Concurrencia y Recuperabilidad

Paradigma Optimista

Lic. Andrea Manna

DEPARTAMENTO
DE COMPUTACION

2021

Introducción

Bibliografía y Definición

- Bibliografía: Database Systems. The Complete Book. Second Edition. Hector García-Molina, J.D. Ullman y Jennifer Widom (Capítulo 18)

Bibliografía y Definición

- Bibliografía: Database Systems. The Complete Book. Second Edition. Hector García-Molina, J.D. Ullman y Jennifer Widom (Capítulo 18)
- Estos métodos asumen que no ocurrirá un comportamiento no serializable y actúan para reparar el problema sólo cuando ocurre una violación aparente.

Bibliografía y Definición

- Bibliografía: Database Systems. The Complete Book. Second Edition. Hector García-Molina, J.D. Ullman y Jennifer Widom (Capítulo 18)
- Estos métodos asumen que no ocurrirá un comportamiento no serializable y actúan para reparar el problema sólo cuando ocurre una violación aparente.
- Métodos:
  - TimeStamping
  - TimeStamping Multiversion
  - Validación

Timestamping

Definición

- Cada transacción T tiene un único número llamado **timestamp**: $TS(T)$. Esta marca de tiempo es asignada en orden ascendente. Es decir si una transaccion T_1 ocurre, posee un timestamp $TS(T_1)$ y si luego una transaccion T_2 ocurre, posee un timestamp $TS(T_2)$ de manera tal que

$$TS(T_1) < TS(T_2)$$

Definición

- Cada transacción T tiene un único número llamado **timestamp**: $TS(T)$. Esta marca de tiempo es asignada en orden ascendente. Es decir si una transaccion T_1 ocurre, posee un timestamp $TS(T_1)$ y si luego una transaccion T_2 ocurre, posee un timestamp $TS(T_2)$ de manera tal que

$$TS(T_1) < TS(T_2)$$

- Para generar los timestamps se puede:
  1. Usar el reloj del sistema.
  2. El *scheduler* o planificador mantiene un contador: Una transacción nueva que comienza siempre tiene un número mayor que una que comenzó antes.

Definición

- Cada transacción T tiene un único número llamado **timestamp**: $TS(T)$. Esta marca de tiempo es asignada en orden ascendente. Es decir si una transaccion T_1 ocurre, posee un timestamp $TS(T_1)$ y si luego una transaccion T_2 ocurre, posee un timestamp $TS(T_2)$ de manera tal que

$$TS(T_1) < TS(T_2)$$

- Para generar los timestamps se puede:
  1. Usar el reloj del sistema.
  2. El *scheduler* o planificador mantiene un contador: Una transacción nueva que comienza siempre tiene un número mayor que una que comenzó antes.
- El planificador debe mantener una tabla de las transacciones y sus *timestamps*.

Definición

- Cada transacción T tiene un único número llamado **timestamp**: $TS(T)$. Esta marca de tiempo es asignada en orden ascendente. Es decir si una transaccion T_1 ocurre, posee un timestamp $TS(T_1)$ y si luego una transaccion T_2 ocurre, posee un timestamp $TS(T_2)$ de manera tal que

$$TS(T_1) < TS(T_2)$$

- Para generar los timestamps se puede:
  1. Usar el reloj del sistema.
  2. El *scheduler* o planificador mantiene un contador: Una transacción nueva que comienza siempre tiene un número mayor que una que comenzó antes.
- El planificador debe mantener una tabla de las transacciones y sus *timestamps*.
- El planificador maneja la ejecución concurrente de tal manera que los timestamps determinan el **orden de serialización**

Definición

Cada elemento de la base de datos, X, debe asociarse a dos *timestamp* y un bit extra.

- RT(X): tiempo de lectura, el timestamp más alto de una transacción que ha leído X

Definición

Cada elemento de la base de datos, X, debe asociarse a dos *timestamp* y un bit extra.

- RT(X): tiempo de lectura, el timestamp más alto de una transacción que ha leído X
- WT(X): tiempo de escritura, el timestamp más alto de una transacción ha escrito X

Definición

Cada elemento de la base de datos, X, debe asociarse a dos *timestamp* y un bit extra.

- RT(X): tiempo de lectura, el timestamp más alto de una transacción que ha leído X
- WT(X): tiempo de escritura, el timestamp más alto de una transacción ha escrito X
- C(X): bit de commit para X, es verdadero si y sólo si la transacción más reciente que escribió X ha realizado commit

Comportamiento Físicamente Irrealizable

- El planificador asume que el orden de llegada de las transacciones es el orden serial en que deberían parecer que se ejecutan.
- El planificador además de asignar timestamps y actualizar RT, WT y C para cada elemento de una transacción, debe verificar que cuando ocurre una lectura o escritura también podría haber ocurrido si cada transacción se hubiera realizado instantáneamente al momento del timestamp.

Definición

Si eso no ocurre entonces el comportamiento se denomina: **físicamente irrealizable**.

Comportamiento Físicamente Irrealizable

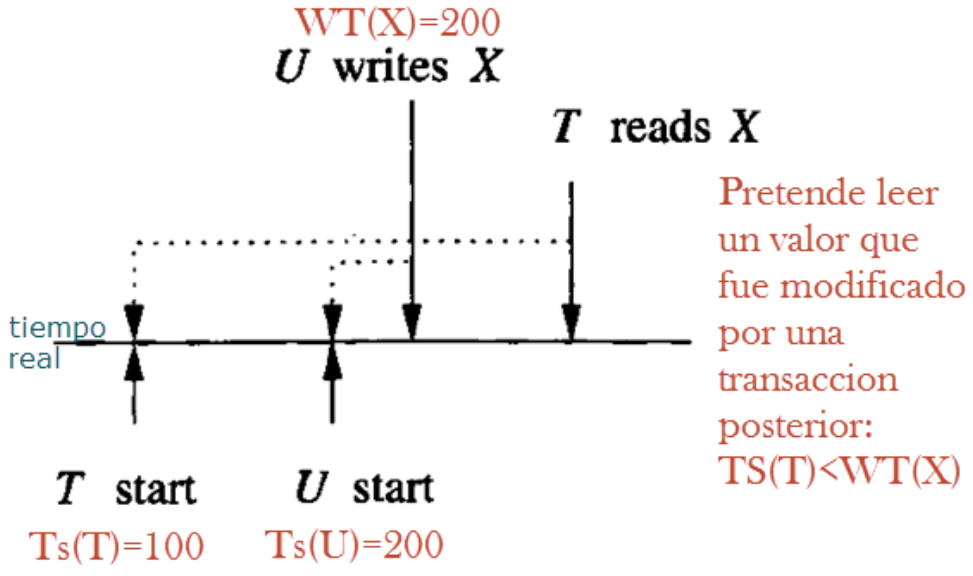
Read too Late

- TS(T) < WT(X)
- Una transacción T intenta leer X pero el valor de escritura indica que X fue escrito después de que teóricamente debería haberlo leído T.

Comportamiento Físicamente Irrealizable

Read too Late

- TS(T) < WT(X)
- Una transacción T intenta leer X pero el valor de escritura indica que X fue escrito después de que teóricamente debería haberlo leído T.



T debe abortar

Comportamiento Físicamente Irrealizable

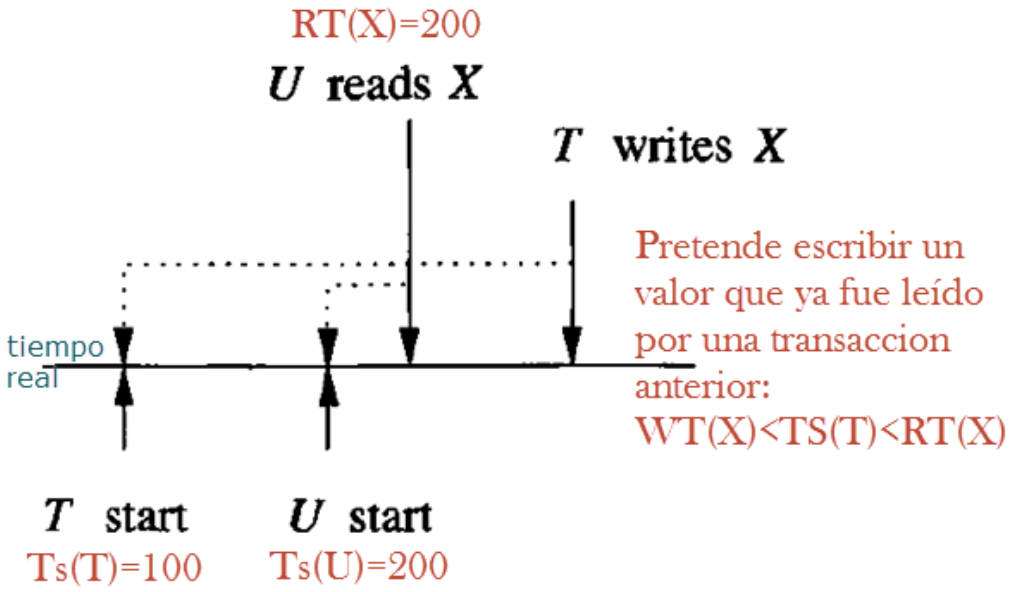
Write too Late

- WT(X) < TS(T) < RT(X).
- T intenta escribir pero el tiempo de lectura de X indica que alguna otra transacción debería haber leído el valor escrito por T (lee otro valor en su lugar).

Comportamiento Físicamente Irrealizable

Write too Late

- WT(X) < TS(T) < RT(X).
- T intenta escribir pero el tiempo de lectura de X indica que alguna otra transacción debería haber leído el valor escrito por T (lee otro valor en su lugar).



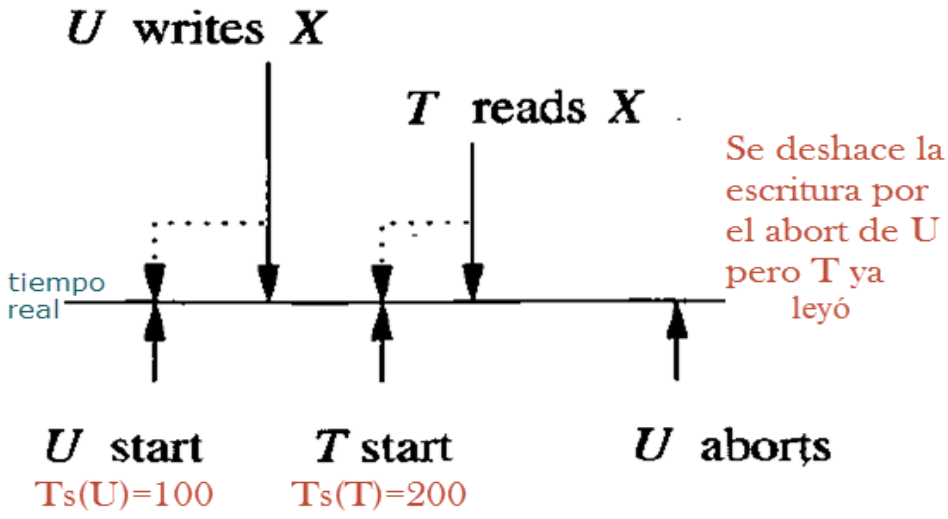
T debe abortar

Dirty data (Lectura Sucia)

Definición: Una lectura sucia ocurre cuando se le permite a una transacción la lectura de un elemento que ha sido modificado por otra transacción concurrente pero que todavía no ha sido cometida (commit).

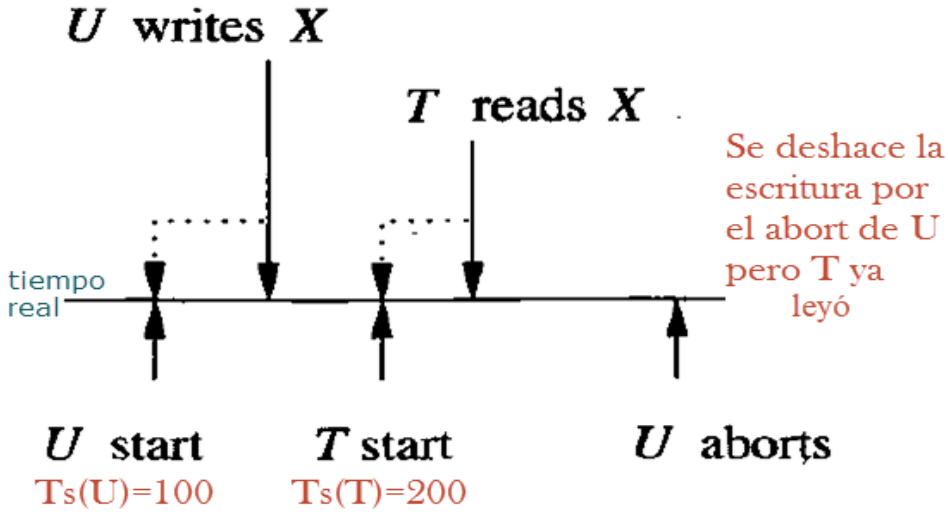
Dirty data (Lectura Sucia)

Definición: Una lectura sucia ocurre cuando se le permite a una transacción la lectura de un elemento que ha sido modificado por otra transacción concurrente pero que todavía no ha sido cometida (commit).



Dirty data (Lectura Sucia)

Definición: Una lectura sucia ocurre cuando se le permite a una transacción la lectura de un elemento que ha sido modificado por otra transacción concurrente pero que todavía no ha sido cometida (commit).



Dirty Data

Por lo tanto, aunque no hay nada **físicamente irrealizable** sobre la lectura de X por parte de T, es mejor retrasar la lectura hasta que U realice el commit o aborte

Dirty Data: Ejemplo

Transacción 1

```
/* Query 1 */
SELECT edad FROM usuarios WHERE id = 1;
/* leerá 20 */
```

Transacción 2

```
/* Consulta 2 */
UPDATE usuarios SET edad = 21 WHERE id = 1;
/* No se hace commit */
```

Transacción 1

```
/* Query 1 */
SELECT edad FROM usuarios WHERE id = 1;
/* leerá 21 */
```

Transacción 2

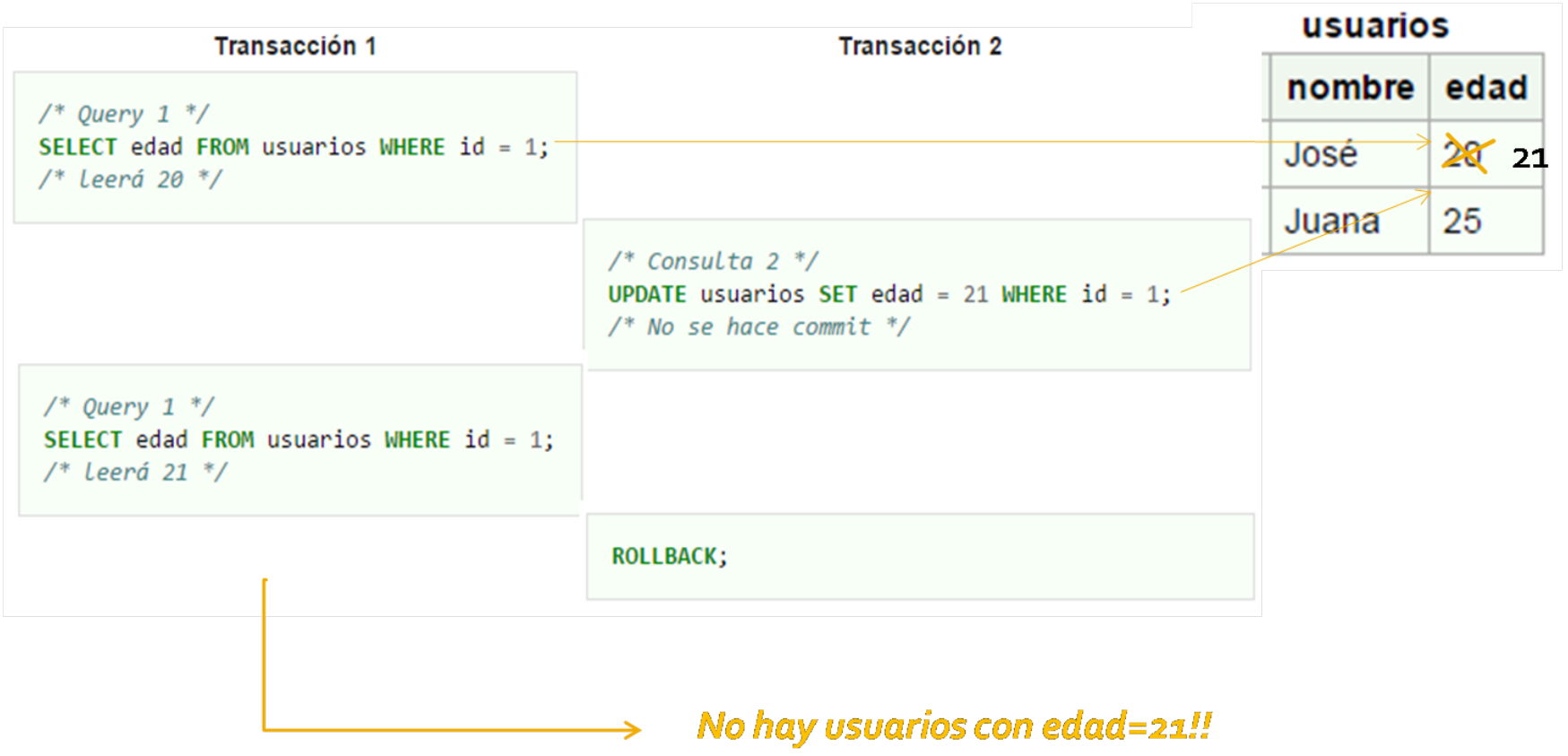
```
ROLLBACK;
```

usuarios

nombre	edad
José	20
Juana	25

Dirty Data: Ejemplo

Dirty Data: Ejemplo



Regla de escritura de Thomas

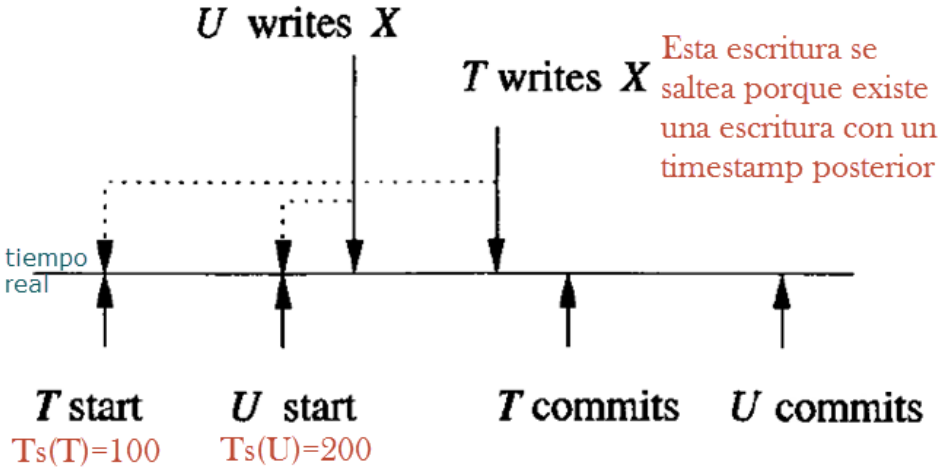
Thomas write rule

La escritura puede “saltearse” cuando ya existe una escritura de una transacción con un timestamp de mayor valor.
Es decir cuando WT(X) > TS(T)

Regla de escritura de Thomas

Thomas write rule

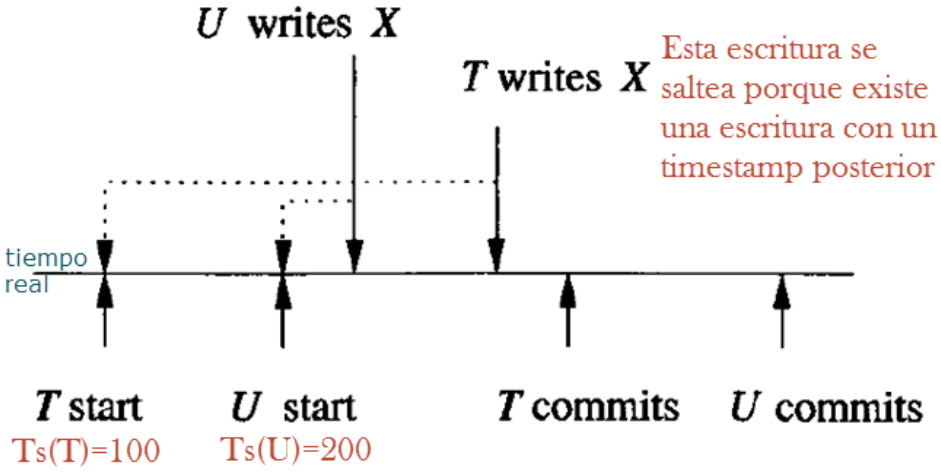
La escritura puede “saltearse” cuando ya existe una escritura de una transacción con un timestamp de mayor valor.
Es decir cuando WT(X) > TS(T)



Regla de escritura de Thomas

Thomas write rule

La escritura puede “saltearse” cuando ya existe una escritura de una transacción con un timestamp de mayor valor.
Es decir cuando WT(X) > TS(T)

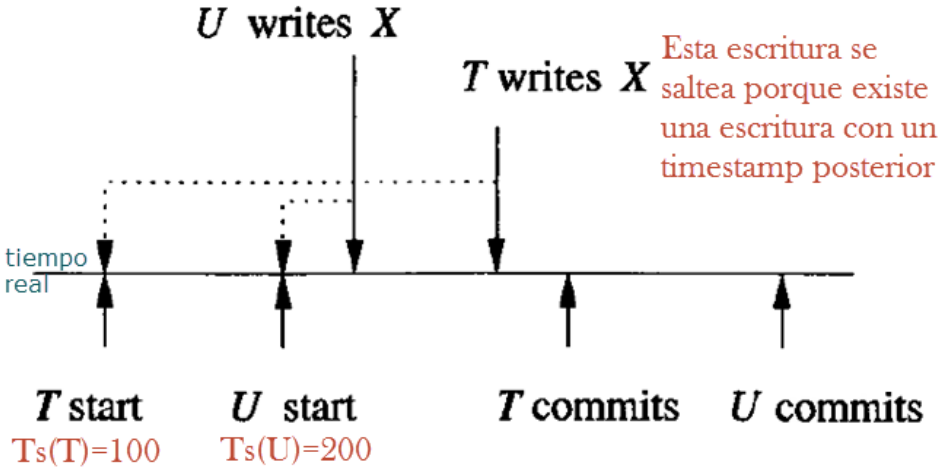


¿Que sucede si U realiza *abort* en vez de *commit*?

Regla de escritura de Thomas

Thomas write rule

La escritura puede “saltearse” cuando ya existe una escritura de una transacción con un timestamp de mayor valor.
Es decir cuando WT(X) > TS(T)



¿Que sucede si U realiza *abort* en vez de *commit*?

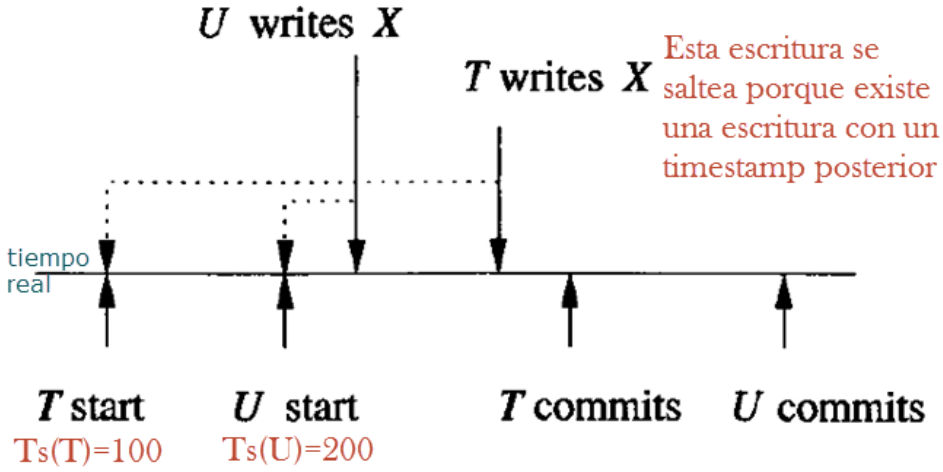
Problema si U aborta

Cuando una transacción U escribe un elemento X, la escritura es **tentativa** y **puede ser deshecha si U aborta**.
C(X) se pone falso y el planificador hace una copia de los valores de X y de WT(X) previos.

Regla de escritura de Thomas

Thomas write rule

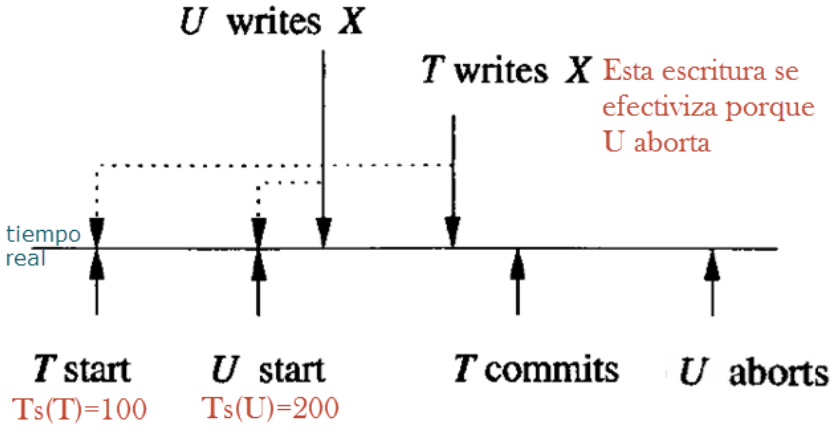
La escritura puede “saltearse” cuando ya existe una escritura de una transacción con un timestamp de mayor valor.
Es decir cuando WT(X) > TS(T)



¿Que sucede si U realiza *abort* en vez de *commit*?

Problema si U aborta

Cuando una transacción U escribe un elemento X, la escritura es **tentativa** y **puede ser deshecha si U aborta**. C(X) se pone falso y el planificador hace una copia de los valores de X y de WT(X) previos.



Reglas para el planificador

Ante la solicitud de una transacción T para una lectura o escritura, el planificador puede:

Reglas para el planificador

Ante la solicitud de una transacción T para una lectura o escritura, el planificador puede:

1. Conceder la solicitud

Reglas para el planificador

Ante la solicitud de una transacción T para una lectura o escritura, el planificador puede:

1. Conceder la solicitud
2. Abortar y reiniciar T con un nuevo timestamp (rollback)

Reglas para el planificador

Ante la solicitud de una transacción T para una lectura o escritura, el planificador puede:

1. Conceder la solicitud
2. Abortar y reiniciar T con un nuevo timestamp (rollback)
3. Demorar T y decidir luego si abortar o conceder la solicitud (si el requerimiento es una lectura que podría ser sucia).

Reglas para el planificador

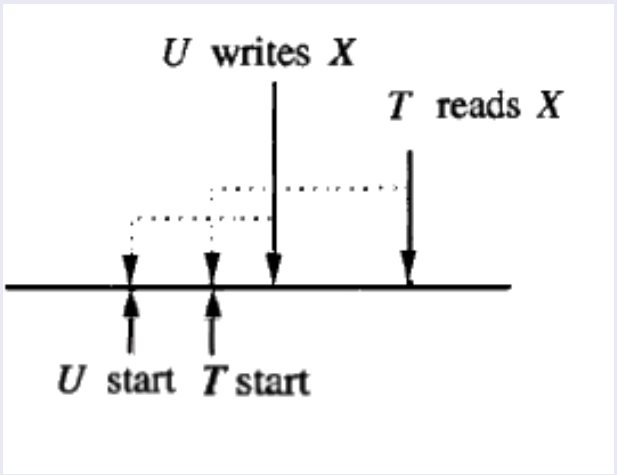
El planificador recibe una solicitud de lectura $r_t(X)$.

Caso 1: Si TS(T) >= WT(X) - es **físicamente realizable** es decir, no sucede read too late

Reglas para el planificador

El planificador recibe una solicitud de **lectura** $r_t(X)$.

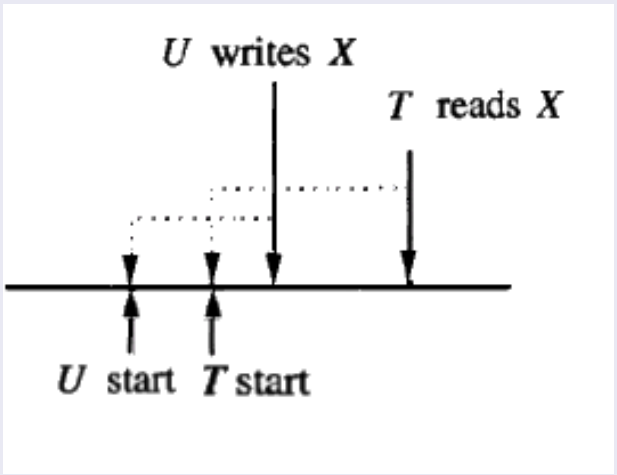
Caso 1: Si TS(T) >= WT(X) - es **físicamente realizable** es decir, no sucede read too late



Reglas para el planificador

El planificador recibe una solicitud de **lectura** $r_t(X)$.

Caso 1: Si TS(T) >= WT(X) - es **físicamente realizable** es decir, no sucede read too late

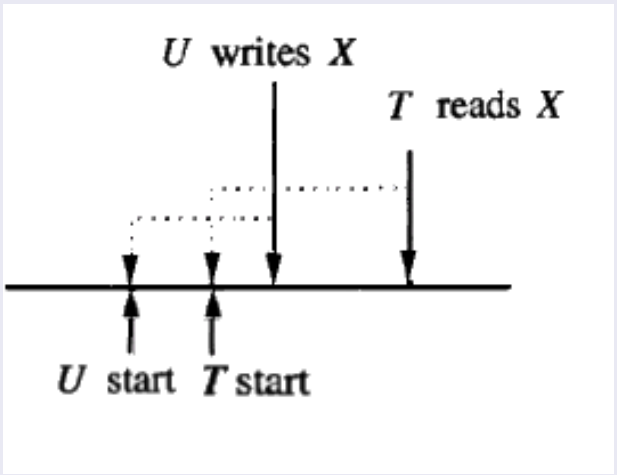


1. Si **C(X) es True**, conceder la solicitud. Si TS(T)>RT(X) hacer RT(X)=TS(T), de otro modo no cambiar RT(X).

Reglas para el planificador

El planificador recibe una solicitud de **lectura** $r_t(X)$.

Caso 1: Si TS(T) >= WT(X) - es **físicamente realizable** es decir, no sucede read too late



1. Si **C(X) es True**, conceder la solicitud. Si TS(T)>RT(X) hacer RT(X)=TS(T), de otro modo no cambiar RT(X).
2. Si **C(X) es False** demorar T hasta que C(X) sea verdadero o la transacción que escribió a X aborte.

Reglas para el planificador

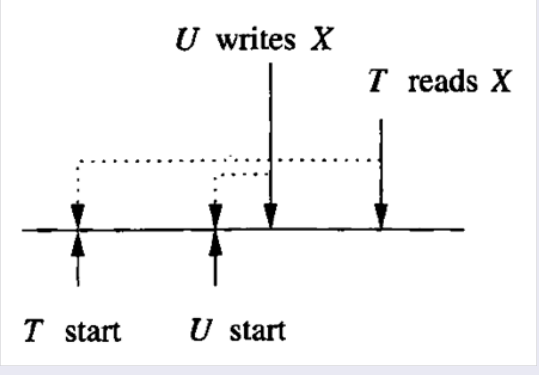
El planificador recibe una solicitud de lectura $r_t(X)$.

Caso 2: Si TS(T) < WT(X) - es **físicamente irrealizable** (***read too late***)

Reglas para el planificador

El planificador recibe una solicitud de **lectura** $r_t(X)$.

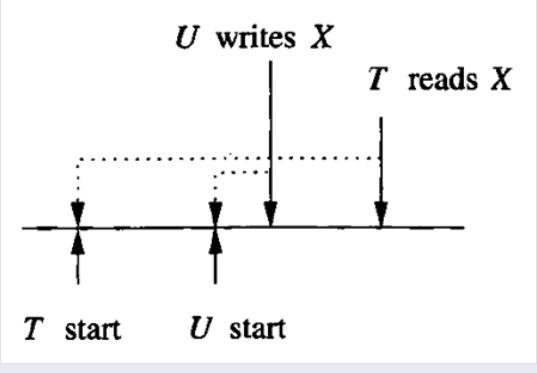
Caso 2: Si TS(T) < WT(X) - es **físicamente irrealizable** (***read too late***)



Reglas para el planificador

El planificador recibe una solicitud de **lectura** $r_t(X)$.

Caso 2: Si TS(T) < WT(X) - es **físicamente irrealizable** (***read too late***)



Se hace **Rollback T** (abortar y reiniciar con un nuevo timestamp).

Reglas para el planificador

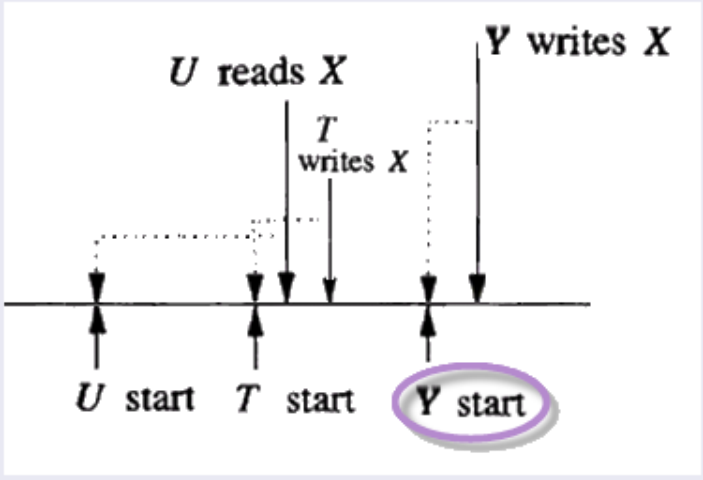
El planificador recibe una solicitud de **escritura** $w_t(X)$.

Caso 1: Si TS(T) >= RT(X) y TS(T) >= WT(X) - es **físicamente realizable**, es decir, no sucede write too late

Reglas para el planificador

El planificador recibe una solicitud de **escritura** $w_t(X)$.

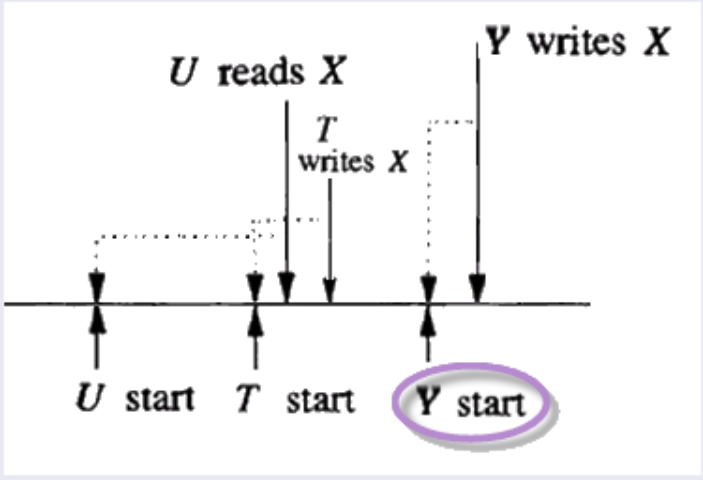
Caso 1: Si TS(T) >= RT(X) y TS(T) >= WT(X) - es **físicamente realizable**, es decir, no sucede write too late



Reglas para el planificador

El planificador recibe una solicitud de **escritura** $w_t(X)$.

Caso 1: Si TS(T) >= RT(X) y TS(T) >= WT(X) - es **físicamente realizable**, es decir, no sucede write too late



1. Escribir el nuevo valor para X
2. WT(X) := TS(T), o sea asignar nuevo WT a X.
3. C(X):= false, o sea poner en falso el bit de commit.

Reglas para el planificador

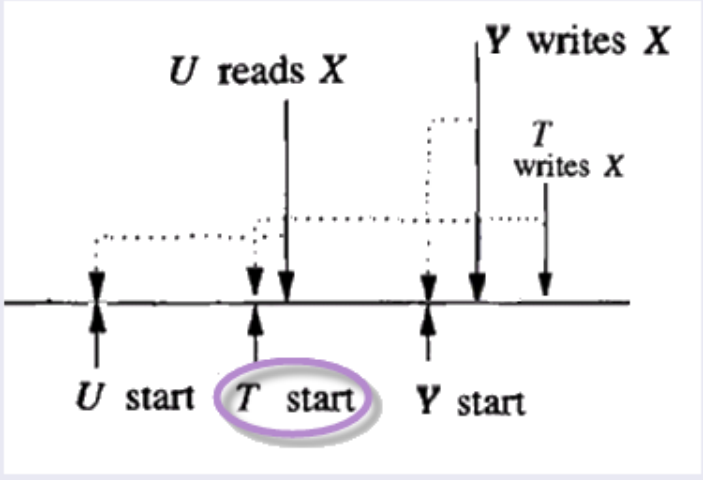
El planificador recibe una solicitud de **escritura** $w_t(X)$.

Caso 2: Si $TS(T) >= RT(X)$ pero $TS(T) < WT(X)$ - es **físicamente realizable**, pero ya hay un valor posterior en X.

Reglas para el planificador

El planificador recibe una solicitud de **escritura** $w_t(X)$.

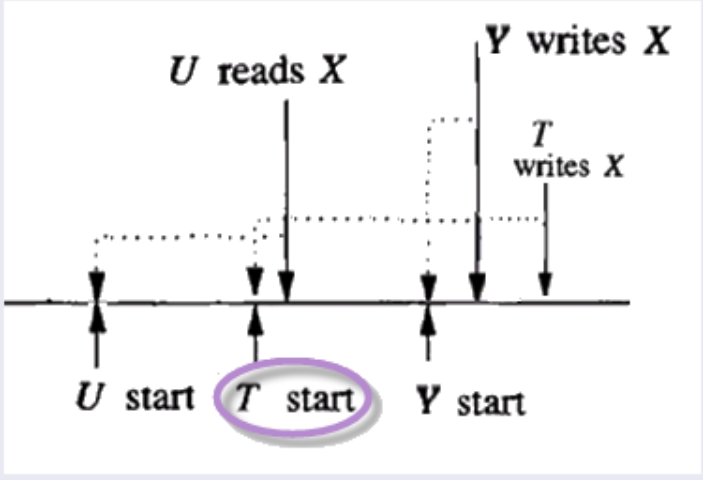
Caso 2: Si $TS(T) >= RT(X)$ pero $TS(T) < WT(X)$ - es **físicamente realizable**, pero ya hay un valor posterior en X.



Reglas para el planificador

El planificador recibe una solicitud de **escritura** $w_t(X)$.

Caso 2: Si $TS(T) >= RT(X)$ pero $TS(T) < WT(X)$ - es **físicamente realizable**, pero ya hay un valor posterior en X.



1. Si C(X) es true, ignora la escritura.
2. Si C(X) es falso demorar T hasta que C(X) sea verdadero o la transacción que escribió a X aborte

Reglas para el planificador

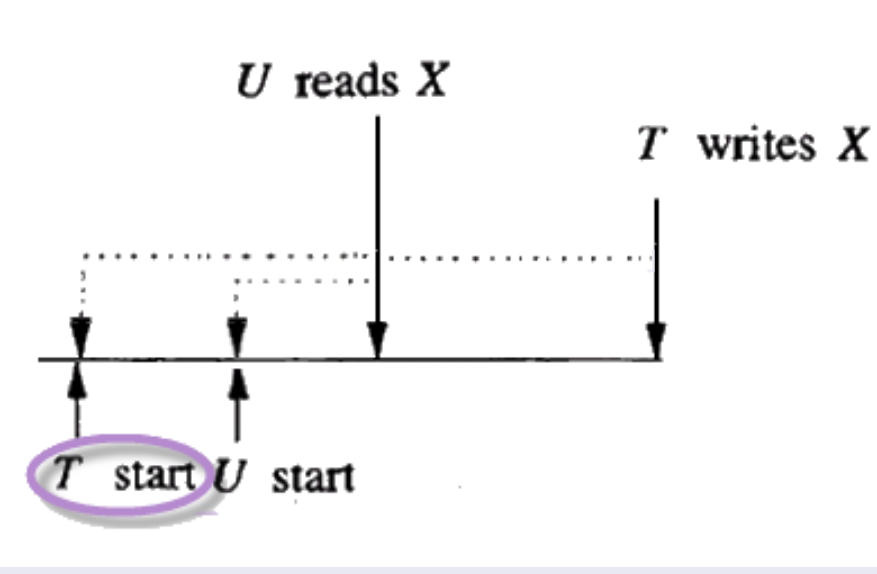
El planificador recibe una solicitud de escritura $w_t(X)$.

Caso 3: Si TS(T) < RT(X) - es **físicamente irrealizable**, es decir, write too late.

Reglas para el planificador

El planificador recibe una solicitud de **escritura** $w_t(X)$.

Caso 3: Si TS(T) < RT(X) - es **físicamente irrealizable**, es decir, write too late.



Se hace **Rollback T** (abortar y reiniciar con un nuevo timestamp).

Reglas para el planificador

El planificador recibe una solicitud de commit $C(T)$.

Para cada uno de los elementos X escritos por T se hace:

Reglas para el planificador

El planificador recibe una solicitud de commit $C(T)$.

Para cada uno de los elementos X escritos por T se hace:

- C(X) := true.
- Se permite proseguir a las transacciones que esperan a que X sea committed

Reglas para el planificador

El planificador recibe una solicitud de abort o rollback A(T) o R(T).

Cada transacción que estaba esperando por un elemento X que T escribió debe repetir el intento de lectura o escritura y verificar si ahora el intento es legal

Reglas para el planificador: Ejemplo

Supongamos transacciones T_1, T_2 y T_3 con los Timestamp como muestra la figura y suponer que el *commit* se realiza inmediatamente después del ultimo *write*.

T_1	T_2	T_3	A	B	C
200	150	175			
$r_1(B)$;					
	$r_2(A)$;				
		$r_3(C)$;			
$w_1(B)$;					
$w_1(A)$;					
	$w_2(C)$;				
		$w_3(A)$;			

Reglas para el planificador: Ejemplo

Atención

Suponemos que inmediatamente luego de la ultima operación, se realiza el commit

T_1	T_2	T_3	A	B	C
200	150	175	RT=0 WT=0	RT=0 WT=0	RT=0 WT=0
$r_1(B)$;					
	$r_2(A)$;				
		$r_3(C)$;			
$w_1(B)$;					
$w_1(A)$;					
	$w_2(C)$;				
		$w_3(A)$;			

Reglas para el planificador: Ejemplo

Atención

Suponemos que inmediatamente luego de la ultima operación, se realiza el commit

T_1	T_2	T_3	A	B	C
200	150	175	~~RT=0~~	~~RT=0~~	~~RT=0~~
			WT=0	WT=0	WT=0
$r_1(B)$;				RT=200	
	$r_2(A)$;		RT=150		
		$r_3(C)$;			RT=175
$w_1(B)$;					
$w_1(A)$;					
	$w_2(C)$;				
		$w_3(A)$;			

TS (T) >= WT(X)? Si! => como TS(T)>RT(X) => RT (X)=TS (T)

Reglas para el planificador: Ejemplo

Atención

Suponemos que inmediatamente luego de la ultima operación, se realiza el commit

T_1	T_2	T_3	A	B	C
200	150	175	~~RT=0~~ WT=0	~~RT=0~~ ~~WT=0~~	~~RT=0~~ WT=0
$r_1(B)$;				RT=200	
	$r_2(A)$;		RT=150		
		$r_3(C)$;			RT=175
$w_1(B)$;				WT=200	
$w_1(A)$;			WT=200		
	$w_2(C)$;				
		$w_3(A)$;			

TS (T) >= WT(X)? Si! => como TS(T)>RT(X) => RT (X)=TS (T)

TS(T) >= RT(X) y TS (T) >= WT (X) => WT (X)=TS (T)

Reglas para el planificador: Ejemplo

Atención

Suponemos que inmediatamente luego de la ultima operación, se realiza el commit

T_1	T_2	T_3	A	B	C
200	150	175	~~RT=0~~ WT=0	~~RT=0~~ ~~WT=0~~	~~RT=0~~ WT=0
$r_1(B)$;				RT=200	
	$r_2(A)$;		RT=150		
		$r_3(C)$;			RT=175
$w_1(B)$;				WT=200	
$w_1(A)$;			WT=200		
	$w_2(C)$;				
		$w_3(A)$;			

TS (T) >= WT(X)? Si! => como TS(T)>RT(X) => RT (X)=TS (T)

TS(T) >= RT(X) y TS (T) >= WT (X) => WT (X)=TS (T)

Atención

Luego de escribir B, se coloca C(B)=false. Luego de escribir A, T_1 commitea, siendo C(B)=true y C(A)=true

Reglas para el planificador: Ejemplo

Atención

Suponemos que inmediatamente luego de la ultima operación, se realiza el commit

T_1	T_2	T_3	A	B	C
200	150	175	~~RT=0~~ WT=0	~~RT=0~~ ~~WT=0~~	~~RT=0~~ WT=0
$r_1(B)$;				RT=200	
	$r_2(A)$;		RT=~~150~~ 0		
		$r_3(C)$;			RT=175
$w_1(B)$;				WT=200	
$w_1(A)$;			WT=200		
	$w_2(C)$;				
	Rollback!				
		$w_3(A)$;			

Ts (T) >= WT(X)? Si! => como TS(T)>RT(X) => RT (X)=Ts (T)

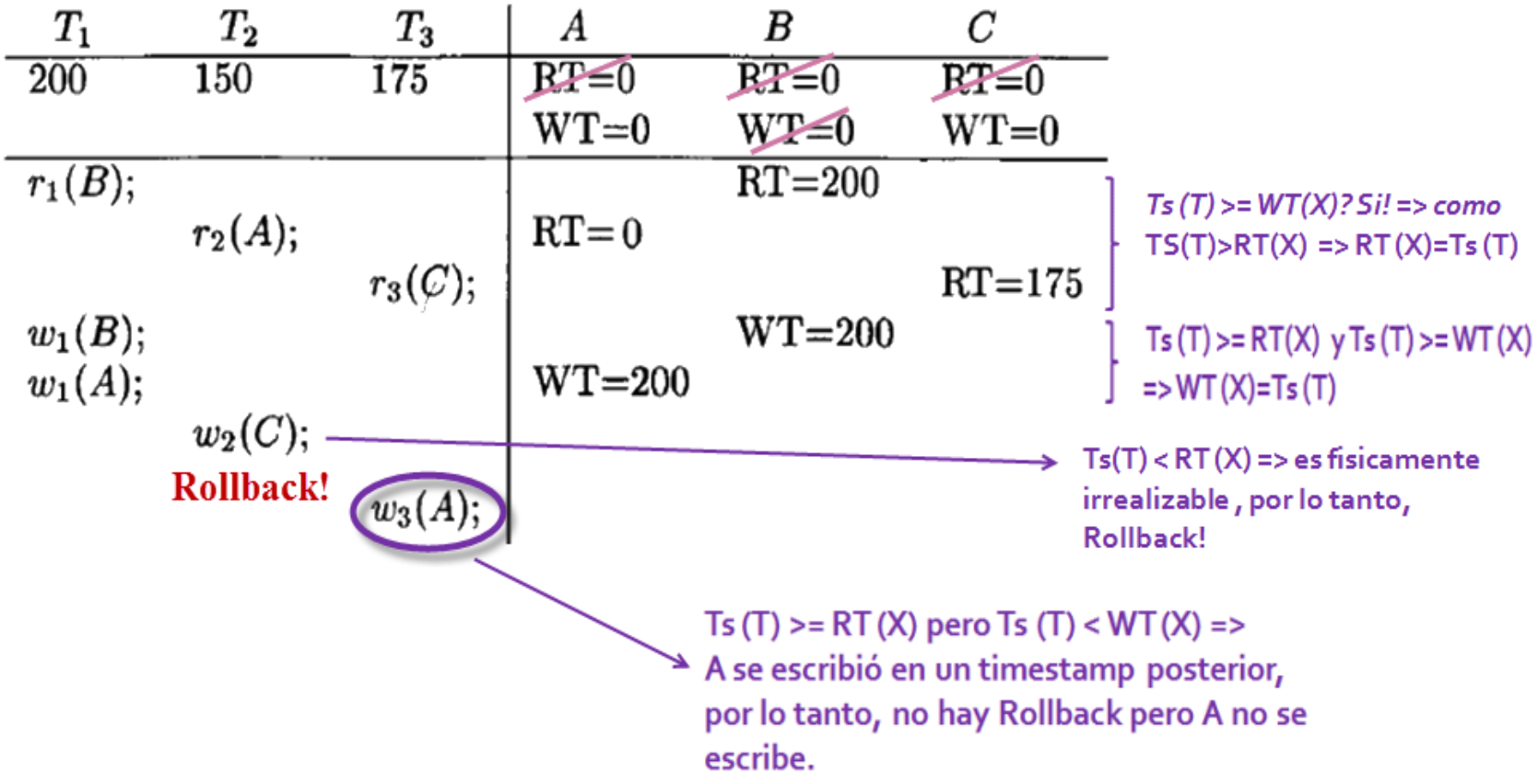
Ts (T) >= RT(X) y Ts (T) >= WT (X) => WT (X)=Ts (T)

Ts(T) < RT (X) => es fisicamente irrealizable , por lo tanto, Rollback!

Reglas para el planificador: Ejemplo

Atención

Suponemos que inmediatamente luego de la ultima operación, se realiza el commit



Preguntas...