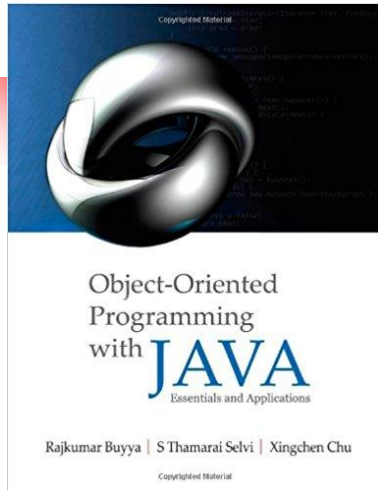# Inter-Process Communication (IPC): Network Programming using TCP Java Sockets

Dr. Rajkumar Buyya

Cloud Computing and **D**istributed **S**ystems (CLOUDS) Laboratory
School of Computing and Information Systems
The University of Melbourne, Australia
http://www.buyya.com

Agenda

- Introduction
- Communication Basics
- Networking Basics
- Understanding Ports and Sockets
- Java Sockets
  - Implementing a Server
  - Implementing a Client
- Sample Examples
- Conclusions

Introduction

- Internet and WWW have emerged as global ubiquitous media for communication and are changing the way we conduct science, engineering, and commerce

- They are also changing the way we learn, live, enjoy, communicate, interact, engage, etc. It appears like the modern life activities are getting completely centered around the Internet

Internet Applications Serving Local and Remote Users

PC client

Internet Server

Local Area Network

PDA
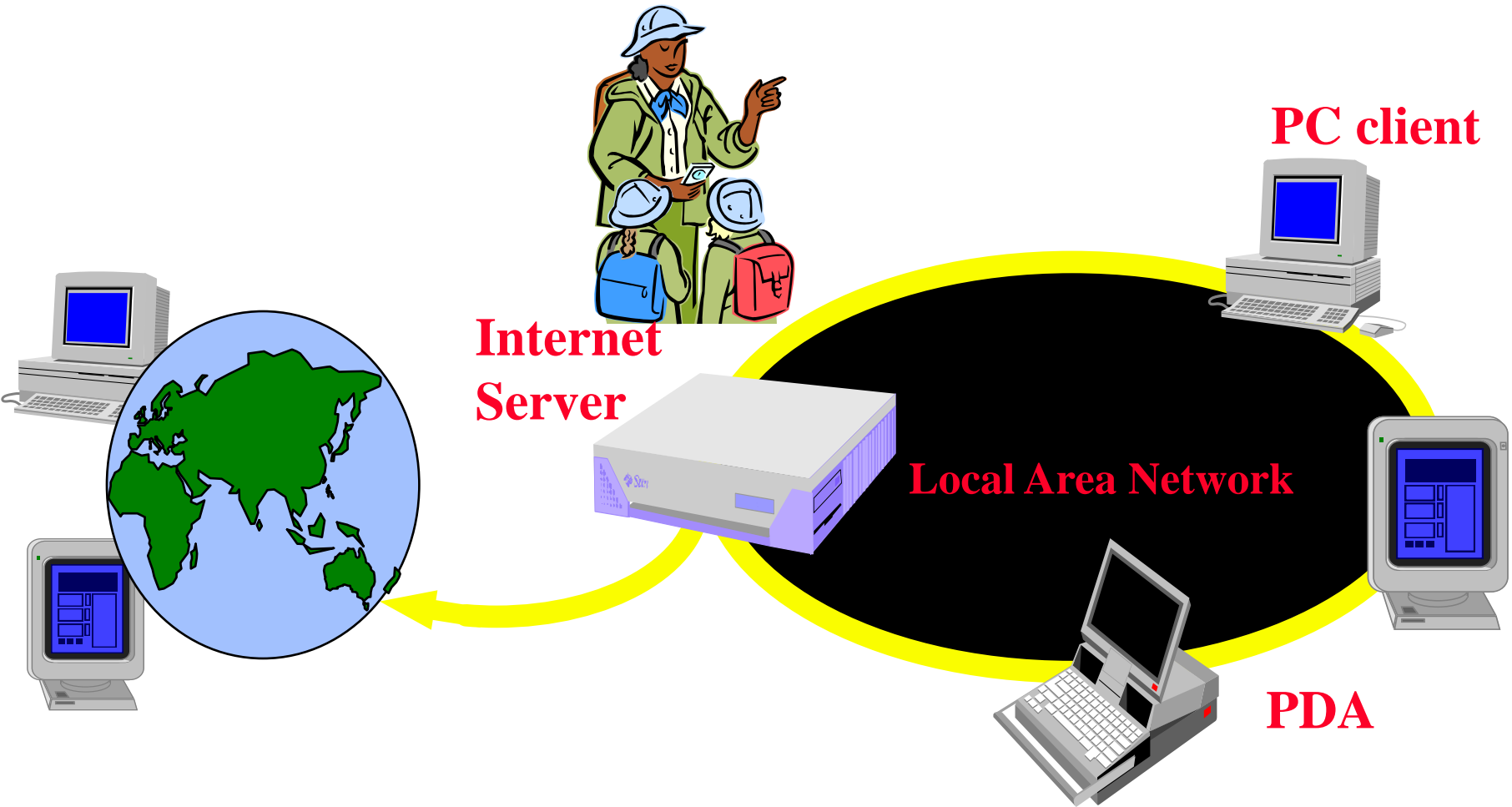
Increasing Demand for Internet Applications

- To take advantage of opportunities presented by the Internet, businesses are continuously seeking new and innovative ways and means for offering their services via the Internet.

- This created a huge demand for software designers with skills to create new Internet-enabled applications or migrate existing/legacy applications to the Internet platform.

- Object-oriented Java technologies—Sockets, threads, RMI, clustering, Web services—have emerged as leading solutions for creating portable, efficient, and maintainable large and complex Internet applications.

Elements of Client-Server Computing/Communication

a client, a server, and network
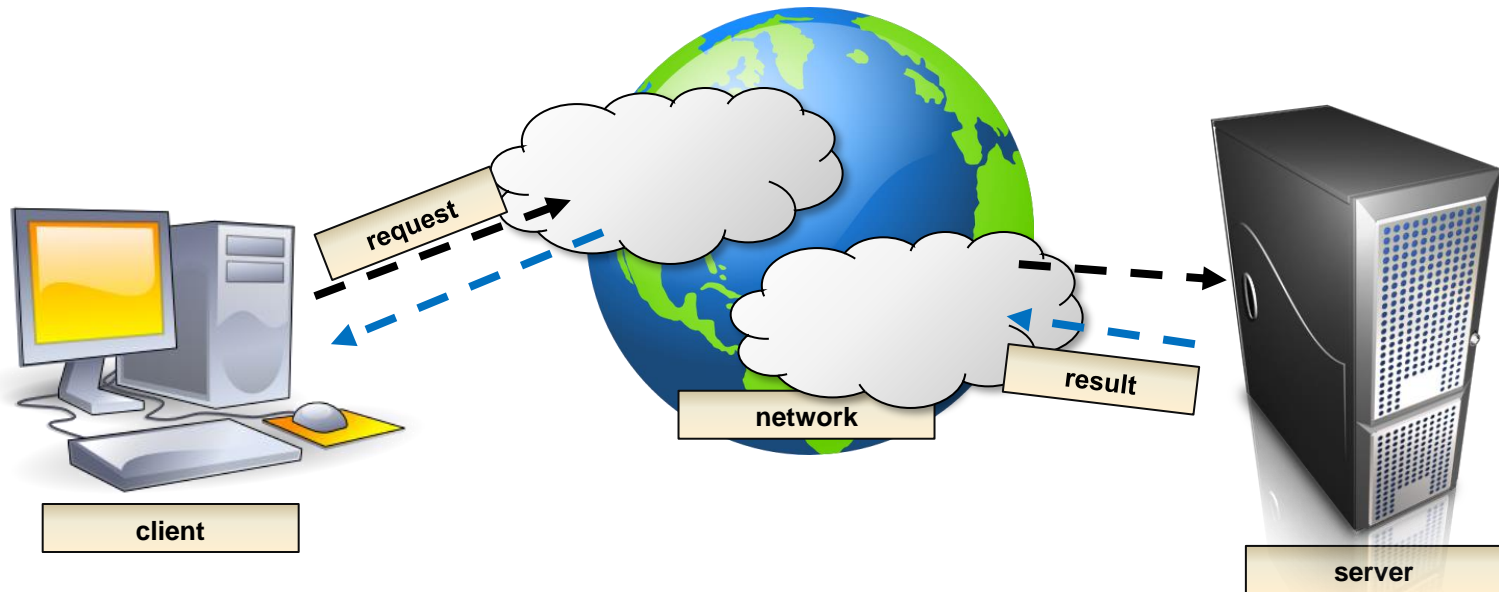
- Processes follow protocols that define a set of rules that must be observed by participants:
    - How the data exchange is encoded?
    - How events (sending, receiving) are synchronized (ordered) so that participants can send and receive data in a coordinated manner?
- In face-to-face communication, humans beings follow unspoken protocols based on eye contact, body language, gesture.

The Characteristics of Interprocess Communication

- **Message Communication Operations.** Message passing between a pair of processes can be supported by two message communication operations, send and receive.

- **Synchronous and asynchronous communication.** A queue is associated with each message destination. Sending processes cause messages to be added to remote queues and receiving processes remove messages from local queues.
  - Synchronous
  - Asynchronous

The Characteristics of Interprocess Communication

- **Reliability.** A point-to-point message service can be described as reliable if messages are guaranteed to be delivered despite a 'reasonable' number of packets being dropped or lost **(unreliable?)**

- **Ordering.** Some applications require that messages be delivered in *sender order*

Communication Basics

- Every distributed system has software components that communicate over a network.

- If a mobile banking app requests the user's current bank account balance, a (very simplified) sequence of communications occurs as follows

Communication Basics

1. The mobile banking app sends a request over the cellular network addressed to the bank to retrieve the user's bank balance.

2. The request is routed across the internet to where the bank's web servers are located.

3. The bank's web server authenticates the request and sends a request to a database server for the account balance.

Communication Basics

4. The database server reads the account balance from disk and returns it to the web server.

5. The web server sends the balance in a reply message addressed to the app, which is routed over the internet and the cellular network until the balance magically appears on the screen of the mobile device.

Is there any hidden complexity within this scenario?

Communication Hardware

- Different types of communication channels exist, and one possible categorization is wireless versus wired

- For physically wired networks, the two most common types are local area networks (LANs) and wide area networks (WANs).

- LANs are networks that can connect devices at "building scale," being able to transmit data over a small number (e.g., 1–2) of kilometers.

Communication Hardware

- WANs are networks that traverse the globe and make up what we call the internet.

- These long-distance connections are the high speed data pipelines connecting cities, countries, and continents with fiber optic cables.

Communication Hardware

- These cables support a networking technology known as wavelength division multiplexing which makes it possible to transmit up to 171 Gbps over 400 different channels.

- Such transmission gives more than 70 terabits per second (Tbps) of total bandwidth for a single fiber link.

- WANs transmit data over hundreds and thousands of kilometers.

14

Communication Software

- Software systems on the internet communicate using the Internet Protocol (IP) suite.

- The IP suite specifies host addressing, data transmission formats, message routing, and delivery characteristics.

Networking Basics

- **Physical/Link Layer**
  - Functionalities for transmission of signals representing a stream of data from one computer to another
- **Internet/Network Layer**
  - IP (Internet Protocols) – a packet of data to be addressed to a remote computer and delivered

- **TCP/IP Stack**

Application (http,ftp,telnet,…)
Transport (TCP, UDP,..)
Internet/Network (IP,..)
Physical/Link (device driver,..)

16
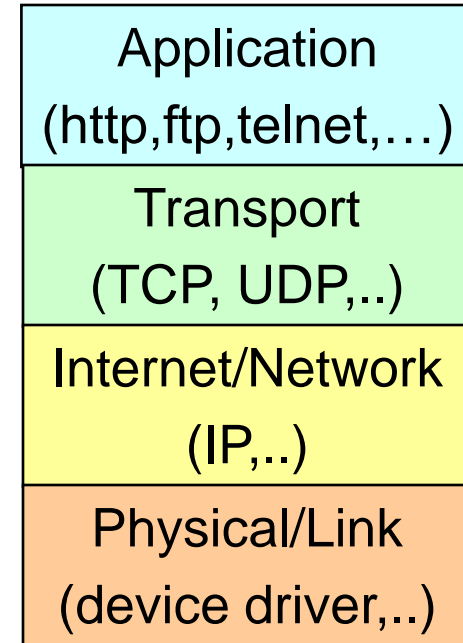
Internet Protocol

- IP is known as a best-effort delivery protocol as it does not attempt to compensate for the various error conditions that can occur during packet transmission.

- Possible transmission errors include data corruption, packet loss, and duplication.

- In addition, every packet is routed across the internet from source to destination independently.

Internet Protocol

- Treating every packet independently is known as packet switching.

- This allows the network to dynamically respond to conditions such as network link failure and congestion, and hence is a defining characteristic of the internet.

- Hence, different packets may be delivered to the same destination
via different network paths, resulting in out-of-order delivery to the receiver
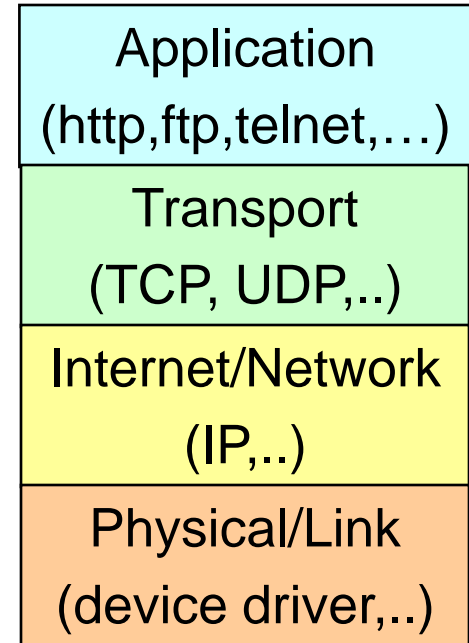
18

Networking Basics

- Transport Layer
  - Functionalities for delivering data packets to a specific process on a remote computer
  - TCP (Transmission Control Protocol)
  - UDP (User Datagram Protocol)
  - Programming Interface:
   - Sockets
- Applications Layer
  - Message exchange between standard or user applications:
    - HTTP, FTP, Telnet, **Skype,…**

- TCP/IP Stack

Application (http,ftp,telnet,…)
Transport (TCP, UDP,..)
Internet/Network (IP,..)
Physical/Link (device driver,..)

Networking Basics

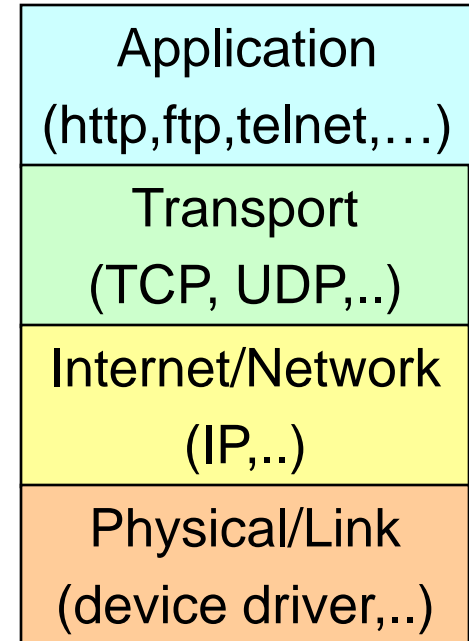- UDP (User Datagram Protocol) is a connectionless communication protocol that sends independent packets of data, called *datagrams*, from one computer to another with <u>no</u> guarantees about arrival or order of arrival

- Similar to sending multiple emails/letters to friends, each containing part of a message.

- Example applications:
  - Live streaming (event/sports broadcasting)

- TCP/IP Stack

Application (http,ftp,telnet,…)
Transport (TCP, UDP,..)
Network (IP,..)
Link (device driver,..)
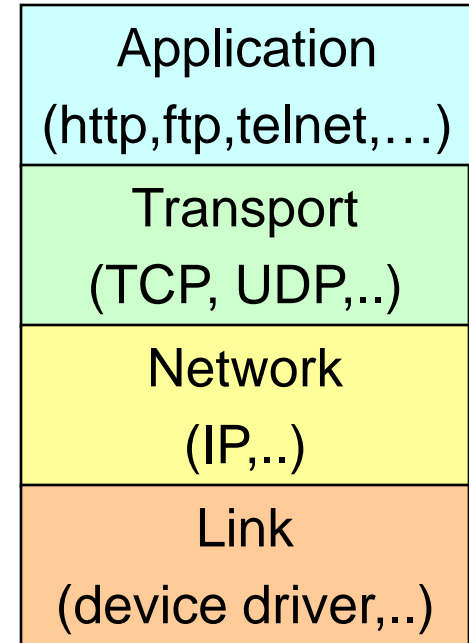
Networking Basics

- TCP (Transmission Control Protocol) is a connection-oriented communication protocol that provides a reliable flow of data between two computers.
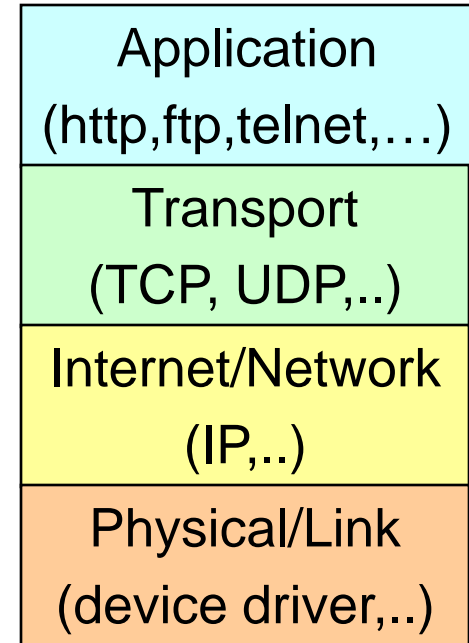
- Analogy: Speaking on Phone

- TCP/IP Stack

Application (http,ftp,telnet,…)
Transport (TCP, UDP,..)
Internet/Network (IP,..)
Physical/Link (device driver,..)

Networking Basics

- TCP (Transmission Control Protocol)

- Mechanisms to meet reliability guarantees:
  - Sequencing
  - Retransmission

- Example applications:
  - HTTP, FTP, Telnet
  - **Skype** uses **TCP** for call signalling, and both **UDP** and **TCP** for transporting media traffic.

- TCP/IP Stack
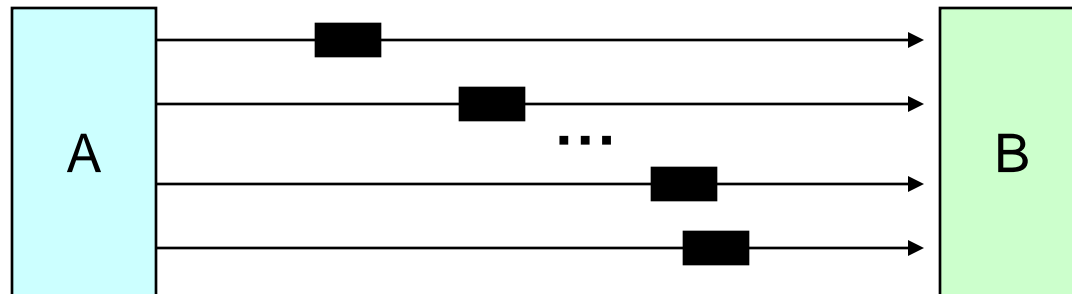
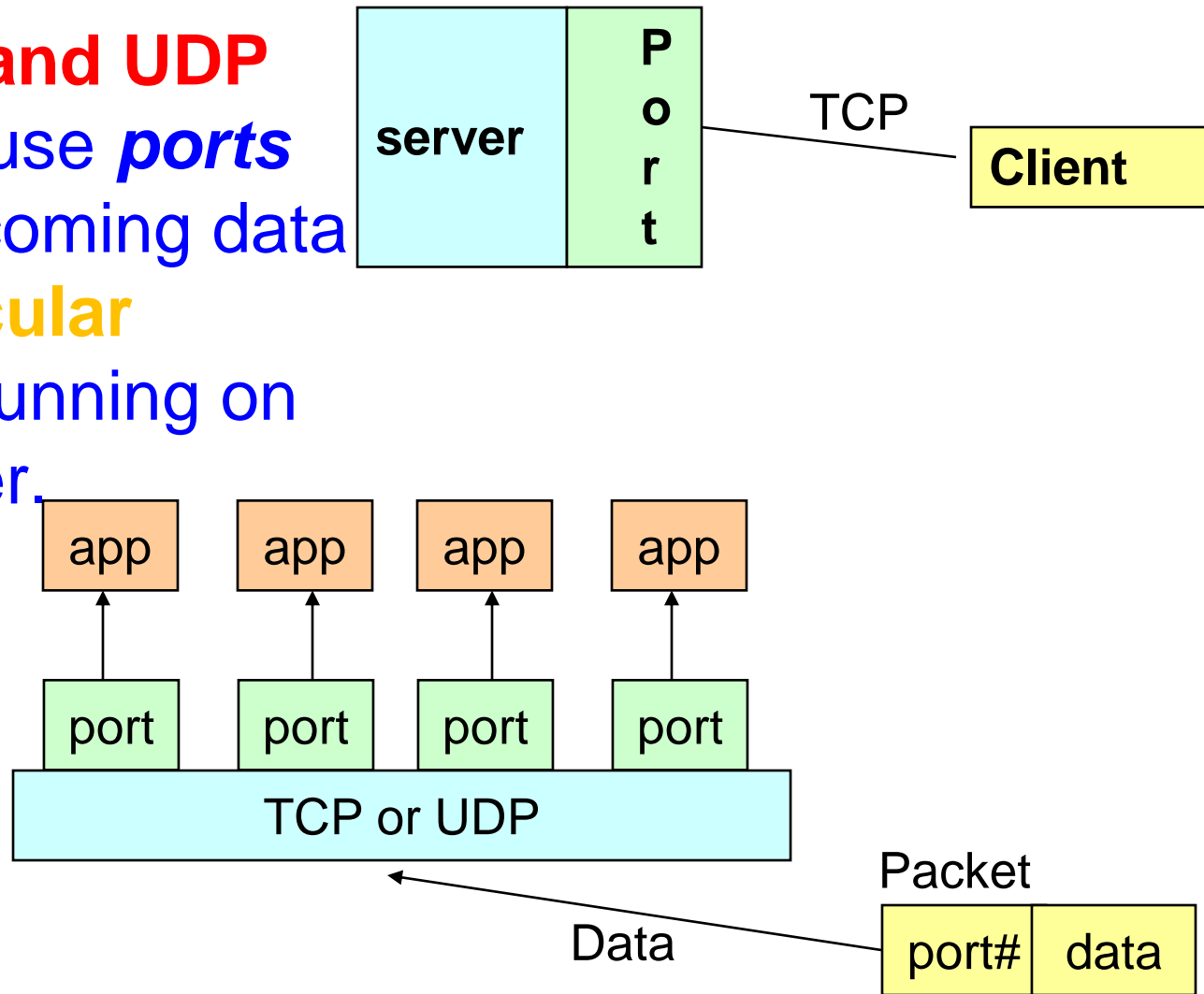| Application (http,ftp,telnet,…) |
| :---: |
| Transport (TCP, UDP,..) |
| Internet/Network (IP,..) |
| Physical/Link (device driver,..) |

TCP Vs UDP Communication

- Connection-Oriented Communication

- Connectionless Communication

Understanding Ports

■ The **TCP and UDP** protocols use *ports* to map incoming data to **a particular** *process* running on a computer.

Understanding Ports

- Port is represented by a positive (16-bit) integer value

- Some ports have been reserved to support common/well known services:
  - ftp   21/tcp
  - telnet 23/tcp
  - smtp 25/tcp
  - login 513/tcp

- User-level processes/services generally use port number value >= 1024

Hosts Identification and Service Ports

- Every computer on the Internet is identified by a unique, 4-byte IP address (e.g. 128.250.25.158) [User friendly?]

- IP addresses are mapped to names like www.google.com.

- Computers often need to communicate and provide more than one type of service or to talk to multiple hosts/computers at a time (Examples?)

- Hence, the need for ports! (How?)

- IP versus port number?

- Sockets or ports?

Sockets

- Sockets provide an interface for programming networks **at the transport layer**
- Network communication using Sockets is very much similar to performing file I/O
  - In fact, socket handle is treated like file handle.
  - The streams used in file I/O operation are also applicable to socket-based I/O
- Socket-based communication is programming language independent.
  - That means, a socket program written in Java language can also communicate to a program written in Java or non-Java socket program
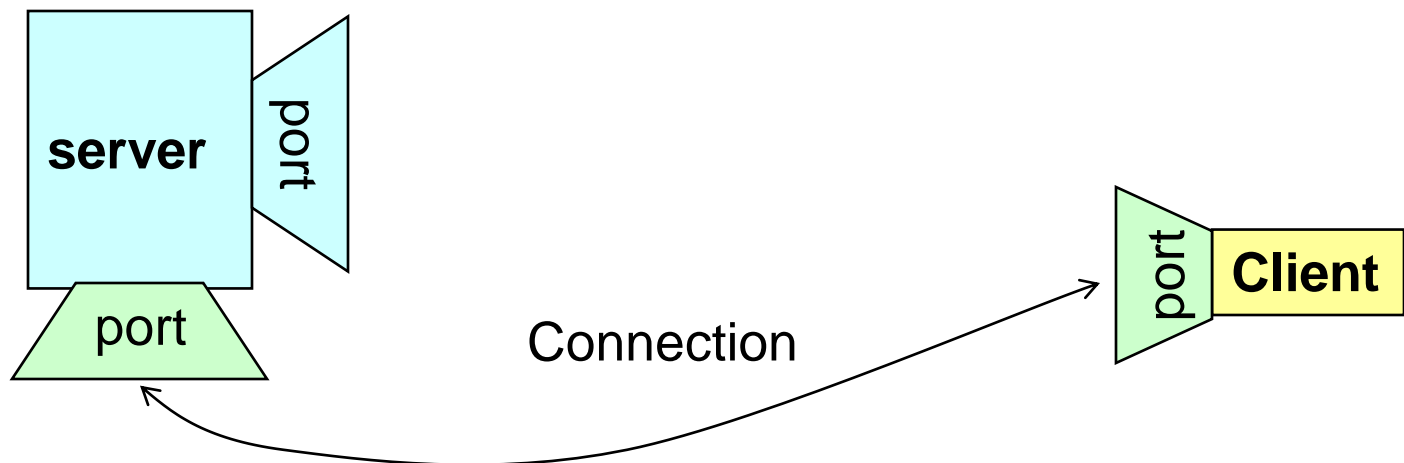
Socket Communication

■ A server (program) runs on a specific computer and has a socket that is bound to a specific port. The server waits and listens to the socket for a client to make a connection request.
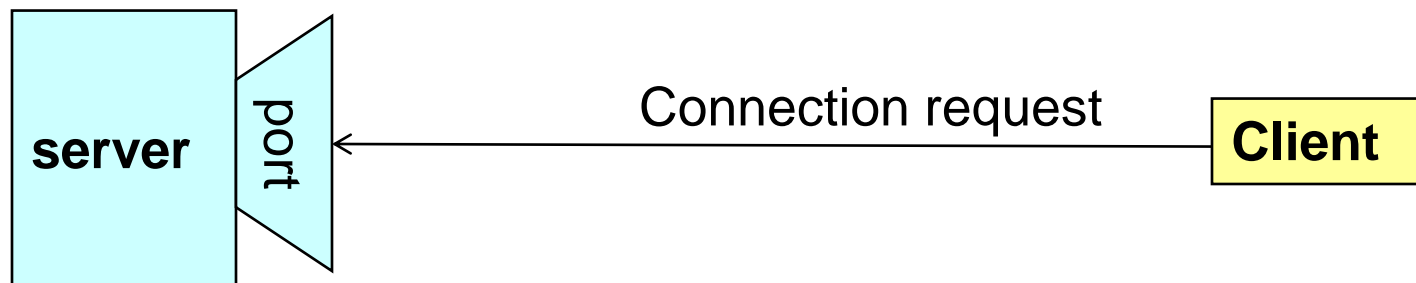
Socket Communication

- **If everything goes well, the server accepts the connection.** Upon acceptance, the server gets a new socket bounds to a different port. It needs a new socket (consequently a different port number) so that it can continue to listen to the original socket for connection requests while serving the connected client.

Sockets and Java Socket Classes

- A socket is an endpoint of a two-way communication link between two programs running on the network.

- A socket is bound to a port number so that the TCP layer can identify the application that data destined to be sent.

TCP/IP Socket Programming

- Java's .net package provides two classes:
    - Socket – for implementing a client
    - ServerSocket – for implementing a server

TCP/IP Socket Programming

Server

ServerSocket(1234)

Client

Output/write stream

Input/read stream

Socket("128.250.22.134", 1234)
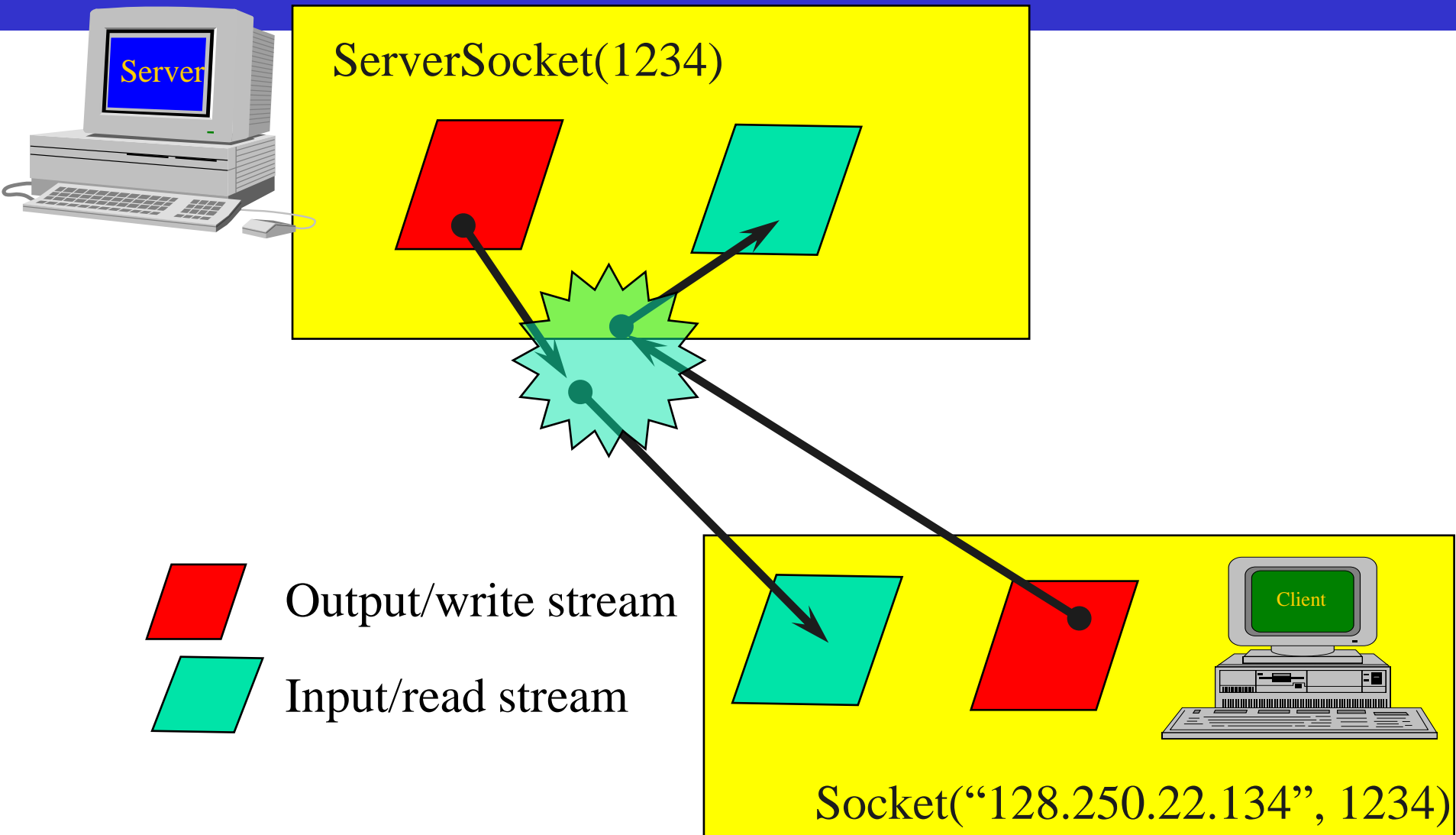
It can be host_name like "jarrett.cis.unimelb.edu.au"

TCP/IP: Implementing a Server

1. Open the Server Socket:

```
ServerSocket server;
  DataOutputStream os;
  DataInputStream is;
  server = new ServerSocket( PORT );
```

2. Wait for the Client Request:

```
Socket client = server.accept();
```

3. Create I/O streams for communicating to the client

```
is = new DataInputStream(
client.getInputStream() );
  os = new DataOutputStream(
client.getOutputStream() );
```

TCP/IP: Implementing a Server

4. Perform communication with client

   Receive from client:

```
String line = is.readLine();
```

Send to client:

```
os.writeBytes("Hello\n");
```

5. Close sockets: `client.close();`

<u>For multithreaded server:</u>

```
while(true) {
```

  i. wait for client requests (step 2 above)

  ii. create a thread with "client" socket as parameter (the thread creates streams (as in step(3) and does communication as stated in (4). Remove thread once service is provided.

```
}
```

TCP/IP: A simple server (simplified code)

```java
// SimpleServer.java: a simple server program
import java.net.*;
import java.io.*;
public class SimpleServer {
  public static void main(String args[])
  throws IOException {
    // Register service on port 1234
    ServerSocket s = new ServerSocket(1234);
    Socket s1=s.accept(); // Wait and accept a
  connection
```

TCP/IP: A simple server (simplified code)

```java
// Get a communication stream associated with
  the socket
    OutputStream s1out = s1.getOutputStream();
    DataOutputStream dos = new
DataOutputStream (s1out);
    // Send a string!
    dos.writeUTF("Hi there");
    // Close the connection, but not the
server socket
    dos.close();
    s1out.close();
    s1.close();
  }
```

TCP/IP: Implementing a Client

1. Create a Socket Object:

```
client = new Socket( server, port_id );
```

2. Create I/O streams for communicating with the server.

```
is = new
DataInputStream(client.getInputStream() );
  os = new DataOutputStream(
client.getOutputStream() );
```

3. Perform I/O or communication with the server:

- **Receive data from the server:**

```
String line = is.readLine();
```

- **Send data to the server:**

```
os.writeBytes("Hello\n");
```

4. Close the socket when done:

```
client.close();
```

37

TCP/IP: A simple client (simplified code)

```java
// SimpleClient.java: a simple client
   program
import java.net.*;
import java.io.*;
public class SimpleClient {
   public static void main(String args[])
   throws IOException {
      // Open your connection to a server, at
   port 1234
      Socket s1 = new
   Socket("jarrett.cis.unimelb.edu.au",1234)
   ;
```

TCP/IP: A simple client (simplified code)

```
// Get an input file handle from the socket
 and read the input
    InputStream s1In = s1.getInputStream();
    DataInputStream dis = new
DataInputStream(s1In);
    String st = new String (dis.readUTF());
    System.out.println(st);
    // When done, just close the connection
and exit
    dis.close();
    s1In.close();
    s1.close();
```

Run

A. Run Server on mundroo.cs.mu.oz.au
   - [raj@mundroo] java SimpleServer &
B. Run Client on any machine (including mundroo):
   - [raj@mundroo] java SimpleClient
     Hi there
C. If you run client when server is not up:
   - [raj@mundroo] sockets [1:147] java SimpleClient
   Exception in thread "main" java.net.ConnectException: **Connection refused**
       at java.net.PlainSocketImpl.socketConnect(Native Method)
       at java.net.PlainSocketImpl.doConnect(PlainSocketImpl.java:320)
       at java.net.PlainSocketImpl.connectToAddress(PlainSocketImpl.java:133)
       at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:120)
       at java.net.Socket.<init>(Socket.java:273)
       at java.net.Socket.<init>(Socket.java:100)

Socket Exceptions

```java
try {
    Socket client = new Socket(host, port);
    handleConnection(client);
}
catch(UnknownHostException uhe) {
    System.out.println("Unknown host: " + host);
    uhe.printStackTrace();
}
catch(IOException ioe) {
System.out.println("IOException: " + ioe);
    ioe.printStackTrace();
}
```

ServerSocket & Exceptions

- public **ServerSocket**(int port) throws IOException
    - Creates a server socket on a specified port
    - A port of 0 creates a socket on any free port. You can use **getLocalPort**() to identify the (assigned) port on which this socket is listening
    - The maximum queue length for incoming connection indications (a request to connect) is set to 50. If a connection indication arrives when the queue is full, the connection is refused
- Throws:
    - IOException - if an I/O error occurs when opening the socket
    - SecurityException - if a security manager exists and its checkListen method doesn't allow the operation

Server in Loop: Always up

```
// SimpleServerLoop.java: a simple server program that
   runs forever in a single thead
public class SimpleServerLoop {
  public static void main(String args[]) throws IOException {
    // Register service on port 1234
    ServerSocket s = new ServerSocket(1234);
    while(true)
    {
         Socket s1=s.accept(); // Wait and accept a connection
        // Get a communication stream associated with the socket
        OutputStream s1out = s1.getOutputStream();
        DataOutputStream dos = new DataOutputStream (s1out);
        // Send a string!
        dos.writeUTF("Hi there");
        // Close the connection, but not the server socket
        dos.close();
```

43

Java API for UDP Programming

- **Java API provides datagram communication by means of two classes**
  - DatagramPacket
    - | Msg | length | Host | serverPort |

  - DatagramSocket

Java API for UDP Programming

- Datagram packets are used to implement a connectionless packet delivery service supported by the UDP protocol.

- Each message is transferred from source machine to destination based on information contained within that packet.

- Hence, each packet needs to have destination address and each packet might be routed differently, and might arrive in any order (Side effect?)

45

UDP Client: Sends a Message and Gets reply

```java
import java.net.*;
import java.io.*;
public class UDPClient
{
    public static void main(String args[]){
// args give message contents and server hostname
// "Usage: java UDPClient <message> <Host name> <Port number>"
        DatagramSocket aSocket = null;
        try {
            aSocket = new DatagramSocket();
            byte [] m = args[0].getBytes();
            InetAddress aHost = InetAddress.getByName(args[1]);
            int serverPort = 6789;  // Or
    Integer.valueOf(args[2]).intValue() if use <Port number> args[2]
            DatagramPacket request = new DatagramPacket(m,
    args[0].length(), aHost, serverPort);
            aSocket.send(request);
```

46

UDP Client: Sends a Message and Gets reply

```java
    byte[] buffer = new byte[1000];
    DatagramPacket reply = new DatagramPacket(buffer,
    buffer.length);
    aSocket.receive(reply);
    System.out.println("Reply: " + new String(reply.getData()));
  }

  catch (SocketException e){System.out.println("Socket: " +
   e.getMessage());}
  catch (IOException e){System.out.println("IO: " + e.getMessage());}
      finally {
        if(aSocket != null) aSocket.close();
      }
    }
}
```

UDP Sever: repeatedly received a request and sends it back to the client

```java
import java.net.*;
import java.io.*;
public class UDPServer{
    public static void main(String args[]){
    DatagramSocket aSocket = null;
      try{
         aSocket = new DatagramSocket(6789); // fixed port
  number
         byte[] buffer = new byte[1000];
         while(true){
            DatagramPacket request = new
      DatagramPacket(buffer, buffer.length);
            aSocket.receive(request);
```
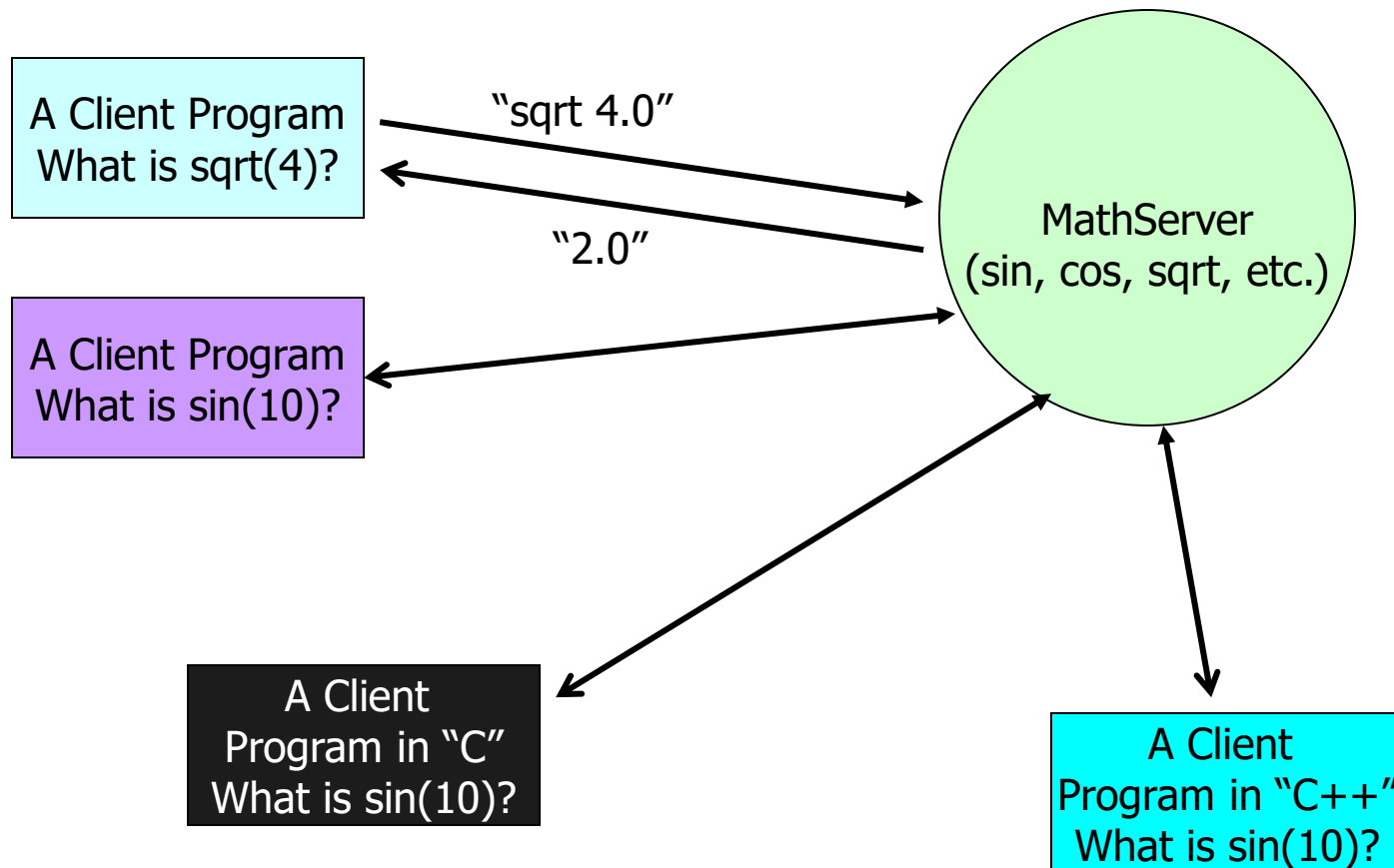
```
DatagramPacket reply = new
DatagramPacket(request.getData(),
            request.getLength(), request.getAddress(),
request.getPort());
        aSocket.send(reply);
      }
    }catch (SocketException e){System.out.println("Socket:
" + e.getMessage());}
      catch (IOException e) {System.out.println("IO: " +
e.getMessage());}
   finally {if(aSocket != null) aSocket.close();}
}
}
```

Any comments?

Example 1: MathServer – Demonstrates the use of Sockets

A Client Program
What is sqrt(4)?

"sqrt 4.0"

"2.0"

MathServer
(sin, cos, sqrt, etc.)

A Client Program
What is sin(10)?

A Client
Program in "C"
What is sin(10)?

A Client
Program in "C++"
What is sin(10)?

Example 1: MathServer

- **What could be the basic interface for usage?**

Program 13.5

```java
// MathService.java: A basic math interface.
public interface MathService {
    public double add(double firstValue, double secondValue);
    public double sub(double firstValue, double secondValue);
    public double div(double firstValue, double secondValue);
    public double mul(double firstValue, double secondValue);
}
```

Example 1: MathServer

- What could be the basic implementation for that interface?

Program 13.6

```java
// PlainMathService.java: An implementation of the MathService interface.
public class PlainMathService implements MathService {

    public double add(double firstValue, double secondValue) {
        return firstValue+secondValue;
    }
    public double sub(double firstValue, double secondValue) {
        return firstValue-secondValue;
    }
    public double mul(double firstValue, double secondValue) {
      return firstValue * secondValue;
    }
    public double div(double firstValue, double secondValue) {
        if (secondValue != 0)
         return firstValue / secondValue;
        return Double.MAX_VALUE;
    }

}
```

Example 1: MathServer

- **What could be the implementation MathServer?**

```java
// MathServer.java : An implementation of the MathServer.
import java.io.*;
import java.net.*;

public class MathServer{
    protected MathService mathService;
    protected Socket socket;
```

Example 1: MathServer

- **What could be the implementation MathServer?**

```
public void setMathService(MathService mathService) {
    this.mathService = mathService;
}
public void setSocket(Socket socket) {
    this.socket = socket;
}
```

Example 1: MathServer

- What could be the implementation MathServer?

```java
public void execute() {
    try {
        BufferedReader reader = new BufferedReader(
            new InputStreamReader(socket.getInputStream()));
        // read the message from client and parse the execution
        String line = reader.readLine();
        double result = parseExecution(line);
        // write the result back to the client
        BufferedWriter writer = new BufferedWriter(
            new OutputStreamWriter(socket.getOutputStream()));
        writer.write(""+result);
        writer.newLine();
        writer.flush();
        // close the stream
        reader.close();
        writer.close();
    }
        catch (Exception e) {
        e.printStackTrace();
    }
}
```

55

Example 1: MathServer

- ## What could be the implementation MathServer?

```
// the predefined protocol for the math operation is
// operator:first value:second value
protected double parseExecution(String line)
                  throws IllegalArgumentException {
    double result = Double.MAX_VALUE;
    String [] elements = line.split(":");
    if (elements.length != 3)
      throw new IllegalArgumentException("parsing error!");
    double firstValue = 0;
    double secondValue = 0;
    try {
      firstValue = Double.parseDouble(elements[1]);
      secondValue = Double.parseDouble(elements[2]);
    }
        catch(Exception e) {
      throw new IllegalArgumentException("Invalid arguments!");
    }
    switch (elements[0].charAt(0)) {
      case '+':
        result = mathService.add(firstValue, secondValue);
      break;
      case '-':
```

Example 1: MathServer

- **What could be the contents of the main methods?**

```java
public static void main(String [] args)throws Exception{
    int port = 10000;
    if (args.length == 1) {
      try {
        port = Integer.parseInt(args[0]);
      }
     catch(Exception e){
     }
    }
    System.out.println("Math Server is running...");
    // create a server socket and wait for client's connection
    ServerSocket serverSocket = new ServerSocket(port);
    Socket socket = serverSocket.accept();
    // run a math server that talks to the client
    MathServer mathServer = new MathServer();
    mathServer.setMathService(new PlainMathService());
    mathServer.setSocket(socket);
    mathServer.execute();
    System.out.println("Math Server is closed...");
}
```

Example 1: main() for MathServer?

```java
public static void main(String [] args)throws Exception{
    int port = 10000;
    if (args.length == 1) {
      try {
        port = Integer.parseInt(args[0]);
      }
     catch(Exception e){
     }
    }
    System.out.println("Math Server is running...");
    // create a server socket and wait for client's connection
    ServerSocket serverSocket = new ServerSocket(port);
    Socket socket = serverSocket.accept();
    // run a math server that talks to the client
    MathServer mathServer = new MathServer();
    mathServer.setMathService(new PlainMathService());
    mathServer.setSocket(socket);
    mathServer.execute();
    System.out.println("Math Server is closed...");
}
```

Example 1: Client for MathServer?

```java
// MathClient.java: A test client program to access MathServer.
import java.io.*;
import java.net.Socket;
public class MathClient {
    public static void main(String [] args){
        String hostname = "localhost";
        int port = 10000;
        if (args.length != 2) {
            System.out.println("Use the default setting...");
        }
            else {
```

Example 1: Client for MathServer?

```java
      hostname = args[0];
      port = Integer.parseInt(args[1]);
    }
    try {
      // create a socket
      Socket socket = new Socket(hostname, port);
      // perform a simple math operation "12+21"
      BufferedWriter writer = new BufferedWriter(
              new OutputStreamWriter(socket.getOutputStream()));
      writer.write("+:12:21");
      writer.newLine();
      writer.flush();
      // get the result from the server
      BufferedReader reader = new BufferedReader(
              new InputStreamReader(socket.getInputStream()));
      System.out.println(reader.readLine());
      reader.close();
      writer.close();
    }

        catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Required Reading

- ■ Chapter 13: Socket Programming
    - ■ R. Buyya, S. Selvi, X. Chu, **"Object Oriented Programming with Java: Essentials and Applications",** McGraw Hill, New Delhi, India, 2009.
    - ■ Sample chapters at book website: http://www.buyya.com/java/