Gestión de Memoria DSM

6

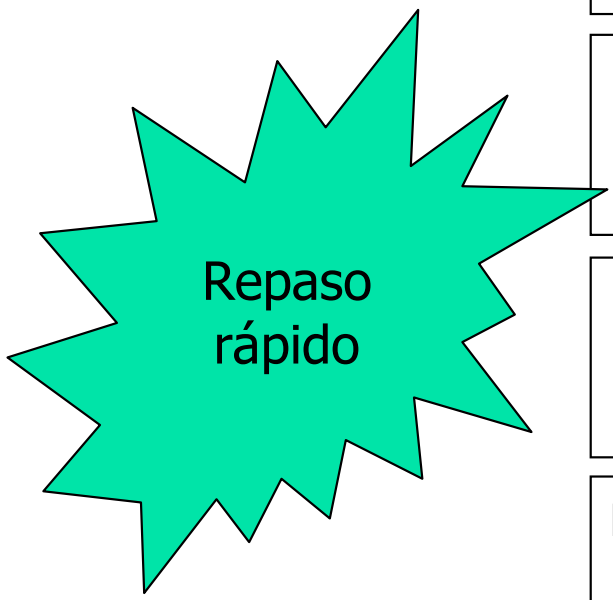
Sistemas Operativos y Distribuidos

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

Mapa Conceptual de esta parte de la clase

<table>
<tr><td>Real</td><td colspan="2">Real</td><td colspan="2">Real</td><td colspan="2">Virtual</td></tr>
<tr><td>Mono Usuario</td><td colspan="2">Multiprogramación</td><td colspan="2">Multiprogramación</td><td colspan="2">Multiprogramación</td></tr>
<tr><td></td><td colspan="2">Particionamiento</td><td>Paginación Simple</td><td>Segmentación Simple</td><td>Paginación Virtual</td><td>Segmentación Virtual</td></tr>
<tr><td></td><td>Fija</td><td>Dinámica</td><td colspan="2">Combinación</td><td colspan="2">Combinación</td></tr>
<tr><td></td><td colspan="6">Reubicación, Protección</td></tr>
</table>

Organización Física de la Memoria



Registros de la CPU

Cache
Nivel 1
Nivel 2

RAM
Memoria Física
Memoria Virtual

Almacenamiento
Temporal o
de acceso aleatorio

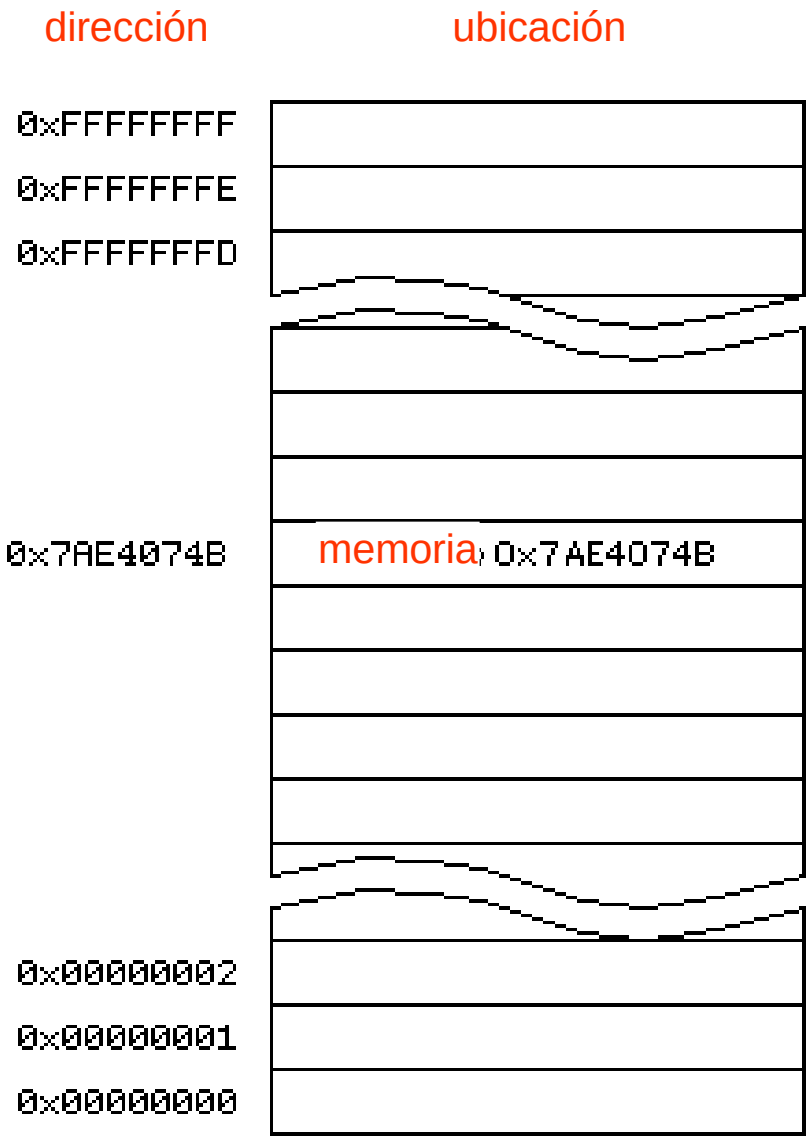
Hard Disk
Drive
Flash Memory

Almacenamiento
Permanente

Dispositivos Externos
Teclado
Mouse
Scaner

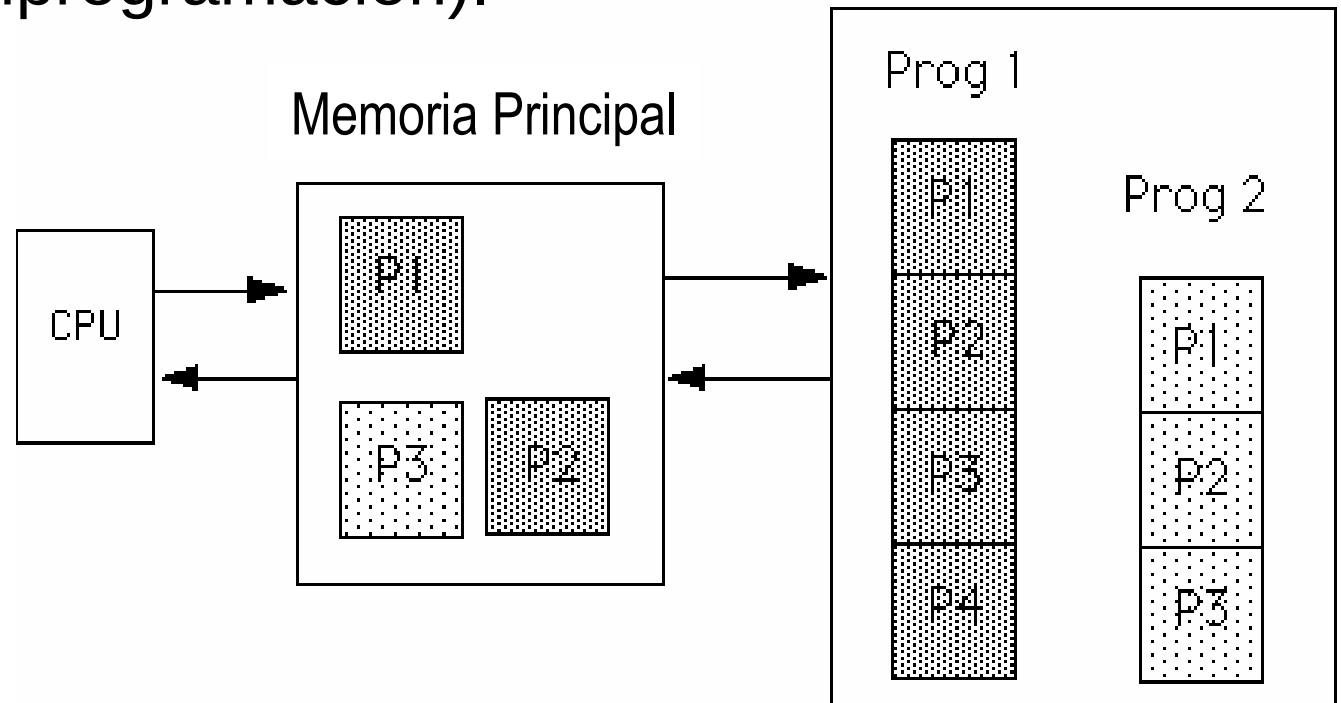
Organización Lógica de la Memoria

- La memoria principal es un arreglo de palabras o bytes, cada uno de los cuales tiene una dirección (espacio de direcciones).
- La interacción es lograda a través de un conjunto de lecturas y escrituras a direcciones especificas realizadas por los procesos.



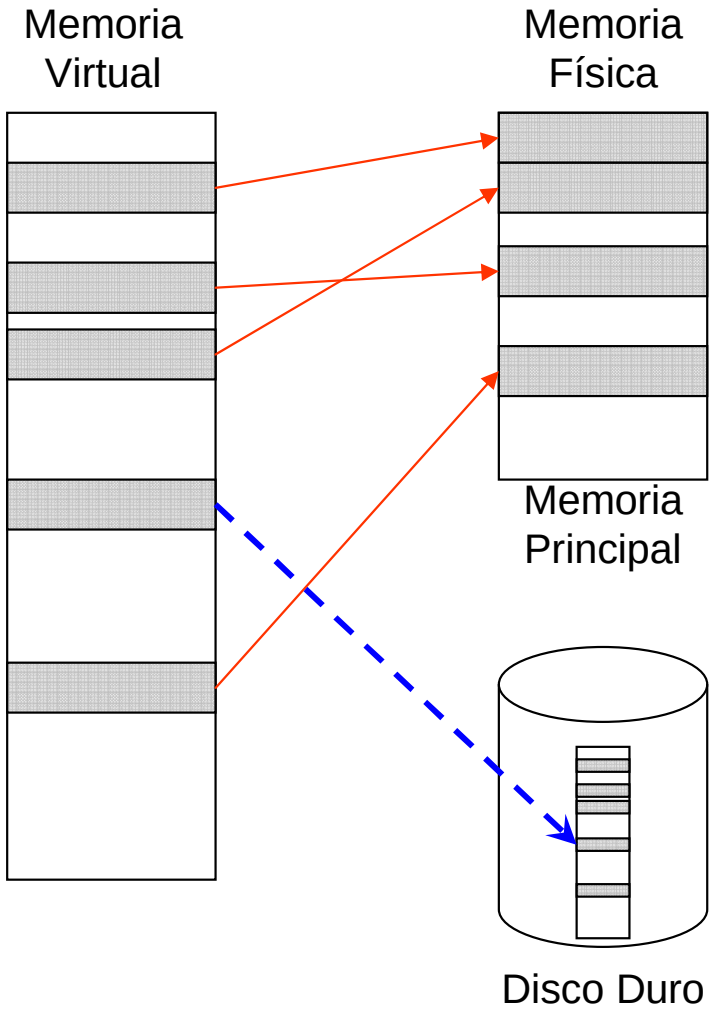
Procesos y Memoria

- Para que un proceso se ejecute se requiere ubicarlo en memoria principal junto con los datos que direcciona.
- Para optimizar el uso de la computadora se requiere tener varios procesos en memoria principal (grado de multiprogramación).

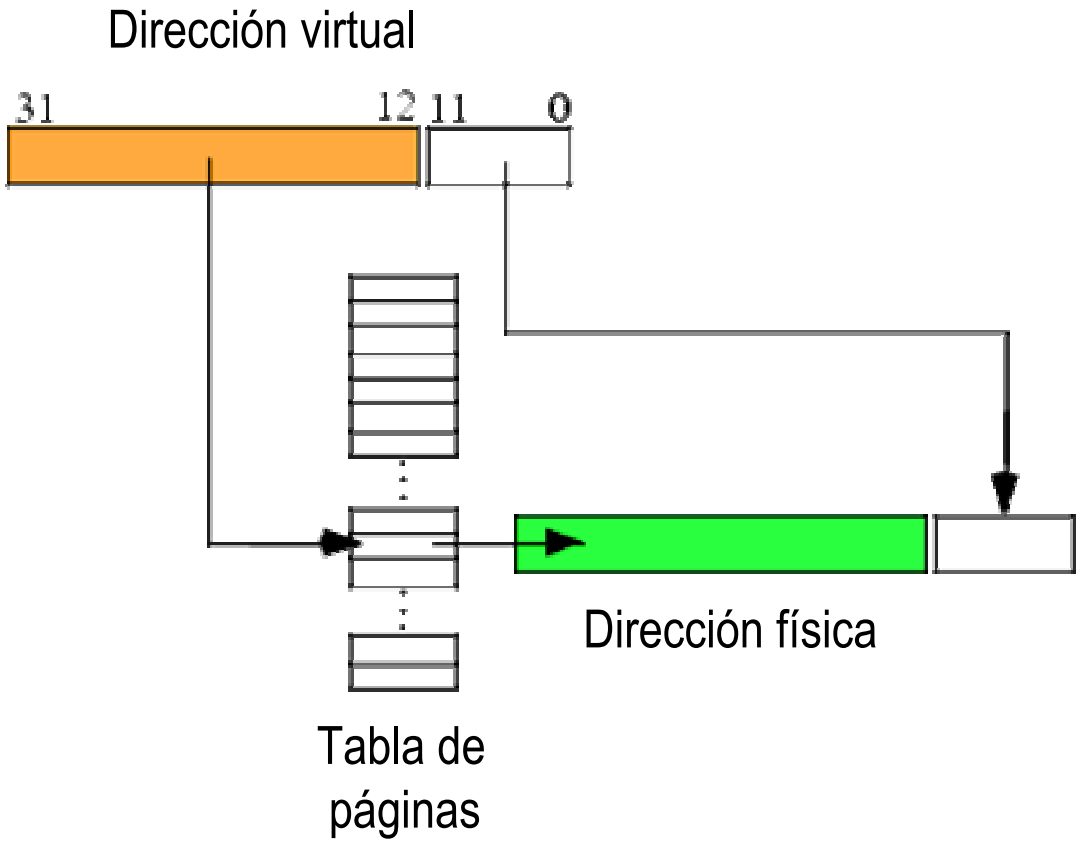


Memoria Virtual

- La memoria principal es pequeña como para acomodar todos programas y datos permanentemente.
- Por lo que es necesario implementar mecanismos de memoria virtual.
- La memoria virtual es una técnica para dar la ilusión de tener más memoria que la memoria principal.

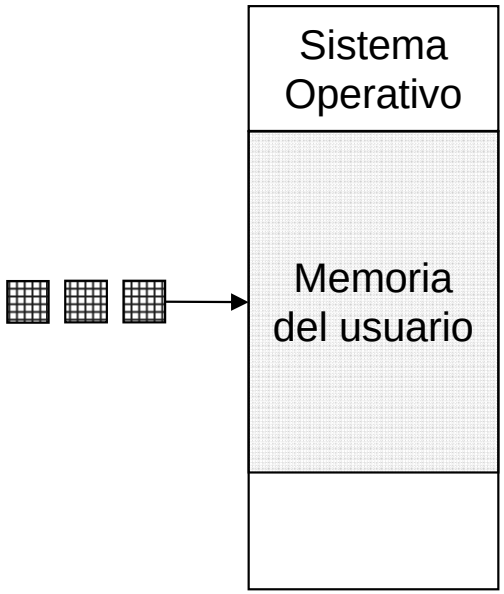


Administrador de memoria



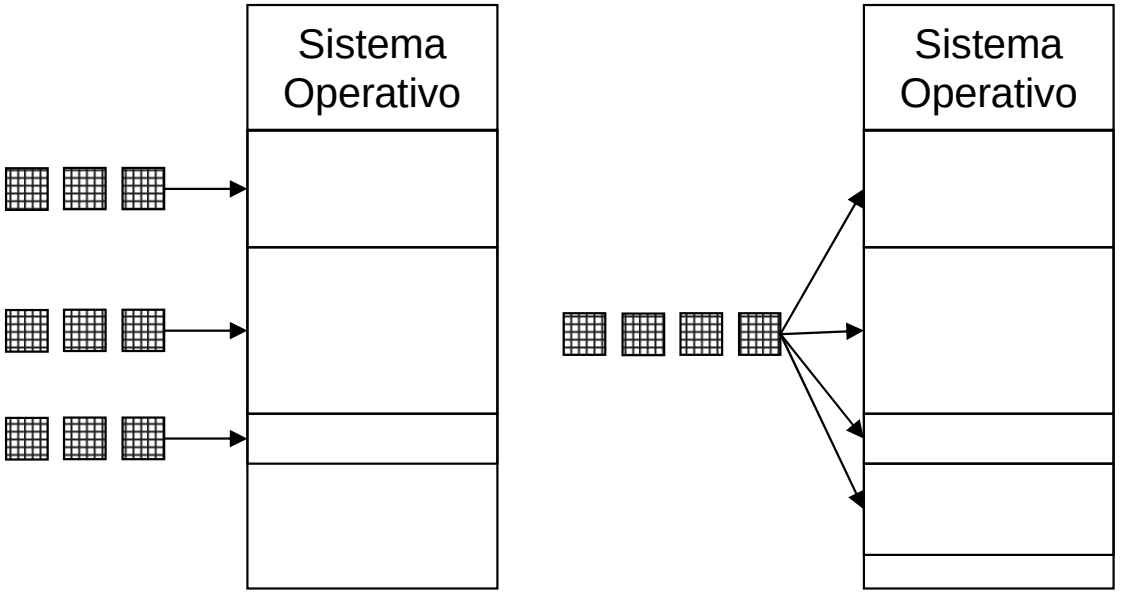
Administrador de Memoria

Sistema monoprogramado



Un programa puede o no ingresar a una única partición de memoria

Sistema multiprogramado



Múltiples programas comparten diversas particiones de memoria
Particiones de tamaño fijo
Particiones de tamaño variable

Administrador de Memoria

- El administrador de memoria tiene como objetivos:
  - Ubicar, reemplazar, cargar y descargar procesos en la memoria principal.
  - Proteger la memoria de acceso indeseados (accidentales o intencionados).
  - Permitir la compartición (*sharing*) de zonas de memoria (indispensable para lograr la cooperación de procesos).

Requisitos del administrador de memoria

1. **Reubicación.** Permitir el recalculo de direcciones de memoria de un proceso reubicado.
2. **Protección.** Evitar el acceso a posiciones de memoria sin el permiso expreso. (no direcciones absolutas).
3. **Sharing.** Permitir a procesos diferentes acceder a la misma porción de memoria.
4. **Organización Lógica.** Permitir que los programas se escriban como módulos compilables y ejecutables por separado.
5. **Organización Física.** Permitir el intercambio de datos en la memoria primaria y secundaria

Estrategias

Están dirigidas a la obtención del mejor uso del recurso memoria principal, estas pueden ser:

1. Estrategia de solicitud (búsqueda)

   (cuando obtener un fragmento de programa)

   - Estrategias de búsqueda por demanda.
   - Estrategias de búsqueda anticipada.

2. Estrategia de ubicación.

   (donde se colocará (cargar) un fragmento de programa nuevo)

3. Estrategia de reposición.

   (qué fragmento de programa elimina, para cargar uno nuevo)

Administrador de Memoria

- Las técnicas usadas son las siguientes:
  1. Partición Fija
  2. Partición Dinámica
  3. Paginación Simple
  4. Segmentación Simple
  5. Memoria Virtual Paginada
  6. Memoria Virtual Segmentada

TECNICAS DE ADMINISTRACION DE MEMORIA

PARTICIONAMIENTO

<table>
<tr><td>Real</td><td colspan="2">Real</td><td colspan="2">Real</td><td colspan="2">Virtual</td></tr>
<tr><td>Mono Usuario</td><td colspan="2">Multiprogramación</td><td colspan="2">Multiprogramación</td><td colspan="2">Multiprogramación</td></tr>
<tr><td></td><td colspan="2">Particionamiento</td><td>Paginación Simple</td><td>Segmentación Simple</td><td>Paginación Virtual</td><td>Segmentación Virtual</td></tr>
<tr><td></td><td>Fija</td><td>Dinámica</td><td colspan="2">Combinación</td><td colspan="2">Combinación</td></tr>
<tr><td></td><td colspan="6">Reubicación, Protección</td></tr>
</table>

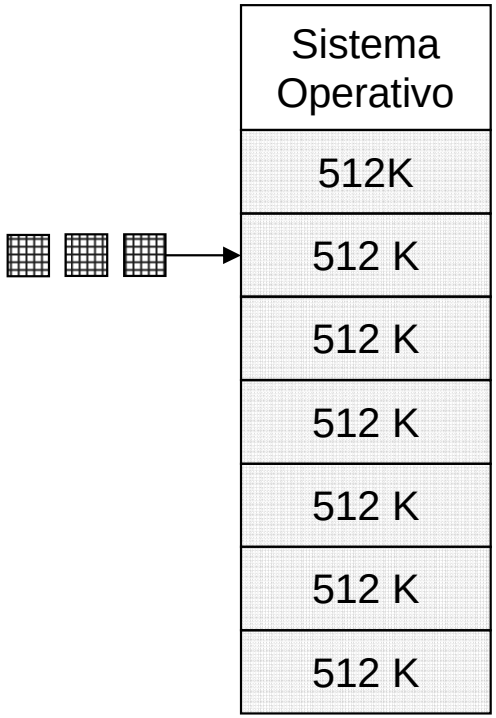
1. Partición Fija

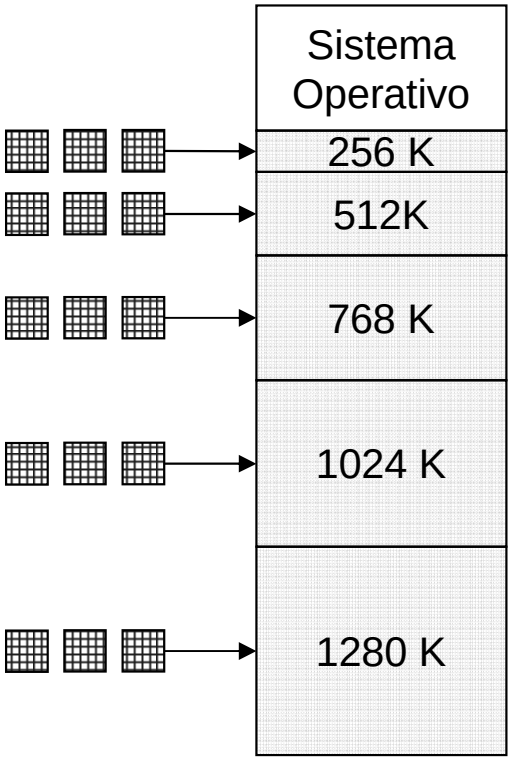
- La memoria principal se divide en un conjunto de particiones de tamaño fijo durante el inicio del sistema.
- Un proceso se puede cargar completamente en una partición de tamaño menor o igual.
- Ventajas. Sencilla de implementar. Poca sobrecarga al SO.
- Desventajas. Fragmentación interna. Nro. fijo de procesos activos.

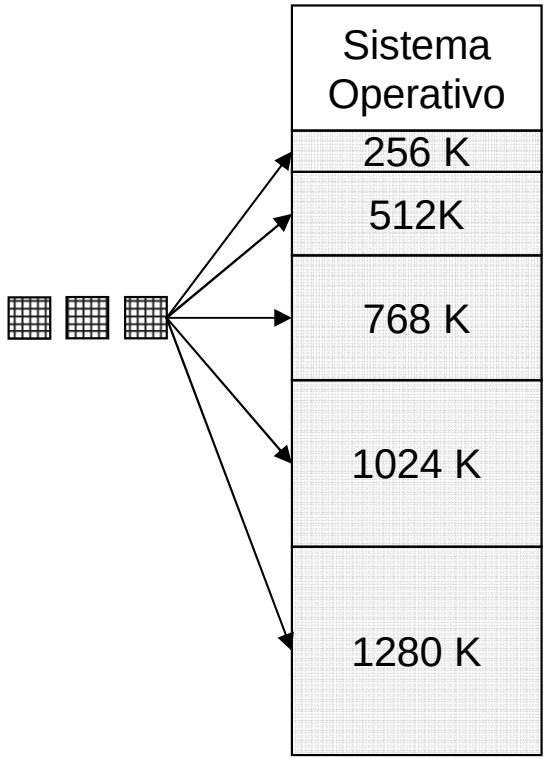
1. Estrategias

- Solicitud.
  - Por demanda
- Ubicación.
  - Partición de igual tamaño.
    - Si el proceso cabe en una partición se puede cargar
  - Partición de diferente tamaño.
    - Asignar a la partición más pequeña.
    - Se genera dos tipos de colas: una cola, varias colas
- Reemplazo.
  - Uno de los proceso se saca, según el planificador.

1. Estrategia de Ubicación







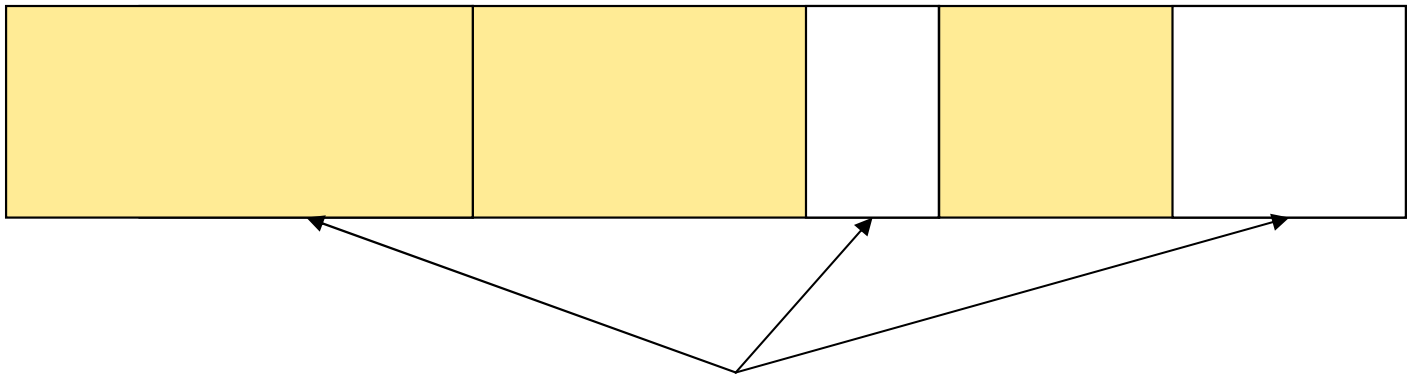
Particiones del mismo tamaño

Particiones de distinto tamaño

1. Partición Fija

- Si un programa no cabe en una partición, el programador debe diseñarlo en módulos cargables.

- El uso de la memoria es muy ineficiente, no importa el tamaño del proceso, ocupara toda la partición, se genera fragmentación interna.



fragmentación interna

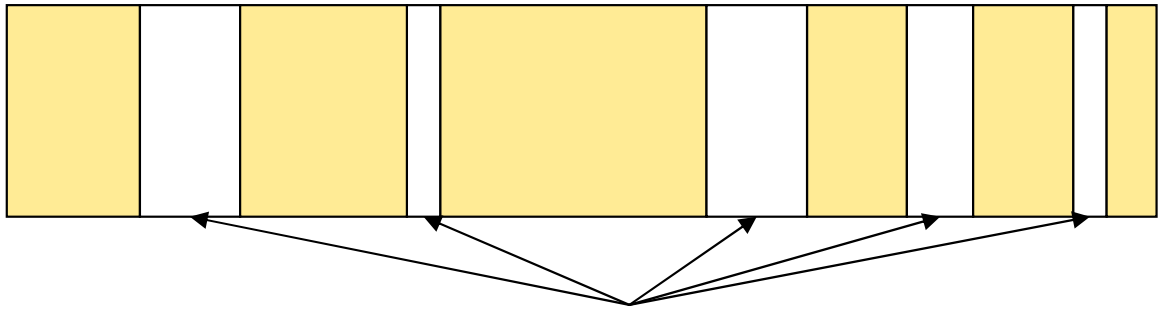
2. Partición Dinámica

- Las particiones se crean dinámicamente por demanda.
- Son variables en tamaño y número.
- Cada proceso se carga completamente en una única partición del tamaño del proceso.

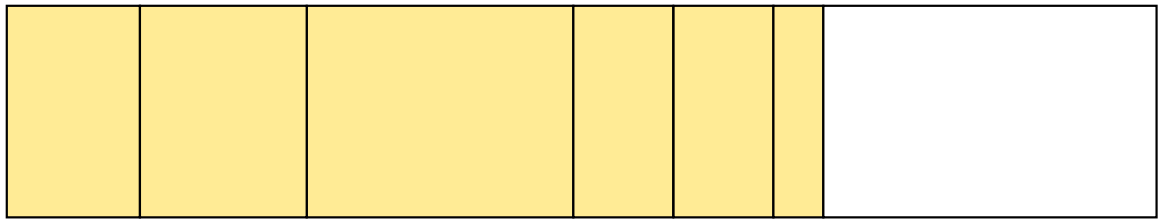
- Ventajas. No existe fragmentación interna.
- Desventajas. Fragmentación externa. Se debe compactar la memoria. El compactado toma tiempo.

2. Partición Dinámica

- El uso de la memoria es muy ineficiente, se generan muchos huecos entre las particiones, cada vez más pequeñas, se genera la fragmentación externa.
- Cada cierto tiempo se debe compactar los segmentos libres, para que estén contiguos.



fragmentación externa



compactación

2. Estrategias

- Solicitud.
  - Por demanda
- Ubicación.
  - Primer ajuste. El primer bloque disponible que ubique (parte del inicio)
  - Siguiente ajuste. El siguiente bloque disponible que ubique (parte desde la ubicación actual)
  - Mejor ajuste. El bloque disponible que deje el menor espacio libre (búsqueda exhaustiva)
- Reemplazo.
  - Uno de los proceso se saca, según el planificador.

2. Estrategias

- Primer ajuste. Es bueno, con baja compactación. Puebla el inicio de la memoria.

- Siguiente ajuste. Puebla el final de la memoria, el siguiente bloque libre siempre está al final de la memoria.

- Mejor ajuste. Tiene peores resultados, dado que busca la partición que deje el hueco más pequeño, la memoria se llena de huecos pequeños. Se compacta con más frecuencia

TECNICAS DE ADMINISTRACION DE MEMORIA

PAGINACION Y SEGMENTACION SIMPLE

<table>
<tr><td>Real</td><td colspan="2">Real</td><td colspan="2">Real</td><td colspan="2">Virtual</td></tr>
<tr><td>Mono Usuario</td><td colspan="2">Multiprogramación</td><td colspan="2">Multiprogramación</td><td colspan="2">Multiprogramación</td></tr>
<tr><td rowspan="3"></td><td colspan="2">Particionamiento</td><td>Paginación Simple</td><td>Segmentación Simple</td><td>Paginación Virtual</td><td>Segmentación Virtual</td></tr>
<tr><td>Fija</td><td>Dinámica</td><td colspan="2">Combinación</td><td colspan="2">Combinación</td></tr>
<tr><td colspan="6">Reubicación, Protección</td></tr>
</table>

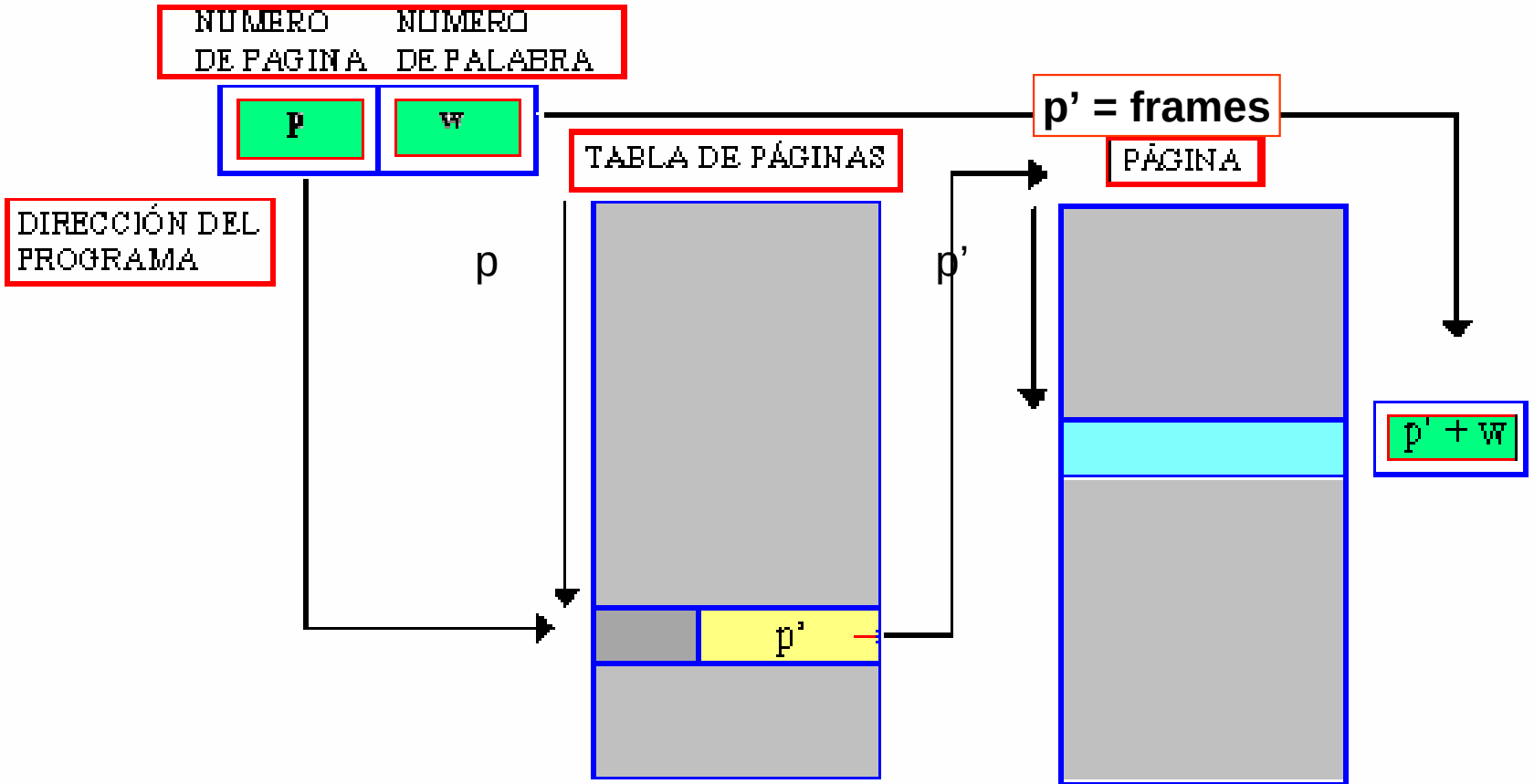
3. Paginación Simple

- La memoria principal se divide en un conjunto de frames de igual tamaño.
- Cada proceso se divide en una serie de páginas del tamaño de los frames.
- Un proceso se carga en los frames que requiera (todas las páginas), no necesariamente contiguos.

- Ventajas. No hay fragmentación externa
- Desventajas. Fragmentación interna pequeña.

3. Paginación Simple

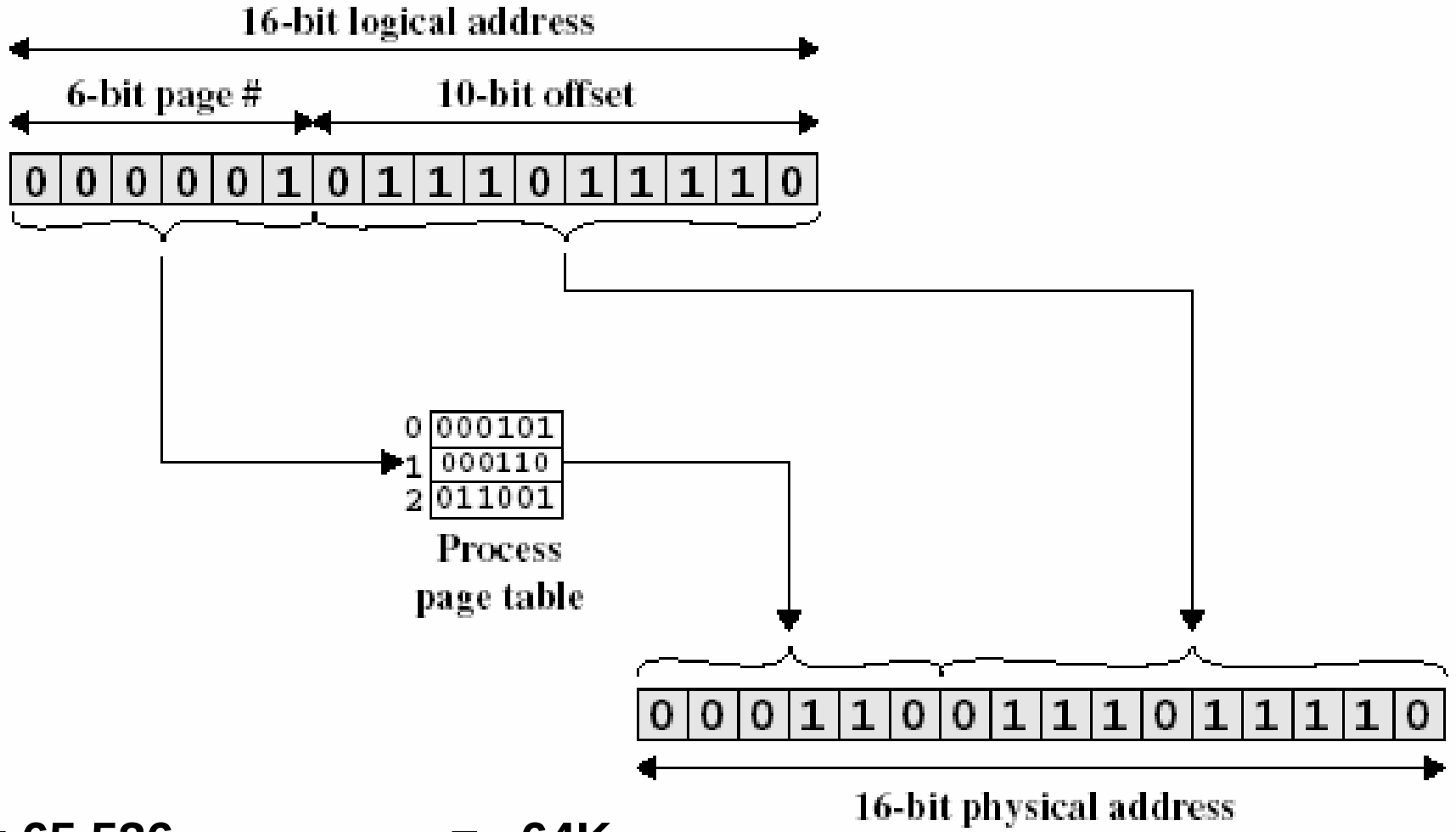
- El SO mantiene una tabla de paginas para cada proceso, que contiene la lista de frames para cada pagina.
- Una dirección de memoria es un número de página (P) y un desplazamiento dentro de la página (W).



3. Estrategias

- Solicitud.
  - Por demanda
- Ubicación.
  - Se cargan todas las páginas de un proceso en los frames libres y se actualiza su tabla de páginas.
- Reemplazo.
  - Una de las páginas se puede sacar y se marca como que no está cargada. Esto es posible por que cada proceso tiene su propia tabla de páginas.
  - No es necesario sacar todas las páginas de un proceso.

3. Capacidad de Direccionamiento



2^{16} = 65,536 = 64K

2^{20} = 1'048,576 = 1MB

2^{24} = 16’777,216 =16MB

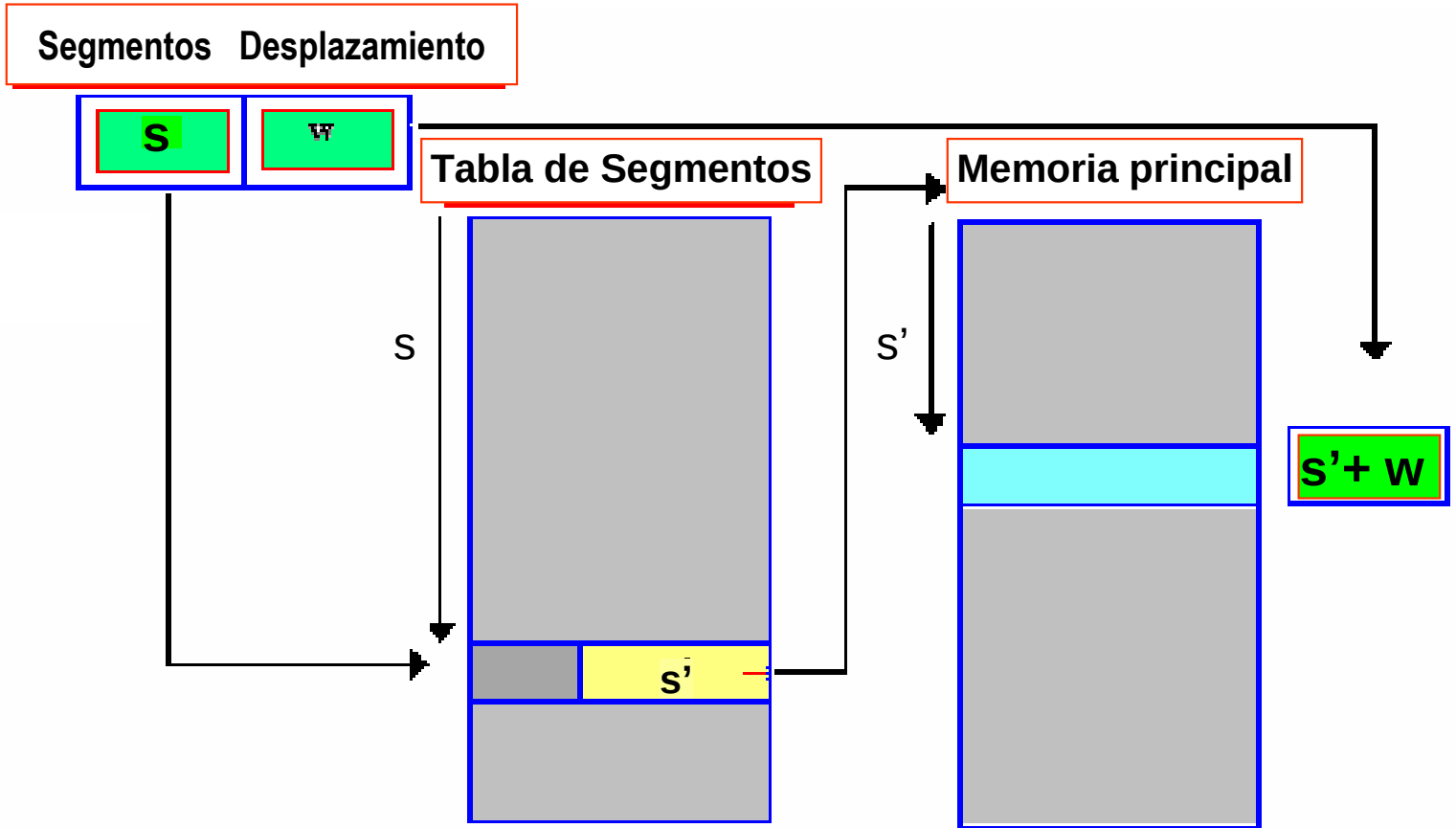
2^{32} = 4,294'967,296 = 4GB

4. Segmentación Simple

- Cada proceso y sus datos se dividen en segmentos de longitud variable.
- Un proceso carga sus segmentos en particiones dinámicas no necesariamente contiguas.
- Todos los segmentos de un proceso se deben de cargar en memoria.
- Se diferencia de la partición dinámica en que un proceso puede ocupar más de un segmento.
- Ventajas. No hay fragmentación interna.
- Desventajas. Fragmentación externa, pero menor (compactación).

4. Segmentación Simple

- El SO mantiene una tabla de segmentos para cada proceso y la lista de bloques libres.
- Una dirección de memoria es un número de segmento (S) y un desplazamiento dentro de segmento (W).



4. Estrategias

- Solicitud.
  - Por demanda
- Ubicación.
  - Se cargan los segmentos de un proceso en los bloques libres y se actualiza su tabla de segmentos.
- Reemplazo.
  - Uno de los segmentos se puede sacar y se marca como que no está cargada. Esto es posible por que cada proceso tiene su propia tabla de segmentos.

4. Validación del Direccionamiento

- No hay correspondencia entre dirección lógica y dirección física.
- El SO trabaja con direcciones lógicas.
- El SO debe asegurar que cada dirección lógica esté dentro del rango de direcciones del proceso.
- El SO implementa la tabla de segmentos como un arreglo de registros base-limite.

La segmentación por lo general es invisible al programador. Es el compilador el que define los segmentos.

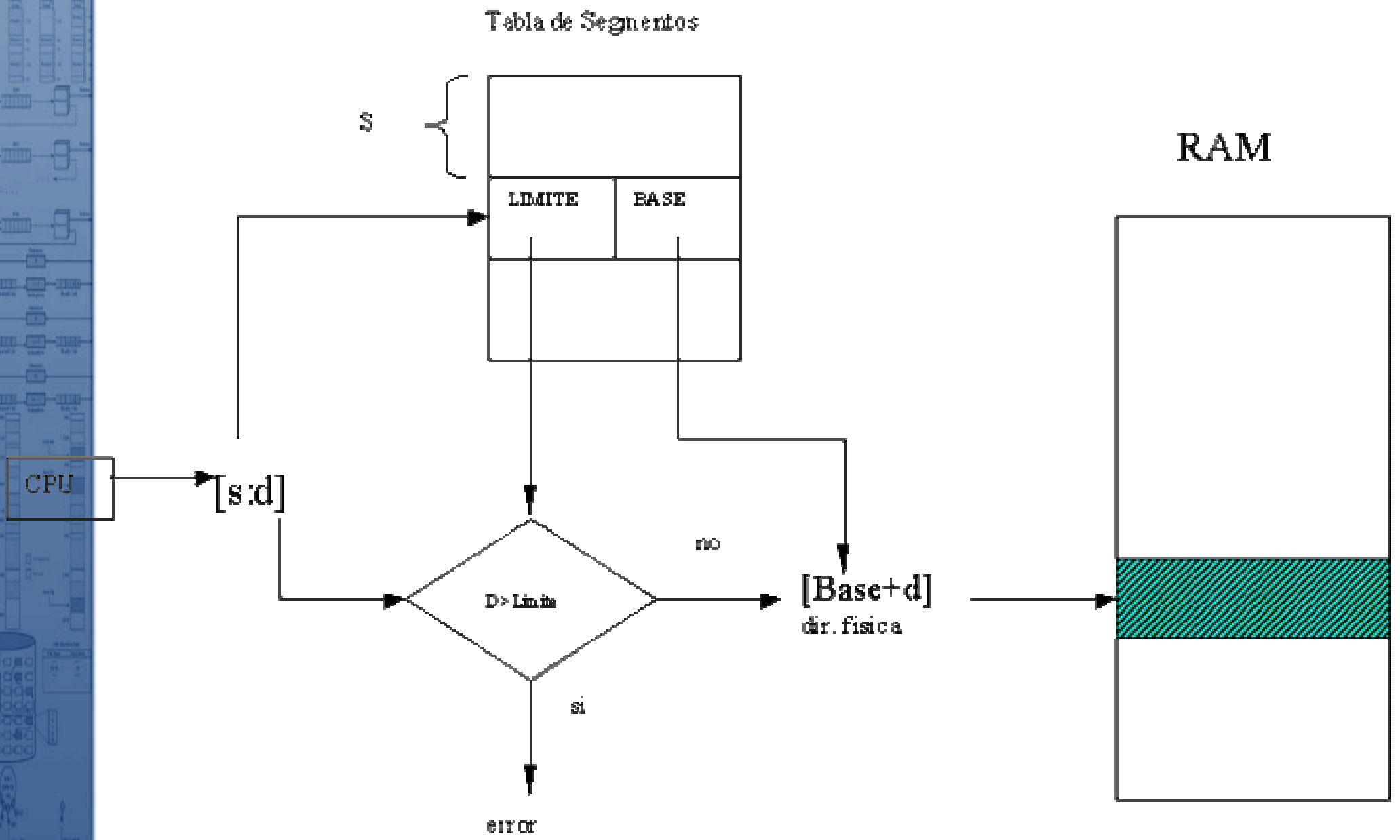


Tabla de Segmentos
S
LIMITE
BASE
RAM
CPU
[s:d]
no
D> Limite
[Base+d]
dir. fisica
si
error

CONCLUSIONES

1. El SAM particionado a diferencia de la paginación o segmentación simple, permite que sólo un proceso se cargue en memoria principal.
2. Cuando se trabaja con bloques de tamaño fijo se genera la fragmentación interna. Si los bloques son de tamaño variable, se genera la fragmentación externa.
3. El SAM de particiones fijas se parece al SAM de paginación simple, diferenciándose en que los primeros requieren que las particiones estén contiguas

Mapa Conceptual de esta parte de la clase

<table>
<tr><td>Real</td><td colspan="2">Real</td><td colspan="2">Real</td><td colspan="2">Virtual</td></tr>
<tr><td>Mono Usuario</td><td colspan="2">Multiprogramación</td><td colspan="2">Multiprogramación</td><td colspan="2">Multiprogramación</td></tr>
<tr><td></td><td colspan="2">Particionamiento</td><td>Paginación Simple</td><td>Segmentación Simple</td><td>Paginación Virtual</td><td>Segmentación Virtual</td></tr>
<tr><td></td><td>Fija</td><td>Dinámica</td><td colspan="2">Combinación</td><td colspan="2">Combinación</td></tr>
<tr><td></td><td colspan="6">Reubicación, Protección</td></tr>
</table>

TECNICAS DE ADMINISTRACION DE MEMORIA

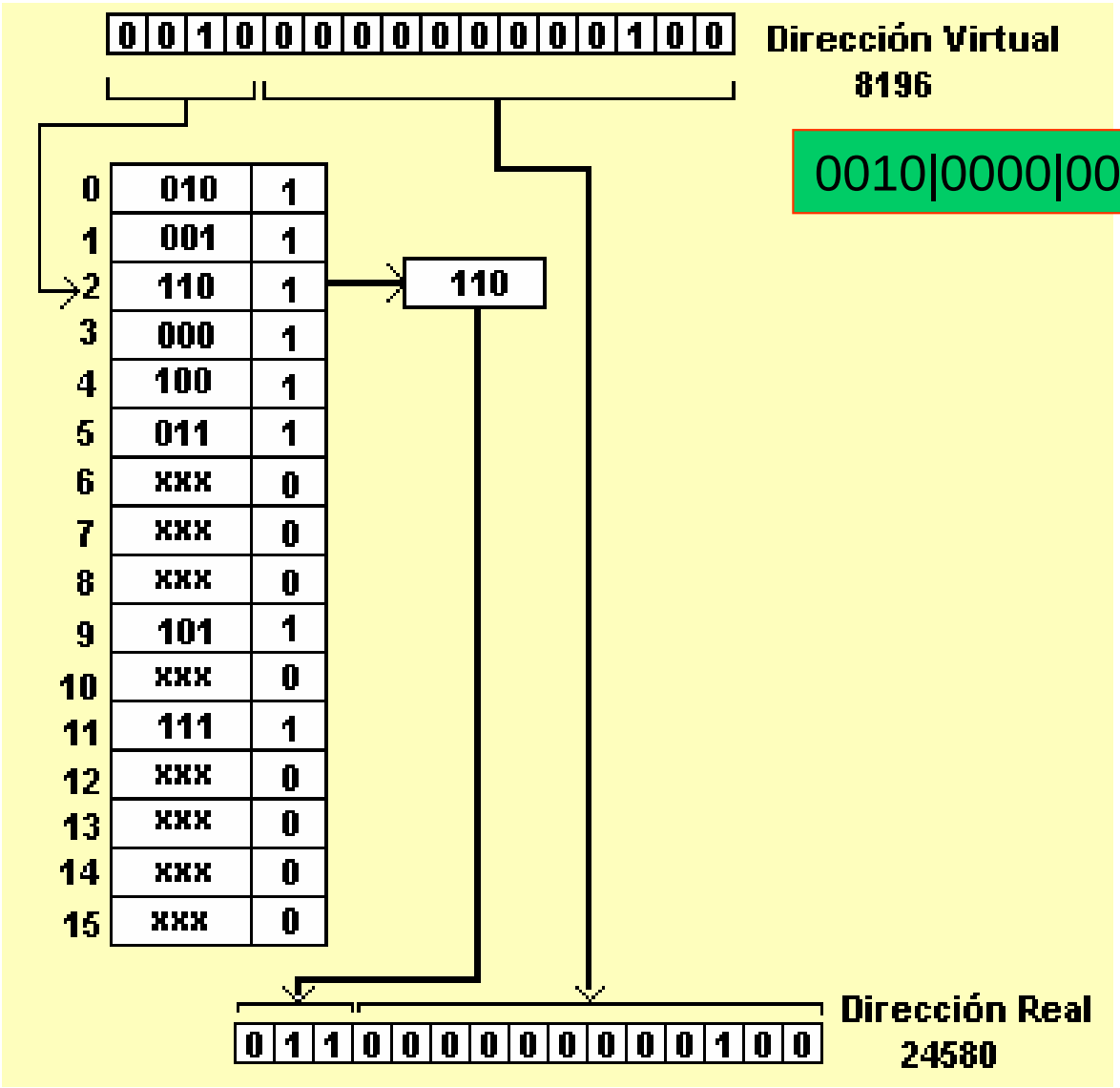
PAGINACION Y SEGMENTACION VIRTUAL

<table>
<tr><td>Real</td><td colspan="2">Real</td><td colspan="2">Real</td><td colspan="2">Virtual</td></tr>
<tr><td>Mono Usuario</td><td colspan="2">Multiprogramación</td><td colspan="2">Multiprogramación</td><td colspan="2">Multiprogramación</td></tr>
<tr><td></td><td colspan="2">Particionamiento</td><td>Paginación Simple</td><td>Segmentación Simple</td><td>Paginación Virtual</td><td>Segmentación Virtual</td></tr>
<tr><td></td><td>Fija</td><td>Dinámica</td><td colspan="2">Combinación</td><td colspan="2">Combinación</td></tr>
<tr><td></td><td colspan="6">Reubicación, Protección</td></tr>
</table>

Memoria Virtual

- La memoria virtual es una técnica para proporcionar la **ilusión** de un espacio de memoria mayor que la memoria física, sin tener en cuenta el tamaño de la memoria física.
- Está soportada por el mecanismo de **traducción de memoria**, junto con un almacenamiento rápido en disco duro (**swap**).
- El **espacio de direcciones virtual**, está mapeado de tal forma que una pequeña parte de él, está en memoria real y el resto almacenado en el disco.

Traducción de Memoria



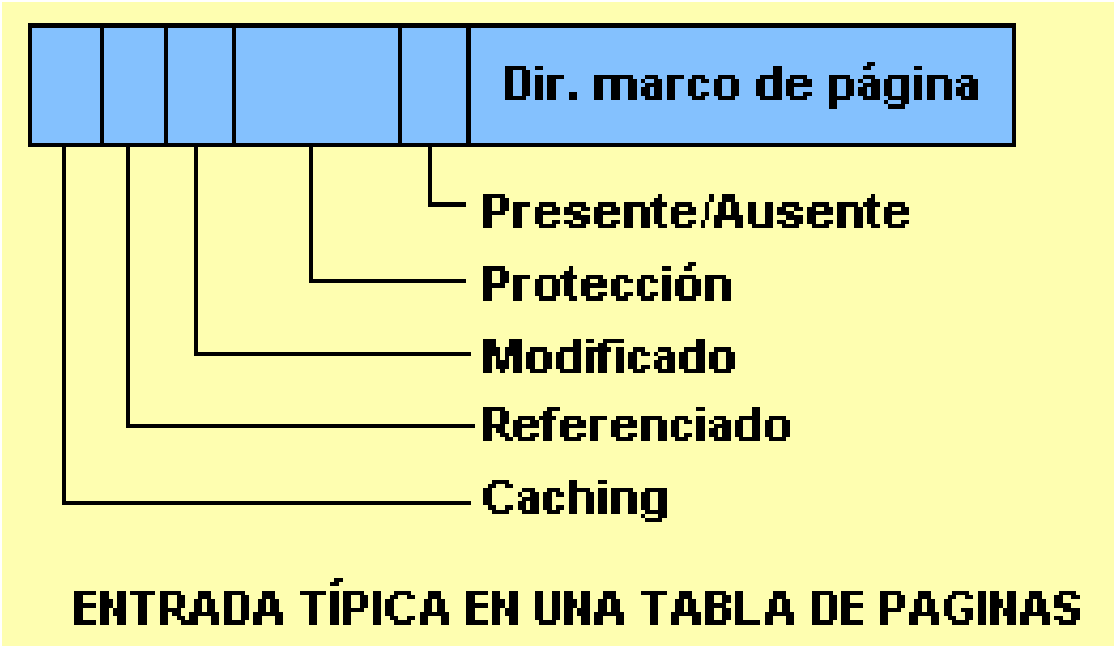
5. Memoria Virtual Paginada

- Igual que la paginación simple.
- No es necesario cargar todas las páginas.
- Las páginas no residentes se cargan por demanda.

- Ventajas. No fragmentación externa. Alto grado de multiprogramación. Gran espacio virtual para el proceso.
- Desventaja. Sobrecarga por gestión compleja de memoria.

5. Memoria Virtual Paginada

- Cada proceso tiene su propia tabla de paginas.

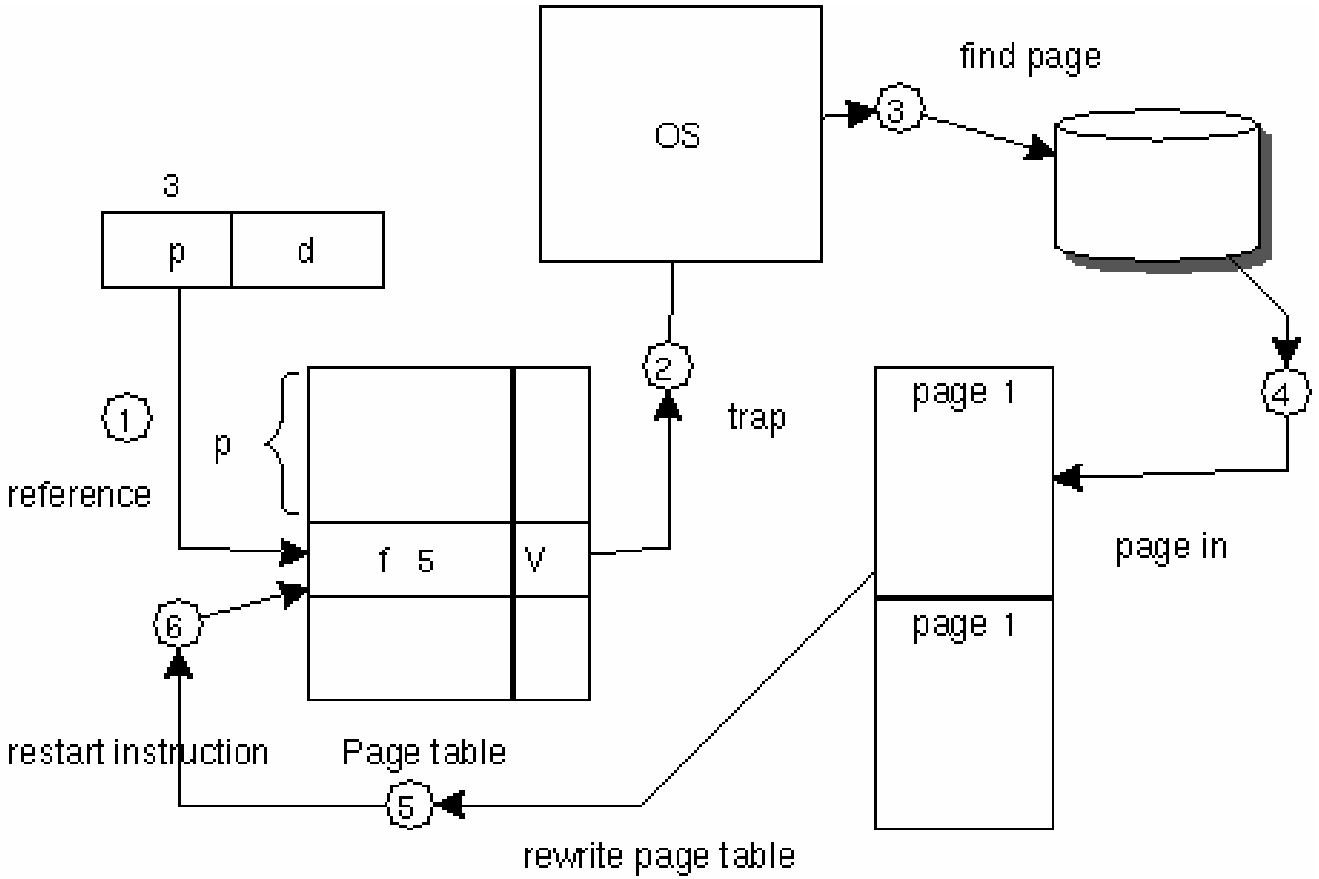


ENTRADA TÍPICA EN UNA TABLA DE PAGINAS

- Si la pagina no se modifica, al realizarse el swap a disco no se necesitara copiar desde la memoria principal a la memoria secundaria.

5. Fallo de Página (page fault)

- Ocurre cuando se referencia a una dirección virtual y ella no reside en la memoria real, se presenta una interrupción ***fallo de página.***



5. Tamaño de Página

- Páginas pequeñas
  - Menos fragmentación interna.
  - Más páginas para el proceso.
  - Muchas páginas por proceso.
  - La tabla de paginas crecerá en tamaño.
  - Se necesita mas MV para carga la tabla.
  - El fallo de página se reduce.
- Páginas grandes
  - Mas fragmentación interna.
  - C/página contiene mas porciones del proceso.
  - Se ocupa memoria innecesariamente.
  - El fallo de página se incrementa.

6. Memoria Virtual Segmentada

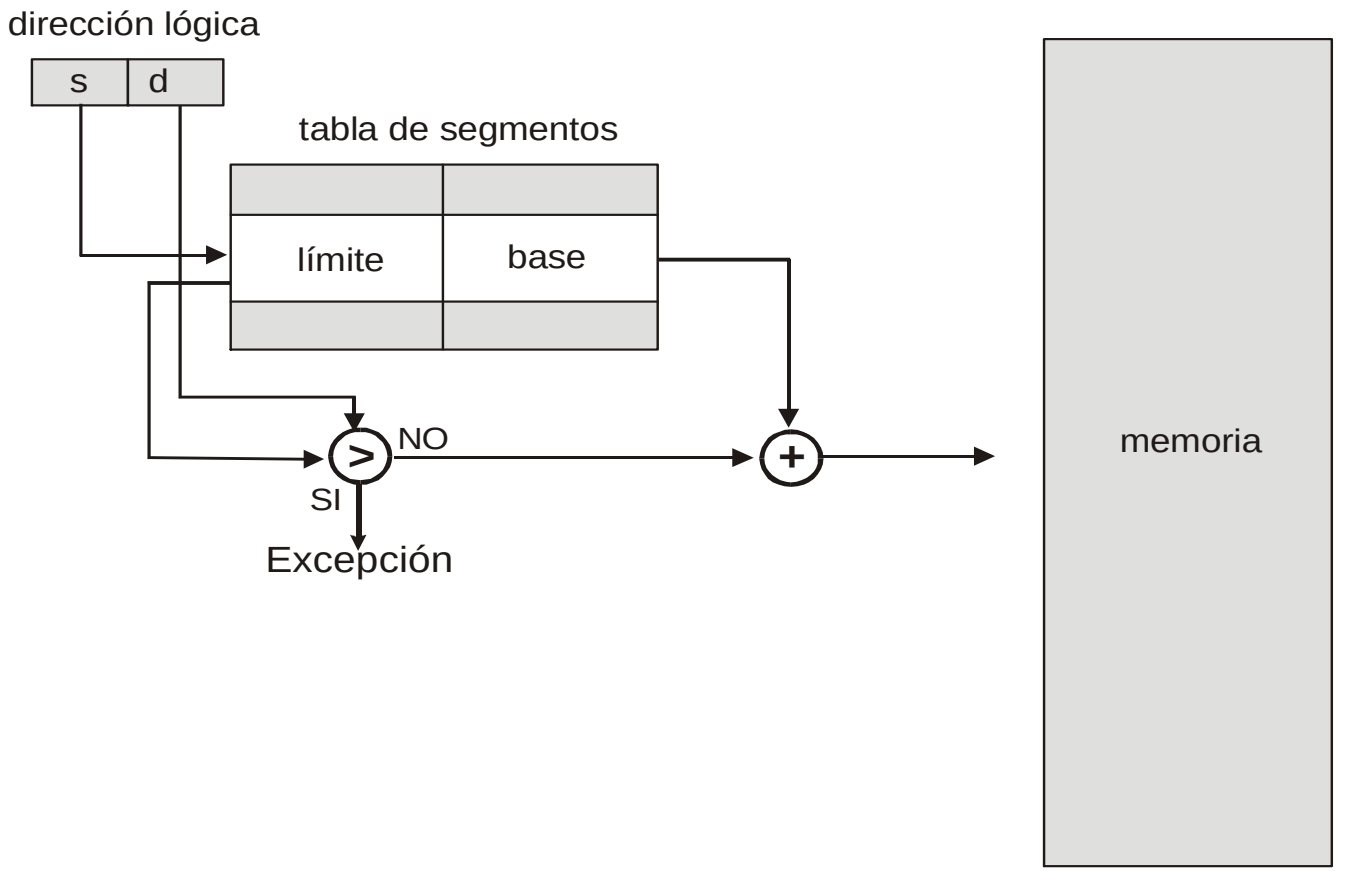
- Igual que la segmentación simple.
- No es necesario cargar todos los segmentos.
- Las segmentos se cargan por demanda.
- Segmentos de tamaño dinámico, según la demanda.
- Se puede alterar los programas y recompilarlos independientemente.

6. Memoria Virtual Segmentada

- Permite compartir datos entre procesos, mediante el uso segmentos compartibles.
- Permite la protección de datos, el administrador otorgar permisos a este segmento.
- Ventajas. No hay fragmentación interna. Alto grado de multiprogramación. Gran espacio virtual para el proceso. Soporte de protección y compartición (sharing).
- Desventajas. Sobrecarga por gestión compleja de memoria.

6. Tabla de Segmentos

- El SO debe mantener una lista de huecos libres.
- Un bit expresa si el segmento se encuentra ya en memoria.
- Un bit expresa si el segmento ha sido modificado.



CONCLUSIONES

1. El SAM particionado a diferencia de la paginación o segmentación simple, permite que sólo un proceso se cargue en memoria principal.
2. Cuando se trabaja con bloques de tamaño fijo se genera la fragmentación interna. Si los bloques son de tamaño variable, se genera la fragmentación externa.
3. El SAM de particiones fijas se parece al SAM de paginación simple, diferenciándose en que los primeros requieren que las particiones estén contiguas

Consistencia, Replicación y Memoria Compartida

6

Sistemas Distribuidos

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

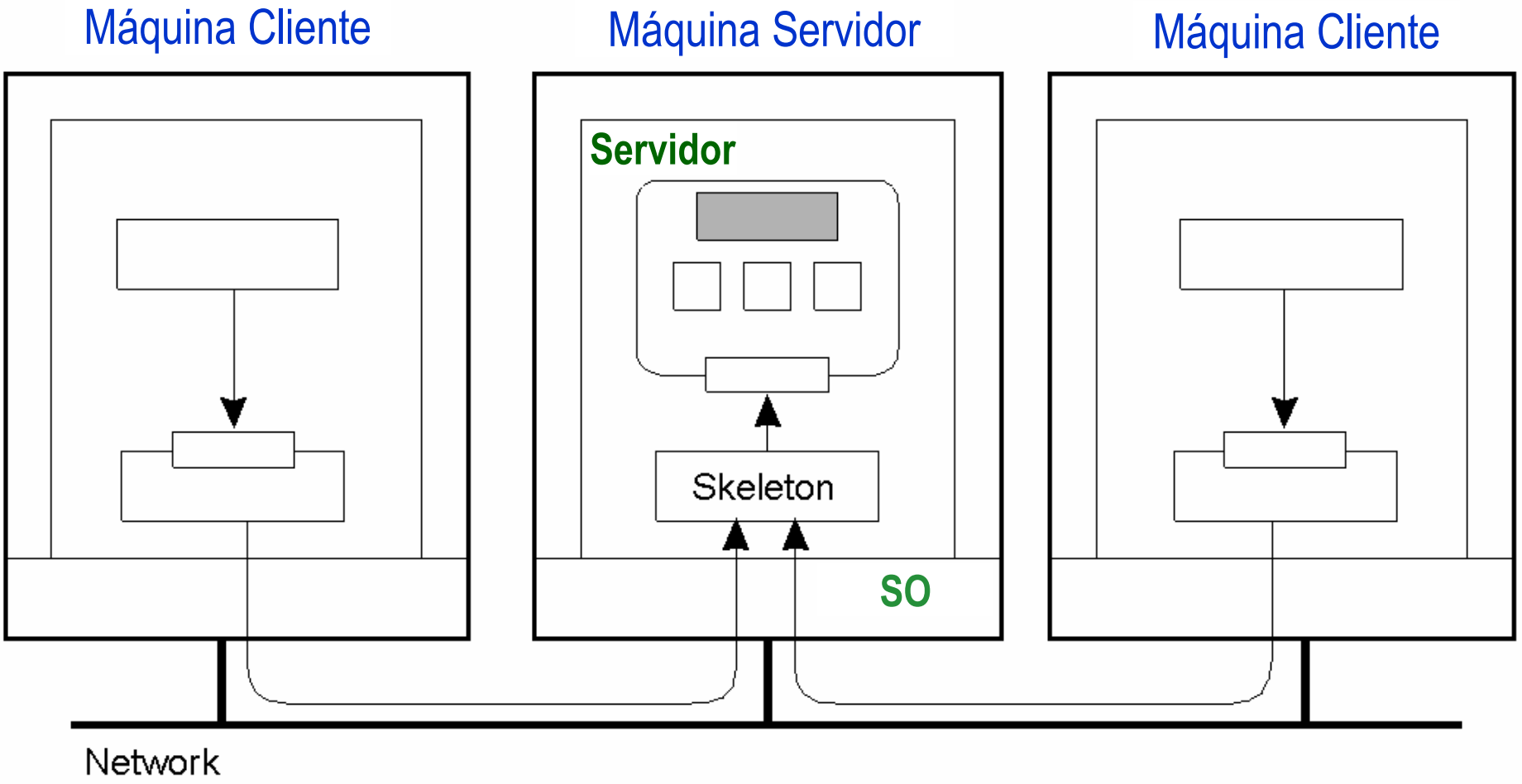
Razones para la Replicación

Hay dos razones principales para la replicación de datos:

- Confiabilidad
  - Continuidad de trabajo ante caída de una réplica.
  - Mayor cantidad de copias mejor protección contra la corrupción de datos (copias para “desempatar”).
- Rendimiento (Performance)
  - El SD escala en número (e.j. réplica de web servers).
  - Escala en área geográfica (disminuye el tiempo de acceso al dato, acerca dato con procesamiento).
  - Consulta simultánea de los mismos datos.

Precio a pagar por la replicación de datos:
Problemas de Consistencia

Replicación de Objetos



- Organización de un objeto remoto distribuido compartido por dos clientes diferentes.

Replicación de Objetos (cont)

Cuando se replican objetos remotos en varias máquinas es necesario resolver el problema de como proteger al objeto contra el acceso simultáneo de múltiples clientes.

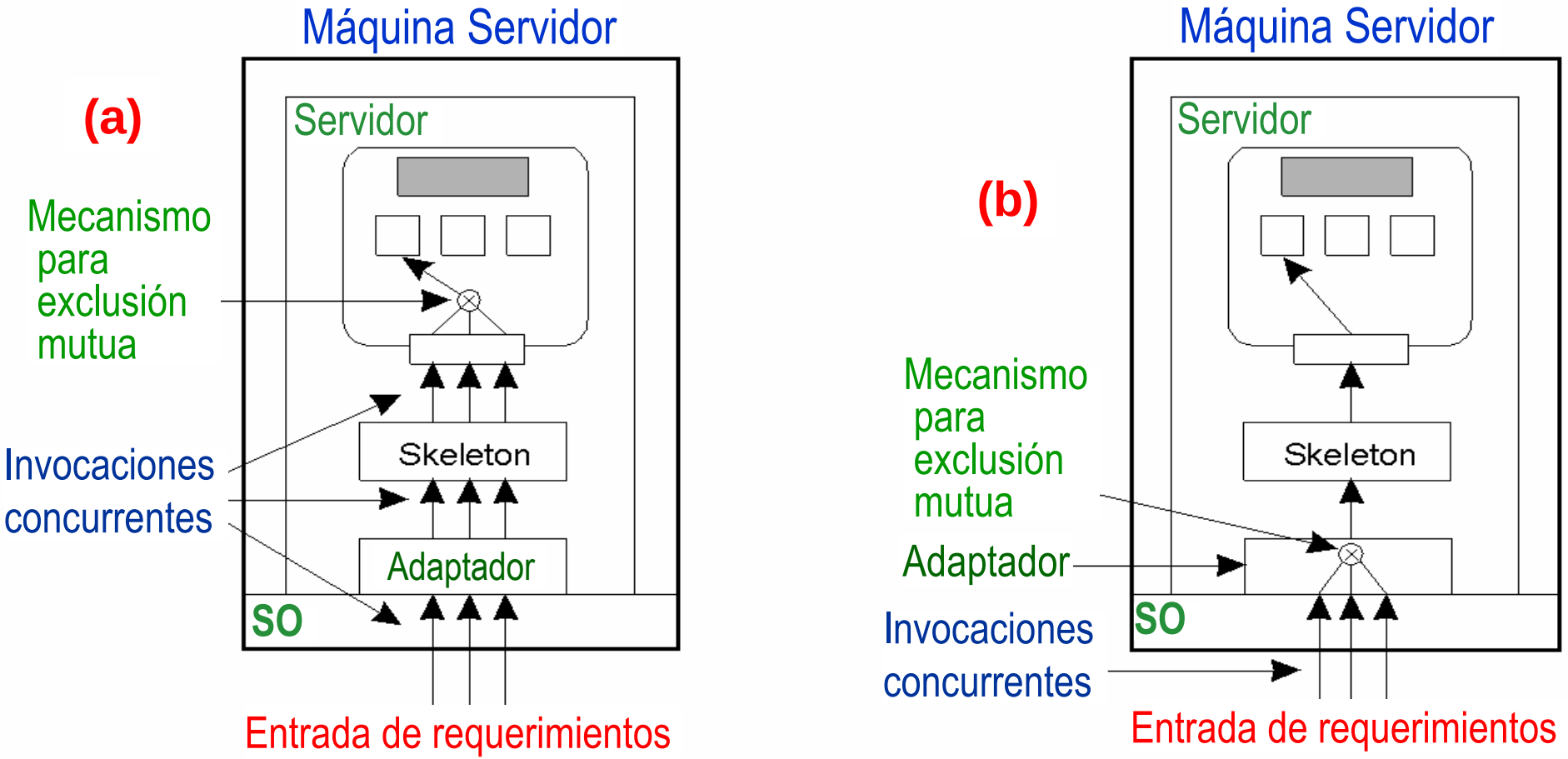
Básicamente hay dos soluciones [Briot et al,1998].

Una solución es que el mismo objeto maneje las invocaciones concurrentes (a). Ej. Declaración de métodos Java como *synchronized*.

Otra solución es que el objeto esté totalmente desprotegido y del control de concurrencia se ocupe el servidor en el cual el objeto reside (b).

Dado que el objeto está replicado debe extremarse el cuidado de modo que cada réplica esté permanentemente actualizada.

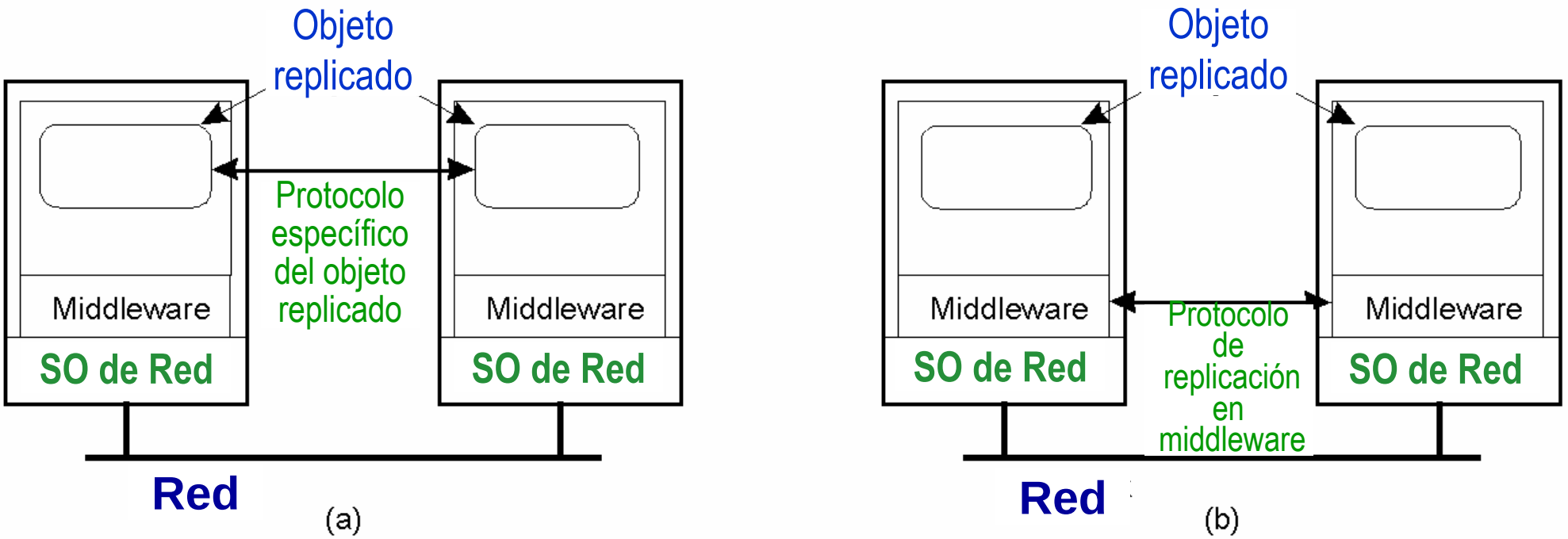
Replicación de Objetos (cont)



a) Un objeto remoto capaz de manejar invocaciones concurrentes por sus propios medios.

b) Un objeto remoto para el cual se requiere un adaptador para manejar las invocaciones concurrentes.

Replicación de Objetos (cont)



a) Un sistema distribuido para manejo propio de la replicación por los objetos distribuidos.

b) Un sistema distribuido responsable del manejo de las réplicas.

Replicación como Técnica de Escalabilidad

En general **lograr escalabilidad va en detrimento del rendimiento**.

Como técnicas para facilitar la escalabilidad se utiliza la **replicación** y el **caching**.

Ubicar copias de datos u objetos cercanos a los procesos que los usan mejora el rendimiento por la reducción del tiempo de acceso y resuelve el problema de escalabilidad.

Problemas:

La actualización de las réplicas consume más ancho de banda de la red.

Mantener múltiples copias consistentes resulta a su vez un serio problema de escalabilidad y más en un contexto de consistencia estricta (*strict consistency*).

La idea es que la actualización se realice con una única operación atómica. Se necesita sincronizar todas las réplicas.

Dilema: Por un lado la replicación tiende a resolver el problema de la escalabilidad (aumenta el rendimiento); por otro mantener consistentes las copias requiere sincronización global. La cura puede ser peor que la enfermedad.

Modelos de Consistencia

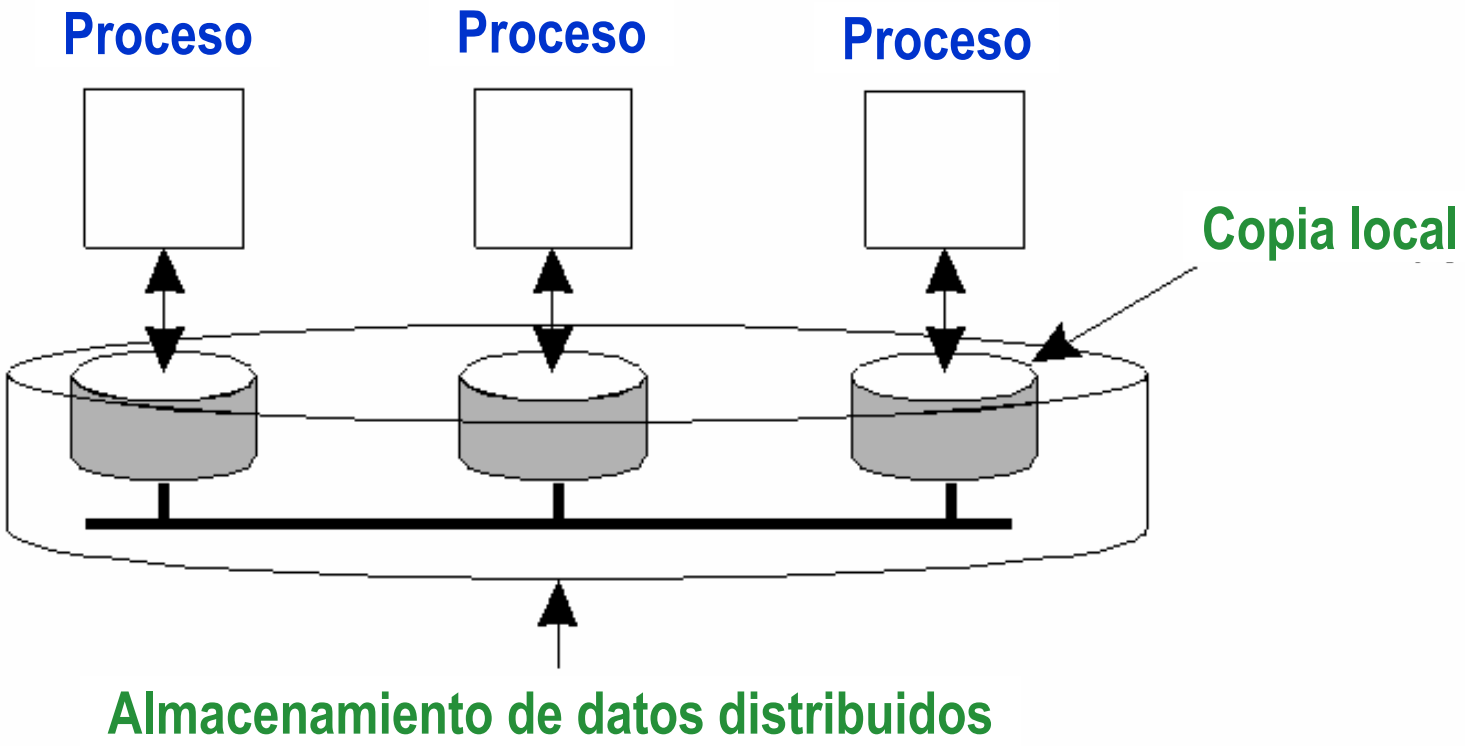
Un modelo de consistencia es esencialmente un contrato entre procesos y el almacenamiento de datos. Es decir: si los procesos *acuerdan* obedecer ciertas reglas, el almacenamiento *promete* trabajar correctamente.

Normalmente un proceso que realiza una operación de lectura espera que esa operación devuelva un valor que refleje el resultado de la última operación de escritura sobre el dato.

Los modelos de consistencia se presentan divididos en dos conjuntos:

- **Modelos de consistencia centrados en los datos.**
- **Modelos de consistencia centrados en el cliente.**

Modelos de Consistencia Centrados en Datos

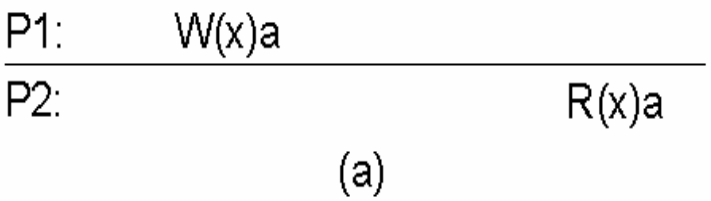


- Organización general de un almacenamiento lógico de datos, físicamente distribuidos y replicados a través de múltiples procesos.

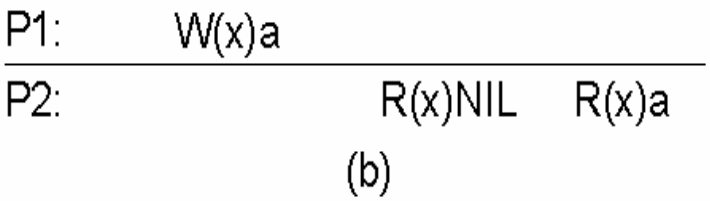
Consistencia Estricta

El modelo de consistencia más restrictivo es llamado **consistencia estricta** y es definido por la siguiente condición:

> Cualquier lectura sobre un item de dato *x* retorna un valor correspondiente con la más reciente escritura sobre *x*



(a)



(b)

a) Un almacenamiento estrictamente consistente.

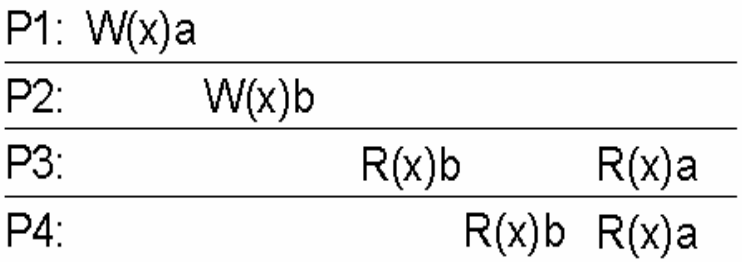
b) Un almacenamiento que no es estrictamente consistente.

Nota: La definición supone un ***tiempo global absoluto***.

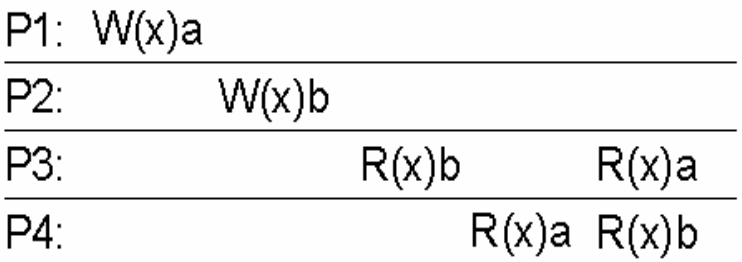
Linealizabilidad y Consistencia Secuencial

La consistencia secuencial es una forma ligeramente más débil de la consistencia estricta. Satisface la siguiente condición:

> El resultado de una ejecución es el mismo si las operaciones (lectura y escritura) de todos los procesos sobre el dato fueron ejecutadas en algún orden secuencial y las operaciones de cada proceso individual aparecen en esta secuencia en el orden especificado por su programa.



(a)



(b)

a) Un dato almacenado secuencialmente consistente.

b) Un dato almacenado que no es secuencialmente consistente.

Linealizabilidad y Consistencia Secuencial (cont)

El modelo de **linealizabilidad** es más débil que la consistencia estricta pero más fuerte que la consistencia secuencial. En este modelo las operaciones reciben una estampilla de tiempo generada por algún reloj global.

Sea $ts_{OP}(x)$ la estampilla de tiempo asignada a la operación *OP* realizada sobre el dato *x*, donde *OP* es lectura (*R*) o escritura (*W*).

Un dato se dice ser linealizable (Herlihy and Wing,1991) cuando cada operación es estampillada y se cumple la condición:

Idem que consistencia secuencial pero con el agregado: si $ts_{OP1}(x) < ts_{OP2}(y)$, la operación *OP*1(*x*) precede *OP*2(*y*) en esa secuencia.

Linealizabilidad y Consistencia Secuencial (cont)

Proceso P1	Proceso P2	Proceso P3
x = 1;	y = 1;	z = 1;
print (y, z);	print (x, z);	print (x, y);

- Tres procesos ejecutándose concurrentemente.

Supóngase que las variables tienen valor inicial 0 y que están almacenadas (posiblemente distribuido) en un almacenamiento compartido y secuencialmente consistente.

Linealizabilidad y Consistencia Secuencial (cont)

- Cuatro secuencias válidas de ejecución para procesos de la slide anterior. El eje vertical es el tiempo.

x = 1;	x = 1;	y = 1;	y = 1;
print (y, z);	y = 1;	z = 1;	x = 1;
y = 1;	print (x, z);	print (x, y);	z = 1;
print (x, z);	print (y, z);	print (x, z);	print (x, z);
z = 1;	z = 1;	x = 1;	print (y, z);
print (x, y);	print (x, y);	print (y, z);	print (x, y);
Prints: 001011	Prints: 101011	Prints: 010111	Prints: 111111
Signature: 001011	Signature: 101011	Signature: 110101	Signature: 111111
(a)	(b)	(c)	(d)

Consistencia Causal

El modelo de consistencia causal (Hutto and Ahamad, 1990) es un debilitamiento de la consistencia secuencial. Se hace una diferenciación entre eventos que están potencialmente relacionados en forma causal y aquellos que no. Las operaciones que no están causalmente relacionadas se dicen concurrentes.

La condición a cumplir para que un datos sean causalmente consistentes es:

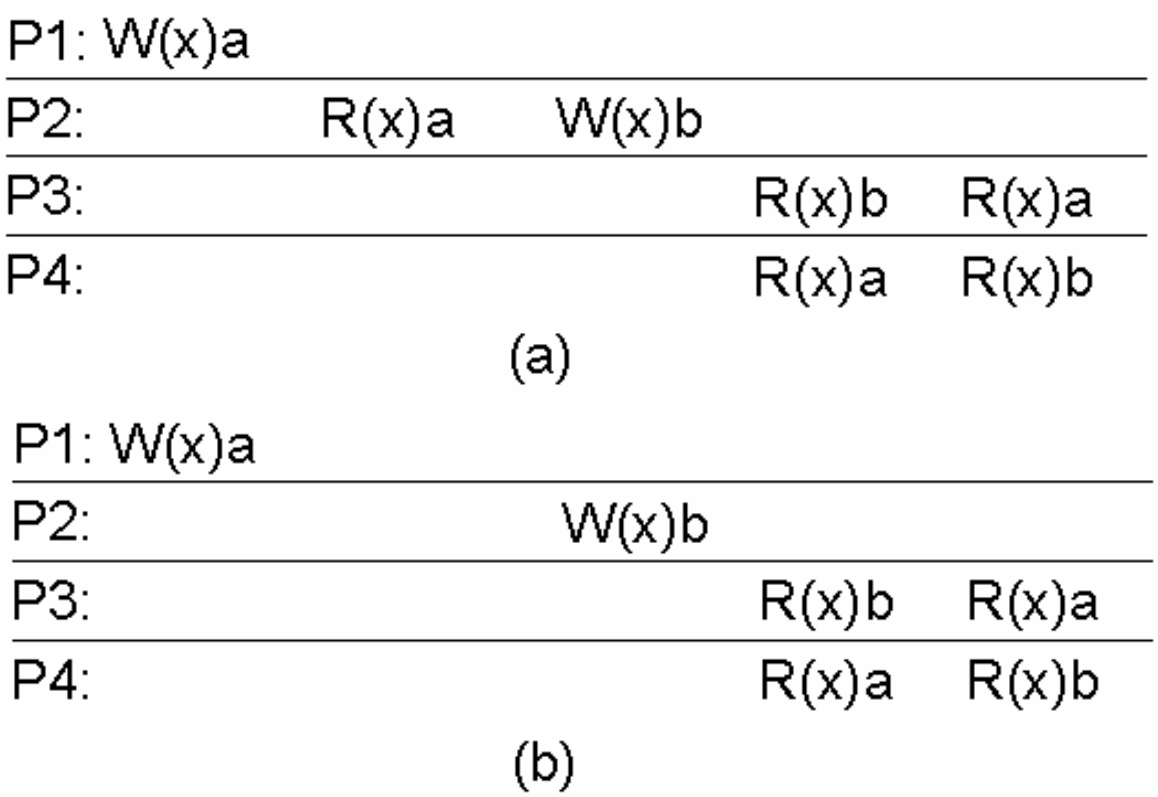
> Escrituras que están potencialmente relacionadas en forma causal deben ser vistas por todos los procesos en el mismo orden. Escrituras concurrentes pueden ser vistas en un orden diferente sobre diferentes máquinas.

Consistencia Causal (cont)

P1:	W(x)a			W(x)c	
P2:		R(x)a	W(x)b		
P3:		R(x)a		R(x)c	R(x)b
P4:		R(x)a		R(x)b	R(x)c

- Esta secuencia es permitida con un almacenamiento causalmente consistente, pero no con un almacenamiento secuencialmente consistente o con un almacenamiento consistente en forma estricta.

Consistencia Causal (cont)



a) Una violación de una consistencia causal.

b) Una correcta secuencia de eventos en una consistencia causal.

Consistencia First Input First Output

- Condición Necesaria :

> Escrituras realizadas por un proceso único son vistas por los otros procesos en el orden en que son hechas, pero escrituras desde diferentes procesos pueden ser vistas en diferente orden por diferentes procesos.

Consistencia First Input First Output (cont)

P1: W(x)a								
P2:		R(x)a		W(x)b		W(x)c		
P3:						R(x)b	R(x)a	R(x)c
P4:						R(x)a	R(x)b	R(x)c

- Una secuencia válida de eventos de una consistencia First Input First Output.

Consistencia First Input First Output (cont)

x = 1;	x = 1;	y = 1;
print (y, z);	y = 1;	print (x, z);
y = 1;	**print(x, z);**	z = 1;
print(x, z);	print (y, z);	**print (x, y);**
z = 1;	z = 1;	x = 1;
print (x, y);	print (x, y);	print (y, z);
Prints: 00	Prints: 10	Prints: 01
(a)	(b)	(c)

- Las sentencias de ejecución como son vistas por los tres procesos de las slides anteriores. Las sentencias en negrita son las que generan la salida.

Consistencia First Input First Output (cont)

Proceso P1

```
x = 1;
if (y == 0) kill (P2);
```

Proceso P2

```
y = 1;
if (x == 0) kill (P1);
```

- Dos procesos concurrentes.

Consistencia Débil

Propiedades

- Los accesos a variables de sincronización asociadas con los datos almacenados son secuencialmente consistentes.
- No se permite operación sobre una variable de sincronización hasta que todas las escrituras previas se hayan completado.
- No se permiten operaciones de escritura o lectura sobre items de datos hasta que no se hayan completado operaciones previas sobre variables de sincronización.

Consistencia Débil (cont)

```
int a, b, c, d, e, x, y;          /* variables */
int *p, *q;                       /* pointers */
int f( int *p, int *q);           /* function prototype */

a = x * x;                        /* a stored in register */
b = y * y;                        /* b as well */
c = a*a*a + b*b + a * b;          /* used later */
d = a * a * c;                    /* used later */
p = &a;                           /* p gets address of a */
q = &b                            /* q gets address of b */
e = f(p, q)                       /* function call */
```

- Un fragmento de programa en el cual algunas variables son mantenidas en registros.

Consistencia Débil (cont)

a)

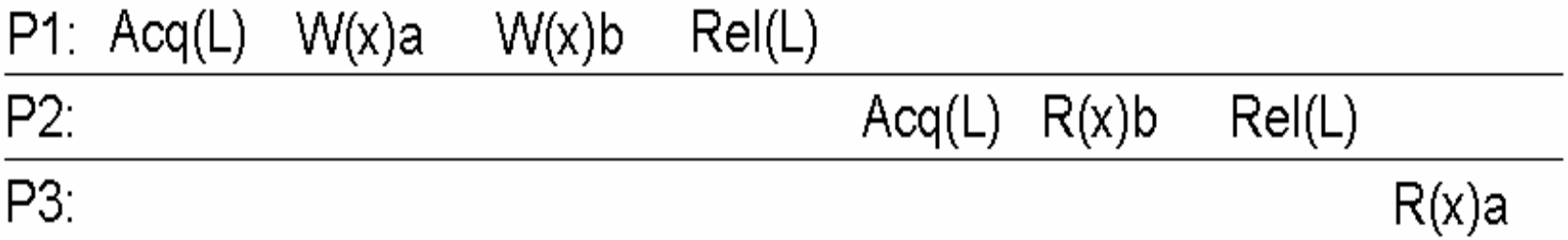
P1: W(x)a	W(x)b	S			
P2:			R(x)a	R(x)b	S
P3:			R(x)b	R(x)a	S

b)

P1: W(x)a	W(x)b	S		
P2:			S	R(x)a

a) Una secuencia válida de eventos para consistencia débil.
b) Una secuencia inválida para consistencia débil.

Consistencia Relajada



- Una secuencia válida de eventos para una consistencia relajada.

Consistencia Relajada (cont)

Reglas:

- Antes de una operación de lectura o escritura sobre variables compartidas sea realizada, todos los “acquires” previos hechos por el proceso deben ser completados exitosamente.
- Antes de permitir realizar una liberación, todas las lecturas o escrituras previas por el proceso deben completarse.
- Los accesos a variables de sincronización son FIFO consistentes (no se requiere consistencia secuencial).

Consistencia *Entry*

Condiciones:

- No se permite realizar un acceso para adquirir una variable de sincronización respecto a un proceso hasta que todas las actualizaciones sobre los datos compartidos han sido realizadas con respecto a ese proceso.
- Antes de que sea permitido realizar un modo exclusivo de acceso a variables de sincronización por un proceso, otros procesos no pueden retener la variable de sincronización, ni aún en el modo no exclusivo.
- Despúes de un modo exclusivo de acceso a la variable de sincronización haya sido realizado, cualquier otro modo de acceso no exclusivo de un proceso posterior, a la variable de sincronización no se puede realizar hasta que lo haya realizado sobre las propias variables.

Consistencia *Entry*

Condiciones:

- Cuando un proceso hace un *adquire* , éste no se completa (no retorna el control a la próxima sentencia) hasta que todos los datos compartidos *guardados* han sido actualizados. Es decir, en un *adquire*, todos los cambios sobre los datos *guardados* deben ser visibles.
- Antes de actualizar un item de dato compartido, un proceso debe entrar en la sección crítica en modo exclusivo para asegurarse que, al mismo tiempo, no hay otros procesos tratando de actualizar el item de dato compartido.
- Si un proceso quiere entrar en la sección crítica en modo no exclusivo, primero debe verificar con el dueño de la variable de sincronización *guardando* la sección crítica la búsqueda de las copias más recientes de los datos compartidos *guardados*.

Consistencia *Entry*

P1:	Acq(Lx)	W(x)a	Acq(Ly)	W(y)b	Rel(Lx)	Rel(Ly)							
P2:							Acq(Lx)	R(x)a				R(y)NIL	
P3:									Acq(Ly)	R(y)b			

- Una secuencia válida para consistencia *entry*.

Sumario de Modelos de Consistencia

Consistencia	Descripción
Estricta	Ordenamiento en tiempo absoluto de todos los accesos compartidos.
Linealizabilidad	Todos los procesos deben ver todos los accesos compartidos en el mismo orden. Los accesos son ordenados de acuerdo a una estampilla de tiempo global.
Secuencial	Todos los procesos ven todos los accesos compartidos en el mismo orden. Los accesos no están ordenados en el tiempo.
Causal	Todos los procesos ven los accesos compartidos causalmente relacionados en el mismo orden.
FIFO	Todos los procesos ven las escrituras de cada uno en el orden en que fueron hechas. Escrituras de diferentes procesos pueden no ser vistas en el mismo orden.

(a) Modelos de consistencia que no usan operaciones de sincronización

Consistencia	Descripción
Débil	Datos compartidos pueden ser contados como consistentes luego de que se haya hecho la sincronización.
Relajada	Datos compartidos son hechos consistentes cuando la región crítica es abandonada.
Entry	Datos compartidos pertenecientes a una región crítica son hechos consistentes cuando se entra a la región crítica.

(b) Modelos con operaciones de sincronización

Modelos de Consistencia Centrados en el Cliente

Este tipo de modelos trata una clase especial de almacenamiento de datos distribuidos.

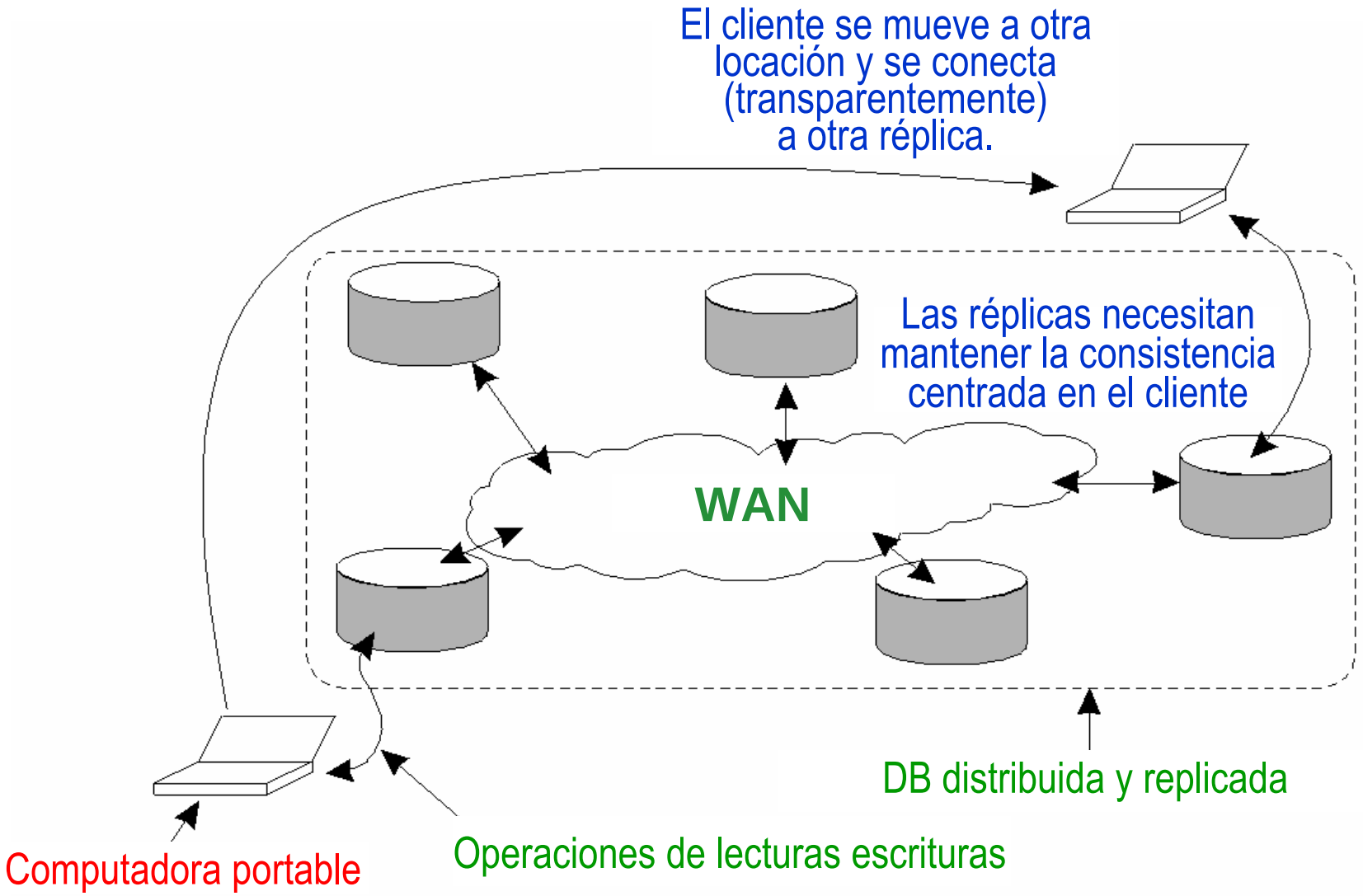
Los almacenamiento de datos referidos están caracterizados por una falta de actualizaciones simultáneas, o cuando dichas actualizaciones ocurren, pueden ser fácilmente resueltas.

La mayoría de las operaciones son de lectura.

La introducción de modelos de consistencia centrados en el cliente permiten esconder muchas inconsistencias de manera relativamente fácil.

En esencia la consistencia centrada en el cliente provee garantías *para un único cliente* concerniente a la consistencia de accesos a los datos de ese cliente. No se dan garantías para accesos concurrentes por diferentes clientes.

Consistencia Eventual



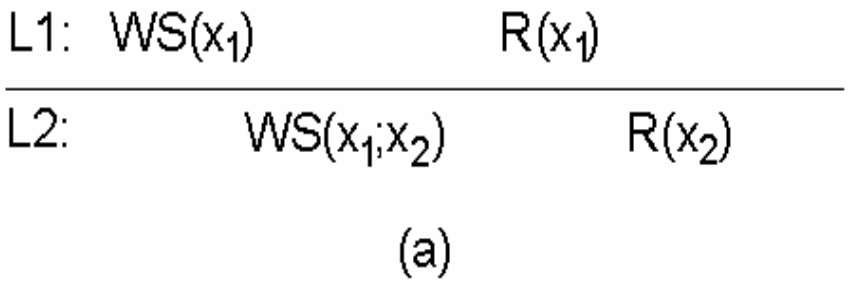
Lecturas Monotónicas (monótonas)

Se dice que un dato ofrece *consistencia de lecturas monotónicas (monótonas)* si se cumple la siguiente condición:

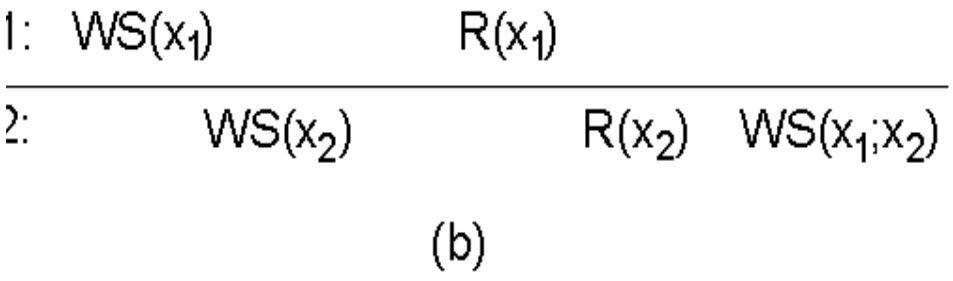
> Si un proceso lee el valor de un item de dato x, cualquier operación de lectura sucesiva sobre x por el proceso siempre retornará el mismo valor o un valor más reciente.

Lecturas Monotónicas

Operaciones de lectura realizadas por un proceso único *P* sobre dos copias locales del mismo dato almacenado.



(a)



(b)

a) Una lectura monotónica consistente sobre un dato almacenado.

b) Un dato almacenado que no provee lecturas monotónicas.

Escrituras Monotónicas

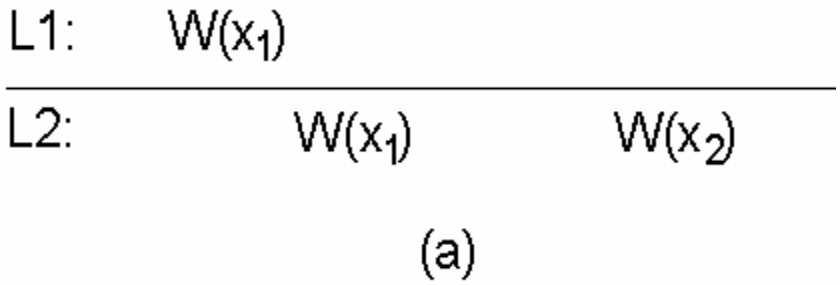
En muchas situaciones, es importante que las escrituras sean propagadas en el orden correcto a todas las copias del almacenamiento de datos. Esta propiedad es expresada en la consistencia de escrituras monotónicas.

En un almacenamiento con modelo de *consistencia escrituras monotónicas* se debe cumplir la siguiente condición:

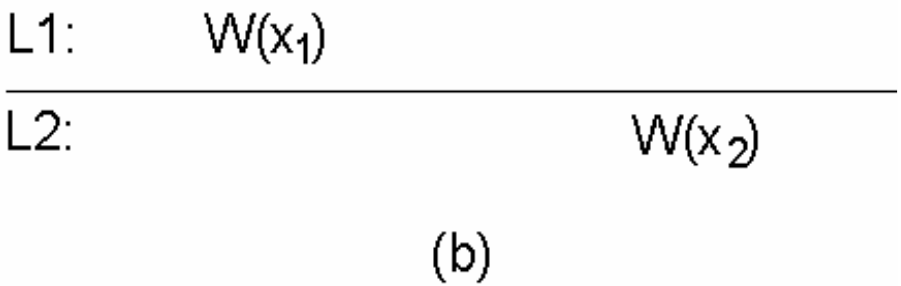
> Una operación de escritura por un proceso sobre un item de dato *x* es completada antes de cualquier otra operación de escritura sobre *x* por el mismo proceso

Escrituras Monotónicas

Operaciones de escritura realizadas por un proceso simple *P* en dos copias locales diferentes del mismo dato almacenado.



(a)



(b)

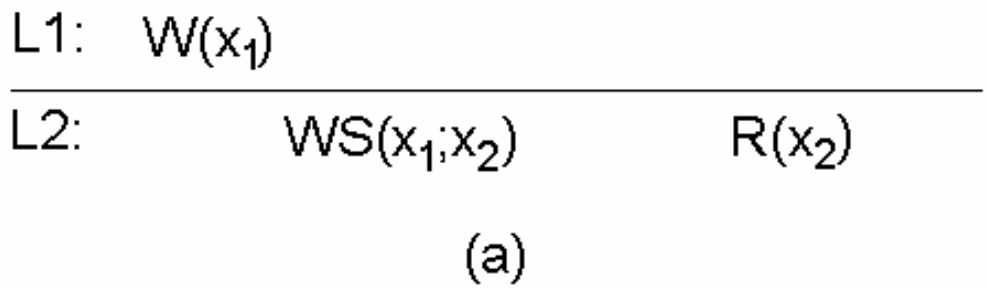
a) Una escritura monotónica consistente.
b) Un dato almacenado que no provee una escritura monotónica consistente.

Lea sus Escrituras

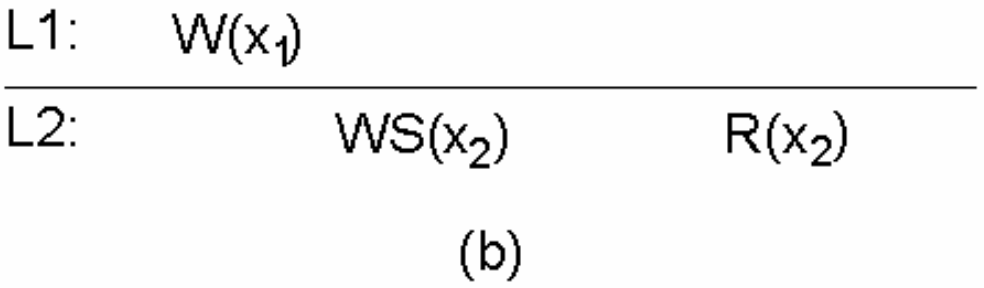
Un almacenamiento de datos se dice que provee *consistencia lea sus escrituras* si se cumple la siguiente condición:

> El efecto de una operación de escritura por un proceso sobre un item de dato x será siempre visto por las sucesivas operaciones de lectura sobre x por el mismo proceso

Lea sus Escrituras



(a)



(b)

a) Un dato almacenado que provee consistencia “lea sus escrituras”.

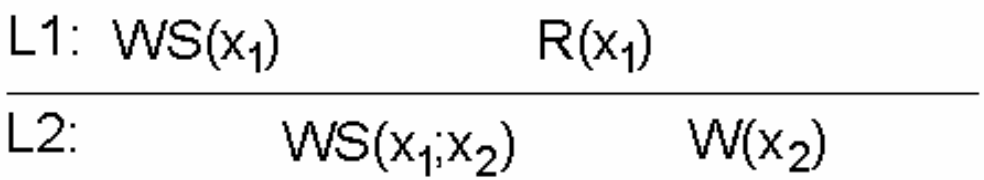
b) Un dato almacenado que no.

Escrituras siguen Lecturas

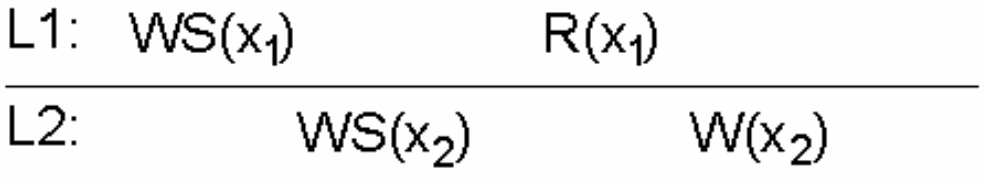
Un almacenamiento de datos se dice que provee *consistencia escrituras siguen lecturas* si se cumple la siguiente condición:

> Una operación de escritura por un proceso sobre un item de dato x que sigue a una operación de lectura previa sobre x por el mismo proceso, es garantido que toma lugar sobre el mismo o el más reciente valor de x leído.

Escrituras siguen Lecturas



(a)

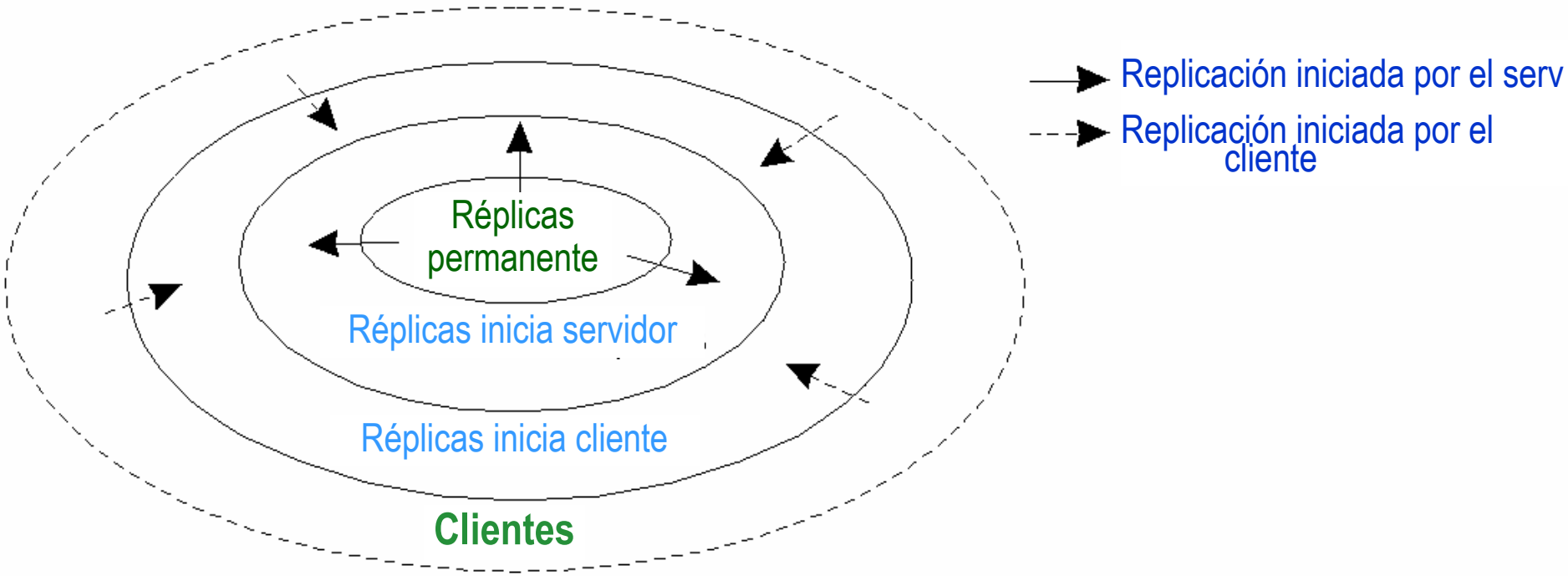


(b)

a) Un dato almacenado consistente “escrituras siguen lecturas”.

b) Un dato almacenado que no provee consistencia “escrituras siguen lecturas”.

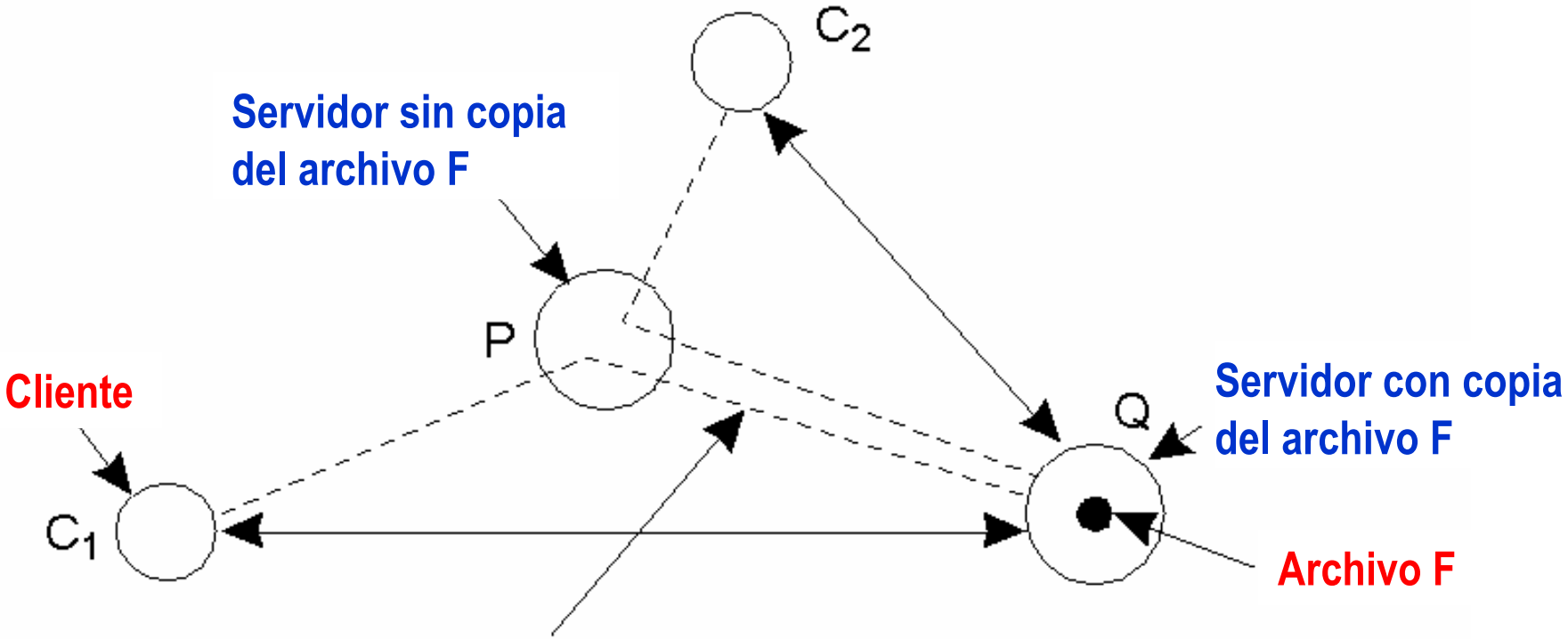
Ubicación de Réplicas



La organización lógica de diferentes clases de copias de datos almacenados en tres anillos concéntricos.

Réplicas Iniciadas por el Servidor

Contando requerimientos de acceso de diferentes clientes.



Protocolos Pull versus Push

Asunto	Basado en Push	Basado en Pull
Estado del servidor	Lista de réplicas de clientes y caches	No
Mensajes enviados	Actualización (y posiblemente búsqueda actualización más tarde)	Pull y actualización
Tiempo de respuesta al cliente	Inmediato (o tiempo búsqueda-actualización)	Tiempo búsqueda-actualización

- Una comparación entre protocolos basados en push y basados en pull en el caso de múltiples clientes y sistema servidor único. Por simplicidad se considera un sistema cliente-servidor consistente de un único servidor no distribuido y un determinado número de procesos cliente con sus propios cachés.

Protocolos de Consistencia

Un *protocolo de consistencia* describe una implementación de un modelo específico de consistencia.

Lejos, los modelos de consistencia en los cuales las operaciones están globalmente seriadas son los modelos más importantes y los más ampliamente aplicados.

Estos modelo incluyen consistencia secuencial, consistencia débil con variables de sincronización y transacciones atómicas.

Los protocolos pueden ser:

- Protocolos basados en el primario
- Protocolos de escritura replicada

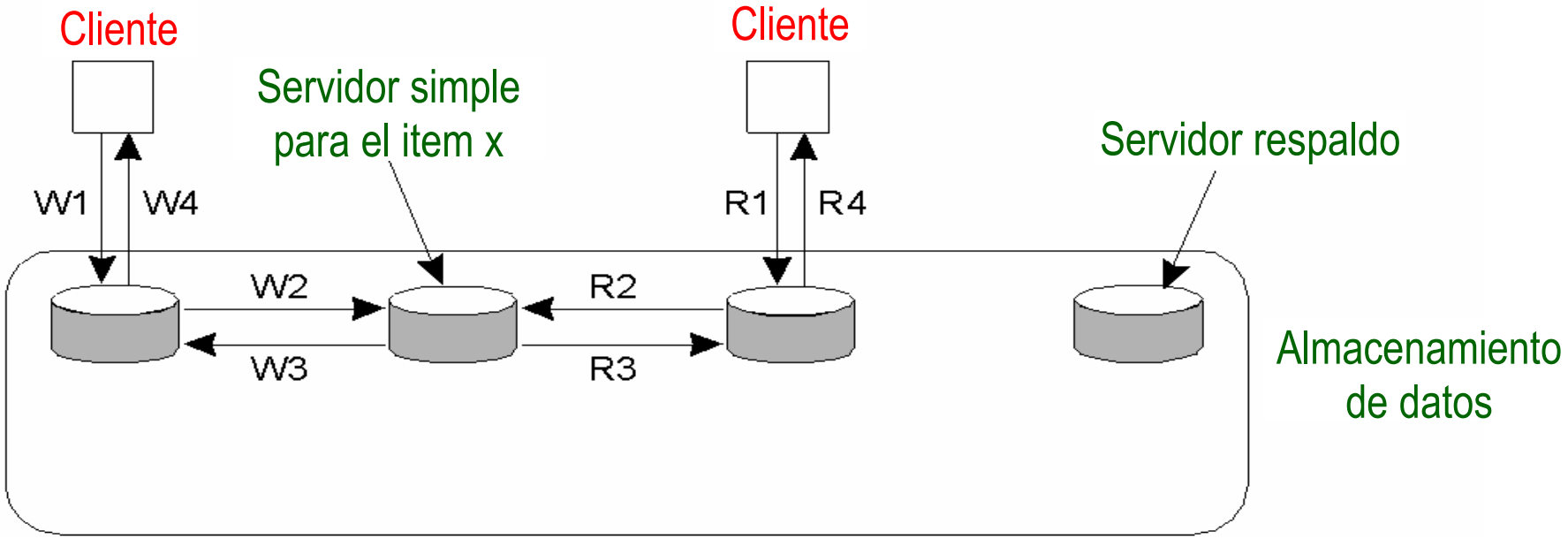
Protocolos Basados en el Primario

Cada item de dato x tiene en el almacenamiento de datos un primario asociado, el cual es responsable de coordinar las operaciones de escritura sobre x.

Se debe hacer una distinción si el primario está fijo en un servidor remoto o si las operaciones de escritura pueden ser llevados localmente luego de mover el primario a donde reside el proceso que inició la operación de escritura.

Protocolos Escritura Remota (1)

Protocolo de escritura remota basado en primario con servidor fijo al cual las lecturas y escrituras son encaminadas.

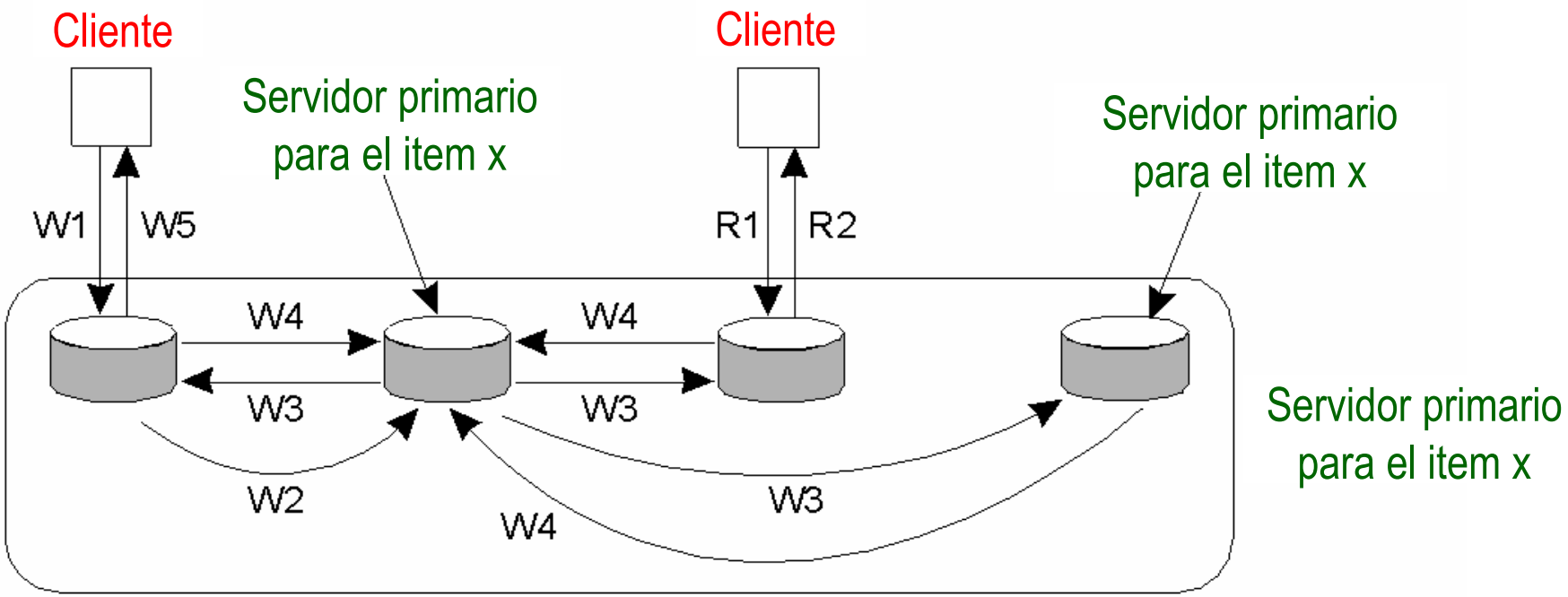


W1. Requerimiento Escritura
W2. Req. encaminado al serv. por x
W3. ACK escritura completa
W4. ACK escritura completa

R1. Requerimiento lectura
R2. Req. encaminado al serv. por x
R3. Retorno respuesta
R4. Retorno. respuesta

Protocolos Escritura Remota (2)

El principio de protocolo de respaldo primario.



W1. Requerimiento escritura

W2. Req. encaminado al primario

W3. Avisa a respaldos actualizar

W4. ACK actualización

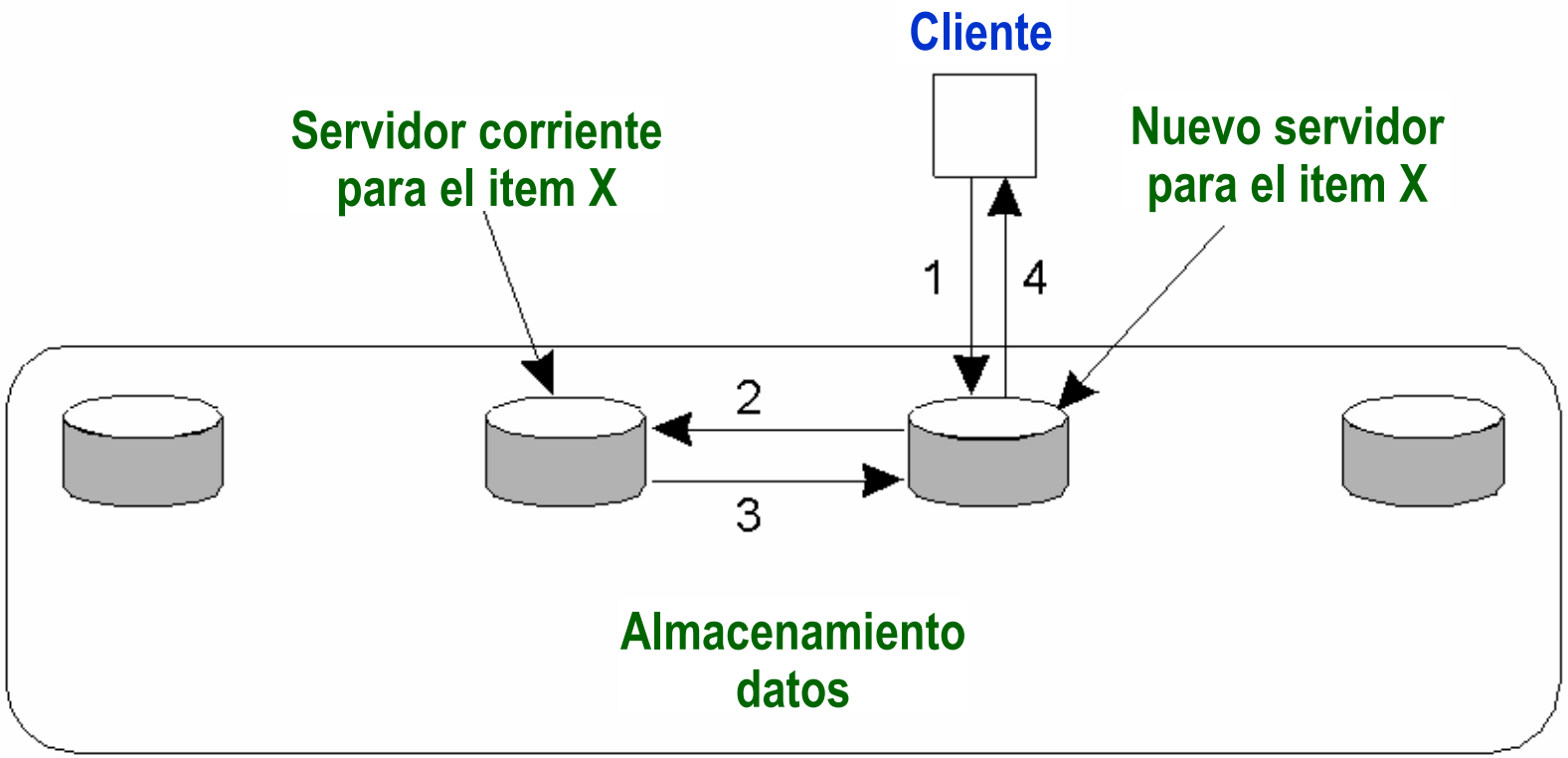
W5. ACK escritura completada

R1. Requerimiento lectura

R2. Respuesta a lectura

Protocolos de Escrituras Locales (1)

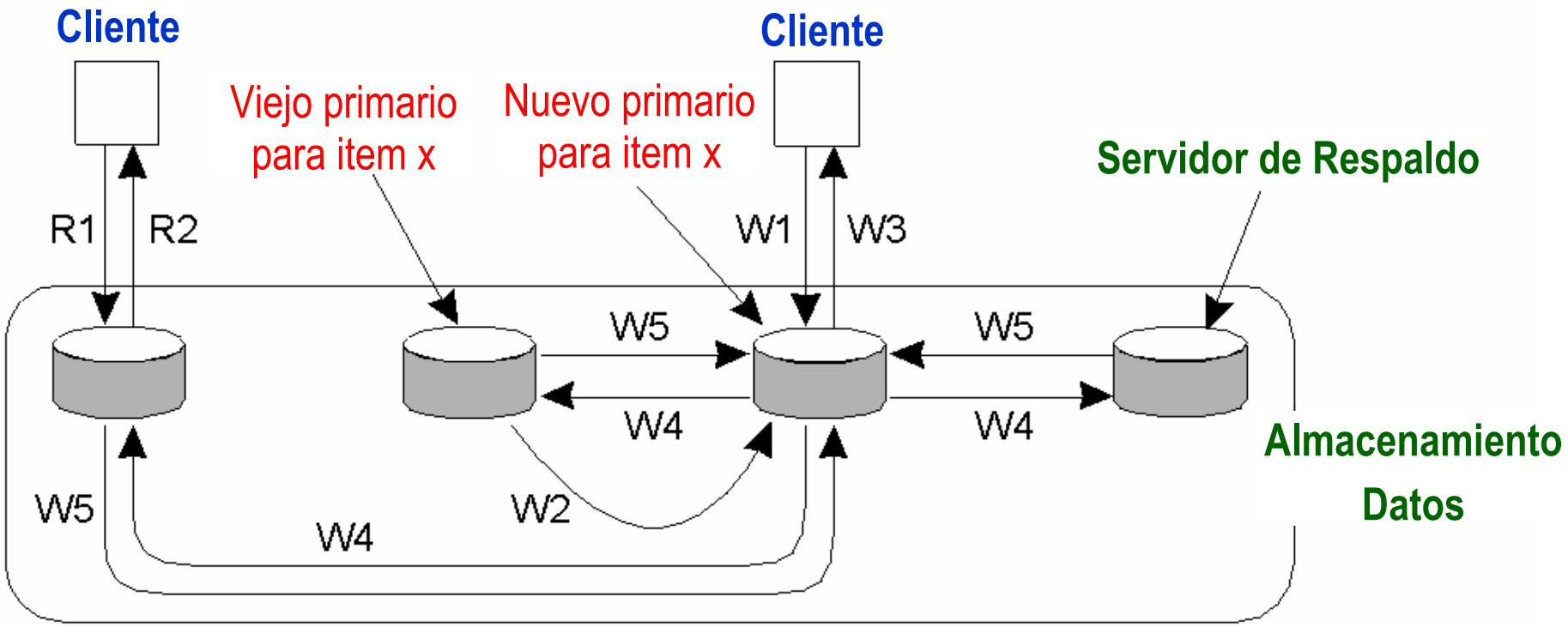
Protocolo escritura local basado en primario en el cual una única copia es migrada entre procesos.



1.Solicitud de lectura o escritura.
2.Continua solicitud al servidor corriente de x.
3.Se mueve el item x al servidor del cliente.
4.Retorna el resultado de la operación sobre el servidor del cliente.

Protocolos de Escritura Local (2)

Protocolo de respaldo primario en el cual el primario migra hacia el proceso que espera realizar una actualización.



W1.Solicitud escritura
W2.Mueve item x a un nuevo primario
W3.ACK escritura completada
W4.Actualiza los respaldos
W5.ACK actualización

R1. Solicitud de lectura
R2. Respuesta a la lectura

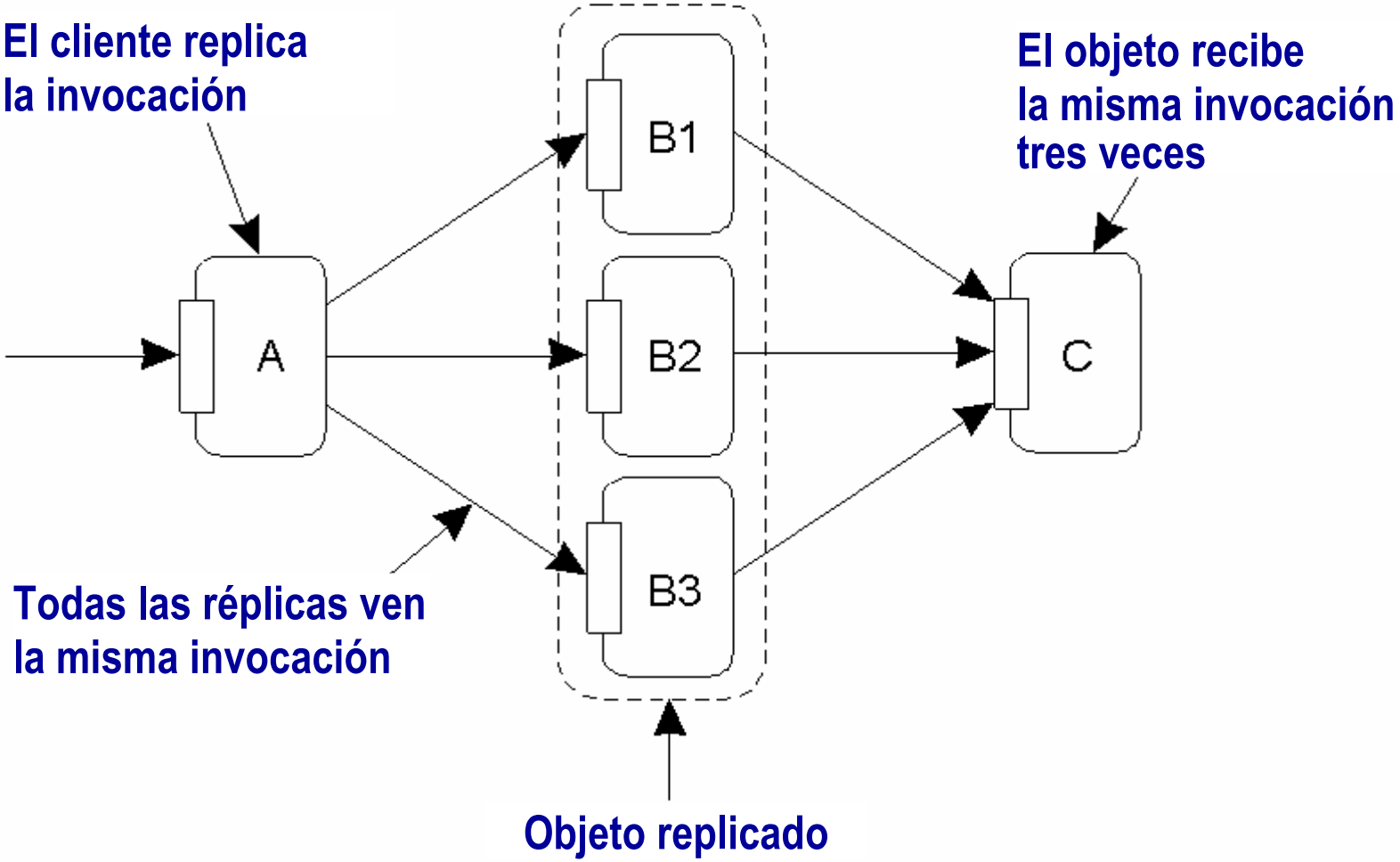
Protocolos de Escritura Replicada

En los protocolos de escrituras replicadas, las operaciones pueden ser llevadas a cabo sobre múltiples réplicas en vez de una sola como en el caso de las réplicas basadas en el primario.

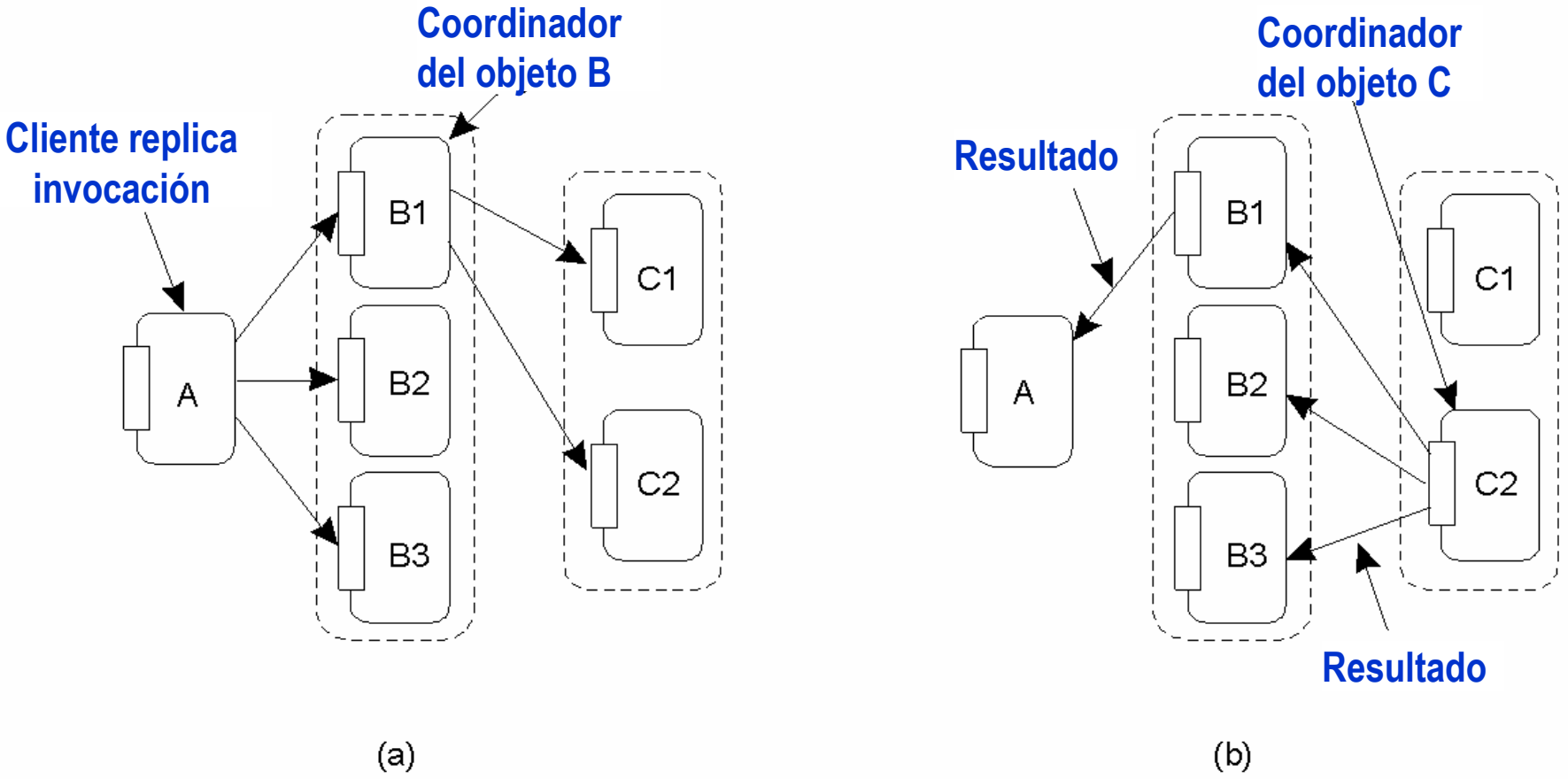
- Replicación Activa
- Protocolos basados en quorum.

Replicación Activa (1)

El problema de invocaciones replicadas.



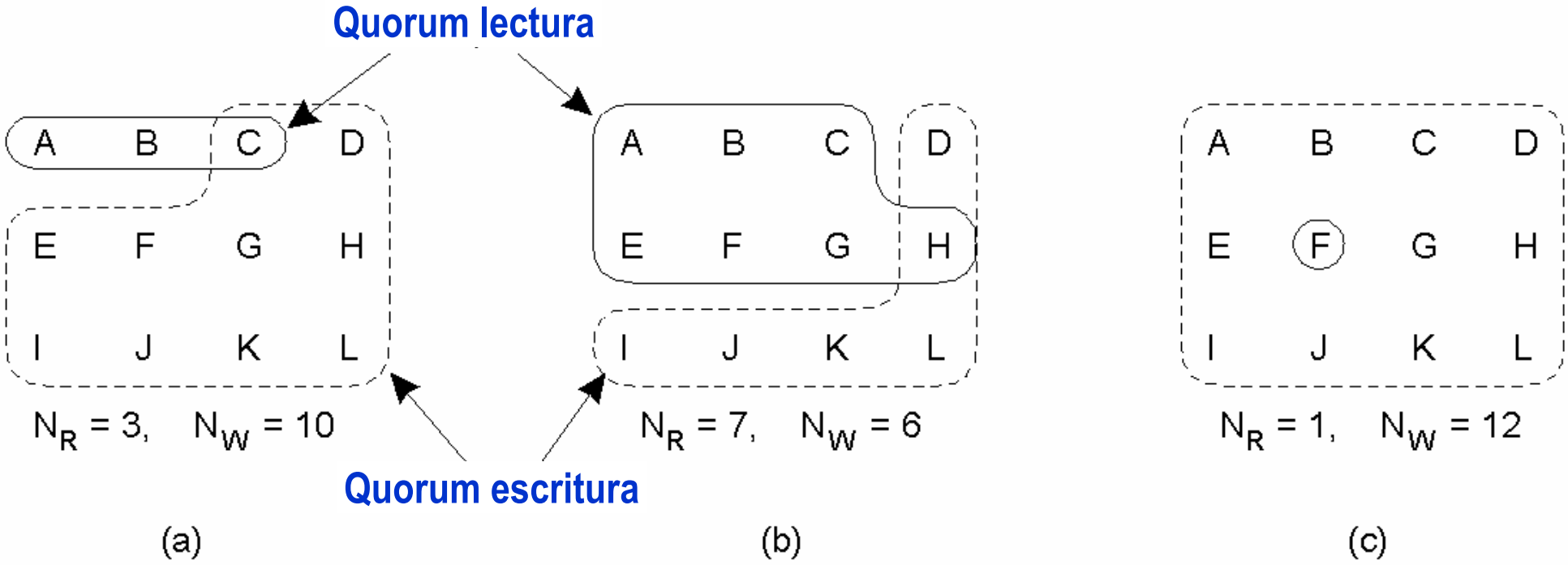
Replicación Activa (2)



a) Reenvío de una invocación desde un objeto replicado.

b) Retorno de una respuesta al objeto replicado.

Protocolos Basados en Quorum



Tres ejemplos del algoritmo de votación:

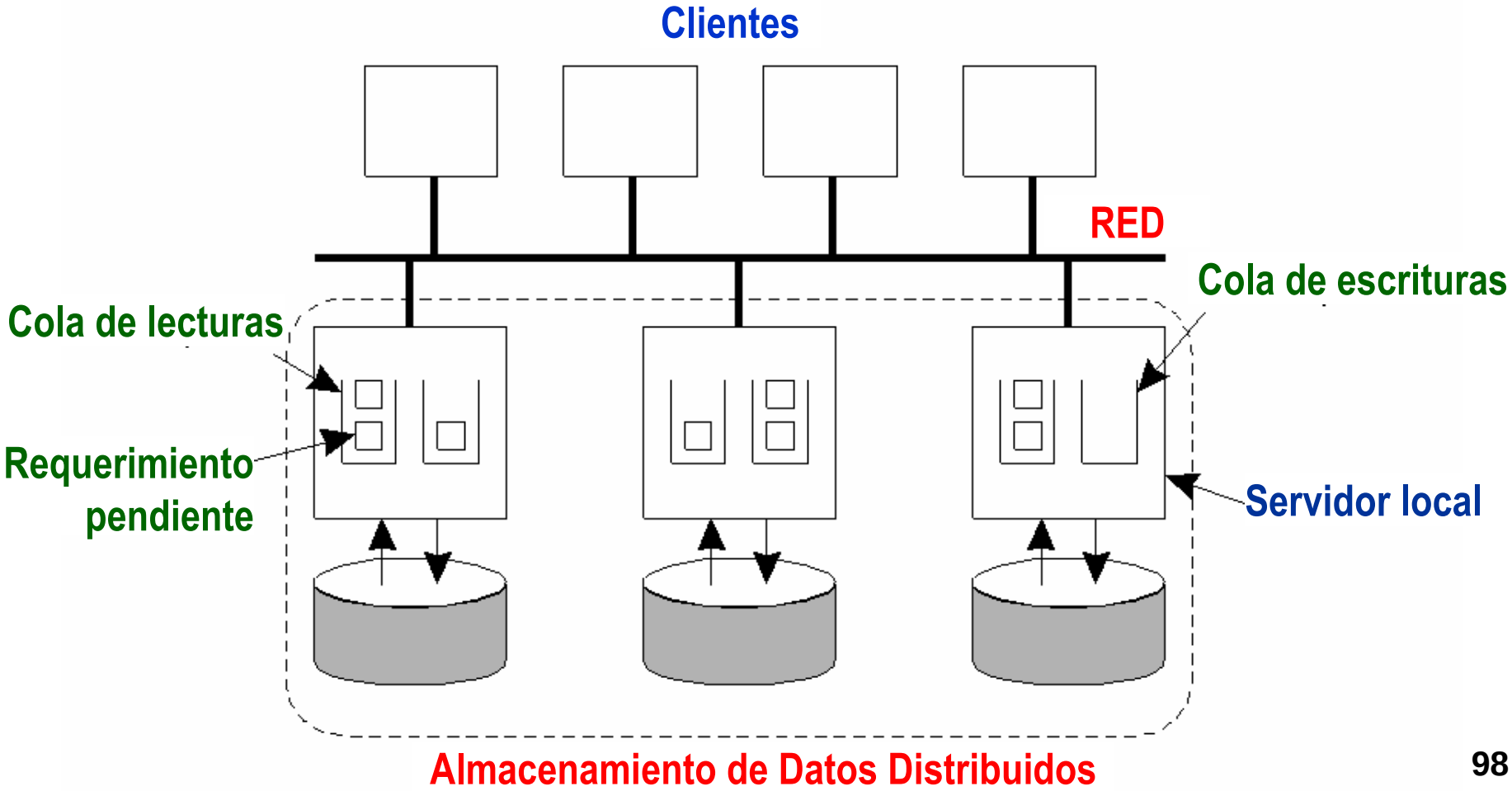
a) Una correcta elección de conjuntos de lectura y escritura.

b) Una elección que puede llevar a conflictos escritura-escritura.

c) Una elección correcta, conocida como ROWA (read one, write all).

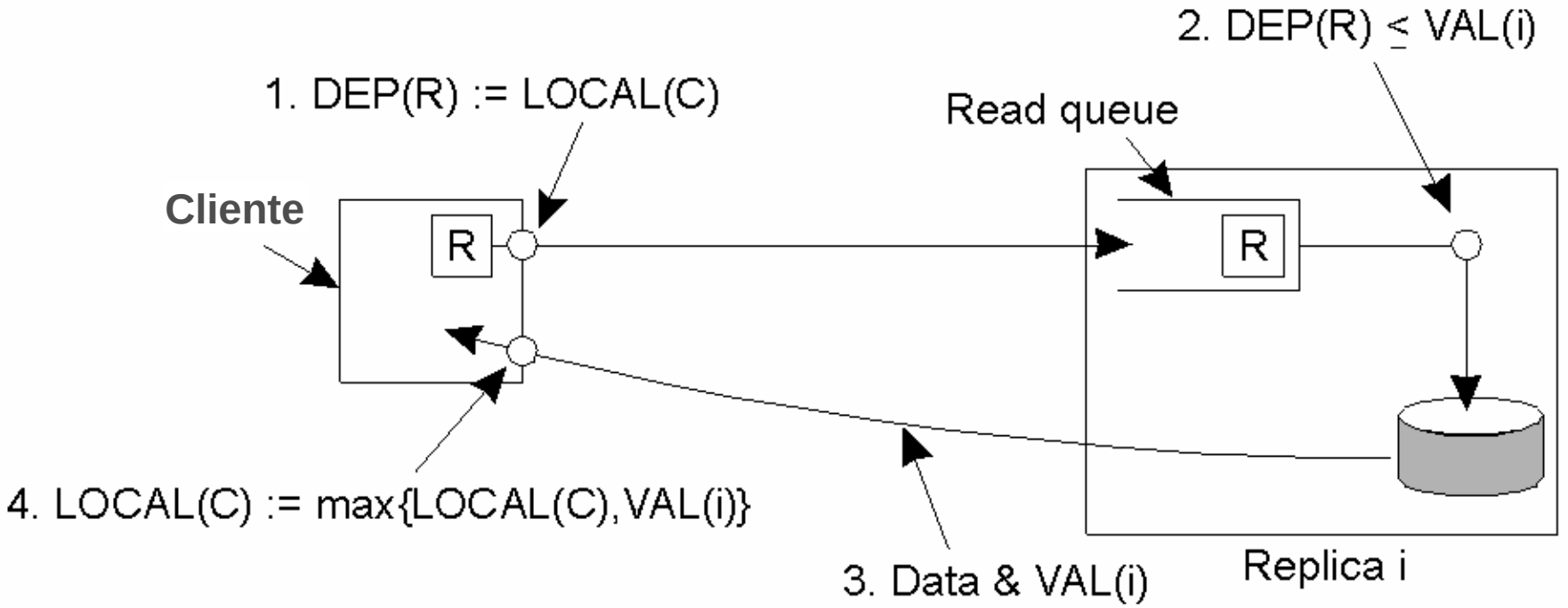
Replicación Relajada Causalmente Consistente

La organización general de un almacenamiento de datos distribuidos. Los clientes manejan la comunicación relacionada con la consistencia.



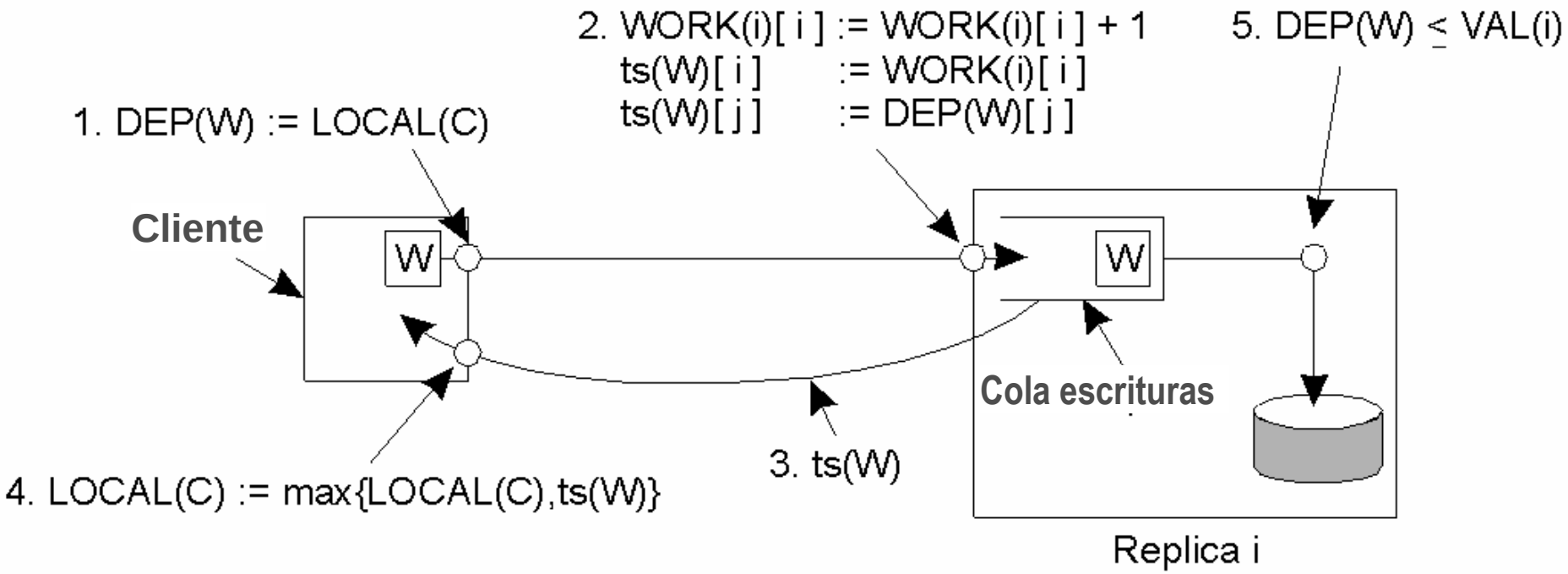
Procesando Operaciones de Lectura

- Operación de lectura sobre una copia local.



Procesando Operaciones de Escritura

Realización de una operación de escritura en la copia local.



Memoria Compartida Distribuida

Comunicación entre procesos:

- Pasaje de Mensajes
- Memoria Compartida

Memoria Compartida Distribuida (MCD):

La capa abstracta se implementa en:

- Kernel
- Rutinas en tiempo de corrida

Memoria Compartida Distribuida

La memoria se parte en “bloques” fijos.

El “caching” evita la *latencia de acceso*.

Puede soportar:

migración y/o replicación

Memoria Compartida Distribuida

Diseño e Implementación de MCD

Aspectos que deben considerarse en el diseño de una MCD.

- Granularidad.
- Estructura del espacio de memoria compartida.
- Coherencia de memoria y sincronización de acceso.
- Localización de datos y accesos.
- Estrategias de reemplazo.
- Thrashing.
- Heterogeneidad.

Memoria Compartida Distribuida

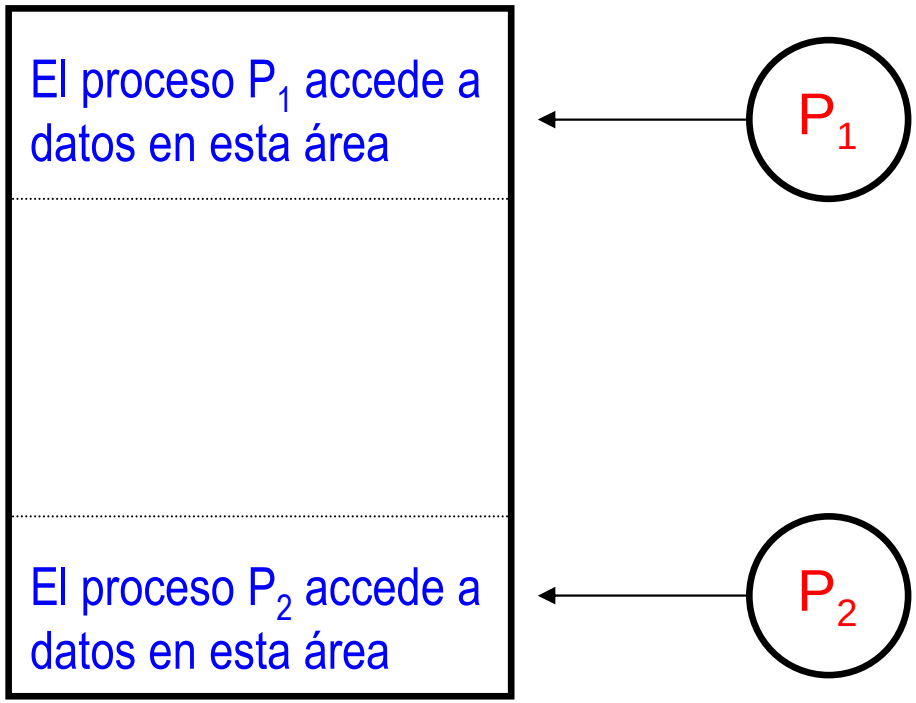
Granularidad

Problemas que se presentan con el tamaño del bloque

- Sobrecarga de paginado.
- Tamaño de directorio.
- Thrashing.
- Falso compartir.

Memoria Compartida Distribuida

Falso compartir (*false sharing*)



Memoria Compartida Distribuida

Ventajas de usar un tamaño de bloque igual a la página de memoria local:

- ⇒ Permite el uso de sistemas existentes de falta de páginas.
- ⇒ Permite el control de derechos de acceso.
- ⇒ Si las páginas pueden colocarse en un *paquete* no impone sobrecarga en la red.
- ⇒ El tamaño es adecuado con respecto a la contención de memoria.

Memoria Compartida Distribuida

Estructura del espacio de memoria compartida

La estructura define la abstracción de la vista a los programadores de aplicaciones en un sistema MCD.

Puede ser vista para unos como palabra y para otros como objetos.

La estructura y la granularidad están relacionadas.

Memoria Compartida Distribuida

No estructurada**:** es un arreglo de palabras.

Por tipo de datos: colección de objetos (Clouds y Orca) o colección variables en lenguaje fuente (Munin y Midway). La granularidad es la variable o el objeto.

Como base de datos: implica una memoria asociativa.

Memoria Compartida Distribuida

Ejemplo de Implementación

La implementación más común es el modelo de *consistencia secuencial*, porque:

- ✓ Es realizable.
- ✓ Soporta semántica más intuitiva para la coherencia de memoria.
- ✓ No impone nada extra al programador.

Desventaja: baja concurrencia.

La consistencia débil y de liberación con sus variables implícitas proveen mejor concurrencia y soportan una semántica más intuitiva.

Memoria Compartida Distribuida

Implementación del modelo de consistencia secuencial para una memoria compartida distribuida

Protocolos: dependen de la estrategia elegida.

Estrategias:

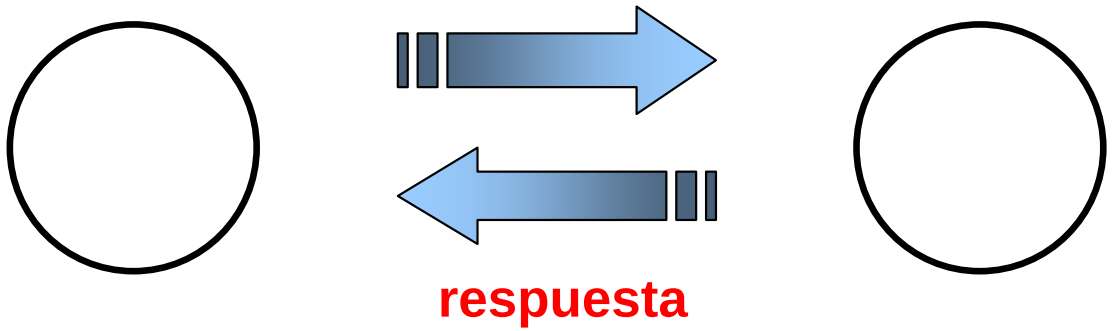
- **No réplica, no migratoria (NRNMB)**
- **No réplica, migratoria (NRMB)**
- **Réplica, migratoria (RMB)**
- **Réplica, no migratoria (RNMB)**

Memoria Compartida Distribuida

No réplica, no migratoria (NRNMB)

Nodo Cliente
Requerimiento-Respuesta

Nodo dueño del bloque
Recibe el requerimiento, hace el acceso, envia respuesta



Desventajas:

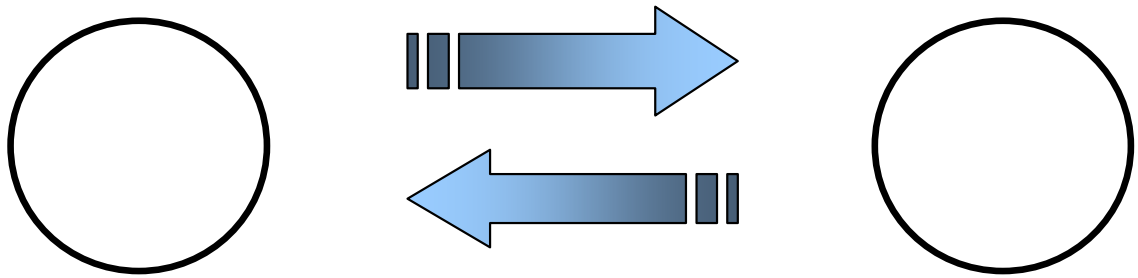
- ✓La serialización de los requerimientos crea un *cuello de botella.*
- ✓No es posible paralelismo.

Memoria Compartida Distribuida

No réplica, migratoria (NRMB)

Nodo Cliente
(Se convierte en el nuevo dueño del bloque luego de su migración)

Nodo dueño del bloque
(dueño del bloque antes de la migración)



Memoria Compartida Distribuida

Ventajas:

- ✓ No se incurre en costos de comunicación cuando un proceso accede a un dato local.
- ✓ Permite tomar ventajas de la localidad de accesos a los datos.

Desventajas:

- ✓ Problemas de *thrashing.*
- ✓ No hay paralelismo.

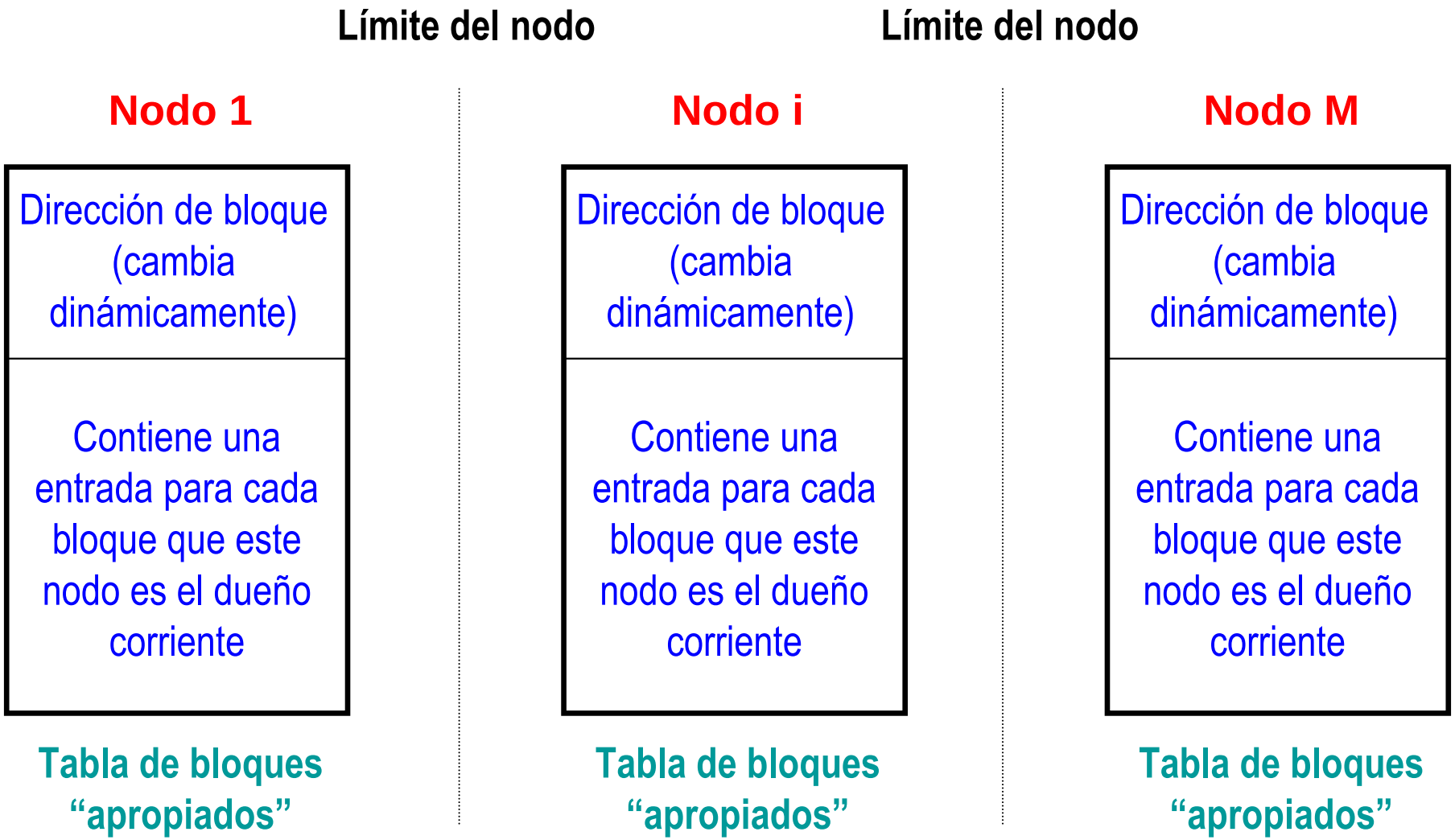
Memoria Compartida Distribuida

Localización de los bloques

1.Broadcasting

- ✓ No escala bien. El sistema de comunicaciones es *cuello de botella*.
- ✓ La latencia de la red es grande.

Memoria Compartida Distribuida



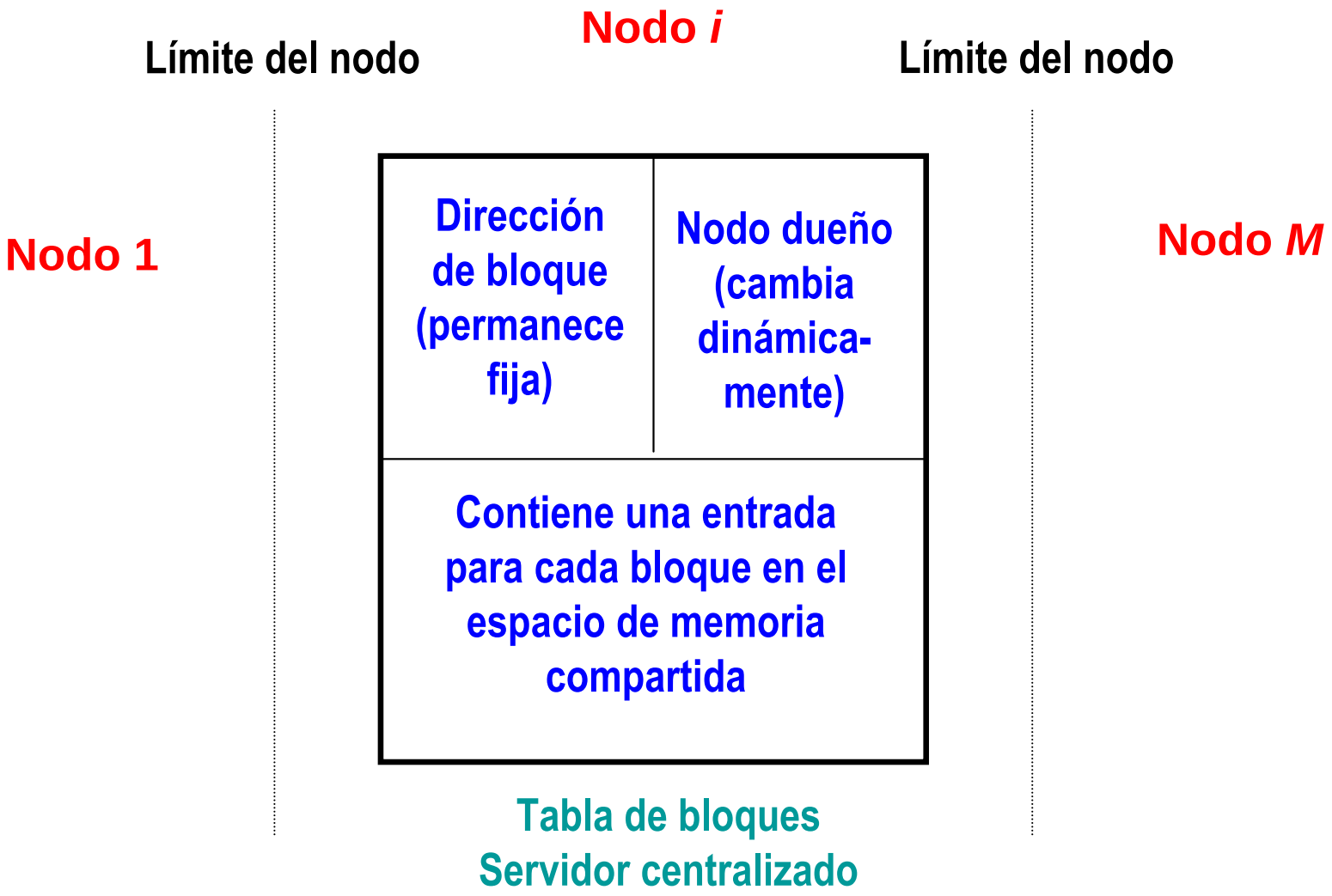
Memoria Compartida Distribuida

2.Algoritmo servidor centralizado

Todos los nodos conocen su dirección.

- ✓ Reduce paralelismo (serializa los requerimientos).
- ✓ Una falla hace caer el sistema.

Memoria Compartida Distribuida

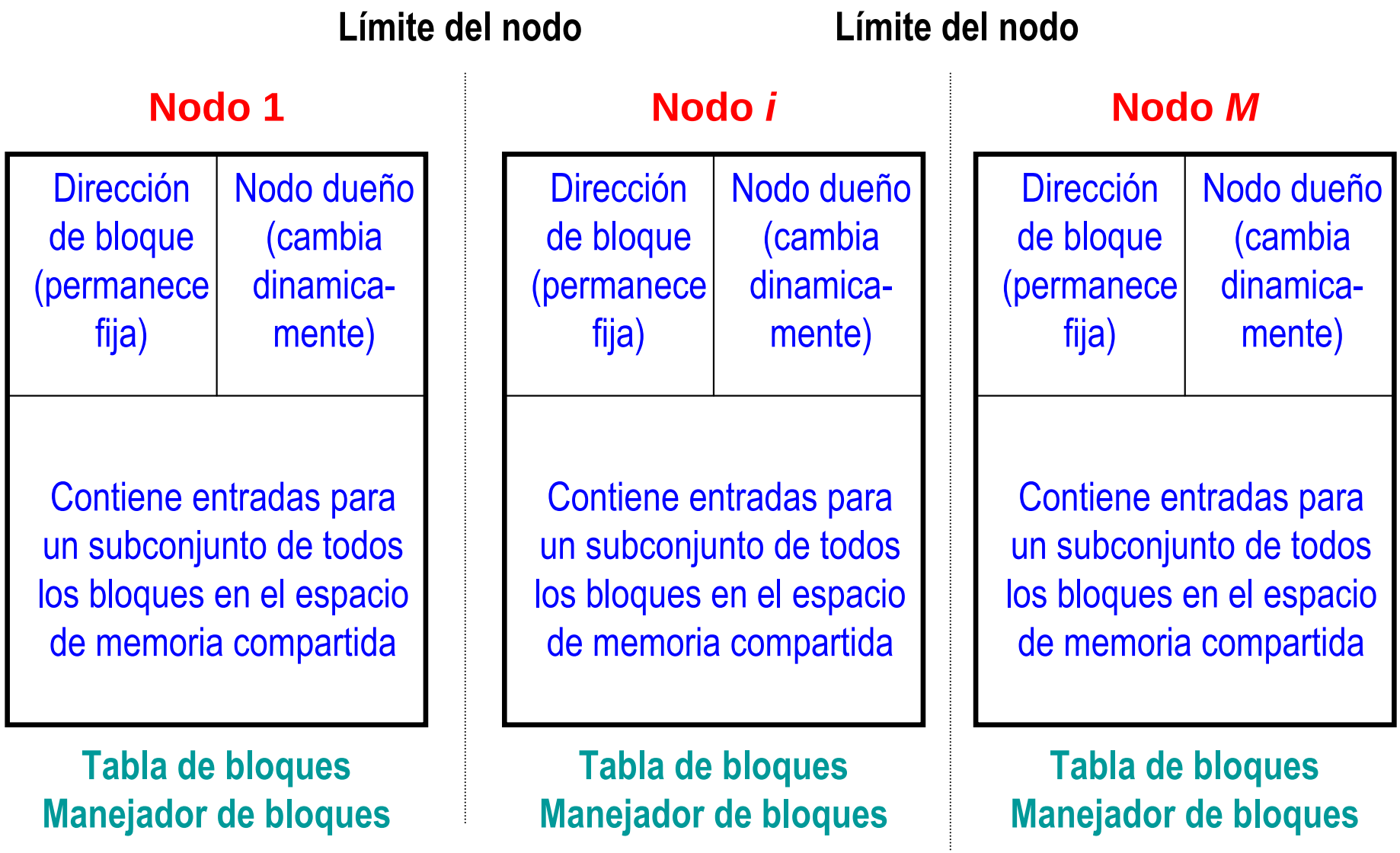


Memoria Compartida Distribuida

3. Algoritmo servidor distribuido fijo

- ✓ Distribuye el rol del servidor.
- ✓ Manejan bloques en varios nodos.
- ✓ Cada uno maneja un subconjunto determinado de bloques de datos.

Memoria Compartida Distribuida

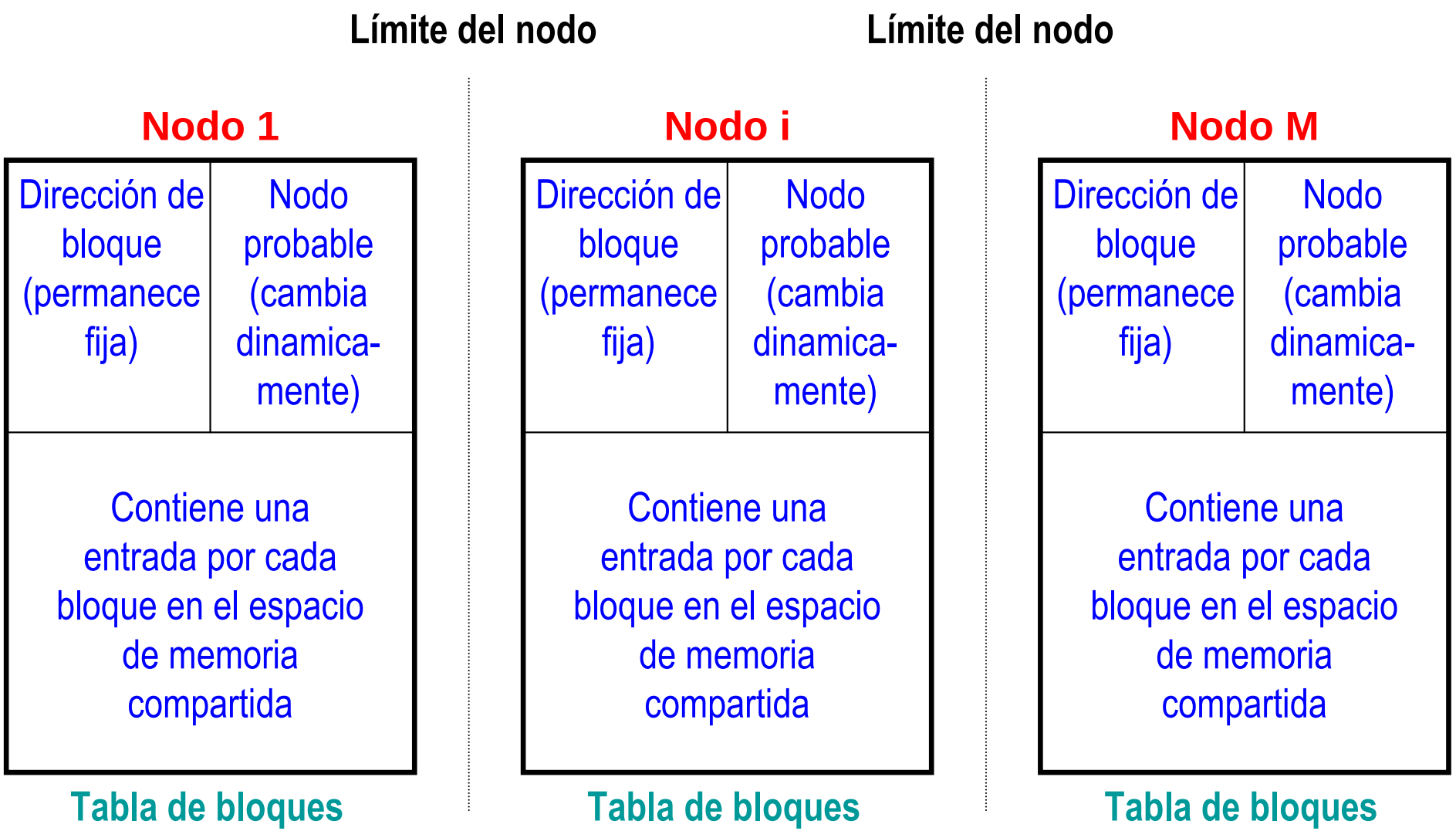


Memoria Compartida Distribuida

4. Algoritmo servidor distribuido dinámico

- ✓ Cada nodo tiene su propia tabla de información de los bloques de la MCD.
- ✓ No siempre la información de esta tabla es correcta.

Memoria Compartida Distribuida



Memoria Compartida Distribuida

Réplica, Migratorio (RMB)

Ataca la desventaja principal de las estrategias no replicantes: la reducción del paralelismo.

Se incrementa el costo de las operaciones de escritura.

Se debe mantener la **consistencia**.

La replicación complica el protocolo de coherencia de memoria.

Memoria Compartida Distribuida

Protocolos:

- Escritura invalidante.
- Escritura actualizada.

Este esquema requiere un ordenamiento total de las operaciones de escritura para lograr la consistencia secuencial.

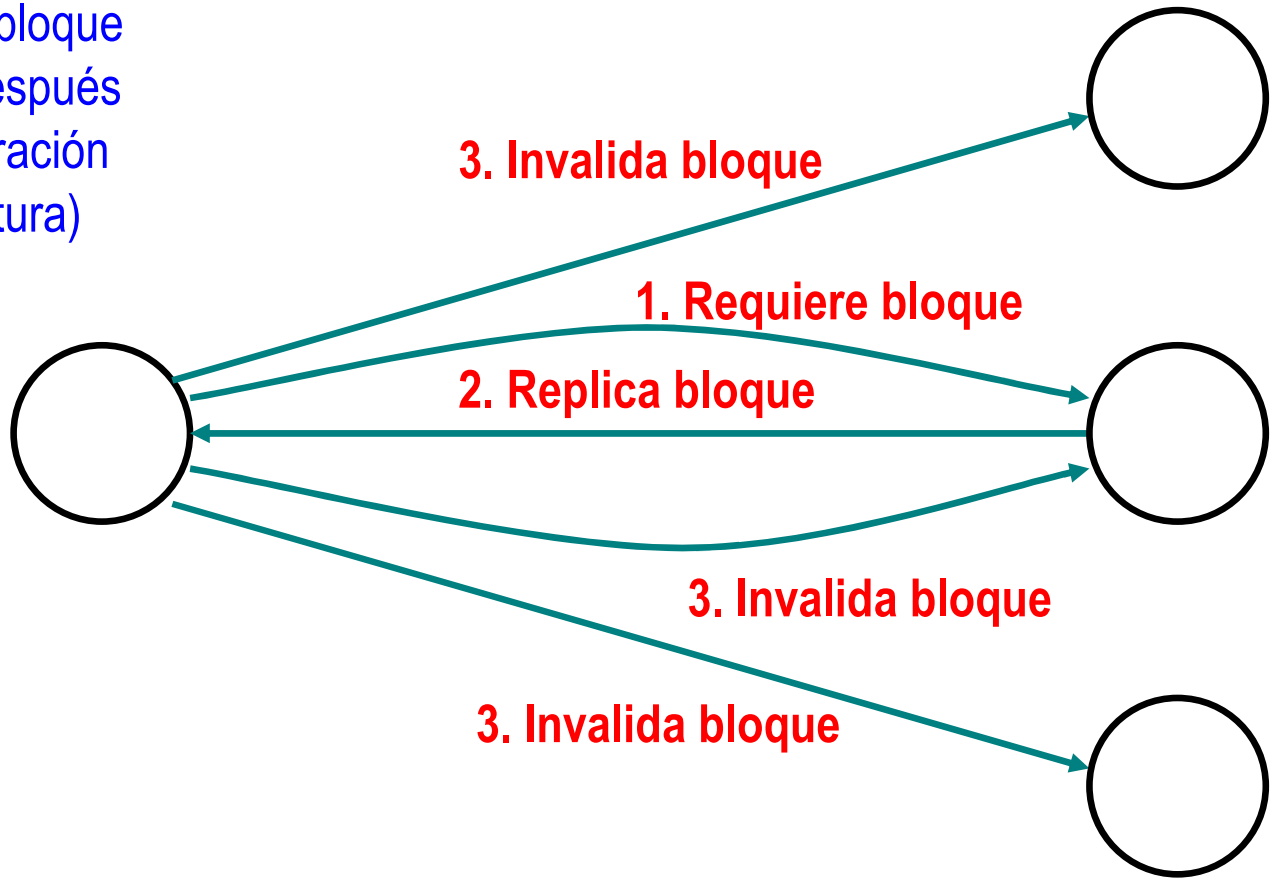
Se utiliza un secuenciador global.

Memoria Compartida Distribuida

Escritura invalidante

Nodo Cliente
(tiene la copia válida del bloque de dato después de la operación de escritura)

Nodos teniendo copias válidas de bloques de datos antes de escritura

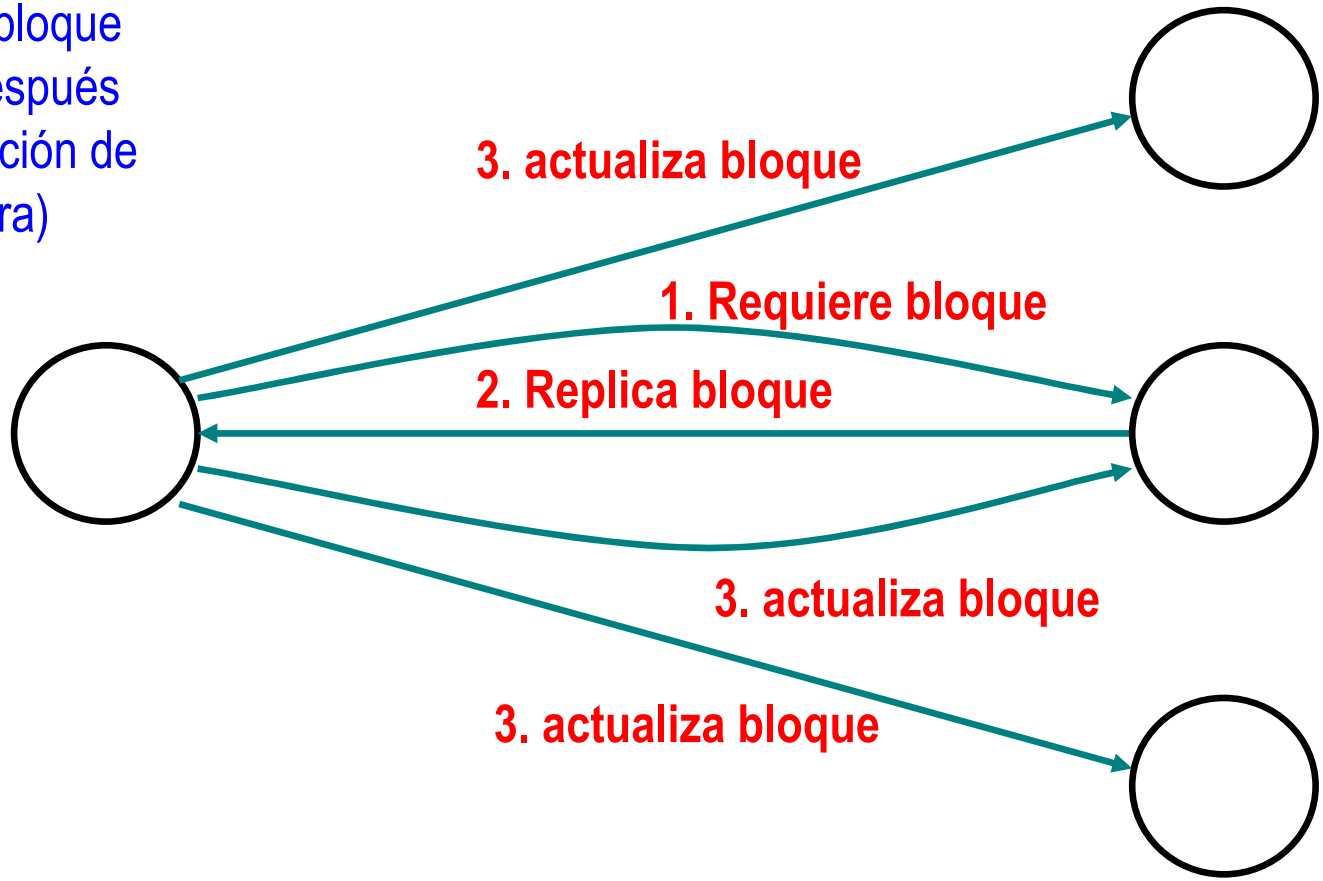


Memoria Compartida Distribuida

Escritura actualizada

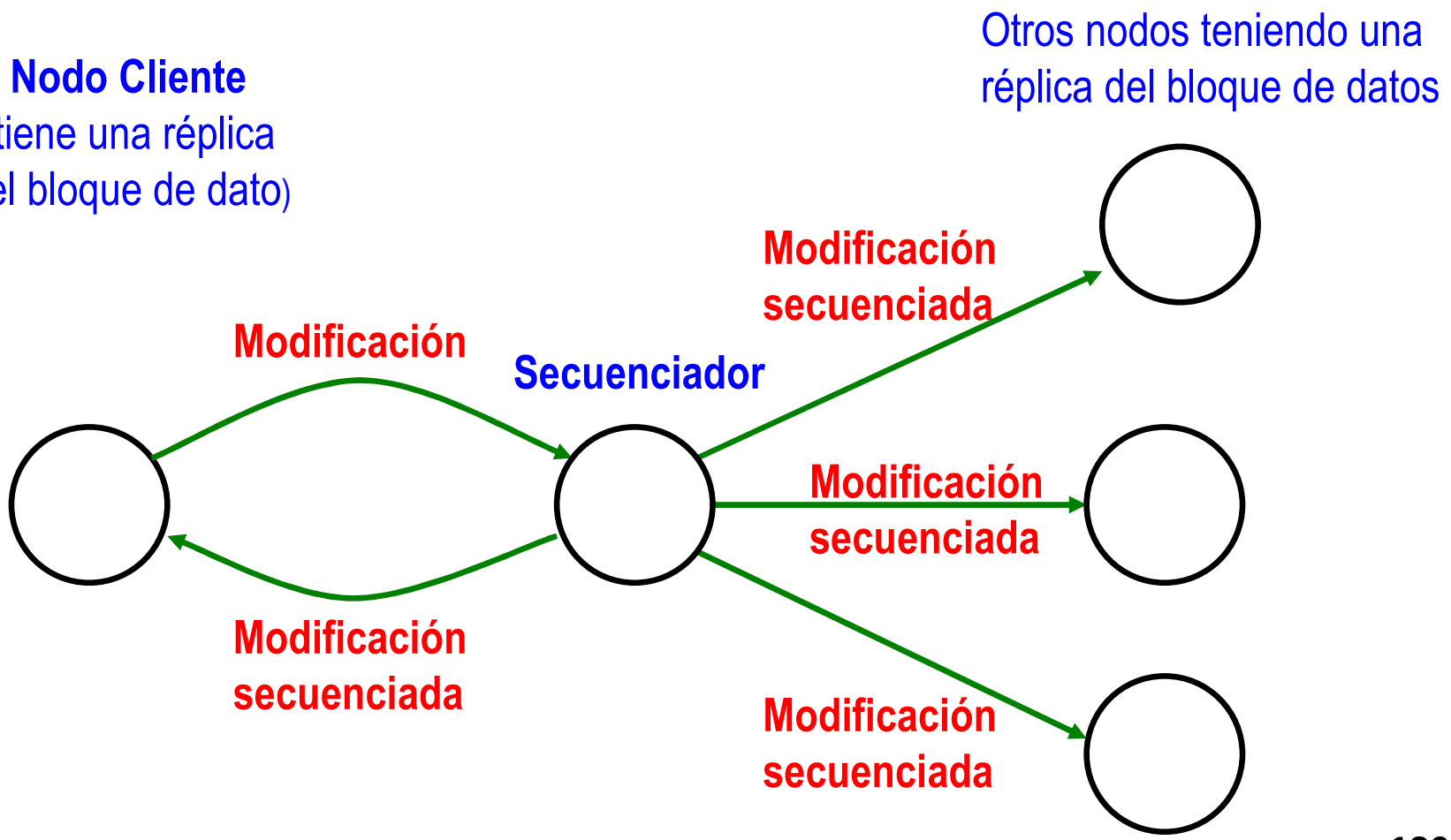
Nodo Cliente
(tiene la copia válida del bloque de dato después de la operación de escritura)

Nodos teniendo copias válidas de bloques de datos antes de escritura



Memoria Compartida Distribuida

Mecanismo secuenciador



Memoria Compartida Distribuida

Localización de datos en RMB

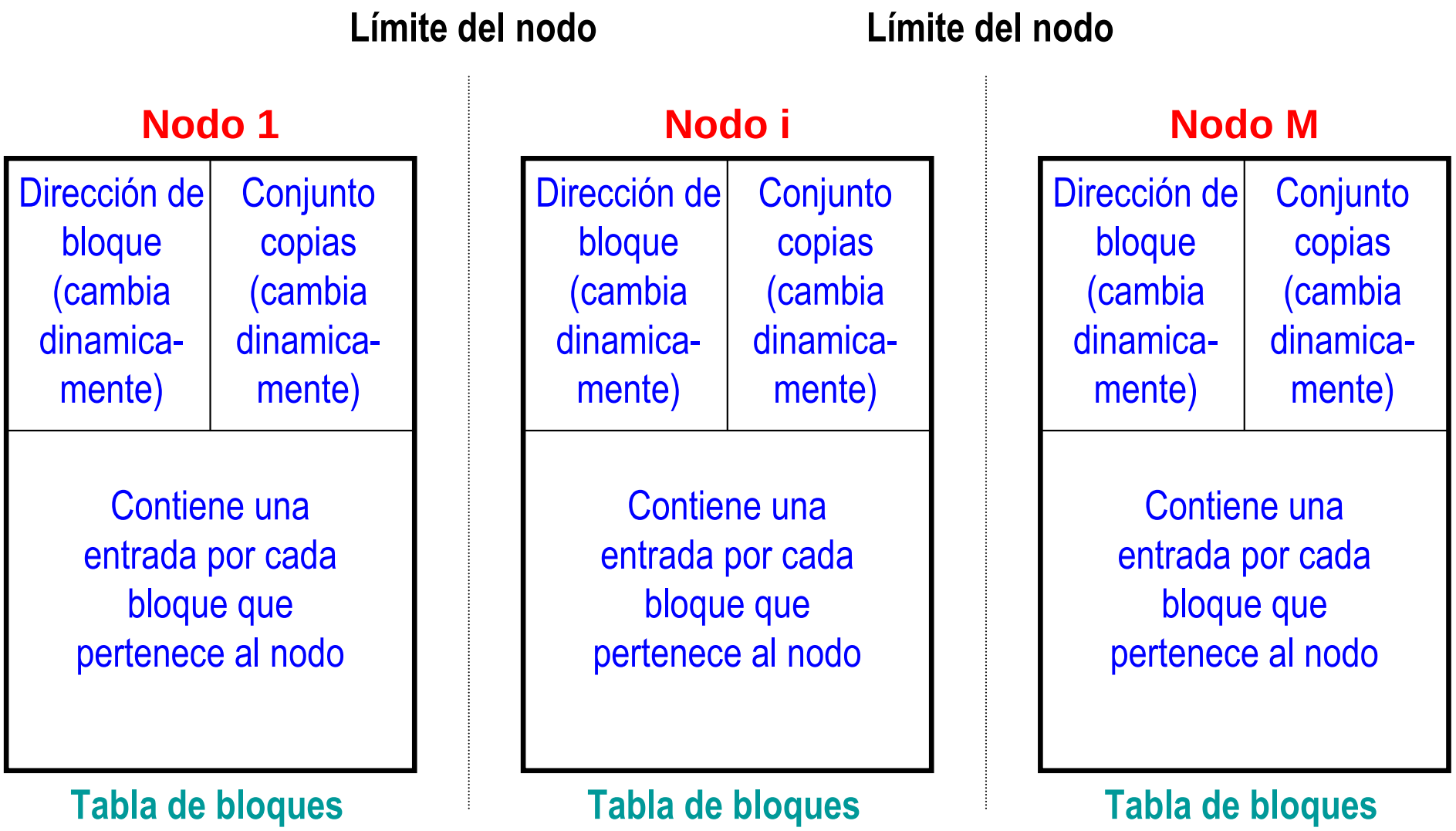
Necesita:

1. Localizar al dueño de un bloque.
2. Tener la pista de los nodos que corrientemente tienen una copia válida del bloque.

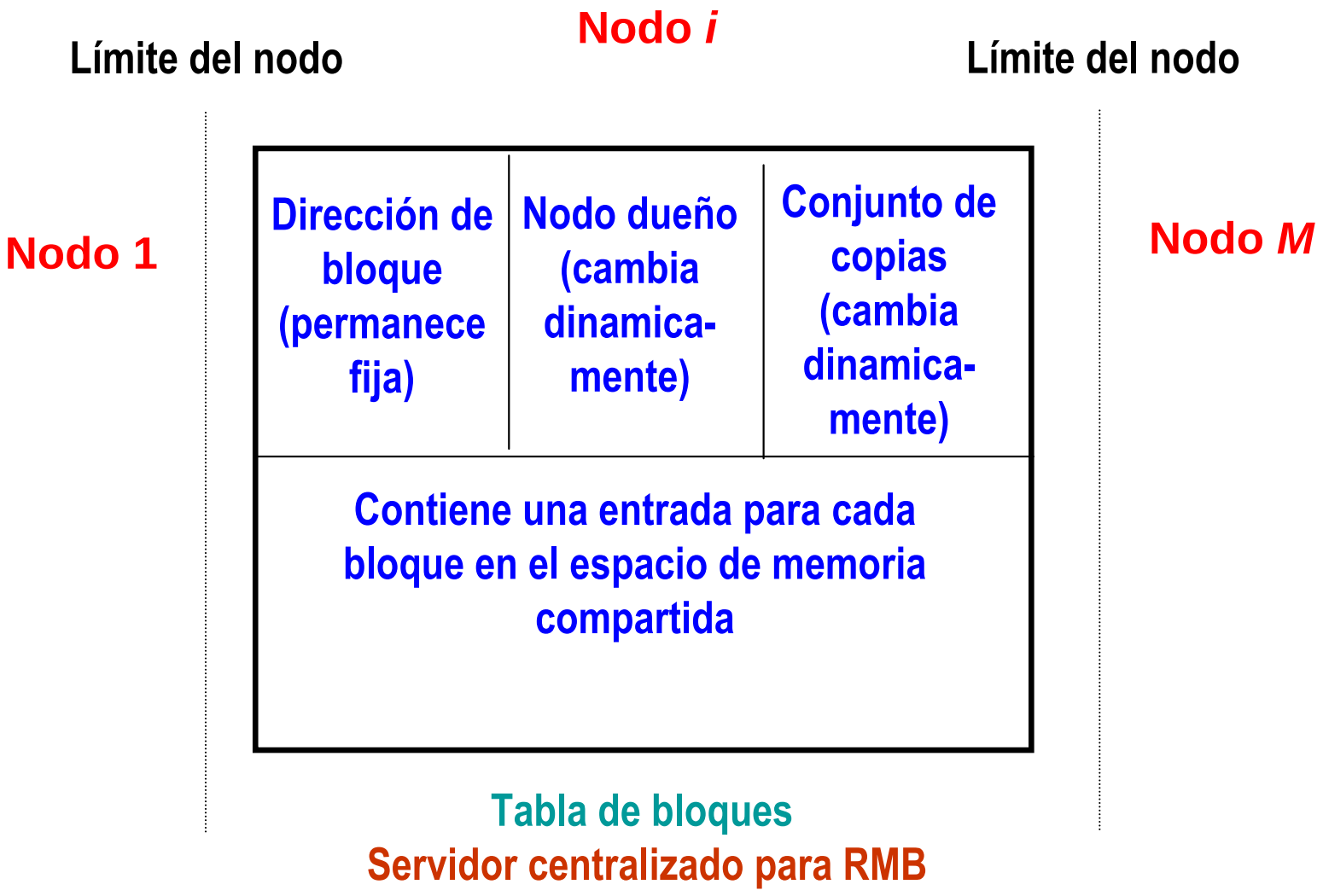
Algoritmos usados:

- Broadcasting.
- Algoritmo servidor centralizado.
- Algoritmo servidor distribuido fijo.
- Algoritmo servidor distribuido dinámico.

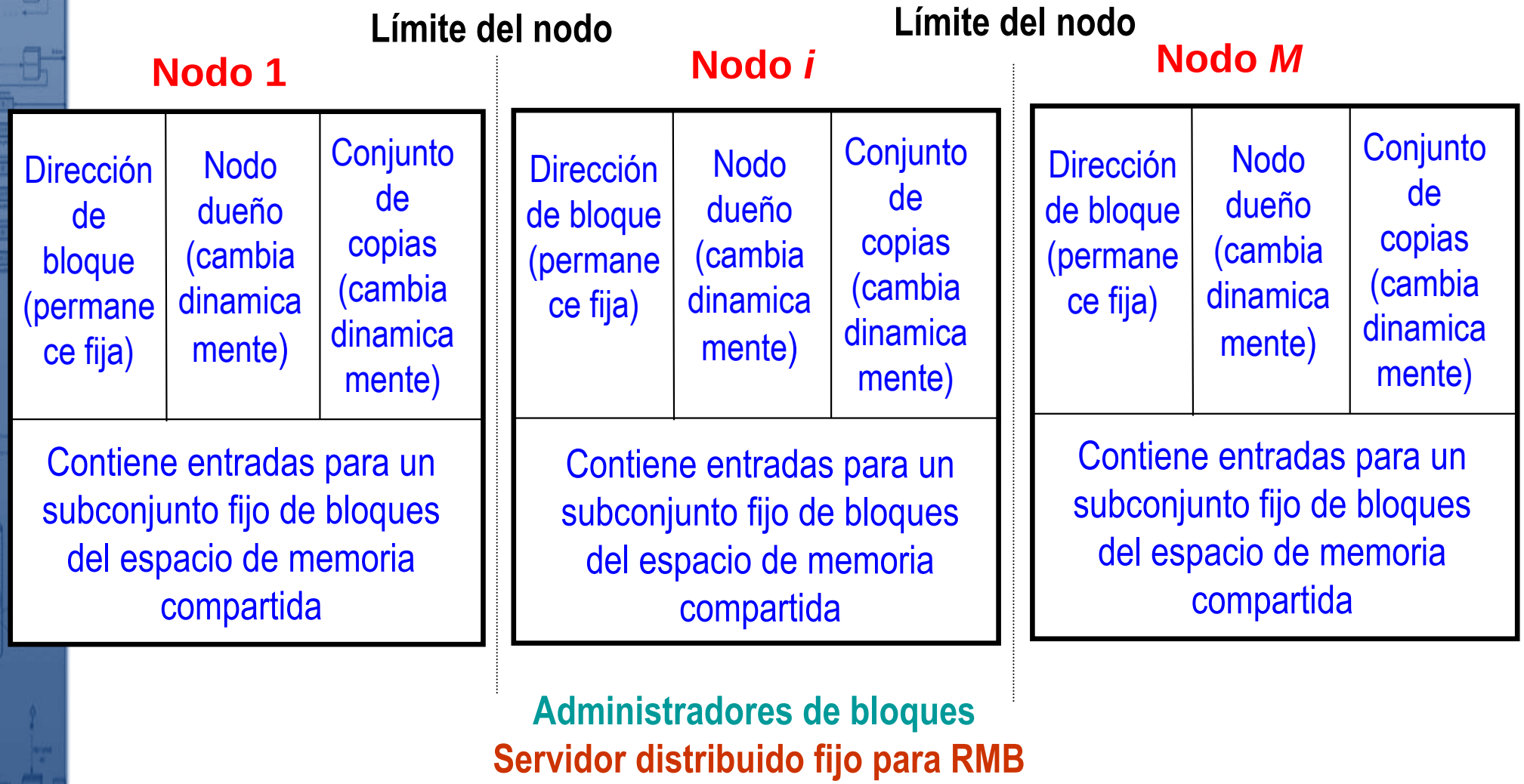
Memoria Compartida Distribuida



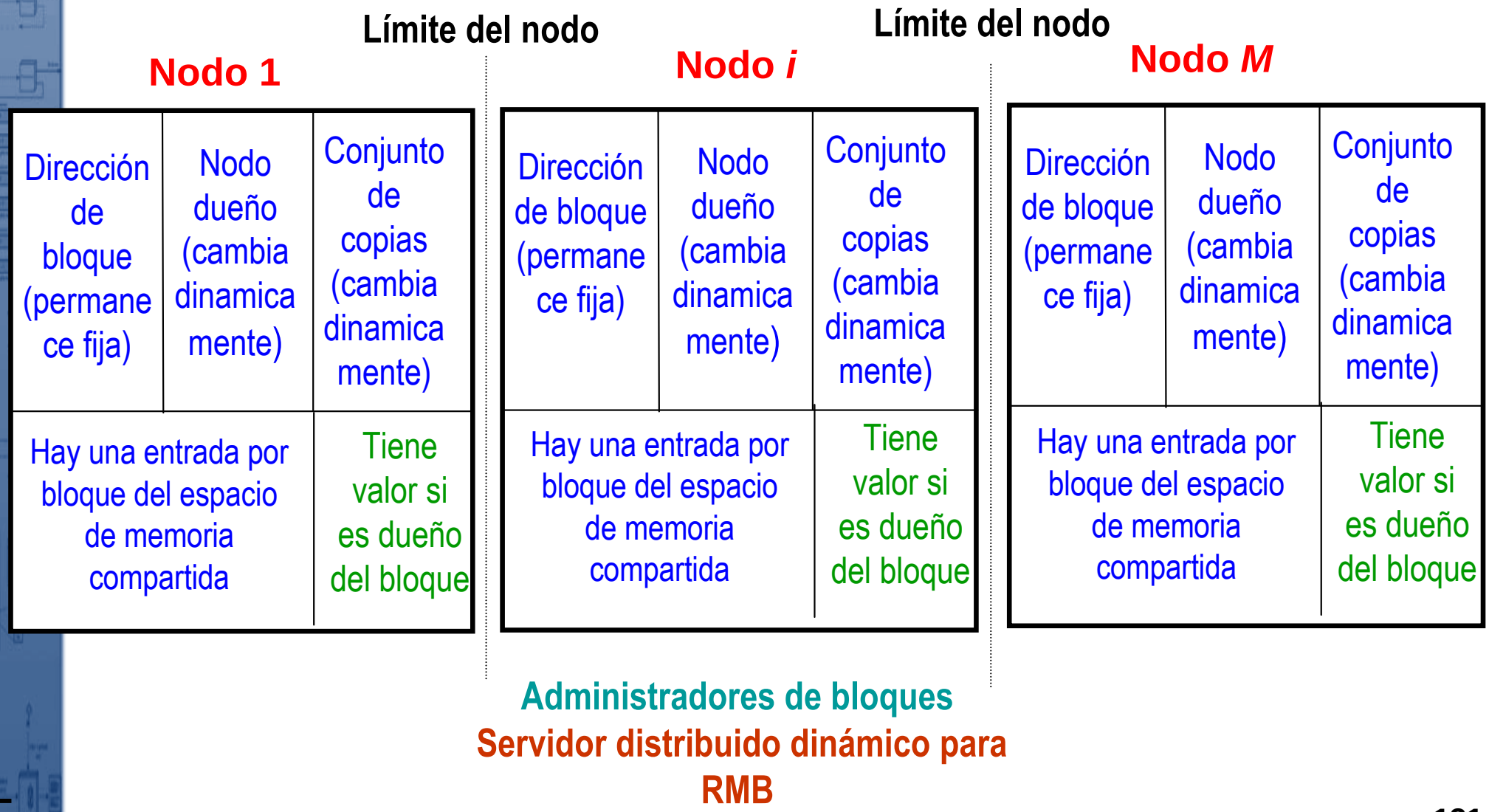
Memoria Compartida Distribuida



Memoria Compartida Distribuida



Memoria Compartida Distribuida



Memoria Compartida Distribuida

Para reducir la cadena de nodos a atravesar para alcanzar el verdadero dueño del bloque, la tabla de bloques del nodo es adaptada de la siguiente forma:

a) Siempre que el nodo recibe un requerimiento de invalidación.

b) Siempre que el nodo pasa la propiedad.

c) Siempre que el nodo pasa un requerimiento que falla.

Memoria Compartida Distribuida

Réplica, no migratoria (RNMB)

La ubicación de cada réplica es fija.

Siempre que se escribe se actualiza.

Localización de datos en RNMB

a) Las localizaciones de las réplicas nunca cambian.

b) Todas las réplicas se mantienen consistentes.

c) Sólo un requerimiento de lectura puede ser directamente enviado a uno de los nodos teniendo una réplica del bloque y todos los requerimientos de escritura deben primero ser enviados al secuenciador.

Memoria Compartida Distribuida

Estructura:

- ✓ Una tabla de bloques por cada nodo.
- ✓ Una tabla de secuencias en el secuenciador.

La tabla del secuenciador tiene los siguientes campos:

a) Dirección del bloque.

b) Conjunto de réplicas.

c) Número de secuencia para cada bloque.

Memoria Compartida Distribuida

Estrategias de reemplazo

a) Qué bloque tiene que ser reemplazado.

b) Dónde debe ubicarse el bloque reemplazado.

¿Qué bloque tiene que ser reemplazado?

a) Usado vs. no usado (LRU).

b) Espacio fijo vs. espacio variable.

Memoria Compartida Distribuida

En algunos sistemas cada bloque es clasificado:

- No usado.
- Nil (invalidado).
- Read-only.
- Read-owned: RO y el nodo es dueño.
- Escribible.

De acuerdo a esto, la prioridad de reemplazo sería:

1. No usado y Nil.
2. Read-only.
3. Read-owned.
4. Escribible.

Memoria Compartida Distribuida

Ubicación el bloque reemplazado

- Usar almacenamiento secundario.
- Usar memoria de otros nodos.

Memoria Compartida Distribuida

Thrashing

Cuando hay migración y se producen las siguientes situaciones:

- Datos entrelazados entre distintos nodos.
- Bloques con permiso RO son repetidamente invalidados inmediatamente que son replicados.

Implica poca **localidad**.

Memoria Compartida Distribuida

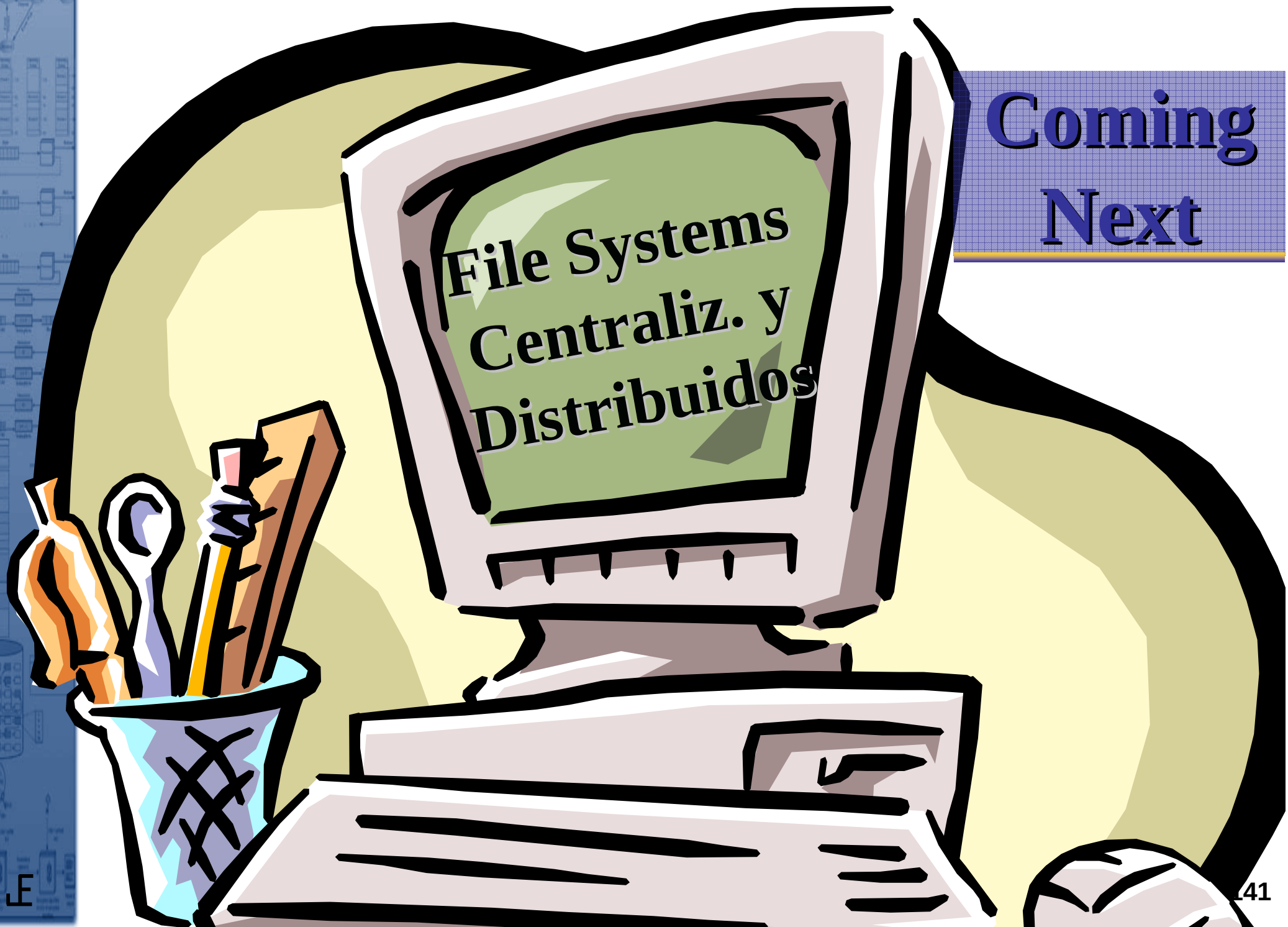
Soluciones:

1. Proveer locks de aplicación controlada: fija el bloque por un tiempo.
2. No permitir que quiten, por un tiempo *t*, un bloque del nodo (*t* está dado por estadística o por accesos).
3. Ajustar el algoritmo de coherencia para usar modelos de datos compartidos.

Memoria Compartida Distribuida

Ventajas de la Memoria Compartida Distribuida

- Abstracción más simple.
- Mejor portabilidad de programas de aplicación distribuidos.
- Mejor desempeño de algunas aplicaciones. Lo hace posible la *localidad de los datos*, el *movimiento de datos por demanda* y *espacio de direcciones más grande*.
- Ambiente de comunicación más flexible.
- Facilidad para la migración de procesos.



Coming Next