Network Applications: Network Programming: UDP, TCP

Qiao Xiang

https://qiaoxiang.me/courses/cnns-xmuf21/index.shtml

10/12/2021

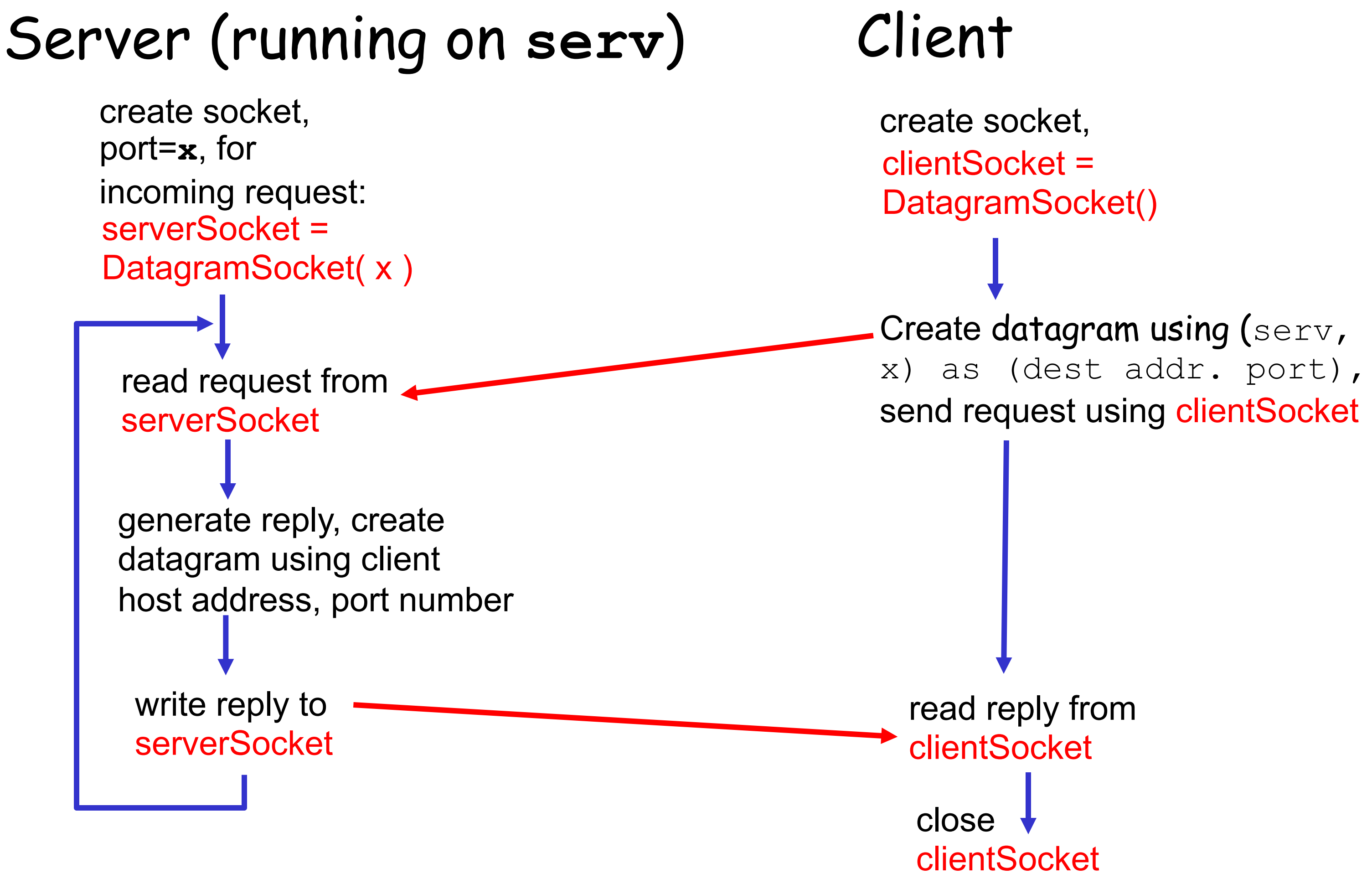
Outline

- Admin. and recap
- Network application programming
  - UDP sockets
  - TCP sockets

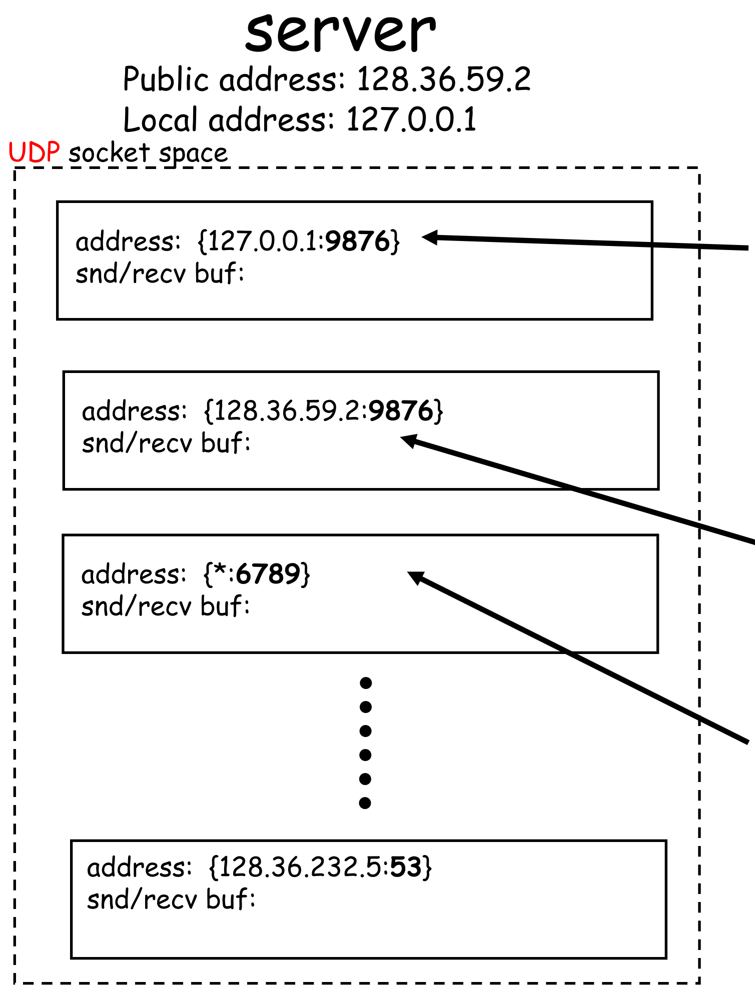
Admin.

- Lab assignment 1 due on Oct. 14
  - By email or in class

Recap: Connectionless UDP: Big Picture (Java version)


Server (running on **serv**)
Client
create socket, port=**x**, for incoming request:
serverSocket = DatagramSocket(x)
read request from serverSocket
generate reply, create datagram using client host address, port number
write reply to serverSocket
create socket,
clientSocket = DatagramSocket()
Create datagram using (`serv, x`) `as (dest addr. port)`,
send request using clientSocket
read reply from clientSocket
close clientSocket

Recap: UDP Sockets



```
InetAddress sIP1 =
  InetAddress.getByName("localhost");
DatagramSocket ssock1 =
  new DatagramSocket(9876, sIP1);


InetAddress sIP2 =
 InetAddress.getByName("128.36.59.2");
DatagramSocket ssock2 =
  new DatagramSocket(9876,sIP2);

DatagramSocket serverSocket =
  new DatagramSocket(6789);
```

UDP demutiplexing is based on matching (dst address, dst port)

Java Server (UDP): Processing

A simple upper-case UDP echo service is among the simplest network service. Are there any problems with the processing?

```
class UDPServer {
  public static void main(String args[]) throws Exception  {

      …
      DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
      serverSocket.receive(receivePacket);

      // process
      String sentence = new String(receivePacket.getData(),
                                   0, receivePacket.getLength());
      String capitalizedSentence = sentence.toUpperCase();
      sendData = capitalizedSentence.getBytes();

      // send
      DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
                                                     IPAddress, port);
      serverSocket.send(sendPacket);
```

Example: Java client (UDP)

```
import java.io.*;
import java.net.*;

class UDPClient {
    public static void main(String args[]) throws Exception
    {

      BufferedReader inFromUser =
        new BufferedReader(new InputStreamReader(System.in));
      String sentence = inFromUser.readLine();

      byte[] sendData = sentence.getBytes();

      DatagramSocket clientSocket = new DatagramSocket();

      InetAddress sIPAddress = InetAddress.getByName("servname");
```

Create input stream

Create client socket

Translate hostname to IP address using DNS

Example: Java client (UDP), cont.

Create datagram with data-to-send, length, IP addr, port

```
DatagramPacket sendPacket =
  new DatagramPacket(sendData, sendData.length, sIPAddress, 9876);
```

Send datagram to server

```
clientSocket.send(sendPacket);
```

```
byte[] receiveData = new byte[1024];
DatagramPacket receivePacket =
  new DatagramPacket(receiveData, receiveData.length);
```

Read datagram from server

```
clientSocket.receive(receivePacket);

String modifiedSentence =
   new String(receivePacket.getData());

System.out.println("FROM SERVER:" + modifiedSentence);
clientSocket.close();
}
    }
```

Demo

```
%ubuntu: java UDPServer
%netstat to see buffer

%ubuntu: java UDPClient <server>

%wireshark to capture traffic
```

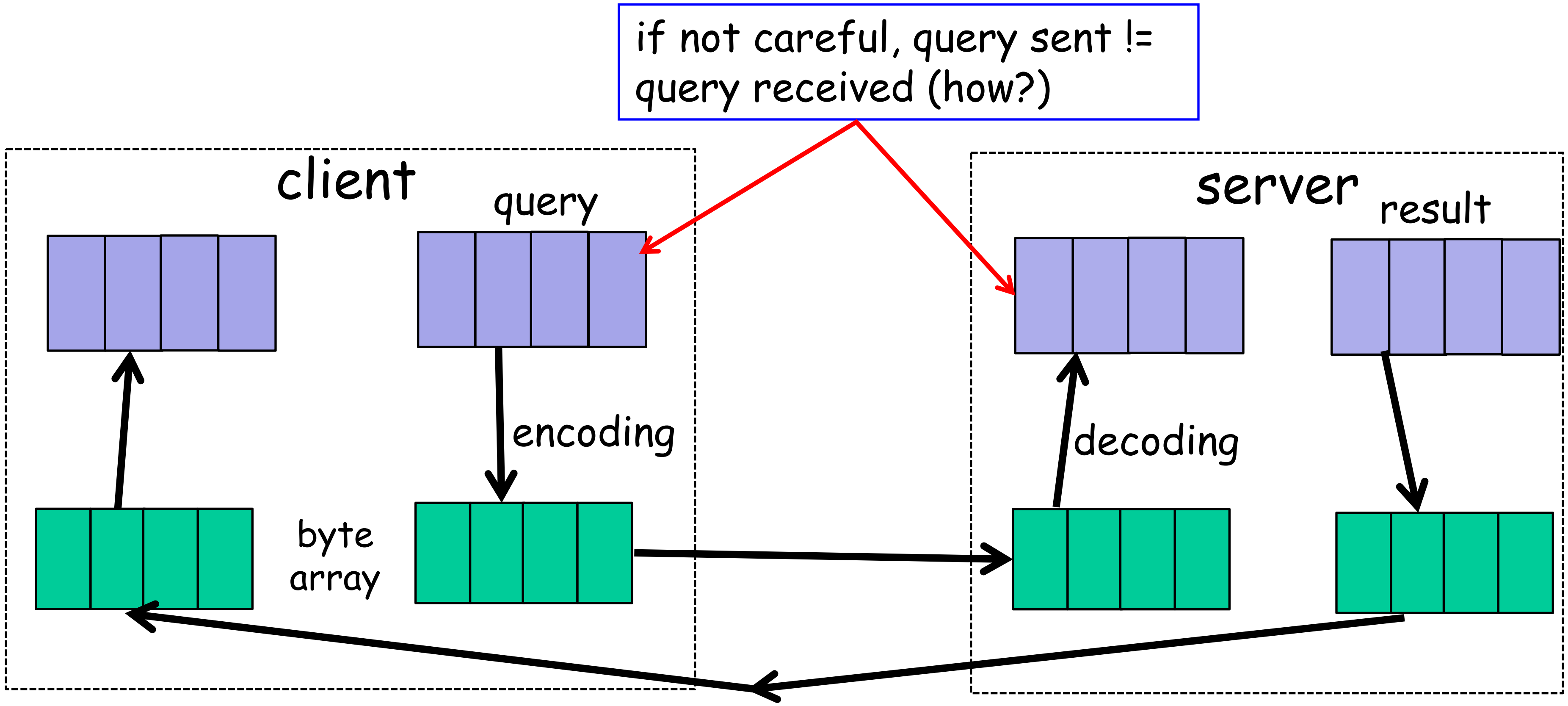
Discussion on Example Code

- A simple upper-case UDP echo service is among the simplest network service.

- Are there any problems with the program?

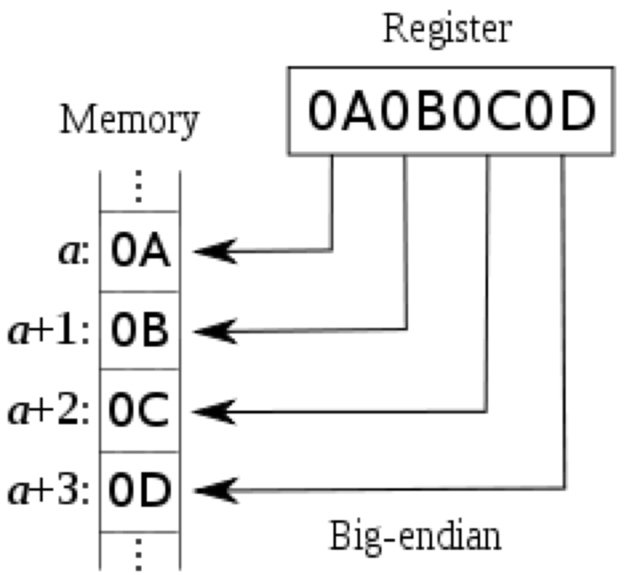
Data Encoding/Decoding

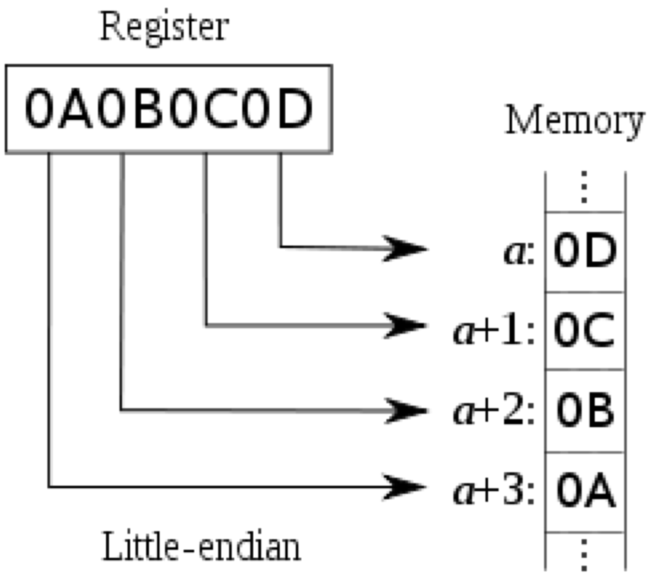
- Rule: ALWAYS pay attention to encoding/decoding of data



Example: Endianness of Numbers

- int var = 0x0A0B0C0D



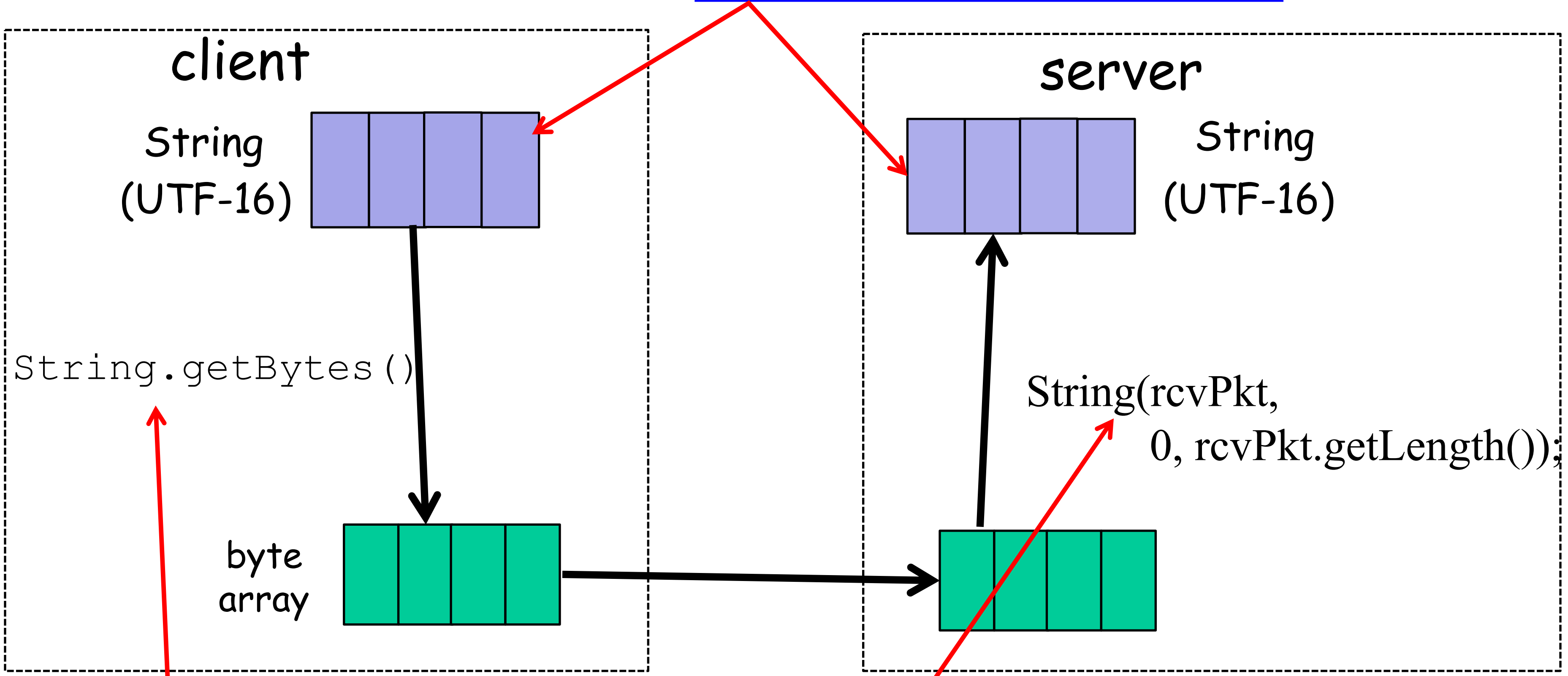


ARM, Power PC, Motorola 68k, IA-64 — Intel x86

- sent != received: take an int on a big-endian machine and send a little-endian machine

Example: String and Chars

Will we always get back the same string?



Depends on default local platform char set :
java.nio.charset.Charset.defaultCharset()

Example: Charset Troubles

- Try
  - java EncodingDecoding US-ASCII UTF-8

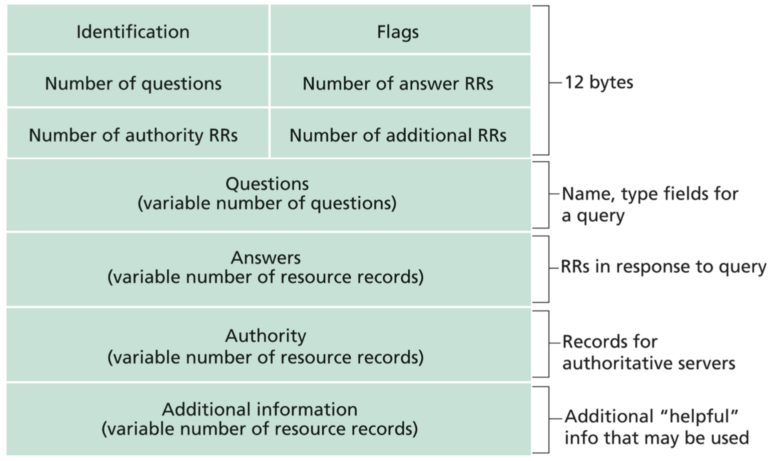
Encoding/Decoding as a Common Source of Errors

- Please read chapter 2 (Streams) of Java Network Programming for more details
    - Java stream, reader/writer can always be confusing, but it is good to finally understand

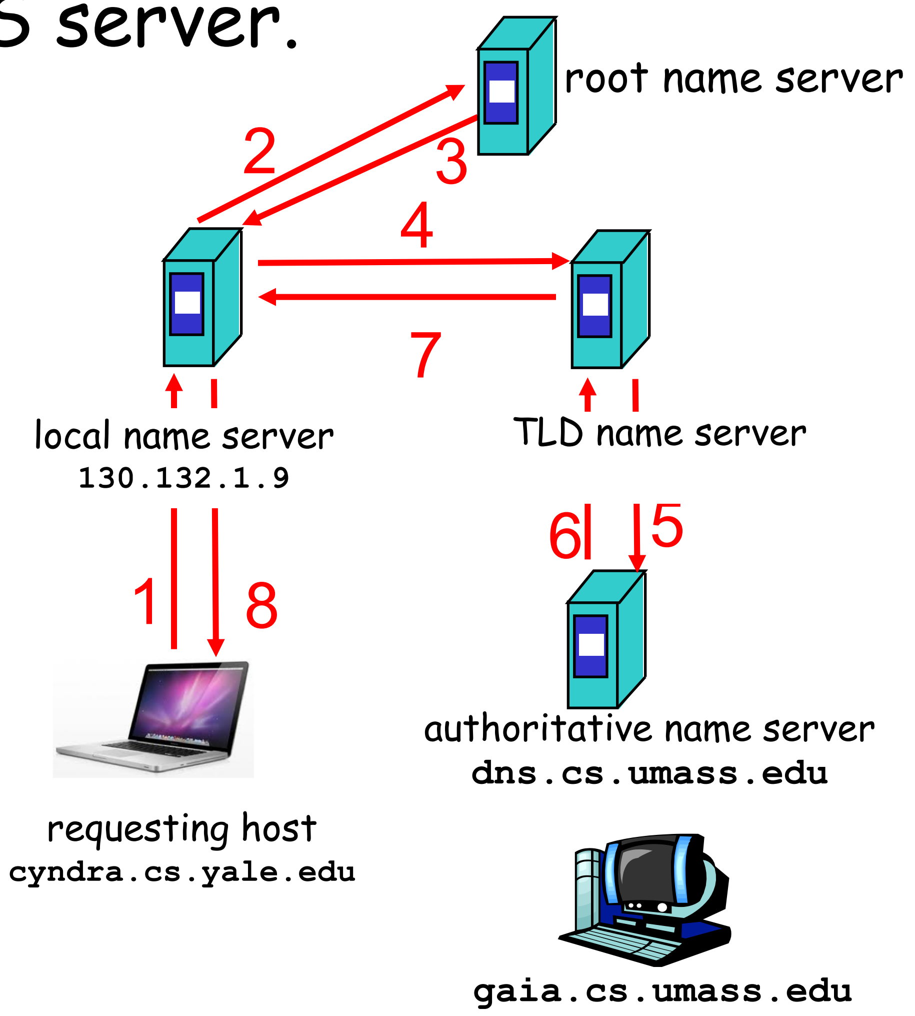
- Common mistake even in many (textbook) examples:
    - http://www.java2s.com/Code/Java/Network-Protocol/UseDatagramSockettosendoutandreceiveDatagramPacket.htm

Exercise: UDP/DNS Server Pseudocode

- Modify the example UDP server code to implement a local DNS server.



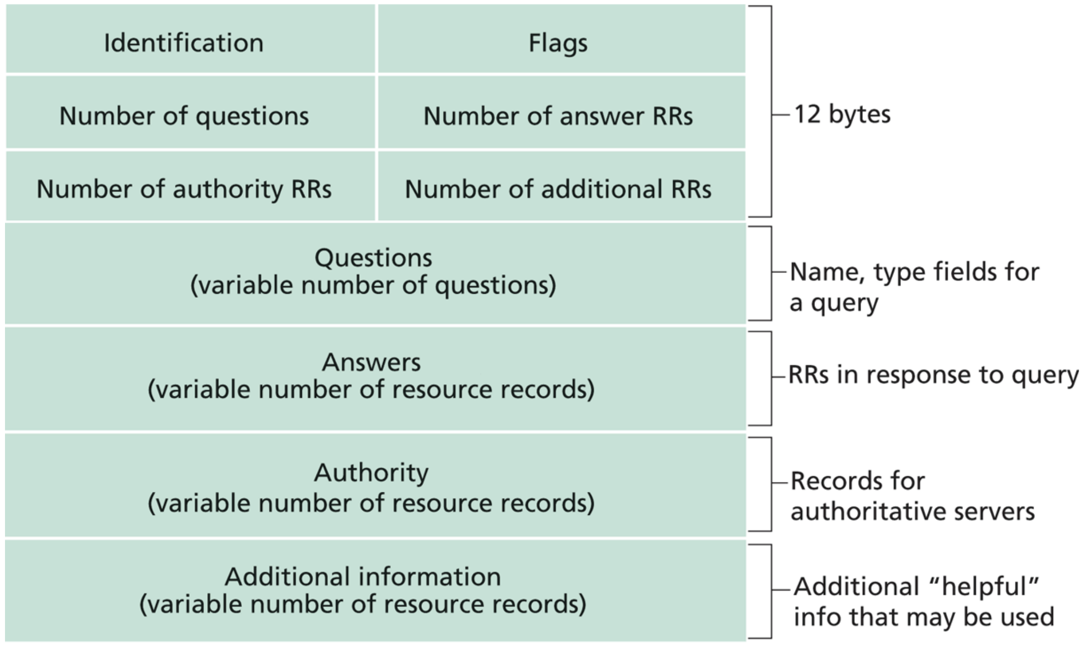
Identification	Flags	12 bytes
Number of questions	Number of answer RRs	
Number of authority RRs	Number of additional RRs	
Questions (variable number of questions)		Name, type fields for a query
Answers (variable number of resource records)		RRs in response to query
Authority (variable number of resource records)		Records for authoritative servers
Additional information (variable number of resource records)		Additional "helpful" info that may be used



UDP/DNS Implementation

- Standard UDP demultiplexing (find out return address by src.addr/src.port of UDP packet) does not always work

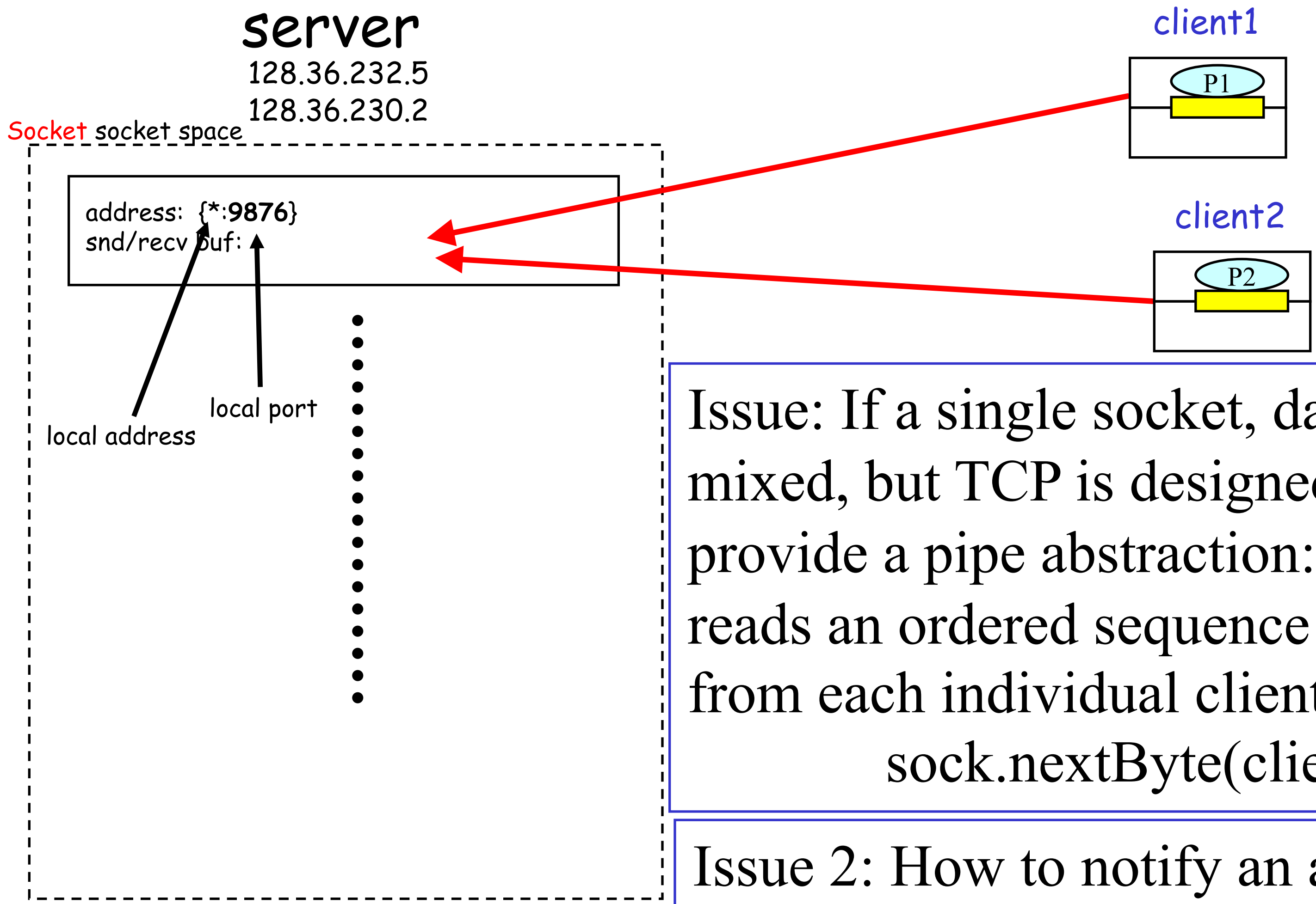
- DNS solution: identification: remember the mapping


Identification
Flags
Number of questions
Number of answer RRs
Number of authority RRs
Number of additional RRs
12 bytes
Questions (variable number of questions)
Name, type fields for a query
Answers (variable number of resource records)
RRs in response to query
Authority (variable number of resource records)
Records for authoritative servers
Additional information (variable number of resource records)
Additional "helpful" info that may be used

Outline

- Admin. and recap
- Network application programming
  - Overview
  - UDP
  - *Basic TCP*

TCP Socket Design: Starting w/ UDP



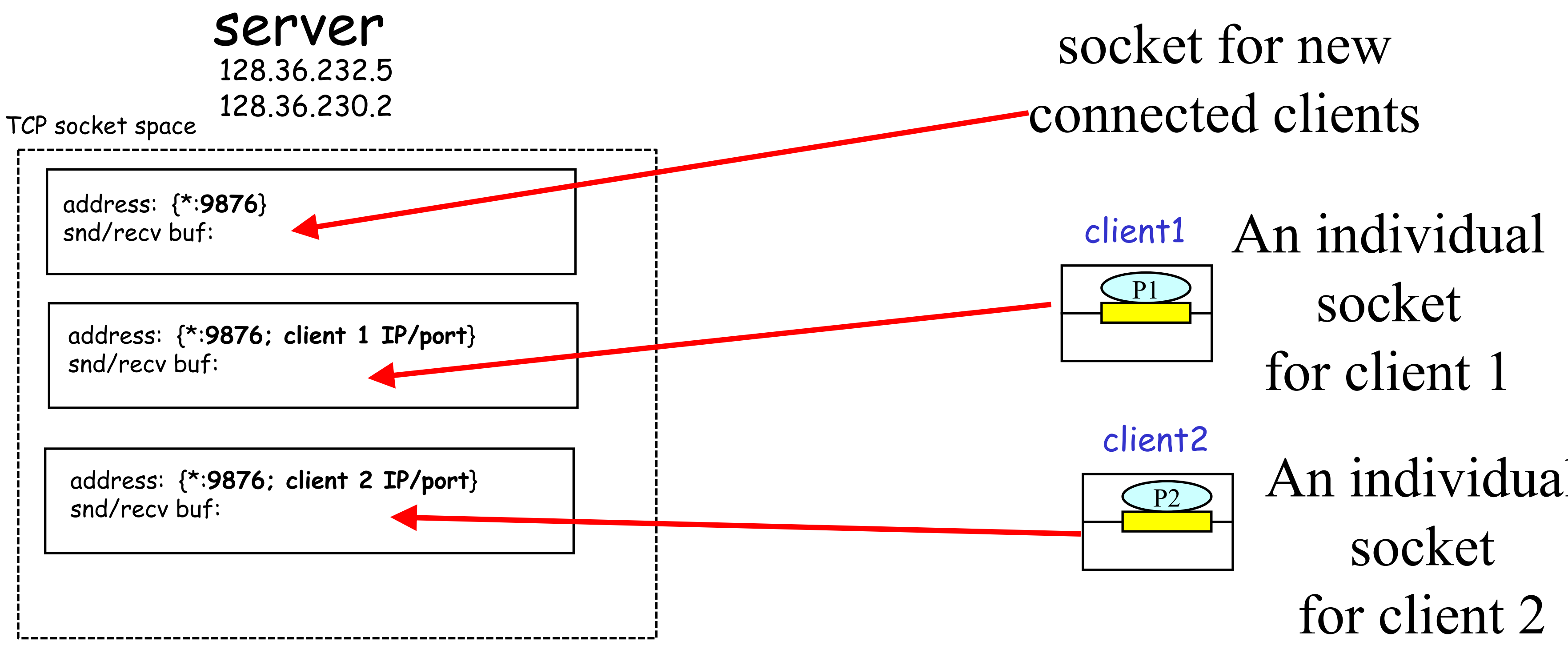
Issue: If a single socket, data can be mixed, but TCP is designed to provide a pipe abstraction: server reads an ordered sequence of bytes from each individual client.

sock.nextByte(client1)?

Issue 2: How to notify an app that a new client is connected?

newClient = sock.getNewClient()?

BSD TCP Socket API Design


server
128.36.232.5
128.36.230.2
TCP socket space
address: {*:9876}
snd/recv buf:
address: {*:9876; client 1 IP/port}
snd/recv buf:
address: {*:9876; client 2 IP/port}
snd/recv buf:
socket for new connected clients
client1
P1
An individual socket for client 1
client2
P2
An individual socket for client 2

Q: How to decide where to put a new TCP packet?

A: Packet demutiplexing is based on four tuples:
(dst addr, dst port, src addr, src port)

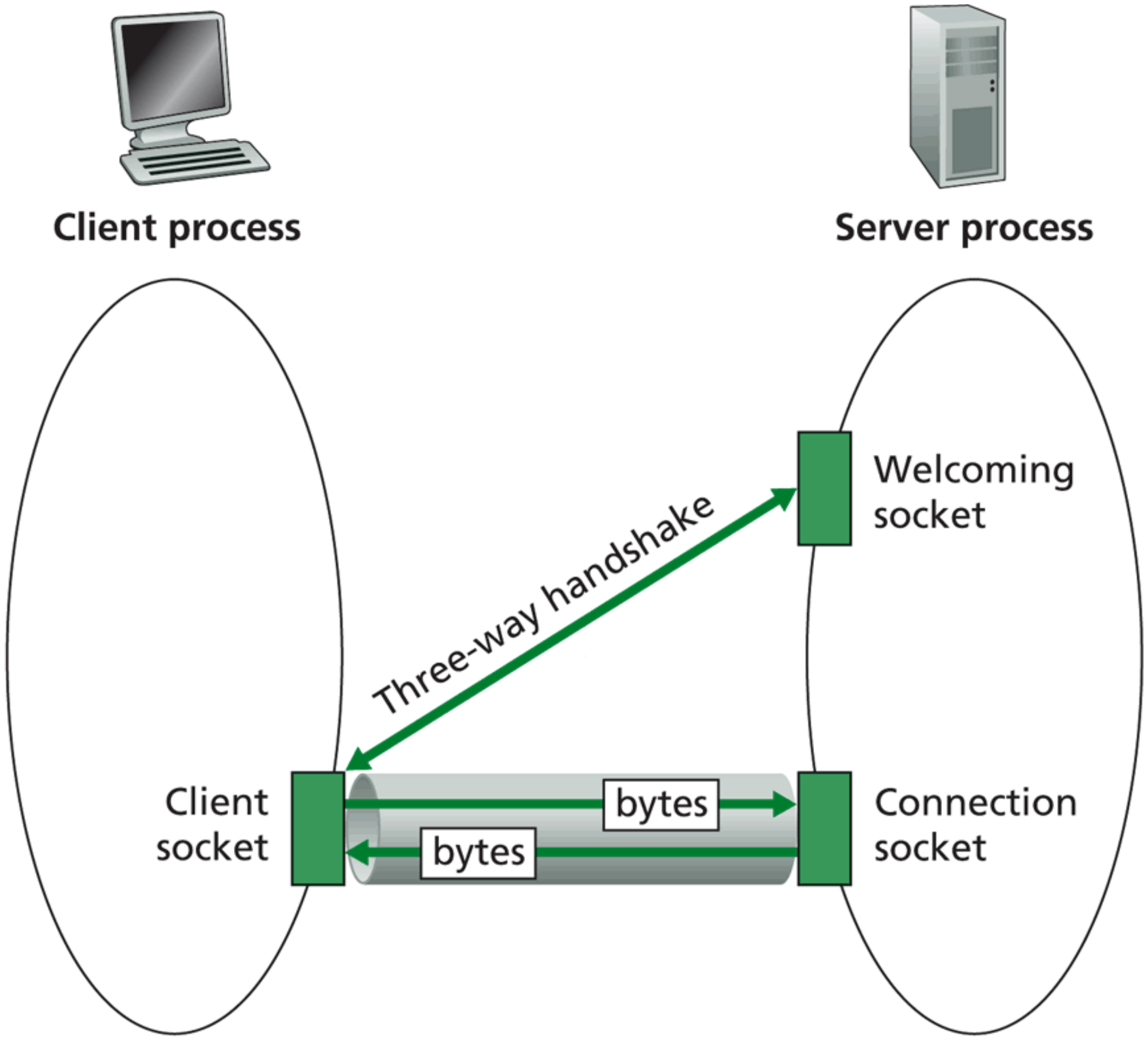
TCP Connection-Oriented Demux

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number

- recv host uses all four values to direct segment to appropriate socket
  - different connections/sessions are automatically separated into different sockets

-Welcome socket: the waiting room
-connSocket: the operation room

TCP Socket Big Picture



Client/server Socket Workflow: TCP

Server (running on `hostid`)

create socket,
port=**x**, for
incoming request:
welcomeSocket =
ServerSocket(x)

wait for incoming
connection request
connectionSocket =
welcomeSocket.accept()

read request from
connectionSocket

write reply to
connectionSocket

close
connectionSocket

TCP
connection setup

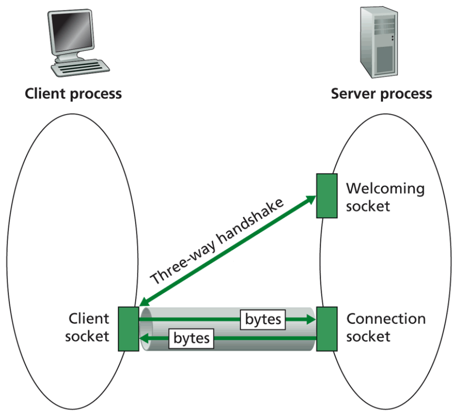
Client

create socket,
connect to hostid, port=**x**
clientSocket =
Socket()

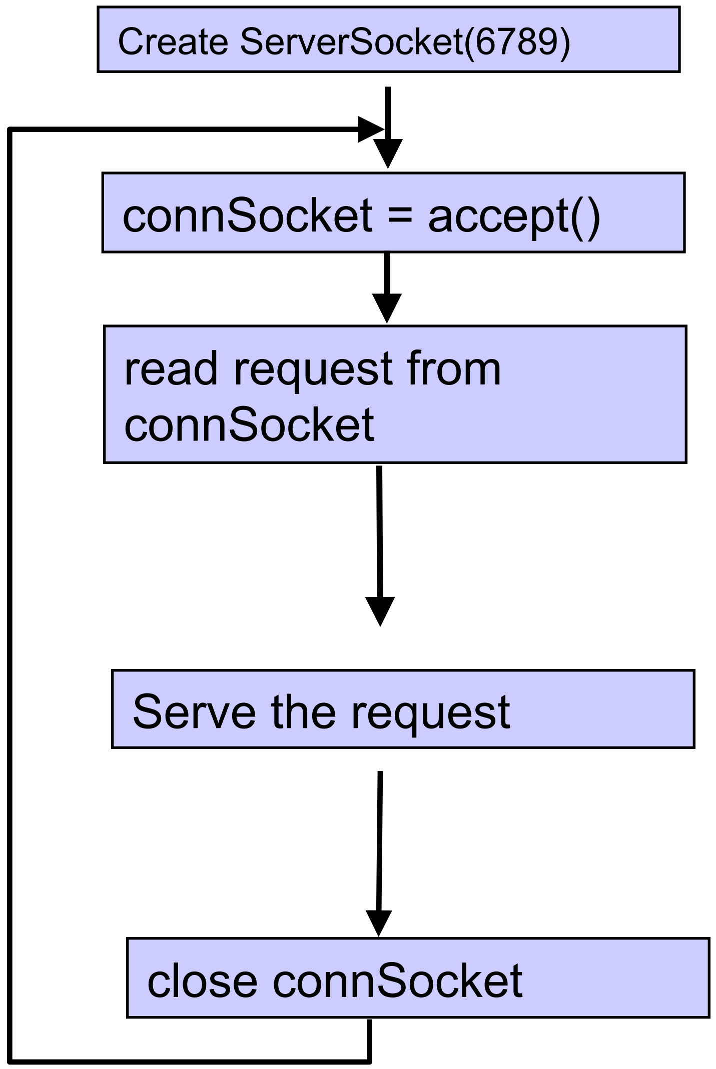
send request using
clientSocket

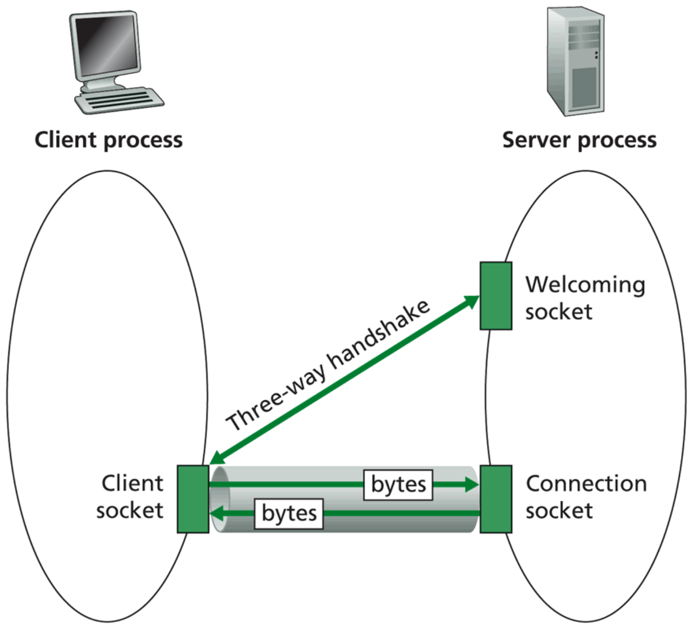
read reply from
clientSocket

close
clientSocket



Server Flow





-Welcome socket: the waiting room
-connSocket: the operation room

ServerSocket

- **ServerSocket**()
  - creates an unbound server socket.
- **ServerSocket**(int port)
  - creates a server socket, bound to the specified port.
- **ServerSocket**(int port, int backlog)
  - creates a server socket and binds it to the specified local port number, with the specified backlog.
- **ServerSocket**(int port, int backlog, InetAddress bindAddr)
  - creates a server with the specified port, listen backlog, and local IP address to bind to.

- **bind**(SocketAddress endpoint)
  - binds the ServerSocket to a specific address (IP address and port number).
- **bind**(SocketAddress endpoint, int backlog)
  - binds the ServerSocket to a specific address (IP address and port number).

- Socket **accept**()
  - listens for a connection to be made to this socket and accepts it.

- **close**()

  closes this socket.

(Client) Socket

- **Socket**(InetAddress address, int port)
  - creates a stream socket and connects it to the specified port number at the specified IP address.
- **Socket**(InetAddress address, int port, InetAddress localAddr, int localPort)
  - creates a socket and connects it to the specified remote address on the specified remote port.
- **Socket**(String host, int port)
  - creates a stream socket and connects it to the specified port number on the named host.

- **bind**(SocketAddress bindpoint)
  - binds the socket to a local address.

- **connect**(SocketAddress endpoint)
  - connects this socket to the server.
- **connect**(SocketAddress endpoint, int timeout)
  - connects this socket to the server with a specified timeout value.

- InputStream **getInputStream**()
  - returns an input stream for this socket.
- OutputStream **getOutputStream**()
  - returns an output stream for this socket.

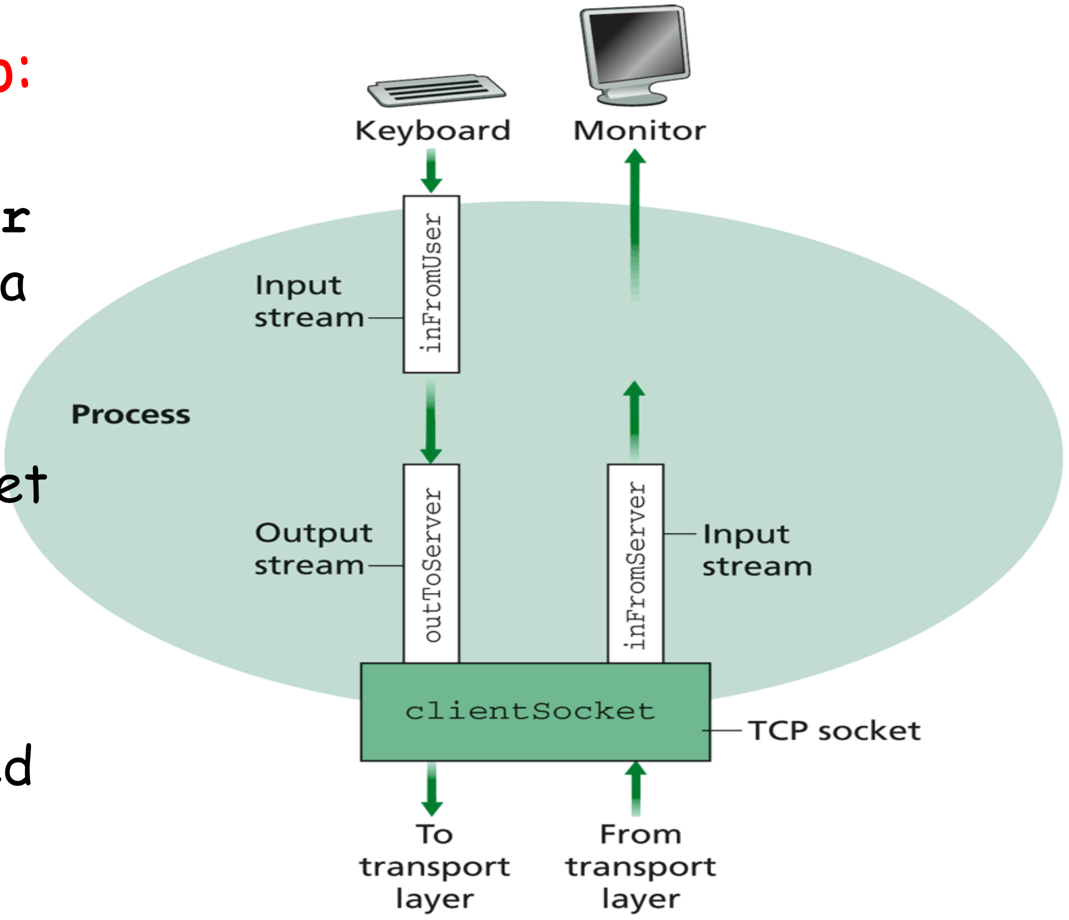
- **close**()

  closes this socket.

Simple TCP Example

Example client-server app:

1) client reads line from standard input (**inFromUser** stream) , sends to server via socket (**outToServer** stream)
2) server reads line from socket
3) server converts line to uppercase, sends back to client
4) client reads, prints modified line from socket (**inFromServer** stream)



Example: Java client (TCP)

```
import java.io.*;
import java.net.*;
class TCPClient {

    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String modifiedSentence;

        BufferedReader inFromUser =
          new BufferedReader(new InputStreamReader(System.in));
        sentence = inFromUser.readLine();

        Socket clientSocket = new Socket(“server.name", 6789);

        DataOutputStream outToServer =
          new DataOutputStream(clientSocket.getOutputStream());
```

Create input stream → (BufferedReader inFromUser …)

Create client socket, connect to server → (Socket clientSocket …)

Create output stream attached to socket → (DataOutputStream outToServer …)

OutputStream

- public abstract class OutputStream
  - public abstract void write(int b) throws IOException
  - public void write(byte[] data) throws IOException
  - public void write(byte[] data, int offset, int length) throws IOException
  - public void flush( ) throws IOException
  - public void close( ) throws IOException

InputStream

- public abstract class InputStream
  - public abstract int read( ) throws IOException
  - public int read(byte[] input) throws IOException
  - public int read(byte[] input, int offset, int length) throws IOException
  - public long skip(long n) throws IOException
  - public int available( ) throws IOException
  - public void close( ) throws IOException

Example: Java client (TCP), cont.

Send line to server

```
        outToServer.writeBytes(sentence + '\n');
```

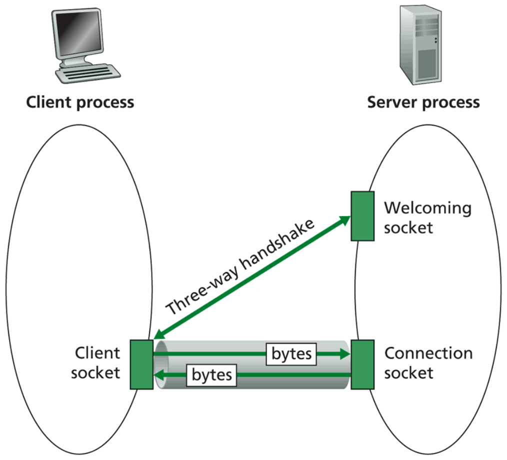
Create input stream attached to socket

```
         BufferedReader inFromServer =
           new BufferedReader(new
           InputStreamReader(clientSocket.getInputStream()));
```

Read line from server

```
          modifiedSentence = inFromServer.readLine();

          System.out.println("FROM SERVER: " + modifiedSentence);

          clientSocket.close();

      }
  }
```

Example: Java server (TCP)



```
import java.io.*;
import java.net.*;

class TCPServer {

 public static void main(String argv[]) throws Exception
  {
    String clientSentence;
    String capitalizedSentence;

    ServerSocket welcomeSocket = new ServerSocket(6789);
```

Create welcoming socket at port 6789

Demo

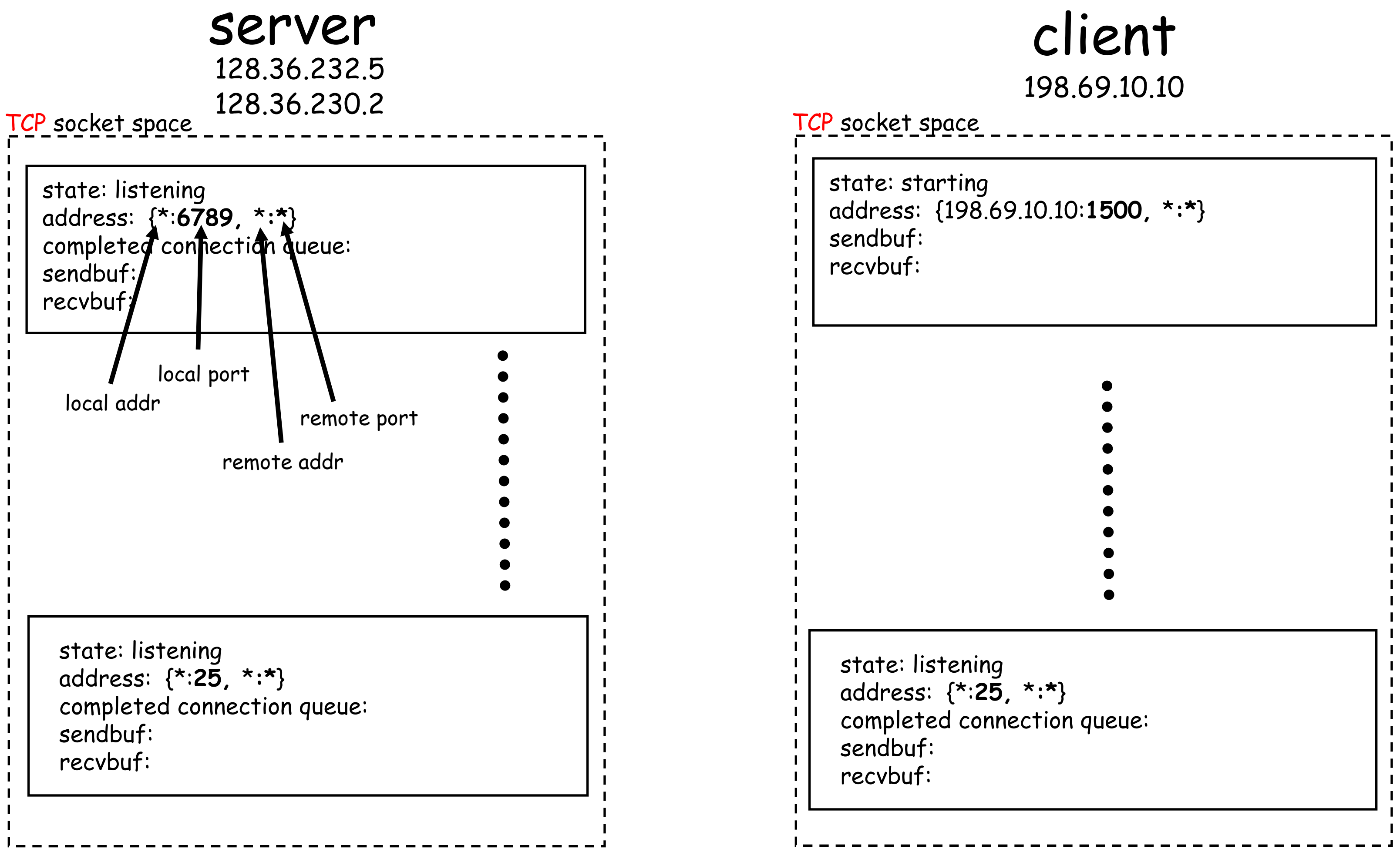
% on MAC

start TCPServer

wireshark to capture our TCP traffic

tcp.srcport==6789 or tcp.dstport==6789

Under the Hood: After Welcome (Server) Socket



TCP socket space

state: listening
address: {*:**6789**, *:*}
completed connection queue:
sendbuf:
recvbuf:

state: listening
address: {*:**25**, *:*}
completed connection queue:
sendbuf:
recvbuf:

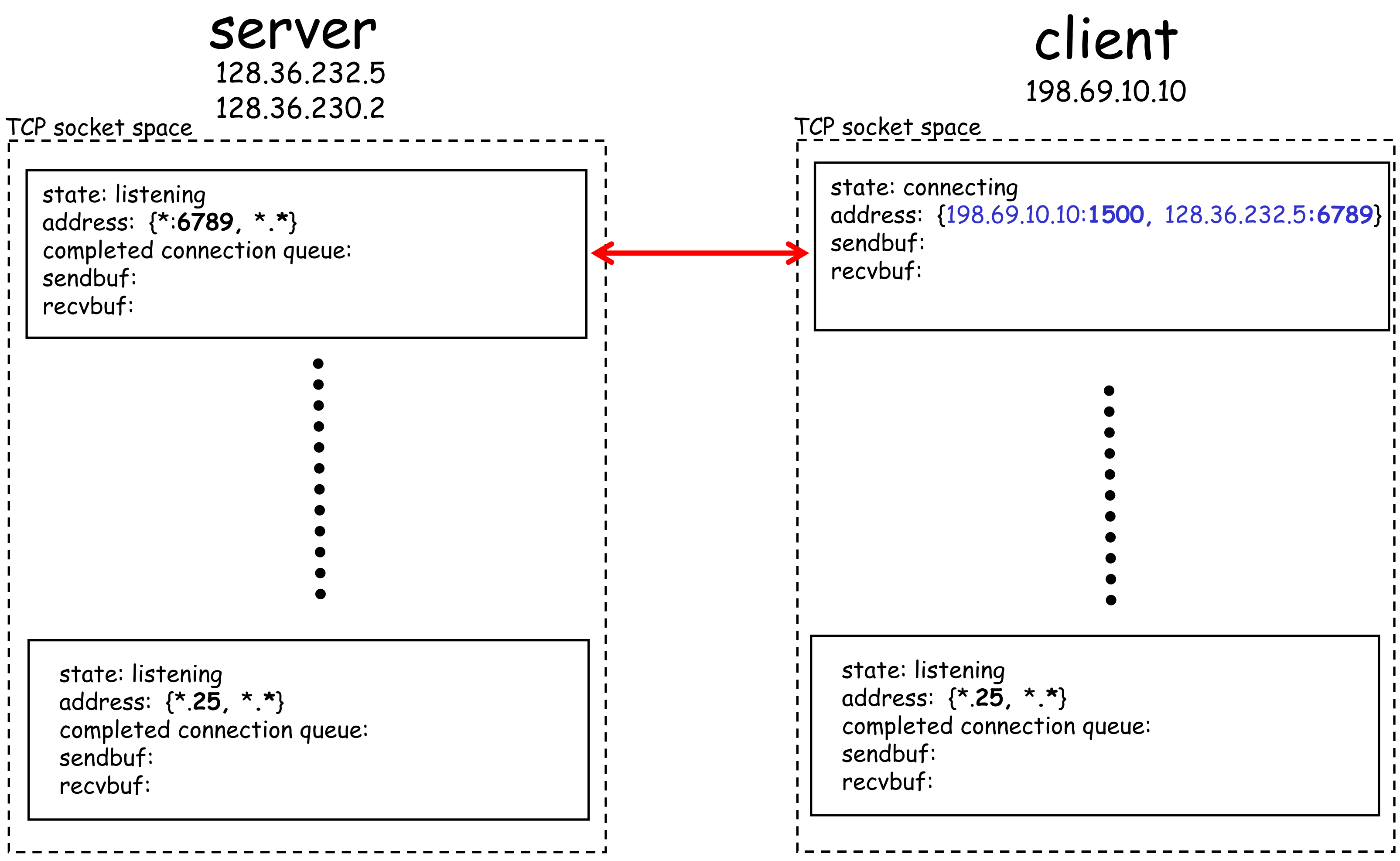
TCP socket space

state: starting
address: {198.69.10.10:**1500**, *:*}
sendbuf:
recvbuf:

state: listening
address: {*:**25**, *:*}
completed connection queue:
sendbuf:
recvbuf:

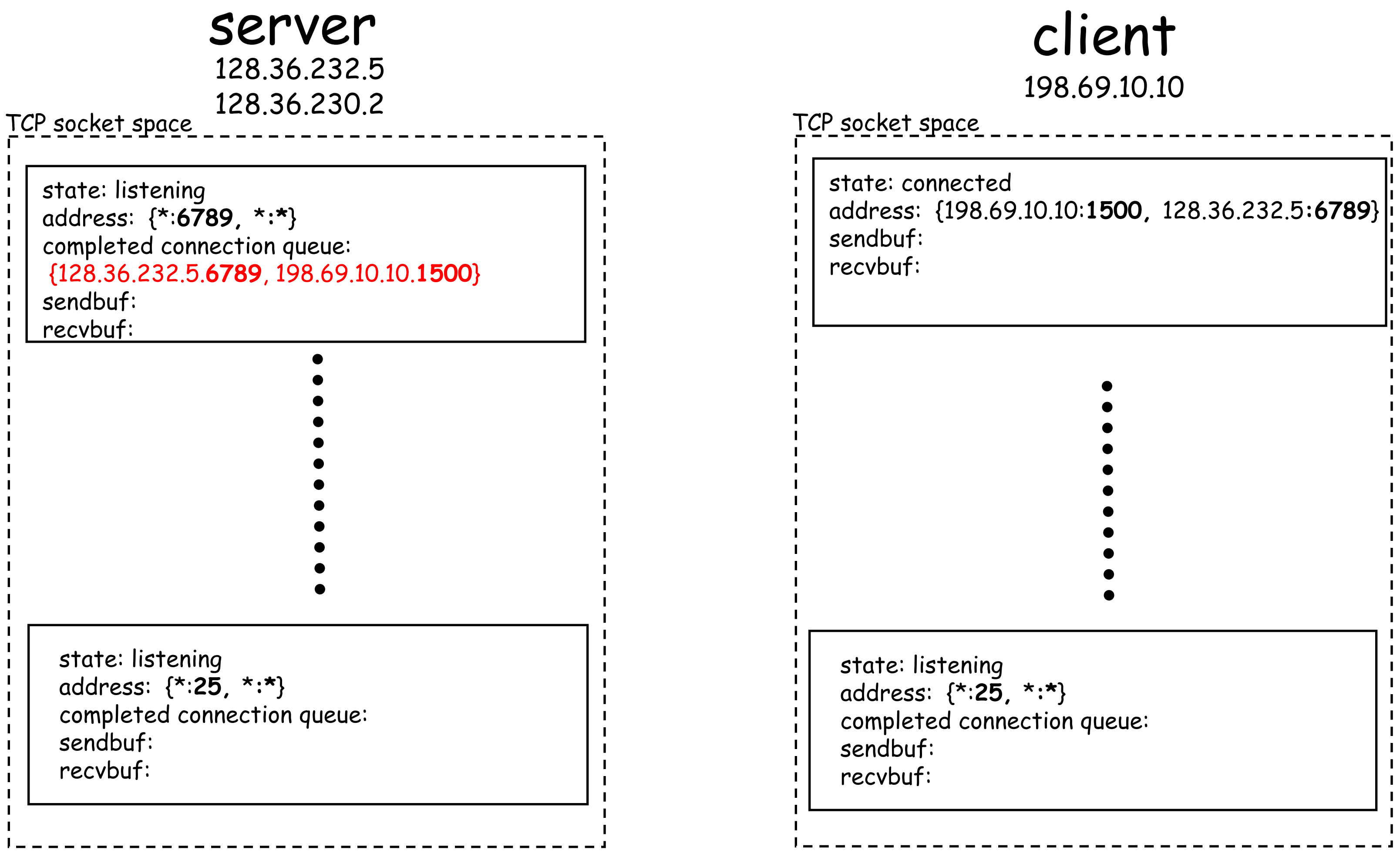
```
%netstat -p tcp -n -a
```

After Client Initiates Connection



```
%ubuntu java TCPClient <server> 6789
```

Example: Client Connection Handshake Done



server
128.36.232.5
128.36.230.2

TCP socket space

state: listening
address: {*:**6789**, *:*}
completed connection queue:
{128.36.232.5.**6789**, 198.69.10.10.**1500**}
sendbuf:
recvbuf:

state: listening
address: {*:**25**, *:*}
completed connection queue:
sendbuf:
recvbuf:

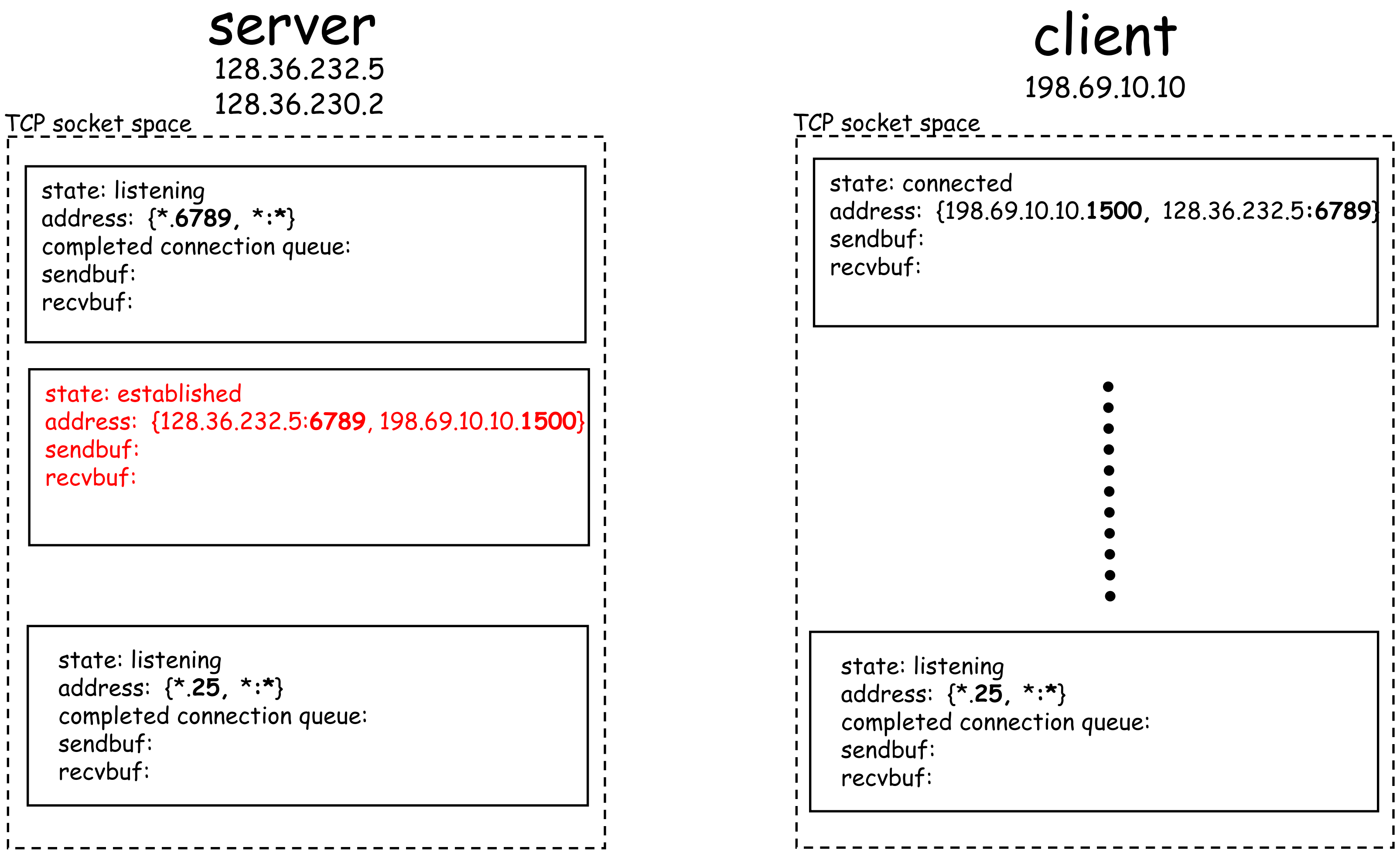
client
198.69.10.10

TCP socket space

state: connected
address: {198.69.10.10:**1500**, 128.36.232.5:**6789**}
sendbuf:
recvbuf:

state: listening
address: {*:**25**, *:*}
completed connection queue:
sendbuf:
recvbuf:

Example: Client Connection Handshake Done


server
128.36.232.5
128.36.230.2

TCP socket space

state: listening
address: {*.**6789**, *:*}
completed connection queue:
sendbuf:
recvbuf:

state: established
address: {128.36.232.5:**6789**, 198.69.10.10.**1500**}
sendbuf:
recvbuf:

state: listening
address: {*.**25**, *:*}
completed connection queue:
sendbuf:
recvbuf:

client
198.69.10.10

TCP socket space

state: connected
address: {198.69.10.10.**1500**, 128.36.232.5:**6789**}
sendbuf:
recvbuf:

state: listening
address: {*.**25**, *:*}
completed connection queue:
sendbuf:
recvbuf:

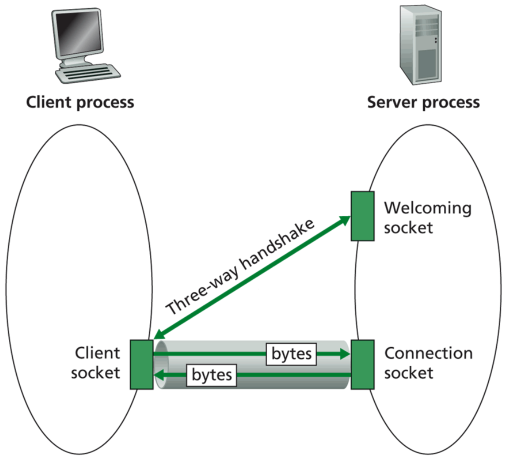
Packet demutiplexing is based on (dst addr, dst port, src addr, src port)

Packet sent to the socket with the best match!

Demo

- What if more client connections than backlog allowed?
  - We continue to start java TCPClient

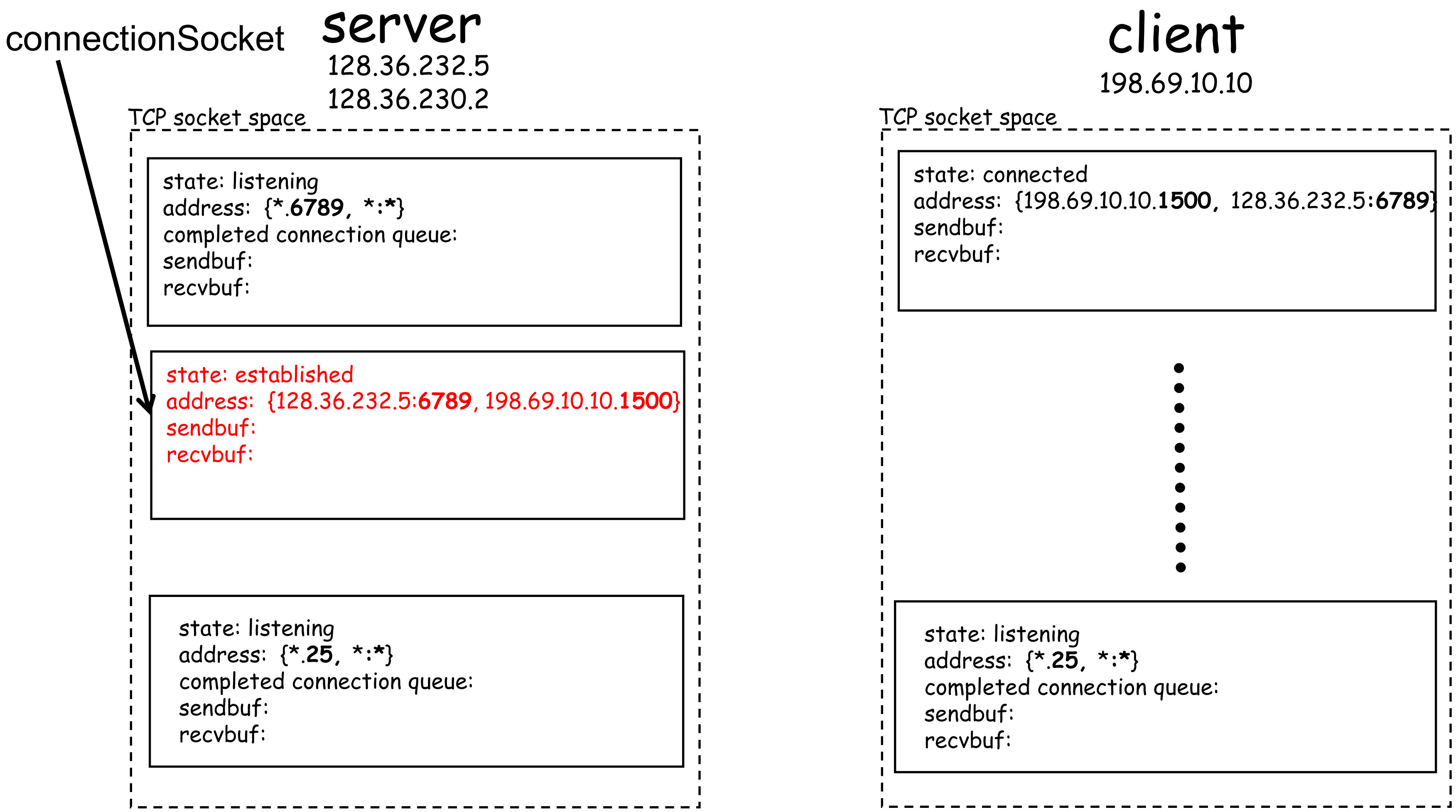
Example: Java server (TCP)



```
import java.io.*;
import java.net.*;

class TCPServer {

  public static void main(String argv[]) throws Exception
   {
     String clientSentence;
     String capitalizedSentence;

     ServerSocket welcomeSocket = new ServerSocket(6789);

     while(true) {

        Socket connectionSocket = welcomeSocket.accept();
```

Wait, on welcoming socket for contact by client

Example: Server accept()



connectionSocket

server

128.36.232.5
128.36.230.2

TCP socket space

state: listening
address: {*.**6789**, *:*}
completed connection queue:
sendbuf:
recvbuf:

state: established
address: {128.36.232.5:**6789**, 198.69.10.10.**1500**}
sendbuf:
recvbuf:

state: listening
address: {*.**25**, *:*}
completed connection queue:
sendbuf:
recvbuf:

client

198.69.10.10

TCP socket space

state: connected
address: {198.69.10.10.**1500**, 128.36.232.5:**6789**}
sendbuf:
recvbuf:

state: listening
address: {*.**25**, *:*}
completed connection queue:
sendbuf:
recvbuf:

Example: Java server (TCP): Processing

Create input stream, attached to socket

```
BufferedReader inFromClient =
      new BufferedReader(new
         InputStreamReader(connectionSocket.getInputStream()));
```

Read in line from socket

```
clientSentence = inFromClient.readLine();

capitalizedSentence = clientSentence.toUpperCase() + '\n';
```

```
          }
       }
    }
```

Example: Java server (TCP): Output

Create output stream, attached to socket

Write out line to socket

End of while loop, loop back and wait for another client connection

```
                DataOutputStream  outToClient =
                  new DataOutputStream(connectionSocket.getOutputStream());

                outToClient.writeBytes(capitalizedSentence);
            }
        }
    }
```