# Sistemas Distribuidos

## Módulo 2

## Comunicación en Sistemas Distribuidos

# Agenda

1. Introducción
    1. Modelos de Comunicaciones.
    2. Tipos de Comunicación.
    3. Paradigmas de Comunicación.
2. Pasaje de Mensajes.
3. Comunicación Directa: mensajes, sockets.
4. Comunicación Remota: request-reply, RPC, RMI.
5. Llamadas a Procedimiento Remoto (RPC): concepto e implementación.
6. Comunicación Indirecta:  Grupo, MOM, Publica-Suscribe.
7. Sockets: concepto e implementación.

# Agenda

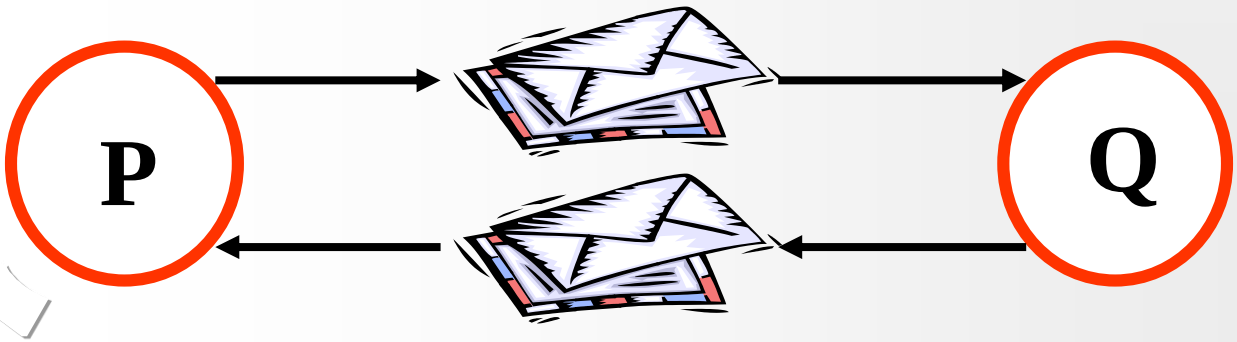
1. Introducción
    1. Modelos de Comunicaciones.
    2. Tipos de Comunicación.
    3. Paradigmas de comunicación.
2. Pasaje de Mensajes.
3. Comunicación Directa: mensajes, sockets.
4. Comunicación Remota: request-reply, RPC, RMI.
5. Llamadas a Procedimiento Remoto (RPC): concepto e implementación.
6. Comunicación Indirecta:  Grupo, MOM, Publica-Suscribe.
7. Sockets: concepto e implementación.

# Comunicación en Sistemas Distribuidos

La comunicación entre procesos necesita compartir información:

a) datos compartidos


b) pasajes de mensajes o copias compartidas



# Comunicación en Sistemas Distribuidos

## Tipos de Comunicación

- ✓Comunicación PERSISTENTE: *almacena* el mensaje (información) enviado por el emisor *el tiempo que tome* entregarlo al receptor.

- ✓Comunicación TRANSITORIA: *almacena* un mensaje *sólo mientras* las aplicaciones del emisor y receptor están en ejecución.

# Comunicación en Sistemas Distribuidos

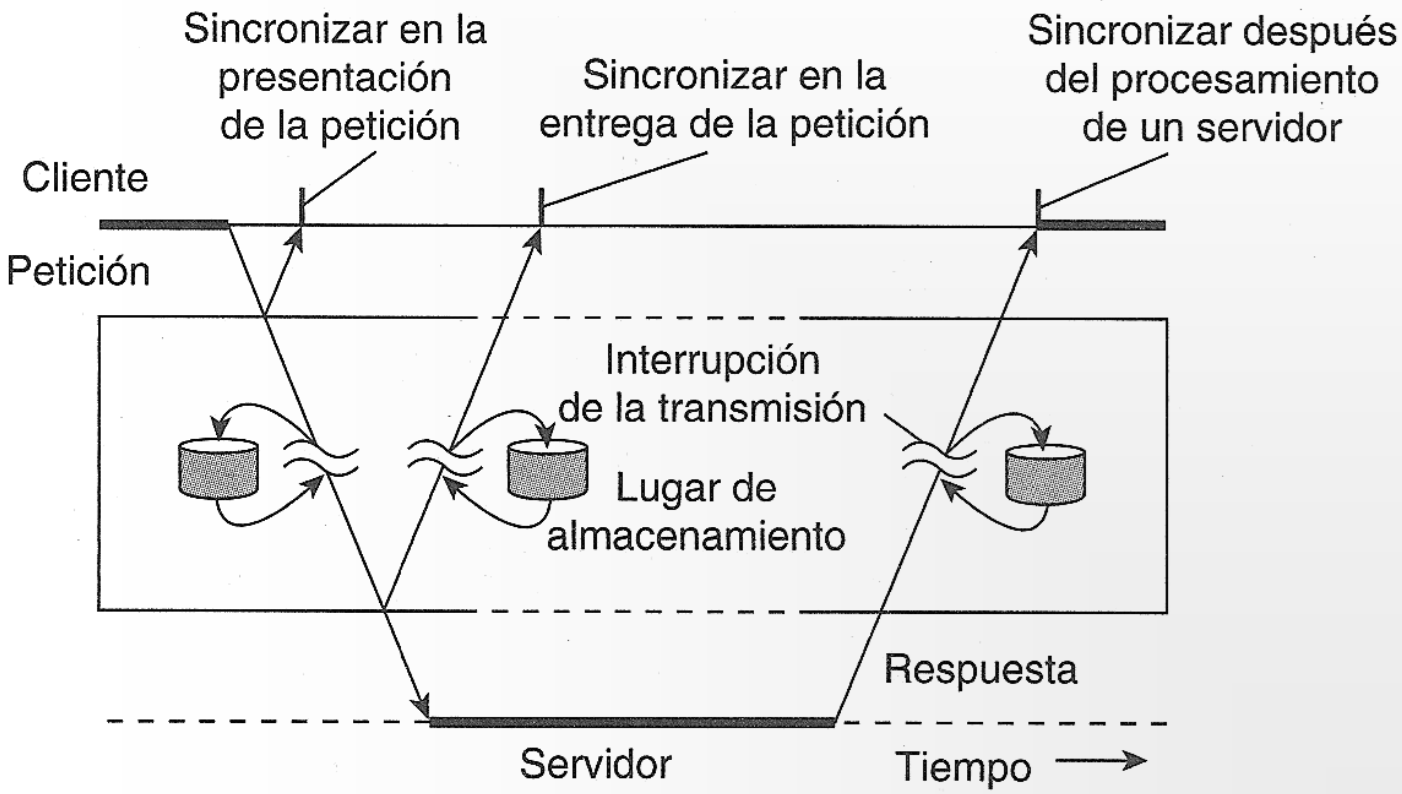
## Tipos de Comunicación

- ✓Comunicación ASINCRÓNICA: el emisor *continúa inmediatamente* después de que ha pasado su mensaje para la transmisión.

- ✓Comunicación SINCRÓNICA: el emisor *es bloqueado* hasta que se sabe que su petición es aceptada.

# Comunicación en Sistemas Distribuidos

## Ejemplo de Comunicación



# Comunicación en Sistemas Distribuidos

## Paradigmas de Comunicación

- ✓Comunicación Directa (entre Procesos)
- ✓Comunicación Remota
- ✓Comunicación Indirecta

# Agenda

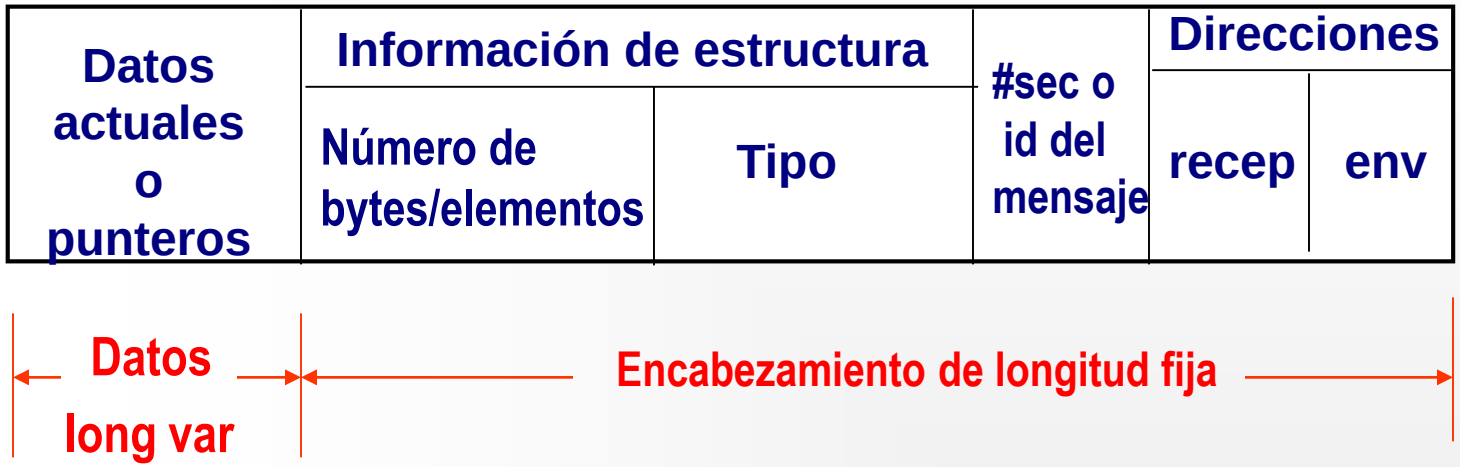
1. Introducción
    1. Modelos de Comunicaciones.
    2. Tipos de Comunicación.
    3. Paradigmas de comunicación.
2. Pasaje de Mensajes.
3. Comunicación Directa: mensajes, sockets.
4. Comunicación Remota: request-reply, RPC, RMI.
5. Llamadas a Procedimiento Remoto (RPC): concepto e implementación.
6. Comunicación Indirecta:  Grupo, MOM, Publica-Suscribe.
7. Sockets: concepto e implementación.

# Pasaje de Mensajes – Característica Deseables

| | |
|---|---|
| **Simplicidad** | Simple y fácil de utilizar. Uso directo |
| **Semántica uniforme** | Comunicaciones locales y remotas |
| **Fiabilidad (Confiabilidad)** | Manejo de las fallas |
| **Eficiencia** | Reducir el número de mensajes intercambiados |
| **Flexibilidad** | Soportar distintos tipos de comunicación |
| **Seguridad** | Autenticación del emisor y receptor. Encriptación de los mensajes |
| **Portabilidad** | Aplicar a nuevos protocolos. Heterogeneidad |
| **Correctitud** | Utilización de multicast. Comunicación de grupos |

# Pasaje de Mensajes – Estructura

Una estructura de mensajes típica:




El *enviador* determina el contenido del mensaje.

El *receptor* tiene en cuenta como interpretar los datos.

# Pasaje de Mensajes

## Sincronización

***No bloqueante***

El receptor conoce la llegada del mensaje

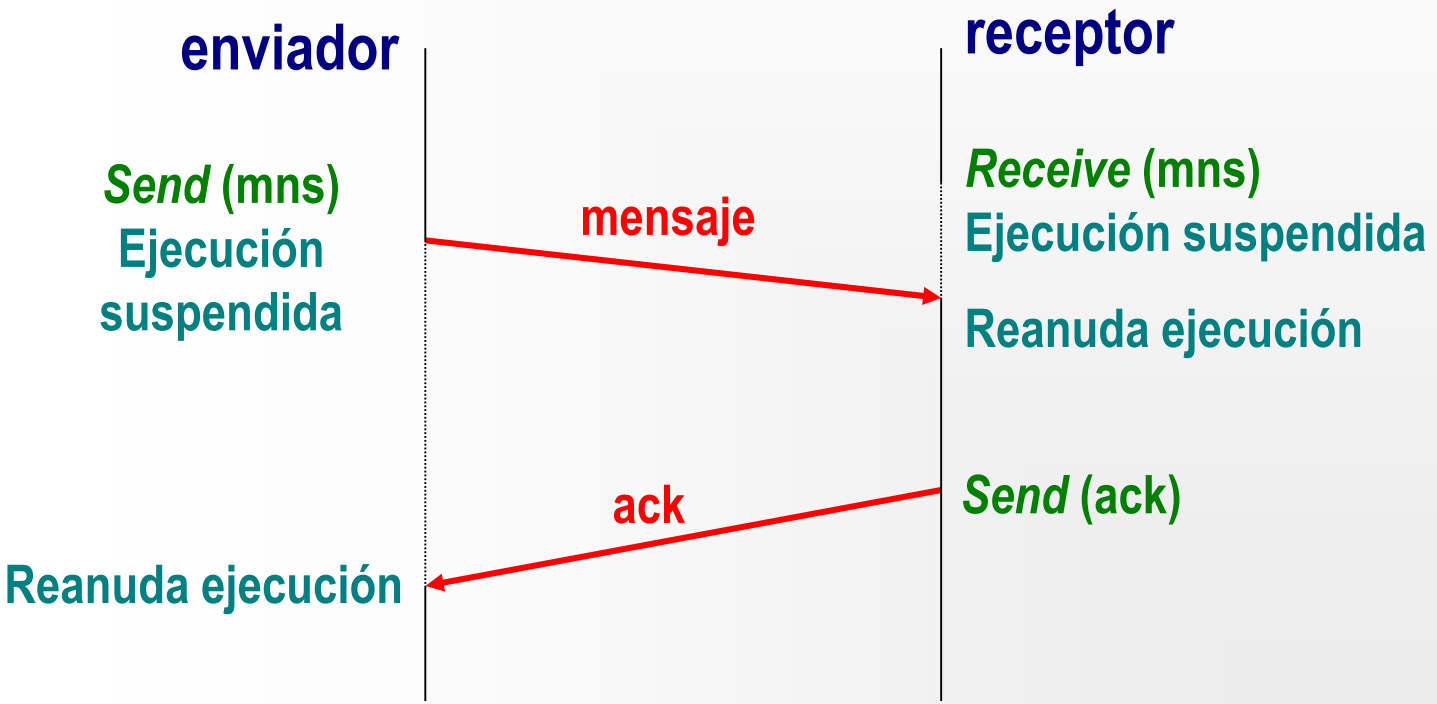
- ✓Polling
- ✓Interrupción

***Bloqueante*** $\Rightarrow$ *sincrónica*

Fácil de implementar pero poca concurrencia

# Pasaje de Mensajes

## Comunicación Sincrónica - Mensajes Bloqueantes



# Pasaje de Mensajes

## Buffering

De *buffer nulo* a *buffer* con capacidad ilimitada

**No *buffer***

- Cita (rendez-vous)
- Descarte

***Buffer* simple**

Adecuado para transferencia sincrónica

***Capacidad infinita***

Almacena todo lo que recibe (asincrónica)

# Pasaje de Mensajes

## Buffering

### *Buffer* límite finito

Puede haber rebalse de *buffer*

- ✓Comunicación no exitosa (lo hace menos confiable)
- ✓Comunicación con flujo controlado (bloquea al enviador hasta que haya espacio)

### *Buffer* múltiple

*Mailbox* o pórtico

# Pasaje de Mensajes

## Mensajes multidatagrama

La mayoría tiene un límite superior en el tamaño del dato que puede ser transmitido en algún momento (MTU).

Esto implica que magnitudes mas grandes deben fragmentarse en paquetes.

El ensamblador y desensamblador es responsabilidad del sistema de pasaje de mensajes.

# Pasaje de Mensajes

## Codificación y decodificación de mensajes de datos

Un puntero absoluto pierde significado cuando es transmitido de un espacio a otro.

Diferentes programas objeto ocupan una cantidad de espacio variada.

Métodos de Intercambio

- Formato general acordado
- Formato emisor

# Pasaje de Mensajes

## Codificación y decodificación de mensajes de datos

XDR (External Data Representation)

- ✓Proceso empaquetado (marshalling)
- ✓Proceso desempaquetado (unmarshalling)

Se usan, en general, dos representaciones:

- Representación etiquetada (MACH, XML)
- Representación no etiquetada (SUN XDR, CORBA CDR)

# Pasaje de Mensajes

## Direccionamiento de los procesos

| Direccionamiento Explícito | Direccionamiento Implícito |
|---|---|
| Problema de nombres de las partes involucradas en una interacción | No se explicita el nombre sino se menciona un servicio |
| Send (process-id,msg) | Send-any (service-id,msg) |
| Receive(process-id,msg) | Receive-any (proceso-mudo,msg) |

# Pasaje de Mensajes

## Manejo de fallas

- Caída de sitio
- Caída de enlace

Problemas posibles:

a) Pérdida del mensaje de requerimiento
b) Pérdida del mensaje de respuesta
c) Ejecución del requerimiento no exitosa

**Protocolos de mensajes confiables**

**Cuatro mensajes**
**Tres mensajes**
**Dos mensajes**

# AGENDA

1. Introducción
    1. Modelos de Comunicaciones.
    2. Tipos de Comunicación.
    3. Paradigmas de comunicación.
2. Pasaje de Mensajes.
3. Comunicación Directa: mensajes, sockets.
4. Comunicación Remota: request-reply, RPC, RMI.
5. Llamadas a Procedimiento Remoto (RPC): concepto e implementación.
6. Comunicación Indirecta:  Grupo, MOM, Publica-Suscribe.
7. Sockets: concepto e implementación.

# Comunicación Directa

- La forma más simple de comunicación entre procesos es a través del pasaje de mensajes.
- El proceso emisor debe especificar el destino del proceso receptor (puede ser con una dirección y un puerto).
- Los mensajes pueden ser:
  - Datagrama
  - Stream
  - Multicast

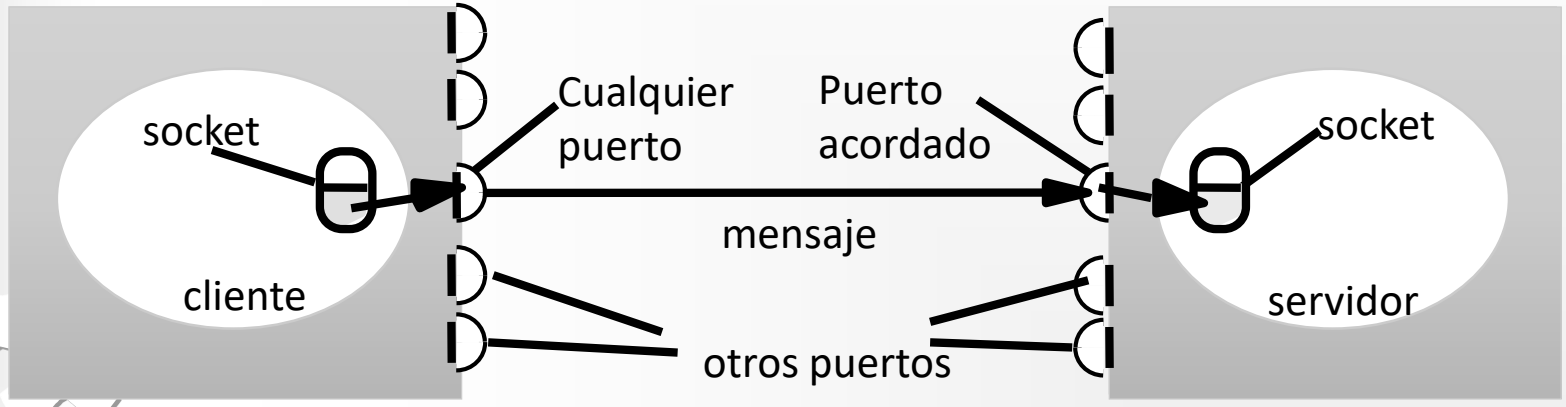
# Comunicación Transitoria y entre Procesos

## Sockets

- Es una interfaz de entrada-salida de datos que permite la intercomunicación entre procesos.
- Es un punto final (endpoint) en la comunicación, el cual una aplicación puede escribir datos que serán enviados por la red y desde el cual ingresará los datos que puede leer.

# Comunicación Transitoria y Entre Procesos

## - Sockets




Dirección Internet = 138.37.94.248

Dirección Internet = 138.37.88.249

# Agenda

1. Introducción
    1. Modelos de Comunicaciones.
    2. Tipos de Comunicación.
    3. Paradigmas de comunicación.
2. Pasaje de Mensajes.
3. Comunicación Directa: mensajes, sockets.
4. Comunicación Remota: request-reply, RPC, RMI.
5. Llamadas a Procedimiento Remoto (RPC): concepto e implementación.
6. Comunicación Indirecta:  Grupo, MOM, Publica-Suscribe.
7. Sockets: concepto e implementación.

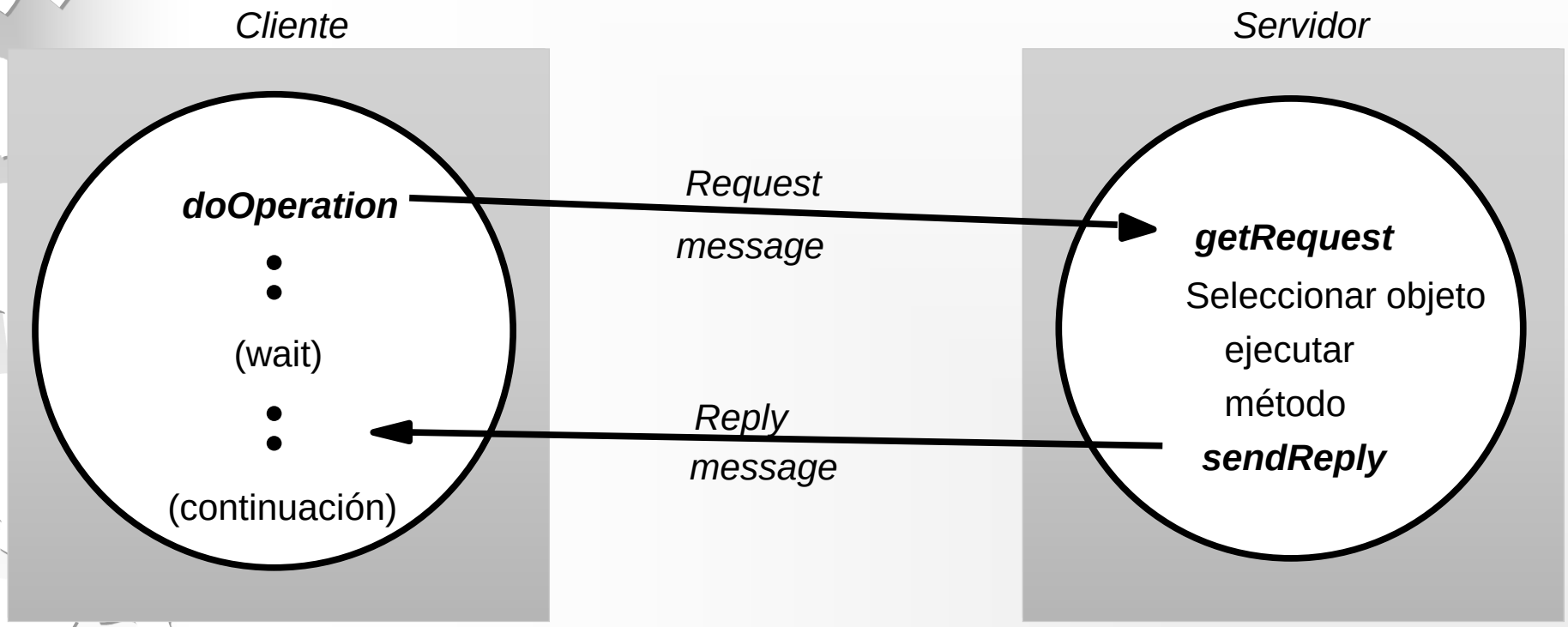
# Comunicación Remota

## Request-Reply

- Soporta el intercambio bidireccional de mensajes.
- Se adapta a los requerimientos del modelo cliente-servidor.
- Es un protocolo de bajo nivel.
- Típicamente la comunicación es de tipo sincrónica.
- Confiable.

# Comunicación Remota

## Request-Reply



# Agenda

1. Introducción
    1. Modelos de Comunicaciones.
    2. Tipos de Comunicación.
    3. Paradigmas de comunicación.
2. Pasaje de Mensajes.
3. Comunicación Directa: mensajes, sockets.
4. Comunicación Remota: request-reply, RPC, RMI.
5. Llamadas a Procedimiento Remoto (RPC): concepto e implementación.
6. Comunicación Indirecta:  Grupo, MOM, Publica-Suscribe.
7. Sockets: concepto e implementación.

# Llamadas a Procedimiento Remoto (RPC)

## El modelo RPC

Es similar al bien conocido y entendido modelo de llamadas a procedimientos usado para transferir control y datos.

El mecanismo de RPC es una extensión del anterior porque habilita a hacer una llamada a un procedimiento que no reside en el mismo espacio de direcciones.

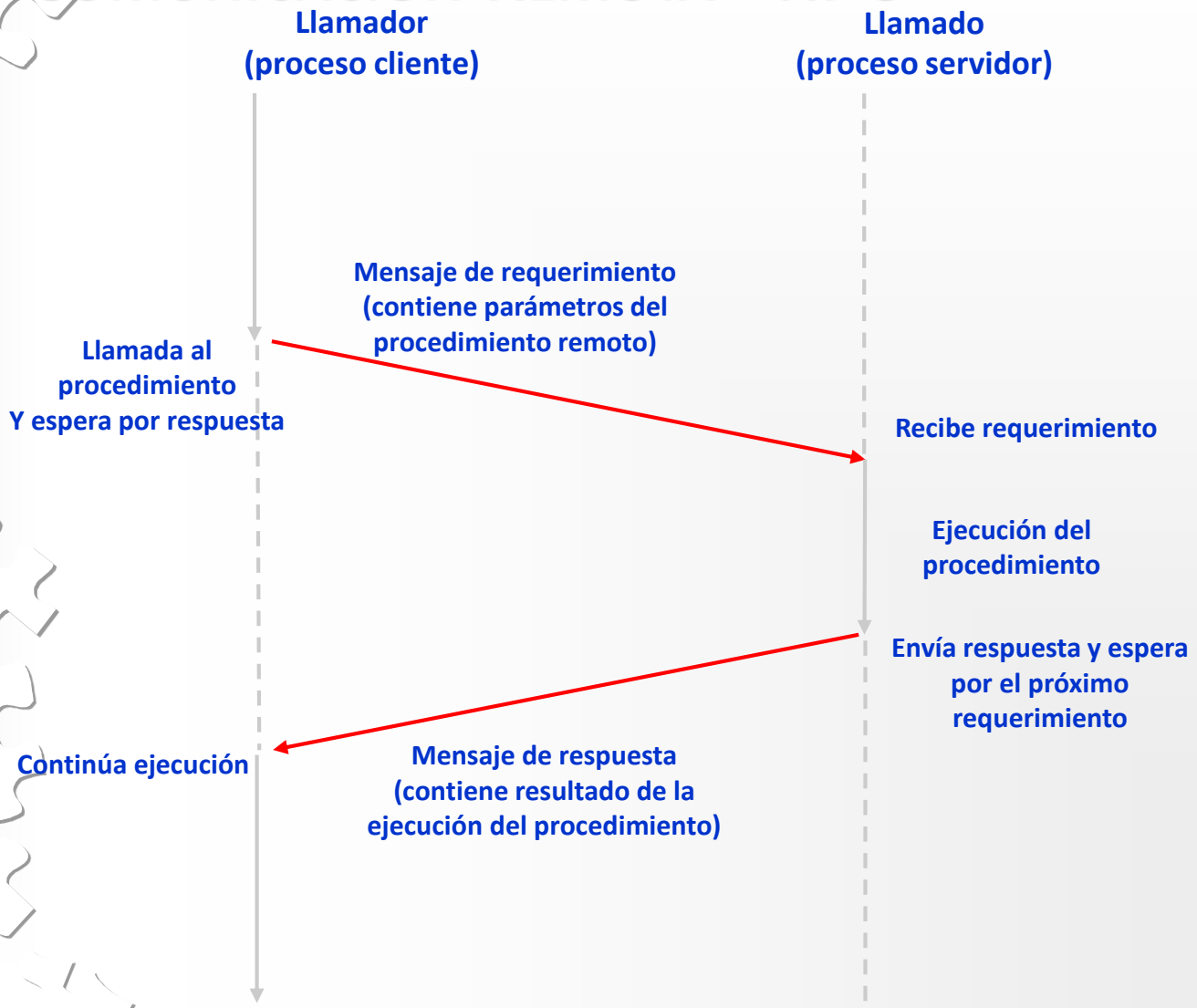
# Comunicación Remota - RPC

La facilidad de RPC usa un esquema de pasaje de mensajes para intercambiar información entre los procesos ***llamador*** (proceso cliente) y ***llamado*** (proceso servidor).

Normalmente el proceso servidor *duerme*, esperando la llegada de un mensaje de requerimiento.

El proceso cliente se bloquea cuando envía el mensaje de requerimiento hasta recibir la respuesta.

# Comunicación Remota - RPC



# Comunicación Remota - RPC

## Transparencia de RPC

**Transparencia SINTÁCTICA**: una llamada a procedimiento remoto debe tener la misma sintaxis que una llamada local.

**Transparencia SEMÁNTICA**: la semántica de un RPC es la misma que para una llamada local.

Diferencias:

a) espacio de direcciones

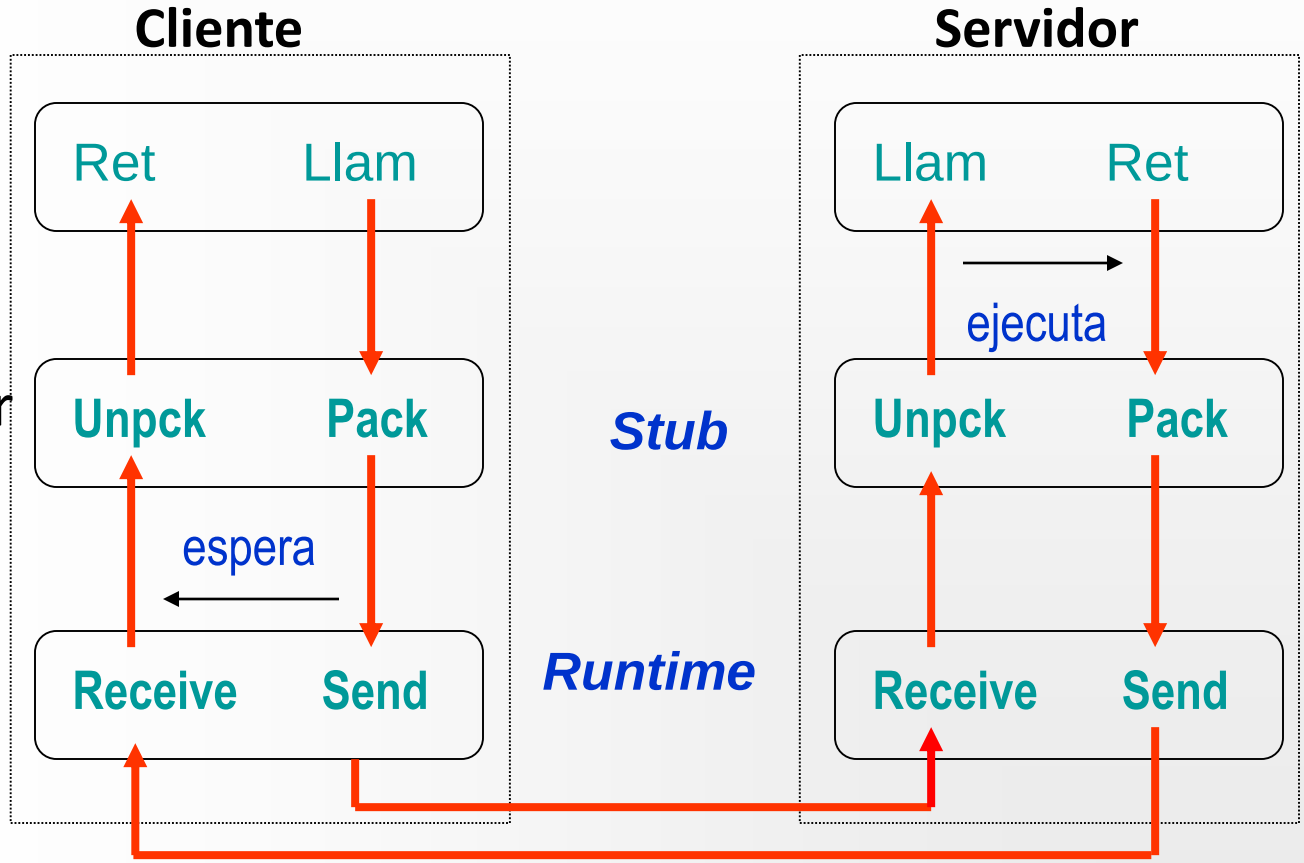
b) fallas

c) Consumen más tiempo

# Comunicación Remota - RPC

## Implementación del mecanismo de RPC

- El cliente
- El *stub* cliente
- El runtime RPC
- El *stub* servidor
- El servidor



# Comunicación Remota - RPC

## Cliente

Es el que inicia el RPC. Hace una llamada que invoca al *stub*.

## Stub cliente

Realiza las siguientes tareas:

a) Empaqueta la especificación del procedimiento objetivo y sus argumentos en un mensaje y pide al *runtime* local que lo envíe al ***stub*** servidor

b) En la recepción de los resultados de la ejecución del proceso, desempaqueta los mismos y los pasa al cliente.

# Comunicación Remota - RPC

## Runtime RPC

Maneja la transmisión de mensajes a través de la red entre las máquinas cliente y servidor.

## Stub servidor

Trabaja en forma simétrica a como lo hace el *stub* cliente.

## Servidor

Cuando recibe un requerimiento de llamada del *stub* servidor, ejecuta el procedimiento apropiado y retorna el resultado de la misma al *stub* servidor.

# Comunicación Remota - RPC

## Mensajes de RPC

### Mensaje de llamada

| Mensaje ID | Mensaje tipo | Cliente Id | Procedimiento Remoto Id | | | Argumentos |
|---|---|---|---|---|---|---|
| | | | Prog. Nro. | Ver. Nro. | Proc.Nro. | |

### Mensaje de respuesta

| Mensaje Id | Mensaje tipo | Respuesta Estado (éxito) | Resultado |
|---|---|---|---|

| Mensaje Id | Mensaje tipo | Respuesta Estado (no exitoso) | Razón de la falla |
|---|---|---|---|

# Comunicación Remota - RPC

## Administración del Servidor

1.- Implementación

- Servidor con estado: mantiene información de un cliente de una llamada a procedimiento remoto a la próxima llamada.
- Servidor sin estado: no mantiene información de estado de un cliente.

¿cuál es mejor?

# Comunicación Remota - RPC

## Administración del Servidor

2.- Semántica de creación

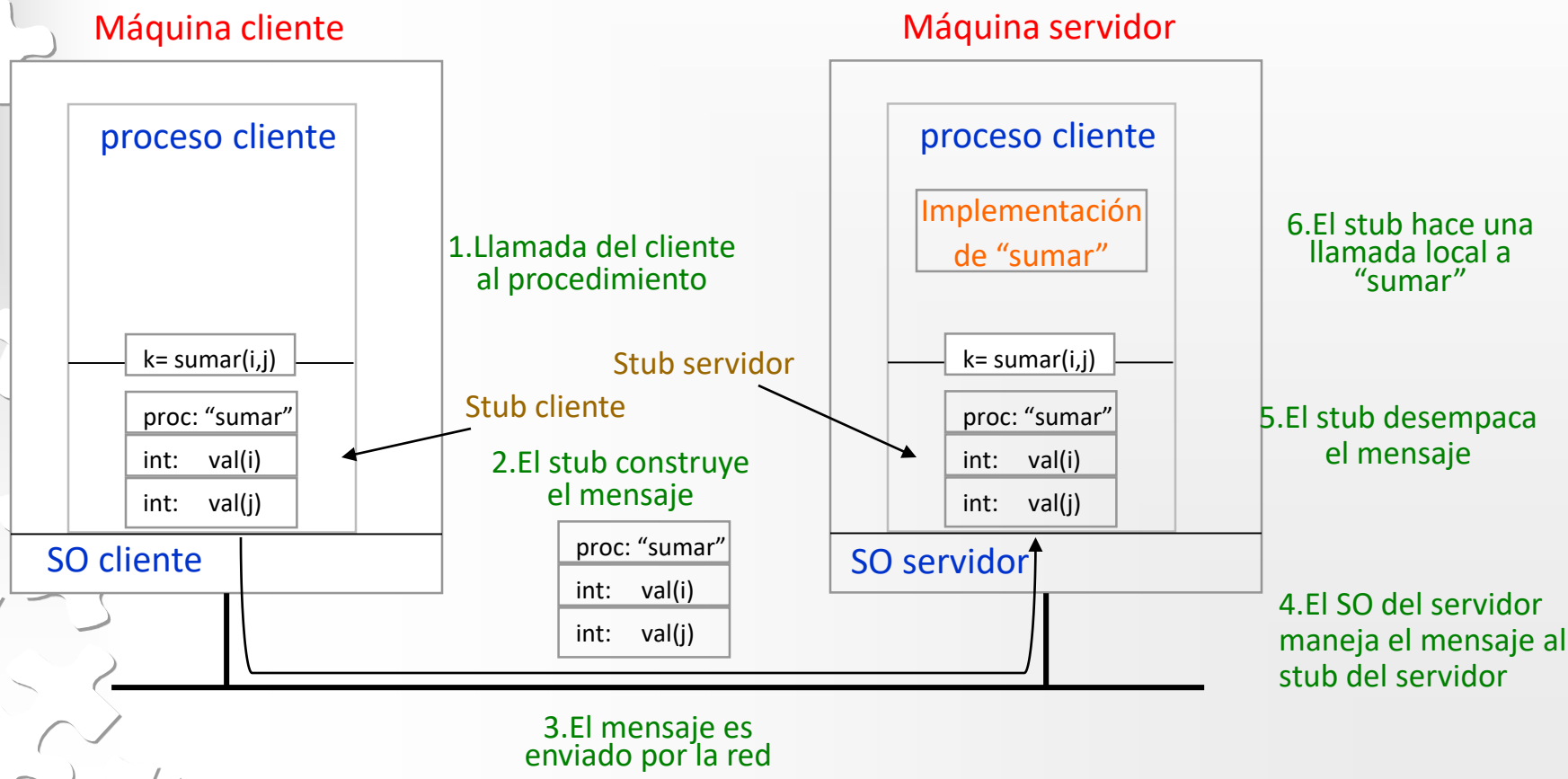
- Instancia por llamada al servidor
- Instancia por sesión con el servidor
- Persistente

3.- Pasaje de parámetros

- por valor
- por referencia

# Comunicación Remota - RPC

## Pasaje de Parámetros por Valor



# Comunicación Remota - RPC

## Semántica de la llamada

1.- Pudiera ser (may be)

2.- Al menos una vez (at-least-once)

3.- A lo sumo una vez (at-most-once)

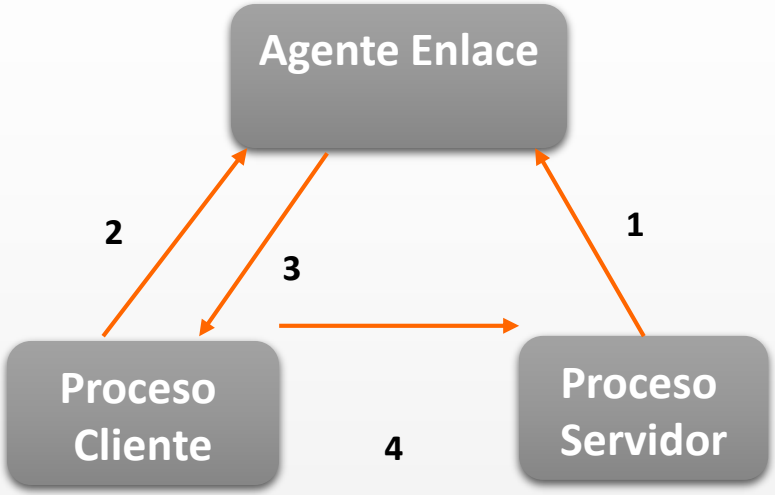
4.- Exactamente una vez

# Enlace de un Cliente y un Servidor en RPC

1.- Nombre Servidor (Servicio)

2.- Ubicación Servidor

- Broadcasting
- Agente de enlace (binding)
  - Ubicación conocida
  - Ubicación desconocida



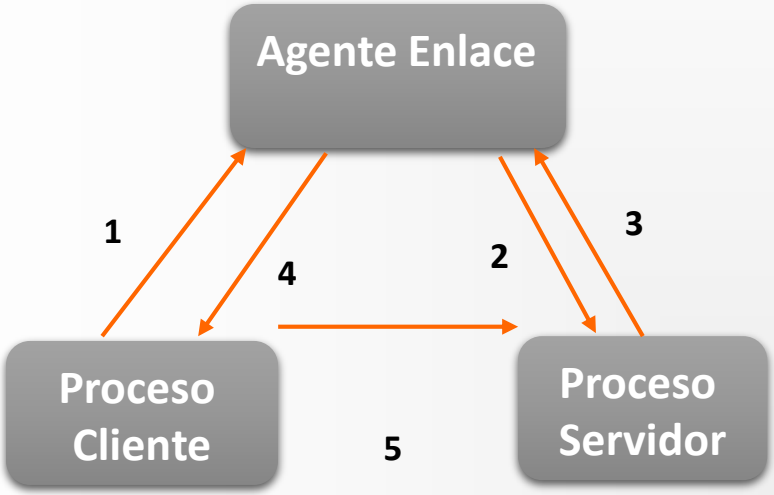

3.- Tiempo de enlace

- compilación.
- link (requiere un manejador o canal para conectarse el cliente con el servidor).
- llamada (ocurre en la primer llamada)

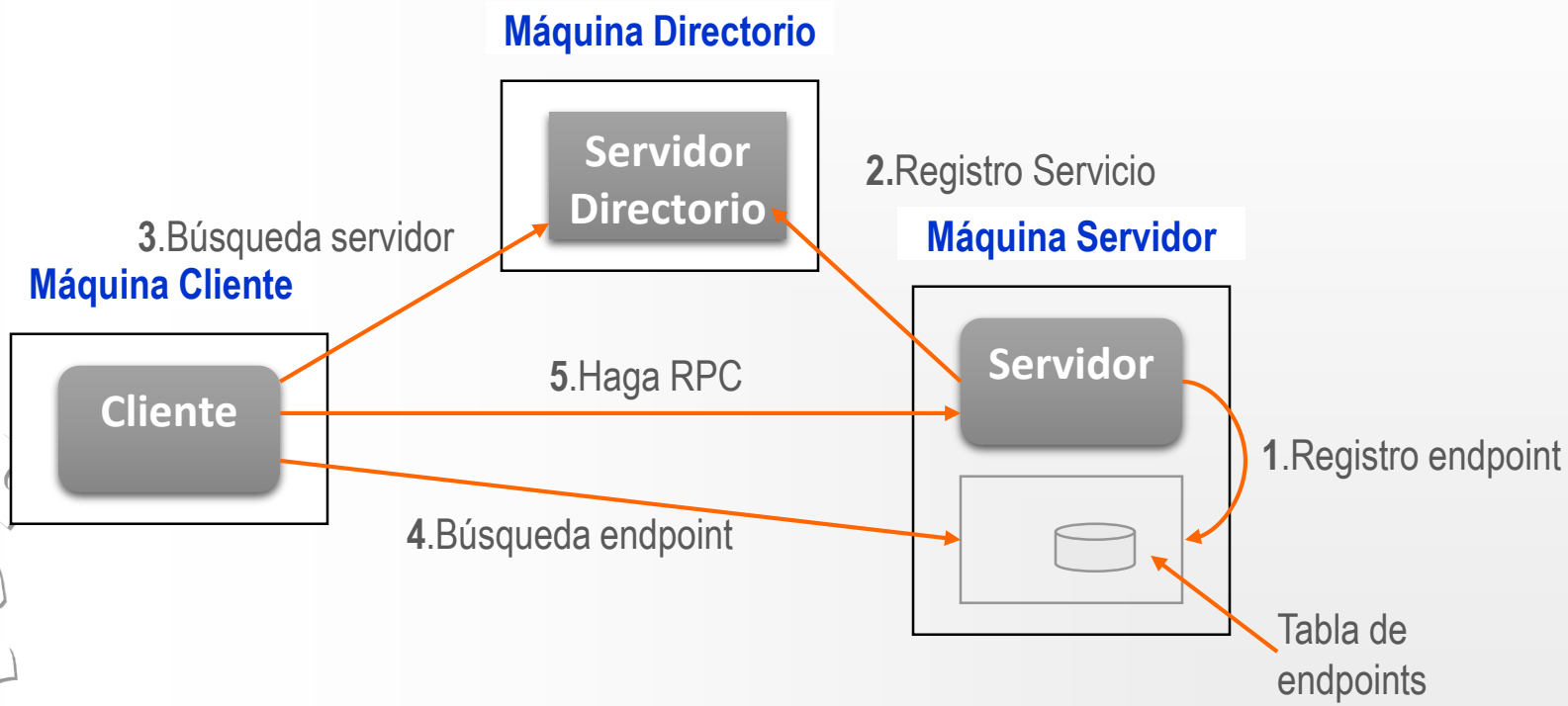
# Enlace de un Cliente y un Servidor en RPC

Vinculación en el tiempo de una llamada.

Utilización del método de llamada indirecta



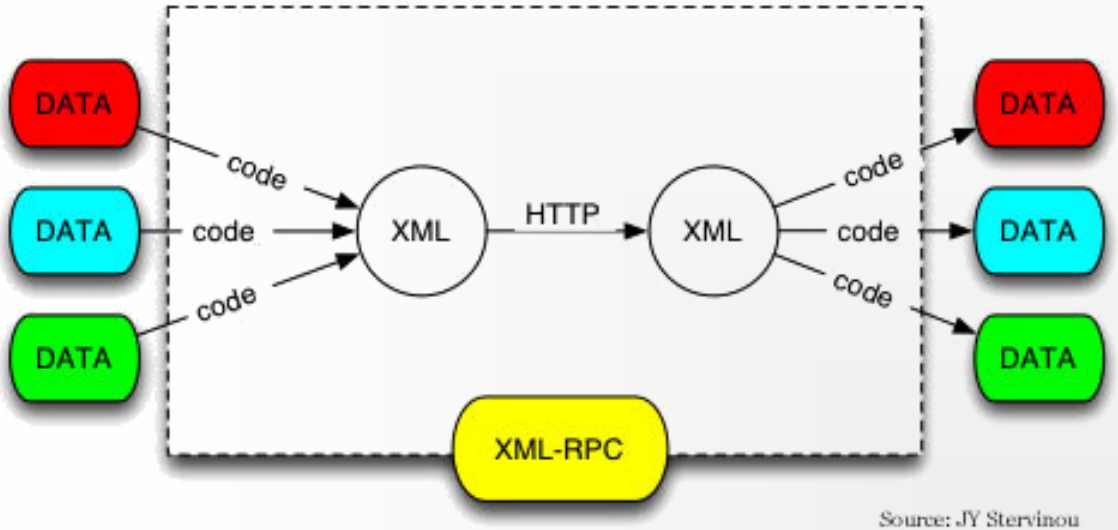
# Enlace de un Cliente y un Servidor en RPC



# Comunicación Remota - RPC

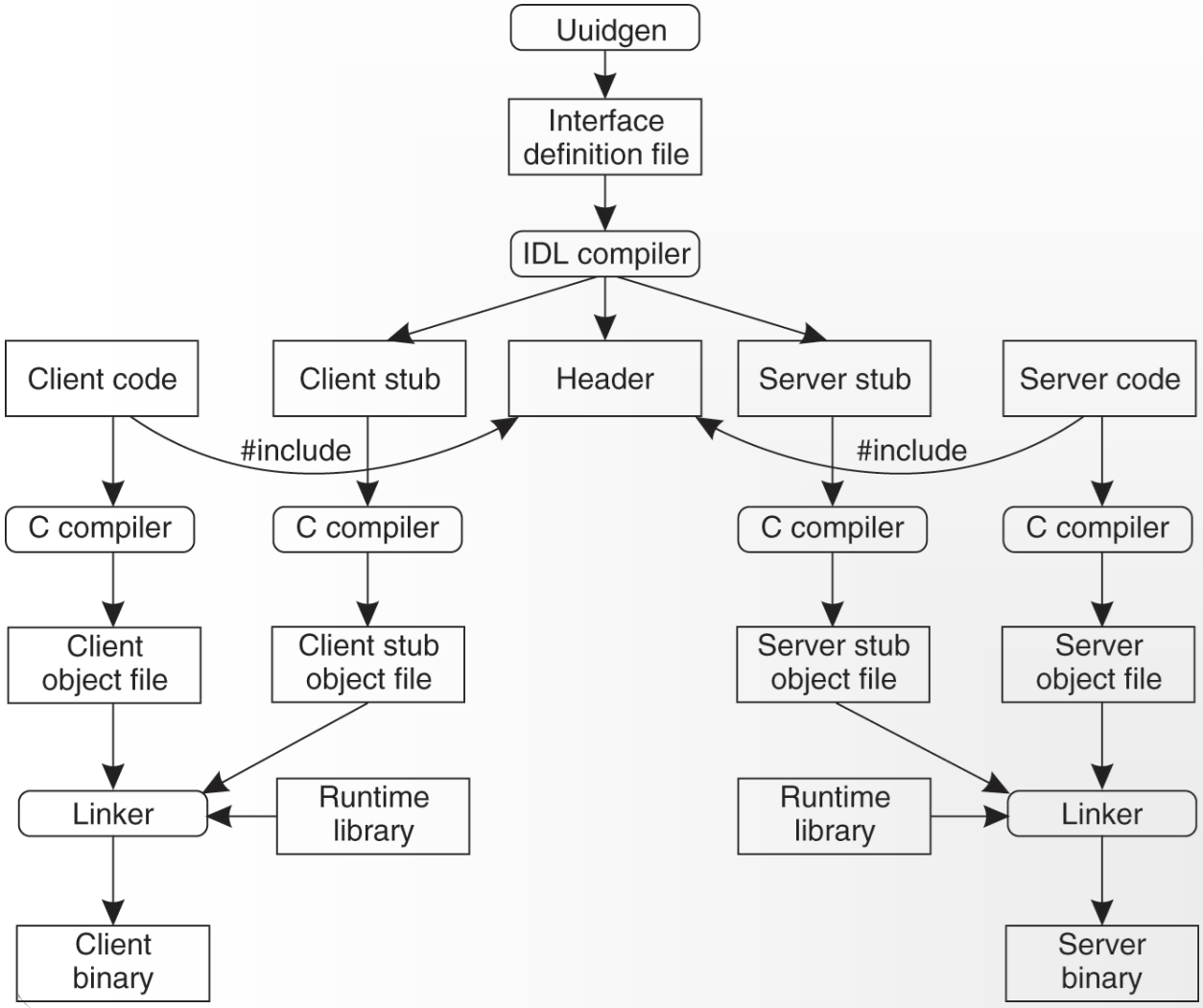
- ONC RPC – Sun RPC
- DCE RPC
- XML-RPC

# Comunicación Remota - RPC




Source: JY Stervinou

# Comunicación Remota - RPC

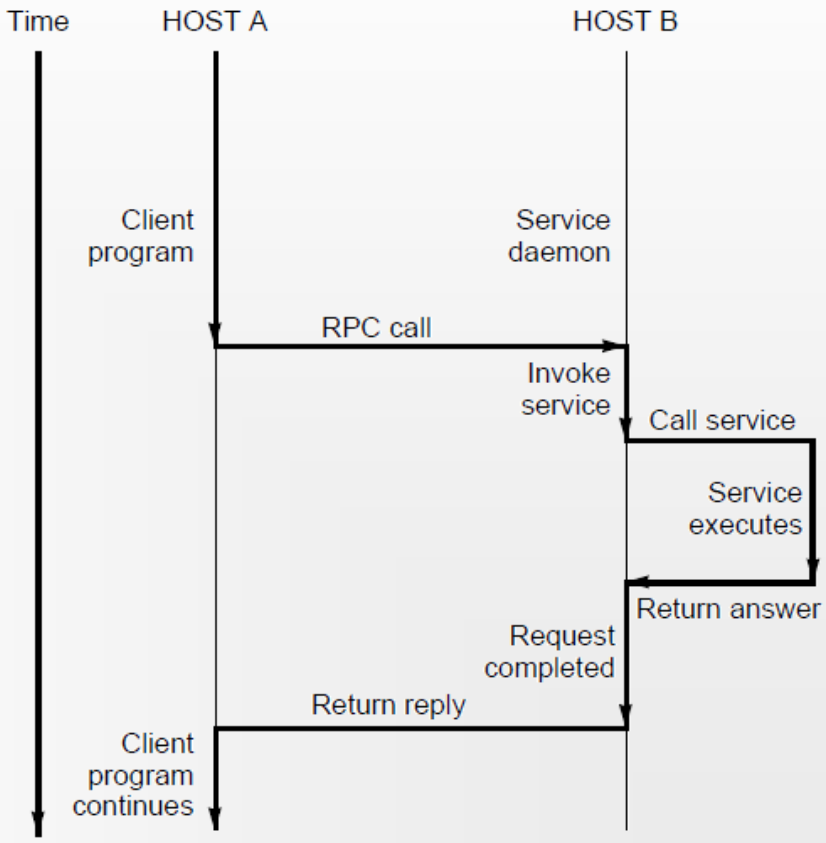


# Sun RPC - Implementación

# Comunicación Remota - Sun RPC

El RPC representa una llamada a una función.

Mantiene la *semántica* de una llamada local, lo que difiere es la *sintaxis*.



# Comunicación Remota - Sun RPC

## Pasos para desarrollar una aplicación RPC

- **Especificar el protocolo de comunicación del cliente y servidor.**
- **Desarrollar el programa del cliente**
- **Desarrollar el programa del servidor**

Los programas serán compilados en forma separada. El protocolo de comunicación es armado mediante *stubs* y estos stubs y rpc (y otras librerías) necesitarán ser linkeadas

# Comunicación Remota - Sun RPC

## 1.- Definir el protocolo

- utilizar un protocolo compiler como ***rpcgen***
- para el protocolo se debe especificar:
  - Nombre del procedimiento del servicio
  - Tipo de dato de los parámetros de E/S

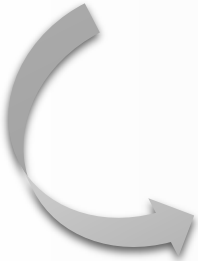
El protocolo compiler lee una definición y automáticamente genera los *stubs* del *cliente* y *servidor*

# Comunicación Remota - Sun RPC

- ***rpcgen***: utiliza su propio lenguaje (RPC lenguaje o RPCL) el cual es similar a las directivas preprocesador.
- ***rpcgen*** existe como un compiler que lee archivos especiales que tienen extensión .x

Para compilar un RPCL:

**rpcgen rpcprog.x**



Genera cuatro posibles archivos

# Comunicación Remota - Sun RPC

- **rpcprog_clnt.c** → el stub del cliente
- **rpcprog_svc.c** → el stub del servidor
- **rpcprog_xdr.c** → si es necesarioXDR (external data representation) filters
- **rpcprog.h** → el encabezado del archivo necesitado por cualquier XDR filters

# Comunicación Remota - Sun RPC

## 2.- Definir el código de Aplicación del Cliente y Servidor

- El código del cliente se comunica vía procedimientos y tipos de datos especificados en el protocolo.
- El servidor tiene que ***registrar*** los procedimientos que van a ser invocados por el cliente y recibir y devolver cualquier dato requerido para el procesamiento.

**El cliente y el servidor deben incluir el archivo *"rpcprog.h"***

# Comunicación Remota - Sun RPC

## 3.- Compilar y Ejecutar la Aplicación

Compilar el código del Cliente:

**gcc –c rpcprog.c**

Compilar el stub del Cliente

**gcc –c rpcprog_clnt.c**

Compilar el XDR filtro

**gcc –c rpcprog_xdr.c**

Construir el ejecutable del Cliente

**gcc –lnsl –o rpcprog rpcprog.o rpcprog_clnt.o rpcprog_xdr.o**

# Comunicación Remota - Sun RPC

## 3.- Compilar y Ejecutar la Aplicación – Cont.

Compilar los procedimientos del Servidor:

**gcc –c rpcsvc.c**

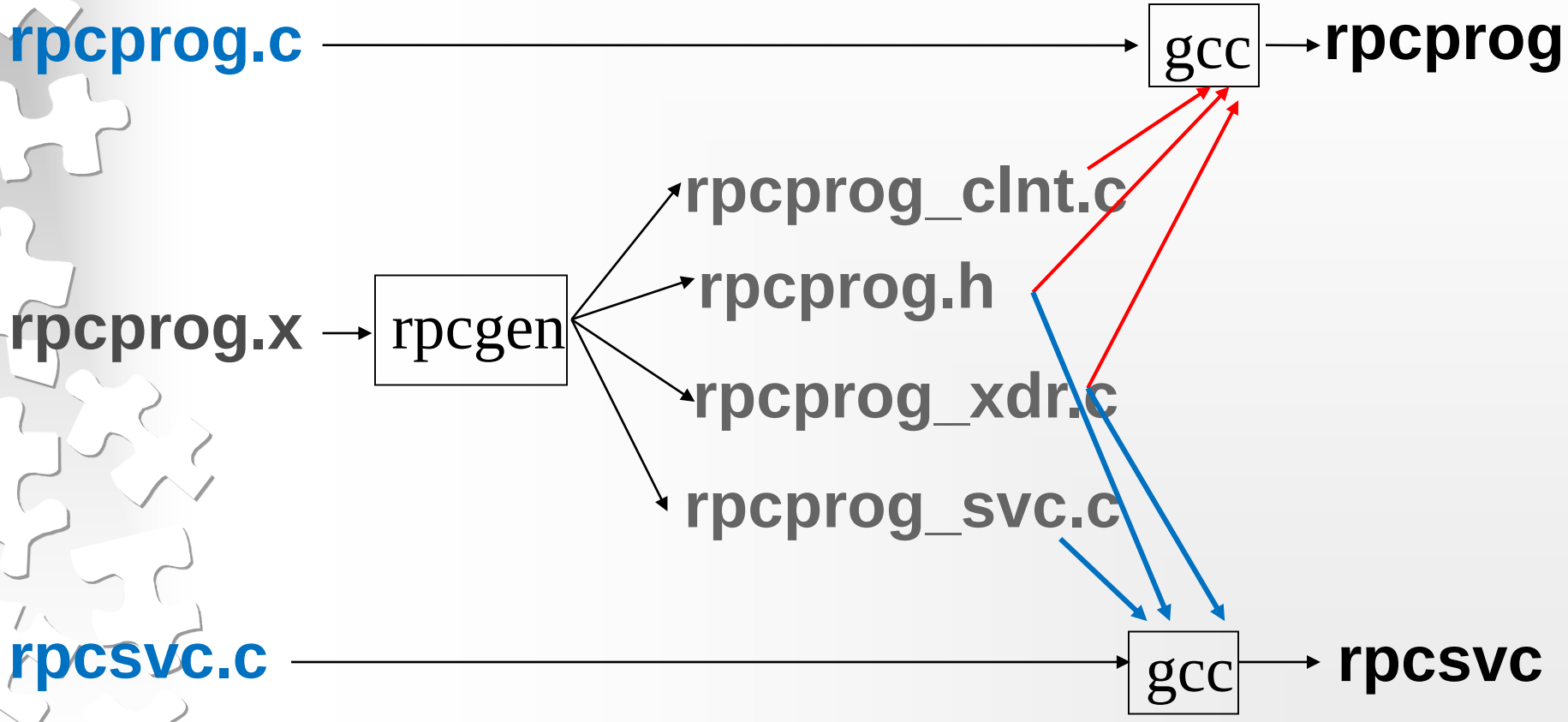
Compilar el stub del Servidor:

**gcc –c rpcprog_svc.c**

Construir el ejecutable del Servidor:

**gcc –lnsl –o rpcsvc rpcsvc.o rpcprog_svc.o rpcprog_xdr.o**

# Comunicación Remota - Sun RPC



# COMUNICACIÓN REMOTA - Sun RPC

## Formato General Protocolo

/* **proto.x** protocolo para … */

***Definición de constantes***

…

***Definición de tipos***

…

program **NOMBREPROGRAMA**{

version **VERSIONPROGRAMA**{

tiporesultadofunción1 **FUNCION1**(tipoparámetro1) =1;

tiporesultadofunción2 **FUNCION2**(tipoparámetro2) =2;

tiporesultadofunción3 **FUNCION3**(tipoparámetro3) =3;

…

} = 1;

} = 0x20000121;

```
#define TAM_BLOQUE 512

Struct  atributos
{
    char nombre[512];
    int   modo;
};
```

# Sun RPC – Ejemplo: Mensaje Remoto

**Definición del protocolo**

```
/* msg.x remote msg printing protocol */

program MESSAGEPROG{
    version PRINTMESSAGEVERS{
            int PRINTMESSAGE(string) =1;
    } = 1;
} = 0x20000001;
```

***Identificación del programa***

***Identificación de la versión***

***Identificación del procedimiento***

Los números de programa están definidos en forma standard:

- 0*x*00000000 - 0*x*1*FFFFFFF*: Defined by Sun
- 0*x*20000000 - 0*x*3*FFFFFFF*: User Defined
- 0*x*40000000 - 0*x*5*FFFFFFF*: Transient
- 0*x*60000000 - 0*xFFFFFFFF*: Reserved

# Sun RPC – Ejemplo: Mensaje Remoto

## Definición del Servicio

```
/* msg_proc.c: implementación del procedimiento remoto */
 #include <stdio.h>
#include "msg.h" /* msg.h generado por el rpcgen */
 int *printmessage_1(msg,req)
   char **msg;
   struct svc_req *req; /*detalle de la llamada */
 {
   static int  result; /*debe ser estático*/
   FILE  *f;
   f = fopen("/dev/console" ,"w");
   if (f == (FILE *)NULL) {
                result = 0;
                return(&result);
   }
   fprintf(f, "%s \n", *msg);
   fclose(f);
   result = 1;    return(&result); }
```

*Definición de datos*
*Y funciones*

*Función o procedimiento remoto*

En linux se denominará
***printmessage_1_svc**

# Sun RPC – Ejemplo: Mensaje Remoto

## Definición del Cliente

```
/* rprintmsg.c: versión remota de "printmsg.c" */
 #include <stdio.h>
#include "msg.h"  /* msg.h generado por rpcgen */
main (argc, argv)
   int argc;
   char *argv[];
{
   CLIENT  *clnt;
   int   *result;
   char  *server;
   char  *message;
   if (argc != 3) {
                fprintf(stderr, "usage: %s host message \n", argv[0]);
                exit(1);
   }
   server = argv[1];
   message = argv[2];
```

**Variable utilizada para Establecer la comunicación**

**Nombre de la máquina**

# Sun RPC – Ejemplo: Mensaje Remoto

```
....
/* Create client "handle" used for calling MESSAGEPROG on the server */
   /* designated on the command line */
     clnt = clnt_create(server, MESSAGEPROG, PRINTMESSAGEVERS, "visible");
    if (clnt == (CLIENT *) NULL) {
                /* Couldn't establish connection with server. Print error message */
                /* and die */
                clnt_pcreateerror(server);
                exit(1);
   }
   /* Call the remote procedure "printmessage" on the server */
   result = printmessage_1(&message, clnt);
   if (result == (int *) NULL) {
                /* An error occurred while calling the server. Print error message */
                /* and die */
                clnt_perror (clnt, server);
                exit(1);
   }
```

**Establecer conexión Con el servidor**

**Para seleccionar el protocolo de comunicación. TCP / UDP**

**Invocación al procedimiento remoto**

# Sun RPC – Ejemplo: Directorio Remoto

## Definición del Protocolo

```
/* dir.x: protocolo para listar un directorio remoto. Este ejemplo muestra las funciones de rpcgen*/
const MAXLENNAME = 255; /* max longitud de la entrada del directorio */
typedef string nametype <MAXLENNAME>; /* directory entry */
typedef struct namenode *namelist; /* link in the listing */
struct namenode {
   nametype name; /* name of the directory entry */
   namelist next; /* next entry */
};
union readdir_res switch (int errno) {
   case 0:
                namelist list; /* no error: return directory listing */
   default:
                void; /* error ocurred: nothing else to return */
};
 program DIRPROG{
   version DIRVERS{
                readdir_res READDIR(nametype) = 1;
   } = 1;
} = 0x20000076;
```

# SUN RPC – EJEMPLO: DIRECTORIO REMOTO

**Dir.h**
**(contenido generado Por rpcgen en Solaris)**

```
/* Please do not edit this file.  It was generated using rpcgen. */
#ifndef _DIR_H_RPCGEN
#define     _DIR_H_RPCGEN
#include <rpc/rpc.h>
#define     MAXLENNAME 255
typedef char *nametype;
typedef struct namenode *namelist;
struct namenode {
        nametype name;
        namelist next;
};
typedef struct namenode namenode;
struct readdir_res {
        int errno;
        union {
                namelist list;
        } readdir_res_u;
};
```

```
#define     DIRPROG ((unsigned long)(0x20000076))
#define     DIRVERS ((unsigned long)(1))
#define     READDIR ((unsigned long)(1))
extern  readdir_res * readdir_1();
extern int dirprog_1_freeresult();

/* the xdr functions */
extern bool_t xdr_nametype();
extern bool_t xdr_namelist();
extern bool_t xdr_namenode();
extern bool_t xdr_readdir_res();
#endif /* !_DIR_H_RPCGEN */
```

# Sun RPC – Ejemplo: Directorio Remoto

## Definición del Cliente

```
#include <stdio.h>
#include "dir.h"  /* generado por rpcgen */
#include <rpc/rpc.h>
extern int errno;
main(argc, argv)
   int argc;
   char *argv[];
{
   CLIENT  *clnt;
   char   *server;
   char   *dir;
   readdir_res   *result;
   namelist  nl;
      if (argc != 3) {
                fprintf(stderr, " %s host dir \n", argv[0]);
                exit(1);
   }
   server = argv[1];
   dir = argv[2];
```

Inclusión de la librería para RPC

Protocolo de comunicación

```
 clnt = clnt_create(server, DIRPROG, DIRVERS, "visible");
if (clnt == (CLIENT *) NULL) {
                clnt_pcreateerror(server);
                exit(1);
   }
   result = readdir_1(&dir, clnt);
   if (result == (readdir_res *) NULL) {
                clnt_perror(clnt, server);
                exit(1);
   }
```

Invocación al Procedimiento remoto

# Sun RPC – Ejemplo: Directorio Remoto

**Definición del Servicio**

```
/* dir_proc.c: remote readdir implementation */
#include <dirent.h>
#include "dir.h" /* creado por rpcgen*/
#include <errno.h>
extern int errno;
extern char *malloc();
extern char *strdup();
readdir_res *readdir_1(dirname, req)
   nametype       *dirname;
   struct svc_req  *req;
{
   DIR  *dirp;
   struct dirent  *d;
   namelist  nl;
   namelist  *nlp;
   static readdir_res  res; /* debe ser estático */
   dirp = opendir(*dirname);
   ...
```

En linux se denominará

***readdir_1_svc**

# SUN RPC – EJEMPLO: DIRECTORIO REMOTO

**STUB DEL CLIENTE**. Dir_clnt.c

```
#include "dir.h"
#include <stdio.h>
#include <stdlib.h> /* getenv, exit */
static struct timeval TIMEOUT = { 25, 0 };
readdir_res *
readdir_1(argp, clnt)
        nametype *argp;
        CLIENT *clnt;
{
        static readdir_res clnt_res;
        memset((char *)&clnt_res, 0, sizeof (clnt_res));
        if (clnt_call(clnt, READDIR, (xdrproc_t) xdr_nametype,
                (caddr_t) argp,(xdrproc_t) xdr_readdir_res,
                (caddr_t) &clnt_res,    TIMEOUT) != RPC_SUCCESS) {
                return (NULL);          }
        return (&clnt_res);
}
```

**Invocación al servicio remoto**

# Sun RPC – Ejemplo: Sumatoria de Enteros

```
// Definición del Protocolo. vadd.x
typedef int iarray<>;

program VADD_PROG {
  version VADD_VERSION {
    int VADD(iarray) = 1;
  } = 1;
} = 555575555;
```

# Sun RPC – Ejemplo: Sumatoria de Enteros

## Generado con Solaris 10

Rpcgen vadd.x

**vadd.h**

```
#ifndef _VADD_H_RPCGEN
#define  _VADD_H_RPCGEN
#include <rpc/rpc.h>

typedef struct {
        u_int iarray_len;
        int *iarray_val;
} iarray;
#define  VADD_PROG
        555575555
#define  VADD_VERSION    1
#define  VADD    1
extern  int * vadd_1();
extern int vadd_prog_1_freeresult();
/* the xdr functions */
extern bool_t xdr_iarray();
# endif /* !_VADD_H_RPCGEN */
```

# Sun RPC – Ejemplo: Sumatoria de Enteros

## Generado con Solaris 10 Rpcgen -C vadd.x

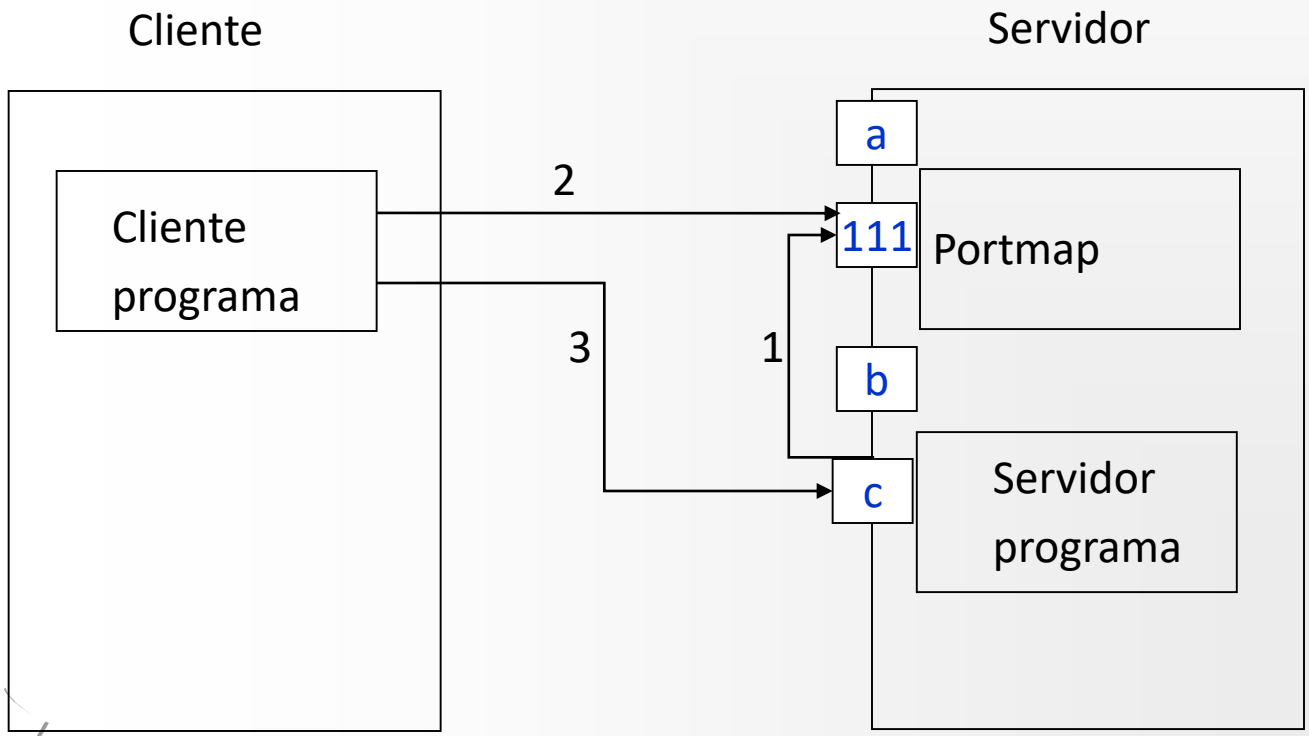
### vadd.h

```
#ifndef _VADD_H_RPCGEN
#define _VADD_H_RPCGEN
#include <rpc/rpc.h>
#ifdef __cplusplus
extern "C" {
#endif
typedef struct {
        u_int iarray_len;
        int *iarray_val;
} iarray;
#define VADD_PROG 555575555
#define VADD_VERSION 1
.....
#define VADD 1
extern int * vadd_1(iarray *, CLIENT *);
extern int * vadd_1_svc(iarray *, struct svc_req *);
extern int vadd_prog_1_freeresult(SVCXPRT *, xdrproc_t, caddr_t);

#else /* K&R C */
#define VADD 1
extern int * vadd_1();
extern int * vadd_1_svc();
extern int vadd_prog_1_freeresult();
#endif /* K&R C */
```

# Comunicación Remota - Sun RPC

- **Utilidad del RPCBind (Portmap)**
  1. Servidor se registra en el portmap (rpcbind)
  2. Cliente obtiene el puerto del servidor desde el portmap (rpcbind)
  3. Cliente invoca al servidor



# Agenda

1. Introducción
    1. Modelos de Comunicaciones.
    2. Tipos de Comunicación.
    3. Paradigmas de comunicación.
2. Pasaje de Mensajes.
3. Comunicación Directa: mensajes, sockets.
4. Comunicación Remota: request-reply, RPC, RMI.
5. Llamadas a Procedimiento Remoto (RPC): concepto e implementación.
6. Comunicación Indirecta:  Grupo, MOM, Publica-Suscribe.
7. Sockets: concepto e implementación.

# Comunicación INDIRECTA

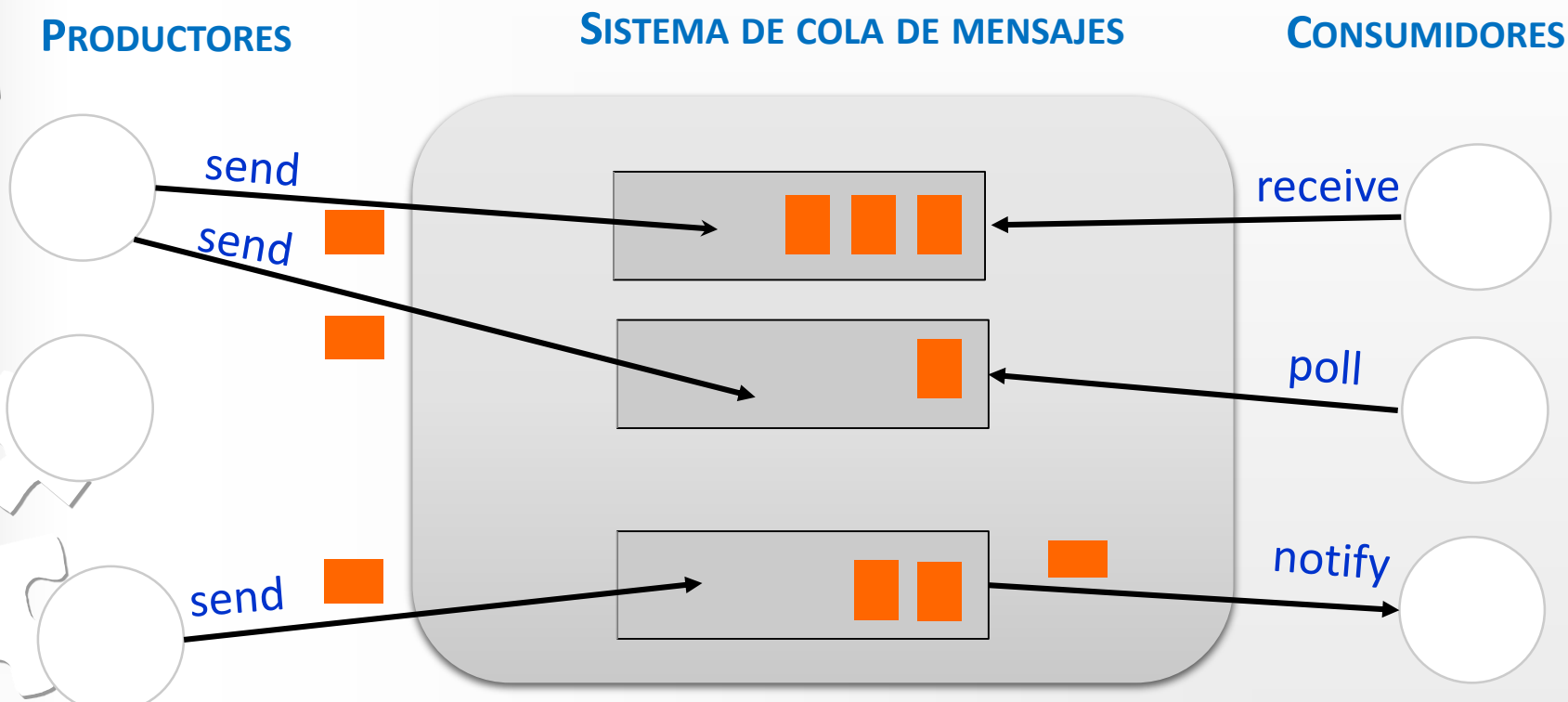
«All problems in computer science can be solved by another level of indirection» Roger Needham, Maurice Wilkes and David Wheeler

La comunicación **INDIRECTA** se define como la comunicación entre entidades de un sistema distribuido a través de un intermediario sin acoplamiento directo entre el emisor y el/los receptor/es.

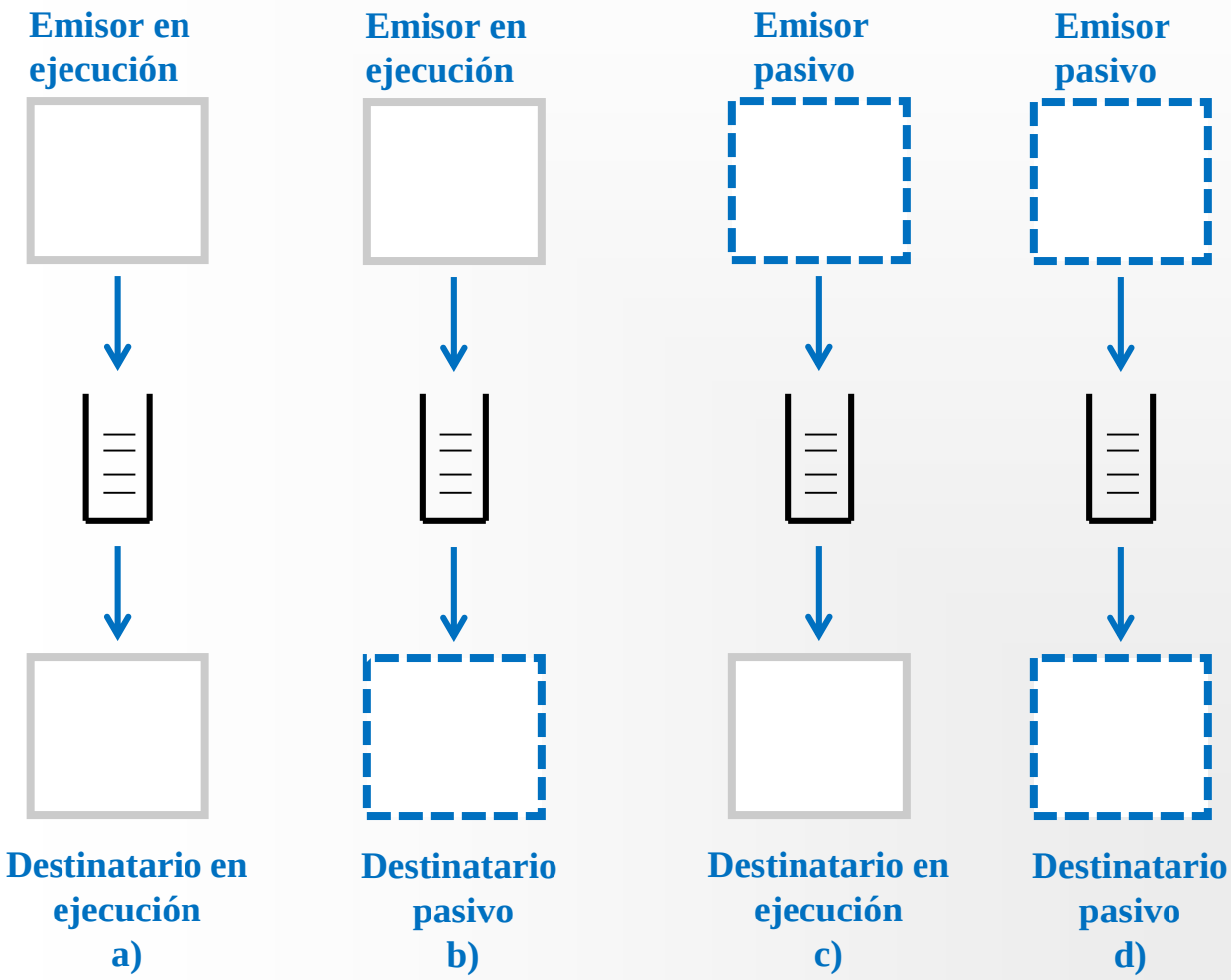
- Desacoplamiento en espacio
- Desacoplamiento en tiempo

# Comunicación Indirecta y Persistente - MOM

## - MOM (Message Oriented Middleware)

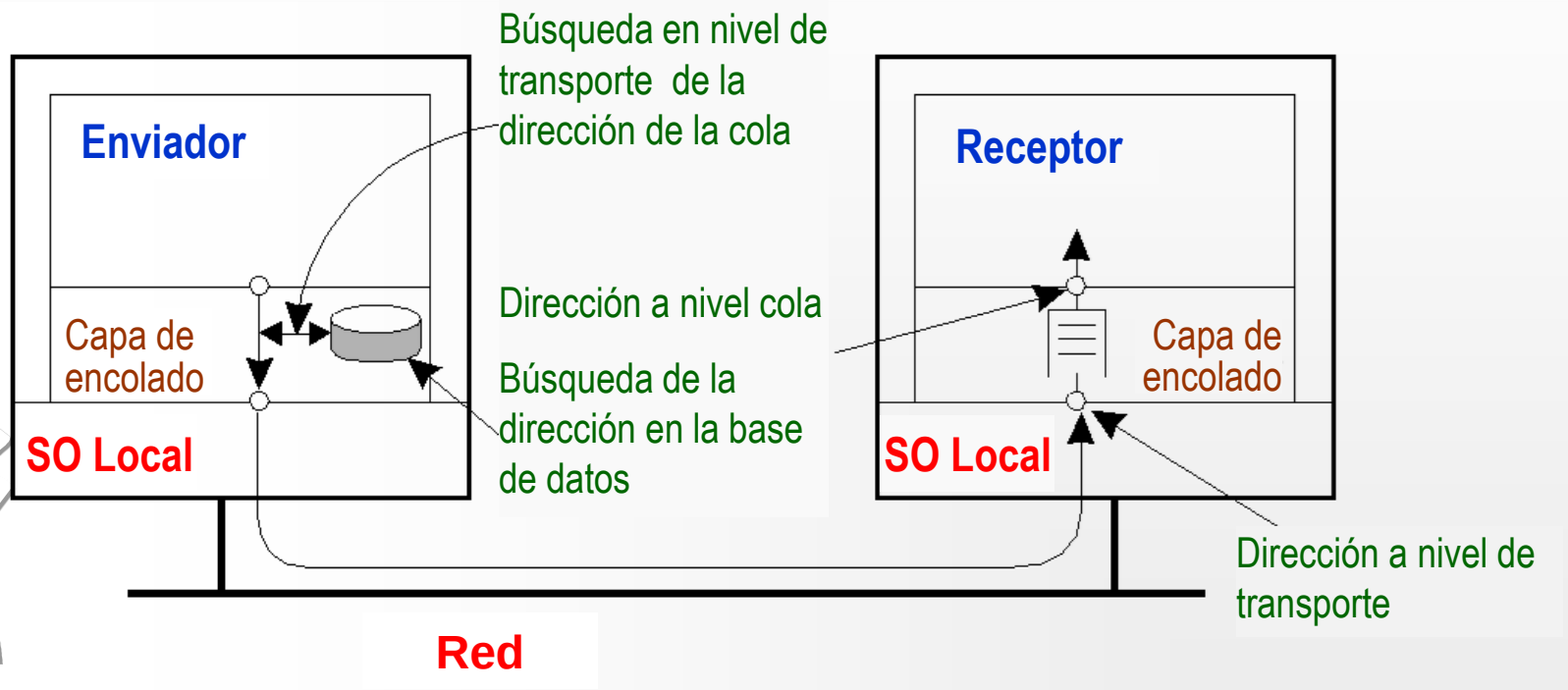


# Comunicación Indirecta y Persistente - MOM



# Comunicación Indirecta y Persistente - MOM

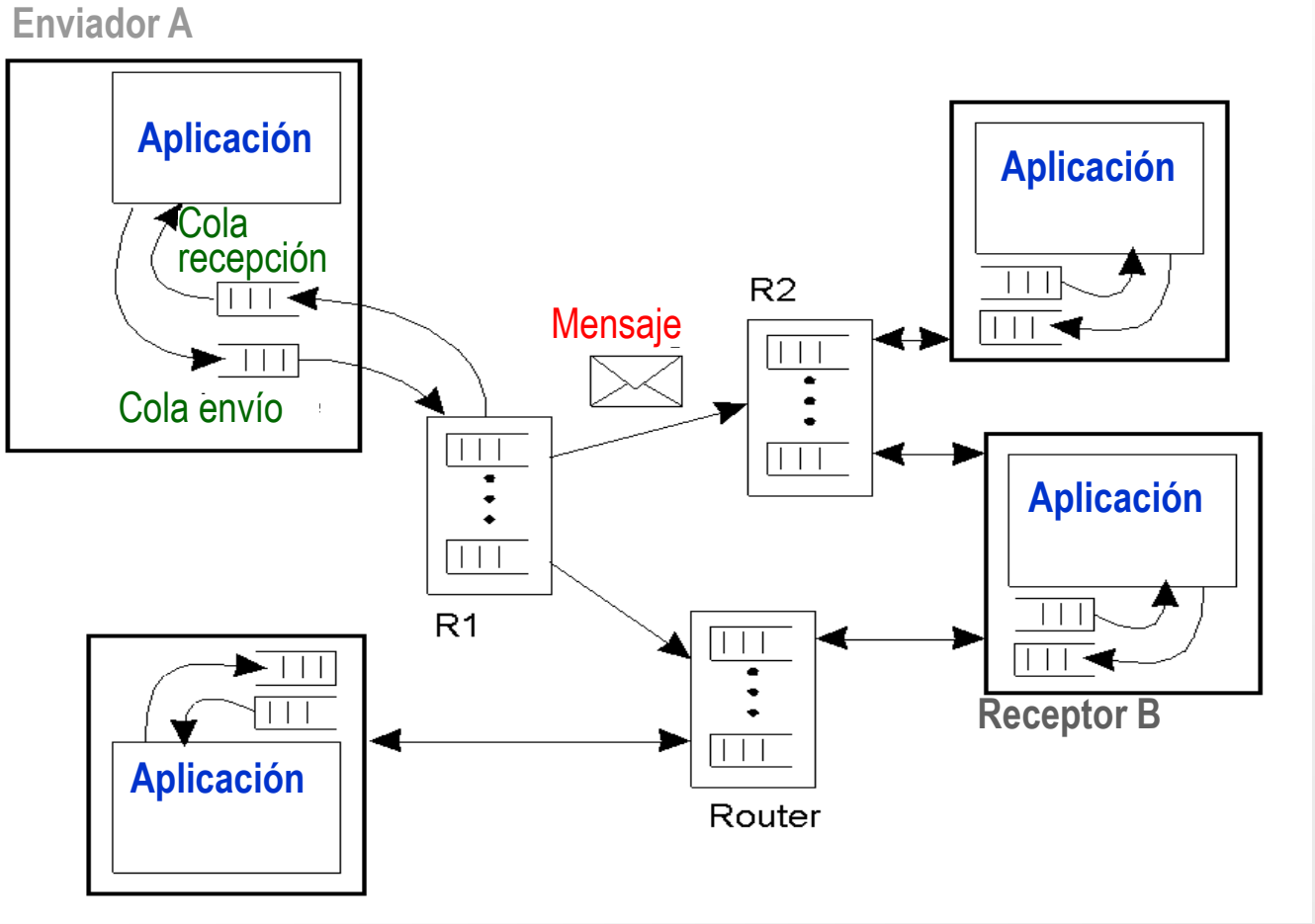
Arquitectura general de un sistema de mensajes en colas



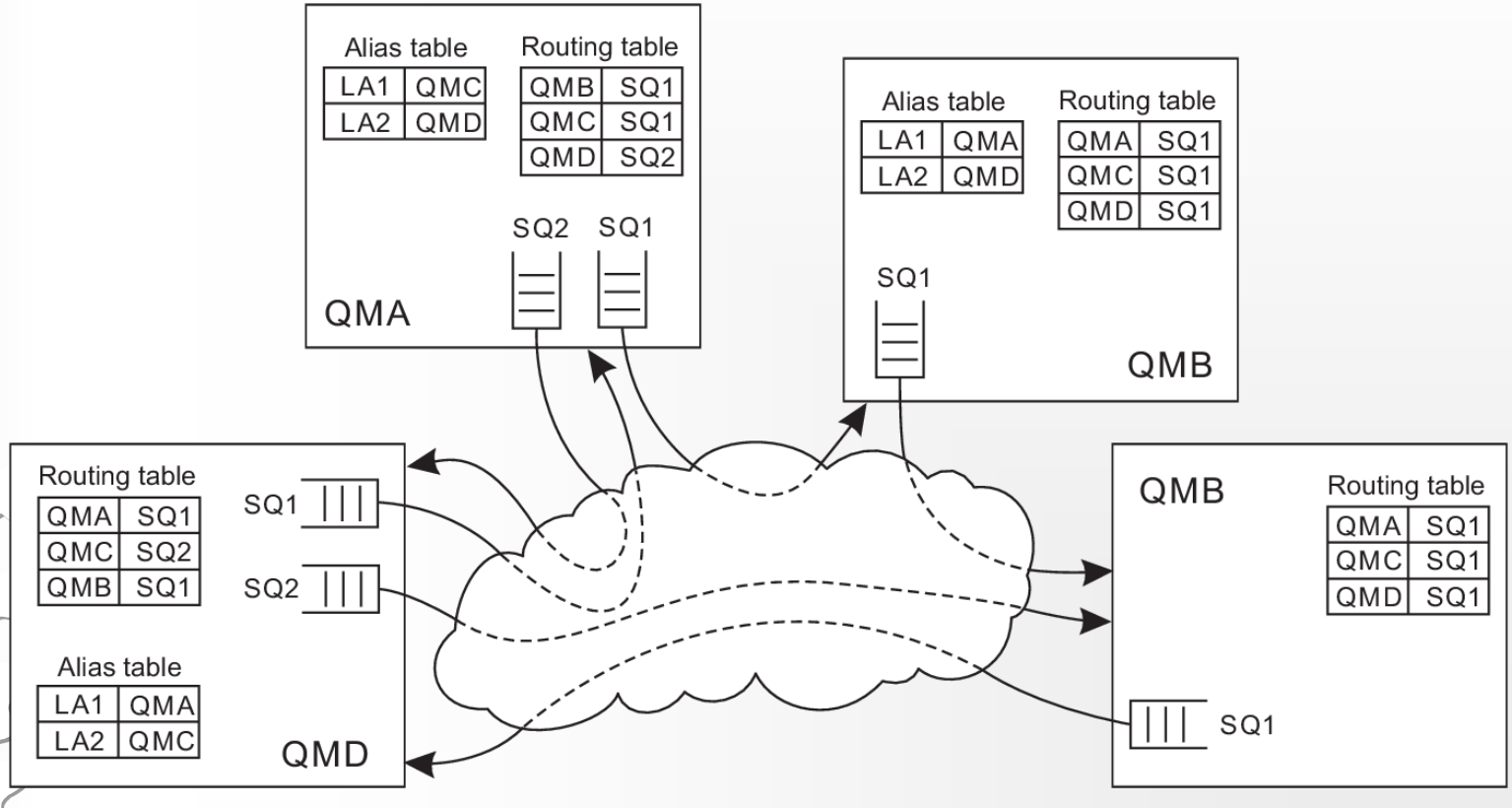

Relación entre direcciones a nivel de colas y direcciones a nivel de red.

# Comunicación Indirecta y Persistente - MOM

Organización general de un sistema de cola de mensajes con routers.

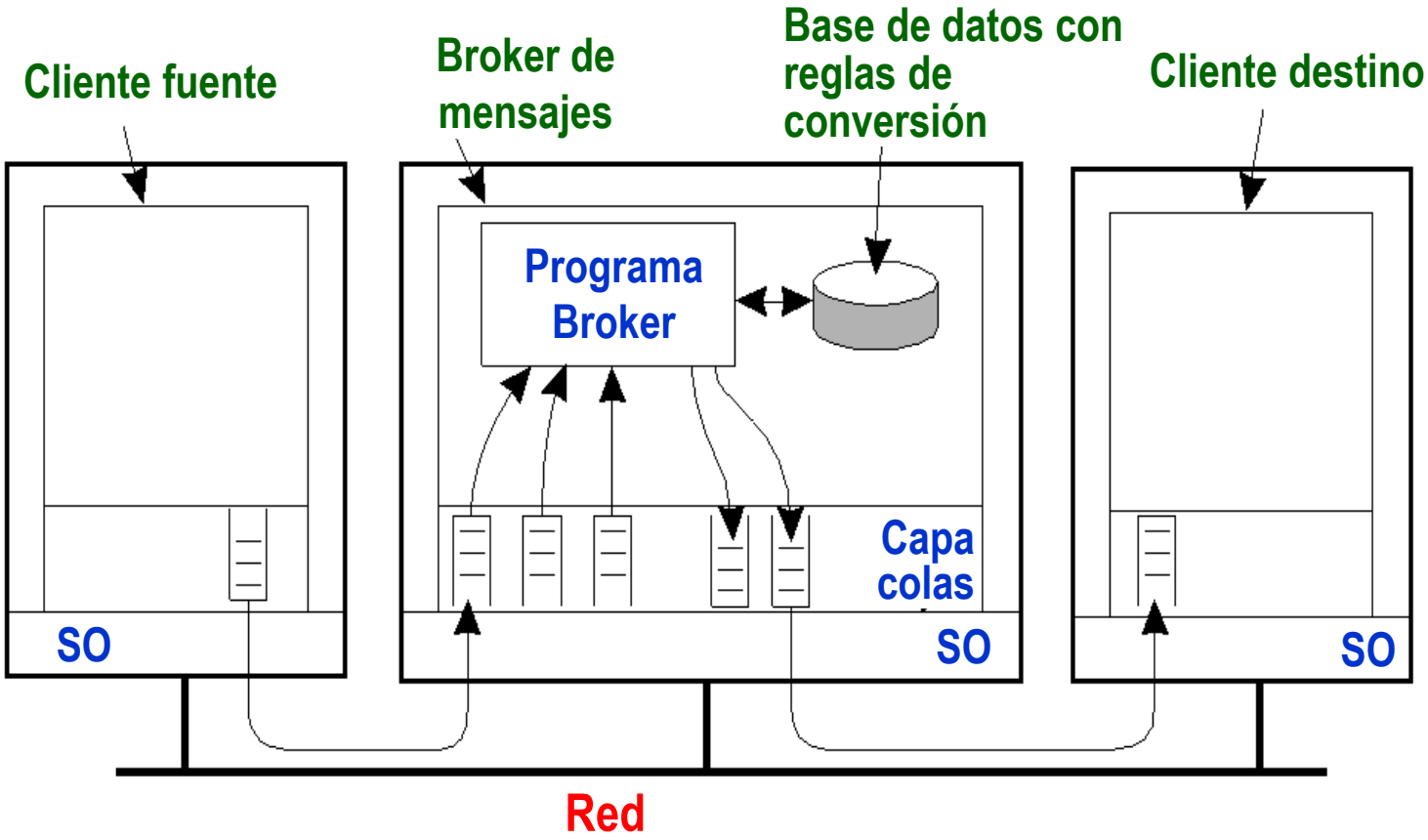


# Comunicación Indirecta y Persistente - MOM



# Comunicación Indirecta y Persistente - MOM

## – Brokers de Mensaje




Organización general de un broker de mensajes en un sistema de mensajes encolados

# Comunicación Indirecta: Grupos

Ofrece un servicio donde un mensaje es enviado a un grupo y luego es entregado a todos los miembros del mismo.

Áreas de interés:

- Difusión fiable de la información a un gran número de clientes
- Soporte para aplicaciones colaborativas
- Soporte para sistema de monitoreo y administración

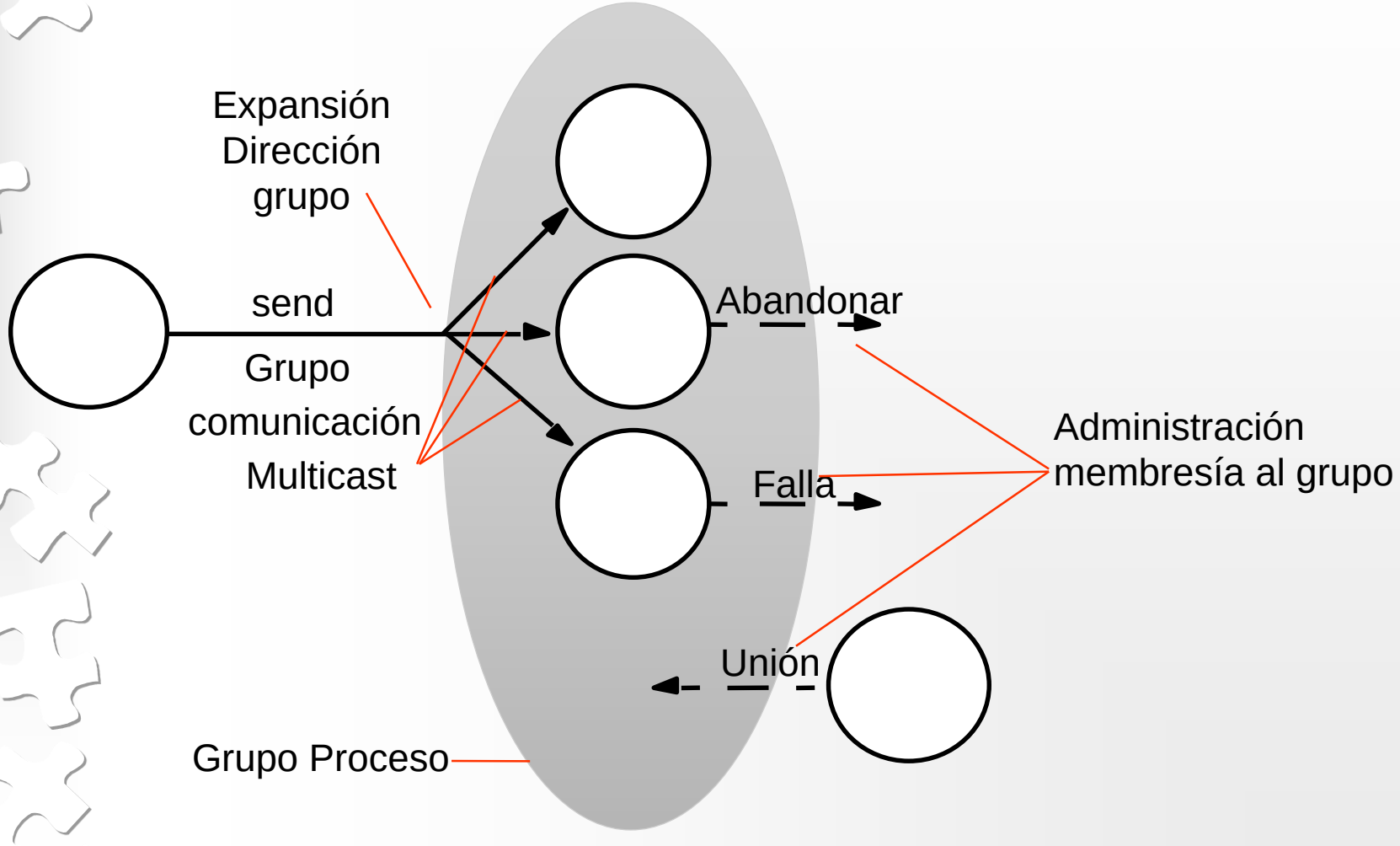
# Comunicación Indirecta: Grupos

Un sistema de pasaje de mensajes con la facilidad de grupo de comunicación provee la flexibilidad de crear y borrar grupos dinámicamente y permitir a un proceso agregarse o dejar un grupo.

Un mecanismo para realizar todo esto es un *servidor de grupos*.

Esta solución sufre de pobre confiabilidad y pobre escalabilidad.

# Comunicación Indirecta: Grupos de comunicación



# Comunicación Indirecta: Grupos

Hay tres tipos de grupos de comunicación:

- Uno a muchos
- Muchos a uno
- Muchos a muchos

# Comunicación Indirecta: Grupos

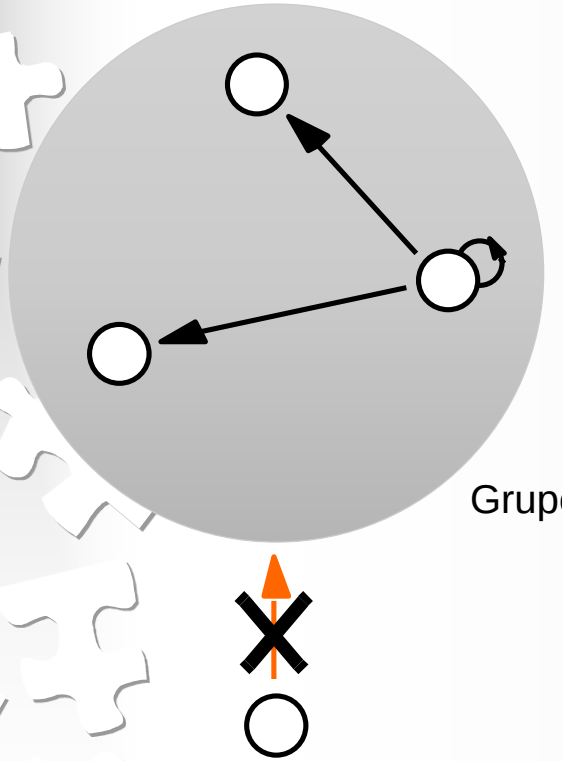
## Uno a muchos

Este esquema es conocido como *comunicación multicast*.

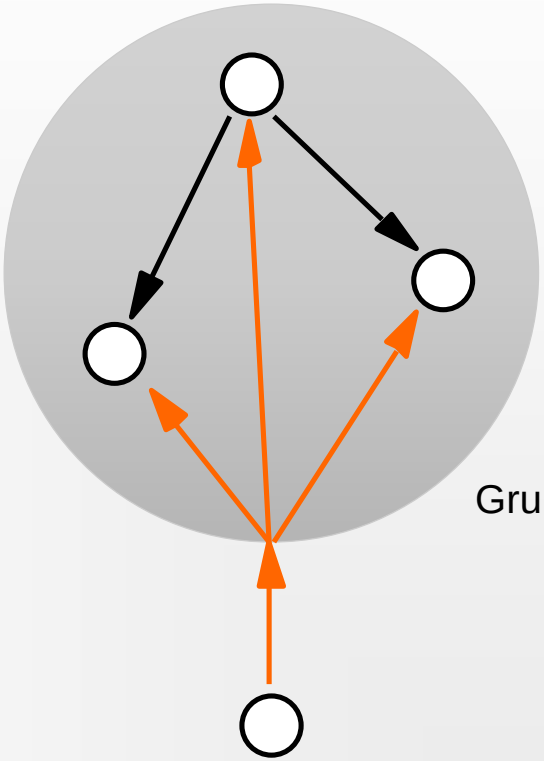
En este caso los procesos receptores de los mensajes constituyen un grupo, que a su vez pueden ser de dos tipos:

- Grupos cerrados
- Grupos abiertos

# Comunicación Indirecta: Grupos



Grupo Cerrado



Grupo Abierto

# Comunicación Indirecta: Grupos

## Muchos a uno

Emisores (*senders*) múltiples envían mensajes a un único receptor.

Hay un *no determinismo.*

## Muchos a muchos

Múltiples emisores envían mensajes a múltiples receptores.

# Comunicación Indirecta: Grupos

## Fiabilidad en la Comunicación Multicast

- ✓Validez – el mensaje eventualmente será entregado.
- ✓Integridad – el mensaje es entregado correctamente a lo sumo una vez.

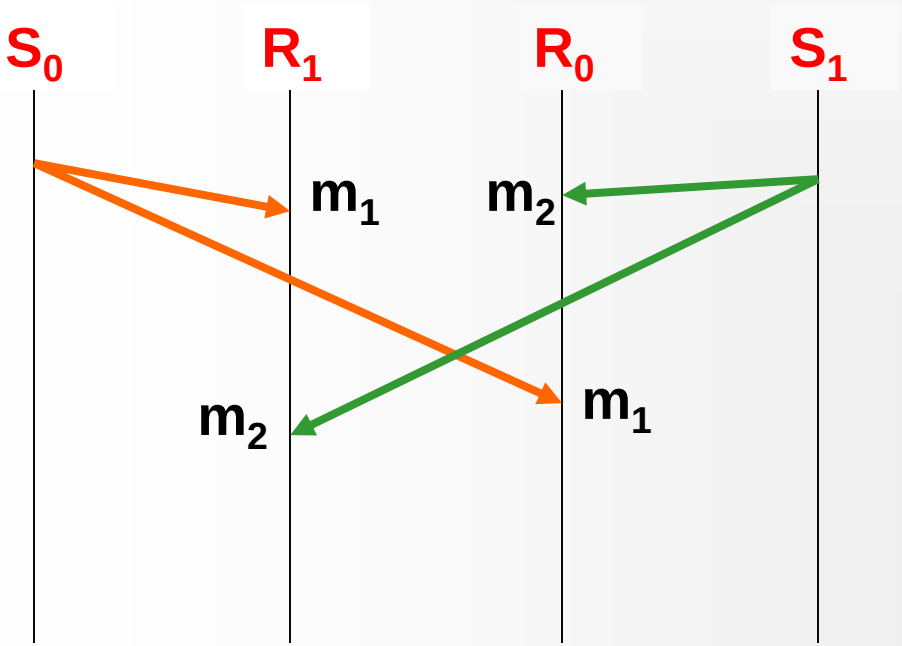


Ordenamiento de los mensajes

# Comunicación en Sistemas Distribuidos

La cuestión más importante en este esquema es el ordenamiento de los mensajes.

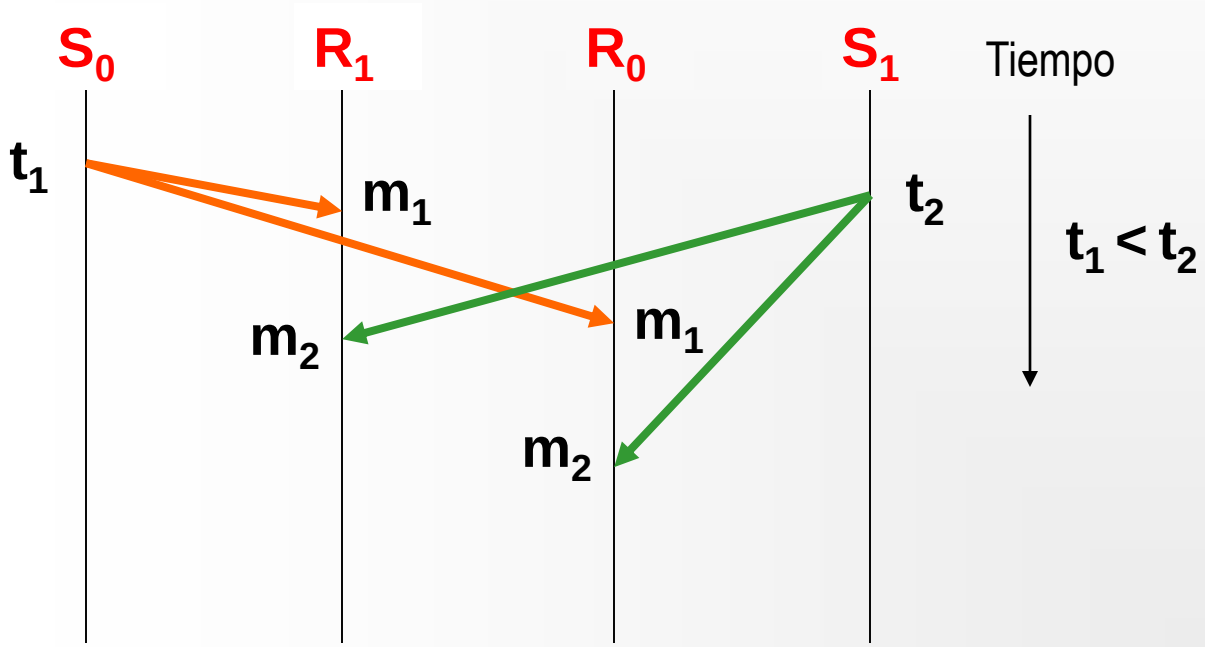
No hay restricción de orden



# Comunicación en Sistemas Distribuidos

## ORDENAMIENTO DE LOS MENSAJES

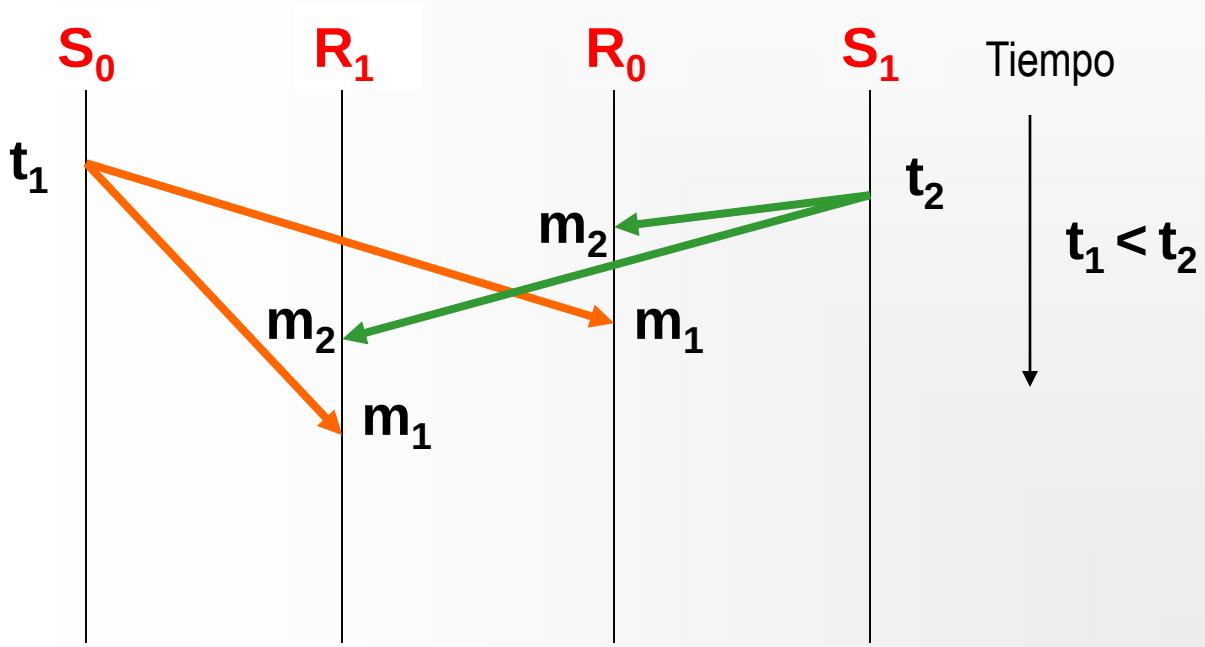
Ordenamiento absoluto



# Comunicación en Sistemas Distribuidos

## ORDENAMIENTO DE LOS MENSAJES

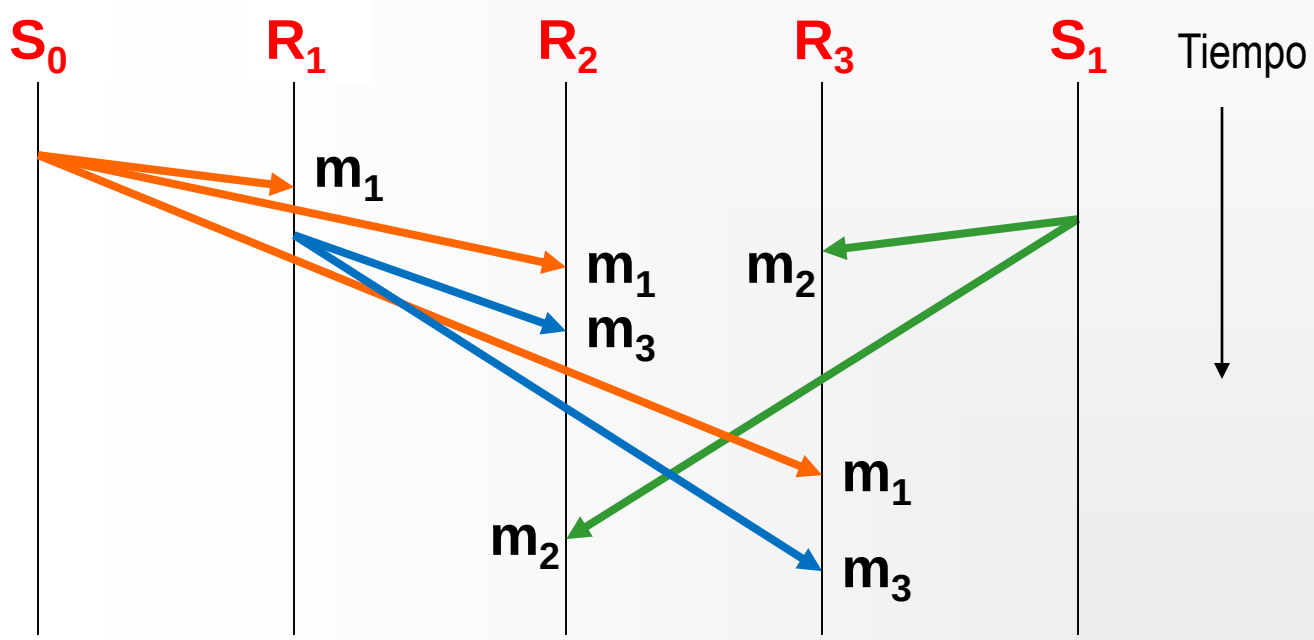
Ordenamiento consistente



# Comunicación en Sistemas Distribuidos

## ORDENAMIENTO DE LOS MENSAJES

Orden causal



# Comunicación Indirecta: Publica-Suscribe

**Un sistema publica-suscribe es un sistema en el que los editores (*publishers*) publican eventos estructurados a un servicio de sucesos y los suscriptores expresan interés en eventos particulares a través de suscripciones.**

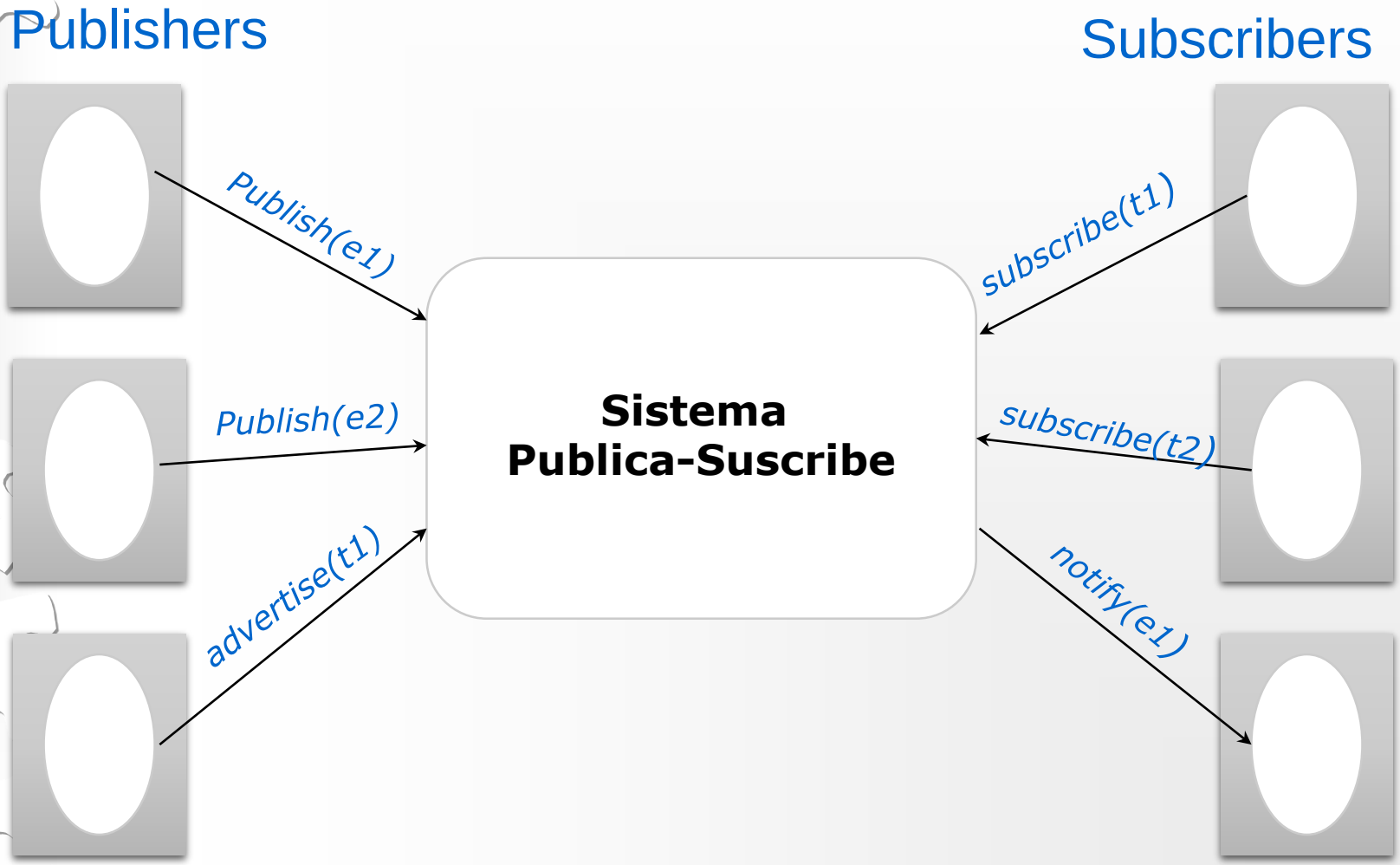
Comunicación es **UNO-A-MUCHOS**

**Características**

**Heterogeneidad**

**Asincrónico**

# Comunicación Indirecta: Publica-Suscribe



# Agenda

1. Introducción
    1. Modelos de Comunicaciones.
    2. Tipos de Comunicación.
    3. Paradigmas de comunicación.
2. Pasaje de Mensajes.
3. Comunicación Directa: mensajes, sockets.
4. Comunicación Remota: request-reply, RPC, RMI.
5. Llamadas a Procedimiento Remoto (RPC): concepto e implementación.
6. Comunicación Indirecta:  Grupo, MOM, Publica-Suscribe.
7. Sockets: concepto e implementación.

# Comunicación Directa - Sockets

¿Qué es un socket?

- Es una interfaz de entrada-salida de datos que permite la intercomunicación entre procesos.
- Es un punto final (endpoint) en la comunicación, el cual una aplicación puede escribir datos que serán enviados por la red y desde el cual ingresará los datos que puede leer.

# Comunicación Directa - Sockets

Dominios de Comunicación

- Se utilizan para especificar dónde se encuentran los procesos que se van a intercomunicar.

- Ejemplos:
  - AF_UNIX
  - AF_INET → **Red TCP/IP**

# Comunicación Directa - Sockets

Tipos de sockets en el dominio AF_INET

- Sockets stream – orientados a la conexión, secuenciado, sin duplicación de paquetes y libres de errores.
- Sockets datagram – orientados a sin-conexión, pueden llegar fuera de secuencia, puede o no contener errores.
- Sockets Raw

# Comunicación Directa - Sockets

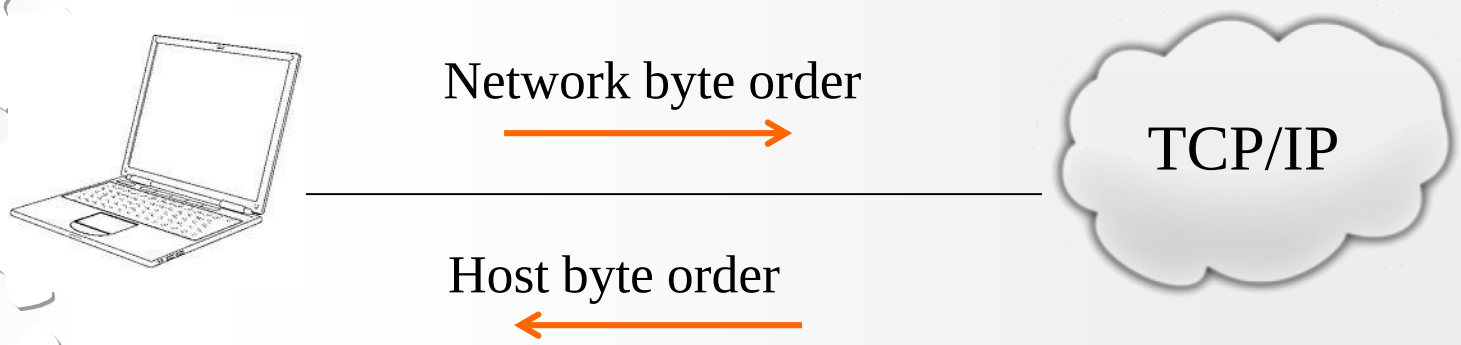
Orden de los bytes

- Es la forma en la cual el sistema almacena los datos en memoria.
  - Orden bytes de la red (network byte order)
  - Orden bytes de la máquina (host byte order)

# Comunicación Directa - Sockets

Orden de los bytes

- Todos los bytes que se transmiten, deben estar en network byte order
- Todos los datos que se reciben de la red, deben convertirse a host byte order.



# Comunicación Directa - Sockets

Orden de los bytes – Funciones de conversión

- Htons() – host to network short
- Htonl() – host to network long
- Ntohs() – network to host short
- Ntohl() – network to host long

**Port = htons(3490);**

# Comunicación Directa - Sockets

Funciones asociadas

- **Socket():** esta función se utiliza para crear un socket, retorna un descriptor del mismo.
- Formato General:
  - Sockfd = socket(int dominio, int tipo, int protocolo)

**AF_INET**
**AF_UNIX**

**SOCK_STREAM**
**SOCK_DGRAM**
**SOCK_RAW**

# Comunicación Directa - Sockets

Funciones asociadas

- **Bind():** esta función se utiliza para darle un nombre al socket, o sea, una dirección IP y número de puerto del host local por donde escuchará.
- Formato general:
  - int bind (int sockfd, struct sockaddr *my_addr, int addrlen)

# Comunicación Directa - Sockets

Funciones asociadas

- **Listen()**: esta función habilita el socket para que pueda recibir conexiones. Sólo se aplica a sockets tipo SOCK_STREAM
- Formato general:
  - int listen(int sockfd, int backlog)



**Número máximo de conexiones**

# Comunicación Directa - Sockets

Funciones asociadas

- **Accept()**: esta función retorna un nuevo descriptor de socket al recibir la conexión del cliente en el puerto configurado.
- Formato general:
  - int accept(int sockfd, void *addr, int *addrlen)

# Comunicación Directa - Sockets

Funciones asociadas

- **Connect():** Inicia la conexión con el servidor remoto, lo utiliza el cliente para conectarse.
- Formato General:
  - int connect ( int sockfd, struct sockaddr *serv_addr, int addrlen)

# Comunicación Directa - Sockets

Funciones asociadas

- Send() y Receive(): estas funciones se utilizan para la transferencia de datos sobre sockets stream
- send (int sockfd, const void *msg, int len, int flags)
- recv (int sockfd, void *buf, int len, unsigned int flags)

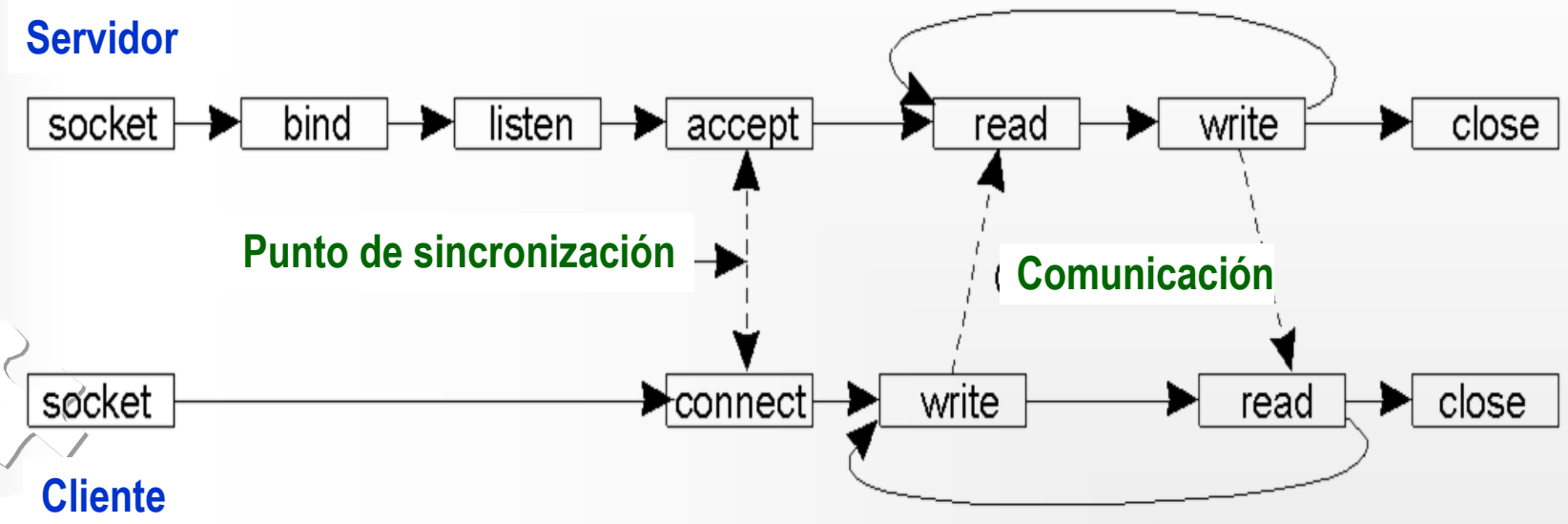
# Comunicación Directa - Sockets

Funciones asociadas

- close () y shutdown (): son funciones que se utilizan para cerrar el descriptor del socket.
- close (sockfd) → **Queda deshabilitado**
- shutdown (sockfd, int how)

**0 : Se deshabilita la recepción.**
**1 : se deshabilita el envío.**
**2 : se deshabilitan la recepción y el envío, igual que en close ()**

# Comunicación Directa - Sockets



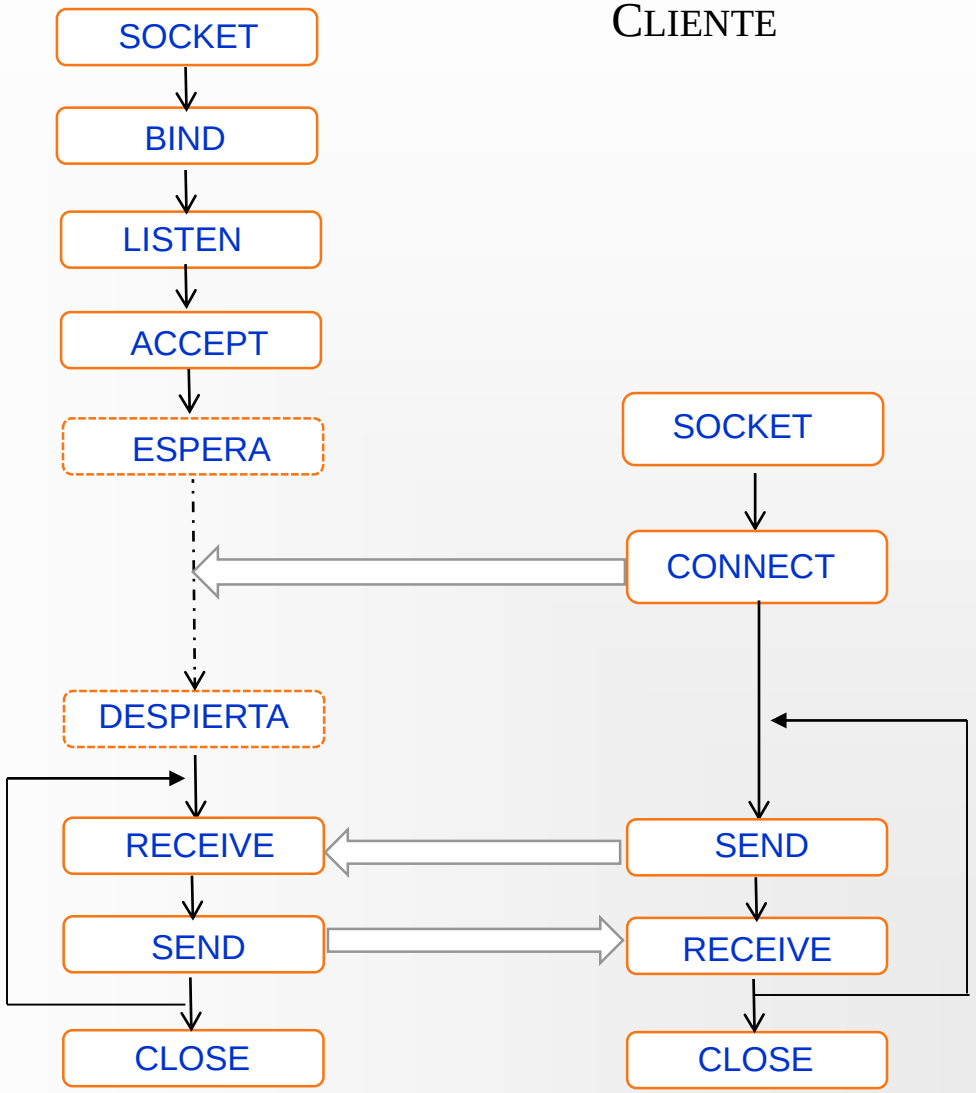

Modelo de comunicación orientado a conexión usando sockets.

# Comunicación Directa - Sockets

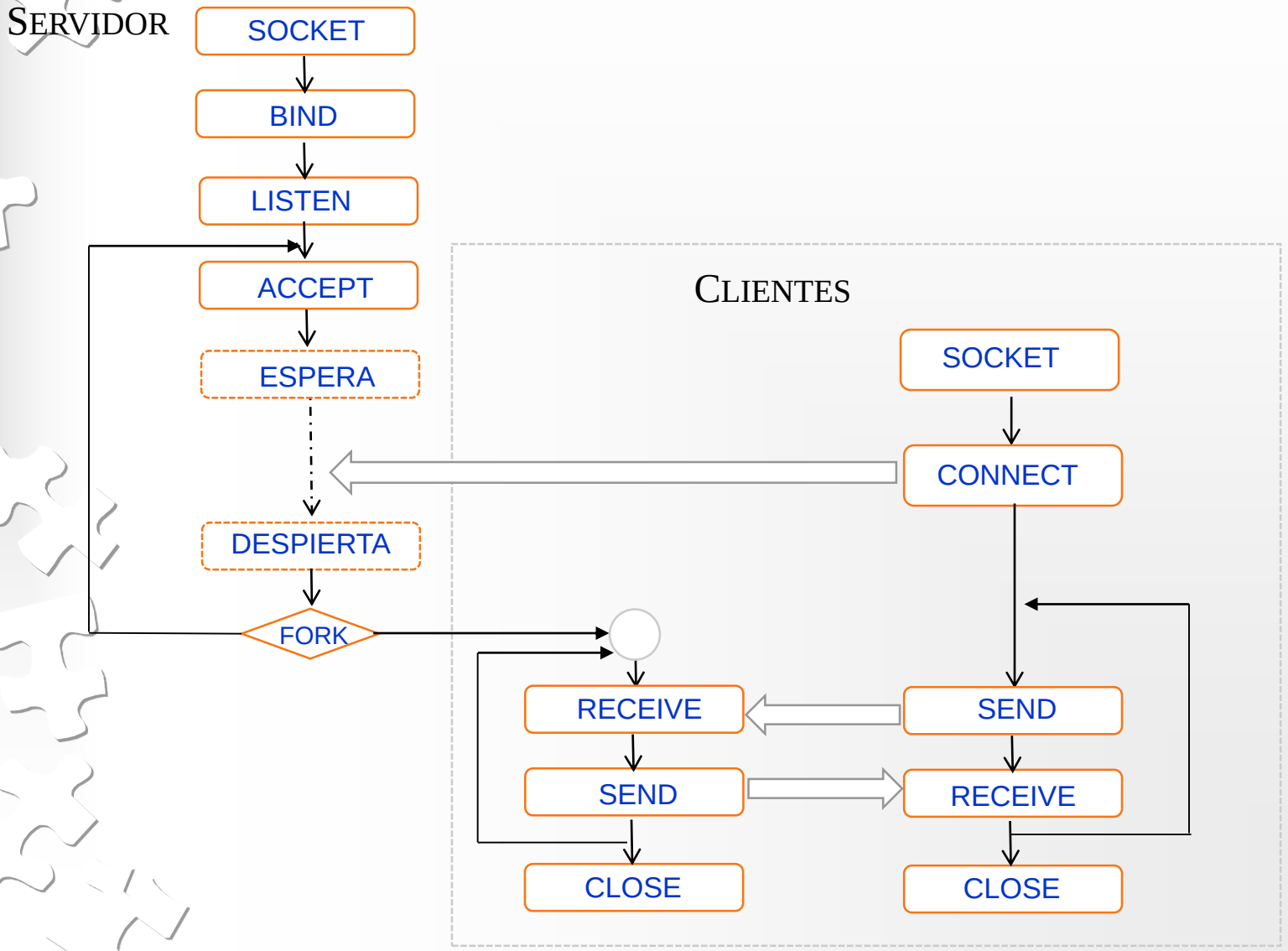
- Formato general de la comunicación con sockets

| Sockets con stream TCP | |
|---|---|
| Cliente | Servidor |
| Socket | Socket<br>Bind<br>Listen |
| Connect | Accept |
| Send / receive | Send / receive |

# Comunicación Directa – Sockets: Ejemplo



# Comunicación Directa – Sockets: Ejemplo



# Comunicación Directa - Sockets

- Formato general de la comunicación con sockets

| Sockets con datagrama UDP | |
|---|---|
| Cliente | Servidor |
| Socket | Socket<br>Bind |
| Sendto / recvfrom | Sendto / recvfrom |

# Comunicación Directa - Sockets

- Ejemplo Servidor TCP – Hola Mundo

```
#define MYPORT 14550 /*Nro de puerto donde se conectaran los clientes*/
#define BACKLOG 10 /* Tamaño de la cola de conexiones recibidas */
main() {
int sockfd;, /* El servidor escuchara por sockfd */
int newfd; /* las transferencias de datos se realizar mediante newfd */
struct sockaddr_in my_addr; /* contendrá la dir IP y el nro de puerto local */
struct sockaddr_in their_addr; //Contendrá la dir IP y nro de puerto del cliente
int sin_size; /* Contendra el tamaño de la escructura sockaddr_in */
if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1)  ....
// Asigna valores a la estruct my_addr para luego poder llamar a la función bind()
my_addr.sin_family = AF_INET;
my_addr.sin_port = htons(MYPORT); /*formato de network byte order */
my_addr.sin_addr.s_addr = INADDR_ANY; /* automaticamente usa IP local */
bzero(&(my_addr.sin_zero), 8); /* rellena con ceros el resto de la estructura */
/* Asigna un nombre al socket */
if ( bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr)) == -1)
```

# COMUNICACIÓN DIRECTA - SOCKETS

```
  ……
/* Habilita el socket para recibir conexiones, con una cola de x conexiones en espera como máximo */
    if (listen(sockfd, BACKLOG) == -1)
     ……
 while(1) /* loop que llama a accept() */
    {        sin_size = sizeof(struct sockaddr_in);
    /*Se espera por conexiones ,*/
    if ((newfd = accept(sockfd, (struct sockaddr *)&their_addr, &sin_size)) == -1)
    ……
    printf("server: conexion desde: %s\n", inet_ntoa(their_addr.sin_addr));
    printf(“Desde puerto: %d \n””, ntohs(their_addr.sin_port));
if (!fork())
    { /* Comienza el proceso hijo, enviamos los datos mediante newfd */
    if (send(newfd, "Hello, world!\n", 14, 0) == -1)
    perror("send");
    close(newfd);
    exit(0);
 } close(newfd);
    ……
```

# Comunicación Directa - Sockets

- Ejemplo Cliente TCP – Hola Mundo

```
#define PORT 14550 /* El puerto donde se conectara */
#define MAXDATASIZE 100
int main(int argc, char *argv[])
{
int sockfd, numbytes;
char buf[MAXDATASIZE]; /* Buffer donde se reciben los datos */
struct hostent *he; /* Se utiliza para convertir el nombre del host
a su dirección IP */
struct sockaddr_in their_addr; /* dirección del server donde se
conectara */
if (argc != 2) { ……. }
if ((he=gethostbyname(argv[1])) == NULL) { ……. }
/* Creamos el socket */
if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
{ … }
```

# Comunicación Directa - Sockets

```
/* Establecemos their_addr con la dirección del server */
    their_addr.sin_family = AF_INET;
    their_addr.sin_port = htons(PORT);
    their_addr.sin_addr = *((struct in_addr *)he->h_addr);
    bzero(&(their_addr.sin_zero), 8);
    /* Intentamos conectarnos con el servidor */
    if (connect(sockfd, (struct sockaddr *)&their_addr, sizeof(struct
    sockaddr)) == -1)
    {  … }
/* Recibimos los datos del servidor */
    if ((umbytes=recv(sockfd, buf, MAXDATASIZE, 0)) == -1)
    {  …. }
    /* Visualizamos lo recibido */
    buf[numbytes] = '\0';
    printf("Recibido: %s\n",buf);
    /* Devolvemos recursos al sistema */
    close(sockfd);
    return 0;
    }
```

## Bibliografía:


- Sinha, P. K.; “Distributed Operating Systems: Concepts and Design”, IEEE Press, 1997.
- Tanenbaum, A.S.; van Steen, Maarten; “Distributed Systems: Principles and Paradigms”. 2nd Edition, Prentice Hall, 2007 and 1st Edition 2002.
- Coulouris,G.F.; Dollimore, J. y T. Kindberg; “Distributed Systems: Concepts and Design”. 5th Edition Addison Wesley, 2011.
- van Steen, Maarten; Tanenbaum, A.S; “Distributed Systems”. 3rd Edition, Prentice Hall, 2017.