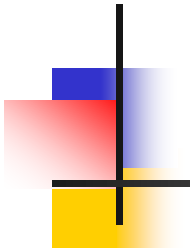
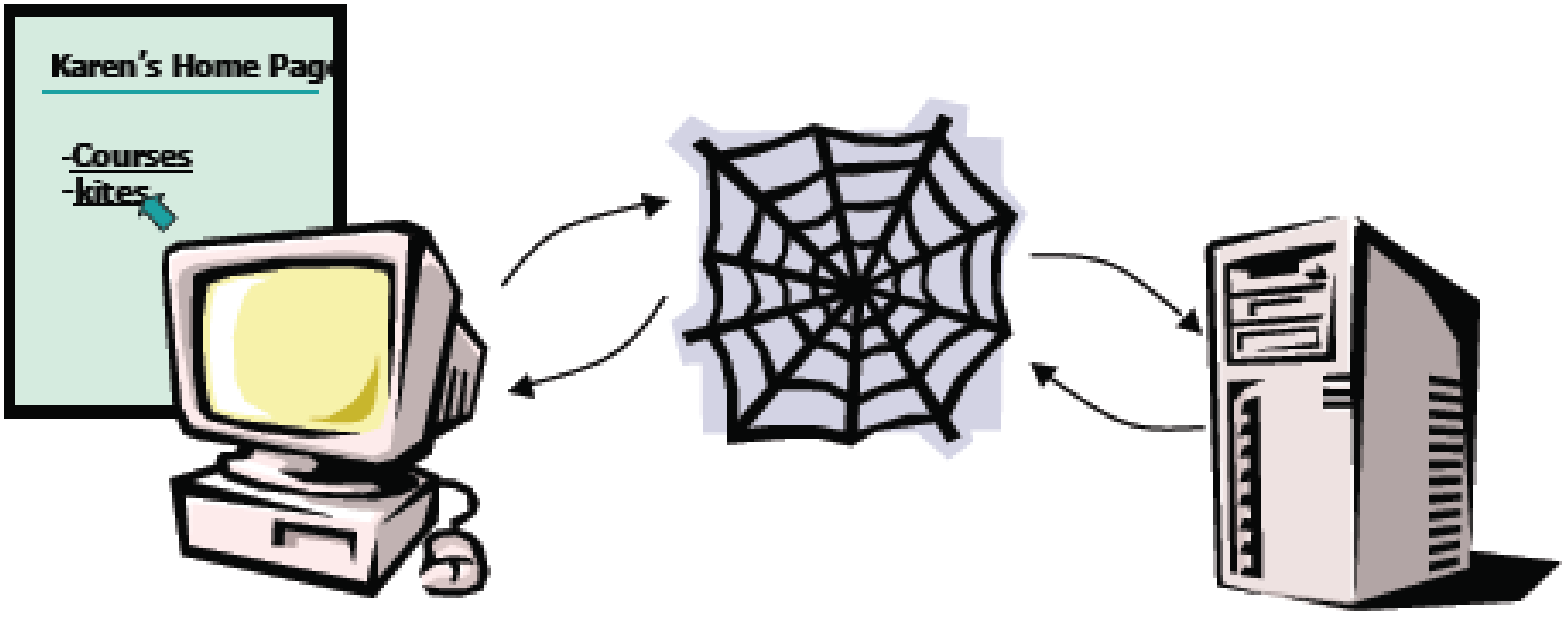


# System Programming

## Sockets

# Simple Web Request

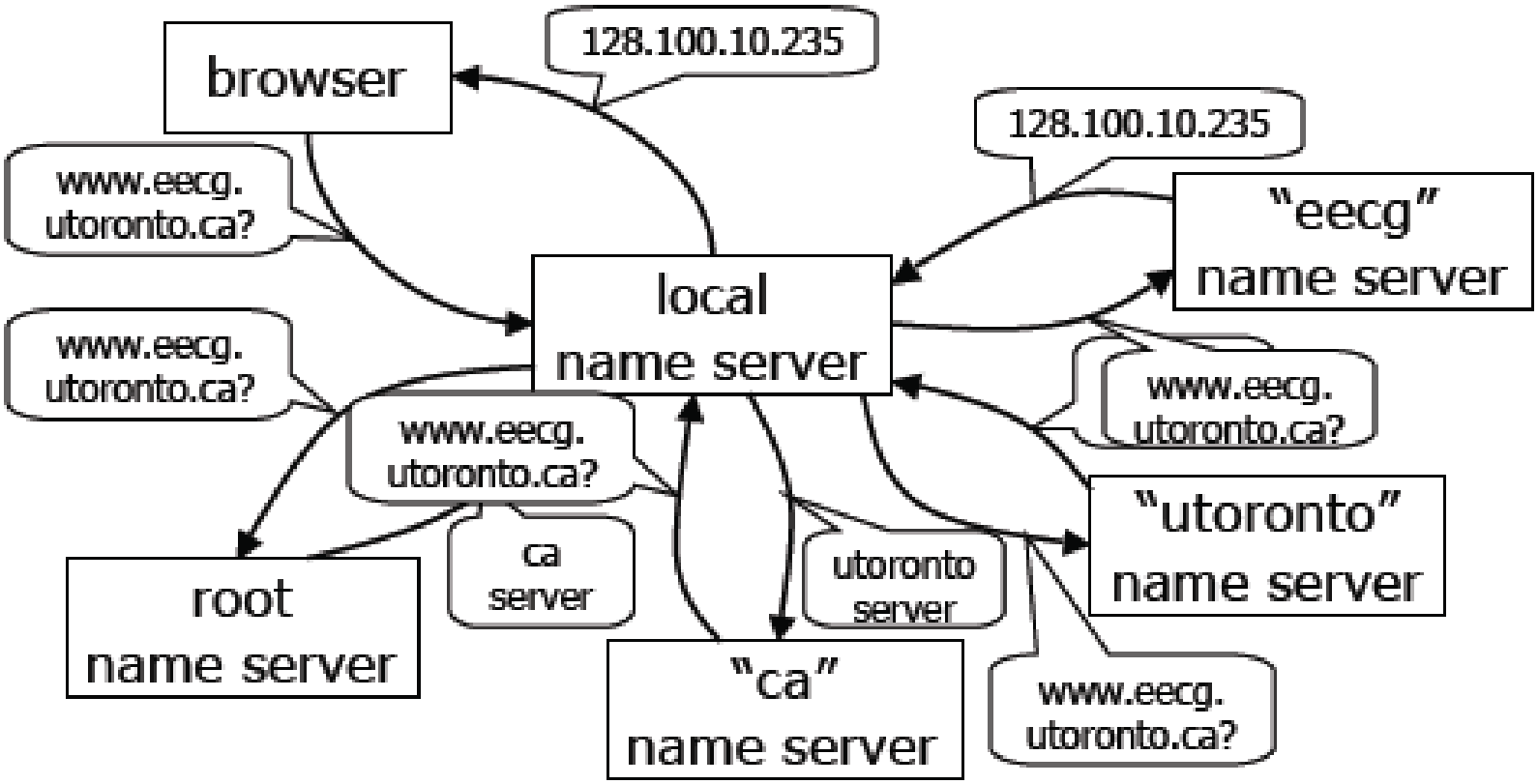


# How do we find the server?

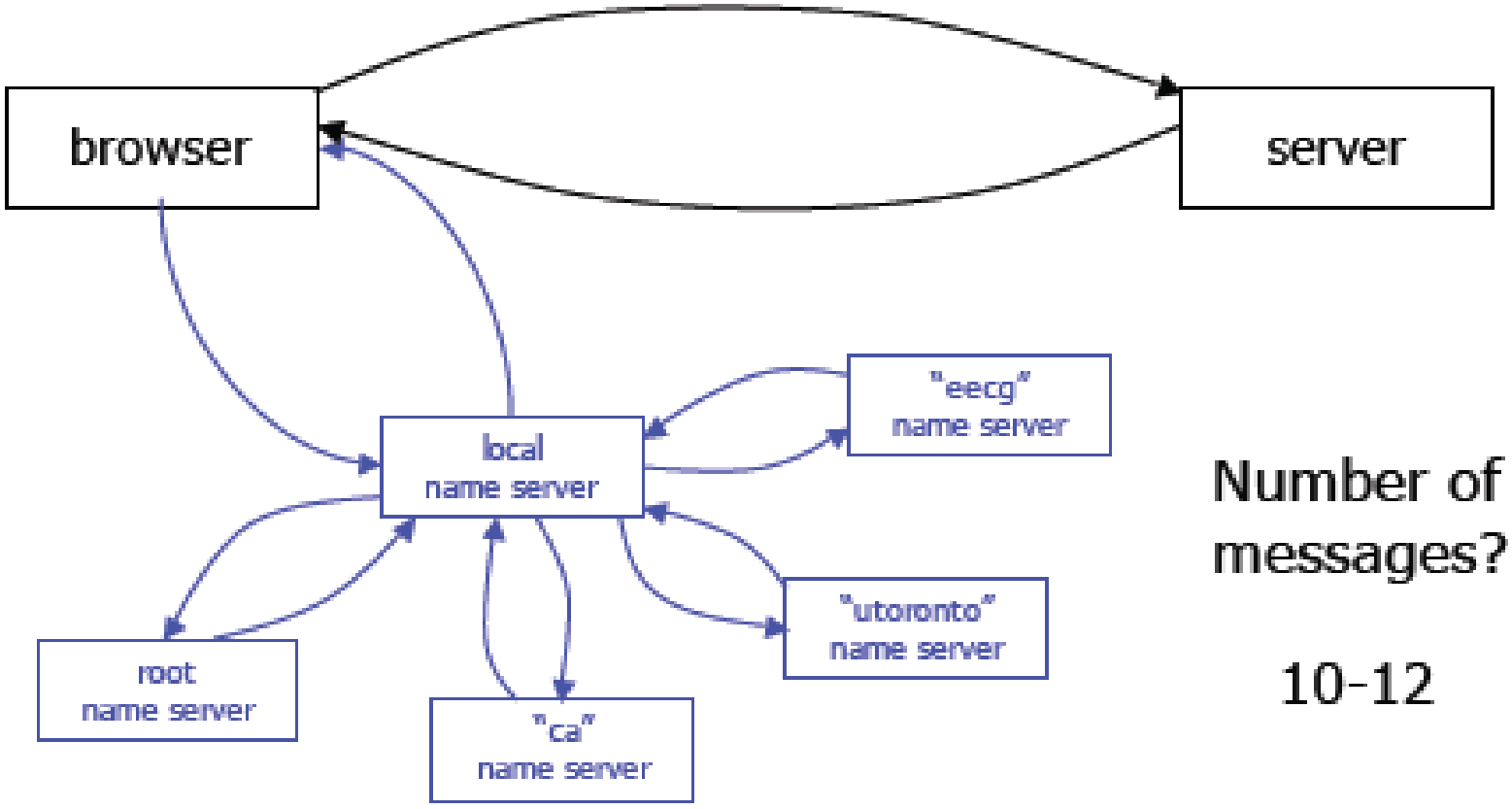
- Every computer on the Internet has an Internet address.
- Called an IP address (Internet Protocol)
- An IP address is 4 8-bit numbers separated by dots.

  www.ktu.edu.tr = 193.140.168.225

# Domain Name Servers



# This is getting complicated!




Number of messages?

10-12

# Protocols

Invoice:
Customer: Karen Reid
Order No: 5379

| Qty: | | Unit Price | Total |
|---|---|---|---|
| 1 | Athalon | 219.00 | 219.00 |
| 2 | 128 MB | 149.95 | 299.90 |
| | | Subtotal | 518.90 |
| | | Tax | 77.84 |
| | | TOTAL | 596.74 |

Karen Reid Feb 18, 2001

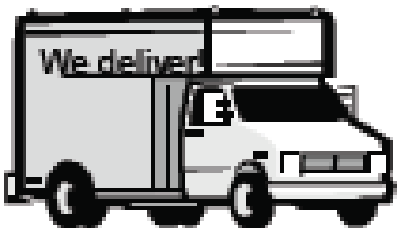
Payable to: CPUS are us $596.74
Five hundred ninety six 74/100

CPUS are us

Karen Reid
Dept. of Computer Science
University of Toronto

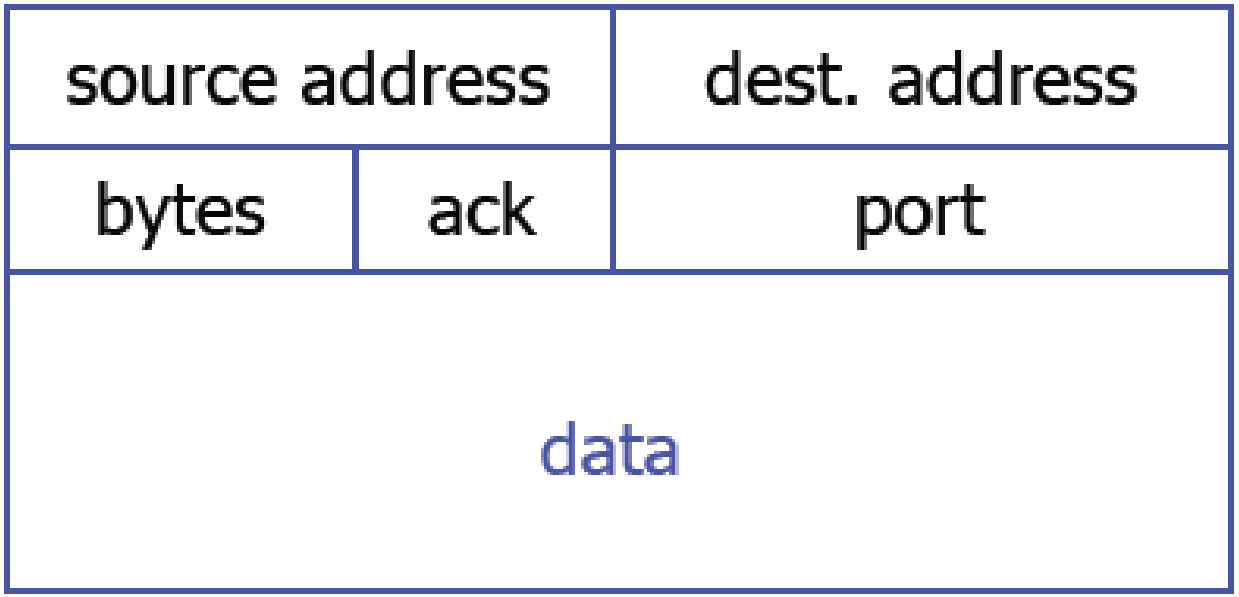
Karen Reid

CPUS are us
0 College Street
Toronto Ontario M5S 3G4

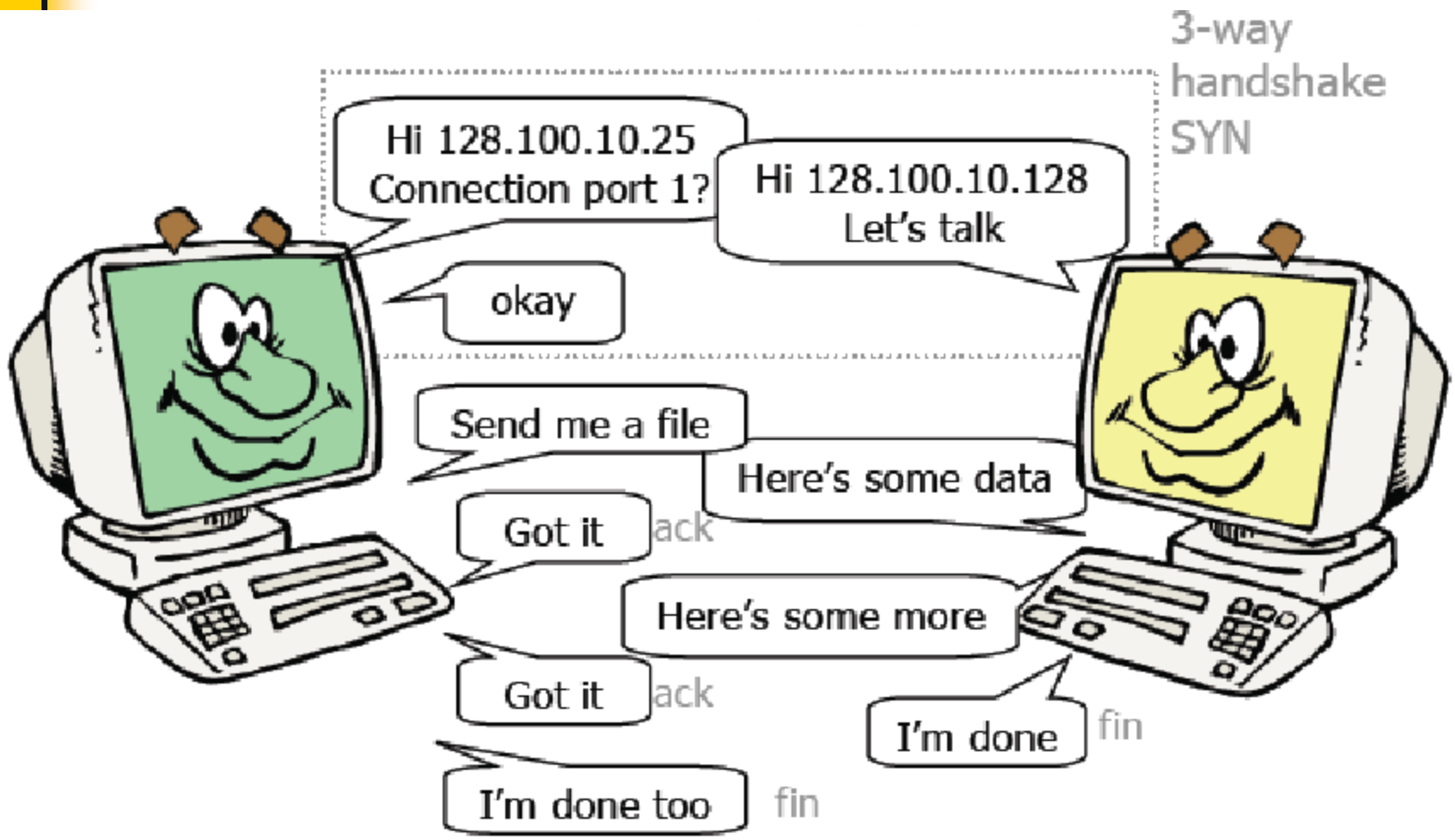



# TCP/IP

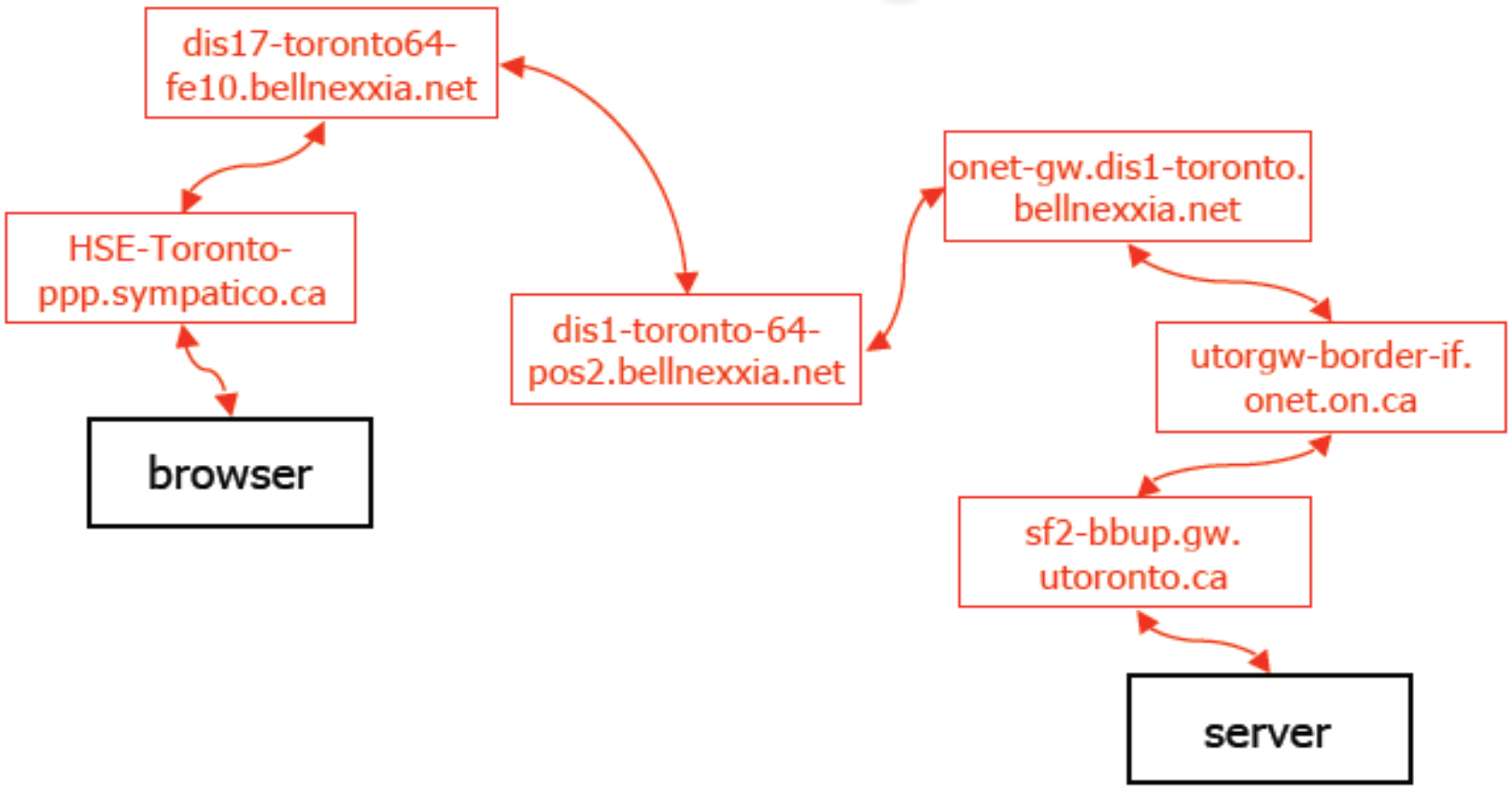
- Transmission Control Protocol.
- Tells us how to package up the data.



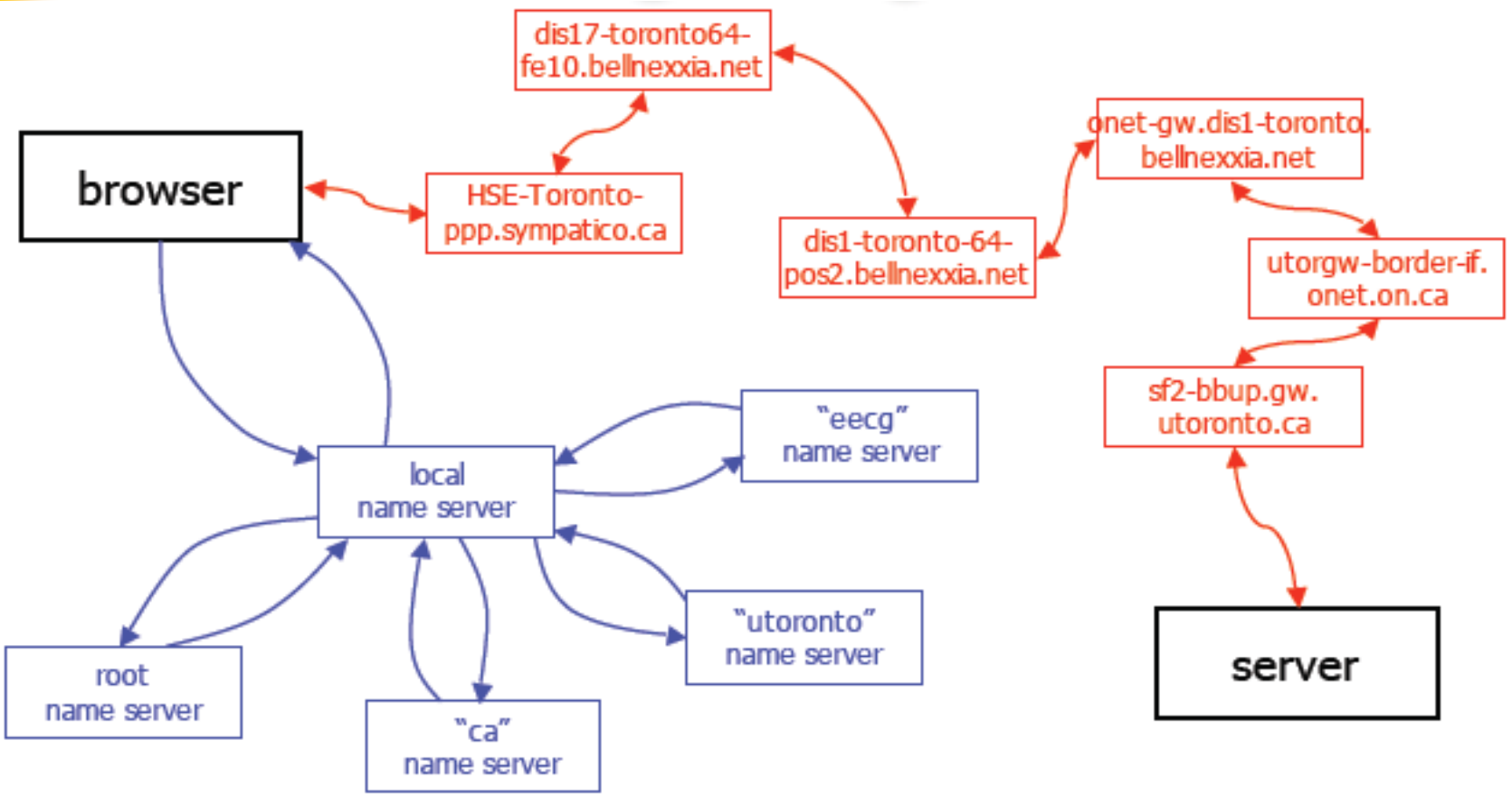
# TCP Connection



# Routing


dis17-toronto64-
fe10.bellnexxia.net
HSE-Toronto-
ppp.sympatico.ca
browser
dis1-toronto-64-
pos2.bellnexxia.net
onet-gw.dis1-toronto.
bellnexxia.net
utorgw-border-if.
onet.on.ca
sf2-bbup.gw.
utoronto.ca
server

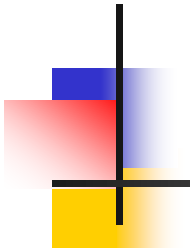
# Putting it together



# How many messages?

- It depends on the size of the web page we retrieve.
- If the web page is 75 Kbytes (small!) it will be broken up into 103 IP packets.
- Remember DNS took 10 messages

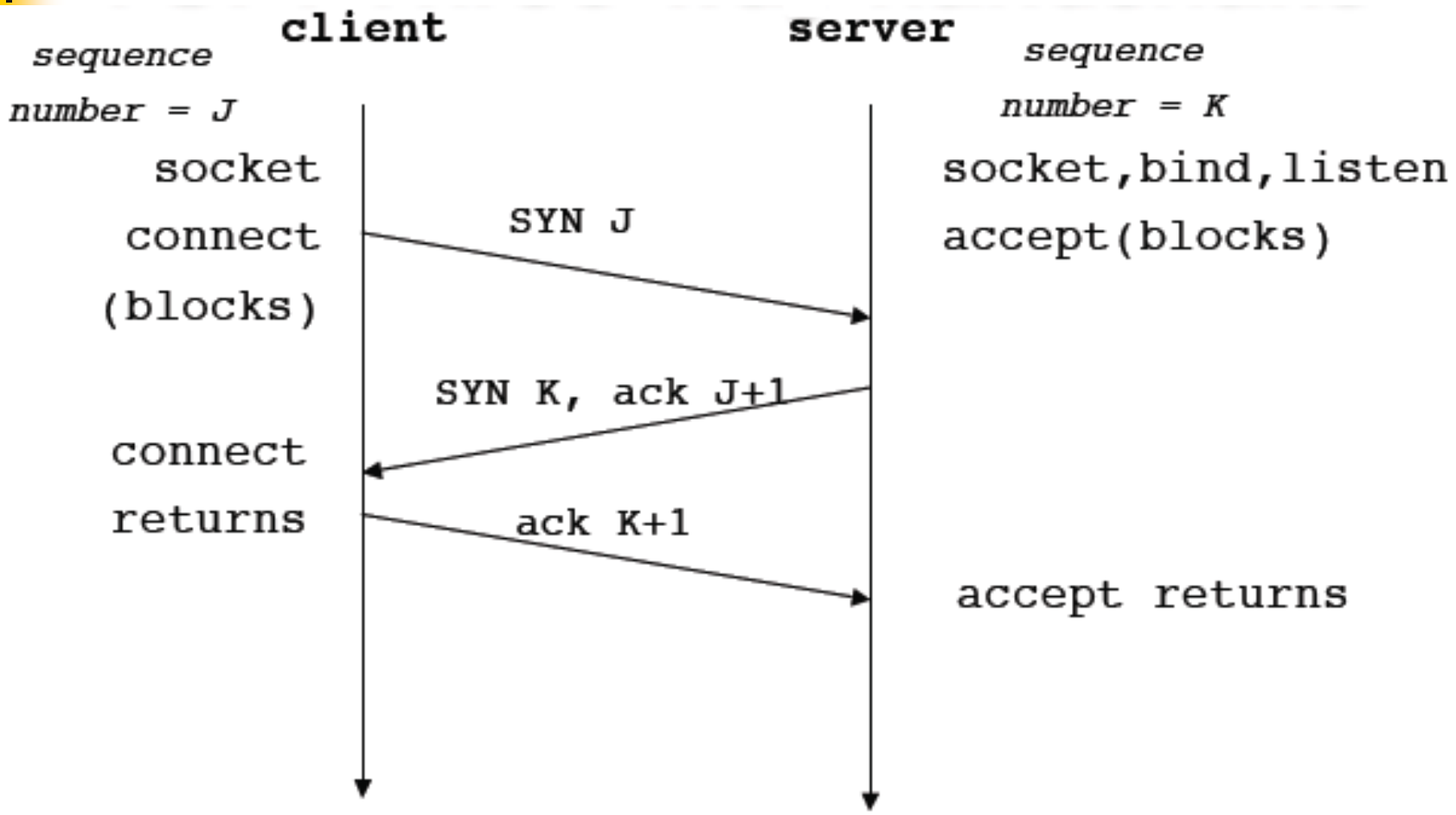
10 + 103 x 7 hops = 731 messages!

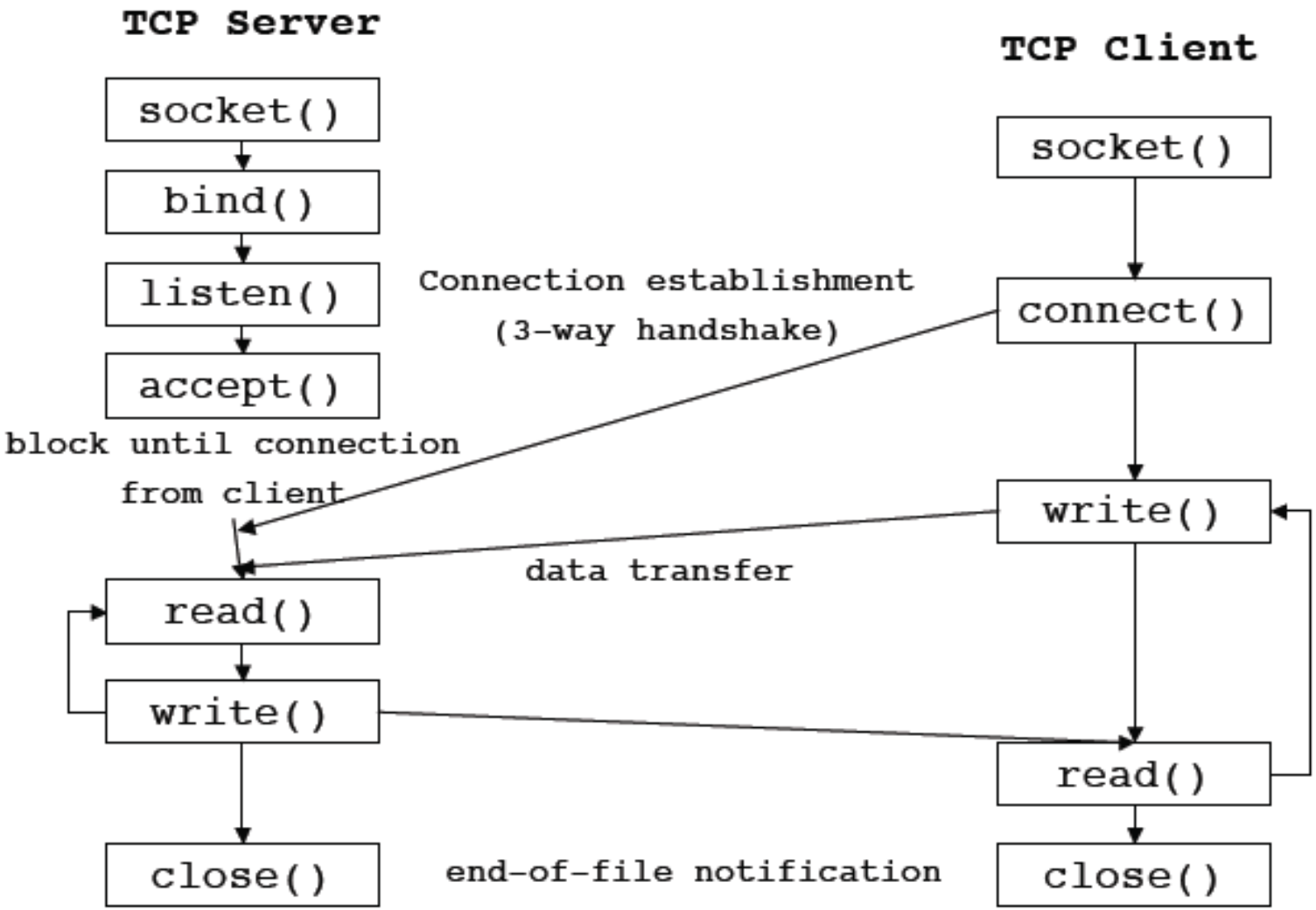


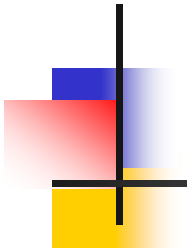
# Sockets

- One form of communication between processes.
- Similar to pipes, except sockets can be used between processes on different machines.
- Use file descriptors to refer to sockets.
- Built on top of TCP layer

# TCP: Three-way handshake

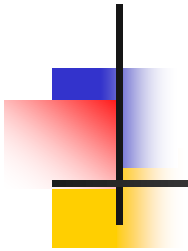



TCP Server
TCP Client
socket()
bind()
listen()
accept()
block until connection
from client
socket()
connect()
write()
Connection establishment
(3-way handshake)
data transfer
read()
write()
read()
close()
end-of-file notification
close()



# Connection-Oriented

- Server
  - Create a socket: `socket()`
  - Assign a name to a socket: `bind()`
  - Establish a queue for connections: `listen()`
  - Get a connection from the queue: `accept()`
- Client
  - Create a socket: `socket()`
  - Initiate a connection: `connect()`



# Socket Types

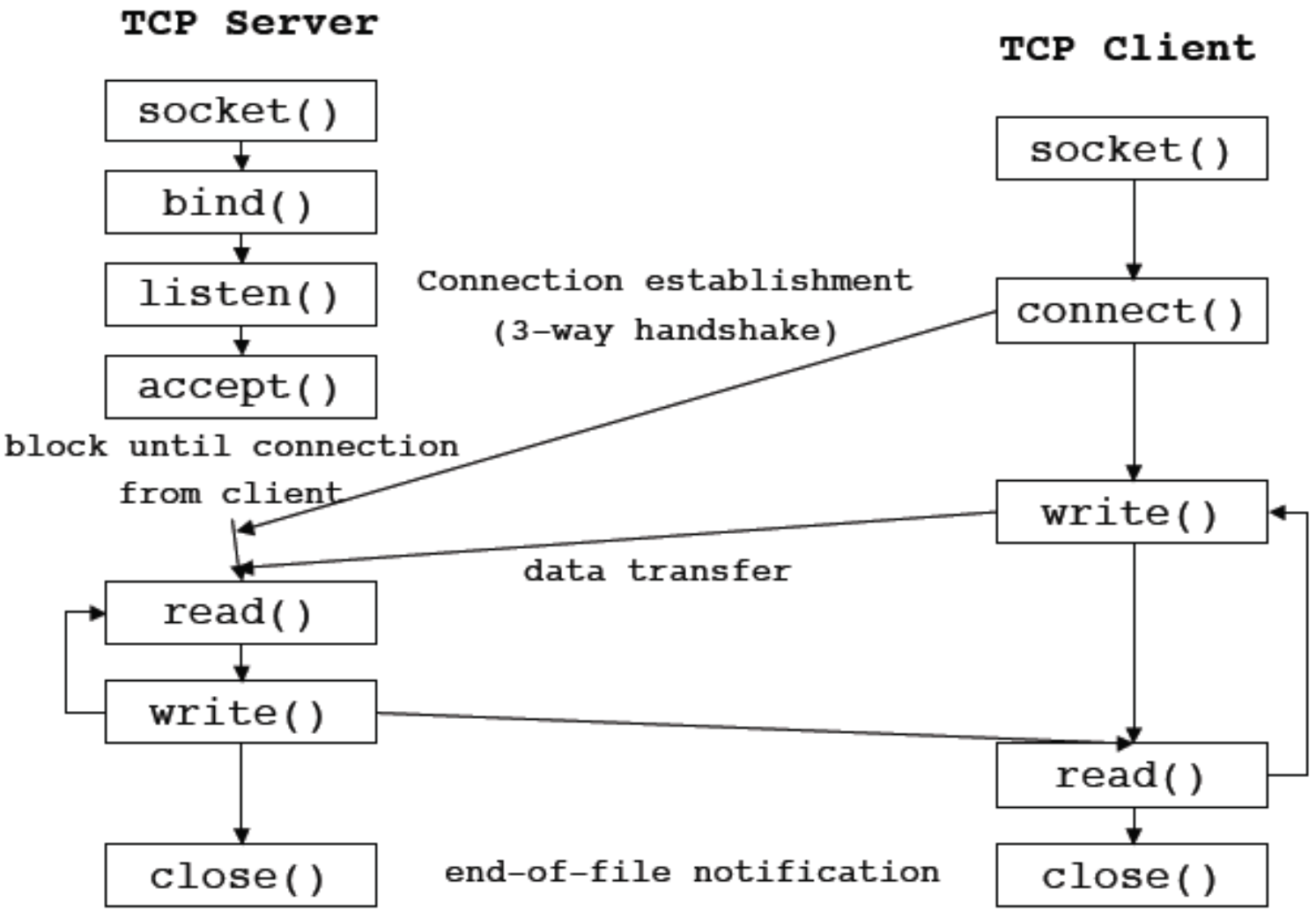
- Two main categories of sockets
  - UNIX domain: both processes on the same machine
  - INET domain: processes on different machines
- Three main types of sockets:
  - SOCK_STREAM: the one we will use
  - SOCK_DGRAM: for connectionless sockets
  - SOCK_RAW

# 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:
  - 128.100.31.156 = penguin
  - 128.100.31.4 = eddie
- Ports
  - because multiple processes can communicate with a single machine we need another identifier.

# More on Ports

- Well-known ports: 0-1023
  - 80 = web
  - 22 = ssh
  - 23 = telnet
  - 21 = ftp
  - 25 = smtp (mail)
  - 194 = irc
- Registered ports: 1024-49151
  - 2709 = supermon
  - 26000 = quake
- Dynamic (private) ports: 49152-65535
  - You should pick ports in this range to avoid overlap


TCP Server
TCP Client
socket()
bind()
listen()
accept()
block until connection
from client
Connection establishment
(3-way handshake)
socket()
connect()
write()
data transfer
read()
write()
read()
close()
end-of-file notification
close()

# Server side

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

- family specifies protocol family:
  - PF_INET – IPv4
  - PF_LOCAL – Unix domain
- type
  - SOCK_STREAM, SOCK_DGRAM, SOCK_RAW
- protocol
  - set to 0 except for RAW sockets
- returns a socket descriptor

# bind to a name

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

- `sockfd` – returned by socket

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

- `sin_addr` can be set to INADDR_ANY to communicate with any host

# Set up queue in kernel

```
int listen(int sockfd, int backlog)
```

- after calling `listen`, a socket is ready to accept connections
- prepares a queue in the kernel where partially completed connections wait to be accepted.
- `backlog` is the maximum number of partially completed connections that the kernel should queue.

# Complete the connection

```
int accept(int sockfd,
           struct sockaddr *cliaddr,
           socklen_t *addrlen);
```

- blocks waiting for a connection (from the queue)
- returns a new descriptor which refers to the TCP connection with the client
  - `sockfd` is the listening socket
  - `cliaddr` is the address of the client
- reads and writes on the connection will use the socket returned by accept

# Client side

- `socket()` – same as server, to say “how” we are going to talk

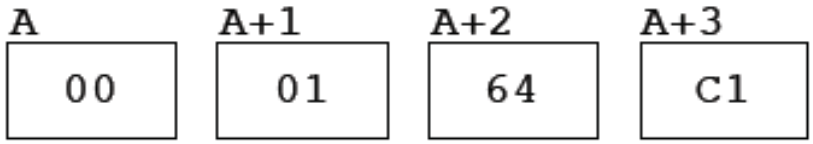
```
int connect(int sockfd,
    const struct sockaddr *servaddr,
    socklen_t addrlen);
```

- the kernel will choose a dynamic port and source IP address.
- returns 0 on success and -1 on failure setting errno.
- initiates the three-way handshake.

# Byte order

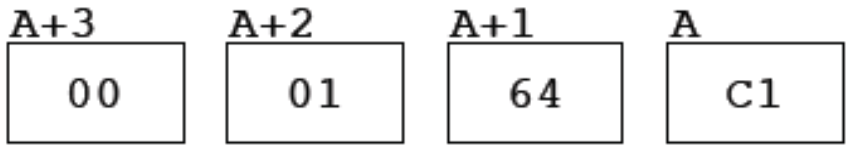
- Big-endian

91,329




- Little-endian

91,329




- 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 (bigendian) 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)`