Sistemas Distribuidos. Temario.

1. Introducción
2. Comunicación
3. Procesos
4. Nombrado
5. Sincronización
6. Consistencia y replicación
7. Tolerancia a fallos

Tema 2.- Comunicación

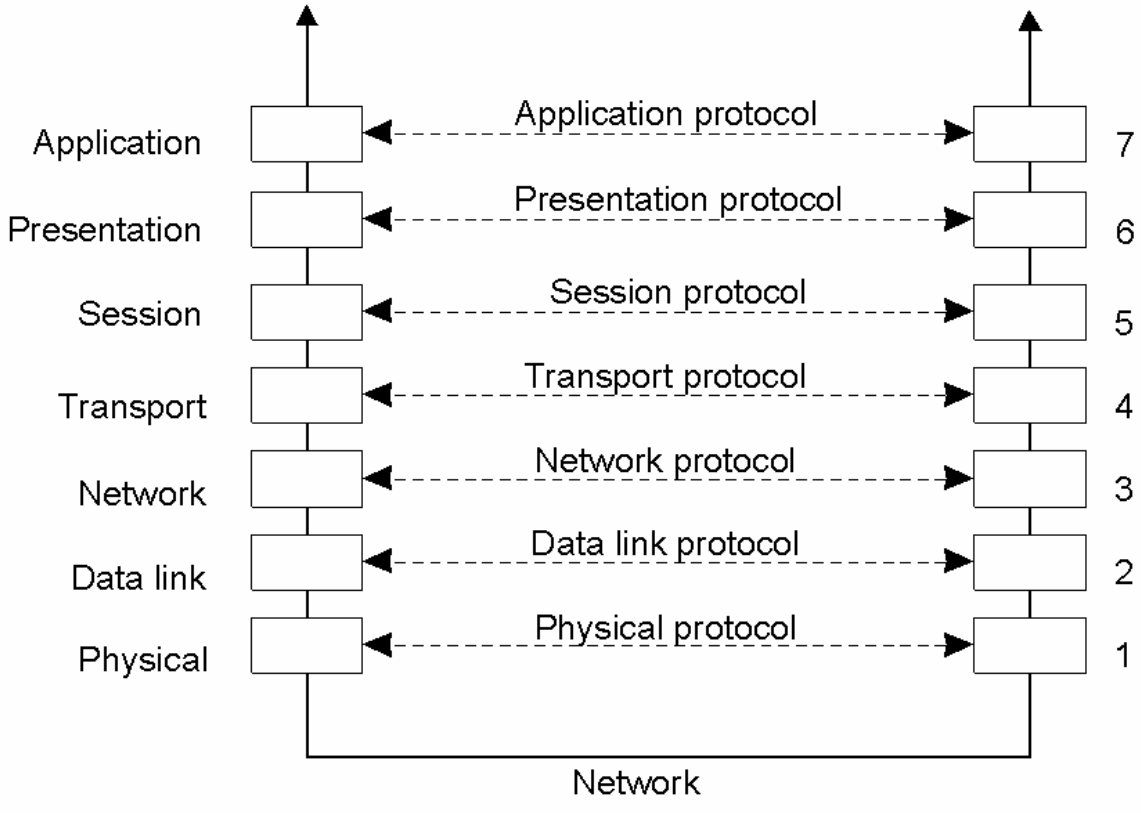
1. Protocolos basados en niveles
2. Llamada a procedimiento remoto (RPC)
3. Invocación a objeto remoto (ROI)
4. Comunicación basada en mensajes
5. Comunicación basada en flujos

Bibliografía: Capítulo 2 de Tanenbaum

1.- Protocolos basados en niveles

1. El modelo ISO/OSI (Open Systems Interconnection Reference Model)
2. Los protocolos de bajo nivel
3. El nivel de transporte
4. Los protocolos de alto nivel
5. Protocolos de middleware

1.1.- El modelo ISO/OSI

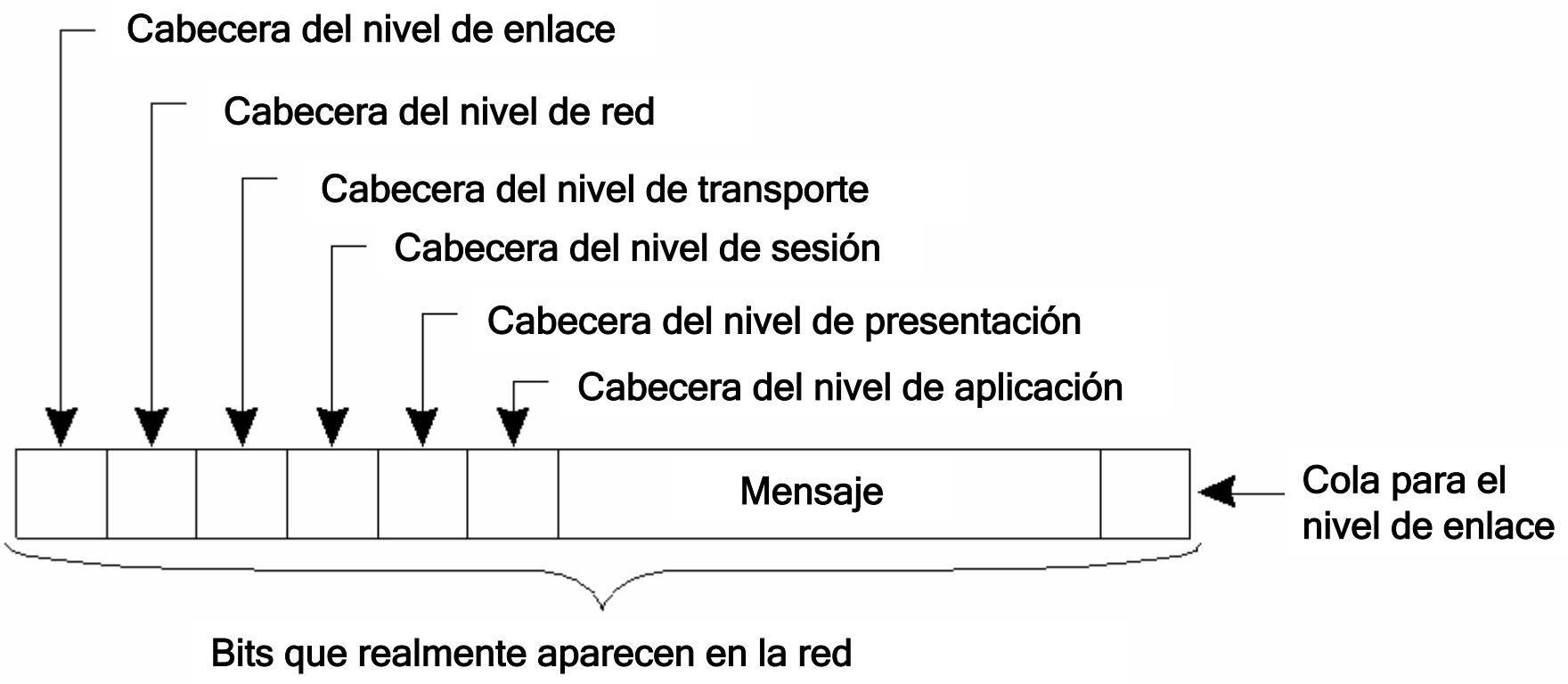


1.1.- El modelo ISO/OSI

- Se trata de un modelo de referencia ampliamente usado (los protocolos OSI que se implementaron se usaron muy poco).
- El modelo plantea la comunicación en 7 niveles.
- Cada nivel trata un aspecto específico de la comunicación.
- Cada nivel proporciona una interfaz al nivel superior. La interfaz no es más que un conjunto de operaciones que pueden utilizarse.
- A la colección de protocolos implementados que se utilicen se los conoce como la **pila de protocolos**.

1.1.- El modelo ISO/OSI

- Cada nivel incluye su propia cabecera / cola en el mensaje a enviar, con los datos necesarios para implementar su protocolo

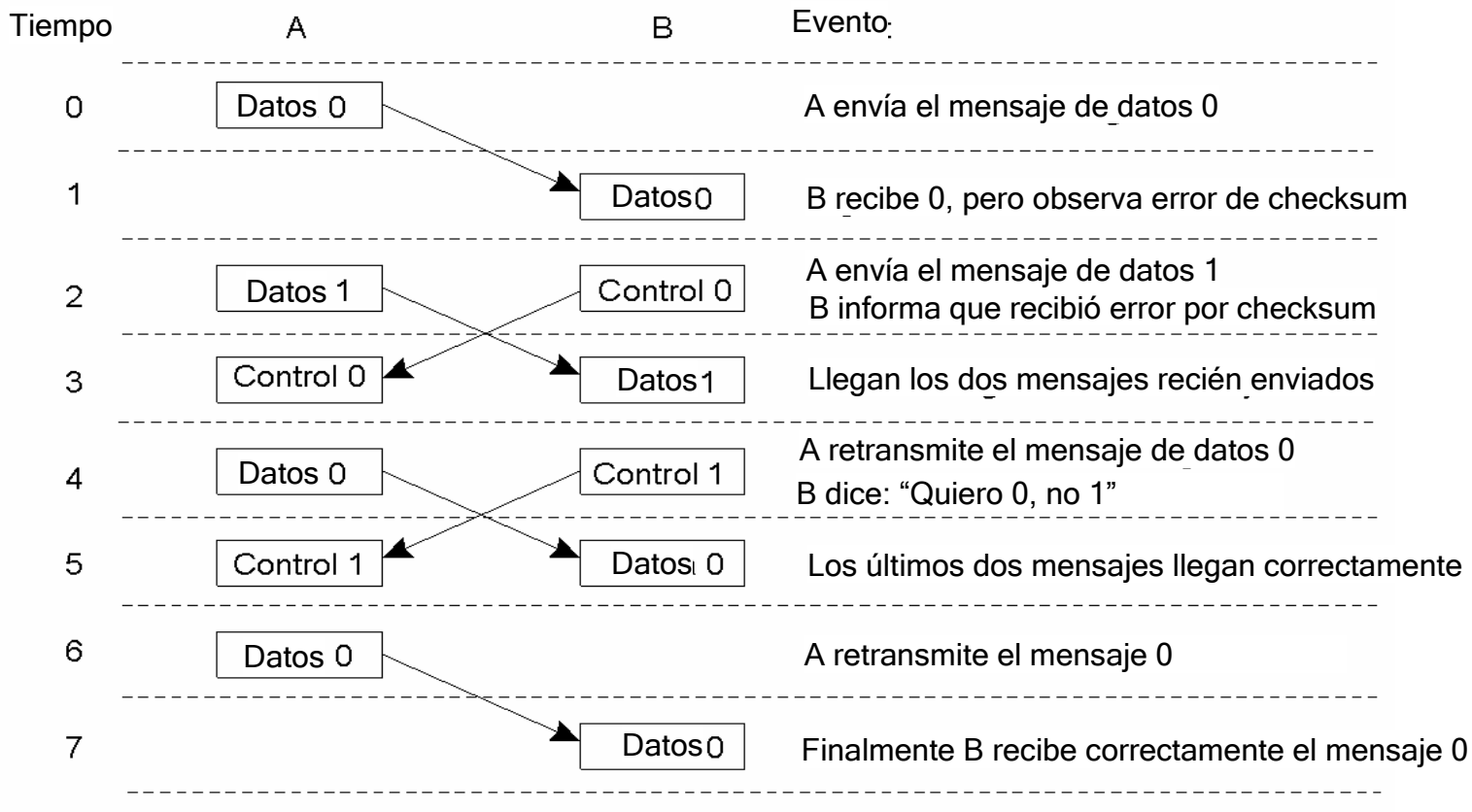


1.2.- Los protocolos de bajo nivel

- **Nivel físico**: Este nivel se dedica a transmitir los 0's y los 1's.
  - Cuántos voltios se emplearán para codificar los 0's y cuántos para los 1's
  - Cuántos bits/segundo
  - Comunicación simplex/duplex
  - Tamaño, forma y características de los conectores
  - Ejemplo: RS-232
- **Nivel de enlace**: Asegurar transmisión libre de errores.
  - Agrupar bits en grupos de bits (**tramas**)
  - Aplicar códigos de redundancia a las tramas para detectar errores.
  - En caso de errores, enviar mensajes de control para pedir la retransmisión.

1.2.- Los protocolos de bajo nivel

- Ejemplo de protocolo de enlace de datos



1.2.- Los protocolos de bajo nivel

- **Nivel de red**: Encaminar los mensajes de una máquina a otra.
  - A este nivel a los mensajes se les llama **paquetes**.
  - Ejemplo: IP, parte de la pila de protocolos de Internet.

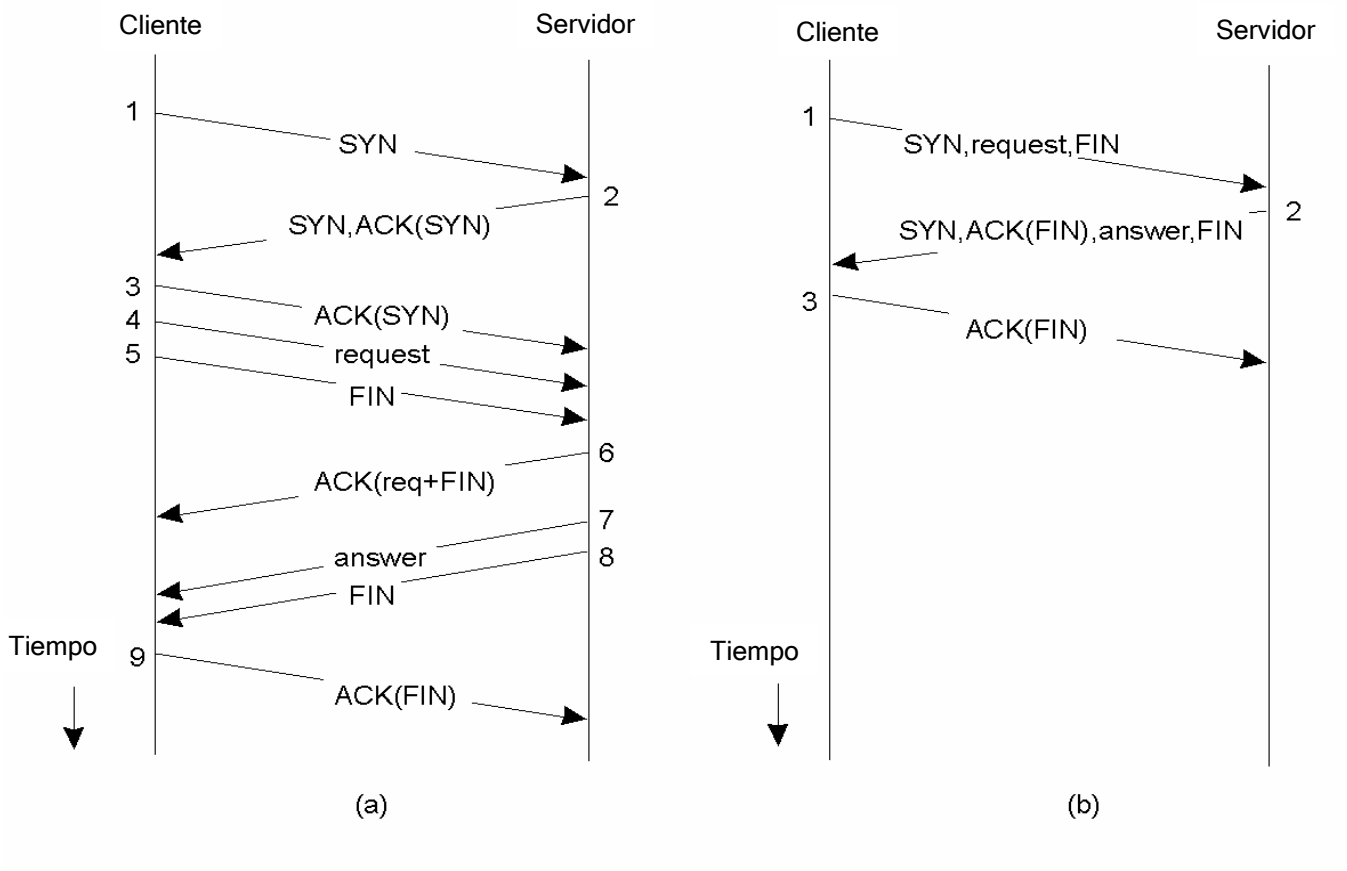
1.3.- Protocolos de transporte

Objetivo: Asegurar la fiabilidad de las comunicaciones

- Los mensajes enviados desde las aplicaciones (o desde los niveles superiores del modelo OSI), son divididos en paquetes, que serán reensamblados en el destino.
- Los paquetes que no lleguen a su destino, serán retransmitidos.
- Ejemplos: TCP, UDP
  - TCP, sobre IP: ofrece mensajes fiables con conexión.
  - UPD, sobre IP: ofrece mensajes no fiables sin conexión. (sin retransmisiones)
- Cuestión: ¿Cómo funciona TCP para cliente/servidor?

1.3.- Protocolos de transporte

a) Mensajes necesarios para hacer una petición cliente/servidor mediante TCP
b) Interacción cliente/servidor mediante TTCP (Transactional TCP)



1.3.- Protocolos de transporte

- TTCP es un protocolo de transporte diseñado para optimizar las interacciones cliente/servidor en Internet.
- Es un protocolo reciente, compatible con TCP, pero poco utilizado.

1.4.- Protocolos de alto nivel

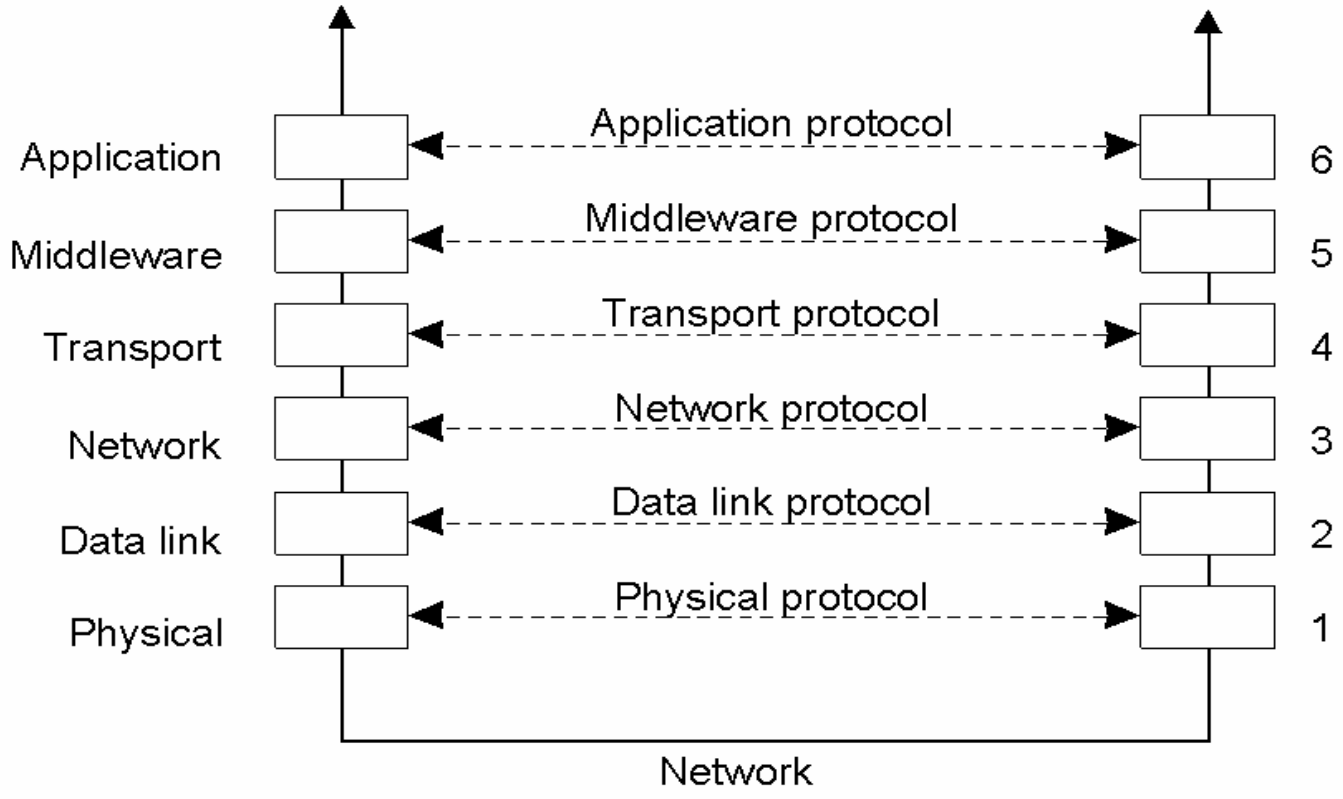
- Por encima del nivel de transporte, OSI define 3 niveles más. En la práctica, los tres se engloban en uno sólo: el nivel de aplicación.
  - **Nivel de sesión**: Proporciona control sobre la conversación y proporciona facilidades de sincronización. Por ejemplo para realizar 'checkpoints' y de esta forma tolerar fallos.
  - **Nivel de presentación**: Dedicado a tratar el significado de la información que se transmite. Por ejemplo registros en lugar de bits.
  - **Nivel de aplicación**: Los protocolos específicos necesarios para dotar de funcionalidad a determinado sistema: FTP, HTTP, MAIL, TELNET, etc.

1.5.- Protocolos de middleware

- Middleware es software que según en modelo OSI reside en el nivel de aplicación.
- Sin embargo, contiene muchos protocolos de propósito general, que podrán ser empleados por aplicaciones.
- Hay multitud de protocolos que soportan gran variedad de servicios. Por ejemplo:
  - Protocolos de autenticación
  - Protocolos de compromiso atómico
  - Protocolos de gestión de cerrojos distribuidos
  - Protocolos para proporcionar mecanismos de comunicación de alto nivel: RPC, ROI, etc.

1.5.- Protocolos de middleware

El modelo ISO/OSI, *reescrito* contemplando el nivel de middleware.



Tema 2.- Comunicación

1. Protocolos basados en niveles
2. Llamada a procedimiento remoto (RPC)
3. Invocación a objeto remoto (ROI)
4. Comunicación basada en mensajes
5. Comunicación basada en flujos

Bibliografía: Capítulo 2 de Tanenbaum

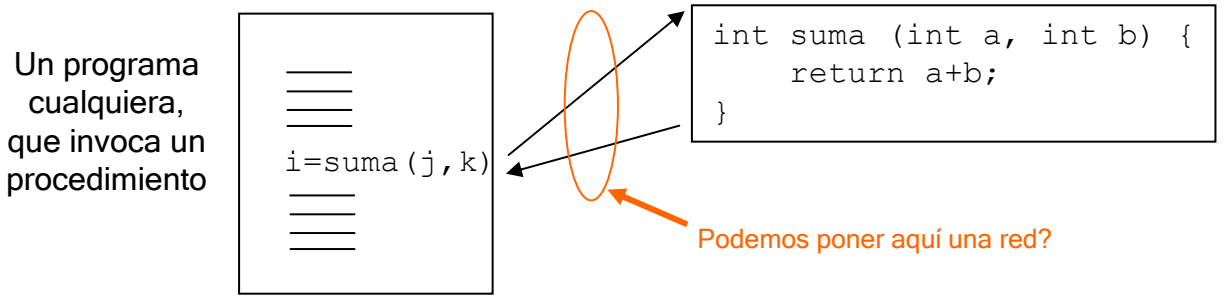
2.- Llamada a procedimiento remoto

1. Funcionamiento de RPC básico
2. Paso de parámetros
3. Variaciones

2.1.- Funcionamiento de RPC básico

- **Observaciones**:
  - Los desarrolladores de aplicaciones estamos acostumbrados a los procedimientos.
  - Los procedimientos bien realizados funcionan como *cajas negras*: dadas unas entradas, proporcionan unas salidas.
  - Resulta natural distribuir la ejecución de procedimientos en diferentes máquinas.
- **Conclusión**: la interacción entre en invocador y el invocado puede ocultarse mediante un mecanismo de llamada a procedimiento

2.1.- Funcionamiento de RPC básico



- La invocación local a procedimientos consiste en seguir cierta convención (protocolo) de paso de argumentos y de cambio del contador de programa.:
  - Se colocan en la pila los argumentos y el contador de programa
  - Se salta al procedimiento.
  - El procedimiento obtiene los argumentos de la pila y se ejecuta.
  - Los resultados del procedimiento se colocan en la pila.
  - Al finalizar el procedimiento se retorna al punto de origen, obteniendo el contador de programa original de la pila.

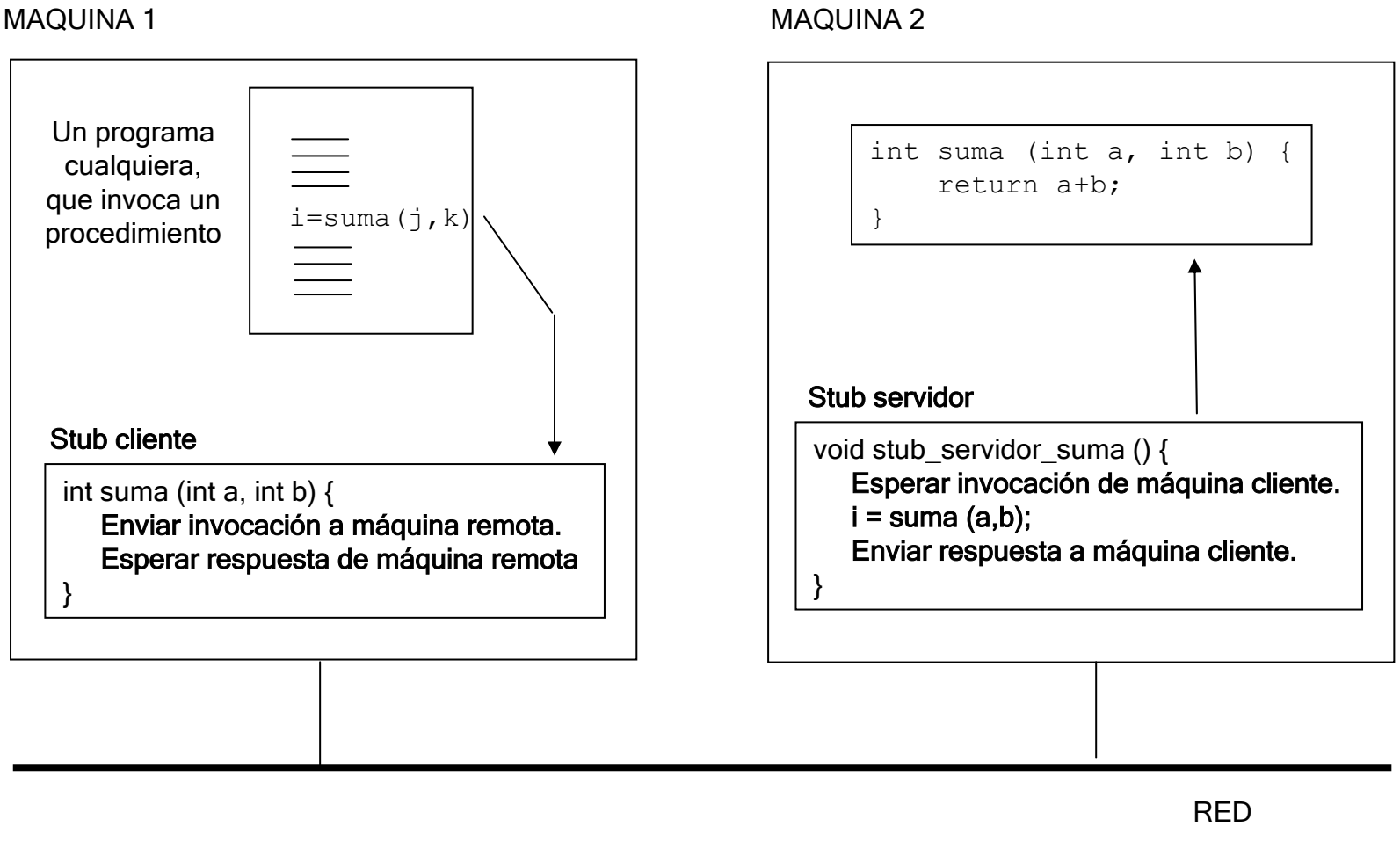
  Todo este trabajo lo hace el compilador de forma **transparente** al programador!!

- Si colocamos los procedimientos en máquinas diferentes a la máquina que invoca, tendremos llamada a procedimiento remoto.

2.1.- Funcionamiento de RPC básico

- Con las llamadas a procedimiento remoto hay que tener en cuenta que queremos la misma **transparencia** de ubicación que tiene el programador que utiliza procedimientos convencionales:
  - Queremos ocultar al cliente el hecho de que el procedimiento está ubicado en otra máquina.
  - Queremos que el programador del cliente, simplemente invoque `i=suma(j,k)` y por "arte de magia", la invocación llegue a la máquina remota
  - Esta "magia" se consigue en RPC mediante los **stubs**.

2.1.- Funcionamiento de RPC básico



2.1.- Funcionamiento de RPC básico

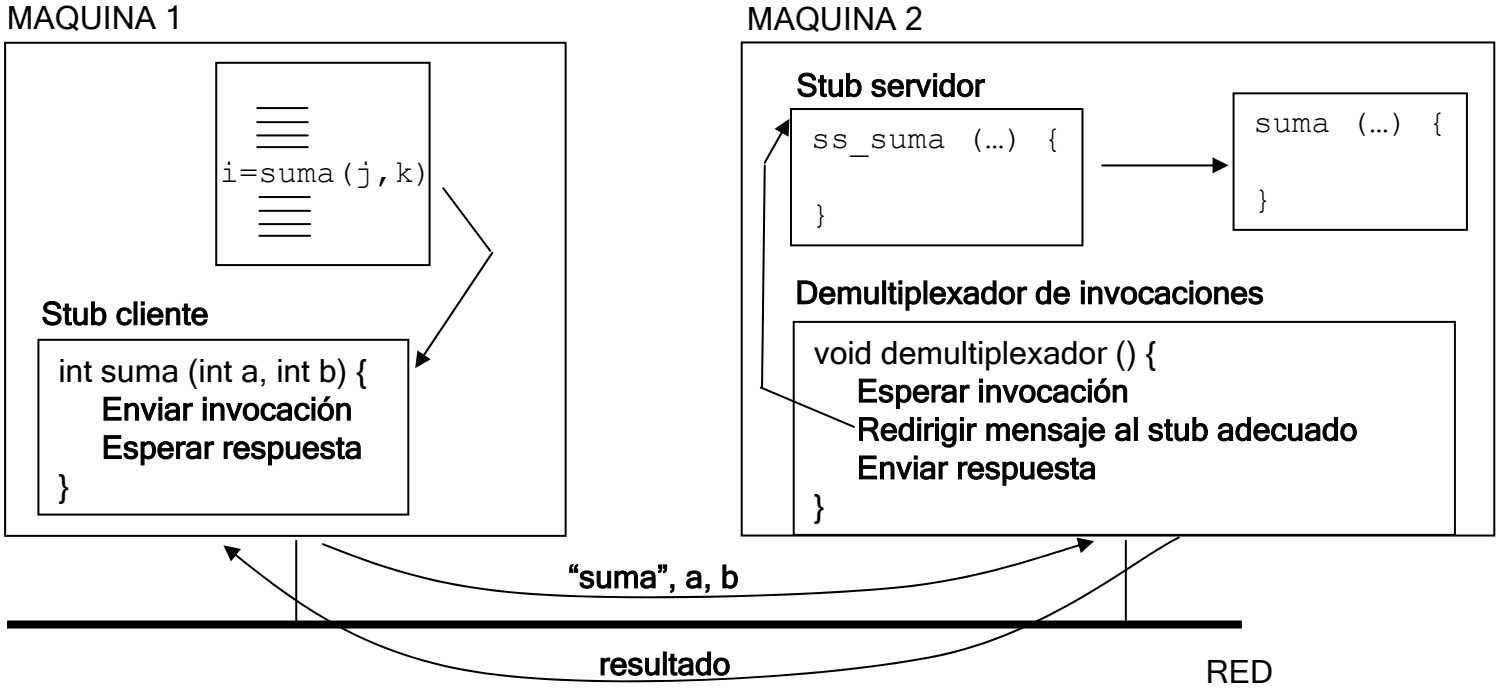
- Los stubs los debe generar el compilador.
- Tanto el programador del programa cliente, como el programador del procedimiento remoto ignoran la existencia de los stubs: **transparencia**.
- Nótese que el procedimiento servidor podría invocar otro procedimiento ubicado en una tercera máquina (incluso en la máquina "cliente"):
  - **Más que de máquina (proceso) cliente o máquina (proceso) servidora, hay que hablar de cliente y servidor para un procedimiento en particular, puesto que el rol cliente/servidor debe entenderse por procedimiento**

2.2.- Paso de parámetros

- El stub cliente debe:
  - Empaquetar los argumentos en un mensaje.
  - Enviar el mensaje de invocación al servidor.
  - Esperar el mensaje de respuesta.
  - Desempaquetar los resultados (argumentos de salida) del mensaje de respuesta
  - Devolver los resultados al código que invocó al stub.
- El stub servidor debe:
  - Esperar mensaje de invocación
  - Desempaquetar los argumentos de la invocación.
  - Realizar la invocación al procedimiento local y esperar a que termine.
  - Empaquetar los argumentos de salida en el mensaje de respuesta.
  - Enviar la respuesta al cliente.

2.2.- Paso de parámetros

- Además de los parámetros, hay que pasar “algo” que identifique al procedimiento que se desea invocar: por ejemplo el nombre del procedimiento.
- Esto es necesario para permitir que una máquina tenga más de un procedimiento invocable.

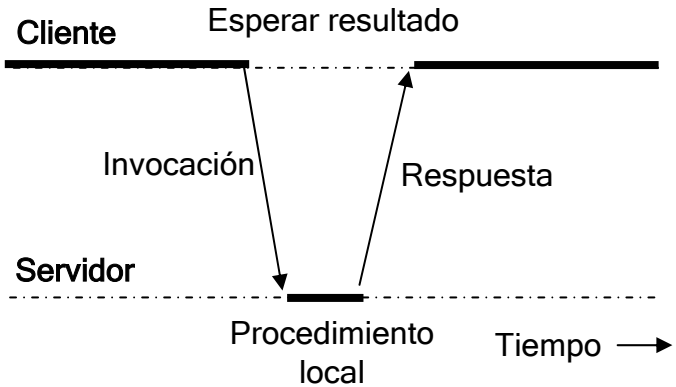


2.2.- Paso de parámetros

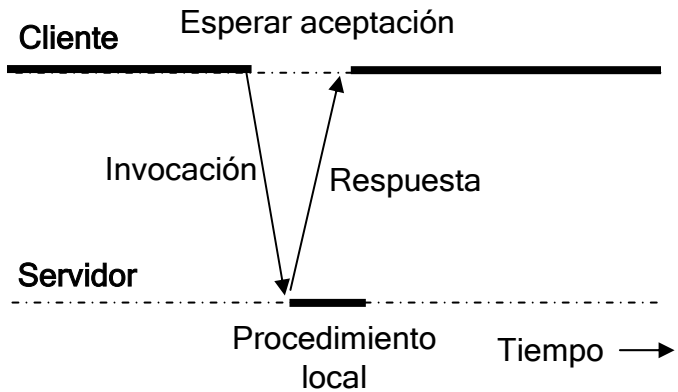
- Al empaquetado de argumentos se le llama **marshaling**.
- Al desempaquetado de argumentos se le llama **unmarshaling**.
- Paso de **argumentos por valor** (argumentos de entrada): el programa cliente copia los argumentos al stub, el stub lo envía por la red, el stub servidor copia los argumentos al procedimiento remoto. Se procede de igual forma con los argumentos de salida.
- Paso de **argumentos por referencia** (resultados, argumentos de salida o argumentos de entrada/salida):
  - ¿Cómo se pasa un puntero? En general no se puede.
  - Para pasar por referencia, lo habitual suele ser pasar por valor, pero el stub cliente debe sobrescribir los datos que reciba de la red encima de los argumentos que recibió del programa cliente.
- ¿Cómo se sabe si los argumentos son de entrada, de salida, o de entrada/salida? depende del **lenguaje!**
  - Por este motivo se creó IDL (Interface Definition Language): En él se especifican argumentos **in**, **out** e **inout**. Con esta interfaz, el compilador crea los stubs realizado correctamente el paso de argumentos.

2.3.- Variaciones

- Hay otras variaciones de RPC:
  - RPC como mecanismo de intercomunicación entre procesos de una misma máquina: Ejemplo doors en Spring
  - RPC asíncrona: optimización de RPC que permite que el cliente no tenga que esperar a que termine el procedimiento, pero **sí** debe esperar un mensaje de respuesta.
  - RPC síncrona retardada: Igual que RPC asíncrono, pero el servidor invocará al cliente cuando termine el procedimiento para proporcionarle los resultados.



RPC Convencional



RPC asíncrona

Tema 2.- Comunicación

1. Protocolos basados en niveles
2. Llamada a procedimiento remoto (RPC)
3. Invocación a objeto remoto (ROI)
4. Comunicación basada en mensajes
5. Comunicación basada en flujos

Bibliografía: Capítulo 2 de Tanenbaum

3.- Invocación a objeto remoto

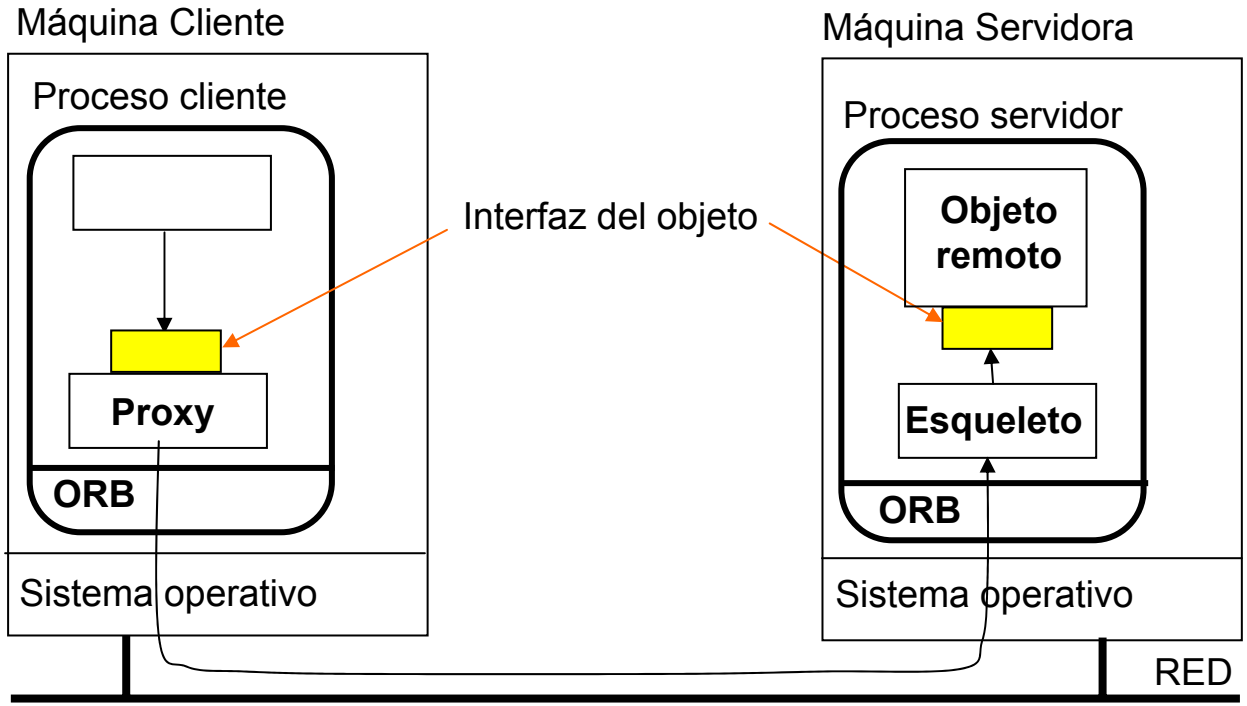
1. Objetos distribuidos
2. Referencias a objeto
3. Optimizaciones

3.1.- Objetos distribuidos

- Sistemas OO en general
  - Los objetos encapsulan datos (**estado**) y operaciones sobre los datos (**métodos**).
  - La única forma de acceder a los objetos es mediante llamadas a sus métodos.
  - Los objetos pueden tener múltiples facetas, es decir, pueden implementar múltiples interfaces.
  - Esta **distinción entre objeto e interfaces** es clave!
  - Una interfaz no es más que la lista de los métodos, detallando sus argumentos, que satisface cierta clase de objetos.
- Objetos distribuidos
  - Para invocar a un objeto remoto con transparencia de ubicación (de forma similar a RPC):
    - El cliente del objeto invocará a un stub cliente (llamado **proxy**)
    - El proxy enviará la invocación al stub servidor por la red.
    - El stub servidor, llamado **esqueleto**, invocará al objeto.
  - Nótese que tanto el objeto remoto como el proxy implementarán la misma interfaz.

3.1.- Objetos distribuidos

- Objeto distribuido = objeto remoto + proxy + esqueleto.
- El proxy y el objeto remoto tienen la misma interfaz.
- Para invocar un objeto remoto, el cliente invoca al proxy y la invocación llegará al objeto remoto.



3.1.- Objetos distribuidos

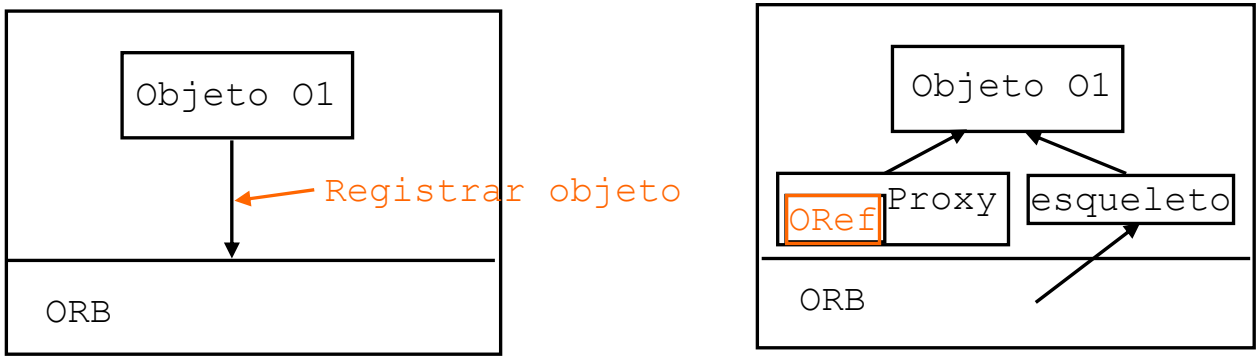
- ORB = Object Request Broker: Gestor de invocaciones a objeto.
- El ORB es una capa de software que facilita el desarrollo de aplicaciones distribuidas orientadas a objeto.
- Proporciona servicios utilizados por los proxies, los esqueletos y por el código de las aplicaciones.
- La misión más importante del ORB es dirigir las invocaciones desde los proxies a los esqueletos adecuados situados en la máquina adecuada.

3.2.- Referencias a objeto

- Una de las diferencias más importantes entre los sistemas RPC y los sistemas OO distribuidos consiste en la posibilidad de los sistemas OO de pasar objetos como argumento en las invocaciones.
  - Por valor: se copia todo el objeto y el objeto remoto que recibe la invocación recibe una copia del objeto.
  - Por referencia: el objeto remoto recibe una **referencia** al objeto.
- En los sistemas de objetos distribuidos, es clave el paso de objetos por referencia.

3.2.- Referencias a objeto

- Un proceso tiene que informar al ORB que dispone de un objeto invocable. Para ello **registra** el objeto:

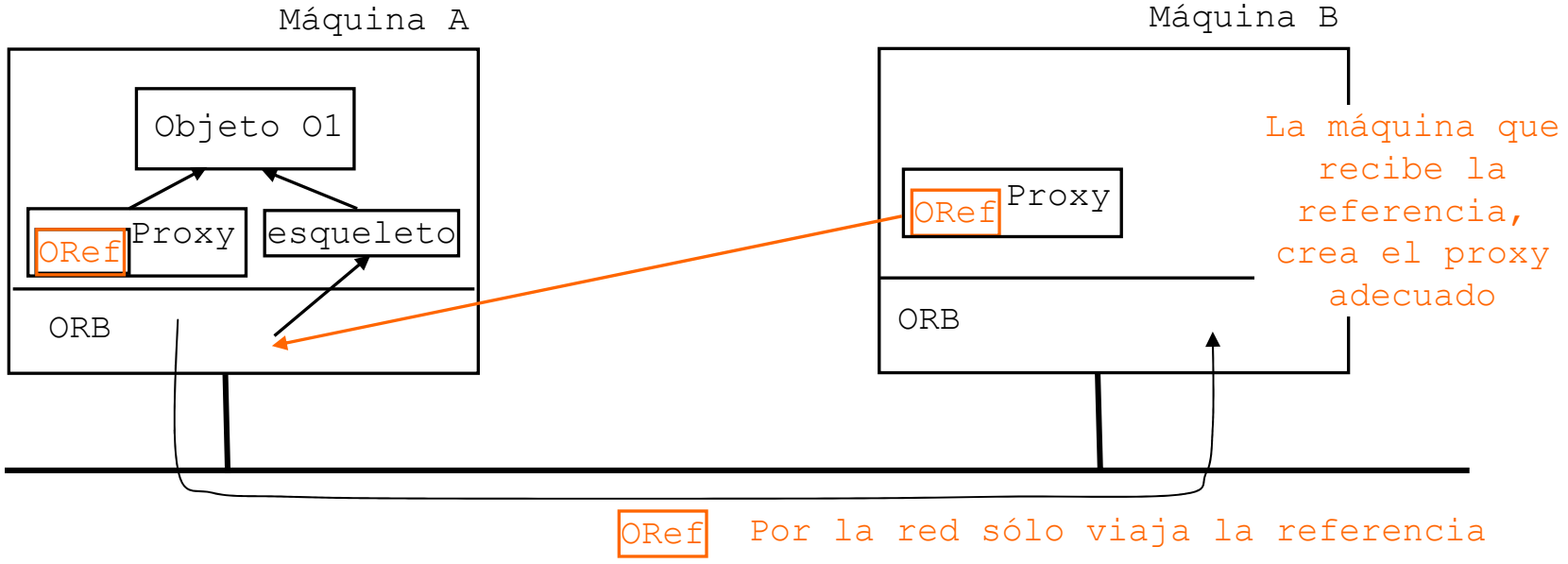


- Como resultado del registro:
  - El ORB crea un esqueleto que permitirá dirigir las invocaciones remotas al objeto
  - El ORB retorna una referencia al objeto. Lo habitual será que retorne un proxy que contendrá la **referencia** al objeto (Oref)

3.2.- Referencias a objeto

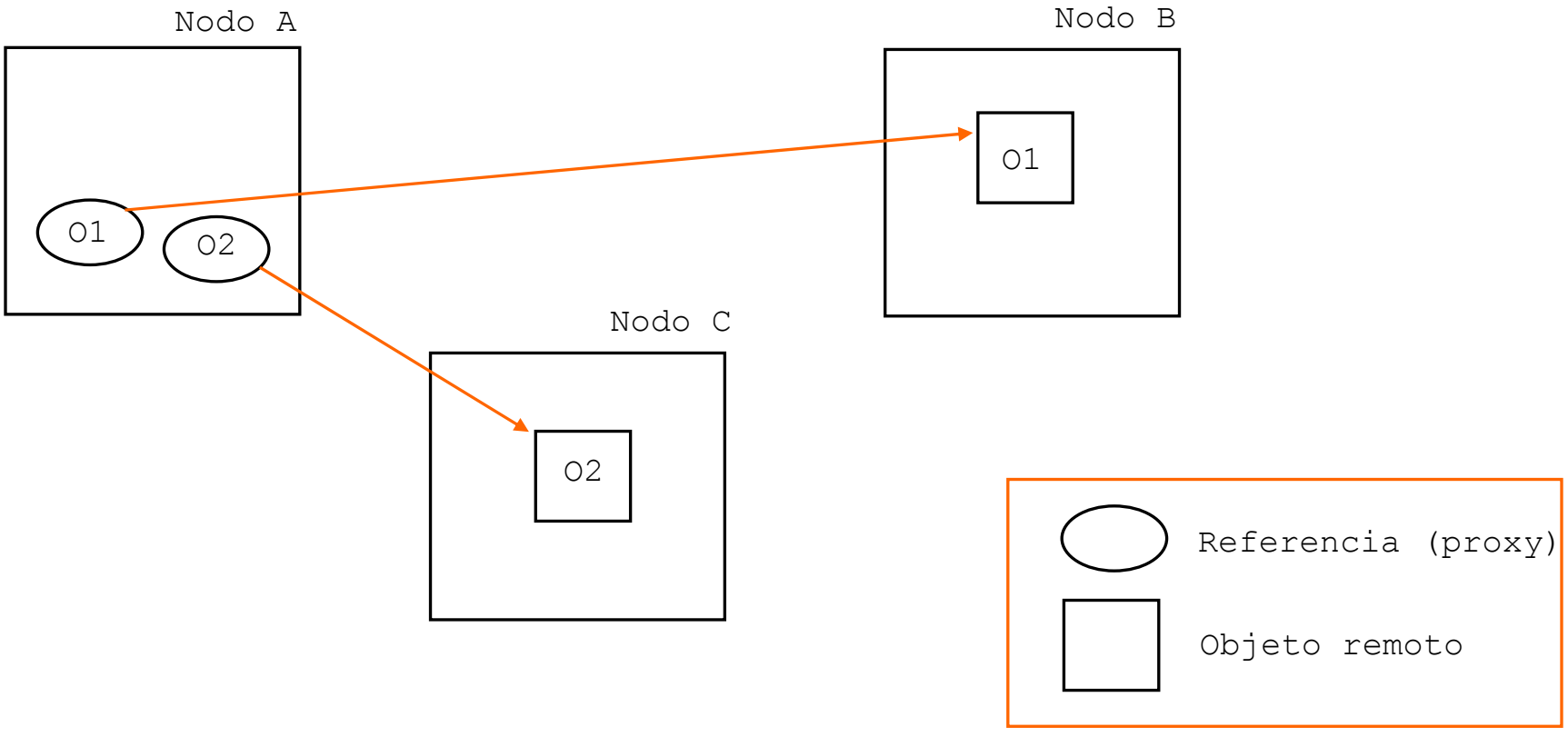
- Contenido de las referencias a objeto:
  - Dirección física del ORB que contiene el objeto (por ejemplo: IP + puerto)
  - Identificador del objeto dentro de la máquina (una máquina puede tener más de uno)
  - Interfaz que se registra (un objeto puede tener más de una)

- Paso de referencias:



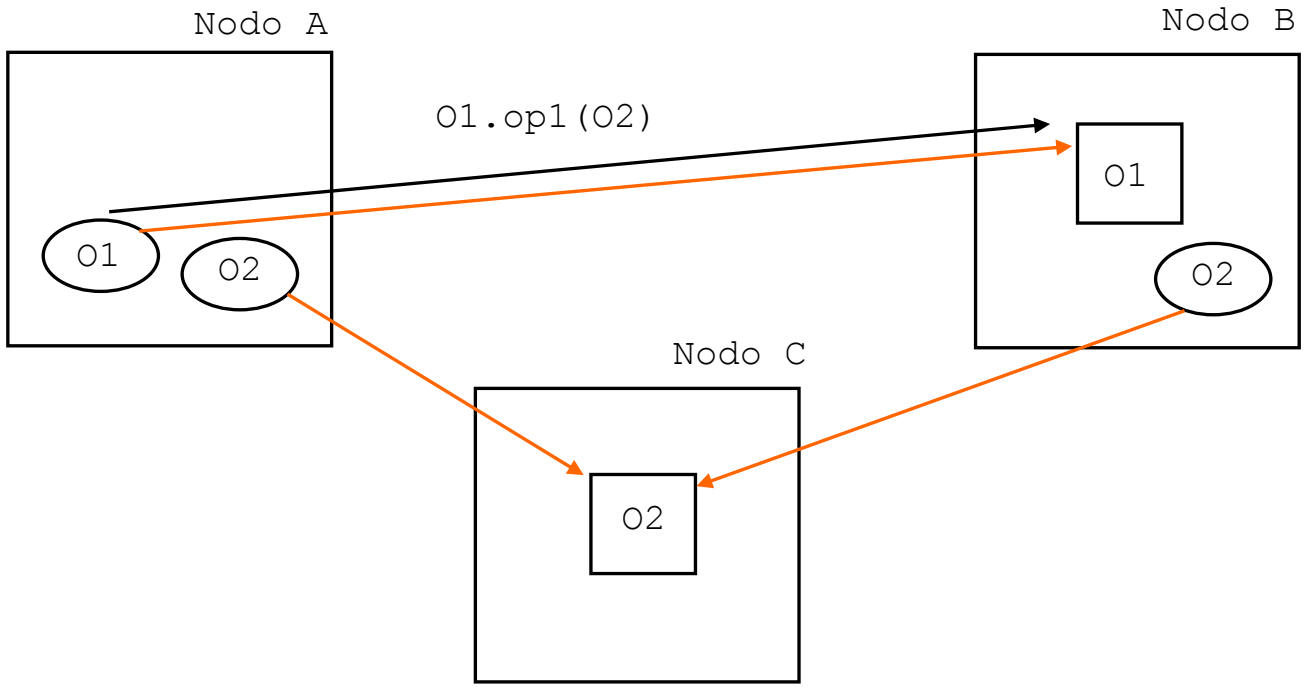
3.2.- Referencias a objeto

Ejemplo de paso de referencias: supongamos que la máquina A tiene una referencia a O1 y otra a O2. Supongamos que la máquina B tiene la implementación de O1 y la máquina C, tiene la implementación de O2.



3.2.- Referencias a objeto

Ejemplo de paso de referencias (cont): Supongamos que la máquina A invoca al método op1 de O1, pasándole como argumento el objeto O2. Es decir, La máquina A invoca **O1.op1(O2)**



Como resultado, la máquina B, recibe una referencia al objeto O2

3.3.- Optimizaciones

- **Objetos volátiles vs objetos persistentes**: Los objetos persistentes almacenan su estado en memoria no volátil, de forma que no se perderá su estado.
- **Invocaciones estáticas vs invocaciones dinámicas**: Es posible invocar objetos sin tener el código del proxy. En este caso, se realizan invocaciones de la forma:
  - ORB.invoke (Oref, método, parámetros);
- **Registro automático**: Es posible evitar el registro de objetos en el ORB. En este caso, el código de los proxies registran los objetos que reciben como argumentos, para aquellos objetos que no estén registrados previamente.

Tema 2.- Comunicación

1. Protocolos basados en niveles
2. Llamada a procedimiento remoto (RPC)
3. Invocación a objeto remoto (ROI)
4. Comunicación basada en mensajes
5. Comunicación basada en flujos

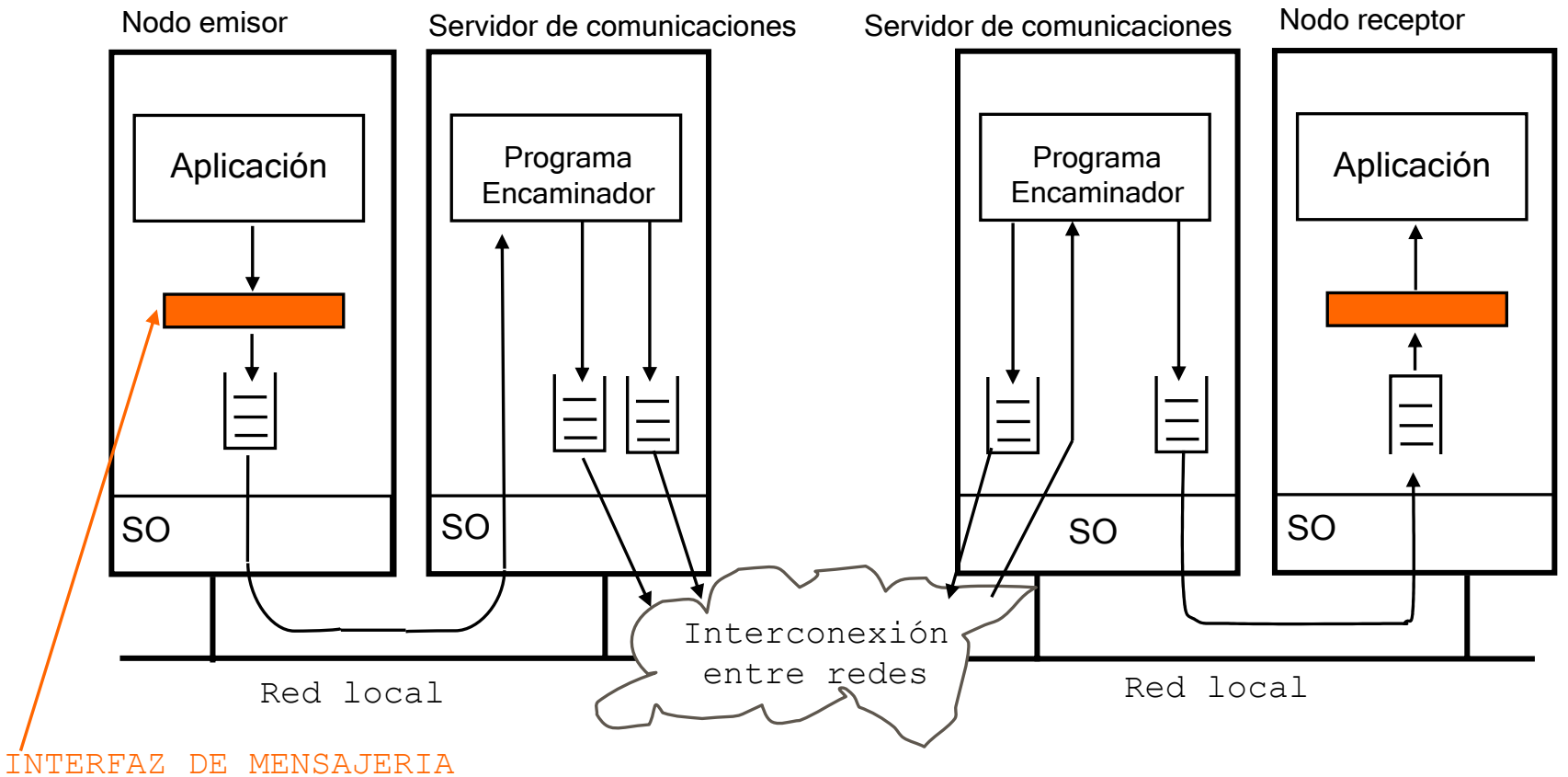
Bibliografía: Capítulo 2 de Tanenbaum

4.- Comunicación basada en mensajes

- Introducción
- Persistencia y sincronización
- Ejemplos
- Middleware orientado a mensajes

4.1.- Introducción

- El paso de mensajes en sistemas distribuidos lo podemos asimilar al siguiente esquema:



4.1.- Introducción

- Comunicación basada en mensajes:
  - La comunicación la inicia una aplicación, que desea enviar un mensaje a otra aplicación situada en una máquina distinta.
  - Las máquinas que intercambian mensajes hacen uso de una interfaz para acceder a los servicios de mensajería.
  - Las máquinas están conectadas a través de una red de servidores de comunicaciones.
  - Los servidores de comunicaciones se encargan de encaminar los mensajes y proporcionar la semántica que requiera el paso de mensajes.
  - Todas las máquinas disponen de “buffers” para el envío de mensajes y al menos la máquina receptora dispone de “buffers” para la recepción.
- Ejemplo: correo electrónico.

4.2.- Persistencia y sincronización

- La semántica que puede proporcionar el servicio de mensajería tiene dos vertientes: persistencia y sincronización.

Persistencia:

- Comunicación persistente: los mensajes enviados al receptor son almacenados por el sistema de comunicación hasta que sean recibidos por el receptor.
  - El emisor no necesita continuar ejecutándose tras el envío
  - El receptor no tiene porqué estar funcionando cuando el emisor envía el mensaje
- Comunicación no-persistente: los mensajes solo se almacenan en el sistema de comunicaciones mientras el emisor y el receptor se encuentren en ejecución.
  - Cada nodo sólo envía al siguiente nodo de la “cadena” si el receptor está funcionando (encaminamiento store & forward)

4.2.- Persistencia y sincronización

Sincronización:

- Comunicación asíncrona: El emisor continúa ejecutándose justo después de haber enviado el mensaje:
  - A su buffer local: mayor asincronía.
  - Al sistema de comunicaciones: menor asíncronía.
- Comunicación síncrona: Se bloquea al emisor hasta que el mensaje haya llegado al receptor:
  - Al buffer local (débil).
  - Esperar hasta que la aplicación procese el mensaje (fuerte).

4.2.- Persistencia y sincronización

Combinaciones:

- **Comunicación persistente asíncrona**: los mensajes se almacenan (p.ej: e-mail):
  - En el buffer local del emisor.
  - En el primer servidor de comunicaciones.
- **Comunicación persistente síncrona**: los mensajes se almacenan en (p.ej: e-mail):
  - El buffer local del receptor (fuerte)
  - En último servidor de comunicaciones (débil)
- **Comunicación no-persistente asíncrona**: p.ej: UPD
- Comunicación no-persistente síncrona:
  - Espera a que el mensaje llegue al buffer local del receptor (**comunicación no-persistente síncrona basada en recepción**)
  - Espera a que el mensaje se entregue a la aplicación (**comunicación no-persistente síncrona basada en entrega**) p.ej: RPC asíncrona
  - Espera que el mensaje sea procesado por la aplicación (**comunicación no-persistente síncrona basada en respuesta**) p.ej: RPC

4.3.- Ejemplos

- Comunicación no-persistente:
  - Berkeley sockets
    - Servidor: `socket(), bind(), listen(), accept(), {read(), write()}, close()`.
    - Cliente: `socket(), connect(), {write(), read()}, close()`.
  - The Message Passing Interface (MPI): Para entornos de computación en paralelo. Permite en envío de mensajes a grupos de procesos con todas las variantes no-persistentes.
- Comunicación persistente:
  - IBM MQSeries: se utiliza en entornos financieros basados en mainframes.

4.4.- Middleware orientado a mensajes

- Los mensajes no-persistentes suelen implementarse en el nivel de transporte de los protocolos de comunicación.
- Los mensajes persistentes requieren la construcción de software adicional: Middleware orientado a mensajes (Message-oriented middleware (MOM)).
- Tipos:
  - **Sistemas de colas de mensajes**: construidos para garantizar la entrega de los mensajes
    - Cola de envío
    - Cola de recepción
    - Colas intermedias = relays
  - **Gestores de mensajes** (message brokers): realizan alguna función adicional sobre los mensajes (p.ej)
    - Cambio de formato para lograr interoperabilidad
    - Generación de varios mensajes por cada entrada de mensaje

Tema 2.- Comunicación

1. Protocolos basados en niveles
2. Llamada a procedimiento remoto (RPC)
3. Invocación a objeto remoto (ROI)
4. Comunicación basada en mensajes
5. Comunicación basada en flujos

Bibliografía: Capítulo 2 de Tanenbaum

5.- Comunicación basada en flujos

- Comunicación basada en mensajes no es suficiente si los diferentes mensajes que se transmiten están ligados entre ellos por aspectos **temporales**.
- Tipos de datos a transmitir atendiendo a las relaciones temporales:
  - **Datos continuos**: las relaciones temporales entre los diferentes datos son esenciales para interpretar el significado de los propios datos. P.ej: sonido, video.
  - **Datos discretos**: las relaciones temporales no son esenciales. P.ej: ficheros de texto, imágenes estáticas, ejecutables, etc.
- Flujos de datos: secuencia de unidades de datos: aplicable tanto a datos discretos como a datos continuos
  - Flujos de datos discretos. P.ej: TCP / tuberías
  - Flujos de datos continuos. Para transmitir datos continuos

5.- Comunicación basada en flujos

- Modos de transmisión:
  - Modo asíncrono: los datos del flujo se transmiten uno tras otro, pero no hay más restricciones temporales.
  - Modo síncrono: hay un tiempo máximo para la transferencia de cada dato (pero no mínimo).
  - Modo isócrono: los datos deben transferirse dentro de un rango temporal delimitado.
- Para datos continuos (modos síncrono e isócrono) es necesaria la reserva de recursos previa a la transmisión.
- Es posible la transmisión de varios flujos de datos que deban estar sincronizados entre ellos. P.ej.: sonido y video.
- Middleware para transmisión multimedia:
  - Sincronización de medias
  - Compresión
  - Calidad del servicio