

Redes de Comunicação 2024/2025

TP04

Programming with sockets and UDP

Jorge Granjal
University of Coimbra



TP04: Programming with sockets and UDP

Overview:

- Multiplexing/demultiplexing
- Socket programming (UDP)
- Socket programming in C (functions)
- Client-Server implementation structure

Multiplexing/demultiplexing

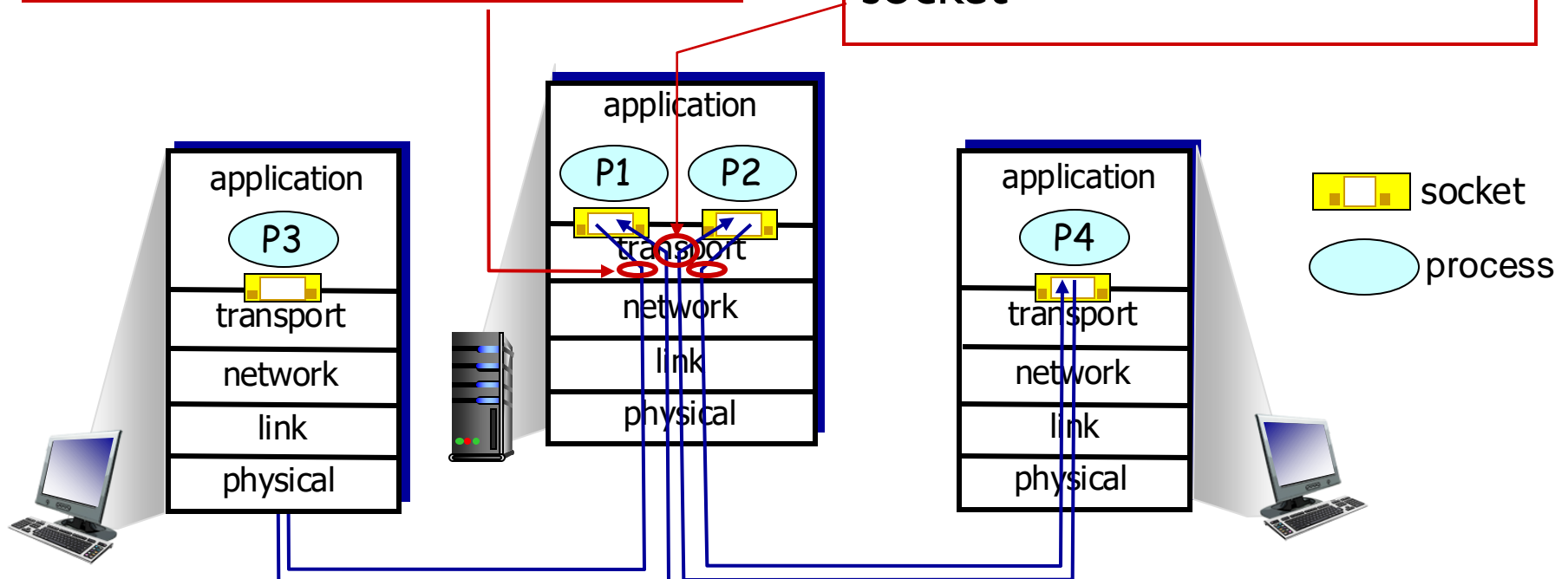
A process (which is part of an application) can have one or more sockets, “doors” through which they exchange data with the network

multiplexing at sender:

handle data from multiple sockets, add transport header (later used for demultiplexing)

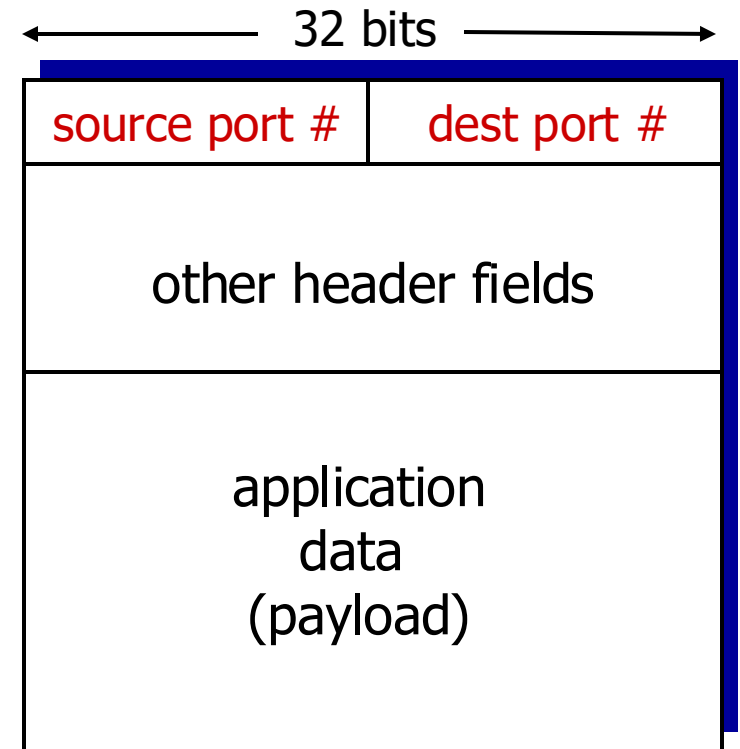
demultiplexing at receiver:

use header info to deliver received segments to correct socket



How demultiplexing works

- host receives IP datagrams
 - each datagram has source IP address, destination IP address
 - each datagram carries one transport-layer segment
 - each segment has source, destination port number
- host uses *IP addresses & port numbers* to direct segment to appropriate socket (and process)



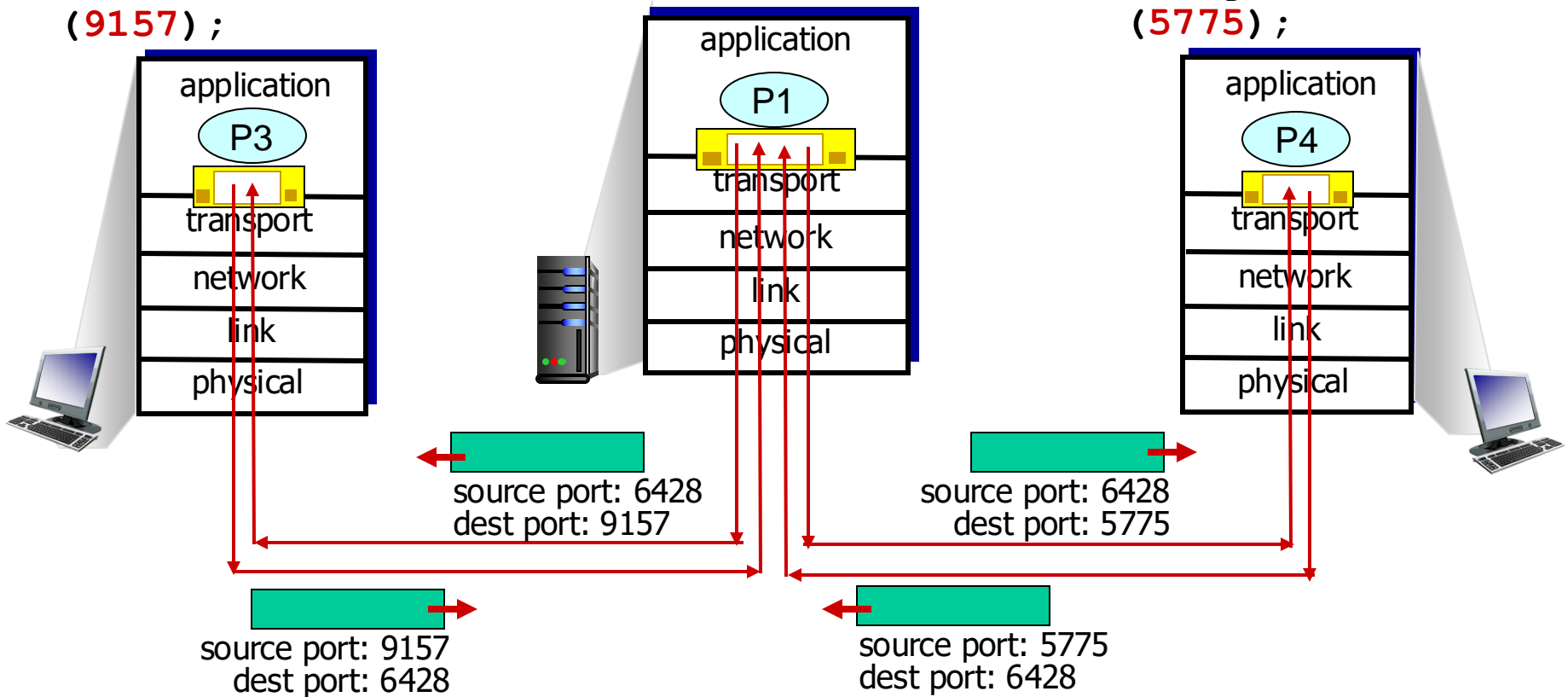
TCP/UDP segment format

Connectionless demux: example

```
DatagramSocket  
mySocket2 = new  
DatagramSocket  
(9157);
```

```
DatagramSocket  
serverSocket = new  
DatagramSocket  
(6428);
```

```
DatagramSocket  
mySocket1 = new  
DatagramSocket  
(5775);
```

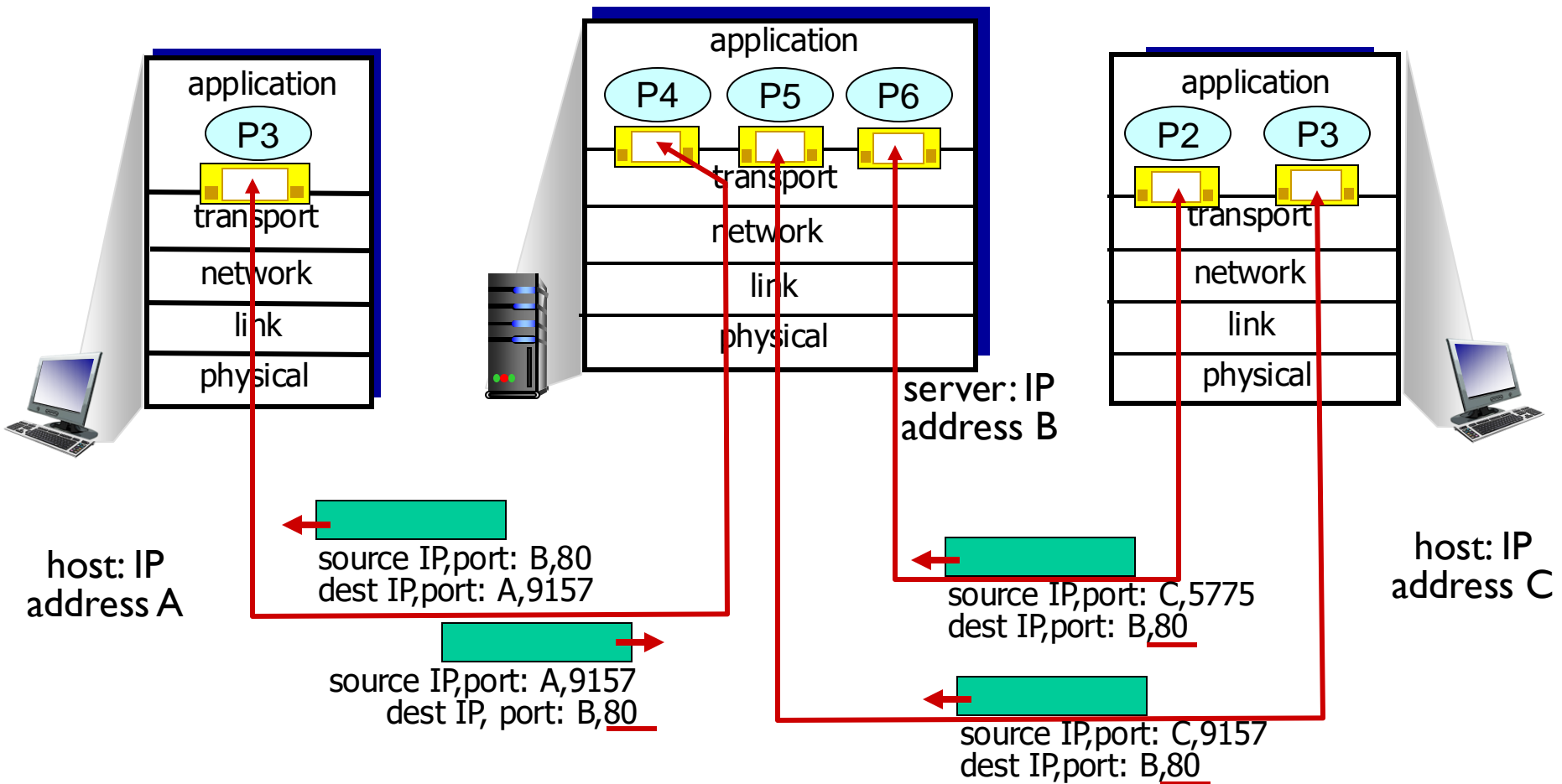


In UDP, the socket API allows one socket to receive from many endpoints, and to send to many endpoints

Connection-oriented demux

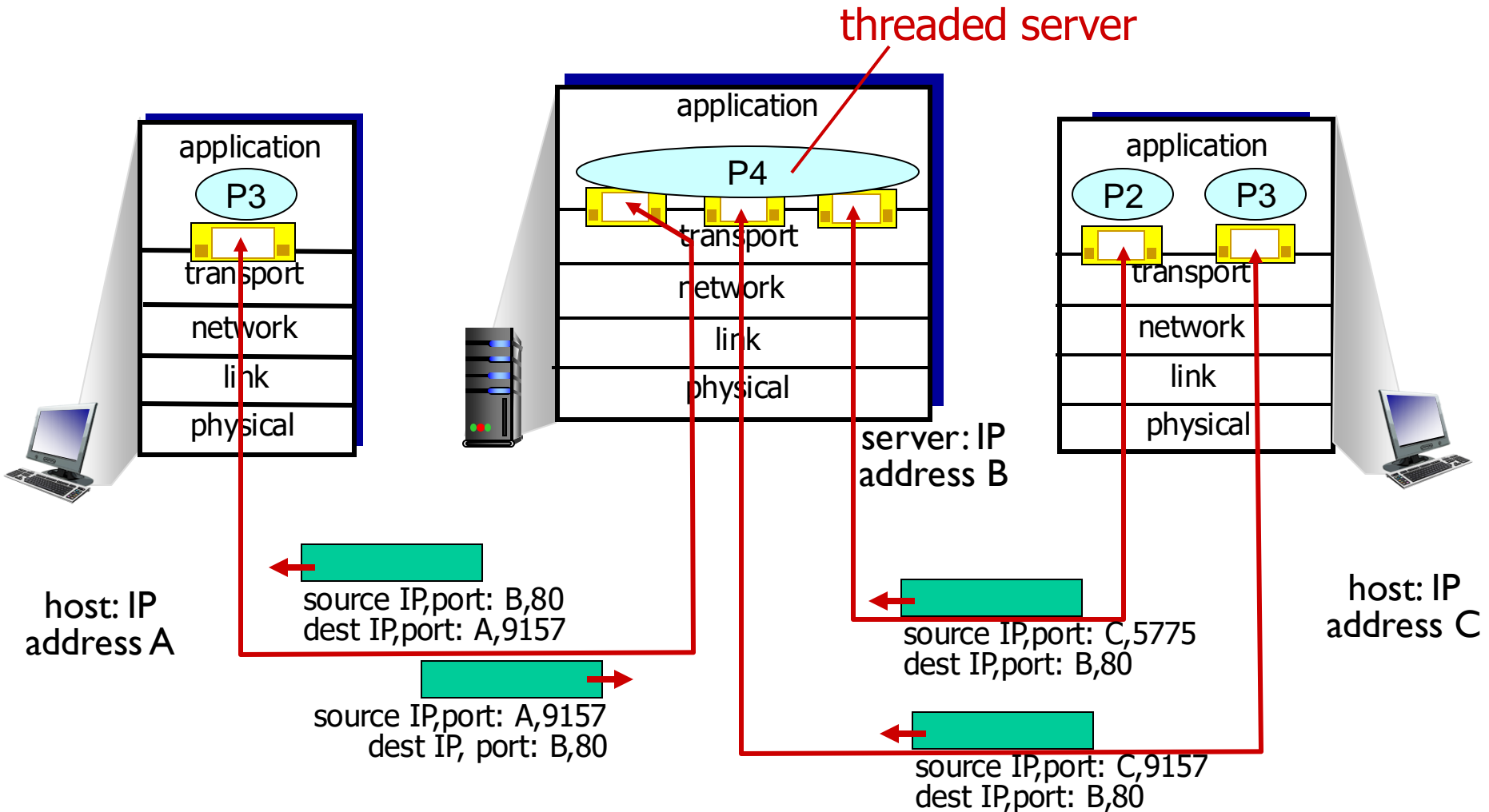
- UDP socket identified by 2-tuple:
 - dest IP address
 - dest port number
- TCP socket identified by 4-tuple:
 - source IP address
 - source port number
 - dest IP address
 - dest port number
- demux: receiver uses all four values to direct segment to appropriate socket
- server host may support many simultaneous TCP sockets:
 - each socket identified by its own 4-tuple
- web servers have different sockets for each connecting client
 - example: non-persistent HTTP will have different socket for each request

Connection-oriented demux: example



three segments, all destined to IP address: B,
dest port: 80 are demultiplexed to *different* sockets

Connection-oriented demux: example



In TCP, the socket API maps one TCP connection to one socket, which is between two endpoints (src IP+port, dst IP+port)

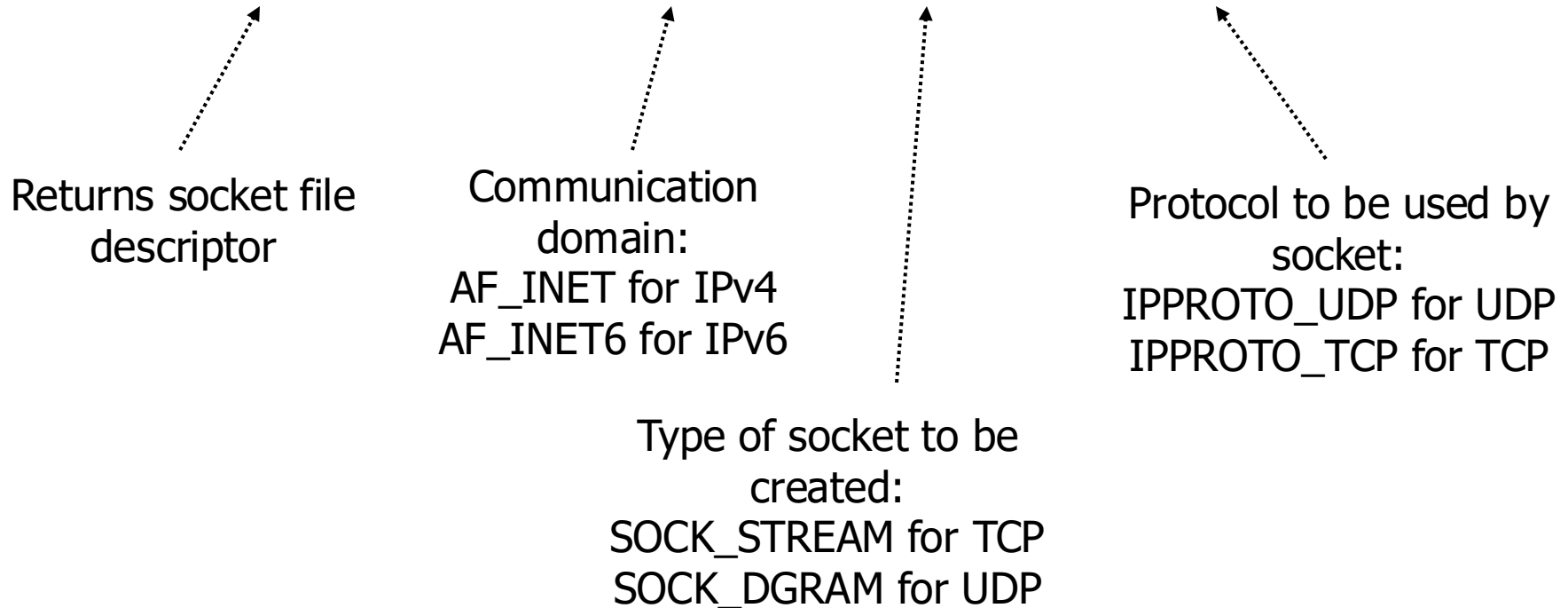
Socket programming (UDP)

- UDP is connectionless
- The client does not form a connection with the server like in TCP, instead just sends a datagram to the server
- The server need not accept a connection and just waits for datagrams to arrive
- Datagrams upon arrival contain the address and port of sender, which the server may use to reply

Socket programming in C (functions)

- The application invokes the **socket** function to create an UDP or TCP socket

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



Socket programming in C (functions)

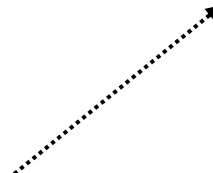
- The **bind** function is used by the server to assign an address to the unbound socket

int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen)

File descriptor of
socket to be
binded



Structure in which
address to be
binded to is
specified



Size of addr structure




Socket address structures

- Most socket functions require a pointer to a socket address structure
- The names of these structures begin with *sockaddr_* and end with a unique suffix for each protocol suite

```
#include <arpa/inet.h>
```

```
struct in_addr {  
    in_addr_t  s_addr;  
  
};
```

Generic structure (used in declarations,
to deal with address structures from any
supported protocol families)

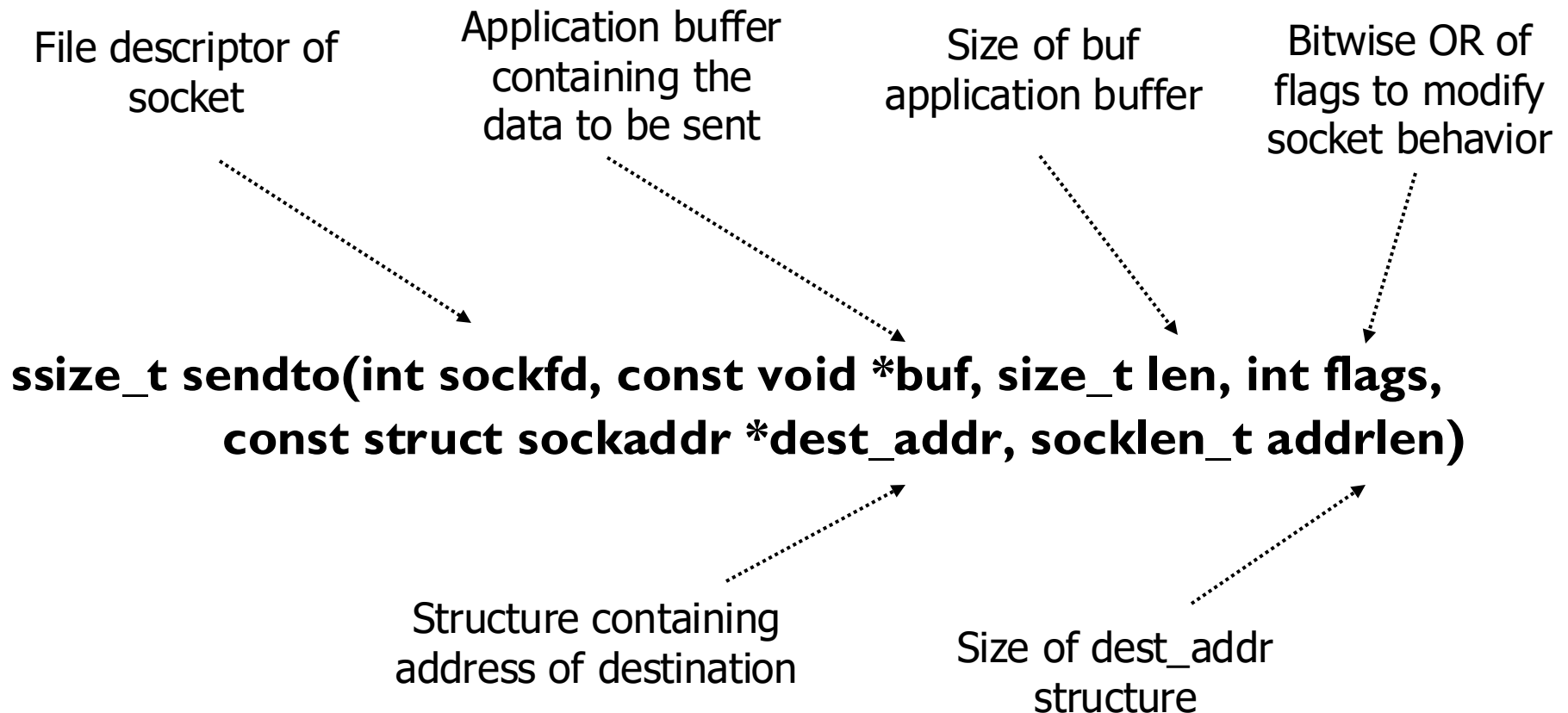


```
/* 32-bit IPv4 address */  
/* network byte ordered */
```

```
struct sockaddr_in {  
    uint8_t      sin_len;           /* length of structure (16) */  
    sa_family_t  sin_family;      /* AF_INET */  
    in_port_t    sin_port;        /* 16-bit TCP or UDP port number */  
                                     /* network byte ordered */  
    struct in_addr sin_addr;      /* 32-bit IPv4 address */  
                                     /* network byte ordered */  
    char         sin_zero[8];      /* unused */  
};
```

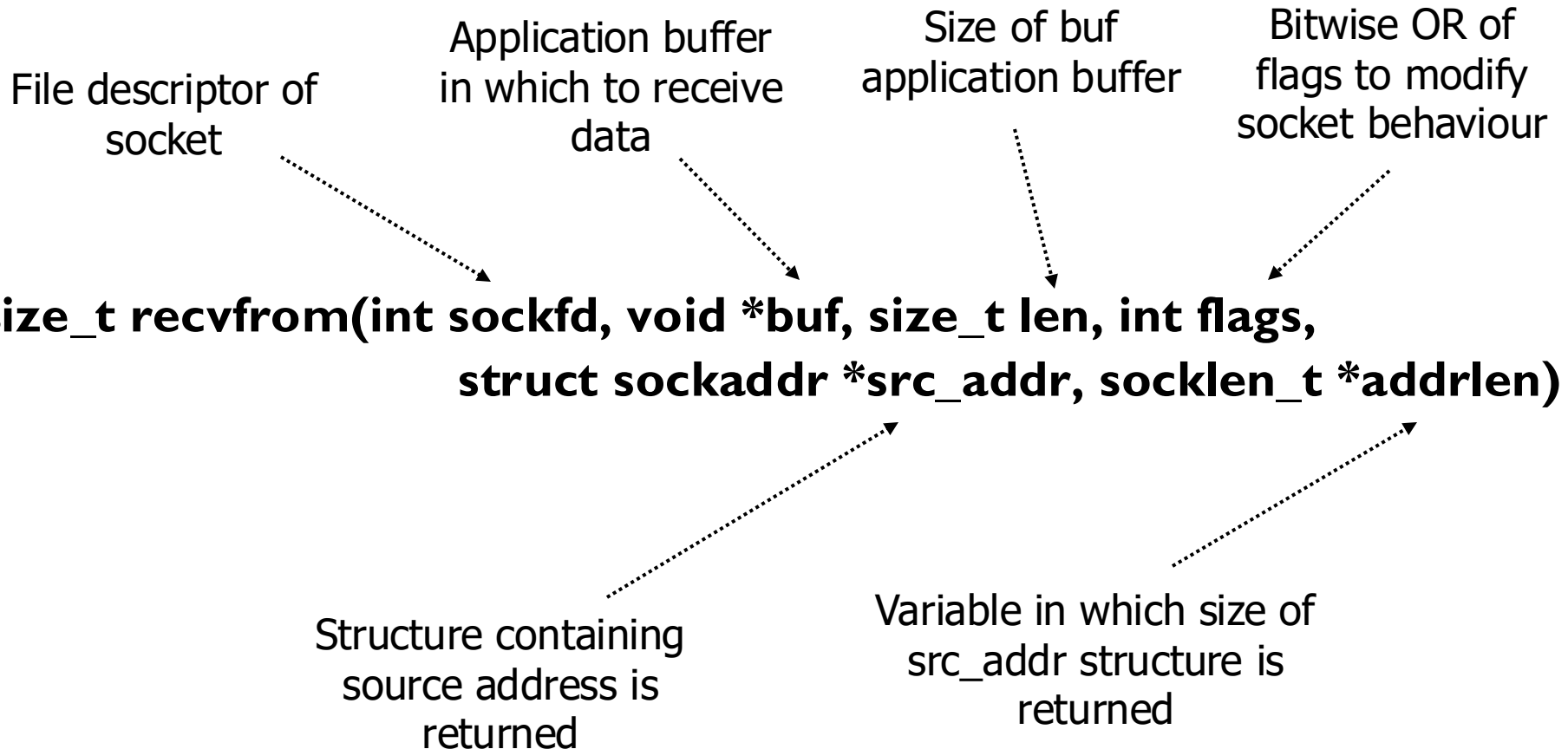
Socket programming in C (functions)

- The **sendto** function is used to send a message on the socket



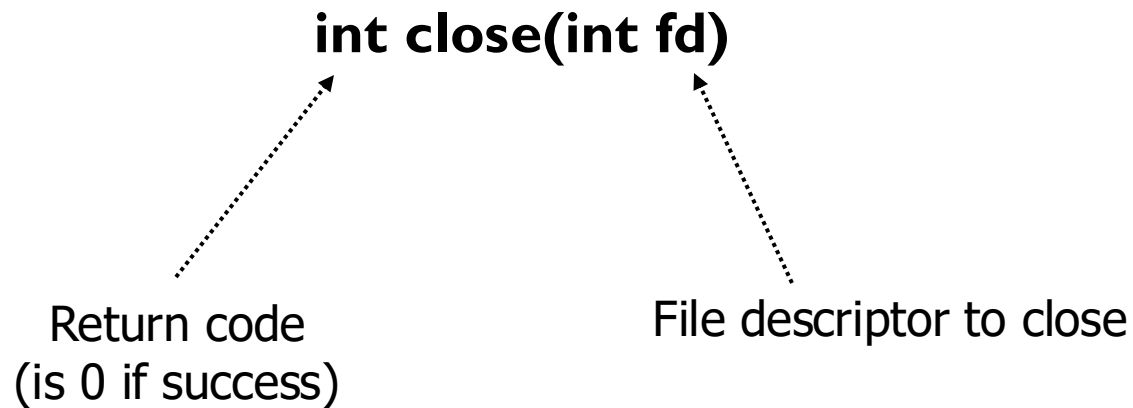
Socket programming in C (functions)

- The **recvfrom** function is used to receive a message from the socket



Socket programming in C (functions)

- The **close** function is used to close a file descriptor



Client-Server implementation

UDP Server:

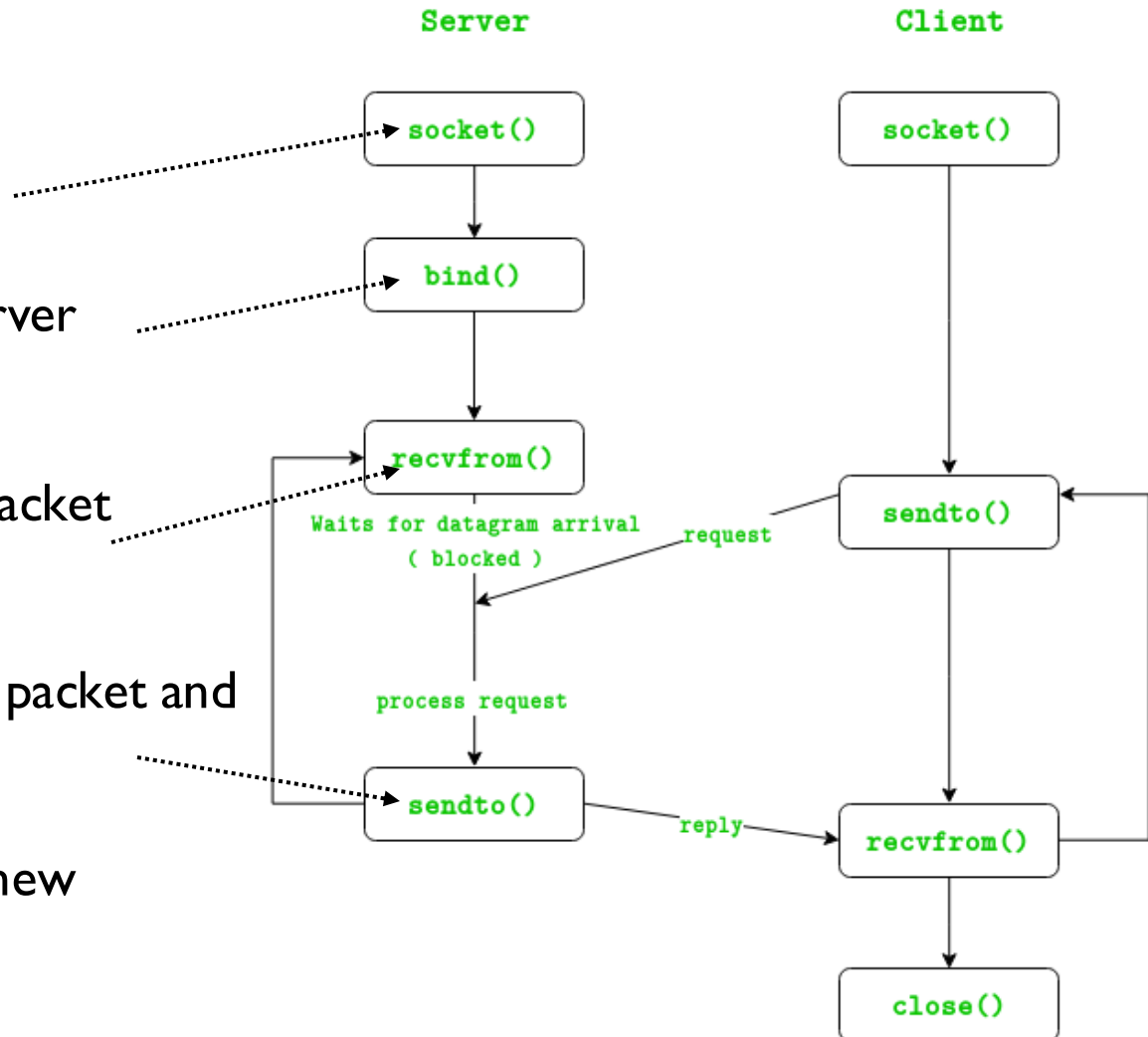
Create UDP socket

Bind the socket to server address

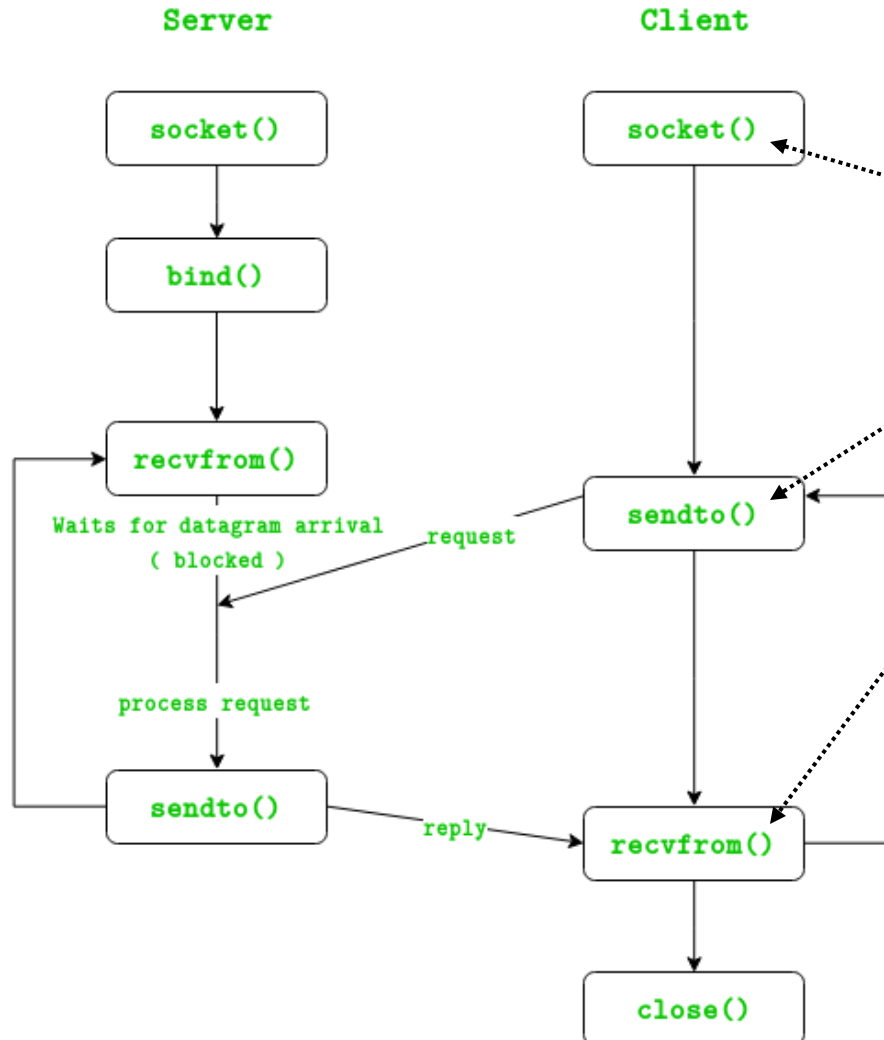
Wait until datagram packet arrives from client

Process the datagram packet and send a reply to client

Go back to Wait for new datagram



Client-Server implementation



UDP Client:

Create UDP socket

Sent message to server

Wait until response is received

Process reply and send it to server, if necessary, or..

Close the socket

TP04: Summary

What we have covered here?

- Multiplexing/demultiplexing
- Socket programming (UDP)
- Socket programming in C (functions)
- Client-Server implementation structure