NETWORK PROGRAMMING

Socket primitives

- defined in <sys/socket.h>

- **SOCKET**: **int socket(int *domain*, int *type*, int *protocol)*;**
  - *domain* := AF_INET (IPv4 protocol)
  - *type* := (SOCK_DGRAM *or* SOCK_STREAM)
  - *protocol* := 0 (IPPROTO_UDP *or* IPPROTO_TCP)
  - *returned*: socket descriptor (*sockfd*), -1 is an error

- **BIND**: **int bind(int *sockfd*, struct sockaddr **my_addr*, int *addrlen*);**
  - *sockfd* - socket descriptor (returned from socket())
  - *my_addr:* socket address, struct sockaddr_in is used
  - *addrlen* := sizeof(struct sockaddr)

```
struct sockaddr_in  {
  unsigned short  sin_family;  /* address family (always AF_INET) */
  unsigned short  sin_port;    /* port num in network byte order */
  struct in_addr  sin_addr;    /* IP addr in network byte order */
  unsigned char   sin_zero[8]; /* pad to sizeof(struct sockaddr) */
};
```

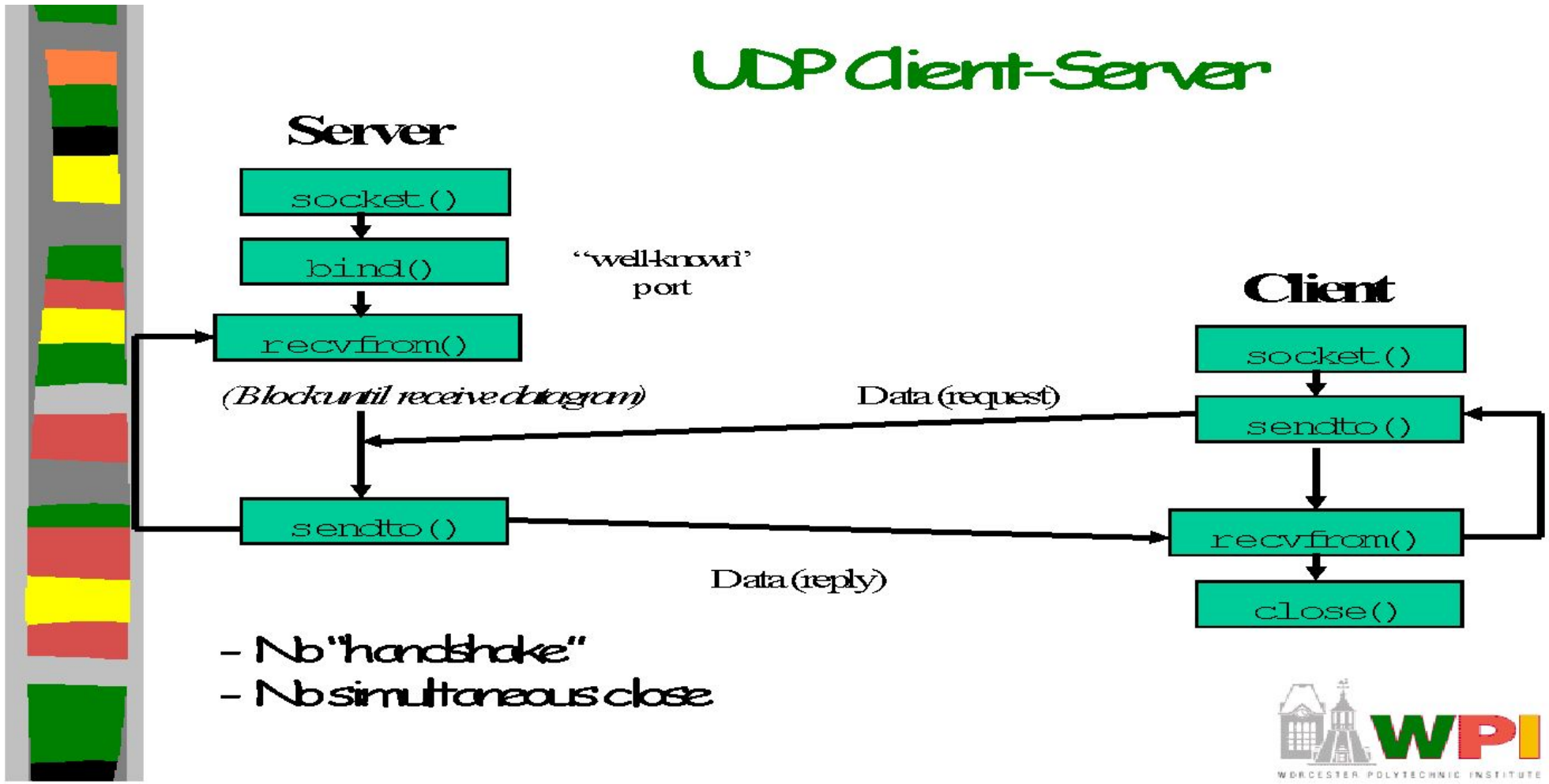
- **LISTEN**: **int listen(int *sockfd*, int *backlog*);**
  - *backlog*: how many connections we want to queue
- **ACCEPT: int accept(int sockfd, void *addr, int *addrlen);**
  - *addr*: here the socket-address of the caller will be written
  - *returned:* a new socket descriptor (for the temporal socket)
- **CONNECT: int connect(int sockfd, struct sockaddr *serv_addr, int addrlen); //used by TCP client**
  - parameters are same as for bind()
- **SEND: int send(int sockfd, const void *msg, int len, int flags);**
  - *msg*: message you want to send
  - *len:* length of the message
  - *flags* := 0
  - *returned:* the number of bytes actually sent
- **RECEIVE: int recv(int sockfd, void *buf, int len, unsigned int flags);**
  - *buf:* buffer to receive the message
  - *len:* length of the buffer (“don’t give me more!”)
  - *flags* := 0
  - *returned:* the number of bytes received

- **SEND** (DGRAM-style): **int sendto(int *sockfd*, const void **msg*, int *len*, int *flags,* const struct sockaddr **to*, int *tolen*);**
  - *msg*: message you want to send
  - *len:* length of the message
  - *flags* := 0
  - *to:* socket address of the remote process
  - *tolen*: = sizeof(struct sockaddr)
  - *returned:* the number of bytes actually sent

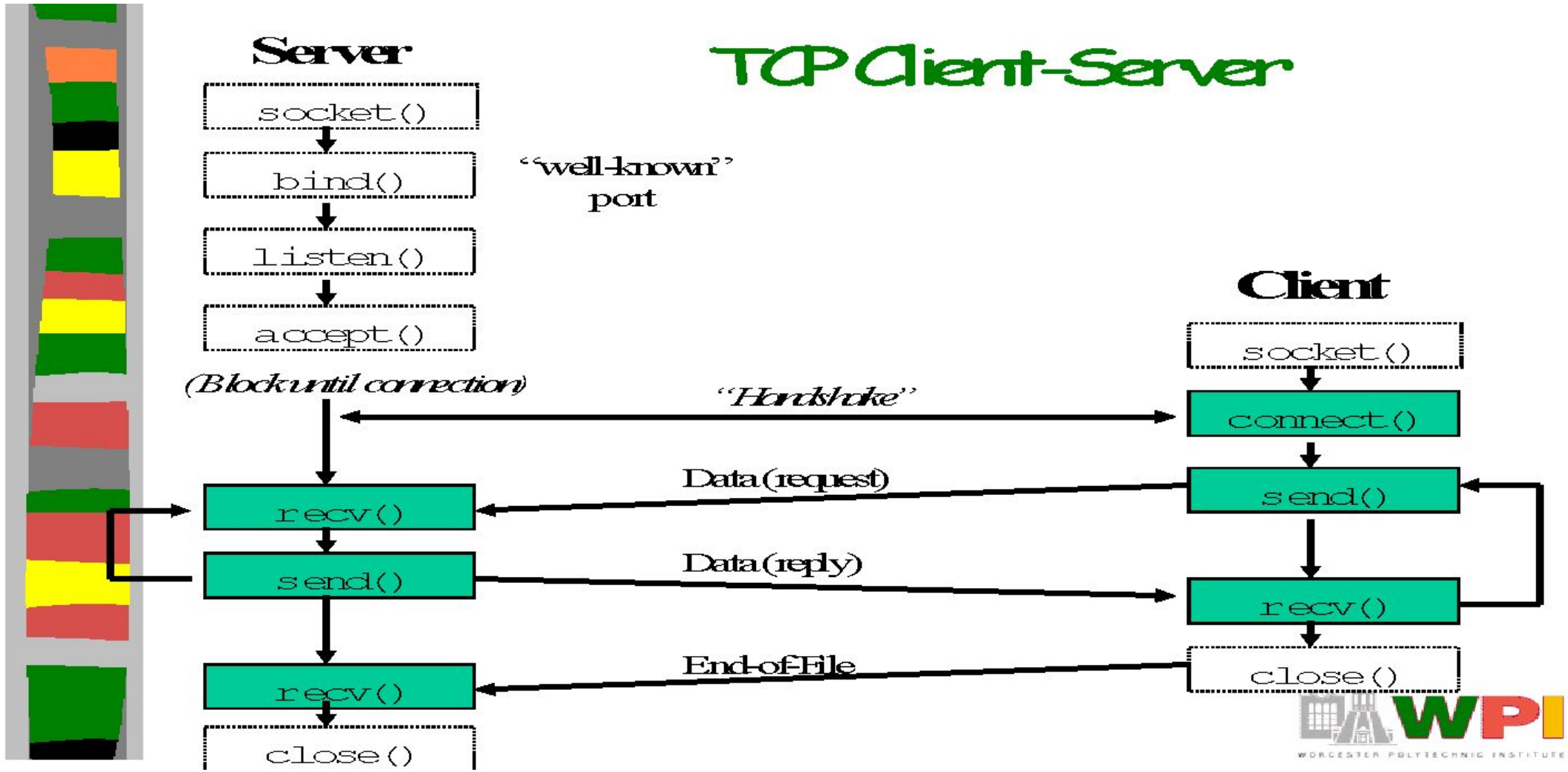
- **RECEIVE (DGRAM-style): int recvfrom(int sockfd, void *buf, int len, unsigned int flags, struct sockaddr *from, int *fromlen);**
  - *buf:* buffer to receive the message
  - *len:* length of the buffer (“don’t give me more!”)
  - *from*: socket address of the process that sent the data
  - *fromlen*:= sizeof(struct sockaddr)
  - *flags* := 0
  - *returned:* the number of bytes received

- **CLOSE: close (socketfd);**

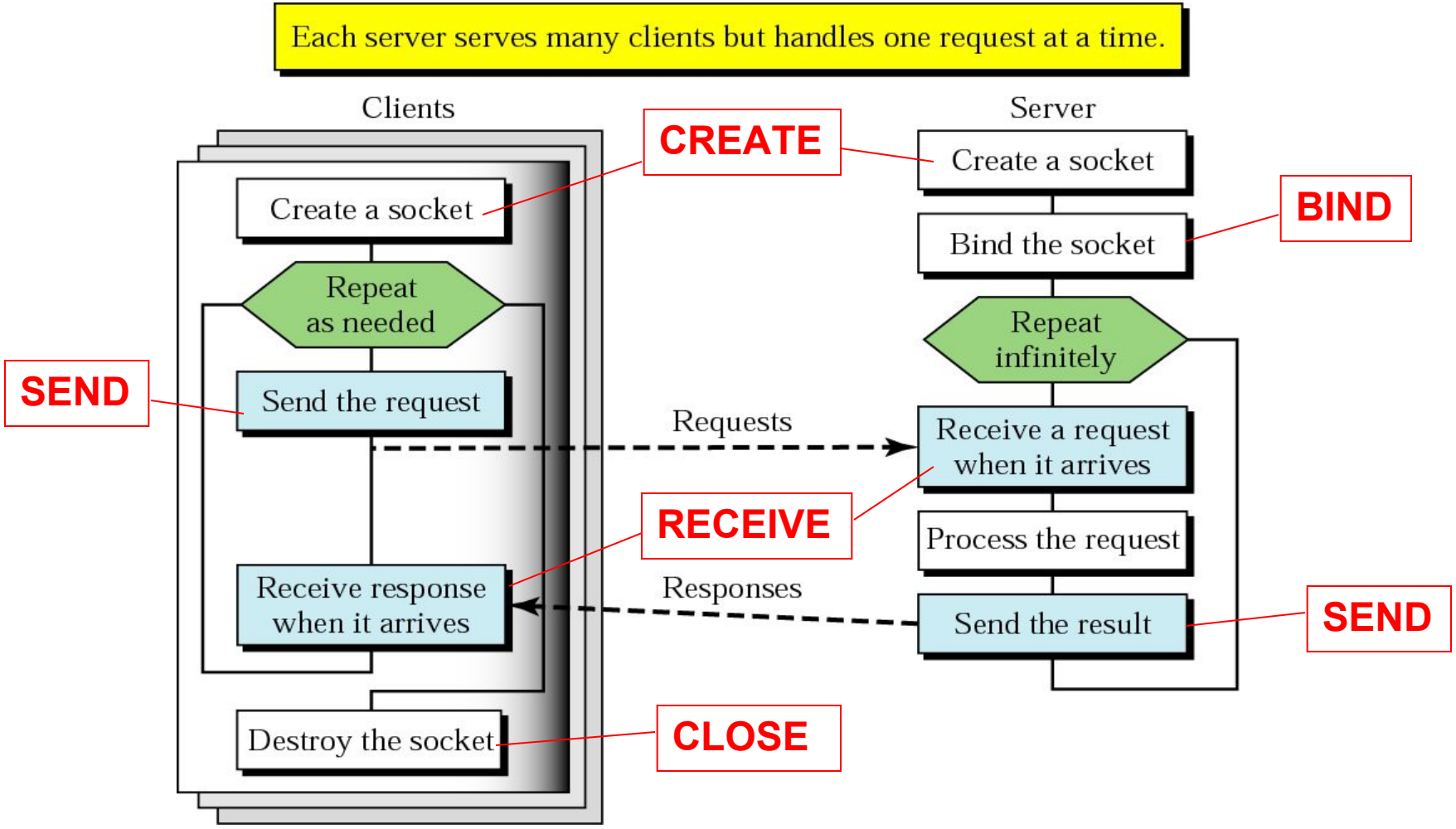
UDP Connection Example



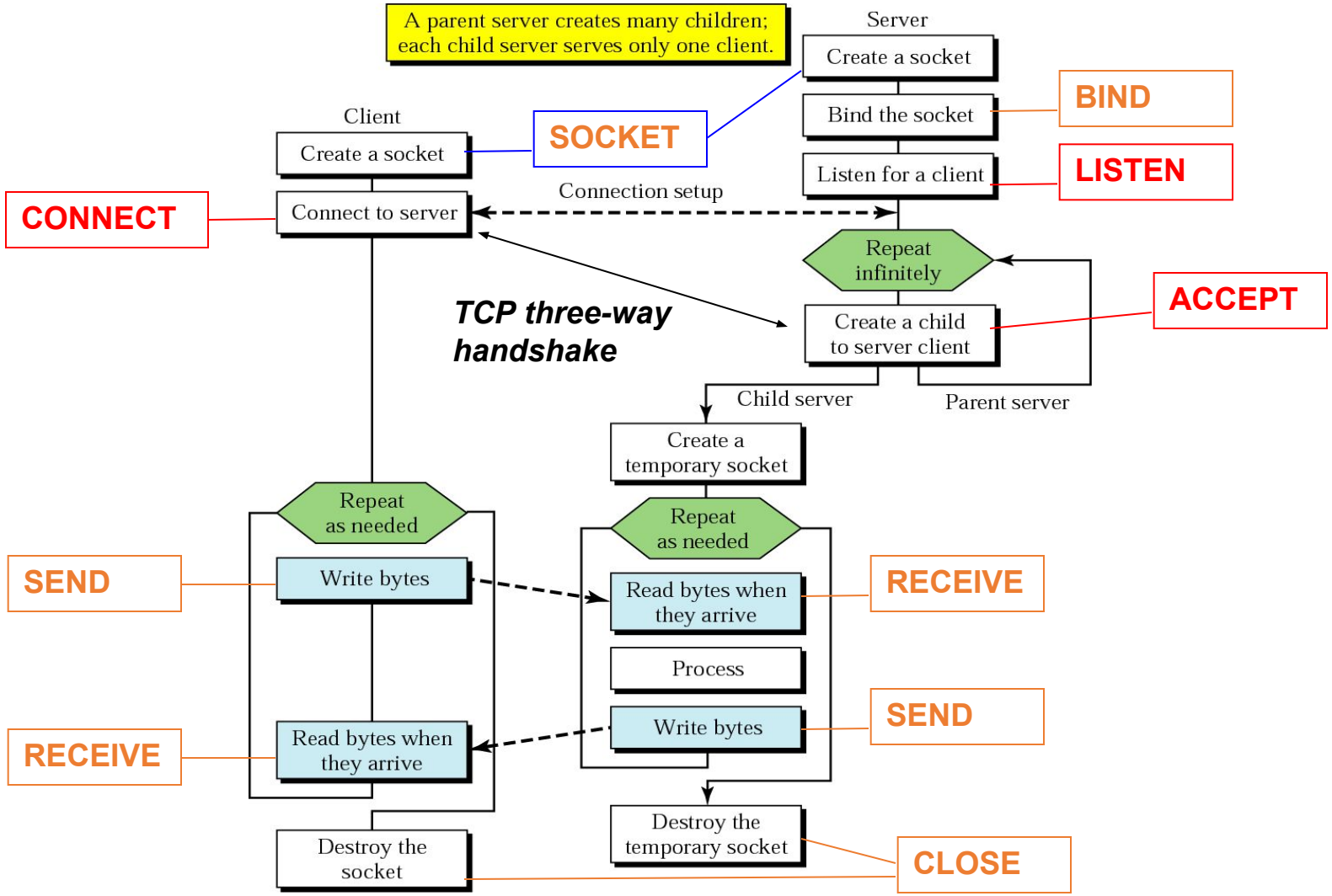
TCP Connection Example



Client+server: **connectionless**



Client+server: **connection-oriented**



Concurrent server

#include's

```
#include <stdio.h>      /* for printf() and fprintf() */
#include <sys/socket.h> /* for socket(), connect(),
                           sendto(), and recvfrom() */
#include <arpa/inet.h>  /* for sockaddr_in and
                           inet_addr() */
#include <stdlib.h>     /* for atoi() and exit() */
#include <string.h>     /* for memset() */
#include <unistd.h>     /* for close() */
```

- **htons:** host to network short : convert a number into a 16-bit network representation. This is commonly used to store a port number into a sockaddr structure.
- **htonl:** host to network long : convert a number into a 32-bit network representation. This is commonly used to store an IP address into a sockaddr structure.
- **ntohs:** network to host short : convert a 16-bit number from a network representation into the local processor's format. This is commonly used to read a port number from a sockaddrstructure.
- **ntohl:** network to host long : convert a 32-bit number from a network representation into the local processor's format. This is commonly used to read an IP address from a sockaddrstructure.

How to make client:

- Create a socket with the *socket ()* system call.
- Connect the socket to the address of the server using the *connect ()* system call.
- Send and receive data. There are a number of ways to do this, but the simplest is to use the *read ()* and *write ()* system calls.

How to make a server:

- Create a socket with the *socket ()* system call.
- Bind the socket to an address using the *bind ()* system call. For a server socket on the Internet, an address consists of a port number on the host machine.
- Listen for connections with the *listen ()* system call.
- Accept a connection with the *accept ()* system call. This call typically blocks until a client connects with the server.
- Send and receive data using the *read ()* and *write ()* system calls.

ALGORITHM-SERVER

Step 1: Start

Step 2: Create socket with socket() system call and bind it to IP address and port number using bind().

Step 3: Listen for connection request from client with the *listen ()* system call.

Step 4: Accept connection request with the *accept ()* system call.

Step 5: Receive string from client using recv()

Step 6: Reverse the received string

Step 7: Send back the reversed string to client using send()

Step 8: Stop

ALGORITHM-client

Step 1: Start

Step 2: Create socket with socket() system call

Step 3: Send connection request to server using connect().

Step 4: Accept string from user

Step 5: Send string to server using send()

Step 6: Receive reversed string from server using recv()

Step 7: Print reversed string on the screen

Step 8: Stop

SERVER

```
main()
{
      struct sockaddr_in client, server;
      int s,n,j,i,sock,flag;
      char st1[20],rev[20];
 s=socket(AF_INET,SOCK_STREAM,0);
      server.sin_family=AF_INET;
      server.sin_port=2000;
      server.sin_addr.s_addr=inet_addr("127.0.0.1");

      bind(s, (struct sockaddr *)&server, sizeof (server));
      listen(s,1);
     n=sizeof (client);
sock=accept(s,(struct sockaddr *)&client, &n);
```

```
 recv(sock,st1,sizeof(st1),0);
     printf("\nThe string received is:%s\n\n",st1);
          j=strlen(st1);
          j=j-1;
          for(i=0;st1[i]!='\0';i++,j--)
         {
          rev[i]=st1[j];
      }
          rev[i]='\0';
send(sock,&rev, sizeof(rev),0);
close(sock);
     close(s);
}
```

CLIENT

```
main()
{
struct sockaddr_in client,server;
      int s,flag;
     char rec[20],buffer[20];
     s=socket(AF_INET,SOCK_STREAM,0);
     client.sin_family=AF_INET;
     client.sin_port=2000;
     client.sin_addr.s_addr=inet_addr("127.0.0.1");
     connect(s,(struct sockaddr *)&server,sizeof(server));
```

```
printf("\nEnter a string: ");
    scanf("%s",buffer);
    printf("\nClient: %s",buffer);
    send(s,buffer,sizeof(buffer),0);
 recv(s,rec,sizeof(rec),0);
        printf("\nReversed String is: %s\n\n",rec);
    close(s);
}
```

SERVER

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

main()
{
int sd,cd;
char buf[1000]="",fname[10];
struct sockaddr_in ser;

sd=socket(AF_INET,SOCK_STREAM,0);

if(sd<0)
printf("SOCKET NOT CREATED\n");
```

```
bzero(&ser,sizeof(struct sockaddr_in));
ser.sin_family=AF_INET;
ser.sin_port=htons(1101);
inet_aton("localhost",&ser.sin_addr);
int b=bind(sd,(struct sockaddr *)&ser,sizeof(ser));
printf("BIND VALUE:%d\n",b);

listen(sd,5);

for(;;)
{
cd=accept(sd,NULL,NULL);
int pid=fork();
if(pid==0)
{
printf("accept value %d\n",cd);
read(cd,buf,1000);
int fd=open(buf,O_RDONLY);
read(fd,buf,1000);
write(cd,buf,strlen(buf));

printf("MESSAGE FROM CLIENT:\n%s\n",buf);
close(cd);
```

CLIENT

```
main()
{
int sd,cd;
char buf[1000]="",buf1[1000]="";
struct sockaddr_in ser;
sd=socket(AF_INET,SOCK_STREAM,0);
bzero(&ser,sizeof(struct sockaddr_in));
ser.sin_family=AF_INET;
ser.sin_port=htons(1101);

inet_aton("localhost",&ser.sin_addr);
connect(sd,(struct sockaddr *)&ser,sizeof(ser));
```

```
for(;;)
{
printf("ENTER THE MESSAGE: ");
scanf("%s",buf);

write(sd,buf,strlen(buf));

read(sd,buf,1000);

printf("RECEIVED FROM SERVER\n%s\n",buf);
}
close(sd);
}
```