

# Recitation 12: ProxyLab Part 1

Instructor: TA(s)

# Outline

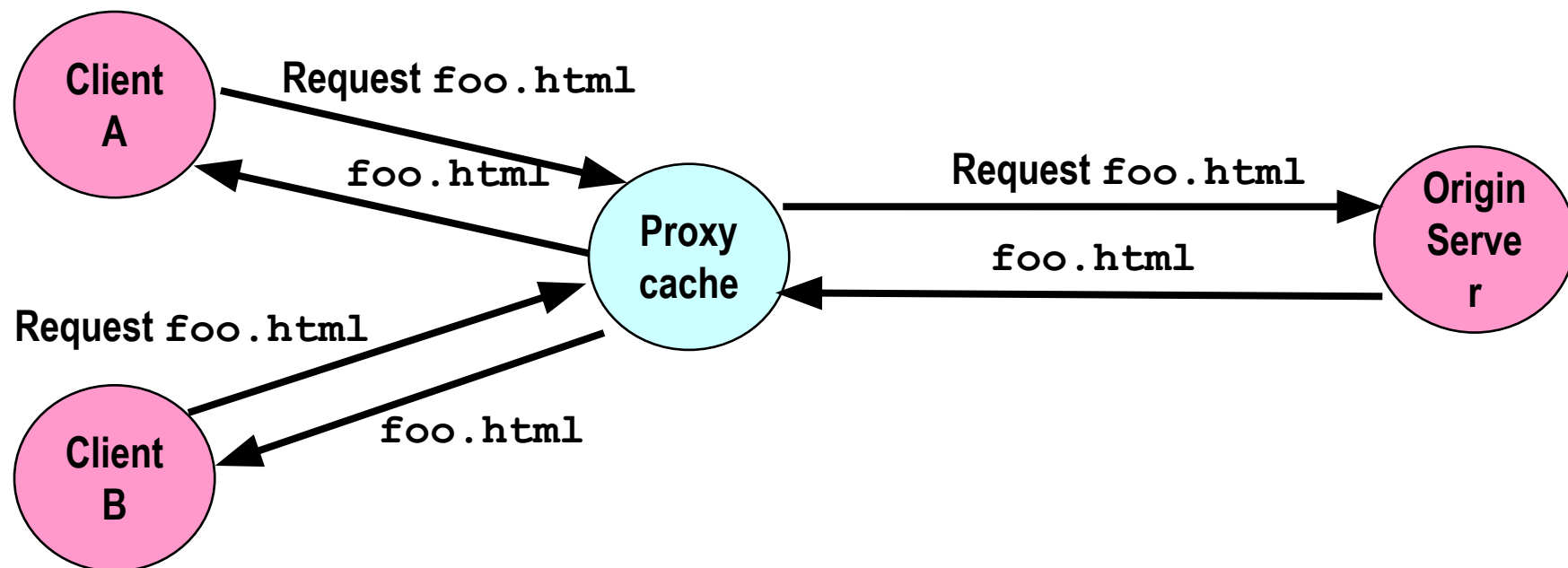
- **Proxies**
- **Networking**
- **Networking Demos**

# Proxy Lab

- **There are no grace days / late submissions**
  - 8% of final grade
- **You are submitting an entire project**
  - Modify the makefile
  - Split source file into separate pieces
- **Submit regularly to verify proxy builds on Autolab**
- **Your proxy is a server, it should not crash!**

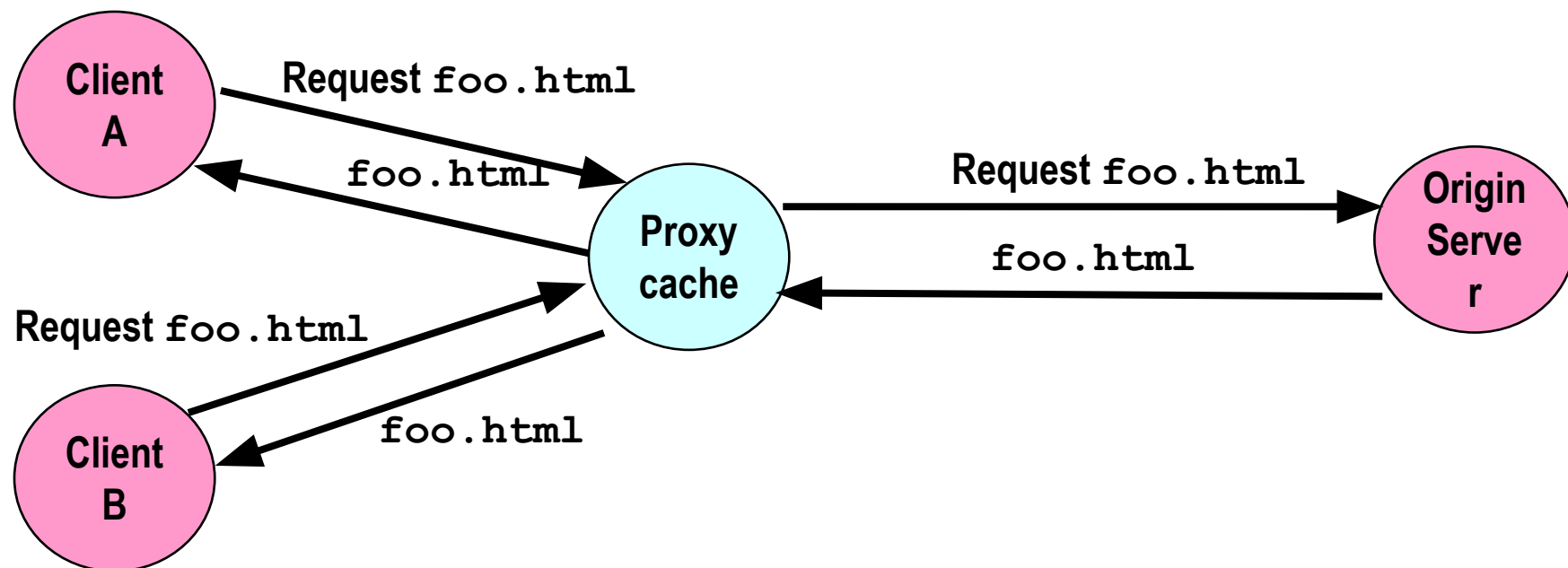
# What is a Proxy?

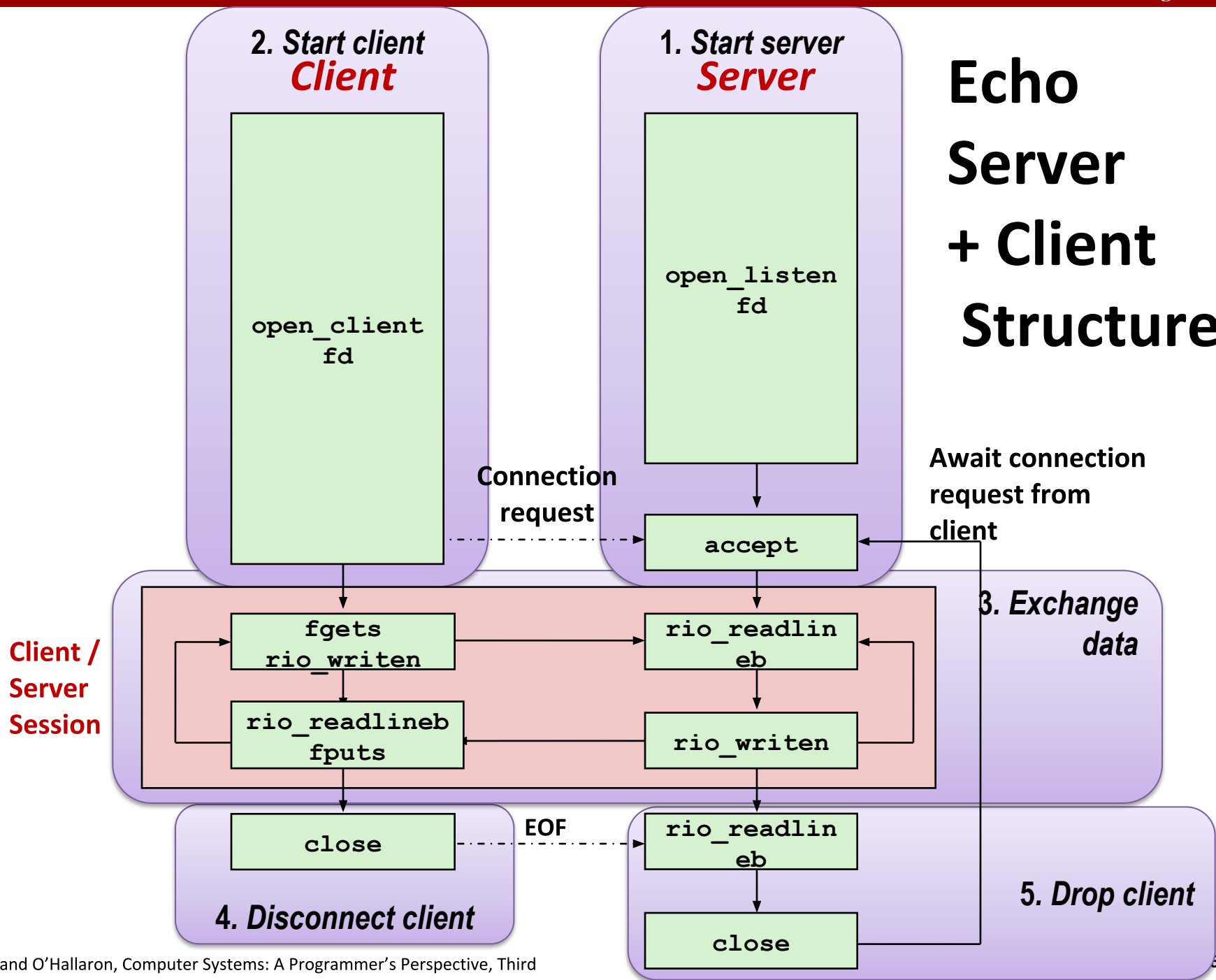
- Proxies are both clients and servers
  - Fetch page from server (eg. [cnn.com](http://cnn.com)) as a client
  - Serve page to client (eg. browser) as a server



# Why Proxies?

- Proxies are both clients and servers
- Can perform useful functions as requests and responses pass by
  - Examples: Caching, logging, anonymization, filtering, transcoding





# Transferring HTTP Data

- If something requests a file from a web server, how does it know that the transfer is complete?
  - A) It reads a NULL byte
  - B) The connection closes
  - C) It reads a blank line
  - D) The HTTP header specifies the number of bytes to receive
  - E) The reading function receives EOF

# Telnet Demo

- **Telnet is valuable for manually testing your proxy**

- What are valid requests to web servers?
- What do valid replies look like?

- **Connect to a shark machine**

- `$ telnet www.cs.cmu.edu 80`

```
GET /~213/recitations/rec12.html HTTP/1.0 <press enter>  
<press enter>
```

# Echo Demo

- **See the instructions written in the telnet results to set up the echo server. Get someone nearby to connect using the echo client.**
- **What does echoserver output?**

# Echo Demo

- See the instructions written in the telnet results to set up the echo server. Get someone nearby to connect using the echo client.

- What does echoserver output? (Sample output:)

```
$ ./echoserver 10101
```

```
Accepted connection from hammerheadshark.ics.cs.cmu.edu:46422
```

```
hammerheadshark.ics.cs.cmu.edu:46422 sent 6 bytes
```

```
Disconnected from hammerheadshark.ics.cs.cmu.edu:46422
```

# Echo Demo

- See the instructions written in the telnet results to set up the echo server. Get someone nearby to connect using the echo client.

- What does echoserver output? (Sample output:)

```
$ ./echoserver 10101
```

```
Accepted connection from hammerheadshark.ics.cs.cmu.edu :46422
```

```
hammerheadshark.ics.cs.cmu.edu:46422 sent 6 bytes
```

```
Disconnected from hammerheadshark.ics.cs.cmu.edu:46422
```

Server  
listening port

Client  
host

Client  
port

# Echo Demo

- **Look at echoclient.c**
  - Opens a connection to the server
  - Reads/writes from the server
- **Look at echoserver output**
  - Why is the printed client port different from the server's listening port?

# Echo Demo

## ■ Look at echoclient.c

- Opens a connection to the server
- Reads/writes from the server

## ■ Look at echoserver output

- Why is the printed client port different from the server's listening port?
- Server opens **one “listening”** port
  - Incoming clients connect to this port
- Once server **accepts** a connection, it talks to client on a **different “ephemeral”** port



# Echo Demo

- Try to connect two clients to the same server.
- What happens?

# Echo Demo

- **Try to connect two clients to the same server.**
- **What happens?**
  - Second client has to wait for first client to finish!
  - Server doesn't even accept second client's connection
  - Where/why are we getting stuck?

# Echo Demo

- **Try to connect two clients to the same server.**
- **What happens?**
  - Second client has to wait for first client to finish!
  - Server doesn't even accept second client's connection
  - Where/why are we getting stuck?
- **Because we're stuck in `echo()` talking to the first client, echoserver can't handle any more clients**
- **Solution: multi-threading**

# Echo Server Multithreaded

- How might we make this server multithreaded?  
(Don't look at echoserver\_t.c)

```
while (1) {  
    // Allocate space on the stack for client info  
    client_info client_data;  
    client_info *client = &client_data;  
  
    // Initialize the length of the address  
    client->addrlen = sizeof(client->addr);  
  
    // Accept() will block until a client connects to the port  
    client->connfd = Accept(listenfd,  
        (SA *) &client->addr, &client->addrlen);  
  
    // Connection is established; echo to client  
    echo(client);  
}
```

# Echo Server Multithreaded

- View the code in `echoserver_t.c`
- I will run `echoserver_t`
  - Try to connect to it

# Echo Server Multithreaded

- **echoserver\_t.c isn't too different from echoserver.c**
  - To see the changes:  
`$ git diff echoserver.c echoserver_t.c`
- **Making your proxy multithreaded will be very similar**
- **However, don't underestimate the difficulty of addressing race conditions between threads!**
  - Definitely the hardest part of proxylab
  - More on this next time...

# Makefiles

- View the Makefile in the recitation tarball

```
$ cat Makefile
```

# Makefile Basics

- List of instructions to build program(s)
- Each rule is called a *recipe*
  - Target specifies what to build
  - Followed by dependencies (on the same line)
- Makefile format:

```
all: $(OBJECTS) $(PROJECTS)
echoclient: csapp.o echoclient.o
```

Target

Dependencies

# Makefiles and Linking

- Recall the Linking lecture
  - What is the first step?
  - How does the final executable get built?

`all: $(OBJECTS) $(PROJECTS)`

**Target**

`echoclient: csapp.o echoclient.o`

**Dependencies**

# Makefiles and Modularity

- Split implementation into multiple files
  - Keeps closely related parts together
  - Easier to debug
  - Easier to reason about
- Use `static` functions for encapsulating helper functions
  - Similar to private methods in OOP

# Reminders

- **Read the writeup**
- **Start early**
  - Remember, no late submissions
  - Come to office hours this week, before it gets crowded!
- **Work incrementally and take breaks**