Distributed Information System

(SID)

Directed by Mr *A. Belgacem*

Academic year **2022/2023**

Plan

I. Introduction to distributed systems

II. Sockets

III. Remote method invocation (RMI)

IV. Common Object Request Broker Architecture (CORBA)

V. Java DataBase Connectivity (JDBC)

History 1960

1980 *1985* *1991* *1999-…*

Mainframe

Personal computer

Internet

ÿ Networks: (LAN, WAN,...)
ÿ Increase in computing power. ÿ Changing needs: ÿRemote access ÿCollaboration ÿMore resources

Grid Computing 1991

cloud computing 2010

Internet of Things 2017

etc

What is a Distributed System ?

ÿ Physically, a distributed system is made up of distributed computers, each of which can execute tasks (processes) in competition with others.

ÿ "A distributed system is a system that prevents me from working when a machine that I have never heard about breaks down " *Leslie Lamport*

Process: set of instructions executable by a computer.

It is a running program.

Process (dynamic) ÿ Program (static)

What is a Distributed System ?

A distributed system meets the following criteria:

- **_Multiple processes :_ the system consists of several sequential processes. These processes can be either system processes or user processes.**
- **_Interprocess communication :_ Processes communicate with each other using messages that take a limited time to move from one process to another (the absence of a common clock). These message links are also called channels.**
- **_Disjoint address spaces :_ Processes have disjoint address spaces (the absence of common physical memory).**
- **_Collective objective :_ the processes must interact with each other to achieve an objective commmon.**
- **_Heterogeneity :_ The presence of different hardware and software environments that need to be integrated**

Why the distributed system ?

- *Geographically distributed environment :* in many situations, the computing environment itself is distributed geographically.
- *Acceleration:* it is necessary to speed up the calculation (image processing, bigdata, etc).
- *Resource sharing :* there is a need for resource sharing. Here, the term resource represents both hardware resources (Ex: print, server) and software (Ex: base of data).
- *Transparency:* the use of remote resources is imperceptible to the user
- *Fault tolerance:*
  - ÿ High availability ÿ
  - The failure of one machine does not affect the others

Examples of distributed systems

- *Internet:* **interconnection of local networks**

- ***Peer-to-peer networks :*** **P2P systems are very popular for sharing files, content distribution and Internet telephony (Ex: Audiogalaxy and Napster).**

- ***Real-time distributed systems :*** **Media (digital decoders), Services telephones (GSM terminal, PABX), Production systems industrial (nuclear power plant, assembly line, chemical plant, etc.).**

Examples of distributed systems

o ***Sensor*** **networks: These networks can potentially be used in a wide class of problems: battlefield surveillance, detection of biological and chemical attacks, medical care health, home automation, ecology and home monitoring.**

o ***Social networks:*** **Millions of users now use their desktop or laptop computers or their smartphones to post and exchange messages, photos and video clips with their friends using these networking sites.**

o ***Grid computing and cloud computing:*** **This is a form of distributed computing that supports the parallel programming on a computer network.**

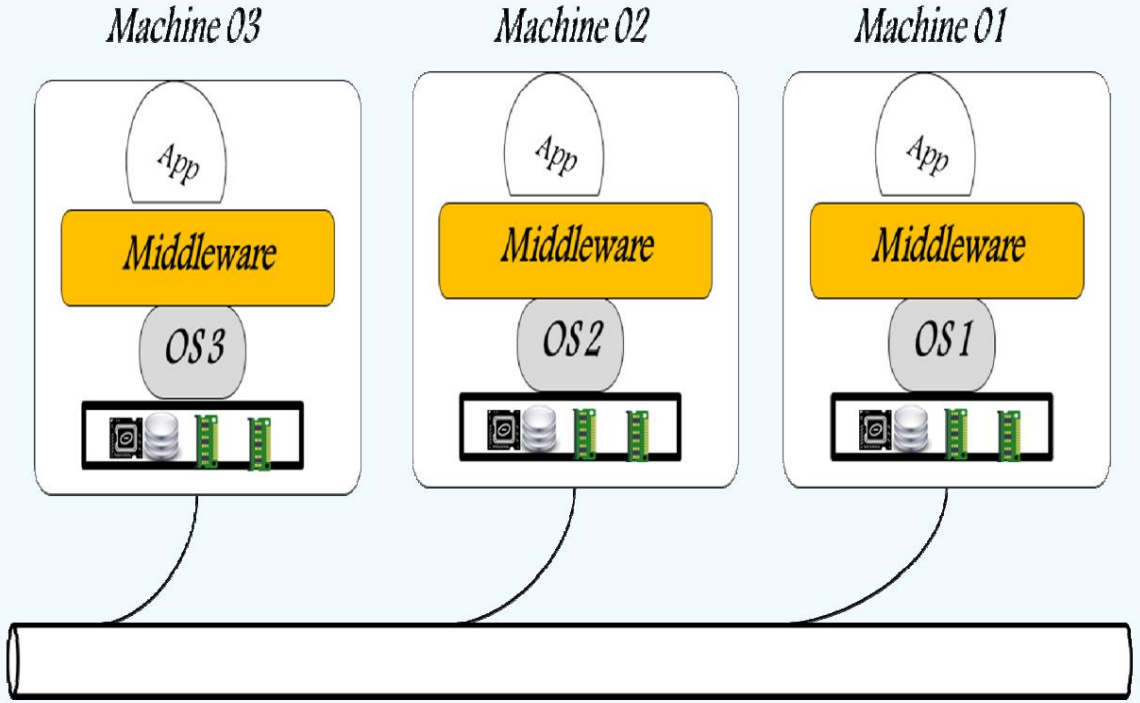
ÿThere are other examples: Amazon Web Services (AWS), Distributed Compute, etc.

Conceptual challenges of distributed systems

Interoperability

ÿ It is the ability to render components
compatible with each other, using for example:
middleware (CORBA, DCOM, etc) and machines
virtual.

ÿ Middeleware is a software layer that
provides a more programming model
practice in masking heterogeneity



Conceptual challenges of distributed systems

The opening

An open system must support:

ÿ The addition/removal of physical (computers or others) or logical (OS, middleware, etc.) components. ÿ Update (even re-implementation) of old services.

Scalability

A system is said to be scale invariant if it remains efficient after adding a large number of its users and the resources it manages. The cost of the addition is proportional to the number of users and resources added. The system must maintain its performance and show a reasonable decrease following the addition

Conceptual challenges of distributed systems

Security

A distributed system must provide message security when communicating between its components.

Unfortunately, during its transmission, a message can be intercepted by malicious entities.

Which may leak its contents and threaten the security of the distributed system.

Competition management

The sharing of resources can cause competition problems, this is the case when one wants

modify information found in a database, for example, or when you want to use the

even print (Resource Allocator).

Failures

A failure occurs when a system as a whole or one or more of its components fails

not behave according to their specifications.

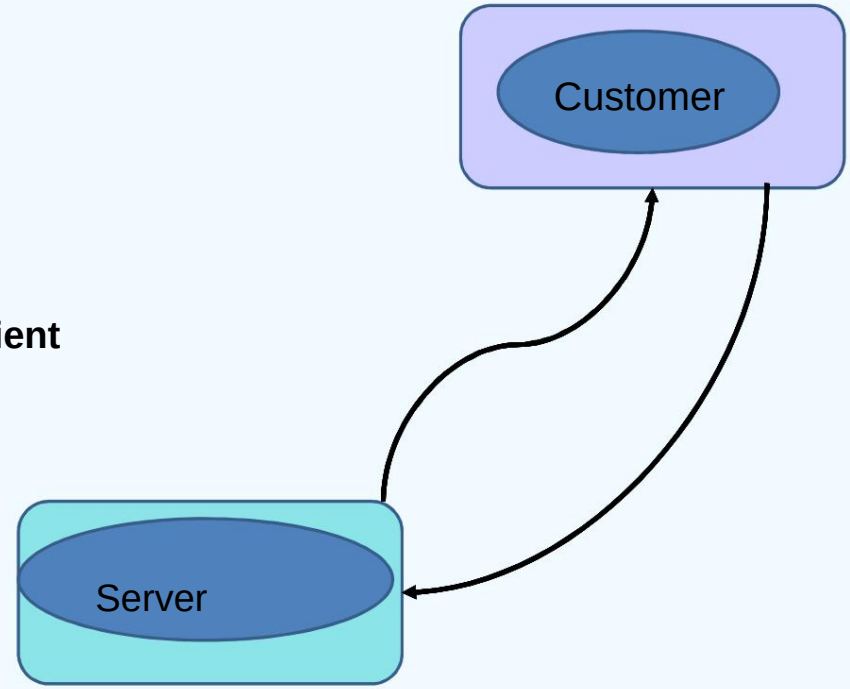
The architecture of distributed applications

Client Server Architecture

There are two roles for the processes of a distributed application

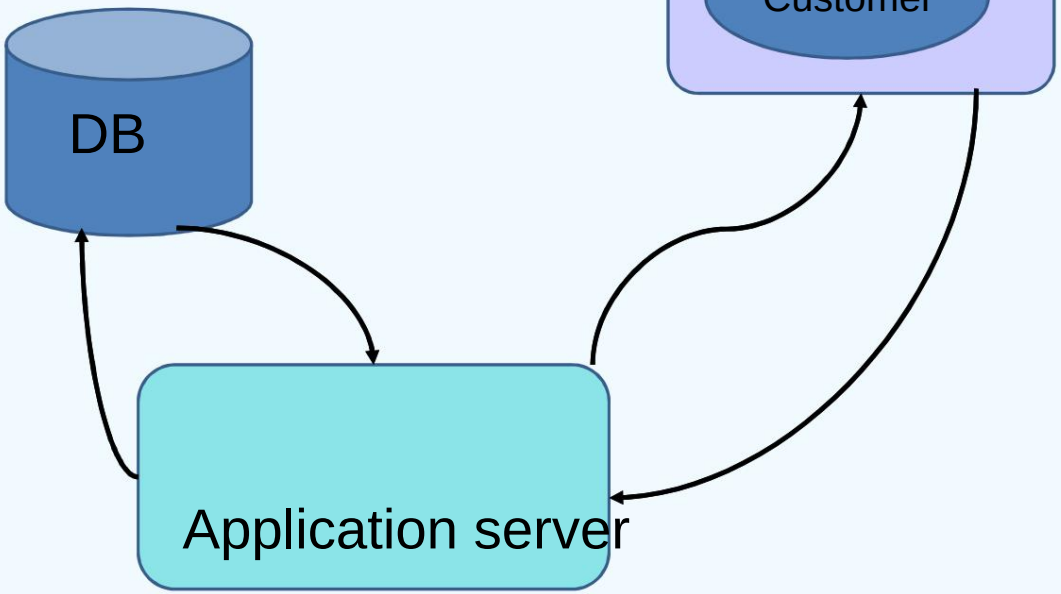
- ÿ **Client:** request for services.
- ÿ **Server :** offers services.

When the server requests services from another server becomes a client

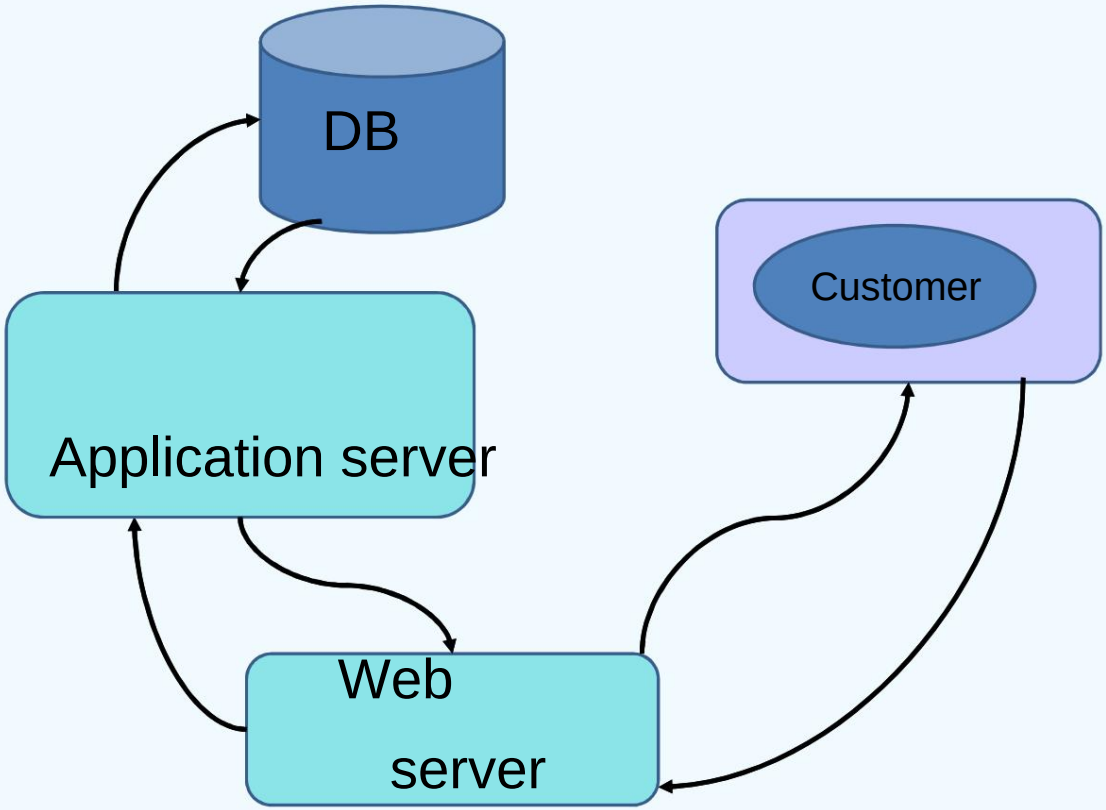


The architecture of distributed applications

Architecture 03 tier



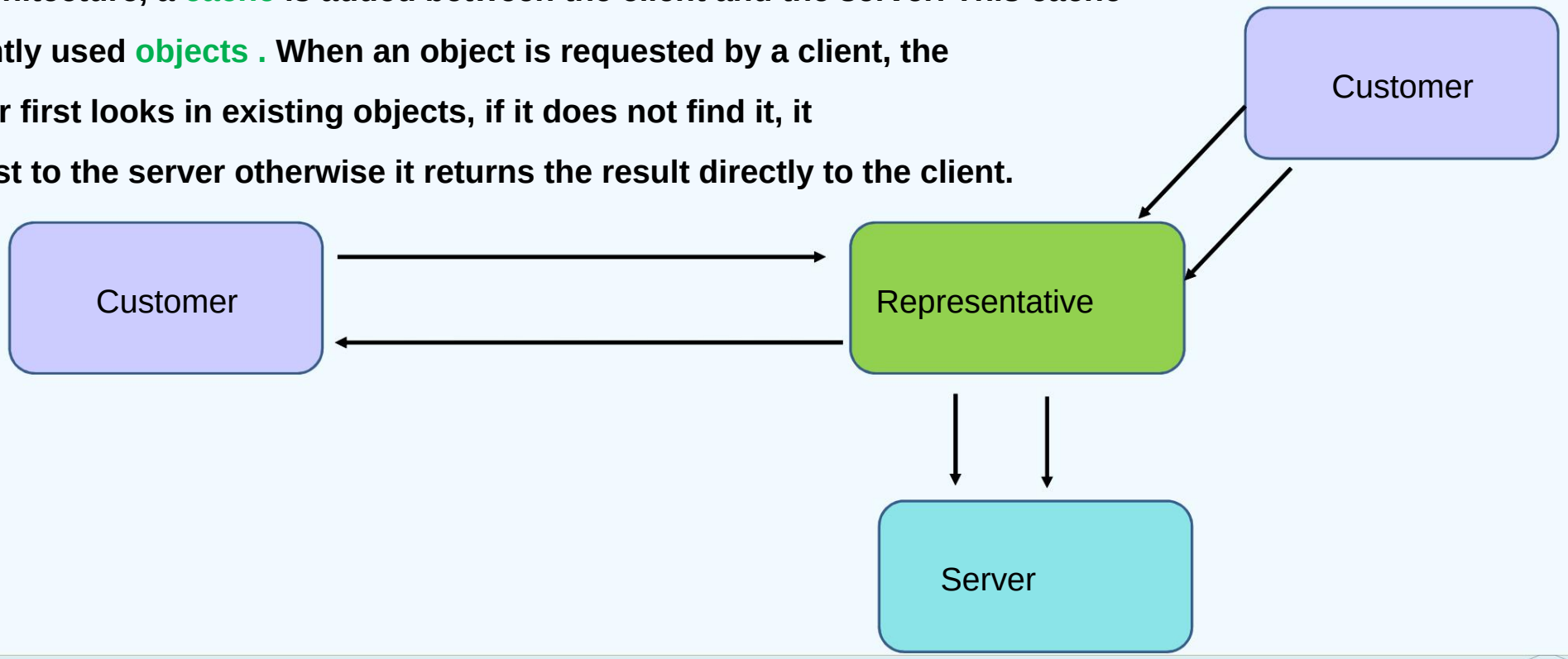
Whole Architecture



The architecture of distributed applications

Proxy Architecture

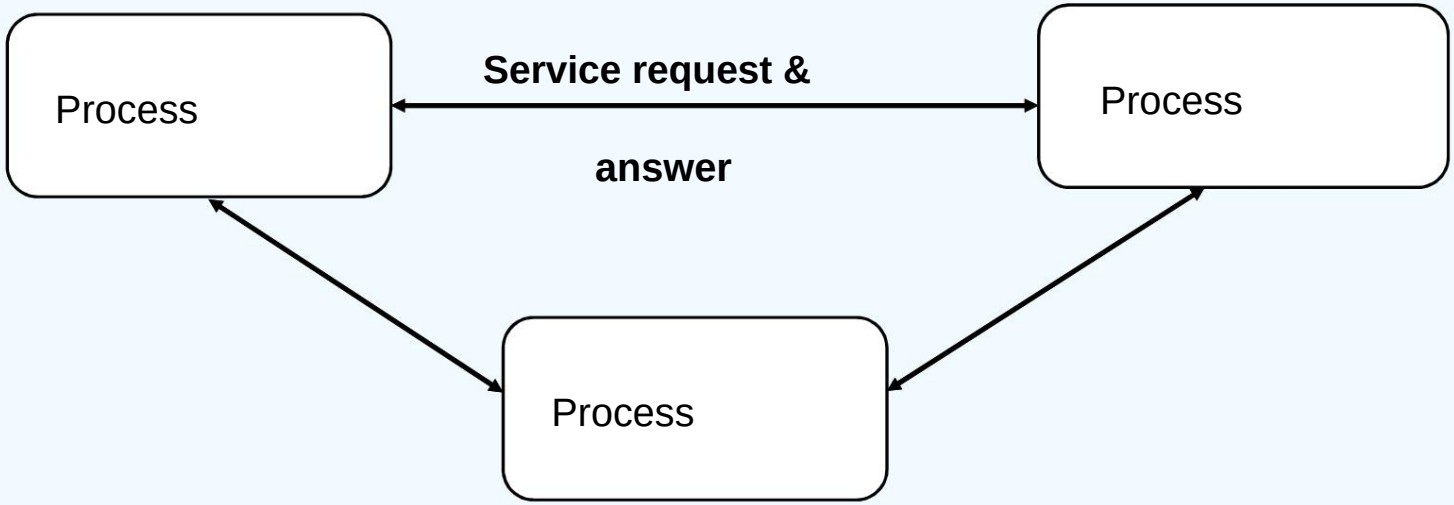
In this architecture, a cache is added between the client and the server. This cache contains recently used objects . When an object is requested by a client, the cache manager first looks in existing objects, if it does not find it, it sends a request to the server otherwise it returns the result directly to the client.



The architecture of distributed applications

Pair to Pair Architecture (Peer to Peer P2P)

In this architecture, the processes share local resources (files, space of storage, etc.) over a TCP/IP connection. Each of these processes can play both roles at once: the client-server. He is a server of what he has and a client of what others want to share. THE operations cooperate as equals to achieve a distributed activity.



Interprocess communication

Communication between distributed entities consists of the exchange of messages. They can be used for various purposes: synchronization, service request, return of results, etc.

We distinguish two types of communication:

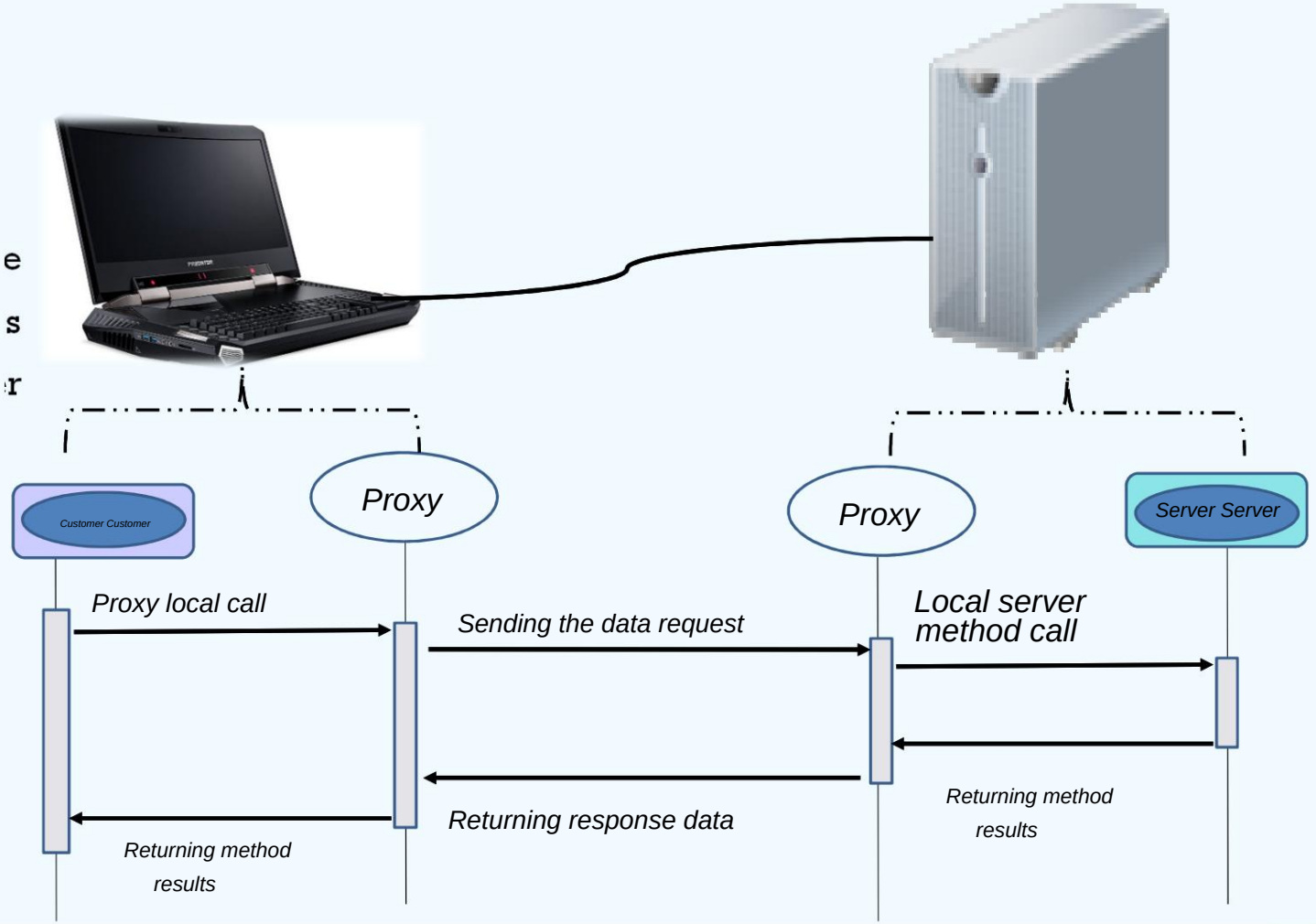
Synchronous communication: the transmitter is blocked until the receiver receives the message and vice versa.

Asynchronous communication: the sender is not blocked. For this, the system operating is responsible for sending the message to its destination.

Interprocess communication

Interprocess communication by calls distance

In a distributed system, for processes are called by remote methods, we need to have a mechanism for making a call from ordinary methods regardless of the location of the summoned object and its implementation. To solve these problems, we can use objects intermediaries called Proxy who deal of the connection.



Interprocess communication

Interprocess communication by remote call

The connection is established according to the proxy implementation technology, there are three choices:

RMI: for calling methods between distributed Java objects.

CORBA: to call methods independently of their programming language.

SOAP: to call methods independently of their programming language. But he uses an XML-based submission format.

For the last two models, it is advisable to provide a description of the interface to specify the signature methods and the data type that manages the objects. These descriptions are formatted in a language called Interface Definition Language (IDL) for CORBA and Web Service Description Language (WSDL) for SOAP.

Interprocess communication

Interprocess communication by shared distributed memory

In a distributed system, there is no physical sharing of memory. However, it will be possible to create a shared virtual memory that allows communication between processes. Memory shared can be created by adapting a hardware (multiprocessor machine) or software solution (middleware).

Interprocess communication by event notification

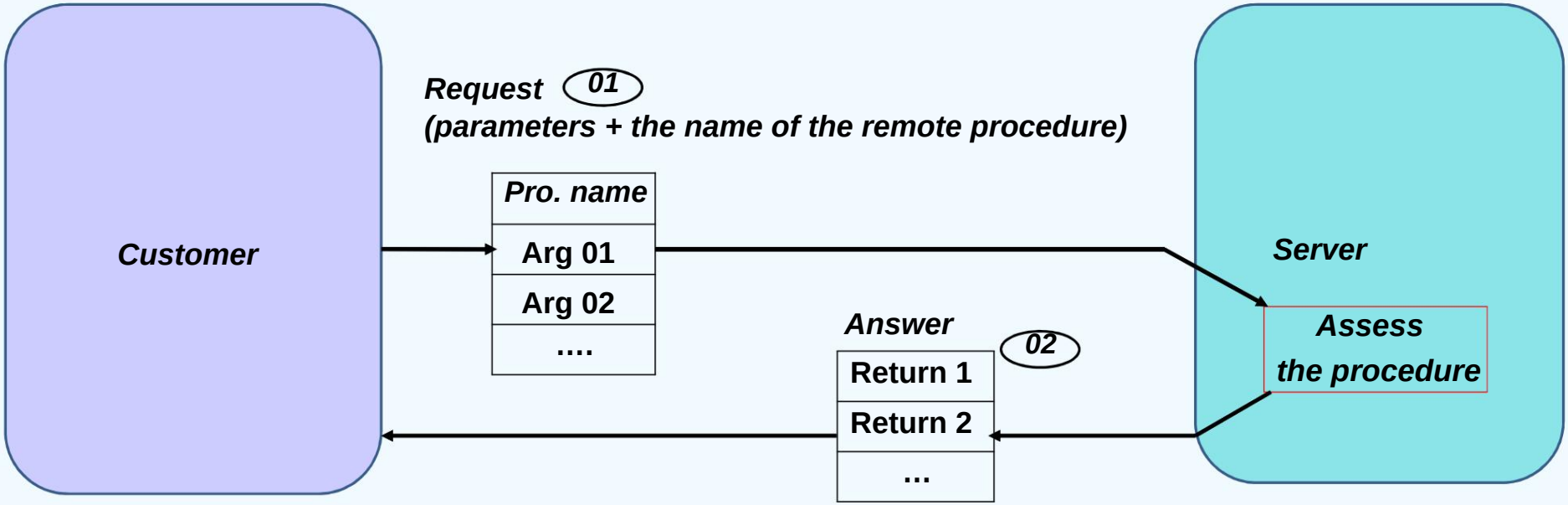
Communication between remote objects can be done using the publication/notification principle. THE objects generating events (method execution for example) can send notifications to objects interested in these events. An object wishing to receive a notification must take out a subscription to the type of poisoning that interests him (ex: Java message service, CORBA Event service, etc.).

Message exchanges

- **Communications take place by sending messages between processes.**
- **Use the communication primitives of type send *(send)* and receive *(receive).***
  - **Messages can be transient or persistent:**
    - ***In a transient communication,* the message is rejected by a communication server as soon as it cannot be delivered to the next server or receiver.**
    - ***In persistent communication,* messages are not lost, but saved in a buffer for possible future retrieval (ex: Email).**
  - **Data Streams: In general, streams are sequences of data items. he can be considered as a virtual connection between a source and a sink (ex: video),**

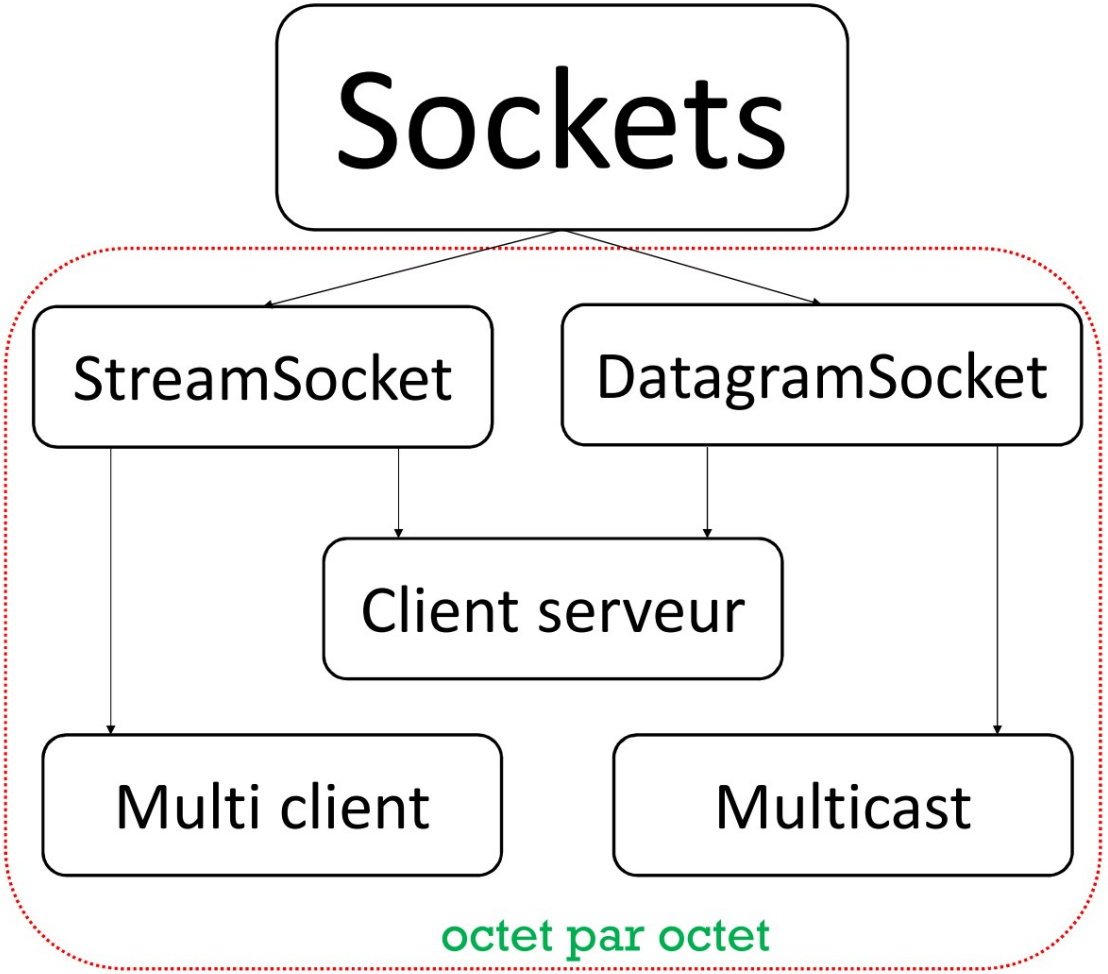
Use by: Sockets

Remote procedure call (RPC) technology



1. Marshalling refers to copying parameters into a buffer in a format suitable for the network transmission.
2. When the request arrives at the server, the parameters are unmarshalled and the server evaluates the procedure.

ÿUse by: CORBA, RMI



What is a socket?

ÿ **Sockets are open communication ports at the network layers of the system of exploitation, and making it possible to pass flows (streams) of bytes.**

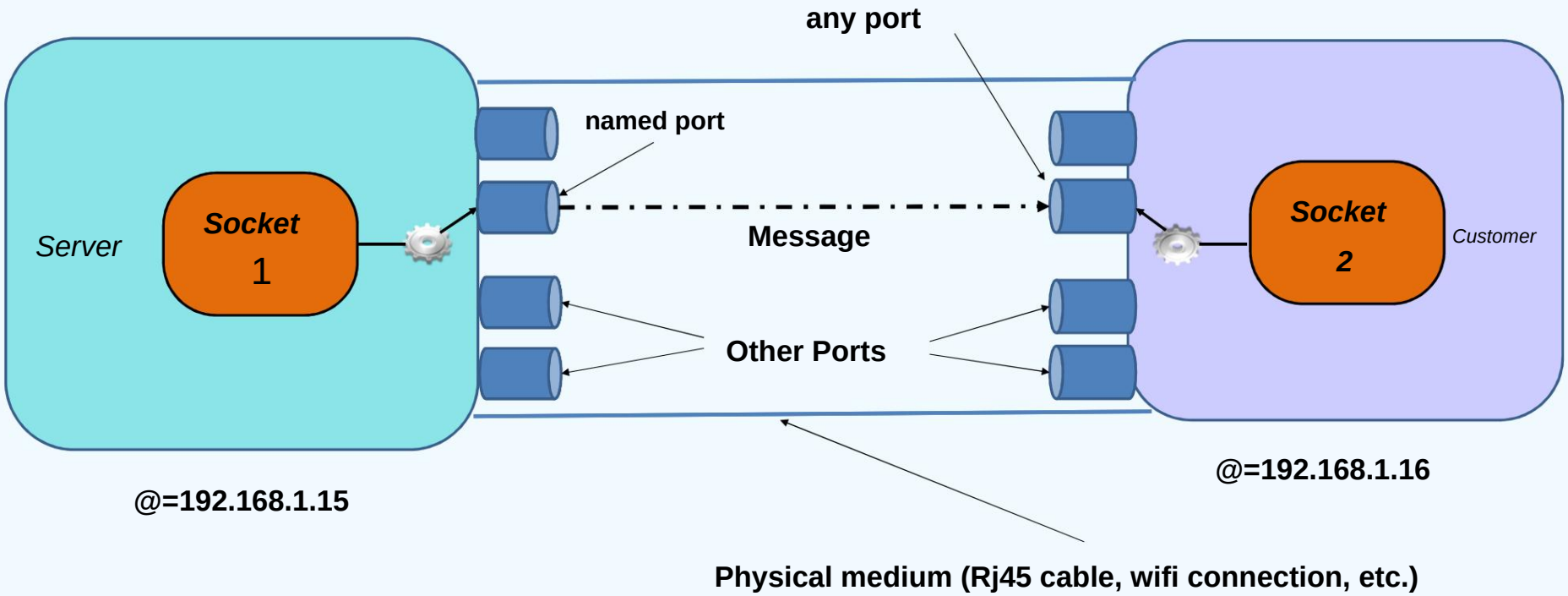
ÿ **Sockets allow point-to-point** communication in **client/server mode.**

ÿ **There are basically two socket communication protocols, one is TCP (Transmission Control Protocol), the other by UDP (User Datagram Protocol).**

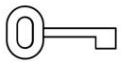
ÿ **A server is defined by a** communication **port on a given machine.**

ÿ **Communication is done through messages.**

Addresses, ports and socket



Addresses, ports and socket

Communication via socket	*Send a letter*	
The app	**You (the person)**	
IP address	**Apartment building address CITE 408 LOGEMENTS BOUMERDES**	
The port	**Letter box**	
Network	**Post Office**	
Socket	**The key that gives you access to the mailbox**	

Socket Types

ÿ Sockets are of two types: **stream sockets** and **datagram sockets.**

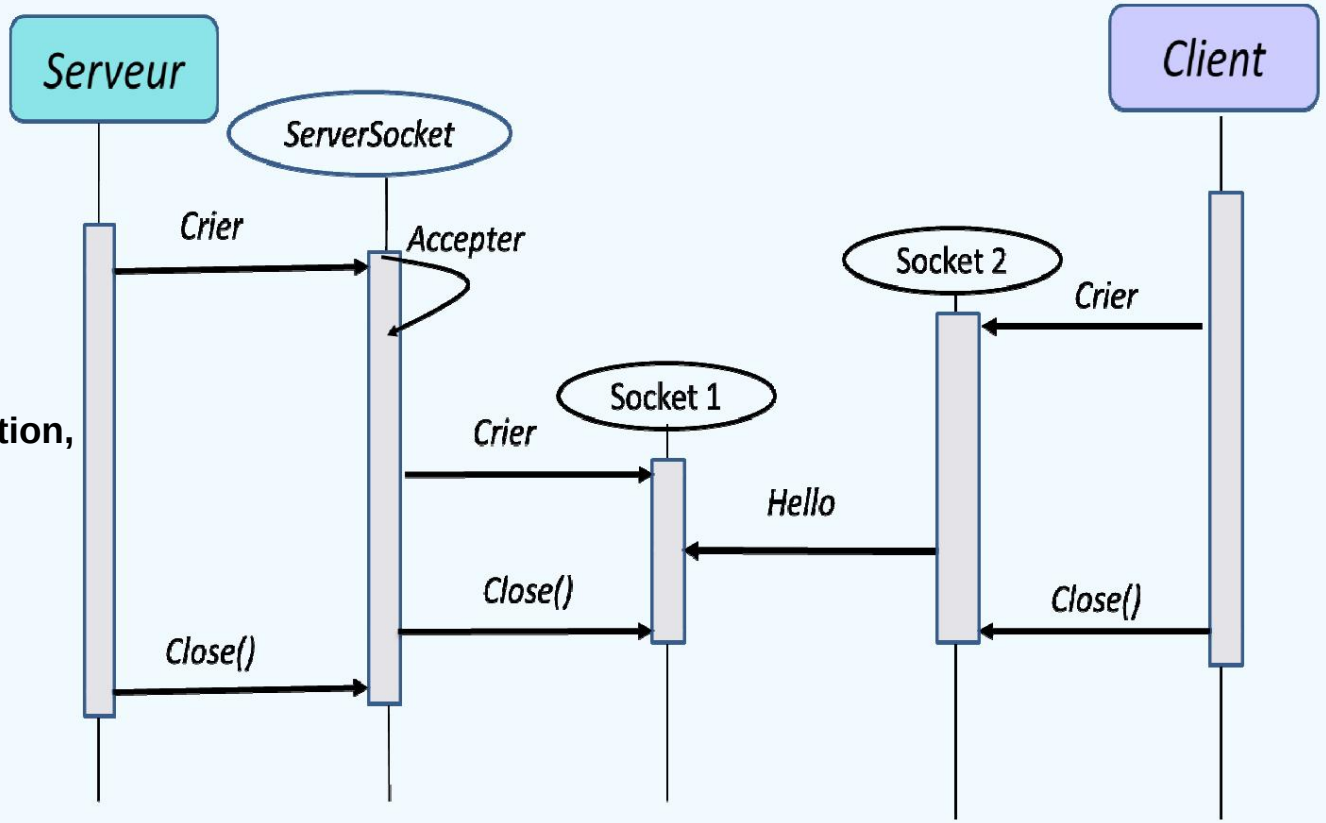
Stream socket (oriented socket)	*Datagram socket (connectionless socket)*
ÿ It provides reliable network service and connected	ÿ It provides unreliable network service
ÿ Uses TCP	ÿ Packets can be lost (uses UDP)
ÿ Applications: telnet / ssh, http, …	ÿ Applications: audio/video streaming (real time),…

Communication with TCP (Streamsocket)

Client-server mode

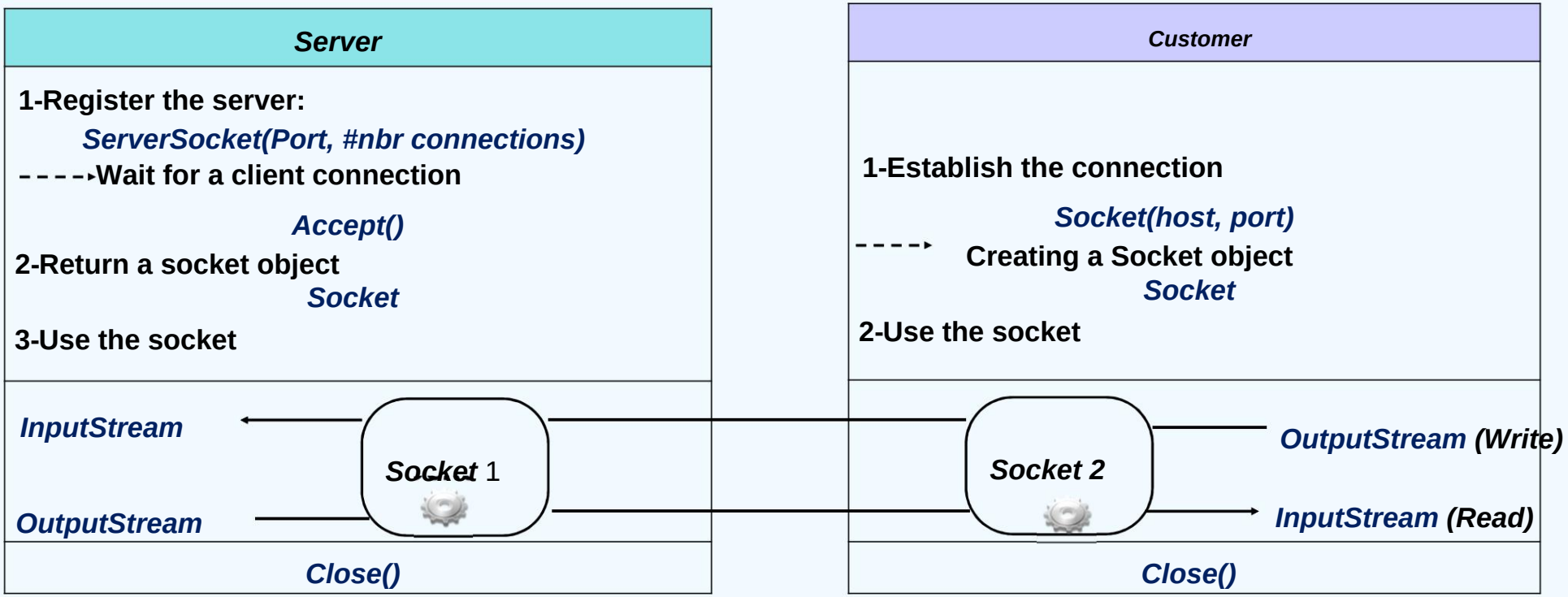
ÿ **The server will listen to requests from clients on an** application- **defined port .**

ÿ **The customer has the initiative of the conversation, but both client and server can send socket** information **to any time, because the flows (stream) in each direction are independent.**



Communication with TCP (Streamsocket)

Client-server mode



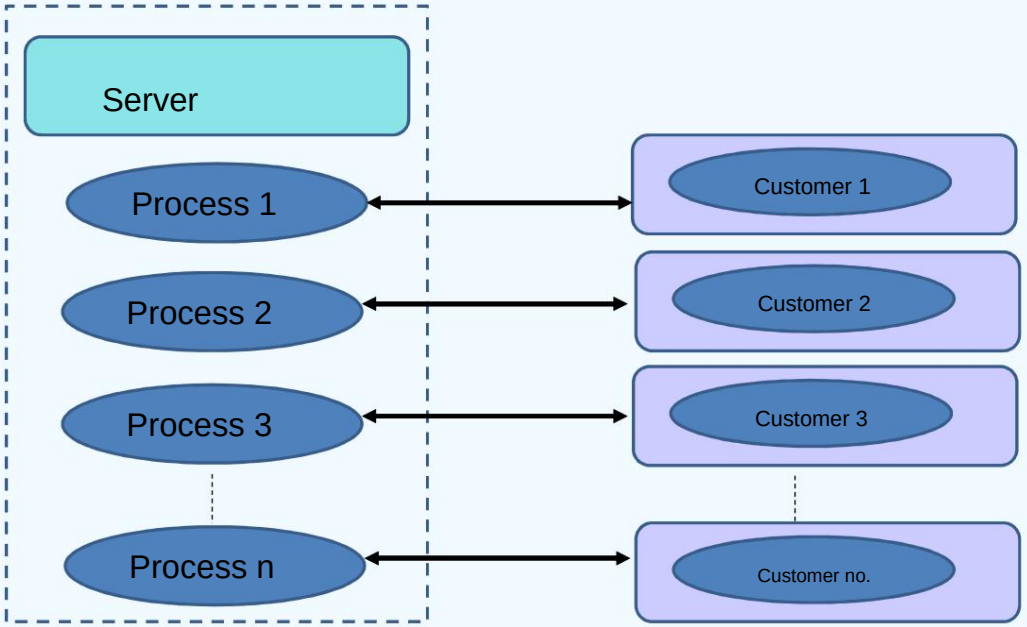
ÿ *In this mode of communication, the server serves a single client and then stops. It therefore has little interest in convenient.*

Communication with TCP (Streamsocket)

Connecting multiple clients (Multiclient)

ÿ **The server uses a *Connection* class implementing the Runnable interface *(thread)* to manage the exchanges background data. It is this *thread* (java *(java.lang.Thread))* that performs the requested service.**

ÿ **This allows the server to accept multiple connections simultaneously and to manage several clients at the same time.**



Communication with TCP (Streamsocket)

Connection of several clients (Multiclients)

```
try
{ serverSocket = new ServerSocket(2023); do {


        // Wait for customer...
    client socket = serverSocket. accept();
  PrintWriter out = new PrintWriter(client.getOutputStream(), true); //
  Create a thread to handle communication with // this client and pass the
  constructor for it // thread a reference to the concerned socket...

  ClientConnection_Handling handler = new ClientConnection_Handling(client);
    handler.start();// Run method call.

    } while (true);
}
```

Communication with TCP (Streamsocket)

Connecting Multiple Clients (Multi-Client)

```
class ClientConnection_Handling implements Runnable{

public ClientConnection_Handling(Socket s)
        { this.s = s;
 try
     { in=new BufferedReader (new
InputStreamReader(s.getInputStream()));

out = new PrintWriter(s.getOutputStream()); }

     catch (IOException e) { ... }
}
```

```
public void run()
{ Try { while (true)
      { String line =
            in.readLine(); if (line ==
             null) break; out.println(line );


        } } catch (IOException e) {s}

 finally
{ //end of client-side
      connection try
    {s.close(); } catch
          (IOException e){ …} }
```

Communication with TCP (Streamsocket)

Connecting Multiple Clients (Multi-Client)

```
try
    { socket = new Socket("localhost",2023);
      Scanner networkInput = new Scanner(socket.getInputStream());
  PrintWriter networkOutput = new PrintWriter(socket.getOutputStream(),true);

 // Configure Stream for keyboard input...
           Scanner userEntry = new Scanner(System.in);
           String message, response;
do {
            message  = userEntry.nextLine();
          networkOutput.println(message); =
          response }while(!message.equals("QUIT")); }
       catch(IOException ioEx) { …}
```

```
// Close
 finally
   { try { socket. close();
 catch(IOException ioEx) {

          …} }
```

Communication with UDP (Datagram socket)

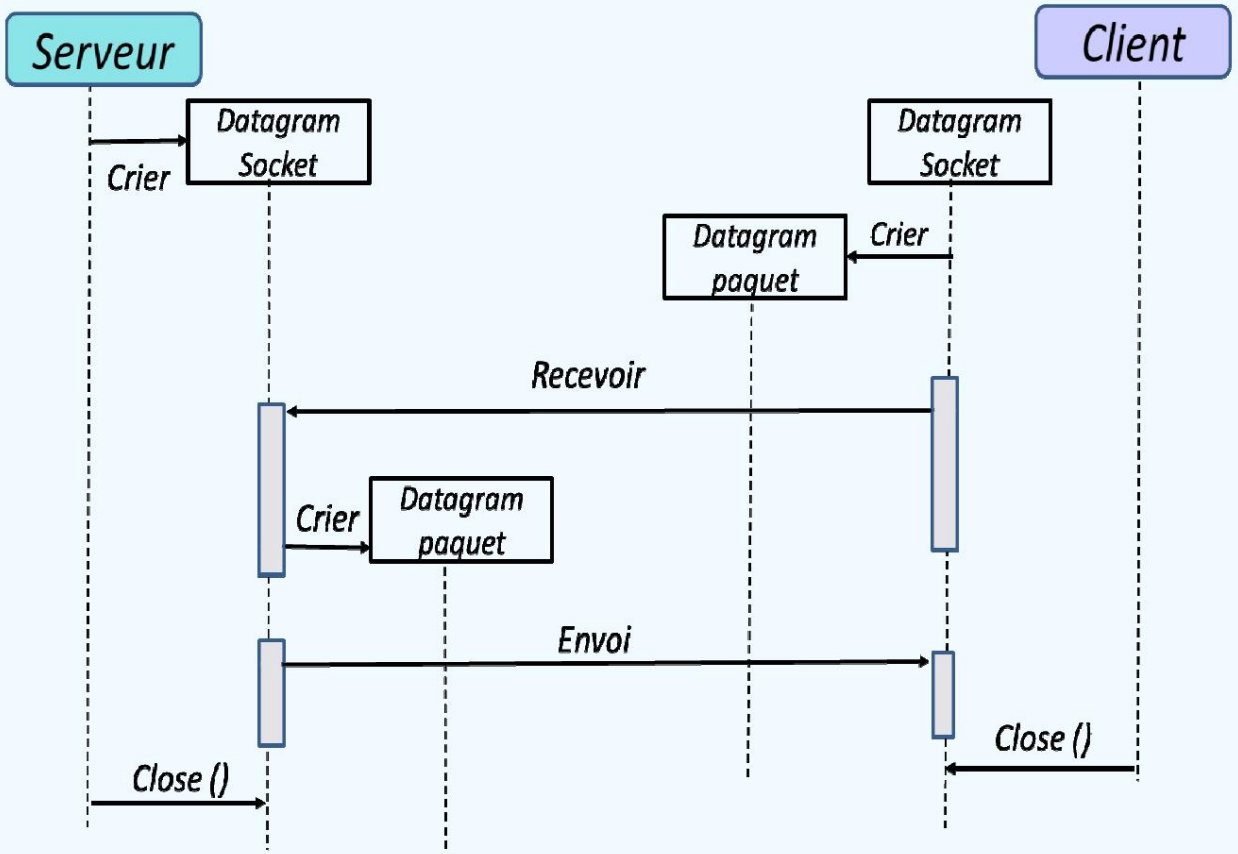
Client server

ÿ This protocol requires segmenting packet information

ÿ It is widely used in the multimedia world, because it corresponds well to a mode of fast data transmission as well nature and in packets (datagram).

ÿ We use the classes DatagramPacket and DatagramSocket

ÿ These objects are initialized differently depending on whether they are used to send or receive packages



Communication with UDP (Datagram socket)

Client server

ÿ Sending a Datagram:

1. Create a *DatagramPacket* by specifying:

- **the data to send**
- **their length**
- **the receiving machine and the port**

2. Use the *send (DatagramPacket)* method of *DatagramSocket.*

for the constructor because

all the information can be found in

the packet sent.

```
// Recipient machine
InetAddress address = InetAddress.getLocalHost();
static final int PORT=2023;

// Create the message to send
String s = new String ("Message to send"); int
 length = s.length(); byte[] message = new
 byte[length]; s.getBytes(0,length,message,0);


// Initialize the package with all the information
DatagramPacket packet = new
DatagramPacket(message,length,address,PORT); • no arguments

// Create socket and send packet
DatagramSocket socket = new DatgramSocket();
socket.send(packet);....
```

Communication with UDP (Datagram socket)

Client server

ÿ Receiving a Datagram:

1. Create a *DatagramSocket* which listens on the port of the recipient's machine

2. Create a *DatagramPacket* to receive the packets sent by the server

- **size the buffer large enough**

3. Use DatagramPacket's receive() method

- **this method is blocking**

```
// Define a receive buffer byte[]
buffer = new byte[1024];

// A packet is associated with an empty buffer for reception
DatagramPacket packet =new DatagramPacket(buffer,buffer.length());

// We create a socket to listen on the port
DatagramSocket socket = new DatagramSocket(PORT);
while (true) {

// Wait to receive
socket.receive(packet);

// Display of received packet
String s = new String(buffer,0,0,package.getLength());
System.out.println( "Packet received:");
```

Communication with UDP (Datagram socket)

Multicast

Sending a Datagram: ÿ

It is possible to use multicast addresses allowing the simultaneous broadcast of a information to multiple recipients.

ÿ There are reserved address ranges for multicast addresses (224.0.0.0 to 239.255.255.255).

ÿ All active clients at the time of issuance and who listen on this address will receive the same information simultaneously .

```
DatagramSocket socket = new DatagramSocket();
byte [ ] buf = "Hello". getBytes();
InetAddress group = InetAddress . getByName("230.0.0.2");
DatagramPacket packet = new DatagramPacket( buf ,
buf.length, group, 2023);
socket. send(packet);
```

Here the multicast address used is 230.0.0.2.
Datagrams are also used.

Communication with UDP (Datagram socket)

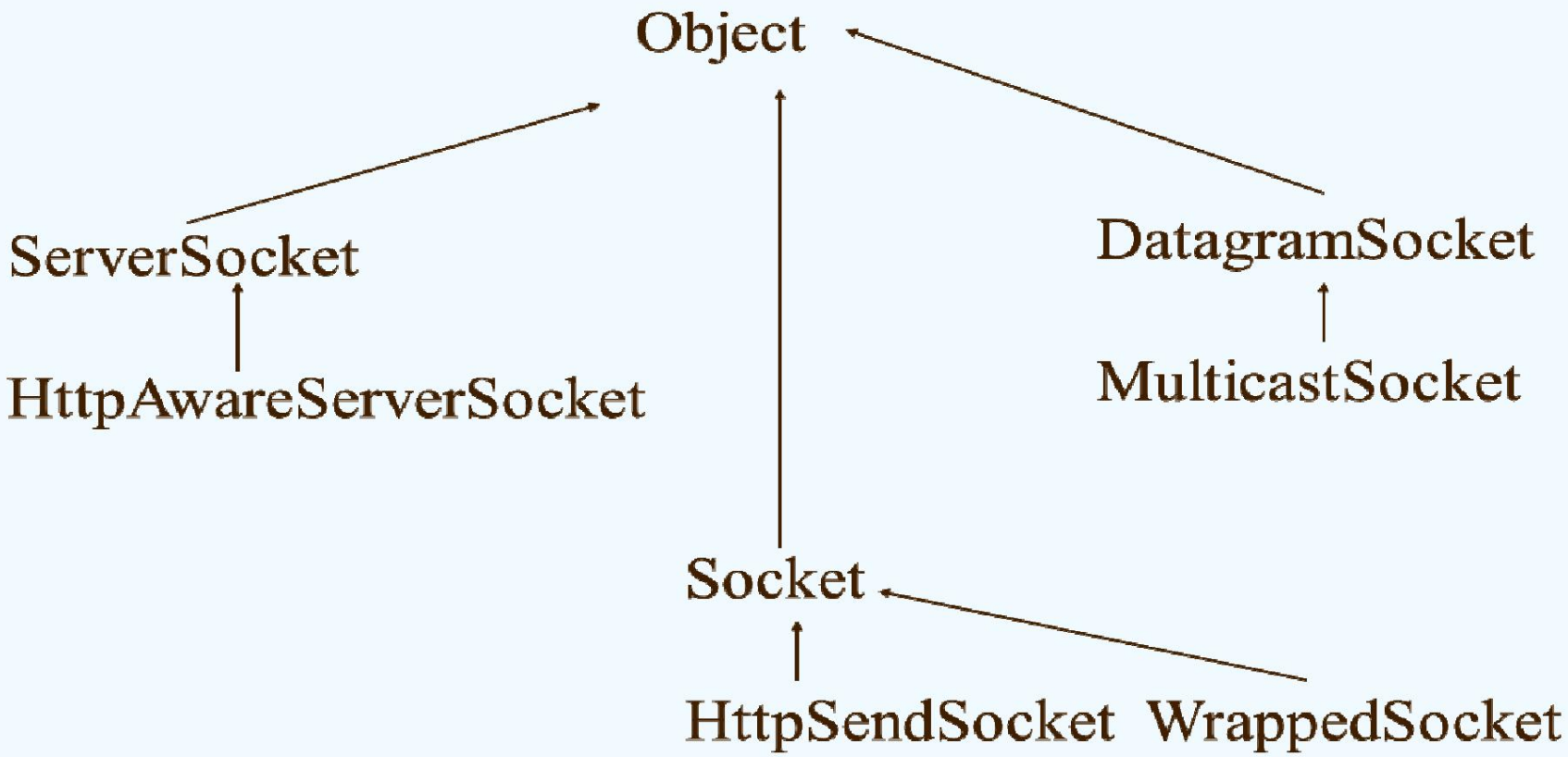
Multicast

Receiving a Datagram:

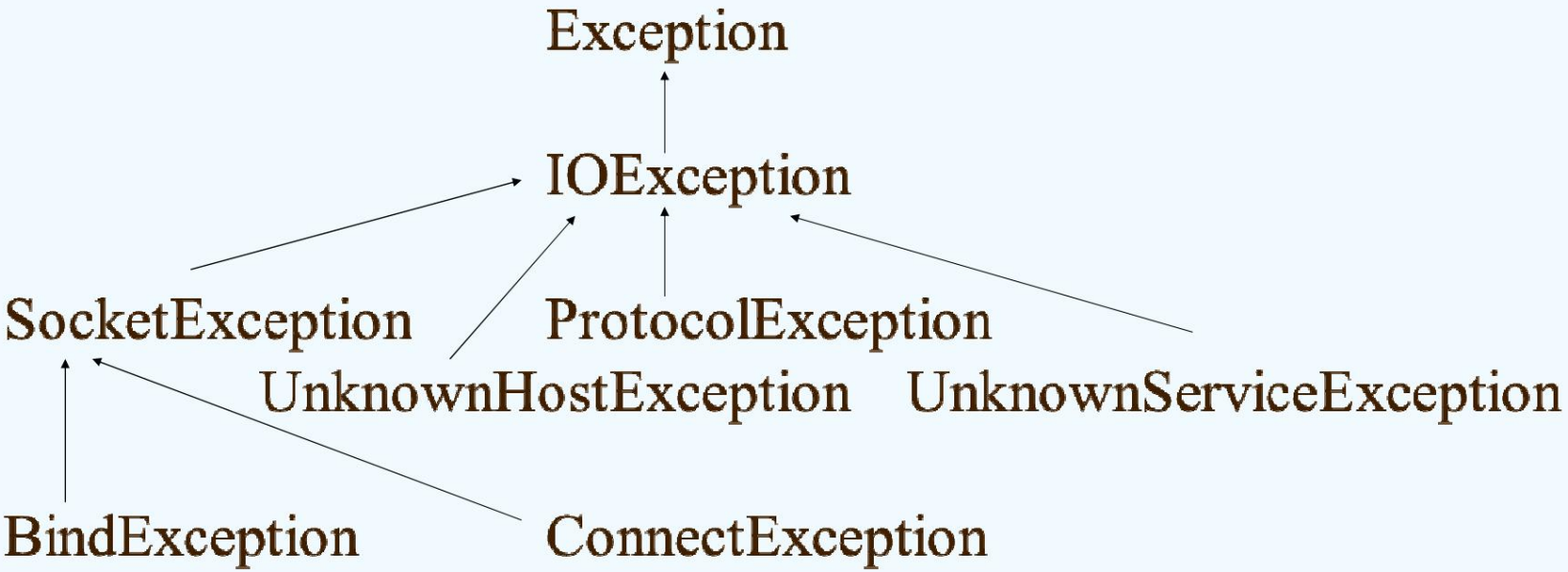
ÿ **Apart from dissemination of information digital, multicast is also widely used for clustering servers or database replication data.**

```
InetAddress addr= InetAddress . getByName("23 0.0.0 .2");
InetSocketAddress address = new InetSocketAddress(addr , 2023);
MulticastSocket socket= new MulticastSocket(2023);
socket. joinGroup(addr);
byte [ ] buf = new byte [256]
DatagramPacket packet = new DatagramPacket(buf, buf .length );
socket. receive( packet );
System.out.println(newString(packet.getData()));
```

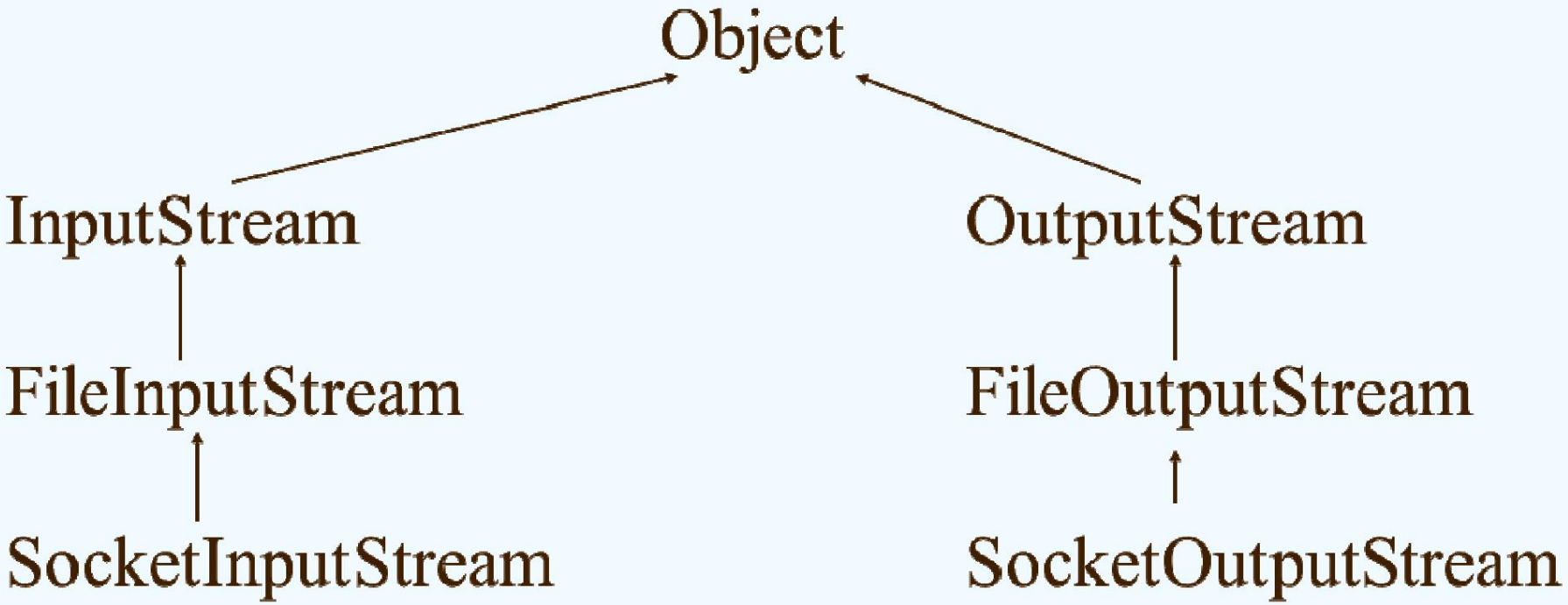
Types of Sockets



Exceptions



Inputs Outputs



How to create an InputStream?

Client side: to receive a response from the server

```
try {input = new
DataInputStream(MyClient.getInputStream());}

catch (IOException e) {System.out.println(e);}
```

DataInputStream : read lines of text, integers,

doubles, characters...

(read, readChar, readInt, readDouble, and readLine,.)

Server side: to receive data from a client

```
DataInputStream input;

try
 { input = new
DataInputStream(serviceSocket.getInputStream());
    }
catch (IOException e) {System.out.println(e);}
```

How to create an OutputStream?

Client side: to send information to the server

(PrintStream or DataOutputStream)

```
PrintStream output;
try {
output = new
PrintStream(MyClient.getOutputStream());
}
catch (IOException e) {System.out.println(e);}
```

PrintStream to display values of types of

database (write and println)

Server side: To send information to the client

```
PrintStream output; try
 { output = new
PrintStream(serviceSocket.getOutputStream());
      }
 catch (IOException e) {System.out.println(e);}
```

DataOutputStream : write primitive data types;

(writeBytes…)

```
output= new
DataOutputStream(serviceSocket.getOutputStream());
```

Other inputs outputs

```
echoSocket = new Socket( "karim", 3);
    out =
     new PrintWriter(echoSocket.getOutputStream(), true);
    in =
   new BufferedReader(new InputStreamReader(
                        echoSocket.getInputStream()));
```

How to close a socket?

Close the output and input stream before the socket.

Client side

```
output.close();
 input.close();
 MyClient.close();
```

Server side

```
output.close();
 input.close();
 serviceSocket.close();
 MyService.close();
```

How to program a client?

Always 4 steps

- **ÿOpen a socket.**
- **ÿOpen an input and an output stream on the socket.**
- **ÿRead and write on the socket *(changes according to the targeted server)***
- **ÿDelete Close**

A few words about Threads

- **Use threads to accept multiple clients simultaneously.**
- **Thanks to which we can carry out several activities within the same process.**
- **An application can have multiple threads running concurrently (Each thread has a priority). • Each thread has a name. Multiple threads can have the same. The name is generated if not specified.**

- **There are 2 ways to create a new execution thread:**
  - ÿ declare a **Thread subclass and override the run method. An instance of the subclass can then be allocated and started.**
  - **ÿ declare a class that implements Runnable and therefore the run method. An instance of the class can be allocated, passed as an argument when creating a thread and started.**

- **Thread constructor**
  - **start**
  - **run**

A few words about Threads

```
// Implementing runnable interface by extending Thread class
public class ThreadTest extends Thread {
    // run() method to perform action for thread.
    public void run() { int X= 44; int Y=12; int
      result = X+Y;



      System.out.println("Thread started running..");
      System.out.println("Sum of two numbers is: "+ result); }
    public static void main( String args[] ) { // Creating
    instance of the class extend Thread class ThreadTest
     th1 = new ThreadTest(); //calling start method to
       execute the run() method of the Thread class
      th1.start(); }


}
```

A few words about Threads

```
public class ThreadTest implements Runnable
    { // run() method to perform action for thread.
    public void run() { int X= 44; int Y=12; int
       result = X+Y;



       System.out.println("Thread started running..");
       System.out.println("Sum of two numbers is: "+ result); }
    public static void main( String args[] ) { // Creating
    instance of the class extend Thread class ThreadTest
     th1 = new ThreadTest(); //calling start method to
        execute the run() method of the Thread class new
       Thread( th1).start();

    }
}
```

The ServerSocket class

Constructeur	Rôle
ServerSocket()	Constructeur par défaut
ServerSocket(int)	Créer une socket sur le port fourni en paramètre
ServerSocket(int, int)	Créer une socket sur le port et avec la taille maximale de la file fournis en paramètres

Méthode	Rôle
Socket accept()	Attendre une nouvelle connexion
void close()	Fermer la socket

The Socket class

Constructeur	Rôle
Socket()	Constructeur par défaut
Socket(String, int)	Créer une socket sur la machine dont le nom et le port sont fournis en paramètres
Socket(InetAddress, int)	Créer une socket sur la machine dont l'adresse IP et le port sont fournis en paramètres

Méthode	Rôle
InetAddress getInetAddress()	Renvoie l'adresse I.P. à laquelle la socket est connectée
void close()	Fermer la socket
InputStream getInputStream()	Renvoie un flux en entrée pour recevoir les données de la socket
OutputStream getOutputStream()	Renvoie un flux en sortie pour émettre les données de la socket
int getPort()	Renvoie le port utilisé par la socket

The DataInputStream class

Method	Role
DataInputStream(InputStream in)	Creates a DataInputStream that uses the InputStream.

Method	Role
int read(byte[]	It is used to read the number of bytes from the input stream.
b) int read(byte[] b, int off, int	It is used to read len bytes of data from the input stream.
len) int readInt() byte	It is used to read input bytes and return an int value.
readByte() char readChar()	It is used to read and return the input byte.
double readDouble() boolean	It is used to read two bytes of input and returns a char value.
readBoolean()	It is used to read eight bytes of input and returns a double value.
	It is used to read an input byte and return true if the byte is non-zero, false if the byte is zero.
int skipBytes(int x)	It is used to discard x bytes of data from the input stream.
String readUTF()	It is used to read a string that has been encoded in UTF-8
void readFully(byte[]	format It is used to read bytes from the input stream and store them in the buffer array
b) void readFully(byte[] b, int off, int len)	It is used to read len bytes from the input stream.

The DataOutputStream class

Builder	Role
DataOutputStream(OutputStream out)	Creates a new data output stream to write data to the underlying output stream specified.

Method	Role
void write(int	It is used to write the specified byte to the underlying output stream.
b) void write(byte[] b, int off, int len)	It is used to write len bytes of data to the output stream
void writeBoolean(boolean v) void writeChar(int v)	It is used to write a boolean to the output stream as a 1 byte value.
v) void writeBytes(String s) void writeInt(int v)	It is used to write a char to the output stream as a 2 byte value.
writeLong(long v) void writeUTF(String str)	It is used to write a string to the output stream as a sequence of characters.
	It is used to write a byte to the output stream as a 1 byte value.
	It is used to write a string to the output stream as a sequence of bytes.
	It is used to write an int to the output stream It
	is used to write a short circuit to the output stream.
	It is used to write a short circuit to the output stream.
	It is used to write a long to the output stream.
	It is used to write a string to the output stream using UTF-8 encoding in a portable way.

The PrintWriter class

Constructeur	Rôle
PrintWriter(Writer)	Le paramètre fourni précise le flux. Le tampon est automatiquement vidé.
PrintWriter(Writer, boolean)	Le booléen permet de préciser si le tampon doit être automatiquement vidé
PrintWriter(OutputStream)	Le paramètre fourni précise le flux. Le tampon est automatiquement vidé.
PrintWriter(OutputStream, boolean)	Le booléen permet de préciser si le tampon doit être automatiquement vidé

Méthode	Rôle
close()	Ferme le tampon et libérer les ressources associées
boolean checkError()	Vide le tampon et renvoie true si une exception est levée lors de l'utilisation du flux sous–jacent
flush()	Vide le tampon en écrivant les données dans le flux.

The BufferedReader class

Constructeur	Rôle
BufferedReader(Reader)	le paramètre fourni doit correspondre au flux à lire.
BufferedReader(Reader, int)	l'entier en paramètre permet de préciser la taille du buffer. Il doit être positif sinon une exception de type IllegalArgumentException est levée.

Méthode	Rôle
String readLine()	lire une ligne de caractères dans le flux. Une ligne est une suite de caractères qui se termine par un retour chariot '\r' ou un saut de ligne '\n' ou les deux.

The BufferedWriter class

Constructeur	Rôle
BufferedWriter(Writer)	le paramètre fourni doit correspondre au flux dans lequel les données sont écrites.
BufferedWriter(Writer, int)	l'entier en paramètre permet de préciser la taille du buffer. Il doit être positif sinon une exception IllegalArgumentException est levée.

Méthode	Rôle
flush()	vide le tampon en écrivant les données dans le flux.
newLine()	écrire un séparateur de ligne dans le flux

The DatagramSocket class

Constructeur	Rôle
DatagramSocket()	Créer une socket attachée à toutes les adresses IP de la machine et à un des ports libres sur la machine
DatagramSocket(int)	Créer une socket attachée à toutes les adresses IP de la machine et au port précisé en paramètre
DatagramSocket(int, InetAddress)	Créer une socket attachée à l'adresse IP et au port précisés en paramètres

Méthode	Rôle
close()	Fermer la socket et ainsi libérer le port
receive(DatagramPacket)	Recevoir des données
send(DatagramPacket)	Envoyer des données
int getPort()	Renvoyer le port associé à la socket
void setSoTimeout(int)	Préciser un timeout d'attente pour la réception d'un message.

The DatagramPacket class

Constructeur	Rôle
DatagramPacket(byte tampon[], int taille)	Encapsule des paquets en réception dans un tampon
DatagramPacket(byte port[], int taille, InetAddress adresse, int port)	Encapsule des paquets en émission à destination d'une machine

Méthode	Rôle
InetAddress getAddress ()	Renvoyer l'adresse du serveur
byte[] getData()	Renvoyer les données contenues dans le paquet
int getPort ()	Renvoyer le port
int getLength ()	Renvoyer la taille des données contenues dans le paquet
setData(byte[])	Mettre à jour les données contenues dans le paquet

The MulticastSocket class

Builder	Role
MulticastSocket()	Constructs a datagram socket that can perform additional multicast operations.
MulticastSocket(int localPort)	The second form of the constructor specifies the local port. If local port is not specified, the socket is bound to any available local port

Method	Role
void joinGroup(InetAddress groupAddress) void leaveGroup(InetAddress groupAddress)	Join/leave a multicast group. A socket can be a member of several groups simultaneously. Joining a group of which this socket is already a member or leaving a group of which this socket is not a member may generate an exception.
void send(DatagramPacket packet, byte TTL)	Send a datagram from this socket with the lifetime specified for this packet (Time to live (TTL)).
InetAddress getInterface() void setInterface(InetAddress interface) */* interface: Address of one of the host's multicast interfaces */*	Returns/sets the interface used for multicast operations on this socket. This is mainly used on hosts with multiple interfaces. Join/leave requests and datagrams will be sent and datagrams will be received using this interface. The default multicast interface is platform dependent.
int getTimeToLive() void setTimeToLive(int TTL)	Returns/sets the lifetime of all datagrams sent on this socket. This can be overridden on a per datagram basis using the ***send()*** method which takes the TTL as a parameter.