CS 5010: PDP

Lecture 11: Networks

CS 5010
Fall 2017

Seattle

Adrienne Slaughter, Ph.D.

ahslaughter@northeastern.edu

Agenda

- Networking

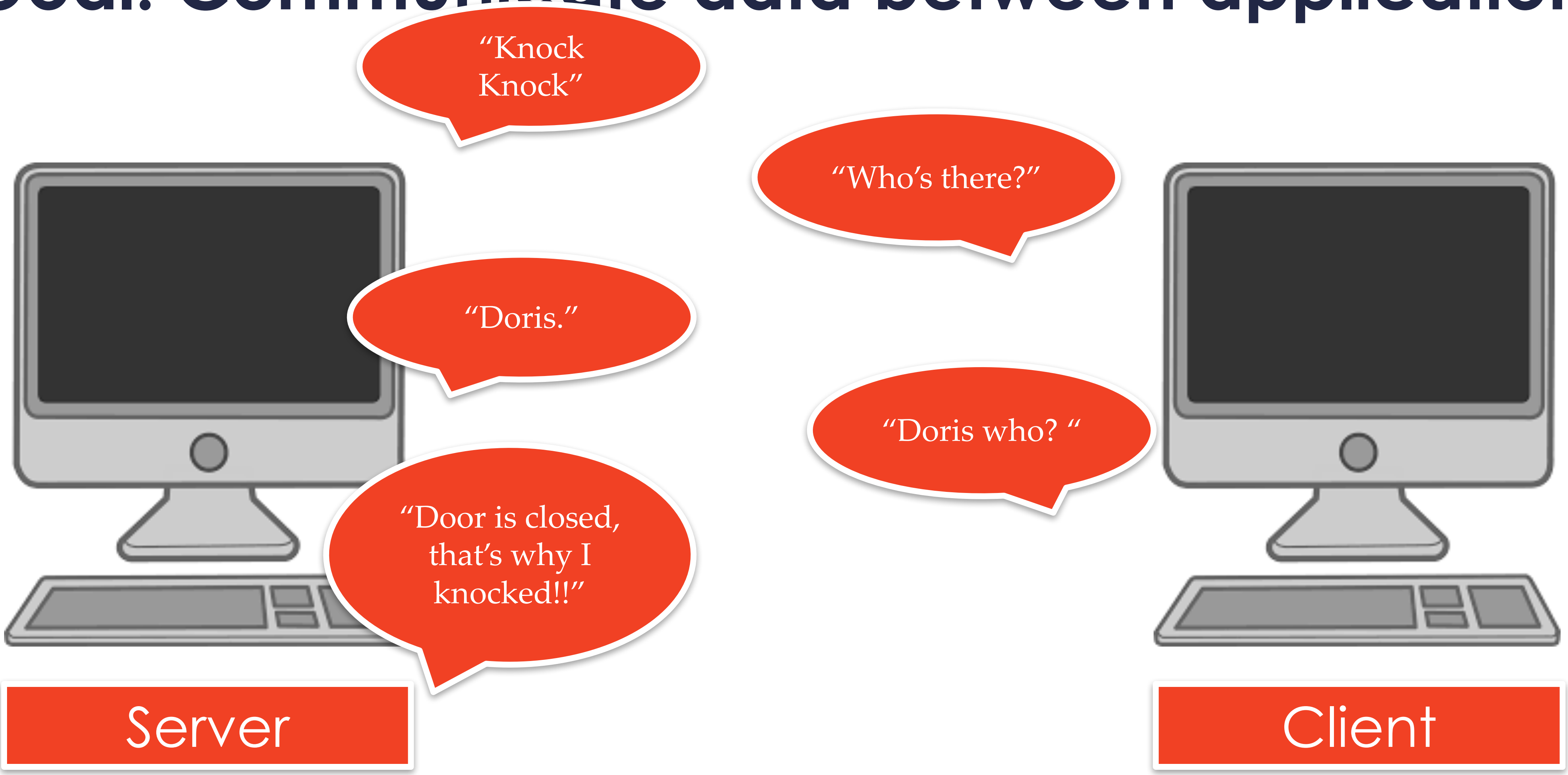
INTRODUCTION

Goal: Communicate data between applications

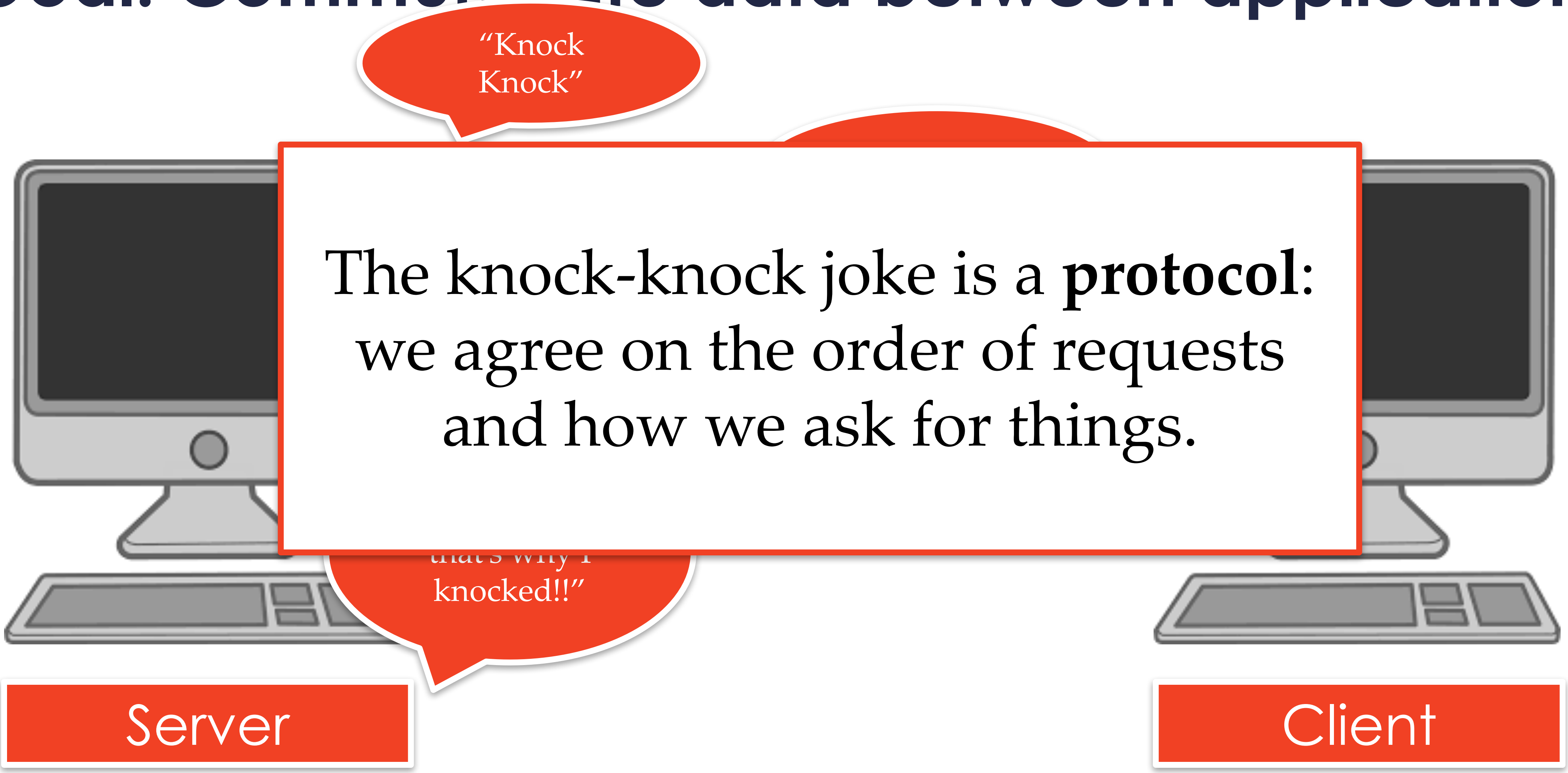
Server

Client

Goal: Communicate data between applications

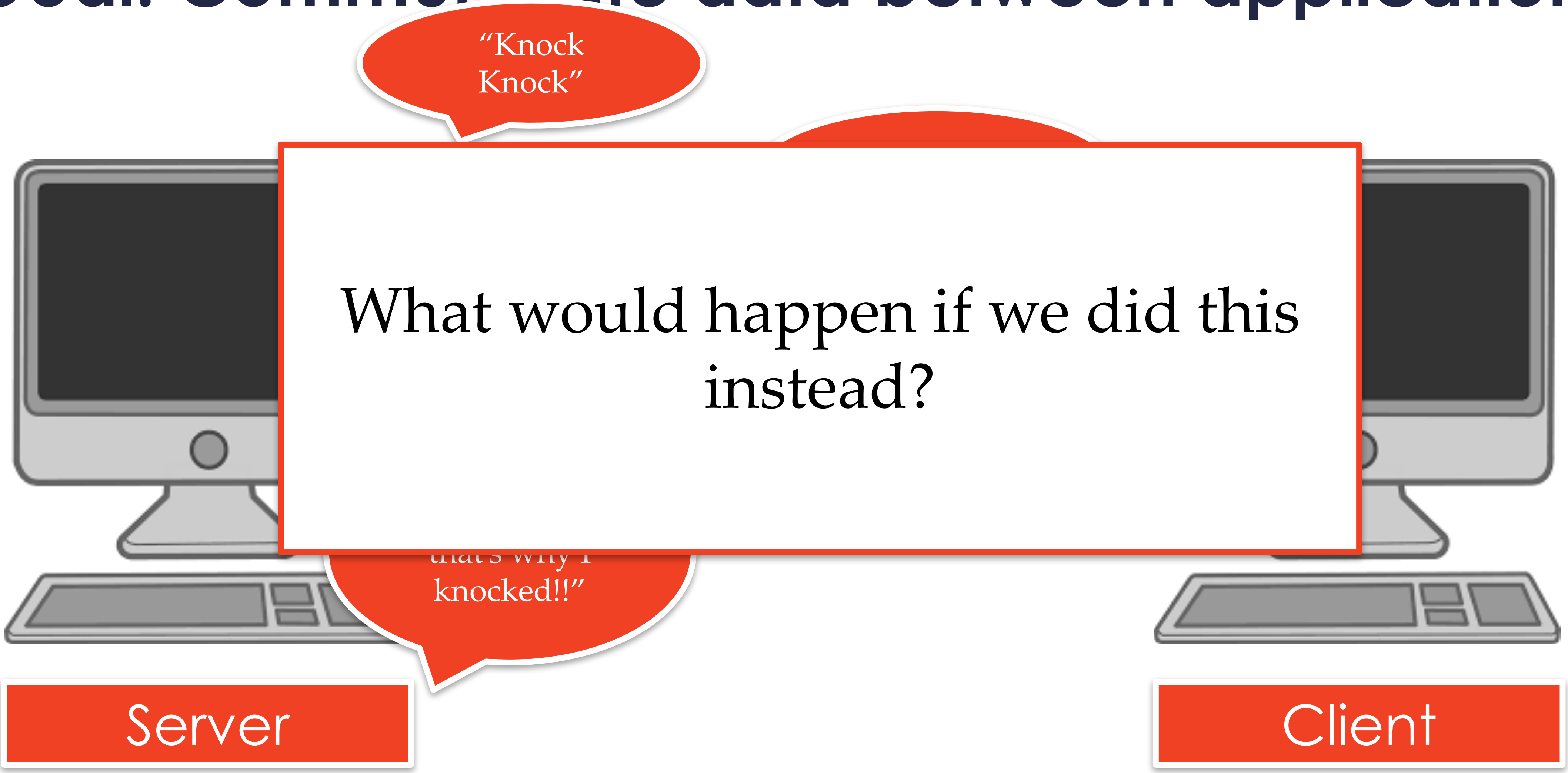


Goal: Communicate data between applications



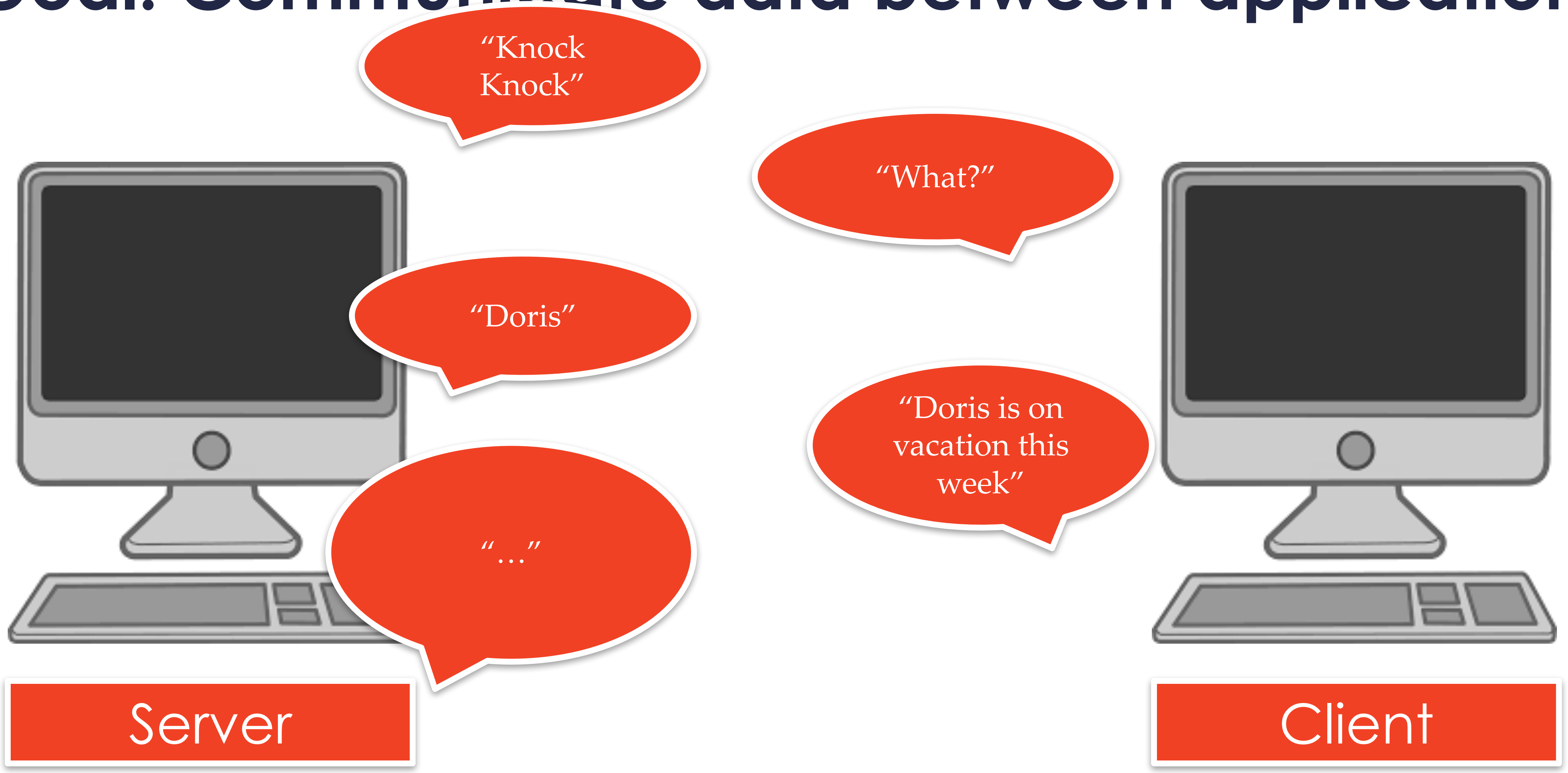
The knock-knock joke is a **protocol**: we agree on the order of requests and how we ask for things.

Goal: Communicate data between applications

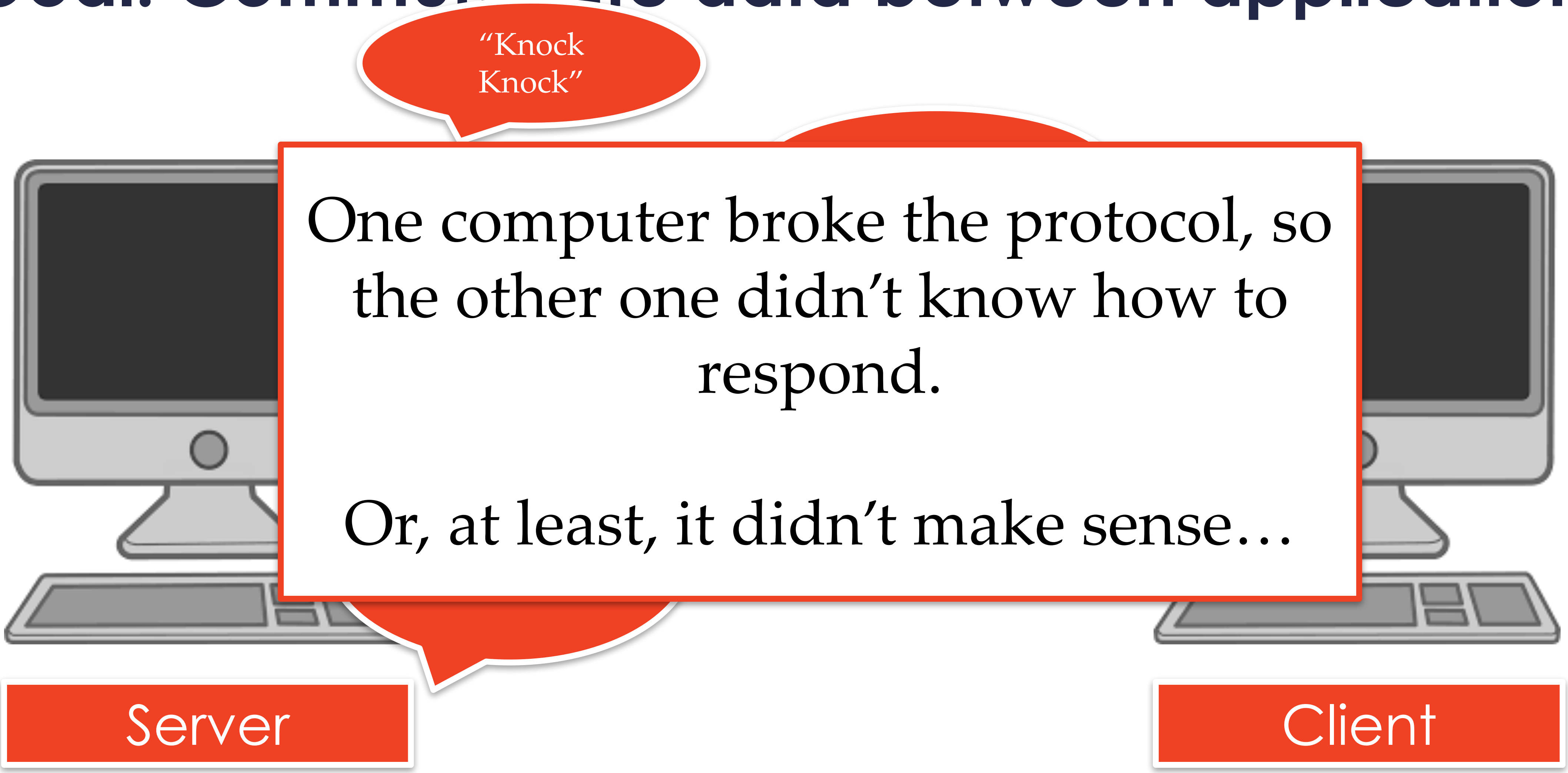


What would happen if we did this instead?

Goal: Communicate data between applications



Goal: Communicate data between applications



One computer broke the protocol, so the other one didn't know how to respond.

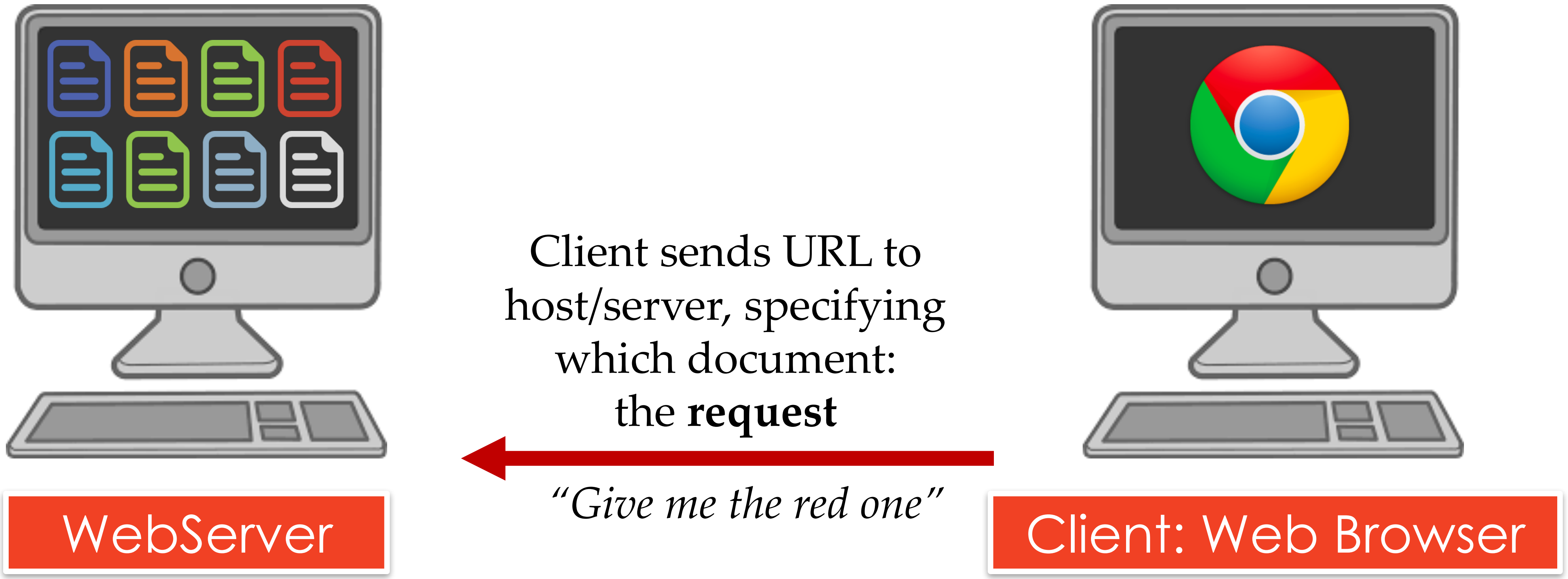
Or, at least, it didn't make sense…

Consider the web:

WebServer

Client: Web Browser

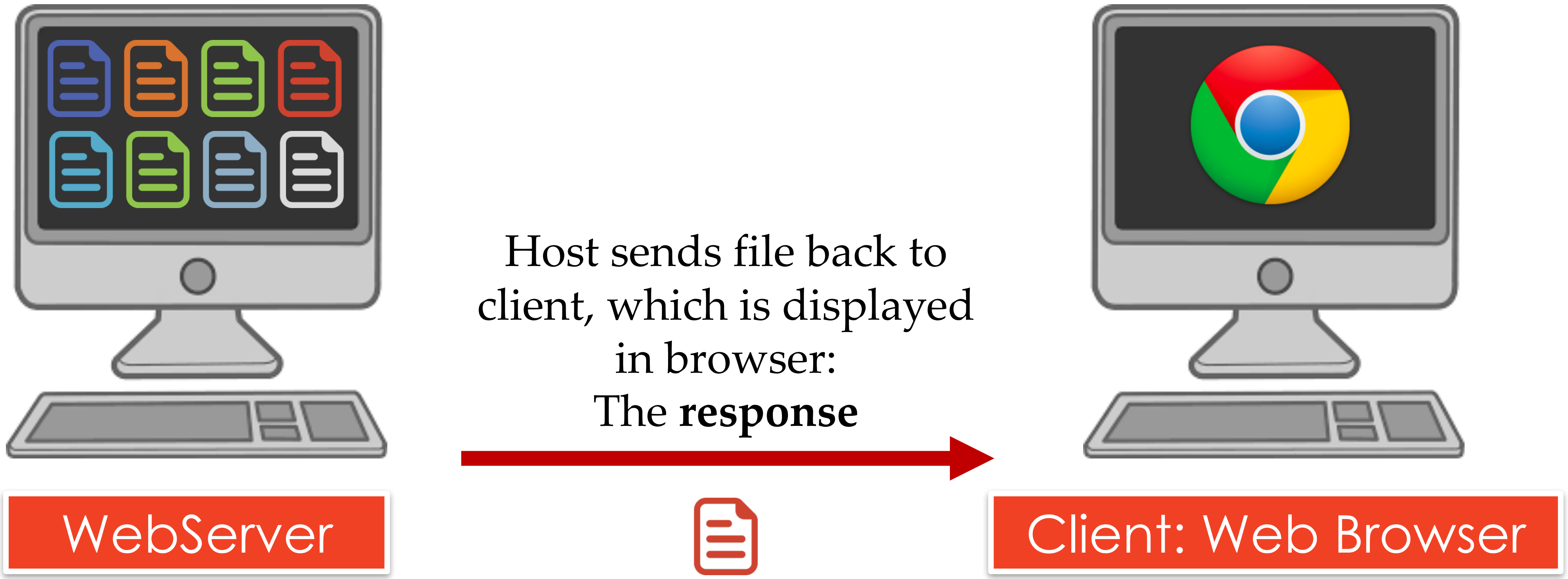
Consider the web:



Client sends URL to host/server, specifying which document: the **request**

"Give me the red one"

Consider the web:



Host sends file back to client, which is displayed in browser:
The **response**

Consider the web:

WebServer

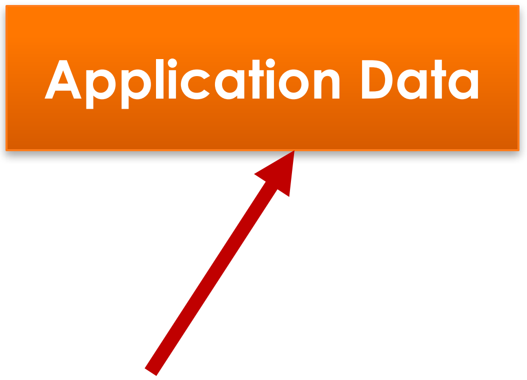
This works because the server and client agree to use the same protocol: HTTP

Client: Web Browser

HTTP

- **HyperText Transfer Protocol**
- Consists of 2 basic messages:
  - Request
  - Response
- Each of the request/response consists of **headers**

But how does the data get transferred?



All that HTTP stuff is just **Application Data**– data that 2 applications (the web server and web browser) use to communicate.

But how does the data get transferred?

Application Data

How do we actually connect to machines and transfer data?

WebServer

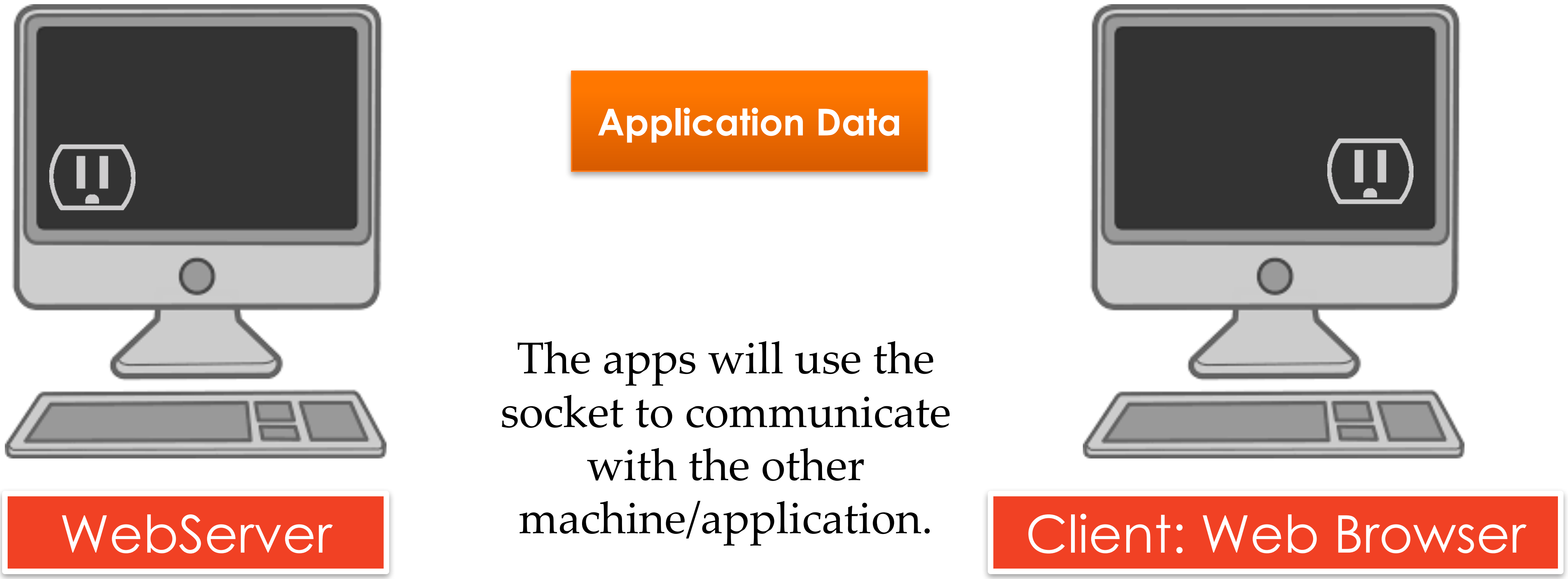
Client: Web Browser

But how does the data get transferred?

Application Data

First, we open a **socket** on each machine

But how does the data get transferred?



The apps will use the socket to communicate with the other machine/application.

But how does the data get transferred?

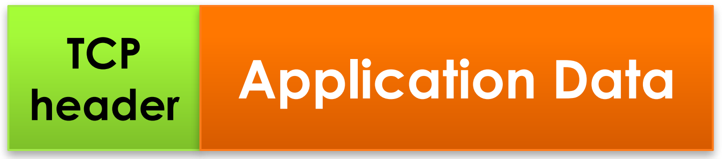
Application Data

The apps will use the socket to communicate with the other machine/application.

WebServer

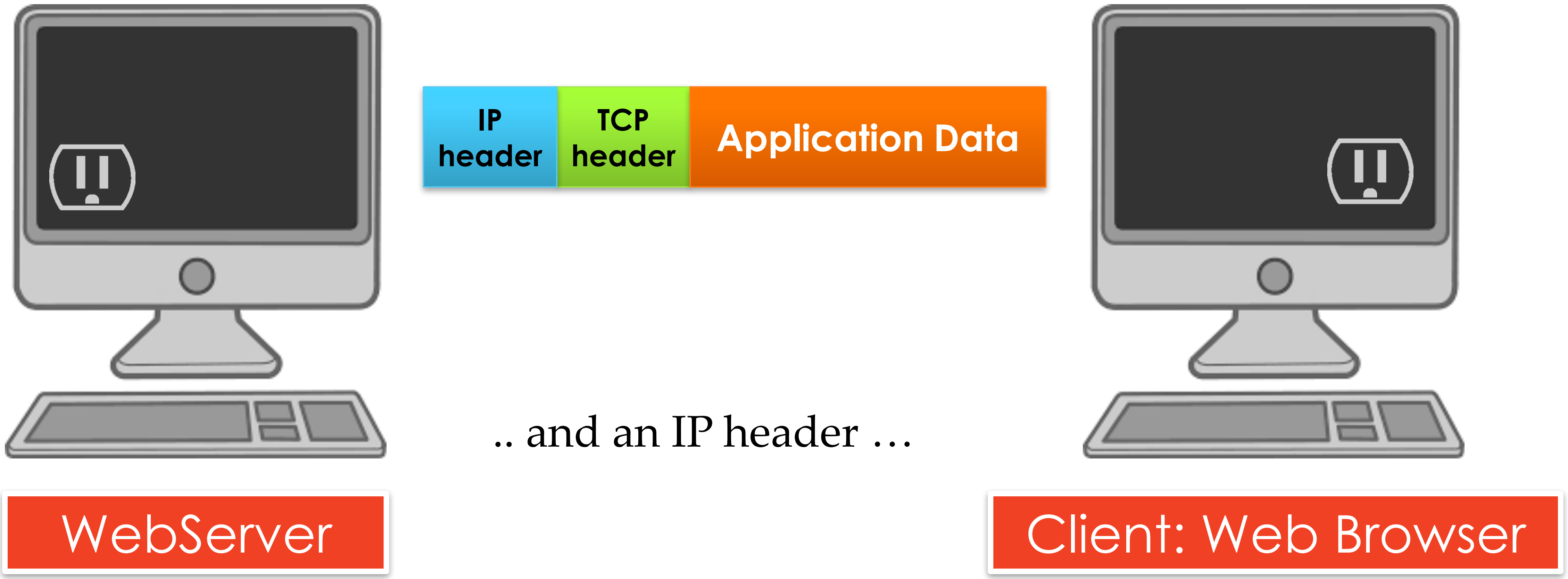
Client: Web Browser

But how does the data get transferred?



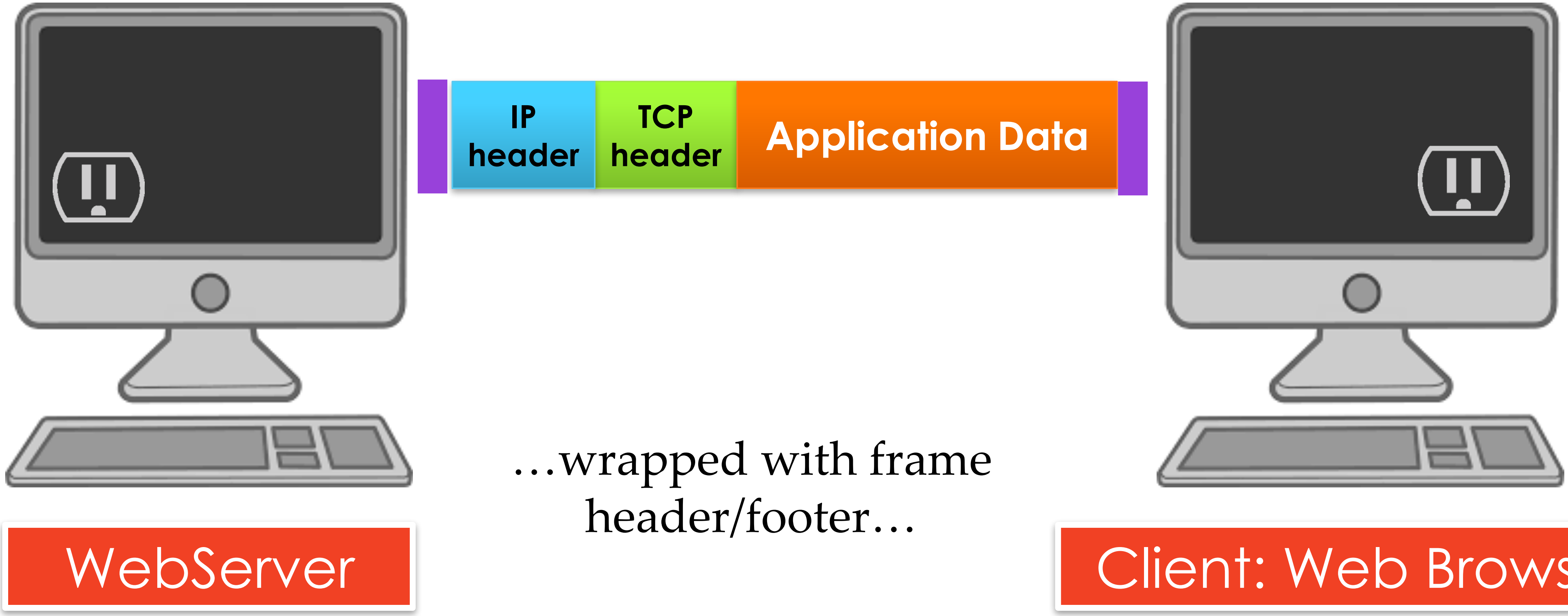
The application data gets a TCP header added to it…

But how does the data get transferred?



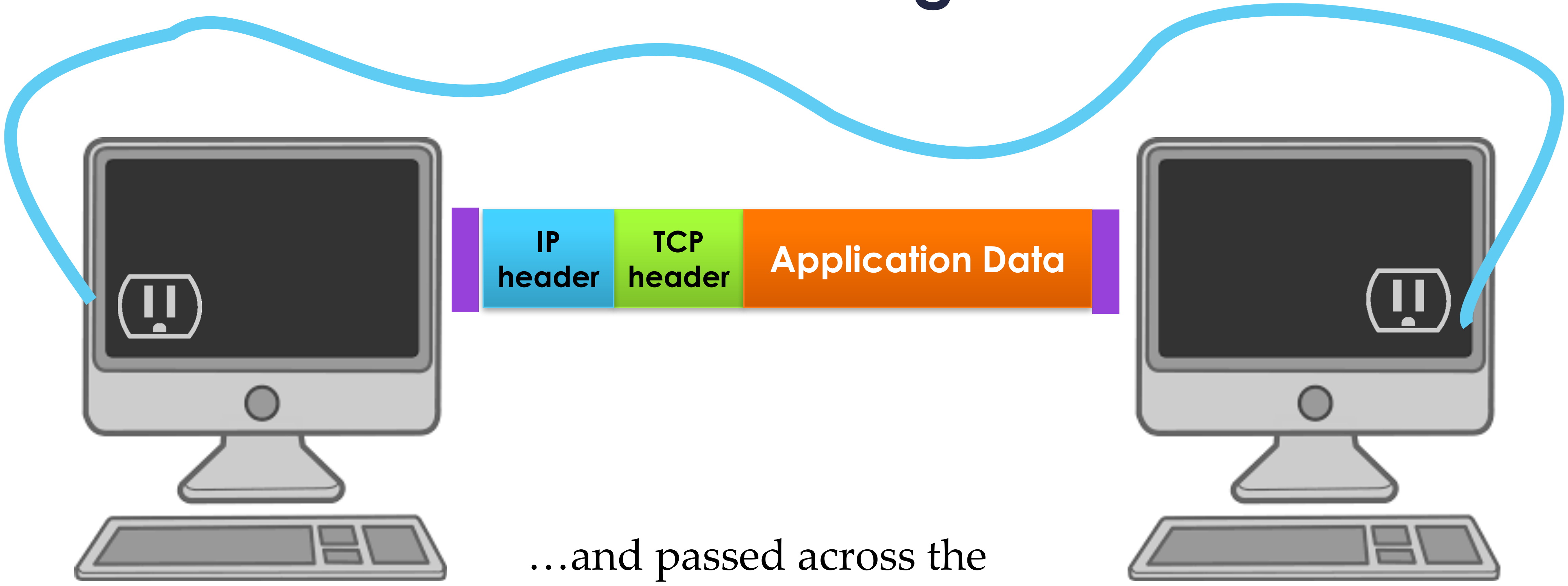
.. and an IP header …

But how does the data get transferred?



…wrapped with frame header/footer…

But how does the data get transferred?



…and passed across the wire.

WebServer

Client: Web Browser

What pieces do we need to worry about?

i.e., lecture objectives

- Naming of network resources
  - How to specify which computer you want to connect to
- Sockets
  - How to allow your computer to talk directly to another computer
- Communication protocols
  - Agreeing on the communication
- HTTP connections
  - Because the web.
- JSON
  - Also, the web.

DOING THIS IN JAVA

Networking Concepts/Issues/Goals

- **Naming:** How to find the computer/host you want to connect to
- **Transfer:** The actual connection
- **Communicating:** Sending data back and forth in a way that both the client and host/server understand

The General Process

- Open a socket.
- Open an input stream and output stream to the socket.
- Read from and write to the stream according to the server's protocol.
- Close the streams.
- Close the socket.

The General Process

Naming
- Open a socket.

Transfer
- Open an input stream and output stream to the socket.

Communicating
- Read from and write to the stream according to the server's protocol.
- Close the streams.
- Close the socket.

Relevant Terminology

- Client
- Server
- Socket: abstraction through which an application may send and receive data
- Port
- DNS
- TCP/IP
- Session

Relevant Terminology

- Client
- Server
- Socket: ab tra cation may send and
- Port
- DNS
- TCP/IP
- Session

}

We know these from our introduction example.

We know the first few terms from our introductory example.

More definitions

- **DNS:** Domain Name System. Translates "http://www.northeastern.edu" into the "Internet Address".
  - It's the difference between going to "Ian's House", and the actual street address. When you ask DNS for the address to Ian's House, it's gives you the street address.
- **TCP/IP:** Transfer Control Protocol and Internet Protocol.
  - Used to break the application data into small pieces to be sent across the wire between the client and server. See the end of this lecture for more details.
- **Session:** A "conversation" between two computers.
  - Consider calling someone on the phone. When you call, you ***initiate the session***. You and the person on the other end take turns talking, or ***exchanging dialog***. When the two of you are done talking, you hang up, or ***close the session***.

Naming

URL, URI

- URI: Uniform Resource Identifier
- URL: Uniform Resource Locator
- Often used interchangeably, but there is a difference:
  - URL is very specific: includes item (e.g. a specific file name) and protocol (how to get the item).
    - Example: http://www.northeastern.edu/index.html
  - URI can be less specific:
    - Example: northeastern.edu
    - Doesn’t specify access (e.g., ftp? http?) or specific page (index.html).

Anatomy of a URL

```
http://www.theimdbapi.org/api/movie?movie_id=tt0089218
```

Protocol Resource name Path Parameters

Anatomy of a URL

```
http://www.theimdbapi.org/api/movie?movie_id=tt0089218
```

Protocol Resource Name Path Parameters

Without protocol & resource name, we can't have a URL. Path and parameters can be null.

Anatomy of a URL

```
http://www.theimdbapi.org/api/movie?movie_id=tt0089218
```

Protocol

Resource Name:
- Hostname
- Filename
- Port Number
- Reference (optional)

Path

Parameters

Anatomy of a URL

```
http://www.theimdbapi.org/api/movie?movie_id=tt0089218
```

Protocol

Resource Name:
- Hostname
- Filename
- Port Number
- Reference (optional)

Path

Parameters

All of this information allows a **socket** to be opened up.

But connecting only via URLs is pretty high level– a lot of abstraction is happening.

What if we want to define our own protocol? We need to open a socket directly.

Java Classes

- java.net.URL
- java.net.URI
- java.net.Socket

```
private static void tryUrl(){
    try {
        // Create URL
        URL myURL = new URL("northeastern.edu");
        System.out.println("The URL is " + myURL);
    }
    catch (MalformedURLException e) {
        // new URL() failed
        e.printStackTrace();
    }
}

private static void tryUri(){
    try {
        // Create URI
        URI myURI = new URI("northeastern.edu");
        System.out.println("The URI is " + myURI);
    } catch (URISyntaxException e) {
        e.printStackTrace();
    }
}
```

```
private static void tryUrl(){
    try {
        // Create URL
        URL myURL = new URL("northeastern.edu");
        System.out.println("The URL is " + myURL);
    }
    catch (MalformedURLException e) {
        // new URL() failed
        e.printStackTrace();
    }
}

private static void tryUri(){
    try {
        // Create URI
        URI myURI = new URI("northeastern.edu");
        System.out.println("The URI is " + myURI);
    } catch (URISyntaxException e) {
        e.printStackTrace();
    }
}
```

Which one throws an exception?

```java
private static void tryUrl(){
    try {
        // Create URL
        URL myURL = new URL("northeastern.edu");
        System.out.println("The URL is " + myURL);
    }
    catch (MalformedURLException e
        // new URL() failed
        e.printStackTrace();
    }
}

private static void tryUri(){
    try {
        // Create URI
        URI myURI = new URI("nort
        System.out.println("The U
    } catch (URISyntaxException e
        e.printStackTrace();
    }
}
```

tryURL() fails, because the string "northeastern.edu" doesn't tell us enough about the protocol or file that we're interested in.

Replacing the string with "http://northeastern.edu" will make it work.

Some popular protocols

- HTTP: Hypertext Transfer Protocol
- FTP: File Transfer Protocol
- SMTP: Simple Mail Transfer Protocol

```
try (
        Socket socket = new Socket(hostName, portNumber);
) {
    // App code goes here:
    // Read from socket, write to socket. (more details soon)
    socket.close();

} catch (UnknownHostException e) {
    System.err.println("Don't know about host " + hostName);
    System.exit(1);
}
```

To go lower-level, open a Socket with a hostname and a portNumber.

Summary of Naming

- We have to have a way of specifying which computer we want to connect to
- In Java, we do this with URIs, URLs, and for lower-level client/server programming, sockets
- A socket requires a hostname and a port
- A URL requires a protocol and a resource name

Transfer

Once we have a name for the host we want to connect to, we need to open a connection and start the data transfer.

Relevant Java Classes

- For naming:
  - java.net.URL
  - java.net.URI
- For connecting:
  - java.net.URLConnection, java.net.HttpUrlConnection
  - java.net.Socket
- For actual transfer:
  - java.io.InputStreamReader
  - java.io.BufferedReader
  - java.io.PrintWriter

Three Examples

1. Reading data from a URL directly
2. Connect to a URL, and initiate a session for input/output
3. Create a socket and connect to it directly

Example 1:
Read directly from URL

```
private static void readUrl(){
    try {
        // Create URL
        URL myURL = new URL("http://www.northeastern.edu");

        BufferedReader in = new BufferedReader(
                new InputStreamReader(myURL.openStream()));

        String inputLine;
        while ((inputLine = in.readLine()) != null)
            System.out.println(inputLine);

        in.close();
    }
    catch (MalformedURLException e) {
        // new URL() failed
        // ...
    }
    catch (IOException e) {
        // openConnection() failed
        // ...
        e.printStackTrace();
    }

}
```

```
private static void readUrl(){
    try {
        // Create URL
        URL myURL = new URL("http://www.northeastern.edu");

        BufferedReader in = new BufferedReader(
                new InputStreamReader(myURL.openStream()));

        String inputLine;
        while ((inputLine = in.readLine()) != null)
            System.out.println(inputLine);

        in.close();
    }
    catch (MalformedURLException e) {
        // new URL() failed
        // ...
    }
    catch (IOException e) {
        // openConnection() failed
        // ...
        e.printStackTrace();
    }

}
```

Open a stream from the defined URL

```
private static void readUrl(){
    try {
        // Create URL
        URL myURL = new URL("http://www.northeastern.edu");

        BufferedReader in = new BufferedReader(
                new InputStreamReader(myURL.openStream()));

        String inputLine;
        while ((inputLine = in.readLine
            System.out.println(inputLin

        in.close();
    }
    catch (MalformedURLException e) {
        // new URL() failed
        // ...
    }
    catch (IOException e) {
        // openConnection() failed
        // ...
        e.printStackTrace();
    }

}
```

Pass it into an InputStreamReader to handle the input.

```
private static void readUrl(){
    try {
        // Create URL
        URL myURL = new URL("http://www.northeastern.edu");

        BufferedReader in = new BufferedReader(
                new InputStreamReader(myURL.openStream()));

        String inputLine;
        while ((inputLine = in.readLine())
            System.out.println(inputLine);

        in.close();
    }
    catch (MalformedURLException e) {
        // new URL() failed
        // ...
    }
    catch (IOException e) {
        // openConnection() failed
        // ...
        e.printStackTrace();
    }

}
```

Pass that into a BufferedReader to make it easy for you to handle the input.

```
private static void readUrl(){
    try {
        // Create URL
        URL myURL = new URL("http://www.northeastern.edu");

        BufferedReader in = new BufferedReader(
                new InputStreamReader(myURL.openStream()));

        String inputLine;
        while ((inputLine = in.readLine()) != null)
            System.out.println(inputLine);

        in.close();
    }
    catch (MalformedURLException e) {
        // new URL() failed
        // ...
    }
    catch (IOException e) {
        // openConnection() failed
        // ...
        e.printStackTrace();
    }
}
```

While there is still text coming in from the stream connection, get it, and print to console.

```
private static void readUrl(){
    try {
        // Create URL
        URL myURL = new URL("http://www.northeastern.edu");

        BufferedReader in = new BufferedReader(
                new InputStreamReader(myURL.openStream()));

        String inputLine;
        while ((inputLine = in.readLine()) != null)
            System.out.println(inputLine);

        in.close();
    }
    catch (MalformedURL
        // new URL() fa
        // ...
    }
    catch (IOException e) {
        // openConnection() failed
        // ...
        e.printStackTrace();
    }

}
```

Don't forget to close your connection!!

Example 1 summary

- Simple, easy way to get data from a URL
- This example was a web page, but could just as easily be a REST endpoint that contains data
- Transfer was only one way: could only read
- Limited: Some web servers require specific HTTP headers/values, and you can’t modify the parameters here.

Example 2: Connect to URL for input/output

```
private static void openHttpConnection(){
    try {
        // Create URL
        String theURL = "http://www.theimdbapi.org/api/movie?movie_id=tt0089218";
        URL myURL = new URL(theURL);

        // Connect to URL
        HttpURLConnection connection = (HttpURLConnection) myURL.openConnection();
        connection.setRequestMethod("GET");
        connection.setRequestProperty("User-Agent", "App/java app
        connection.setRequestProperty("Content-Type", "applicatio

        connection.connect();

        // Read from/Write to the connection
        BufferedReader in = new BufferedReader(new InputStreamRea
                connection.getInputStream()));
        String inputLine;
        while ((inputLine = in.readLine()) != null) {
            System.out.println(inputLine);
        }
        in.close();
    }
    // Handle exceptions (omitted for clarity)
}
```

Rather than just calling "openStream()" on the URL, call openConnection() to create a connection object that we can set parameters on before calling.

```
private static void openHttpConnection(){
    try {
        // Create URL
        String theURL = "http://www.theimdbapi.org/api/movie?movie_id=tt0089218";
        URL myURL = new URL(theURL);

        // Connect to URL
        HttpURLConnection connection = (HttpURLConnection) myURL.openConnection();
        connection.setRequestMethod("GET");
        connection.setRequestProperty("User-Agent", "App/java app demo");
        connection.setRequestProperty("Content-Type", "application/json");
```

Now, set some parameters:

- requestMethod specifies a GET rather than a POST.
- This particular server requires a User-Agent.
- Content-type just says I expect json in return.
- These are all details that are not always relevant, and change from application to application.

```
                                                            reamReader(

        }
        in.close();
    }
    // Handle exceptions (omitted for clarity)
}
```

```
private static void openHttpConnection(){
    try {
        // Create URL
        String theURL = "http://www.theimdbapi.org/api/movie?movie_id=tt0089218";
        URL myURL = new URL(theURL);

        // Connect to URL
        HttpURLConnection connection = (HttpURLConnection) myURL.openConnection();
        connection.setRequest
        connection.setRequest                                         a app demo");
        connection.setRequest                                         cation/json");

        connection.connect();

        // Read from/Write to the connection
        BufferedReader in = new BufferedReader(new InputStreamReader(
                connection.getInputStream()));
        String inputLine;
        while ((inputLine = in.readLine()) != null) {
            System.out.println(inputLine);
        }
        in.close();
    }
    // Handle exceptions (omitted for clarity)
}
```

Connect!

This actually opens the connection with the given parameters.

```
private static void openHttpConnection(){
    try {
        // Create URL
        String theURL = "http://www.theimdbapi.org/api/movie?movie_id=tt0089218";
        URL myURL = new URL(theURL);

        // Connect to URL
        HttpURLConnection connection = (HttpURLConnection) myURL.openConnection();
        connection.setRequestMethod("GET");
        connection.setRequestProperty("User-Agent", "App/java app demo");
        connection.setRequestProperty("Content-Type", "application/json");

        connection.connect();

        // Read from/Write to the connection
        BufferedReader in = new BufferedReader(new InputStreamReader(
                connection.getInputStream()));
        String inputLine;
        while ((inputLine = in.readLine()) != null) {
            System.out.println(inputLine);
        }
        in.close();
    }
    // Handle exceptions (omitted for clarity)
}
```

But now, just do the same thing we did last time:
Create an inputStreamReader, wrap it in a BufferedReader, and dump the response to the console.

```
private static void openHttpConnection(){
    try {
        // Create URL
        String theURL = "http://www.theimdbapi.org/api/movie?movie_id=tt0089218";
        URL myURL = new URL(theURL);

        // Connect to URL
        HttpURLConnection connection = (HttpURLConnection) myURL.openConnection();
        connection.setRequestMethod("GET");
        connection.setRequestProperty("User-Agent", "App/java app demo");
        connection.setRequestProperty("Content-Type", "application/json");

        connection.connect();

        // Read from/Write to the connection
        BufferedReader in = new BufferedReader(new InputStreamReader(
                connection.getInputStream()));
        String inputLine;
        while ((inputLine = in.readLine()) != null) {
            System.out.println(inputLine);
        }
        in.close();
    }
    // Handle exceptions (omitted for clarity)
}
```

Don't forget to close!!

Example 2 summary

- Fairly easy way to connect to a URL
- Gives more control over the connection:
  - Can set parameters, header info
- We didn't use this, but we can use the connection to do output as well
- Still constrained to using a pre-specified protocol (HTTP, FTP, …)

Example 3:
Connect to Socket

Adapted from: https://docs.oracle.com/javase/tutorial/networking/sockets/clientServer.html

In this example, we're looking at an implementation of the Knock-Knock client-server we saw earlier.

Knock-Knock Demo Components

- KnockKnockServer:
  - Listens for clients.
  - Parses client input
  - Sends a response
- KnockKnockClient:
  - Takes in user input
  - Sends it to the server
  - Displays server response to the user
- KnockKnockProtocol: (We'll talk about this in the next section)
  - Determines appropriate output for given input

First the client…
(It's pretty similar to what we've seen before)

KnockKnockClient.java

```
try (
        Socket kkSocket = new Socket(hostName, portNumber);
        PrintWriter out = new PrintWriter(kkSocket.getOutput
        BufferedReader in = new BufferedReader(
                new InputStreamReader(kkSocket.getInputStrea
) {
    BufferedReader stdIn =
            new BufferedReader(new InputStreamReader(System.in));
    String fromServer;
    String fromUser;

    while ((fromServer = in.readLine()) != null) {
        System.out.println("Server: " + fromServer);
        if (fromServer.equals("Bye."))
            break;

        fromUser = stdIn.readLine();
        if (fromUser != null) {
            System.out.println("Client: " + fromUser);
            out.println(fromUser);
        }
    }
    kkSocket.close();
} catch (Exceptions)//Handle exceptions properly here. Omitted for clarity.
```

This time, start by opening a socket, giving a hostname and a portnumber.

KnockKnockClient.java

```
try (
        Socket kkSocket = new Socket(hostName, portNumber);
        PrintWriter out = new PrintWriter(kkSocket.getOutputStream(), true);
        BufferedReader in = new BufferedReader(
                new InputStreamReader(kkSocket.getInputStrea
) {
    BufferedReader stdIn =
            new BufferedReader(new InputStreamReader(System.
    String fromServer;
    String fromUser;

    while ((fromServer = in.readLine()) != null) {
        System.out.println("Server: " + fromServer);
        if (fromServer.equals("Bye."))
            break;

        fromUser = stdIn.readLine();
        if (fromUser != null) {
            System.out.println("Client: " + fromUser);
            out.println(fromUser);
        }
    }
    kkSocket.close();
} catch (Exceptions)//Handle exceptions properly here. Omitted for clarity.
```

In addition to reading from the server, we need to write to the server.
Do this by creating a PrintWriter.

KnockKnockClient.java

```
try (
        Socket kkSocket = new Socket(hostName, portNumber);
        PrintWriter out = new PrintWriter(kkSocket.getOutputStream(), true);
        BufferedReader in = new BufferedReader(
                new InputStreamReader(kkSocket.getInputStream()));
) {
    BufferedReader stdIn =
            new BufferedReader(new InputStreamReader(System.
    String fromServer;
    String fromUser;

    while ((fromServer = in.readLine()) != null) {
        System.out.println("Server: " + fromServer);
        if (fromServer.equals("Bye."))
            break;

        fromUser = stdIn.readLine();
        if (fromUser != null) {
            System.out.println("Client: " + fromUser);
            out.println(fromUser);
        }
    }
    kkSocket.close();
} catch (Exceptions)//Handle exceptions properly here. Omitted for clarity.
```

But since we also need to read from the server, also create the BufferedReader from an InputStreamReader.

KnockKnockClient.java

```
try (
        Socket kkSocket = new Socket(hostName, portNumber);
        PrintWriter out = new PrintWriter(kkSocket.getOutputStream(), true);
        BufferedReader in = new BufferedReader(
                new InputStreamReader(kkSocket.getInputStream()));
) {
    BufferedReader stdIn =
            new BufferedReader(new InputStreamReader(System.in));
    String fromServer;
    String fromUser;

    while ((fromServer = in.readLine()) != null) {
        System.out.println("Server: " + fromServer);
        if (fromServer.equals("Bye."))
            break;

        fromUser = stdIn.readLine();
        if (fromUser != null) {
            System.out.println("Client: " + fromUser);
            out.println(fromUser);
        }
    }
    kkSocket.close();
} catch (Exceptions)//Handle exceptions properly here. Omitted for clarity.
```

This client takes input from the user and sends it to the server.
Use another BufferedReader with another InputStreamReader to get input from System.in.

Note this pattern: System.in is a source of input to your program, just as the data we get from the server either via a socket or URLConnection.

KnockKnockClient.java

```
try (
        Socket kkSocket = new Socket(hostName, portNumber);
        PrintWriter out = new PrintWriter(kkSocket.getOutputStream(), true);
        BufferedReader in = new BufferedReader(
                new InputStreamReader(kkSocket.getInputStream()));
) {
    BufferedReader stdIn =
            new BufferedReader(new InputStreamReader(System.in));
    String fromServer;
    String fromUser;

    while ((fromServer = in.readLine()) != null) {
        System.out.println("Server: " + fromServer);
        if (fromServer.equals("Bye."))
            break;

        fromUser = stdIn.readLine();
        if (fromUser != null) {
            System.out.println("Client: " + fromUser);
            out.println(fromUser);
        }
    }
    kkSocket.close();
} catch (Exceptions)//Handle exceptions properly here. Omitted for clarity.
```

While the server is still sending us data, keep getting input from the user and sending it.

KnockKnockClient.java

```
try (
        Socket kkSocket = new Socket(hostName, portNumber);
        PrintWriter out = new PrintWriter(kkSocket.getOutputStream(), true);
        BufferedReader in = new BufferedReader(
                new InputStreamReader(kkSocket.getInputStream()));
) {
    BufferedReader stdIn =
            new BufferedReader(new InputStreamReader(System.in));
    String fromServer;
    String fromUser;

    while ((fromServer = in.readLine()) != null) {
        System.out.println("Server: " + fromServer);
        if (fromServer.equals("Bye."))
            break;

        fromUser = stdIn.readLine();
        if (fromUser != null) {
            System.out.println("Client: " + fromUser);
            out.println(fromUser);
        }
    }
    kkSocket.close();
} catch (Exceptions)//Handle exceptions properly here. Omitted for clarity.
```

The server sent us a message saying “Bye”, which is defined by the protocol as being time to finish.

KnockKnockClient.java

```
try (
        Socket kkSocket = new Socket(hostName, portNumber);
        PrintWriter out = new PrintWriter(kkSocket.getOutputStream(), true);
        BufferedReader in = new BufferedReader(
                new InputStreamReader(kkSocket.getInputStream()));
) {
    BufferedReader stdIn =
            new BufferedReader(new InputStreamReader(System.in));
    String fromServer;
    String fromUser;

    while ((fromServer = in.readLine()) != null) {
        System.out.println("Server: " + fromServer);
        if (fromServer.equals("Bye."))
            break;

        fromUser = stdIn.readLine();
        if (fromUser != null) {
            System.out.println("Client: " + fromUser
            out.println(fromUser);
        }
    }
    kkSocket.close();
} catch (Exceptions)//Handle exceptions properly here. Omitted for clarity.
```

Read a line from the terminal.

KnockKnockClient.java

```
try (
        Socket kkSocket = new Socket(hostName, portNumber);
        PrintWriter out = new PrintWriter(kkSocket.getOutputStream(), true);
        BufferedReader in = new BufferedReader(
                new InputStreamReader(kkSocket.getInputStream()));
) {
    BufferedReader stdIn =
            new BufferedReader(new InputStreamReader(System.in));
    String fromServer;
    String fromUser;

    while ((fromServer = in.readLine()) != null) {
        System.out.println("Server: " + fromServer);
        if (fromServer.equals("Bye."))
            break;

        fromUser = stdIn.readLine();
        if (fromUser != null) {
            System.out.println("Client: " + fromUser);
            out.println(fromUser);
        }
    }
    kkSocket.close();
} catch (Exceptions)//Handle exceptions properly here. Omitted for clarity.
```

Write that line to the terminal, then send the text to the server.

KnockKnockClient.java

```
try (
        Socket kkSocket = new Socket(hostName, portNumber);
        PrintWriter out = new PrintWriter(kkSocket.getOutputStream(), true);
        BufferedReader in = new BufferedReader(
                new InputStreamReader(kkSocket.getInputStream()));
) {
    BufferedReader stdIn =
            new BufferedReader(new InputStreamReader(System.in));
    String fromServer;
    String fromUser;

    while ((fromServer = in.readLine()) != null) {
        System.out.println("Server: " + fromServer);
        if (fromServer.equals("Bye."))
            break;

        fromUser = stdIn.readLine();
        if (fromUser != null) {
            System.out.println("Client: " + fromUser);
            out.println(fromUser);
        }
    }
    kkSocket.close();
} catch (Exceptions)//                               Omitted for clarity.
```

Don’t forget to close the connection when you’re done!!

Now the server…

KnockKnockServer.java

```
try (
       ServerSocket serverSocket = new ServerSocket(portNumber);
       Socket clientSocket = serverSocket.accept();
       PrintWriter out =
              new PrintWriter(clientSocket.getOutputS
       BufferedReader in = new BufferedReader(
              new InputStreamReader(clientSocket.getInputStream()));
) {
   String inputLine, outputLine;
   out.println("The knock knock server is here! Just come on along. ");
   // Initiate conversation with client
   KnockKnockProtocol kkp = new KnockKnockProtocol();
   outputLine = kkp.processInput(null);
   out.println(outputLine);

   while ((inputLine = in.readLine()) != null) {
       outputLine = kkp.processInput(inputLine);
       out.println(outputLine);
       if (outputLine.equals("Bye."))
          break;
   }
} catch (IOException e)// Do the right thing here. You should know by now.
```

Set up the socket to be a server listening on a specified port number (keep it >1000).

KnockKnockServer.java

```
try (
       ServerSocket serverSocket = new ServerSocket(portNumber);
       Socket clientSocket = serverSocket.accept();
       PrintWriter out =
              new PrintWriter(clientSocket.getOutputS
       BufferedReader in = new BufferedReader(
              new InputStreamReader(clientSocket.getI
) {
   String inputLine, outputLine;
   out.println("The knock knock server is here! Just come on along. ");
   // Initiate conversation with client
   KnockKnockProtocol kkp = new KnockKnockProtocol();
   outputLine = kkp.processInput(null);
   out.println(outputLine);

   while ((inputLine = in.readLine()) != null) {
       outputLine = kkp.processInput(inputLine);
       out.println(outputLine);
       if (outputLine.equals("Bye."))
           break;
   }
} catch (IOException e)// Do the right thing here. You should know by now.
```

When a client comes along and connects to the socket, go ahead and accept the connection.
Now you have a way to communicate directly with the client!

KnockKnockServer.java

```
try (
        ServerSocket serverSocket = new ServerSocket(portNumber);
        Socket clientSocket = serverSocket.accept();
        PrintWriter out =
                new PrintWriter(clientSocket.getOutputStream(), true);
        BufferedReader in = new BufferedReader(
                new InputStreamReader(clientSocket.getI
) {
    String inputLine, outputLine;
    out.println("The knock knock server is here! Just c
    // Initiate conversation with client
    KnockKnockProtocol kkp = new KnockKnockProtocol();
    outputLine = kkp.processInput(null);
    out.println(outputLine);

    while ((inputLine = in.readLine()) != null) {
        outputLine = kkp.processInput(inputLine);
        out.println(outputLine);
        if (outputLine.equals("Bye."))
            break;
    }
} catch (IOException e)// Do the right thing here. You should know by now.
```

Use the PrintWriter to send data out through the clientSocket.

KnockKnockServer.java

```
try (
        ServerSocket serverSocket = new ServerSocket(portNumber);
        Socket clientSocket = serverSocket.accept();
        PrintWriter out =
                new PrintWriter(clientSocket.getOutputStream(), true);
        BufferedReader in = new BufferedReader(
                new InputStreamReader(clientSocket.getInputStream()));
) {
    String inputLine, outputLine;
    out.println("The knock knock server is here! Just c
    // Initiate conversation with client
    KnockKnockProtocol kkp = new KnockKnockProtocol();
    outputLine = kkp.processInput(null);
    out.println(outputLine);

    while ((inputLine = in.readLine()) != null) {
        outputLine = kkp.processInput(inputLine);
        out.println(outputLine);
        if (outputLine.equals("Bye."))
            break;
    }
} catch (IOException e)// Do the right thing here. You should know by now.
```

Once again, get the input stream from the socket, wrap it in a input stream, then wrap it in a BufferedReader.

KnockKnockServer.java

```
try (
        ServerSocket serverSocket = new ServerSocket(portNumber);
        Socket clientSocket = serverSocket.accept();
        PrintWriter out =
                new PrintWriter(clientSocket.getOutputStream(), true);
        BufferedReader in = new BufferedReader(
                new InputStreamReader(clientSocket.getInputStream()));
) {
    String inputLine, outputLine;
    out.println("The knock knock server is here! Just come on along. ");
    // Initiate conversation with client
    KnockKnockProtocol kkp = new KnockKnockProtocol();
    outputLine = kkp.processInput(null);
    out.println(outputLine);

    while ((inputLine = in.readLine()) != null) {
        outputLine = kkp.processInput(inputLine);
        out.println(outputLine);
        if (outputLine.equals("Bye."))
            break;
    }
} catch (IOException e)// Do the right thing here. You should know by now.
```

We'll discuss this later, but it keeps track of the joke state and determines what should be said.

KnockKnockServer.java

```
try (
        ServerSocket serverSocket = new ServerSocket(portNumber);
        Socket clientSocket = serverSocket.accept();
        PrintWriter out =
                new PrintWriter(clientSocket.getOutputStream(), true);
        BufferedReader in = new BufferedReader(
                new InputStreamReader(clientSocket.getInputStream()));
) {
    String inputLine, outputLine;
    out.println("The knock knock server is here! Just come on along. ");
    // Initiate conversation with client
    KnockKnockProtocol kkp = new KnockKnockProtocol();
    outputLine = kkp.processInput(null);
    out.println(outputLine);

    while ((inputLine = in.readLine()) != null) {
        outputLine = kkp.processInput(inputLine);
        out.println(outputLine);
        if (outputLine.equals("Bye."))
            break;
    }
} catch (IOException e)// Do the right thing here. You should know by now.
```

We'll discuss this later, but it keeps track of the joke state and determines what should be said.

KnockKnockServer.java

```
try (
        ServerSocket serverSocket = new ServerSocket(portNumber);
        Socket clientSocket = serverSocket.accept();
        PrintWriter out =
                new PrintWriter(clientSocket.getOutputStream(), true);
        BufferedReader in = new BufferedReader(
                new InputStreamReader(clientSocket.getInputStream()));
) {
    String inputLine, outputLine;
    out.println("The knock knock server is here! Just come on along. ");
    // Initiate conversation with client
    KnockKnockProtocol kkp = new KnockKnockProtocol();
    outputLine = kkp.processInput(null);
    out.println(outputLine);

    while ((inputLine = in.readLine()) != null) {
        outputLine = kkp.processInput(inputLine);
        out.println(outputLine);
        if (outputLine.equals("Bye."))
            break;
    }
} catch (IOException e)// Do the right thing here. You should know by now.
```

Read the input from the client.

KnockKnockServer.java

```
try (
        ServerSocket serverSocket = new ServerSocket(portNumber);
        Socket clientSocket = serverSocket.accept();
        PrintWriter out =
                new PrintWriter(clientSocket.getOutputStream(), true);
        BufferedReader in = new BufferedReader(
                new InputStreamReader(clientSocket.getInputStream()));
) {
    String inputLine, outputLine;
    out.println("The knock knock server is here! Just come on along. ");
    // Initiate conversation with client
    KnockKnockProtocol kkp = new KnockKnockProtocol();
    outputLine = kkp.processInput(null);
    out.println(outputLine);

    while ((inputLine = in.readLine()) != null) {
        outputLine = kkp.processInput(inputLine);
        out.println(outputLine);
        if (outputLine.equals("Bye."))
            break;
    }
} catch (IOException e)// Do the right thing here. You should know by now.
```

Send the input from the client to the protocol to determine how to respond.

KnockKnockServer.java

```
try (
        ServerSocket serverSocket = new ServerSocket(portNumber);
        Socket clientSocket = serverSocket.accept();
        PrintWriter out =
                new PrintWriter(clientSocket.getOutputStream(), true);
        BufferedReader in = new BufferedReader(
                new InputStreamReader(clientSocket.getInputStream()));
) {
    String inputLine, outputLine;
    out.println("The knock knock server is here! Just come on along. ");
    // Initiate conversation with client
    KnockKnockProtocol kkp = new KnockKnockProtocol();
    outputLine = kkp.processInput(null);
    out.println(outputLine);

    while ((inputLine = in.readLine()) != null) {
        outputLine = kkp.processInput(inputLine);
        out.println(outputLine);
        if (outputLine.equals("Bye."))
            break;
    }
} catch (IOException e)// Do the right thing here. You should know by now.
```

If the protocol says to say "Bye", the session is over and we can quit.

KnockKnockServer.java

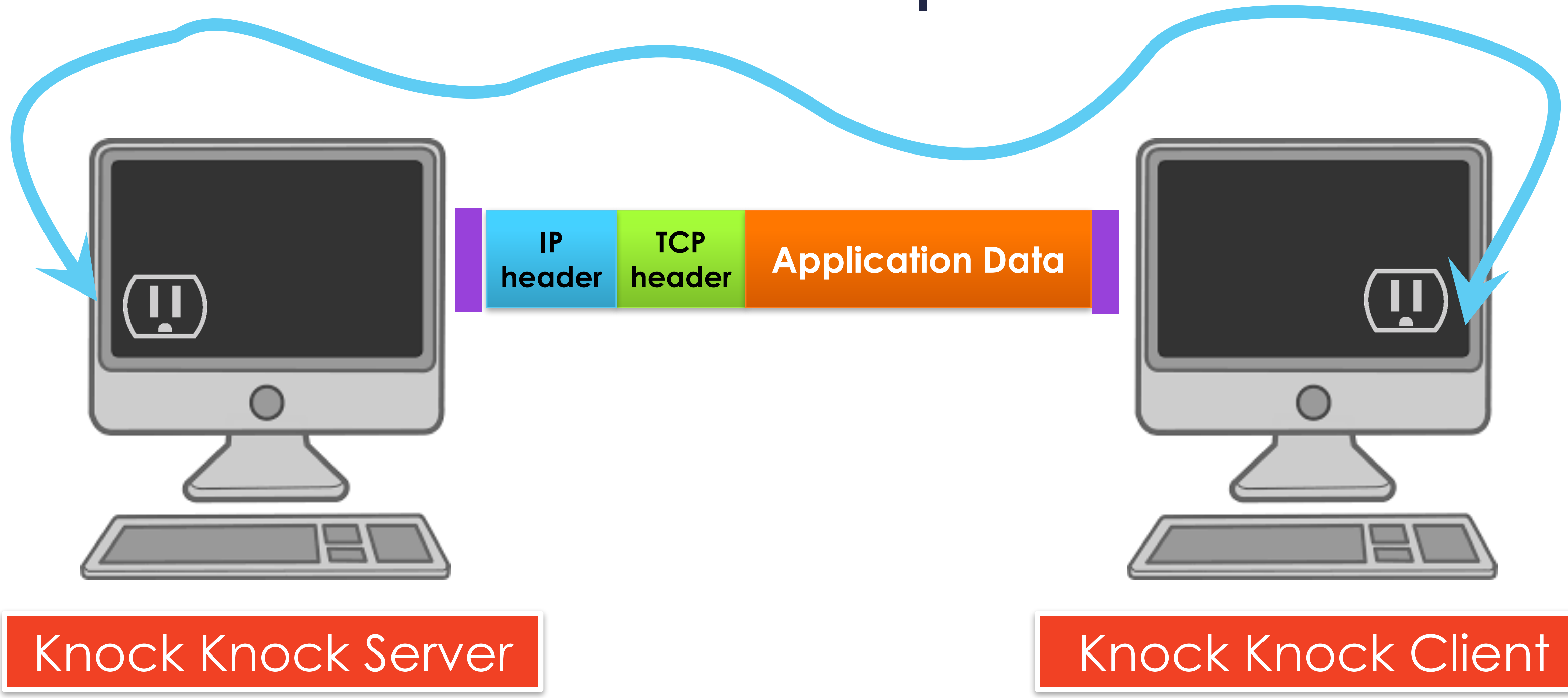
```java
try (
        ServerSocket serverSocket = new ServerSocket(portNumber);
        Socket clientSocket = serverSocket.accept();
        PrintWriter out =
                new PrintWriter(clientSocket.getOutputStream(), true);
        BufferedReader in = new BufferedReader(
                new InputStreamReader(clientSocket.getInputStream()));
) {
    String inputLine, outputLine;
    out.println("The knock knock server is here! Just come on along. ");
    // Initiate conversation with client
    KnockKnockProtocol kkp = new KnockKnockProtocol();
    outputLine = kkp.processInput(null);
    out.println(outputLine);

    while ((inputLine = in.readLine()) != null) {
        outputLine = kkp.processInput(inputLine);
        out.println(outputLine);
        if (outputLine.equals("Bye."))
            break;
    }
} catch (IOException e)// Do the right thing here. You should know by now.
```

Don't forget to close your connection!!

Some notes, now that we've seen the code.

- The server runs and opens up a socket on a specific port (e.g. 1200)
- The client runs, and we provide it with the name of the server (hostname) and the port (e.g. 1200)
- When the server and client are running on the same machine (e.g., testing), the hostname is "localhost"

Remember this picture?



Example 3 Summary

- The client reads input from the server, and sends data to the server.
- The server reads input from the client, and sends the data to the client.
- The protocol decides how to interpret the messages sent between the client and the server.

Communicating

Imagine two people talking to each other.

One is speaking in French, the other is speaking in English.

How much communication is happening?

True communication can't happen if we don't agree on what words mean what thing.

This is where the protocol comes in.

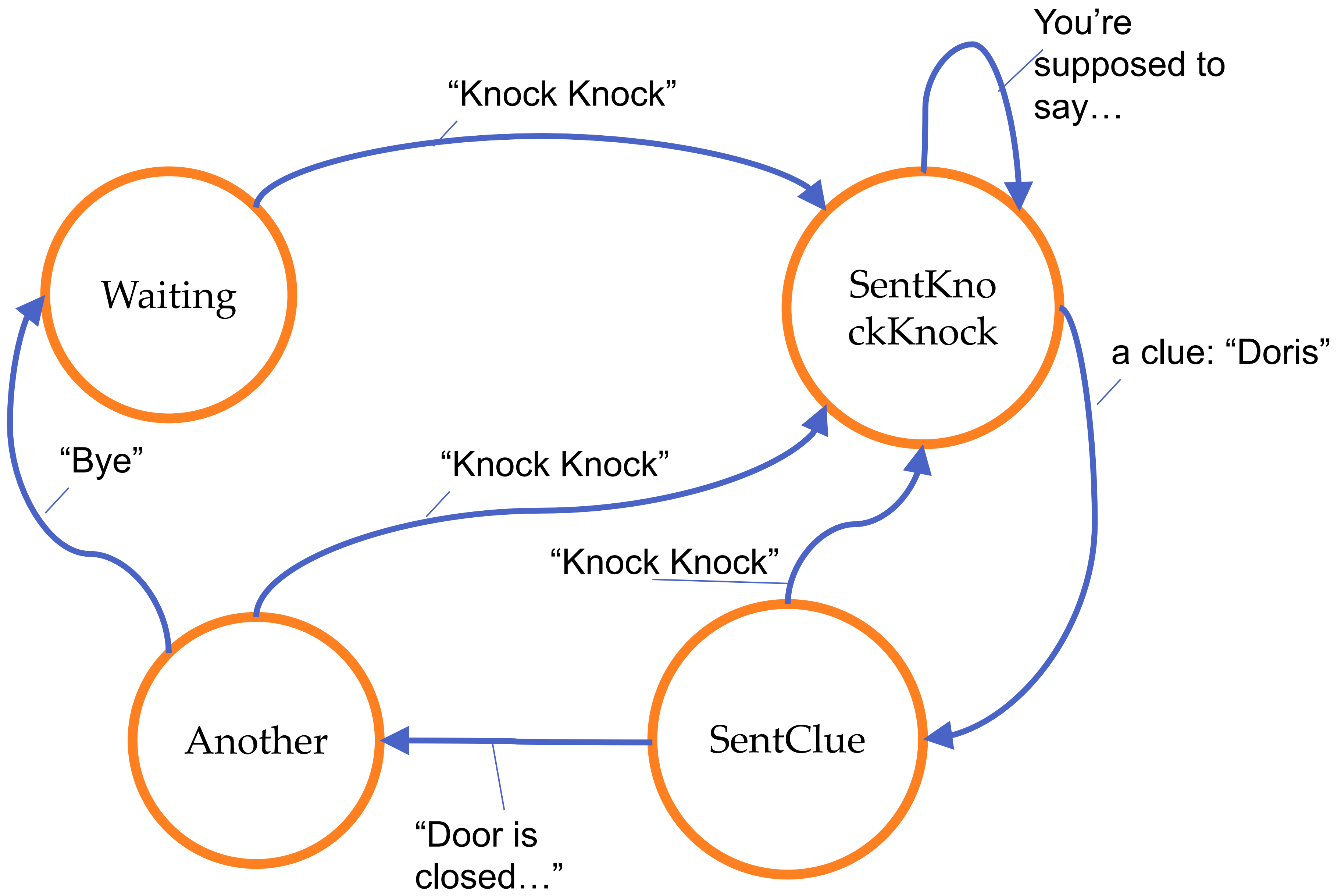
© Northeastern University

All About Protocols

- Usually defined in a document
- Sometimes implemented as a library that can be included in your code
- Whether your code uses an external library or not, it needs to conform to the protocol

Knock Knock Protocol

- Can be represented by a state diagram (next slide)
- The output is a combination of the current state and the input (from the client)



You're supposed to say…
"Knock Knock"
Waiting
SentKnockKnock
a clue: "Doris"
"Bye"
"Knock Knock"
"Knock Knock"
Another
SentClue
"Door is closed…"

```
switch(state){
    case WAITING:
        theOutput = "Knock Knock";
        state = SENTKNOCKKNOCK;
        break;

    case SENTKNOCKKNOCK:

        if (theInput.equalsIgnoreCase("")){
            theOutput = clues[currentJoke];
            state = SENTCLUE;
        }
        else{
            theOutput = "You're supposed to say Who's there?";
        }
        break;

    case SENTCLUE:
        if (theInput.equalsIgnoreCase(clues[currentJoke] + " who?")){
            theOutput = answers[currentJoke] + " Want another? (y/n)";
            state = ANOTHER;
        }
        else{//...
        state = ANOTHER;
    }
    else{


        theOutput = "You're supposed to say... ";
        state = WAITING;
    }
    break;

case ANOTHER:
    if (theInput.equalsIgnoreCase("y")) {
        theOutput = "Knock! Knock!";
        if (currentJoke == (NUMJOKES - 1))
            currentJoke = 0;
        else
            currentJoke++;
        state = SENTKNOCKKNOCK;
    } else {
        theOutput = "Bye.";
        state = WAITING;
    }
    break;
default:
    theOutput = "Whaaaat?";
    state = WAITING;
    break;
```

Summary

Ways of Networking in Java

- Via URL Connection
  - Create a URL
  - Establish a connection
  - Make requests:
    - PUT
    - GET
  - Process response
  - Can either read directly, or establish session and communicate
- Via Sockets
  - Direct connection to a server via a socket listening on a port
  - Must follow agreed-upon protocol