# Sistemas Distribuidos

# Comunicación en Sistemas Distribuidos

Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Índice

- Paradigmas de comunicación
  - Paso de mensajes
  - Llamadas a procedimientos remotos (RPC)
    - RPC Sun/ONC (no se estudia en detalle pero se plantea trabajo optativo)
      - http://laurel.datsi.fi.upm.es/~ssoo/SOD.dir/practicas/guiarpc.html
  - Invocación de métodos remotos (RMI)
    - Java RMI (usado en las prácticas de grupo)
      - http://laurel.datsi.fi.upm.es/~ssoo/SD.dir/practicas/guiarmi.html
- Aspectos internos de la comunicación
  - *Zero-copy*
  - Integridad de los mensajes
  - Serialización
  - Sincronización


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Sistemas Distribuidos

## Paso de mensajes

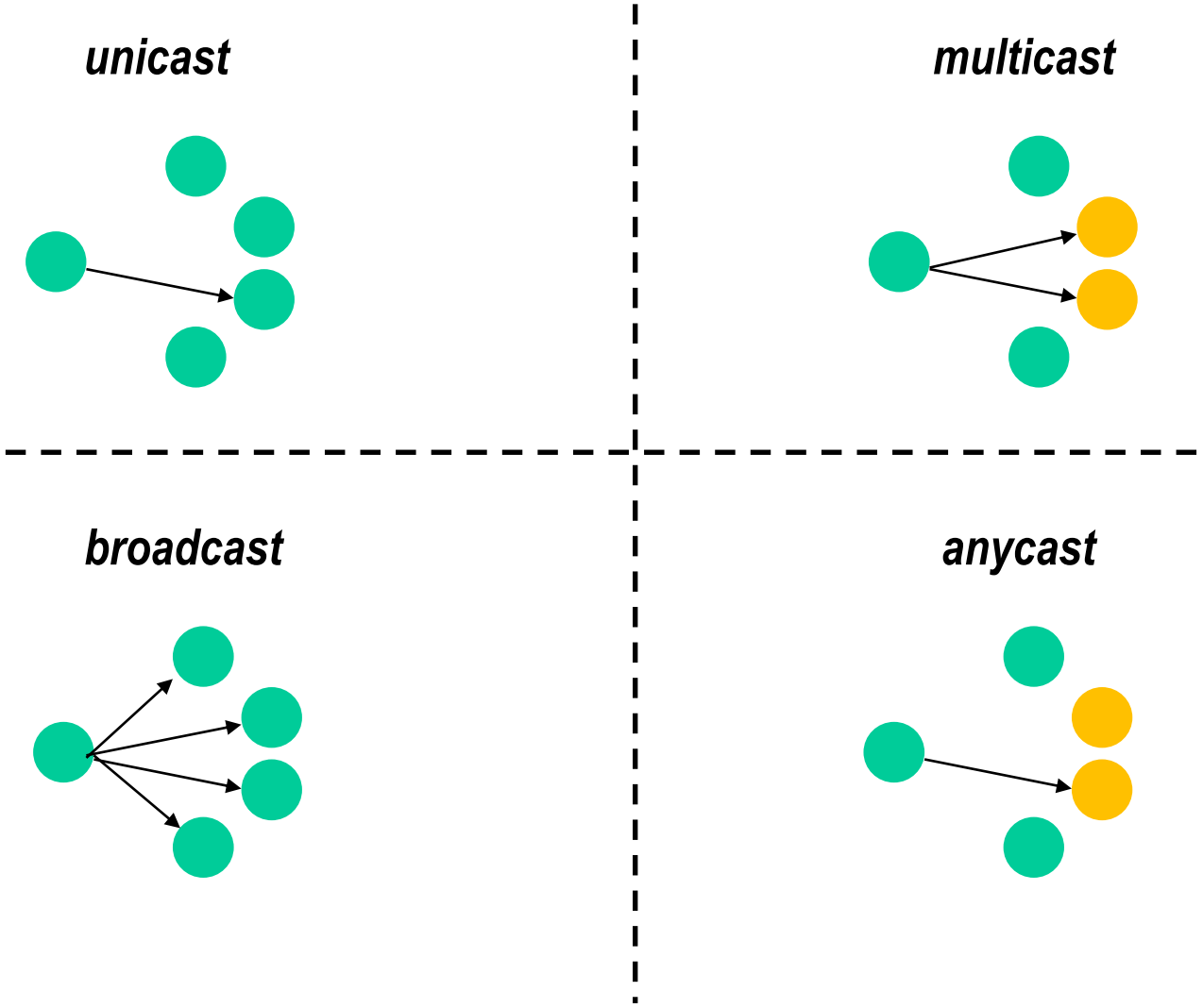
Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Introducción al paso de mensajes

- API para envío/recepción de mensajes
- Alternativas de diseño
  - Cardinalidad
  - Persistencia:
    - ¿Se guardan los mensajes hasta que aparezca un receptor?
    - Posibilita el desacoplamiento temporal
  - Esquemas de direccionamiento
- Paradigmas de comunicación por paso de mensajes
  - Comunicación punto a punto no persistente
    - Sockets (ya estudiados en SS.OO.)
    - *http://laurel.datsi.fi.upm.es/~ssoo/sockets/*
  - Sistemas de colas de mensajes (comunicación persistente)
  - Comunicación de grupo (no persistente)


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

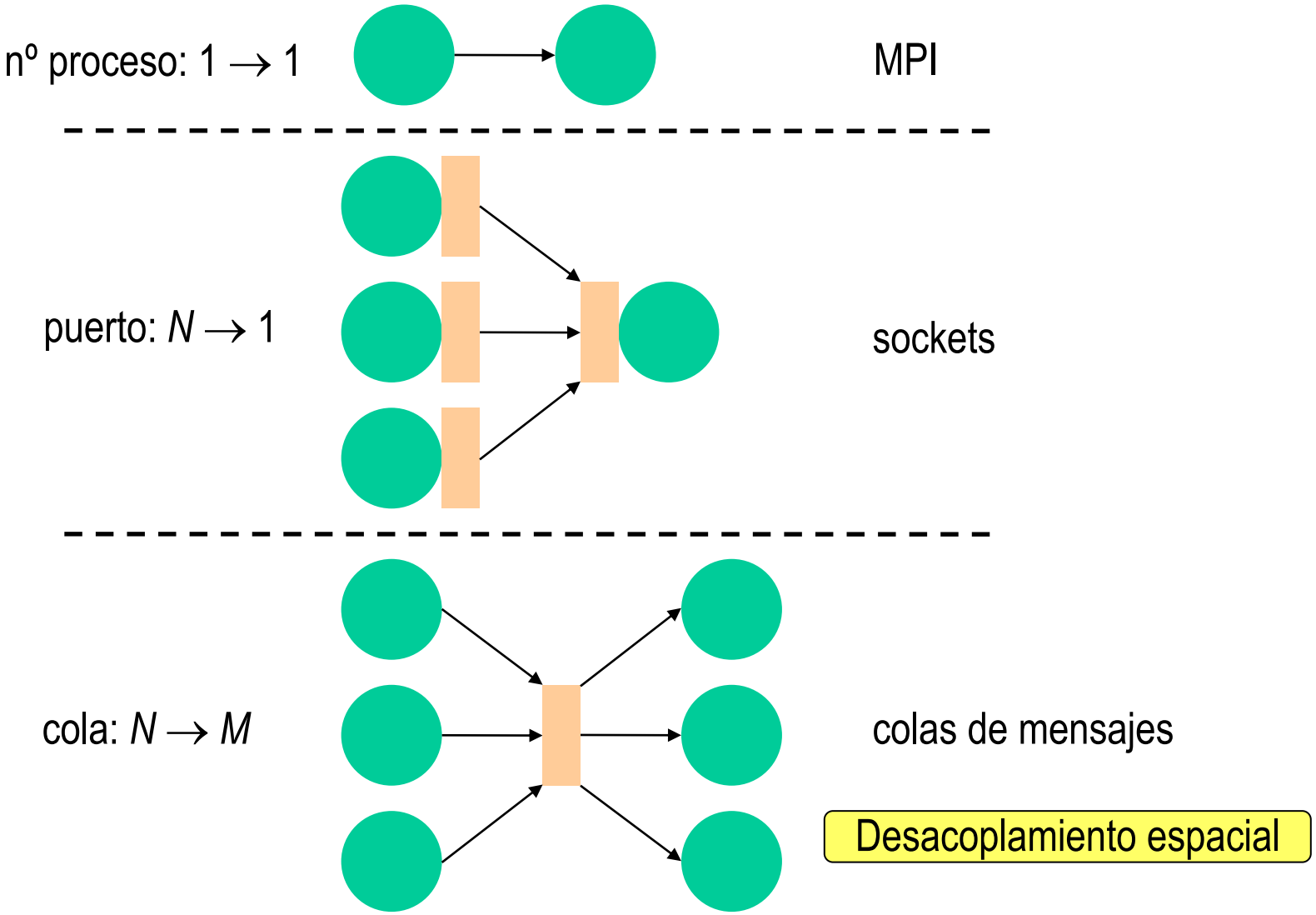
# Cardinalidad





Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Esquemas de direccionamiento



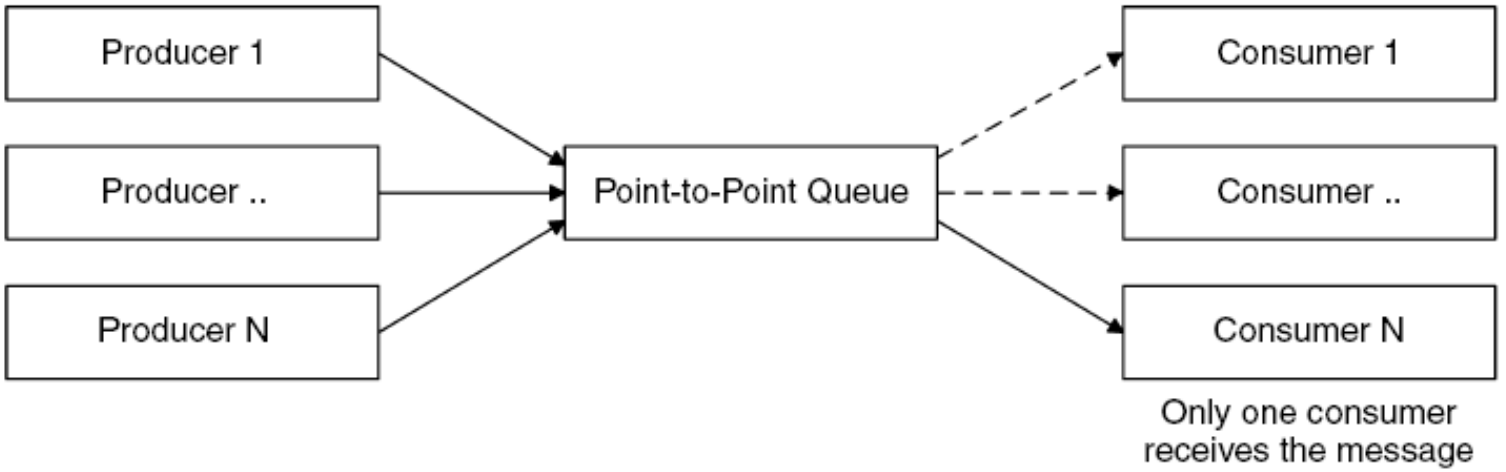


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# MOM – Sistemas de colas de mensajes

- *Message-oriented middleware*:
  - *RabbitMQ, ZeroMQ, Apache Kafka, ActiveMQ*
  - *AMQP* protocolo estándar para MOM
- Comunicación persistente: desacoplada en espacio y tiempo
- Habitualmente, configurable modo de operación:
  - Productor/consumidor
    - Mensaje solo lo recibe un proceso
  - Editor/subscriptor
    - Mensaje lo reciben todos los procesos subscritos
- Características avanzadas habituales:
  - Mensajería con transacciones
  - Transformaciones de mensajes
- Apropiado para integración de aplicaciones de empresa (EAI)

# Productor/consumidor vs. editor/subscriptor





***Message-Oriented Middleware*. Edward Curry**

# Multidifusión: comunicación de grupo

Destino de mensaje → grupo de procesos
- Envío/recepción especifican dirección de grupos de procesos
- Desacoplamiento espacial

Trabajo seminal: ISIS (posteriores Horus, Ensemble, JGroups)

Aplicaciones:
- Gestión de réplicas
  - Réplicas son miembros del mismo grupo
  - Cliente envía peticiones al grupo
- Modelo editor/subscriptor
  - Uso de 1 grupo/tema
  - Subscriptor se hace miembro del grupo
  - Editor envía mensajes al grupo

Implementación depende de si red tiene *multicast* (*IP-multicast*)
- Si no, se implementa enviando *N* mensajes

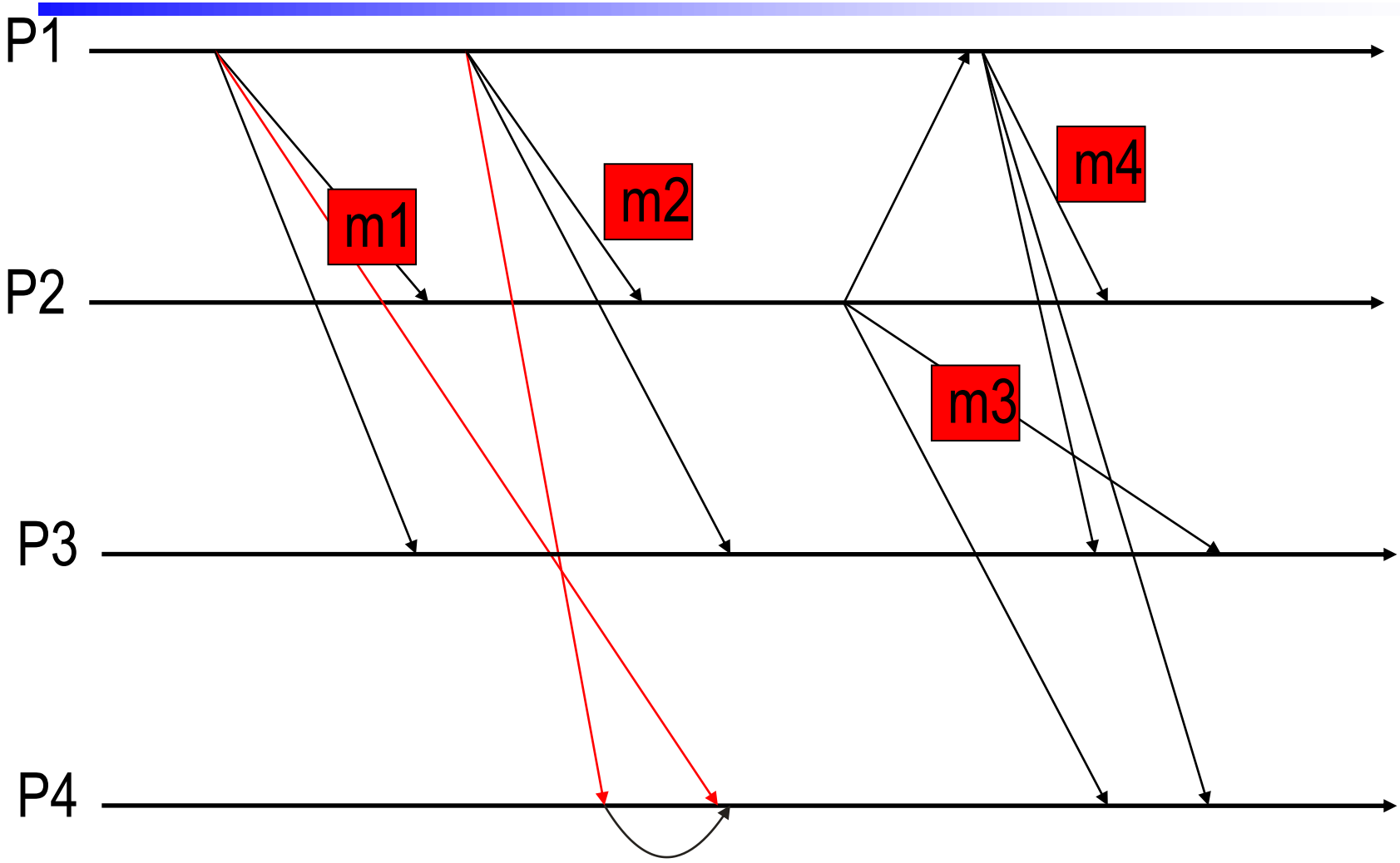

Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Aspectos de diseño de com. de grupo

- Atomicidad: o reciben el  mensaje o ninguno
  - Con unidifusión fiable (TCP): en medio, se puede caer emisor
  - Con multicast IP: pérdida de mensajes
- Orden de recepción de los mensajes
  - **FIFO**: mensajes de misma fuente llegan en orden de envío
    - No garantía sobre mensajes de distintos emisores
  - **Causal**: entrega respeta relación "causa-efecto"
    - Si no hay relación, no garantiza ningún orden de entrega
  - **Total**: Todos los mensajes recibidos en mismo orden por todos
- El grupo suele tener carácter dinámico
  - Se pueden incorporar y retirar procesos del grupo
  - **Vista**: conjunto de procesos en el grupo en un instante dado
  - Procesos son notificados de los cambios de vista
  - Pertenencia debe coordinarse con comunicación→ **sincronía virtual**


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

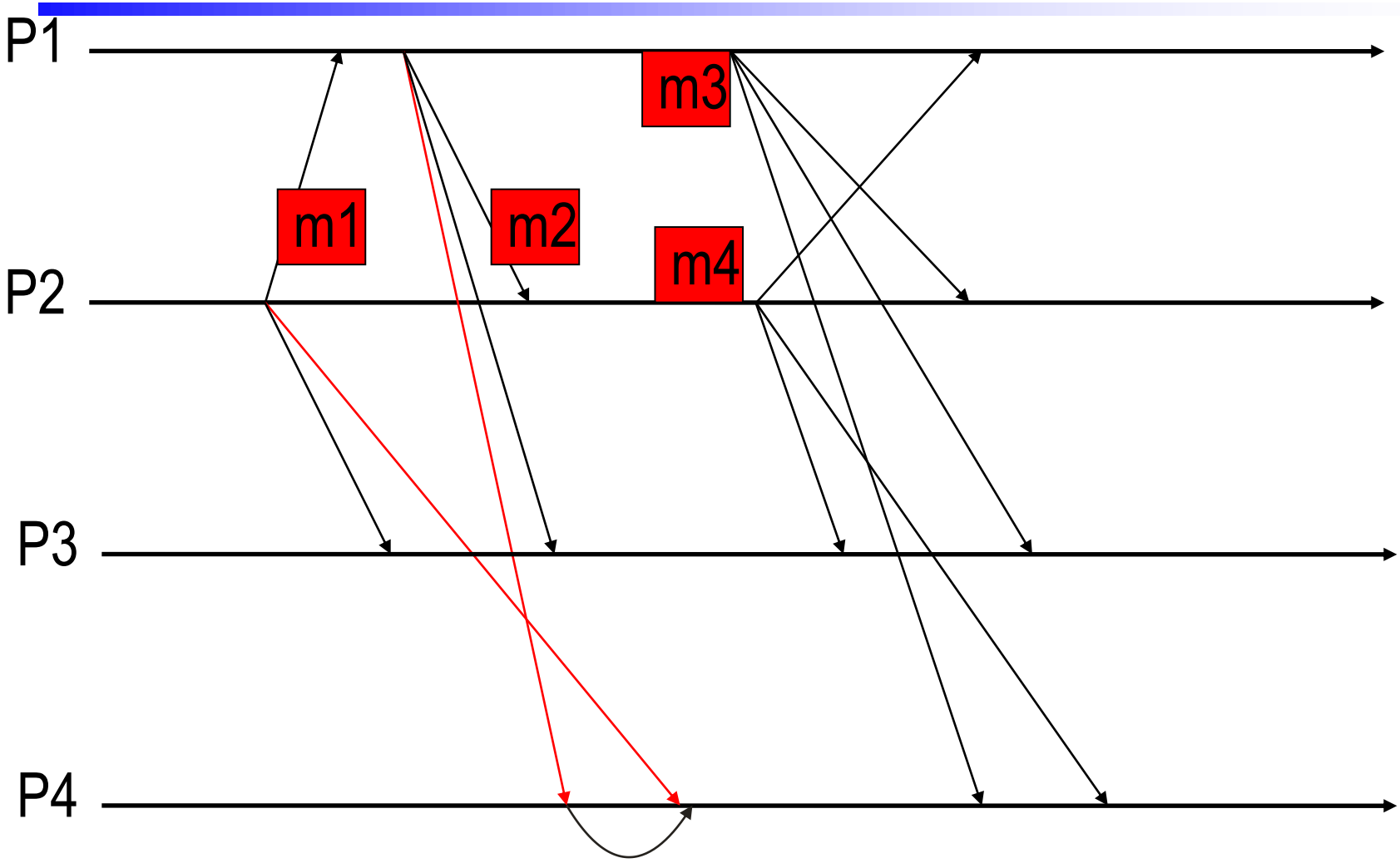
# Orden FIFO





Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

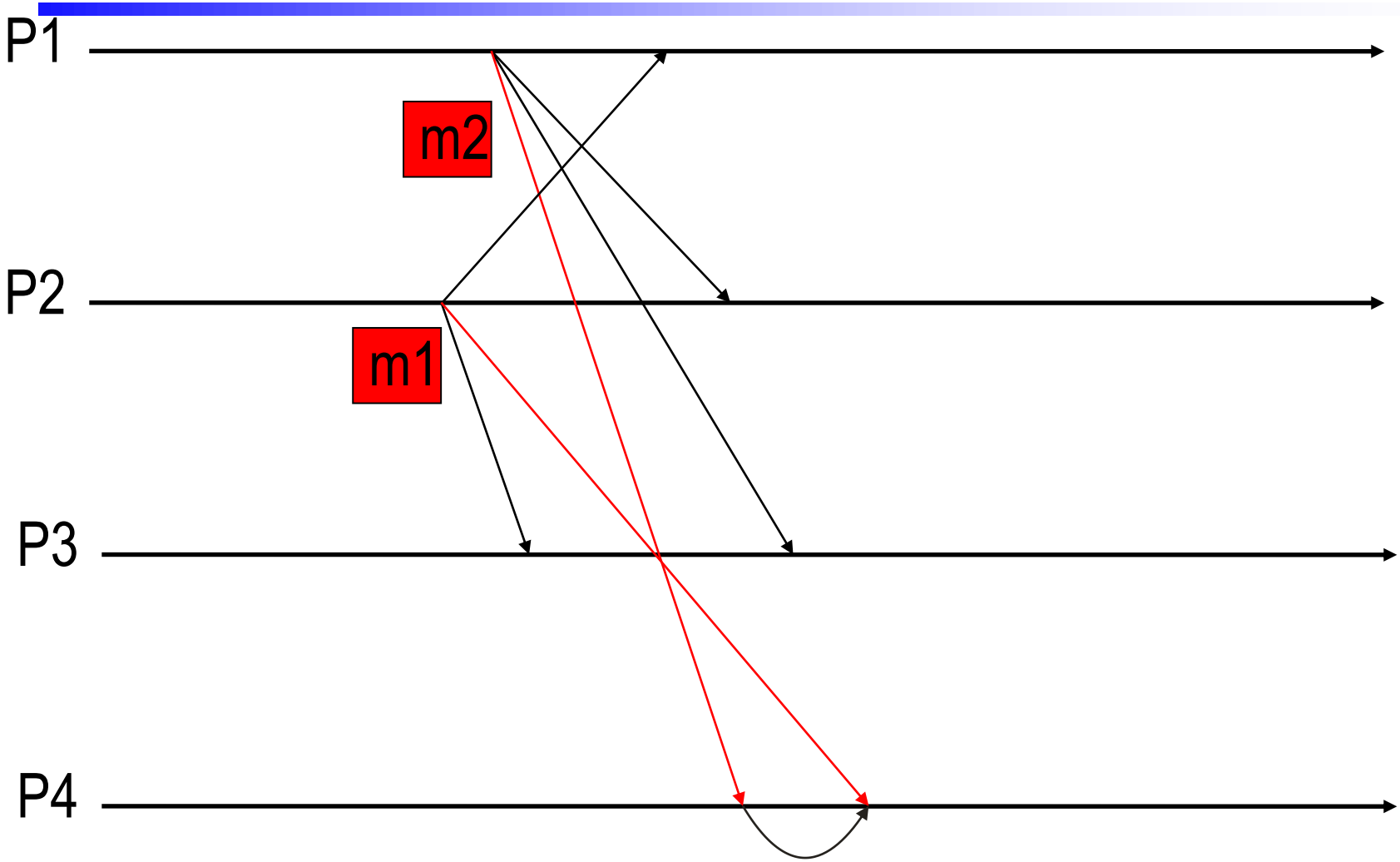
# Orden causal





Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

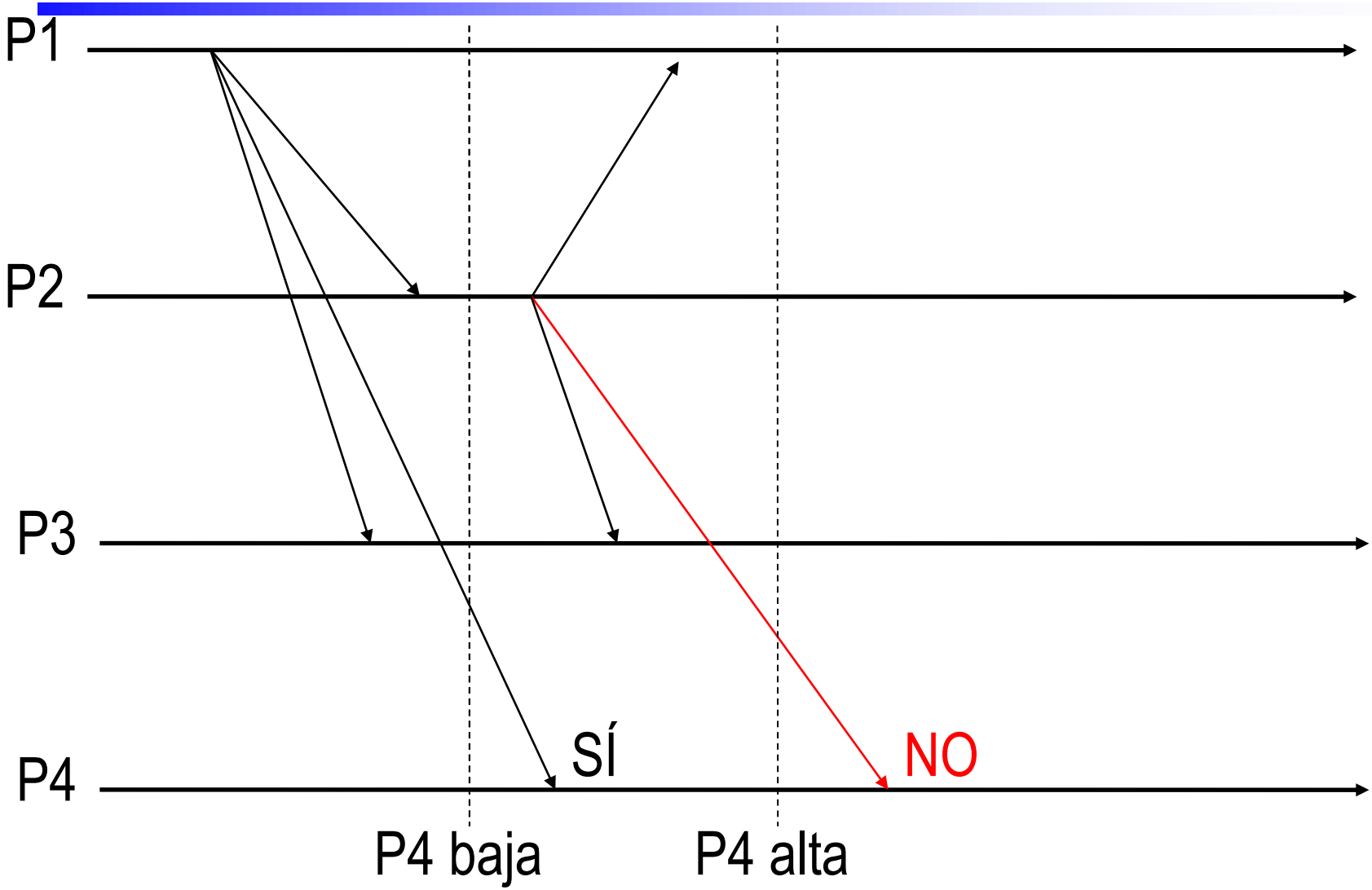
# Orden total





Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Sincronía virtual





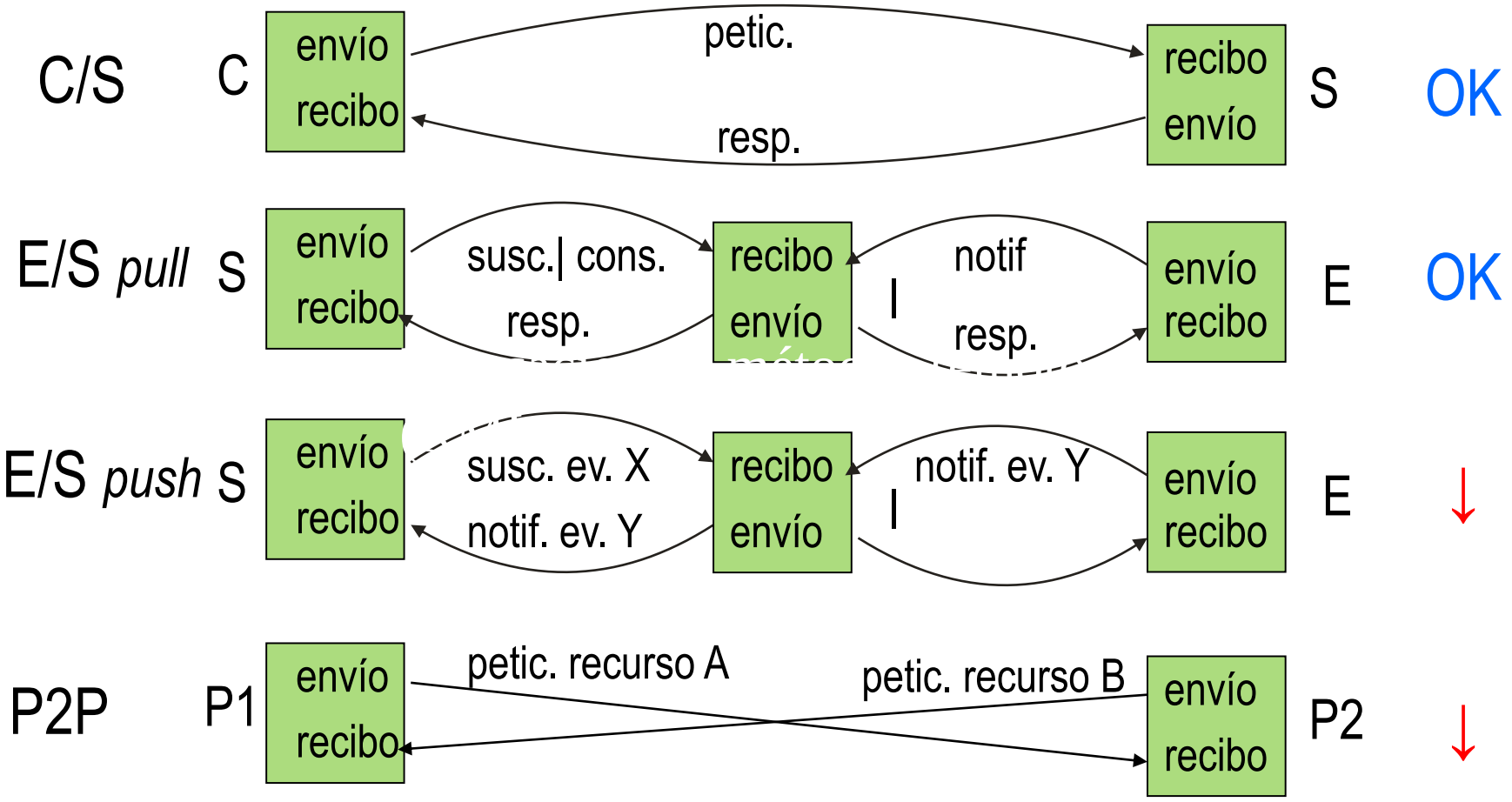
Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Adecuación a arquitecturas del SD

- Paso de mensajes adecuado para cualquier arquitectura
  - Pero cuidado con su asimetría: uno envía y otro recibe
- Cliente/servidor: su asimetría encaja con la del paso mensajes
  - Cliente: envía petición y recibe respuesta
  - Servidor: recibe petición y envía respuesta
- Editor/subscriptor: su asimetría no siempre encaja con p. mens.
  - Si *pull* con intermediario I: buen encaje
    - Su|Ed envían ops. a I y reciben respuestas de I → I siempre pasivo
  - Si *push* con intermediario I: encaje problemático
    - Su envía suscripción(evento X) y espera confirmación de I
    - Pero justo antes I envía notificación de evento Y a Su
    - Soluciones: uso de múltiples puertos y concurrencia en subscriptor
- P2P: arquitectura simétrica
  - ¿Quién envía y quién recibe?


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Adecuación a arquitecturas del SD



# Sistemas Distribuidos

# RPC: Llamada a procedimiento remoto

Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Provisión de servicios en S. no Dist.

**Aplicación**

```
int main(...) {
    ....
    r=op1(p, q, ...);
    .........
}
```

**Biblioteca**

```
t1 op1(ta a, tb b, ...) {
    .....
    return r1;
}
t2 op2(tx x, ty y, ...) {
    .....
    return r2;
}
...........
```


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Provisión de servicios (C/S) en S. Dist.

## Servidor

```
int main(...) {
  Mensaje msj,resp;
  .........
  alta(IDservicio,dir_srv);
  while (TRUE) {
    recepción(&msj,&dir_clie);
    switch(msj.op) {
    case OP1:
      resp.r=op1(msj.arg1,...);
    case OP2:
      resp.r=op2(msj.arg1,...);
    .........
    envío(resp,dir_clie);
  }
}
t1 op1(ta a, tb b, ...) {
}
...........
```

## Cliente

```
int main(...) {
  Mensaje msj,resp;
  ..........
  dir_srv=busca(IDservicio);
  msj.op=OP1;
  msj.arg1=p;
  msj.arg2=q;
  ...........
  envío(msj,dir_srv);
  recepción(&resp, NULL);
  r=resp.r;
  .........
}
```


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Fundamento de las RPC

- Código añadido a provisión de servicios en SD
  - Es independiente de la implementación del cliente y del servidor
  - Sólo depende de la interfaz de servicio
  - Puede generarse automáticamente a partir de la misma
- Objetivo de las RPC
  - Provisión de servicios igual que en sistema no distribuido
  - Sólo hay que programar bibliotecas de servicio y aplicaciones
  - Código restante generado automáticamente
  - Lograr semántica convencional de llamadas a procedimiento en SD


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Provisión de servicios en SD con RPC

## Aplicación

```
int main(...) {
    ....
    r=op1(p, q, ...);
    .........
}
```

```
init() {
  dir_srv=busca(IDservicio);
}
t1 op1(ta a, tb b, ...) {
  Mensaje msj,resp;
  msj.op=OP1;
  msj.arg1=a;
  msj.arg2=b;
  envío(msj,dir_srv);
  recepción(&resp, NULL);
  return resp.r;
}
```

## Biblioteca

```
int main(...) {
  Mensaje msj,resp;
  .........
  alta(IDservicio,dir_srv);
  while (TRUE) {
    recepción(&msj,&dir_clie);
    switch(msj.op) {
    case OP1:
      resp.r=op1(msj.arg1,...);
    case OP2:
      resp.r=op2(msj.arg1,...);
    .........
    envío(resp,dir_clie);
  }
}
```

```
t1 op1(ta a, tb b, ...) {
}
...........
```


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Historia y evolución de las RPC

- Primeras ideas: White 1975; White se enrola en Xerox.
- Desarrollo en Xerox: primer sistema de RPC *Courier* (1981)
- Artículo clásico de Birrel y Nelson en 1984.
- Sun: *Open Network Computing* (1985)
  - RPC de Sun/ONC es la base para servicios (NFS o NIS)
- Apollo (competidora de Sun) crea NCA: RPC de NCA
  - HP compra Apollo; HP miembro de Open Group
- Open Group: DCE (*Distributed Computing Environment* 1990)
  - RPC de DCE basada en NCA
  - RPC de Microsoft (sustento de DCOM) basada en RPC de DCE
- RPC de Sun/ONC vs. RPC de DCE
  - RPC de Sun/ONC menos sofisticada pero más extendida
  - *http://laurel.datsi.fi.upm.es/~ssoo/SOD.dir/practicas/guiarpc.html*

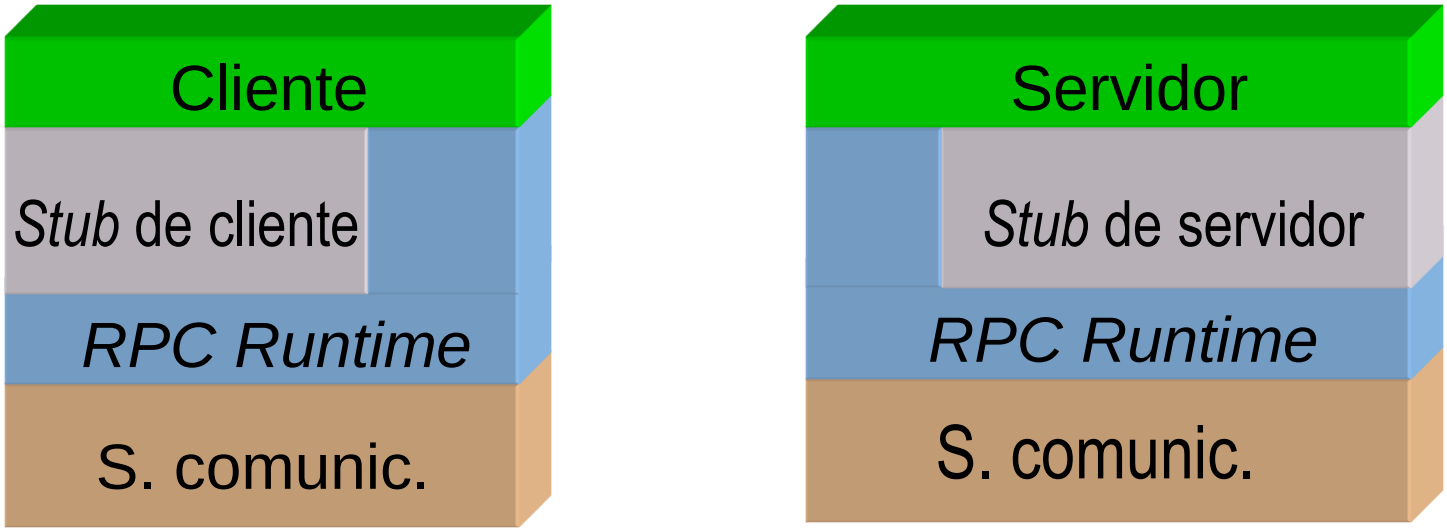

Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Componentes de un sistema de RPC

- En tiempo de construcción del programa:
  - Generador automático de código: Compilador de IDL
    - Definición de interfaz de servicio → Resguardos
    - Heterogeneidad: disponibles para múltiples lenguajes
- En tiempo de ejecución del programa:
  - Resguardos (*stubs*)
    - Módulos que se incluyen en cliente y en servidor
    - Ocultan comunicación dando abstracción de llamada a procedimiento
  - *Runtime* de RPC
    - Capa que proporciona servicios que dan soporte a RPC
      - *Binding*, conversión datos, autenticación, comportamiento ante fallos, ...
    - Uso normalmente por resguardos pero también por aplicación/biblioteca
  - *Binder*: Localización de servicios; alternativas ya presentadas
    - Ámbito no global: ID servicio + dir. servidor (Sun/ONC RPC)
    - Ámbito global: ID servicio
      - Uso sólo de *binder* global vs. *binder* global + *binders* locales (DCE RPC)


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Pila de comunicación en RPC





Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Lenguajes de definición de interfaces

- Los resguardos se generan a partir de la interfaz de servicio
  - ¿Cómo se define esta interfaz?
- Deben de poder definirse entidades tales como:
  - Un tipo interfaz de servicio con un número de versión asociado
  - Prototipos de funciones, constantes, tipos de datos, …
    - Tipo de parámetros: de entrada, de salida o de entrada/salida
  - Directivas específicas (por ejemplo, si una función es idempotente)
  - Sólo definiciones; nunca implementación
- Alternativa: Uso de lenguaje específico vs. uno convencional
  - Permite definición neutral de interfaces
  - Supera limitaciones en expresividad de lenguajes convencionales
  - Pero hay que aprenderlo...
- Procesamiento: Compilador IDL (por ejemplo para C)
  - fichero IDL⟶ fichero .h + resguardo cliente + resguardo servidor

# Sistemas Distribuidos

## RMI: Invocación de método remoto

- Java RMI

Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Modelo de objetos en sis. distribuidos

- Sistemas distribuidos.
  - Aplicaciones inherentemente distribuidas.
  - Se caracterizan por su complejidad.
- Sistemas orientados a objetos.
  - Más cercanos al lenguaje natural.
  - Facilitan el diseño y la programación.


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Objetos-Distribuidos

Características:

- Uso de un *Middleware*: Nivel de abstracción para la comunicación de los objetos distribuidos. Oculta:
  - Localización de objetos.
  - Protocolos de comunicación.
  - Hardware de computadora.
  - Sistemas Operativos.
- Modelo de objetos distribuidos: Describe los aspectos del paradigma de objetos que es aceptado por la tecnología: Herencia, Interfaces, Excepciones, Polimorfismo, ...
- Recogida de basura (*Garbage Collection*): Determina los objetos que no están siendo usados para liberar recursos.


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Ventajas respecto a paso de mensajes

- Paso de mensajes:
  - Procesos fuertemente acoplados
  - Paradigma orientado a datos: No adecuado para aplicaciones muy complejas que impliquen un gran número de peticiones y respuestas entremezcladas.
- Paradigma de objetos distribuidos
  - Mayor abstracción
  - Paradigma orientado a acciones:
    - Hace hincapié en la invocación de las operaciones
    - Los datos toman un papel secundario


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Ventajas respecto a RPC

- Ventajas derivadas al uso de programación orientada a objetos:
  - Encapsulación
  - Reutilización
  - Modularidad
  - Dinamismo


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Objetos distribuidos

- Minimizar las diferencias de programación entre las invocaciones de métodos remotos y las llamadas a métodos locales
- Ocultar las diferencias existentes:
  - Tratamiento del empaquetamiento de los datos (*marshalling*)
  - Sincronización de los eventos
  - Las diferencias deben quedar ocultas en la **arquitectura**


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Sistemas Distribuidos

## Java RMI

Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Java RMI

Java: *Write Once, Run Anywhere*

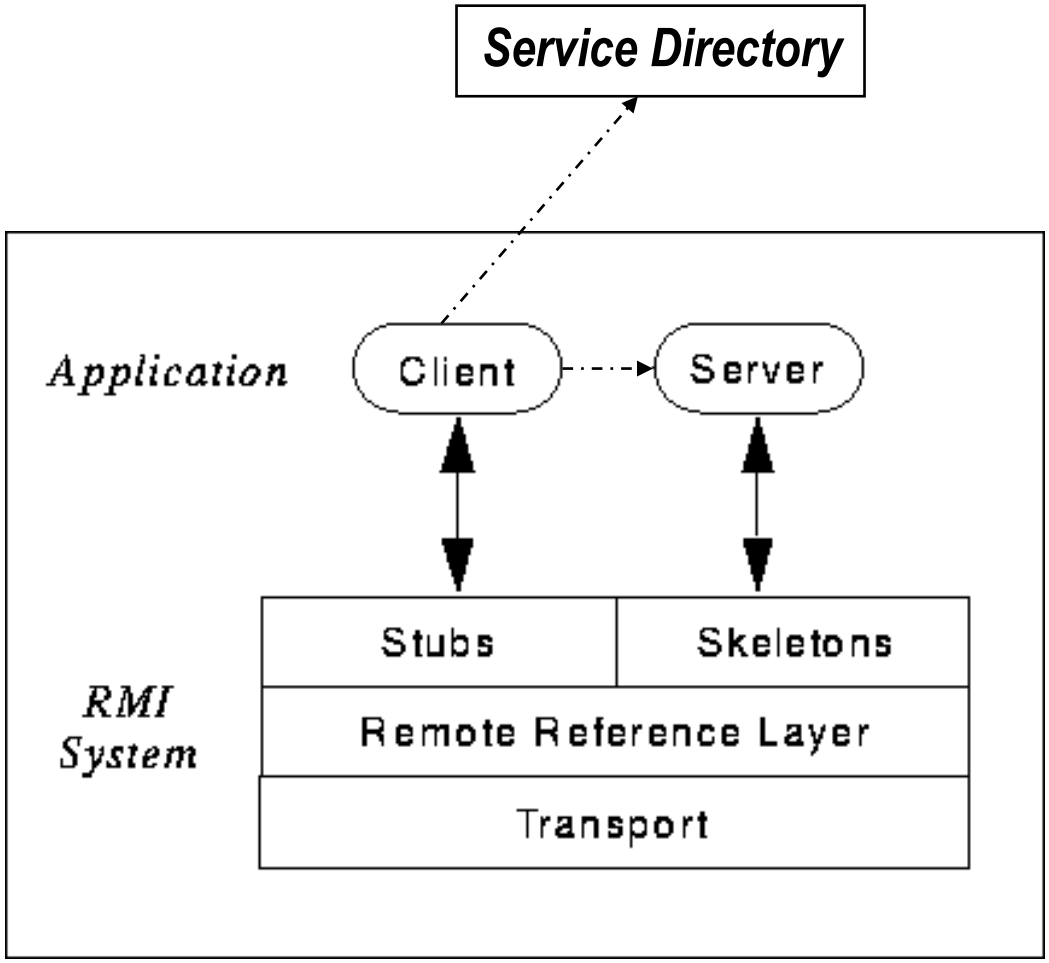
Java RMI extiende el modelo Java para la filosofía *"Run Everywhere"*

Guía para la programación en Java RMI:

*http://laurel.datsi.fi.upm.es/~ssoo/SD.dir/practicas/guiarmi.html*


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Arquitectura de Java RMI





Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Arquitectura de Java RMI

***Nivel de resguardo/esqueleto*** (*proxy/skeleton*) que se encarga del aplanamiento (serialización) de los parámetros

*proxy*: resguardo local. Cuando un cliente realiza una invocación remota, en realidad hace una invocación de un método del resguardo local.

Esqueleto (*skeleton*): recibe las peticiones de los clientes, realiza la invocación del método y devuelve los resultados.

***Nivel de gestión de referencias remotas***: interpreta y gestiona las referencias a los objetos de servicio remoto

***Nivel de transporte***: se encarga de las comunicaciones y de establecer las conexiones necesarias. Basada en TCP (orientada a conexión)


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Servicio de directorios

Se pueden utilizar diferentes servicios de directorios para registrar un objeto distribuido

- Ejemplo: JNDI (Java Naming and Directory Interface)

El registro RMI, *rmiregistry*, es un servicio de directorios sencillo proporcionado por SDK (Java Software Development Kit)


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Java RMI

El soporte para RMI en Java está basado en las interfaces y clases definidas en los paquetes:

- *java.rmi*
- *java.rmi.server*

Características de Java RMI:

- Se basa en una interfaz remota Java (hereda de la clase Java *Remote*).
- Es necesario tratar mayor número de excepciones que en el caso de invocación de métodos locales
  - Errores en la comunicación entre los procesos (fallos de acceso, fallos de conexión)
  - Problemas asociados a la invocación de métodos remotos (no encontrar el objeto, el resguardo o el esqueleto)
  - Los métodos deben especificar la excepción *RemoteException*.


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Ejemplo de interfaz remota Java

```
public interface InterfazEj extends
   java.rmi.Remote
{
   public String metodoEj1()
       throws java.rmi.RemoteException;
   public int metodoEj2(float param)
       throws java.rmi.RemoteException;
}
```


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Java RMI

Localización de objetos remotos:

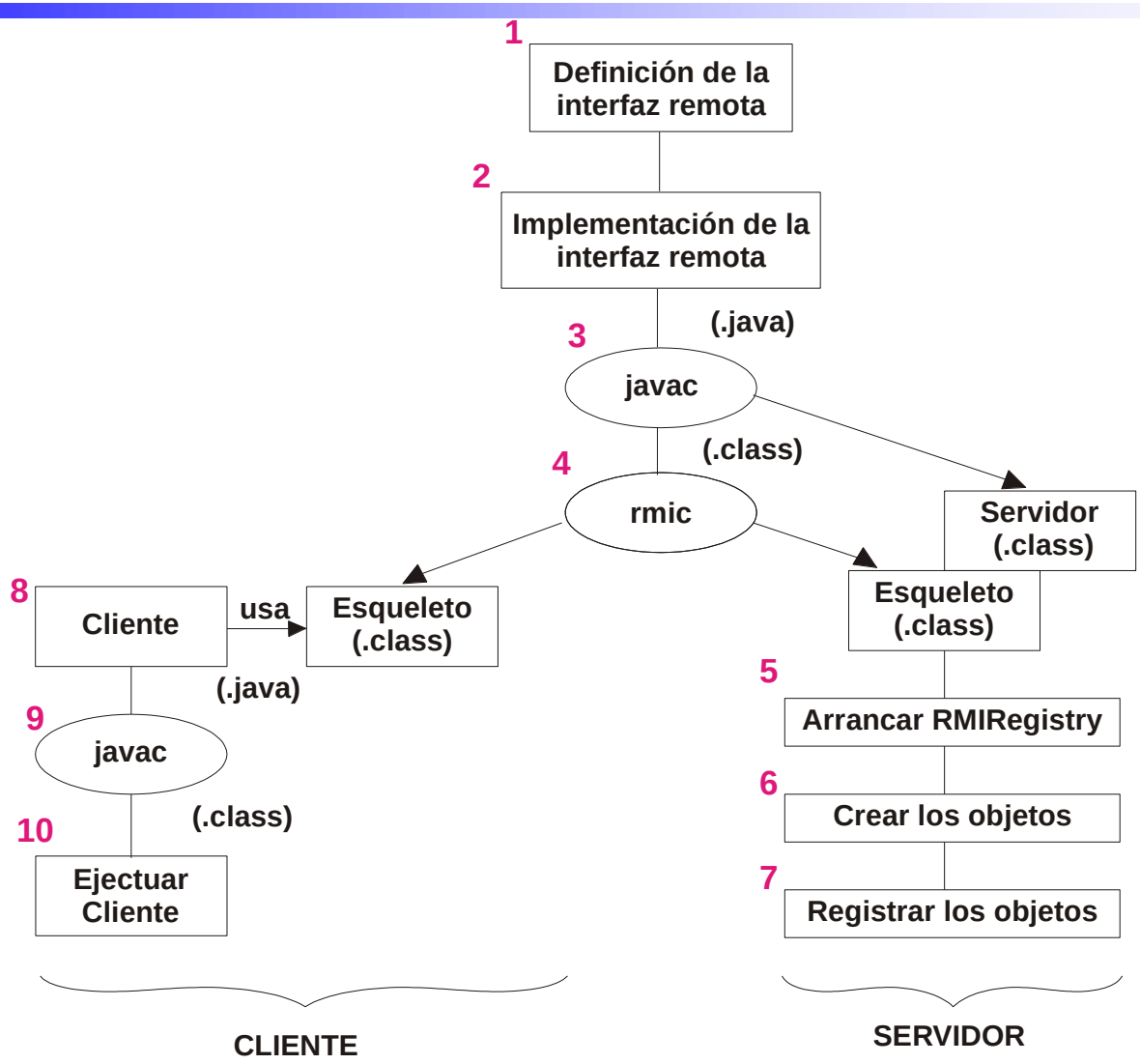
Servidor de nombres: *java.rmi.Naming*

Ejemplo:

```
Cuenta cnt = new CuentaImpl();
String url = "rmi://java.Sun.COM/cuenta";
// enlazamos una url a un objeto remoto
java.rmi.Naming.bind(url, cnt);
      ....
// búsqueda de la cuenta
cnt=(Cuenta)java.rmi.Naming.lookup(url);
```


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

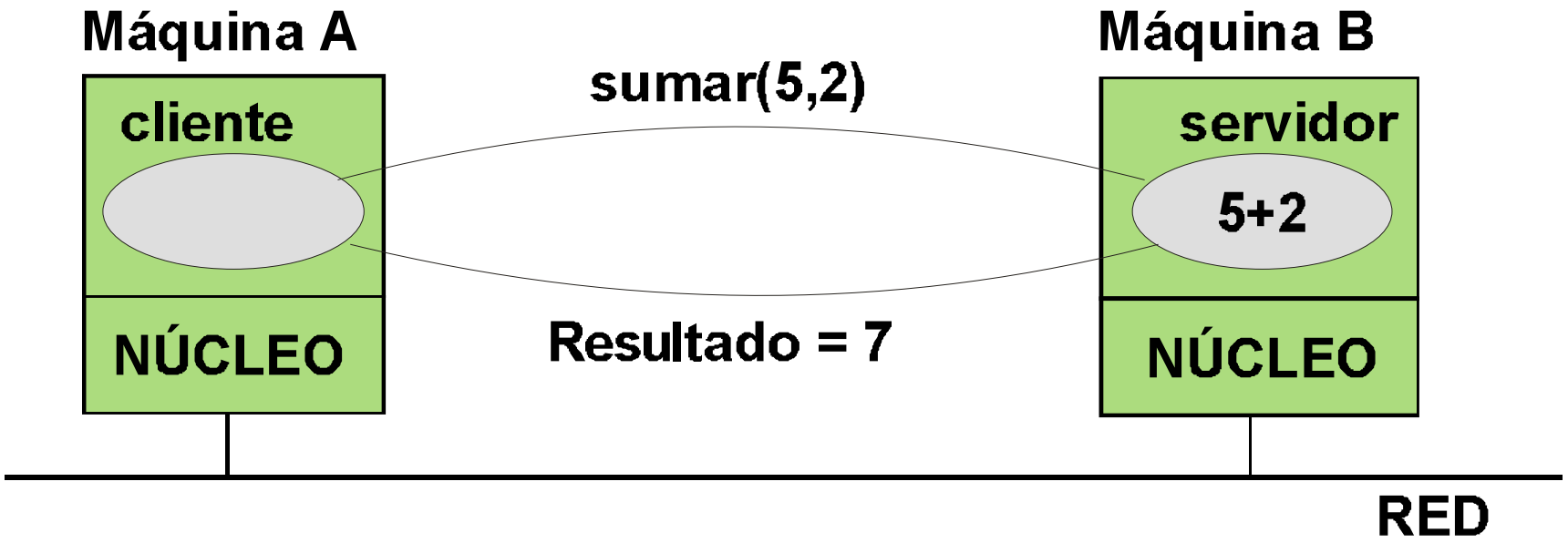
# Desarrollo de Aplicaciones RMI





Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Ejemplo





Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Modelización de la interfaz remota (Sumador)

```
public interface Sumador extends java.rmi.Remote
{

   public int sumar(int a, int b)
        throws java.rmi.RemoteException;

}
```


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Clase que implementa la interfaz (SumadorImpl)

```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
public class SumadorImpl extends UnicastRemoteObject
   implements Sumador {
    public SumadorImpl(String name) throws RemoteException {
        super();
       try {
        System.out.println("Rebind Object " + name);
            Naming.rebind(name, this);
       } catch (Exception e){
        System.out.println("Exception: " + e.getMessage());
                  e.printStackTrace();
       }
   }
    public int sumar (int a, int b) throws RemoteException {
        return a + b;  } }
```


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Código del servidor (SumadorServer)

```
import java.rmi.*;
import java.rmi.server.*;
public class SumadorServer  {
   public static void main (String args[]) {
      try {
            SumadorImpl misuma = new
                  SumadorImpl("rmi://localhost:1099"
                  +"/MiSumador");
            } catch(Exception e) {
            System.err.println("System exception" +
  e);
      }
   }
}
```


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Registro del servicio

Antes de arrancar el cliente y el servidor, se debe arrancar el programa *rmiregistry* en el servidor para el servicio de nombres. El puerto que utiliza el *rmiregistry* por defecto es el 1099.

```
rmiregistry [port_number]
```

El método *rebind* es utilizado normalmente en lugar del método *bind*, porque garantiza que si un objeto remoto se registró previamente con dicho nombre, el nuevo objeto reemplazará al antiguo.

El método *rebind* almacena en el registro una referencia al objeto con un URL de la forma:

```
rmi://<nombre máquina>:<número puerto>/<nombre referencia>
```


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Código en el cliente (SumadorClient)

```
import java.rmi.registry.*;
import java.rmi.server.*;
import java.rmi.*;
public class SumadorClient {
   public static void main(String args[]){
   int res = 0;
   try {
        System.out.println("Buscando Objeto ");
        Sumador misuma = (Sumador)Naming.lookup("rmi://" +
   args[0] +                                    "/" +"MiSumador");
         res = misuma.sumar(5, 2);
        System.out.println("5 + 2 = " + res);
    }
    catch(Exception e){
              System.err.println(" System exception");     }
    System.exit(0);     } }
```


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Búsqueda

Cualquier programa que quiera instanciar un objeto remoto debe realizar una búsqueda de la siguiente forma:

```
Sumador misuma = (Sumador)Naming.lookup("rmi://" +  args[0] +
"/"                +"MiSumador");
```

El método *lookup* devuelve una referencia remota a un objeto que implementa la interfaz remota.

El método *lookup* interactúa con *rmiregistry*.


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

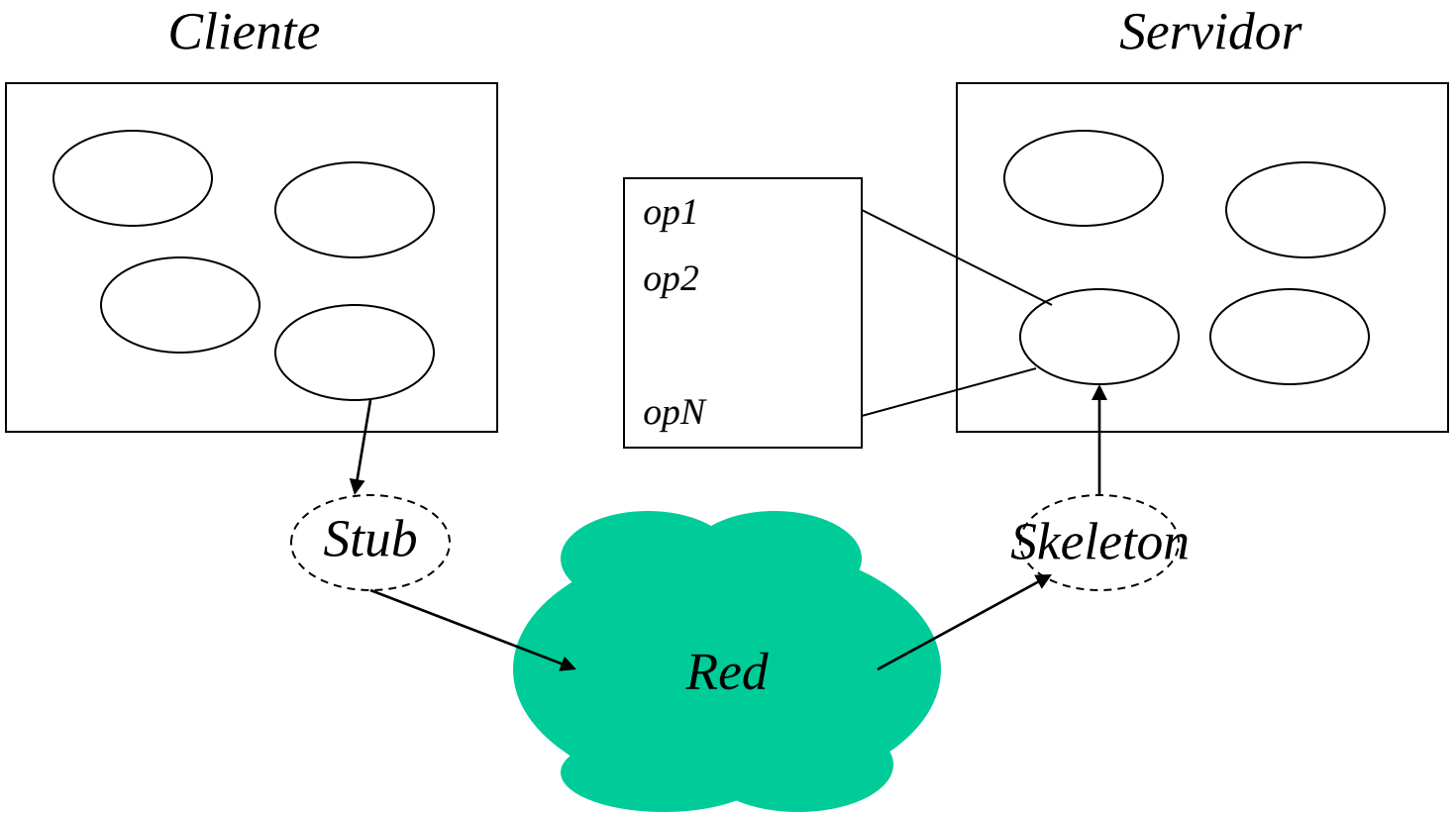
# Pasos

Java RMI:

- Enlace a un nombre: bind(), rebind()
- Encontrar un objeto y obtener su referencia: lookup()
- refObj.nombre_met()


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Cuadro general



# ¿Cómo se ejecuta?

**Compilación**

javac Sumador.java
javac SumadorImpl.java
javac SumadorClient.java
javac SumadorServer.java

**Generación de los esqueletos (No necesario a partir de Java 1.5)**

rmic SumadorImpl

**Ejecución del programa de registro de RMI**

rmiregistry <número puerto>
*Por defecto, en número de puerto es el 1099*

**Ejecución del servidor**

java SumadorServer

**Ejecución del cliente**

java SumadorCliente <host-del-servidor>


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# *Callback* de cliente

Hay escenarios en los que los servidores deben notificar a los clientes la ocurrencia de algún evento. Ejemplo: chat.

- Problema: llamada a método remoto es unidireccional

Posibles soluciones:

- Sondeo (*polling)*: Cada cliente realiza un sondeo al servidor, invocando repetidas veces un método, hasta que éste devuelva un valor *true*.
  - Problema: Técnica muy costosa (recursos del sistema)
- ***Callback***: Cada cliente interesado en la ocurrencia de un evento se registra a sí mismo con el servidor, de modo que el servidor inicia una invocación de un método remoto del cliente cuando ocurra dicho evento.
  - Las invocaciones de los métodos remotos se convierten en bidireccionales


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Extensión de la parte cliente para *callback* de cliente

El cliente debe proporcionar una interfaz remota: Interfaz remota de cliente

```
public interface CallbackClient extends java.rmi.Remote {
    public String notificame (String mensaje) throws
   java.rmi.RemoteException;
}
```

Es necesario implementar la interfaz remota de cliente, de forma análoga a la interfaz de servidor (CallbackClientImpl)

En la clase cliente se debe añadir código para que instancie un objeto de la implementación de la interfaz remota de cliente y que se registre para callback (método implementado por el servidor):

```
CallbackServer cs = (CallbackServer) Naming.lookup(URLregistro);
CallbackClient objCallback = new CallbackClientImpl();
cs.registrarCallback(objCallback);
```


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Extensión de la parte servidora para *callback* de cliente

Añadir el método remoto para que el cliente se registre para *callback*

```
public void registrarCallback (CallbackClient
objCallbackClient) throws java.rmi.RemoteException;
```

Se puede proporcionar un método `eliminarRegistroCallback` para poder cancelar el registro

Ambos métodos modifican una estructura común (por ejemplo, un objeto Vector) que contiene referencias a los *callbacks* de clientes. Se utilizan métodos *synchronized* para acceder a la estructura en exclusión mutua.

# Descarga dinámica de resguardo

- Mecanismo que permite que los clientes obtengan dinámicamente los resguardos necesarios
  - Elimina la necesidad de copia de la clase del resguardo en la máquina cliente
  - Se transmite bajo demanda desde un servidor web a la máquina cliente (Similar a la descarga de los applets)
- El servidor exporta un objeto a través del registro RMI (registro de una referencia remota al objeto mediante nombre simbólico) e indica el URL donde se almacena la clase resguardo.
- La clase resguardo descargada no es persistente
  - Se libera cuando la sesión del cliente finaliza


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Comparativa RMI vs Sockets

Los sockets están más cercanos al sistema operativo, lo que implica una menor sobrecarga de ejecución.

RMI proporciona mayor abstracción, lo que facilita el desarrollo de software. RMI es un buen candidato para el desarrollo de prototipos.

Los sockets suelen ser independientes de plataforma y lenguaje.


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Sistemas Distribuidos

## Aspectos internos de la comunicación

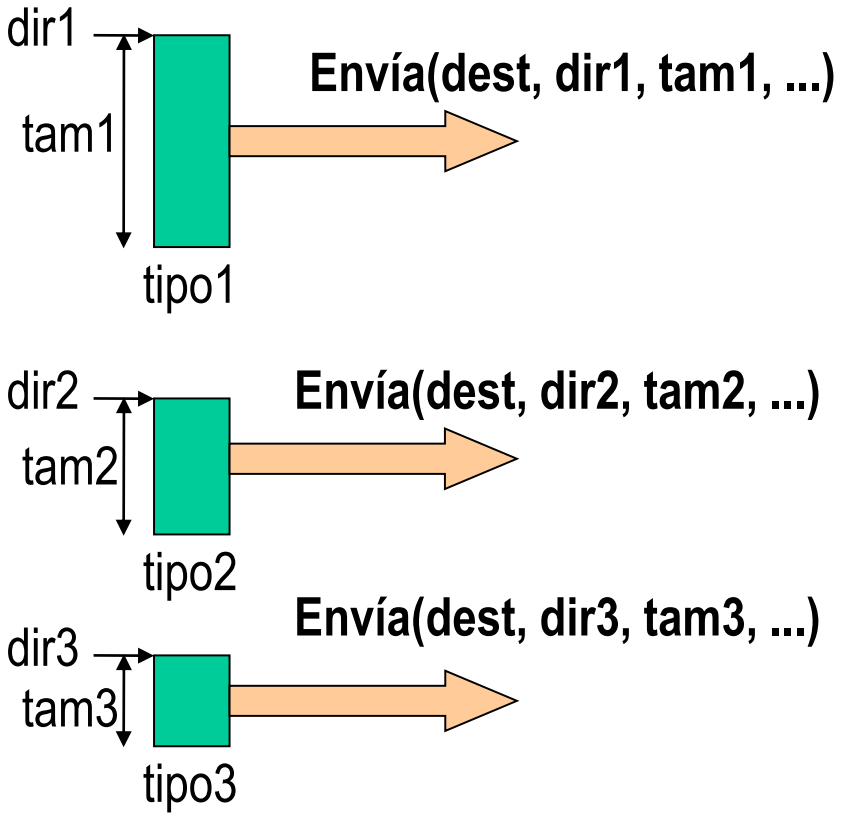
Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Zero-Copy

- Reducir al mínimo (≈ a cero) copias entre zonas de memoria
- App, S.O. y hardware de comunicación colaboran para intentar:
  - Enviar info. desde *buffer* app emisora al de la receptora sin copias
- Aplicación debería evitar copias entre sus variables:
  - *strcpy(m.ncola, ncola);*
- Entre *buffers* de usuario y del SO y entre *buffers* del propio SO
- Reduciendo nº llamadas al sistema usadas para la transmisión
  - Sobrecarga de cambios de modo usuario a sistema y viceversa
  - Envíos separados pueden acabar en mensajes independientes
- Ejemplos: http://laurel.datsi.fi.upm.es/~ssoo/SD.dir/zerocopy.tgz
  - Envío de datos dispersos
  - Envío de un fichero
  - Escenario combinado: servidor web
    - Envío de cabecera + envío del contenido del fichero pedido


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

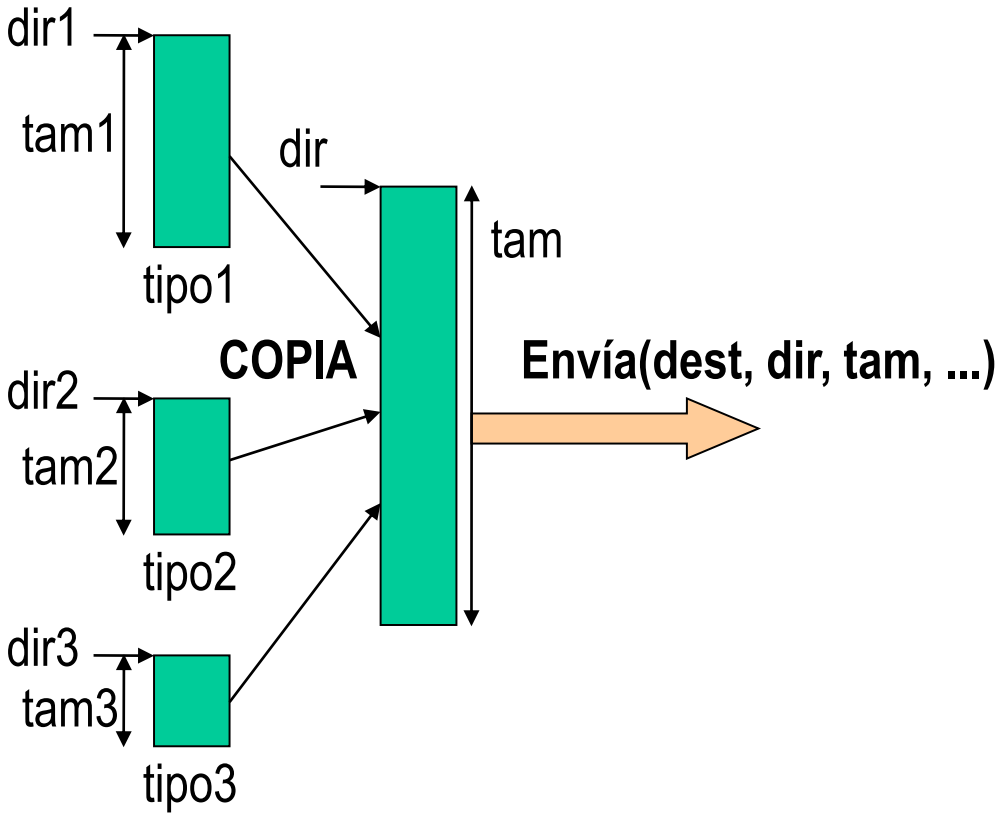
# Datos dispersos: Envío múltiple




sobrecarga de llamadas + fragmentación de mensajes


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

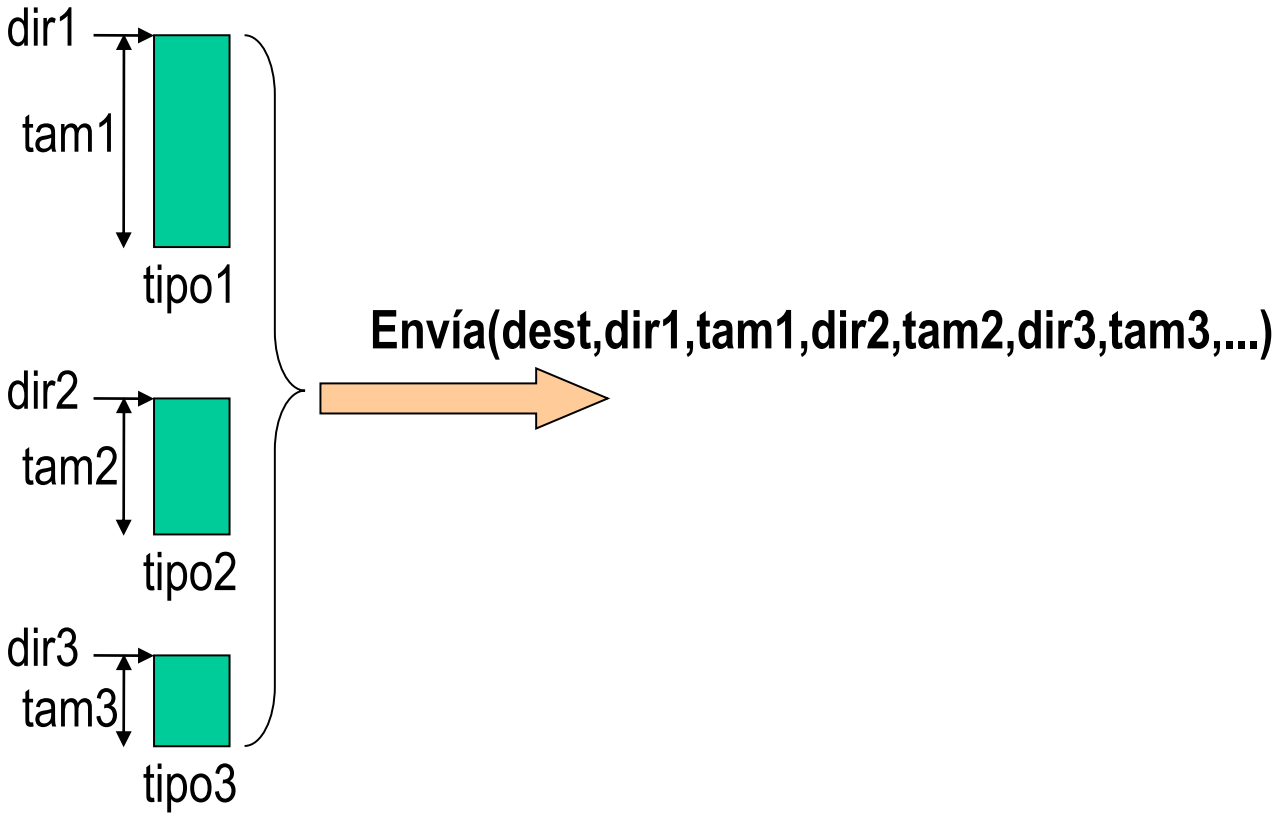
# Datos dispersos: Envío con copia




sobrecarga por copias


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

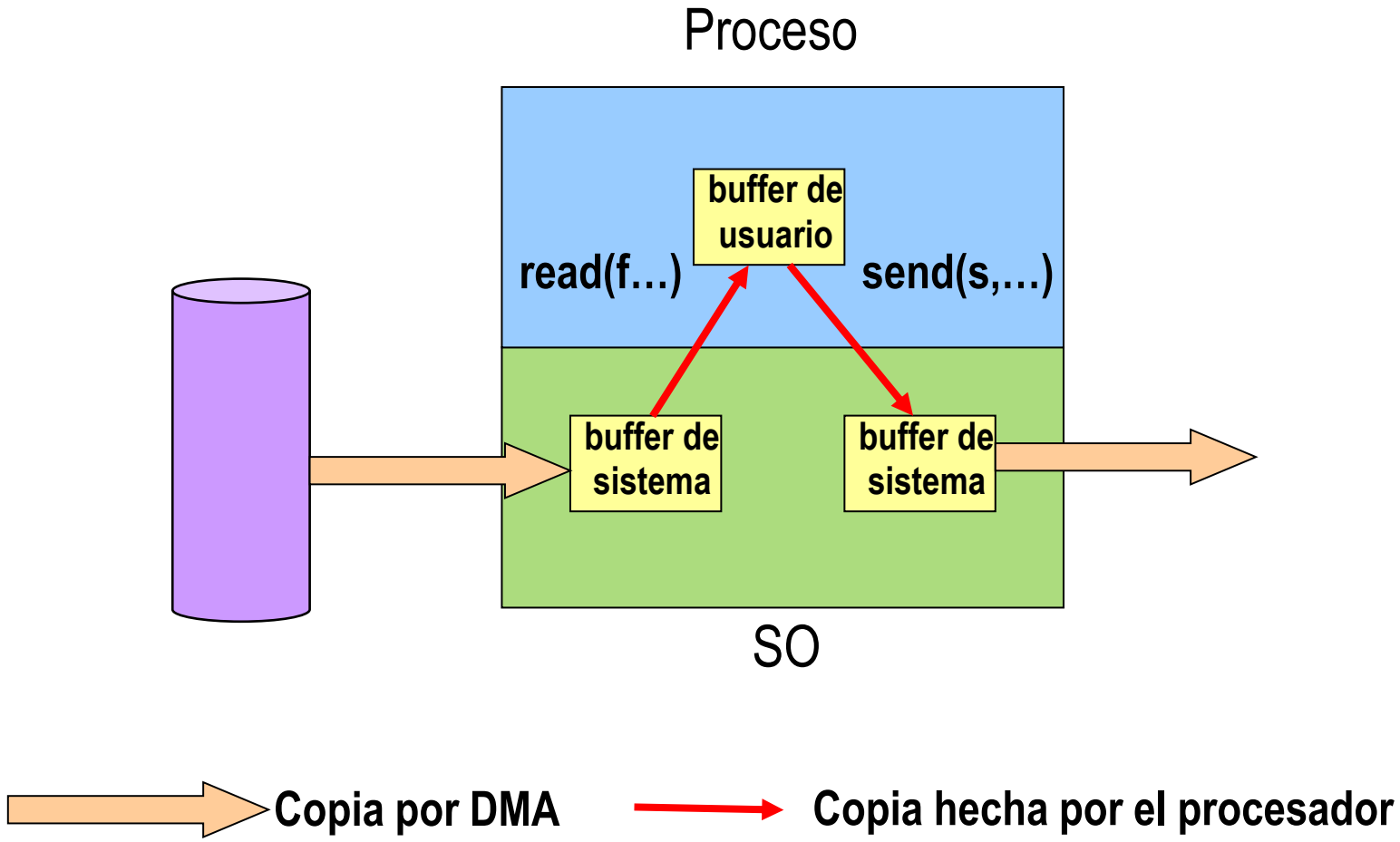
# Datos dispersos: Envío *gather*




Uso de *writev*


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

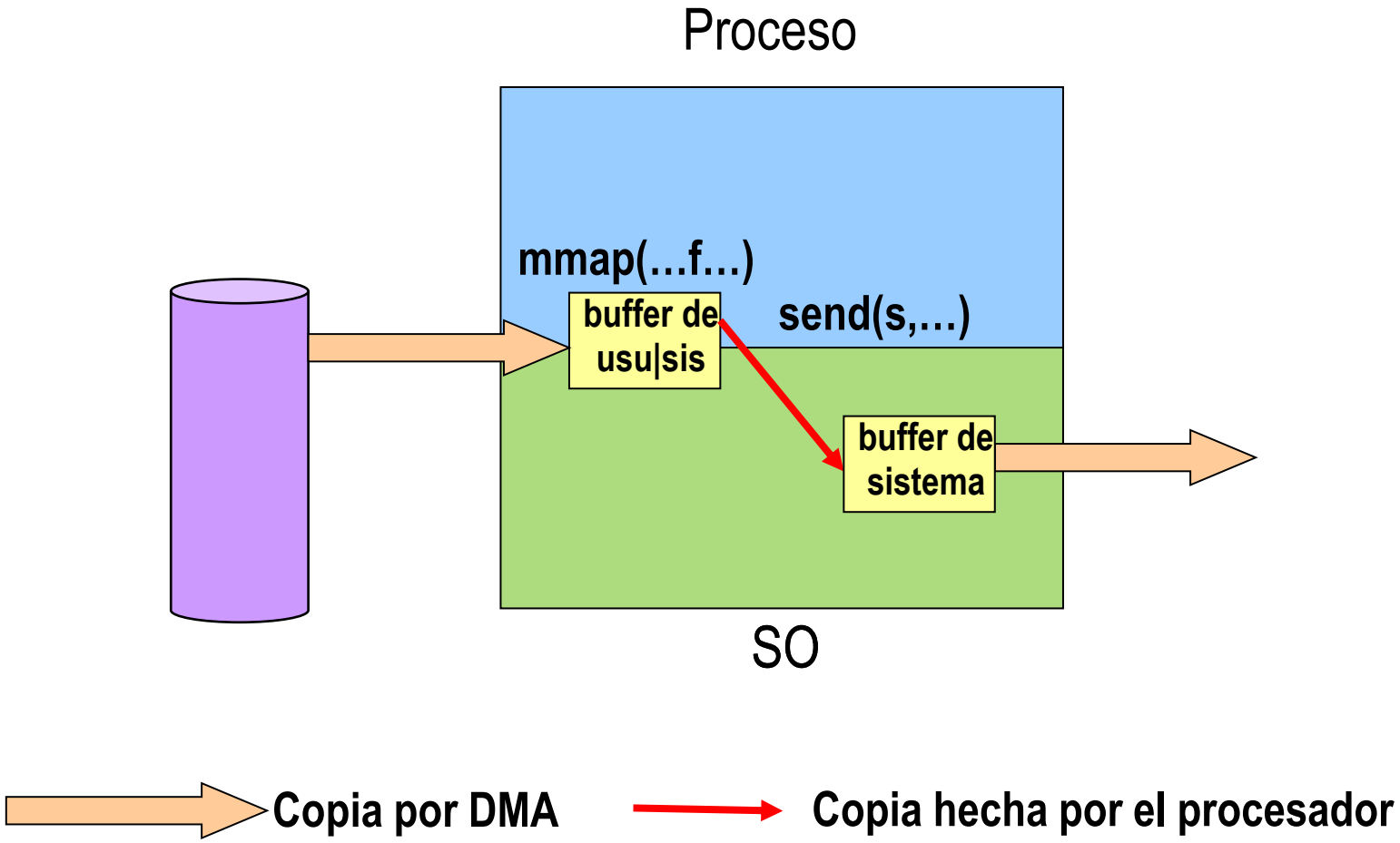
# Envío convencional de fichero





Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

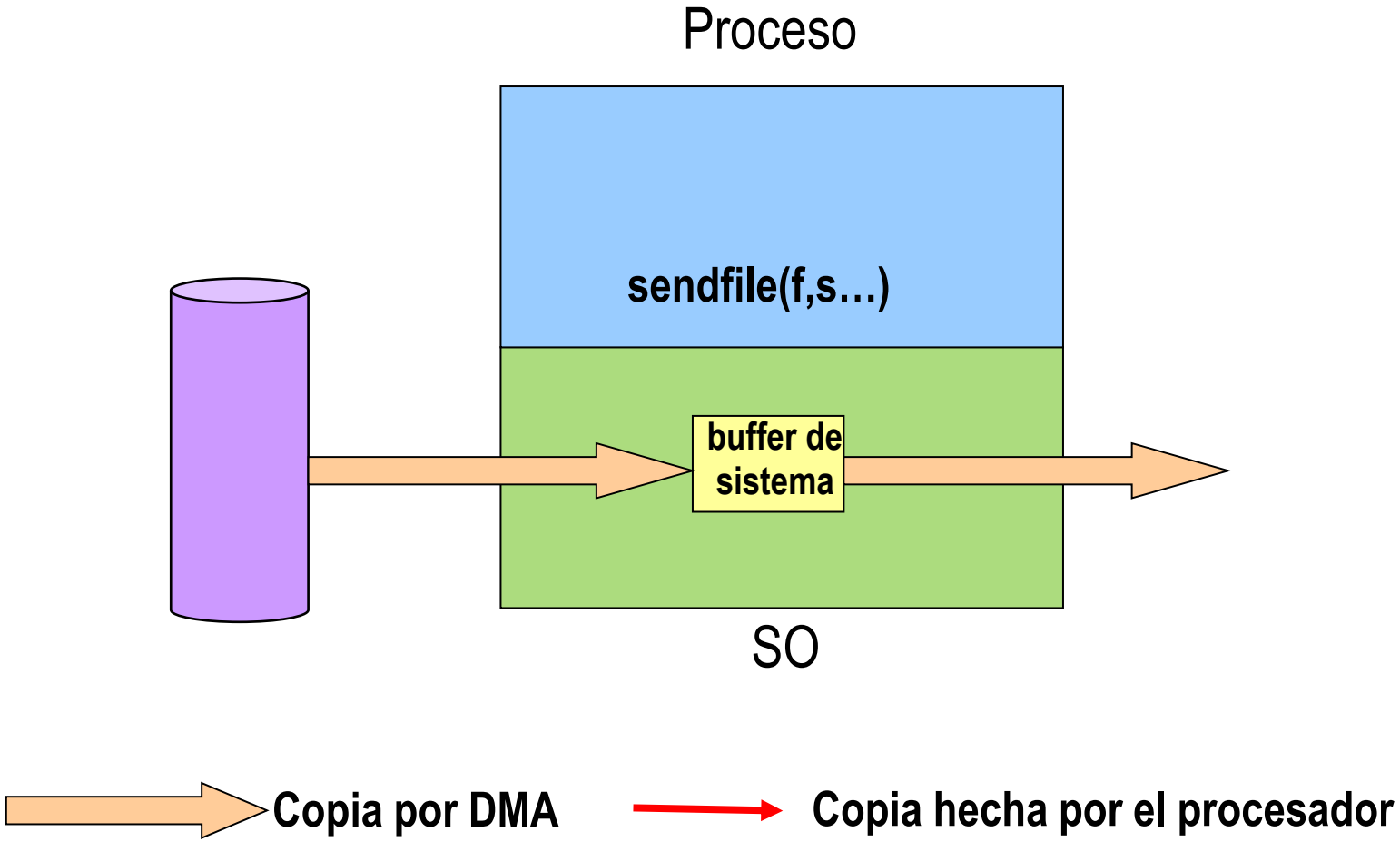
# Envío con proyección de fichero





Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Envío *zero-copy* de fichero





Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Fake Web Server: read y send

```
static int crear_socket(int puerto);
static int recibir_peticion(int s, char *pagina);
static void preparar_cabecera(int tam_fich, char *cabecera);
static int abrir_fichero(char *fich, int *tam_fich, int *tam_acceso_recom);

int main(int argc, char *argv[]) {
        int s, f, s_conec, leido;
        int tam_fich, tam_acceso_recom;
        struct tms to, tn;
        clock_t t0, t1;
        char cabecera[TAM];
        char pagina[PATH_MAX];

        if (argc!=2) {
                fprintf(stderr, "Uso: %s puerto\n", argv[0]); return 1;
        }
        s=crear_socket(atoi(argv[1]));
        while (1) {
                s_conec=recibir_peticion(s, pagina);
                f=abrir_fichero(pagina, &tam_fich, &tam_acceso_recom);
                preparar_cabecera(tam_fich, cabecera);
                char buf[tam_acceso_recom];

                t0=times(&to);
                send(s_conec, cabecera, strlen(cabecera), MSG_MORE);
                while ((leido=read(f, buf, tam_acceso_recom))>0)
                        send(s_conec, buf, leido, MSG_MORE);
                t1=times(&tn);
                close(s_conec); close(f);
                printf("Real %ld Usuario %ld Sistema %ld\n", t1-t0,
                        tn.tms_utime-to.tms_utime,
                        tn.tms_stime-to.tms_stime);
        }
        return 0;
}
```

50,0-1 14%



Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Fake Web Server: mmap y writev

```
static int crear_socket(int puerto);
static int recibir_peticion(int s, char *pagina);
static void preparar_cabecera(int tam_fich, char *cabecera);
static int abrir_fichero(char *fich, int *tam_fich, int *tam_acceso_recom);

int main(int argc, char *argv[]) {
        int s, f, s_conec;
        int tam_fich, tam_acceso_recom;
        struct tms to, tn;
        clock_t t0, t1;
        char cabecera[TAM];
        char pagina[PATH_MAX];
        void *p;
        struct iovec iov[2];

        if (argc!=2) {
                fprintf(stderr, "Uso: %s puerto\n", argv[0]); return 1;
        }
        s=crear_socket(atoi(argv[1]));
        while (1) {
                s_conec=recibir_peticion(s, pagina);
                f=abrir_fichero(pagina, &tam_fich, &tam_acceso_recom);
                preparar_cabecera(tam_fich, cabecera);

                t0=times(&to);
                p = mmap(NULL, tam_fich, PROT_READ, MAP_PRIVATE, f, 0);
                madvise(p, tam_fich, MADV_SEQUENTIAL);
                iov[0].iov_base = cabecera; iov[0].iov_len = strlen(cabecera);
                iov[1].iov_base = p; iov[1].iov_len = tam_fich;
                writev(s_conec, iov, 2);
                t1=times(&tn);
                close(s_conec); close(f);
                munmap(p, tam_fich);
                printf("Real %ld Usuario %ld Sistema %ld\n", t1-t0,
                        tn.tms_utime-to.tms_utime,
                        tn.tms_stime-to.tms_stime);
                                                                17,17         15%
```



Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Fake Web Server: sendfile

```
static int crear_socket(int puerto);
static int recibir_peticion(int s, char *pagina);
static void preparar_cabecera(int tam_fich, char *cabecera);
static int abrir_fichero(char *fich, int *tam_fich, int *tam_acceso_recom);

int main(int argc, char *argv[]) {
        int s, f, s_conec;
        int tam_fich, tam_acceso_recom;
        struct tms to, tn;
        clock_t t0, t1;
        char cabecera[TAM];
        char pagina[PATH_MAX];

        if (argc!=2) {
                fprintf(stderr, "Uso: %s puerto\n", argv[0]); return 1;
        }
        s=crear_socket(atoi(argv[1]));
        while (1) {
                s_conec=recibir_peticion(s, pagina);
                f=abrir_fichero(pagina, &tam_fich, &tam_acceso_recom);
                preparar_cabecera(tam_fich, cabecera);

                t0=times(&to);
                send(s_conec, cabecera, strlen(cabecera), MSG_MORE);
                sendfile(s_conec, f, NULL, tam_fich);
                t1=times(&tn);
                close(s_conec); close(f);
                printf("Real %ld Usuario %ld Sistema %ld\n", t1-t0,
                        tn.tms_utime-to.tms_utime,
                        tn.tms_stime-to.tms_stime);
        }

        return 0;
}
static int crear_socket(int puerto) {
        int s;
                                                                    48,10-17      15%
```



Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Integridad de los mensajes

- ¿Se mantiene la integridad de los mensajes?
  - Nunca se entregan restos de mensajes ni dos mensajes juntos
- Sí en sockets datagrama: *buffer* < mensaje, pérdida del resto
- No en sockets *stream*: en destino se funden los mensajes
  - Lectura puede obtener partes de varios mensajes
  - Recepción de *N* bytes puede obtener cualquier nº de bytes ≤ *N*
    - Incluso aunque se hayan enviado mensajes también de *N* bytes
      - http://laurel.datsi.fi.upm.es/~ssoo/sockets/7_recepcion_completa/
  - ¿Cómo asegurar que se reciben *N* bytes?
    - Bucle que repite la llamada de recepción y va acumulando hasta *N*
    - Uso del *flag MSG_WAITALL* en *recv*
    - Uso de funciones de la biblioteca de un lenguaje (p.e. *fread* en C)
  - Para gestionar mensajes de tamaño variable se puede:
    - Enviar longitud antes del mensaje o usar un separador
    - Usar 1 conexión/mensaje y hacer *shutdown* de socket de envío


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Serialización/*marshaling* de datos

- Emisor y receptor misma interpretación de información
  - Misma cuestión, y soluciones, para lector y escritor de un fichero
- Procesadores, lenguajes, compiladores difieren en:
  - Orden de bytes en tipos numéricos (*endian*)
  - Tamaño de datos numéricos (en C: ¿tamaño de int, long,…?)
  - *Strings* (con longitud vs. carácter terminador (¿qué carácter?))
  - Formatos de texto (ISO-8859-1, UTF-8…)
  - Organización estructuras datos (compactación, alineamientos,…),…
- Se necesitan "**serializar**" los datos para enviar/almacenar
  - Asegurando misma interpretación en sistema heterogéneo
  - Eficientemente (en tiempo de *serialización*, en uso de red/disco,…)
- Formato de *serialización*: cómo se transmiten/almacenan datos
  - Secuencia de bits que representan cada dato
  - Texto vs. binario; mejor estándar; ¿envío info de tipos?¿y de campos?


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Serialización con metainformación

datos a enviar

| Campo | Tipo | Valor |
|---|---|---|
| x | int | 6 |
| y | int | 8 |

solo valores


Receptor conoce de forma implícita la metainformación

con tipos


y nombres


Deserialización genérica

Más sobrecarga vs. flexibilidad


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

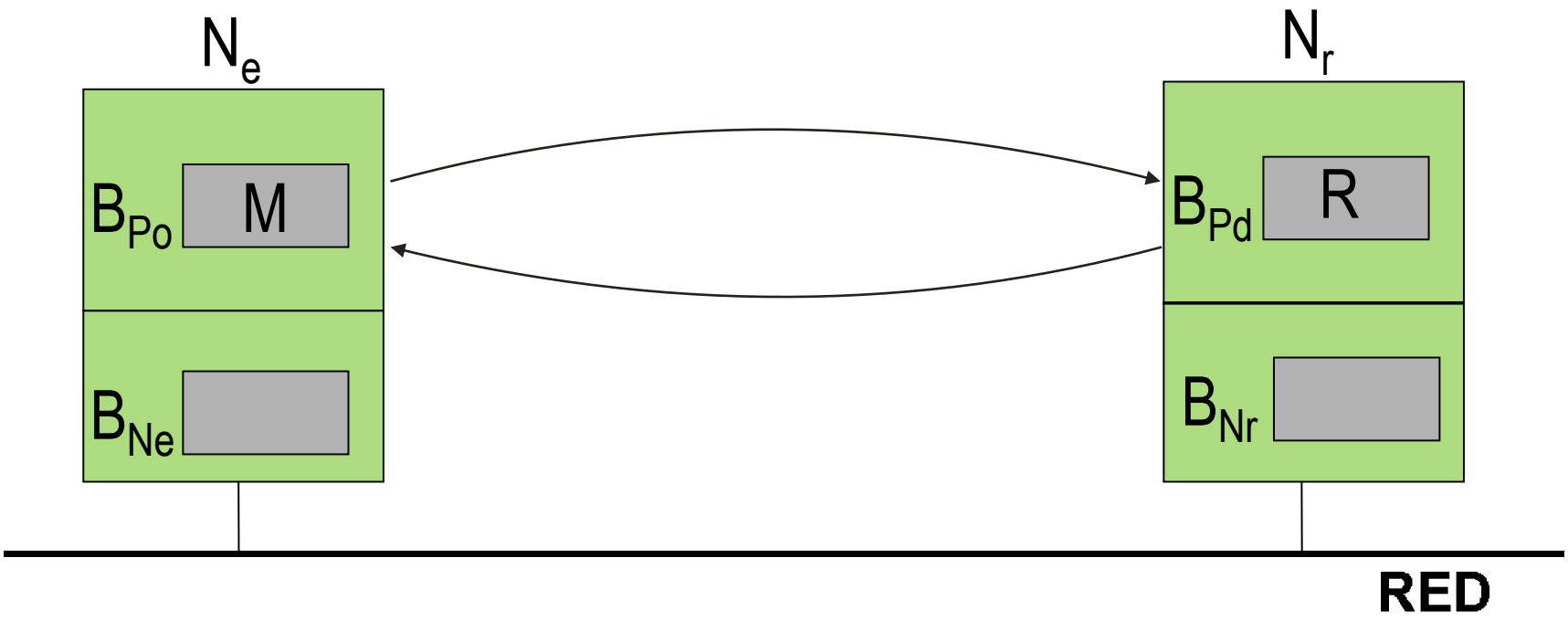
# Ejemplos de formatos de *serialización*

- XDR (RFC 1832): binario, info. implícita campos y tipos
- Soluciones basadas XML: texto, inf. explícita campos y tipos
  - Info de tipos mediante referencia a XML Schema
- JSON: texto, info. explícita campos y tipos
- *Protocol Buffers* (Google): binario, no explícita campos y tipos
  - Pero sí viaja ID único y longitud de cada campo con datos
    - Facilita cambios incrementales en protocolo
- *Java Serialization*: binario, info. explícita campos y tipos
  - Info de campos y tipos mezclada con datos; no requiere IDL
- Muchos otros: ASN.1, Apache Thrift, Apache Avro, BSON,…
- *Wikipedia: Comparison data serialization formats*
- Ejemplos: *http://laurel.datsi.fi.upm.es/~ssoo/SD.dir/serializacion*


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Grado de sincronía en envío y *buffering*

$P_o$ envía M a $P_d$: copia entre *buffers* de procesos: $B_{Po} \rightarrow B_{Pd}$




Relación entre sincronía (cuándo retorna la llamada de envío) y *buffers* requeridos


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Retorno inmediato

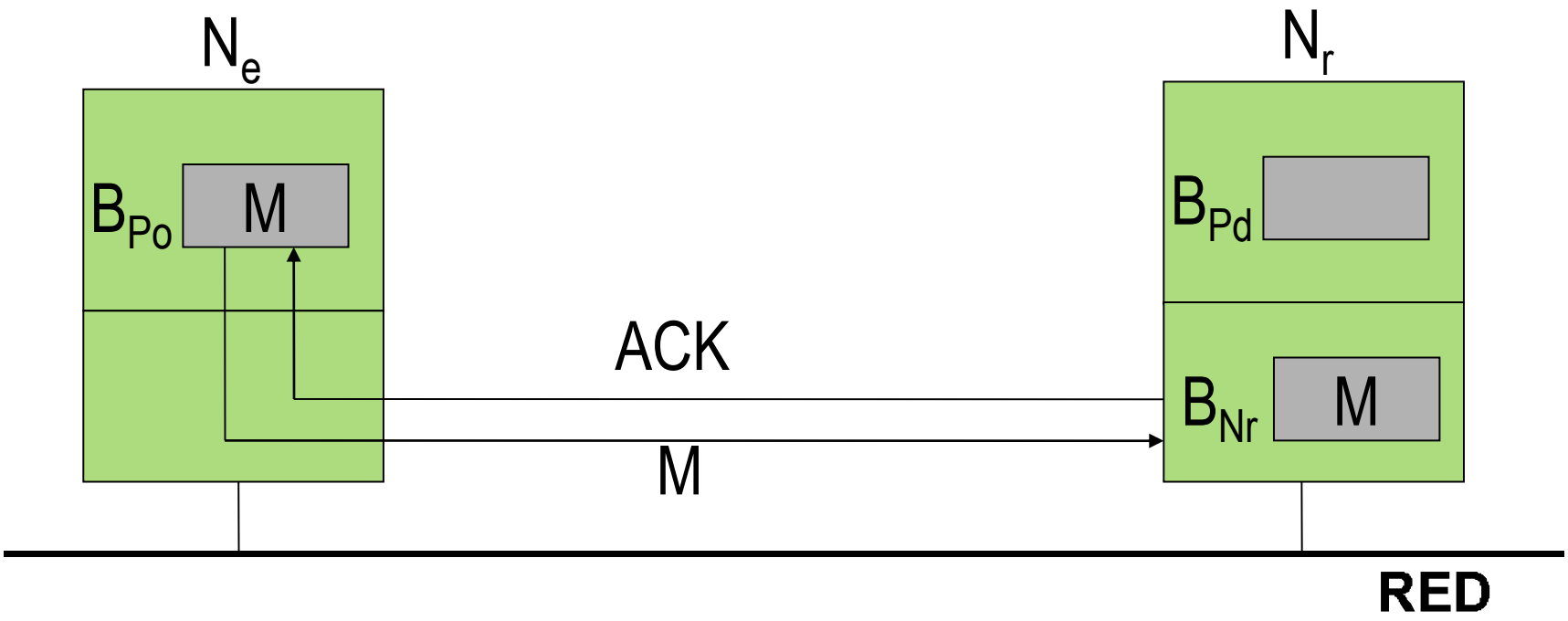

Emisor no puede reutilizar *buffer* de envío inmediatamente


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Retorno después de copia local


Emisor puede reutilizar *buffer* envío inmediatamente pero posible bloqueo si $B_{Ne}$ lleno

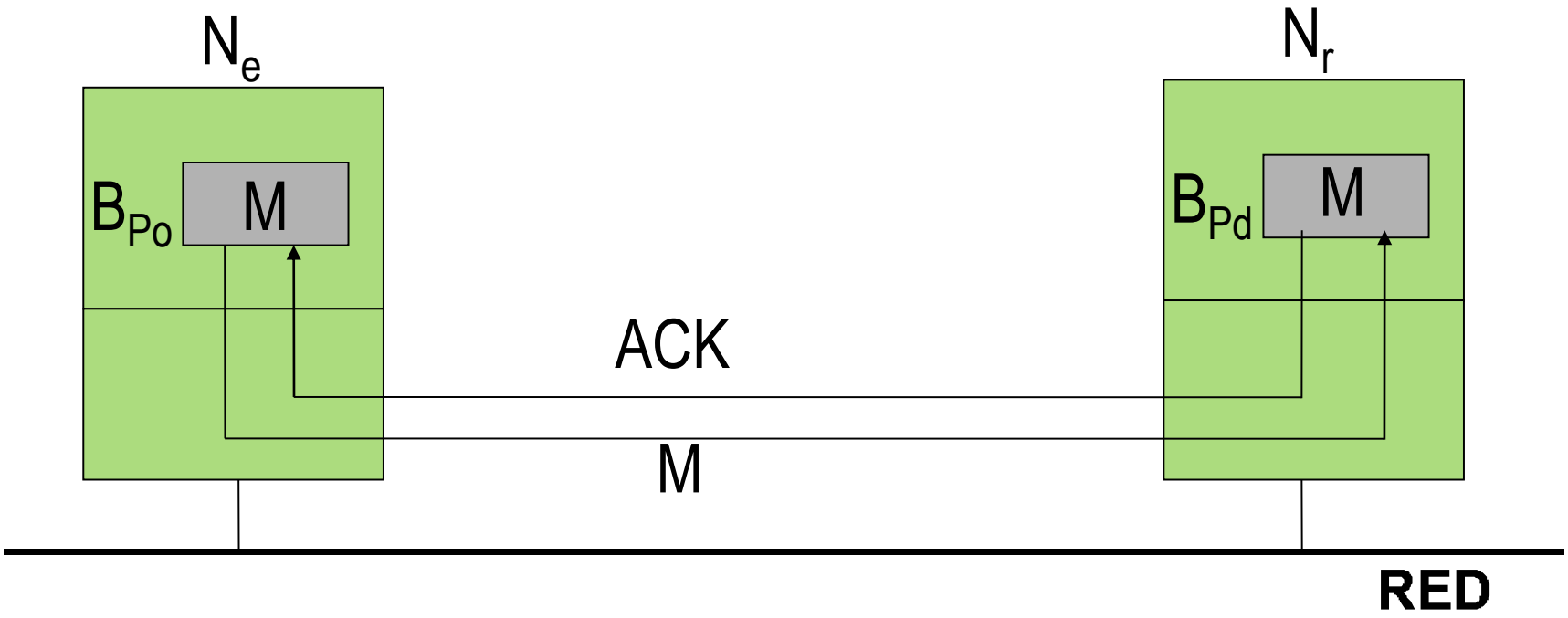
# Retorno después de llegada





Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

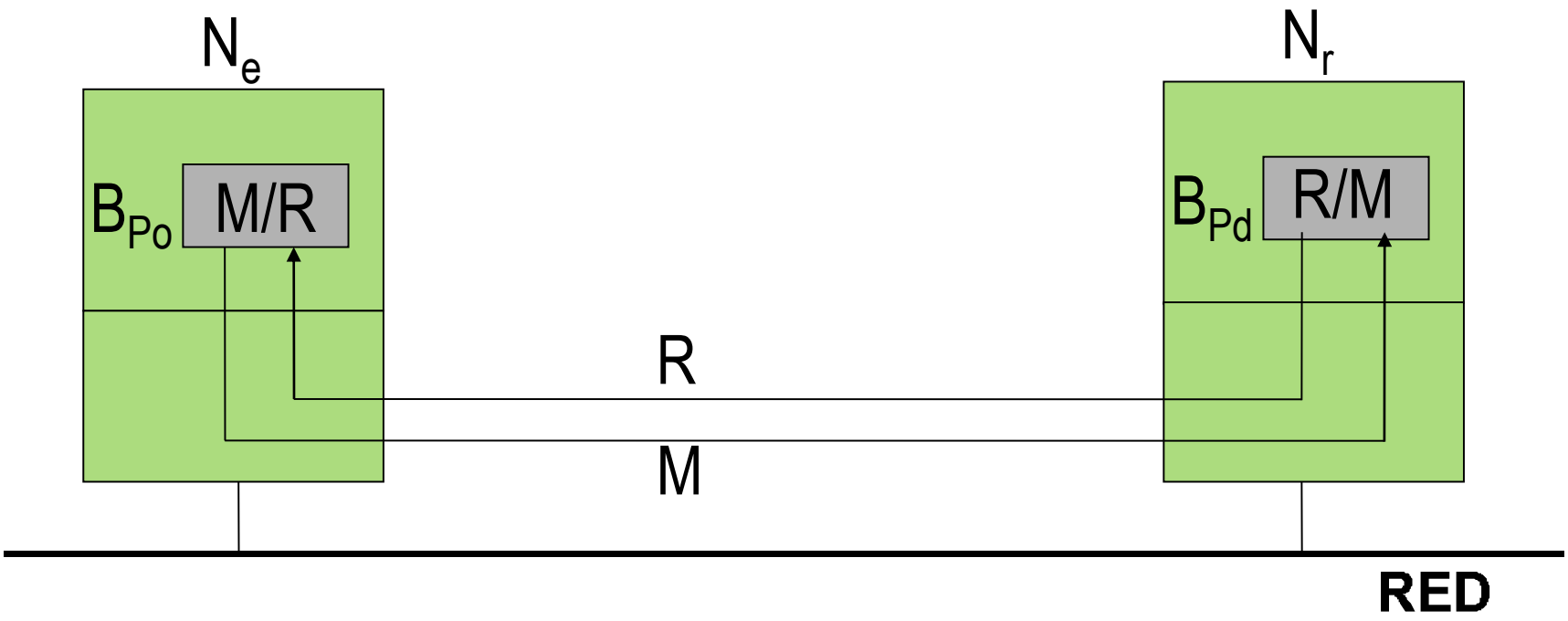
# Retorno después de recepción





Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Retorno después de respuesta




Respuesta sirve de ACK


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Modo de operación en recepción

- Recepción generalmente bloqueante
- Opción no bloqueante: retorna si no hay datos
- Opción asíncrona:
  - Especifica *buffer* donde se almacenará el mensaje y
  - Retorna inmediatamente
  - S. comunicaciones realiza recepción mientras proceso ejecuta
  - Proceso no puede reutilizar *buffer* mientras tanto
- Espera temporizada: se bloquea un tiempo máximo
- Espera múltiple: espera por varias fuentes de datos


Fernando Pérez Costoya — José Mª Peña Sánchez
Mª de los Santos Pérez Hernández

# Sockets: grado de sincronía y *buffering*

- Modo de operación de envío
  - Retorno después de copia local con bloqueo si *buffer* local lleno
  - *Buffer* reservado por SO
- Si aplicación no quiere bloquearse en envío:
  - Usar modo no bloqueante en descriptor socket: error si *buffer* lleno
  - Usar *select/poll/epoll* para comprobar que envío no bloquea
  - Usar E/S asíncrona (*aio_write*)
- Modo de operación de recepción bloqueante
- Si aplicación no quiere bloquearse en recepción:
  - Usar modo no bloqueante en descriptor socket: error si *buffer* vacío
  - Usar *select/poll/epoll* para comprobar que hay datos que recibir
  - Usar E/S asíncrona (*aio_read*)
- Espera múltiple temporizada mediante *select/poll/epoll*