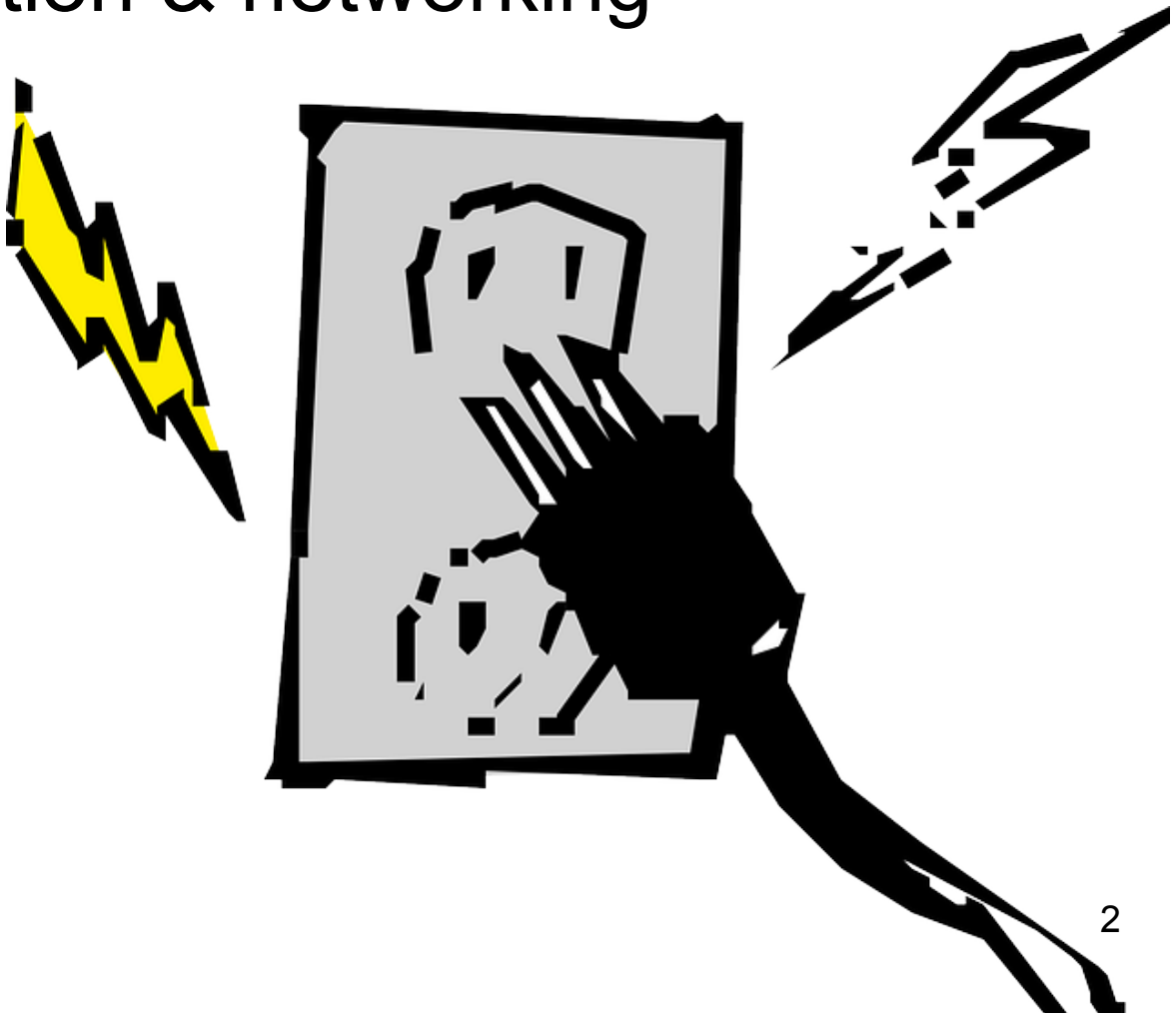
CSCB09: Software Tools and Systems Programming

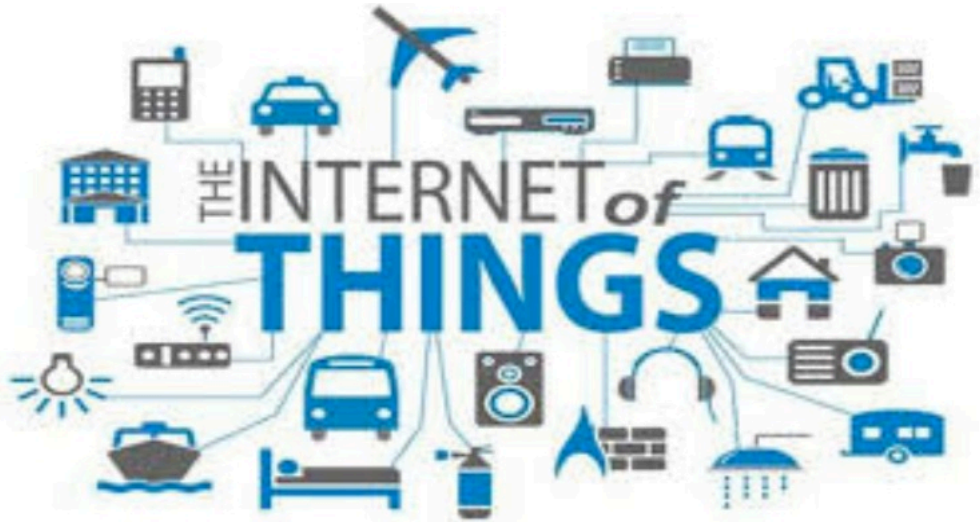
Bianca Schroeder
bianca@cs.toronto.edu
IC 460

Today

- Communication & networking
- Sockets

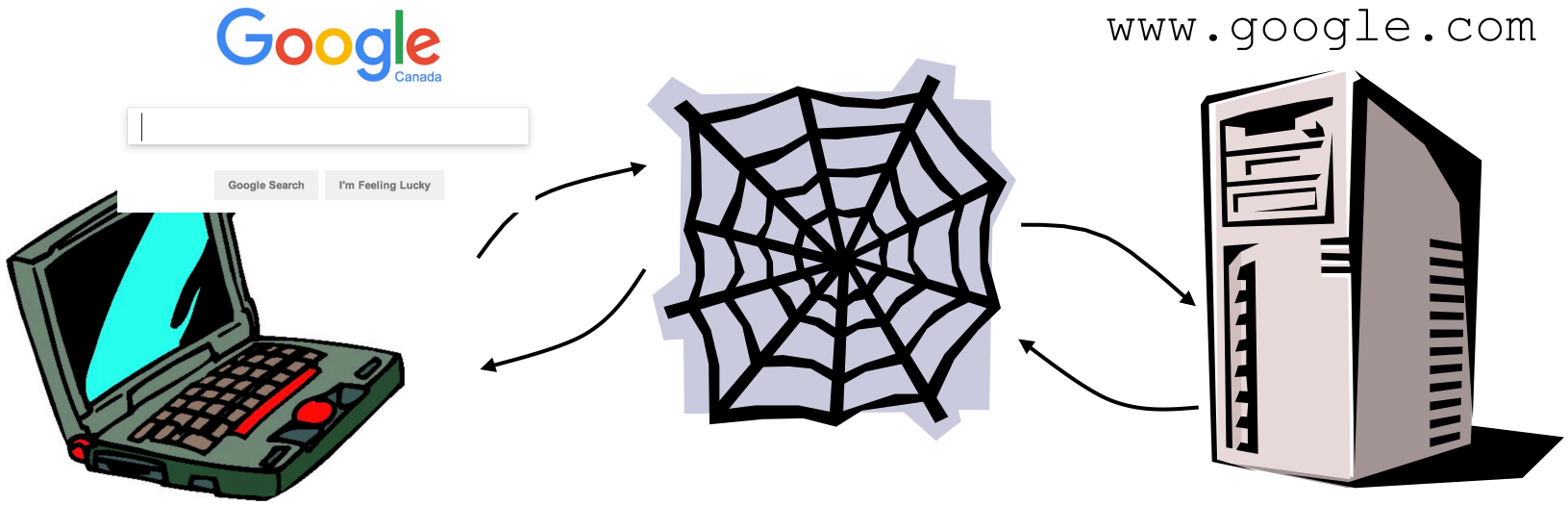


Why?



Everything is networked now!

Simple Web Request



Web browser process
(e.g. Mozilla, Firefox, …)

Web server process
(e.g. Apache)

- Two processes communicating: the web browser and the web server
- How exactly does this work?

Client Server Model

Server: Always available at a known location

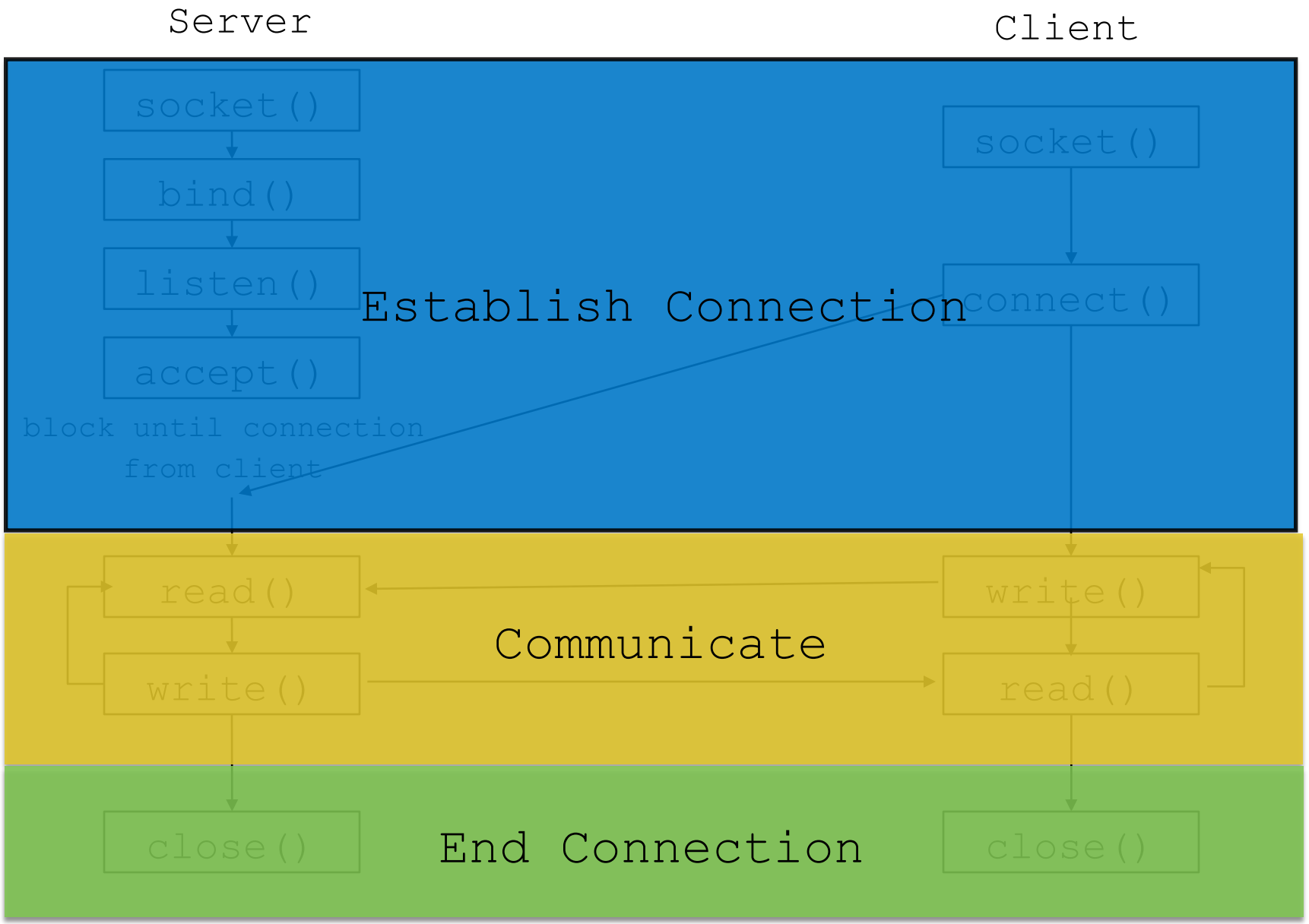
Clients: Show up and initiate the connection

Some basic questions

- How do the server and browser talk to each other?
  - Pipes? Only for processes on same machine.
  - Solution: Sockets

Sockets

- One form of communication between processes.
- Similar to pipes, except sockets can be used between processes on different machines and they are bi-directional.
  - Use file descriptors to refer to sockets.
  - Reading and writing similar to pipes
- The client and the server independently each create a socket using `socket()` system call


Server
Client
socket()
bind()
listen()
accept()
block until connection from client
socket()
connect()
Establish Connection
read()
write()
write()
read()
Communicate
close()
close()
End Connection

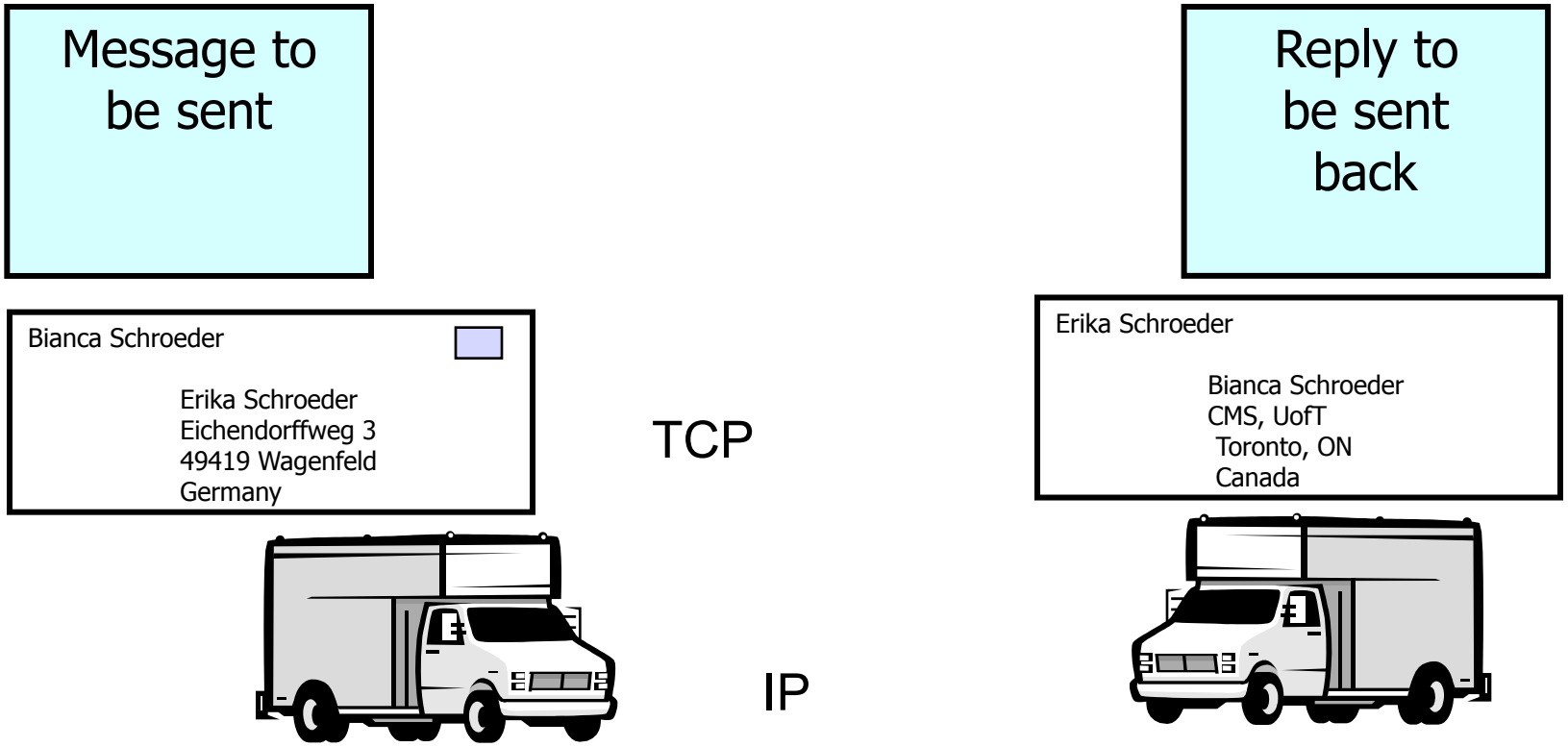
The socket() system call

```
int socket(int family, int type,
           int protocol);
```

- Returns a socket descriptor (similar to file descriptor)
- Arguments specify type of socket, including protocols to be used when processes exchange message on the socket

- Need to know a bit more about internet protocols …

Internet protocols



- The IP protocol enables routing of messages to the destination (based on correct address)
- The TCP protocol enables reliable, in-order delivery of messages. E.g. will resend lost message.

The socket() system call

```
int socket(int family, int type,
           int protocol);
```

- family specifies protocol family / domain:
  - `AF_INET` – IPv4
  - `AF_LOCAL` – Unix domain
- type
  - `SOCK_STREAM` (for reliable, in-order connection-based communication)
  - `SOCK_DGRAM`
  - `SOCK_RAW`
- protocol
  - set to 0 except for RAW sockets
- returns a socket descriptor

Let’s set up a socket
(Task 1 of today’s worksheet) …

How do we specify the server?

`socket()`

Next: Client needs to specify which server to connect to

`socket()`

www.google.com

IP address: 172.217.1.46

Port: 80

- Every computer on the Internet has an Internet address.
- Called an IP address (Internet Protocol)
- An IPv4 address is 4 numbers separated by dots.
- In addition client needs to specify, which process at server it wants to talk to: port number

Addresses and Ports

- A socket pair is the two endpoints of the connection.
- An endpoint is identified by an IP address and a port.
- IPv4 addresses are 4 8-bit numbers:
  - 142.1.96.164 = mathlab.utsc.utoronto.ca
  - 142.1.96.30 = www.utsc.utoronto.ca
- Ports
  - because multiple processes can communicate with a single machine we need another identifier.

More on Ports

- Well-known ports: 0-1023
  - 80 = http
  - 22 = ssh
  - 21 = ftp
  - 25 = smtp (mail)
- Registered ports: 1024-49151
  - 2709 = supermon
  - 26000 = quake
  - 3724 = world of warcraft
- Dynamic (private) ports: 49152-65535

The connect() system call

```
int connect(int sockfd,
      const struct sockaddr *servaddr,
      socklen_t addrlen);
```

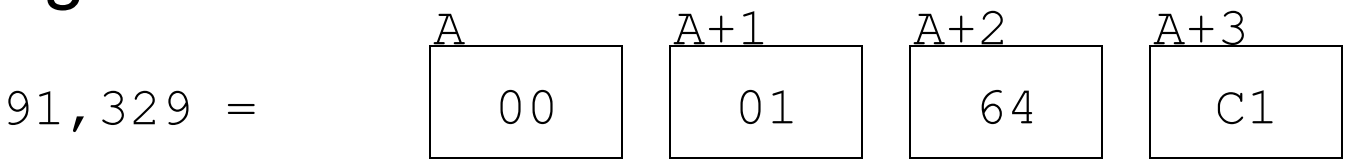
- Returns 0 on success.
- `servaddr` specifies which server to connect to.

```
struct sockaddr_in {
   short          sin_family; /*AF_INET */
   u_short        sin_port;
   struct in_addr sin_addr;
   char           sin_zero[8];
};
```

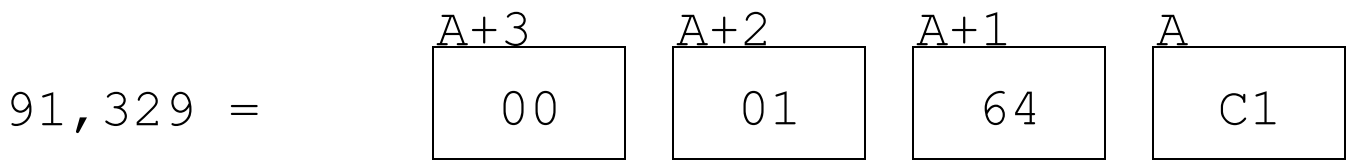
Let’s see whether we can
connect a socket
(Task 2+3 of today’s worksheet)
…

Byte order

- Big-endian



- Little-endian



- Different hardware architectures have different endian-ness
- Intel is little-endian, and Sparc is big-endian

Network byte order

- To communicate between machines with unknown or different “endian-ness” we convert numbers to network byte order (big-endian) before we send them.
- There are functions provided to do this:
  - `unsigned long htonl(unsigned long)`
  - `unsigned short htons(unsigned short)`
  - `unsigned long ntohl(unsigned long)`
  - `unsigned short ntohs(unsigned short)`

Now your socket should be ready
for reading and writing
(Task 4+5 of today's worksheet)
…