Dpto. Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur

Elementos de Bases de Datos

Segundo Cuatrimestre 2015

Clase 17:

Fallos en un SMBD

Sistema de Recuperación

Mg. María Mercedes Vitturini
[mvitturi@uns.edu.ar]

DBMS y Fallos

Cualquier sistema está sujeto a fallos: un fallo eventualmente puede ocasionar pérdidas e inconsistencia de información.

"*La ocurrencia de un fallo se puede presentar en cualquier etapa de la vida de las distintas transacciones que gestiona el SMBD.*"

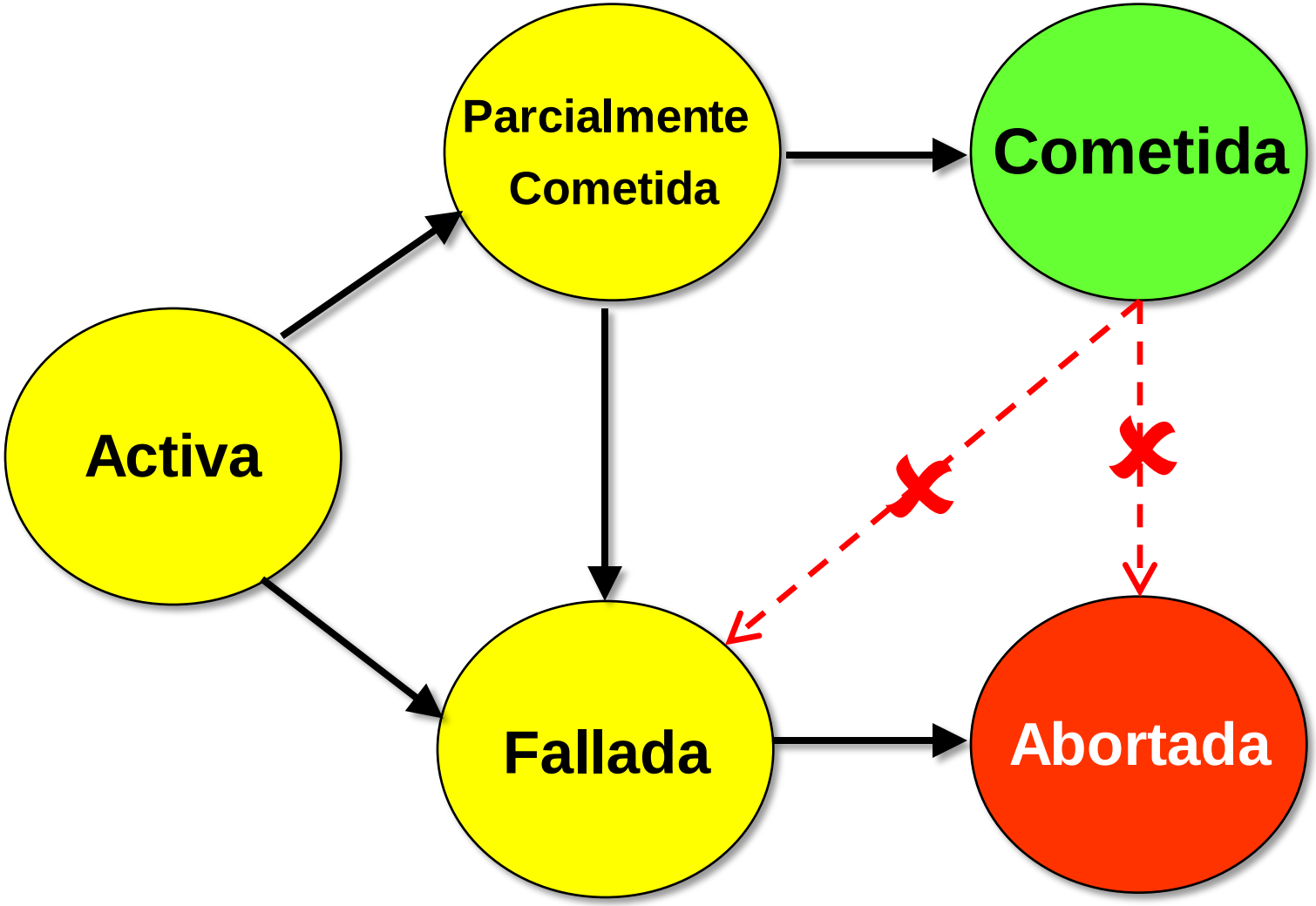
- Ante el fallo, el DBMS se debe *'recuperar'* y dejar sus bases de datos en el estado consistente anterior al fallo.

Ejemplo:

Sea T una transacción que realiza transferencias de fondo de la cuenta A a la cuenta B. El fallo podría ocurrir después de descontar 50$ de A. Los valores iniciales: A=1000, B=2000. El fallo ocurre después del débito, pero antes de acreditarlos en B. (A=950, B=2000)

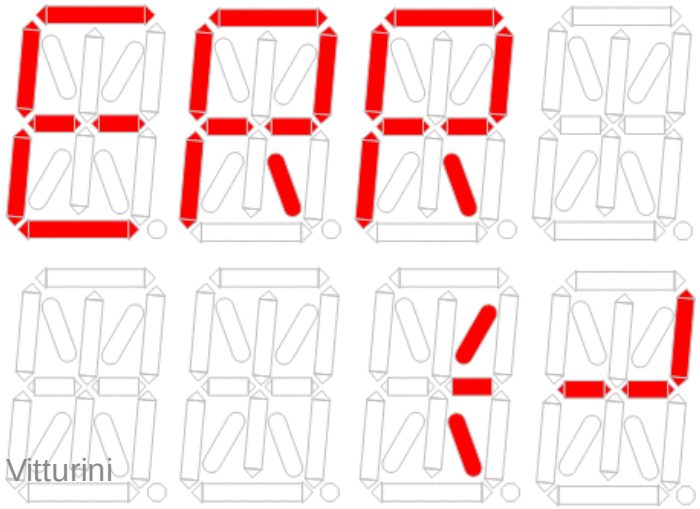
- El DBMS debe recuperarse al estado consistente anterior, antes de iniciar la transferencia (A=1000, B=2000).

Repaso – Estados de una Transacción



Tipos de Fallos

- Para un **SMBD centralizado concurrente** se identifican los siguientes tipos de fallos:
  - Fallo de Transacción.
  - Caída de Sistema.
  - Fallo con pérdida de disco



Fallo de Transacción

FALLO DE TRANSACCIÓN – cualquier error que cause que una transacción deba retroceder:

- **Error lógico**: la transacción no puede continuar su ejecución a causa de alguna condición interna.
- **Error bajo el entorno del sistema**: el sistema debe terminar con la ejecución de una transacción activa debido a alguna condición de error (deadlock, problemas de estampillas, etc.)
  - La transacción vuelve a ejecutarse más tarde.

Alcance del Fallo de Transacción – la transacción activa que falla y potencialmente las transacciones dependientes activas por el efecto 'fallo en cascada o cadena'. Otras transacciones activas no afectadas prosiguen su ejecución normal.

Caída del Sistema

Caída del sistema (crash) – fallos de hardware o software que causan la falla general y caída del sistema.

- Características:
  - Causan pérdida de información en memoria volátil (memoria principal y cache).
  - El contenido de la información en la memoria no volátil (disco) permanece intacto.
  - El DBMS posee numerosos chequeos de consistencia para prevenir la corrupción de los datos.

Alcance del fallo Caída de Sistema – afecta a *todas* las transacciones activas al momento del error y potencialmente *también* a transacciones recientemente cometidas.

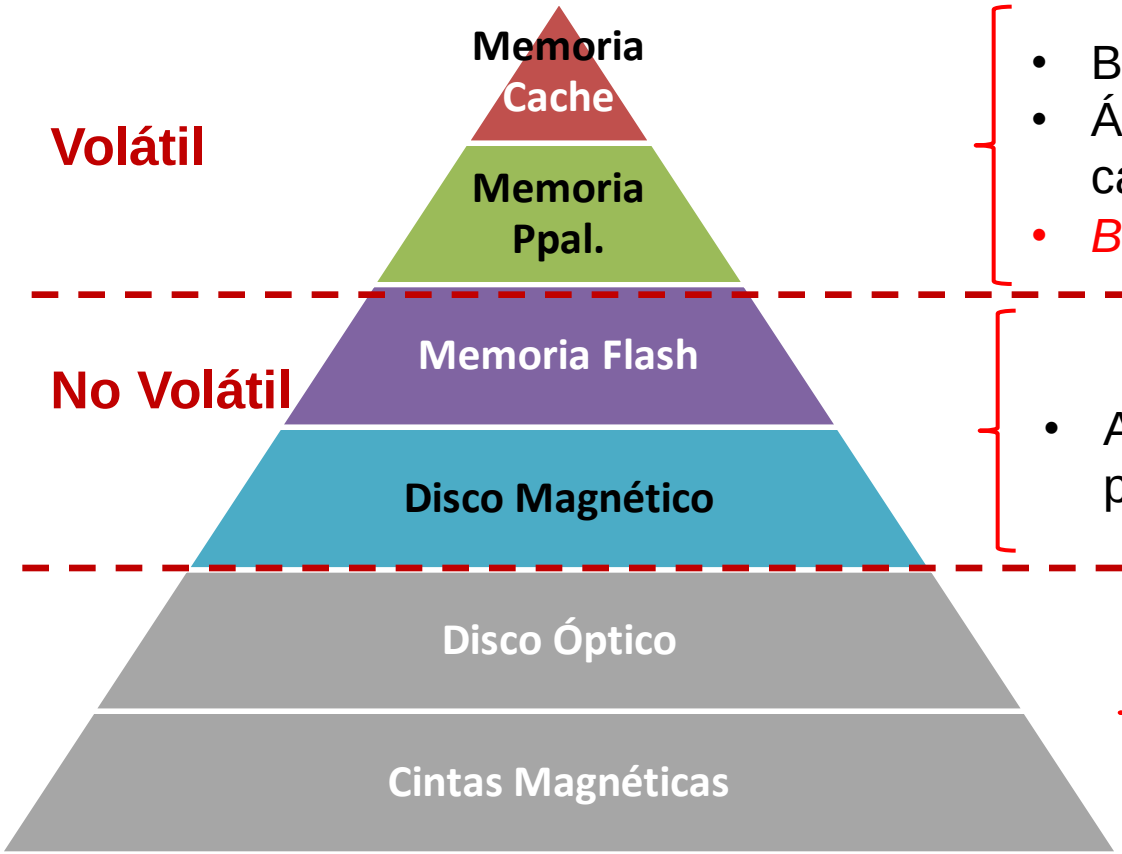
Fallo de Disco

Fallo de disco – i.e. daño físico en el medio de almacenamiento masivo.

- Se asume que el error es detectable (checksum).
- Se requiere de copias de datos de backup en otros discos u otros medios de almacenamiento para recuperar el fallo.
- La recuperación combina los pasos de recuperar un backup con recuperación por fallo del tipo caída del sistema.

backup + recuperación por caída de sistema

Repaso Estructura de Almacenamiento



- Buffer de datos compartido.
- Área de trabajo propia de cada transacción
- *Buffer de Logs*

- Archivos de dato persistente

- Backup

Estructura de Almacenamiento

Almacenamiento volátil

- Incluye memoria principal (MP) y memoria cache (MC).
- No sobrevive a las caídas de sistemas.

Almacenamiento no volátil

- Ejemplo: discos.
- No representa por sí solo almacenamiento estable.

Almacenamiento estable

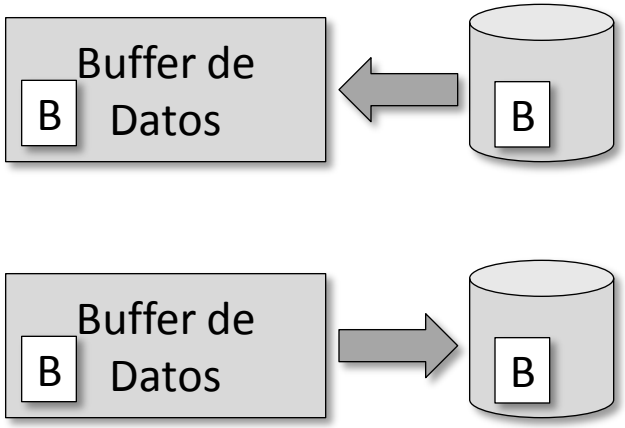
- Organización del almacenamiento que asegura *'nunca'* perder información.
- Sobrevive a todos los fallos.
- Se basa en replicas en distintos medios de almacenamiento con modos de fallo independiente, geográficamente separados y actualización de la información controlada.

Memoria Estable

- La información que reside en memoria estable **nunca** se pierde.
- Mecanismo para alcanzarlo: repetir la información (de manera controlada) en varios medios de almacenamiento.
  - En el caso de discos espejados, ambos bloques están físicamente en la misma locación.
  - En el caso de copias remotas, uno de los bloques es local y el otro está en un sitio remoto.

Repaso - Acceso a los datos

- La base de datos reside en archivos en memoria estable y se organiza en unidades denominadas **bloques físicos.**
- Los datos se trasladan desde/hacia los *bloques físicos* a **bloques de registro intermedio o buffer de MP**.
- Las transferencias disco/MP se inician a través de las operaciones:
  - **INPUT(B):** transfiere el bloque físico B a memoria principal.
  - **OUTPUT(B):** transfiere el bloque de registro intermedio B (buffer) y sustituye el bloque físico apropiado.



Acceso a los datos

- Las operaciones de entrada y salida desde el disco a la MP se hacen en *unidad de bloque.*
- Cada transacción posee su *área de trabajo* (work-area) en MP donde mantiene las variables locales con los valores de los ítems de datos accedidos y actualizados por ella.
- Las **transacciones** transfieren ítems de datos entre el **buffer de datos compartido** y su propia **área de trabajo** usando las operaciones: **READ (X)** y **WRITE(X).**

Transferencias de Datos

READ(X, x_i) asigna el valor del elemento de dato X a la variable local x_i.

Esta operación se ejecuta como sigue:

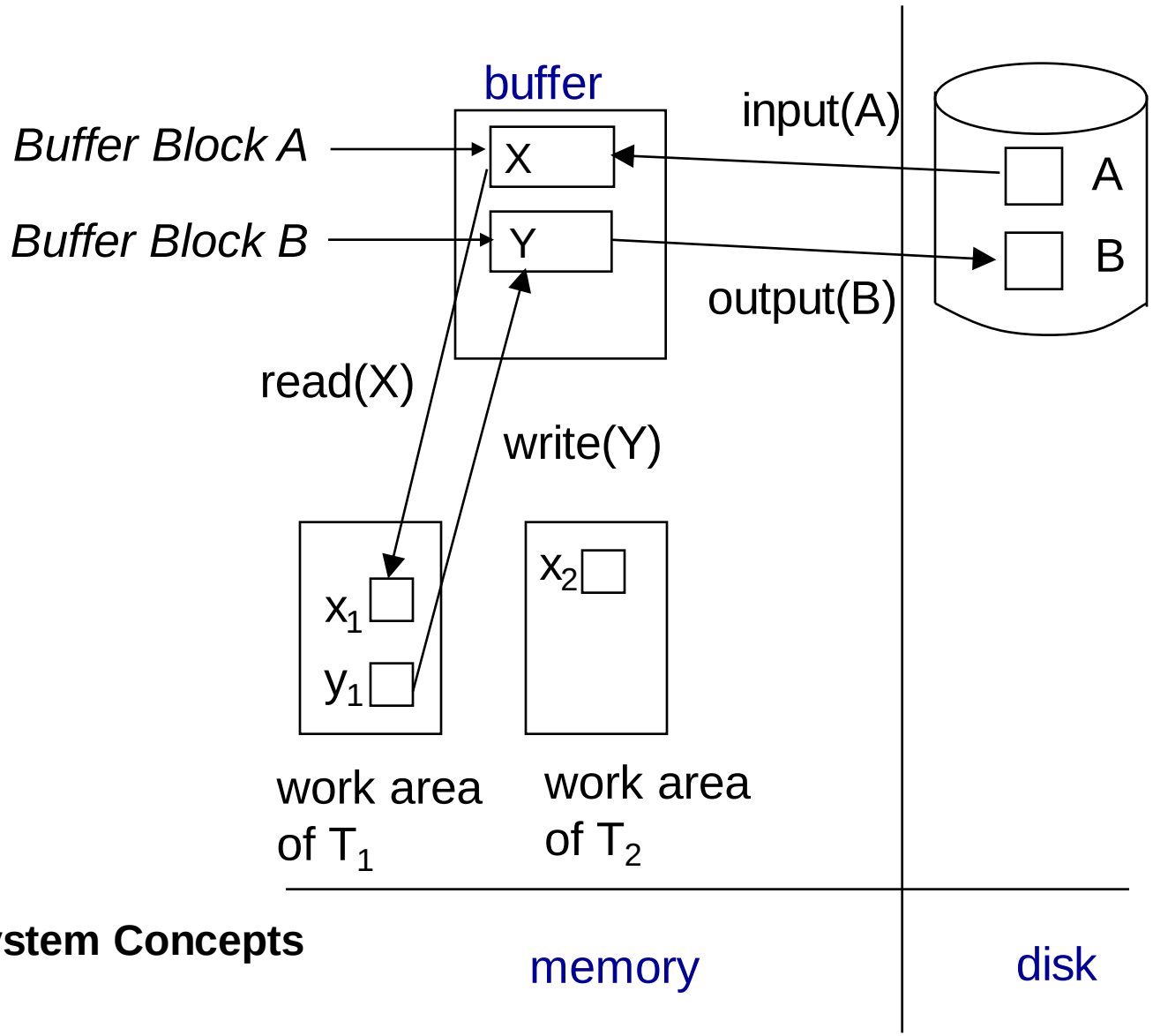
- Si el bloque B_X en el que reside X **no está en memoria principal**, entonces se ejecuta **INPUT(B_X)**.
- Se asigna a x_i el valor de X del bloque en el registro intermedio.

Transferencias de Datos

WRITE(X, x_i): asigna el valor de la variable local x_i al elemento de información X.

Esta operación se ejecuta como sigue:

- Si el bloque B_X en el que reside X **no está en memoria principal**, entonces se ejecuta **INPUT(B_X)**.
- Se asigna el valor de x_i a X en el bloque que contiene a X del buffer compartido.
- En algún momento se realiza el **OUTPUT(B_X).**



Fuente: Database System Concepts
A. Silberschatz

Write (X, x) – Output(X)

- El Output(B) de bloques modificados a almacenamiento estable tiene lugar en algún momento posterior actualizar el bloque en el buffer de datos.
- El orden en el que se realiza el output puede ser diferente del orden en el que se modificaron.
- La operación Output(X) no se siempre ejecuta inmediatamente después de write (X, xi).
  - Por ejemplo, el bloque en el que reside X contiene otros elementos de información a los que se accede todavía.

- Ante un fallo de sistema después de ejecutar write(X,xi) y antes de ejecutar Output(X), existe riesgo de perder el nuevo valor para X.

Sistema de Recuperación

Los sistemas están sujetos a fallos que pueden ocasionar estados inconsistentes

Sistema de Recuperación

" A computer system, like any other device, is subject to failure from a variety of causes: disk crash, power outage, software error, a fire in the machine room, even sabotage. In any failure, *information may be lost*. Therefore, the database system must take actions in advance to ensure that the **atomicity** and **durability** properties of transactions, introduced in Chapter 14, are preserved. An integral part of a database system is a recovery scheme that can **restore the database to the consistent state that existed before the failure**. The recovery scheme must also provide high availability; that is, it must minimize the time for which the database is not usable after a failure."

DATABASE System Concepts – A. Silberschazt

Algoritmo de Recuperación (AR)

Los **algoritmos de recuperación (AR)** son procesos usados para asegurar *atomicidad* y *durabilidad* de las transacciones a pesar de los fallos, así como la *consistencia* de la BD.

- En un AR se identifican:
  - **Acciones preventivas** a tomar durante el procesamiento normal de las transacciones para asegurar que exista suficiente información para permitir la recuperación de fallos (**acciones *preventivas***).
  - **Acciones de recuperación** como consecuencia de un fallo para asegurar la consistencia, atomicidad y durabilidad de las transacciones (**acciones *paleativas***).

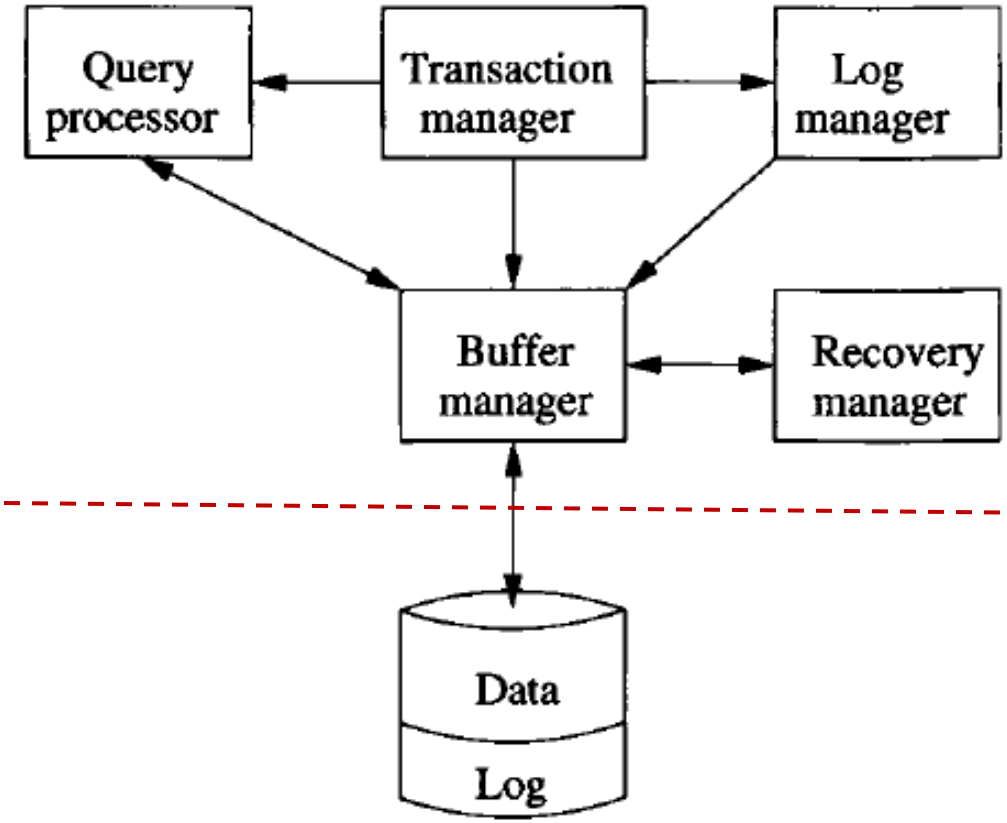
Acciones Preventivas

- Modificar directamente la base de datos antes de saber si la transacción cometerá puede conducir a estados inconsistentes. **Se necesita guardar en memoria estable información sobre las modificaciones realizadas.**

La **bitácora** (o registros *log*) se mantiene en memoria estable y mantiene una secuencia de registros con información sobre las actualizaciones realizadas por cada transacción.

- **Framework:** la bitácora y el buffer de datos **son compartidos por todas las transacciones del sistema**.
- **Framework inicial:** **cada uno de los registros de la bitácora** se graban en **memoria estable** y **las transacciones se ejecutan en serie**.

Esquema de Recuperación con Bitácora



RAM

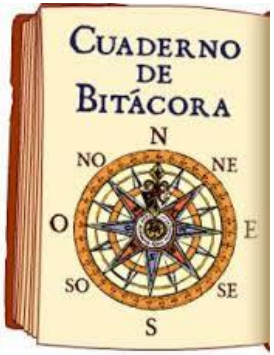
Memoria Estable

Figure 17.1: The log manager and transaction manager

Ref - DataBase System – The Complete Book – H. Molina, J. Ullman

Registros Bitácora

- Una estructura muy utilizada para registrar las modificaciones a la base de datos es a través del uso de **registros log** o **registros bitácora**.
- Los **registros log dejan pista de todas las actividades de actualización**.
- *Típicamente* un registro log mantiene la siguiente información:
  - Identificador de la transacción.
  - Identificador del dato modificado.
  - Valor viejo del dato, anterior a la modificación.
  - Valor nuevo del dato, posterior a la modificación.



Registros Bitácora

- Cuando la transacción T_i se inicia, se agrega en la bitácora el registro:

$$< T_i, \text{start} >$$

- Cada vez que la transacción T_i ejecuta write(X), se agrega en bitácora un registro:

$$< T_i, X, v_1, v_2 >$$

  - donde v_1 representa el valor viejo de X antes de la escritura,
  - v_2 el nuevo valor después de la misma,
  - T_i y X identifican la transacción y el dato modificado.

Registros de la Bitácora

- Cuando la transacción T_i finalizó con su última instrucción agrega en bitácora el registro:

< T_i, commit >

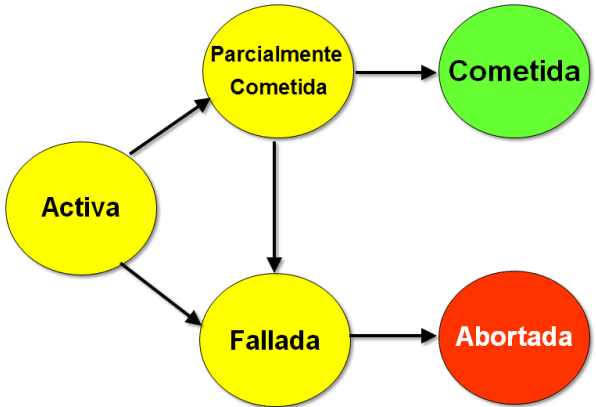
- Si la transacción T_i aborta agrega en bitácora el registro:

< T_i, abort >

Transacción cometida

- Una transacción se dice que está **cometida** si el registro **<T, commit>** de bitácora está **en memoria estable**.
  - **Todos los registros log anteriores de la transacción también deben haber sido almacenados en memoria estable previamente**.

- Dada una transacción en estado cometida, eventualmente, las escrituras realizadas por ella podrían haber sido actualizada en el buffer de datos y el output estar pendiente.



Operaciones Redo y Undo

- Undo de un registro: $<T_i, X, V_1, V_2>$ escribe a X con el valor **anterior** V_1.
- Redo de un registro: $<T_i, X, V_1, V_2>$ escribe a X con el valor **nuevo** V_2.

Undo y Redo de Transacciones

- **undo**(T_i) restaura el valor todos los ítems de datos modificados por T_i a sus valores anteriores, recorriendo la bitácora en sentido hacia atrás desde el último registro para T_i
- **redo**(T_i) escribe el valor de todos los ítems de datos actualizados por T_i con el nuevo valor, recorriendo la bitácora hacia adelante desde el primer registro de T_i

Recuperación con Bitácora

IMPORTANTE:

- Los registros de la bitácora deben siempre ser escritos antes que la modificación en la base de datos.
  - Asumimos además que la operación output de la bitácora es inmediata.
  - El ouput de los registros de los datos puede ocurrir en cualquier momento posterior.

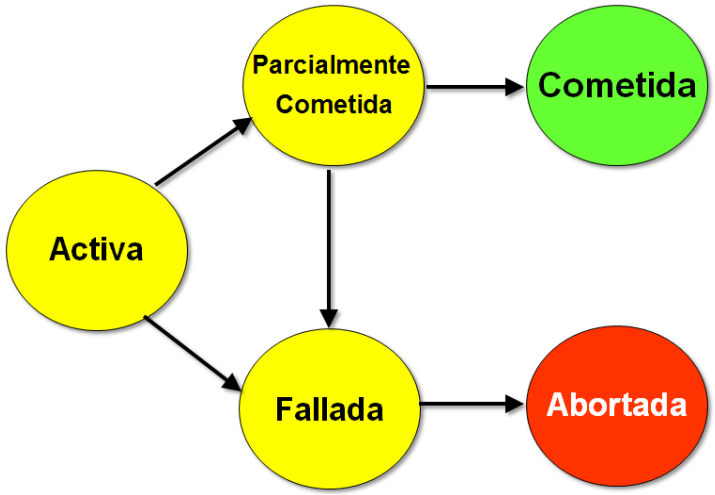
Framework: la bitácora y el buffer de datos **son compartidos por todas las transacciones del sistema.**

Framework inicial: cada uno de los registros de la bitácora se graban en **memoria estable** y **las transacciones se ejecutan en serie.**

Esquemas de Modificación

Existen dos esquemas de modificación

- Modificación Inmediata
- Modificación Diferida



Esquema modificación inmediata

- Bajo el esquema de **modificación inmediata** las modificaciones a la base de datos se realizan en cualquier momento *aún* mientras la *transacción está activa* (modificaciones no cometidas).
  - En caso de que ocurra un fallo, es necesario **deshacer los cambios** hechos por una transacción no cometida.

- Bajo el esquema de **modificación diferida** las modificaciones a la base de datos se demoran hasta que la transacción este cometida.

Modificación inmediata: acciones preventivas

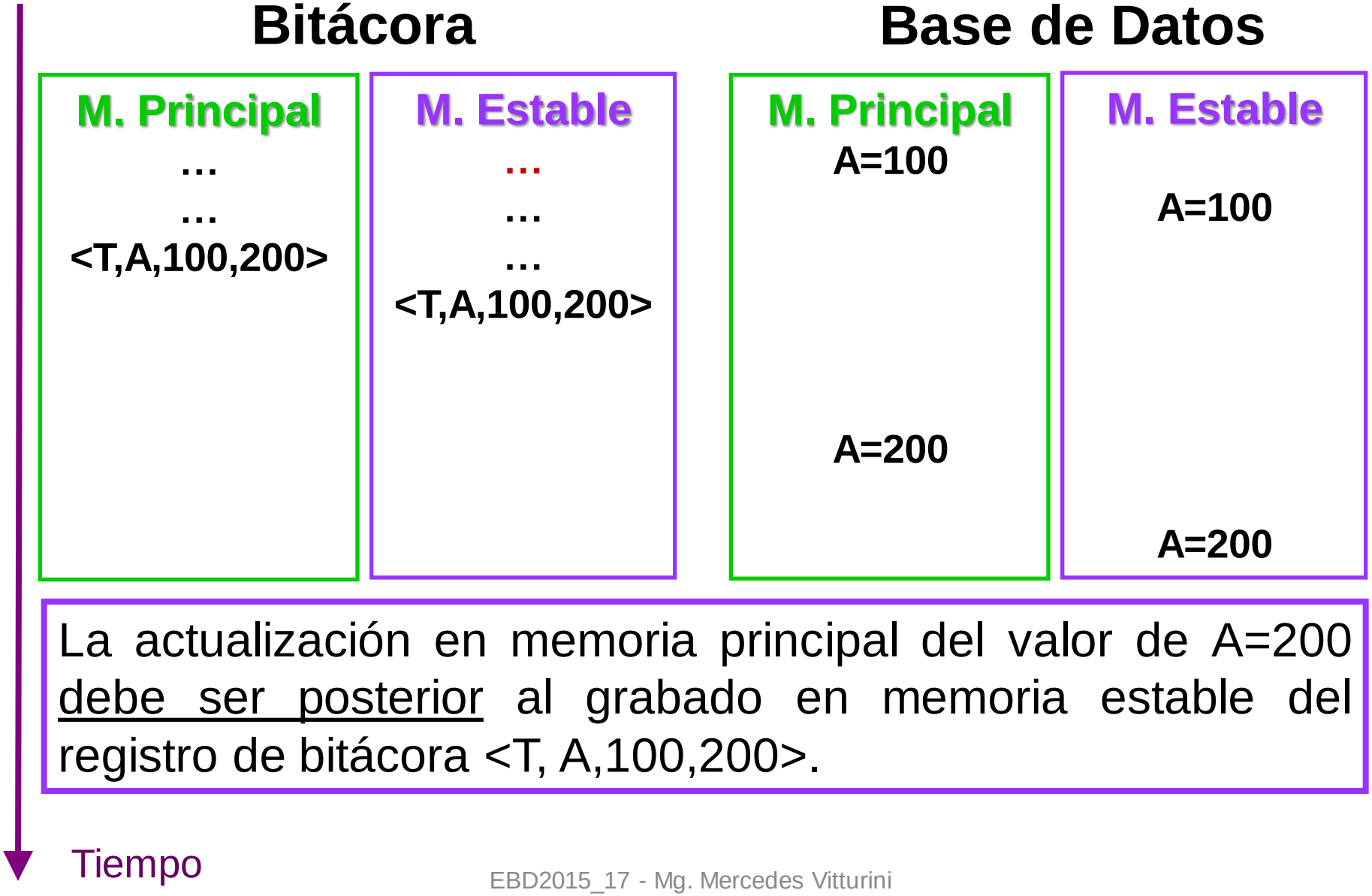
- Cuando la transacción T se inicia se agrega en bitácora el registro **<T, start>**
- Cada operación **Write(X)** de T, agrega el registro de bitácora con los *valores anterior* y *actual* para X,

$$\mathbf{<T, X, v_1, v_2>}$$

- Luego se actualiza la base de datos (*buffer de datos*).
- Cuando T está cometida se agrega en bitácora el registro **<T, commit>**

Nunca se actualiza la base de datos antes de que se escriba el registro de bitácora en memoria estable.

Actualización efectiva de los datos



La actualización en memoria principal del valor de A=200 debe ser posterior al grabado en memoria estable del registro de bitácora <T, A,100,200>.

Ejemplo

T0:Read(A,a_1)
$a_1 \leftarrow a_1$-50
Write(A,a_1)
Read(B,b_1)
$b_1 \leftarrow b_1$+50
Write(B,b_1)

T1:Read(C,c_1)
$c_1 \leftarrow c_1$-100
Write(C,c_1)

Bitácora (M. Estable)

<T0,start>
<T0,A,1000,950>
<T0,B,2000,2050>

<T0,commit>
<T1,start>
<T1,C,700,600>

<T1,commit>

Base de Datos (M. Estable)

A=1000
B=2000
C= 700

A= 950
B=2050
C= 700

A= 950
B=2050
C= 600

Modificación inmediata: recuperación ante un fallo

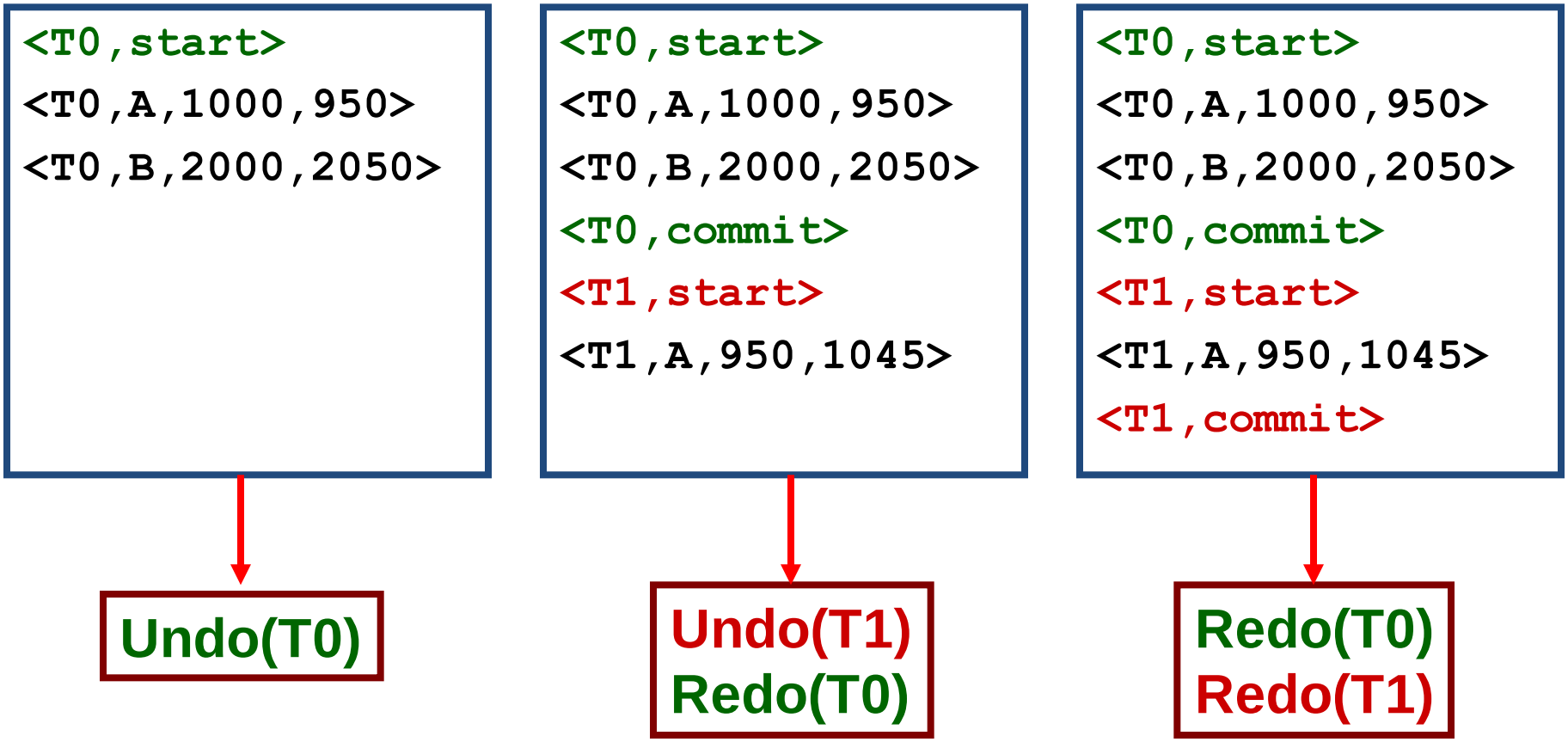
- El esquema de recuperación con modificación inmediata necesita de las operaciones **UNDO** y **REDO** usando información de la bitácora.
  - UNDO(T): restaura todos los ítems de dato que T actualiza a los valores que tenía anteriormente.
  - REDO(T): asigna los nuevos valores a todos los ítems de dato que actualiza la transacción T.
- Las operaciones undo y redo son idempotentes:

UNDO(T)=UNDO(UNDO(T)) Y **REDO(T)=REDO(REDO(T))**

Recuperación ante fallos

- Consideremos que se produjo un fallo del sistema. El esquema de recuperación inmediata consulta la bitácora para determinar cuáles transacciones deben deshacerse o rehacerse.
- ¿Cuándo aplicar undo(T)?
  - Cuando la bitácora contiene el registro <T,start> pero no contiene el registro <T,commit>.
- ¿Cuándo aplicar redo(T)?
  - Cuando la bitácora contiene tanto el registro <T,start> como el registro <T,commit>.

Recuperación de fallos



Es necesario realizar las operaciones de undo antes que las operaciones redo.

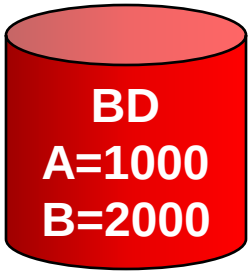
Esquema Modificación Diferida

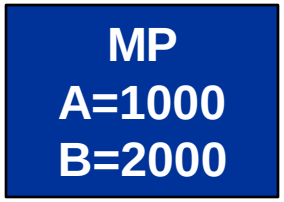
- En el esquema de *modificación diferida* las modificaciones que realice una transacción (*write*)
  - Durante la ejecución de la transacción sólo se graban en los registros de la bitácora,
  - Las escrituras en la base de datos se demoran hasta que la transacción este cometida.
- La información en la bitácora se utiliza cuando se ejecutan las escrituras diferidas.
- Si ocurre un *fallo de sistema* o *fallo de transacción* antes de que la transacción este cometida se ignora la información de la bitácora.

Modificación diferida: acciones preventivas

- Antes de que comience la ejecución de la transacción T_i, se agrega en bitácora el registro *<T_i, start>.*
- Si T_i realiza una operación *write(X,v)*, se agrega en bitácora el registro *< T_i, X, v>.*
  - Modificación diferida requiere solamente del nuevo valor del dato, omitiendo el valor antiguo.
- Cuando T_i está cometida, se escribe el registro de bitácora *< T_i, commit>.*
- Los registros *log* se usan para las escrituras diferidas.

Modificación Diferida



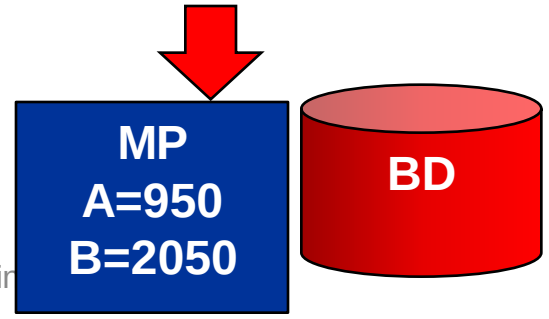


T1

```
Read (A, a1)
a1 = a1 - 50
Write (A, a1)
Read (B, b1)
b1=  b1 + 50
Write (B, b1)
```

Bitacora

```
<start T1>
<T1, A, 950>
<T1, B, 2050>
<commit T1>
```



Recuperación y modificación diferida

- Supongamos una secuencia de transacciones T_1, T_2, … T_n ejecutándose en serie.
- En cualquier momento puede ocurrir una caída del sistema.
- Para recuperar el estado consistente, *necesitarán rehacerse* **(*REDO* T_i)** de todas aquellas transacciones que tienen en bitácora **<T_i, start> y < T_i, commit>.**

- ✎ ¿Y si ocurriese una caída del sistema mientras se realizan tareas de recuperación?

Ejemplo

T0 transfiere 50 pesos de la cuenta A a la cuenta B.

T1 quita 100 pesos de la cuenta C.

```
T0:Read(A,a1)
   a1 ← a1-50
   Write(A,a1)
   Read(B,b1)
   b1 ← b1+50
   Write(B,b1)
```

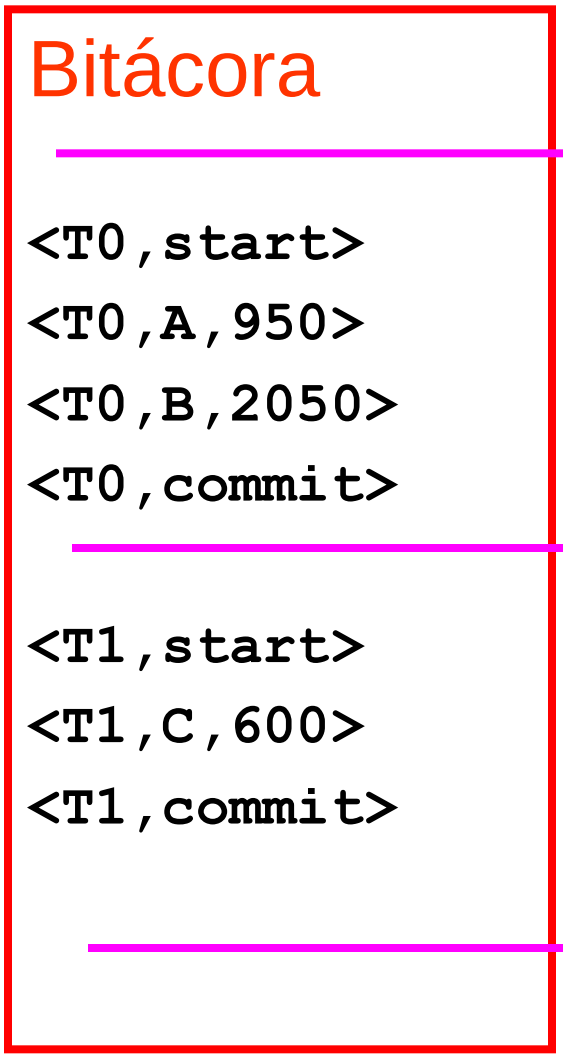
```
T1:Read(C,c1)
   c1 ← c1-100
   Write(C,c1)
```

Inicialmente se asume <u>única transacción activa por vez</u> (ejecuciones en serie)

Ejemplo

T0:Read(A,a_1)
$a_1 \leftarrow a_1$-50
Write(A,a_1)
Read(B,b_1)
$b_1 \leftarrow b_1$+50
Write(B,b_1)

T1:Read(C,c_1)
$c_1 \leftarrow c_1$-100
Write(C,c_1)



Buffer de Datos

A=1000
B=2000
C= 700

A= 950
B=2050
C= 700

A= 950
B=2050
C= 600

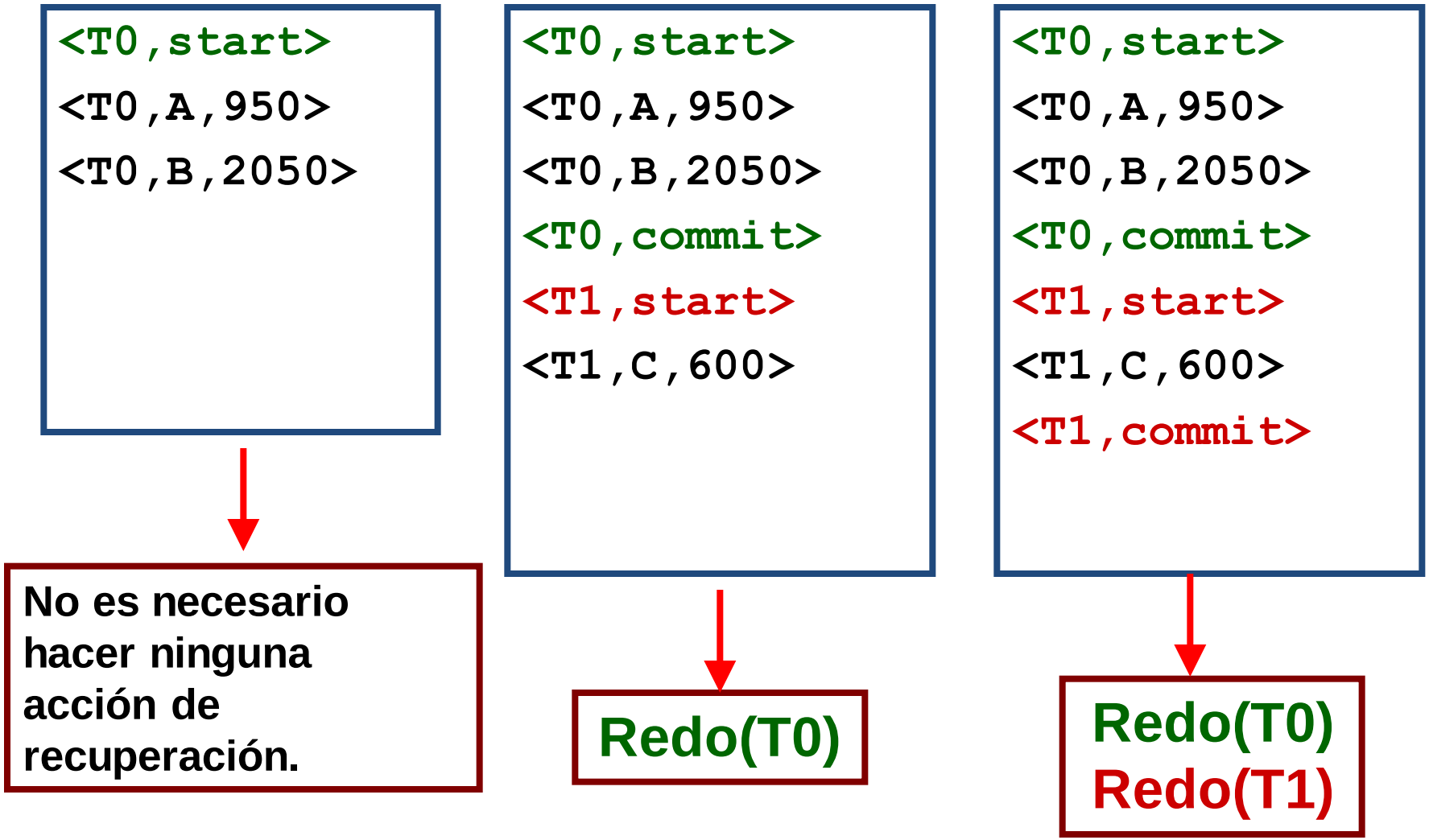
Recuperación ante Fallos

- El esquema de recuperación ante fallos con modificación diferida usa la operación **REDO** y la información de la bitácora.
  - **REDO(T):** asigna los nuevos valores a todos los datos que actualiza la transacción T.
- La operación redo es *idempotente*, esto es:

$$\text{REDO}(T) = \text{REDO}(\text{REDO}(T))$$

- ¿Cuando aplicar redo(T) después de un fallo?
  - Siempre que en bitácora existan los registros <T, start> y <T, commit>.

Recuperación de fallos



Temas de la clase de hoy

- Sistema de Recuperación.
  - Tipos de Fallos.
  - Estructuras de Almacenamiento.
- Mecanismos de recuperación usando bitácora para ambientes no concurrentes.
    - Modificación Diferida.
    - Modificación Inmediata.

Bibliografía

- *Database System Concepts* – A. Silberschatz. Capítulos 16 (ed. 2010).
- *DataBase System – The Complete Book* – H. Molina, J. Ullman. Capítulo 17.