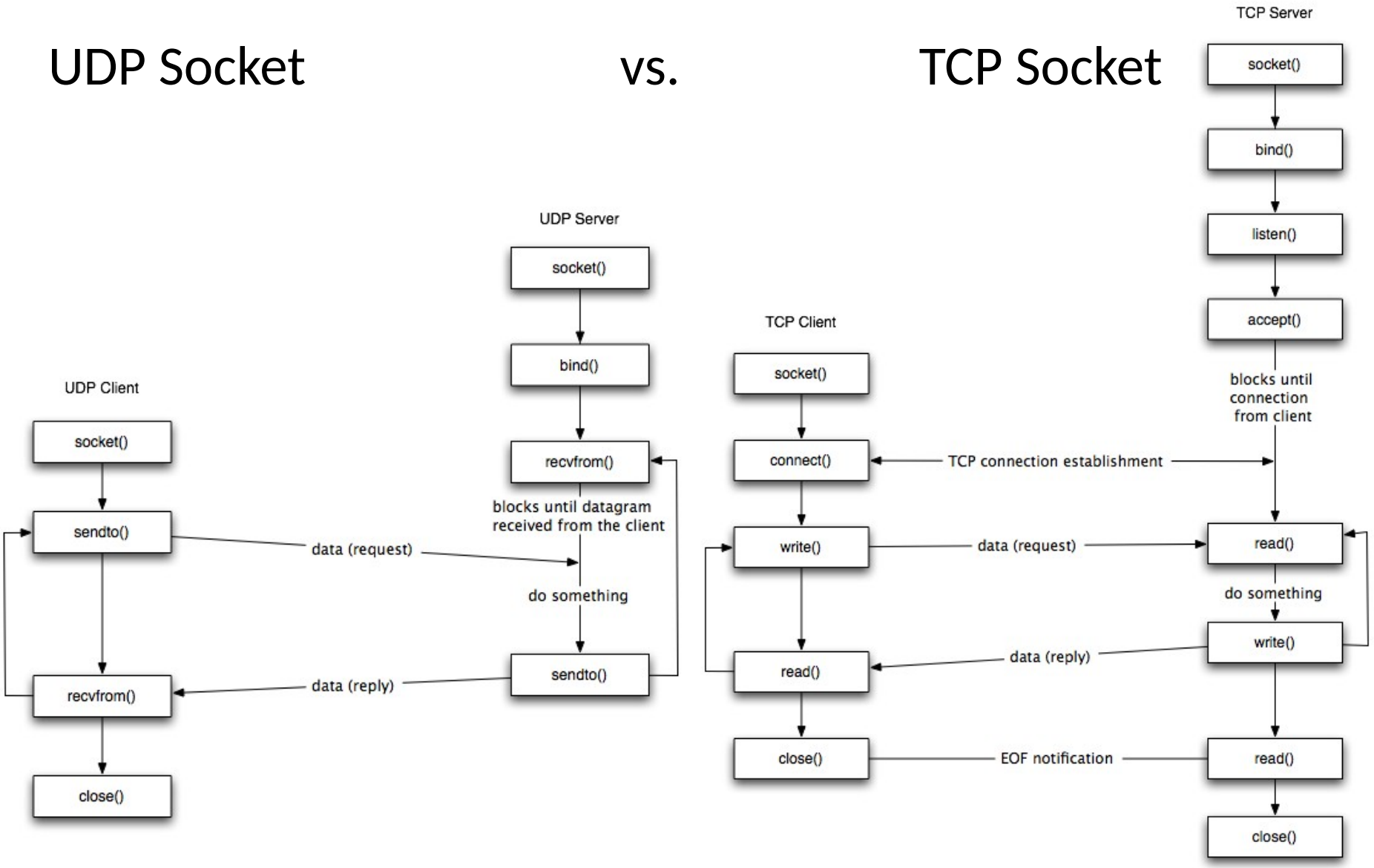
Fall 17 Computer Networks
PG3 Tutorial

https://www3.nd.edu/~dwang5/courses/fall18/

Review: TCP vs UDP

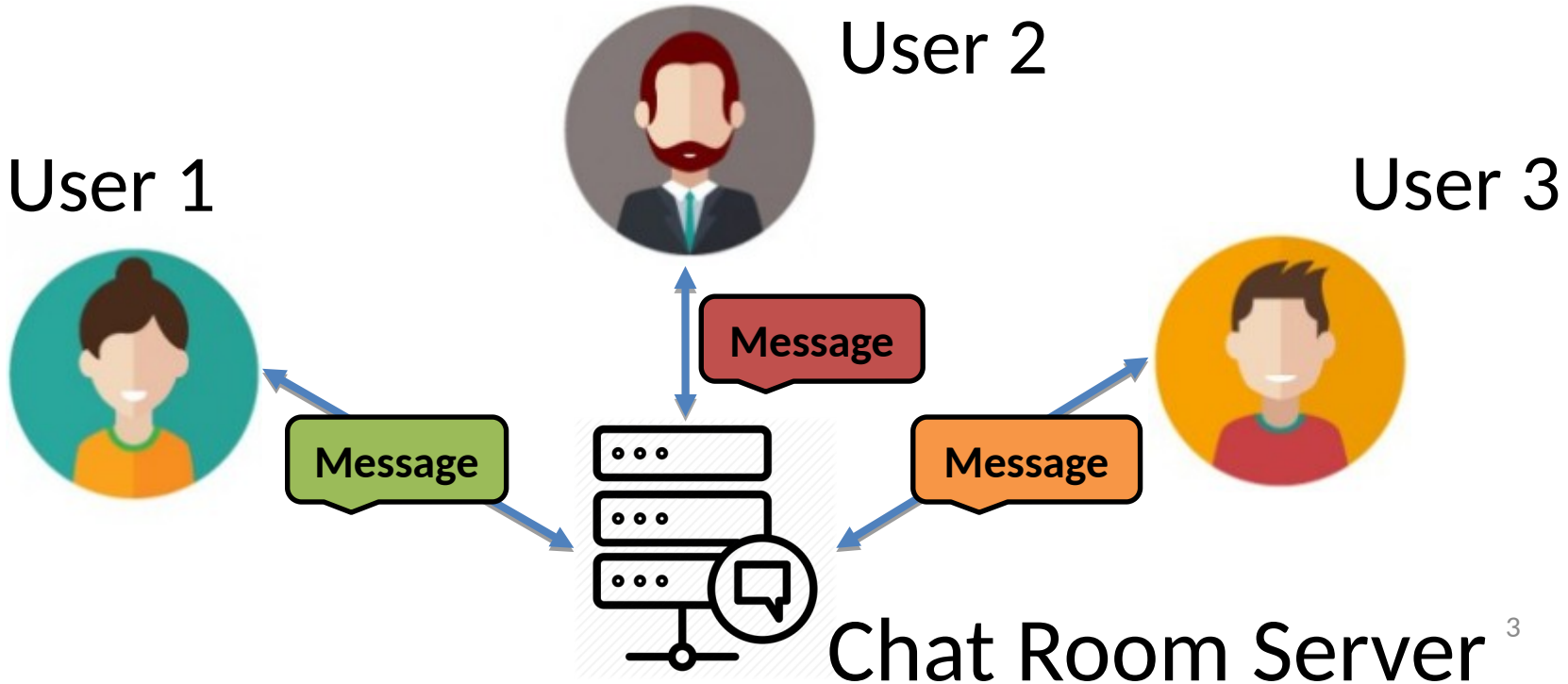
UDP Socket vs. TCP Socket



Source: http://www.cs.dartmouth.edu/~campbell/cs60/socketprogramming.html

PG3 – Overview

- In PG4, you will build a **prototype of online chat room**.
- This program allows multiple users to register and log on to a chat room server simultaneously.
- The users can broadcast messages to all users (**public mode**), or send private message (**direct mode**) to a specific user.

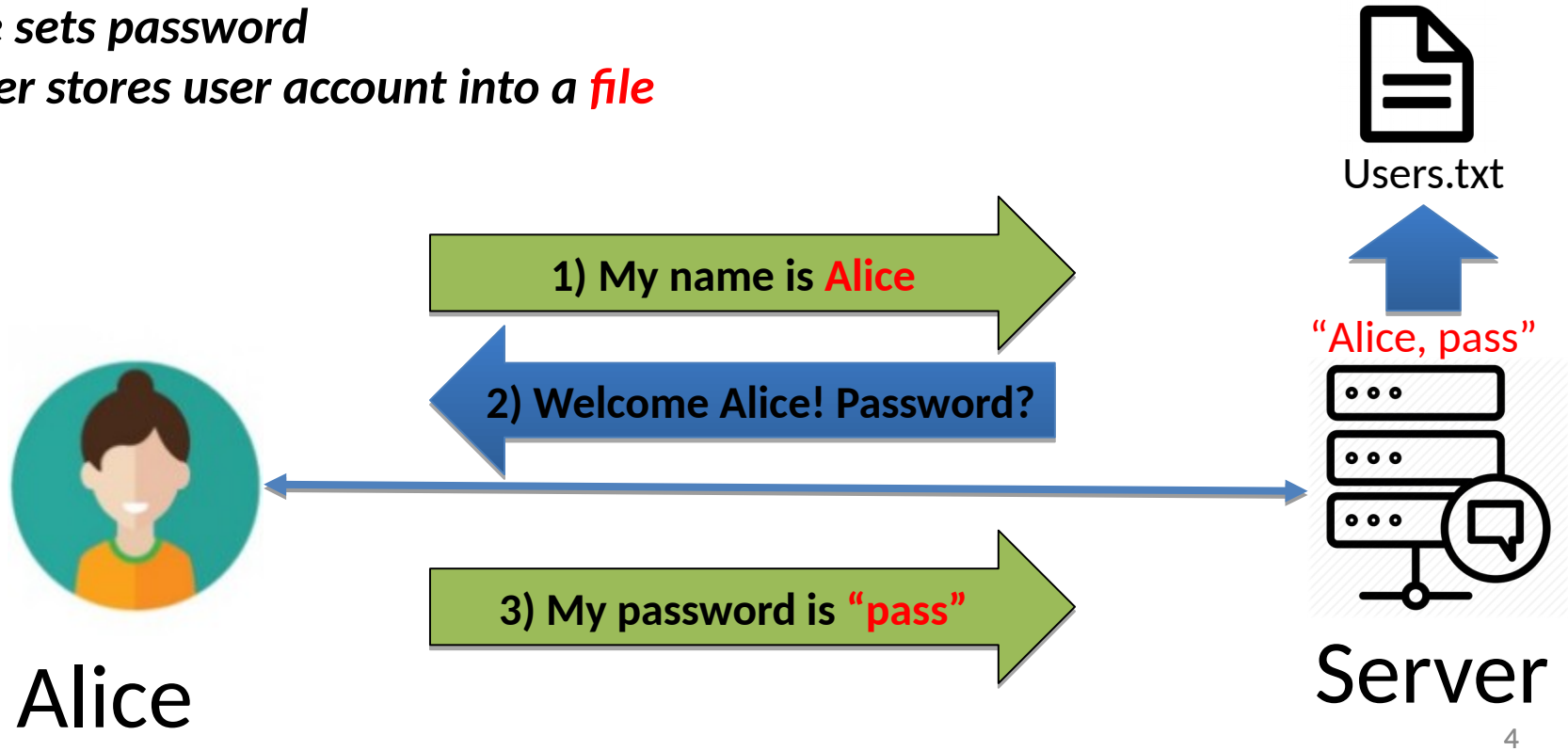


PG3 – User Registration

User first needs to register an account by setting up a **username** and **password**.

Registration Protocol:

1) Alice connects to server: ./chatclient student00.cse.nd.edu 41100 Alice
2) Server checks New User? Yes - > Prompt for password
3) Alice sets password
4) Server stores user account into a file

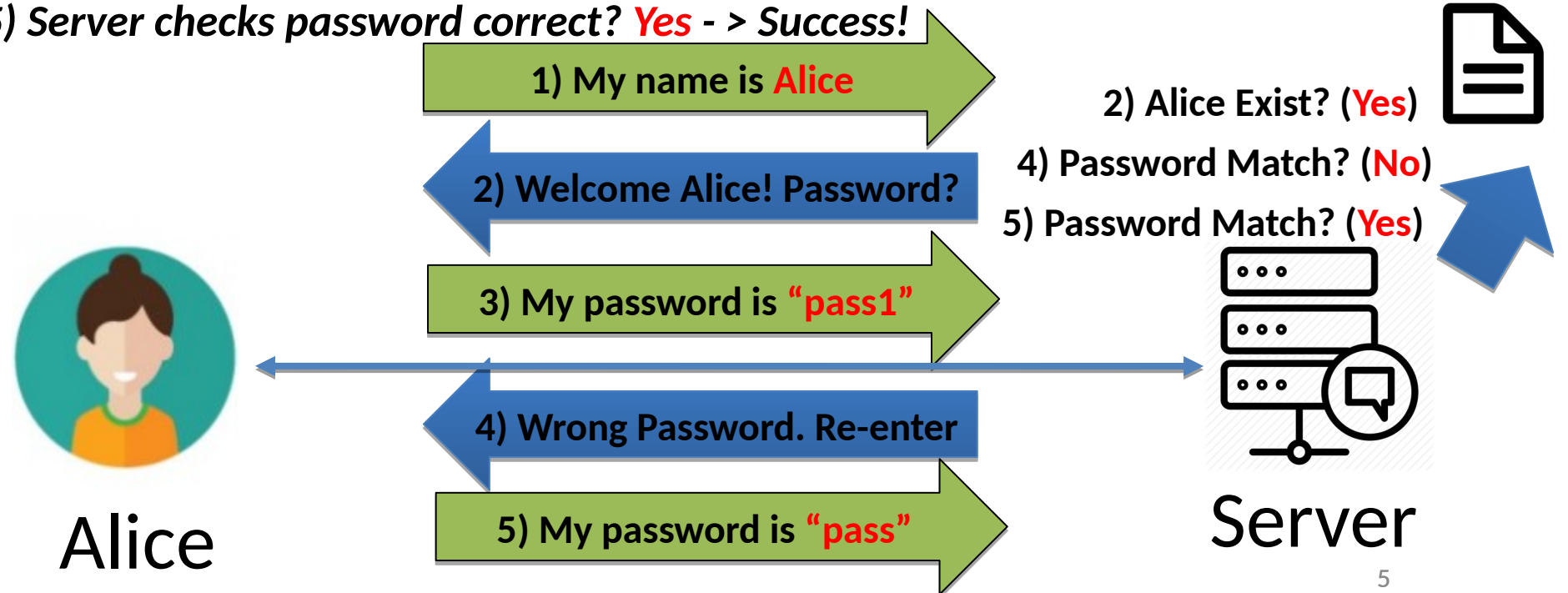


PG3 – User Login

A registered user can login with his/her credentials.

Login Protocol:

1) Alice connects to server: ./chatclient student00.cse.nd.edu 41100 Alice
2) Server checks Existing User? Yes - > Prompt for password
3) Alice enters password
4) Server checks password correct? No - > Prompt for password again
5) Server checks password correct? Yes - > Success!

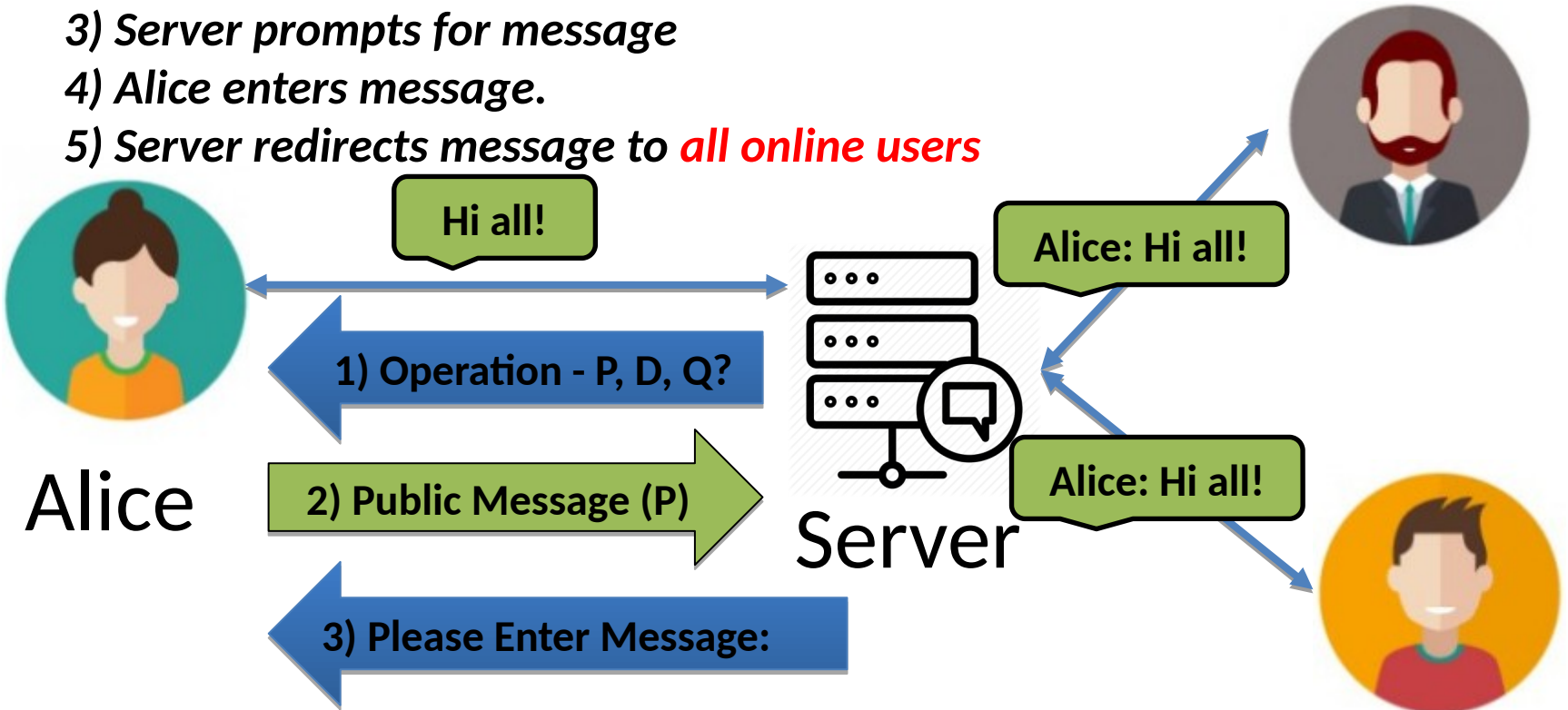


PG3 – Public Messaging

A logged in user can send broadcasting messages.

Public Messaging Protocol:

1) Server prompts for operation P for public message, D for direct messaging, Q for quit
2) Alice enters P
3) Server prompts for message
4) Alice enters message.
5) Server redirects message to all online users



PG3 – Direct Messaging

A logged in user can send private messages to a specific user.

Direct Messaging Protocol:

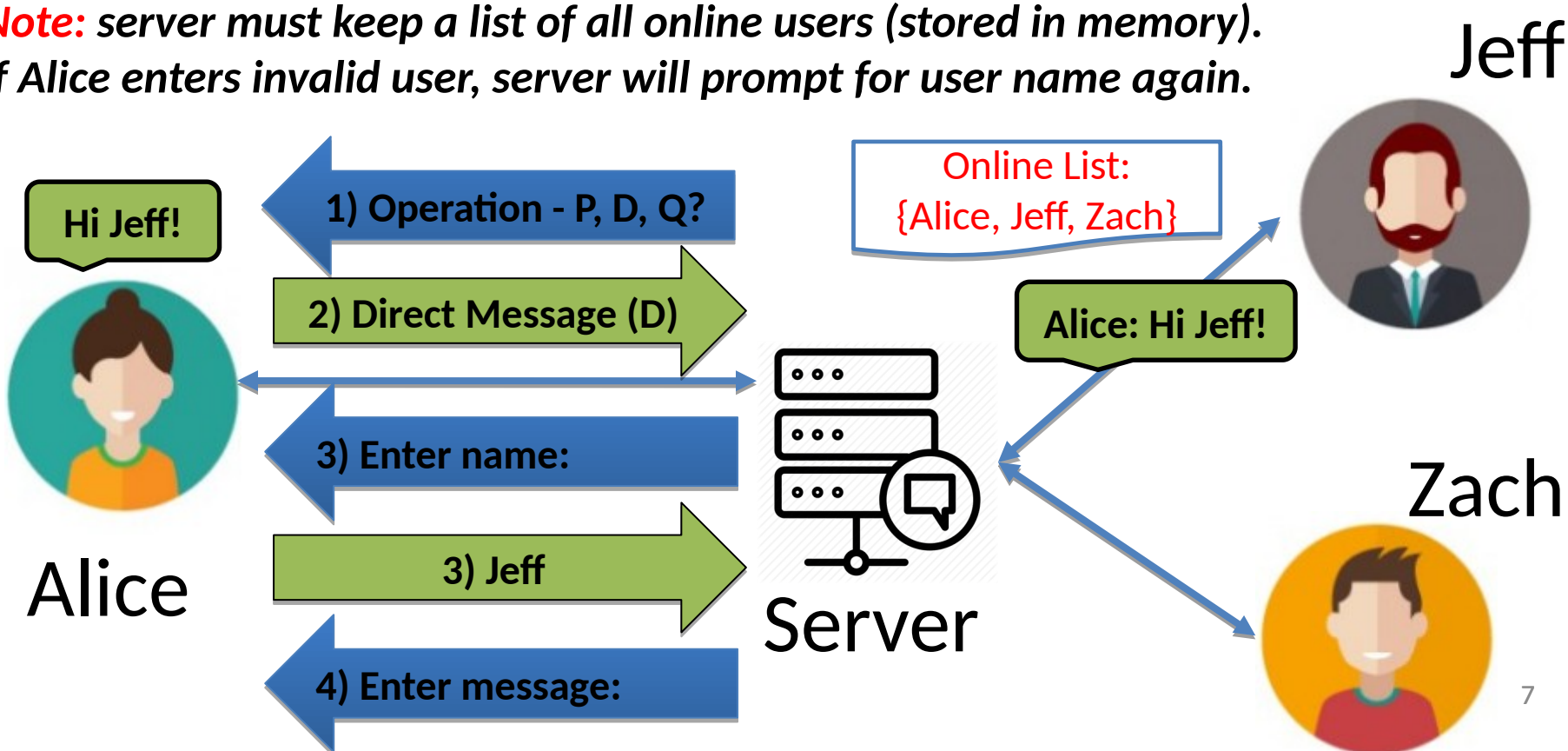
1) Server prompts for operation P for public message, D for direct messaging, Q for quit

2) Alice enters D

3) Server prompts for username then 4) message

Note: server must keep a list of all online users (stored in memory).

If Alice enters invalid user, server will prompt for user name again.

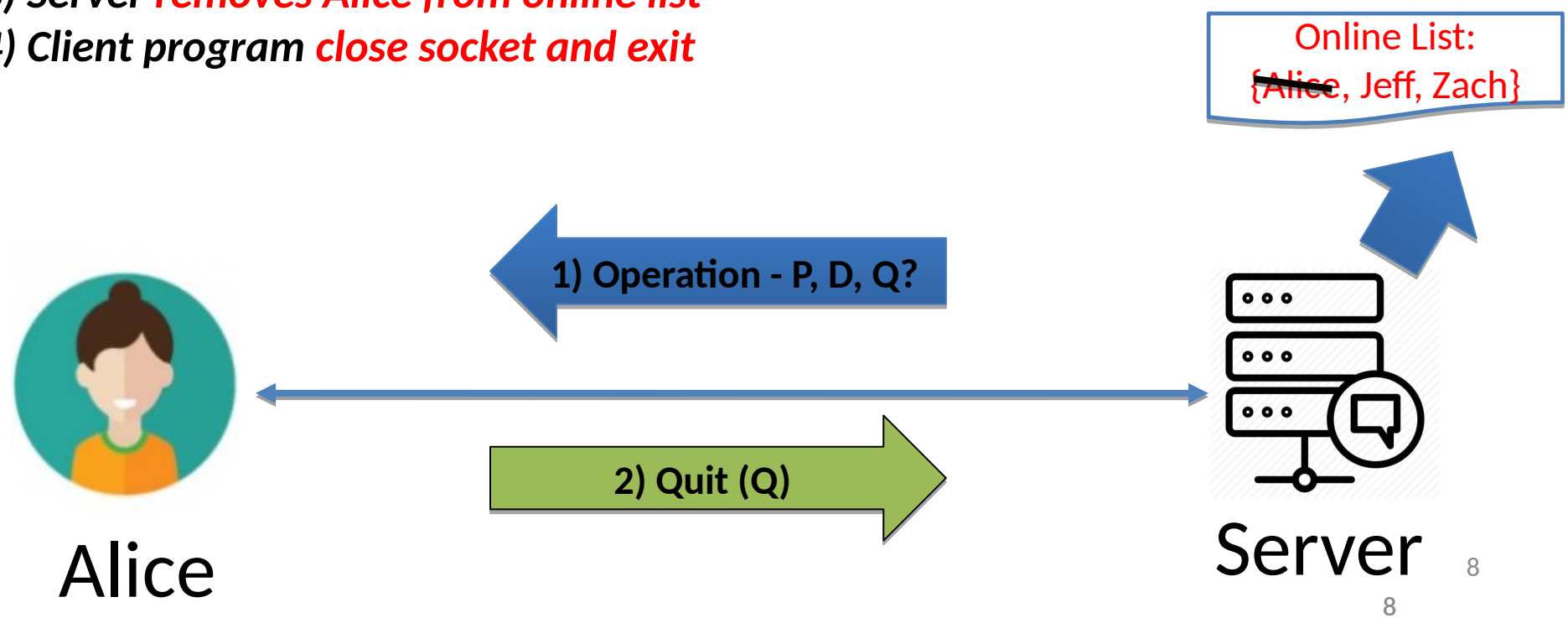


PG3 – User Exit

A logged in user can exit the program by using the E command.

Exit Protocol:

1) Server prompts for operation P for public message, D for direct messaging, Q for quit
2) Alice enters Q
3) Server removes Alice from online list
4) Client program close socket and exit



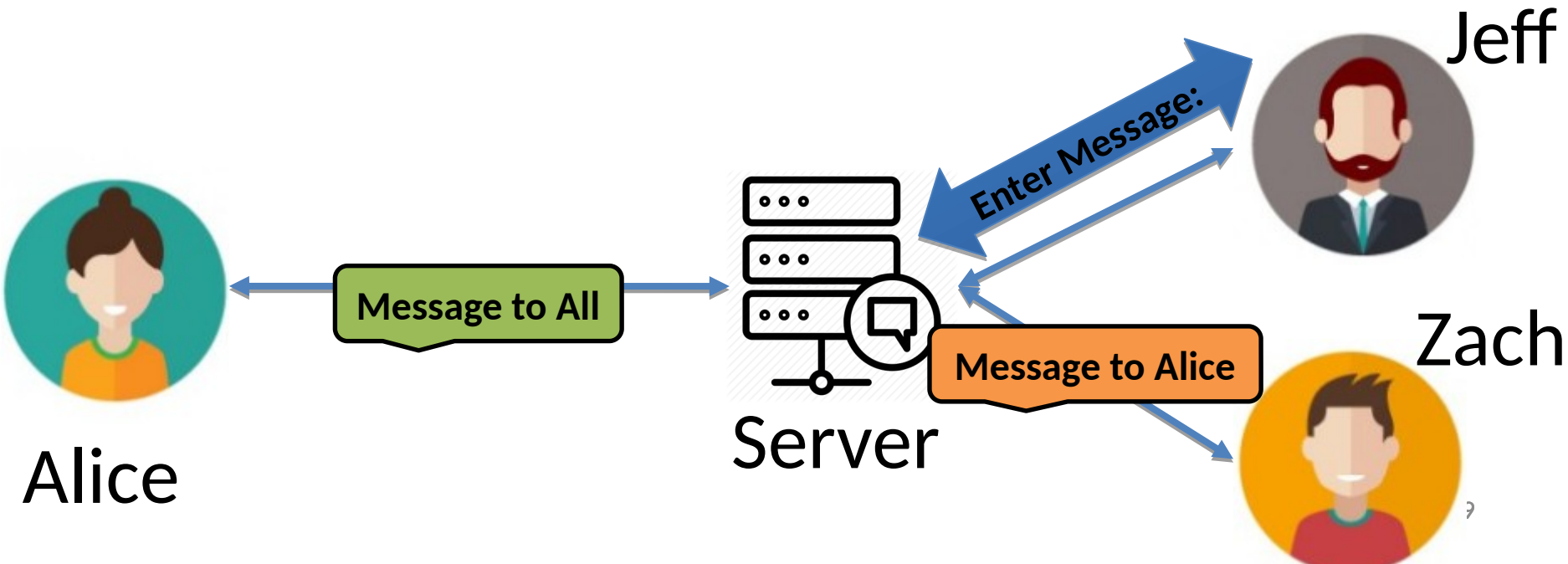
8

PG3 – Multithreading

Multithreading is the ability of a CPU in a multi-core processor to execute multiple processes or threads concurrently

Why Multithreading? Concurrency!

- **Jeff is interacting with server (e.g., server is prompting Jeff for command options)**
- **Alice is sending broadcast message to Jeff and Zach**
- **Zach is sending private message to Alice**
- **Both clients and server need to handle multiple messages at the same time!**



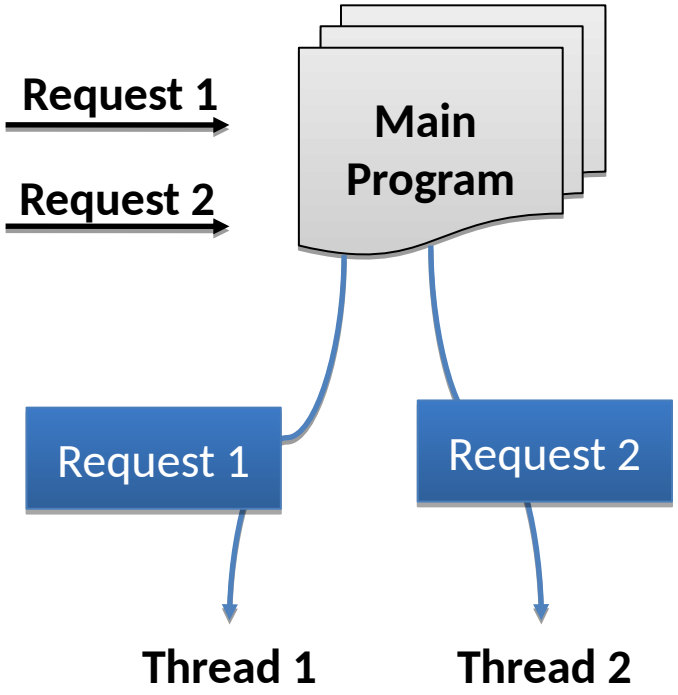
PG3 – Multithreading

Use *pthread* library http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html

pthread_create() // create new thread

pthread_join() //suspend current thread until a thread finishes

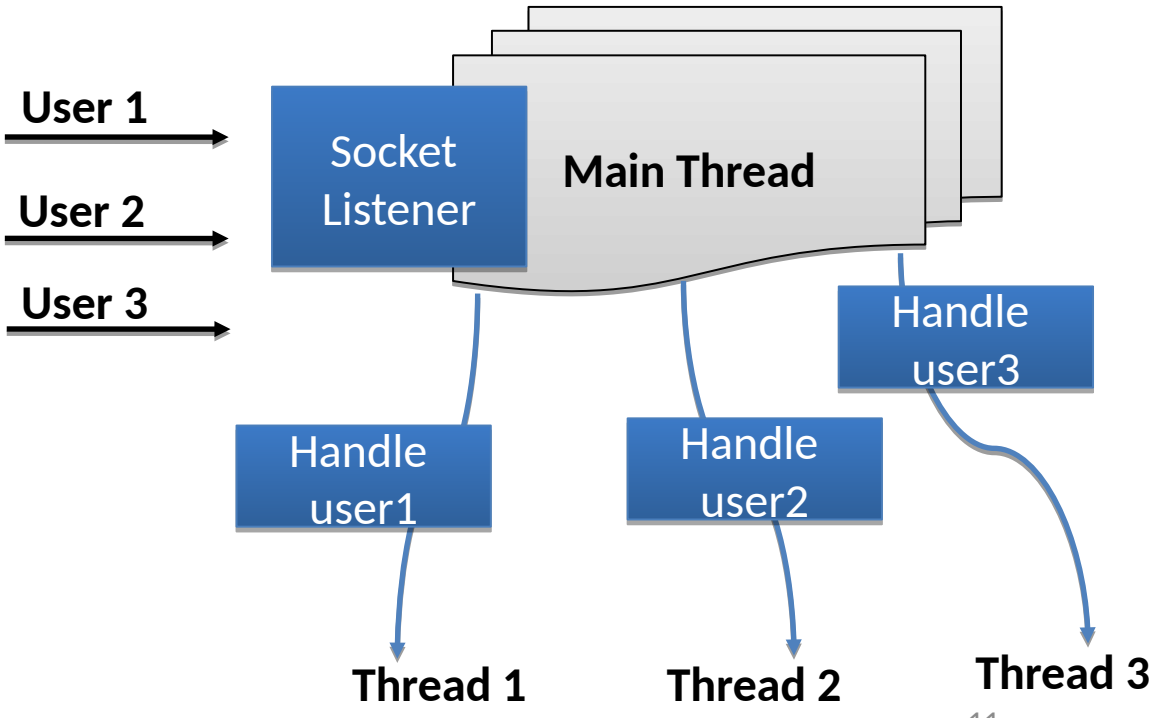
pthread_exit() // exit a thread



PG3 – Multithreaded Server

- Server – needs to interact with multiple users concurrently.

- Recommended solution:
  - Main thread (socket listener): keep listening for new connections
  - Create a new thread (client handler) for each online user.



PG3 – Multithreaded Server

- **Server – needs to interact with multiple users concurrently.**

- **Recommended solution:**
  - **Main thread (socket listener): keep listening for new connections**
  - **Create a new thread (client handler) for each online user.**

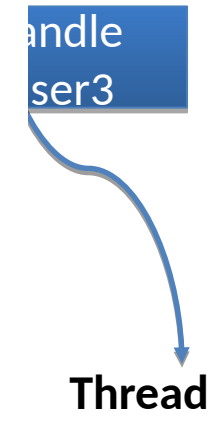
New Connection?

```
while((client_sock = accept(s.get_s(), (struct sockaddr *)&s.s_in, &s_inlen)) > 0){
    if (NUM_THREADS == 10) {
        std::cout << "Connection Refused: Max Clients Online.\n";
        continue;
    }

    std::cout << "Connection accepted.\n";
    pthread_t thread;
    struct thread_args args;
    NUM_THREADS++;

    if (pthread_create(&thread, NULL, clientinteraction, (void*)&args) < 0) {
        perror("Error creating thread");
        return 1;
    }
}
```

New thread for each connected client

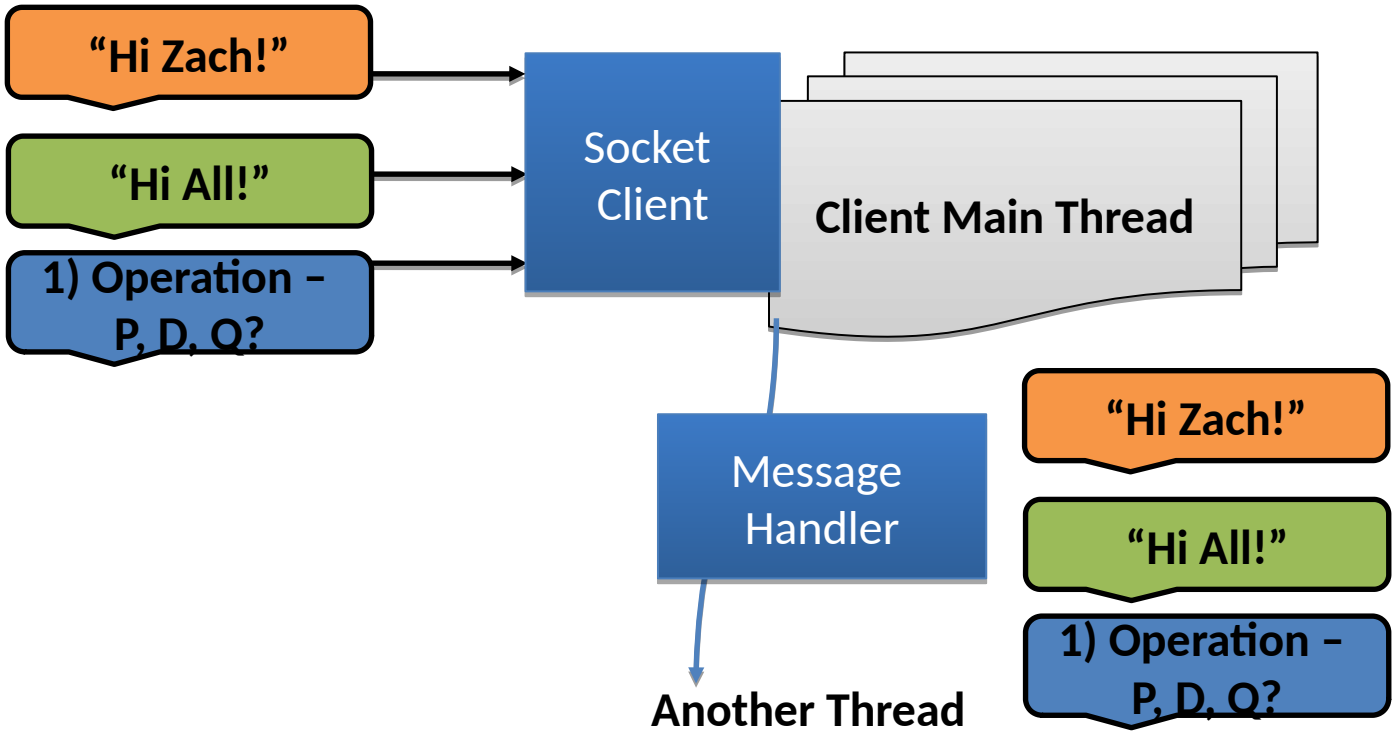


Thread 1 Thread 2 Thread 3

PG3 – Multithreaded Client

- Client – needs to handle both messages from server and messages from other clients.

- Recommended solution:
  - Main thread (socket client): normal client operations (prompt user for input, interact with server)
  - Another thread (message handler): handle each message



PG3 – Multithreaded Client

- **Client – needs to handle both messages from server and messages from other clients.**

- **Recommended solution:**

```
int main(int argc, char * argv[]) {

    while (!EXIT) {
        // Log In //
        login(username);

        pthread_t thread;
        int rc = pthread_create(&thread, NULL, handle_messages, NULL);

        // Interact //
        while (1) {
            if (rc) {
                std::cout << "Error: unable to create thread\n";
                exit(-1);
            }

            std::cin >> op;

            if (op == "P") {
                private_message();
            }
            else if (op == "B") {
                broadcast();
            }
            else if (op == "E") {
                exit(-1);
            }
            else {
                std::cout << "Invalid Entry\n" << OPTIONS;

            }
        }

        s.close_socket();
    }
    return 0;
}
```

Create an extra thread at the start

```
void *handle_messages(void *) {
    while (ACTIVE) {
        std::string message;
        s.recv_string(message);

        if (message is Data Message) {
            //handle data message
        }
        else {
            //handle command message
        }
    }
    return 0;
}
```

Handler simultaneously distinguishes and handles messages

Normal client operations
(Same as PG1&2)

PG3 – Message Frames

- Client must understand different types of messages

- Two types:
  - Data Message
  - Command Message

Recommended solution:

Use an extra byte at the beginning of each message (e.g. "C" for command message, "D" for data message).

PG3 – More Info

- **Protocol**: For PG3, you can use either **UDP or TCP** at your choice.
- **Programming language**: **C or C++**
- **Communication Specifics**: Much more open-ended than PG1-2. You decide the structure,
- **Testing**: use **4** student machines, **one as server**, **three as clients**

PG3 – Demo

- Server: **./chatserver Port**
- Client: **./chatclient Server_Name Port Username**

- Video of the demo can be found at:

https://www3.nd.edu/~dwang5/courses/fall18/programming/prog3.html

Thank You!

Questions?