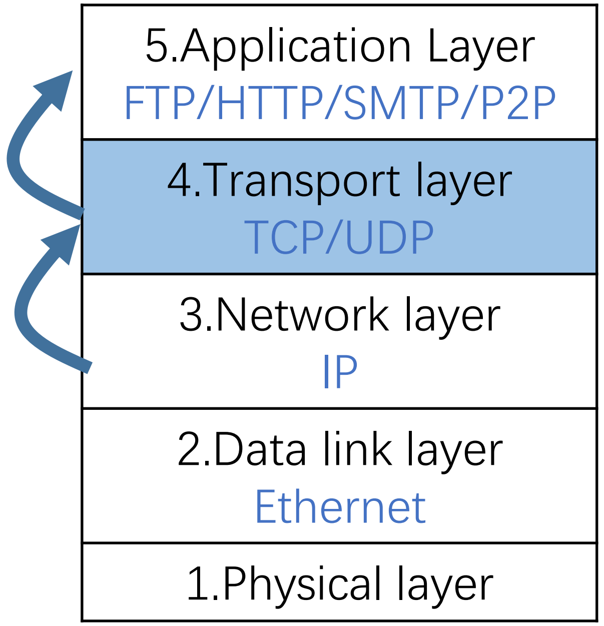
The Transfer Layer

First View and Socket Programming

School of Software Engineering
South China University of Technology
Dr. Chunhua Chen
chunhuachen@scut.edu.cn
2019 Spring

Recall the Layered Architecture

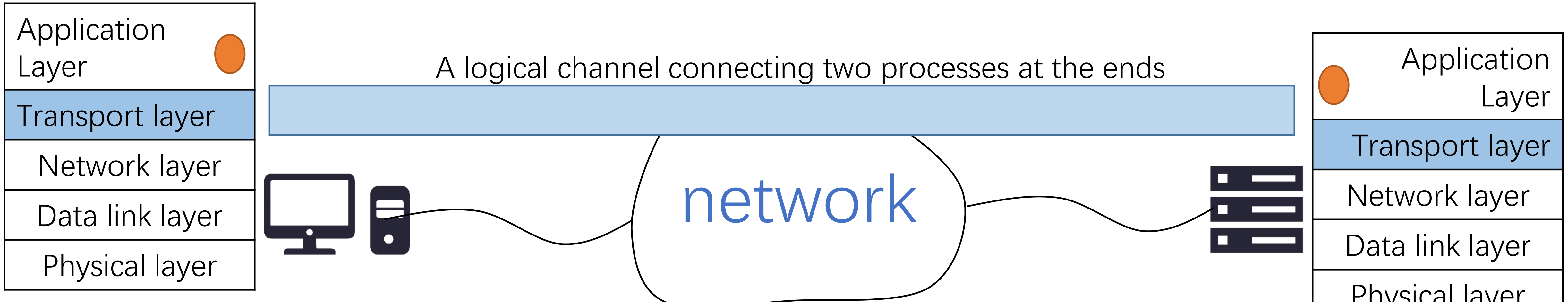
- Applications use data transfer services from the Transfer Layer
  - No Loss of data, e.g. Web/HTTP
  - Loss-tolerant, e.g. Streaming stored audio/video
- The Transfer Layer uses packet delivery services from the Network Layer
  - Note that the network infrastructure is not reliable!
  - Direct use of network
    - Best effort Service, means no guarantees
  - Building a reliable data channel on top of network
    - Guarantees: no loss of data, timing guarantee… and others



The logical Communication Channel Model

All is done at two ends, with no help from the network

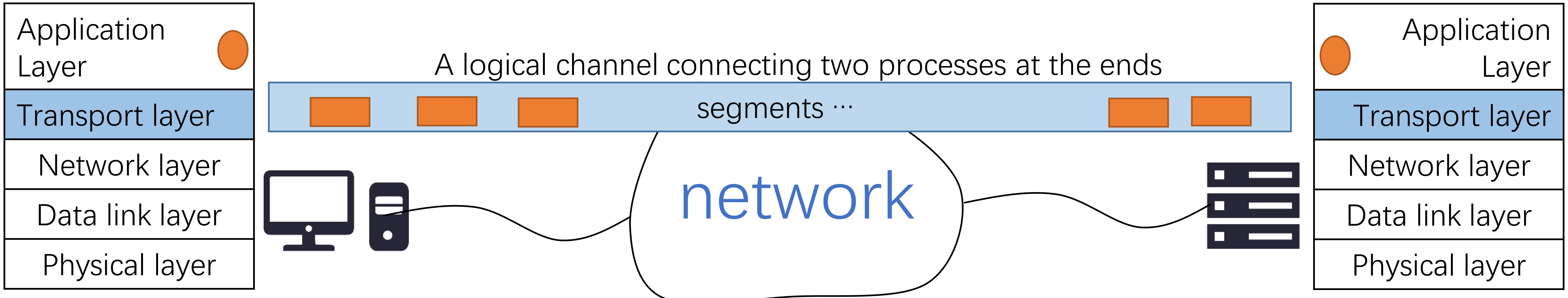
- Any two communicating hosts may be connected via numerous routers and a wide range of link types.
- The transfer layer extend the network service between two hosts to a data transfer service between two processes running on the end systems.
  - as if the hosts running the processes were directly connected, by a logical channel



The logical Communication Channel Model

All is done at two ends, with no help from the network

- **Logical**: Apps use the logical communication provided by the transport layer to send messages to each other, free from the worry of the details of the physical infrastructure used to carry these messages.
    - the sending break a message into chunks, and add a transfer layer header to each chunk creating a segment; then passes the segment to the network layer … (receiving side?)



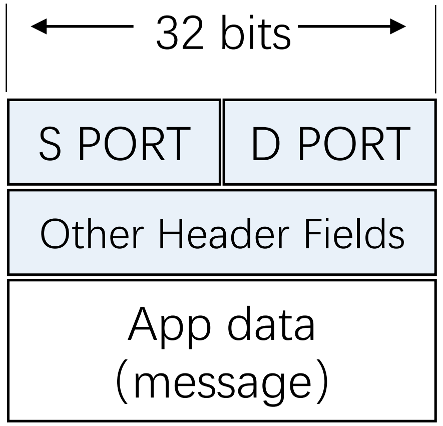
The Transfer Services of the Internet

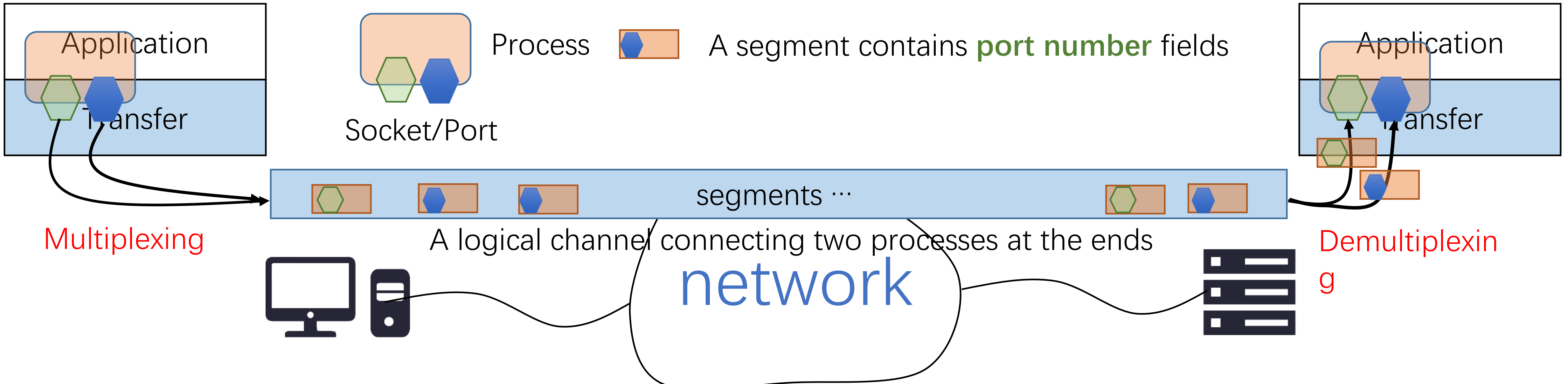
- About the Internet's network Layer
  - Protocol: the Internet Protocol, IP
  - Addressing hosts with 32bits IP address
  - The IP service: best-effort delivery service, which is unreliable!
- The transfer layer basic services (基础服务)
  - **UDP** (User Datagram Protocol) , unreliable, connectionless service
  - **TCP** (Transmission Control Protocol), reliable, connection-oriented service
- Flow control (流量控制)
  - Adapting to the receiving rate
- Congestion Control (拥塞控制)
  - For common network healthy

Multiplexing and Demultiplexing

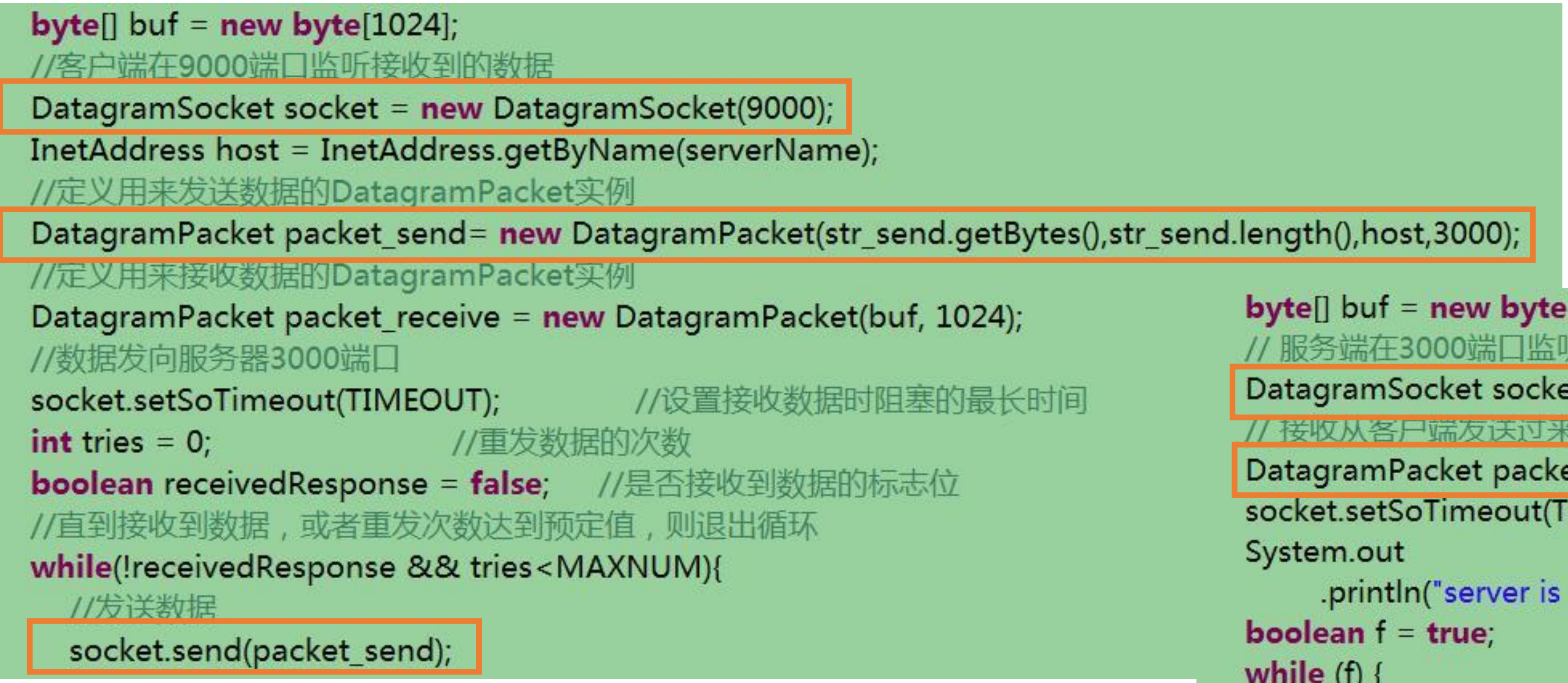
multiplexing is necessary when sharing

- Multiples processes sharing a logical channel between the two hosts
  - Recall TDM and FDM of sharing a link in Circuit Switching(?)
- **Socket Interface**, apps send message chunks through
  - Identifying of socket using is needed when sending and receiving





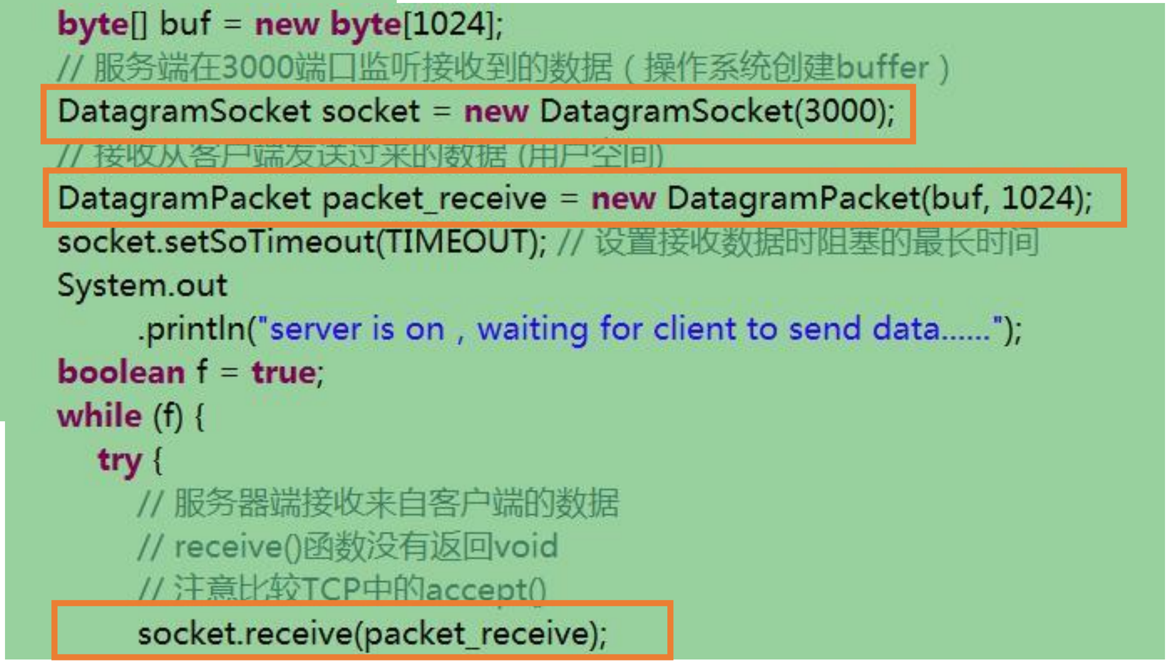
Socket Programming for UDP



```
byte[] buf = new byte[1024];
//客户端在9000端口监听接收到的数据
DatagramSocket socket = new DatagramSocket(9000);
InetAddress host = InetAddress.getByName(serverName);
//定义用来发送数据的DatagramPacket实例
DatagramPacket packet_send= new DatagramPacket(str_send.getBytes(),str_send.length(),host,3000);
//定义用来接收数据的DatagramPacket实例
DatagramPacket packet_receive = new DatagramPacket(buf, 1024);
//数据发向服务器3000端口
socket.setSoTimeout(TIMEOUT);           //设置接收数据时阻塞的最长时间
int tries = 0;                     //重发数据的次数
boolean receivedResponse = false;    //是否接收到数据的标志位
//直到接收到数据，或者重发次数达到预定值，则退出循环
while(!receivedResponse && tries<MAXNUM){
   //发送数据
   socket.send(packet_send);
```

Client

Server

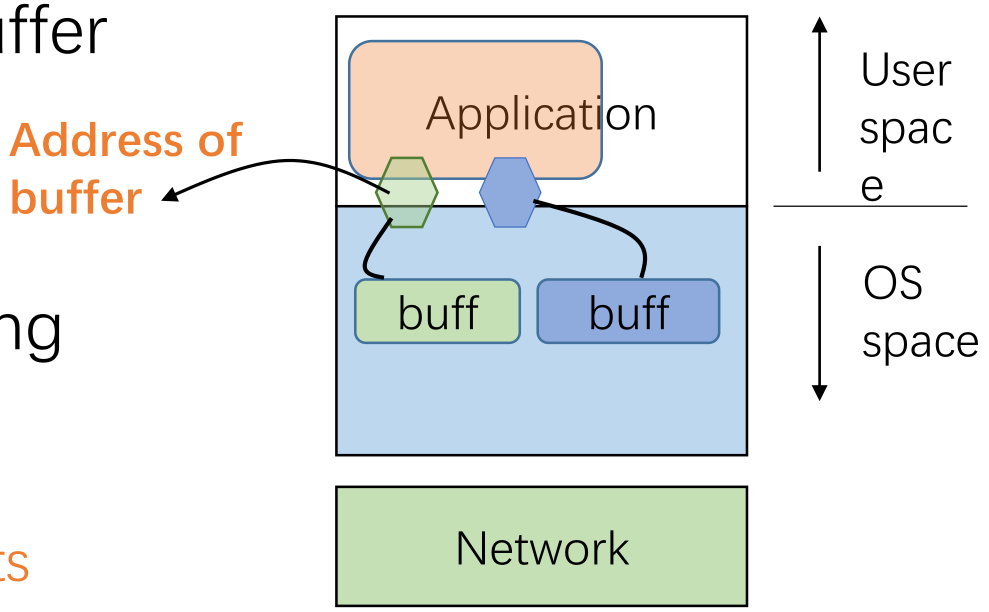


```
byte[] buf = new byte[1024];
// 服务端在3000端口监听接收到的数据（操作系统创建buffer）
DatagramSocket socket = new DatagramSocket(3000);
// 接收从客户端发送过来的数据 (用户空间)
DatagramPacket packet_receive = new DatagramPacket(buf, 1024);
socket.setSoTimeout(TIMEOUT); // 设置接收数据时阻塞的最长时间
System.out
      .println("server is on , waiting for client to send data......");
boolean f = true;
while (f) {
   try {
      // 服务器端接收来自客户端的数据
      // receive()函数没有返回void
      // 注意比较TCP中的accept()
      socket.receive(packet_receive);
```

Sockets from the OS Perspective

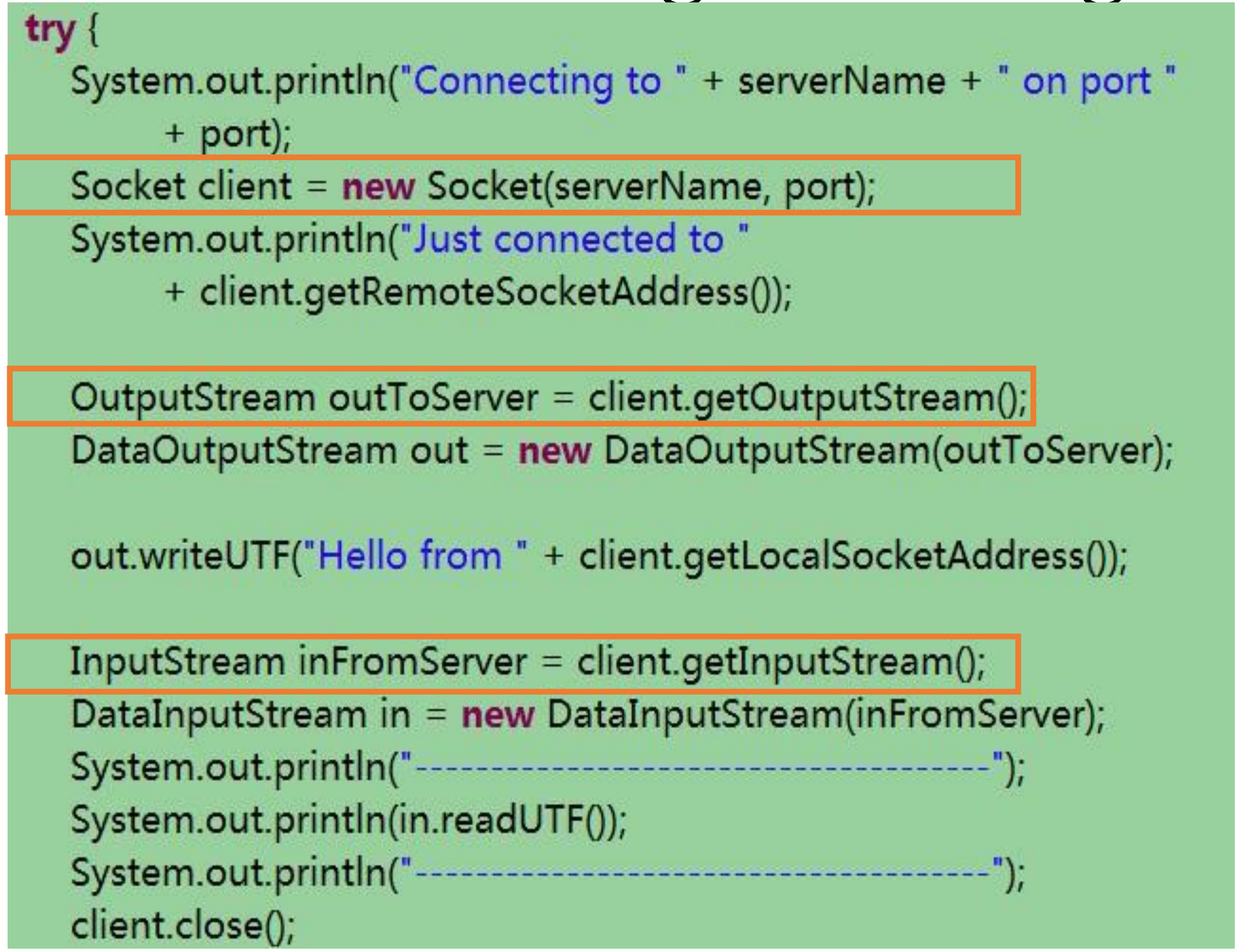
it is the receiving buffer matters

- Opening a socket means OS allocates a buffer for receiving data, and returns the buffer address to the opening process.
    - So there is socket-OS buffer mapping!
- Receiving data from a socket means copying data from the OS buffer to an App buffer.
- Connectionless sockets (e.g. UDP)
    - Server sockets are ready for receiving segments carrying app messages after opened
        - Server OS: new buffer for app messages
    - Clients sockets send app messages without informing the server (i.e. make no connection before sending)



Identifying a socket, is in fact addressing its corresponding buffer!

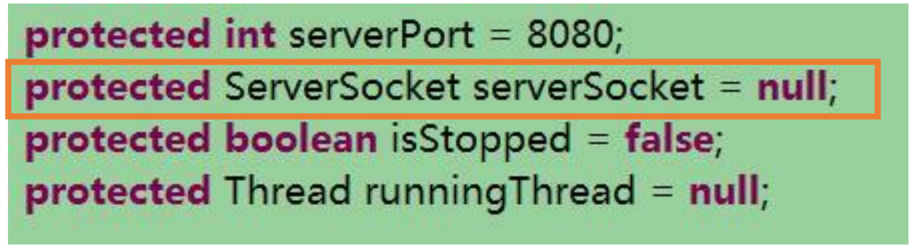
Socket Programming for TCP



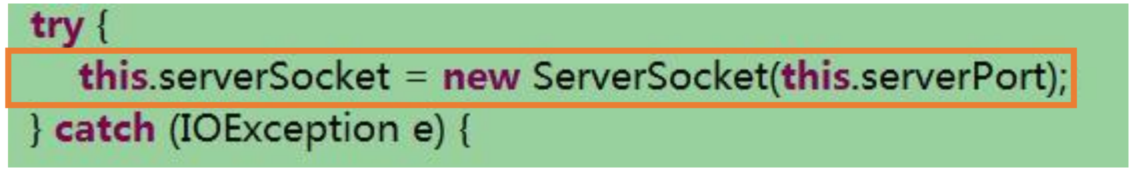
```
try {
    System.out.println("Connecting to " + serverName + " on port "
            + port);
    Socket client = new Socket(serverName, port);
    System.out.println("Just connected to "
            + client.getRemoteSocketAddress());

    OutputStream outToServer = client.getOutputStream();
    DataOutputStream out = new DataOutputStream(outToServer);

    out.writeUTF("Hello from " + client.getLocalSocketAddress());

    InputStream inFromServer = client.getInputStream();
    DataInputStream in = new DataInputStream(inFromServer);
    System.out.println("----------------------------------------");
    System.out.println(in.readUTF());
    System.out.println("----------------------------------------");
    client.close();
```

Client

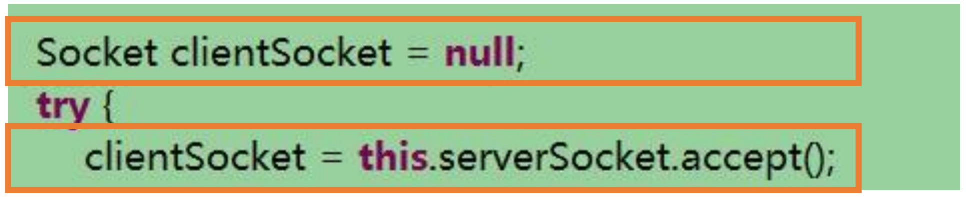
Server



```
protected int serverPort = 8080;
protected ServerSocket serverSocket = null;
protected boolean isStopped = false;
protected Thread runningThread = null;
```



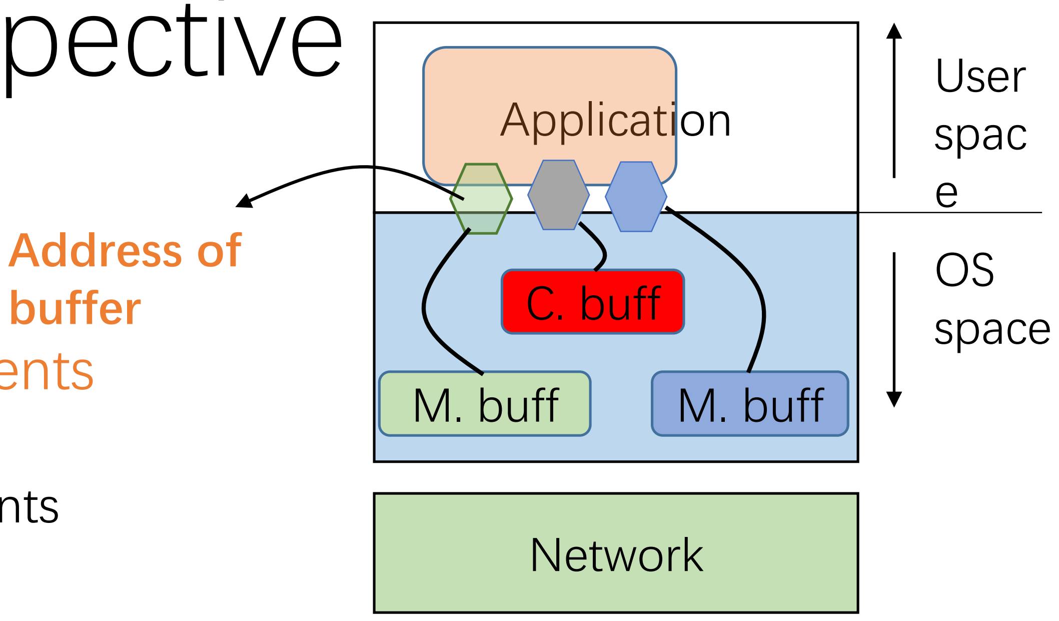
```
try {
    this.serverSocket = new ServerSocket(this.serverPort);
} catch (IOException e) {
```



```
Socket clientSocket = null;
try {
    clientSocket = this.serverSocket.accept();
```

Sockets from the OS Perspective

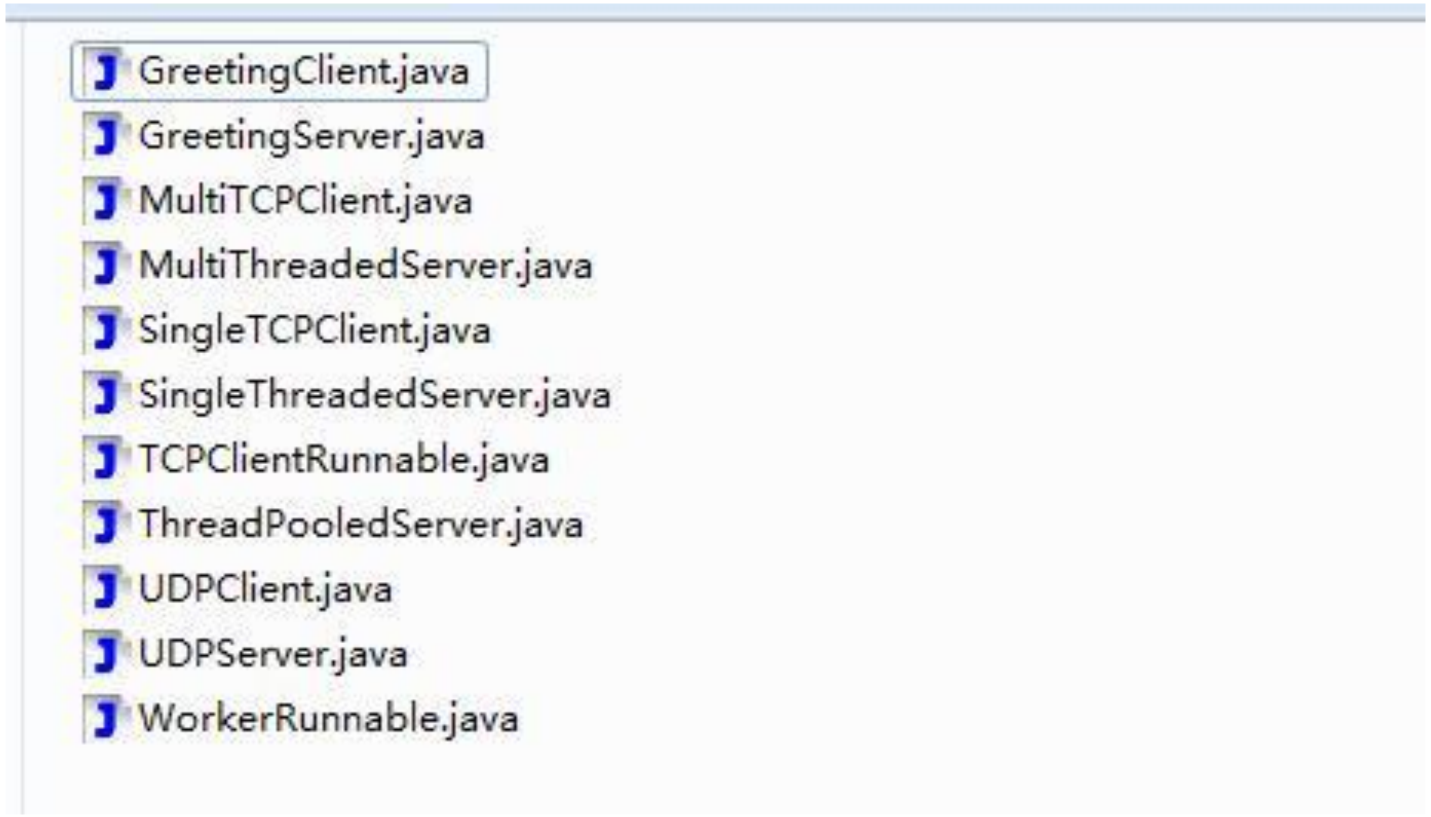
more about establishing of a connection



- Connection-oriented sockets
  - Server sockets are ready for receiving segments carry connection messages after opened
    - Server OS: new a buffer for connection segments
  - Client sockets send connection segments to establish a connection
    - Server OS: new a **connection socket**, in fact, new a buffer for coming app messages
  - Client sockets send app messages after connection would be put in the buffer of the connection socket.
- A client process can make multiple connection to the same server socket!

A connection-oriented socket has multiple buffers!!

Test Socket Programs



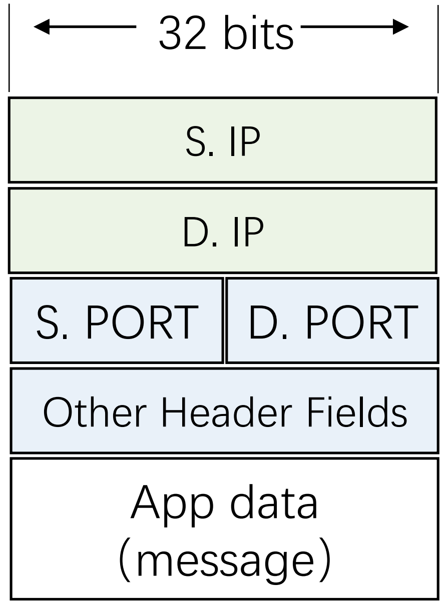
GreetingClient.java
GreetingServer.java
MultiTCPClient.java
MultiThreadedServer.java
SingleTCPClient.java
SingleThreadedServer.java
TCPClientRunnable.java
ThreadPooledServer.java
UDPClient.java
UDPServer.java
WorkerRunnable.java

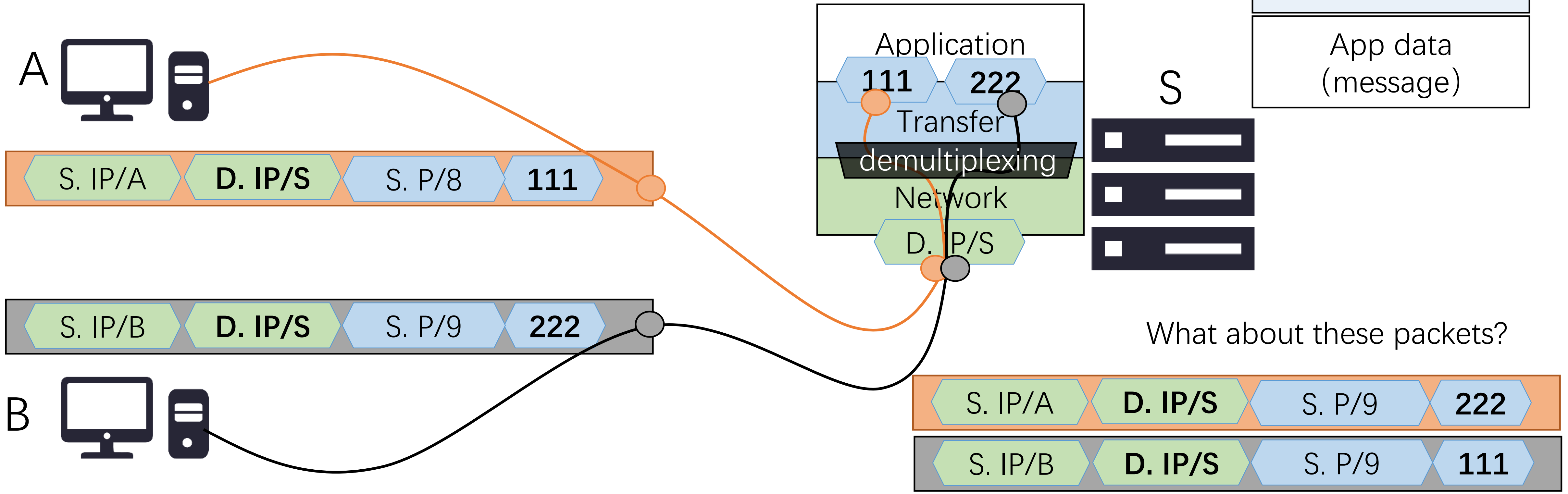
More @https://www.cs.uic.edu/~troy/spring05/cs450/sockets/socket.html

Connectionless multiplexing and demultiplexing

take UDP as an example

- A UDP socket is fully identified by a **two-tuple** consisting of a destination IP address and a destination port number, i.e. **{D. IP, D. P}.**
  - D. IP: addressing the host; D. P: identifying the OS buffer

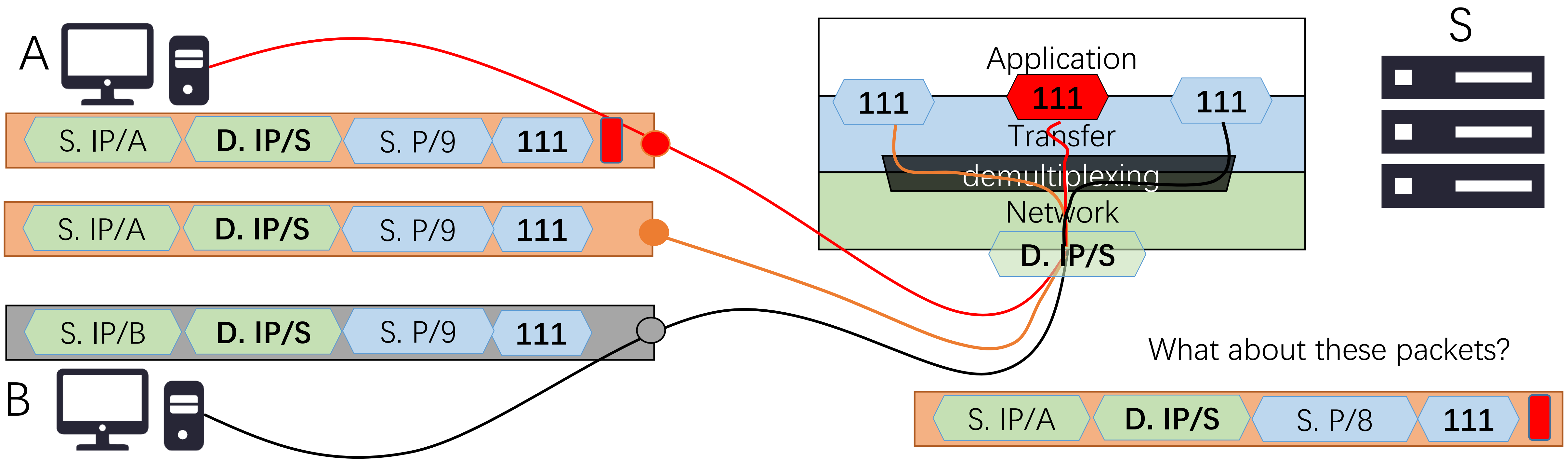




Connection-oriented multiplexing and demultiplexing

take TCP as an example

- A TDP socket is fully identified by a **four-tuple** consisting of **{S.IP, D. IP, S. P, D. P}.**
  - Recall that it is to identify the OS buffer



Web Server and TCP

- Web servers are basically TCP servers handling HTTP messages!
  - HTTP over TCP
- Being a TCP server means listening on a connection-oriented server socket!
  - Accepting connections from client sockets
  - During connection establishment, create a new connection socket/buffer!
- Single-threaded web server
  - Handling the server socket and connection sockets with the same execution process!
- Multi-threaded web server
  - Handling connection sockets by separated threats forked by the process managing the server socket!