15-213 Recitation 12

Introduction to Computer Systems

Fahim Dalvi

21 November, 2013

Malloc Lab

- Due Yesterday
- Hopefully you are past the 60% mark
- Any questions?

Proxy Lab

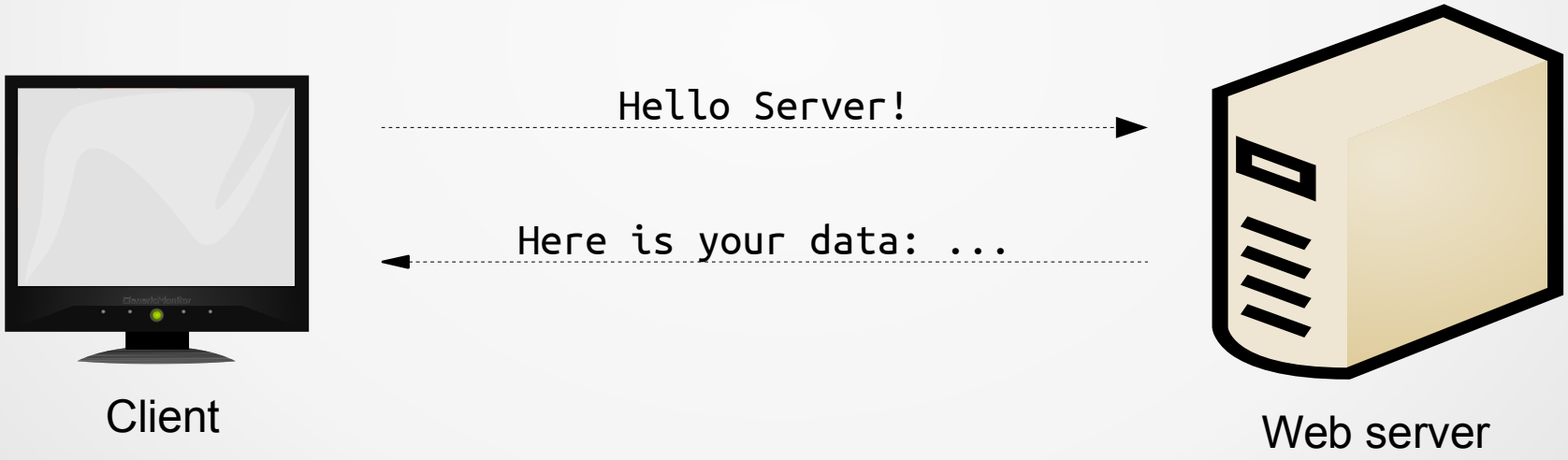
- Due 4th December, 2013
- No late days for the assignment!
- Kind of not autograded
- We will have interviews with each of you
- NOT a group based assignment

Proxy lab

- Three main steps
  - Step 1: Implement a sequential proxy
  - Step 2: Make the proxy concurrent
  - Step 3: Implement a web-cache

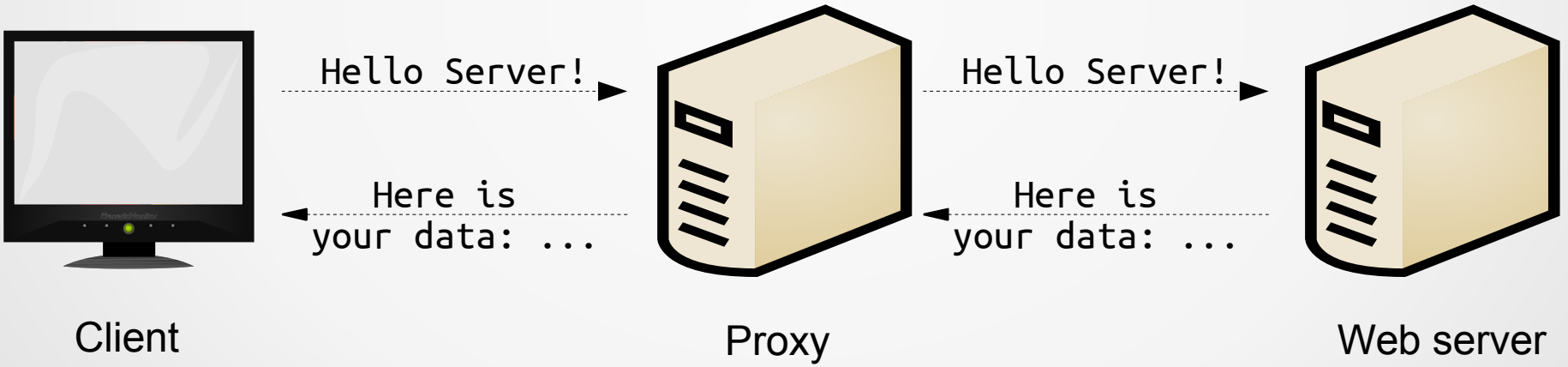
Implementing a sequential proxy

- What is a proxy?



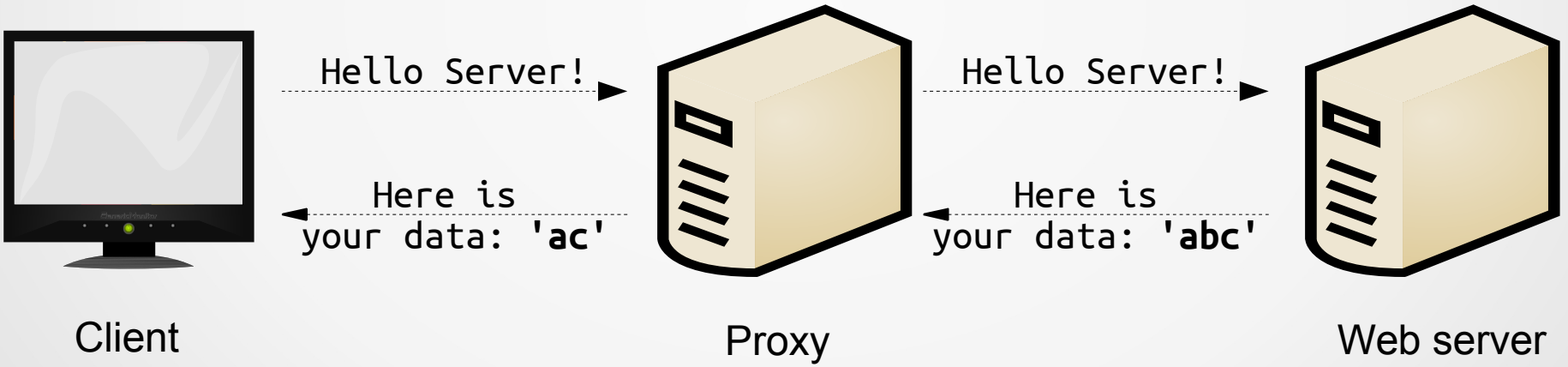
Implementing a sequential proxy

- What is a proxy?



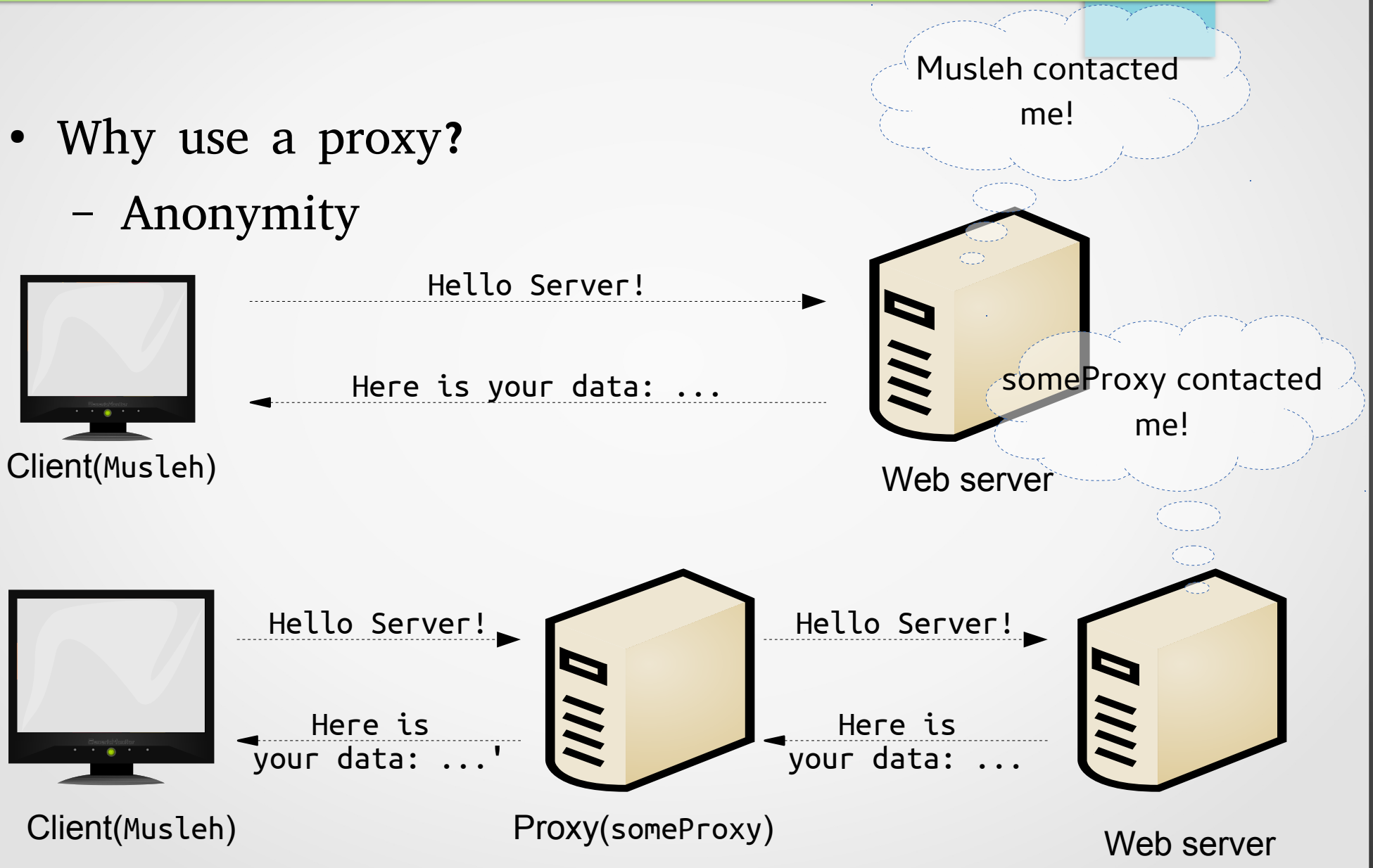
Implementing a sequential proxy

- Why use a proxy?
  - Content filtering



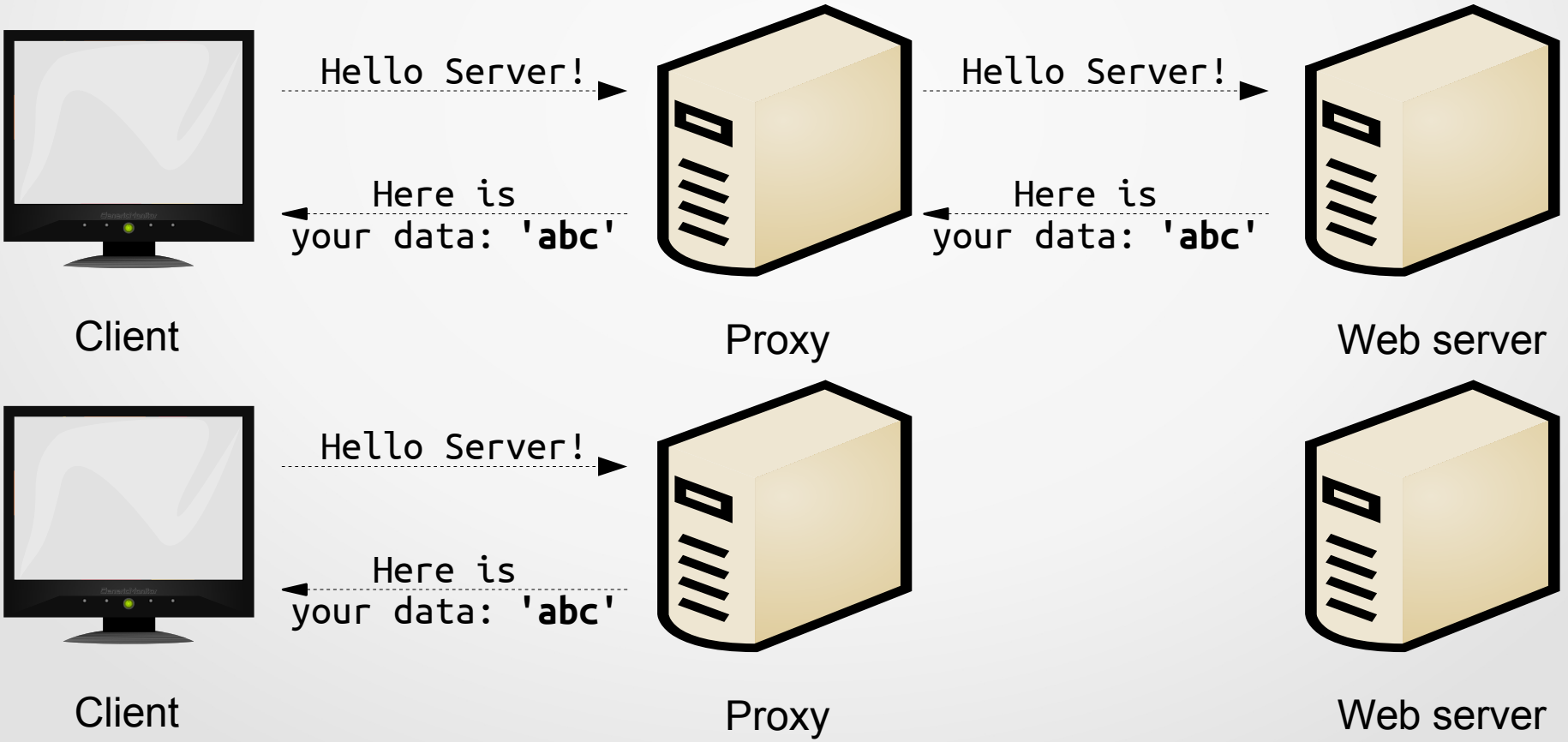
Implementing a sequential proxy

- Why use a proxy?
  - Anonymity



Implementing a sequential proxy

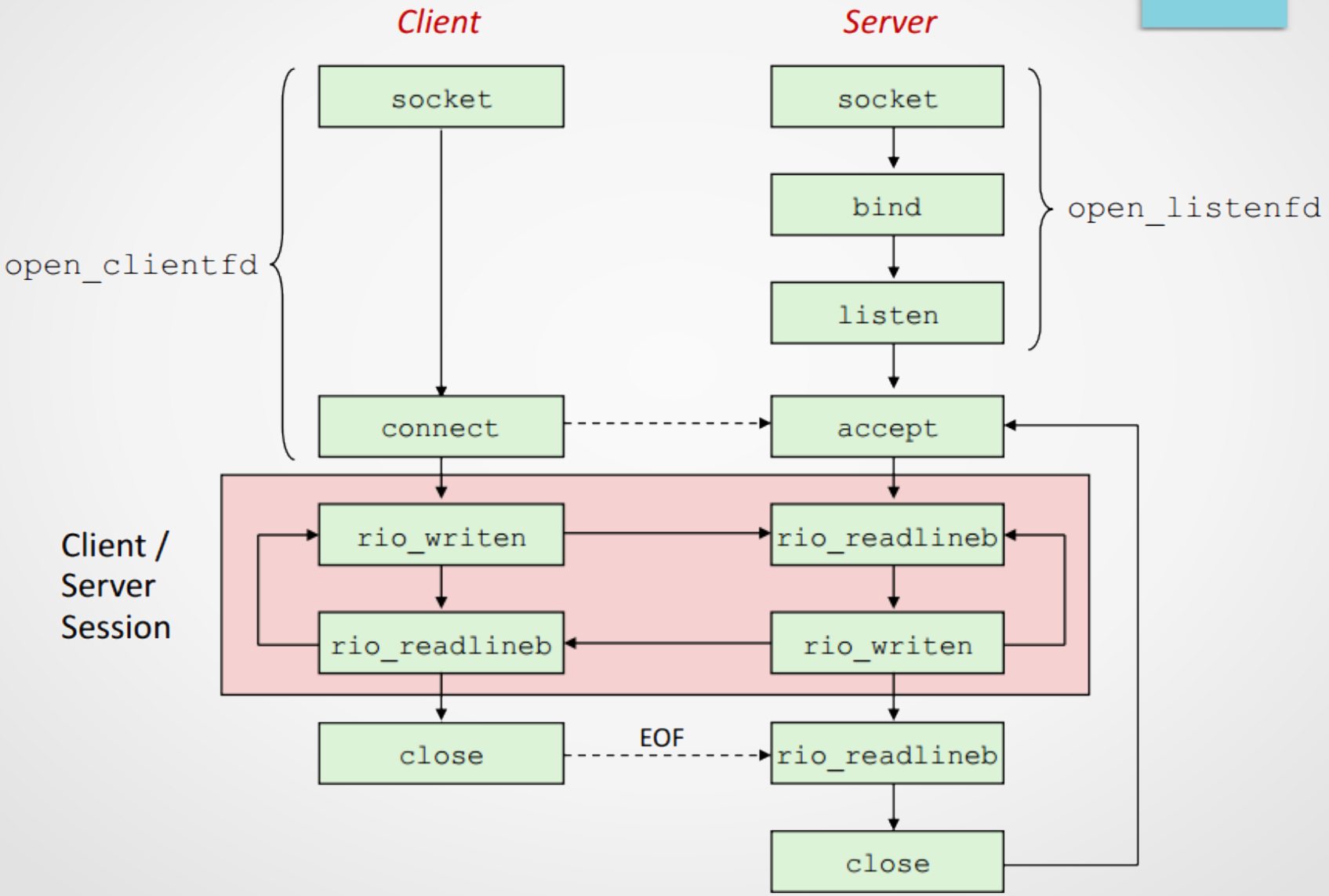
- Why use a proxy?
  - Caching



Implementing a sequential proxy

- So far, you've seen basic client-server communication
- A proxy is a special entity
  - It is a server to the clients
  - And a client to the servers!
- You've seen (hopefully...) code for a simple server and a client, use this to your advantage

Implementing a sequential proxy



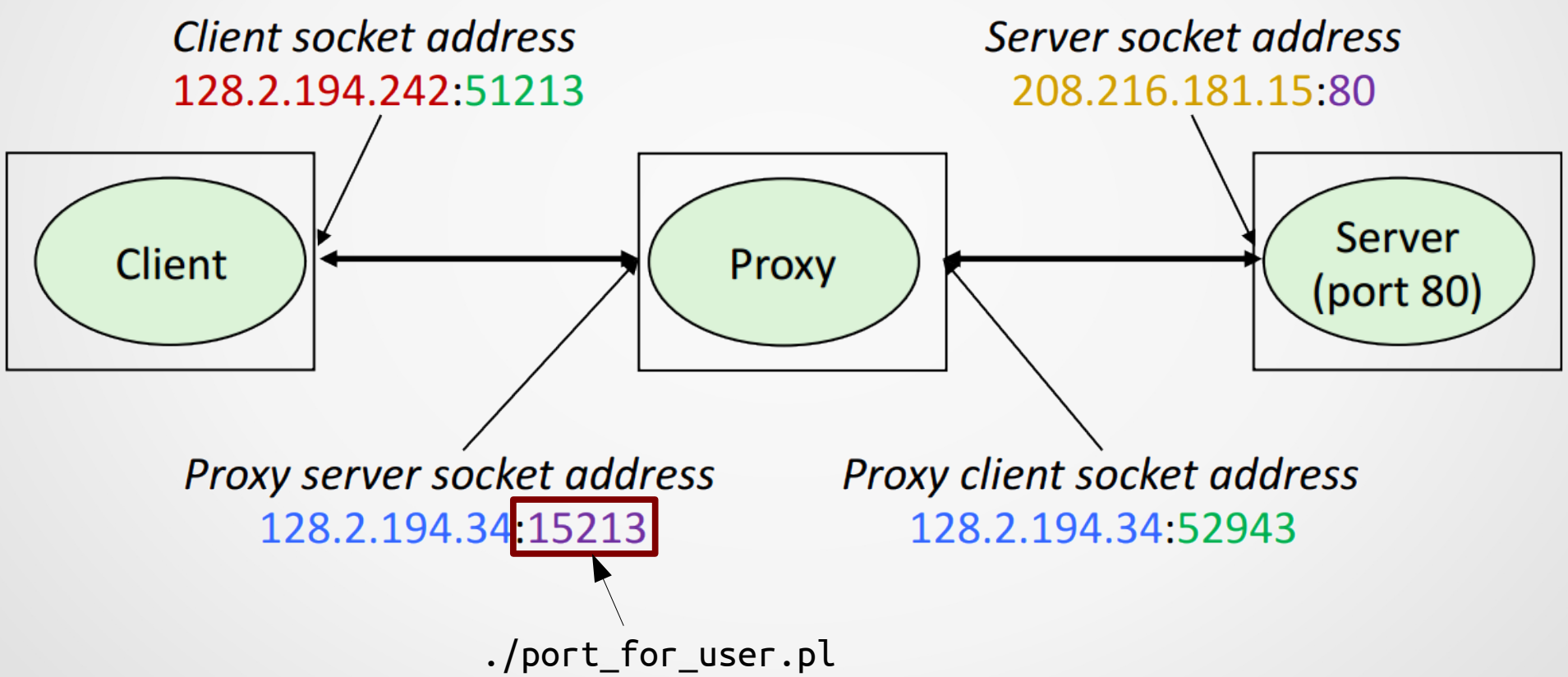
Implementing a sequential proxy

- Lets look at the technical details
  - You will be implementing the HTTP/1.0 GET request protocol
    - Fairly simple, might get a little tedious
  - Hence, not all websites will work
    - Websites that use POST requests
    - Websites with HTTPS
  - Examples of websites that do work:
    - `cs.cmu.edu`
    - `qatar.cmu.edu/~kharras/`

Implementing a sequential proxy

- Lets look at the technical details
  - In any case, your server must be robust (After all, it is a server!)
  - It should not crash on malformed requests, or requests to non-existent websites

Implementing a sequential proxy



Implementing a sequential proxy

- Some really helpful tools
  - `telnet`
  - `curl`
  - `netcat`

Tools!

- `telnet`
  - Really rudimentary 'client'
  - You have to write the entire request yourself
    - Including the headers!
  - This is good, you can test your proxy incrementally
  - Try writing malformed requests to test the robustness of your proxy

Tools!

- `curl`
  - A more intelligent “client”
  - Builds valid HTTP requests automatically
  - `curl http://www.cmu.edu`
  - `curl -proxy lemonshark.ics.cs.cmu.edu:15213 http://www.cmu.edu`

Tools!

- `netcat`
  - “client” and a “server”!
  - Server:
    - `netcat -l -p 15213`
  - Client:
    - `netcat localhost 15213`

Tools!

- Host a `netcat` server
- Request to this server through your proxy
  - This will help you look at the EXACT request your proxy is performing to the web servers

Sockets API

- `int socket(int domain, int type, int protocol);`
- `int bind(int socket, const struct sockaddr *address, socklen_t address_len);`
- `int listen(int socket, int backlog);`
- `int accept(int socket, struct sockaddr *address, socklen_t *address_len);`
- `int connect(int socket, struct sockaddr *address, socklen_t address_len);`
- `int close(int fd);`
- `ssize_t read(int fd, void *buf, size_t nbyte);`
- `ssize_t write(int fd, void *buf, size_t nbyte);`

Step 2: Make it concurrent

- When you are done with your sequential proxy, and it is working PERFECTLY, you can move on to making it sequential
- You will learn threads in the next week, you will need them to complete this part and get a full grade
- In all the implementations you have seen so far, every client is processed fully before moving on to the next client

Step 2: Make it concurrent

- But this is bad!
  - A typical webpage has many, many requests
    - The webpage source
    - Stylesheets, scripts
    - Images etc..
  - Do you want to load each of these one at a time?
  - More importantly, if I am viewing the same website, do you really want to wait until the webpage is complete sent to me?
- Hence, your proxy will need to handle multiple requests at the same time

Step 2: Make it concurrent

We will look at this in detail next time, once you have done threading in class!

Step 3: Caching web objects!

- Your proxy should cache previously requested objects
  - With certain limits ofcourse! See the writeup for details
  - This has NOTHING to do with cachelab!
  - You will need to implement the LRU eviction policy

Step 3: Caching web objects!

- Wait, didn't we say we're going to have multiple threads?
  - What happens if two threads try to modify the cache at the same time?
  - Specifically, what happens if two threads try to modify the same object in the cache at the same time?
  - Even more specifically, what happens if one thread is reading an object, while another is modifying the object?

Step 3: Caching web objects!

- Welcome to the world of concurrency!
  - You will need to implement “mutual exclusion”
  - Only one thread can “write” at any time
  - Multiple threads can “read” at the same time
  - How will you implement this stuff?

Step 3: Caching web objects!

- Mutexes
  - Allow only one thread to run a section of code at a time
  - If other threads are trying to run the same section of the code, they will wait
- Semaphores
  - A more general version of a mutex
  - Allows a specified number of threads to access the section at any point

Step 3: Caching web objects!

- Luckily, most of this stuff is already implemented for you!
  - You will learn about locks in the next week
  - We will look at them in more detail after you see them

    Quick Reference:
    - `pthread_rwlock_t lock;`
    - `pthread_rwlock_init(&lock,NULL);`
    - `pthread_rwlock_rdlock(&lock);`
    - `pthread_rwlock_wrlock(&lock);`
    - `pthread_rwlock_unlock(&lock);`

Grading

- You will need to test your proxy very well
- There is no autograder, so you will have to come up with your own tests
- We will interview each of you, and test all aspects of your proxy like
  - Handling multiple requests
  - Proper caching
  - Implementation of LRU policy
  - And everything else!

Any Questions?