TEMA 4 – Control de la concurrencia

1. Ejecución concurrente de transacciones: anomalías

RECORDAMOS (TEMA 2)

Transacción: secuencia de operaciones de acceso a la BD (consulta o actualización) que constituyen una unidad de ejecución.

Procesar correctamente una transacción significa:

a) Todas las operaciones de la transacción se ejecutan con éxito y sus cambios quedan grabados permanentemente en la BD.

b) La transacción no tiene ningún efecto en la BD.

Principio ACID: **A**tomicidad, **C**onsistencia, **A**islamiento, **P**ersistencia

Ejecución concurrente de transacciones: ejecución intercalada de las operaciones de dos o más transacciones. Puede generar anomalías:

- **Anomalías de lectura:**
  - **Lectura sucia:** T2 lee un elemento de datos actualizado por T1 que todavía no ha finalizado. Si T1 es anulada, T2 ha leído un valor de datos que nunca ha existido.

    → *Se respeta la propiedad de aislamiento*
  - **Lectura no repetible:** T2 actualiza un elemento de datos que ha sido leído por T1, que todavía no ha finalizado. Si T1 vuelve a leer el mismo elemento, leerá un valor distinto.

    → *No se respeta la propiedad de aislamiento*
  - **Lectura de fantasmas:** se da cuando una transacción solicita la lectura de tuplas que cumplan una condición. Si T2 actualiza una tupla que ha sido leída por T1, que todavía no ha finalizado, T1 no está haciendo sus lecturas en el mismo estado de la BD.

    → *No se respeta la propiedad de aislamiento*
- **Anomalías de escritura:**
  - **Pérdida de actualizaciones:** T1 actualiza un elemento de datos que ha sido leído por T2 que todavía no ha finalizado. Si T2 vuelve a acceder al mismo elemento para actualizarlo, se pierde la actualización de T1.

    → *No se respeta la propiedad de aislamiento*

2. Control de la concurrencia en SQL

RECORDAMOS (TEMA 2)

```
SET TRANSACTION modo [,modo]
modo := nivel de aislamiento | modo de acceso | área de diagnóstico

modo de acceso := READ ONLY | READ WRITE

área de diagnóstico := DIAGNOSTICS SIZE número
```

El **nivel de aislamiento** nos permite elegir el nivel de concurrencia que debe realizar el SGBD:

```
nivel de aislamiento := ISOLATION LEVEL {READ UNCOMITTED
                                        | READ COMMITTED
                                        | REPETEABLE READ
                                        | SERIALIZABLE}
```

Si una transacción se ejecuta en un nivel de aislamiento distinto a **serializable**, entonces pueden darse algunas de las anomalías estudiadas.

	Anomalía			
Nivel de aislamiento	Lectura sucia	Lectura no repetible	Lectura de fantasmas	Pérdida de actualizaciones
READ UNCOMMITED	SÍ	SÍ	SÍ	SÍ
READ COMMITED	NO	SÍ	SÍ	SÍ
REPETIBLE READ	NO	NO	SÍ	SÍ
SERIALIZABLE	NO	NO	NO	NO

3. Planes serializables por conflictos

*El objetivo del control de la ejecución concurrente de transacciones es asegurar que el efecto final de la ejecución concurrente sea el mismo que el que se obtendría si las transacciones se ejecutaran de forma aislada (**en serie**).*

Para entender qué operaciones provocan las anomalías y como se pueden evitar, es necesario establecer un **plan de ejecución de un conjunto de transacciones**.

Nomenclatura para las operaciones:

- $r_i(x)$: la transacción T_i lee x
- $w_i(x)$: la transacción T_i escribe x
- c_i: la transacción T_i se confirma
- a_i: la transacción T_i se anula

Plan de ejecución de un conjunto de transacciones

Ordenamiento de las operaciones de las transacciones en el que las operaciones de cada transacción aparecen en el mismo orden que aparecen en la transacción.

Plan en serie: para cada transacción, todas sus operaciones se ejecutan consecutivamente.

→ *No presenta anomalías, siempre es correcto*

Plan concurrente: las operaciones de cualquier transacción pueden aparecer intercaladas entre sí.

→ *Puede presentar anomalías*

Operaciones en conflicto

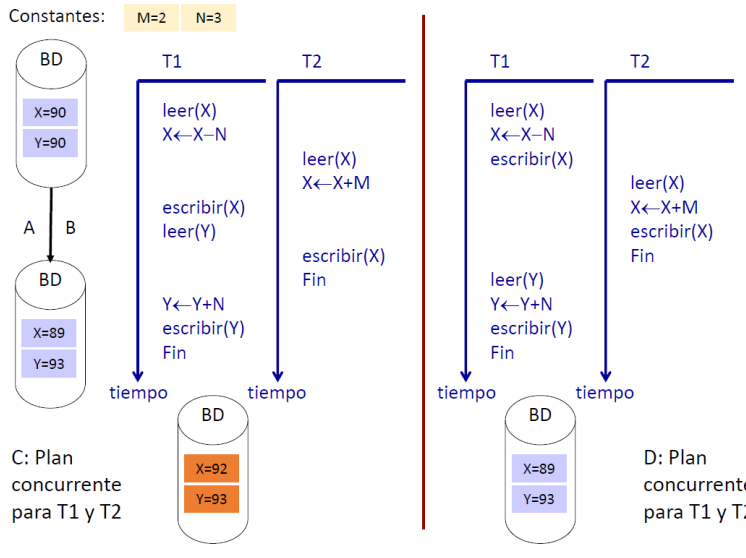
Dos operaciones de un plan están en conflicto si satisfacen las siguientes tres condiciones:

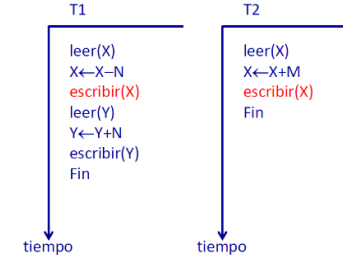
1) pertenecen a distintas transacciones
2) acceden al mismo elemento de datos X
3) al menos una de ellas es una operación $escribir(X)$

Ejemplo: $\mathrm{P}: r_1(x), r_2(x), w_1(x), w_2(x), c_1, c_2$

- $r_1(x)$ y $w_2(x)$ están en conflicto
- $r_2(x)$ y $w_1(x)$ están en conflicto
- $w_1(x)$ y $w_2(x)$ están en conflicto

A continuación se muestran dos ejemplos de planes concurrentes para la ejecución de dos transacciones T_1 y T_2 :





El **plan C** no es correcto ya que se pierde la actualización de T_1, por lo que el valor X que lee T_2 es erróneo, se produce la anomalía de pérdida de actualizaciones.

El **plan D** es correcto ya que el efecto es el mismo que si se hubiese ejecutado primero T_1 y luego T_2 (plan en serie), como si la dos transacciones se hubiesen ejecutado de forma aislada.

Planes equivalentes y planes serializables

Un plan concurrente de un conjunto de transacciones es **serializable** si es equivalente a un plan en serie para las mismas transacciones.

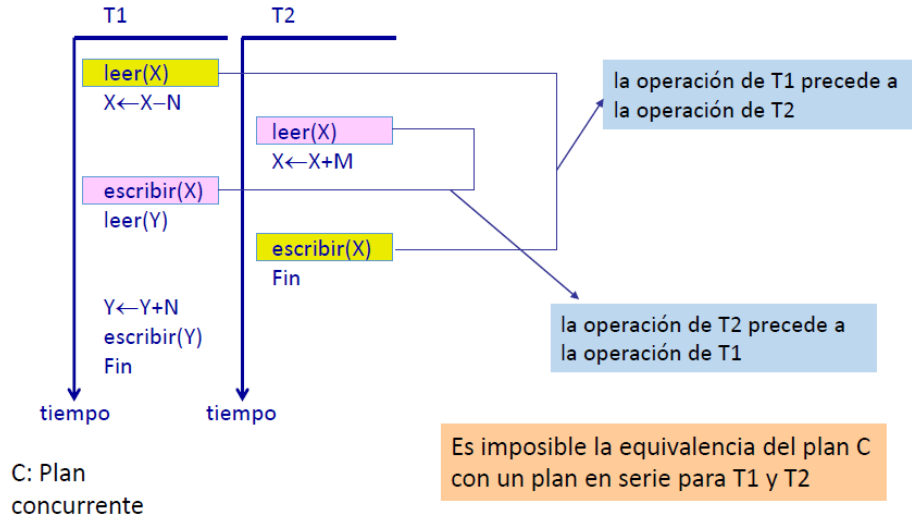
¿QUÉ SIGNIFICA SER EQUIVALENTE? Como la causa de las anomalías está en las operaciones en conflicto, introducimos una definición de equivalencia basada en las operaciones en conflicto:

→ **Equivalencia por conflictos**

*Dos planes son **equivalentes por conflictos** si el orden de dos operaciones cualesquiera en conflicto es el mismo en ambos planes.*

*Un plan es **serializable por conflictos** si es equivalente por conflictos a un **plan en serie**.*

Si revisamos el **plan C** del ejemplo anterior comprobamos que es imposible la equivalencia de este plan con un plan en serie para T_1 y T_2 :



C: Plan concurrente para T1 y T2

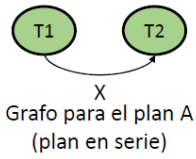
Prueba de seriabilidad por un conflicto de plan

Grafo de serialización: grafo dirigido $G = (N, A)$ que consiste en un conjunto de nodos N y un conjunto de arcos A.

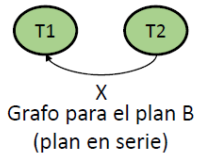
- ✓ El grafo contiene un **nodo** por cada transacción que participa en el plan.
- ✓ Se crea un **arco** $a_{ij} = (T_i \rightarrow T_j)$ si una operación de T_i aparece en el plan antes de una operación en conflicto de T_j.

*Un plan concurrente **no es serializable por conflictos** si su grafo de serialización tiene un **ciclo.***

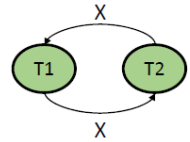
Ejemplos de grafos de serialización:



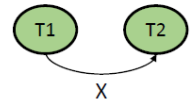
Grafo para el plan A
(plan en serie)



Grafo para el plan B
(plan en serie)

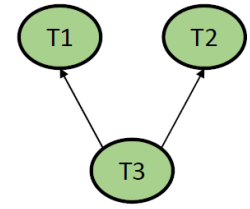


Grafo para el plan C
(plan no serializable)



Grafo para el plan D
(plan seriabilizable)

Puede haber más de un plan en serie equivalente a un plan serializable dado:



Planes en serie equivalentes:

T3 – T1 – T2

T3 – T2 – T1

En la práctica no es viable comprobar la seriabilidad de un plan una vez ha terminado:

→ Los sistemas reales aplican **protocolos** que aseguran, durante la ejecución del plan, que éste será **serializable** sin tener que comprobarlo una vez finalizado.

4. Protocolos de bloqueo

Los **protocolos de bloqueo de elemento de datos** se basan en la idea de restringir el acceso al mismo elemento de datos por varias transacciones.

Las transacciones solicitan el bloqueo de un elemento de datos para acceder a él, y lo liberan posteriormente.

→ *Cuando una transacción solicita el **bloqueo** de un elemento y las **reglas de bloqueo** se lo impiden, la transacción queda en **espera**.*

→ ***BLOQUEO:** estado de un elemento de datos, provocado por una transacción, que determina las operaciones que otras transacciones pueden realizar sobre él.*

Reglas del protocolo de bloqueo de lectura/escritura

Un elemento puede tener **tres estados:** "bloqueado para lectura", "bloqueado para escritura" o "desbloqueado".

- Una transacción debe **"bloquear para lectura"** el elemento de datos X antes de realizar una operación ***leer(X): bloquear_lectura(X)***.

- Una transacción debe **"bloquear para escritura"** el elemento de datos X antes de realizar una operación ***escribir(X)***. También podrá realizar ***leer(X): bloquear_escritura(X)***.

- Si **bloqueo(X) = "bloqueado para lectura"**, otras transacciones pueden "bloquear para lectura el elemento de datos (**bloqueo compartido**).

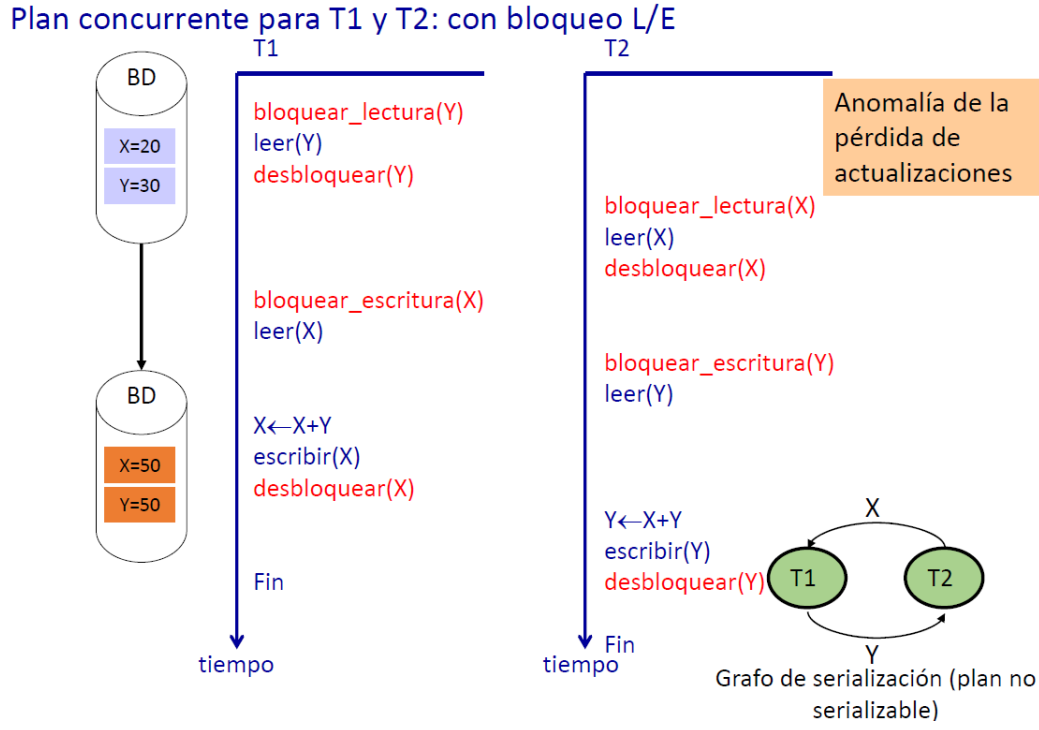
- Si **bloqueo(X) = "bloqueado para escritura"**, ninguna otra transacción puede bloquear el elemento de datos (**bloqueo exclusivo**).

- Una transacción debe **"desbloquear"** el elemento de datos X cuando ya no necesite acceder a él mediante ***desbloquear(X)***.

En estos protocolos, se pueden flexibilizar las operaciones de bloqueo y desbloqueo, con la **conversión de bloqueos**.

- **Promoción del bloqueo:** una transacción T que tiene un bloqueo de lectura sobre el elemento de datos X puede solicitar conseguir el bloqueo de escritura (*solo si T es la única transacción con bloqueo de lectura sobre X*).

- **Degradación del bloqueo:** una transacción T que tiene un bloqueo de escritura sobre el elemento X puede solicitar bajar el bloqueo a lectura liberando el bloqueo de escritura.



PROBLEMA: *se siguen presentando anomalías. El problema del plan concurrente es que T_2 ha desbloqueado demasiado pronto el elemento de datos X que iba a usar luego. Entre el intermedio de ambas operaciones T_1 ha actualizado X.*

Bloqueo en dos fases (B2F)

El protocolo de bloqueo de lectura/escritura no garantiza la seriabilidad (por conflictos) de los planes: hace falta una regla adicional.

BLOQUEO EN DOS FASES (B2F)

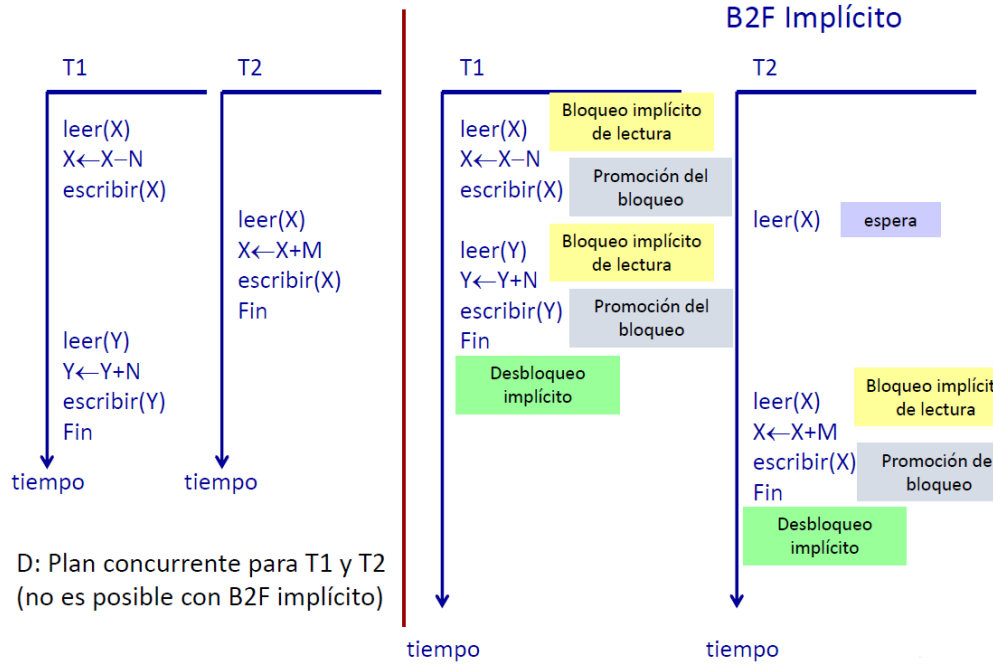
Regla de las dos fases (2F): en una transacción todas las operaciones de bloqueo (*bloquear_lectura*, *bloquear_escritura*) preceden a la primera operación de desbloqueo (*desbloquear*) de la transacción.

El protocolo B2F limita la concurrencia, pero **asegura la seriabilidad** (por conflictos) de los planes sin tener que examinarlos.

B2F Implícito: evita tener que escribir las instrucciones de bloqueos, desbloqueos y promociones.

El SGBD se encarga de generar implícitamente los bloqueos de lectura y de escritura y los desbloqueos sobre elementos de datos:

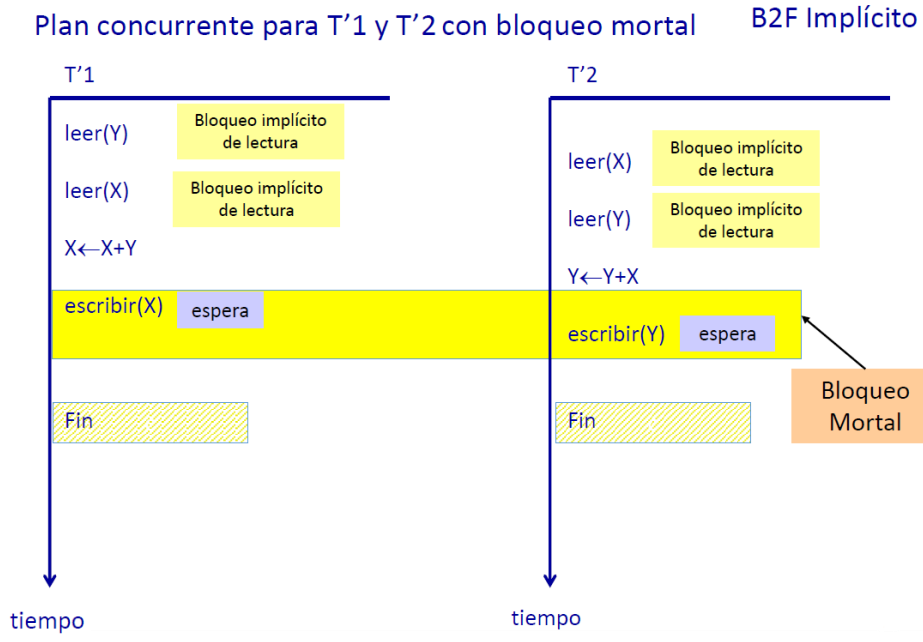
- *Leer(X)* genera un bloqueo para lectura sobre X
- *Escribir(X)* genera un bloqueo para escritura sobre X.
- La finalización de la transacción (anulación o confirmación) genera el desbloqueo de todos los elementos de datos bloqueados por la transacción.



Bloqueo mortal

Bloqueo mortal: cada transacción T_i en un conjunto de dos o más transacciones está esperando un elemento de datos que está bloqueado por otra transacción T_j de dicho conjunto.

→ **Algoritmos de detección del bloqueo mortal:** el sistema verifica si el plan está en un estado de bloqueo mortal.



Grafo de espera: grafo dirigido $G = (N, A)$ que consiste en un conjunto de nodos N y un conjunto de arcos A.

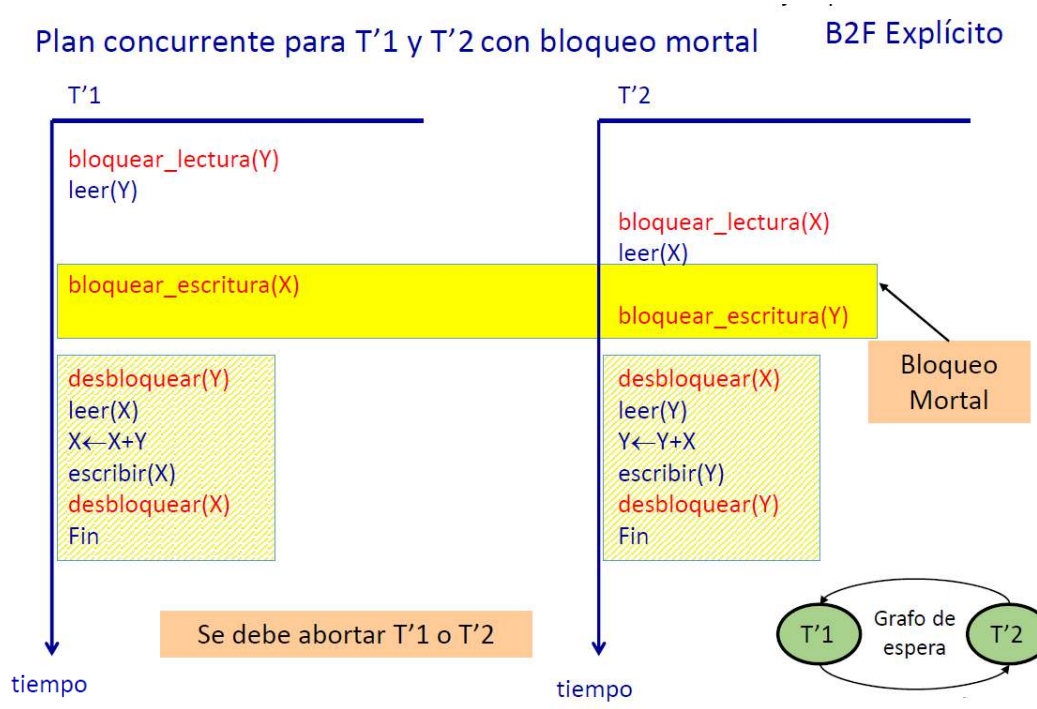
- ✓ El grafo contiene un **nodo** por cada transacción que se está ejecutando actualmente.
- ✓ El **arco** $a_{ij} = (T_i \rightarrow T_j)$ se crea si la transacción T_i está esperando bloquear un elemento X que está bloqueado por la transacción T_j (*uno de los dos bloqueos es exclusivo*). Cuando T_j librea el elemento X se borra el arco.

*Un plan concurrente, controlado por un protocolo de bloqueo, está en una **situación de bloqueo mortal** si su grafo de espera tiene un **ciclo**.*

Si el plan está en un estado de bloqueo mortal:

→ Abortar alguna de las transacciones que lo están provocando.

Ejemplo:



Operaciones para usar bloqueo de lectura/escritura

```
Bloquear_escritura (X):
B: IF bloqueo(X)='desbloqueado' (el elemento está desbloqueado)
    THEN  bloqueo(X) ← 'bloqueado para escritura'
          añadir T a la lista de transacciones del elemento X
     ELSE  WAIT (hasta que bloqueo(X)='desbloqueado' y el
                 gestor de bloqueos considere a T)
           GOTO B
     END IF.
```

```
Bloquear_lectura (X):
B: IF bloqueo(X) = 'desbloqueado' (el elemento está desbloqueado)
    THEN  bloqueo(X) ← 'bloqueado para lectura'
          nro_lecturas ← 1
          añadir T a la lista de transacciones del elemento X
     ELSE
          IF bloqueo(X) = 'bloqueado para lectura'
          THEN  nro_lecturas ← nro_lecturas + 1
                añadir T a la lista de transacciones del
                  elemento X
          ELSE WAIT (hasta que bloqueo(X) = 'desbloqueado' y el
                     gestor de bloqueos considere a T)
               GOTO B
          END IF
END IF.
```

```
Desbloquear (X):
IF bloqueo(X)='bloqueado para escritura'
THEN   bloqueo(X) ← 'desbloqueado'
       eliminar T de la lista de transacciones del elemento X
         --considerar alguna transacción que espera para bloquear X--
ELSE
    IF bloqueo(X) = 'bloqueado para lectura'
    THEN  nro_lecturas ← nro_lecturas - 1
          eliminar T de la lista de transacciones del elemento X
          IF nro_lecturas=0
          THEN bloqueo(X) ← 'desbloqueado'
               -- considerar alguna transacción que espera para
                          bloquear X --
          END IF
       END IF
END IF
```

5. Protocolos de ordenamiento por marcas de tiempo (OMT)

Los **protocolos de ordenamiento por marcas de tiempo** se basan en la idea de usar las marcas de tiempo de las transacciones para **controlar el orden de ejecución de las operaciones en conflicto** dentro de un plan.

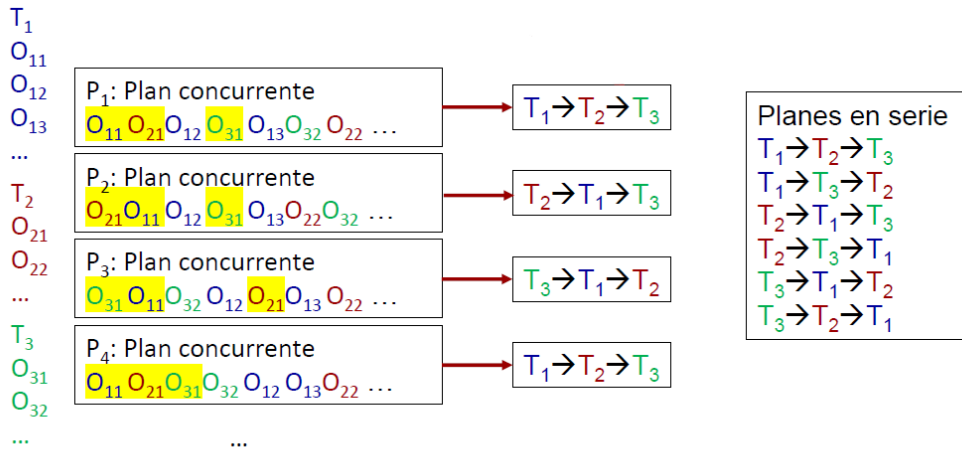
Marca de tiempo de una transacción: valor que el SGDB asigna a cada transacción, representa el instante de tiempo en que se inicia una transacción.

- ✓ Están ordenadas según el orden en el que las transacciones se inician en el sistema.
- ✓ Se pueden generar usando un contador entero que se va incrementando cada vez o usando el reloj del sistema.
- ✓ La marca de tiempo de una transacción **no cambia nunca**.

Plan en serie cronológico para un plan concurrente: **plan** en serie en el que las transacciones se procesan siguiendo el orden de sus marcas de tiempo en el plan concurrente.

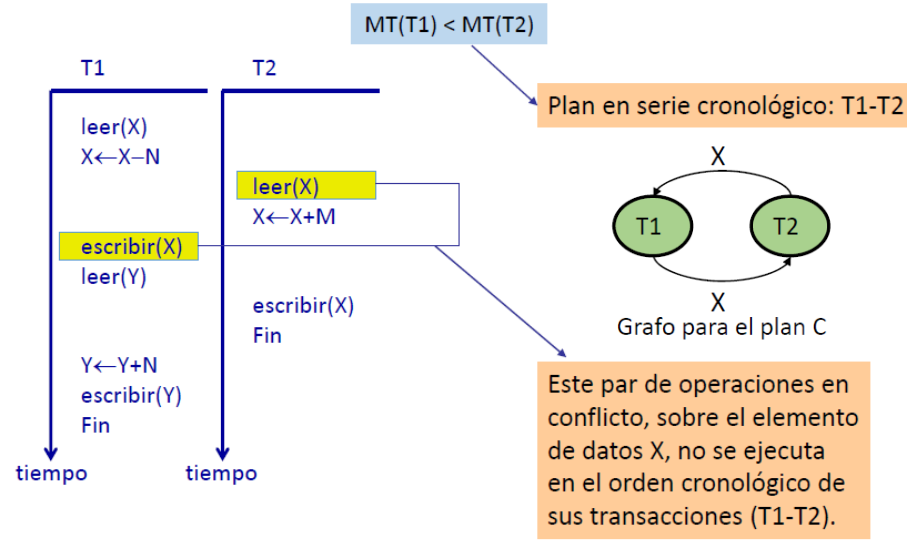
Un plan concurrente, admitido por el protocolo de ordenamiento por marcas de tiempo, es **equivalente por conflictos** a su plan en serie cronológico.

→ *Para cualquier par de operaciones en conflicto en el plan, se debe ejecutar primero la perteneciente a la transacción más vieja (con marca de tiempo menor).*



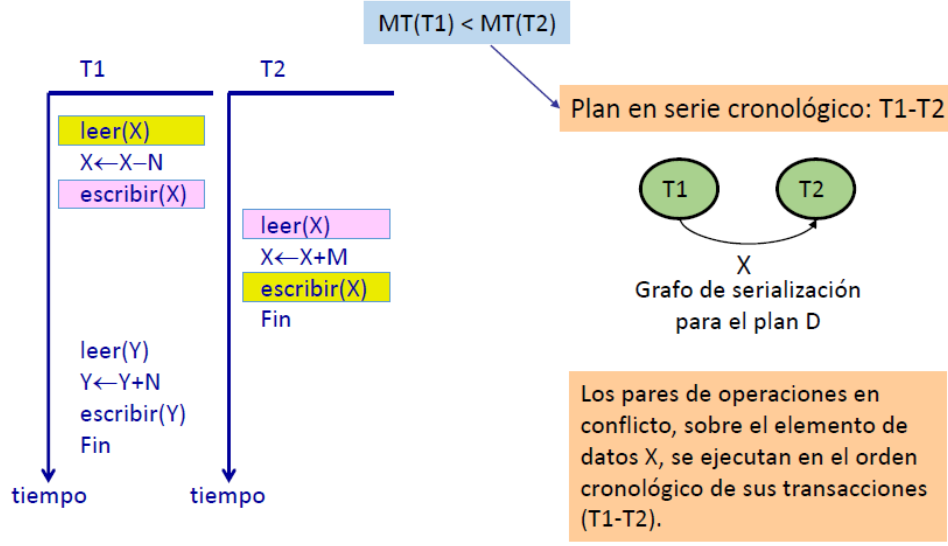
O_{ij}: Operación *j* de la transacción *i*

Como ya se vio anteriormente, el **plan concurrente C no es serializable por conflictos**. En el caso del protocolo OMT, este rechaza la transacción ya que el orden de las operaciones *leer(X)* de T_2 y *escribir(X)* de T_1 no coincide con el orden cronológico de las transacciones:



Plan C: no es admitido por el protocolo OMT

En cambio, **el plan D sí que es admitido por el protocolo OMT**, ya que en todos los pares de operaciones en conflicto, primero ocurre la operación de T_1 y luego la operación de T_2, el plan es equivalente por conflictos al plan en serie cronológico T_1-T_2



Plan D: admitido por el protocolo OMT

Información mantenida por el sistema

Marcas de tiempo de un elemento de datos X: información que el SGBD necesita guardar para controlar el orden de ejecución de las operaciones en conflicto.

- **MT_Lectura(X):** es la **mayor** de todas las marcas de tiempo de las transacciones que han leído el elemento X.
- **MT_Escritura(X):** es la **mayor** de todas las marcas de tiempo de las transacciones que han escrito con éxito el elemento X.

Funcionamiento interno del protocolo

a) Sea T' la transacción más joven que ha escrito un dato X actualizando la marca de escritura de X: $\mathrm{MT\_}escritura(\mathrm{X}) \leftarrow \mathrm{MT(T')}$. Si una transacción T ejecuta después de esa escritura una operación de lectura de X, el protocolo OMT compara la marca de tiempo de esa transacción ($\mathrm{MT(T)}$) con la marca de tiempo de escritura del elemento de datos X ($\mathrm{MT\_}escritura(\mathrm{X})$):

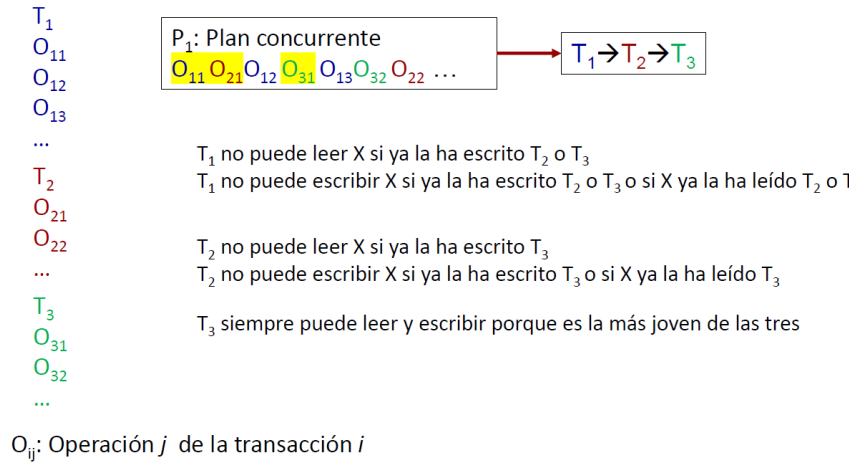
1. Si $\mathrm{MT(T)} < \mathrm{MT\_}escritura(\mathrm{X}) \rightarrow \mathrm{MT(T)} < \mathrm{MT(T')}$:
   - T es cronológicamente anterior a T'.
   - La operación de lectura de T va a entrar en conflicto con la operación de escritura de la transacción T'.
   - El orden de esas dos operaciones es el inverso al orden cronológico (la escritura de X por T' sería anterior a la lectura de X por T).
   - La lectura de X es rechazada y la transacción T es anulada por el SGBD.
2. Si $\mathrm{MT(T)} > \mathrm{MT\_}escritura(\mathrm{X}) \rightarrow \mathrm{MT(T)} > \mathrm{MT(T')}$:
   - T es cronológicamente posterior a T'.
   - La operación de lectura de T va a entrar en conflicto con la operación de escritura de la transacción T'.
   - El orden de estas dos operaciones es el mismo que en el orden cronológico (la escritura de X por T' es anterior a la lectura de X).
   - La lectura de X es aceptada.
3. Si $\mathrm{MT(T)} = \mathrm{MT\_}escritura(\mathrm{X}) \rightarrow \mathrm{MT(T)} = \mathrm{MT(T')}$: T y T' son la misma transacción y no hay par de operaciones en conflicto. La lectura de X es aceptada.

b) Sea T' la transacción más joven que ha leído un dato X actualizando la marca de lectura de X: MT_$lectura$(X) ← MT(T'). Y sea T'' la transacción más joven que ha escrito X actualizando la marca de escritura de X: MT_$escritura$(X) ← MT(T''). Si una transacción T ejecuta después de esa lectura y de esa escritura una operación de escritura de X, el protocolo OMT compara la marca de tiempo de esa transacción (MT(T)) con las marcas de tiempo de escritura y de lectura del elemento de datos X (MT_$escritura$(X) y MT_$lectura$(X)):

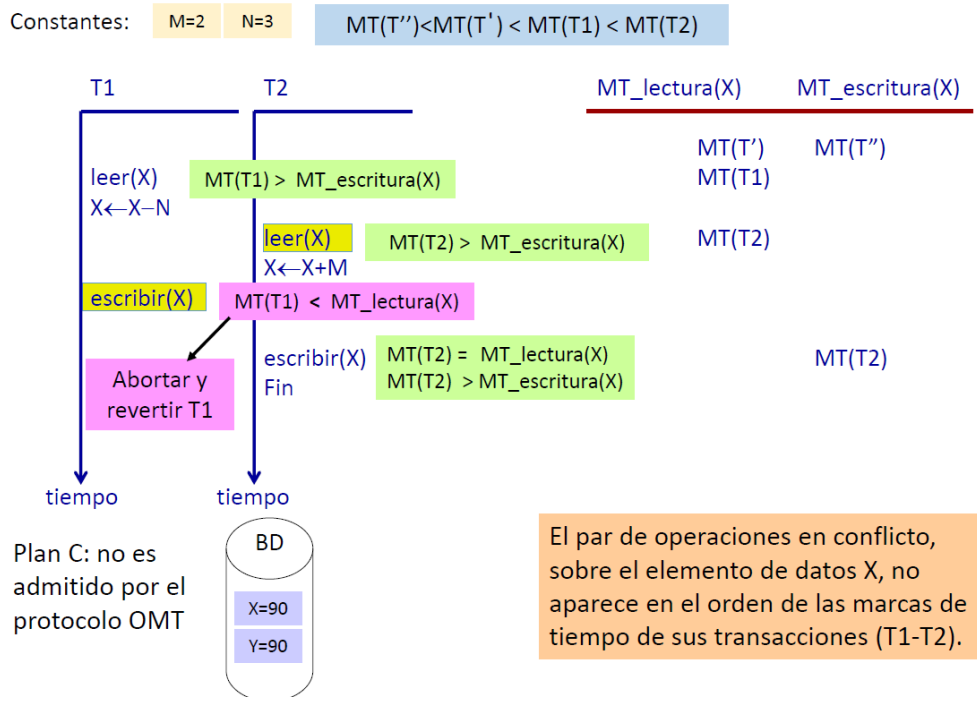
1. Si MT(T) < MT_$lectura$(X) o MT(T) < $MT\_escritura$(X) → MT(T) < MT(T') o MT(T) < MT(T''):
   - T es cronológicamente anterior a T' o a T''.
   - La operación de escritura de T va a entrar en conflicto con la operación de lectura de la transacción T' o con la operación de escritura de T''.
   - El orden de al menos un par de esas operaciones en conflicto es el inverso al orden cronológico.
   - La escritura de X es rechazada y la transacción T es anulada por el SGBD.

2. Si MT(T) ≥ MT_$lectura(X)$ y MT(T) ≥ MT_$escritura$(X) → MT(T) ≥ MT(T') y MT(T) ≥ MT(T'') (*con al menos una de las desigualdades estricta*):
   - T es cronológicamente posterior a T' y a T''.
   - La operación de escritura de T va a entrar en conflicto con la operación de lectura de la transacción T' o con la operación de escritura de T'' o con ambas.
   - El orden de esos dos pares de operaciones en conflicto es el mismo que en el orden cronológico.
   - La escritura de X es aceptada por el SGBD.

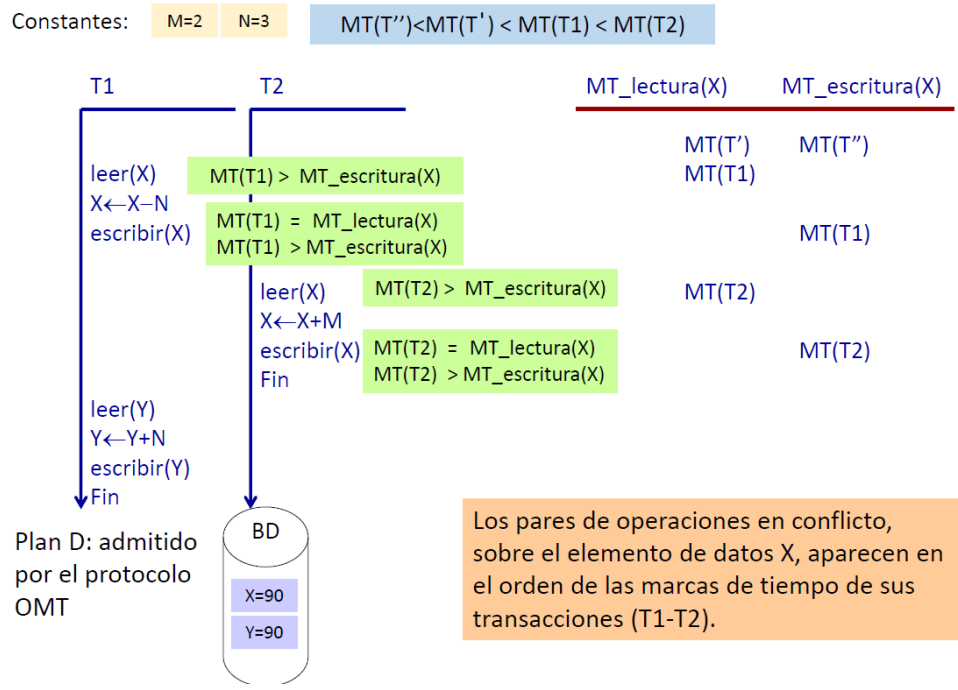
3. Si MT(T) = MT_$lectura$(X) y MT(T) = MT_$escritura$(X) → MT(T) = MT(T') y MT(T) = MT(T''): T, T' y T'' son la misma transacción y no hay pares de operaciones en conflicto. La escritura de X es aceptada por el SGBD.



Nuevamente, comprobamos que el **plan C** no es admitido por el protocolo OMT. A continuación se muestra la traza del protocolo:



Y la siguiente traza es la del **plan D**, que sí que es admitido por el protocolo OMT:



No se han incluido los controles sobre las marcas de tiempo de Y dado que sobre el elemento Y no hay operaciones en conflicto.

CONCLUSIONES SOBRE EL PROTOCOLO OMT

- ✓ El algoritmo OMT detecta operaciones en conflicto que ocurren en **orden incorrecto** (con respecto al orden cronológico) y rechaza la operación más reciente de las dos (anula la transacción que la emitió).
- ✓ Puede provocar la **espera indefinida** si una transacción se aborta y se reinicia continuamente.
- ✓ No evita la anomalía de la la **lectura sucia**.

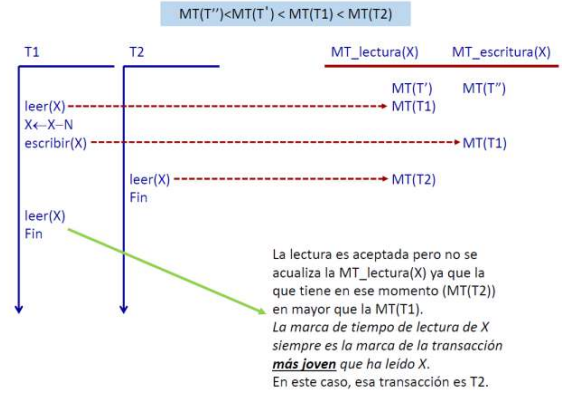
Resumen del funcionamiento del protocolo

La transacción T emite una operación leer(X):

a) Si $\mathrm{MT\_}escritura(\mathrm{X}) > \mathrm{MT(T)}$, entonces **abortar** y **revertir*** T.

b) Si $\mathrm{MT\_}escritura \leq \mathrm{MT(T)}$, entonces **ejecutar** la operación $leer(\mathrm{X})$ y **asignar** a $\mathrm{MT\_}lectura(X)$ el mayor valor entre los valores $\mathrm{MT(T)}$ y $\mathrm{MT\_}lectura(\mathrm{X})$.

c) Si el elemento de datos X no existe, entonces la operación no se puede ejecutar.



La transacción T emite una operación escribir(X):

a) Si $\mathrm{MT\_}lectura(\mathrm{X}) > \mathrm{MT(T)}$ o $\mathrm{MT\_}escritura(X) > \mathrm{MT(T)}$ entonces **abortar** y **revertir*** T.

b) Si $\mathrm{MT\_lectura} \leq \mathrm{MT(T)}$ y $\mathrm{MT} \leq \mathrm{MT(T)}$, entonces **ejecutar** la operación $escribir(\mathrm{X})$ y **asignar** a $\mathrm{MT\_}escritura(\mathrm{X})$ el valor $\mathrm{MT(T)}$.

c) Si el elemento de datos X no existe, entonces **ejecutar** la operación $escribir(\mathrm{X})$ y **asignar** a $\mathrm{MT\_}escritura(\mathrm{X})$ el valor $\mathrm{MT(T)}$.

*T *se vuelve a introducir en el sistema con una nueva marca de tiempo*.

6. Protocolos multiversión

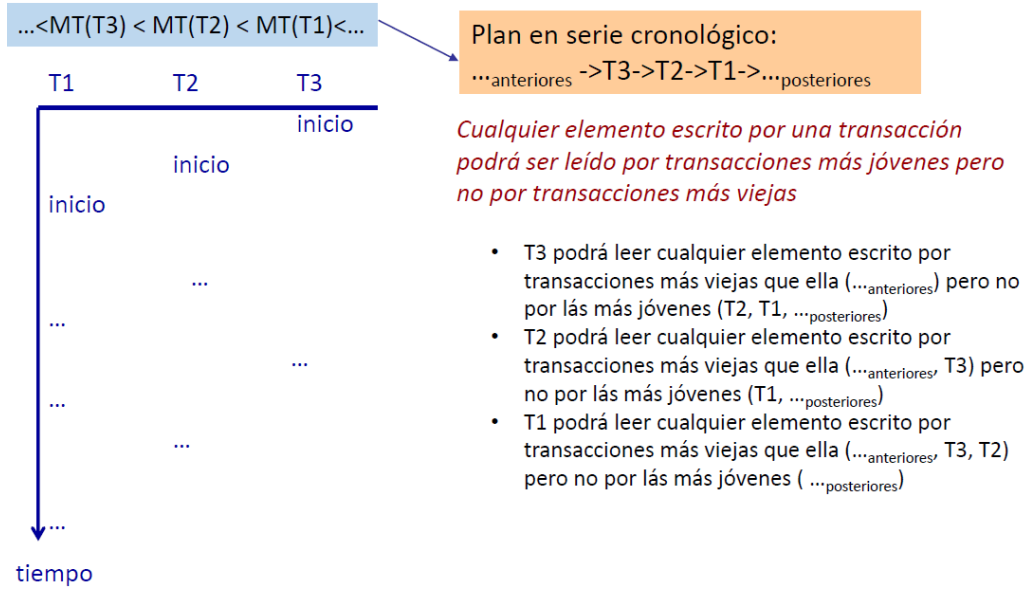
Los **protocolos multiversión** se basan en la idea de mantener, durante la ejecución del plan, **varias versiones** de los elementos de datos actualizados, con el objetivo de flexibilizar la concurrencia en las operaciones de lectura.

Plan en serie cronológico para un plan concurrente: plan en serie en el que las transacciones se procesan siguiendo el orden de sus marcas de tiempo en el plan concurrente.

Cuando el protocolo multiversión controla un plan concurrente, el plan ejecutado* es el **plan en serie cronológico.**

→ *Cuando una transacción solicita la lectura de un elemento de datos se elige una versión del elemento que asegure que se está ejecutando el plan en serie cronológico: se flexibiliza la concurrencia para las operaciones de lectura.*

**El uso de las versiones en las operaciones de lectura, hace que el plan ejecutado no sea el plan concurrente original, sino el plan cronológico.*



Información mantenida por el sistema

El sistema conserva para cada elemento de datos X varias versiones: $X_1, X_2, X_3, \dots, X_k$. Para cada versión X_i se conserva el valor de la versión y las marcas de tiempo:

- **MT_lectura(X_i):** la marca de tiempo **mayor** entre todas las marcas de tiempo de las transacciones que han leído con éxito la versión X_i.
- **MT_escritura(X_i):** es la marca de tiempo de la transacción que escribió el valor X_i (de la transacción que creó la versión).

Funcionamiento interno del protocolo

a) Cuando una transacción T ejecuta una operación de lectura del dato X, el protocolo MV busca la versión de X, sea X_i, que tiene mayor marca de tiempo de escritura entre todas las versiones que cumplen $MT\_escritura(X_i) \leq MT(T)$ y proporciona esa versión a T. Busca la versión de X más reciente creada por una transacción cronológicamente anterior a T.

   - Una operación de lectura siempre se satisface si existe el elemento a leer.

b) Cuando una transacción T ejecuta una operación de escritura del dato X, el protocolo MV busca la versión de X, sea X_i, que tiene mayor marca de tiempo de escritura entre todas las versiones que cumplen $MT\_escritura(X_i) \leq MT(T)$. Sea T' la transacción más joven que leyó esa versión de X ($MT\_lectura(X_i) = MT(T')$):

   a. Si $MT(T) < MT\_lectura(X_i) \rightarrow MT(T) < MT(T')$

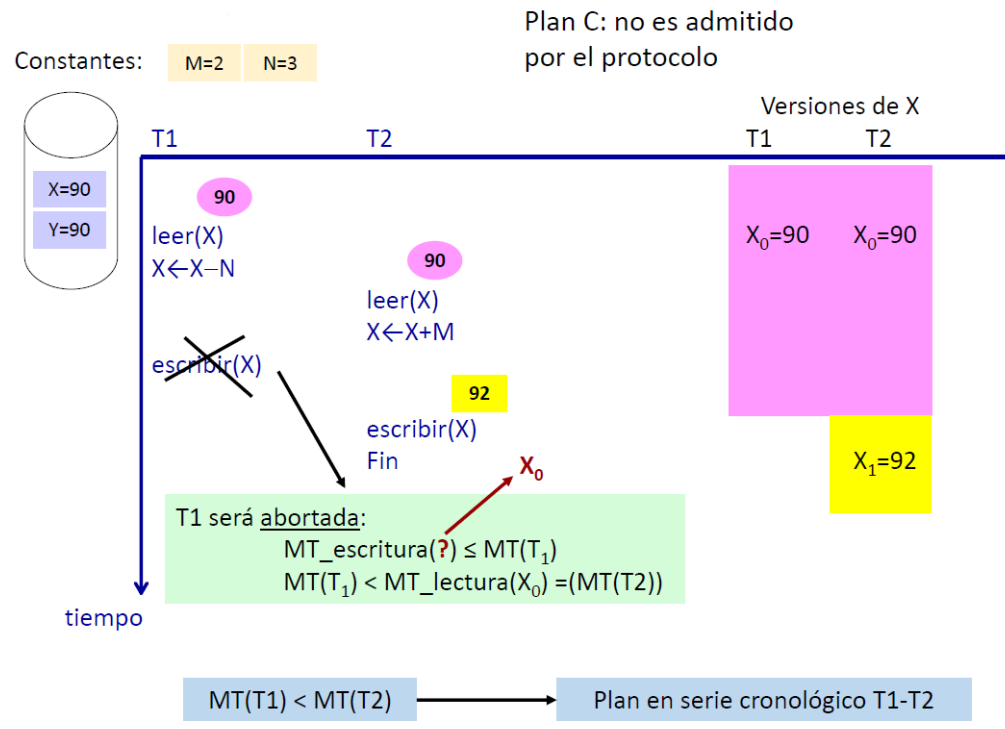
      - T es cronológicamente anterior a T'.
      - La operación de escritura de T va a entrar en conflicto con la operación de lectura de T'.
      - El orden de ese par de operaciones en conflicto es el inverso al orden cronológico (la escritura de X por T es posterior a la lectura de X por T').
      - La escritura de X es rechazada y la transacción T es anulada por el SGBD.

   b. Si $MT(T) > MT\_lectura(X_i) \rightarrow MT(T) > MT(T')$

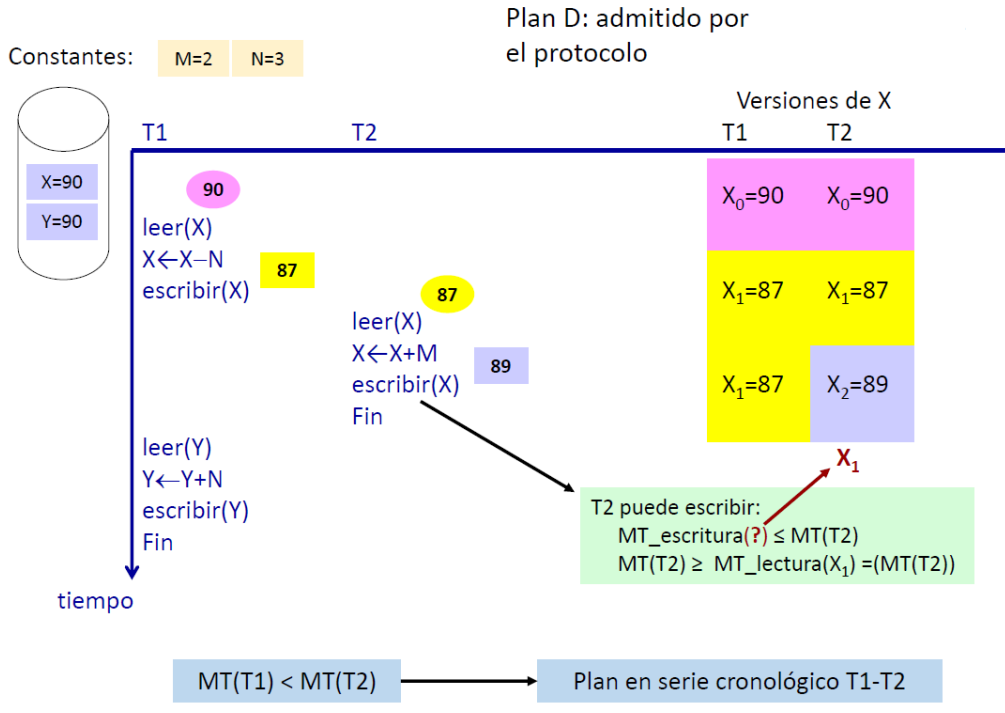
      - T es cronológicamente posterior a T'.
      - La operación de escritura de T va a entrar en conflicto con la operación de lectura de T'.
      - El orden de ese para de operaciones en conflicto es el mismo que en el orden cronológico (la escritura de X por T es posterior a la lectura de X por T').
      - La escritura de X es aceptada.

   c. Si $MT(T) = MT\_lectura(X_i) \rightarrow MT(T) = MT(T')$: T y T' son la misma transacción. La escritura es aceptada.

Si consideramos una vez más el **plan C** introducido al principio del tema, comprobamos que tampoco es admitido por este protocolo.



El **plan D**, en cambio, si que es admitido por el protocolo MV. En este plan concurrente, no se produce la anomalía de la "pérdida de actualizaciones", pero se podría haber producido la anomalía de la "lectura sucia" (punto 7 del tema).



CONCLUSIONES SOBRE EL PROTOCOLO MULTIVERSIÓN BASADO EN MARCAS DE TIEMPO

- ✓ Siempre se satisfacen las operaciones de lectura.
- ✓ El plan serie equivalente es el plan determinado por las marcas de tiempo de las transacciones (cronológico).
- ✓ Puede provocar el problema de la lectura sucia.
- ✓ Exige más espacio de almacenamiento que los otros protocolos.

Resumen del funcionamiento del protocolo

La transacción T emite una operación leer(X):

a) **Buscar** la versión X_i que tiene la marca de tiempo de escritura mayor entre todas las versiones de X que cumplen que $MT\_escritura(X_i) \leq MT(T)$, **devolver** a T el valor de X_i y **asignar** a $MT\_lectura(X_i)$ el mayor valor entre los valores $MT(T)$ y $MT\_lectura(X_i)$.

b) Si el elemento de datos X no existe, entonces la operación no puede ejecutarse.

*Una operación de lectura **siempre se satisface** si el dato a leer existe.*

La transacción T emite una operación escribir(X):

a) Si la versión X_i es la versión que tiene la marca de tiempo de escritura mayor ente todas las versiones de X tales que $MT\_escritura(X_i) \leq MT(T)$ y $MT\_lectura(X_i) > MT(T)$, entonces **abortar** T.

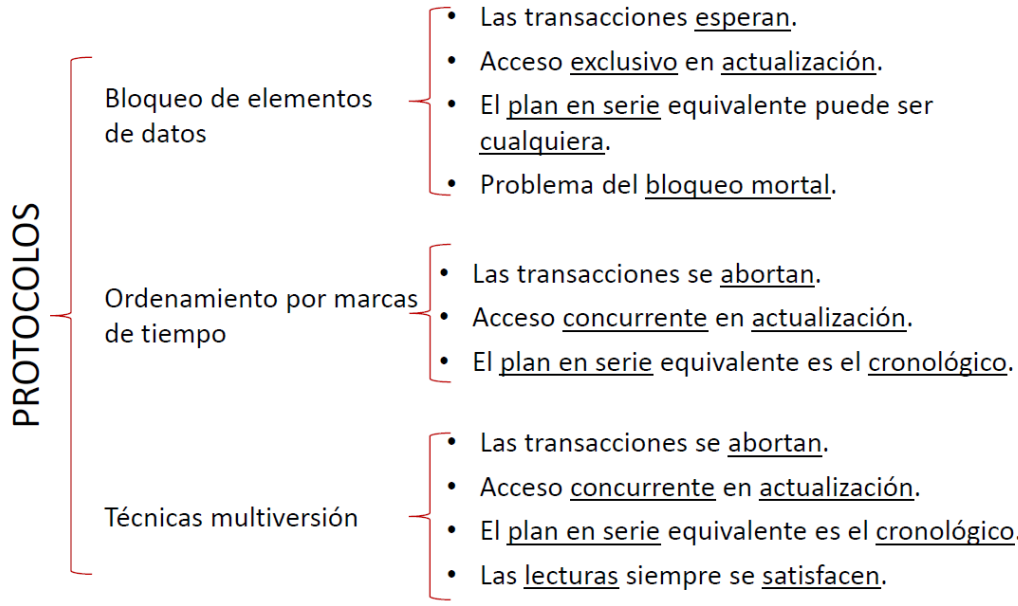
b) Si no se cumple la condición de a), entonces **crear** una nueva versión X_j y asignar a $MT\_lectura(X_j)$ y a $MT\_escritura(X_j)$ el valor $MT(T)$.

La transacción T sólo puede generar una nueva versión de un dato, si la versión anterior de ese dato no ha sido leída por una transacción posterior a T.

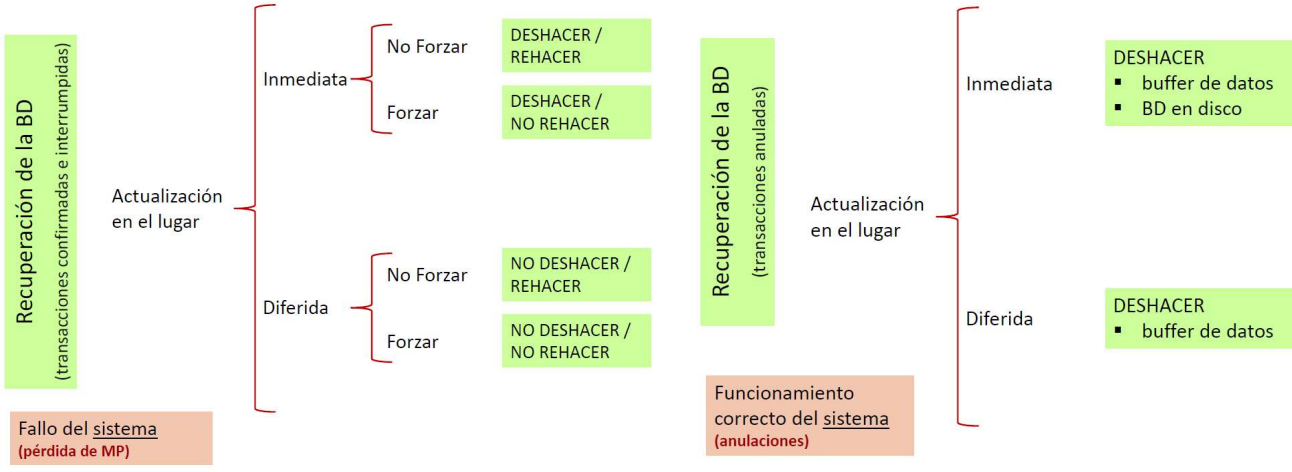
No se aborta necesariamente la transacción más reciente (más joven).

Si un elemento de datos existe, siempre tiene una versión con marca de lectura y marca de escritura.

Resumen general de los tres protocolos



7. Concurrencia y recuperación



RECORDAMOS (TEMA 3)

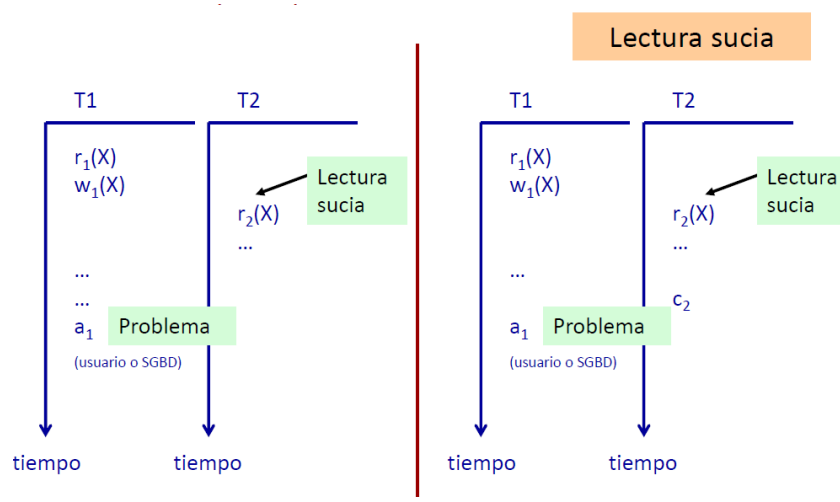
→ Cuando una transacción T es **anulada** (usuario o SGBD) o **interrumpida** por algún motivo, el SGBD debe deshacer sus efectos de forma que el resto de transacciones se sigan ejecutando como si T no hubiese ocurrido.

→ El procedimiento para realizar la recuperación dependerá de la estrategia de actualización de la BD seguida por el SGBD (inmediata o diferida) y del protocolo para el control de la concurrencia (control de la lectura sucia).

Anomalía de la "lectura sucia"

La **lectura sucia** se da cuando una transacción T_2 lee un dato actualizado por T_1, que todavía no ha sido confirmada.

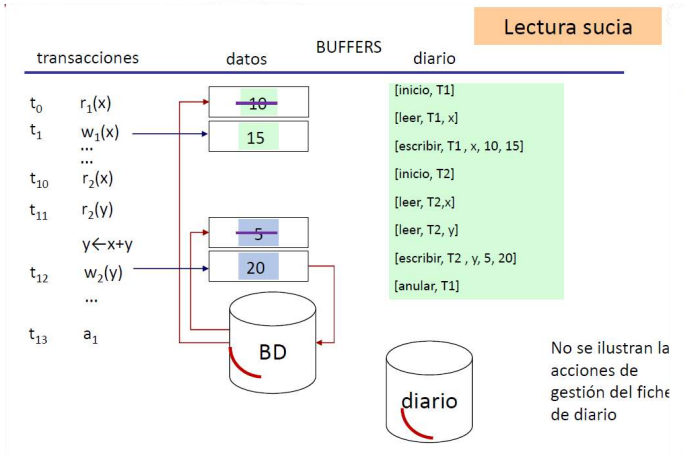
Es una **pseudoanomalía**: la ejecución concurrente es correcta, el plan es serializable. La lectura sucia se convierte en un problema cuando la transacción de la que se ha hecho la lectura sucia (T_1) es anulada o interrumpida por un fallo.



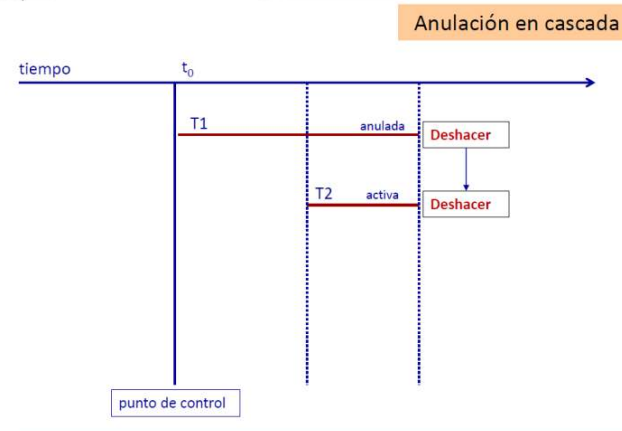
Anomalía de la lectura sucia: T2 lee un dato actualizado por T1 que todavía no ha sido confirmada

Cuando una transacción T es anulada o interrumpida, si el protocolo de control de la concurrencia no evita la anomalía de lectura sucia, deberán **anularse en cascada** todas aquellas transacciones que hayan hecho una lectura sucia de T.

Anulación en cascada: una transacción debe ser anulada porque ha leído (lectura sucia) un elemento de datos de una transacción que falla, ya sea por anulación o interrupción (en este último caso la anulación se realizará en la recuperación).



Actualización de la BD: en el lugar + inmediata + no forzar



T2 ha leído (**lectura sucia**) un elemento de datos actualizado por una transacción que ha sido anulada: T2 debe ser anulada sin haber finalizado.

Cuando una transacción es anulada (o interrumpida), si el protocolo de control de la concurrencia controla (**evita**) la lectura sucia, no puede producirse el problema de la anulación en cascada:

→ *Las entradas del diario de tipo [leer, T, X] no son necesarias*.

Planes y recuperabilidad

- ✓ Las transacciones pueden ser anuladas (usuario o SGBD) o interrumpidas por un fallo.
- ✓ Los protocolos de control de la concurrencia pueden anular transacciones.
- ✓ Si el protocolo no evita la lectura sucia, la anulación de una transacción puede provocas anulaciones en cascada de otras transacciones.

Objetivo: caracterizar planes de ejecución concurrentes para flexibilizar la recuperación de transacciones falladas (anuladas o interrumpidas).

→ Planes recuperables

→ Planes sin anulación en cascada

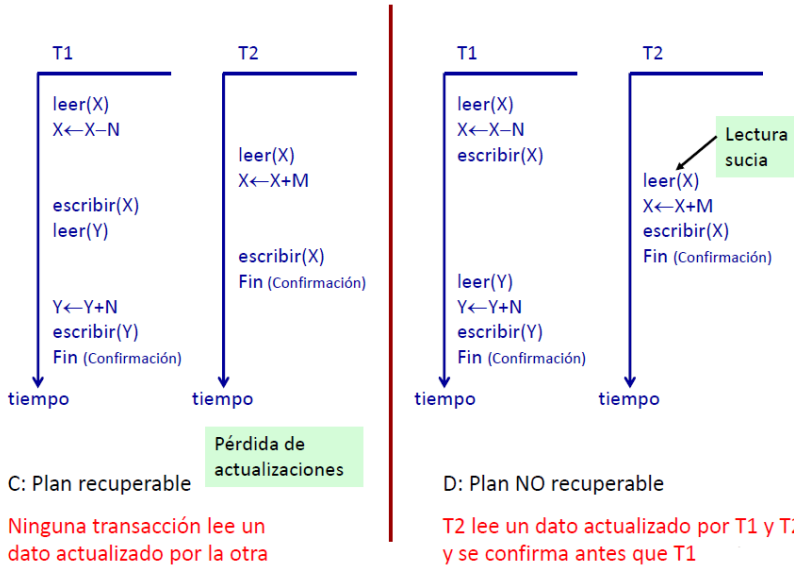
→ Planes estrictos

Planes y recuperabilidad: planes recuperables

Plan recuperable: una transacción T no se puede confirmar antes de que se confirmen todas las transacciones que han actualizado (escrito) un elemento de datos leído por T.

En un plan recuperable una transacción confirmada nunca se deshace

→ No se evita la anomalía de la **lectura sucia**



C no es un plan serializable

D es un plan serializable

Aunque D no da problemas, no es un plan recuperable: si T_1 fuese anulada habría que anular en cascada T_2, que ya ha sido confirmada.

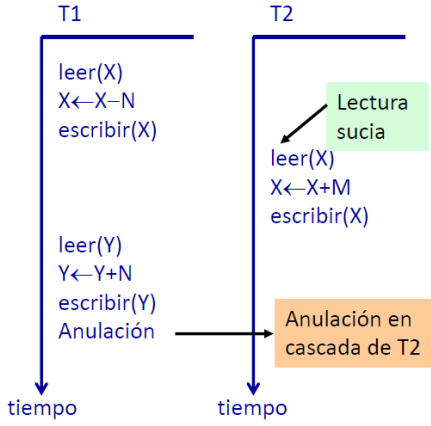
En un **plan recuperable** una transacción confirmada nunca debe deshacerse pero es posible que aparezca la lectura sucia y el problema de la anulación en cascada.

Anulación en cascada en planes recuperables: una transacción que **todavía no ha finalizado** debe ser anulada porque ha leído un elemento de datos de una transacción que ha fallado.

> *La **anulación en cascada** puede consumir muchos recursos porque la anulación de una transacción puede provocar muchas anulaciones en cadena.*

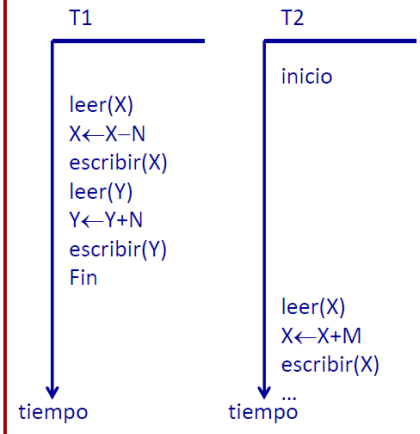
Planes y recuperabilidad: planes "sin anulación en cascada"

Plan "sin anulación en cascada": las transacciones del plan sólo pueden leer elementos de datos actualizados (escritos) por transacciones confirmadas → Se evita la anomalía de la **lectura sucia**.



G: Plan con anulación en cascada

T2 lee un dato actualizado por T1, pero no se confirma antes de que se confirme T1



H: Plan "sin anulació en cascada"

T2 lee un dato actualizado por T1, pero no se confirma antes de que se confirme T1

G es un plan serializable y recuperable, con lectura sucia.

Se produciría la anulación en cascada.

El plan H es un plan sin anulación en cascada.

Un plan sin anulación en cascada no tiene por qué ser un plan en serie.

Plan "sin anulación en cascada"	➡	No hay lectura sucia
Plan "sin anulación en cascada"	➡	Plan "recuperable"

P: $\dots w_1(x), \dots, w_2(x), a_1 \dots$

P es "sin anulación en cascada"

- ✓ Aunque no se da la lectura sucia (y por tanto la anulación en cascada), si T_1 se anula, la recuperación es más compleja: si se aplicara el *valor_antes* de la actualización de T_1 se perdería la actualización de T_2.
- ✓ La recuperación obliga al SGBD a analizar si ha habido actualizaciones posteriores del mismo elemento de datos.

Planes y recuperabilidad: planes estrictos

Plan estricto: las transacciones del plan no pueden leer ni escribir un elemento de datos X hasta que no finalice (confirmación o anulación) la última transacción que escribió X.

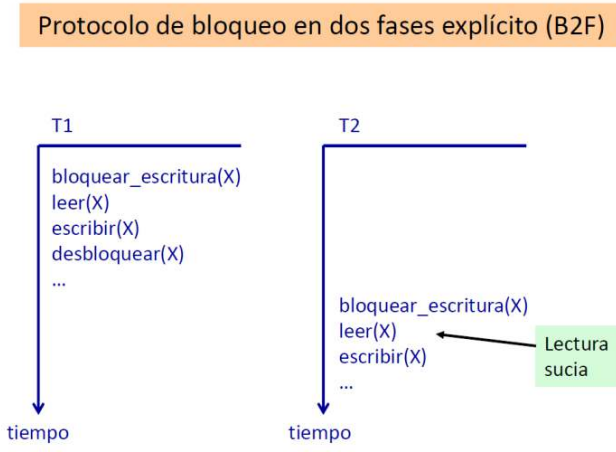
> *Los **planes estrictos** simplifican en proceso de recuperación: deshacer una operación escribir(X) de una transacción fallada significa restaurar (exclusivamente) el valor anterior (valor_antes) del elemento X.*

Si el protocolo de control de la concurrencia asegura planes estrictos, los algoritmos de recuperación (Tema 3) son válidos: aplican el procedimiento DESHACER a todas las transacciones interrumpidas por el fallo (en un entorno concurrente puede haber varias transacciones activas).

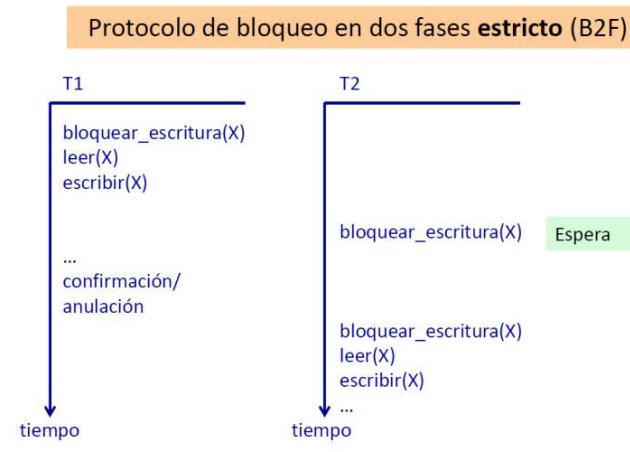
Análisis del tratamiento de la *lectura sucia* e los protocolos

Bloqueo en dos fases estricto: una transacción no libera ninguno de sus bloqueos exclusivos hasta que finaliza la transacción.

→ *Ninguna transacción puede leer ni escribir un elemento de datos escrito por otra transacción hasta que esta última finalice.*



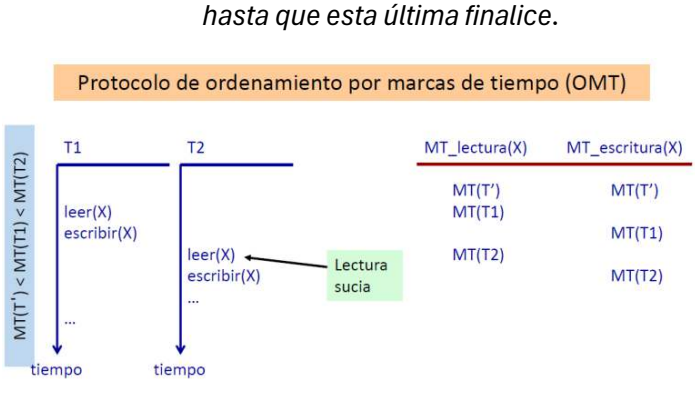
Se produce la anomalía de la "lectura sucia"



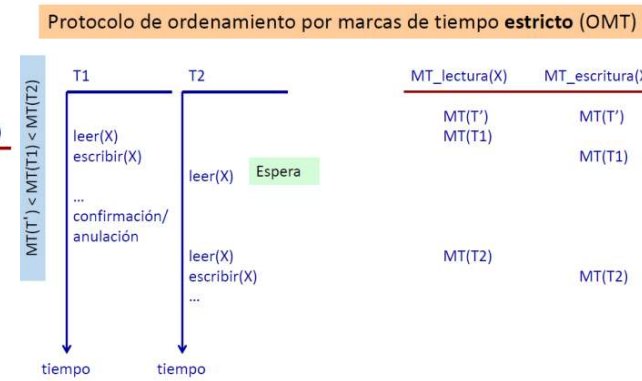
No se produce la anomalía de la "lectura sucia"

<u>Protocolo de ordamiento OMT estricto:</u> una transacción T que emite una operación $leer(\mathrm{X})$ o $escribir(\mathrm{X})$ tal que $\mathrm{MT(T)} > \mathrm{MT\_}escritura(\mathrm{X})$, sufre un retraso de su operación de lectura o escritura hasta que la transacción T' que escribió el valor de X ($\mathrm{MT\_}escritura(X) = \mathrm{MT(T')}$) finalice.

→ *Ninguna transacción puede leer ni escribir un elemento de datos escrito por una transacción hasta que esta última finalice.*



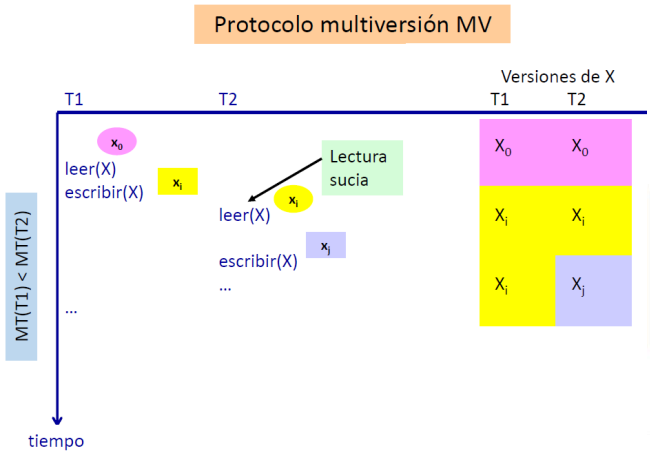
Se produce la anomalía de la "lectura sucia"



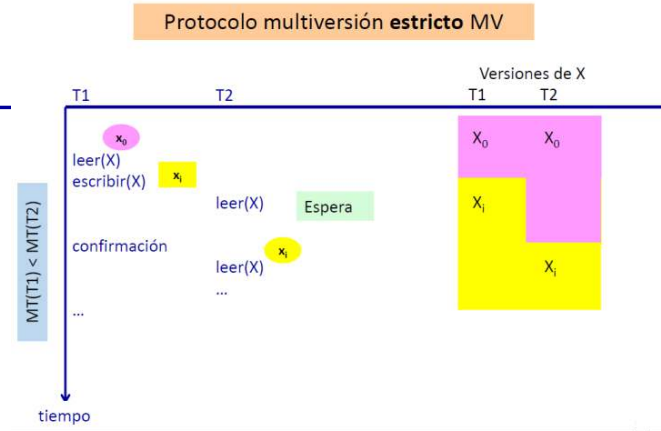
No se produce la anomalía de la "lectura sucia"

<u>Protocolo de multiversión estricto:</u> una transacción T que emite una operación $leer(\mathrm{X})$ tal que $\mathrm{MT(T)} > \mathrm{MT\_}escritura(\mathrm{X})$, sufre un retraso de su operación de lectura hasta que la transacción T' que escribió el valor de X ($\mathrm{MT\_}escritura(\mathrm{X}) = \mathrm{MT(T')}$) finalice.

→ *Ninguna transacción puede leer un elemento de datos escrito por una transacción hasta que esta última finalice.*



Se produce la anomalía de la "lectura sucia"



No se produce la anomalía de la "lectura sucia"