

# Recitation 5: Programming Assignment 1 (Part 2)

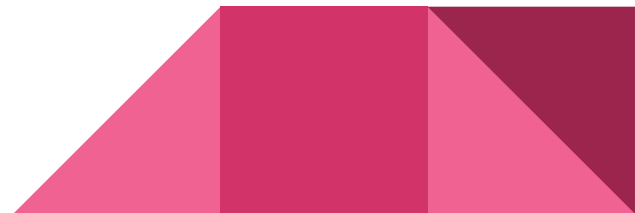
# Outline

Part 1: PA1 Overview

Part 2: Testing

Part 3: Debugging


Part 4: Submission



# Important Dates

- Project 1
  - Due Date: Friday, October 30 at 11:59 PM

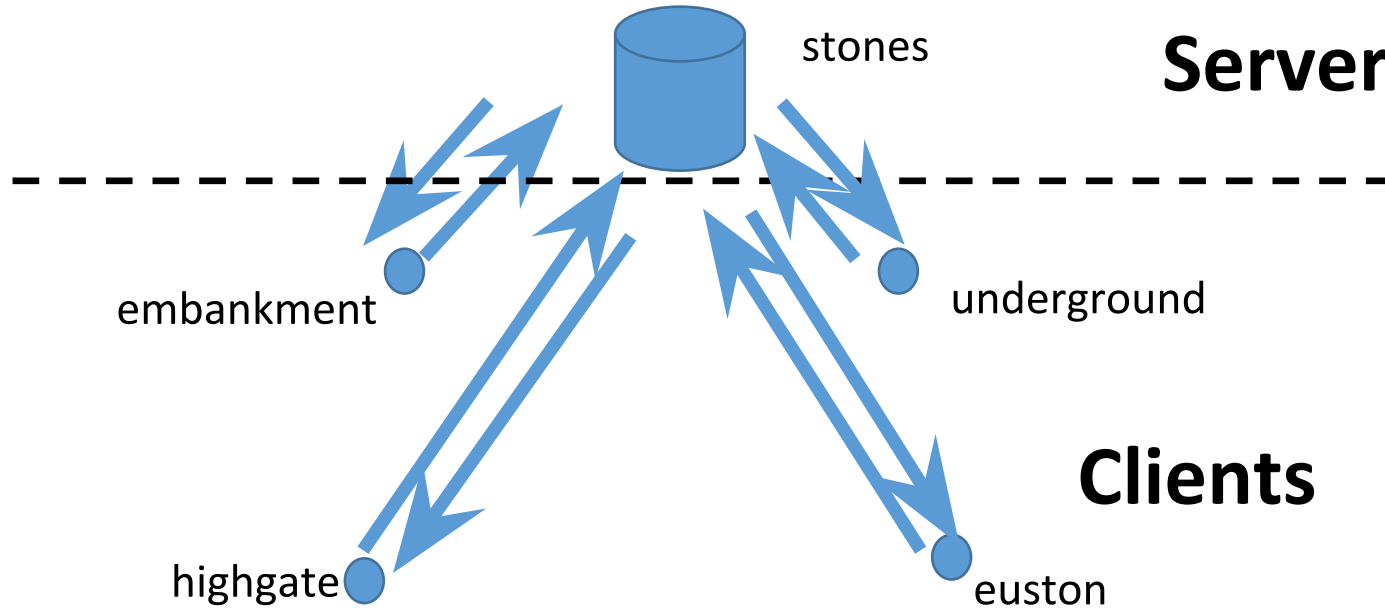




# Part 1: PA1 Overview

# Project Objective

- Develop a text chat application for message exchange among remote hosts:
  - One Server
  - Multiple (at most 4) Clients
  - Communication using TCP sockets



# Running the Project

- Your project will consist of one executable file that takes two additional command line arguments.
  - The first command line argument will indicate whether the program should act like a server or act like a client.
  - The second command line argument will indicate what port number the client or server should listen for connections on.
    - You do not need to set up a listening socket on the client if you are not implementing the bonus.
- Once the program is launched, the user can start typing commands with the keyboard.
- Example Run:
  - Start Server:
    - `./assignment1 s 4322`
  - Start Client:
    - `./assignment1 c 4322`
- Example Command Usage on Terminal Screen:
  - IP
  - [Output of IP Command]
  - LOGIN 128.56.78.31 4322
  - [Output of LOGIN Command]

# Running the Project (Assumptions)

- When a server instance is started, no clients are connected to it.
- When a client instance is started, it has not logged in to any server yet.
- When a client **EXITS** the program, it won't ever log back in.



# Commands and Events

- For each command or event there is mandatory output
  - The expected output format for each command is shown in the project document
- It's imperative that you follow the format exactly
  - The grading-script will treat any typos, extra characters or whitespaces as errors in your program
- All mandatory output should be done using the `cse4589_print_and_log(char* format, ...)` function
  - This will write to stdout (the terminal) and to a file within the “log” directory
  - It's perfectly fine to include normal print statements in your code because the grader doesn't monitor stdout
- You can recognize and parse command input in your program by using basic string functions like `strcmp()` for comparisons and `strtok()` for splitting the string on space characters.



# Commands

## Client Commands

- Can be executed at anytime:
  - AUTHOR
  - IP
  - PORT
  - EXIT
- Can be executed when not logged in:
  - LOGIN
- Can be executed only when logged in:
  - LIST
  - REFRESH
  - SEND
  - BROADCAST
  - BLOCK
  - UNBLOCK
  - LOGOUT

## Server Commands

- Can be executed at anytime:
  - AUTHOR
  - IP
  - PORT
  - LIST
  - STATISTICS
  - BLOCKED

# Events

## Client Events

- When receiving a message from the server:
  - RECEIVED
- Ex :

```
[RECEIVED:SUCCESS]
msg from:128.205.36.33
[msg]:This is msg1
[RECEIVED:END]
```

## Server Events

- When relaying a message to one or multiple clients:
  - RELAYED
- Ex :

```
[RELAYED:SUCCESS]
msg from:128.205.36.33, to:128.205.36.34
[msg]:This is msg1
[RELAYED:END]
```

# Command Explanation

- AUTHOR

- Prints a predefined statement to the screen which includes your UBIT name.
- **Warning:** This command must be implemented successfully in order for your project to receive a grade.

- PORT

- Prints the listening port number -i.e. the port number that the user passed in from the command line.

- IP

- Displays the external IP address of the machine. (127.0.0.1 is **not** the right answer).
- Steps to Find Correct IP Address:
  - Create a UDP socket.
  - Call connect ( ) on the UDP socket. Connect the socket to any IP address and port number (for example: 8.8.8.8 on port 53).
  - Call getsockname ( ) on the UDP socket in order to get the machine's IP address.
  - More Info:

<https://ubmnc.wordpress.com/2010/09/22/on-getting-the-ip-name-of-a-machine-for-chatty/>

# Command Explanation

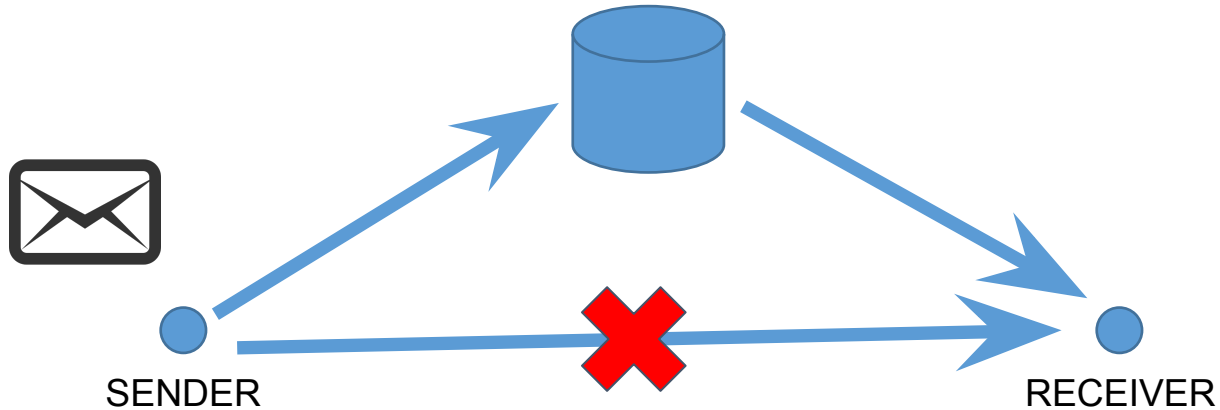
- LOGIN [Server IP Address] [Server Port]
  - Client logs into the specified server.
    - Server will add client to its list of connected clients.
    - Server sends the client a list of all other clients that are currently logged in to the server.
    - Server also sends any messages that it has buffered for the client.
- LIST
  - Display a list of all currently logged in clients.
    - List is sorted by increasing listening port number.
    - The server does **not** appear in the list.
    - **Warning:** Most automated tests will fail if LIST is not implemented correctly.

|   |                            |               |      |
|---|----------------------------|---------------|------|
| 1 | stones.cse.buffalo.edu     | 128.205.36.32 | 4545 |
| 2 | embankment.cse.buffalo.edu | 128.205.36.35 | 4800 |
| 3 | highgate.cse.buffalo.edu   | 128.205.36.33 | 5000 |
| 4 | euston.cse.buffalo.edu     | 128.205.36.34 | 5100 |

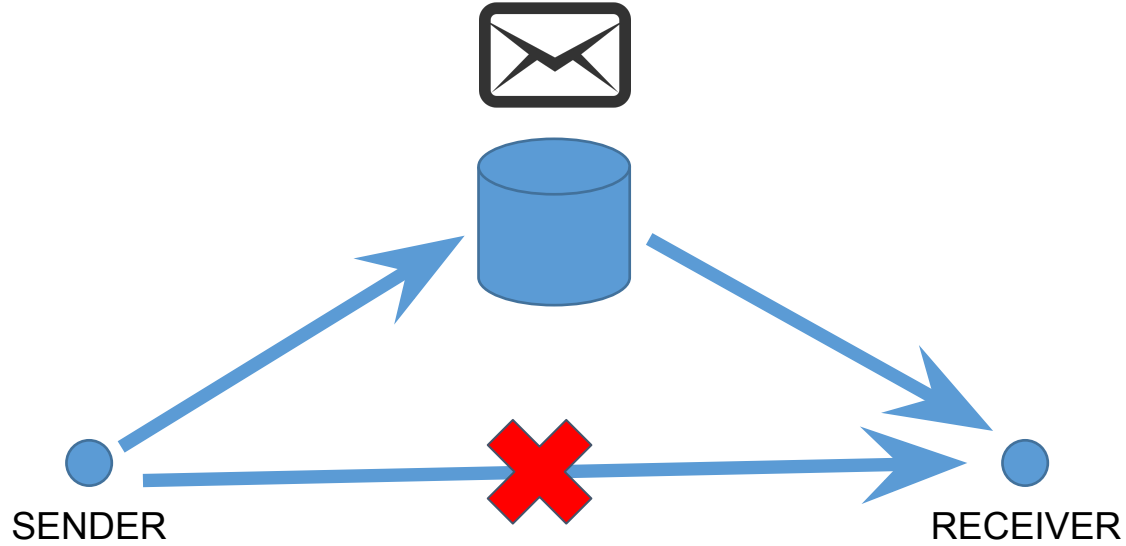
# Clients

- When the client portion of the program is launched, the user will only be able to execute a few commands
  - AUTHOR, IP, PORT, EXIT, LOGIN
- A user must log in if they want to successfully execute the other commands
  - REFRESH
  - LOGOUT
  - LIST ...
- Clients can send two types of messages to other clients
  - Unicast → 1 message (SEND command)
  - Broadcast → N - 1 messages (messages are duplicated by the server) (BROADCAST command)
  - The max length of a message is 256 bytes and a message can only contain valid ASCII characters.
- Clients communicate with each other **ONLY** through the server

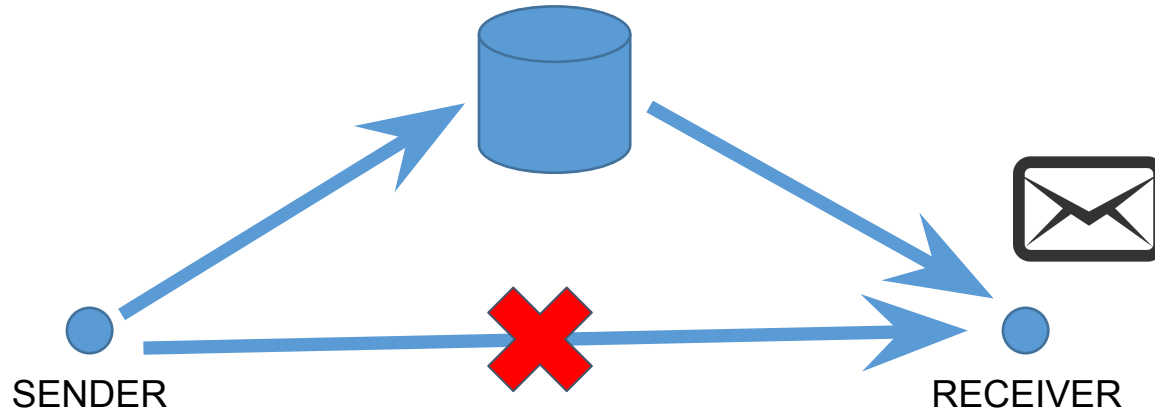
# Unicast (SEND Command)



# Unicast (SEND Command)

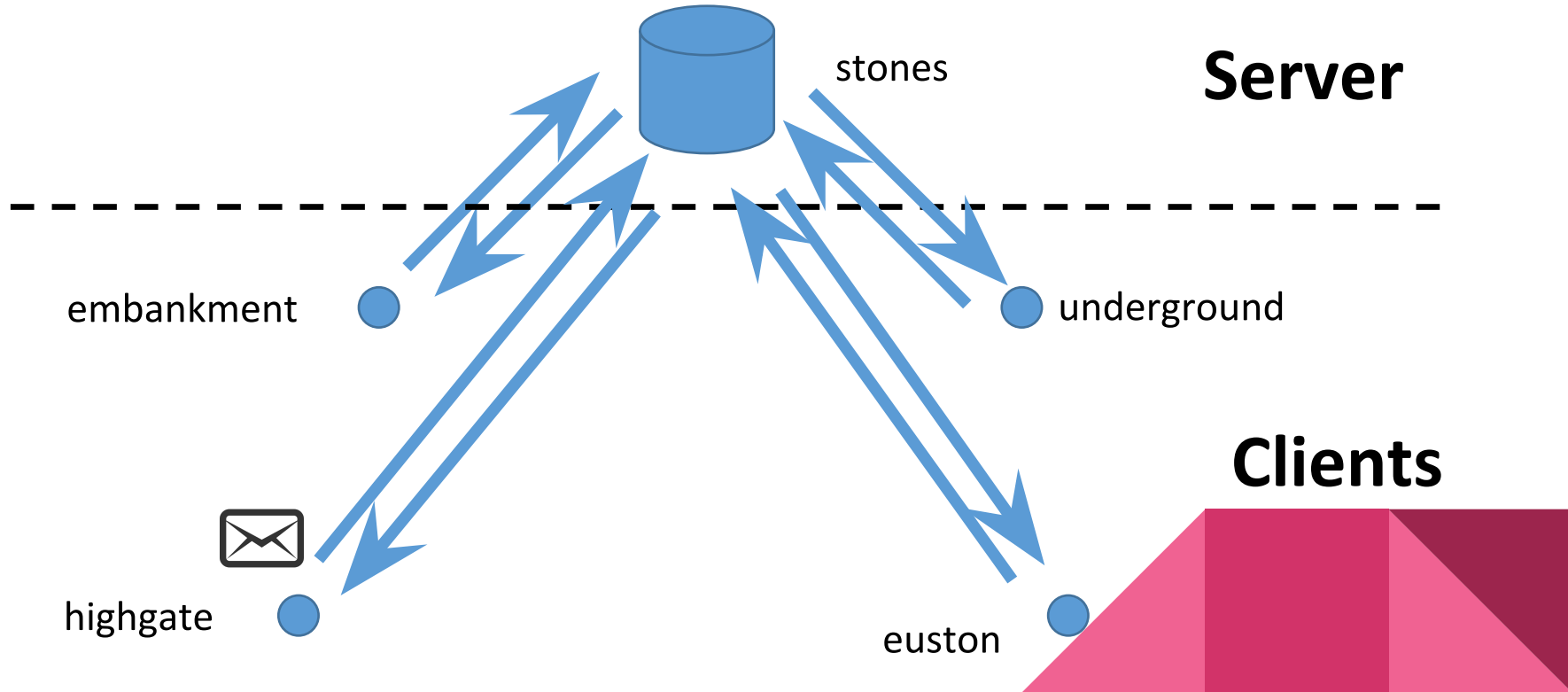


# Unicast (SEND Command)

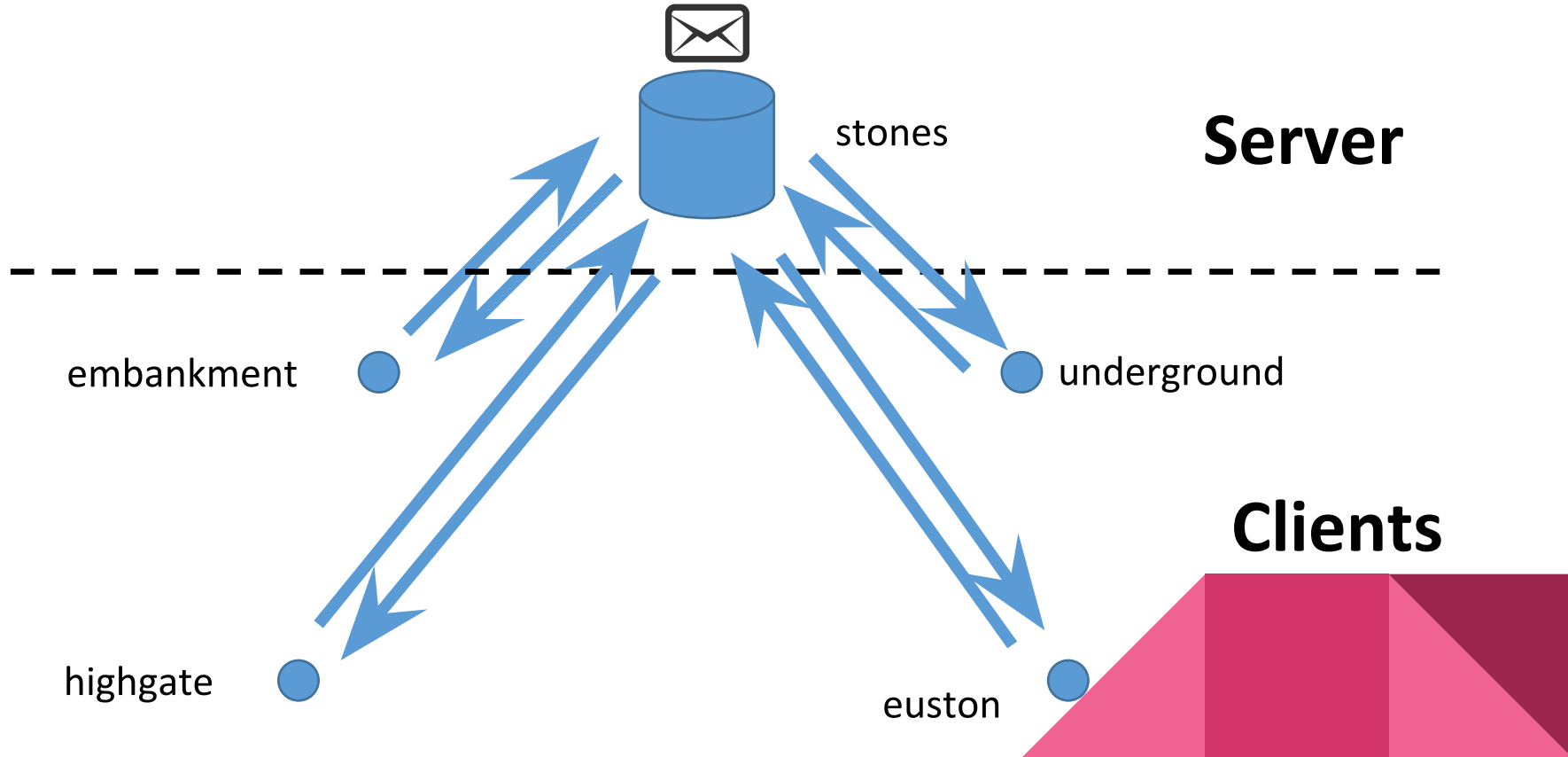




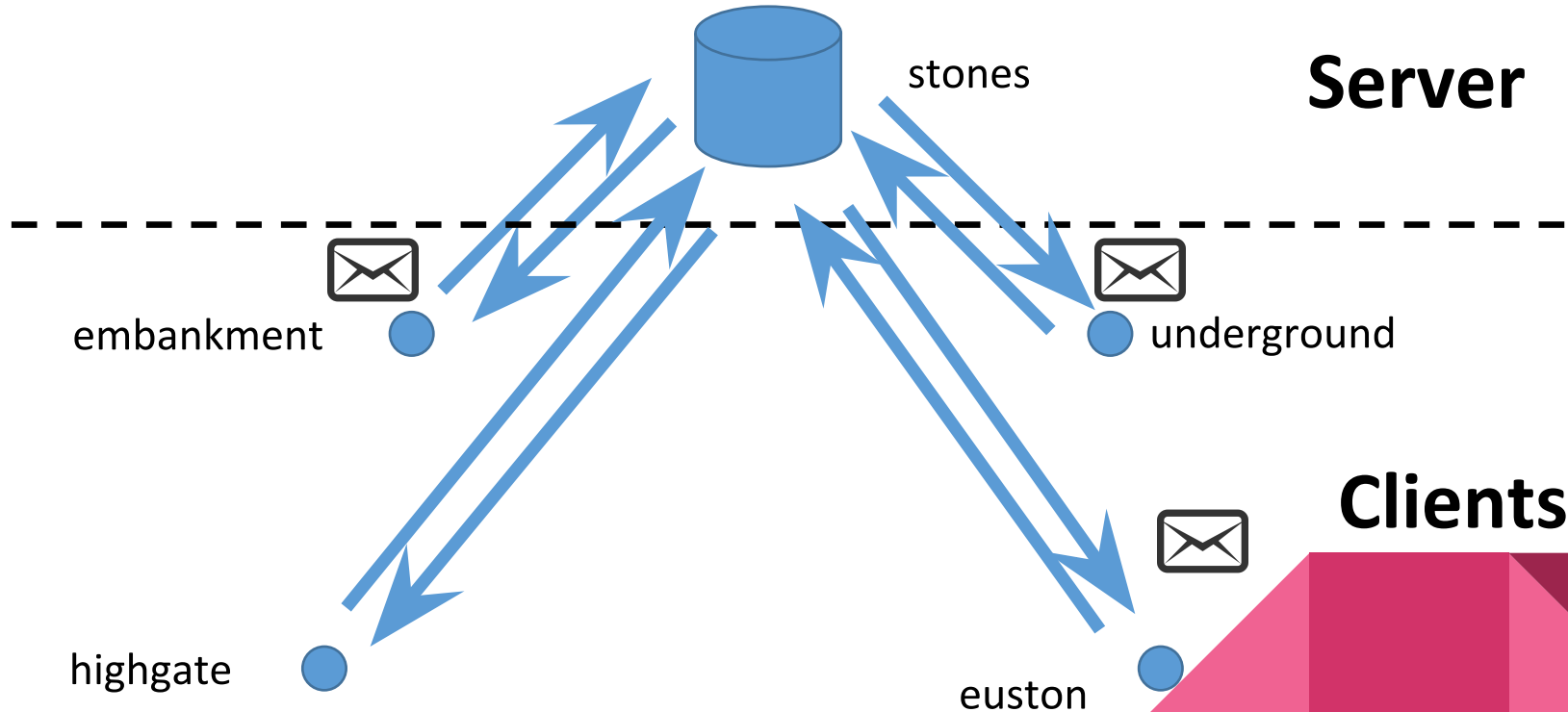
# Broadcast (BROADCAST Command)



# Broadcast (BROADCAST Command)



# Broadcast (BROADCAST Command)



# Server

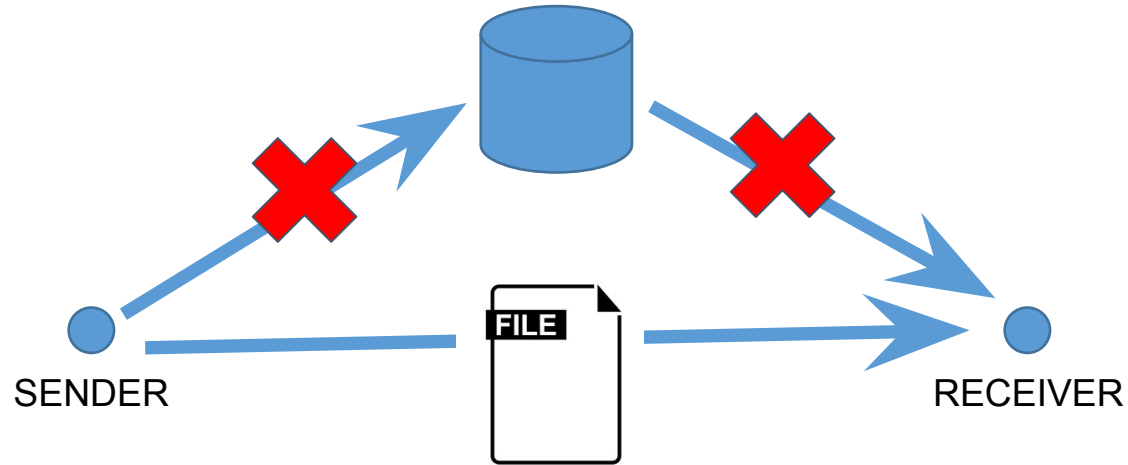
- The server has three main functions
  - Forward incoming messages to the proper recipient(s)
    - If a recipient blocked another client from receiving their messages, then the server should not deliver the message to that recipient
  - Keep track of various statistics
    - Number of messages each client has sent
    - A list of logged-in clients
  - Buffer messages for clients not logged in
    - Client should receive buffered messages in the order that they were sent - i.e. like a queue → First in First out
    - Messages do not have to be buffered for clients that have EXITed.
    - If client A has blocked client B, then messages from client B to client A do not have to be buffered.
    - Total number of buffered messages will not exceed 100.

# Bonus: P2P File Transfer

- Implement additional functionality to allow clients to send/receive files
  - The transfer will take place directly between two clients and will not involve the server
  - You only need to modify your client code!
- Here's what you need to do:
  - Implement select in your client code (Use server.c as a reference)
  - Read <file> into a buffer when client types in "SENDFILE <client-ip> <file>"
  - Send buffer to <client-ip>
- An extra 20 points can be added to your overall PA1 grade
  - 2 separate file transfers; 10 points for a file with ascii data and 10 points for file with binary data
  - You can assume the files won't exceed 10 MB
- Remember that string functions don't work on binary data!



# Send File (SENDFILE Command)



# Additional Project Requirements

- Make sure your project successfully compiles and runs on CSE Servers (stones, underground, embankment, euston, highgate)
  - **Do not compile your code manually. Use the Makefile provided by the template.**
- Only one program is running on each server, but takes different arguments:
  - `./chat_app s 4321`
  - `./chat_app c 4322`
- C or C++
  - No external libraries for socket programming
  - No external binaries/utilities (e.g., `ifconfig`, `nslookup`, etc.)
- TCP Sockets & No Multithreading (Must Use `select()`)
- Use the PA1 template **[MANDATORY]**
  - Template Instructions: <https://goo.gl/L2kqb5>
- This presentation only covers a subset of the commands that are required for the project.
  - View the project description document for more details: <https://goo.gl/bqf2E1>



# Part 2: Testing



# Dedicated Servers

- Ensure that your code compiles and operates correctly on the following 5 servers
  - stones.cse.buffalo.edu
  - euston.cse.buffalo.edu
  - embankment.cse.buffalo.edu
  - highgate.cse.buffalo.edu
  - underground.cse.buffalo.edu
- For development and testing you should only use the directory we created for you on each of the 5 servers at **/local/Fall\_2020/<Your-UBIT-Name>/**
  - You should use these directories for development and testing (not your home directories), and never change the permissions (currently only you can access your folder).



# Testing

- To avoid unnecessary problems, test your code **ONLY** on the 5 dedicated UB servers
- Use the autograder to ensure correctness of your code
- The autograder is included in the PA1 template.



# Testing

- Before using the autograder, you must first package your code into a tarball.
- Use the supplied script in the template to create a package (.tar) from your code.
  - `./assignment1_package.sh`
- Do NOT package manually.
- Make sure to run the package script whenever you modify your source code.



# Grader Configuration File (grader.cfg)

- Before you can use the autograder, you need to set the port number equal to 2730 in your grader.cfg file.
- The first two lines of your file should look like this:
  - [HTTPLauncher]
  - port: 2730



# The Autograder

- The PA1 template comes with an autograder
- To run it, you need to supply the path to grader.cfg, your submission (\*.tar), and which test to run
  - Ex (assuming you're in the grader directory):
  - `./grader_controller -c grader.cfg -s ../<ubit>_pa1.tar -t author`
  - There are also the optional -nu and -nb parameters
    - nu = no upload
    - nb = no build
    - Only supply these parameters when you have NOT updated your submission AND just want to test a different command.
- Can only test one command at a time
- What you see is what you get
  - You have access to all of the tests we'll use to grade your code

# The Autograder Tests


- The following tests can be executed by the autograder (with the -t parameter):

- startup
- author
- ip
- port
- \_list
- refresh
- send
- broadcast
- block
- blocked
- unblock
- logout
- buffer
- exit
- statistics
- exception\_login
- exception\_send
- exception\_block
- exception\_unblock
- exception\_blocked
- bonus

- More info about grader: <https://goo.gl/L2kqb5>

- Test Descriptions:

[https://docs.google.com/document/u/1/d/e/2PACX-1vTlvLiOlmg-wUrHmfBvxL-v6D0RAUMDVDkGzhPfU\\_nhsLZzTgviQAvuuQW7ZPKvyrfhoqYfpLHi\\_MXF/pub](https://docs.google.com/document/u/1/d/e/2PACX-1vTlvLiOlmg-wUrHmfBvxL-v6D0RAUMDVDkGzhPfU_nhsLZzTgviQAvuuQW7ZPKvyrfhoqYfpLHi_MXF/pub)



# Part 3: Debugging

# Debugging

- There are two main ways to debug your code
  - Printf/cout statements and GDB
- “What if I get the correct output when I run my code, but I’m still losing points?”
  - For this scenario, it’s helpful to observe what the grader sees.
  - To accomplish this, you can write to the /tmp directory so you can see what your code is outputting.
  - If you do this, be aware that your file is “world” readable
    - Everyone can see the contents of your debug file
  - Here’s what the code would look like:





# Writing to /tmp

```
#include <stdio.h>
...
FILE * pFile;
pFile = fopen ("/tmp/myfile.txt", "a");
...
if(strcmp("AUTHOR", input_str) == 0) {
    fprintf (pFile, "[%s:SUCCESS]\n", input_str);
    fprintf (pFile, "I, %s, have read and understood the course academic
integrity policy.\n", UBIT);
    fprintf (pFile, "[%s:END]\n", input_str);
    fflush (pFile);

    cse4589_print_and_log("[%s:SUCCESS]\n", input_str);
    cse4589_print_and_log("I, %s, have read and understood the course
academic integrity policy.\n", UBIT);
    cse4589_print_and_log("[%s:END]\n", input_str);
}
```

# Writing to /tmp (Lazy Person's Method)

```
#include <stdio.h>
#include <stdlib.h>

...
// redirect all standard output to /tmp/filename.txt
if (freopen ("/tmp/myfile.txt", "a", stdout) == NULL) {
    exit(1);
}

...
if(strcmp("AUTHOR", input_str) == 0) {
    cse4589_print_and_log("[%s:SUCCESS]\n", input_str);
    cse4589_print_and_log("I, %s, have read and understood the course
academic integrity policy.\n", UBIT);
    cse4589_print_and_log("[%s:END]\n", input_str);
    fflush (stdout);
}
```

# Accessing Files Written to /tmp

- You can view the files written to /tmp by using a command like cat
  - In the previous two code snippets, we wrote data to the following file: /tmp/myfile.txt
  - In order to see the contents of this file, we first have to SSH into one of the five dedicated CSE servers. Once there we can view the contents of the file by executing the following command:  
`cat /tmp/myfile.txt`
- Files written to /tmp are local to the machine they are written on.
  - In the previous two code snippets, running `cat /tmp/myfile.txt` on stones will show the output of the program when it was run on stones. Likewise, running `cat /tmp/myfile.txt` on underground will show the output of the program when it was run on underground. The same is true for the remaining three servers.




# Gentle Guide to GDB

- GDB allows you to see what is going on `inside' another program while it executes
  - More effective than printf statements
- To start GDB for PA1, type “gdb --args ./<path-to-PA1 <s/c> <port-num>”
  - --args lets you pass in parameters for your program into GDB
- To set a breakpoint, type “b <where>”
  - <where> could be a line number in a file (server.c:51), or it could be a function name (main)
  - This lets you set a breakpoint and will suspend execution in a specific place in your program
- To step to the next instruction, type “n”
- To print the value of a variable, type “p <var-name>”
- To restart/run your program, type “r”
- That's all you need to track down bugs!
- Refer to here for even more commands:

<https://darkdust.net/files/GDB%20Cheat%20Sheet.pdf>





# Part 4: Submission

# Submitting

- Before submitting, you must first package your code into a tarball (.tar file).
- To do this, use the script included with the template:
  - `./assignment1_package.sh`
- Do NOT package manually
- This ONLY packages; **Does NOT SUBMIT**



# Submitting

- After packaging your code, you must submit it using the CSE 489/589 submission scripts.
- Submitting your assignment follows the same procedure as labs and homeworks.
  - Undergraduates: `submit_cse489 <ubit_pa1.tar>`
  - Graduates: `submit_cse589 <ubit_pa1.tar>`
- If you're working with another student, only one group member needs to submit
  - But make sure **both** UBIT names are located in the `ubitname_assignment1.c/cpp` file.

```
timberlake {~/Downloads} > submit_cse489 swetankk_pa1.tar
Submission of "swetankk_pa1.tar" successful.
timberlake {~/Downloads} > date
Mon Feb  1 17:38:25 EST 2016
```

# Resources

- Socket Programming
  - Beej's Network Programming Tutorial:
    - PDF Version: [https://beej.us/guide/bgnet/pdf/bgnet\\_usl\\_c\\_1.pdf](https://beej.us/guide/bgnet/pdf/bgnet_usl_c_1.pdf)
    - HTML Version: <https://beej.us/guide/bgnet/html/>
- General C/C++ Reference: <https://en.cppreference.com/w/>
- Programming Assignment 1
  - Description: <https://goo.gl/bqf2E1>
  - Template & Packaging and Submission: <https://goo.gl/L2kgb5>
  - Grading Rubric: <https://goo.gl/UAVWgY>
  - Description of Grading Tests:  
[https://docs.google.com/document/u/1/d/e/2PACX-1vTlvLi0ImG-wUrHmfBvxL-v6D0RAUMDVdKgzhpFU\\_nhsLZzTgviQAvuuQW7ZPKvyrfhoqYfpLHi\\_MXF/pub](https://docs.google.com/document/u/1/d/e/2PACX-1vTlvLi0ImG-wUrHmfBvxL-v6D0RAUMDVdKgzhpFU_nhsLZzTgviQAvuuQW7ZPKvyrfhoqYfpLHi_MXF/pub)



# Next Week

- Wireshark Lab 3



Any Questions?



# Acknowledgements

- Some of the slides were adapted from material made by the previous authors of the CSE 489 recitation slides.

