Sistemas de Archivos Distribuidos

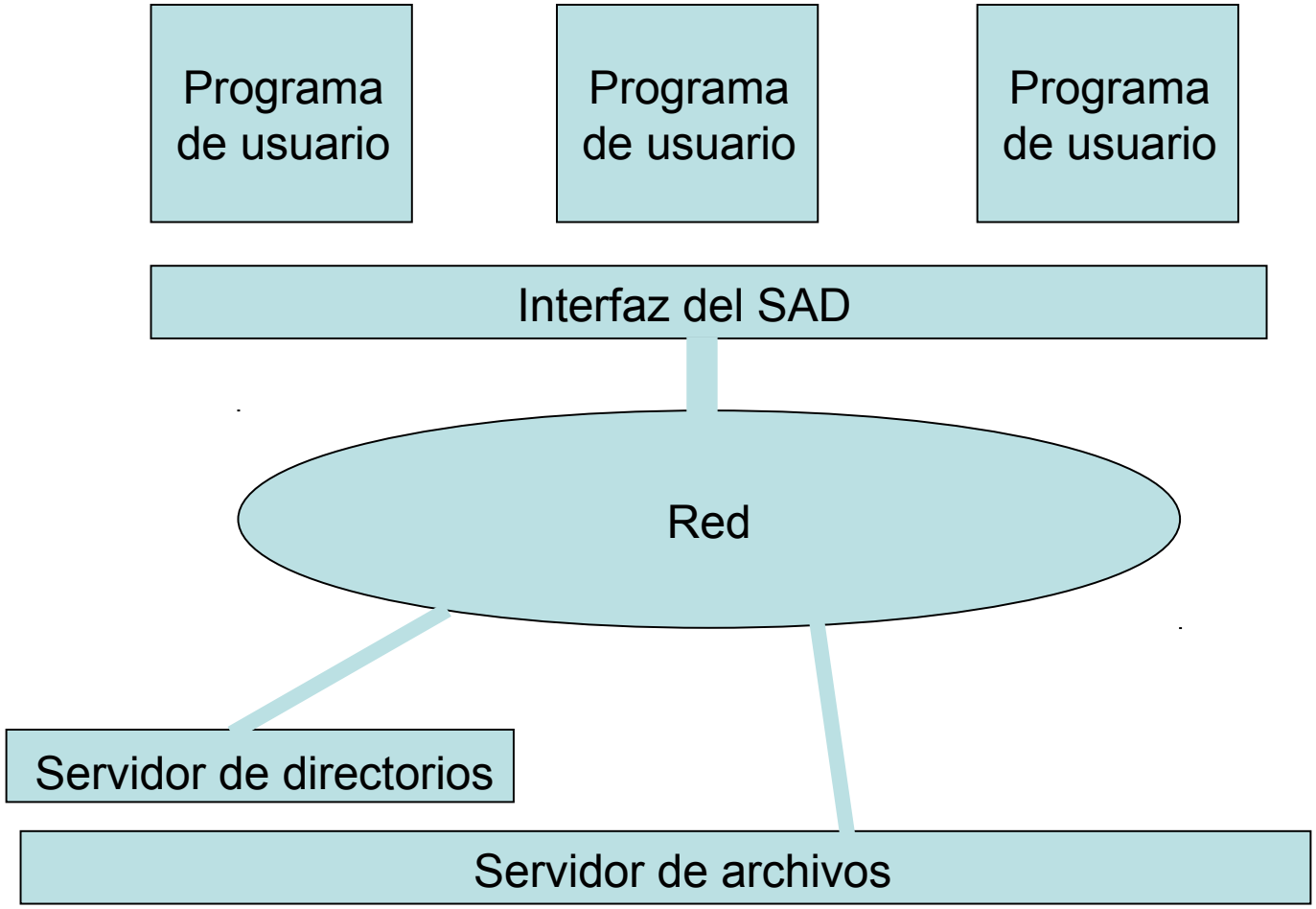
Conceptos básicos

- Sistema de archivos distribuido (SAD)
  - Objetivo principal: compartir datos entre usuarios ofreciendo transparencia
  - Objetivos secundarios:
    - disponibilidad
    - rendimiento (debería ser comparable al de un sistema tradicional)
    - tolerancia a fallos

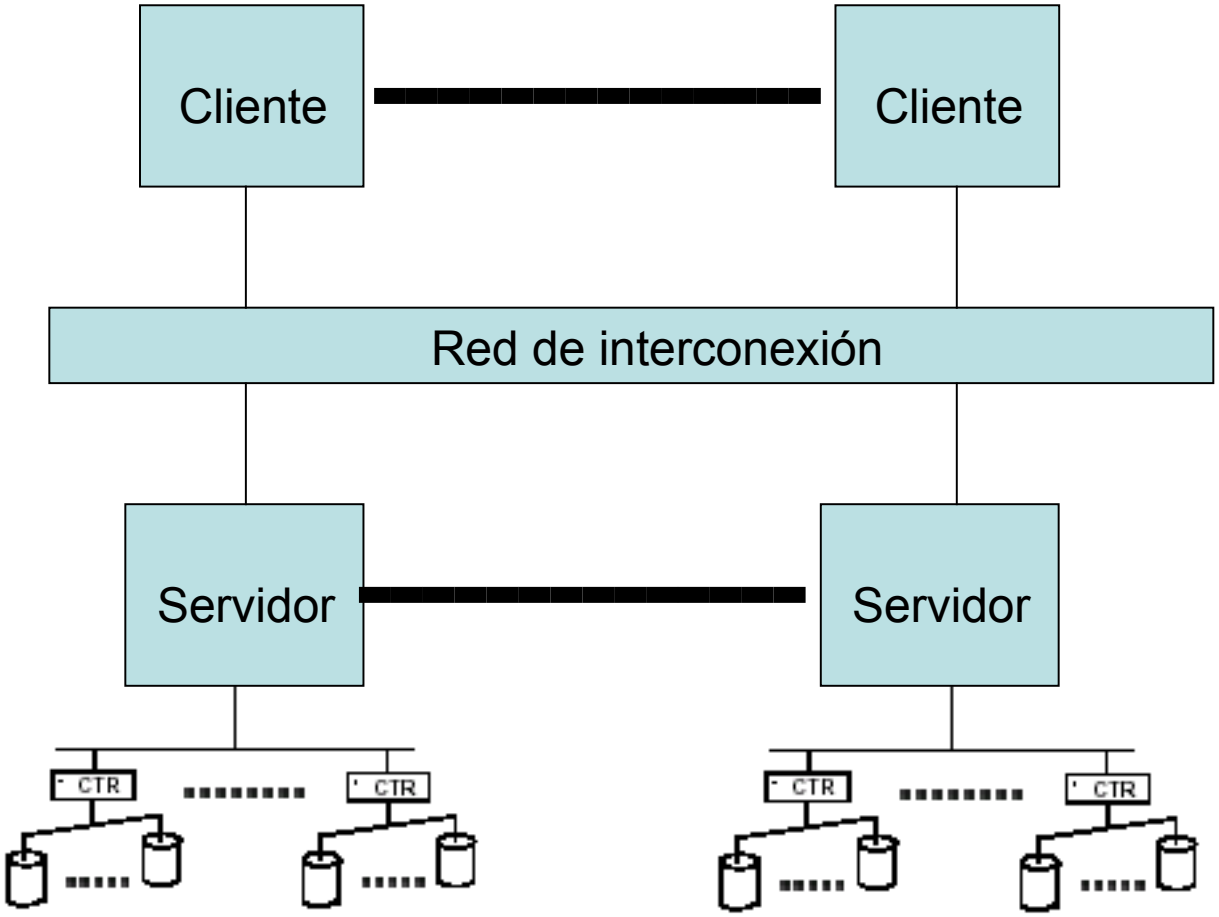
Conceptos básicos

- **Modelo cliente-servidor**
  - Servicios del sistema de archivos. Operaciones proporcionadas a los clientes
  - Servidores del sistema de archivos. Procesos de usuario o del sistema que ofrecen los servicios correspondientes (servidores multithread)
- **Transparencia**
  - Mismas operaciones para acceso locales y remotos.
  - Imagen única del sistema de archivos.
- **Rendimiento**. Un SAD tiene sobrecargas adicionales.
  - Red de comunicación, protocolos, posible necesidad de realizar más copias, etc.
- **Facilidad de crecimiento**. Eliminar los cuellos de botella
- **Tolerancia a fallos**: replicación, funcionamiento degrado.

Componentes de un SAD



Estructura de un SAD



Servicio de directorio

- Se encarga de la traducción del nombres de usuario a nombres internos
- Directorio: relaciona de forma única nombres de archivos con nombres internos
- Dos opciones:
  - Los directorios son objetos independientes gestionados por un servidor de directorios (SD)
  - Los directorios son archivos especiales. Servidor de archivos y de directorios combinados

Gestión de nombres: principios básicos

- Sistema operativo distribuido: servicio uniforme de nombres para todos los objetos
- En muchos casos: diferentes esquemas para diferentes objetos (archivos). Varios servidores de nombres
- Transparencia de la posición: el nombre del objeto no permite obtener directamente el lugar donde está 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 archivo
      - Ni transparencia, ni independencia
    - Montar un sistema de archivos remoto sobre la jerarquía local (NFS)
      - Espacio de nombres diferente en cada máquina
    - Único espacio de nombres en todas las máquinas
      - Proporciona transparencia
- Nombres internos: identificador único de archivo utilizado por el sistema

Servicio de archivos

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

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 archivo
- **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
  - Difícil de implementar en sistemas distribuidos
    - Mantener una copia única

Semánticas de coutilización

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

Métodos de acceso a archivos

- **Modelo carga/descarga**
  - Transferencias completas del archivo
  - 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 archivo
- **Modelo de servicios remotos**
  - El servidor debe proporcionar todas las operaciones sobre el archivo.
  - Acceso por bloques
  - Modelo cliente/servidor
- **Empleo de caché en el cliente**
  - Combina los dos modelos anteriores.

Tipos de servidores

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

Tipos de servidores

- **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

Caché de bloques

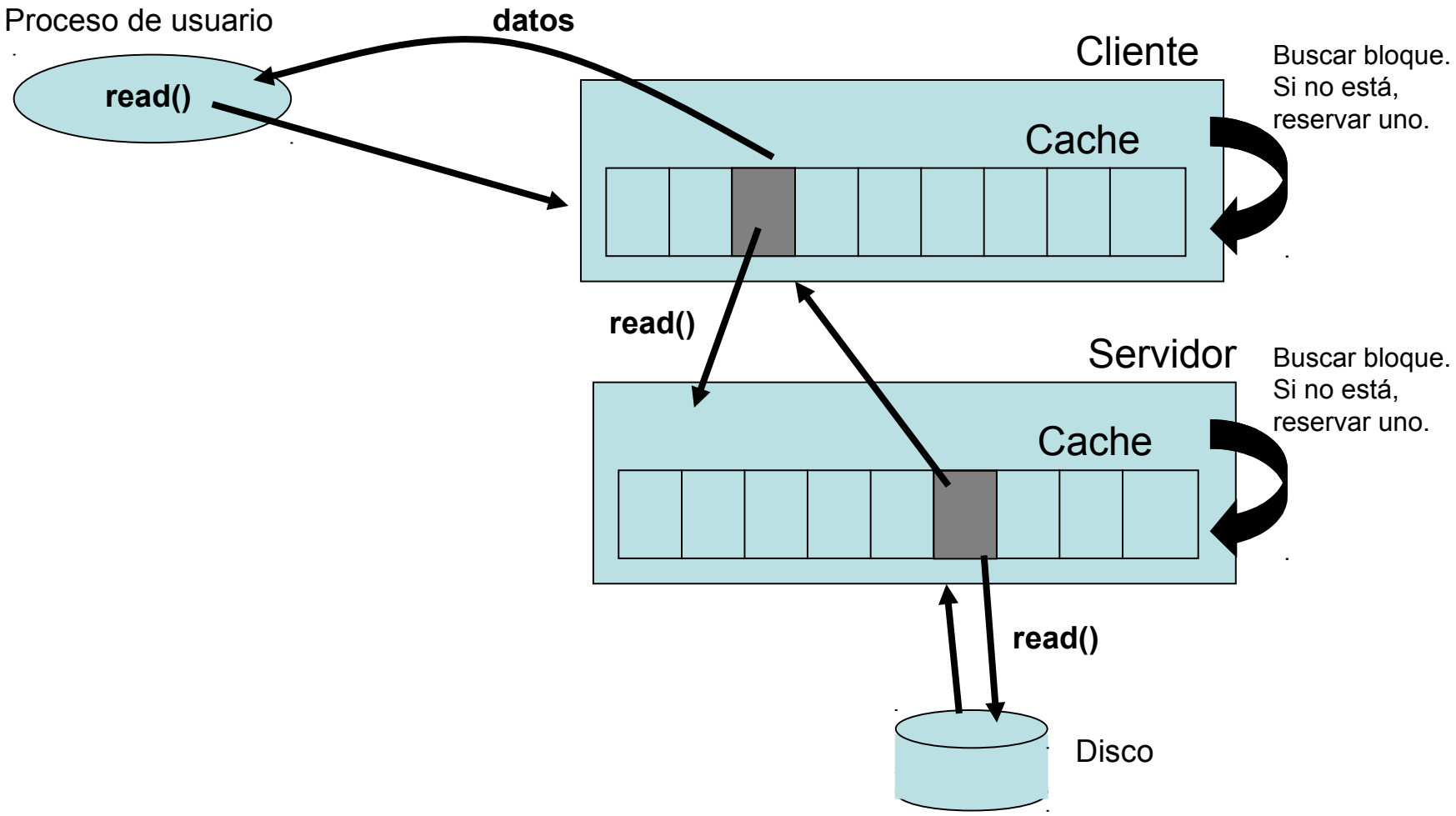
- El empleo de cache de bloques permite mejorar el rendimiento
  - Explota el principio de proximidad de referencias
    - Proximidad temporal
    - Proximidad espacial
  - Lecturas adelantadas
    - Mejora el rendimiento de las operaciones de lectura, sobre todo si son secuenciales
  - Escrituras diferidas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

Localización de las cache en un SAD

- **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



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
  - 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

Problema de la coherencia de caché

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

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
- Empleo de protocolos de coherencia de caché

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

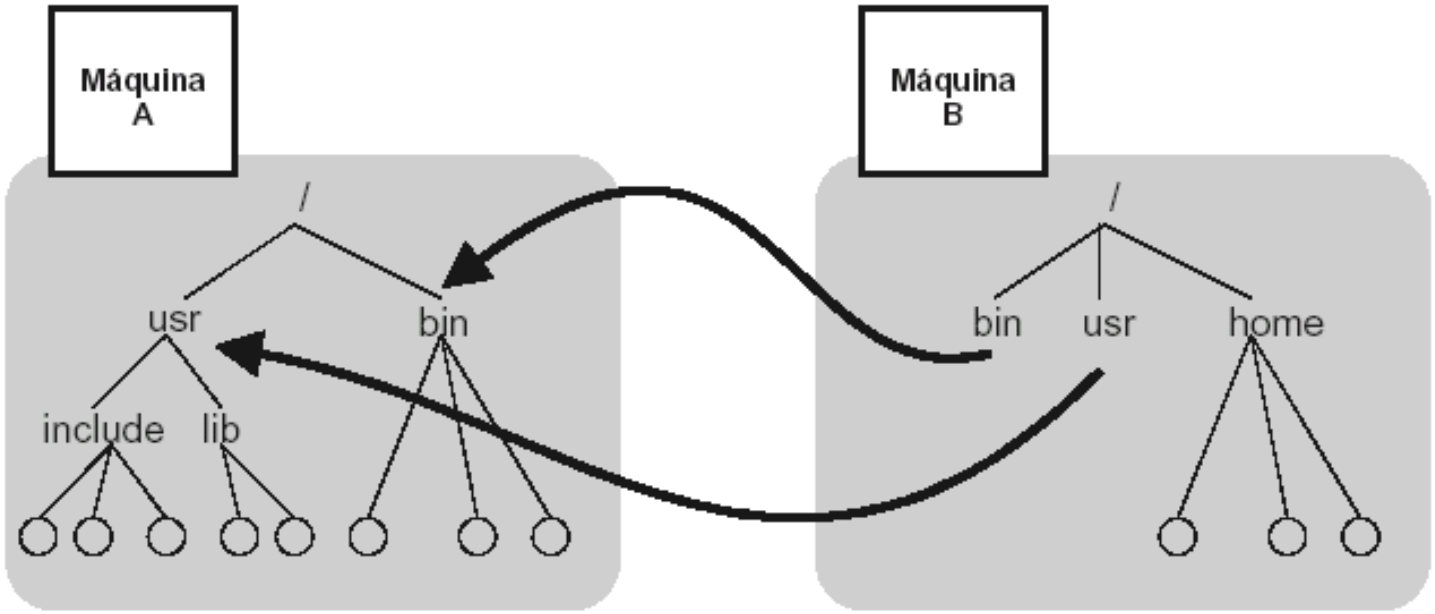
NFS: Network File System

- Implementación y especificación de un software de sistema para acceso a archivos 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 (eXternal Data Representation)
- Las diferentes máquinas montan un directorio remoto en el sistema de archivos local
  - El espacio de nombres es diferente en cada máquina
  - El montado no es transparente, debe proporcionarse el nombre de la máquina remota
- No es un verdadero sistema de archivos distribuido

Montado en NFS

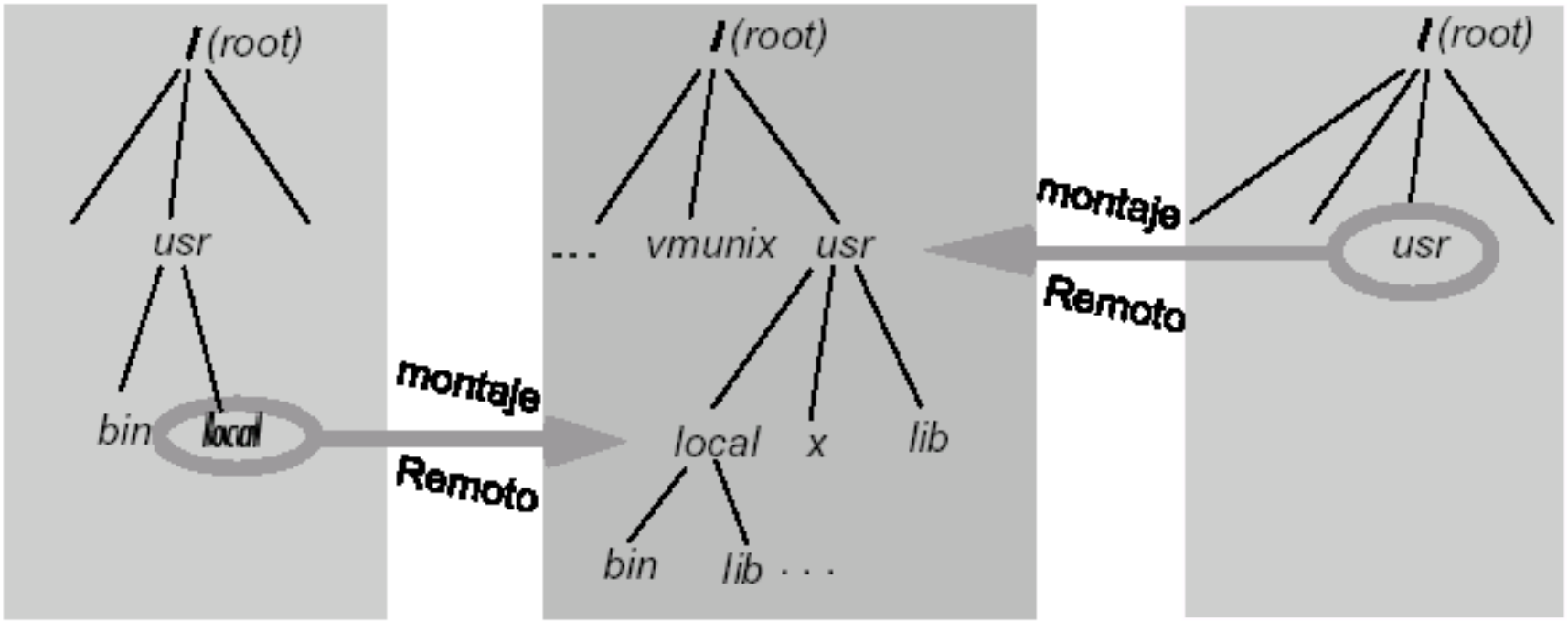
- Establece una conexión lógica entre el servidor y el cliente
- La máquina A exporta /usr y /bin
- En la máquina B:

```
mount maquinaA:/usr /usr
```



Montado en NFS

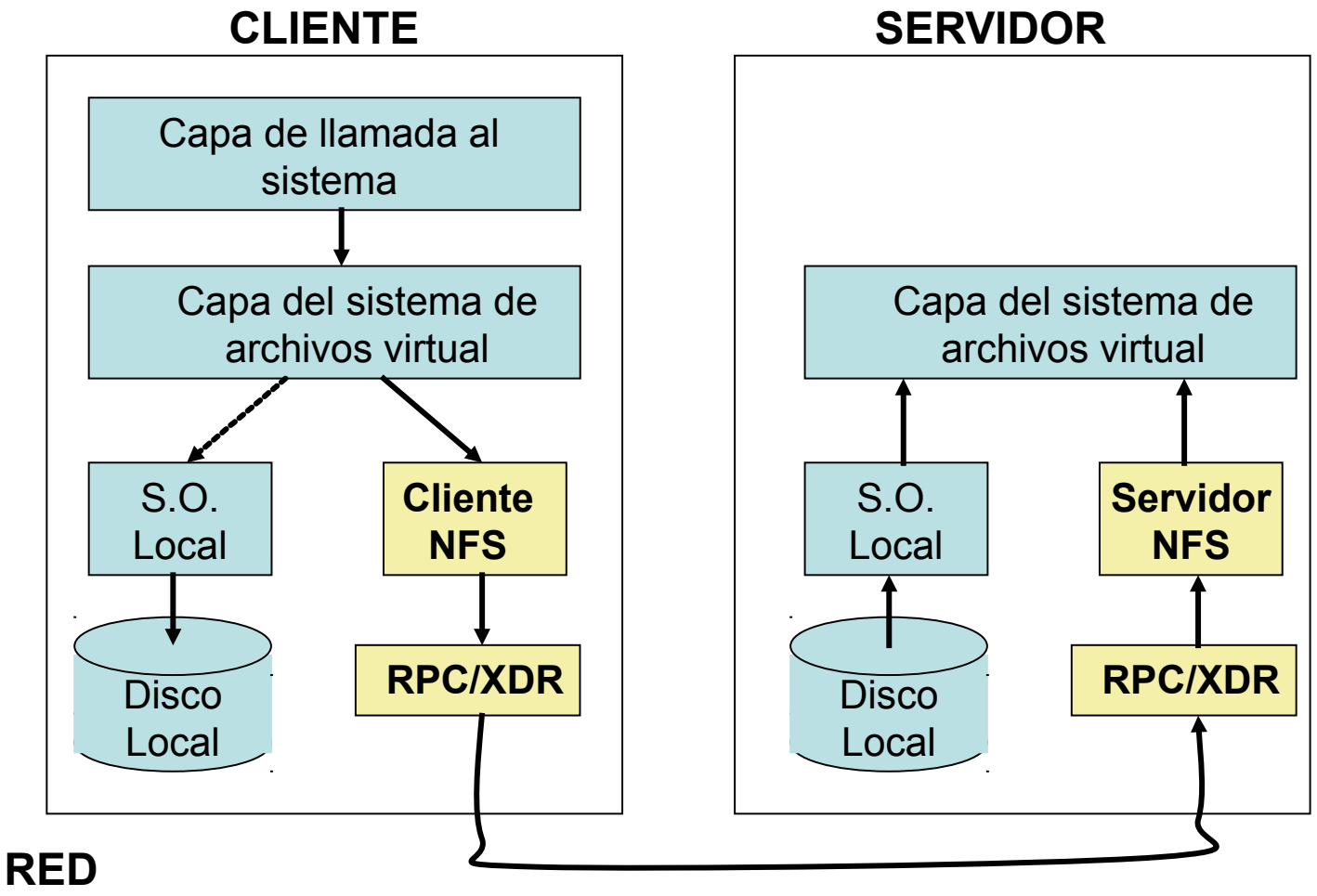
Imagen diferente del sistema de archivos



Protocolo NFS

- Ofrece un conjunto de RPC para realizar operaciones sobre archivos remotos
  - Búsqueda de un archivo en un directorio
  - Lectura de entradas de directorio
  - Manipulación de enlaces y directorios
  - Acceso a los atributos de un archivo
  - Lectura y escritura de archivos
- Los servidores de NFS no almacenan estado
  - Operaciones autocontenidas
- El protocolo no ofrece mecanismos de control de concurrencia para asegurar una semántica UNIX

Arquitectura de NFS



Traducción de nombres

- VFS almacena una entrada por cada archivo abierto (*vnode*)
- Cada *vnode* apunta a un inodo local o a un inodo remoto
- El cliente realiza la traducción de un nombre (path) componente a componente. Cuando un *vnode* apunta a inodo remoto la búsqueda se realiza en el servidor (un componente cada vez)
- El cliente almacena una cache de nombres para acelerar las traducciones (cache de atributos)
  - Las entradas se validan si llevan más de 30s

Acceso a los archivos

- Las transferencias se realizan en bloques de 8 KB
- Los bloques se almacenan en la cache de los clientes
- Los clientes realizan lecturas adelantadas de un bloque
- Las escrituras se realizan localmente. Los bloques se envían al servidor cuando se llena un bloque de 8 KB o cuando se cierra el archivo
- Problemas de coherencia de cache (no se asegura la semántica UNIX)
- ¿Cómo intenta conseguir una semántica UNIX?
  - Cuando un archivo se abre en un cliente se comprueba en el servidor si la información se ha modificado
  - Cuando se accede a un bloque que lleva más de 3s en la cache del cliente se valida

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;
        RMDIR3res NFSPROC3_RMDIR(RMDIR3args) = 13;
        RENAME3res NFSPROC3_RENAME(RENAME3args) = 14;
        LINK3res NFSPROC3_LINK(LINK3args) = 15;
        READDIR3res NFSPROC3_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<>;
};
```