Introducción a SD

8

Sistemas Operativos y Distribuidos

Prof. Javier Echaiz
D.C.I.C. – U.N.S.
http://cs.uns.edu.ar/~jechaiz
je@cs.uns.edu.ar

Estructura de los Sistemas Distribuidos

Sistemas Distribuidos

Computación Paralela y Distribuida

Computación Paralela (1)

La computación paralela se orienta a resolver rápidamente una tarea empleando múltiples procesadores simultáneamente.

Esta práctica se vuelve popular a partir de fines de los ’80s.

La posibilidad de construir hardware paralelo de alto rendimiento no fue suficiente, el real desafío fue el software.

Computación Paralela (2)

Los ambientes de programación paralela fueron dificultosos de usar y más todavía, estaban atados a arquitecturas particulares. Solo se desarrolló esta práctica en problemas científicos y de ingeniería … el mercado era muy restringido.

Sin embargo la perspectiva de la programación paralela se ha tornado más atractiva con las redes de PCs o estaciones de trabajo (**clusters**).

Computación Distribuida

Un sistema distribuido es una colección de computadoras autónomas que están conectadas unas con otras y cooperan compartiendo recursos (*e.g.* impresoras y bases de datos).

Son usados en aplicaciones comerciales y procesamiento de datos. Van desde pequeñas configuraciones cliente-servidor sobre redes en varias escalas hasta Internet.

Los sistemas distribuidos no son vendidos como tal pero crecen natural e incrementalmente.

Las computadoras individuales son interconectadas por redes locales y éstas pueden estar interconectadas en una red amplia en la medida que la necesidad lo amerita.

Computación Distribuida

El crecimiento de la computación distribuida ha creado una serie de nuevos problemas que están en investigación. Los problemas elementales incluyen la ausencia de reloj común y la posibilidad de fallas de transmisión.

En el más alto nivel:

- El acceso a recursos compartidos deben administrarse de modo tal que los diferentes programas de usuarios no se interfieran.
- Heterogeneidad operativa (hardware, sistemas operativos y los lenguajes, que deberían ser interoperativos).
- Seguridad.

Comparación entre Computación Paralela y Distribuida

Aunque los dos campos han evolucionado de manera diferente, tienen características comunes:

- Son usados múltiples procesadores
- Los procesadores están conectados por algún tipo de red.
- Actividades computacionales múltiples están en progreso al mismo tiempo y cooperan unas con otras.

La computación paralela divide una aplicación en tareas que son ejecutadas al *mismo tiempo*, mientras que la computación distribuida divide una aplicación en tareas que son ejecutadas en *distintas locaciones* usando *diferentes recursos.*

Características de la Computación en Paralelo

- Una aplicación es dividida en subtareas que son resueltas simultáneamente, general-mente de manera fuertemente acoplada.
- Se considera una aplicación por vez y el objetivo es el *speed-up* de procesamiento de la misma.
- Los programas usualmente corren en arquitecturas homogéneas y pueden tener memoria compartida

Características de la Computación Distribuida

Pone énfasis en lo siguiente:

- Las computaciones usan múltiples recursos que están situados en locaciones físicamente distantes.
- Corren múltiples aplicaciones a la vez, éstas pueden pertenecer a distintos usuarios.
- Los sistemas distribuidos son generalmente heterogéneos.
- Una cuestión de interés es esconder las partes internas del sistema de manera tal que el sistema distribuido luzca como una única máquina para los usuarios.
- Los sistemas distribuidos no tienen memoria compartida (a nivel de hardware).

Motivaciones para la Computación Paralela y Distribuida (1)

- Rendimiento absoluto: aplicaciones científicas e ingeniería. Por ejemplo: modelado del clima y el tiempo, simulaciones astrofísicas y diseño de materiales, autos y aviones; aplicaciones comerciales como bases de datos, optimizaciones combinatorias, inteligencia artificial.
- Relación precio/rendimiento.
- Razones tecnológicas: p.e. número de chips, frecuencia de trabajo del reloj, etc.

Motivaciones para la Computación Paralela y Distribuida (2)

- Aplicaciones con paralelismo o distribución inherentes: p.e. sistemas de información de empresas o compañías, mundo real.
- Recursos compartidos.
- Crecimiento incremental.
- Otras razones: balance de carga, utilización de capacidad ociosa.

Rendimiento de Aplicaciones Simples

Métrica obvia: "tiempo de corrida" (o costo de ejecución).

Speed-up:

$$\text{speed-up}(P) = T_1/T(P)$$

donde:

T(P): tiempo de corrida del programa paralelo en P procesadores.

T_1: tiempo de corrida de un programa secuencial de referencia.

En general, este último es el programa secuencial más rápido que soluciona el problema.

Eficiencia:

$$\text{eficiencia}(P) = \text{speed-up}(P) / P$$

donde:

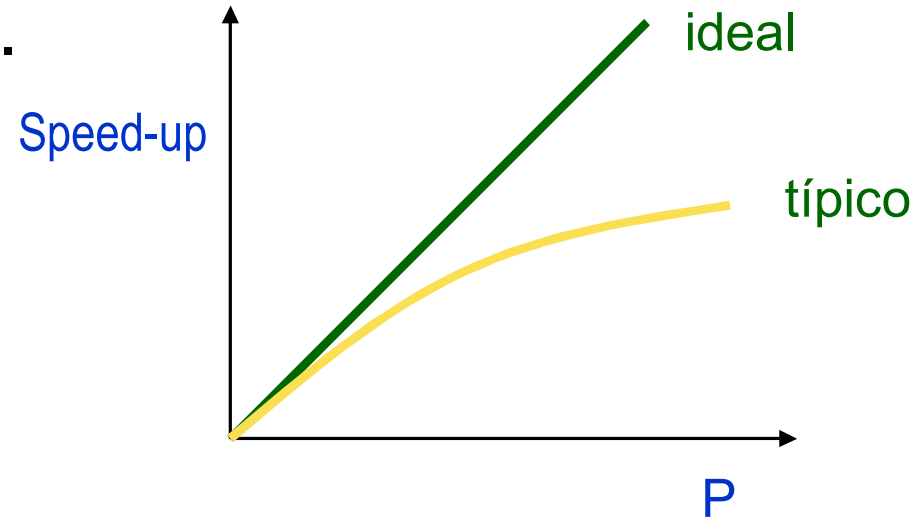
P: número de procesadores.

Rendimiento de Aplicaciones Simples (cont.)

Idealmente se espera que el speed-up crezca linealmente y la eficiencia sea 1(uno) para todo P.

Hay casos donde el speed-up es *superlineal* o sea que k procesadores resuelven una tarea en menos que *un k-ésimo* del tiempo de corrida secuencial.

Comportamiento explicable por el aumento del tamaño del caché.



Rendimiento de Aplicaciones Simples (cont.)

Razones de la diferencia entre el speed-up ideal y típico:

- ***Ley de Amdahl***: primera observación [Amdahl, *Validity of the single processor approach to achieving large scale computer capabilities*; Proc. AFIPS, pp30, 1967]: cada computación contiene una porción serial de ejecución, es decir, alguna parte *s* del código no es posible paralelizarlo; segunda observación (ley Gustafson-Barsis)[El-Rewini,Lewis; Distributed and Parallel Computing, *Manning Publications*, 1998]: establece que muy frecuentemente se usan programas paralelos para resolver instancias más grandes de un problema que su contraparte secuencial; así en la medida que el número de procesadores crece, T_1 crece mientras que *s* permanece casi constante, en la práctica T_1/s no es una constante.

Rendimiento de Aplicaciones Simples (cont.)

- **Administración de tareas y balance de carga**: El manejo de un conjunto de tareas induce cierta sobrecarga. Es más, a veces es difícil sino imposible tratar de dividir el trabajo entre procesadores.

- **Comunicación y sincronización**: La paralelización introduce la necesidad de comunicación y sincronización. En las arquitecturas actuales estas actividades son lentas comparadas con la computación (por órdenes de magnitud). Los costos de comunicación son medidos en términos de latencia y ancho de banda. *Latencia* es el tiempo que se toma un mensaje para ir de una locación a otra. *Ancho de banda* es la cantidad de datos que pueden ser transferidos por unidad de tiempo en estado estable. Los costos de comunicación pueden ser reducidos pero no evitados.

Rendimiento en Aplicaciones Múltiples

En aplicaciones cliente-servidor, en la cual hay múltiples jobs enviados al servidor, el rendimiento es medido por el *tiempo de respuesta*. El número de jobs es un factor importante de rendimiento.

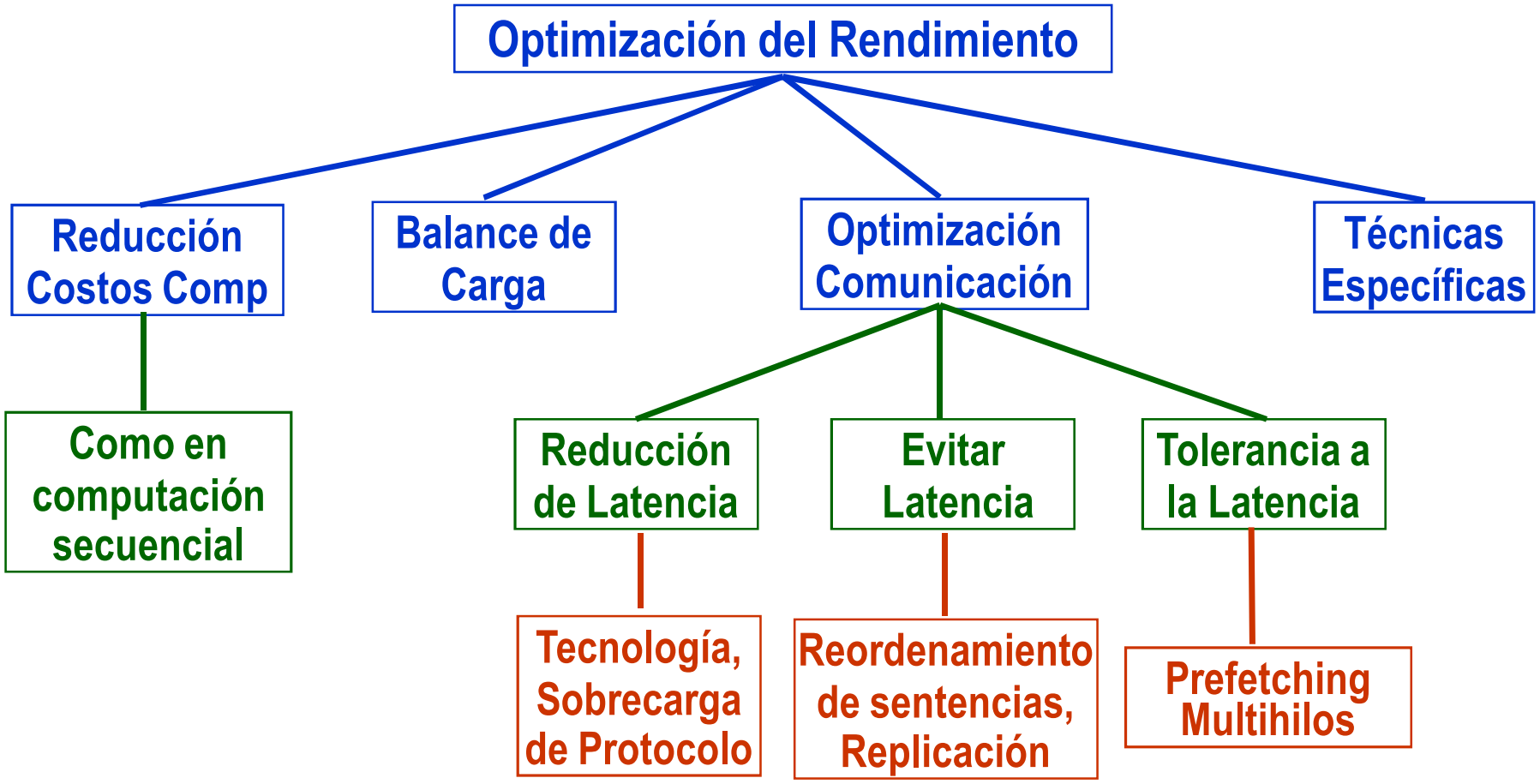
Influye la forma en que el servidor trata internamente los jobs (secuencial vs. paralelo).

La habilidad del sistema para manejar cargas grandes es referida como *procesamiento total* (throughput).

El rendimiento también puede medirse como la *utilización de recursos* (p.e. *utilización del procesador*).

Aplicaciones como video por demanda dependen de la disponibilidad de recursos que se comparten con otras aplicaciones, p.e. *ancho de banda* y *espacio de buffers*. *Calidad de Servicios* es concerniente con la provisión de una cantidad garantizada de esos recursos.

Optimización del Rendimiento



Complejidad de Diseño de Programa

No solo la optimización del rendimiento es complejo sino que también lo es el diseño de programas con respecto al caso secuencial.

La aplicación debe ser dividida en tareas individuales, esto implica la elección de una apropiada *granularidad* o *tamaño de tarea*. Puede ser *fina*, *media* o *gruesa*. En general una granularidad gruesa da una buena relación alta computación-comunicación.

Otro punto es la *planificación de tareas* en sentido espacial y temporal y la *distribución de datos*. Esto tiene una gran influencia en el rendimiento.

El *manejo de las comunicaciones* y la *sincronización* (llamado en conjunto *coordinación*) son muy importantes. Solo coordinadamente pueden cooperar los procesos y compartir recursos. La *correctitud* es fundamental dado que la coordinación puede generar efectos no deseados.

Otro aspecto importante es el *interbloqueo*.

Problemas Intratables

En muchos casos hay serias limitaciones.

Por ejemplo: dentro del grupo de problemas intratables resulta el referido a la planificación de tareas, mas específicamente el de mapear un grafo de programa a la arquitectura.

Los procesos y procesadores son representados por nodos y las precedencias y/o comunicaciones por los lados.

El objetivo es encontrar un mapeo de los nodos del grafo de programa en los nodos del grafo de la arquitectura, que minimiza algunas funciones de costo de rendimiento.

Para varias prácticas relevantes de costos de rendimiento toma mucho tiempo encontrar una solución óptima o una solución que garantice estar próxima a la óptima.

Soporte de Desarrollo de Software

En la medida que los programas paralelos se hacen cada vez mas grandes, los objetivos de la Ingeniería de Software para computación secuencial deben ser redireccionados. Las herramientas existentes para desarrollo de la computación secuencial deberían ser transferidas al dominio paralelo.

La situación es mejor en computación distribuida donde se han desarrollado, con el soporte de la Ingeniería de Software, estándares como CORBA.

Soporte de Desarrollo de Software (cont.)

Para computación distribuida y paralela, el soporte de la Ingeniería de Software es más importante que para computación secuencial porque:

- El diseño de programas es más complejo.
- Son abordados problemas más grandes (especialmente en computación distribuida).
- El campo es nuevo para muchos programadores.
- La idea de sistemas descentralizados es contra intuitiva para el pensamiento humano.

La transición desde computación secuencial a computación distribuida y paralela es realizada extendiendo lenguajes de programación secuencial en lugar de desarrollar nuevos.

Transparencia

La ***transparencia*** es el ocultamiento al usuario o al programador de las funcionalidades del sistema.

La funcionalidad escondida es manejada automáticamente de manera que el usuario no necesite conocer el detalle.

Por otro lado el usuario pierde control sobre la funcionalidad.

Hay diferentes grados de transparencia exhibida a usuarios y programadores que trabajan en diferentes niveles del sistema. Por ejemplo: un programador que maneja la distribución de datos entre procesadores, e implementa una interfaz de usuario para el cual la distribución es transparente.

El concepto de transparencia se asocia cercanamente a la imagen de un sistema simple.

Transparencia (Cont.)

Bien conocidas formas de transparencia incluyen:

Acceso Transparente: habilita a que objetos de información locales y remotos sean accedidos usando operaciones idénticas.

Locación Transparente: permite que objetos de información locales y remotos sean accedidos sin conocimiento de su locación.

Concurrencia Transparente: habilita a varios procesos a operar concurrentemente sobre objetos de información compartida sin interferencias entre ellos.

Replicación Transparente: implica múltiples instancias de objetos de información usados para incrementar confiabilidad y rendimiento sin conocimiento de las réplicas por los usuarios o las aplicaciones.

Transparencia (Cont.)

Fallas Transparentes: permite el encubrimiento de fallas, los usuarios y/o aplicaciones completan sus tareas a despecho de fallas de hardware o software.

Migración Transparente: permite el movimiento de objetos de información en el sistema sin afectar las operaciones de usuarios o aplicaciones.

Rendimiento Transparente: el sistema se reconfigura para mejorar el rendimiento cuando la carga varía.

Escalabilidad Transparente: el sistema y las aplicaciones se expanden escalarmente sin cambiar la estructura del sistema o los algoritmos de aplicación.

Transparencia (Resumen)

Transparencia	Descripción
Acceso	Esconde diferencias en la representación de datos y como un recurso es accedido.
Locación	Esconde la locación del recurso.
Migración	Esconde el movimiento de un recurso a otra locación.
Relocación	Esconde que un recurso pueda ser movido a otra locación mientras está en uso.
Replicación	Esconde desde donde es utilizado un recurso compartido por varios usuarios competidores.
Concurrencia	Esconde que un recurso pueda ser compartido por varios usuarios competidores.
Fallas	Esconde la falla y recuperación de un recurso.
Persistencia	Esconde si un recurso (software) esta en memoria o disco.

Portabilidad del Código y del Rendimiento

Un programa es *portable* si corre en una variedad de arquitecturas, inclusive las futuras. Tiene claras ventajas:

- El esfuerzo de escribir un programa se amortiza en el tiempo si es muy usado.
- Se puede pasar fácilmente a arquitecturas mas potentes si es necesario más poder de computación.
- Se puede pasar fácilmente a arquitecturas alternativas si el sistema original *capotó*.
- Los programas pueden ser desarrollados en plataformas relativamente baratas.

Muchas herramientas permiten hoy en día escribir código portable.

Portabilidad del Código y del Rendimiento (Cont.)

El siguiente aspecto de la portabilidad, la *portabilidad del rendimiento*, no está suficientemente resuelto. Requiere que un programa corra en una variedad de arquitecturas de modo que el rendimiento observado refleje el rendimiento potencial de la respectiva arquitectura.

La portabilidad de rendimiento es difícil de lograr, dado que portabilidad y rendimiento son objetivos conflictivos.

El problema es que no se pueden aprovechar las especificidades de las arquitecturas.

Escalabilidad

La *escalabilidad* es un objetivo obvio con un significado difuso.

Se dice que un sistema es escalable si sus recursos pueden ser expandidos para acomodarse a un mayor poder de computación.

A nivel de hardware una típica expansión es el agregado de procesadores. Cuando se agregan procesadores es necesario mejorar, también, las comunicaciones del sistema.

También se pueden reemplazar recursos por otros más poderosos.

A nivel de software se puede reemplazar el sistema operativo por una nueva versión del mismo.

Se puede aplicar a programas, puede ser que el mismo es puesto en más procesadores. En otros casos puede ser aprovechable adaptar el programa a una nueva situación (p.e. eligiendo un nuevo algoritmo).

Escalabilidad (Cont.)

Escalando un programa o sistema se espera cierto incremento de rendimiento. El término escalabilidad puede o no implicar la garantía de un incremento.

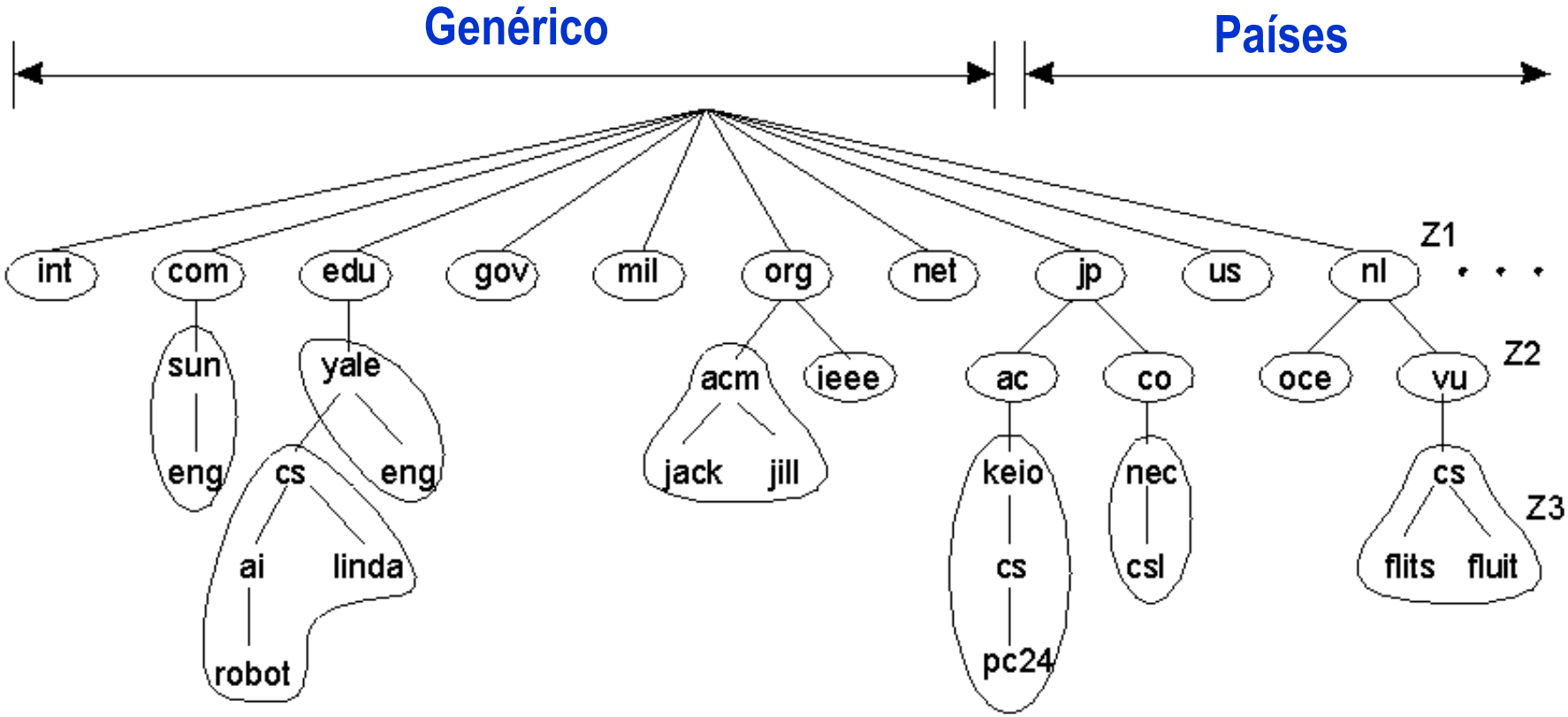
Debido a la variedad de significados, sólo unas pocas sentencias generales pueden hacerse sobre la escalabilidad. Una es que la centralización debe evitarse pues implica "cuellos de botella" y limita el paralelismo..

Algunos ejemplos de las limitaciones de la escalabilidad

Concepto	Ejemplo
Servicios Centralizados	Un único servidor para todos los usuarios.
Datos Centralizados	Una sola guía telefónica en línea.
Algoritmos Centralizados	Ruteo basado en información completa.

Escalabilidad (Cont.)

Ejemplo de técnica de escalamiento: dividir el espacio de nombres DNS en zonas.



Heterogeneidad

Un sistema es *heterogéneo* si está compuesto por hardware y software distinto.

Muchos sistemas distribuidos son heterogéneos, mientras que programas paralelos son escritos frecuentemente para máquinas homogéneas.

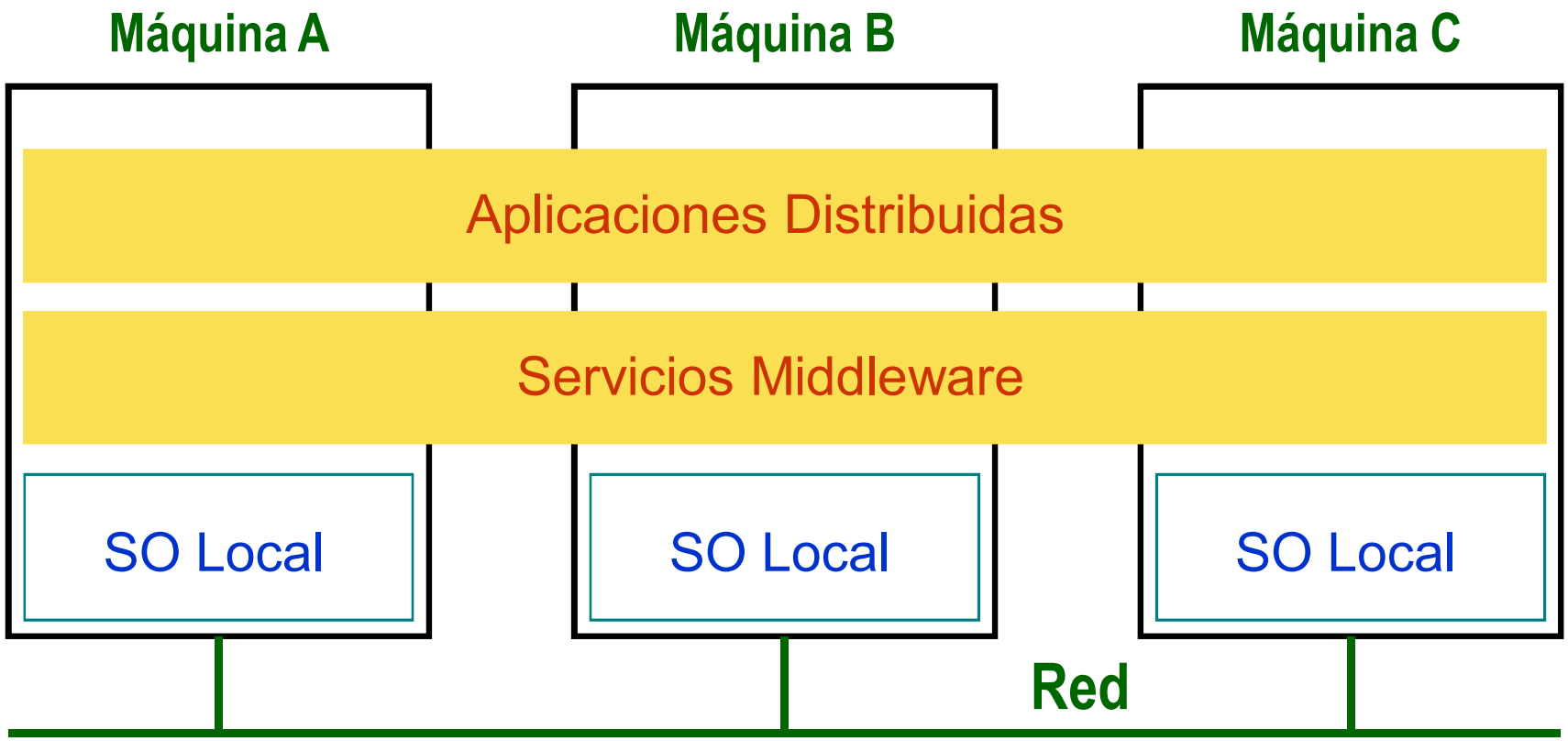
Aquí aparece la noción de interoperabilidad: denota la habilidad de diferentes componentes, posiblemente de distintos proveedores, para interactuar. Estas partes pueden ser hardware o software.

Es de particular interés la interoperabilidad entre partes de programas que fueron escritos en diferentes lenguajes.

Los componentes, para interoperar, deben respetar determinadas interfaces estándares.

Un sistema es *abierto* si los creadores han hecho previsión para que los usuarios o terceras partes agreguen o reemplacen componentes, ….. interfaces públicas.

Sistemas Distribuidos como Middleware



Un sistema distribuido organizado como ***middleware***.

La capa middleware se extiende sobre múltiples máquinas.

Introducción a los Sistemas Distribuidos

Desventajas y Limitaciones

Sistemas Distribuidos: Desventajas

Desventajas de los sistemas distribuidos

- ***Software***: Hay poco software disponible para sistemas distribuidos. La algorítmica es menos controlable.
- ***Redes***: Se pueden saturar o causar otros problemas
- ***Seguridad***: Generalmente la seguridad es escasa o inexistente.

Sistemas Distribuidos: Limitaciones

Limitaciones que crean problemas tecnológicos en los sistemas distribuidos.

- No existe una memoria global (cada nodo tiene su memoria local).
- Establecer un estado global es complejo.
- No se puede asegurar un tiempo global.

Sistemas Distribuidos: Conceptos de Software (1)

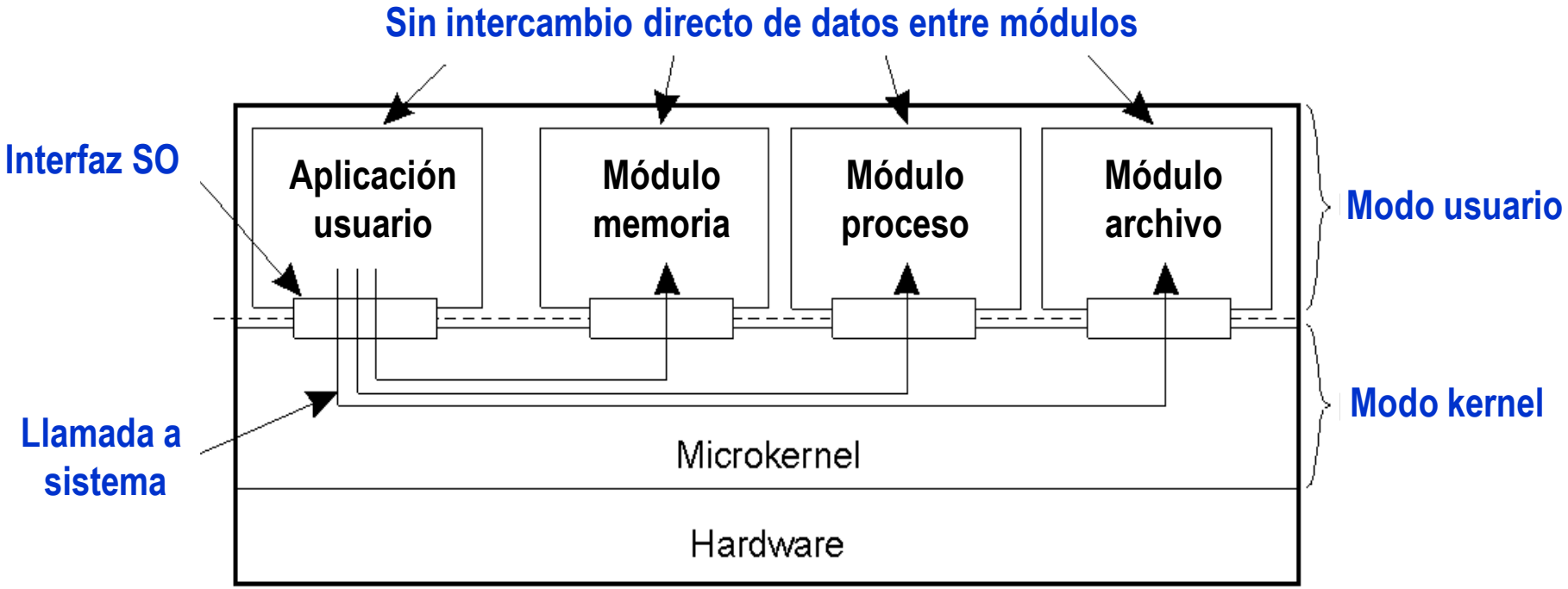
Una vista de

- SOD (Sistemas Operativos Distribuidos)
- SOR (Sistemas Operativos de Red)
- Middleware

Sistema	Descripción	Objetivo Principal
SOD	Sistemas operativos fuertemente acoplados para multiprocesadores y multicomputadoras homogéneas	Esconde y maneja los recursos de hardware
SOR	Sistemas operativos flojamente acoplados para multicomputadoras heterogéneas (LAN y WAN).	Ofrece servicios locales a clientes remotos
Middleware	Capa adicional sobre un SOR implementando servicios de propósito general.	Provee distribución transparente

Sistemas Distribuidos: Conceptos de Software (2)

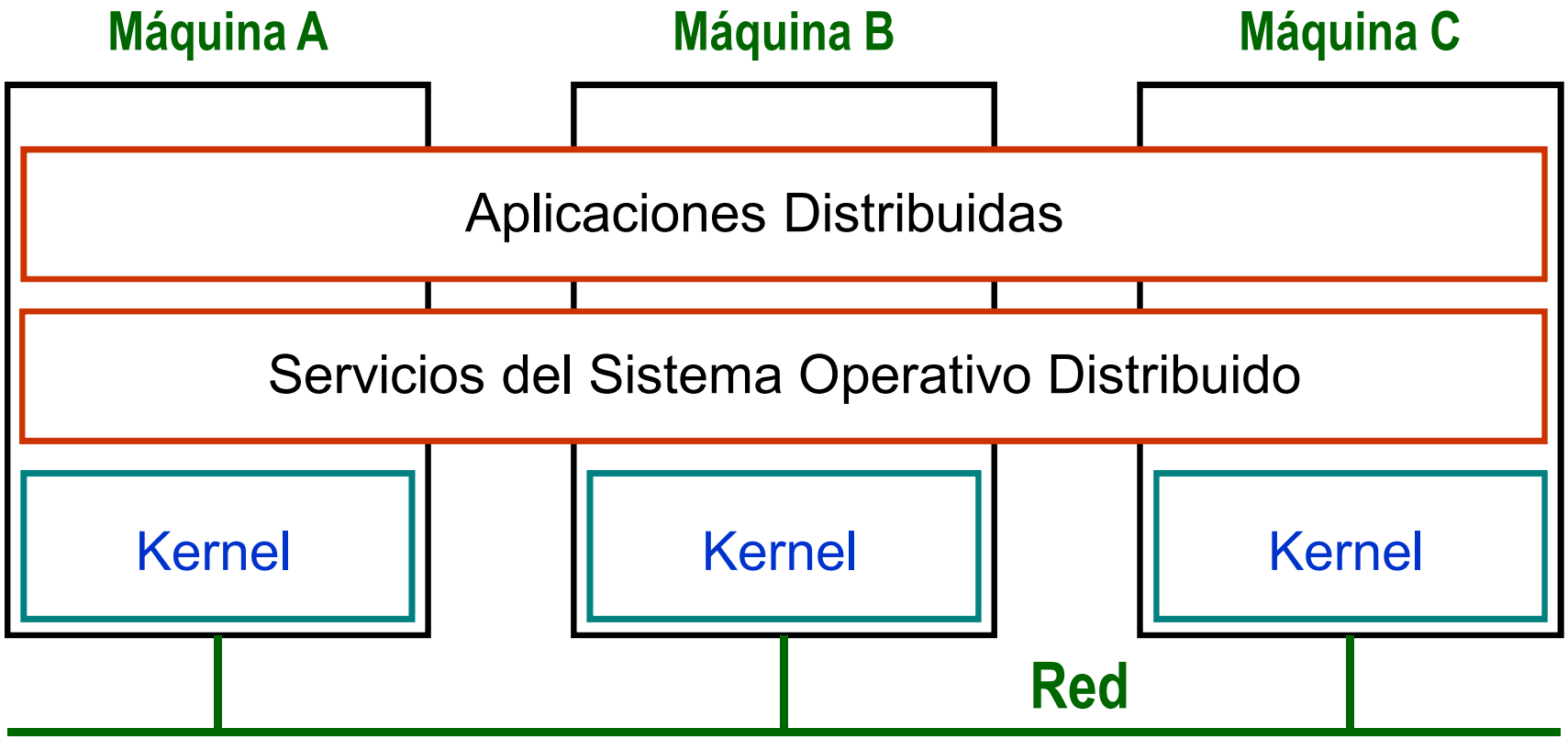
Sistema operativo uniprocesador



Separando aplicaciones del código del sistema operativo mediante un microkernel.

Sistemas Distribuidos: Conceptos de Software (2)

Sistemas Operativos Multicomputadora - 1



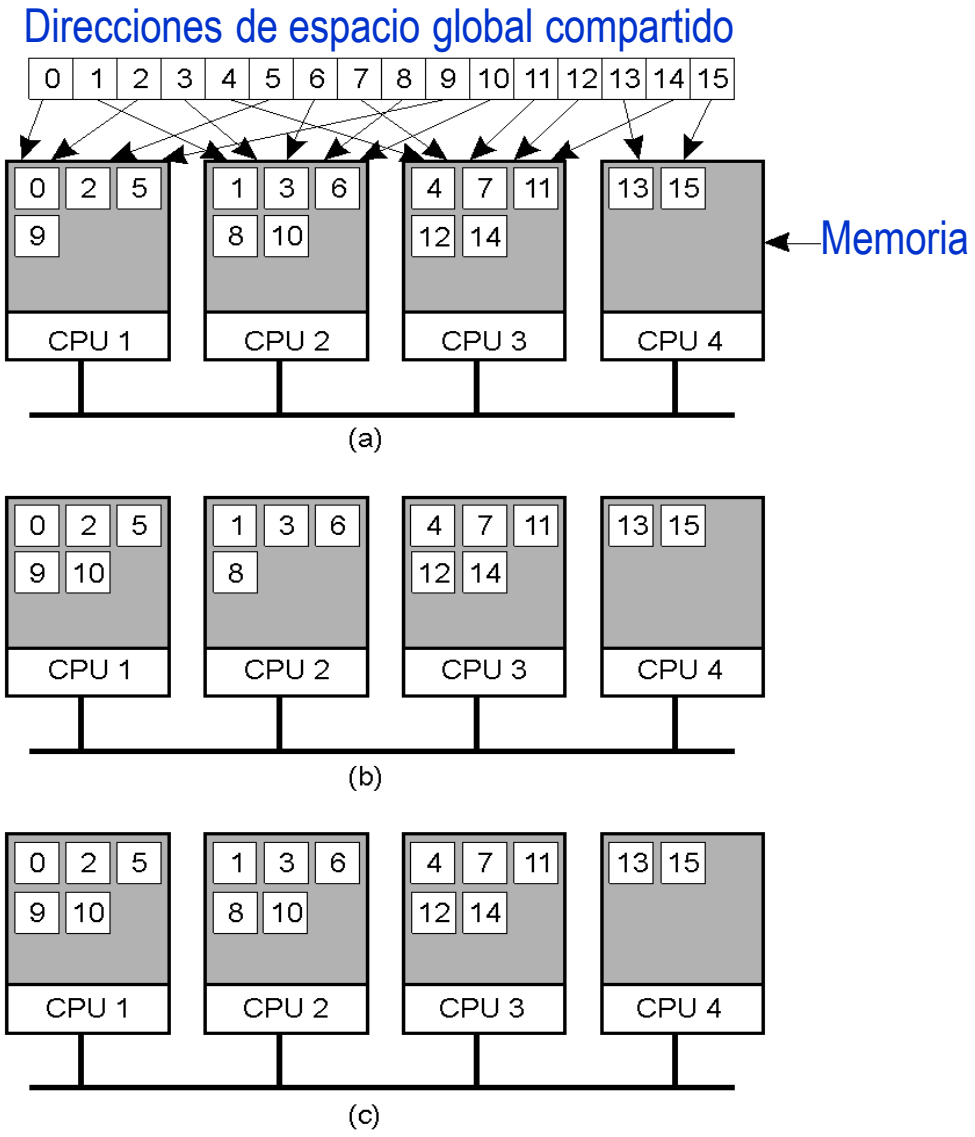
Sistemas Distribuidos: Conceptos de Software (3)

Sistema de Memoria Compartida Distribuida

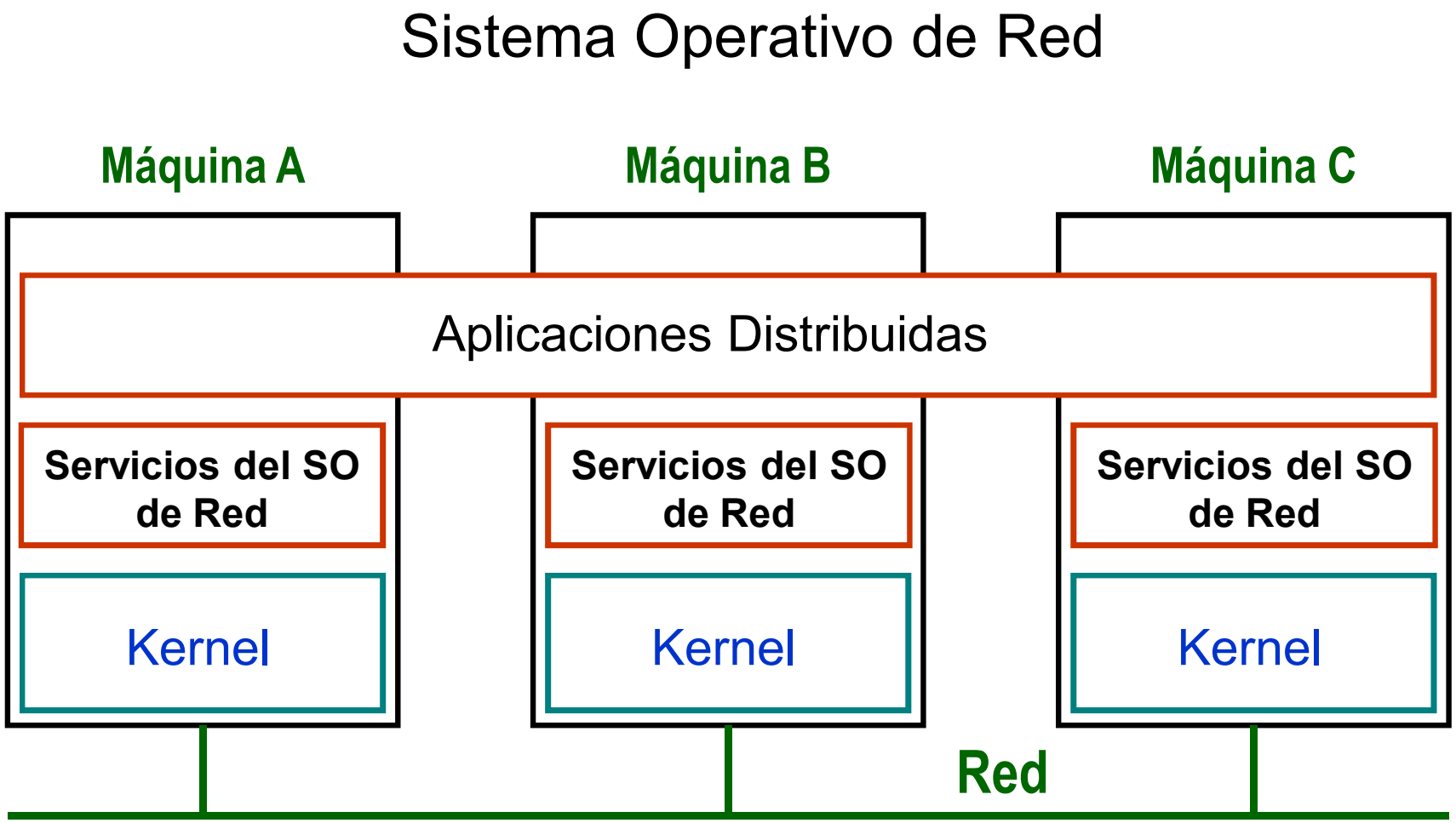
a) Páginas del espacio de direcciones distribuido entre cuatro máquinas.

b) Situación después que la CPU 1 referenció la página 10.

c) Situación si la página 10 es read only y se usa replicación.

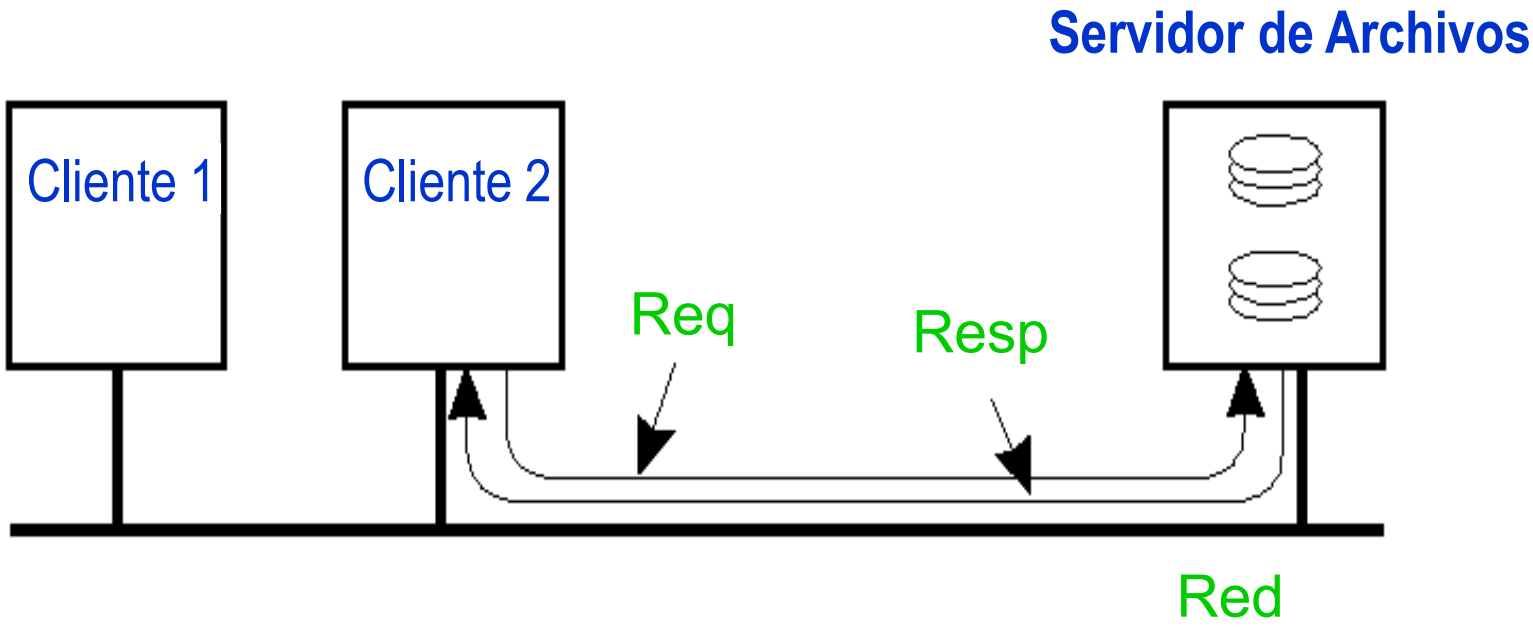


Sistemas Distribuidos: Conceptos de Software (4)



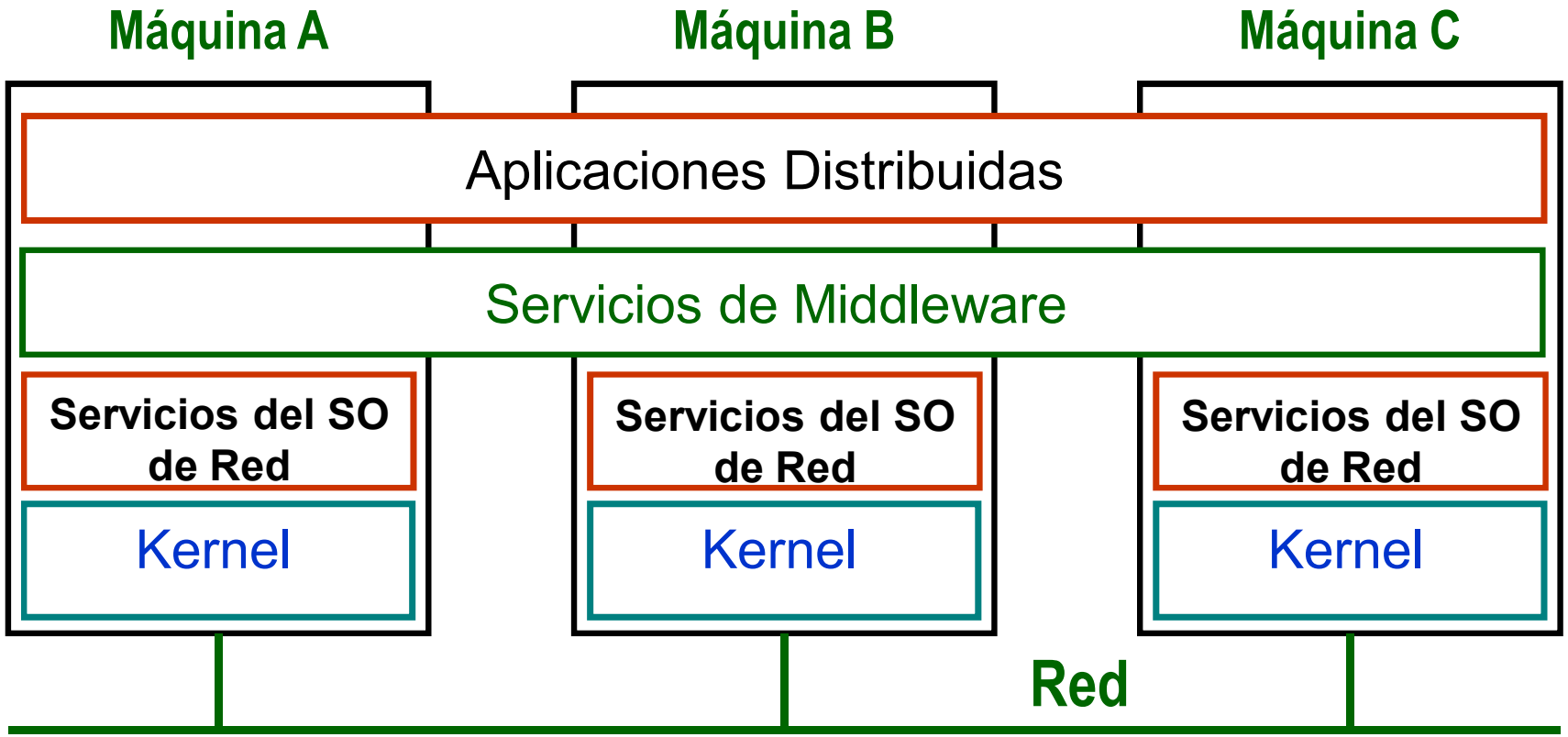
Sistemas Distribuidos: Conceptos de Software (5)

Sistema Operativo de Red



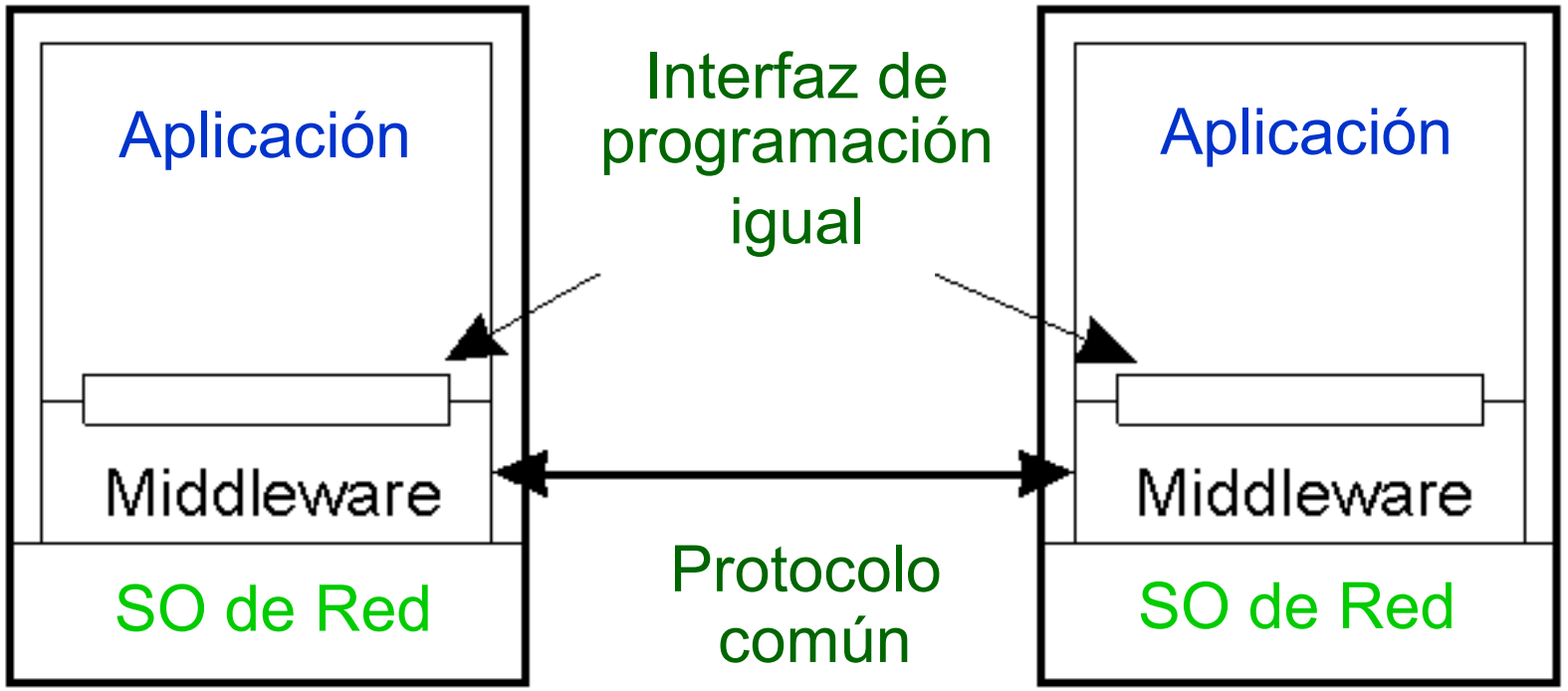
Sistemas Distribuidos: Conceptos de Software (6)

Posición del Middleware



Sistemas Distribuidos: Conceptos de Software (7)

Middleware y Apertura



En un sistema distribuido basado en middleware abierto, tanto los protocolos usados en cada capa como las interfaces que ofrecen a las aplicaciones deberían ser los mismos.

Sistemas Distribuidos: Conceptos de Software (8)

Comparación entre Sistemas

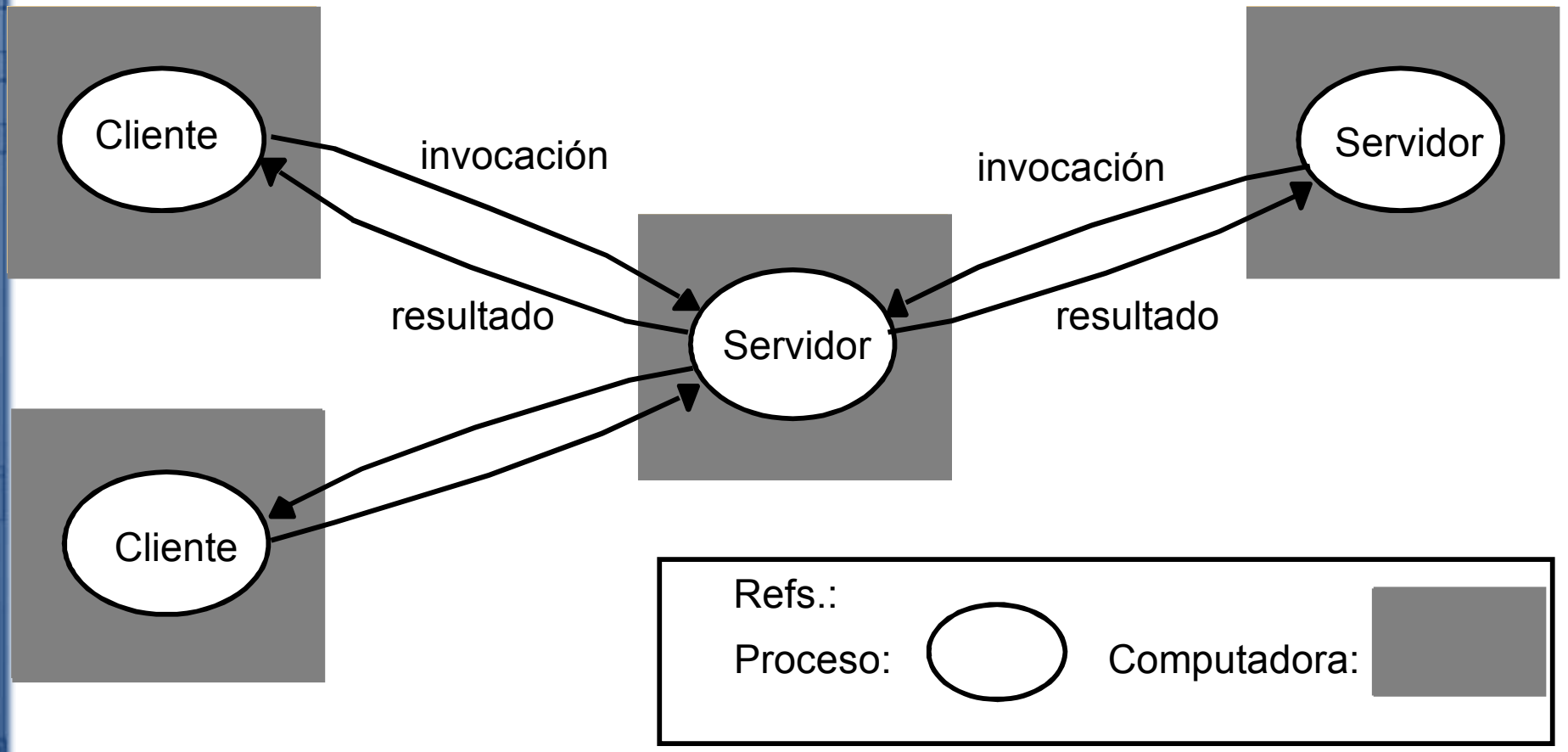
Item	SO Distribuido		SO de Red	SO basado en Middleware
	Multiproces.	Multicompu.		
Grado de transparencia	Muy alto	Alto	Bajo	Alto
Igual SO en todos los nodos	Si	Si	No	No
Número de copias de SO	1	N	N	N
Base para comunicaciones	Memoria compartida	Mensajes	Archivos	Modelo específico
Manejo de Recursos	Global, central	Global, distribuido	Por nodo	Por nodo
Escalabilidad	No	Moderada	Si	Varía
Apertura	Cerrado	Cerrado	Abierto	Abierto

Sistemas Distribuidos: Modelos de Sistemas (1)

Capas de Software y Hardware

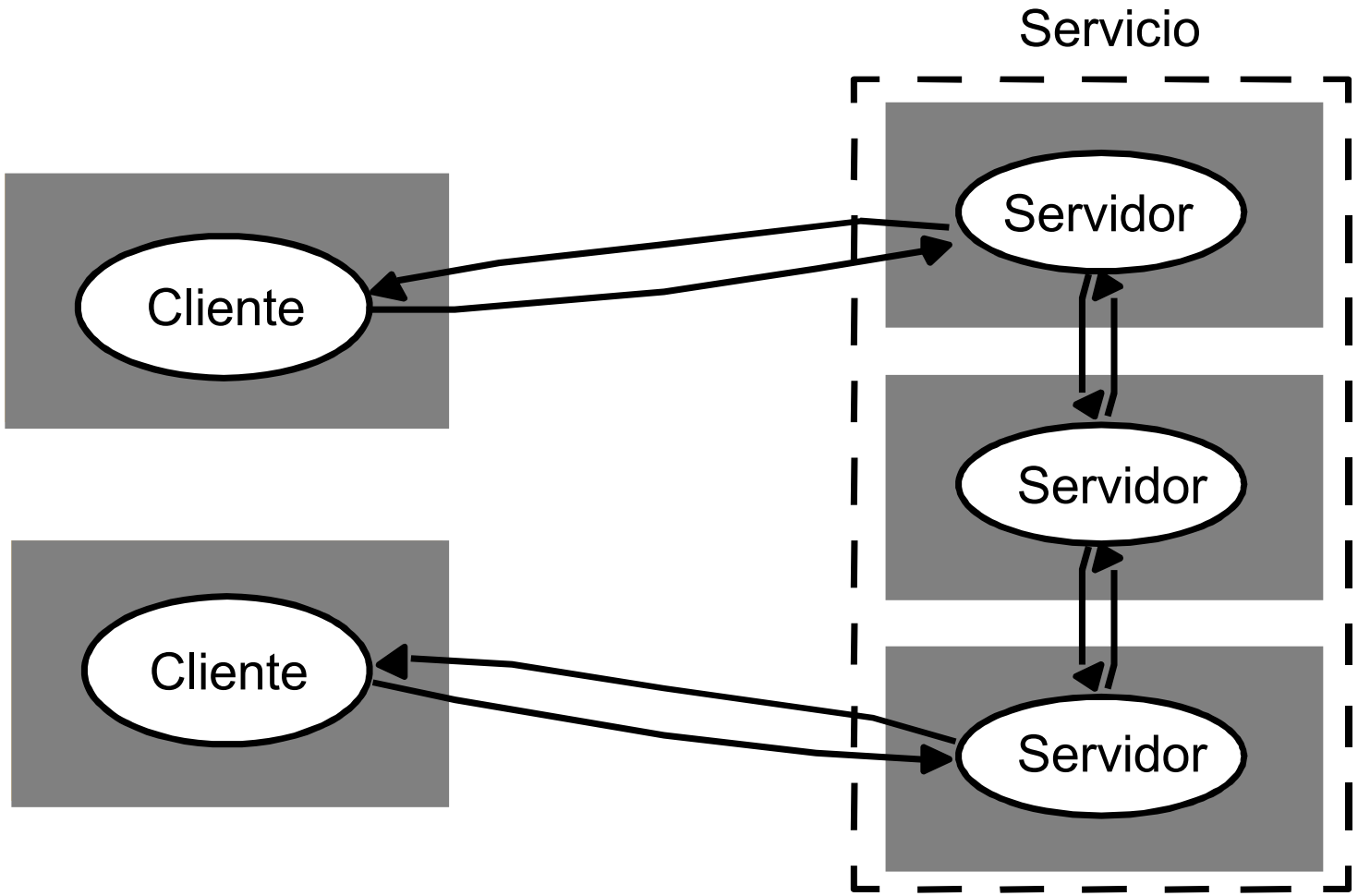
Sistemas Distribuidos: Modelos de Sistemas (2)

Modelo Cliente-Servidor



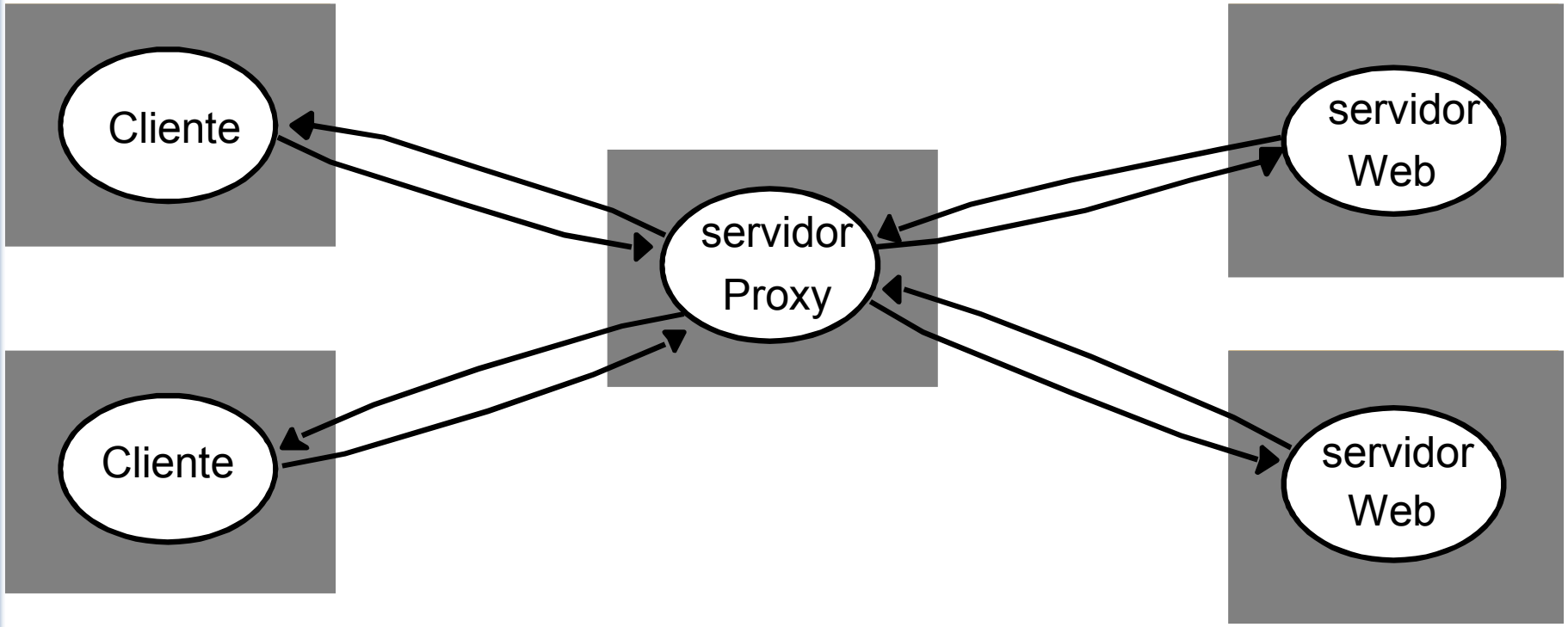
Sistemas Distribuidos: Modelos de Sistemas (3)

Servicio provisto por múltiples servidores



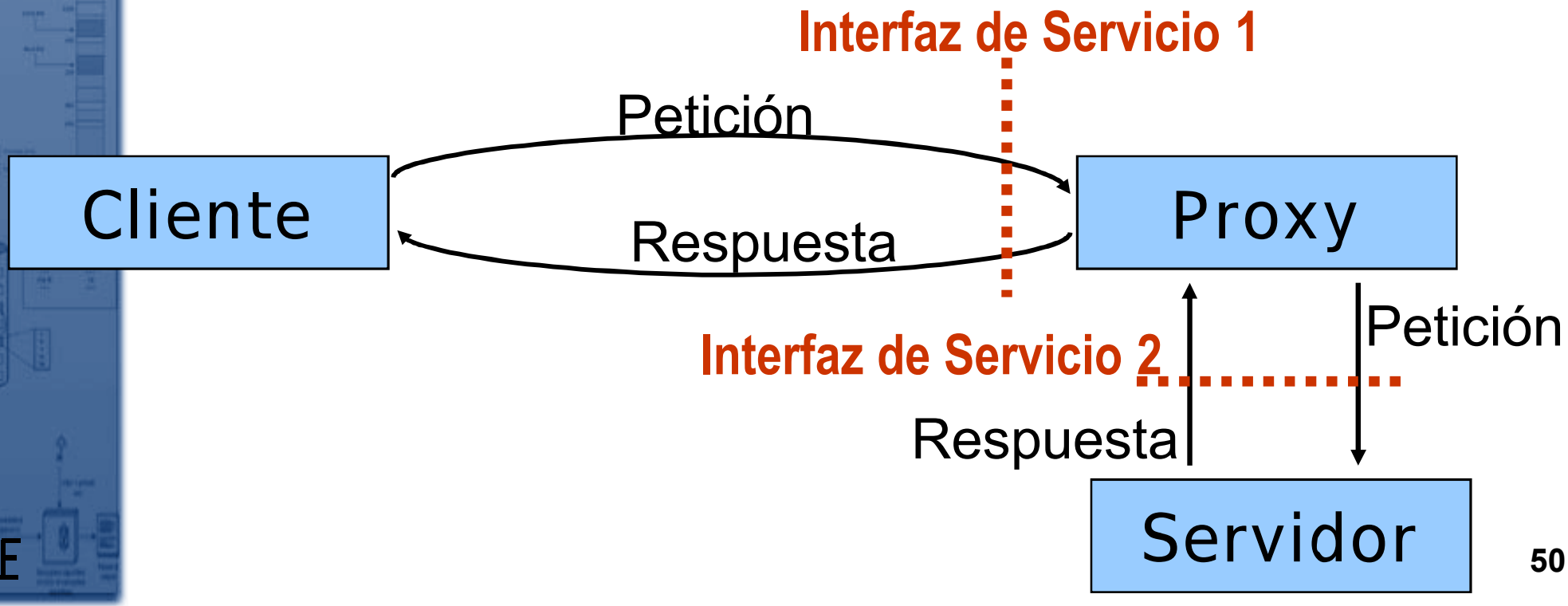
Sistemas distribuidos: Modelos de Sistemas (4)

Servidor Proxy



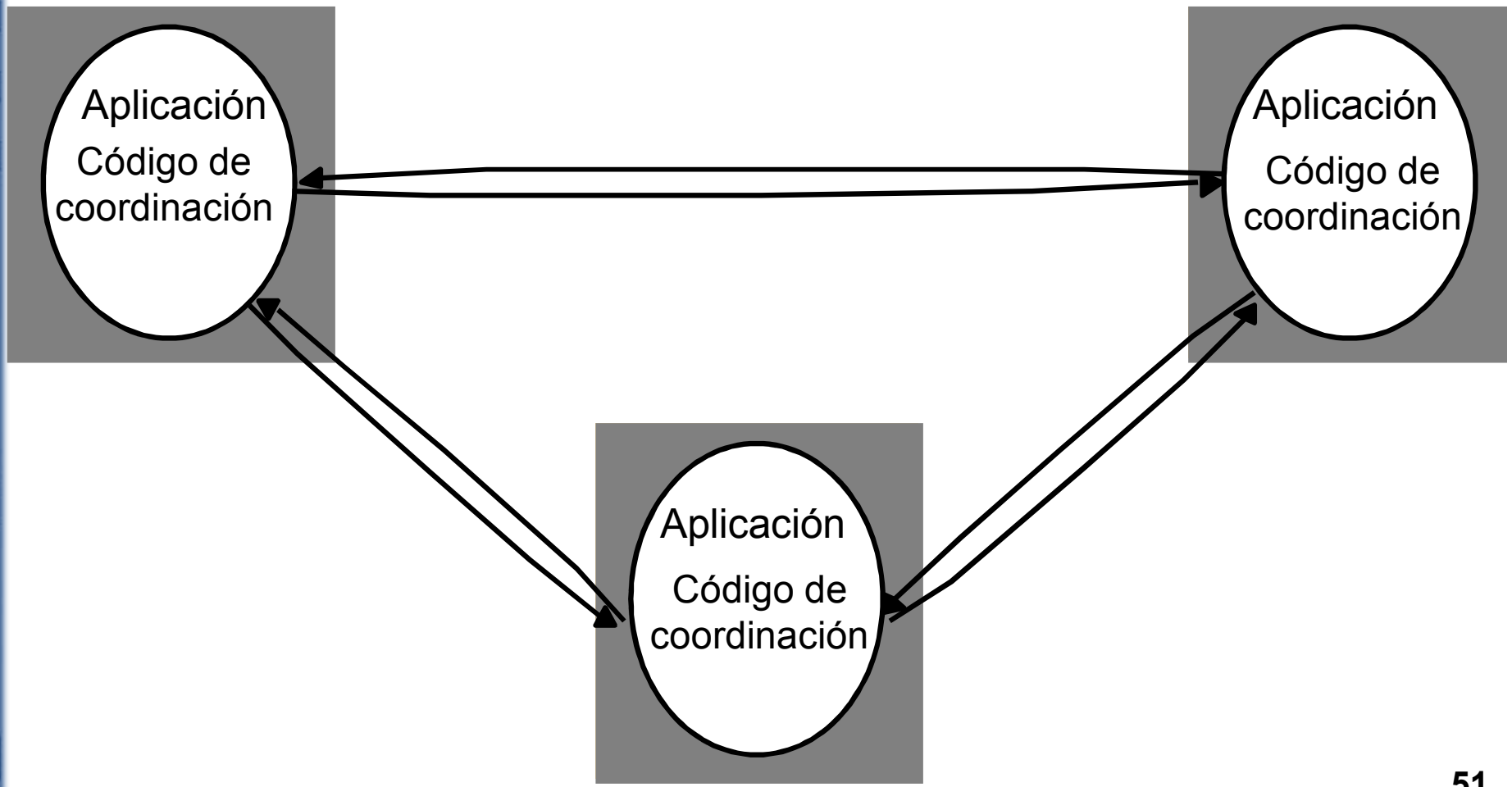
Sistemas distribuidos: Modelos de Sistemas (5)

- Tres roles diferentes en la interacción
  - Cliente: Solicita servicio.
  - Servidor: Proporciona servicio.
  - Proxy: Intermediario (si tiene *memoria* se denomina caché).



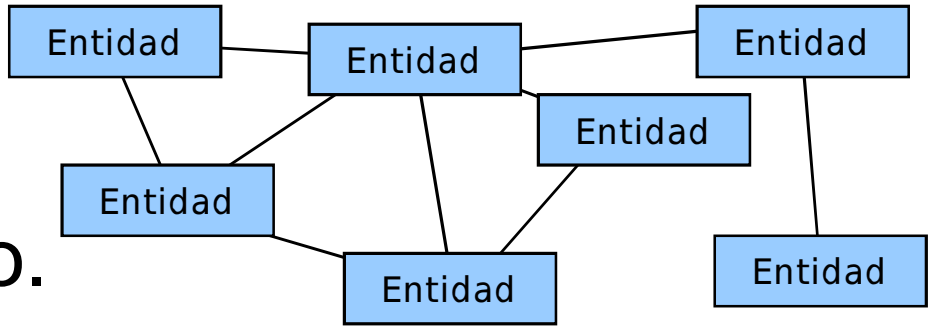
Sistemas distribuidos: Modelos de Sistemas (6)

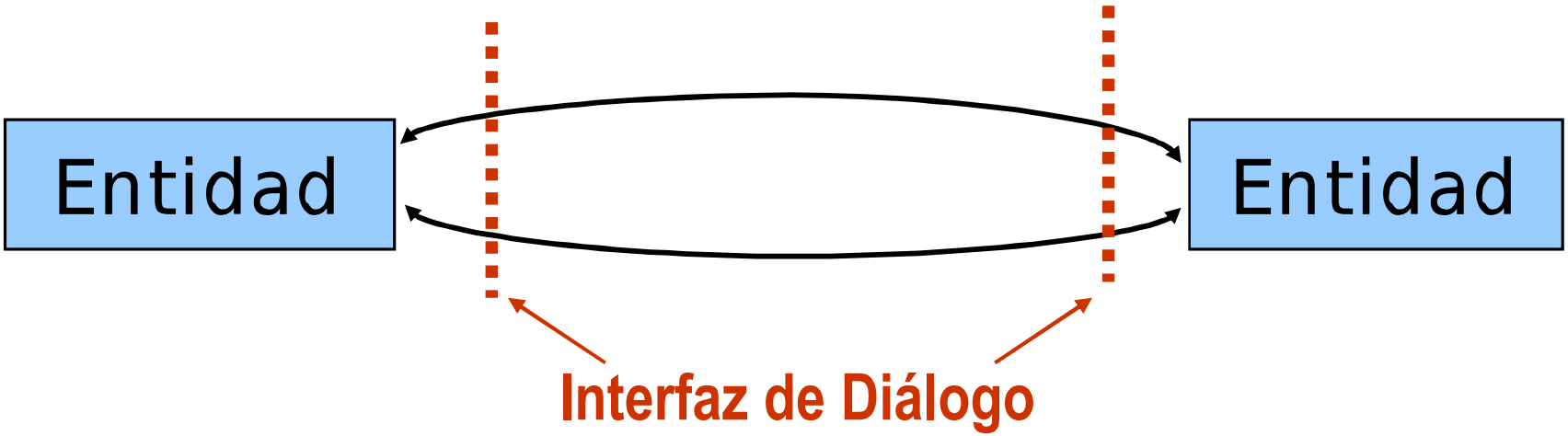
Aplicación basada en procesos *peers*



Sistemas distribuidos: Modelos de Sistemas (7)

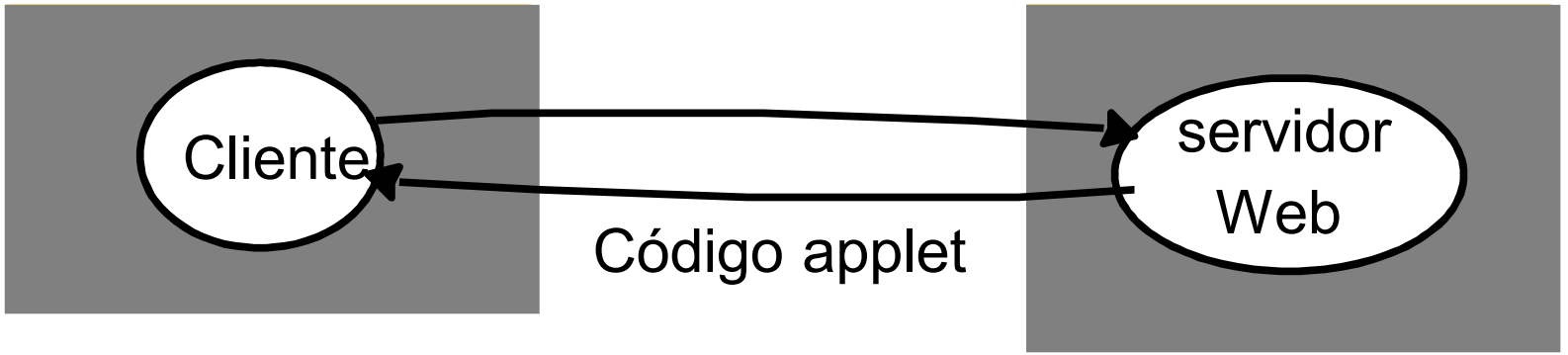
- Un único rol:
  - Entidad (*peer*).
- Protocolo de diálogo.
  - Primitivas de interacción.





Sistemas distribuidos: Modelos de Sistemas (8)

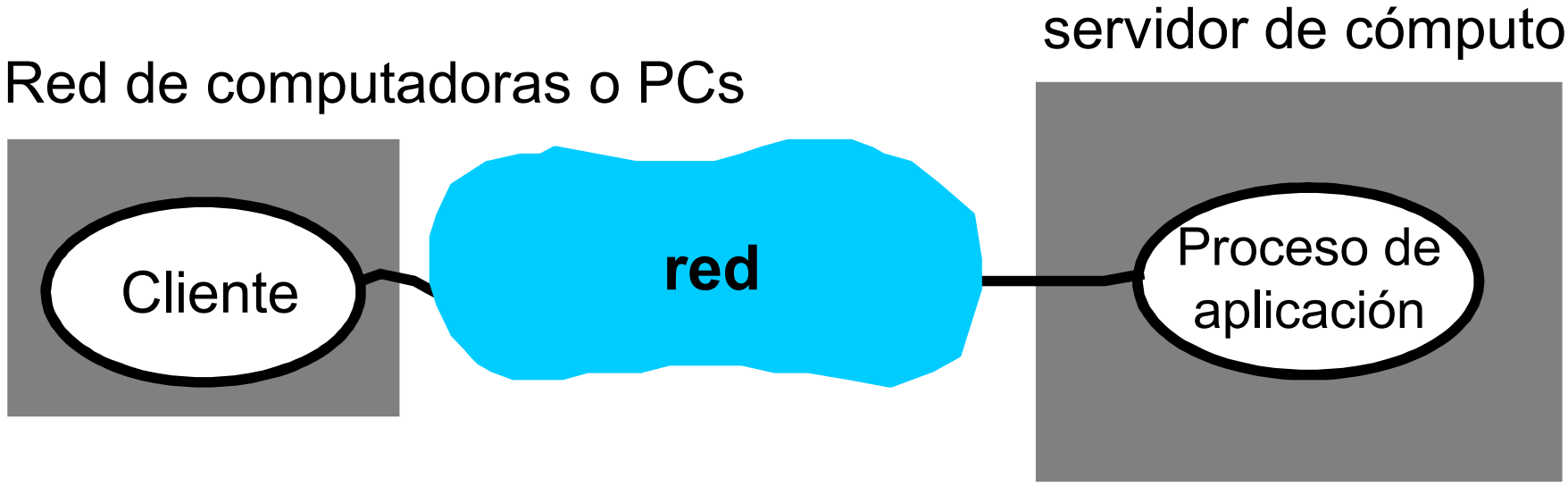
a) El requerimiento del cliente resulta en la bajada de un código applet .



b) El cliente interactúa con el applet.

Sistemas distribuidos: Modelos de Sistemas (9)

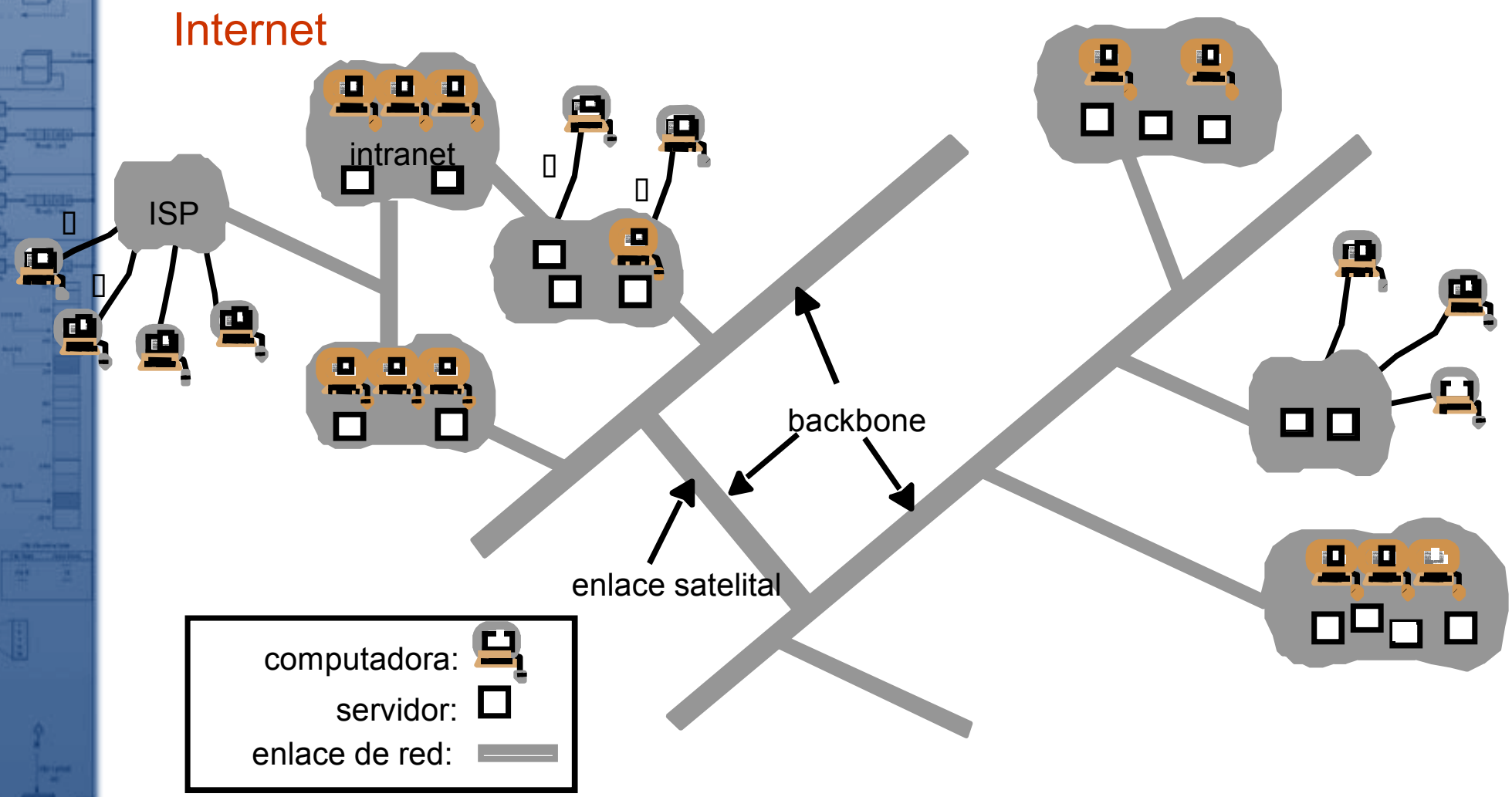
Clientes y servicios de cómputo



Introducción a los Sistemas Distribuidos

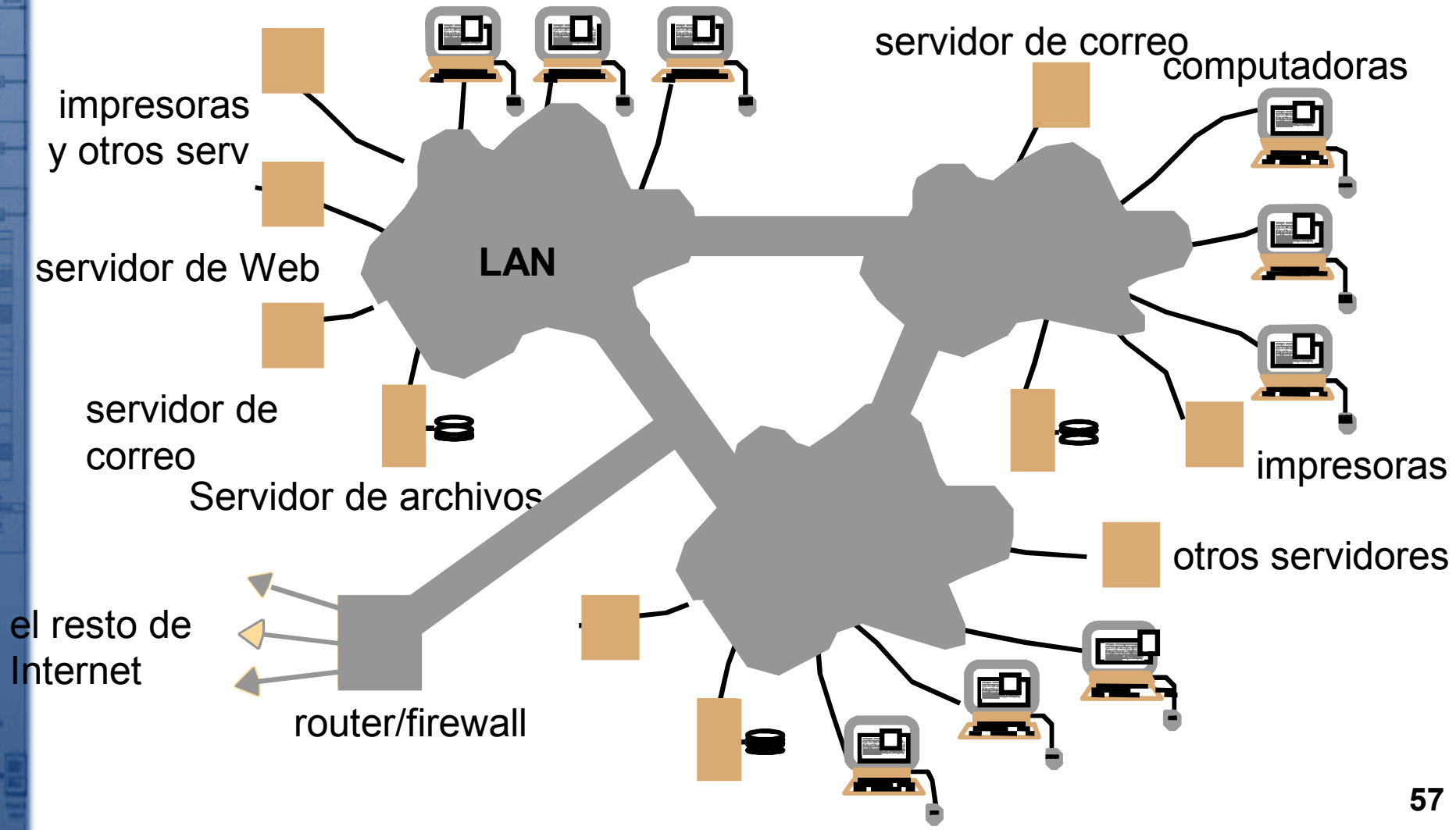
Ejemplos de Sistemas Distribuidos

Sistemas distribuidos: Ejemplos



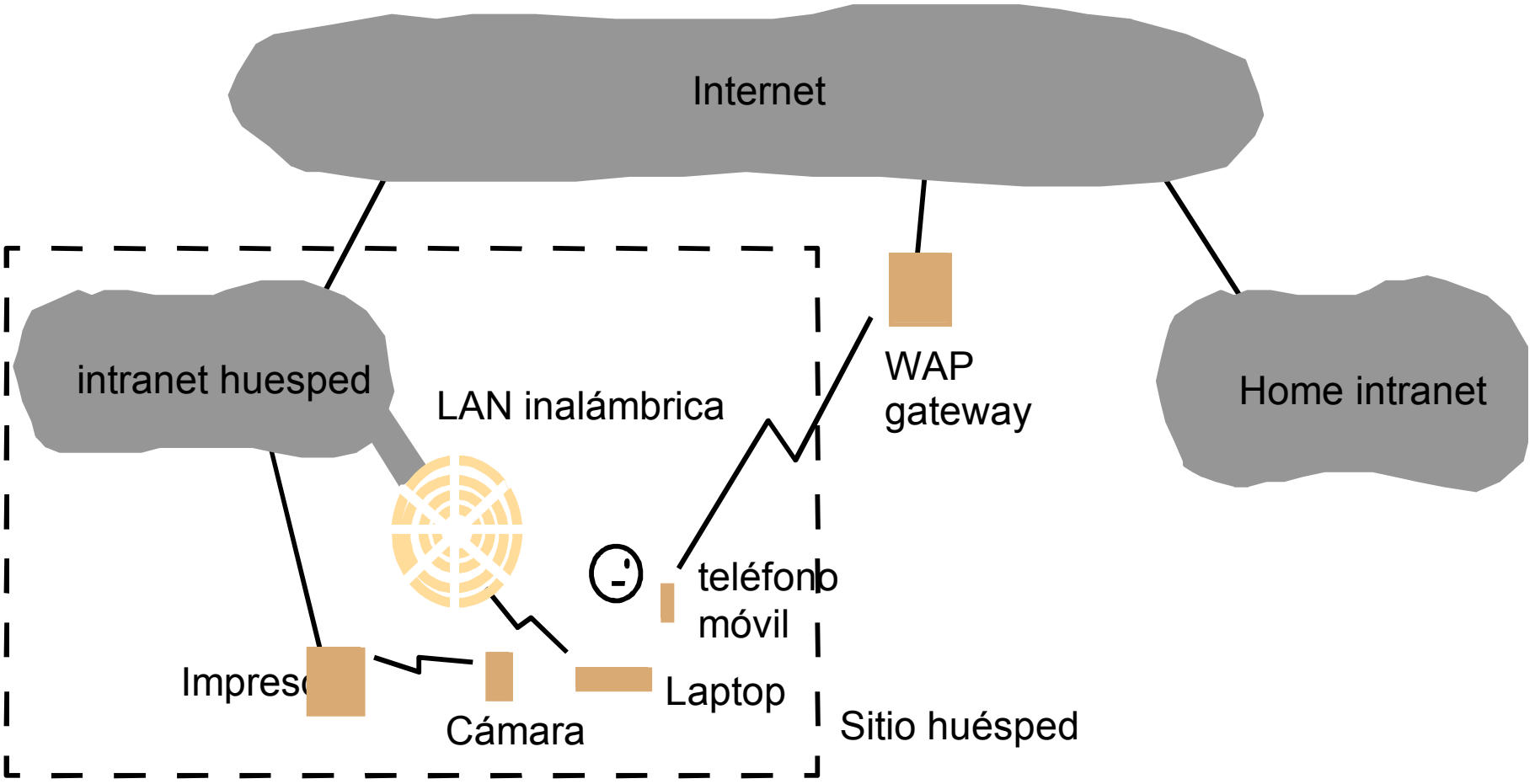
Sistemas distribuidos: Ejemplos

Una intranet típica



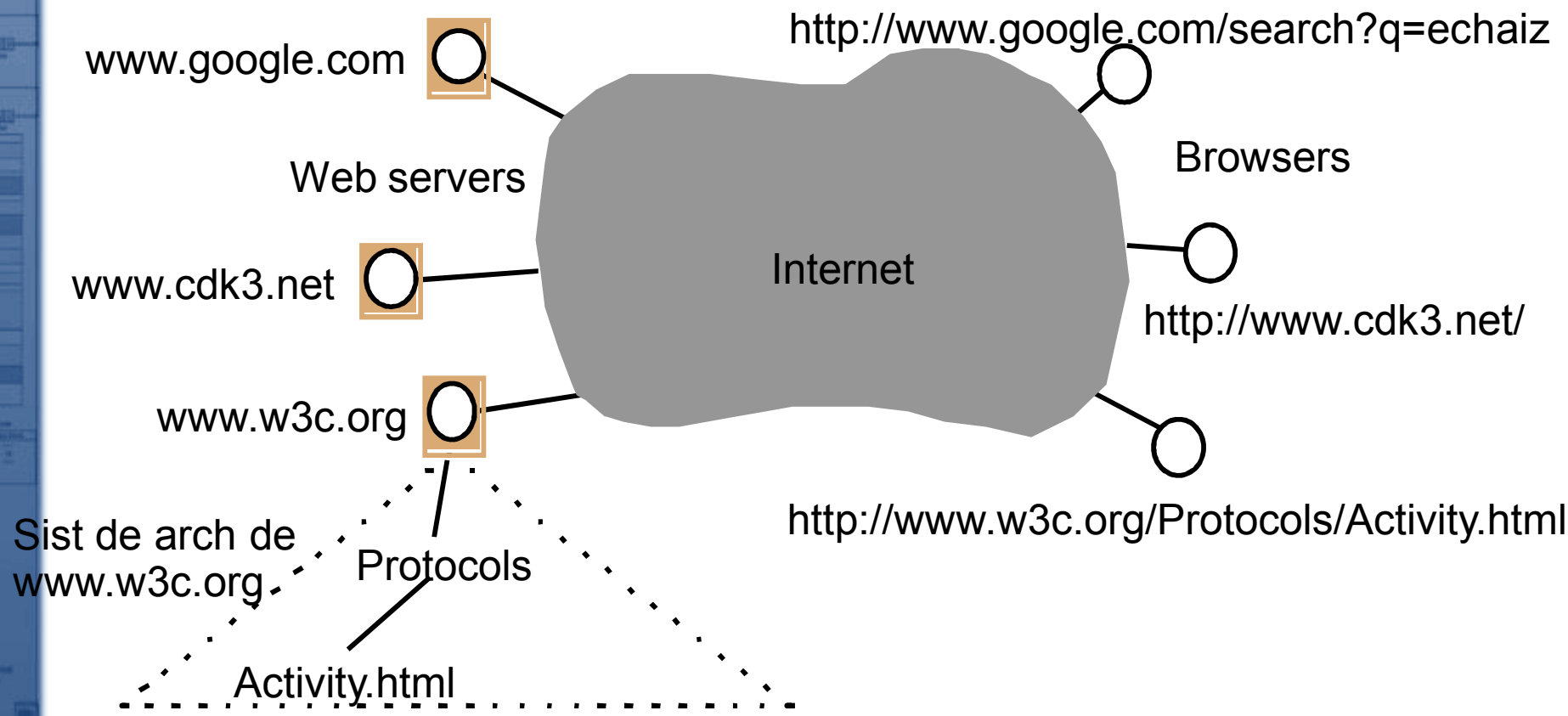
Sistemas distribuidos: Ejemplos

Dispositivos portables y manuales en un sistema distribuido



Sistemas distribuidos: Ejemplos

Servidores de Web y navegadores de Web



Coming Next