CS 10: Problem solving via Object Oriented Programming

Winter 2017

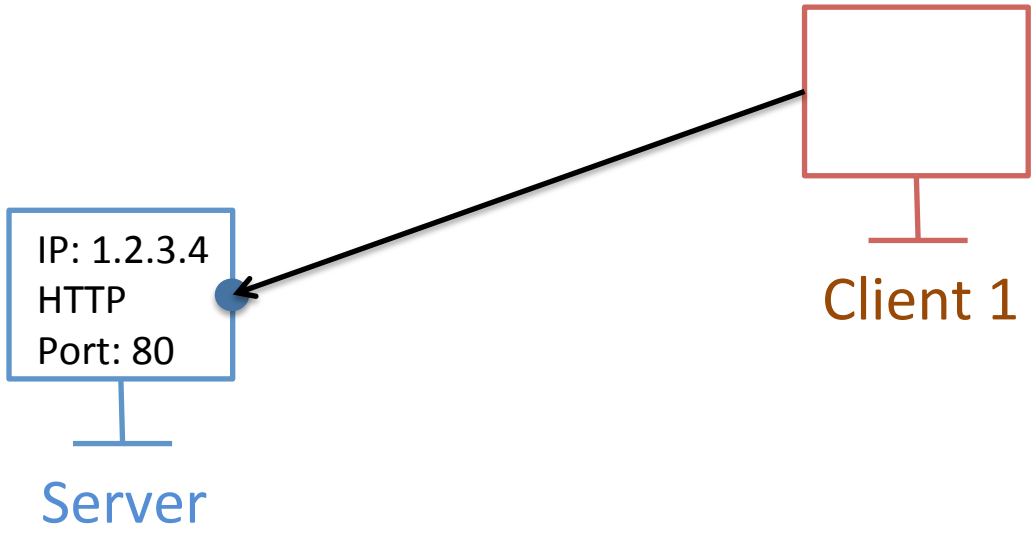
Tim Pierson
260 (255) Sudikoff

Day 22 – Client/Server

Agenda

1. Sockets
2. Server
3. Multithreaded server
4. Chat server

Sockets are a way for computers to communicate

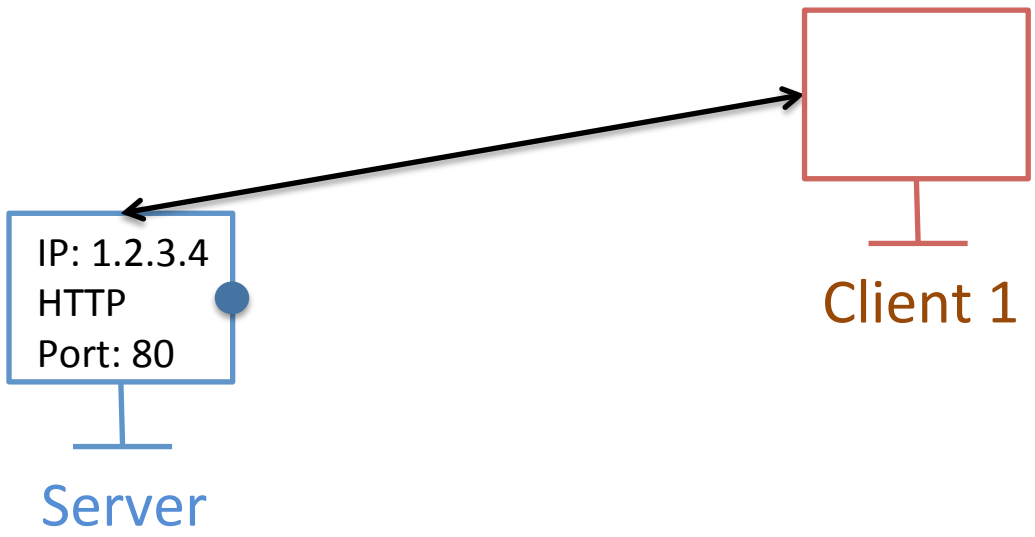


- Client 1 makes connection over socket
- Server receives connection, moves communications to own socket

Server is listening on a socket
(socket = IP address
+ protocol
+ port)

Port 80 = HTTP

Sockets are a way for computers to communicate

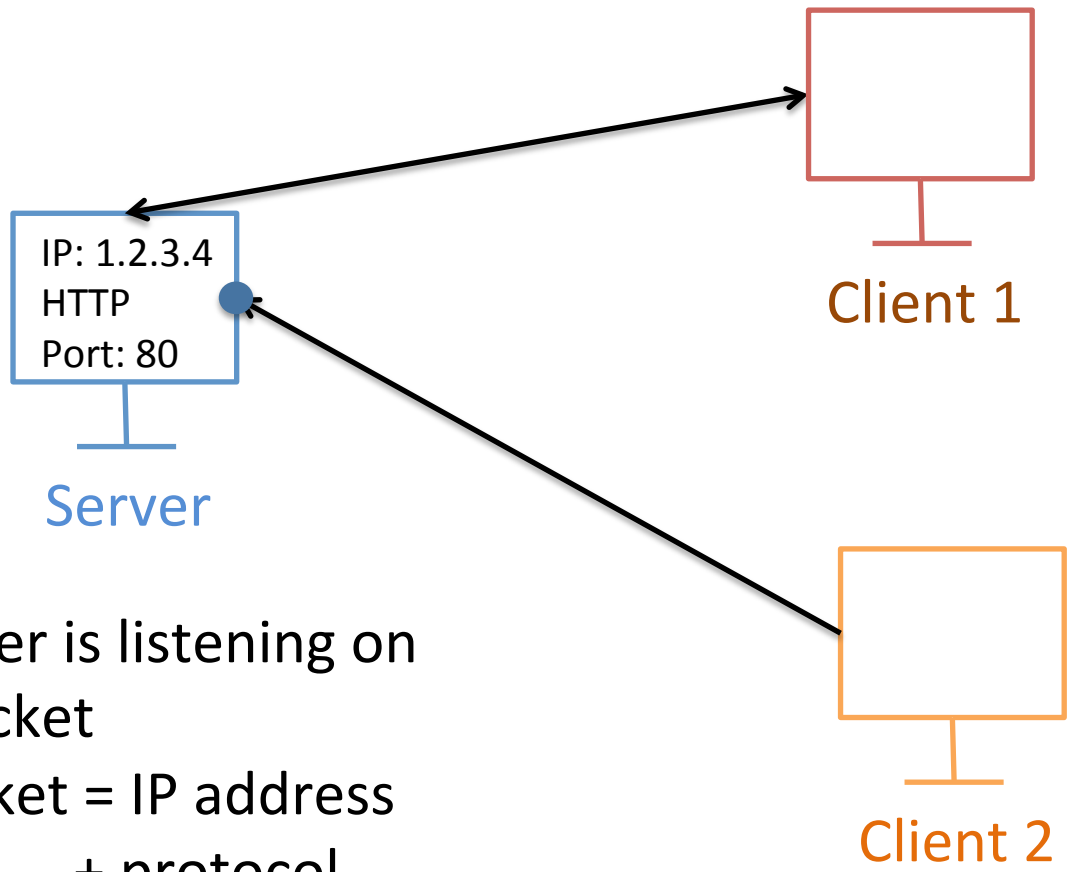


Server is listening on a socket
(socket = IP address
+ protocol
+ port)

Port 80 = HTTP

- Client 1 makes connection over socket
- Server receives connection, moves communications to own socket
- Server returns to listening
- Server talking to Client 1 and ready for others

Sockets are a way for computers to communicate

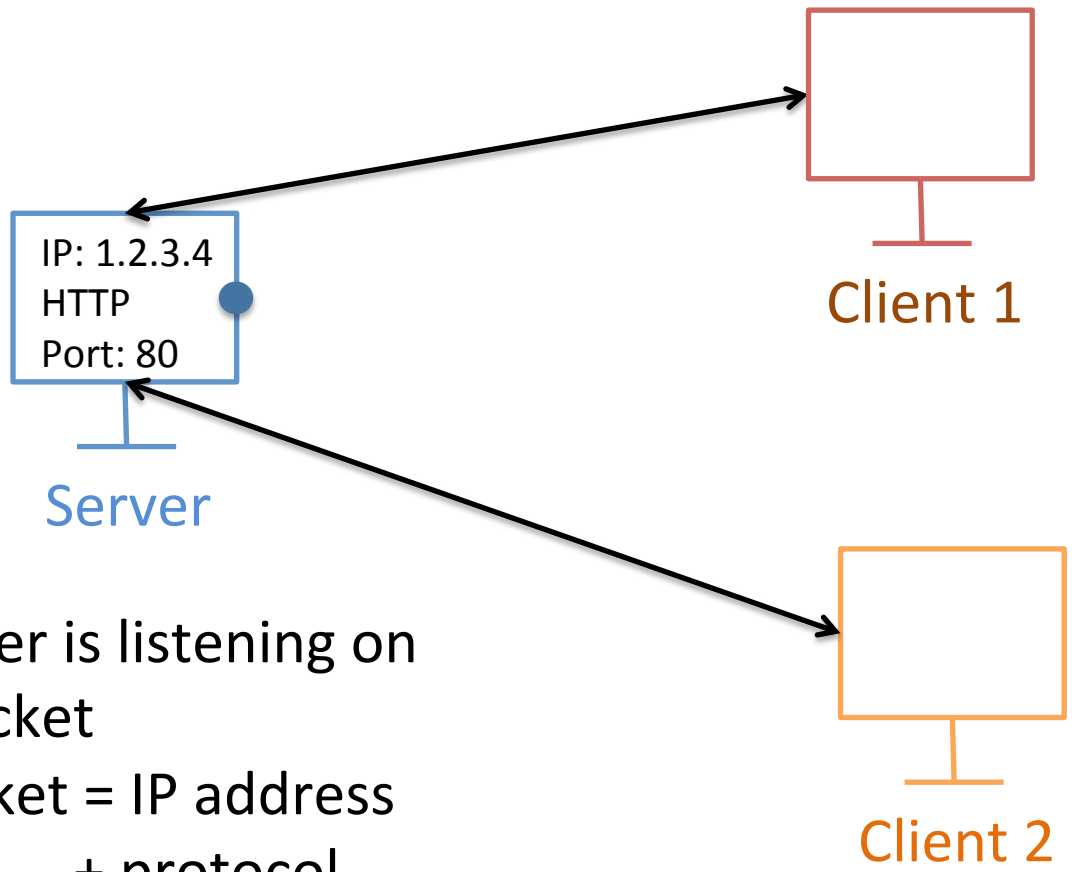


- Client 2 makes connection over socket

Server is listening on a socket
(socket = IP address
+ protocol
+ port)

Port 80 = HTTP

Sockets are a way for computers to communicate



Server is listening on a socket
(socket = IP address
+ protocol
+ port)

Port 80 = HTTP

- Client 2 makes connection over socket
- Server receives connection, moves communications to own socket
- Server returns to listening
- Server talking to client 1 and 2 ready for others

Java provides a convenient Socket class

WWWSocket.java

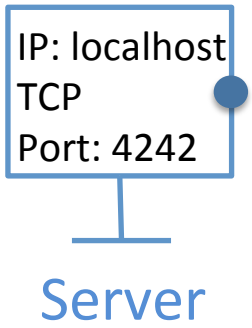
- Run, type ~tjp/cs10/index.php
- Output stream = from your computer to somewhere else
  - `out.println` sends data to another computer
- Input stream = from another computer to your computer
  - `in.readLine` reads data sent to your computer

Agenda

1. Sockets
2. Server
3. Multithreaded server
4. Chat server

We can create our own server

HelloServer.java

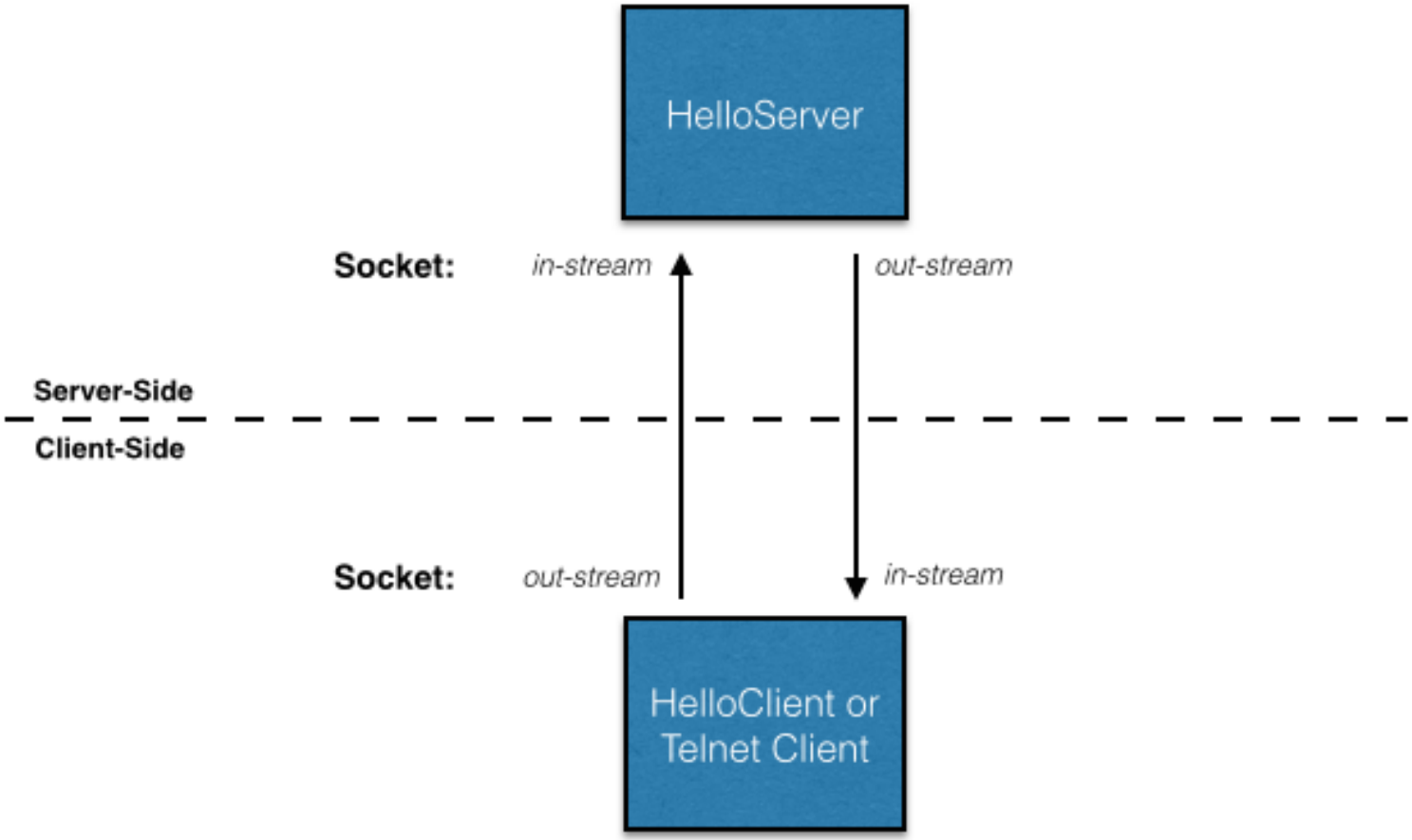



```
public static void main(String[] args) throws IOException {
        // Listen on a server socket for a connection
        System.out.println("waiting for someone to connect");
        ServerSocket listen = new ServerSocket(4242);
        // When someone connects, create a specific socket for them
        Socket sock = listen.accept();
        System.out.println("someone connected");

        // Now talk with them
        PrintWriter out = new PrintWriter(sock.getOutputStream(), true);
        BufferedReader in = new BufferedReader(new InputStreamReader(sock.getInputStream()));
        out.println("who is it?");
        String line;
        while ((line = in.readLine()) != null) {
              System.out.println("received:" + line);
              out.println("hi " + line + "!  anybody else there?");
        }
        System.out.println("client hung up");

        // Clean up shop
        out.close();
        in.close();
        sock.close();
        listen.close();
}
```

Run, then from terminal type telnet localhost 4242

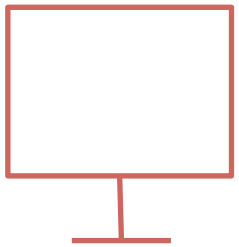
We can also create our own client too

HelloServer.java and HelloClient.java



We can create our own client too

HelloClient.java

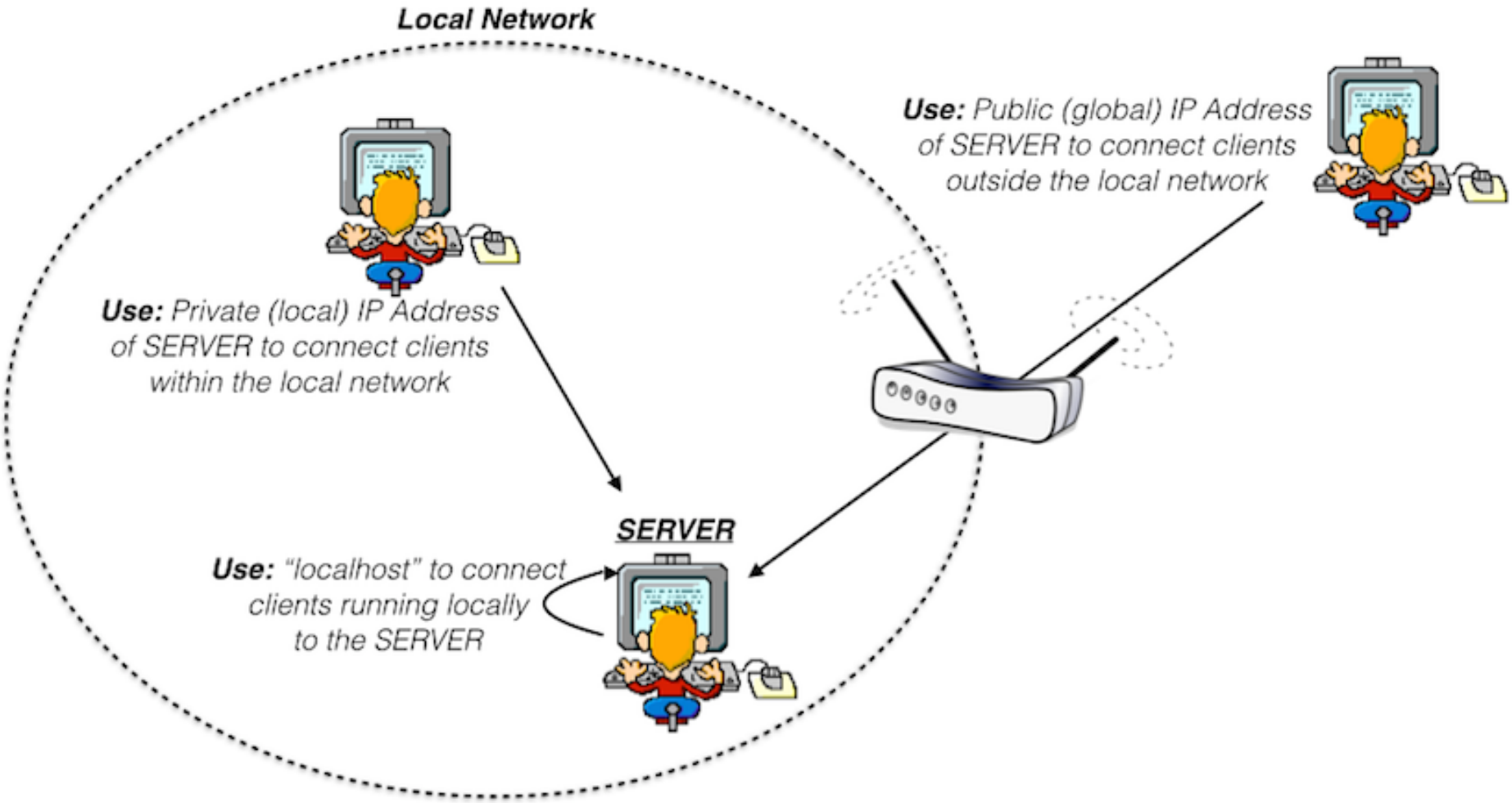



```
public class HelloClient {
      public static void main(String[] args) throws Exception {
            Scanner console = new Scanner(System.in);

            // Open the socket with the server, and then the writer and reader
            System.out.println("connecting...");
            Socket sock = new Socket("localhost",4242); //new Socket("129.170.212.159", 4242);
            PrintWriter out = new PrintWriter(sock.getOutputStream(), true);
            BufferedReader in = new BufferedReader(new InputStreamReader(sock.getInputStream()));
            System.out.println("...connected");

            // Now listen and respond
            String line;
            while ((line = in.readLine()) != null) {
            // Output what you read
                System.out.println(line);

            // Get some more input (from the user) to write to the open socket (server)
                  String name = console.nextLine();
                  out.println(name);
            }
            System.out.println("server hung up");

            // Clean up shop
            out.close();
            in.close();
            sock.close();
      }
}
```

Run HelloServer.java
Then run HelloClient.java

Friends can connect to your server if they connect to the right IP address

Run MyIPAdressHelper.java to get your address, edit HelloClient.java



Connecting from another machine

HelloServer.java and HelloClient.java

- Run MyIPAddressHelper on server to get IP
- Start HelloClient.java on server
- Edit HelloClient.java to change localhost to server IP address
- Run HelloClient on client machines and make connection
- Connect from student machine?

Agenda

1. Sockets
2. Server
3. Multithreaded server
4. Chat server

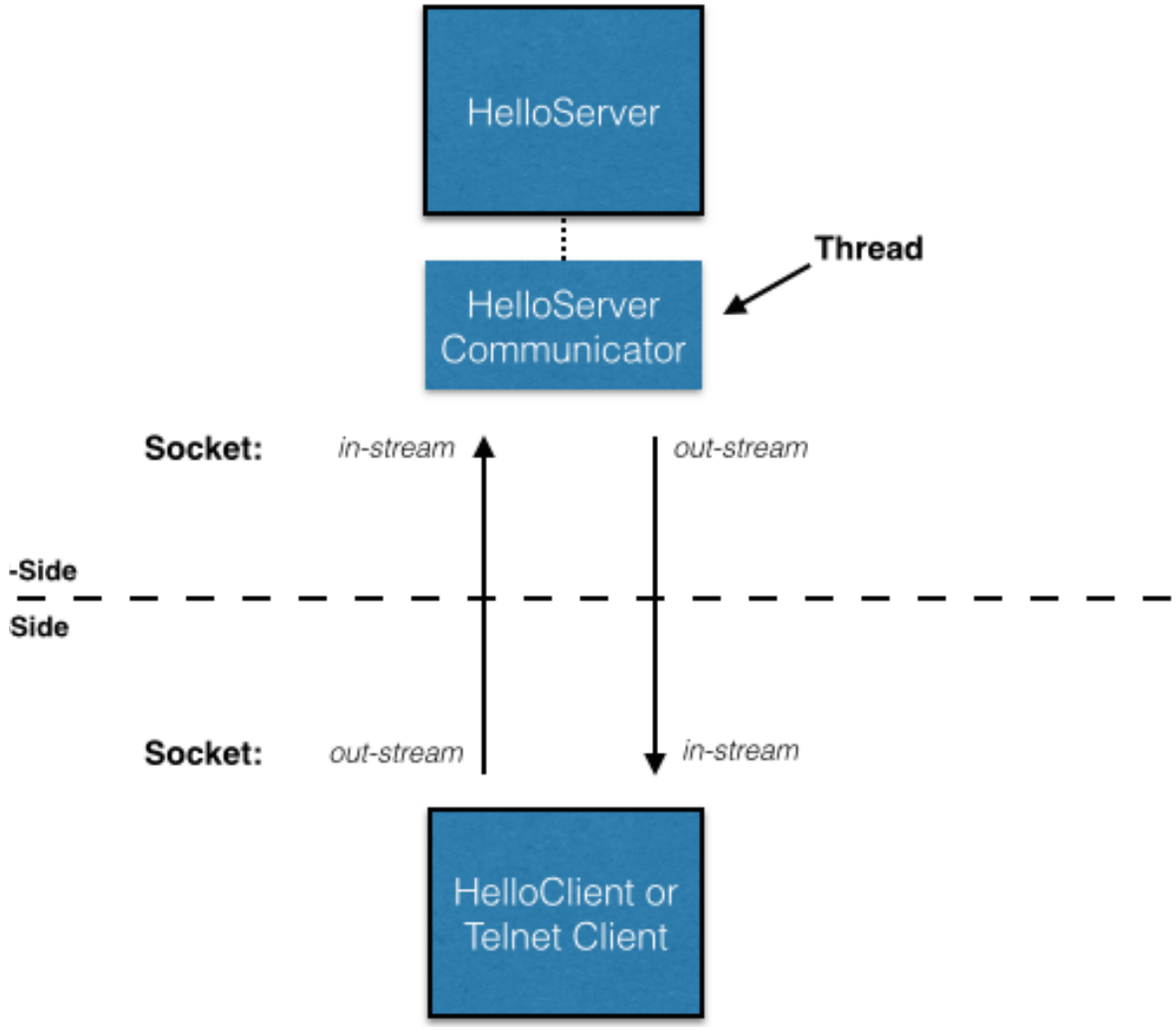
Currently our server can only handle one client at a time

Using Java's Thread mechanism to overcome single client issue

- We would like our server to talk to multiple clients at a time
- Trick is to give each client its own socket
- That way the server can talk "concurrently" with multiple clients
- Java provides a Thread class to handle concurrency (multiple processes running at same time)
- Threads are much lighter than running multiple instances of a program (more on threads next class)
- Inherit from Thread class and override `run` method
- Start thread using `start` method

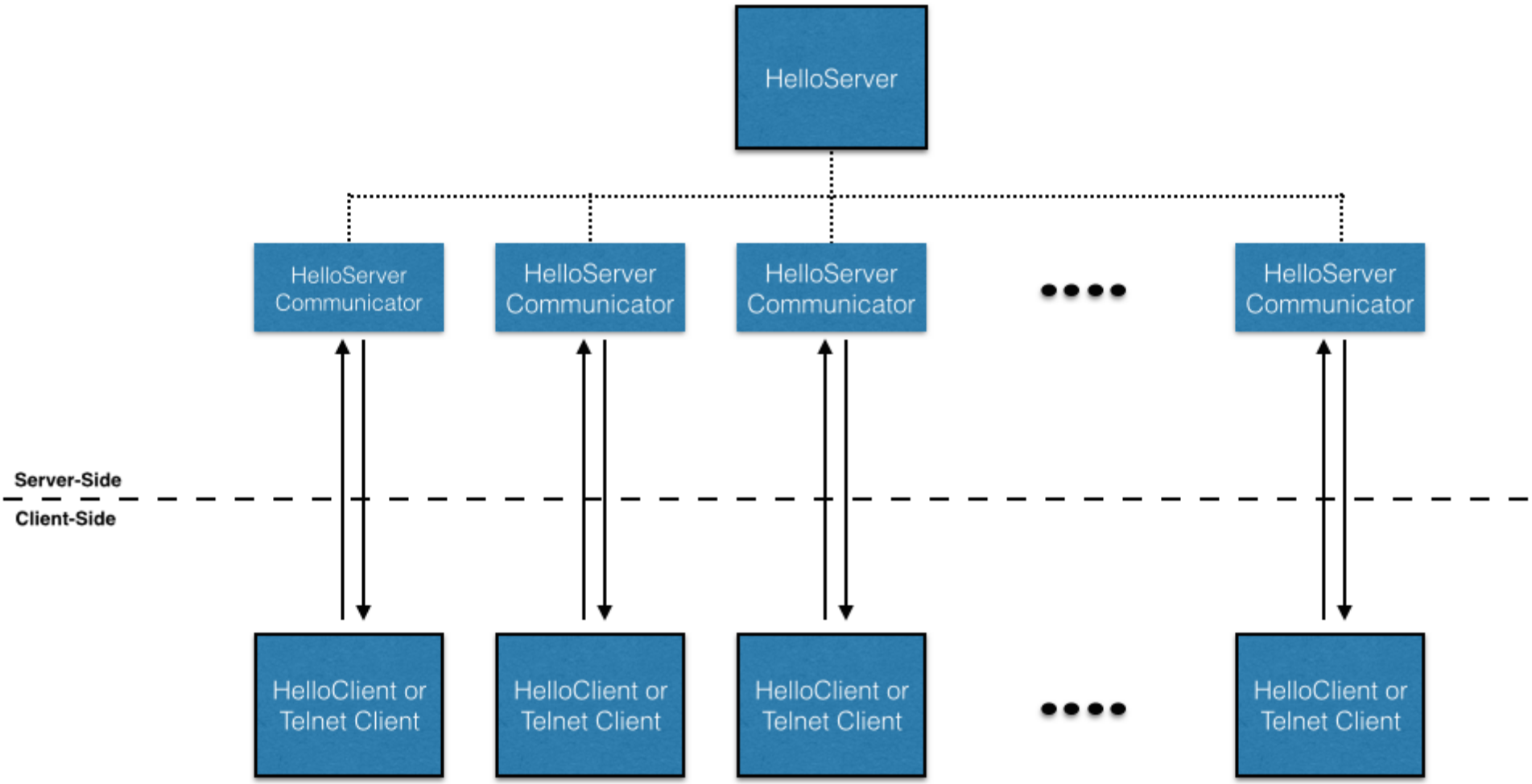
We can create a "Communicator" on a separate thread for each connection

One Communicator allocated for a single client



We can create a "Communicator" on a separate thread for each connection

Multiple Communicators allocated for multiple clients



We can create a “Communicator” on a separate thread for each connection

HelloMultithreadedServer.java

- Starts new thread with new HelloServerCommunicator on each connection

HelloServerCommunicator.java

- Extends Thread
- Override run
- Tracks thread ID
- Otherwise the same as single threaded version

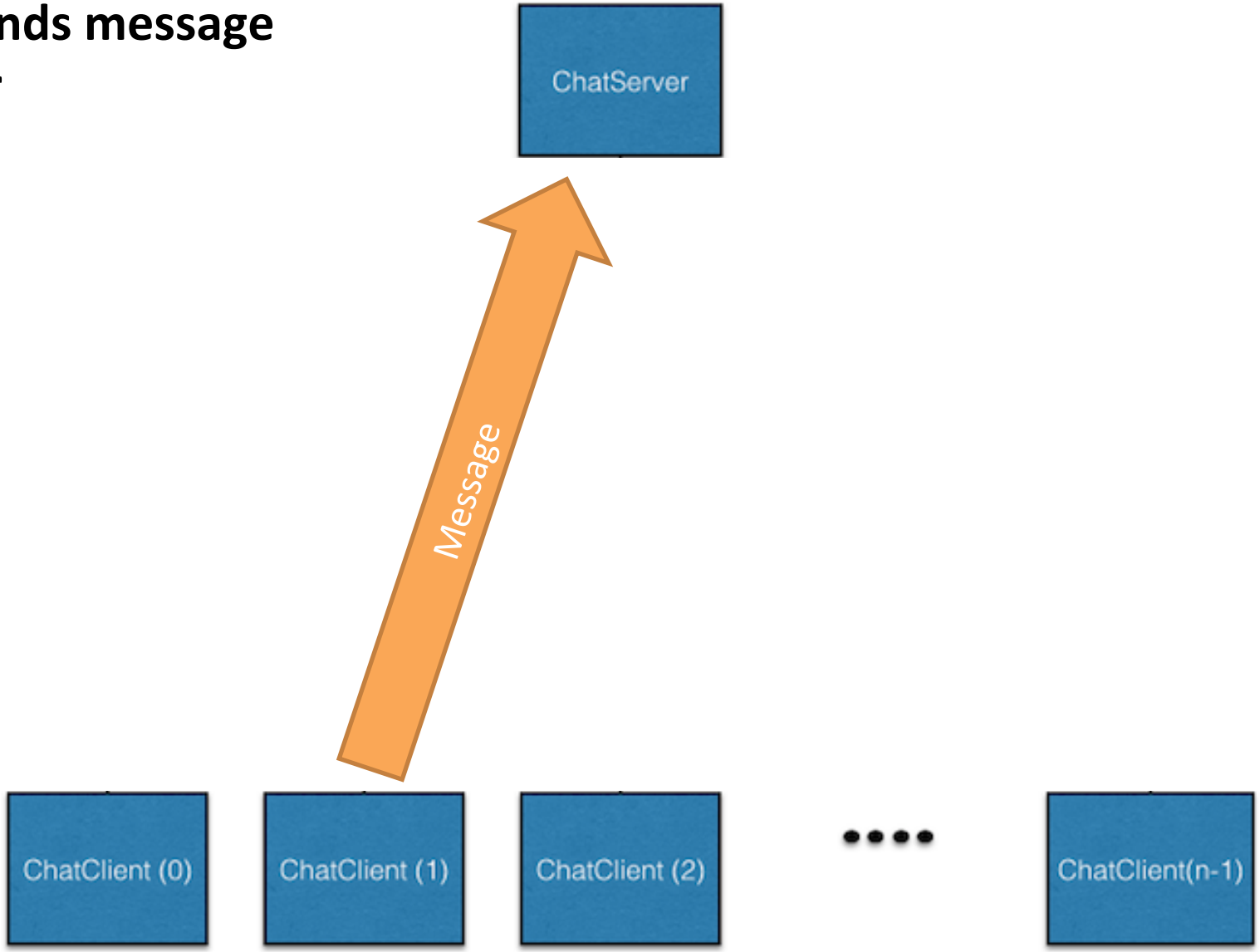
Run HelloMultithreadedServer.java with multiple telnets

Agenda

1. Sockets
2. Server
3. Multithreaded server
4. Chat server

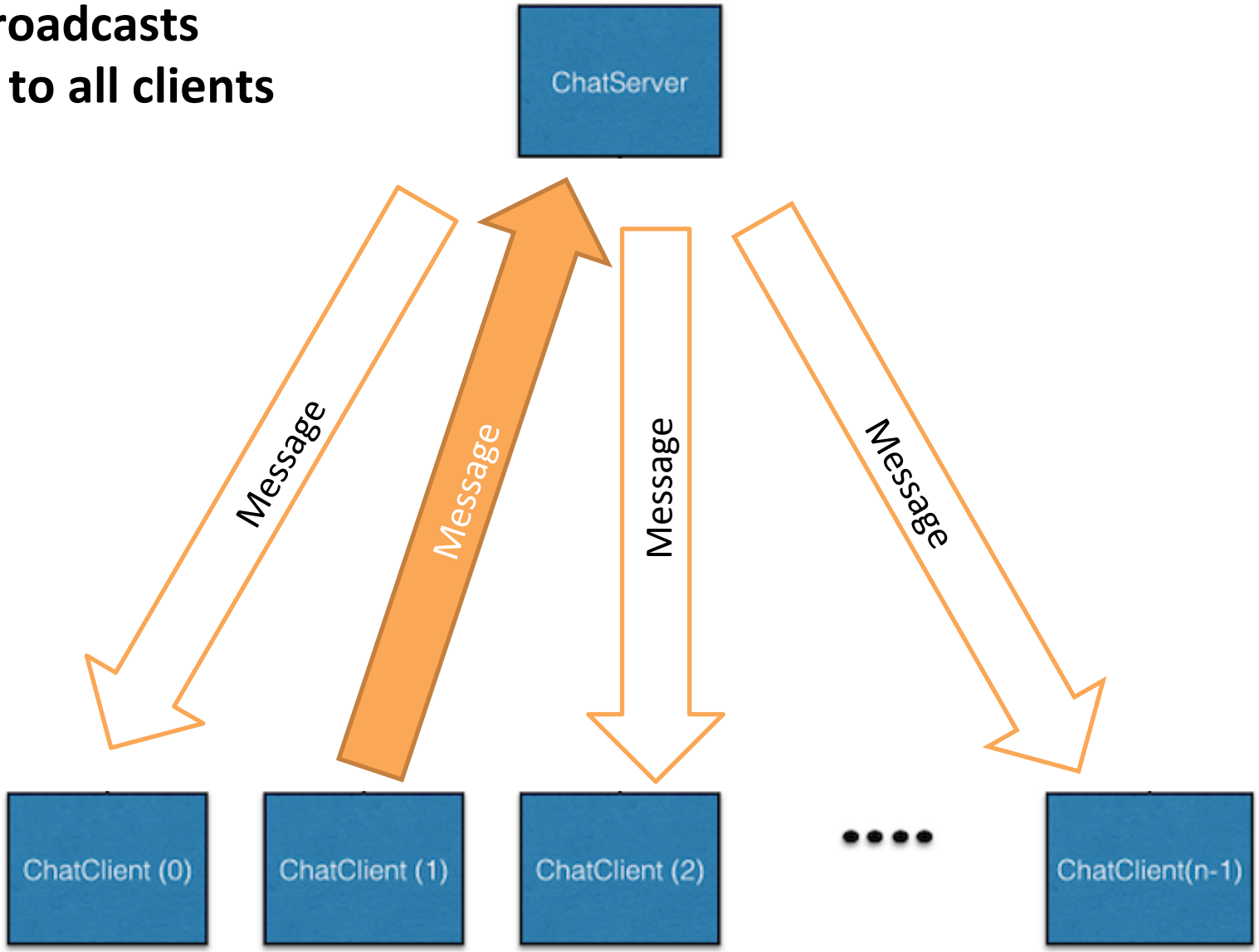
Goal: Chat server allows communication between multiple clients

Client sends message to server



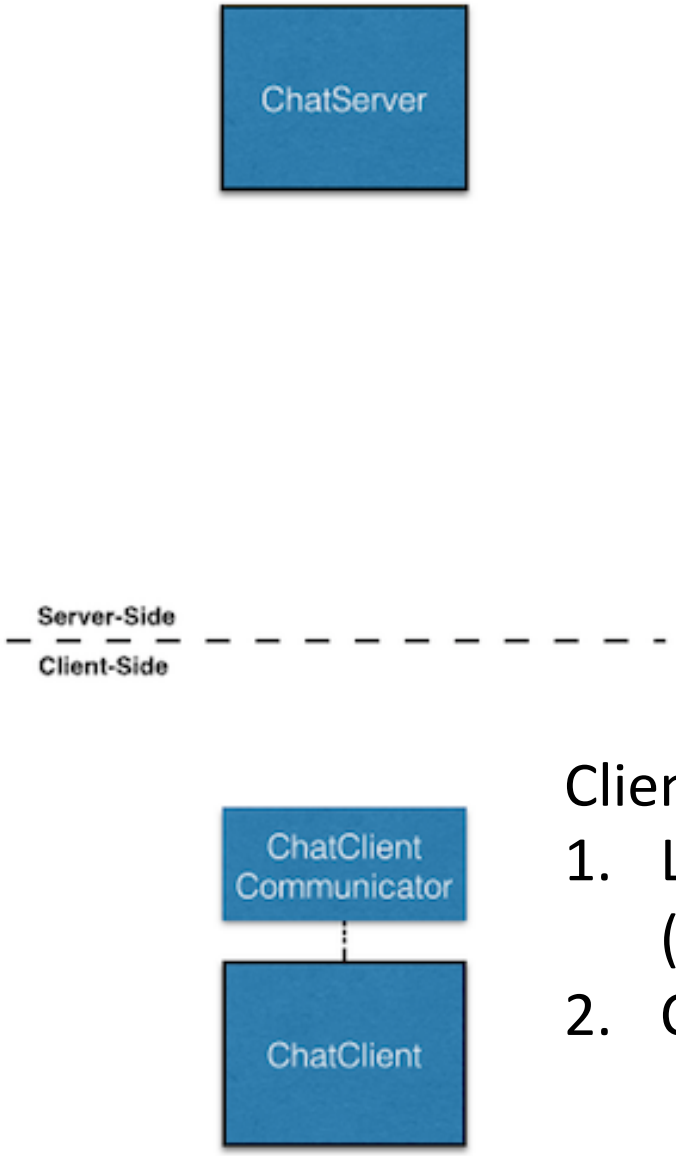
Goal: Chat server allows communication between multiple clients

Server broadcasts message to all clients



Client listens for keyboard on main thread creates Communicator on second thread

Client

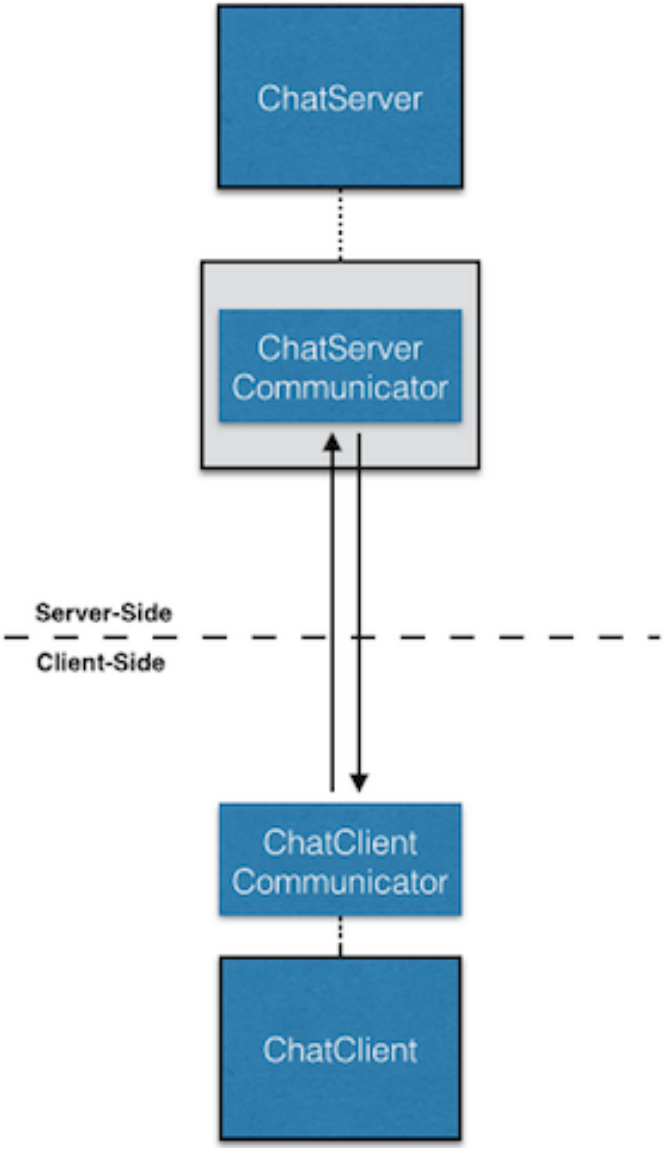


Client uses two threads:

1. Listen for keyboard input (blocks in between entries)
2. Communicates with server

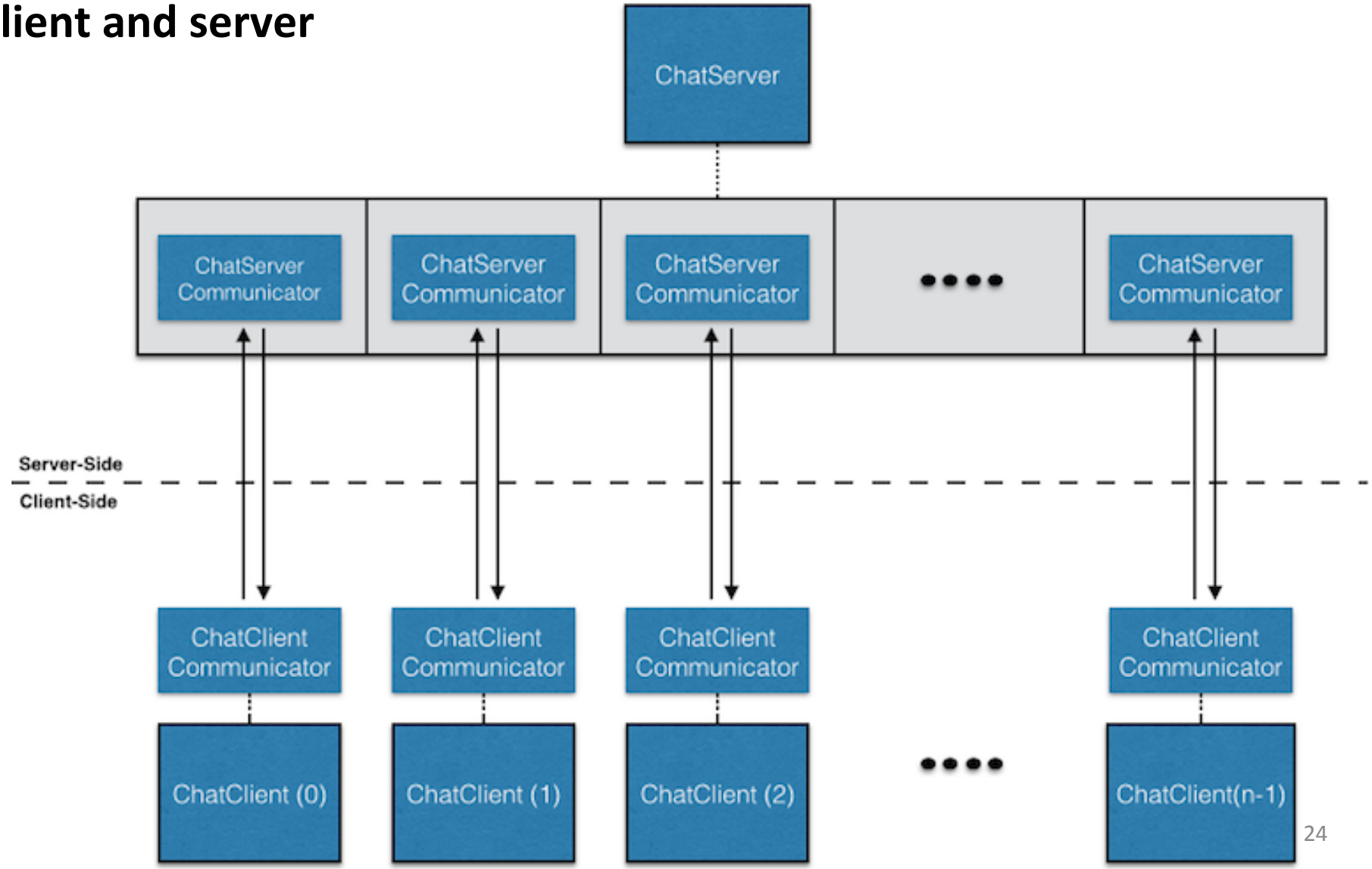
ChatServer creates a Communicator for each client

Server



ChatServer handles multiple clients and broadcasts message to each client

Client and server



ChatServer handles multiple clients and broadcasts message to each client

ChatServer.java

- Starts thread with `ChatServerCommunicator` on each connection
- Tracks all new threads in `comms` ArrayList of `ChatServerCommunicators`
- Calls `send` method on each `ChatServerCommunicator` when messages arrive from any client (except self)
- Provides `add` and `removeCommunicator` methods for `ChatServerCommunicator` to call

ChatServerCommunicator.java

- Similar to `MultithreadedServerCommunicator`
- Tracks `ChatServer` that started it
- Has `send` method to output messages sent by Server
- Calls `broadcast` on `ChatServer` when new message typed
- Calls remove on `ChatServer` when client hangs up

ChatServer handles multiple clients and broadcasts message to each client

ChatClient.java

- Starts thread with `ChatClientCommunicator`
- Listens for keyboard input on main thread
- Gets name as first input
- Sends subsequent keyboard input to Server via `ChatClientCommunicator` `send` method

ChatClientCommunicator.java

- Tracks `client` that created it
- Listens for incoming messages and outputs to console in `run`
- `send` method sends console text entered by keyboard to Server for broadcast

We can build a Chat server that will broadcast messages to all clients

Chat server

- Client connects to server and gives name
- Server now broadcasts messages to all clients, attributing message to client name
- Server side works similarly to HelloMultithreadedServer.java, but keeps track of all threads it creates using `comms` ArrayList of `Communicators`
- Communicators are removed if client hangs up
- Each communicator has a `send` method that the server can call to send a message to it
- Adding and removing communicators use `synchronized` to make sure only one talks at a time (more about this next class)

Clients must listen for both keyboard input and message from server

Chat client

- Clients use two threads
  - Main thread listens for keyboard input
  - Second thread listens for messages from Server
- Create a “communicator” for the client side