Internet Applications

Lecture 24

CS106A, Summer 2019

Sarai Gould && Laura Cruz-Albrecht

With inspiration from slides created by Keith Schwarz, Mehran Sahami, Eric Roberts, Stuart Reges, Chris Piech, Brahm Capoor, & others.

Announcements

- Blank lecture code on website Course Schedule

Learning Goals for Today

1. Write a program that can **make internet requests**

2. Write a program that can **respond to internet requests**

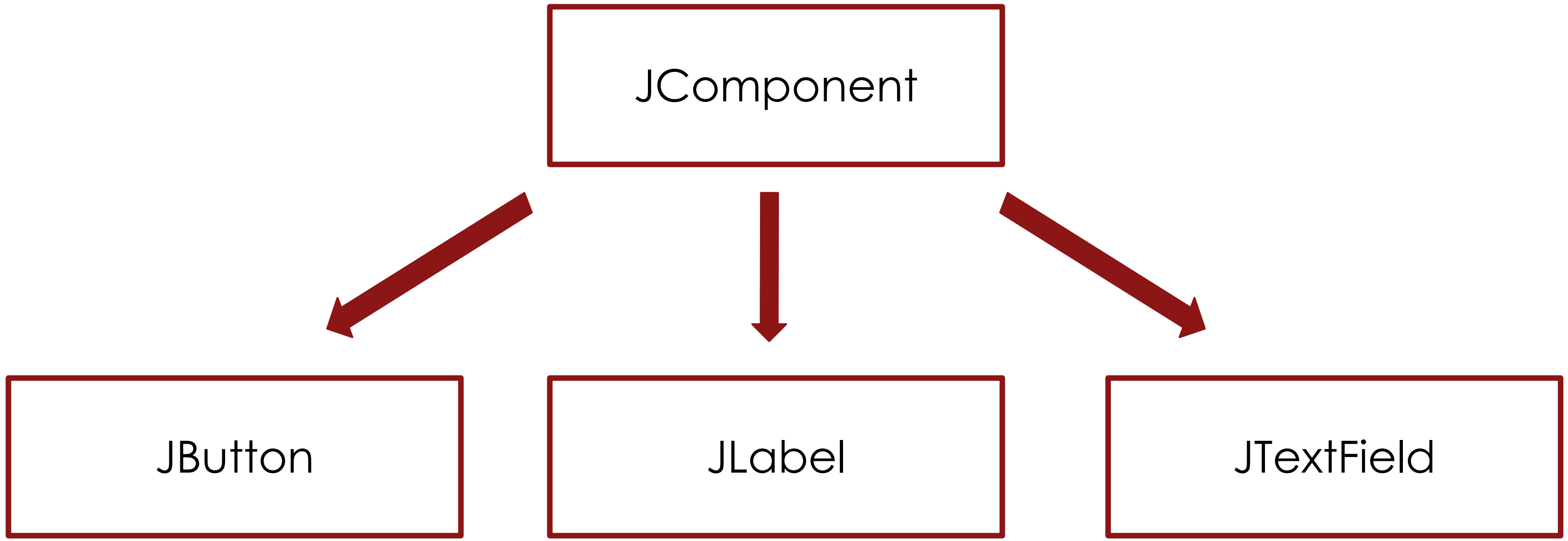
Plan for Today

- Review: Interactors
- Internet 101
- Servers & Clients
- Practice: Polling

Plan for Today

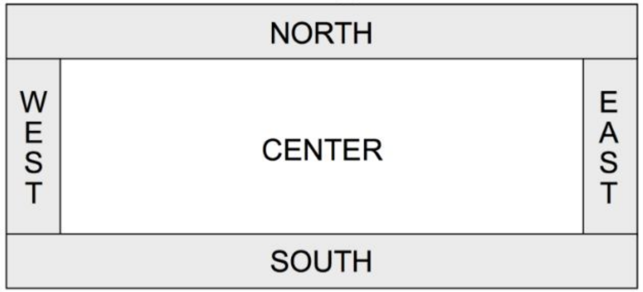
- Review: Interactors
- Internet 101
- Servers & Clients
- Practice: Polling

Review: Interactors



Review: Interactors

Interactors can be placed in 5 regions on the screen.

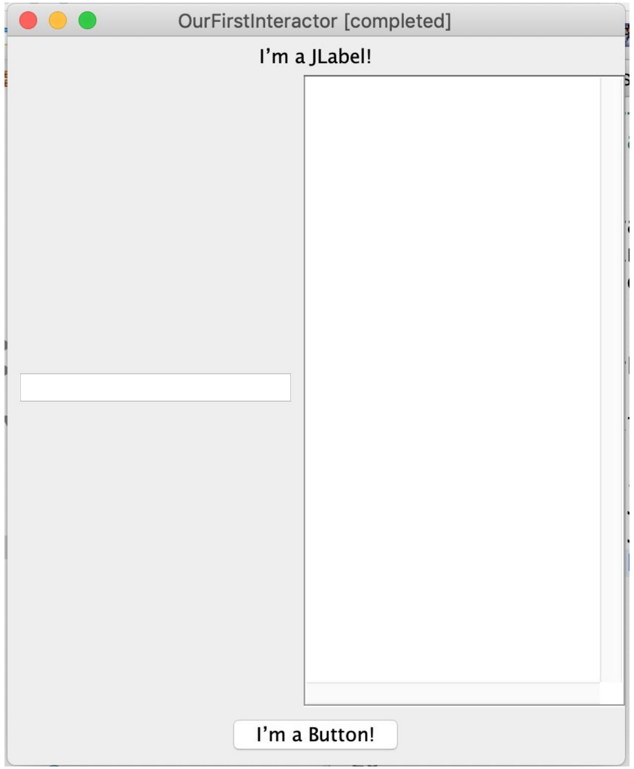


- The center is usually where things happen!
  - The ConsoleProgram adds the Console there.
  - The GraphicsProgram add the Canvas there.
- We only see the other regions of the screen if we add interactors there using `add(component, REGION)`
- Interactors are automatically centered in their region.

Review: Our First Interactor

```
import javax.swing.*;
import java.awt.event*;

public class ourFirstInteractor extends ConsoleProgram {

    private JTextField textField = new JTextField(15);

    public void init(){
        add(new JLabel("I'm a JLabel!"), NORTH);
        add(new JButton("I'm a Button!"), SOUTH);
        add(textField, WEST);
        addActionListeners();
    }

}
```

In order to detect actions in these fields, we must `addActionListeners()`



Review: actionPerformed

Method	Description
e.getActionCommand()	a text description of the event *(e.g., the text of the button clicked)*
e.getSource()	the interactor that generated the event

```
public void actionPerformed(ActionEvent e){

    String command = e.getActionCommand();
    if(command.equals(“Button 1”)){
        println(“Button 1 was pressed”);
    } else if (command.equals(“Button 2”)){
        println(“Button 2 was pressed”);
    }

}
```

Plan for Today

- Review: Interactors
- Internet 101
- Servers & Clients
- Practice: Polling

https://www.information-age.com/year-internet-critical-things-123465491/

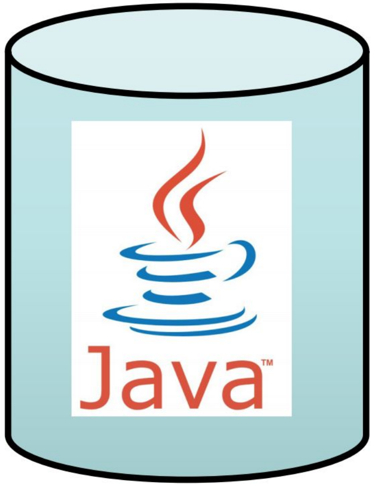
Programs and the Internet

How does your phone communicate with Facebook?

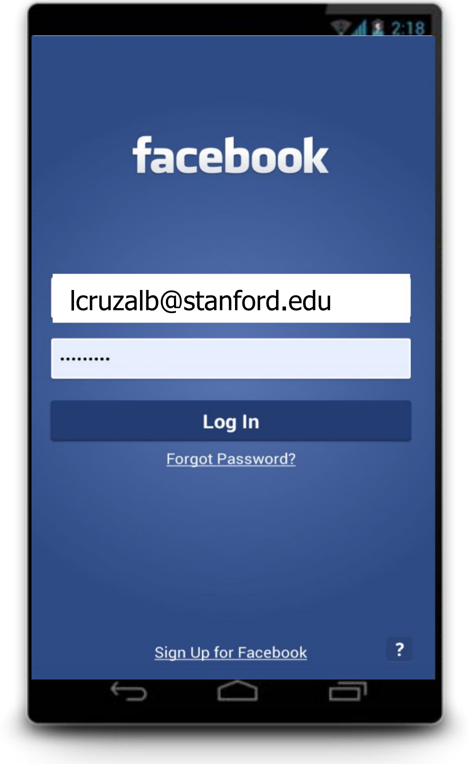
Programs and the Internet

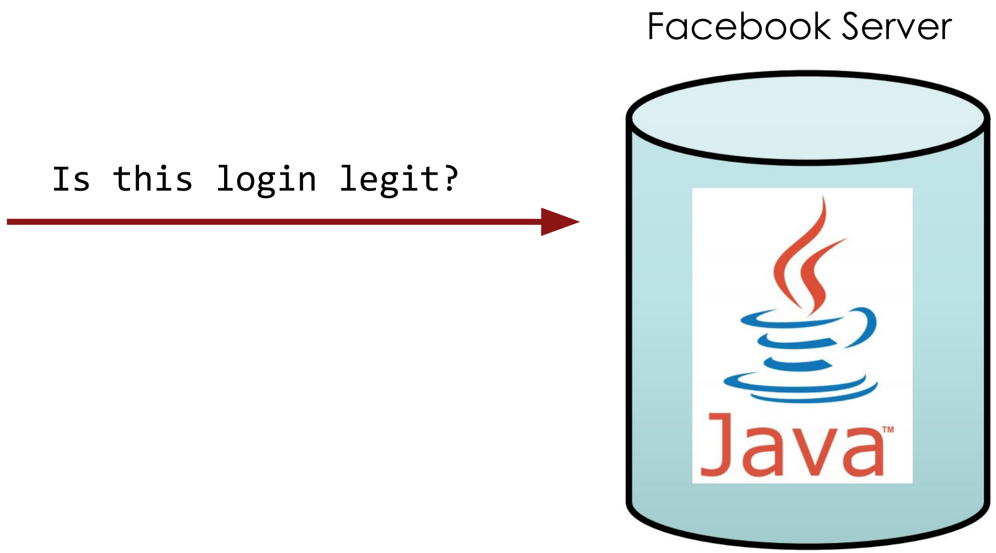
The Java program on your *phone* talks to the Java program at *Facebook*.

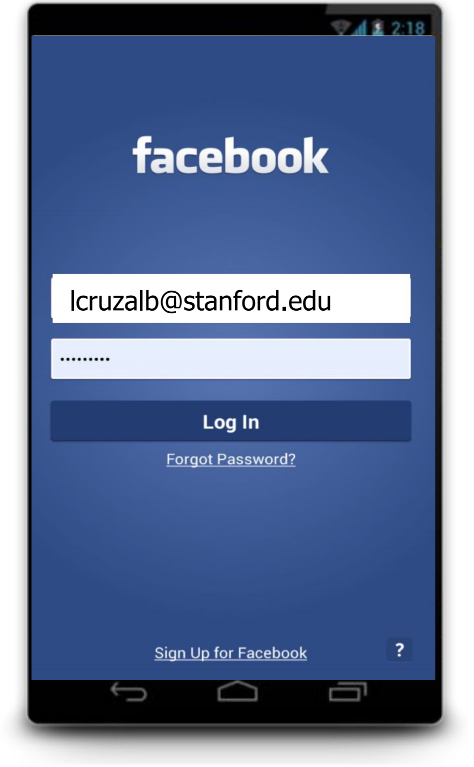
Facebook Server



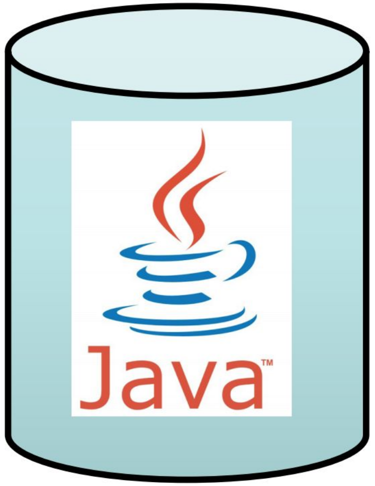
* Android phones run Java. So do Facebook servers.


2:18
facebook
lcruzalb@stanford.edu
....
Log In
Forgot Password?
Sign Up for Facebook
?


Facebook Server
Is this login legit?
Java™

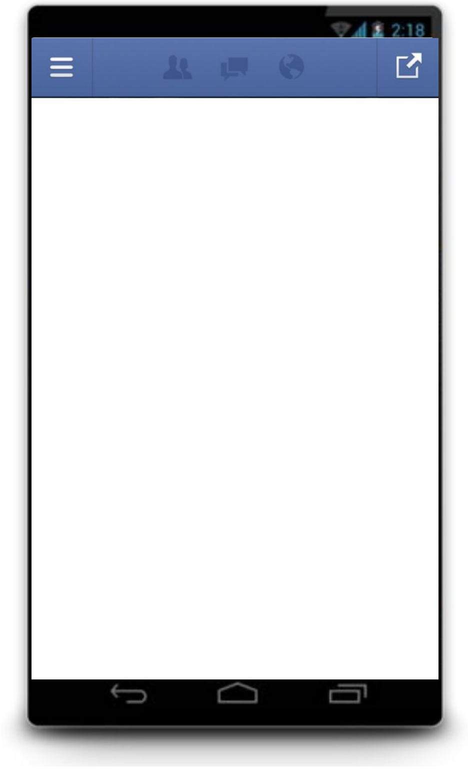


Facebook Server

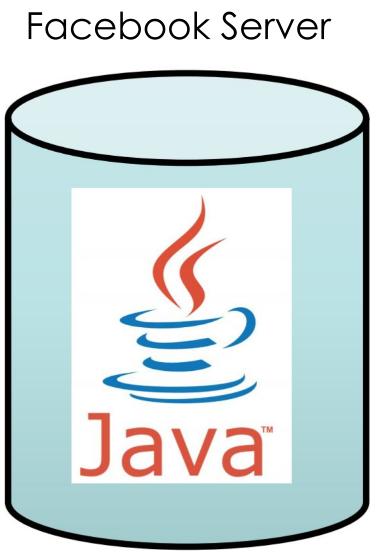


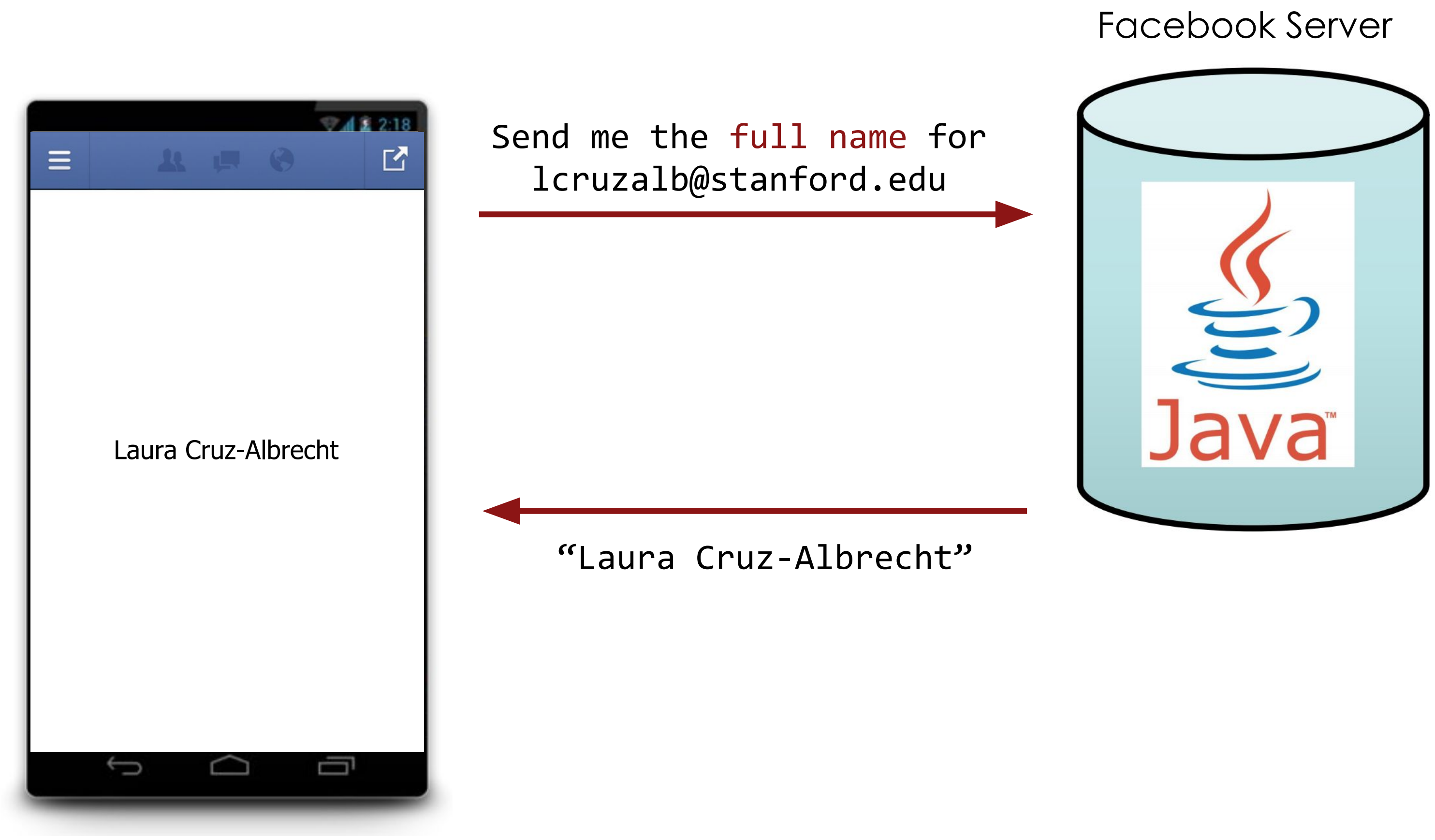
```
Is this login legit?
```

```
Confirmed.
lcruzalb@stanford.edu
is now logged in.
```



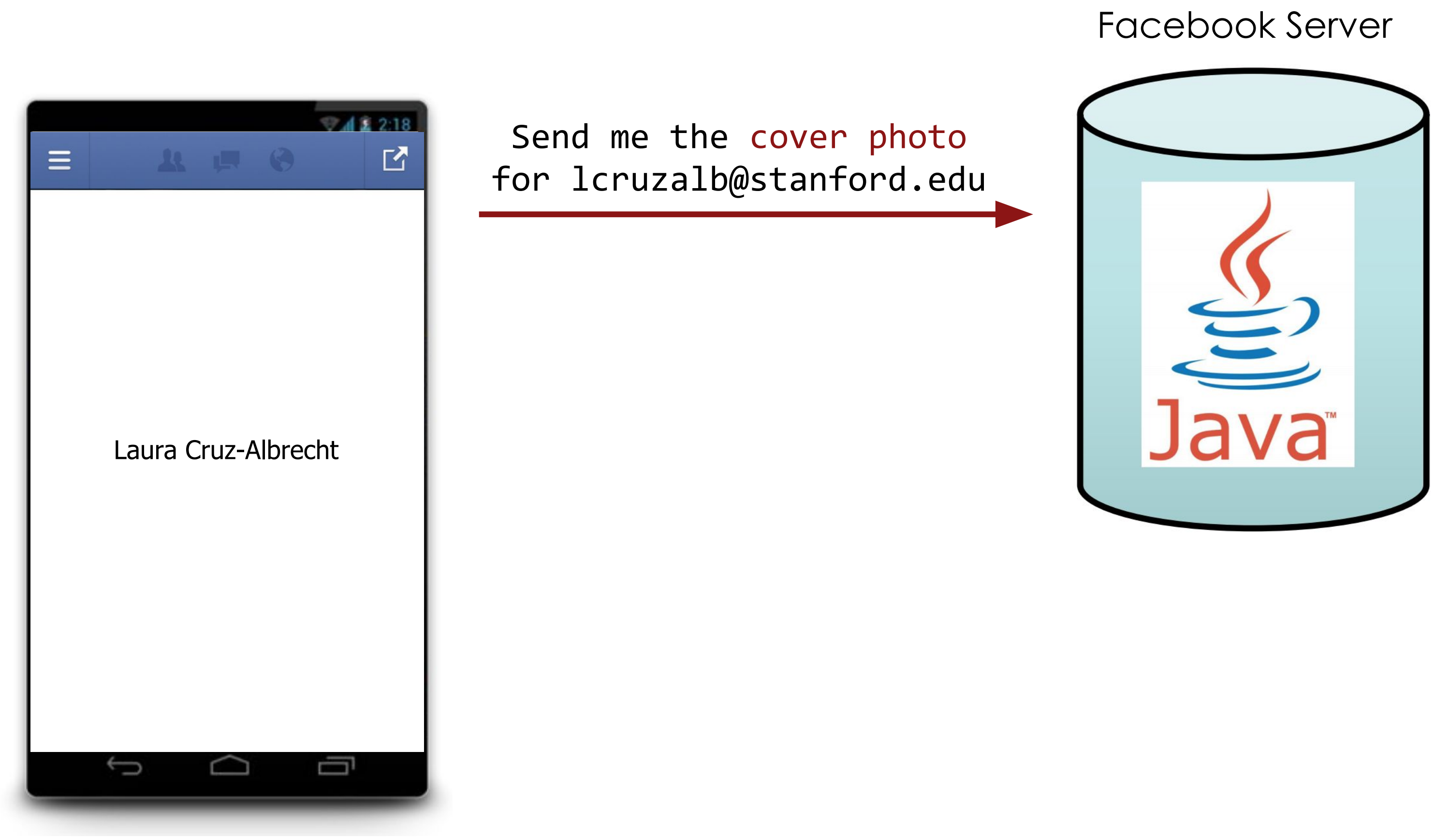
Send me the full name for lcruzalb@stanford.edu

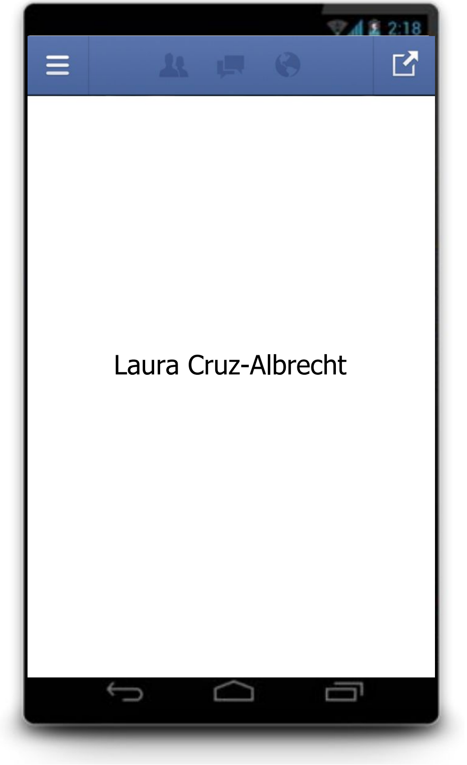




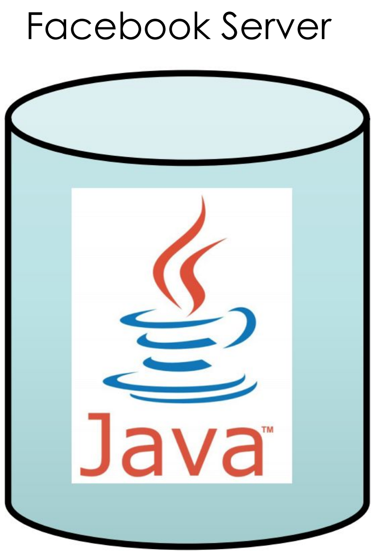
```
Send me the full name for
  lcruzalb@stanford.edu
```

```
"Laura Cruz-Albrecht"
```

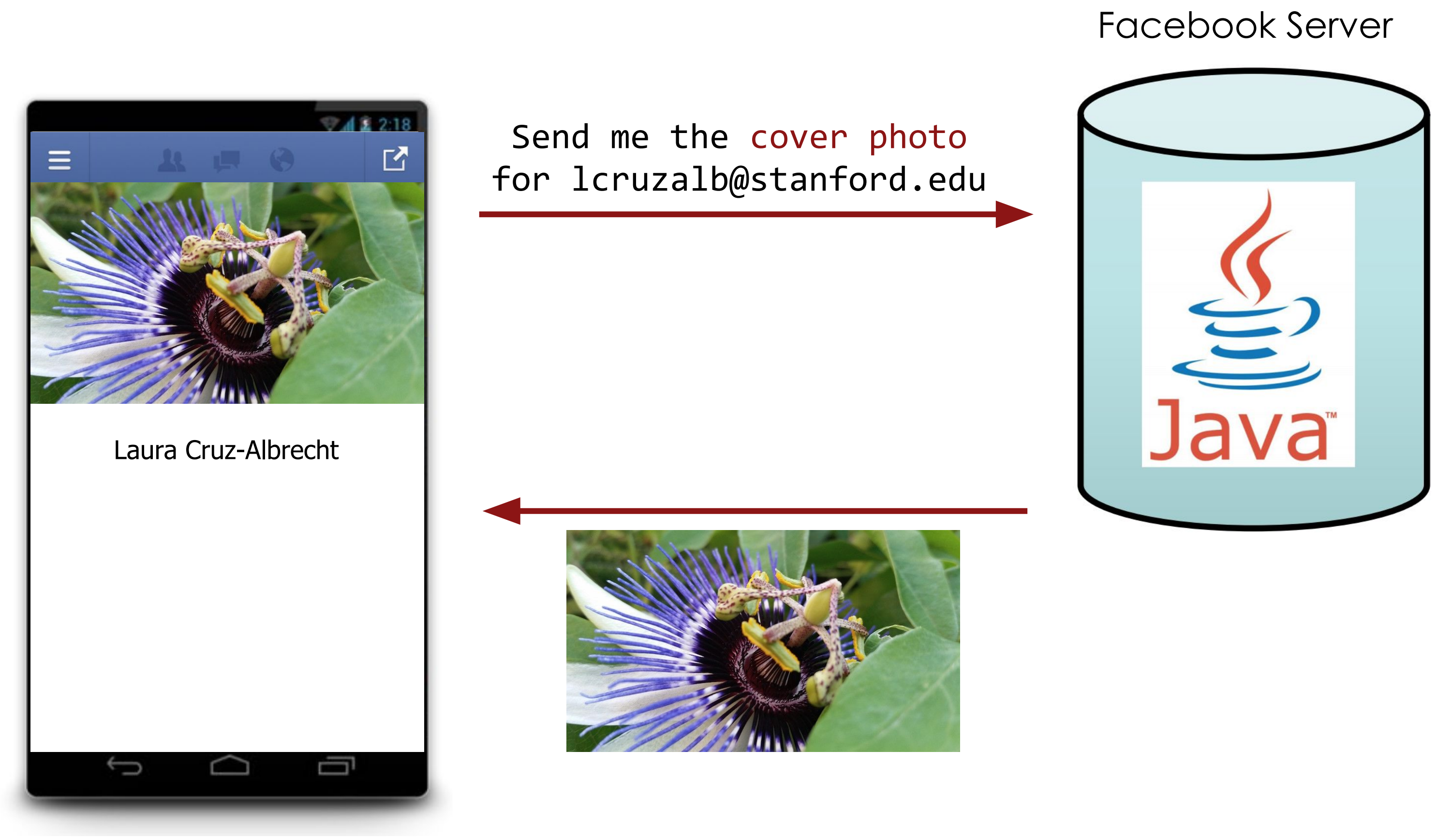

Facebook Server
Send me the cover photo
for lcruzalb@stanford.edu
Laura Cruz-Albrecht
2:18
Java



Send me the cover photo
for lcruzalb@stanford.edu



where did I put
that picture...


Facebook Server
Send me the cover photo
for lcruzalb@stanford.edu
Laura Cruz-Albrecht
Java™
2:18

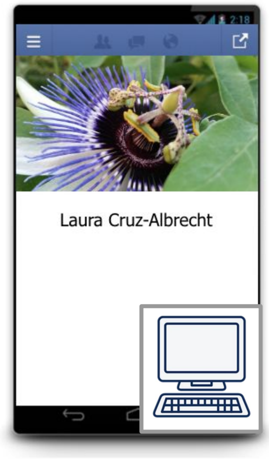
Plan for Today

- Review: Interactors
- Internet 101
- Servers & Clients
- Practice: Polling

There are two types of internet programs: **servers** and **clients**.

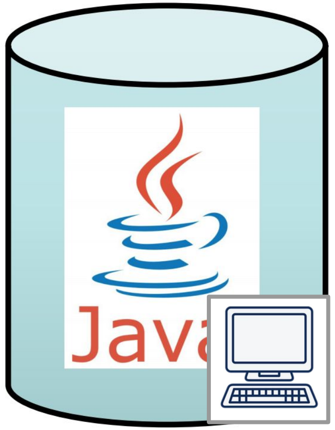
Clients send ***requests*** to servers. ***Servers respond*** to those requests.

Your phone/computer



“Client”

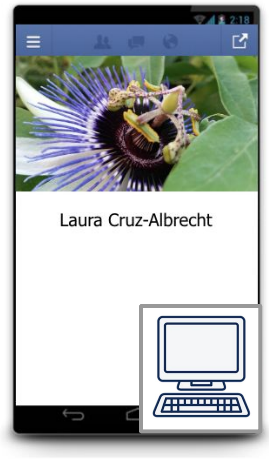
Facebook Server



“Server”

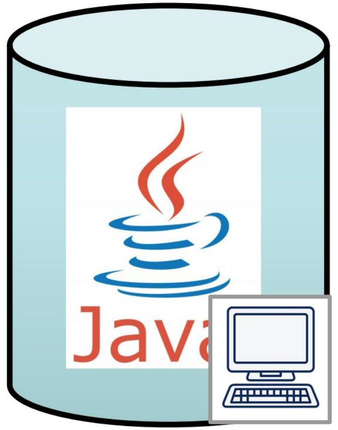
The internet is just a bunch of computers yelling at each other.

Your phone/computer



"Client"

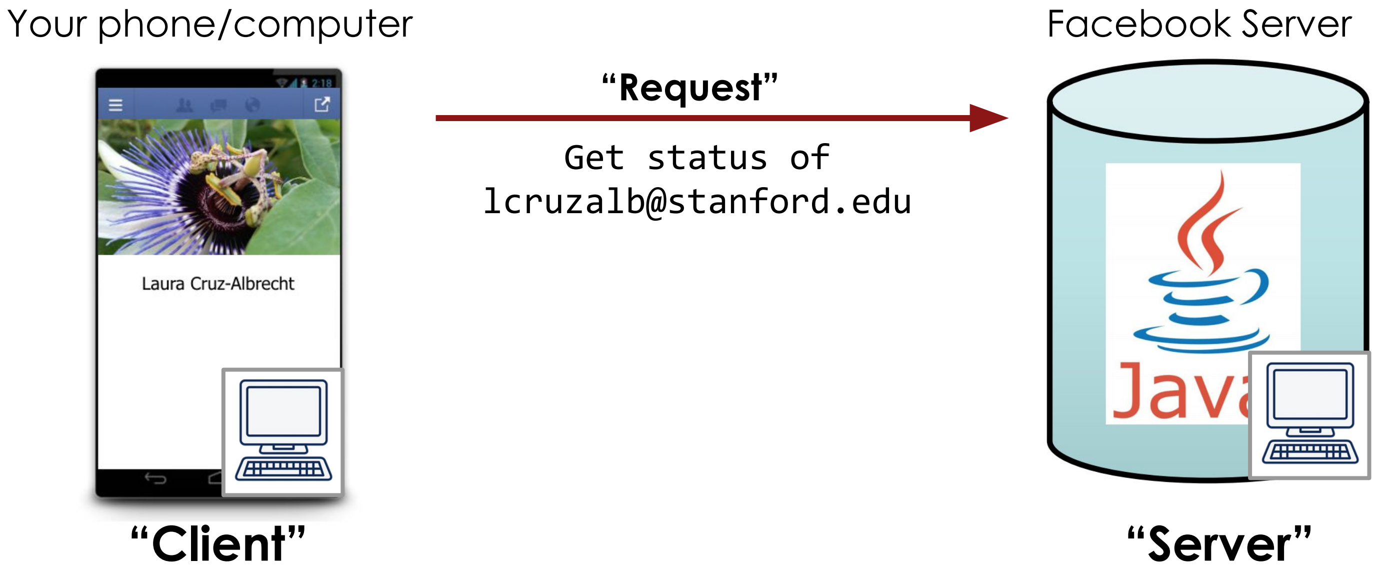
Facebook Server



"Server"

The internet is just a bunch of computers yelling at each other.

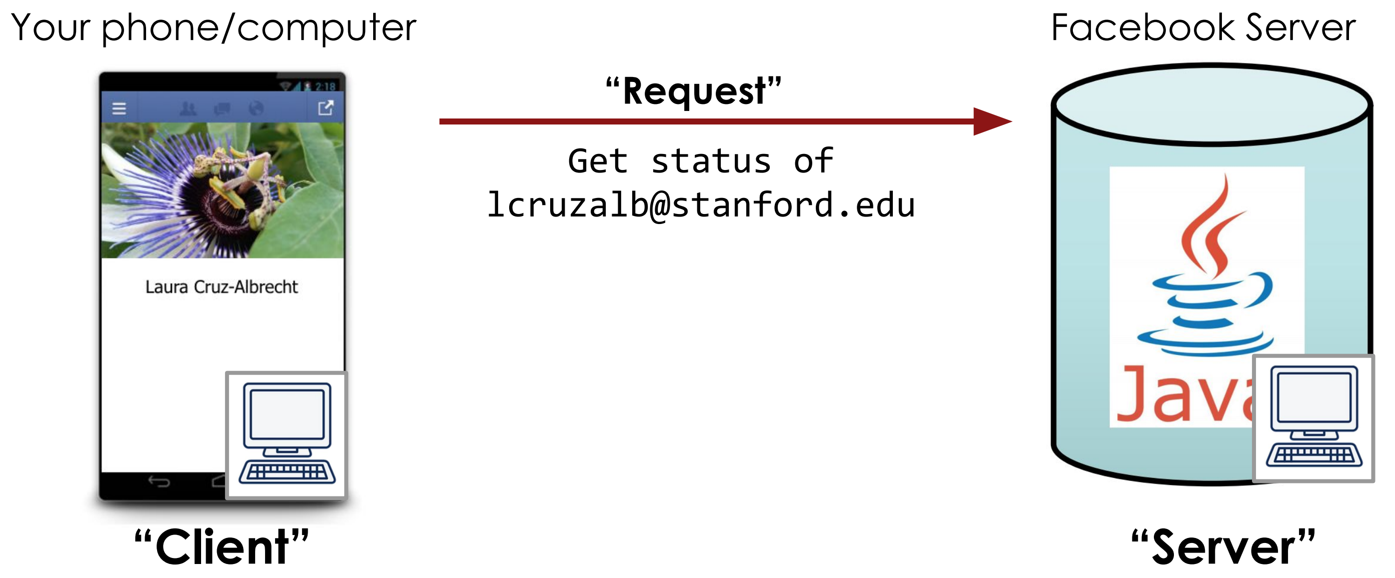
The computers that yell first are **clients**



```
Get status of
lcruzalb@stanford.edu
```

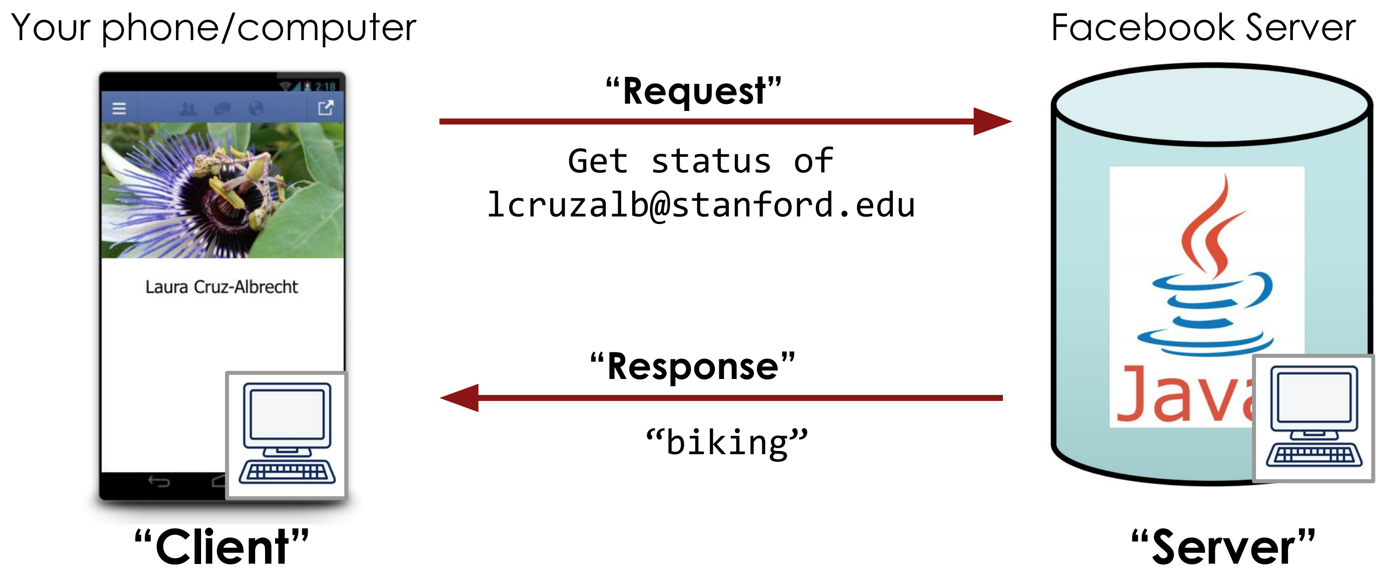
The internet is just a bunch of computers yelling at each other.

The computers that yell first are **clients**



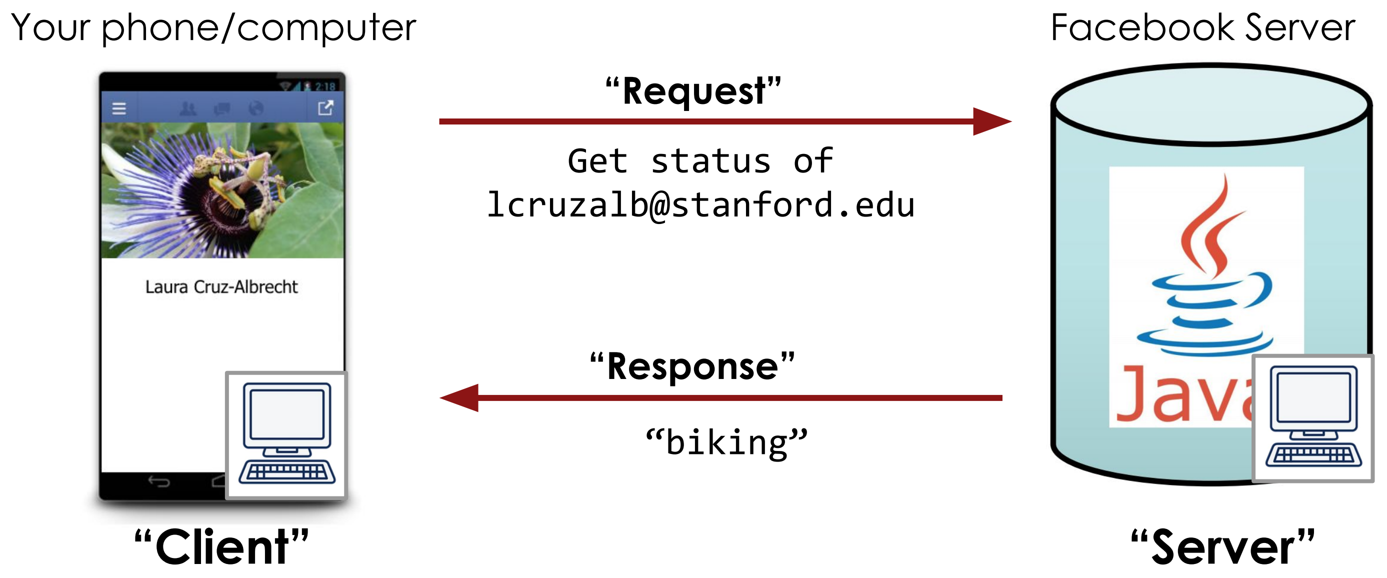
The internet is just a bunch of computers yelling at each other.

The computers that yell first are **clients**, and the computers that yell back are **servers**.



The internet is just a bunch of computers yelling at each other.

The computers that yell first are **clients**, and the computers that yell back are **servers**.



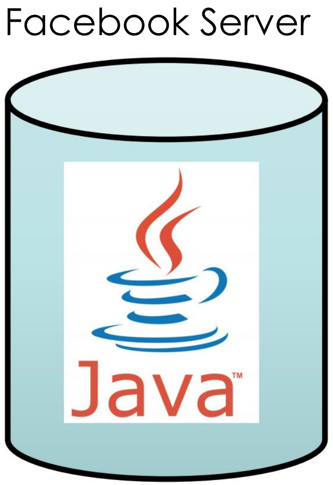
The internet is just a bunch of computers yelling at each other.

The computers that yell first are **clients**, and the computers that yell back are **servers**.

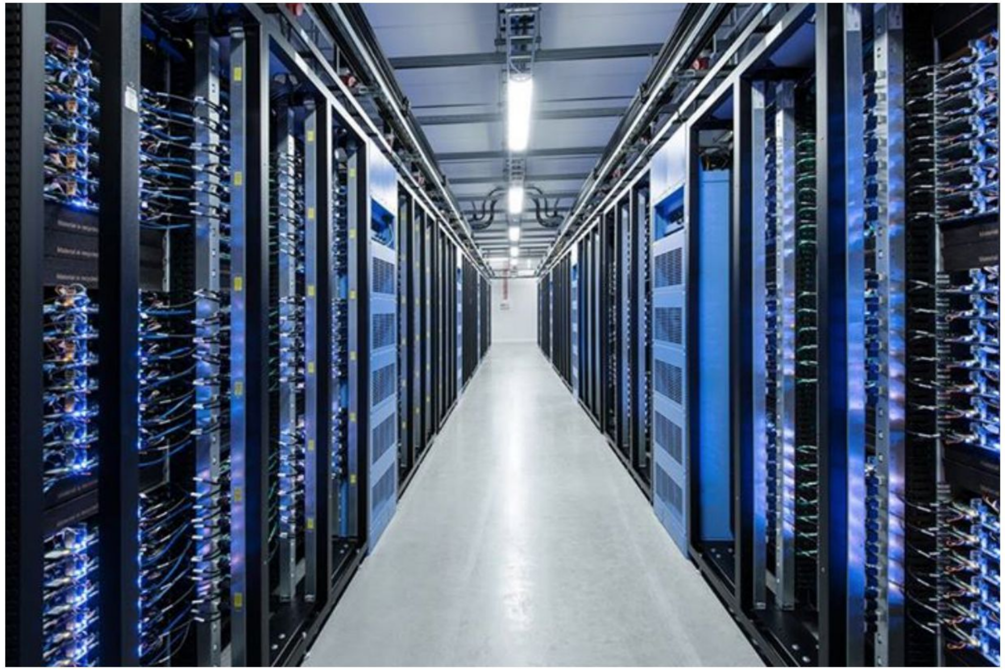
Each yell is a specially formatted `String`.

There are two types of internet programs: $\boxed{\textbf{servers}}$ and **clients**.

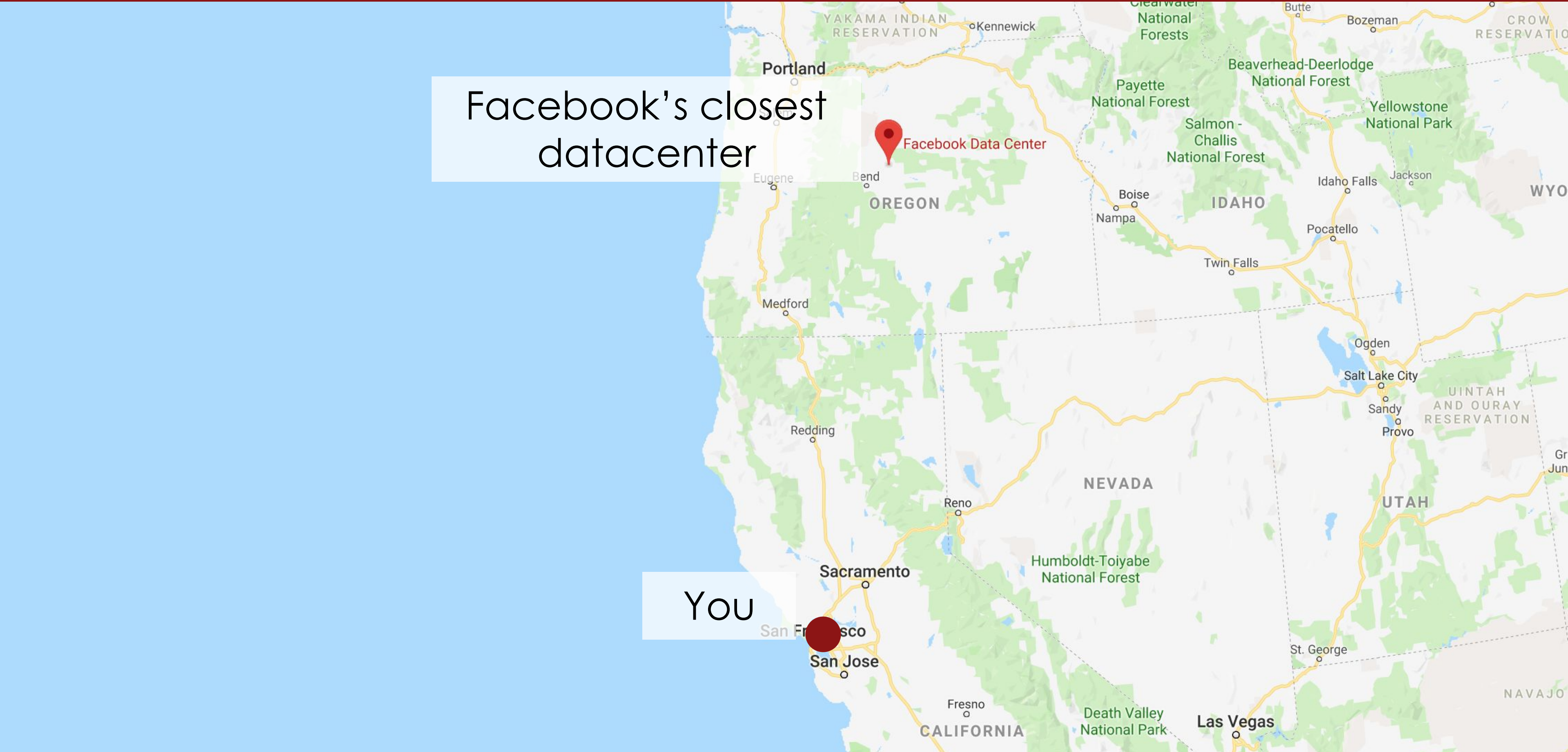
Servers are Computer Programs



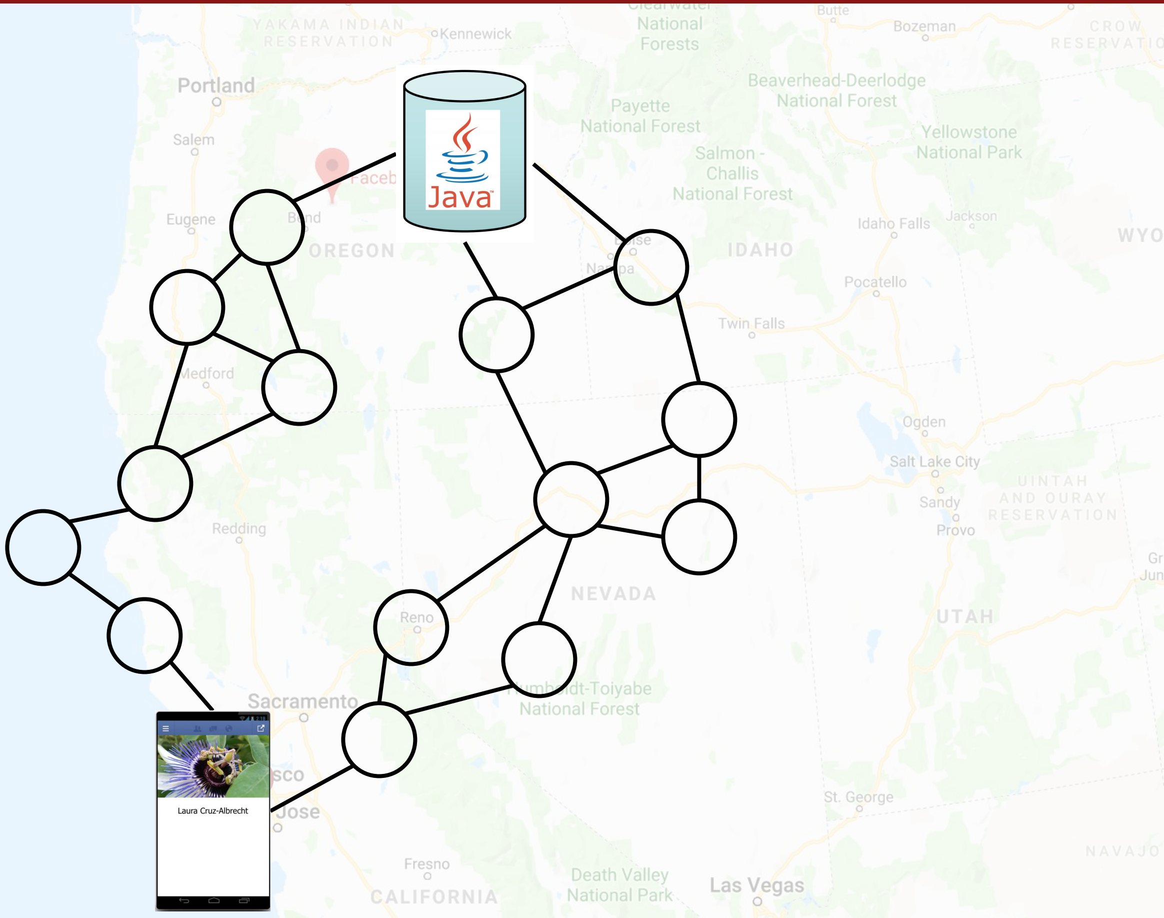
=



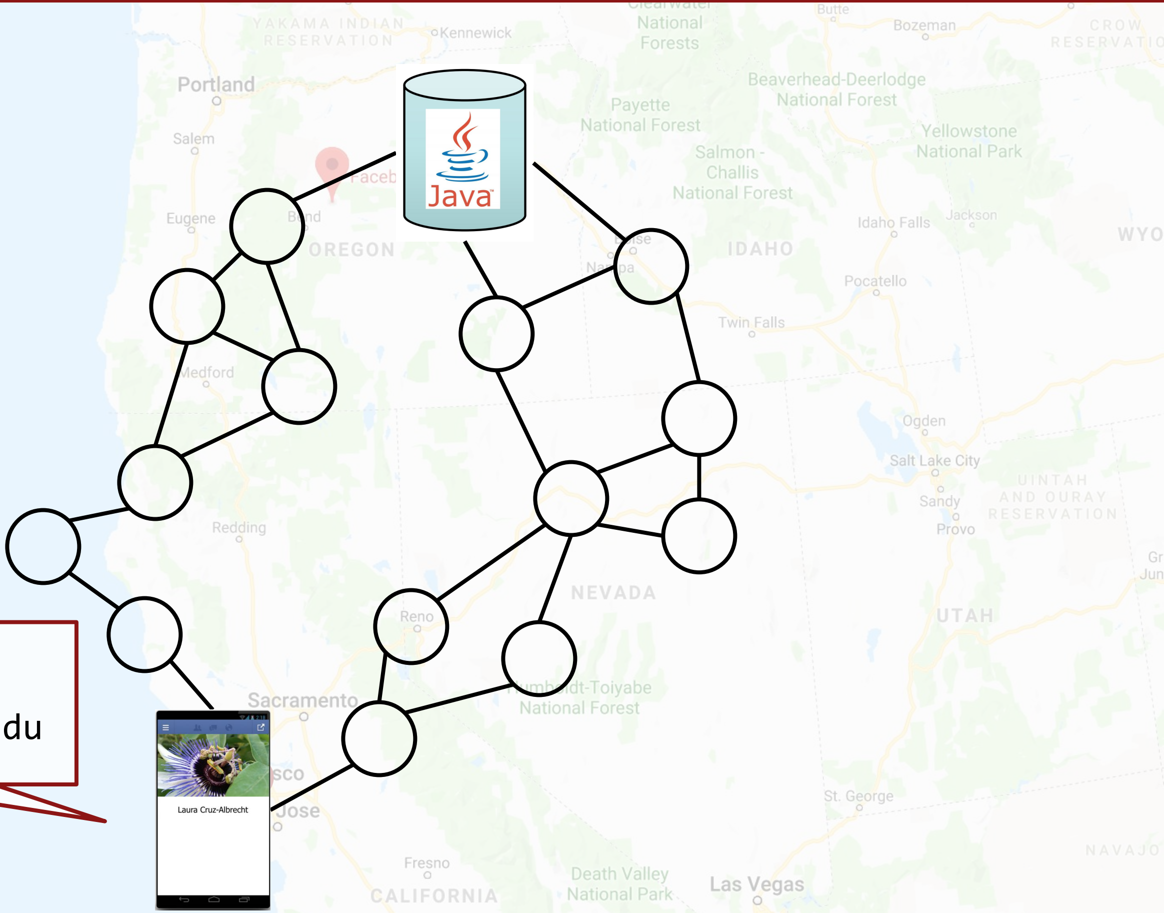
The Internet



The Internet

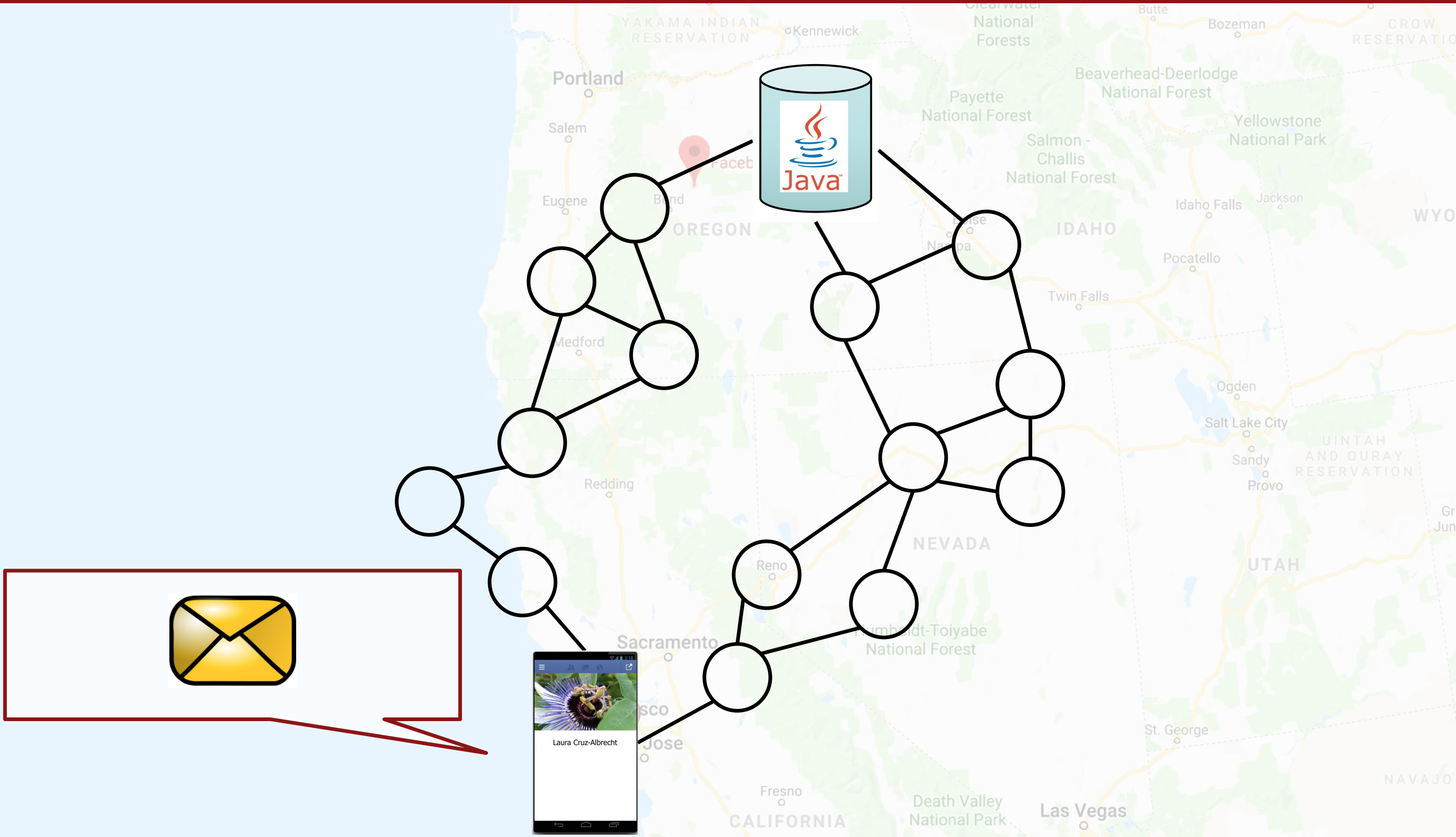


The Internet

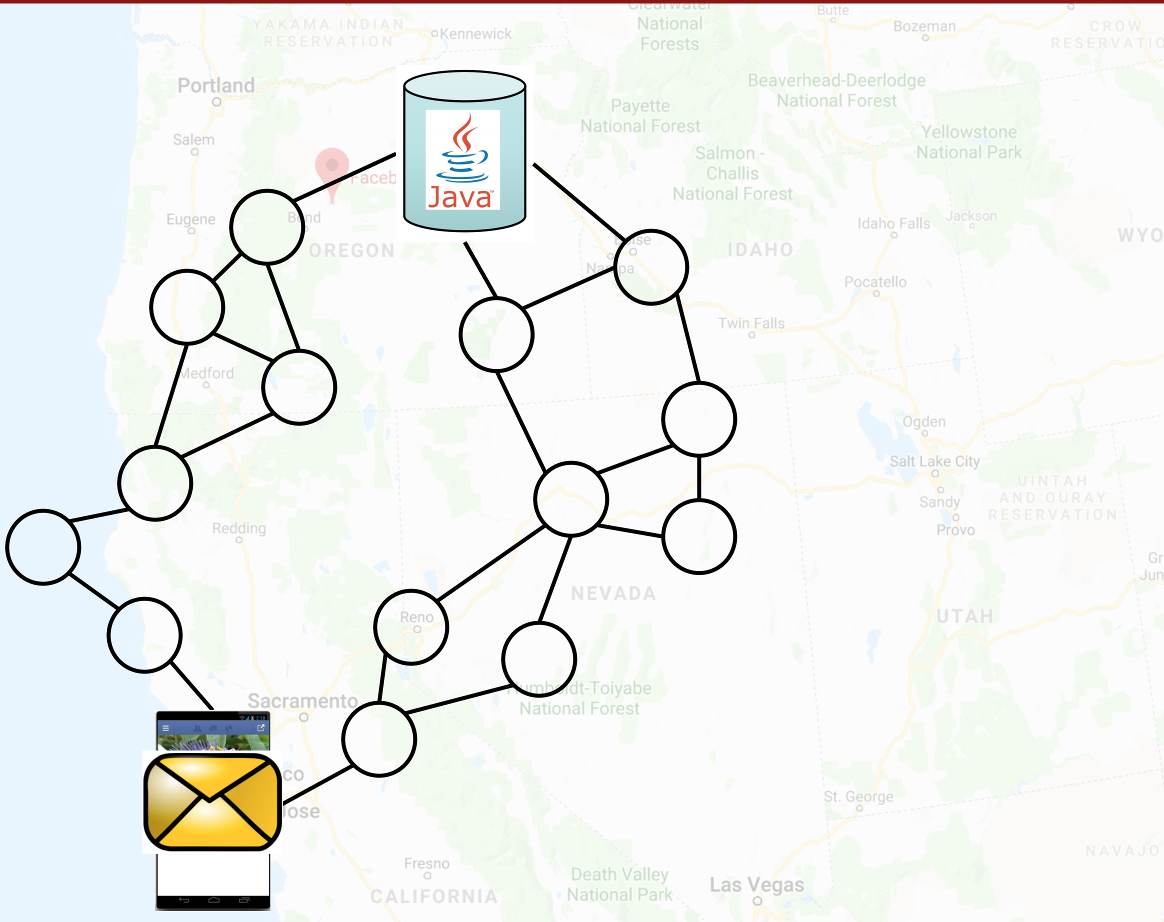


Get status for
lcruzalb@stanford.edu

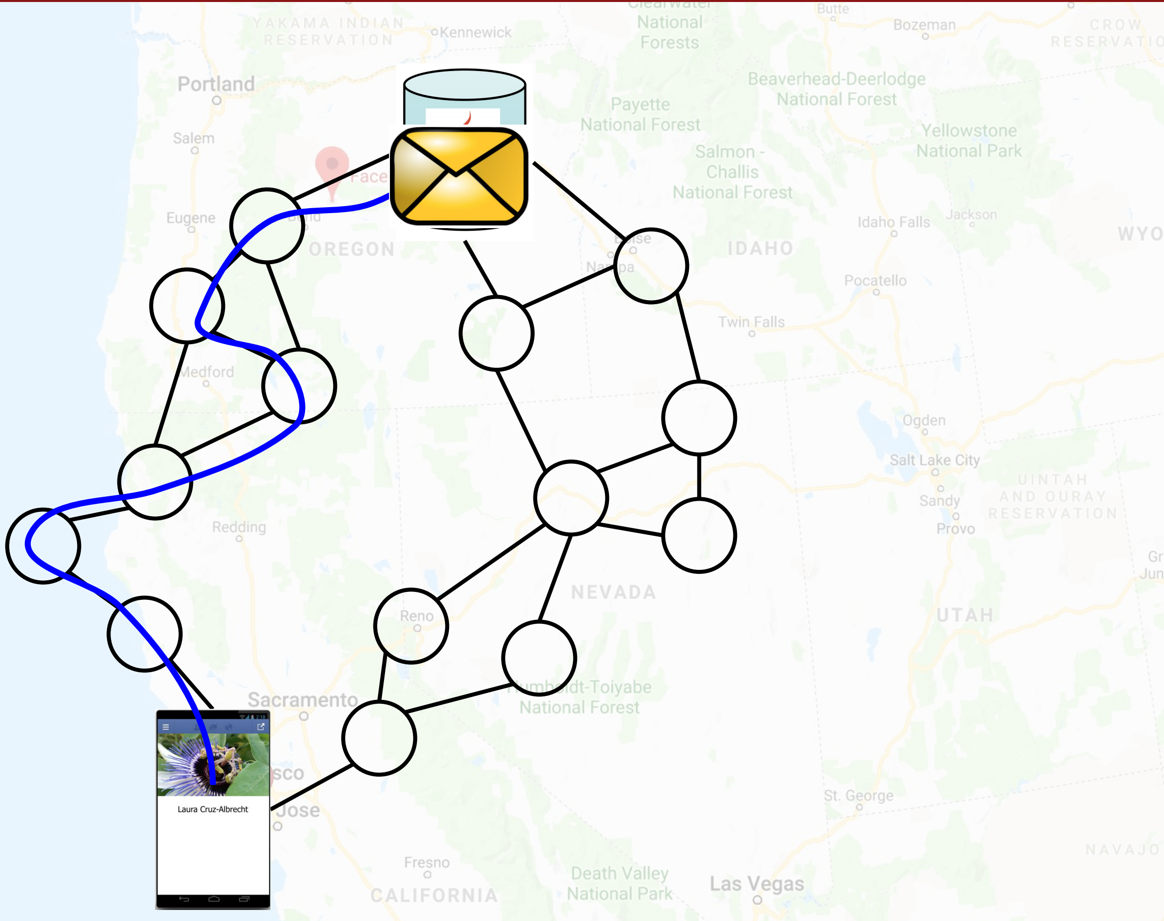
The Internet



The Internet

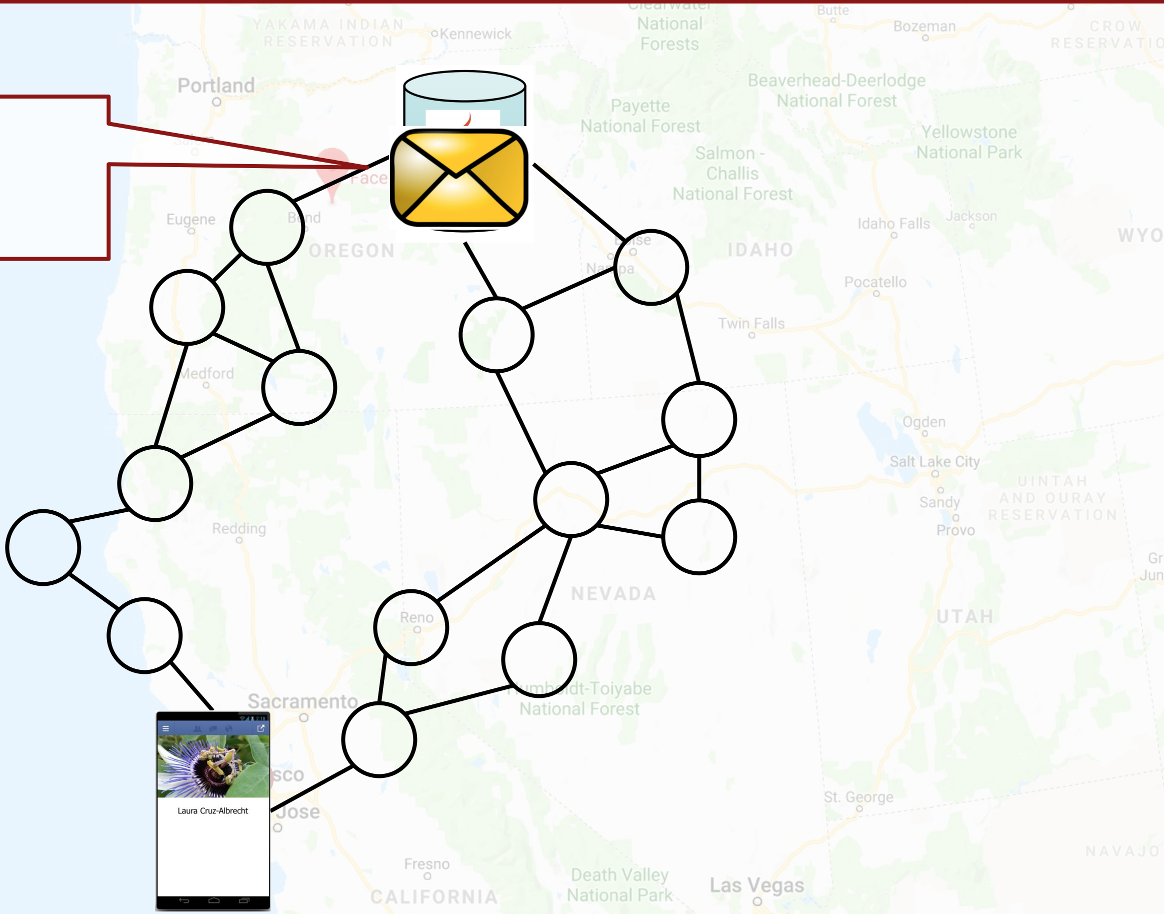


The Internet

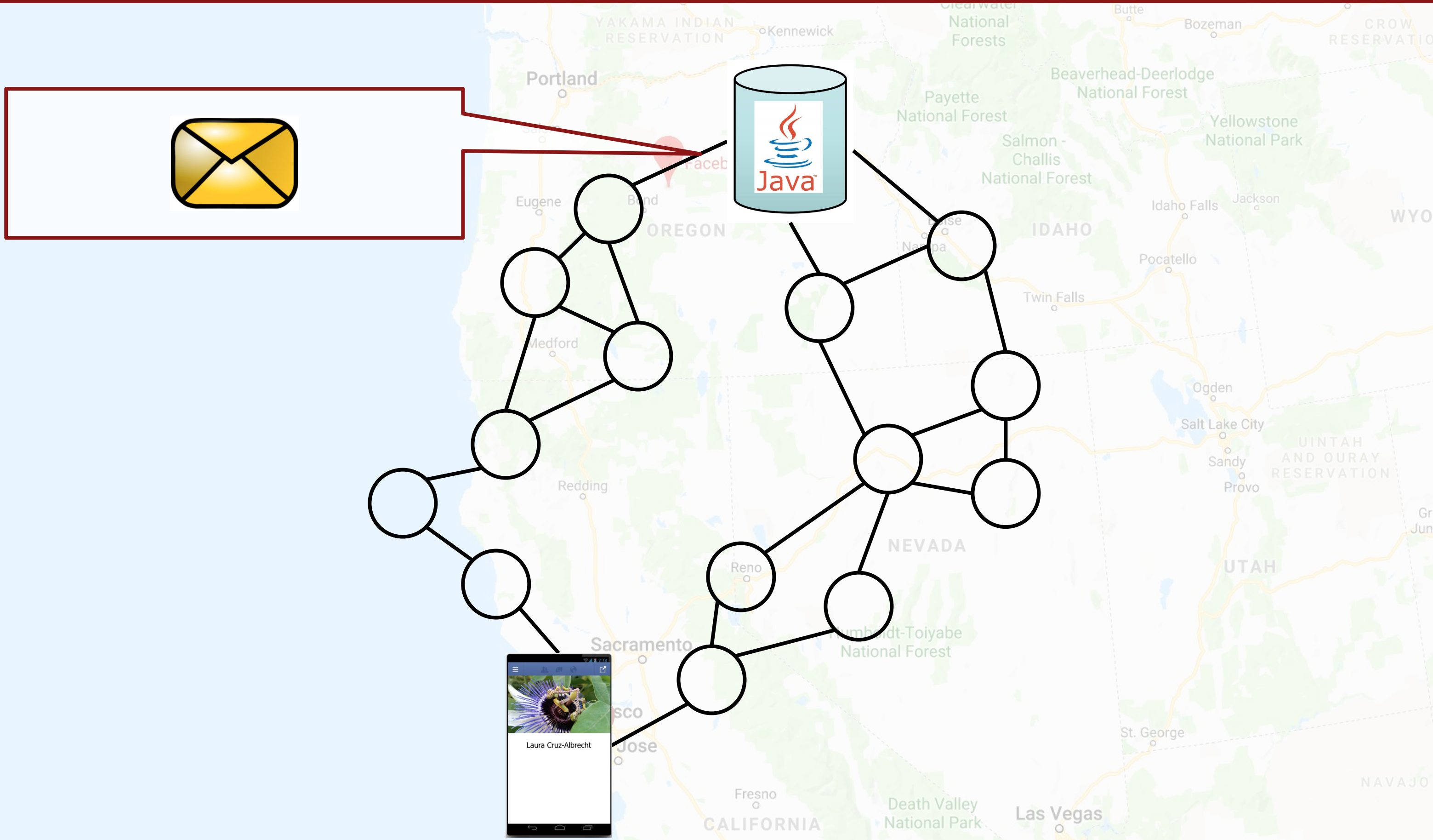


The Internet

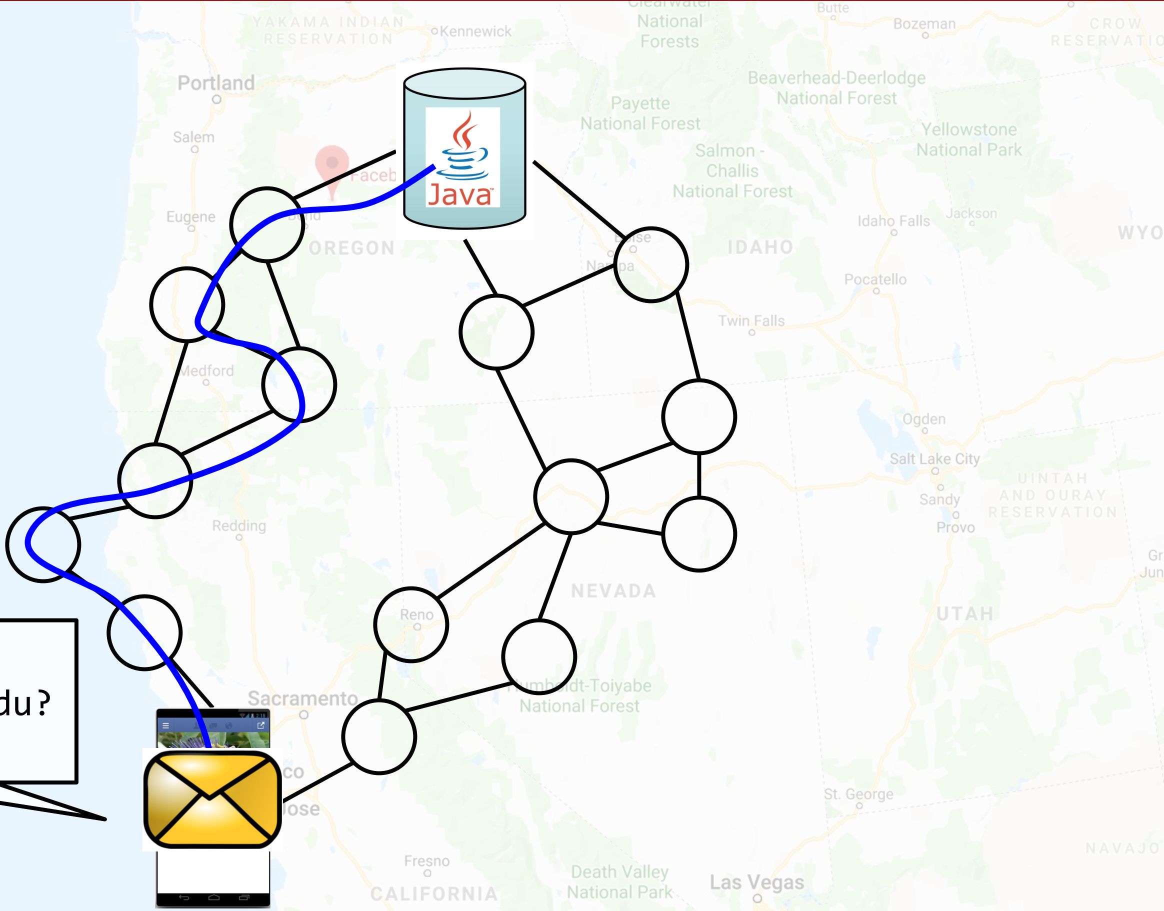
biking



The Internet

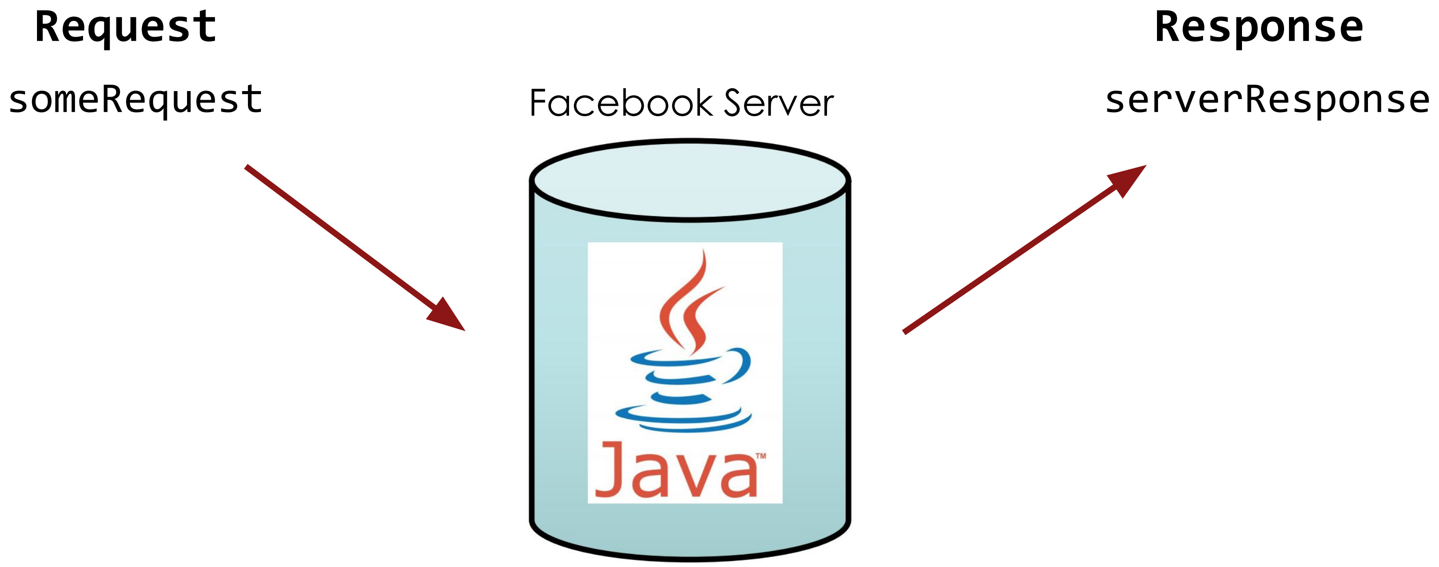


The Internet



Status for
lcruzalb@stanford.edu?
biking!

A Server's Simple Purpose



What is a Request?

```
/* Request has a command */
String command;

/* Request has parameters */
HashMap<String, String> params;
```

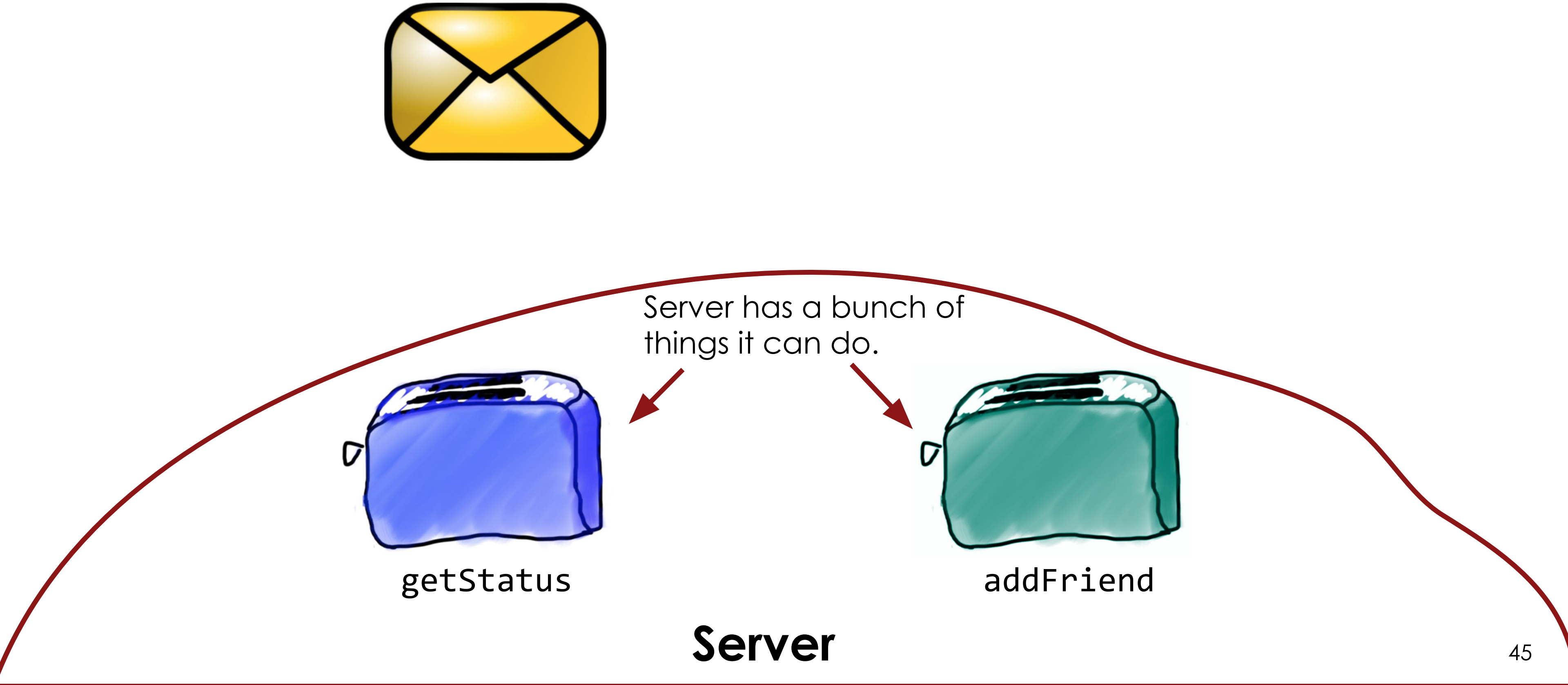
What is a Request?

```
Request request
```

```
/* Request has a command */
String command;

/* Request has parameters */
HashMap<String, String> params;
```

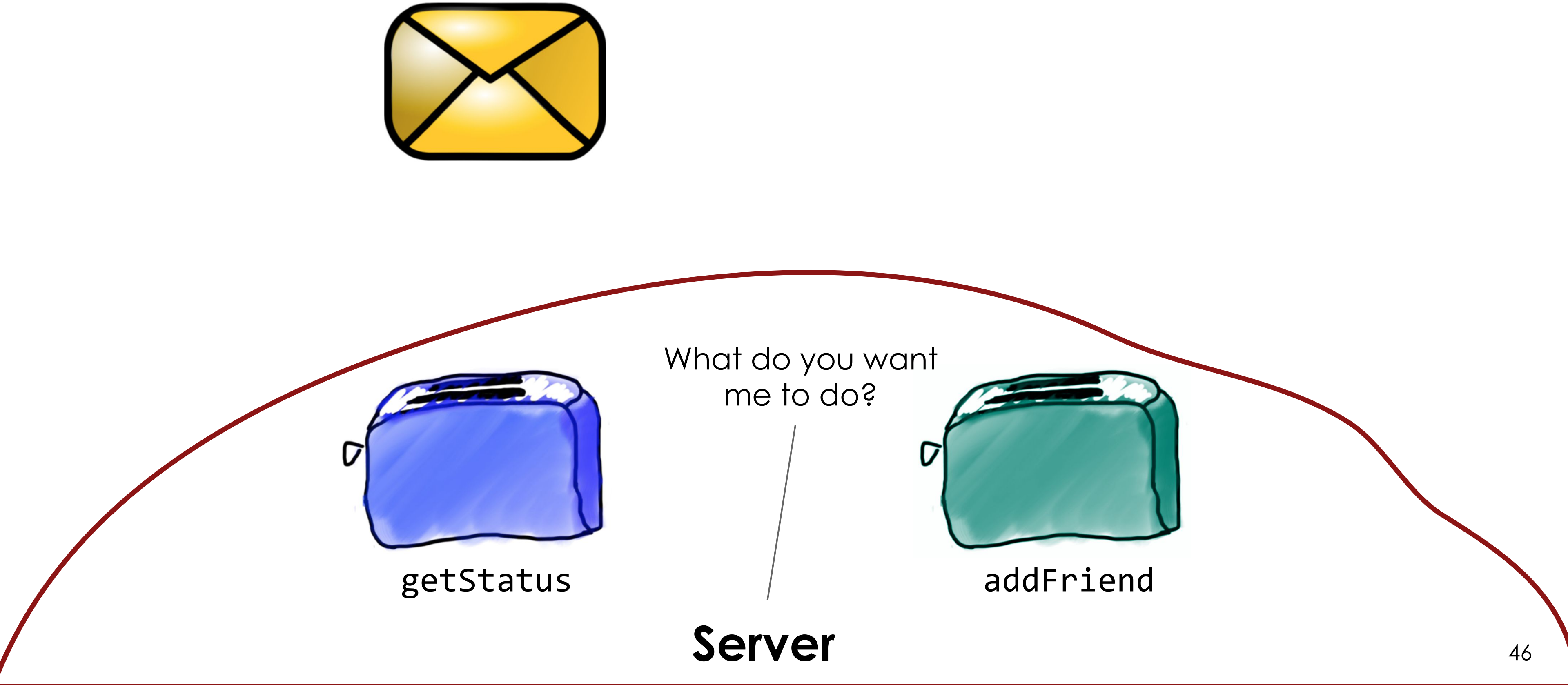
```
// Methods that the server calls on Request objects
request.getCommand();
request.getParam(key);    // returns associated value in map
```

Requests are like Remote Method Calls

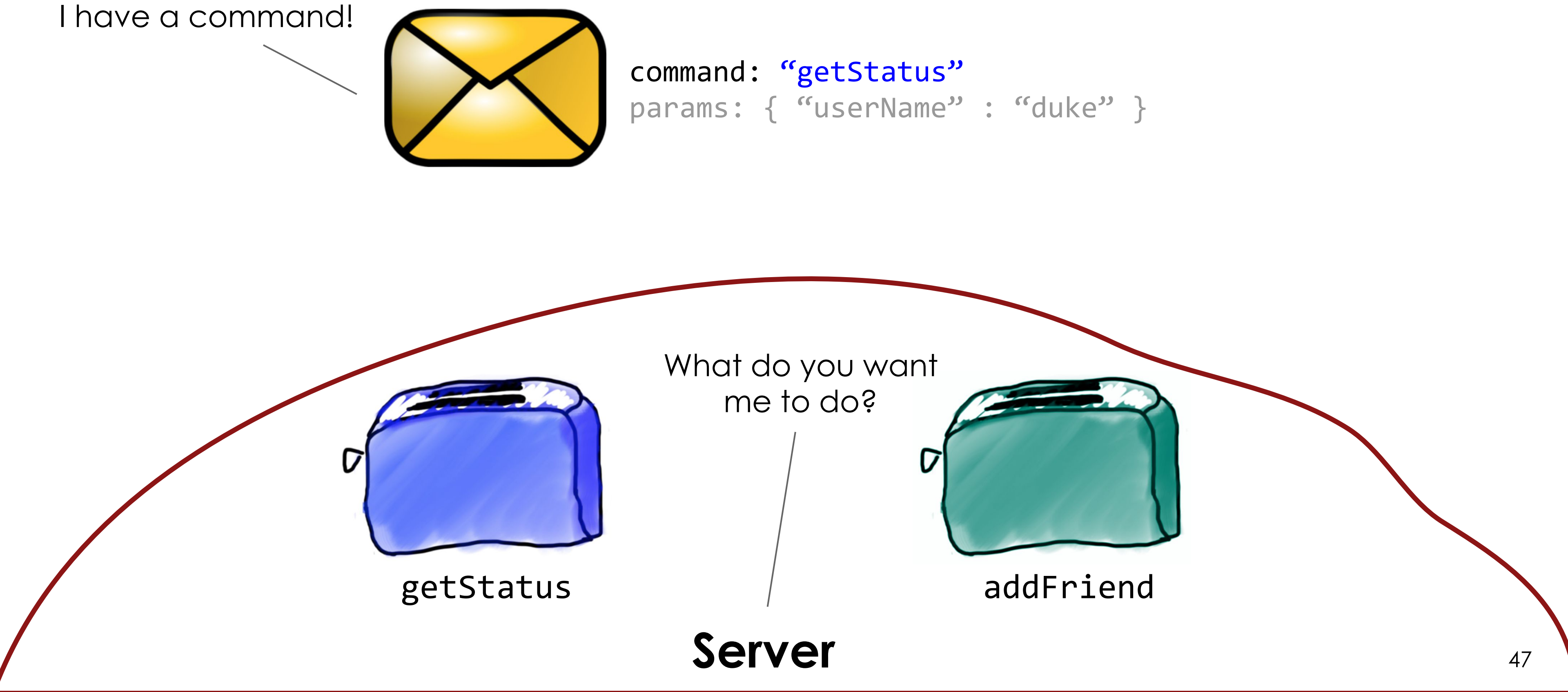


Server has a bunch of things it can do.

Requests are like Remote Method Calls



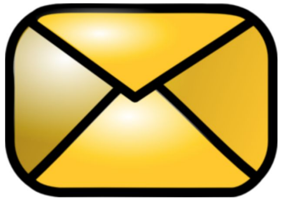
Requests are like Remote Method Calls



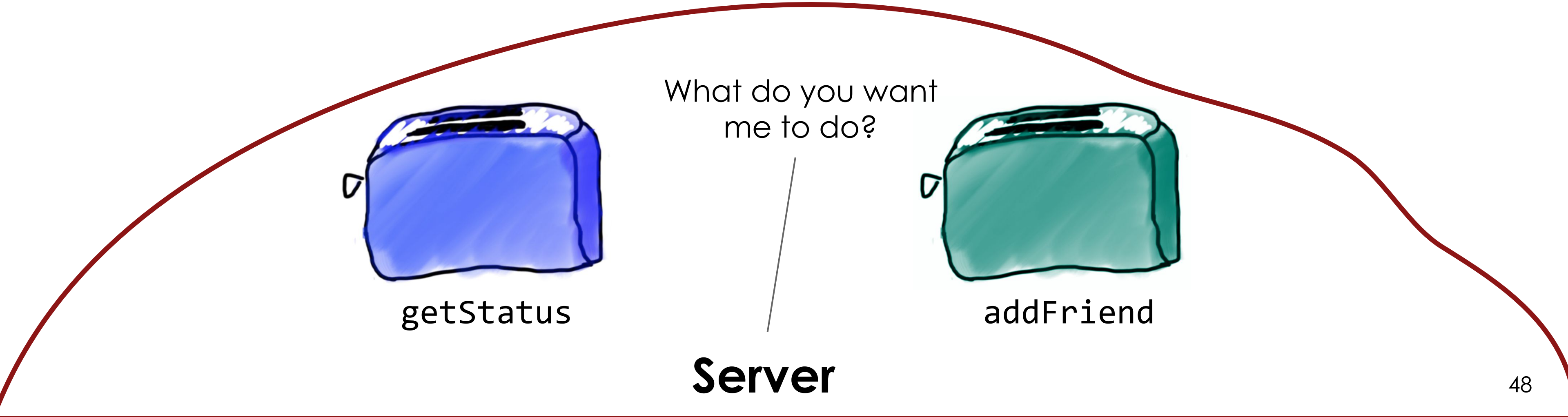
```
command: “getStatus”
params: { “userName” : “duke” }
```

Requests are like Remote Method Calls

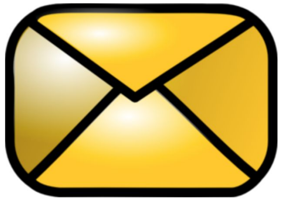
I have a command!



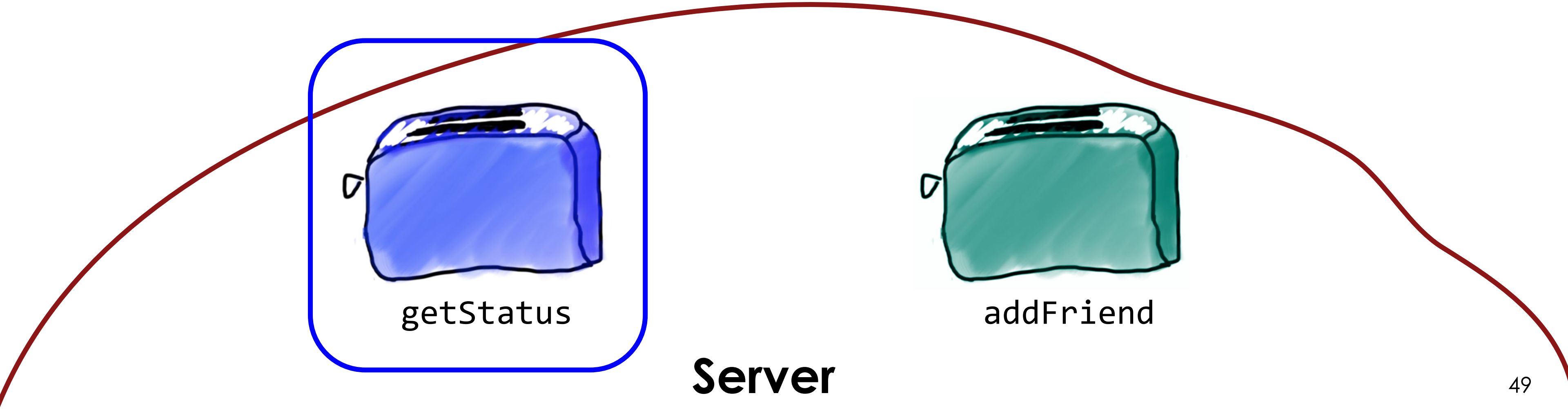
```
command: "getStatus"
params: { "userName" : "duke" }
```



Requests are like Remote Method Calls



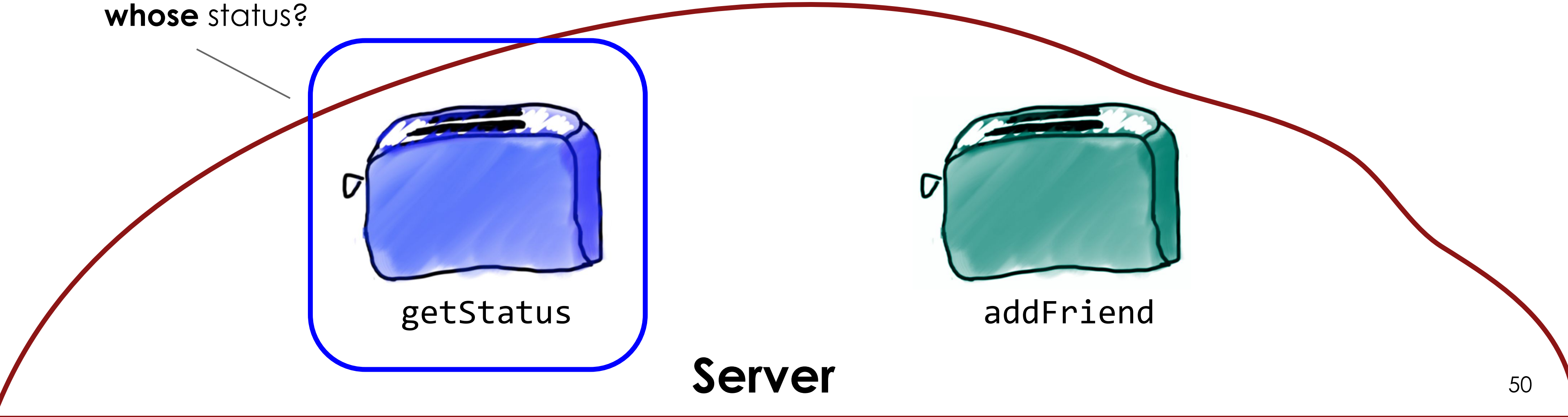
```
command: “getStatus”
params: { “userName” : “duke” }
```



Requests are like Remote Method Calls

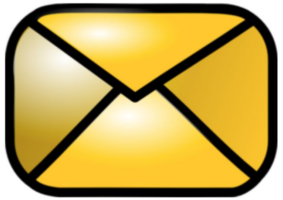
```
command: “getStatus”
params: { “userName” : “duke” }
```

I need a parameter:
whose status?



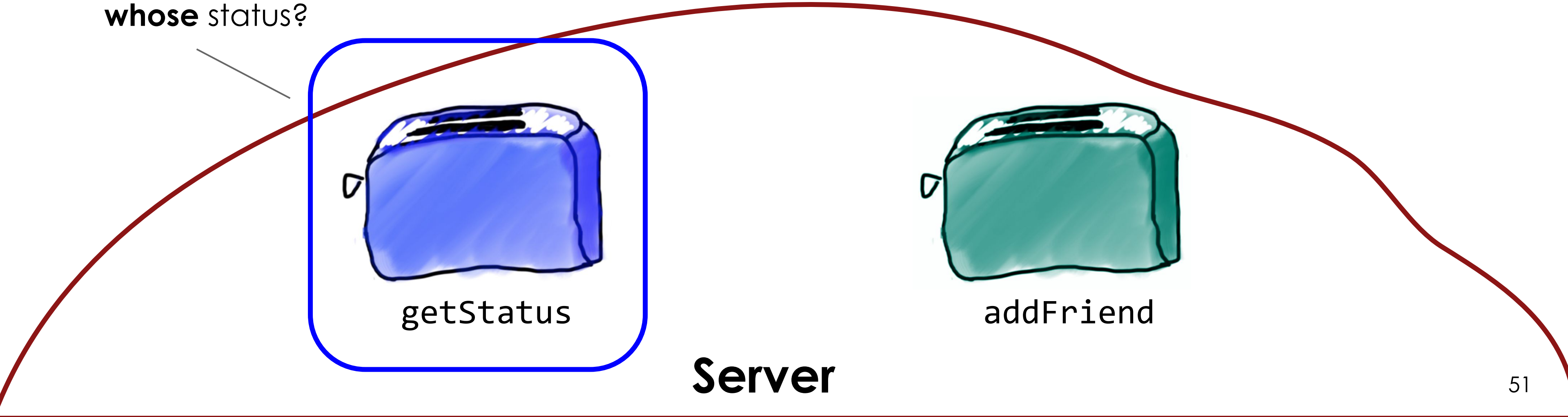
Requests are like Remote Method Calls

I have a parameter!

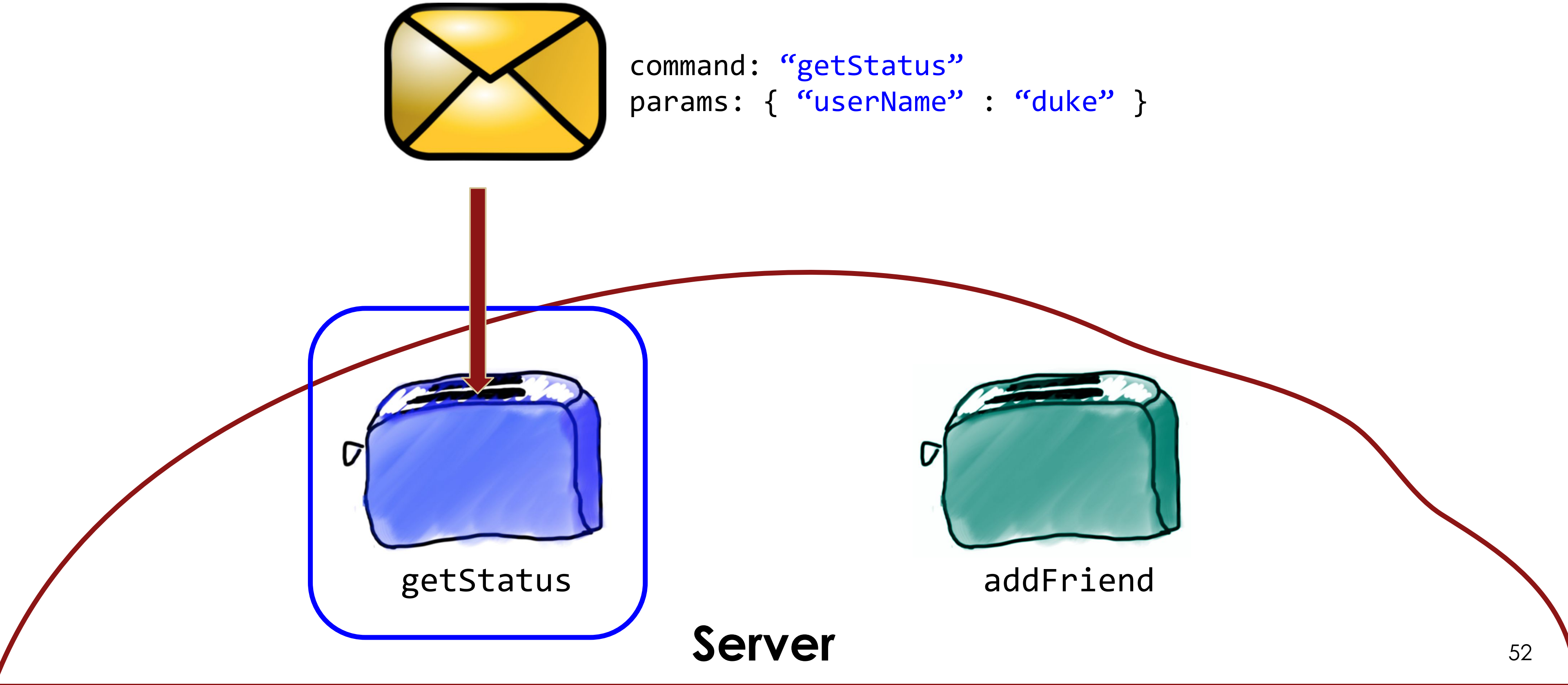


```
command: “getStatus”
params: { “userName” : “duke” }
```

I need a parameter:
whose status?

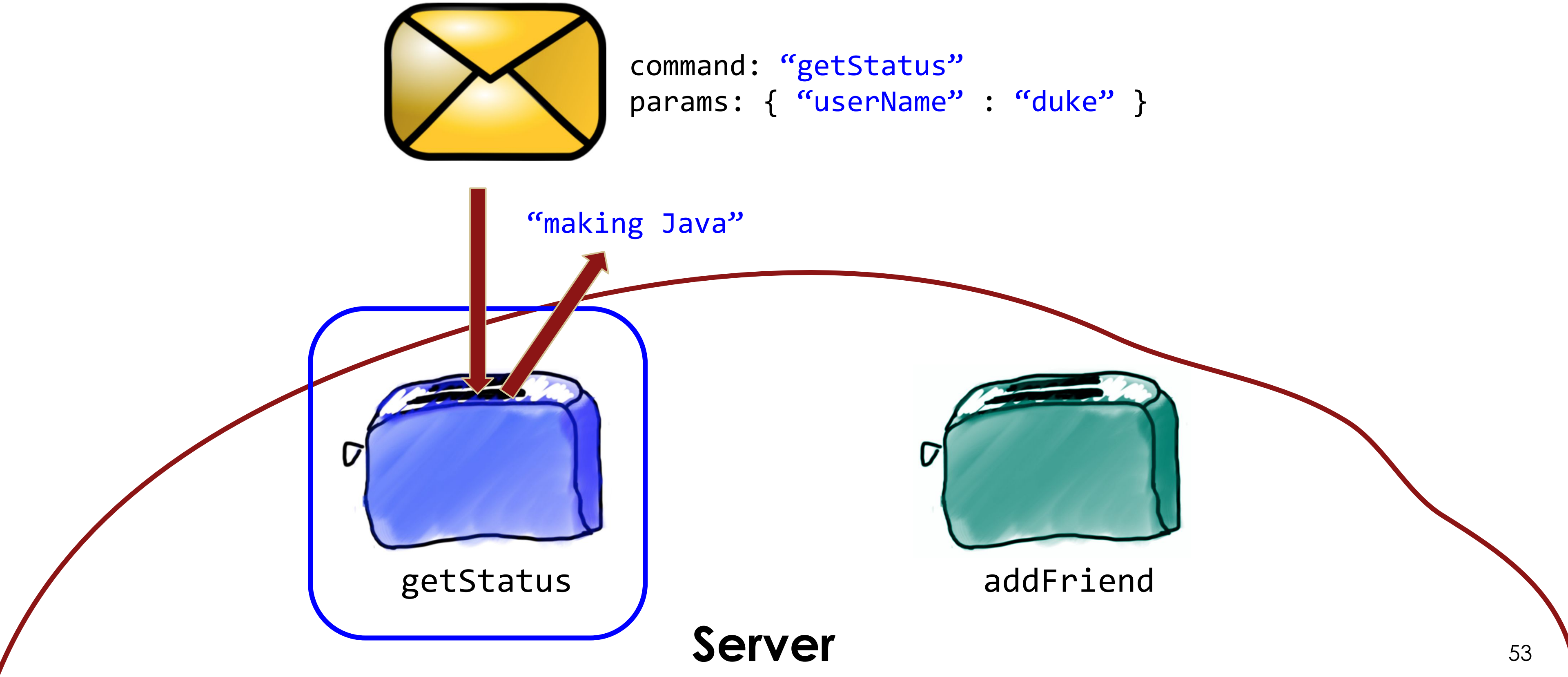


Requests are like Remote Method Calls



```
command: “getStatus”
params: { “userName” : “duke” }
```

Requests are like Remote Method Calls



Servers on one slide

1

```java
public String requestMade(Request request) {
    // server code goes here
}
```

2

```java
// make a Server object
private SimpleServer server = new SimpleServer(this, 8000);
```

3

```java
public void run(){
    // start the server
    server.start();
}
```

requestMade

Request request

```
command: “getStatus”
params: { “userName” : “duke” }
```

```
public String requestMade(Request request) {
     String cmd = request.getCommand();  // “getStatus”
     if(cmd.equals(“getStatus”)) {
          String user = request.getParam(“userName”); // “duke”
          String status = runGetStatus(user);         // “making Java”
          return status;
     }
     ...
}
```

Servers on one slide

1

```
public String requestMade(Request request) {
    // server code goes here
}
```

2

```
// make a Server object
private SimpleServer server = new SimpleServer(this, 8000);
```

3

```
public void run(){
    // start the server
    server.start();
}
```

Servers on one slide

1

```
public String requestMade(Request request) {
    // server code goes here
}
```

This is a port

2

```
// make a Server object
private SimpleServer server = new SimpleServer(this, 8000);
```

3

```
public void run(){
    // start the server
    server.start();
}
```

What is a Port?

45
39
35
33
29
23
17
9
3
PIER 39
FISHERMAN'S WHARF
NORTH BEACH
Coit Tower
Lombard Street
Palace of Fine Arts
EMBARCADERO
Chinatown San Francisco
FINANCIAL DISTRICT
Rincon Park
450 Sutter Building
Union Square
San Francisco Museum of Modern Art
The Warfield
SOUTH BEACH
SHN Orpheum Theatre
Oracle Park
SOMA
San Francisco Station
PRESIDIO OF SAN FRANCISCO
Lyon Street Steps
PRESIDIO HEIGHTS
LAUREL HEIGHTS
University of San Francisco
The Painted Ladies
Marina Blvd
Geary Blvd

Servers on one slide

1

```
public String requestMade(Request request) {
    // server code goes here
}
```

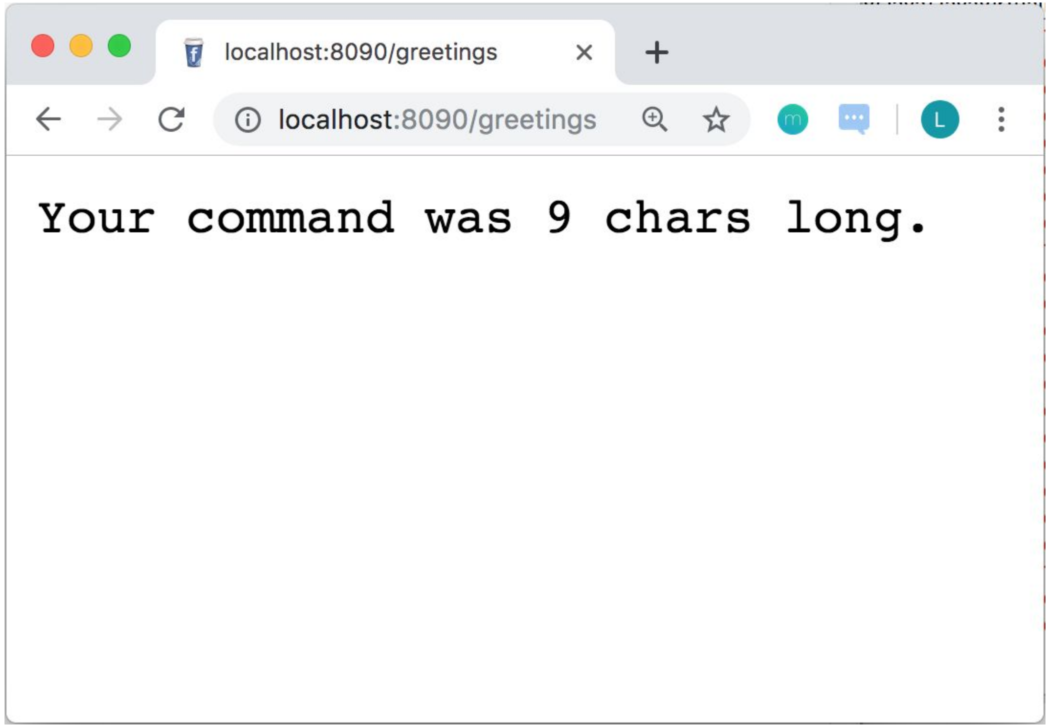
2

```
// make a Server object
private SimpleServer server = new SimpleServer(this, 8000);
```

3

```
public void run(){
    // start the server
    server.start();
}
```

Echo Server



```
Your command was 9 chars long.
```

Echo Server

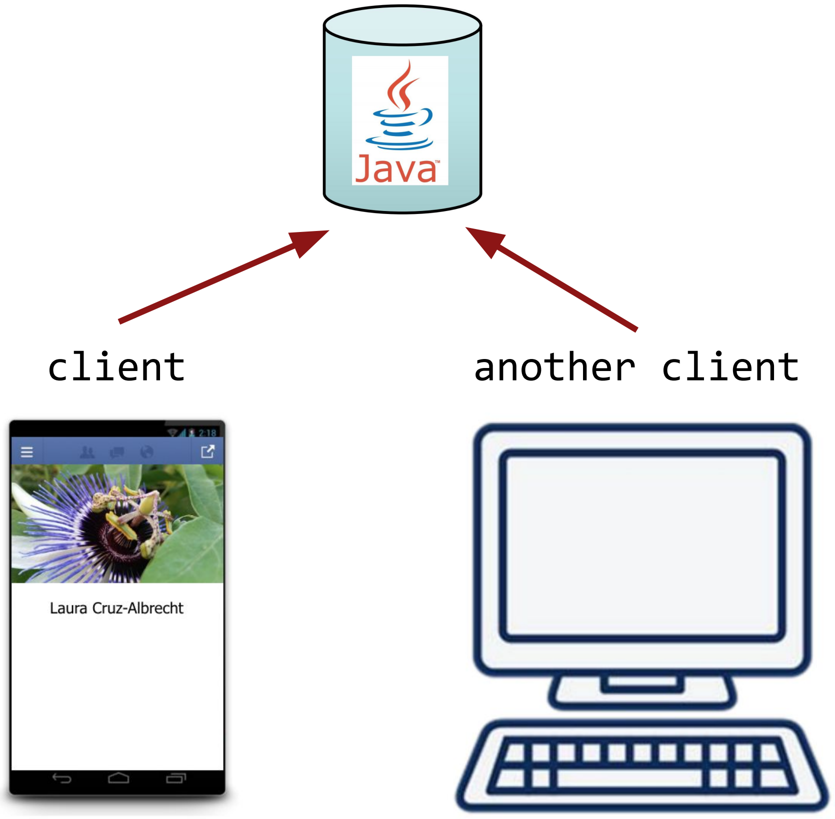
```
public class EchoServer extends ConsoleProgram implements SimpleServerListener{

    // 1. make a server object
    private SimpleServer server = new SimpleServer(this, 8080);

    public void run() {
        // 2. start the server
        server.start();
        println("Starting server...");
    }

    public void init() {
        setFont("Courier-24");
    }

    // 3. implement requestMade
    public String requestMade(Request request) {
        String cmd = request.getCommand();
        int len = cmd.length();
        return "Your command was " + len + " chars long.";
    }
}
```

There are two types of internet programs: $\boxed{\textbf{servers}}$ and **clients**.

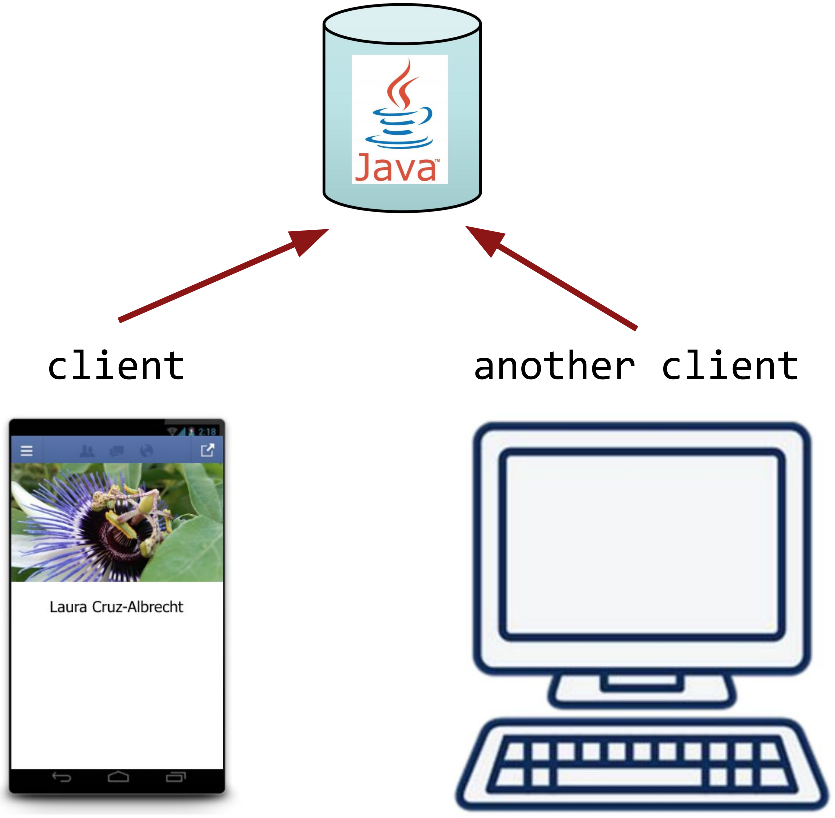
✓

There are two types of internet programs: **servers** and $\boxed{\textbf{clients}.}$

Clients



Clients



1. **Interact** with the user
2. **Get data** from its server
3. **Save data** to its server

Clients on one slide

```
try {
    // 1. construct a new request
    Request request = new Request("getStatus");

    // 2. add parameters to the request
    request.addParam("name", "duke");

    // 3. send the request to a computer on the internet
    String result = SimpleClient.makeRequest(HOST, request);

} catch(IOException e) {
    // The internet is a wild place
}
```

Clients on one slide

```
try {
    // 1. construct a new request
    Request request = new Request("getStatus");

    // 2. add parameters to the request
    request.addParam("name", "duke");

    // 3. send the request to a computer on the internet
    String result = SimpleClient.makeRequest(HOST, request);

} catch(IOException e) {
    // The internet is a wild place
}
```

Clients on one slide

```
try {
    // 1. construct a new request
    Request request = new Request("getStatus");

    // 2. add parameters to the request
    request.addParam("name", "duke");

    // 3. send the request to a computer on the internet
    String result = SimpleClient.makeRequest(HOST, request);

} catch(IOException e) {
    // The internet is a wild place
}
```

Clients on one slide

```
try {
    // 1. construct a new request
    Request request = new Request("getStatus");

    // 2. add parameters to the request
    request.addParam("name", "duke");

    // 3. send the request to a computer on the internet
    String result = SimpleClient.makeRequest(HOST, request);

} catch(IOException e) {
    // The internet is a wild place
}
```

Clients on one slide

```
try {
    // 1. construct a new request
    Request request = new Request("getStatus");

    // 2. add parameters to the request
    request.addParam("name", "duke");

    // 3. send the request to a computer on the internet
    String result = SimpleClient.makeRequest(HOST, request);

} catch(IOException e) {
    // The internet is a wild place
}
```

There are two types of internet programs: **servers** and $\boxed{\textbf{clients}.}$

✓

Plan for Today

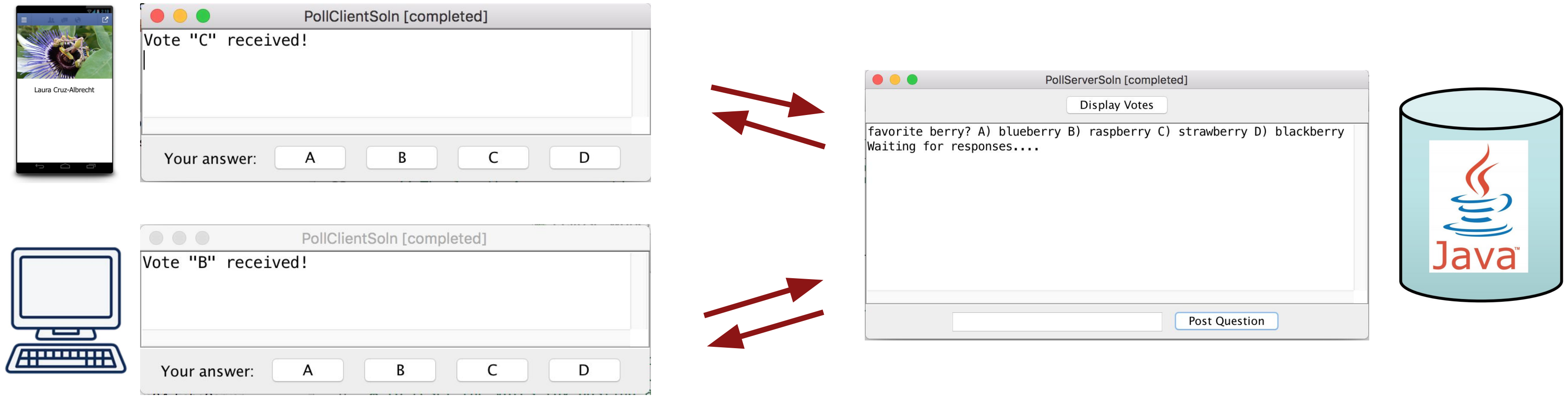
- Review: Interactors
- Internet 101
- Servers & Clients
- Practice: Polling

Polling

Let's write a program that lets users answer questions over the internet!

It will involve:

- 1 server: keeps track of the votes
- Multiple clients: anyone who wants to vote

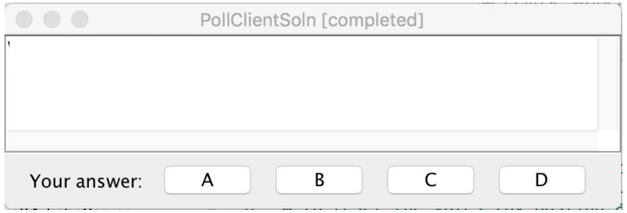


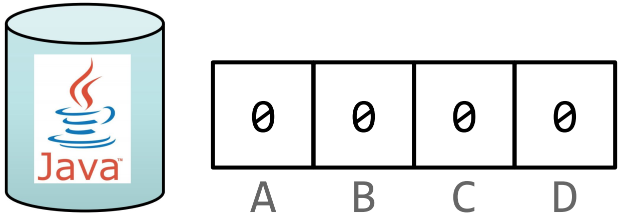
```
Vote "C" received!
```

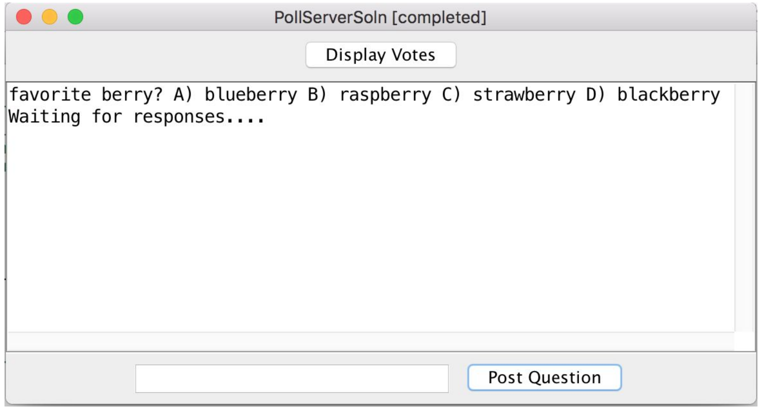
```
Vote "B" received!
```

```
favorite berry? A) blueberry B) raspberry C) strawberry D) blackberry
Waiting for responses....
```

Polling



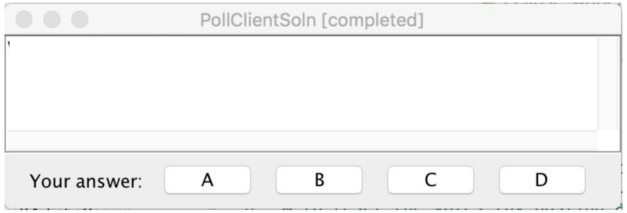


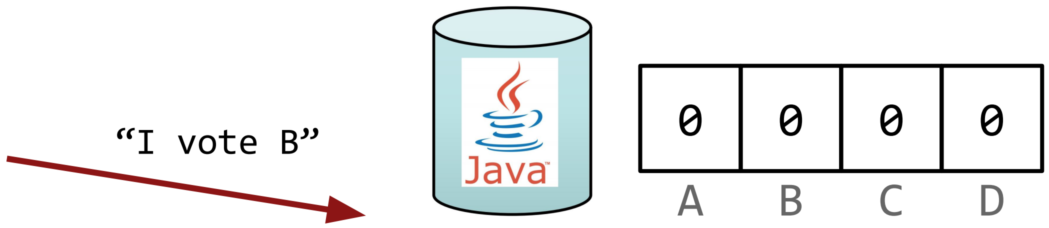


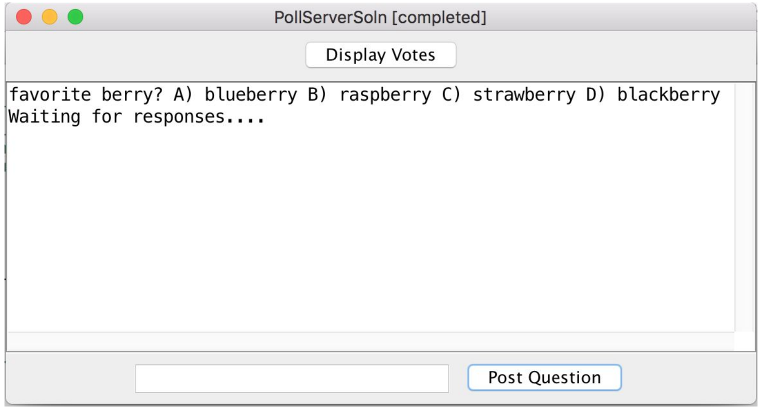
```
favorite berry? A) blueberry B) raspberry C) strawberry D) blackberry
Waiting for responses....
```

Clients **Server**

Polling



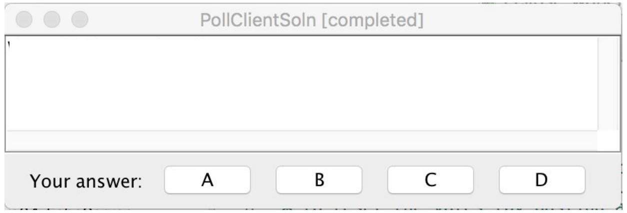


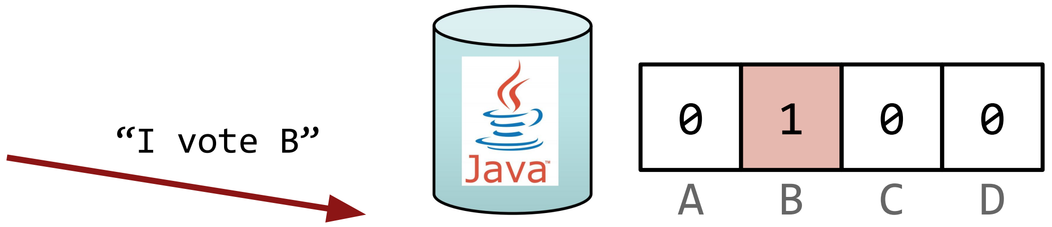


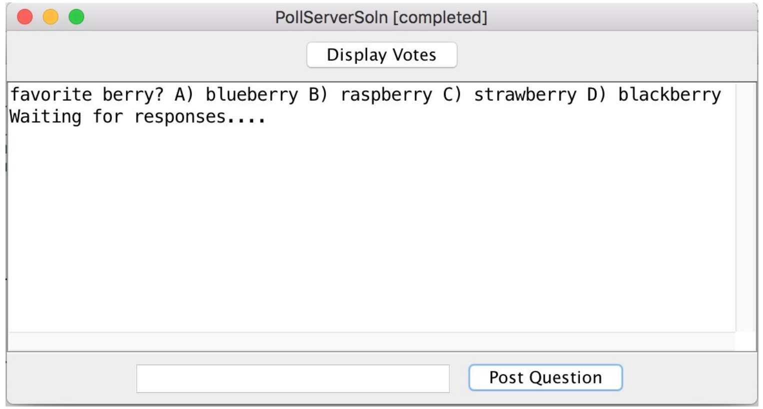
```
favorite berry? A) blueberry B) raspberry C) strawberry D) blackberry
Waiting for responses....
```

Clients **Server**

Polling



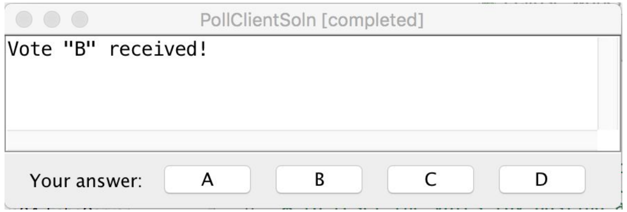




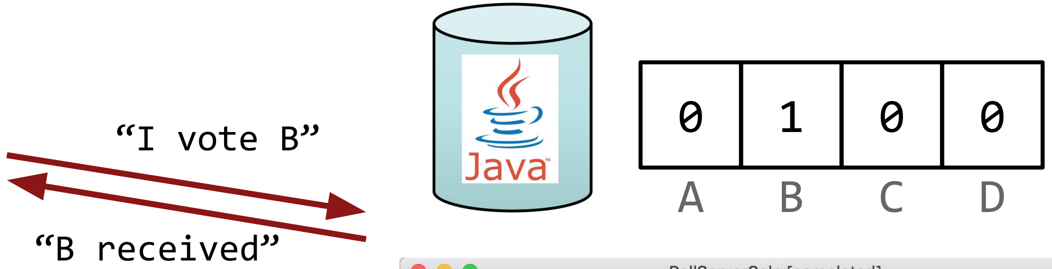
```
favorite berry? A) blueberry B) raspberry C) strawberry D) blackberry
Waiting for responses....
```

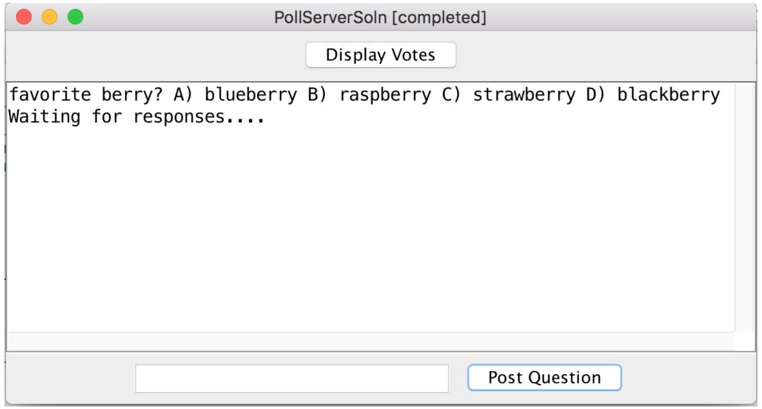
Clients **Server**

Polling



```
Vote "B" received!
```

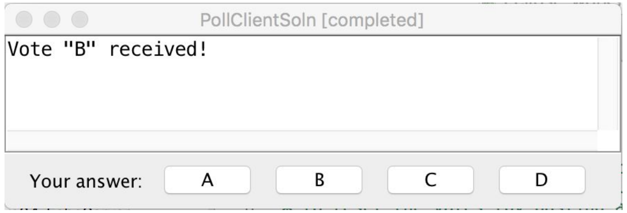




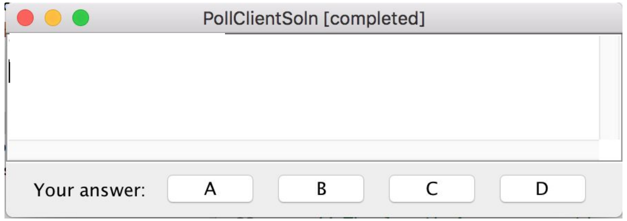
```
favorite berry? A) blueberry B) raspberry C) strawberry D) blackberry
Waiting for responses....
```

Clients **Server**

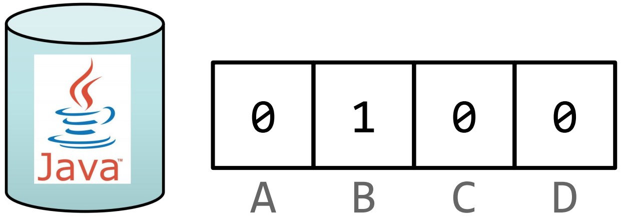
Polling

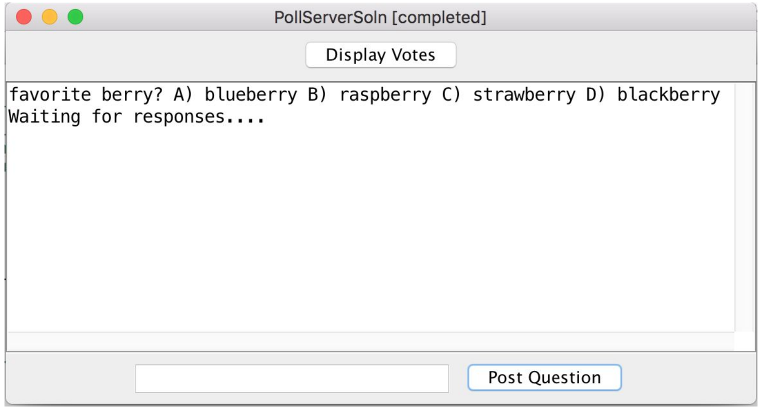

PollClientSoln [completed]

```
Vote "B" received!
```


PollClientSoln [completed]

“I vote C”

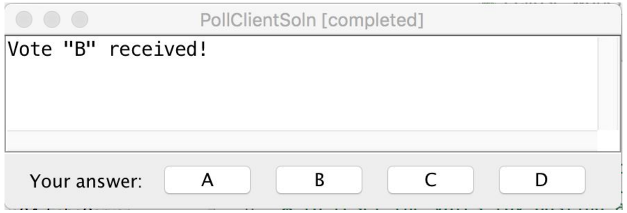



PollServerSoln [completed]

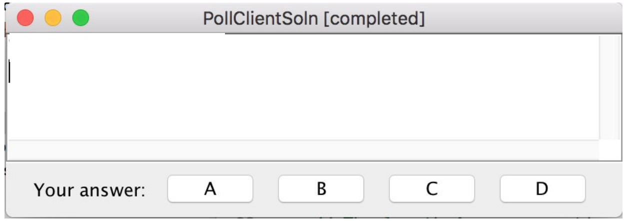
```
favorite berry? A) blueberry B) raspberry C) strawberry D) blackberry
Waiting for responses....
```

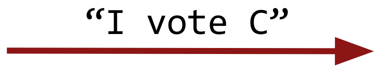
Clients **Server**

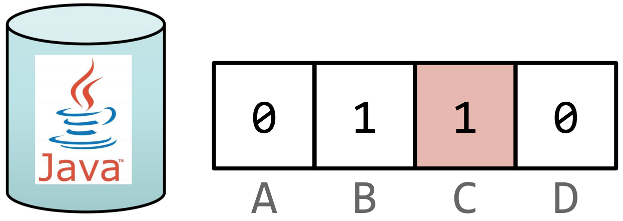
Polling

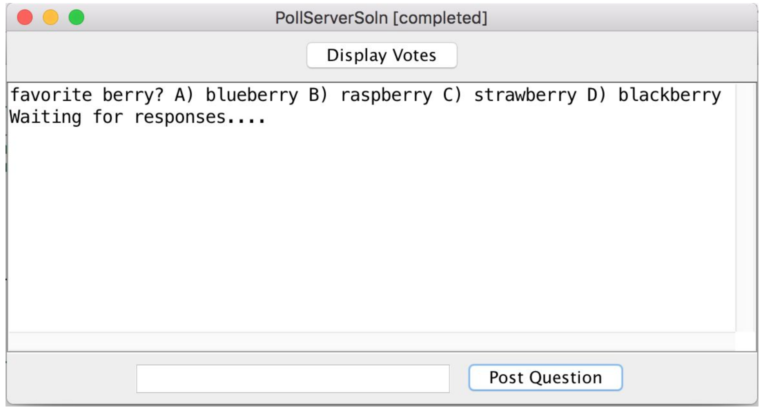


```
Vote "B" received!
```





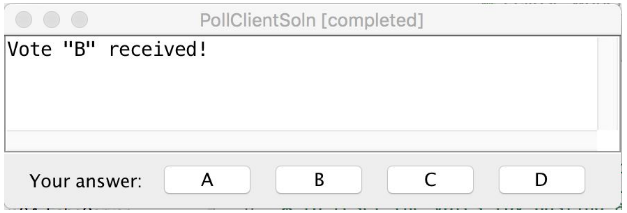




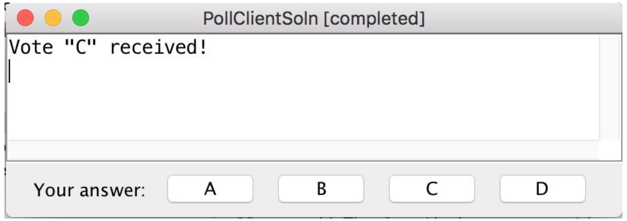
```
favorite berry? A) blueberry B) raspberry C) strawberry D) blackberry
Waiting for responses....
```

Clients **Server**

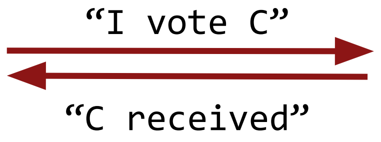
Polling

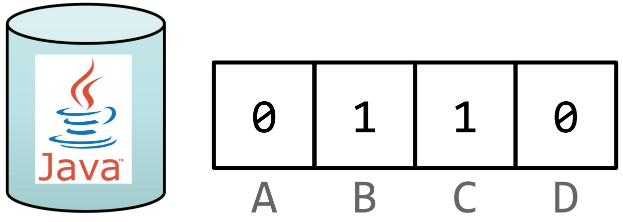


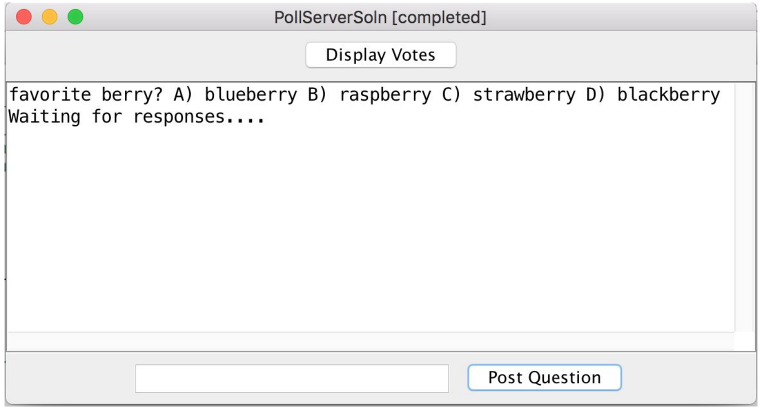
```
Vote "B" received!
```



```
Vote "C" received!
```



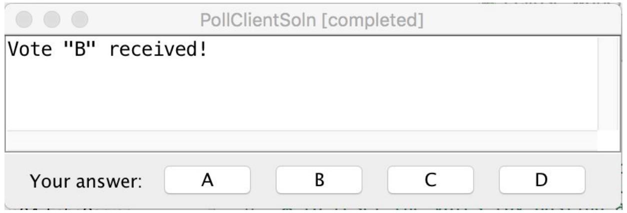




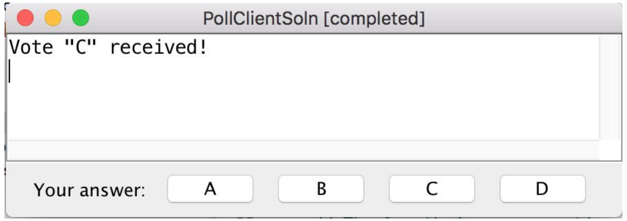
```
favorite berry? A) blueberry B) raspberry C) strawberry D) blackberry
Waiting for responses....
```

Clients **Server**

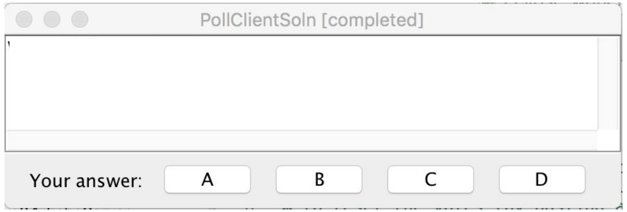
Polling



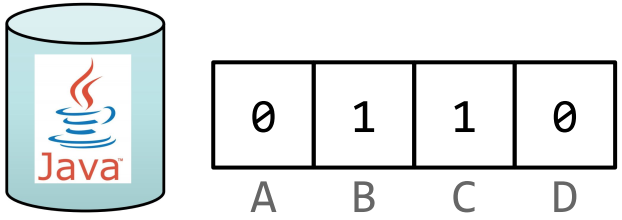
```
Vote "B" received!
```

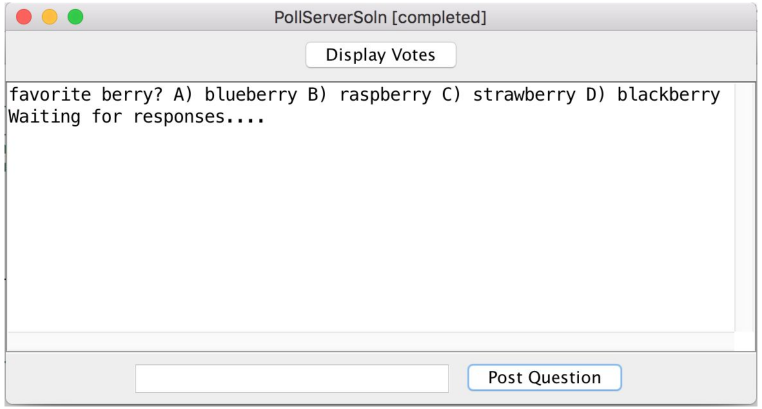


```
Vote "C" received!
```



“I vote B”

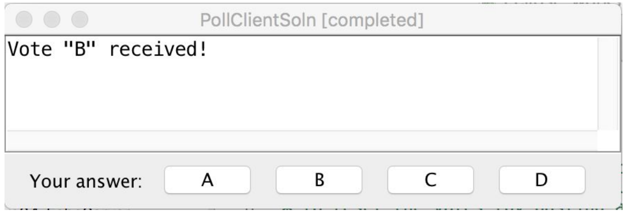




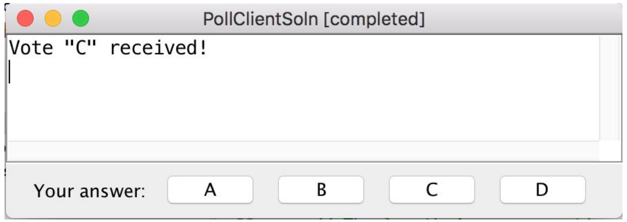
```
favorite berry? A) blueberry B) raspberry C) strawberry D) blackberry
Waiting for responses....
```

Clients **Server**

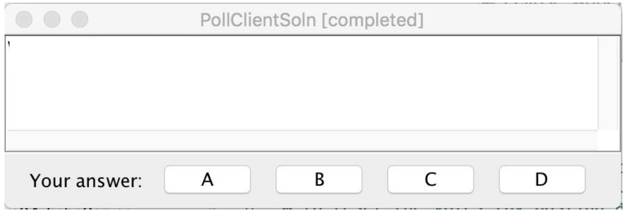
Polling

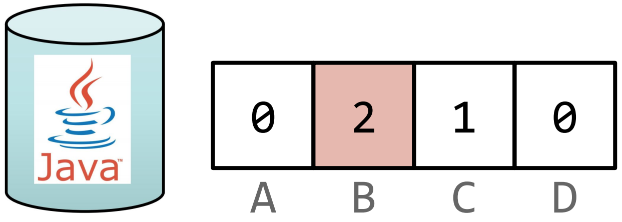


```
Vote "B" received!
```

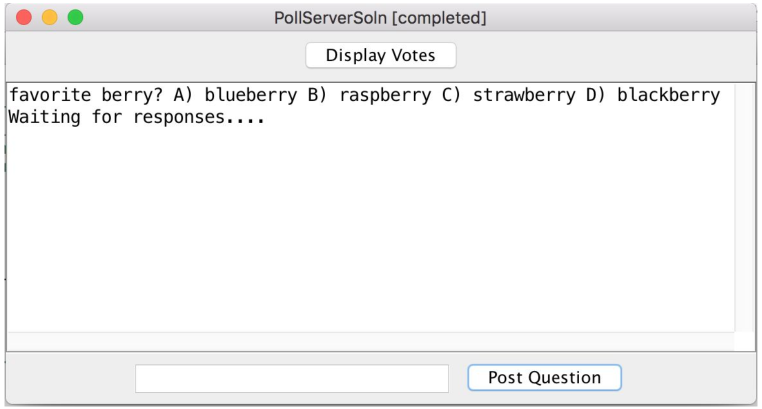


```
Vote "C" received!
```





“I vote B”

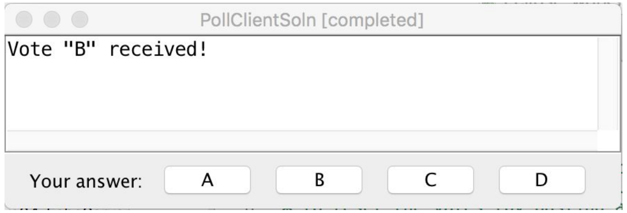


```
favorite berry? A) blueberry B) raspberry C) strawberry D) blackberry
Waiting for responses....
```

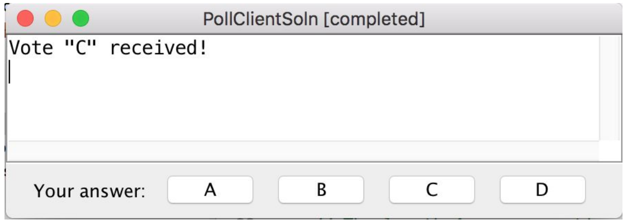
Clients

Server

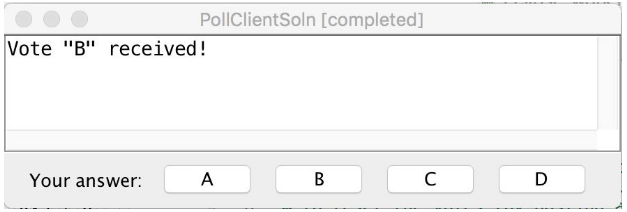
Polling



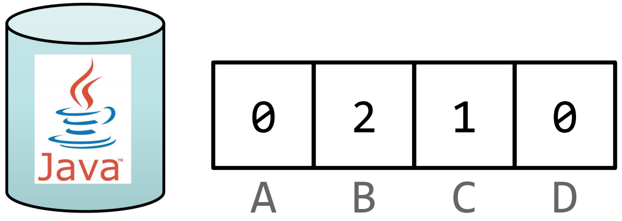
```
Vote "B" received!
```

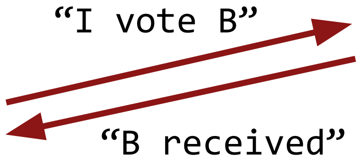


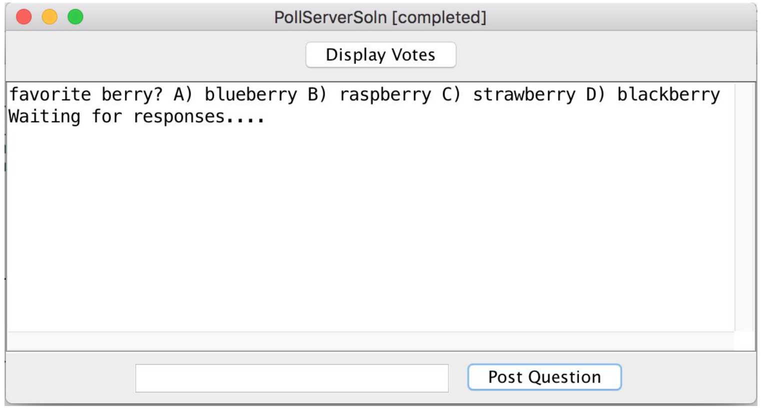
```
Vote "C" received!
```



```
Vote "B" received!
```



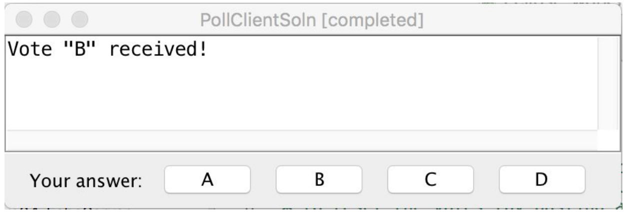




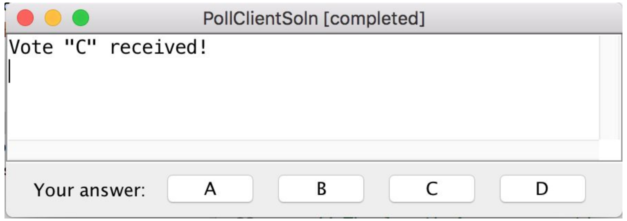
```
favorite berry? A) blueberry B) raspberry C) strawberry D) blackberry
Waiting for responses....
```

Clients **Server**

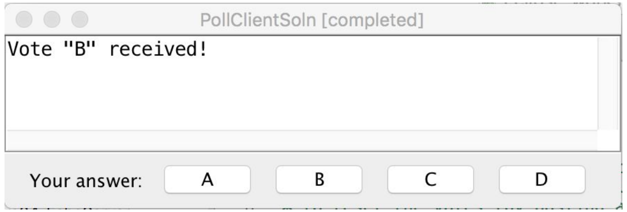
Polling



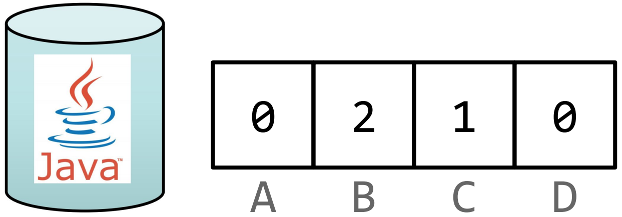
```
Vote "B" received!
```

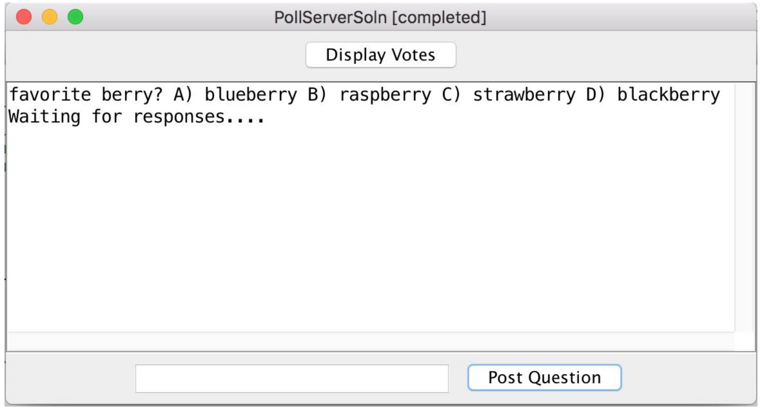


```
Vote "C" received!
```



```
Vote "B" received!
```



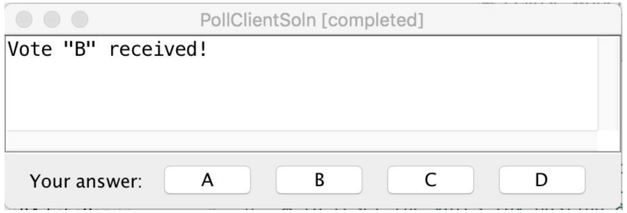


```
favorite berry? A) blueberry B) raspberry C) strawberry D) blackberry
Waiting for responses....
```

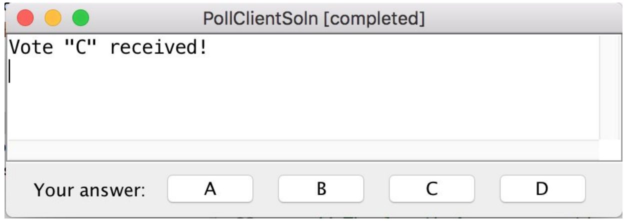
Clients

Server

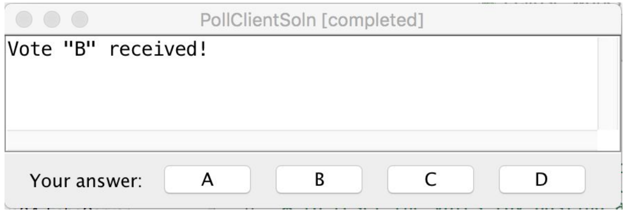
Polling



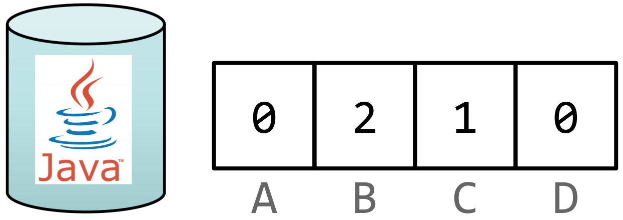
```
Vote "B" received!
```

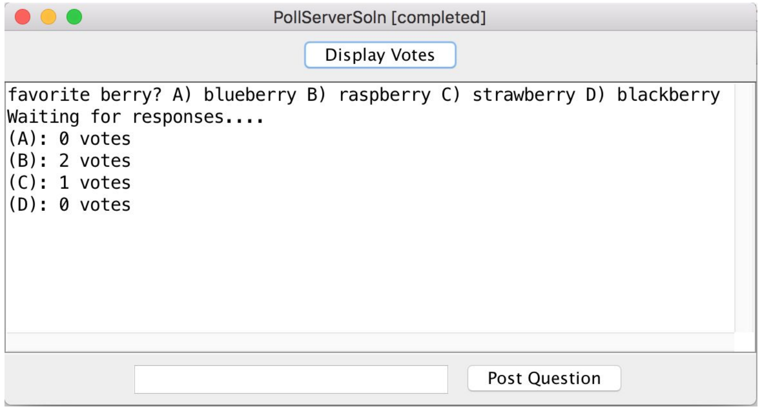


```
Vote "C" received!
```



```
Vote "B" received!
```





```
favorite berry? A) blueberry B) raspberry C) strawberry D) blackberry
Waiting for responses....
(A): 0 votes
(B): 2 votes
(C): 1 votes
(D): 0 votes
```

Clients **Server**

Let’s Code It!

Polling

```
public class PollServer extends ConsoleProgram implements SimpleServerListener {

    // 1. make a Server object
    // The Server object that notifies us when we receive a Request
    private SimpleServer server = new SimpleServer(this, 8080);

    // The text field where the user enters questions
    private JTextField textField;

    // The length-4 array counting the votes for A/B/C/D
    int[] votes = new int[4];

    public void init() {
        setFont("Courier-20");

        // 2. Start listening for requests
        server.start();

        // Add interactors
        add(new JButton("Display Votes"), NORTH);
        textField = new JTextField(20);
        add(textField, SOUTH);
        add(new JButton("Post Question"), SOUTH);
        addActionListeners();
    }

    public void actionPerformed(ActionEvent event) {
        if (event.getActionCommand().equals("Display Votes")) {
            // Display the votes for A/B/C/D
            for (int i = 0; i < votes.length; i++) {
                char currentAnswer = (char)('A' + i);
                println("(" + currentAnswer + "): " + votes[i] + " votes");
            }
        } else if (event.getActionCommand().equals("Post Question")) {
            // Clear the screen and the vote counts for the new question
            clearConsole();
            votes = new int[4];
            println(textField.getText());
            println("Waiting for responses....");
            textField.setText("");
        }
    }

    // 3. Implement requestMade
    // This method is called whenever a request is received.
    public String requestMade(Request request) {
        if (request.getCommand().equals("vote")) {
            // Add one to our array of vote counts for their vote
            String vote = request.getParam("answer");
            votes[vote.charAt(0) - 'A']++;
            return "Vote \"" + vote + "\" received!";
        } else {
            return "Unknown command.";
        }
    }
}
```

Polling

```
public class PollClient extends ConsoleProgram {

    // The URL where the host program is running
    private static final String HOST = "http://localhost:8080";

    public void init() {
        setFont("Courier-24");

        // Add interactors
        add(new JLabel("Your answer: "), SOUTH);
        add(new JButton("A"), SOUTH);
        add(new JButton("B"), SOUTH);
        add(new JButton("C"), SOUTH);
        add(new JButton("D"), SOUTH);
        addActionListeners();
    }

    public void actionPerformed(ActionEvent event) {
        // When the user clicks a button, send a new Request with our vote.
        try {
            Request request = new Request("vote");
            String answerStr = event.getActionCommand();
            request.addParam("answer", answerStr);
            String response = SimpleClient.makeRequest(HOST, request);
            println(response);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Plan for Today

- Review: Interactors
- Internet 101
- Servers & Clients
- Practice: Polling

Next Time: How to start your own Java project