Proxy

Web & Concurrency

Hartaj Singh Dugal

15/18-213 - Section H

Recitation 13

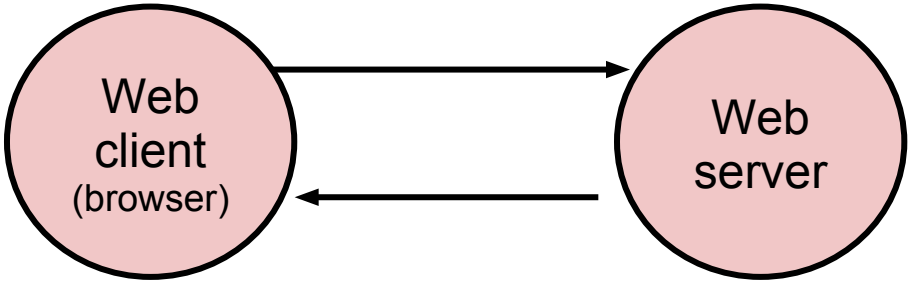
April 14th, 2014

Outline

- **Getting content on the web: Telnet/cURL Demo**
- **How the web really works**
- **Proxy**
  - Out Tuesday, April 15th
  - Due Tuesday, April 29th
  - Will be working with a partner this year
- **Threading**
  - Semaphores & Mutexes
  - Reader-Writer Locks

The Web in a Textbook

- **Client requests a page -> Server provides the desired page.**
  Transaction Completed.



- **A sequential server can handle this. We just need to serve one page at a time.**
- **This works great for simple text pages with embedded styles.**

Telnet/Curl Demo

Telnet

- Interactive remote shell – like ssh without security
- Must build HTTP request manually
  - This can be useful if you want to test response to malformed headers.

```
hartaj@ubuntu:~$ telnet www.cmu.edu 80
Trying 128.2.42.52...
Connected to WWW-CMU-PROD-VIP.ANDREW.cmu.edu.
Escape character is '^]'.
GET http://www.cmu.edu/ HTTP/1.0

HTTP/1.1 301 Moved Permanently
Date: Sun, 13 Apr 2014 22:21:11 GMT
Server: Apache/1.3.42 (Unix) mod_gzip/1.3.26.1a mod_pubcookie/3.3.4a mod_ssl/2.8.
31 OpenSSL/0.9.8e-fips-rhel5
Location: http://www.cmu.edu/index.shtml
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>301 Moved Permanently</TITLE>
</HEAD><BODY>
<H1>Moved Permanently</H1>
The document has moved <A HREF="http://www.cmu.edu/index.shtml">here</A>.<P>
<HR>
<ADDRESS>Apache/1.3.42 Server at <A HREF="mailto:webmaster@andrew.cmu.edu">www.cm
u.edu</A> Port 80</ADDRESS>
</BODY></HTML>
```

Telnet/cURL Demo

cURL

- "URL transfer library" with a command line program
- Builds valid HTTP requests for you!

```
hartaj@ubuntu:~$ curl http://www.cmu.edu/
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>301 Moved Permanently</TITLE>
</HEAD><BODY>
<H1>Moved Permanently</H1>
The document has moved <A HREF="http://www.cmu.edu/index.shtml">here</A>.<P>
<HR>
<ADDRESS>Apache/1.3.42 Server at <A HREF="mailto:webmaster@andrew.cmu.edu">www.cm
u.edu</A> Port 80</ADDRESS>
</BODY></HTML>
```

- Can also be used to generate HTTP proxy requests:

```
hartaj@ubuntu:~$ curl --proxy bambooshark.ics.cs.cmu.edu:47910 http://www.cmu.edu
/
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>301 Moved Permanently</TITLE>
</HEAD><BODY>
<H1>Moved Permanently</H1>
The document has moved <A HREF="http://www.cmu.edu/index.shtml">here</A>.<P>
<HR>
<ADDRESS>Apache/1.3.42 Server at <A HREF="mailto:webmaster@andrew.cmu.edu">www.cm
u.edu</A> Port 80</ADDRESS>
</BODY></HTML>
```

How the Web Really Works

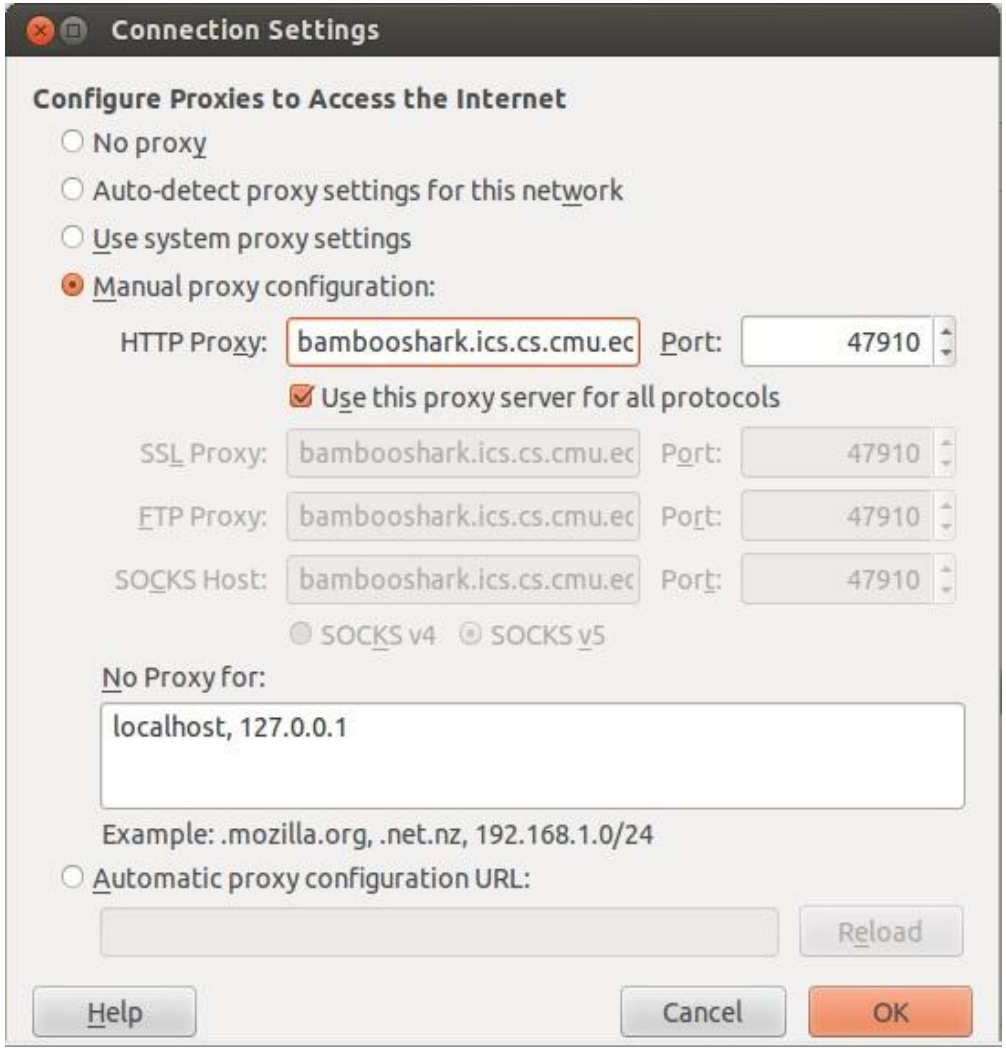
- **In reality, a single HTML page today may depend on 10s or 100s of support files (images, stylesheets, scripts, etc.)**
- **Builds a good argument for concurrent servers**
  - Just to load a single modern webpage, the client would have to wait for 10s of back-to-back request
  - I/O is likely slower than processing, so back to caching.
- **Caching is simpler if done in pieces rather than whole page**
  - If only part of the page changes, no need to fetch old parts again
  - Each object (image, stylesheet, script) already has a unique URL that can be used as a key

How the Web Really Works

- **Excerpt from www.cmu.edu/index.html:**

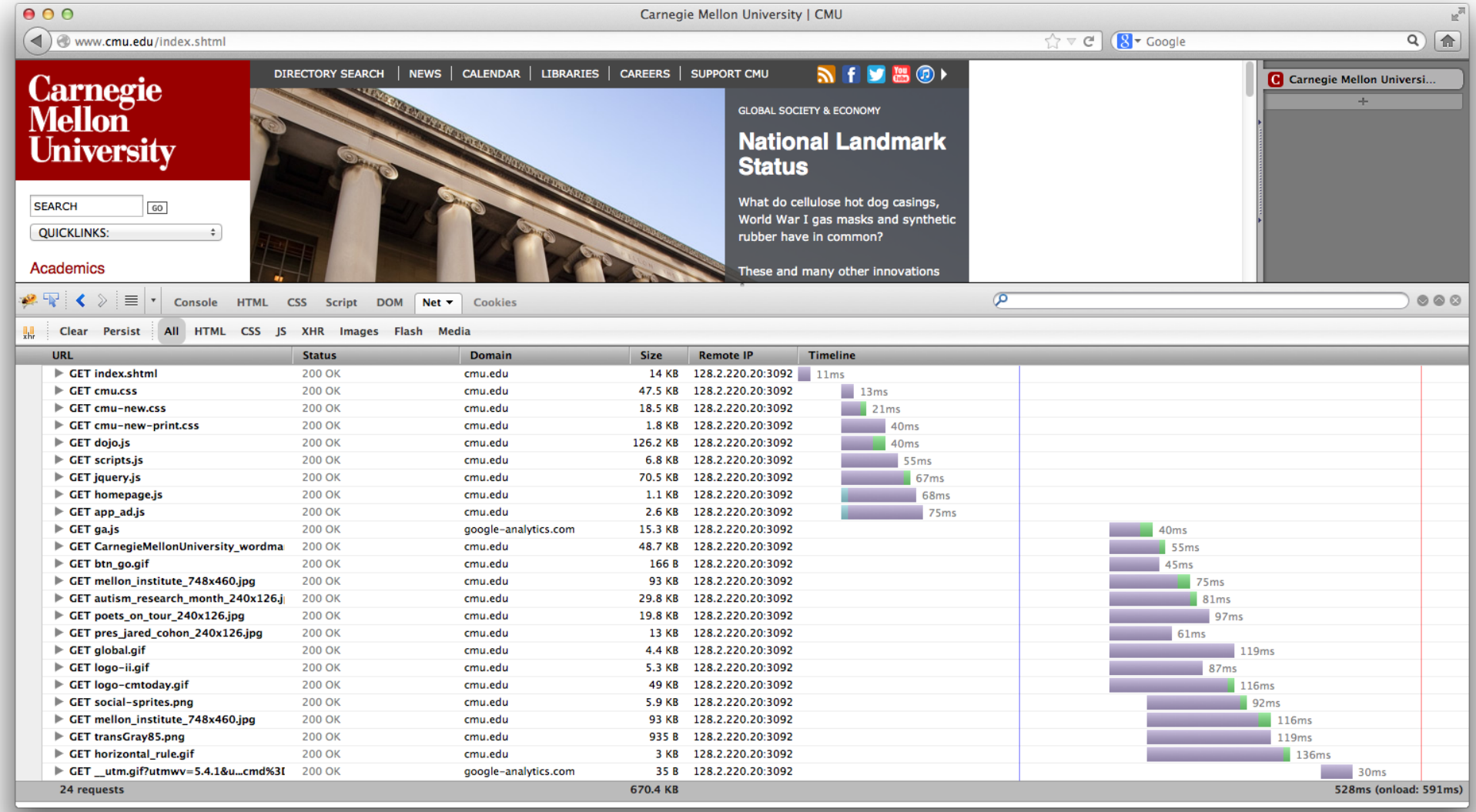
```
<html lang="en" xml:lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
  ...
  <link href="homecss/cmu.css" rel="stylesheet" type="text/css"/>
  <link href="homecss/cmu-new.css" rel="stylesheet" type="text/css"/>
  <link href="homecss/cmu-new-print.css" media="print" rel="stylesheet" type="text/css"
/>
  <link href="http://www.cmu.edu/RSS/stories.rss" rel="alternate" title="Carnegie
Mellon Homepage Stories" type="application/rss+xml"/>
  ...
  <script language="JavaScript" src="js/dojo.js" type="text/javascript"></script>
  <script language="JavaScript" src="js/scripts.js" type="text/javascript"></script>
  <script language="javascript" src="js/jquery.js" type="text/javascript"></script>
  <script language="javascript" src="js/homepage.js" type="text/javascript"></script>
  <script language="javascript" src="js/app_ad.js" type="text/javascript"></script>
  ...
  <title>Carnegie Mellon University | CMU</title>
</head>
<body> ...
```

Aside: Setting up Firefox to use a proxy



- You may use any browser, but we'll be grading with Firefox
- Preferences > Advanced > Network > Settings… (under Connection)
- Check "Use this proxy for all protocols" or your proxy will appear to work for HTTPS traffic.

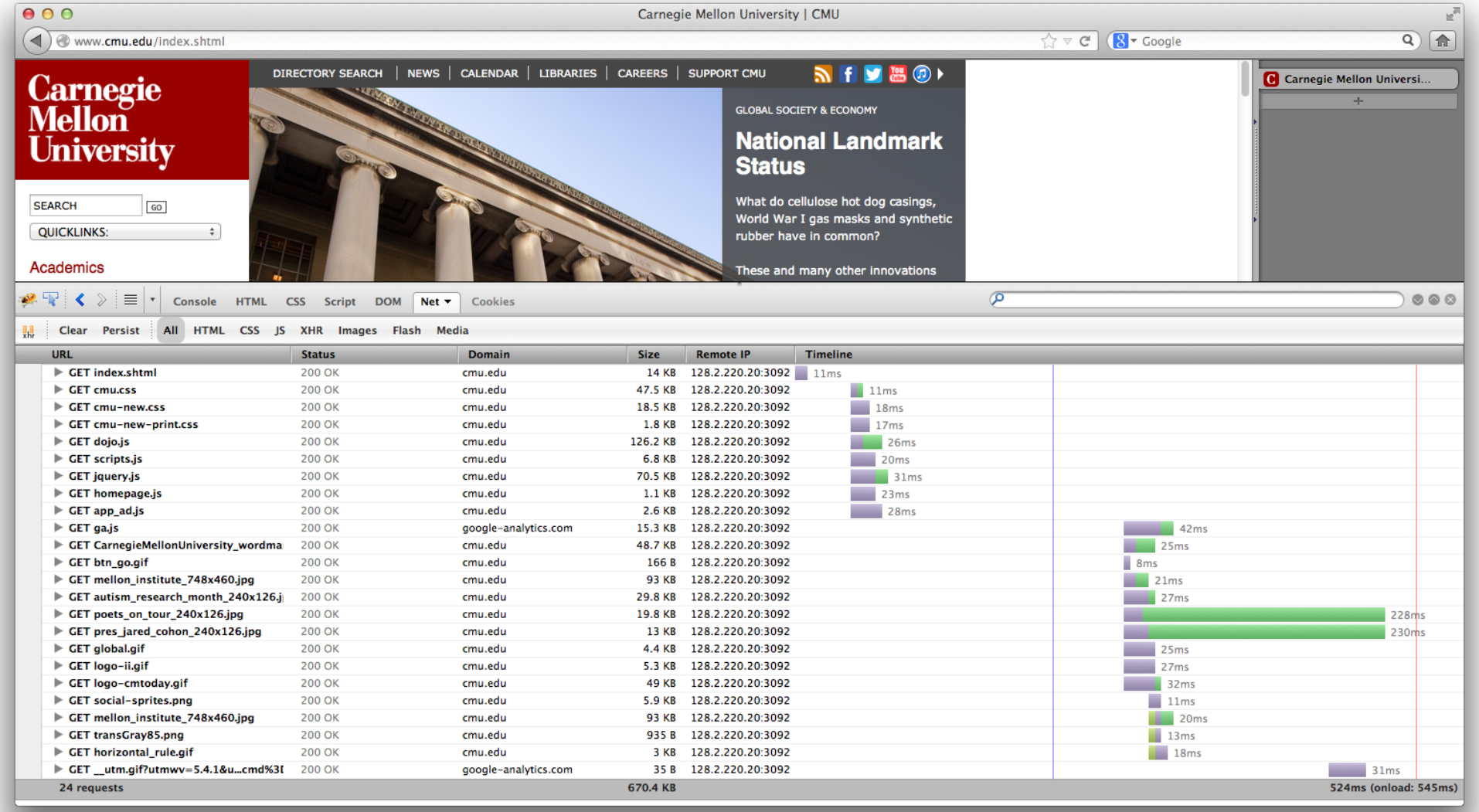
Sequential Proxy



Sequential Proxy

- **Note the sloped shape of when requests finish**
  - Although many requests are made at once, the proxy does not accept a new job until it finishes the current one
  - Requests are made in batches. This results from how HTML is structured as files that reference other files.
- **Compared to the concurrent example (next), this page takes a long time to load with just static content**

Concurrent Proxy



Concurrent Proxy

- **Now, we see much less purple (waiting), and less time spent overall.**
- **Notice how multiple green (receiving) blocks overlap in time**
  - Our proxy has multiple connections open to the browser to handle several tasks at once

How the Web Really Works

- **A note on AJAX (and XMLHttpRequests)**
  - Normally, a browser will make the initial page request then request any supporting files
  - And XMLHttpRequest is simply a request from the page once it has been loaded & the scripts are running
  - The distinction does not matter on the server side – everything is an HTTP Request

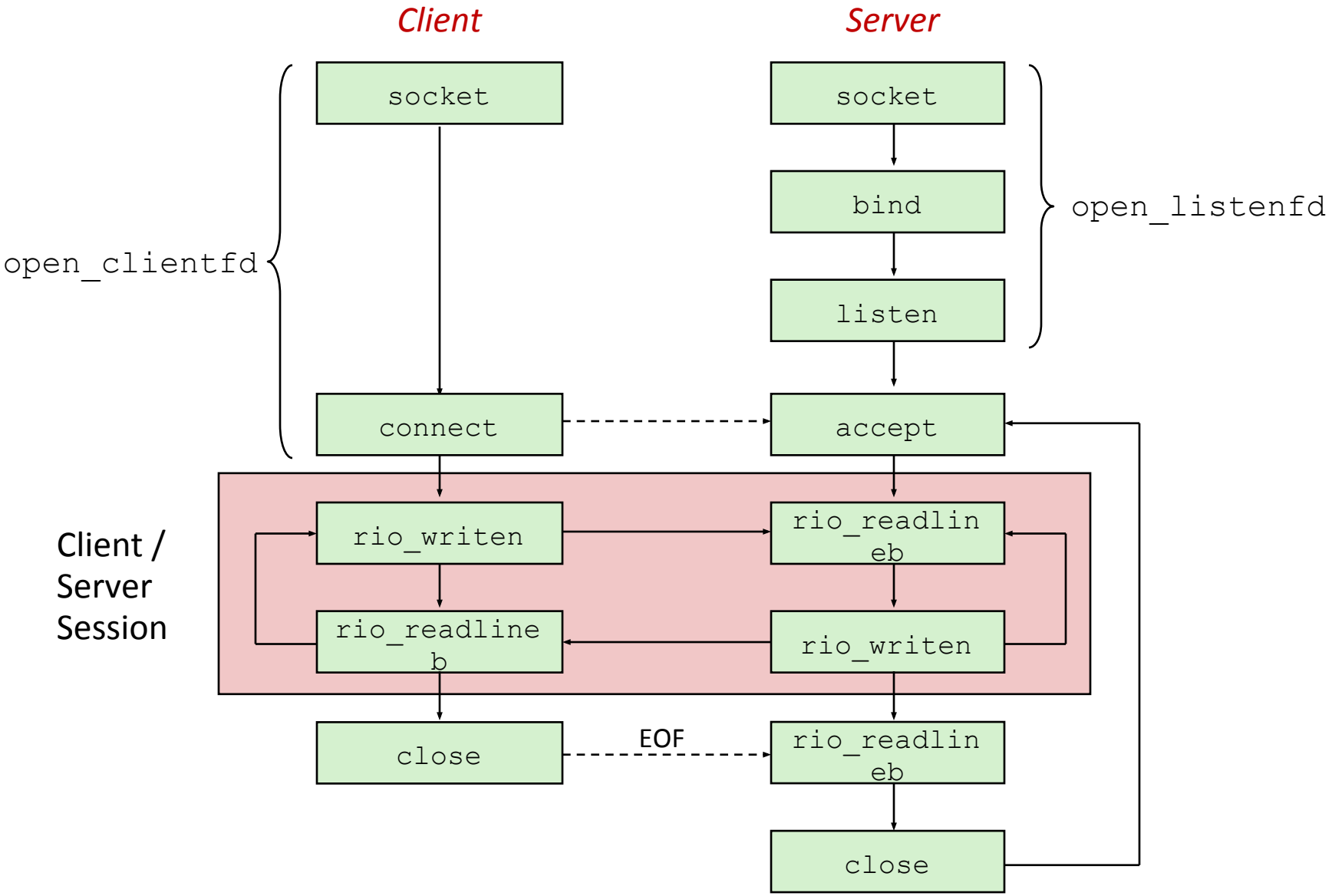
Proxy - Functionality

- **Should work on vast majority of sites**
  - Reddit, Vimeo, CNN, YouTube, NY Times, etc.
  - Some features of sites which require the POST operation (sending data to the website), will not work
    - Logging in to websites, sending Facebook message
  - HTTPS is not expected to work
    - Google (and some other popular websites) now try to push users to HTTPs by default; watch out for that
- **Cache previous requests**
  - Use LRU eviction policy
  - Must allow for concurrent reads while maintaining consistency
  - Details in write up

Proxy - Functionality

- **Why a multi-threaded cache?**
  - Sequential cache would bottleneck parallel proxy
  - Multiple threads can read cached content safely
    - Search cache for the right data and return it
    - Two threads can read from the same cache block
  - But what about writing content?
    - Overwrite block while another thread reading?
    - Two threads writing to same cache block?

Proxy - How

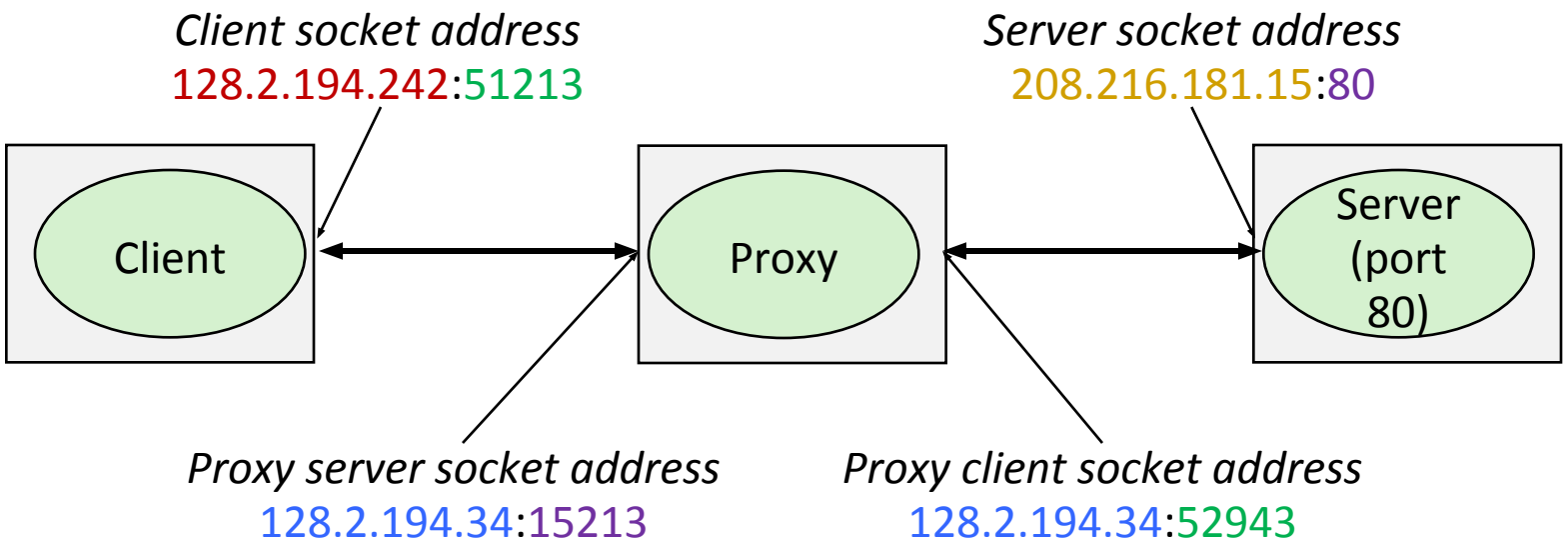

Client
Server
socket
socket
bind
listen
open_clientfd
open_listenfd
connect
accept
Client / Server Session
rio_writen
rio_readlineb
rio_readlineb
rio_writen
close
EOF
rio_readlineb
close

Proxy - How

- **Remember that picture?**
  - Proxies are a bit special; they are a server and a client at the same time.
  - They take a request from one computer (acting as the server), and make it on their behalf (as the client).
  - Ultimately, the control flow of your program will look like a server, but will have to act as a client to complete the request
- **Start small**
  - Grab yourself a copy of the echo server (pg. 910) and client (pg. 909) in the book
  - Also review the tiny.c basic web server code to see how to deal with HTTP headers
    - Note that tiny.c ignores these; you may not

Proxy - How

- **What you end up with will resemble:**



Proxy – Testing & Grading

- **Autograder**
  - ./driver.sh will run the same tests as autolab:
    - Ability to pull basic web pages from a server
    - Handle a (concurrent) request while another request is still pending
    - Fetch a web page again from your cache after the server has been stopped
  - This should help answer the question “is this what my proxy is supposed to do?”
  - Please don’t use this grader to definitively test your proxy; there are many things not tested here

Proxy – Testing & Grading

- **Test your proxy liberally**
  - The web is full of special cases that want to break your proxy
  - Generate a port for yourself with ./port-for-user.pl [andrewid]
  - Generate more ports for web servers and such with ./free-port.sh
  - Consider using your andrew web space (~/www) to host test files
    - You have to visit https://www.andrew.cmu.edu/server/publish.html to publish your folder to the public server
- **Create a handin file with *make handin***
  - Will create a tar file for you with the contents of your proxylab-handin folder

Mutexes & Semaphores

- **Mutexes**
  - Allow only one thread to run code section at a time
  - If other threads are trying to run the code, they will wait
- **Semaphores**
  - Allows a fixed number of threads to run the code
  - Mutexes are a special case of semaphores, where the number of threads=1
    - Examples will be done with semaphores to illustrate

Read-Write Lock

- **Also called a Read*ers*-Writ*er* lock in the notes**
- **Cache can be read in parallel safely**
- **If thread is writing, no other thread can read or write**
- **If thread is reading, no other thread can write**
- **Potential issues**
  - Writing starvation
    - If threads always reading, no thread can write
    - Fix: if a thread is waiting to write, it gets priority over any new threads trying to read
- **How can we lock out threads?**

Read-Write Locks Cont.

- **How would you make a read-write lock with semaphores?**
  - Luckily, you don't have to!
    - pthread_rwlock_* handles that for you
      - pthread_rwlock_t lock;
      - pthread_rwlock_init(&lock,NULL);
      - pthread_rwlock_rdlock(&lock);
      - pthread_rwlock_wrlock(&lock);
      - pthread_rwlock_unlock(&lock);

Questions?