COMP 206 – Intro to Software Systems

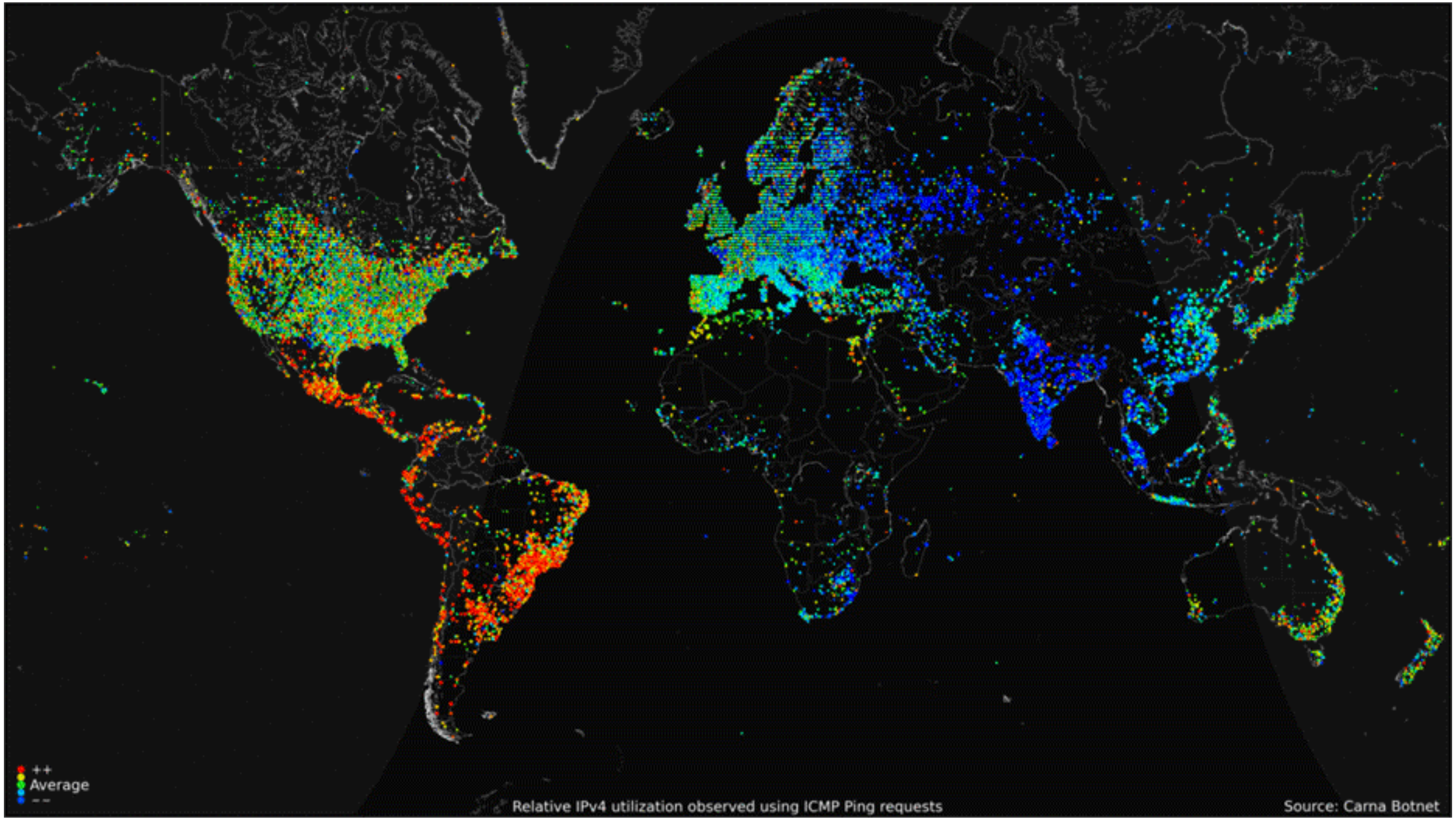
Lecture 17 – Networking and Multi-process Overview

Nov 7, 2018

Outline

- A crazy quick tour of the remaining concepts in 206 – no detail just a preview of the Internet, Web and distributed computing
- The beginnings: how do two processes connect and communicate?
- A first code example: networking hello world.

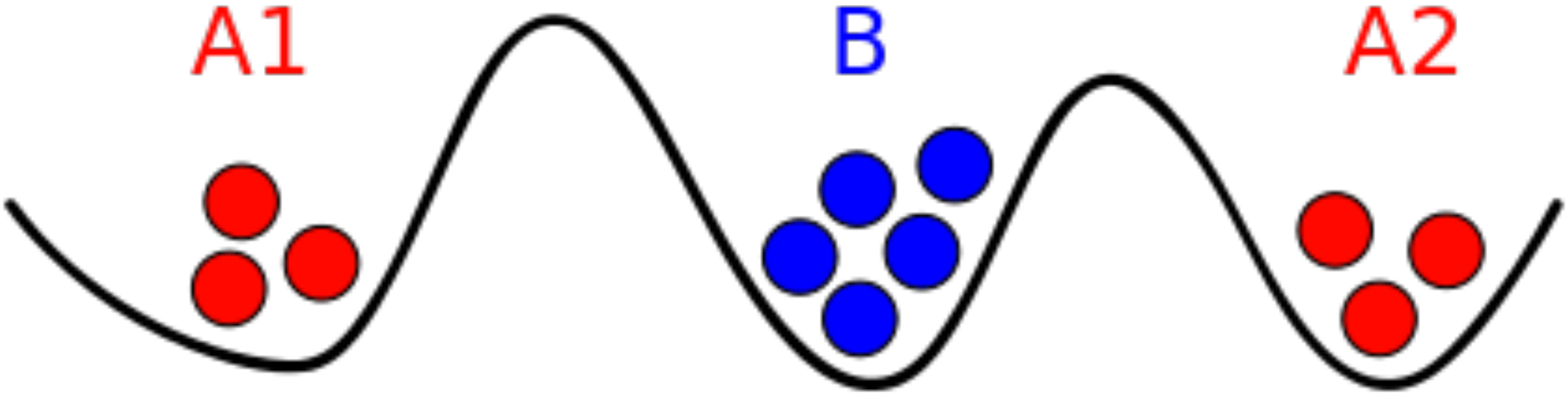
Where are we going?



Relative IPv4 utilization observed using ICMP Ping requests

Source: Carna Botnet

The Two Generals Problem

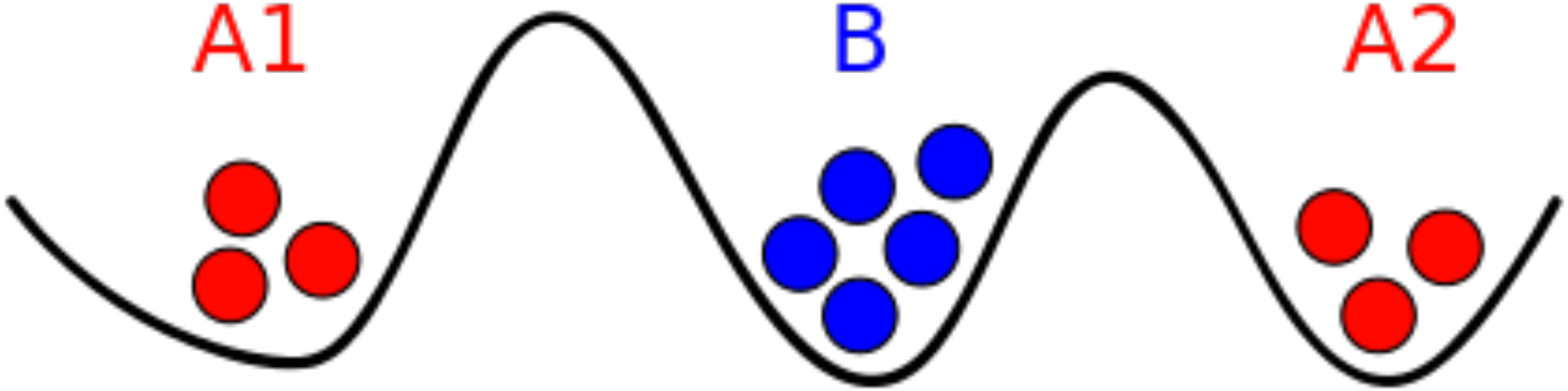


The Two Communicating General Game

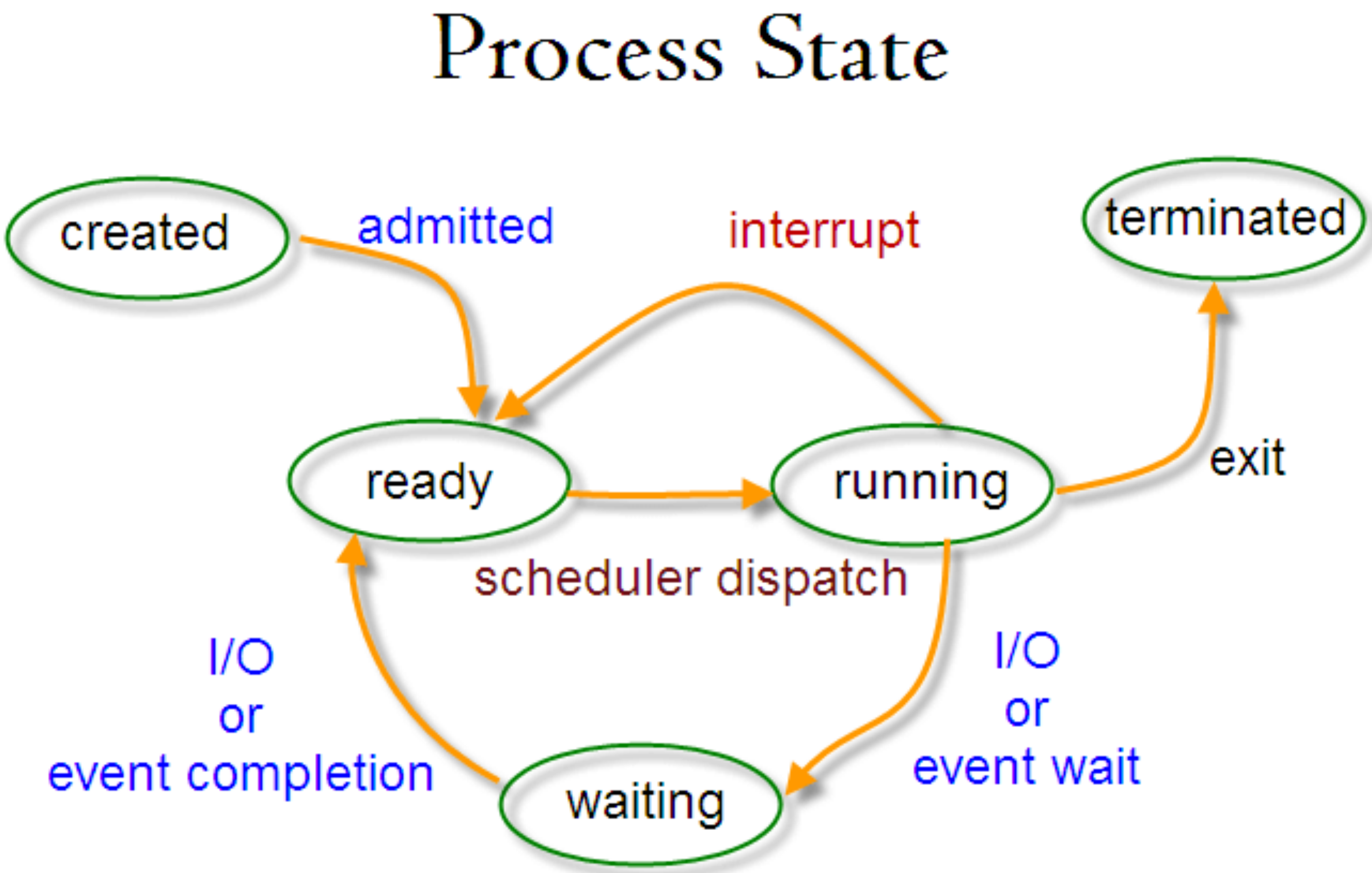
- Attackers will win easily if attack at the same time
- Defenders will win easily if only a single side attacks
- Attackers can send messengers to eachother, with whatever note they desire, but no way to tell if they arrived (the hills block sound, light, cell signals, perfectly!)
- Must then make a decision. Based on what I sent, and/or received, will I attack today?

- Let’s play this in class (sorry middle section…)

The Two Generals: A fundamental problem
Is it possible to guarantee coordination?



A process does not simply "run"


Process State

Multiple Processes that work together

- In principle, spreading computation across N computers gives the possibility for an N-fold increase in computing power. Finish jobs in 1/N of the time.
- In practice, due to communication delays, congestion, waiting, it can be difficult to get any speedup:
    - Think about a bad experience in a group project. Ouch!
- It is even possible to design a distributed system that never completes!

An example of discoordination

- The script on the left, called addTen.bash in ExampleCode is supposed to increase the single value in a text file
- Suppose you start with 0:
  - echo 0 > file
- Then want to "quickly" add 40:
  - bash add_ten.bash &

    bash add_ten.bash &

    bash add_ten.bash &

    bash add_ten.bash

```
#!/bin/bash
for i in 0 1 2 3 4 5 6 7 8 9
do
        x=`cat file`
        expr $x + 1 > file
done
```

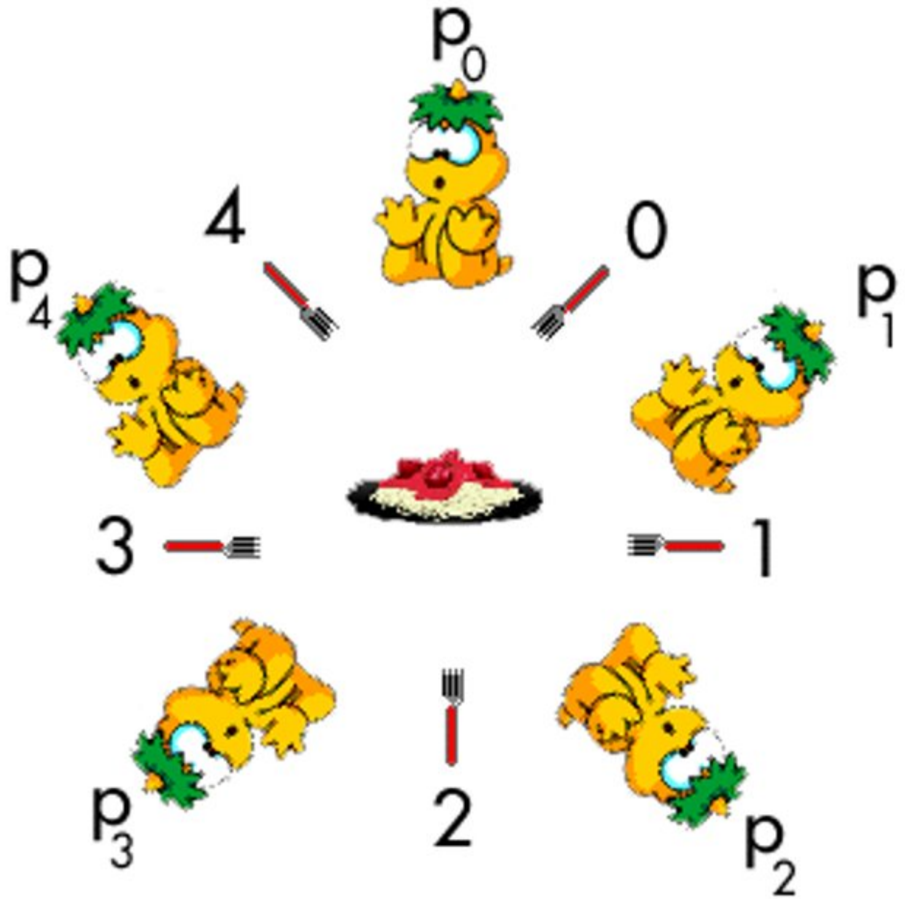
An example of discoordination

- The script on the left, called addTen.bash in ExampleCode is supposed to increase the single value in a text file
- Suppose you start with 0:
  - echo 0 > file
- Then want to “quickly” add 40:
  - bash add_ten.bash &
    bash add_ten.bash &
    bash add_ten.bash &
    bash add_ten.bash

```
#!/bin/bash
for i in 0 1 2 3 4 5 6 7 8 9
do
        x=`cat file`
        expr $x + 1 > file
done
```

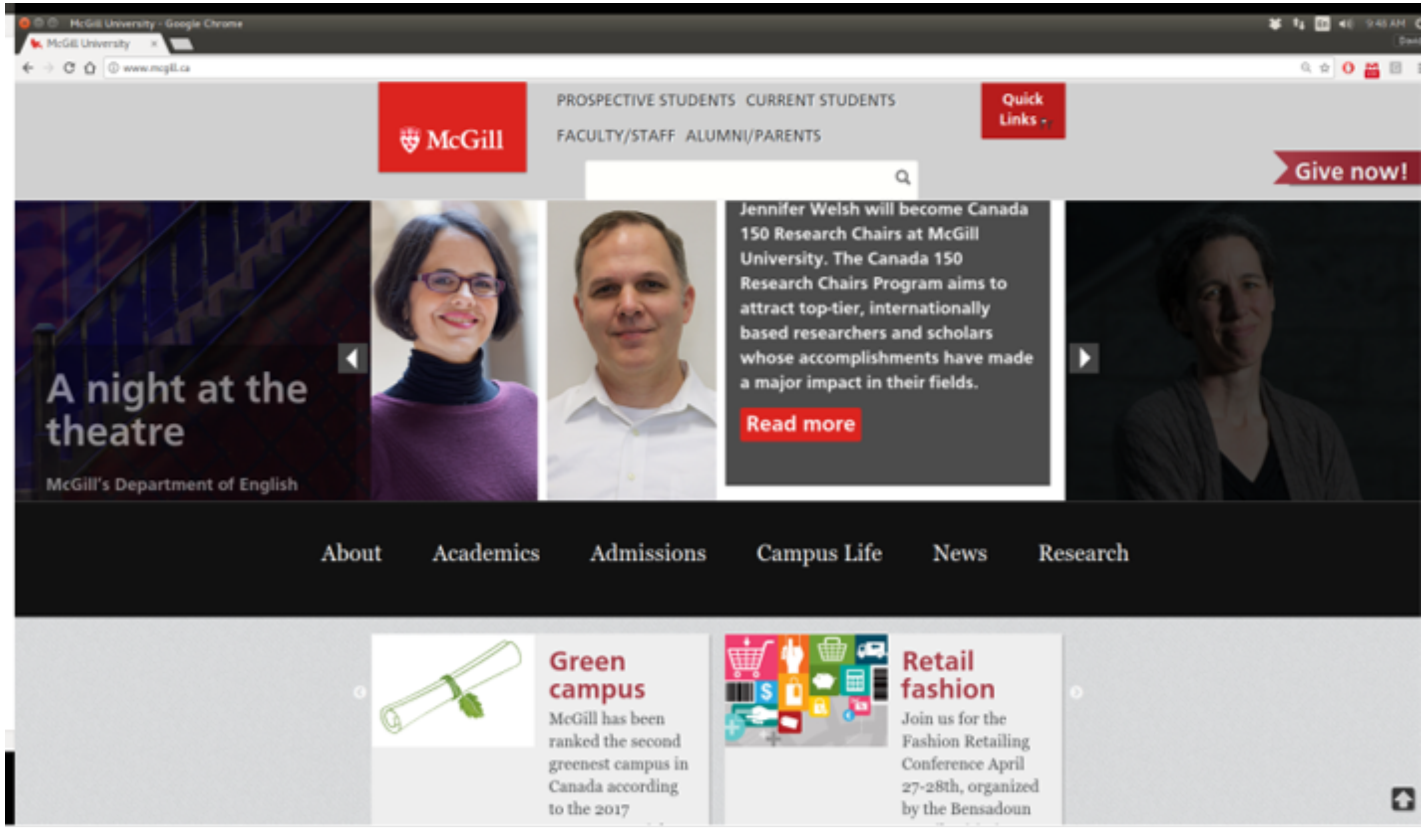
Another 206 weirdness… the answer is almost never 40!

A second fundamental problem:
The Dining (or starving?) Philosophers



Your Web Browser as a Software System

- You see rendered HTML including images, UI elements, color. How did this get there?



A night at the theatre

McGill's Department of English

Jennifer Welsh will become Canada 150 Research Chairs at McGill University. The Canada 150 Research Chairs Program aims to attract top-tier, internationally based researchers and scholars whose accomplishments have made a major impact in their fields.

Green campus

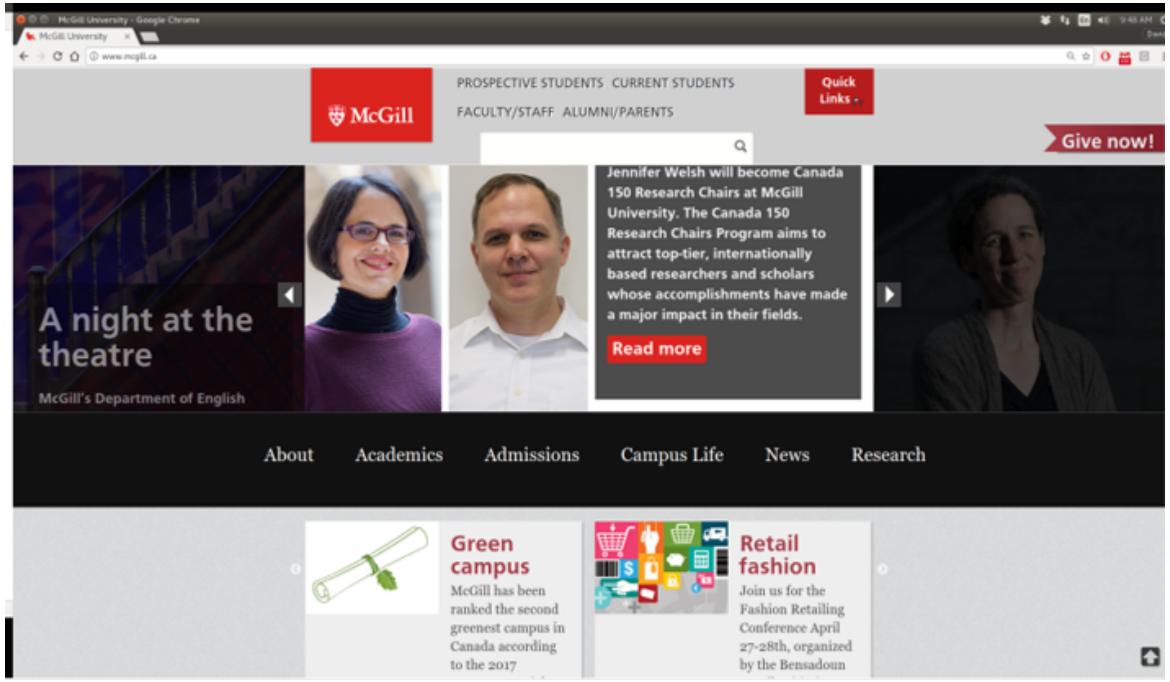
McGill has been ranked the second greenest campus in Canada according to the 2017

Retail fashion

Join us for the Fashion Retailing Conference April 27-28th, organized by the Bensadoun

Your Web Browser as a Software System

- Q: You see rendered HTML including images, UI elements, color. How did this get there?
- A: Your browser communicates with the server that provides the content. It has functionality to display a user interface that interprets the data received in a useful way and send requests for new content when you, for example, click a link



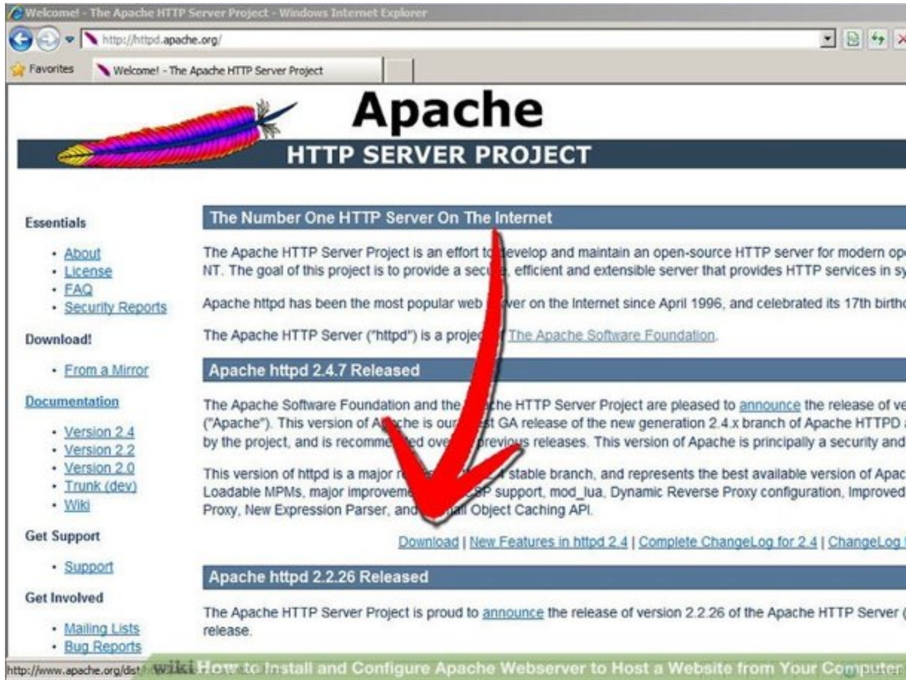
Jennifer Welsh will become Canada 150 Research Chairs at McGill University. The Canada 150 Research Chairs Program aims to attract top-tier, internationally based researchers and scholars whose accomplishments have made a major impact in their fields.

McGill has been ranked the second greenest campus in Canada according to the 2017

Join us for the Fashion Retailing Conference April 27-28th, organized by the Bensadoun

What about the web server?

- A server is on the other end making this transaction complete.
- The server holds the information and passes it to your browser when requested.
- This can be a single-shot: mcgill.ca homepage
- This can be on-going: play a movie



Welcome! - The Apache HTTP Server Project - Windows Internet Explorer
http://httpd.apache.org/
Favorites
Welcome! - The Apache HTTP Server Project
Apache
HTTP SERVER PROJECT

Essentials

- About
- License
- FAQ
- Security Reports

Download!

- From a Mirror

Documentation

- Version 2.4
- Version 2.2
- Version 2.0
- Trunk (dev)
- Wiki

Get Support

- Support

Get Involved

- Mailing Lists
- Bug Reports

The Number One HTTP Server On The Internet

The Apache HTTP Server Project is an effort to develop and maintain an open-source HTTP server for modern op
NT. The goal of this project is to provide a secure, efficient and extensible server that provides HTTP services in sy

Apache httpd has been the most popular web server on the Internet since April 1996, and celebrated its 17th birthd

The Apache HTTP Server ("httpd") is a project of The Apache Software Foundation.

Apache httpd 2.4.7 Released

The Apache Software Foundation and the Apache HTTP Server Project are pleased to announce the release of ve
("Apache"). This version of Apache is our latest GA release of the new generation 2.4.x branch of Apache HTTPD
by the project, and is recommended over all previous releases. This version of Apache is principally a security and

This version of httpd is a major r stable branch, and represents the best available version of Apac
Loadable MPMs, major improveme SP support, mod_lua, Dynamic Reverse Proxy configuration, Improved
Proxy, New Expression Parser, and all Object Caching API.

Download | New Features in httpd 2.4 | Complete ChangeLog for 2.4 | ChangeLog

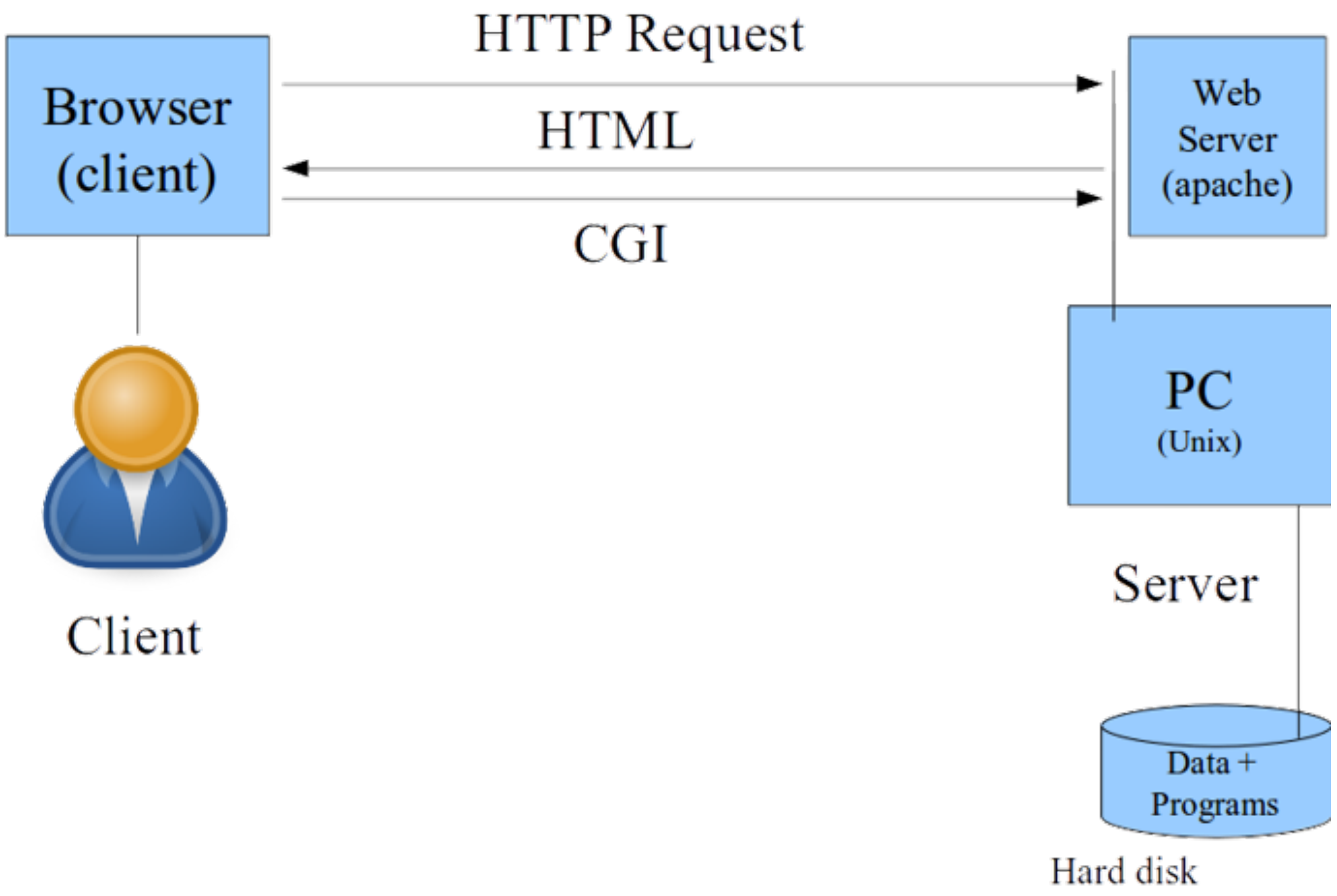
Apache httpd 2.2.26 Released

The Apache HTTP Server Project is proud to announce the release of version 2.2.26 of the Apache HTTP Server (
release.

http://www.apache.org/dist

wiki How to Install and Configure Apache Webserver to Host a Website from Your Computer

The web software system diagram



The first step on our journey: communication

- We’ll first conclude our “game”, the 2 Generals

- Based on its conclusion, we can understand something about the design of networked software systems

- Then we’ll complete today by writing a bit of our own communication code

Potential Solution #1

- Decide tomorrow is the day to attack
- Send a note "Attack at dawn"
- GO TO BATTLE (fingers crossed we're not alone!)

Outcome of Solution #1

- This is a (hopefully) clear failure. Whenever the message fails to arrive, we will be attacking alone.

Potential Solution #2

- Decide tomorrow is the day to attack
- Send a note "Attack at dawn: please acknowledge."
- If the acknowledgement is received:
    - Attack!
- Else:
    - Pick a new time and send the same note

Outcome of Solution #2

- If we attack, our allies will also (they have sent the acknowledgement)

- Sometimes neither of us attack (the first message was lost)

- Worse, sometimes, our allies will attack alone (they sent the ack, but it was lost)

Solution #3: Getting Impatient

- Want to make really sure about this, decide to attack in one week
- In the mean time, send one message per day "Attack on the 10th: please acknowledge."
- If any acknowledgement is received:
  - Attack!
- Else:
  - Feel quite depressed and call your Internet Service Provider

Generalizing this situation

- The answer to the Two Generals Problem is "NO"

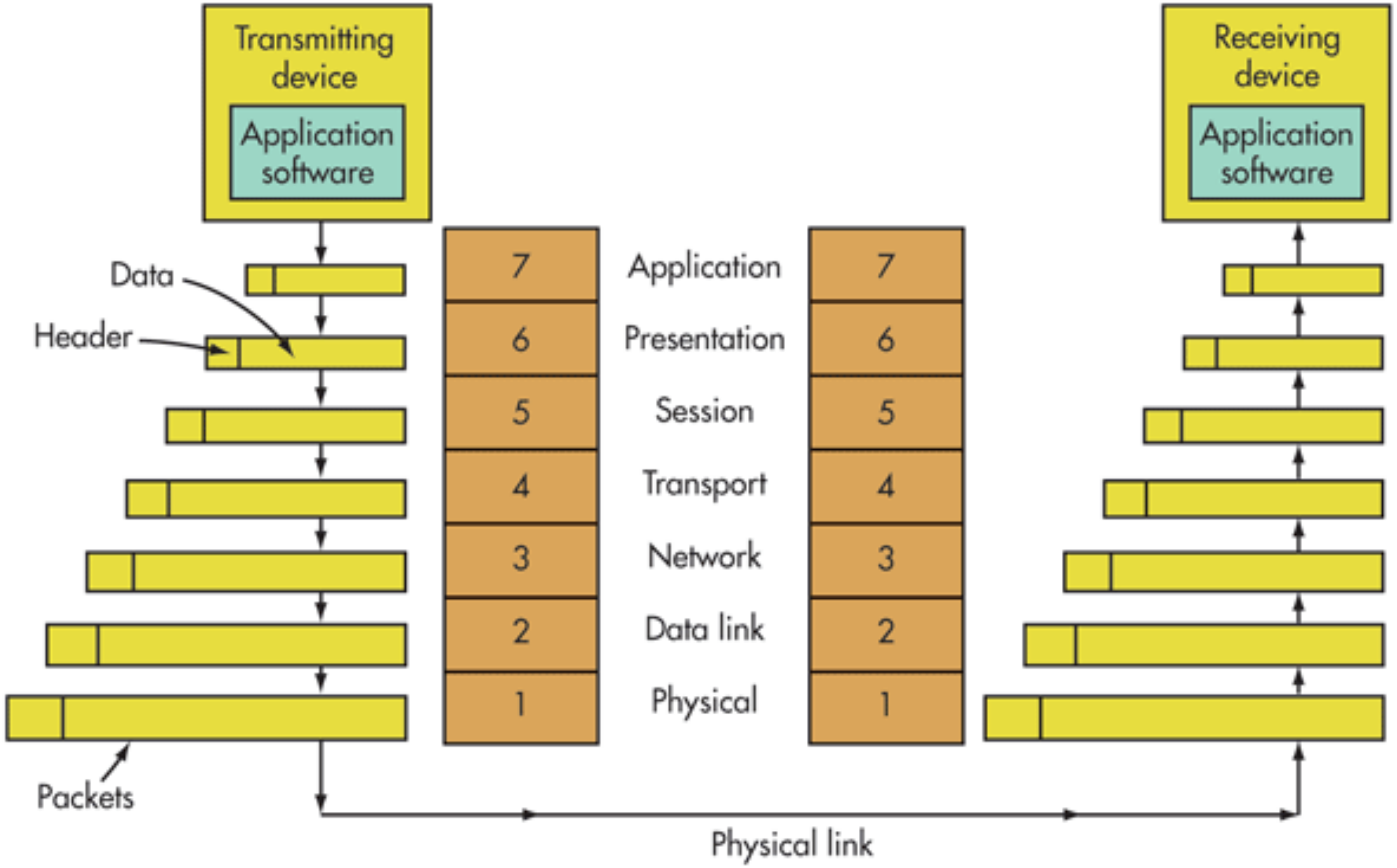
- There provably does not exist any protocol with finite number of messages for the generals to coordinate with certainty:
  - Proof sketch: The generals must decide on a pattern of messages that triggers attacking. Let an adversary block one of the B->A messages. Now side B sees the same pattern as previous, but side A is uncertain or thinks the attack is off.

- If we can send infinitely with non-zero probability of receipt, we can coordinate (but that's a lot of ink!)

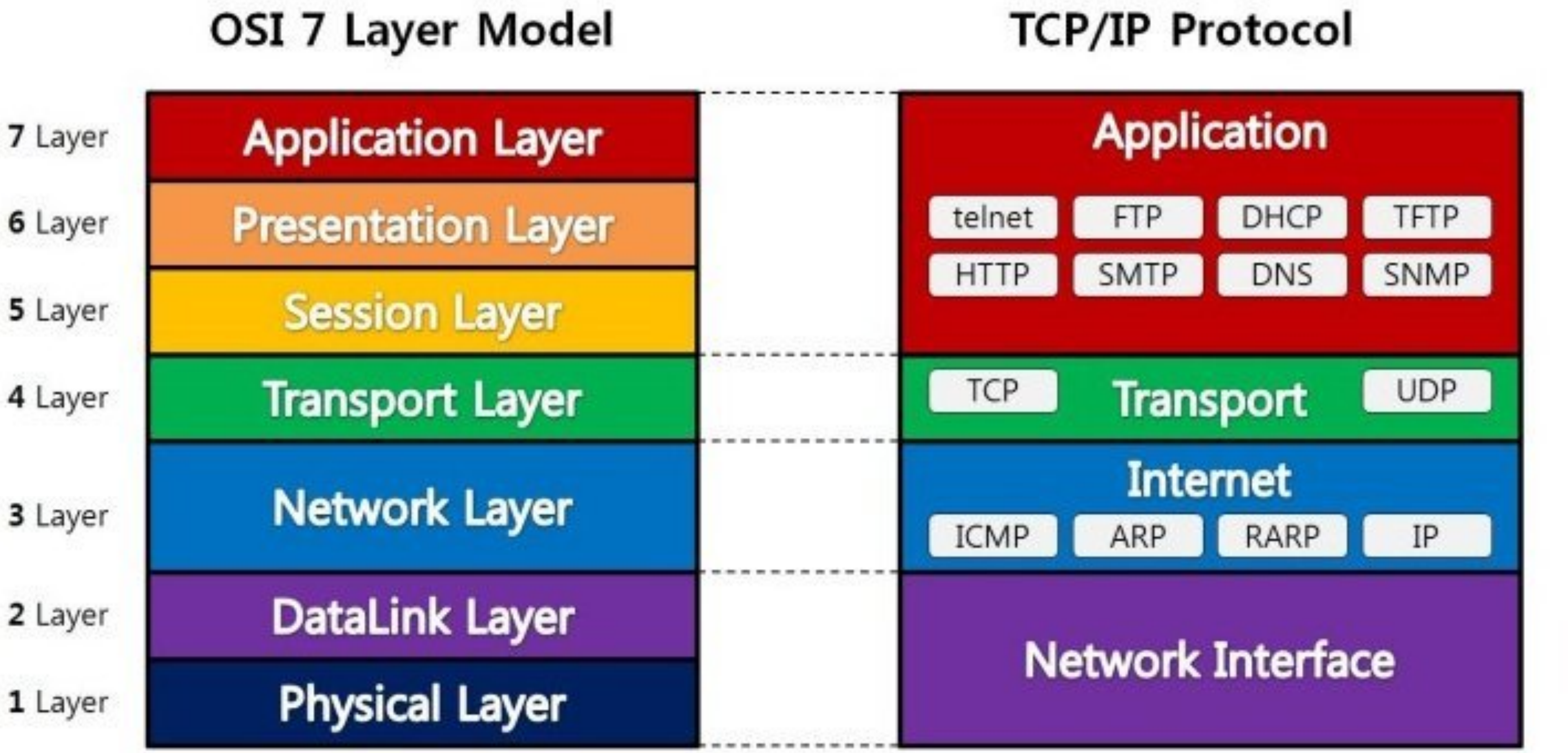
The Generals Applied to Networked Software Systems

- Sending a message on any realistic network is the same as the notes our Generals were passing
- Internet traffic is not guaranteed because:
  - The other end-point can have a failure (user ctrl-c, power turned off, etc)
  - There are many devices in between the end-points. They can be busy, malfunction, be turned off, etc.
  - There are physical connections involved: copper cables, fibre, radios transmitting Wifi signals. Each of these has the potential to fail.
- As a result, the Internet is a ***Best Effort*** transmission medium and our software system code must be prepared for this situation.

The Internet's Layers:
Open Systems Interconnection (OSI) model



What real software?



The Bottom Layers: Physical and Datalink

- This is not the business of COMP 206, but let's mention it just for completeness

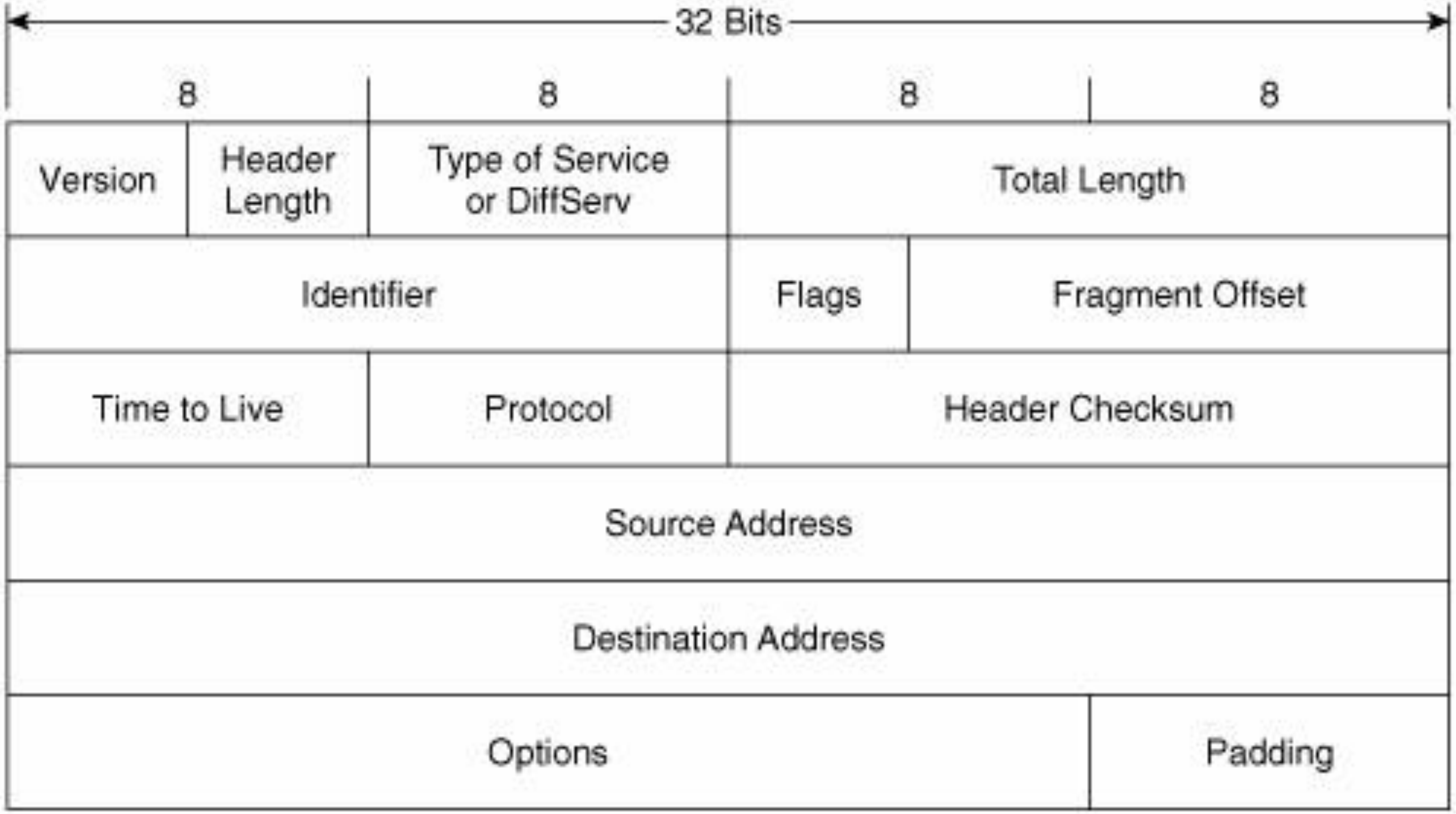
- Your network interface is managed by the Operating System. Wifi and Ethernet are the most common two options. The kernel may manage the network hardware directly or delegate to ***drivers***.

- The rest of the software on the computer doesn't need to think about this. It can ignore which way we're connected and even handle swapping connections without interruption!

The Network Layer: Internet Protocol

- The network layer provides functionality to get data across the world by bundling it into datagrams (aka packets) that are stamped with crucial information to allow transmission
    - Addresses of the source and destination. IP addresses can be 32-bit (IPv4) or 128-bit (IPv6).
    - Message length field (helps to ensure we've sent everything)
    - Time-to-live (helps to find out if the Internet has a cycle: It can happen!)
    - Header checksum (protects against errorful transmission)

IP Headers



<table>
<tr><th colspan="8">32 Bits</th></tr>
<tr><th colspan="2">8</th><th colspan="2">8</th><th colspan="2">8</th><th colspan="2">8</th></tr>
<tr><td>Version</td><td>Header Length</td><td colspan="2">Type of Service or DiffServ</td><td colspan="4">Total Length</td></tr>
<tr><td colspan="4">Identifier</td><td>Flags</td><td colspan="3">Fragment Offset</td></tr>
<tr><td colspan="2">Time to Live</td><td colspan="2">Protocol</td><td colspan="4">Header Checksum</td></tr>
<tr><td colspan="8">Source Address</td></tr>
<tr><td colspan="8">Destination Address</td></tr>
<tr><td colspan="6">Options</td><td colspan="2">Padding</td></tr>
</table>

The IP Header Checksum: bit operations!

- The checksum field is the 16-bit one's complement (negation) of the one's complement sum (handling overflow) of all 16-bit words in the header. For purposes of computing the checksum, the value of the checksum field is zero.

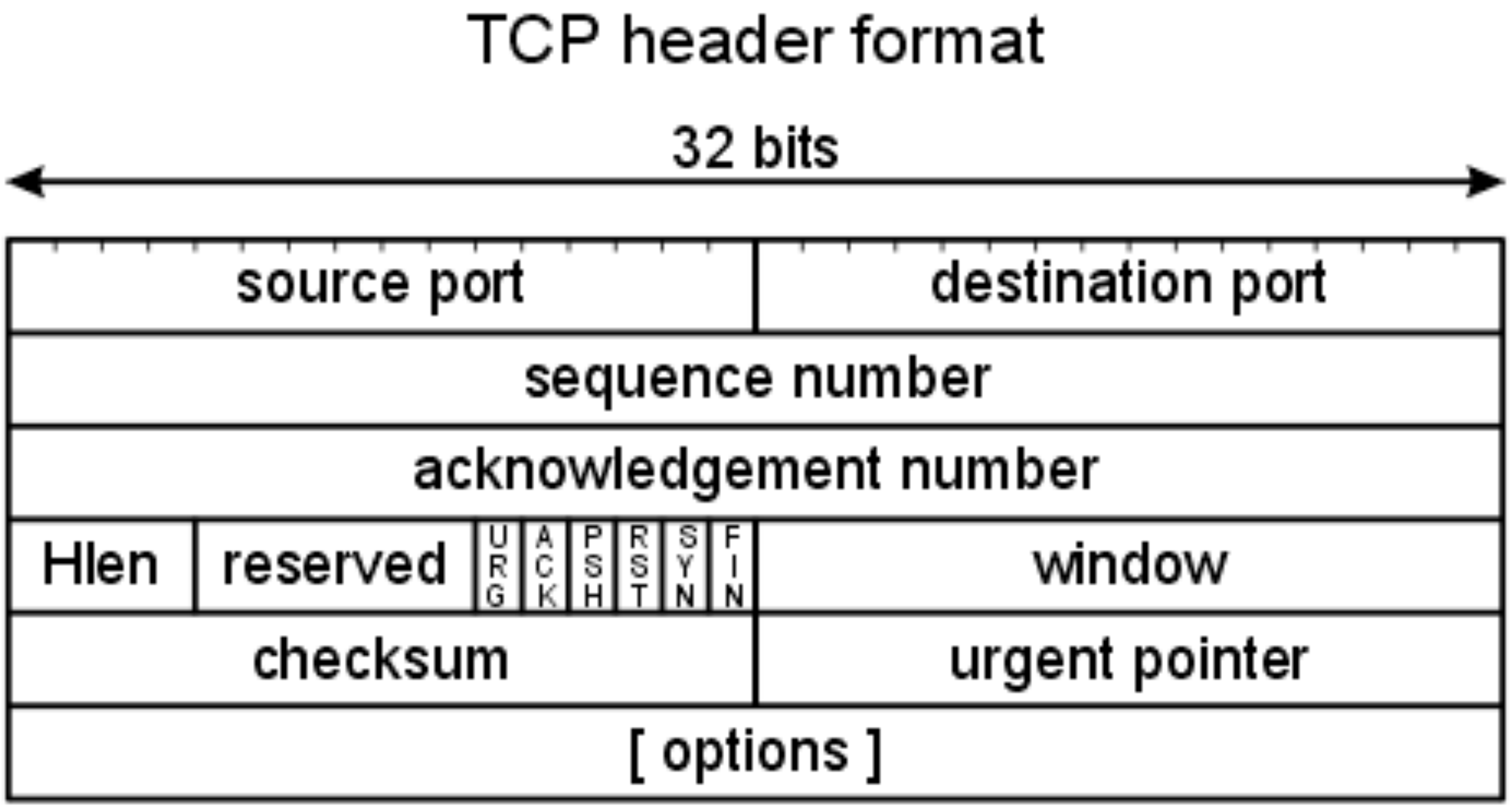
- The result of summing the entire IP header, including checksum, should be zero if there is no corruption.

Transport Layer: Transmission Control Protocol (TCP)

- TCP provides reliable, ordered, and error-checked delivery of a stream of bytes between applications running on hosts communicating by an IP network.

- TCP is the main way that the internet attempts quality control. Several tools are used in combination to achieve this:
  - Sequence numbers: each packet in a stream of data is numbered
  - Acknowledgements: receiver confirms receipt
  - Re-transmission requests: in case of missing data detected
  - Congestion control: avoid being too chatty (very important!)

TCP Headers



TCP header format

32 bits

<table>
<tr><td colspan="9">source port</td><td>destination port</td></tr>
<tr><td colspan="10">sequence number</td></tr>
<tr><td colspan="10">acknowledgement number</td></tr>
<tr><td>Hlen</td><td>reserved</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td></td><td>window</td></tr>
<tr><td colspan="9">checksum</td><td>urgent pointer</td></tr>
<tr><td colspan="10">[options]</td></tr>
</table>

High-level Layers: Applications

- These are the things we see every day:
  - HyperText Transfer Protocol: The web!
  - Secure shell: connecting to servers
  - File Transfer Protocol
  - SMTP: Email
  - DHCP: Obtaining an address on the network
  - DNS: Looking up computers with convenient names

What does this all look like in C?

- Linux and C provide us an interface to the Internet that is called Sockets, so that we can write end-point code such as chat clients, web browsers, web servers, video streams, phone apps, etc.

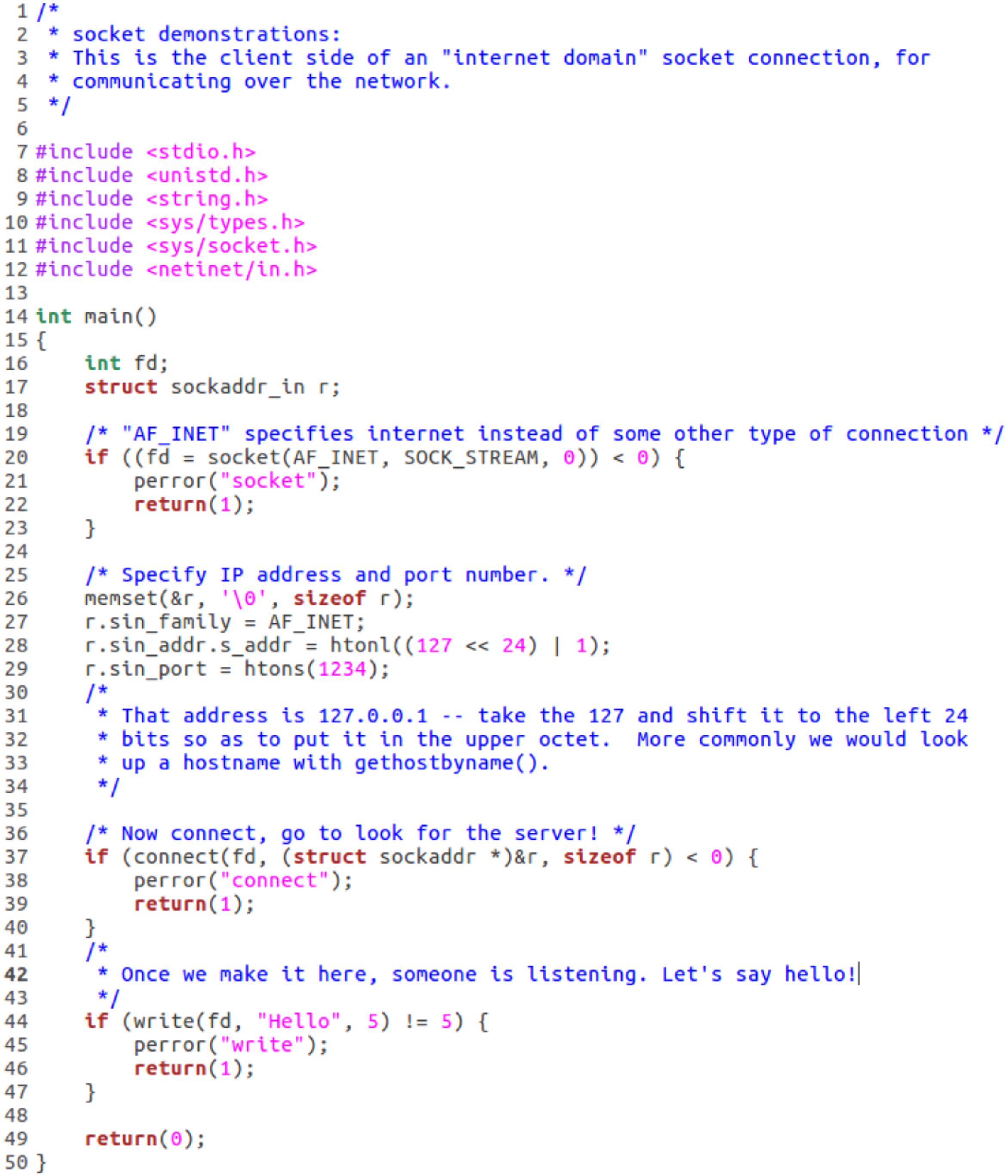
- We must be aware of what's happening in between our end-points because it sometimes affects us

- Most of the time, life is good and we treat the network just like it was a file!

UNIX Socket: Functions

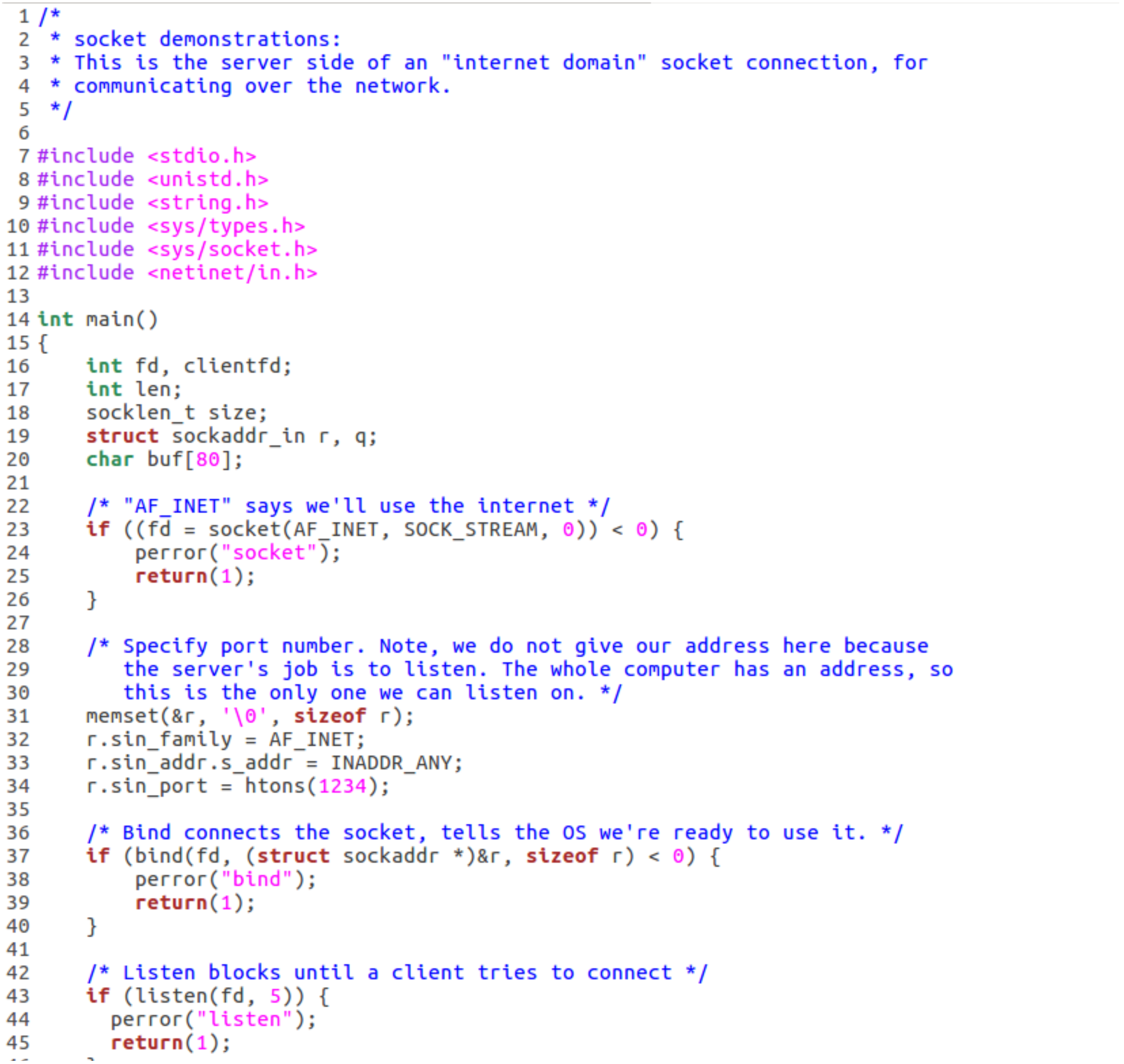
- socket() : Create a socket, specify it’s protocol type (AF_INET)
- bind() : Specify the coordinates: address and port number
- listen() : Wait for someone to respond, the OS does all the checking
- accept() : If someone is there, acknowledge and start the comms
- read() : Just like a file, put data into the socket
- write() : Just like a file, get data out of the socket

First Example: The client



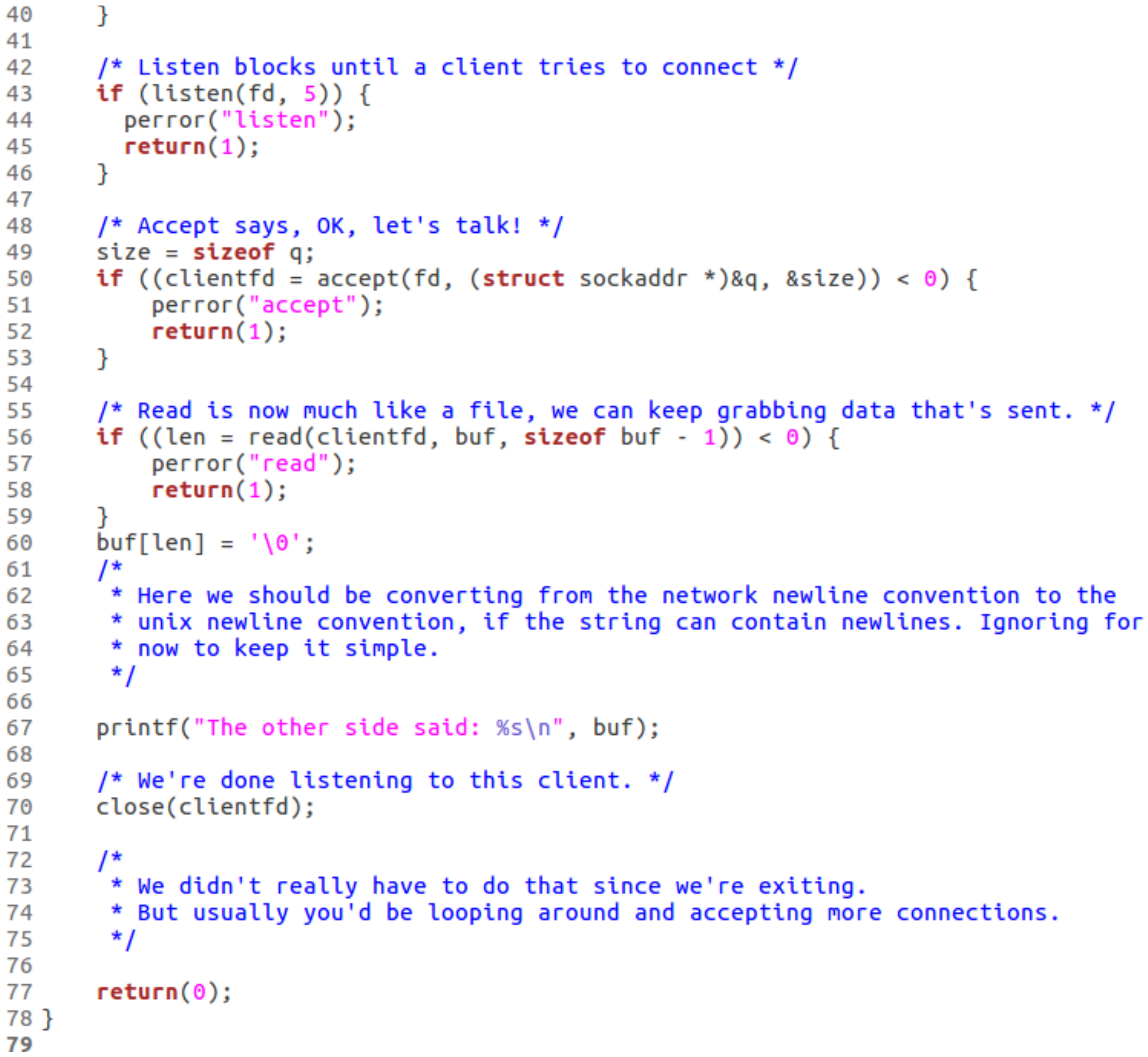
```c
/*
 * socket demonstrations:
 * This is the client side of an "internet domain" socket connection, for
 * communicating over the network.
 */

#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

int main()
{
    int fd;
    struct sockaddr_in r;

    /* "AF_INET" specifies internet instead of some other type of connection */
    if ((fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket");
        return(1);
    }

    /* Specify IP address and port number. */
    memset(&r, '\0', sizeof r);
    r.sin_family = AF_INET;
    r.sin_addr.s_addr = htonl((127 << 24) | 1);
    r.sin_port = htons(1234);
    /*
     * That address is 127.0.0.1 -- take the 127 and shift it to the left 24
     * bits so as to put it in the upper octet.  More commonly we would look
     * up a hostname with gethostbyname().
     */

    /* Now connect, go to look for the server! */
    if (connect(fd, (struct sockaddr *)&r, sizeof r) < 0) {
        perror("connect");
        return(1);
    }
    /*
     * Once we make it here, someone is listening. Let's say hello!
     */
    if (write(fd, "Hello", 5) != 5) {
        perror("write");
        return(1);
    }

    return(0);
}
```

First Example:
The server (part 1)



```c
/*
 * socket demonstrations:
 * This is the server side of an "internet domain" socket connection, for
 * communicating over the network.
 */

#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

int main()
{
    int fd, clientfd;
    int len;
    socklen_t size;
    struct sockaddr_in r, q;
    char buf[80];

    /* "AF_INET" says we'll use the internet */
    if ((fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket");
        return(1);
    }

    /* Specify port number. Note, we do not give our address here because
       the server's job is to listen. The whole computer has an address, so
       this is the only one we can listen on. */
    memset(&r, '\0', sizeof r);
    r.sin_family = AF_INET;
    r.sin_addr.s_addr = INADDR_ANY;
    r.sin_port = htons(1234);

    /* Bind connects the socket, tells the OS we're ready to use it. */
    if (bind(fd, (struct sockaddr *)&r, sizeof r) < 0) {
        perror("bind");
        return(1);
    }

    /* Listen blocks until a client tries to connect */
    if (listen(fd, 5)) {
      perror("listen");
      return(1);
```

First Example:
The server (part 2)



```
    }

    /* Listen blocks until a client tries to connect */
    if (listen(fd, 5)) {
      perror("listen");
      return(1);
    }

    /* Accept says, OK, let's talk! */
    size = sizeof q;
    if ((clientfd = accept(fd, (struct sockaddr *)&q, &size)) < 0) {
        perror("accept");
        return(1);
    }

    /* Read is now much like a file, we can keep grabbing data that's sent. */
    if ((len = read(clientfd, buf, sizeof buf - 1)) < 0) {
        perror("read");
        return(1);
    }
    buf[len] = '\0';
    /*
     * Here we should be converting from the network newline convention to the
     * unix newline convention, if the string can contain newlines. Ignoring for
     * now to keep it simple.
     */

    printf("The other side said: %s\n", buf);

    /* We're done listening to this client. */
    close(clientfd);

    /*
     * We didn't really have to do that since we're exiting.
     * But usually you'd be looping around and accepting more connections.
     */

    return(0);
}
```

Wrapping Up

- Many software systems today live entirely on the Internet. So far we saw a little of what a wild and crazy place this can be, but luckily there are mountains of good tools to help.

- In 206, we should know the basic concepts of how the Internet software system is organized and how we can start getting our code online.

- Try out the simple socket examples on My Courses for now. We'll look at more involved network coding in the next couple of lectures!