CS 5004: Lecture 14

Northeastern University, Spring 2021

At the start of every lecture

1. Pull the latest code from the lecture-code repo
2. Open the Evening_lectures folder
3. Copy this week's folder somewhere else
   - So you can edit it without causing GitHub conflicts
4. Open the code:
   1. Find the build.gradle file in the folder called LectureX
   2. Double click it to open the project

Agenda

- Functional programming – a little more:
  - Functional interfaces
  - Predicates
- Networking
- Final exam info

Functional Java

Functional interfaces & Predicates

Functional interfaces

- A special type of interface
- Allows exactly one abstract method
  - i.e. one public method that clients will implement
  - may have static/default/overridden methods too
- Can be used to store a lambda
  - Variable type is the functional interface, value is an implementation of it's method

Functional interfaces: `Comparator`

```
Comparator<Book> sortByPrice = Comparator.comparing(Book::getPrice);
```

- one abstract method: `int compare(T o1, T o2);`

Some non-abstract methods

`comparing` - static

`naturalOrder` - static

`equals` - overridden

Functional interfaces: `Function`

```
Function<Book, Book> discount
      = b -> new Book(b.getTitle(),
                      b.getAuthor(),
                      b.getYear(),
                      b.getPrice() * 0.9f);
```

Function contains one abstract method: `R apply(T t);`

Functional interfaces: `Predicate`

Predicate = boolean-valued function

`Stream<T>`	`filter(Predicate<? super T> predicate)`	Returns a stream consisting of the elements of this stream that match the given predicate.

Functional interfaces: `Predicate`

Predicate = boolean-valued function

`Stream<T>`	`filter(Predicate<? super T> predicate)`	Returns a stream consisting of the elements of this stream that match the given predicate.

- Takes a single value as input → outputs a boolean

```
books.stream().filter(book -> book.getYear() < 1995)...
```

Functional interfaces: `Predicate`

Predicate = boolean-valued function

`Stream<T>`	`filter(Predicate<? super T> predicate)`	Returns a stream consisting of the elements of this stream that match the given predicate.

- Takes a single value as input → outputs a boolean

```
books.stream().filter(book -> book.getYear() < 1995)...
```

Predicate

Functional interfaces: `Predicate`

Predicate = boolean-valued function

`Stream<T>`	`filter(Predicate<? super T> predicate)`	Returns a stream consisting of the elements of this stream that match the given predicate.

- Takes a single value as input → outputs a boolean

```
books.stream().filter(book -> book.getYear() < 1995)...
```

Single value as input

Functional interfaces: `Predicate`

Predicate = boolean-valued function

`Stream<T>`	`filter(Predicate<? super T> predicate)`	Returns a stream consisting of the elements of this stream that match the given predicate.

- Takes a single value as input → outputs a boolean

```
books.stream().filter(book -> book.getYear() < 1995)...
```

boolean output

Functional interfaces: `Predicate`

Predicate as variable:

```
Predicate<Book> pre1900 = book -> book.getYear() < 1900;
Predicate<Book> bargainBook = book -> book.getPrice() < 5;
```

Pass the variable as a parameter:

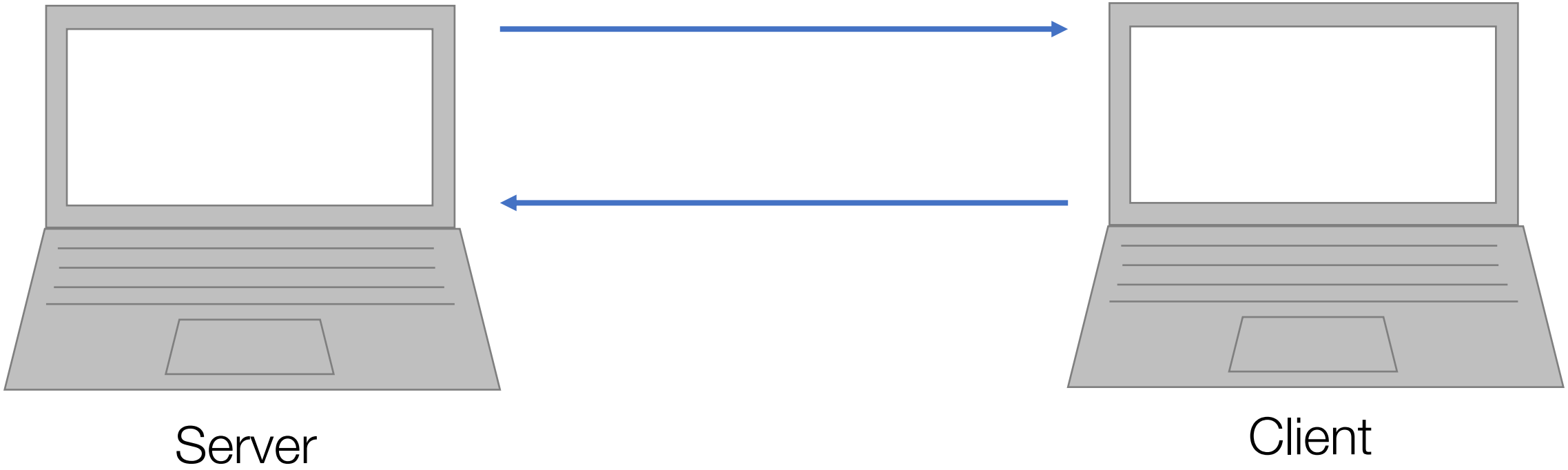
```
books.stream().filter(pre1900)...
books.stream().filter(bargainBook)...
```

Networking in Java

Credit: Adrienne Slaughter

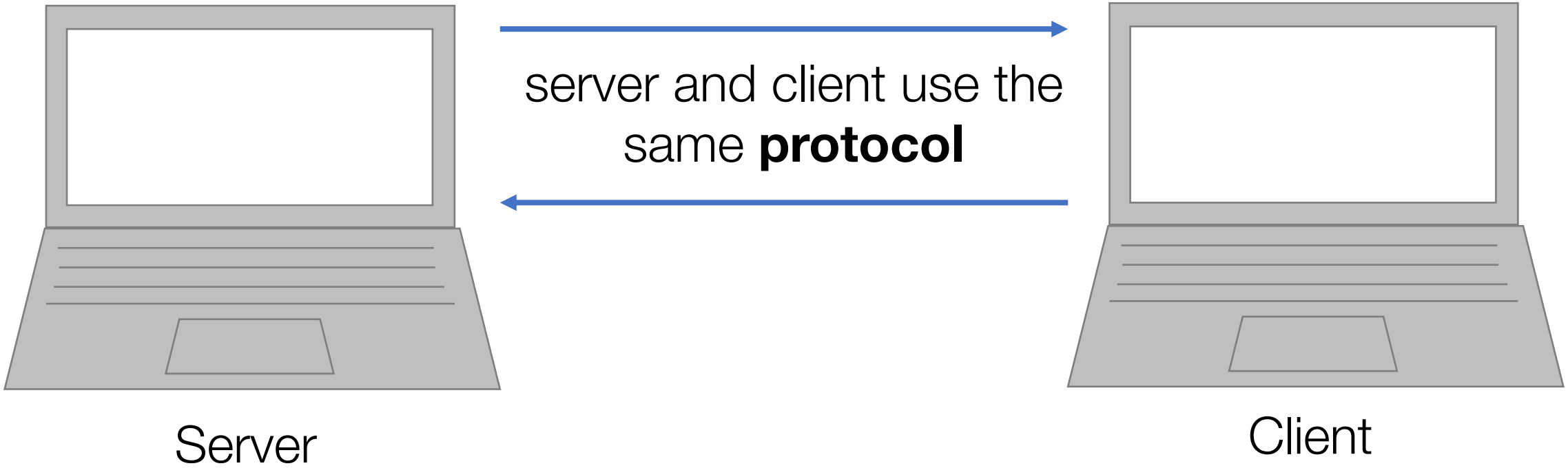
Networking

Goal: communicate data between applications

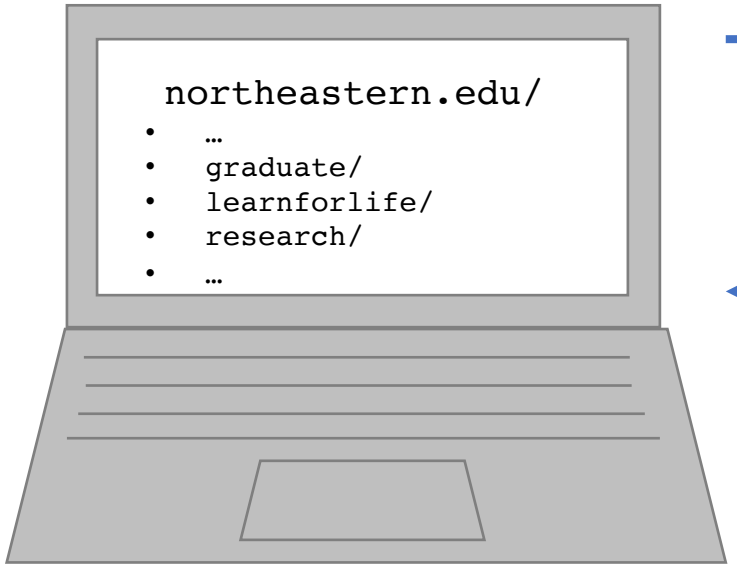


Protocol

A standardized set of steps in a standardized format

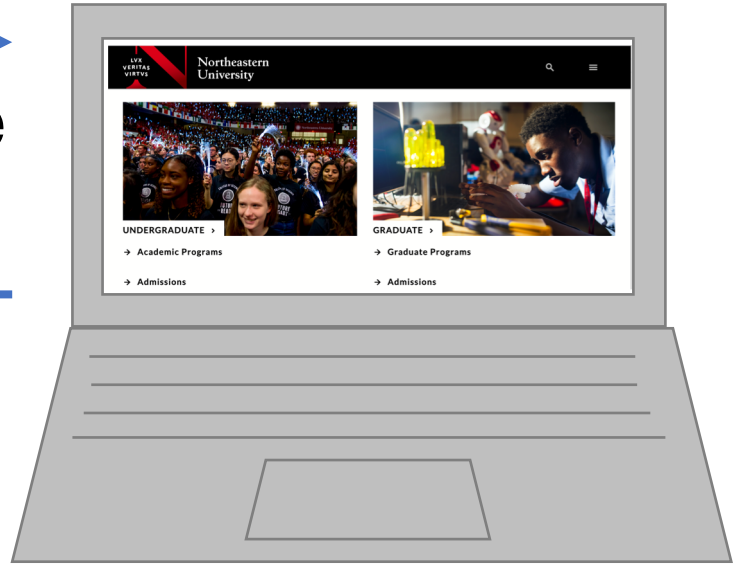


Browsing the internet



WebServer

server and client use the same **protocol: HTTP**



Client: Browser

Common protocols

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

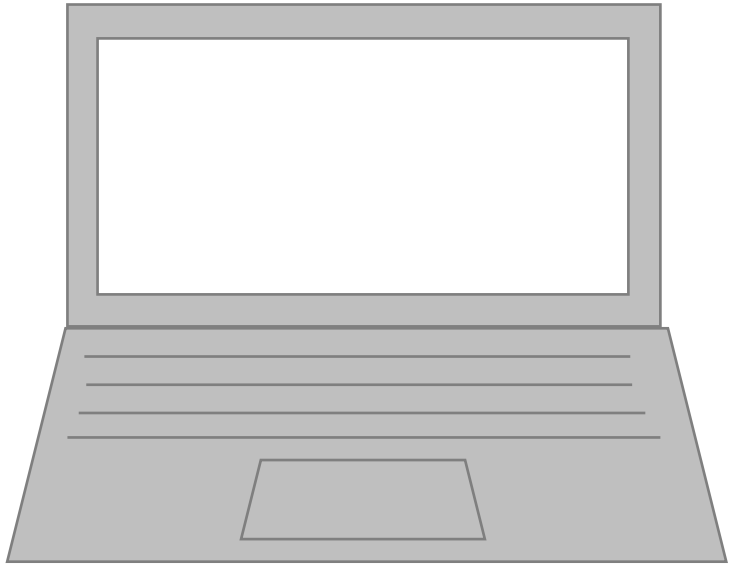
HTTP

- HyperText Transfer Protocol
- Consists of 2 basic messages:
  - Request
  - Response
- Each message consists of headers

How does the data get transferred?

How does the website get from the server to the client?

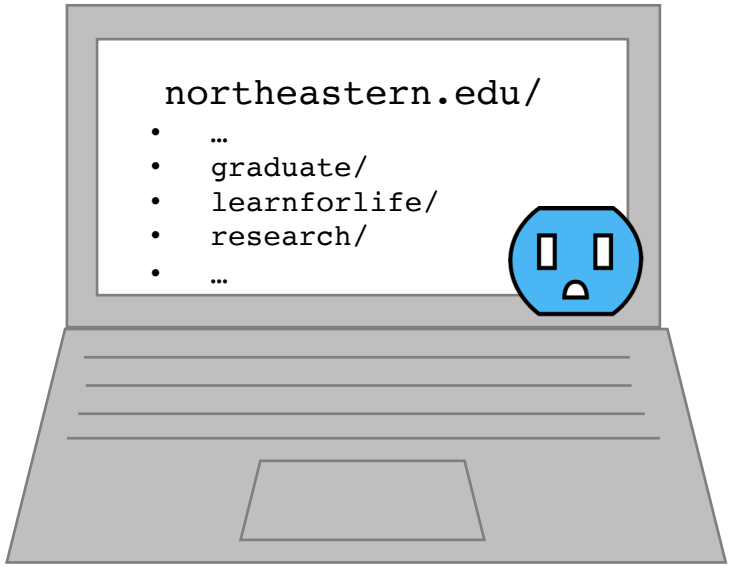
WebServer



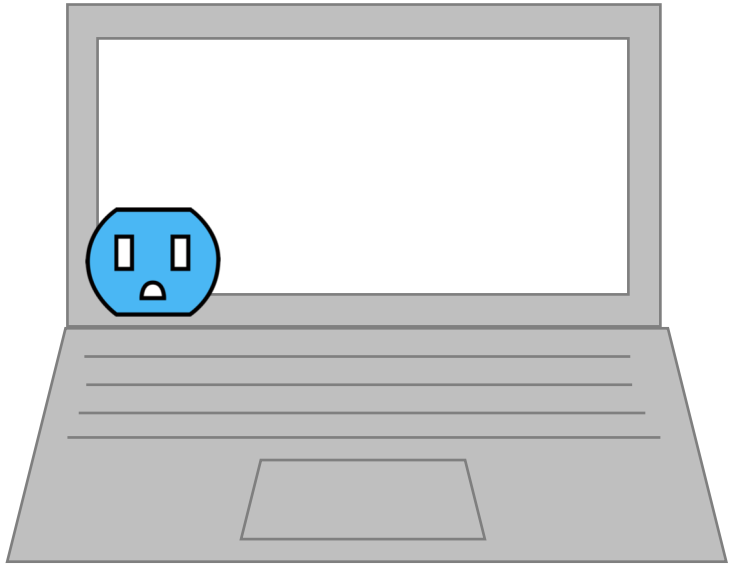
Client: Browser

How does the data get transferred?

Open a **socket** on each machine



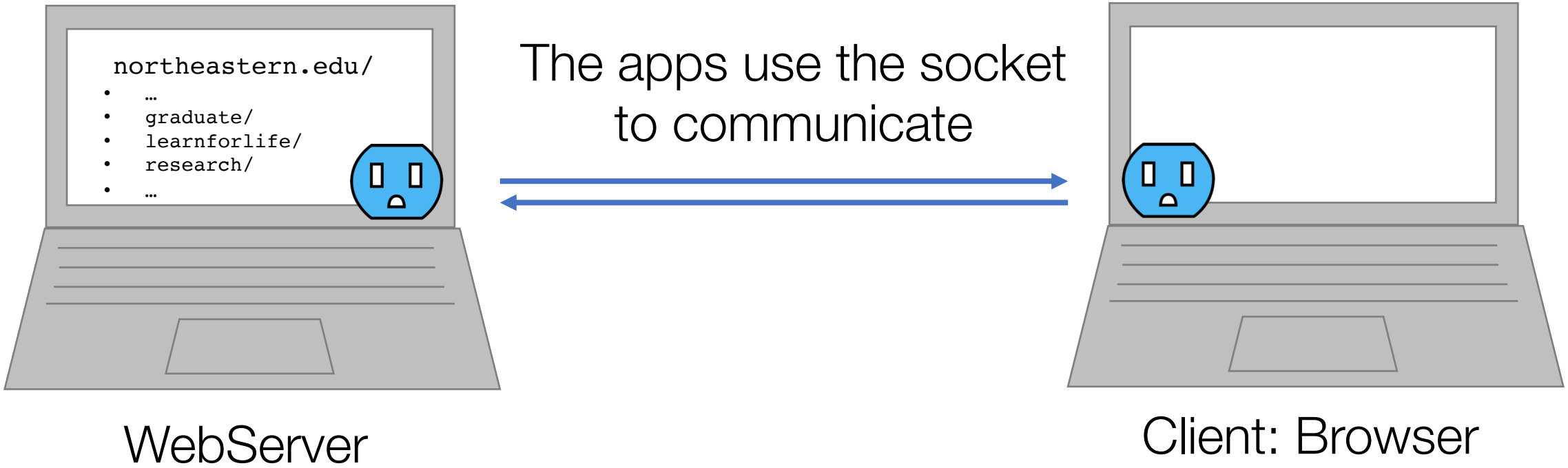
WebServer



Client: Browser

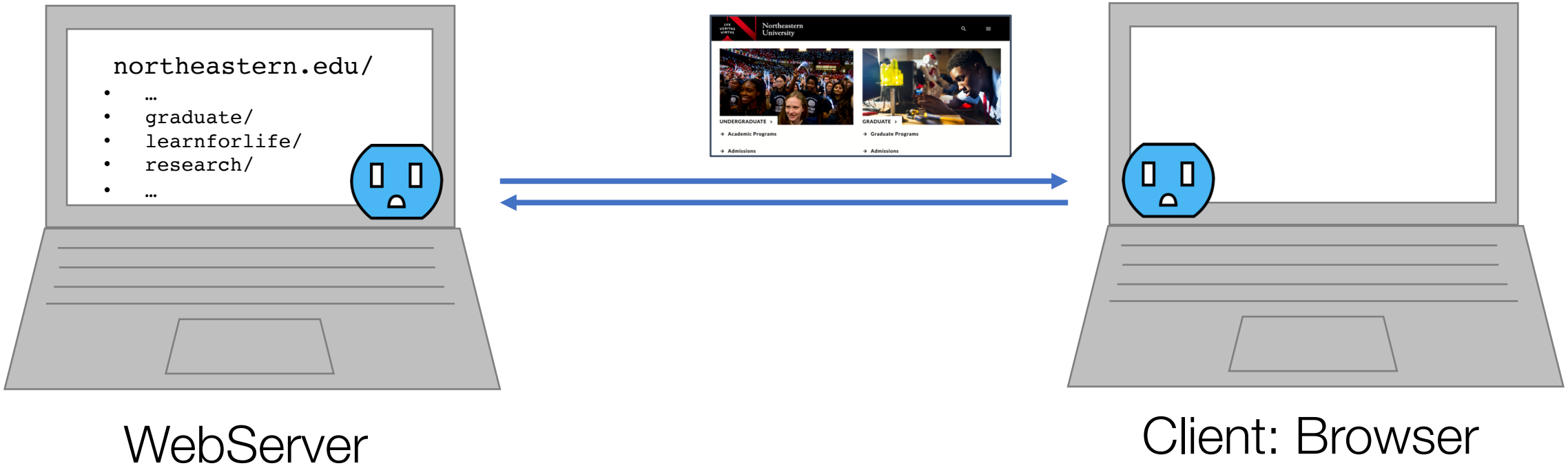
How does the data get transferred?

Open a **socket** on each machine



Transferring the data

We have application data to transfer…

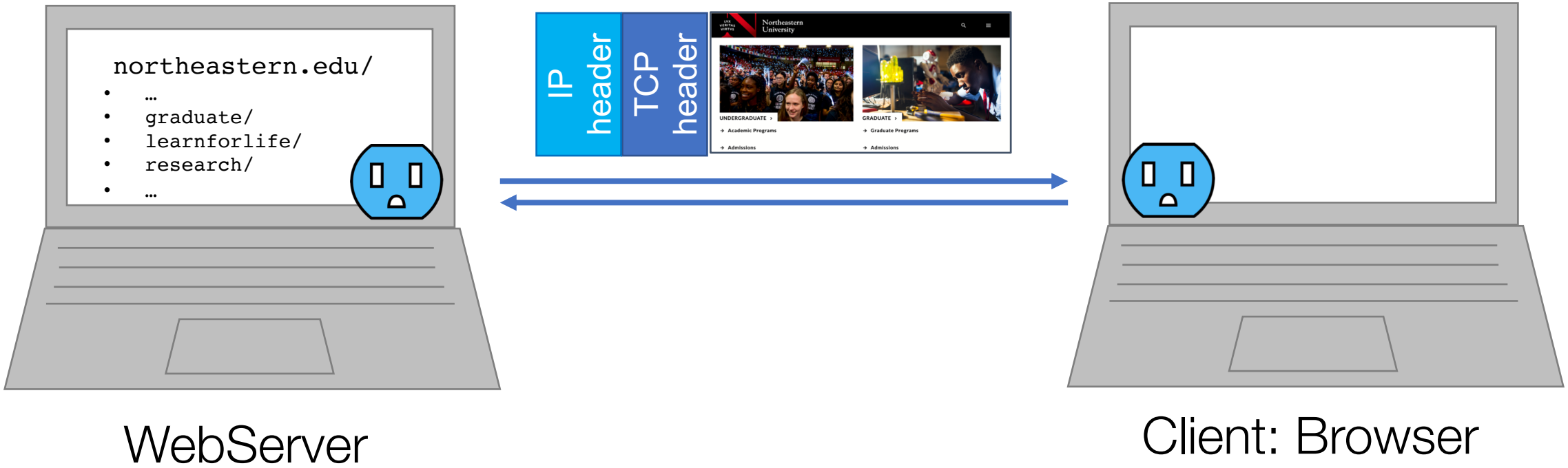


Transferring the data

A TCP header is added

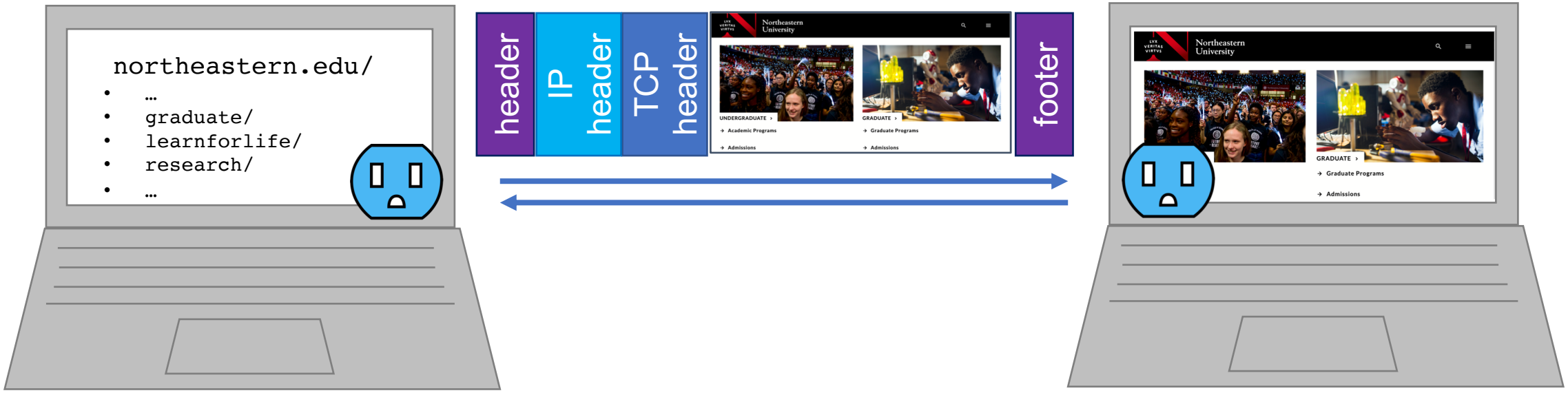
Transferring the data

…and an IP header is added



Transferring the data

Everything is wrapped in a “frame header/footer”

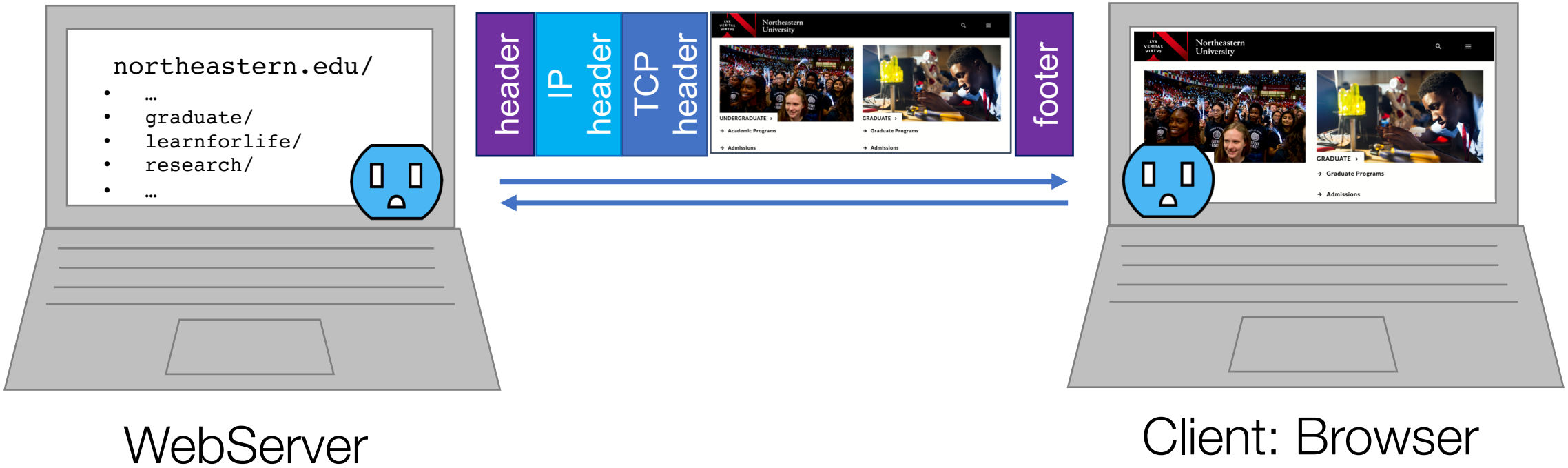


WebServer

Client: Browser

Transferring the data

Everything is passed through the connection



WebServer

Client: Browser

What do we need to know?

- Naming of network resources
  - How to specify which computer to connect to

What do we need to know?

- Naming of network resources
  - How to specify which computer to connect to
- Sockets
  - How to allow one computer to talk directly to another

What do we need to know?

- Naming of network resources
  - How to specify which computer to connect to
- Sockets
  - How to allow one computer to talk directly to another
- Communication protocols
  - Agreeing on the same protocols (don’t need the details)

What do we need to know?

- Naming of network resources
  - How to specify which computer to connect to
- Sockets
  - How to allow one computer to talk directly to another
- Communication protocols
  - Agreeing on the same protocols (don’t need the details)
- HTTP connections

What do we need to know?

- Naming of network resources
  - How to specify which computer to connect to
- Sockets
  - How to allow one computer to talk directly to another
- Communication protocols
  - Agreeing on the same protocols (don’t need the details)
- HTTP connections
- JSON
  - Very common format for sending data

The general process

1. Open a socket
2. Open an input stream and output stream to the socket
3. Read from/write to the stream according to protocol
4. Close the streams
5. Close the socket

Naming of network resources

- **URI**: Uniform Resource Identifier
- **URL**: Uniform Resource Locator
- They are different!

Naming of network resources

URL

- Very specific
- Includes item (e.g. a file name) AND protocol
- e.g. https://www.northeastern.edu/index.html

URI

Naming of network resources

URL

- Very specific
- Includes item (e.g. a file name) AND protocol
- e.g. https://www.northeastern.edu/index.html

URI

- Can be less specific
- Doesn't specify item or protocol
- e.g. northeastern.edu

Anatomy of a URL

https://www.imdb.com/title/tt0944947/episodes?season=8

Protocol

Resource name

Path

Parameters

Anatomy of a URL

https://www.imdb.com/title/tt0944947/episodes?season=8

Protocol

Resource name

Path

Parameters

- Protocol & resource name required
- Path & parameters can be null

Anatomy of a URL

https://www.imdb.com/title/tt0944947/episodes?season=8

Protocol

Resource name

Path

Parameters

- Connecting via URL hides a lot of details
- When defining a protocol → open a socket directly

Relevant Java Classes

For naming:

- `java.net.URL`
- `java.net.URI`

For connecting:

- `java.net.Socket`
- `java.net.URLConnection`

For data transfer:

- `java.io.InputStreamReader`
- `java.io.BufferedReader`
- `java.io.PrintWriter`

Relevant Java Classes

For naming:

- **`java.net.URL`**
- **`java.net.URI`**

For connecting:

- `java.net.Socket`
- `java.net.URLConnection`

For data transfer:

- `java.io.InputStreamReader`
- `java.io.BufferedReader`
- `java.io.PrintWriter`

Which one throws an exception?

```
try {
  URL myURL =
  new URL("northeastern.edu");
}
catch (MalformedURLException e)
{
  …
}
```

```
try {
  URI myURI =
  new URI("northeastern.edu");
}
catch (URISyntaxException e) {
  …
}
```

Which one throws an exception?

```
try {
  URL myURL =
  new URL("northeastern.edu");
}
catch (MalformedURLException e)
{
  …
}
```

URL must include the protocol:

https://northeastern.edu

```
try {
  URI myURI =
  new URI("northeastern.edu");
}
catch (URISyntaxException e) {
  …
}
```

Relevant Java Classes

For naming:

- `java.net.URL`
- `java.net.URI`

For connecting:

- **`java.net.Socket`**
- `java.net.URLConnection`

For data transfer:

- `java.io.InputStreamReader`
- `java.io.BufferedReader`
- `java.io.PrintWriter`

Lower level: open a Socket

```
try {
  Socket socket = new Socket(hostName, portNumber);
  // Read from socket, write to socket
  socket.close();
}
catch (UnknownHostException e) {
  …
}
```

Lower level: open a Socket

```
try {
  Socket socket = new Socket(hostName, portNumber);
  // Read from socket, write to socket
  socket.close();
}
catch (UnknownHostException e) {
  …
}
```

Required

Relevant Java Classes

For naming:

- `java.net.URL`
- `java.net.URI`

For connecting:

- `java.net.Socket`
- `java.net.URLConnection`

For data transfer:

- **`java.io.InputStreamReader`**
- **`java.io.BufferedReader`**
- `java.io.PrintWriter`

Data transfer: 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

Reading data from a URL

```
try {
  URL myURL = new URL("http://northeastern.edu");
}
catch (MalformedURLException e) {…}
```

Reading data from a URL

```
try {
  URL myURL = new URL("http://northeastern.edu");
}
catch (MalformedURLException e) {…}
```

Needed when creating a URL object

Reading data from a URL

```
try {
  URL myURL = new URL("http://northeastern.edu");
  BufferedReader in = new BufferedReader(
    new InputStreamReader(myURL.openStream());
}
catch (MalformedURLException e) {…}
catch (IOException e) {…}
```

Reading data from a URL

```
try {
  URL myURL = new URL("http://northeastern.edu");
  BufferedReader in = new BufferedReader(
    new InputStreamReader(myURL.openStream());
}
catch (MalformedURLException e) {…}
catch (IOException e) {…}
```

Open a stream from the URL

- Need to catch IOException incase of bad stream

Reading data from a URL

```
try {
  URL myURL = new URL("http://northeastern.edu");
  BufferedReader in = new BufferedReader(
    new InputStreamReader(myURL.openStream());
}
catch (MalformedURLException e) {…}
catch (IOException e) {…}
```

Pass it into an InputStreamReader

- does the actual reading

Reading data from a URL

```
try {
  URL myURL = new URL("http://northeastern.edu");
  BufferedReader in = new BufferedReader(
    new InputStreamReader(myURL.openStream());
}
catch (MalformedURLException e) {…}
catch (IOException e) {…}
```

Pass that to a BufferedStreamReader

- easier for you to handle the input
- better for system resources

Reading data from a URL

```
try {
  URL myURL = new URL("http://northeastern.edu");
  BufferedReader in = new BufferedReader(
    new InputStreamReader(myURL.openStream());
  String line;
  while((line = line.readLine()) != null) {...}
}
catch (MalformedURLException e) {…}
catch (IOException e) {…}
```

Just like reading a file

Reading data from a URL

```
try {
  URL myURL = new URL("http://northeastern.edu");
  BufferedReader in = new BufferedReader(
    new InputStreamReader(myURL.openStream());
  String line;
  while((line = line.readLine()) != null) {...}
  in.close();
}
catch (MalformedURLException e) {…}
catch (IOException e) {…}
```

Don't forget to close!

Reading data from a URL: REST API

```
try {
  URL myURL
       = new URL("https://ron-swanson-quotes.herokuapp.com/v2/quotes");
  BufferedReader in = new BufferedReader(
    new InputStreamReader(myURL.openStream());
  String line;
  while((line = line.readLine()) != null) {...}
  in.close();
}
catch (MalformedURLException e) {…}
catch (IOException e) {…}
```

API endpoint

Reading from a URL

- Easy way to get data from a URL
- Limited:
  - Read-only data transfer
  - Some servers require specific HTTP headers/values, can’t modify with this approach

Making a more specific request

```
try {
  URL url
      = new URL("https://opentdb.com/api.php?amount=1");
  HttpURLConnection conn
      = (HttpURLConnection) url.openConnection());
}
```

HttpURLConnection makes it possible to set request parameters

Making a more specific request

```
try {
  URL url
     = new URL("https://opentdb.com/api.php?amount=1");
  HttpURLConnection conn
     = (HttpURLConnection) url.openConnection());
  conn.setRequestMethod("GET");
  conn.setRequestProperty("User-Agent", "App/Demo");
  conn.setRequestProperty("Content-Type",
                                  "application/json");
}
```

Set parameters – varies by application

Making a more specific request

```
try {
  URL url
     = new URL("https://opentdb.com/api.php?amount=1");
  HttpURLConnection conn
     = (HttpURLConnection) url.openConnection());
  conn.setRequestMethod("GET");
  conn.setRequestProperty("User-Agent", "App/Demo");
  conn.setRequestProperty("Content-Type",
                                  "application/json");
  conn.connect();
}
```

Open the connection

Making a more specific request

```
try {
  URL url = new URL("https://opentdb.com/api.php?amount=1");
  HttpURLConnection conn
      = (HttpURLConnection) url.openConnection());
  conn.setRequestMethod("GET");
  conn.setRequestProperty("User-Agent", "App/Demo");
  conn.setRequestProperty("Content-Type", "application/json");
  conn.connect();
  BufferedReader in =
      new BufferedReader(
      new InputStreamReader(conn.getInputStream());
  // Read as before
  in.close();
}
```

Proceed as before

Making a more specific request

- Gives more control over the connection
- Can be used to do output as well
- Still constrained to using a pre-specified protocol
  - HTTP
  - FTP
  - etc.

Connecting to a Socket

A Ron Swanson quote client-server

- Asks the user how many quotes they want
- Prints the requested number of quotes (retrieved from a REST API)

Connecting to a Socket

A Ron Swanson quote client-server

- Server:
  - Listens for clients
  - Receives client input
  - Sends a response
- Client:
  - Takes in user input
  - Sends it to server
  - Displays server response
- Protocol (later)
  - Determines appropriate output for given input

The client: Client.java

We can use try-with-resources this time:

```
try (Socket socket = new Socket(address, port);
    PrintWriter out =
      new PrintWriter(socket.getOutputStream(), true);
    BufferedReader in = new BufferedReader(
      new InputStreamReader(clientSocket.getInputStream()));
    BufferedReader stdIn = new BufferedReader(
      new InputStreamReader(System.in)))
{


}
```

The client: Client.java

```
try (Socket socket = new Socket(address, port);
    PrintWriter out =
      new PrintWriter(socket.getOutputStream(), true);
    BufferedReader in = new BufferedReader(
      new InputStreamReader(clientSocket.getInputStream()));
    BufferedReader stdIn = new BufferedReader(
      new InputStreamReader(System.in)))
{


}
```

Open a socket

The client: Client.java

```
try (Socket socket = new Socket(address, port);
    PrintWriter out =
      new PrintWriter(socket.getOutputStream(), true);
    BufferedReader in = new BufferedReader(
      new InputStreamReader(clientSocket.getInputStream()));
    BufferedReader stdIn = new BufferedReader(
      new InputStreamReader(System.in)))
{


}
```

To write to the server

The client: Client.java

```
try (Socket socket = new Socket(address, port);
    PrintWriter out =
      new PrintWriter(socket.getOutputStream(), true);
    BufferedReader in = new BufferedReader(
      new InputStreamReader(clientSocket.getInputStream()));
    BufferedReader stdIn = new BufferedReader(
      new InputStreamReader(System.in)))
{


}
```

To read from the server

The client: Client.java

```
try (Socket socket = new Socket(address, port);
    PrintWriter out =
      new PrintWriter(socket.getOutputStream(), true);
    BufferedReader in = new BufferedReader(
      new InputStreamReader(clientSocket.getInputStream()));
    BufferedReader stdIn = new BufferedReader(
      new InputStreamReader(System.in)))
{


}
```

To read from the command line

The client: Client.java

```
try (...) {
  // Some lines omitted
  while ((fromServer =  in.readLine() != null) {
    System.out.println(fromServer);
    if (fromServer.equals("BYE")) {return; }
    System.out.println(INSTRUCTIONS);
    fromUser = stdIn.readLine();
    if (!fromUser.isEmpty()) {
      out.println(fromUser);
    }
  }
}
```

As long as the server is sending data to the client...

The client: Client.java

```
try (...) {
  // Some lines omitted
  while ((fromServer =  in.readLine() != null) {
    System.out.println(fromServer);
    if (fromServer.equals("BYE")) {return; }
    System.out.println(INSTRUCTIONS);
    fromUser = stdIn.readLine();
    if (!fromUser.isEmpty()) {
      out.println(fromUser);
    }
  }
}
```

Print the server's message

The client: Client.java

```
try (...) {
  // Some lines omitted
  while ((fromServer =  in.readLine() != null) {
    System.out.println(fromServer);
    if (fromServer.equals("BYE")) {return; }
    System.out.println(INSTRUCTIONS);
    fromUser = stdIn.readLine();
    if (!fromUser.isEmpty()) {
      out.println(fromUser);
    }
  }
}
```

If the server says "BYE" → close the connection

The client: Client.java

```
try (...) {
  // Some lines omitted
  while ((fromServer =  in.readLine() != null) {
    System.out.println(fromServer);
    if (fromServer.equals("BYE")) {return; }
    System.out.println(INSTRUCTIONS);
    fromUser = stdIn.readLine();
    if (!fromUser.isEmpty()) {
      out.println(fromUser);
    }
  }
}
```

Read a line from the terminal

The client: Client.java

```
try (...) {
  // Some lines omitted
  while ((fromServer =  in.readLine() != null) {
    System.out.println(fromServer);
    if (fromServer.equals("BYE")) {return; }
    System.out.println(INSTRUCTIONS);
    fromUser = stdIn.readLine();
    if (!fromUser.isEmpty()) {
      out.println(fromUser);
    }
  }
}
```

Send the message to the server

The server: Server.java

We can use try-with-resources this time:

```
try (ServerSocket serverSocket = new ServerSocket(port);
    Socket clientSocket = serverSocket.accept();
    PrintWriter out =
      new PrintWriter(clientSocket.getOutputStream(), true);
    BufferedReader in = new BufferedReader(
      new InputStreamReader(clientSocket.getInputStream())))
{


}
```

The server: Server.java

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

```
try (ServerSocket serverSocket = new ServerSocket(port);
    Socket clientSocket = serverSocket.accept();
    PrintWriter out =
      new PrintWriter(clientSocket.getOutputStream(), true);
    BufferedReader in = new BufferedReader(
      new InputStreamReader(clientSocket.getInputStream())))
{


}
```

The server: Server.java

When a client shows up, accept the connection

```
try (ServerSocket serverSocket = new ServerSocket(port);
    Socket clientSocket = serverSocket.accept();
    PrintWriter out =
      new PrintWriter(clientSocket.getOutputStream(), true);
    BufferedReader in = new BufferedReader(
      new InputStreamReader(clientSocket.getInputStream())))
{


}
```

The server: Server.java

```
try (ServerSocket serverSocket = new ServerSocket(port);
    Socket clientSocket = serverSocket.accept();
    PrintWriter out =
     new PrintWriter(clientSocket.getOutputStream(), true);
    BufferedReader in = new BufferedReader(
     new InputStreamReader(clientSocket.getInputStream())))
{


}
```

To send data to the client

The server: Server.java

```
try (ServerSocket serverSocket = new ServerSocket(port);
    Socket clientSocket = serverSocket.accept();
    PrintWriter out =
      new PrintWriter(clientSocket.getOutputStream(), true);
    BufferedReader in = new BufferedReader(
      new InputStreamReader(clientSocket.getInputStream()))
{

}
```

To read data from the client

The server: Server.java

```
try {
  // Some code omitted
  Protocol protocol = new Protocol();
  outputLine = protocol.processInput(null);
  out.println(outputLine);

}
```

We'll come back to this... Keeps track of where we are in the protocol, determines how to respond

The server: Server.java

```
try {
  // Some code omitted
  Protocol protocol = new Protocol();
  outputLine = protocol.processInput(null);
  out.println(outputLine);
}
```

Send the response to the client

The server: Server.java

```
try {
  // More code omitted
  while((inputLine = in.readLine()) != null) {
    outputLine = protocol.processInput(inputLine);
    out.println(outputLine);
    if (outputLine.equals("BYE")) { break;}
  }
}
```

Read the input from the client

The server: Server.java

```
try {
  // More code omitted
  while((inputLine = in.readLine()) != null) {
    outputLine = protocol.processInput(inputLine);
    out.println(outputLine);
    if (outputLine.equals("BYE")) { break;}
  }
}
```

Determine how to respond

The server: Server.java

```
try {
  // More code omitted
  while((inputLine = in.readLine()) != null) {
    outputLine = protocol.processInput(inputLine);
    out.println(outputLine);
    if (outputLine.equals("BYE")) { break;}
  }
}
```

Send the response to client

The server: Server.java

```
try {
  // More code omitted
  while((inputLine = in.readLine()) != null) {
    outputLine = protocol.processInput(inputLine);
    out.println(outputLine);
    if (outputLine.equals("BYE")) { break;}
  }
}
```

Time to quit

What we've seen so far

- The server runs and opens a socket on a specific port (e.g. 5000)
- The client runs, and we provide it with the server and port
  - e.g. "127.0.0.1" and 5000
  - "127.0.0.1" - the address of the local machine.

What we've seen so far

- The client reads input from the server and sends user input data to the server
- The server reads input from the client and sends a response back to the client
- The protocol decides how to interpret the messages sent between the client and server

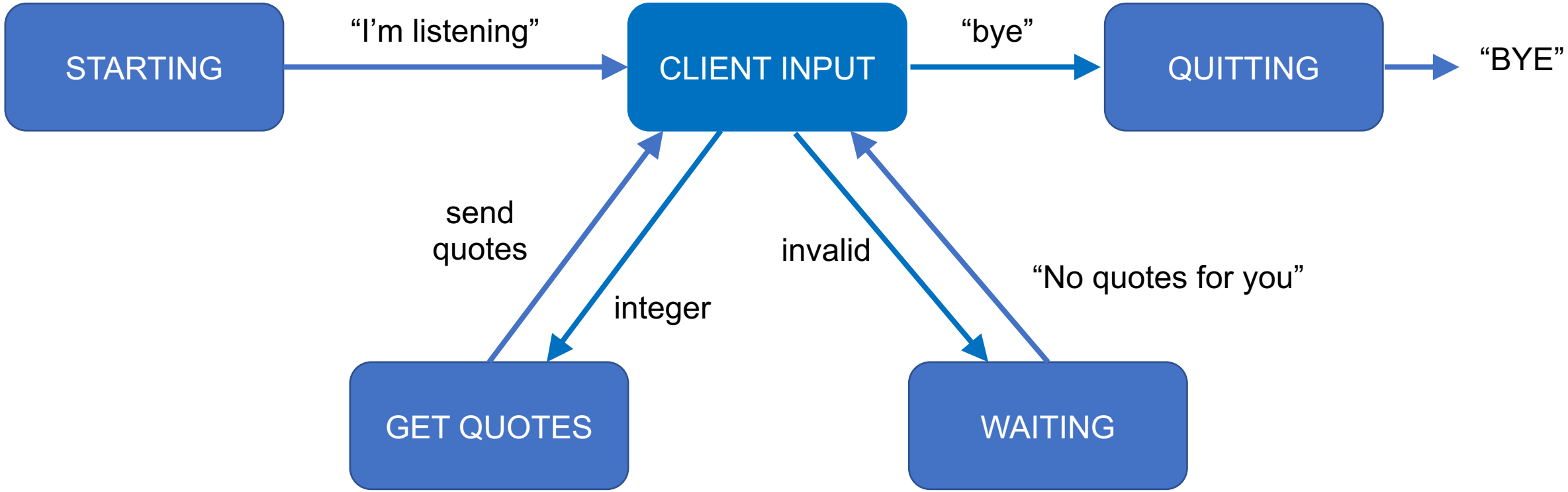
The protocol: managing communication

- Provides a “script” for the interaction between client and server
- Usually defined in a document
- Sometimes implemented as a library and included in code
- Either way, code must conform to the protocol

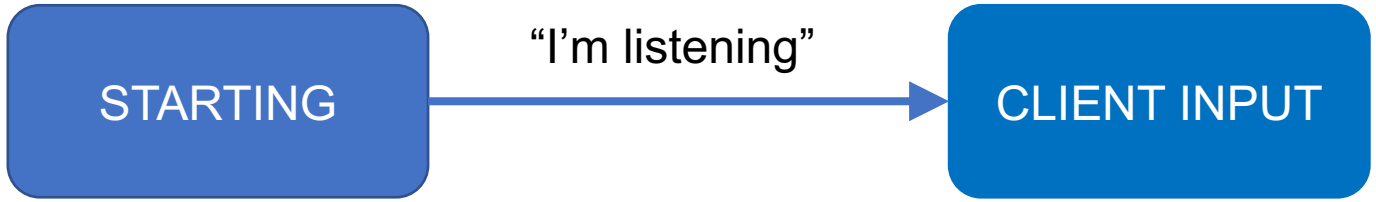
The Ron Swanson quote protocol

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

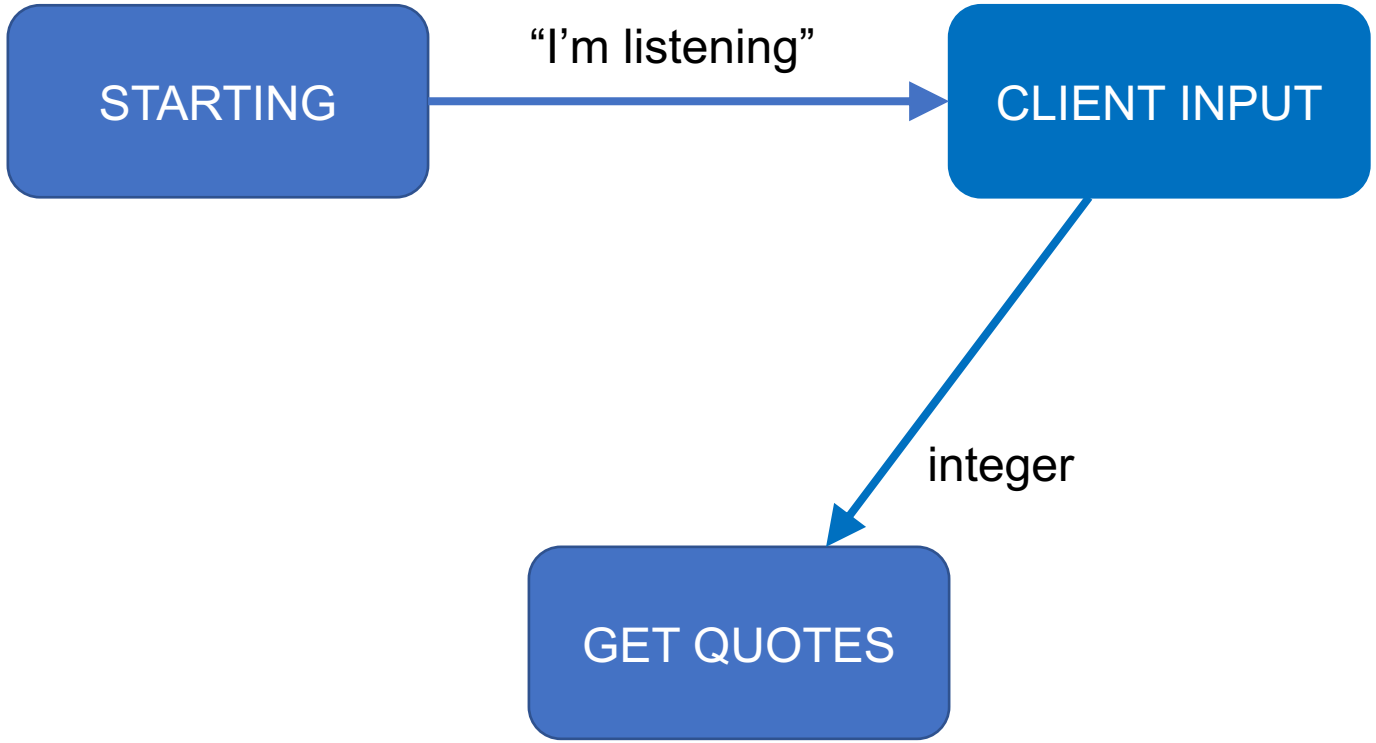
The Ron Swanson quote protocol



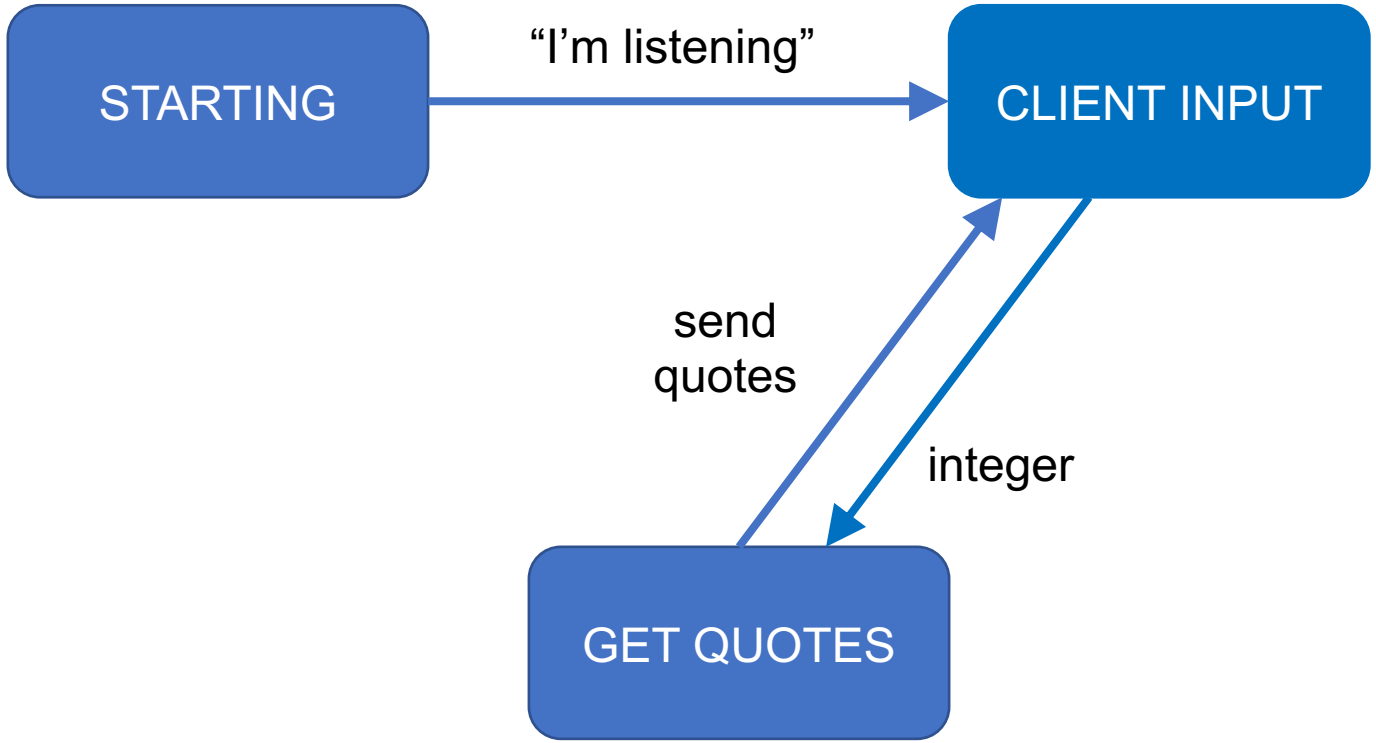
The Ron Swanson quote protocol



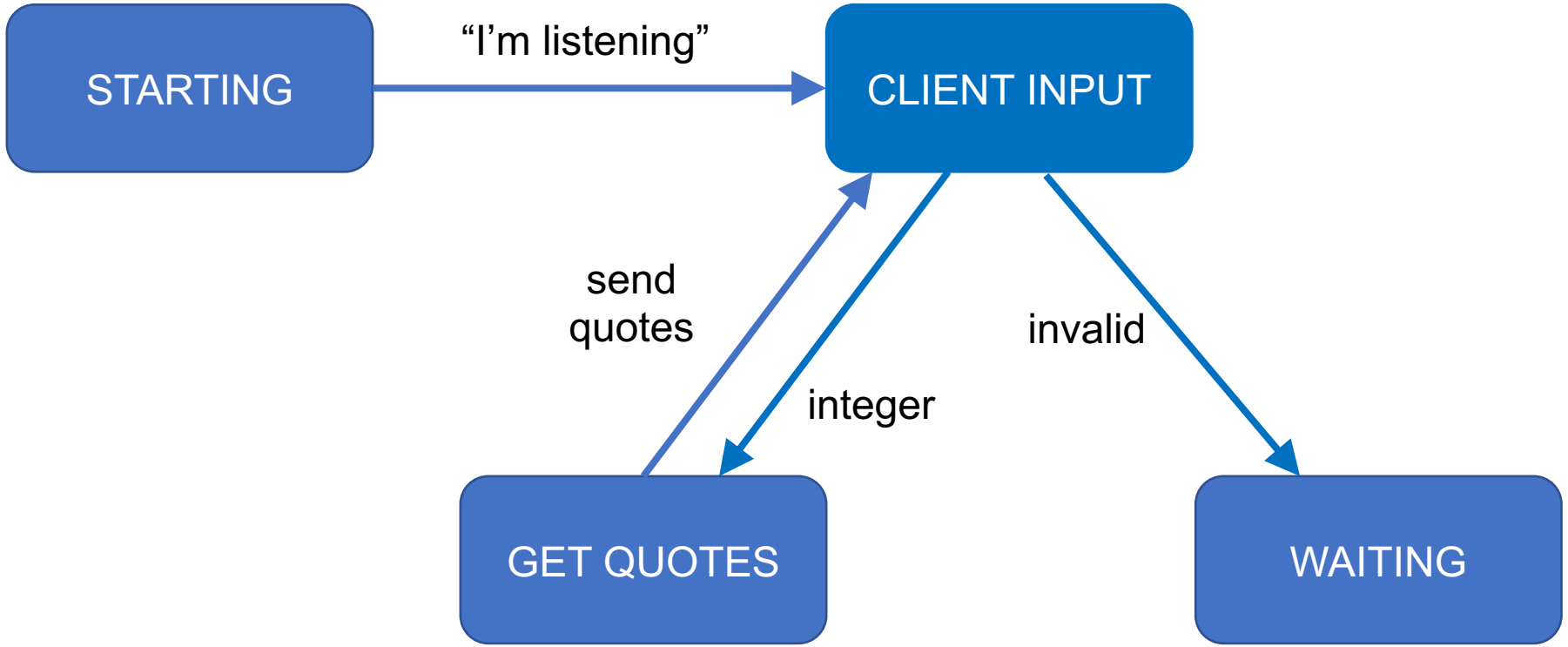
The Ron Swanson quote protocol



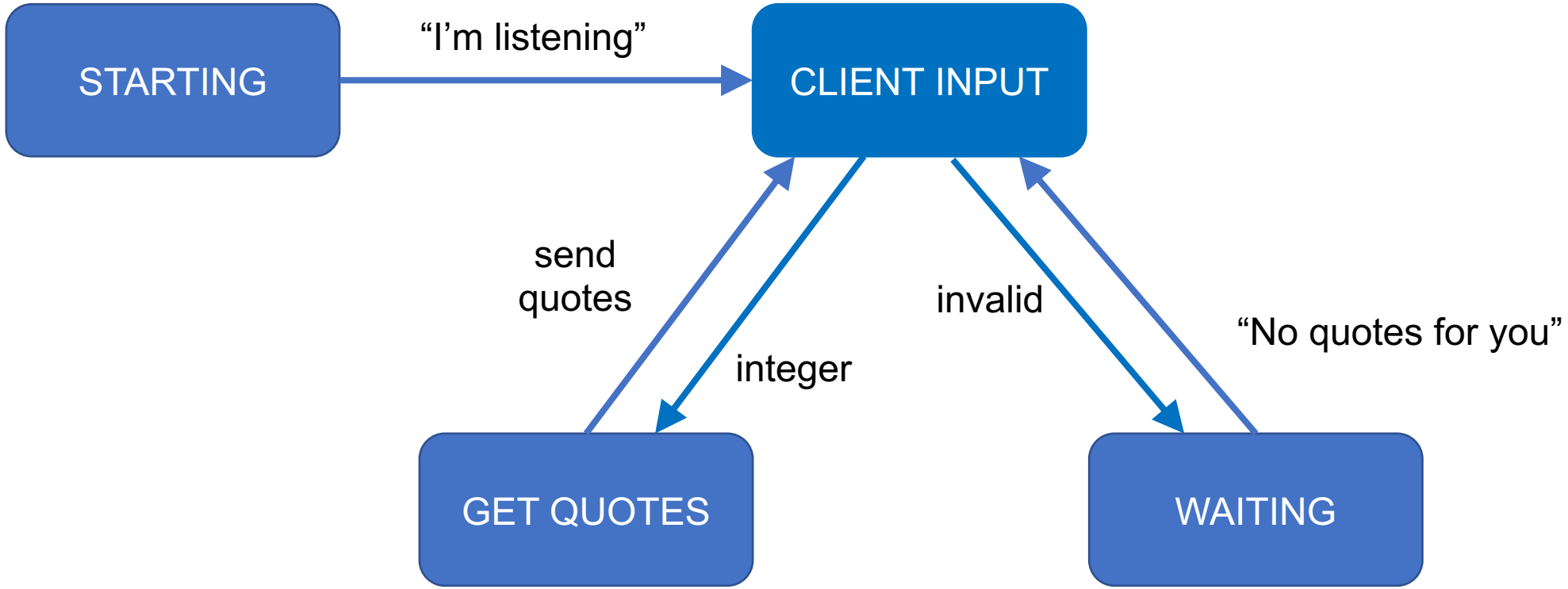
The Ron Swanson quote protocol



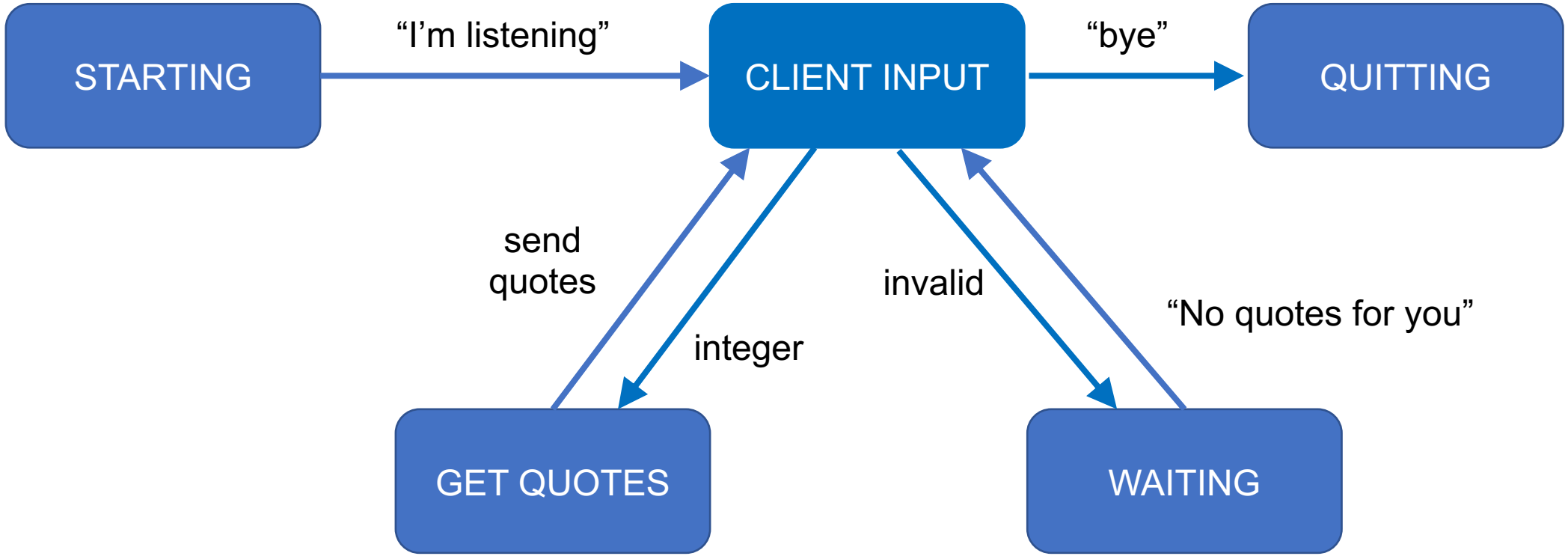
The Ron Swanson quote protocol



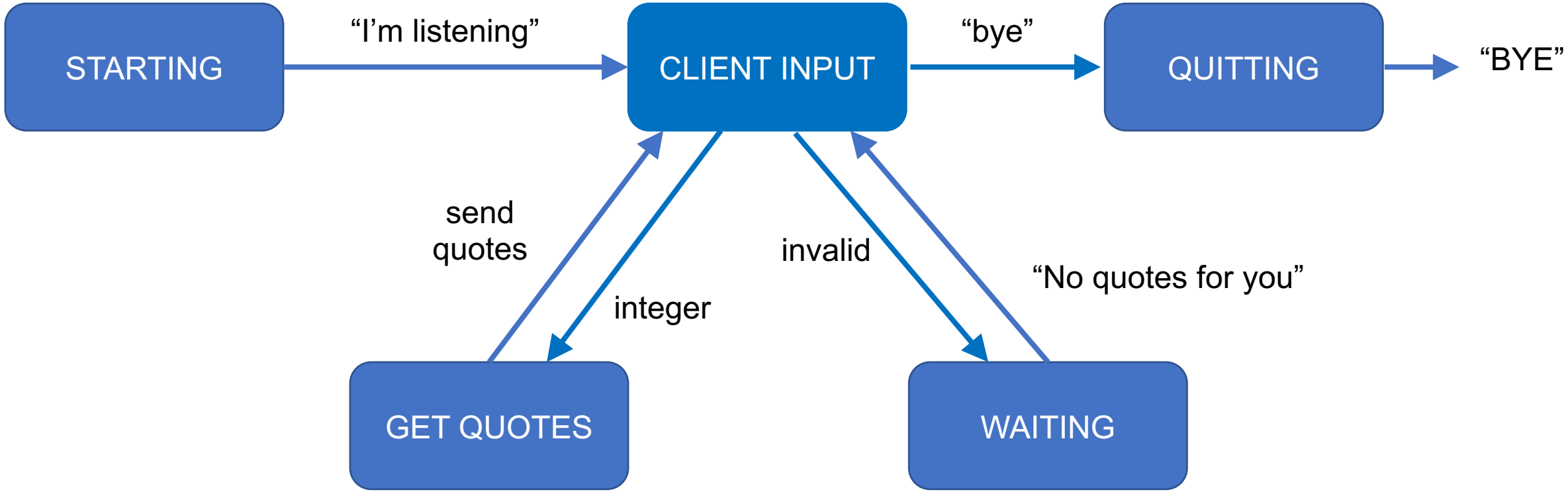
The Ron Swanson quote protocol



The Ron Swanson quote protocol



The Ron Swanson quote protocol



Code on GitHub

lecture-code > evening_section > Lecture14 > networking

- Client.java
- Server.java
- Protocol.java

To run:

- Code must be compiled (Build > Build Project)
- Open up two Terminal windows
- cd to project folder in build > java > main directory
- java networking.Server
- java networking.Client
- Follow prompts in client window

Break (10 mins)

Summary: Networking in Java

Via URL connection

- Create a URL object
- Establish a connection
- Make requests
- Process response
- Can either read directly, or establish a session an communicate

Via Sockets

- Direct connection to a server via a socket listening on a port
- Must follow agreed-upon protocol

Final exam prep

Final exam logistics

- Remote, computer-based exam – your individual work
- Submit your solutions by pushing them to your individual repos
  - Create a release!
- Use the course build.gradle
- Continue to use the typical organization and naming conventions that we've been using all semester

Final exam topics

- Three problems on the exam
- Cumulative, but the focus is on the second half of the semester
  - Java I/O
  - Regular expressions
  - Inner and nested classes
  - Interfaces comparable and iterable
  - Design of programs. SOLID principles
  - MVC architecture
  - Design patterns
  - Functional Java

  - Networking

Study resource

Finals from previous years:

Canvas > Modules > last module > practice finals