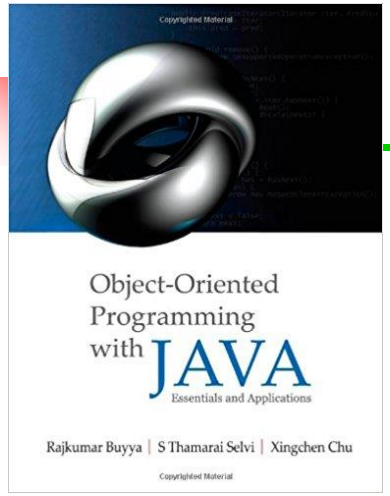
Inter-Process Communication (IPC): Network Programming using TCP Java Sockets



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
- 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, work, etc. It appears like the modern life activities are getting completely drive by the Internet

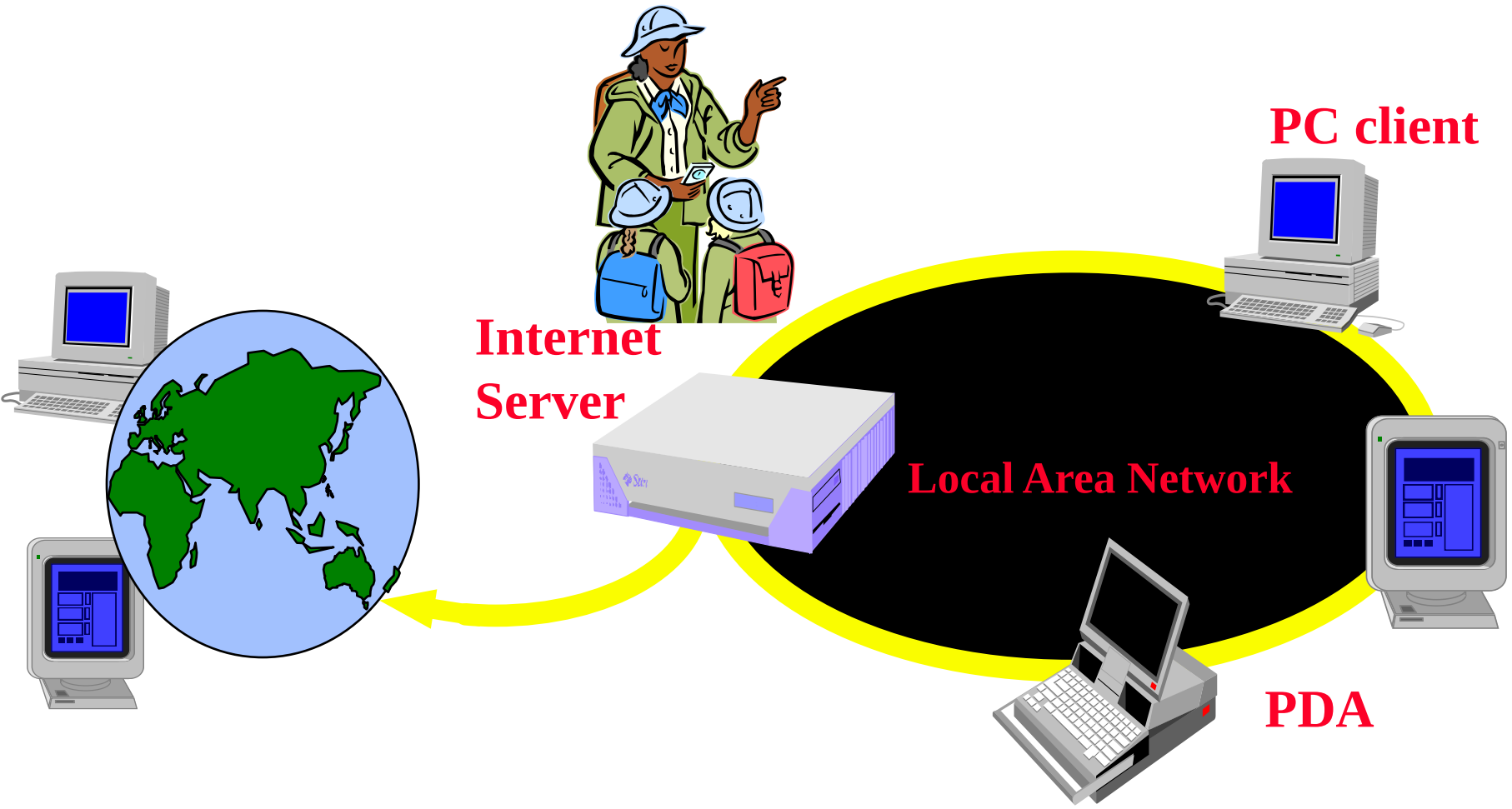
facebook

skype
For Business

WeChat

zoom

Internet Applications Serving Local and Remote Users

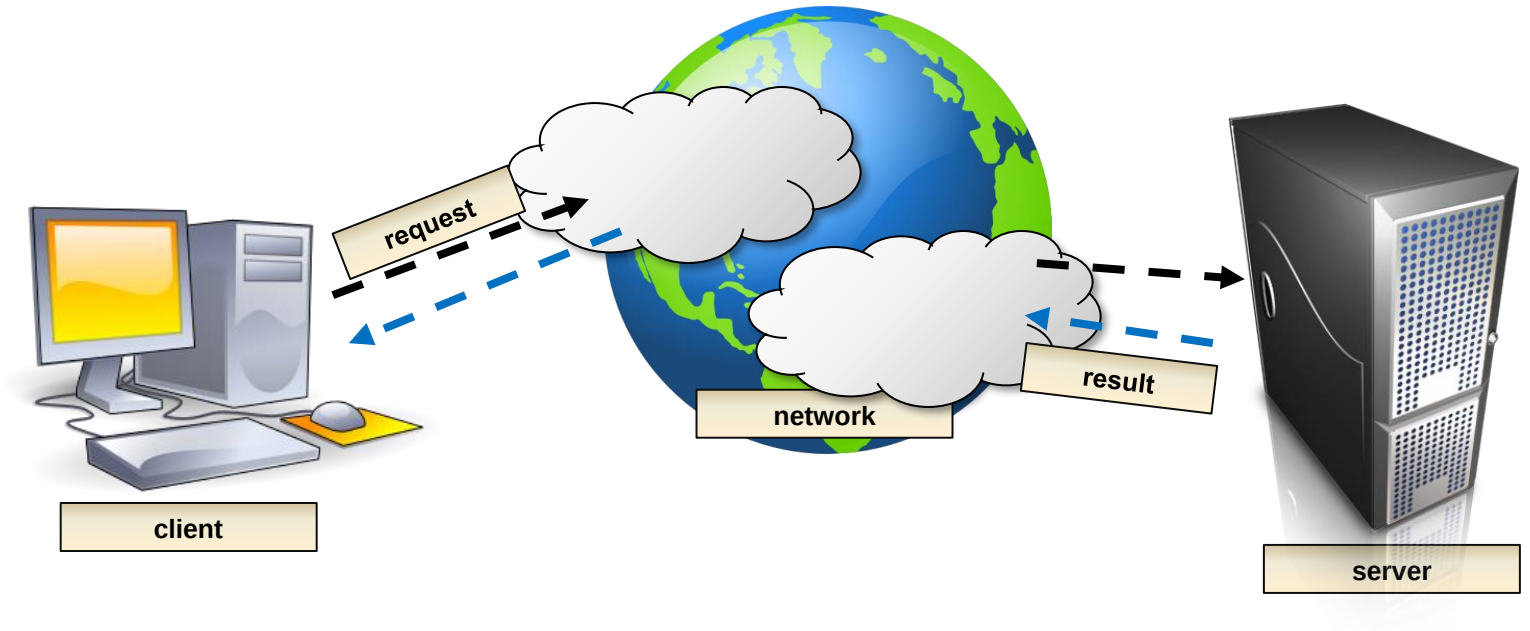


Increasing Demand for Internet Applications

- 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

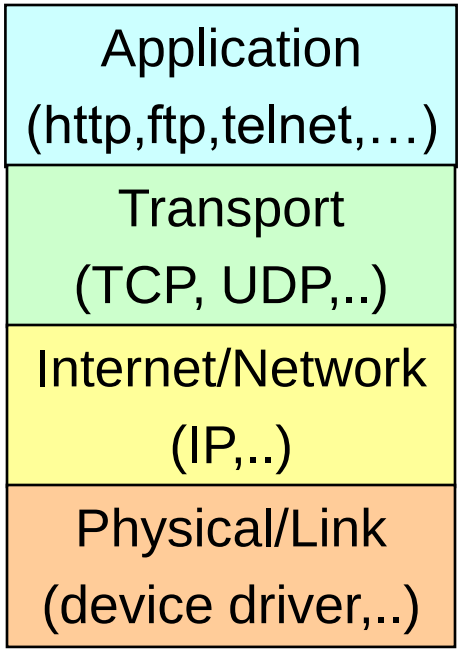


- Processes follow protocols that define a set of rules that must be observed by participants:
  - How the data exchange is encoded?
  - How events (sending, receiving) are synchronized (ordered) so that participants can send and receive data in a coordinated manner?
- In face-to-face communication,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

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
- 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,…**

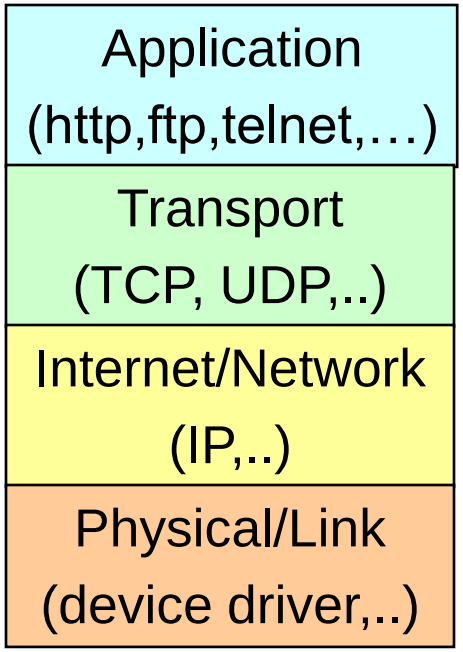
- TCP/IP Stack



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

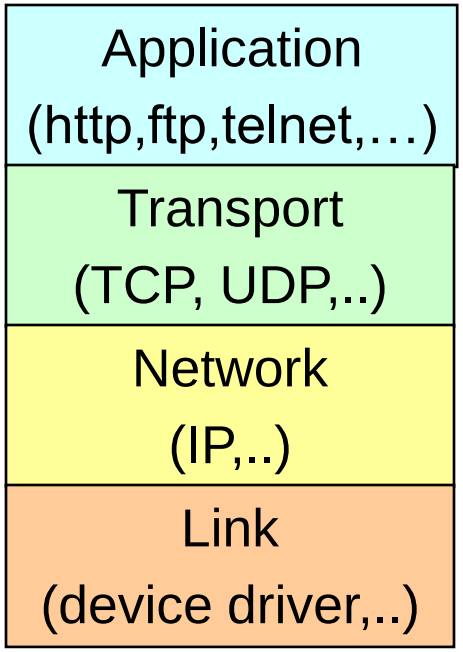
- TCP/IP Stack



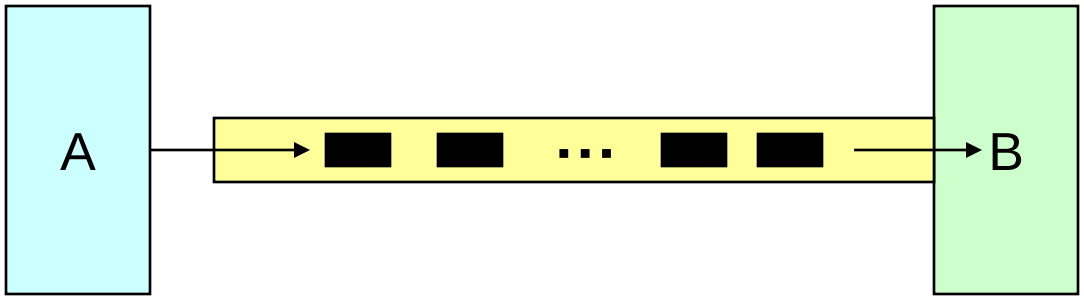
Networking Basics

- UDP (User Datagram Protocol) is a connectionless communication protocol that sends independent packets of data, called *datagrams*, from one computer to another with no guarantees about arrival or order of arrival
- Similar to sending multiple emails/letters to friends, each containing part of a message.
- Example applications:
  - Clock server
  - Ping
  - Live streaming (event/sports broadcasting)

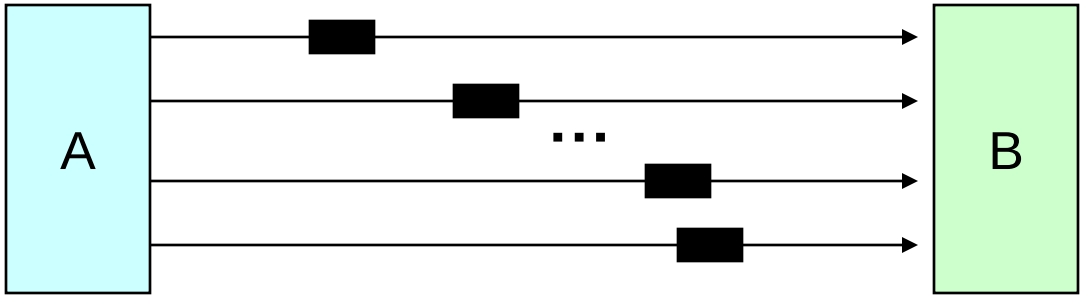
- TCP/IP Stack



TCP Vs UDP Communication



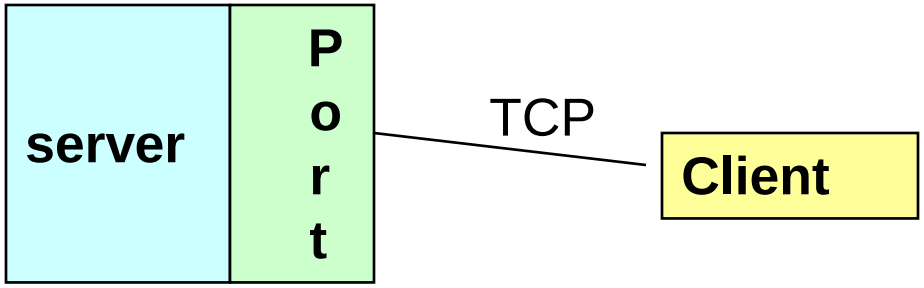
- Connection-Oriented Communication

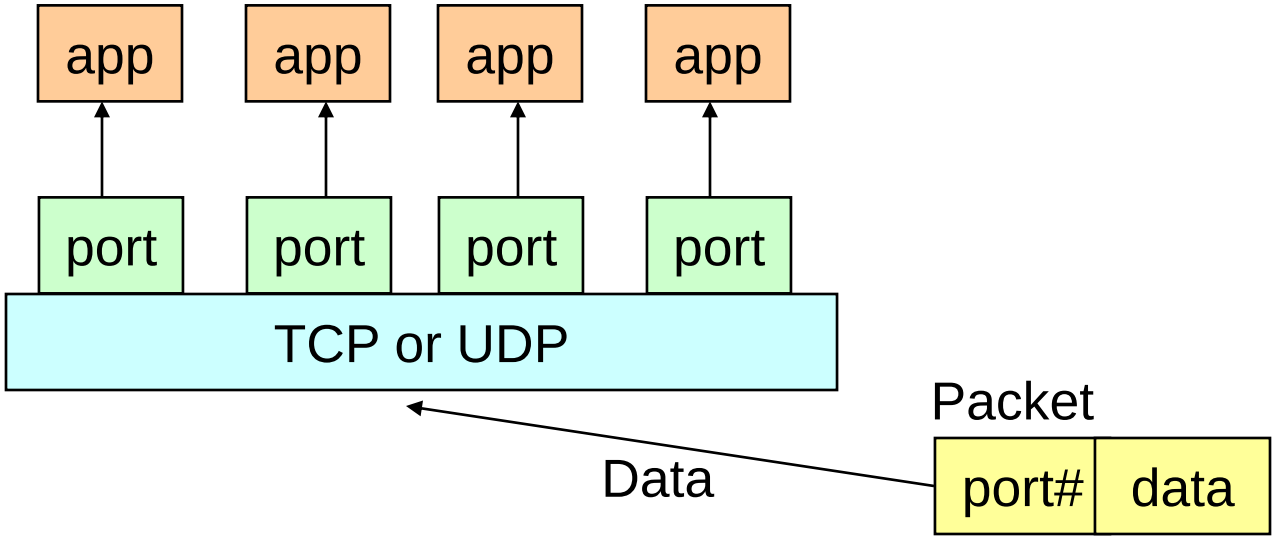


- Connectionless Communication

Understanding Ports

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





Understanding Ports

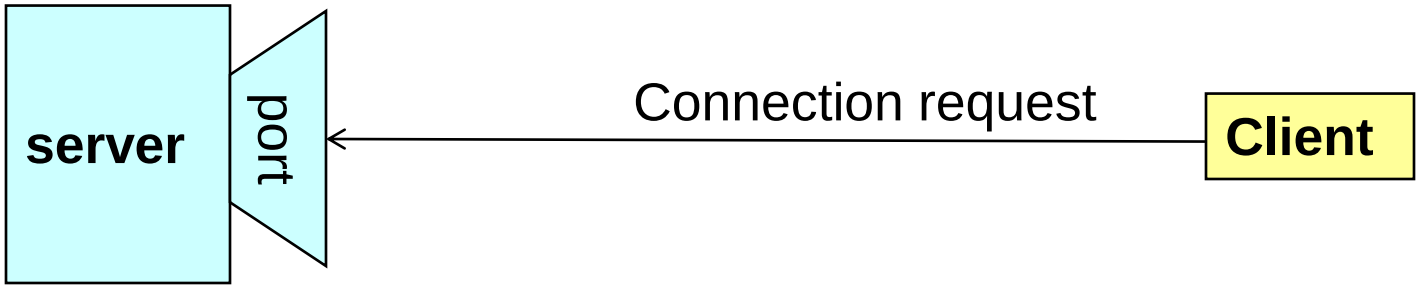
- Port is represented by a positive (16-bit) integer value
- Some ports have been reserved to support common/well known services:
  - ftp 21/tcp
  - telnet 23/tcp
  - smtp 25/tcp
  - http 80/tcp
  - login 513/tcp
  - https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers
- User-level processes/services generally use port number value >= 1024

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

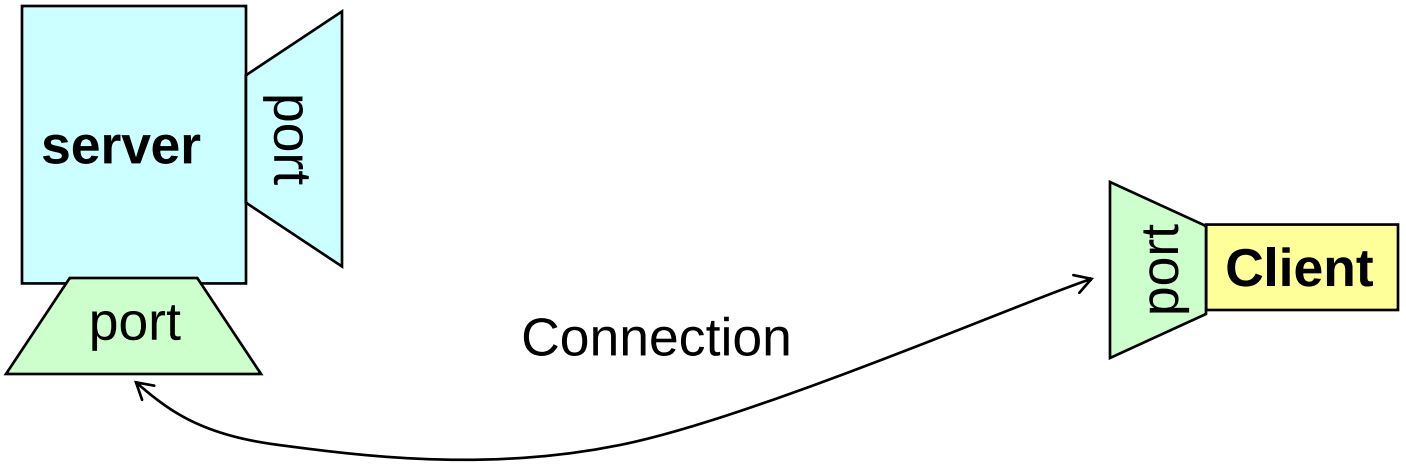
Socket Communication

- 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

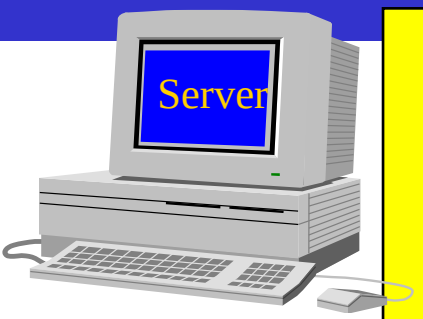
- If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bounds to a different port. It needs a new socket (consequently a different port number) so that it can continue to listen to the original socket for connection requests while serving the connected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



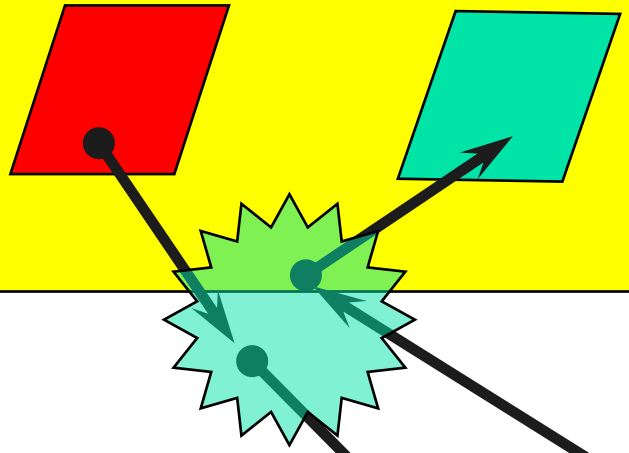
Sockets and Java Socket Classes

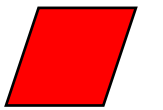
- A socket is an endpoint of a two-way communication link between two programs running on the network.
- A socket is bound to a port number so that the TCP layer can identify the application that data destined to be sent.
- Java's .net package provides two classes:
  - Socket – for implementing a client
  - ServerSocket – for implementing a server

Java Sockets

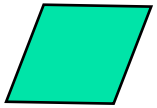


ServerSocket(1234)

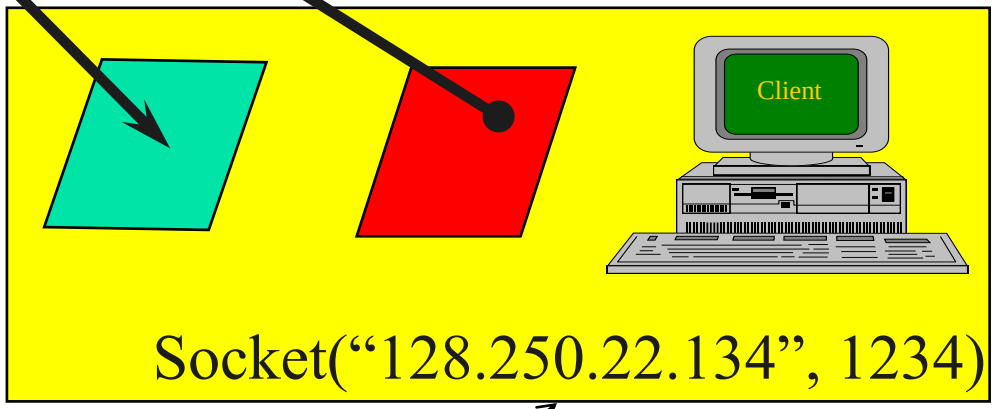




Output/write stream



Input/read stream



It can be host_name like "jarrett.cis.unimelb.edu.au"

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() );
```

4. Perform communication with client

```
Receive from client: String line = is.readLine();
Send to client: os.writeBytes("Hello\n");
```

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

For multithreaded server:

```
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.

```
}
```

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();
```

A simple server (simplified code)

```
// 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
    // 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();
  }
}
```

A simple client (simplified code)

```
// 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);
    // 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

- Run Server on mundroo.cs.mu.oz.au
  - [raj@mundroo] java SimpleServer &
- Run Client on any machine (including mundroo):
  - [raj@mundroo] java SimpleClient
    Hi there
- 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)
        at SimpleClient.main(SimpleClient.java:6)
```

Socket Exceptions

```
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
import java.net.*;
import java.io.*;
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();
            s1out.close();
            s1.close();
    }
  }
}
```

Java API for UDP Programming

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

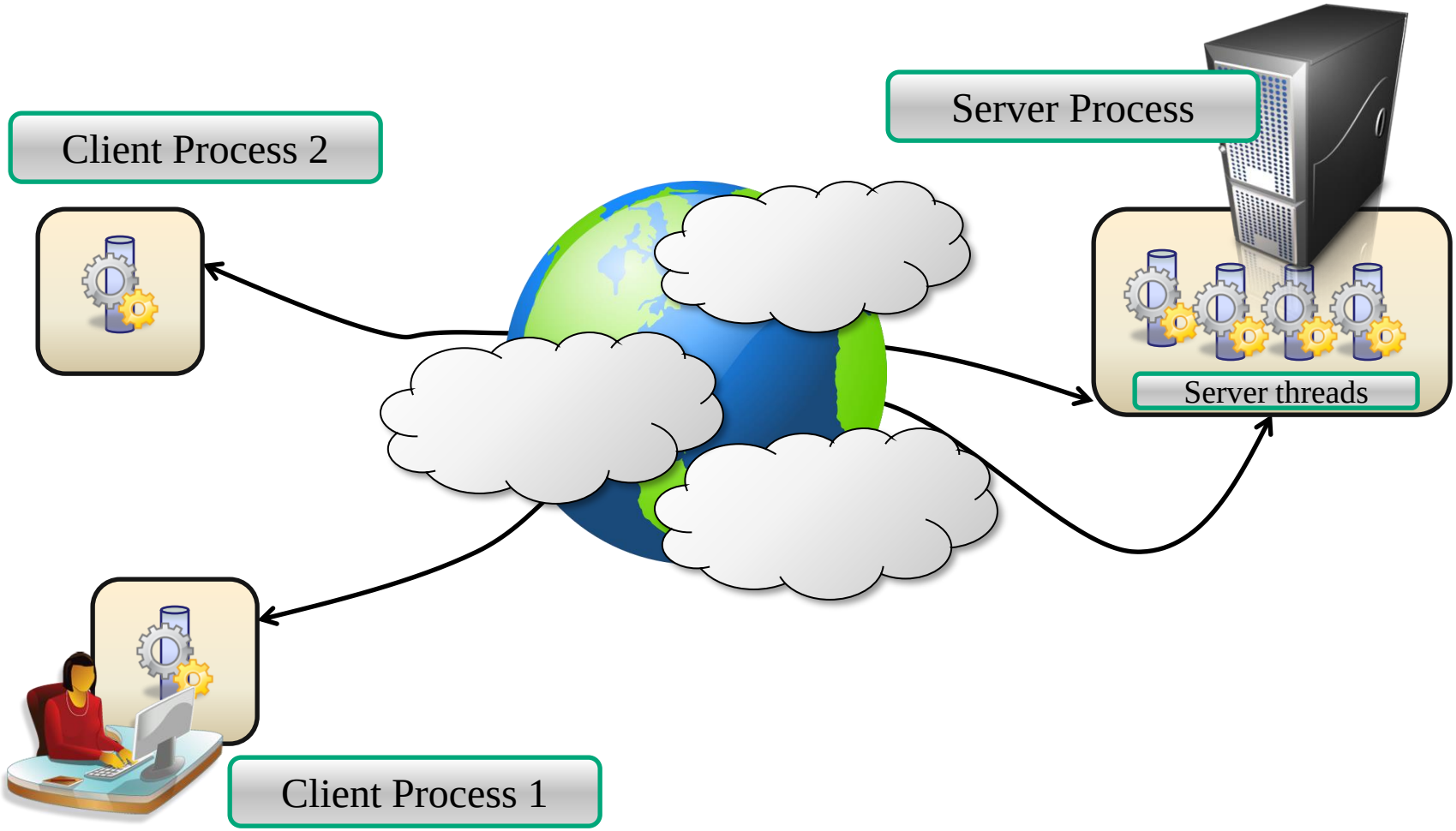
UDP Client: Sends a Message and Gets reply

```
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);
                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

```
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();}
    }
}
```

Multithreaded Server: For Serving Multiple Clients Concurrently



Summary

- Programming client/server applications in Java is fun and challenging
- Programming socket programming in Java is much easier than doing it in other languages such as C
- TCP for Connection-oriented communication, more reliable, flow control
- UDP for connection-less communication
- Keywords:
  - Clients, servers, TCP/IP, port number, sockets, Java sockets

References

- 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/