Control de Concurrencia

- Bloqueos
- Control de Concurrencia Optimista

- Fuente principal:
  - Tanenbaum. Sistemas Operativos Distribuidos.
    - Capítulo 3.4.4, pág. 154

Control de concurrencia.

- Problema similar al de la sección crítica. Un proceso impide, de alguna manera, el uso de un conjunto de datos a otros procesos que realizan una determinada acción.
- Los problemas que se pueden dar en transacciones concurrentes por falta de aislamiento son:
  - Pérdida de actualización (lost update).
    - se producen por dos lecturas concurrentes del mismo dato por parte de dos transacciones y este es actualizado por ambas.
  - Datos inconsistentes (inconsistent retrieval).
    - se producen por acceder a valores intermedios de una transacción.

Problema “lost update”

a.Deposit(200)

```
bal= a.GetBalance()          (1)

a.SetBalance(bal+200)        (3)
```

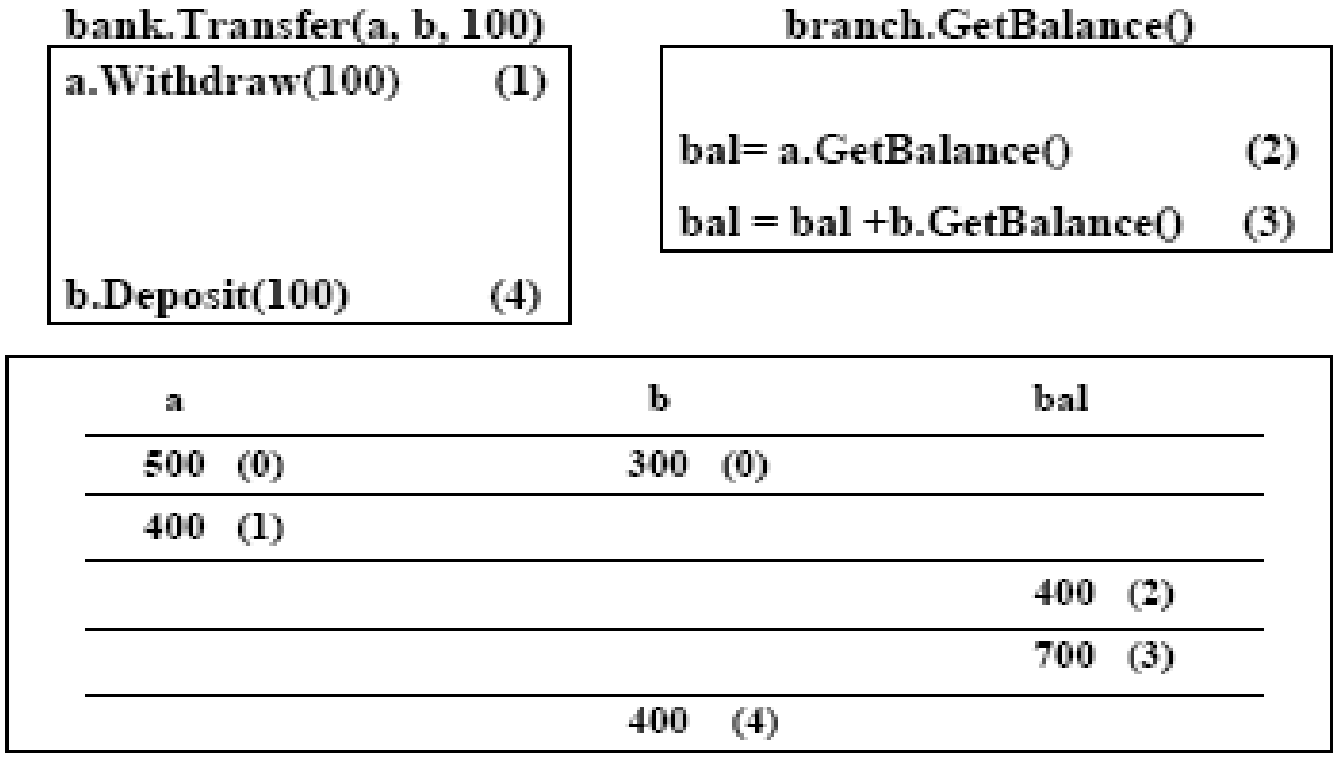
a.Deposit(300)

```
bal= a.GetBalance()          (2)

a.SetBalance(bal+300)        (4)
```

bal	a	bal
	500 (0)	
500 (1)		
		500 (2)
	700 (3)	
	800 (4)	

Problema “inconsistent retrieval”



- Problema: El acumulado de las cuentas es 700 cuando debe ser 800, porque se tomó el dato en el medio de una transacción

Protocolos para control de concurrencia.

- Clasificación:
  - Pesimistas: Cuando dos transacciones acceden al mismo dato se retrasa a una de ellas hasta que la otra finalice.
    - Ejemplo: cerraduras (cerrojos, locks).
    - Puede haber interbloqueos (deadlocks).
  - Optimistas: Se deja acceder libremente a cualquier dato. Cuando la transacción finaliza se efectúa un test para averiguar si hubo conflicto, en cuyo caso se aborta. Si no hay problemas se realiza commit.
    - Ejemplo: protocolo marcas de tiempo (timestamps), que se explica más adelante.
    - Pueden provocar un número elevado de aborts.

Transacciones *T* y *U* con locks (cerraduras) exclusivos

Transaction *T*:		**Transaction *U*:**	
balance = b.getBalance() *b.setBalance(bal*1.1)* *a.withdraw(bal/10)*		*balance = b.getBalance()* *b.setBalance(bal*1.1)* *c.withdraw(bal/10)*	
Operations	Locks	Operations	Locks
openTransaction			
bal = b.getBalance()	lock *B*		
*b.setBalance(bal*1.1)*		*openTransaction*	
a.withdraw(bal/10)	lock *A*	*bal = b.getBalance()*	waits for *T*'s lock on *B*
closeTransaction	unlock *A*, *B*	• • •	
			lock *B*
		*b.setBalance(bal*1.1)*	
		c.withdraw(bal/10)	lock *C*
		closeTransaction	unlock *B*, *C*

Compatibilidad en locks

For one object		*Lock requested*	
		read	*write*
Lock already set	*none*	OK	OK
	read	OK	wait
	write	wait	wait

Control basado en cerraduras simple.

- En su forma más sencilla, cuando un proceso necesita leer o escribir un archivo u otro objeto como parte de su transacción, primero lo cierra.
- Cerradura por:
  - Controlador centralizado
  - Controlador en cada máquina, que maneje los archivos locales.
- Es muy restrictivo.

Control basado en cerraduras (L/E)

- L/E. Lecturas / Escrituras.
- Granularidad. Pueden realizarse a nivel DB, archivo o registro.
- La cerradura de grano fino necesita mayor número de cerraduras, es más cara y es más probable que ocurran bloqueos.
- Cuando una transacción accede a un dato solicita una cerradura en el modo correspondiente (L/E).
- Si la cerradura es compatible con los cerrojos activos (lectura con lectura) sobre el dato, se concede, en caso contrario la transacción se bloquea.

Control basado en cerraduras (L/E)

- Two phase locking: El proceso adquiere todas las cerraduras durante la fase de crecimiento (growing) y las libera en la de contracción (shrinking), absteniéndose de actualizar todos los archivos hasta que pase a la fase de contracción.

- Strict two phase locking: La fase de contracción se realiza al momento de terminar la ejecución y se ha realizado commit o abort.

Control de concurrencia optimista.

- Kung y Robinson, 1981. Se crea porque el control de concurrencia pesimista impone sobrecarga, cuando hay pocos conflictos.
- El proceso realiza la tarea y al finalizar la misma revisa si hubo un problema.
- Mantiene un registro de los archivos leídos o en los que ha escrito algo. En el momento del commit, verifica todas las otras transacciones para ver si alguno de los archivos ha sido modificado desde el inicio de la transacción. Si lo fue aborta.
- Útil cuando las transacciones concurrentes no utilizan los mismos archivos.

Control de concurrencia por marcas de tiempo.

- Reed (1983). Al momento de BEGIN TRANSACTION se coloca tiempo Lamport, con marcas para lectura y escritura que indican el timestamp de la última transacción que realizó la L/E.
- Si hay problemas (una transacción iniciada posteriormente a la activa quiere entrar al archivo o accedió y realizó commit), la transacción aborta.
- Similar al Control Optimista solo que en este caso puede utilizar los mismos archivos, siempre que la transacción con el número menor esté primero.

Bloqueos.

- Se produce cuando dos procesos intentan adquirir la misma pareja de cerraduras, pero en orden opuesto.

Bloqueos.

- Pueden producirse en:
  - Comunicación. Proceso A envía mensaje a B, que lo hace a C y este a A.
  - Recursos. Acceso exclusivo a dispositivo E/S.
- Estrategias:
  - Ignorar problema (avestruz)
  - Detección. Permitir que ocurran, detectarlos e intentar corregirlos.
  - Prevención. Lograr que sean imposibles desde el punto de vista estructural.
  - Evitarlos. Asignar cuidadosamente los recursos. No es utilizado ya que se necesita conocer de antemano la necesidad de recursos.

Bloqueos. Detección.

- Detección/resolución
  - En un SO convencional se eliminan uno o más procesos.
  - Se permiten los interbloqueos, pero hay un algoritmo encargado de reconocerlos y eliminarlos.
  - Se mantiene el grafo de esperas y cuando el coordinador detecta un ciclo, se aborta una de las transacciones

Bloqueos. Detección.

– Métodos.

- **Centralizado**. Los procesos envían al coordinador los mensajes de cambios en los recursos que utiliza. Hay problemas con falsos bloqueos, que se pueden solucionar si se agrega tiempo Lamport y si el coordinador antes de bloquear pregunta: “Si hay algún proceso con mensaje con marca anterior envíelo de inmediato”.

- **Distribuido**. Chandy et al. 1983. Se agrega un mensaje de exploración con 3 números:
  - Proceso recién bloqueado
  - Proceso que envía el mensaje
  - Proceso al que se le envía.

Detección. Algoritmo centralizado.

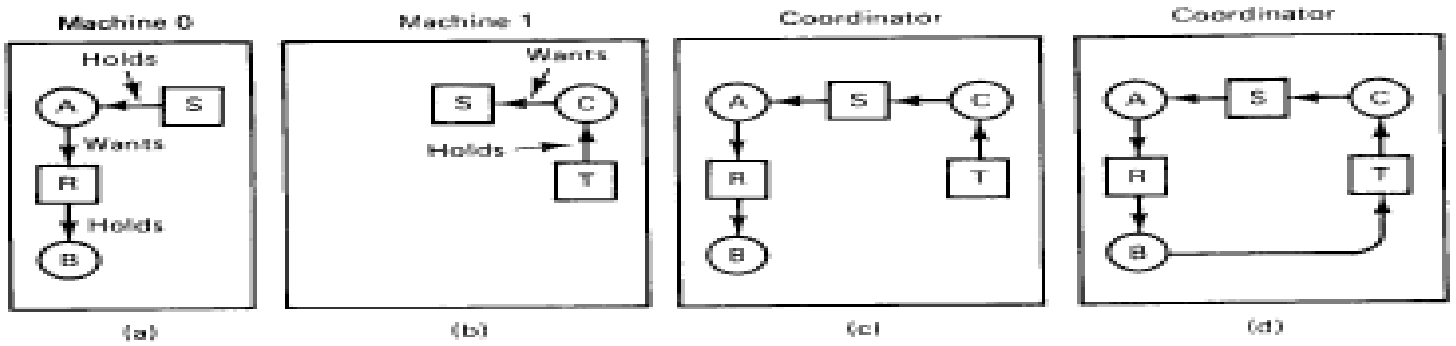


Fig. 3-23. (a) Initial resource graph for machine 0. (b) Initial resource graph for machine 1. (c) The coordinator's view of the world. (d) The situation after the delayed message.

Problema. B libera R y pide T.
La máquina 0 envía el mensaje y la máquina 1 envía otro al coordinador para decir que B espera el recurso T, pero el mensaje de la máquina 1 llega antes que el de la máquina 0 (d).

Detección. Algoritmo distribuido.

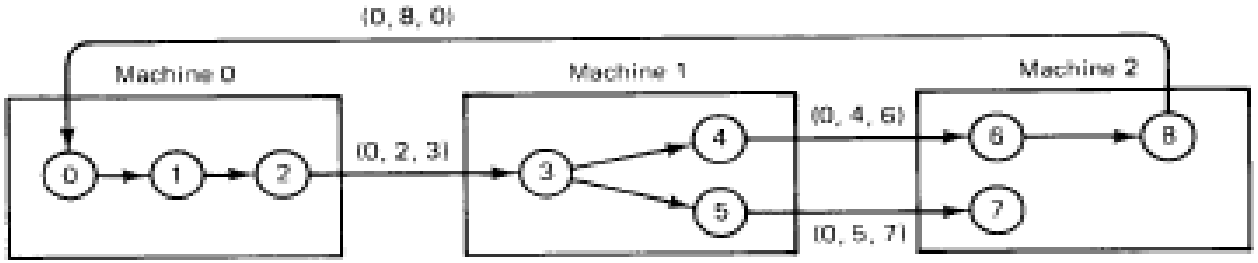


Fig. 3-24. The Chandy-Misra-Haas distributed deadlock detection algorithm.

- El mensaje inicial de 0 a 1, contiene (0,0,1). Al llegar a 1 el mensaje se actualiza, conservando el primer campo y reemplazando el segundo por su número de proceso y el tercero por el proceso al cual espera.
- El mensaje (0,2,3) es un mensaje remoto.
- Si el mensaje recorre todo el camino y regresa al emisor original (primer campo), entonces hay un ciclo.

Bloqueos. Prevención.

- Consiste en garantizar que nunca se van a producir interbloqueos, por ejemplo permitiendo que los procesos solo conserven un recurso por vez.
- Otro método consiste en ordenar los recursos y exigir a los procesos que los adquieran en orden creciente.

Bloqueos. Prevención.

- Otra posibilidad es asociar a cada transacción una marca de tiempo global al momento de su inicio.
- Idea principal. Cuando un proceso está a punto de bloquearse en espera de un recurso que viene siendo usado por otro, se verifica cual de ellos tiene la marca de tiempo mayor (es más joven).
- Hay algoritmos que permiten la espera solo si el que espera tiene marca de tiempo menor (más antiguo) y otros que lo hacen con la marca de tiempo mayor (más joven).
- Por lo general se da prioridad a los más antiguos porque se ha invertido más tiempo en ellos y probablemente conserven más recursos.

Bloqueos. Prevención. Espera - Muerte

- Transacción antigua espera por recursos en transacción nueva.
- Una transacción nueva aborta si espera por recursos existentes en una antigua.

Fig. 3-25. The wait-die deadlock prevention algorithm.

Bloqueos. Prevención. Herida-Espera

- También llamado Derecho de Propiedad.
- Una transacción antigua hace abortar a una nueva si esta última le hace esperar.
- Transacción nueva espera por recursos existentes en una antigua.

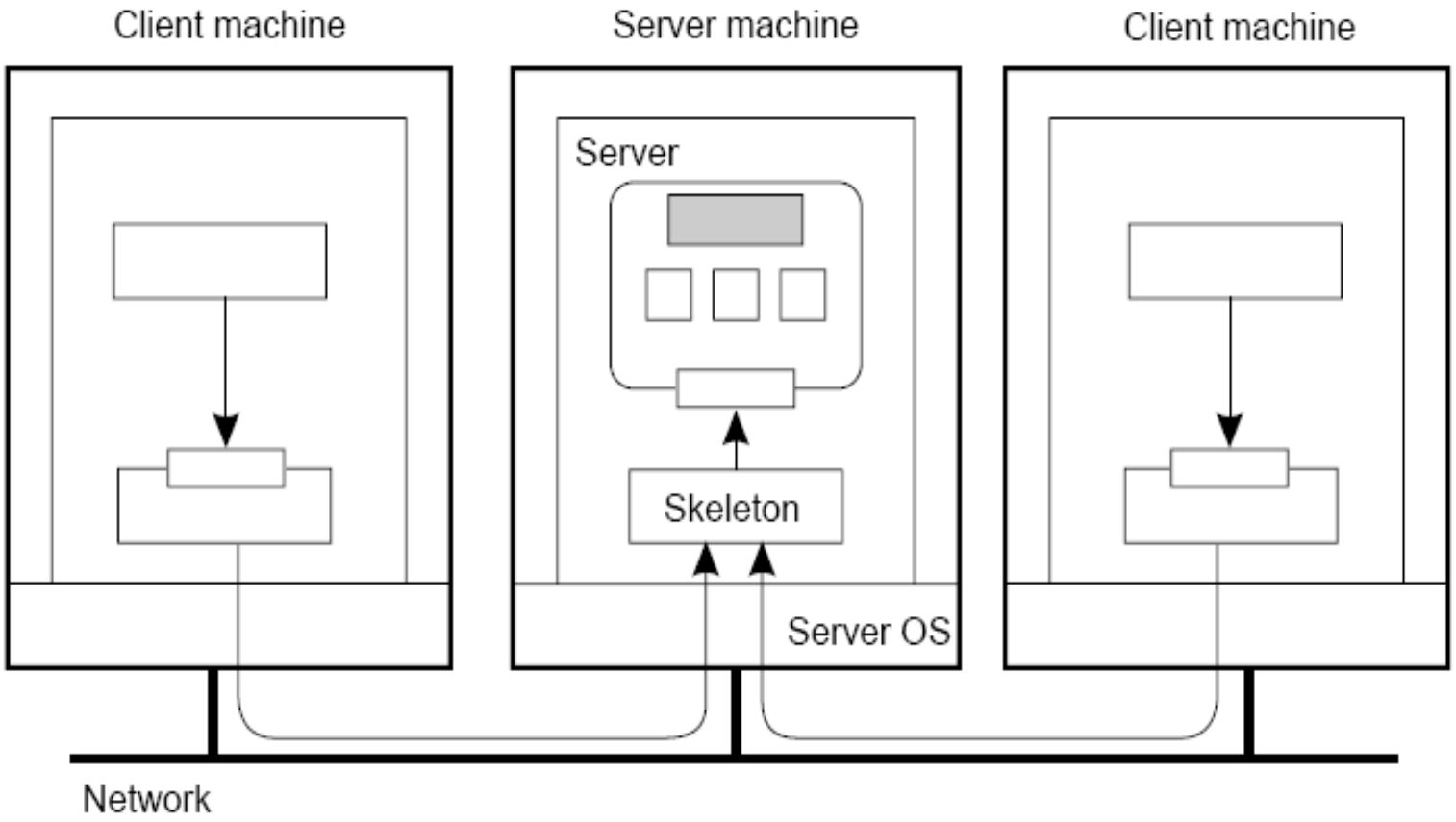
Fig. 3-26. The wound-wait deadlock prevention algorithm.

Preempts. Derecho de propiedad.

Replicación

- Útil para proveer:
  - buena performance
  - alta disponibilidad (ej. caída de una réplica)
  - tolerancia a fallos (mejor protección contra corrupción de datos)
- Problema: mantener la **consistencia** cuando se actualizan múltiples réplicas y a la vez mantener tiempos de respuesta razonables.

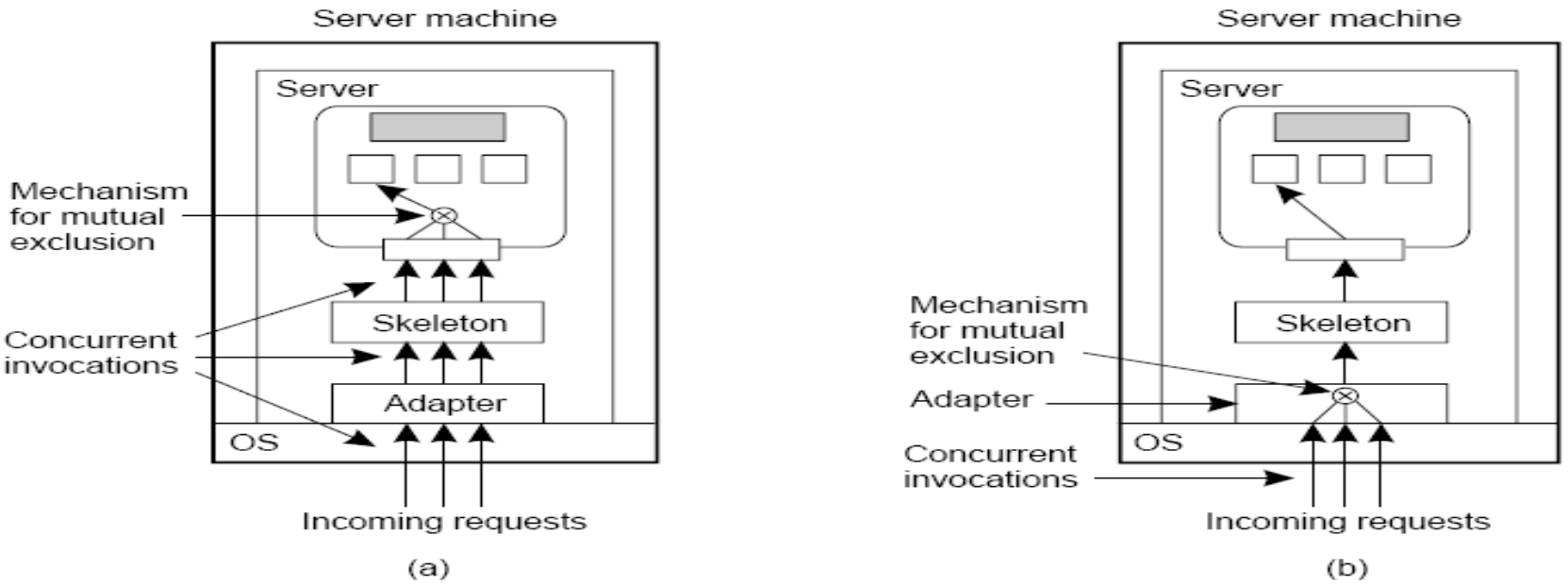
Protección de objetos ante accesos simultáneos.



Protección de objetos ante accesos simultáneos.

- Existen 2 soluciones, según Briot, 1998
  - El propio objeto maneja invocaciones concurrentes.
    - Ejemplo: objeto java *syncronized* permitirá un solo acceso.
  - El objeto está totalmente desprotegido, pero el servidor en el cual reside se encarga del control de concurrencia.

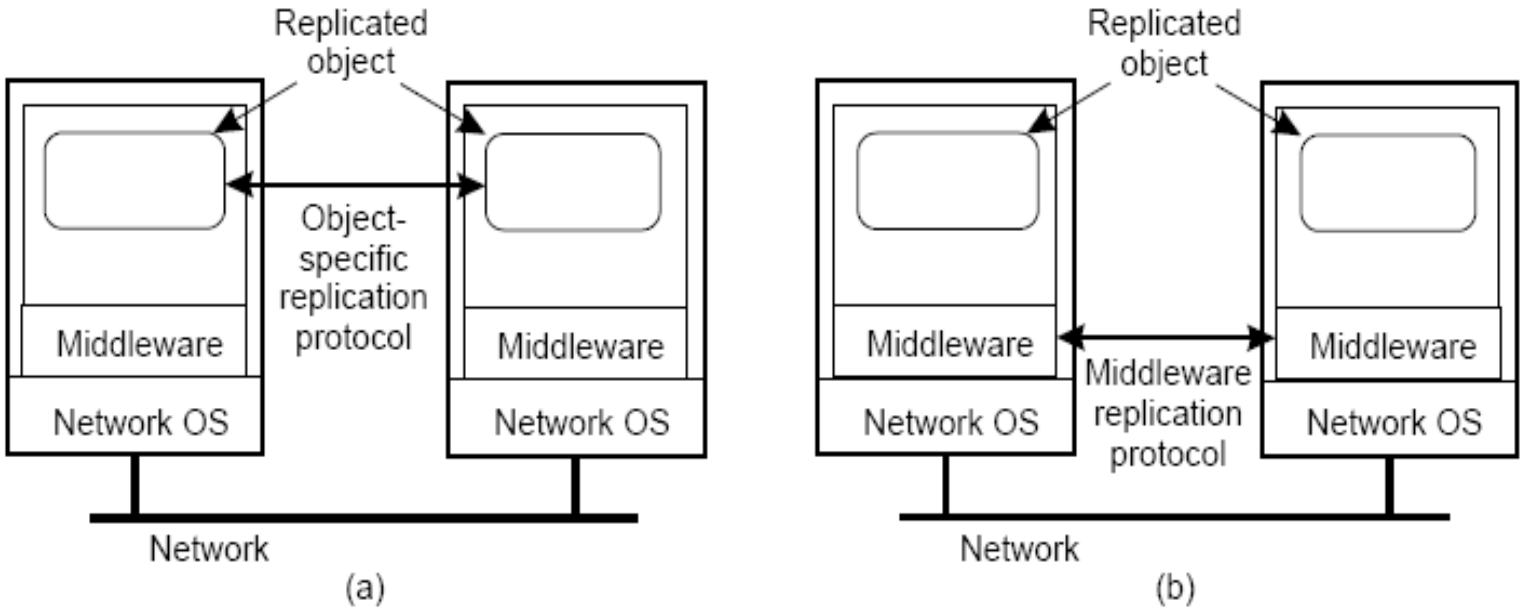
Protección de objetos ante accesos simultáneos.



a) Objeto remoto capaz de manejar invocaciones concurrentes por si mismo.

b) Objeto remoto para el cual se requiere un adaptador de objetos para manejar las invocaciones concurrentes

Control para la replicación de objetos.



a) SD para manejo propio de la replicación.

b) SD responsable de la administración de las réplicas.

Replicación como técnica de escalabilidad.

- La **replicación** y el **caching** son técnicas utilizadas para mejorar la performance.
- El hecho de ubicar copias de datos u objetos cercanos a los procesos que los usan mejora el rendimiento por la reducción del tiempo de acceso.
- Problemas:
  - Mantener las copias actualizadas consume más ancho de banda si las réplicas están distribuidas.
  - Mantener múltiples copias consistentes puede ser objeto de problemas de escalabilidad, máxime en un contexto de consistencia estricta.
- Dilema: Por un lado la replicación tiende a resolver el problema de la escalabilidad al aumentar el rendimiento pero por otro, mantener consistentes las copias es muy difícil.

Consistencia

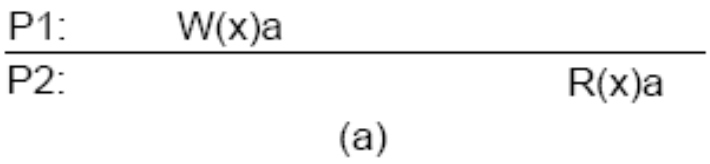
- Operaciones conflictivas. Ocurre cuando se realizan dos o más operaciones sobre el mismo item de datos y al menos una de ellas es un write.
  - Se distingue conflicto:
    - read-write (una sola operación es un write)
    - write-write (más de una operación es un write).
  - Para ser consistente un data store debe ver todas las operaciones write-write en el mismo orden por todos los clientes.
- Orden:
  - Parcial. Todas las operaciones ejecutadas en una misma réplica ocurren en un orden particular
  - Total. Si se piensa en múltiples réplicas en una línea de tiempo se habla de orden total.

Modelo de consistencia.

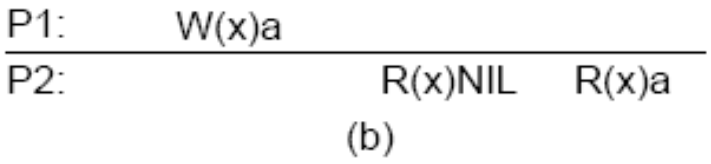
- Es un contrato entre los procesos y el almacenamiento de datos. Refiere a que si el proceso acuerda obedecer ciertas reglas, el almacenamiento promete trabajar correctamente.
- Normalmente un proceso que realiza una operación de lectura espera que esa operación devuelva un valor que refleje el resultado de la última operación de escritura sobre el dato.
- En ausencia de un reloj global, es difícil precisar cual fue el último write, por lo que aparecen los modelos de consistencia.

Centrado en datos. Consistencia estricta.

- Es el modelo de consistencia más restrictivo.
- Definición: Cualquier lectura sobre un item de dato x retorna un valor correspondiente con la más reciente escritura sobre x
- Asume existencia de tiempo global.



(a)



(b)

Dos procesos operando sobre el mismo item de datos.

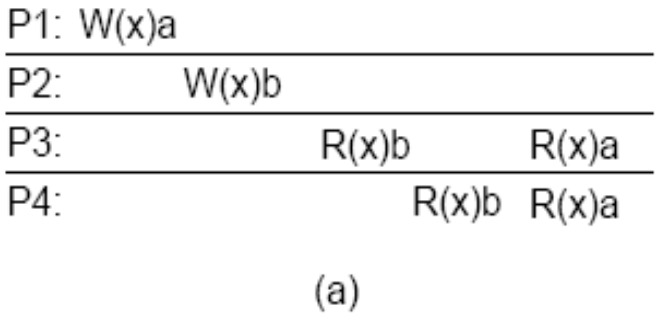
a) Almacenamiento estrictamente consistente.

b) Almacenamiento no estrictamente consistente.

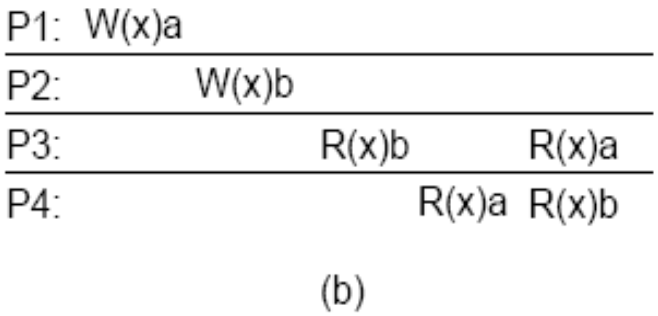
Consistencia secuencial.

- Forma ligeramente más débil que la consistencia estricta.
- Definición (Lamport, 1979): El resultado de una ejecución es lo mismo que 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.
- No se habla de tiempo. La definición implica que no importa la relación read-write si esta es vista de la misma manera por todos los procesos

Consistencia secuencial.



(a)



(b)

a) Dato almacenado secuencialmente consistente.

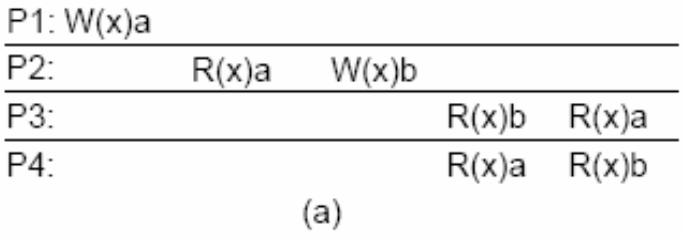
b) Dato almacenado no secuencialmente consistente.

Consistencia causal.

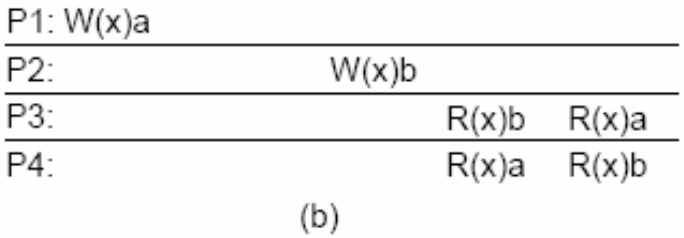
- Debilitamiento de la consistencia secuencial (Hutto y Ahamad, 1990). **Se hace una diferenciación entre eventos que están potencialmente relacionados en forma causal y aquellos que no.** Las operaciones que no están causalmente relacionadas se dicen **concurrentes**.
- La condición a cumplir para que 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.

- En (a) W(x)b depende de W(x)a ya que 2 puede ser un resultado de un cálculo que implique el valor leído por R(x)a, por lo que (a) es incorrecta.
- En (b) se ha eliminado la lectura R(x)a por lo que las dos escrituras son concurrentes y la figura es correcta.



(a)



(b)

Consistencia débil.

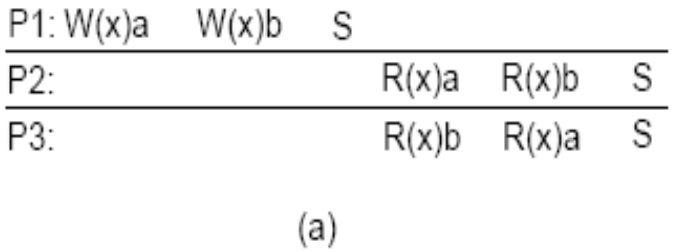
- Muchas veces no es necesario replicar instantáneamente todos los cambios.
  - Ejemplo: proceso dentro de una región crítica, donde actualiza variables.

- Variable de sincronización. Las operaciones en ella se utilizan para sincronizar almacenamiento. Cuando termina una sincronización, todas las escrituras realizadas se propagan hacia fuera y todas las escrituras realizadas en otras máquinas son traídas a esa máquina.

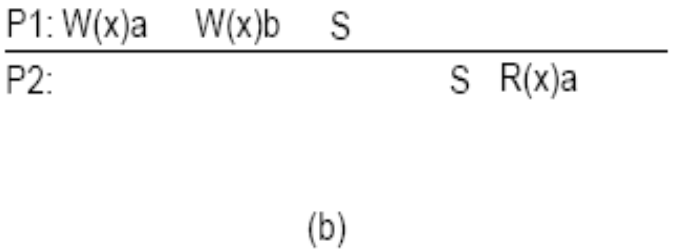
Consistencia débil

- Tiene 3 propiedades:
  - Los accesos a las variables de sincronización son secuencialmente consistentes.
  - No se permite el acceso a una variable de sincronización hasta que las escrituras anteriores no hayan terminado en todas partes.
  - No se permite el acceso a los datos (lectura y escritura) hasta realizar todos los accesos anteriores a las variables de sincronización.

Consistencia débil.



(a)



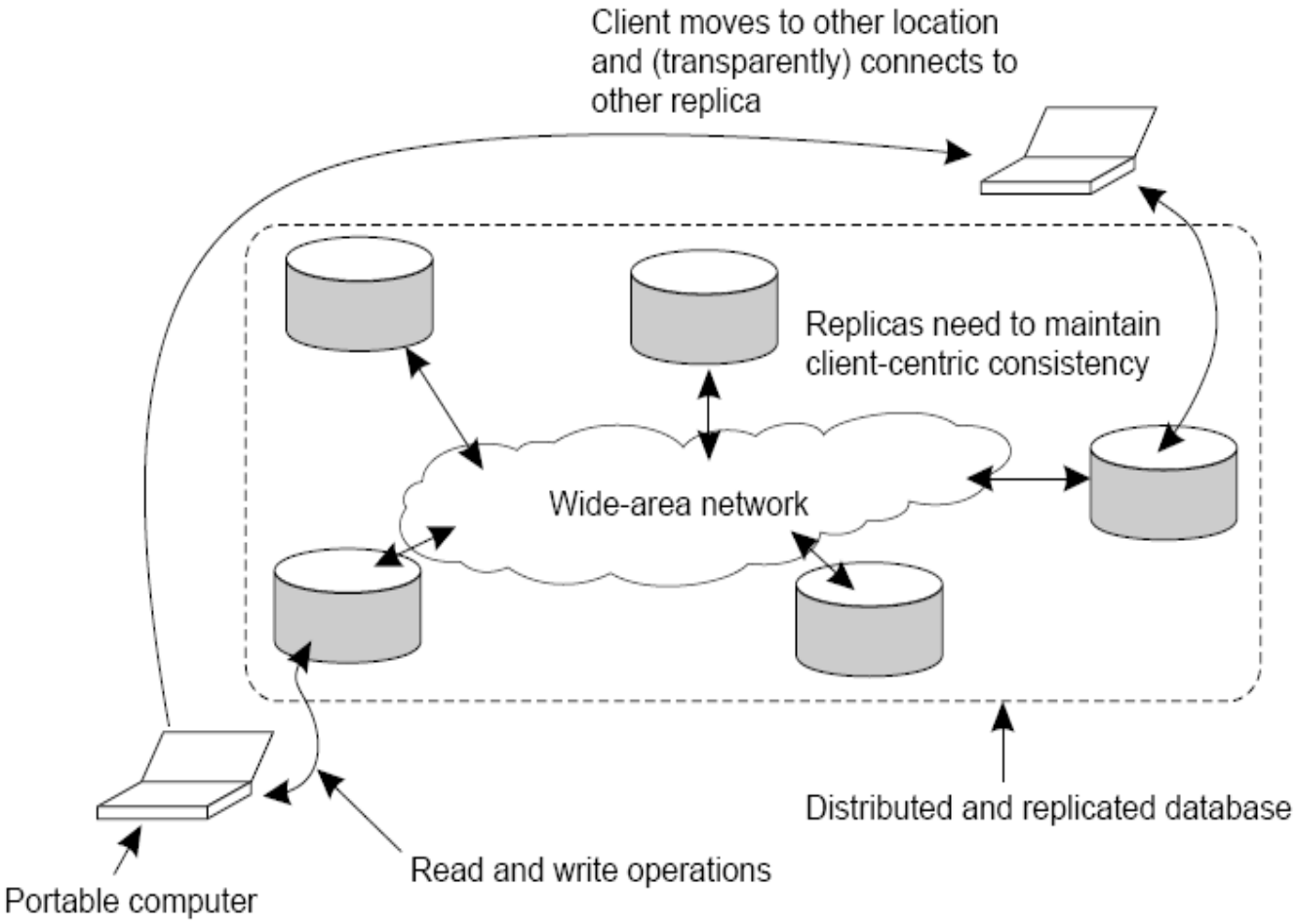
(b)

a) Válida ya que como en P2 y en P3 no ha existido sincronización, nada garantiza lo que verán.
b) No válida para consistencia débil, ya que la memoria se ha actualizado y el valor que se debe leer es b.

Modelos de consistencia centrados en el cliente.

- Casi todos los modelos **centrados en datos** asumen que el número de lecturas es aproximadamente igual al número de escrituras y que frecuentemente ocurren escrituras concurrentes.
- Los modelos **centrados en el cliente**, asumen que los clientes habitualmente realizan mayor cantidad de lecturas que escrituras y que hay pocas escrituras concurrentes. Asumen también clientes móviles.
- Son relativamente fáciles de implementar.
- Write set (WS) = contiene la historia de writes de un item de datos particular en una determinada réplica.

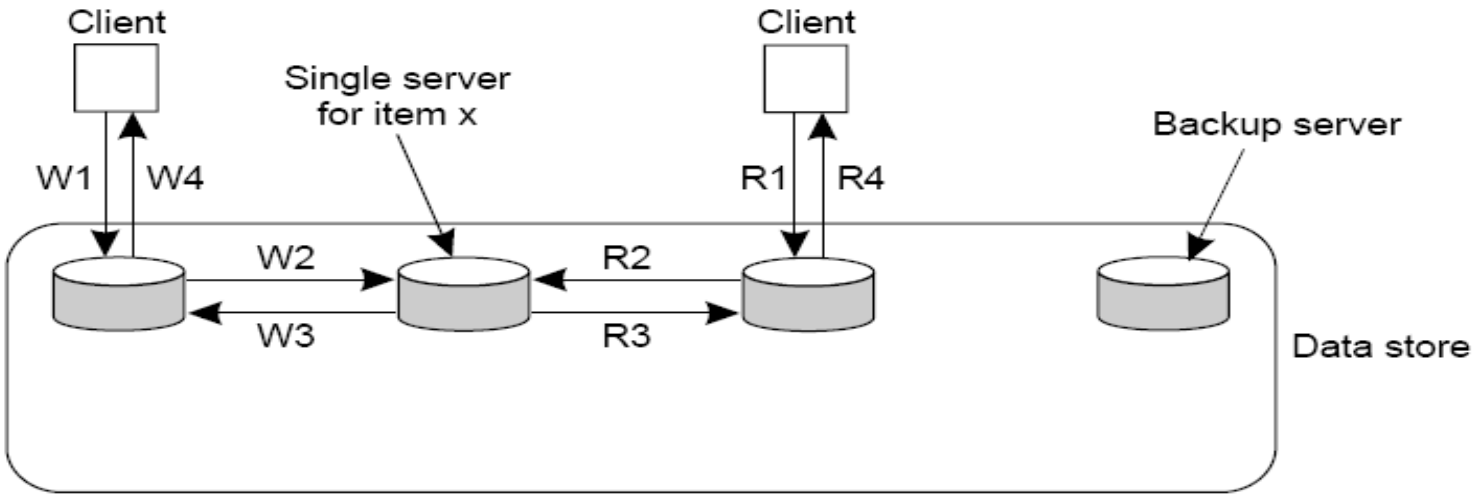
Clientes móviles.



Protocolos de Consistencia.

- Implementación del modelo de consistencia.
- Administra el orden de las operaciones.
- Hay 2 clases de protocolos de consistencia centrados en datos:
  - **Basados en el primario**. Requiere que cada item de datos tenga una copia primaria con todas las escrituras realizadas. Pueden a su vez clasificarse en:
    - Remote-write – Posible ejecución de writes en réplica remota.
    - Local-write – Los write se realizan siempre en réplica local.
  - **Basados en las réplicas**. Requiere que todas las escrituras sean realizadas simultáneamente.

Basados en el primario – Remote write

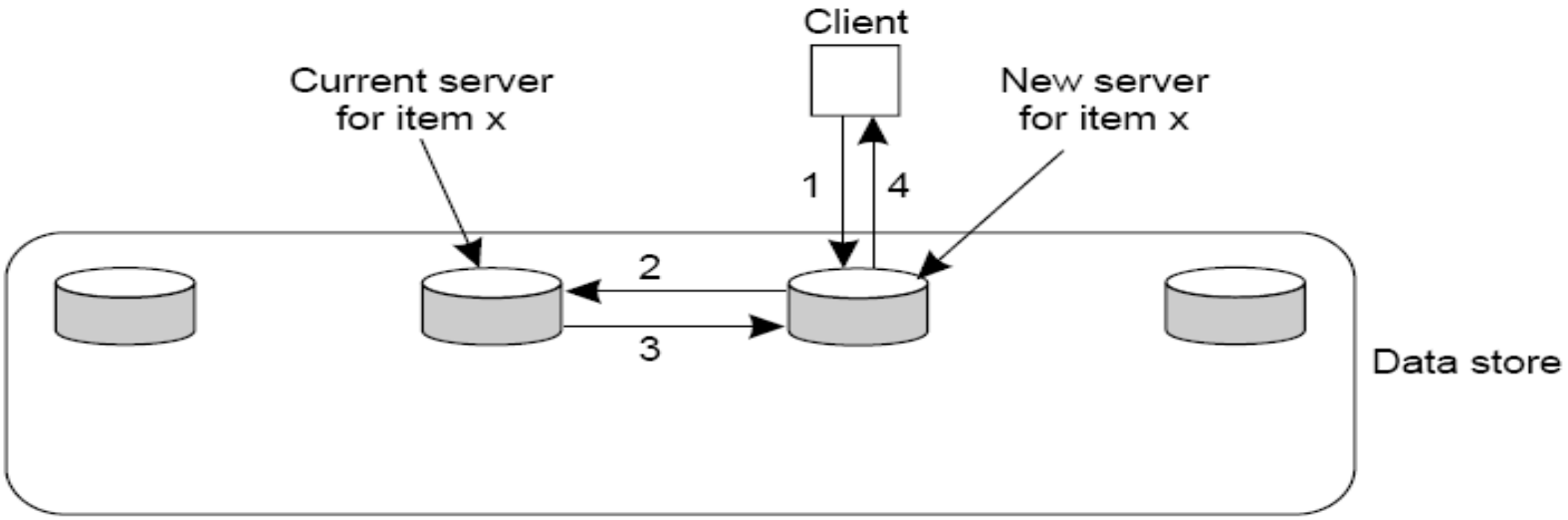


W1. Write request
W2. Forward request to server for x
W3. Acknowledge write completed
W4. Acknowledge write completed

R1. Read request
R2. Forward request to server for x
R3. Return response
R4. Return response

Fig. 6-27. Primary-based remote-write protocol with a fixed server to which all read and write operations are forwarded.

Basados en el primario – Local write



1. Read or write request
2. Forward request to current server for x
3. Move item x to client's server
4. Return result of operation on client's server

Fig. 6-29. Primary-based local-write protocol in which a single copy is migrated between processes.

Basados en el primario – Local write

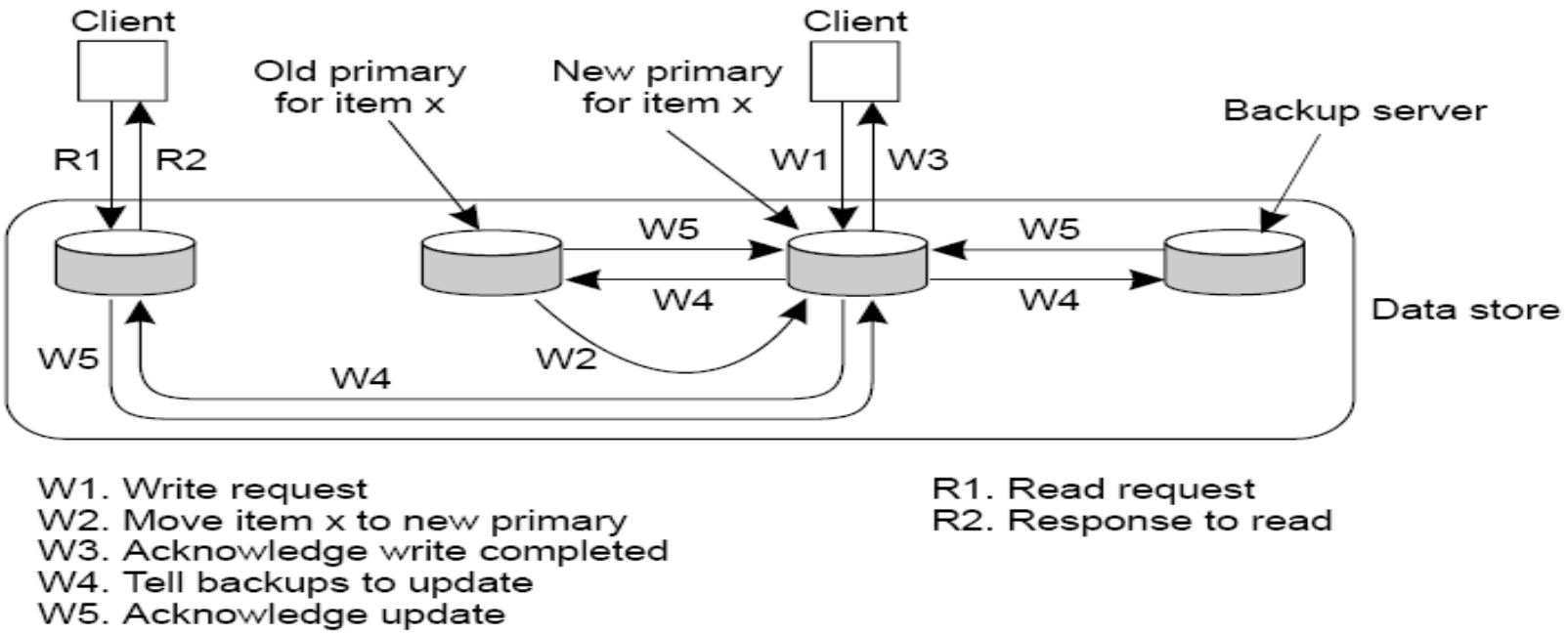


Fig. 6-30. Primary-backup protocol in which the primary migrates to the process wanting to perform an update.

Basados en las réplicas – Replicación activa

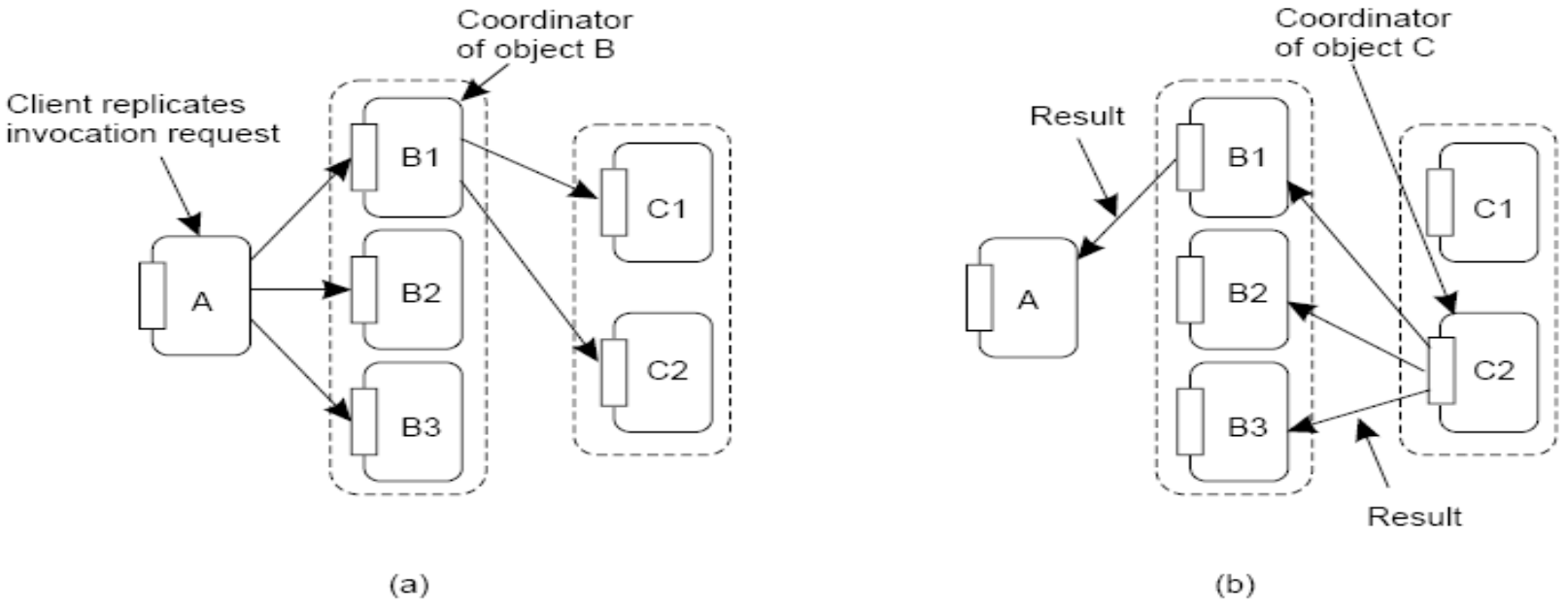


Fig. 6-32. (a) Forwarding an invocation request from a replicated object to another replicated object. (b) Returning a reply from one replicated object to another.

Protocolo de servidor único.

- Implementa consistencia secuencial centralizando los datos básicos y los que deben ser replicados. Todas las operaciones de write en un item de datos se derivan al servidor que mantiene la copia primaria. Las lecturas también se envían a ese servidor.
- Problema: poca escalabilidad.
- Protocolo Primary-backup. Variante que permite realizar las lecturas de las réplicas. La operación de write bloquea hasta que se terminen las grabaciones en todas las réplicas.

Protocolo Primary-Backup

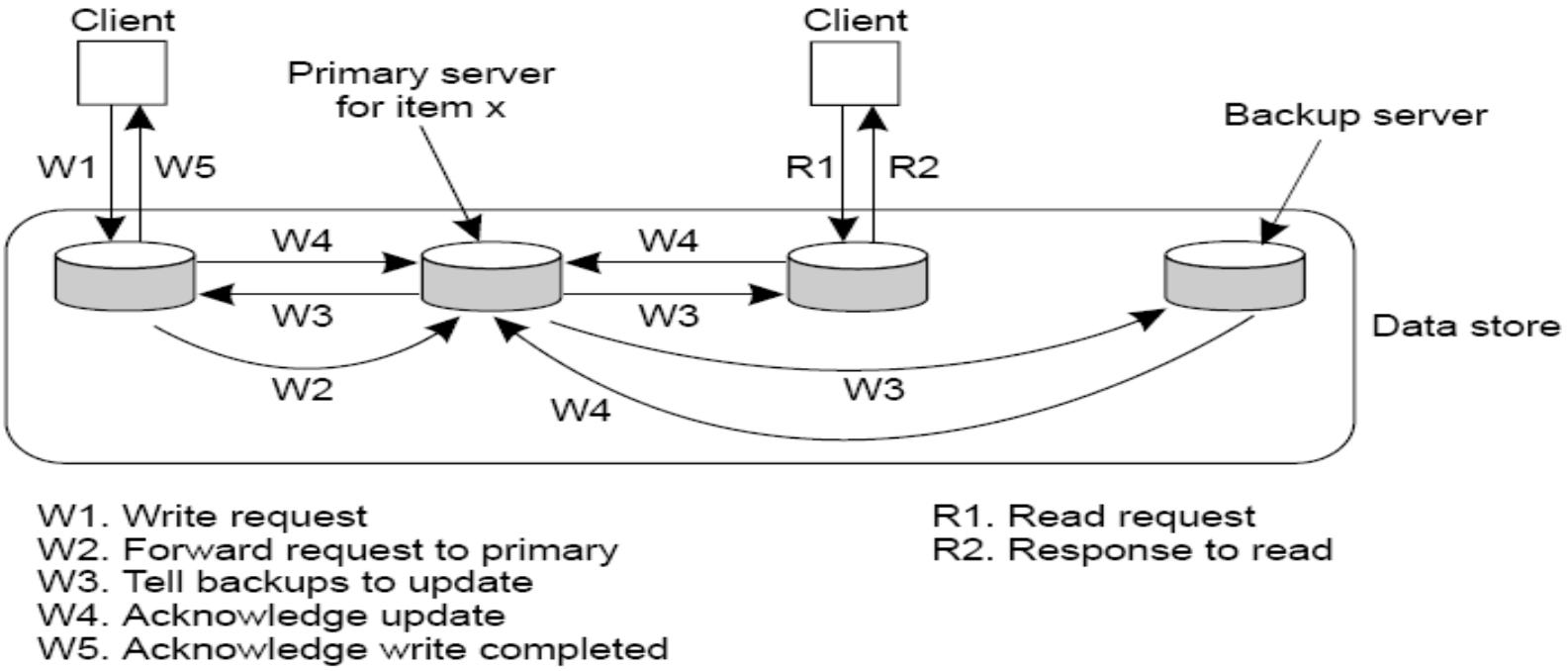


Fig. 6-28. The principle of a primary-backup protocol.