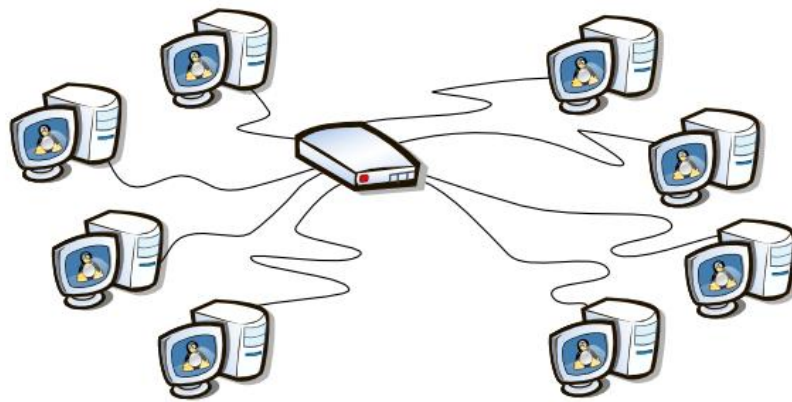


# **Sistemas Operativos en Red**



## Índice

<b>TEMA 1: FUNDAMENTOS DE LOS SISTEMAS DISTRIBUIDOS .....</b>	<b>3</b>
1. Introducción.....	3
2. Paradigmas de comunicación .....	5
3. Modelo de comunicación .....	8
4. Recursos de comunicación .....	15
5. Enfoques .....	21
<b>TEMA 2: INTRODUCCIÓN AL MODELO DE OBJETOS DISTRIBUIDOS.....</b>	<b>23</b>
1. Tecnología Web.....	23
2. Middleware .....	28
3. Integración .....	33
4. Servicios Web.....	34
<b>TEMA 3: SERVICIOS DE NOMBRES.....</b>	<b>38</b>
1. Servicios de nombres.....	38
2. Servicios de directorio .....	46
3. Servicios de descubrimiento .....	50
<b>TEMA 4: TIEMPOS Y ESTADOS GLOBALES.....</b>	<b>55</b>
1. Relojes, eventos y estados .....	55
2. Sincronización de relojes físicos .....	56
3. Tiempo lógico y relojes lógicos .....	59
4. Estados globales .....	61
<b>TEMA 5: SEGURIDAD .....</b>	<b>65</b>
1. Introducción.....	65
2. Técnicas de seguridad.....	67
3. Algoritmos criptográficos.....	72
<b>TEMA 6: SISTEMAS DE ARCHIVO DISTRIBUIDO .....</b>	<b>79</b>
1. Introducción.....	79
2. El servicio de ficheros .....	80
3. Técnicas de implementación .....	81
4. Casos de estudio .....	84
<b>TEMA 7: MEMORIA COMPARTIDA DISTRIBUIDA.....</b>	<b>88</b>
1. Introducción.....	88
2. Algoritmos de MCD .....	89
3. Modelos de consistencia de la memoria.....	93
<b>TEMA 8: INTERACCIÓN Y COOPERACIÓN.....</b>	<b>96</b>
1. Coordinación Distribuida .....	96
2. Algoritmo de elección .....	108

## TEMA 1: FUNDAMENTOS DE LOS SISTEMAS DISTRIBUIDOS

### 1. Introducción

Un **sistema distribuido** es aquel en el que los componentes HW/SW, localizados en computadores unidos mediante la red, comunican y coordinan sus acciones sólo mediante el paso de mensajes. Los elementos de computación, presentan las siguientes características:

- **Independencia:** cada accesorio hardware se añade de manera add-hoc (para un propósito determinado, débilmente acoplado).
- **Interconectados:** los distintos componentes hardware se conectan entre ellos de cualquier forma física.
- **Comunicación y coordinación:** aunque las máquinas estén conectadas a Internet, la diferencia entre estas y las disponibles en Internet es que trabajan de manera coordinada, estableciendo una comunicación continua.

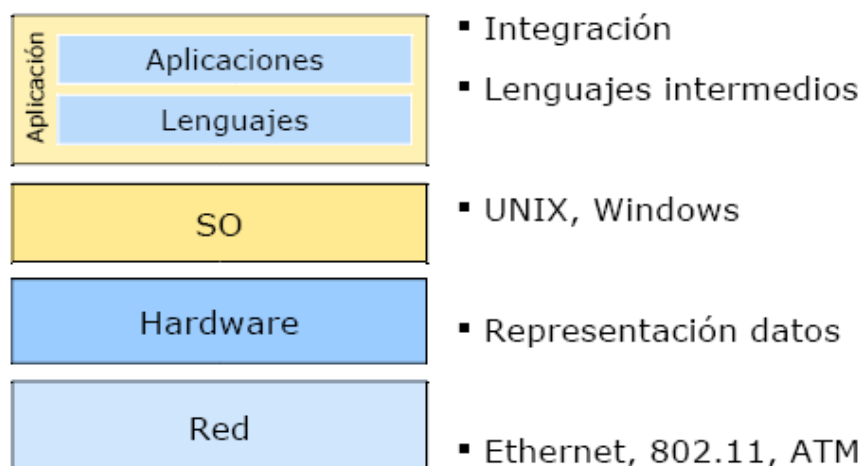
Ejemplos de sistemas distribuidos: Internet, intranets, computación ubicua.

La **computación distribuida** es un nuevo modelo para resolver problemas de computación masiva utilizando un gran número de computadoras organizadas en racimos incrustados en una infraestructura de telecomunicaciones distribuida. Consiste en compartir **recursos heterogéneos** situados en distintos lugares y pertenecientes a diferentes dominios de administración sobre una red que utiliza estándares abiertos.

#### Aspectos relacionados:

Las propiedades a medir en un sistema distribuido son:

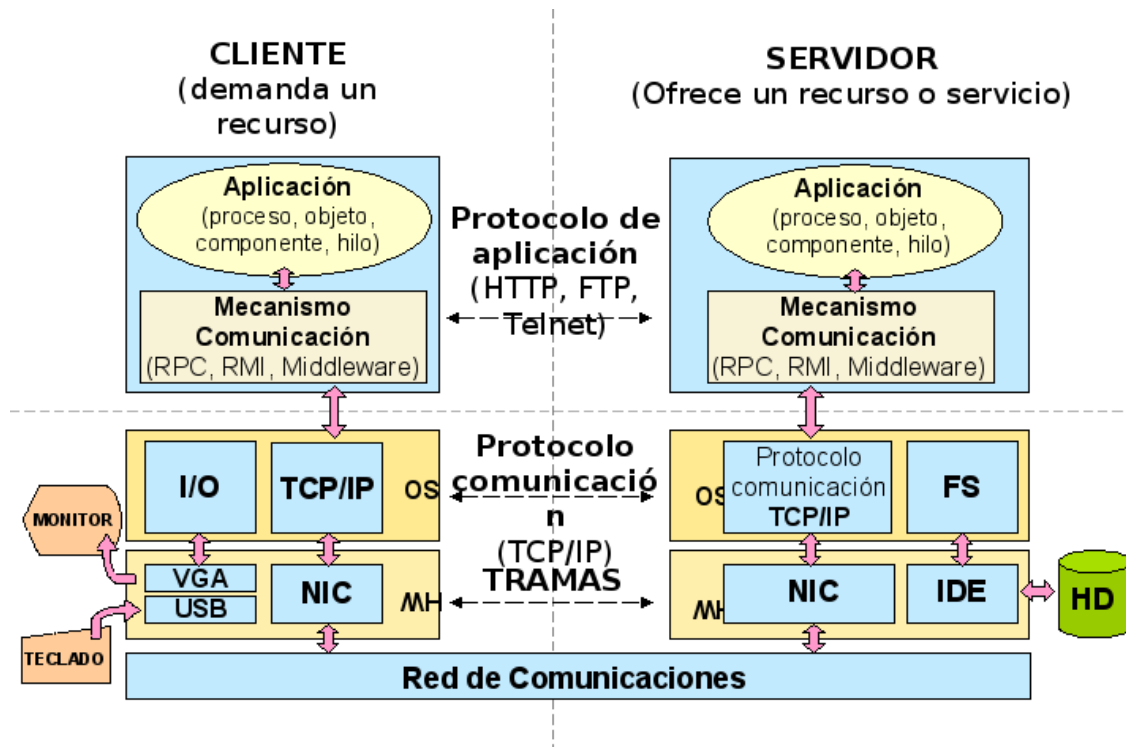
- **Heterogeneidad:** a nivel de red, HW, SW. Internet permite que los usuarios accedan a los servicios y ejecuten aplicaciones sobre un conjunto heterogéneo de redes y computadores (variedad de maquinaria). Los elementos afectados son: redes, hardware de computadores, sistemas operativos, lenguaje de programación. A pesar de que las redes de Internet son muy distintas, las diferencias son transparentes gracias al empleo de protocolos de Internet.



- **Extensibilidad:** grado en el que en un sistema distribuido se pueden añadir nuevos servicios de compartición de recursos y ponerlos a disposición de los clientes.
- **Escalabilidad:** un sistema es escalable cuando su efectividad no se ve afectada en el aumento de recursos y número de usuarios.
- **Seguridad:** los sistemas distribuidos son entornos proclives a ataques externos. La seguridad de los recursos de información tiene tres componentes: Confidencialidad, Integridad y Disponibilidad (Firewalls, SSL,...)
- **Concurrencia y sincronización:** varios usuarios intentan acceder al mismo tiempo a un mismo recurso. Hay que sincronizar las peticiones y las concesiones de servicio. Es por eso que los servicios permiten, procesar concurrentemente múltiples peticiones de los clientes.
- **Tolerancia a fallos:** capacidad del sistema para seguir funcionando tras un fallo o error. Algunas técnicas empleadas para solucionar los posibles fallos en el sistema son: detección de fallos, enmascaramiento de fallos, tolerancia a fallos, recuperación frente a fallos, redundancia
- **Transparencia:** ocultación al usuario y al programador de la separación de los componentes de un sistema distribuido, de forma que perciba el sistema como un todo. Tipos:
  - **De acceso:** permite acceder a recursos remotos y locales con las mismas operaciones.
  - **De ubicación:** permite acceder a recursos sin conocer su localización.
  - **De movilidad:** permite la reubicación de los recursos y clientes en un sistema sin afectar la operación de los usuarios y los programas.
  - **De escalabilidad:** permite al sistema y a las aplicaciones aumentar de tamaño sin afectar a la estructura o a las aplicaciones.
  - **Frente a fallos:** permite ocultar los fallos, para no interferir en las tareas de los programadores o usuarios.

## Arquitectura

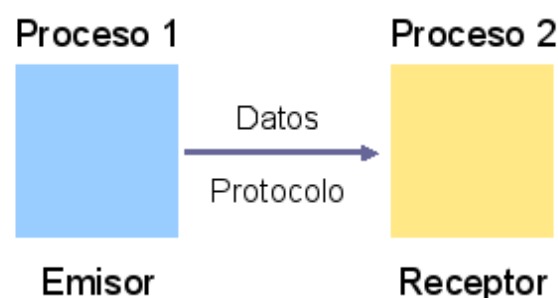
Consta de una red de comunicaciones para la comunicación, con su correspondiente hardware de red, un sistema operativo que de soporte al mecanismo de comunicación que se utilice, el mecanismo de comunicación de procesos (IPC Inter-process communication), los procesos y el protocolo de comunicación.



## 2. Paradigmas de comunicación

La comunicación entre procesos es el mecanismo básico de comunicación en los sistemas distribuidos: la posibilidad de que procesos separados e independientes se comuniquen entre sí para colaborar en una tarea. Normalmente la **comunicación** entre procesos se hace de forma **transparente** y por tanto para el programador parece que simplemente se pasa un mensaje.

En la imagen, dos procesos independientes, con la posibilidad de ejecutarse en máquinas separadas, intercambian datos en una red de comunicaciones:



Las comunicaciones se establecen por medio de protocolos acordados por los procesos.

Modelos básicos:

- **Unidifusión:** cuando la comunicación es desde un proceso a únicamente otro proceso.
- **Multidifusión:** cuando la comunicación es desde un proceso a un grupo de procesos.

Comunicación entre procesos mediante IPC:



La **interfaz** mínima para un sistema de comunicación es: enviar y recibir. Otros elementos típicos de una interfaz de comunicación son: conectar / desconectar. De modo que el cliente solicita conexión al servidor y el servidor la acepta.

Toda comunicación se hace siguiendo un protocolo para que sea posible el entendimiento entre ambos extremos. Los protocolos pueden ser basados en texto (HTTP, POP3), de tipo solicitud respuesta (HTTP, FTP), Orientado o no a conexión y con o sin estado. Las especificaciones de los protocolos abiertos se encuentran publicadas y uno de los sitios más conocido son los RFCs.

La forma más sencilla de **sincronización** de eventos es por medio de peticiones bloqueantes: supresión de la ejecución del proceso hasta que la operación invocada haya finalizado.

- **Bloqueantes o síncronas:** el proceso queda bloqueado hasta recibir la respuesta. Para que un proceso no quede bloqueado indefinidamente se utilizan temporizadores y/o se crean procesos hijos para atender o realizar las operaciones.
- **No bloqueantes o asíncronas:** no esperan a que llegue la respuesta. No causan bloqueos. Más adelante se informará si ha tenido éxito o no.

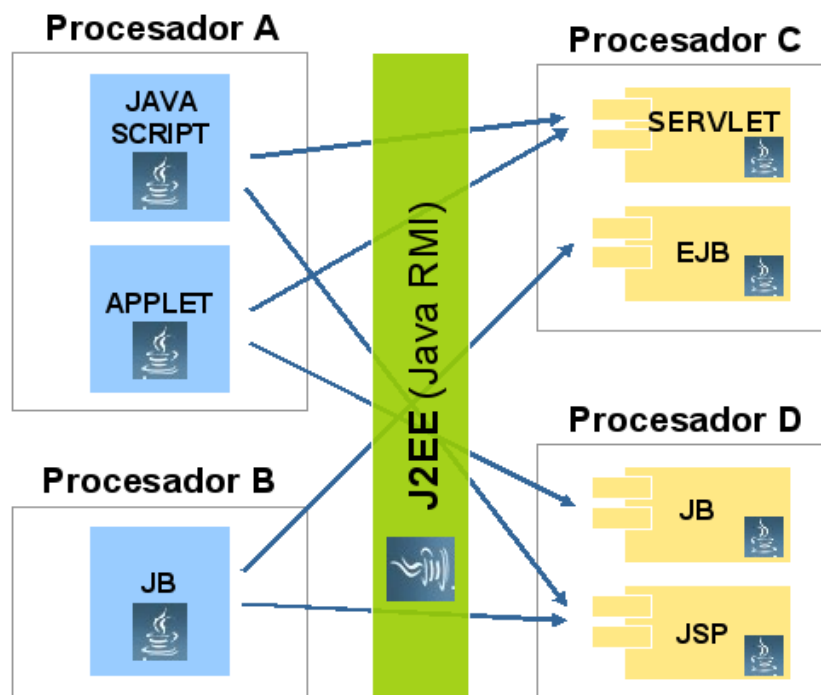
La opción bloqueante o no bloqueante se realiza a nivel de sistema operativo y se inicia por medio de las funcionalidades proporcionadas por el mecanismo de comunicación entre procesos.

## Componentes distribuidos

**Plataformas:** son el nivel de HW y las capas más bajas de SW en los sistemas distribuidos y aplicaciones. Son implementadas independientemente en cada computador, proporcionando una interfaz de programación del sistema a un nivel que facilita la comunicación y la coordinación entre procesos. (java, .net)

**Middleware:** capa de SW cuyo propósito es enmascarar la heterogeneidad y proporcionar un modelo de programación conveniente para los programadores de aplicaciones. Se representa mediante procesos u objetos en un conjunto de computadores que interactúan entre sí para implementar mecanismos de comunicación y de recursos compartidos para aplicaciones distribuidas. Mejora el nivel de comunicación de los programas gracias a: invocación remota, comunicación entre un grupo, notificación de eventos, etc. (Java RMI, servicios web, Framework remoting).

**Servicios:** el middleware también puede proporcionar servicios para su uso en los programas de aplicación. Existen servicios de infraestructuras ligadas fuertemente al modelo de programación distribuida que proporciona el middleware. También existen servicios por encima de los middleware.



### 3. Modelo de comunicación

aplicaciones colaborativas
Servicios de red, (ORB) object request broker, agentes móviles
Llamadas a procedimientos remotos, invocación de métodos remotos
Cliente/Servidor, Peer to Peer (P2P)
Paso de mensajes

#### *Paradigmas de computación distribuida*

**Paso de mensajes:** intercambio de unidades lógicas (mensajes). El emisor envía un mensaje que representa una petición. El mensaje se entrega a un receptor, que procesa la petición y envía un mensaje como respuesta. En secuencia, la réplica puede disparar posteriores peticiones, que lleven a sucesivas respuestas. La única interfaz obligatoria es enviar y recibir. Y si lo orientamos a conexión también conectar/desconectar. Ejemplo: **sockets**.

**Cliente/Servidor:** creado sobre el paradigma de paso de mensajes. Cada proceso tiene un rol. El servidor espera peticiones del cliente de forma pasiva. Se necesita concurrencia puesto que un servidor puede dar servicio a varios clientes y estos pueden solicitar servicios al mismo tiempo. Puede implementar mantenimiento de sesión. Ejemplos: **HTTP, DNS, FTP, SMTP**.

**Peer to Peer (P2P):** el comportamiento es parecido al C/S pero cada proceso actúa como cliente y servidor a la vez. Cliente (envío de peticiones, recepción de respuesta), Servidor (recepción de solicitudes, procesamiento de solicitudes, envío de respuesta, propagación de solicitudes). Apropiado para mensajería instantánea o compartición de archivos. Ejemplos: **Napster, Gnutella, BitTorrent**.

**Llamada a procedimientos remotos (RPC):** proporciona la misma abstracción que si el software distribuido se programase de una manera similar a las aplicaciones convencionales que se ejecutan sobre un único procesador. Utilizando este modelo, la comunicación entre procesos se realiza utilizando un concepto similar al de una llamada a un procedimiento local.

**Invocación de métodos remotos (RMI):** es el equivalente en orientación a objetos de las llamadas a procedimientos remotos: un proceso invoca métodos de un objeto que reside en un ordenador remoto. Como en RPC, los argumentos se pueden pasar con la invocación y se puede devolver un valor cuando la operación ha concluido.

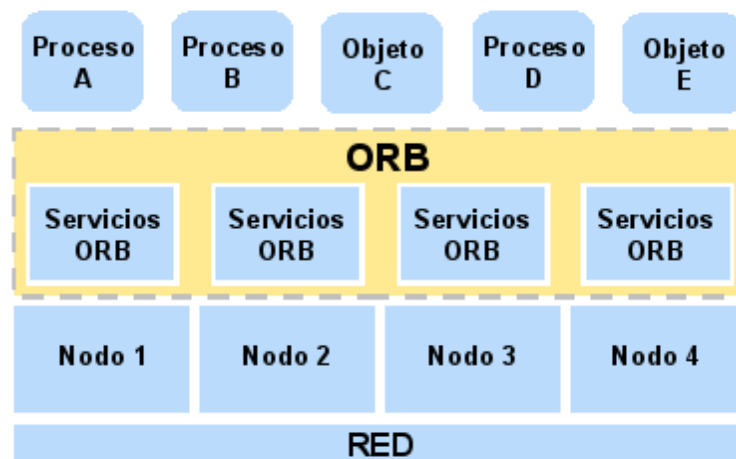
**Object Request Broker (ORB):** en este paradigma, un proceso solicita una petición a un ORB, el cual redirige la petición al objeto apropiado que proporciona dicho servicio. Se parece mucho a RMI, con la diferencia de que el ORB funciona como middleware, permitiendo a una aplicación solicitante



acceder a varios objetos (remotos o locales) y ofrecerles una cantidad de servicios estándar (directorio de servicios, nombres, transacciones, persistencia, notificación, seguridad, gestión de bloqueo y concurrencia, ciclo de vida de objetos, negociación, blog, planificación). Dichos objetos pueden ofrecer heterogeneidad (pueden ser diferentes). Base de la arquitectura CORBA Ejemplos: **J2EE, .NET**.

El ORB se encuentra dividido en:

- El stub del cliente.
- La interfaz de invocación dinámica (DII, Dynamic Invocation Interface)
- La interfaz del ORB y el núcleo del ORB, incluyendo el repositorio de interfaces (IR Interface Repository) y el repositorio de implementaciones.
- Los adaptadores de los objetos.
- Un esqueleto estático IDL para cada objeto conectado a él.
- Una interfaz para esqueletos dinámica (DSI Dynamic Skeleton Interface).
- La especificación de interoperabilidad CORBA 2.0



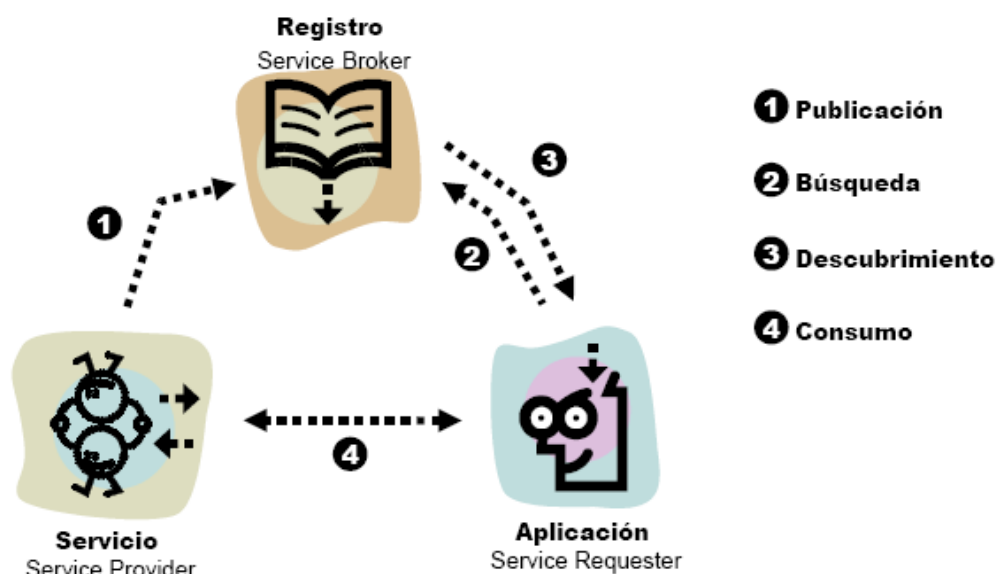
### *Paradigma de objetos remotos*

Maneja la transferencia de estructuras de datos, de manera que sean compatibles entre los dos objetos. Para ello utiliza un estándar para convertir las estructuras de datos en un flujo de bytes, conservando el orden de los bytes entre distintas arquitecturas. Este proceso se denomina marshalling (y también su opuesto, unmarshalling) que se basa en la definición del tipo de objeto, ubicación (IP+puerto) y identificación del objeto. El cliente utiliza un servicio de nombrado para localizar el objeto.

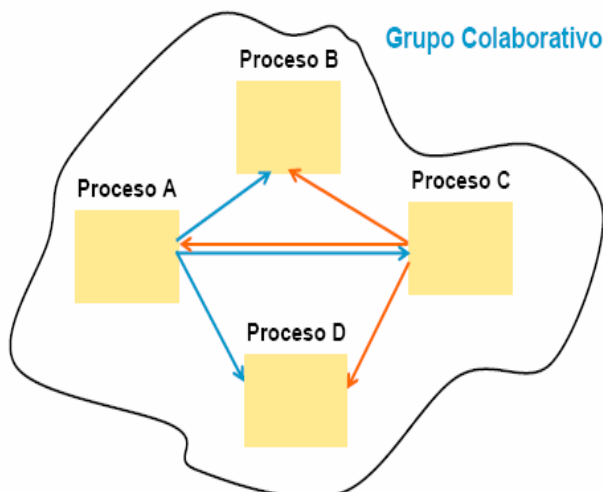
**Agentes móviles:** define programas u objetos transportables con autonomía propia de acuerdo a un itinerario. Implementa serialización. Cada objeto/programa de este tipo contiene su identificación, su itinerario (lista de direcciones de ordenadores) y los datos que requiere para realizar la tarea. Permite la localización de servidores e implementa elementos de seguridad.

Es una buena idea pero aún es una tecnología poco madura y por su definición propensa a ataques (deja ejecutar un programa en nuestra máquina). Está en desarrollo y de momento es una plataforma insegura y lenta. Ejemplos: **Aglets, Concordia.**

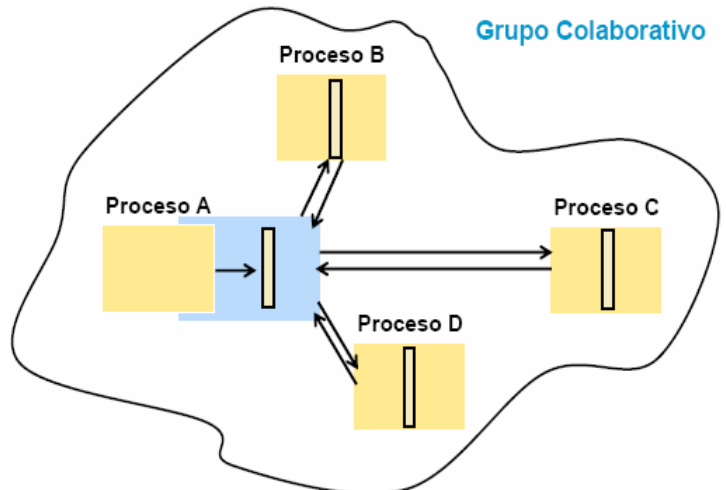
**Servicios de red:** en este modelo, los proveedores de servicios se registran en los servidores de directorios de una red. Un proceso que desee un servicio particular contacta con el servidor del directorio en tiempo de ejecución, devolviendo la referencia al servicio solicitado si se halla disponible, para interactuar con el. Es una extensión de RMI, ya que la única diferencia es que los servidores se registran en un directorio global. Si se emplean identificadores, se obtiene una mayor transparencia de localización. Ejemplos: **SOAP, JINI, UPNP.**



**Aplicaciones colaborativas (groupware):** los procesos participan en grupo en una sesión colaborativa. Cada proceso participante puede hacer contribuciones a todos o parte del grupo. Los procesos realizan esto mediante mensajes de multidifusión o mediante el uso de pizarras virtuales para enviar datos o permitir ver a cada participante los datos sobre una visualización compartida. Ejemplos: Lotus QuickPlaces, Messenger, JSDT (API de Java), NSYP (protocolo de estándar).



Paradigma de aplicaciones colaborativas: basado en mensajes

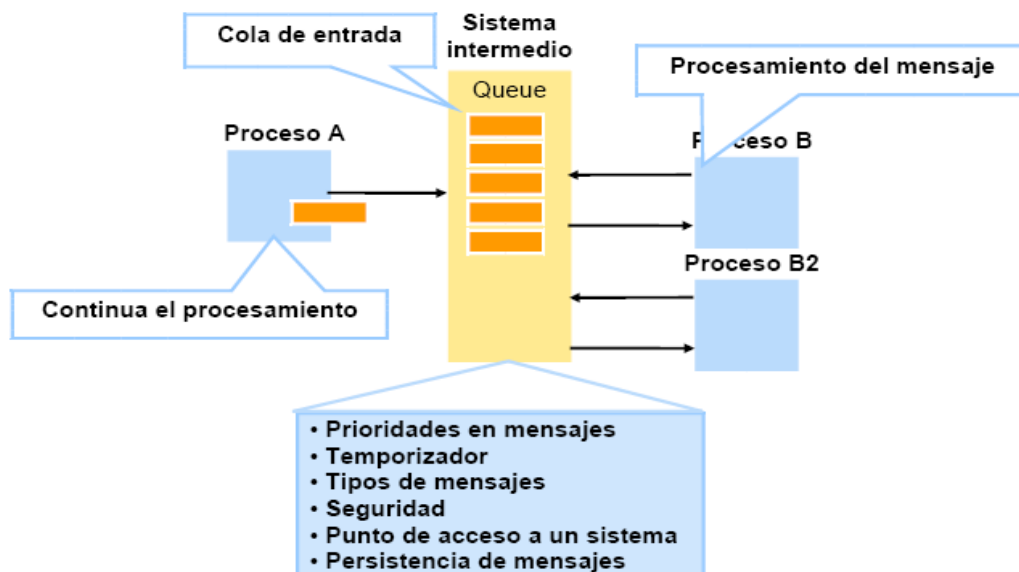


Paradigma de aplicaciones colaborativas: basado en pizarra

**Sistema de mensajes (MOM):** se trata de una evolución del paso de mensajes. En este paradigma, un sistema de mensajes sirve de intermediario entre procesos separados e independientes. El sistema de mensajes actúa como un conmutador para mensajes, a través del cual los procesos intercambian mensajes asíncronamente, de una forma desacoplada. El emisor deposita el mensaje en el sistema de mensajes, el cual redirige el mensaje a la cola de entrada del receptor. Una vez enviado, el emisor queda libre para realizar otras tareas. Ejemplos: **JMS, MSMQ, MQUEUE**. Hay de dos tipos:

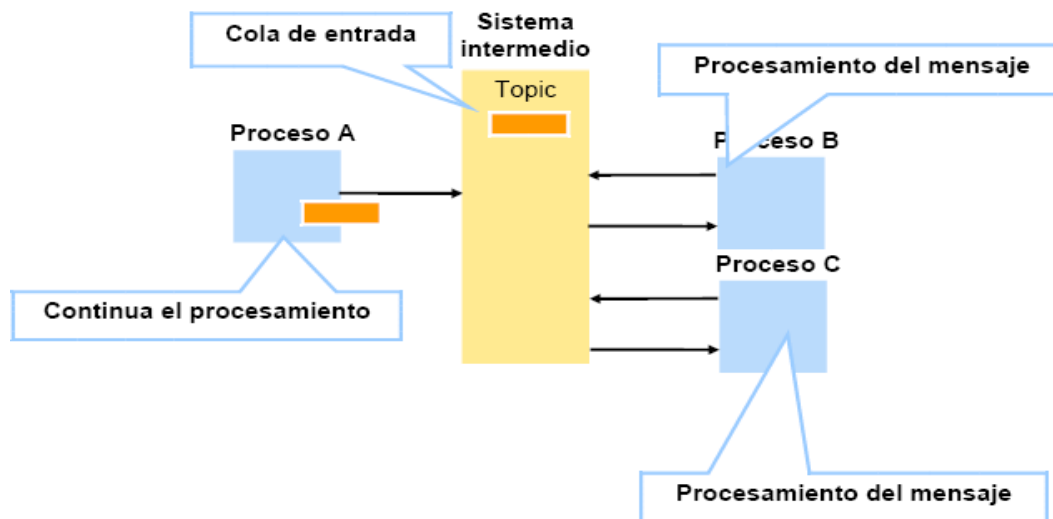
- **Punto a punto:** el emisor envía mensajes al receptor por medio de un middleware, permitiendo un total desacoplamiento entre el envío y la recepción de mensajes.

Paradigma de sistema de mensajes punto a punto



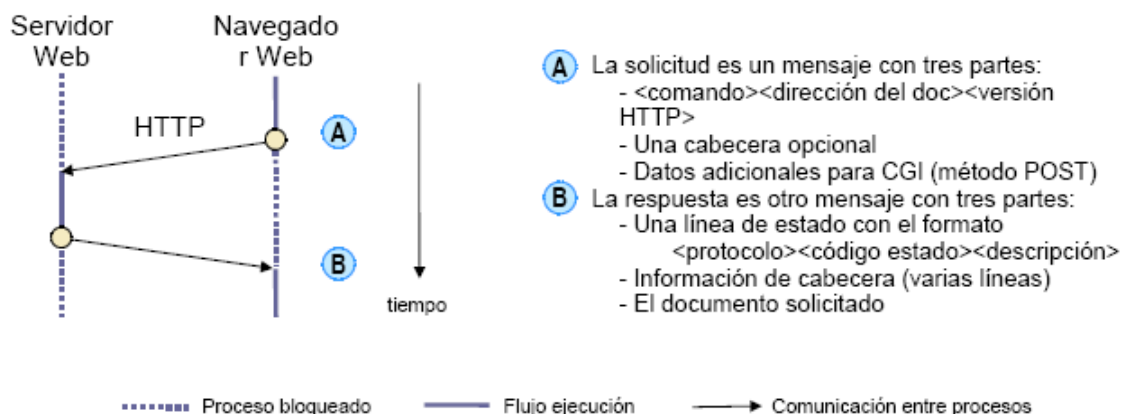
- **Publicación/suscripción:** cada mensaje se asocia a un determinado tema o evento. Las aplicaciones interesadas en el suceso de un evento específico se pueden suscribir a los mensajes de dicho evento. Cuando el evento ocurre, el proceso publica un mensaje anunciando el evento o asunto. El middleware se encargará de distribuir a todos los suscriptores. Se produce una gran abstracción de multidifusión.

*Paradigma de sistema de mensajes publicación/suscripción*



**Protocolos:** existen distintos tipos:

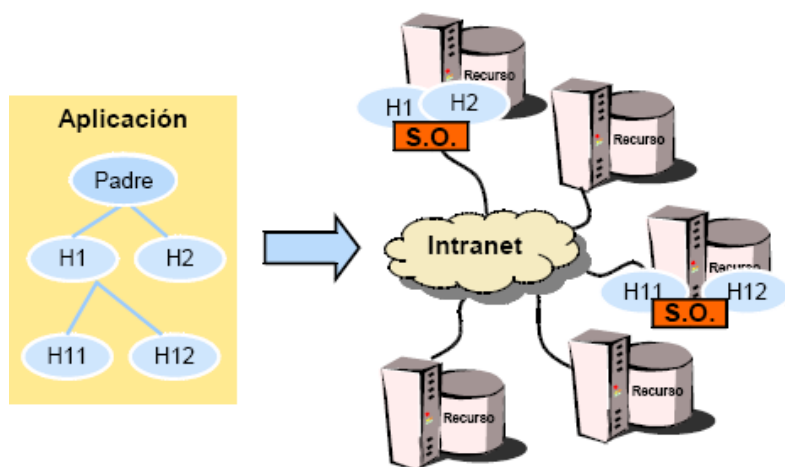
- **Basados en texto:** El empaquetamiento de datos es el más simple: se intercambian cadenas de texto o texto codificado. Lo bueno es que puede ser fácilmente analizado por un programa y mostrado a un humano. Ejemplos: **HTTP, SMTP, POP3, FTP.**
- **De Solicitud-Respuesta:** En estos protocolos, un lado invoca una petición y espera una respuesta del otro extremo. Posteriormente, puede ser enviada otra solicitud y esperar una nueva respuesta. Ejemplos: **FTP, HTTP, SMTP.**



- **Orientados a conexión:** Dos procesos establecen una conexión y posteriormente insertan o extraen datos de esa conexión. Tras establecer la conexión, no se necesita la identificación del emisor o receptor.
- **No orientados a conexión:** Los datos son intercambiados por medio de paquetes independientes, cada uno de los cuales necesita explícitamente la dirección del receptor.
- **Con estado:** Los paquetes guardan un valor que definirá el estado en el que se halla una operación.
- **Sin estado:** No se considera en ningún momento el estado de una operación.

**Paradigmas Hardware:** se trata de infraestructuras hardware y software para ofrecer un alto rendimiento y calidad de servicio, esto se consigue mediante la unión de un conjunto de computadores para formar supercomputadores, capaces de ofrecer una mayor disponibilidad (calidad) y balanceo de carga (rendimiento). Es necesario que los programas sean paralelizables en procesos para sacar partido a estos paradigmas.

- **Cluster:** conjuntos de computadoras contruidos mediante la utilización de componentes de hardware comunes y que se comportan como si fuesen una única computadora. Los ordenadores se conectan entre sí mediante Intranet (red local) y ofrecen una gran **homogeneidad** para garantizar una mayor disponibilidad y rendimiento. Los programas ejecutables en los clusters suelen ejecutarse de manera paralela con tal de equilibrar la carga de las computadoras de los clusters. El gestor de recursos del cluster se halla de manera centralizada. Ejemplo: **OpenMosix**.



*Paradigma de cluster*

- **Grid:** se trata de la unión en malla de un conjunto de clusters no necesariamente cercanos (desacoplamiento) para actuar como un único superordenador de manera transparente, con tal de aumentar la capacidad de procesamiento y almacenamiento. Los clusters se conectan entre sí mediante Internet y ofrecen una gran **heterogeneidad**. Se respetan una serie de políticas de seguridad y aplicaciones internas. Ejemplos: **Globos, OSGA**



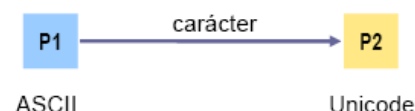
Importante saber cuándo usar un grid y cuando un cluster.

**Representación de datos:** los equipos que forman parte de un sistema distribuido pueden ser heterogéneos y las formas en que representan los datos los procesos también.

Para solucionar este problema se tiene que utilizar la representación externa que puede, además, incluir empaquetamiento de datos (marshalling en las listas de valores o serialización en los objetos). La idea es que ambos procesos adapten (traduzcan) su representación a una representación externa estándar elegida para comunicarse. Esta codificación normalizada de los datos puede ser entre otras XDR, ANS.1, XML.



Comunicación de un dato entero entre dos equipos heterogéneos



Comunicación de un carácter entre dos procesos con diferente mecanismo de representación de datos

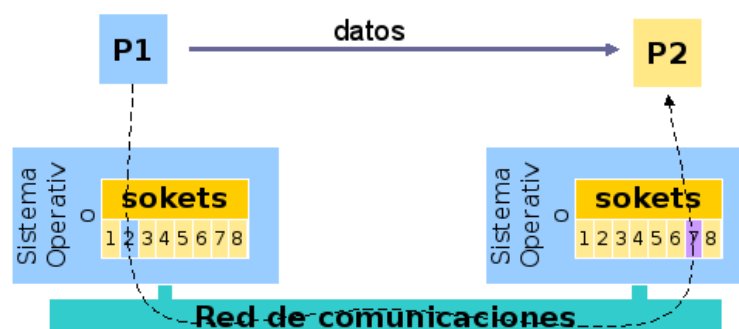
## 4. Recursos de comunicación

### API de Sockets

Es una estructura lógica de intercambio. El API de sockets es un mecanismo que proporciona un nivel bajo de abstracción para IPC. La comprensión del API de sockets es necesaria porque los mecanismos superiores de comunicación se construyen mediante operaciones proporcionadas por el API de sockets y porque es el mecanismo de IPC más apropiado para plataformas con recursos limitados o necesidad de tiempos cortos de respuesta. Al tratarse de una biblioteca de programación IPC, su uso se extiende a la mayoría de sistemas operativos.

El modelo de paso de mensajes identifica dos tipos de mensajes:

- **Con conexión:** se retransmiten flujo de datos, la comunicación es confiable y el envío de paquetes es ordenado.
- **Sin conexión:** se retransmiten datagramas, a comunicación es no confiable y el envío de paquetes es desordenado.



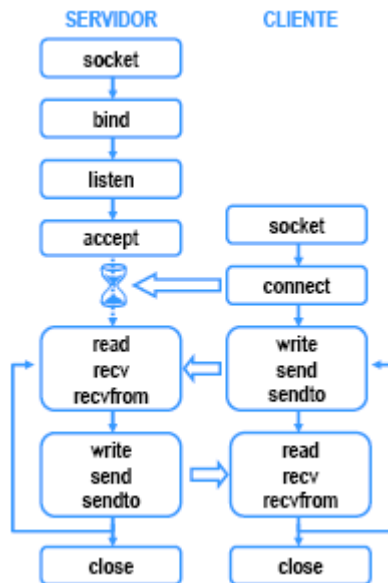
*Comunicación entre dos procesos  
remotos a través de sockets*

## Comunicación simple entre dos sockets

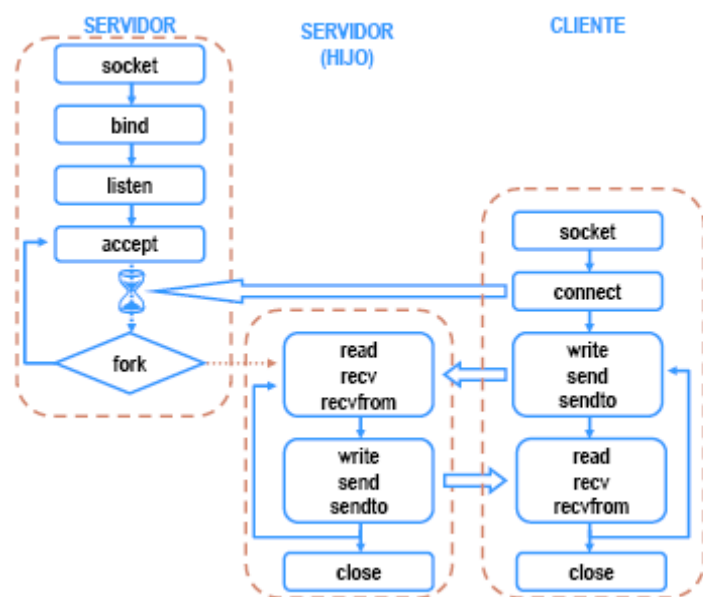
	Descripción	Entrada	Salida	Ejemplo
Socket	Crea un extremo de una comunicación	<b>Dominio:</b> AF_UNIX: Local AF_INET: Internet <b>Tipo:</b> SOCK_STREAM SOCK_DGRAM <b>Protocolo:</b> 0	<b>-1:</b> Error  <b>&gt;0:</b> Descriptor del socket	<pre>int Descriptor;  Descriptor = socket(AF_INET,SOCK_STREAM,0);  if (Descriptor == -1) printf ("Error\n");</pre>
Bind	Enlaza un nombre (IP + Puerto) a un socket	Socket Dirección Tamaño dirección	<b>0:</b> Éxito  <b>-1:</b> Error	<pre>Direccion.sin_family = AF_INET; Direccion.sin_port = htons(8989); Direccion.sin_addr.s_addr INADDR_ANY;  if (bind(Descriptor,(struct sockaddr *)&amp;Direccion,sizeof (Direccion)) == -1) printf ("Error\n");</pre>
Listen	Prepara un socket para aceptar peticiones	Socket Tamaño de la cola	<b>0:</b> Correcto  <b>-1:</b> Error	<pre>if (listen (Descriptor, 5) == -1){ printf("Error\n"); }</pre>
Accept	Acepta una conexión sobre un socket	socket  Dirección (REF)  Tamaño de dirección (REF)	<b>&gt;0:</b> Conector aceptado  <b>-1:</b> Error	<pre>struct sockaddr Cliente; int Descriptor_Cliente; int Longitud_Cliente;      Descriptor_Cliente = accept(Descriptor,&amp;Cliente,&amp;Longitud_Cliente) if (Descriptor_Cliente == -1) printf("Error\n");</pre>
Connect	Inicia una conexión en un socket	Socket  Dirección  Tamaño dirección	<b>0:</b> Correcto  <b>-1:</b> Error	<pre>struct sockaddr_in Direccion; Direccion.sin_family=AF_INET; Direccion.sin_addr.s_addr=inet_addr("127.0.0.0"); Direccion.sin_port=htons(8989);  if (connect(Descriptor,(struct sockaddr *)&amp;Direccion, sizeof(Direccion)) == -1) printf ("Error\n");</pre>
Read	Lee (recibe) de un socket (STREAM)	socket  Buffer (REF)  Tamaño del buffer	<b>0:</b> Final de lectura  <b>&gt;0:</b> Bytes leídos <b>-1:</b> Error	<pre>char Buffer[100]; int TamLeido; TamLeido = read(Descriptor, Buffer, 100);  if (TamLeido == -1) printf("Error\n");</pre>
Write	Escribe (envía) con socket (STREAM)	socket  Buffer  Bytes a enviar	<b>&gt;=0:</b> Bytes enviados  <b>-1:</b> Error	<pre>char Buffer[11]="Mi mensaje"; int TamEnviado; TamEnviado = write(Descriptor, Buffer, 11);  if (TamEnviado == -1) printf("Error\n");</pre>
Close				



### Comunicación simple



### Comunicación concurrente



### RPC

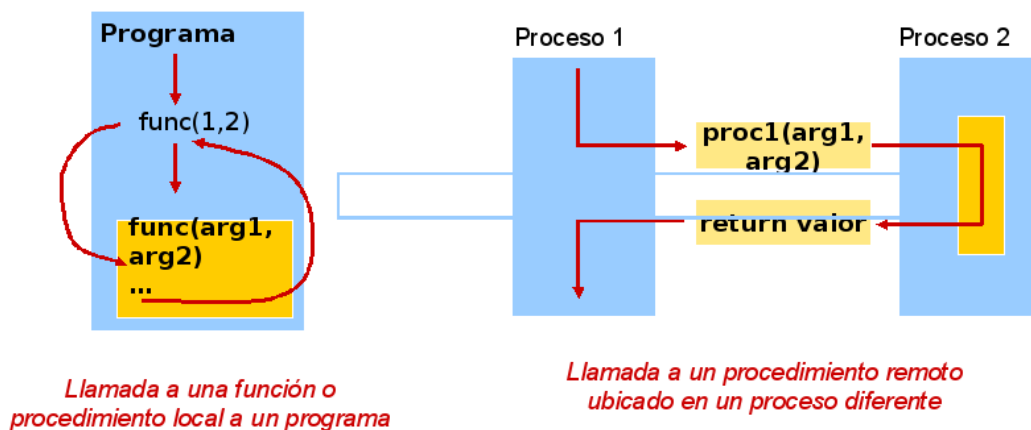
Brinda la oportunidad de llamar a procedimientos/funciones remotos (de otros procesos). Implementado sobre el protocolo petición/respuesta. Es un sistema abierto (SUN/UNIX). Orientado a lenguaje C. Provee un compilador de interfaces (RPCGEN) y usa la representación externa XDR.

Se tiene que definir un proceso cliente (que llama al método, teniendo la dirección del servidor), un proceso servidor (que implementa el método) y una interfaz (cod.x).

El servidor registra el servicio en un puerto del portmap. El cliente pide el servicio. El servidor le indica el puerto. El cliente invoca el método remoto a través del puerto indicado y el servidor devuelve un resultado.

Consta de dos componentes:

- Compilador de interfaces (RPCGEN).
- Representación externa de datos (SUN XDR).



Para construir procedimientos remotos:

```

program SUMA_PROG {
    version SUMA_VERS {
        int SUMA(numeros) = 1;
    } = 1;
} = 0x20000001;
    
```

Número de procedimiento

Número de versión

Número de programa

*Declaración de procedimientos del proceso remoto*

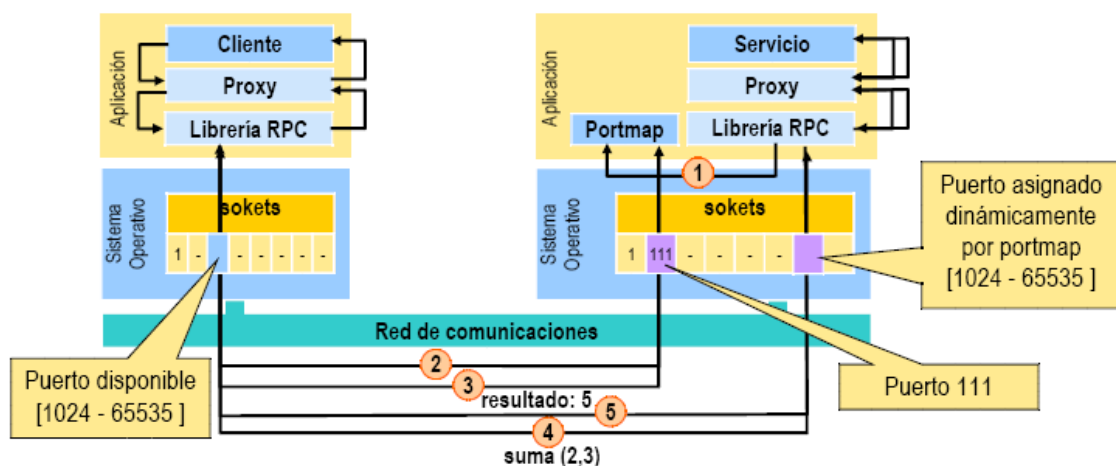
- 0x00000000 - 0x1FFFFFFF: Definidos por Sun
- 0x20000000 - 0x3FFFFFFF: **Definidos por el usuario**
- 0x40000000 - 0x5FFFFFFF: Temporales
- 0x60000000 - 0xFFFFFFFF: Reservados

*Obtención del número de programa*

Donde asignamos el número de procedimiento, número de versión y el número de programa (en ese orden).

Funcionamiento de RPC:

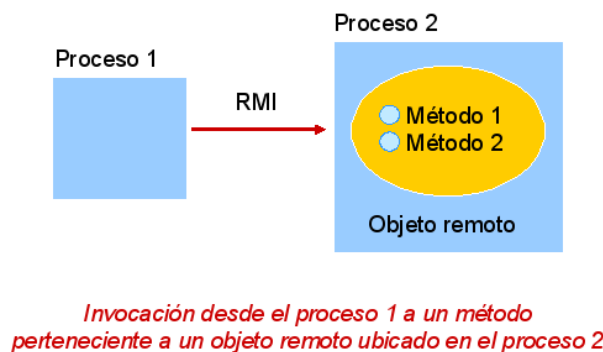
- 1 Registro del servicio
- 2 Cliente realiza petición al servicio remoto
- 3 Portmap envía el puerto de conexión del servicio requerido al cliente
- 4 El cliente instancia un método remoto a través del puerto indicado
- 5 El proceso servidor devuelve el resultado



**RMI**

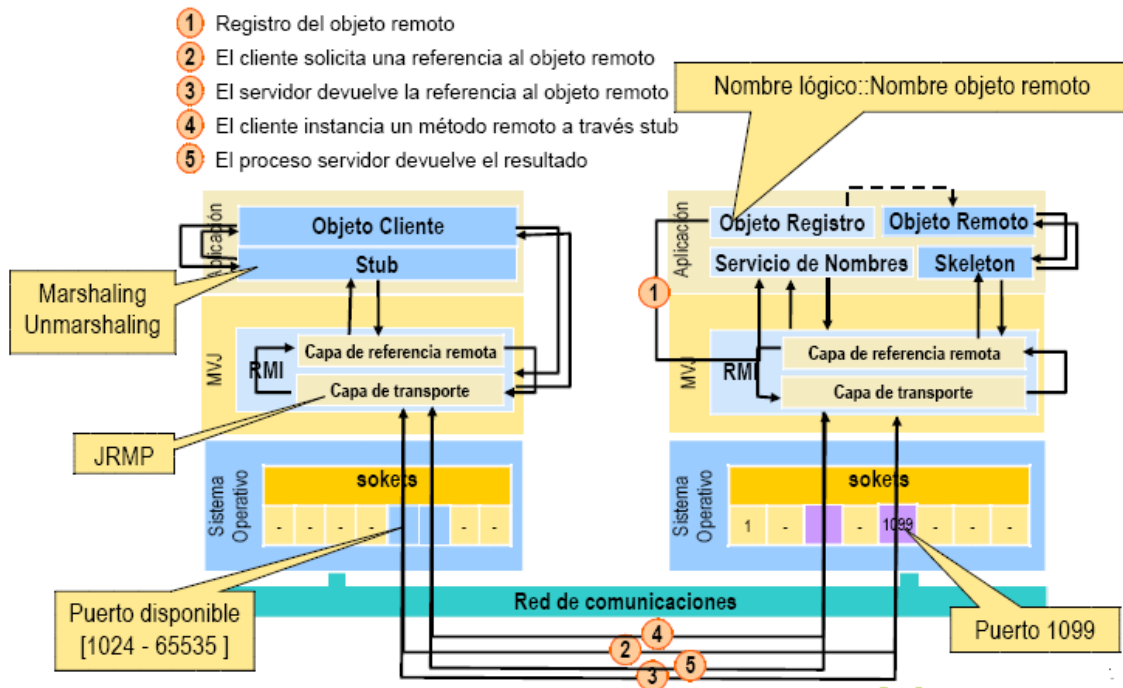
Es el equivalente a RPC pero para Java y orientado a objetos. Para la representación externa utiliza serialización. Se ejecuta sobre la máquina virtual de java (MVJ) con lo que puede implementarse sobre diferentes sistemas operativos. Los componentes básicos son: Interfaz remota, objeto remoto, clases proxies (stub y skeleton), servicio de registro, servicio de nombres y el cliente.

El servidor registra el objeto remoto, el cliente solicita una referencia al objeto remoto y el servidor se la devuelve mediante el servidor de nombres. El cliente instancia un método remoto a través del stub y el servidor devuelve el resultado a través del skeleton.

**Componentes de RMI:**

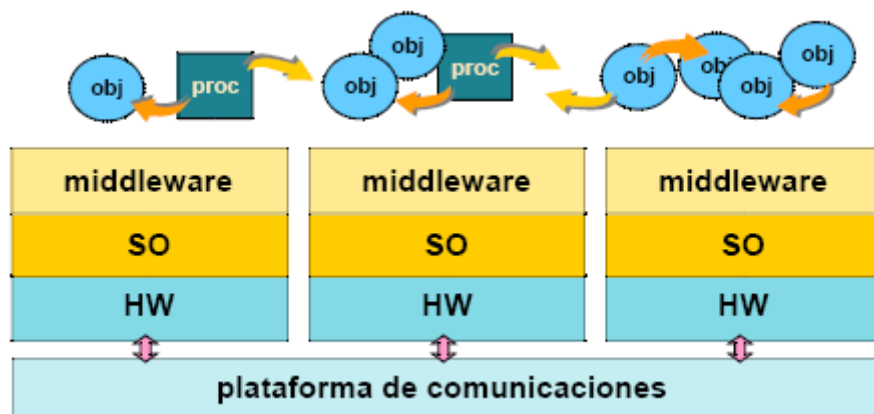
- **Interfaz remota:** Es la clase que se utiliza como plantilla para los clientes que vayan a invocar los métodos de un objeto remoto. Está compuesta por la declaración de los métodos remotos. Utiliza sintaxis RMI.
- **Objeto remoto:** contiene los métodos remotos.
- **Clase proxie stub:** La invocación de un método remoto por parte de un proceso cliente es dirigida a un objeto proxy, conocido como resguardo o stub. Esta capa se encuentra debajo de la capa de aplicación y sirve para interceptar las invocaciones de los métodos remotos hechas por los programas cliente. Una vez interceptadas, se envían a la referencia remota.
- **Clase proxie skeleton:** Se utiliza para interactuar con el objeto stub. Lee los parámetros de la invocación al método de enlace, realiza la llamada al objeto que implementa el servicio remoto, acepta el valor de retorno y devuelve el valor de retorno al stub.
- **Servicio de registro (rmiregistry):** Es un servicio de directorios para registrar el objeto remoto. Se ejecuta en el servidor del objeto.
- **Servicio de nombres:** Contiene los nombres de los servidores donde se hallan los objetos remotos.
- **Cliente:** Programa o usuario que invoca los métodos remotos.

## Arquitectura RMI:



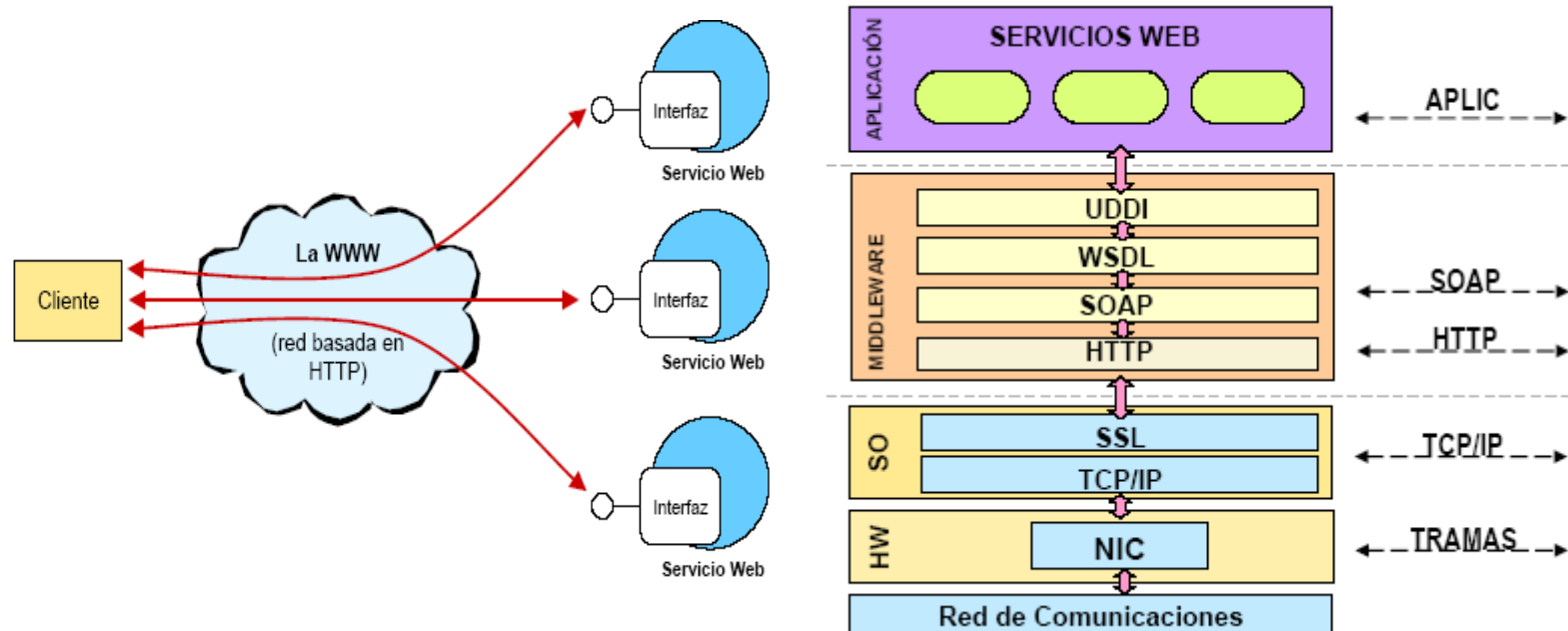
## Objetos distribuidos (ORB Object Request Broker)

Arquitectura que da soporte a la comunicación de procesos con procesos accediendo o comunicándose con diversos objetos a modo de middleware. Tiene un lenguaje para la definición de interfaces, tiene proxies (stub/skeleton), brinda una serie de serviditos por defecto como el de nombres, log, seguridad, entre otros. Posee un protocolo de interoperatividad. Este IPC es base para CORBA (especificación estándar de ORB) y los middlewares de J2EE, .NET Framework.



### Servicios web

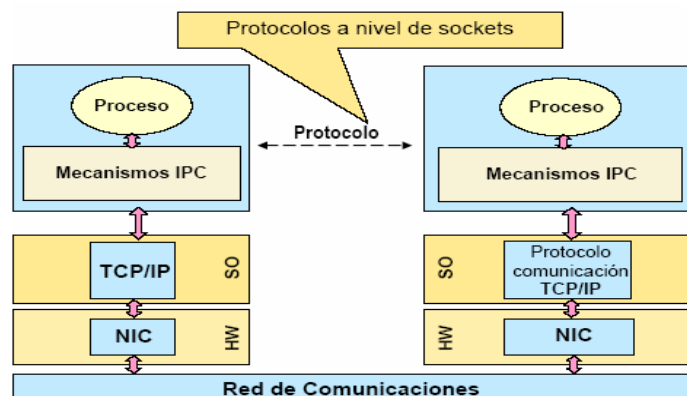
Se basan en un conjunto de estándares XML sobre protocolos de internet (HTTP, FTP). Lenguajes de definición de interfaces WSDL. Servicio de descubrimiento UDDI y protocolo de transporte SOAP.



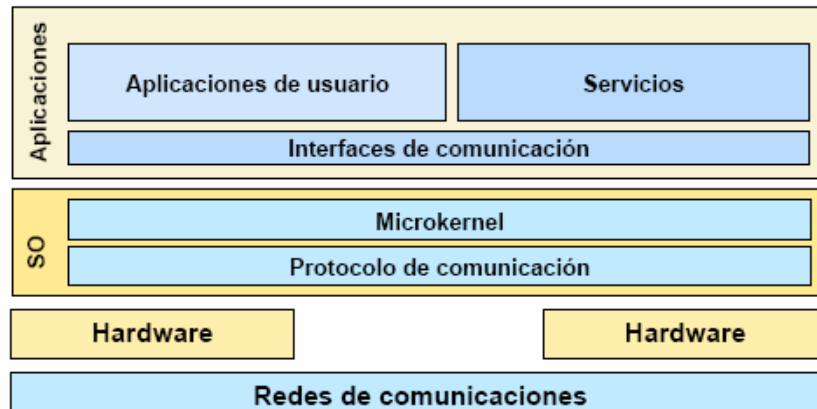
## 5. Enfoques

Un sistema operativo es la capa de software que separa las aplicaciones del hardware, controlando los recursos y brindando los servicios necesarios a las aplicaciones para una ejecución segura y eficiente. Los sistemas operativos se encargan de la gestión de los recursos del sistema y ofrece soporte de programación de aplicaciones distribuidas. Existen 3 tipos:

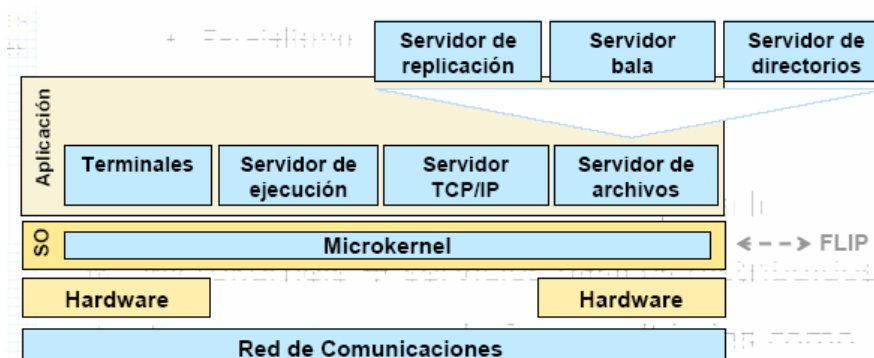
- **Sistema operativo en red:** donde el SW y HW están **débilmente acoplados**. Se ejecuta sobre una computadora, permitiendo que sus procesos accedan a recursos sobre computadoras remotas. No existe transparencia, ya que cada máquina del conjunto de la red es totalmente independiente. Los sistemas son **heterogéneos** a todos los niveles. El acceso a los recursos compartidos se realiza de manera explícita, mediante intercambio de información, ejecución de comandos remotos, transferencia de archivos, etc. Sobre este tipo de sistemas operativos, el desarrollo de aplicaciones distribuidas es más complejo debido a la poca cohesión entre las computadoras del conjunto. Para ello, se utilizan protocolos abiertos, tales como telnet, ftp, aunque el propio sistema operativo cuenta con protocolos de comunicación. Ej: **Windows, Linux**.



- **Sistema operativo distribuido:** donde el SW está *fuertemente acoplados* y el HW está débilmente acoplado. Existe un único sistema operativo que administra recursos sobre más de una computadora. Los objetivos son: escalabilidad, tolerancia a fallos, funcionamiento transparente y consistencia. Los **S.O** empleados son específicos y debe ser el **mismo en todos los nodos**. No hay tanta independencia entre las máquinas (empleo de microkernels que ofrecen servicios básicos distribuidos), aunque la ubicación de los recursos es independiente. Además son más fácilmente escalables. Ejemplos: **Amoeba, Mach,**



- **Sistemas distribuidos:** basados en el concepto de middleware y componentes de software distribuidos. Se ofrece una visión única del sistema mediante la abstracción en términos de servicio ofrecido por el middleware. Dado que los servicios y los protocolos están estandarizados, los sistemas operativos de los componentes del sistema son heterogéneos. Ejemplos: **Corba, .NET Framework, J2EE,...**
- **Amoeba:** es un ejemplo de sistema operativo distribuido implementado en la universidad de Vrije (Holanda) en 1981. Su funcionamiento se caracteriza por presentar un alto rendimiento, paralelismo, transparencia y funcionamiento distribuido. El término de máquina local desaparece debido a la implementación de microkernels en los sistemas operativos de los nodos para ofrecer una serie de servicios básicos distribuidos, además de la existencia de servicios que sirven para cubrir el resto de funcionalidades como tareas de usuario. Su funcionamiento se basa en un mecanismo RPC y mensajes (arquitectura C/S).



## TEMA 2: INTRODUCCIÓN AL MODELO DE OBJETOS DISTRIBUIDOS

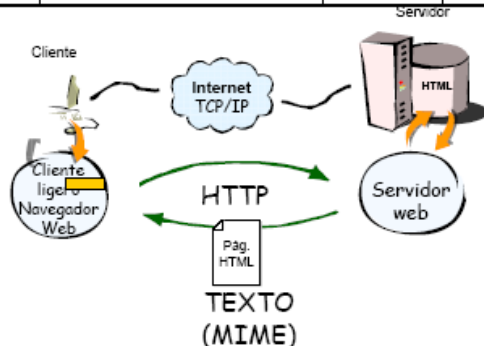
### 1. Tecnología Web

Internet permite que los usuarios accedan a los servicios y ejecuten aplicaciones sobre un conjunto heterogéneo de redes y computadores.

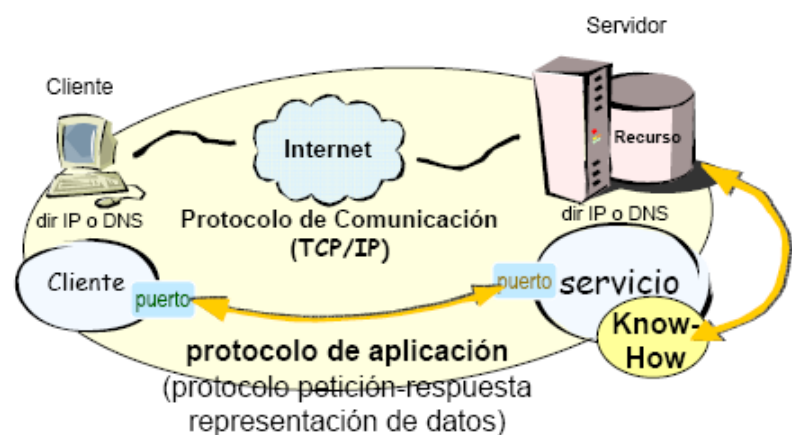
**Modelos básicos:** como modelos básicos tenemos el modelo cliente/servidor, el servicio de HTML básico y el modelo CGI.

#### Servicio HTML básico

Método	URI	Versión	Cabecera
GET	/index.html	HTTP/1.0	



#### Modelo C/S sobre Internet



**HTTP** es un protocolo orientado a conexión, sin estado y de petición-respuesta basado en texto. Se ejecuta por defecto en el puerto 80. Los principales métodos http son:

- **GET:** solicitar contenido (URI).
- **HEAD:** solicitar únicamente la cabecera.
- **POST:** envía datos al servidor.
- **PUT:** envía contenido para almacén (URI).

```

Línea  POST /cgi/miAplicacion.cgi HTTP/1.0
Cabecera  Accept: */*
          Connection: Keep-Alive
          User-Agent: Generic
          [línea en blanco]
Cuerpo    Nombre=Paco&eMail=pmacia@dtic.ua.es
    
```

#### Ejemplo de solicitud HTTP

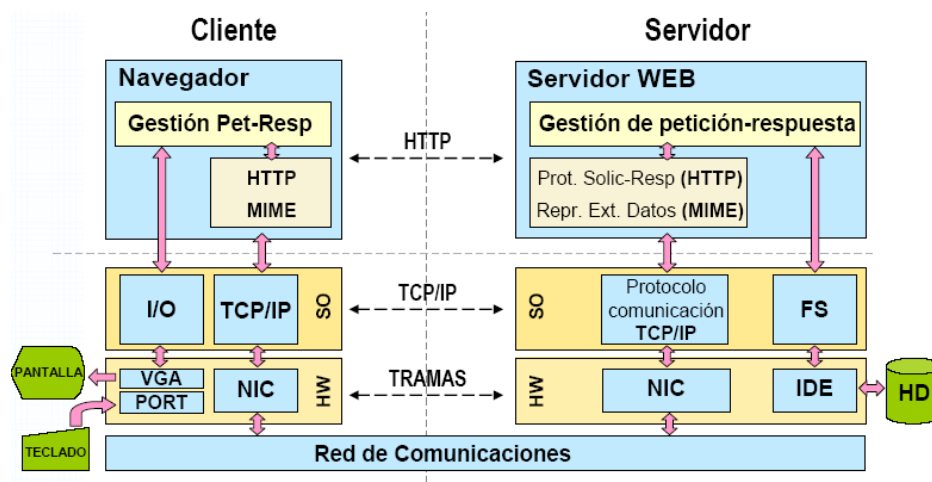
HTTP también tiene unos códigos de respuesta. Algunos son 1xx: mensajes de información, 4xx: error por parte del cliente, 5xx: error por parte del servidor, 3xx: redirección hacia otra URL, 2xx: operación exitosa.



El HTTP es un protocolo de transporte sin estado ¿Cómo podemos mantenerlo? (sesión) En el servidor se pueden introducir ficheros o BD que se encarguen de ello y en el cliente podemos utilizar cookies o campos ocultos.

**HTML** es un lenguaje de etiquetado utilizado para crear documentos que pueden ser recuperados empleando la WWW. Está basado en SGML. Puede representar noticias, correos, menús de opciones, resultados de BD...

Arquitectura HTML básico:



**MIME** se usa para enviar por correo electrónico cosas que no necesariamente son simple texto plano, nació para poder enviar archivos binarios por correo electrónico. Transforma los datos binarios en texto para poderlo transmitir. SMTP tradicional es adecuado para la transmisión de mensajes de texto en inglés, pero es inadecuado para texto en otro idioma o para datos no textuales.

Mime soporta un gran conjunto de tipos de contenido predefinidos, especificados con el formato tipo/subtipo.

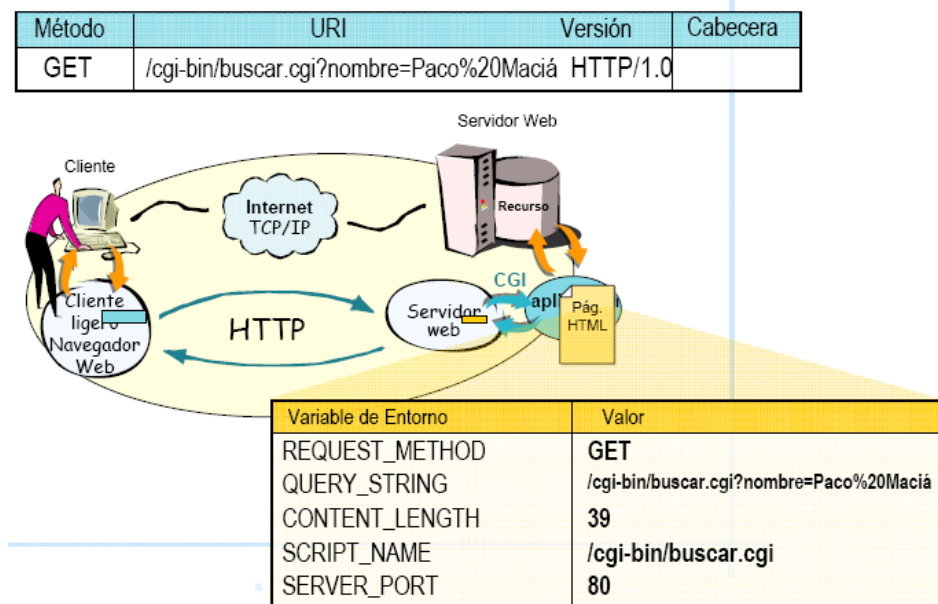
Tipo	Subtipo
Text	Plain, rich text, html, xml
Message	Email, news
Image	Jpeg, gif
Audio	Mp3, midi
Video	Mpeg, quicktime

MIME no sólo ha sido integrado en los programas de correo, sino que se ha integrado en el protocolo HTTP de la Web, de forma que los servidores Web pueden identificar los tipos de ficheros que tienen que enviar a sus clientes. A su vez los navegadores pueden conocer la codificación a aplicar con cada uno de los contenidos recibidos.



**GCI** para introducir acción a las páginas Web podemos utilizar el modelo CGI. Para utilizar un CGI se introduce como acción en un formulario. Los datos se le pueden pasar por GET o POST.

Su funcionamiento es como sigue: un navegador manda una petición de documento estático. El servidor busca el documento estático y devuelve su contenido al navegador dentro del cuerpo de la respuesta. El contenido contiene una llamada a un script Web denominado "algo.cgi". Cuando se presiona el botón "enviar", se manda una nueva petición al servidor Web especificando "algo.cgi" como objeto Web a llamar. El servidor comienza la ejecución del script pasándole los datos de entrada del navegador del usuario. En el transcurso de la ejecución se genera el contenido de la página Web como salida del programa y se transmite al servidor. Por último el servidor envía la página generada dinámicamente al cliente Web en el cuerpo de la respuesta.

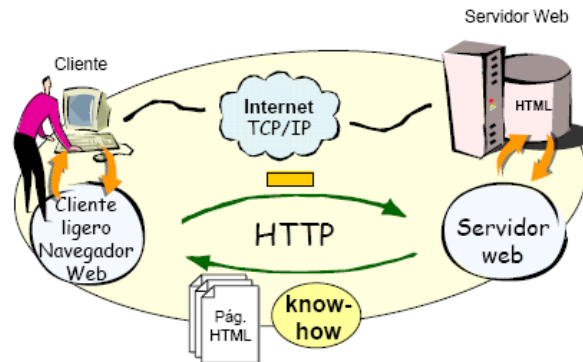


**Ventajas:** respuesta dinámica, libertad de elección del lenguaje de programación.

**Inconvenientes:** aplicación externa al servidor Web (no integrada), No existe control de ejecución, se tiene que instanciar en cada solicitud y sobrecarga recursos.

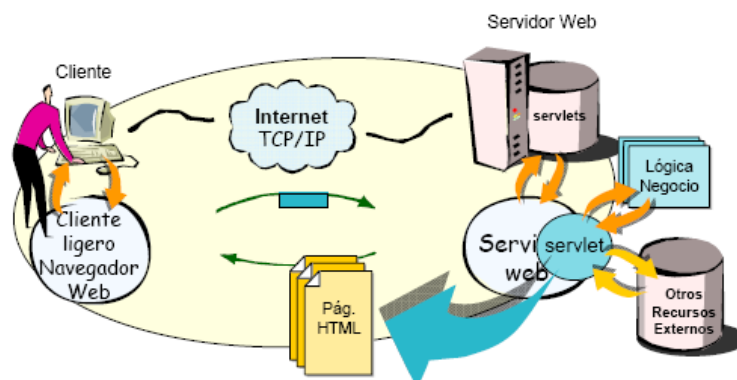
**Ampliaciones:** para una mayor participación en el mundo de los negocios se necesita generar contenidos dinámicos. Por tanto, es necesario ampliar el concepto de Web estática. Y para ello aparecen nuevas tecnologías en la parte del cliente (JavaScript, DHTML, ActiveX, Applets) y en la parte del servidor (Páginas activas: ASP, JSP, PHP, Serverlets, servidor de aplicaciones: EJB, COM+).

Las **ampliaciones en el cliente** son más fiables (contra ataques), seguro que tienen un resultado en la página final real. Por el contrario se pierde la compatibilidad (no sabemos en que plataforma en la que se está ejecutando y a lo mejor no va). Imposibilidad de obtener el Know-How debido a la plataforma o a versión no válida.

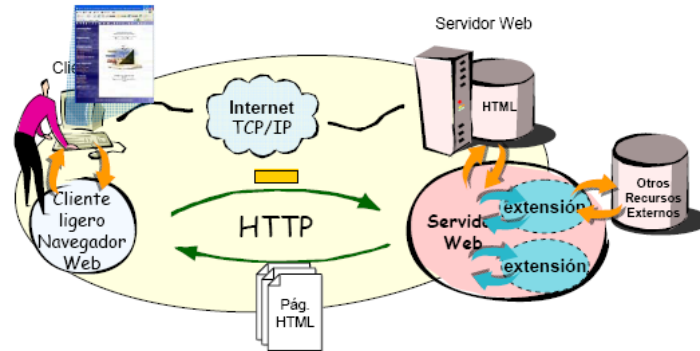


Las **ampliaciones en el servidor** sólo se instancian una vez, se gana en rendimiento y aprovechamiento de los recursos del equipo. Mantienen la sesión y se ejecutan en el servidor. Como desventaja tiene que algunas tecnologías son dependientes del fabricante (ASP). Y hay que conocer los diferentes lenguajes.

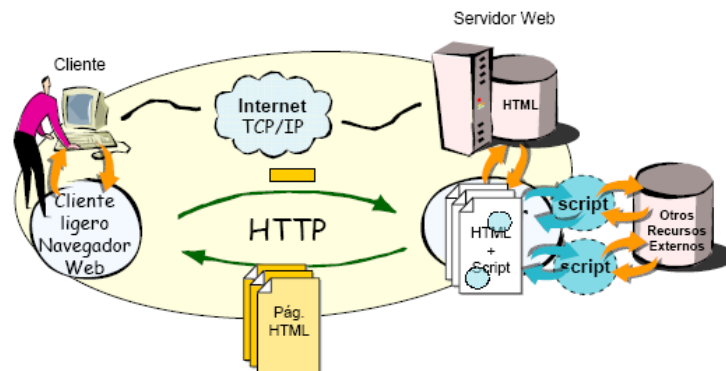
**Modelo de servlets:** cada servlet se ejecuta en el contexto proporcionado por el motor de servlets de la máquina servidora. El código de un servlet se carga en el motor de servlets. Posteriormente, el servidor actúa como intermediario entre el cliente y el servlet: el cliente realiza peticiones al servlet a través del servidor, y el servlet manda la respuesta al cliente a través del servidor. Dependiendo de la implementación del servidor, un servlet puede persistir mientras siga teniendo peticiones, o de forma indefinida hasta que se apague el servidor, debido a esta persistencia un servlet puede mantener datos de estado de las sesiones de los clientes durante su tiempo de vida. Tienen como ventaja la portabilidad, ya que solo necesitan el motor JVM.



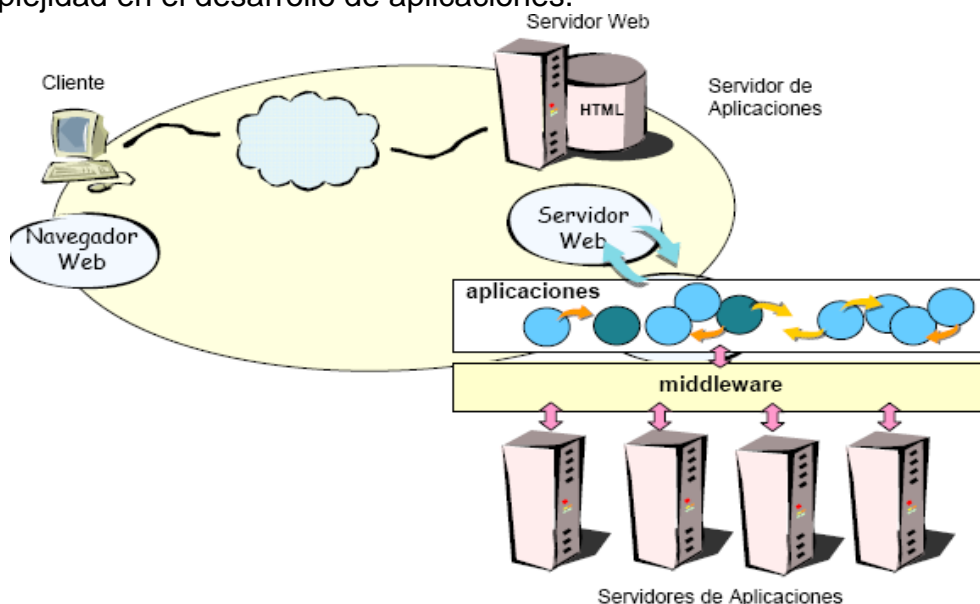
**Modelo de extensiones:** se logran servidores a medida, pero son dependientes de la tecnología, el servidor y la versión del mismo. Se gana en rendimiento y aprovechamiento de los recursos del equipo.



**Modelo de páginas activas:** una tecnología de páginas activas que permite el uso de diferentes scripts y componentes en conjunto con el tradicional HTML para mostrar páginas generadas dinámicamente. Habilita el concepto de sesión de usuario. Realiza una buena gestión de recursos externos como BD's. Realiza cierto control sobre el protocolo petición/respuesta. Las operaciones ejecutadas en el servidor Web independizan la aplicación del navegador.



**Servidor de aplicaciones:** es una aplicación que hace el papel de contenedor de aplicaciones. Se puede llamar servidor de aplicaciones tanto el HW como al SW. Un servidor de aplicaciones generalmente gestiona la mayor parte (o la totalidad) de las funciones de lógica de negocio y de acceso a los datos de la aplicación. Los principales beneficios de la aplicación de la tecnología de servidores de aplicación son la centralización y la disminución de la complejidad en el desarrollo de aplicaciones.



## 2. Middleware

**Punto de vista general:** conforma un nivel de abstracción en términos de servicios cuya finalidad es proporcionar una visión única del sistema, independientemente de la infraestructura (hw, S.O.) que lo forme. Se realiza todo de forma transparente.

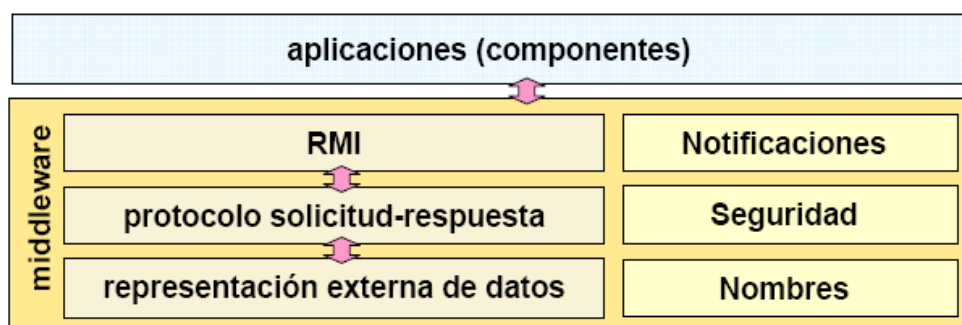
Capa de software intermedio entre el cliente y el servidor. Es la capa de software que nos permiten gestionar los mecanismos de comunicaciones. Ejemplo si se hace la petición de una página Web desde un browser en el cliente, el middleware determina la ubicación y envía una petición para dicha página. El servidor Web, interpreta la petición y envía la página al software intermedio, quien la dirige al navegador de la máquina cliente que la solicitó. 2 tipos:

- **Software intermedio general.** Servicios generales que requieren todos los clientes y servidores, por ejemplo: software para las comunicaciones usando el TCP/IP, software parte del sistema operativo que, por ejemplo, almacena los archivos distribuidos, software de autenticación, el software intermedio de mensajes de clientes a servidores y viceversa.
- **Software intermedio de servicios.** Software asociado a un servicio en particular, por ejemplo: software que permite a dos BD conectarse a una red cliente/servidor (ODBC: Conectividad abierta de BD), software de objetos distribuidos, por ejemplo la tecnología CORBA permite que objetos distribuidos creados en distintos lenguajes coexistan en una misma red (intercambien mensajes).

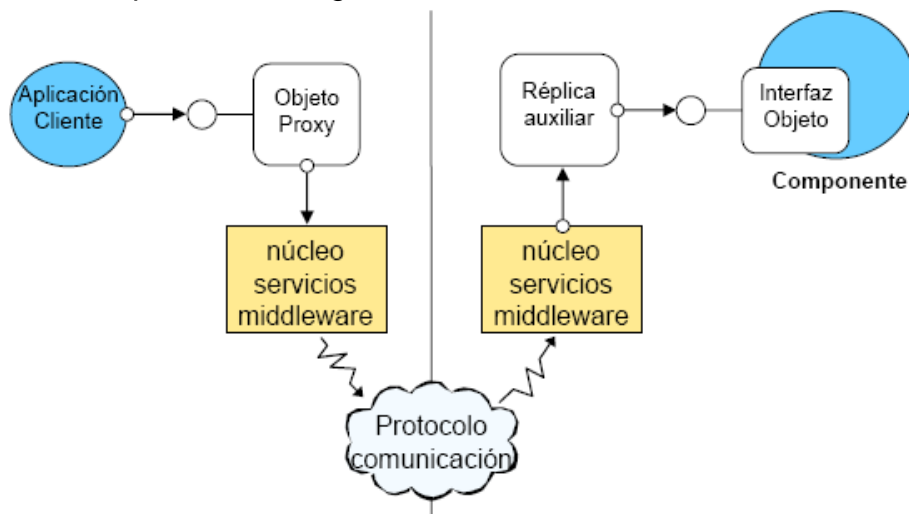
**Punto de vista de la arquitectura:** es un modelo de servicios ofrece **servicios** a la capa superior donde se ejecutaran las aplicaciones. Estas aplicaciones serán componentes que implementaran los servicios.

Los elementos de la arquitectura son:

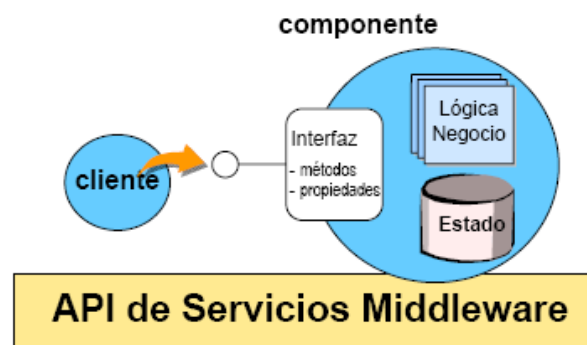
- Interfaz de los componentes (métodos + atributos).
- Proxy y réplica la interfaz de un componente (skeleton).
- Núcleo: localiza al Proxy, lo pone en memoria y se lo ofrece al cliente. Es el encargado de recoger las solicitudes, enviarlas a otro núcleo.
  - El modelo de representación externa de los datos
  - El modelo de comunicación entre componentes
  - Servicios: seguridad, nombres, eventos y notificaciones



La arquitectura queda como sigue:



**Punto de vista del programador:** el programador ve **código**. La estructura de un componente es: su interfaz (nombre de métodos+atributos), su estado (valores de los atributos, datos) y su comportamiento (lógica de aplicación, código en si). Para acceder a los componentes se hace a través del API (Interfaz):

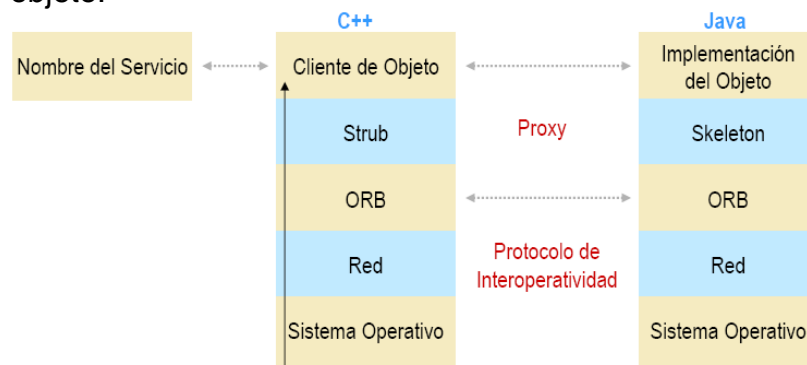


**Modelo de componentes** middleware brinda reutilización de código, transparencia con respecto a la plataforma sobre la que se ejecutan y con respecto al lenguaje de programación (.NET: C#, J#, VB, J2EE: Java y CORBA: independiente del leng.). Pueden programarse componentes de cliente y de servidor (encapsulado de servicios o almacén de datos). Existe una API que permite escribir componentes software en distintos lenguajes de programación.

Tienen capacidad de personalización a través de las propiedades y realizan la comunicación de forma **transparente** entre ellos y con el contexto mediante eventos.

**Plataforma middleware CORBA:** es una arquitectura estándar de objetos distribuidos diseñada para permitir que dichos objetos distribuidos interopereen sobre sistemas **heterogéneos** (Soporta multitud de plataformas y lenguajes de programación). CORBA proporciona un conjunto de protocolos y estándares, no una herramienta.

**La arquitectura de CORBA:** guarda una gran semejanza con la arquitectura RMI. Desde el punto de vista lógico, un cliente de objeto realiza una llamada a un método de objeto distribuido. El cliente interactúa con un Proxy, un stub, mientras que la implementación del objeto interactúa con un Proxy de servidor, un skeleton. A diferencia el caso de RMI, una capa adicional de software, conocida como ORB es necesaria. En el lado del cliente, la capa ORB actúa con intermediaria entre el stub y la red y S.O del cliente. En el lado del servidor sirve de intermediaria entre el skeleton y la red y el S.O del servidor. Utilizando un protocolo común, las capas ORB de ambos extremos son capaces de resolver las diferencias en lo referente a lenguajes de programación, así como las relativas a las plataformas. El cliente utiliza un servicio de nombrado para localizar el objeto.



Protocolos Inter-ORB:

- **ORB** (Object Request Broker): Capa Intermedia que da transparencia respecto a la plataforma.
- **IDL** (Interface Definition Language): Define mediante un lenguaje universal (similar a Java) las interfaces.
- **GIOP** (General Inter-ORB Protocol): Protocolo con el que interoperean los OMG's.
- **IIOP** (Internet Inter-ORB Protocol): Protocolo que se corresponde con un GIOP sobre TCP/IP.
- **IOR** (Interoperable Object Reference): Protocolo para referenciar objetos distribuidos. Un IOR es una cadena de caracteres donde se codifica:
  - El tipo del objeto.
  - El equipo donde reside.
  - El puerto del servidor del objeto.
  - Una clave identificativa.

**Servicios de objetos CORBA:** usados por los objetos distribuidos para construir aplicaciones:

- **Servicio de nombres:** servicio de nombres. Está basado en URL



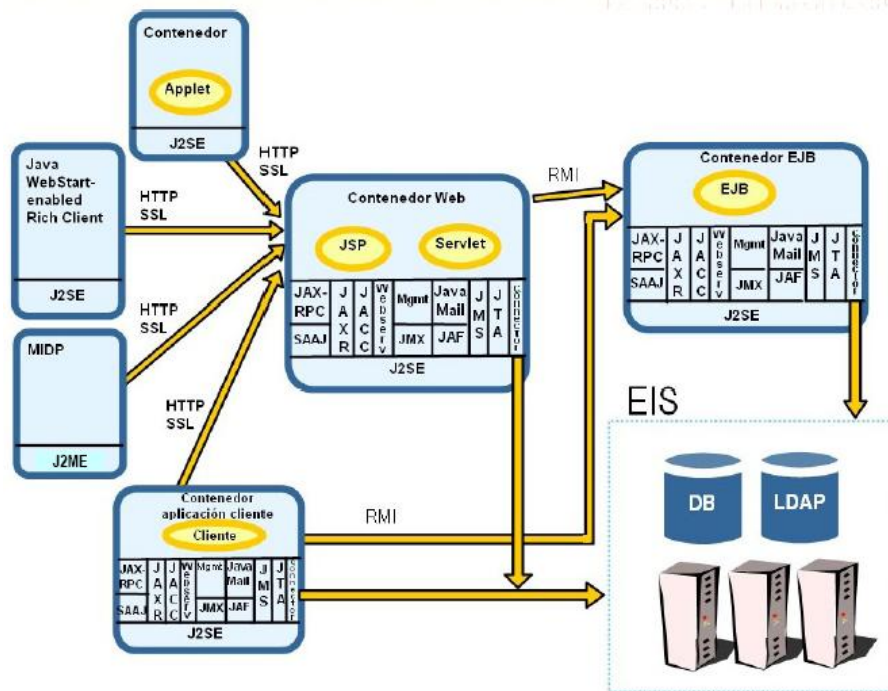
- **Servicio de concurrencia:** servicio que proporciona control de concurrencia.
- **Servicio de eventos:** servicio para la sincronización de eventos.
- **Servicio de log:** registro de eventos.
- **Servicio de planificación:** planificación de eventos.
- **Servicio de seguridad:** gestión de seguridad.
- **Servicio de negociación:** localización de servicios por tipo (no por nombre).
- **Servicio de tiempo:** eventos relativos al tiempo.
- **Servicio de notificación:** notificación de eventos.
- **Servicio de transacciones de objetos:** para procesamiento de transacciones.

**Plataformas middleware J2EE:** basado en Java. Ofrece tecnologías para el desarrollo de componentes:

- **Componentes Web:** que responden a solicitudes HTTP. JSP páginas que contienen código Java y Servlets programas Java que componen páginas Web.
- **Componentes Enterprise JavaBean (EJB):** componentes distribuidos. Unidades de software reusables que contienen la lógica de empresa. Existen 3 tipos de componentes que son:
  1. **Beans de sesión:** ejecuta solicitudes del cliente y es destruido cuando las operaciones del cliente se completan. Con estado: mantienen datos entre solicitud y solicitud. Sin estado: no mantienen estado entre solicitudes.
  2. **Beans de entidad:** Objeto persistente que modela los datos de un almacén
  3. **Beans dirigidos por mensaje:** no son invocados por el cliente. Procesan mensajes asíncronos.

Hay cuatro tipos de contenedores: Web (jsp,serverlets), EJB, Applets, Aplicaciones java estándar (cliente).

## Arquitectura J2EE:



## API's plataforma J2EE de SUN:

- **JCA:** arquitectura que para interactuar con una variedad de **EIS**, incluye ERP, CRM y otra serie de sistemas heredados. Medio estandarizado mediante el cual podemos acceder a una serie de sistemas heredados normal mente ERP's (SAP R/3, PeopleSoft).
- **JDBC:** acceso a **base de datos** relacionales. Api que nos proporciona la manera de conectar con bases de datos relacionales. Controlado por el contenedor.
- **JTA:** manejo y la coordinación de transacciones a través de EIS heterogéneos. Api para trabajar con **transacciones distribuidas**. Controladas por el contenedor de EJB's.
- **JNDI:** acceso a información en **servicios de directorio** y servicios de **nombres**. Proporciona los medios para ejecutar operaciones estándar en un recurso de servicio de directorio. Es utilizada para buscar interfaces para crear EJB's y conexiones JDBC...
- **JMS:** envío y recepción de mensajes. Proporciona la funcionalidad que nos permite enviar y recibir **mensajes** de manera asíncrona. Es importante en la integración de aplicaciones
- **JMail:** envío y recepción de **correo**. Nos permite de una forma muy sencilla enviar y recibir e-mail abstrayéndose de instalaciones. (IMAP4, POP3 y SMTP). Utiliza JAF para determinar los contenidos MIME y que operación debe realizar
- **JIDL:** mecanismo para interactuar con **servicios CORBA**.
- **RMI:** proporciona un mecanismo de acceso a **objetos remotos** como si fueran locales. Con RMI-JRMP solamente se podían instanciar remotamente objetos java. Con RMI-IIOP se pueden instanciar objetos CORBA.



- **JAAS**: proporciona un medio para **conceder permisos**, basado en quién ejecuta el código. Soporta módulos basados en diferentes sistemas de autenticación como Kerberos.
- **JAXP**: API para el manejo de **documentos XML**. Hace que su código sea independiente del procesador XML utilizado.
- **Otros APIs**: tratamiento de XML, integración con sistemas heredados utilizando Servicios Web,...

	J2EE	.NET
<b>Componentes</b>	EJB	.NET Component
<b>Mensajería</b>	JMS	MSMQ
<b>Transacciones</b>	JTA	MTS
<b>Acceso Distribuido</b>	RMI RMI-IIOP	.NET Remoting
<b>Conectores</b>	JCA	Host Integration Server, Biztlak Server
<b>Registro de Componentes</b>	JNDI	Active Directory
<b>Bases de Datos Relacionales</b>	JDBC	ADO.NET
<b>Manejo XML</b>	JAXP	MSXML

### 3. Integración

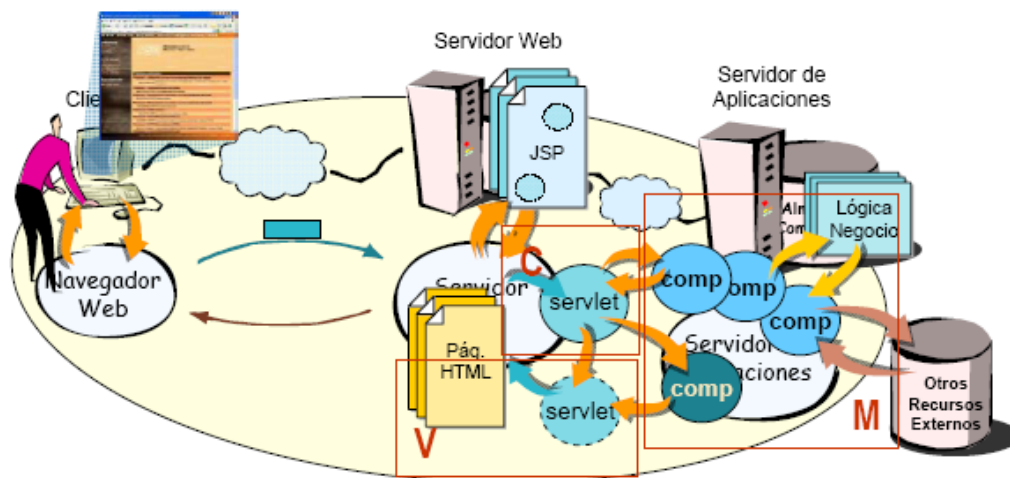
Integrar tecnologías Web y middleware aprovechando lo mejor de cada modelo. El modelo HTTP para la comunicación entre aplicación y el usuario (transparencia para el usuario). El modelo de componentes sobre servidor de aplicaciones con middleware para ejecutar la lógica de aplicación (transparencia en el servidor).

**El paradigma MVC**: (Modelo, vista, controlador) es un paradigma ampliamente difundido que proporciona un patrón de diseño que desacopla las funcionalidades específicas en tres partes.

- **Modelo**: Información de negocio y las reglas o funcionalidades. (aplicación)
- **Vista**: Lo que se muestra al usuario. (Página Html).
- **Controlador**: Elemento central que recibe las peticiones del cliente y las pasa a la vista para mostrarlas. (Servlets).

MVC facilita el mantenimiento, reusabilidad y adaptabilidad de los componentes. Es independiente de infraestructuras, capas de aplicación y plataformas de desarrollo. Es adecuado para constituir un equipo multidisciplinar y así se pueden elegir herramientas para cada tarea.

Hay diferentes combinaciones: EJB+JSP+Servlets, JavaBeans+XML+Servlets.

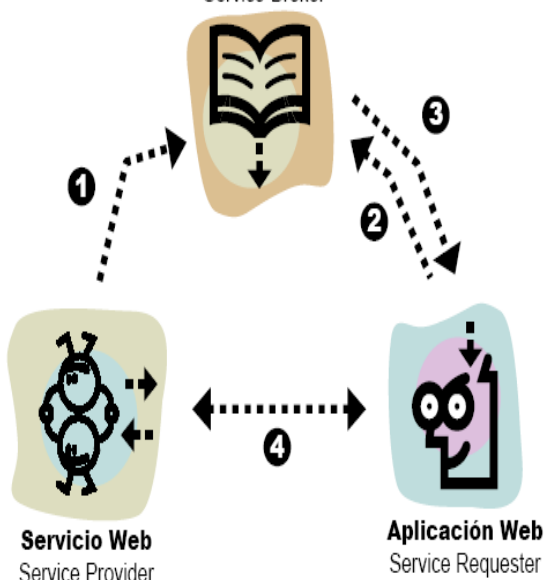


## 4. Servicios Web

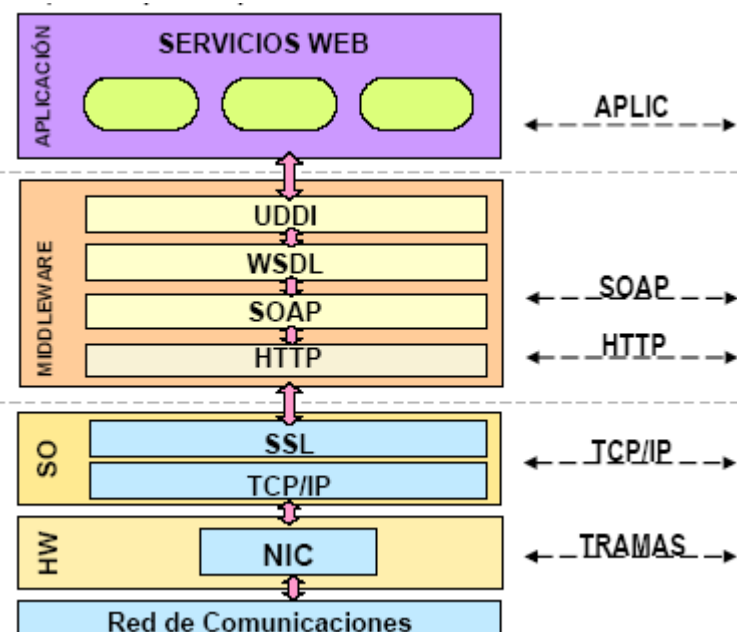
**Definición:** componentes que ejecutan procesos con una interfaz claramente definida y accesible a través de Internet, basada en el intercambio de documentos electrónicos en formato XML, y que pueden ser combinados entre sí. XML proporciona una forma transparente para el intercambio de datos (representación externa estándar). La diferencia entre un servicio Web y una aplicación Web se ve claramente en la frase: "Se puede llamar a un servicio Web XML desde una aplicación Web ASP.NET". Una aplicación Web es toda aquella que utiliza tecnologías Web.

Son los elementos fundamentales en la evolución hacia la computación distribuida a través de Internet. Se están convirtiendo en la plataforma de integración de aplicaciones gracias a los estándares abiertos (XML, SOAP, UDDI y WSDL) y a que varios Web Services de origen distinto que funcionan conjuntamente, sin importar su ubicación o la forma en que se implementaron.

Registro UDDI  
Service Broker



### Arquitectura:



La idea general es:

- Los Web Services ofrecen funciones empleando un protocolo Web estándar que, en casi todos los casos, es **SOAP**.
- Los servicios XML Web Services permiten describir sus interfaces con suficiente detalle para que el usuario diseñe una aplicación cliente que permita comunicarse con ellas. Esta descripción se proporciona normalmente en un documento XML denominado **WSDL**.
- Los servicios XML Web Services se registran para que los futuros usuarios los encuentren fácilmente. Este registro se realiza a través de **UDDI**.

**WSDL**: (Web Service Description Language) un documento XML para la descripción de servicios Web, dónde se ubican, y como se pueden invocar. Describe un conjunto de mensajes SOAP y la forma en la que éstos se intercambian (las interfaces). Utiliza un IDL (Interface Definition Language) de servicios, no solo escribe la interfaz sino además las características de localización e implementación. Los registros UDDI apuntan a una hoja WSDL.

Algunas etiquetas del XML:

- **<types>**: proveen definiciones de los tipos de datos utilizados para describir los mensajes intercambiados.
- **<definitions>**: contiene la definición de uno o más servicios. Secciones conceptuales:
  - **<message>**: representa una **definición abstracta de los datos** que están siendo transmitidos. Un mensaje se divide en una serie de partes lógicas, cada una de las cuales se asocia con alguna definición de algún sistema de tipos.
  - **<portType>**: conjunto de **operaciones abstractas**. Cada operación hace referencia a un mensaje de entrada y uno de salida.
  - **<binding>**: especifica un protocolo concreto y las especificaciones del formato de los datos de las operaciones y los mensajes definidos por un portType en concreto, cómo se **invocan las operaciones**.
  - **<service>**: dónde se **ubica el servicio**. Para unir un conjunto de puertos relacionados.
    - **<port>**: especifica una dirección para un binding, para así definir un único nodo de comunicación.
- **<documentation>**: puede contener información del servicio para el usuario.

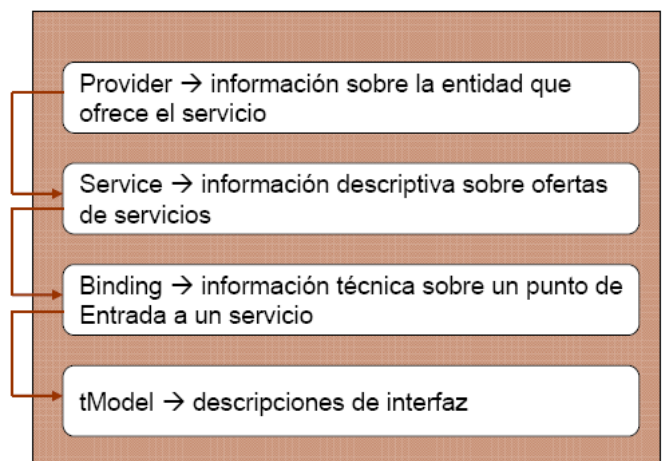
**UDDI** (Universal Description, Discovery and Integration) permite descubrir servicios (con quién comunicarse y dónde).

Una entrada en un listín UDDI es un archivo XML que describe un servicio Web y los servicios que ofrece. Cada entrada tiene tres secciones conceptuales:

- Las "**páginas blancas**" similar a la información que aparece en el directorio telefónico como nombre, dirección, información de contacto, etc.
- Las "**páginas amarillas**" incluyen las categorías industriales, ubicación geográfica. Similar al listín telefónico.
- Las "**páginas verdes**" describen la interfaz del servicio con información suficiente para que alguien escriba una aplicación para usar un servicio Web. (información técnica).

#### Estructura central de un registro UDDI

- **businessEntity**: información sobre un negocio o entidad. Utilizada por el negocio para publicar información descriptiva sobre si mismo y los servicios que ofrece
- **businessService**: servicios o procesos de negocios que provee la estructura businessEntity
- **bindingTemplate**: datos importantes que describen las características técnicas de la implementación del servicio ofrecido
- **tModel**: especificación técnica



#### Características de UDDI: dos categorías de API:

- **De publicación**: mecanismo para que los proveedores de servicios se registren (ellos mismos y sus servicios) en el Registro UDDI.
- **De consulta**: permite a los subscriptores buscar los servicios disponibles y obtener el servicio una vez localizado

**SOAP**: (Simple Object Access Protocol) es el protocolo ligero basado en XML para el intercambio de información en un entorno descentralizado y distribuido. Facilita la intercomunicación entre objetos de cualquier tipo, sobre cualquier plataforma, en cualquier lenguaje. Comunicación (débilmente acoplada) máquina-a-máquina. Una petición/respuesta de SOAP puede viajar sobre cualquier protocolo de aplicación: HTTP, SMTP, permite el intercambio a través de firewalls.

#### Petición de SOAP en XML: tres partes:

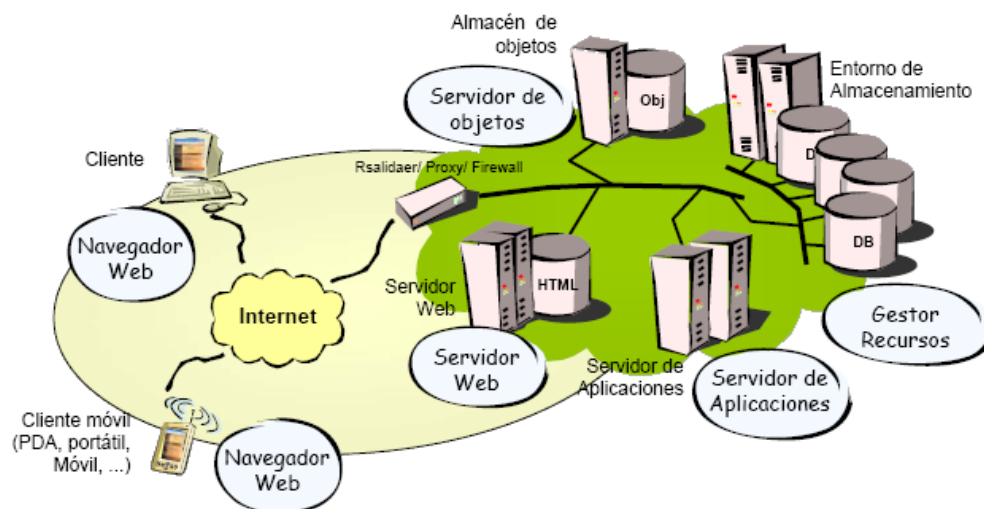
- **Sobre ó envoltura**: define un marco de referencia general para expresar qué hay en el mensaje, quién debe de atenderlo, y si es opcional u obligatorio. Identifica un mensaje XML como SOAP.

- **Reglas de codificación:** definen un mecanismo de serialización que se puede utilizar para intercambiar instancias de tipos de datos definidos por la aplicación.
- **Representación RPC:** define una convención que puede ser utilizada para representar las llamadas y respuestas a procedimientos remotos.

Respuesta SOAP/http:documento XML dentro de una respuesta HTTP estándar.

**RPC mediante SOAP:** uno de los objetivos de diseño de SOAP Version 1.2 es la encapsulación de la funcionalidad de las llamadas a procedimientos remotos utilizando la extensibilidad y funcionalidad de XML.

Información necesaria para la invocación RPC mediante SOAP: dirección de nodo SOAP destino. Nombre del método o procedimiento. Identidades y valores de argumentos que se deben pasar y parámetros de salida y su valor. Separación clara de los argumentos utilizados para identificar el recurso Web que es el destino real para la RPC, en contraste a aquellos que transportan datos o información de control utilizada para que el recurso de destino procese la llamada. Patrón de intercambio de mensajes que será utilizado para el transporte junto con la identificación del Método Web. Opcionalmente, los datos que deben ser transportados como parte de bloques de encabezado SOAP



## TEMA 3: SERVICIOS DE NOMBRES

### 1. Servicios de nombres

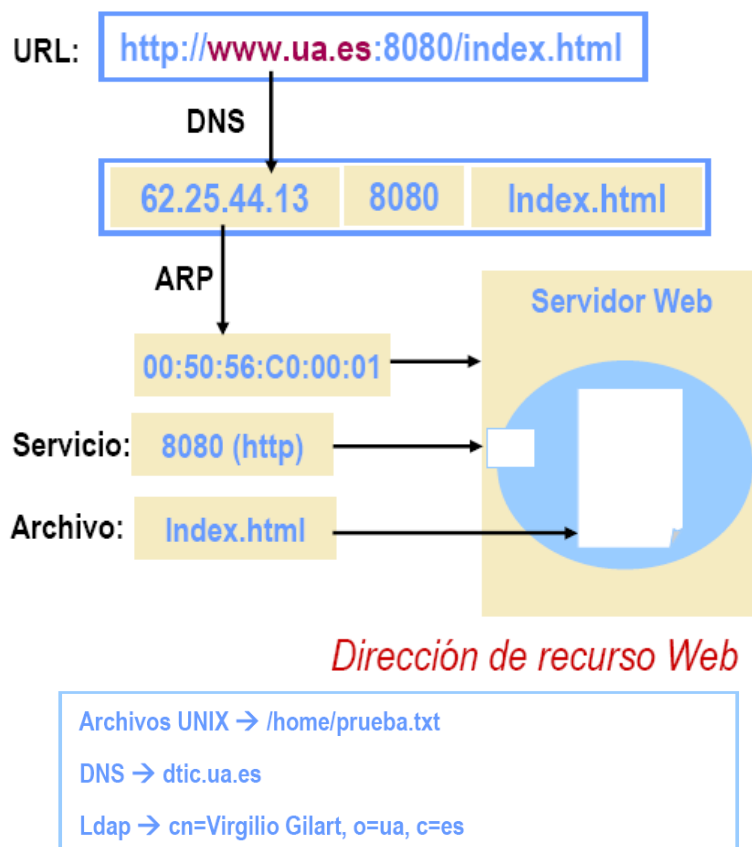
**Definición:** es una BD con información que facilita la localización de un recurso ya que contiene la referencia a recursos independientemente de su localización (ej.: sólo conociendo su nombre, no su localización IP).

En un sistema distribuido, los nombres se utilizan para hacer referencia a una amplia variedad de recursos, como computadoras, servicios, objetos remotos y archivos, así como a los usuarios., emails, extensiones, etc. Esto hace posible la comunicación y el compartimiento de recursos de los sistemas operativos distribuidos, ya que facilitan su lectura.

Un nombre está resuelto cuando está traducido a datos relacionados con el recurso u objeto nombrado. Forma un enlace **nombre lógico:: Atributo** utilizando alguna convención de nombrado.

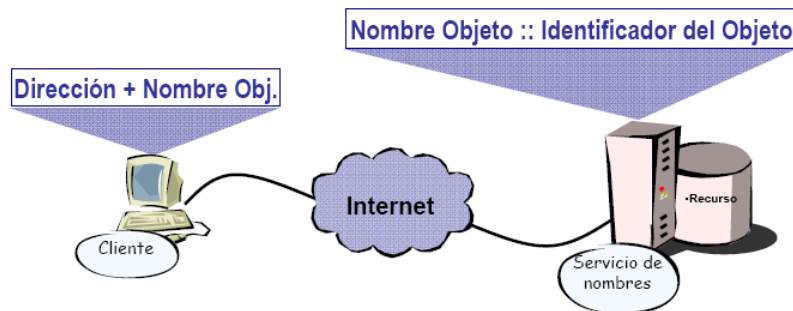
- Archivos (sistemas de archivos) -> /home/prueba.txt.
- Objetos remotos (jndi, rmiregistry).
- Computadoras (dns) -> dtic.ua.es.
- Servicios (url).
- Usuarios (servicios de directorios, LDAP).

Un **atributo** es el valor de una propiedad asociada a un objeto, siendo un atributo clave la dirección. La dirección puede ser considerada a menudo simplemente como otro nombre que debe ser buscado o que puede contener dicho nombre.





Los **contextos** son conjuntos de enlaces entre nombres textuales y atributos de objetos (usuarios, computadores, servicios, objetos remotos, ...), es decir, **nombre::objeto**.



El **espacio de nombres** es un conjunto de nombres válidos en un sistema de nombrado. Es la colección de todos los nombres válidos reconocidos por un servicio particular. Que un nombre sea válido significa que el servicio intentará su búsqueda, aunque el nombre esté desvinculado.

Un **servicio de nombrado** devuelve un atributo a partir de un nombre conocido. Almacena una colección de uno o más contextos de nominación. Forman una base de datos cuya principal tarea que facilita es la resolución de un nombre, es decir la búsqueda de atributos dado un cierto nombre, y la localización de recursos. La **información** se almacena **jerárquicamente**. La jerarquía y la **estructura** se definen en función de las necesidades de la organización.



### Características

- **Base de datos optimizada:** la base de datos está orientada a la lectura de información para agilizar las consultas (que son la mayoría de operaciones). Los datos de una entrada ocupan un único registro. No son necesarias las transacciones ni bloqueos.
- **Modelo distribuido de almacenamiento de información:** dado que es posible distribuir la información de manera no centralizada, la flexibilidad es mayor.
- **Débil consistencia** de replicación entre servidores de directorios (servidores secundarios).
- **Escalabilidad:** posibilidad de aumentar el tamaño de la base de datos.
- **Alta disponibilidad,** de los datos de la Base de Datos.

**DNS:** establece una jerarquía de nombres para nodos en redes TCP/IP. Asocia cada nombre con una IP. Todo un sistema basado en BD distribuida que permite obtener una IP a partir de un nombre de dominio (resolución directa) o viceversa (resolución inversa). Los nombres DNS se denominan **nombres de dominio** y son cadenas similares a los nombres absolutos de archivos en UNIX.

El espacio de nombres DNS tiene una estructura jerárquica: un nombre de dominio está formado por una o más cadenas separadas por puntos, llamadas componentes de nombre o etiquetas. No hay delimitadores de principio y fin. Un prefijo de un nombre es una sección inicial del nombre que contiene cero o más componentes completos. Algunos ejemplos son: **dccia.ua.es**.

La raíz de esta jerarquía es administrada por el Internet Network Information Center (interNIC), empresa que asigna nombres a cada dirección IP única. Ya desde los inicios, en un sencillo archivo HOST.TXT figuraban los pocos centenares de servidores que la componían. Para realizar cambios en este fichero, los administradores de los diferentes servidores enviaban las modificaciones por correo electrónico y recibían el nuevo fichero HOST.TXT actualizado por FTP.

#### **Elementos:**

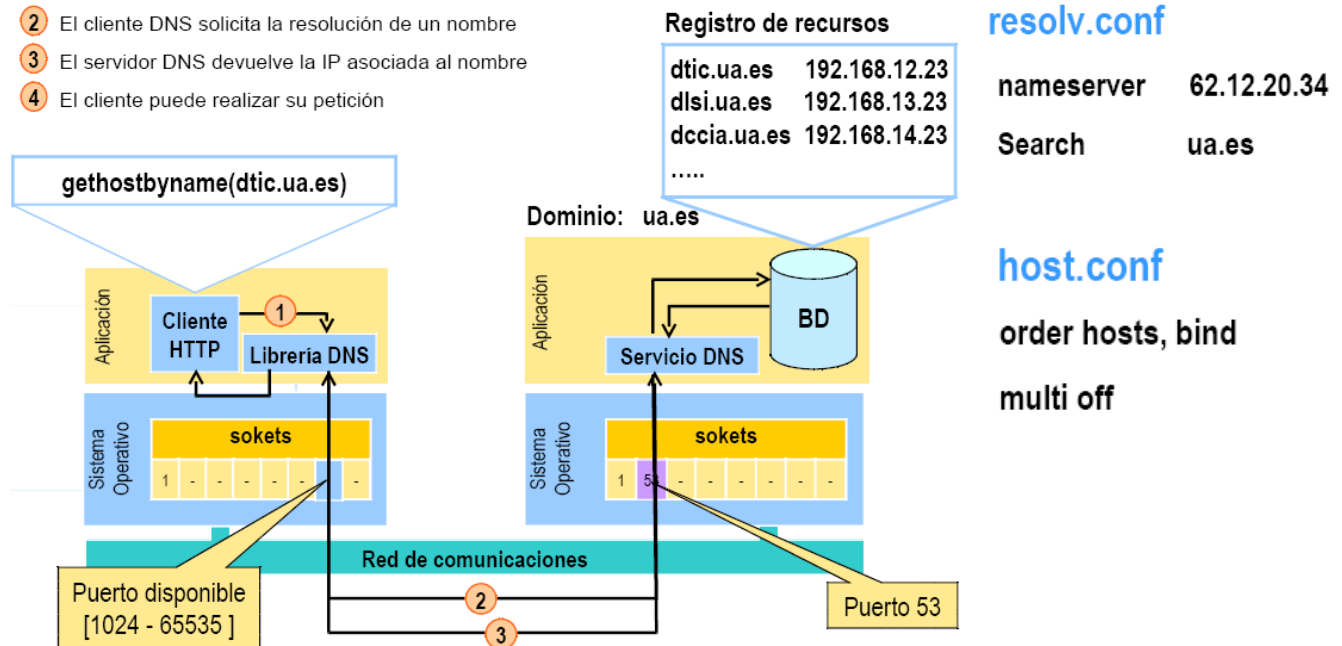
- **Espacio de nombres:** jerarquía estructurada de dominios para organizar los nombres. En UNIX es (bin, home, etc,...).
- **Registros de recursos:** asignan nombres a un tipo específico de información de recurso (utilizada para resolver el nombre del espacio de nombres). Ejemplo: [nombre][TTL][clase] Tipo de registro Valor del dato.
- **Servidores DNS:** almacenan y responden a las consultas de nombres.
- **Clientes DNS:** consultan a los servidores para buscar y resolver nombres de un tipo de registro de recursos.

#### **El funcionamiento es el siguiente:**

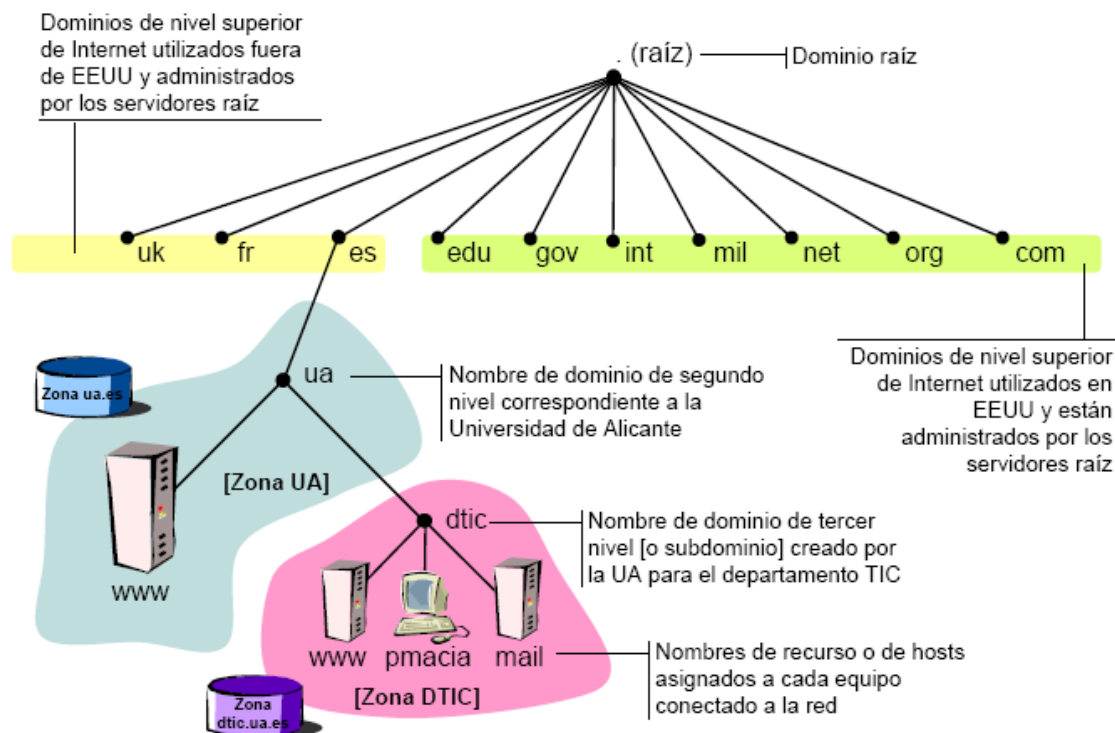
1. El cliente hace la petición `http://dtic.ua.es`
2. La librería DNS llama a `gethostbyname(dtic.ua.es)` por el puerto 53.
3. El servidor busca en la BD `dtic.ua.es` y devuelve su IP.
4. Finalmente el cliente obtiene la dirección del dominio que buscaba.



- 1 Petición de un cliente
- 2 El cliente DNS solicita la resolución de un nombre
- 3 El servidor DNS devuelve la IP asociada al nombre
- 4 El cliente puede realizar su petición



**Dominios, subdominios y zonas:** una zona se refiere a la gestión de un dominio o subdominio. Siempre tiene un fichero que configura los recursos existentes en la zona. Un dominio es una agrupación de nombre. Una zona puede gestionar varios dominios y subdominios.



### Ejemplo de configuración:

#### /etc/named.conf

```
options {
    directory "/var/lib/named"
    allow recursion {
        183.165.75.0/24;
    };
    allow transfer {
        150.165.43.176;
    };
    forward first;
    forwarders {
        232.154.178.35;
        157.246.33.2;
    };
};
```

```
zone "." {
    type hint;
    file "root.hint";
};

zone "ua.es" {
    type master;
    file "zone/ua.es";
};

zone "75.165.183.in-addr.arpa" {
    type master;
    file "zone/183.165.75";
};

zone "dtic.ua.es" {
    type forward;
    forwarders {192.168.1.1}
};
```

### Tipos de registros:

Tipo de registro	Significado	Contenidos principales
A	Una dirección de computador	Número IP
NS	Un servidor de nombres autorizado	Nombre de dominio para un servidor
CNAME	El nombre canónico de un alias	Nombre de dominio para un alias
SOA	Marca el comienzo de datos en una zona	Parámetros que gobiernan en una zona
WKS	Una descripción de servicio bien conocido	Lista de nombres de servicio y protocolo
PTR	Puntero de nombre de dominio (búsquedas inversas)	Nombre de dominio
HINFO	Información de host	Arquitectura de la máquina y del sistema operativo
MX	Intercambio de correo	Lista de pares <preferencia, host>
TXT	Cadena de texto	Texto arbitrario

### Ejemplo:

pc01.dtic.ua.es. IN A 192.168.1.1  
 pc02.dtic.ua.es. IN A 192.168.1.2  
 1.1.168.192.in-addr.arpa. IN PTR pc01.dtic.ua.es.

**Formato de los mensajes:** los mensajes que se transmiten entre el cliente y el servidor DNS son bastante simples. Cuentan con:

- **Identificador:** para que el cliente compruebe la respuesta.
- **Parámetro:** que es el la operación solicitada o el código de respuesta.

Aparte cuentan con otros campos de información adicionales:

Campos adicionales

Parámetros

Identificación	Parámetro
Número de solicitudes	Número de RR de respuesta
Número de RR de autorización	Número de RR adicionales
Solicitudes	
Respuestas	
Autorización	
Información Adicional	

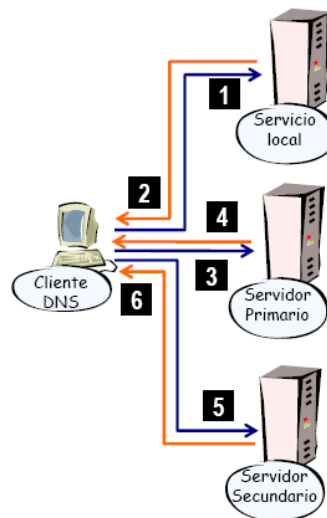
Bits	Descripción
0	Tipo de Operación: 0 solicitud; 1 respuesta
1-4	Tipo de Solicitud: 0 estándar, 1 inversa
5	Activado si el servidor de nombres que responde tiene autoridad para el nombre de dominio enviado en la consulta.
6	Activado si el mensaje está truncado
7	Activado si desea recursión
8	Activado si la recursión está disponible
9-11	Reservado para uso futuro
12-15	Tipo de respuesta: 0 sin error; 1 error de formato; 2 fallo en el servidor; 3 el nombre no existe; 4 consulta no disponible; 5 rechazado por el servidor por políticas

**Resolución:** la resolución es un **proceso iterativo** en el que se presenta de forma repetitiva un nombre sobre los contextos de nominación. Para resolver un nombre, en primer lugar se presenta sobre un contexto de nominación inicial; el proceso de resolución se reitera mientras se generen otros contextos y otros nombres derivados. Un contexto puede asociar un cierto nombre sobre un conjunto de atributos básicos, bien de forma indirecta o bien asociando el nombre sobre otro contexto de nominación y sobre otro nombre derivado para ese contexto.

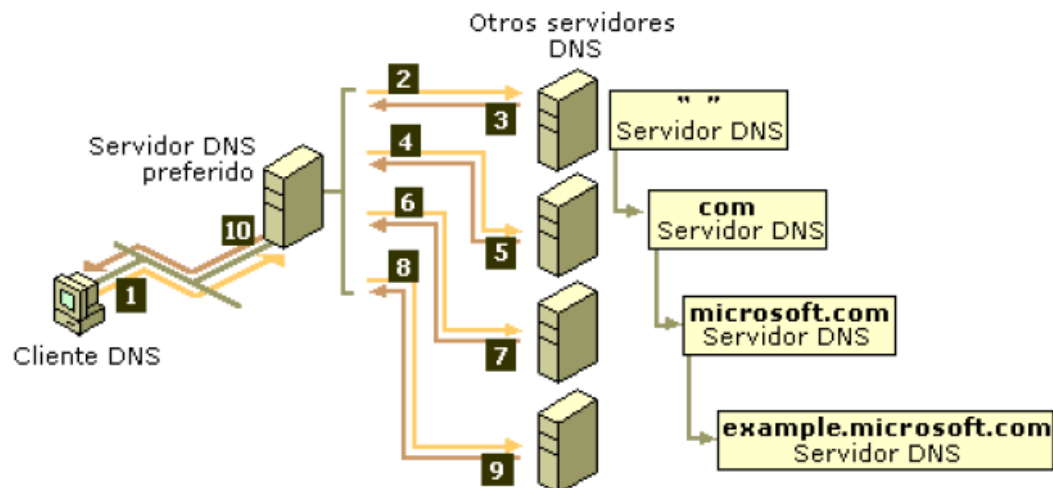
La resolución iterativa consta de lo siguiente: para resolver un nombre, un cliente lo presenta al servidor de nombres local, el cual intenta resolverlo. Si el servidor de nombres local tiene dicho nombre, devuelve el resultado inmediatamente. Si no es así, se lo envía a otro servidor capaz de ayudarlo. La resolución se propaga en caso de que en este servidor tampoco se halle, hasta que localice el nombre o bien descubra que no existe.

## Métodos de resolución

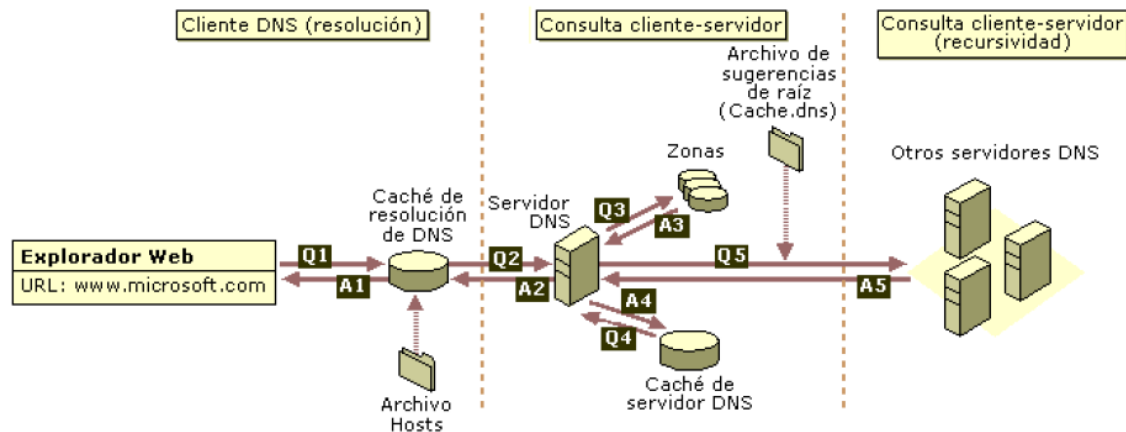
- Resolución directa (recursiva?): dado un DNS, localizar su dirección IP asociada.
- Resolución inversa (recursiva?): dado una dirección IP, localizar el nombre DNS asociado.
- Resolución iterativa gestionada por el propio cliente.



- Resolución iterativa gestionada por el propio servidor.



- Resolución recursiva gestionada por el servidor.



**Caché:** en DNS y otros servicios de nombres, el software de resolución de nombre del cliente y los servidores mantienen una caché de resultados de resoluciones previas. Cuando un cliente solicita la búsqueda de un nombre, el software de resolución de nombres consulta su caché. Si encuentra un resultado reciente de una búsqueda previa de dicho nombre, la devuelve al cliente, en otro caso se la envía al servidor.

El almacenamiento de datos en caché es clave en la prestaciones de un servicio de nombres y ayuda en el mantenimiento de la disponibilidad, tanto del servicio como de otros servicios que seguirán funcionando incluso después de que el servidor de nombres falle. Se consigue una mejora en el tiempo de respuesta gracias al ahorro de comunicaciones, ya que, además, cuenta con las resoluciones ya realizadas. Ofrece una alta disponibilidad y consistencia de respuesta o bien devuelve el más cercano al dominio.

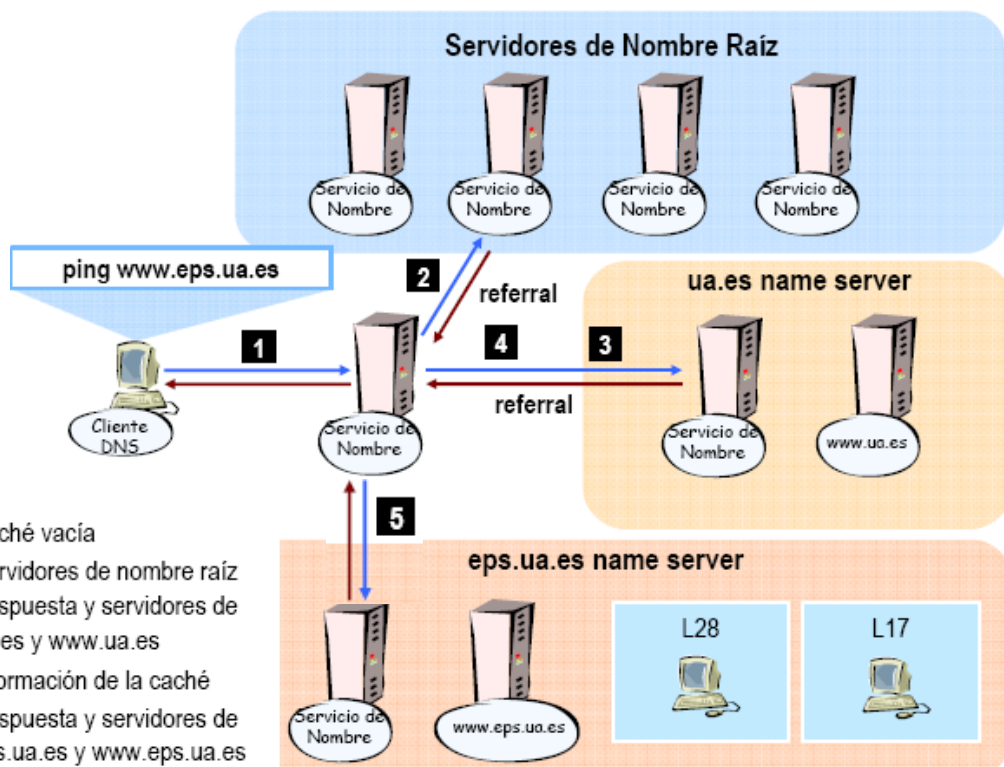
# 1 Caché del cliente

IP de las peticiones

Servidores que las resuelven

ipconfig /flushdns

ipconfig /displaydns



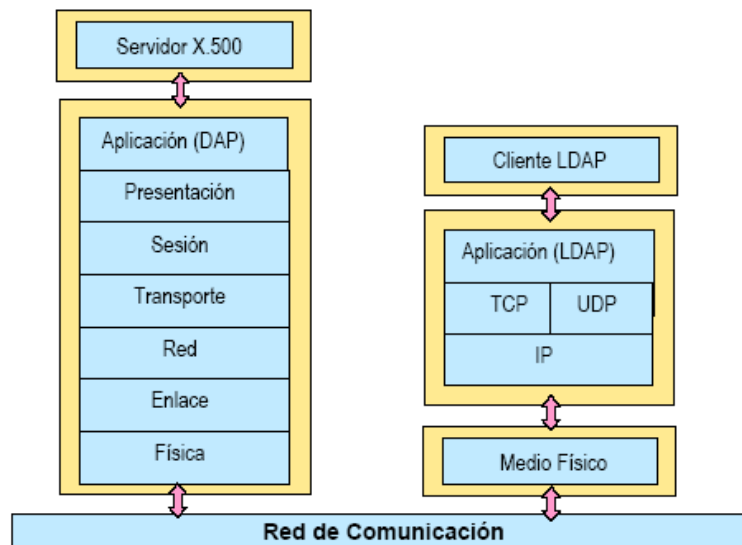
- 1 Caché vacía
- 2 Servidores de nombre raíz
- 3 Respuesta y servidores de ua.es y www.ua.es
- 4 Información de la caché
- 5 Respuesta y servidores de eps.ua.es y www.eps.ua.es

## 2. Servicios de directorio

**X.500:** es un protocolo que especifica un **modelo de conexión** de servicios de directorio locales para formar un directorio global distribuido en Internet. Es el intento de impulsar la construcción de un servicio de Directorio a nivel mundial, totalmente distribuido, para proporcionar servicio tanto a personas como a máquinas. Esta normativa desarrollada conjuntamente por ISO y el CCITT. Usado en la mayoría de los casos como páginas blancas

Los protocolos definidos por X.500 incluyen, protocolo de acceso al directorio (DAP), el protocolo de sistema de directorio, el protocolo de ocultación de información de directorio, y el protocolo de gestión de enlaces operativos de directorio.

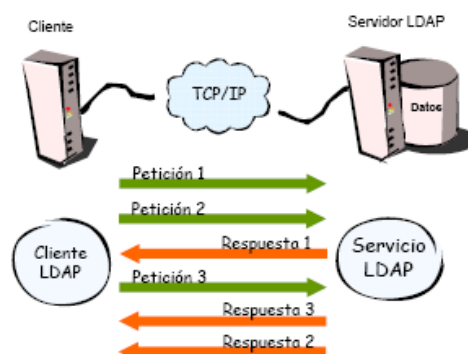
El protocolo LDAP fue creado como una versión liviana de X.500 y terminó por reemplazarlo. Por esta razón algunos de los conceptos y estándares que utiliza LDAP provienen de la serie de protocolos X.500.



*Comparación entre el modelo arquitectónico X.500 y LDAP*

**LDAP:** protocolo ligero para acceder al servicio de directorio, especialmente al basado en X.500, con el fin de agilizar y aumentar la capacidad de búsqueda. LDAP se ejecuta sobre TCP/IP o sobre otros servicios de transferencia orientado a conexión. Almacena **información** acerca de **objetos relacionados** (recursos de red, personas, departamentos). Si el servicio de nombres es como las páginas blancas, el servicio de directorios viene a ser como las páginas amarillas. Abarca un ámbito más general y por tanto aumenta las capacidades de búsqueda puesto que podemos buscar por cualquiera de los atributos. LDAP se puede llamar al protocolo, al servidor, al árbol (estructura) y a una herramienta administrativa y de gestión.

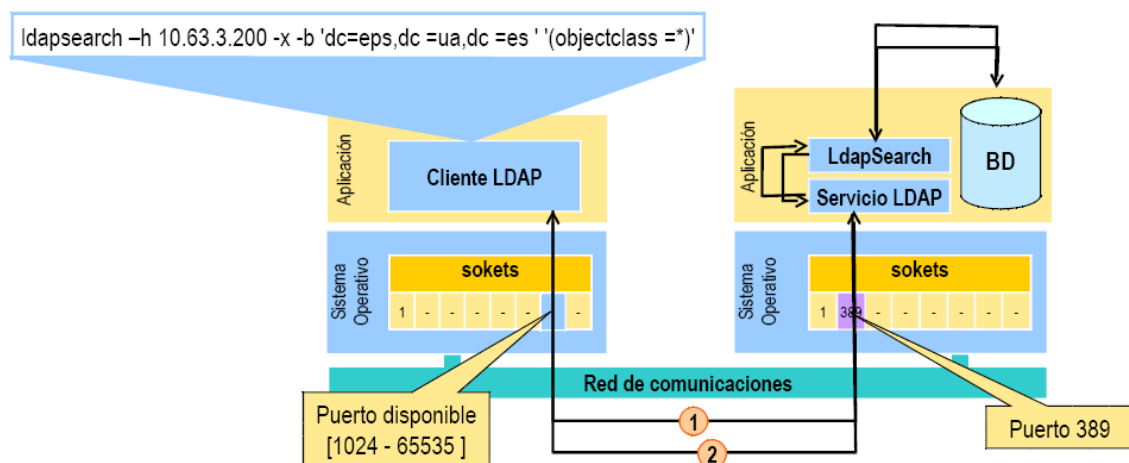
usuario que lo



El modelo de información de LDAP está **basado en entradas**. Una entrada es una **colección de atributos** que tienen un único y global Nombre Distinguido (DN). El DN se utiliza para referirse a una entrada sin ambigüedades. Cada atributo de una entrada posee un tipo y uno o más valores. Los tipos son normalmente palabras nemotécnicas, como "cn" para common name, o "mail" para una dirección de correo. Podemos almacenar recursos de red (servidores, impresoras,...), nombres de personas, departamentos, etc.

### Funcionamiento LDAP:

- 1 Petición de búsqueda de un cliente
- 2 El servidor devuelve la el mensaje de respuesta



**Modelos LDAP:** existen cuatro modelos posibles en LDAP, cada uno de los cuales representa los servicios LDAP vistos por el cliente:

- **Modelo de información:** estructura y tipos de datos (esquemas, entradas, atributos). Una **entrada** es un nodo del árbol con información sobre el padre y pares **atributo:valor**, refleja conceptos del mundo real (usuarios, organizaciones). Los **atributos** pueden ser **atómicos** (firstname, surname, telef) ó **conjuntos** (ObjectClass). Cada entrada incluye un atributo object class, el cual determina la clase (o clases) del objeto al que se refiere dicha entrada. La definición de una clase debe indicar qué atributos son obligatorios y cuales son opcionales para las entradas en dicha clase. Se pueden definir nuevas clases si es necesario. Las definiciones de clases se organizan en una jerarquía de herencias en las que todas las clases excepto la topclass deben contener un atributo objectClass y el valor de dicho atributo debe ser el nombre de una o más clases. Posee un identificador único (OID) que es un número asignado por la Autoridad de Internet Asignadora de Números (IANA). Esta información se define en ficheros LDIF

objectClass::person
cn:
sn:
-----
userPassword:
telephoneNumber:
seeAlso:
description:



#Ejemplo de archivo LDIF para la entrada de un usuario

```
dn: cn=Alex García Pérez, ou=profesores, dc=dtic, dc=ua, dc=es
objectclass: top
objectclass: person
cn: Alex García Pérez
sn: García Pérez
telephoneNumber: 96 590 3400
telephoneNumber: 96 590 3405
```

- **Modelo de asignación de nombres:** define cómo referenciar las entradas y los datos en el árbol de directorios de manera única en el árbol de directorios. RDN (nombre relativo al nodo padre "alex") y DN (nombre completo único alex.dccia.ua.es).

RDN → cn=Alex Garcia

DN → cn=Alex Garcia, ou=profesores, dc=dtic, dc=ua, dc=es

- **Modelo funcional:** describe qué operaciones se pueden realizar en la información almacenada en un directorio de LDAP (actualizaciones, solicitudes, autentificación), que es ni más ni menos el protocolo a emplear.

```
ldapadd -x -D "cn=Manager , dc=eps , dc=ua , dc=es " -W -f example.ldif
```

```
ldapsearch -b "dc=dtic,dc=ua,dc=es" "objectclass=person" sn
```

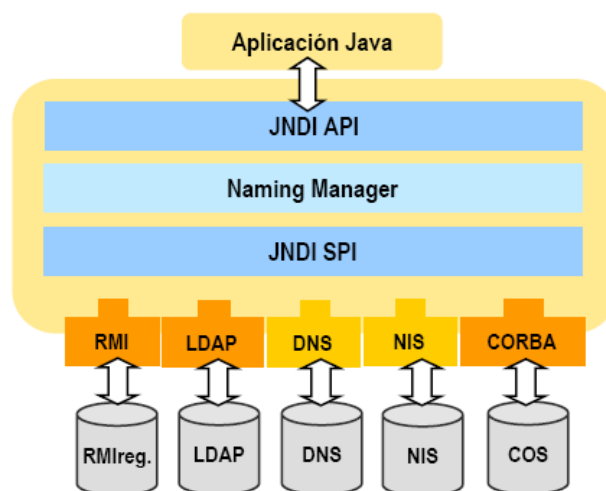
- **Modelo de seguridad:** para el cliente, cómo probar su identidad (autenticación) y para el servidor, cómo controlar el acceso (autorización). Es un protocolo orientado a conexión y de intercambio de mensajes. Todas las operaciones se controlan mediante el nivel de autorización del usuario autenticado mediante un login y una password encriptada en MD5 ó SHA. También se puede negociar una capa de transporte segura (SSL). Tipos de autenticación:
  - Anónima: dn vacío + password, acceso a información pública.
  - Simple: dn + password, la información viaja sin codificar (no recomendada).
  - Simple sobre Capa de conexión segura y Seguridad de capa de transporte (SSL/TLS): se negocia una capa de transporte seguro. La información viaja de forma segura.
  - Simple y Capa de Seguridad (SASL): módulo configurable que permite negociar previamente el mecanismo de autenticación y la capa de transporte seguro (kerberos, GSSAPI, S/Key, EXTERNAL).

**Características distribuidas:** puede usar bases de datos como Back-Storages (copias de seguridad). Puede dividir el árbol de directorios en subárboles gestionados por diferentes servidores LDAP con el fin de aumentar el rendimiento, descentralizar su localización geográfica o por cuestiones administrativas. Cada subárbol o rama será referenciada desde el árbol padre (aquel marcado como objectClass::referral). Cuenta con dos modos de funcionamiento: o bien es el servidor LDAP el que resuelve las solicitudes o es el cliente el que las resuelve.

**Escenarios LDAP:** LDAP tiene una gran presencia en el mundo empresarial, gracias a la cantidad de recursos que ofrece para su uso empresarial. Principalmente se usan las siguientes características:

- Directorios de información
- Sistemas de Autenticación/Autorización
- Sistemas de correo electrónico
- Sistemas de alojamiento páginas Web y FTP
- Grandes sistemas de autenticación basados en RADIUS
- Servidores de certificados públicos y llaves de seguridad
- Perfiles de usuarios centralizados

**JNDI:** la Interfaz de Nombrado y Directorio Java (JNDI) es una Interfaz de Programación de Aplicaciones para servicios de directorio. Esto permite a los clientes descubrir y buscar objetos y nombres a través de un nombre y, como todas las APIs de Java que hacen de interfaz con sistemas host, es **independiente de la implementación** subyacente (funciona como plug-in, conexión dinámica de diferentes implementaciones). Adicionalmente, especifica una interfaz de proveedor de servicio (SPI) que permite que las implementaciones del servicio de directorio sean enchufadas en el framework. Las implementaciones pueden hacer uso de un servidor, un fichero, o una base de datos; la elección depende del vendedor.



Arquitectura JNDI

La API JNDI se usa por Invocación a Método Remoto de Java (RMI) y a las APIs J2EE para buscar objetos en una red. JINI tiene su propio servicio de búsqueda y no usa la API de JNDI.

La API suministra:

1. Un mecanismo para asociar(bind) un objeto a un nombre
2. Una interfaz de búsqueda de directorio que permite consultas generales (servicio de nombres y servicio de directorios)
3. Una interfaz de eventos que permite a los clientes determinar cuando las entradas de directorio han sido modificadas
4. Extensiones LDAP para soportar las capacidades adicionales de un servicio LDAP

Mediante JNDI se definen varios estándares para realizar búsquedas en diversos sistemas de información como LDAP Servers, NDS, COS. Los diversos vendedores de directorios distribuidos y "LDAP Servers" deben definir un SPI ("Service Provider Interface") para su producto. Este SPI ofrecerá las funcionalidades del producto vía un ambiente en Java.

JNDI organiza sus nombres en una jerarquía. Un nombre puede ser cualquier string tal como "com.mydomain.ejb.MyBean". Un nombre también puede ser un objeto que soporte la interfaz Name, sin embargo la forma más común de nombrar a un objeto es un string. Un nombre se asocia a un objeto en el directorio almacenando el objeto o una referencia JNDI al objeto en el servicio de directorio identificado por el nombre.

La API JNDI define un contexto que especifica donde buscar un objeto. El contexto inicial se usa normalmente como punto de partida.

En el caso más simple, un contexto inicial debe crearse usando la implementación específica y los parámetros extra requeridos por la implementación. El contexto inicial será usado para buscar un nombre. El contexto inicial es análogo a la raíz de un árbol de directorios para un sistema de ficheros.

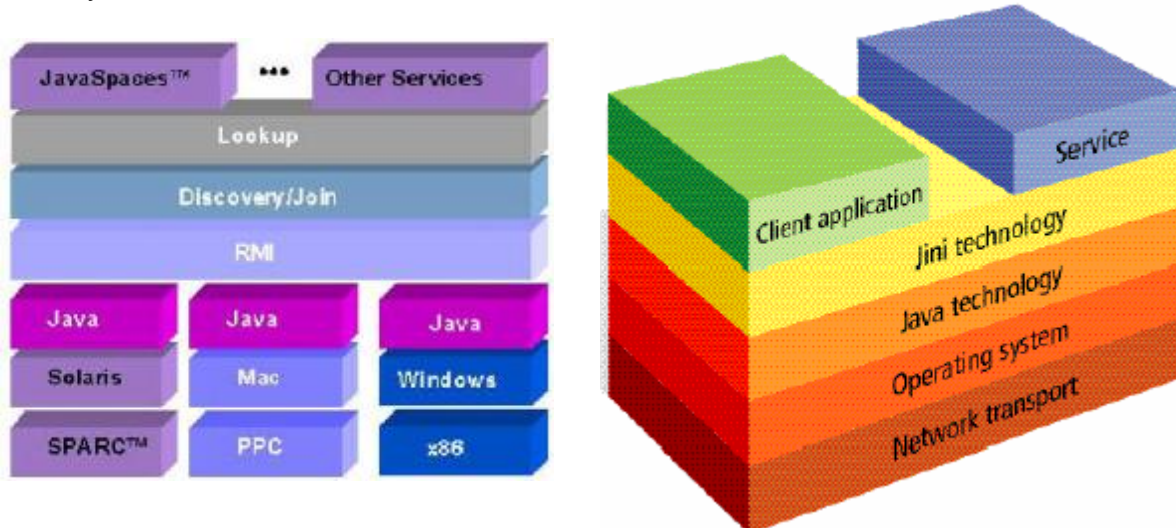
### **3. Servicios de descubrimiento**

Debido al incremento del número de dispositivos interconectados se ha creado la necesidad de conocer: que dispositivos están conectados a la red, que pueden hacer, cómo pueden colaborar y con quien. Por tanto hay que crear un mecanismo que facilite la interoperatividad y la coordinación de dispositivos y servicios (sin intervención del usuario, sin administración) y esto es un servicio de descubrimiento. Da servicios como: servicios de búsqueda, anuncio y publicación, visualizar servicios, servicios de descubrimiento, clasificación de servicios, catálogo de servicios disponibles, eventos, recolector.

Las propuestas son JINI, UPnP, Servicios web y JXTA.

**JINI:** es un sistema integrado con J2EE que posibilita las redes espontáneas de servicios software integrándolos en grupos de trabajo. Da soporte a servicios en red, tanto hardware como software,

Es multiplataforma, usa serialización, trabaja sobre RMI, e implementa un modelo de seguridad. Se basa en la creación de interfaces accesibles para todos y de todos los servicios.



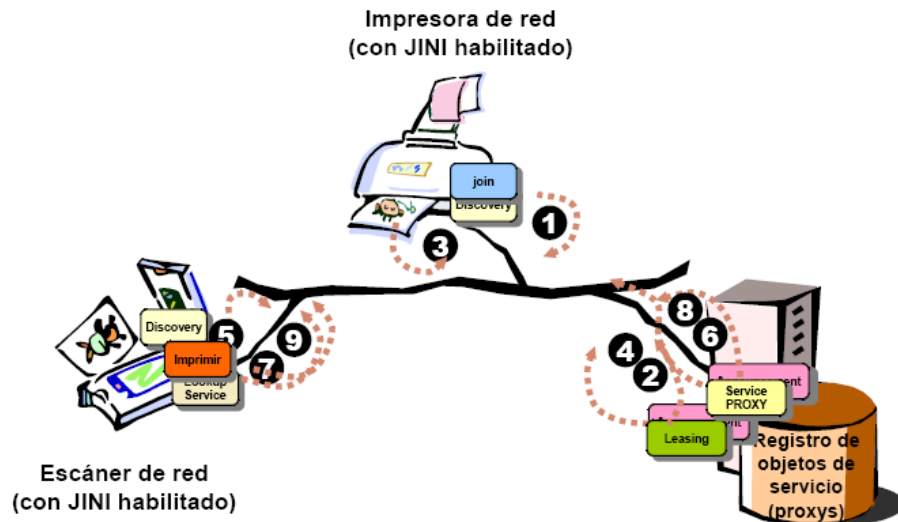
### Los protocolos y su funcionamiento:

- **Lookup:** permite mantener un registro de servicios, localizarlos y proporcionar un proxy para su utilización.
- **Discovery:** permite localizar los servicios de lookup.
- **Join:** permite registrar un servicio en un servidor de lookup.
- **Leasing:** proporciona permisos a un cliente durante un tiempo que deberá renovar.
- **Eventos:** permite conocer cambios en el entorno (ej: desconexión de un dispositivo).

Un proveedor de servicios (impresora) busca un servicio de lookup (discovery) haciendo un broadcasting. Si lo hay, el lookup responderá. Entonces el proveedor de servicios registra un objeto de servicio (Proxy) y sus atributos en el lookup (join). El lookup le asigna unos permisos (temporales) mediante leasing.

Un cliente solicita un servicio (escáner busca servicios de impresión). Se le proporciona una copia del objeto de servicio (Proxy)(de la impresora) y éste la utiliza para “hablar” con el servicio y utilizarlo (lookup).

- |  |   |
|--|---|
| 1 Broadcasting para localizar servidores lookup                            | 5 Broadcasting para localizar servidores lookup |
| 2 Respuesta (proxy del servidor lookup)                                    | 6 Respuesta (proxy del servidor lookup)         |
| 3 Registro del servicio (proxy y atributos)                                | 7 Busca servicios (de impresión)                |
| 4 Permisos (temporales)  | 8 Proporciona información del servicio (proxy)  |
| 9 Usa el servicio a partir del proxy del objeto (tomado del Servidor HTTP) |   |

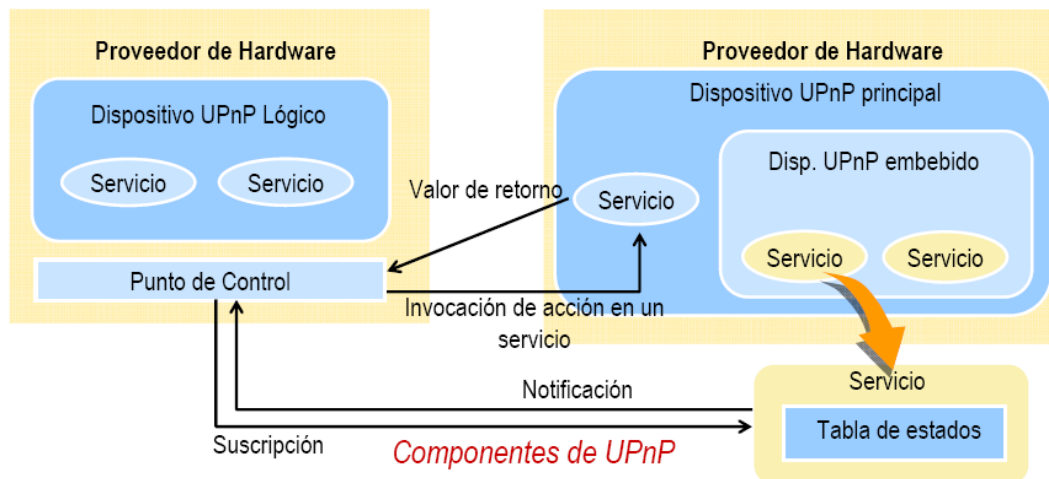


Inconvenientes: sólo programación Java, requiere interfaces estáticas, requiere servicios adicionales de lookup (persistencia, transacciones) y el servidor lookup puede ser un cuello de botella y debemos tener más de uno si queremos que sea tolerante a fallos.

UPnP: es la solución de Microsoft (en desarrollo) basada en protocolos TCP/IP. La arquitectura UpnP permite a un dispositivo: autoconfigurarse, anunciarse en la red, describir sus habilidades y descubrir e interactuar con otros dispositivos.

#### Componentes:

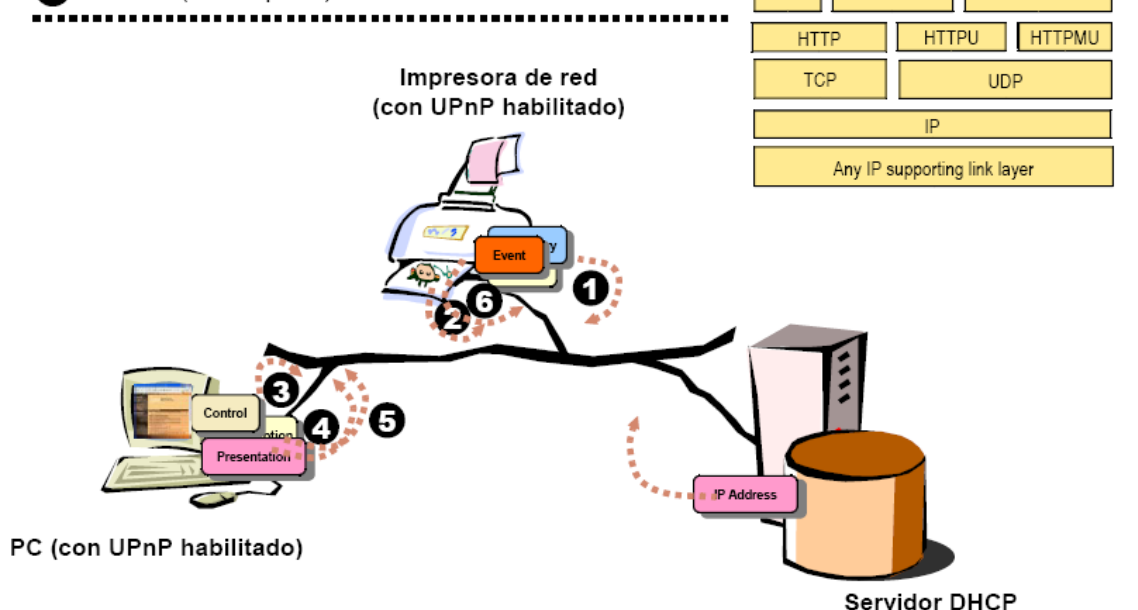
- **Dispositivos:** contenedor de servicios y dispositivos. UPnP, agrupa tipos de dispositivos. XML.
- **Servicios:** determinados por el tipo de dispositivo. Unidad de funcionalidad, provee acciones, métodos y estado. XML.
- **Puntos de control:** control y descubrimiento de dispositivos. C/S, el punto de control = cliente. Suscribirse a notificaciones de cambios de estado



### El funcionamiento es el siguiente:

Un dispositivo utiliza direccionamiento para obtener una IP mediante un servidor DHCP. Utiliza el descubrimiento para anunciarse y enviar su descripción al resto de dispositivos de red. Los demás dispositivos se presentan y envían también su descripción. Un dispositivo usa el control de otro para utilizarlo.

- ❶ Direccionamiento
- ❷ Descubrimiento – (anuncio, búsqueda)
- ❸ Descripción
- ❹ Control
- ❺ Presentación
- ❻ Evento – (suscripción)



**JINI vs UPnP:** en comparación con JINI que actúa por encima del SO a modo middleware, UPnP está definido en el núcleo del SO.

UPnP puede trabajar con puntos de control o sin ellos (no tiene porque haber un servidor cada disp. es autónomo), en cambio JINI necesita siempre servidores Lookup. UPnP está basado en TCP/IP, HTTP y XML, JINI en cambio está basado en proxys de objetos.

UPnP no define el modo de acceso a métodos, JINI usa RMI.

Ambos permiten proporcionar información de registro desde servidores externos Web. Ambos trabajan y soportan eventos para actualización dinámica. Ambos cubren el mismo tipo de problemas pero con enfoques estructurales diferentes.

En conclusión se avanza hacia la búsqueda de soluciones que permitan a los servicios actuar e interactuar de forma totalmente autónoma.



## TEMA 4: TIEMPOS Y ESTADOS GLOBALES

### 1. Relojes, eventos y estados

La monitorización y temporización de ejecuciones surge de la necesidad de conocer cuando ocurrió un **evento**. La noción del tiempo es importante en un sistema distribuido puesto que **no existe un reloj global al sistema**. No existe un reloj universal de referencia, ya que el tiempo es relativo.

Se han desarrollado algoritmos que dependen de la sincronización del reloj para varios problemas en distribución. Estos incluyen el mantenimiento de la consistencia de los datos distribuidos (el uso de **marcas de tiempo** para serializar transacciones), la comprobación de la autenticidad de una solicitud enviada a un servidor.

Cada computador de la red tiene su propio reloj interno y dado que no son perfectos es muy **difícil sincronizarlos** puesto que aunque todos se sincronizasen variarían con el tiempo, ya que procesos en computadoras distintos pueden tener marcas de tiempo distintas.

**La tasa de deriva del reloj:** es la diferencia por unidad de tiempo en que el reloj del computador difiere del reloj perfecto. (Ejemplo: +- 5 seg/mes). ¿Qué puede influir en la tasa de deriva del reloj? El estado de las baterías, la temperatura.

¿Cuáles son las necesidades del make de Unix con respecto al reloj en un sistema NFS? Podemos estar enlazando con un fichero con fecha posterior al make. También es posible que recompile porque la fecha es posterior. Valores de reloj monótonos crecientes. Relojes de servidor y clientes sincronizados.

Un sistema distribuido está definido como una **colección de procesos**  $P$  de  $N$  procesos. Cada proceso  $P_i$   $i = 1..N$ , tiene un estado  $S_i$  formado por todas sus variables u objetos y que puede cambiar en ejecución. Los procesos se comunican mediante mensajes (enviar, recibir, cambiar estado).

Un **evento** es una ocurrencia de una acción que lleva a cabo un proceso (enviar, recibir, cambiar estado). Los eventos en el proceso  $P_i$  pueden ordenarse de forma total por el orden en que suceden. De forma que la historia del proceso  $P_i$  es una serie de sus eventos  $historia(P_i) = \langle e_{i0}, e_{i1}, \dots \rangle$ .

**Relojes:** para establecer las **marcas temporales** se usa el reloj del computador. En un instante  $t$  el S.O lee el valor del reloj hardware  $H_i(t)$  y calcula el tiempo en el reloj software  $C_i(t) = \alpha H_i(t) + \beta$  que a pesar de no ser exacto puede ser usado como marcador de los eventos de  $P_i$ . La resolución del reloj debe ser menor que el intervalo entre dos posibles eventos.

La resolución del reloj tiene que ser menor que el intervalo de tiempo entre dos posibles eventos consecutivos.

Los relojes de cada proceso no siempre están en perfecto acuerdo y a su diferencia entre ellos se le llama **sesgo**.

**El Tiempo Universal Coordinado:** los relojes de un computador pueden sincronizarse con fuentes externas de tiempo de gran precisión. Los relojes físicos utilizan osciladores atómicos, cuyo ritmo de deriva es aproximadamente en  $10^{13}$ . La salida de estos relojes atómicos se utiliza como estándar para el tiempo real transcurrido, conocido como el Tiempo Atómico Internacional.

El UTC es un estándar internacional de cronometraje, está basado en el tiempo atómico. Este estándar es transmitido vía satélite y repetidores terrestres. Si pusiésemos un GPS en todos los computadores podríamos recibir una señal de sincronización precisa pero es muy costoso económicamente y las paredes son una barrera para el GPS.

## 2. Sincronización de relojes físicos

**La sincronización externa:** para conocer en qué hora del día ocurren los sucesos en los procesos de nuestro SD, es necesario sincronizar los relojes de los procesos  $C_i$  con una **fuentes externa autorizada S (UTC)**.

$S(t) - C(t) < D$ , para  $i=1,2,.. N$  en un intervalo  $I$  de tiempo real. Los relojes  $C_i$  son precisos con el límite  $D$ .

**La sincronización interna:** si la sincronización de los relojes  $C_i$  están sincronizados con otro con **un grado de precisión conocido**, entonces podemos medir el intervalo entre dos eventos que ocurren en diferentes computadores llamando a sus relojes locales, incluso aunque ellos no estén necesariamente sincronizados con una fuente externa de tiempo, puesto que pueden derivar juntos.

$C_i(t) - C_j(t) < D$ , para  $i,j=1,2,..N$   $i \neq j$  en un intervalo  $I$  de tiempo. Los relojes  $C_i$  y  $C_j$  son precisos con el límite  $D$ .

La sincronización externa es mejor, pero la interna no es tan mala como para dejarla fuera de consideración teniendo en cuenta su facilidad y bajo coste.

**La sincronización de relojes físicos:** es la solución. Se dice que un reloj HW ( $H$ ) es correcto si su límite de deriva es conocido  $p > 0$  y por tanto el error en la medida de dos eventos  $t$  y  $t'$  está limitado.

$$(1 - p)(t' - t) \leq H(t') - H(t) \leq (1 + p)(t' - t) \text{ donde } t' > t$$

Esto impide que se produzcan saltos traumáticos en el valor leído.

Se puede relajar la condición mediante la monotonicidad (lo mismo pero con un reloj SW).

$$t' > t \rightarrow C(t') > C(t)$$

Si un reloj no cumple las condiciones de corrección es defectuoso. Un fallo de ruptura del reloj se produce cuando se para (no emite tics) y cualquier otro fallo es llamado arbitrario. Un reloj no tiene por qué ser preciso para ser correcto.

**Sincronización en un SD síncrono:** un SD es síncrono si están definidos los límites siguientes:

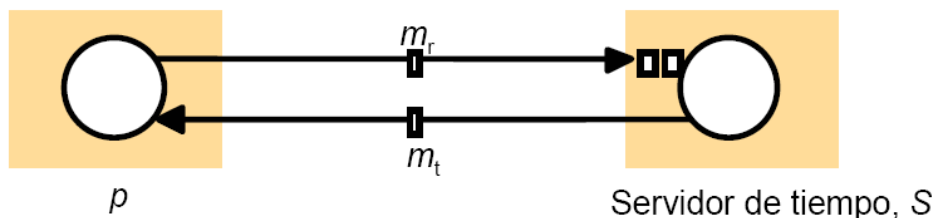
- $T_{max}$  y  $T_{min}$  para ejecutar cada paso de un proceso.
- $T_{max}$  y  $T_{min}$  de recepción de un mensaje.
- Límites de deriva de cada reloj local.

¿Es internet un SD síncrono? No, es asíncrono puesto que no existe una cota superior de tiempo de transmisión de un mensaje.

Primera aproximación (Sincronización interna):

- un proceso **p1** envía su tiempo local **t** al proceso
- **p2** en un mensaje **m**. **p2** podría poner su reloj a **t+Ttrans**, donde **Ttrans** es el tiempo de emisión de **m**.
- **Ttrans** desconocido pero **Tmin ≤ Ttrans ≤ Tmax** es conocido.
- La incertidumbre **u = Tmax - Tmin**. Si se establece el reloj a **t + (Tmax - Tmin)/2** entonces el sesgo  $\leq u/2$

**Método de Cristian:** sugirió la utilización de un servidor de tiempo, conectado a un dispositivo que recibe señales de una fuente de UTC, para **sincronizar computadores externamente**. Bajo solicitud, el proceso servidor **S** proporciona el tiempo de acuerdo con su reloj tal como se muestra:



El proceso **p** solicita el tiempo en un mensaje **mr** y recibe **t** en **mt** de **S**. **p** establece su tiempo a  $t + T_{round}/2$  ( $T_{round}$  es el tiempo de ida y vuelta).

La precisión es de  $\pm(T_{round}/(2 \cdot \min))$  ( $\min$  es el mínimo estimado de transmisión). El momento más temprano en que **S** pone a **t** en **mt** es  $\min$  después de que **p** enviara **mr**. El momento más tardío es  $\min$  antes de que **mt** llegue a **p**. El tiempo de **S** cuando **mt** llega a **p** está en el rango  $[t + \min - T_{round} - \min]$

Problemas: es un método probabilístico y se basa en un único servidor (no tolerancia a fallos).

Solución: usar varios servidores y el primer mensaje (**mt**) que llegue es el válido.

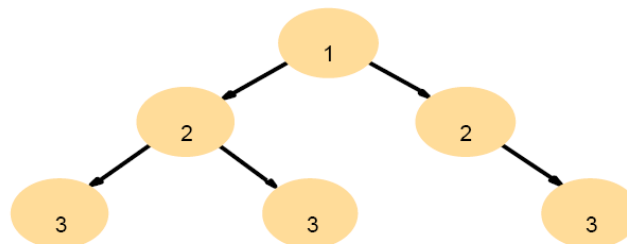
**Algoritmo de Berkeley:** un algoritmo para la **sincronización interna**, se elije un computador coordinador para actuar como maestro y recoge valores de reloj del resto de computadores esclavos.

El maestro utiliza los tiempos de ida y vuelta de los mensajes para estimar el valor de los relojes locales esclavos y promedia los resultados incluyéndose la propia lectura del reloj y eliminando cualquier valor inconsistente y envía la magnitud de ajuste (+-) de cada reloj a cada computador.

Ventaja: Si el maestro falla se puede elegir otro maestro (tolerancia a fallos).

**Protocolo de tiempo en red (NTP):** los dos métodos anteriores están pensados para utilizar en intranets, mientras que NTP define una arquitectura para el tiempo **sobre Internet**. Proporciona un servicio que permita a los clientes a lo largo de Internet estar sincronizados de forma precisa a UTC, a pesar de los retardos largos y variables. Proporciona un servicio fiable que pueda sobrevivir a pérdidas largas de conectividad, hay servidores redundantes y recorridos redundantes entre los servidores, escalable y seguro (con autenticación de las fuentes de tiempo). Permite a los clientes resincronizar con suficiente frecuencia para compensar las tasas de deriva encontradas en la mayoría de los computadores, el servicio está diseñado para escalar a gran número de clientes y servidores. Proporciona protección contra la interferencia con el servicio de tiempo, ya sea maliciosa o accidentas.

El servicio NTP está proporcionado por una red de servidores localizados a través de Internet. Los servidores primarios están conectados con fuentes UTC, los secundarios se conectan a los primeros y forman la subred de sincronización. En el nivel más bajo de servidores (3) están los PCs. (nivel=estrato).

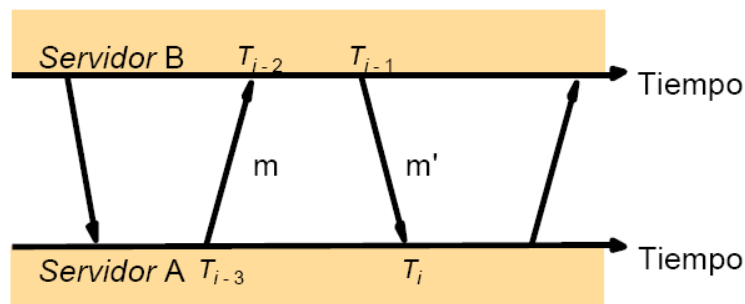


La subred de sincronización se puede reconfigurar si se produce un fallo (Un primario que pierde su conexión con UTC puede pasar a secundario, un secundario que pierde a su primario puede elegir otro primario).

**Modos de sincronización:**

- **Multidifusión (multicast):** un servidor reparte el tiempo al resto y estos establecen su tiempo asumiendo un retraso. Poco preciso. Esta pensado para el uso en una LAN.
- **Llamada a procedimiento:** similar a Cristian. El servidor acepta solicitudes de otros computadores, el responde con su marca de tiempo (lectura actual del reloj). Precisión más alta.
- **Simétrica:** pensado para LANs, pares de servidores se intercambian mensajes conteniendo información de tiempo. Es la más precisa (se puede implementar en los primeros niveles (estratos más bajos)).

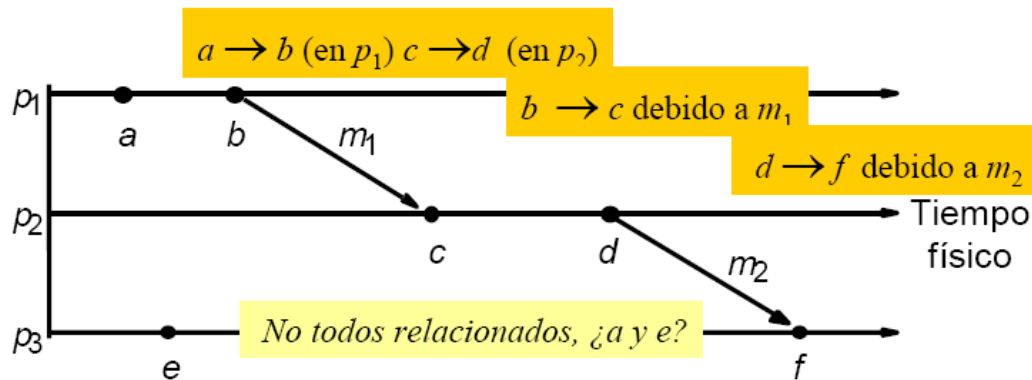
**Intercambio de mensajes:** se realiza utilizando UDP. En el modo de llamada a procedimiento y simétrica, los procesos intercambian pares de mensajes. Cada mensaje lleva marcas de tiempo. Se mandan los tiempos de los eventos recientes (envío, recepción). Puede haber retrasos y se pueden perder mensajes.



**Precisión:** los servidores NTP mantienen pares  $\langle o(\text{derivai}), d(\text{precisión}) \rangle$  estimando la fiabilidad de las variaciones. La precisión suele ser de mseg en LAN y de decenas de mseg en Internet.

### 3. Tiempo lógico y relojes lógicos

**Lamport:** no se sincronizan los relojes, sino que se ordenan los eventos según la relación de orden parcial “suceder antes”. Es decir, si los eventos ocurren en  $P_i$  1,2,3...N entonces ocurren en ese orden. Cuando se envía un mensaje entre dos procesos, el suceso de enviar el mensaje ocurrió antes del de recepción del mismo.

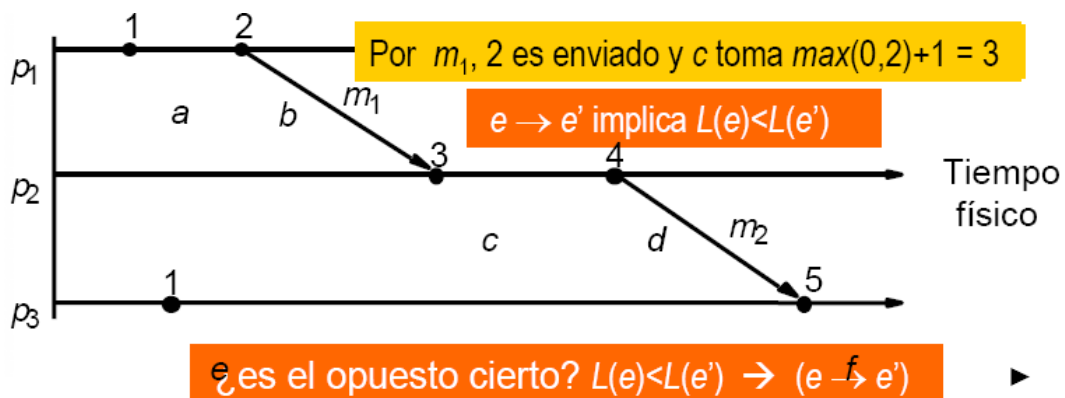


Pero ¿Qué ocurre cuando tenemos muchos procesos y no todos los eventos relacionan todos los procesos? Esto es porque ocurren en procesos diferentes y no hay una cadena de mensajes que intervengan entre ellos. Los sucesos  $a$  y  $e$  son concurrentes.

Señalar que aun produciéndose la relación "suceder antes" entre dos sucesos, el primero podría o no haber causado realmente el segundo. La relación debe ordenar estos sucesos.

**Un reloj lógico** es un contador software que se incrementa monótonamente, cuyos valores no necesitan tener ninguna relación particular con ningún reloj físico. Cada proceso  $P_i$  tiene su reloj lógico ( $L_i$ ) que utiliza para **fijar las marcas** temporales a los eventos según:

- R1:  $L_i$  se incrementa en 1 antes de cada evento propio de  $p_i$
- R2.a cuando  $p_i$  envía  $m$ , adjunta al mensaje el valor  $t = L_i$
- R2.b cuando  $p_j$  recibe  $(m, t)$  establece  $L_j := \max(L_j, t)$  y aplica R1 antes de establecer la marca de tiempo de  $\text{recibe}(m)$



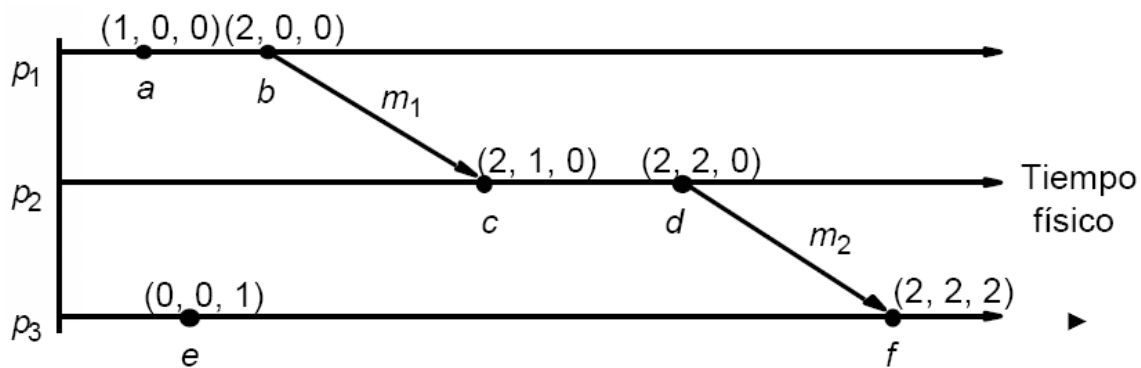
**Los relojes vectoriales** (Mattem y Fidge) fueron desarrollados para superar la diferencia de los relojes lógicos de Lamport del hecho que  $(L(e) < L(e'))$  no implica  $(e \rightarrow e')$ .

Un reloj vectorial  $V_i$  en el proceso  $p_i$  es un array de  $N$  enteros, que cada proceso utiliza para establecer marcas de sus eventos locales,

- R1: inicialmente  $Vi[j] = 0$  for  $i, j = 1, 2, \dots, N$
- R2: antes de marcar un nuevo evento  $pi$  incrementa  $Vi[i] := Vi[i] + 1$
- R3:  $pi$  adjunta  $t = Vi$  en cada mensaje que envía
- R4: cuando  $pi$  recibe  $(m, t)$  establece  $Vi[j] := \max(Vi[j], t[j])$   $j = 1, 2, \dots, N$

$Vi[i]$  es el número de eventos que  $pi$  ha marcado

$Vi[j]$  ( $j \neq i$ ) número de eventos en  $pj$  que han sido afectados por  $pi$

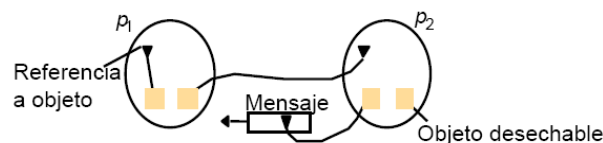


Mejoran a los de Lamport ya que determinan si dos eventos están ordenados por la relación “suceder antes” o son concurrentes. Un reloj vectorial  $Vi$  en el proceso  $Pi$  es un array de  $N$  enteros, que cada proceso utiliza para establecer marcas de sus eventos locales.

## 4. Estados globales

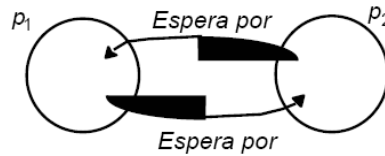
Se examina si una propiedad particular de un SD es cierta cuando éste se ejecuta (Ej.: Compactación automática de memoria, Detección distribuida de bloqueos indefinidos, Detección de terminación distribuida, Depuración distribuida). Ilustran la necesidad de observar el estado del SD globalmente.

**Compactación de memoria:** antes de eliminar información se deben analizar las referencias existentes y los canales de comunicación (ej.: mensajes).

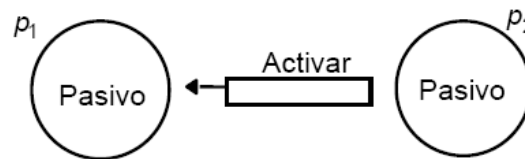


**Bloqueo indefinido:** el “abrazo mortal” clásico, pero entre procesos no ubicados en la misma máquina. Ocurre cuando cada uno de los procesos espera por otro proceso enviarle un mensaje y este espera un mensaje del otro.





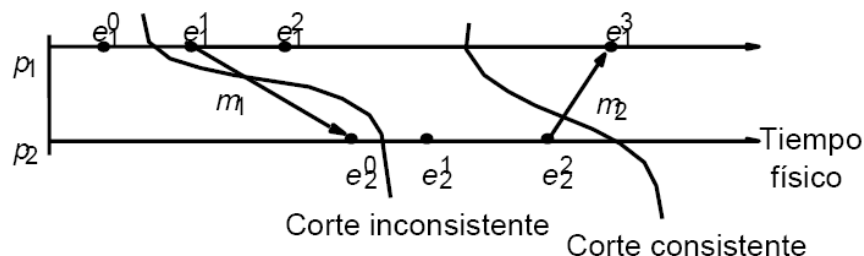
**Terminación:** el problema aquí es detectar que un algoritmo distribuido ha terminado. Dos procesos pueden en un instante ser pasivos, pero no ser susceptibles de ser eliminados. (P1 puede estar pasivo pero ser activado por un mensaje).



Así pues entendemos que la gestión de estados globales es importante.

La historia de un proceso es el conjunto ordenado de sus eventos. En un momento determinado podemos querer extraer el prefijo finito de eventos del  $n$  al  $m$ . La historia global de un sistema es la unión de todas las historias de sus procesos.

**El corte** es un subconjunto de la historia global que es la unión de los prefijos de las historias de los procesos.



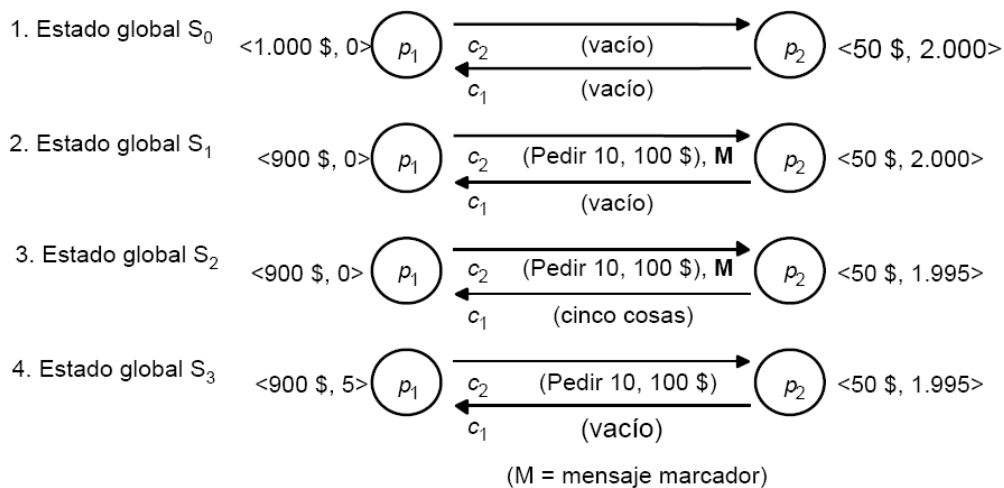
Un corte es consistente si para cada suceso que contiene, también contiene todos los sucesos que sucedieron antes del suceso.

Un estado global consistente es aquel que se corresponde con un corte consistente.

La ejecución se caracteriza como transiciones entre estados globales ( $S1 \rightarrow S2 \rightarrow \dots$ ).

Linealización o ejecución consistente: ordenación de los sucesos en una historia global consistente.





El

estado registrado puede diferir de todos los estados globales que realmente han pasado por el sistema, el algoritmo sólo selecciona un corte de historia de la ejecución cuyo estado registrado es consistente. El algoritmo de instantánea termina, si se cumplen las restricciones de conectividad total e inexistencia de fallo en la comunicación.

Conclusiones: los algoritmos como Cristian o NTP sincronizan los relojes a pesar de sus derivas y el retardo de los mensajes.

La sincronización de relojes no siempre es suficiente para satisfacer los requisitos de ordenación de dos eventos arbitrarios que sucedan en dos computadores.

## TEMA 5: SEGURIDAD

### 1. Introducción

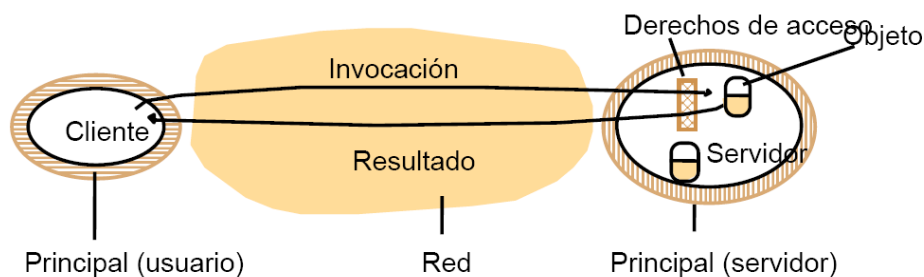
Lo que vamos a ver de seguridad estará orientado a los sistemas distribuidos, por lo que algunos aspectos de seguridad no se verán.

Existen amenazas que no se basan en la lectura de datos que pasan por la red, como por ejemplo para hacer daño, jugar, etc.

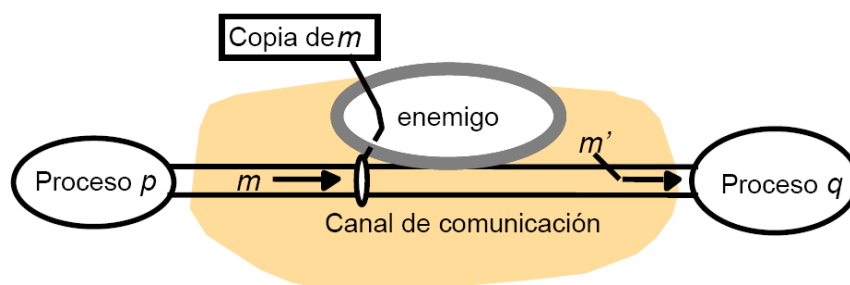
La auditoria de perfiles se está convirtiendo en un complemento ideal para los sistemas de seguridad.

Los sistemas de seguridad americanos, si se ciñen a la ley, son más inseguros que los que se pueden desarrollar en Europa.

**Planteamiento básico:** en un primer momento sólo se pensaba en la seguridad cuando existía un cable y se podía “enganchar”, etc. Ahora, con la aparición de la wireless, cualquiera puede obtener los datos con cierta facilidad. Un **objeto** (o recurso) puede ser parte de una Web comercial o un sistema de archivo. Un **principal** es un usuario o proceso que tiene derechos para realizar las acciones. La identidad del principal es importante.



**Enemigo:** ataques en aplicaciones que manejan transacciones comerciales u otra información cuyo secreto o integridad es crucial. Existen amenazas a los canales de comunicación a los procesos, denegación de servicio.



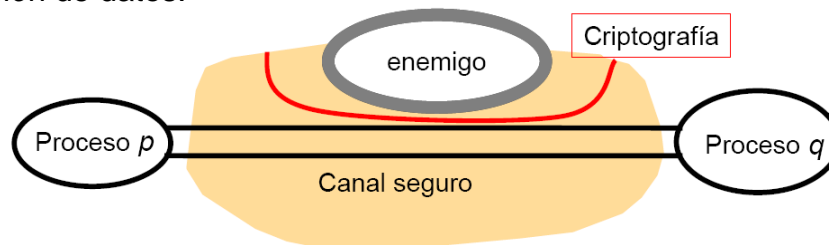
En un sistema transaccional se debe asegurar que la información llegue, que no la describen, y que ésta sea válida, es decir, que nadie la haya modificado por el camino.

Debemos asegurar que los datos que enviamos son únicos, para evitar las copias (evitar que te facturen dos veces el mismo producto, etc.).

**Canal seguro:** el secreto se reserva mediante la ocultación criptográfica, está basada:

- **La confusión:** se basa en cambiar unos símbolos por otros. Hoy en día se intenta que un mismo símbolo del alfabeto de entrada tenga varios símbolos del alfabeto de salida. La confusión más básica es muy vulnerable. Con técnicas estadísticas y sabiendo el idioma del mensaje original es posible descifrarlo.
- **La difusión:** trata de subsanar el problema de no aleatoriedad. De esta manera reordenamos el mensaje para que parezca una sucesión de símbolos complementemente aleatoria.
- **La posesión:** posesión de secretos en claves. Los algoritmos de clave compartida son muy rápidos. Los de clave pública/privada son más seguros pero mucho más lentos.

Cada proceso tiene que estar seguro de la identidad del otro. Los datos son privados y protegidos contra la manipulación, protección contra repeticiones y reordenación de datos.



#### **Amenazas y formas de ataque:**

- **Escuchar a escondidas:** y obtener información privada (secretos)
- **Enmascararse:** (suplantar) asumiendo la identidad de otro usuario/principal.
- **Alterar mensajes:** alterando el contenido de los mensajes en tránsito
- **Reenviar mensajes:** seguros que hemos almacenado y reenviamos más tarde
- **Negación del servicio (DoS):** inundando, sobrecargando un canal o recurso e impidiendo el acceso para los otros (ejemplo: el ataque a Amazon y Yahoo del 2000). Desde un servidor malicioso (o varios) se hace un ping a la víctima utilizando diferentes IPs falsas provocando así muchos pong que posiblemente tendrán respuesta de error.
- **Troyanos y otros virus:** Los virus sólo pueden entrar cuando se importa su código de programa y esto suele hacerse mediante la instalación de nuevo software o ejecución de código dinámico (aplets,...).

**Nomenclatura:**

Alice	Primer participante
Bob	Segundo participante
Carol	Otro participante en los protocolos a tres o cuatro bandas
Dave	Participante en los protocolos a cuatro bandas
Eve	Fisgón
Mallory	Atacante malevolente
Sara	Un servidor
Ka	Clave secreta de Alice
Kb	Clave secreta de Bob
KApriv	Clave privada de Alice (sólo conocida por Alice)
KApub	Clave pública de Alice (publicada por Alice para la lectura de cualquiera)
{M}k	Mensaje M encriptado con la clave K
[M]k	Mensaje M firmado con la clave K

**2. Técnicas de seguridad**

**Secreto con clave compartida:** dos interlocutores comparten una clave secreta. Ambos interlocutores tendrán una comunicación segura mientras la clave no sea conocida.

Alice y Bob comparten una clave secreta KAB

1. Alice usa KAB y acuerda una función de encriptación  $E(KAB, M)$  para codificar y enviar una serie de mensajes  $\{M_i\}_{KAB}$
2. Bob lee los mensajes encriptados usando la correspondiente función  $D(KAB, M)$ .

Alice y Bob pueden funcionar con KAB mientras estén seguros que KAB no es conocida

Si el mensaje incluye algún valor acordado entre Alice y Bob, tal como una suma de comprobación del mensaje, entonces Bob conoce con certeza que el mensaje proviene de Alice y no ha sido saboteado, pero hay otros problemas:

**Problemas:** ¿Cómo envía un interlocutor a otro una clave de forma segura? El primer problema es el envío de la clave. ¿Cómo sabe un interlocutor que el mensaje proviene del otro interlocutor y no de un tercero con intenciones maliciosas? No hace falta saber la clave, sólo basta con reproducir el mensaje y si era un pago el cliente pagará 2 veces.

**Autenticación con servidor:** versión simplificada del protocolo N-S (Needham y Schroeder) y Kerberos. Estamos solicitando una clave y “algo más”.

Suponemos que el servidor tiene sus mecanismos de seguridad para comunicarse con los usuarios (S). Alice desea acceder a los archivos guardados en Bob.

Bob es un servidor de ficheros; Sara es un servidor de autenticación. Sara comparte KA con Alice y KB con Bob.

1. Alice envía un mensaje no encriptado a Sara identificándose y solicitando un ticket para acceder a Bob. □
2. Sara responde a Alice con  $\{\{\text{Ticket}\}\text{KB}, \text{KAB}\}\text{KA}$ . Consistente en un mensaje codificado según KA con un ticket (para comunicar con Bob para cada fichero) encriptado según KB y una nueva clave KAB.
3. Alice usa KA para desencriptar la respuesta.
4. Alice envía a Bob el ticket, su identidad y una respuesta R para acceder al fichero:  $\{\text{Ticket}\}\text{KB}, \text{Alice}, \text{R}$ .
5. El ticket es realmente  $\{\text{KAB}, \text{Alice}\}\text{KB}$ . Bob usa KB para desencriptarlo, chequea la identidad y usa KAB para encriptar las respuestas a Alice.

Un ticket es un mensaje encriptado conteniendo la identidad del principal solicitante y una clave compartida para la sesión

**Problema:** en esta versión simplificada no existe protección frente a repetición de mensajes de autenticación antiguos, el servidor conoce todas las claves y por tanto solo es aplicable donde se pueda tener el servidor bien controlado. No aplicable en el comercio electrónico.

**Autenticada con claves públicas:** está dirigido de la parte pública hacia la parte privada. Debemos asegurarnos que cuando obtenemos una clave pública es verdaderamente del propietario de la misma. Podríamos comprobar si el mensaje se ha manipulado mediante el uso de varias técnicas. Lo que se cifra con la clave pública se puede descifrar con la clave privada. A su vez, lo que se cifra con la clave privada, se puede descifrar con la clave pública.

Supongamos que Bob ha generado un par de claves pública/privada  $\langle \text{KBpub}, \text{KBpriv} \rangle$ , así que les permite a Bob y Alice establecer una clave secreta compartida KAB.

1. Alice obtiene un certificado firmado por una autoridad de confianza que posee la clave pública de Bob, KBpub
2. Alice crea una clave compartida KAB, la encripta según KBpub un algoritmo de clave pública y envía el resultado a Bob, de esta forma sólo podrá desencriptarlo Bob con la clave KBpriv.
3. Bob usa KBpriv para desencriptar KAB.

Si desean asegurar que el mensaje no ha sido manipulado, Alice puede incluir algún dato aceptado por ambos y Bob chequearlo.



¿Por que no comunicarse directamente con sus claves públicas? Porque resulta muy lento debido al gran tamaño de las claves.

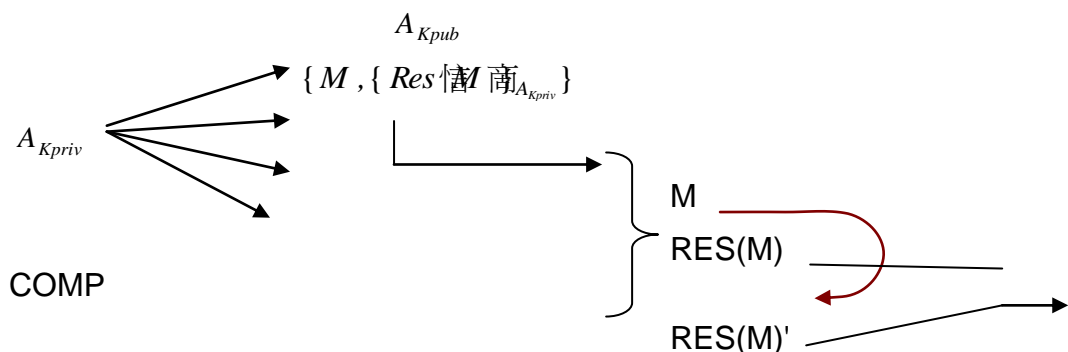
**Problema:** un interlocutor malicioso intercepta la petición de la clave pública a la entidad certificadora y la suplanta respondiendo con su clave pública y así se puede descryptar los mensajes que A pretendía enviar a B.

**Solución:** firma digital.

**Firma digital con resumen seguro:** se utilizan para comprobar si el mensaje ha sido manipulado, se basan en una modificación irreversible sobre el mensaje que únicamente conoce el firmante. Esto se consigue encriptando el mensaje empleando una clave sólo conocida por el firmante (su clave privada).

Alice quiere publicar un documento M de forma que cualquiera pueda verificar su procedencia.

1. Alice calcula un resumen de longitud fija del documento Resumen (M).
2. Alice encripta el resumen con su clave privada, lo adjunta a M y hace el resultado  $(M, \{Resumen(M)\}_{K_{priv}})$  público.
3. Bob obtiene el documento firmado, extrae M y computa Resumen (M).
4. Bob usa la clave pública de Alice para descryptar  $\{Resumen(M)\}_{K_{priv}}$  y lo compara con el resumen calculado por él. Si coincide, entonces la firma es válida.



Si son iguales tenemos dos datos:

- El mensaje pertenece a Alice.
- Como los resúmenes son iguales podemos decir que son del mismo M.

El interlocutor A genera un resumen, lo encripta con su clave privada y lo adjunta al mensaje y hace el resultado público. Esto es firmar un mensaje.

El interlocutor B obtiene dicho documento, extrae M y computa resumen  $h=H(M)$  para comprobar que su resultado es igual al que obtiene descryptando con la clave pública de A el resumen generado por A. Si coincide entonces la firma es válida.

Importante: el verificador (B) debe asegurarse que la clave pública empleada es la que realmente pertenece al firmante y esto debe hacerse mediante certificados.

Dado M debe ser fácil calcular h, pero dado h, es muy difícil calcular M. Y dado M debe ser difícil encontrar otro M' tal que  $H(M)=H(M')$ . Esto es la dispersión en un sentido.

La función de resumen debe ser segura frente al **ataque de cumpleaños**. La probabilidad de encontrar un par idéntico en un conjunto es mucho mayor que la de encontrar una pareja para un individuo dado. Con paciencia...

Para entendernos: caso práctico el sujeto A es un director de empresa corrupto y quiere contratar a un sujeto (B) con un contrato desfavorable. Bien el sujeto A prepara dos contratos uno M (favorable) y otro M' (desfavorable). El sujeto A fabrica varias versiones de M y M' sutilmente diferentes y compara los valores de dispersión de todos los M con todos los M' buscando un par igual. Al encontrarlo envía el contrato favorable al candidato a empleado (B). B lo firma digitalmente usando su clave privada. Cuando A lo recibe sustituye M por M' pero manteniendo la firma digital del sujeto B.

**Certificados:** es una sentencia firmada por un principal (entidad certificadora de confianza) que sirve de credencial y/o autenticación. Un certificado necesita:

- Formato estándar.
- Acuerdo sobre la forma en que se constituyen las cadenas de certificados, para que todos puedan constituirlos e interpretarlos.
- Fechas de expiración de forma que puedan ser revocados si están caducados.

Bob es un banco. Cuando sus clientes establecen contacto con él necesitan estar seguros de estar hablando con Bob en el banco, incluso si nunca han contactado con él antes. Bob necesita autenticar sus clientes para darles acceso. Alice atestiguando su número de cuenta obtiene el certificado de su banco:

---

1. <i>Tipo de certificado:</i>	Número de cuenta
2. <i>Nombre:</i>	Alice
3. <i>Cuenta:</i>	6262626
4. <i>Autoridad certificadora:</i>	Banco de Bob
5. <i>Firma:</i>	$\{Resumen(campo\ 2 + campo\ 3)\}_{K_{Bpriv}}$

---

El certificado se firma con la clave privada de Bob. Alice utiliza el certificado para comprar, un vendedor Carol puede aceptar el certificado siempre q ella pueda validar la firma. Carol necesita la clave pública de Bob para estar segura de su autenticidad. Alice puede firmar con un certificado falso asociando su nombre a la cuenta de otro. Simplemente generaría un nuevo par de claves KB'pub y KB'priv y generaría un certificado falso.

Carol necesita un certificado donde figure la clave pública de Bob, firmado por una entidad conocida y de fiar. Supongamos que Fred representa a la Federación de Banqueros, y este certifica las claves públicas de los bancos. Entonces Fred emitirá un certificado de clave pública para Bob:

1. <i>Tipo de certificado:</i>	Clave pública
2. <i>Nombre:</i>	Banco de Bob
3. <i>Cuenta:</i>	$K_{Bpub}$
4. <i>Autoridad certificadora:</i>	Fred, la Federación de Banqueros
5. <i>Firma:</i>	$\{Resume(campo2+campo3)\}_{K_{Fpriv}}$

Es lógico que este certificado dependa de la clave pública de Fred, de modo que tenemos un problema recursivo de autenticidad.

La dificultad está al elegir una autoridad fiable de donde pueda arrancar una cadena de comunicaciones.

**Control de acceso:** digamos que además de la autenticación y reconocimiento del cliente, los servidores de recursos distribuidos necesitan comprobar si dicho cliente tiene permisos para acceder a dicho recurso en concreto. Para ello se crean dominios de protección que son un conjunto de **pares <recurso, derechos>** que listan los recursos que pueden ser accedidos por todos los procesos en ejecución dentro de un dominio y especificando las operaciones permitidas sobre cada recurso.

Hay dos implementaciones posibles:

- **Listas de control de accesos (ACL):** se almacena una lista con cada recurso, con una entrada de la forma <dominio, operación> para cada dominio que tenga acceso al recurso y especificando las operaciones permitidas en el dominio. Los accesos son de la forma **<operación, principal, recursos>**. (como los permisos de UNIX).
- **Habilitaciones asociadas a un proceso:** cada proceso, según el dominio en que esté ubicado, aloja un conjunto de habitaciones. Una habitación es un valor binario que actúa como clave de acceso permitiendo al que lo posee acceder a ciertas operaciones sobre un recurso específico. Los accesos son de la forma **<operación, usuario, habitación>**. Las habitaciones deben ser infalsificables. Las habitaciones pueden caer en manos de otros principales como resultado de un ataque. Resulta difícil cancelar las habitaciones.

**Firewalls:** es una protección en una Intranet. Filtra las comunicaciones entrantes y salientes. No protegen de ataques desde el interior, ni de denegación de servicio. Solo protegen de ciertos tipos de mensajes externos definidos como maliciosos.

### 3. Algoritmos criptográficos

**Simétricos:** son los que usan la **misma clave para codificar y decodificar** (clave secreta compartida). Ataque: fuerza bruta. Solución: aumentar la longitud de la clave. (DES Data Encryption Standard, TEA más sencillo, más rápido, y más seguro).

$$E(K, M) = \{M\}_k \quad D(K, E(K, M)) = M$$

La misma clave para E y D. M es fácil de computar si se desconoce K.

**Asimétricos:** claves de encriptación  $K_e$  y desencriptación  $K_d$  **diferentes** (clave pública). Tienen un alto coste computacional ya que las claves son muy grandes (512 bits). Se basa en el uso de funciones de **puerta falsa**, que es una función de un solo sentido con una salida secreta, es fácil calcularla en dirección pero impracticable de calcular su inversa a menos que se conozca un secreto.

$$D(K_d, E(K_e, M)) = M$$

**Protocolos híbridos:** usados en SSL (actualmente TLS). Es lo más típico. Criptografía asimétrica para transmitir la clave simétrica que se usará.

**De bloque o cadena (CBC):** se suele trabajar en bloques de 64 bits. Cada mensaje se subdivide en bloques, el último bloque se rellena hasta la longitud estándar si fuera necesario y cada bloque se encripta independientemente. La debilidad del primer bloque es la que permite averiguar la clave utilizando un algoritmo de tipo estadístico a partir de paquetes capturados.

Problema: los patrones repetidos pueden ser detectados. Se utiliza un vector de inicialización (debilidad WEP). En WiredEP el número de vectores de inicialización es limitado y en 30 minutos podemos tener todos los posibles vectores. Esto se solucionó con WPA.

**De flujo:** útil en aplicaciones en tiempo real, conversaciones telefónicas. Las muestras de datos pueden ser tan pequeñas como 8 bits o incluso 1 bit. Idea similar al white noise. Puede combinarse con CBC.

**Algoritmos de encriptación simétrica:** se han desarrollado y publicado muchos algoritmos, entre ellos:

- **TEA:** un simple pero efectivo algoritmo desarrollado en Cambridge. Emplea vueltas de sumas enteras, XOR y desplazamientos lógicos de bits para obtener la difusión y la confusión de los patrones de bits en el texto en claro. La clave tiene 128bits. Proporciona una encriptación de clave secreta rápida y razonablemente segura. Triple veloz que el DES y seguro contra ataques de fuerza bruta.
- **DES:** la función de encriptación proyecta un texto claro de 64 bits en una salida encriptada de 64 bits usando una clave de 56 bits. Debido al coste computacional se implemento VLSI sobre hardware.
- **IDEA:** emplea una clave de 128 bits para encriptar bloques de 64. La resistencia de IDEA ha sido analizada extensamente, y no se han encontrado debilidades significativas.
- **AES:** es el algoritmo de encriptación simétrica más ampliamente utilizado. Claves de 128 – 512.

**Algoritmos de encriptación asimétrica:** todos ellos dependen del uso de puerta falsa, funciones en un solo sentido con una salida secreta.

- **RSA:** se basa en el uso del producto de dos números primos muy grandes, empleando el hecho de que la determinación de los factores primos de números tan grandes es computacionalmente difícil como para considerarlo posible calcular. Hoy en día se usa ampliamente. El tamaño de la clave es de 512 – 2048 bits.

Para encontrar el par de claves e, d:

1. Elegir dos primos muy grandes, P y Q (mayor de 10100), y calcular:
  - $N = P \times Q$
  - $Z = (P-1) \times (Q-1)$
2. Para d elegir un número primo respecto a Z (es decir, d no tiene factores comunes con Z). Ilustramos los cálculos con valores pequeños de P y Q:
  - $P = 13, Q = 17 \rightarrow N = 221, Z = 192$
  - $d = 5$
3. Para encontrar e se resuelve la ecuación:
  - $e \times d = 1 \bmod Z$
  - e x d es el elemento más pequeño divisible por d en la serie  $Z+1, 2Z+1, 3Z+1, \dots$

$$\begin{aligned} e \times d &= 1 \bmod 192 = 1, 193, 385, \dots \\ 385 &\text{ es divisible por } d \\ e &= 385/5 = 77 \end{aligned}$$

Para encriptar según RSA, el texto se divide en bloques de  $k$  bits donde  $2k < N$  (el valor numérico de un bloque es siempre menor que  $N$ ;  $k$  entre 512 y 1024)

$$k = 7, \text{ entonces } 27 = 128 (< N = 221)$$

La **función de encriptación** de un bloque de texto  $M$  es:

$$E'(e, N, M) = M^e \bmod N, \text{ para } M, \text{ el texto cifrado es } M^{77} \bmod 221$$

La **función de desencriptación** del bloque cifrado  $c$  es:

$$D'(d, N, c) = c^d \bmod N$$

Rivest, Shamir and Adelman probaron que  $E'$  y  $D'$  son inversas mutuas:

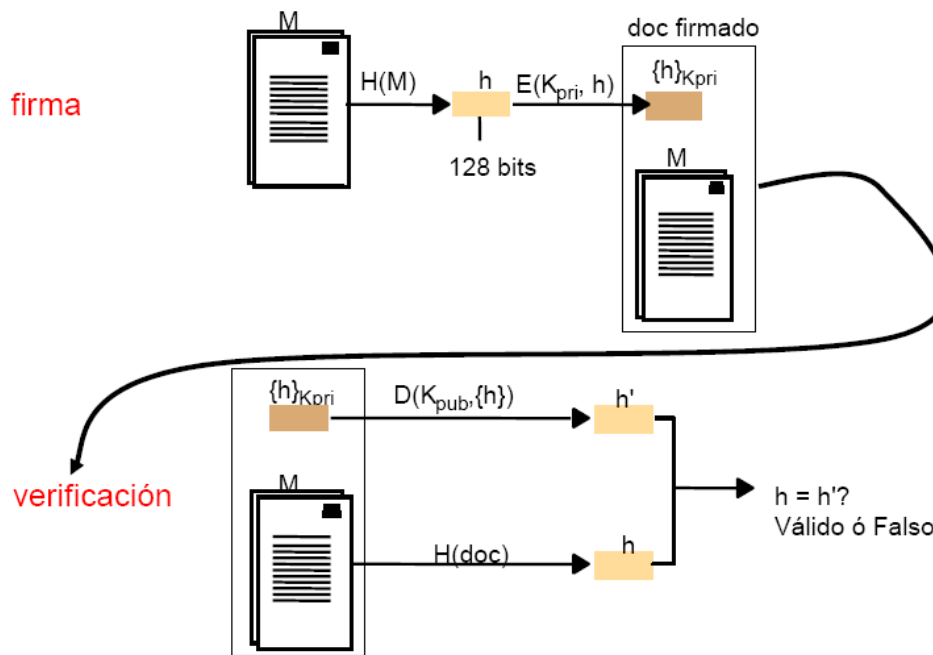
$$E'(D'(x)) = D'(E'(x)) = x \quad 0 \leq P \leq N$$

- **Curvas elípticas:** genera claves pública/privada basándose en las propiedades de las curvas elípticas.

#### **Algoritmos de resumen seguro:**

- **MD5:** Desarrollado por Rivest (1992). Calcula un resumen de 128 bits. Velocidad: 1740 kbytes/s. Cuatro vueltas con una de cuatro funciones no lineales sobre cada 32 bits de un bloque de 512 bits de texto. Es uno de los algoritmos más eficientes de hoy día.
- **SHA:** (1995) basado en MD4 de Rivest, pero más seguro, produce un resumen de 160-bit. Velocidad: 750 kbytes/s. Ofrece mayor seguridad contra los ataques por fuerza bruta y del cumpleaños.

**Firma digital con claves públicas:** la criptografía de clave pública se adapta bien a la generación de firmas digitales dado que es relativamente simple y no requiere ninguna comunicación entre el destinatario de un documento firmado y el firmante o cualquier otro.



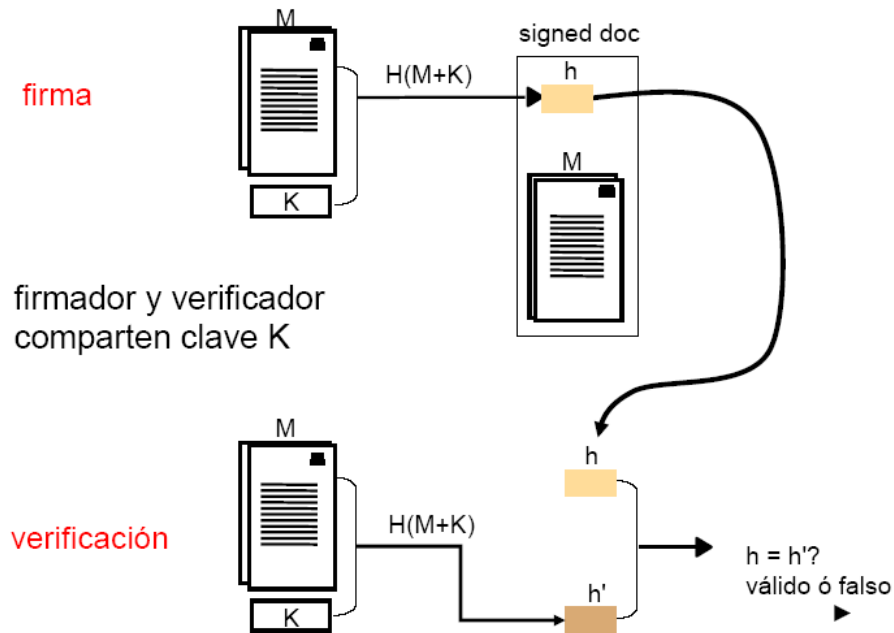
El algoritmo RSA es bastante adecuado para su uso para la construcción de firmas digitales.

**Firma digital con clave secreta (MAC):** para verificar las firmas habrá que descubrir la clave, y esto origina unos problemas, como son que el firmante debe conseguir que el verificador reciba la clave secreta empleada, debe ser necesario poder verificar una firma en varios contextos diferentes y el descubrimiento de una clave secreta empleada para una firma es poco deseable puesto que debilita la seguridad de las firmas realizadas con ella.

Dado un canal seguro proporciona comunicaciones seguras entre pares de procesos, se puede establecer una clave secreta empleando el método híbrido y así producir firmas de coste bajo, estas firmas se denominan Códigos de Autenticación de Mensajes (MAC), autentican la comunicación entre pares de principales basándose en el secreto compartido.

**Funcionamiento:** una vez establecido el canal seguro, se genera una clave aleatoria  $K$  para firmar, y la distribuye utilizando el canal a los principales que necesiten autenticar los mensajes recibidos de  $A$ . Se confía en que los principales no descubran la clave compartida. Para cualquier documento  $M$  que  $A$  desee firmar,  $A$  concatena  $M$  con  $K$ , calcula el resumen y envía el documento firmado a cualquiera que desee verificar la firma.  $K$  no queda comprometida por la publicación del resumen, dado que la función de dispersión oscurece su valor. El receptor  $B$ , concatena la clave secreta  $K$  con el documento recibido  $M$  y calcula su resumen. La firma es válida si  $h = h'$ .





**Comparativa algoritmos:**

	Tamaño de clave/tamaño de dispersión (bits)	Velocidad extrapolada (kbytes/sec.)	PRB optim. (kbytes/s)
TEA – simétricos	128	700	-
DES	56	350	7.746
Triple-DES	112	120	2.842
IDEA	128	700	4.469
RSA – asimétricos	512	7	-
RSA	2.048	1	-
MD5 - resumen	128	1.740	62.425
SHA	160	750	25.162

*perfil de prueba: Pentium II a 330 MHz*

**Protocolo de autenticación Needham- Schroeder:** es un protocolo de autenticación y distribución de claves secretas a los clientes para uso en una red local. El trabajo del servidor es proporcionar una forma segura por la que pares de procesos obtienen sus claves compartidas. Debe comunicarse con sus clientes usando mensajes encriptados.

El protocolo se describe para dos procesos arbitrarios A y B, pero en sistemas C/S. A es cualquier cliente que inicia una secuencia de solicitudes hacia el servidor B. La clave le viene dada a A por dos formas, una que A puede usar para encriptar los mensajes de B y otra para transmitir en modo seguro a B, esta clave es conocida por B, pero no por A.

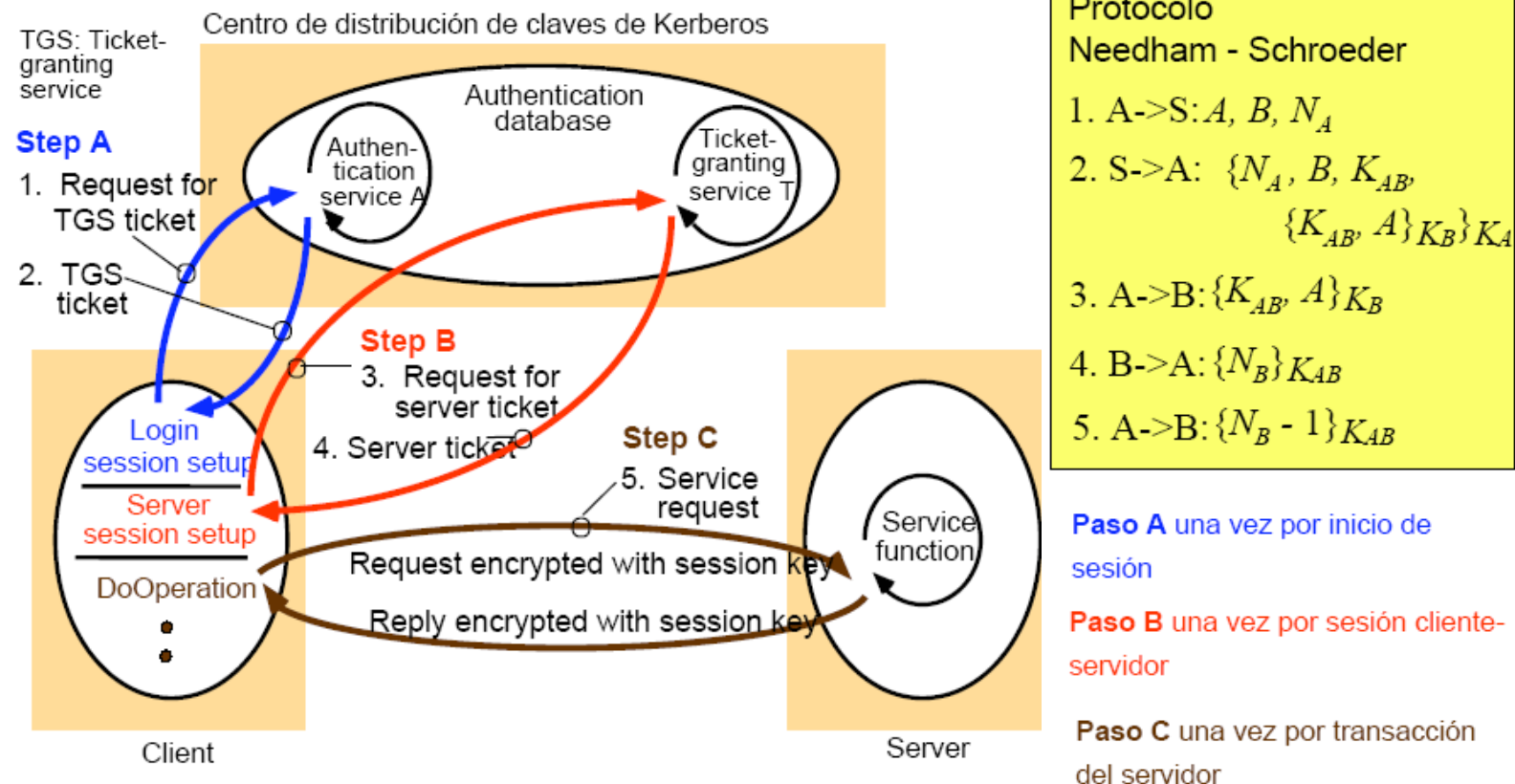
Encabezado Mensaje	Notas
1. A->S:     A, B, $N_A$	A solicita una clave a S para comunicarse con B
2. S->A: $\{N_A, B, K_{AB}, \text{Ticket}\}_{K_A}$	S devuelve un mensaje encriptado en la clave secreta de A, con una clave nueva $K_{AB}$ y un "ticket" encriptado en la clave secreta de B. La ocasión $N_A$ demuestra que el mensaje fue enviado en respuesta al anterior. A confía en que S envió el mensaje porque sólo S conoce la clave secreta de A
3. A->B: $\{K_{AB}, A\}_{K_B}$	A envía el "ticket" a B
4. B->A: $\{N_B\}_{K_{AB}}$	B descripta el "ticket" y utiliza la nueva clave $K_{AB}$ para encriptar otra ocasión $N_B$
5. A->B: $\{N_B - 1\}_{K_{AB}}$	A demuestra a B que fue el emisor del mensaje anterior devolviendo una transformación acordada sobre $N_B$ .

Existe una debilidad en el protocolo en la que B no encuentre razones para creer que el mensaje 3 sea reciente. Un intruso que se las apañe para obtener la clva  $K_{AB}$  y que haga copia del ticket y el autenticador  $\{K_{AB}, A\}_{K_B}$ , podría utilizarlos para iniciar un intercambio posterior con B suplantando a A. para que ocurra este ataque debe comprometerse un antiguo valor de  $K_{AB}$ . La debilidad puede remediarse añadiendo una ocasión o una marca temporal en el mensaje 3, de modo que se convierte en  $\{K_{AB}, A, t\}_{K_B}$ , B descripta este mensaje y comprueba que t es reciente. Ésta es la solución adoptada en Kerberos.

**Kerberos:** basado en el anterior (N-S), estandarizado e incluido en muchos SO (Windows, Linux). El servidor Kerberos crea una clave secreta compartida para cada usuario y la envía encriptada al computador del usuario dicha clave es el secreto compartido inicial. Funciona como el N-S.

Trata con 3 clases de objetos de seguridad:

- **Ticket**: una palabra enviada al cliente por el servicio de concesión de tickets y que verifica que el emisor se ha autenticado recientemente frente a kerberos. Los tickets incluyen un tiempo de expiración y una clave de sesión generada en ese momento.
- **Autenticación**: una palabra construida por un cliente y enviada al servidor para demostrar la identidad del usuario y la actualidad de cualquier comunicación con un servidor. Sólo puede usarse una vez. Contiene el nombre del cliente y una marca temporal y se encripta con la clave de sesión.
- **Clave de sesión**: una clave secreta generada aleatoriamente por kerberos y enviada a un cliente para su uso en la comunicación con un servidor en particular.



La seguridad de kerberos depende de la limitación de los tiempos de vida de sesión: el periodo de validez de los tickets se limita a unas pocas horas.

## TEMA 6: SISTEMAS DE ARCHIVO DISTRIBUIDO

### 1. Introducción

El compartir información almacenada es quizás el aspecto más importante de la compartición de recursos distribuidos. Se obtiene a gran escala en Internet principalmente por el uso de servidores Web, pero los requisitos para compartir en redes de área local e intranet conduce a una necesidad de un tipo de servicio diferente, uno que soporte el almacenamiento persistente de datos y programas de todos los tipos, en nombre de los clientes, y la consiguiente distribución de datos actualizados.

La gestión tradicional de ficheros engloba aspectos como: **organización**, **almacenamiento persistente**, recuperación, **nombrado**, compartición, protección, base de otros servicios o servidores (servicio de nombres o de directorio, servicio de impresión).

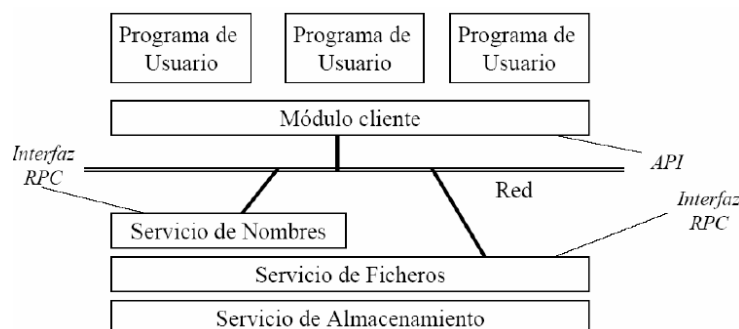
Un sistema de ficheros distribuido (SFD) soporta la compartición de la información en forma de archivos y recursos HW que dan soporte al almacenamiento persistente a través una Intranet. Proporciona acceso a los sistemas almacenados en un servidor con prestaciones y fiabilidad semejantes a los archivos almacenados en discos locales.

Un sistema de ficheros distribuido:

- Tiene las mismas funciones que el sistema de ficheros de un SO convencional, pero más complejo.
- Los usuarios y los dispositivos de almacenamiento se encuentran dispersos por la red.
- Adicionalmente, un SFD debe permitir: compartir información remotamente, movilidad de los usuarios, disponibilidad.

Un SFD se puede estructurar en cuatro componentes:

1. servicio de almacenamiento.
2. servicio de ficheros: identifica de forma única un fichero.
3. servicio de nombres o de directorio: a partir de un nombre obtiene la referencia, el camino al fichero.
4. módulo cliente (biblioteca de funciones o API).



Los objetivos que se pretenden son: transparencia, movilidad de usuarios, tolerancia a fallos, concurrencia, fiabilidad, seguridad y heterogeneidad del HW y SW.

## 2. El servicio de ficheros

Un servicio de ficheros permite a los programas almacenar y acceder a los archivos remotos del mismo modo que se hace con los locales, permitiendo a los usuarios acceder a sus archivos desde cualquier computador de una Intranet.

Otros servicios, como el servicio de nombres, el servicio de autenticación de usuarios y el servicio de impresión, pueden ser implementados más fácilmente si hacen llamadas al servicio de archivos.

El **servicio de ficheros** define el concepto de fichero, así como las características y propiedades de los mismos.

La estructura de los ficheros, para la mayoría de los sistemas operativos los ficheros son meras secuencias de bytes, sin ningún tipo de estructura.

Los ficheros suelen tener atributos, información acerca del fichero, pero que no forman parte del contenido del mismo.

La mayoría de los ficheros son mutables: existen propuestas de ficheros inmutables, ventajas para aplicar técnicas de caché y replicación, requieren mucho espacio en disco.

**Modelos de acceso a ficheros remotos:** se pueden combinar los dos modelos, ejemplo: NFS.

- **Servicio remoto:** las operaciones se realizan en el servidor (poco eficiente).
- **Caché de datos:** los ficheros se acceden de forma local (mejor rendimiento pero peor consistencia).

**Unidades de transferencia de datos:** en los sistemas que usan caché un aspecto básico es la unidad de información a transmitir. Existen 4 unidades básicas:

- **Ficheros:** simplifica el proceso (caché). Sobrecarga la red. AFS, Amoeba.
- **Bloques:** se baja los bloques que necesita. La caché es bastante sofisticada, menos sobrecarga. NFS
- **Bytes:** sólo se baja los bytes necesarios. No sobrecarga la red. La caché es muy sofisticada. Máxima flexibilidad, dificulta la implementación de la caché al tener los datos longitud variable.
- **Registros:** casos concretos de información por bloques, en sistemas en los que los ficheros tienen estructura.

**Semántica de ficheros compartidos:** varios usuarios pueden compartir un mismo fichero, hay que definir la semántica de las operaciones de lectura y escritura. El principal inconveniente es el sistema de comunicaciones.

- **Semántica UNIX:** a) Toda operación sobre un fichero es visible de forma instantánea por todos los procesos. (fácil implementación. con un servidor centralizado pero varios inconvenientes). b) emplear cachés en los clientes (problemas de inconsistencias).
- **Semántica tipo sesión:** una sesión es la secuencia de operaciones sobre un fichero que comienza por su apertura y acaba por su cierre. Las modificaciones realizadas sobre un fichero no son visibles hasta que el fichero se cierra. Al cerrar el cliente envía la copia al servidor.
- **Semántica ficheros inmutables:** los ficheros no se pueden modificar una vez se han creado. No hay inconsistencias, pero qué ocurre cuando dos procesos intentan crear el mismo fichero? O si un proceso lee un fichero que va a ser reemplazado?
- **Semántica de transacciones atómicas:** evolución del tipo sesión. En vez de tratar con cierres de fichero trata con pequeñas transacciones como en BD "commit". Evita inconsistencias y si algo falla puede volver al último punto consistente, difícil implementación.

### 3. Técnicas de implementación

Es importante estudiar la forma en la que los ficheros van a ser usados. El acceso aleatorio es poco frecuente, si se accede a un fichero se accede de principio a fin, los ficheros suelen ser pequeños, se lee más que se escribe.

Es raro compartir ficheros ya que estadísticamente la probabilidad de que dos aplicaciones distintas estén utilizando el mismo fichero es muy baja, además la mayoría de los procesos utilizan pocos ficheros.

**Tipos de servidores:** los servidores pueden ser de dos tipos:

- **Sin estado:** no almacenan información entre solicitudes de un mismo cliente. De esta forma no se requieren llamadas para abrir y cerrar ficheros, no se desperdicia memoria en tablas, no existe límite para el número de ficheros en uso y no se producen problemas si cae un cliente. Tolerancia a fallos. Ejemplo: **HTTP**.
- **Con estado:** almacenan información entre solicitudes de un mismo cliente. Se obtiene un mejor rendimiento, los mensajes de solicitud de servicio son cortos, es posible realizar operaciones de lectura anticipada y permiten el bloqueo de ficheros. Ejemplo: **FTP**.

**Técnicas de caché:** las técnicas de caché se basan en retener en memoria principal aquellos datos que se han sido usados más recientemente (mejora la rapidez), como consecuencia, los accesos a la misma información se llevan a cabo en memoria. Uno de los aspectos que tenemos que considerar es la granularidad de la caché (grande, pequeña) y el tamaño (grande, pequeña, fija, variable) junto con la política de reemplazo.

Es importante la **localización** de la caché. Ordenadas de mejor a peor las posibles localizaciones son:

Localización	Coste acceso (acierto)	Ventajas	Inconvenientes
<b>Memoria cliente</b>	Nulo	Máximo rendimiento en caso de acierto. Poco Trásiego red. Escalable	Inconsistencia
<b>Memoria servidor</b>	Transferencia red	Transparente para cliente (NFS) rápido. Consistente. Permite semántica UNIX.	Trásiego red
<b>Disco cliente</b>	Acceso a disco	Fiabilidad frente a caídas del cliente. Gran capacidad. Útil para funcionamiento desconectado. Escalable.	Inconsistencia Lento (por tener que acceder al disco)
<b>Disco servidor</b>	Tred+ac.disco		Demasiado costoso

Cuando se utiliza caché en el cliente se pueden producir **inconsistencias** y entonces es necesario considerar cuando propagar las modificaciones y como verificar la validez de los datos.

#### Propagación:

**Escritura inmediata:** (write-through), cuando se modifica la caché se envían los cambios inmediatamente al servidor. Fiable antes fallos del cliente, pero su rendimiento se ve perjudicado. Adecuado para semántica UNIX.

**Post escritura:** (writeback), las modificaciones en la caché se posponen un tiempo y después se envían todas a la vez. Existen varias variantes:

Periódicamente.

Al cerrar el fichero (on close).

Cuando lo determine la política de reemplazamiento

Las escrituras no implican comunicación, enviar todas las modificaciones juntas es mejor que de una en una, menos tráfico, pero menos fiable. Adecuada para semántica de sesión o atómica.

**Validación:** la política de propagación de modificaciones especifica cuándo se actualiza la copia principal de un fichero cuando una de las copias de la caché es modificada, pero no establece cuándo actualizar las cachés. El contenido de una caché se vuelve inválido cuando otro cliente modifica la copia principal, es necesario comprobar si la caché del cliente es consistente con la principal, en caso contrario es necesario invalidar la caché y actualizar los datos. Existen dos estrategias:

- **Iniciada por el cliente:** contactan con el servidor para comprobar la validez de sus datos. Los datos se validan comparando la fecha de última modificación de la caché con la de la copia principal. La frecuencia de las comprobaciones depende de la semántica de acceso a los ficheros en la caché y del tráfico de mensajes en la red y el consumo



de la CPU. Existen 3 posibilidades: antes de cada acceso, periódicamente, al abrir el fichero.

- **Iniciada por el servidor:** el cliente informa al servidor cuando abre el fichero, indicando el tiempo de apertura del mismo, el servidor mantiene un registro de los ficheros abiertos, lo cual le permite detectar posibles inconsistencias, no hay problemas si hay varias aperturas en modo lectura. En caso de solicitud de apertura conflictiva, el servidor puede denegar la apertura del fichero, encolar la petición o deshabilitar la caché y trabajar en modo de operación tipo servicio remoto. Cuando un cliente cierra el fichero, envía las modificaciones al servidor. Algunos de los inconvenientes son que viola el modelo C/S, el código de los clientes y servidores es más complejo y es necesario utilizar servidores con estado.

**Replicación de ficheros:** un fichero replicado es aquél del que existen varias copias, cada una de las cuales está en un servidor diferente. Una réplica se asocia a un servidor, mientras que una copia en caché se asocia a un cliente. Una réplica es más persistente, conocida, disponible y segura que una copia en caché. La existencia de réplicas depende generalmente del rendimiento y la disponibilidad, una copia de caché depende de la localidad en los accesos a los ficheros.

La replicación de ficheros aporta las siguientes ventajas: aumenta la disponibilidad, aumenta la fiabilidad, mejora el tiempo de respuesta, reduce el tráfico en la red, mejora el rendimiento, beneficia la escalabilidad, permite trabajar en modo de operación desconectada.

Para que la replicación sea transparente a los usuarios hemos de considerar:

- **Sistema de denominación de las réplicas:** ¿cómo distinguir una réplica de otra si ambas tienen el mismo identificador? Un servidor de nombres debería hacer corresponder al identificador la réplica más conveniente.
- **Control de la replicación:** explícita o implícita.

Protocolos de actualización de réplicas:

- **Replicación de sólo lectura:** se aplica a ficheros inmutables
- **Protocolo Escribir en todos-Leer de cualquiera:** las escrituras son costosas, problemas si un servidor cae.
- **Protocolo basado en la disponibilidad de copias:** problemas de inconsistencias en caso de partición de la red.
- **Protocolo de copia primaria:** se lee de cualquiera, se escribe en una, problemas si cae el servidor primario
- **Protocolo basado en quórum:** los clientes requieren un quórum de  $N_r$  servidores para leer y  $N_w$  para escribir  
$$N_r + N_w > N \text{ (} N = \text{número de réplicas)}$$

#### 4. Casos de estudio

**NFS (Network File System):** desarrollado por SUN, es el sistema de ficheros distribuido estándar, el más empleado. Su objetivo es **permitir el acceso transparente** a ficheros remotos entre clientes y servidores que se ejecutan en sistemas heterogéneos de máquinas, S.O y redes. Se basa en el modelo **C/S**, normalmente cada computador puede actuar como cliente y servidor, la comunicación entre clientes y servidores se lleva a cabo utilizando **RPC** de Sun.

Utiliza dos **protocolos** que describen sus dos acciones principales:

- **Montaje de directorios exportados por servidores:** el cliente solicita el montaje de un directorio exportado, el cliente no le indica al servidor el lugar donde montará dicho directorio (los nombres de ruta no se traducen en el servidor). El servidor responde con un manejador, toda la operación incluirá el manejador. Existen dos tipos de montaje: rígido o flexible y automontaje.
- **Acceso a ficheros y a directorios.**

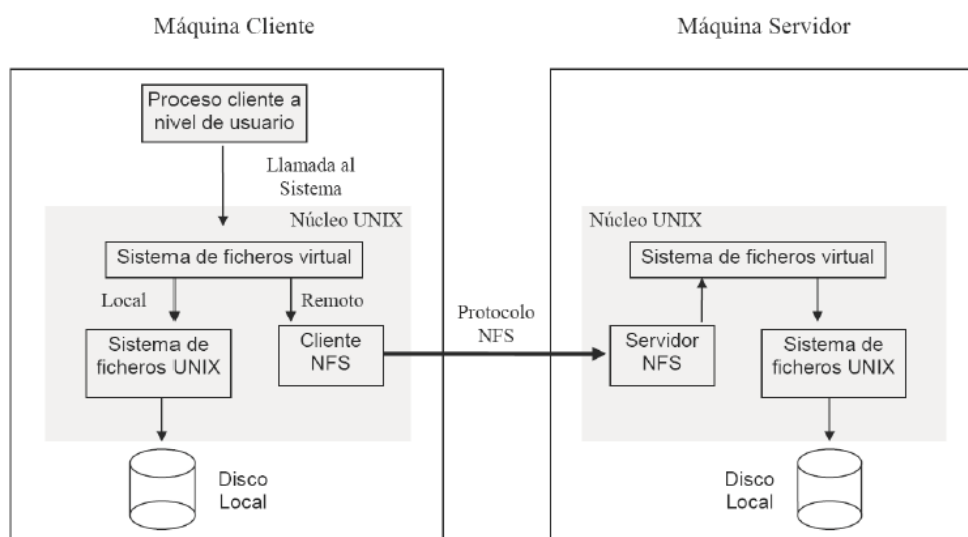
En cuanto a la **transparencia**, se consigue en cuando:

- **Acceso:** parece un SF local, aunque sea distribuido y heterogéneo.
- **Localización:** el fichero cuelga del árbol local de directorios.
- **Contra fallos:** un servidor falla, se suspenden los servicios que proporcionaba, pero los procesos no se bloquean y cuando se recupera el servidor → los procesos vuelven a tener acceso a los servicios y siguen de donde se quedaron.
- **Rendimiento:** es bueno, parece que estemos trabajando en local.

Pero **no** es **transparente** en cuanto a:

- **Migración:** montado de directorios.
- **Replicación:** no soporta replicación de ficheros actualizables.
- **Escalabilidad:** para altas cargas hay que aumentar servidores. (mejor utilizar AFS o Coda).

A continuación se observa la arquitectura NFS:



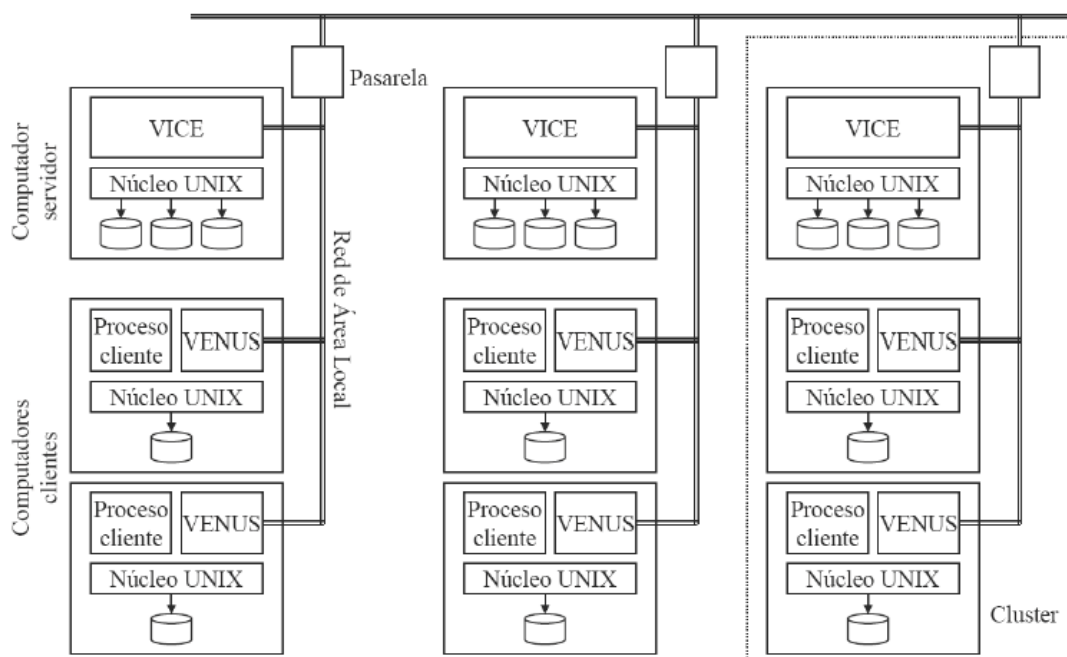
El **sistema de ficheros virtual (VFS)**: es similar a la tabla de ficheros de UNIX, cada entrada contiene un nodo índice virtual (vnode), si el fichero es local el v-node contiene un par dispositivo-nodo índice, si es remoto, contiene el manejador. Módulo cliente emula la semántica de las primitivas del sistema de ficheros de UNIX, está integrado en el núcleo de sistema operativo. (Acceso transparente y eficiencia).

Técnicas para **aumentar el rendimiento**: se transfieren 8K bytes en cada mensaje, el cliente solicita la lectura de bloques de forma anticipada (read ahead), las escrituras no se transmiten al servidor hasta que se escriben 8K bytes o se cierra el fichero, se utilizan cachés en el cliente y en el servidor:

- **Caché en el servidor**: se utiliza el sistema de buffer caché de UNIX, diferencia: se utiliza un esquema write-through en el caso de ficheros remotos.
- **Caché en el cliente**: una caché para atributos y otra para bloques de datos, para solucionar los problemas de inconsistencias las cachés son validadas periódicamente (3 segundos en el caso de ficheros, 30 segundos en el caso de directorios)

**AFS (Andrew File System)**: los objetivos de diseño son proporcionar un **buen rendimiento** en sistemas grandes (más de 5000 estaciones, escalable), su diseño se deriva del conocimiento de cómo se usan los ficheros en sistemas UNIX típicos. Los ficheros se transmiten **enteros** entre servidores y clientes, los ficheros transmitidos se almacenan en el disco local durante largos periodos de tiempo. Es compatible con NFS y los programas no necesitan recompilación ni nada especial para utilizar este SF (**transparencia** para el usuario/programador).

La arquitectura ASF es:



Existen **dos espacios de nombres**, ficheros locales y ficheros remotos, que se muestran en el árbol de directorios (como si fuesen uno solo) juntos gracias a los enlaces simbólicos. Cuando un cliente abre un fichero, si no tiene copia local el servidor que lo tenga se lo envía y trabaja con la copia local. Al cerrarlo el cliente lo actualiza en el servidor y mantiene su copia local.

Los ficheros se agrupan en volúmenes, particiones. Los servidores implementan el servicio de ficheros y la estructura jerárquica de directorios la proporciona Venus (cliente). Los programas de usuario utilizan los ficheros de acuerdo con el esquema de nombres de UNIX. Cada fichero tiene un identificador único de 96 bits. Implementa las llamadas al sistema open, close, read y write. Utiliza semántica de tipo sesión.

**CODA**: objetivos de diseño: es el sucesor de AFS y los objetivos de su diseño se centran en mejorar las limitaciones de AFS que son:

- Replicación sólo aplicable a volúmenes de sólo lectura (limita escalabilidad)
- Retrasos debido a un gran número de componentes, periodos de espera por parte de los usuarios. (limita la rapidez)
- No es aplicable a computación móvil (portátiles) (limita la heterogeneidad).

Las **ventajas** que introduce la replicación de volúmenes completos son:

- Ficheros accesibles a cualquier cliente mientras exista alguna réplica que no haya caído.
- Aumento del rendimiento a consecuencia de compartir la carga de atender peticiones.

El objetivo es **solventar las limitaciones** manteniendo sus buenas características:

- Cada usuario puede beneficiarse de un **almacén de ficheros compartidos (VSG: conjunto de servidores que tienen una réplica de un volumen)**.
- Los usuarios pueden seguir **trabajando en modo desconectado** si el almacén no está disponible (**AVSG: subconjunto de VSG que puede ser usado por un cliente en un instante dado y cambia de tamaño cuando se incorporan o desconectan servidores**).

Al **modo desconectado** se puede llegar forzosamente debido a un fallo de red, caída de los servidores o voluntariamente porque estamos trabajando con un portátil. Lo cual requiere que la estación de trabajo desconectada tenga todos los ficheros que necesita.

**El principio de diseño** es “Es más seguro mantener las copias de los ficheros en los servidores que en las caches de las estaciones de trabajo”, puesto que han de ser revalidadas contrastándose con las copias en servidor. Si se está en

modo desconectado la validación se produce cuando se cambia de modo de trabajo (al volver a modo conectado).

En modo normal el funcionamiento de Coda es similar a AFS.

Utiliza una variante débil de la **semántica de tipo sesión**, es una consecuencia del esquema de replicación optimista. Cuando se abre un fichero se garantiza el acceso a la copia más reciente en el AVSG actual, si no hay ningún servidor disponible se accede a la copia local.

**La estrategia de replicación optimista:** utiliza un Coda Vector Version CVV para controlar los cambios en las versiones de los ficheros para que se puedan realizar modificaciones en modo desconectado y luego puedan ser validadas en el servidor. El CVV permite actualizar los ficheros y detectar inconsistencias. Las inconsistencias se pueden reparar de forma automática.

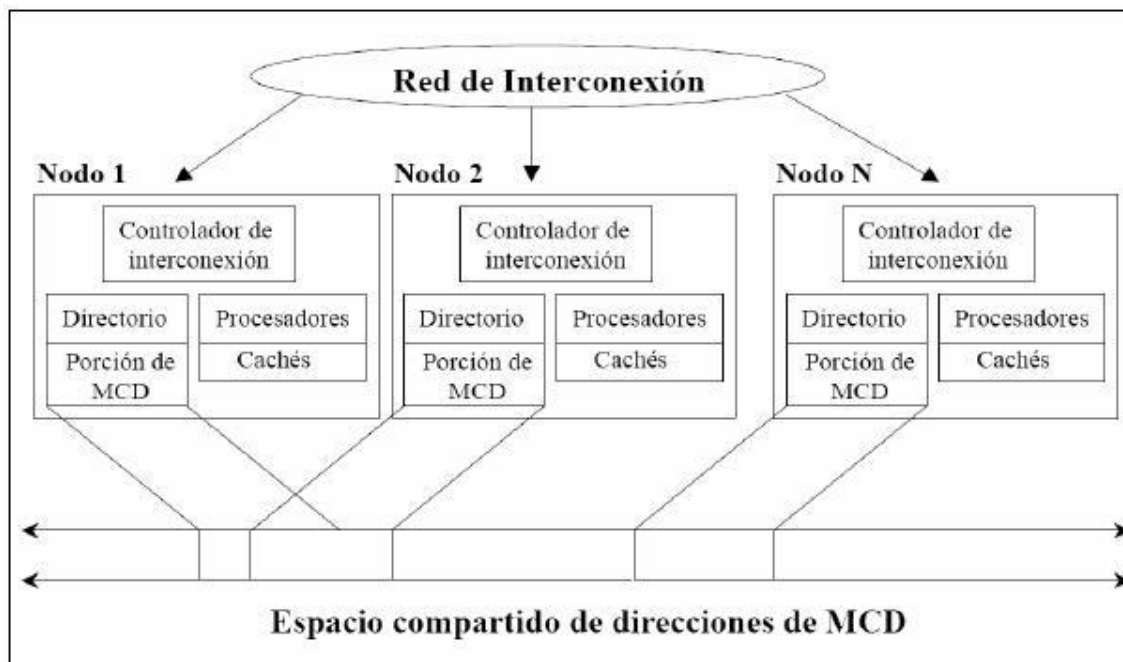
## TEMA 7: MEMORIA COMPARTIDA DISTRIBUIDA

### 1. Introducción

La memoria compartida distribuida (MCD) es una abstracción utilizada para compartir datos entre computadores que no comparten memoria física.

Tradicionalmente, la comunicación entre procesos que se ejecutan en sistemas distribuidos se basa en un modelo de intercambio de información: paso de mensajes (PM), RPC, etc. Éstos procesos comparten información mediante el paso de la misma **por valor** utilizando primitivas del tipo *enviar()* y *recibir()*. Sin embargo éstas técnicas suponen un modelo de programación complejo. En PM, los accesos remotos son explícitos por el programador, se pueden producir pérdidas de mensajes, interbloqueos, problemas para compartir estructuras complejas, etc.

Por ello, la principal característica de la MCD es que ahorra al programador todo lo concerniente al paso de mensajes al escribir sus aplicaciones. Se proporciona a los procesos un espacio de direcciones compartido, al que se accede mediante primitivas *leer()* y *escribir()* pasando la información **por referencia**. Además, desde un punto de vista hardware, proporciona ventajas como facilidad de construcción, escalabilidad, tolerancia a fallos, heterogeneidad, bajo coste, etc.



Arquitectura de un sistema de memoria compartida distribuida

En cuanto a la organización de la MCD, se presentan varias posibilidades para determinar qué es lo que realmente se comparte: líneas de caché (*Dash*), páginas de memoria virtual (*Ivy*), variables (*Munin*), tipos abstractos de datos (*Orca*, *Linda*), objetos (*Jaco*), etc.

## 2. Algoritmos de MCD

Los algoritmos que implementan MCD se pueden clasificar dependiendo de si utilizan técnicas de migración y/o replicación de datos.

- **Migración:** explotan la localidad de los accesos y reducen el número de accesos remotos.
- **Replicación:** permiten que se puedan realizar a la vez múltiples accesos locales de lectura.

	Sin Replicación	Con Replicación
Sin migración	Centralizado	Actualización
Con Migración	Migración	Invalidación

*Configuraciones de los algoritmos de MCD*

### 2.1 Algoritmo de Servidor Central

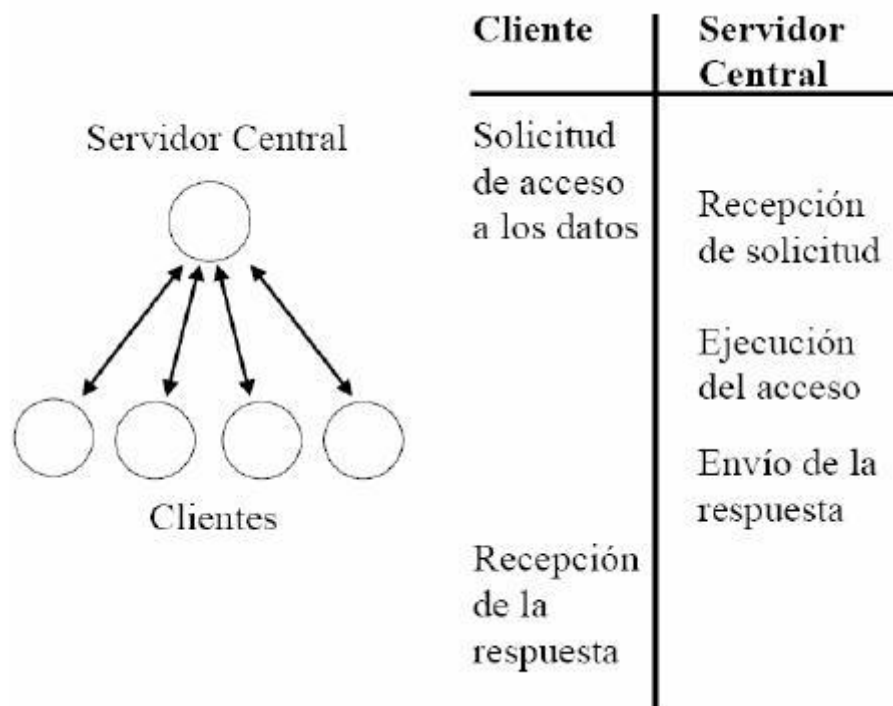
Un servidor central es el responsable de servir todos los accesos a los datos compartidos y mantener la única copia de los mismos. Tanto las operaciones de lectura como las de escritura requieren el envío de un mensaje de petición al servidor central por parte del proceso que ejecuta la operación. El servidor responde con el dato si se trata de una operación de lectura o con una aceptación en el caso de una operación de escritura. Sólo un proceso a la vez puede realizar una operación de lectura o escritura.

La principal ventaja de este algoritmo es su simplicidad, implementable mediante un protocolo de tipo solicitud-respuesta. En cambio, es un algoritmo centralizado y acarrea los inconvenientes de este tipo de estructuras (dependencia máxima del servidor).

Este algoritmo es suficiente cuando los accesos a los datos compartidos no son frecuentes, y cuando el ratio de lecturas frente a escrituras es bajo.



Se propone una alternativa a éste método, que consiste en repartir los datos entre distintos servidores. Ello supondría que los clientes deben ser capaces de localizar el servidor adecuado para realizar un determinado acceso. Los datos se pueden dividir por direcciones y usar una función que determine el servidor con el que contactar.



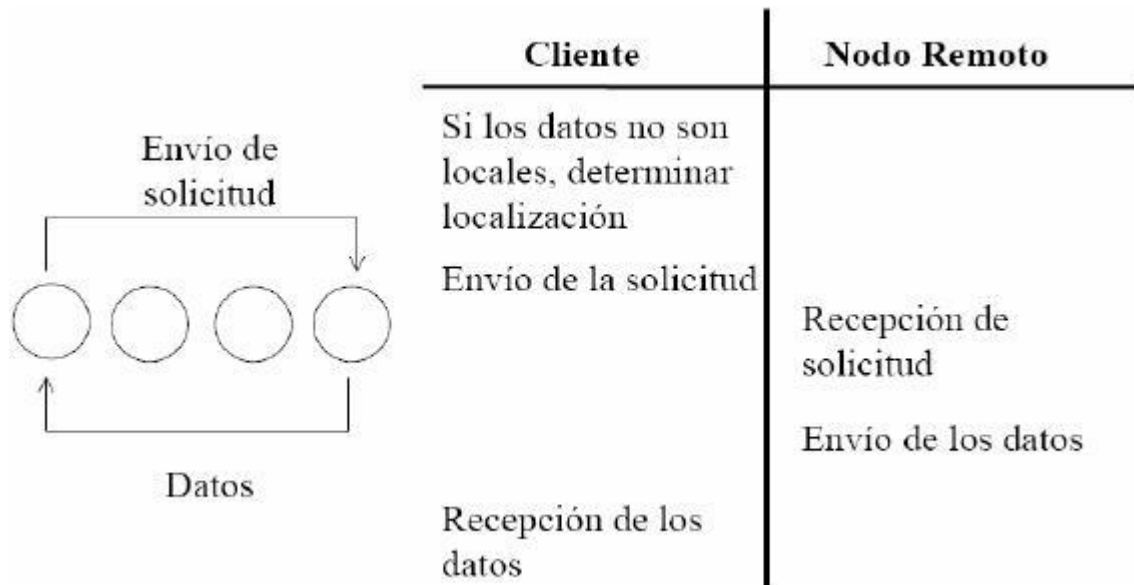
*Esquema de funcionamiento del algoritmo de servidor central*

## 2.2 Algoritmo de migración

Este algoritmo es de tipo *servidor distribuido dinámico*, lo que implica que cada nodo es a su vez cliente y servidor. Cada nodo tiene información sobre la localización de los bloques de MCD, aunque esta información no siempre está actualizada.

La principal ventaja de este método es que si el grado de localidad de acceso es alto, no se producen costes de comunicación, aunque la situación contraria puede ser un inconveniente importante. Además, si el tamaño de bloque de datos es igual al tamaño de una página de memoria virtual (o un múltiplo) se puede insertar dentro del sistema de memoria virtual del S.O.

Para localizar un bloque dentro del sistema de MCD se pueden usar varias técnicas: localización estática, difusión, búsqueda encadenada, etc.



*Esquema de funcionamiento del algoritmo de migración*

### 2.2.1 Búsqueda encadenada

Esta técnica de localización de bloques consiste en asignar a cada uno de los bloques un propietario, de modo que todos los nodos conocen dónde está el bloque de datos.

Hay dos tipos de propietarios: el real y el probable. El propietario real es aquél en que se encuentra el objeto realmente, mientras que el probable fue un propietario real que sabe a qué nodo ha emigrado el bloque de datos. De este modo las solicitudes se envían al propietario conocido, si éste es un propietario probable, se reenvía el mensaje al procesador que él cree que es el propietario real.

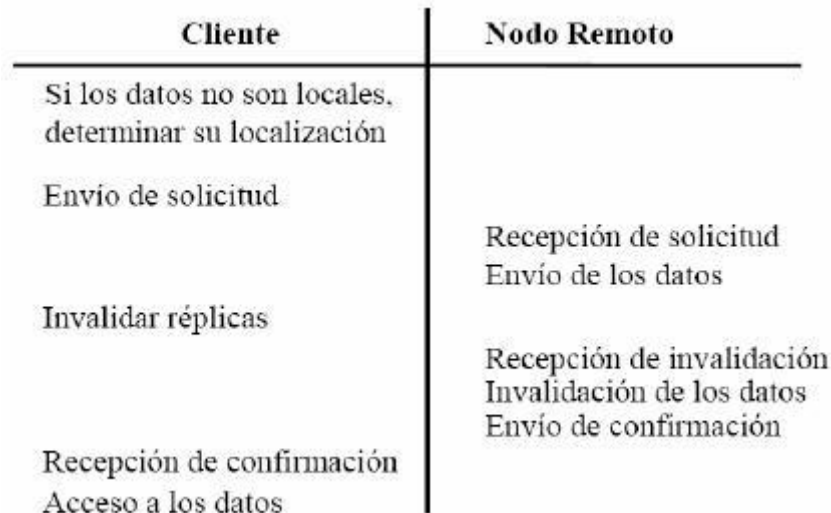
Si la migración es muy frecuente y hay muchos nodos, se aconseja difundir la dirección del propietario real al resto de procesadores con cierta frecuencia.

### 2.3 Algoritmo de replicación mediante invalidación

Este algoritmo aporta una ventaja respecto a los anteriores, y es que permite que varias operaciones de lectura se puedan ejecutar concurrentemente de forma local en varios nodos. El mismo bloque de memoria puede estar en varios nodos en modo lectura, sin embargo, sólo uno tendrá acceso de escritura. Cuando un bloque se modifica, se invalidan las demás copias de ese bloque en el resto de nodos. Esto provoca que las operaciones de escritura sean costosas, aunque supone una ventaja si el ratio de lecturas respecto al de escrituras es elevado.

Una operación de lectura sobre un bloque que no es local implica una comunicación para adquirir una copia de sólo lectura del bloque de datos y cambiar, si es el caso, a sólo lectura la copia de lectura/escritura.

Una operación de escritura sobre los datos de un bloque que no es local o que es local pero tiene permisos de sólo lectura, requiere que todas las copias de lectura sean invalidadas antes de que la operación de escritura se pueda llevar a cabo.



*Ejemplo de acceso a un bloque para lectura*

Cada bloque de datos tiene asignado un propietario, que es el nodo responsable de mantener una lista de los procesadores que tienen una copia de sólo lectura de dicho bloque.

Cuando se produce un fallo de lectura o escritura se transmite un mensaje al propietario del bloque. Si el fallo es de lectura, el propietario replica con una copia del bloque de datos, y añade al nodo solicitante a la lista de procesadores con copia.

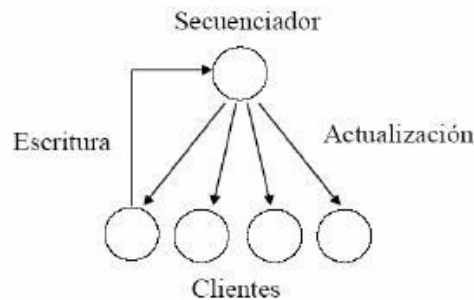
Si el fallo es de escritura, la propiedad del bloque se transmite del procesador propietario al que solicita la operación de escritura, se envía un mensaje de invalidación a todos los procesadores presentes en la lista de procesadores con copia. Los derechos de acceso del nuevo propietario se establecen a lectura y la lista de procesadores con copia se vacía.

## **2.4 Algoritmo de replicación mediante actualización**

Los bloques de datos pueden ser modificados en cualquier nodo siguiendo un protocolo de tipo múltiples lectores/múltiples escritores. Por tanto, los accesos a los bloques replicados han de ser serializados para mantener la consistencia. Esto implica que existe un orden global de acceso a los datos compartidos y que todos los nodos implicados observan los accesos en dicho orden.

Se pueden seguir dos estrategias de serialización:

- Serializar globalmente las operaciones de escritura: las operaciones de lectura se realizan de forma local, mientras que las de escritura se difunden al resto de réplicas de forma atómica.
- Usar un secuenciador global: un proceso que se ejecuta en un nodo asigna números de secuencia a las operaciones de escritura. Cuando una modificación llega a un nodo, el número de secuencia es verificado.



Cliente	Secuenciador	Resto de clientes
Si es una escritura, enviar los datos	Recepción de datos Adición de un número de secuencia Difundir	
Recepción de los datos Actualización de los datos locales		Recepción de los datos Actualización de los datos locales

*Uso de un secuenciador global para serializar los datos*

### 3. Modelos de consistencia de la memoria

Un modelo de consistencia de memoria especifica las garantías de consistencia que un sistema MCD realiza sobre los valores que los procesos leen desde los objetos, dado que en realidad acceden sobre una réplica cada objeto y que múltiples procesos pueden actualizar los objetos.

En los sistemas monoprocesadores tradicionales, el modelo de consistencia de la memoria es conocido como consistencia estricta o atómica. Este modelo establece que la lectura de una posición de memoria siempre devuelve el valor más reciente de la misma. Sin embargo, presenta el inconveniente de no poderse garantizar sobre un sistema basado en MCD.

Los modelos de consistencia más fuertes presentan la ventaja de ser más intuitivos y fáciles de utilizar, aunque limitan el conjunto de posibles optimizaciones que se podrían usar para aumentar el rendimiento de los

programas. Además, tienen que satisfacer restricciones de tiempo más estrictas.

Los modelos de consistencia más débiles, en cambio, permiten aplicar un gran número de técnicas de optimización, y tienen que satisfacer menos restricciones de tiempo. Además, se obtiene un mayor beneficio cuanto mayor es el tiempo de latencia de la memoria. Sin embargo, el modelo de programación es más complejo, ya que la memoria se comporta de forma diferente a lo que normalmente se espera.

### **3.1 Modelo de consistencia secuencial**

Es el modelo que utilizaron los primeros sistemas de MCD, ligeramente más débil que el de consistencia estricta.

Se caracteriza porque el resultado de cualquier ejecución es el mismo que se produciría si las operaciones de todos los procesos fuesen ejecutadas en algún orden secuencial y las operaciones de cada proceso individual apareciesen en esta secuencia en el orden especificado por su programa.

Aunque no garantiza que una lectura devuelve el valor de la última escritura, sí garantiza que todos los procesos observan las referencias a memoria en un mismo orden.

### **3.2 Modelos híbridos de consistencia**

Estos modelos son más débiles que el modelo secuencial. Distinguen dos tipos de acceso a memoria: accesos ordinarios o regulares, y accesos de sincronización.

Algunas motivaciones de estos modelos son que no siempre es necesaria que las escrituras realizadas por un proceso sean observadas por el resto. Se basan en la idea que los accesos concurrentes suelen usar objetos de sincronización para acceder a secciones críticas. Cuando un proceso termina de realizar un determinado número de accesos (al salir de la sección crítica) ha de informar al resto del resultado final de los mismos, sin importar si los resultados intermedios han sido propagados al resto en orden o no.

Una de las técnicas utilizadas en estos métodos es la sincronización mediante variables cerrojo (lock), que no es más que un semáforo binario que regula el acceso a un bloque de datos compartido.

- Consistencia al liberar (release consistency)

Realiza dos operaciones: adquirir y liberar el cerrojo. Al adquirirlo, implica que los datos asociados están actualizados localmente y se pueden usar en exclusión mutua. La liberación provoca la propagación de las modificaciones de los datos.

- Consistencia al acceder (entry consistency)

Tenemos dos tipos de adquisición del cerrojo: exclusiva y compartida. Si la adquisición es exclusiva, la réplica local del dato compartido es la única copia de lectura/escritura. Si es compartida, la réplica local es de sólo lectura, pudiendo coexistir con otras en otros nodos. La liberación no provoca ninguna propagación de modificaciones de los datos, las réplicas son actualizadas siempre al adquirir.

## TEMA 8: INTERACCIÓN Y COOPERACIÓN

### 1. Coordinación Distribuida

Los procesos distribuidos necesitan coordinar sus actividades. Si un grupo de procesos comparten un conjunto de recursos normalmente se requiere la exclusión mutua para prevenir interferencias y asegurar la consistencia cuando se está accediendo al recurso.

En un sistema distribuido los procesos pueden acceder a recursos compartidos comunes pero lo hacen en una sección crítica.

Ni variables compartidas ni las utilidades proporcionadas por un núcleo central sirven, se requiere una solución basada en el paso de mensajes.

#### ○ REQUISITOS DE LA EXCLUSIÓN MUTUA

- EM1: Seguridad, a lo sumo un solo proceso puede estar accediendo a la sección crítica.
- EM2: Vitalidad. Las peticiones de entrar/salir en la sección crítica son concedidas en algún momento, no hay inanición ni abrazo mortal.
- EM3: Ordenación. La entrada en la región crítica debe concederse según la relación sucedió antes. Tenemos 2 nodos A y B que lanzan una petición al servidor. Si el nodo A lanza la petición antes que el nodo B, puede ocurrir que B llegue antes que A. El servidor debe intentar sin embargo atender a A pero esto siempre no se consigue y la ordenación hay que relajarla en determinadas circunstancias.

#### ○ ALGORITMOS

Cualquiera de las estructuras (servidor central, anillo...) son procesos para acceder a secciones críticas.

#### ***Algoritmo basado en servidor central***

##### Explicación

La forma más fácil de conseguir la exclusión mutua es emplear un servidor central que de los permisos para entrar en la sección crítica.

Para entrar en la SC un proceso envía un mensaje a un servidor y espera una respuesta. El servidor central concede permisos en forma de testigo. Si ningún otro proceso tiene el testigo el servidor responderá inmediatamente. Si otro proceso lo tiene el servidor no responde y pone esta petición en la cola de



espera. Al salir de la SC, el proceso devuelve el testigo al servidor.

#### Requisitos de la exclusión mutua

Suponiendo que no hay caídas y no se pierden mensajes:

- Se cumplen E1 y E2
- E3 está asegurada en el orden de llegada de los mensajes al servidor. Es decir que si un proceso A lanza la petición antes que un proceso B pero la petición de B llega antes al servidor central atenderá primero al B.

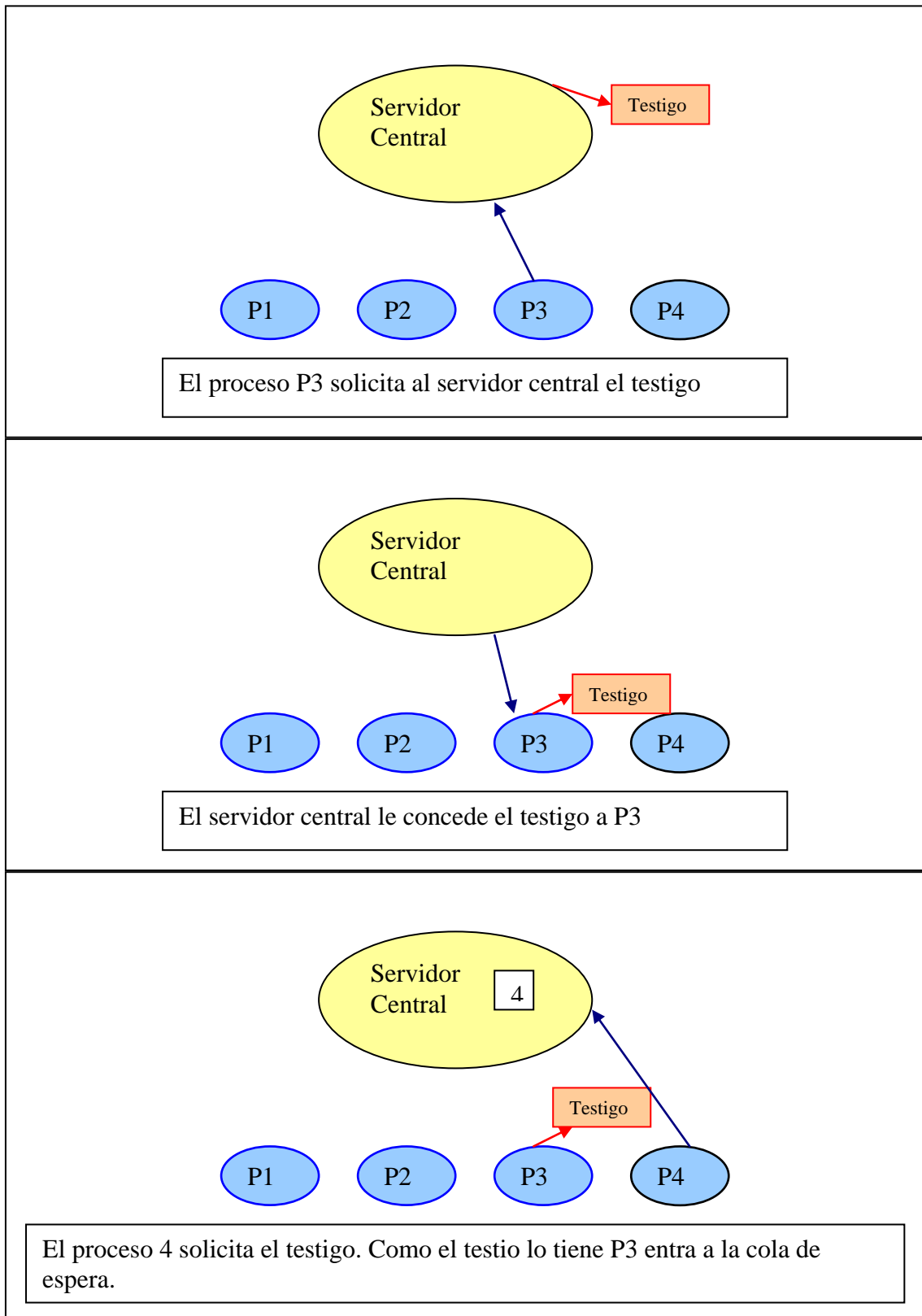
#### Rendimiento

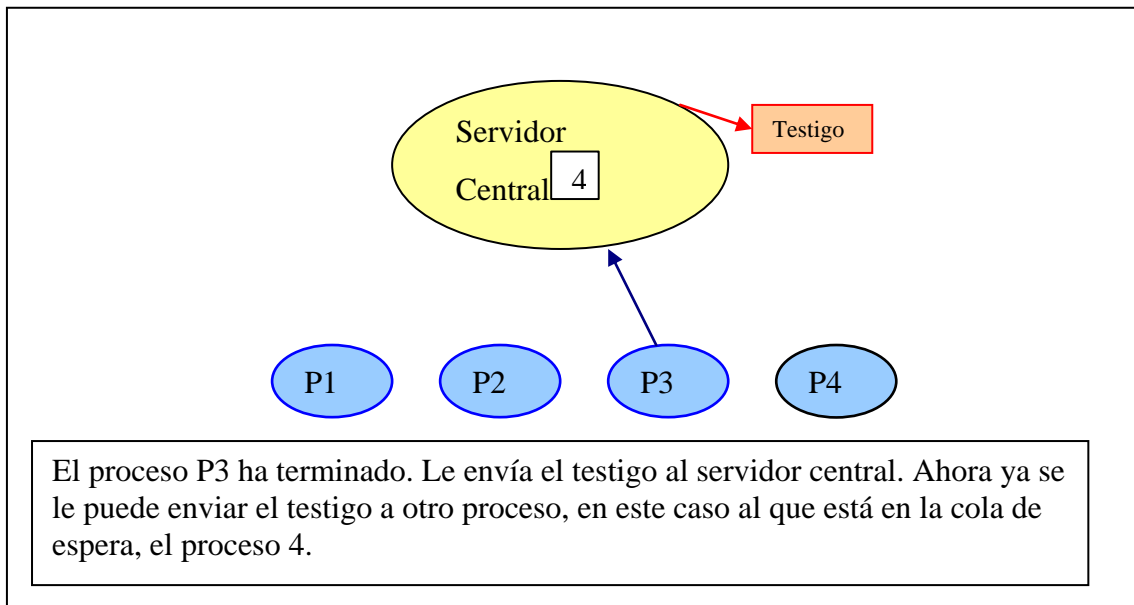
- **2 mensajes** para la entrada en la sección crítica. Incluso cuando no haya ningún proceso ocupándola la entrada implica dos mensajes, uno de petición y otro de concesión. Esto retrasa al proceso que hizo la petición debido al viaje de ida y vuelta.
- **1 mensaje** para salir de la sección crítica, un mensaje de liberación.

#### Problemas

- Todas las solicitudes se envían al servidor, cuello de botella.
- Si sucede una caída o fallo del servidor central habría que utilizar un algoritmo de elección para designar un nuevo servidor central pero probablemente se perderían todas las peticiones y sería necesario reiniciarlas.
- Caída o fallo de un proceso en la sección crítica, lo que implica pérdida del testigo. Para ello se pueden establecer cotas de tiempo o procesos supervisores.

Ejemplo





### **Algoritmo basado en anillo**

#### Explicación

Se crea un anillo lógico ( no físico, cada nodo tiene la dirección de su vecino, pero físicamente no hay un anillo). La exclusión se logra por la obtención de un testigo, mensaje que se pasa de un nodo a otro en una única dirección y que está siempre circulando por el anillo. Cuando un proceso recibe el testigo:

- Si no quiere entrar en la SC, lo envía a su vecino y si quiere entrar en la SC, lo retiene. Al salir de la SC lo vuelve a enviar a su vecino

#### Requisitos de exclusión mutua

- E1 se verifica pues solo hay un testigo. Sólo el proceso que posea el testigo va a poder acceder a la sección crítica, así que sólo un proceso accederá a la sección crítica en un instante determinado.
- E2 se verifica.
- E3 no se puede asegurar. Porque si por ejemplo tenemos un anillo en el que el nodo 3 pide el testigo y cuando se dirige hacia él 2 lo solicita, 2 lo cogería antes. Una posible forma de cumplir E3 sería ordenar por orden de pid y si es distribuido tener en cuenta que también por orden de máquina.

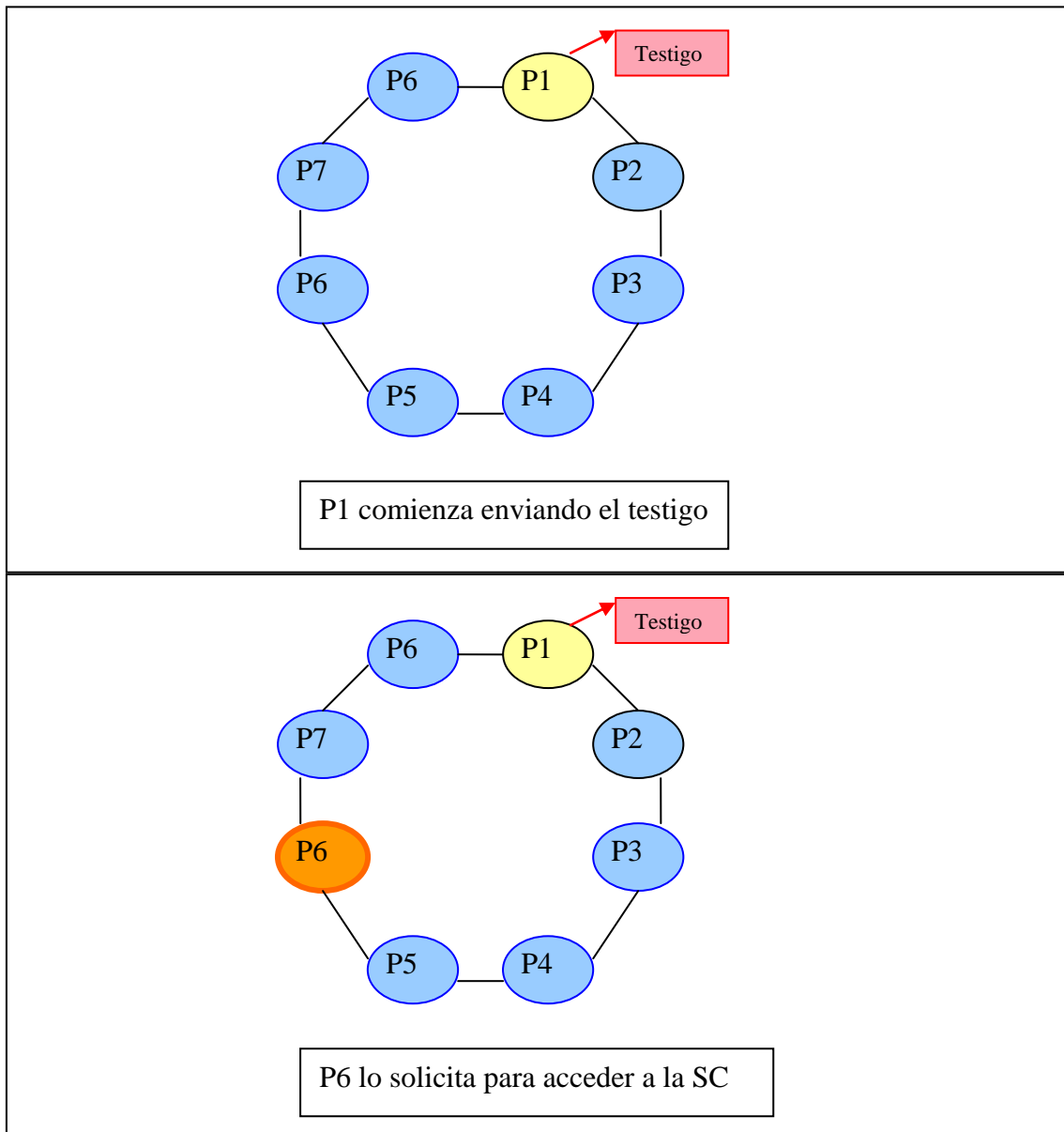
### Rendimiento

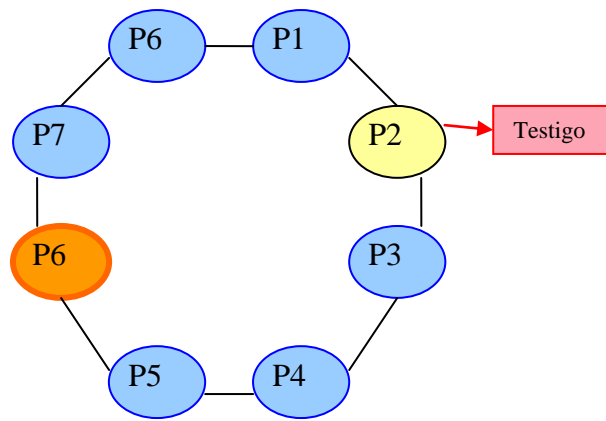
- Los procesos envían mensajes aunque ninguno quiera entrar en la sección crítica, excepto cuando un proceso está dentro de la sección crítica.
- El retraso de un proceso que quiere entrar en la sección crítica está entre 0 mensajes (cuando acaba de recibir el testigo) y N mensajes (cuando acaba de enviar el testigo).
- El retraso de sincronización está entre 1 mensaje y N mensajes.

### Problemas

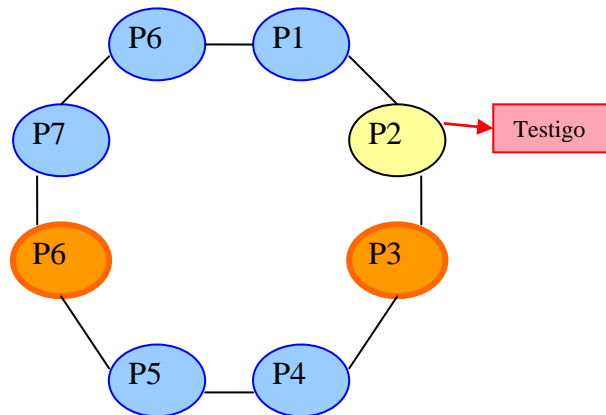
- Se carga la red aun cuando ningún proceso quiera entrar en la SC (sección crítica).
  - El mensaje de solicitudes de acceso recorre el anillo.
- Si un proceso cae necesita reconfiguración.
  - Si además tenía el testigo: elección para regenerar el testigo.
  - Cuidado cuando el sistema es distribuido.
- Asegurarse de que el proceso ha caído.
  - Varios testigos. Si un procesos cae o está incomunicado momentáneamente sacamos otro testigo pero esto tiene un problema cuando se vuelva a reconectar el proceso podríamos tener dos testigos por el anillo.
- Desconexión o ruptura de la red, lo que implica una pérdida del testigo.

Ejemplo

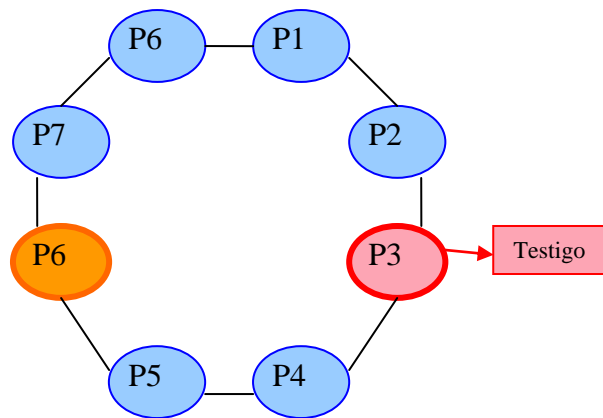




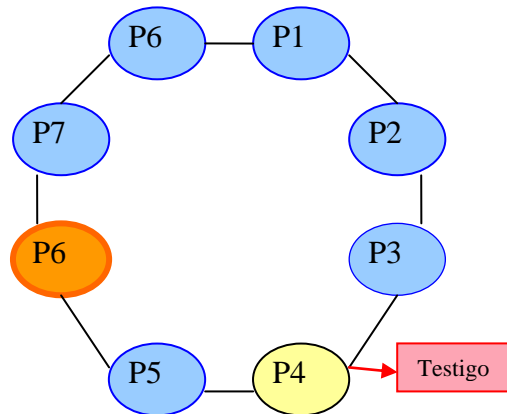
El testigo pasa a P2



Ahora P3 solicita el testigo. Como llega antes, se le concederá antes a P6. No cumple E3 de los requisitos de exclusión mutua.



P3 accede a la sección crítica cuando tiene el testigo y cuando finaliza se lo pasa a P4.



Continuaría el mismo proceso para llegar a P6.

### **Algoritmo basado en relojes lógicos**

Para evitar estos puntos donde los algoritmos anteriores fallan se han creado los algoritmos distribuidos de acceso a la sección crítica. Uno de ellos es el de Ricart y Agrawala. Este requiere la existencia de un orden total entre cualquiera de los eventos ocurridos en el sistema. Para ello se puede utilizar el algoritmo de Lamport.

Para los algoritmos basados en relojes lógicos se tienen que cumplir 2 premisas: cada proceso debe conocer las direcciones de los demás y cada proceso posee un reloj lógico.



## RICART Y AGRAWALA

### Explicación

Todos los procesos piden y conceden acceso a la SC. Si un proceso quiere entrar a la SC les pregunta a todos los demás si puede entrar. Cuando todos los demás contestan entonces entra.

El acceso se obtiene a través de un mensaje que almacena el nombre del proceso y la hora actual, tupla  $\langle Ti, Pi \rangle$ . Cada proceso guarda el estado en relación a la SC: liberada, buscada o tomada.

El algoritmo funciona como sigue:

- Si un proceso no está en la SC y no desea entrar en ella entonces contesta inmediatamente al emisor.
- Si un proceso está en la SC entonces no responde y encola la petición.
- Si un proceso no está en la SC pero ha solicitado su acceso comprueba si lo hizo antes. Para ello compara la marca de tiempo en el mensaje obtenido con la marca de tiempo del mensaje que él envió. Si el otro proceso solicitó antes entonces le contesta al emisor sino el receptor encola la petición y no responde nada.

Cuando un proceso sale de la SC avisa a todos los demás y responde a cualquiera de las peticiones de su cola.

### Requisitos de exclusión mutua

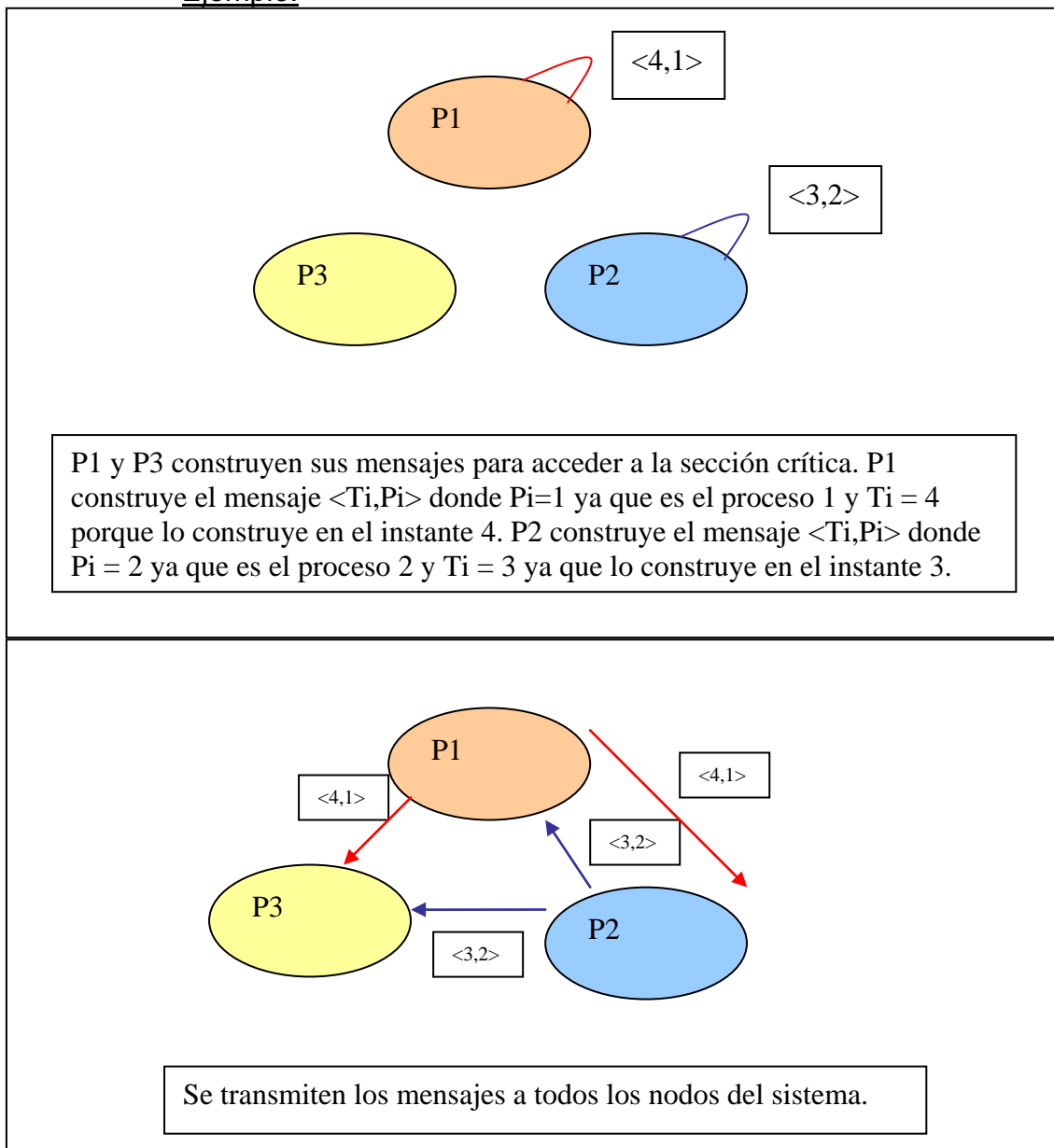
- E1 se cumple, solo un proceso puede entrar en un momento a la SC.
- E2 se cumple no hay inanición ni interbloqueos, en el caso de que el sistema no falle.
- E3 se cumple por el uso de los relojes lógicos de Lamport.

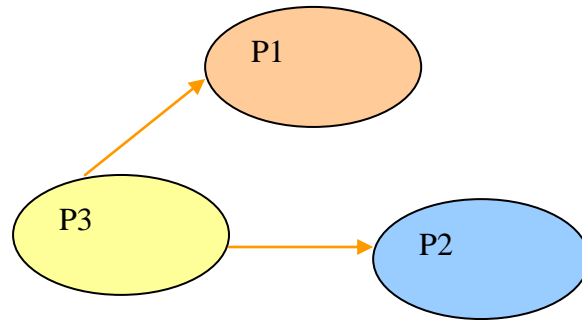
### Rendimiento

- Sin soporte multicast  $2(n-1)$  mensajes, donde  $n$  es el número total de procesos.  $N-1$  mensajes para solicitar la entrada en la región crítica y  $n-1$  mensajes de otorgamiento de que puede entrar en la sección crítica.
- Con soporte multicast  $n$  mensajes para salir de la región crítica.

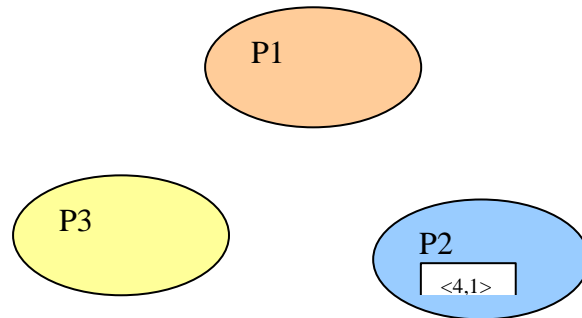
Problemas

- Algoritmo más costoso que el del servidor central
- Pese a ser algoritmos distribuidos, el fallo de cualquier proceso bloquea el sistema ya que no podrá responder a las peticiones y se interpretará que está en la SC y se bloquearán los siguientes intentos de los demás procesos de entrar a la SC. La solución está en que el receptor siempre envíe una respuesta otorgando el permiso o denegándolo. El emisor espera esta respuesta y si no la recibe puede volver a intentar o concluir que el destino está muerto.
- Los procesos implicados reciben y procesan cada solicitud. Igual o peor congestión que el servidor central.

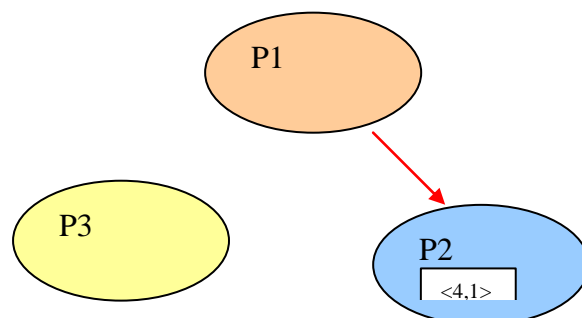
Ejemplo:



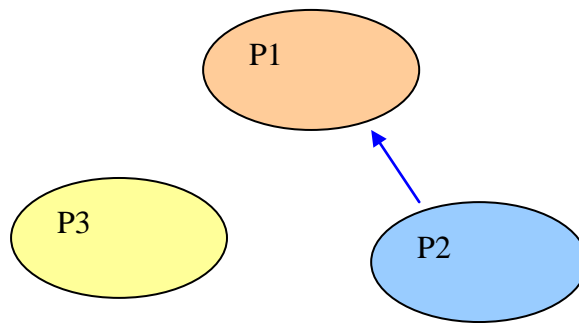
P3 no está en la SC ni pretende acceder a ella por ello contesta inmediatamente a los procesos P1 y P2.



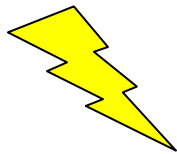
P2 ha recibido el mensaje  $\langle 4, 1 \rangle$ . Lo compara con el suyo propio,  $\langle 3, 2 \rangle$ . Como  $3 < 4$ , él pidió antes el acceso no contesta.



P1 ha recibido el mensaje  $\langle 3, 2 \rangle$ . Lo compara con el suyo propio  $\langle 4, 1 \rangle$ . Como  $4 > 3$  contesta a P2.



P2 entra en la SC. Cuando sale de ella en su cola de mensajes tiene el mensaje  $\langle 4, 1 \rangle$  así que envía el mensaje OK al proceso P1 y vacía la cola.



¡¡¡¡EJERCICIO EN LAS TRANSPARENCIAS!!!!  
Presentación de Power Point -> Transparencia 18

### ○ CONCLUSIONES

Ninguno puede tratar el problema de la caída de un computador o proceso. El algoritmo de servidor central es el que tiene menor número de mensajes, pero supone un cuello de botella. Es preferible que el servidor que gestiona el recurso implemente también la exclusión mutua.

Algoritmo	Mensajes para entrar y salir de la SC	Mensajes antes de poder entrar en la SC	Problemas
Directorio Centralizado	3	2	Fallo del coordinador
Anillo	1 a infinito	0 a n-1	Pérdida del testigo
Distribuido(Reloes)	$2(n-1)$	$2(n-1)$	Fallo de cualquier proceso

## 2. Algoritmo de elección

Muchos de los algoritmos distribuidos necesitan que un proceso actúe como coordinador, iniciador, secuenciador o que desempeñe un papel especial. En general, los algoritmos de elección intentan localizar al proceso con el identificador más alto, diferenciándose en la forma de hacerlo.

Supondremos que cada proceso conoce los identificadores del resto de procesos, pero desconoce si están activos o inactivos.

El objetivo de todos estos algoritmos es garantizar que un único proceso llega a ser coordinador, aunque varios procesos hayan lanzado el algoritmo concurrentemente.

En la elección se debe asegurar:

E1(Seguridad) → Todo proceso participante tiene elegido como líder el proceso vivo con mayor identificador o tiene el valor indefinido

E2 (Vivacidad) → Todo proceso tiene elegido un líder o ha caído.

- ANILLO

Elige al proceso con identificador más alto, supone que entre los procesos hay un anillo lógico (cada proceso sólo sabe comunicarse con su vecino) y considera que no se pueden producir fallos durante la elección. Tanenbaum hizo una variante en 1992 en donde los procesos sí podían caer.

- ALGORITMOS

-- Todos los procesos comienzan siendo no candidatos

-- Cualquiera de ellos se percata que el coordinador ha caído, se marca como candidato y lanza un mensaje de elección con su identificador.

Recepción de mensaje de elección

- Si el identificador del mensaje es **mayor** que el del proceso → transmite el mensaje a sus vecinos

- Es candidato → no hará nada, porque ya ha mandado su mensaje de elección.

- Si el identificador es **menor**

- Es no-candidato → sustituye el identificador por el suyo, se marca como candidato y lo reenvía al vecino

- Si el identificador es el **suyo** → se marca como no-candidato y manda un mensaje de elegido con su identificador.

Recepción de mensaje de elegido

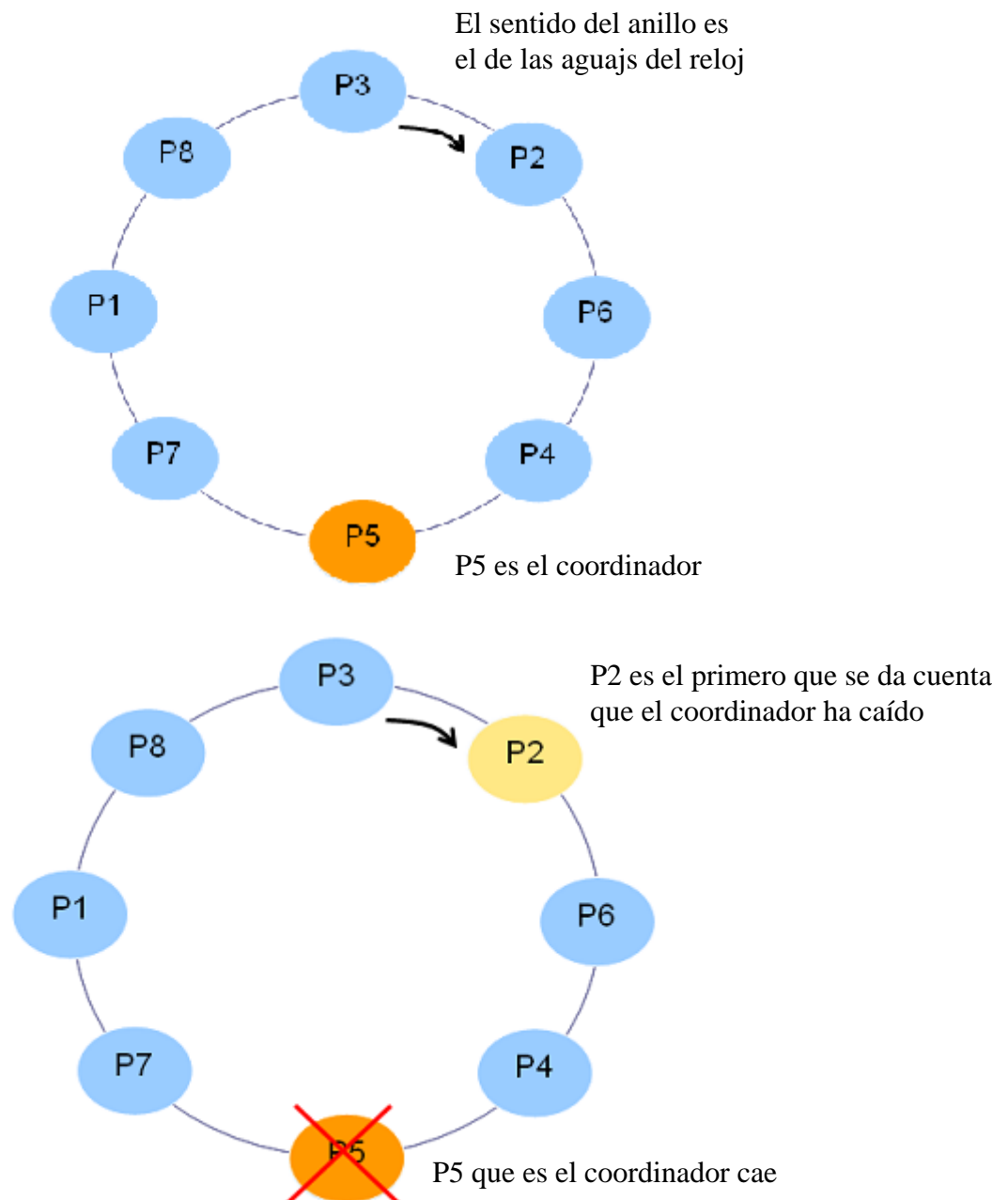
- Si el identificador no es el suyo → se marca como no-candidato y lo manda a su vecino.

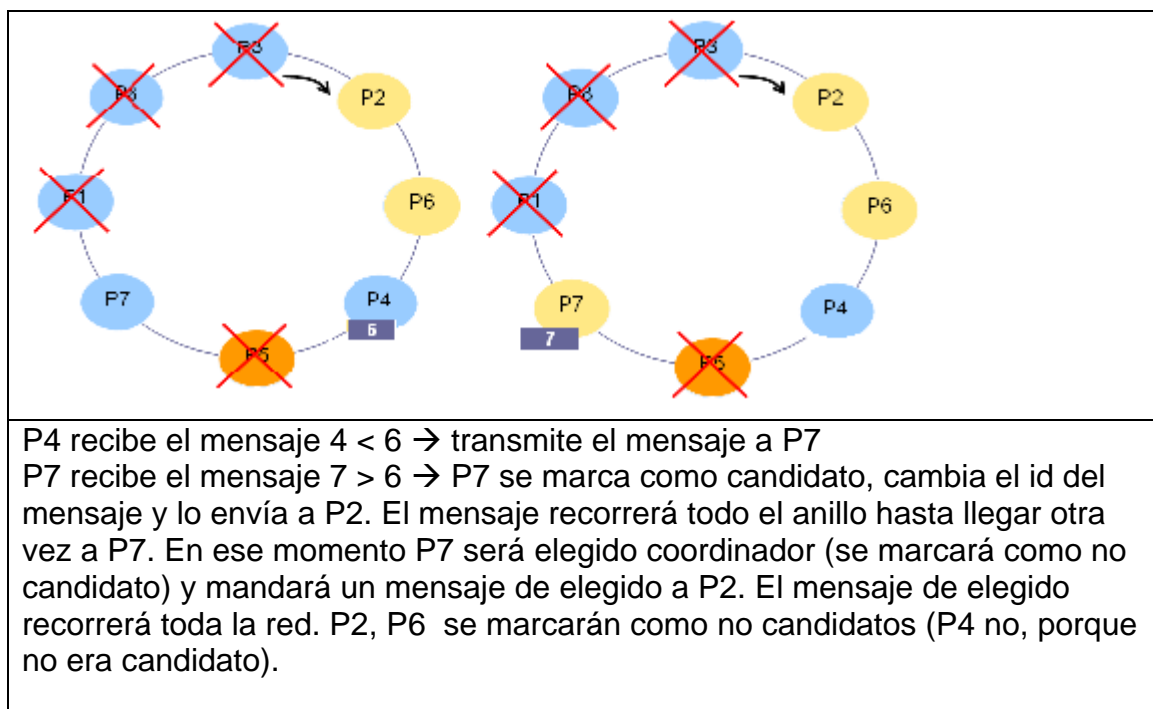
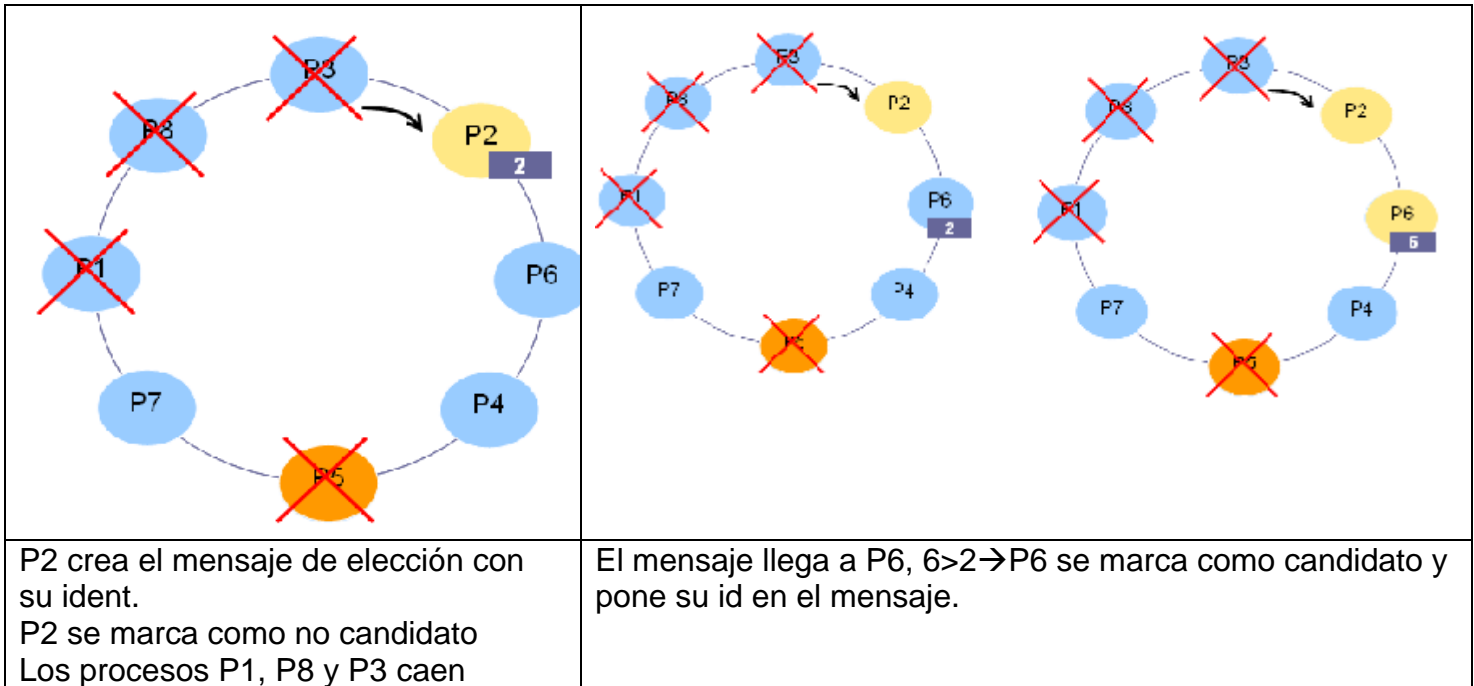
El número de mensajes necesarios para que un proceso se erija como coordinador serán:

Peor caso: Se da cuenta que el coordinador ha caído el nodo siguiente al que conseguirá ser coordinador  $\rightarrow 3n-1$  mensajes

Mejor caso: Se da cuenta de que el coordinador ha caído el futuro coordinador.

○ EJEMPLO DE EJCUCIÓN DEL ALGORITMO





- BULLY

En este algoritmo los procesos pueden caer durante la elección. Se seleccionará el proceso con mayor identificador.

- Requisitos:

- todos los miembros del grupo deben conocer las identidades y direcciones de los demás miembros
- se supone comunicación fiable

- Hay 3 tipos de mensajes:



- mensaje de elección: para anunciar una elección
- mensaje de respuesta a un mensaje de elección
- mensaje de coordinador: anuncia identidad de nuevo coordinador

○ Número de mensajes para elegir coordinador:

- caso mejor: se da cuenta el segundo más alto  $\rightarrow (n-2)$  mensajes
- caso peor: se da cuenta el más bajo  $\rightarrow O(n^2)$  mensajes

El algoritmo comienza cuando alguno de los procesos se percató de que el coordinador ha caído. En ese momento envía un mensaje de elección a todos los procesos con identificador mayor al suyo y se dispondrá a esperar respuesta.

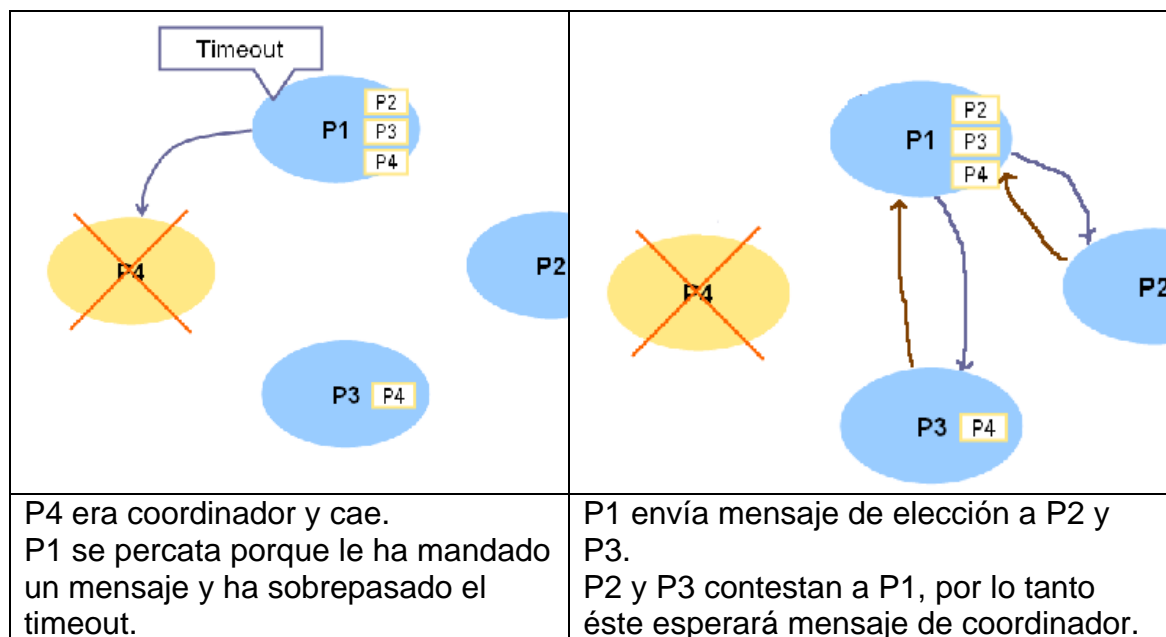
- Si no recibe respuesta  $\rightarrow$  se nombra coordinador y envía un mensaje de coordinador a todos los procesos con id más bajo
- Si recibe respuesta  $\rightarrow$  se limitará a esperar el mensaje del coordinador.

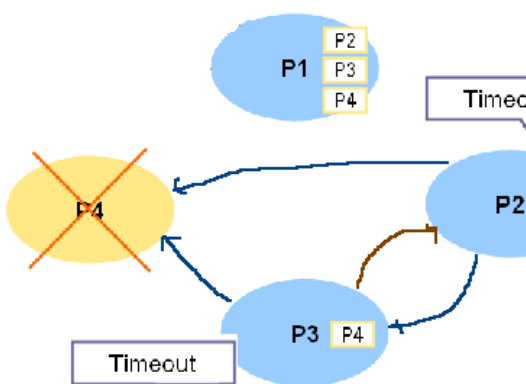
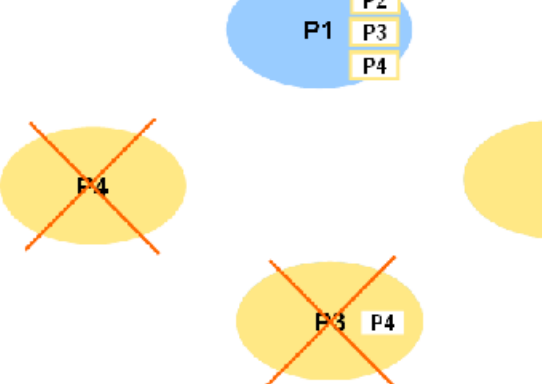
Si un proceso recibe un mensaje de elección  $\rightarrow$  responderá a él y enviará un mensaje de elección (recibir un mensaje de elección significa que hay procesos con ids más bajos que el tuyo, por lo tanto tú podrías ser el proceso con id más alto y por lo tanto mandas el mensaje de elección)

Si un proceso recibe un mensaje de coordinador  $\rightarrow$  guardará el id del mensaje y tratará a éste como coordinador.

Si un proceso se reinicia  $\rightarrow$  si tiene el id más alto se erigirá como nuevo coordinador, si no, lanzará un mensaje de elección.

○ EJEMPLO DE EJECUCIÓN



	
<p>P2 envía mensaje de elección a P3 y P4. P3 lo envía a P4. P3 responde a P2 y al no recibir ninguna respuesta se proclama coordinador.</p>	<p>P3 cae se da cuenta P1 y se hace el mismo proceso, resultando coordinador P2. Si P3 se reestablece se erigirá coordinador y mandará un mensaje de coordinador a P1 y P2</p>