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

Elementos de Bases de Datos

Segundo Cuatrimestre 2015

Clase 18:

Fallos en un SMBD

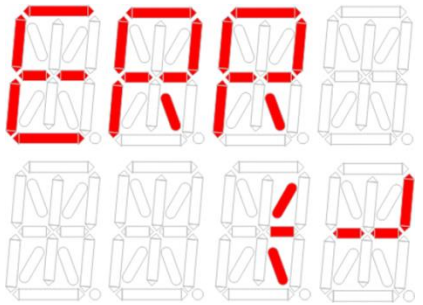
Sistema de Recuperación – Segunda Parte

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

Recuperación del Sistema

1. Los SMBD están sujeto a fallos.
2. Un fallo potencialmente podría generar pérdida de datos y/o inconsistencias.
3. Ante un fallo, el SMBD debe realizar su *'recuperación',* dejando la base de datos en el estado consistente anterior al fallo.

- Tipos de fallos:
  - **Fallo de Transacción**
  - **Caída de Sistema**
  - **Rotura de disco**



Repaso - Tipos de Fallos

☒ **Fallos de la Transacción**

- *Error lógico*: la transacción no puede continuar su ejecución a causa de alguna condición interna.
- *Error del sistema*: el sistema alcanza un estado no correcto. La transacción puede volver a ejecutarse más tarde.

Otras transacciones continúan su ejecución El sistema sigue operativo

☒ **Caída de sistema:** por ejemplo errores del hardware o software de base de datos o software del sistema operativo.

☒ **Fallo de disco:** daño físico en el medio de almacenamiento masivo.

Varias transacciones deben ser restauradas. El sistema no sigue operativo

Algoritmos de Recuperación (AR)

- Los **algoritmos de recuperación (AR)** son técnicas para asegurar la *consistencia, atomicidad* y *durabilidad* de las transacciones a pesar de los fallos.

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

AR mediante Bitácora - Repaso

La **bitácora** es una estructura que se utiliza para guardar información sobre las modificaciones que se realizaron a los datos. La implementación que vimos contiene registros de control y registros con los atributos:

- **Nombre de la Transacción**: el nombre de la transacción que ejecutó la actualización (Write).
- **Nombre del Dato:** el nombre único del dato escrito.
- **Valor Antiguo:** el valor del dato anterior a la escritura.
- **Valor Nuevo:** el valor que tendrá el dato después de la escritura.

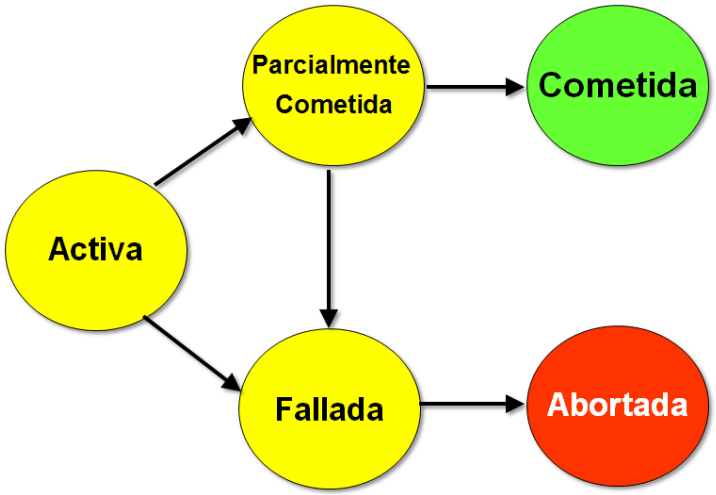
Repaso – Registros de la Bitácora

- Cuando la transacción T_i se inicia, se registra agregando en bitácora: **< T_i, start>**
- Cada vez que la transacción T_i ejecuta **write** (X,x_2), se registra *previamente* en bitácora agregando: **< 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.
- Cuando la transacción T_i finalizó con su última instrucción agrega en bitácora el registro: **<T_i, commit>**

Esquemas de Modificación

Existen dos esquemas de modificación

- Modificación Inmediata
- Modificación Diferida



Modificación diferida

Implementa la propiedad de atomicidad de las transacciones grabando las modificaciones sólo en los registros bitácora y aplazando las operaciones write en la base de datos hasta que la transacción comete.

- La información en la bitácora asociada a la transacción parcialmente cometida se usa durante la ejecución de las escrituras diferidas.
- Para el esquema de recuperación sólo se requiere de la operación **redo** (rehacer) con información de la bitácora.
  - **REDO (T)** cuando en la bitácora existen los registros <T, start> y <T, commit>.

Modificación inmediata

Las modificaciones en la base de datos se realizan mientras la transacción está activa, y en distintos momentos antes de alcanzar el estado de cometida.

- Cuando se produce un fallo, el esquema de recuperación consulta a la bitácora para determinar que transacciones deben deshacerse o rehacerse.
  - **UNDO (T):** cuando la bitácora contiene el registro <T, start> pero no contiene el registro <T, commit>.
  - **REDO (T):** cuando la bitácora contiene tanto el registro <T, start> como el registro <T, commit>

☛ Los registros de la bitácora se deben grabar en memoria estable antes de cualquier modificación de la base de datos.

Sistema de Recuperación con Logs

Simplificaciones asumidas a liberar

1. Output de registros de buffer log inmediato
   - Output del buffer log por bloques. Implicancias
     - Estados de la Transacción: Cometida
     - Operación Output + Write Ahead Logging
2. Recuperación completa del archivo log
   - Checkpointing
     - El proceso de checkpoint
     - El proceso de recuperación
3. Entorno Concurrente
   - Varias transacciones activas
   - Checkpoint y recuperación

Restricciones impuestas

Hasta ahora impusimos las siguientes *restricciones*:

1. La operación output de la bitácora es inmediato. Los registros de la bitácora están "inmediatamente" en memoria estable.
   - La escritura de los registros bitácora en memoria estable siempre es anterior a la escritura de los correspondientes bloques de datos.
   - El ouput de los registros de los datos ocurre en cualquier momento posterior.
2. Durante el proceso de recuperación se recupera al sistema completo (desde la primera transacción de la bitácora).
3. La ejecución de las transacciones es en serie.

Buffering de Registros de Bitácora

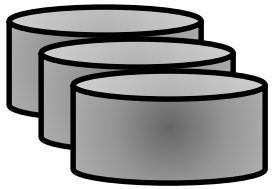
"Hasta ahora supusimos que cada registro de bitácora se graba en memoria estable inmediatamente después de cuando se crea "

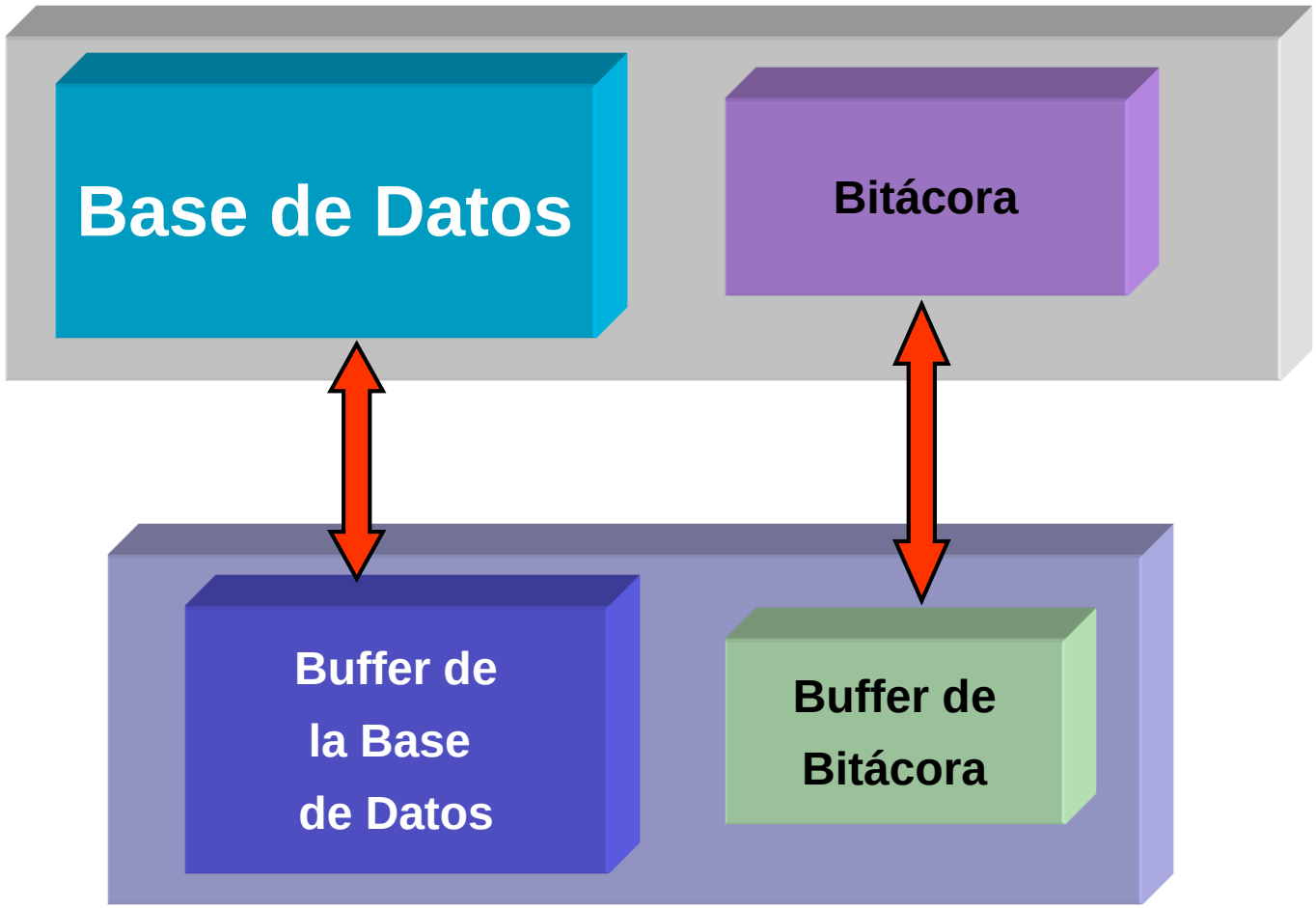
- Esto generaría un *overhead extra* Demasiado costoso.

- Cada registro de bitácora es mucho más pequeño que el tamaño del bloque que lo contiene, la grabación de cada registro de bitácora se traduce en una grabación mucho mayor en el nivel físico y por cada registro.

Organización de la Memoria

Memoria Estable



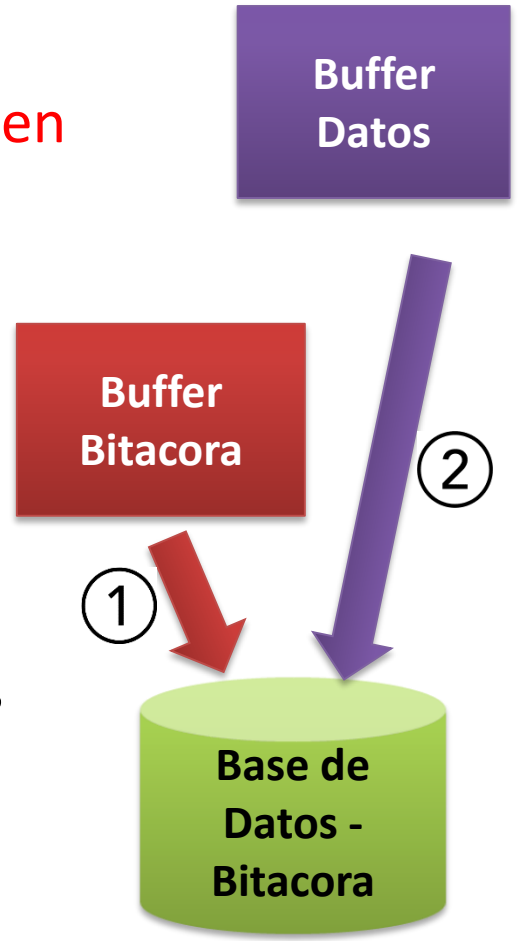


Memoria Principal

Buffering + Registros de bitácora

Reglas para grabar el buffer de registros de bitácora:

1. El output de registros de bitácora es en el orden en que se crean.
2. La transacción T_i alcanza el estado cometida cuando se grabó en memoria estable <T_i *commit*>.
3. *Antes de un output de cualquier bloque de datos* desde el buffer de memoria principal, todos los registros de bitácora asociados a los datos del bloque deben ser grabados en memoria estable previamente: **write-ahead logging.**



Buffering de Base de datos

- Los datos se traen a memoria principal según se requieran mediante un esquema de transferencia de bloques.

☛ **Si un bloque B1 en memoria principal ha sido modificado y necesita ser reemplazado por un bloque B2, entonces**:

1. Grabar B1 (Output(B1)).
2. Luego cargar B2 (Input(B2)).

- Este es el concepto estándar de memoria virtual.

Buffering Base de Datos + Buffering de Bitácora

- **Supongamos que la entrada del bloque B2 hace que el bloque B1 deba salir de memoria principal.**

Los pasos a seguir y en ese orden son:

1. Grabar en memoria estable hasta todos los registros de bitácora pertenecientes al bloque **B1** (ie. grabar en memoria estable los bloques de bitácora hasta cubrir los bloques que contienen registros de **B1**)
2. Grabar el bloque **B1** en memoria estable.
3. Cargar el bloque **B2** desde el disco a memoria principal.

Buffering: base de datos + bitácora

Transacción T

...

Read(B,b_1)

...

Bitácora

<T,start>

<T,A,1000,950>

Se graba en memoria estable:
1)<T,start> ,
<T,A,1000,950>

2) Bloque donde reside el dato A.

Bloque donde reside B fuera de MP

Se elige sacar bloque donde reside A

✍Ejercicio

- Plantee una situación en la que grabar primero los bloques del buffer de datos en memoria estable antes que los registros correspondientes de bitácora provoque una situación de inconsistencia ante una situación de recuperación.

El proceso de recuperación

Ante un fallo de sistema las acciones a seguir son:

1. Iniciar el proceso de recuperación.
   - **Se suspende la atención de requerimientos de usuarios.**

2. Analizar la bitácora determinando el conjunto de transacciones a rehacer y *deshacer*.
   - *Deshacer* depende de la estrategia de modificación.

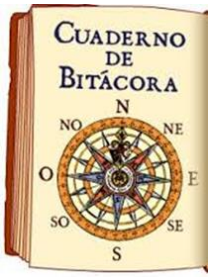
3. Reiniciar la operación normal del sistema.

Recuperación con Registros Bitácora

- Cada vez que ocurre **un fallo de sistema** sería necesario consultar *la bitácora completa* para ver qué transacciones deben rehacerse y cuáles deshacerse.

☒ Problemas:

- El proceso de recuperación consume demasiado tiempo.
- Varias de las transacciones cometidas que según el algoritmo deben rehacerse, seguramente ya escribieron sus actualizaciones en la base de datos en memoria estable (output).
- Volver a hacerlas no causa daño, pero insume tiempo.



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

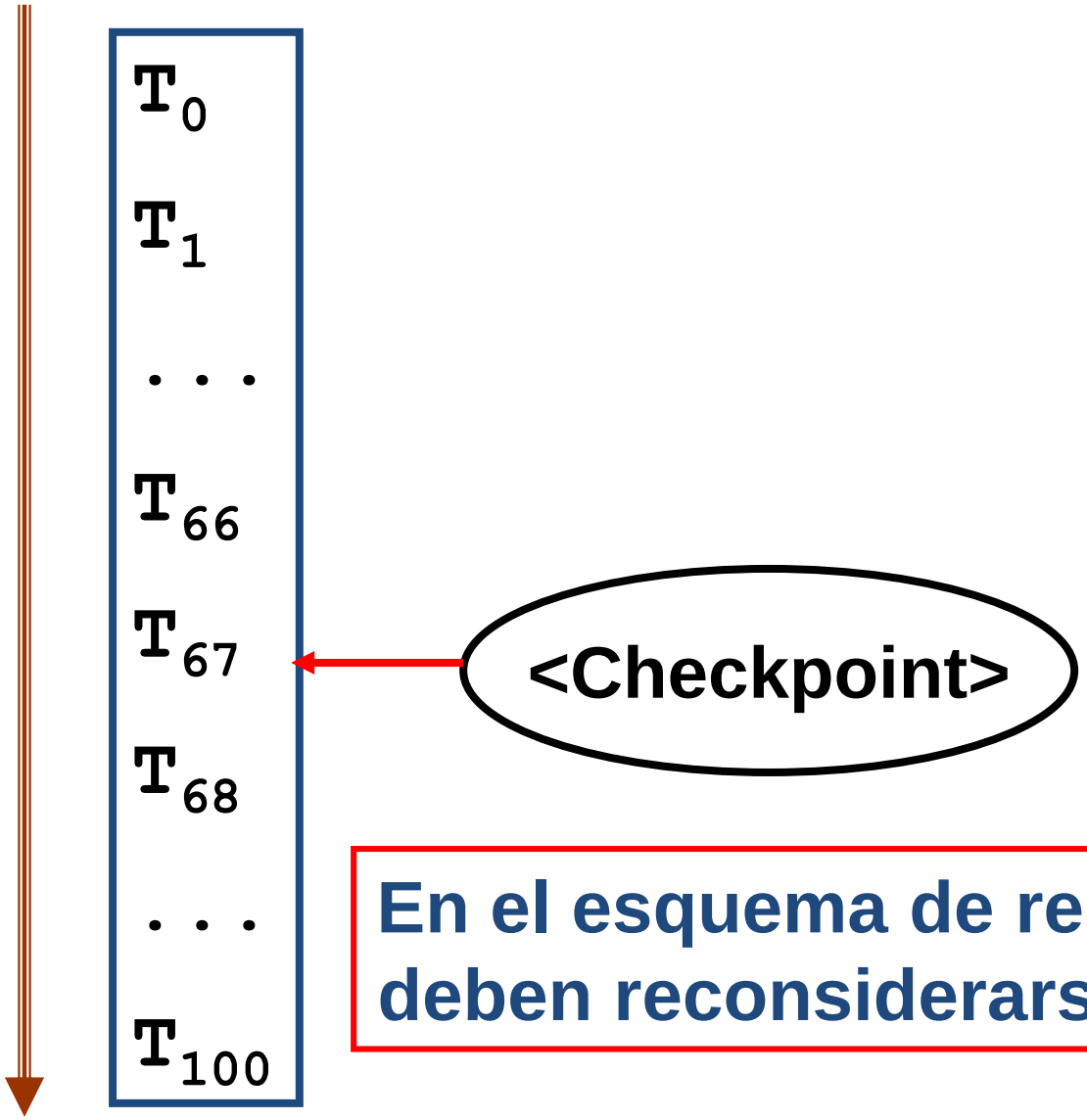
Puntos de Verificación (Checkpoints)

Se agrega un procedimiento al sistema de recuperación de **checkpointing** o puntos de verificación, que sigue los pasos:

1. Grabar en disco (output B) todos los bloques de registros de bitácora que están en memoria principal.
2. Grabar en disco (output B) los bloques de datos modificados de los registros intermedios (buffer de MP).
3. **Grabar un registro de bitácora <checkpoint> en memoria estable.**

☛ Durante el proceso de checkpointing se *suspenden todas las tareas* para realizar sólo *tareas de sistema.*

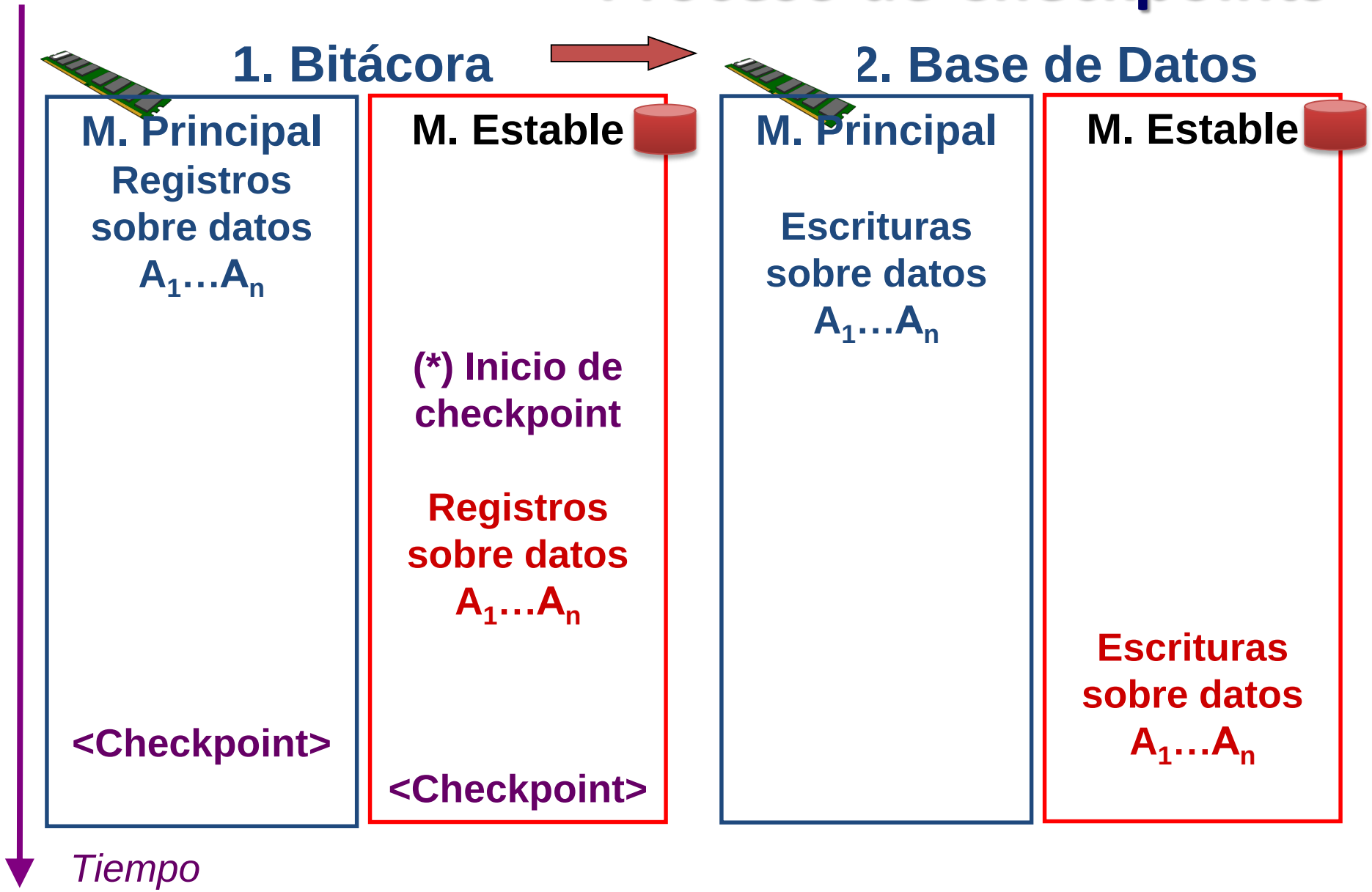
Ejemplo



En el esquema de recuperación sólo deben reconsiderarse T_{67}, T_{68},..., T_{100}.

Fallo de Sistema

Proceso de Checkpoints

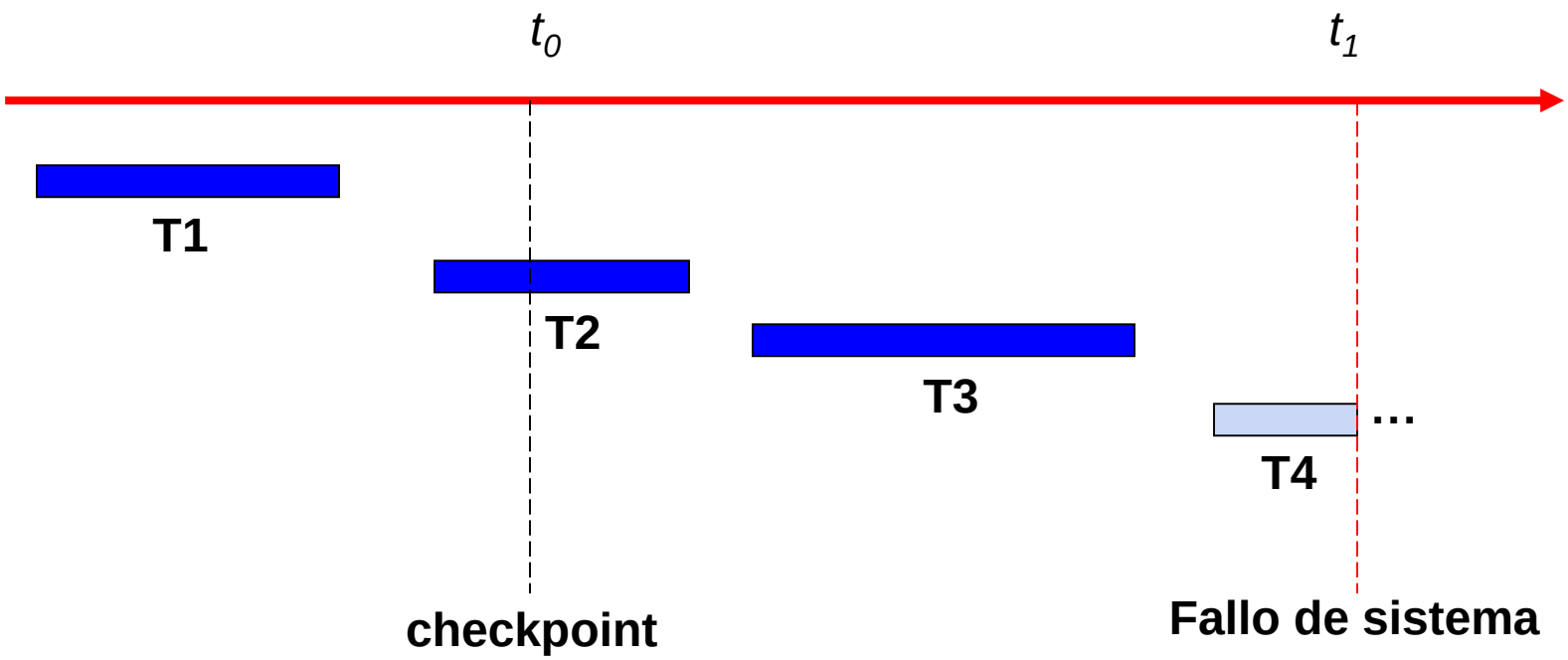


Recuperación + Checkpoints

- Para cada transacción T_i tal que aparece en la bitácora el registro <T_i, *commit*> **antes** del registro <*checkpoint*>, no se requiere ejecutar un REDO.
- Después de un fallo de sistema:
  - Examinar la bitácora hacia atrás buscando el primer registro **<*checkpoint*>** y cual es el registro **<T_i, start>** más cercano (determina la última transacción T_i que comenzó a ejecutarse antes del *checkpoint*).
  - Luego, se aplica UNDO o REDO sobre T_i y todas las transacciones T_k que le suceden.

Ejemplo

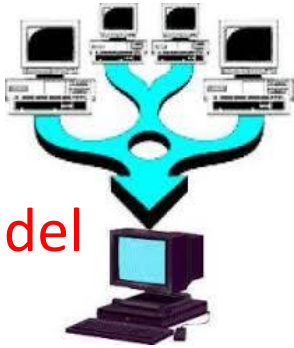
Checkpointing + Modificación Inmediata



- ☑ **T1** se puede *ignorar*.
- ☑ **T4** debe se *deshecha* (**undo**).
- ☑ **T2** y **T3** deben ser *rehechas* (**redo**).

Checkpoints + Modelo Concurrente

- El esquema secuencial considera:
  - Transacciones que comenzaron después del checkpoint más reciente (pueden ser varias).
  - La única transacción activa (si existe) al momento de dicho checkpoint.
  - La única transacción activa (si existe) al momento que se produce un fallo de sistema.
- **Diferencias con un esquema concurrente:**
  - Podrían existir varias transacciones activas al momento del checkpoint.
  - Podrían existir varias transacciones activas al momento de un fallo de sistema.



Recuperación y Concurrencia

IMPORTANTE

- Un sistema centralizado, aún cuando incorpora facilidades de concurrencia, cuenta con:
  - un único buffer de registros de disco y
  - una única bitácora.

Checkpoint + Modelo Concurrente

El esquema de checkpoint se modifica levemente si existen transacciones concurrentes.

1. Reemplazar el registro <checkpoint> por un registro de la forma: **<checkpoint L>,** donde ***L*** define la *lista de transacciones activas* al momento del checkpoint.
2. La lista ***L*** limita la búsqueda hacia atrás del checkpoint en la bitácora.

- Como en el caso anterior, no existen actualizaciones a los bloques de buffer ni a la bitácora durante el proceso de checkpointing.

Checkpoints + Modelo Concurrente

Caída de Sistema (*Crash*)

- El proceso de recuperación involucra a todas las transacciones ejecutadas y en ejecución (posiblemente más de una) a partir del último punto de verificación (*checkpoint*).
- El sistema de recuperación recorre la bitácora para construir dos listas:
  - **undo-list**
  - **redo-list,**

Undo-List y Redo-List

1. Recorrer secuencialmente la bitácora hacia atrás hasta el primer <*checkpoint*, L>:

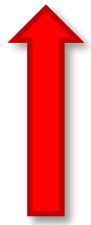
Por cada registro **<T_i, *commit*>**, se agrega T_i a **redo-list.**

Por cada registro **<T_i, *start*>**, si T_i no está en *redo-list* entonces se agrega a la lista **undo-list.**

2. Al llegar <*checkpoint*, L> controlar la lista ***L*** de transacciones activas:
   - Por cada transacción T_i de ***L*** si T_i no esta incluida en *redo-list* se agrega a *undo-list.*

Recuperación ante Fallo de Sistema

1 Volver a recorrer la bitácora hacia atrás, realizando **undo de cada registro** que corresponda a una transacción de la lista *undo-list*. El proceso se detiene al alcanzar en la bitácora el item <T_i, start> de cada transacción T_i en la lista *undo-list*.



2 Avanzar hasta el más reciente <checkpoint L> de la bitácora.

3 Recorrer la bitácora hacia adelante desde el más reciente <checkpoint L> y realizar **redo de cada registro** que corresponda a una transacción T_i en la lista *redo-list*.

Recuperación ante Fallos

- Es importante respetar el orden de ejecución de las operaciones para la recuperación.
- Las operaciones UNDO deben realizarse siempre antes que las operaciones REDO.
- Las operaciones de UNDO se realizan recorriendo la bitácora desde abajo hacia arriba, esto es, en orden inverso al que se ejecutaron.
- Las operaciones de REDO se realizan recorriendo la bitácora desde arriba hacia abajo, esto es, en el mismo orden en que se actualizó.

CRASH de Sistema

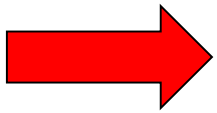
<T_1,start>
<T_1,D,0,15>
<T_2,start>
<T_2,A,20,30>
<checkpoint, <T_1, T_2>>
<T_1,B,50,20>
<T_1,commit>
<T_3,start>
<T_3,B,20,40>
<T_4,start>
<T_4,B,40,80>
<T_4,C,100,50>
<T_3,commit>
... *Crash* ...

Dada la siguiente foto de la bitácora antes de un crash de sistema.

Undo-List: **T_4, T_2**

Redo-List: **T_1, T_3**

B: 50,
A: 20,
C: 100,
D: 15,



B: 40,
A: 20,
C: 100,

Fallo de Transacciones

- El **falla de una transacción** también se resuelve usando la bitácora.
- Si una transacción T tiene que ser retrocedida también implica recorrer la bitácora hacia atrás: una misma transacción puede realizar más de un cambio sobre el mismo dato.

$$\ldots,\langle T_5,A,10,20\rangle,\ldots,\langle T_5,A,20,30\rangle,\ \ldots$$

- Esto es, T_5 primero modifica A de 10 a 20 y luego de 20 a 30. Si no la recorremos hacia atrás, se restauraría A con el valor 20 (cuando debe ser 10).

Ejemplo – Fallo de una transacción

T1	T2
read (A)	
write (A)	
	read (A)
	write(B)
	write (A)
read (B)	
…	…

Estampillas

R-ts(A)= ts(T_1)

W-ts(A)=ts(T_1)

R-ts(A)=ts(T_2)

W-ts(B)=ts(T_2)

W-ts(A)=ts(T_2)

Bitacora

<T_1 start>

<T_1, A, 100, 200>

<T_2 start>

<T_2, B, 400, 200>

<T_2, A, 200, 500>

- **Protocolo**: Estampilla de tiempo con ts(T_1) < ts(T_2)
- **T_1** debe retroceder por no cumplir con el protocolo
- **T_2** debe retroceder en cascada.

Checkpoints Difusos (Fuzzy)

- El proceso para incluir un punto de verificación (checkpointing) requiere que las actualizaciones a las base de datos se suspendan temporalmente.
  - No se atienden requerimientos de usuarios para realizar tareas de sistema
- La técnica de **puntos de verificación difusos** (*fuzzy checkpointing*) permite que se reinicien las actualizaciones después de grabar el registro de checkpoint *difuso* en bitácora en memoria estable pero antes de que los bloques modificados se escriban en disco.

Checkpoints Difusos

1. Se suspenden todas las tareas de usuario.
   - Se graba en memoria estable la bitácora.
   - Se graba <*checkpoint* L> en bitácora en memoria estable.
   - Se identifica la *lista M* de bloques buffer modificados.
2. Se permite a las transacciones proceder.
3. En forma concurrente se continua con el output de cada uno de los bloques de la lista M del buffer de datos
   - No se pueden modificar los bloques de M mientras están en proceso output.
4. En la recuperación usando fuzzy checkpoint, se comienza el recorrido desde el registro checkpoint apuntado por *last_checkpoint*

Administración de buffer

El buffer de base de datos se puede manejar de 2 formas:

- El SMBD se reserva parte de memoria principal para usarla como buffer.
  - El tamaño del buffer está acotado por los requerimientos de memoria por parte de otras aplicaciones.
  - Aunque no se use el espacio del buffer, no puede usarse para otro fines (menos flexible).
- El SMBD implementa el buffer en un espacio de memoria virtual provisto por el sistema operativo.
  - El tamaño del buffer es flexible.
  - El SO decide sacar los bloques de memoria, de modo tal que el SMBD nunca tiene control sobre el número de bloques de buffer.

Rotura de Disco

- Aunque el fallo de memoria no volatil es menos frecuente, el sistema debe estar preparado también para este tipo de fallos.
- La técnica es copiar periódicamente el contenido entero (***dump***) de la base de datos a un almacenamiento estable (por ejemplo, cintas magnéticas, dump en servidores remotos).
- Se ejecuta un proceso similar al de checkpointing.
- Ninguna transacción debe estar activa durante el proceso de dump

Copias de Seguridad (dump)

El proceso de Dump de Base de Datos

1. Se vuelca el buffer de los registros de bitácora residiendo actualmente en memoria principal al almacenamiento estable.
2. Se vuelcan todos los bloques de buffer de datos al disco.
3. Se copian los contenidos de la base de datos al medio de almacenamiento estable.
4. Se vuelca un registro de bitácora *<dump>* al almacenamiento estable.

Fallo de Disco

Si falla el almacenamiento no volátil:

1. Esto restaura la base de datos al estado en que se encontraba cuando se realizó la copia de la bases de datos completa a memoria estable (dump).
2. Luego, el sistema consulta la bitácora para hacer redo de *todas las transacciones cometidas* después del registro <*dump*>.

Temas de la clase de hoy

- Sistema de Recuperación
  - Puntos de Verificación. Modificación para ambientes concurrentes.
  - Puntos de Verificación Difusos.
  - Recuperación y Buffering.
  - Implementación de memoria estable.

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