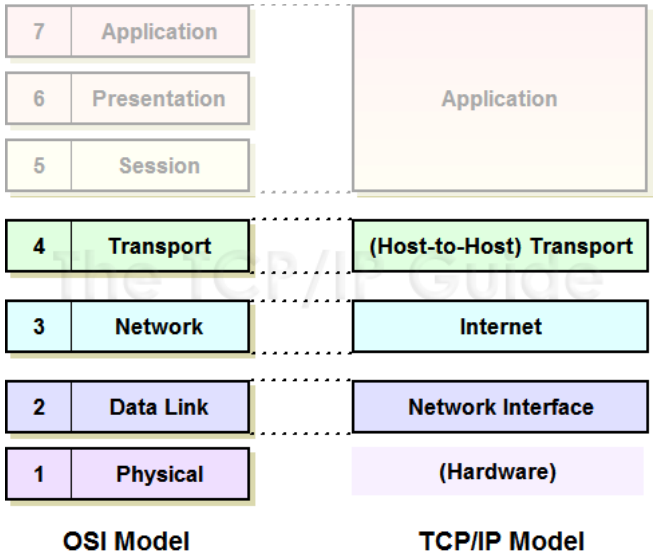
Socket Programming in Java

Required reading:
Kurose 2.7

EECS 3214, Winter 2020
Instructor: N. Vlajic


7
Application
6
Presentation
5
Session
Application
4
Transport
(Host-to-Host) Transport
3
Network
Internet
2
Data Link
Network Interface
1
Physical
(Hardware)
OSI Model
TCP/IP Model

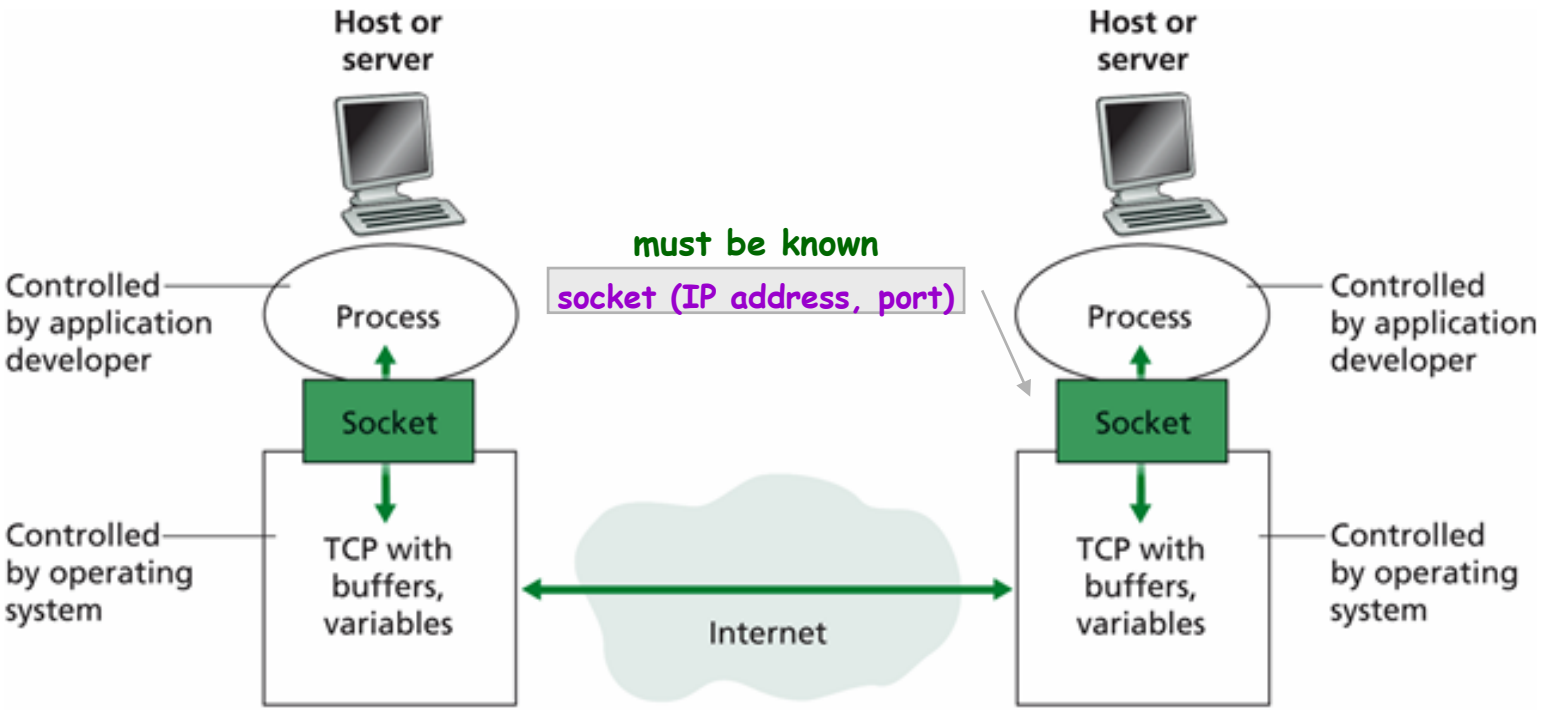
Socket Programming

Socket – a local-host, application created, OS controlled interface (a "door") into which application process can send/receive messages to/from another application process

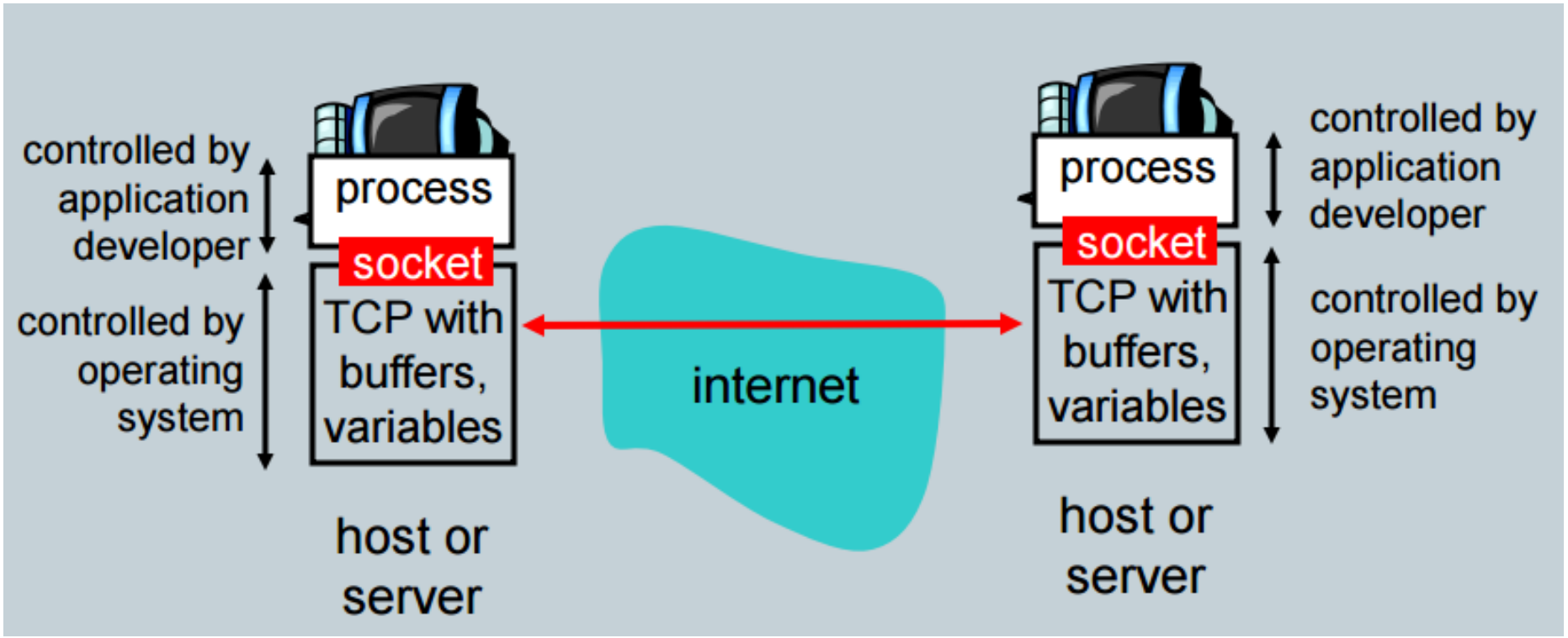
- also, a door between application process and end-to-end transport protocols (TCP or UDP)
- each TCP/UDP socket is uniquely identified with 2 pieces of information

  (1) name or address of the host (IP address)

  (2) identifier of the given process in the destination host (port number)



Socket Programming (cont.)



Socket Programming – **development of client/server application(s) that communicate using sockets**

- **developer has control of everything on application side but has little control of transport side of socket**
- **only control on transport-layer side is**
  - **(1) choice of transport protocol (TCP or UDP)**
  - **(2) control over a few transport-layer parameters e.g. max buffer and max segment size**

Socket programming refers to programming at the application level/layer!

TCP vs. UDP in Socket Programming – **to decide which transport-layer protocol, i.e. which type of socket, our application should use, we need to understand how TCP and UDP differ in terms of**

- **reliability**
- **timing**
- **overhead**

TCP vs. UDP Reliability

- **UDP - there is no guarantee that the sent datagrams will be received by the receiving socket** 👎
- **TCP - it is guaranteed that the sent packets will be received in exactly the same order in which they were sent** 👍

TCP vs. UDP Timing

- **UDP - does not include a congestion-control mechanism, so a sending process can pump data into network at any rate it pleases** (although not all the data may make it to the receiving socket) 👍
- **TCP - TCP congestion control mechanism throttles a sending process when the network is congested – TCP guarantees that data will eventually arrive at the receiving process, but there is no limit on how long it may take** 👎

TCP vs. UDP Overhead

- **UDP - every time a datagram is passed into the socket, the local and receiving socket address need to be passed along with it** (processing overhead) 👎
- **TCP - a connection must be established before communications between the pair of sockets start** (connection setup time overhead) 👎

TCP vs. UDP in Socket Programming (cont.)

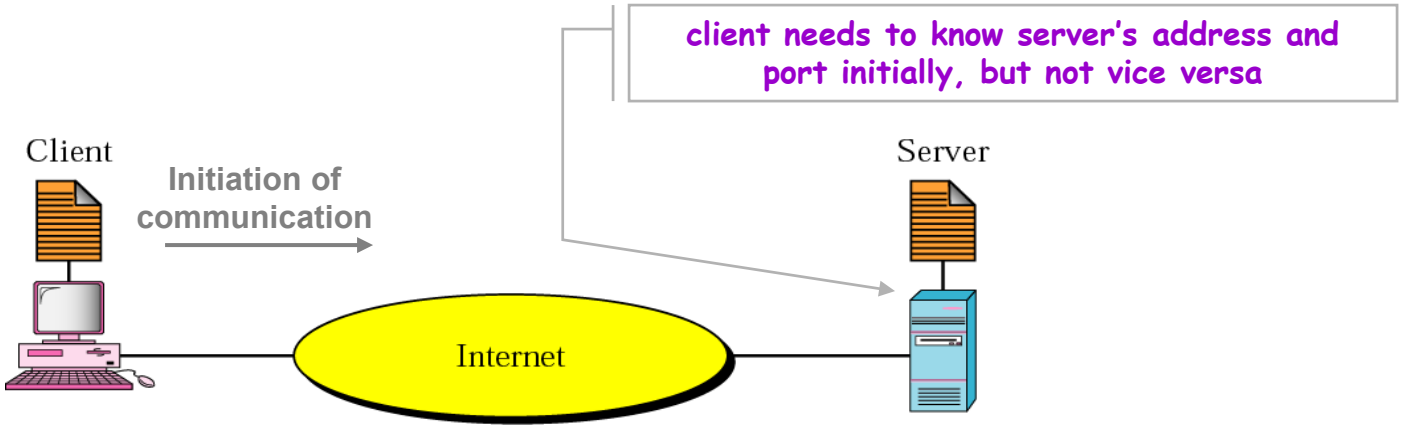
- **TCP is useful when indefinite amount of data need to be transferred 'in order' and reliably**
  - **otherwise, we end up with jumbled files or invalid information**
  - **examples: HTTP, ftp, telnet, …**
- **UDP is useful when data transfer should not be slowed down by extra overhead of reliable TCP connection**
  - **examples: real-time applications**
  - **e.g. consider a clock server that sends the current time to its client – if the client misses a packet, it doesn't make sense to resend it because the time will be incorrect when the client receives it on the second try**

In socket programming we pick transport-layer protocol that has services that best match the needs of our application.

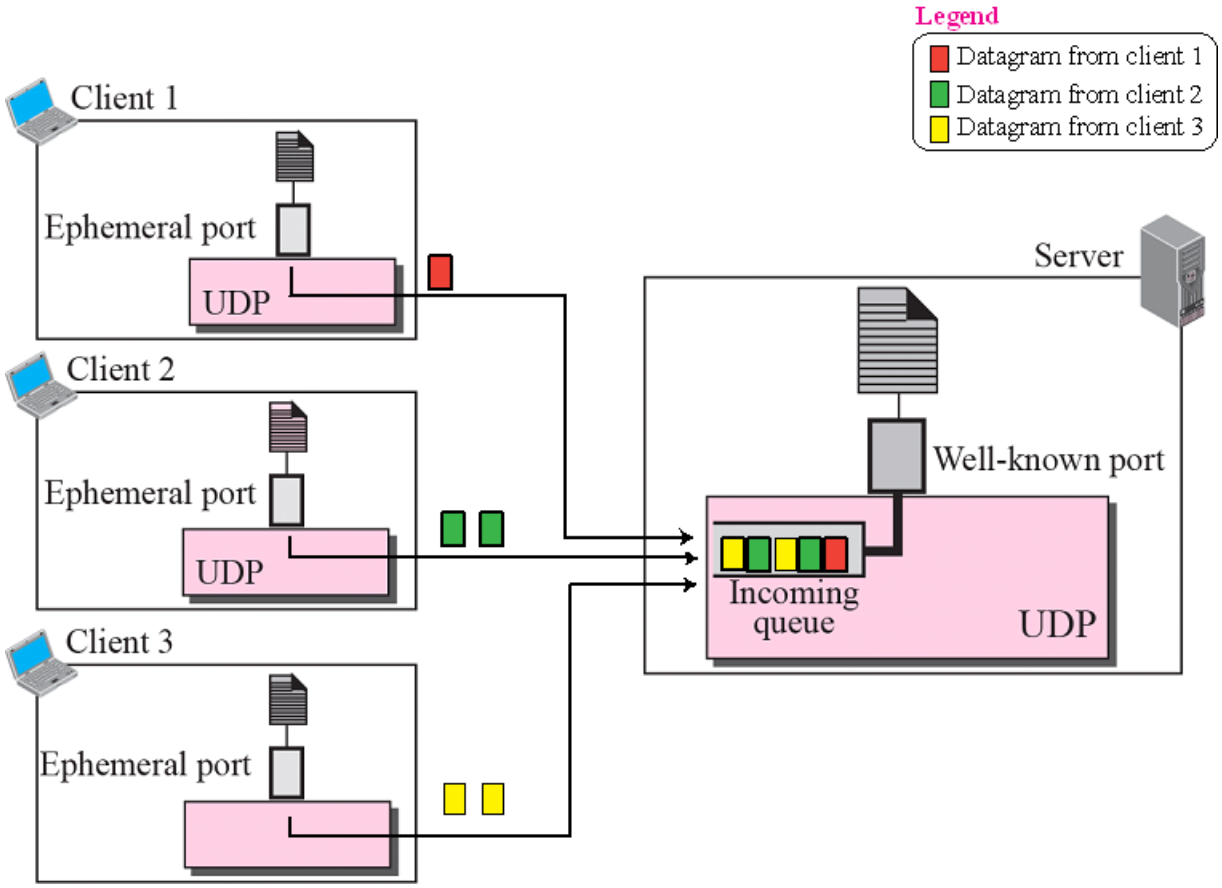
Client-Server Model

Client-Server Model – most common form of network communication in the Internet whose purpose is to enable/provide various types of service to users

- **CLIENT**: process that initiates communication, requests service, and receives response
  - although request-response part can be repeated several times, whole process is finite and eventually comes to an end
- **SERVER**: process that passively waits to be contacted and subsequently provides service to clients
  - runs infinitely
  - can be iterative or concurrent



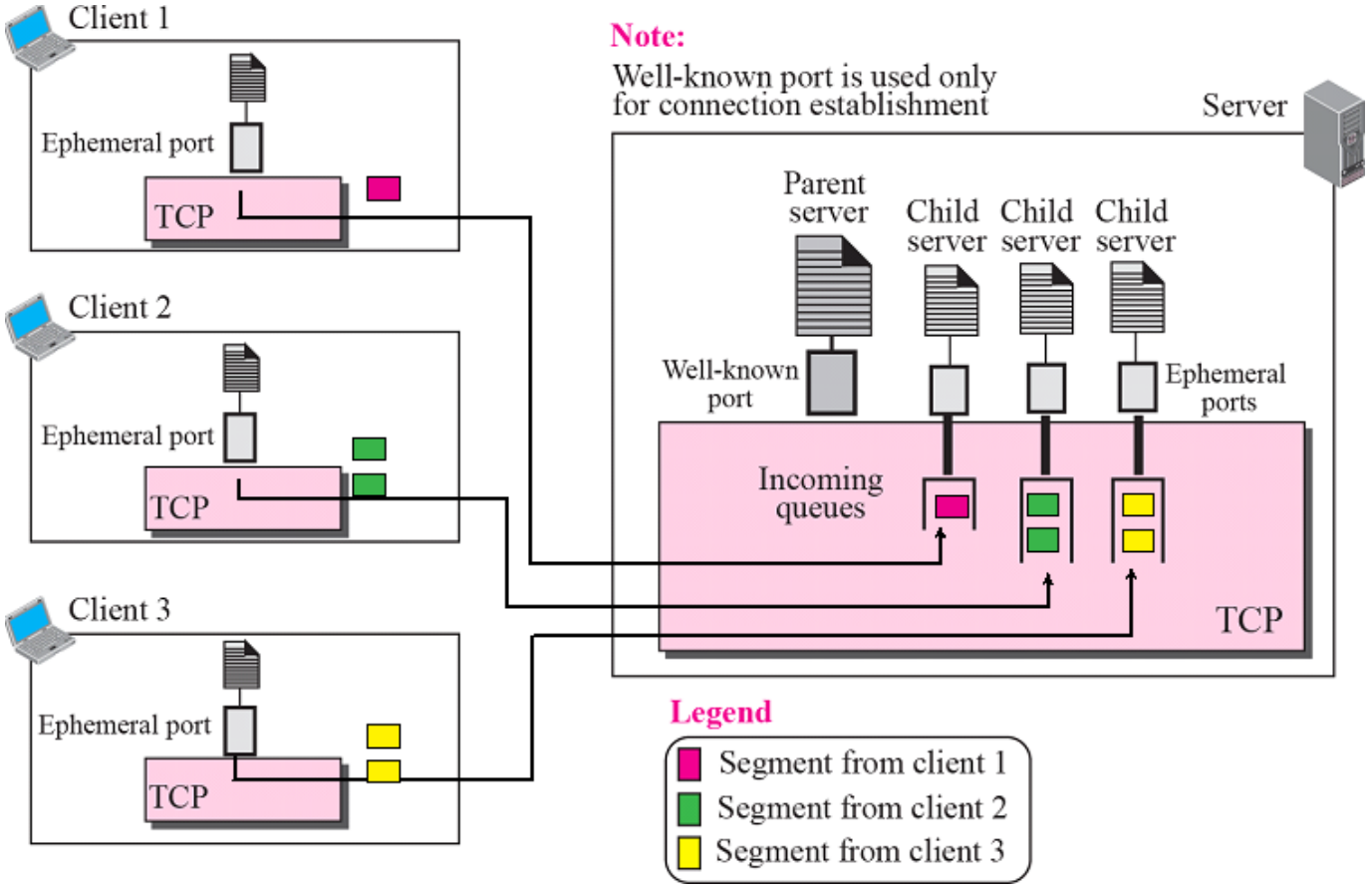
Example [iterative vs. concurrent servers]



An iterative server can process only <u>one request at a time</u> – it receives a request, processes it, and sends the response to the requestor before handling another request.

The servers that use UDP (i.e. connectionless servers) are normally iterative.

Example **[iterative vs. concurrent servers]**



A concurrent server can process <u>many requests at the same time</u>.

The servers that use TCP (i.e., connection-oriented servers) are normally concurrent.

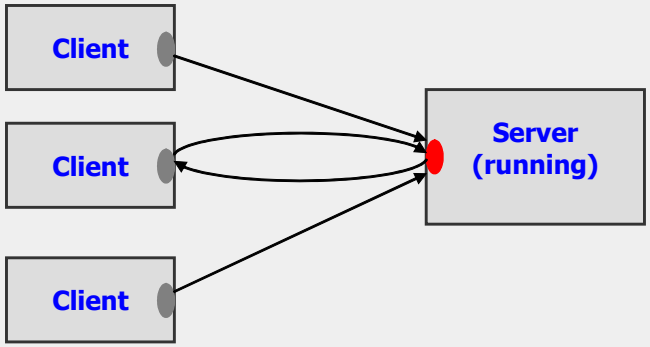
Principles of Client-Server Communication with UDP

- **CLIENT**:

(1) create socket

(2) repeat:
  (2.a) send
  (2.b) receive
  (2.c) process

(3) destroy



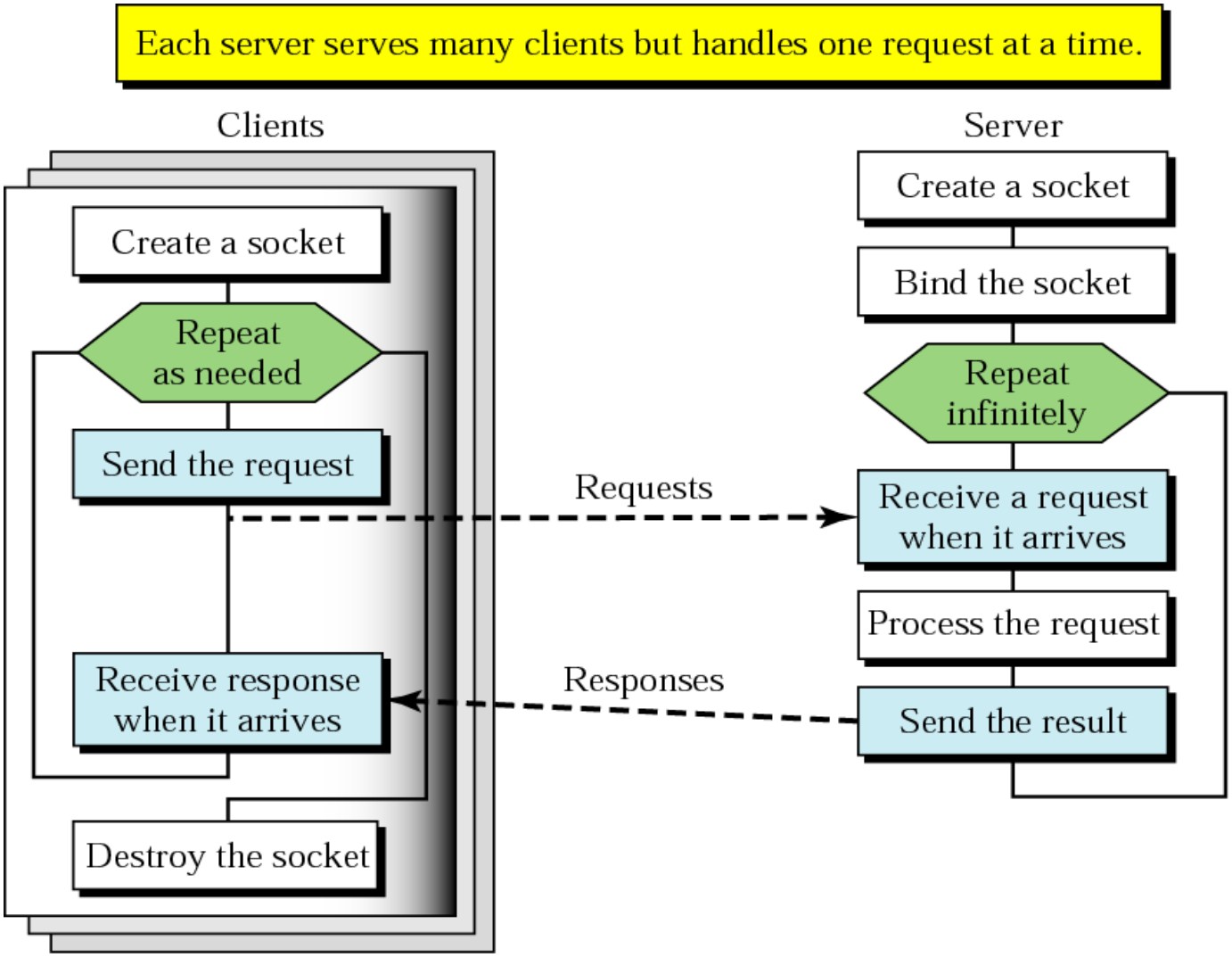
- **SERVER**:

(1) create socket

(2) repeat:
  (2.a) receive
  (2.b) process
  (2.c) send

- the server uses the same (listening) socket to communicate with all the clients
- clients and server exchange packets (datagrams)
- no handshaking
- sender explicitly attaches IP address and port of destination to each packet
- server must extract IP and port of sender from received packet to be able to send its response back

Principles of Client-Server Communication with UDP (cont.)

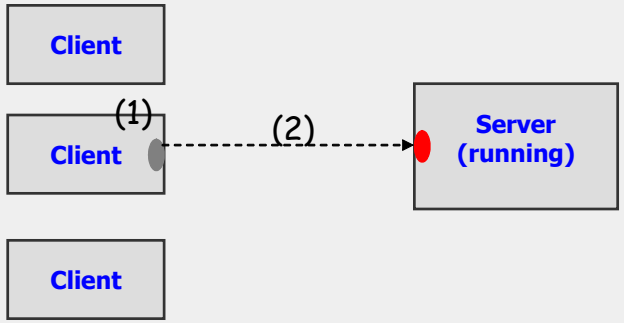


Principles of Concurrent Client-Server Communication with TCP

- **CLIENT**:

(1) create socket

(2) connect



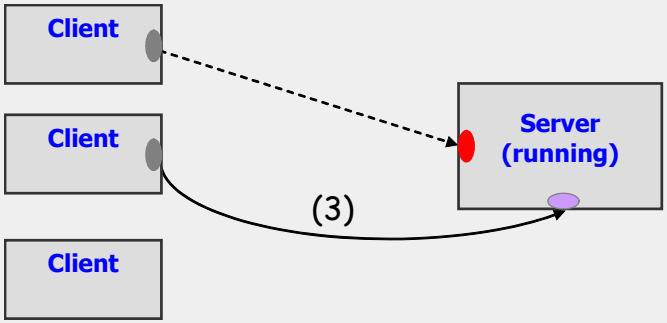
- **SERVER**:

(1) create socket

(2) listen

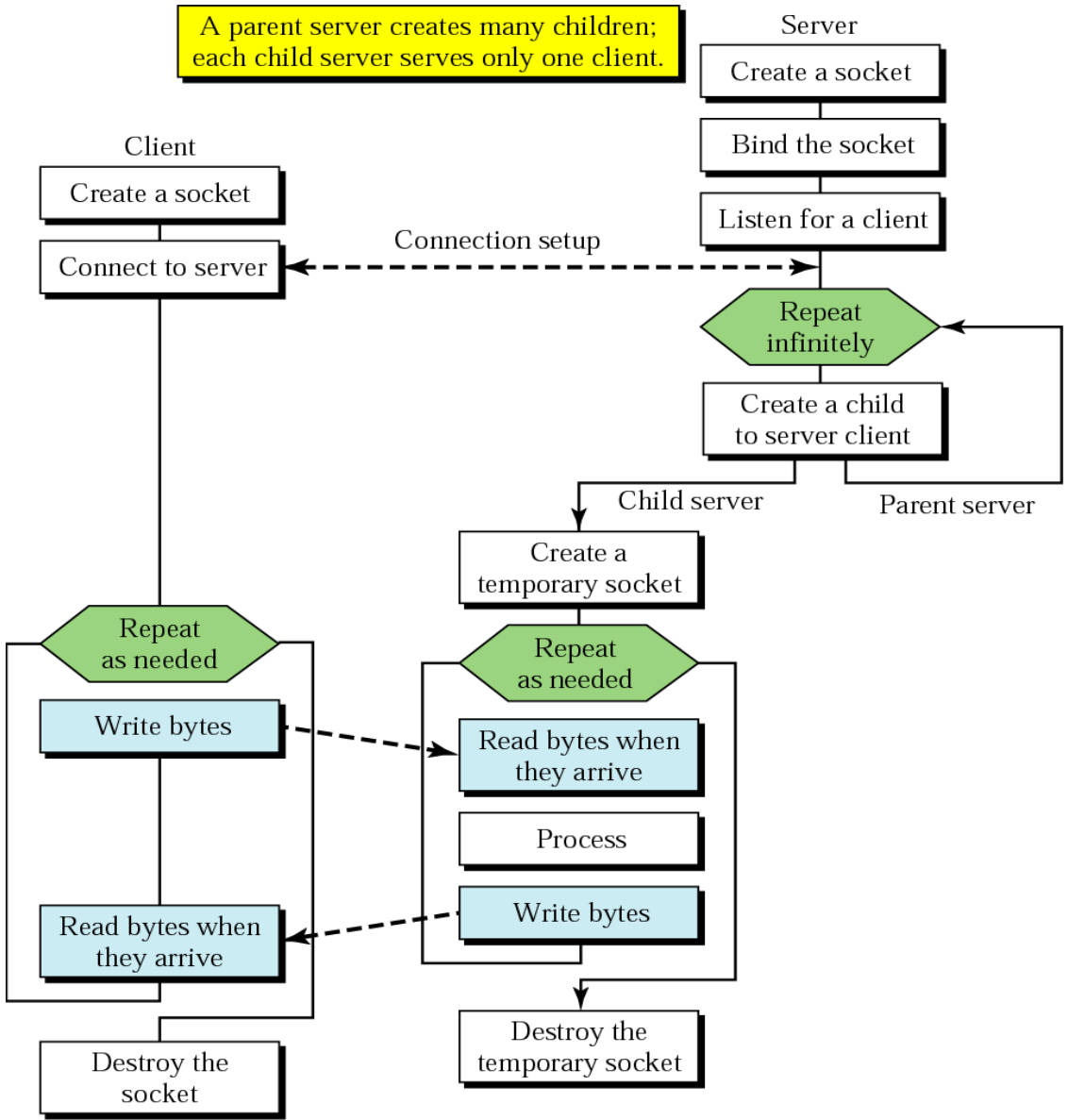
(3) repeat:
- (3.a) write
- (3.b) read
- (3.c) process

(4) destroy



(3) repeat
- (3.a) create new socket
- (3.b) repeat:
  - (3.b.1) read
  - (3.b.2) process
  - (3.b.3) write
- (3.c) destroy socket

Principles of Client-Server Communication with TCP (cont.)



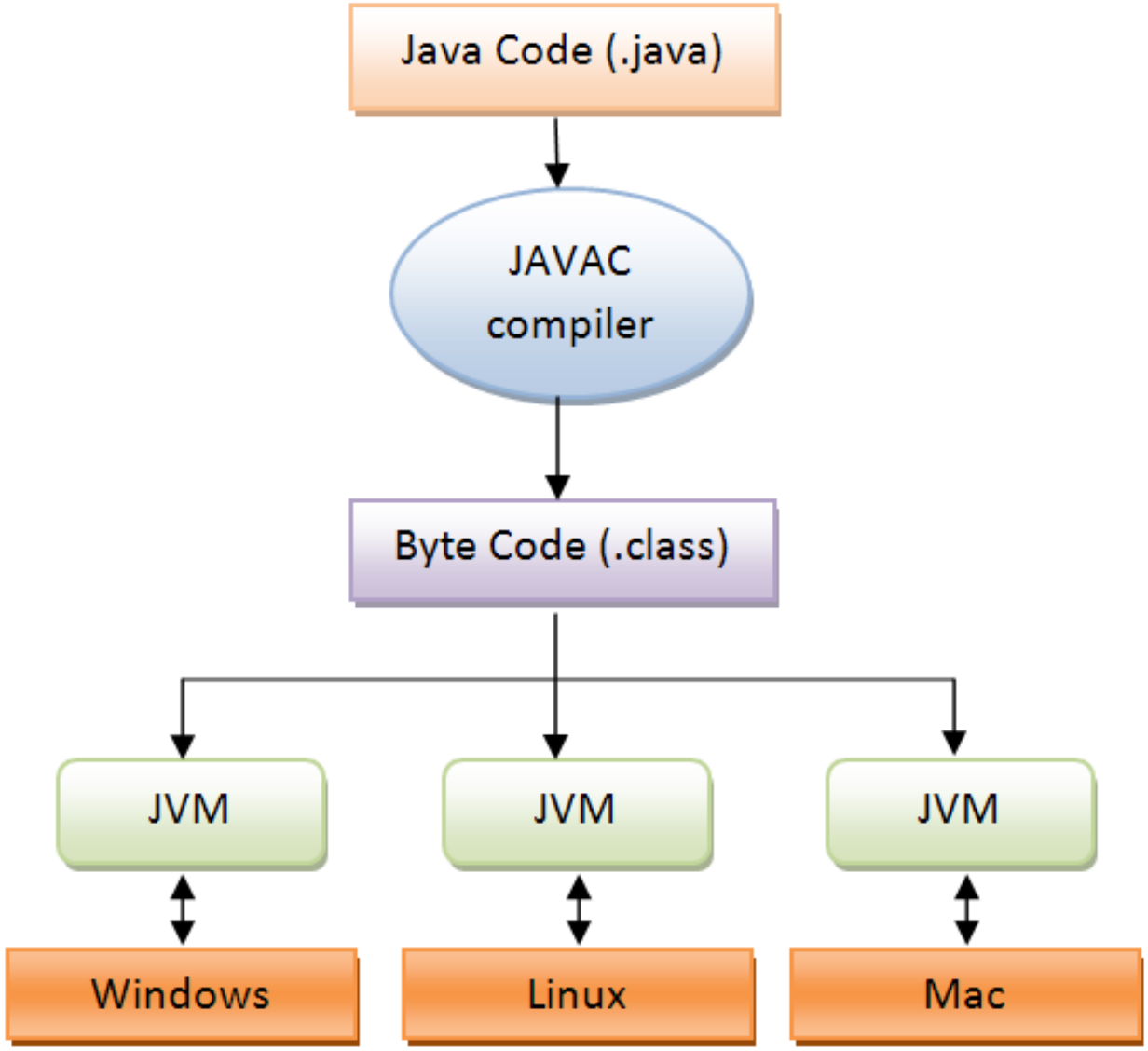
Java Socket Programming

Advantages of Socket Programming in Java

- **applications are more neatly and cleanly written in Java than in C or C++**
  - **there are fewer lines of code and each line can be explained to novice programmer without much difficulty**
- **Java keeps all socket transport-layer complexity "under the cover"**
  - **developer can focus on application rather than worrying about how network and transport layer operate**
- **Java does not rely on native code** ⇒ **programs can communicate over network (the Internet) in platform-independent fashion**

Disadvantages of Socket Programming in Java

- **Java does not expose the full range of socket possibilities to developer**


Java Code (.java)
JAVAC compiler
Byte Code (.class)
JVM
JVM
JVM
Windows
Linux
Mac

Example [Java vs. C socket programming]

C code to establish a socket

```
int set_up_socket(u_short port) {
    char myname[MAXHOSTNAME+1];
    int s;
    struct sockaddr_in sa;
    struct hostent *he;
    bzero(&sa,sizeof(struct sockaddr_in));          /* clear the address */
    gethostname(myname,MAXHOSTNAME);                /* establish identity */
    he= gethostbyname(myname);                      /* get our address */
    if (he == NULL)                                 /* if addr not found... */
        return(-1);
    sa.sin_family= he->h_addrtype;                  /* host address */
    sa.sin_port= htons(port);                       /* port number */
    if ((s= socket(AF_INET,SOCK_STREAM,0)) <0)      /* finally, create socket */
        return(-1);
    if (bind(s, &sa, sizeof(sa), 0) < 0) {
        close(s);
        return(-1);                                 /* bind address to socket */
    }
    listen(s, 3);                                   /* max queued connections */
    return(s);
}
```

Java code to establish a socket

```
ServerSocket servsock = new ServerSocket(port, backlog, bindAddr);
```

java.net package

InetAddress class	represents IP address – implements Serializable (2 subclasses **Inet4Address**, **Inet6Address** – final classes)
ServerSocket class	**passive TCP (server) socket** – used on server side to wait for client connection requests
Socket class	**active TCP socket** – can be used as communication end point both on client and server side
DatagramSocket class	**connectionless (UDP) socket** – used for sending and receiving datagrams (packets that are individually addressed and routed)
DatagramPacket class	datagram packet – in addition to data also contains IP address and port information – **used in UDP**!
MulticastSocket class	**subclass of DatagramSocket** – can be used for sending and receiving packets to/from multiple users

https://docs.oracle.com/javase/7/docs/api/java/net/package-summary.html

```java
package com.javacodegeeks.examples;

import java.net.Inet4Address;
import java.net.UnknownHostException;

public class SimpleInet4AddressExample {

    public static void main(String[] args) {
        String url = "javacodegeeks.com";

        try {
            Inet4Address address = (Inet4Address) Inet4Address.getByName(url);

            System.out.println("The IP of "+url+" is "+address.getHostAddress());
        } catch (UnknownHostException e) {
            e.printStackTrace();
        }


    }

}
```

getByAddress

```
public static InetAddress getByAddress(String host,
                       byte[] addr)
                                throws UnknownHostException
```

Creates an InetAddress based on the provided host name and IP address. No name service is checked for the validity of the address.

The host name can either be a machine name, such as "`java.sun.com`", or a textual representation of its IP address.

No validity checking is done on the host name either.