Applications – Socket Communication

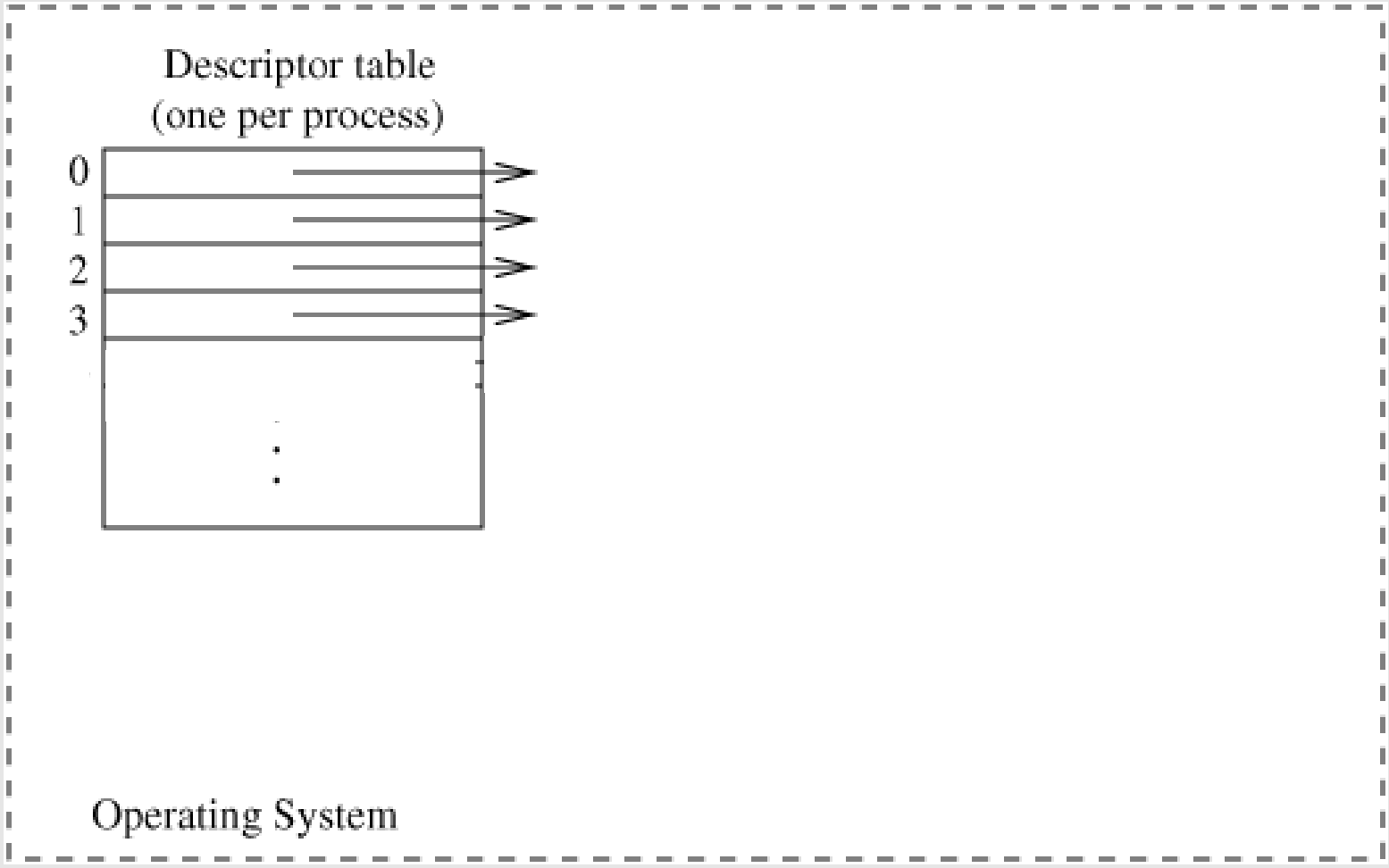
Creating A Socket

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

- Parameters
  - domain: PF_INET
  - type: SOCK_DGRAM, SOCK_STREAM, SOCK_RAW
  - protocol: usually = 0 (i.e., default for type)
- Example

```
#include <sys/types.h>
#include <sys/socket.h>
…
if ((s = socket(PF_INET, SOCK_STREAM, 0) < 0)
      perror("socket");
```

After Creating A Socket



Generic Address Structure

- Each protocol family defines its own address representation
- For each protocol family there is a corresponding address family
  - (e.g., PF_INET → AF_INET, PF_UNIX → AF_UNIX)
- Generalized address format:
  - <address family, endpoint address in family>

```
struct  sockaddr {
    u_char  sa_len;       /* total length */
    u_short sa_family;    /* type of address  */
    char    sa_data[14]; /* value of address */
};
```

Binding the Local Address

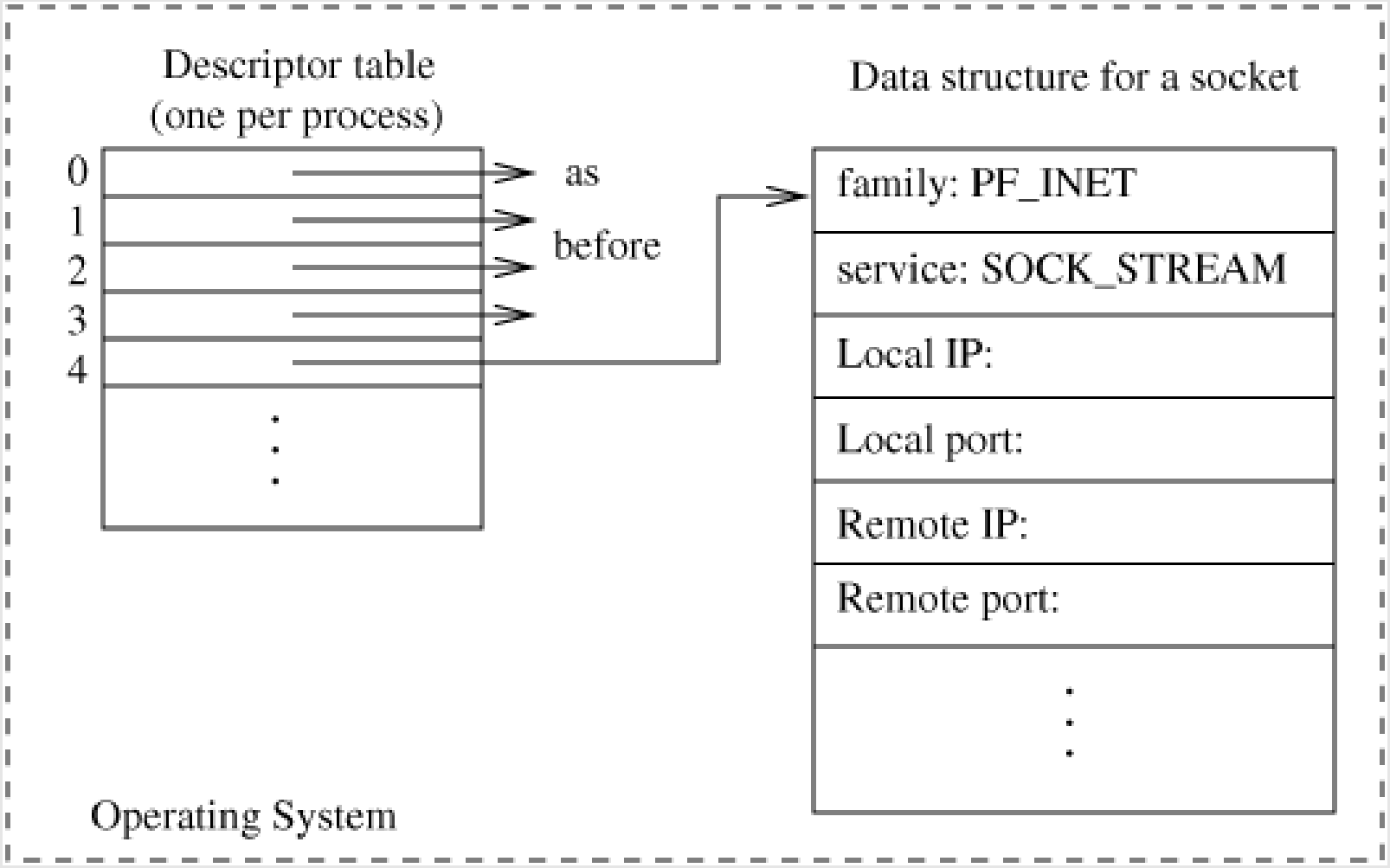
```
int  bind(int s, struct sockaddr *addr, int addrlen);
```

- Used primarily by servers to specify their well-known port
- Optional for clients
  - normally, system chooses a “random” local port
- Use INADDR_ANY to allow the socket to receive datagrams sent to *any* of the machine's IP addresses

Binding the Local Address (cont'd)

```
…
sin.sin_family      = AF_INET;
sin.sin_port        = htons(6000); /* if 0:system
  chooses */
sin.sin_addr.s_addr = INADDR_ANY;  /* allow any
  interface */
if (bind(s, (struct sockaddr *)&sin, sizeof(sin)) < 0)
```

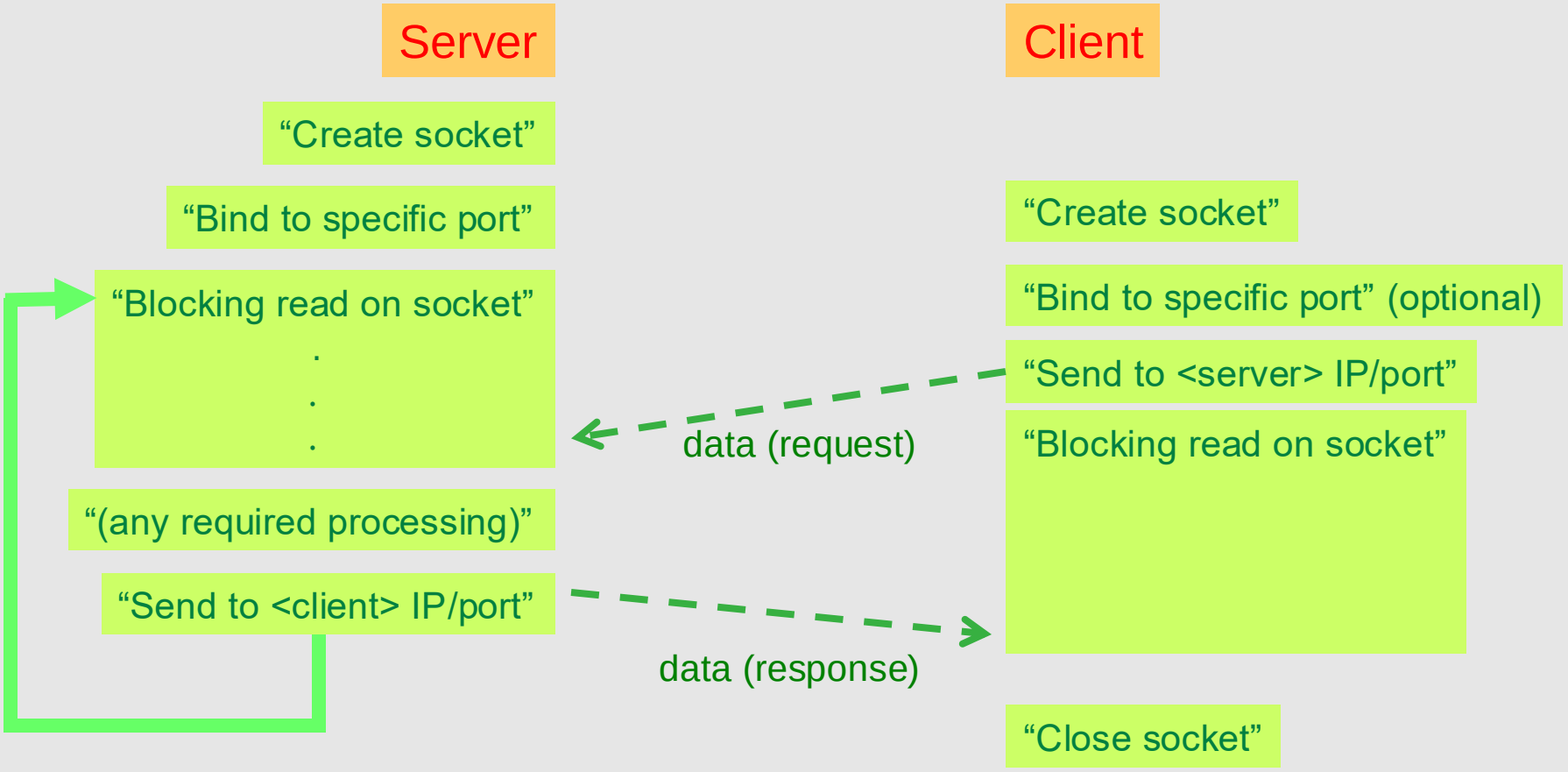
After Binding the Local Address



UDP Sockets – Logical Interaction

	Connectionless	**Connection-Oriented**
Iterative (one-at-a-time)	Iterative Connectionless (*normal* UDP)	Iterative Connection-Oriented (*less common*)
Con-current	Concurrent Connectionless (*uncommon*)	Concurrent Connection-Oriented (*normal* TCP)

UDP Sockets – Logical Interaction



Sample UDP Client (C)

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

int main(int argc, char**argv)
{
   int sockfd,n;
   struct sockaddr_in servaddr,cliaddr;
   char sendline[1000];
   char recvline[1000];

   if (argc != 2)
   {
      printf("usage:  udp-client <IP address>\n");
      exit(1);
   }

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

Create address data structure

Create space for input and received strings

"Create socket"

Sample UDP Client (C)

Initialize address data structure

"Send to IP/port"

"Receive datagram from socket"

```
bzero(&servaddr,sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr=inet_addr(argv[1]);
servaddr.sin_port=htons(32000);

while (fgets(sendline, 10000,stdin) != NULL)
{
   sendto(sockfd,sendline,strlen(sendline),0,
          (struct sockaddr *)&servaddr,sizeof(servaddr));
   n=recvfrom(sockfd,recvline,10000,0,NULL,NULL);
   recvline[n]=0;
   fputs(recvline,stdout);
}
}
```

Sample UDP Server (C)

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

int main(int argc, char**argv)
{
   int sockfd,n;
   struct sockaddr_in servaddr,cliaddr;
   socklen_t len;
   char mesg[1000];

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

Create address data structure

"Create socket"

Sample UDP Server (C)

Initialize address data structure

"Bind to port"

"Receive datagram from socket"

"Send to IP/port"

```
bzero(&servaddr,sizeof(servaddr));
  servaddr.sin_family = AF_INET;
  servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
  servaddr.sin_port=htons(32000);
  bind(sockfd,(struct sockaddr *)&servaddr,sizeof(servaddr));

  for (;;)
  {
    len = sizeof(cliaddr);
    n = recvfrom(sockfd,mesg,1000,0,(struct sockaddr *)&cliaddr,&len);
    sendto(sockfd,mesg,n,0,(struct sockaddr *)&cliaddr,sizeof(cliaddr));
    printf("-------------------------------------------------------\n");
    mesg[n] = 0;
    printf("Received the following:\n");
    printf("%s",mesg);
    printf("-------------------------------------------------------\n");
  }
}
```

TCP Server Socket behavior

	Connectionless	**Connection-Oriented**
Iterative (one-at-a-time)	Iterative Connectionless (*normal* UDP)	Iterative Connection-Oriented (*less common*)
Con-current	Concurrent Connectionless (*uncommon*)	Concurrent Connection-Oriented (*normal* TCP)

TCP Server Socket behavior

	Connectionless	**Connection-Oriented**
Iterative (one-at-a-time)	Iterative Connectionless (*normal* UDP)	Iterative Connection-Oriented (*less common*)
Con-current	Concurrent Connectionless (*uncommon*)	Concurrent Connection-Oriented (*normal* TCP)

TCP Server Socket behavior

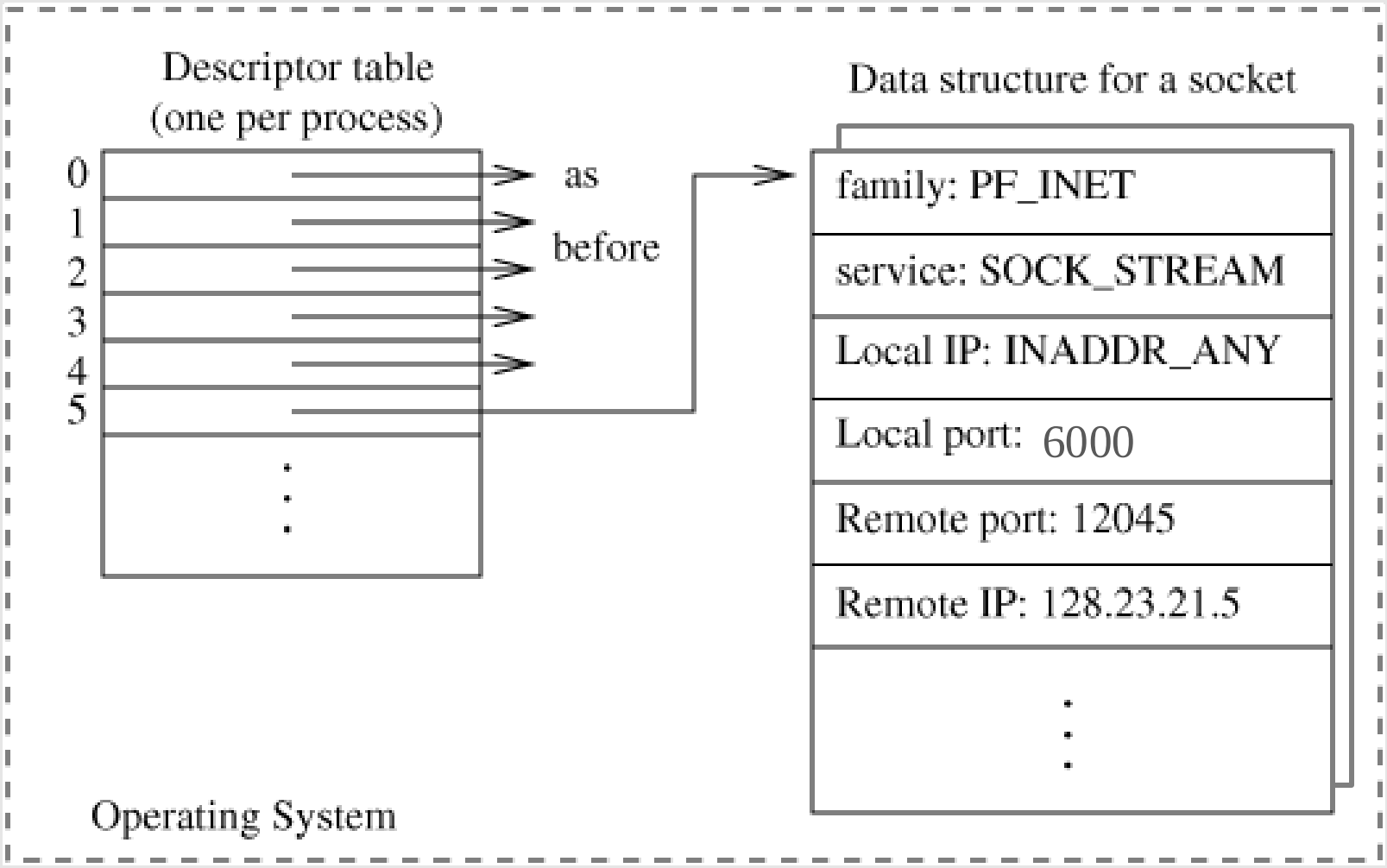
- TCP is geared for servers wanting to operate concurrently as well as with connection orientation
- Adds a new semantic
  - Existing: receive data on socket (block or check)
  - New: listen for and accept incoming connections
- How should the new semantic behave?
  - Server expects *connected* socket after call
    - Save connection far-end (client) information in socket before call returns
  - Server expects to be able to spawn thread or process for new connection
    - *Clone* existing socket, save connection information in new one
    - Original one continues as “listening” socket

TCP Server Socket behavior

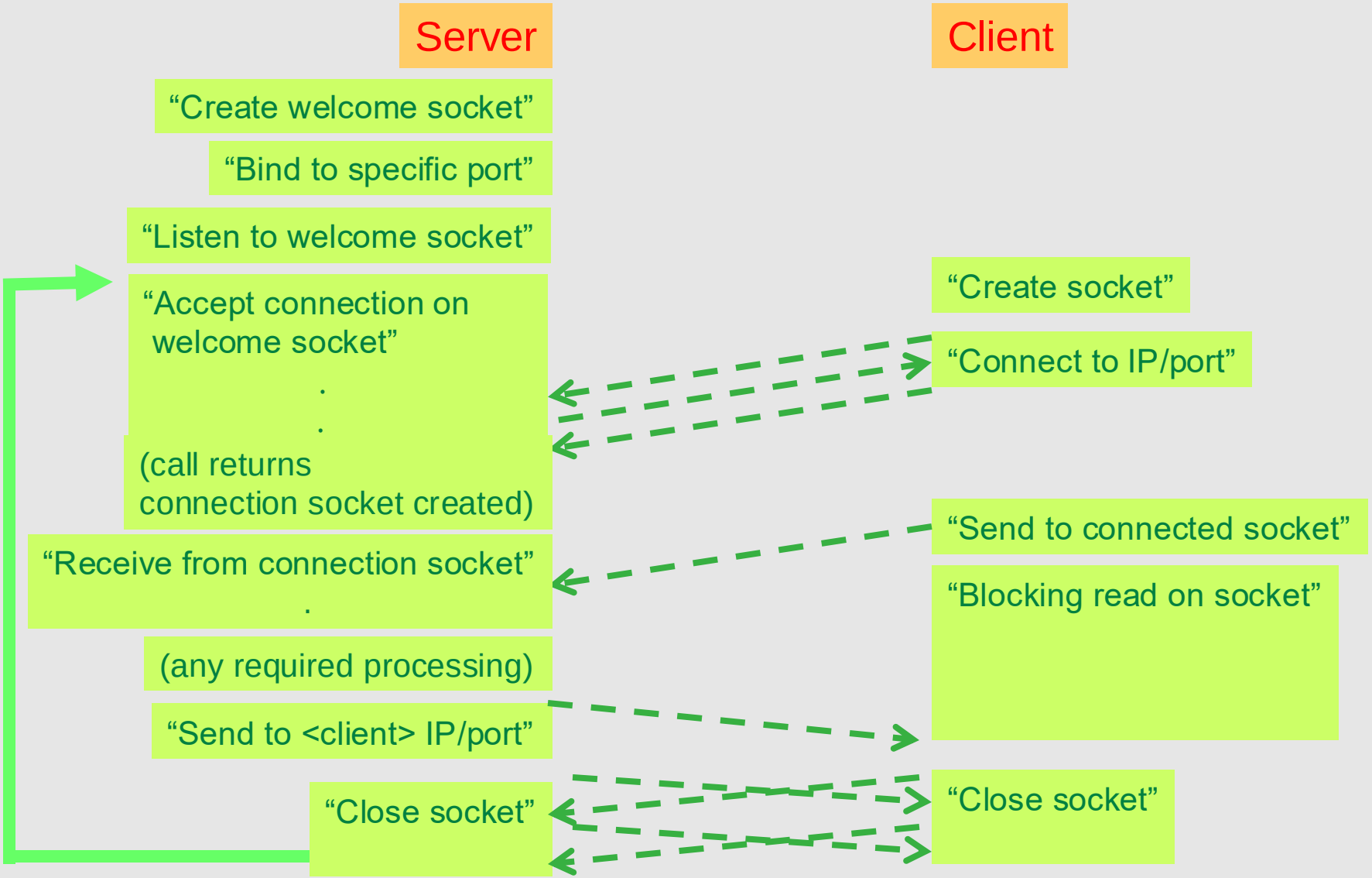
"Create socket" → creates and returns socket

"Listen for connection" → creates and returns socket

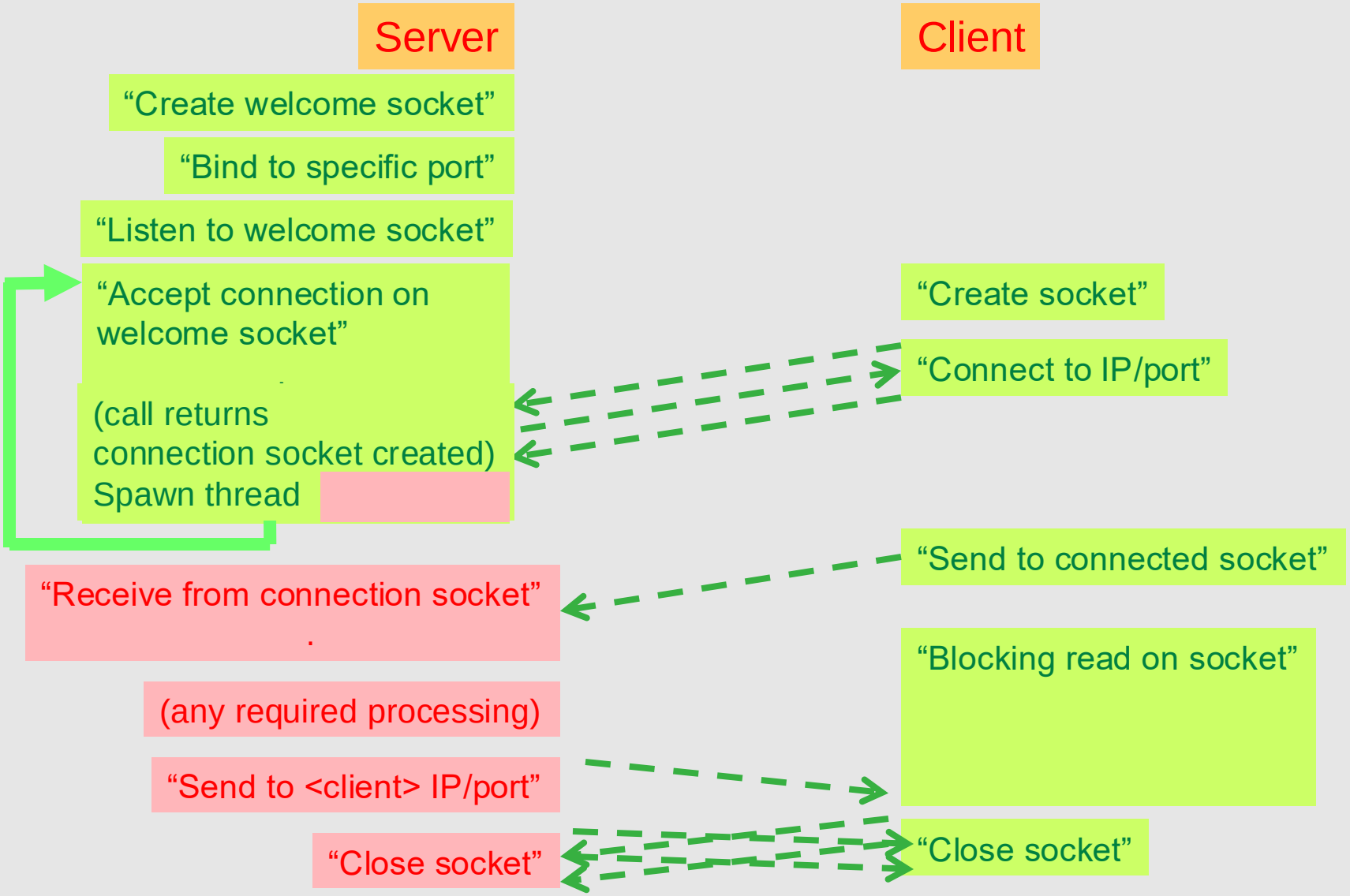
TCP Listen as Socket Data Structure



TCP Sockets – Logical Interaction



TCP Sockets – Concurrent Server



Sample TCP Client (C)

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

int main(int argc, char *argv[])
{
    int sockfd, portno, n;

    struct sockaddr_in serv_addr;
    struct hostent *server;

    char buffer[256];
    if (argc < 3) {
       fprintf(stderr,"usage %s hostname port\n", argv[0]);
       exit(0);
    }
    portno = atoi(argv[2]);
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
        error("ERROR opening socket");
    server = gethostbyname(argv[1]);
```

Create address data structure

Create space for input and received strings

"Create socket"

Sample TCP Client (C)

Initialize address data structure

"Connect to IP/port"

Read input string

"Write data to socket"

"Read data from socket"

```
if (server == NULL) {
    fprintf(stderr,"ERROR, no such host\n");
    exit(0);
}
bzero((char *) &serv_addr, sizeof(serv_addr));
serv_addr.sin_family = AF_INET;
bcopy((char *)server->h_addr,
     (char *)&serv_addr.sin_addr.s_addr,
     server->h_length);
serv_addr.sin_port = htons(portno);
if (connect(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr)) < 0)
    error("ERROR connecting");
printf("Please enter the message: ");
bzero(buffer,256);
fgets(buffer,255,stdin);
n = write(sockfd,buffer,strlen(buffer));
if (n < 0)
     error("ERROR writing to socket");
bzero(buffer,256);
n = read(sockfd,buffer,255);
if (n < 0)
     error("ERROR reading from socket");
printf("%s\n",buffer);
return 0;
}
```

Sample TCP Server (C) Fragment

"Create socket"

```
sd = socket (PF_INET, SOCK_STREAM, ptrp->p_proto);
 if (sd < 0) {
                fprintf(stderr, "socket creation failed\n");
                exit(1);
}
```

"Bind to port"

```
if (bind(sd, (struct sockaddr *)&sad, sizeof (sad)) < 0) {
                fprintf(stderr,"bind failed\n");
                exit(1);
}
```

"Listen to socket"

```
if (listen(sd, QLEN) < 0) {
                fprintf(stderr,"listen failed\n");
                 exit(1);
}
alen = sizeof(cad);
fprintf( stderr, "Server up and running.\n");
```

"Accept connection"

```
while (1) {
    printf("SERVER: Waiting for contact ...\n");
    if (  (sd2=accept(sd, (struct sockaddr *)&cad, &alen)) < 0) {
                fprintf(stderr, "accept failed\n");
                exit (1);
    }
```

Hand off connected socket

```
    pthread_create(&tid, NULL, serverthread, (void *) sd2 );
}
close(sd);
```

Sample TCP Server (C) Fragment

```
void * serverthread(void * parm)
{
   int tsd, tvisits;
   char     buf[100];          /* buffer for string the server sends */

   tsd = (int) parm;

   pthread_mutex_lock(&mut);
       tvisits = ++visits;
   pthread_mutex_unlock(&mut);

   sprintf(buf,"This server has been contacted %d time%s\n",
          tvisits, tvisits==1?".":"s.");

   printf("SERVER thread: %s", buf);
   send(tsd,buf,strlen(buf),0);
   close(tsd);
   pthread_exit(0);
}
```

Thread concurrency management

"Send to connected socket"

APDU Delineation

- TCP is stream-oriented
- How does application know what units peer wants sent data to be processed in?
  - Hardcode APDU size
    - Not very flexible (different websites)
  - First just send APDU size (in hardcoded size, need syn)
  - Hardcode APDU termination or other structural characteristics
- Requires either:
  - Reading data byte-by-byte
  - Peeking ahead to read upto a point
  - Putting data back on socket

Programmer Controls APDU

- Use of software libraries can produce additional “encapsulations” implicitly
  - For example the use of “BufferedReader” in sample code
- When working cross-platform, programmer must be sure of what bytes get exchanged
  - Must know socket syntax and semantics in legacy systems and platforms
  - Can benefit from Wireshark or similar traffic analyzer