
Network Applications: Network Programming: UDP, TCP

Qiao Xiang

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

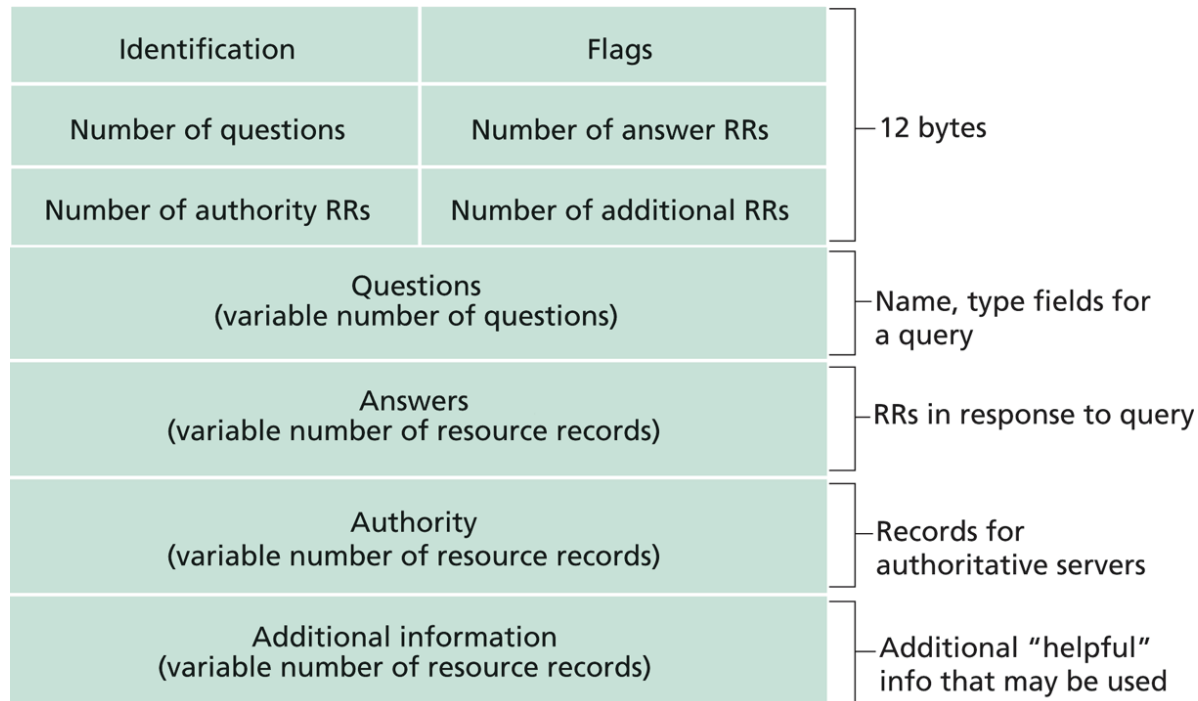
10/07/2021

Outline

- ❑ Admin. and recap
- ❑ Basic network applications
 - Email
 - DNS
- ❑ Java in a Nutshell
- ❑ Network application programming
 - UDP sockets

Recap: DNS Protocol, Messages

Many features: typically over **UDP** (can use TCP); *query* and *reply* messages with the **same** message format; *length/content encoding* of names; simple *compression*; additional info as *server push*



Name Encoding

▼ Queries

▼ xmu.edu.cn: type A, class IN

Name: xmu.edu.cn

[Name Length: 10]

[Label Count: 3]

Type: A (Host Address) (1)

Class: IN (0x0001)

[\[Response In: 3\]](#)

0000	34 71 46 af 81 aa 08 00	27 ba b3 67 08 00 45 00	4qF..... ' ..g..E.
0010	00 38 02 d4 40 00 40 11	b0 5b c0 a8 03 34 c0 a8	.8..@.@. .[...4..
0020	03 01 9b 8f 00 35 00 24	87 bb ed 00 01 00 00 015.\$
0030	00 00 00 00 00 00 03 78	6d 75 03 65 64 75 02 63x mu.edu.c
0040	6e 00 00 01 00 01		n.....

Diagram illustrating the Name Encoding for xmu.edu.cn. The table shows the hexadecimal representation of the name, with labels and their lengths indicated by red brackets and arrows:

- cn**: Label at offset 0040, length 2 (01 00).
- xmu**: Label at offset 0030, length 5 (03 78 6d 75 03).
- edu**: Label at offset 0020, length 5 (03 01 9b 8f 00).
- length**: Label at offset 0010, length 2 (00 38).

Message Compression (Label Pointer)

Diagram illustrating Message Compression (Label Pointer) in a DNS response.

The header shows a 16-bit field with bits 1 and 1 set, indicating a pointer. The field is labeled "OFFSET".

▼ Domain Name System (response)

- Transaction ID: 0xed00
- Flags: 0x8180 Standard query response, No error
- Questions: 1
- Answer RRs: 1
- Authority RRs: 0
- Additional RRs: 0
- Queries
- Answers
 - xmu.edu.cn: type A, class IN, addr 210.34.0.35

Annotations:

- DNS start
- question
- Answer: offset 12

Packet capture data (hex and ASCII):

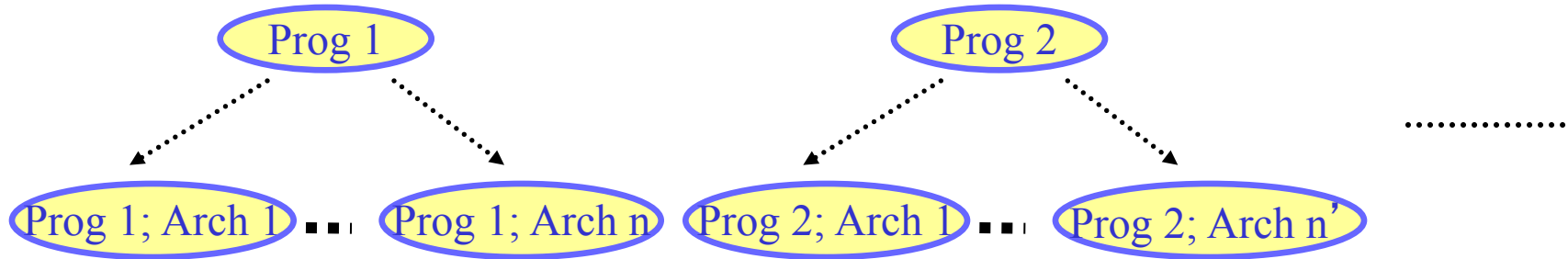
Offset	Hex	ASCII
0000	08 00 27 ba b3 67 34 71 46 af 81 aa 08 00 45 00	..'.g4q F...E.
0010	00 48 7f 0e 40 00 40 11 34 11 c0 a8 03 01 c0 a8	.H..@.@.4.....
0020	03 34 00 35 9b 8f 00 34 91 ee ed 00 81 80 00 01	.4.5...4.....
0030	00 01 00 00 00 00 03 78 6d 75 03 65 64 75 02 63x mu.edu.c
0040	6e 00 00 01 00 01 c0 0c 00 01 00 01 00 00 00 44	n.....D
0050	00 04 d2 22 00 23	...".#

Outline

- ❑ Admin. and recap
- ❑ Basic network applications
 - Email
 - DNS
- *Java in a Nutshell*

High-level Picture

C/C++



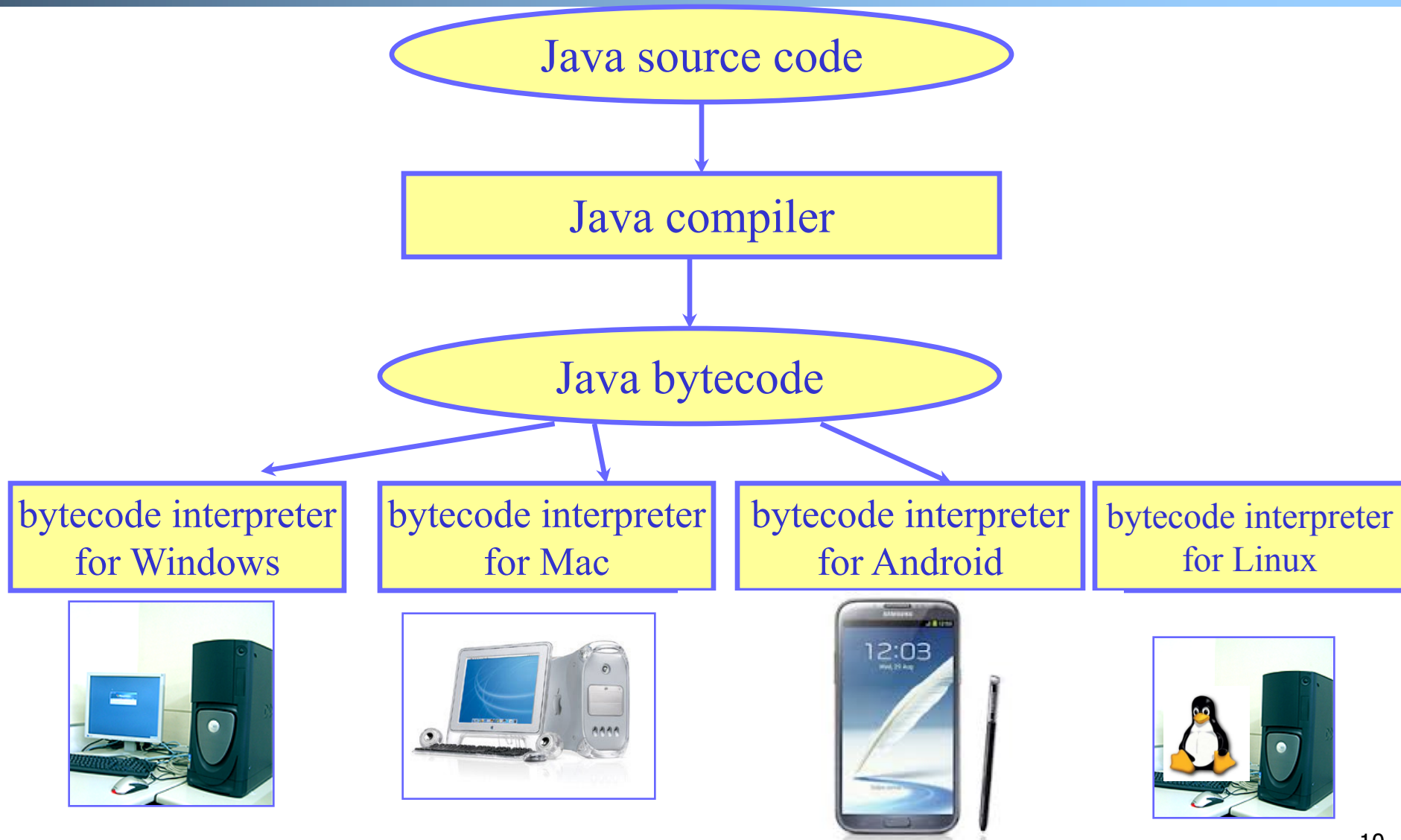
Java Virtual Machine

- ❑ To be platform independent, Java designers introduced Java Virtual Machine (JVM), a machine different from any physical platform, but a *virtual machine*
 - The language of the virtual machine is referred to as *bytecode*
 - Thus Java actually has two programming languages
- ❑ A Java compiler translates Java source code (.java files) into *bytecode* (in .class files)
 - Each Java software program needs to be compiled only once: from the Java source code to bytecode
- ❑ Other languages (e.g., Jruby, Jython, Scala) may also compile to bytecode

Java Execution

- ❑ To execute a Java program, another piece of software called an *interpreter*, translates between bytecode and the actual machine
 - an interpreter is specific to a specific platform
 - the interpreter understands java bytecode, and then issues instructions in the specific platform for which it is written
 - we also say that an interpreter provides a java virtual machine (JVM)

Java Translation and Execution



Comparing Traditional (e.g., C/C++) and Java Software Development

Traditional, e.g., C/C++

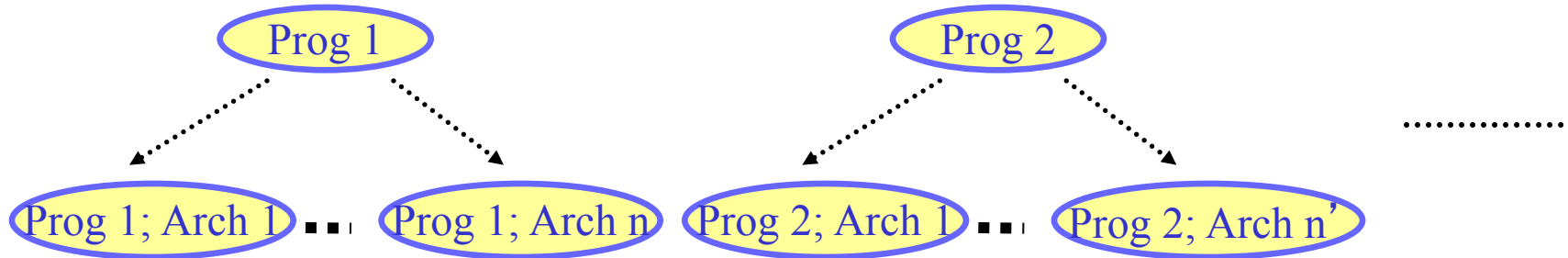
- ❑ A developer writes a program in C/C++
- ❑ The C/C++ source code is generally considered proprietary, and not released
- ❑ The developer compiles the C/C++ program for each platform it intends to support, and distributes one version for each platform
 - thus each program has **multiple compiled** versions
 - each compiled version can run by itself
- ❑ **Platform dependency handled by each software developer**

Java

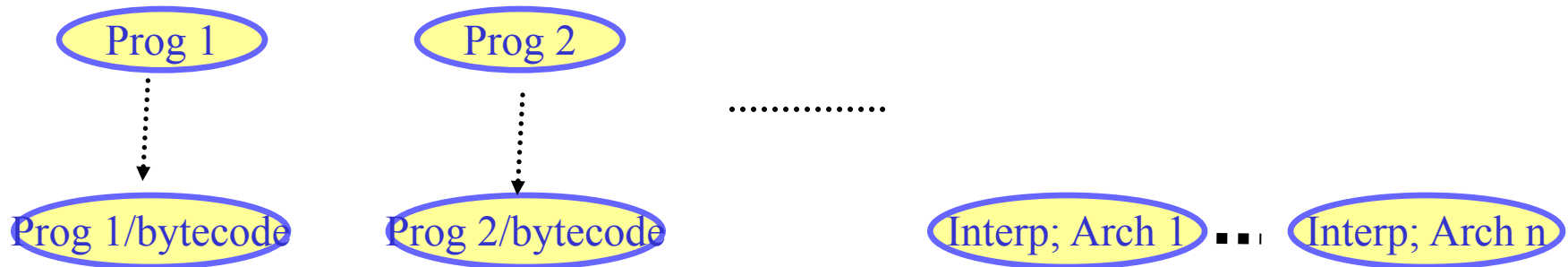
- ❑ A developer writes a program in Java
- ❑ The Java source code is generally considered proprietary, and not released
- ❑ The developer compiles the Java program to bytecode, and distributes the bytecode version
 - thus each program has only **one compiled version**
 - the compiled bytecode needs an interpreter for each platform
- ❑ **Platform dependency handled by platform vendor**

High-level Picture

C/C++

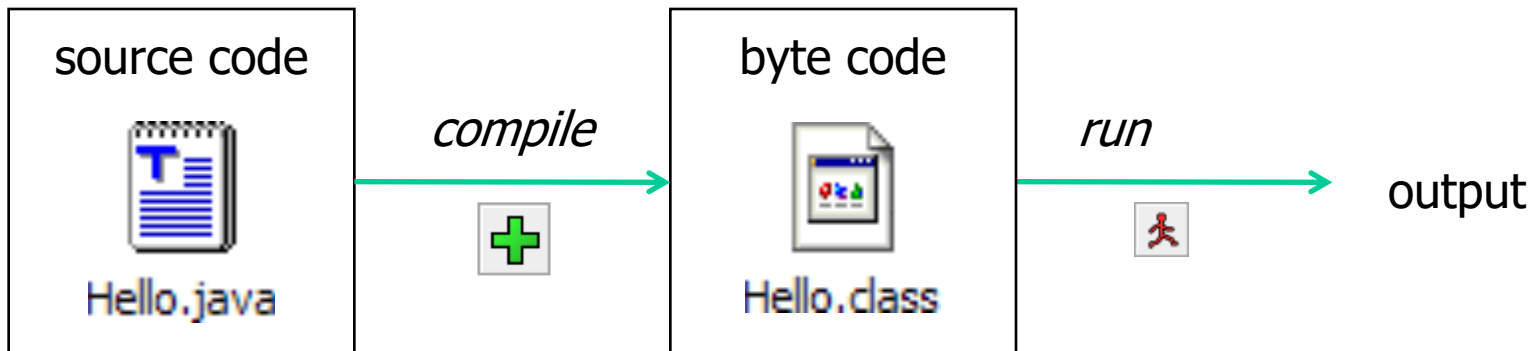


Java



Recall: Java Programming Steps

- ❑ Programming in Java consists of 3 simple steps
 - Create and edit "Java source code" (.java files)
 - Compile into "Java bytecode" (.class files)
 - Execute bytecode with a "Java interpreter"



Programming in Java (Step 1): Create/Edit

- ❑ The basic way is to use a text editor
 - Example editors: vim, sublime, Notepad, TextEdit (Format/Make Plain Text) etc.
 - **Note: MS Word is NOT a text editor**
 - The key is that your .java file *cannot* include any markup or stylistic formatting; just text.
 - You enter your Java code following Java Language syntax (more soon).

Programming in Java (Step 2): Compile

- ❑ Compile a Java program

`$ javac HelloWorld.java`

- ❑ Take a look to see that HelloWorld.class is generated

`$ ls`

`HelloWorld.java HelloWorld.class`

Programming in Java (Step 3): Execute

- ❑ Run Java interpreter

\$ java HelloWorld

First Java Program

```
/******  
 * Prints "Hello World"  
 * Everyone's first Java program.  
 *****/  
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello, world!");  
    }  
}
```

Another Java Program

```
public class Hello2 {  
    public static void main(String[] args) {  
        System.out.println("Hello, world!");  
        System.out.println();  
        System.out.println("This program produces");  
        System.out.println("four lines of output");  
    }  
}
```

Programming in Java: Method 2

- ❑ Another way is to use an Integrated Development Environment (IDE)
 - Example IDEs: Eclipse, IDEA, DrJava, etc.
 - An IDE usually presents the user with a space for text (like an editor) but layers *additional features* on top of the text for the user's benefit.
 - **Note: The underlying file contains pure text, just like a text editor.**
 - These features can be very useful and save time.
 - Example features are GUI compile, GUI execution, code completion, and syntax highlighting.
 - IDEs take more time to get started than a simple text editor, e.g.,
 - set up where to find the "java" and "javac" programs
 - find out where does the IDE save my file

Java Syntax Structure: A Top-Down View

A class:

- has a name, defined in a **file with same name**
Convention we follow: capitalize each English word
- starts with {, and ends with }
- includes a group of methods

```
public class <class name> {  
    public static void main(String[] args) {
```

```
        <statement>;
```

```
        <statement>;
```

```
        ...
```

```
        <statement>;
```

```
    }
```

```
}
```

A method:

- has a name
Convention we follow: lowercase first word, capital following
- starts with {, and ends with }
- includes a group of statements

statement:

- a command to be executed
- end with ;

The `System.out.println` statement

- ❑ A statement that prints a line of output on the console.
 - pronounced "print-linn"
- ❑ Two ways to use `System.out.println` :
 - `System.out.println(<string>);`
Prints the given message `<string>` as output.
 - `System.out.println();`
Prints a blank line of output.

Java program structure

- ❑ A top-down view
- ❑ A bottom-up view

Java Syntax: A Bottom-Up View

```
// Comment 1: A Java program
/* Comment 2: a long comment
*****/
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
        System.out.println();
        System.out.println("This program produces");
        System.out.println("four lines of output");
    }
}
```

Java Syntax: A Bottom-Up View

```
// Comment 1: A Java program
/* Comment 2: a long comment
*****/
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
        System.out.println();
        System.out.println("This program produces");
        System.out.println("four lines of output");
    }
}
```

- ❑ Basic Java syntax units
 - white space and comments
 - identifiers (words)
 - symbols: { } " () < > [] ; = ...
 - strings
 - numbers

Syntax: White Space

- ❑ White space
 - includes spaces, new line characters, tabs
 - white space is used to separate other entities
 - extra white space is ignored

- ❑ White space allows a Java program to be formatted in many ways, and should be formatted to enhance readability
 - the usage of white space forms part of programming style

Syntax: Comments

- ❑ **comment:** A note written in source code by the programmer to describe or clarify the code.
 - Comments are ignored by the compiler
 - Useful for other people (and yourself!) to understand your code

- ❑ **Two types of comments in Java**
 - single-line comments use `//...`
`// this comment runs to the end of the line`
 - multi-lines comments use `/* ... */`
`/* this is a very long
multi-line comment */`

Syntax: Identifier

- ❑ **Identifier:** A name given to an item in a program.

- ❑ **Syntax requirement on identifier:**
 - must start with a letter or `_` or `$`
 - subsequent characters can be any of those or a number
 - Important: Java is case sensitive:
 - Hello and hello are different identifiers

Three Types of Identifiers

1. Identifiers chosen by ourselves when writing a program (such as `HelloWorld`)
2. Identifiers chosen by another programmer, so we use the identifiers that they chose (e.g., `System`, `out`, `println`, `main`)

```
public class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println("Hello World!");
    }
}
```

Three Types of Identifiers

3. Special identifiers called *keywords* or *reserved words*: A keyword has a special meaning in Java.

abstract	default	if	private	this
boolean	do	implements	protected	throw
break	double	import	public	throws
byte	else	instanceof	return	transient
case	extends	int	short	try
catch	final	interface	static	void
char	finally	long	strictfp	volatile
class	float	native	super	while
const	for	new	switch	
continue	goto	package	synchronized	

Java reserved words: they are all **lowercase!**

Syntax: Strings

- ❑ **string:** A sequence of characters that starts and ends with a " (quotation mark character).

- The quotes do not appear in the output.

- Examples:

- `"hello"`

- `"This is a string. It is very long!"`

- ❑ **Restrictions:**

- May not span multiple lines

- `"This is not
a legal String."`

Examples

❑ Which of the following are legal strings in Java?

- `"This is a string. It's very long!"`
- `"This cool string spans
two lines. "`
- `"It is a great thing when children cry, "I
want my mommy"! "`

Escape Sequences

- ❑ **escape sequence:** A special sequence of characters used to represent certain special characters in a string.

<code>\b</code>	backspace
<code>\t</code>	tab character
<code>\n</code>	new line character
<code>\"</code>	quotation mark character
<code>\\</code>	backslash character

- **Example:**

```
System.out.println("\\hello\\nhow\\tare  \"you\"?\\\\\\");
```

- **Output:**

```
\\hello
how      are  "you"?\\
```


Comment on syntax errors

- ❑ **A syntax/compile error:** A problem in the structure of a program that causes the compiler to fail, e.g.,
 - Missing semicolon
 - Too many or too few { } braces
 - Class and file names do not match
 - ...
- ❑ Compilers can't (DO not) read minds.
- ❑ Compilers don't make mistakes.
- ❑ If the program is not doing what you want, do NOT blame the computer---it's **YOU** who made a mistake.

Socket Programming

Socket API

- ❑ introduced in BSD4.1 UNIX, 1981
- ❑ Two types of sockets
 - connectionless (UDP)
 - connection-oriented (TCP)

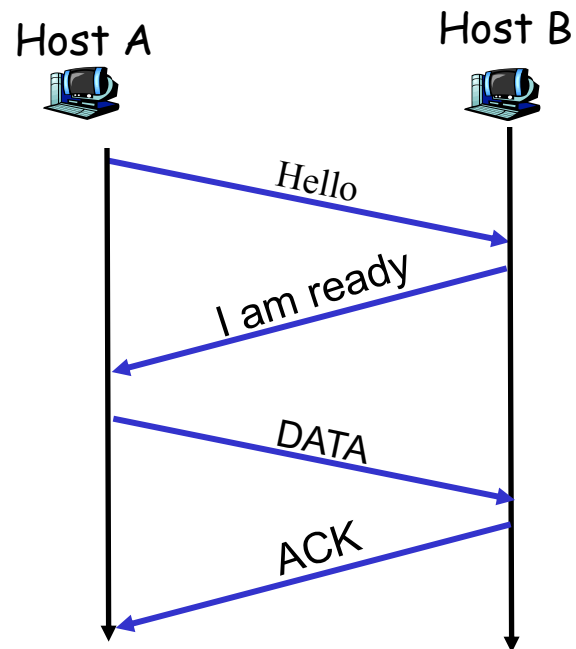
socket

an interface (a “door”) into which one application process can **both send and receive** messages to/from another (remote or local) application process

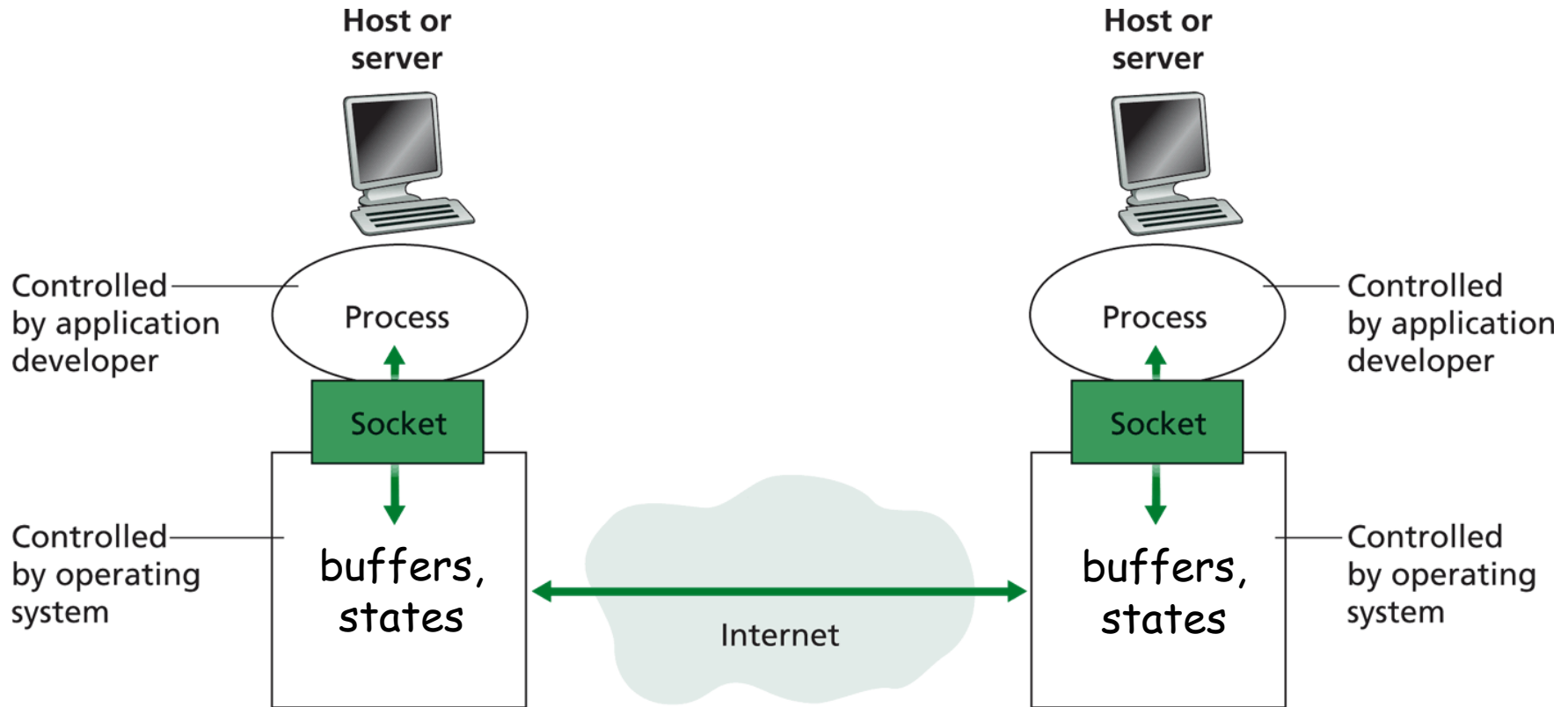
Services Provided by Transport

- User data protocol (UDP)
 - multiplexing/demultiplexing

- Transmission control protocol (TCP)
 - multiplexing/demultiplexing
 - reliable data transfer
 - rate control: flow control and congestion control



Big Picture: Socket



Outline

- ❑ Admin. and recap
- ❑ Basic network application programming
 - Overview
 - *UDP (Datagram Socket)*

DatagramSocket (Java) (Basic)

- ❑ **DatagramSocket()**
constructs a datagram socket and binds it to any available port on the local host
- ❑ **DatagramSocket(int lport)**
constructs a datagram socket and binds it to the specified port on the local host machine.
- ❑ **DatagramPacket(byte[] buf, int length)**
constructs a DatagramPacket for receiving packets of length length.
- ❑ **DatagramPacket(byte[] buf, int length, InetAddress address, int port)**
constructs a datagram packet for sending packets of length length to the specified port number on the specified host.
- ❑ **receive(DatagramPacket p)**
receives a datagram packet from this socket.
- ❑ **send(DatagramPacket p)**
sends a datagram packet from this socket.
- ❑ **close()**
closes this datagram socket.

Connectionless UDP: Big Picture (Java version)

Server (running on `serv`)

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`

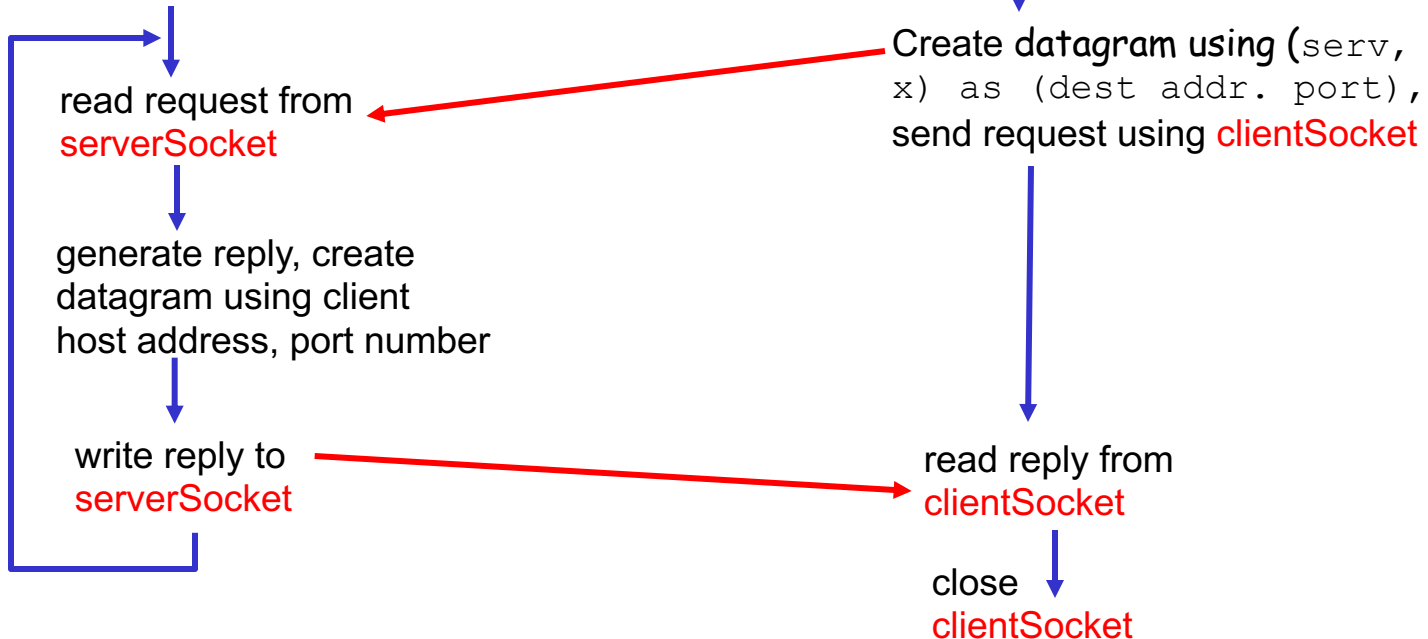
Client

create socket,
`clientSocket =`
`DatagramSocket()`

Create datagram using (`serv`,
`x`) as (`dest addr. port`),
send request using `clientSocket`

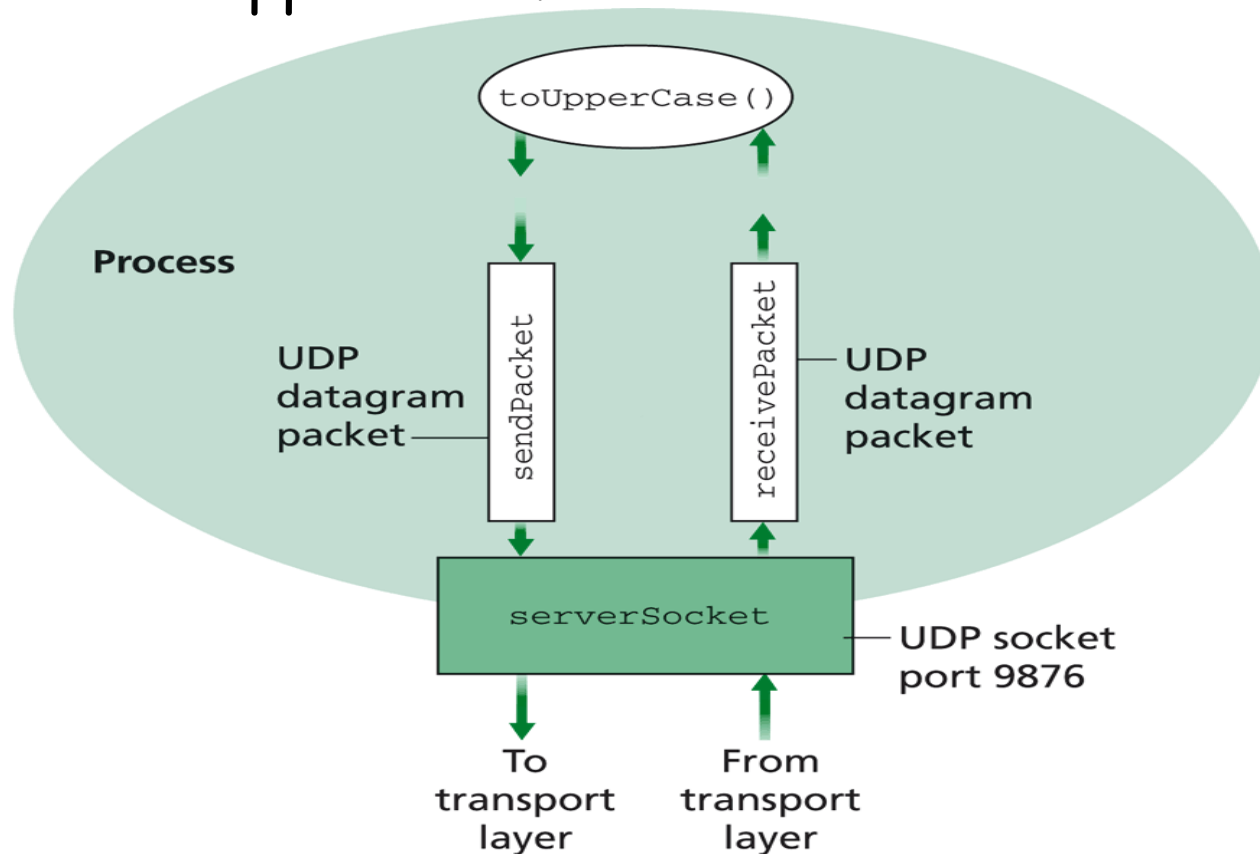
read reply from
`clientSocket`

close
`clientSocket`



Example: UDPServer.java

- A simple UDP server which changes any received sentence to upper case.



Java Server (UDP): Create Socket

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

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

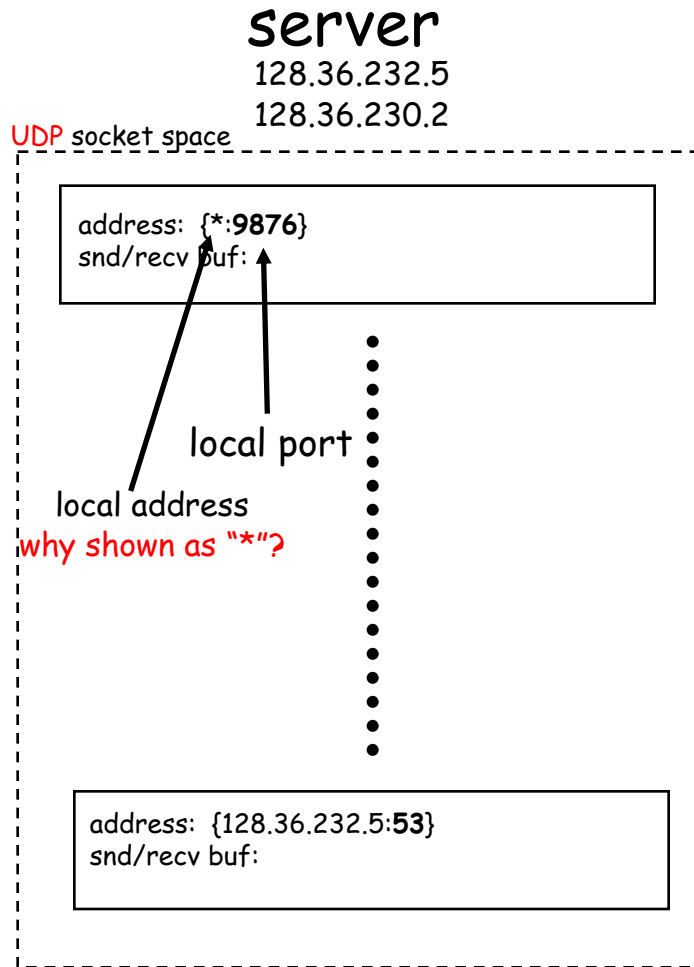
Create
datagram socket
bind at port 9876



```
DatagramSocket serverSocket = new DatagramSocket(9876);
```

Check socket state:
%netstat -a -u -n

System State after the Call



"*" indicates that the socket binds to **all** IP addresses of the machine:

% ifconfig -a

Binding to Specific IP Addresses

server

Public address: 128.36.59.2

Local address: 127.0.0.1

UDP socket space

address: {127.0.0.1:9876}
snd/rcv buf:

address: {128.36.59.2:9876}
snd/rcv buf:

address: {*:6789}
snd/rcv buf:

⋮

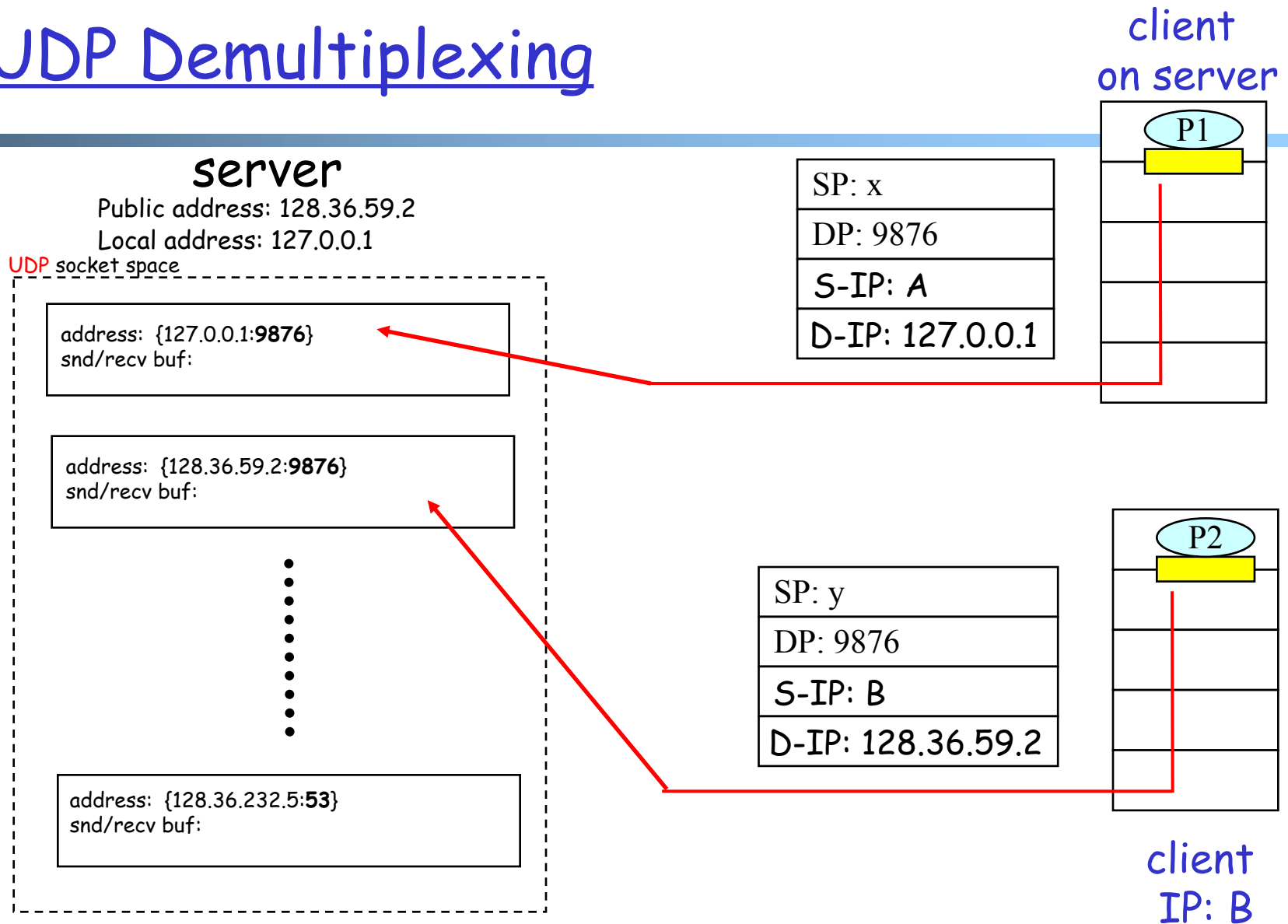
address: {128.36.232.5:53}
snd/rcv buf:

```
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);
```

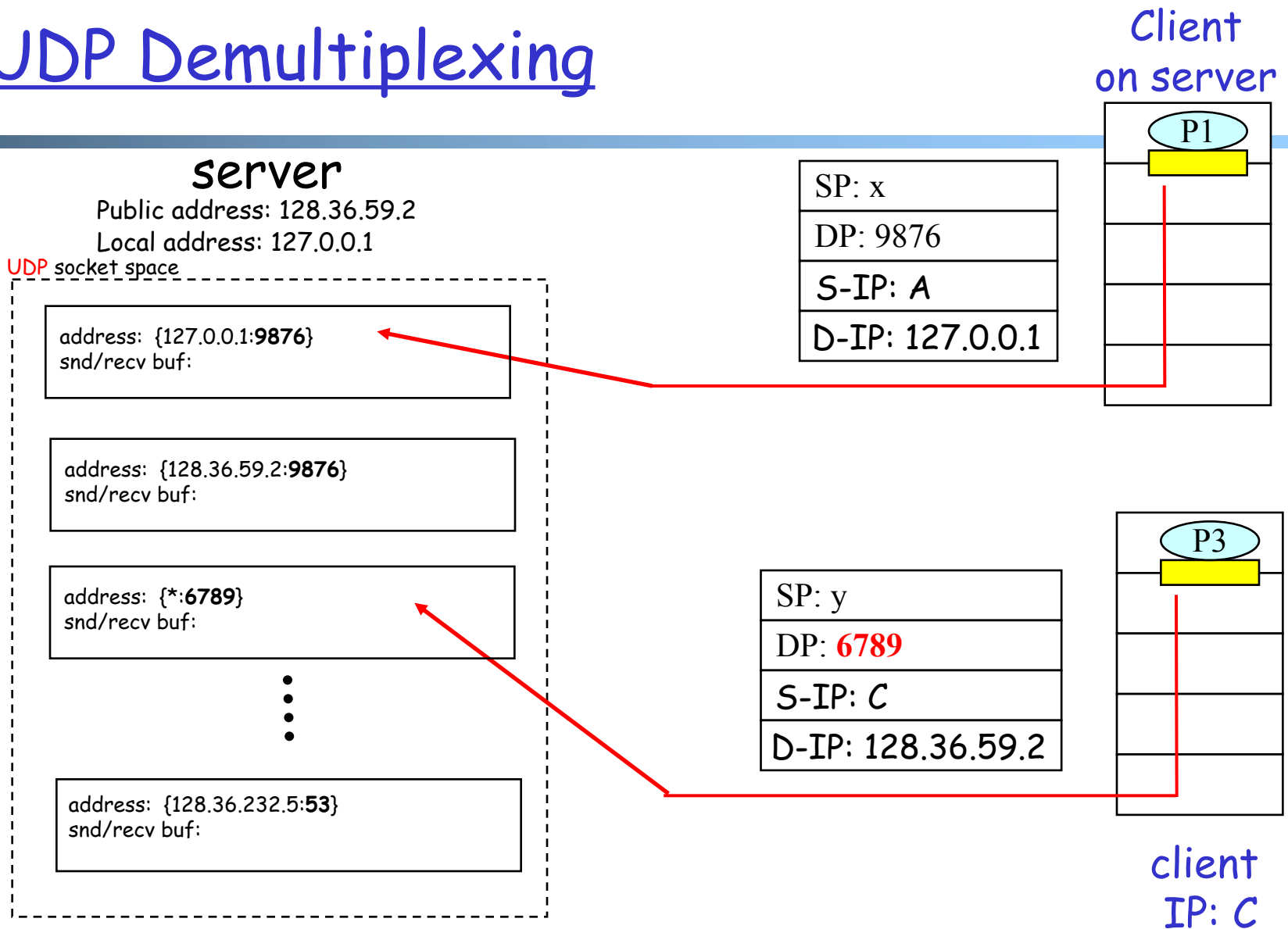
```
DatagramSocket serverSocket = new  
    DatagramSocket(6789);
```

UDP Demultiplexing



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

UDP Demultiplexing



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

Per Socket State

- ❑ Each Datagram socket has a set of states:
 - local address
 - send buffer size
 - receive buffer size
 - timeout
 - traffic class

See

<http://download.java.net/jdk7/archive/b123/docs/api/java/net/DatagramSocket.html>

Example: socket state after clients sent msgs to the server

Java Server (UDP): Receiving

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

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

```
        DatagramSocket serverSocket = new DatagramSocket(9876);
```

```
        byte[] receiveData = new byte[1024];  
        byte[] sendData = null;
```

```
        while(true)  
        {
```

Create space for
received datagram



```
            DatagramPacket receivePacket =  
                new DatagramPacket(receiveData, receiveData.length);
```

Receive
datagram



```
            serverSocket.receive(receivePacket);
```

DatagramPacket

❑ Receiving

- **DatagramPacket(byte[] buf, int length)**
constructs a DatagramPacket for receiving packets of length length.
- **DatagramPacket(byte[] buf, int offset, int length)**
constructs a DatagramPacket for receiving packets starting at offset, length length.

❑ Sending

- **DatagramPacket(byte[] buf, int length, InetAddress address, int port)**
constructs a datagram packet for sending packets of length length to the specified port number on the specified host.
- **DatagramPacket(byte[] buf, int offset, int length, InetAddress address, int port)**

Java Server (UDP): Processing

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

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

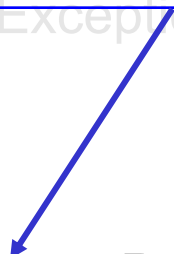
```
        ...
```

```
        // process data
```

```
        String sentence = new String(receivePacket.getData(),  
                                     0, receivePacket.getLength());
```

```
        String capitalizedSentence = sentence.toUpperCase();  
        sendData = capitalizedSentence.getBytes();
```

getData() returns a pointer to an underlying buffer array;
for efficiency, don't assume receive() will reset the rest of the array



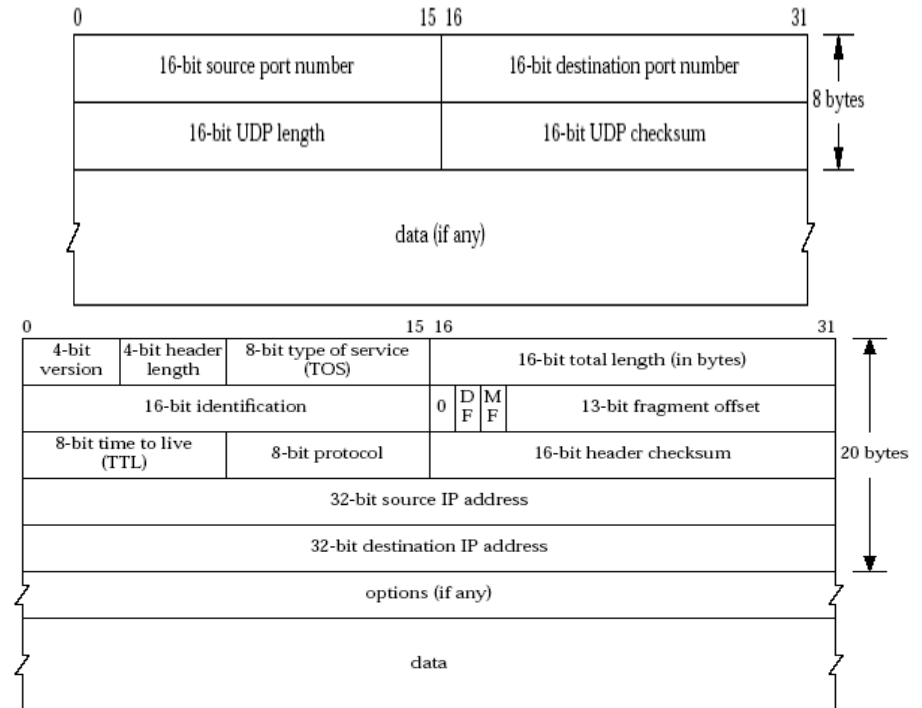
getLength() returns how much data is valid.



Java Server (UDP): Response

❑ Java DatagramPacket:

- getAddress() / getPort() returns the **source** address/port



Java server (UDP): Reply

Get IP addr
port #, of
sender

```
InetAddress IPAddress = receivePacket.getAddress();  
int port = receivePacket.getPort();
```

Create datagram
to send to client

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

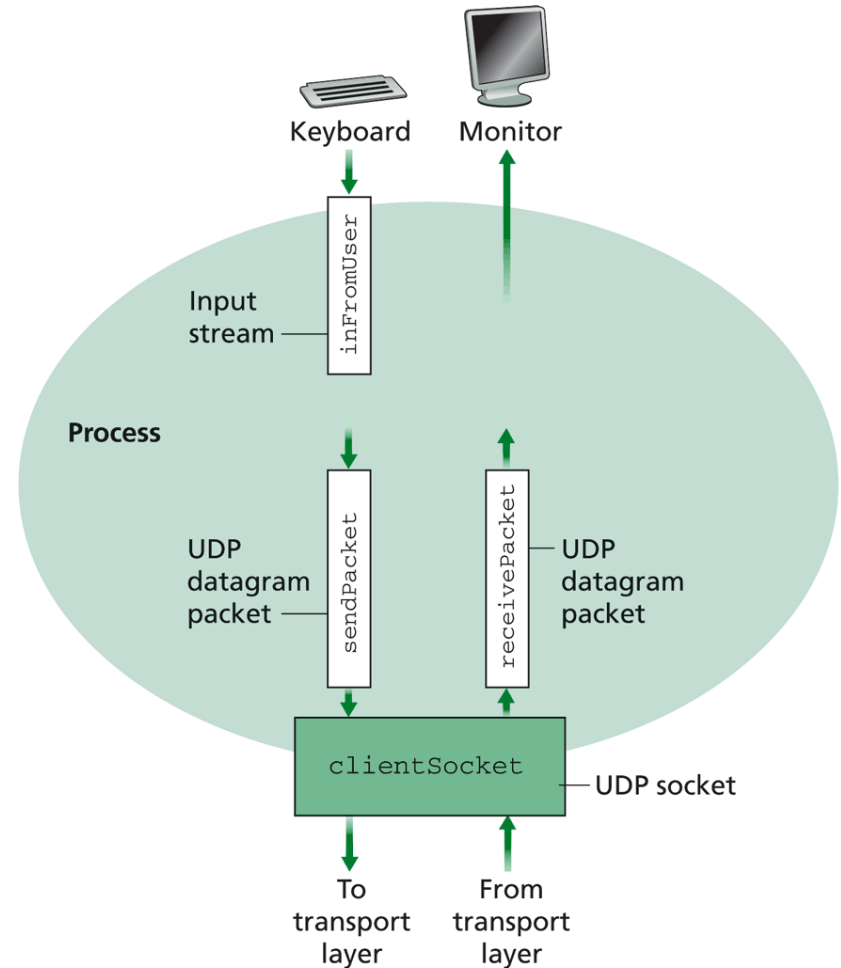
Write out
datagram
to socket

```
serverSocket.send(sendPacket);  
}
```

End of while loop,
loop back and wait for
another datagram

Example: UDPClient.java

- ❑ A simple UDP client which reads input from keyboard, sends the input to server, and reads the reply back from the server.



Example: Java client (UDP)

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

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

Create
input stream

```
        BufferedReader inFromUser =  
            new BufferedReader(new InputStreamReader(System.in));  
        String sentence = inFromUser.readLine();  
        byte[] sendData = sentence.getBytes();
```

Create
client socket

```
        DatagramSocket clientSocket = new DatagramSocket();
```

Translate
hostname to IP
address using DNS

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

Example: Java client (UDP), cont.

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

Send datagram
to server

Read datagram
from server

```
DatagramPacket sendPacket =  
    new DatagramPacket(sendData, sendData.length, sIPAddress, 9876);  
  
clientSocket.send(sendPacket);  
  
byte[] receiveData = new byte[1024];  
DatagramPacket receivePacket =  
    new DatagramPacket(receiveData, receiveData.length);  
  
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?