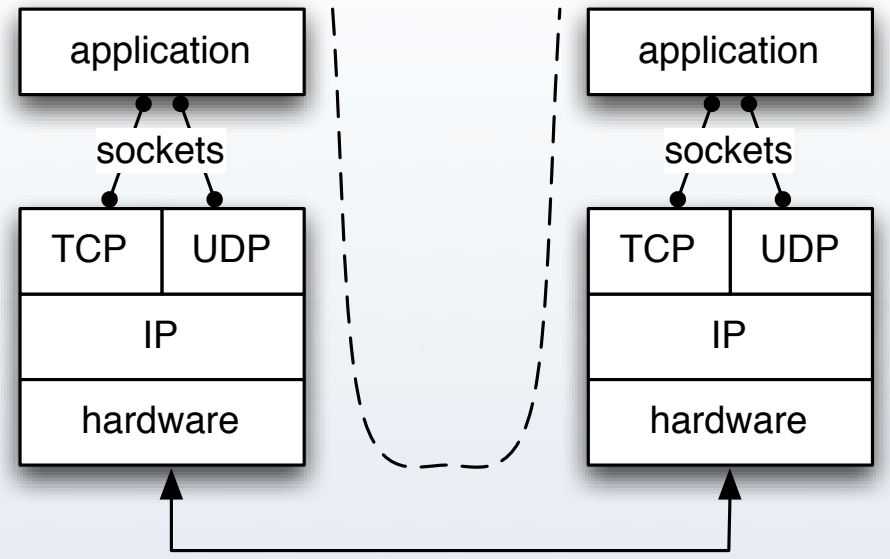
Introduction

Why do we need them?

- TCP and UDP are the basis of any networked application
- IP is standardized by RFCs and implemented in the operating systems
- a ***socket*** = the end-point of a process-to-process communication flow across an IP based network



- communication:
  - connection-oriented
  - connection-less

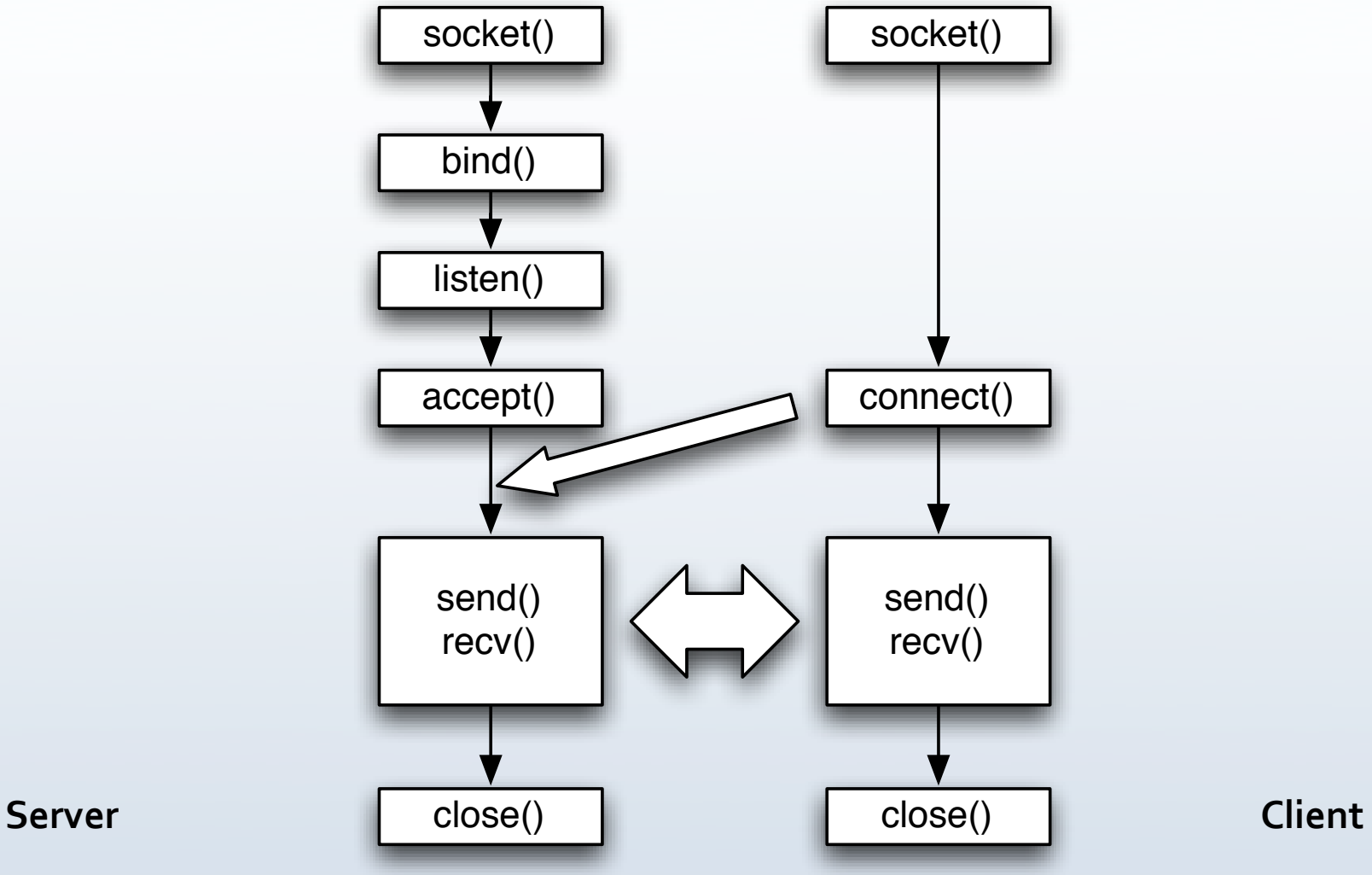
Introduction

Stream sockets

- stream socket = virtual circuit between two processes
- the connection is:
  - sequenced
  - reliable
  - bi-directional
- uniquely identified by the **IP addresses** and **port numbers** (remote and local)
- setup by:
  - a server that awaits for connections
  - a client that connects to a server

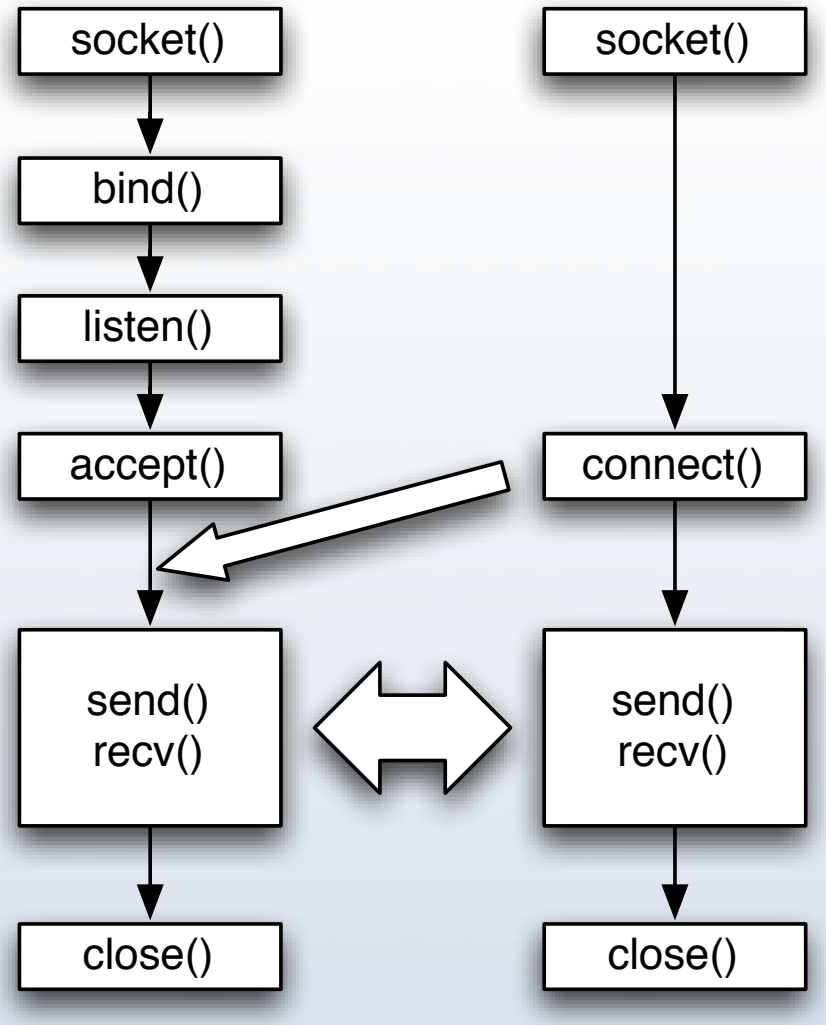
Introduction

Stream sockets



Introduction

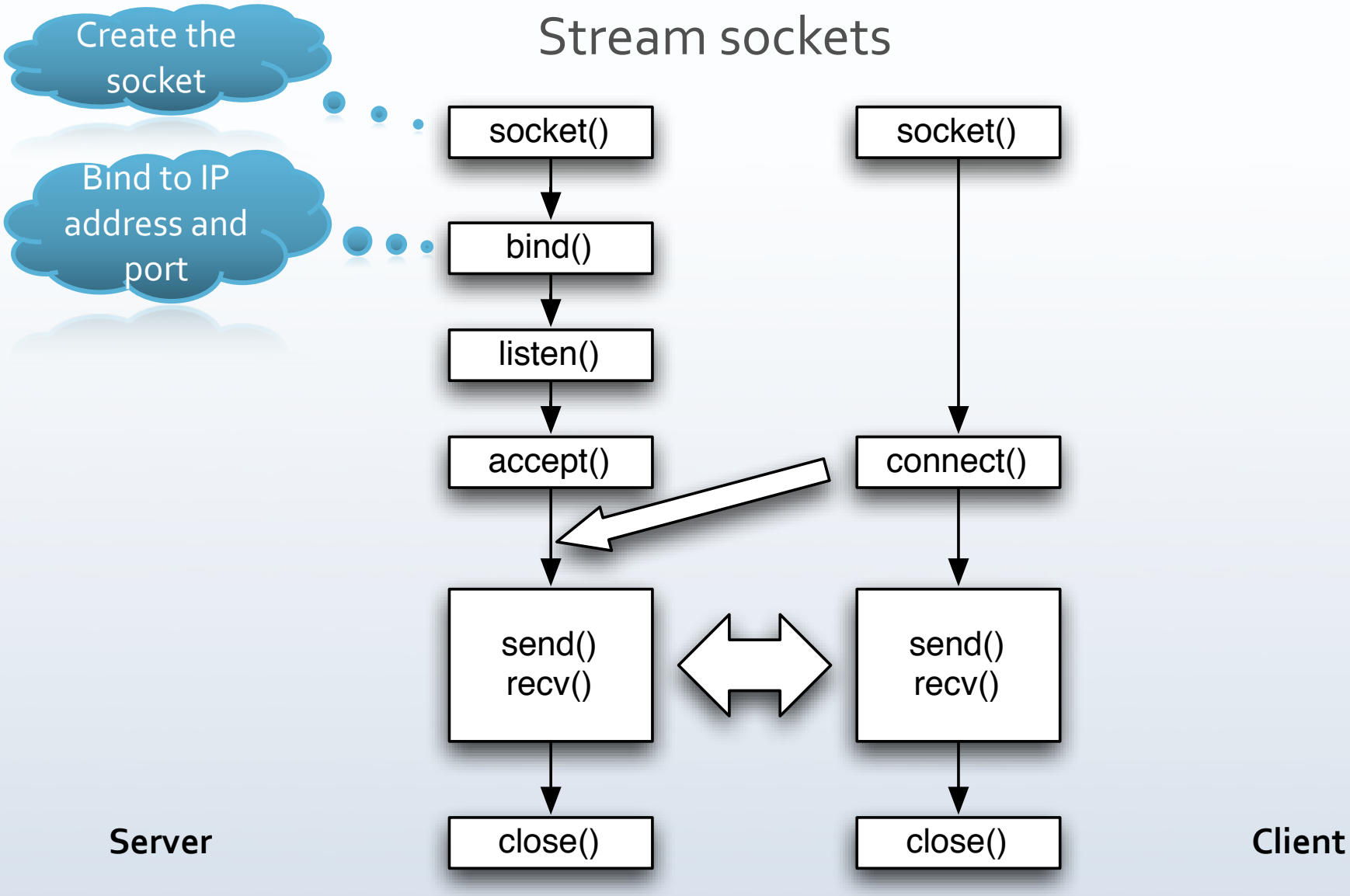
Stream sockets



Server

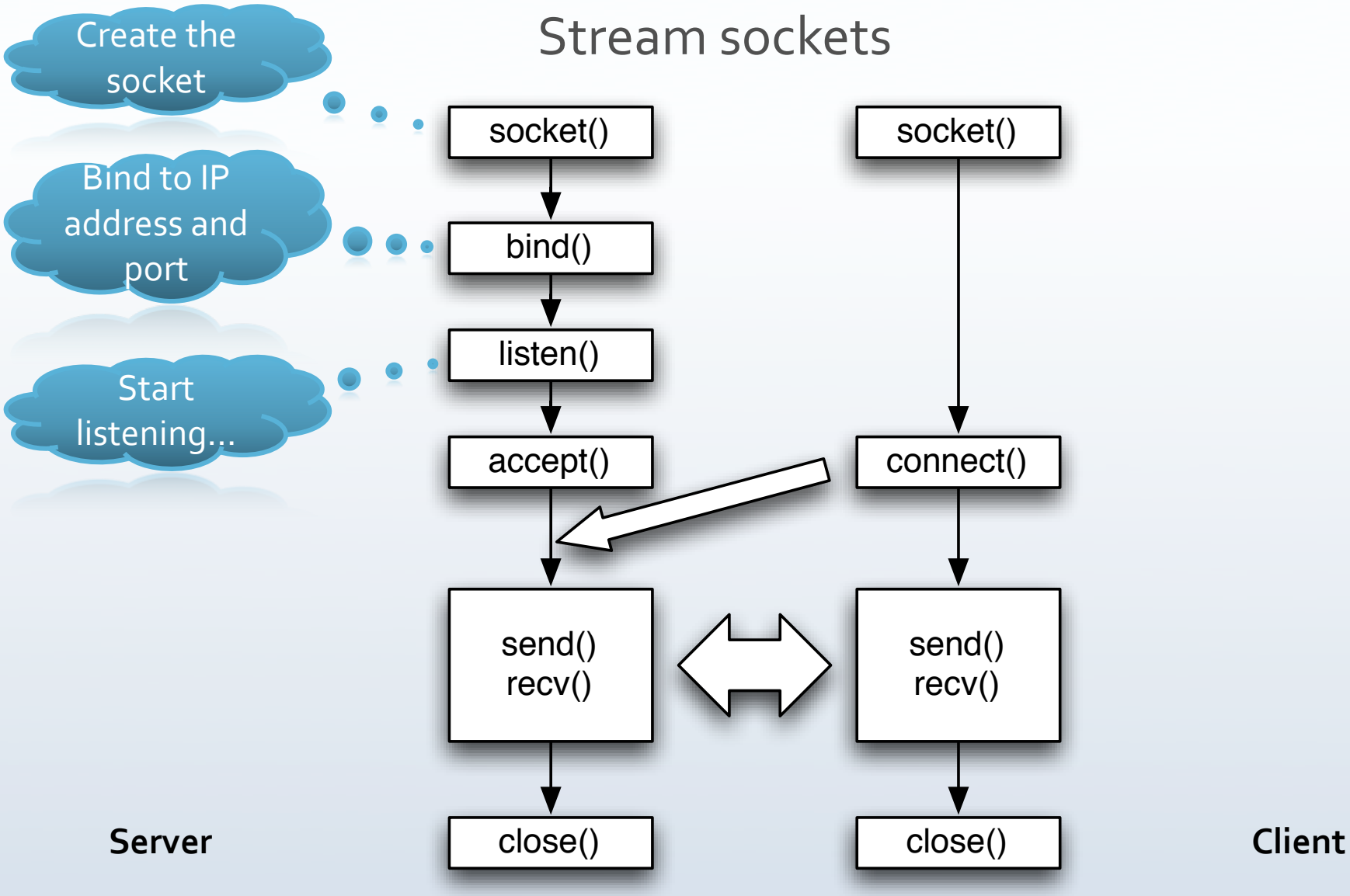
Client

Introduction



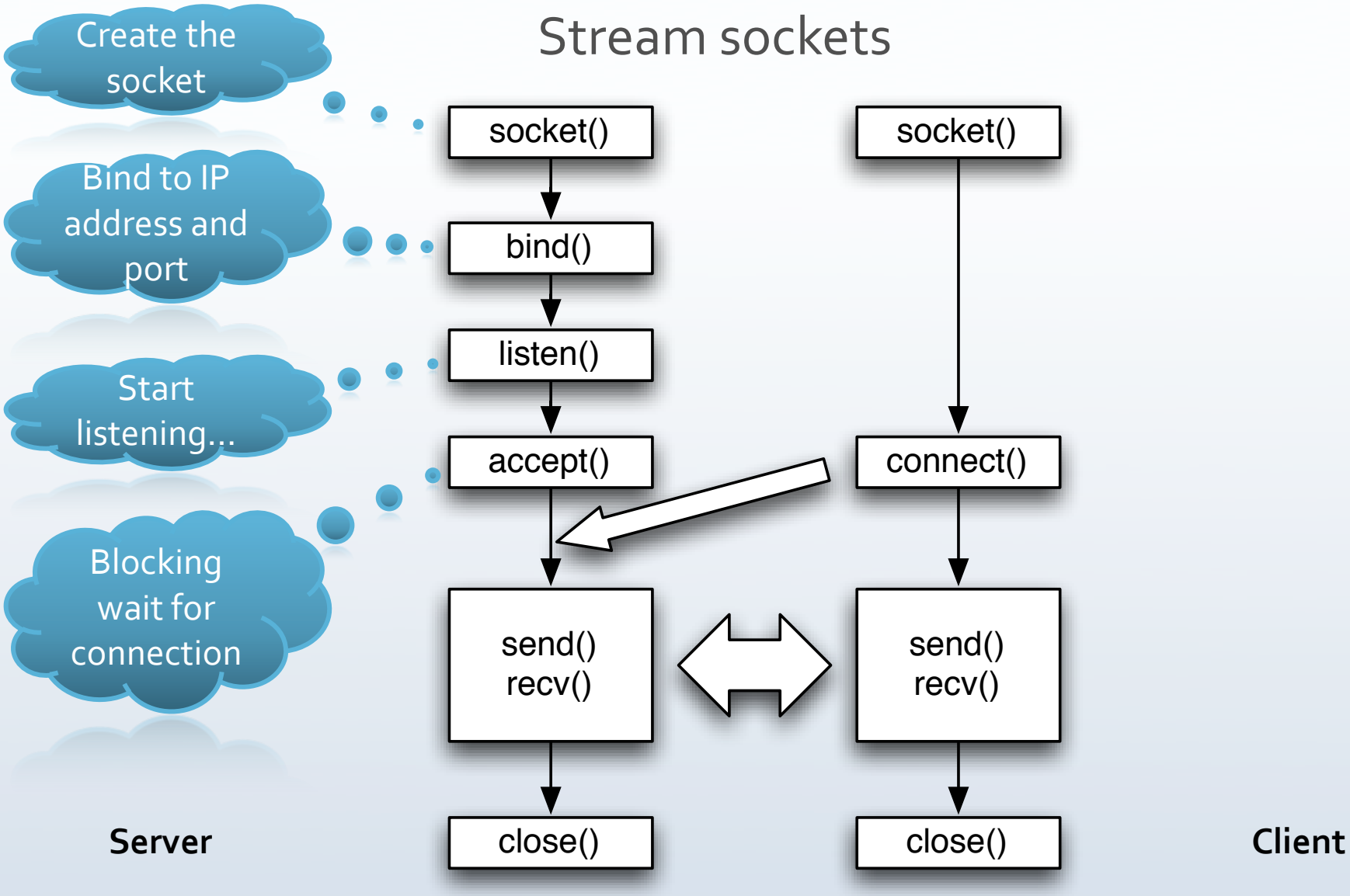
Introduction

Stream sockets

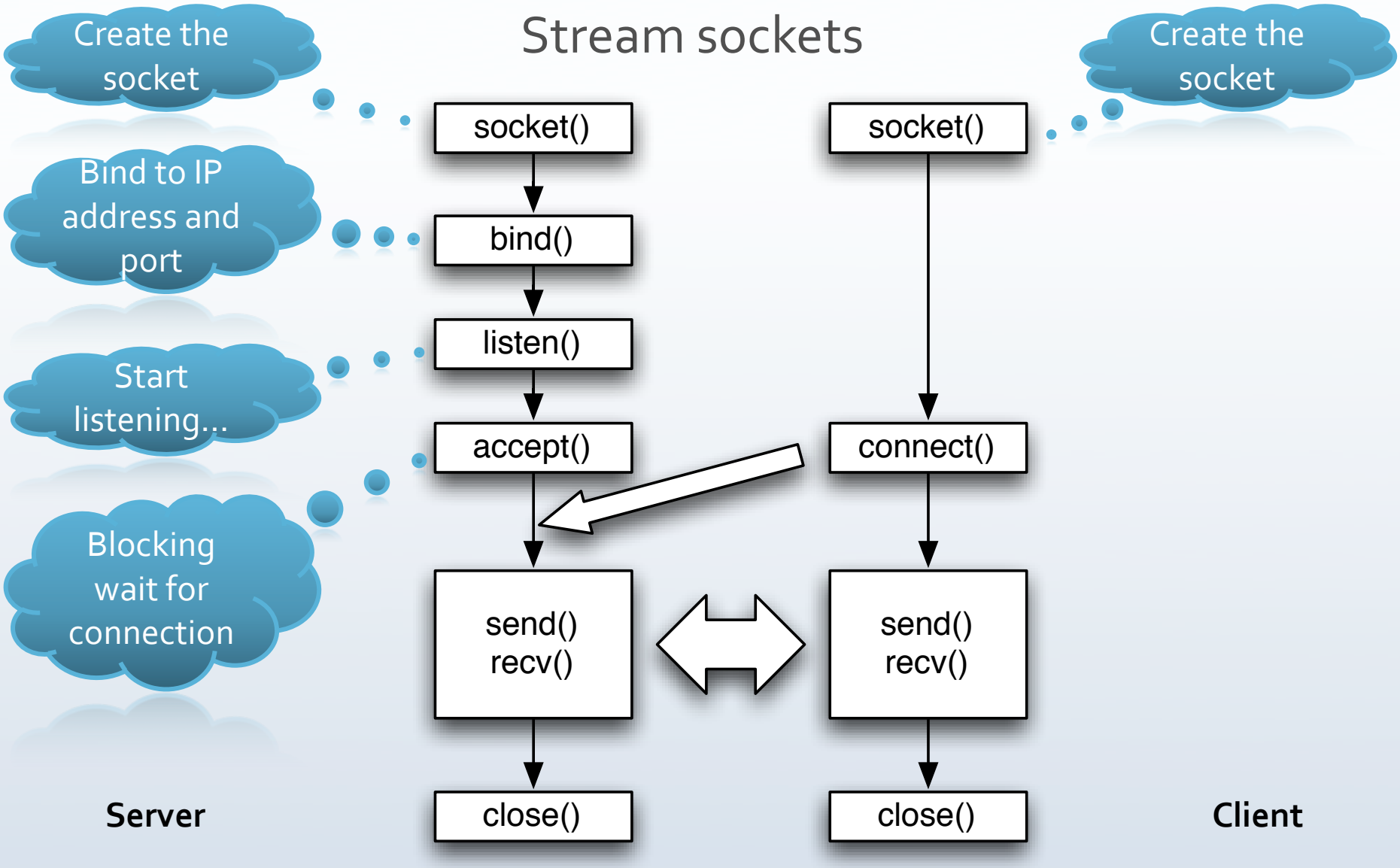


Introduction

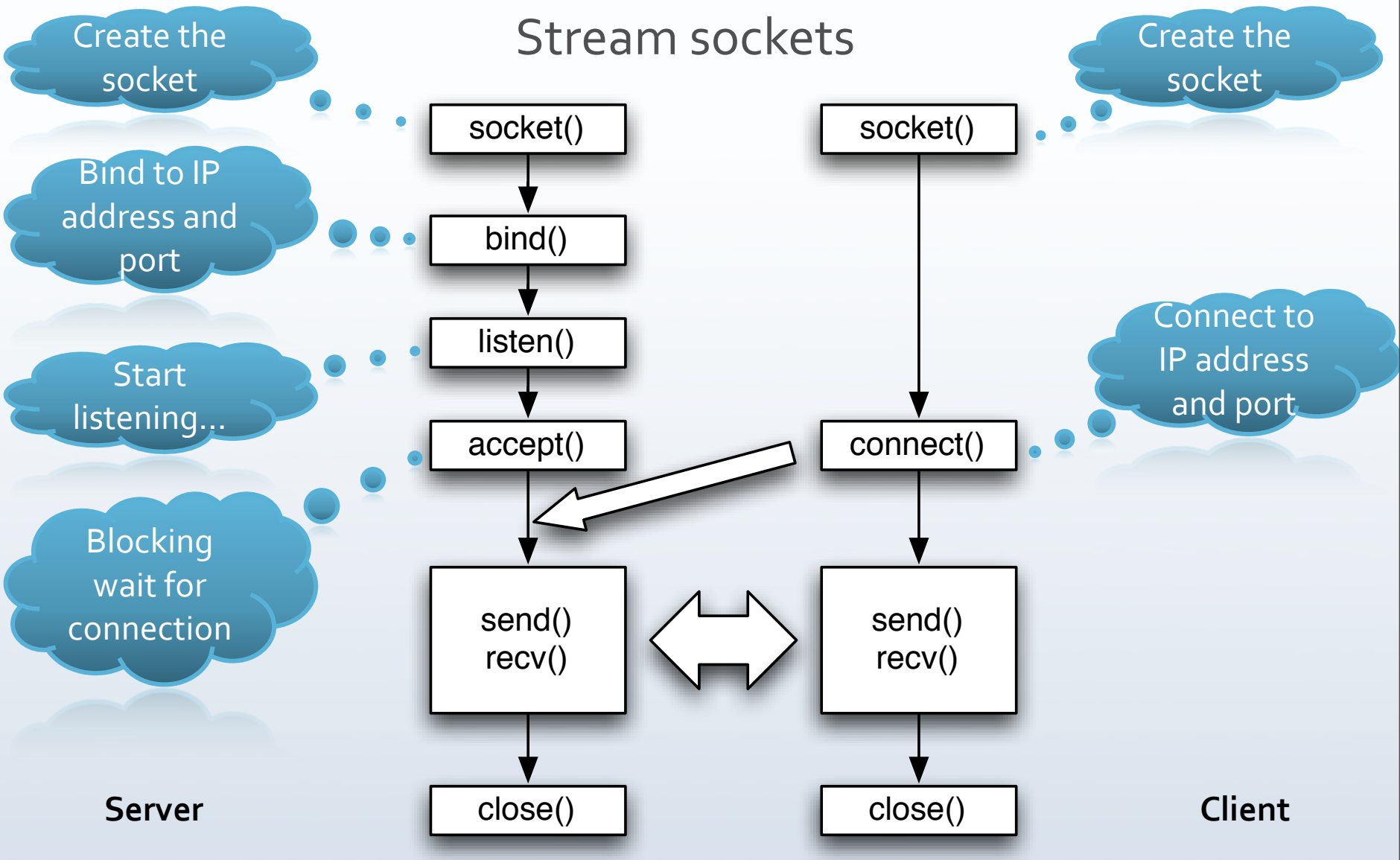
Stream sockets



Introduction



Introduction



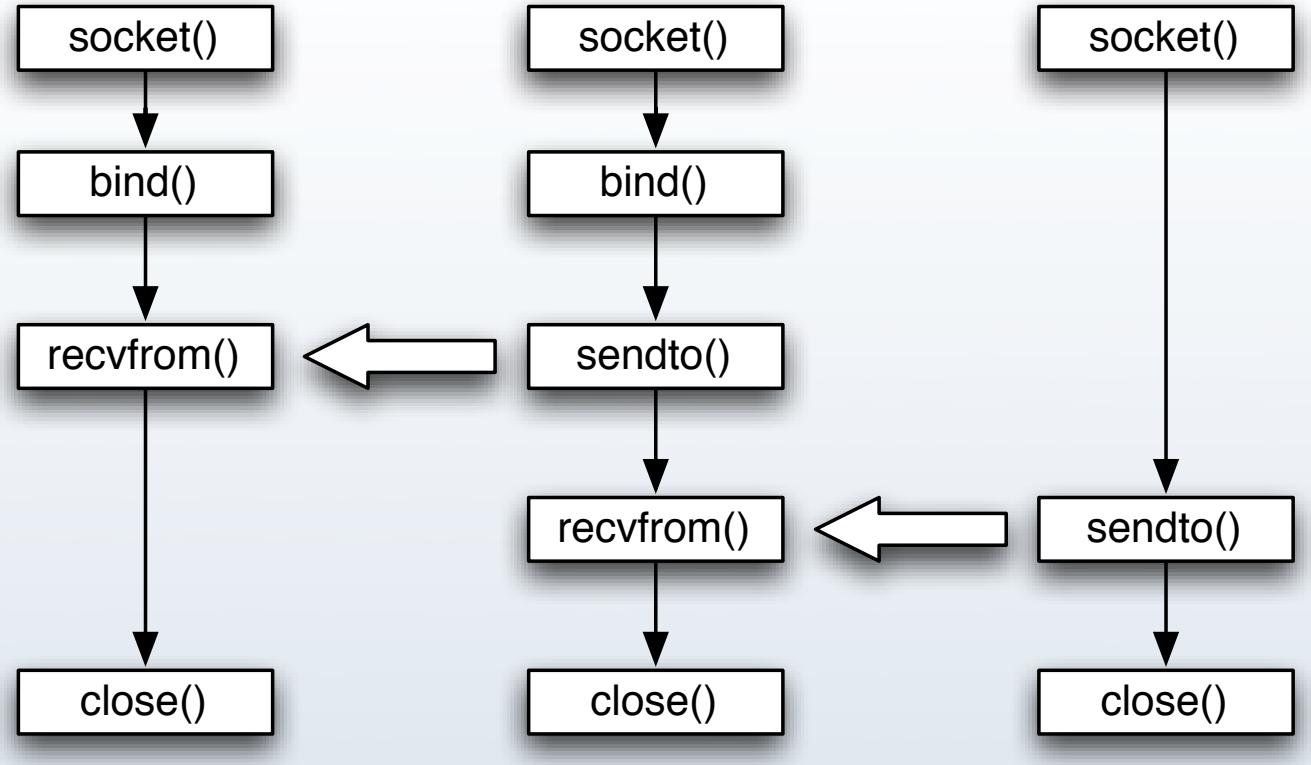
Introduction

Datagram sockets

- no distinction between server and client
- no connection and unreliable
- the same socket can be used to send/receive datagrams to/from multiple processes

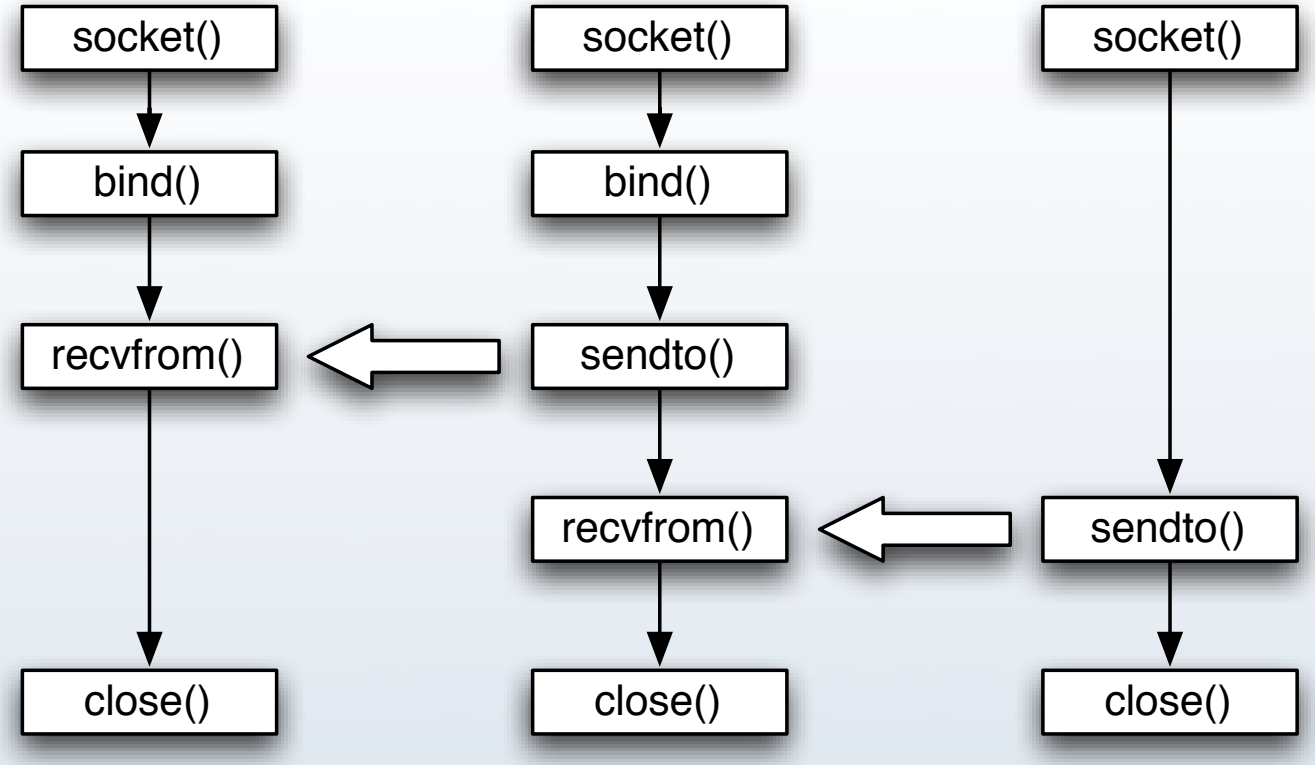
Introduction

Datagram sockets



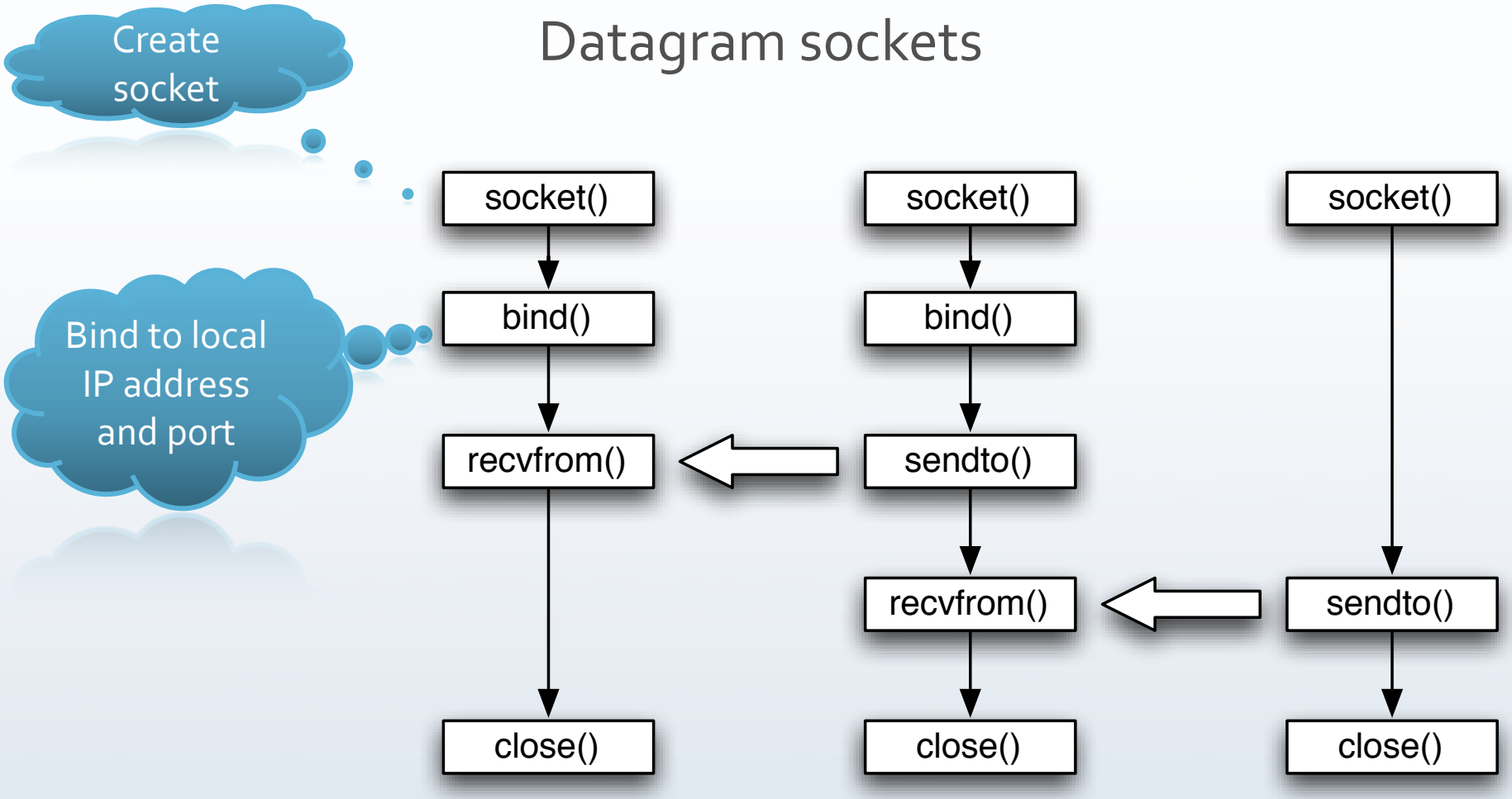
Introduction

Datagram sockets



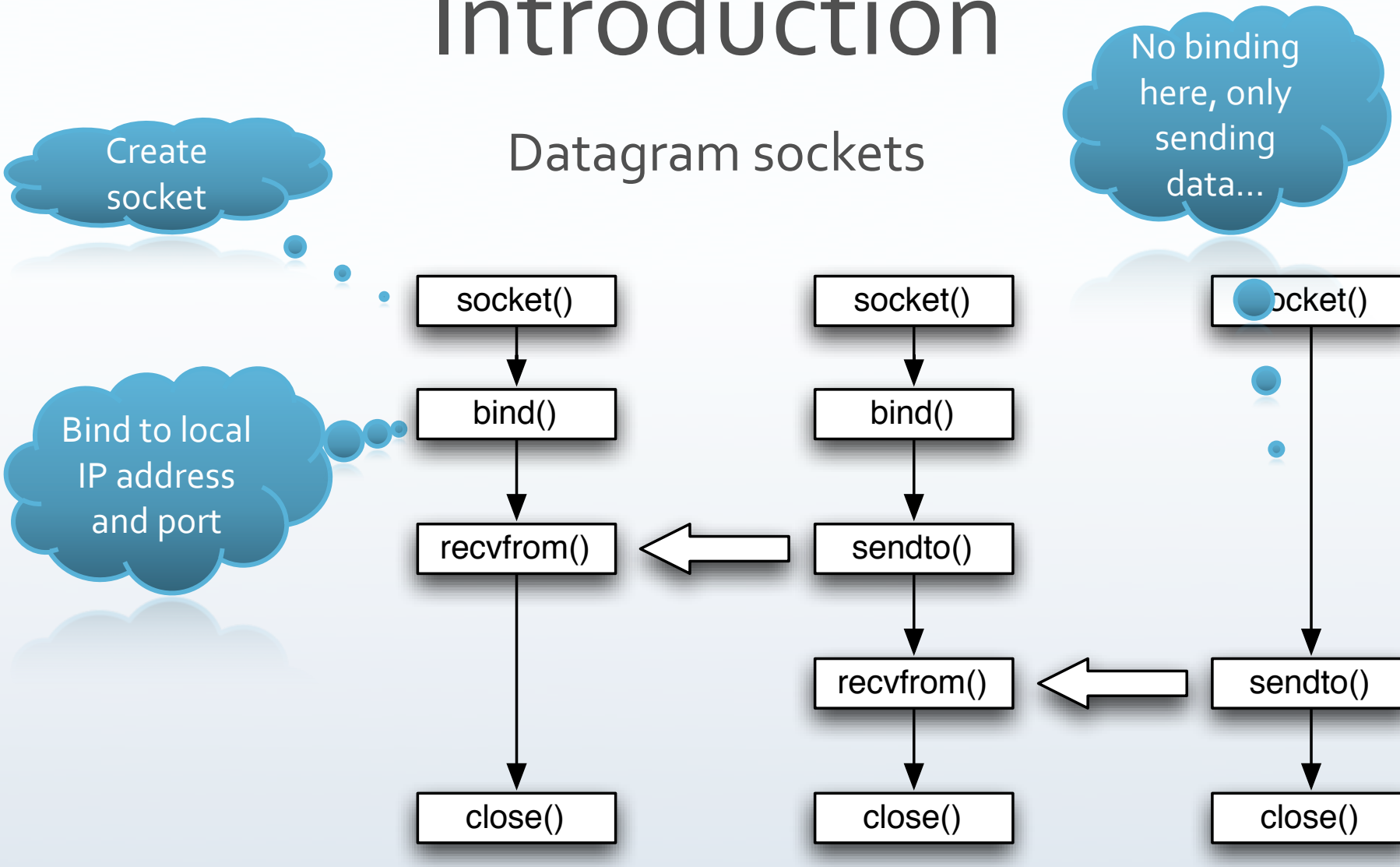
Introduction

Datagram sockets



Introduction

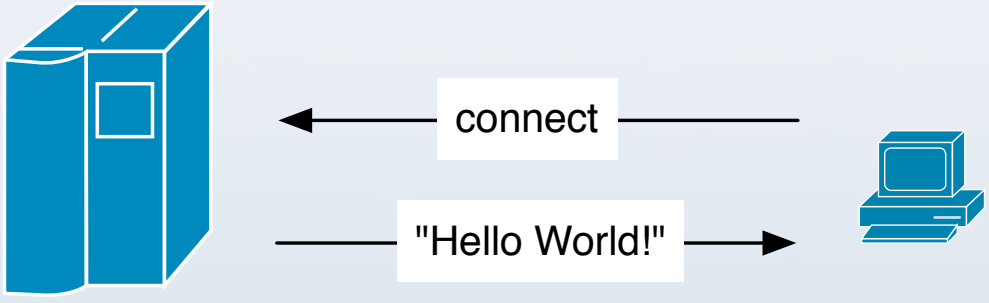
Datagram sockets



Example 1

Hello World! using TCP

- clients connects to server
- server transmits “Hello World!”
- client displays the message
- connection terminates



Example 1

server: creating the socket

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

int main()
{
    int sockfd;

    /* create server-side socket */
    sockfd = socket(AF_INET, SOCK_STREAM,
                    IPPROTO_TCP);
…
```

a stream socket
otherwise: SOCK_DGRAM

protocol
family

using TCP
otherwise: IPPROTO_UDP

Example 1

server: binding the address

```
struct sockaddr_in my_addr;
my_addr.sin_family = AF_INET; /*IPv4 *
my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
my_addr.sin_port = htons(2000);

bind(sockfd, &my_addr, sizeof(my_addr));

listen(sockfd, MAX_CONNECTIONS);
```

any local address

Example 1

server: dealing with clients

a new socket is created for each client

```
while(1) {
  struct sockaddr_in client_addr;
  int client_len = sizeof(client_addr);

  int client_fd =
    accept(sockfd, &client_addr,
           &client_len);


  strcpy(buf, "Hello World!");
  send(client_fd, buf, strlen(buf) + 1, 0);
  close(client_fd);
}
```

Example 1

client: looking up the server

hostname
try
"www.google.com"

```
struct hostent *server_host;
server_host = gethostbyname("localhost");

/* configure the server address */
struct sockaddr_in server_addr;
server_addr.sin_family = AF_INET;
memcpy(&server_addr.sin_addr,
       server_host->h_addr,
       sizeof(struct in_addr));
server_addr.sin_port = htons(PORT);
```

Example 1

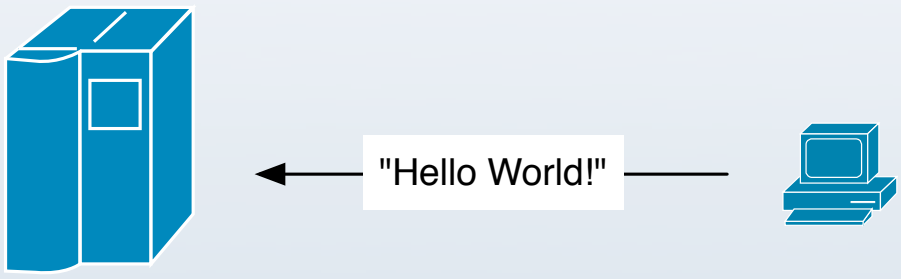
client: the other steps

```
/* create client-side socket */
sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

   /* connect to server */
connect(sockfd, &server_addr, sizeof(server_addr));

char buf[MAX_BUF];

int len = recv(sockfd, buf, MAX_BUF, 0);

printf("received %d bytes: '%s'\n", len, buf);

close(sockfd);
```

Example 2

Hello World! using UDP

- client sends an UDP message "Hello World!" to server
- the server is almost identical to the TCP version
- the client does not need to call `connect`



Example 2

server

```
int sockfd = socket(AF_INET, SOCK_DGRAM,IPPROTO_UDP);

struct sockaddr_in my_addr, client_addr;
my_addr.sin_family = AF_INET;
my_addr.sin_addr.s_addr = htonl(INADDR_ANY)
my_addr.sin_port = htons(MY_PORT);

/* bind it to the local address */
bind(sockfd, &my_addr, sizeof(my_addr));
listen(sockfd, MAX_CONNECTIONS);
int client_addr_len;
int len = recvfrom(sockfd, buf, MAX_BUF, 0,
            &client_addr, &client_addr_len);
```

the receiver should know who sent the data

Example 2

client

```
struct hostent *server_host;
server_host = gethostbyname("localhost");

/* configure the server address */
struct sockaddr_in server_addr;
server_addr.sin_family = AF_INET; // IPv4
memcpy(&server_addr.sin_addr, server_host->h_addr,
                sizeof(struct in_addr));
server_addr.sin_port = htons(SERVER_PORT);

int sockfd = socket(AF_INET, SOCK_DGRAM,
                IPPROTO_UDP);

/* send a message */
strcpy(buf, "Hello World!");
sendto(sockfd, buf, strlen(buf) + 1, 0,
                &server_addr, sizeof
  (server_addr));
```