

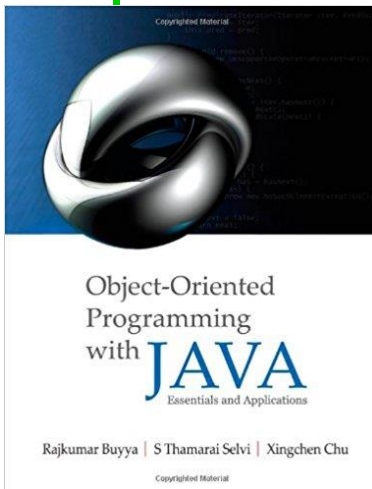


中国科学院深圳先进技术研究院  
SHENZHEN INSTITUTE OF ADVANCED TECHNOLOGY  
CHINESE ACADEMY OF SCIENCES

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

**Dr. Minxian Xu**

**Associate Professor**  
**Research Center for Cloud Computing**  
**Shenzhen Institute of Advanced Technology, CAS**  
**<http://www.minxianxu.info/dcp>**



云中谁寄锦书来，雁字回时，月满西楼。

——（宋）李清照

# Review

---





1. Provide a definition of a Distributed System

# Review

---

## 1. Provide a definition of a Distributed System

-  A system in which hardware or software components located at networked computers communicate and coordinate their actions only by passing message [Coulouris]
  
-  A collection of independent computers that appears to its users as a single coherent system [Tanenbaum]

# Review

---



2. Briefly explain the difference between a computer network and a distributed system.

# Review

---



2. Briefly explain the difference between a computer network and a distributed system.

**A Computer Network:** Is a collection of spatially separated, interconnected computers that exchange messages based on specific protocols. Computers are addressed by IP addresses.

**A Distributed System:** Multiple computers on the network working together as a system. The spatial separation of computers and communication aspects are hidden from users.

# Review

---



3. List three reasons for using a distributed system.

# Review

---

3. List three reasons for using a distributed system.

- 📖 Economy (cost effective)
- 📖 Reliability (fault tolerance)
- 📖 Availability (high uptime)
- 📖 Scalability (extendible)
- 📖 Functional Separation (Modularity)

The main motivation to build and use distributed systems is Resource Sharing

- 📖 Hardware Resources (Disks, printers, scanners etc.)
- 📖 Software Resources (Files, databases etc)
- 📖 Other (Processing power, memory, bandwidth)

# Review

---



4. Briefly explain four consequences when using distributed systems, i.e. issues that arise that are not present otherwise.



# Review

---



4. Briefly explain four consequences when using distributed systems, i.e. issues that arise that are not present otherwise.

- 📁 Concurrency
- 📁 Heterogeneity
- 📁 No Global Clock
- 📁 Independent Failures

# 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



# Turing Award 2016



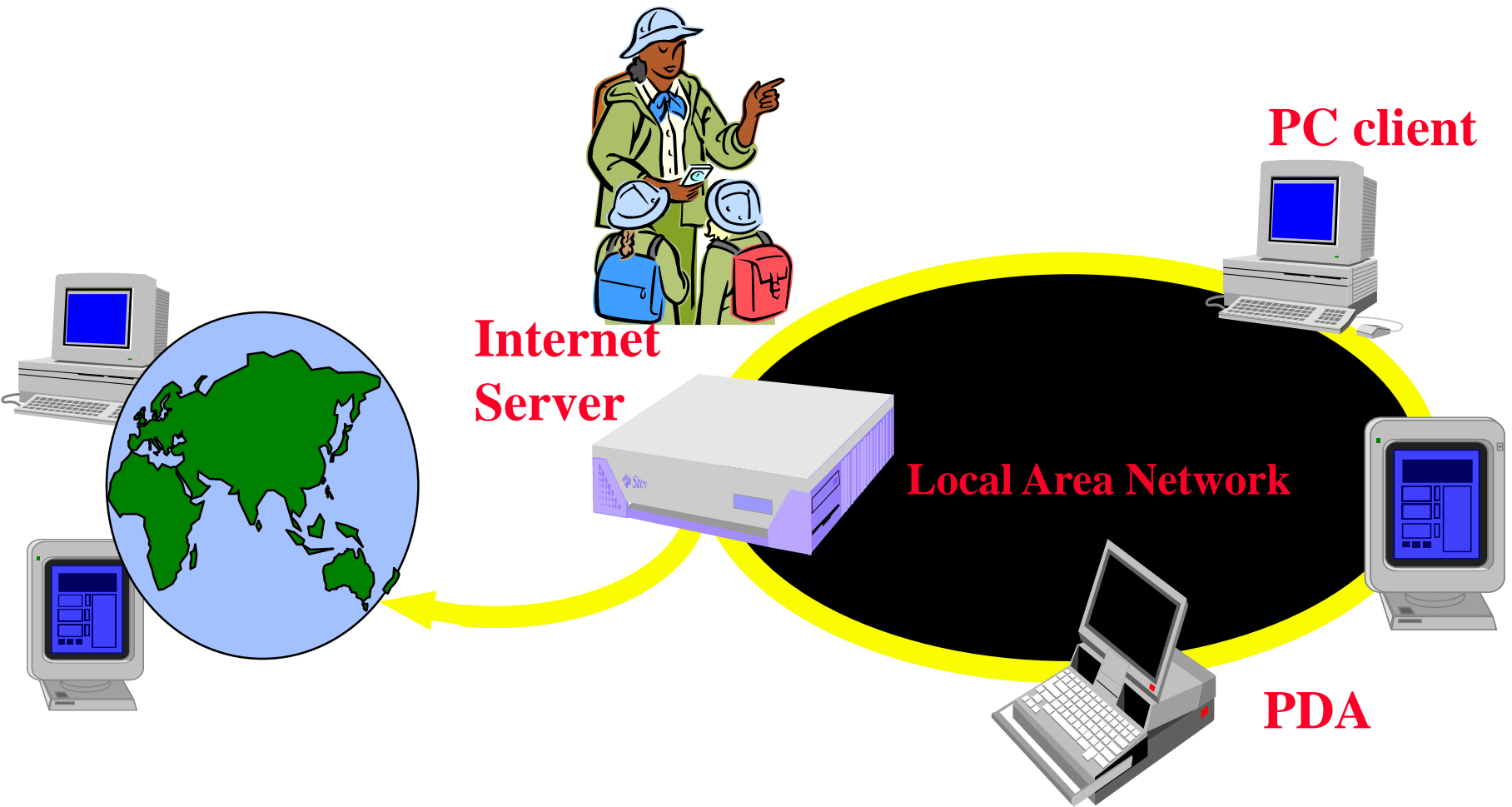
- Professor at MIT
- Invented:
  - URI that would serve to allow any object
  - HTTP that allows for the exchange, retrieval, or transfer of an object over the Internet
  - Web browser that that retrieves and renders resources on the World Wide Web
  - HTML that allows web browsers to translate documents or other resources



**Tim Berners-Lee**

Citation: *“For inventing the World Wide Web, the first web browser, and the fundamental protocols and algorithms allowing the Web to scale.”*

# Internet Applications Serving Local and Remote Users



# Increasing Demand for Internet Applications

---

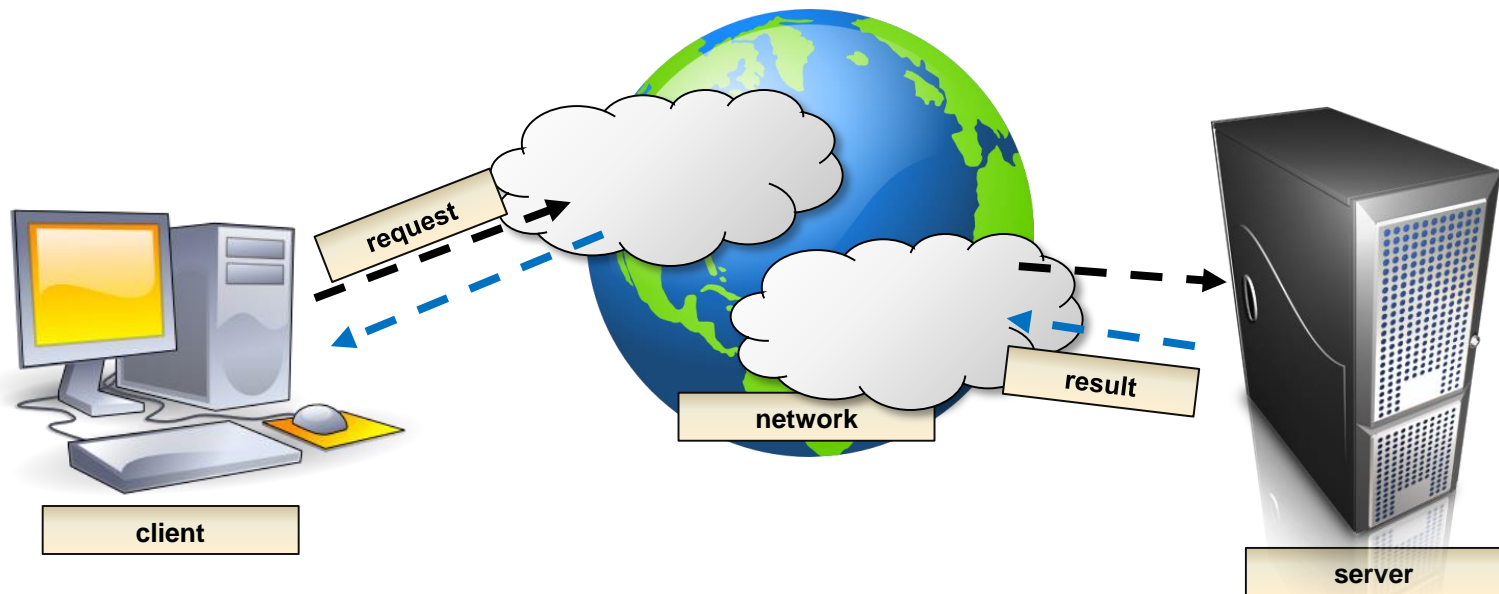


- To take advantage of opportunities presented by the Internet, businesses are continuously seeking new and innovative ways and means for offering their services via the Internet.
- This created a huge demand for software designers with skills to create new Internet-enabled applications or migrate existing/legacy 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

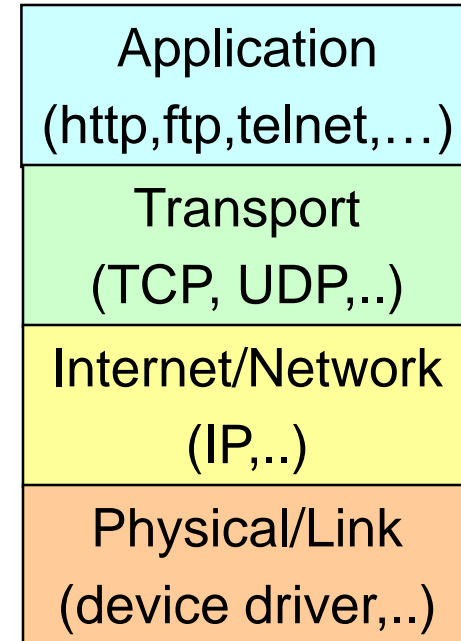
## ■ Transport Layer

- 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, **WeChat**,...

## ■ TCP/IP Stack

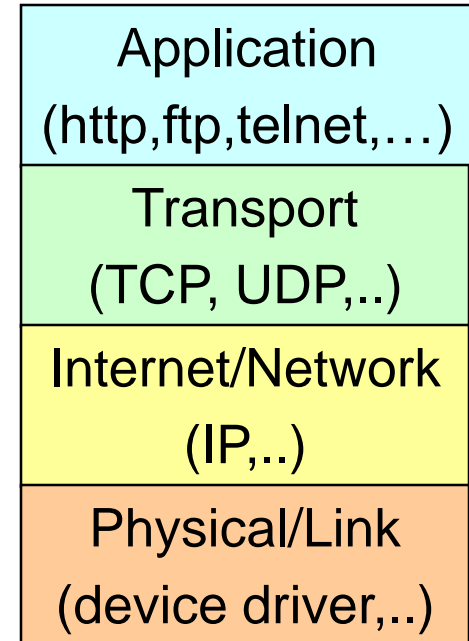




# Networking Basics

- 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
  - **WeChat** uses **TCP** for call signalling, and both **UDP** and **TCP** for transporting media traffic.

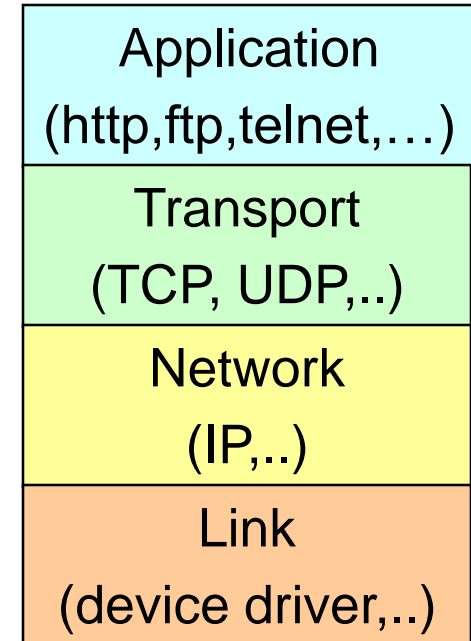
- TCP/IP Stack



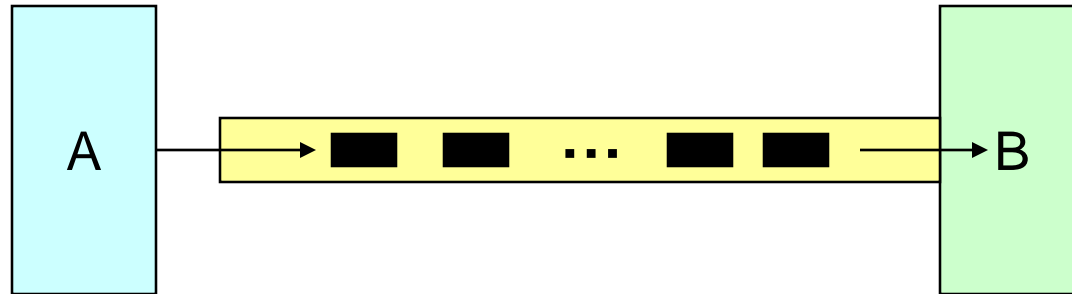
# Networking Basics

- 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)

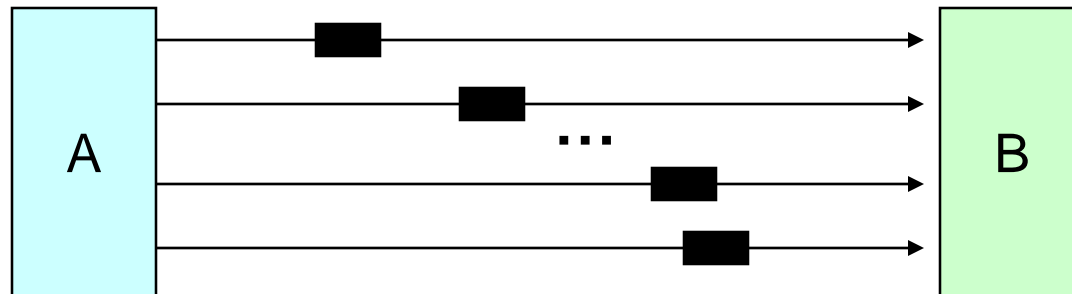
## ■ TCP/IP Stack



# TCP Vs UDP Communication



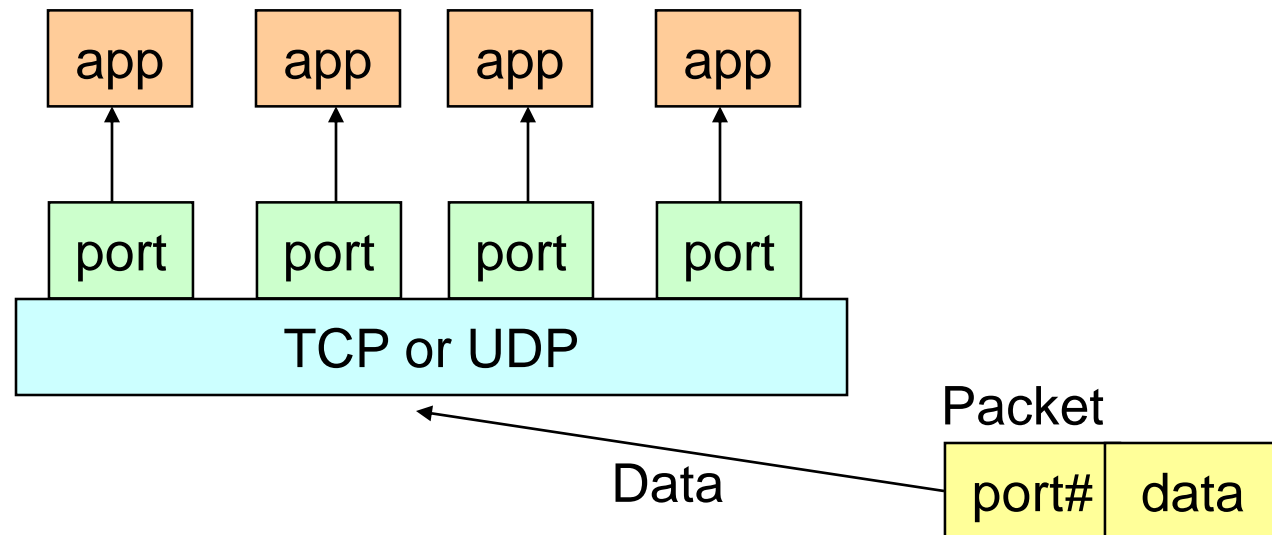
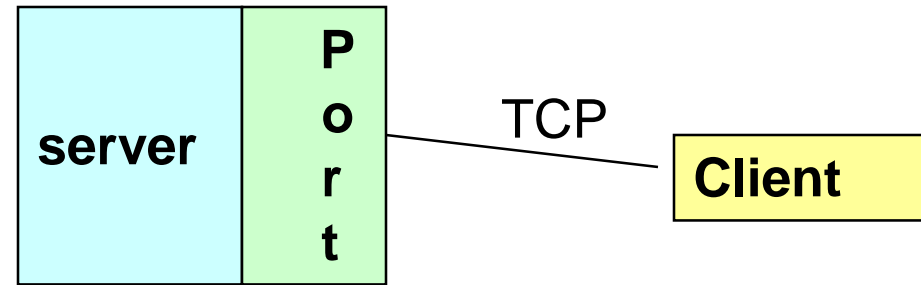
- Connection-Oriented Communication



- 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 (0~65535)
- 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](https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers)
- User-level processes/services generally use port number value  $\geq 1024$

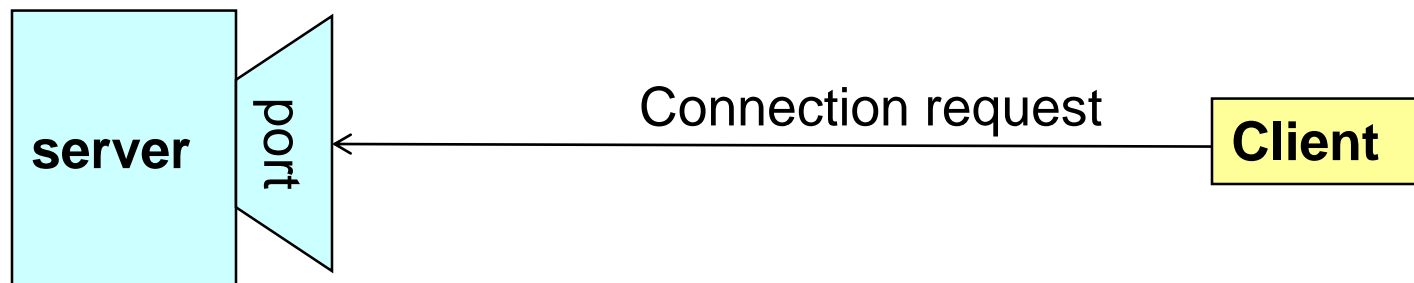
# Sockets



- 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

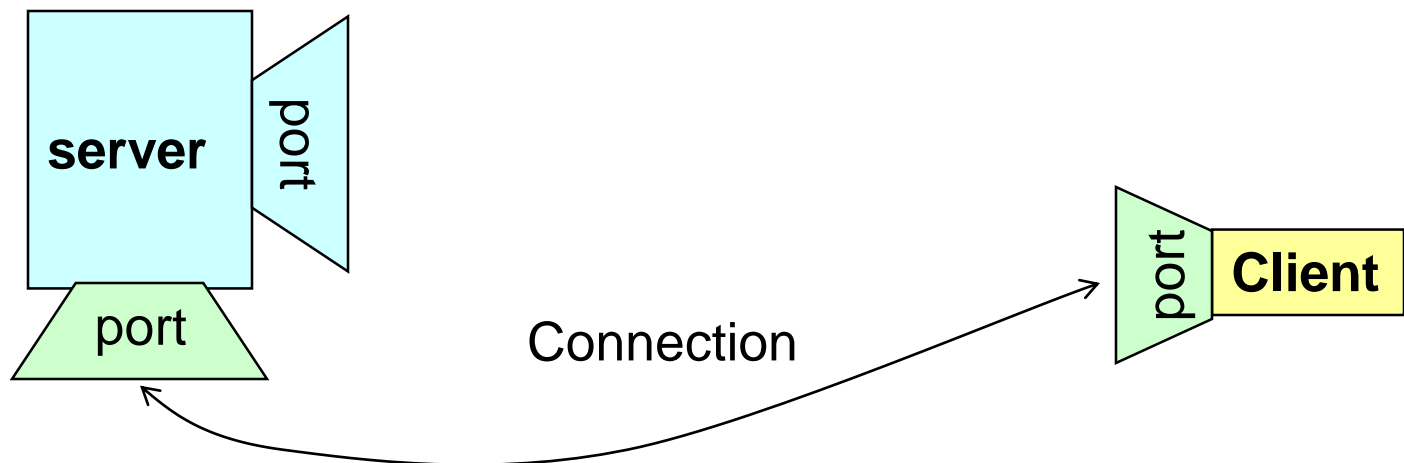
# Socket Communication

- 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.





# Multi-Client vs Server

- Be like John Snow facing troops
- OR Captain Jack Sparrow's running from savages



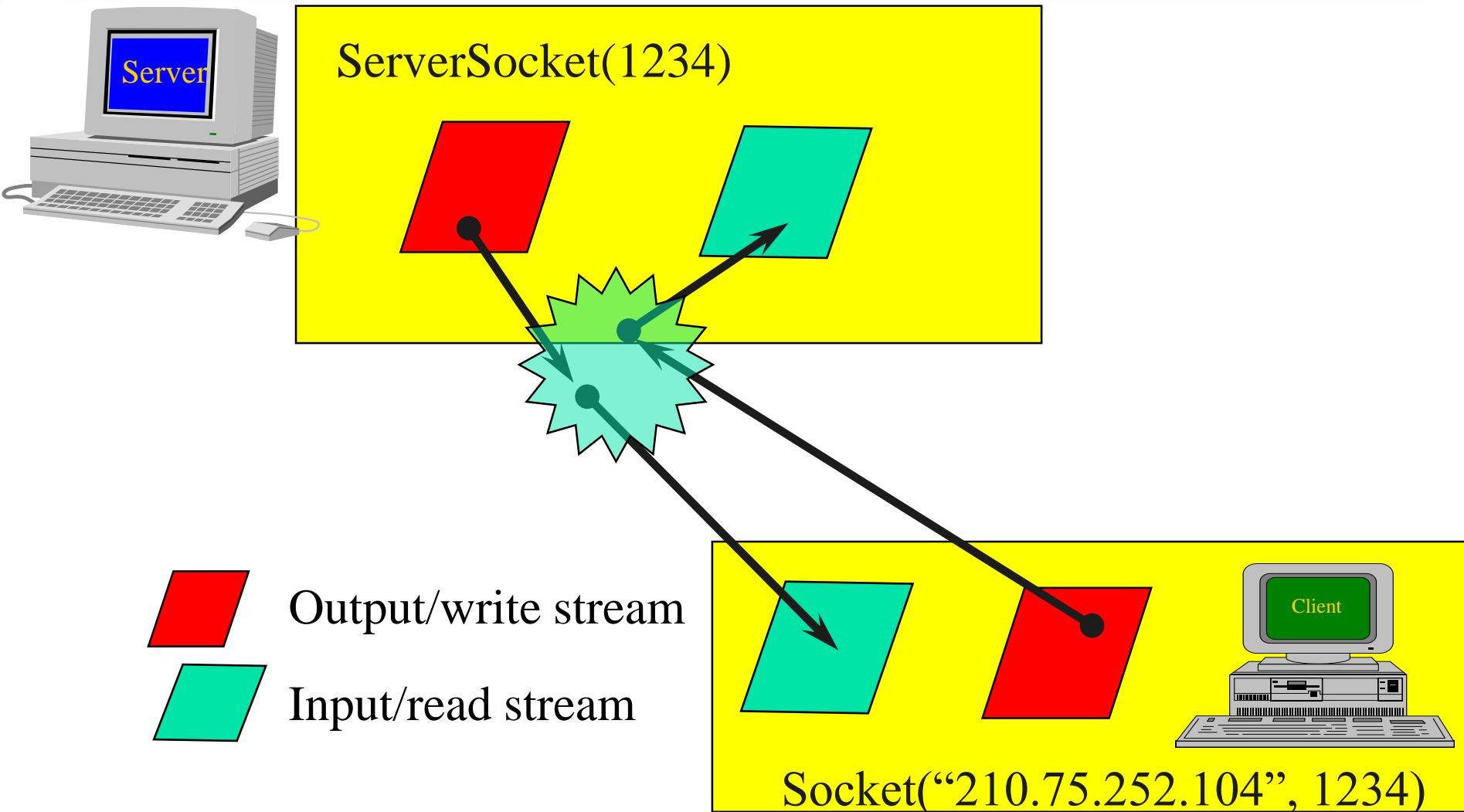
Game of Thrones



Pirates of the Caribbean

- 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



It can be host\_name like "https://www.siat.ac.cn/"

# 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 slout = s1.getOutputStream();
        DataOutputStream dos = new DataOutputStream (slout);
        // Send a string!
        dos.writeUTF("Hi there");
        // Close the connection, but not the server socket
        dos.close();
        slout.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("www.siat.ac.cn",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 a host at SIAT
  - [mx@siat] java SimpleServer &
- Run Client on any machine (including SIAT):
  - [mx@siat] java SimpleClient  
Hi there
- If you run client when server is not up:
  - [mx@siat] 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 thread

```
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

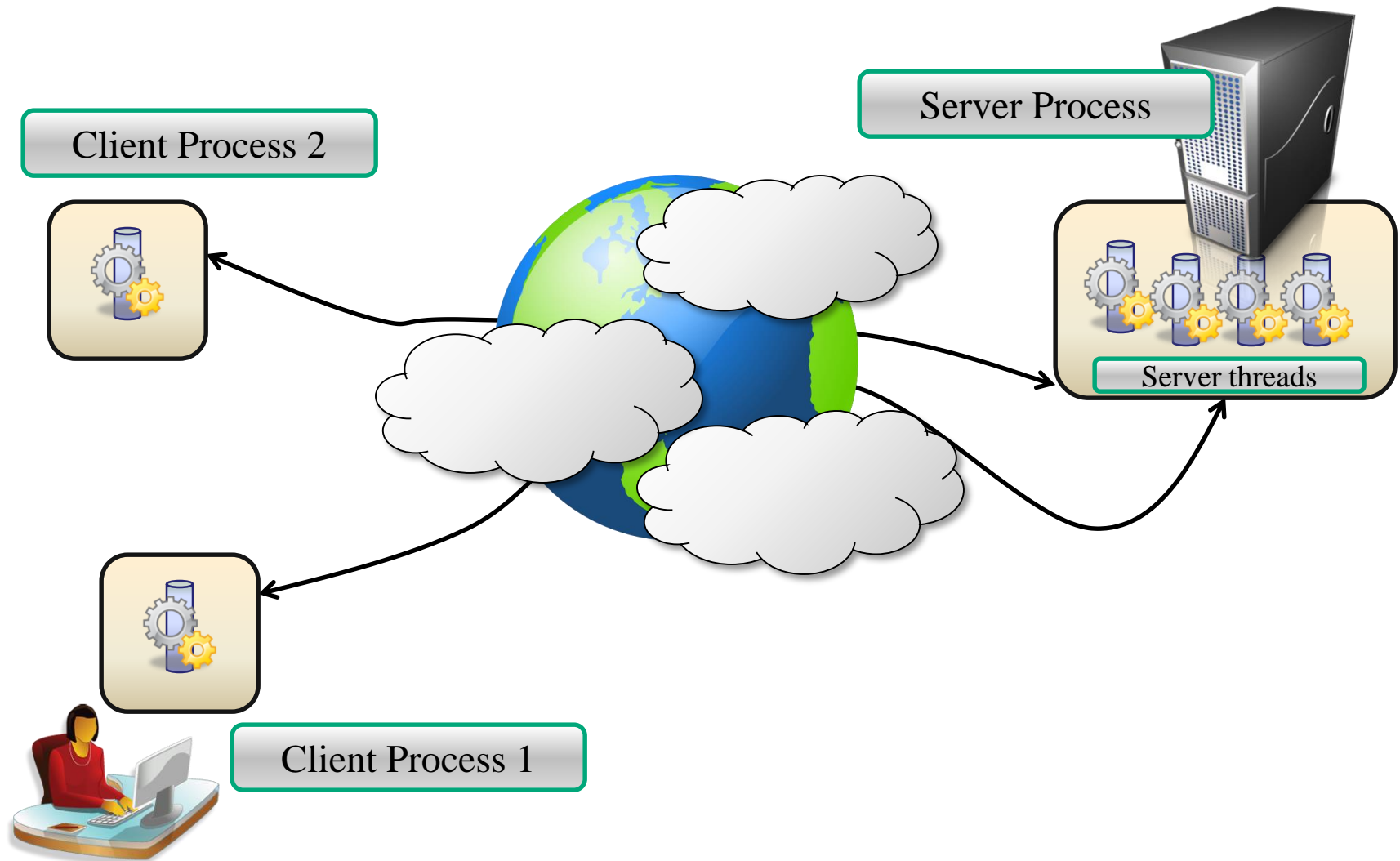
```
import java.net.*;
import java.io.*;
public class UDPClient
{
    public static void main(String args[]){
        // args give message contents and server hostname
        // "Usage: java UDPClient <message> <Host name> <Port number>"
        DatagramSocket aSocket = null;
        try {
            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



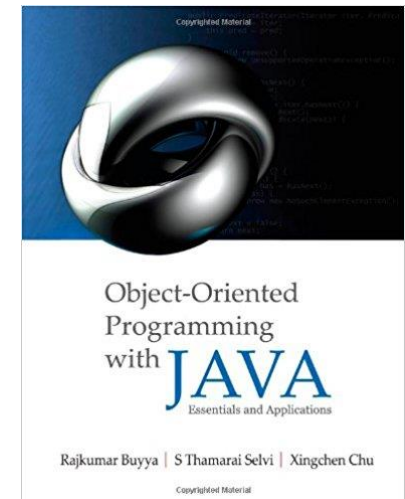
# Summary



- 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



- 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/>



## Exploring an Interactive Client/Server

### **Client:**

1. Create a socket specifying the server address and port
2. Read data from user inputs using the Scanner class
3. Write data using the stream associated with the socket

### **Server:**

1. Create a listening socket bound to a server port
2. Wait for clients to request a connection (Listening socket maintains a queue of incoming connection requests)
3. Server accepts a connection and creates a new stream socket for the server to communicate with the client. A pair of sockets in client and server are connected by a pair of streams, one in each direction. A socket has an input stream and an output stream.

# Paper Review (Assignment 2)



- Golgi: Performance-Aware, Resource-Efficient Function Scheduling for Serverless Computing
- Lifting the Fog of Uncertainties: Dynamic Resource Orchestration for the Containerized Cloud
- $\mu$ ConAdapter: Reinforcement Learning-based Fast Concurrency Adaptation for Microservices in Cloud
- Is Machine Learning Necessary for Cloud Resource Usage Forecasting?
- LatenSeer: Causal Modeling of End-to-End Latency Distributions by Harnessing Distributed Tracing
- Gödel: Unified Large-Scale Resource Management and Scheduling at ByteDance
- Carbon Containers: A System-level Facility for Managing Application-level Carbon Emissions