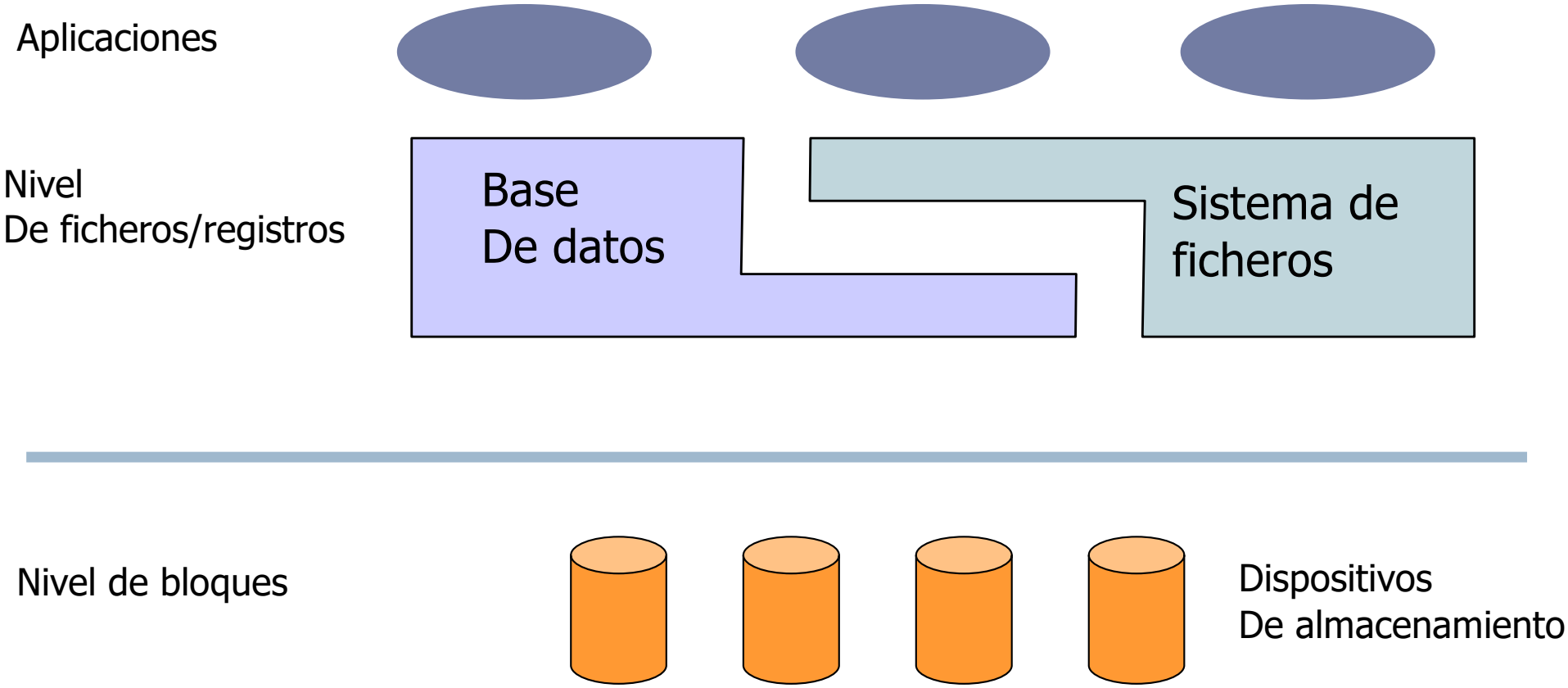
Tema 7
Sistemas de ficheros distribuidos

Sistemas Distribuidos
Grado en Ingeniería Informática
Universidad Carlos III de Madrid

Contenido

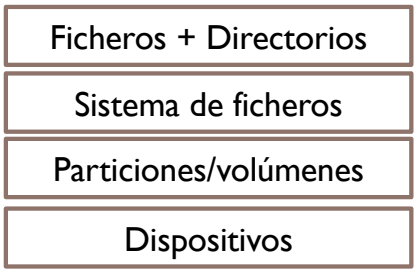
- Conceptos básicos
- Estructura de un sistema de ficheros distribuido
- Servicio de directorio
- Servicio de ficheros
- Implementación
  - Semántica de coutilización
  - Métodos de acceso
  - Caché y coherencia de caché
- Ejemplos de sistemas de ficheros distribuidos
- Sistemas de ficheros paralelos
- Redes de almacenamiento

Modelo de almacenamiento

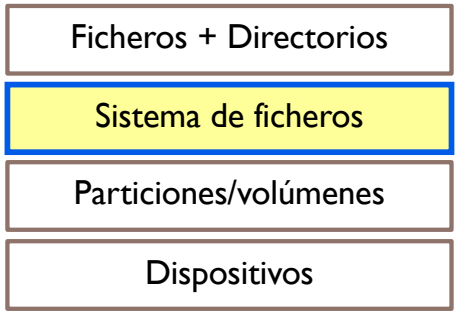


Sistemas de ficheros

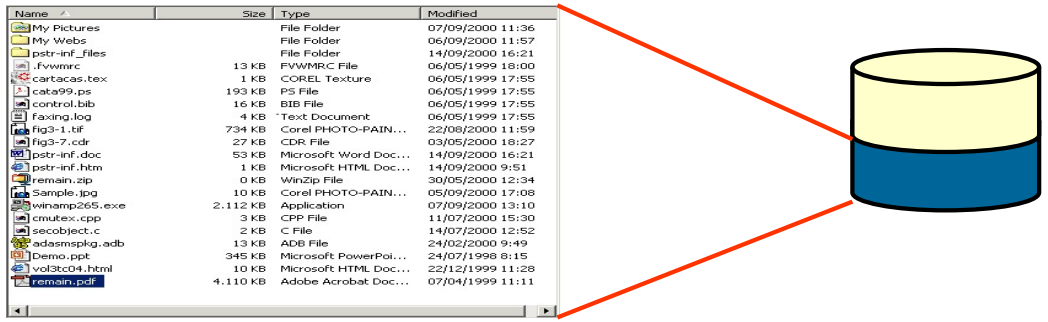
- Un sistema de ficheros es un SW de sistemas que establece una correspondencia lógica entre los ficheros y directorios y los dispositivos de almacenamiento
- Funciones:
  - Organización, almacenamiento, recuperación, gestión de nombres, coutilización y protección de los ficheros
  - Ofrece un mecanismo de abstracción que oculta todos los detalles relacionados con el almacenamiento y distribución de la información en los dispositivos, así como el funcionamiento de los mismos.



Sistema de ficheros

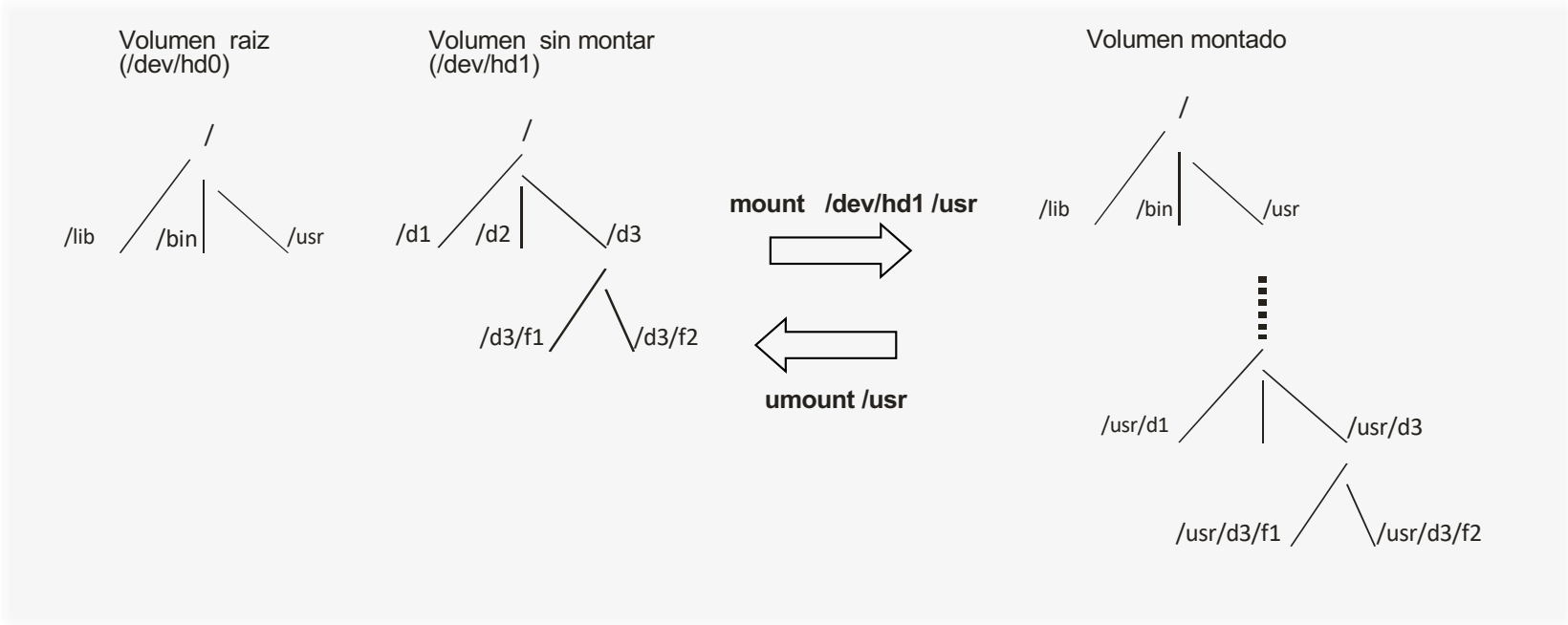


- **Exporta** una jerarquía de ficheros y directorios con un conjunto de restricciones y métodos de acceso
- Ofrece una semántica e interfaz de acceso
- Coordina el acceso a los datos y metadatos desde varias peticiones
- Gestiona la utilización de los discos
- Gestiona y mantiene la integridad de los datos

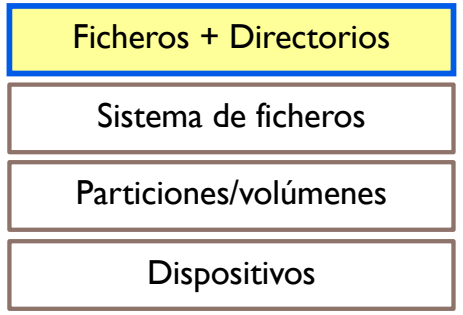


Operaciones sobre el sistema de ficheros

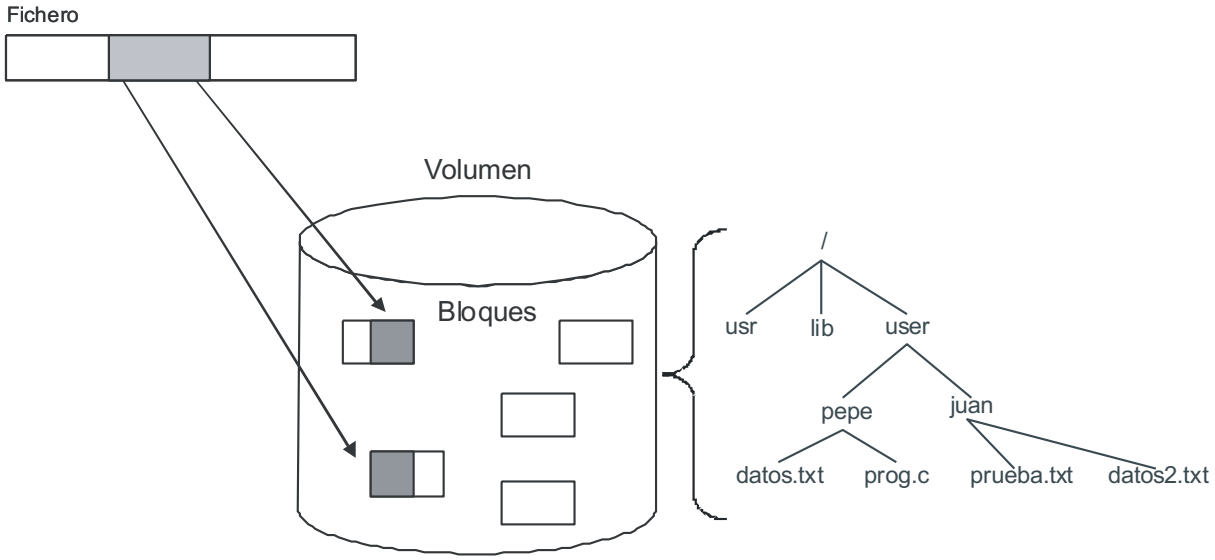
- Operaciones con sistemas de ficheros:
  - Crear
  - Montar
  - Desmontar



Ficheros y directorios



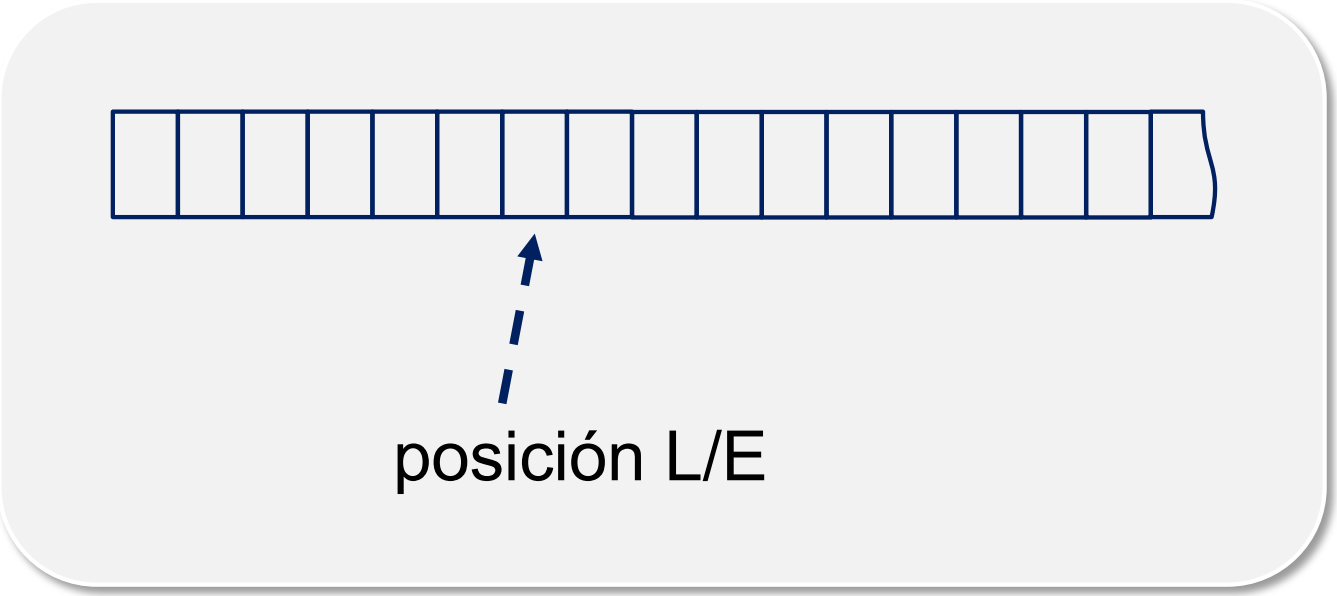
- **Ficheros**: datos + atributos
- **Directorio**: relaciona nombres con ficheros



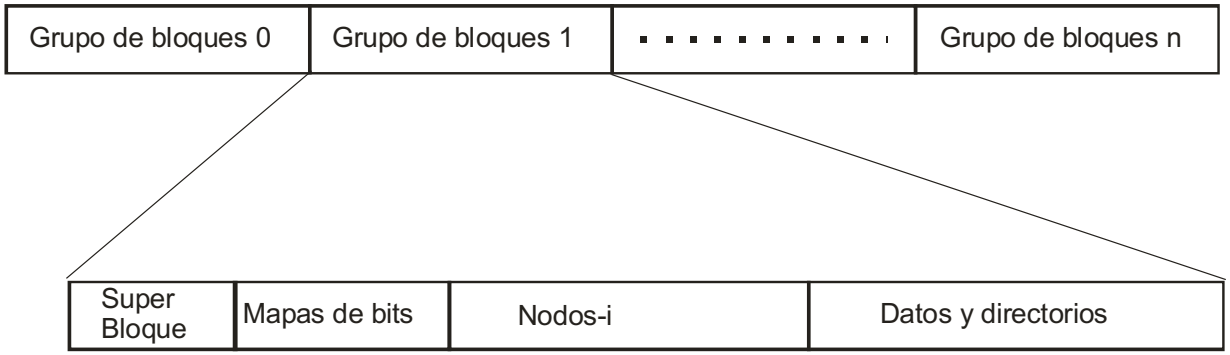
- **Metadatos**: información almacenada por el sistema de ficheros para la gestión de ficheros y directorios

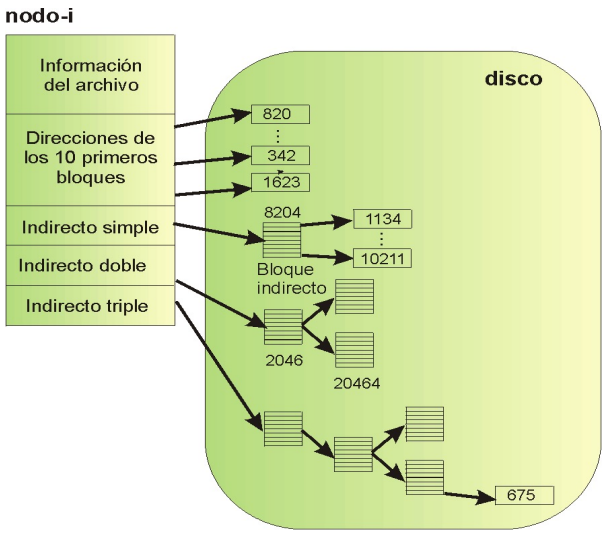
Abstracción típica de un fichero

- Habitualmente el contenido es representado por una secuencia o tira de bytes



Ejemplo de estructura de un sistema de ficheros y metadatos



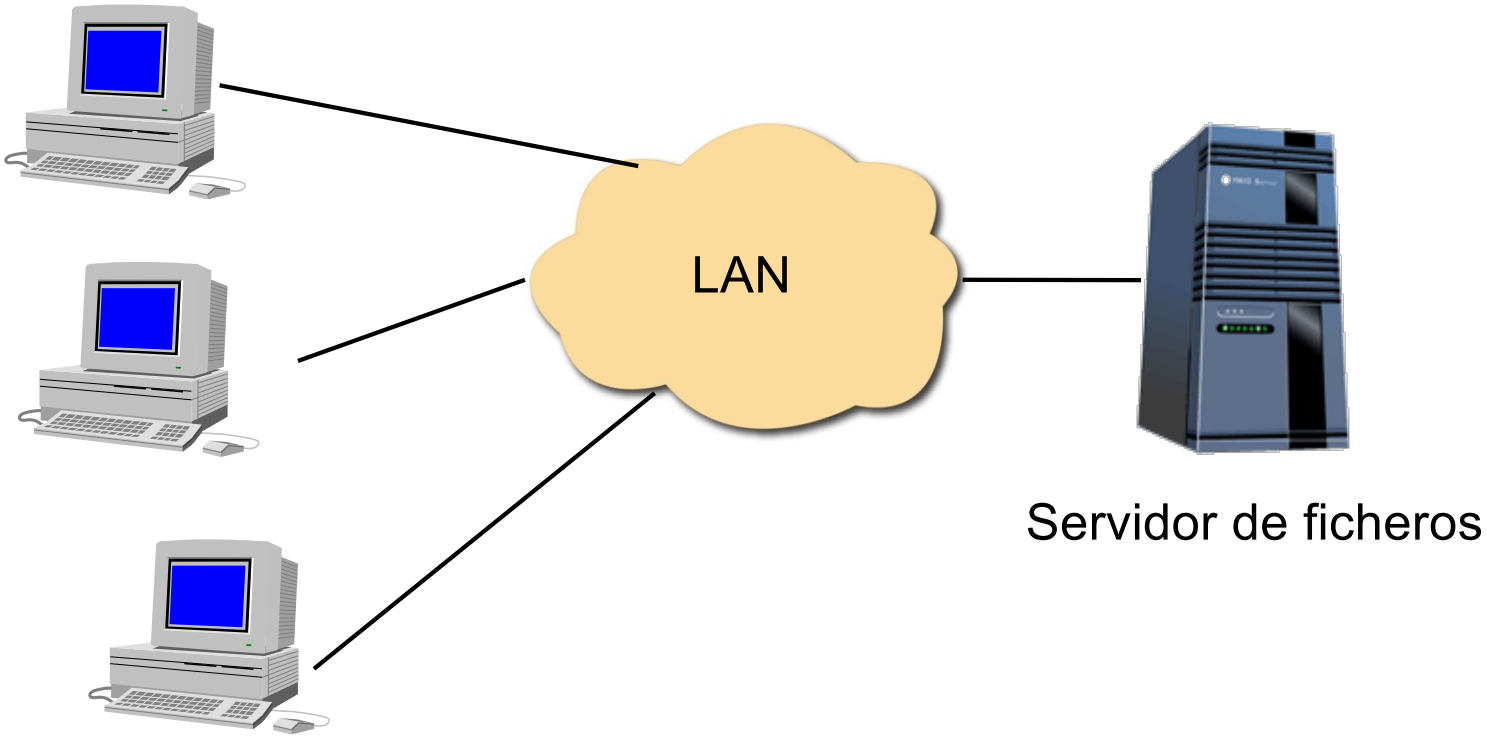


Tipos de sistemas de ficheros

- Sistemas de ficheros locales
  - UFS, QFS, VXFS, JFS, JFS2, FAT, NTFS, EXT3, EXT2, Reiser,…
- Sistemas de ficheros distribuidos
  - NFS, CIFS, …
- Sistemas de ficheros paralelos
  - PFS, GPFS
- Sistemas de ficheros para discos compartidos (SAN y Clusters)

Sistemas de ficheros distribuidos

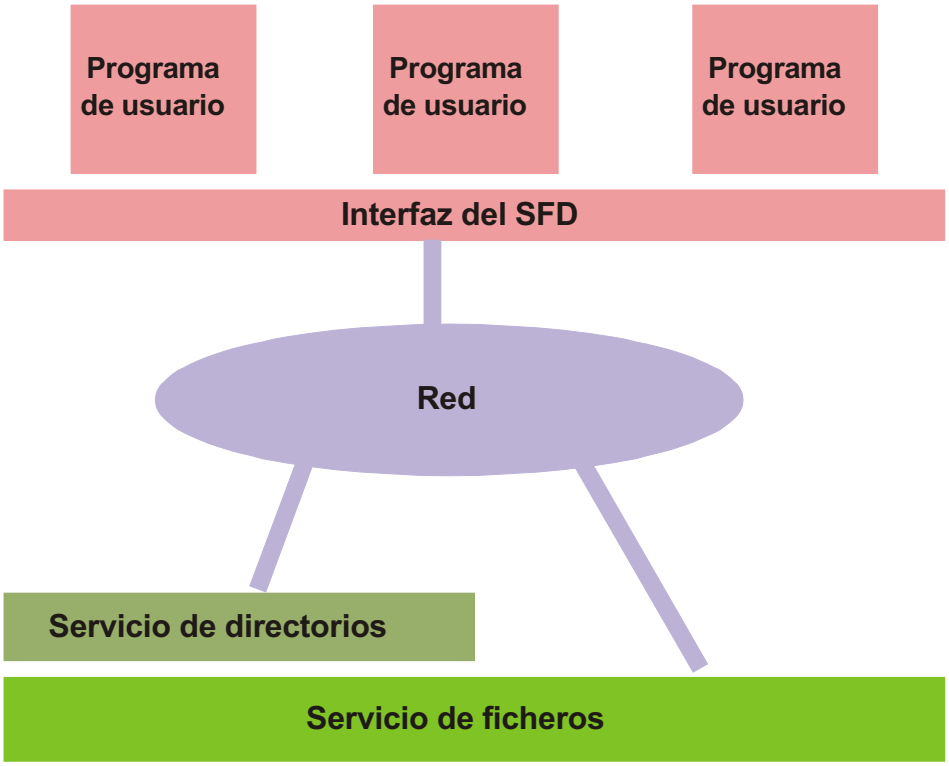
- Ofrece **acceso** a **ficheros remotos** de forma transparente
- Permite **compartir datos** entre usuarios



Requisitos de un sistema de ficheros distribuido

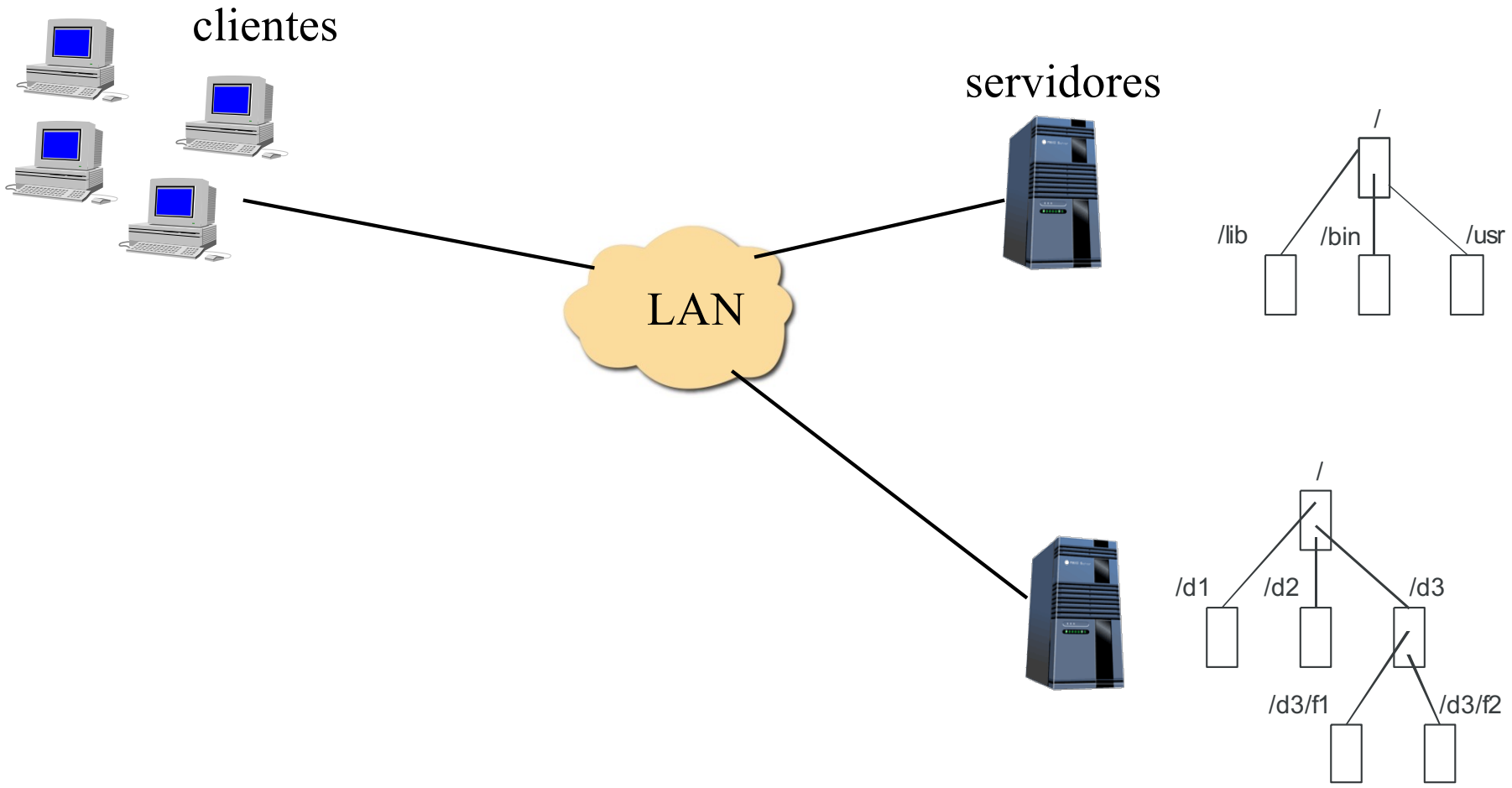
- Transparencia
  - Mismas operaciones para acceso locales y remotos
  - Imagen única del sistema de ficheros
- Eficiencia. Un SFD tiene sobrecargas adicionales
  - Red de comunicación, protocolos, posible necesidad de realizar más copias, etc.
- Tolerancia a fallos:
  - Replicación, funcionamiento degradado, etc.
- Facilidad de crecimiento (escalabilidad)
  - Eliminar los cuellos de botella
- Consistencia
- Actualizaciones concurrentes
- Seguridad

Componentes de un SFD

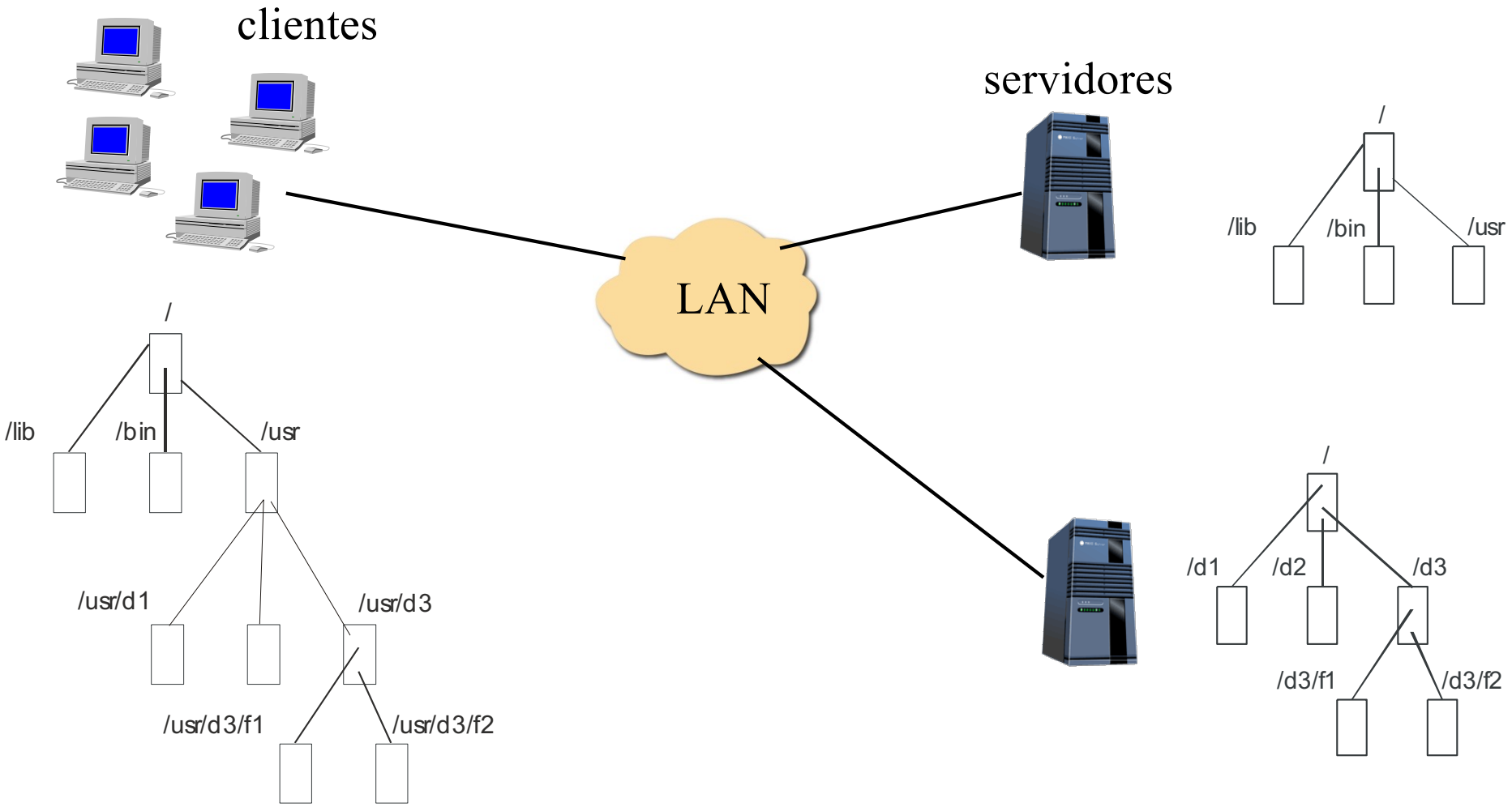


- **Modelo cliente-servidor**
  - Servicios del sistema de ficheros. Operaciones proporcionadas a los clientes
    - Convierten los accesos de las aplicaciones en peticiones al servidor de ficheros
  - Servidores del sistema de ficheros. Procesos de usuario o del sistema que ofrecen los servicios correspondientes y responden a las peticiones de los clientes (servidores *multithread*)

Empleo de múltiples servidores



Empleo de múltiples servidores



Servicio de directorio

- Se encarga de la **traducción** de nombres de usuario a Identificadores de ficheros únicos (UFID)
  - Ej: En UNIX/Linux: fichero a nodo-i
- **Directorio**: relaciona de forma única nombres de fichero con UFID
  - Los UFID permiten obtener los atributos de los ficheros
- Dos opciones:
  - Los directorios son objetos independientes gestionados por un servidor de directorios (SD)
  - Los directorios son ficheros especiales. Servidor de ficheros y de directorios combinados

Gestión de nombres: principios básicos

- Transparencia de la posición: el nombre del objeto no permite obtener directamente el lugar donde esta almacenado
- Independencia de la posición: el nombre no necesita ser cambiado cuando el objeto cambia de lugar
  - Asociación entre nombre y posición dinámica
  - Propiedad más exigente que la transparencia
- Facilidad de crecimiento
- Replicación
- Nombres orientados al usuario

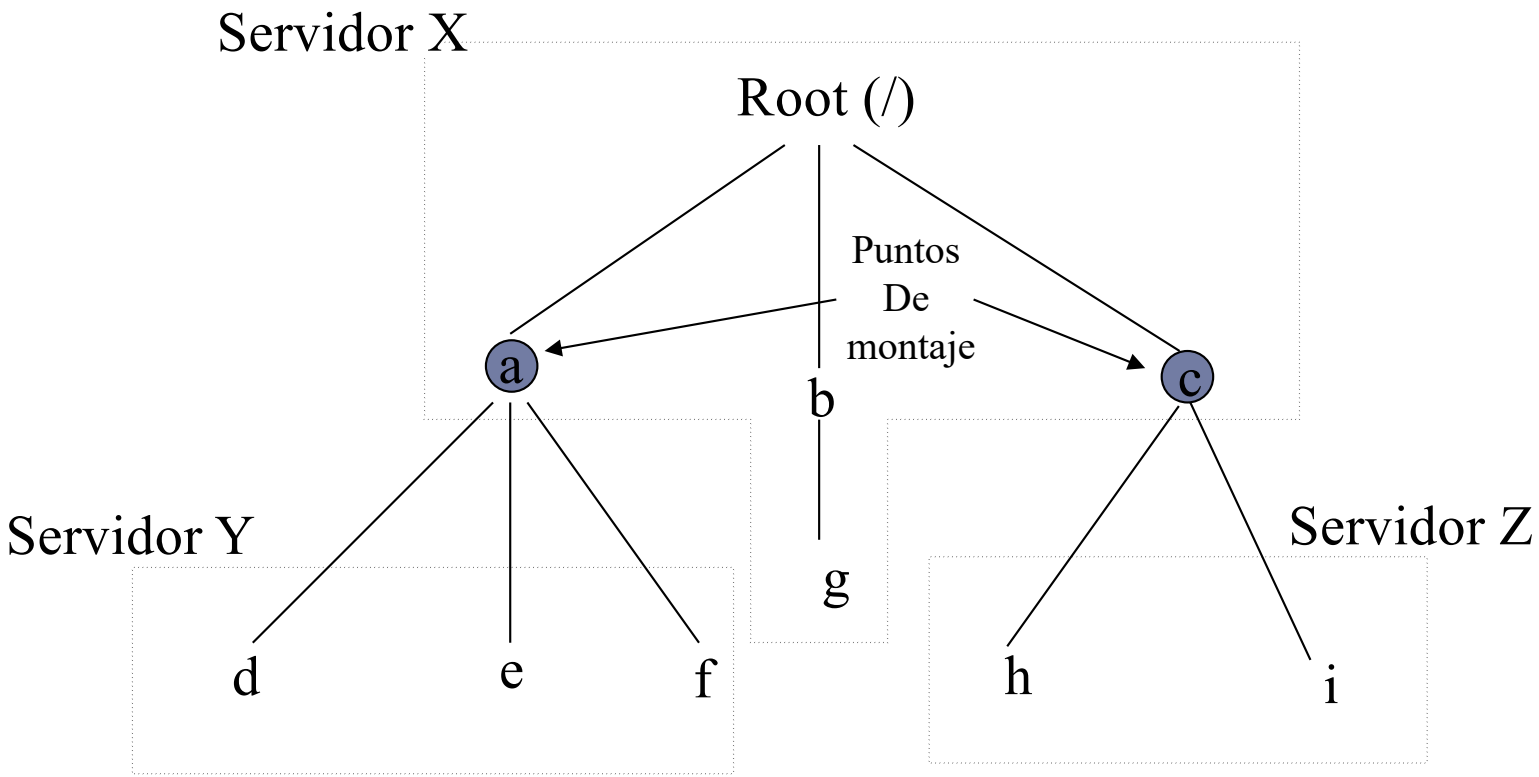
Nombrado de dos niveles

- Nombres de usuario
  - Generalmente el espacio de nombres es jerárquico
  - Tres alternativas
    - Máquina: nombre de fichero
      - Ni transparencia, ni independencia
    - Montar un sistema de ficheros remoto sobre la jerarquía local (NFS)
      - Espacio de nombres diferente en cada máquina
    - Único espacio de nombres en todas las máquinas
      - Proporciona transparencia
- Identificadores de ficheros:
  - Identificador único de fichero utilizado por el sistema

Operaciones básicas de un servicio de directorios

- Lookup(dir, name) → FileId
  - Busca un nombre en un directorio
- AddName(dir, name, FileId)
  - Añade un nombre (name, FileId) a un directorio
- RemoveName(dir, name)
  - Elimina una nombre de un directorio
- GetNames(dir) → ListName
  - Devuelve los nombre de un directorio

Espacio de nombres jerárquico



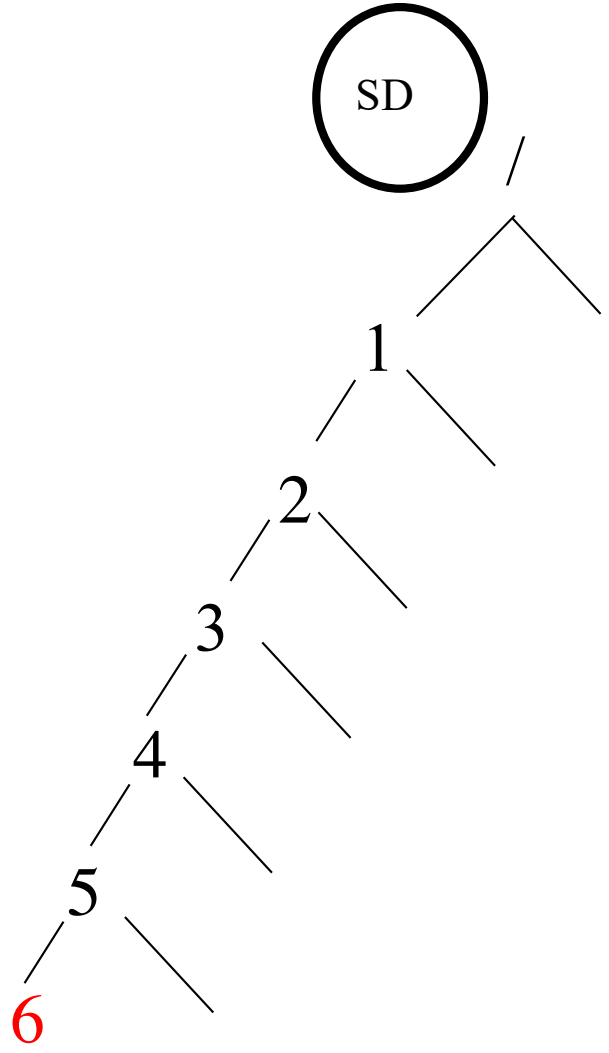
Resolución de nombres

- Dirigida por los clientes: NFS
- Dirigida por los servidores
  - **Resolución iterativa**
    - El cliente envía el nombre al SD
    - El SD realiza la traducción hasta que termina en un componente que pertenece a otro SD
    - El SD envía el resultado al cliente, el cual si no ha terminado la traducción continúa con el SD correspondiente
  - **Resolución transitiva**
    - Los SD implicados contactan entre si para llevar a cabo la traducción. El último SD devuelve la traducción al cliente
    - Rompe el modelo cliente/servidor (no adecuado para RPC)
  - **Resolución recursiva**
    - El último SD implicado devuelve el resultado al anterior y así sucesivamente hasta que el primero responde al cliente

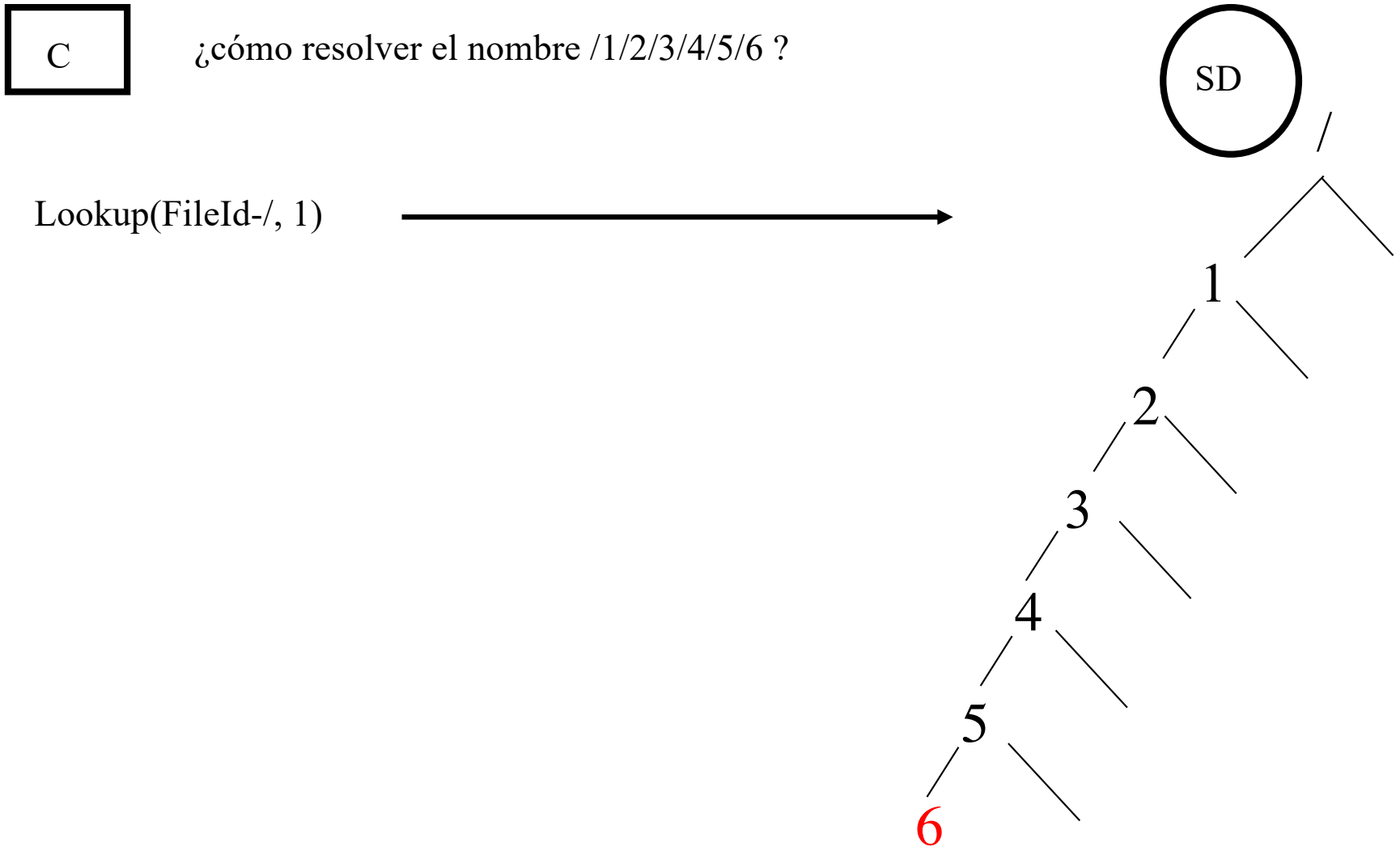
Dirigida por los clientes

C

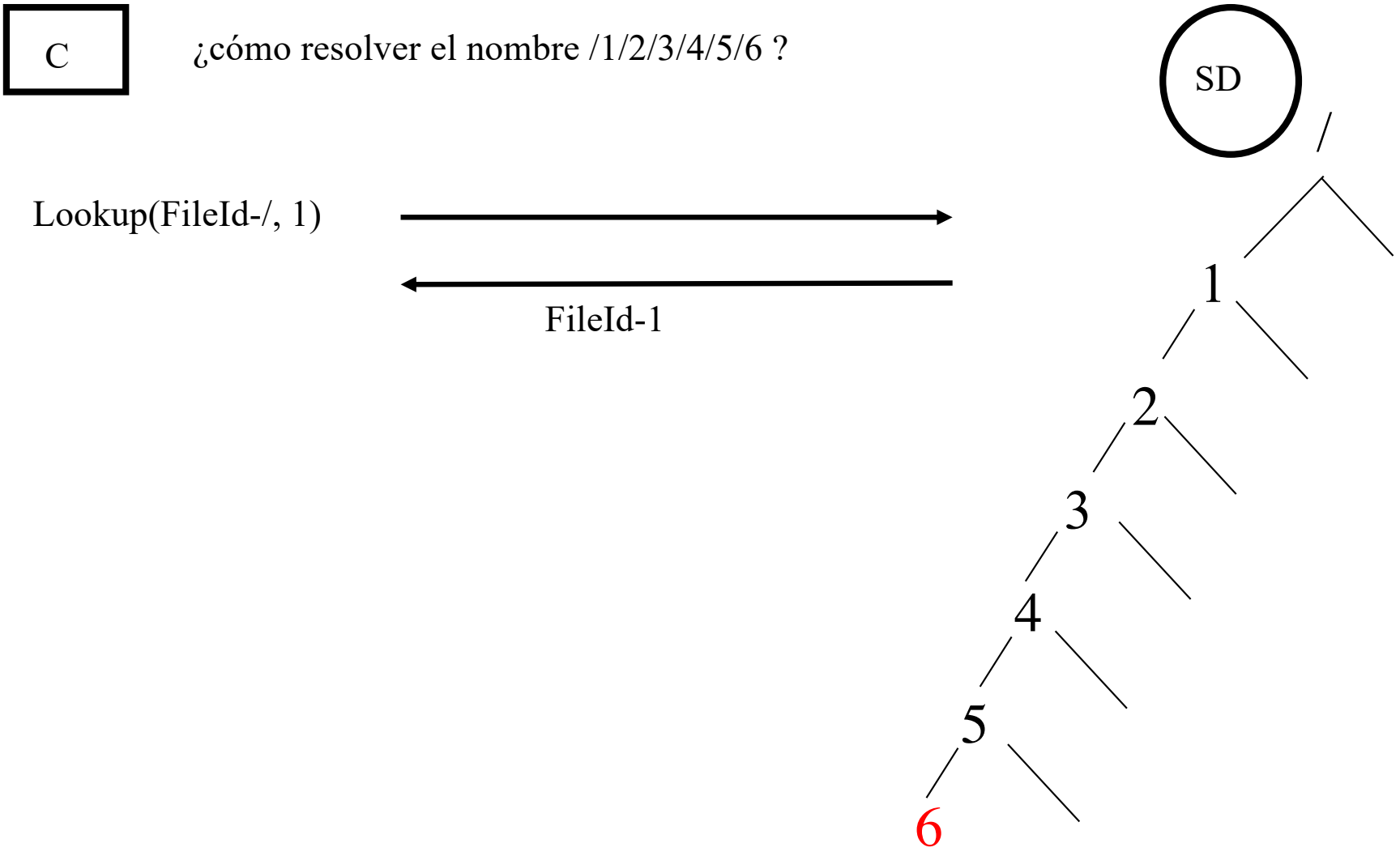
¿cómo resolver el nombre /1/2/3/4/5/6 ?



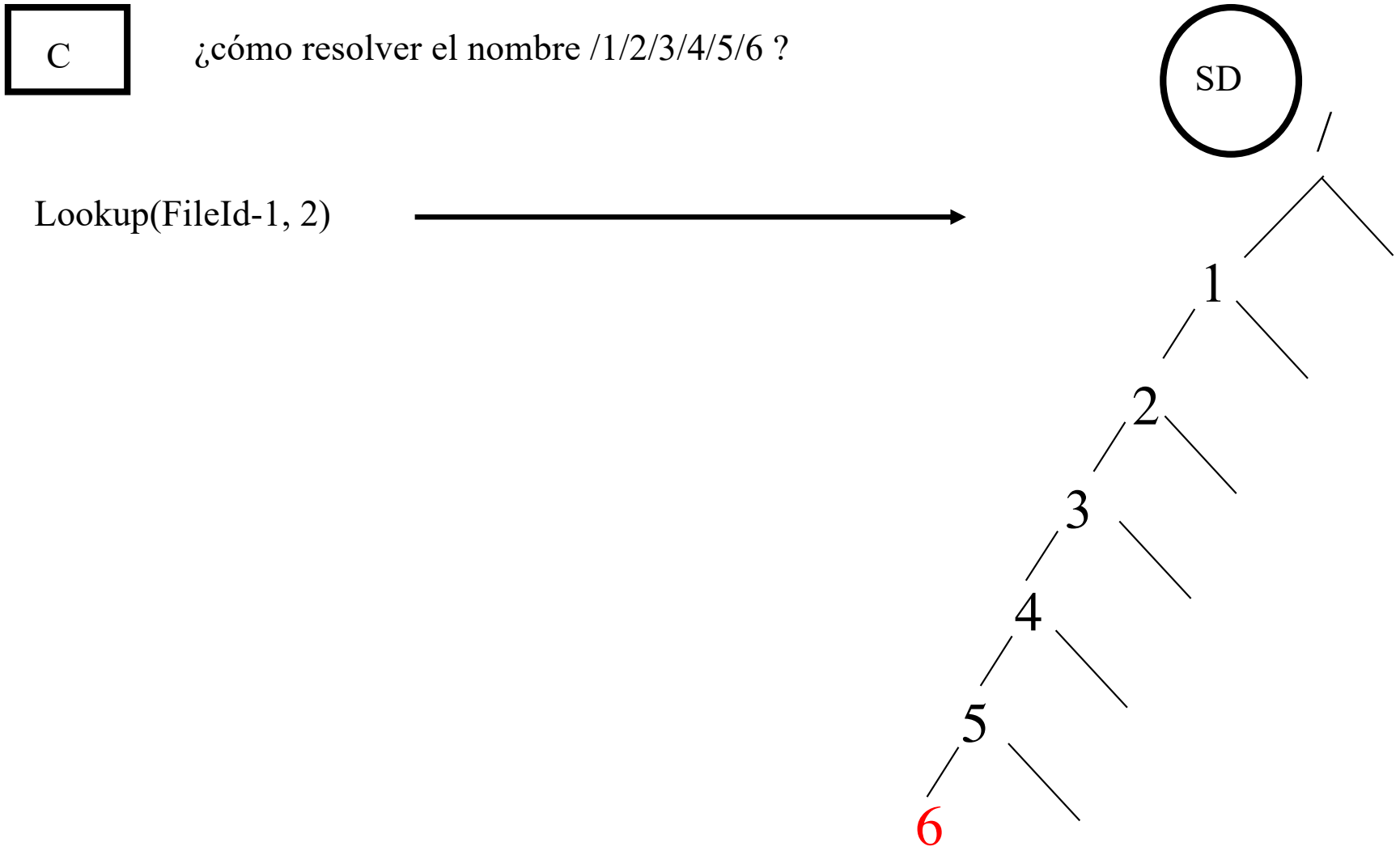
Dirigida por los clientes



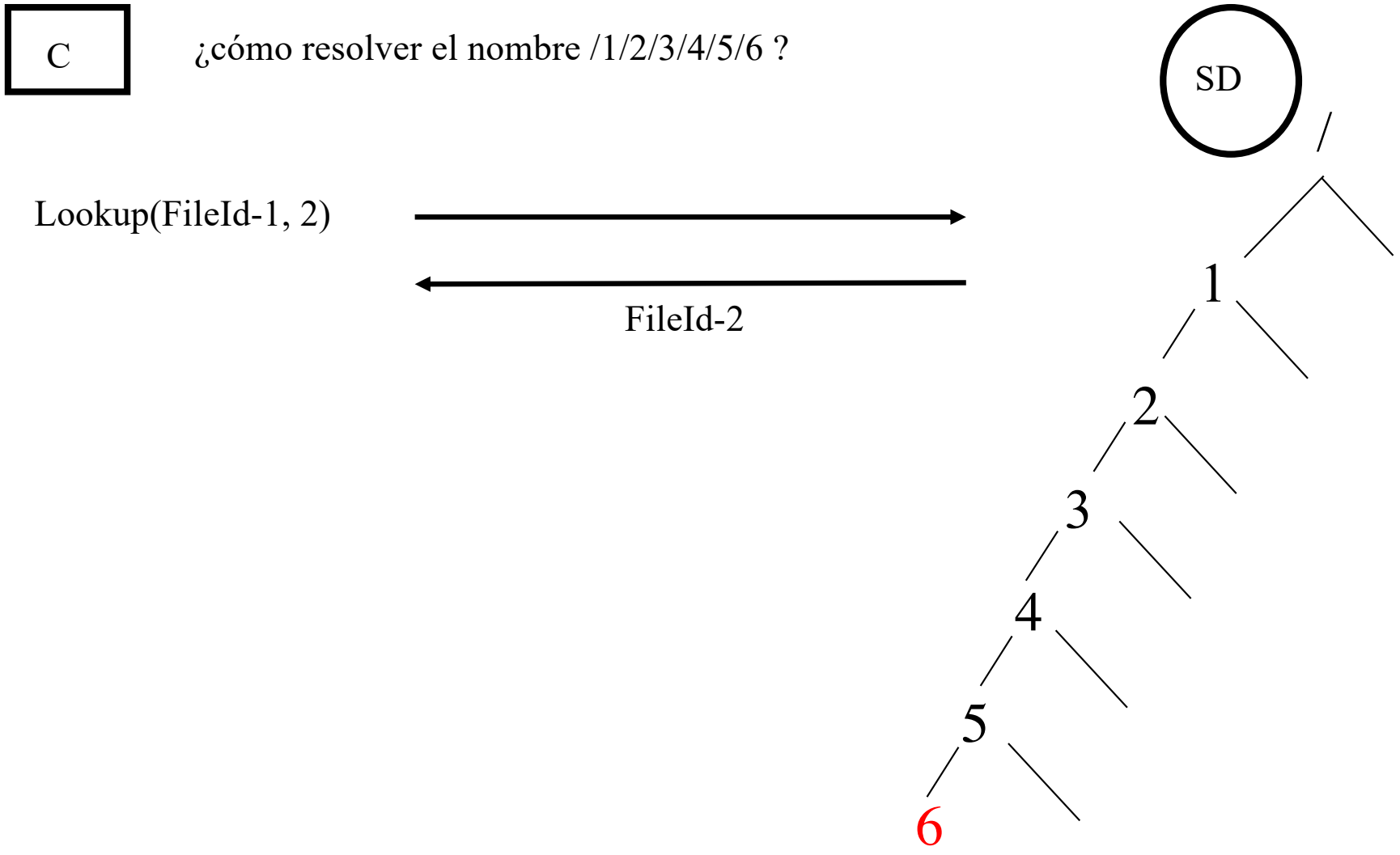
Dirigida por los clientes



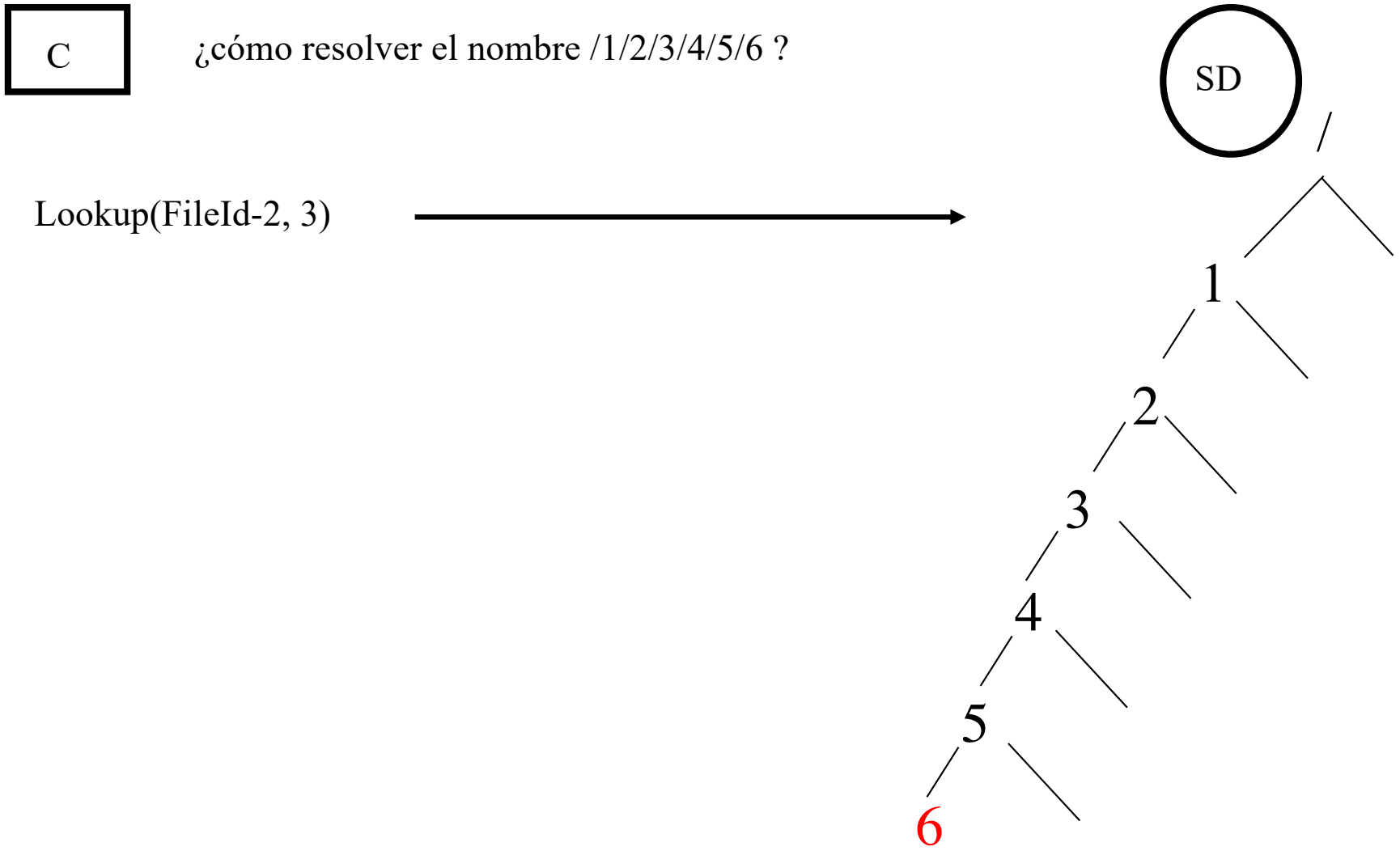
Dirigida por los clientes



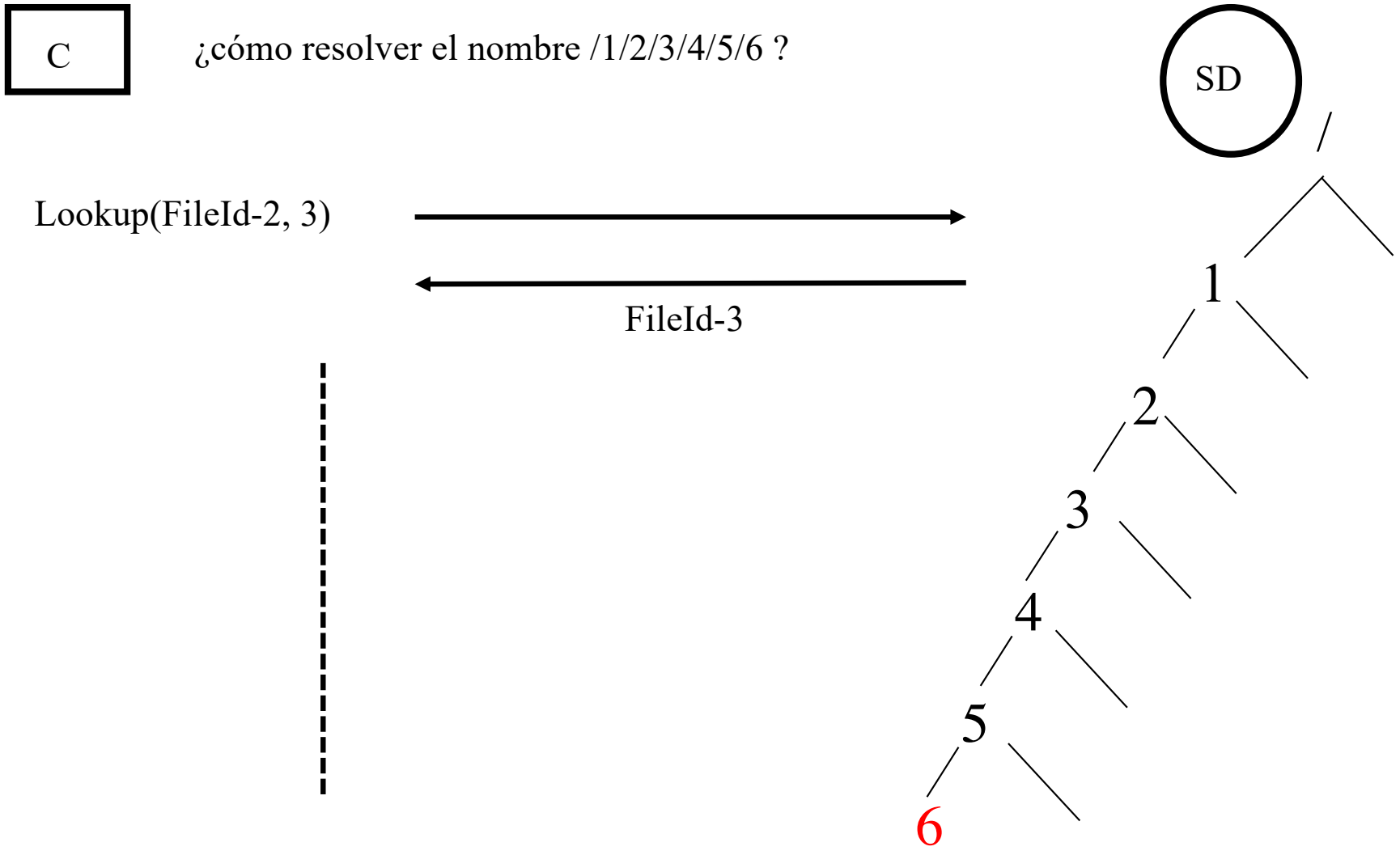
Dirigida por los clientes



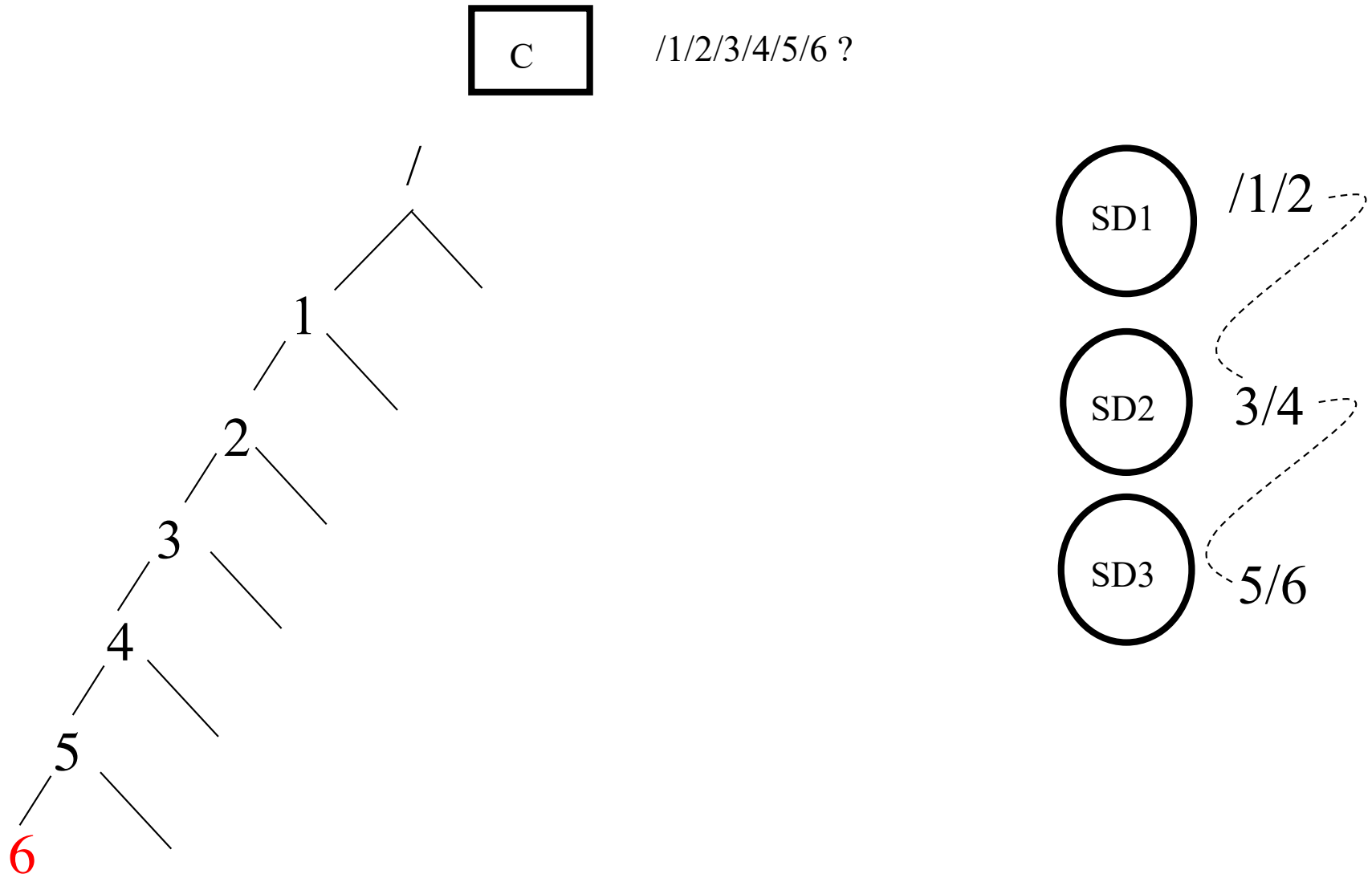
Dirigida por los clientes



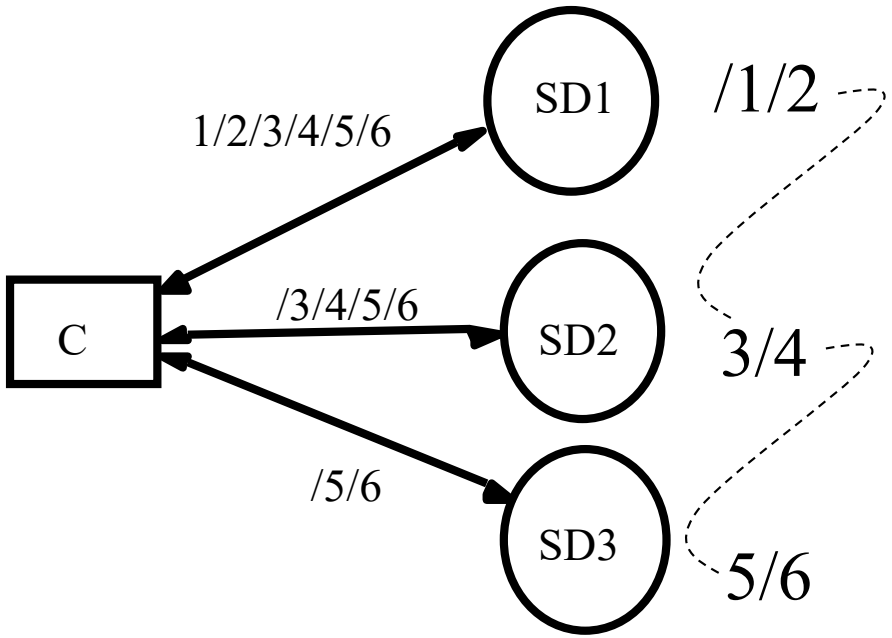
Dirigida por los clientes



Dirigida por los servidores

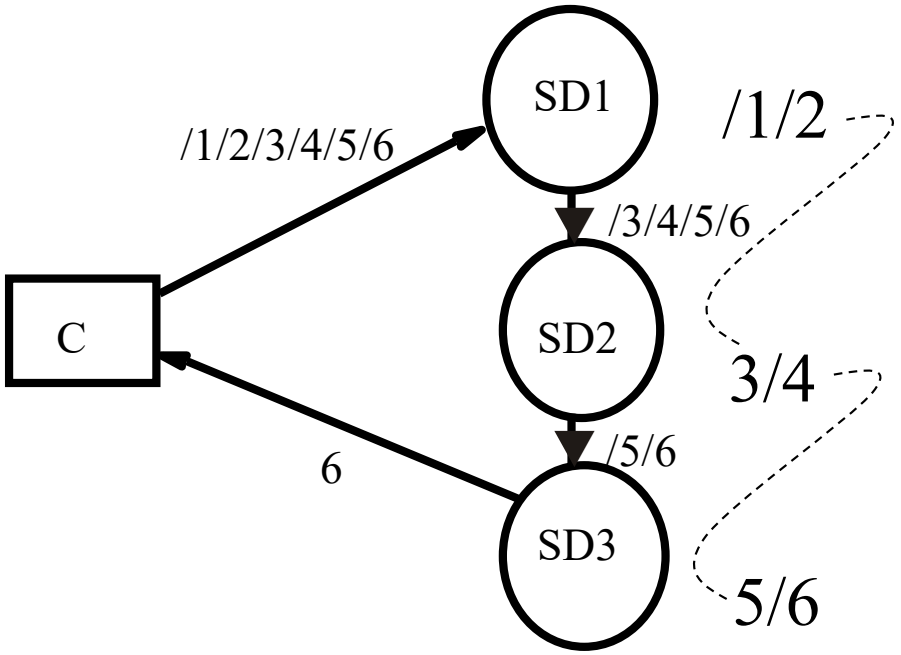


Resolución iterativa



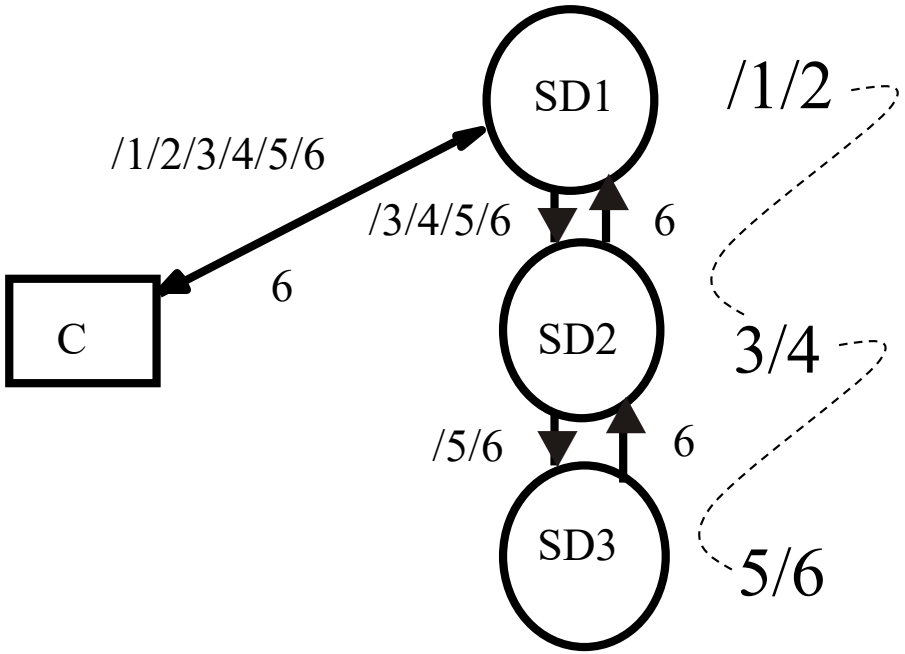
- El cliente envía el nombre al SD
- El SD realiza la traducción hasta que termina en un componente que pertenece a otro SD
- El SD envía el resultado al cliente, el cual si no ha terminado la traducción continúa con el SD correspondiente

Resolución transitiva



- Los SD implicados contactan entre si para llevar a cabo la traducción. El último SD devuelve la traducción al cliente
- Rompe el modelo cliente/servidor (no adecuado para RPC)

Resolución recursiva



- El último SD implicado devuelve el resultado al anterior y así sucesivamente hasta que el primero responde al cliente

Servicio de ficheros

- Se encarga de la gestión de los ficheros y del acceso a los datos
- Aspectos relacionados
  - Semántica de coutilización
  - Métodos de acceso
  - Caché de bloques
  - El problema de la coherencia de cache
  - Métodos para mejorar el rendimiento

Operaciones básicas de un servicio de ficheros

- ReadFile(FileId, pos, n) ➔ Data
  - Lee n bytes a partir de una determinada posición
- WriteFile(FileId, pos, n Data)
  - Escribe n buytes (Data) a partir de una determinada posición
- Create(name) ➔ FileId
  - Crea un nuevo fichero de longitud 0 bytes.
- Delete(FileId)
  - Borra el fichero
- GetAttributes(FileId) ➔ Attr
  - Devuelve los atributos de un fichero
- SetAttributes(FileId, Attr)
  - Modifica los atributos de un fichero

Semánticas de coutilización

- Sesión: serie de accesos que realiza un cliente entre un *open* y un *close*
- La semántica de coutilización especifica el efecto de varios procesos accediendo de forma simultánea al mismo fichero
- Semánticas
  - Semántica UNIX
  - Semántica de sesión
  - Semántica de ficheros inmutables
  - Semántica de transacciones

Semánticas de coutilización

- Semántica UNIX
  - Una lectura *ve* los efectos de todas las escrituras previas
  - El efecto de dos escrituras sucesivas es el de la última de ellas
  - Los procesos pueden compartir el puntero de la posición (si están emparentados)
  - Difícil de implementar en sistemas distribuidos
    - Mantener un copia única

Semánticas de coutilización

- Semántica de sesión:
  - Cambios a un fichero abierto son visibles únicamente en el proceso (nodo) que modificó el fichero
  - Una vez cerrado el fichero, los cambios se hacen visibles a futuras sesiones
  - Múltiples imágenes del fichero
  - Dos sesiones sobre el mismo fichero que terminan concurrentemente: la última deja el resultado final
  - Si dos procesos quieren compartir datos deben abrir y cerrar el fichero para propagar los datos
    - No adecuado para procesos que acceden de forma concurrente a un fichero
  - No existen punteros compartidos

Semánticas de coutilización

- Semántica de ficheros inmutables
  - El contenido de un fichero no puede modificarse
  - El nombre del fichero no puede reutilizarse
  - Sólo se puede compartir un fichero para sólo lectura
  - Ejemplo: Sistema HDFS utilizado en aplicaciones Big Data

- Semántica de transacciones
  - Todo acceso al fichero se realiza entre
    - BEGIN_TRANSACTION
    - END_TRANSACTION
  - El sistema asegura que dos transacciones ejecutadas de forma concurrente tienen el efecto equivalente a ejecutarlas secuencialmente en algún orden

Métodos de acceso remoto

- Modelo carga/descarga
  - Transferencias completas del fichero del servidor al cliente
  - Localmente se almacenan en memoria o discos locales
  - Normalmente utilizan semántica de sesión
  - Eficiencia en las transferencias
  - Llamada open con mucha latencia
  - Múltiples copias de un fichero
- Modelo de servicios remotos
  - El servidor debe proporcionar todas las operaciones sobre el fichero.
  - Acceso por bloques
  - Modelo cliente/servidor
- Empleo de caché en el cliente
  - Combina los dos modelos anteriores.

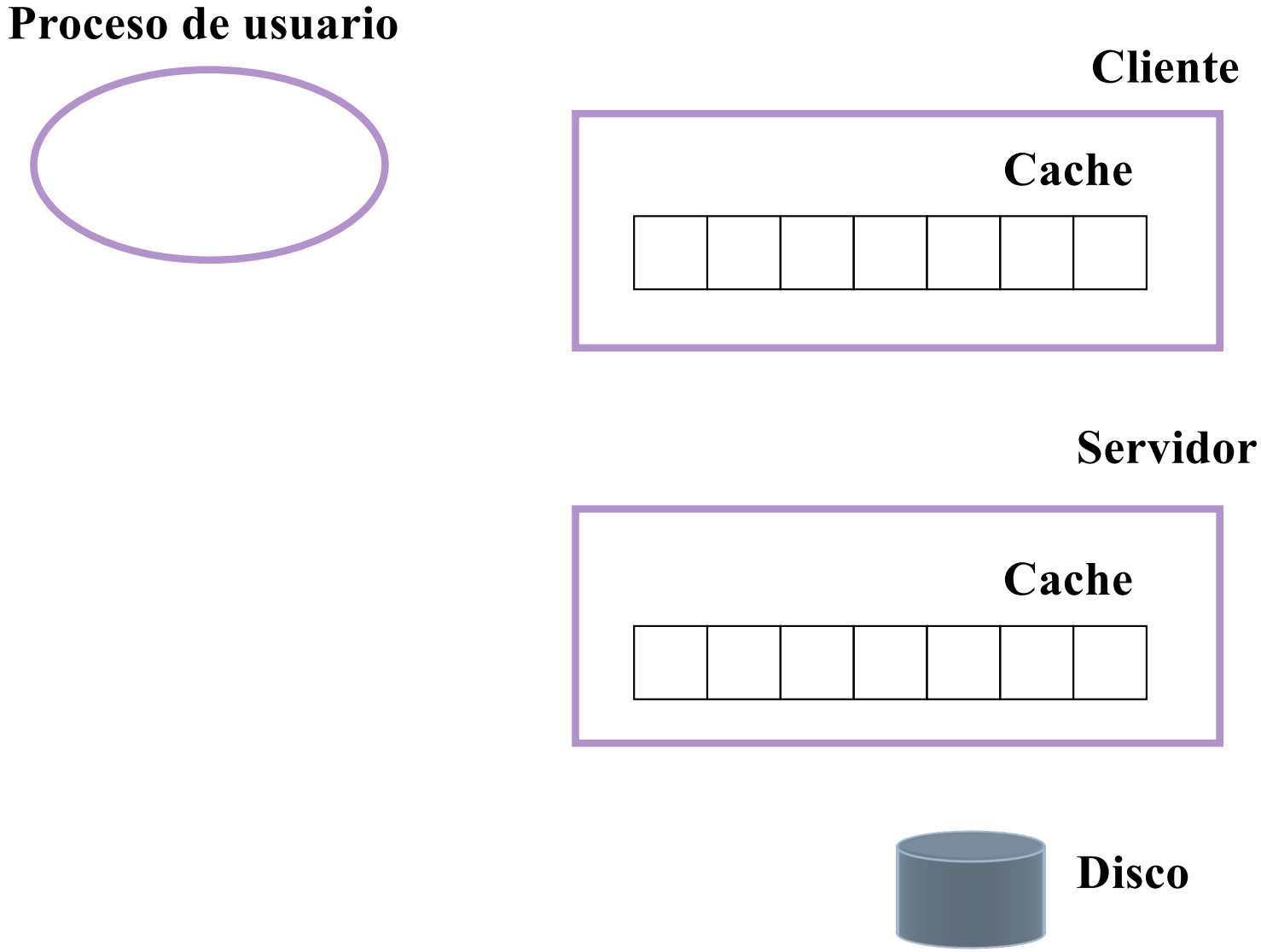
Caché de bloques

- El empleo de caché de bloques permite mejorar el rendimiento
  - Explota el principio de proximidad de referencias
    - Proximidad temporal
    - Proximidad espacial
  - Lecturas adelantadas
    - Mejora el rendimiento de las operaciones de lectura, sobre todo si son secuenciales
  - Escrituras diferidas
    - Mejora el rendimiento de las escrituras
- Otros tipos de caché
  - Caché de nombres
  - Caché de metadatos del sistema de ficheros

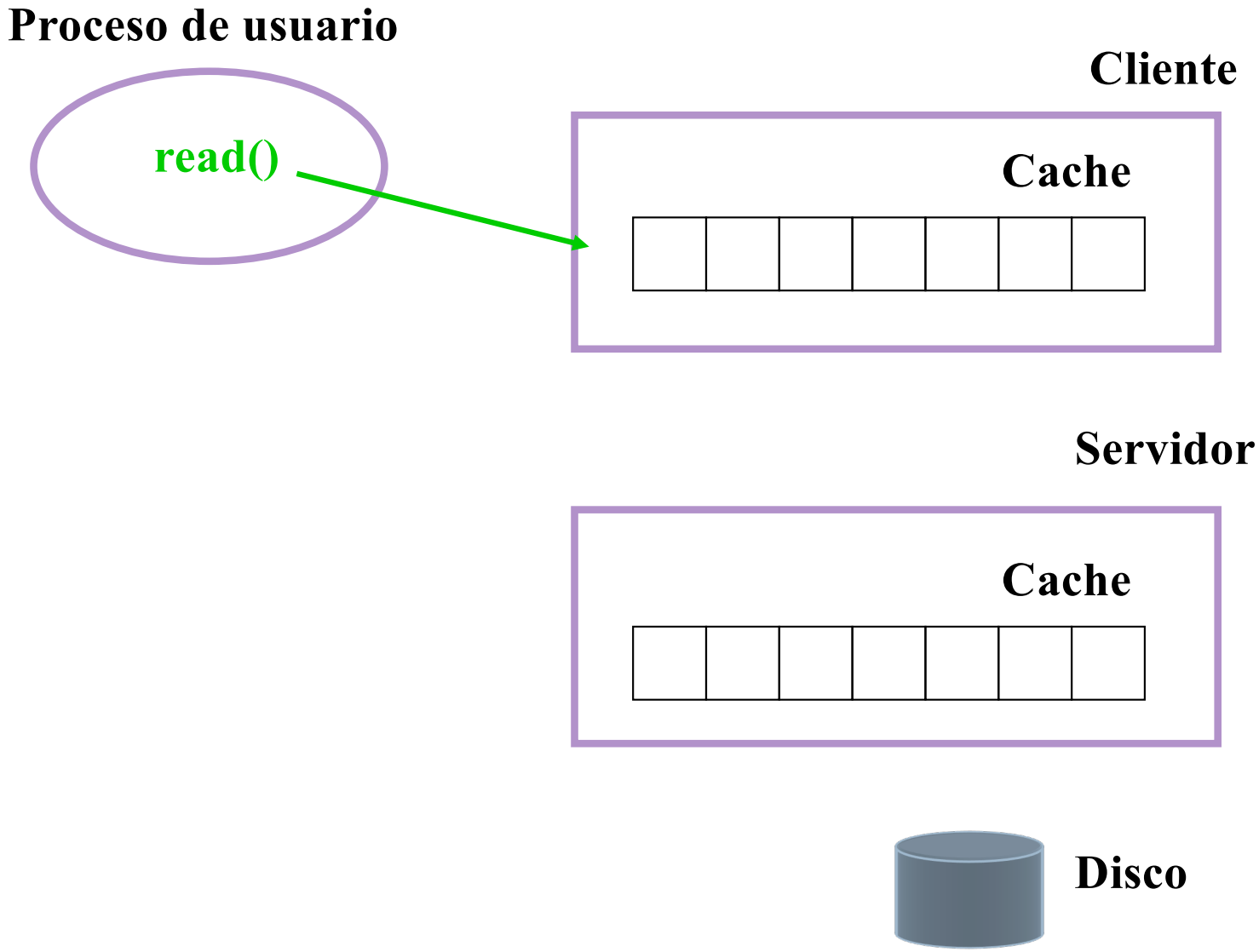
Localización de las caché en un SFD

- **Caché en los servidores**
  - Reducen los accesos a disco
- **Caché en los clientes**
  - Reducen el tráfico por la red
  - Reducen la carga en los servidores
  - Mejora la capacidad de crecimiento
  - Dos posibles localizaciones
    - En discos locales
      - Más capacidad,
      - Más lento
      - No volátil, facilita la recuperación
    - En memoria principal
      - Menor capacidad
      - Más rápido
      - Memoria volátil

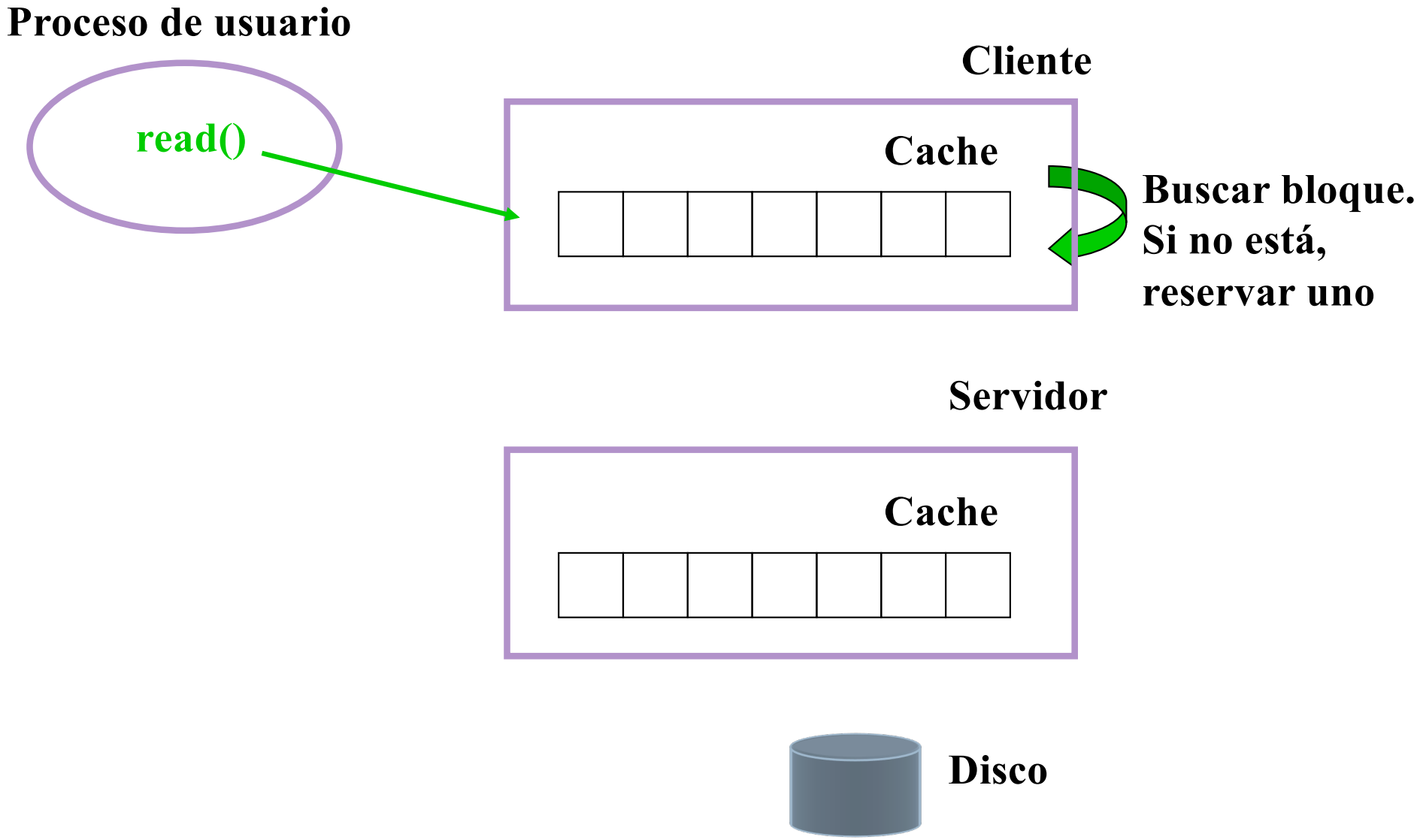
Funcionamiento de una caché de bloques



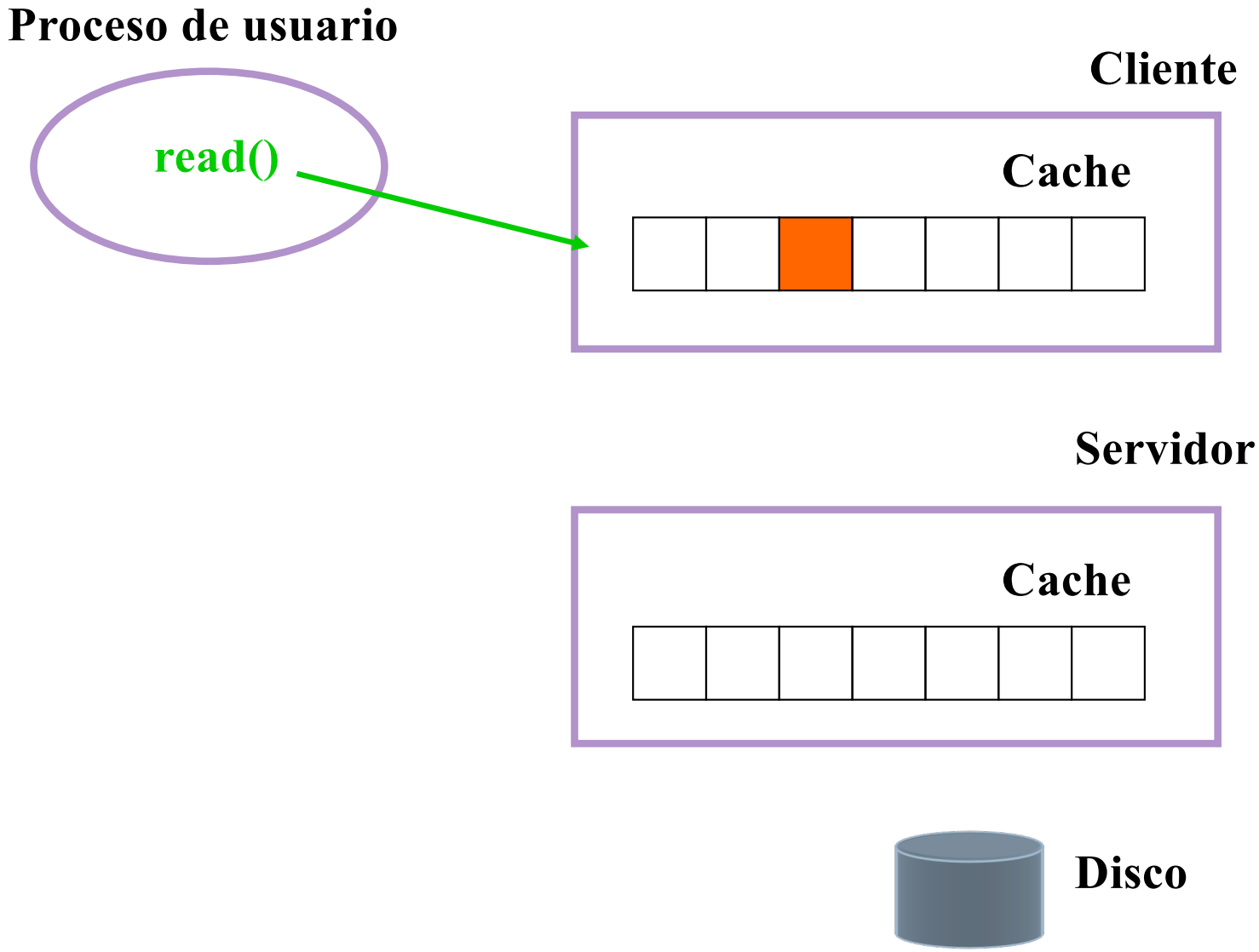
Funcionamiento de una caché de bloques



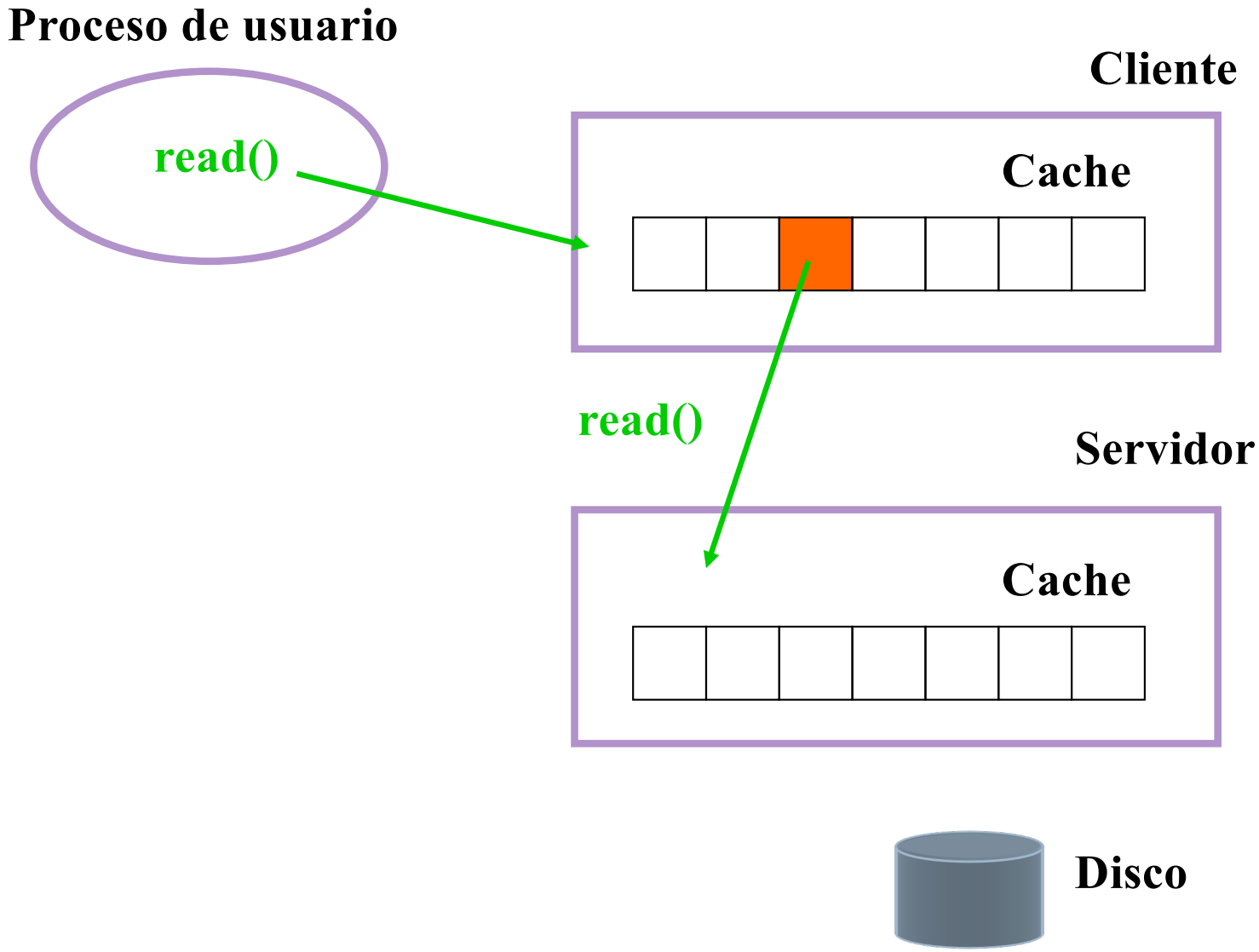
Funcionamiento de una caché de bloques



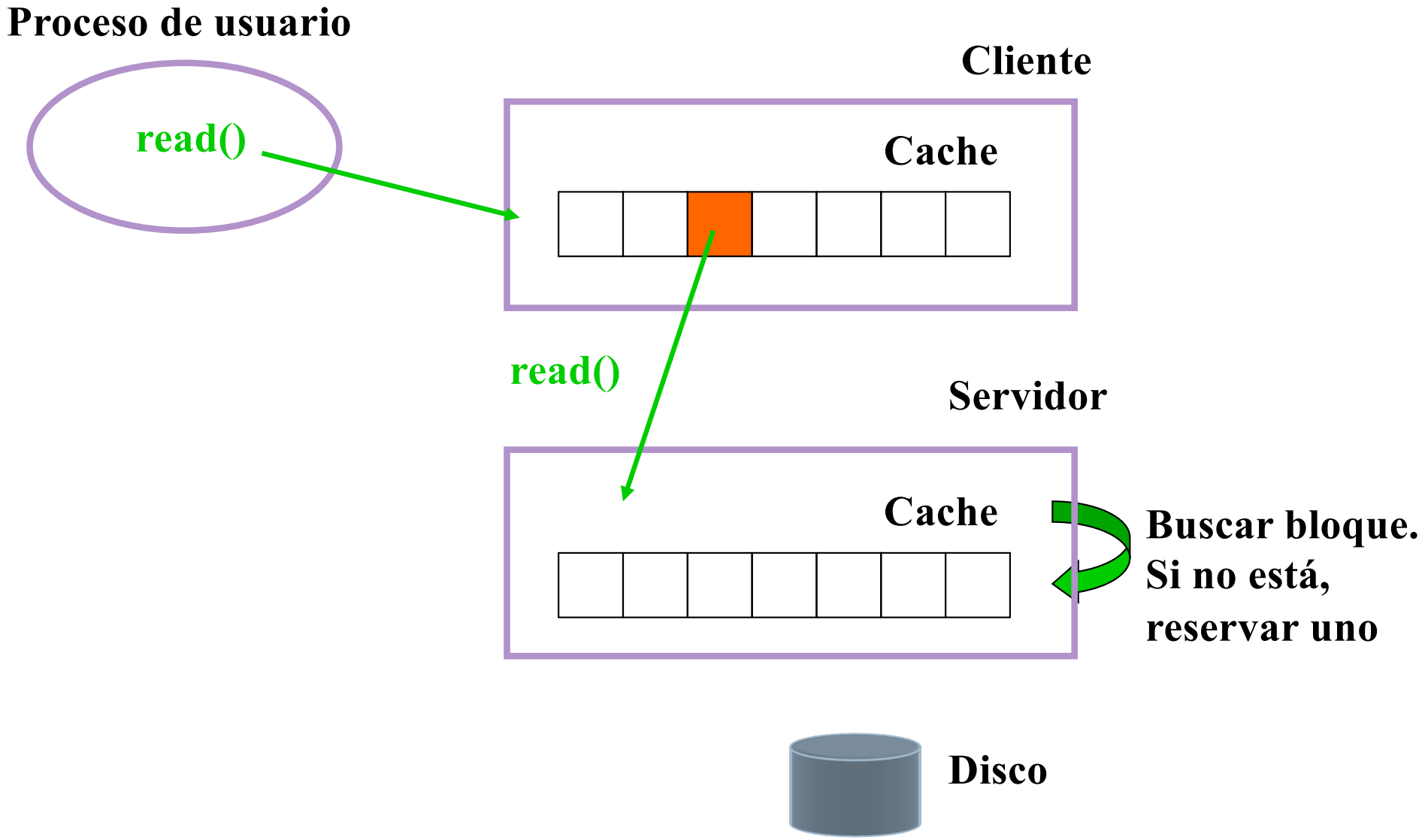
Funcionamiento de una caché de bloques



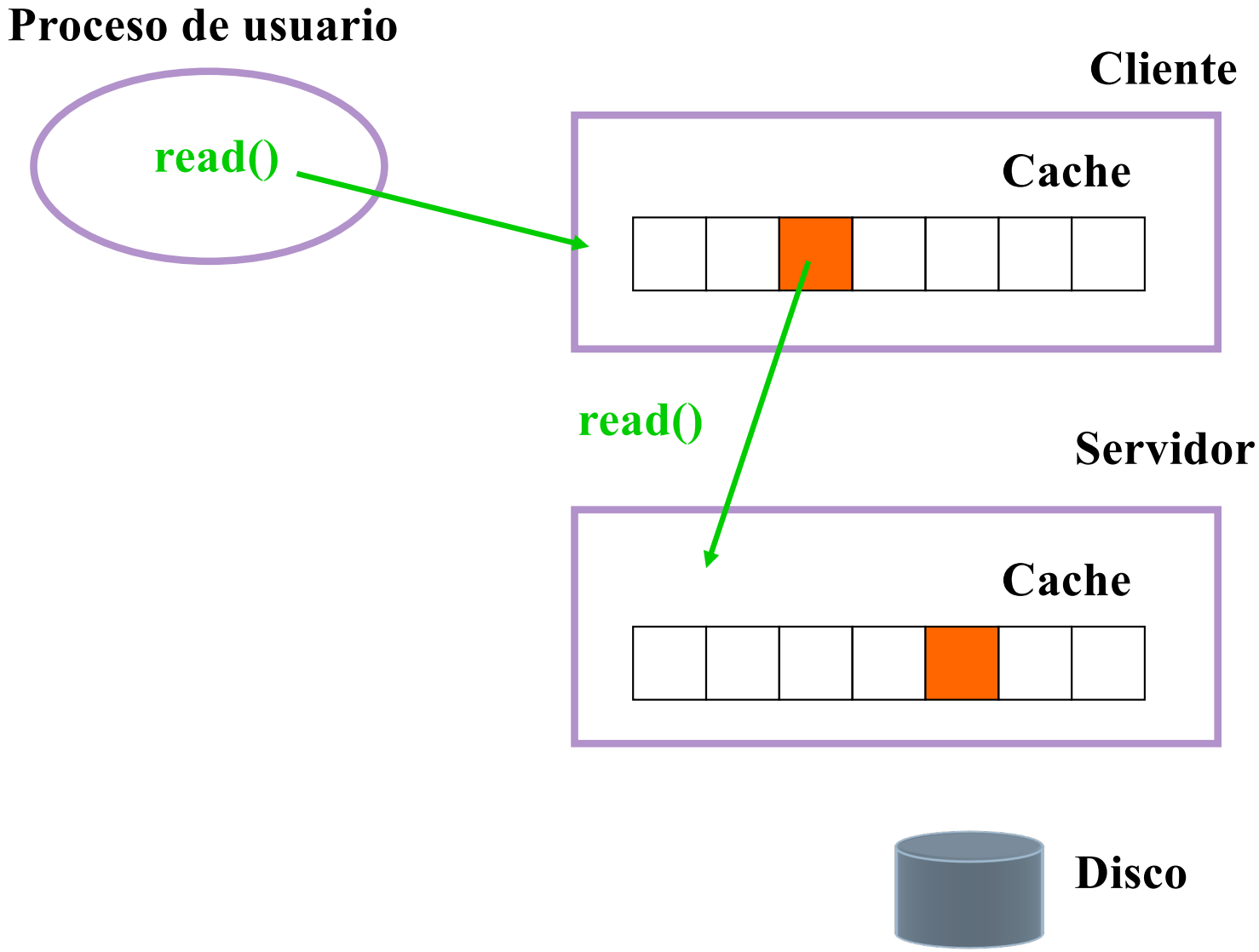
Funcionamiento de una caché de bloques



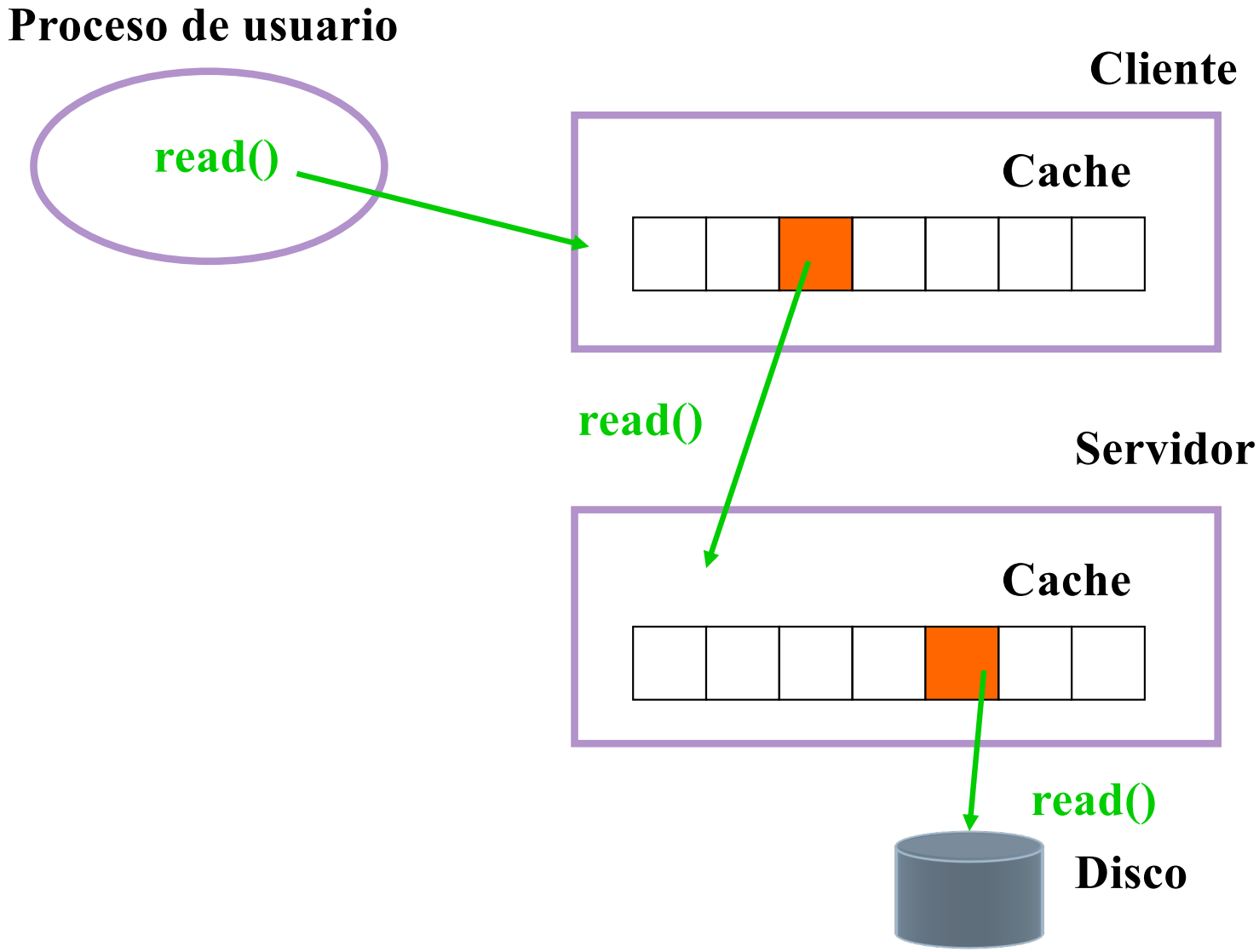
Funcionamiento de una caché de bloques



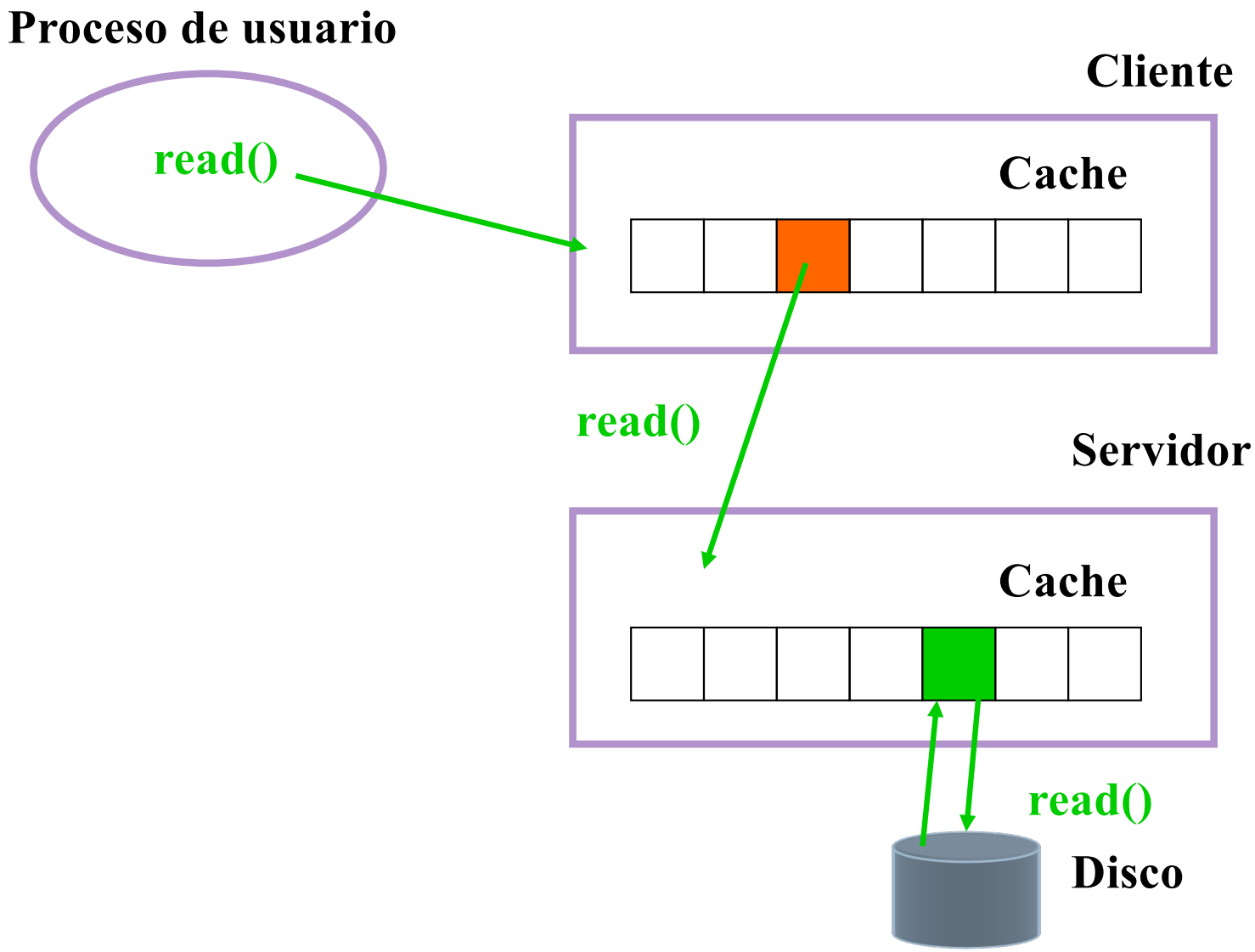
Funcionamiento de una caché de bloques



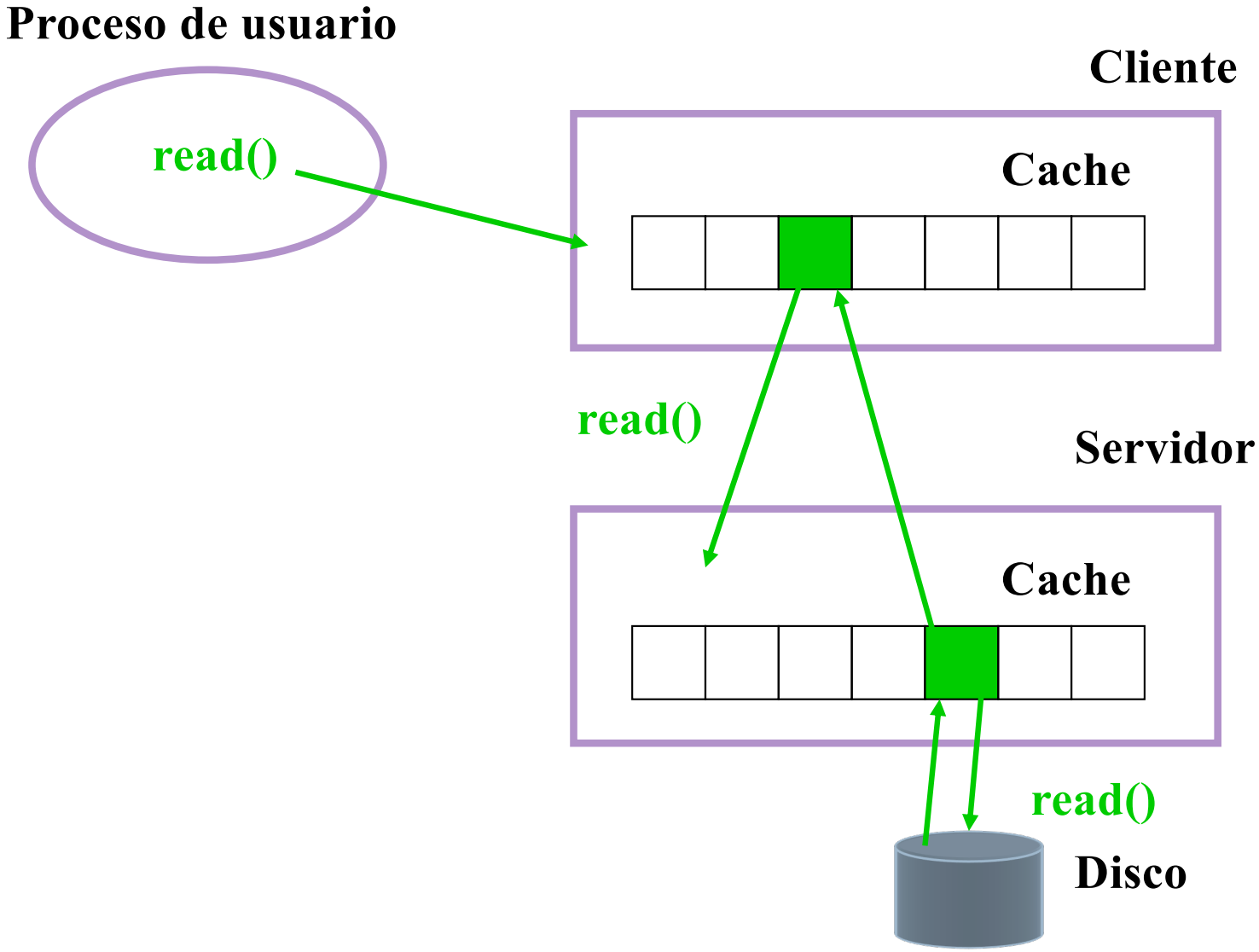
Funcionamiento de una caché de bloques



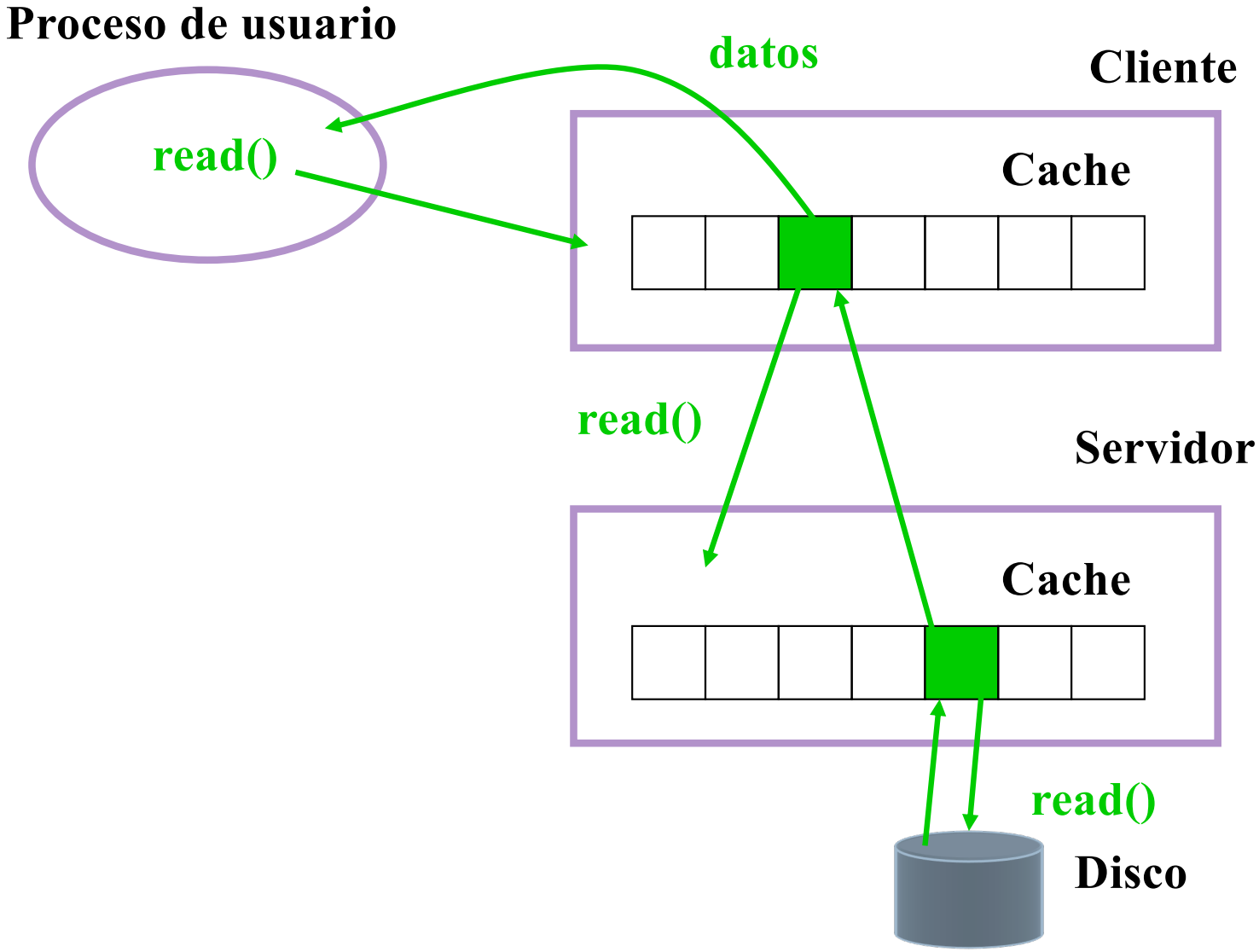
Funcionamiento de una caché de bloques



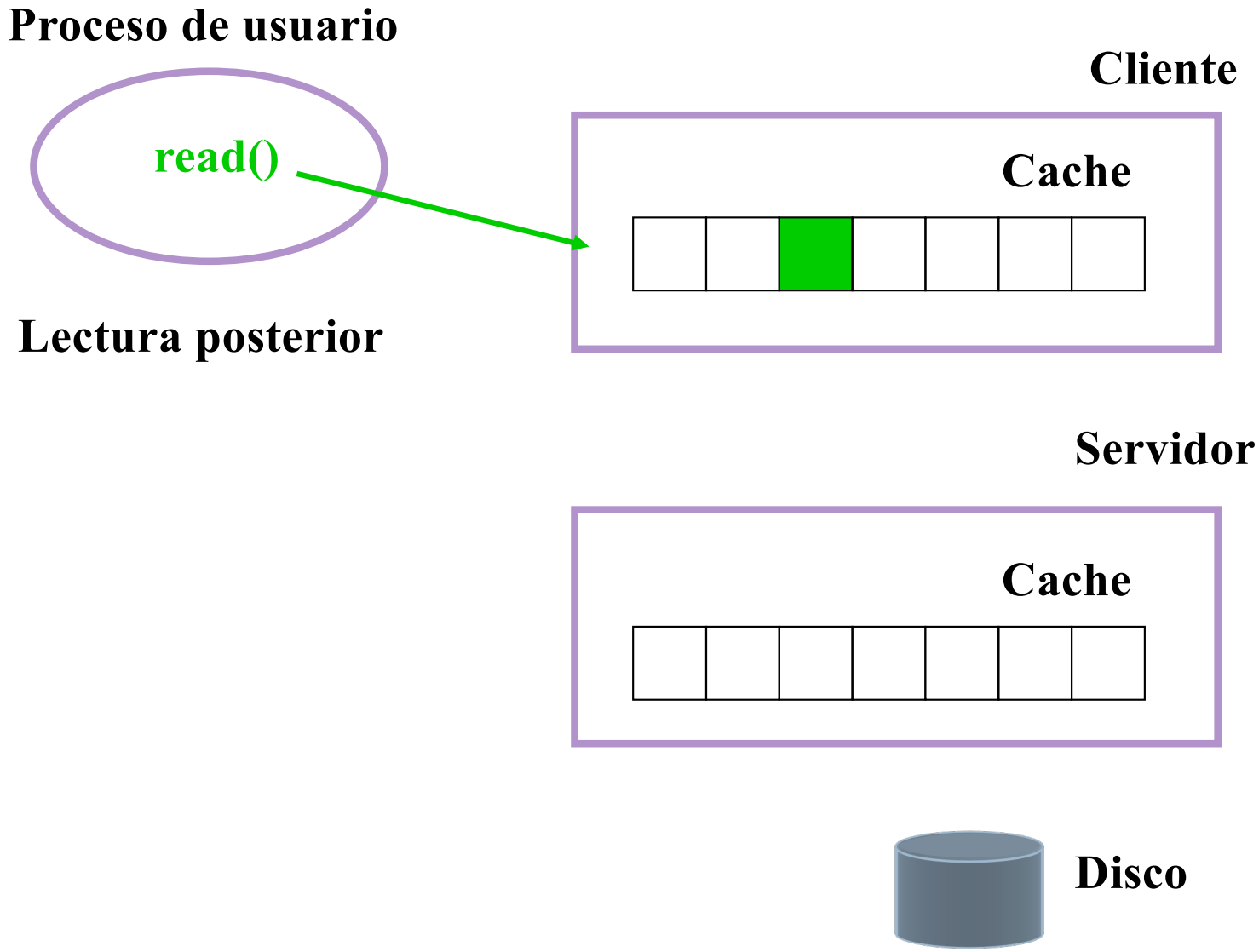
Funcionamiento de una caché de bloques



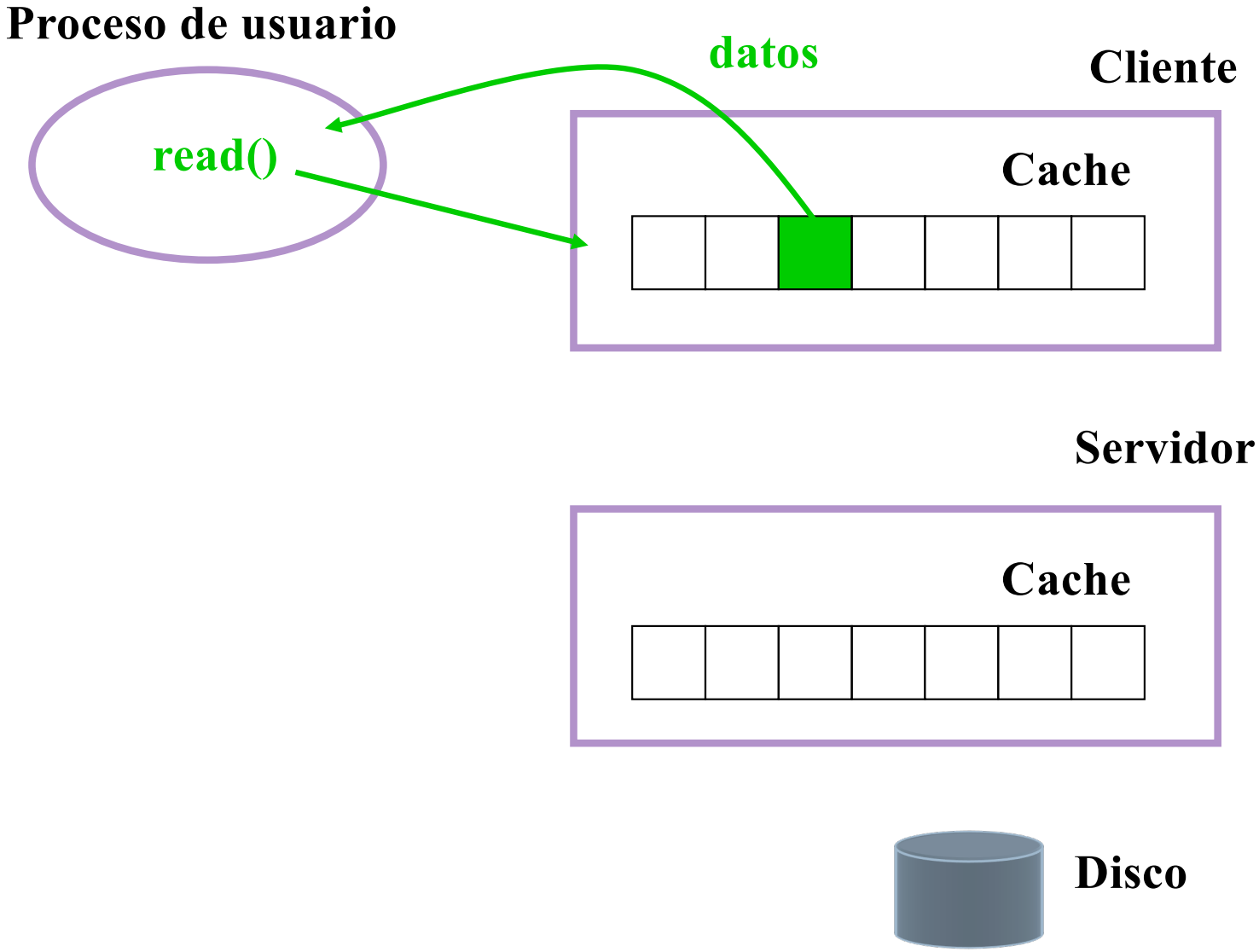
Funcionamiento de una caché de bloques



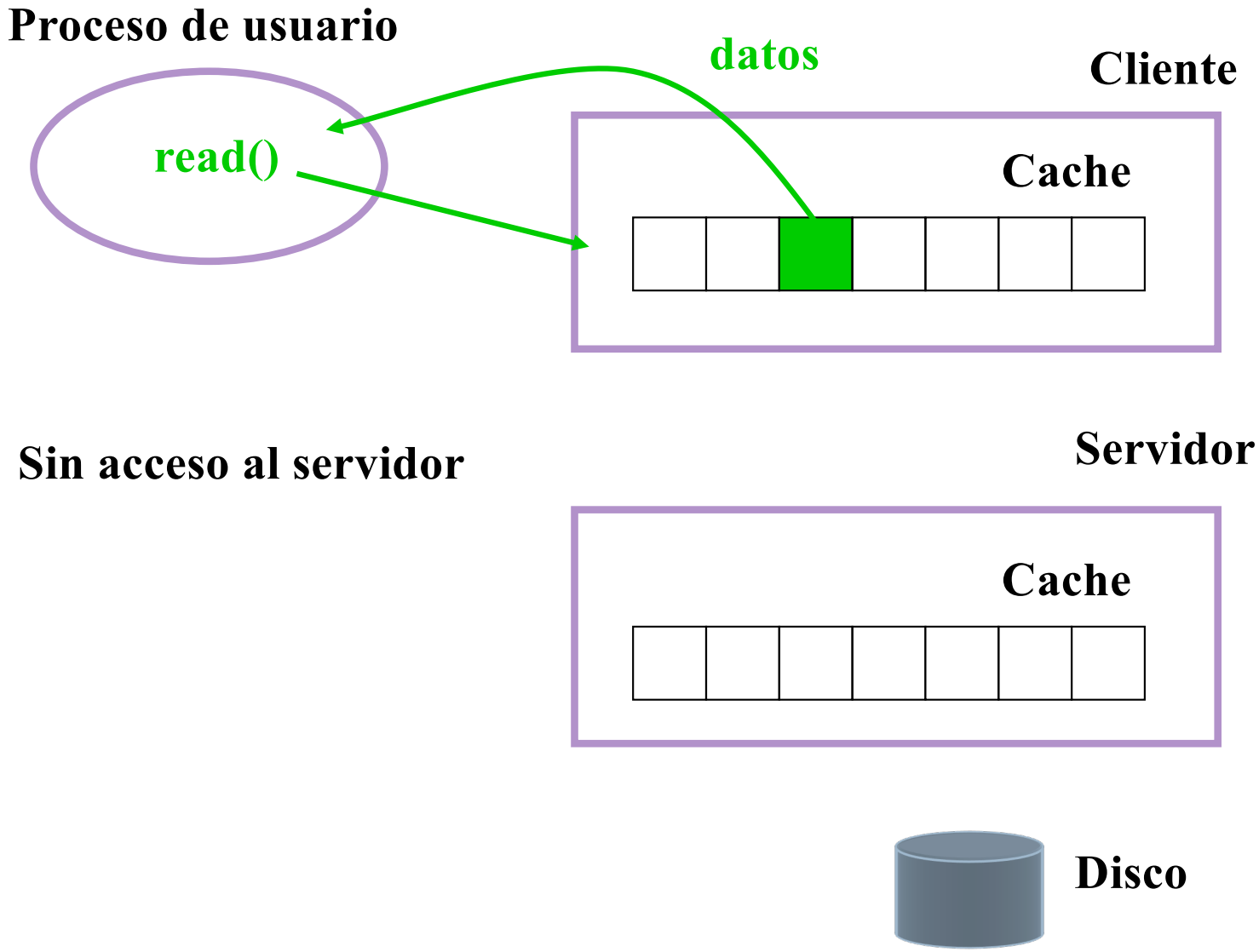
Funcionamiento de una caché de bloques



Funcionamiento de una caché de bloques



Funcionamiento de una caché de bloques



Tamaño de la unidad de caché

- Mayor tamaño puede incrementar la tasa de aciertos y mejorar la utilización de la red pero
  - Aumentan los problemas de coherencia
- Depende de las características de las aplicaciones
- En memoria caché grandes
  - Es beneficioso emplear bloques grandes (8 KB y más)
- En memorias pequeñas
  - El uso de bloques grandes es menos adecuado

Políticas de actualización

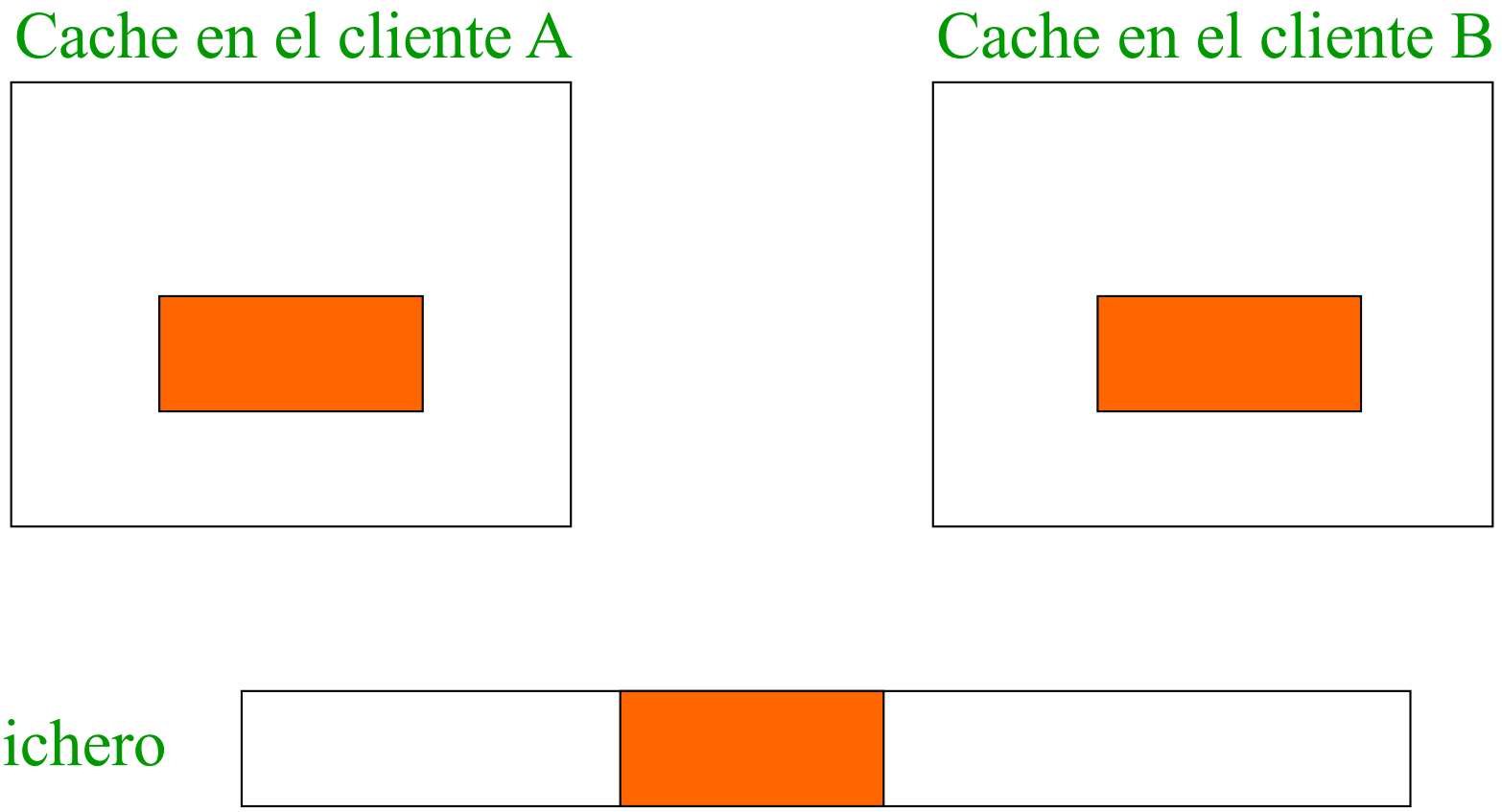
- Escritura inmediata (*write-through*)
  - Buena fiabilidad
  - En escrituras se obtiene el mismo rendimiento que en el modelo de accesos remotos
- Escritura diferida (*write-back, delayed-write*)
  - Escrituras más rápidas. Se reduce el tráfico en la red
  - Los datos pueden borrarse antes de ser enviados al servidor
  - Alternativas
    - Volcado (*flush*) periódico (*Sprite*)
    - Write-on-close

Problema de la coherencia de caché

- El uso de caché en los clientes de un sistema de ficheros introduce el problema de la coherencia de caché:
  - Múltiples copias.
- El problema surge cuando se **coutiliza** un fichero en escritura:
  - Coutilización en escritura secuencial
    - Típico en entornos y aplicaciones distribuidas.
  - Coutilización en escritura concurrente
    - Típico en aplicaciones paralelas.

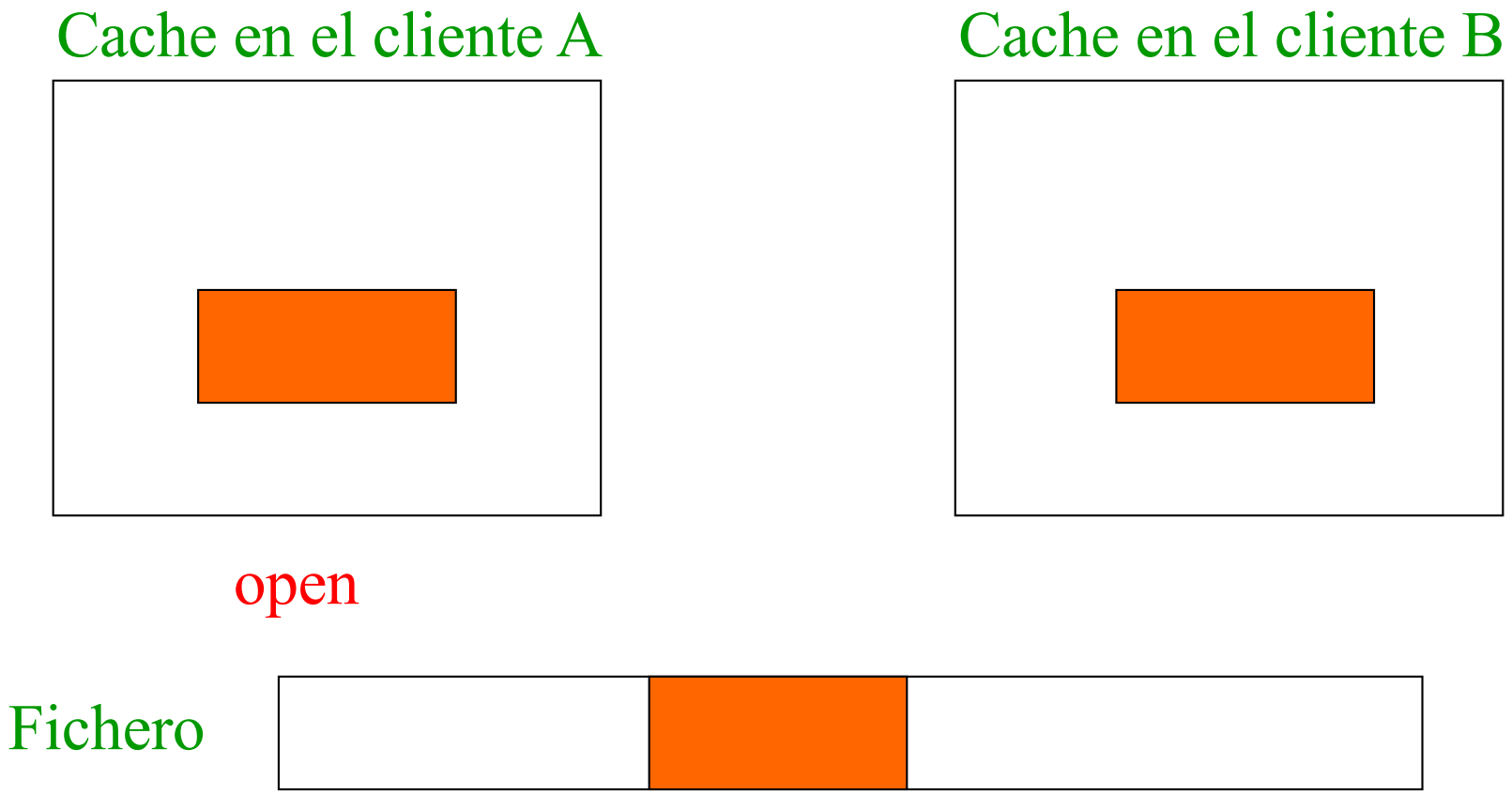
Problema de la coherencia de caché

Inconsistencia en escritura secuencial debido a la no actualización de las copias



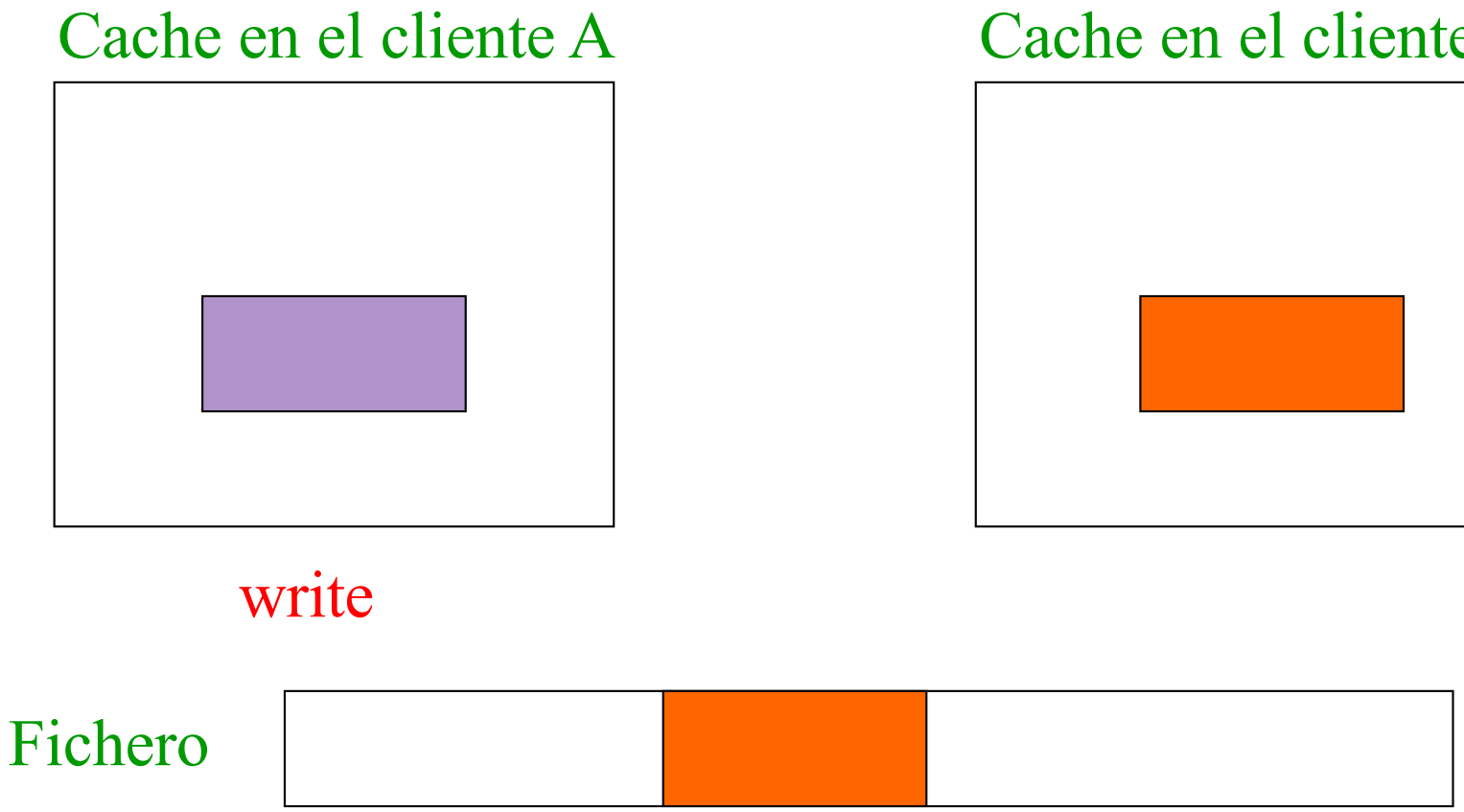
Problema de la coherencia de caché

Inconsistencia en escritura secuencial debido a la no actualización de las copias



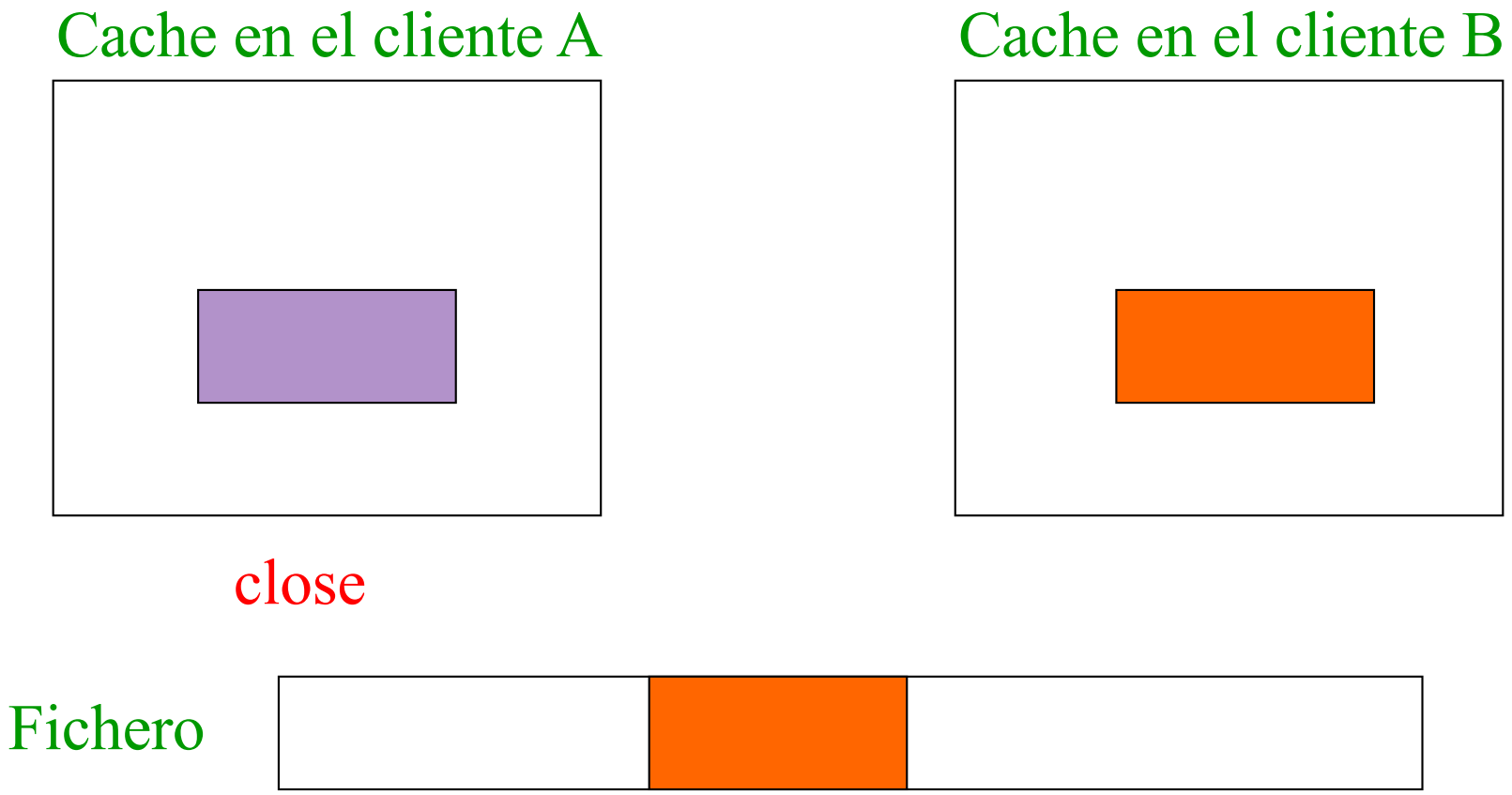
Problema de la coherencia de caché

Inconsistencia en escritura secuencial debido a la no actualización de las copias



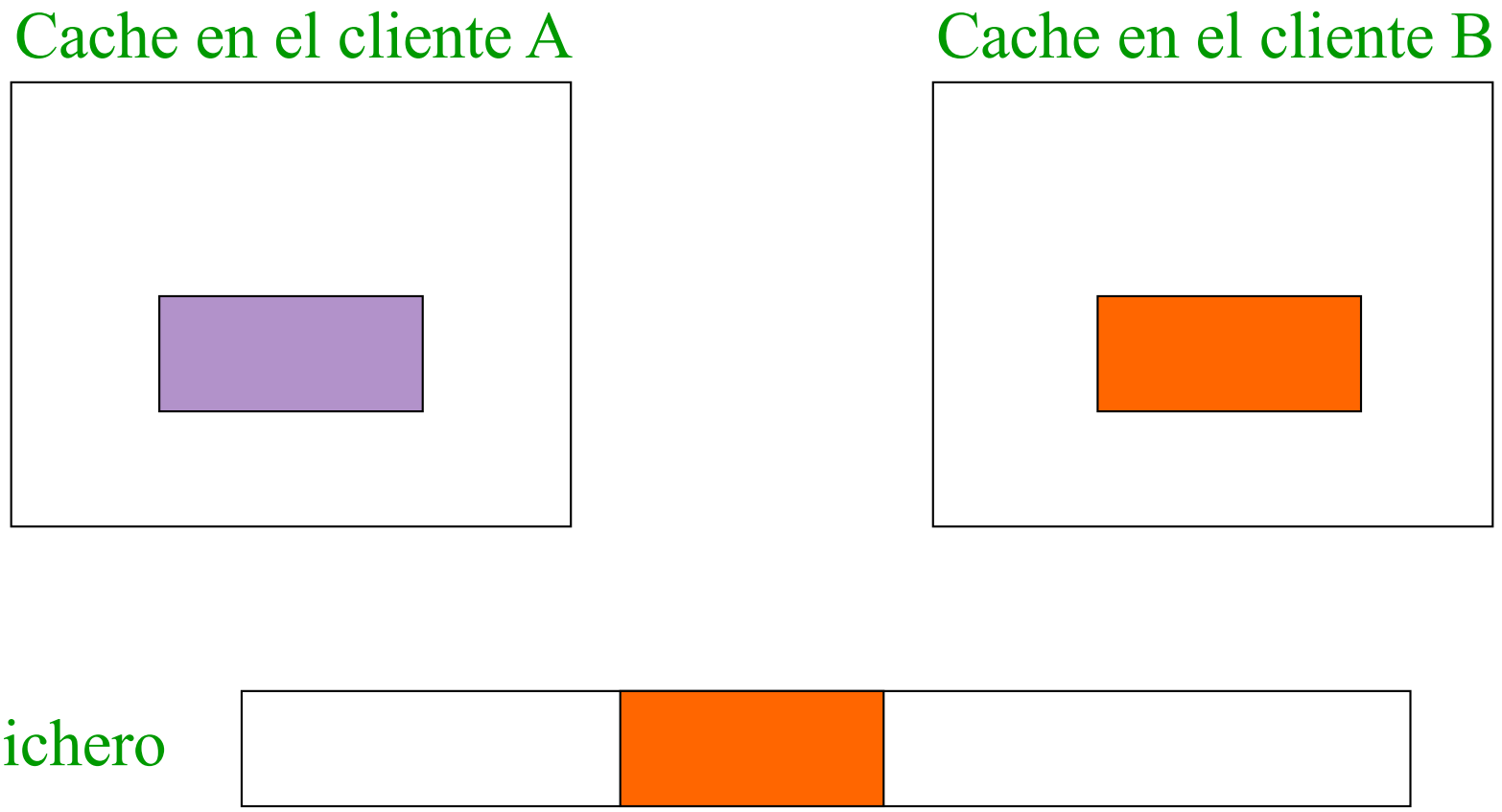
Problema de la coherencia de caché

Inconsistencia en escritura secuencial debido a la no actualización de las copias



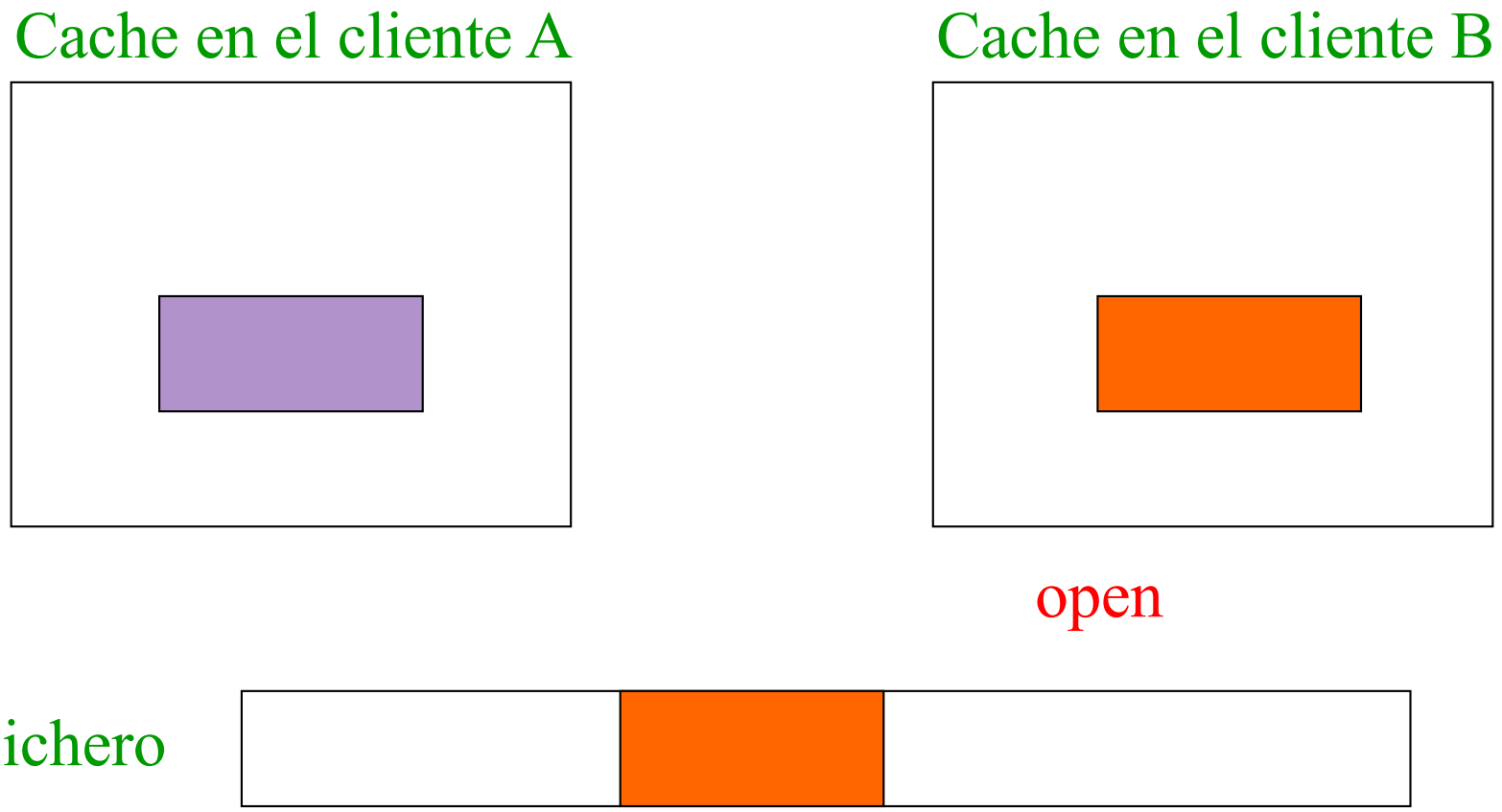
Problema de la coherencia de caché

Inconsistencia en escritura secuencial debido a la no actualización de las copias



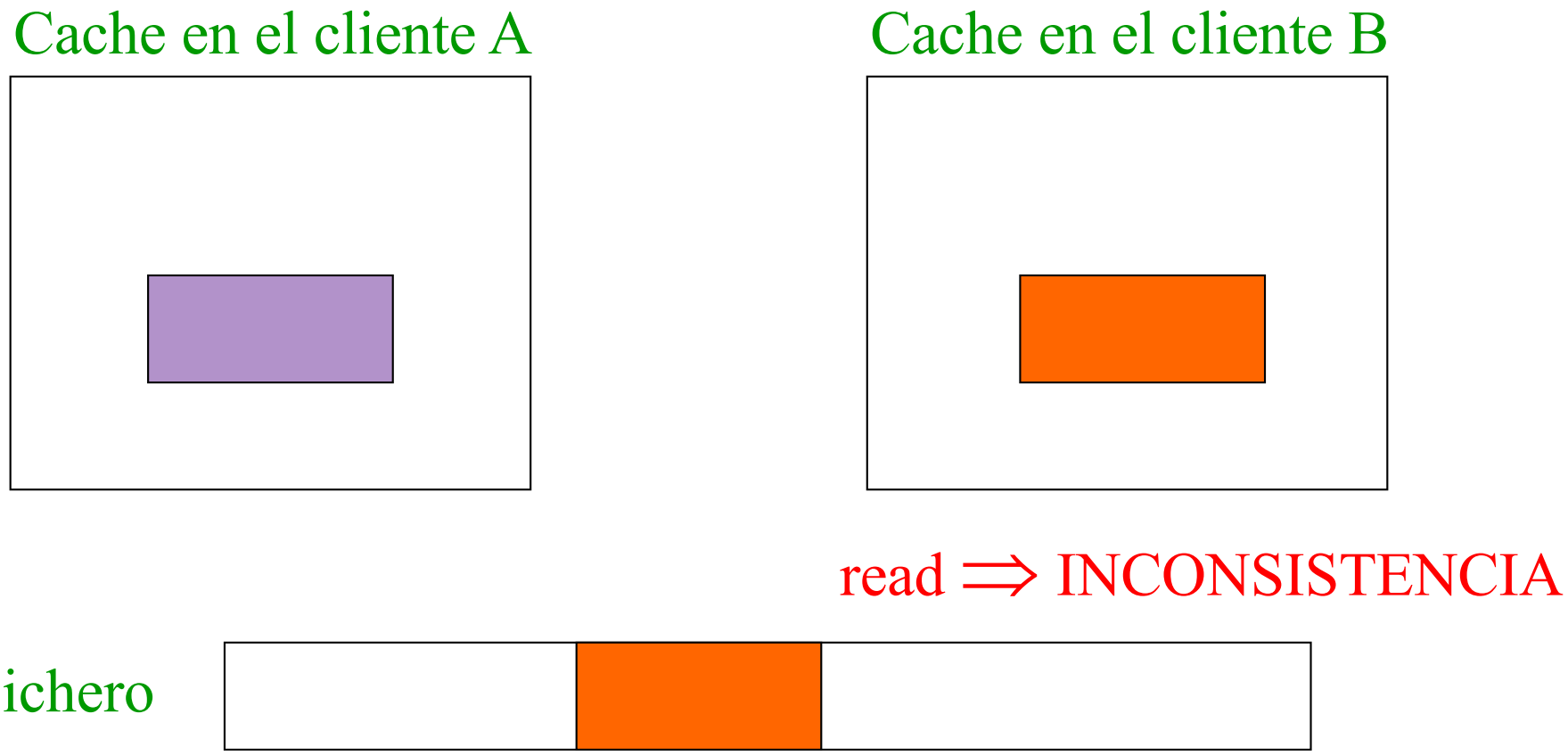
Problema de la coherencia de caché

Inconsistencia en escritura secuencial debido a la no actualización de las copias



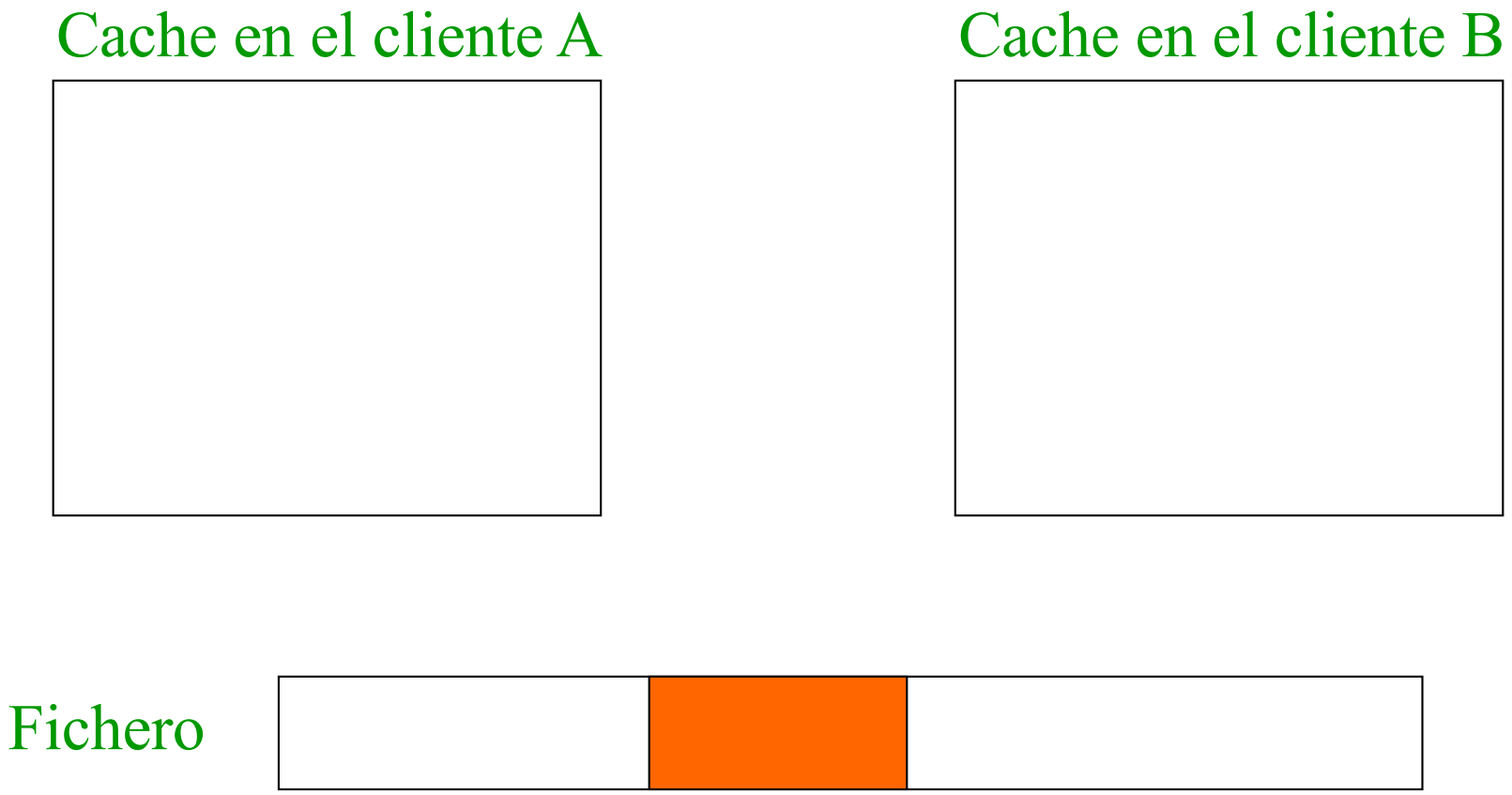
Problema de la coherencia de caché

Inconsistencia en escritura secuencial debido a la no actualización de las copias



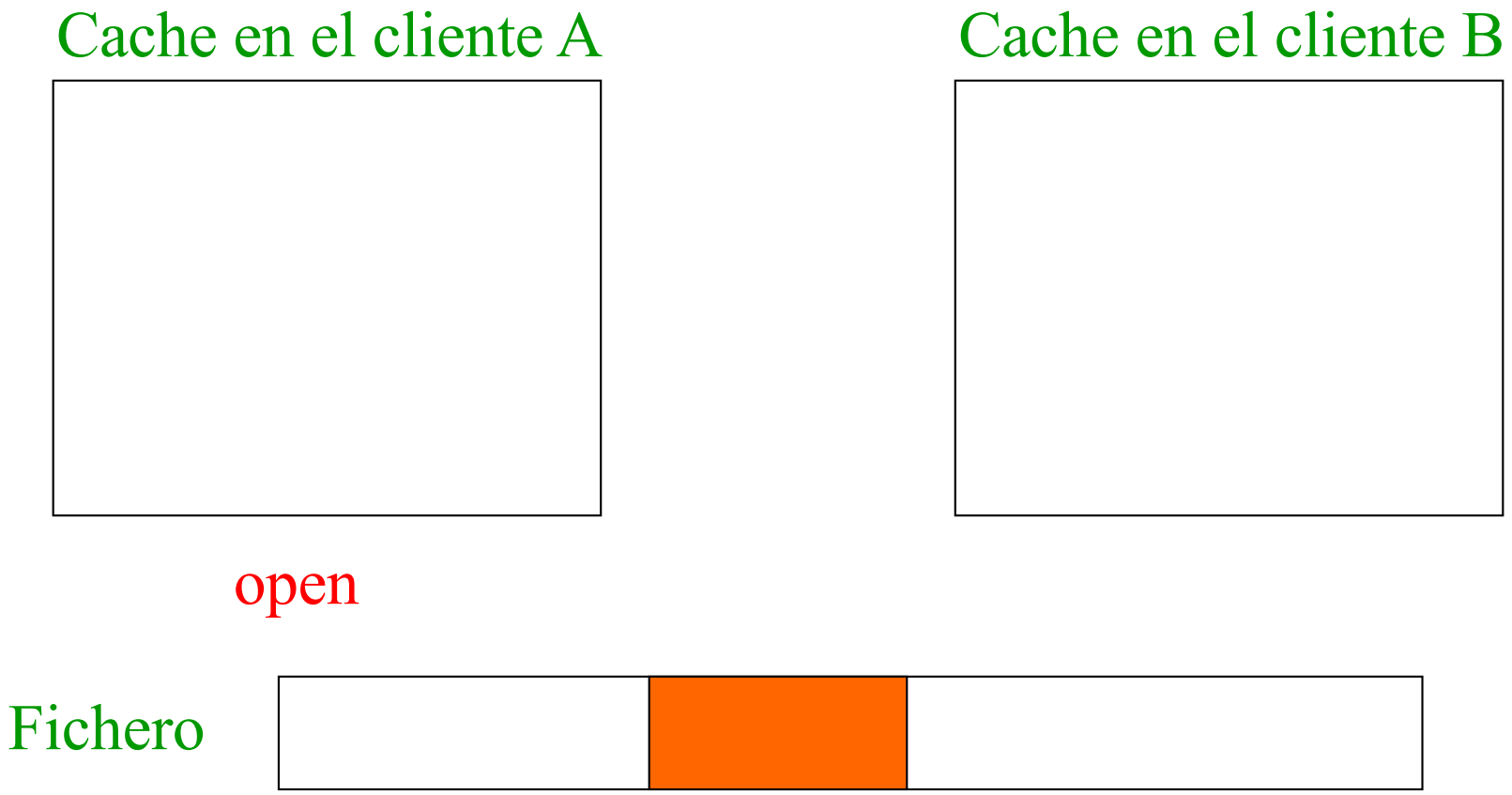
Problema de la coherencia de caché

Inconsistencia en escritura secuencial debido a la utilización de técnicas de escritura diferida



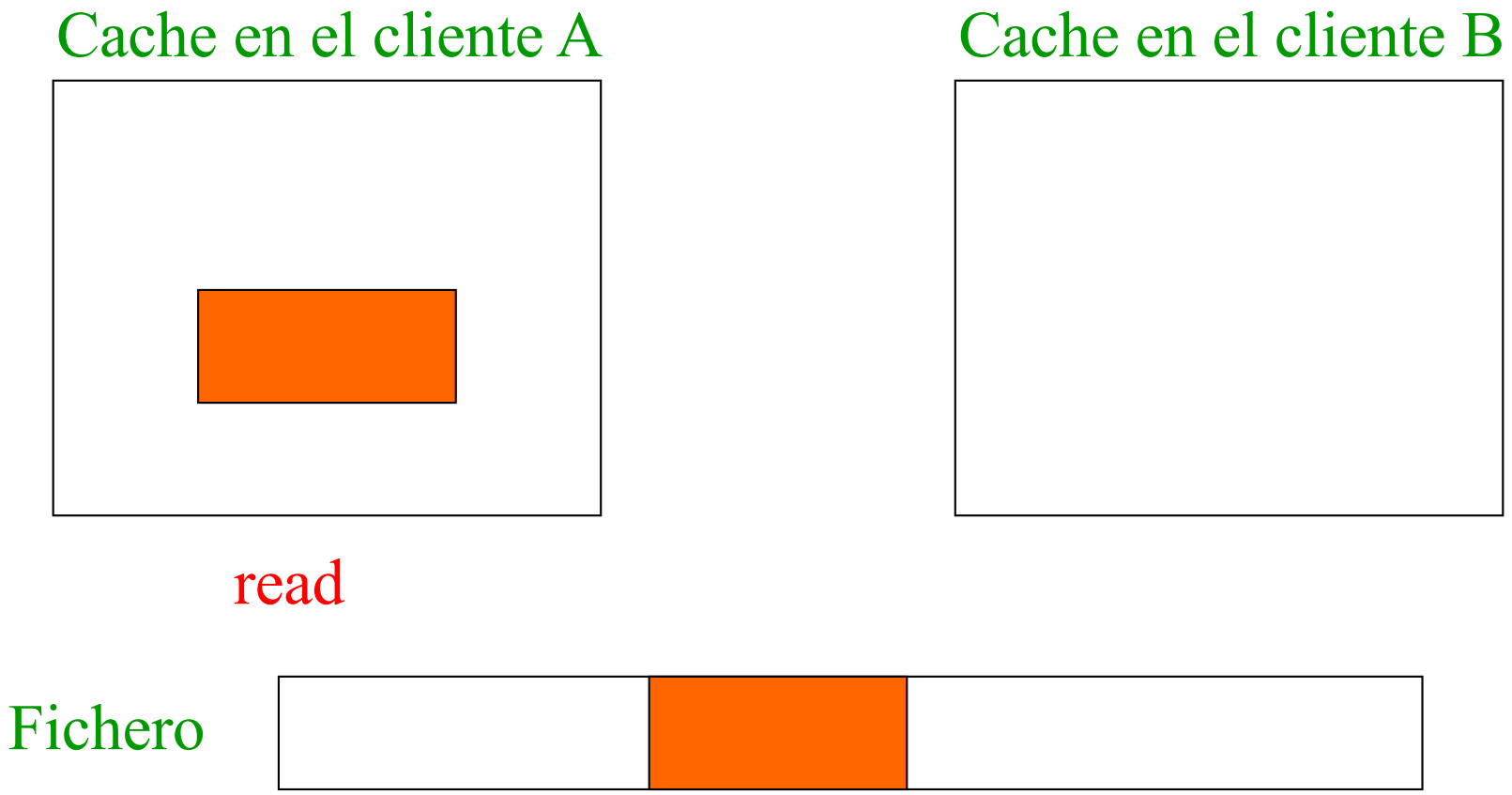
Problema de la coherencia de caché

Inconsistencia en escritura secuencial debido a la utilización de técnicas de escritura diferida



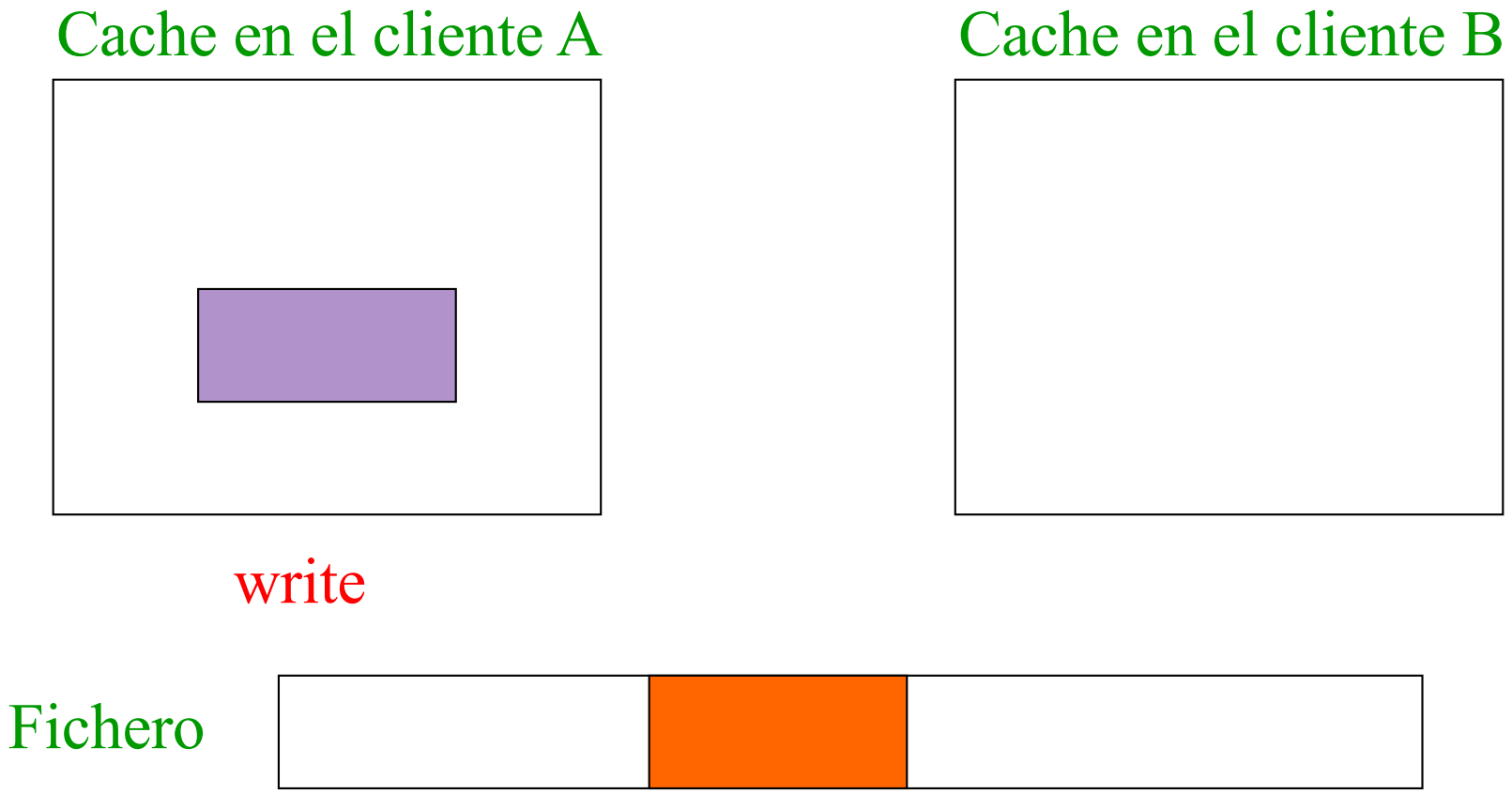
Problema de la coherencia de caché

Inconsistencia en escritura secuencial debido a la utilización de técnicas de escritura diferida



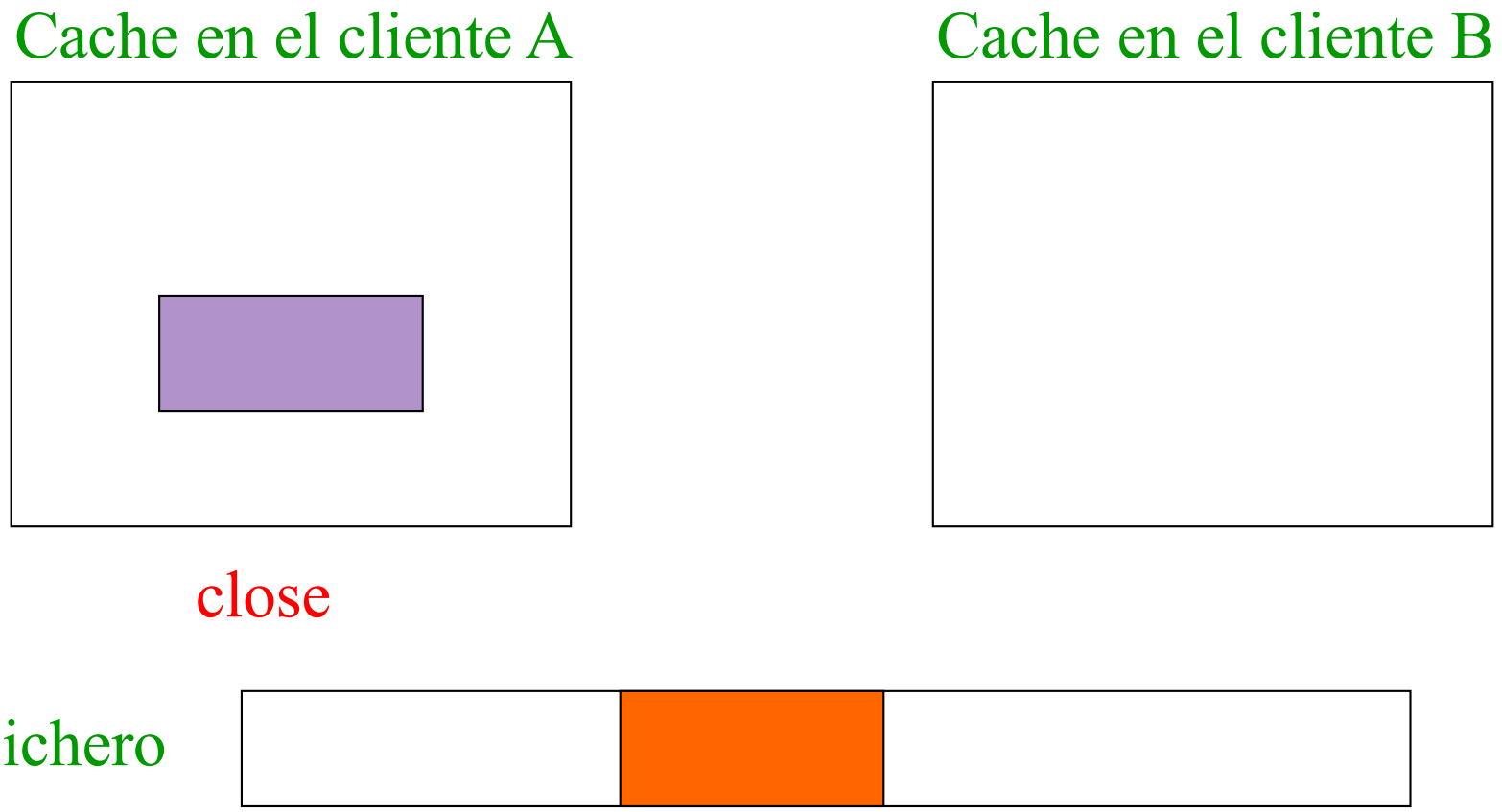
Problema de la coherencia de caché

Inconsistencia en escritura secuencial debido a la utilización de técnicas de escritura diferida



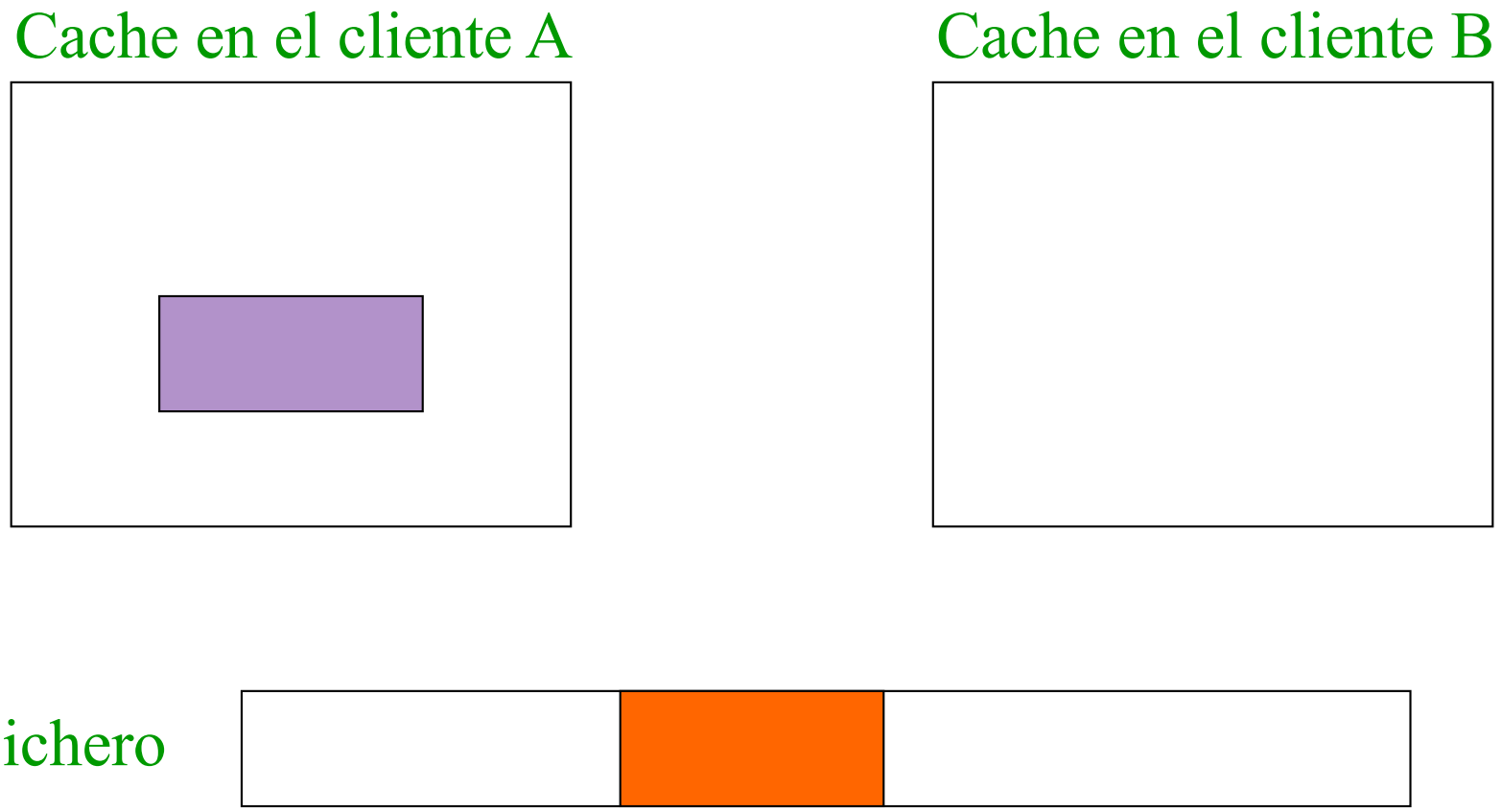
Problema de la coherencia de caché

Inconsistencia en escritura secuencial debido a la utilización de técnicas de escritura diferida



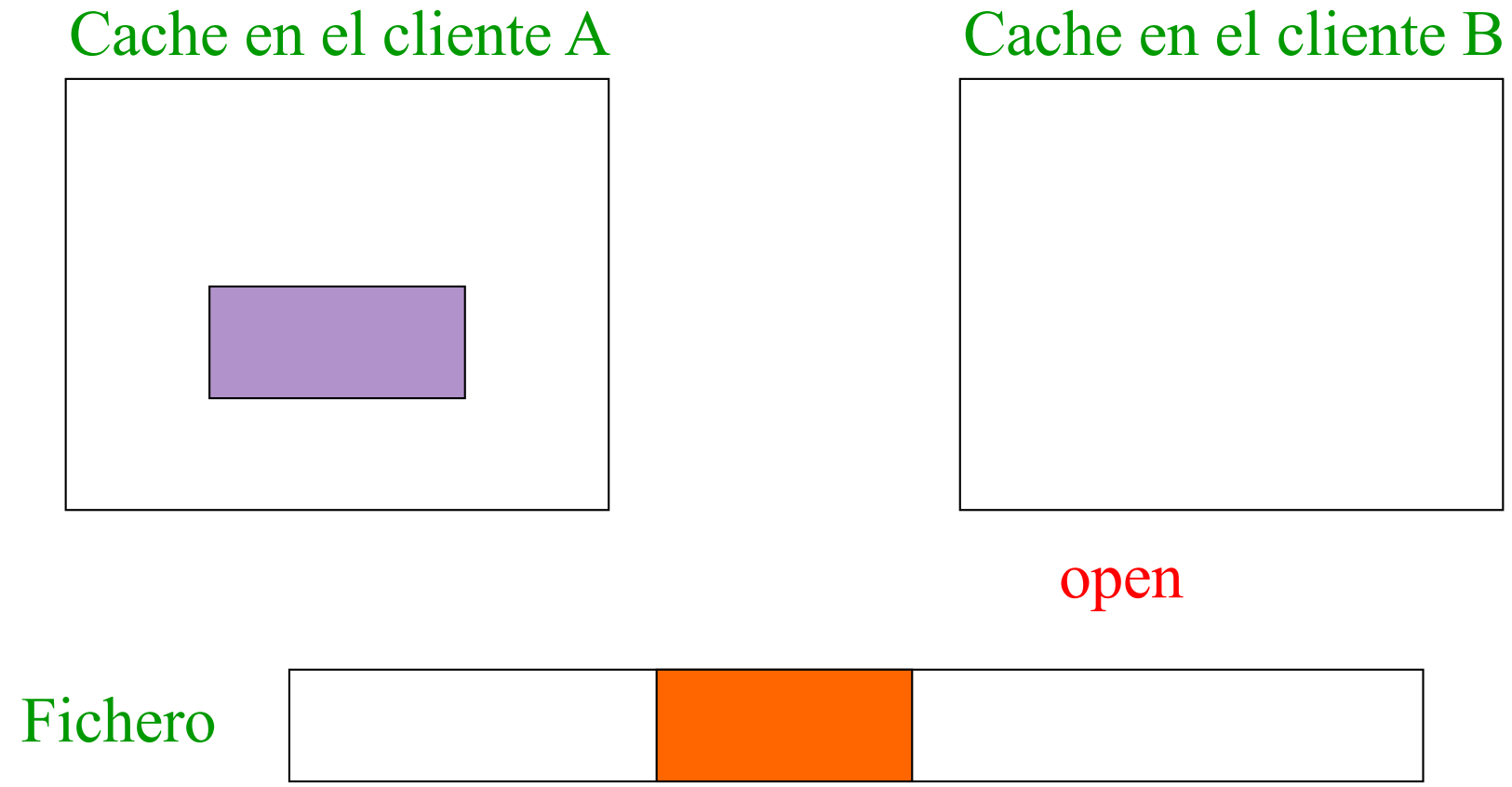
Problema de la coherencia de caché

Inconsistencia en escritura secuencial debido a la utilización de técnicas de escritura diferida



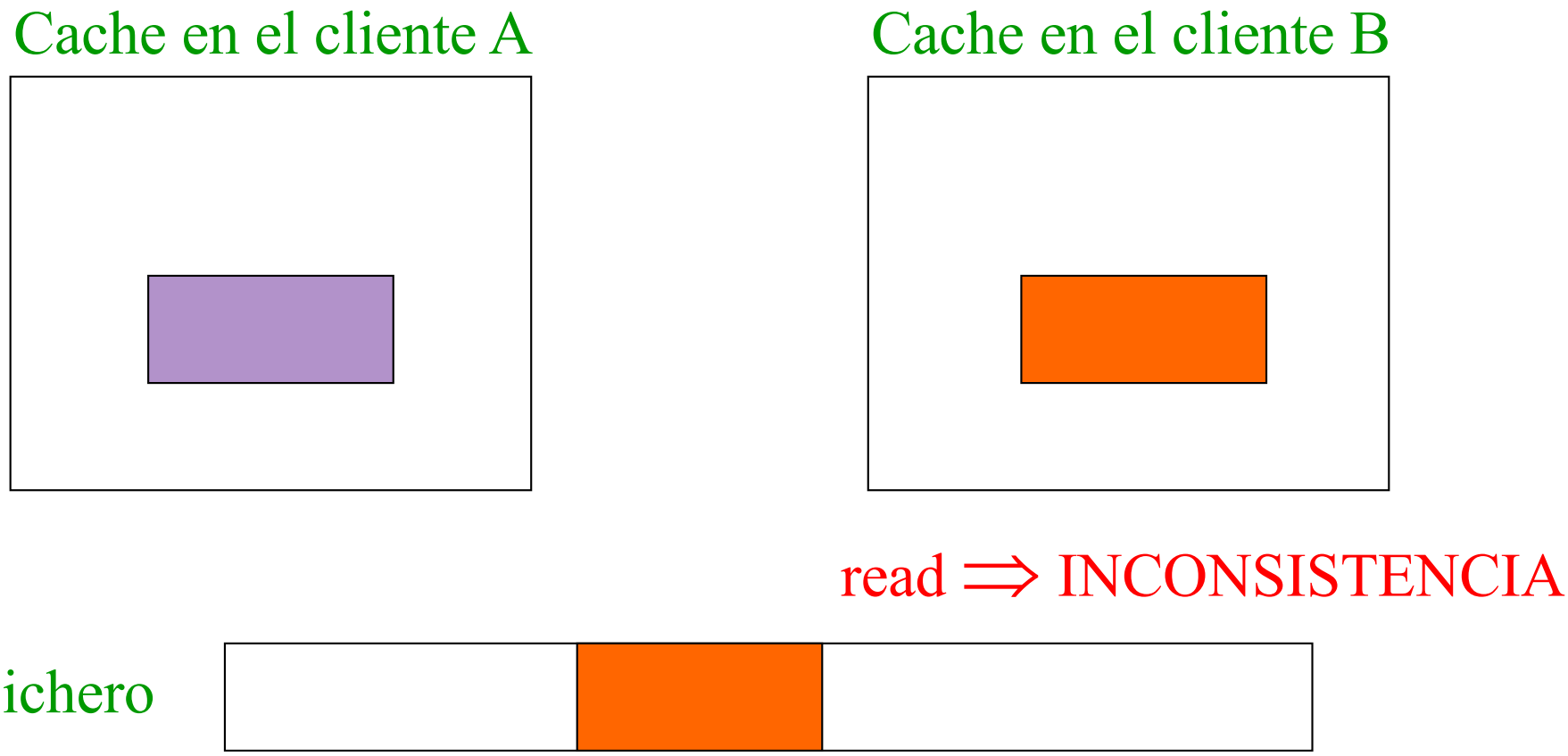
Problema de la coherencia de caché

Inconsistencia en escritura secuencial debido a la utilización de técnicas de escritura diferida



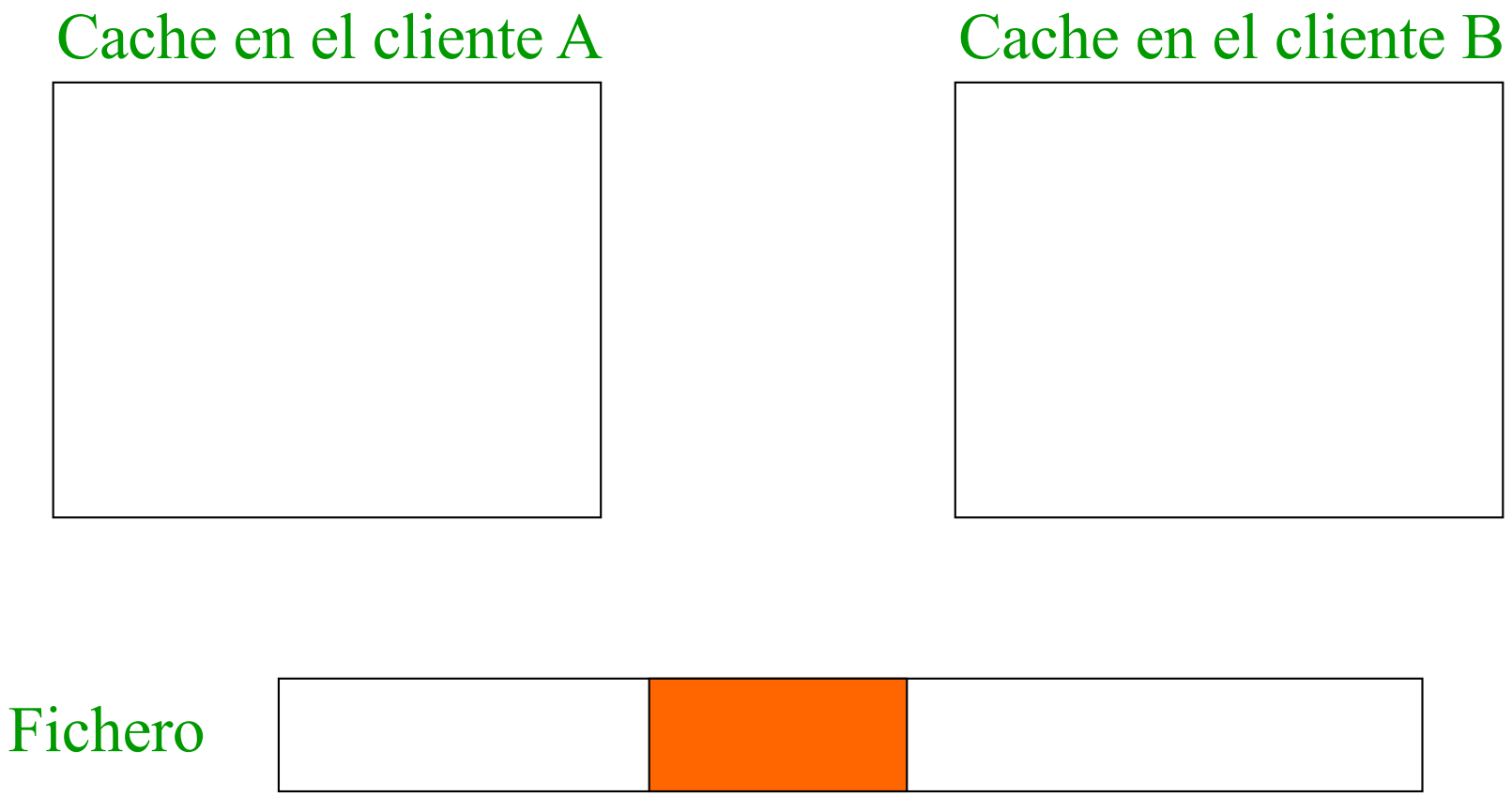
Problema de la coherencia de caché

Inconsistencia en escritura secuencial debido a la utilización de técnicas de escritura diferida



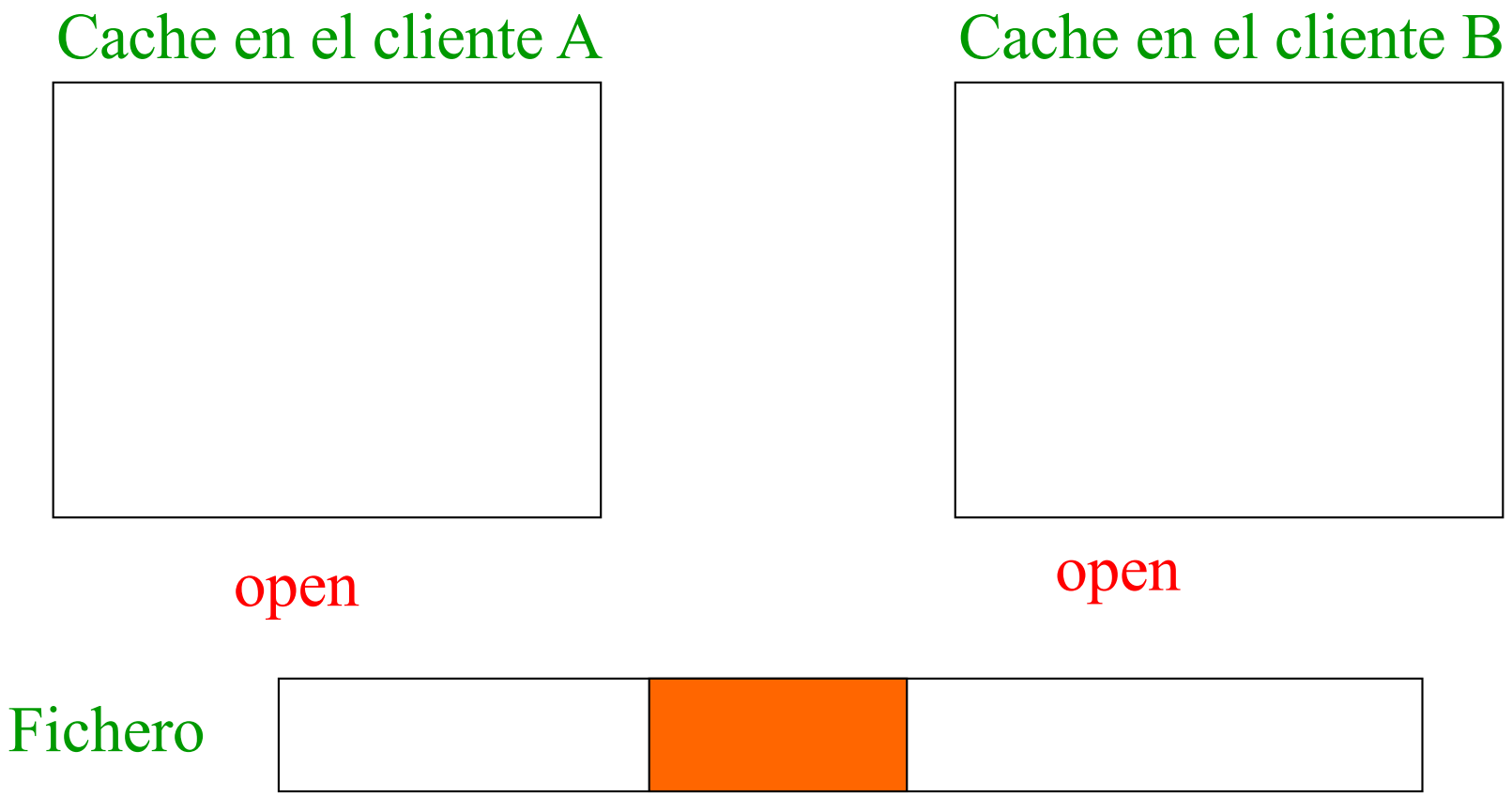
Problema de la coherencia de caché

Inconsistencia debido a la coutilización en escritura concurrente



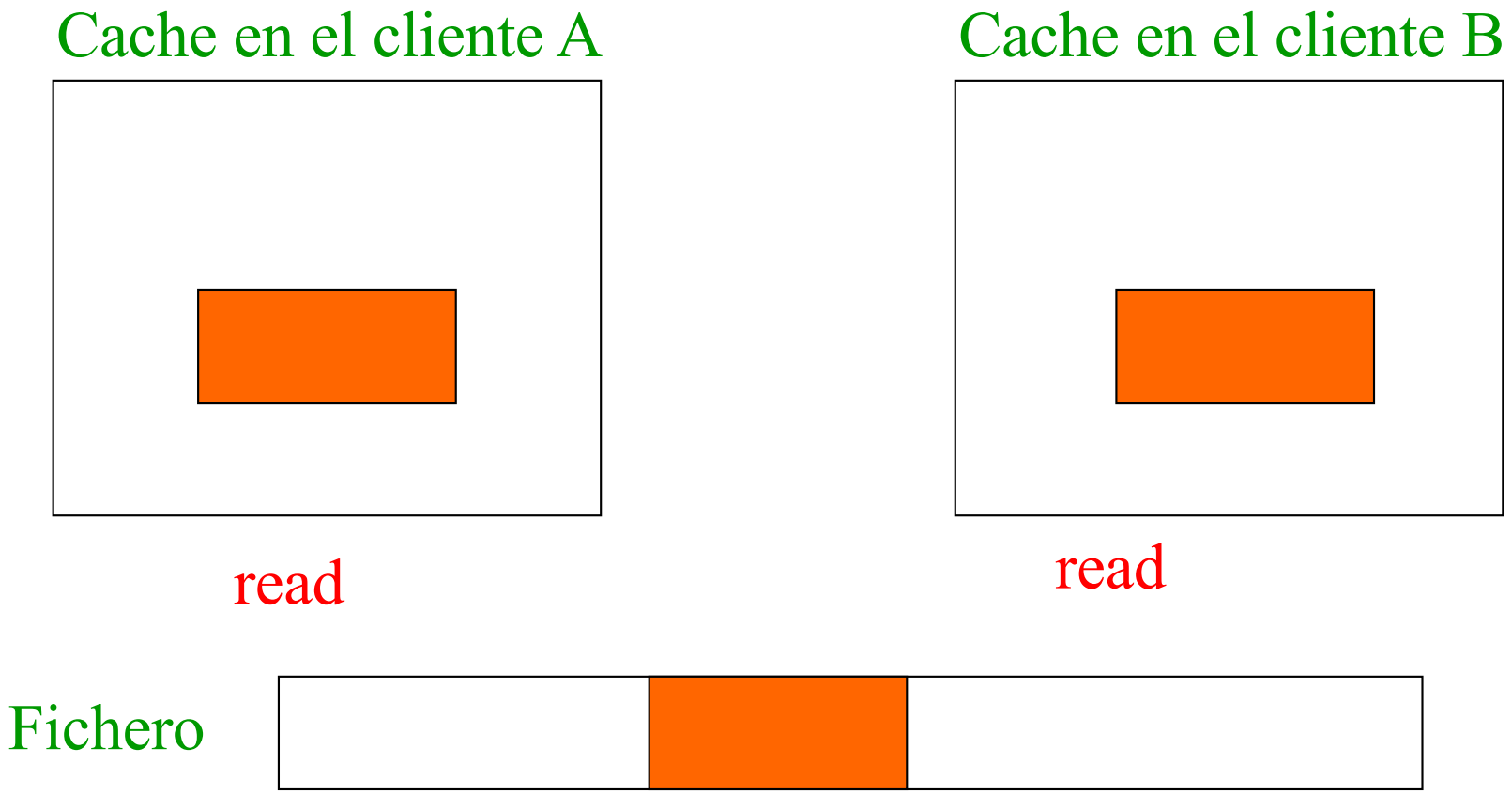
Problema de la coherencia de caché

Inconsistencia debido a la colutilización en escritura concurrente



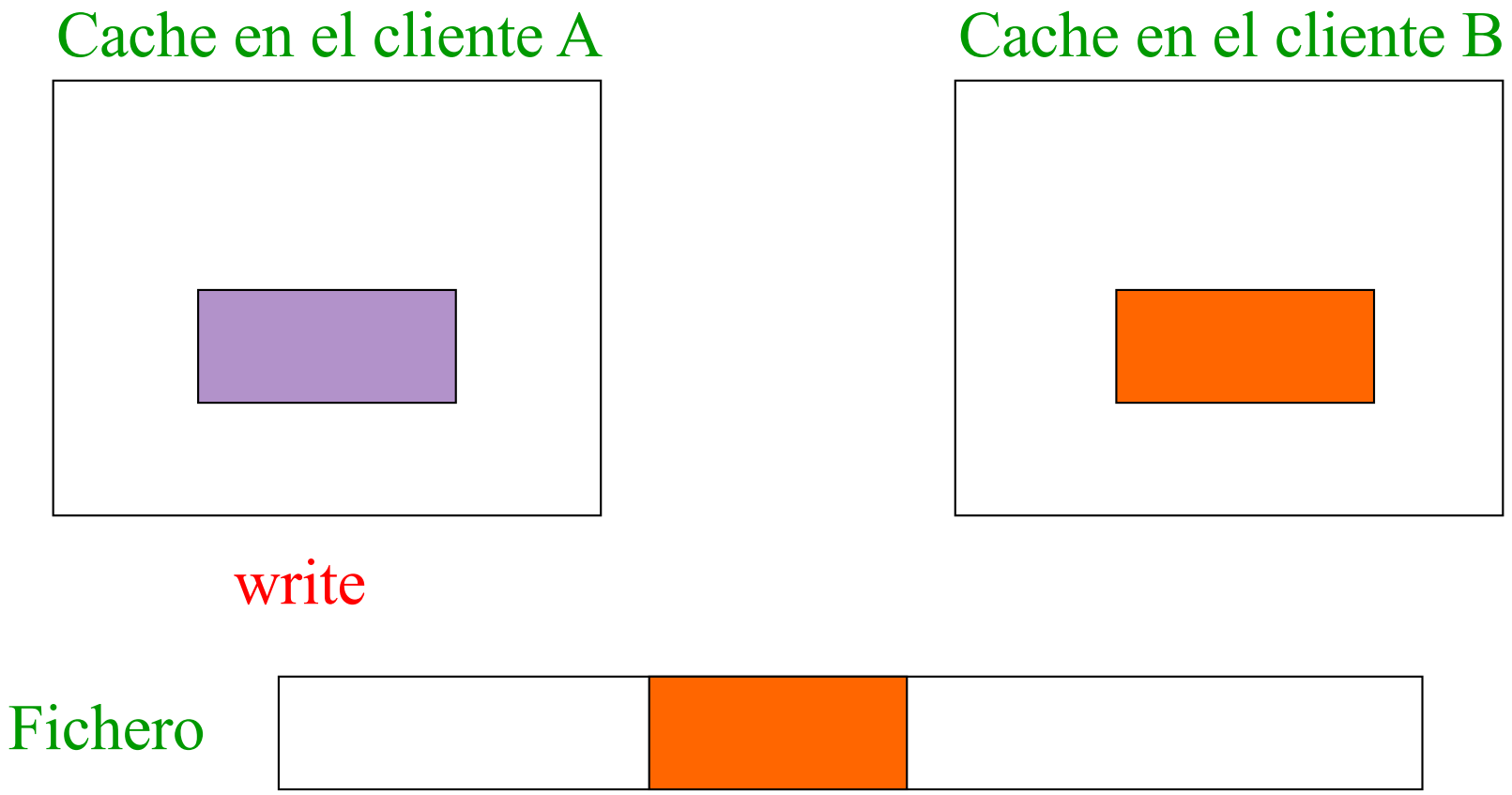
Problema de la coherencia de caché

Inconsistencia debido a la colutilización en escritura concurrente



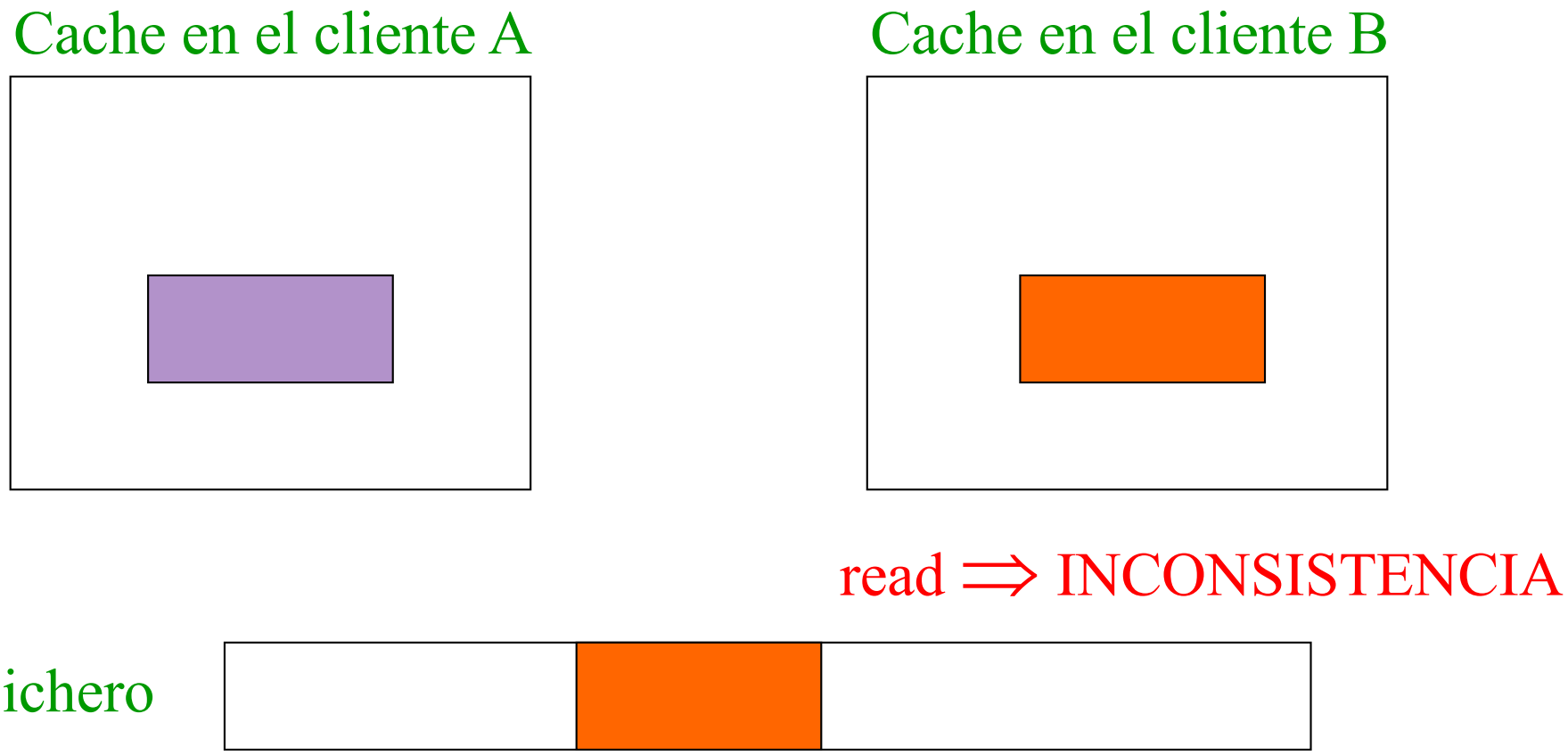
Problema de la coherencia de caché

Inconsistencia debido a la colutilización en escritura concurrente



Problema de la coherencia de caché

Inconsistencia debido a la colutilización en escritura concurrente



Soluciones al problema de la coherencia

- No emplear caché en los clientes
  - Solución trivial que no permite explotar las ventajas del uso de caché en los clientes (reutilización, lectura adelantada y escritura diferida)
- No utilizar caché en los clientes para datos compartidos en escritura (Sprite)
  - Accesos remotos sobre una única copia asegura semántica UNIX
- Mecanismos de caché sin replicación de datos
  - Basado en esquemas cooperativos que definen un único espacio global formado por la unión de todas las caché del sistema.
  - Los datos fluyen a través de las caches sin replicación
- Empleo de protocolos de coherencia de caché

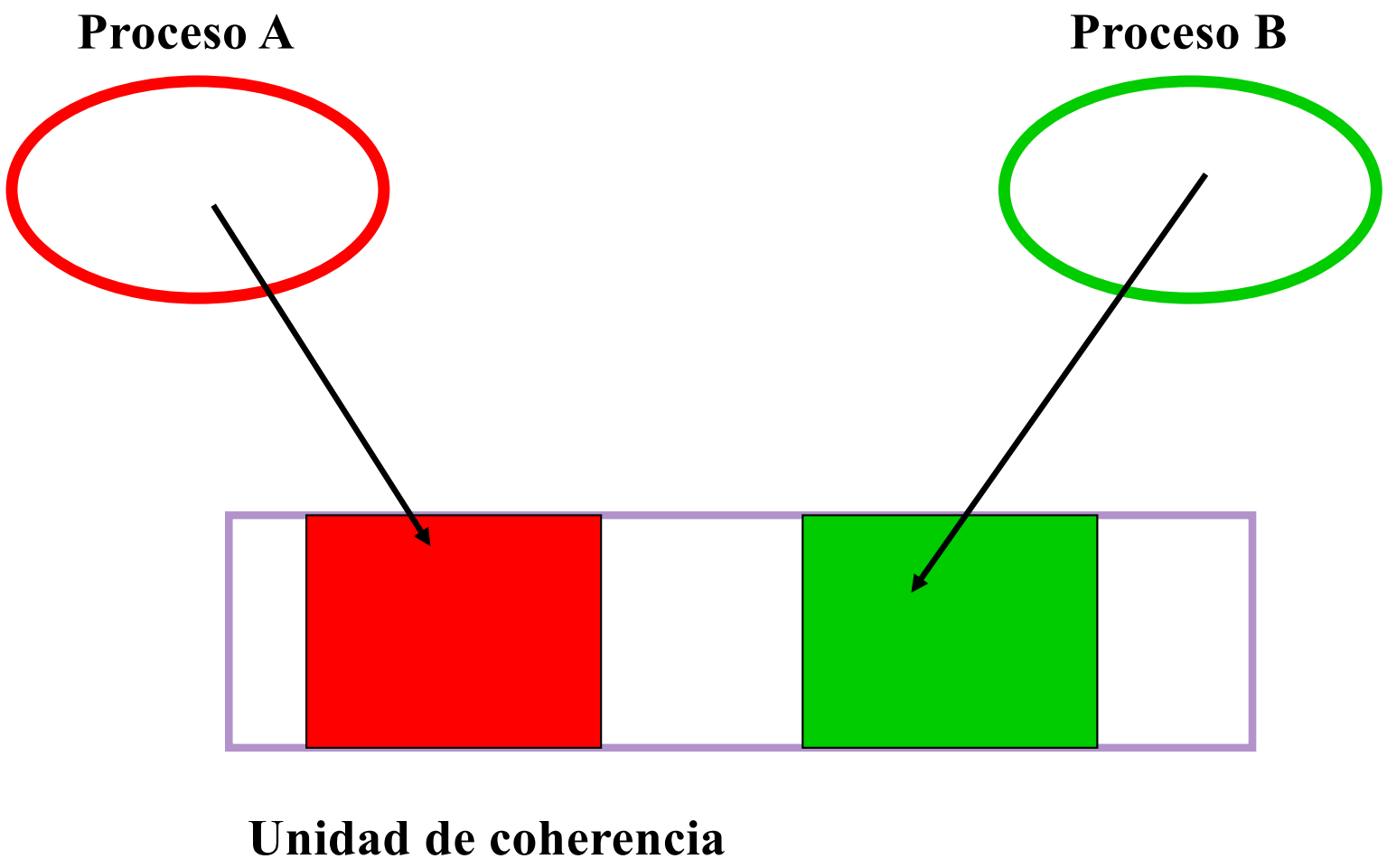
Protocolos de coherencia de cache

- Objetivos:
  - Permitir el uso de caché en los clientes manteniendo coherente la información de acuerdo a la semántica de coutilización que define el sistema de ficheros
- Aspectos de diseño a considerar
  - Granularidad del protocolo
  - Mecanismo de validación
  - Mecanismos de actualización
  - Localización de las copias en las caches de los clientes

Granularidad del protocolo

- Tamaño de la **unidad** sobre la que se mantiene coherencia
  - Puede variar desde un único byte hasta el fichero completo
- Ejemplos:
  - *Sprite*: fichero completo
  - *Coda*: conjunto de ficheros (volumen)
  - *ParFiSys*: regiones de tamaño y forma variable definidas por el usuario
- Con semánticas UNIX, unidades grandes y acceso concurrentes:
  - Posible falsa coutilización

Falsa coutilización



Validación de la cache

- Determinar si un dato almacenado en la caché es consistente
- Validación **iniciada por el cliente**. Contactar con el servidor para determinar el estado de la copia
  - Frecuencia de validación
    - En cada acceso
    - Al abrir el fichero
    - Periódicamente
  - Necesidad de almacenar información sobre la última actualización en caso de escritura diferida.
  - Crece el tráfico de la red, consume CPU en el servidor y aumenta el tiempo de servicio de las peticiones

Validación de la cache

- Validación **iniciada por el servidor**
  - El servidor almacena información sobre los datos accedidos por cada cliente y el modo de acceso
  - Servidores con estado (implicaciones sobre la tolerancia a fallos)
  - Si un dato es accedido por dos o más clientes, al menos uno en modo escritura, se notifica a los clientes para que invaliden o actualicen las caches
  - Rompe el modelo cliente/servidor

Mecanismos de actualización

- Cuando hay una modificación sobre una copia y es necesario mantener una visión coherente hay que actualizar el resto de copias.
- Métodos:
  - **Actualización de las copias**
  - **Invalidación de las copias**

Mecanismos de actualización

- Actualización de todas las copias
  - Excesivo tráfico en la red
  - Inviable en sistemas de ficheros
- Invalidación de las copias. Se invalidan las copias y los futuros accesos se realizan sobre una copia consistente.
  - Mensajes más cortos, menos tráfico en la red
  - Mayor capacidad de crecimiento

Localización de las copias

- Es necesario conocer qué clientes almacenan copias de datos para realizar las acciones del protocolo
- Se utilizan esquemas basados en directorios. Cada entrada almacena la lista de clientes con copias en su cache.
  - Directorio de mapa completo
  - Directorio limitado
  - Lista encadenada
- **Directorios centralizados**: un único gestor se encarga de la coherencia de todos los ficheros del sistema
  - Posible cuello de botella
- **Directorios distribuidos**: existen varios gestores
  - Mejor capacidad de crecimiento

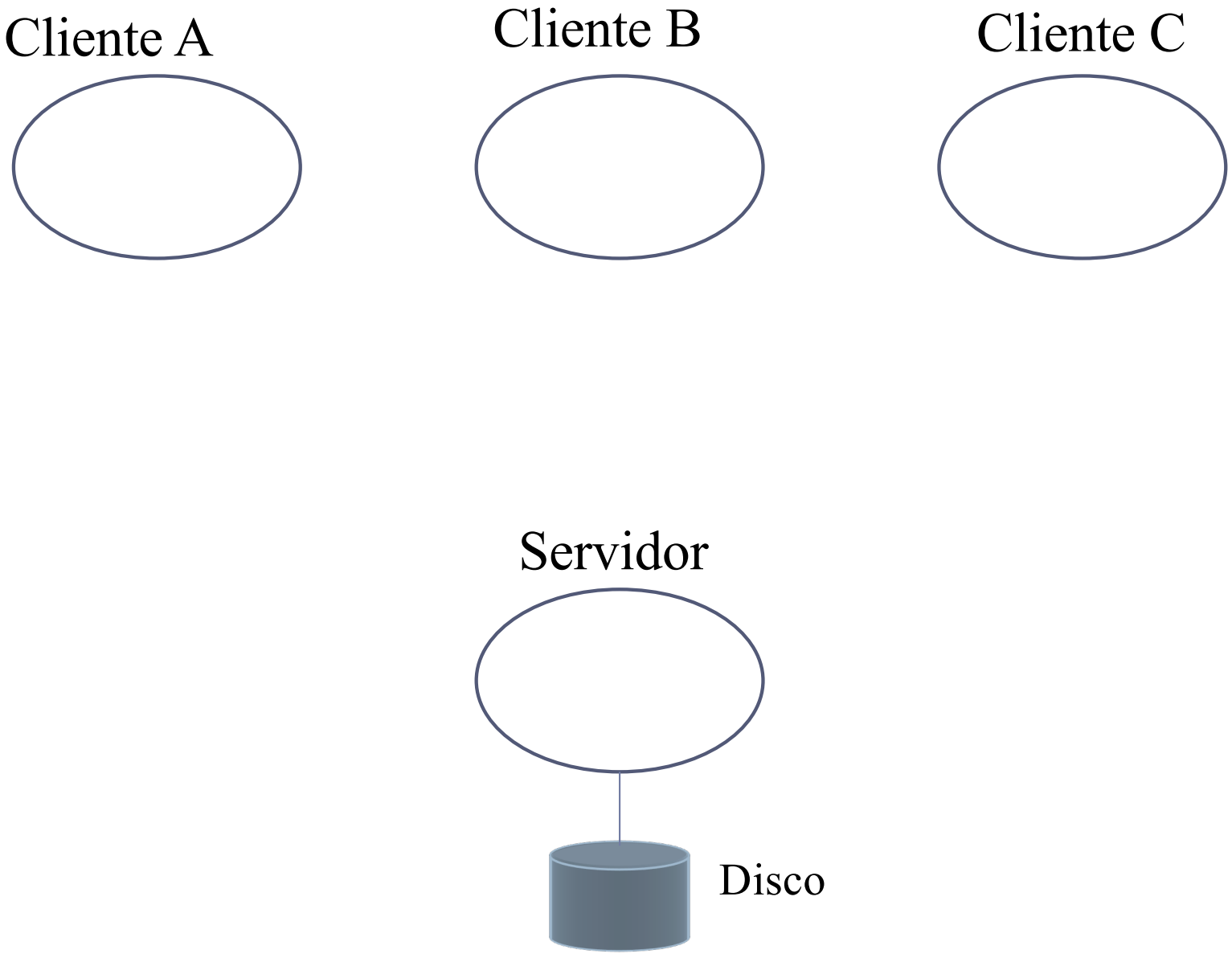
Caché en los clientes contra acceso remoto

- Rendimiento cercano al de un sistema centralizado
- Menor carga en el servidor y en la red. Se permiten transferencias más grandes por la red
- Facilita el crecimiento proporcional del rendimiento del sistema
- Dificultades relacionadas con el mantenimiento de la coherencia

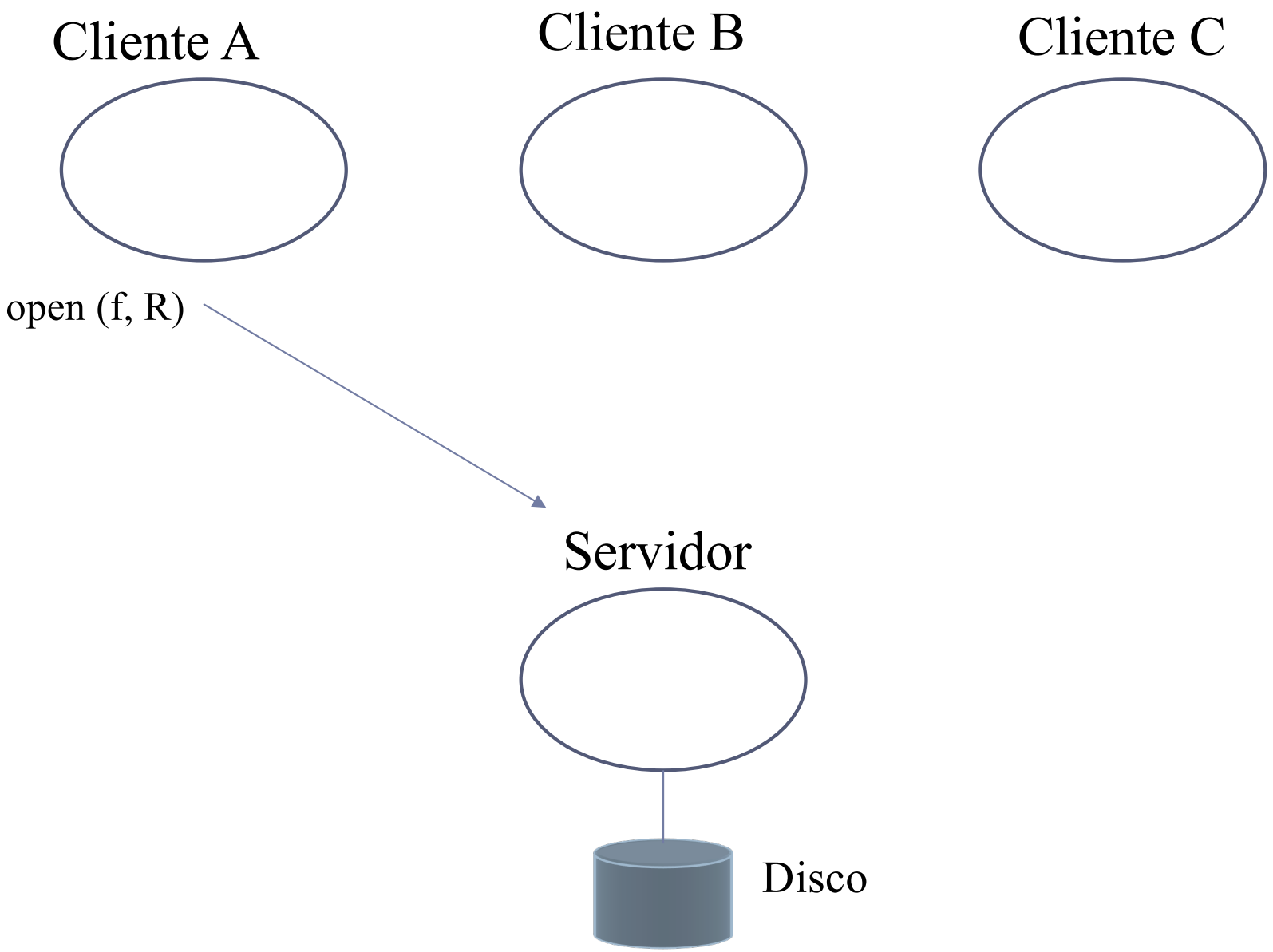
Ejemplo: coherencia de caché del SF Sprite

- Sprite: Sistema de ficheros desarrollador en Berkeley a finales de los 80
- Implementa consistencia secuencial
  - Solo se almacena en caché los datos, no los metadatos
  - Cuando un servidor detecta que un fichero está abierto en múltiples clientes (máquinas) y en alguna de ellas se abre para escritura, se desactiva la caché en todos los clientes;
    - Todas las lecturas y escritura del ficheros se hacen de forma remota
  - Si solo hay un escritor se utiliza *write-back*
- Para identificar los bloques obsoletos se utiliza
  - Número de versión para el fichero
  - Se sigue la pista del último escritor

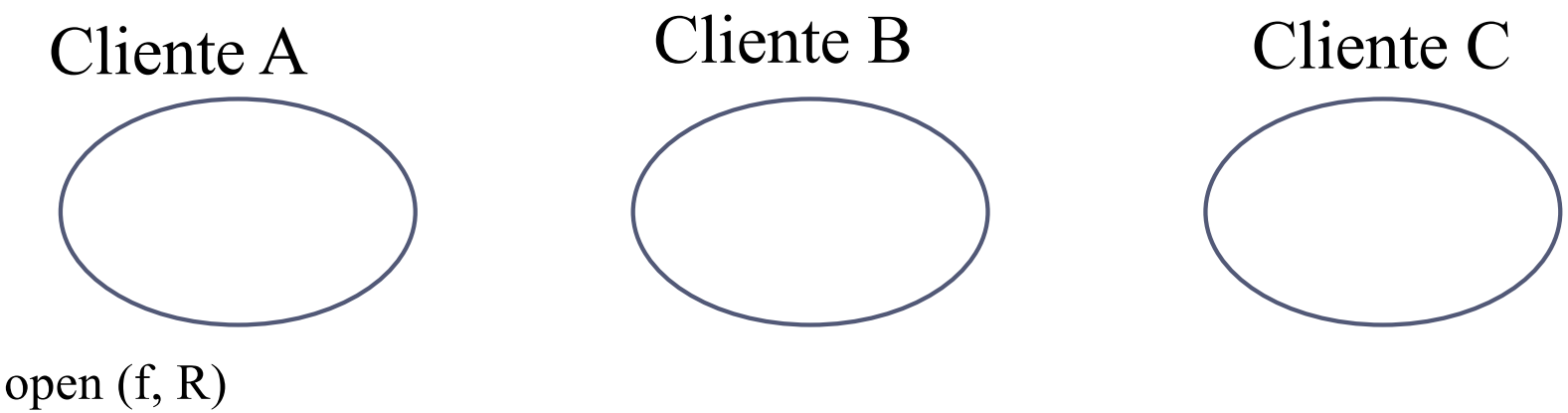
Coherencia de caché en Sprite



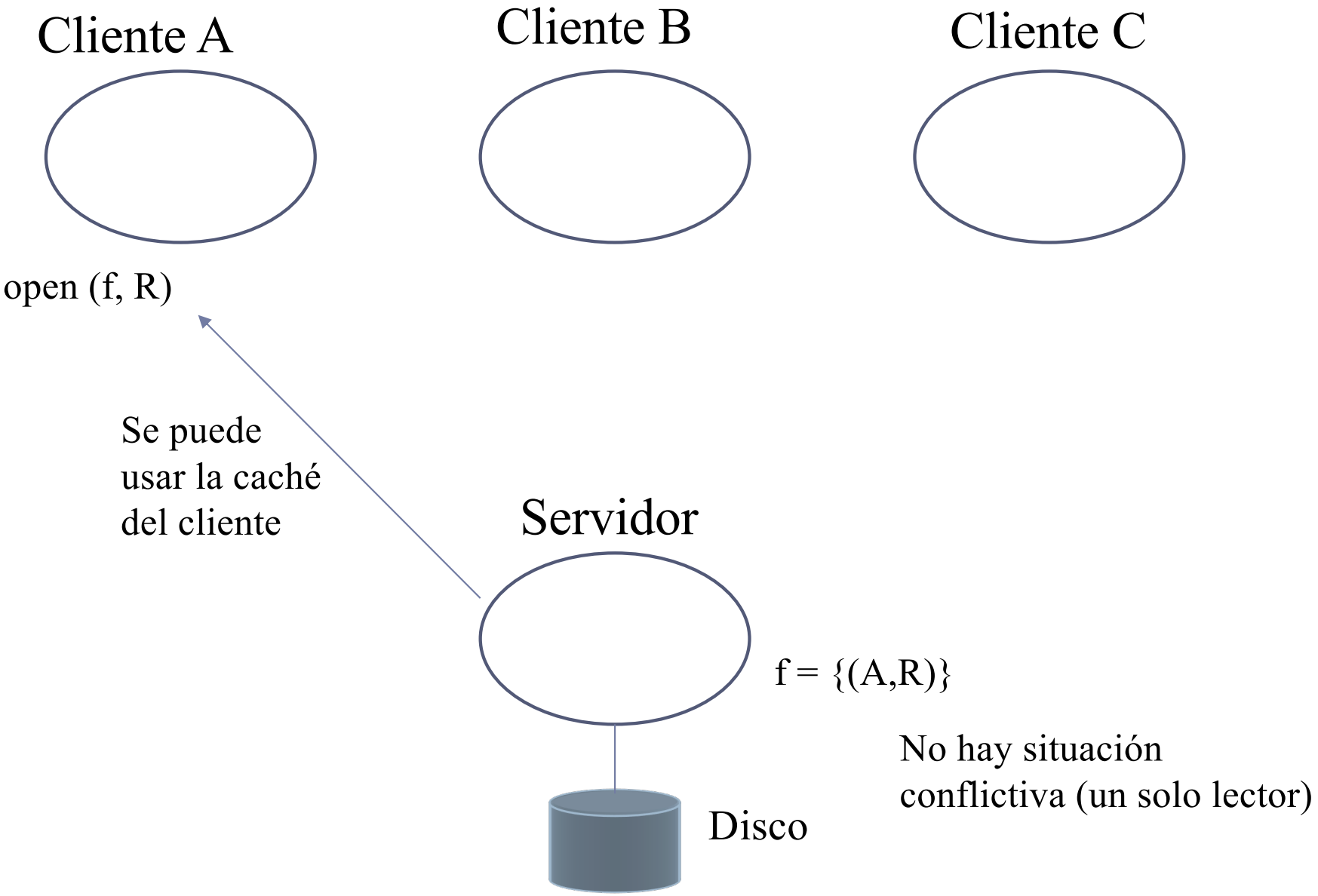
Coherencia de caché en Sprite



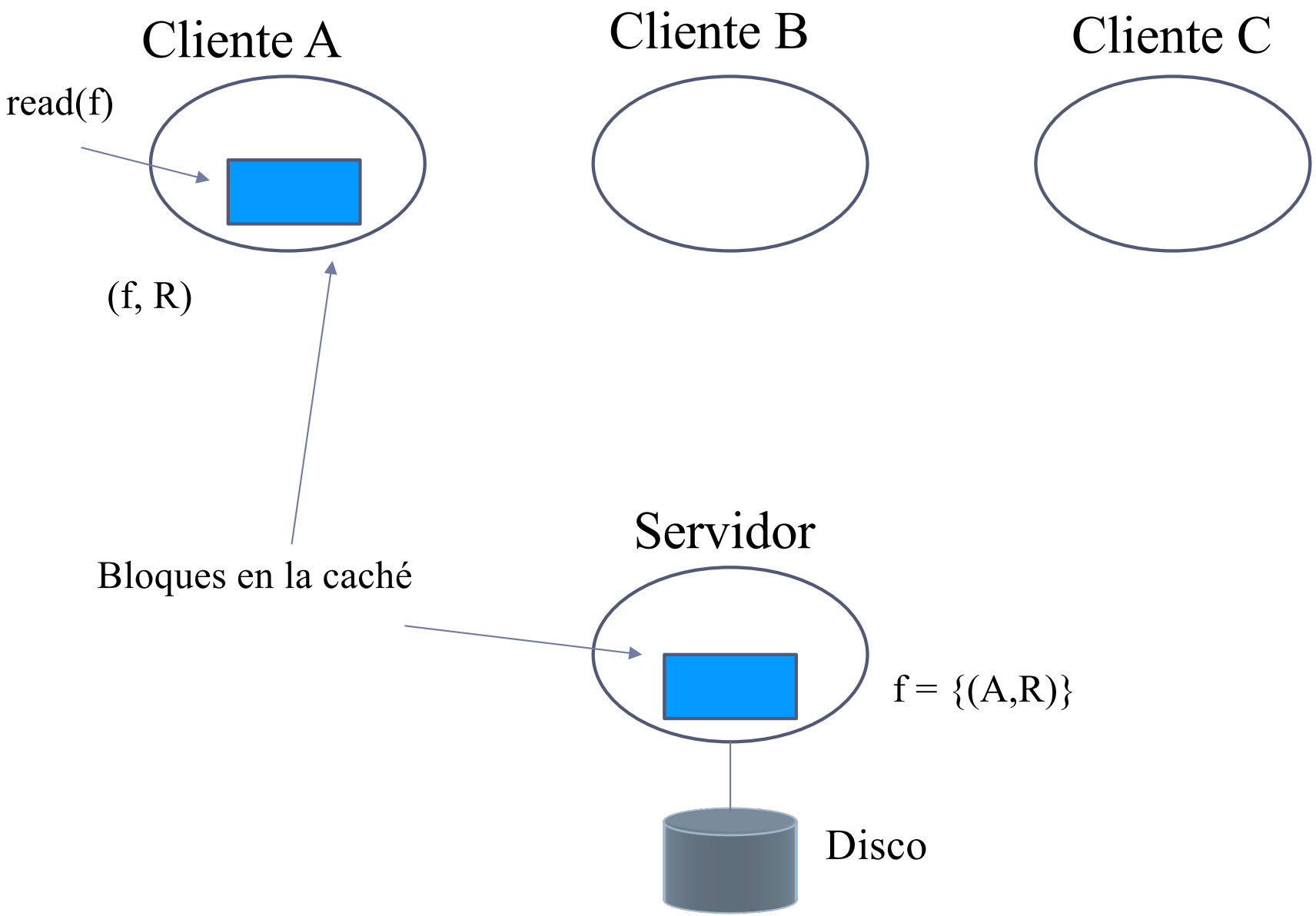
Coherencia de caché en Sprite



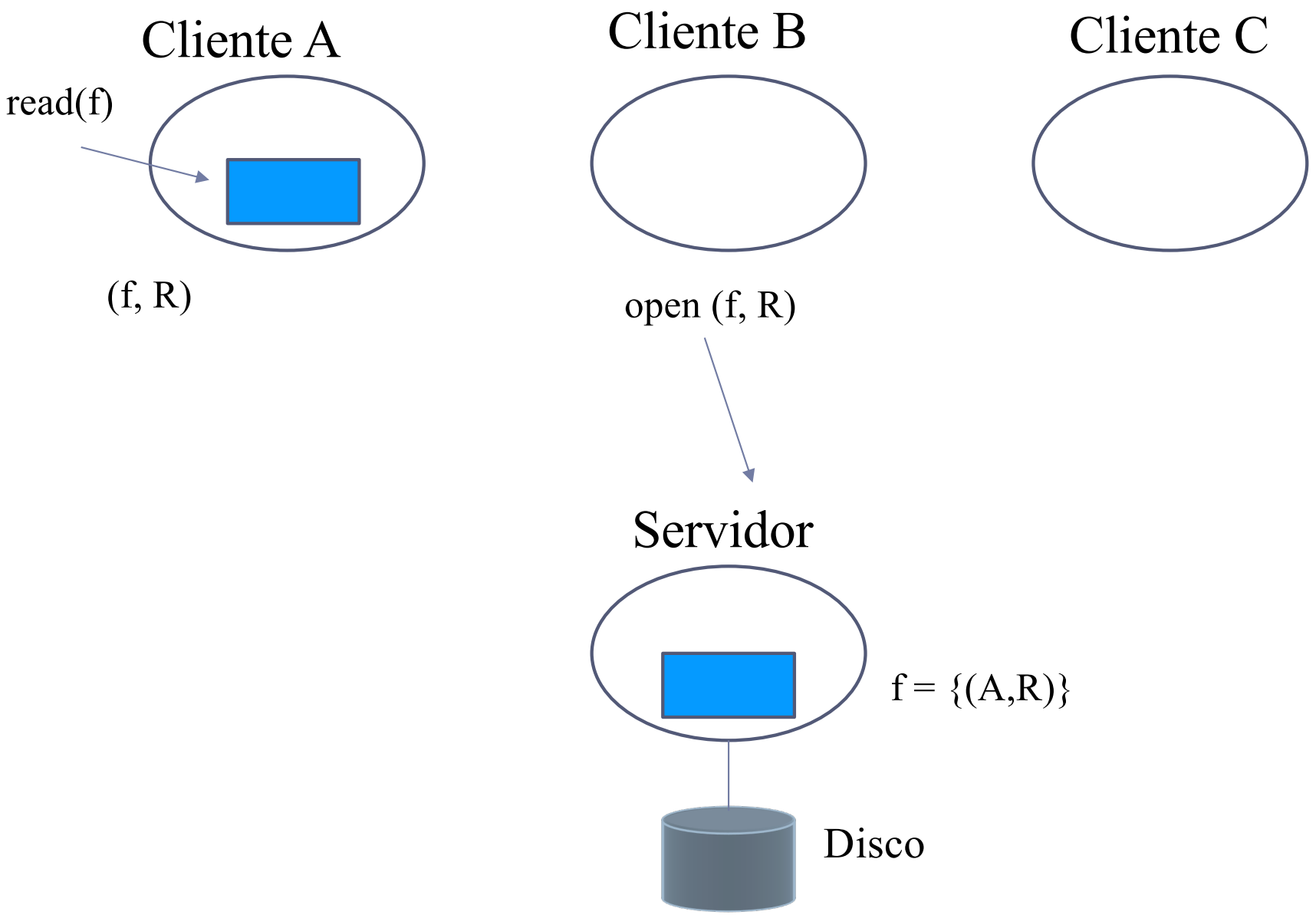
Coherencia de caché en Sprite



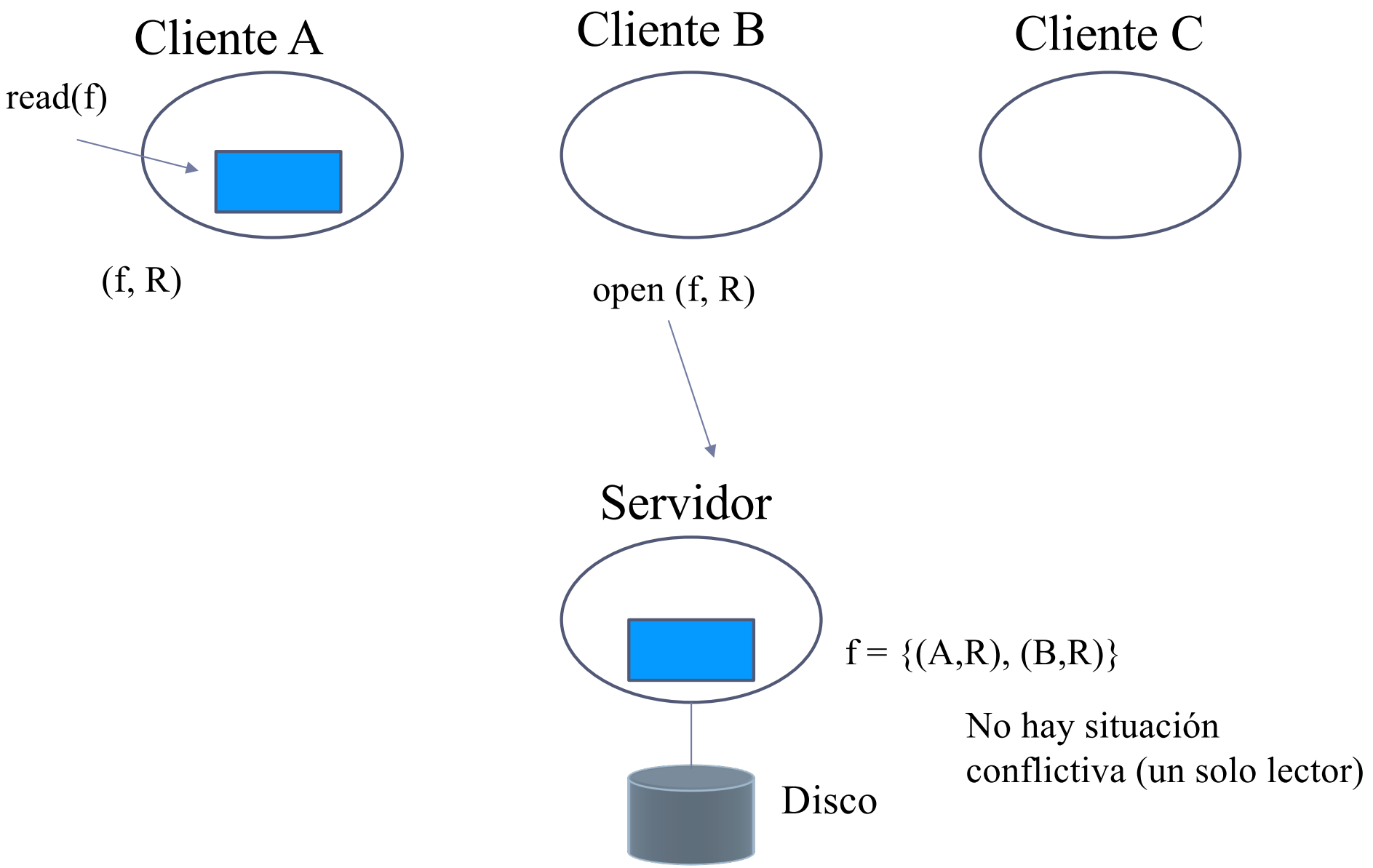
Coherencia de caché en Sprite



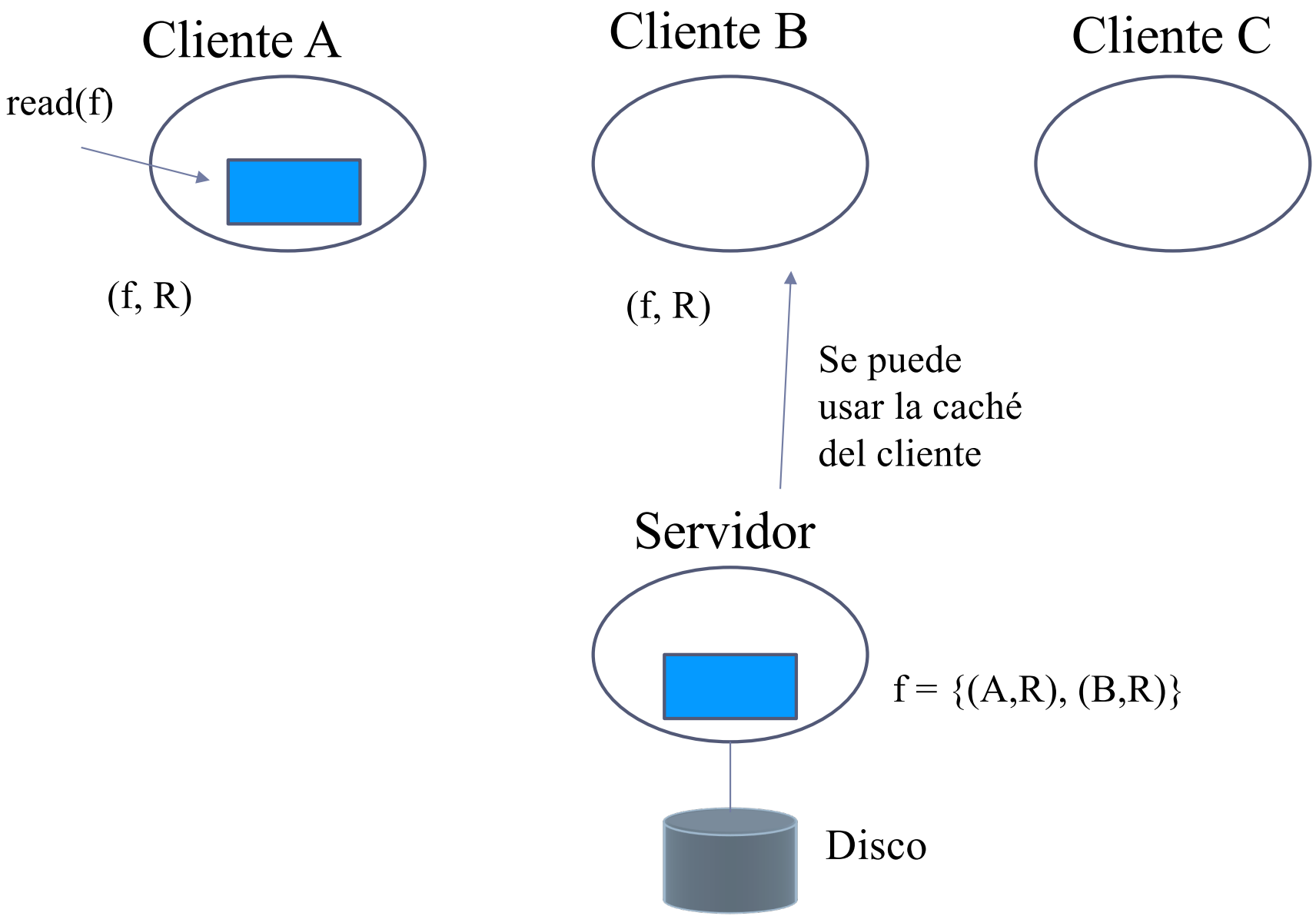
Coherencia de caché en Sprite



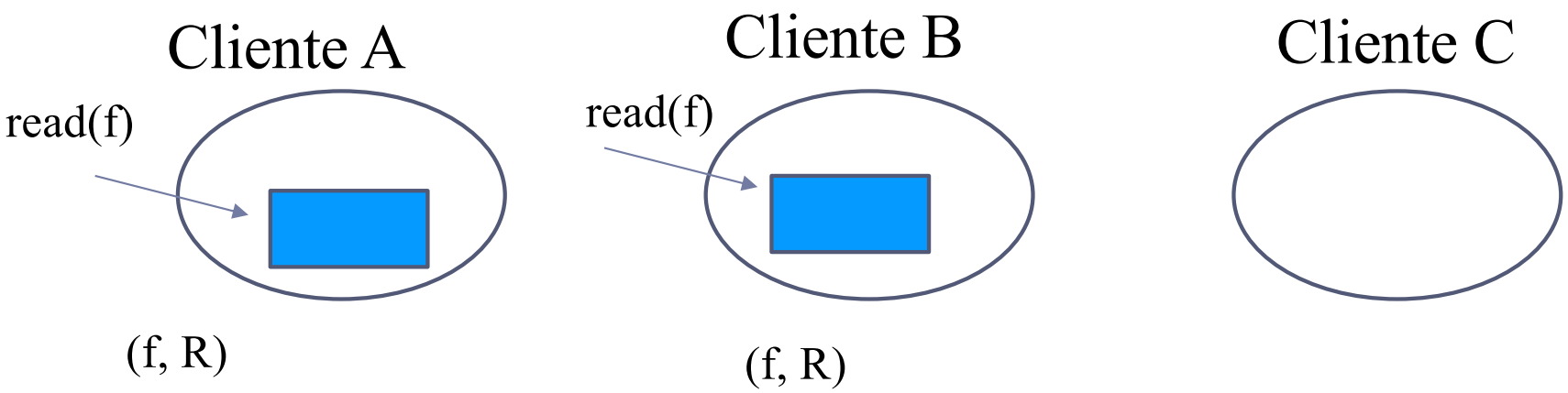
Coherencia de caché en Sprite

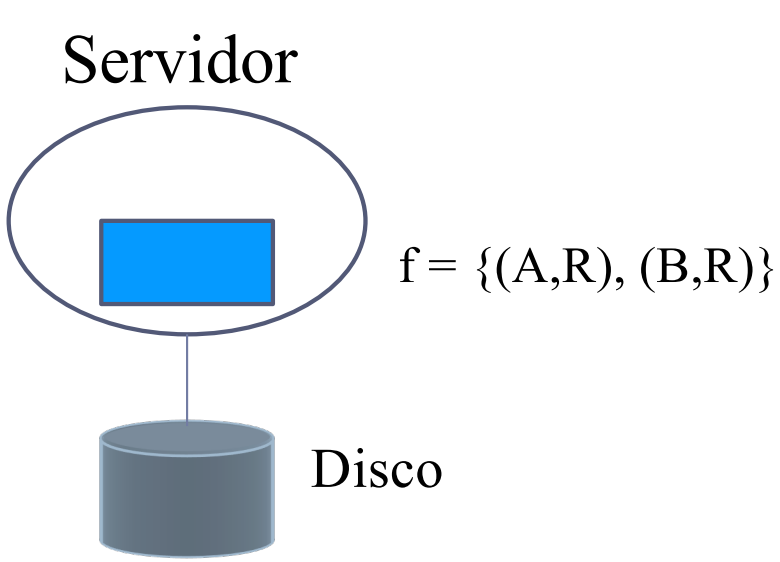


Coherencia de caché en Sprite

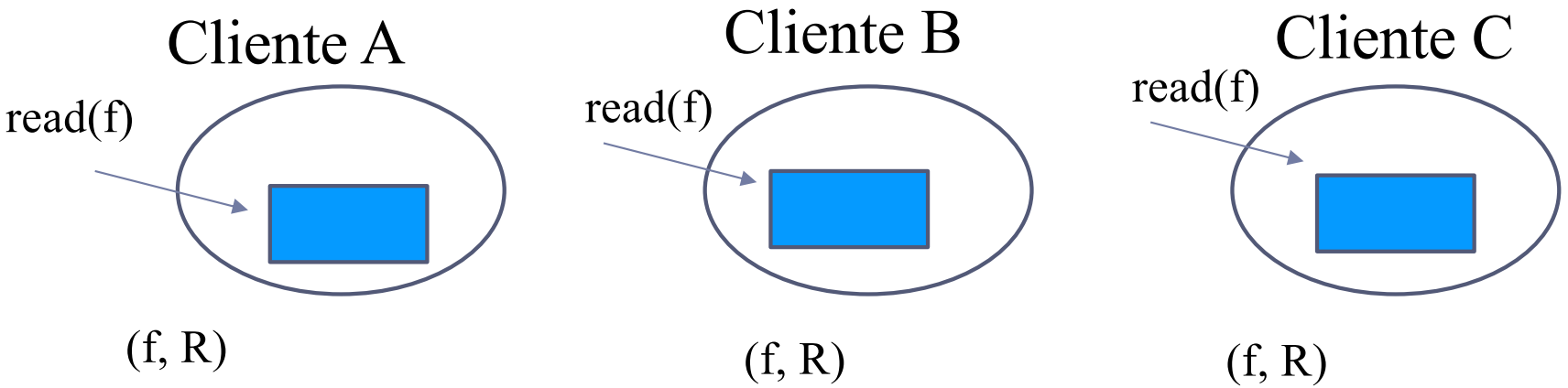


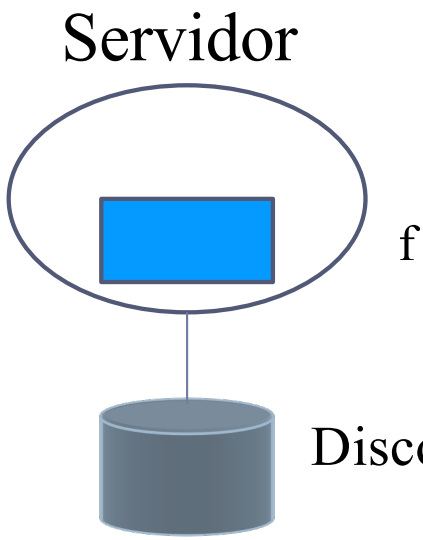
Coherencia de caché en Sprite





Coherencia de caché en Sprite

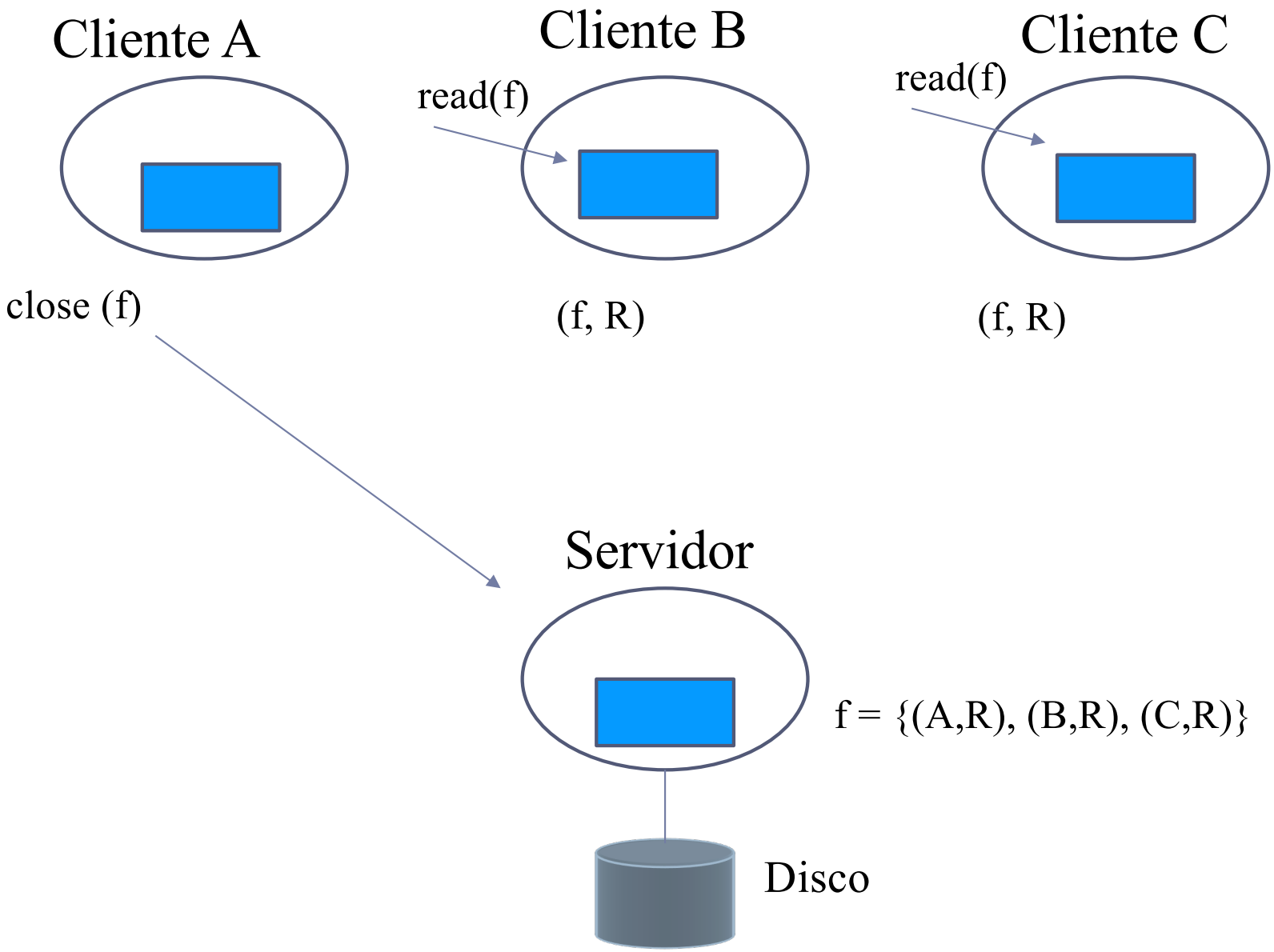




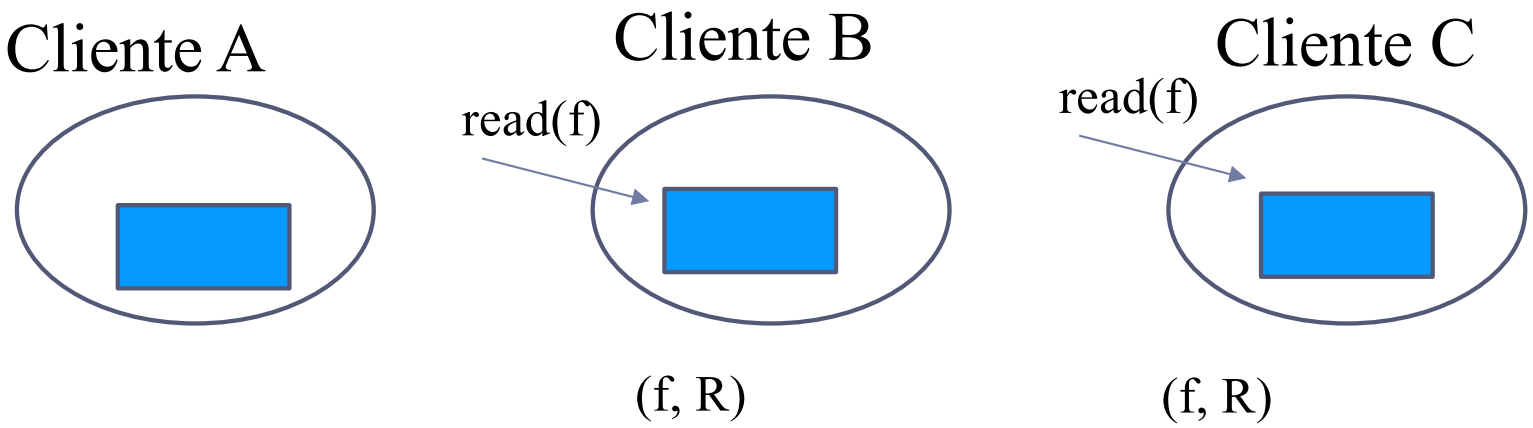
f = {(A,R), (B,R), (C,R)}

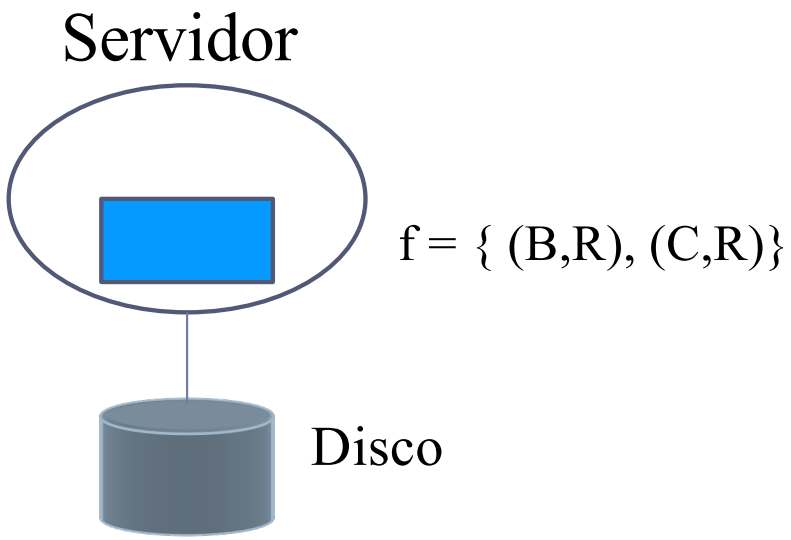
No hay situación conflictiva (todos lectores)

Coherencia de caché en Sprite

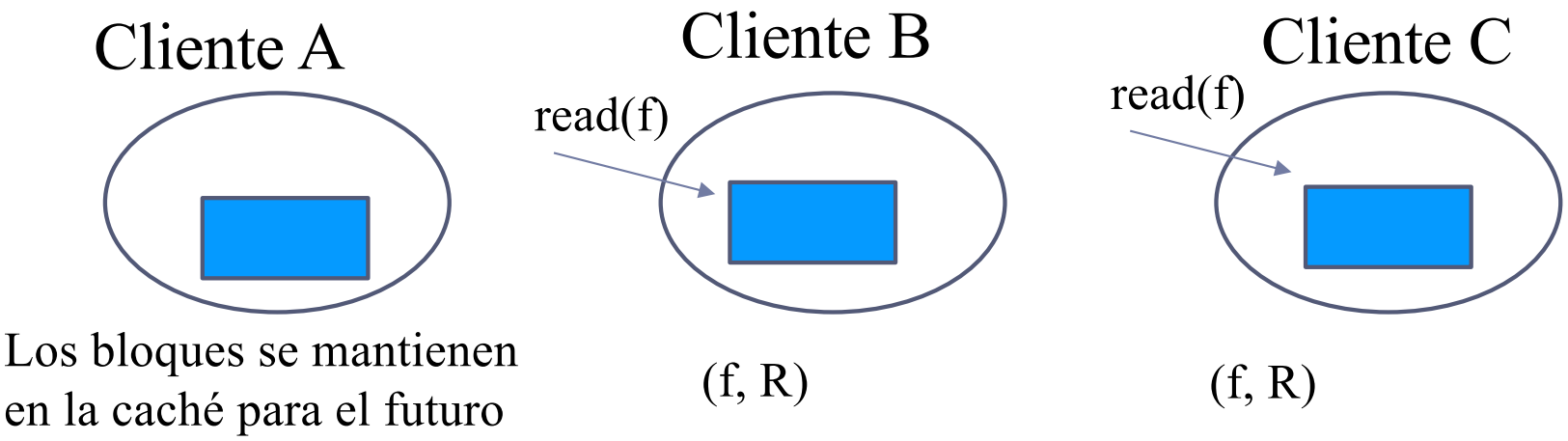


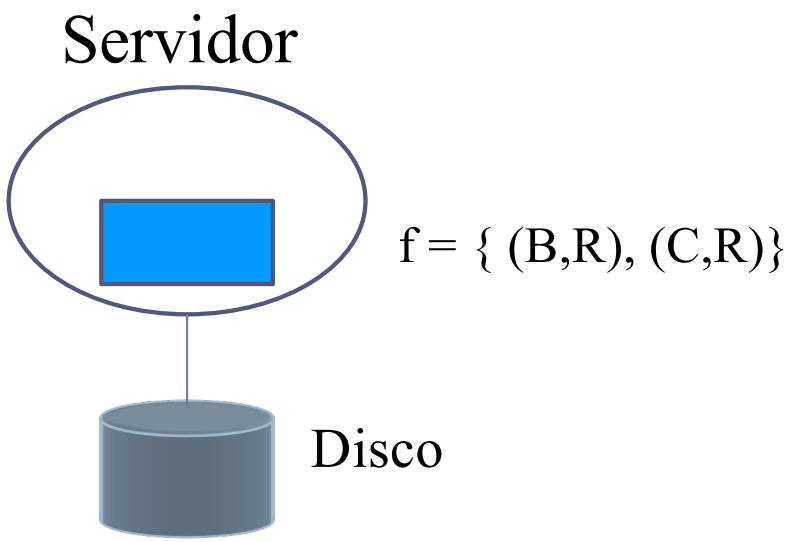
Coherencia de caché en Sprite



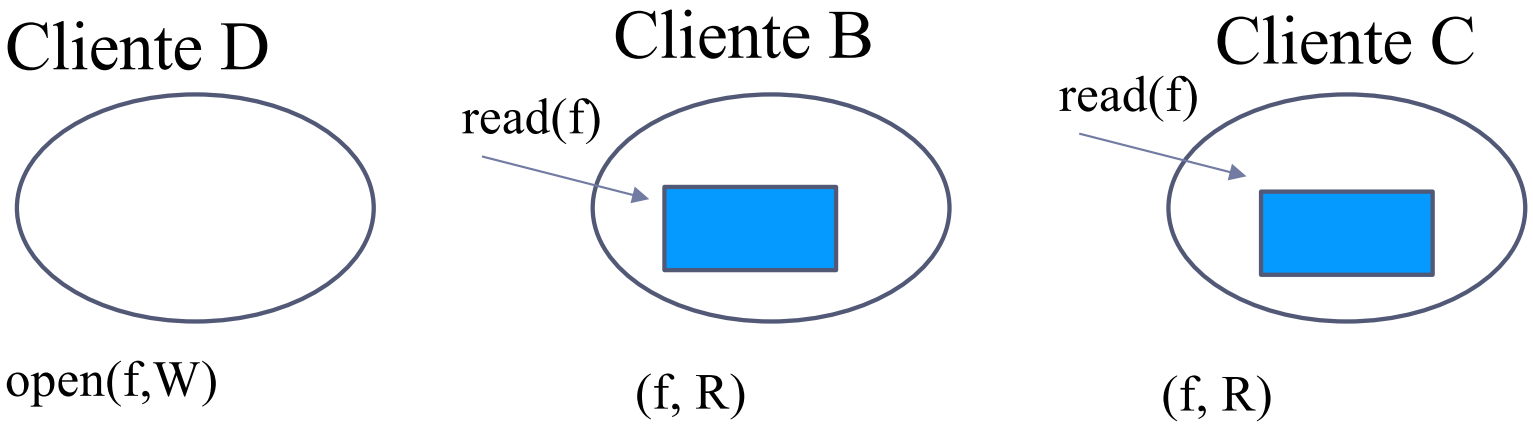


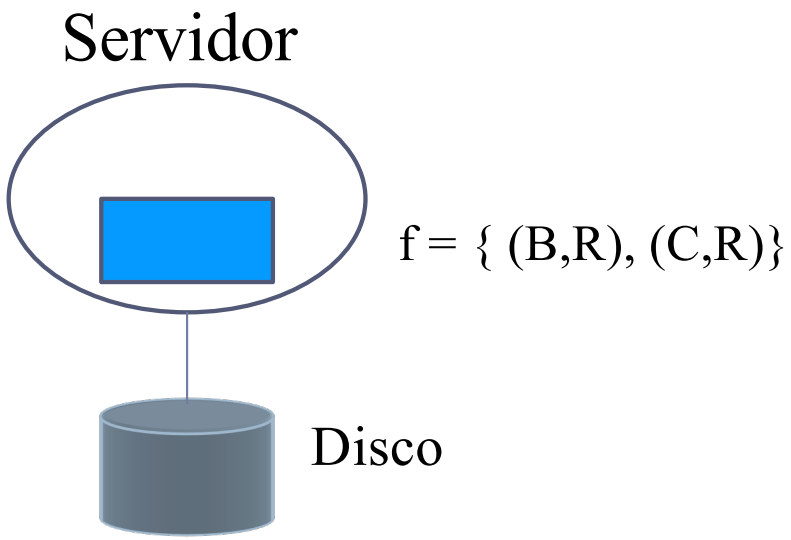
Coherencia de caché en Sprite



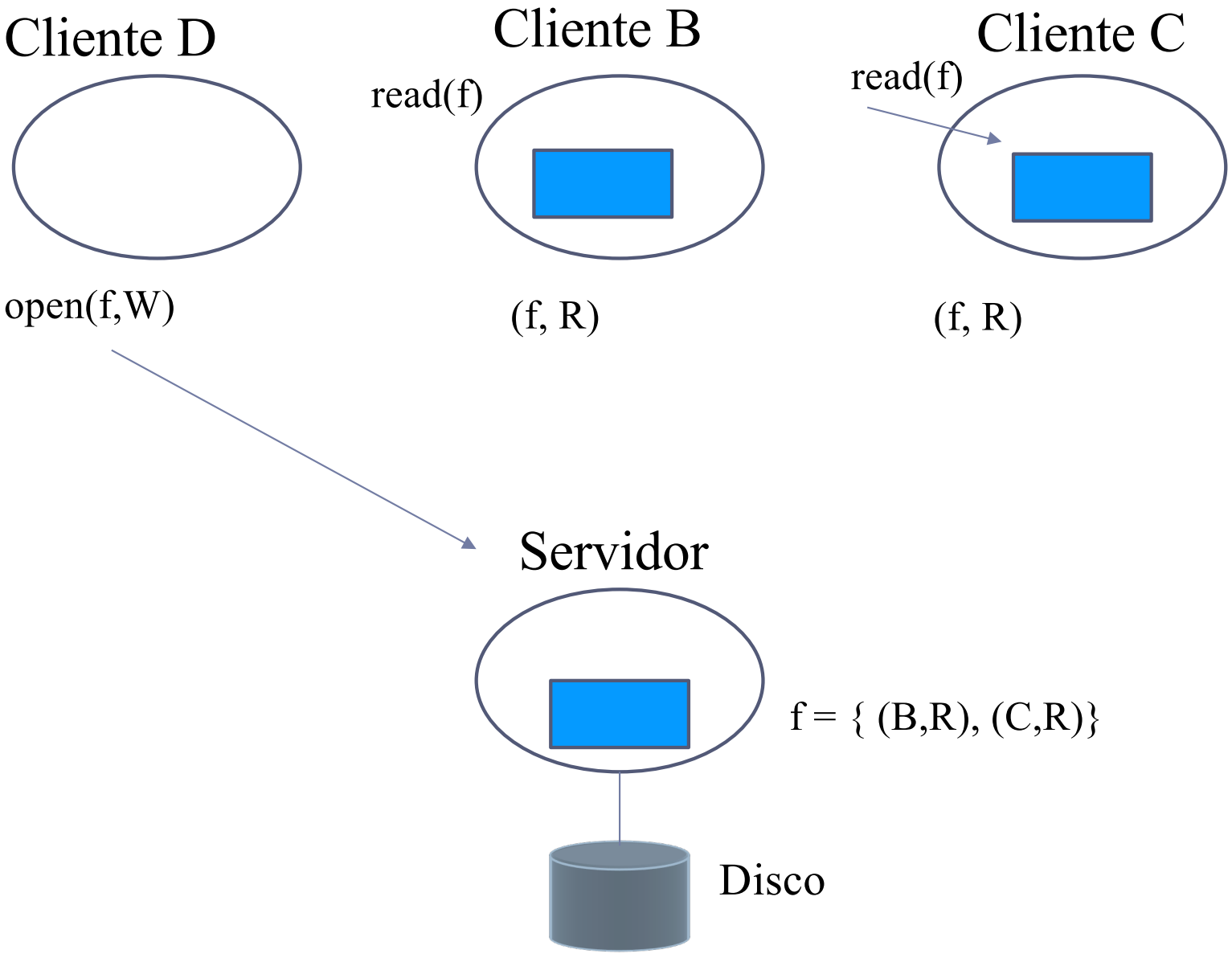


Coherencia de caché en Sprite

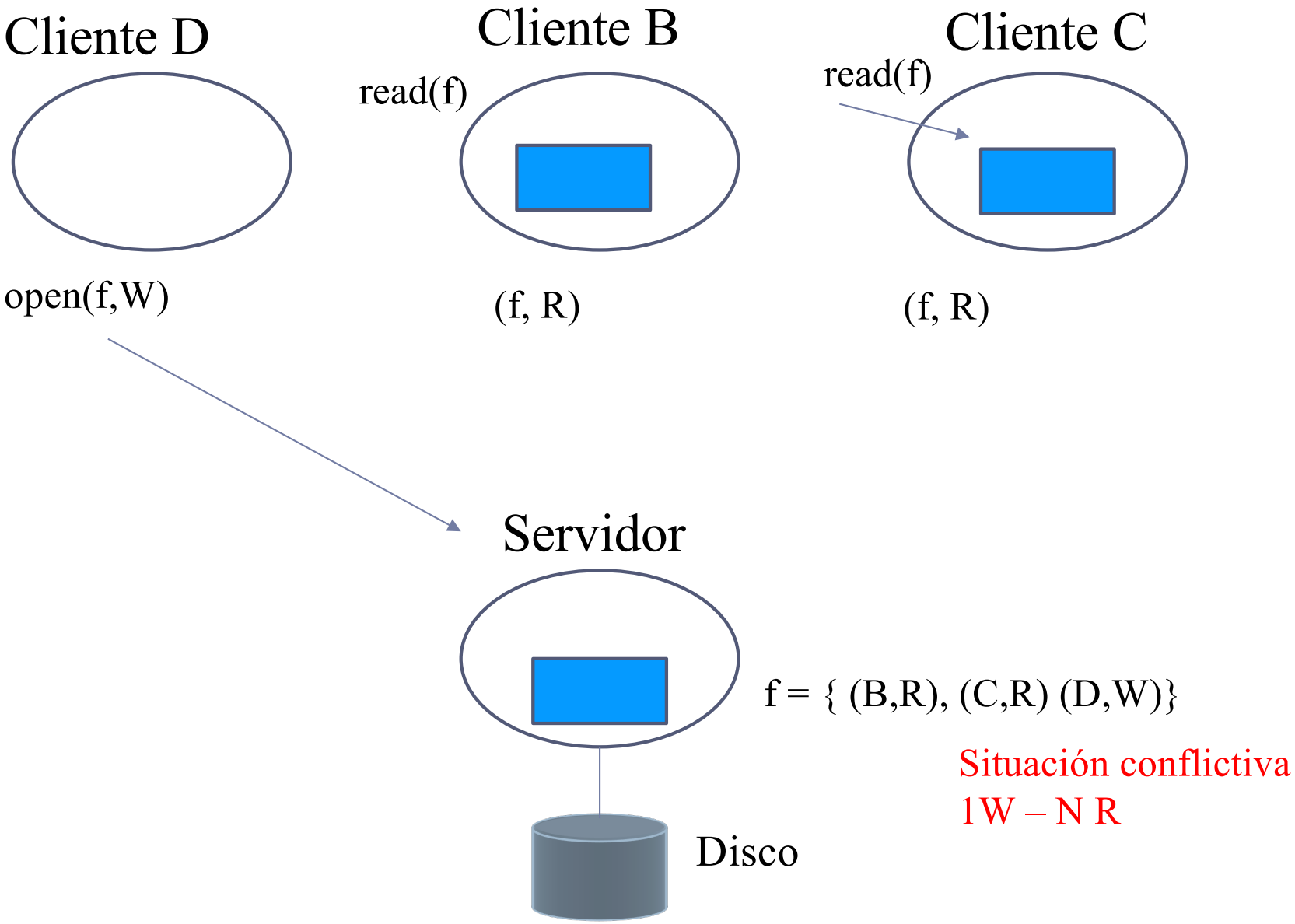




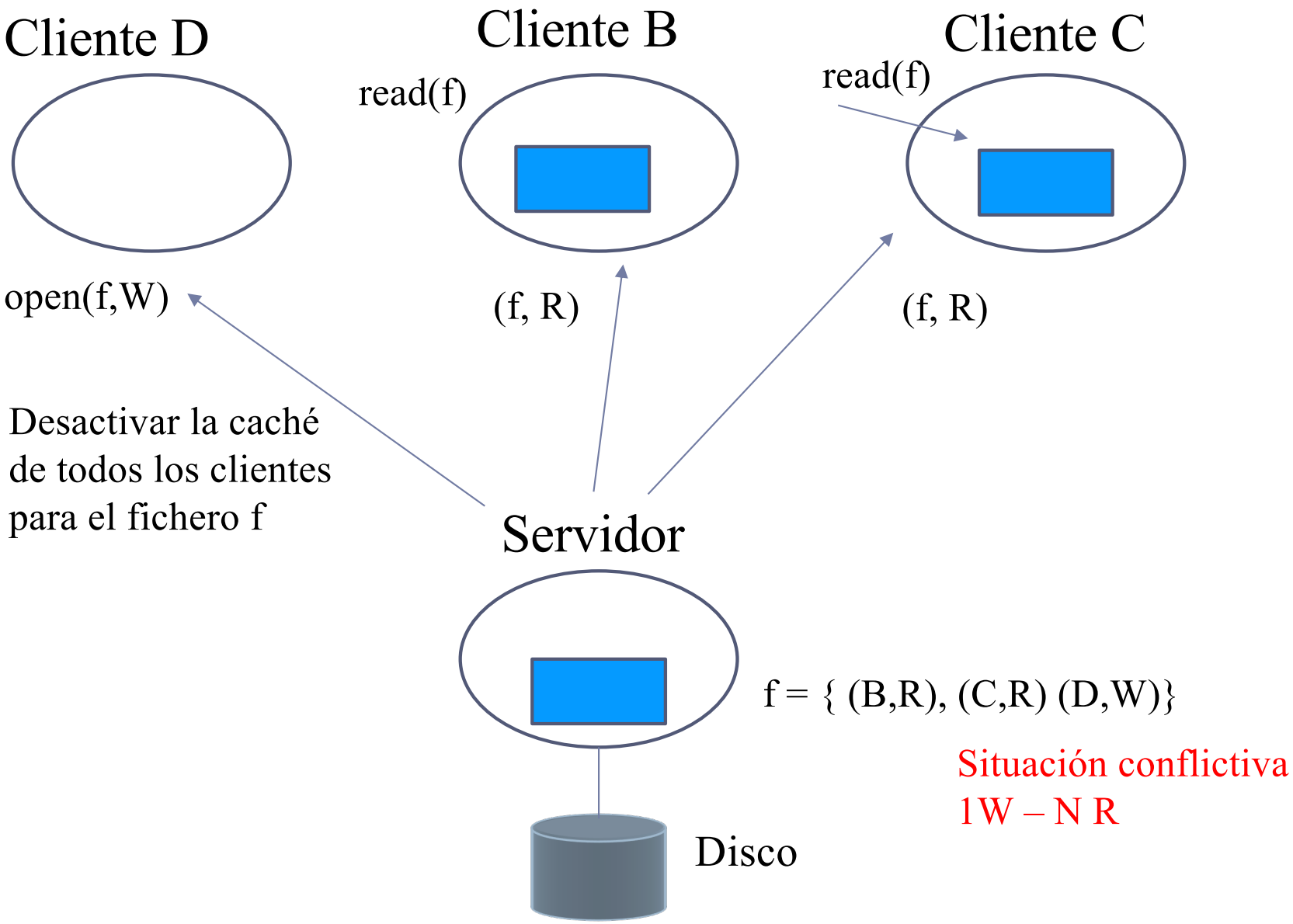
Coherencia de caché en Sprite



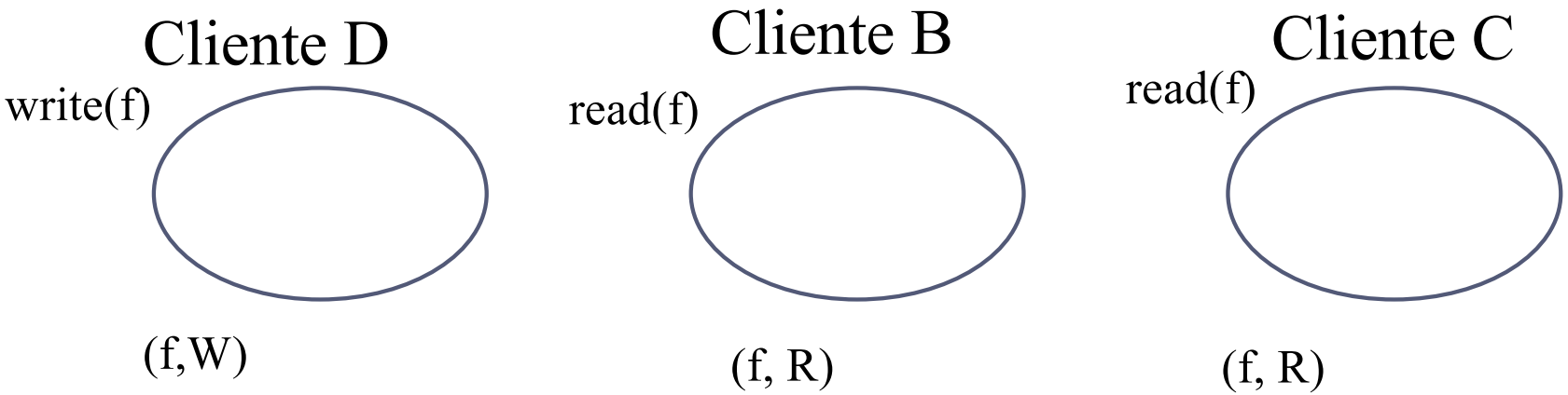
Coherencia de caché en Sprite

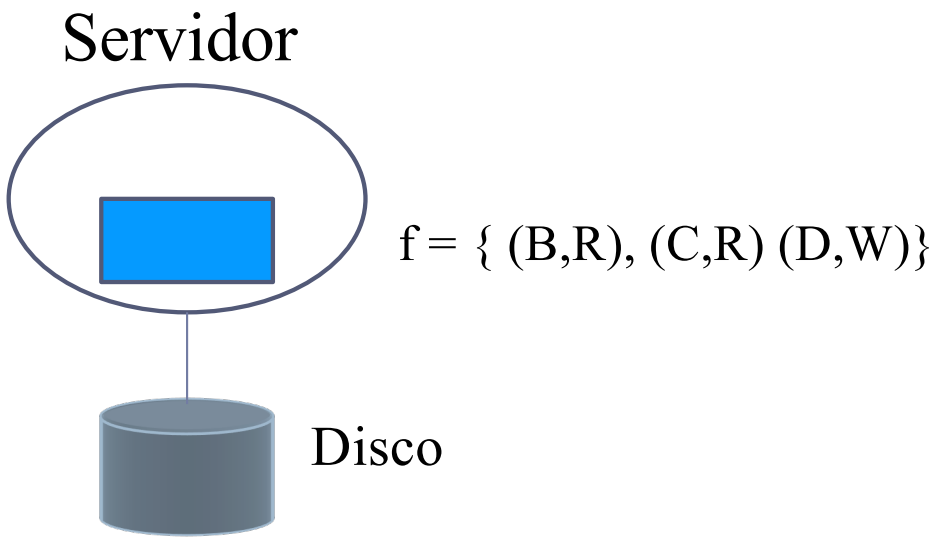


Coherencia de caché en Sprite

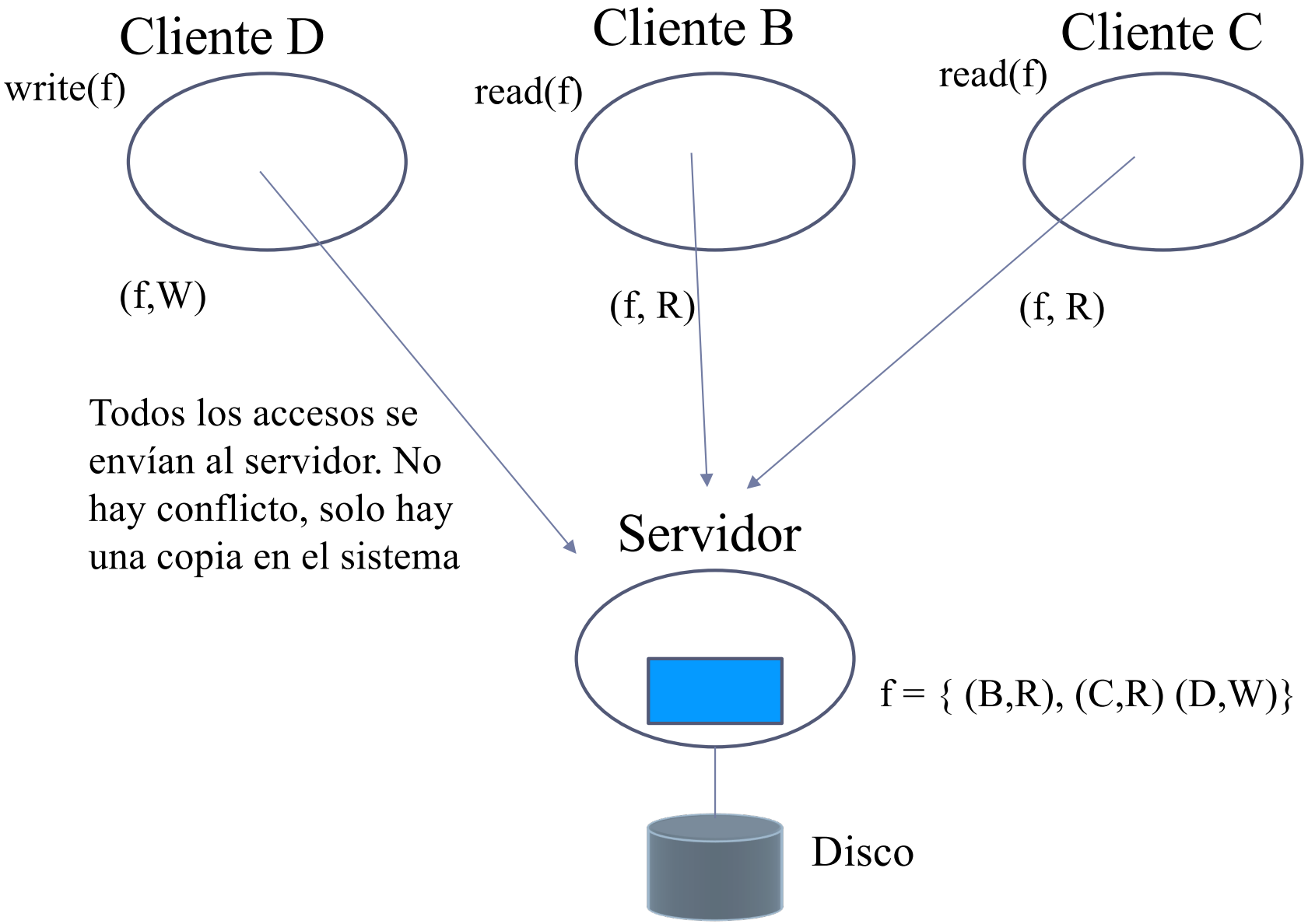


Coherencia de caché en Sprite

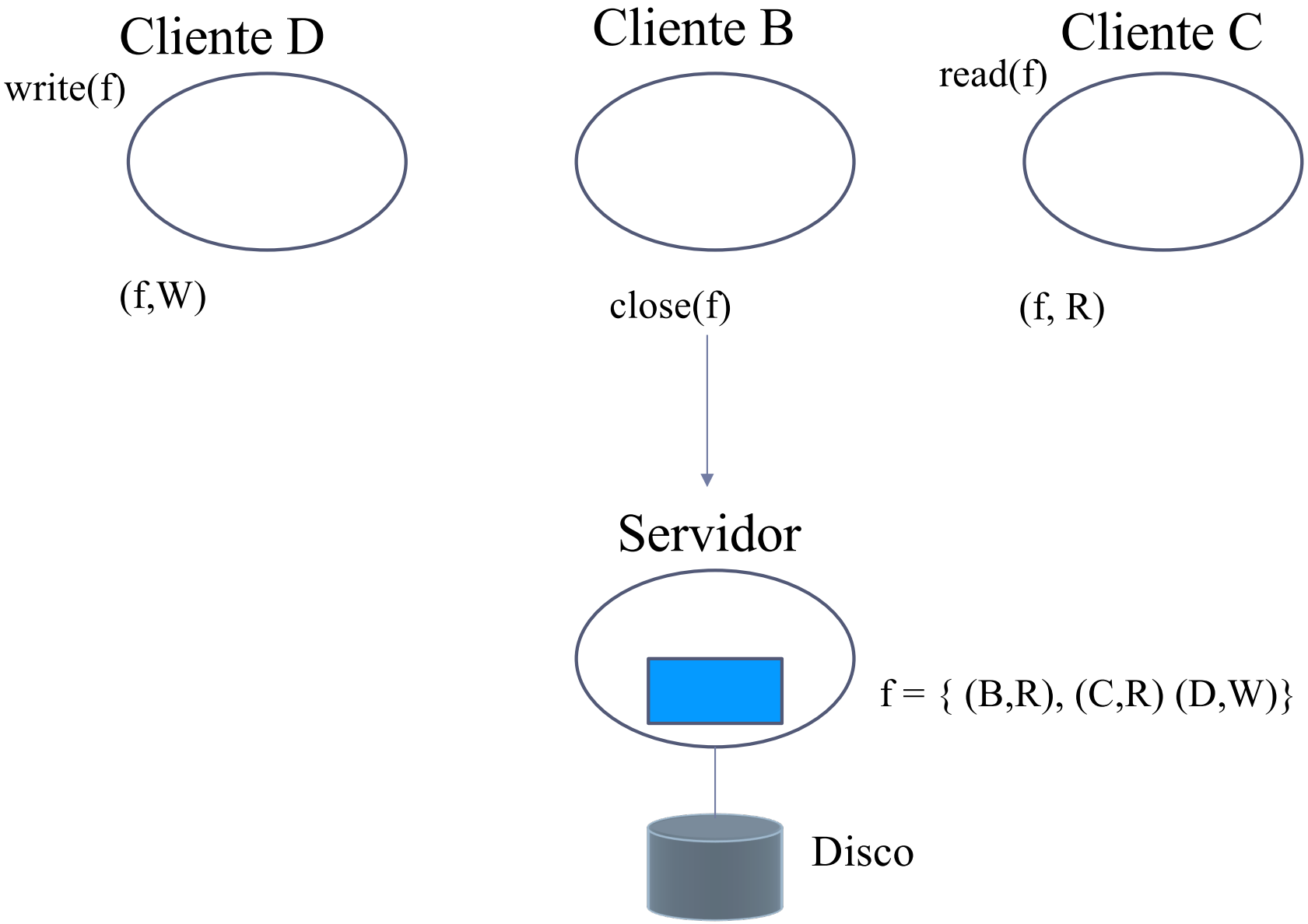




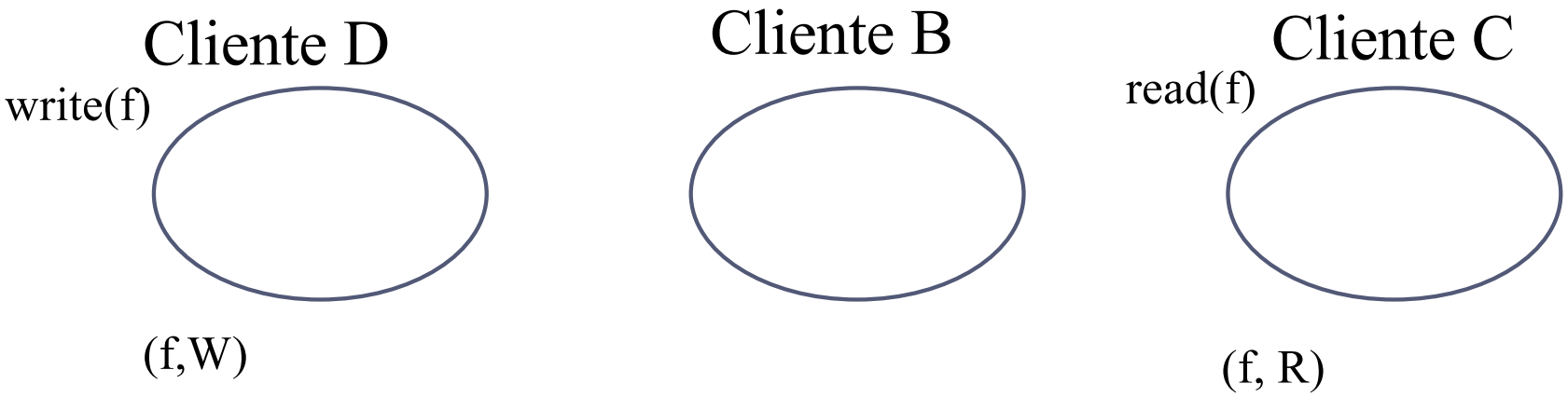
Coherencia de caché en Sprite



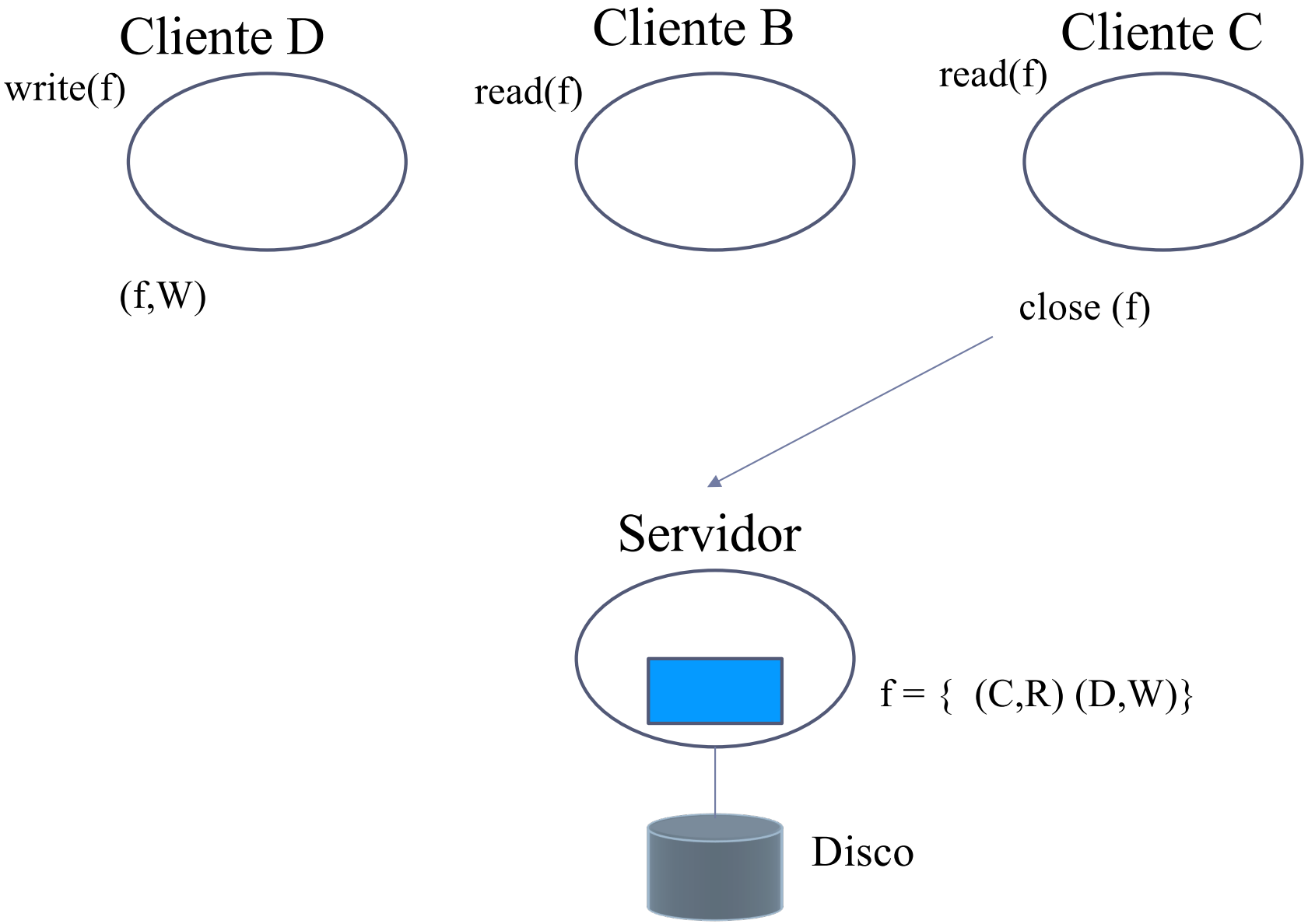
Coherencia de caché en Sprite



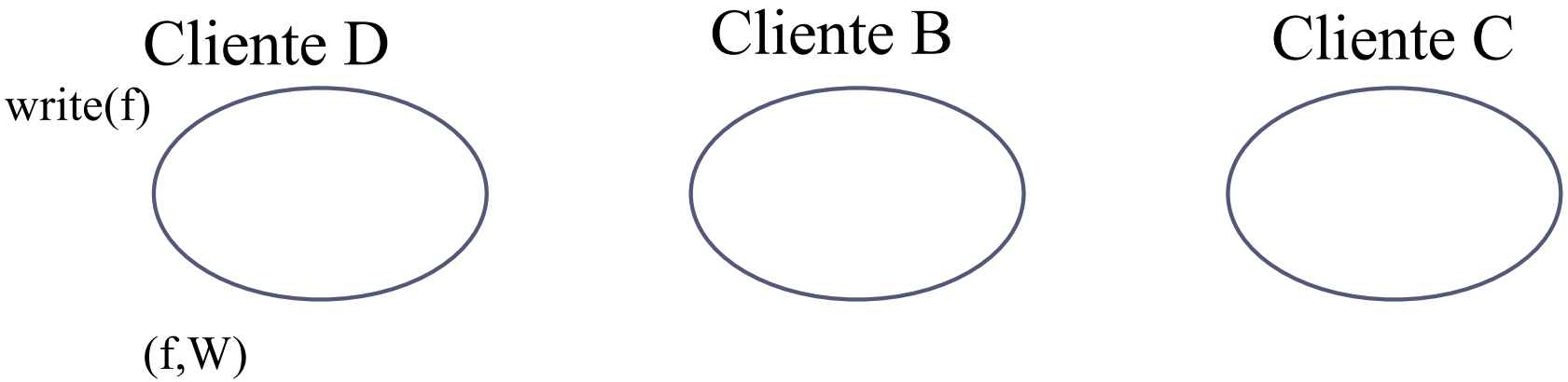
Coherencia de caché en Sprite



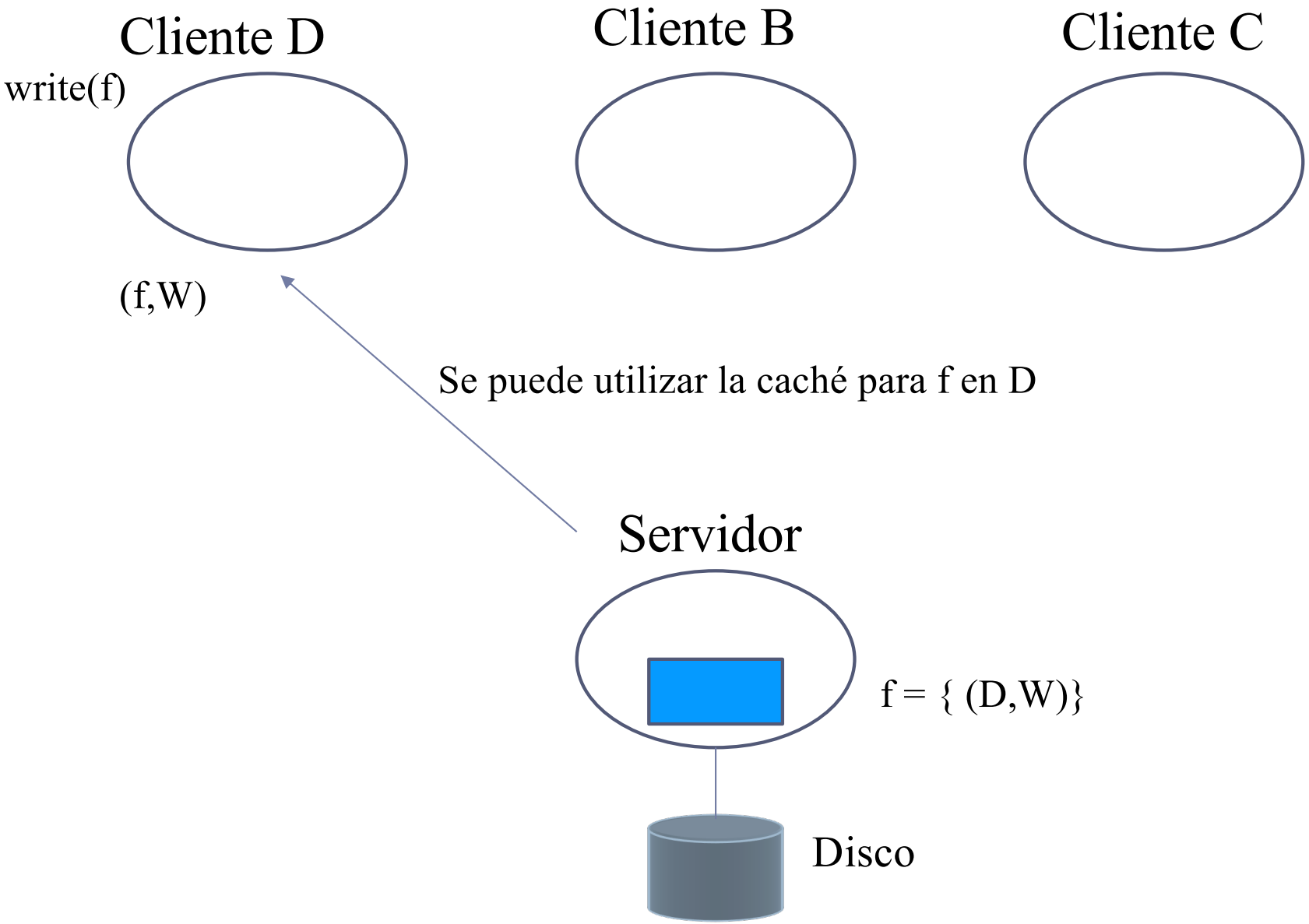
Coherencia de caché en Sprite



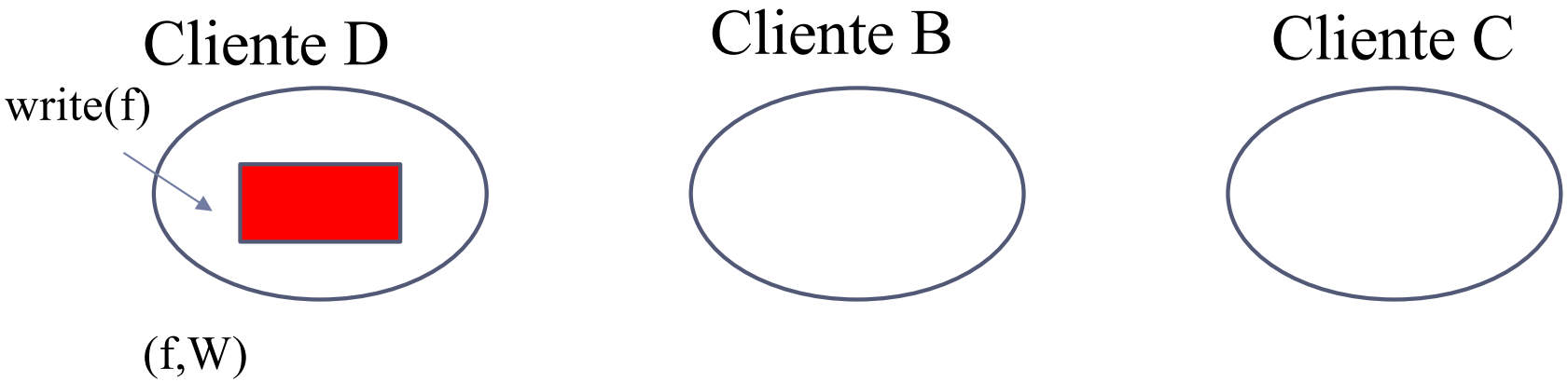
Coherencia de caché en Sprite

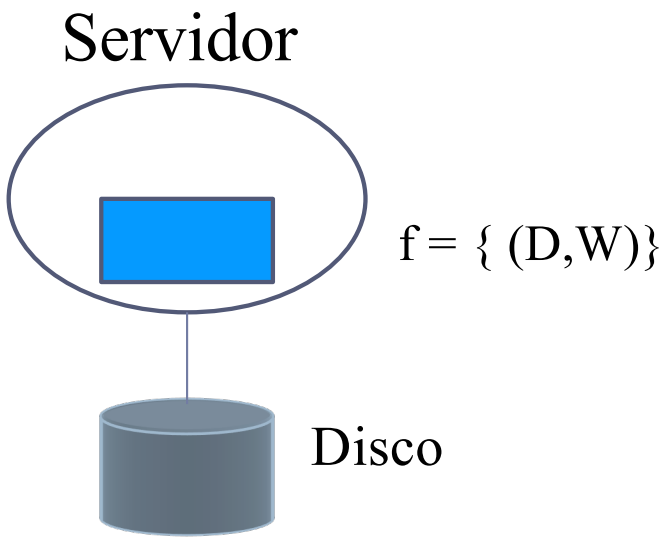


Coherencia de caché en Sprite

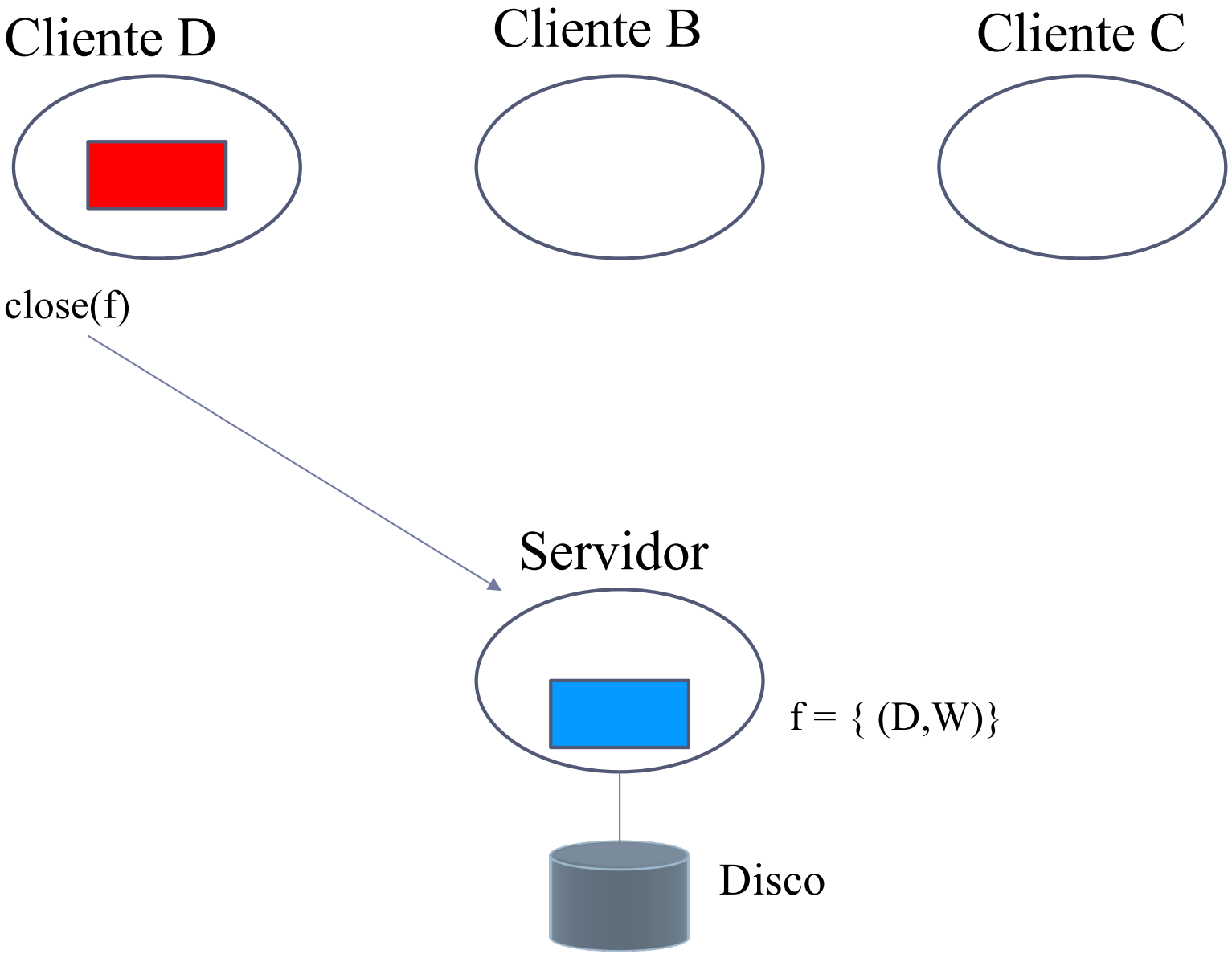


Coherencia de caché en Sprite

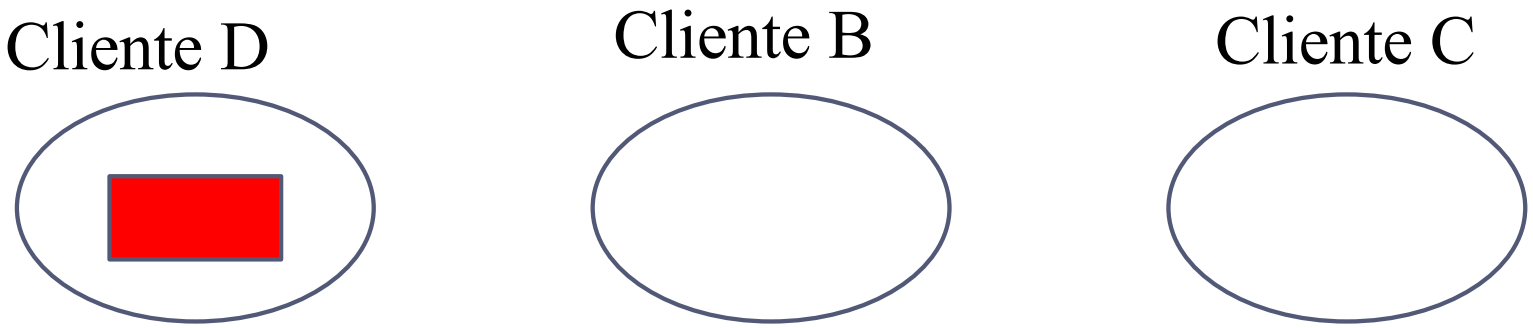




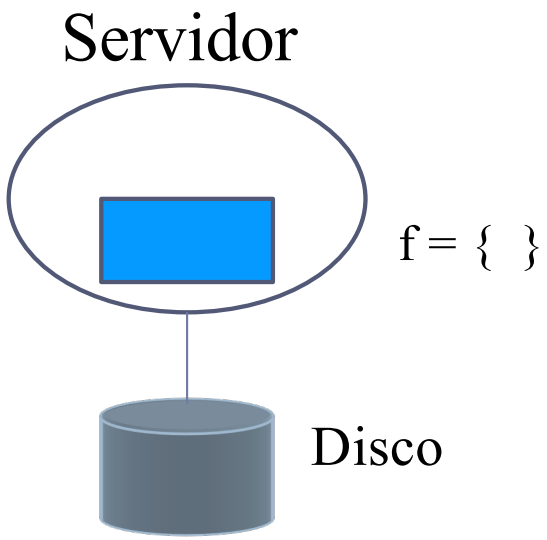
Coherencia de caché en Sprite



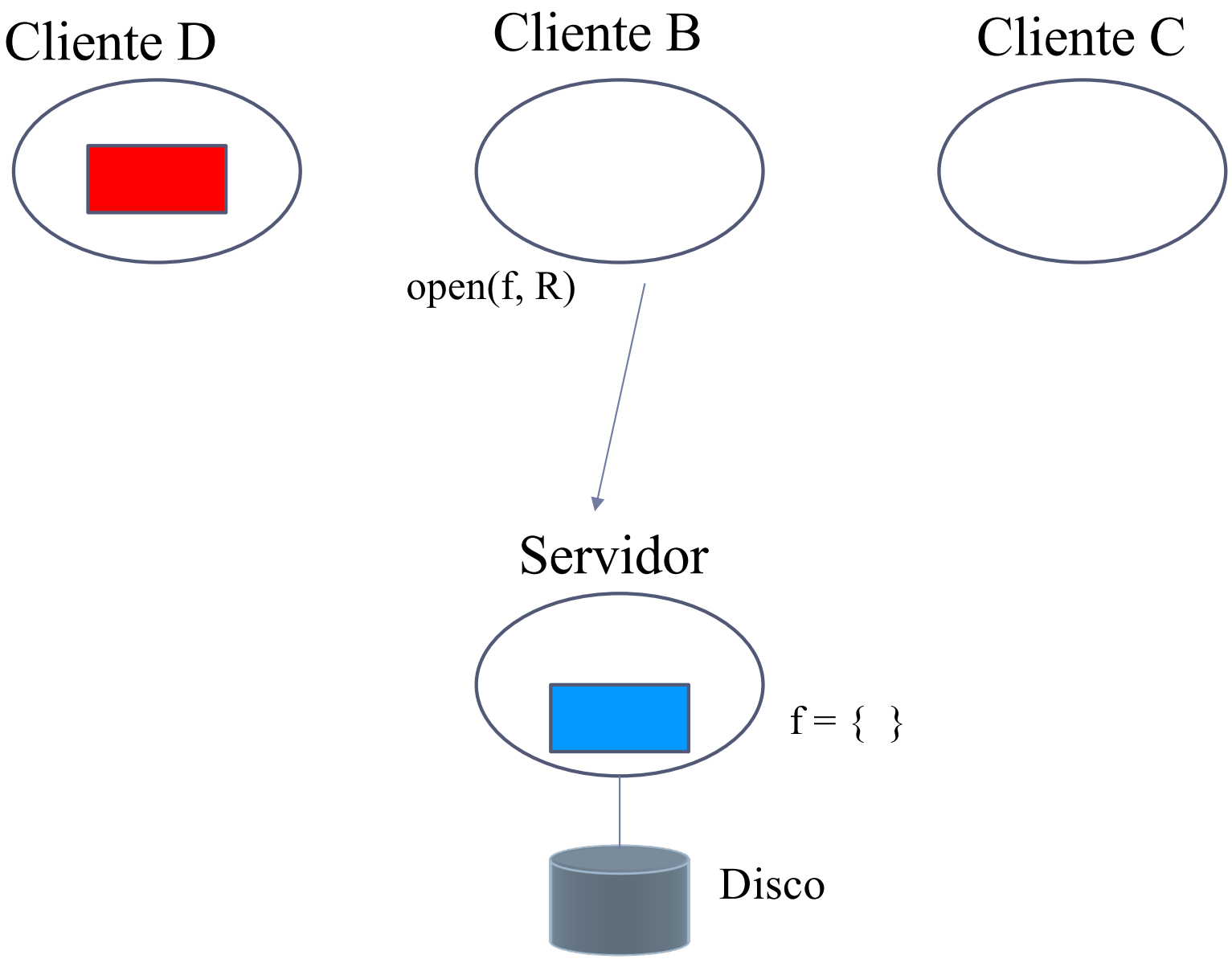
Coherencia de caché en Sprite



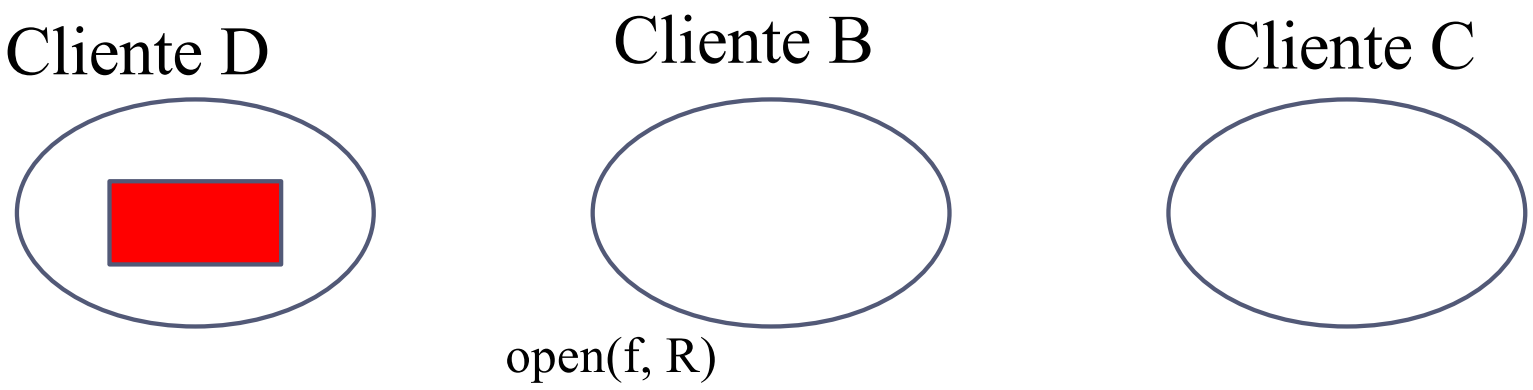
Los bloques se mantienen en
la caché, política de escritura diferida
Estos bloques pueden diferir de los del servidor

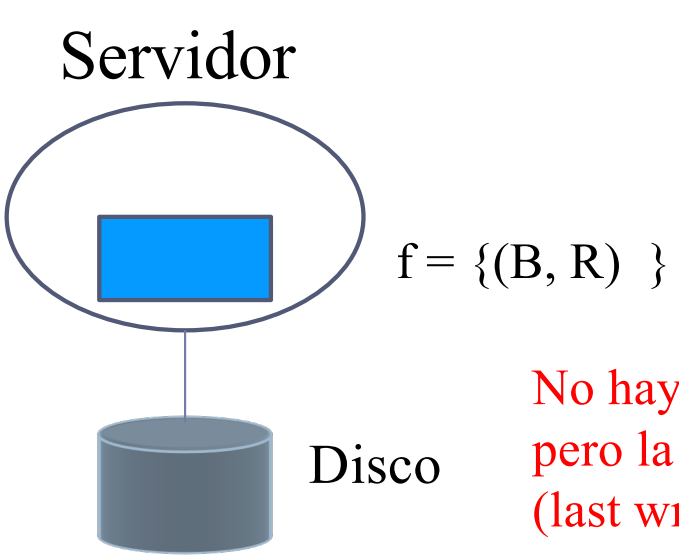


Coherencia de caché en Sprite



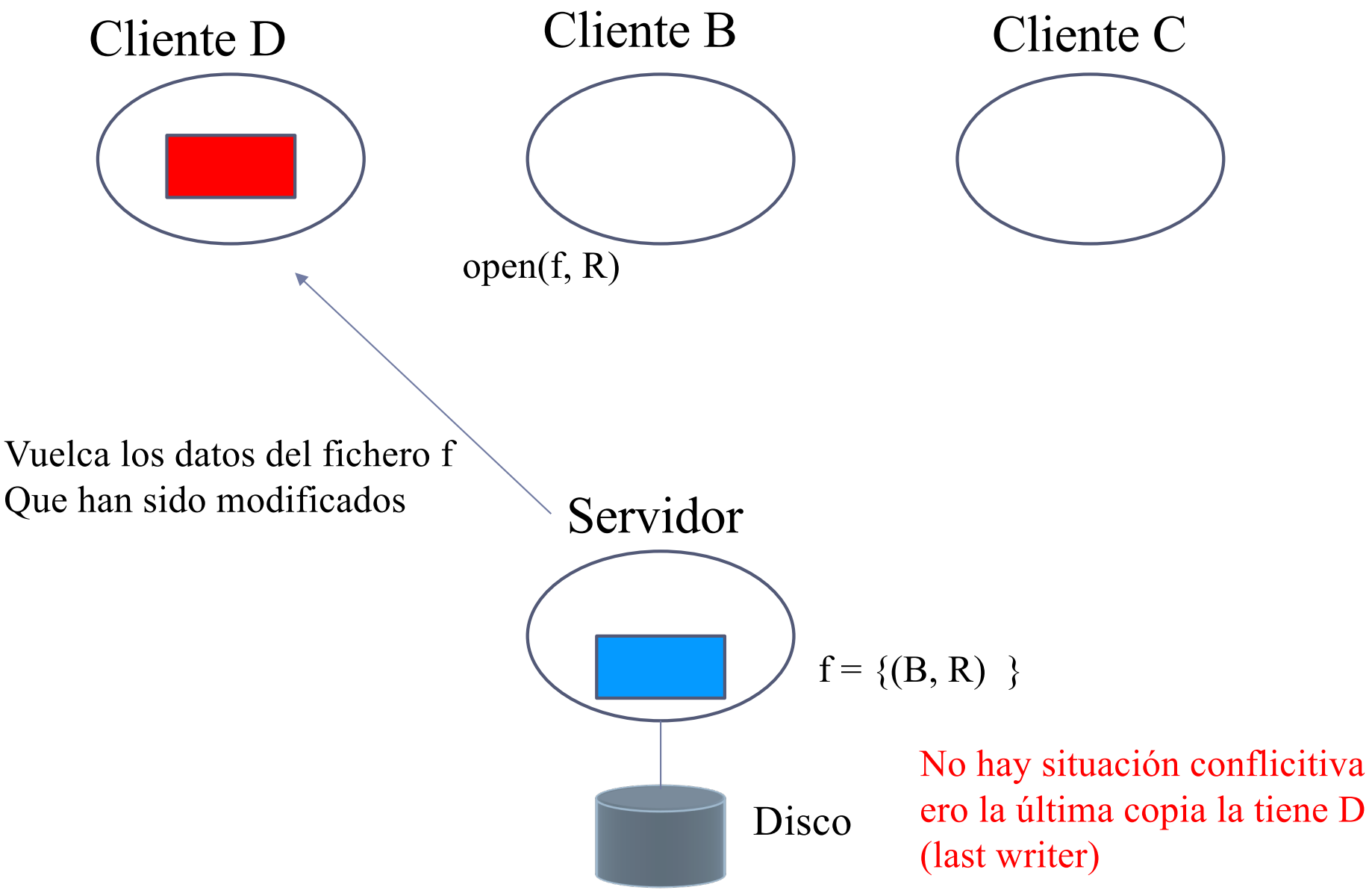
Coherencia de caché en Sprite



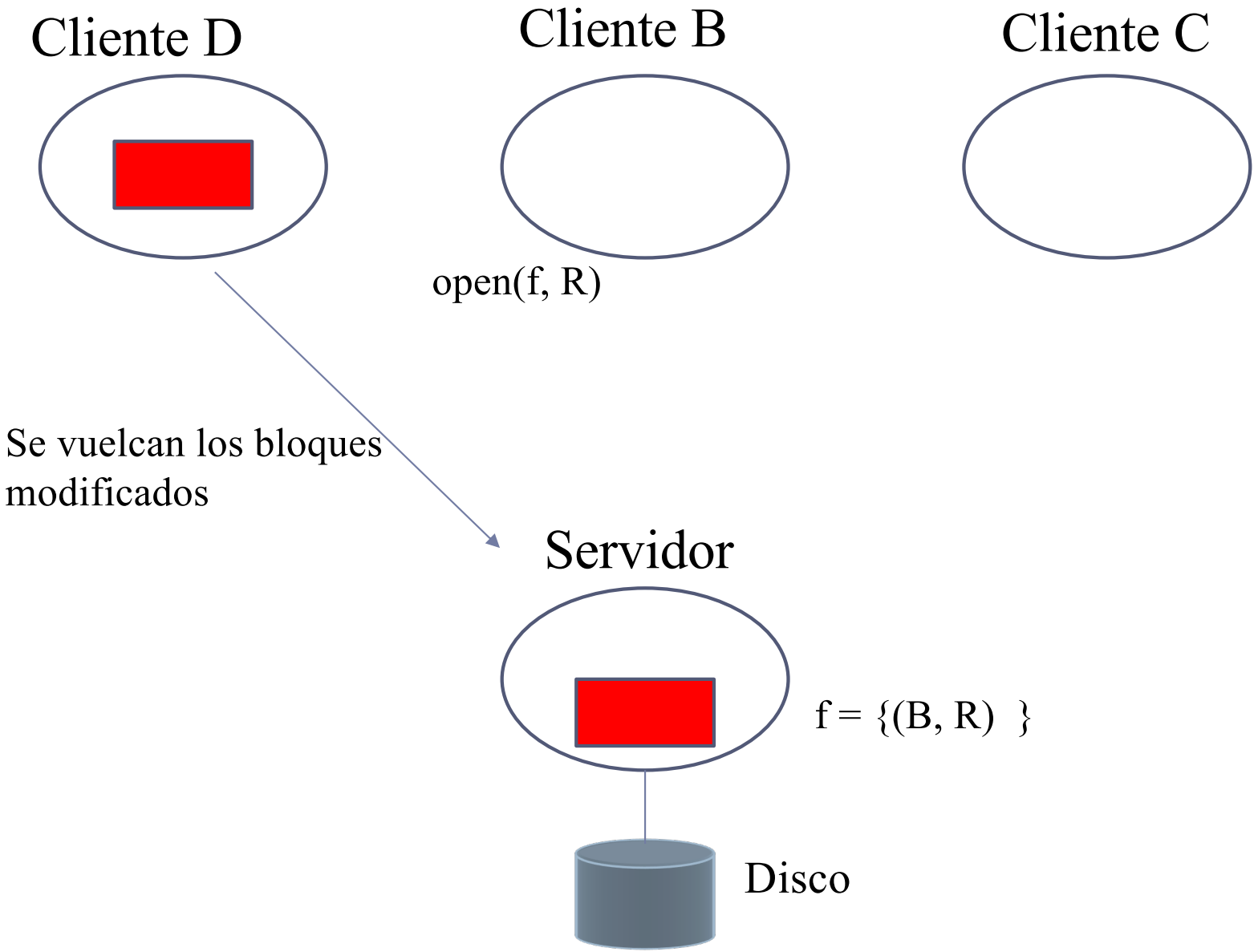


No hay situación conflicitiva pero la última copia la tiene D (last writer)

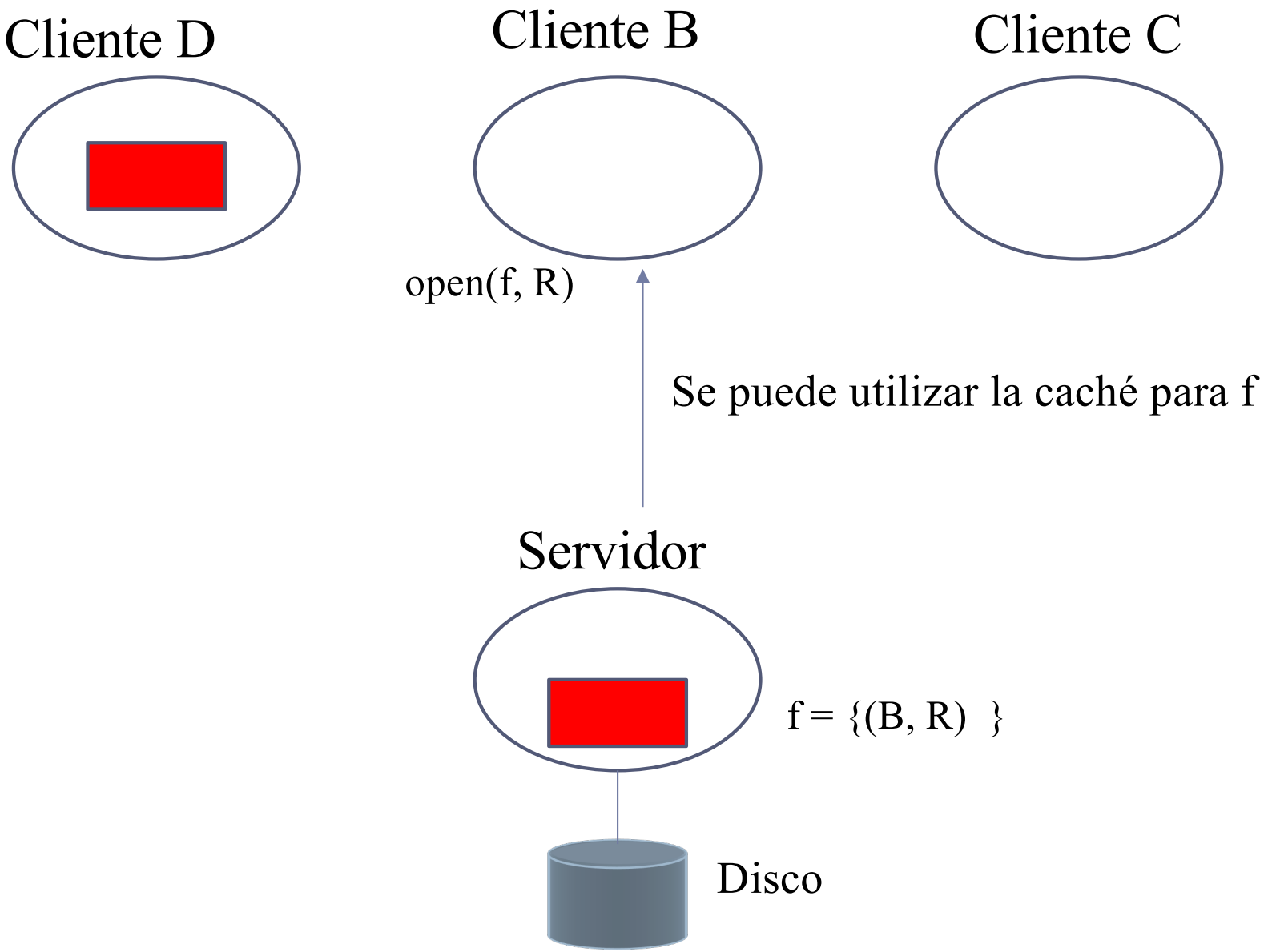
Coherencia de caché en Sprite



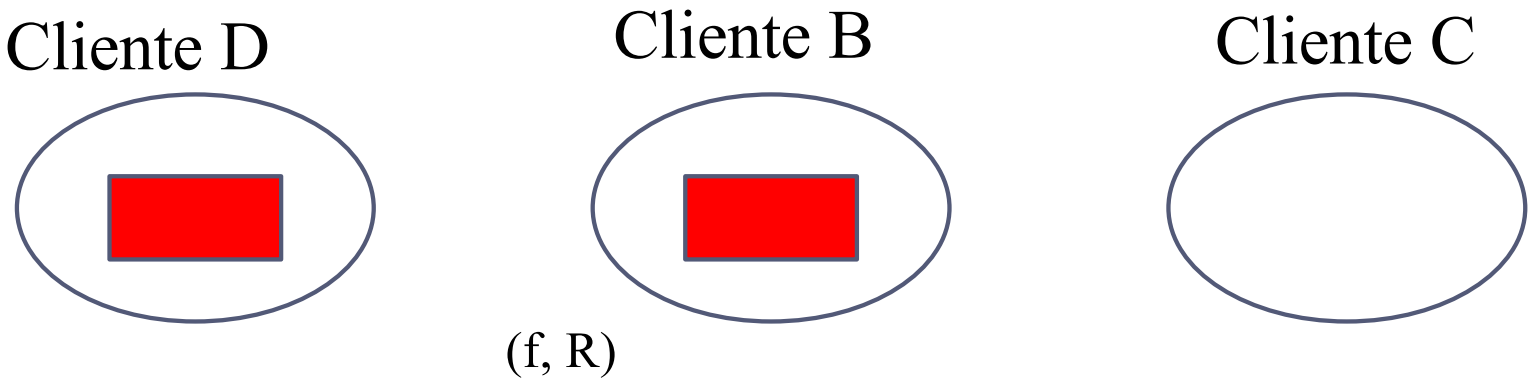
Coherencia de caché en Sprite

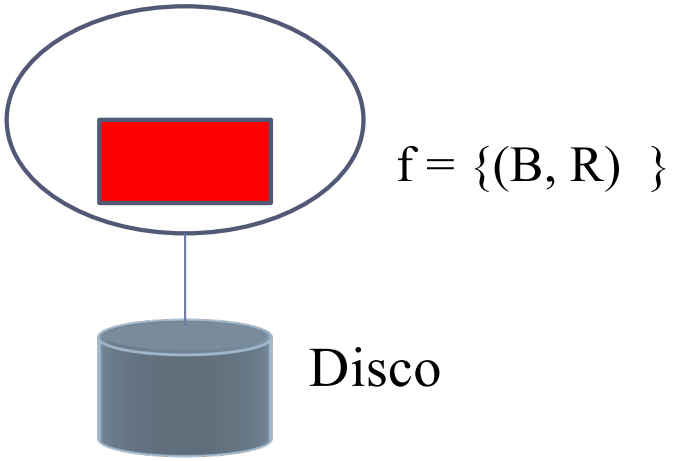


Coherencia de caché en Sprite

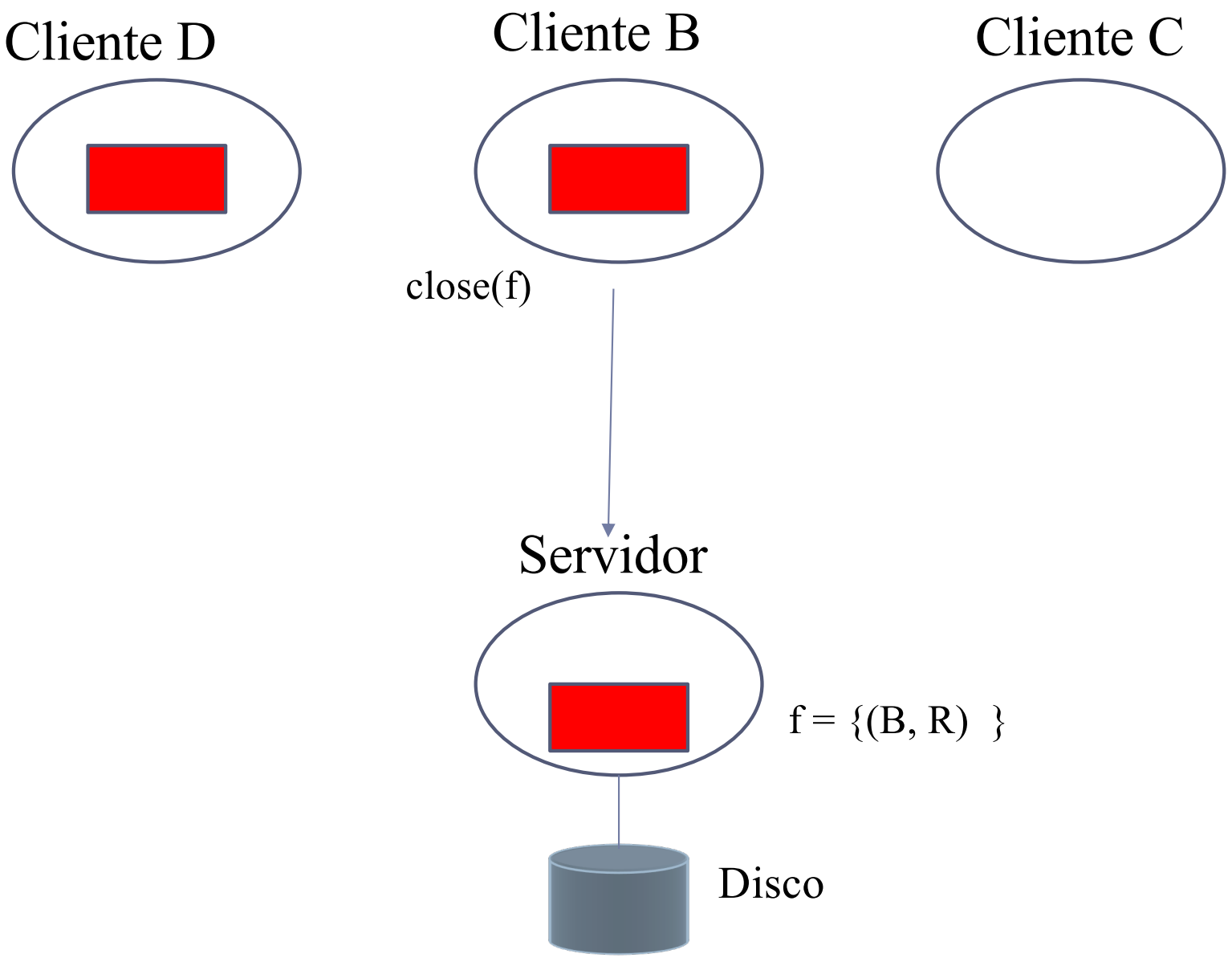


Coherencia de caché en Sprite

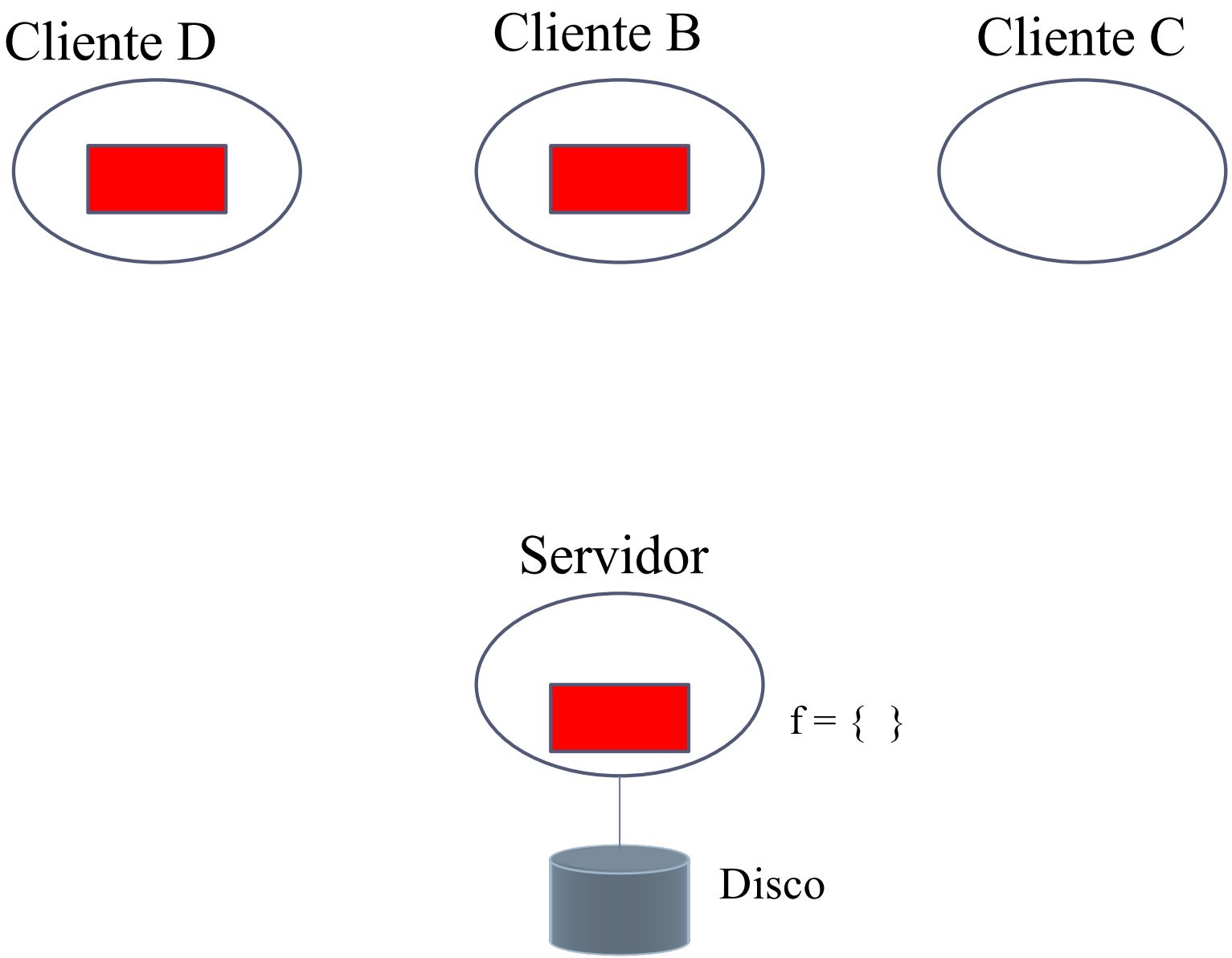




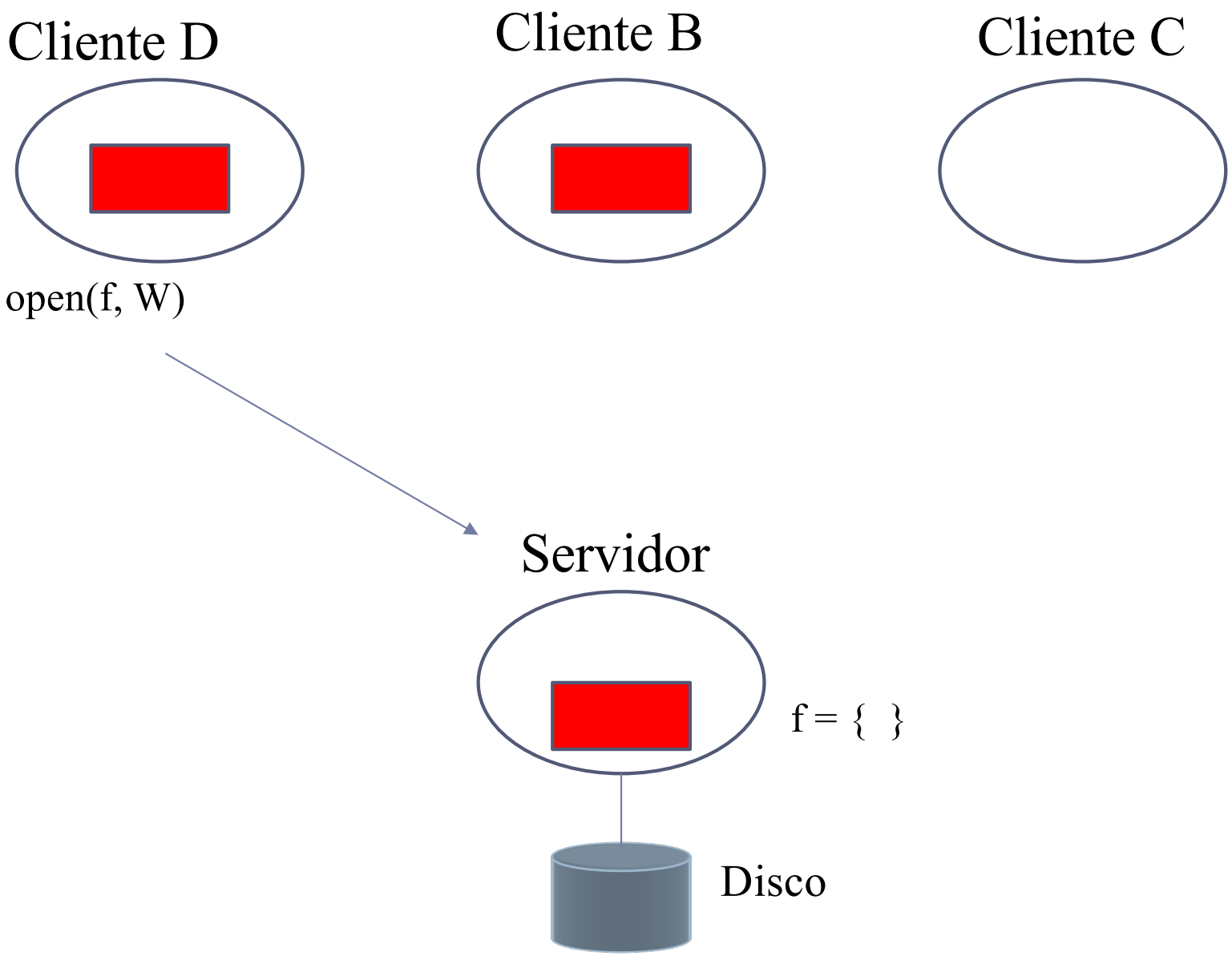
Coherencia de caché en Sprite



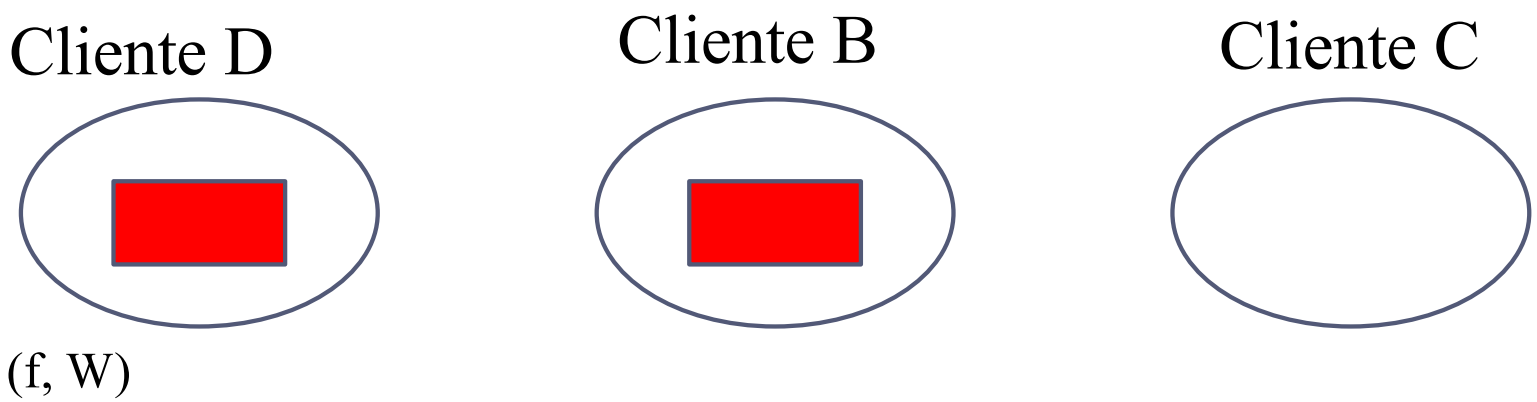
Coherencia de caché en Sprite

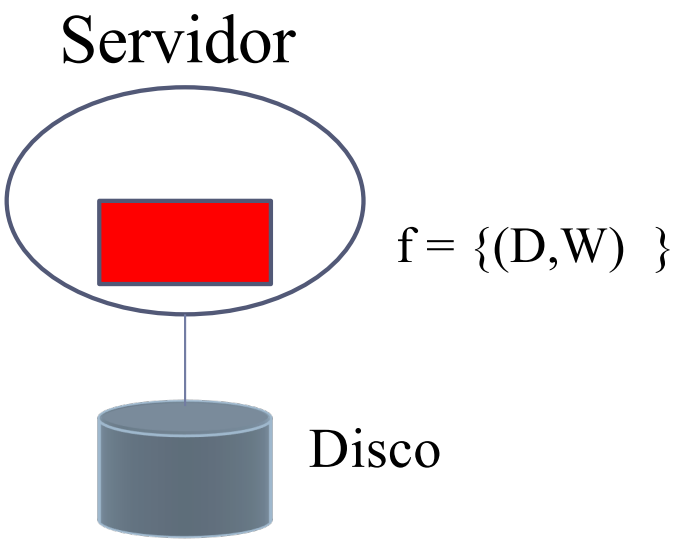


Coherencia de caché en Sprite

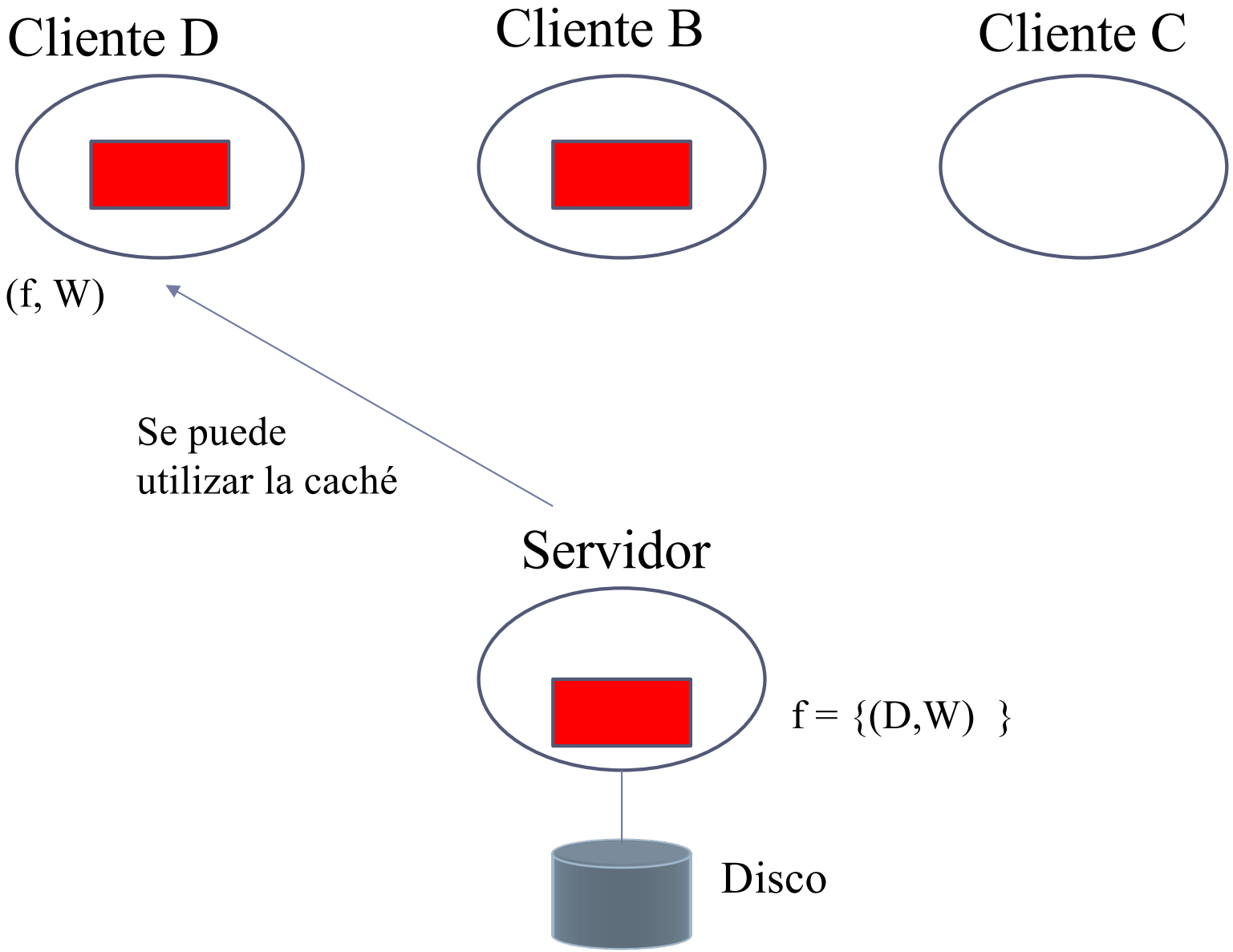


Coherencia de caché en Sprite

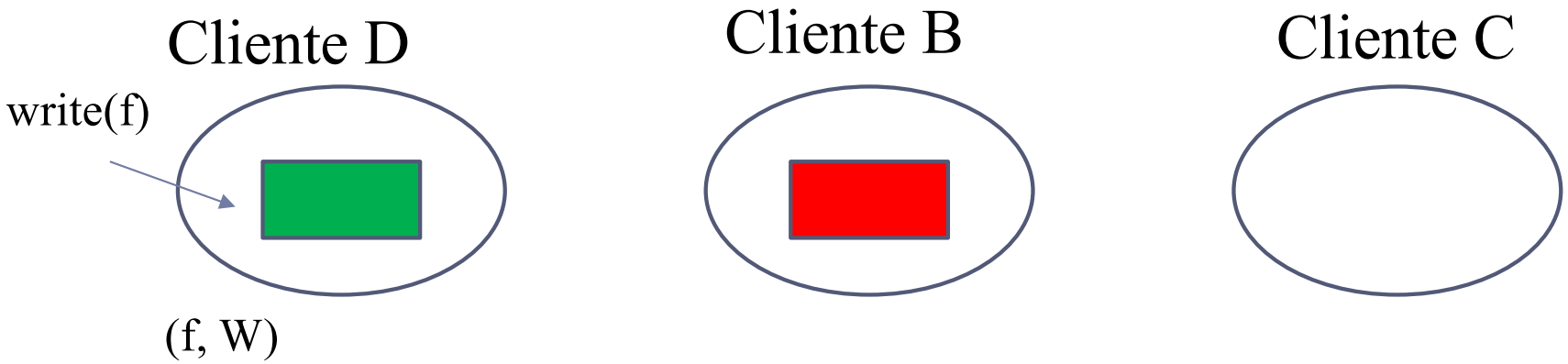


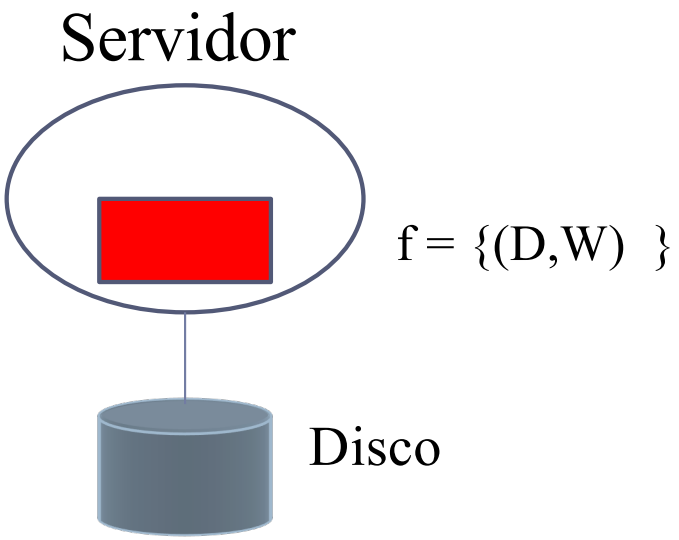


Coherencia de caché en Sprite

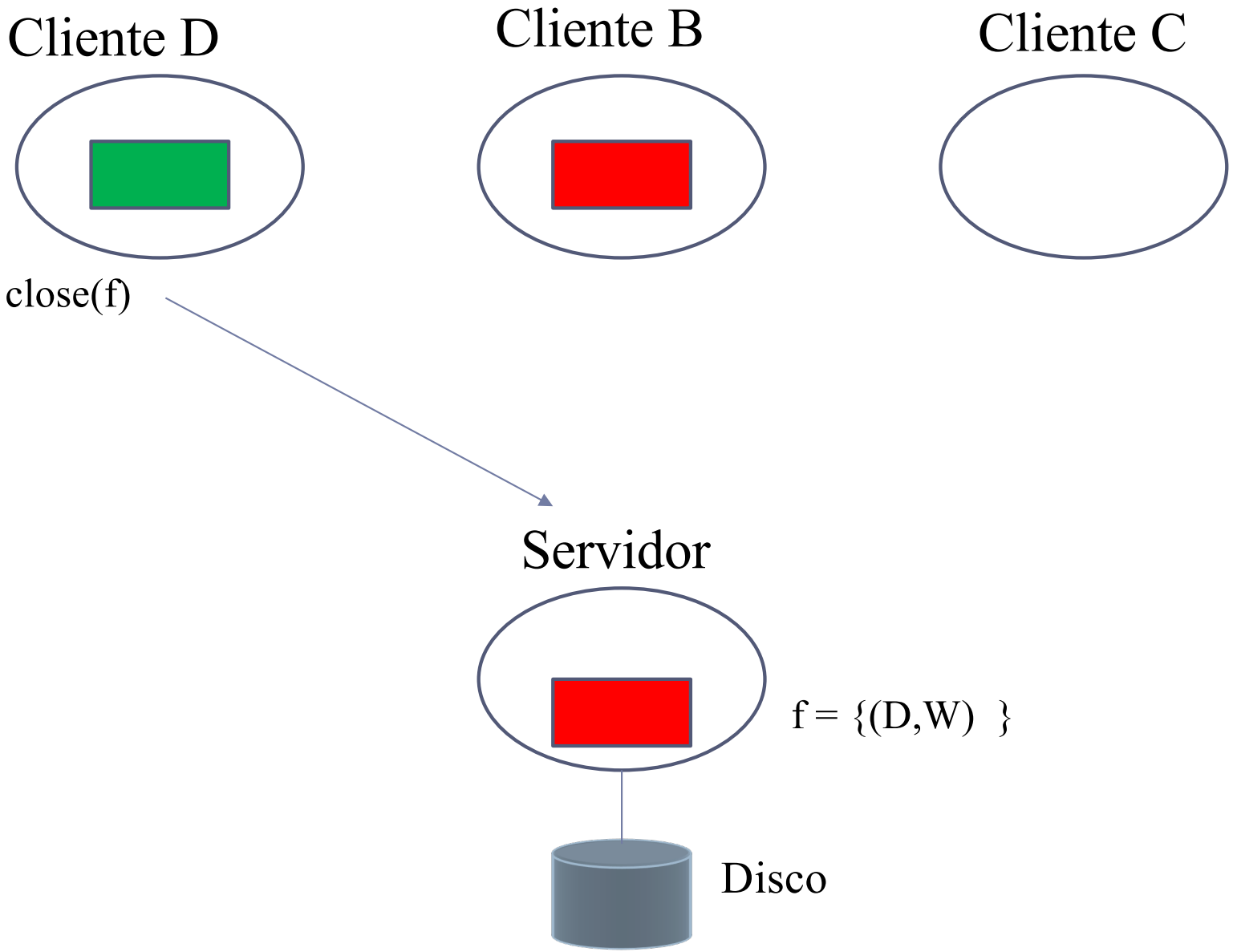


Coherencia de caché en Sprite

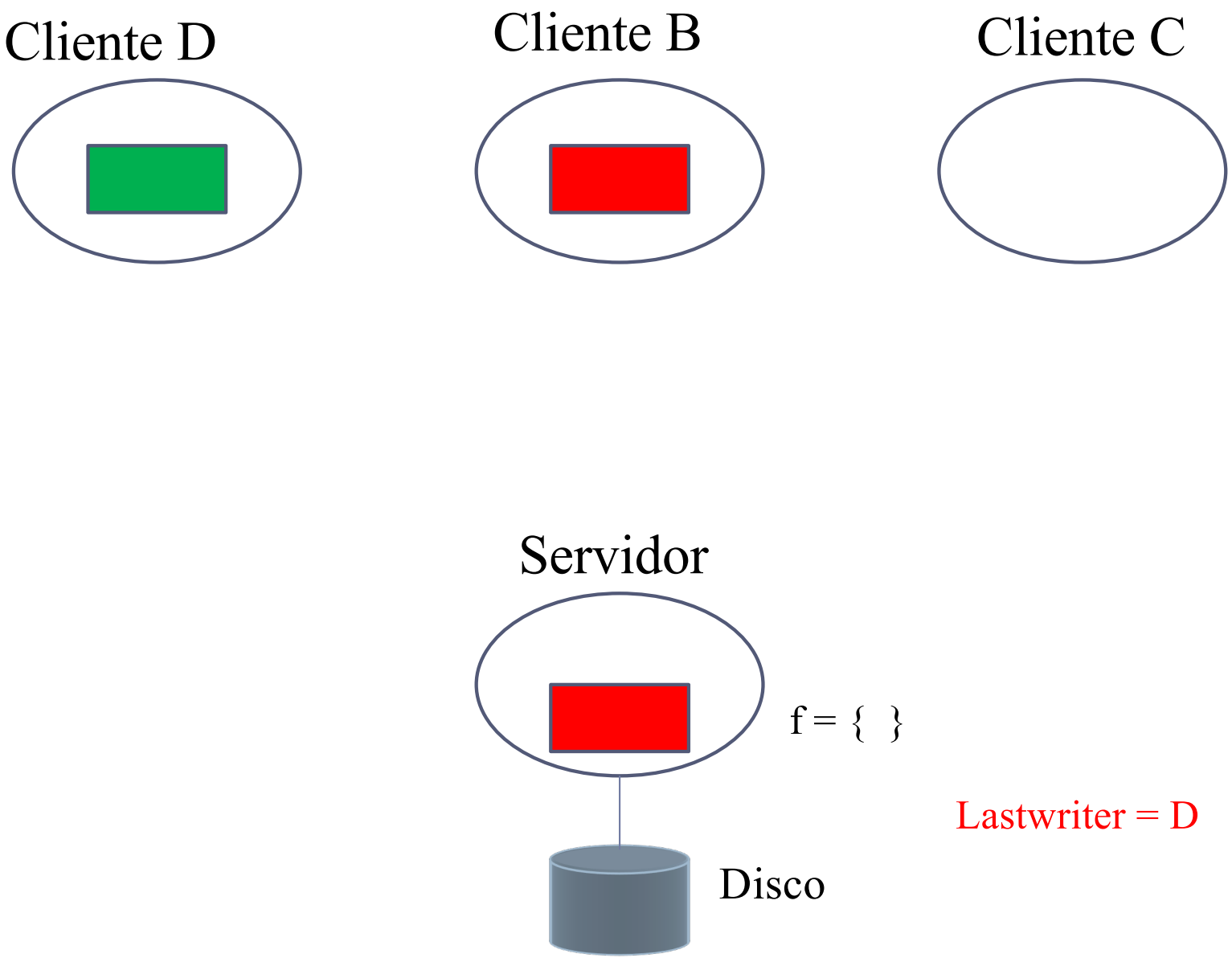




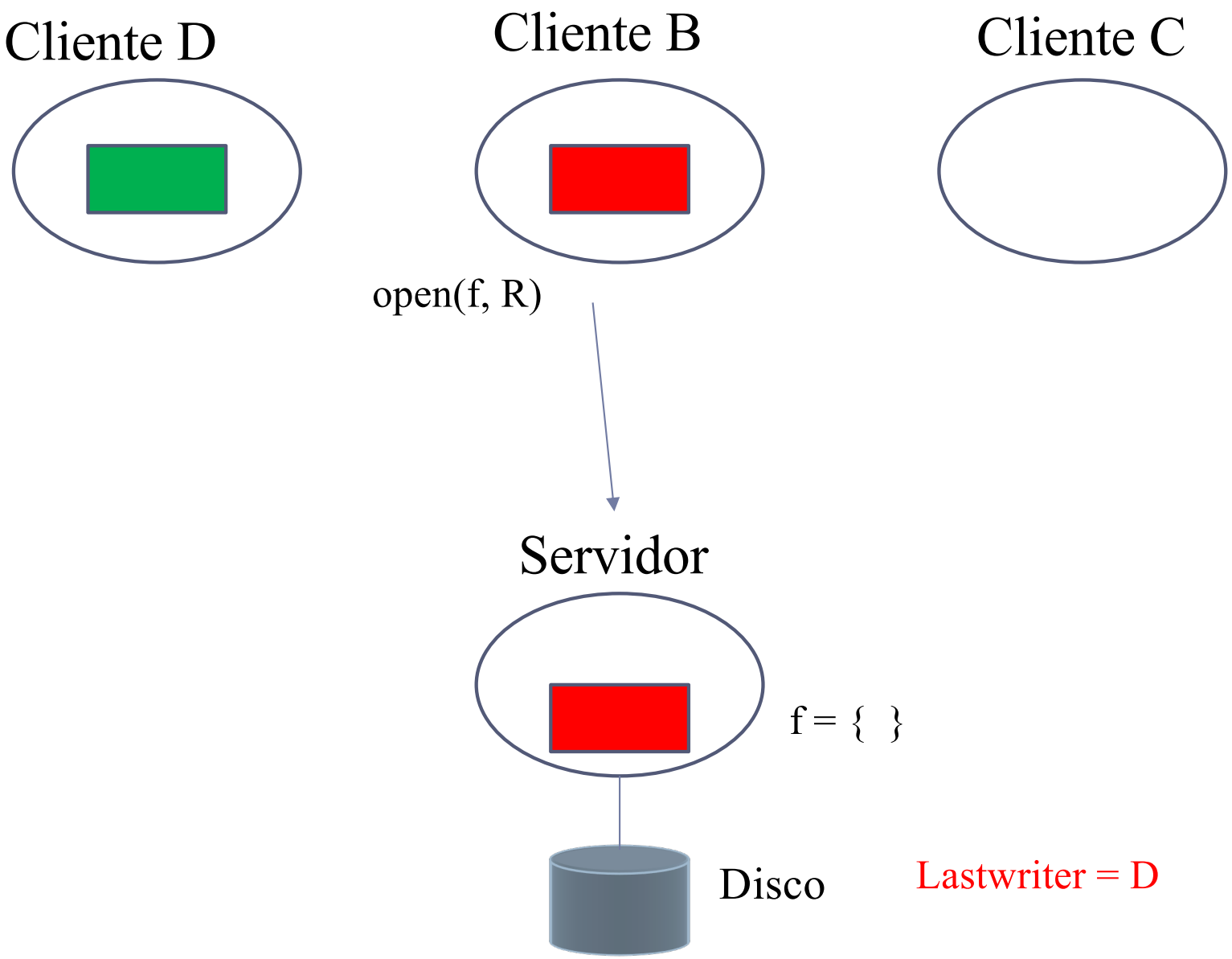
Coherencia de caché en Sprite



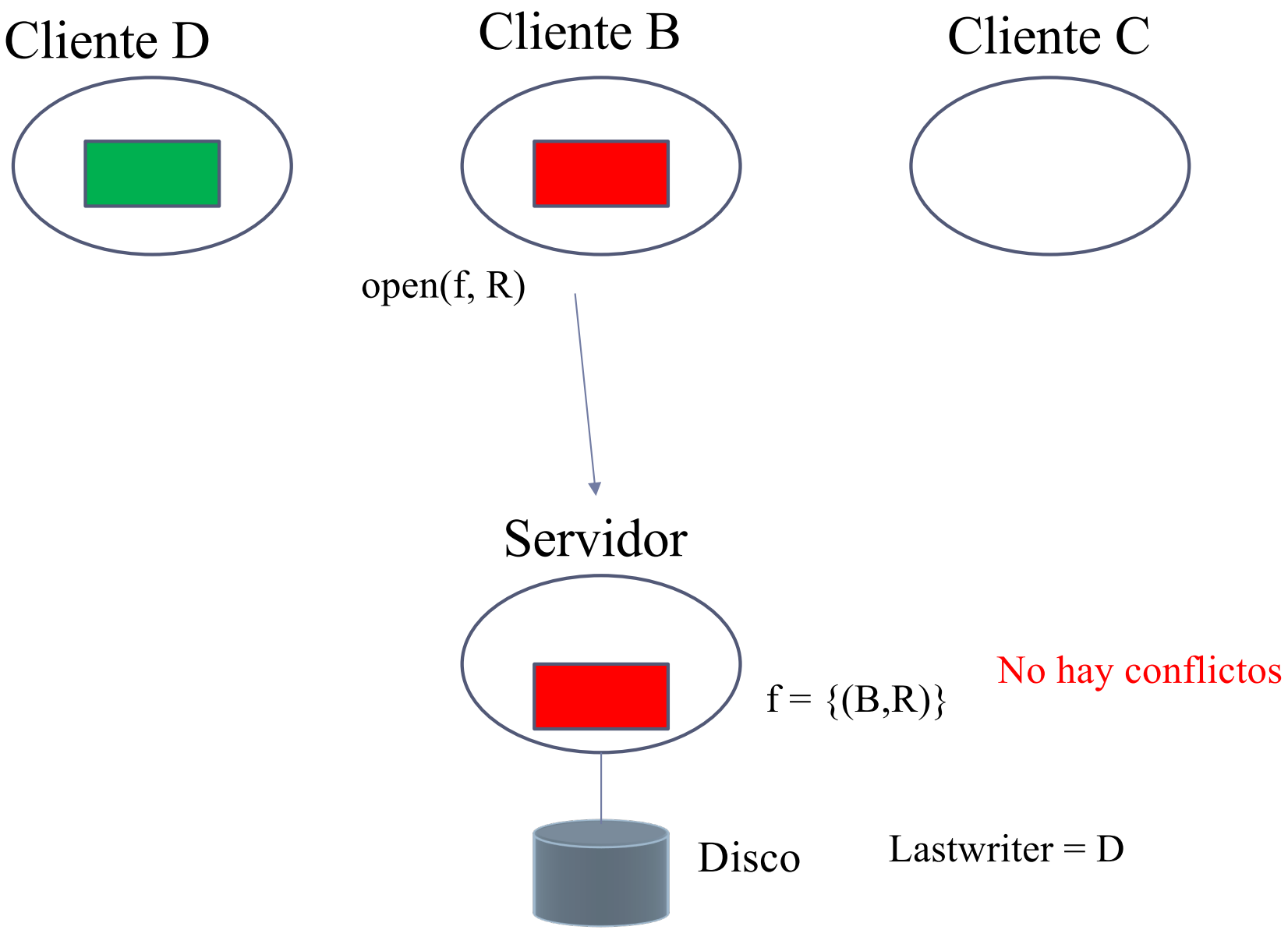
Coherencia de caché en Sprite



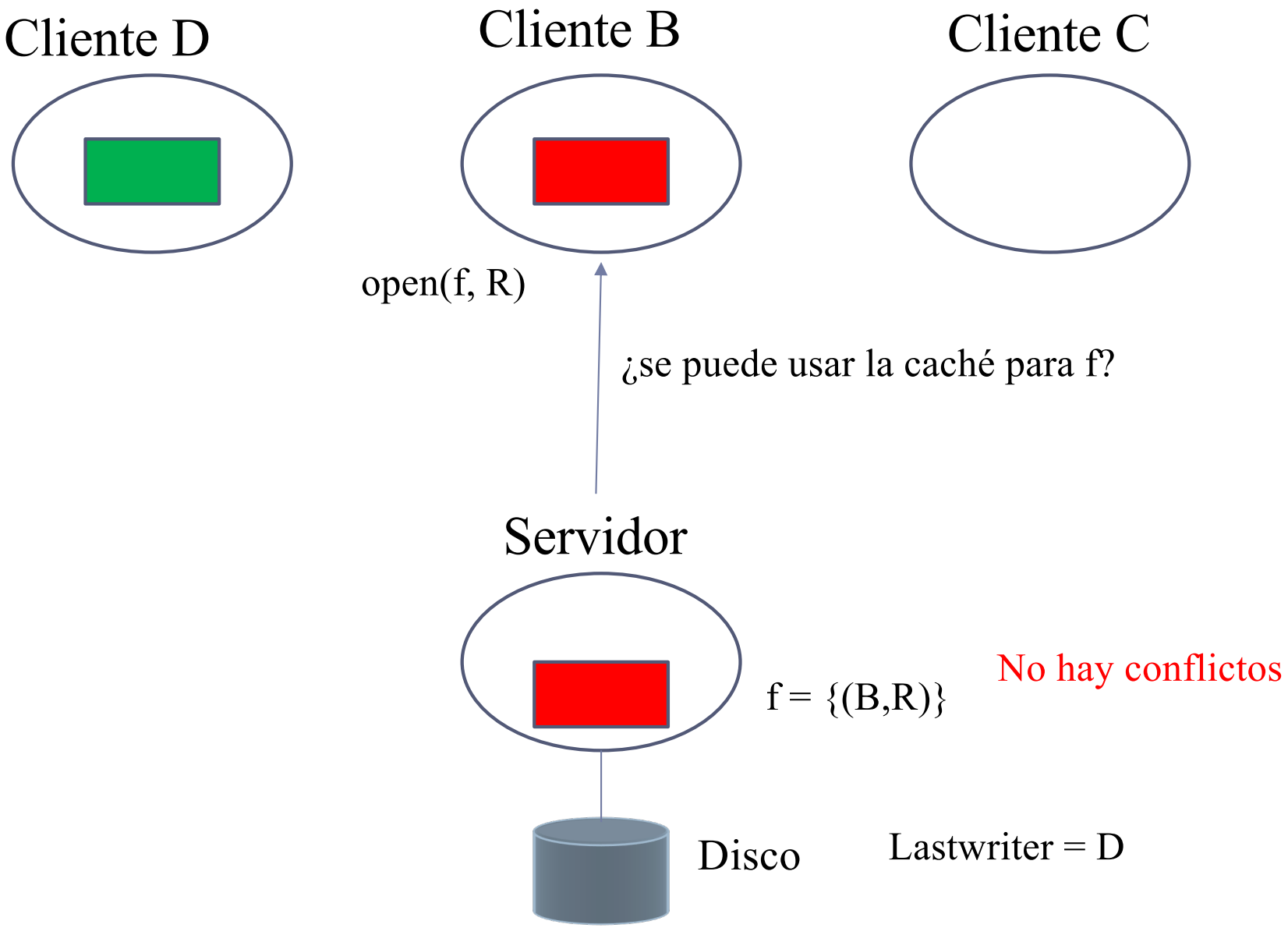
Coherencia de caché en Sprite



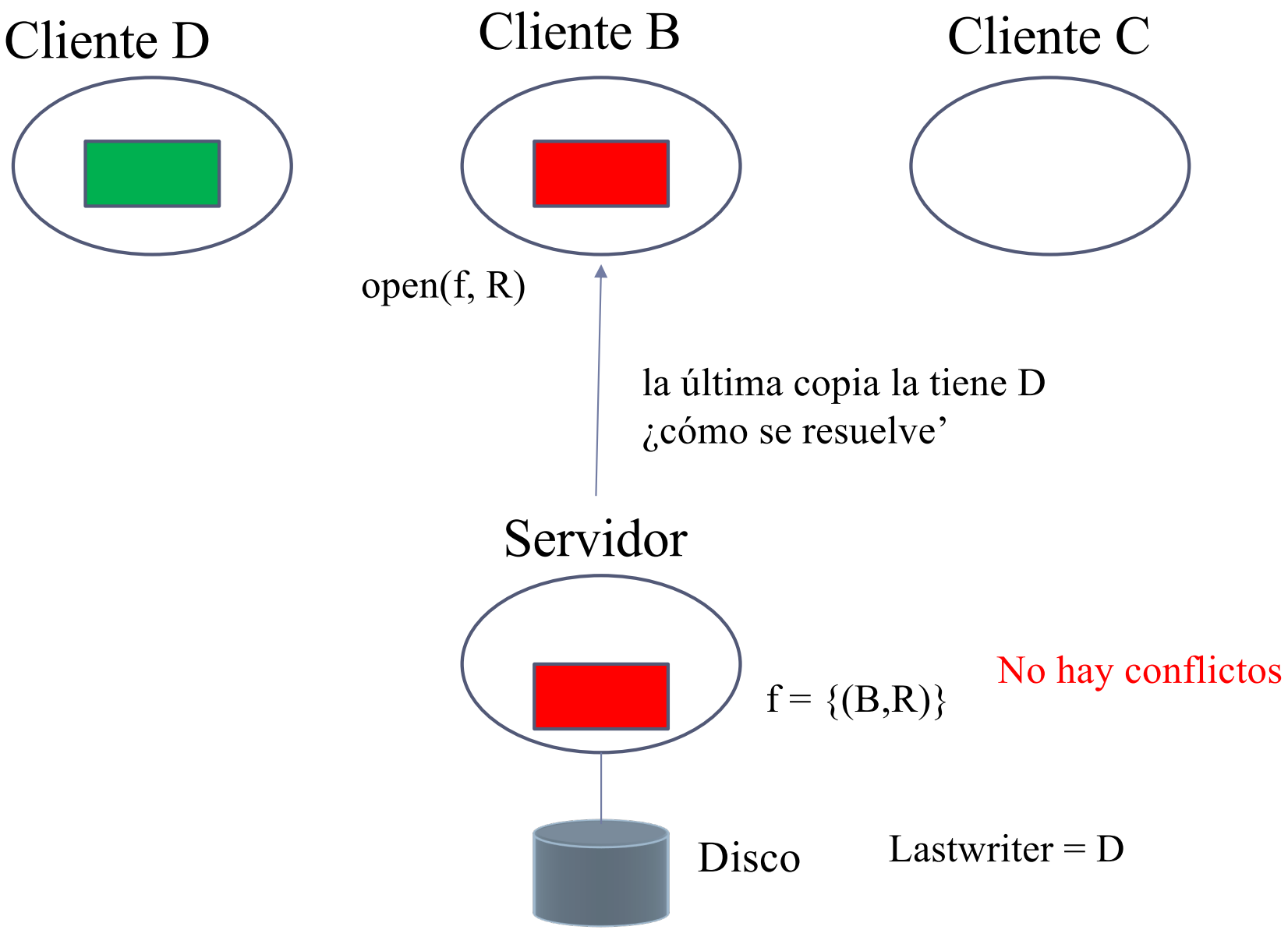
Coherencia de caché en Sprite



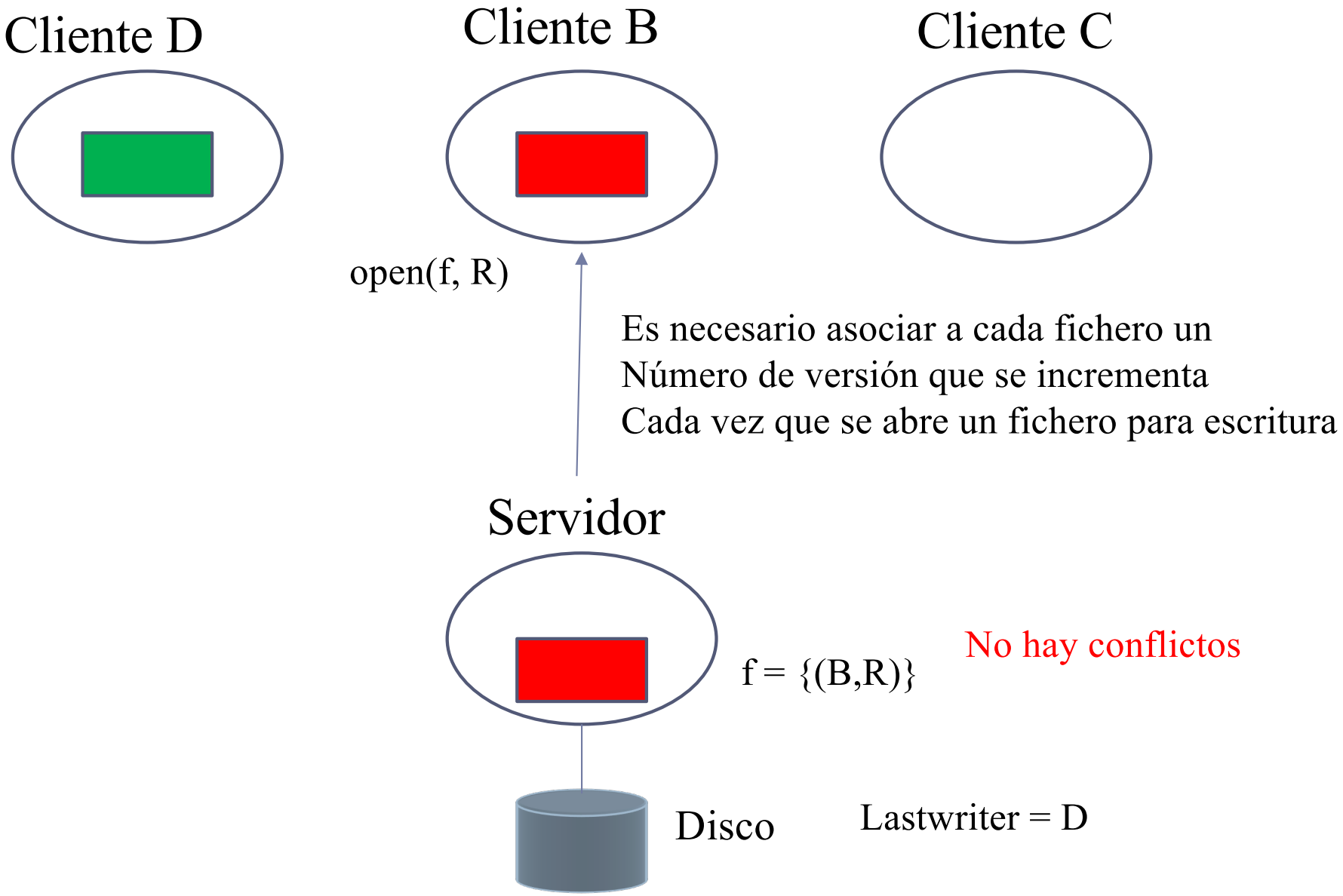
Coherencia de caché en Sprite



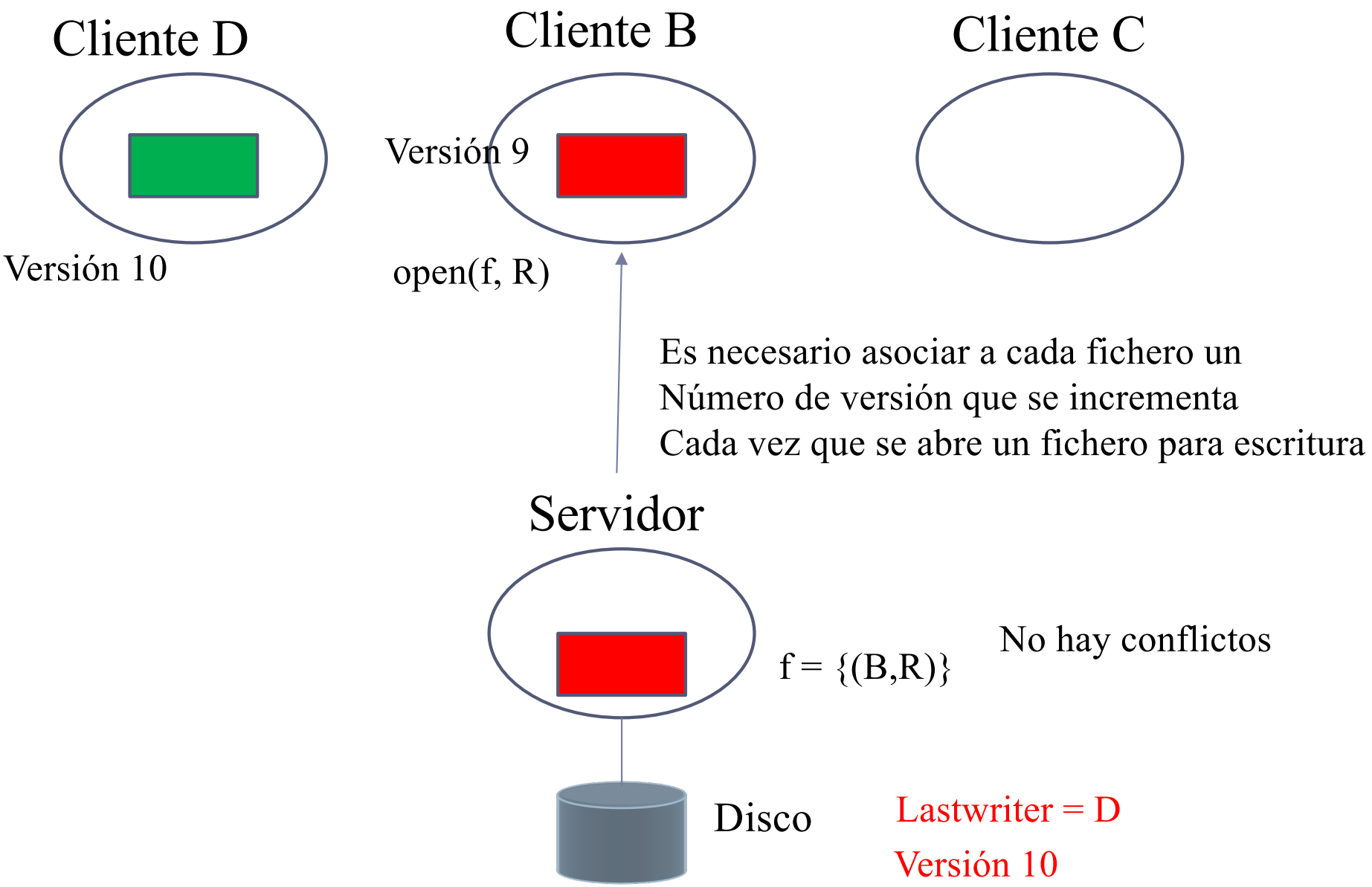
Coherencia de caché en Sprite



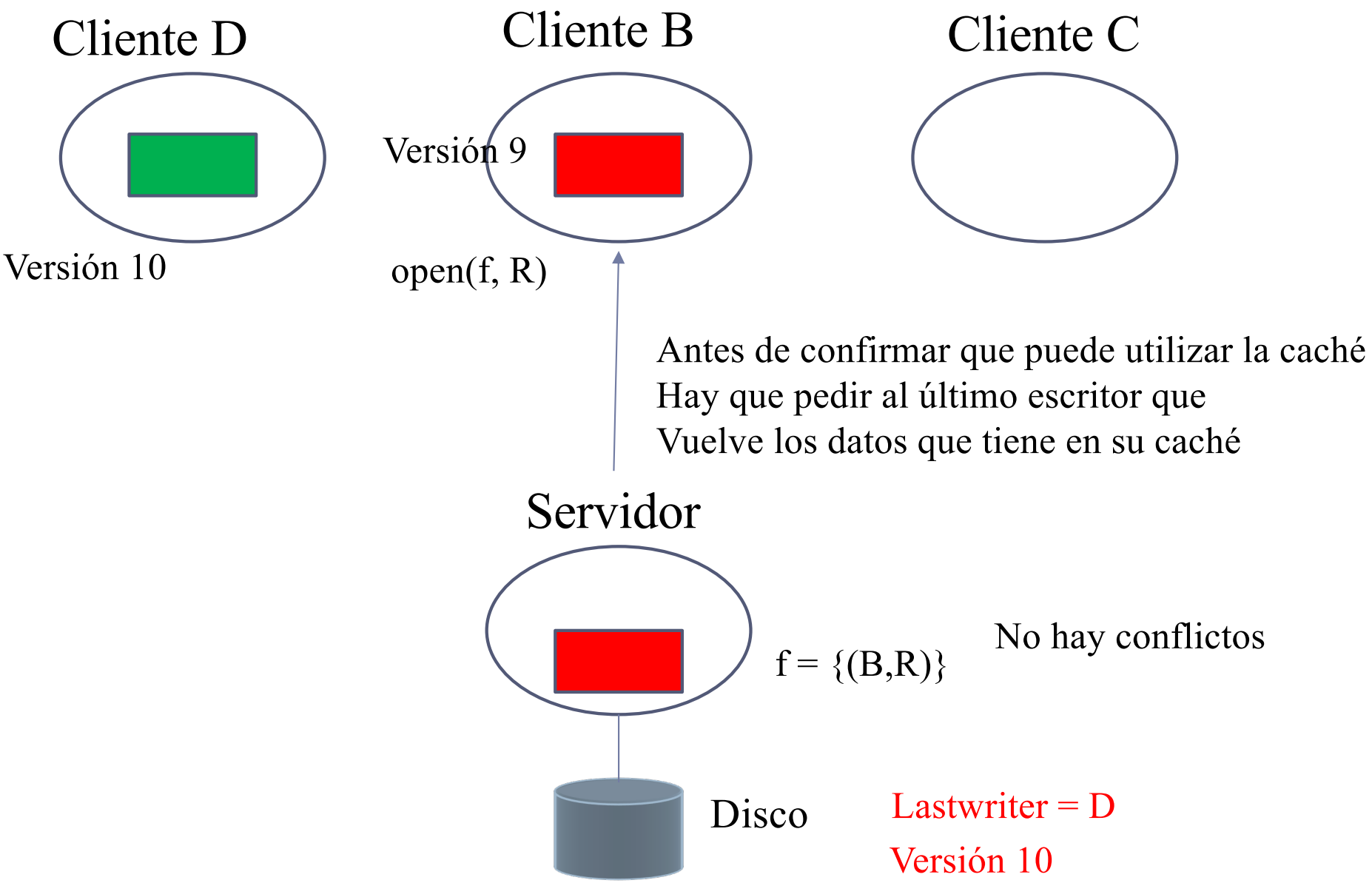
Coherencia de caché en Sprite



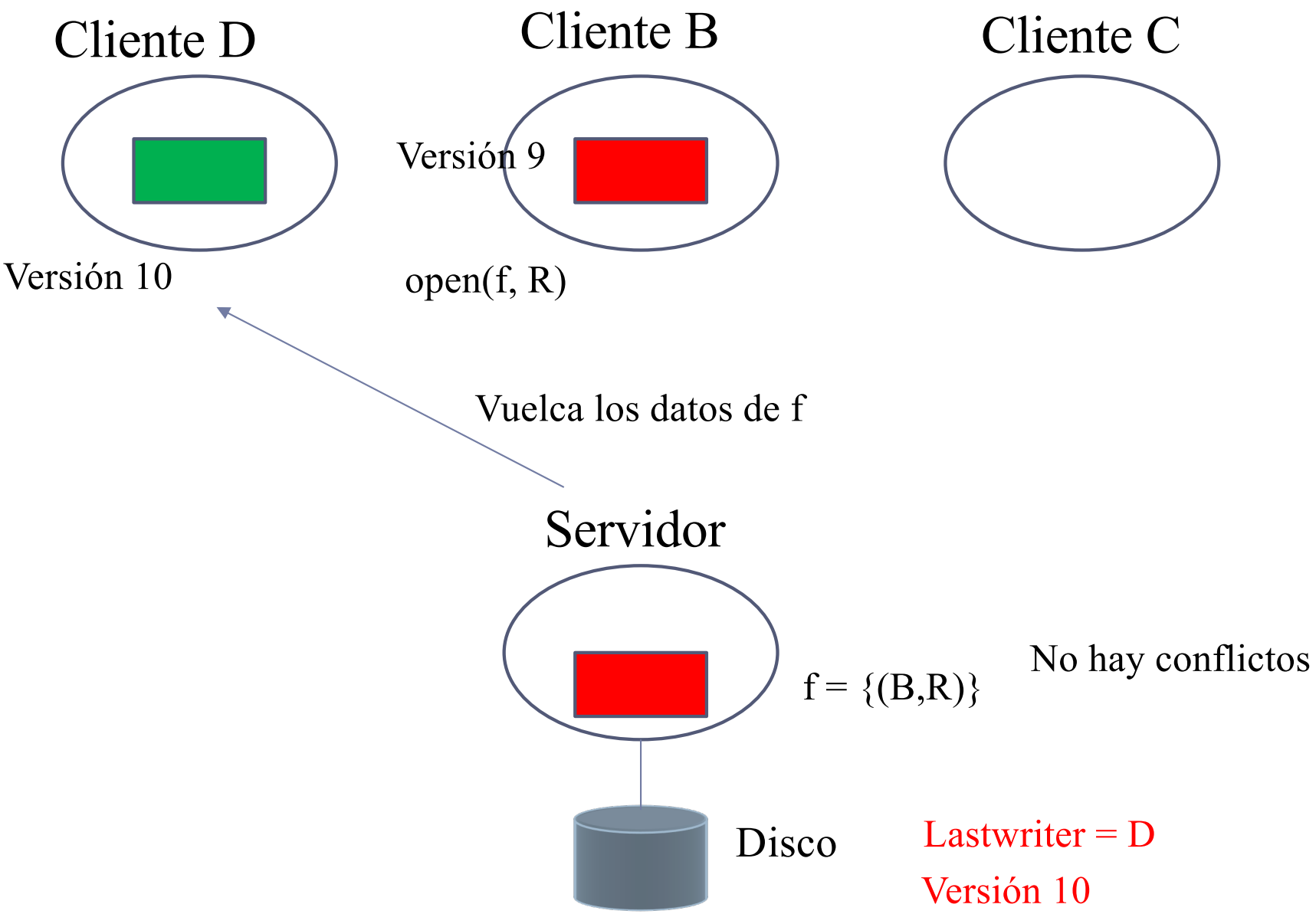
Coherencia de caché en Sprite



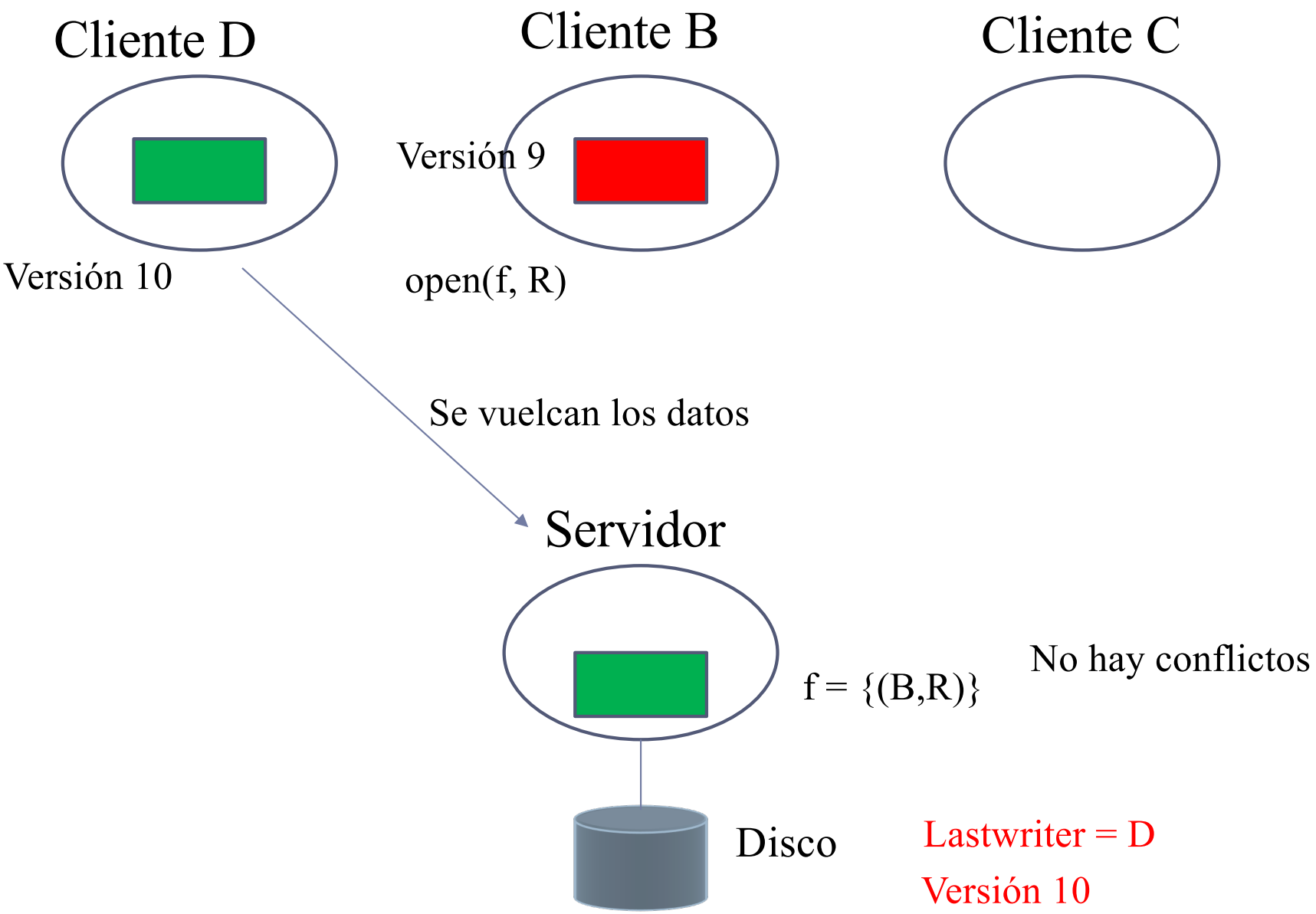
Coherencia de caché en Sprite



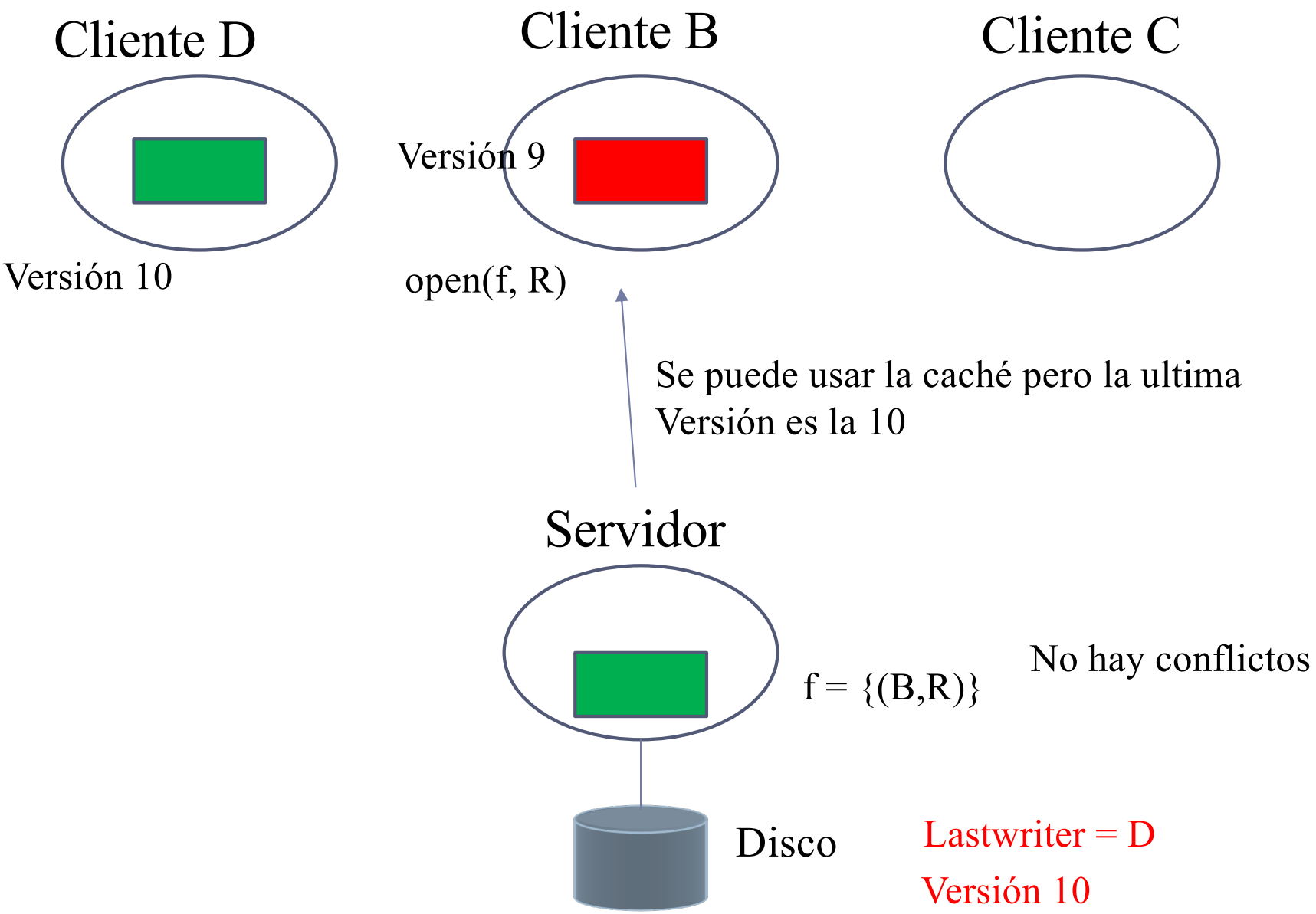
Coherencia de caché en Sprite



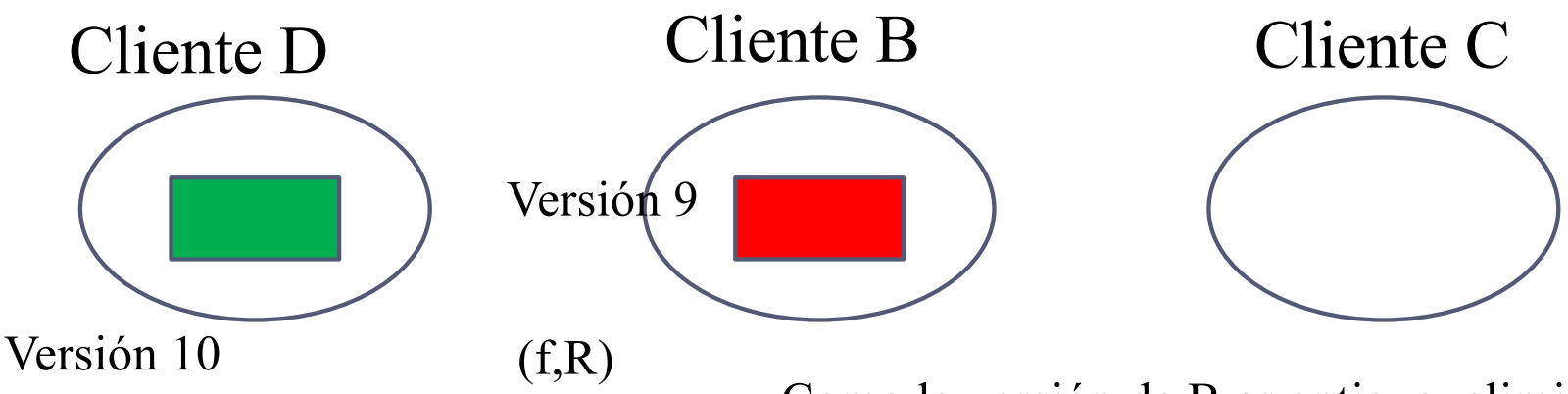
Coherencia de caché en Sprite



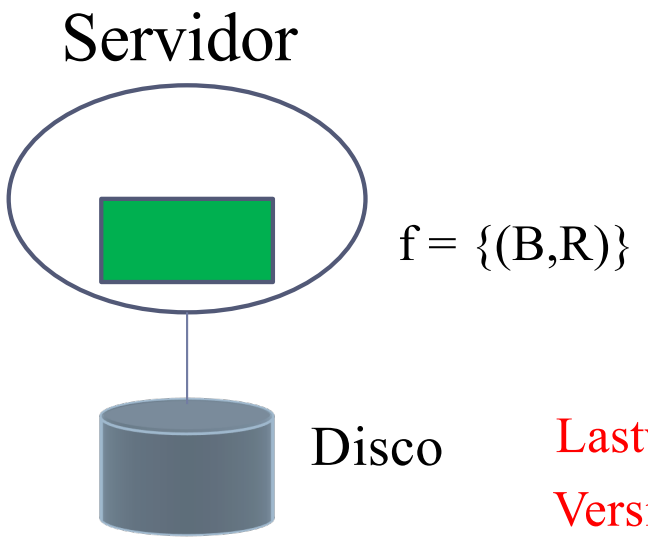
Coherencia de caché en Sprite



Coherencia de caché en Sprite



Como la versión de B es antigua, elimina
El contenido de la caché

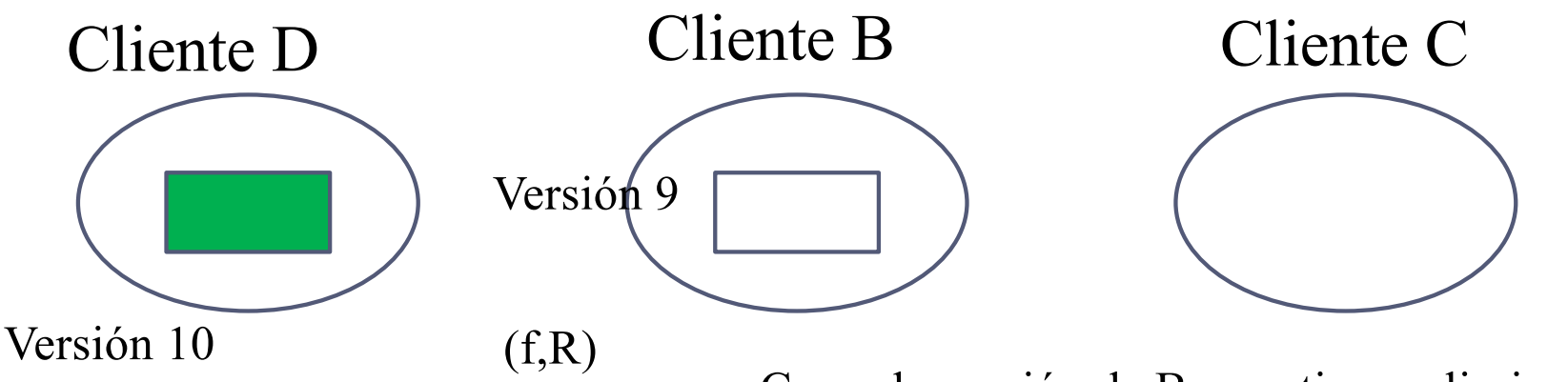


No hay conflictos

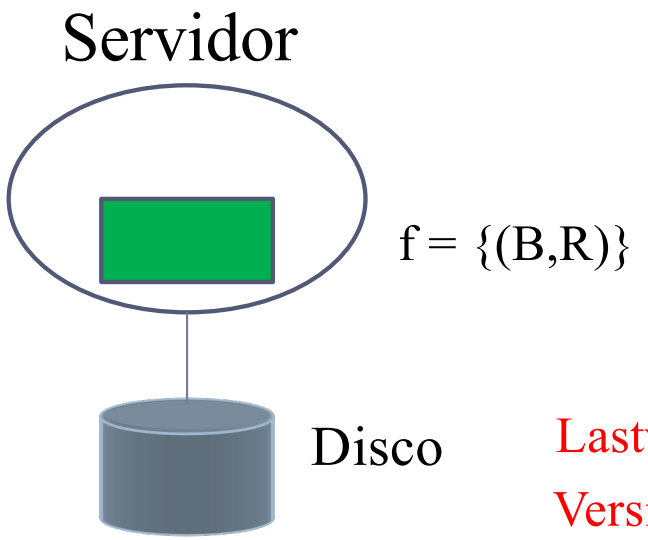
Lastwriter = D

Versión 10

Coherencia de caché en Sprite



Como la versión de B es antigua, elimina
El contenido de la caché

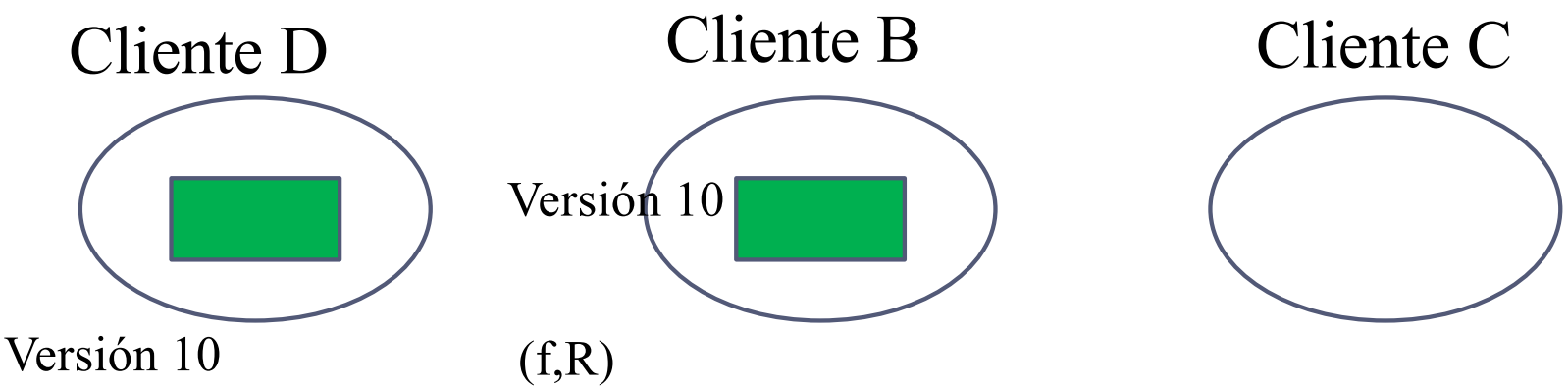


No hay conflictos

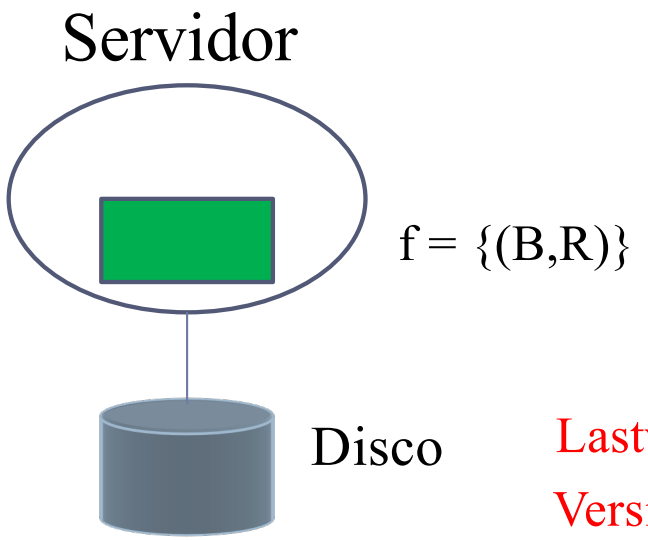
Lastwriter = D

Versión 10

Coherencia de caché en Sprite



Los futuros accesos de B a f se irán
Trayendo lo nuevos bloques del servidor



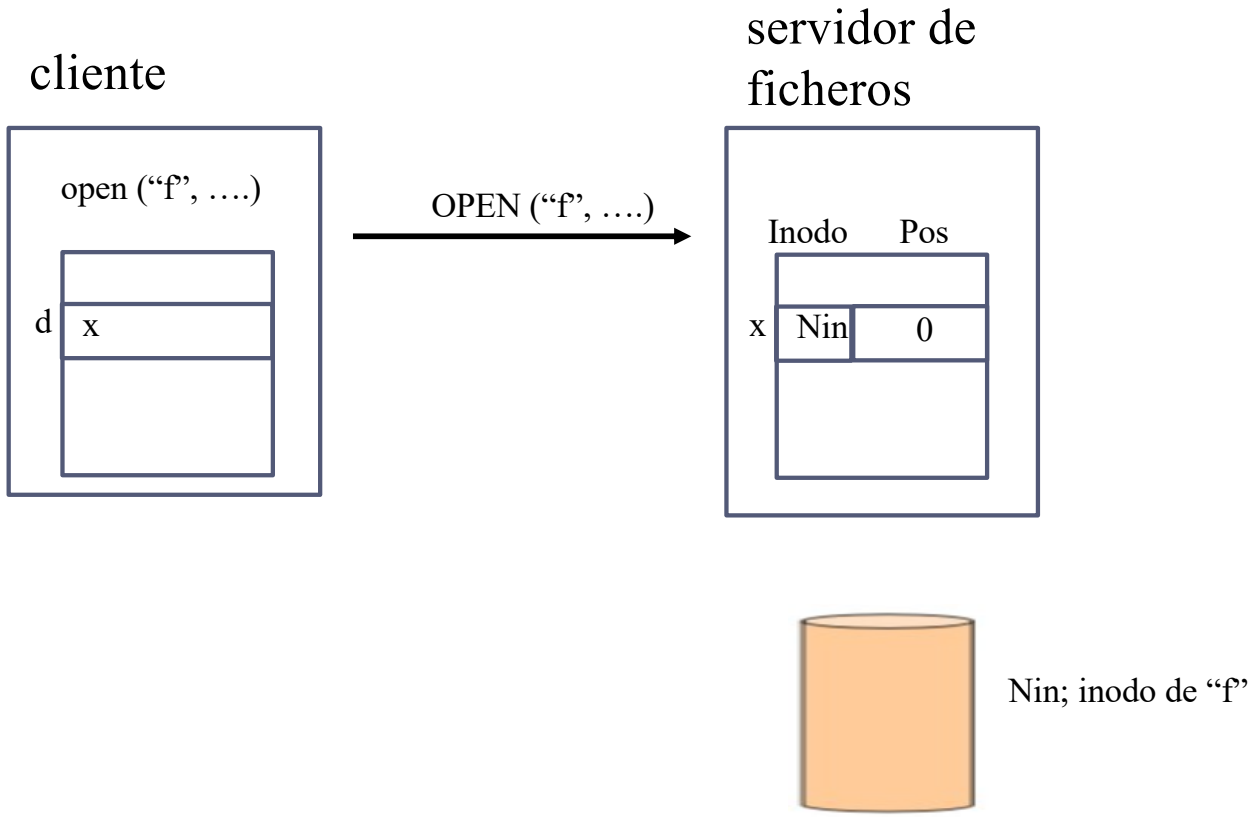
No hay conflictos

Lastwriter = D
Versión 10

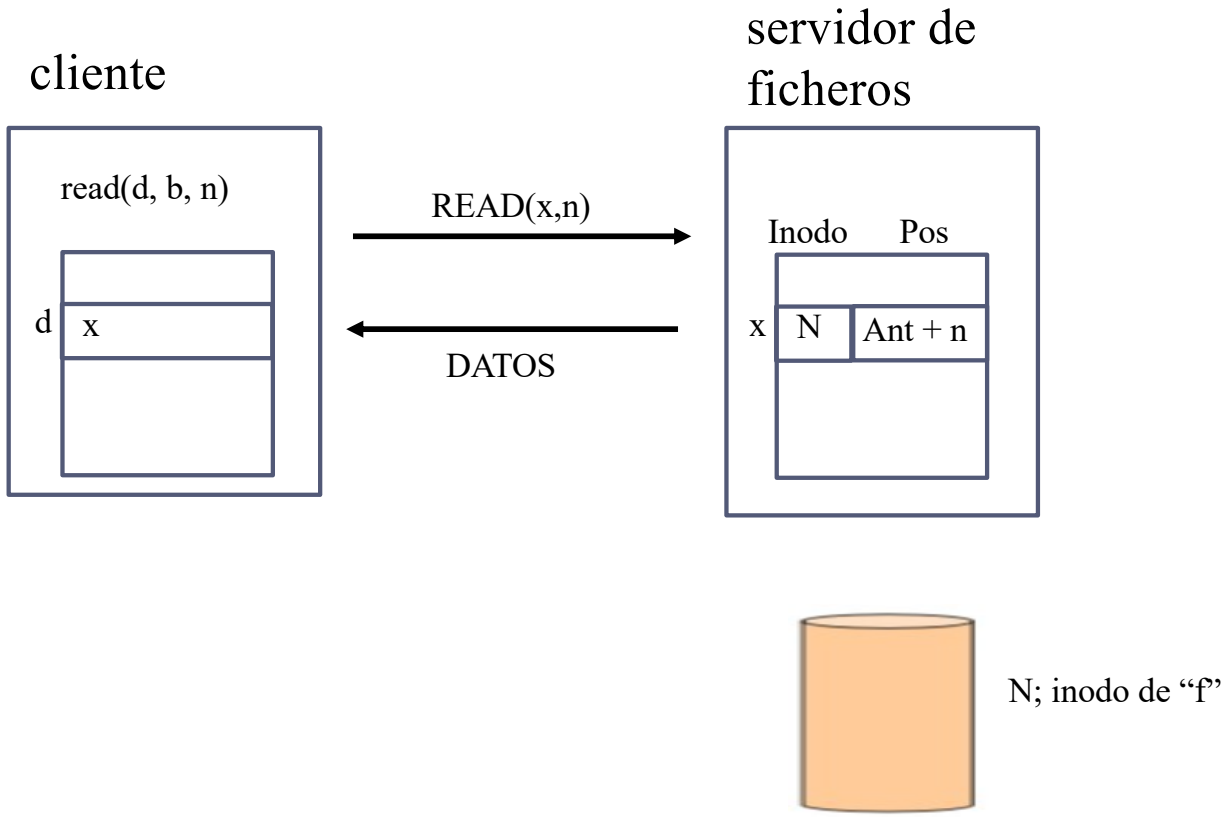
Aspectos de tolerancia a fallos

- **Servidores con estado**
  - Cuando se abre un fichero, el servidor almacena información y da al cliente un identificador único a utilizar en las posteriores llamadas
  - Cuando se cierra un fichero se libera la información
- **Servidores sin estado**
  - Cada petición es autocontenida (fichero y posición)

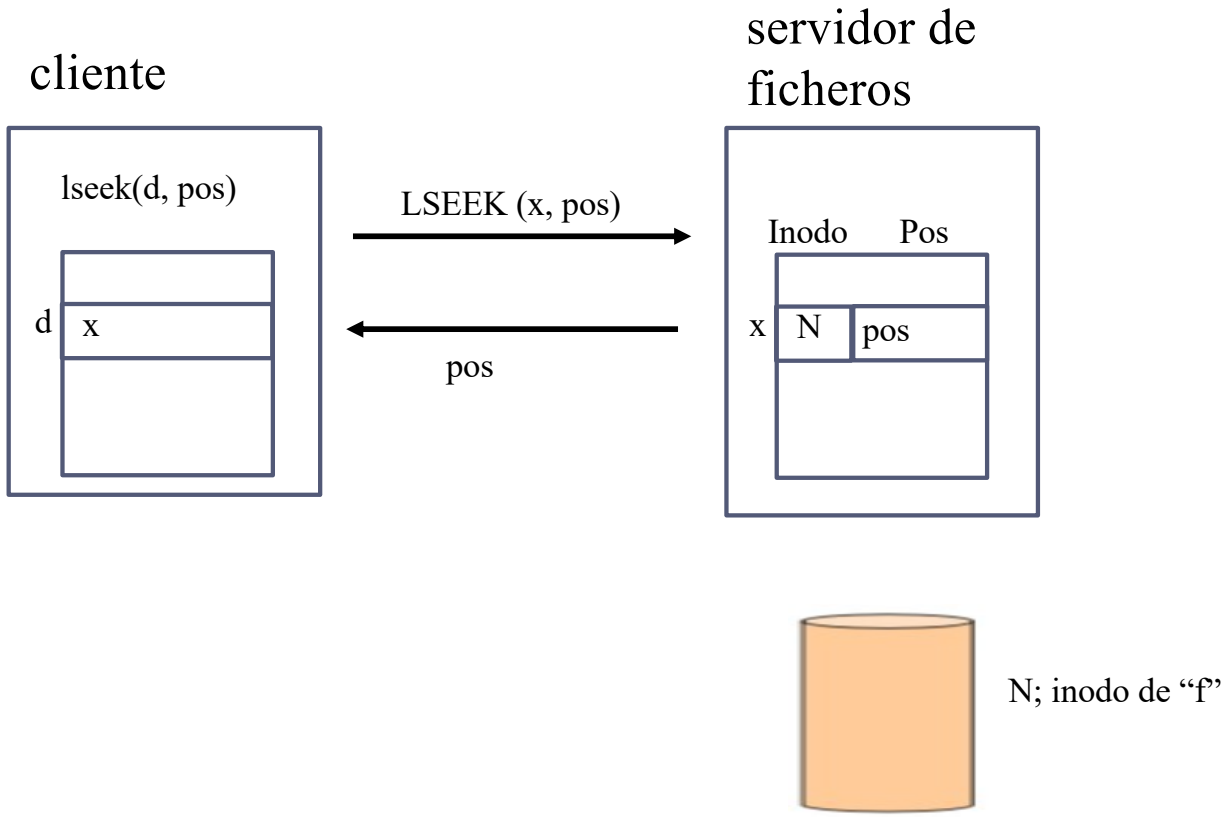
Servicio con estado
Ejemplo: servicio de ficheros



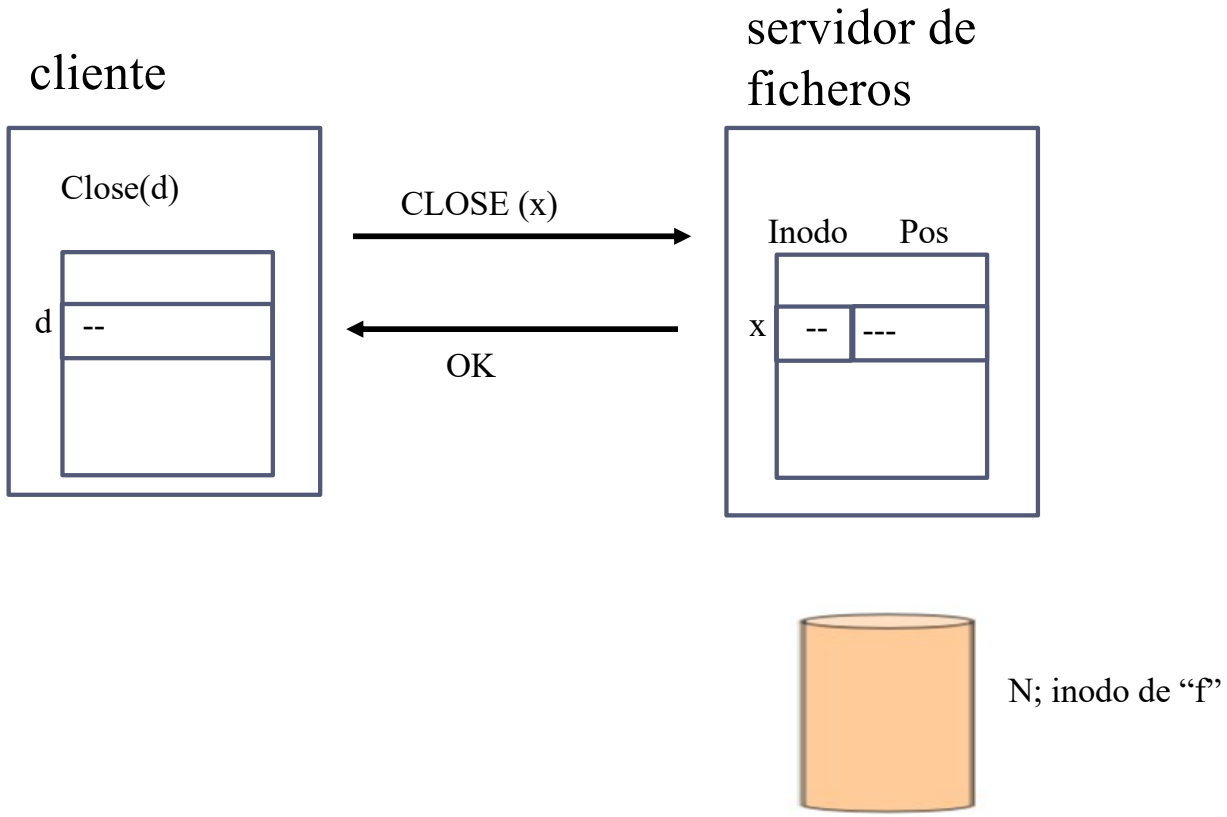
Servicio con estado
Ejemplo: servicio de ficheros



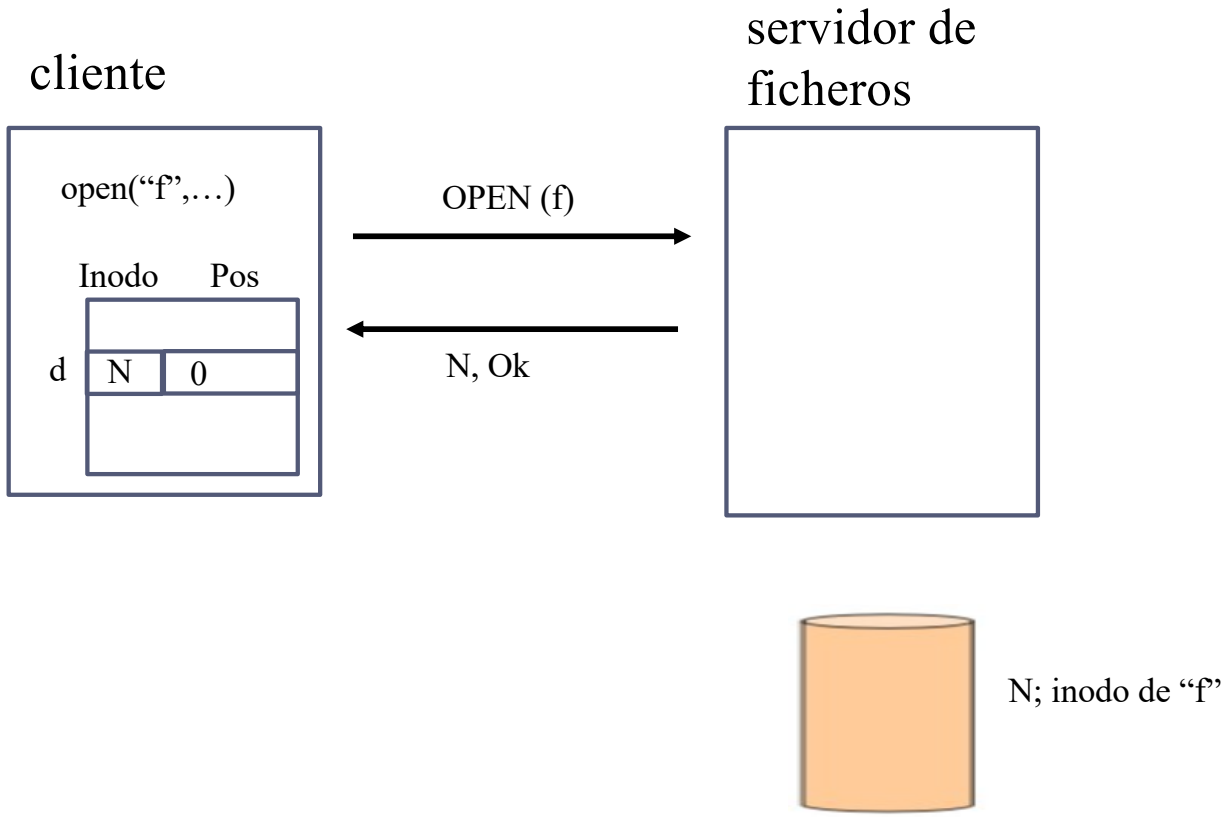
Servicio con estado
Ejemplo: servicio de ficheros



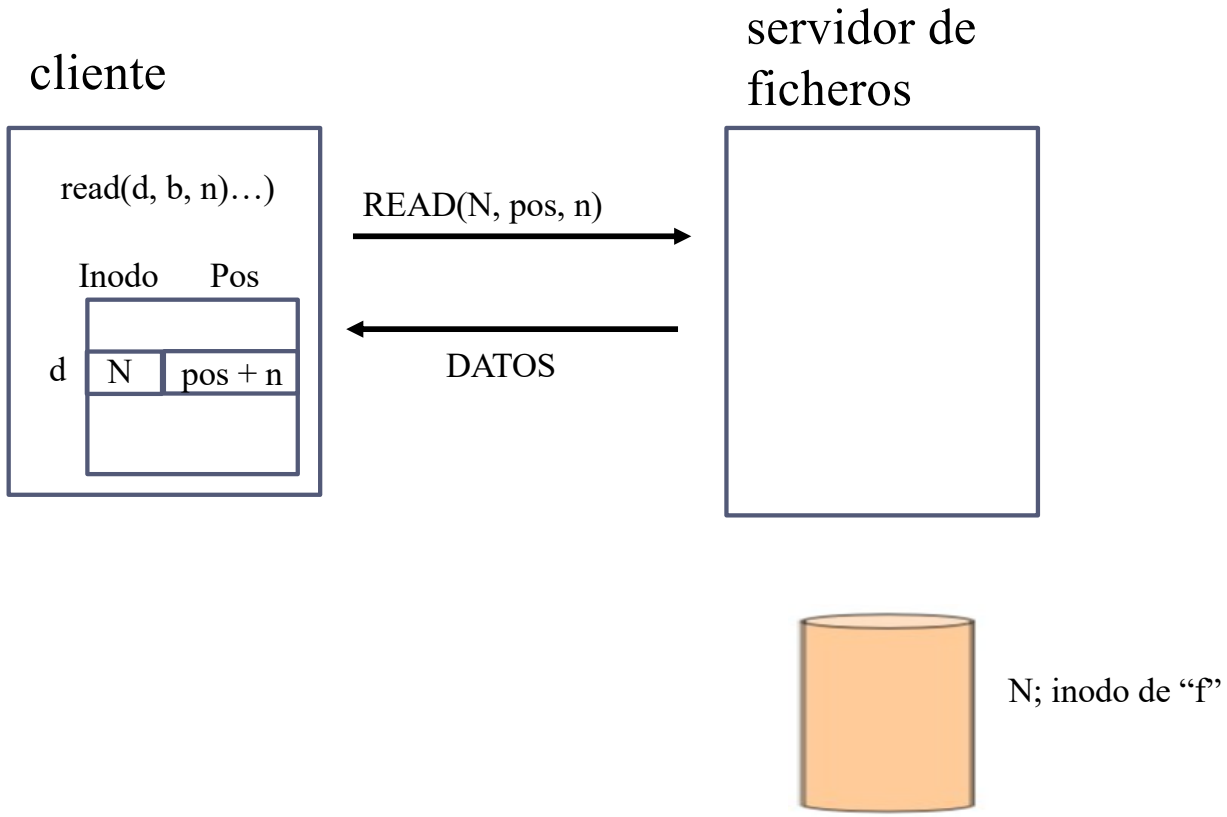
Servicio con estado
Ejemplo: servicio de ficheros



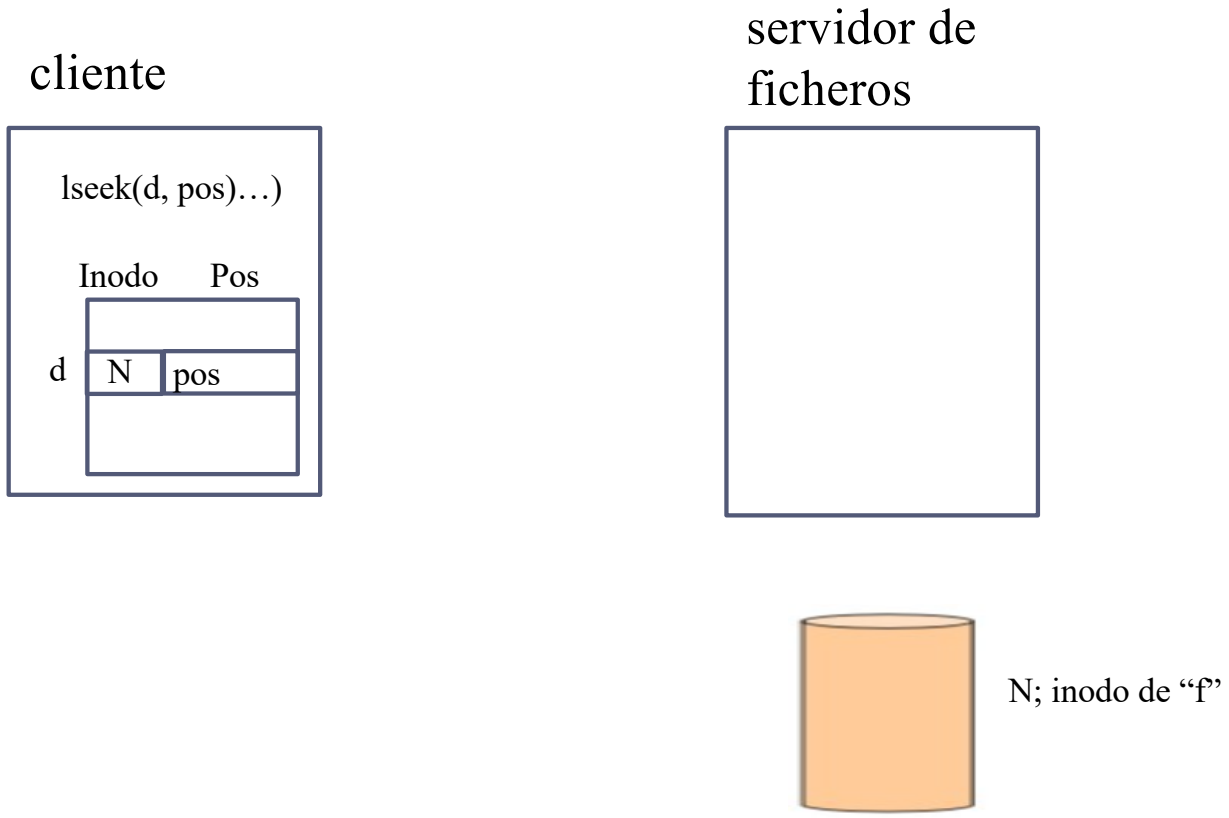
Servicio sin estado
Ejemplo: servicio de ficheros



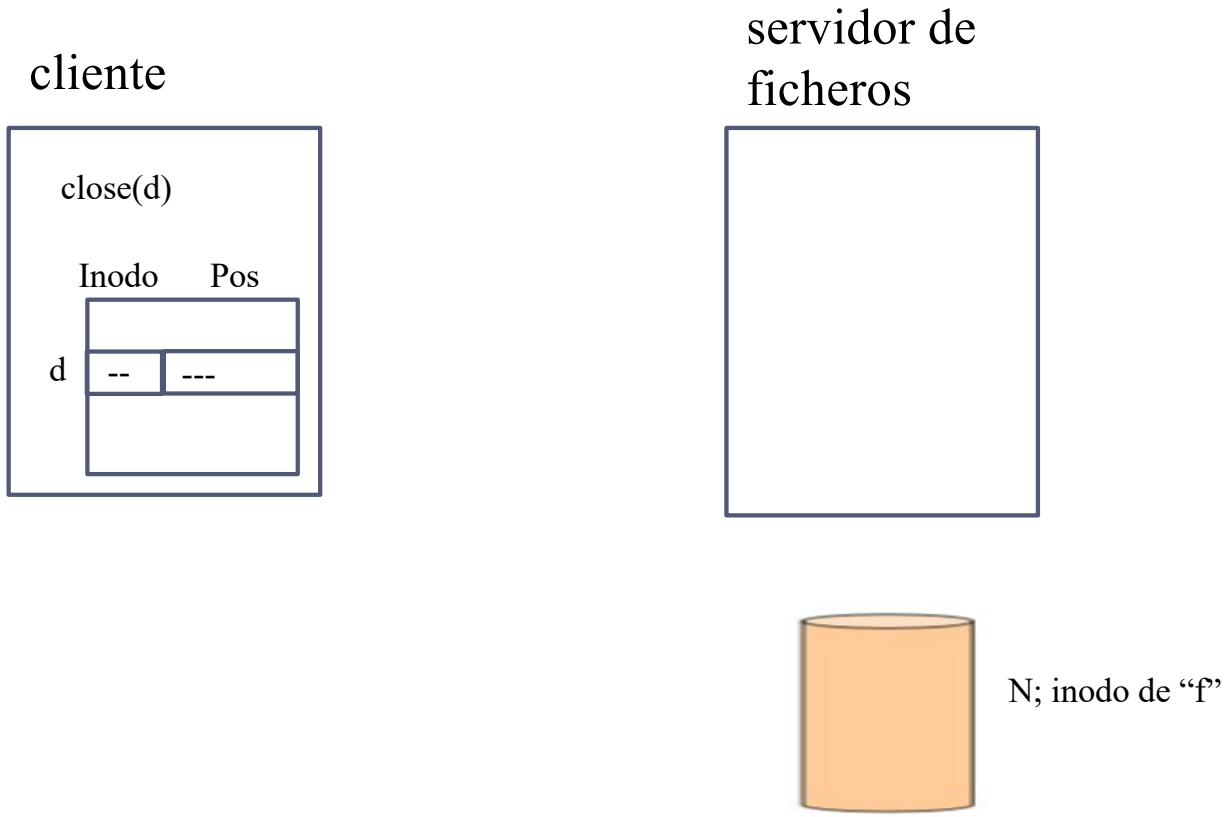
Servicio sin estado
Ejemplo: servicio de ficheros



Servicio sin estado
Ejemplo: servicio de ficheros



Servicio sin estado
Ejemplo: servicio de ficheros



Aspectos de tolerancia a fallos

- Ventajas de los servidores con estado
  - Mensajes de petición más cortos
  - Mejor rendimiento (se mantiene información en memoria)
  - Facilita la lectura adelantada. El servidor puede analizar el patrón de accesos que realiza cada cliente
  - Es necesario en invalidaciones iniciadas por el servidor
- Ventajas de los servidores sin estado
  - Más tolerante a fallos
  - No son necesarios *open* y *close*. Se reduce el nº de mensajes
  - No se gasta memoria en el servidor para almacenar el estado
- Otros aspectos: **replicación de ficheros**

¿Cómo mejorar el rendimiento?

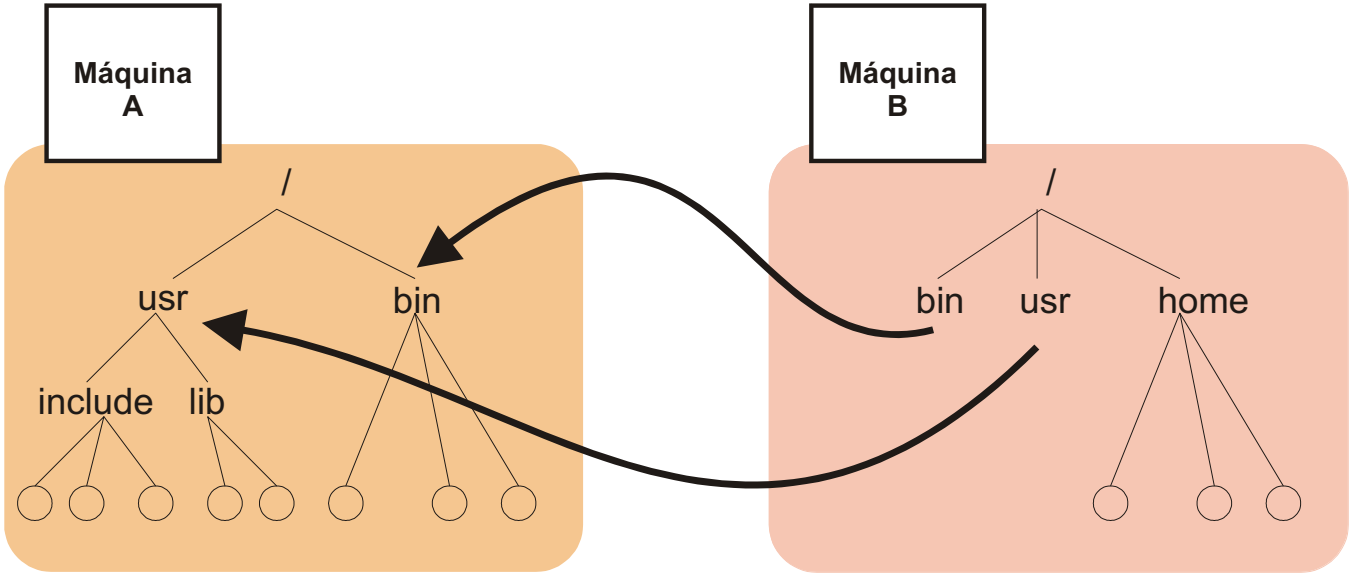
- Empleo de caché
  - Cuidado con los protocolos de coherencia de cache
- Uso de paralelismo
  - Múltiples discos y nodos de E/S
  - Distribución de los datos entre los discos y los nodos de E/S
  - Sistemas de ficheros paralelos

NFS (Network File System)

- Sistemas de ficheros de Sun Microsystem (**1985**)
- Implementación y especificación de un servicio de acceso a ficheros remotos
- Diseñado para trabajar en entornos heterogéneos (diferentes máquinas, sistemas operativos, ...)
- La independencia se consigue mediante el **uso de las RPC** construidas sobre el protocolo XDR
- Las diferentes máquinas **montan** un directorio remoto en el sistema de ficheros local
  - El espacio de nombres es diferente en cada máquina
  - El montado no es transparente, debe proporcionarse el nombre de la máquina remota

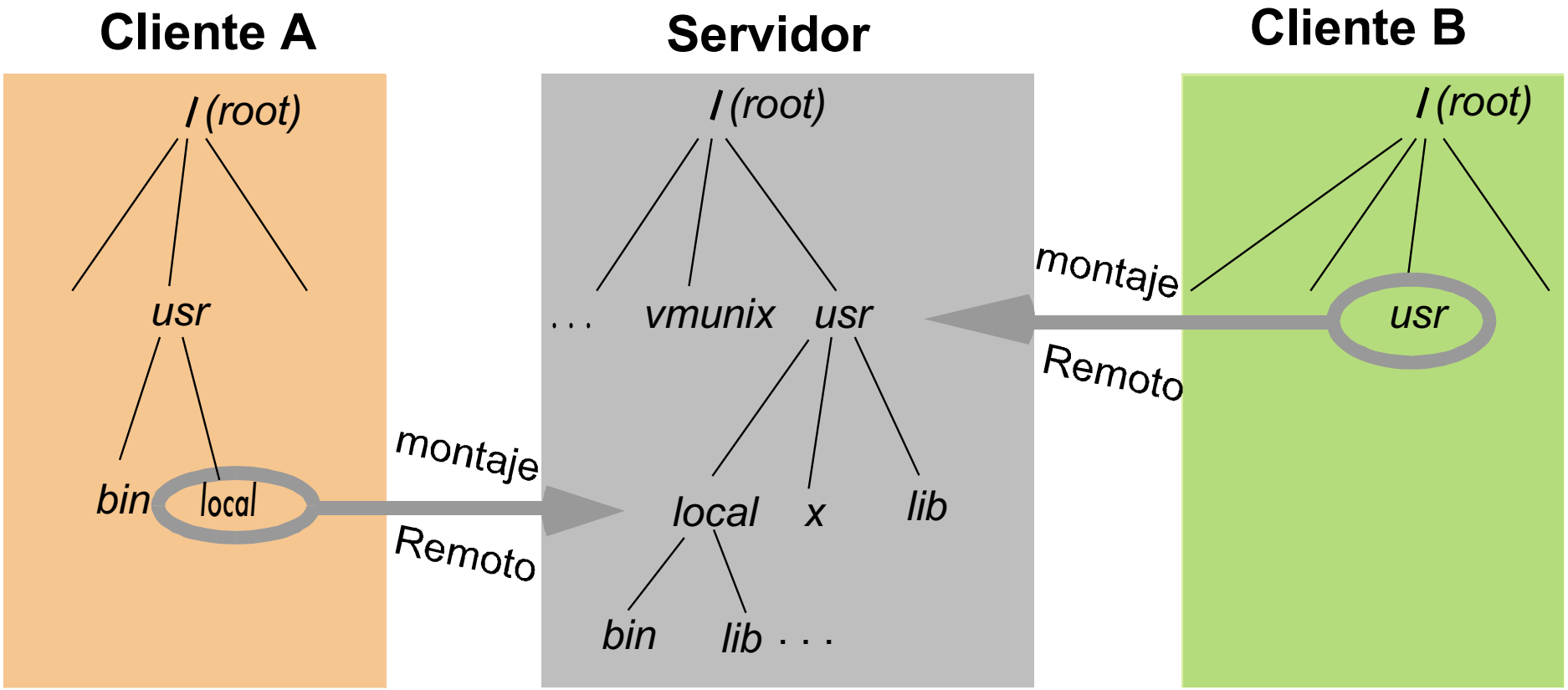
Montado en NFS

- Establece una conexión lógica entre el servidor y el cliente
- La máquina A exporta */usr* y */bin*
- En la máquina B:
  - *mount máquinaA:/usr /usr*

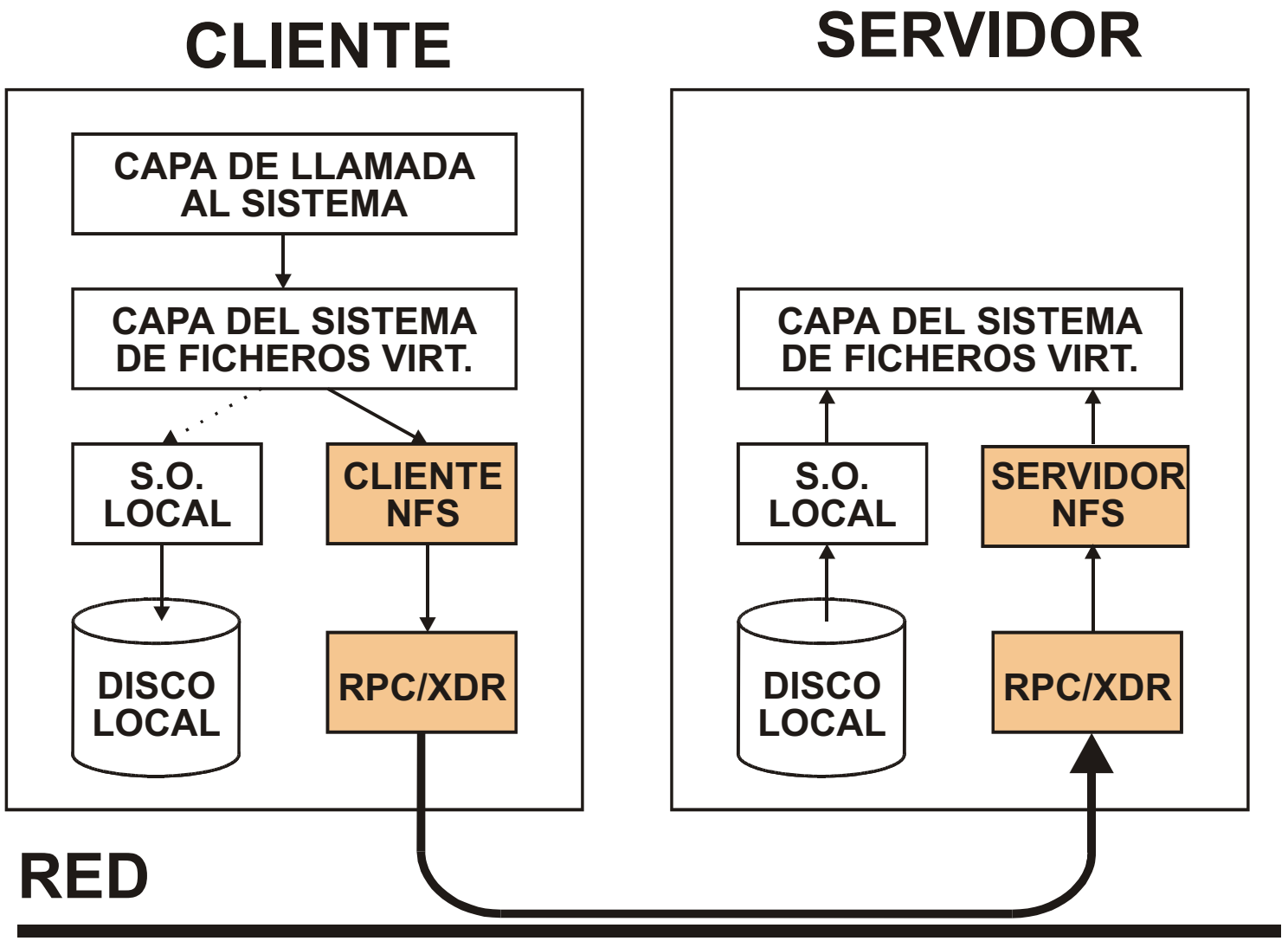


Montado en NFS

- La imagen del SF puede ser diferente en distintos clientes



Arquitectura de NFS



Protocolo NFS

- La versión 3 es estándar en Internet (**RFC 1813**)
- Ofrece un conjunto de RPC para realizar operaciones sobre ficheros remotos
  - Búsqueda de un fichero en un directorio
  - Lectura de entradas de directorio
  - Manipulación de enlaces y directorios
  - Acceso a los atributos de un fichero
  - Lectura y escritura de ficheros
- Los servidores de NFS **no** almacenan estado
  - Operaciones autocontenidas
    - El servidor no puede almacenar el puntero de la posición del fichero, lo debe mantener el cliente
    - El servidor no almacena información entre llamadas
- El protocolo **no** ofrece mecanismos de control de concurrencia para asegurar una semántica UNIX

Manejadores de ficheros (*file handle*)

- Estructura opaca que identifica a un fichero remoto
- Típico *filehandle* en UNIX

Id. del sistema de ficheros	Número de nodo-i	Número de generación de nodo-i

- VFS almacena una entrada por cada archivo abierto (*vnode*)
- Cada *vnode* apunta a un *inodo* local o a un *filehandle*
  - Cuando un fichero es local el nodo-v contiene el nodo-i.
  - Cuando el fichero es remoto el nodo-v contiene el *filehandle*

Operaciones

lookup(dirfh, name) -> fh, attr	Busca un fichero en un directorio dado por dirfh
create(dirfh, name, attr) -> newfh, attr	Crea un fichero con unos atributos en un directorio
remove(dirfh, name) status	Elimina un fichero de un directorio
getattr(fh) -> attr	Devuelve los atributos de un ficheros
setattr(fh, attr) -> attr	Modifica los atributos. Si el tamaño se pone a 0 se trunca el fichero
read(fh, offset, count) -> attr, data	Lectura de un fichero. Devuelve los últimos atributos del fichero.
write(fh, offset, count, data) -> attr	Escritura en un fichero. Devuelve los atributos después de realizada la escritura
rename(dirfh, name, todirfh, toname) -> status	Cambia el nombre de un fichero .
link(newdirfh, newname, dirfh, name) -> status	Crea un enlace

Operaciones

symlink(newdirfh, newname, string) -> status	Crea un enlace simbólico
readlink(fh) -> string	Devuelve el nombre asociado con un enlace simbólico
mkdir(dirfh, name, attr) -> newfh, attr	Crea un directorio
rmdir(dirfh, name) -> status	Elimina un directorio vacio
readdir(dirfh, cookie, count) -> entries	Devuelve hasta count bytes de entradas de un directorio Cada entrada contiene un nombre de fichero, su manejador y un Puntero opaco utilizado en la siguiente llamada readdir. Si cookie es 0 se lee la primera entrada del directorio.
statfs(fh) -> fsstats	Devuelve información del sistema de ficheros (block size, número de bloques libres, etc.

Traducción de nombres

- El cliente realiza la traducción de un nombre (*path*) componente a componente. Cuando un *vnode* apunta a *filehandle* la búsqueda se realiza en el servidor (una componente cada vez)
- **lookup(dir, name)** *devuelve* **(fh, attr)**
  - Devuelve el fh y los atributos de un fichero del directorio dir
  - Falla si el cliente no tiene permisos

Traducción de nombres

- Una única llamada open

  fd = open("/usr/joe/6360/list.txt")

  Da lugar a varias operaciones remotas:

  lookup(rootfh, "usr") devuelve **(fh0, attr)**
  lookup(fh0, "joe") devuelve **(fh1, attr)**
  lookup(fh1, "6360") devuelve **(fh2, attr)**
  lookup(fh2, "list.txt") devuelve **(fh, attr)**

- El *filehandle* **fh** identifica al fichero **list.txt** y se utilizará en las operaciones de lectura y escritura posteriores.

Acceso a los ficheros

- Se realiza con el filehandle:
  - **read(fh, offset, cont)** devuelve **(datos, attr)**
  - **write(fh, offset, cont, data)** devuelve **(attr)**

- Transferencias en bloques
  - En la versión 2 hasta 8 KB
  - En la versión 3 se puede negociar
- Los clientes realizan **lecturas adelantadas** de un bloque
- Las **escrituras** se realizan **localmente**. Los bloques se envían al servidor cuando se llena un bloque o cuando se cierra el fichero

Escritura en la caché del servidor

- Política de actualización en la versión 2: **write-through**
- Política de actualización en la versión 3: la operación de escritura ofrece dos posibilidades:
  - **Write-through:** se escribe a disco antes de contestar al cliente
  - **Delayed-write:** se escribe solo en memoria caché
    - La operación commit fuerza la escritura en disco. Normalmente los clientes envían esta operación en el close.

Caché en los clientes

- Necesaria para mejorar el **rendimiento**
- El **cliente NFS** almacena en caché el resultado de las operaciones: *read*, *write*, *getattr*, *lookup* y *readdir*
  - Caché de datos
  - Caché de atributos

- La caché introduce un posible **problema** de **coherencia**:
  - Dado un fichero modificado concurrentemente por dos clientes
  - Si el fichero se almacena en la caché de los dos clientes:
    - A no verá los cambios hechos por B
    - B no verá los cambios hechos por A

- **Actualizaciones inconsistentes**
- No se garantiza la semántica UNIX

Solución

- Los clientes deben:
  - Enviar periódicamente los bloques modificados al servidor
  - Cuando un fichero se abre en un cliente se comprueba en el servidor si la información se ha modificado
    - Si se ha modificado se descartan los bloques de la caché
  - Piden al servidor que revalide la información que tiene en su caché
    - 3-30 segundos para ficheros
    - 30-60 segundos para directorios
    - El bloque se descarta si ha sido modificado en el servidor
  - Los atributos en caché se descartan después de
    - 3 segundos para atributos de ficheros
    - 30 segundos para atributos de directorios

Protocolo de montado

```
program MOUNT_PROGRAM {
        version MOUNT_V3 {
           void      MOUNTPROC3_NULL(void)    = 0;
           mountres3 MOUNTPROC3_MNT(dirpath)  = 1;
           mountlist MOUNTPROC3_DUMP(void)    = 2;
           void      MOUNTPROC3_UMNT(dirpath) = 3;
           void      MOUNTPROC3_UMNTALL(void) = 4;
           exports   MOUNTPROC3_EXPORT(void)  = 5;
        } = 3;
      } = 100005;
```

Protocolo NFS

```
program NFS_PROGRAM {
    version NFS_V3 {
      void              NFSPROC3_NULL(void)                    = 0;
      GETATTR3res NFSPROC3_GETATTR(GETATTR3args)          = 1;
      SETATTR3res NFSPROC3_SETATTR(SETATTR3args)          = 2;
      LOOKUP3res  NFSPROC3_LOOKUP(LOOKUP3args)            = 3;
      ACCESS3res  NFSPROC3_ACCESS(ACCESS3args)            = 4;
      READLINK3res        NFSPROC3_READLINK(READLINK3args)       = 5;
      READ3res            NFSPROC3_READ(READ3args)               = 6;
      WRITE3res   NFSPROC3_WRITE(WRITE3args)              = 7;
      CREATE3res  NFSPROC3_CREATE(CREATE3args)            = 8;
      MKDIR3res   NFSPROC3_MKDIR(MKDIR3args)              = 9;
      SYMLINK3res NFSPROC3_SYMLINK(SYMLINK3args)          = 10;
      MKNOD3res   NFSPROC3_MKNOD(MKNOD3args)              = 11;
      REMOVE3res  NFSPROC3_REMOVE(REMOVE3args)            = 12;
```

Protocolo NFS

```
    RMDIR3res   NFSPROC3_RMDIR(RMDIR3args)              = 13;
    RENAME3res  NFSPROC3_RENAME(RENAME3args)            = 14;
    LINK3res            NFSPROC3_LINK(LINK3args)                = 15;
    READDIR3resNFSPROC3_READDIR(READDIR3args)           = 16;
    READDIRPLUS3res     FSPROC3_READDIRPLUS(READDIRPLUS3args) = 17;
    FSSTAT3res  NFSPROC3_FSSTAT(FSSTAT3args)            = 18;
    FSINFO3res  NFSPROC3_FSINFO(FSINFO3args)            = 19;
    PATHCONF3res        NFSPROC3_PATHCONF(PATHCONF3args)        = 20;
    COMMIT3res  NFSPROC3_COMMIT(COMMIT3args)            = 21;
  } = 3;
} = 100003;
```

LOOKUP3res
NFSPROC3_LOOKUP(LOOKUP3args)

```
struct LOOKUP3args {
          diropargs3  what;
      };
struct diropargs3 {
         nfs_fh3     dir;
         filename3   name;
      };

union LOOKUP3res switch (nfsstat3 status) {
      case NFS3_OK:
           LOOKUP3resok    resok;
      default:
           LOOKUP3resfail  resfail;
      };
struct LOOKUP3resok {
           nfs_fh3      object;
           post_op_attr obj_attributes;
           post_op_attr dir_attributes;
      };
```

WRITE3res
NFSPROC3_WRITE(WRITE3args)

```
struct WRITE3args {
          nfs_fh3     file;
          offset3     offset;
          count3      count;
          stable_how  stable;
          opaque      data<>;
     };
union WRITE3res switch (nfsstat3 status) {
     case NFS3_OK:
          WRITE3resok    resok;
     default:
          WRITE3resfail  resfail;
     };
struct WRITE3resok {
          wcc_data    file_wcc;
          count3      count;
          stable_how  committed;
          writeverf3  verf;
     };
```

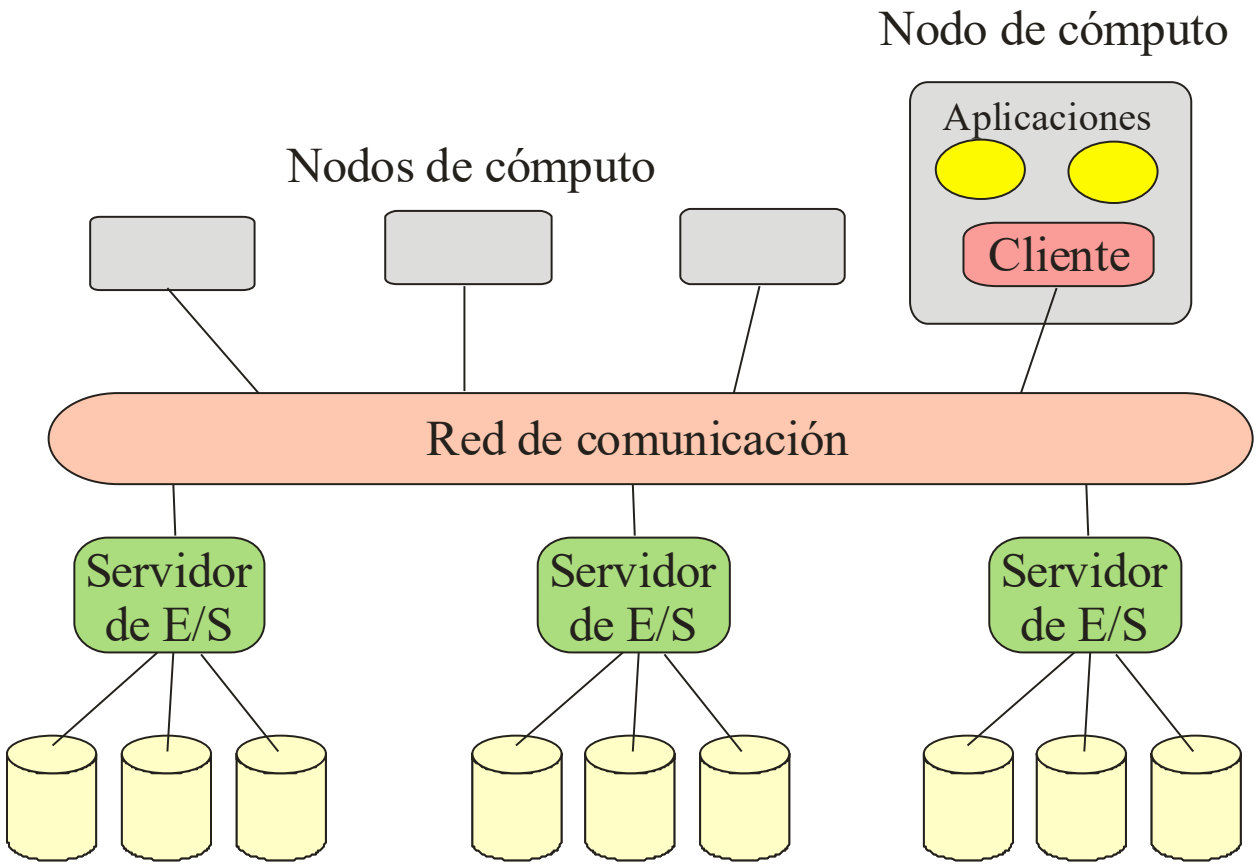
READ3res NFSPROC3_READ(READ3args)

```
struct READ3args {
          nfs_fh3  file;
          offset3  offset;
          count3   count;
     };
 union READ3res switch (nfsstat3 status) {
     case NFS3_OK:
          READ3resok   resok;
     default:
          READ3resfail resfail;
     };
 struct READ3resok {
          post_op_attr   file_attributes;
          count3         count;
          bool           eof;
          opaque         data<>;
     };
```

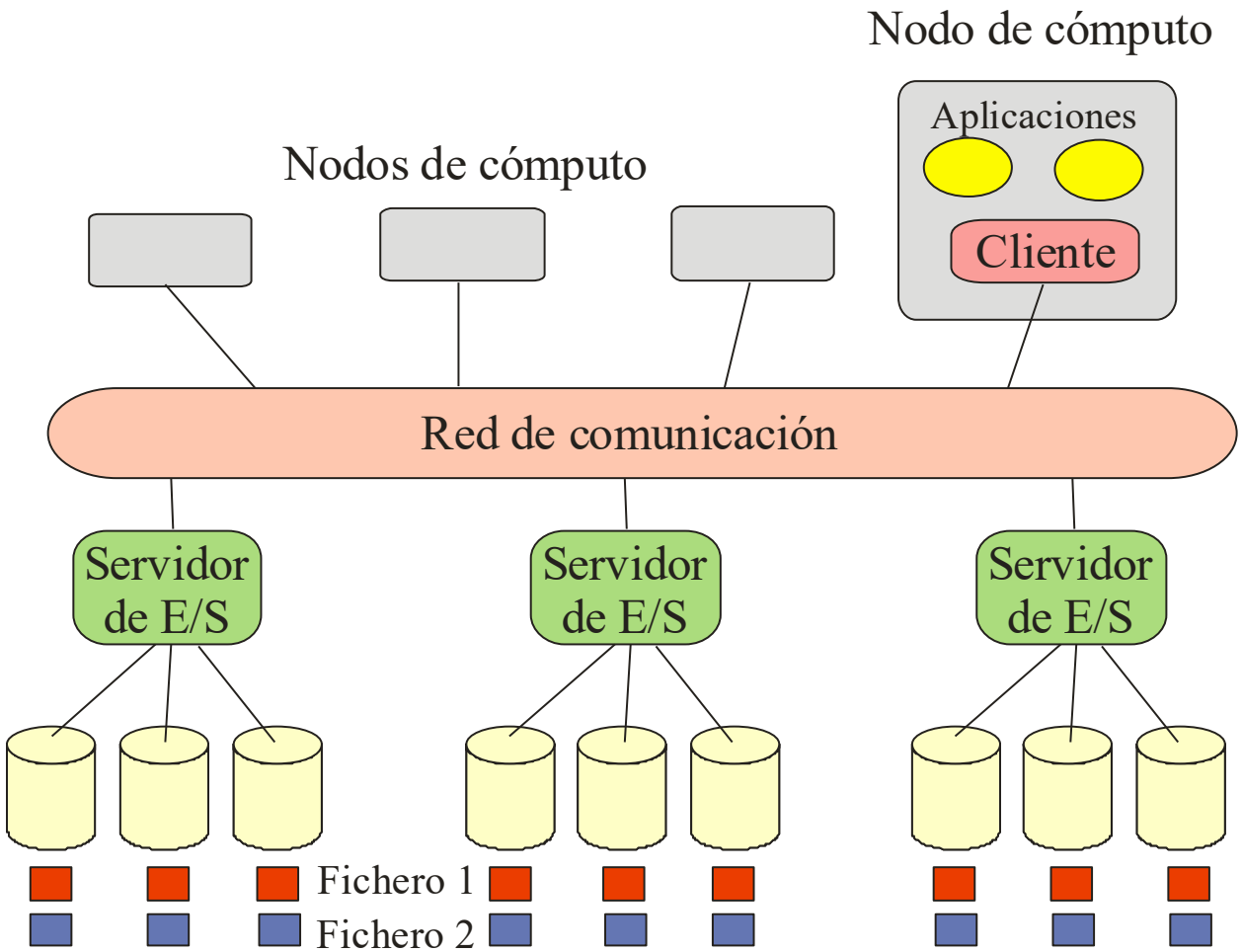
Sistemas de ficheros paralelos (PFS)

- Múltiples nodos de E/S
  - Incrementa el ancho de banda
- Fichero distribuido entre diferentes nodos de E/S
  - Acceso paralelo
    - A diferentes ficheros
    - Al mismo fichero
- Interfaces de E/S paralela
  - MPI-IO
- Optimizaciones
  - E/S colectiva
  - Acceso a datos no contiguos

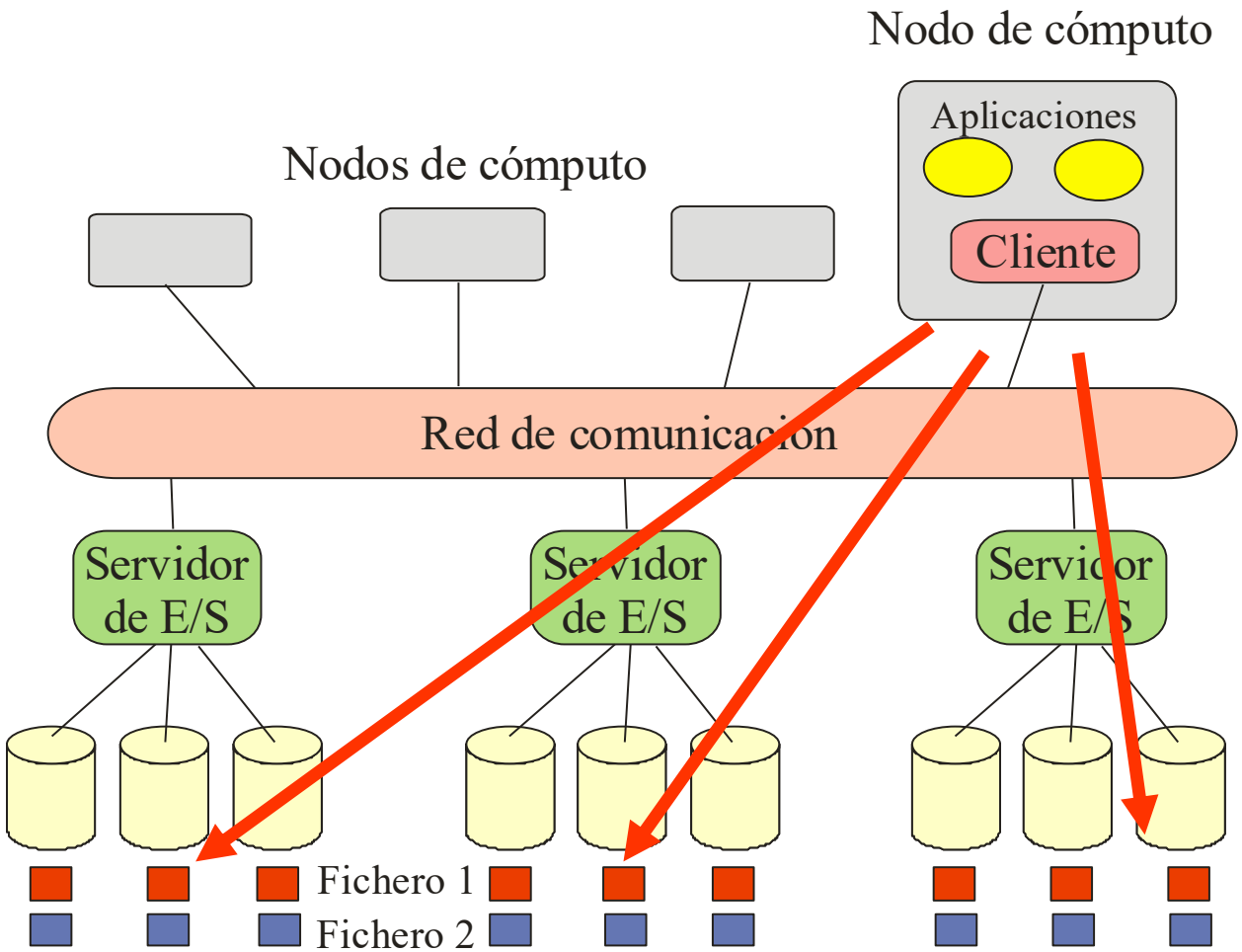
Arquitectura de un sistema de ficheros paralelo



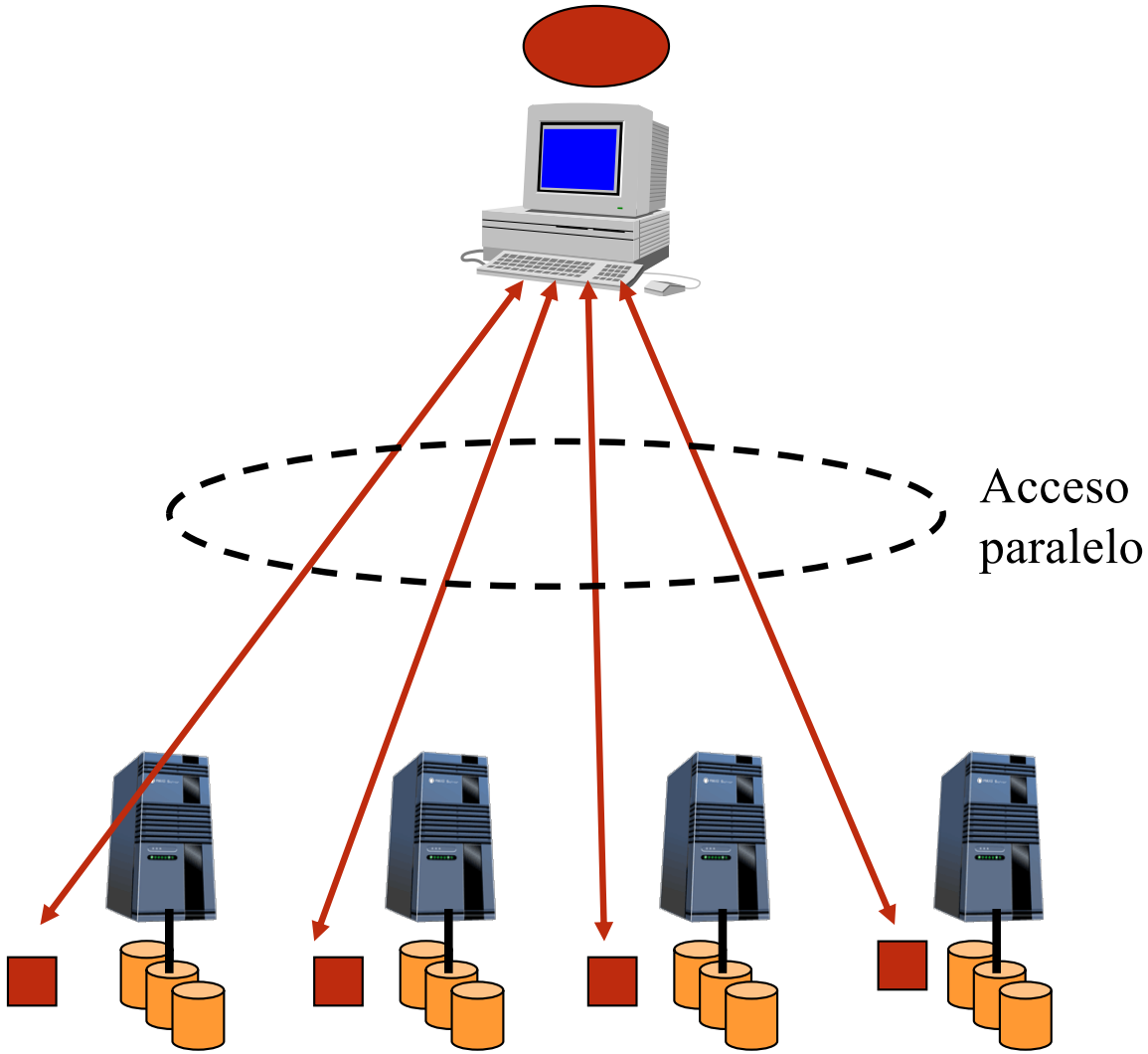
Arquitectura de un sistema de ficheros paralelo



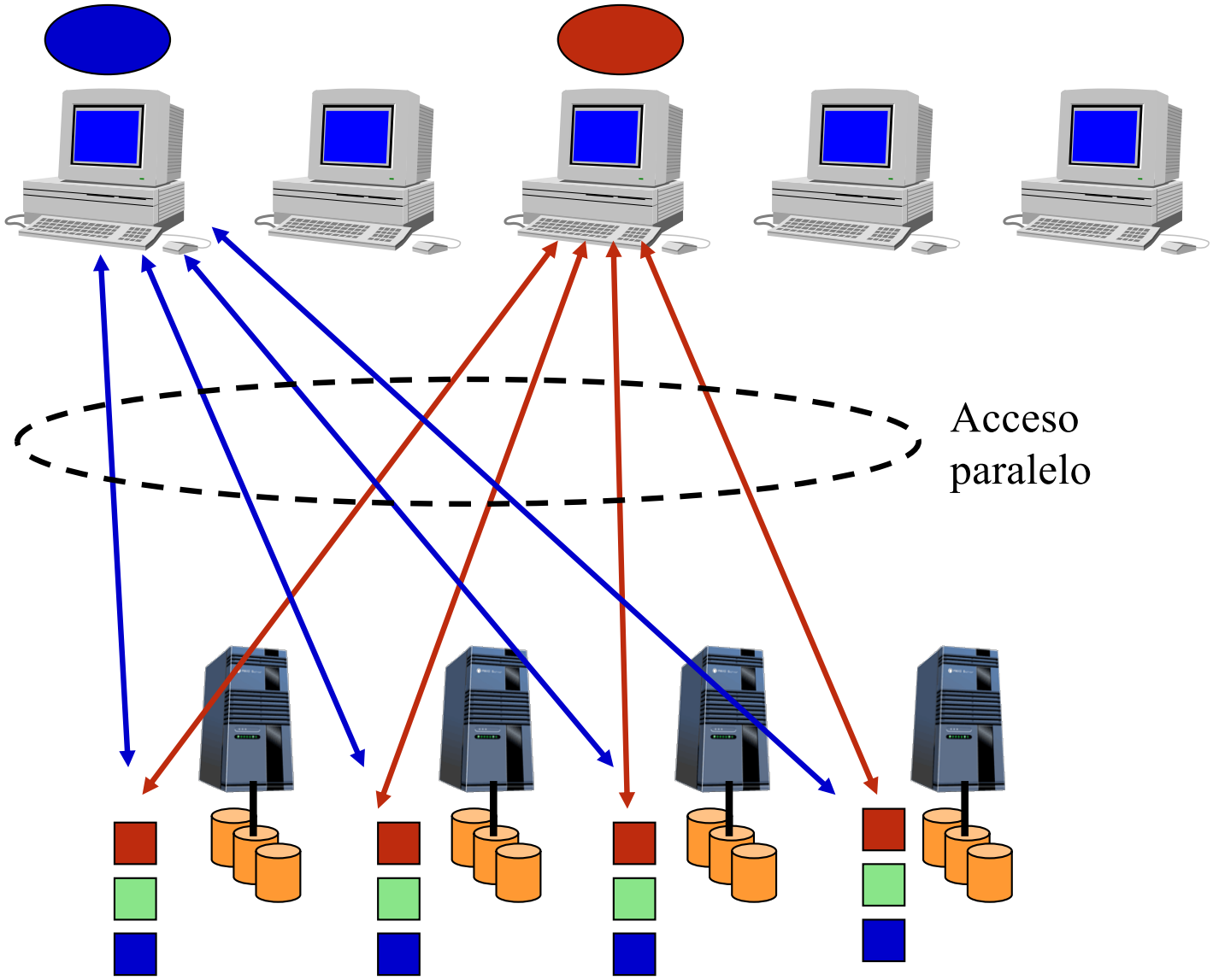
Arquitectura de un sistema de ficheros paralelo



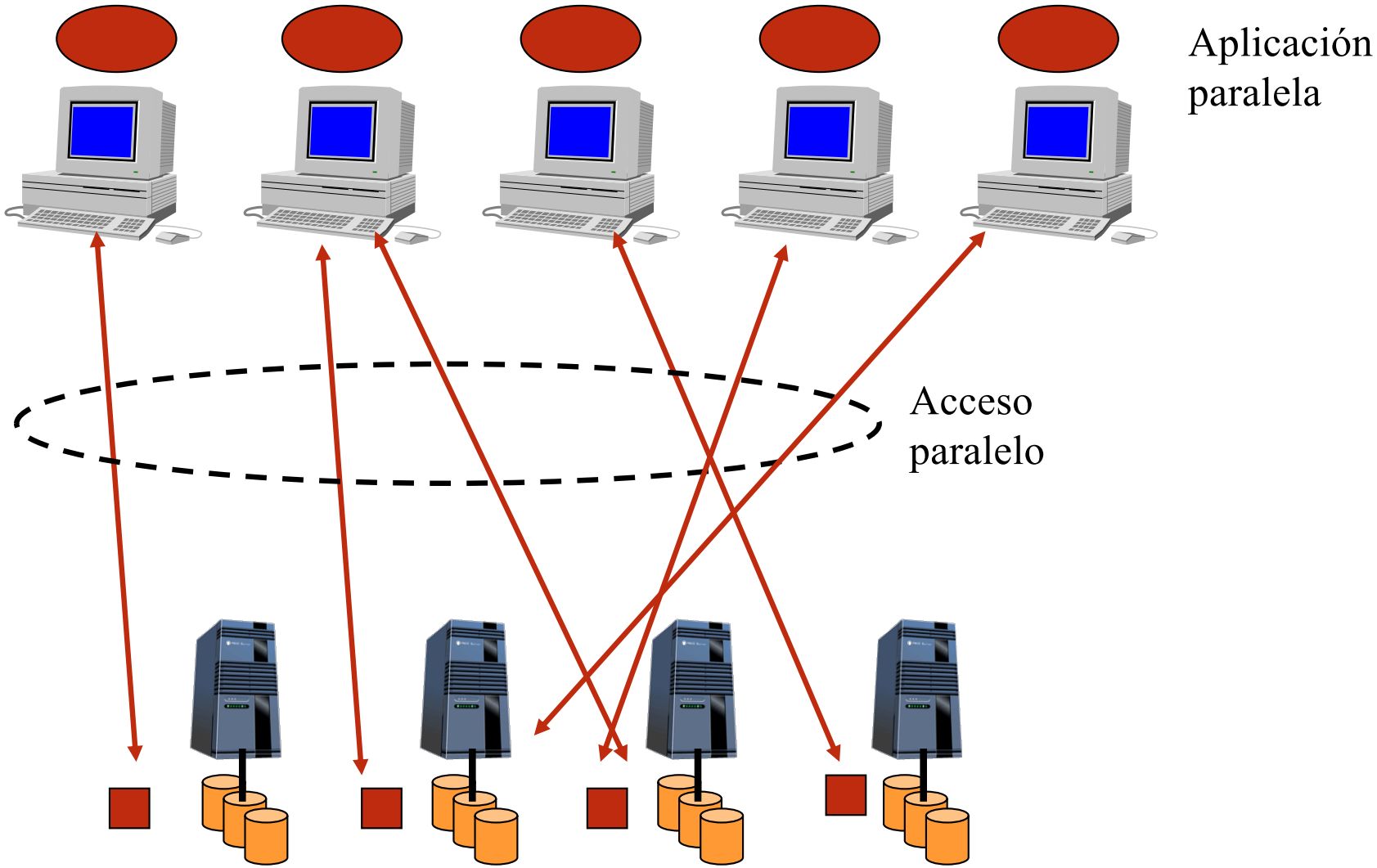
Acceso paralelo



Acceso paralelo



Acceso paralelo

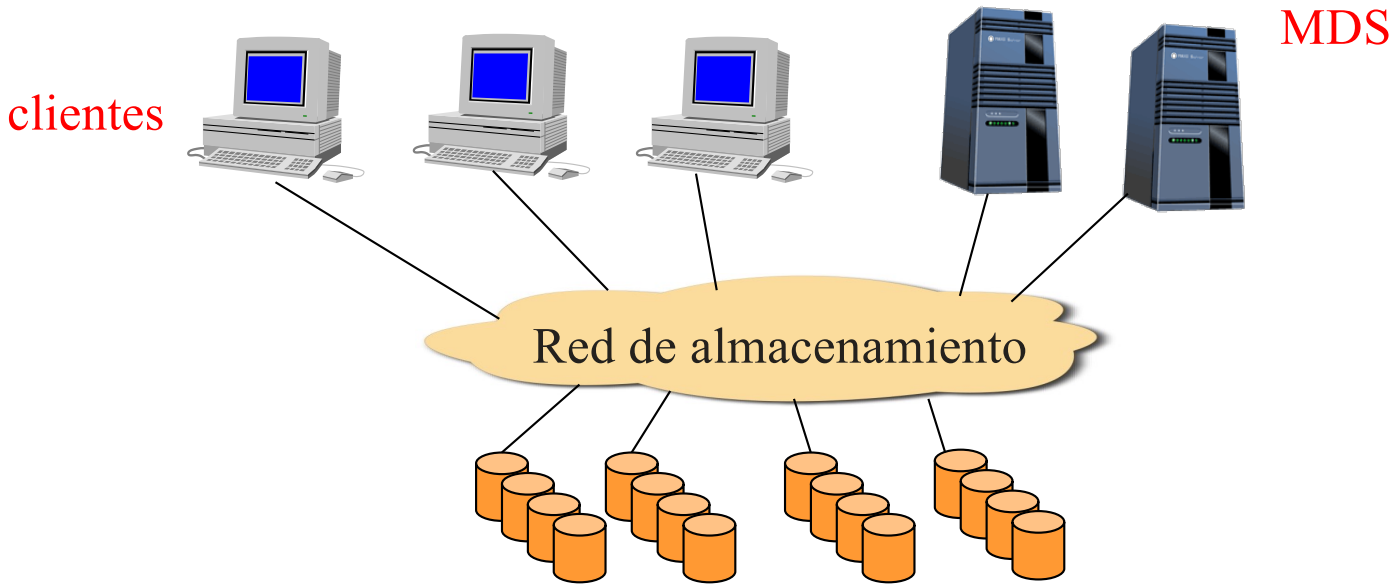


Ejemplos

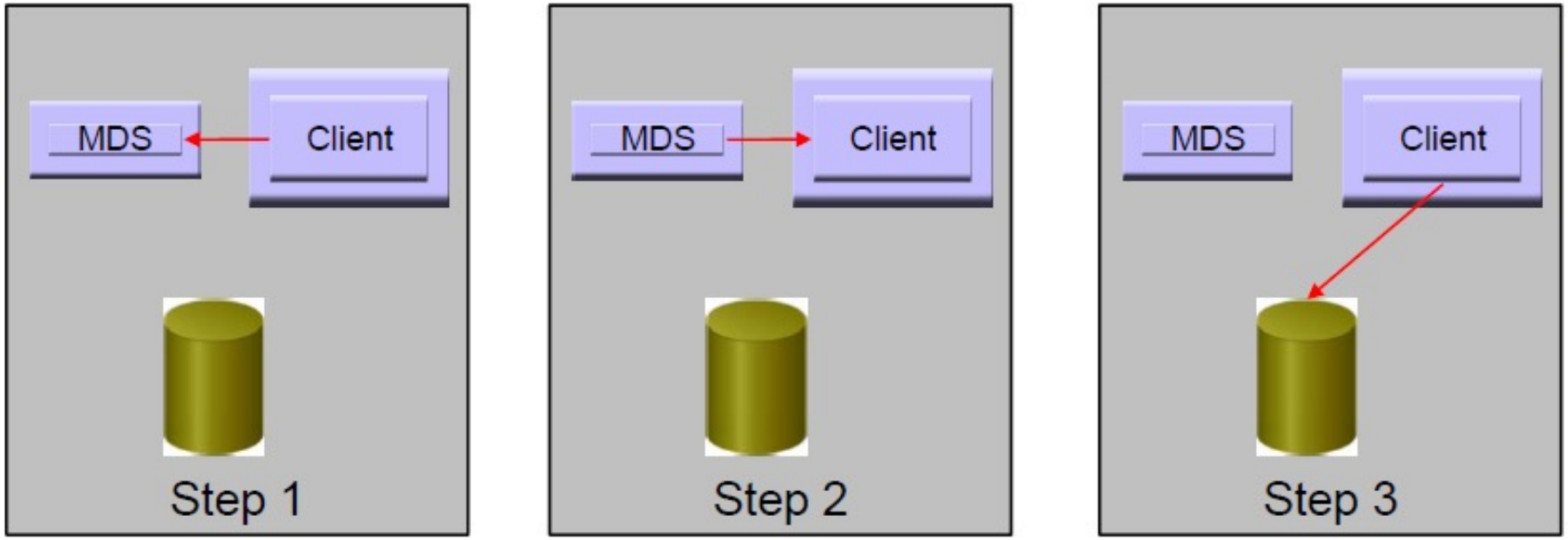
- PVFS
- GPFS

Sistemas de ficheros de discos compartidos

- Sistema de almacenamiento compartido por múltiples clientes
- Separación entre la localización física y lógica
- Servidores de metadatos (MDS)
- Los clientes acceden directamente al almacenamiento sin pasar por servidores



Mecanismo de acceso



Ventajas

- Se mejora el rendimiento
  - Acceso directo a los discos sin intervención de un servidor
  - Acceso paralelo
    - Acceso a grandes ficheros
    - Acceso a muchos ficheros pequeños
  - Mejor disponibilidad
    - No existe un servidor central, requiere disponibilidad en los discos

Ejemplos

- GPFS
- GFS
- OCFS
- HDFS

Hadoop file system

- Sistema de ficheros distribuido diseñado por Apache Foundation Inc.
- http://hadoop.apache.org/hdfs/

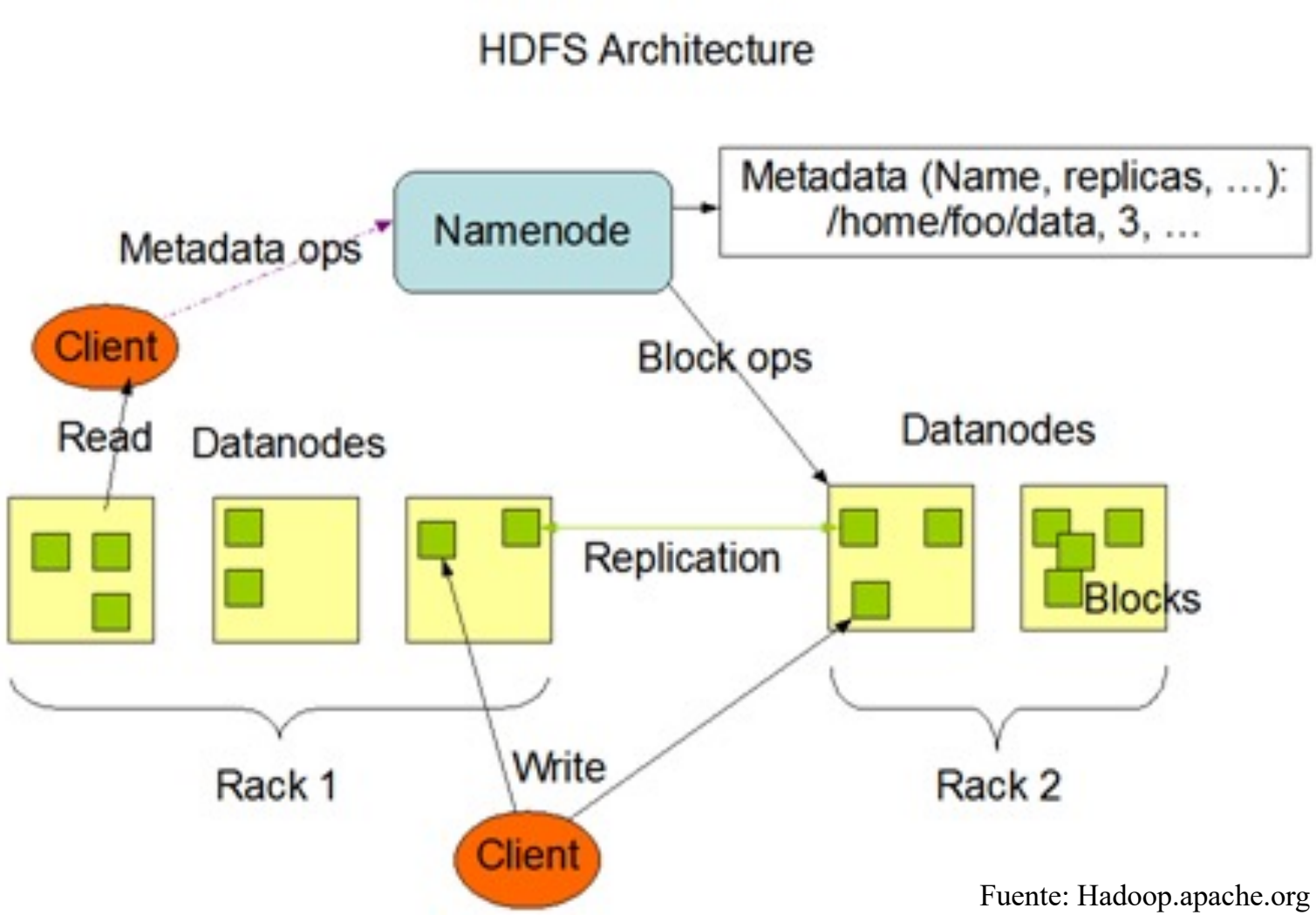
Características básicas

- Adecuado para ejecutar sobre HW de propósito general (clusters)
- Tolerancia a fallos
- Gran ancho de banda
  - El objetivo es gran ancho de banda, no baja latencia
  - No está pensado para uso interactivo (procesamiento *batch*)
- Adecuado para aplicaciones que procesan grandes cantidades de datos
  - Grandes ficheros y conjuntos de datos: GB hasta TB
  - Decenas de millones de ficheros
- *Streaming data access*
- Modelo de coherencia: *write-once-read-many*

Arquitectura

- Arquitectura maestro/esclavo
- Nodo **Namenode** (único)
  - Gestiona el espacio de nombres (único para todo el cluster) y regula el acceso a los ficheros
  - Espacio de nombres jerárquico
- **DataNodes** (normalmente uno por nodo en el cluster)
  - Servidor de bloques en el sistema de ficheros local (ej: ext3)
  - Gestiona el almacenamiento del nodo
  - Los ficheros se dividen en bloques de 128 MB
  - Cada bloque se replica en varios DataNodes
  - Los clientes acceden a los DataNodes directamente

Arquitectura



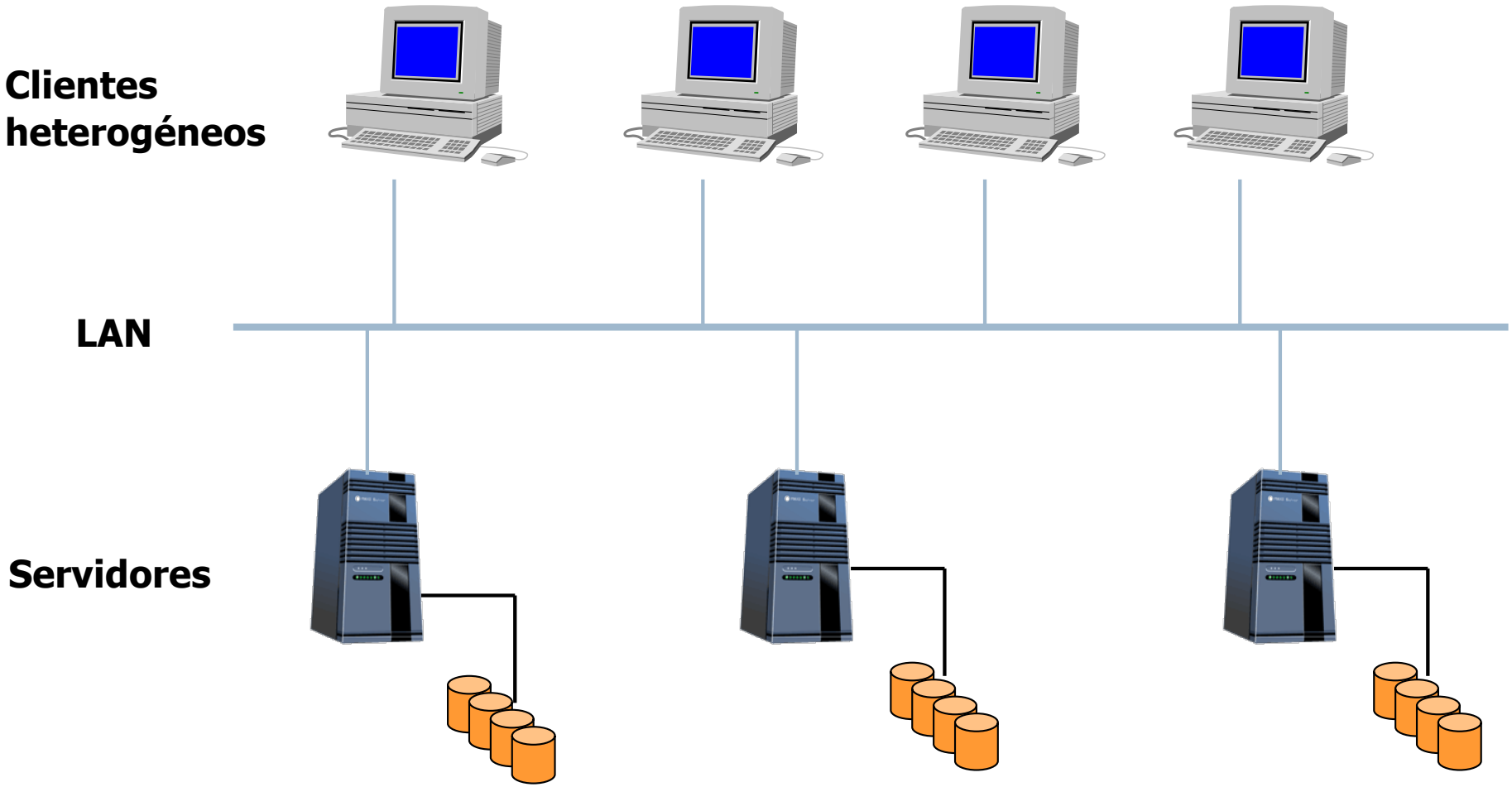
Fuente: Hadoop.apache.org

Sistemas de almacenamiento en red

- Infraestructura tradicional
  - Discos conectados a los servidores (DAS, *Direct Attach Storage*)
- Redes de almacenamiento
  - NAS (*Network Attached Storage*)
  - SAN (*Storage Area Networks*)
- Sistemas de ficheros para discos compartidos

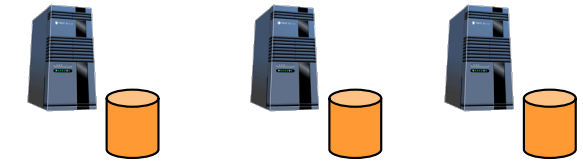
Infraestructura tradicional

DAS (*Direct Attach Storage*) Conexión directa al servidor

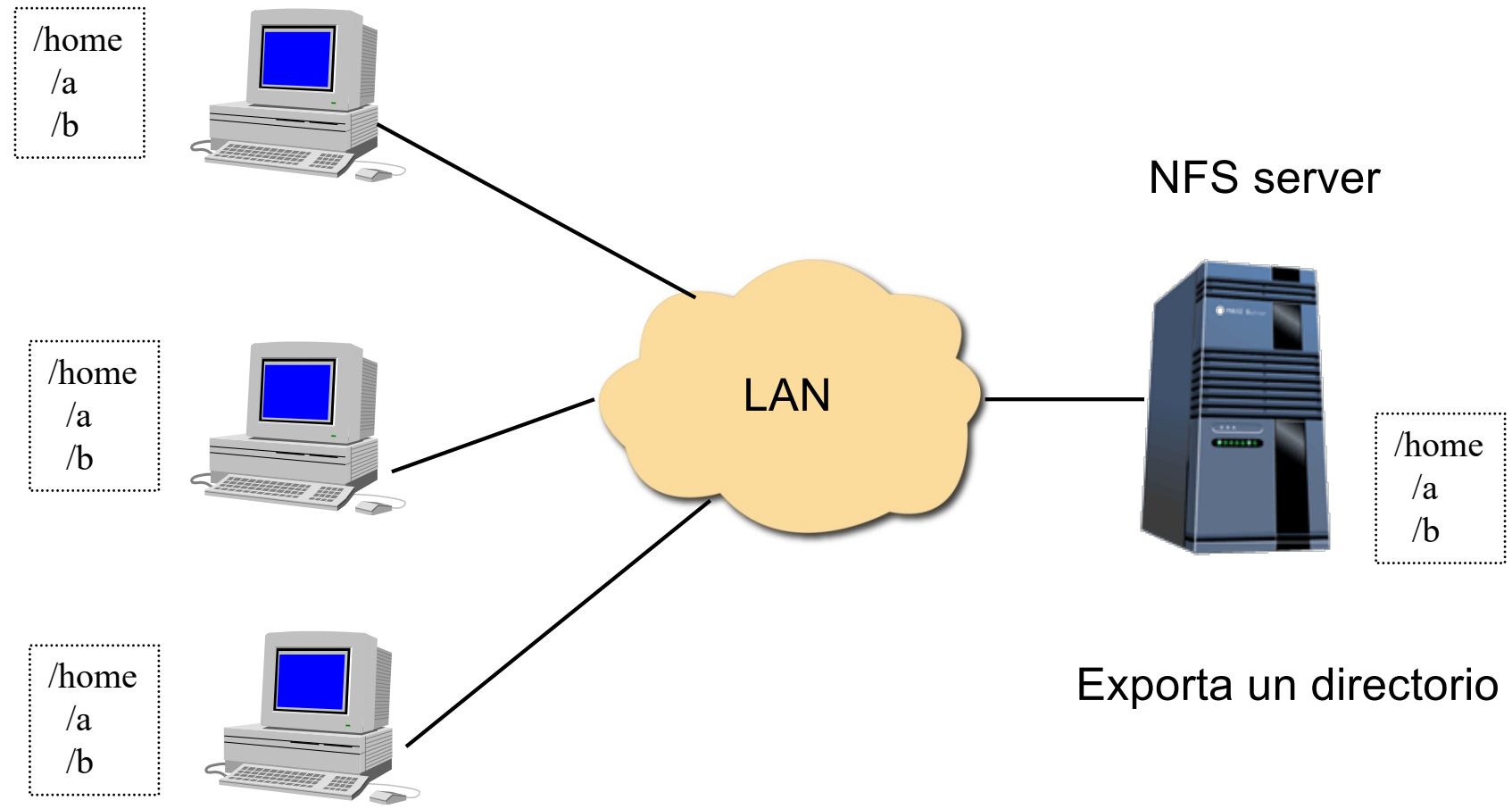


Conexión directa al servidor

- Ventajas:
  - Servicios independientes
- Inconvenientes:
  - Administración y mantenimiento
  - Saturación de servidores y LAN en los Backup
  - Escalabilidad
  - Coste
  - Rendimiento
  - Dificultades para un correcto aprovisionamiento de almacenamiento
  - Mala utilización de los recursos

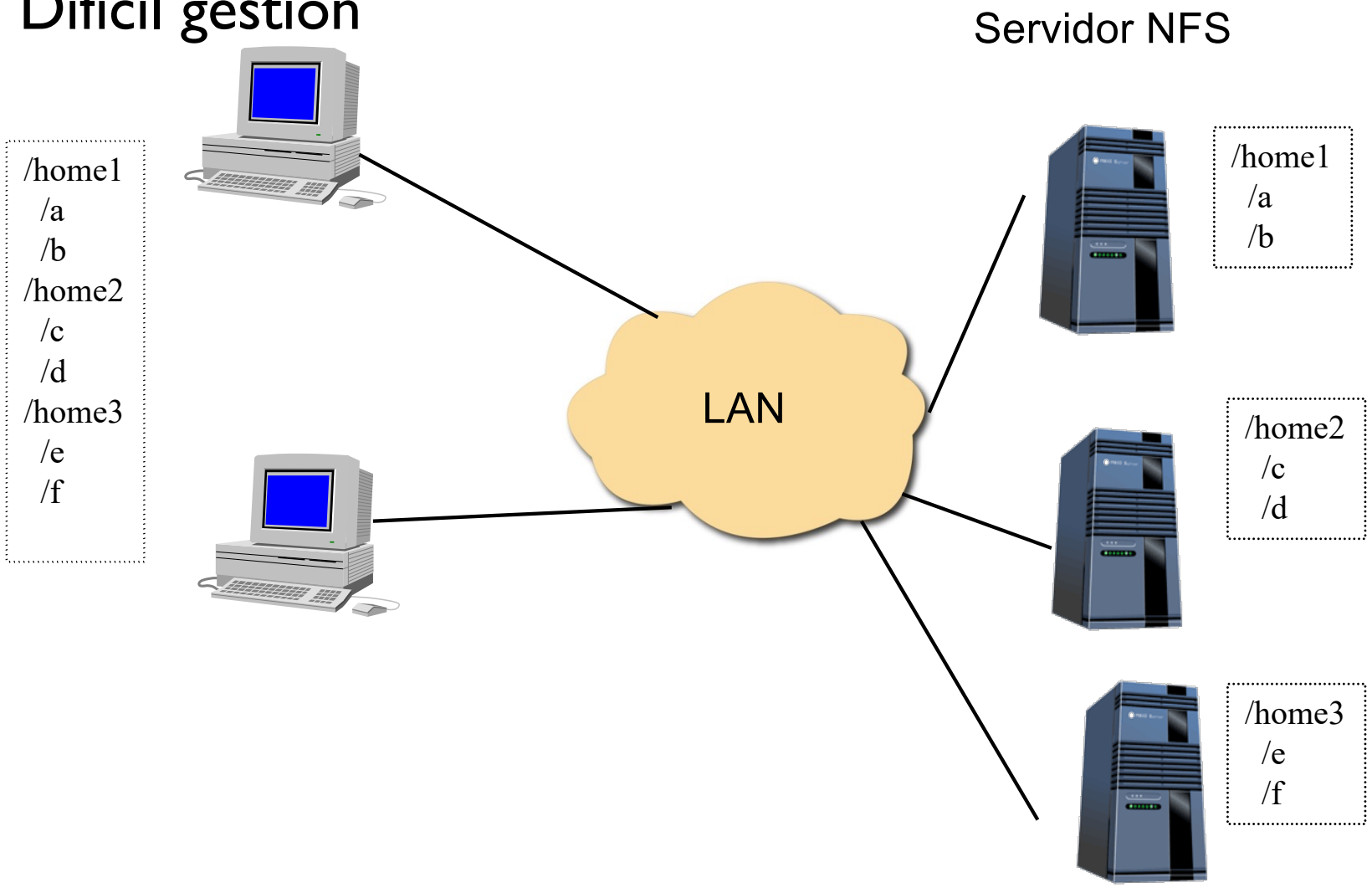


Ejemplo: NFS

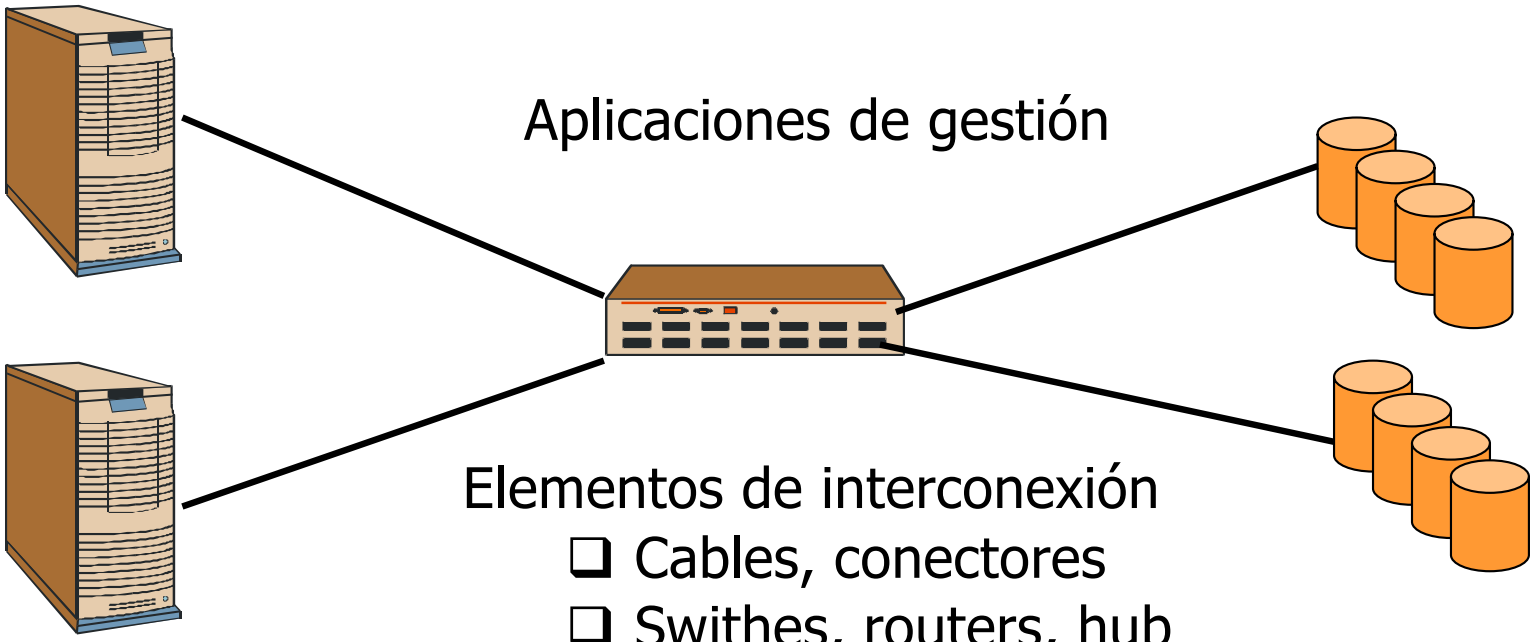

/home
/a
/b
/home
/a
/b
/home
/a
/b
LAN
NFS server
/home
/a
/b
Exporta un directorio
Montan el directorio remoto

Ejemplo de particionamiento

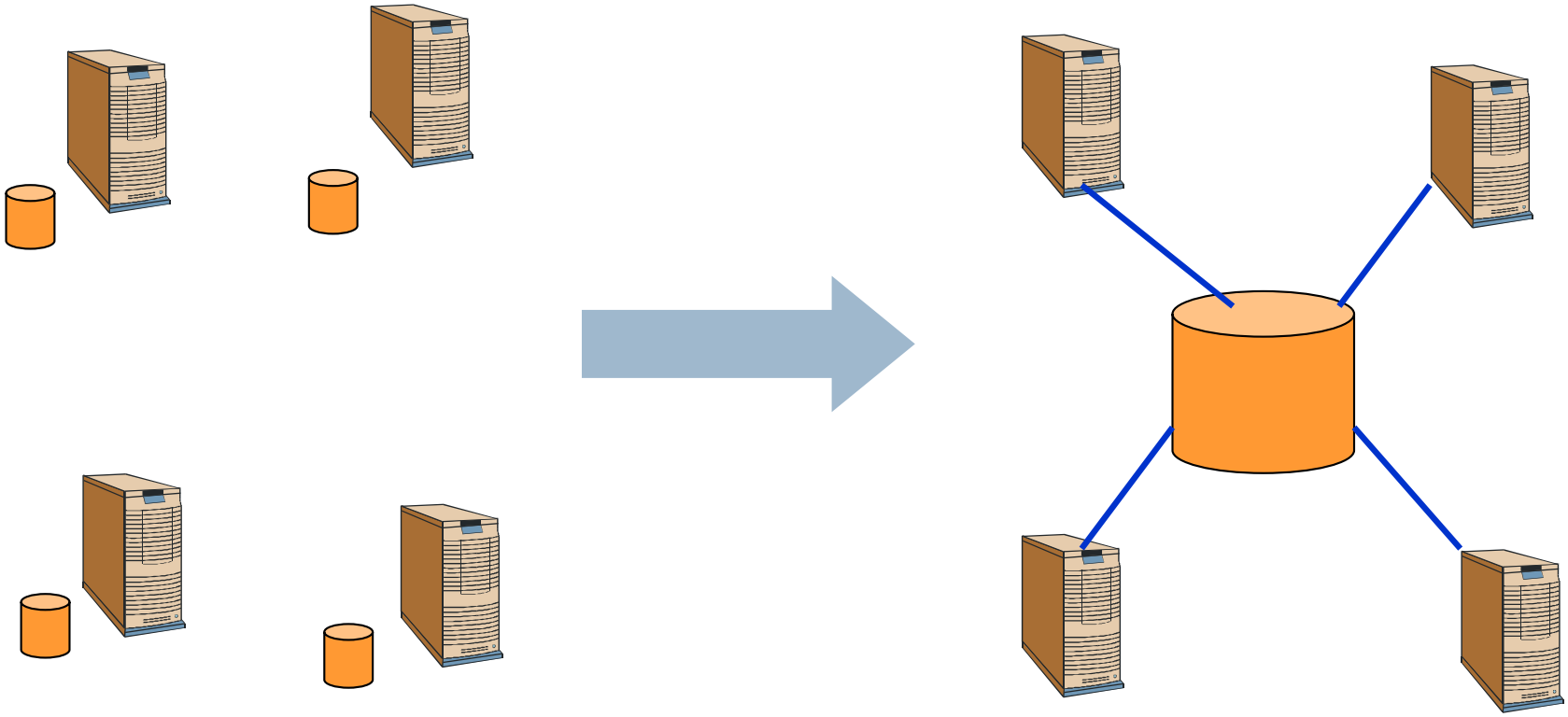
- Difícil gestión



Almacenamiento en red

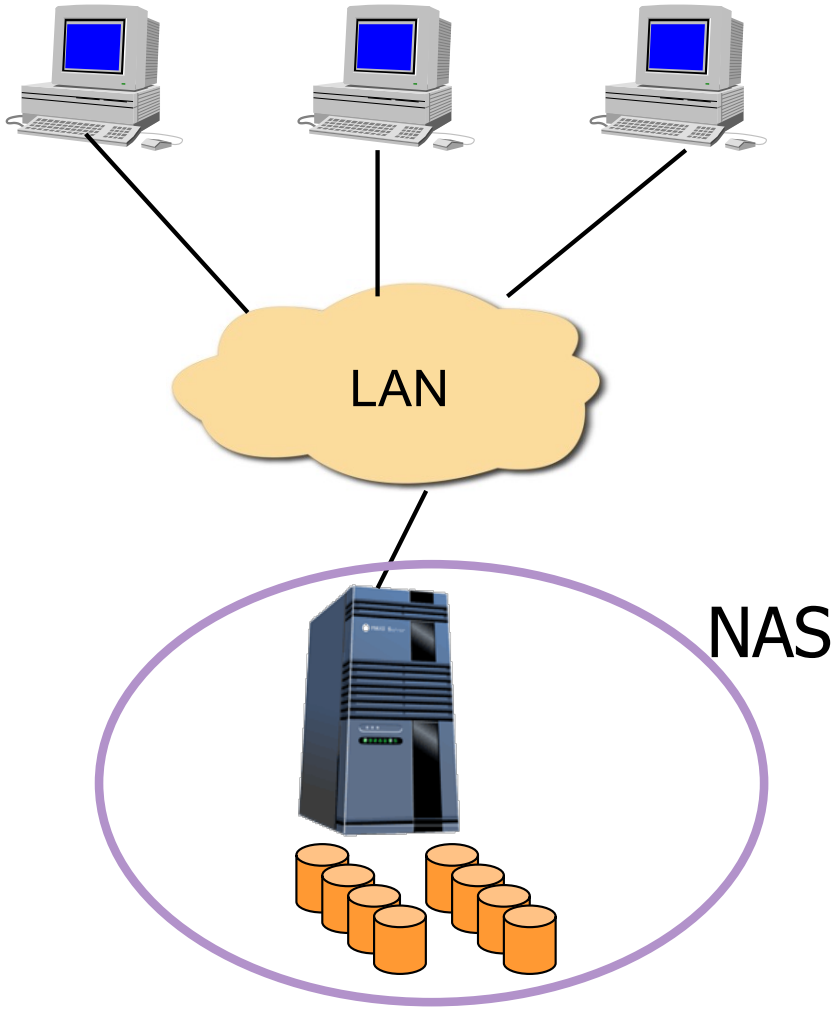


Consolidación del almacenamiento

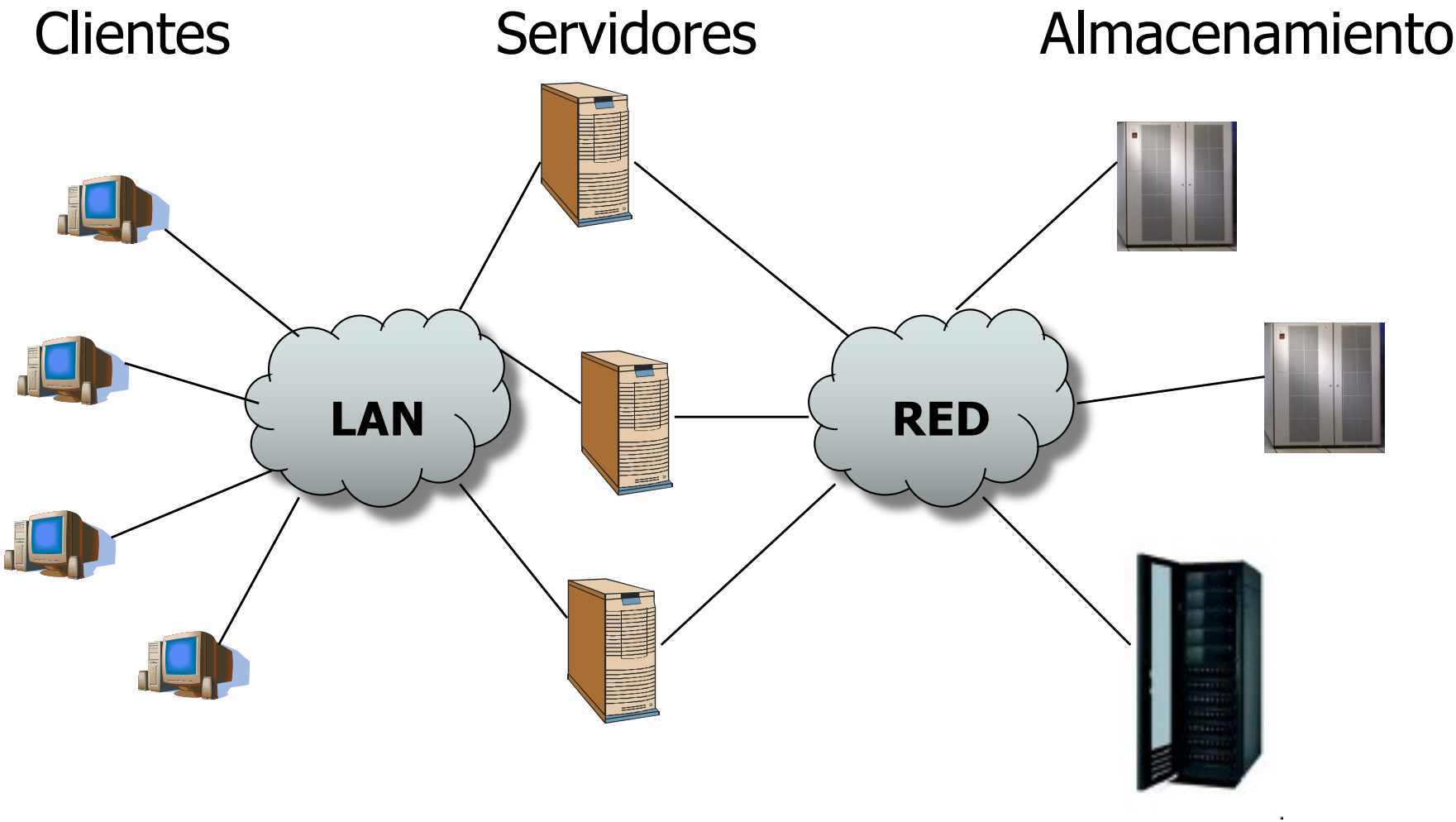


NAS (*Network Attached Storage*)

- Dispositivo de almacenamiento con conexión directa a la red
  - Almacenamiento
  - Servidor
- Protocolos estándar
  - NFS
  - CIFS (common Internet file system)
  - HTTP

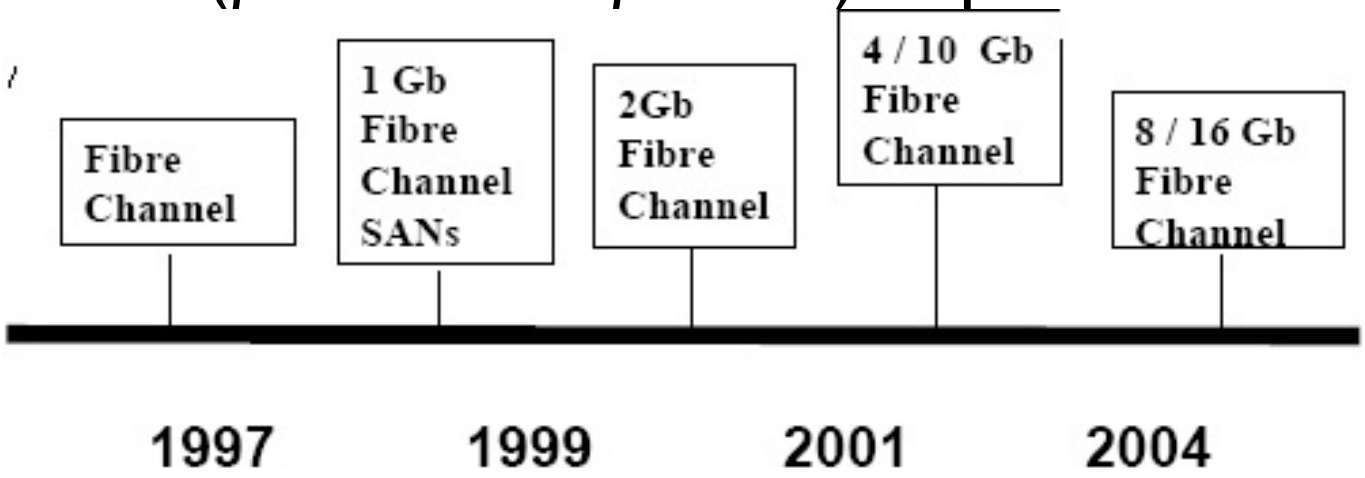


SAN (*Storage Area Networks*)

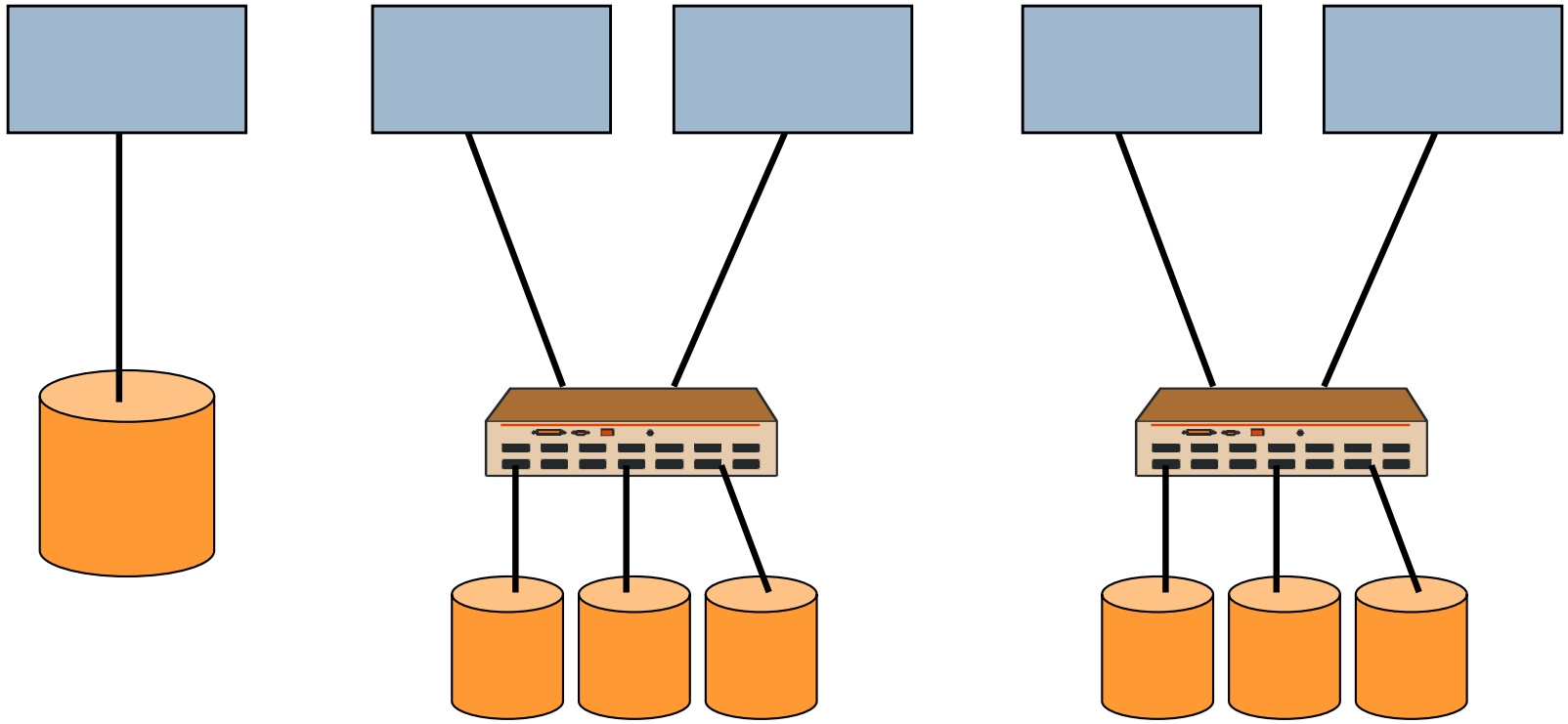


Fibre Channel

- Mecanismo de transporte genérico
- Puede utilizar cables de fibra óptica y cobre
- Interfaz para transferencia serie de datos a alta velocidad y baja latencia (GB/s) a largas distancias (hasta 10 Km)
  - 100/200/400/800/1200 MB/s
- Utiliza FCP (*fibre channel protocol*): implementación serie de



Diferentes topologías



Punto a Punto hub switch

FC tiene un espacio de direcciones de 24 bits

iSCSI

- Acceso a dispositivos de almacenamiento a través de redes TCP/IP.
- Tramas SCSI sobre paquetes IP.
- El protocolo TCP/IP realiza:
  - Control de congestión
  - Entrega ordenada de paquetes
  - Corrección de errores
- iSCSI habilita acceso universal

- Dos tipos de tramas:
  - Comandos SCSI
  - Bloques de datos

Ventajas

- Alta disponibilidad
- Reducción de los costes de alta disponibilidad
- Backup sin intervención de los servidores