

OpenCourseWare  
**Sistemas Paralelos y Distribuidos**

Félix García Carballeira

Alejandro Calderón Matos

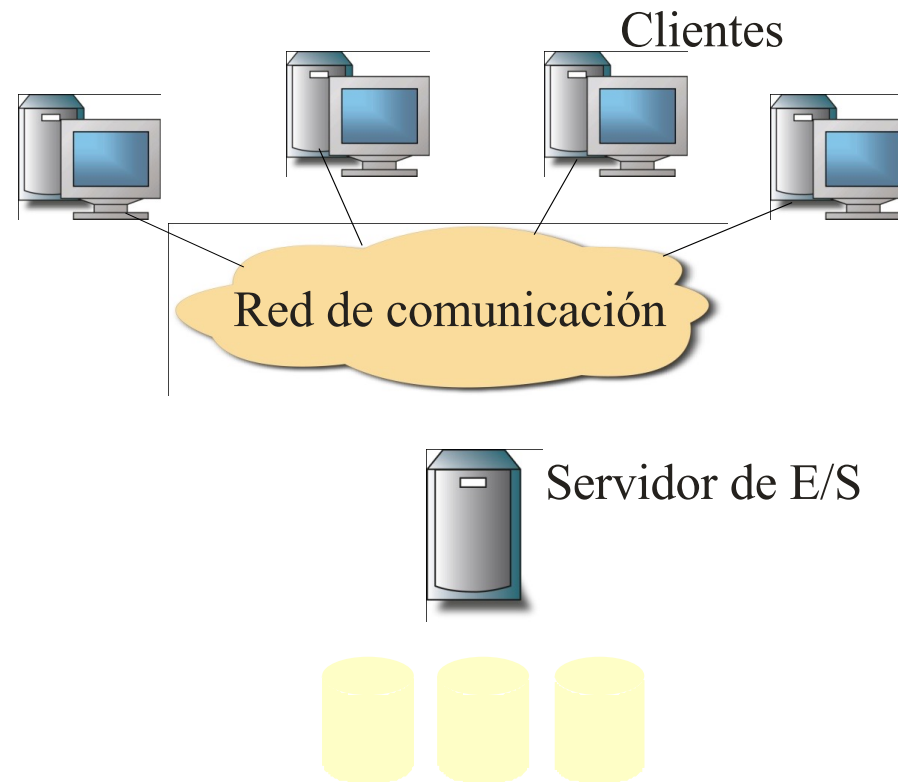
**Tema 1. Sistemas de ficheros distribuidos y paralelos**



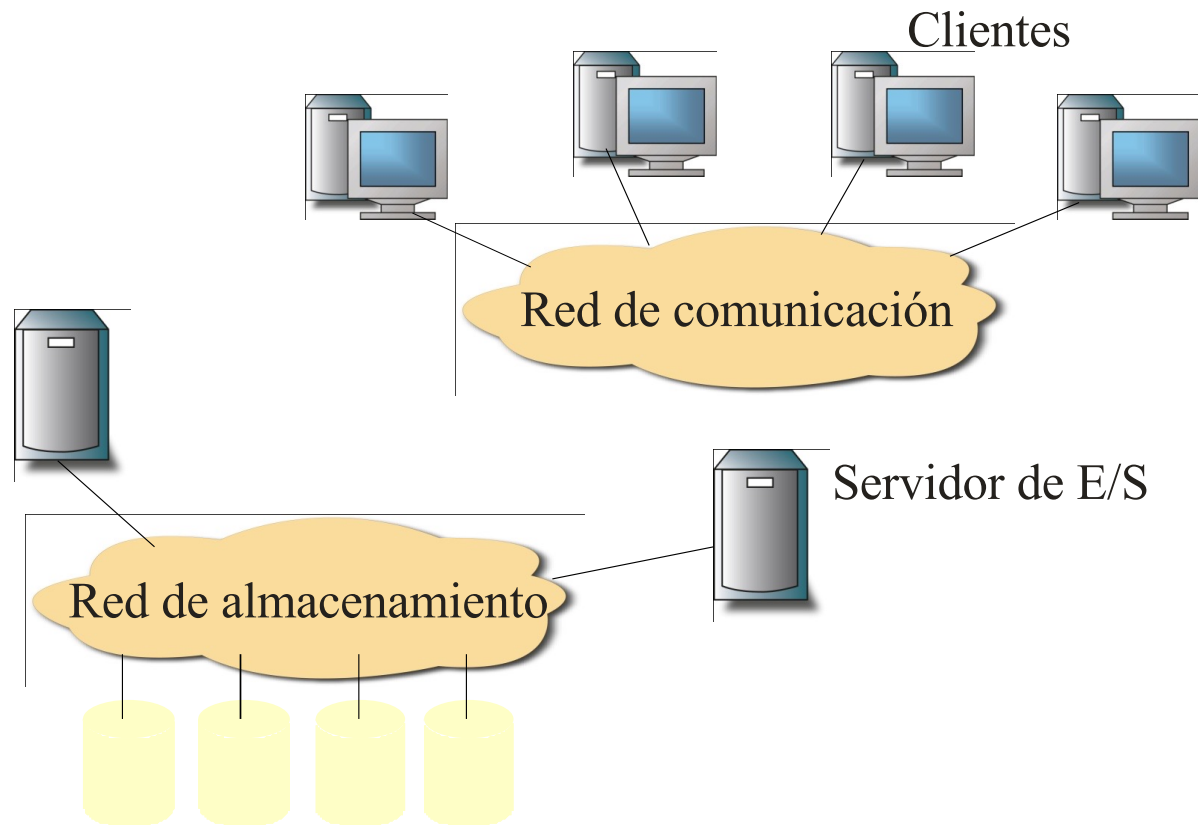
# Gestión de datos en entornos distribuidos

- Arquitecturas de almacenamiento
- Sistemas de ficheros en red y distribuidos
- Sistemas de ficheros de discos compartidos
- Sistemas de ficheros paralelos
- Sistemas de ficheros paralelos ad-hoc
- Sistema de ficheros distribuidos para Big Data

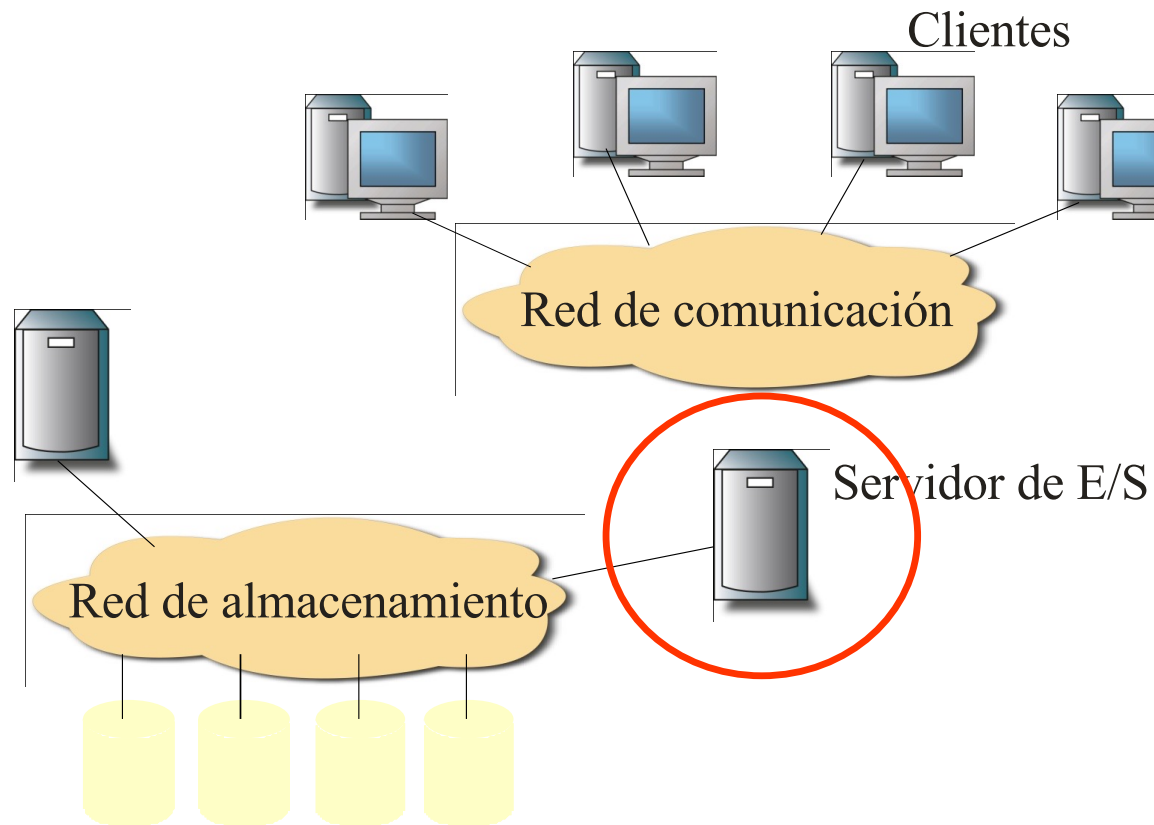
# Arquitectura tradicional de un sistema de almacenamiento



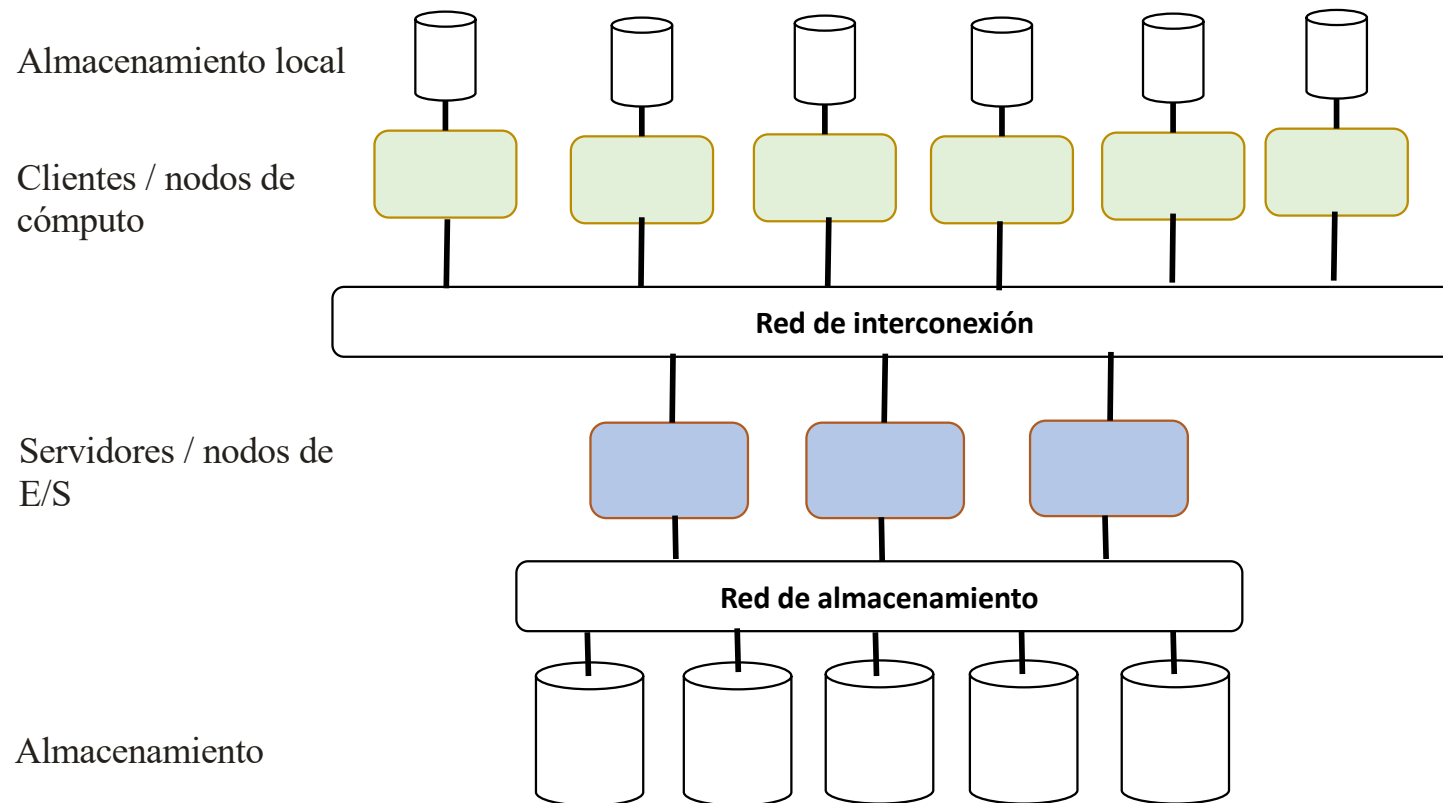
# Arquitectura tradicional de un sistema de almacenamiento



# Arquitectura tradicional de un sistema de almacenamiento

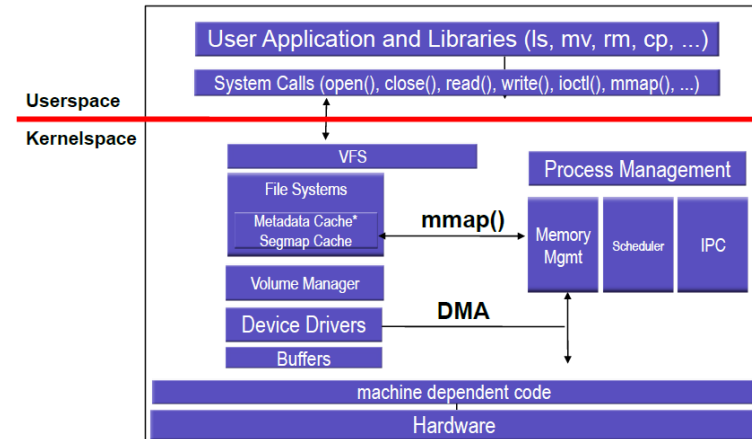


# Arquitectura de almacenamiento de un cluster



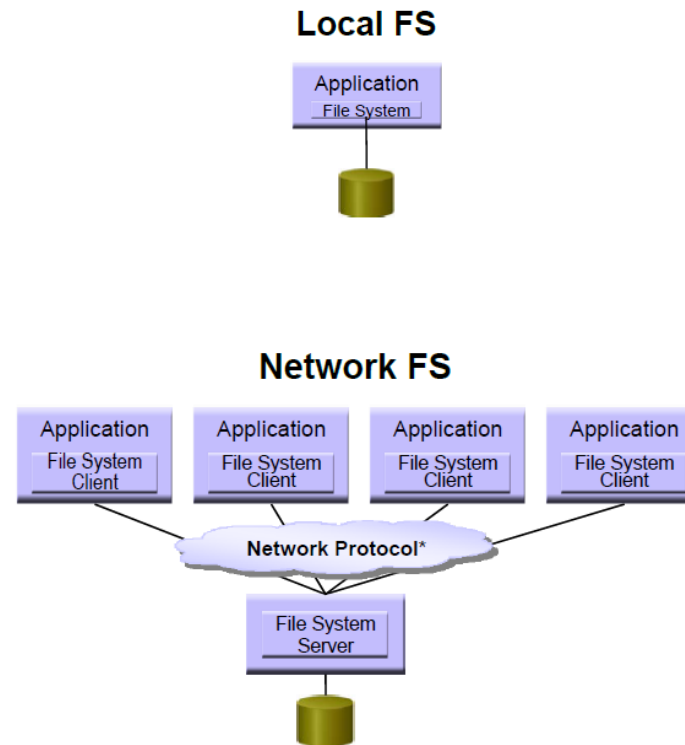
# Sistema de ficheros

- Un sistema de ficheros establece una correspondencia lógica entre los ficheros y directorios y los dispositivos
- **Funciones:**
  - Organización, almacenamiento, recuperación, gestión de nombres, coutilización y protección de los ficheros (**metadatos**)
  - 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.

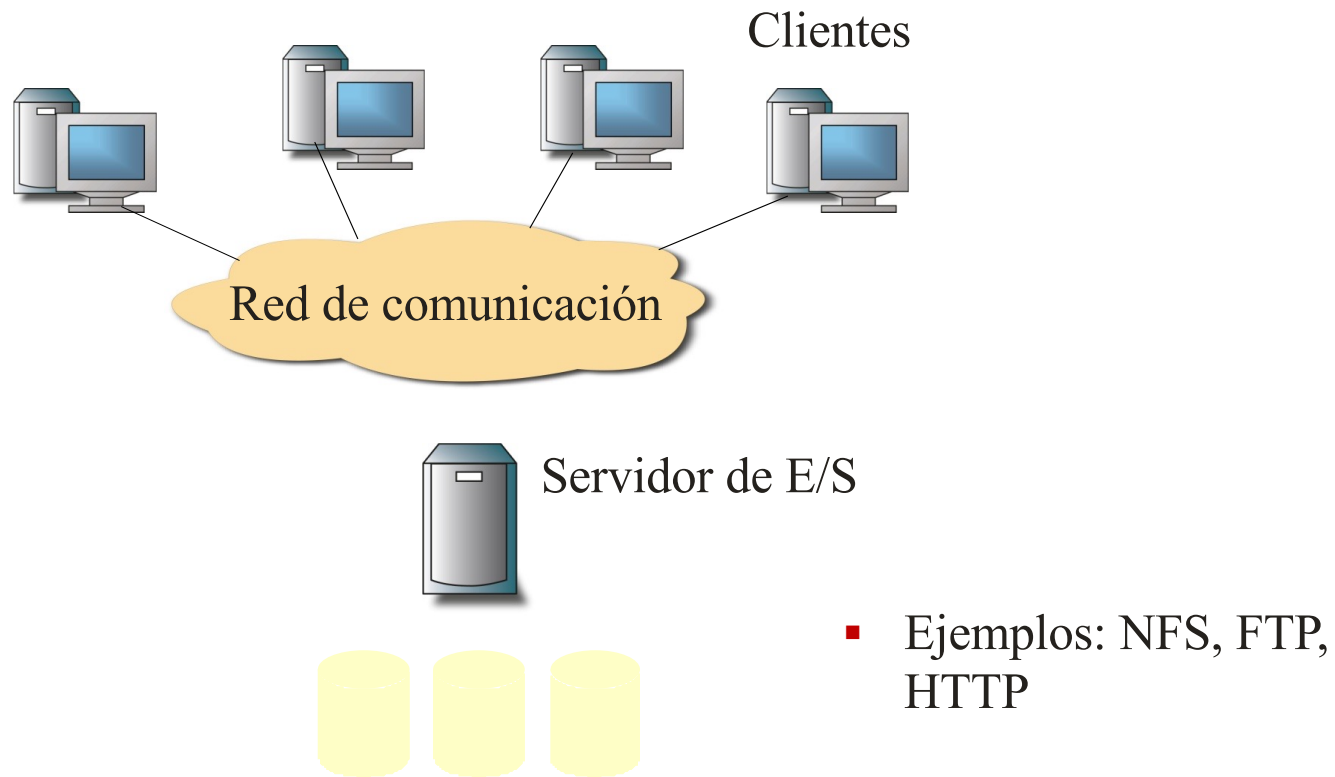


# Sistema de fichero en red

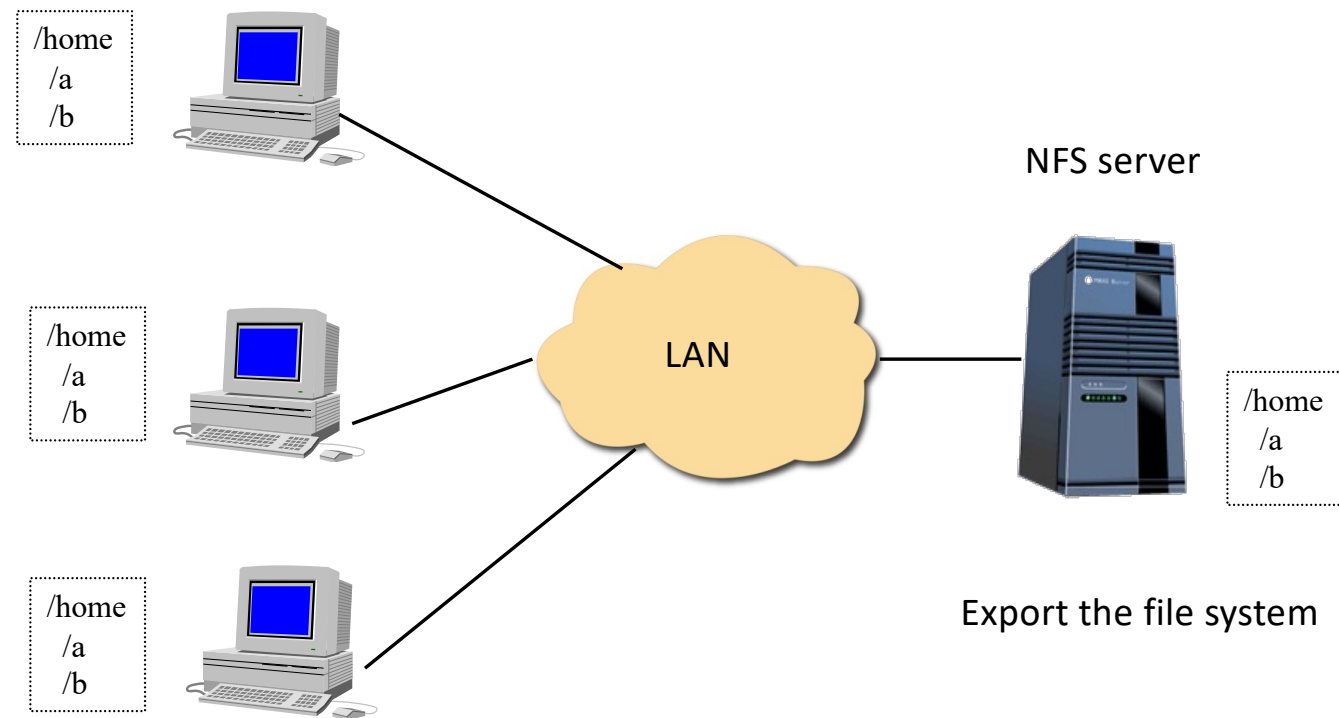
- Sistema de ficheros que permite el acceso a los ficheros de un **servidor** utilizando un determinado **protocolo**
  - Extensión de un sistema de ficheros local



# Sistema de ficheros en red

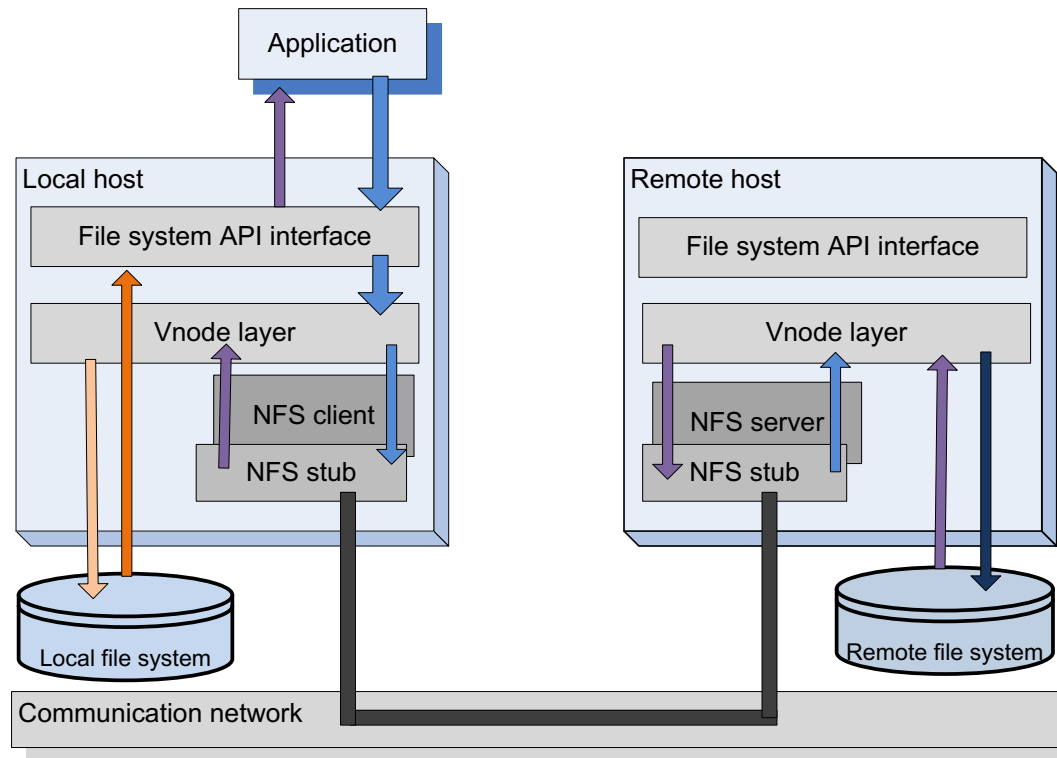


# Ejemplo: NFS



Clients mount the remote file system

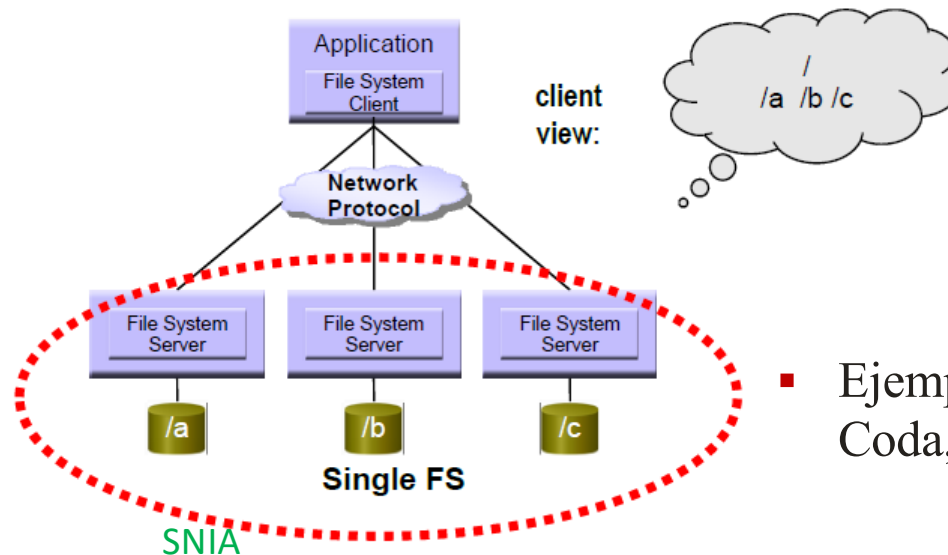
# Arquitectura de NFS



© Cloud Computing: Theory and Practice  
Dan C. Marinescu

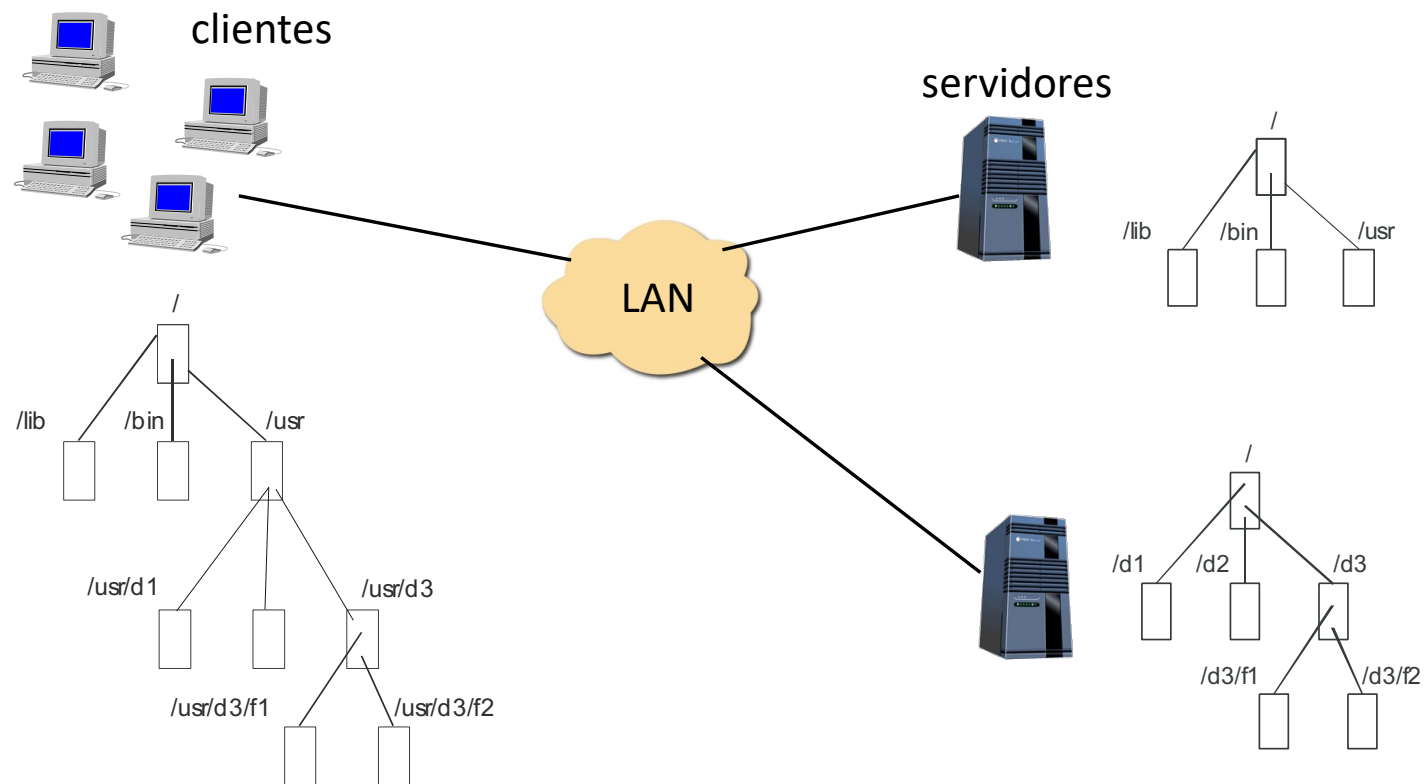
# Sistemas de ficheros distribuidos

- Sistema de ficheros en red que permite el acceso a ficheros distribuidos por varios servidores (no es un sistema de ficheros paralelo)

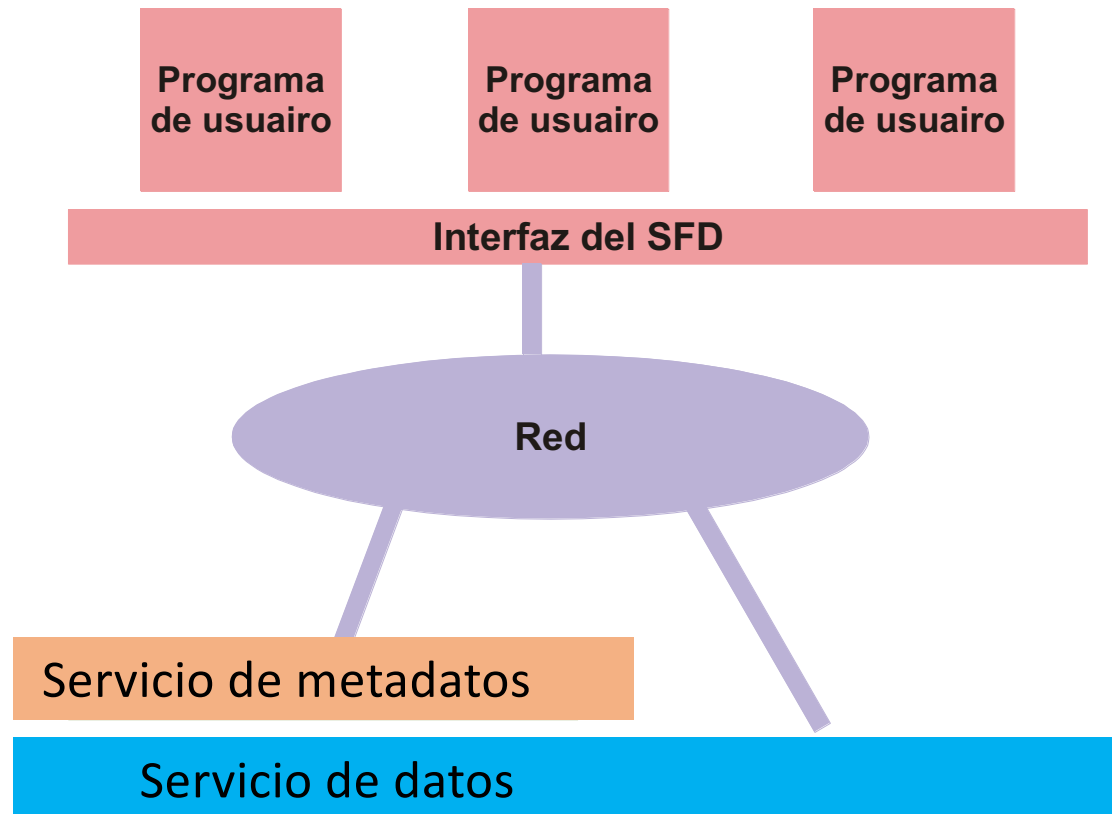


- Ejemplos: AFS, DFS, Coda, Sprite

# Sistemas de ficheros distribuidos



# Componentes de un SFD



# Métodos de acceso remotos

- **Modelo carga/descarga**
  - Transferencias completas del fichero
  - 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 cache en el cliente**
  - Combina los dos modelos anteriores.

# Caché de bloques

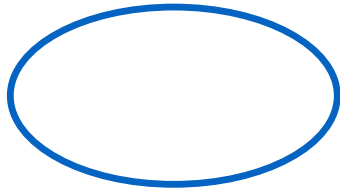
- El empleo de caché de bloques permite mejorar el rendimiento
  - Explota el principio de proximidad de referencias
    - Proximidad temporal
    - Proximidad espacial
  - Lecturas adelantadas
    - Mejora el rendimiento de las operaciones de lectura, sobre todo si son secuenciales
  - Escrituras diferidas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

# Localización de la caché en un SFD

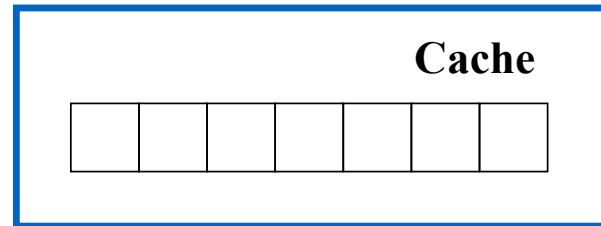
- ***Caché en los servidores***
  - Reducen los accesos a disco
- ***Caché en los clientes***
  - Reducen el tráfico por la red
  - Reducen la carga en los servidores
  - Mejora la capacidad de crecimiento
  - Dos posibles localizaciones
    - En discos locales
      - Más capacidad,
      - Más lento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

# Funcionamiento de una cache de bloques

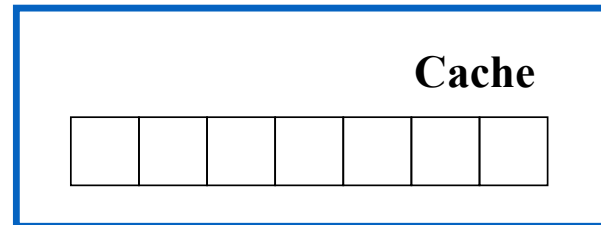
**Proceso de usuario**



**Cliente**



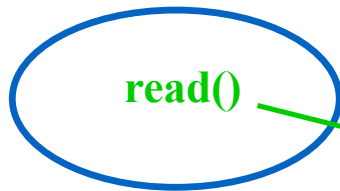
**Servidor**



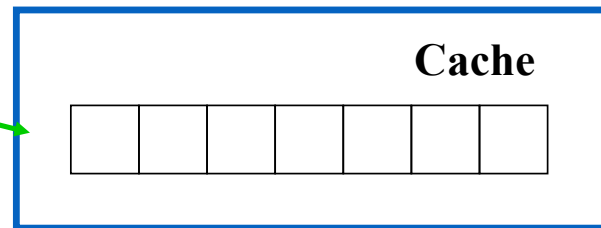
**Disco**

# Funcionamiento de una cache de bloques

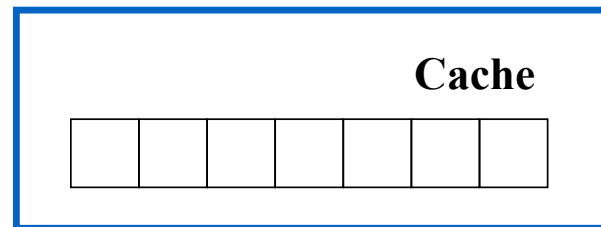
Proceso de usuario



Cliente



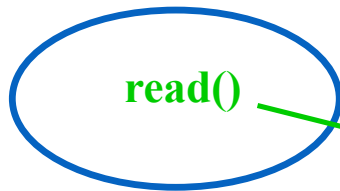
Servidor



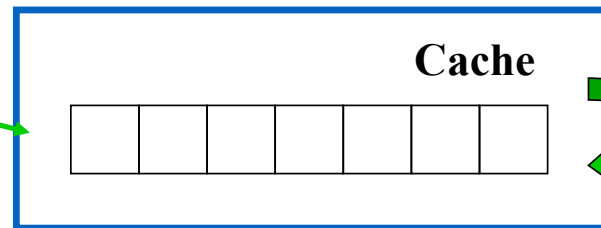
Disco

# Funcionamiento de una cache de bloques

Proceso de usuario

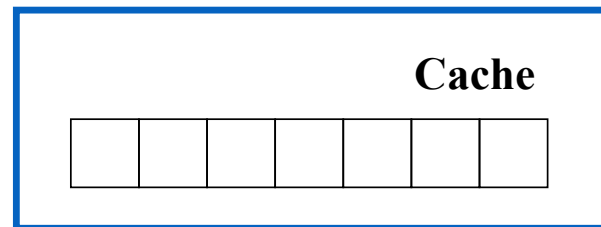


Cliente



Buscar bloque.  
Si no está,  
reservar uno

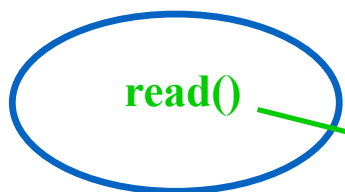
Servidor



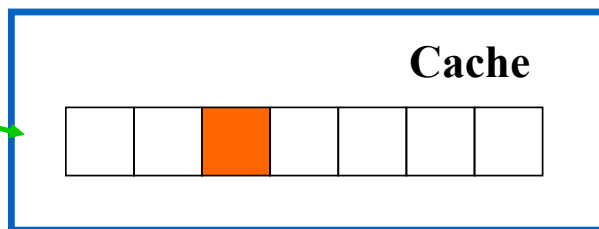
Disco

# Funcionamiento de una cache de bloques

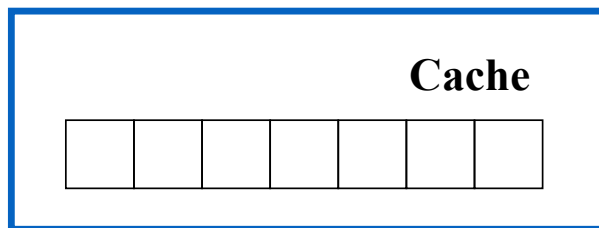
Proceso de usuario



Cliente



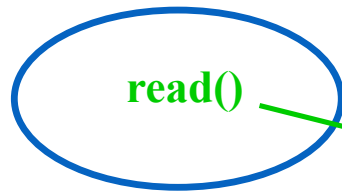
Servidor



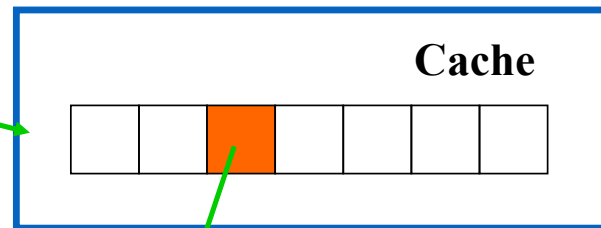
Disco

# Funcionamiento de una cache de bloques

Proceso de usuario

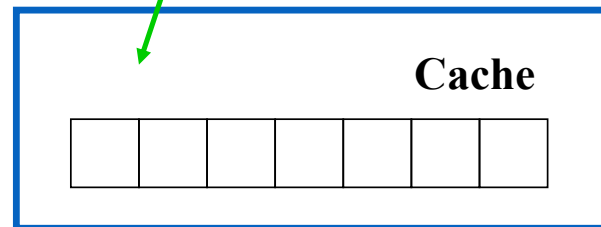


Cliente



`read()`

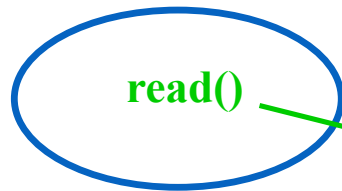
Servidor



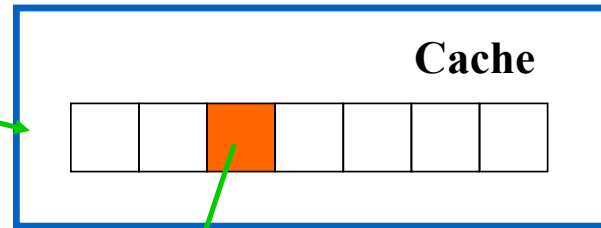
Disco

# Funcionamiento de una cache de bloques

Proceso de usuario

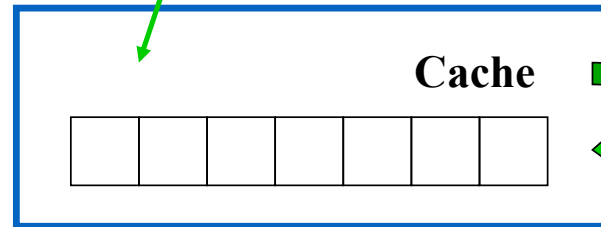


Cliente



`read()`

Servidor



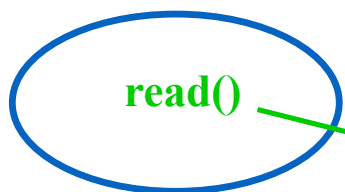
**Buscar bloque.  
Si no está,  
reservar uno**



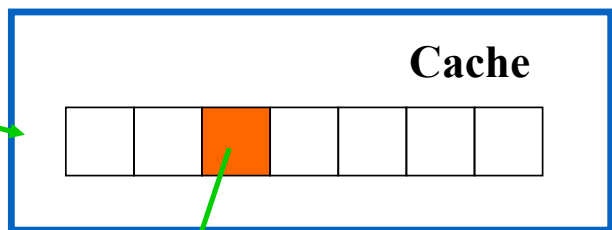
Disco

# Funcionamiento de una cache de bloques

Proceso de usuario

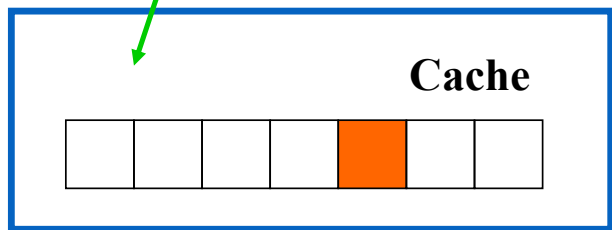


Cliente



`read()`

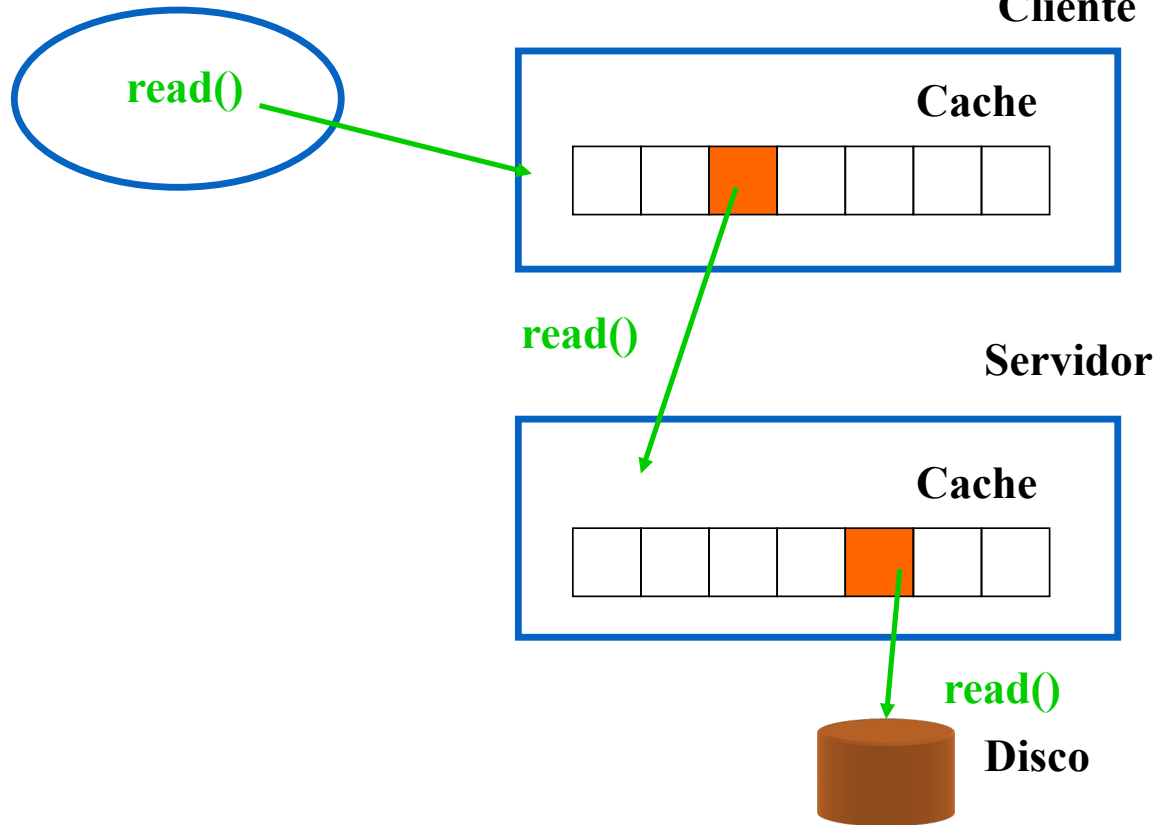
Servidor



Disco

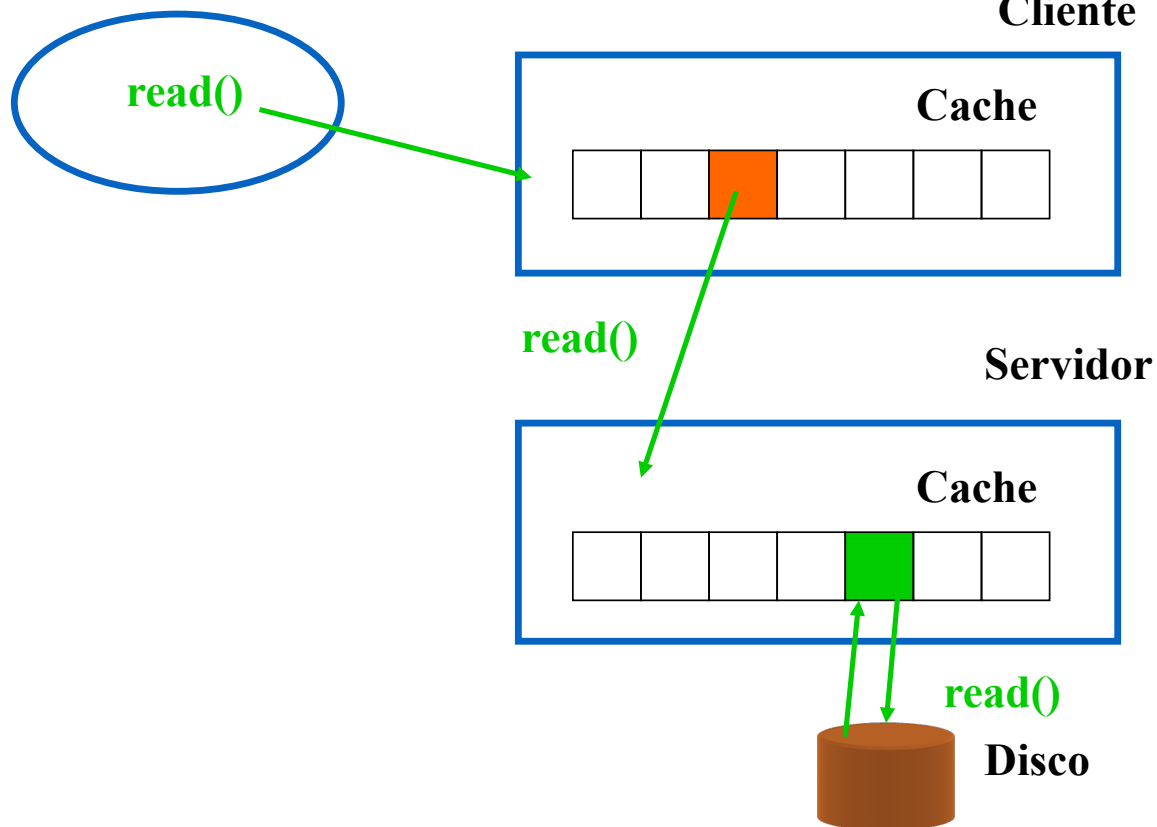
# Funcionamiento de una cache de bloques

Proceso de usuario



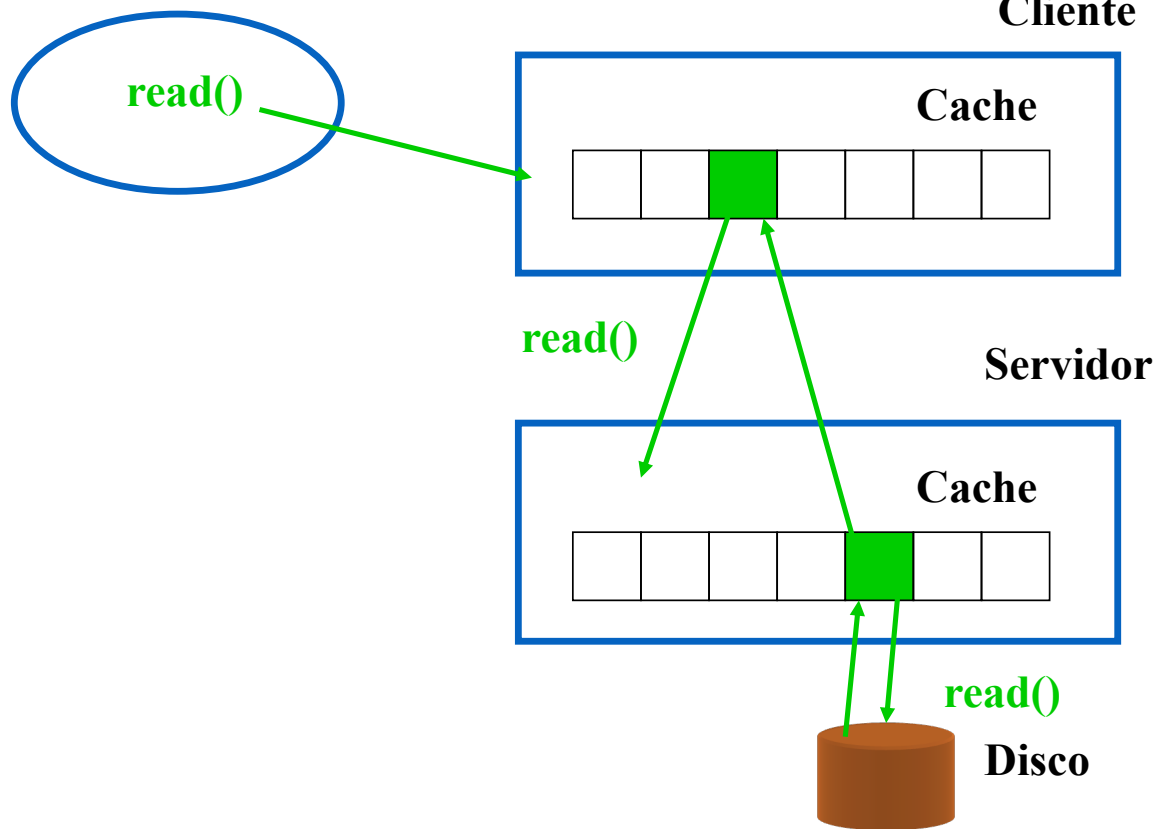
# Funcionamiento de una cache de bloques

Proceso de usuario

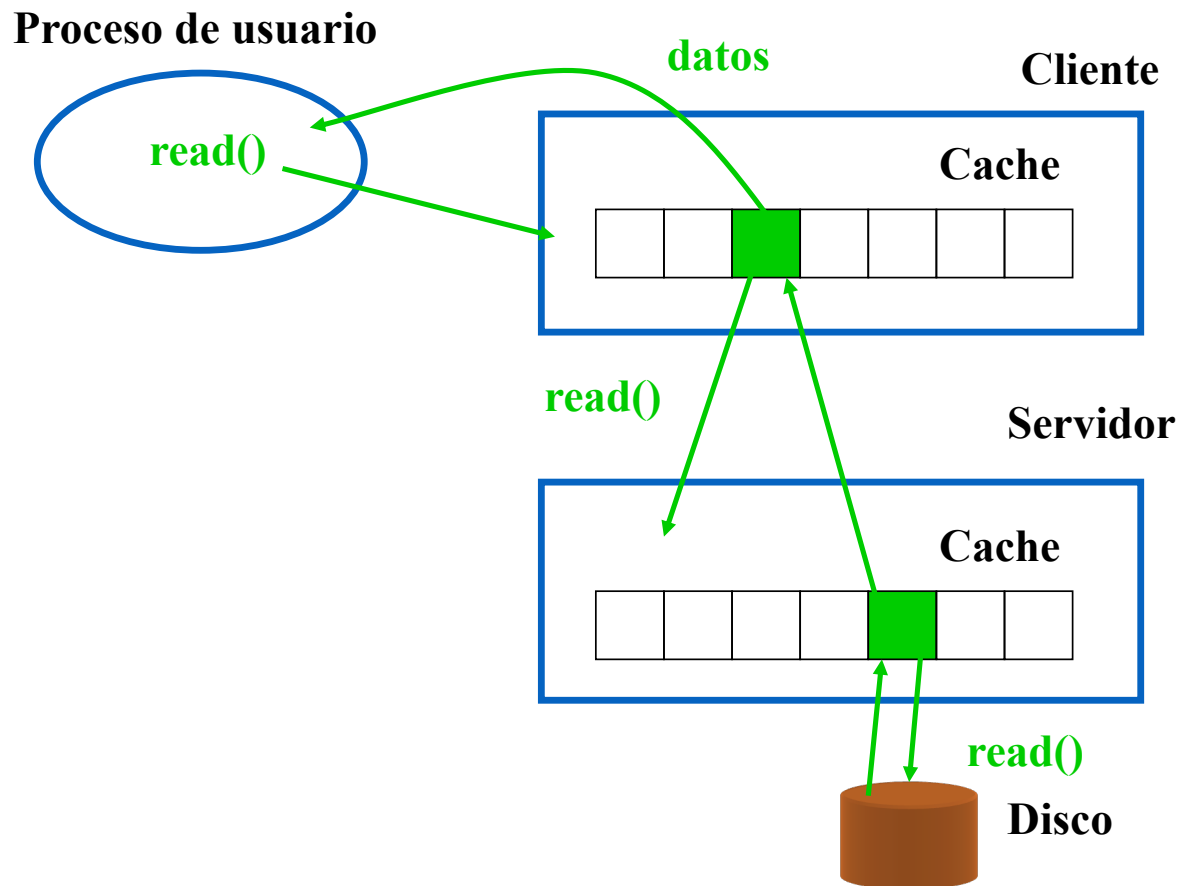


# Funcionamiento de una cache de bloques

Proceso de usuario

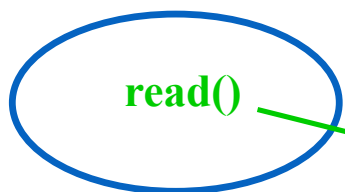


# Funcionamiento de una cache de bloques

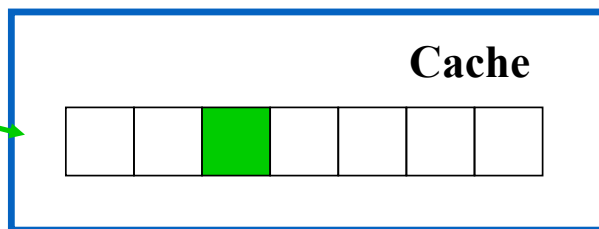


# Funcionamiento de una cache de bloques

Proceso de usuario

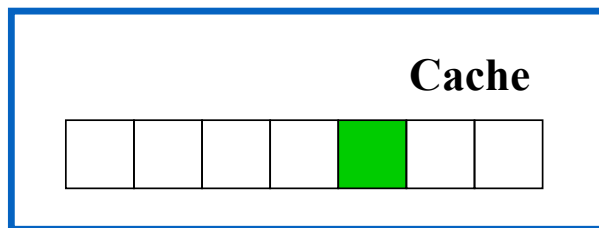


Cliente



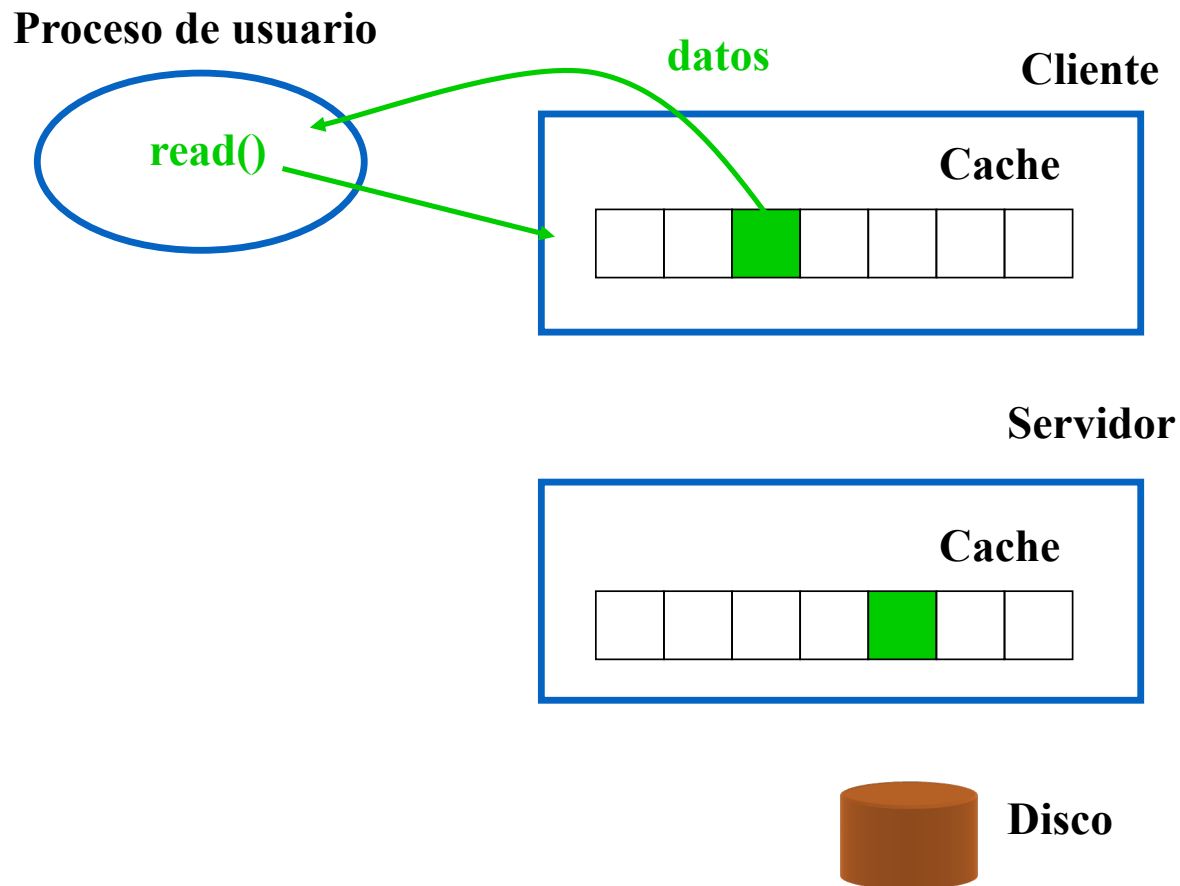
Futuro acceso

Servidor



Disco

# Funcionamiento de una cache de bloques

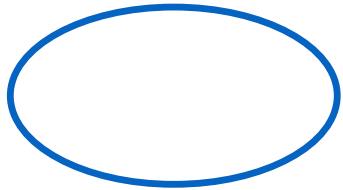


# Políticas de actualización

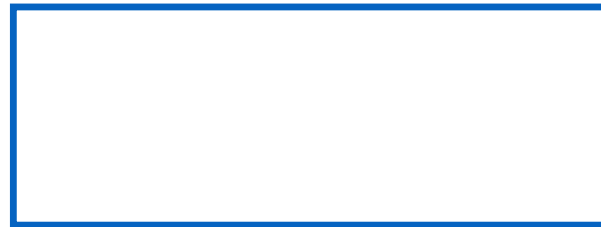
- Escritura inmediata (*write-through*)
  - Buena fiabilidad
  - En escrituras se obtiene el mismo rendimiento que en el modelo de accesos remotos
  - Las escrituras son más lentas
- Escritura diferida (*write-back*)
  - Escrituras más rápidas. Se reduce el tráfico en la red
  - Los datos pueden borrarse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

# Motivación de *write-before-full*

**Proceso de usuario**



**Cliente**



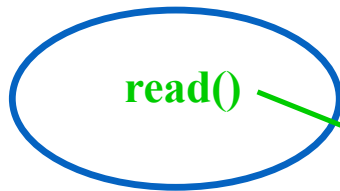
**Servidor**



**Disco**

# Motivación de write-before-full

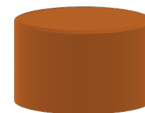
Proceso de usuario



Cliente



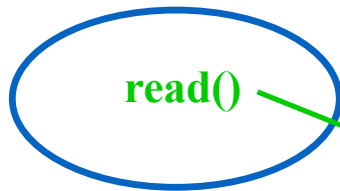
Servidor



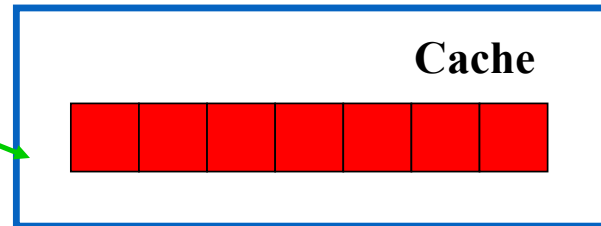
Disco

## Motivación de write-before-full

**Proceso de usuario**



**Cliente**



**Servidor**

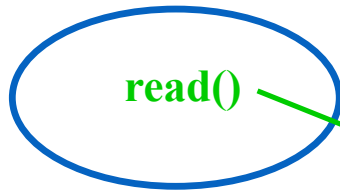


 **Bloque sucio**

 **Disco**

## Motivación de write-before-full

**Proceso de usuario**



**Cliente**



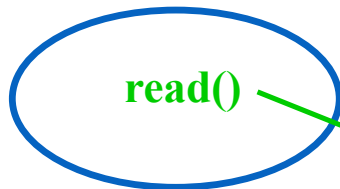
**Algoritmo de reemplazo**

**Servidor**



## Motivación de write-before-full

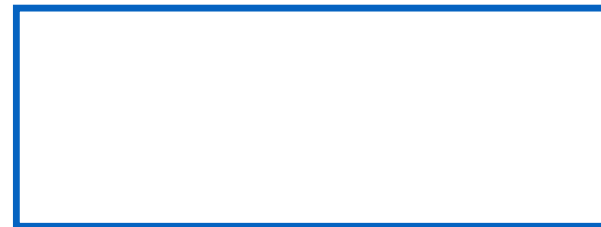
Proceso de usuario



Cliente



Servidor

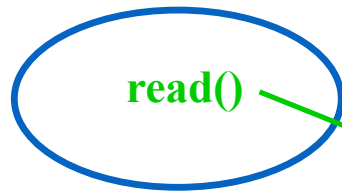


 **Bloque a expulsar**

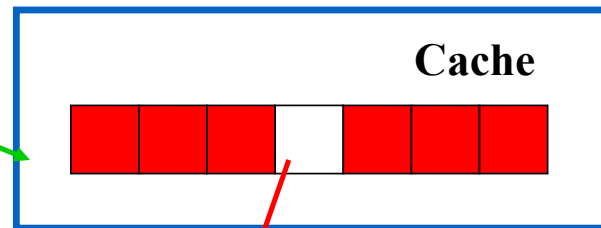
 **Disco**

## Motivación de write-before-full

Proceso de usuario

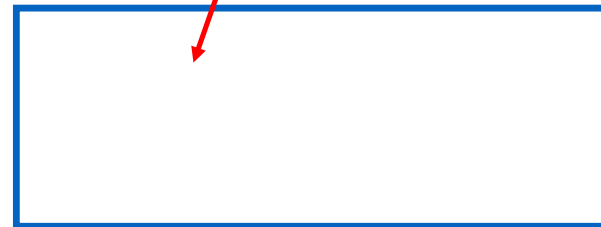


Cliente



`write()`

Servidor

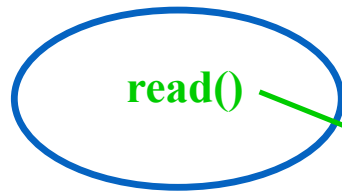


 **Bloque limpio**

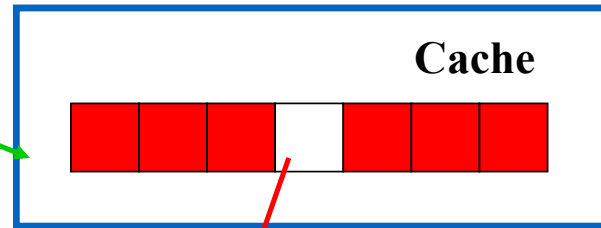
 **Disco**

## Motivación de write-before-full

Proceso de usuario

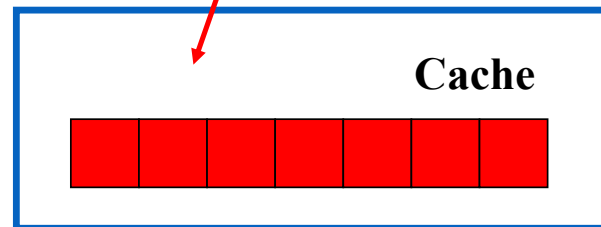


Cliente



`write()`

Servidor

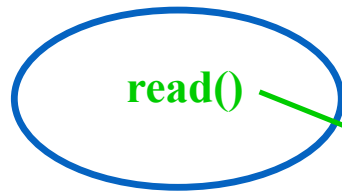


 **Bloque sucio**

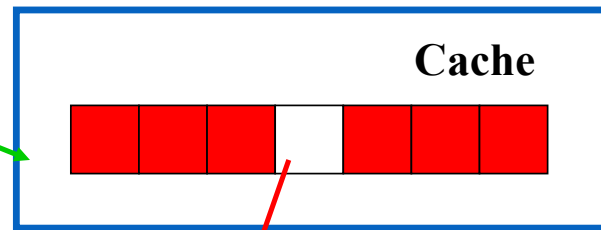
 **Disco**

## Motivación de write-before-full

Proceso de usuario

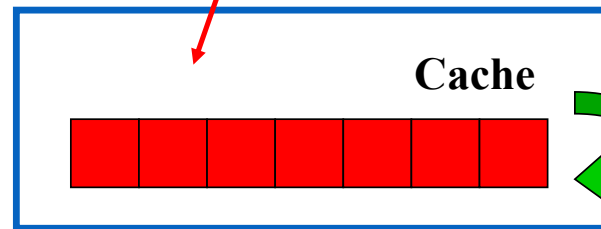


Cliente



`write()`

Servidor



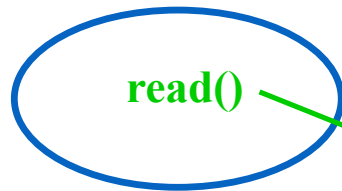
Algoritmo de reemplazo



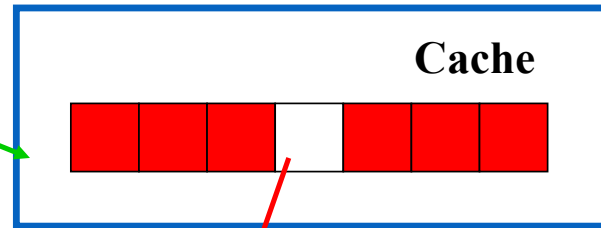
Disco

## Motivación de write-before-full

Proceso de usuario



Cliente



`write()`

Servidor

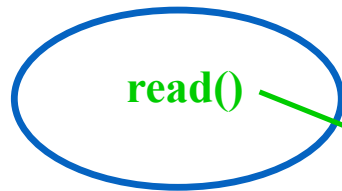


 **Bloque a expulsar**

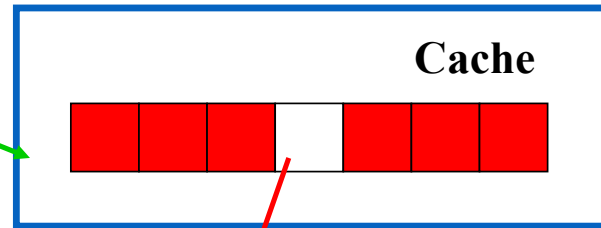
 **Disco**

## Motivación de write-before-full

Proceso de usuario

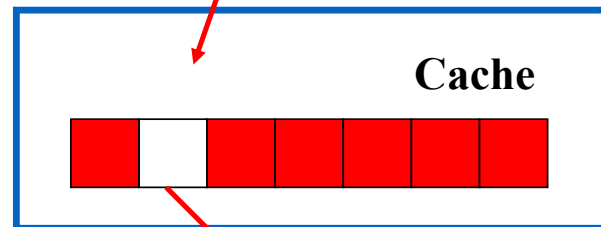


Cliente

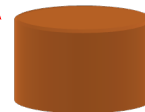


`write()`

Servidor



`write()`

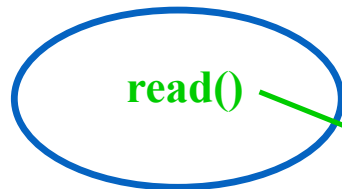


Disco

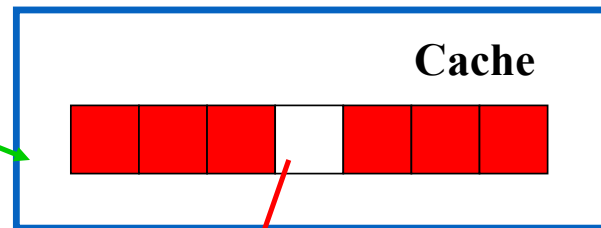
 **Bloque limpio**

## Motivación de write-before-full

Proceso de usuario

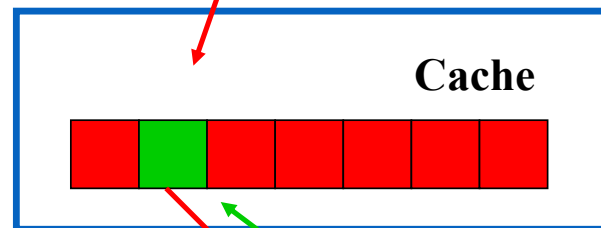


Cliente



`write()`

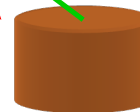
Servidor



`write()`

`read()`

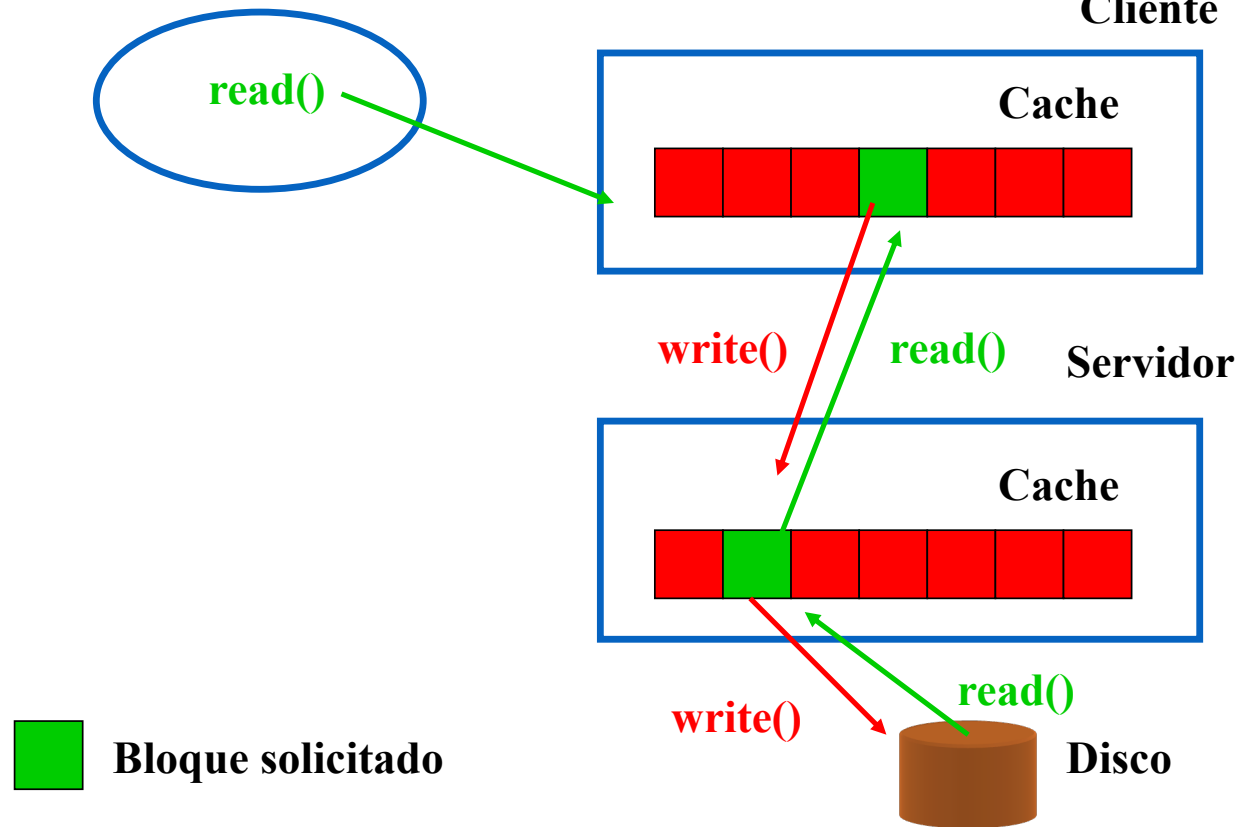
Disco



 **Bloque solicitado**

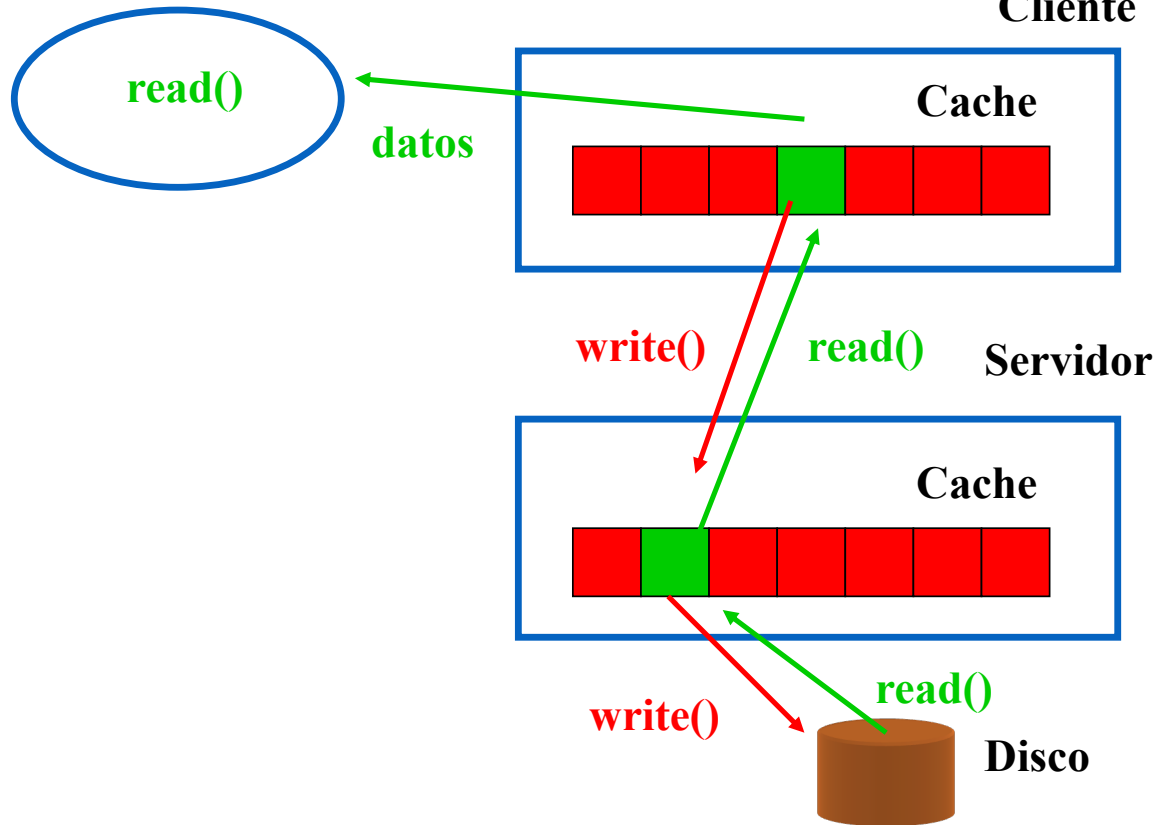
## Motivación de write-before-full

Proceso de usuario



## Motivación de write-before-full

Proceso de usuario



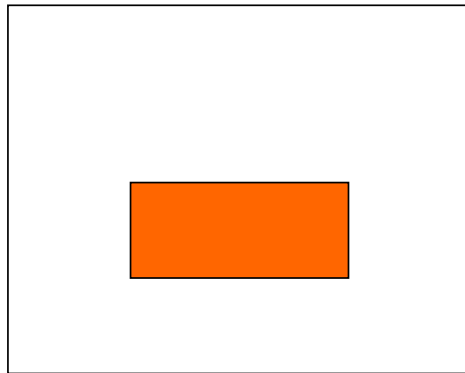
# Problema de coherencia

- El uso de caché en los clientes de un sistema de ficheros introduce el problema de la coherencia de cache:
  - 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.

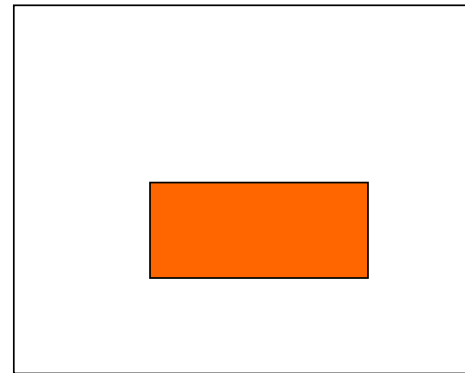
# Problema de la coherencia de cache

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

Cache en el cliente A



Cache en el cliente B

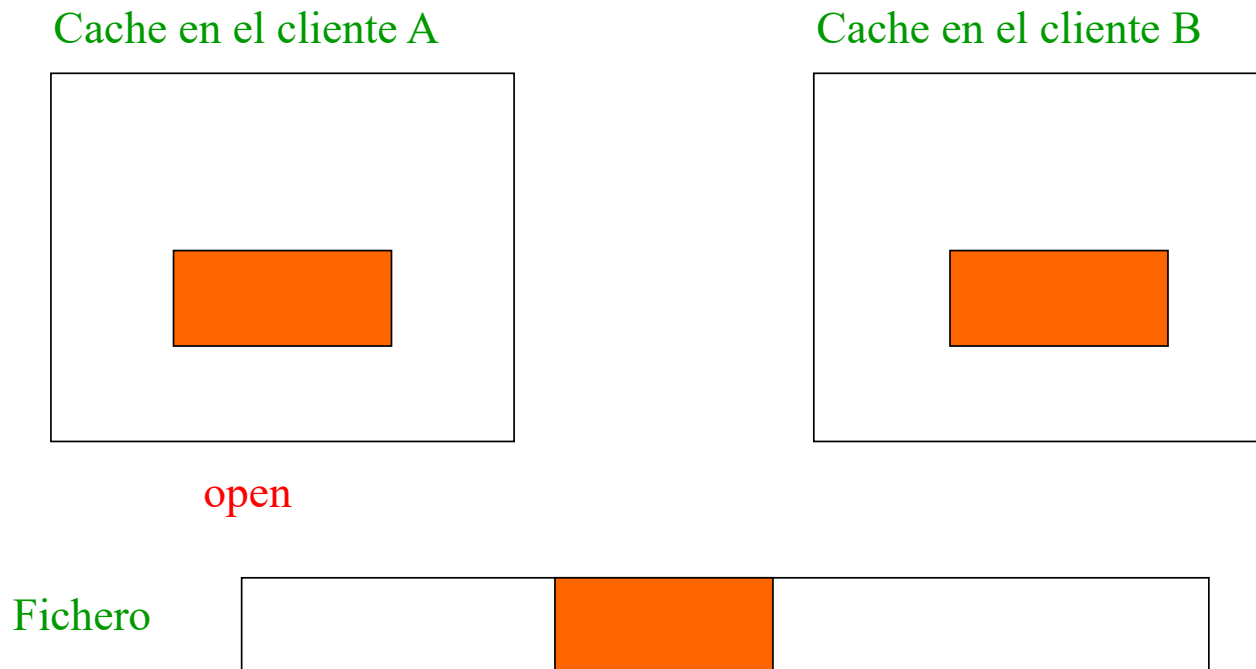


Fichero



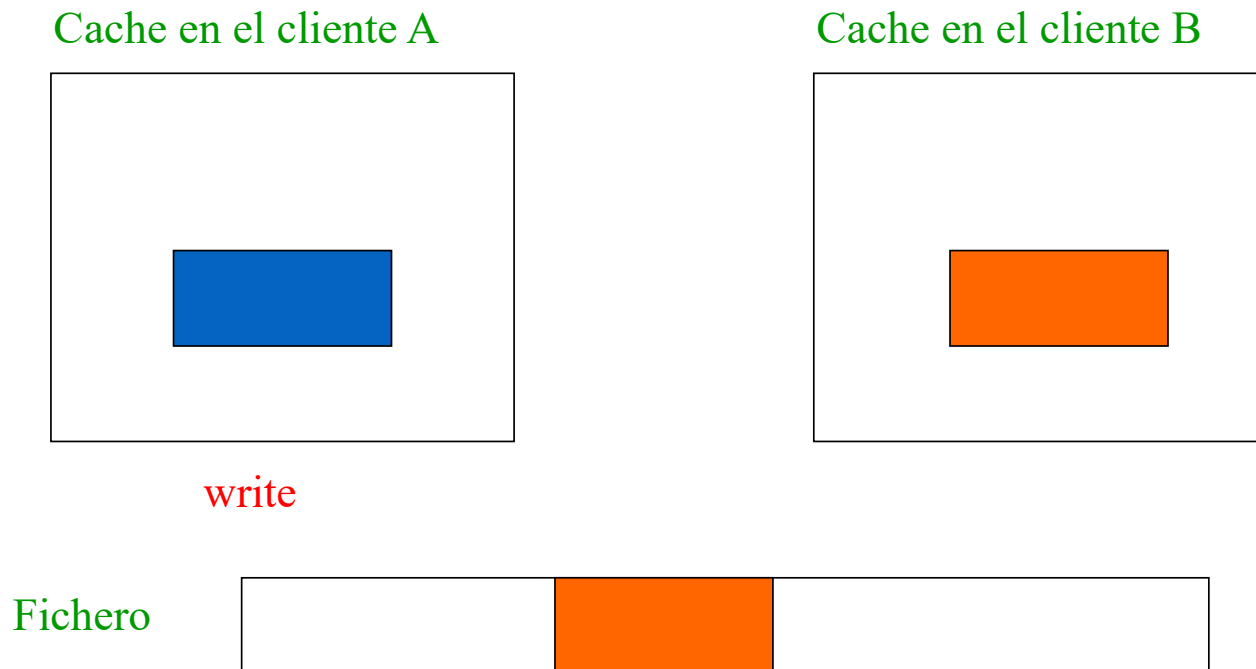
# Problema de la coherencia de cache

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



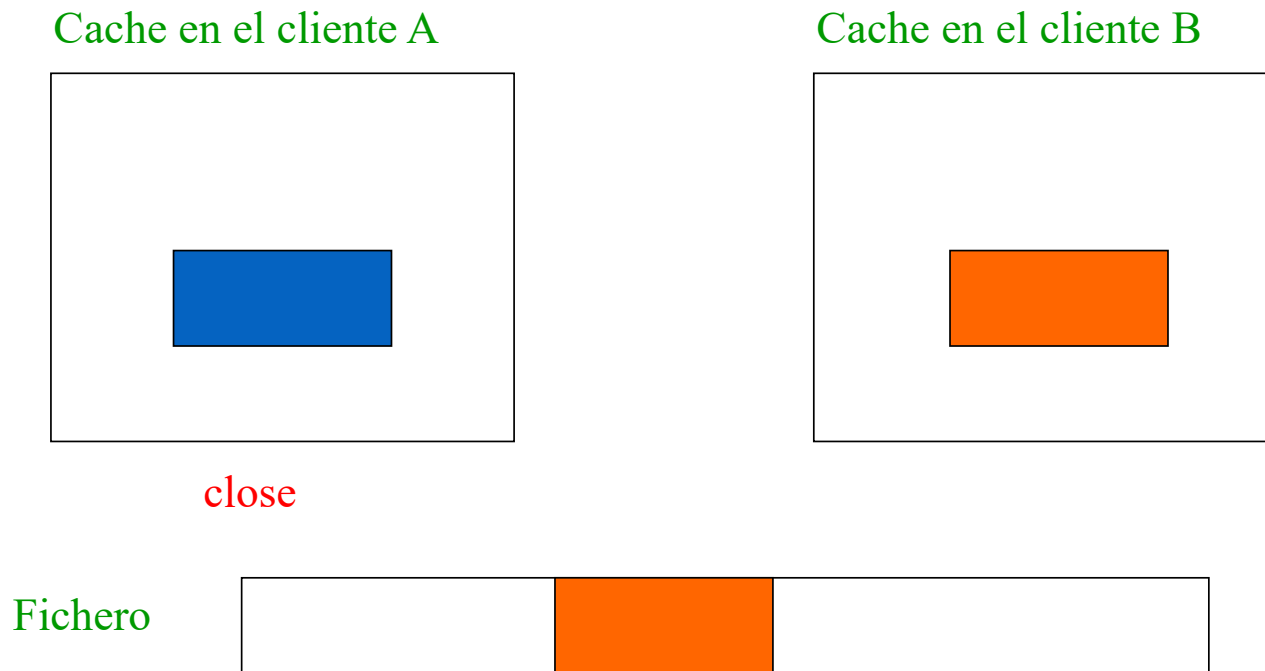
# Problema de la coherencia de cache

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



# Problema de la coherencia de cache

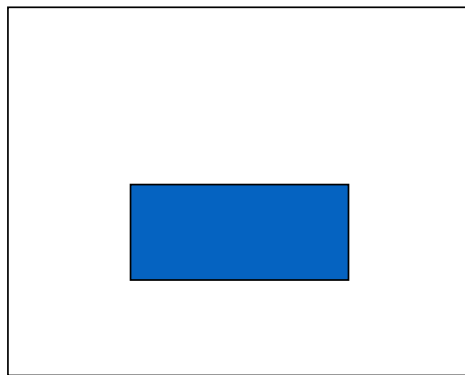
- Inconsistencia en escritura secuencial debido a la no actualización de las copias



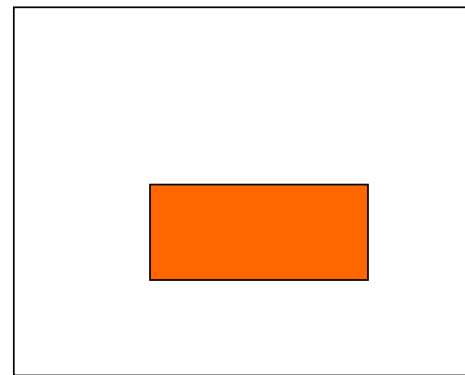
# Problema de la coherencia de cache

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

Cache en el cliente A



Cache en el cliente B

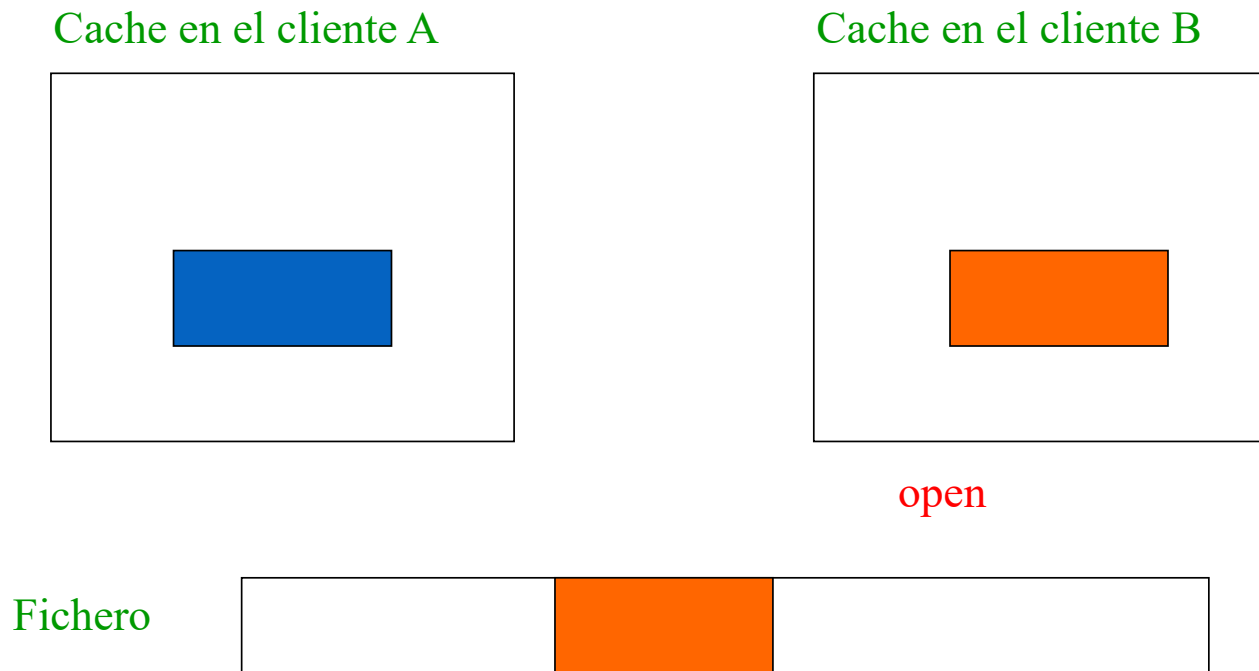


Fichero



# Problema de la coherencia de cache

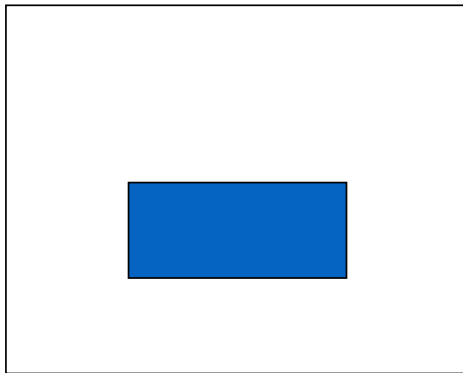
- Inconsistencia en escritura secuencial debido a la no actualización de las copias



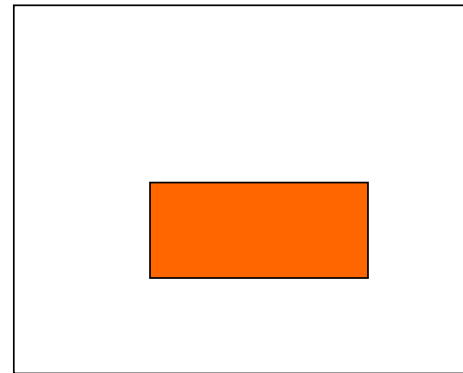
# Problema de la coherencia de cache

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

Cache en el cliente A



Cache en el cliente B



read  $\Rightarrow$  INCONSISTENCIA

Fichero



# Problema de la coherencia de cache

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

Cache en el cliente A



Cache en el cliente B

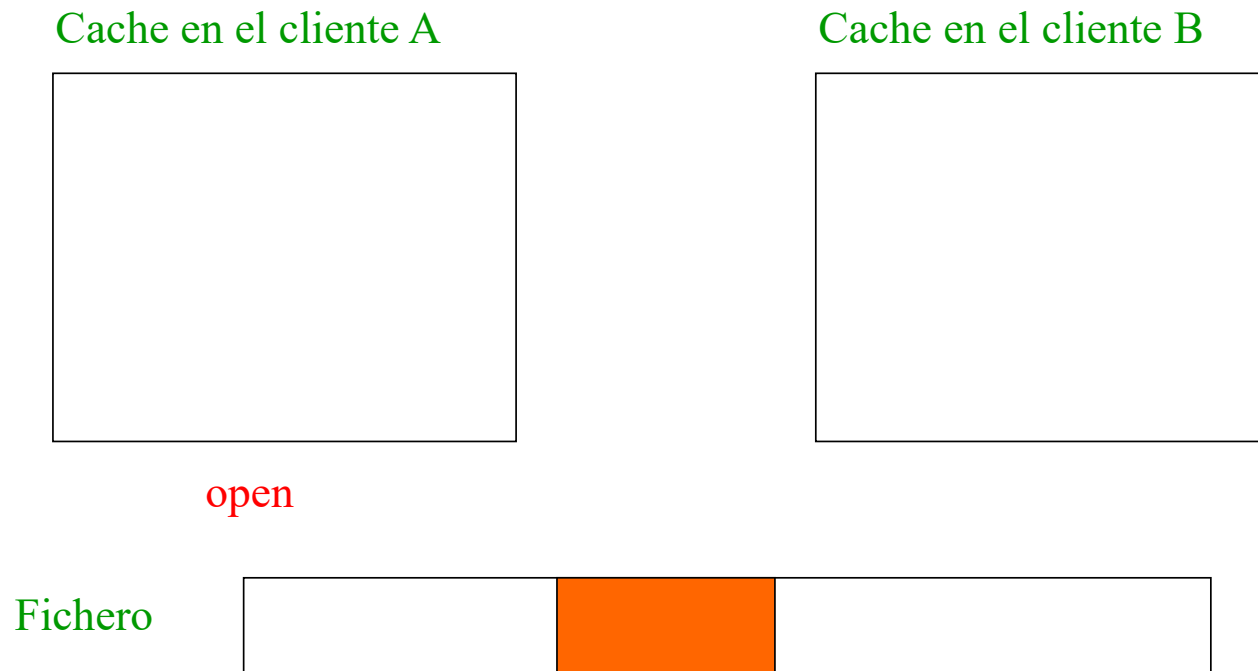


Fichero



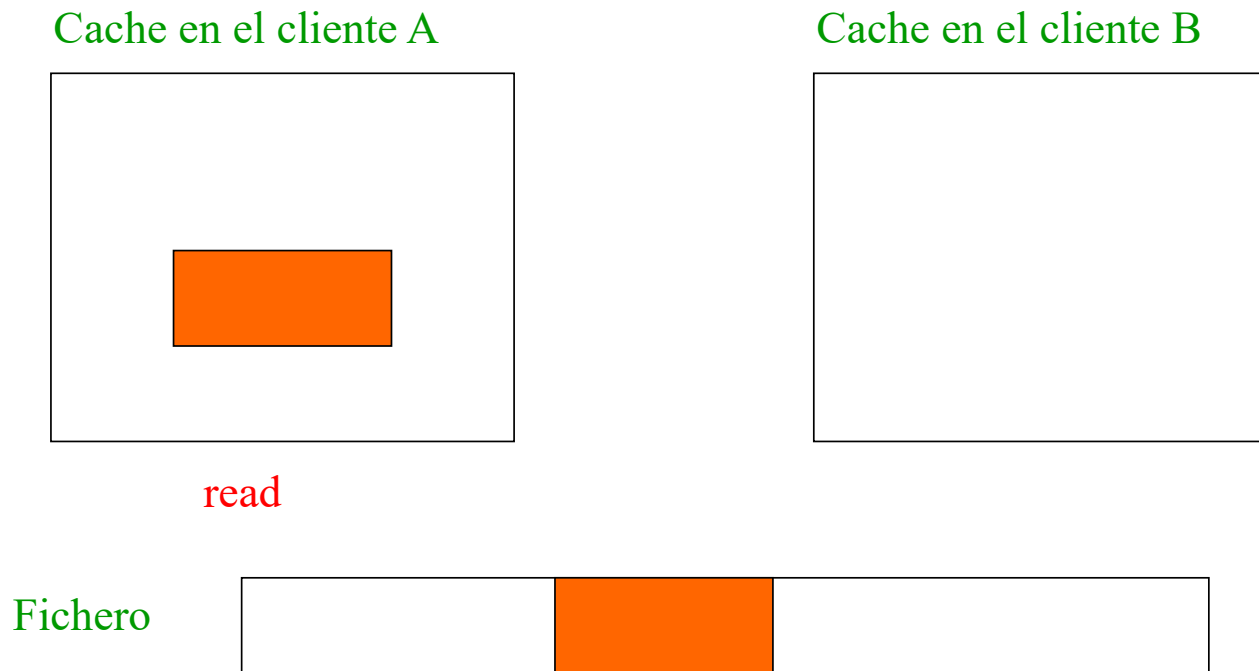
# Problema de la coherencia de cache

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



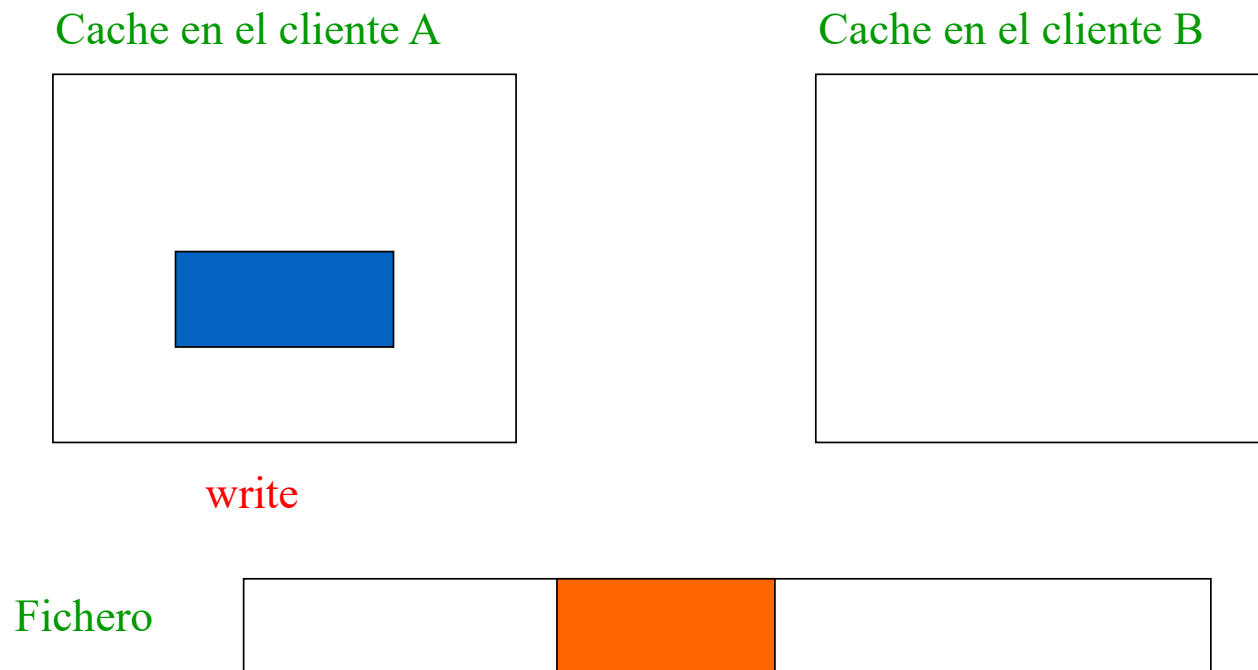
# Problema de la coherencia de cache

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



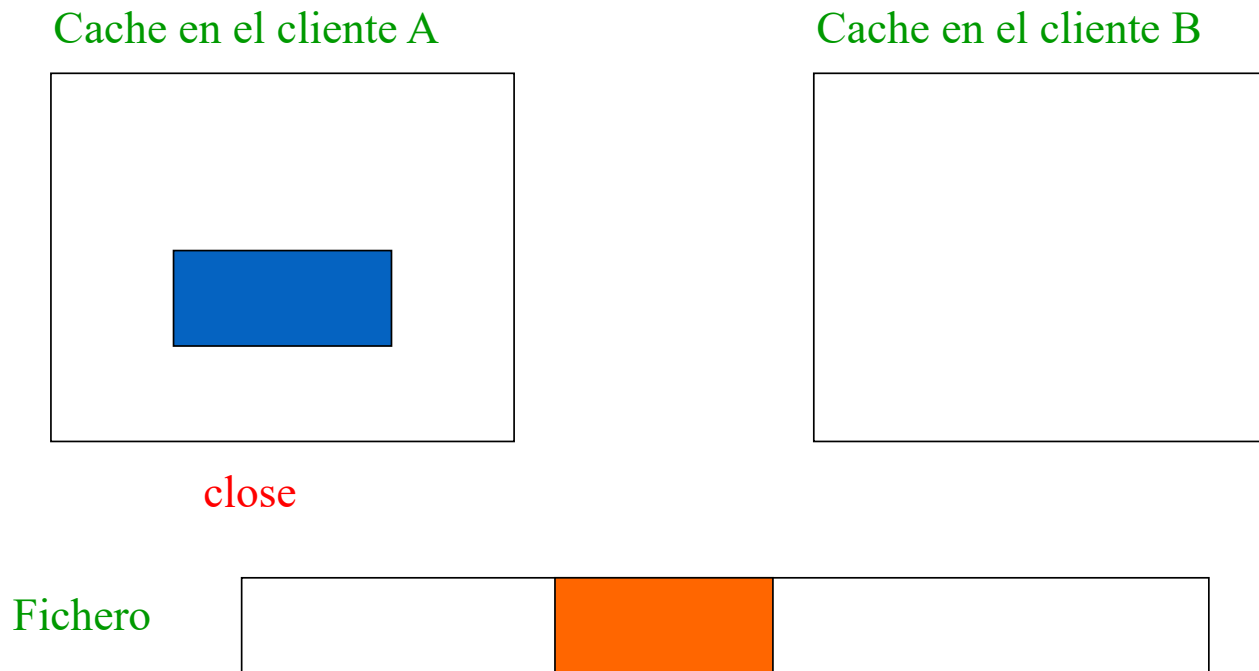
# Problema de la coherencia de cache

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



# Problema de la coherencia de cache

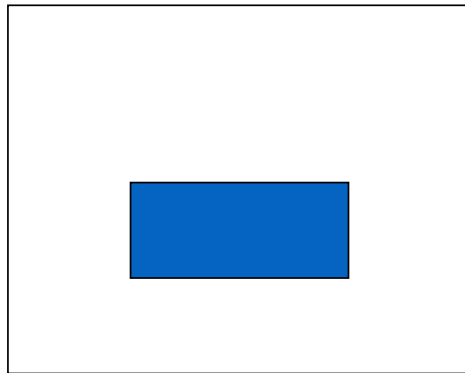
- Inconsistencia en escritura secuencial debido a la utilización de técnicas de escritura diferida



# Problema de la coherencia de cache

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

Cache en el cliente A



Cache en el cliente B

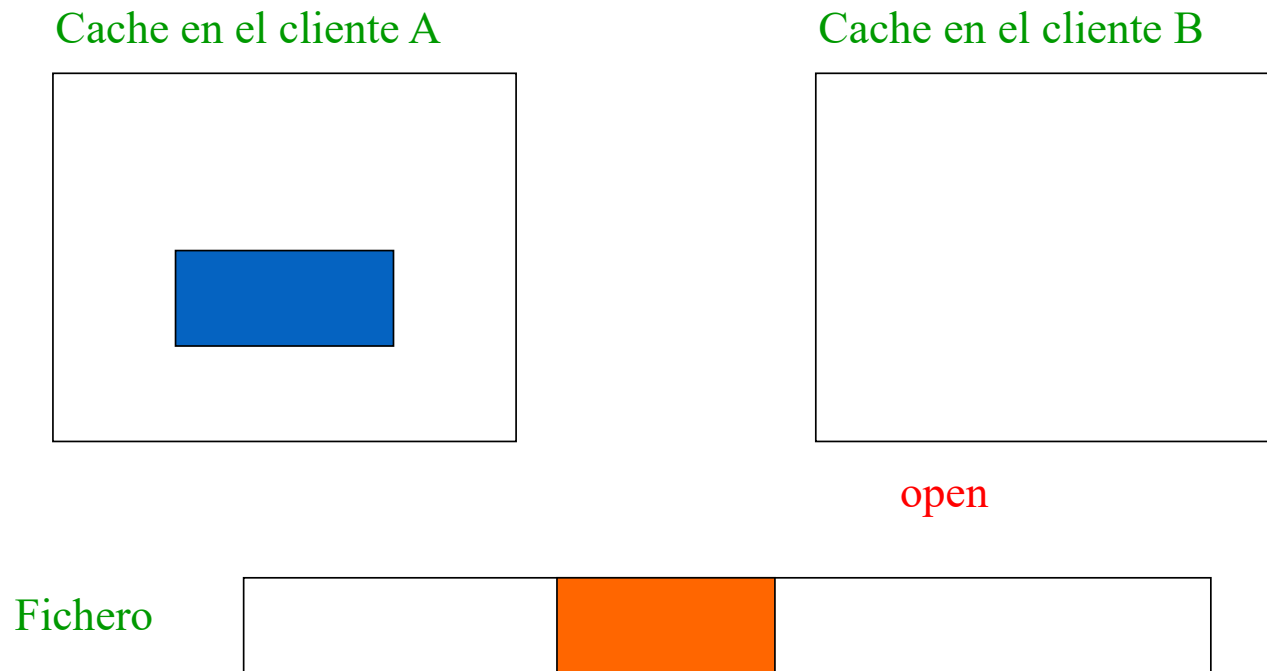


Fichero



# Problema de la coherencia de cache

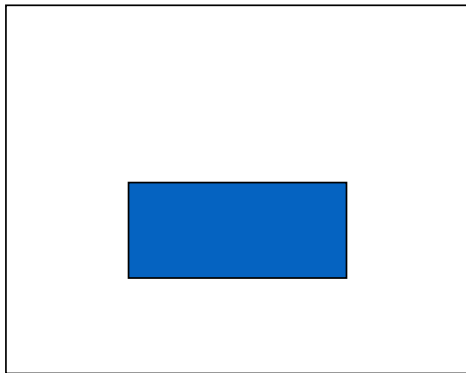
- Inconsistencia en escritura secuencial debido a la utilización de técnicas de escritura diferida



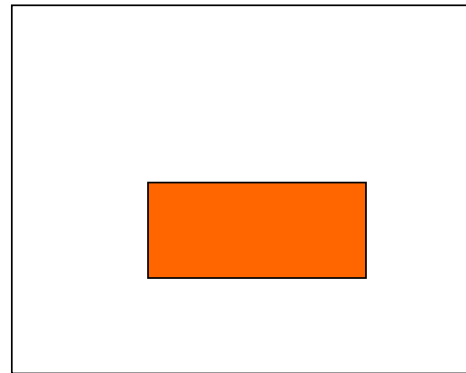
# Problema de la coherencia de cache

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

Cache en el cliente A



Cache en el cliente B



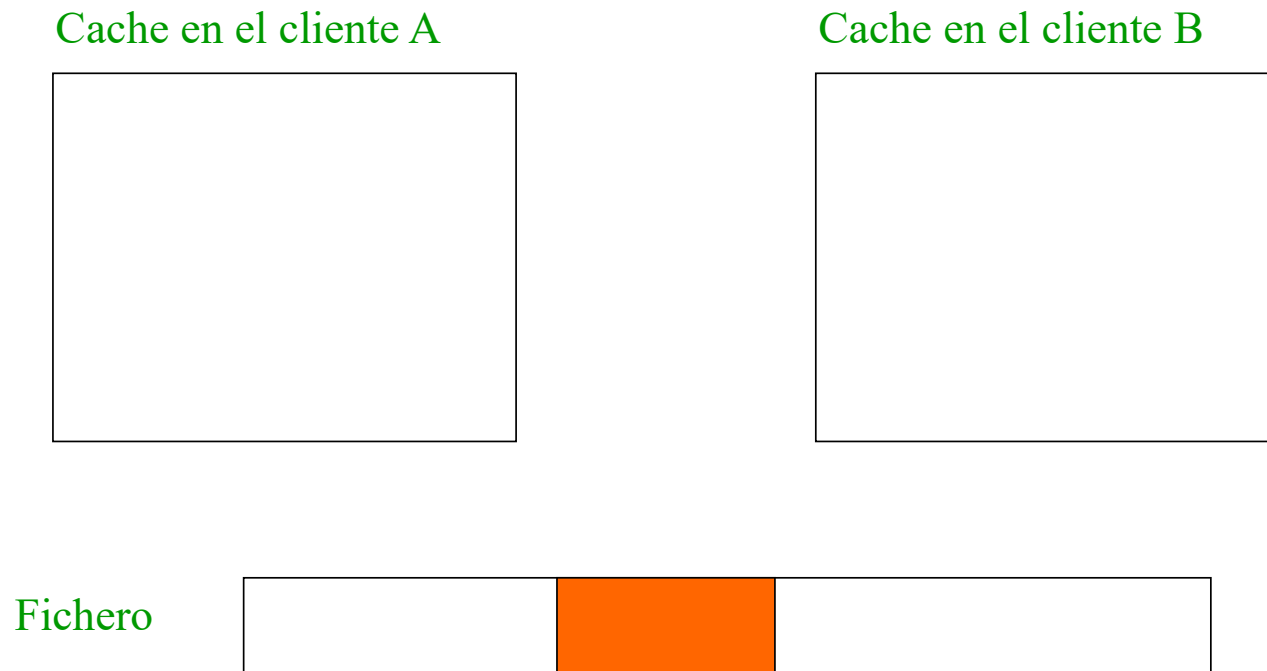
read  $\Rightarrow$  INCONSISTENCIA

Fichero



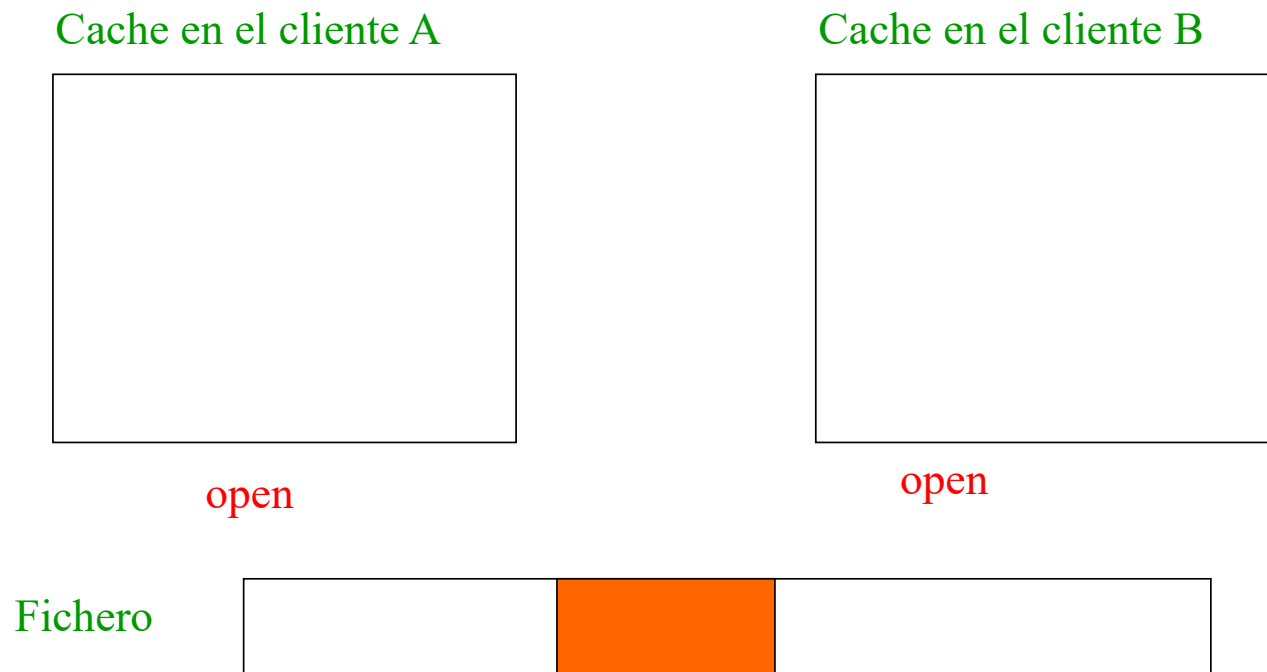
# Problema de la coherencia de cache

- Inconsistencia debido a la coutilización en escritura concurrente



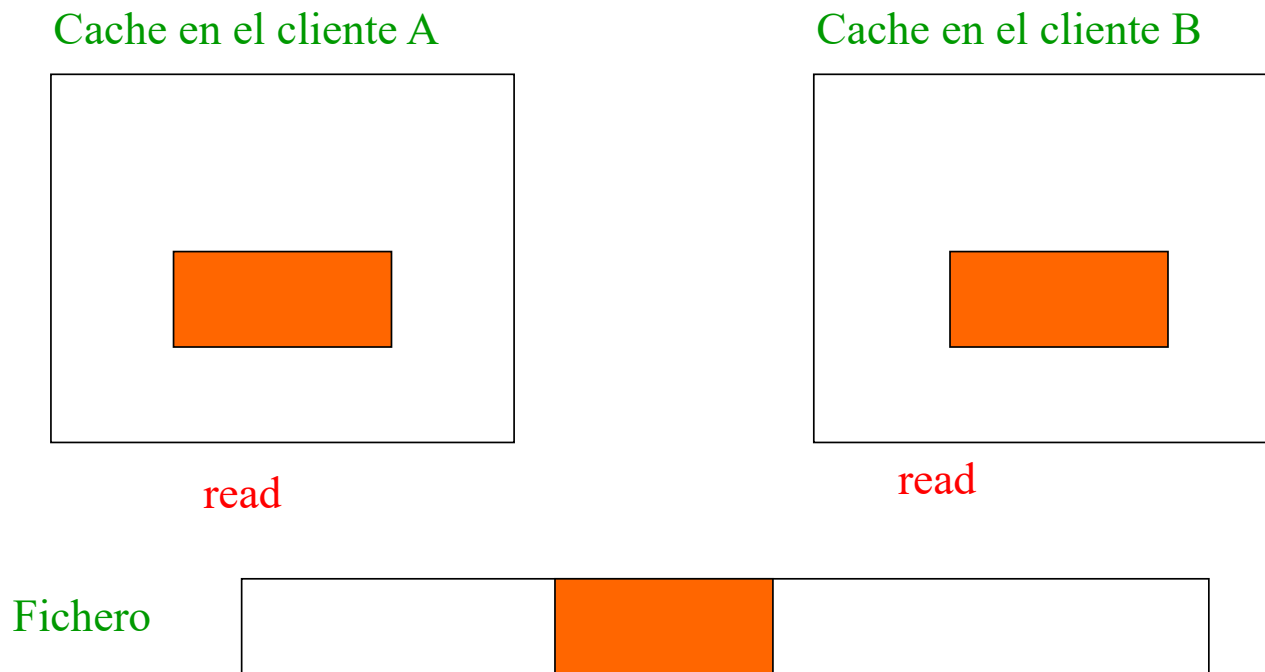
# Problema de la coherencia de cache

- Inconsistencia debido a la colutilización en escritura concurrente



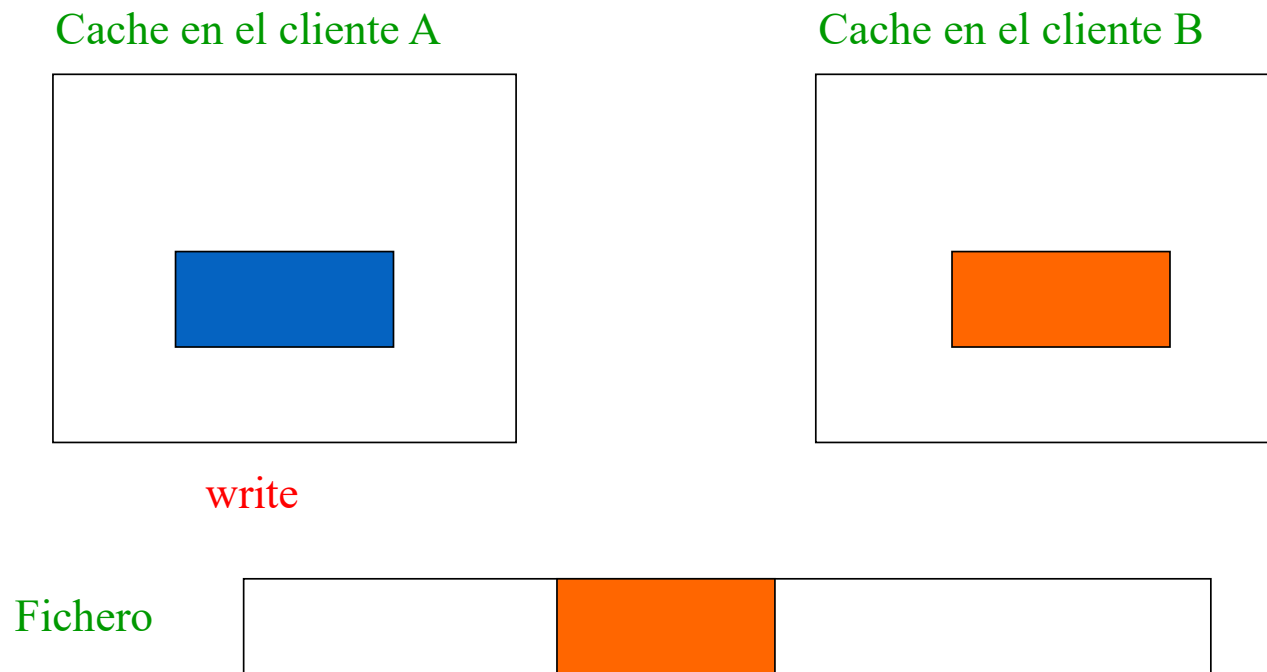
# Problema de la coherencia de cache

- Inconsistencia debido a la colutilización en escritura concurrente



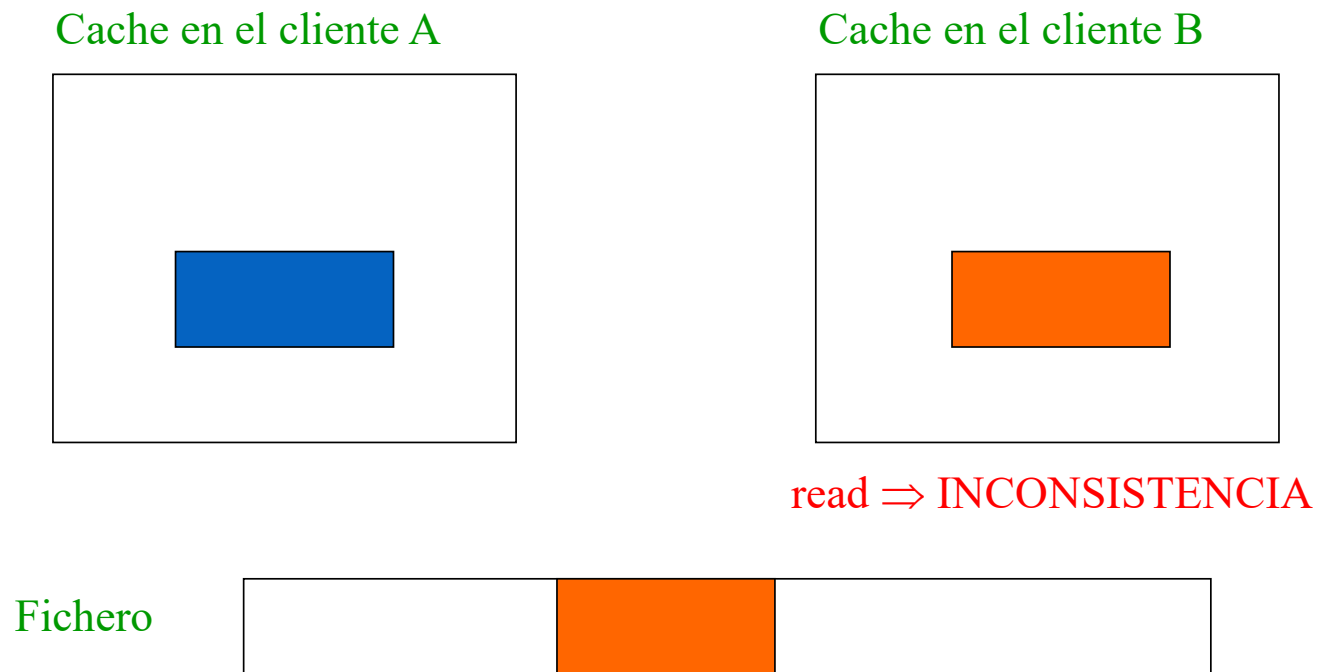
# Problema de la coherencia de cache

- Inconsistencia debido a la colutilización en escritura concurrente



# Problema de la coherencia de cache

- 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 cache 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
- Empleo de protocolos de coherencia de caché
  - Tokens (GPFS)

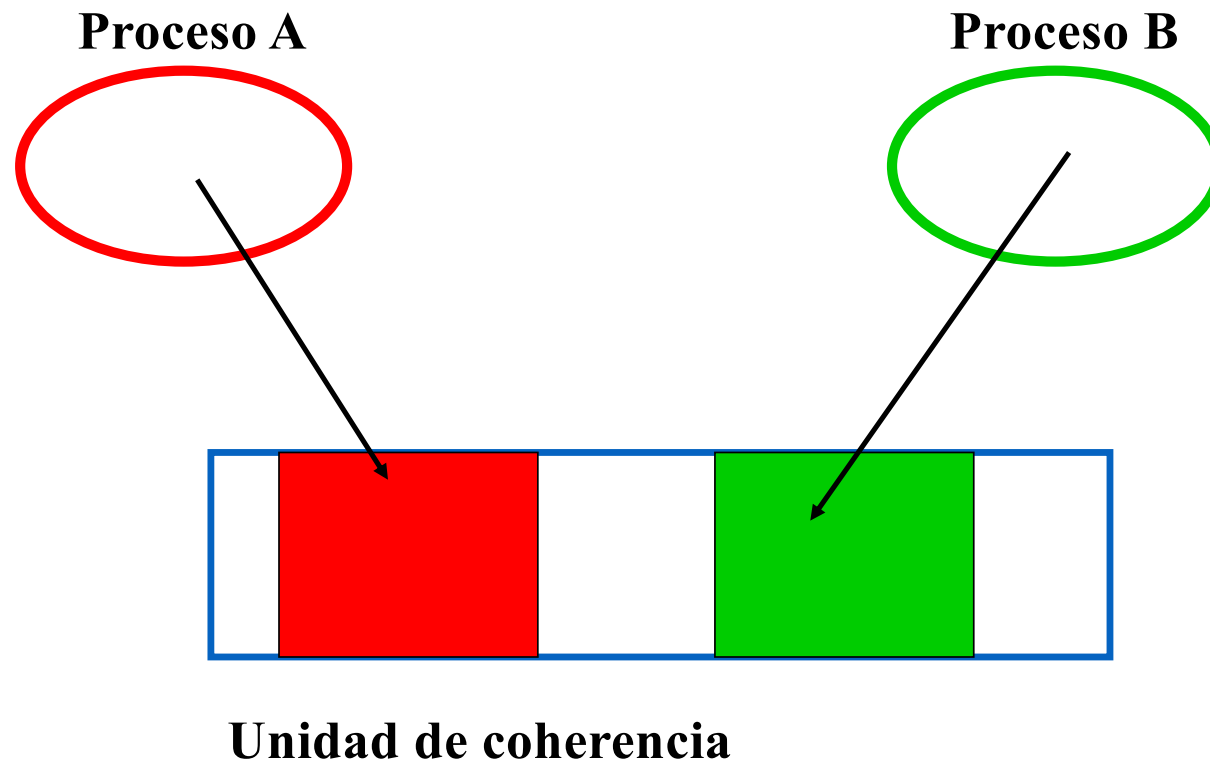
# Protocolos de coherencia de cache

- **Objetivos:**
  - Permitir el uso de cache en los clientes manteniendo coherente la información de acuerdo a la semántica de utilización 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

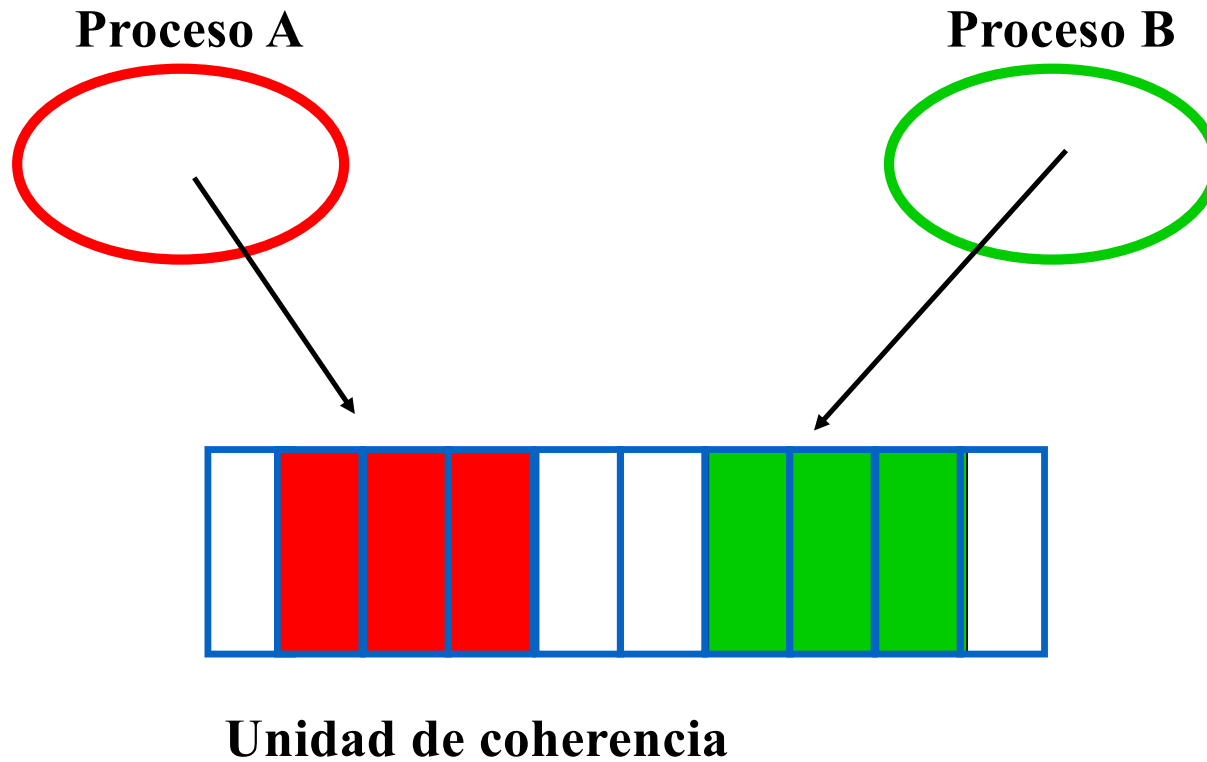
# Granularidad

- Tamaño de la unidad sobre la que se mantiene coherencia
- Puede variar desde un único byte hasta el fichero completo
- Ejemplos:
  - Sprite: fichero completo
  - GPFS: byte
- Con semánticas UNIX, unidades grandes y acceso concurrentes:
  - Posible falsa utilización

# Falsa coutilización



# Sin problemas de coherencia



# Validación de la caché

- Determinar si un dato almacenado en la cache 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

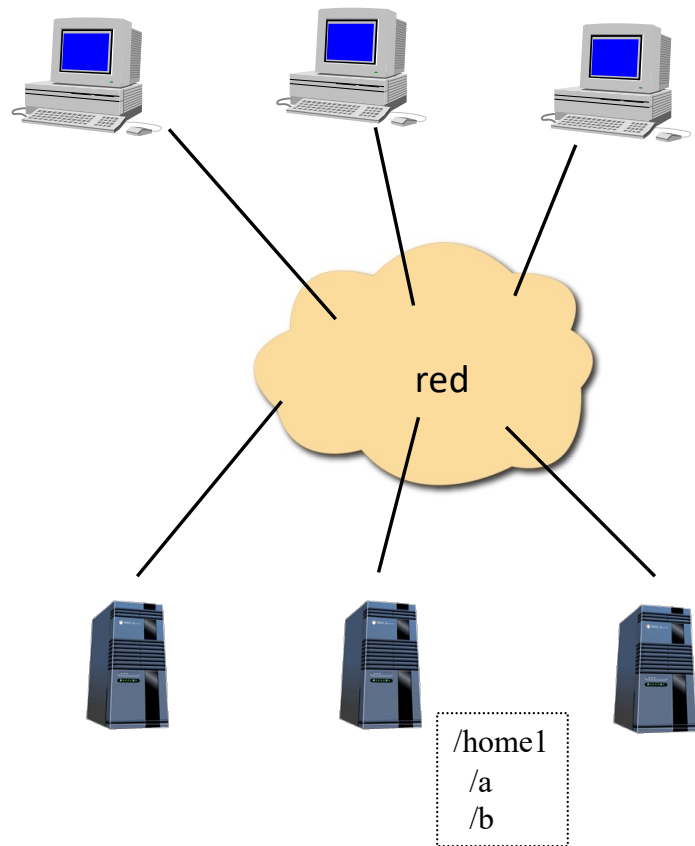
# Validación de la caché

- 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

# Ejemplo de protocolo de coherencia 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

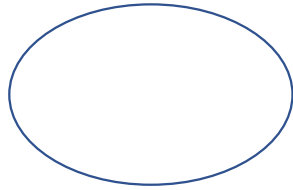
# Ejemplo de protocolo de coherencia Sprite



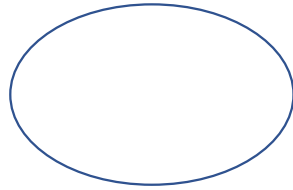
- En cliente operaciones:
  - Open, read, write, close,...
  - Caché en clientes y servidores
  - Semántica UNIX
  - Cliente con caché write-back
  - ¿Protocolo?

# Coherencia de caché en Sprite

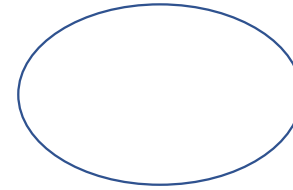
Cliente A



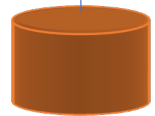
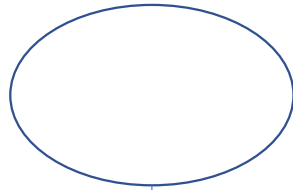
Cliente B



Cliente C

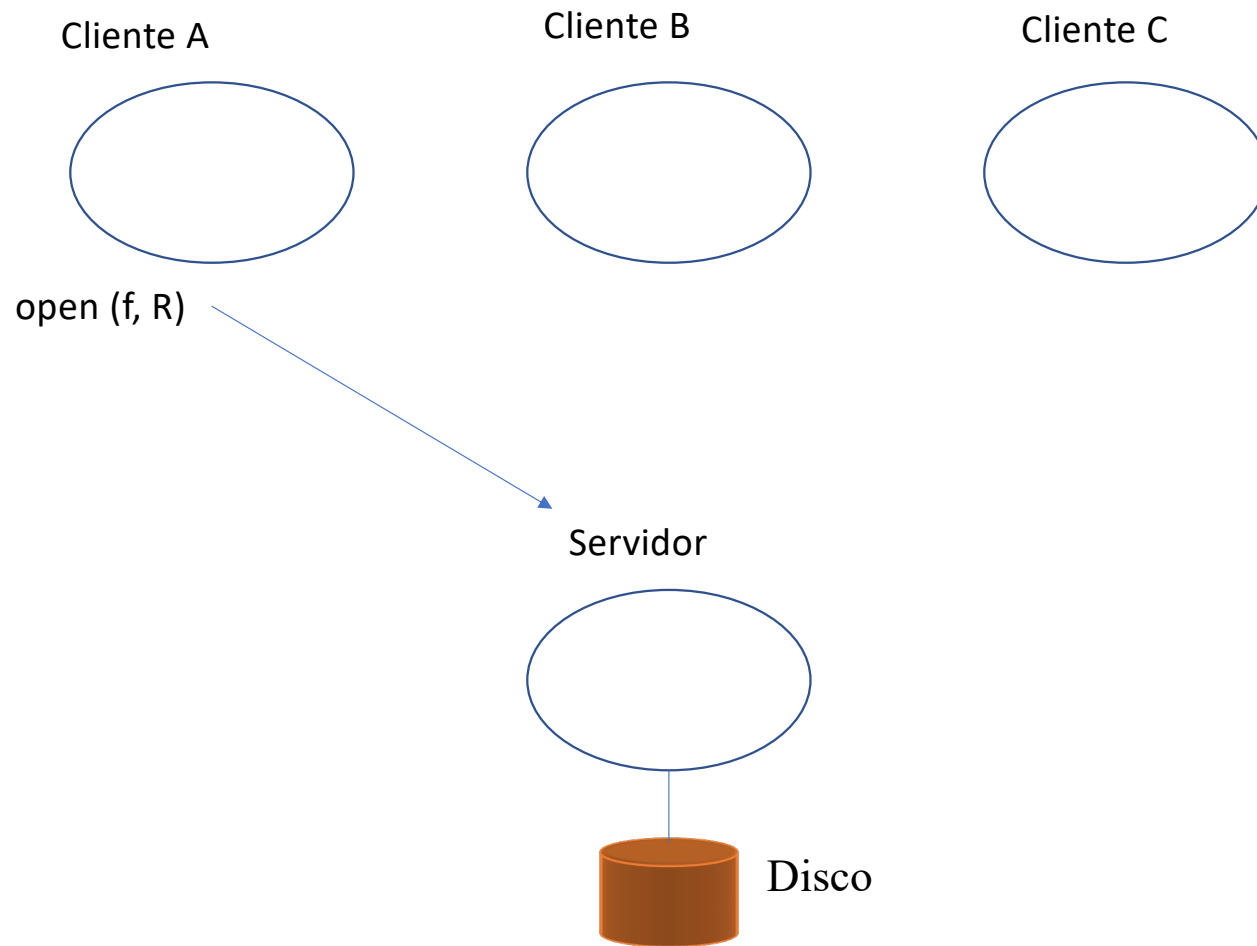


Servidor

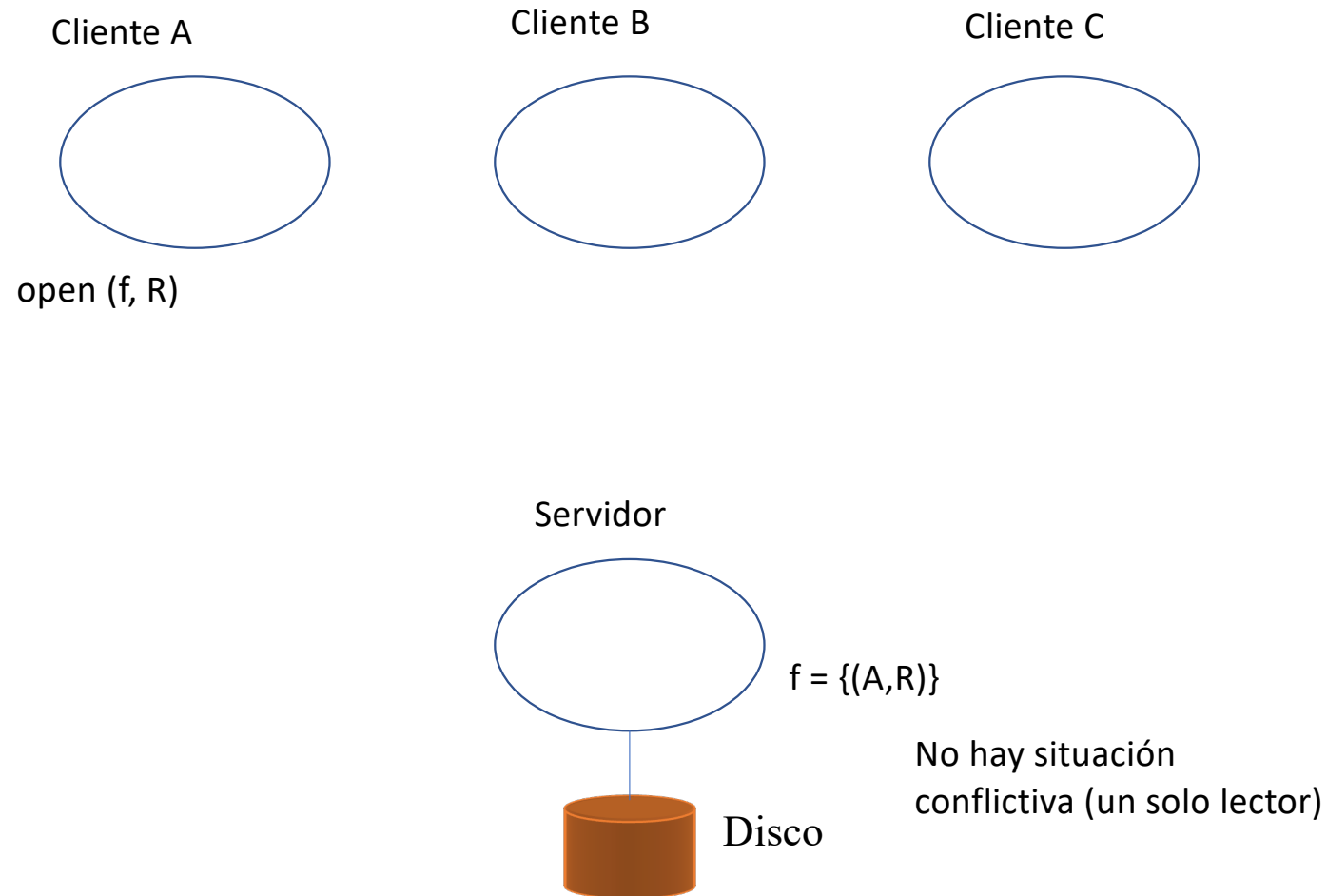


Disco

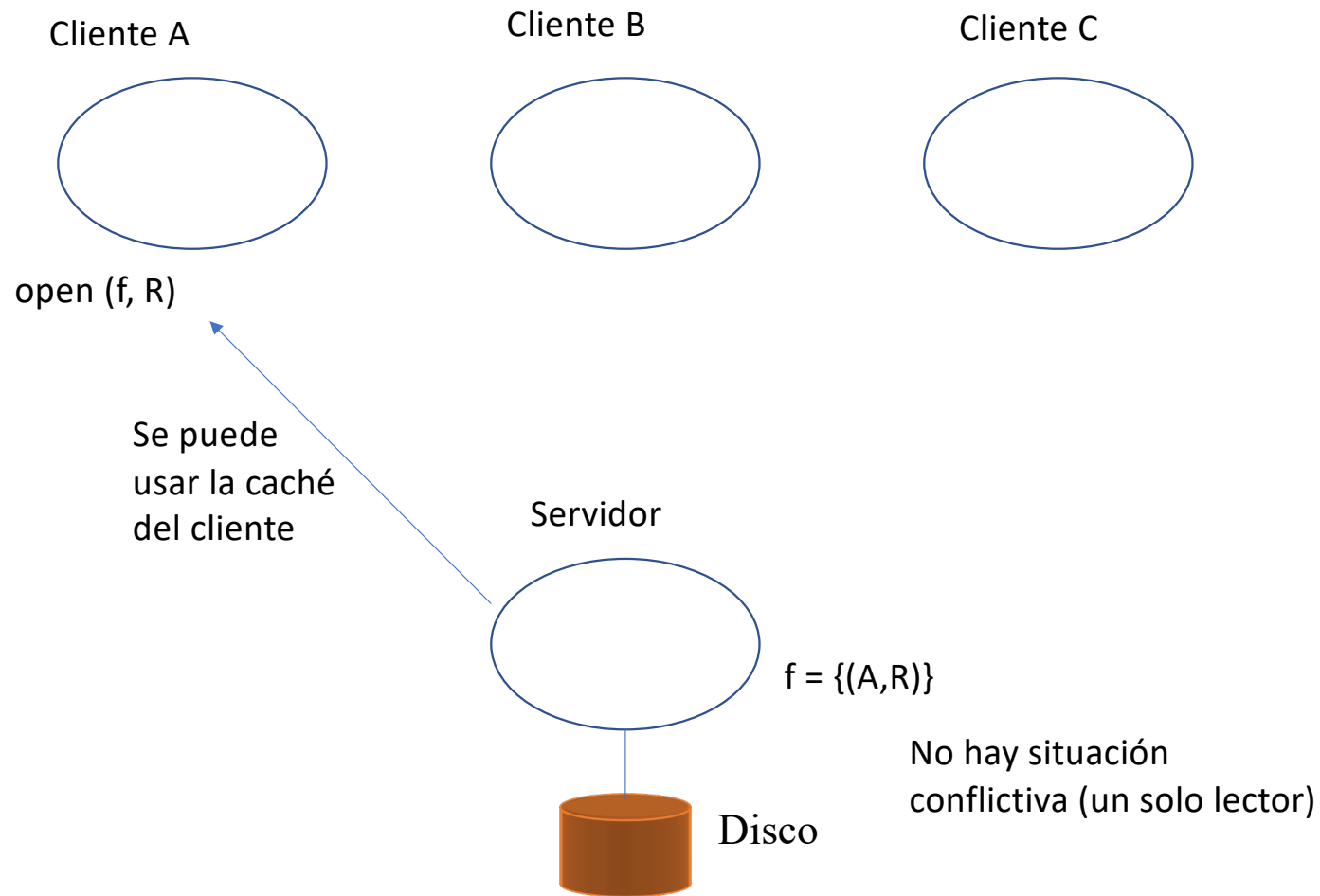
# Coherencia de caché en Sprite



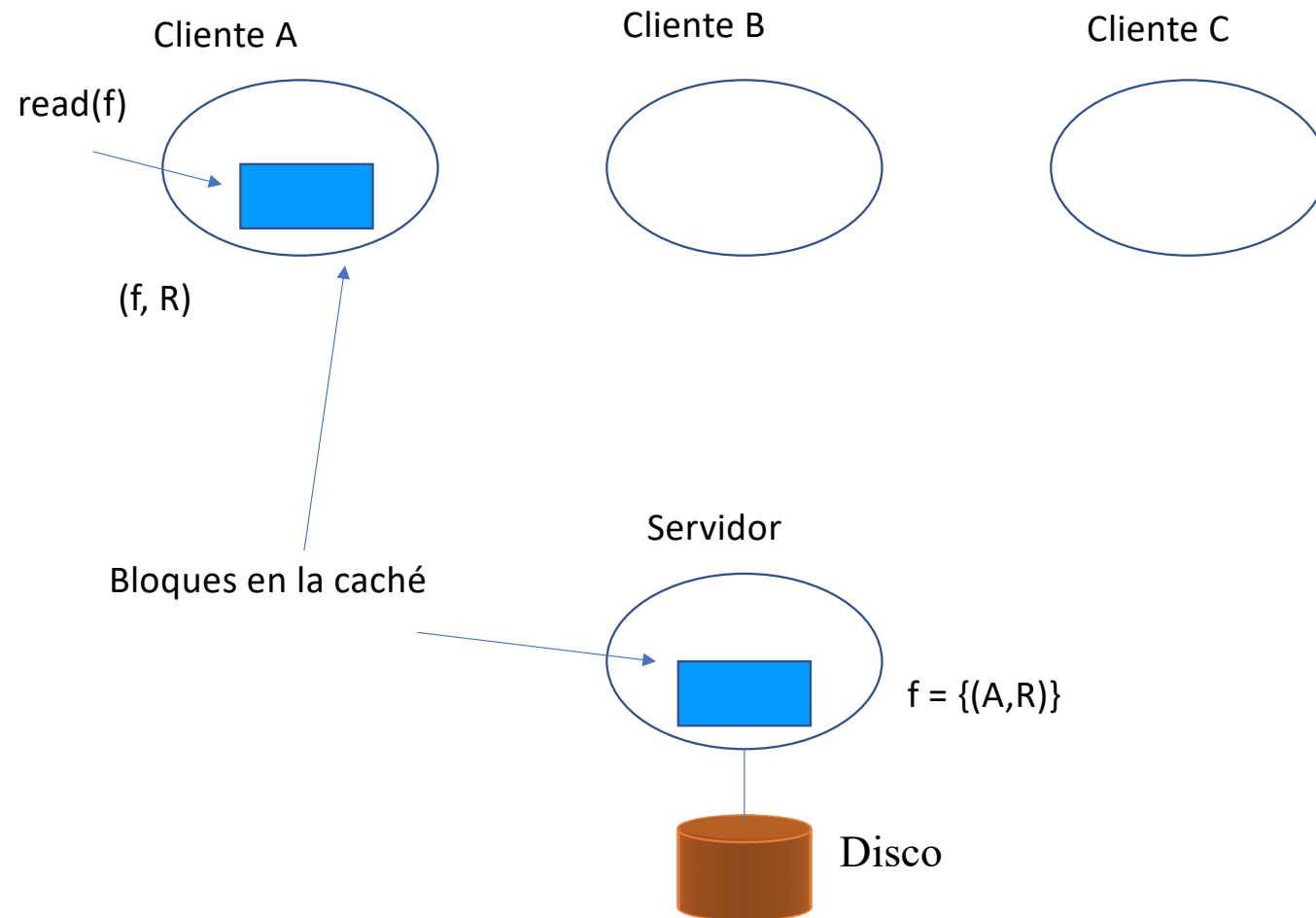
# Coherencia de caché en Sprite



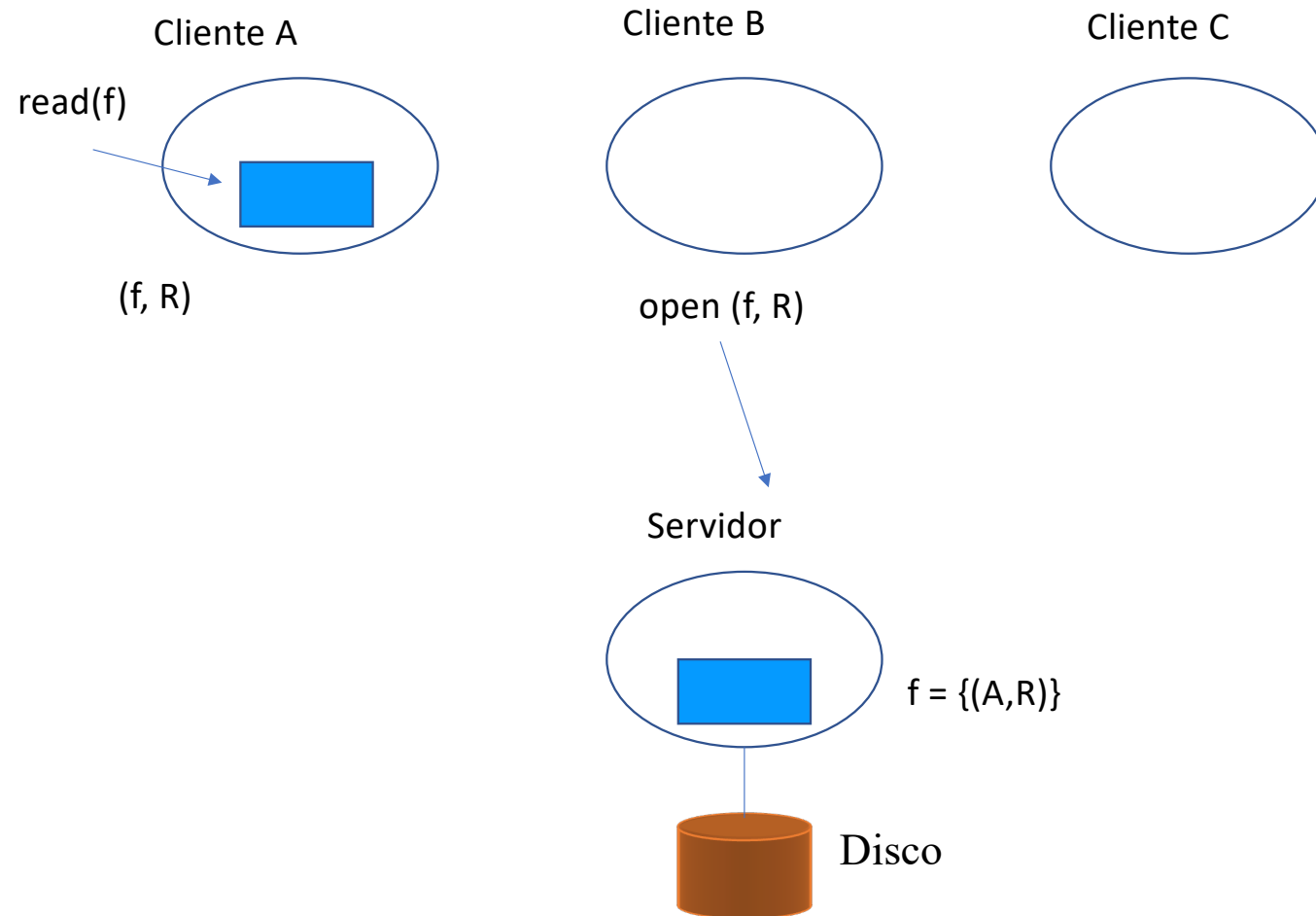
# Coherencia de caché en Sprite



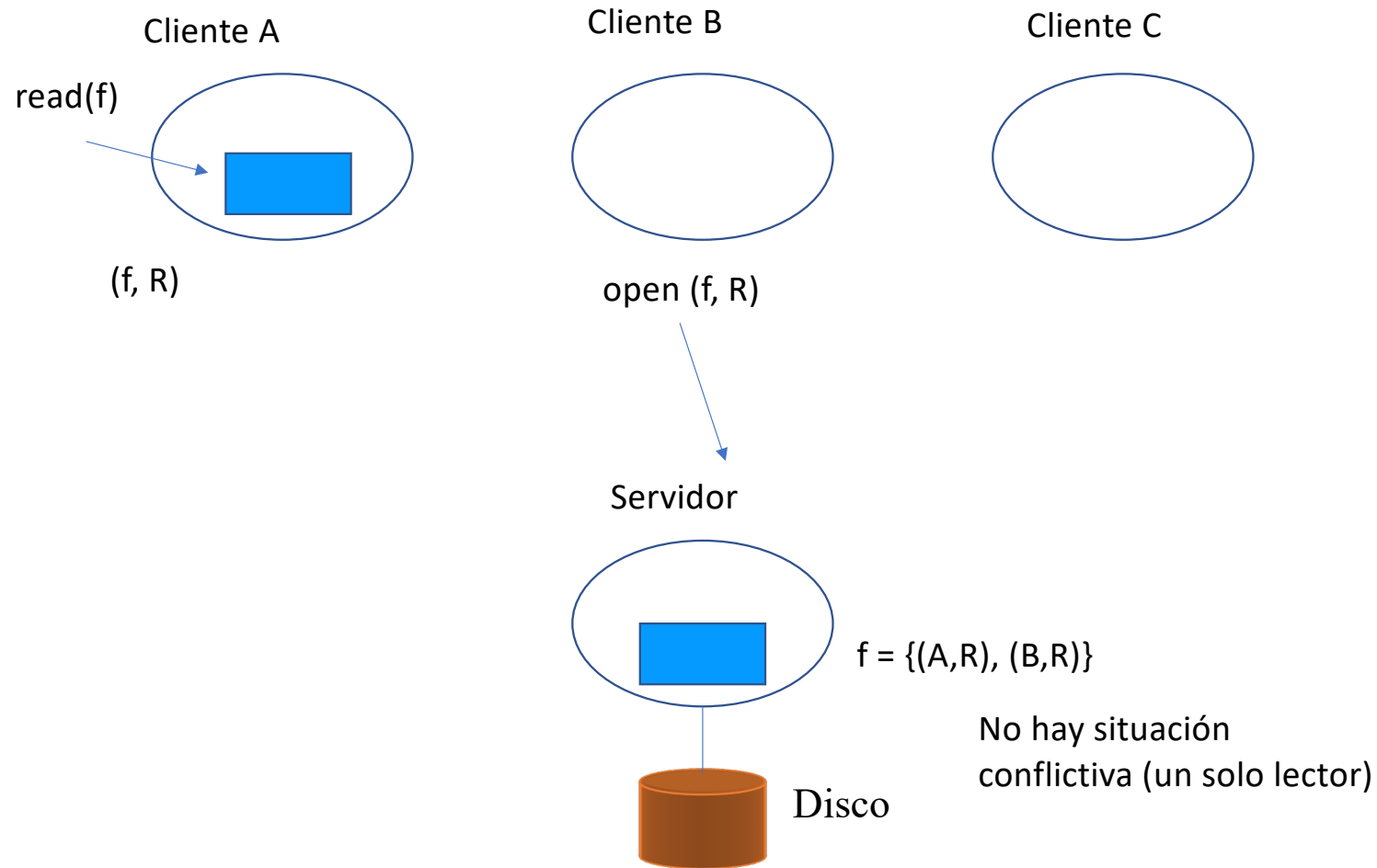
# Coherencia de caché en Sprite



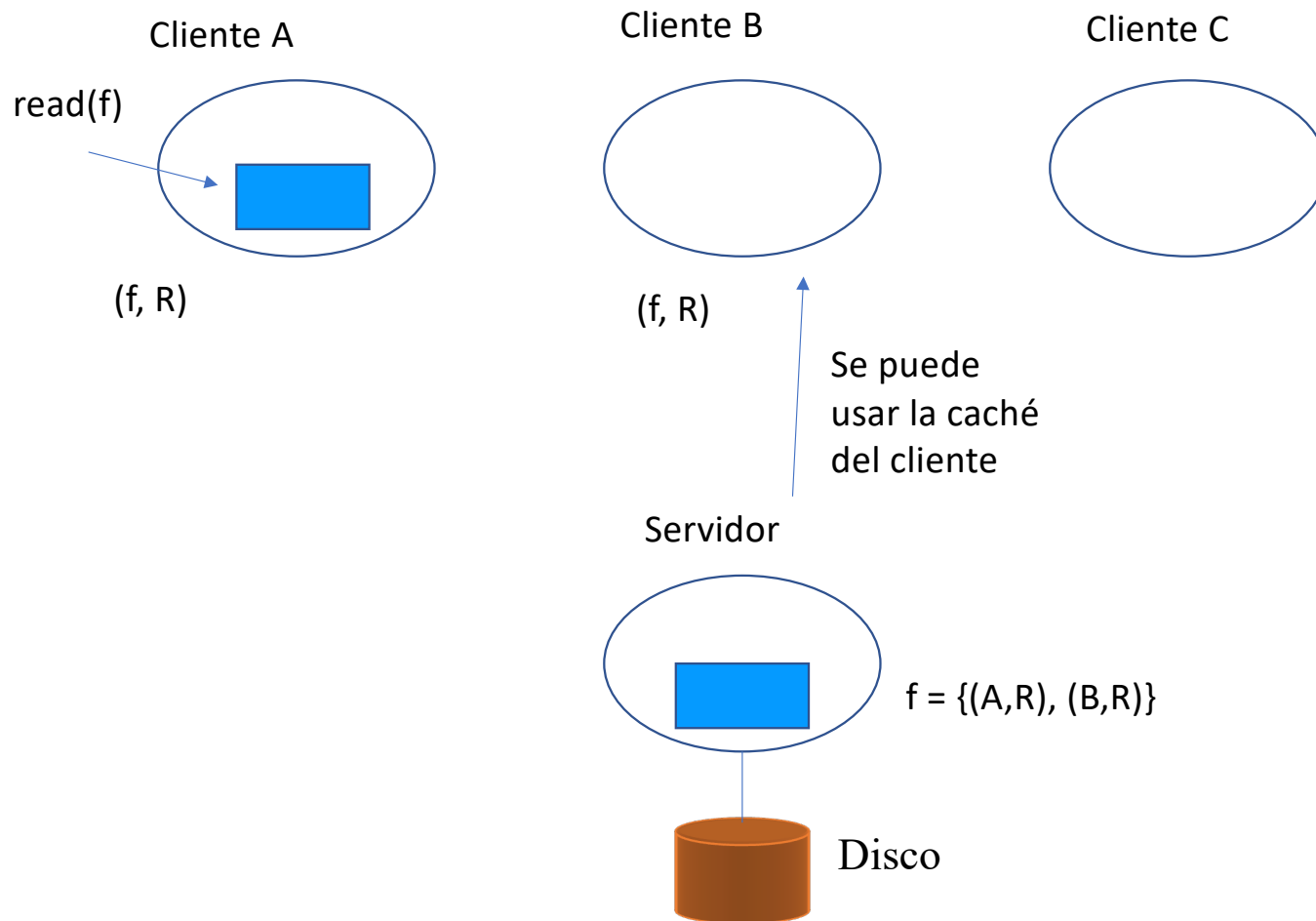
# Coherencia de caché en Sprite



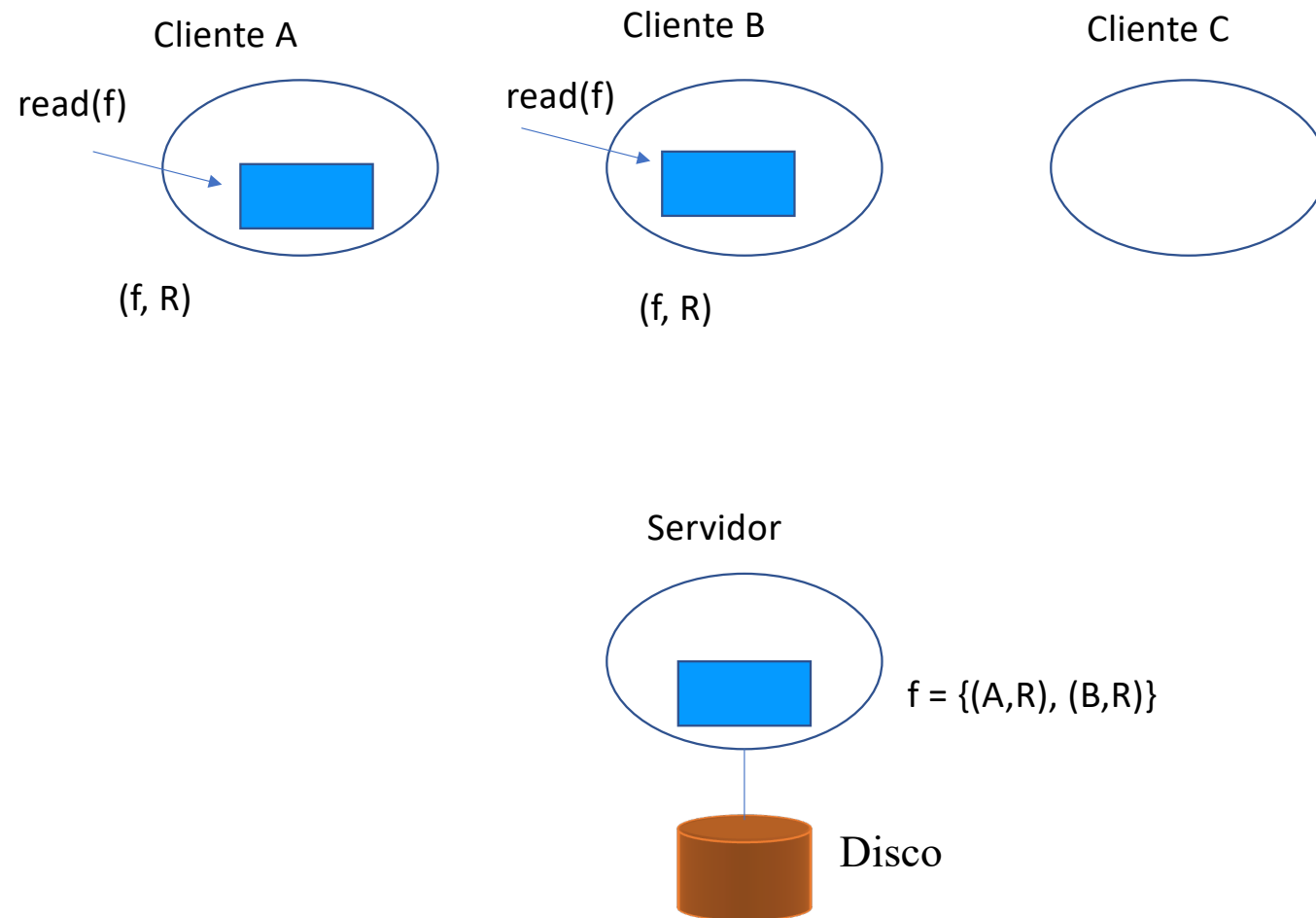
# Coherencia de caché en Sprite



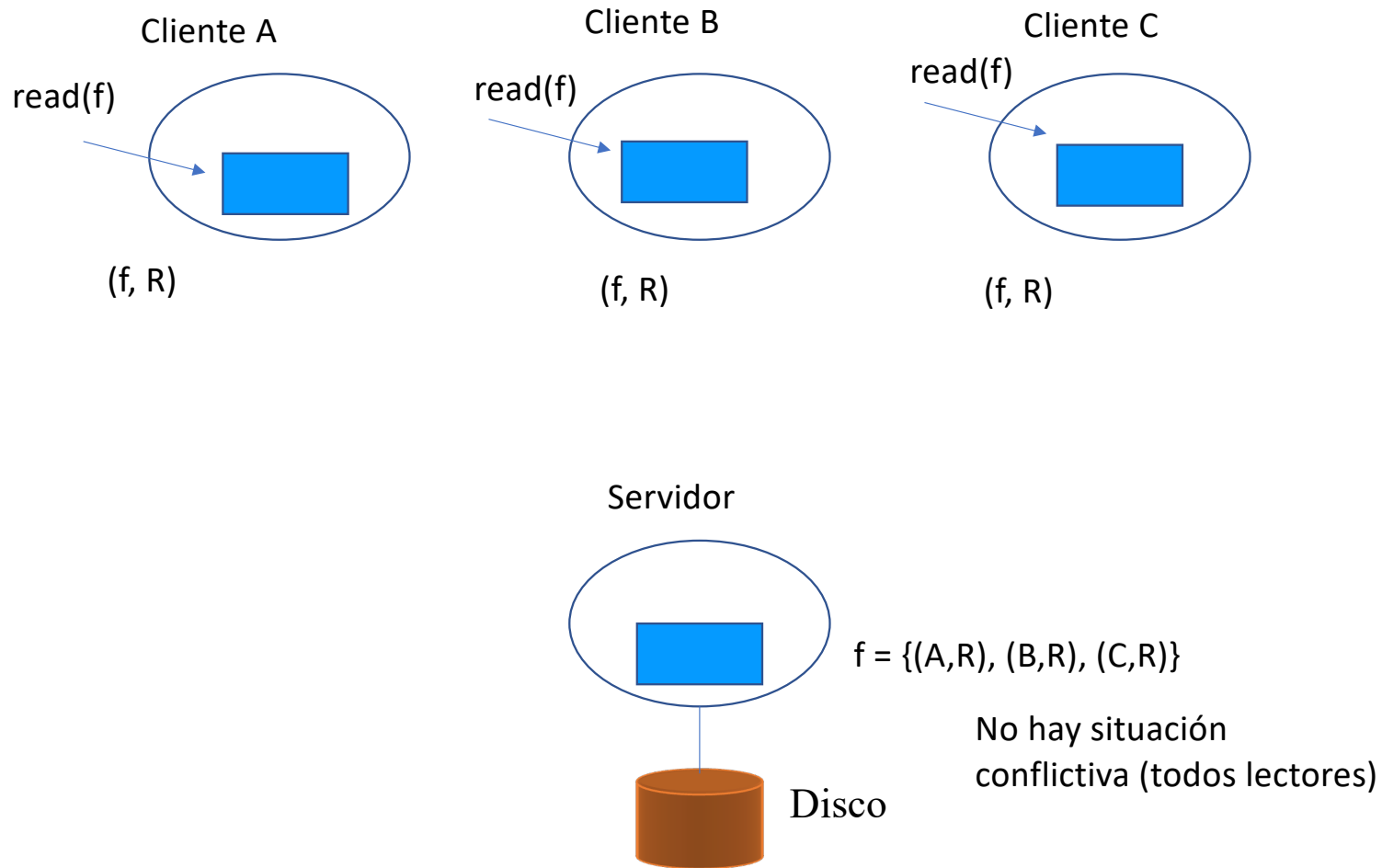
# Coherencia de caché en Sprite



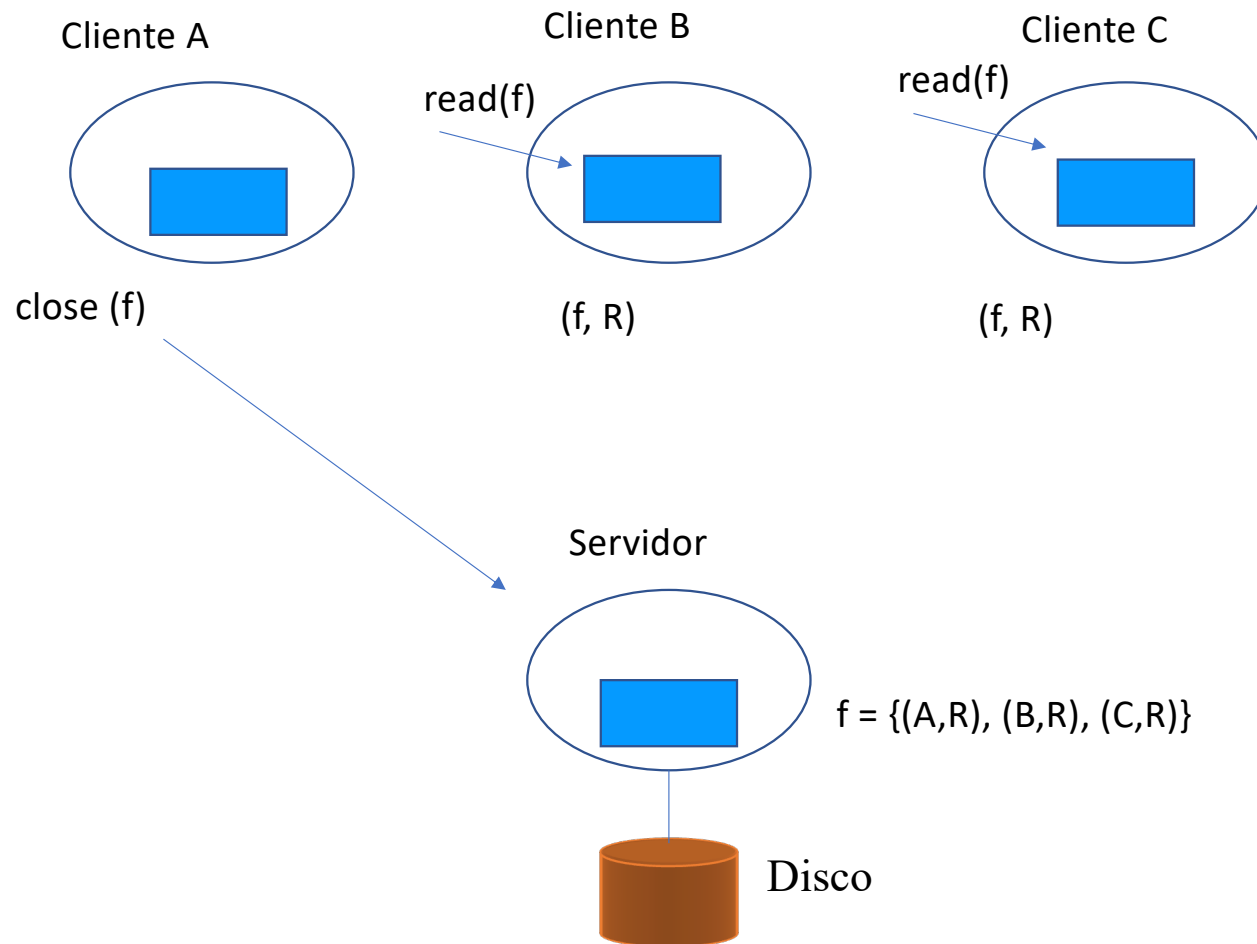
# Coherencia de caché en Sprite



# Coherencia de caché en Sprite

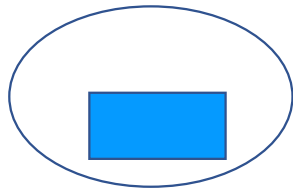


# Coherencia de caché en Sprite

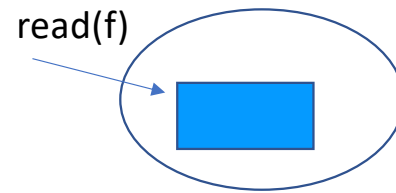


# Coherencia de caché en Sprite

Cliente A

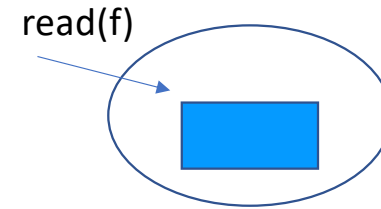


Cliente B



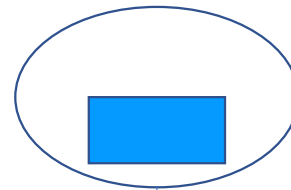
(f, R)

Cliente C

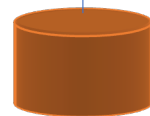


(f, R)

Servidor

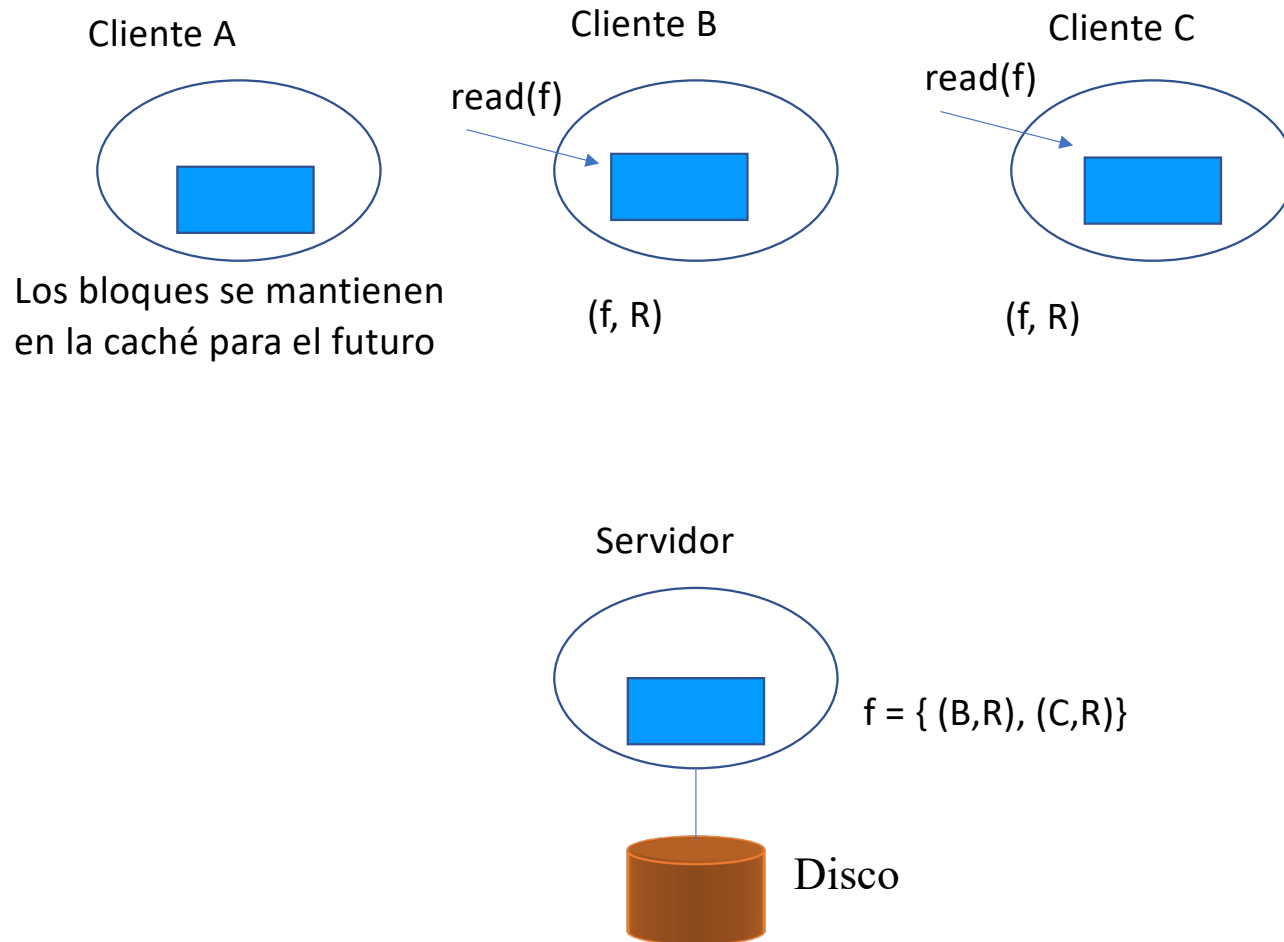


$f = \{ (B,R), (C,R) \}$



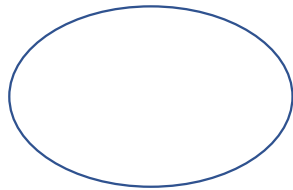
Disco

# Coherencia de caché en Sprite



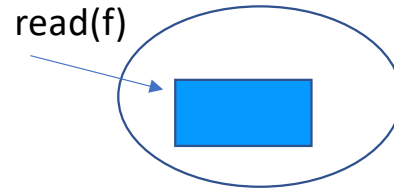
# Coherencia de caché en Sprite

Cliente D



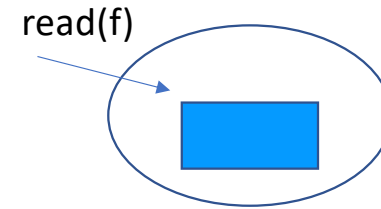
open(f,W)

Cliente B



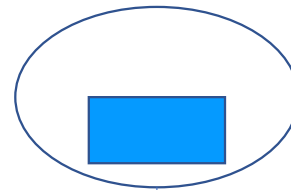
(f, R)

Cliente C



(f, R)

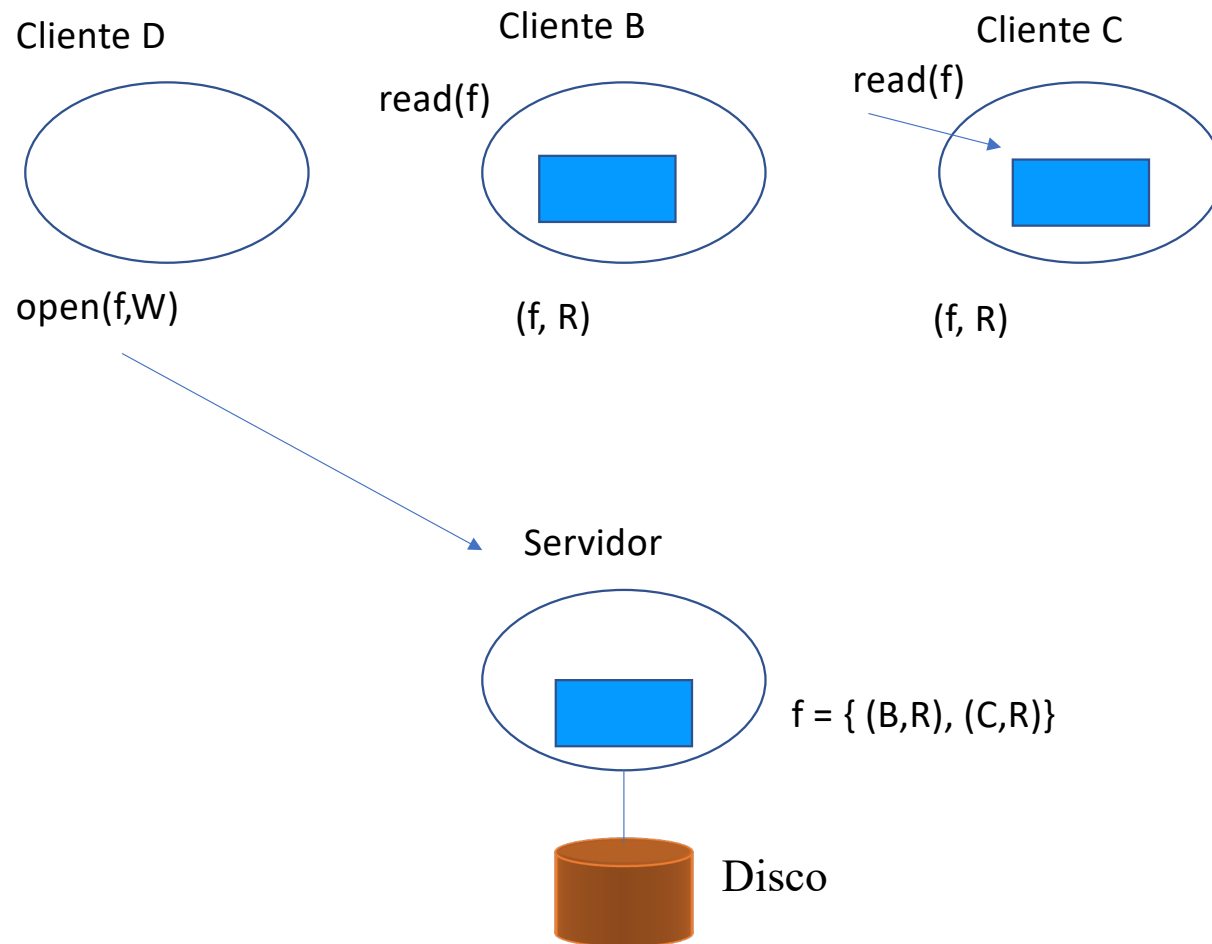
Servidor



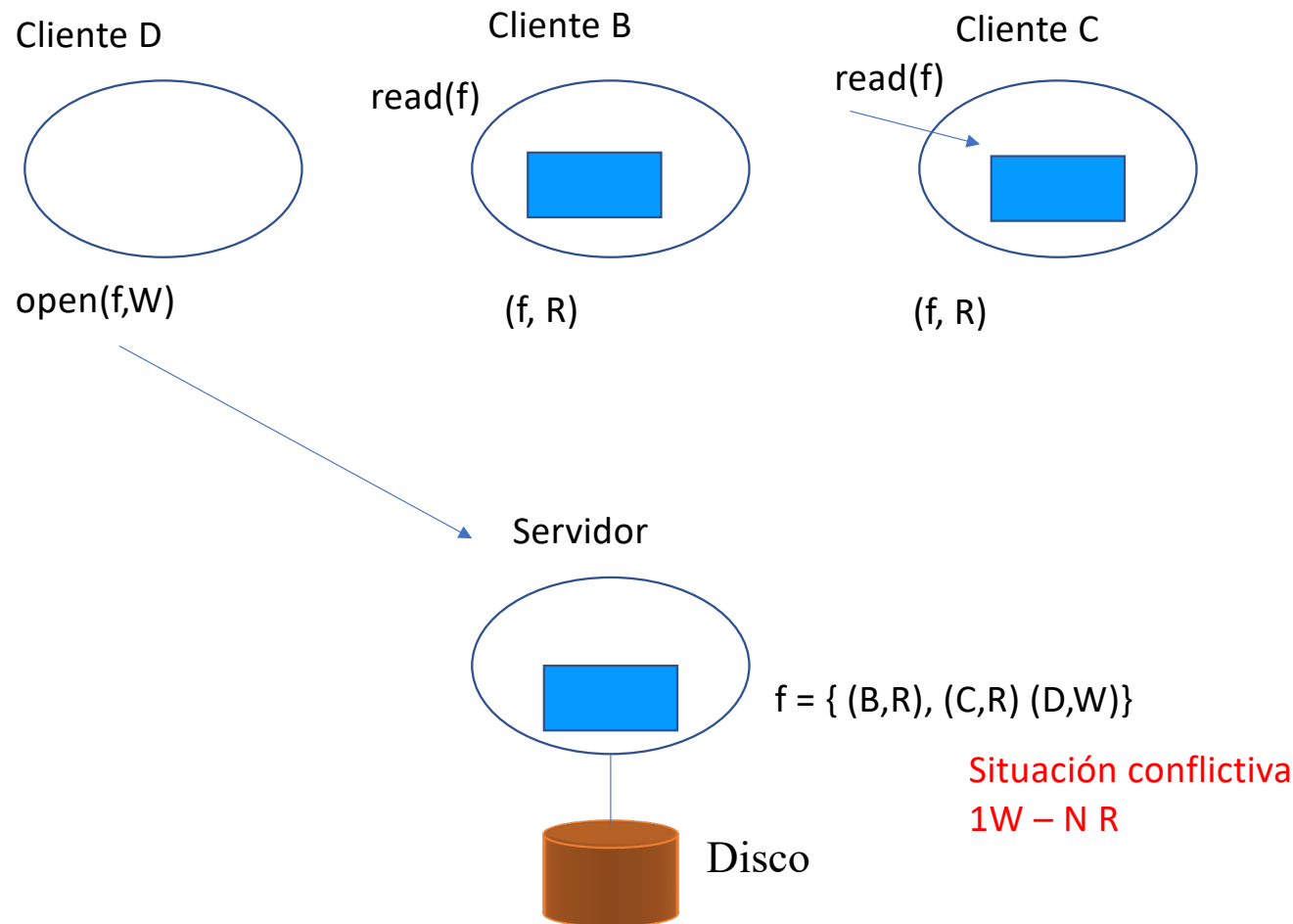
$f = \{ (B,R), (C,R) \}$

Disco

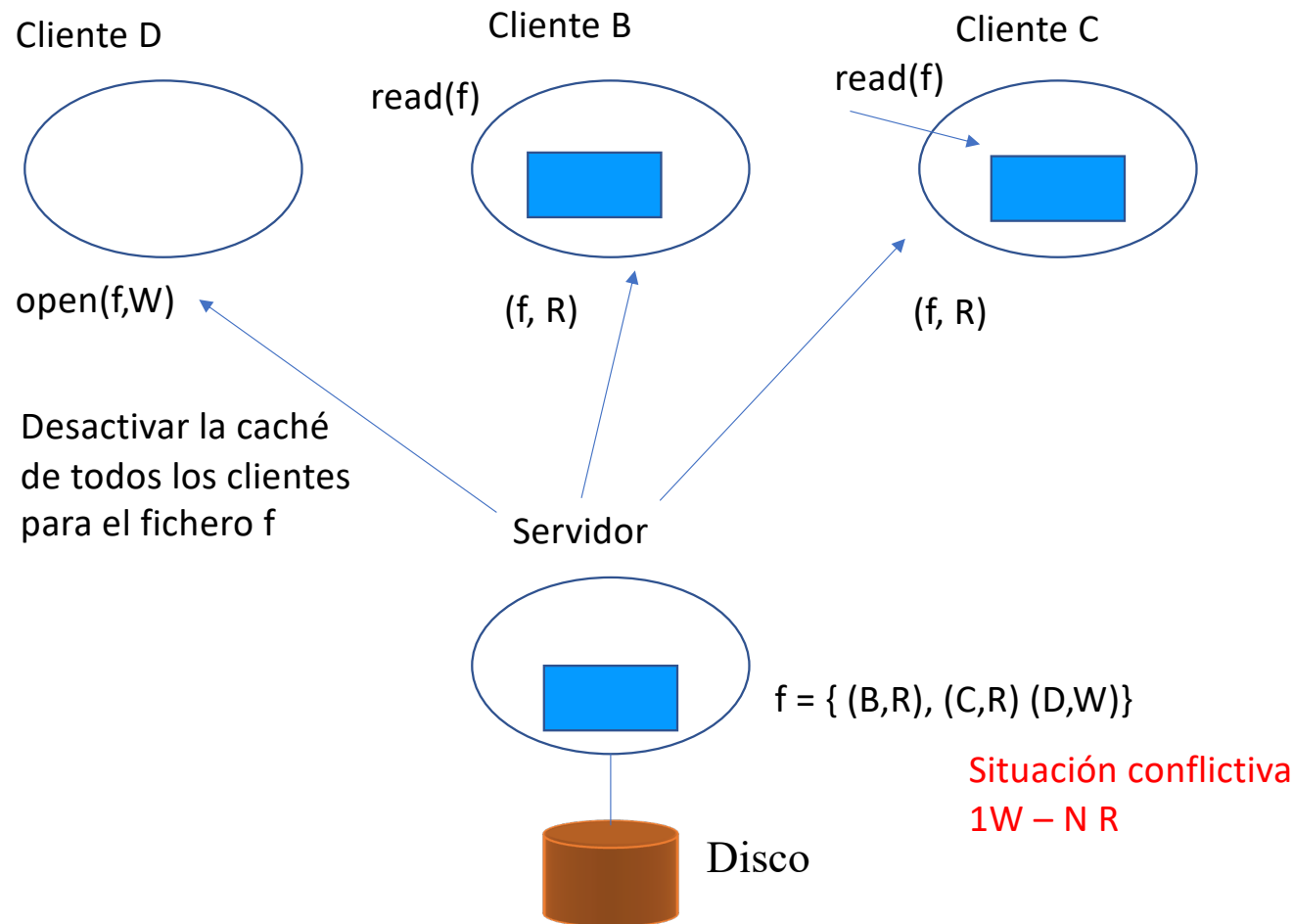
# Coherencia de caché en Sprite



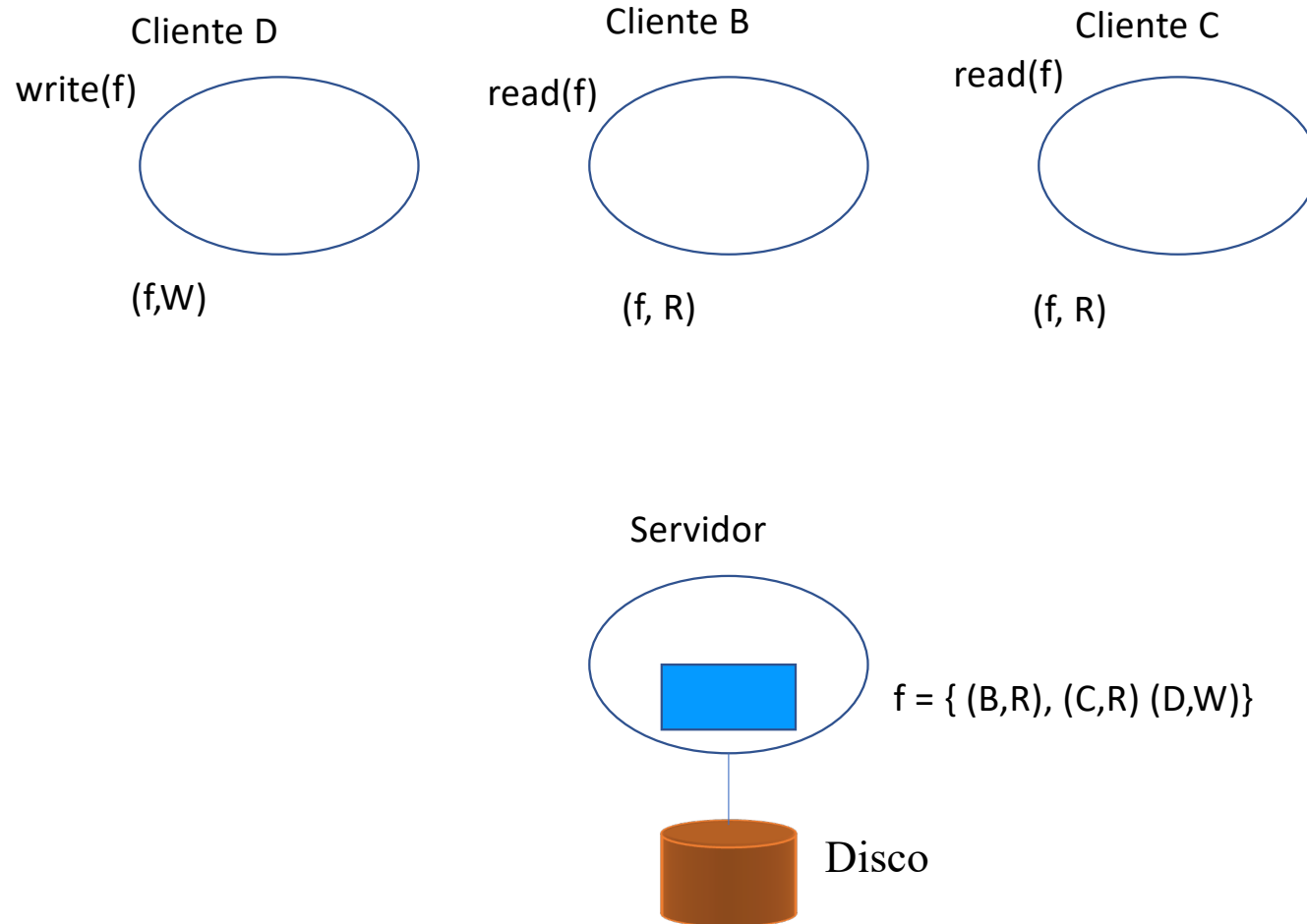
# Coherencia de caché en Sprite



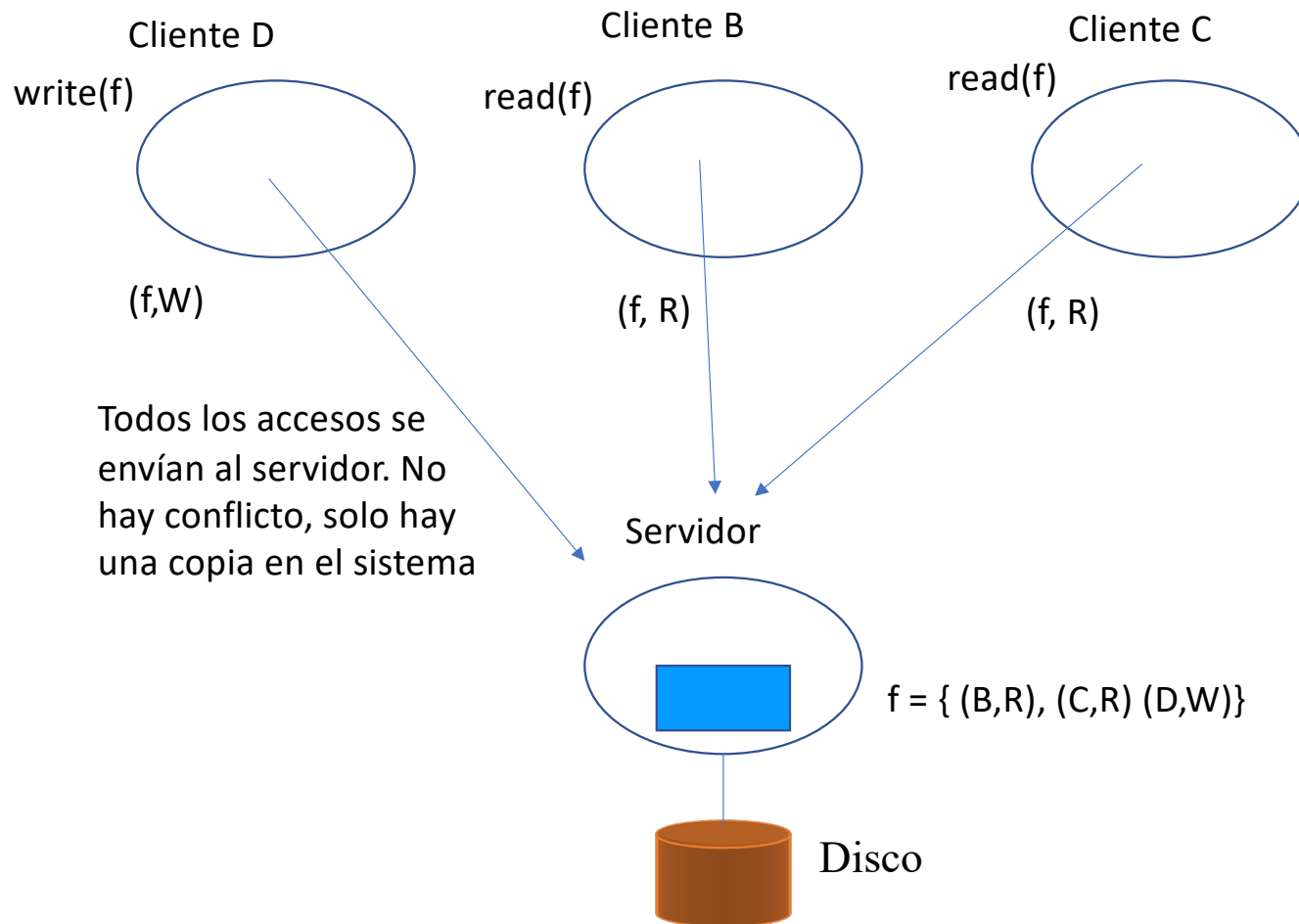
# Coherencia de caché en Sprite



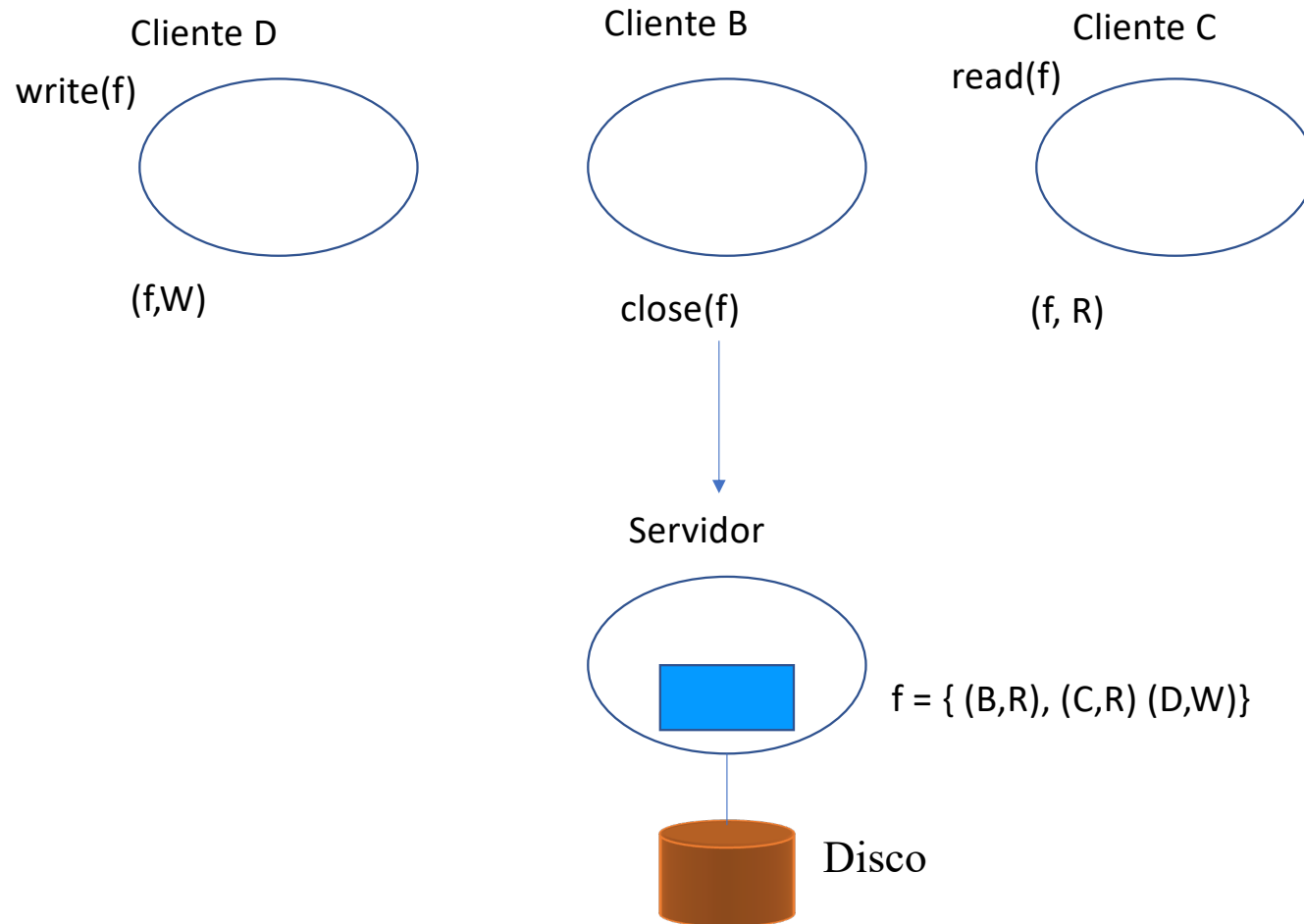
# Coherencia de caché en Sprite



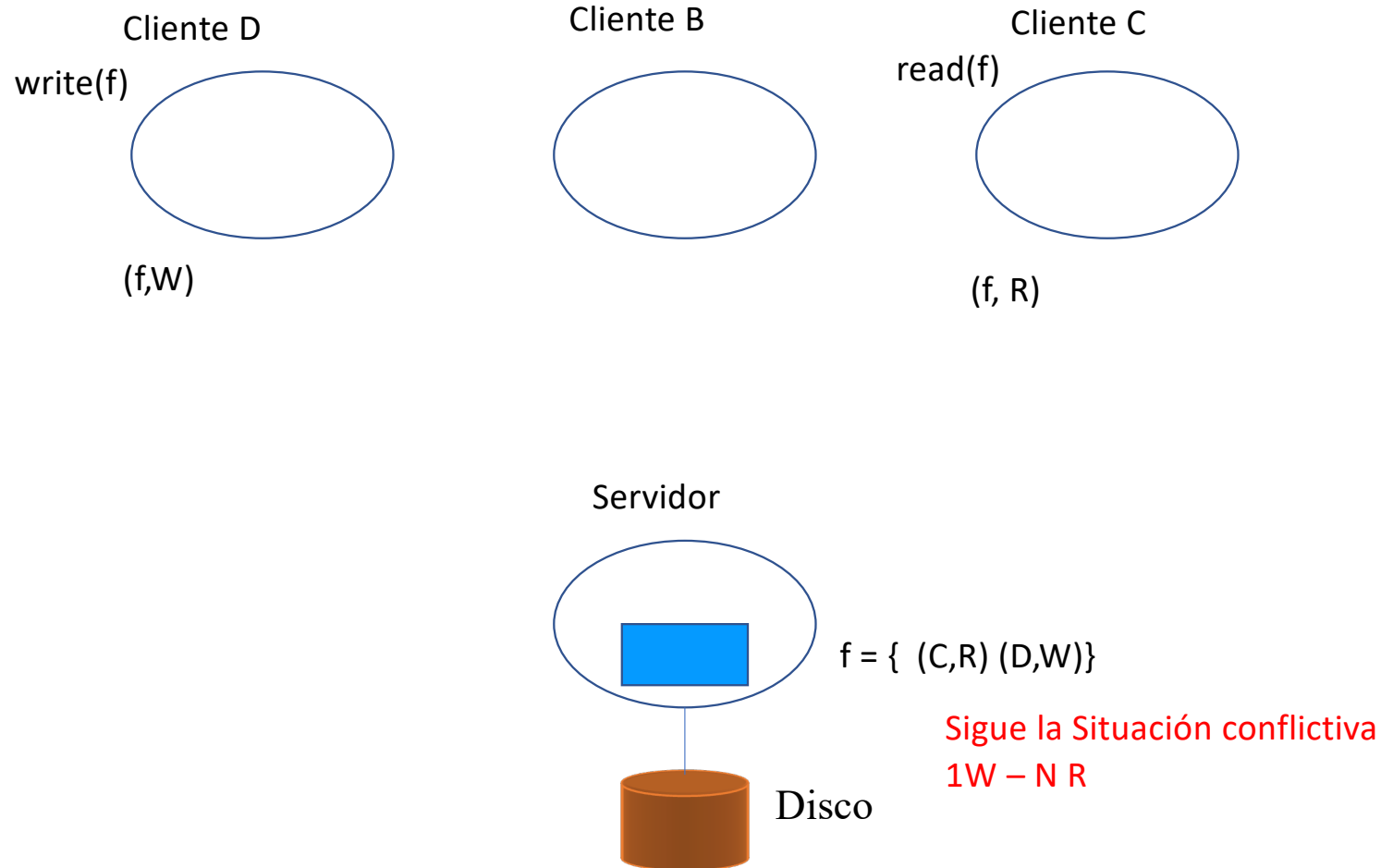
# Coherencia de caché en Sprite



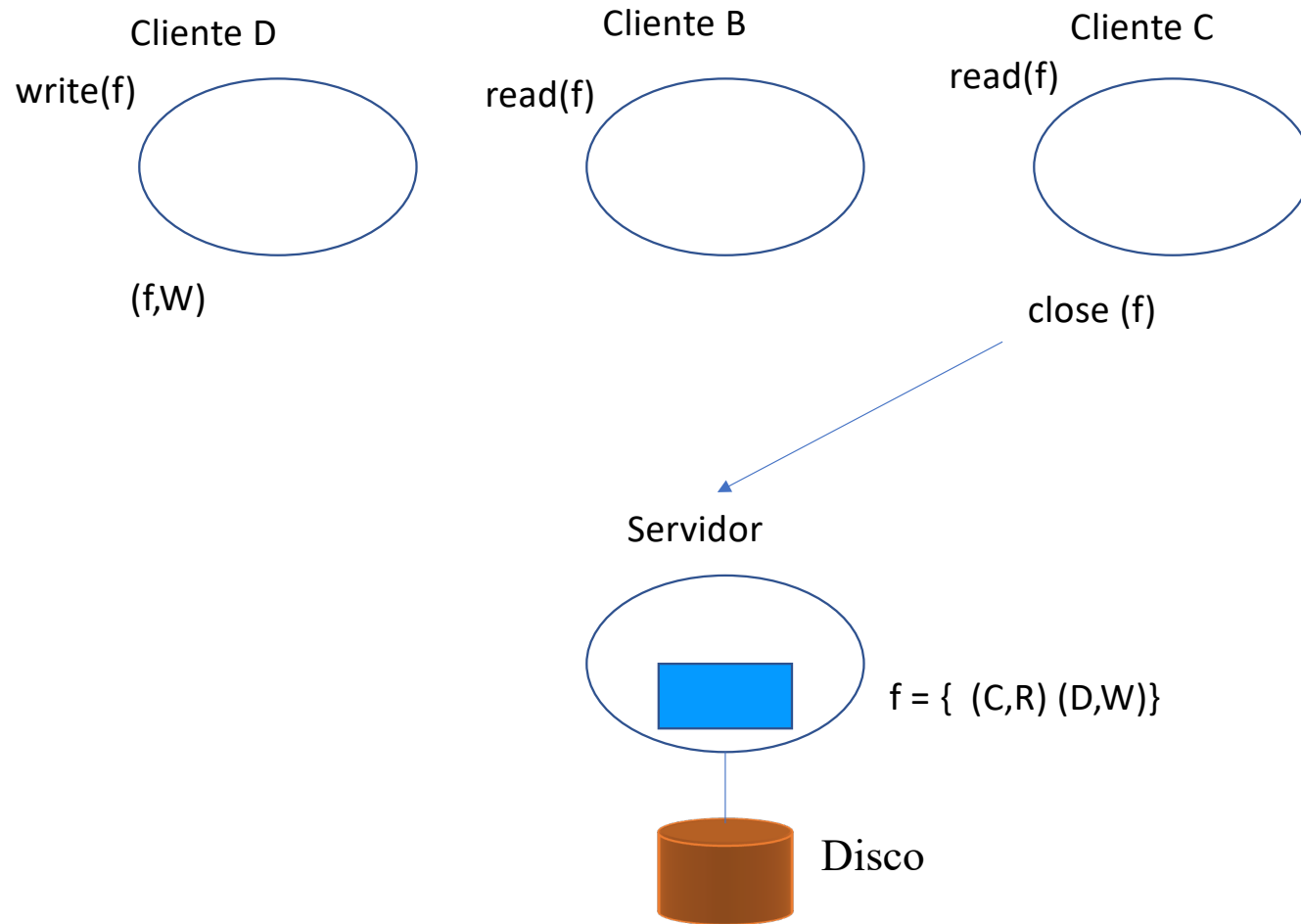
# Coherencia de caché en Sprite



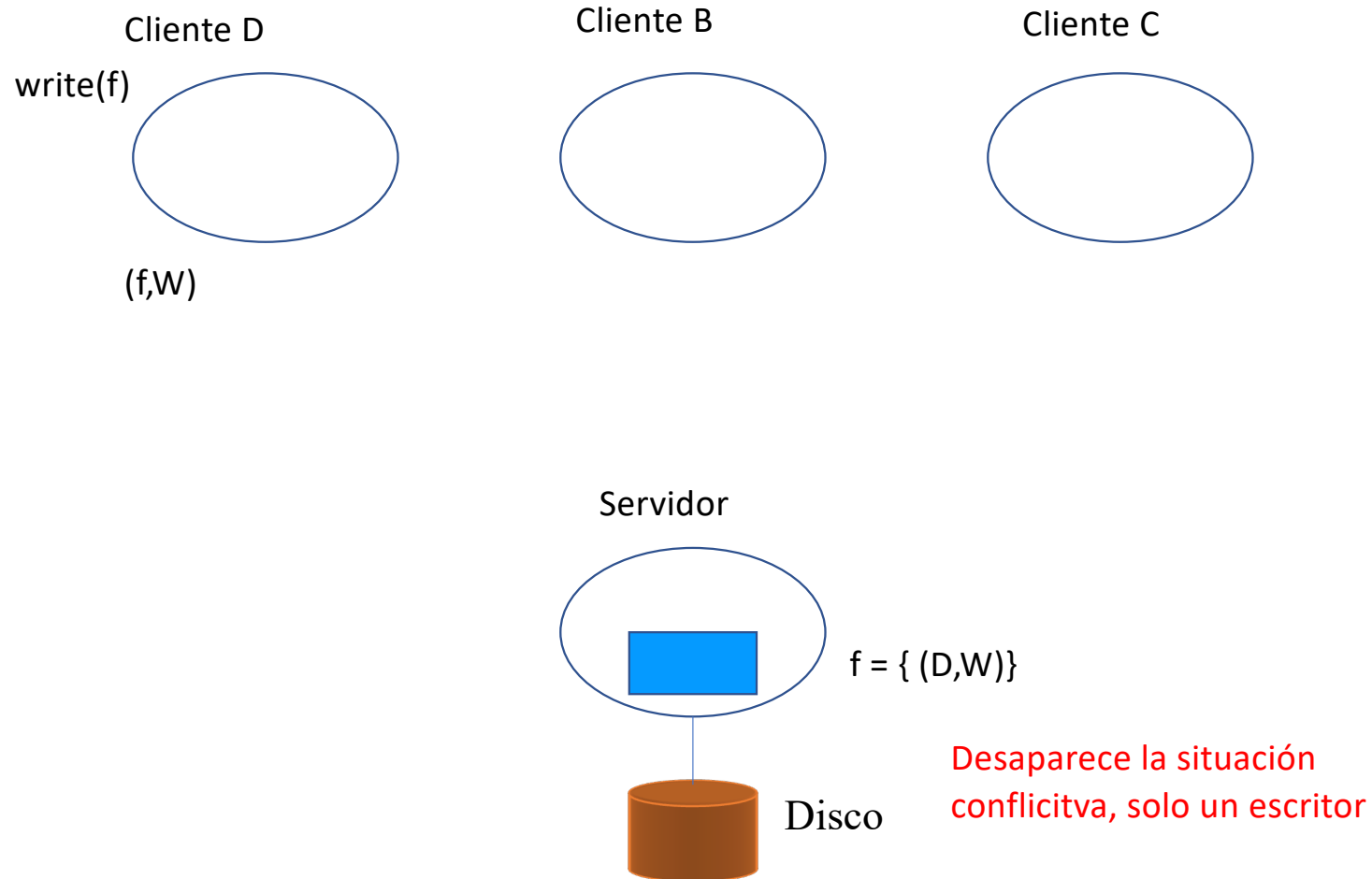
# Coherencia de caché en Sprite



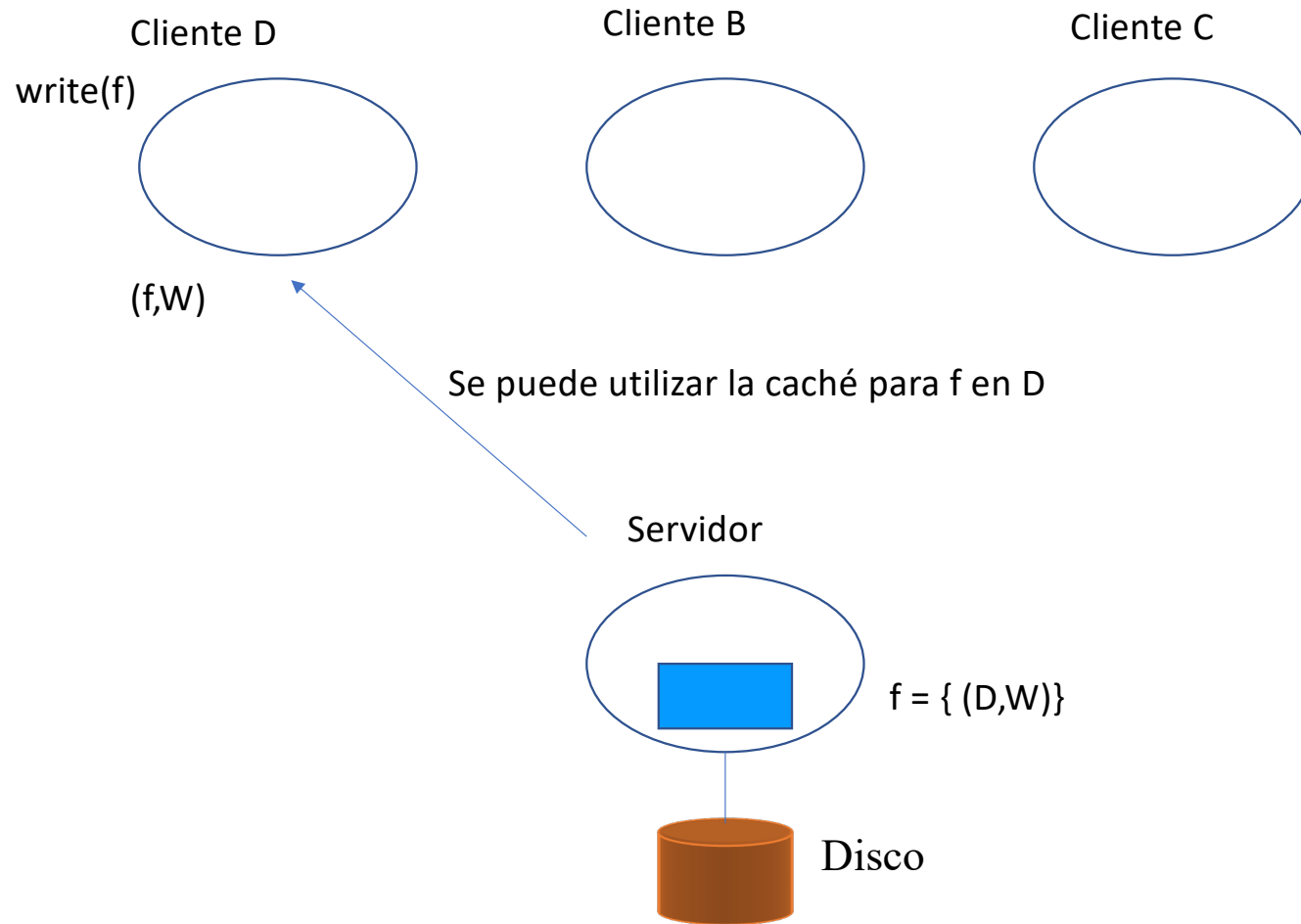
# Coherencia de caché en Sprite



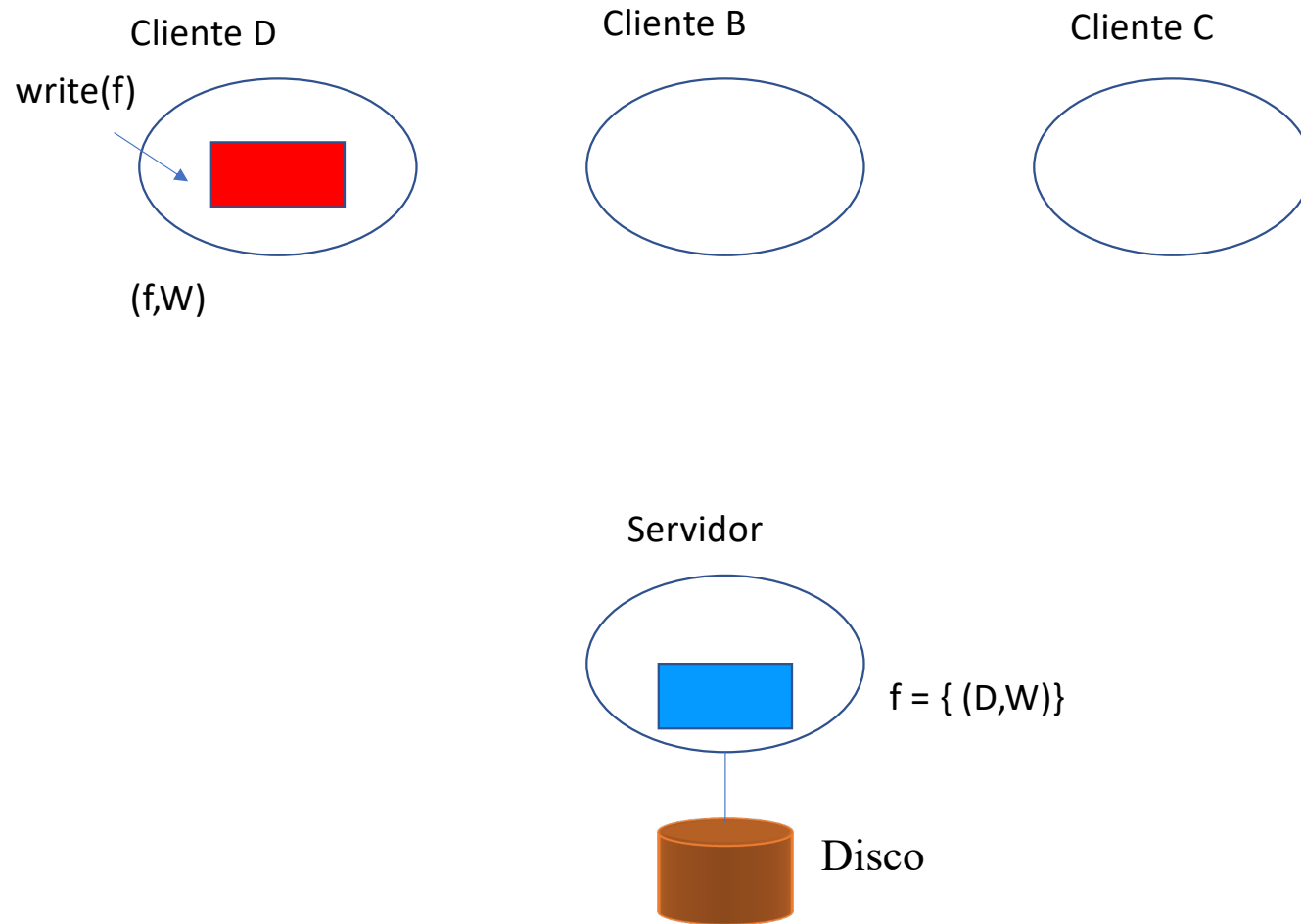
# Coherencia de caché en Sprite



# Coherencia de caché en Sprite

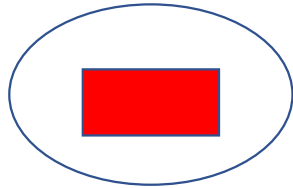


# Coherencia de caché en Sprite

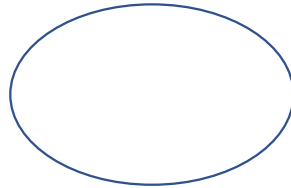


# Coherencia de caché en Sprite

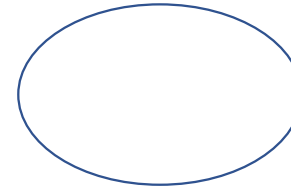
Cliente D



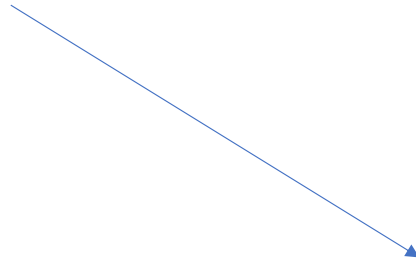
Cliente B



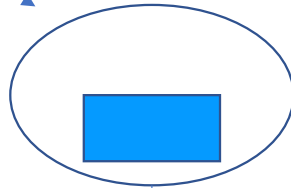
Cliente C



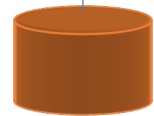
close(f)



Servidor



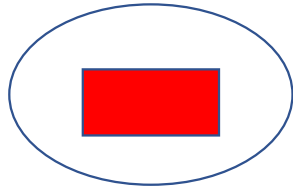
$f = \{ (D,W) \}$



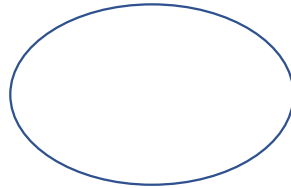
Disco

# Coherencia de caché en Sprite

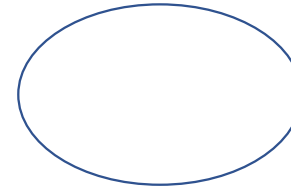
Cliente D



Cliente B

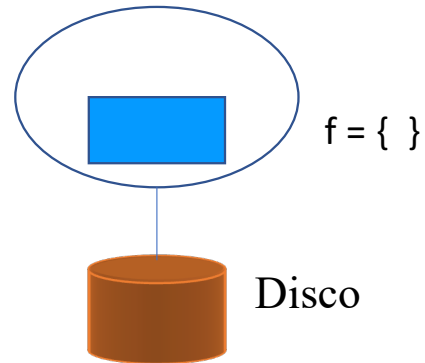


Cliente C



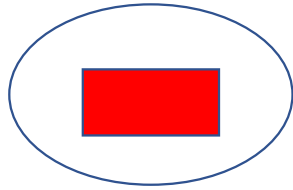
Los bloques se mantienen en la caché, política de escritura diferida  
Estos bloques pueden diferir de los del servidor

Servidor

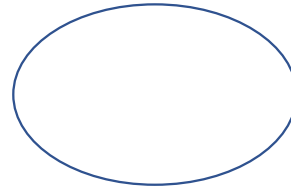


# Coherencia de caché en Sprite

Cliente D



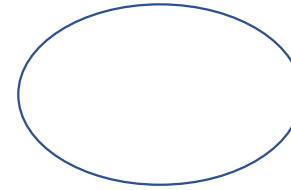
Cliente B



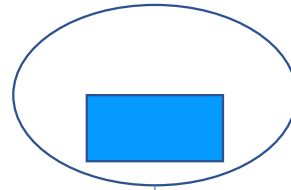
open(f, R)



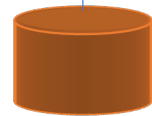
Cliente C



Servidor



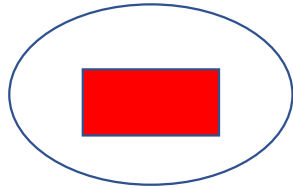
f = { }



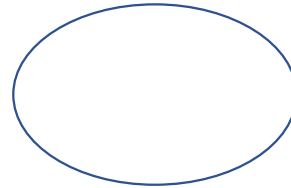
Disco

# Coherencia de caché en Sprite

Cliente D

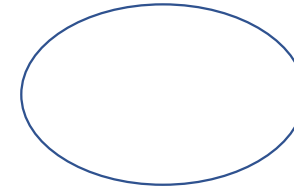


Cliente B

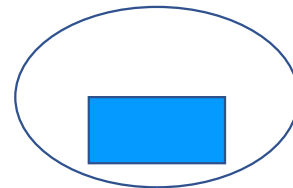


open(f, R)

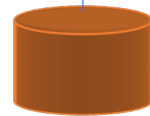
Cliente C



Servidor



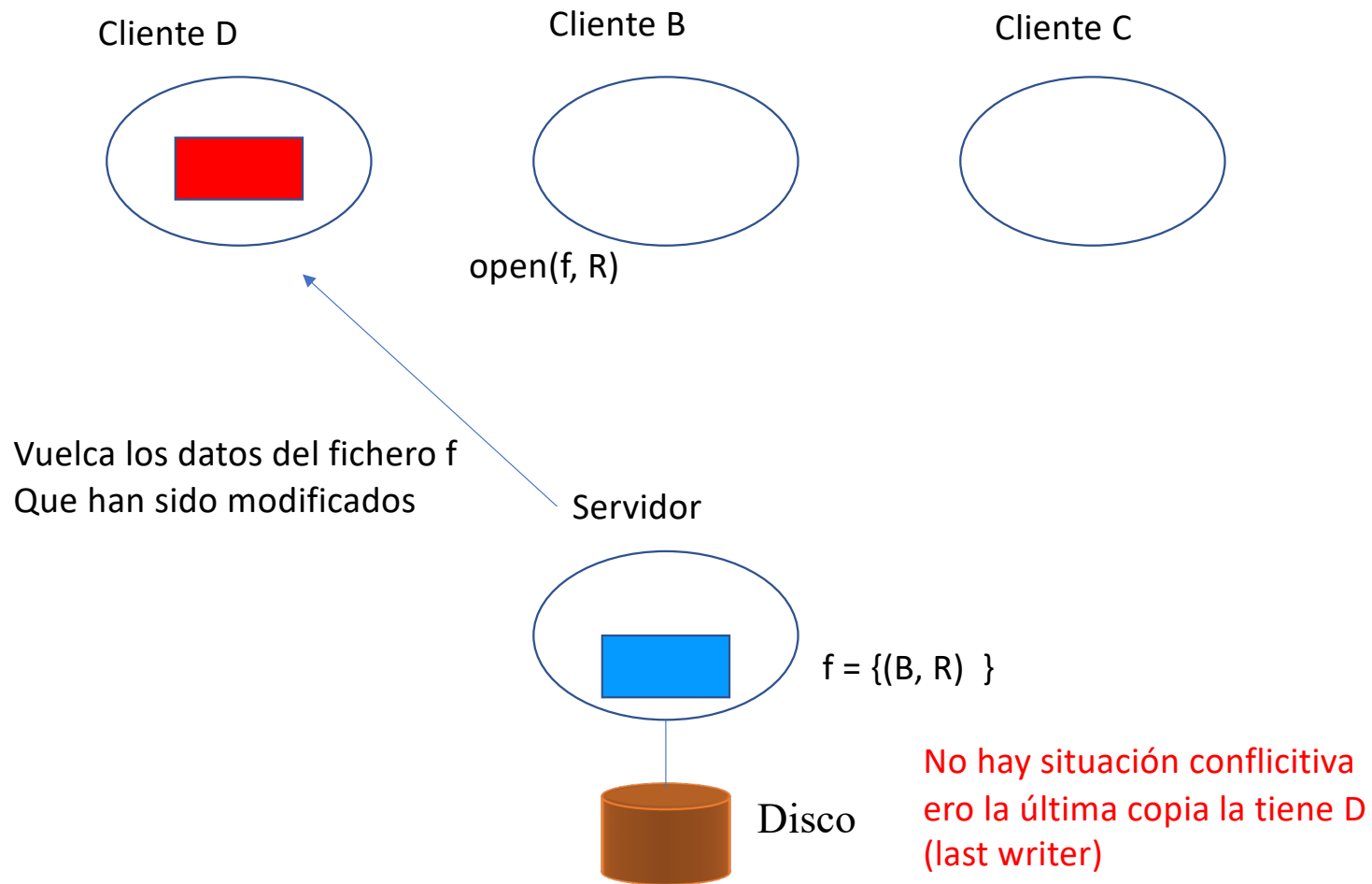
$f = \{(B, R)\}$



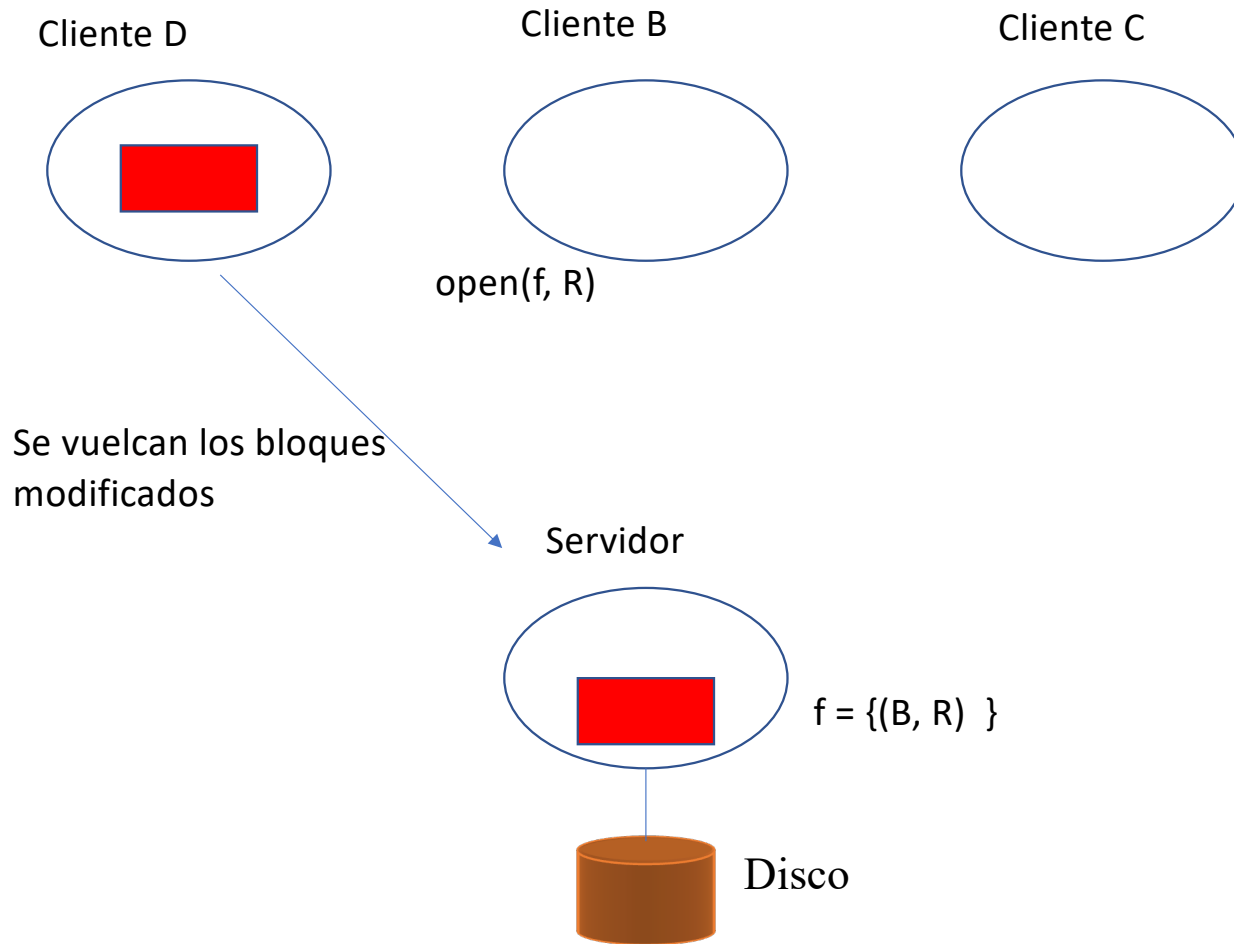
Disco

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

# Coherencia de caché en Sprite

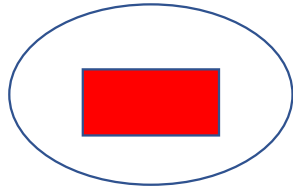


# Coherencia de caché en Sprite

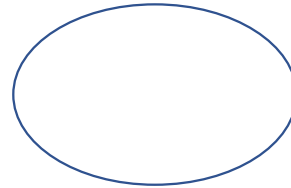


# Coherencia de caché en Sprite

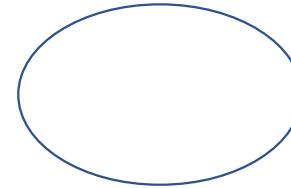
Cliente D



Cliente B



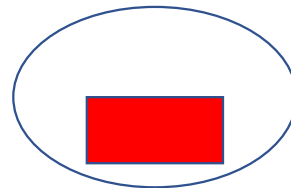
Cliente C



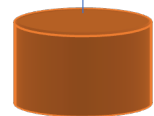
open(f, R)

Se puede utilizar la caché para f

Servidor



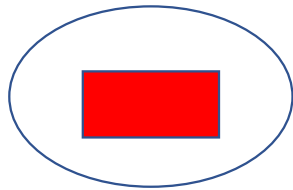
$f = \{(B, R)\}$



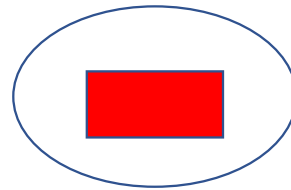
Disco

# Coherencia de caché en Sprite

Cliente D

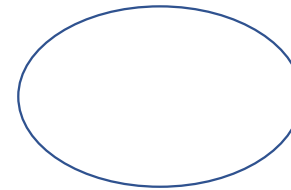


Cliente B

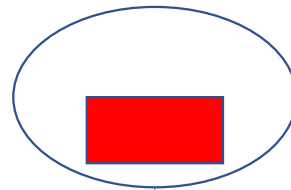


(f, R)

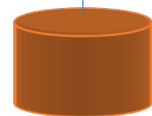
Cliente C



Servidor

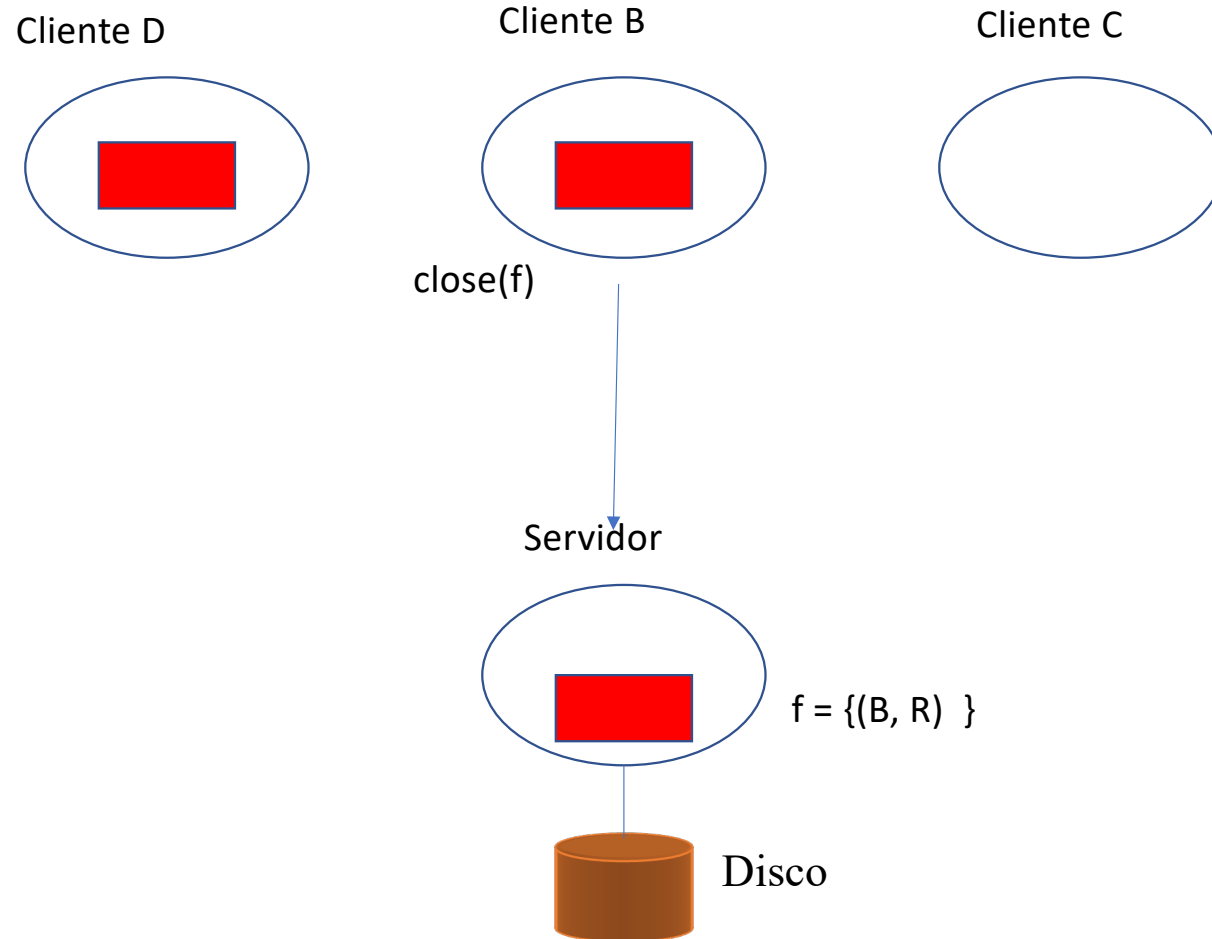


$f = \{(B, R)\}$



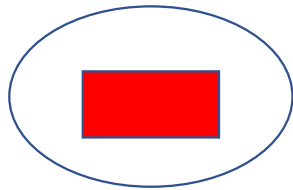
Disco

# Coherencia de caché en Sprite

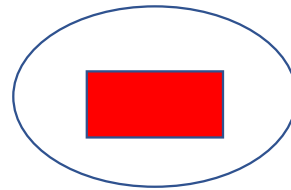


# Coherencia de caché en Sprite

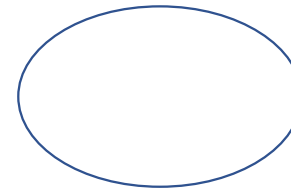
Cliente D



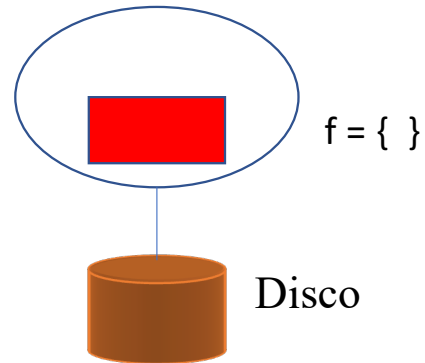
Cliente B



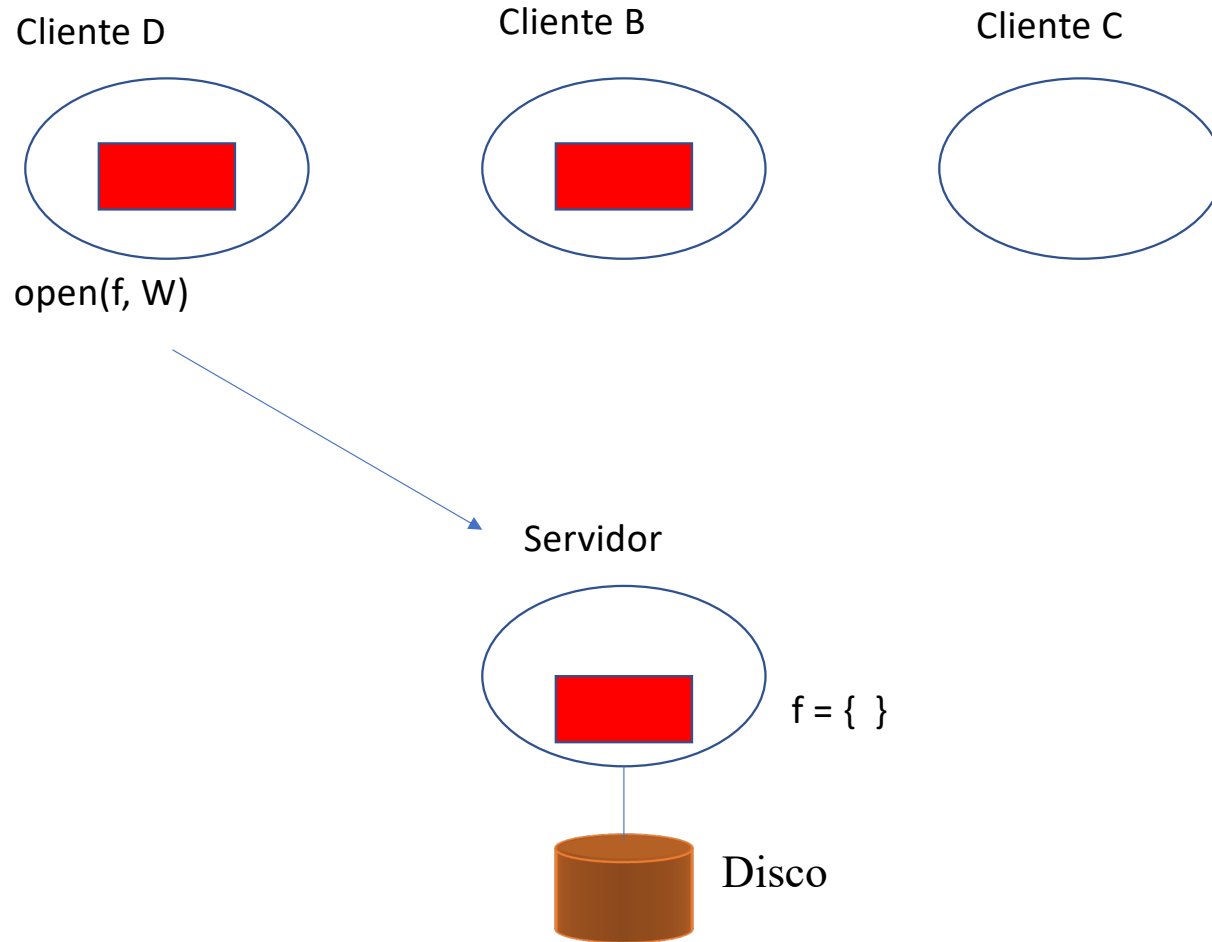
Cliente C



Servidor

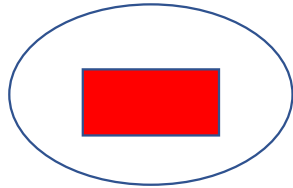


# Coherencia de caché en Sprite



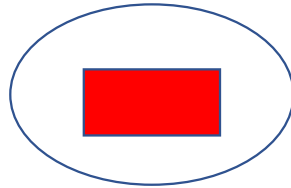
# Coherencia de caché en Sprite

Cliente D

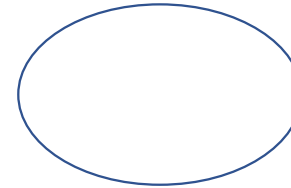


(f, W)

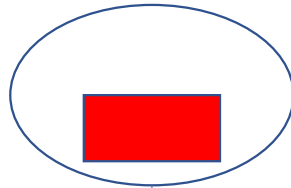
Cliente B



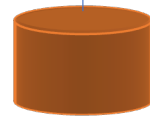
Cliente C



Servidor

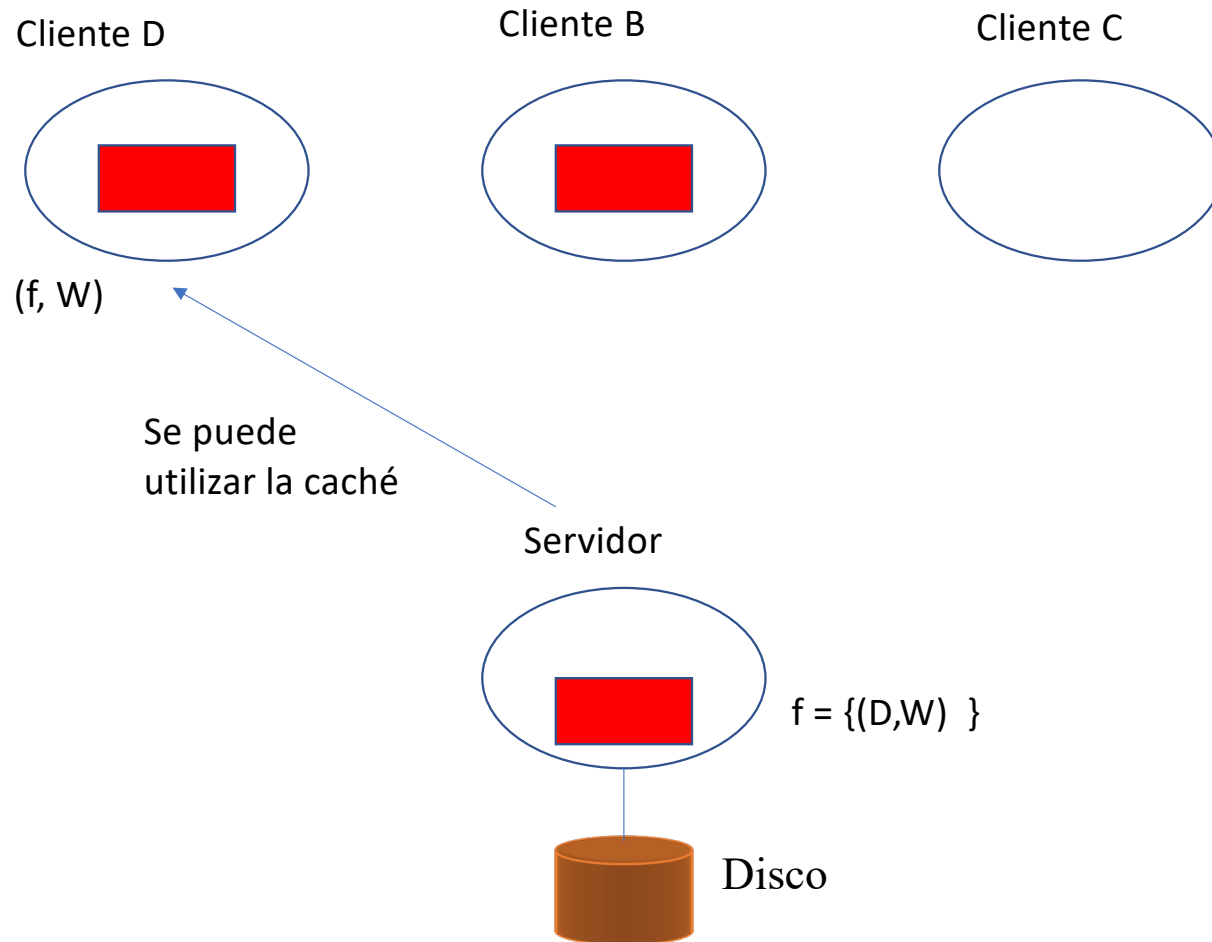


$f = \{(D,W)\}$

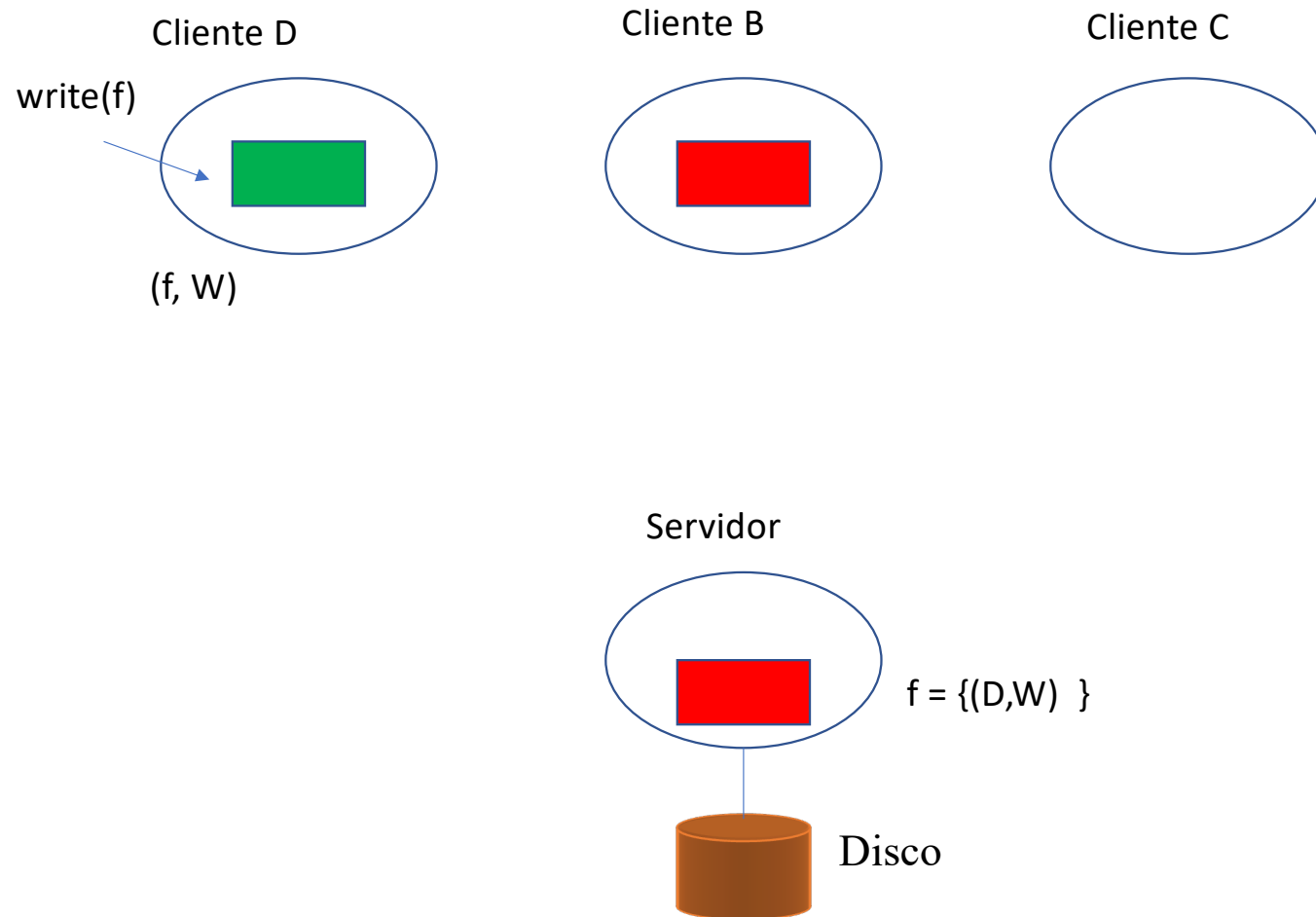


Disco

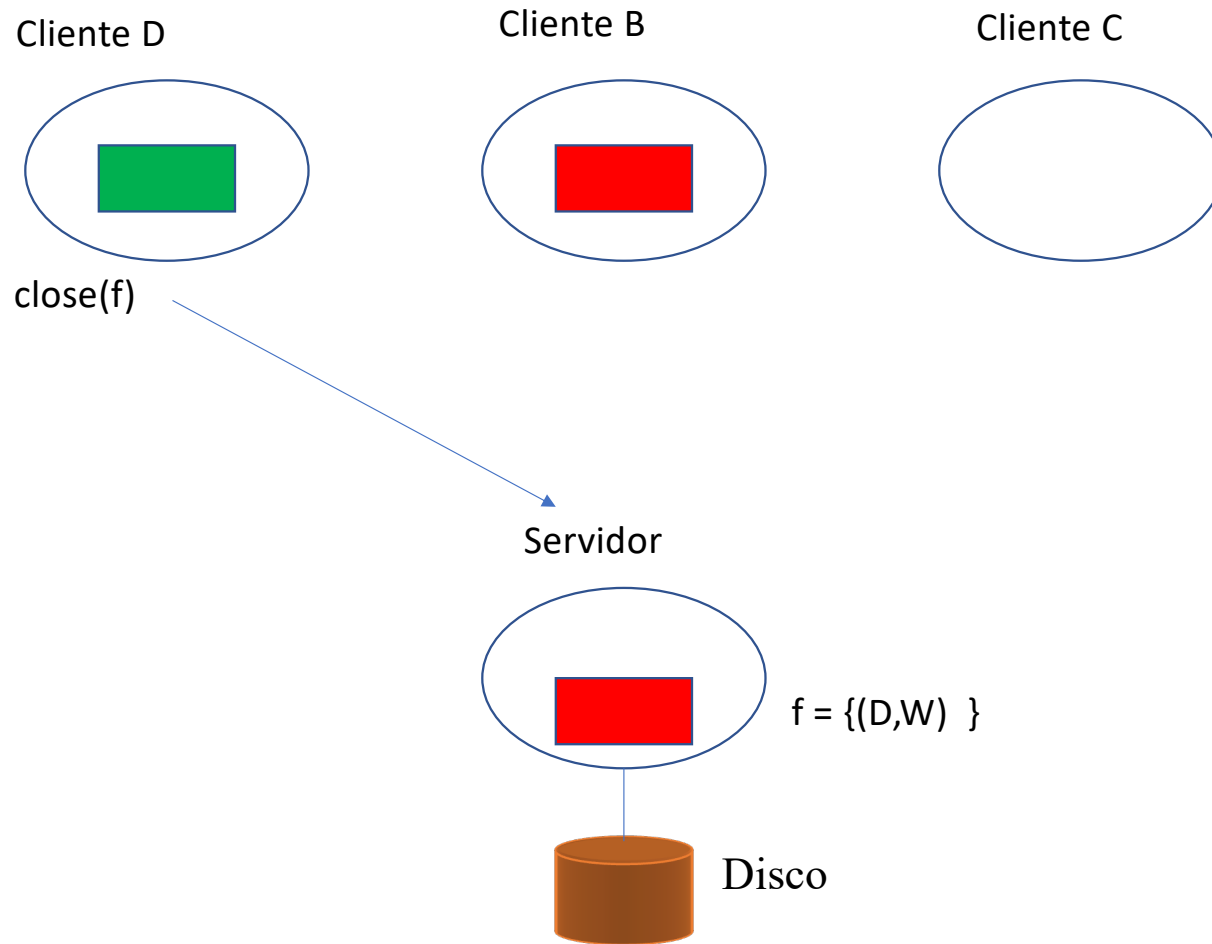
# Coherencia de caché en Sprite



# Coherencia de caché en Sprite

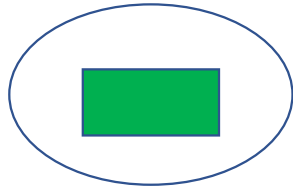


# Coherencia de caché en Sprite

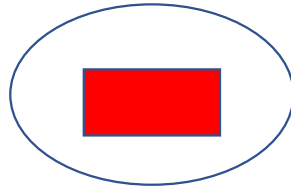


# Coherencia de caché en Sprite

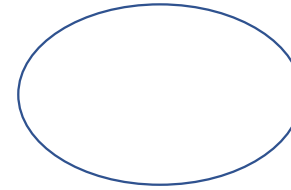
Cliente D



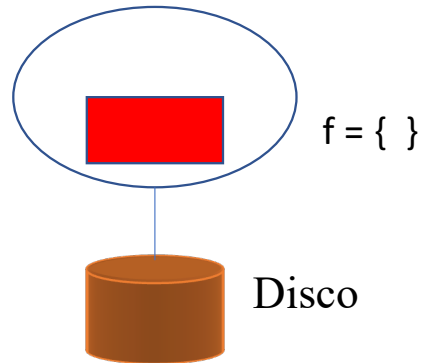
Cliente B



Cliente C

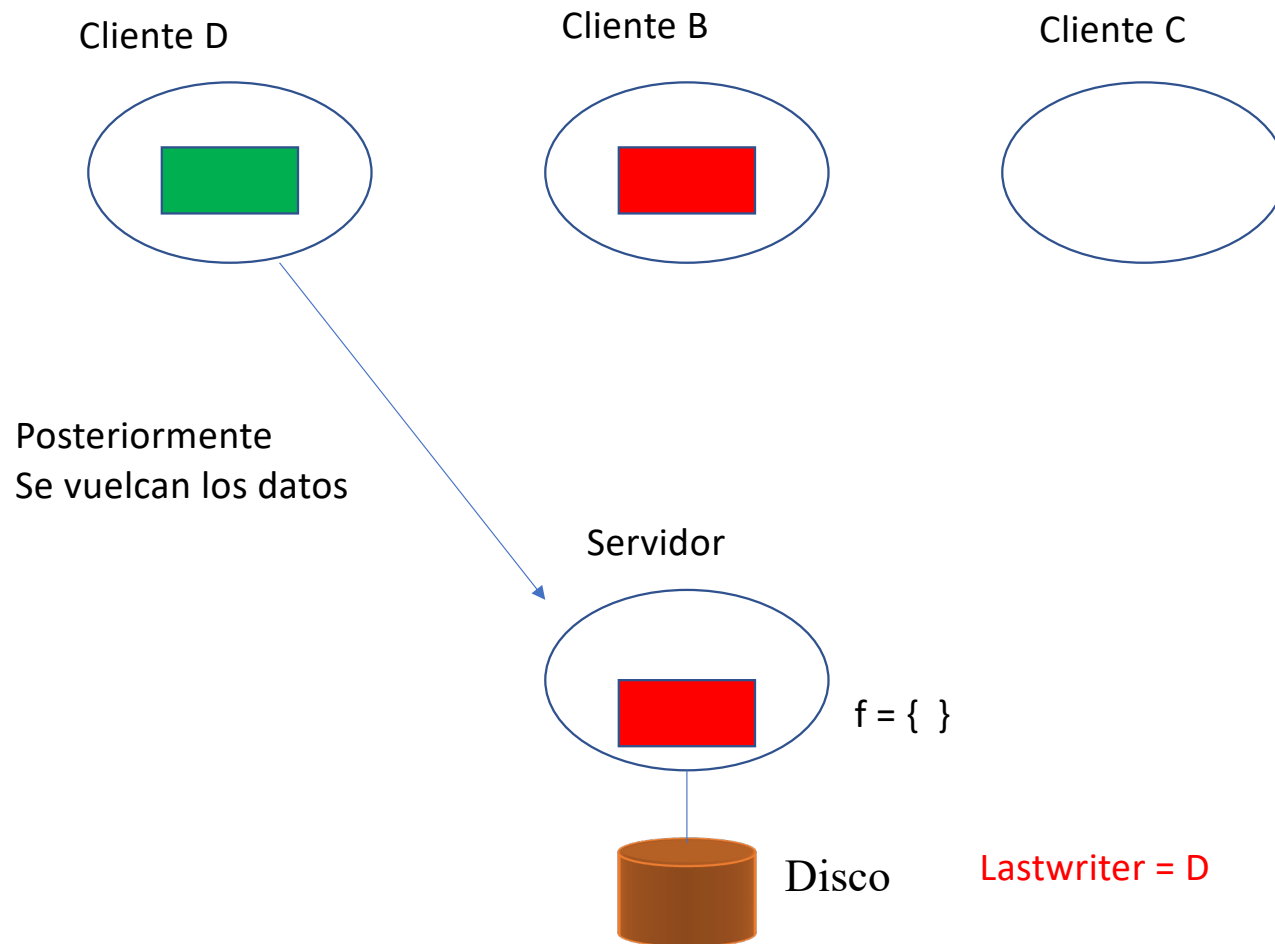


Servidor

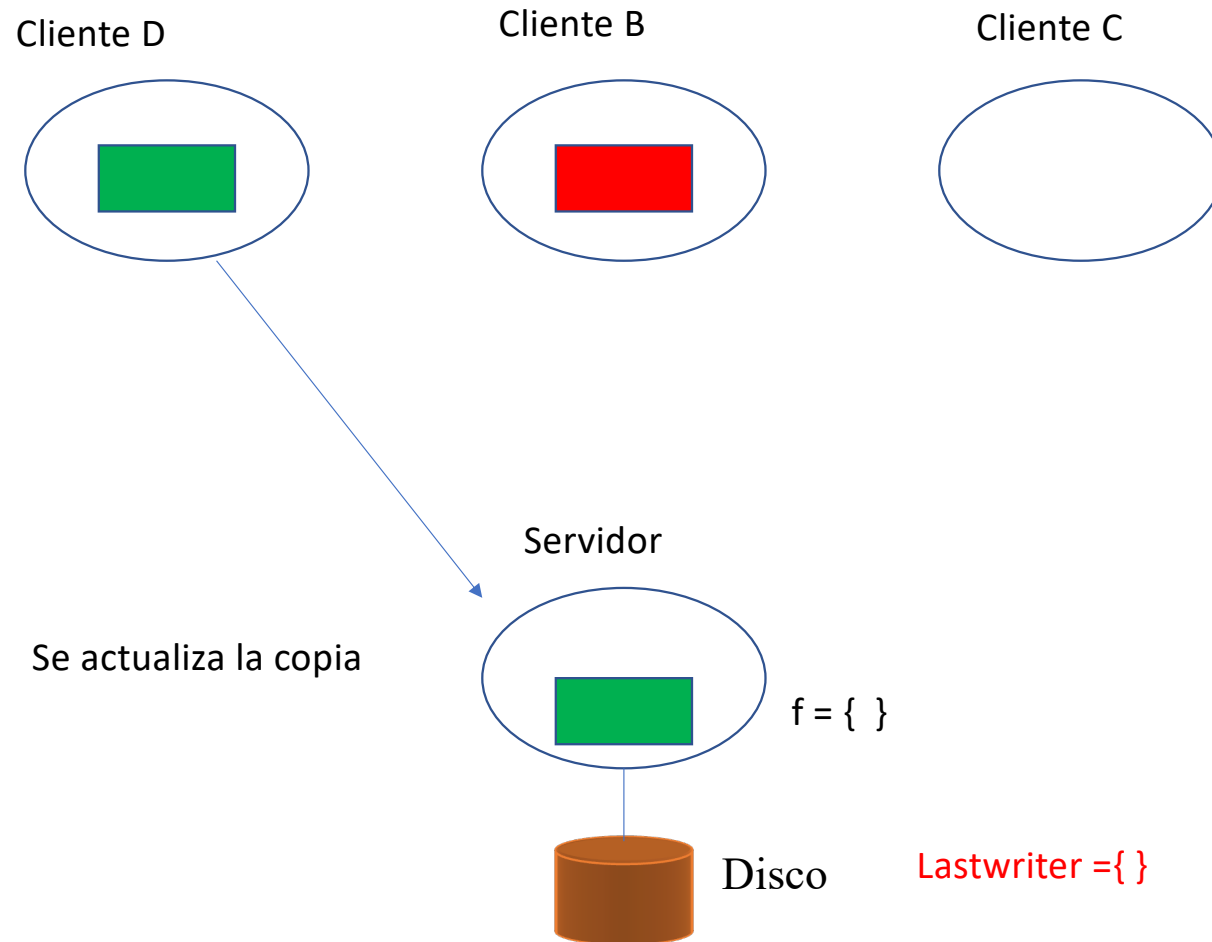


Lastwriter = D

# Coherencia de caché en Sprite

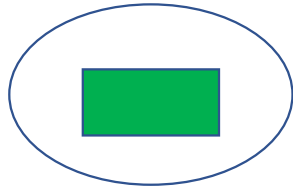


# Coherencia de caché en Sprite

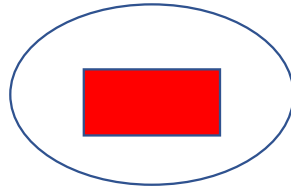


# Coherencia de caché en Sprite

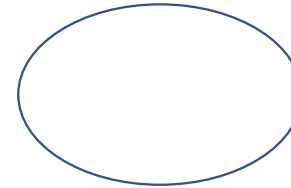
Cliente D



Cliente B



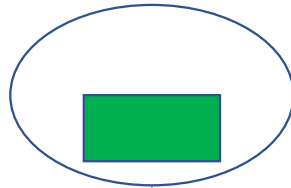
Cliente C



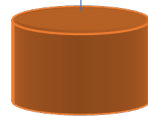
open(f, R)



Servidor



f = { }

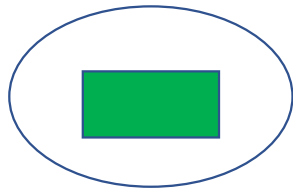


Disco

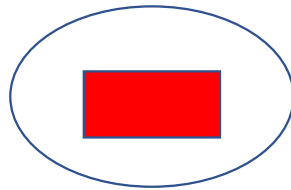
Lastwriter = { }

# Coherencia de caché en Sprite

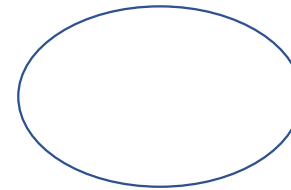
Cliente D



Cliente B



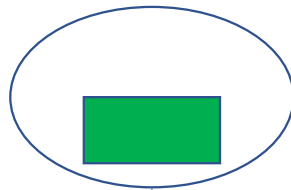
Cliente C



open(f, R)

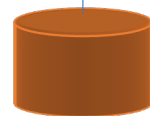


Servidor



$f = \{(B,R)\}$

No hay conflictos

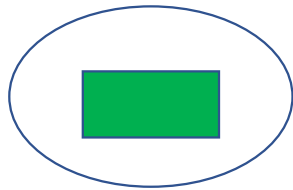


Disco

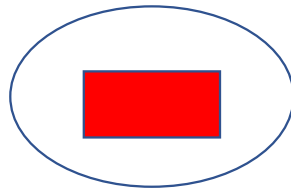
Lastwriter = { }

# Coherencia de caché en Sprite

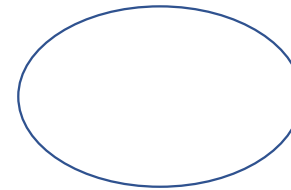
Cliente D



Cliente B



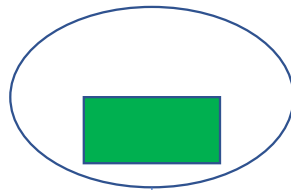
Cliente C



open(f, R)

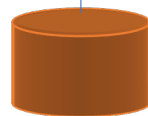
¿se puede usar la caché para f?

Servidor



$f = \{(B,R)\}$

No hay conflictos

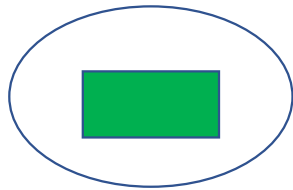


Disco

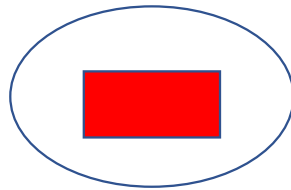
Lastwriter = { }

# Coherencia de caché en Sprite

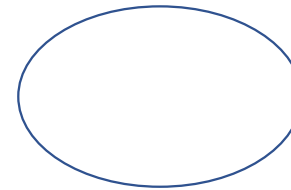
Cliente D



Cliente B



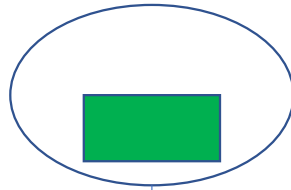
Cliente C



open(f, R)

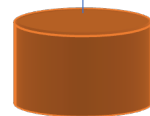
¿cómo se resuelve?

Servidor



$f = \{(B,R)\}$

No hay conflictos

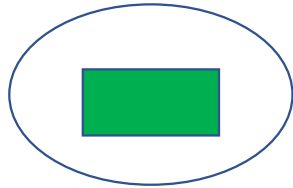


Disco

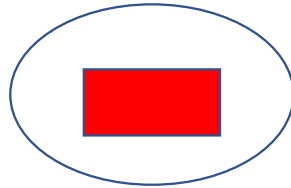
Lastwriter = { }

# Coherencia de caché en Sprite

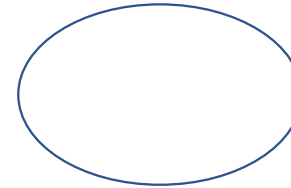
Cliente D



Cliente B

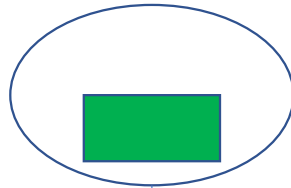


Cliente C



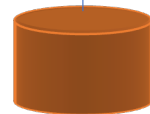
open(f, R)

Servidor



$f = \{(B,R)\}$

No hay conflictos

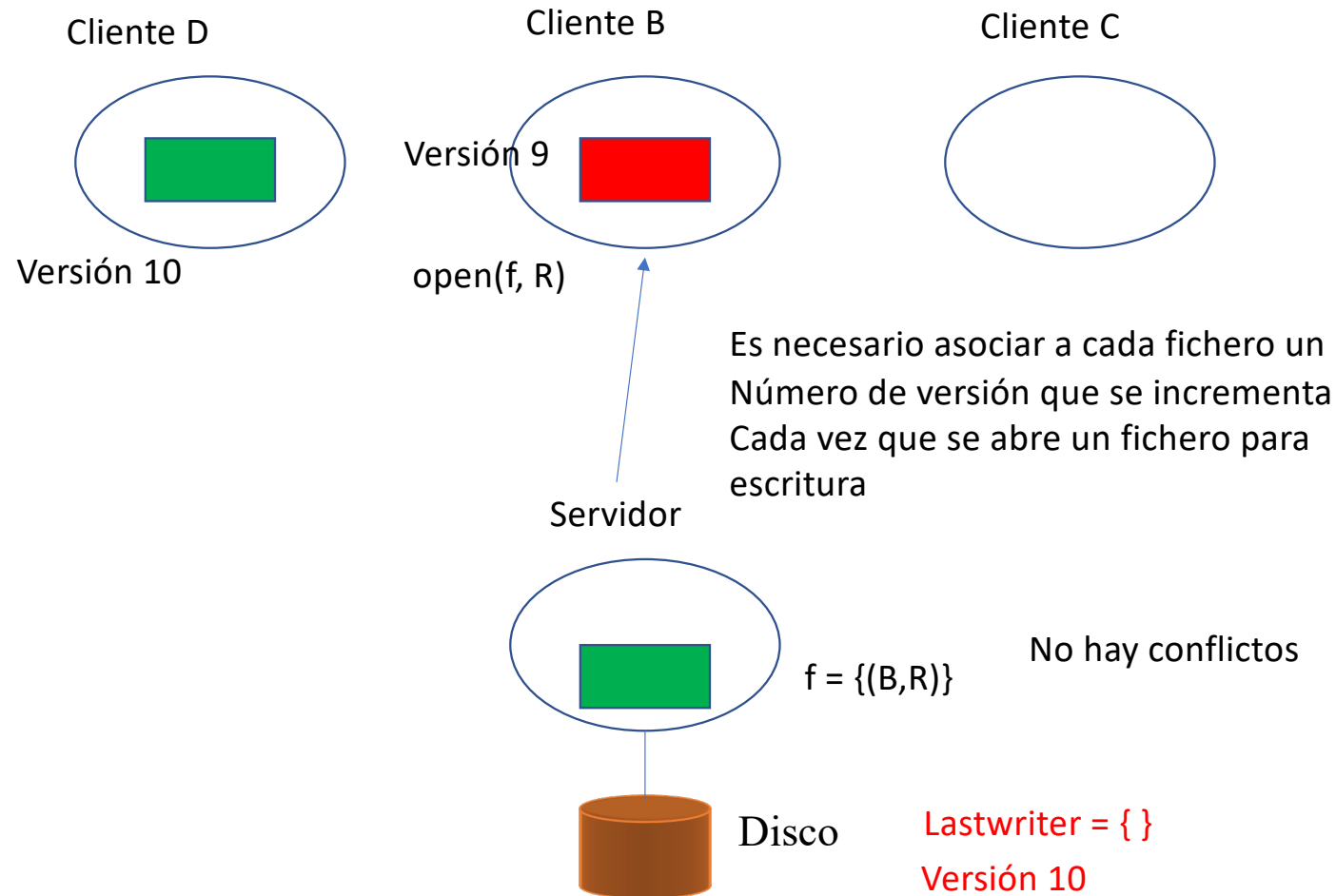


Disco

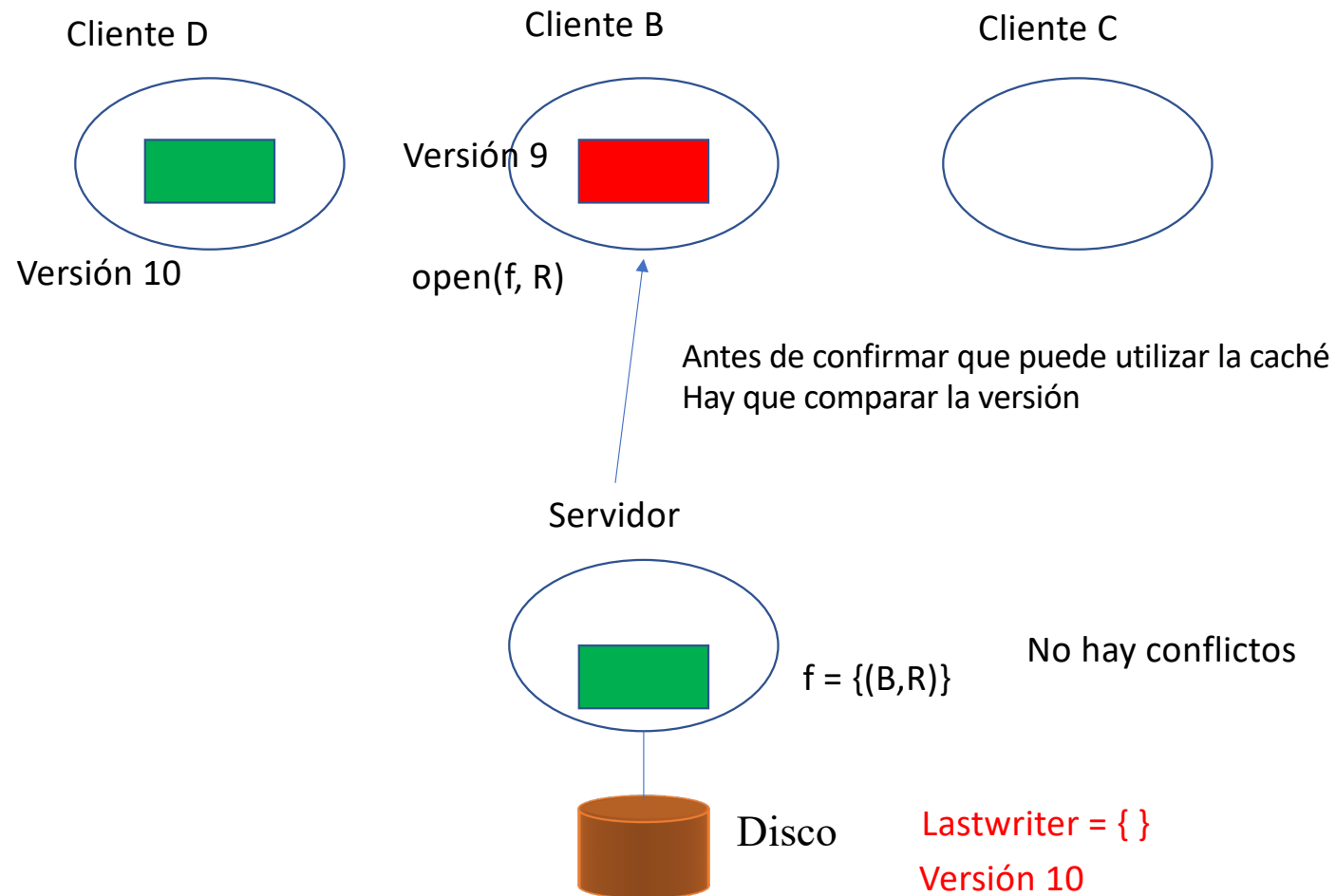
Lastwriter = { }

Es necesario asociar a cada fichero un Número de versión que se incrementa Cada vez que se abre un fichero para escritura

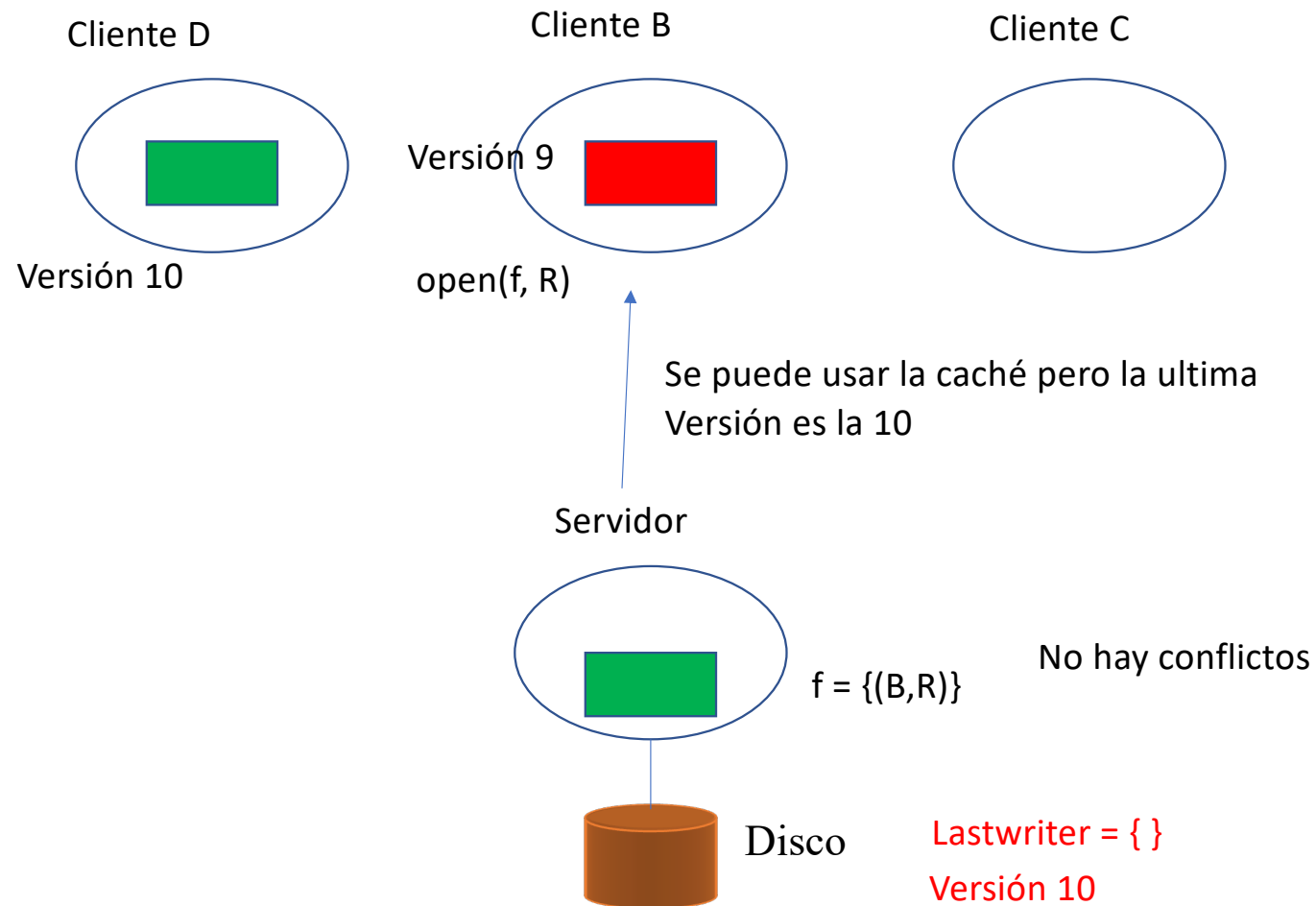
# Coherencia de caché en Sprite



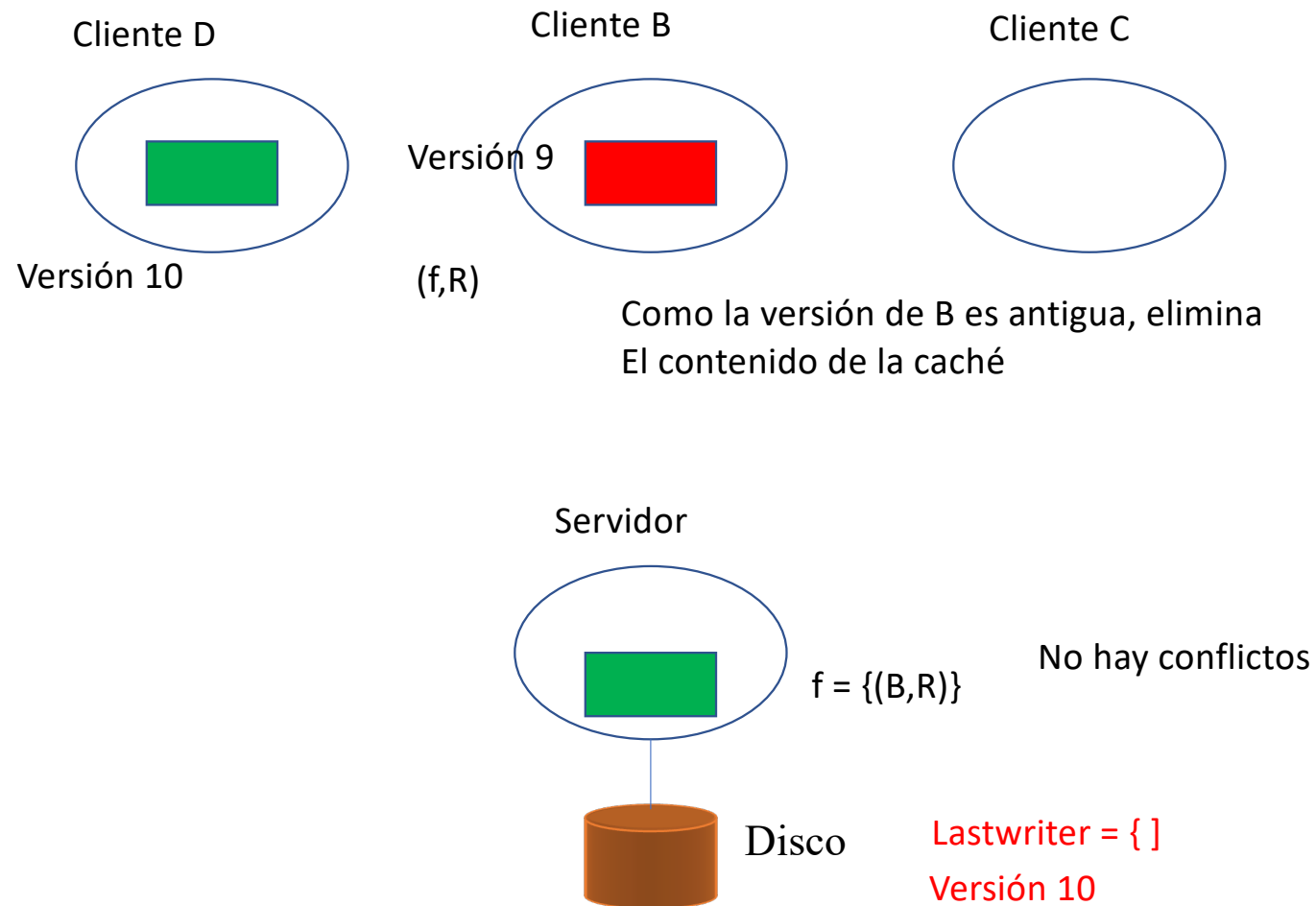
# Coherencia de caché en Sprite



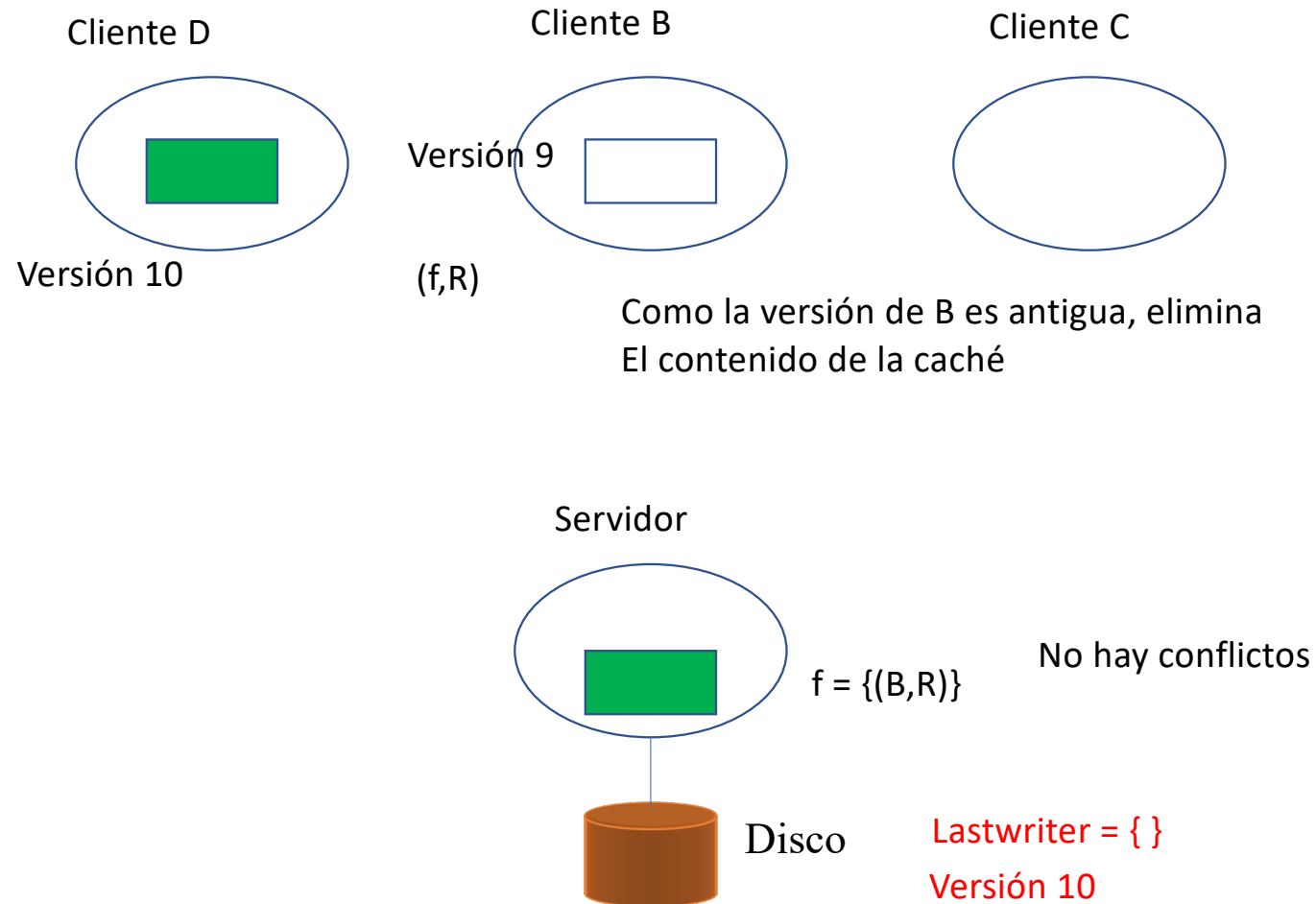
# Coherencia de caché en Sprite



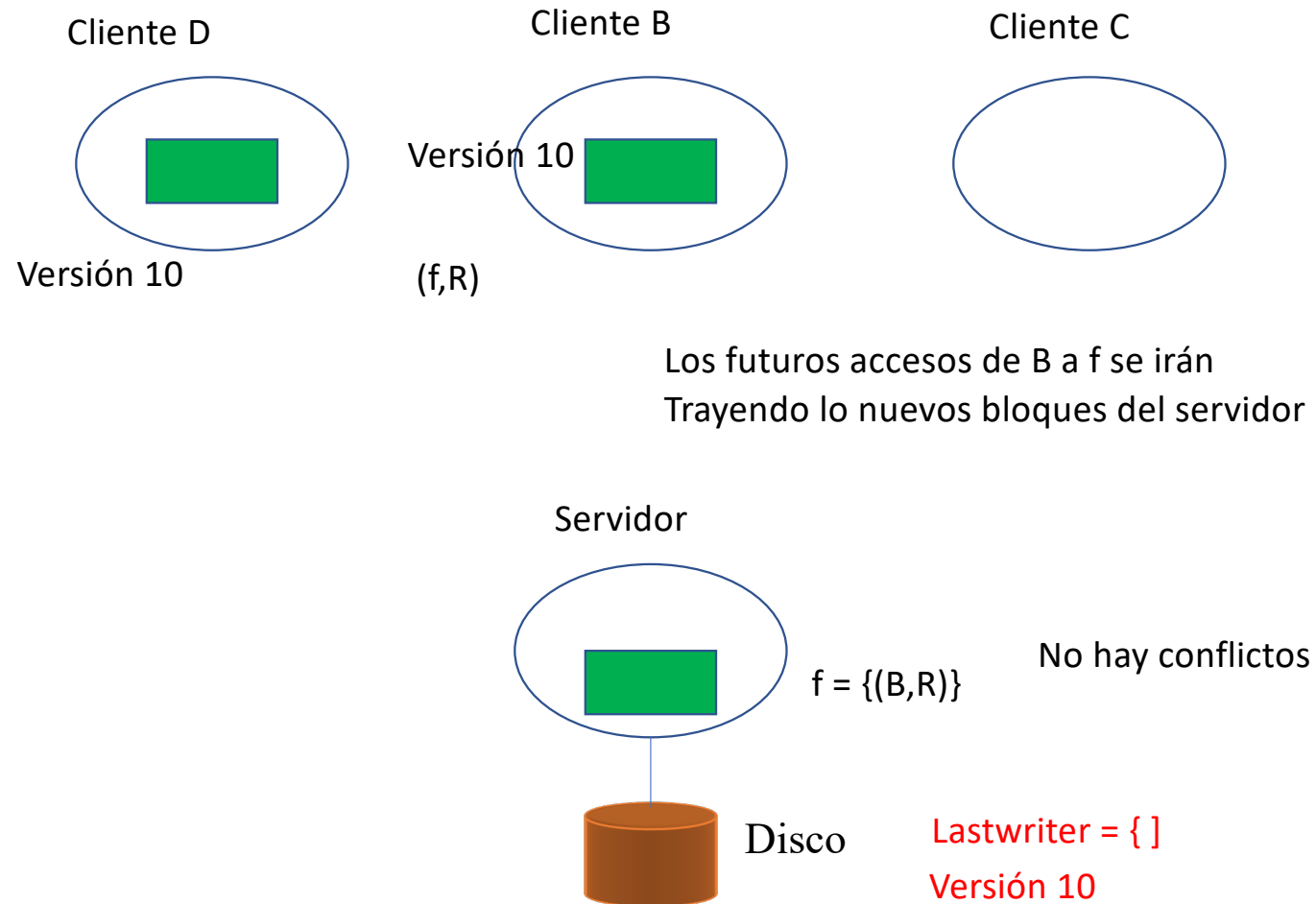
# Coherencia de caché en Sprite



# Coherencia de caché en Sprite



# Coherencia de caché en Sprite



# Ejemplo de protocolo de coherencia Sprite

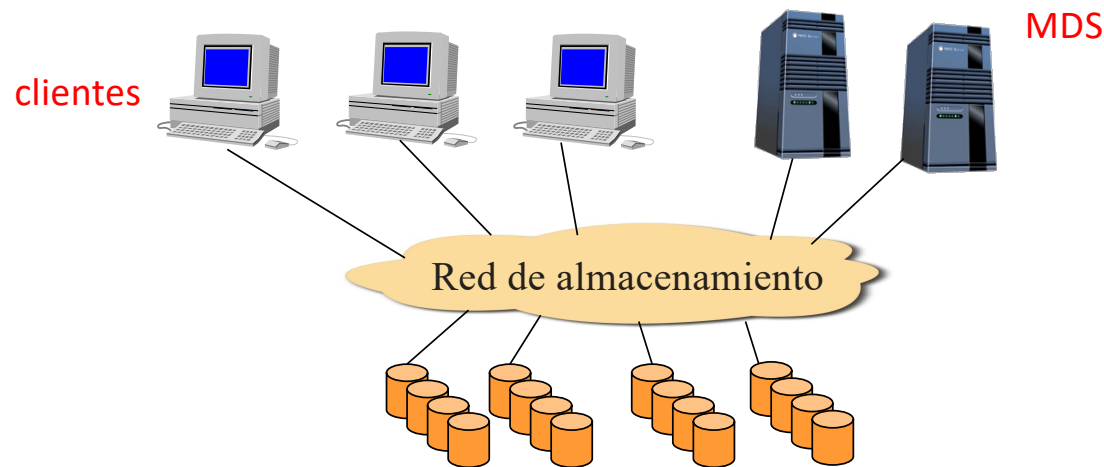
- Si acceso concurrente conflictivo (1 escritor + otro(s) cliente(s))
  - Se anula cache y se usa acceso remoto en nodos implicados
- En `open` el cliente siempre contacta con servidor especificando:
  - Modo de acceso (R, W)
  - Número de versión de la copia en cache del fichero (si la hay)
- En `close` no se vuelca el fichero
- Tratamiento del `open` en el servidor
  - Si la copia del fichero en el servidor no está actualizada
    - Solicita al último cliente (*last-writer*) que vuelque datos del fichero
  - Si no hay conflicto de acceso:
    - Si la versión del cliente es más antigua, se le indica en mensaje de respuesta que la invalide

# Ejemplo de protocolo de coherencia Sprite (cont.)

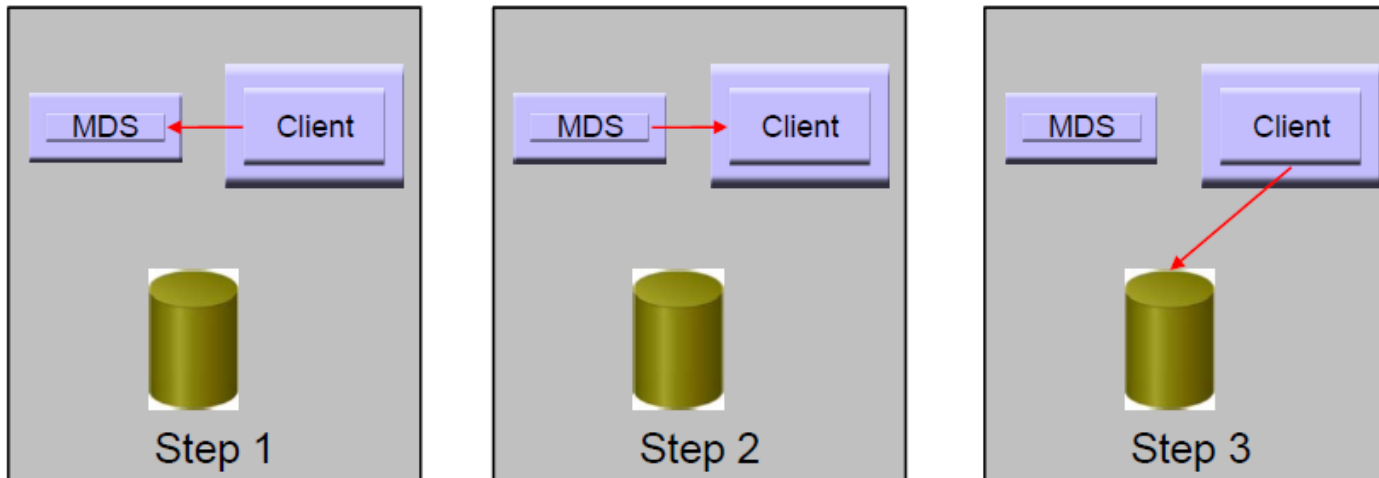
- Tratamiento del `open` en el servidor (cont.)
  - Si la petición produce un conflicto de acceso:
    - Se le envía a los clientes con el fichero abierto una orden de invalidación y desactivación de la cache para ese fichero
      - Si era un escritor se le pide un volcado previo
    - Se le envía en la respuesta al cliente una petición para invalidar y desactivar la cache para ese fichero
  - Si la petición se encuentra que ya hay conflicto
    - Se le envía en la respuesta una petición de invalidar y desactivar la cache para ese fichero

# 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



SNIA

# Ejemplos

- OCFS (Oracle Cluster file System)
- GPFS
- GFS
- GlusterFS
- Veristas File System

# Paralelismo en el acceso a los datos

# Paralelismo en el acceso a los datos

Dispositivos en paralelo



# Paralelismo en el acceso a los datos

Sistemas de ficheros paralelos

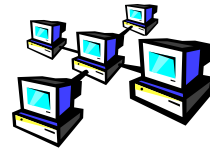


Dispositivos en paralelo



# Paralelismo en el acceso a los datos

Computadores paralelos



Sistemas de ficheros paralelos



Dispositivos en paralelo

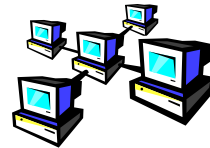


# Paralelismo en el acceso a los datos

Aplicaciones paralelas



Computadores paralelos



Sistemas de ficheros paralelos



Dispositivos en paralelo



# Paralelismo en el acceso a los datos

Aplicaciones paralelas



Computadores paralelos



Sistemas de ficheros paralelos



Dispositivos en paralelo



- Explotar el paralelismo en múltiples niveles

# Paralelismo en el acceso a los datos

Aplicaciones paralelas



Computadores paralelos



**Sistemas de ficheros paralelos (SFP)**



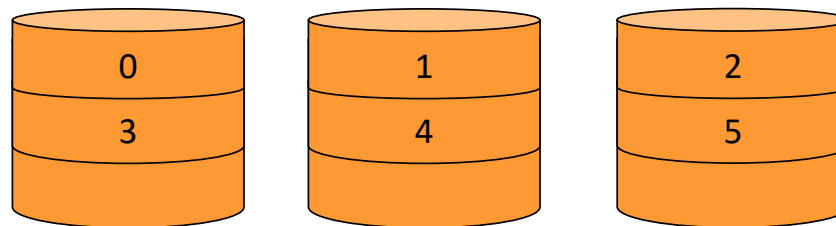
Dispositivos en paralelo



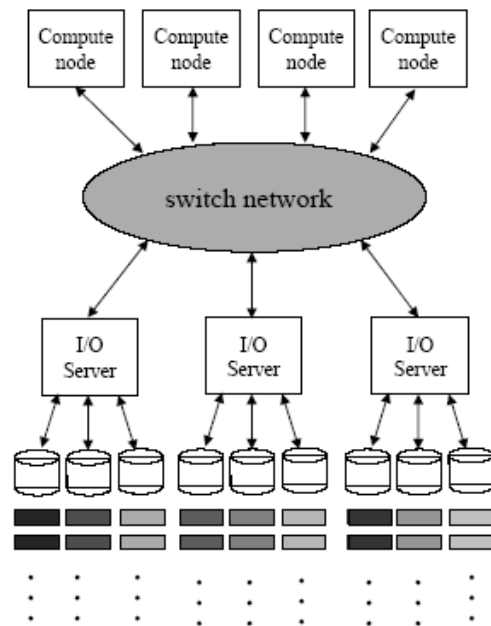
- Explotar el paralelismo en múltiples niveles

# Discos en paralelo: *disk striping*

- Los datos se dividen en bloques y se almacenan en discos diferentes
- RAID 0 (*Redundant Array of Inexpensive Disk 0*)
- Mejora el rendimiento
  - Distribuye la carga de E/S entre diferentes discos

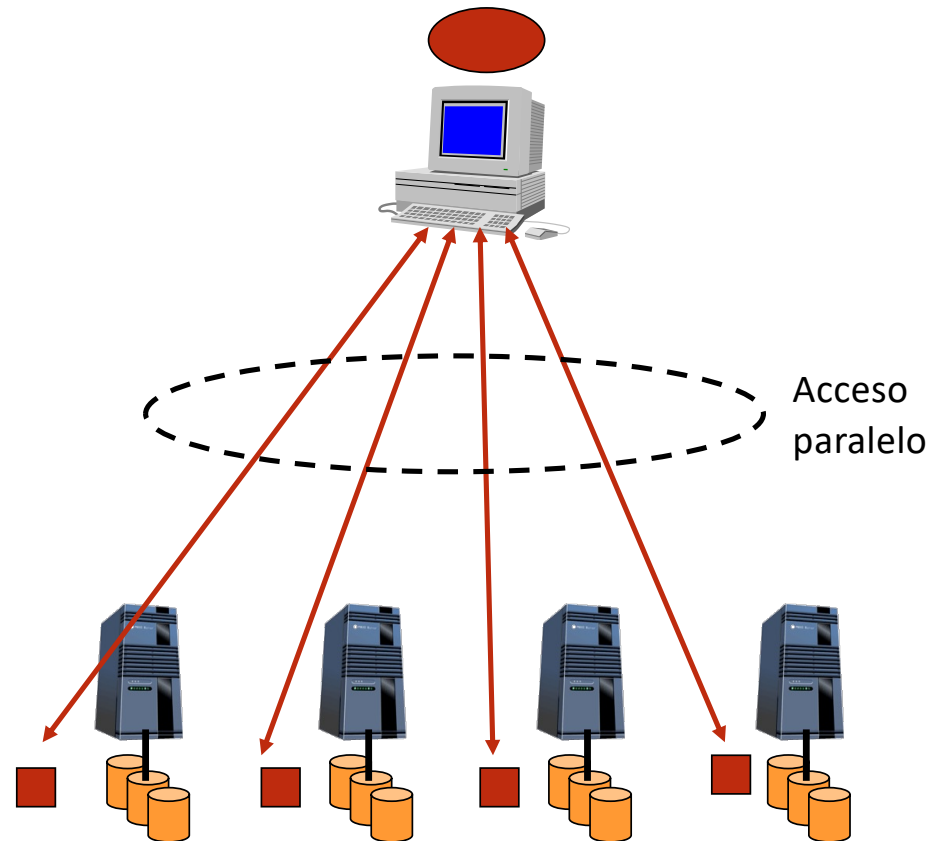


# Sistemas de ficheros paralelos(PFS)

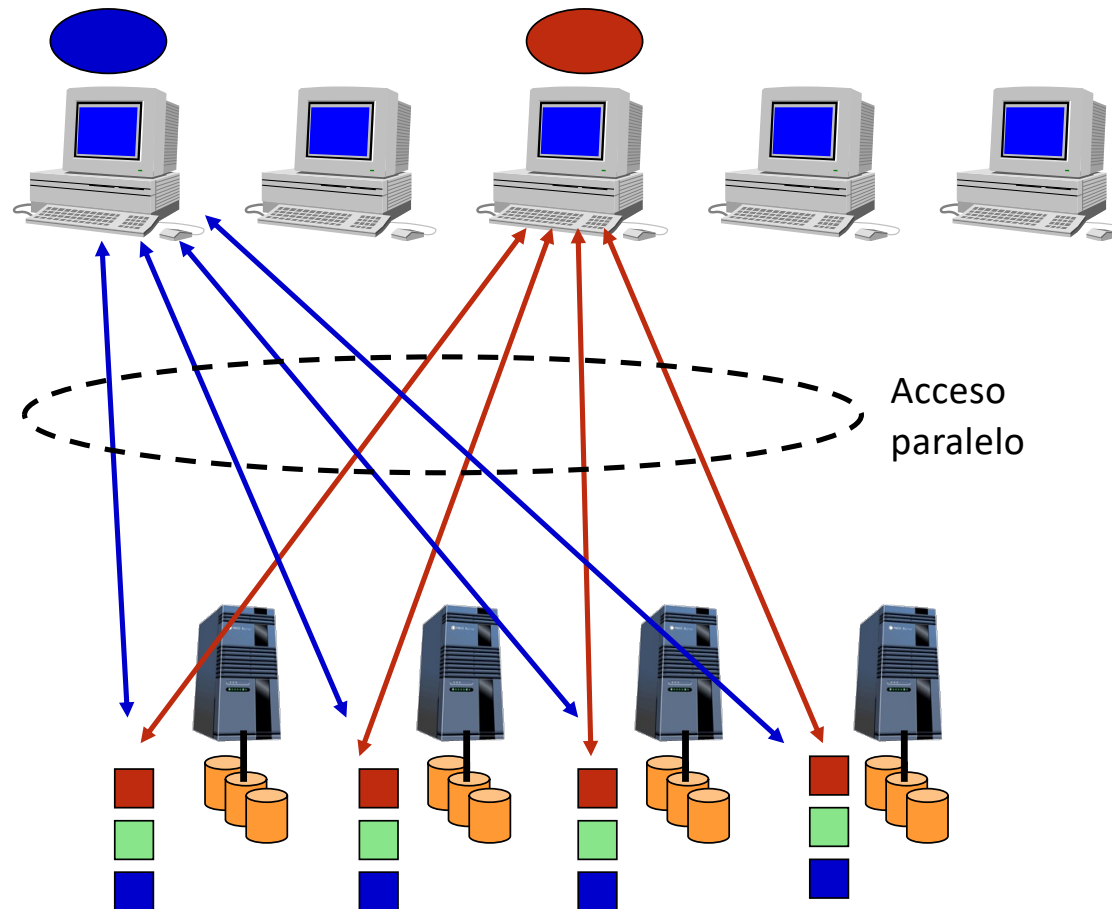


- Múltiples nodos de E/S
  - Incrementa el ancho de banda
- Un fichero se distribuye entre varios nodos de E/S
  - Acceso paralelo
    - A diferentes ficheros
    - Al mismo fichero
- Interfaces paralelas de E/S
  - MPI-IO
- Optimizaciones
  - E/S colectiva
  - Acceso a datos no contiguos

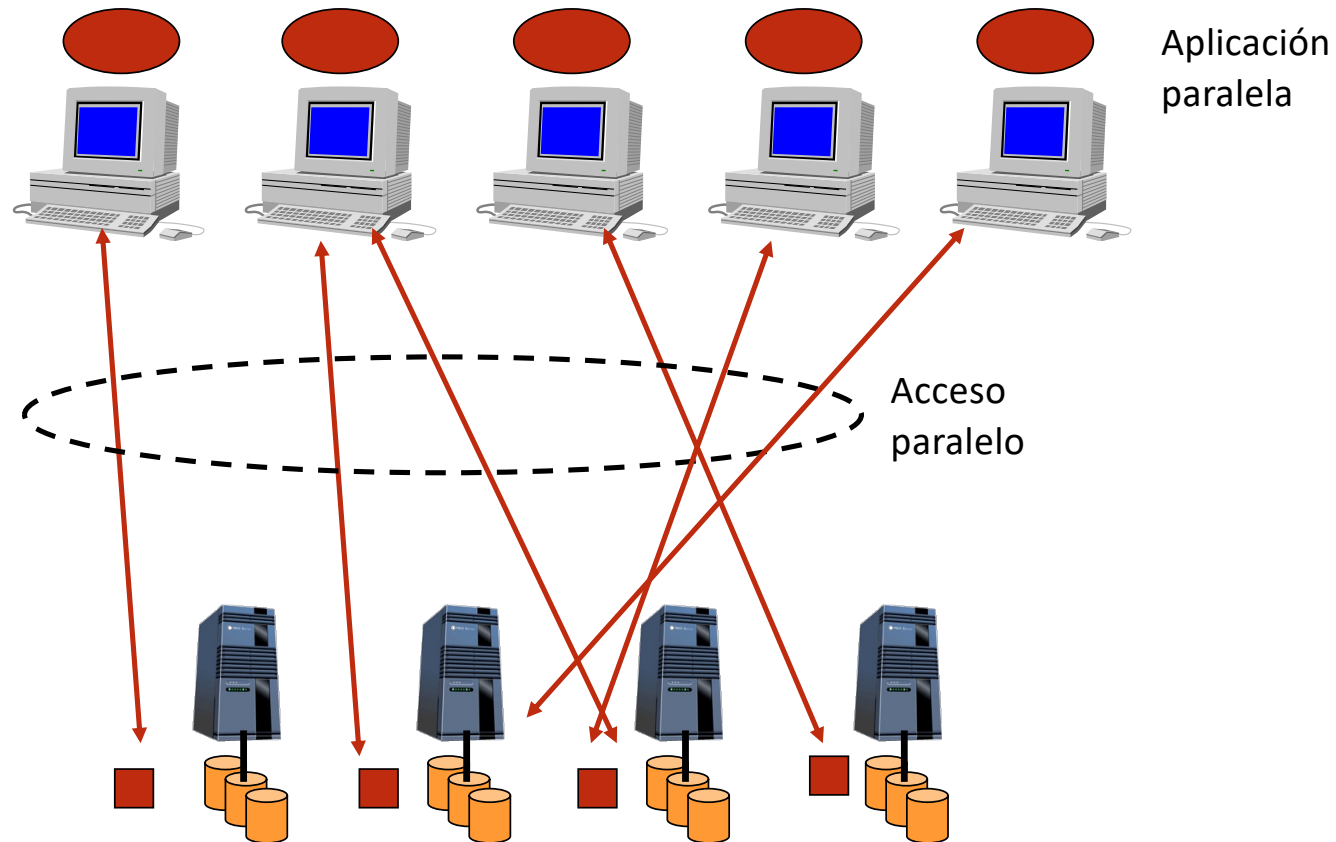
# Acceso paralelo



# Acceso paralelo



# Acceso paralelo



# Ventajas de usar un Sistema de ficheros paralelo

- Uso de API estándar (POSIX) y API especiales (MPI-IO)
- Acceso paralelo a archivos (para lectura y escritura) desde varios nodos cliente
- Aumenta el ancho de banda mediante la distribución de los archivos en múltiples discos y nodos de E/S
- Equilibrio de carga entre discos y nodos de E/S. Maximiza el ancho de banda
- Permite almacenar grandes conjuntos de datos

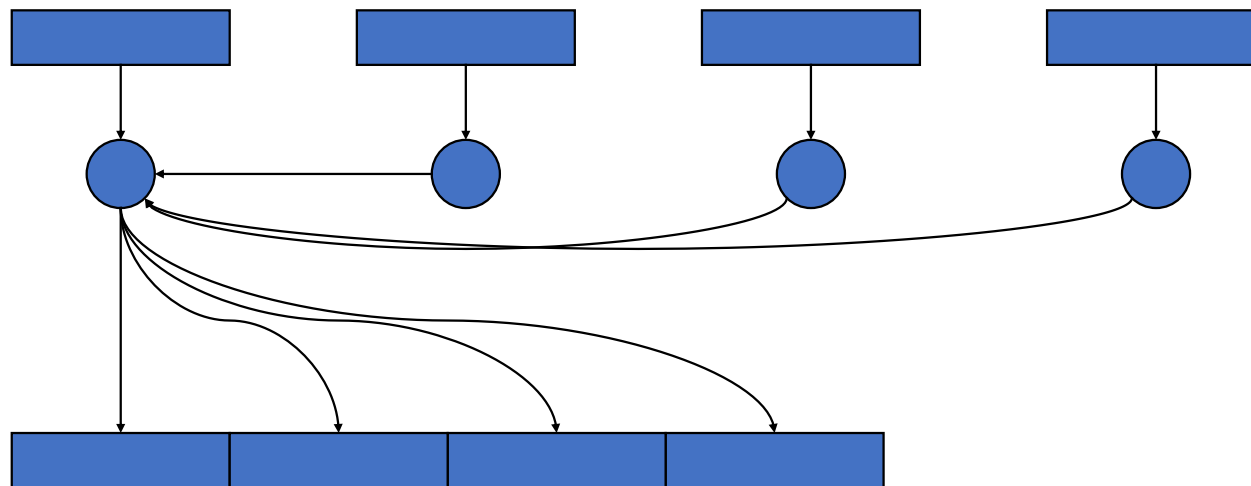
# Ejemplos de SFP

- GPFS
- Lustre
- BeeGFS
- Expand

# Entrada/salida (E/S) en programas paralelos

- E/S secuencial

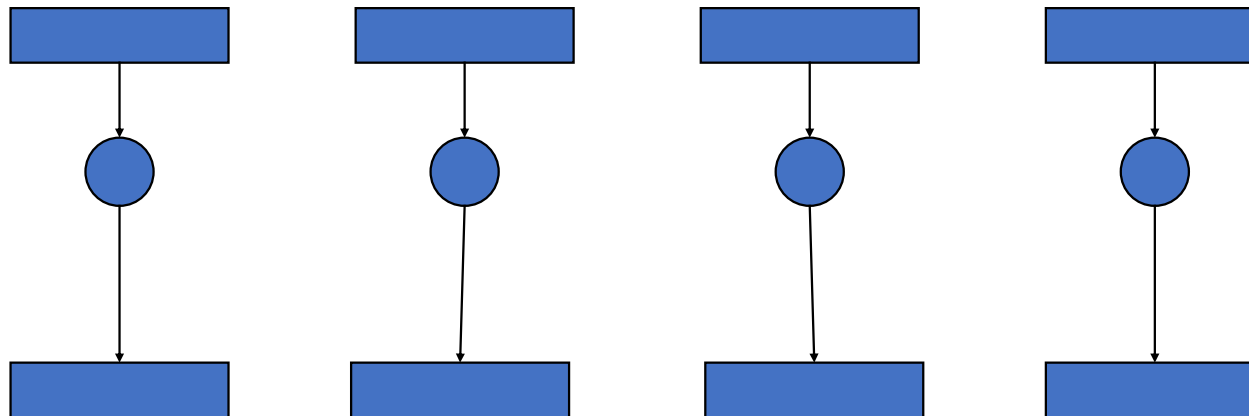
- Todos los procesos envían los datos al proceso 0 y él escribe los datos en el fichero



No existe paralelismo, se limita la escalabilidad y el rendimiento

# E/S en programas paralelos

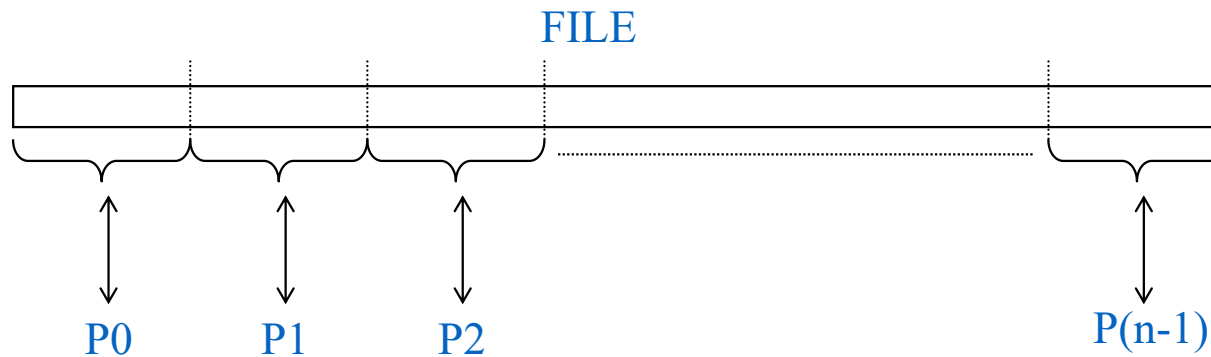
- Cada proceso escribe en un fichero diferente



- Ventajas:
  - Paralelismo, alto rendimiento
- Desventajas:
  - Muchos ficheros, dificultad de gestión
  - Dificultad para el procesamiento posterior de los datos

# E/S en programas paralelos

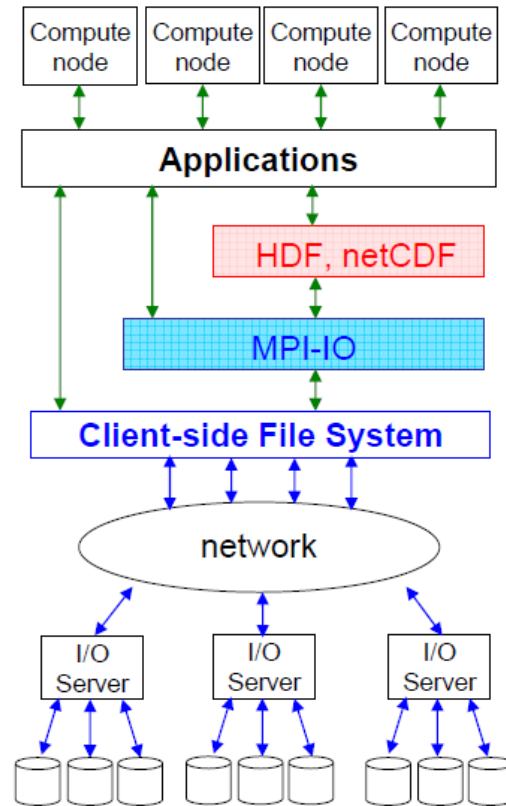
- E/S paralela
  - Los procesos de un programa paralelo acceden a un **único** fichero para realizar operaciones de lectura o escritura.



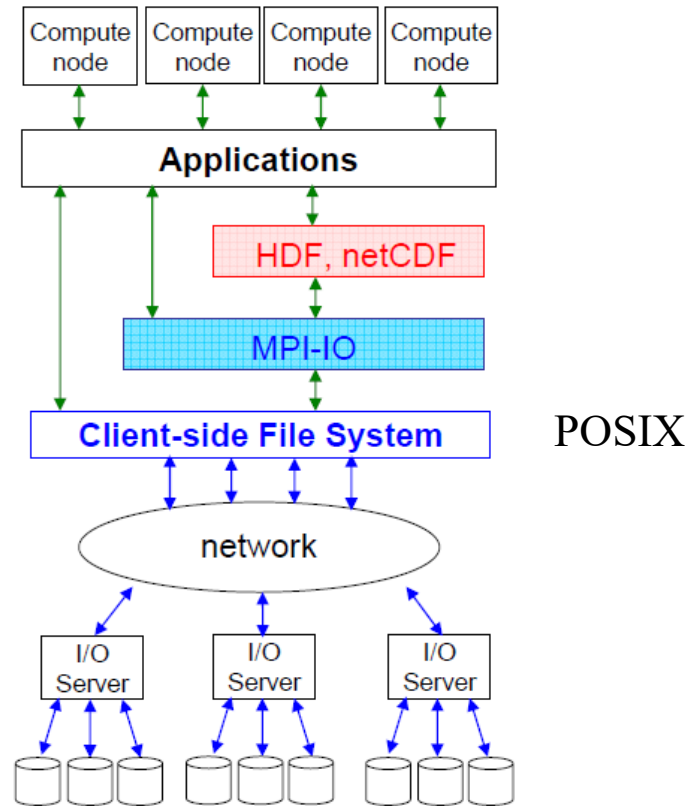
# ¿Por qué utilizar E/S paralela?

- La E/S secuencial es sencilla pero:
  - Ofrece poco rendimiento
  - Dificulta el procesamiento posterior de varios ficheros independientes
- E/S paralela:
  - Mejora el rendimiento en los accesos
  - Ofrece un único fichero que puede ser procesado y utilizado de forma sencilla por otras herramientas, como programas de visualización

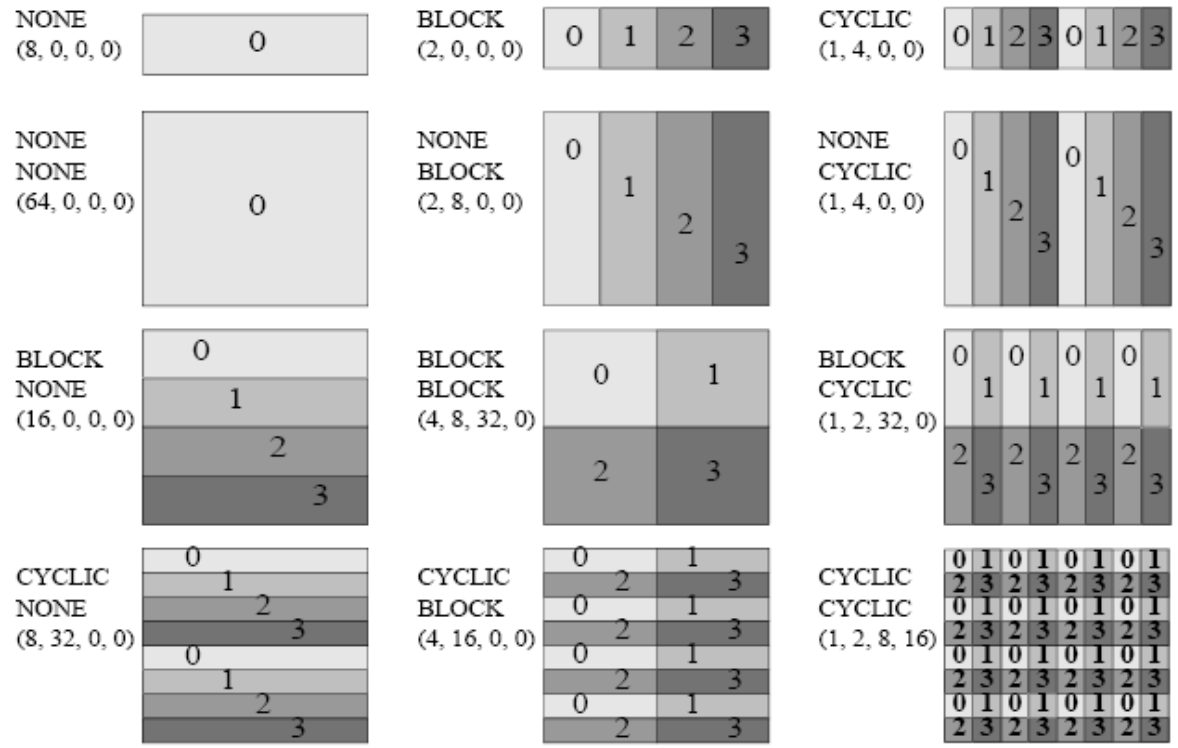
# Niveles de E/S paralela



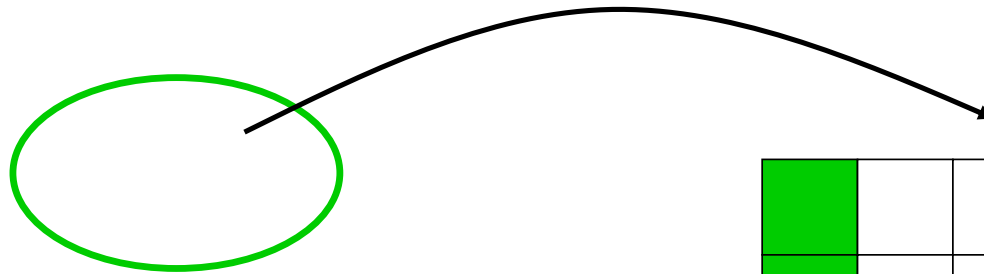
# Niveles de E/S paralela



# Patrones de acceso en aplicaciones paralelas



# Problemas de las interfaces de E/S clásicas como POSIX

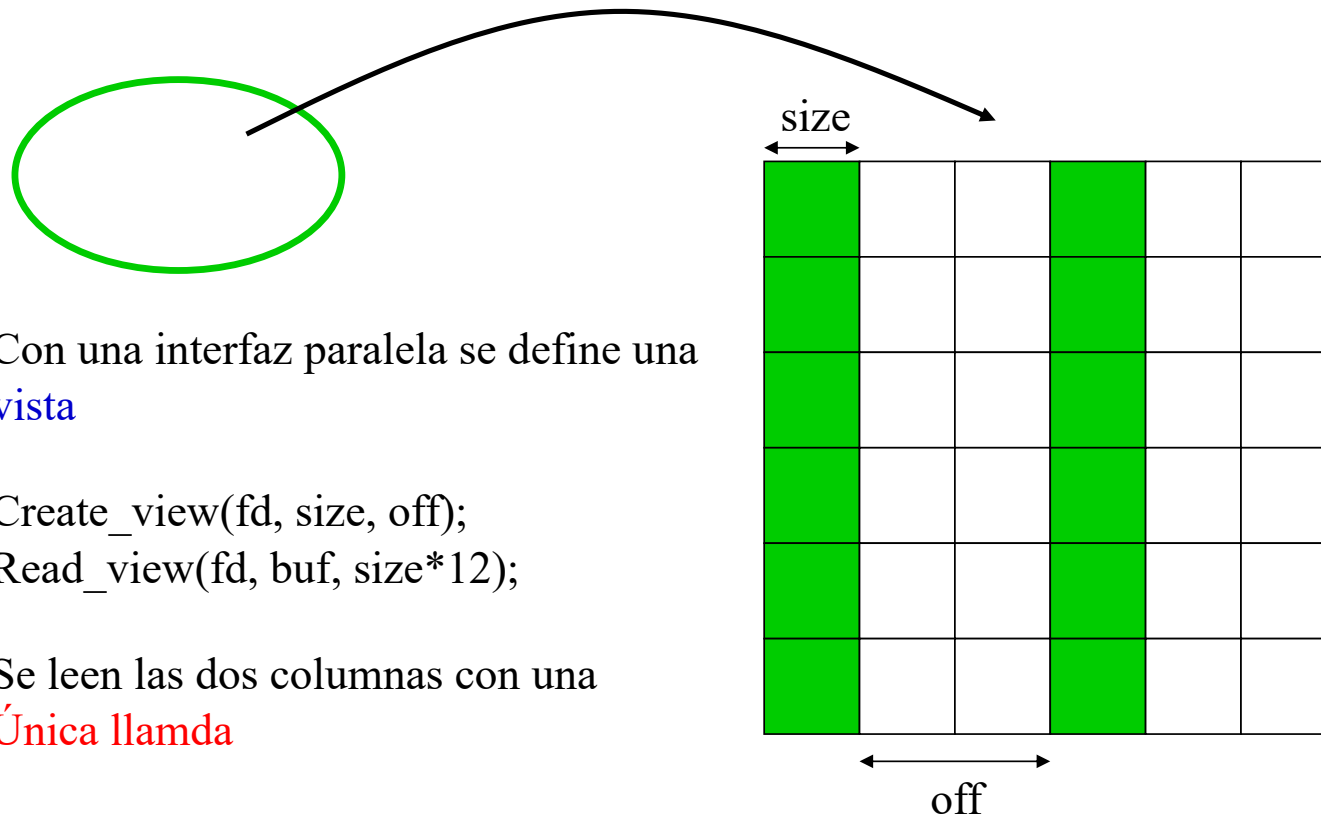


Con servicios POSIX:

```
read();  
lseek();  
read();  
lseek();  
read();
```

·  
· **Se necesitan 23 llamadas para acceder a datos no contiguos**

# Objetivo de las interfaces paralelas



Con una interfaz paralela se define una **vista**


```
Create_view(fd, size, off);  
Read_view(fd, buf, size*12);
```

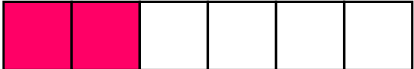
Se leen las dos columnas con una **Única llamada**

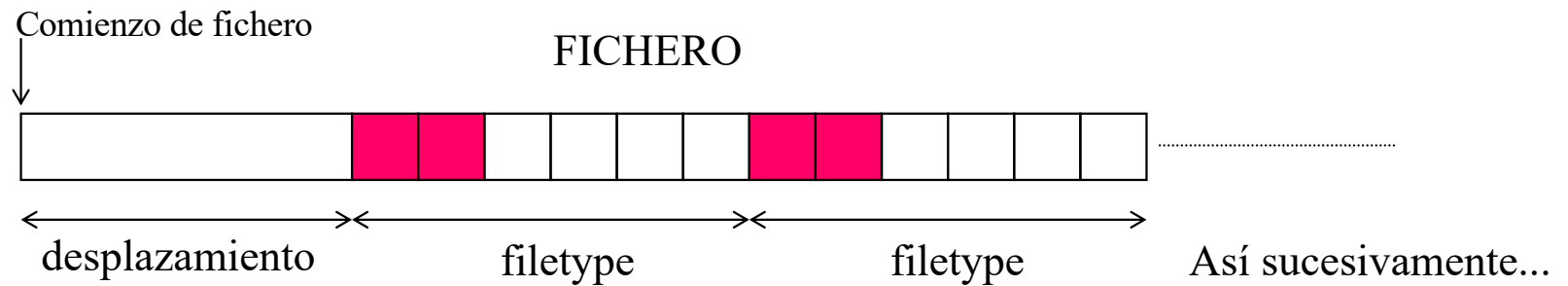
# MPI-IO

- Lectura/escritura paralela.
- Lectura/escritura a datos no contiguos.
- Lectura/escritura sin bloqueos
- Lecturas/escrituras colectivas.
- Representación de datos portable entre plataformas.
- Tipos de datos derivados

# Vista de un fichero (File view) en MPI-IO

 etype = MPI\_INT

 filetype = dos two MPI\_INTs seguidos por 4 MPI\_INTs



# Ejemplo: escritura en un fichero compartido

```
#include<stdio.h>
#include "mpi.h"
int main(int argc, char **argv){
    int i, rank, size, offset, nints, N=16;
    MPI_File fhw;
    MPI_Status status;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    int buf[N];
    for ( i=0;i<N;i++){
        buf[i] = i ;
    }
}
```

# Ejemplo: escritura en un fichero compartido

```
offset = rank*(N/size)*sizeof(int);
```

```
MPI_File_open(MPI_COMM_WORLD, "datafile3",  
             MPI_MODE_CREATE|MPI_MODE_WRONLY, MPI_INFO_NULL, &fhw);  
printf("\nRank: %d, Offset: %d\n", rank, offset);
```

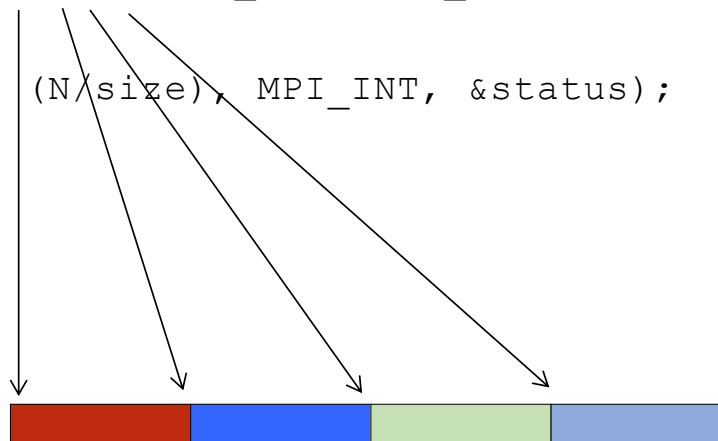
```
MPI_File_set_view(fhw, offset, MPI_INT, MPI_INT, "native",  
                MPI_INFO_NULL);
```

```
MPI_File_write(fhw, buf, (N/size), MPI_INT, &status);
```

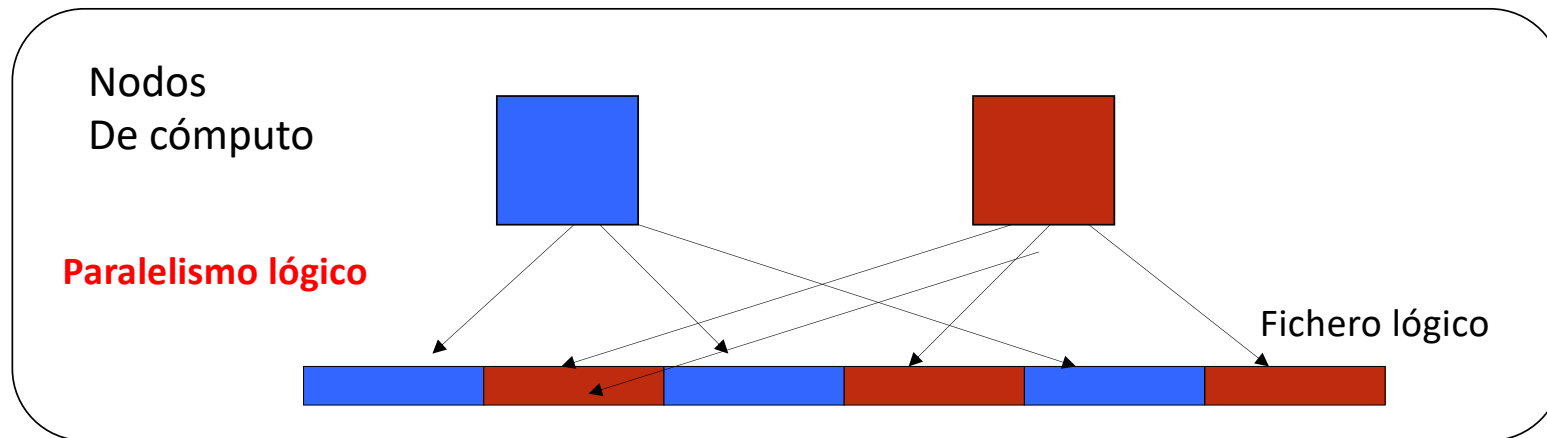
```
MPI_File_close(&fhw);
```

```
MPI_Finalize();  
return 0;
```

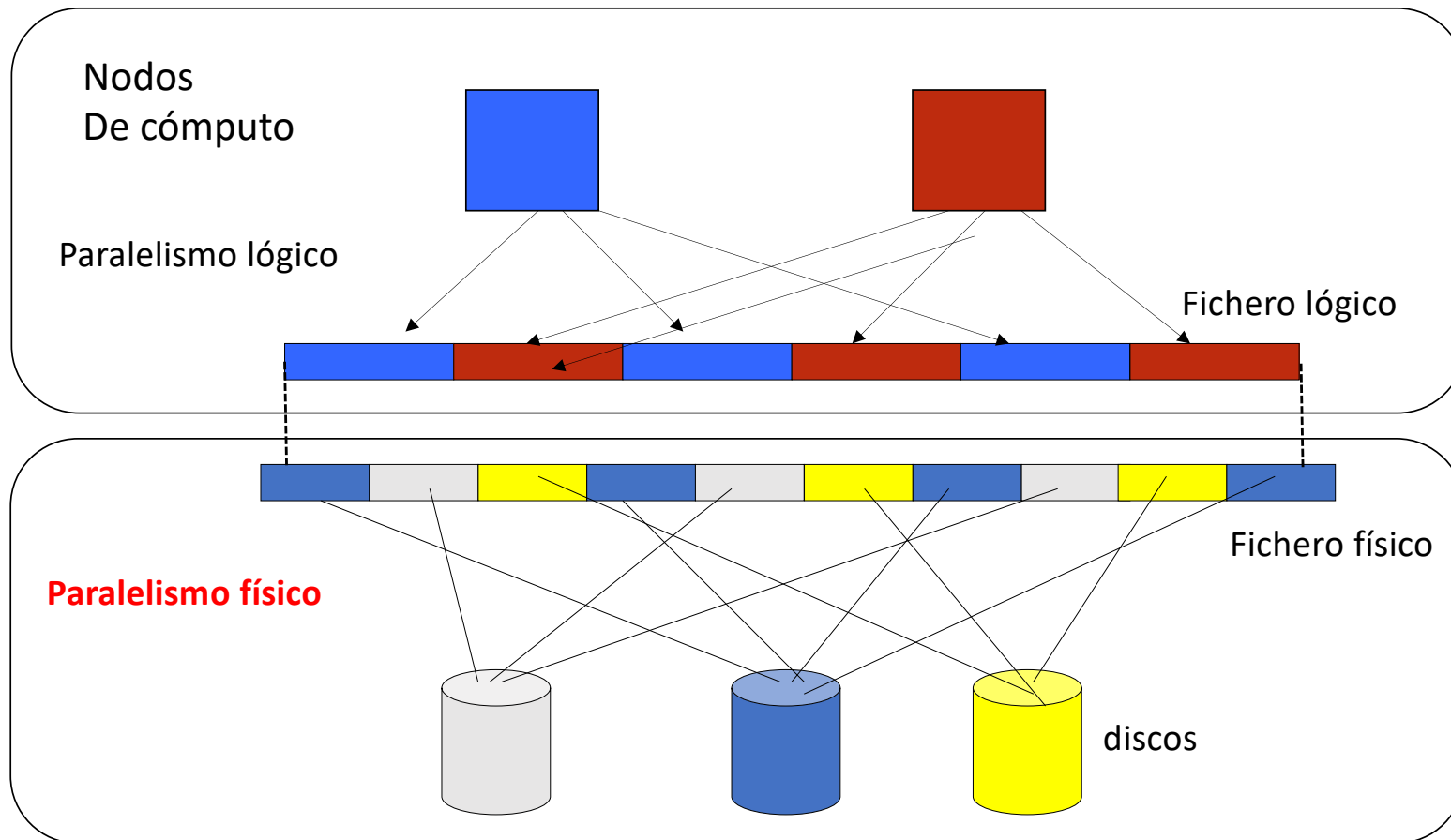
```
}
```



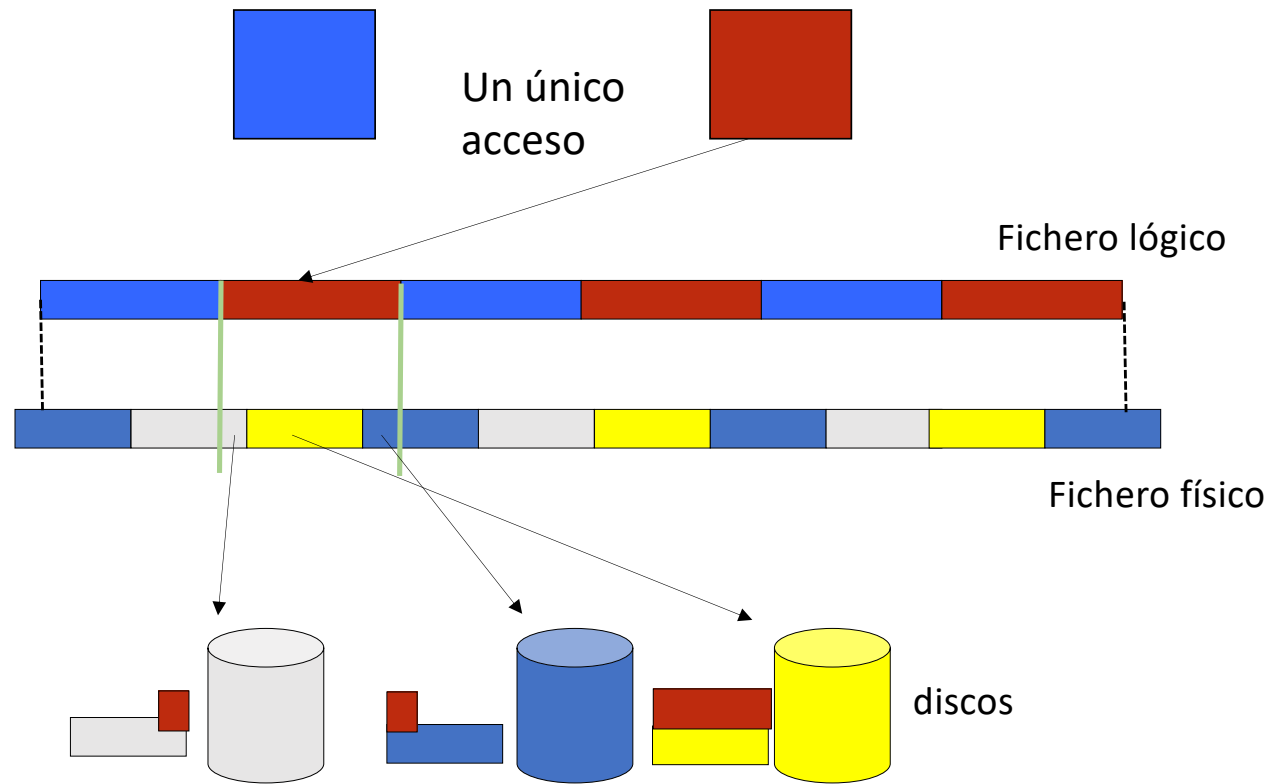
# Distintos tipos de paralelismo



# Distintos tipos de paralelismo



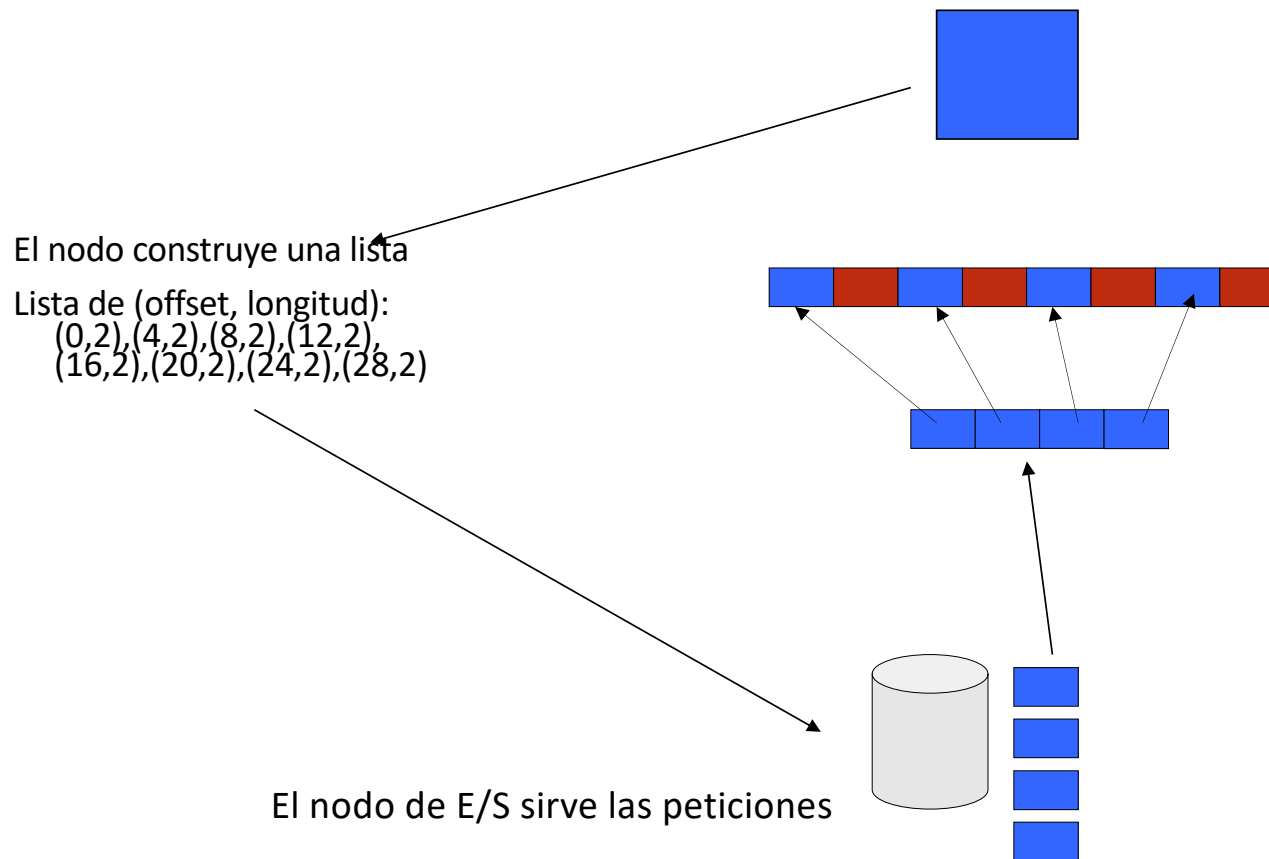
# Problema: Muchos accesos a datos no contiguos



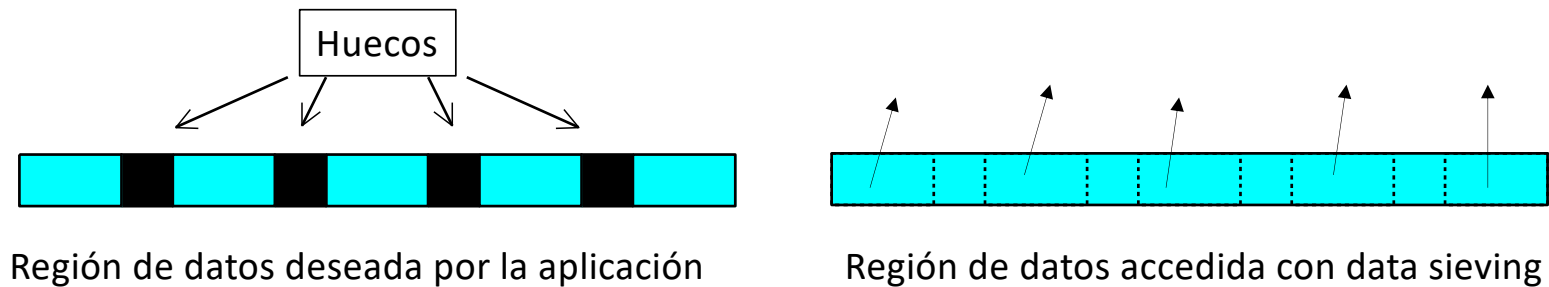
# Optimizaciones

- E/S basada en listas (List I/O)
- Cribado de datos (Data sieving)
- E/S en dos fases (Two-phase I/O)
- Agregación (gregation)

# E/S basada en listas (List I/O)

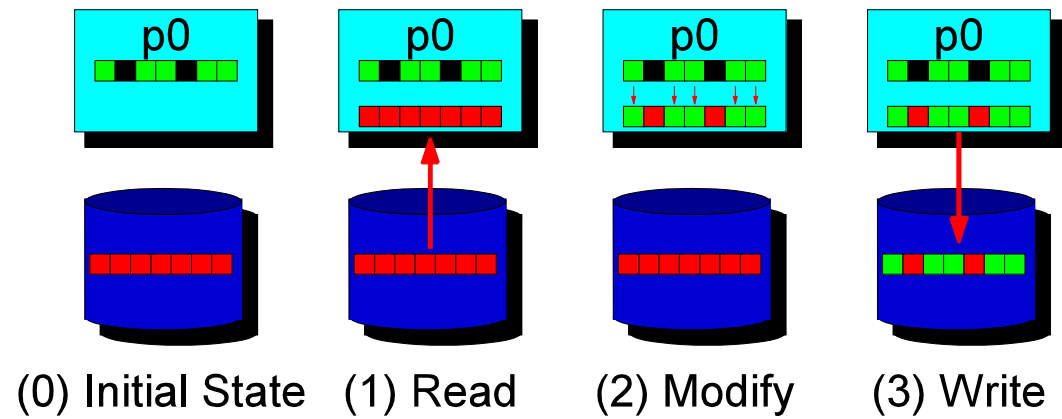


# Cribado de datos (Data sieving)

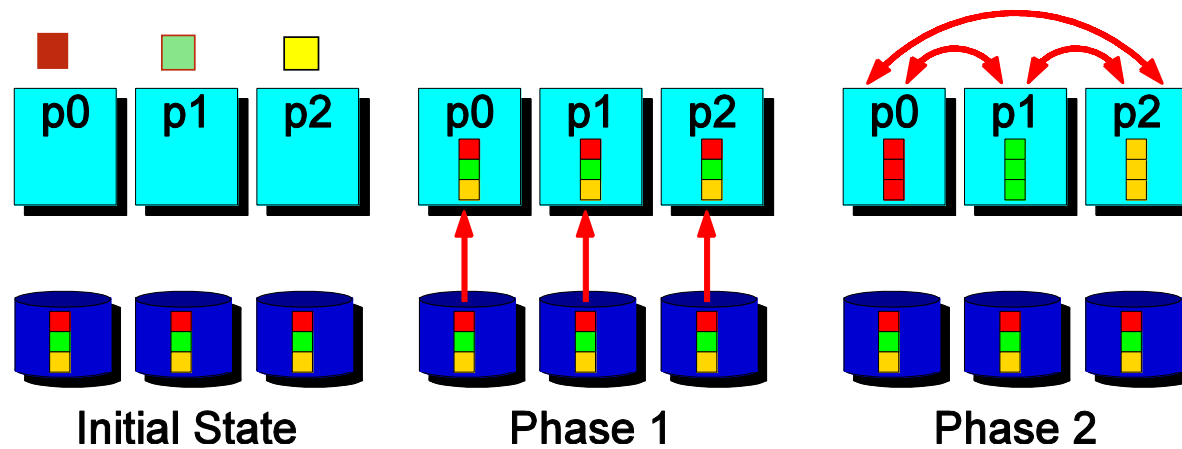


- El cribado de datos se utiliza para combinar muchos accesos pequeños en uno solo más grande
  - Los sistemas de ficheros remotos (paralelos o no) suelen tener latencias elevadas.
  - Es importante reducir el número de operaciones.

# Escritura con Data Sieving

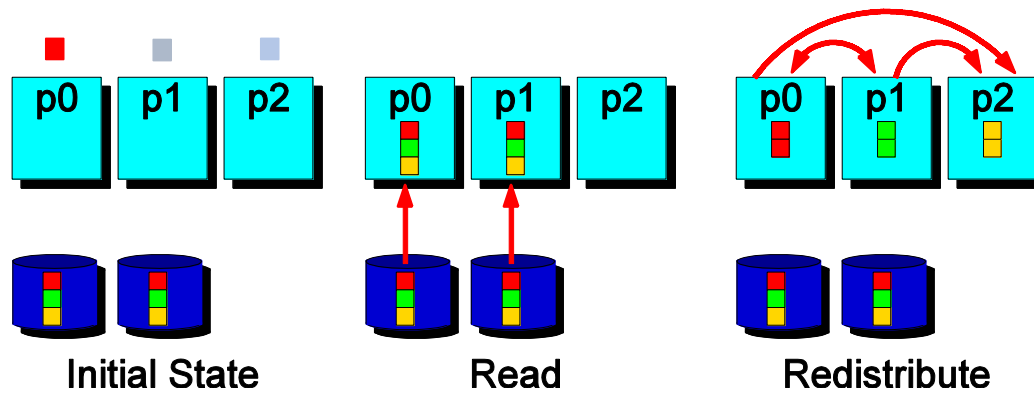


# E/S colectiva en dos fases



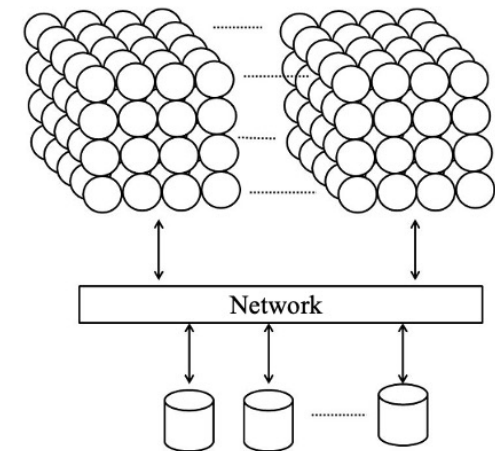
- Objetivo: reducir el número de accesos no contiguos a los servidores

# Agregación



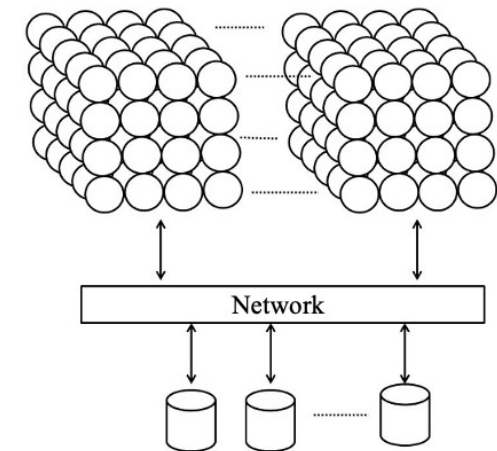
# Sistemas de ficheros paralelos ad-hoc

- Arquitectura típica de un supercomputador:
  - Número de nodos de cómputo mucho mayor que el de nodos de E/S:
    - **Possible cuello de botella**



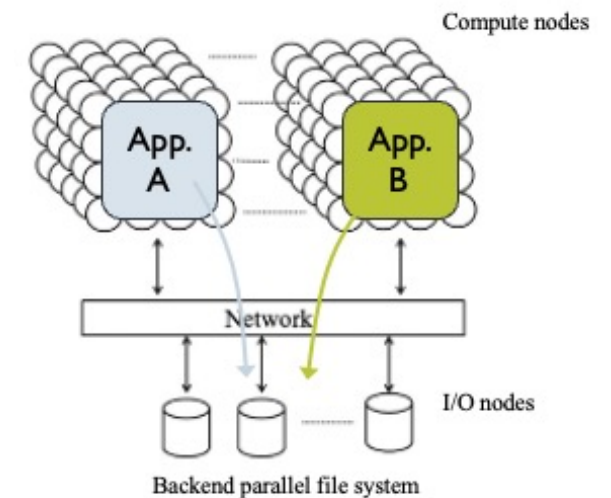
# Sistemas de ficheros paralelos ad-hoc

- Arquitectura típica de un supercomputador:
  - Número de nodos de cómputo mucho mayor que el de nodos de E/S:
    - Posible cuello de botella
  - Múltiples aplicacionesData away from applications:
    - Uso de red
    - Se reduce el rendimiento en el acceso a los datos



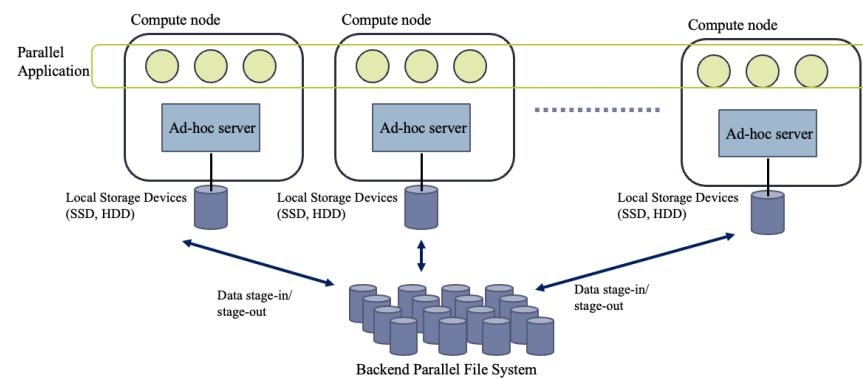
# Sistemas de ficheros paralelos ad-hoc

- Arquitectura típica de un supercomputador:
  - Número de nodos de cómputo mucho mayor que el de nodos de E/S:
    - Posible cuello de botella
  - Múltiples aplicacionesData away from applications:
    - Uso de red
    - Se reduce el rendimiento en el acceso a los datos
  - Diferentes aplicaciones ejecutando al mismo tiempo ::
    - Conflictos e interferencias en el Sistema de ficheros



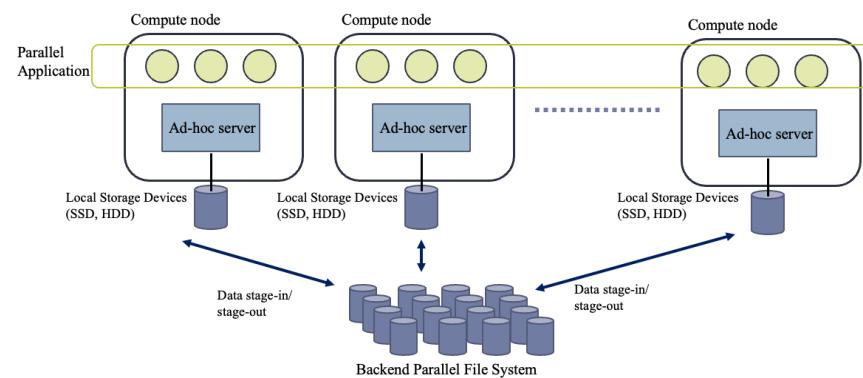
# Sistema de ficheros ad-hoc

- Sistema efímero que ofrece un sistema de almacenamiento intermedio en un cluster.
  - Reduce la presión sobre el sistema de ficheros (back-end) del cluster
  - Incrementa el rendimiento de la aplicación, puede ajustarse el número de nodos de E/S al número de nodos de procesamiento
  - Se adapta a la ejecución de workflows científicos
  - Ficheros temporales de checkpoint no necesitan almacenarse en el sistema de ficheros (back-end) del cluster

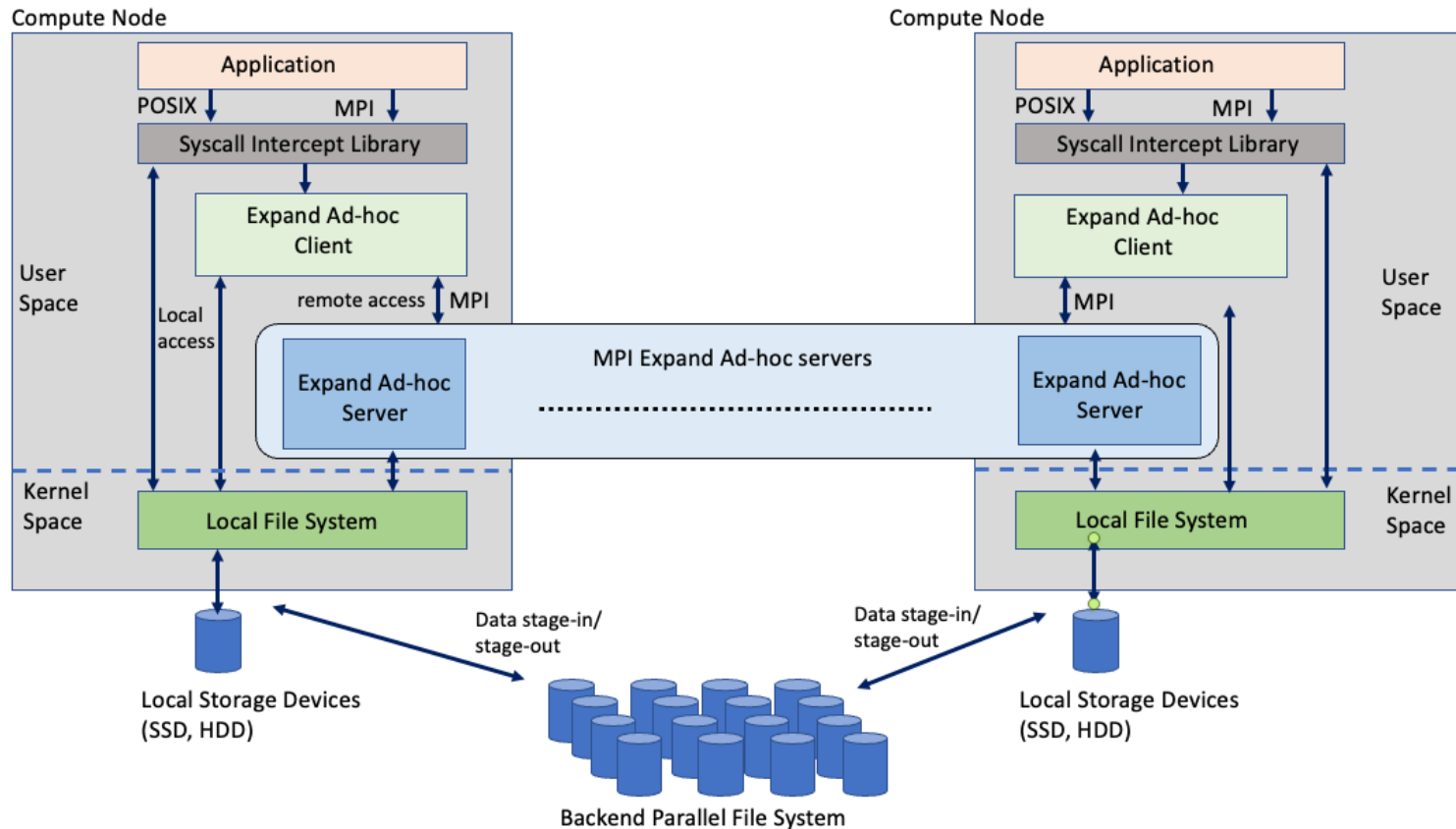


# Sistema de ficheros ad-hoc. Ventajas

- Datos cercanos a la aplicación: incrementa el rendimiento
- Reduce el uso de la red del supercomputador
- Reduce los cuellos de botella en el sistema de ficheros backend
- Los datos temporales (checkpoints, datos intermedios) no necesitan almacenarse en el sistema de ficheros backend
- Mejora las prestaciones de workflows de aplicaciones
- Se adapta al número de procesos/nodos de la aplicación



# Ejemplo: Expand Ad-hoc

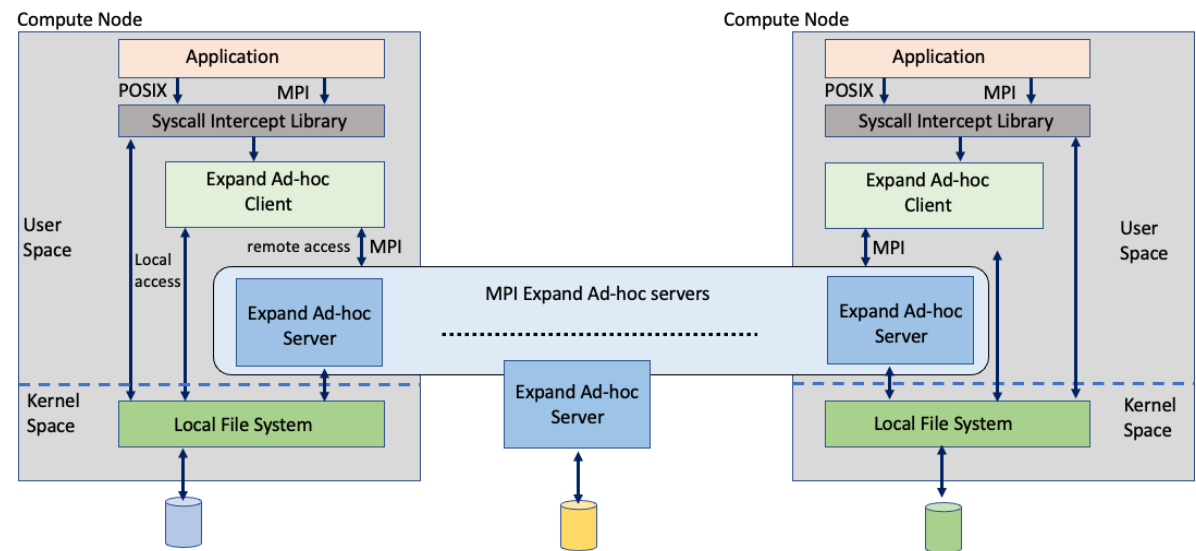


# Distribución de datos

Fichero en Expand



- Fichero en Expand:
  - Subfichero de metadatos
  - Varios subficheros de datos
- Datos distribuidos en varios servidores

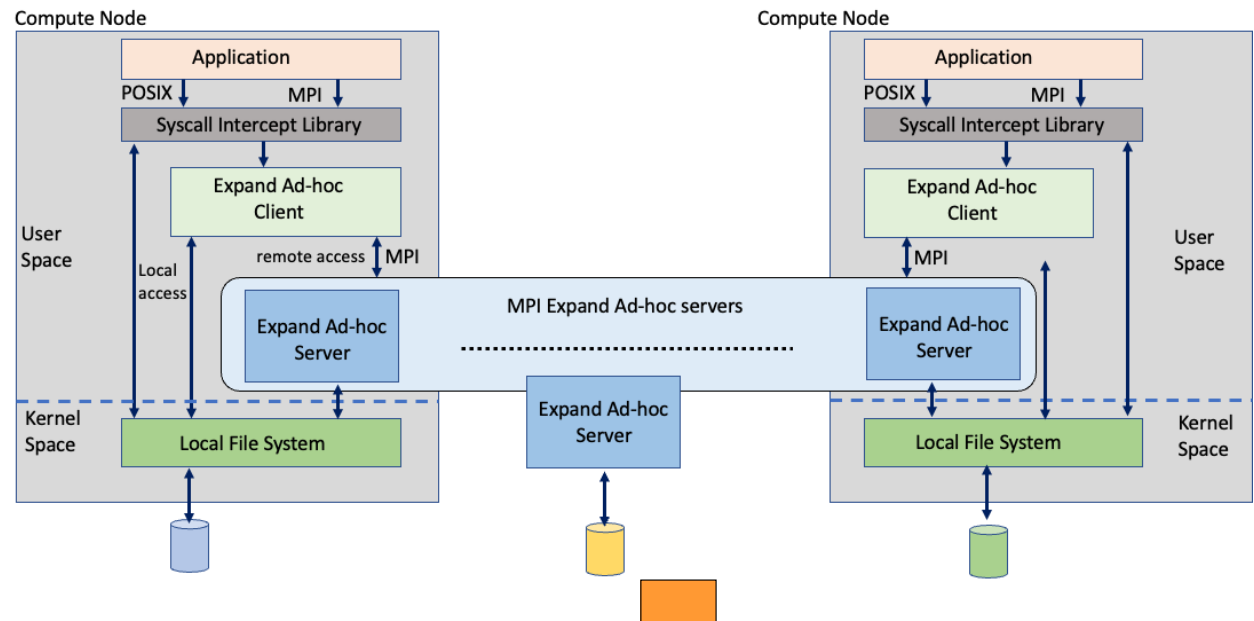


# Distribución de datos

Fichero en Expand

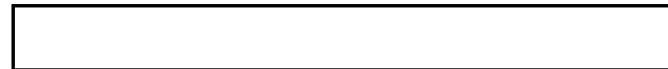


- Fichero en Expand:
  - Subfichero de metadatos
  - Varios subficheros de datos
- Datos distribuidos en varios servidores

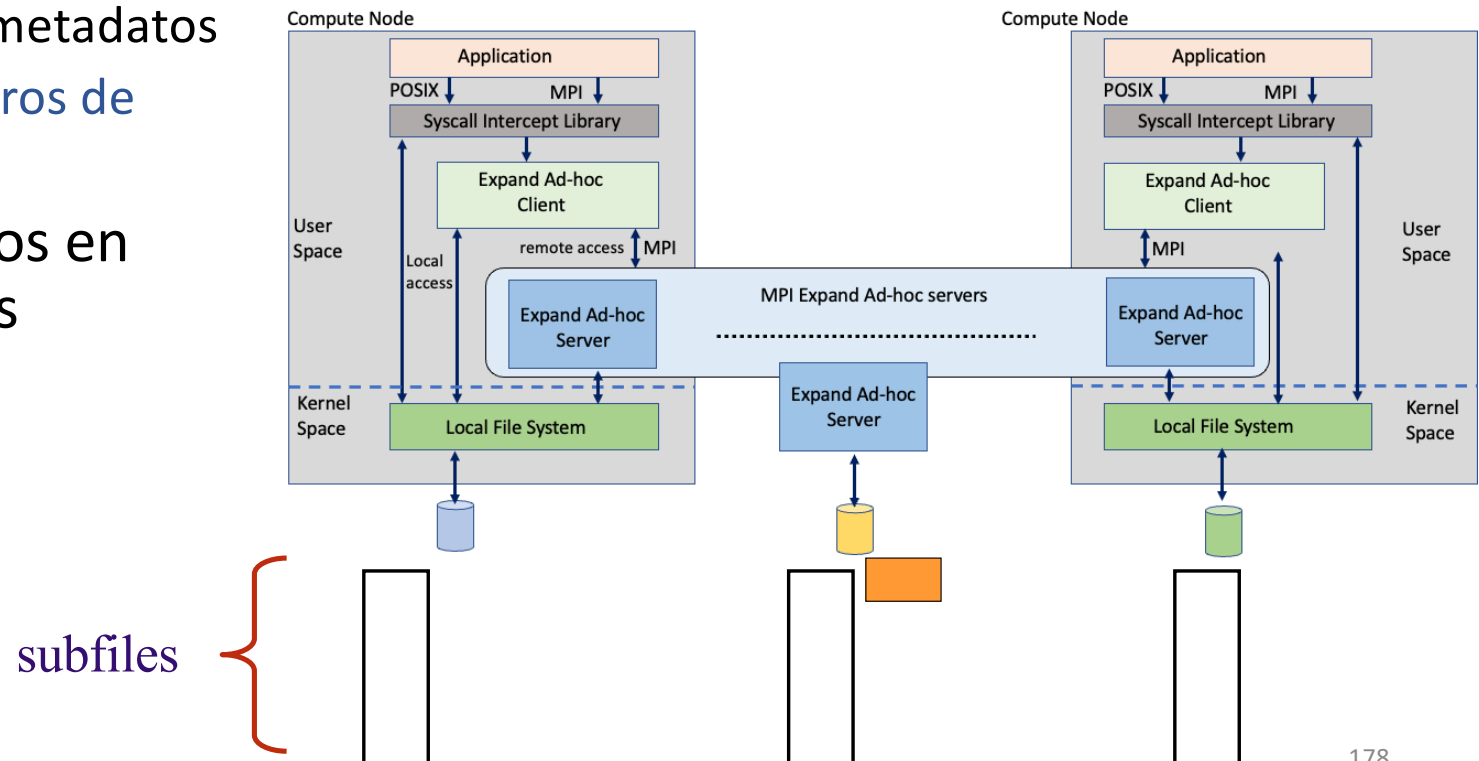


# Data distribution

Fichero en Expand

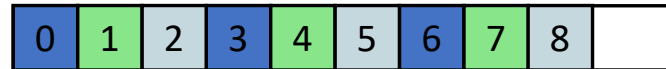


- Fichero en Expand:
  - Subfichero de metadatos
  - Varios subficheros de datos
- Datos distribuidos en varios servidores

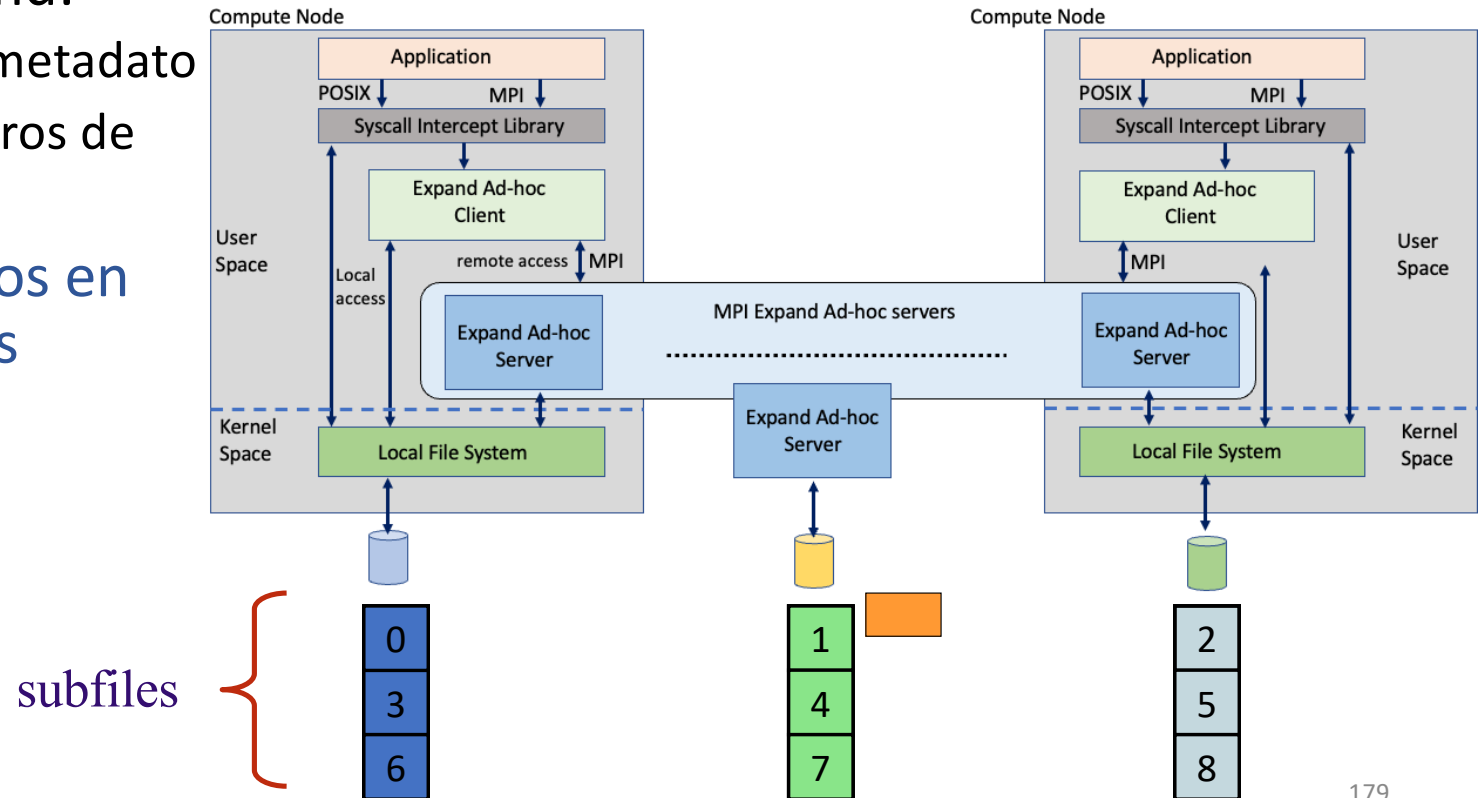


# Data distribution

Fichero en Expand

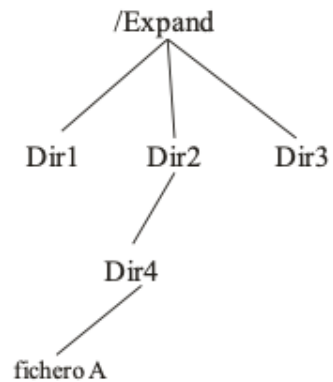


- Fichero en Expand:
  - Subfichero de metadato
  - Varios subficheros de datos
- Datos distribuidos en varios servidores

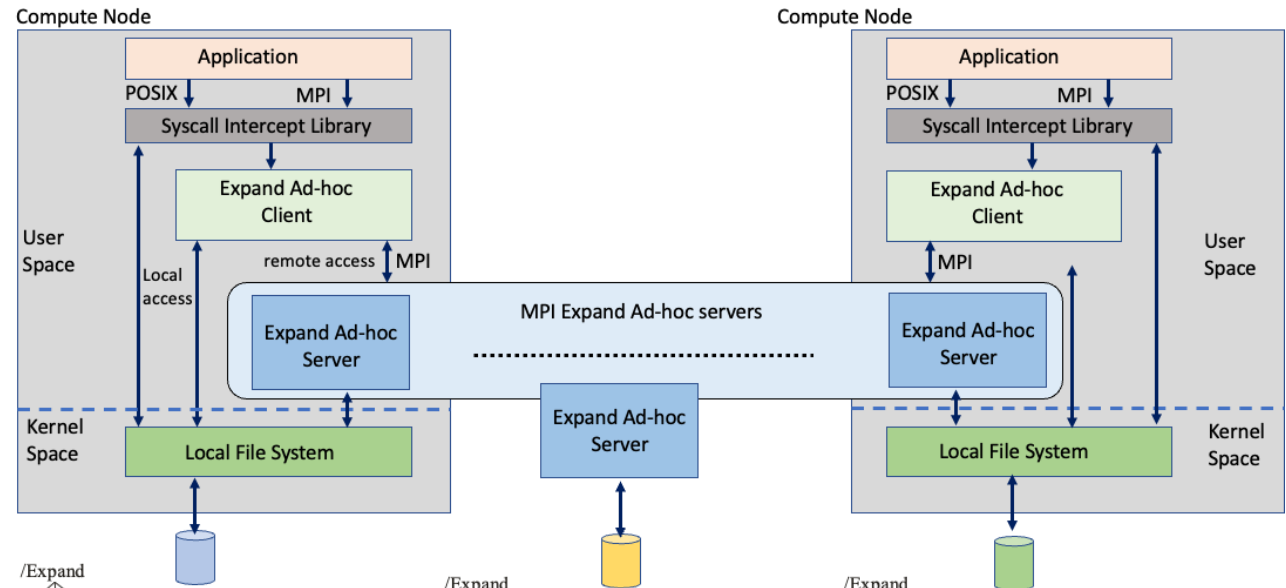
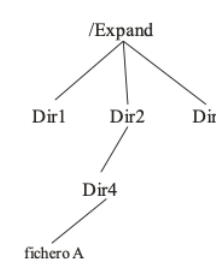
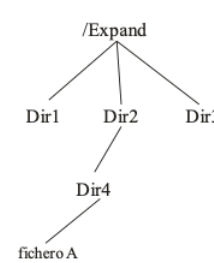
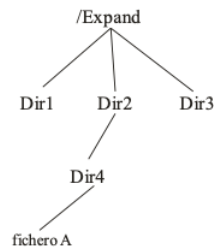


# Estructura de directorios

Visión lógica

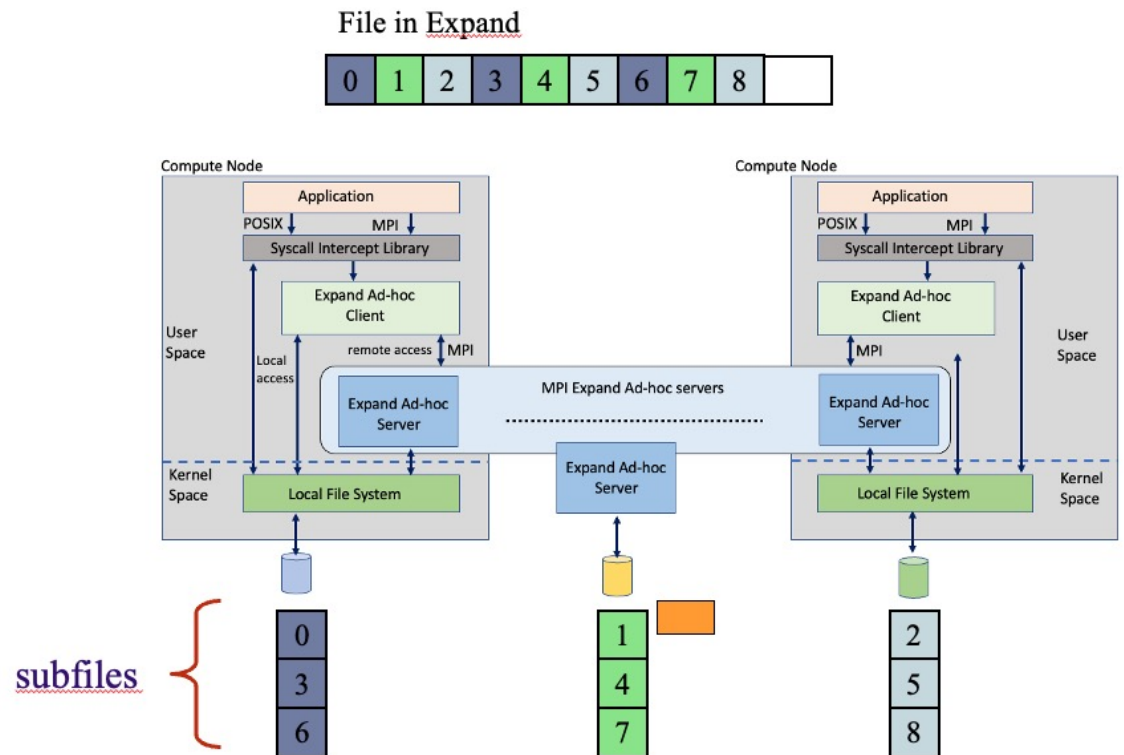


Visión física



# Gestión de metadatos

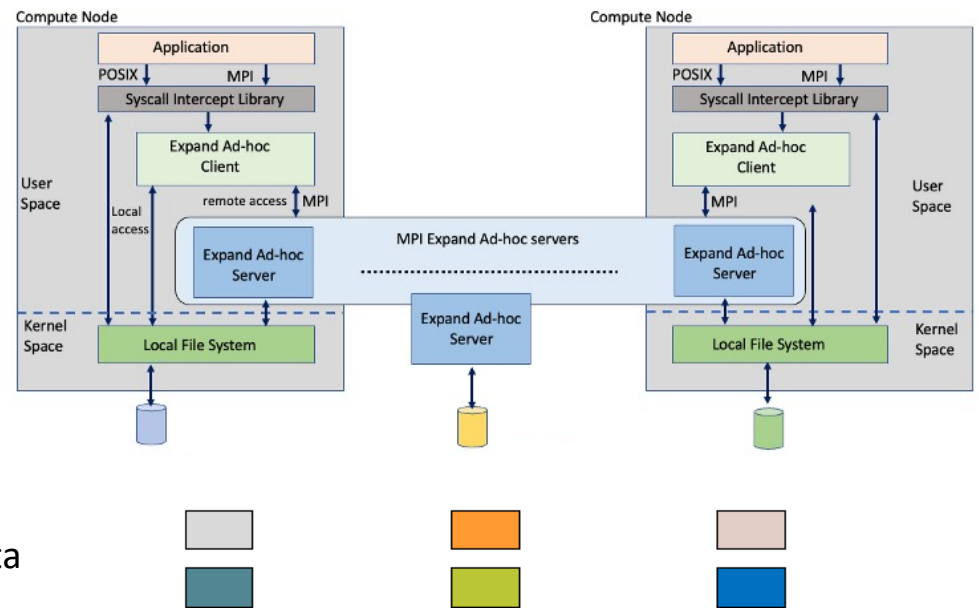
- Gestión de datos distribuida:
  - Dos niveles
  - Sin cerrojos
  - Sin gestor de metadatos
- Metadatos distribuidos entre los servidores:
  - Nodos master
  - Función hash(name)
  - Equilibrio de carga



# Gestión de metadatos

- Gestión de datos distribuida:
  - Dos niveles
  - Sin cerrojos
  - Sin gestor de metadatos
- Metadatos distribuidos entre los servidores:
  - Nodos master
  - Función hash(name)
  - Equilibrio de carga

File in Expand



# Sistema de ficheros distribuido de gran escala: Hadoop file system

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



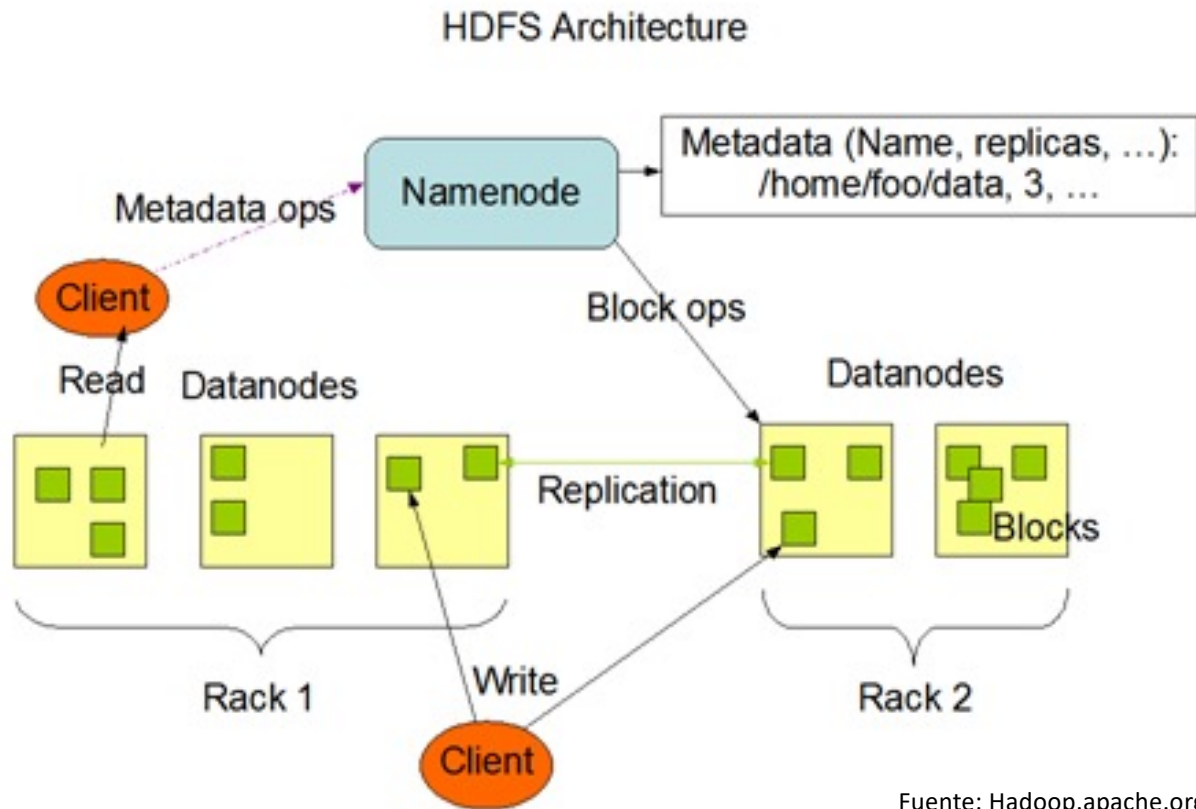
# Características básicas

- 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

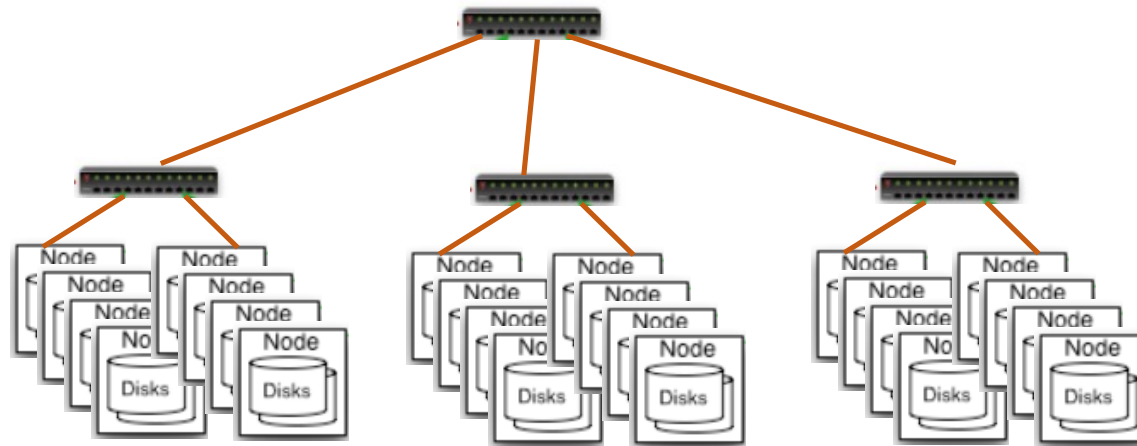
# Arquitectura



Fuente: Hadoop.apache.org

# Plataforma HW típica

- Clusters de PC organizados en racks



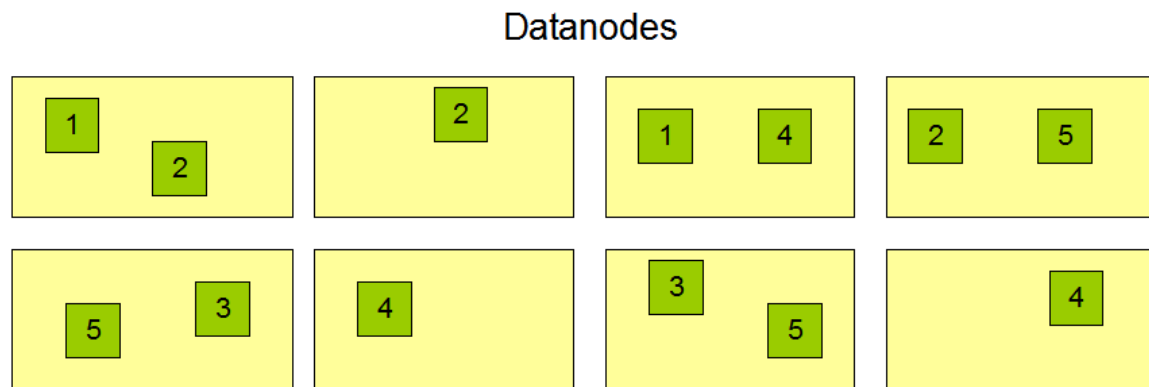
# Espacio de nombres

- Espacio de nombres jerarquico
- Operaciones: crear, eliminar, renombar, etc.
- El espacio de nombres y los metadatos se almacenan en el Namenode
- Las aplicaciones pueden especificar el número de réplicas para un fichero: factor de replicación (se almacena como parte de los metadatos)

# Replicación de datos

- HDFS está diseñado para almacenar grandes ficheros
- Cada fichero consta de una secuencia de bloques (64, 128 MB)
- Todos los bloques son del mismo tamaño, excepto el último
- Los bloques se replican en diversos DataNodes
- El NameNode monitoriza el estado de los DataNodes. Recibe periódicamente un *Heartbeat*.

# Replicación



Fuente: Hadoop.apache.org

# Metadatos

- Todos los metadatos se almacenan en el NameNode
- El sistema de ficheros entero se almacena en un fichero denominado FsImage que se almacena en el sistema de ficheros local del NameNode
- Mantiene una imagen del sistema de ficheros en memoria
- El NameNode es un punto único de fallo en el sistema
- NameNode secundario

# DataNodes

- Servidor de bloques
- No tienen conocimiento de la estructura de HDFS
- Cada bloque se almacena en un fichero diferente utilizando el sistema de ficheros local
- No almacena todos los ficheros en el mismo directorio
- Utiliza heurísticas para determinar el número óptimo de ficheros por directorio

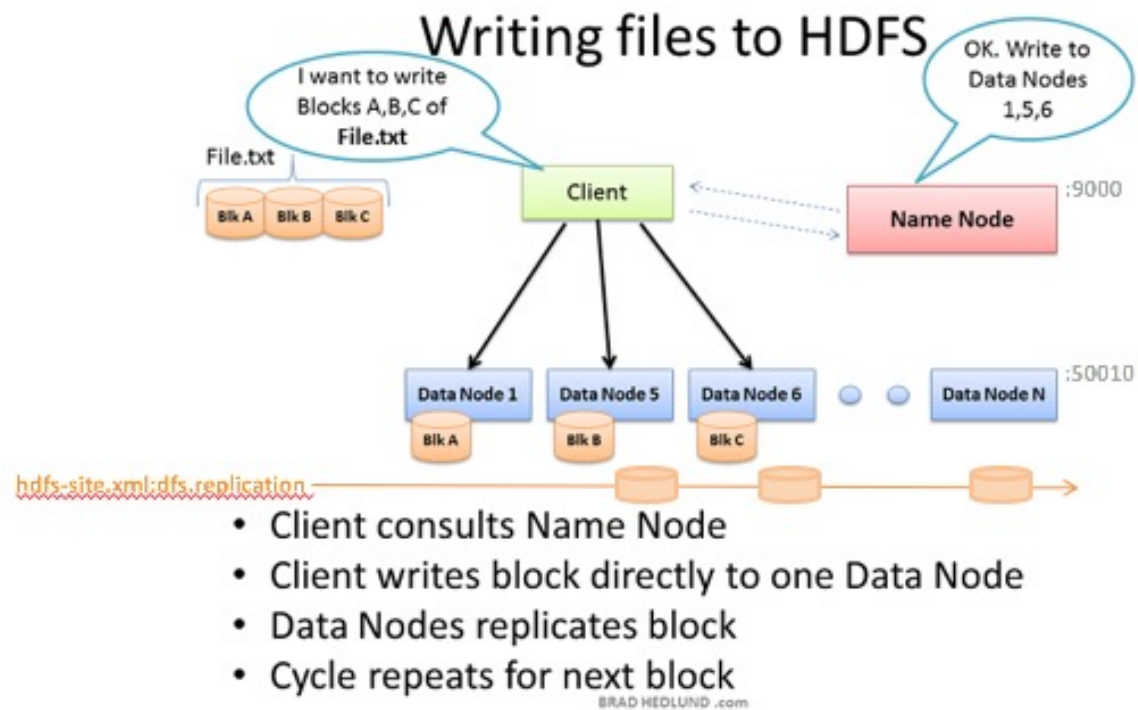
# Réplicas

- Localización
  - Una réplica se almacena en un nodo del rack local, otra en otro nodo del mismo rack y otra en un nodo de otro rack. Replicas adicionales se almacenan aleatoriamente
- Selección
  - Los clientes leen de la réplica más cercana
  - Si hay una réplica en el mismo rack del lector se utiliza ésta

# Creación de un fichero

- Cuando un cliente crea un fichero no contacta con el NameNode inmediatamente
- El cliente HDFS almacena los datos en una cache local.
- Cuando los datos alcanzan el tamaño del bloque se contacta con el NameNode
- El NameNode inserta el nombre del fichero y asigna bloques
- El NameNode responde al cliente con el destino de las replicas para el bloque
- El cliente vuelca los datos

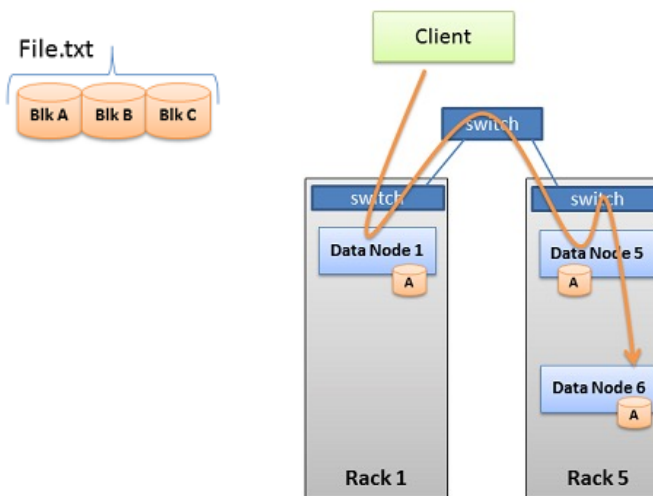
# Escritura de un fichero



<http://blog.csdn.net/suifeng3051/article/details/17288047>

# Pipeline de replicación

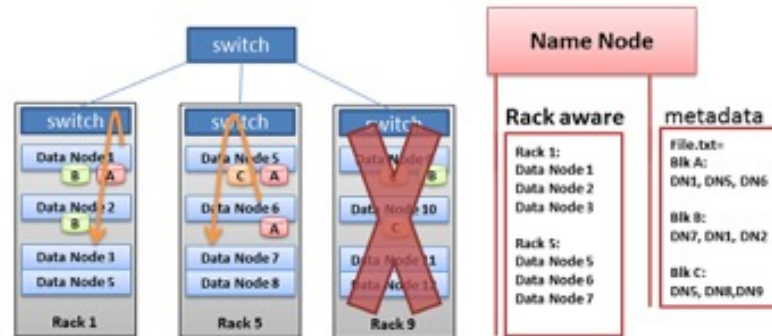
- Cuando un cliente vuelca los datos de un bloque lo hace en trozos de 4 KB
- Cuando el primer trozo de 4 KB llega al primer DataNode escribe su porción en disco y transfiere el bloque al siguiente DataNode y así sucesivamente



Fuente: Hadoop.apache.org

# Replicación

## Hadoop Rack Awareness – Why?



- Never loose all data if entire rack fails
- Keep bulky flows in-rack when possible
- Assumption that in-rack is higher bandwidth, lower latency

BRAD HEDLUND .com

# Otros ejemplos

- Google File System
- Colossus (GFS)
- BigTable
- Spanner
- Dynamo
- Megastore