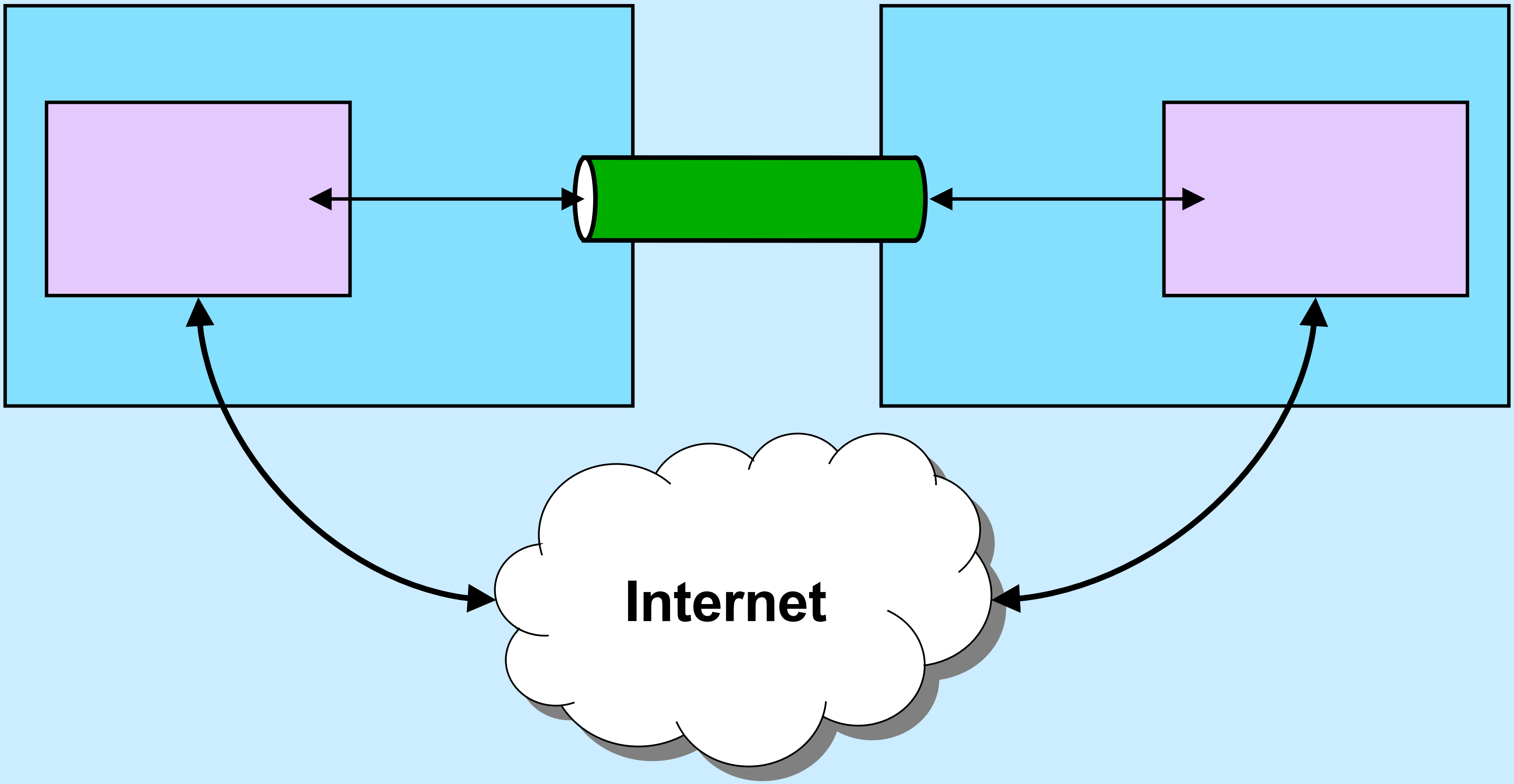
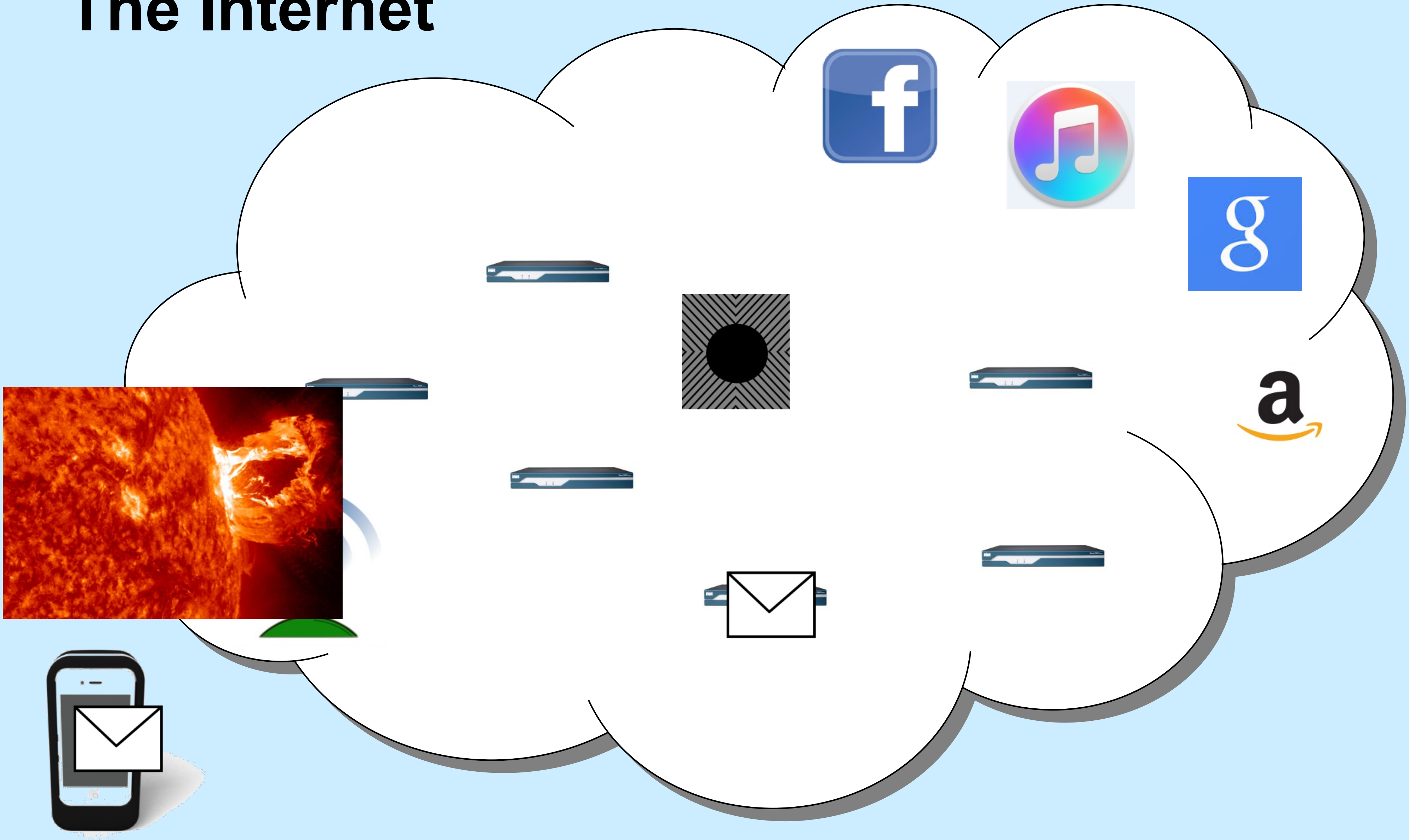
CS 33

Network Programming

Communicating Over the Internet



The Internet



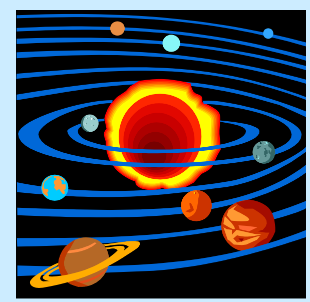
Names and Addresses

- **cslab1c.cs.brown.edu**
  – the name of a computer on the internet
  – mapped to an internet address
- **www.nyt.com**
  – the name of a service on the internet
  – mapped to a number of internet addresses

- **How are names mapped to addresses?**
  – domain name service (DNS): a distributed database
- **How are the machines corresponding to internet addresses found?**
  – with the aid of various routing protocols

Internet Addresses

- IP (internet protocol) address
  - one per network interface
  - 32 bits (IPv4)
    - » 5527 per acre of RI
    - » 25 per acre of Texas
  - 128 bits (IPv6)
    - » 1.6 billion per cubic mile of a sphere whose radius is the mean distance from the Sun to the (former) planet Pluto
- Port number
  - one per application instance per machine
  - 16 bits
    - » port numbers less than 1024 are reserved for privileged applications



Notation

- **Addresses (assume IPv4: 32-bit addresses)**
  - written using dot notation
    - » 128.48.37.1
      - dots separate bytes
  - address plus port (1426):
    - » 128.48.37.1:1426

Reliability

- Two possibilities
  - don't worry about it
    - » just send it
      - if it arrives at its destination, that's good!
        - no verification
  - worry about it
    - » keep track of what's been successfully communicated
      - receiver "acks"
    - » retransmit until
      - data is received

      or
      - it appears that "the network is down"

Reliability vs. Unreliability

- Reliable communication
  - good for
    - » email
    - » texting
    - » distributed file systems
    - » web pages
  - bad for
    - » streaming audio
    - » streaming video

a little noise is better than a long pause

The Data Abstraction

- Byte stream
  - sequence of bytes
    - as in pipes
  - any notion of a larger data aggregate is the responsibility of the programmer
- Discrete records
  - sequence of variable-size “records”
  - boundaries between records maintained
  - receiver receives discrete records, as sent by sender

What’s Supported

- **Stream**
  - byte-stream data abstraction
  - reliable transmission
- **Datagram**
  - discrete-record data abstraction
  - unreliable transmission

Quiz 1

The following code is used to transmit data over a reliable byte-stream communication channel. Assume sizeof(data) is large.

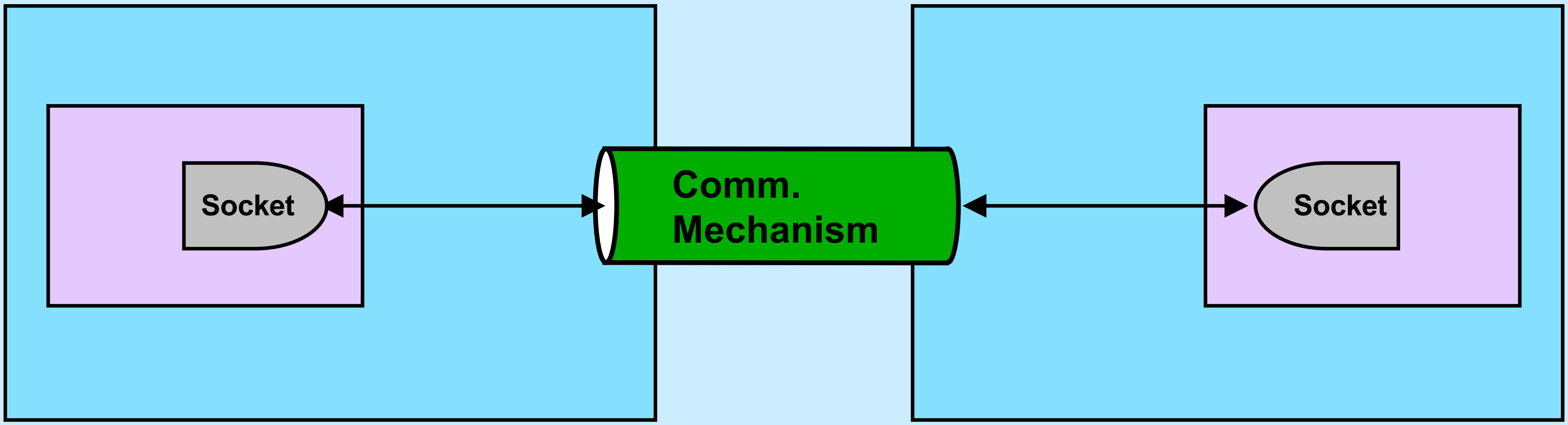
```
// sender
record_t data=getData();
write(fd, &data,
  sizeof(data));
```

```
// receiver
read(fd, &data,
  sizeof(data));
useData(data);
```

Does it work?

- a) always
- b) always, assuming no network problems
- c) sometimes
- d) never

Sockets



- You tell the system what you want by setting up the socket
- The system deals with all the other details

Socket Parameters

- Styles of communication:
  - stream: reliable, two-way byte streams
  - datagram: unreliable, two-way record-oriented
  - and others
- Communication domains
  - UNIX
    - endpoints (sockets) named with file-system pathnames
    - supports stream and datagram
    - trivial protocols: strictly for intra-machine use
  - Internet
    - endpoints named with IP addresses
    - supports stream and datagram
  - others
- Protocols
  - the means for communicating data
  - e.g., TCP/IP, UDP/IP

Setting Things Up

- Socket (communication endpoint) is given a name
  - *bind* system call
- Datagram communication
  - use *sendto* system call to send data to named recipient
  - use *recvfrom* system call to receive data and name of sender
- Stream communication
  - client connects to server
    - » server uses *listen* and *accept* system calls to receive connections
    - » client uses *connect* system call to make connections
  - data transmitted using *send* or *write* system calls
  - data received using *recv* or *read* system calls

Datagrams in the Internet Domain (1)

- **Steps**

  1) create socket

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

fd = socket(AF_INET, SOCK_DGRAM, 0);
```

Datagrams in the Internet Domain (2)

```
struct sockaddr_in {
  sa_family_t     sin_family; /* address family: AF_INET */
  in_port_t       sin_port;   /* port in network byte order */
  struct in_addr  sin_addr;   /* internet address */
};

struct in_addr {
  uint32_t        s_addr;     /* address in network byte order */
};


struct sock_addr_in my_addr;

my_addr.sin_family = AF_INET;
inet_pton(AF_INET, "10.116.72.109", &my_addr.sin_addr.s_addr);
my_addr.sin_port = htons(3333);
```

Datagrams in the Internet Domain (3)

2) **bind name to socket**

```
if (bind(fd, (struct sockaddr *)&my_addr,
        sizeof(my_addr)) < 0) {
    perror("bind");
    exit(1);
}
```

Datagrams in the Internet Domain (4)

3) **receive data**

```
ssize_t recvfrom(int fd, void *buf,
    ssize_t len,
    int flags, struct sockaddr *from,
    socklen_t *from_len);

struct sockaddr_in from_addr;
int from_len = sizeof(from_addr);

recvfrom(fd, buf, sizeof(buf), 0,
    (struct sockaddr *)&from_addr,
    &from_len);
```

Datagrams in the Internet Domain (5)

4) **send data**

```
ssize_t sendto(int fd, const void *buf,
    ssize_t len, int flags,
    const struct sockaddr *to,
    socklen_t to_len);
```

```
sendto(fd, buf, sizeof(buf), 0,
    (struct sockaddr *)&from_addr,
    from_len);
```

Quiz 2

Suppose a process on one machine sends a datagram to a process on another machine. The sender uses *sendto* and the receiver uses *recvfrom*. There's a momentary problem with the network and the datagram doesn't make it to the receiving process. Its call to *recvfrom*

a) doesn't return
b) returns –1 (indicating an error)
c) returns 0
d) returns some other value

Using DNS

- **Translate names to addresses using *getaddrinfo***
  - **looks up name in DNS, gets list of possible addresses**

getaddrinfo()

- ```
  int getaddrinfo(
        const char *node,
        const char *service,
        const struct addrinfo *hints,
        struct addrinfo **res);
  ```

```
struct addrinfo {
    int                 ai_flags;
    int                 ai_family;
    int                 ai_socktype;
    int                 ai_protocol;
    socklen_t          *ai_addrlen;
    struct sockaddr    *ai_addr;
    char               *ai_canonname;
    struct addrinfo    *ai_next;
};
```
```

Using *getaddrinfo* (1)

```
struct addrinfo hints, **res, *rp;
// zero out hints
memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_INET;
  // want IPv4
hints.ai_socktype = SOCK_DGRAM;
  // want datagram communication

getaddrinfo("cslab1a.cs.brown.edu", "3333",
    &hints, &res);
```


Using *getaddrinfo* (2)

```
for (rp = res; rp != NULL; rp = rp->ai_next) {
  // try each interface till we find one that works
  if ((sock = socket(rp->ai_family, rp->ai_socktype,
         rp->ai_protocol)) < 0) {
      continue;
  }
  if (communicate(sock, ...)) // try using the socket
      break; // it worked!
  close(sock); // didn't work
}
if (rp == NULL) {
  fprintf(stderr, "Could not contact %s\n", argv[1]);
  exit(1);
}
freeaddrinfo(res); // free up storage allocated for list
```


Client-Server Interaction

- Client sends requests to server
- Server responds
- Server may deal with multiple clients at once
- Client may contact multiple servers

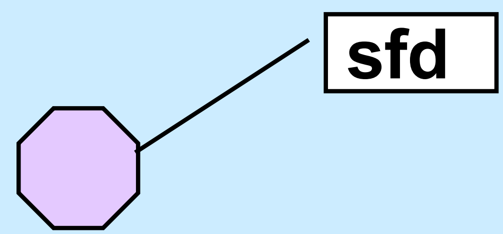
Reliable Communication

- **The promise …**
  - what is sent is received
  - order is preserved
- **Set-up is required**
  - two parties agree to communicate
  - within the implementation of the protocol:
    - » each side keeps track of what is sent, what is received
    - » received data is acknowledged
    - » unack'd data is re-sent
- **The standard scenario**
  - server receives connection requests
  - client makes connection requests

Streams in the Inet Domain (1)

- **Server steps**
  1) create socket

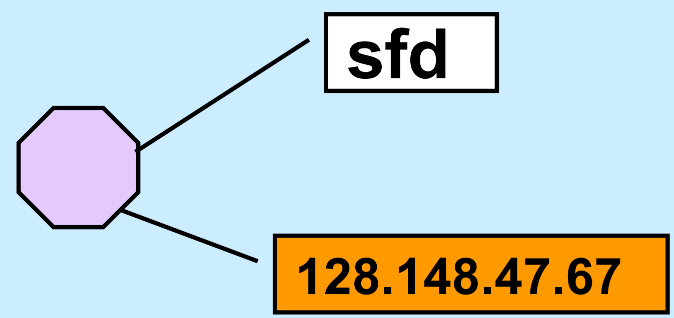
```
sfd = socket(AF_INET, SOCK_STREAM, 0);
```



Streams in the Inet Domain (2)

- **Server steps**
  2) bind name to socket

```
bind(sfd,
  (struct sockaddr *)&my_addr, sizeof(my_addr));
```



Some Details …

- **Server may have multiple interfaces; we want to be able to receive on all of them**

```
struct sockaddr_in {
  sa_family_t sin_family;
  in_port_t sin_port;
  struct in_addr sin_addr;
} my_addr;

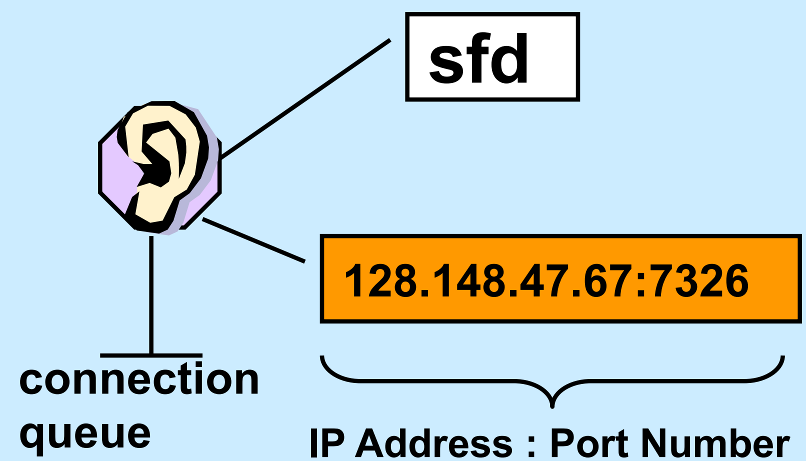
my_addr.sin_family = AF_INET;
my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
my_addr.sin_port = htons(port);
```

"Wildcard" address

Streams in the Inet Domain (3)

- **Server steps**
  3) put socket in "listening mode"

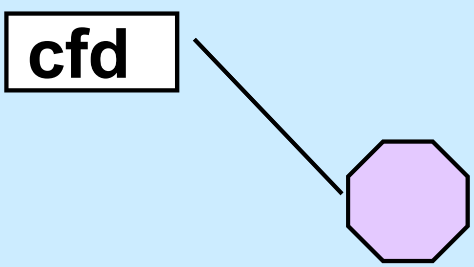
```
int listen(int sfd, int MaxQueueLength);
```



Streams in the Inet Domain (4)

- **Cient steps**
  - **1) create socket**

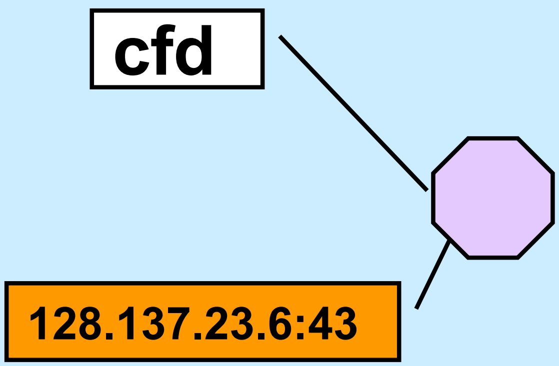
```
cfd = socket(AF_INET, SOCK_STREAM, 0);
```



Streams in the Inet Domain (5)

- **Client steps**
  - **2) bind name to socket**

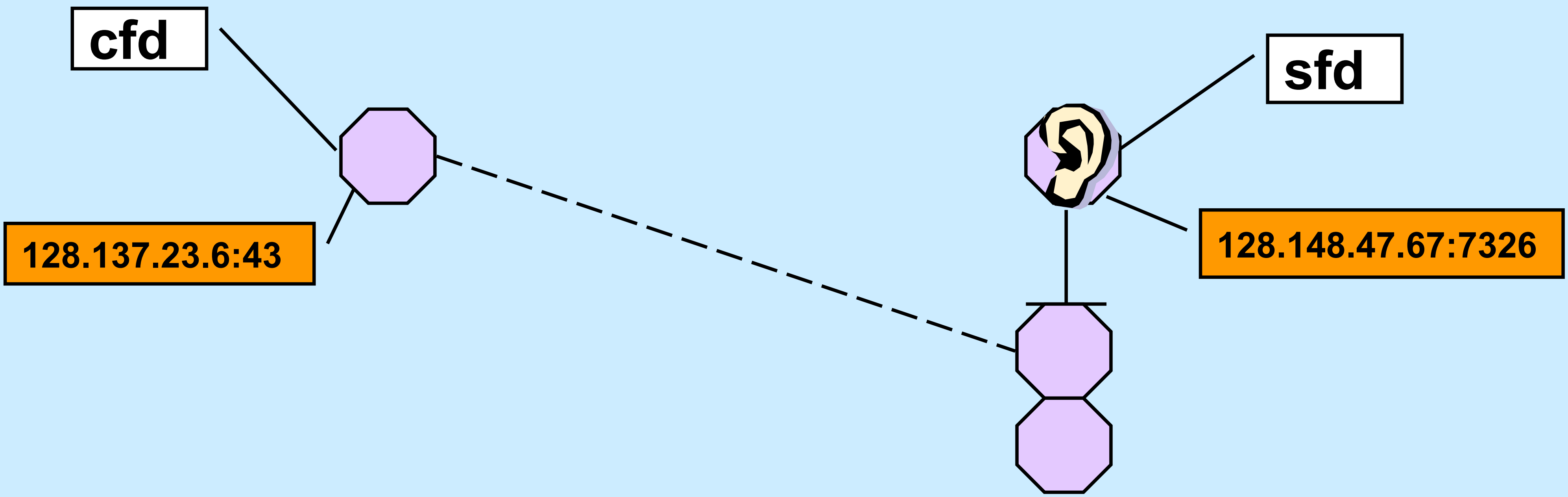
```
bind(cfd,
  (struct sockaddr *)&my_addr, sizeof(my_addr));
```



Streams in the Inet Domain (6)

- **Client steps**
  - 3) connect to server

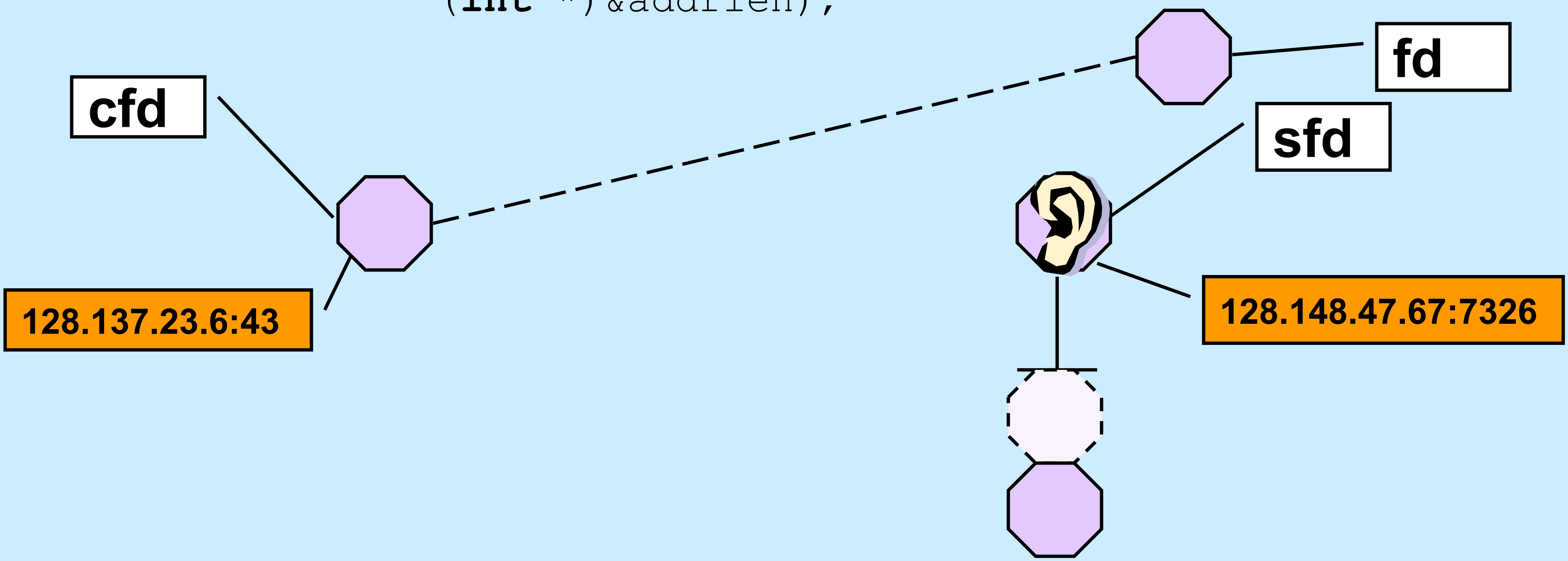
```
connect(cfd, (struct sockaddr *)&server_addr,
    sizeof(server_addr));
```



Streams in the Inet Domain (7)

- **Server steps**
  - 4) accept connection

```
fd = accept((int)sfd, (struct sockaddr *)addr,
    (int *)&addrlen);
```



Inet Stream Example (1)

- **Server side**

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int main(int argc, char *argv[ ]) {
  struct sockaddr_in my_addr;
  int lsock;
  void serve(int);
  if (argc != 2) {
    fprintf(stderr, "Usage: tcpServer port\n");
    exit(1);
  }
```

Inet Stream Example (2)

```
// Step 1: establish a socket for TCP
if ((lsock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
  perror("socket");
  exit(1);
}
```

Inet Stream Example (3)

```
/* Step 2: set up our address */
memset(&my_addr, 0, sizeof(my_addr));
my_addr.sin_family = AF_INET;
my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
my_addr.sin_port = htons(atoi(argv[1]));

/* Step 3: bind the address to our socket */
if (bind(lsock, (struct sockaddr *)&my_addr,
    sizeof(my_addr)) < 0) {
  perror("bind");
  exit(1);
}
```

Inet Stream Example (4)

```
/* Step 4: put socket into "listening mode" */
if (listen(lsock, 100) < 0) {
  perror("listen");
  exit(1);
}
while (1) {
  int csock;
  struct sockaddr_in client_addr;
  int client_len = sizeof(client_addr);

  /* Step 5: receive a connection */
  csock = accept(lsock,
      (struct sockaddr *)&client_addr, &client_len);
  printf("Received connection from %s#%hu\n",
      inet_ntoa(client_addr.sin_addr), client_addr.sin_port);
```

Inet Stream Example (5)

```
    switch (fork( )) {
    case -1:
      perror("fork");
      exit(1);
    case 0:
      // Step 6: create a new process to handle connection
      serve(csock);
      exit(0);
    default:
      close(csock);
      break;
    }
  }
}
```

Inet Stream Example (6)

```
void serve(int fd) {
  char buf[1024];
  int count;

  // Step 7: read incoming data from connection
  while ((count = read(fd, buf, 1024)) > 0) {
    write(1, buf, count);
  }
  if (count == -1) {
    perror("read");
    exit(1);
  }
  printf("connection terminated\n");
}
```

Inet Stream Example (7)

- **Client side**

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <string.h>
// + more includes ...

int main(int argc, char *argv[]) {
  int s, sock;
  struct addrinfo hints, *result, *rp;

  char buf[1024];
  if (argc != 3) {
      fprintf(stderr, "Usage: tcpClient host port\n");
      exit(1);
  }
```

Inet Stream Example (8)

```
// Step 1: find the internet address of the server
memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;

if ((s=getaddrinfo(argv[1], argv[2], &hints, &result)) != 0) {
     fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(s));
     exit(1);
}
```

Inet Stream Example (9)

```
// Step 2: set up socket for TCP and connect to server
for (rp = result; rp != NULL; rp = rp->ai_next) {
     // try each interface till we find one that works
     if ((sock = socket(rp->ai_family, rp->ai_socktype,
        rp->ai_protocol)) < 0) {
            continue;
     }
     if (connect(sock, rp->ai_addr, rp->ai_addrlen) >= 0) {
            break;
     }
     close(sock);
}
if (rp == NULL) {
     fprintf(stderr, "Could not connect to %s\n", argv[1]);
     exit(1);
}
freeaddrinfo(result);
```

Inet Stream Example (10)

```
  // Step 3: send data to the server
  while(fgets(buf, 1024, stdin) != 0) {
      if (write(sock, buf, strlen(buf)) < 0) {
          perror("write");
          exit(1);
      }
  }
  return 0;
}
```

Quiz 3

The previous slide contains
`write(sock, buf, strlen(buf))`

If data is lost and must be retransmitted

a) **write returns an error so the caller can retransmit the data.**

b) **nothing happens as far as the application code is concerned, the data is retransmitted automatically.**

Quiz 4

A previous slide contains

```
write(sock, buf, strlen(buf))
```

We lose the connection to the other party (perhaps a network cable is cut).

a) write returns an error so the caller can reconnect, if desired.

b) nothing happens as far as the application code is concerned, the connection is reestablished automatically.