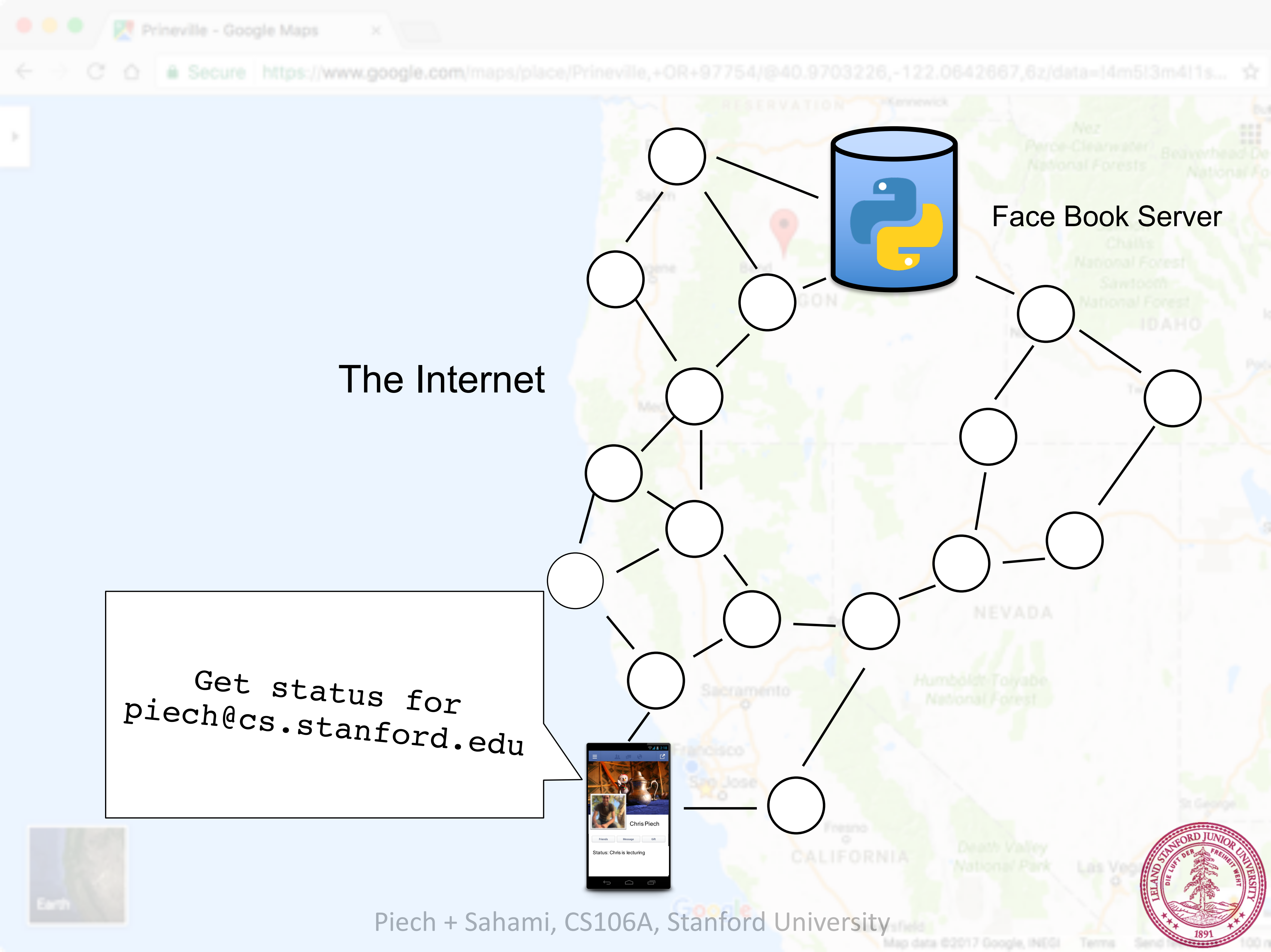
Servers are computers (running code)



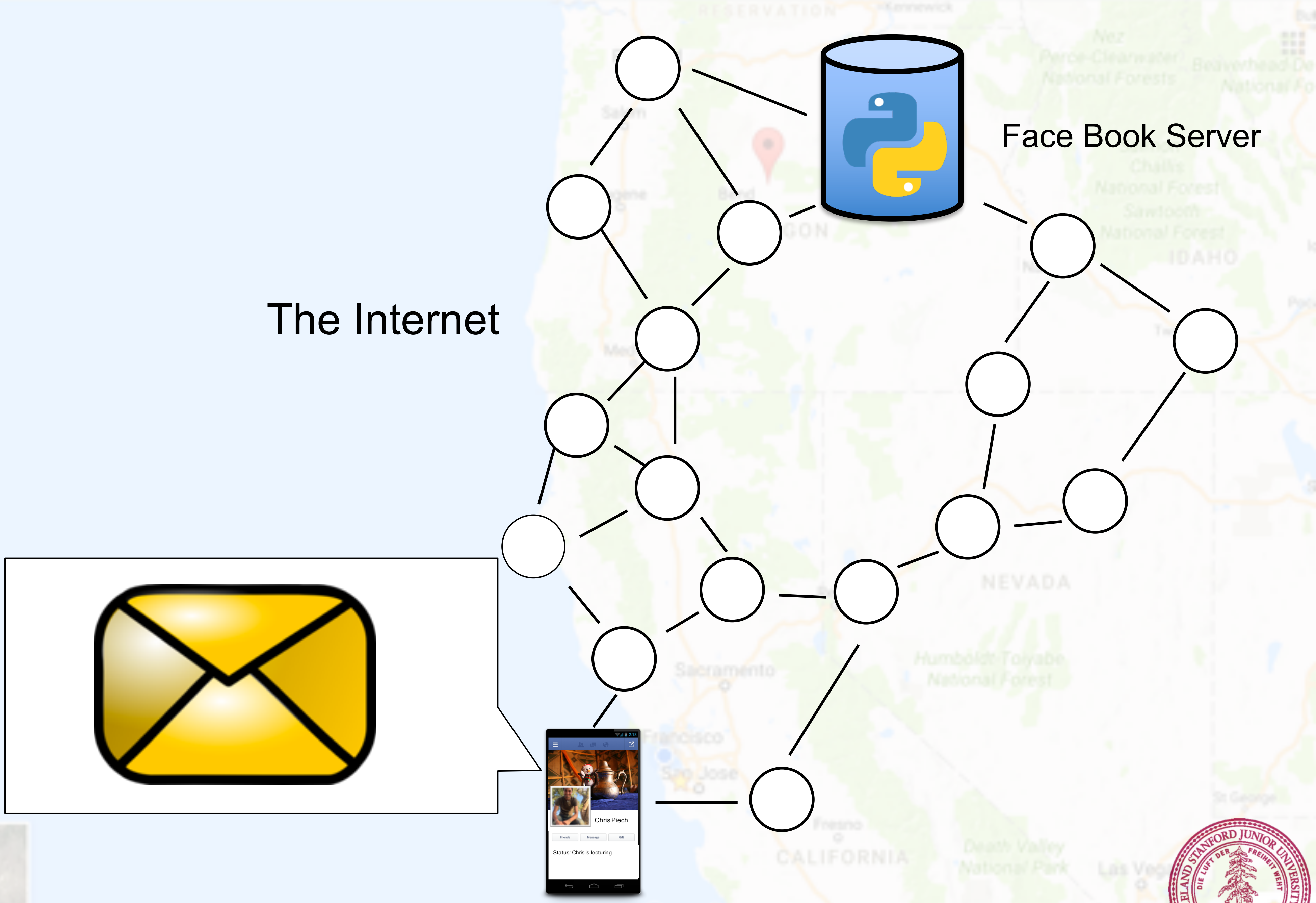
=

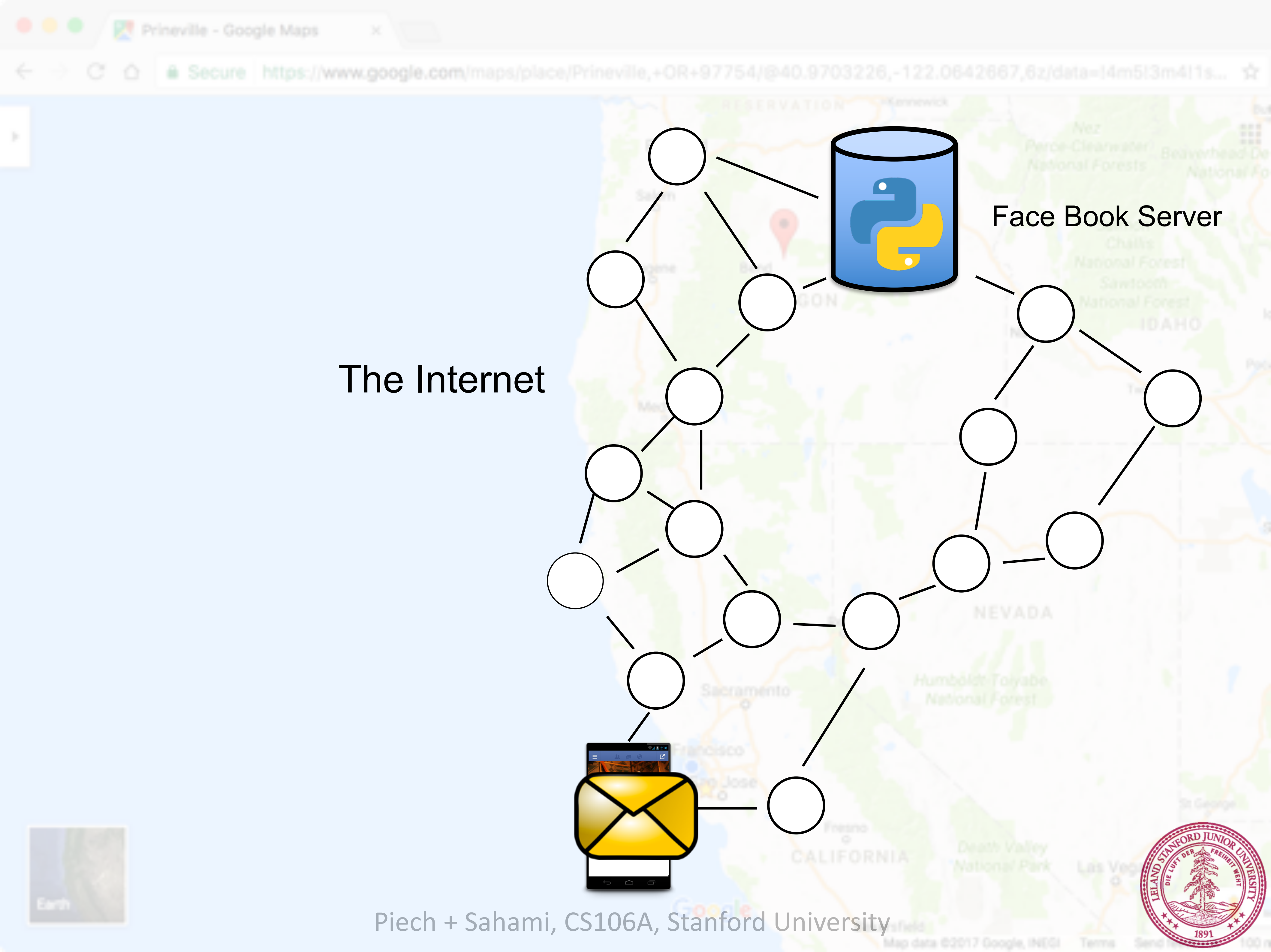

Prineville - Google Maps
Secure
https://www.google.com/maps/place/Prineville,+OR+97754/@40.9703226,-122.0642667,6z/data=!4m5!3m4!1s...
Facebook's closest datacenter is here
I am here
OREGON
IDAHO
NEVADA
CALIFORNIA
Portland
Eugene
Bend
Medford
Redding
Reno
Sacramento
San Francisco
San Jose
Fresno
Boise
Nampa
Twin Falls
Las Vegas
Bakersfield
Google
Map data ©2017 Google, INEGI
Terms
Send feedback

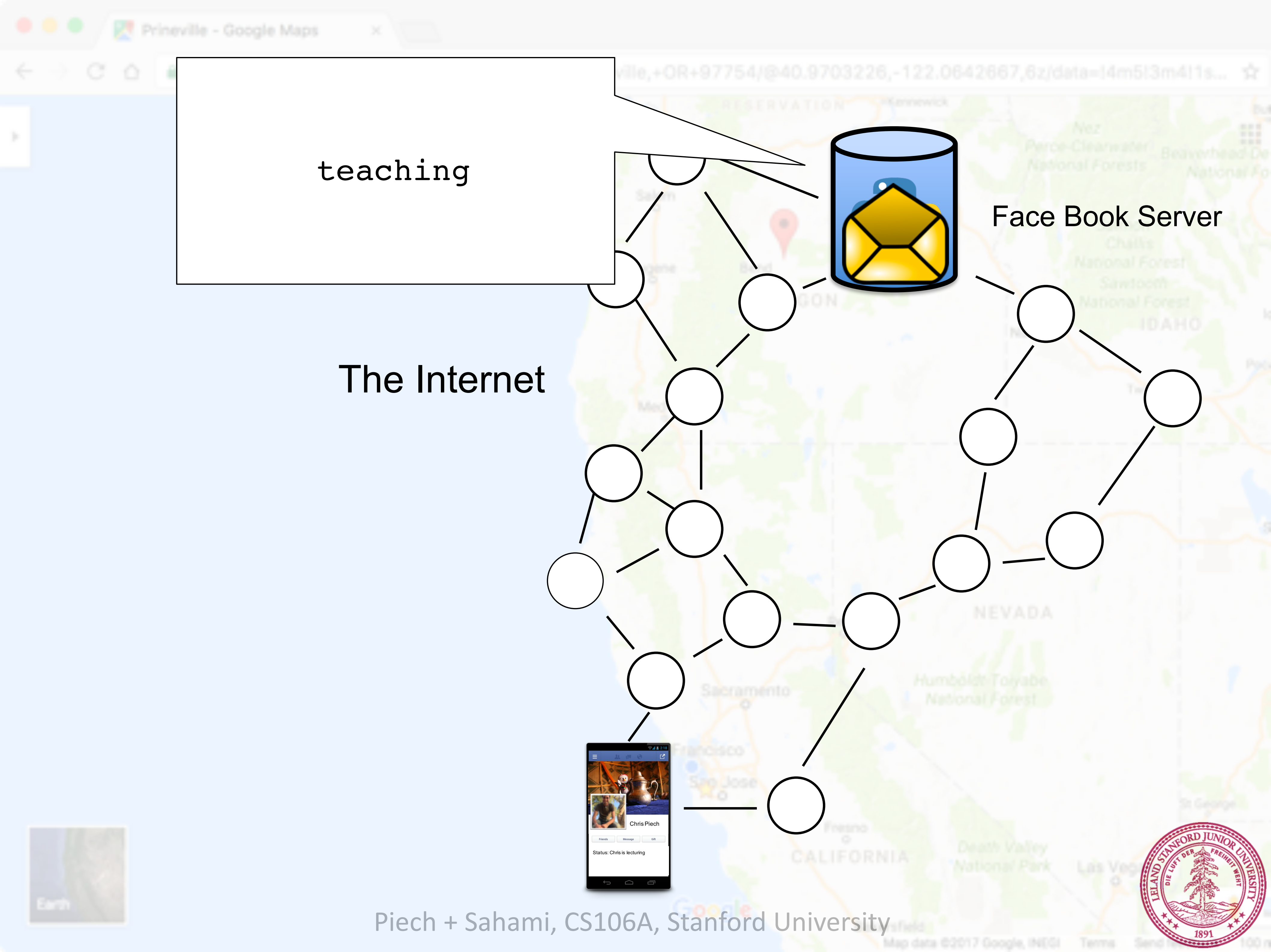

The Internet
Face Book Server
Chris Piech
Status: Chris is lecturing

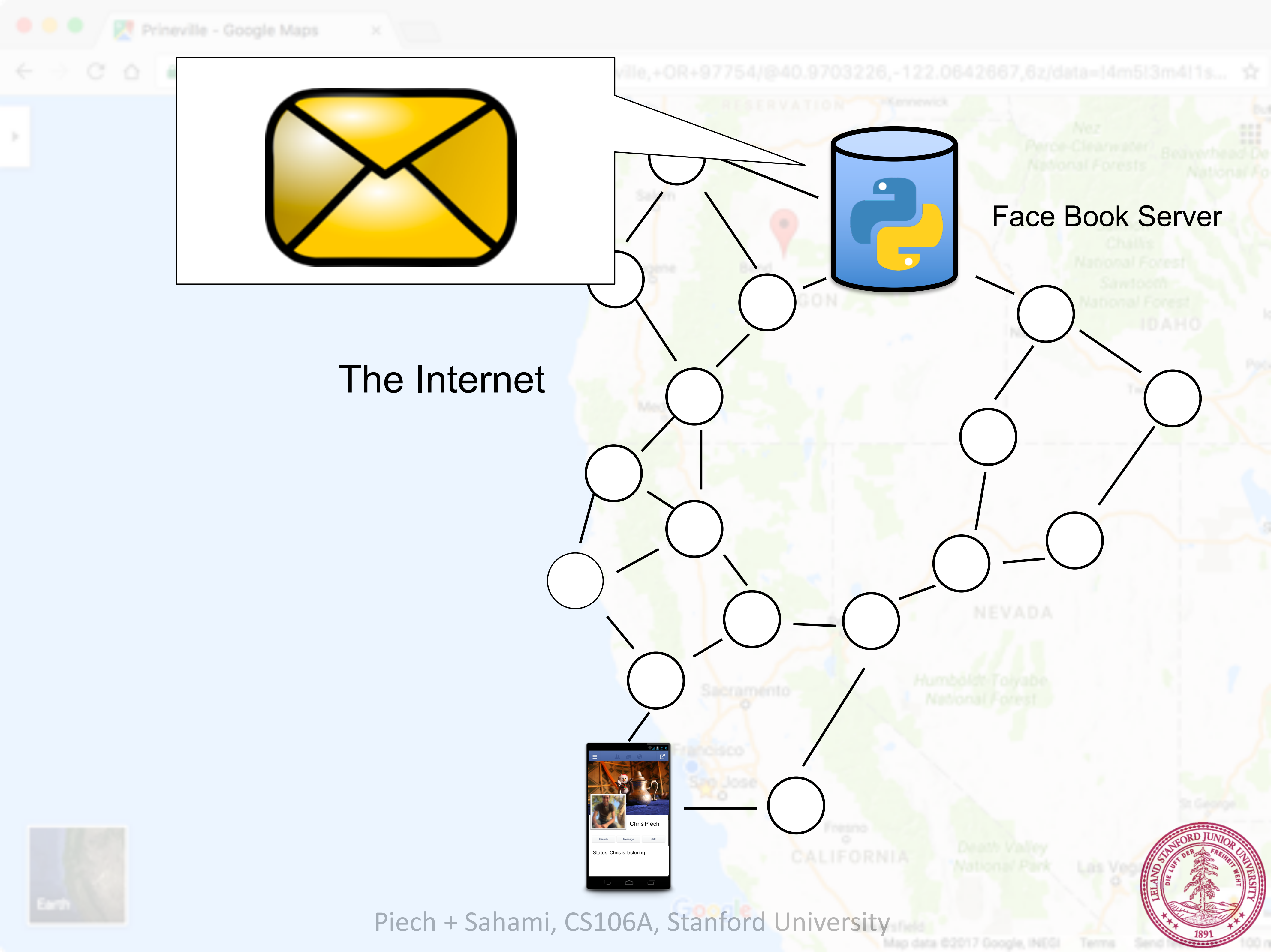


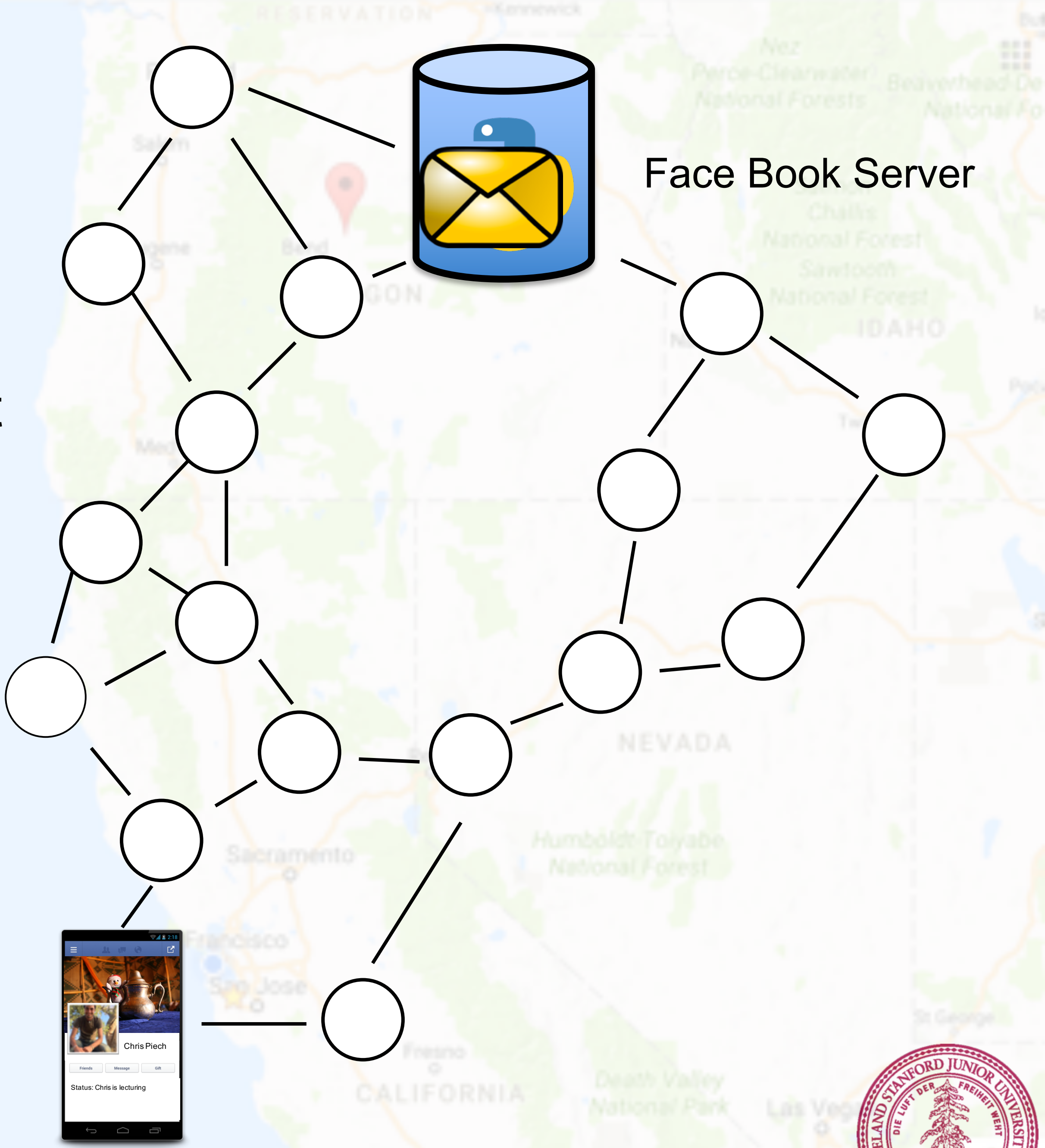
```
Get status for
piech@cs.stanford.edu
```


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


Face Book Server
The Internet

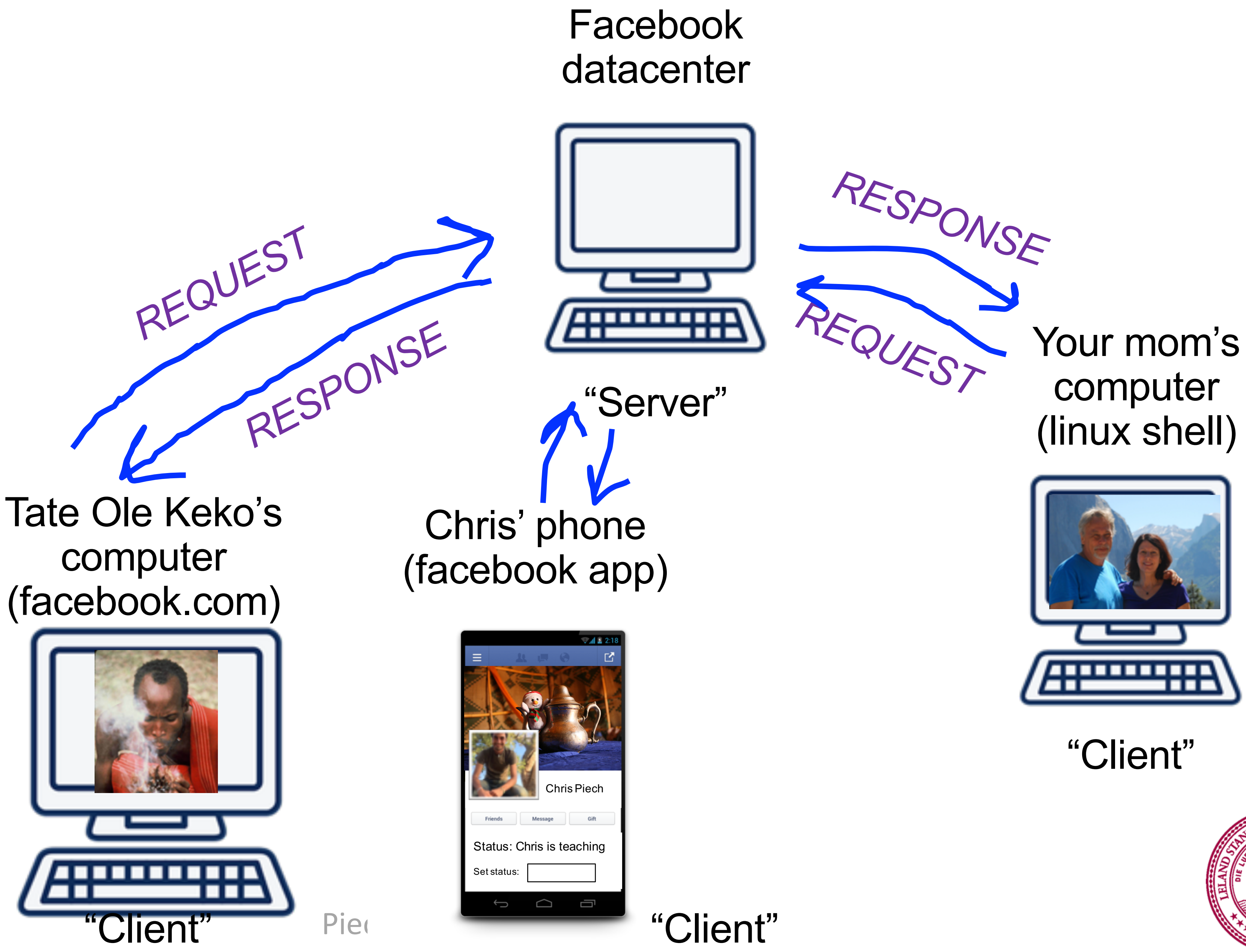

teaching
Face Book Server
The Internet
Chris Piech
Status: Chris is lecturing


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

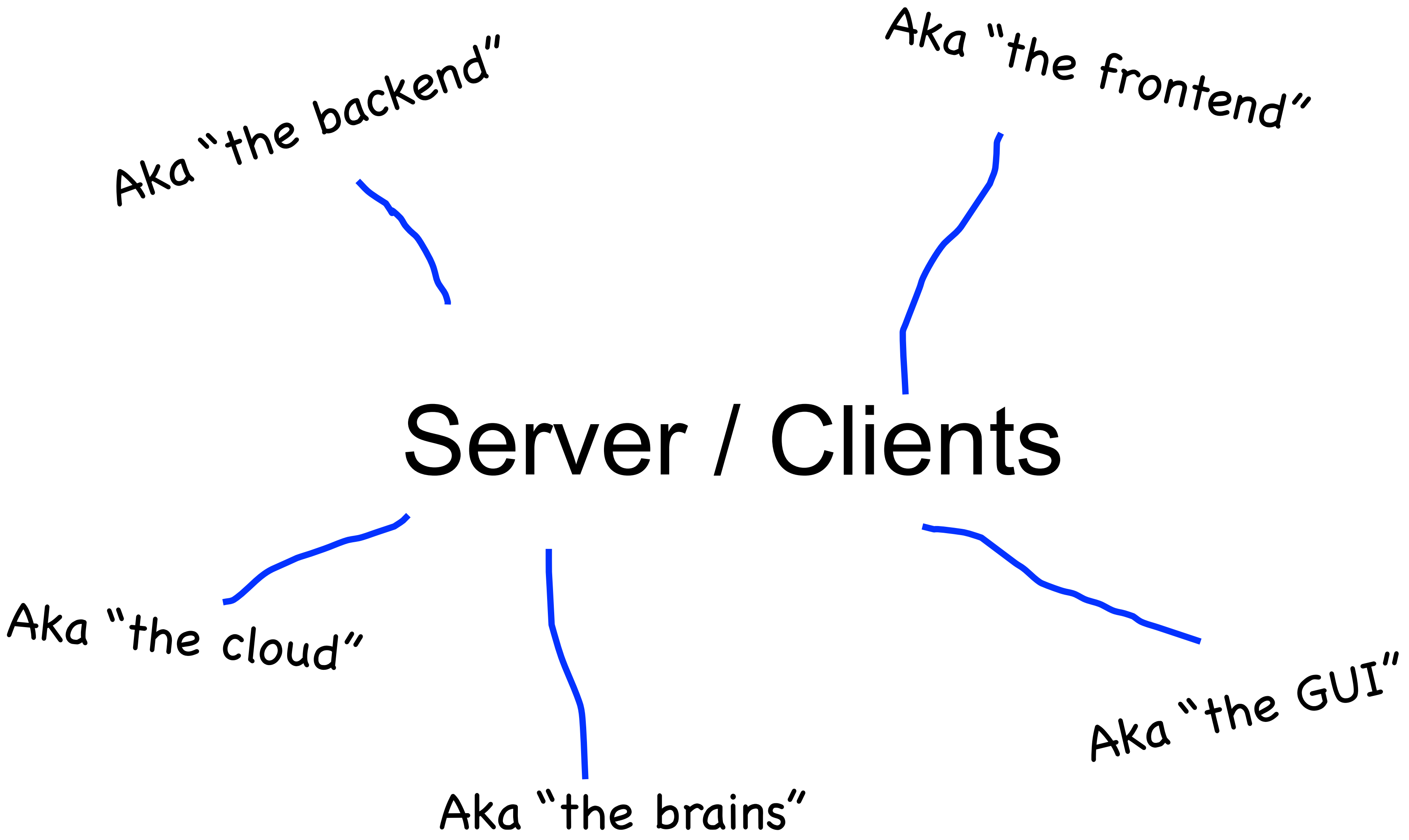

Face Book Server
The Internet
Chris Piech
Status: Chris is lecturing

Many computers can connect to the same server

The Internet


Facebook datacenter
"Server"
REQUEST
RESPONSE
RESPONSE
REQUEST
Tate Ole Keko's computer (facebook.com)
"Client"
Chris' phone (facebook app)
Chris Piech
Friends
Message
Gift
Status: Chris is teaching
Set status:
"Client"
Your mom's computer (linux shell)
"Client"

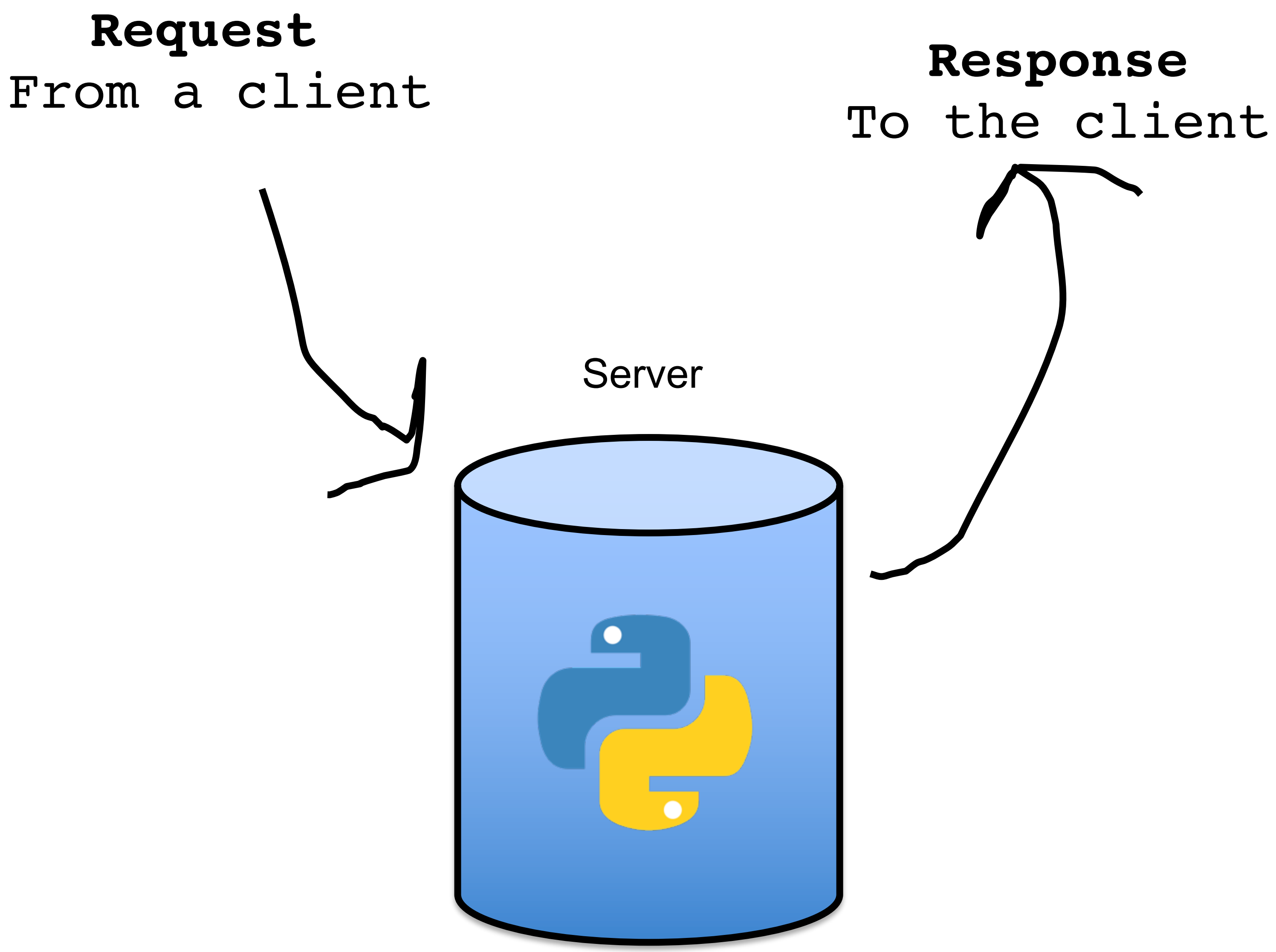
Most of the Internet



Today, the server

A server's main job is to respond to requests

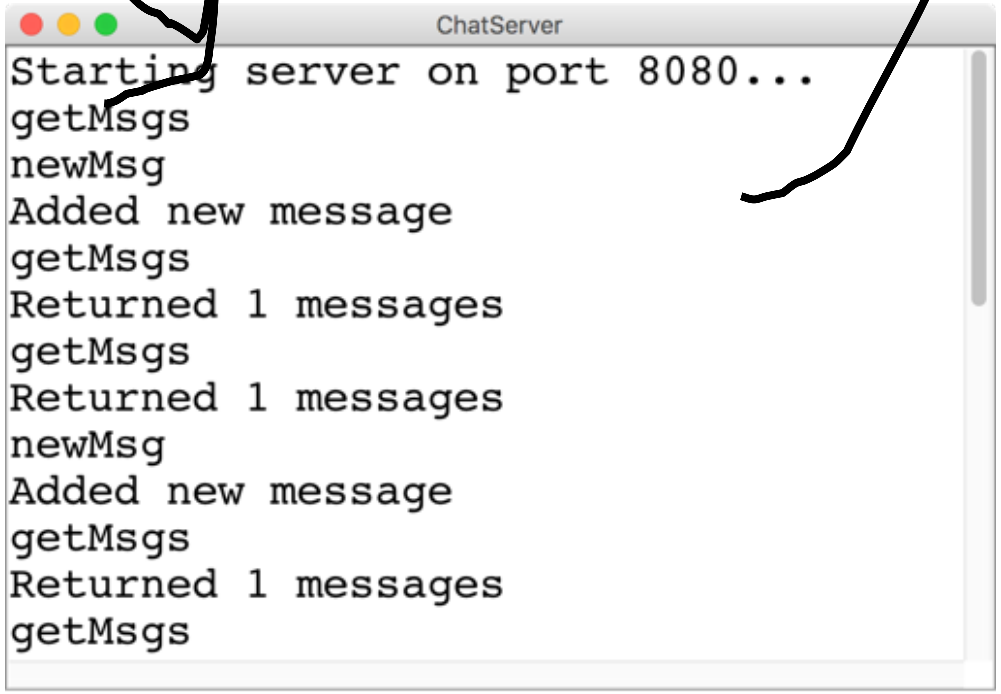
A Server's Simple Purpose



A Server’s Simple Purpose

Request
someRequest

String
serverResponse



```
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
```
class MyServer:
  # 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

```
class MyServer:
  # 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

```
class MyServer:
  # 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
```
class MyServer:
  # 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

```
class MyServer:
  # 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

```
class MyServer:
  # 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

```
class MyServer:
  # 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

```
class MyServer:
  # 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

```
class MyServer:
  # 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

```
class MyServer:
  # 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
```

What is a Port?

Servers on one slide

1

```
class MyServer:
  # 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

```
class MyServer:
  # 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

```
class MyServer:
  # 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
```

What is a Request?

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

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

```
// instance methods of a request
request.get_command()
request.get_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/')
```

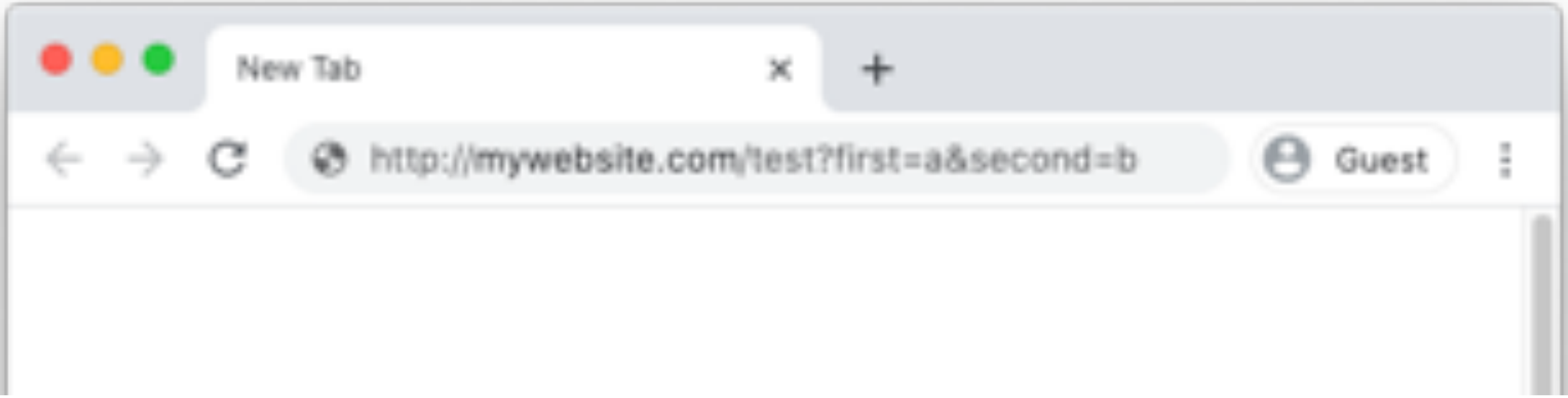
Who makes requests?

Other programs can send requests!

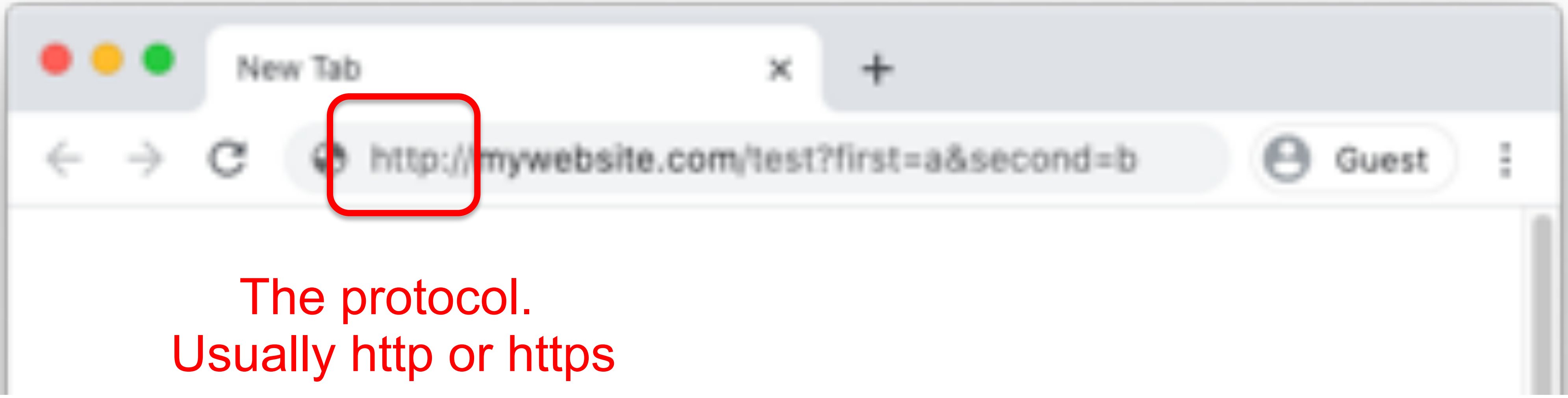
```
response = requests.get('https://xkcd.com/353/')
```

Web browsers can send requests!

Anatomy of a Browser Request

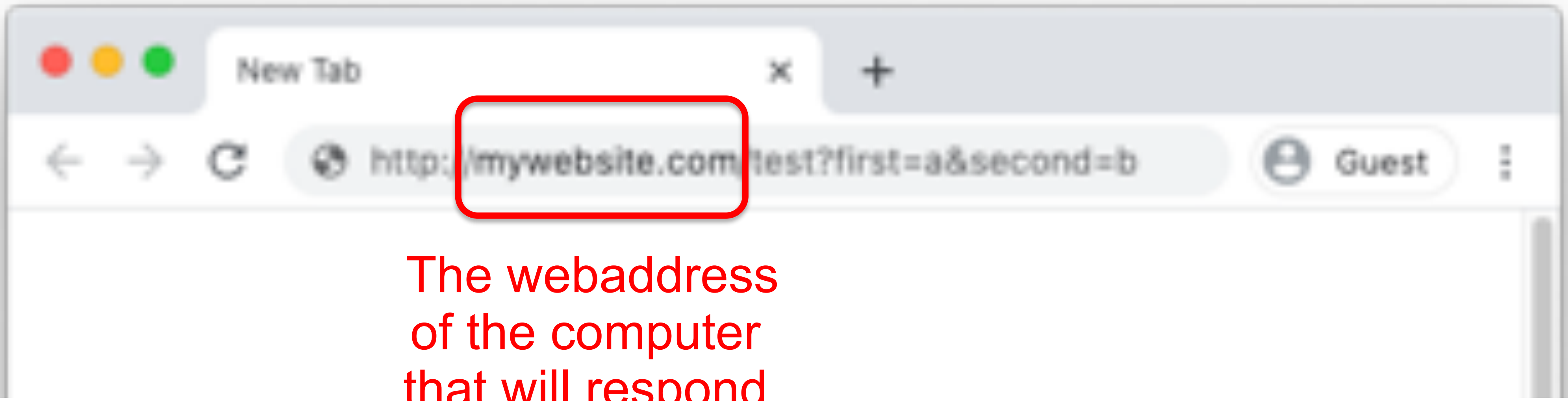


Anatomy of a Browser Request



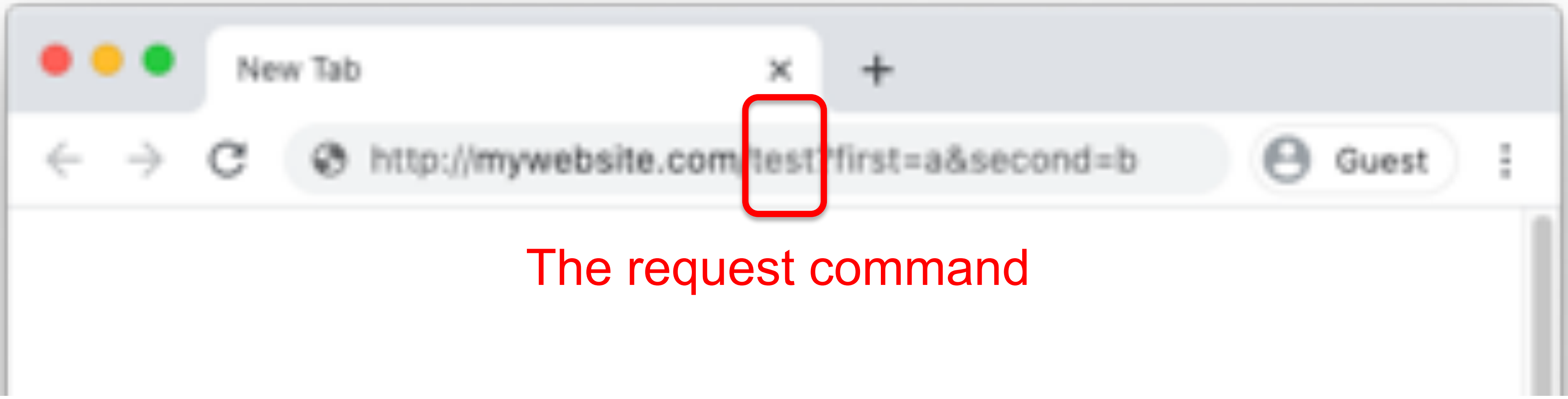
The protocol.
Usually http or https

Anatomy of a Browser Request

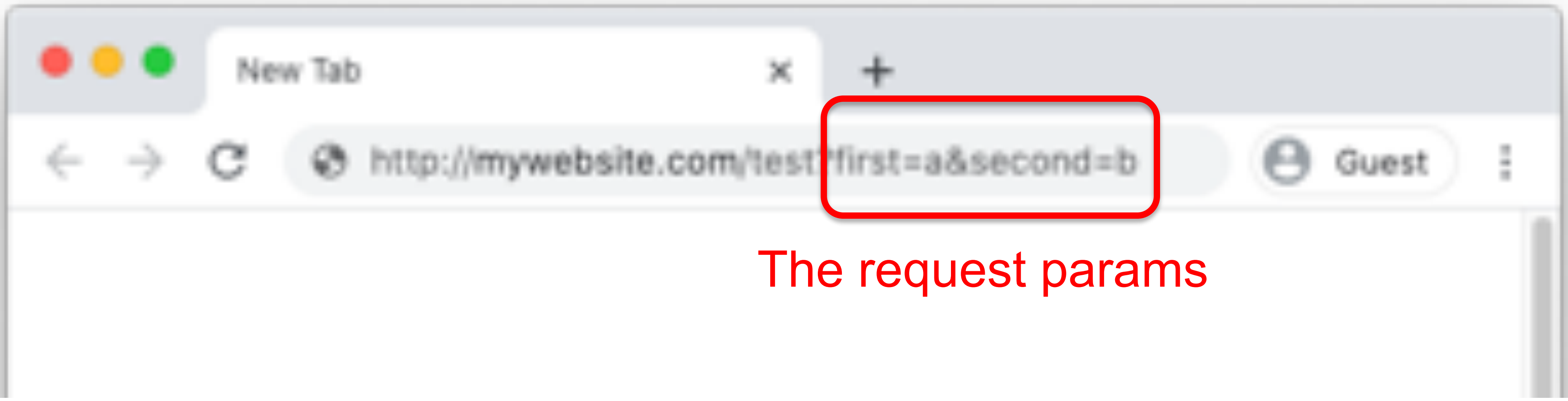


The webaddress of the computer that will respond to the request

Anatomy of a Browser Request



Anatomy of a Browser Request



The request 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)
```

Hit Counter

Recall Requests

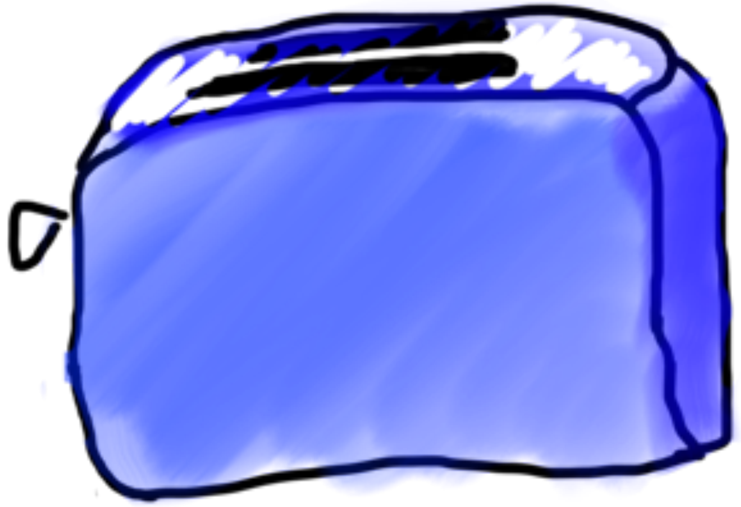
```
/* Request has a command */
command (string)

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

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

Requests are like Remote Method Calls

Server has a bunch of discrete things it can do

Server

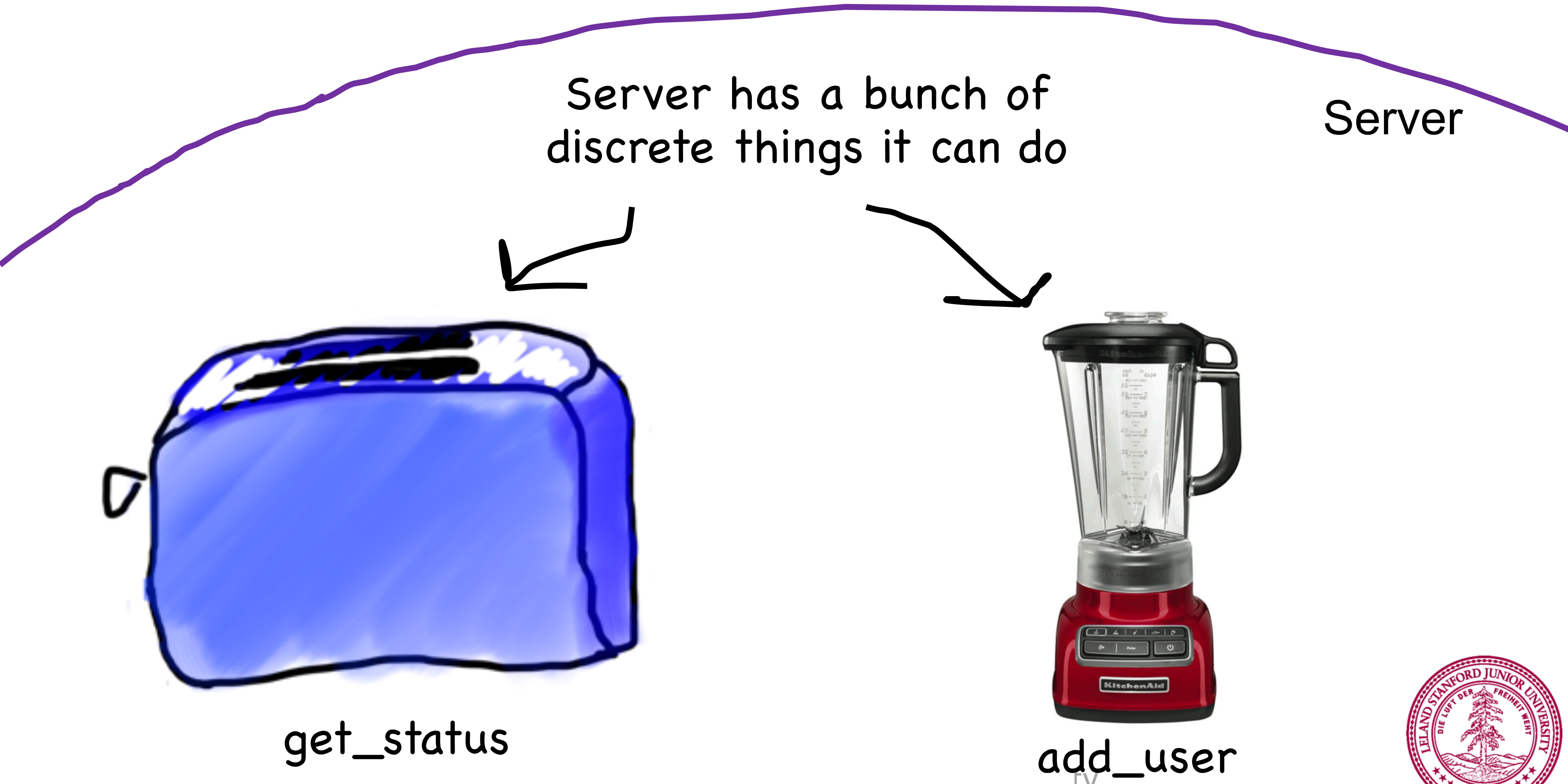


make_toast

blend

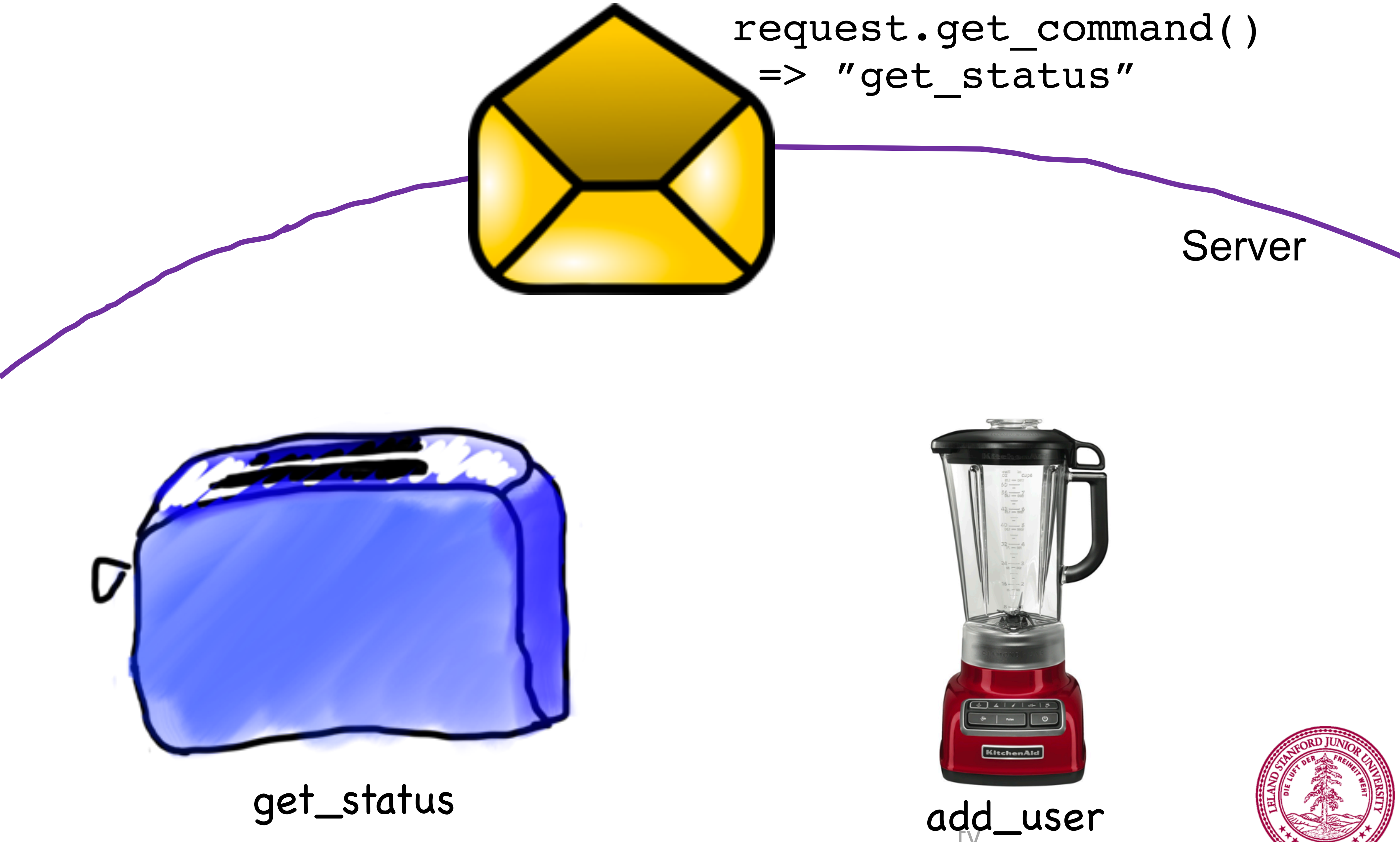
ty

Requests are like Remote Method Calls



Requests are like Remote Method Calls

Requests are like Remote Method Calls



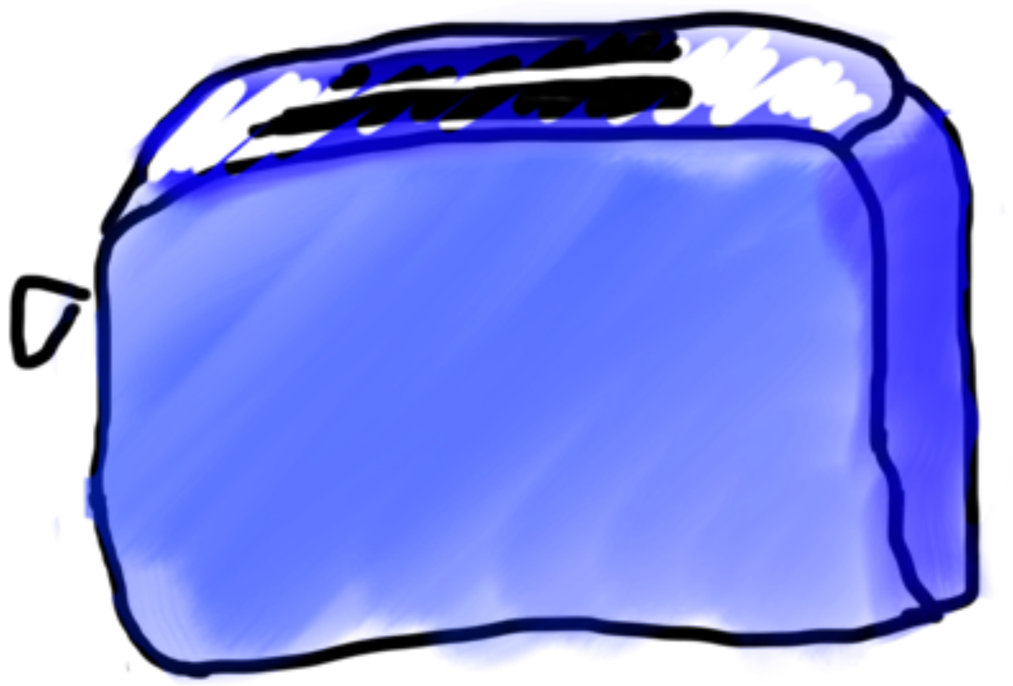
```
request.get_command()
 => "get_status"
```

Requests are like Remote Method Calls

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

ty

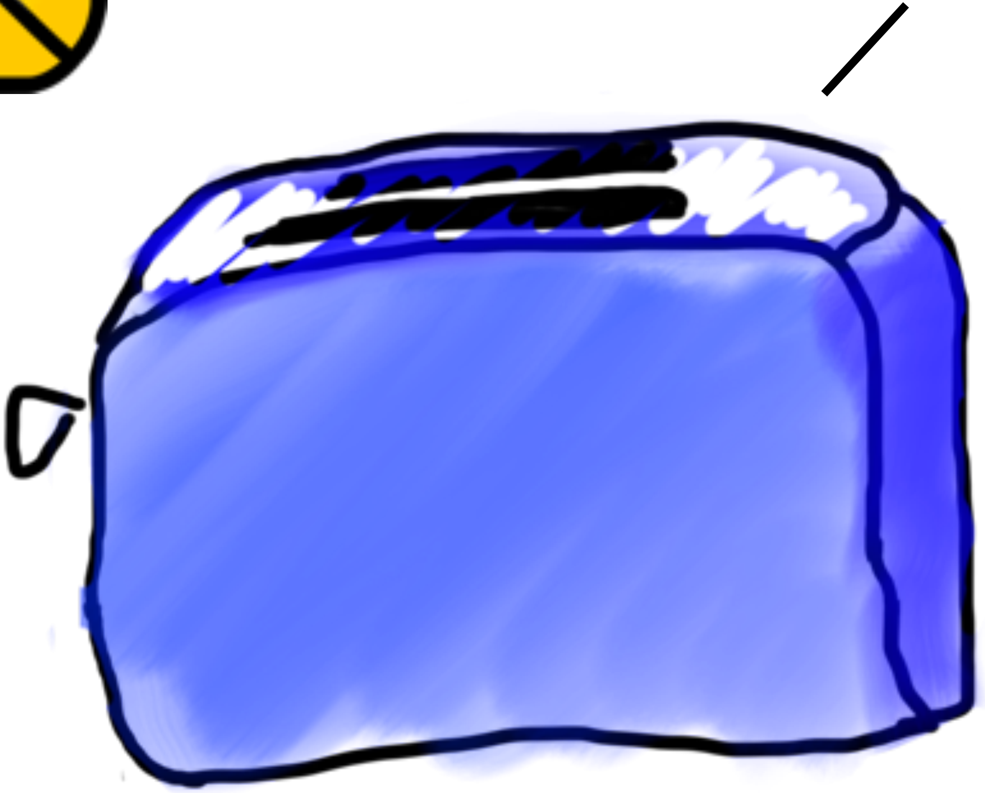
Requests are like Remote Method Calls



ty

Requests are like Remote Method Calls

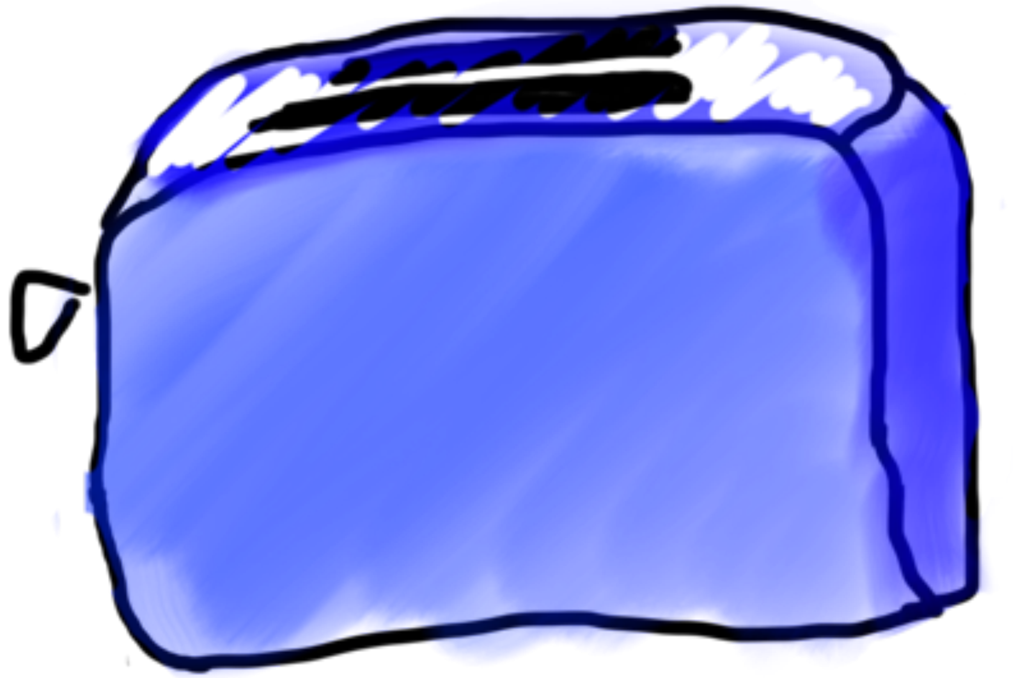
```
request.params[“userName”]
```



get_status

ty

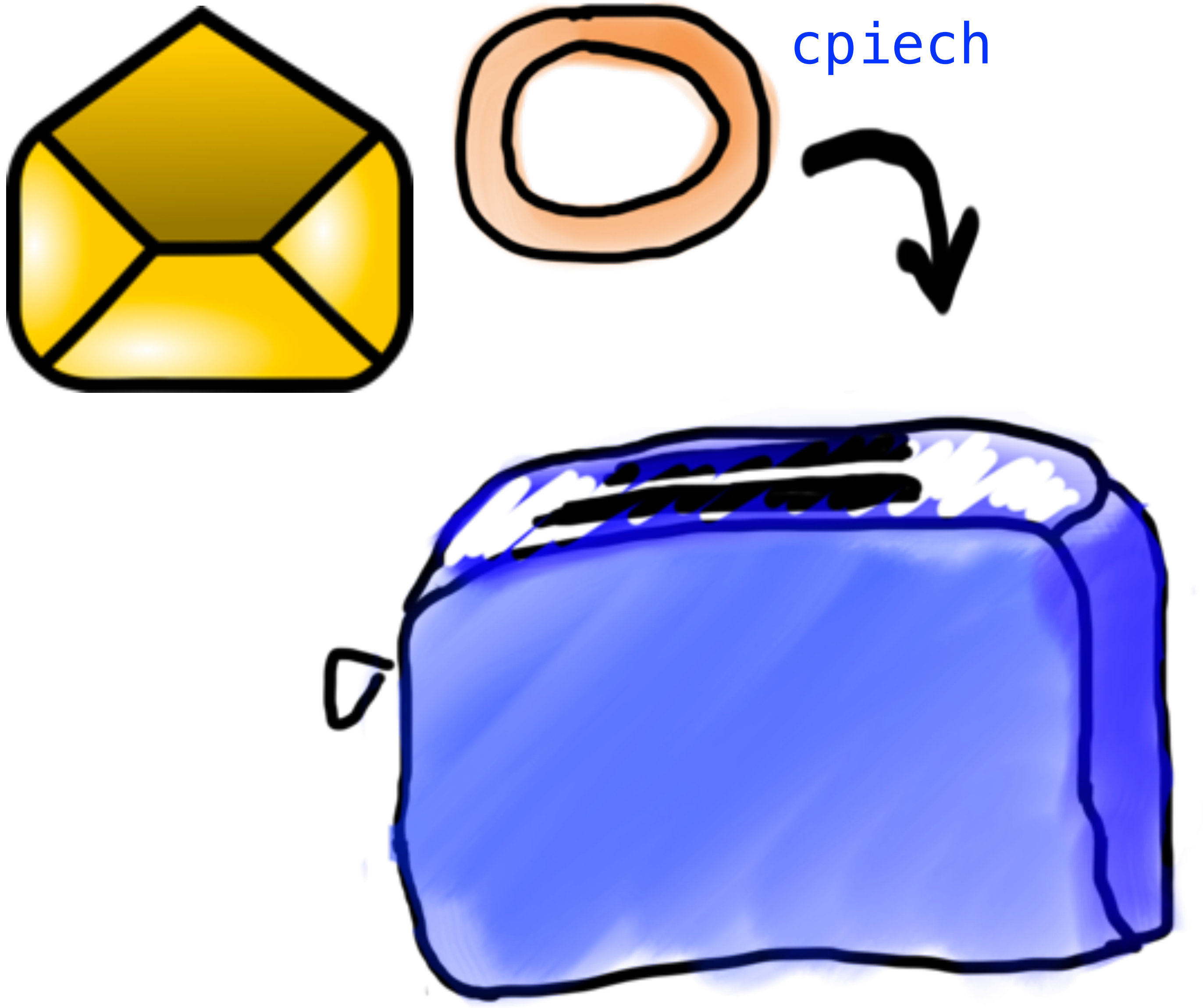
Requests are like Remote Method Calls



get_status

ty

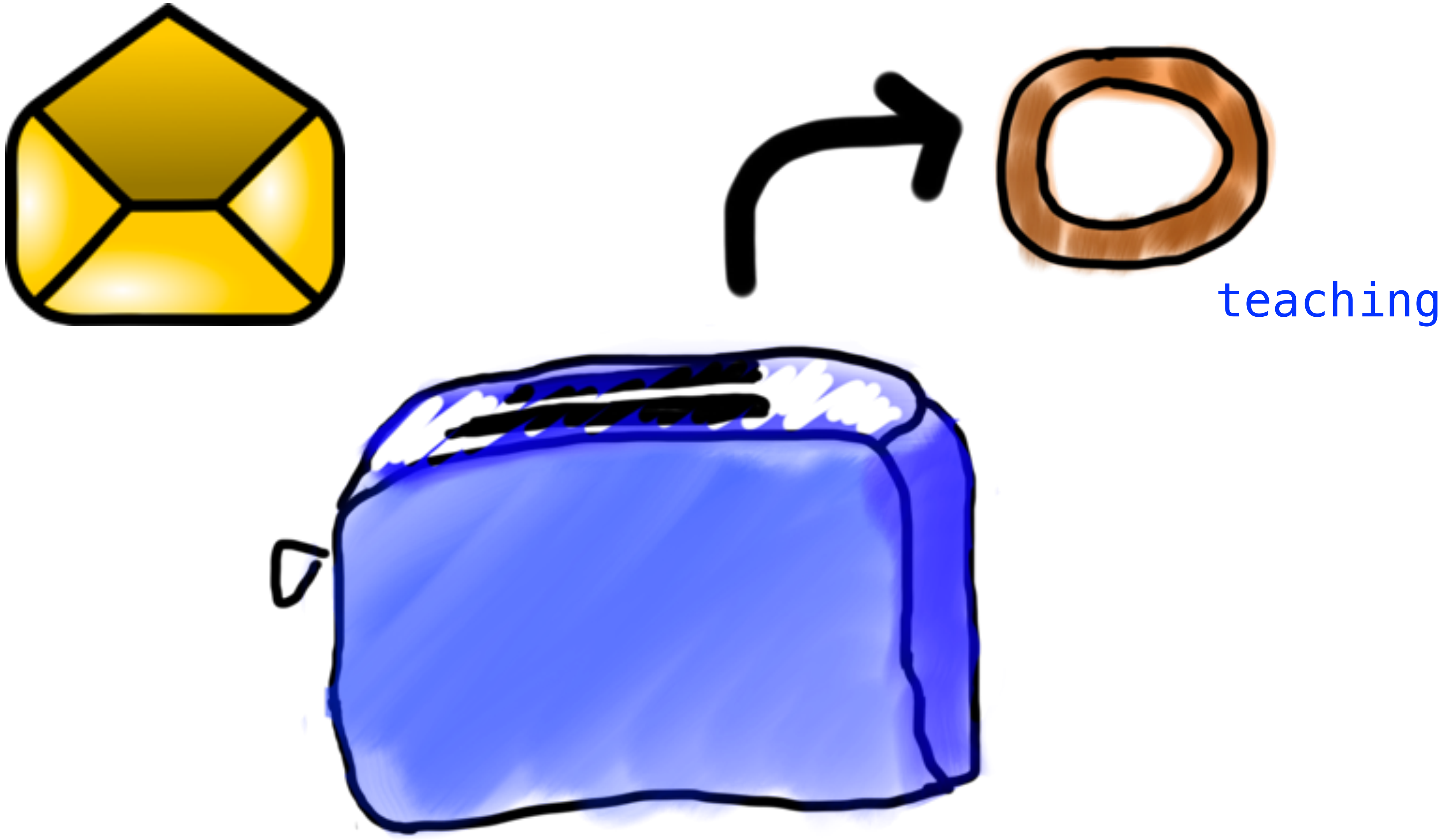
Requests are like Remote Method Calls



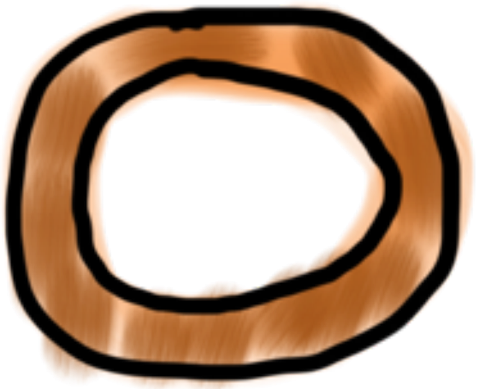
get_status

ty

Requests are like Remote Method Calls

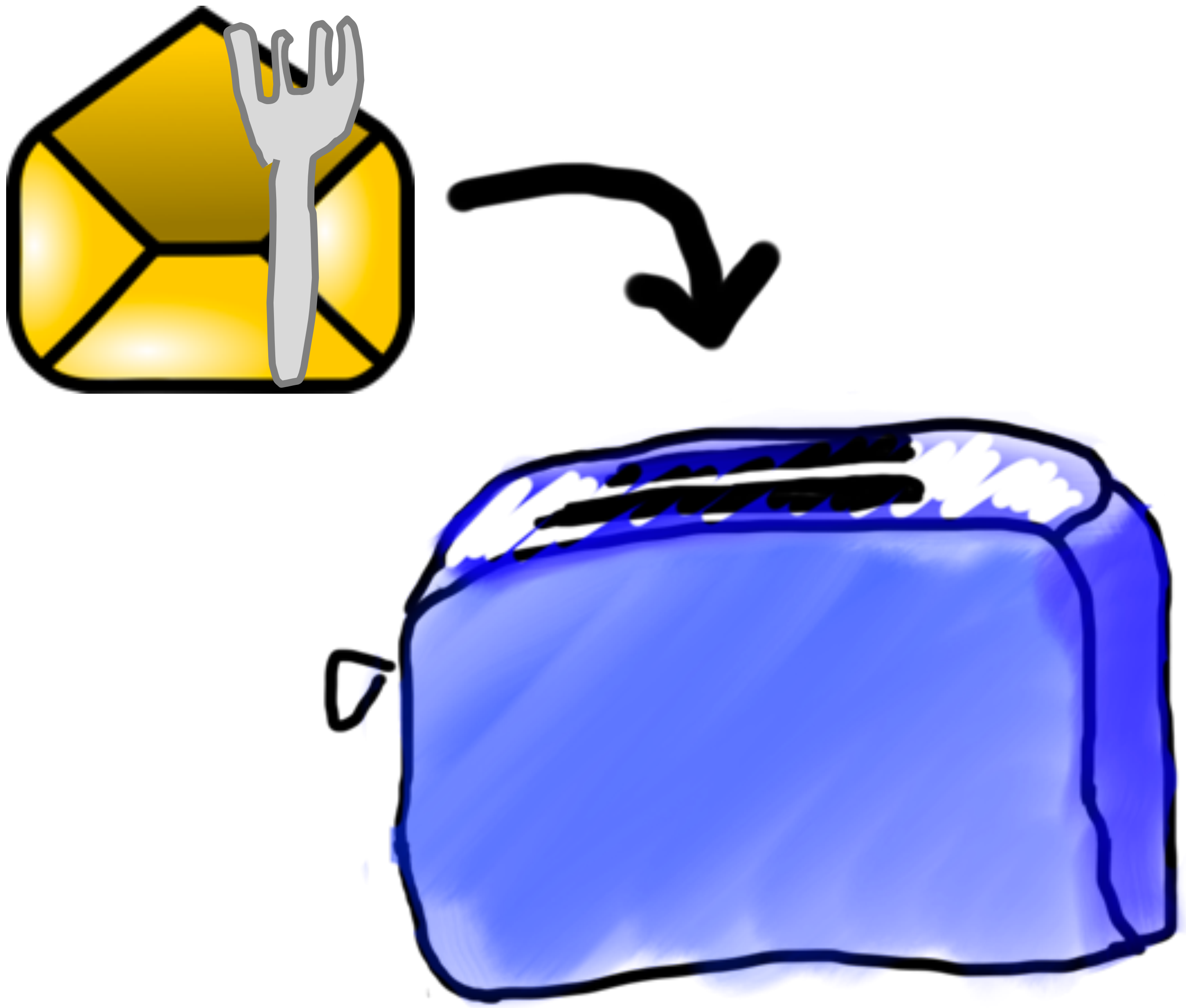


```
def handle_request(self, request):
    cmd = request.command
    if cmd == 'get_status':
        user = request.params['userName']
        status = self.get_status(user)
        return status
```



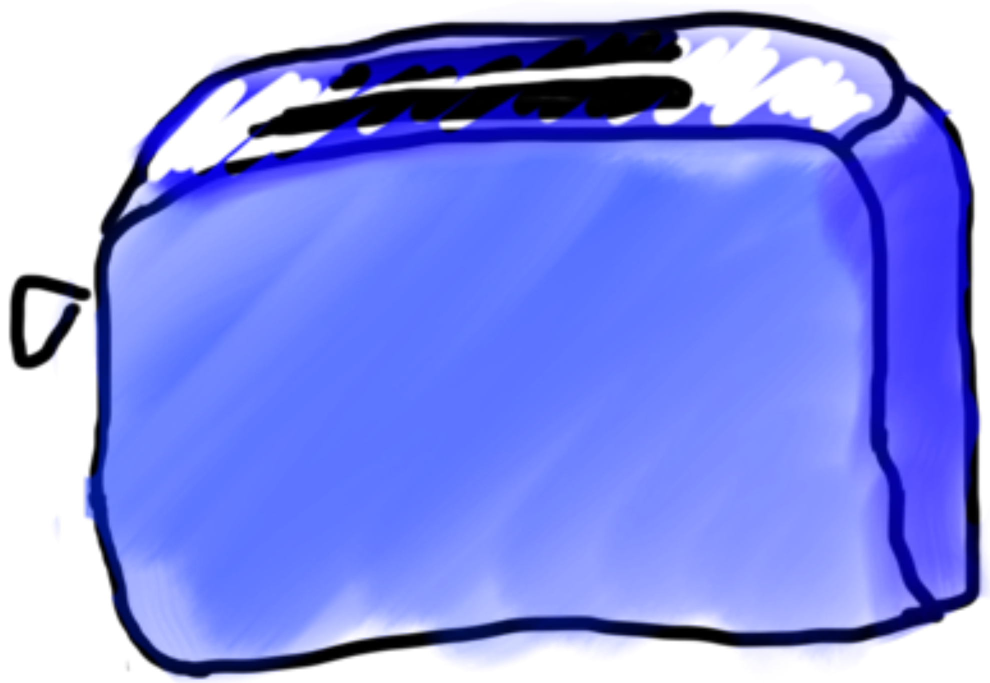
Must be a string!

Requests are like Remote Method Calls

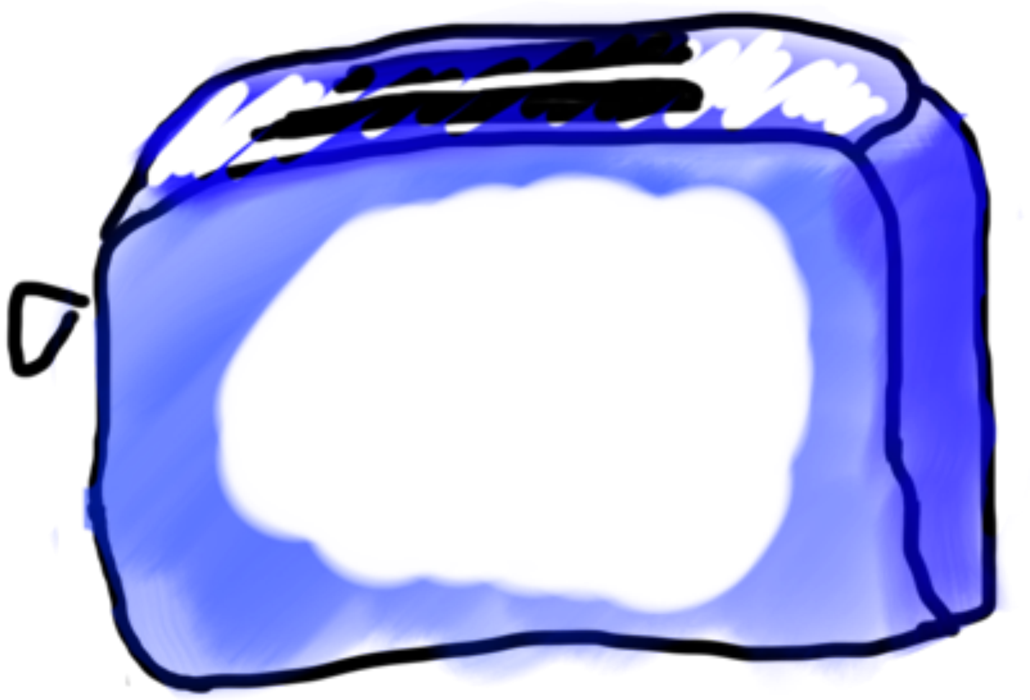


Requests are like Remote Method Calls

Requests are like Remote Method Calls

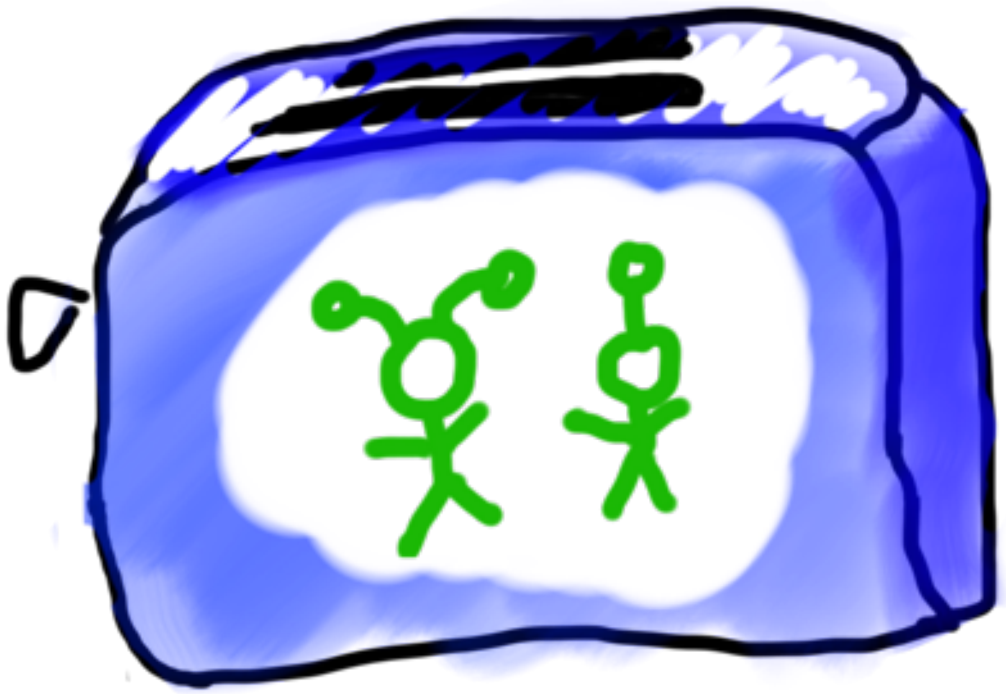


Requests are like Remote Method Calls



Requests are like Remote Method Calls

Requests are like Remote Method Calls



Internet sends data as strings…

How do you send a list or a dictionary?

Requests responses are strings, often encoded using JSON

Recall JSON

ages.json

```
{
    "Chris":32,
    "Gary":70,
    "Mehran":50,
    "Juliette":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,
    "Juliette":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

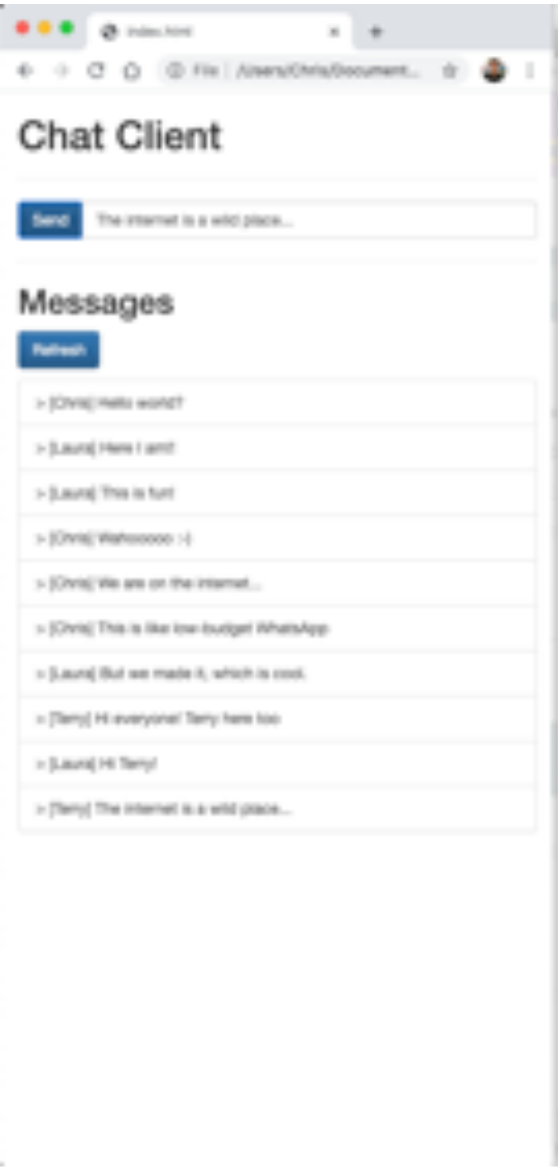
ages.json

```
{
    "Chris":32,
    "Gary":70,
    "Mehran":50,
    "Juliette":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

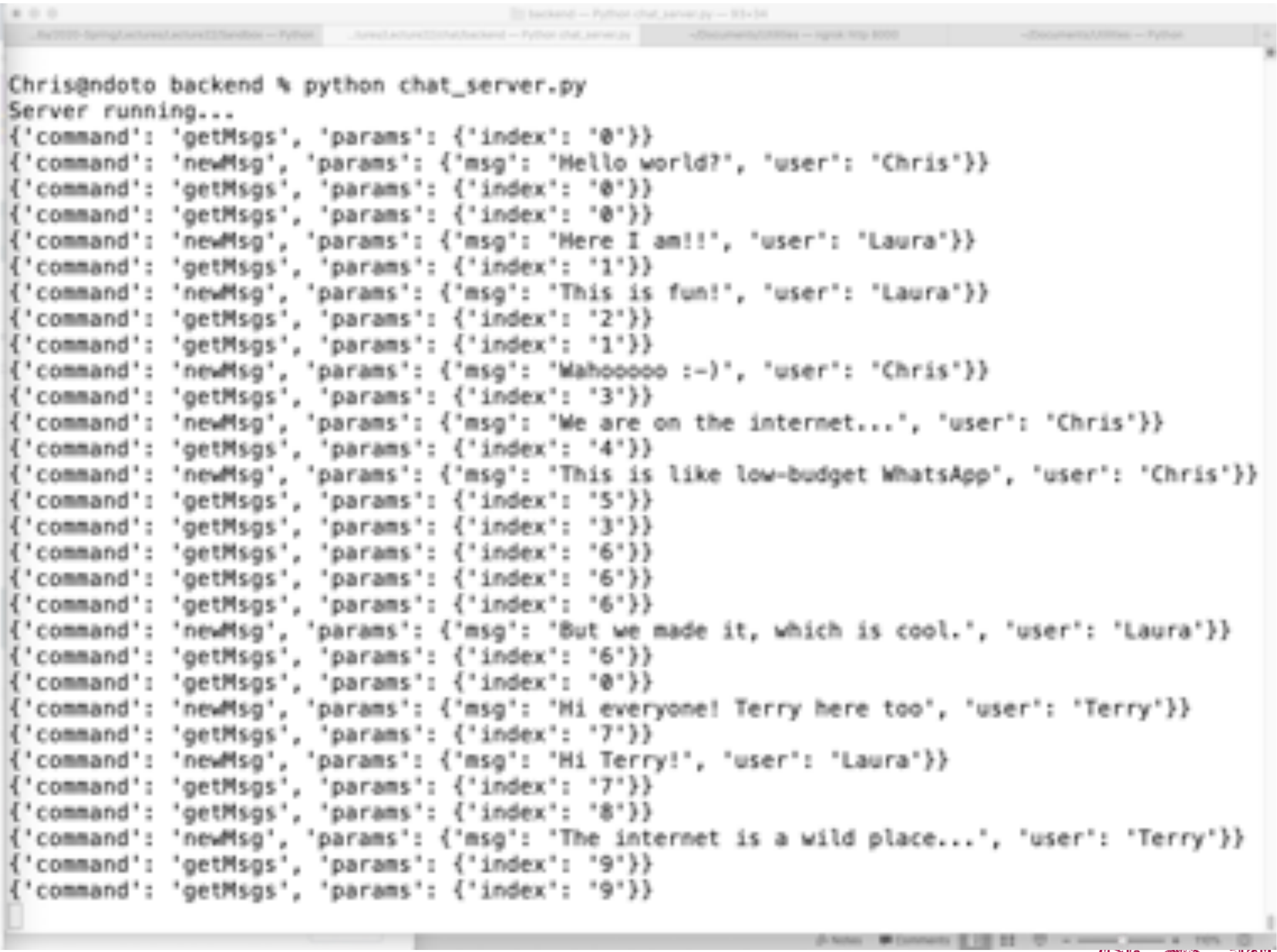


Chat Client

The internet is a wild place...

Messages

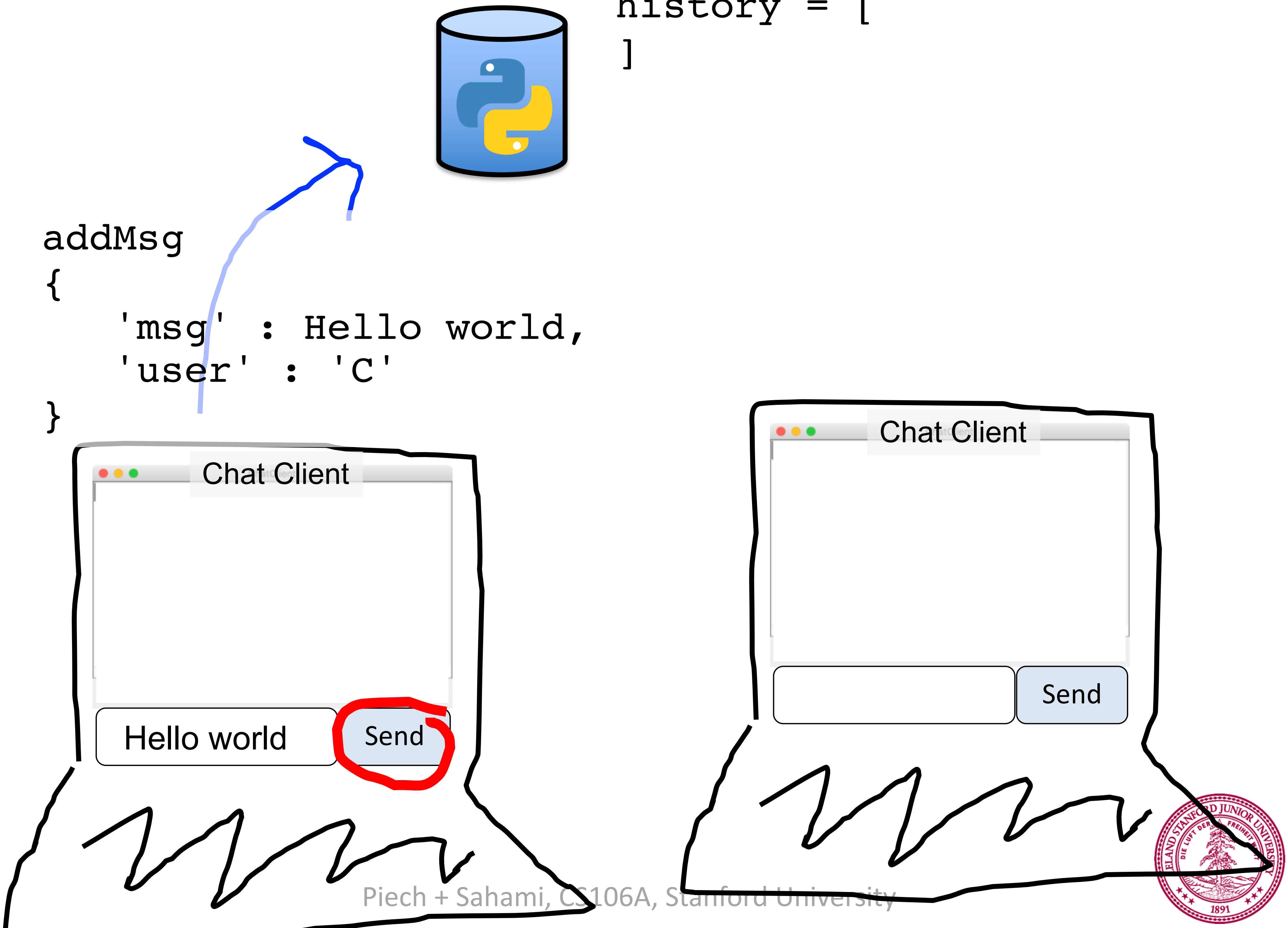
- [Chris] Hello world?
- [Laura] Here I am!!
- [Laura] This is fun!
- [Chris] Wahoooooo :-)
- [Chris] We are on the internet...
- [Chris] This is like low-budget WhatsApp
- [Laura] But we made it, which is cool.
- [Terry] Hi everyone! Terry here too
- [Laura] Hi Terry!
- [Terry] The internet is a wild place...

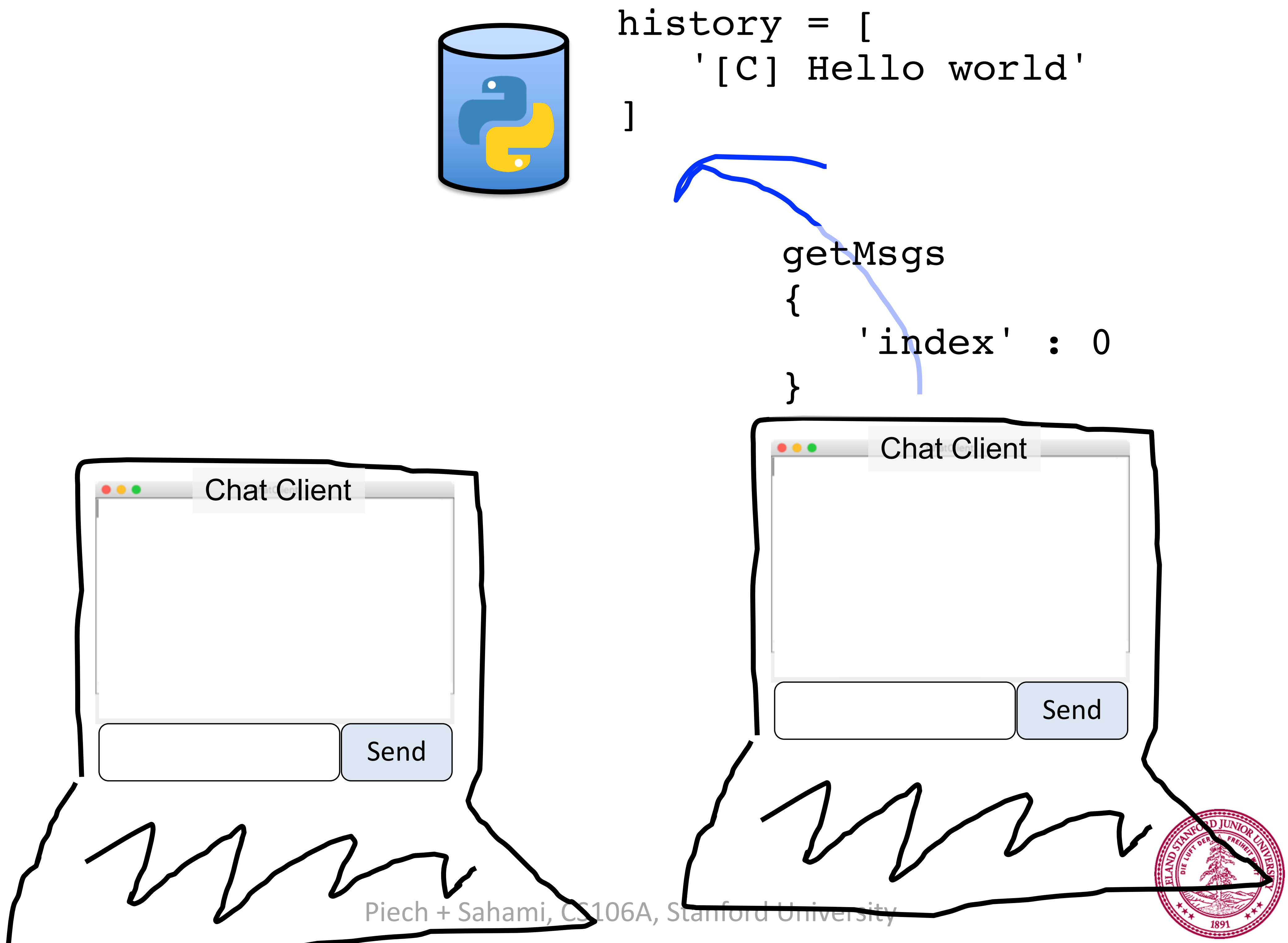


```
Chris@ndoto backend % python chat_server.py
Server running...
{'command': 'getMsgs', 'params': {'index': '0'}}
{'command': 'newMsg', 'params': {'msg': 'Hello world?', 'user': 'Chris'}}
{'command': 'getMsgs', 'params': {'index': '0'}}
{'command': 'getMsgs', 'params': {'index': '0'}}
{'command': 'newMsg', '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': 'Wahoooooo :-)', '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'
}
```

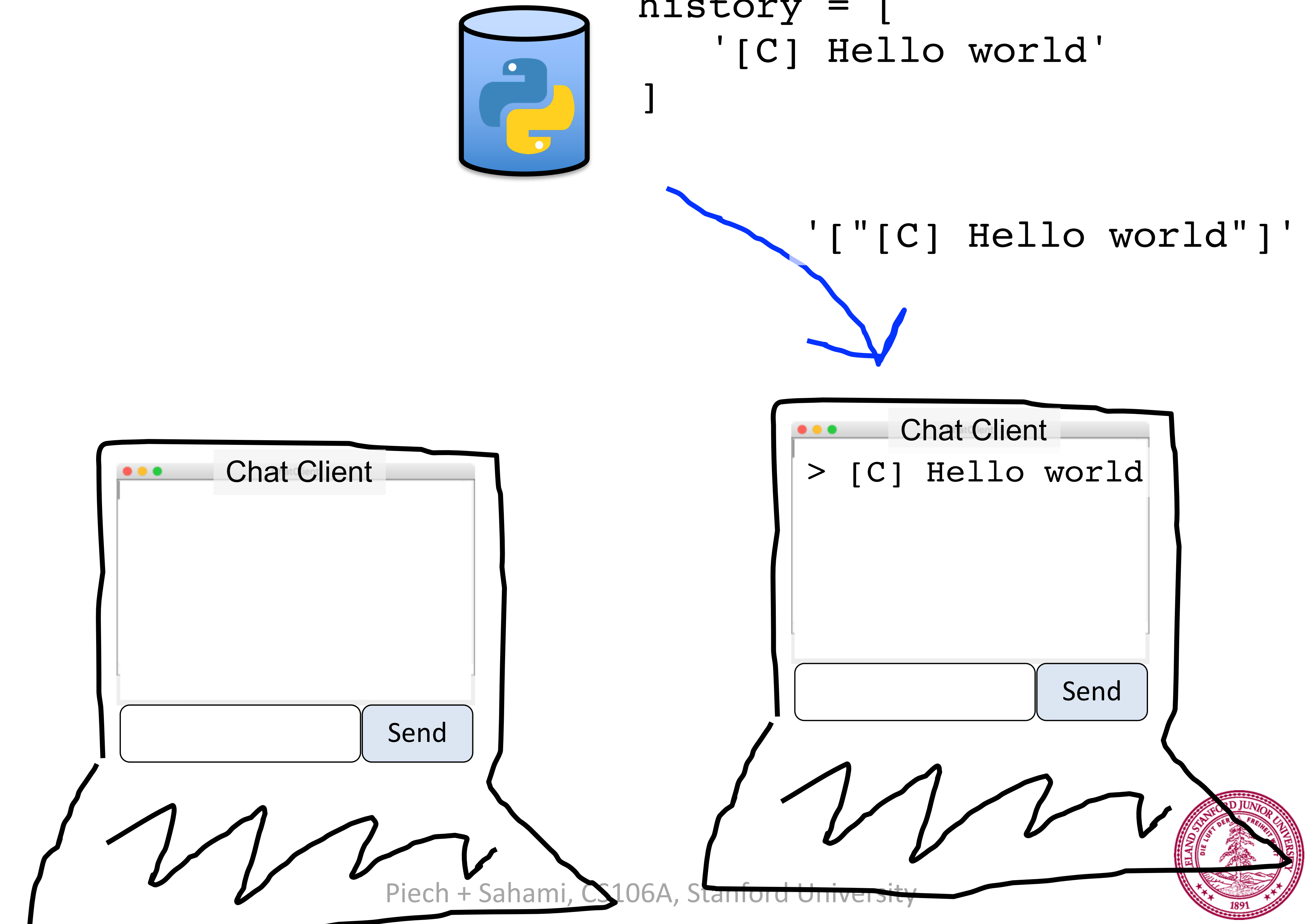




```
history = [
    '[C] Hello world'
]
```

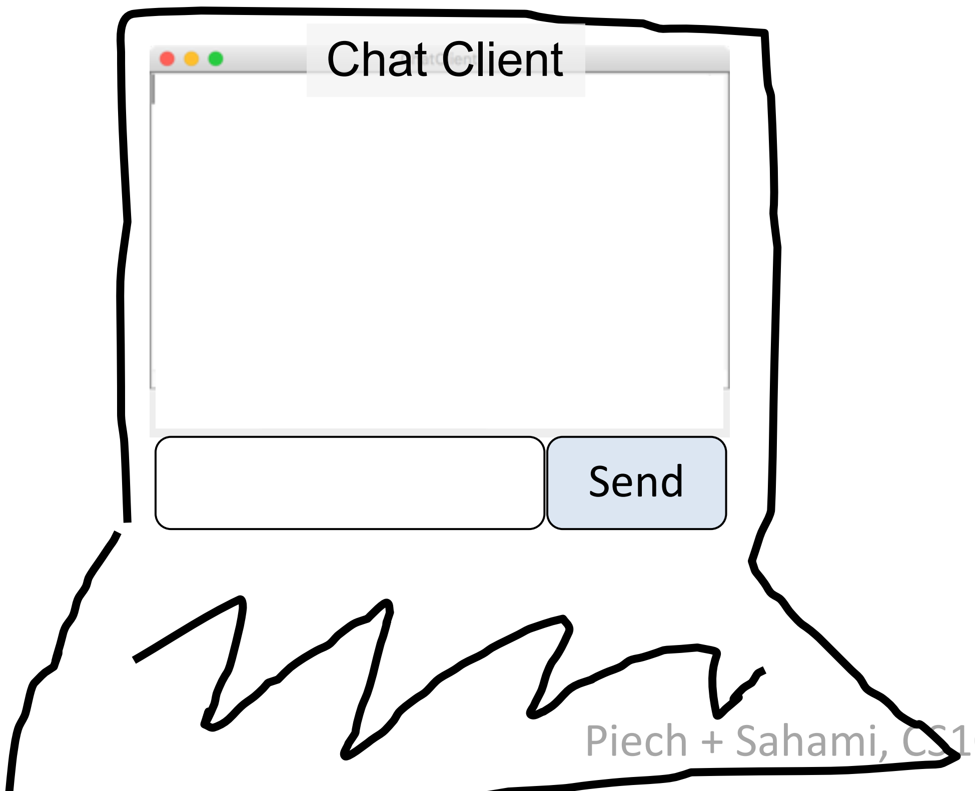
```
getMsgs
{
    'index' : 0
}
```

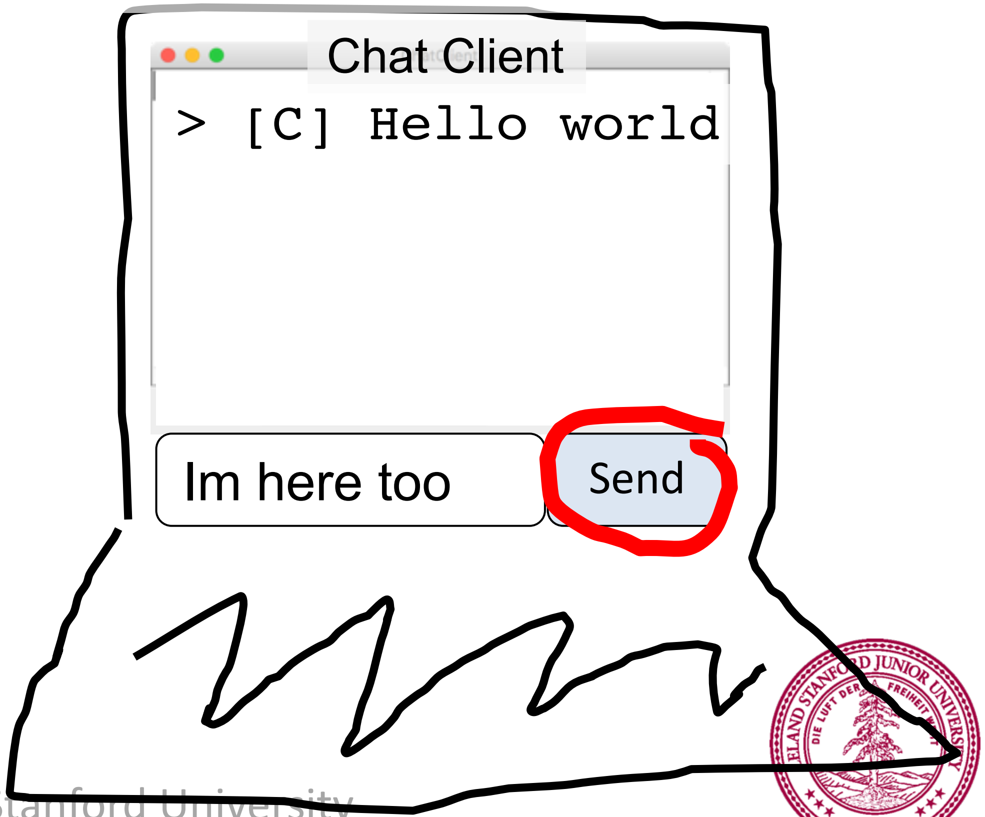
```
history = [
    '[C] Hello world'
]
```

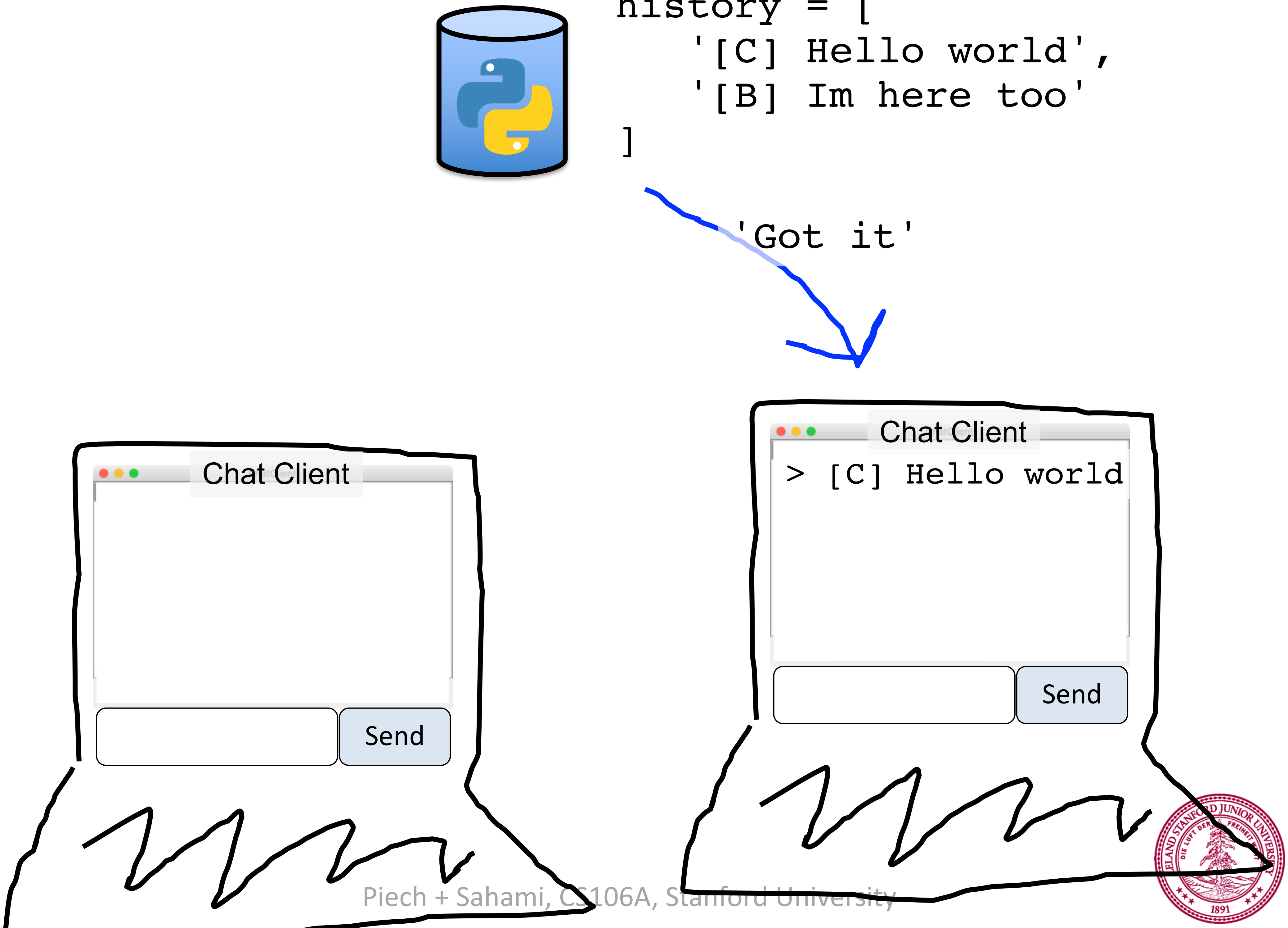


```
history = [
    '[C] Hello world'
]
```

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

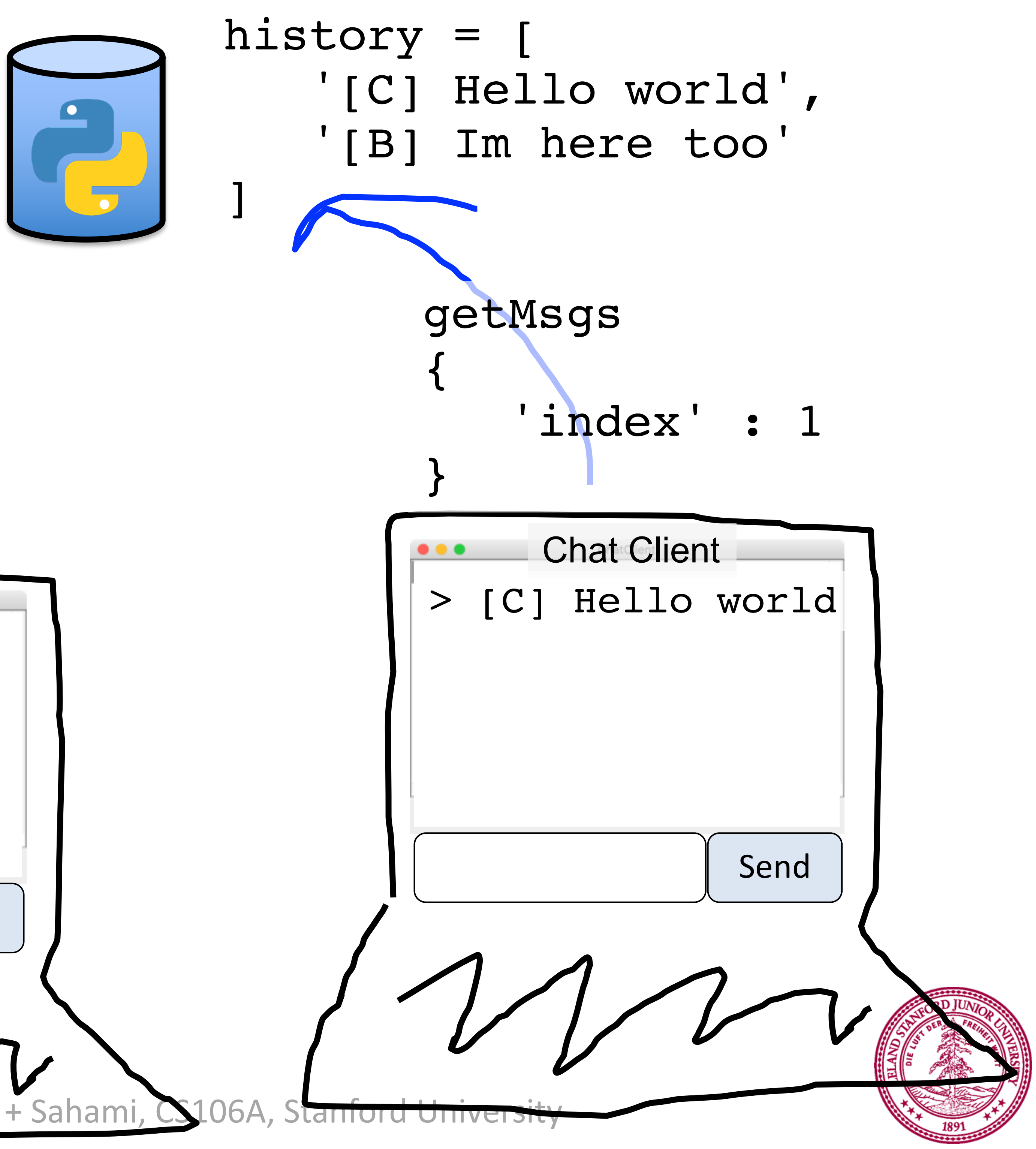






```
history = [
    '[C] Hello world',
    '[B] Im here too'
]
```

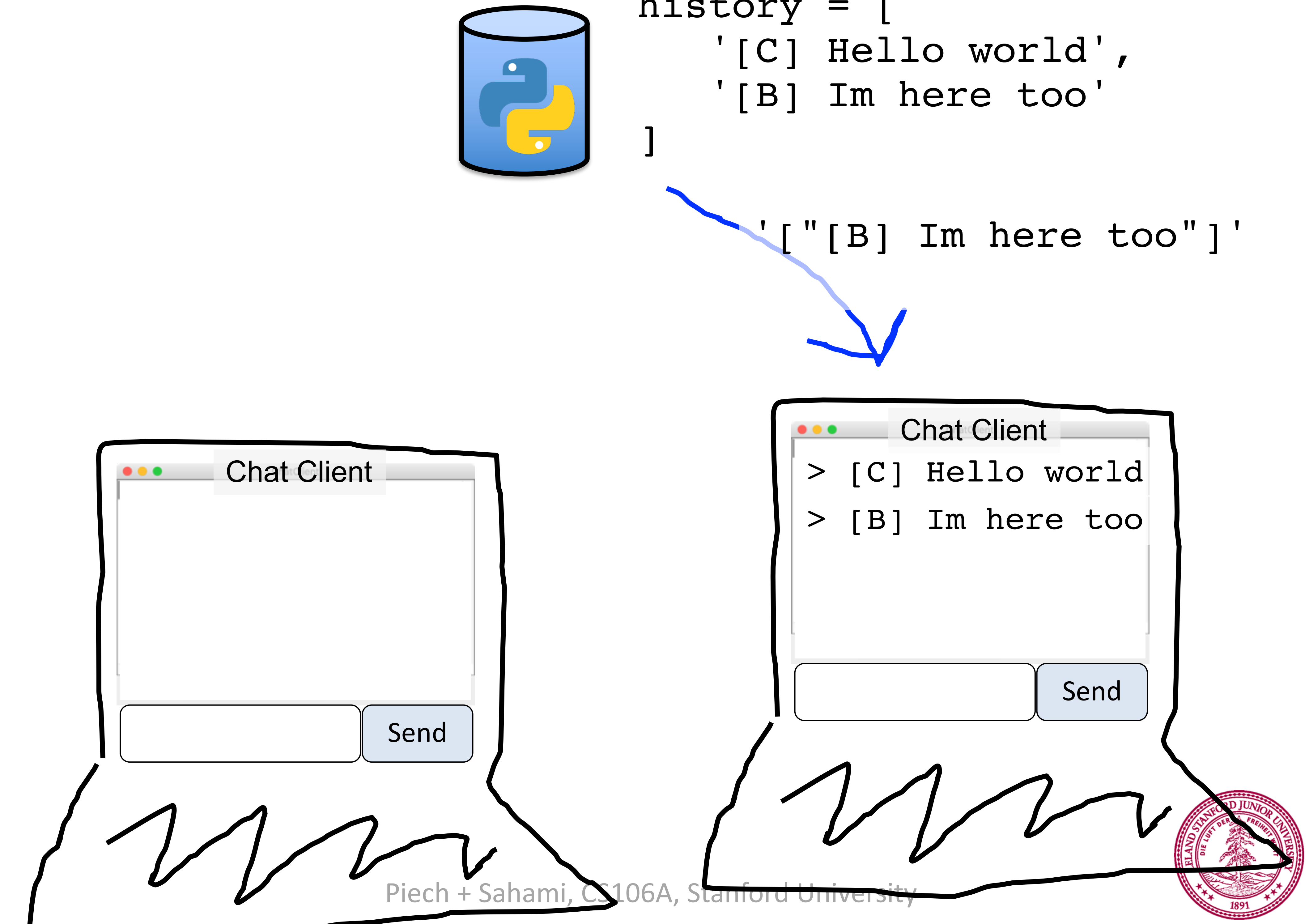
```
> [C] Hello world
```

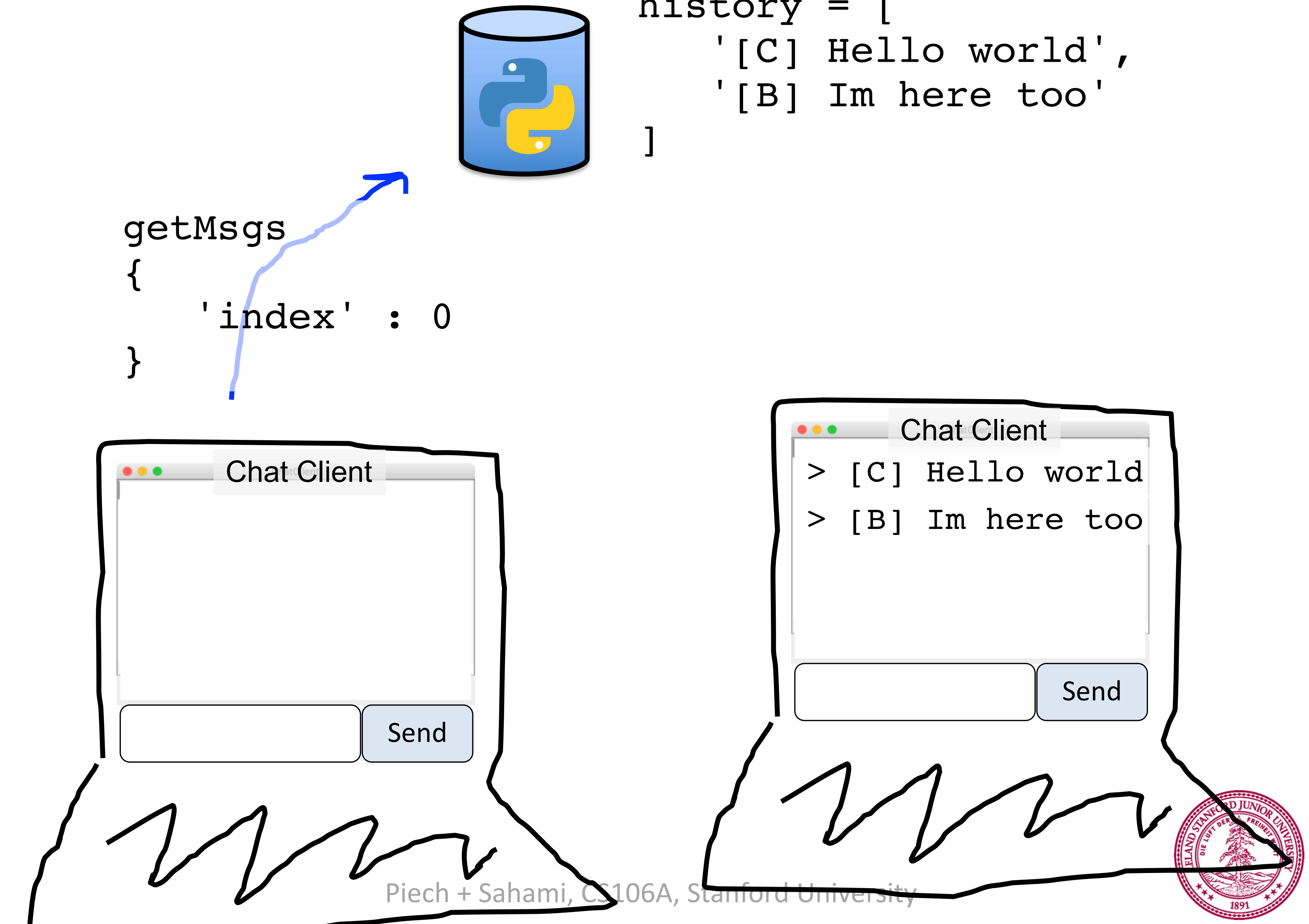


```
history = [
    '[C] Hello world',
    '[B] Im here too'
]
```

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

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

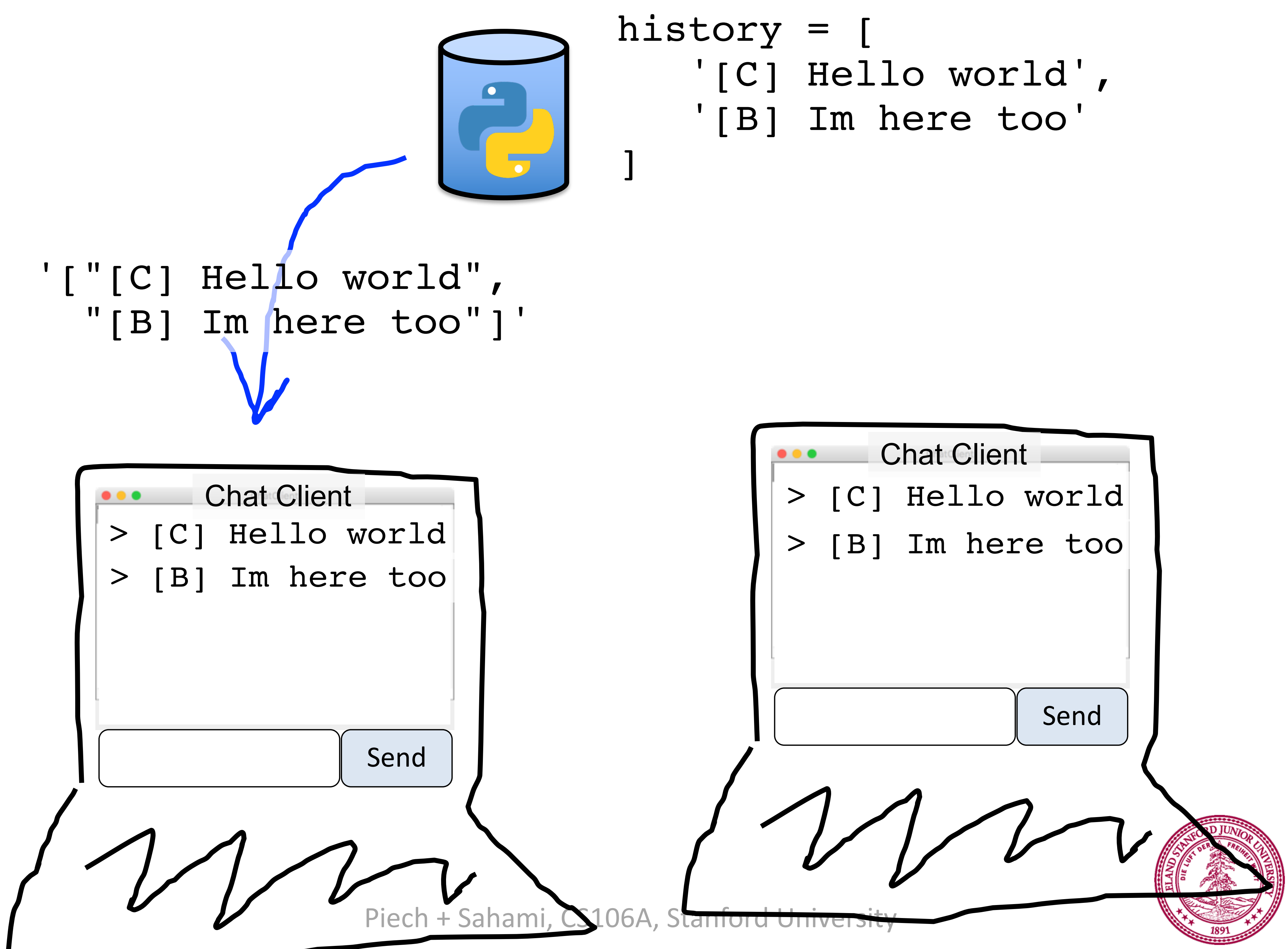




```
history = [
    '[C] Hello world',
    '[B] Im here too'
]
```

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

```
> [C] Hello world
> [B] Im here too
```



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

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

Chat Client

```
> [C] Hello world
> [B] Im here too
```

Chat Client

```
> [C] Hello world
> [B] Im here too
```

Chat Server

`Chat Server`

`addMsg`

msg = *text*

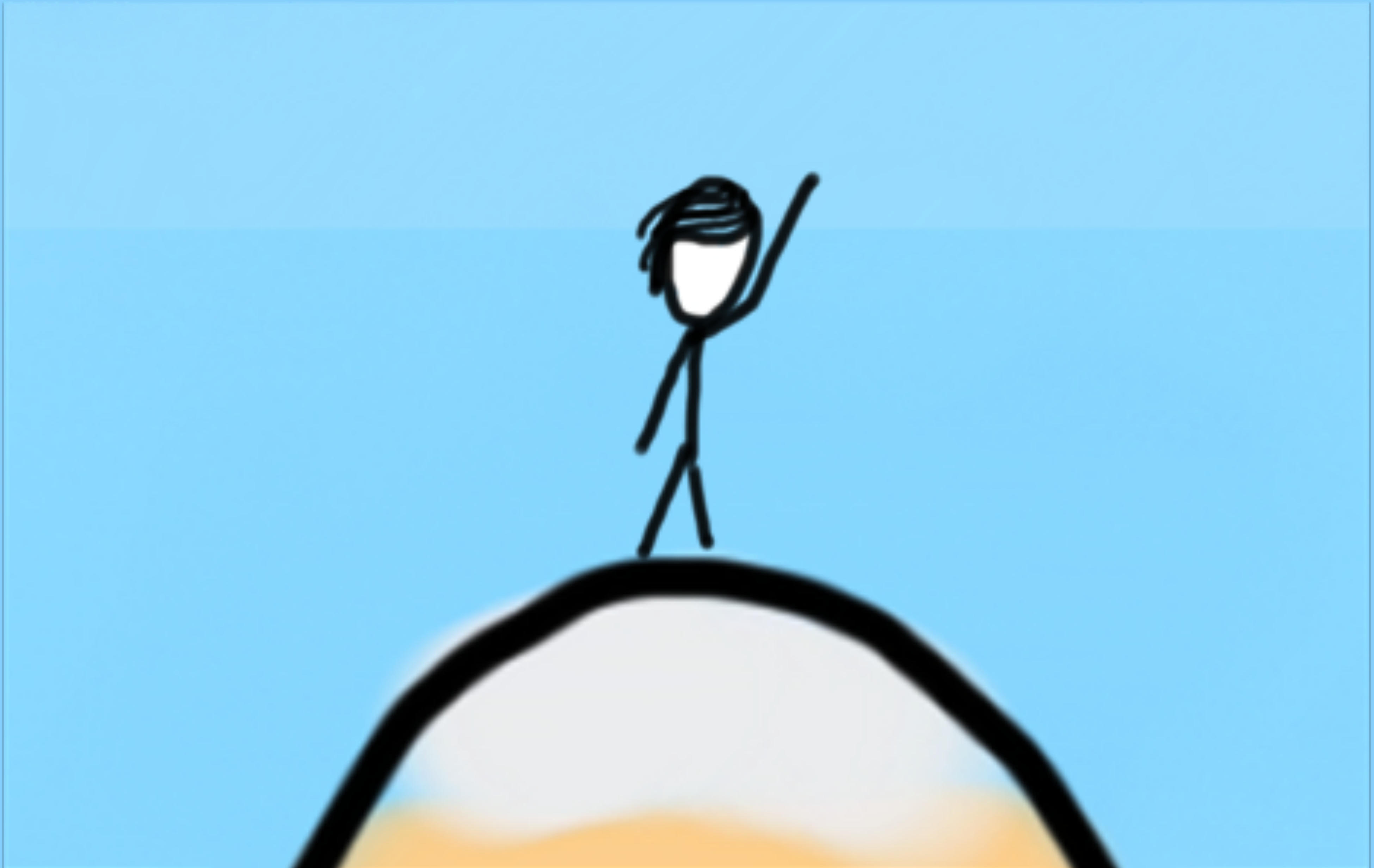
user = user

`getMsgs`

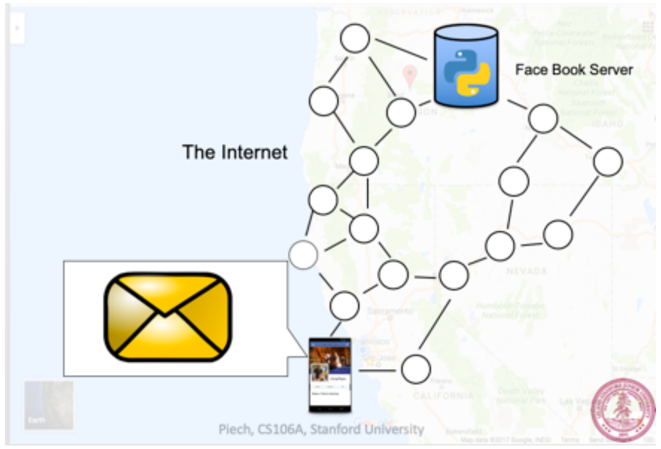
index = start_*index*

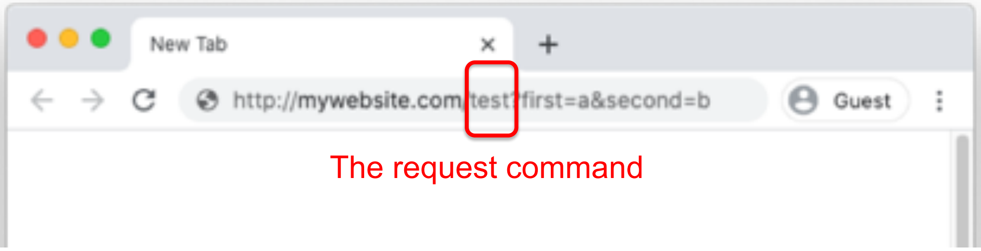
Learning Goals

1. Write a program that can respond to internet requests



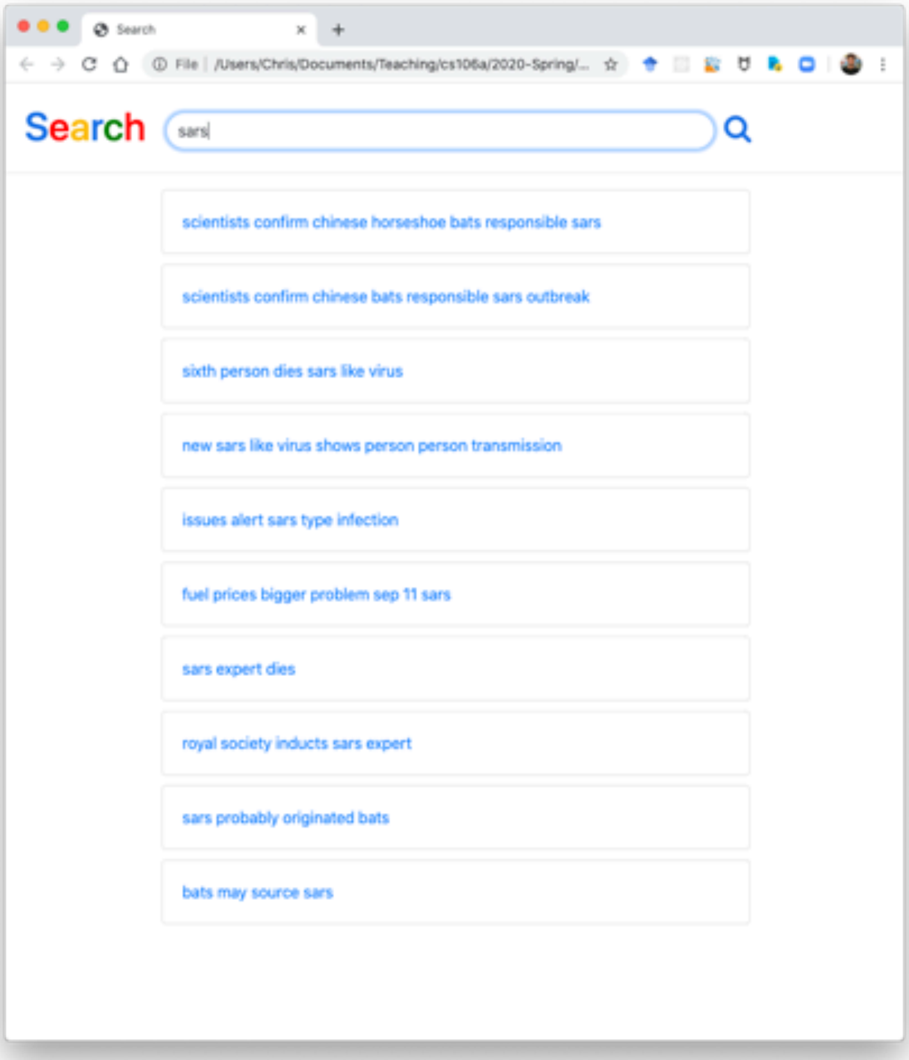
Things we saw along the way





```
response = requests.get(url)

 data_str = json.dumps(data)
```

Now you are ready...



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