Concurrencia y Recuperabilidad

Paradigma Optimista

Lic. Andrea Manna

2017

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

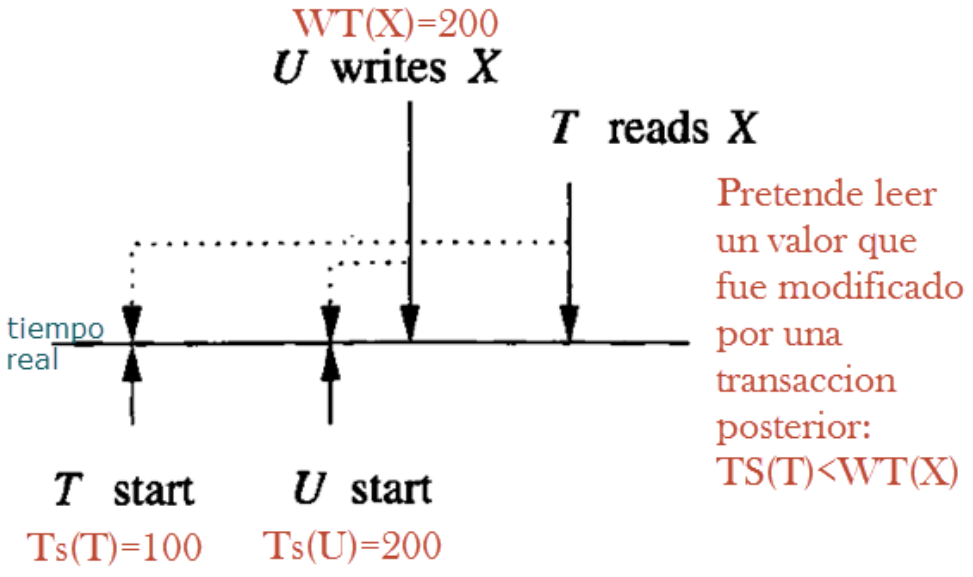
Read too Late

- 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

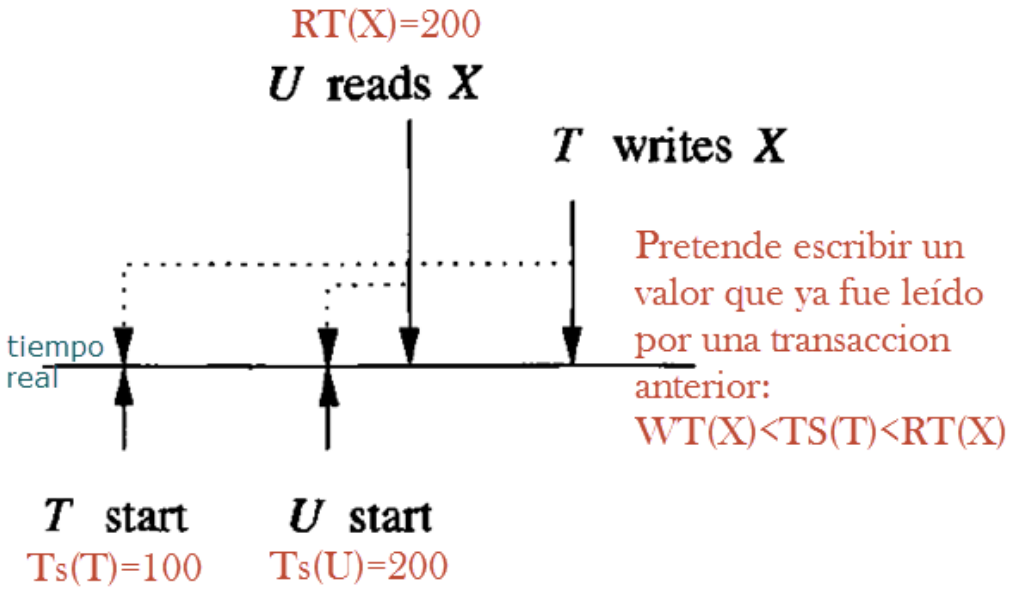
Write too Late

- 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).



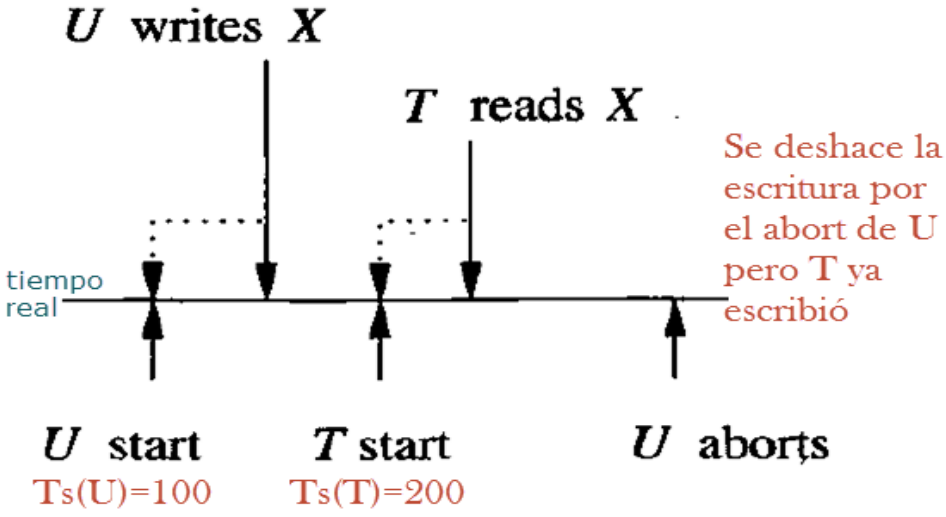
T debe abortar

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).

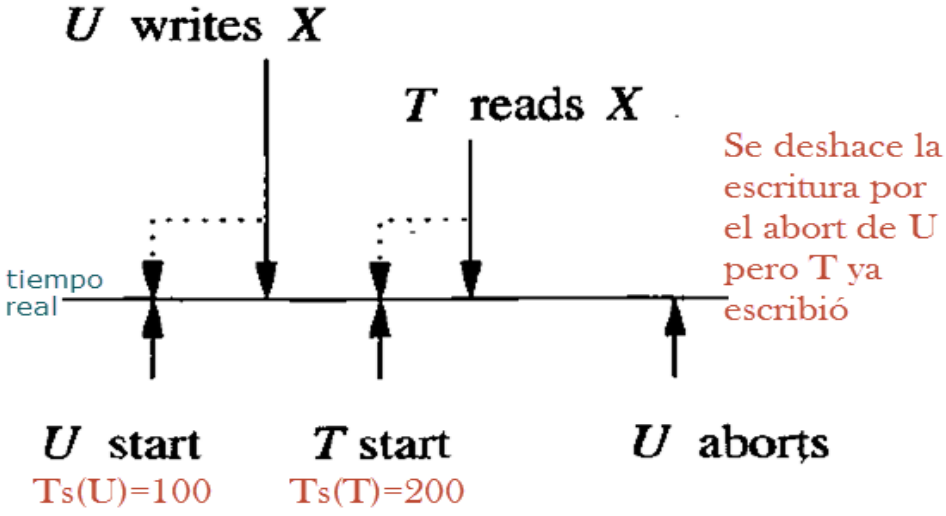
Dirty data (Lectura Sucia)

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 */
```

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

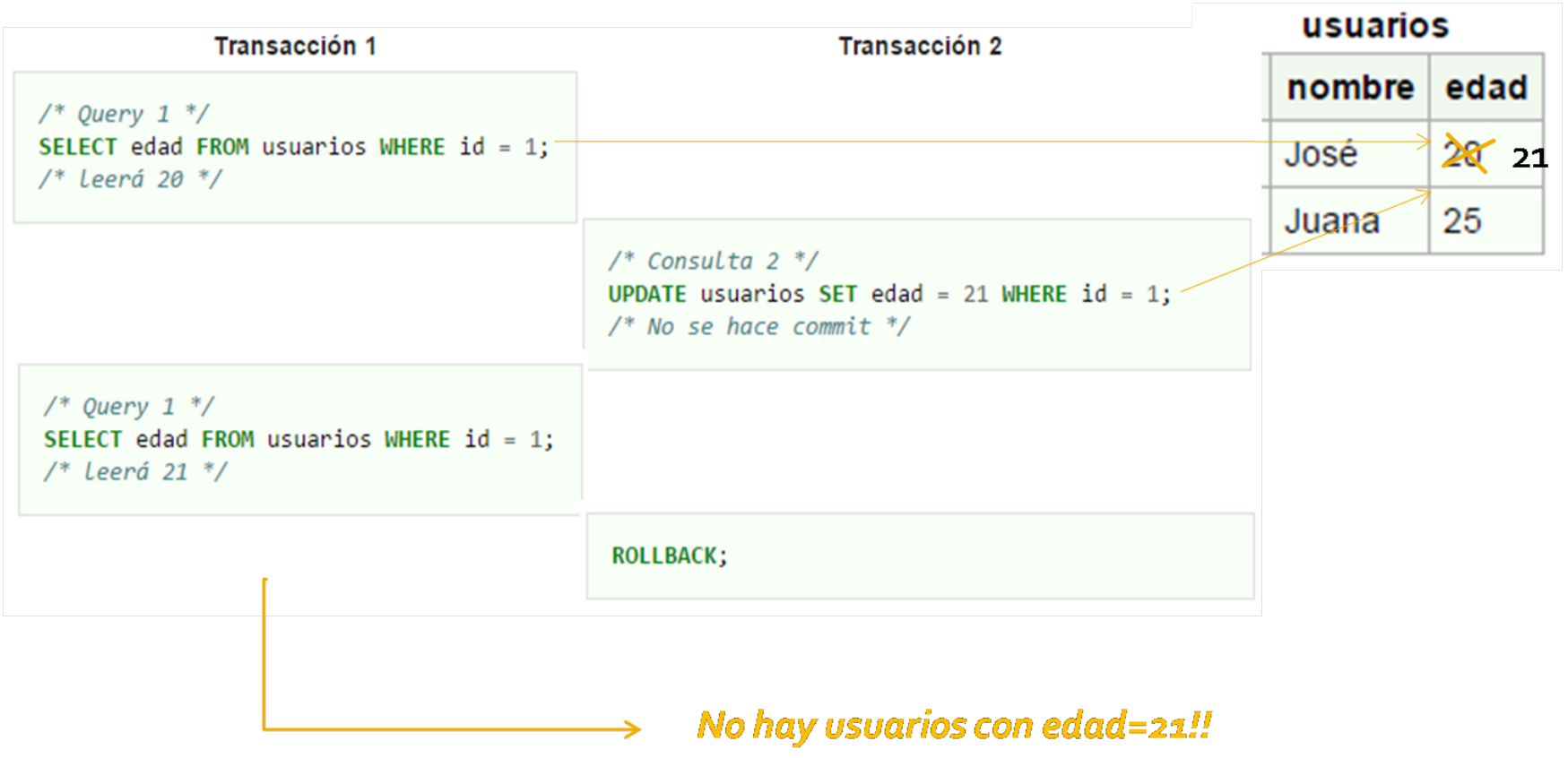
```
ROLLBACK;
```

usuarios

nombre	edad
José	20
Juana	25

Dirty Data: Ejemplo

Dirty Data: Ejemplo



Regla de escritura de Thomas

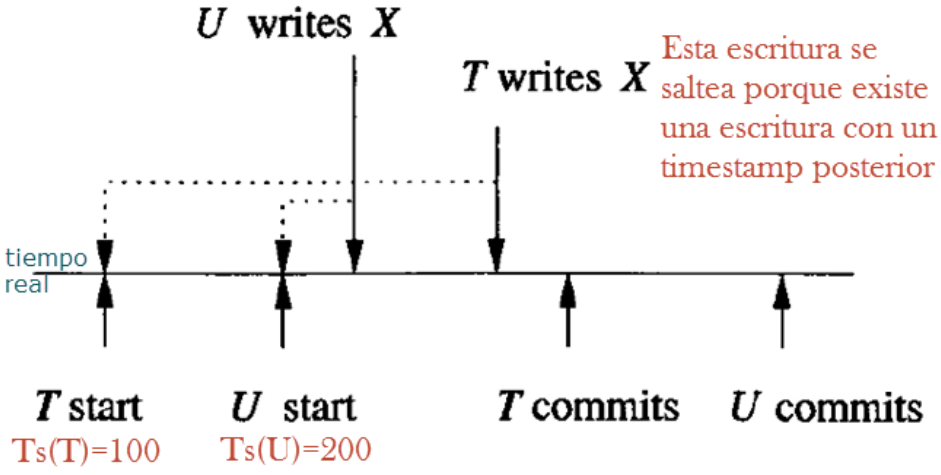
Thomas write rule

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

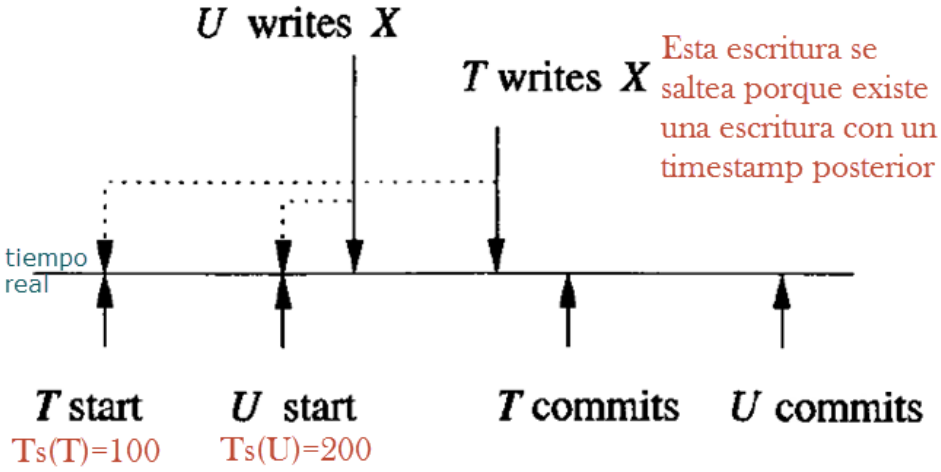
La escritura puede “saltearse” cuando ya existe una escritura de una transacción con un timestamp de mayor valor.
Es decir cuando WT(X) > TS(T)



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)

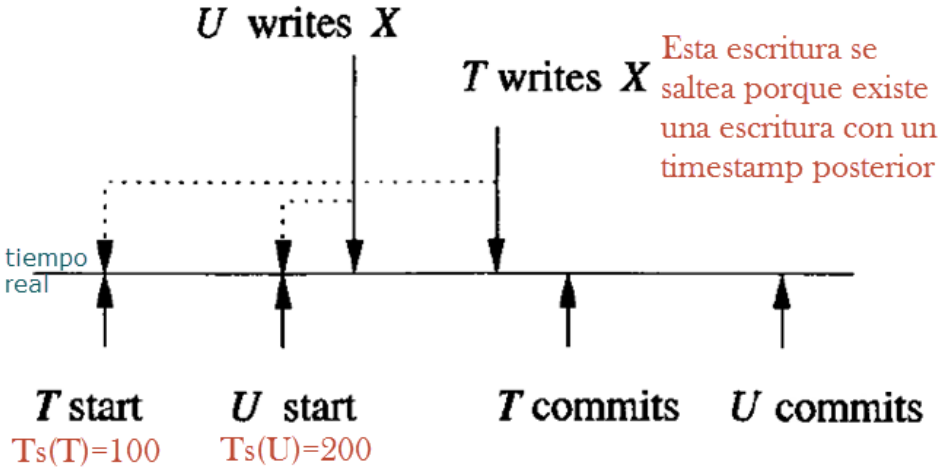


¿Que sucede si U realiza *abort* en vez de *commit*?

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*?

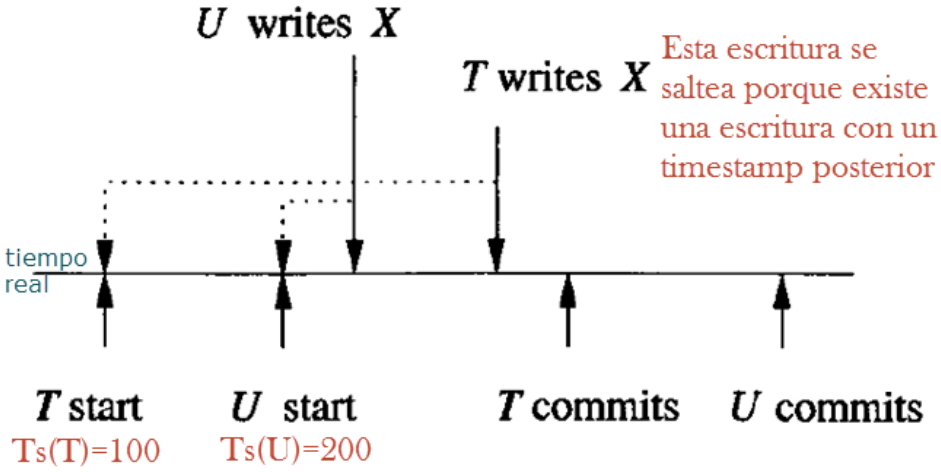
Problema si U aborta

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

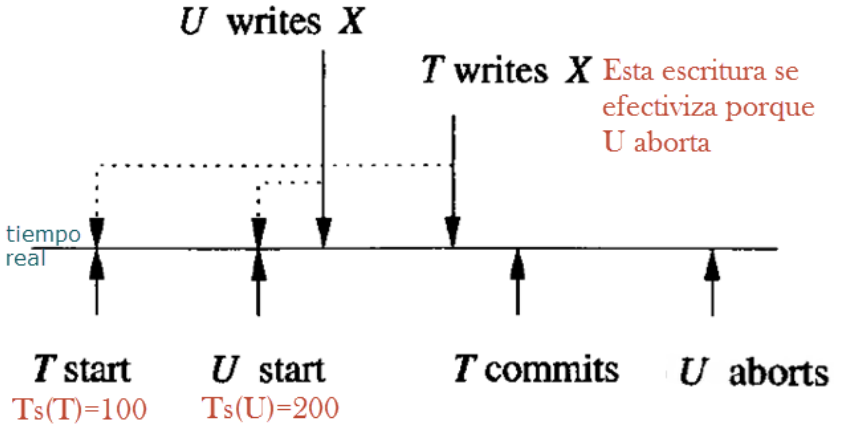
La escritura puede “saltearse” cuando ya existe una escritura de una transacción con un timestamp de mayor valor.
Es decir cuando WT(X) > TS(T)



¿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

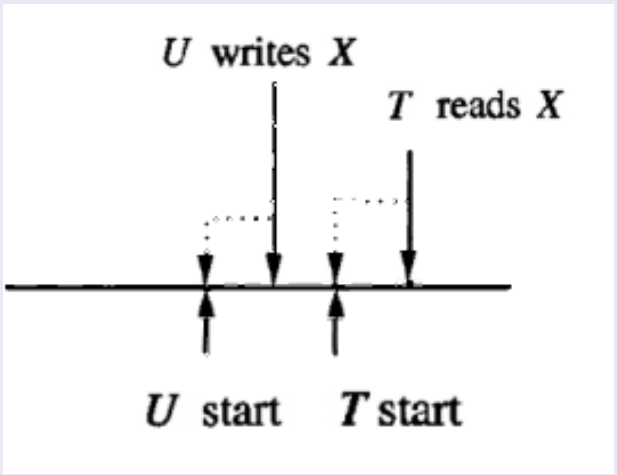
El planificador recibe una solicitud de lectura $r_t(X)$.

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** $r_t(X)$.

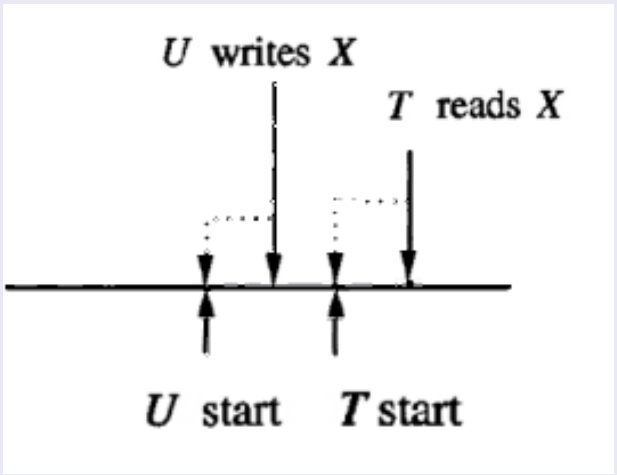
Caso 1: Si TS(T) >= WT(X) - es **físicamente realizable** es decir, no sucede read too late



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

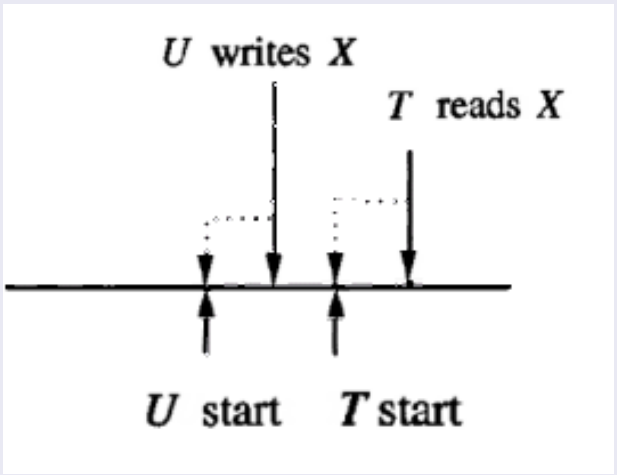


1. Si **C(X) es True**, conceder la solicitud. Si TS(T)>RT(X) hacer RT(X)=TS(T), de otro modo no cambiar RT(X).

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

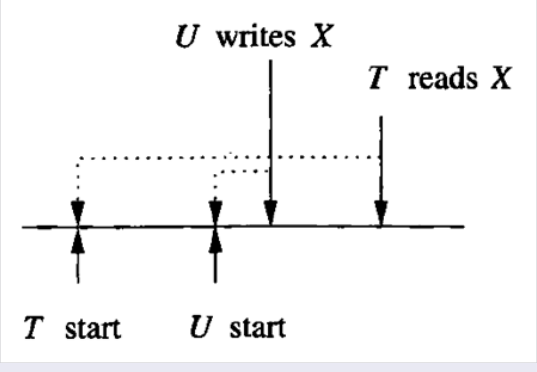
El planificador recibe una solicitud de lectura $r_t(X)$.

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

Reglas para el planificador

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

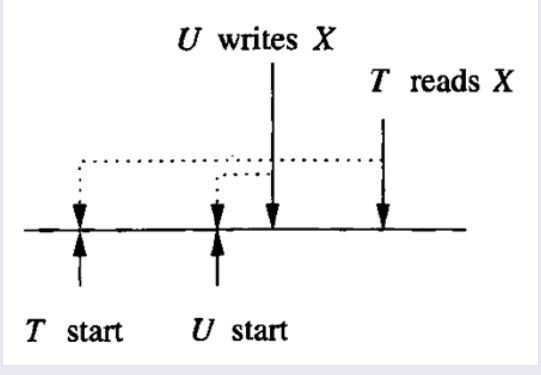
Caso 2: Si TS(T) < WT(X) - es **físicamente irrealizable** (***read too late***)



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

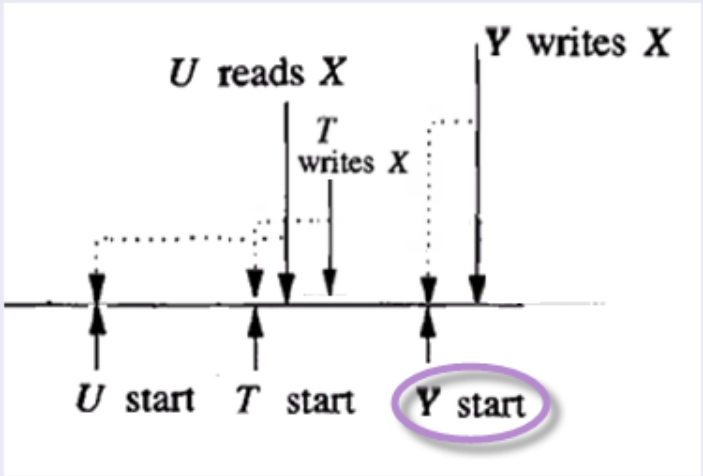
El planificador recibe una solicitud de escritura $w_t(X)$.

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** $w_t(X)$.

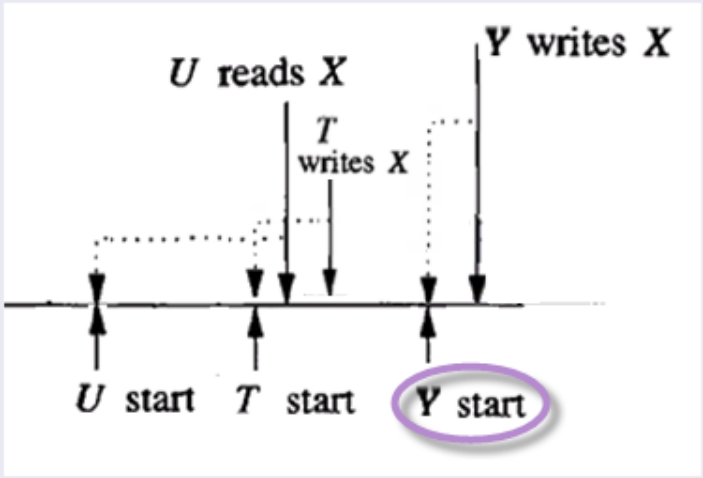
Caso 1: Si TS(T) >= RT(X) y TS(T) >= WT(X) - es **físicamente realizable**, es decir, no sucede write too late



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

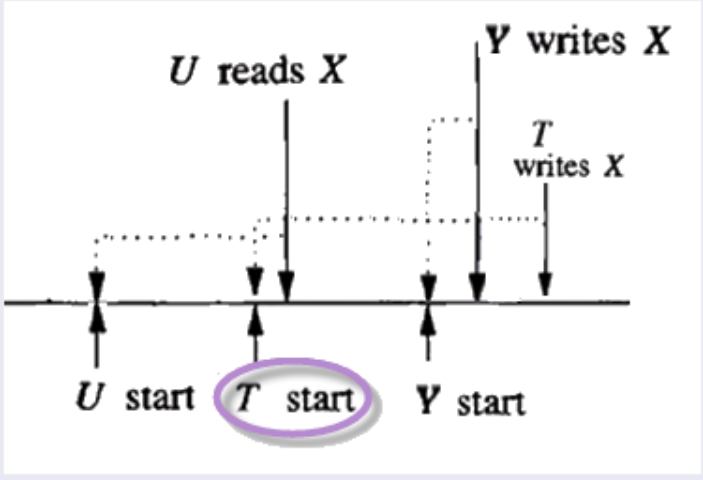
El planificador recibe una solicitud de escritura $w_t(X)$.

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** $w_t(X)$.

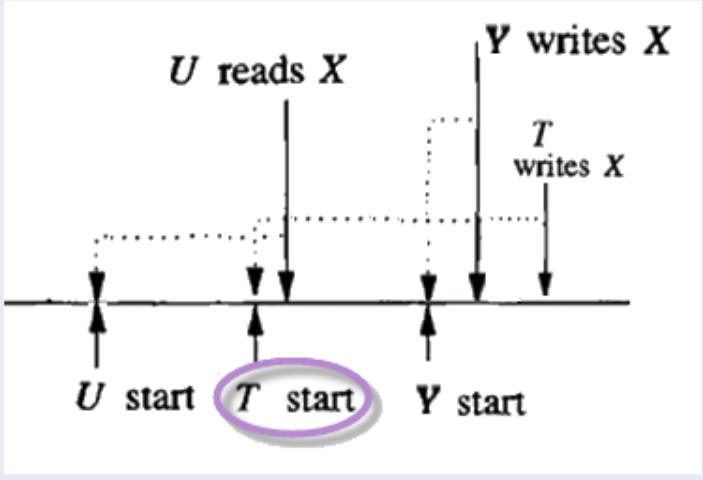
Caso 2: Si $TS(T) >= RT(X)$ pero $TS(T) < WT(X)$ - es **físicamente realizable**, pero ya hay un valor posterior en X.



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

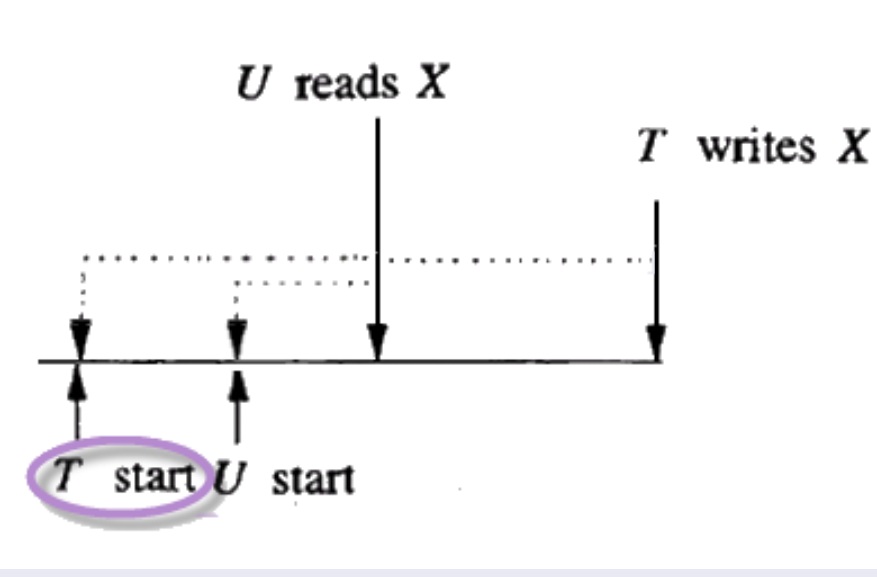
El planificador recibe una solicitud de escritura $w_t(X)$.

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* de cada operacion de escritura se realiza inmediatamente después del correspondiente *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

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

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)

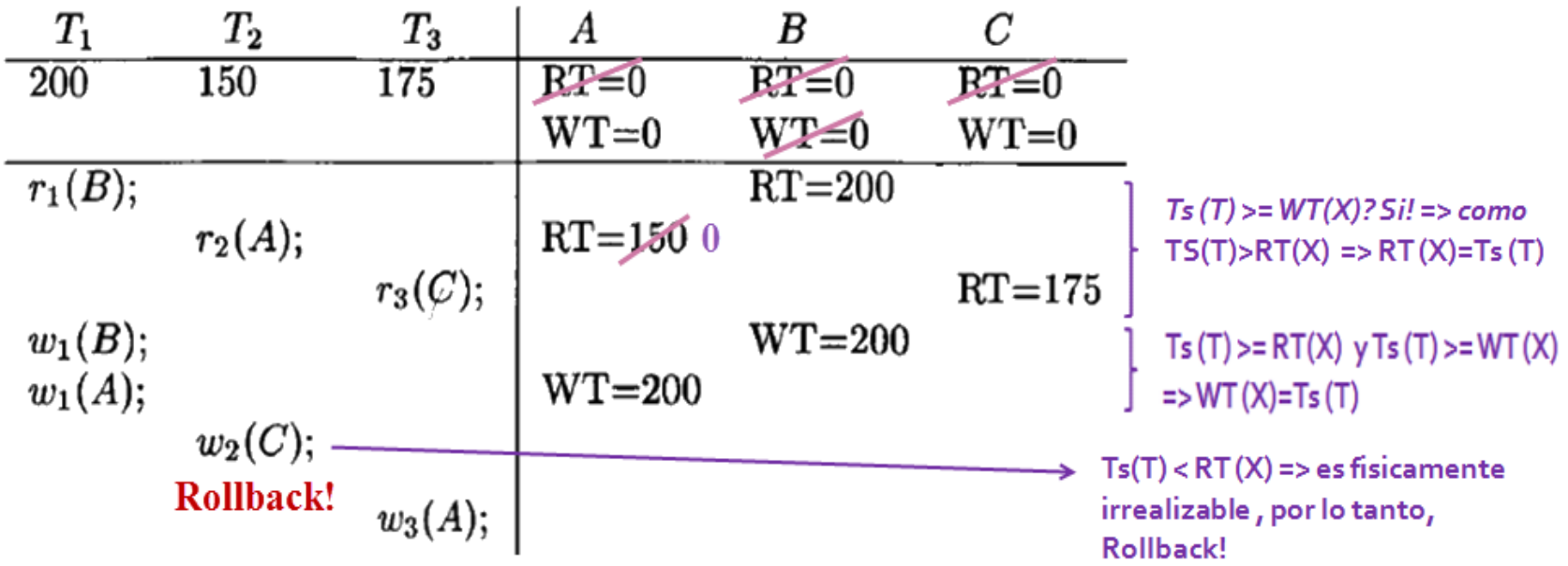
Reglas para el planificador: Ejemplo

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
$w_1(B)$;				WT=200	
$w_1(A)$;			WT=200		
	$w_2(C)$;				
		$w_3(A)$;			

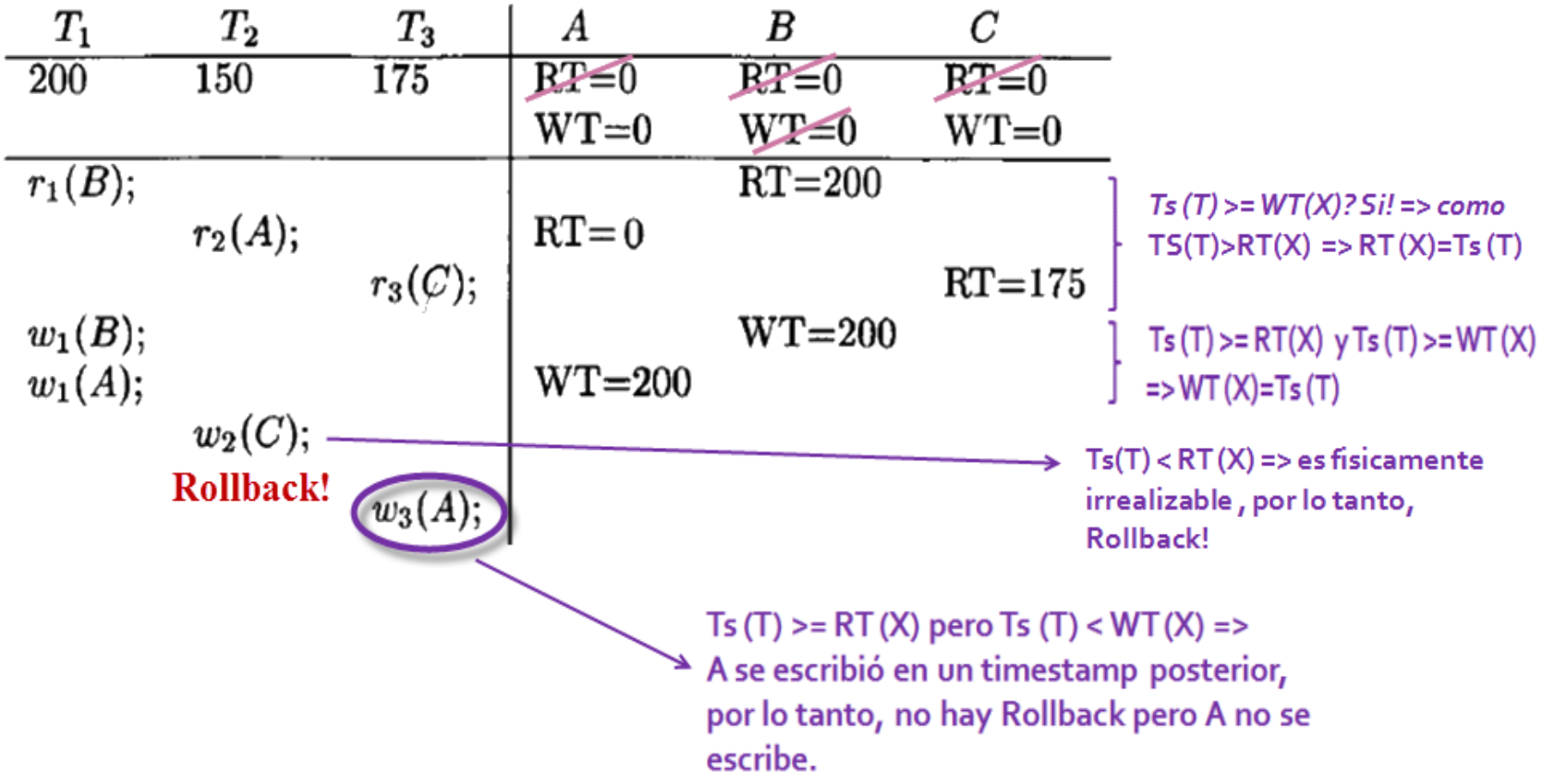
TS (T) >= WT(X)? Si! => como TS(T)>RT(X) => RT (X)=TS (T)

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

Reglas para el planificador: Ejemplo



Reglas para el planificador: Ejemplo



Timestamping Multiversion

Planificador multiversión

Definición

Planificador multiversión

Definición

- Variación del mecanismo de timestamp que mantiene versiones antiguas de los elementos de la base de datos

Planificador multiversión

Definición

- Variación del mecanismo de timestamp que mantiene versiones antiguas de los elementos de la base de datos
- Permite RT(X) que en otras ocasiones causarían que la transacción T aborte debido a que la versión actual de X fue escrita por una transacción posterior.

Planificador multiversión

Definición

- Variación del mecanismo de timestamp que mantiene versiones antiguas de los elementos de la base de datos
- Permite RT(X) que en otras ocasiones causarían que la transacción T aborte debido a que la versión actual de X fue escrita por una transacción posterior.
- Permite leer la versión de X apropiada según el timestamp de T.

Planificador multiversión

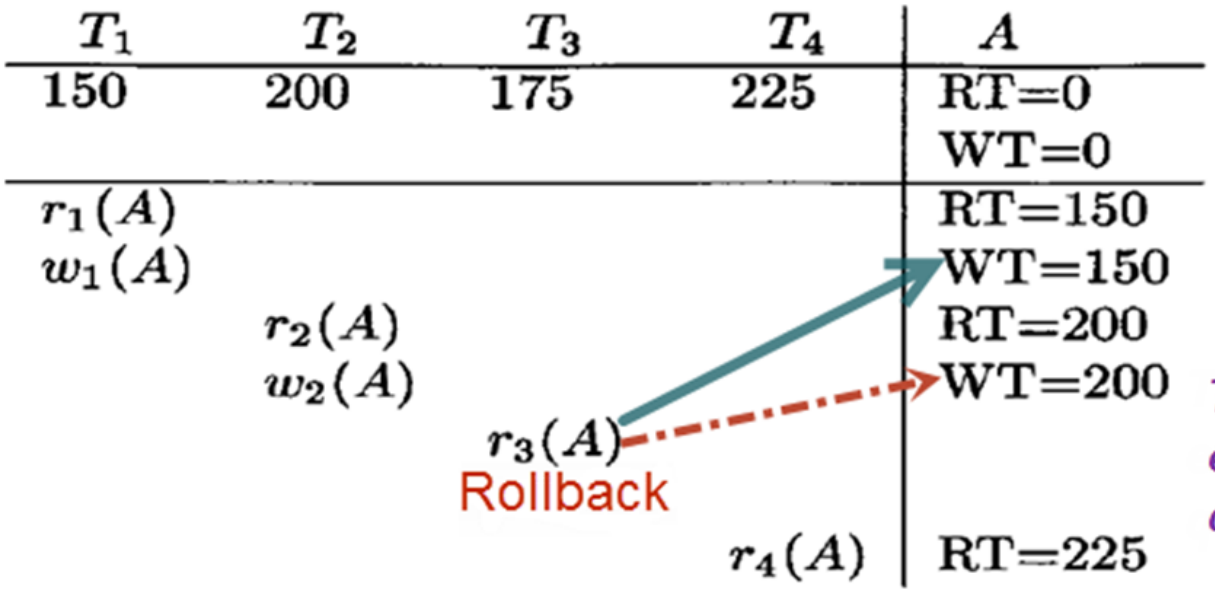
Definición

- Variación del mecanismo de timestamp que mantiene versiones antiguas de los elementos de la base de datos
- Permite RT(X) que en otras ocasiones causarían que la transacción T aborte debido a que la versión actual de X fue escrita por una transacción posterior.
- Permite leer la versión de X apropiada según el timestamp de T.

Planificador multiversión

T_1	T_2	T_3	T_4	A
150	200	175	225	RT=0 WT=0
$r_1(A)$				RT=150
$w_1(A)$				WT=150
	$r_2(A)$			RT=200
	$w_2(A)$			WT=200
		$r_3(A)$		
			$r_4(A)$	RT=225

Planificador multiversión



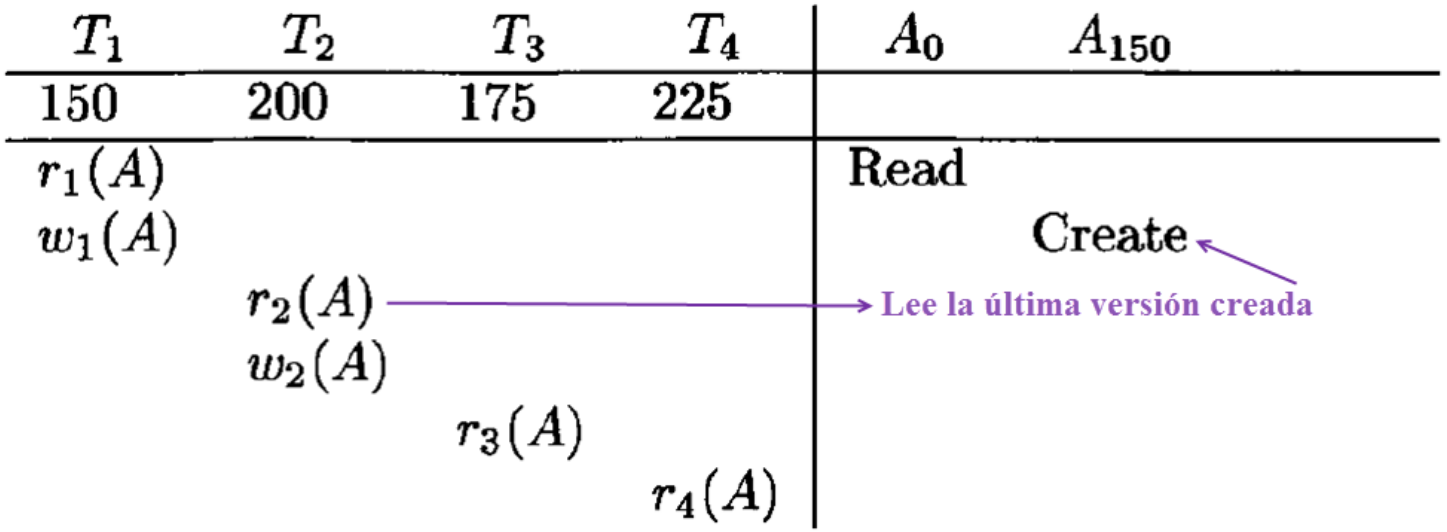
Ojo!

T_3 debería realizar rollback al no poder un leer valor de X que fue escrito por una transacción posterior. En vez de esto, T_3 lee el valor escrito por T_1, mientras que T_4 leerá el valor escrito por T_2

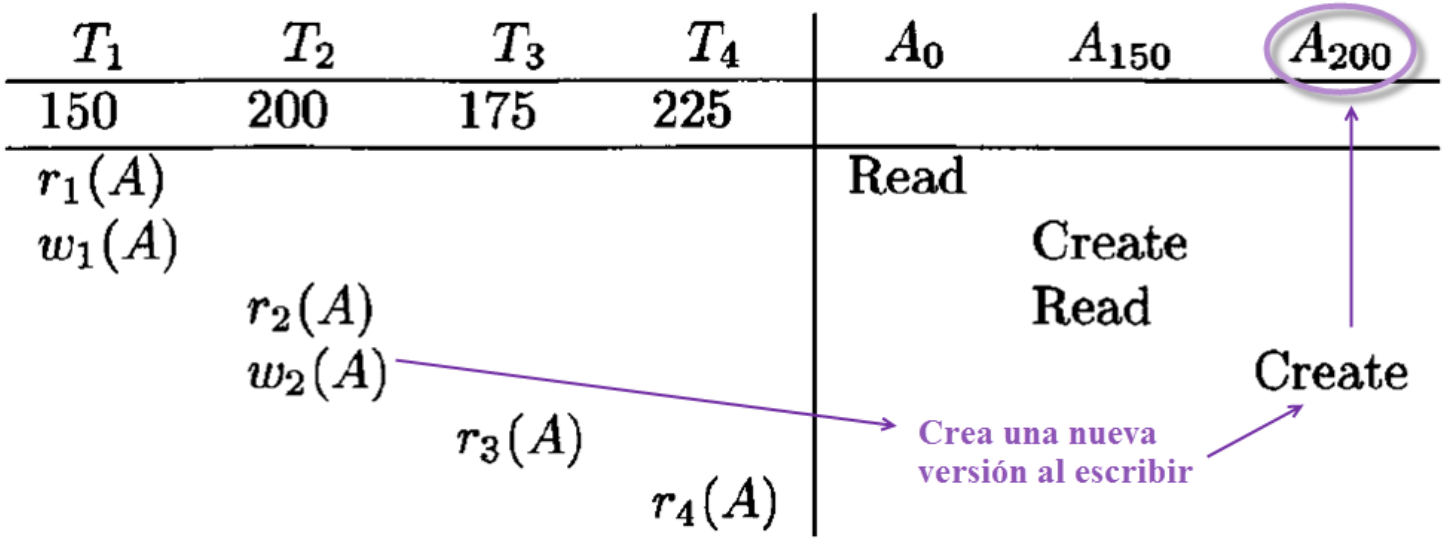
Planificador multiversión: Ejemplo

T_1	T_2	T_3	T_4	A_0	A_{150}
150	200	175	225		
$r_1(A)$				Read	
$w_1(A)$					Create
	$r_2(A)$				
	$w_2(A)$				
		$r_3(A)$			
			$r_4(A)$		

Planificador multiversión: Ejemplo



Planificador multiversión: Ejemplo



Planificador multiversión: Ejemplo

T_1	T_2	T_3	T_4	A_0	A_{150}	A_{200}
150	200	175	225			
$r_1(A)$				Read		
$w_1(A)$					Create	
	$r_2(A)$				Read	
	$w_2(A)$					Create
		$r_3(A)$			Read	
			$r_4(A)$			

Tiene que leer una versión anterior a la última creada porque su TS es menor al de la última versión.

Planificador multiversión: Ejemplo

T_1	T_2	T_3	T_4	A_0	A_{150}	A_{200}
150	200	175	225			
$r_1(A)$				Read		
$w_1(A)$					Create	
	$r_2(A)$				Read	
	$w_2(A)$					Create
		$r_3(A)$			Read	
			$r_4(A)$			Read

Planificador multiversión

Considerando las reglas anteriores...

- Cuando ocurre $w_t(X)$, si es legal (**segun las reglas vistas**) entonces se crea una nueva versión del elemento X. Su tiempo de escritura es Ts(T) y nos referimos a él como X_t , donde $t = TS(T)$.
- Cuando ocurre una lectura $r_t(X)$ el planificador busca una versión X_t de X tal que $t <= TS(T)$ y que no haya otra versión X'_t tal que $t < t' <= TS(T)$.
- Los tiempos de escritura están asociados a versiones de un elemento y nunca cambian.
- Los tiempos de lectura también son asociados con versiones. Lo podemos notar como $X_{t,tr}$ (donde tr es el último tiempo de lectura de X_t) . Una transacción T' que quiere escribir debe hacer rollback cuando existe alguna $X_{t,tr}$ tal que $t < TS(T')$ y $tr > TS(T')$

Panificador Multiversión

Consideramos el siguiente ejemplo

T_1	T_2	T_3	T_4	A_0	A_{50}	A_{100}
50	60	80	100			
$w_1(A)$				Create		
			$w_4(A)$			Create
		$r_3(A)$		Read		
	$w_2(A)$					
	ABORT					

Panificador Multiversión

Consideramos el siguiente ejemplo

T_1	T_2	T_3	T_4	A_0	A_{50}	A_{100}
50	60	80	100			
$w_1(A)$				Create		
			$w_4(A)$			Create
		$r_3(A)$		Read		
	$w_2(A)$					
	ABORT					

Ejemplo

Tenemos A_{50} y A_{100}. T con TS(T)=80 lee A_{50}. T' con $TS(T') = 60$ intenta escribir. En este caso T' debe hacer rollback.

Panificador Multiversión

Consideramos el siguiente ejemplo

T_1	T_2	T_3	T_4	A_0	A_{50}	A_{100}
50	60	80	100			
$w_1(A)$				Create		
			$w_4(A)$			Create
		$r_3(A)$		Read		
	$w_2(A)$					
	ABORT					

Ejemplo

Tenemos A_{50} y A_{100}. T con TS(T)=80 lee A_{50}. T' con $TS(T') = 60$ intenta escribir. En este caso T' debe hacer rollback.

Físicamente Irrealizable!

$$t = 50 < TS(T') = 60 \text{ y}$$
$$tr = 80 > TS(T') = 60$$

Panificador Multiversión

Consideramos el siguiente ejemplo

T_1	T_2	T_3	T_4	A_0	A_{50}	A_{100}
50	60	80	100			
$w_1(A)$				Create		
			$w_4(A)$			Create
		$r_3(A)$		Read		
	$w_2(A)$					
	ABORT					

Ejemplo

Tenemos A_{50} y A_{100}. T con TS(T)=80 lee A_{50}. T' con $TS(T') = 60$ intenta escribir. En este caso T' debe hacer rollback.

Físicamente Irrealizable!

$$t = 50 < TS(T') = 60 \text{ y}$$
$$tr = 80 > TS(T') = 60$$

Regla de Borrado de Versiones

Si una versión X_t tiene un tiempo de escritura tal que no existe una transacción activa T tal que TS(T) sea menor t, se puede borrar cualquier versión de X previa a X_t.

Optimista vs. Pesimista

Compromiso

- Generalmente el timestamping es mejor cuando la mayoría de las transacciones son de lectura o es raro que haya transacciones que traten de leer y escribir el mismo elemento.
- En situaciones de mucho conflicto, locking suele comportarse mejor.
- Se establece entonces un compromiso utilizado en los sistemas comerciales:

1. Transacciones read-only vs. Transacciones read-write.
2. Transacciones read-write se manjean con locking pero crean versiones de los elementos.
3. Transacciones read-only se manejan con versiones creadas por transacciones read-write

Validation

Planificador con validación

Definición

Planificador con validación

Definición

- Se debe tener para cada transacción T los conjuntos:

Planificador con validación

Definición

- Se debe tener para cada transacción T los conjuntos:
  1. $R_S(T)$ elementos leídos por T.
  2. $W_S(T)$ elementos escritos por T.

Planificador con validación

Definición

- Se debe tener para cada transacción T los conjuntos:
  1. $R_S(T)$ elementos leídos por T.
  2. $W_S(T)$ elementos escritos por T.
- Transacciones se ejecutan en 3 fases:

Planificador con validación

Definición

- Se debe tener para cada transacción T los conjuntos:
  1. $R_S(T)$ elementos leídos por T.
  2. $W_S(T)$ elementos escritos por T.
- Transacciones se ejecutan en 3 fases:
  1. **Lectura**: Lee desde la base de datos todos los elementos en su $R_S(T)$. Calcula en su espacio de direcciones local los elementos a escribir.

Planificador con validación

Definición

- Se debe tener para cada transacción T los conjuntos:
  1. $R_S(T)$ elementos leídos por T.
  2. $W_S(T)$ elementos escritos por T.
- Transacciones se ejecutan en 3 fases:
  1. **Lectura**: Lee desde la base de datos todos los elementos en su $R_S(T)$. Calcula en su espacio de direcciones local los elementos a escribir.
  2. **Validación**: el planificador valida la transacción comparando su R_S y W_S con los de otras transacciones. Si la validación falla se ejecuta un rollback y se reinicia, sino se pasa al paso 3.

Planificador con validación

Definición

- Se debe tener para cada transacción T los conjuntos:
  1. $R_S(T)$ elementos leídos por T.
  2. $W_S(T)$ elementos escritos por T.
- Transacciones se ejecutan en 3 fases:
  1. **Lectura**: Lee desde la base de datos todos los elementos en su $R_S(T)$. Calcula en su espacio de direcciones local los elementos a escribir.
  2. **Validación**: el planificador valida la transacción comparando su R_S y W_S con los de otras transacciones. Si la validación falla se ejecuta un rollback y se reinicia, sino se pasa al paso 3.
  3. **Escritura**: Los elementos de W_S son escritos en la base de datos.

Planificador con validación

Definición

El planificador mantiene tres conjuntos:

Planificador con validación

Definición

El planificador mantiene tres conjuntos:

1. **START**: conjunto de transacciones que han comenzado pero aún no completaron la validación. Para cada transacción T en este conjunto se mantiene START(T) que es el tiempo en el cual T comenzó.

Planificador con validación

Definición

El planificador mantiene tres conjuntos:

1. **START**: conjunto de transacciones que han comenzado pero aún no completaron la validación. Para cada transacción T en este conjunto se mantiene START(T) que es el tiempo en el cual T comenzó.
2. **VAL**: el conjunto de transacciones que han sido validadas pero aún no finalizaron la fase de escritura. Para cada transacción T en este conjunto se mantienen dos valores START(T) y VAL(T). Este último es el tiempo en el cual T es validada

Planificador con validación

Definición

El planificador mantiene tres conjuntos:

1. **START**: conjunto de transacciones que han comenzado pero aún no completaron la validación. Para cada transacción T en este conjunto se mantiene START(T) que es el tiempo en el cual T comenzó.
2. **VAL**: el conjunto de transacciones que han sido validadas pero aún no finalizaron la fase de escritura. Para cada transacción T en este conjunto se mantienen dos valores START(T) y VAL(T). Este último es el tiempo en el cual T es validada
3. **END**: el conjunto de transacciones que han completado la fase 3. El planificador mantiene para estas transacciones START(T), VAL(T) y END(T). Este conjunto, que crecería indefinidamente, puede ser limpiado eliminado aquellas transacciones T tales que para cualquier transacción activa U pase que END(T) < START(U)

Planificador con validación

Definicion

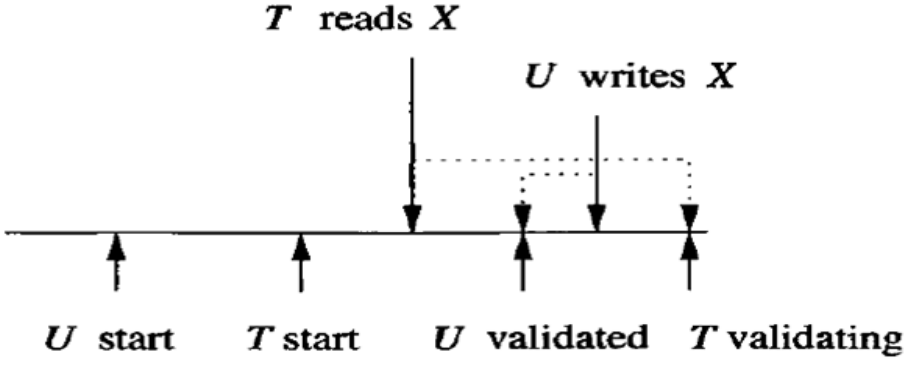
- El orden serial puede pensarse usando el tiempo de validación. Es decir la transacción T debería ejecutarse en el momento de su validación formando un orden serial hipotético.
- Se puede pensar al tiempo de validación como el tiempo de ejecución en un hipotético orden serial.

Planificador con validación

¿Qué puede ir mal?

Supongamos una transacción U y una transacción T tal que:

- **U está en VAL o END**; o sea: **U** fue validada
- **END(U) > START(T)**, **U** no terminó antes que el comienzo de T
- $R_S(T) \cap W_S(U)$ no es vacío.

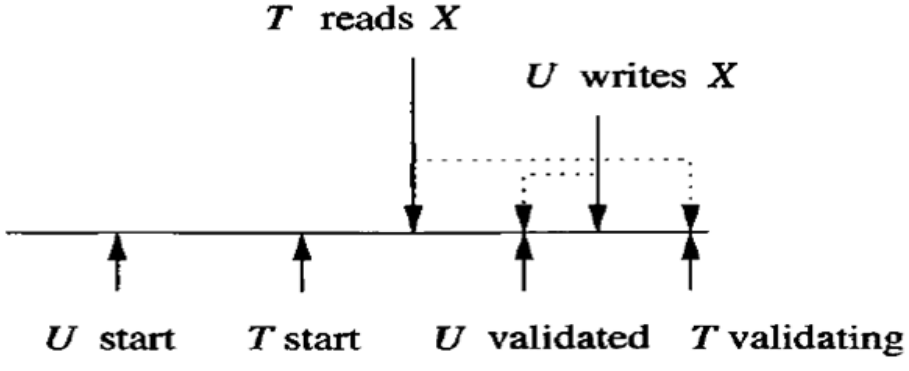


Planificador con validación

¿Qué puede ir mal?

Supongamos una transacción U y una transacción T tal que:

- **U está en VAL o END**; o sea: **U** fue validada
- **END(U) > START(T)**, **U** no terminó antes que el comienzo de T
- $R_S(T) \cap W_S(U)$ no es vacío.



Ejemplo

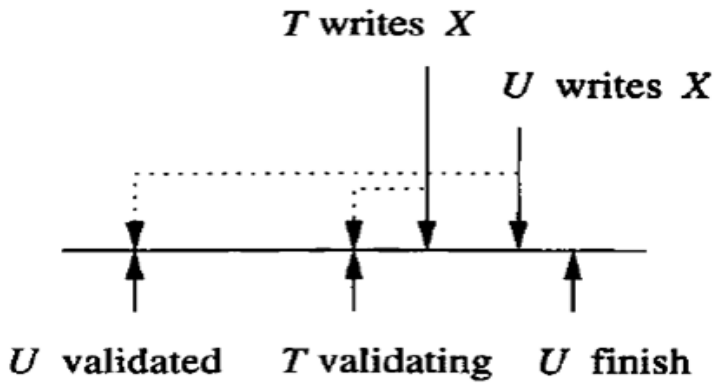
T no puede validar si una transacción anterior está escribiendo algo que T debería haber leido.
Por lo tanto **T aborta**

Planificador con validación

¿Qué puede ir mal?

Supongamos una transacción U y una transacción T tal que:

- **U está en VAL o END**; o sea: **U** fue validada exitosamente
- **END(U) > VAL(T)**, **U** no terminó antes de que T hay entrado en su fase de validación
- $W_S(T) \cap W_S(U)$ no es vacío. Por ejemplo, X está en ambos conjuntos de escritura

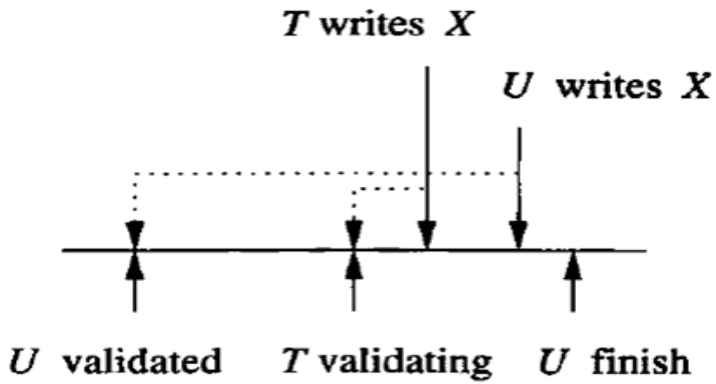


Planificador con validación

¿Qué puede ir mal?

Supongamos una transacción U y una transacción T tal que:

- **U está en VAL o END**; o sea: **U** fue validada exitosamente
- **END(U) > VAL(T)**, **U** no terminó antes de que T hay entrado en su fase de validación
- $W_S(T) \cap W_S(U)$ no es vacío. Por ejemplo, X está en ambos conjuntos de escritura



Ejemplo

T no puede validarse exitosamente si podría llegar a escribir algo antes que una transacción anterior.
Por lo tanto **T aborta**

Planificador con validación

Reglas

Para validar una transacción T hay que:

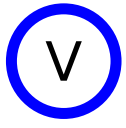
- Verificar que $\boldsymbol{R_S(T) \cap W_S(U)}$ es vacío para cualquier transacción **U validada** previamente y que no finalizó antes de que **T comience**.
- Verificar $\boldsymbol{W_S(T) \cap W_S(U)}$ es vacío para cualquier transacción **U validada** previamente y que no finalizó antes de que **T sea validada**.

Ejemplo de Planificador con Validación

$R_P = \{B\}$
$W_P = \{D\}$

$R_W = \{A, D\}$
$W_W = \{A, C\}$

$R_S = \{A, B\}$
$W_S = \{A, C\}$



$R_V = \{B\}$
$W_V = \{D, E\}$

- Verificar que $\boldsymbol{R_T} \cap \boldsymbol{W_U}$ es vacío para cualquier transacción $\boldsymbol{U}$ validada previamente y que no finalizo antes de que $\boldsymbol{T}$ comenzara.
- Verificar $\boldsymbol{W_T} \cap \boldsymbol{W_U}$ es vacío para cualquier transacción $\boldsymbol{U}$ validada previamente y que no finalizo antes de que $\boldsymbol{T}$ sea validada.

Ejemplo de Planificador con Validación

$R_P = \{B\}$
$W_P = \{D\}$

P

$R_W = \{A, D\}$
$W_W = \{A, C\}$

W

$R_S = \{A, B\}$
$W_S = \{A, C\}$

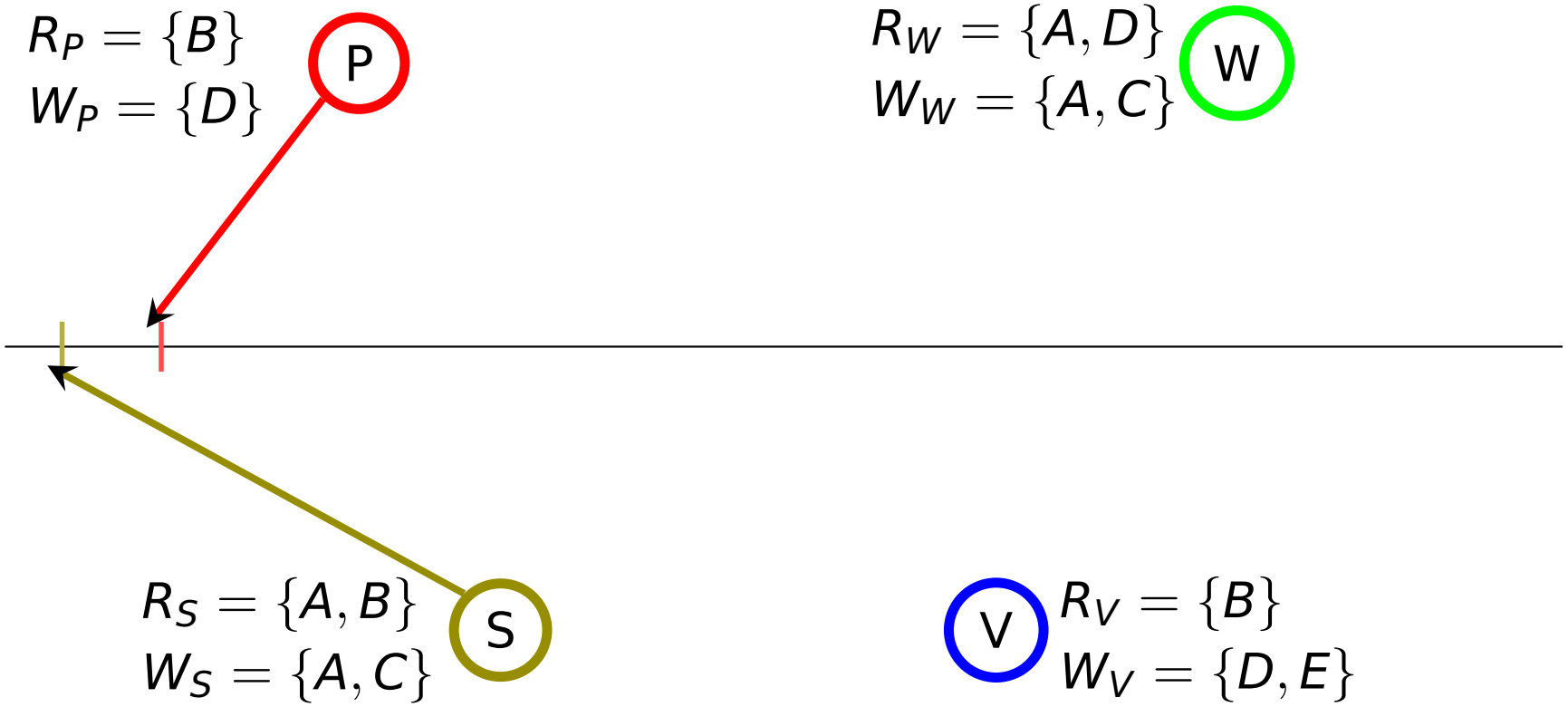
S

V

$R_V = \{B\}$
$W_V = \{D, E\}$

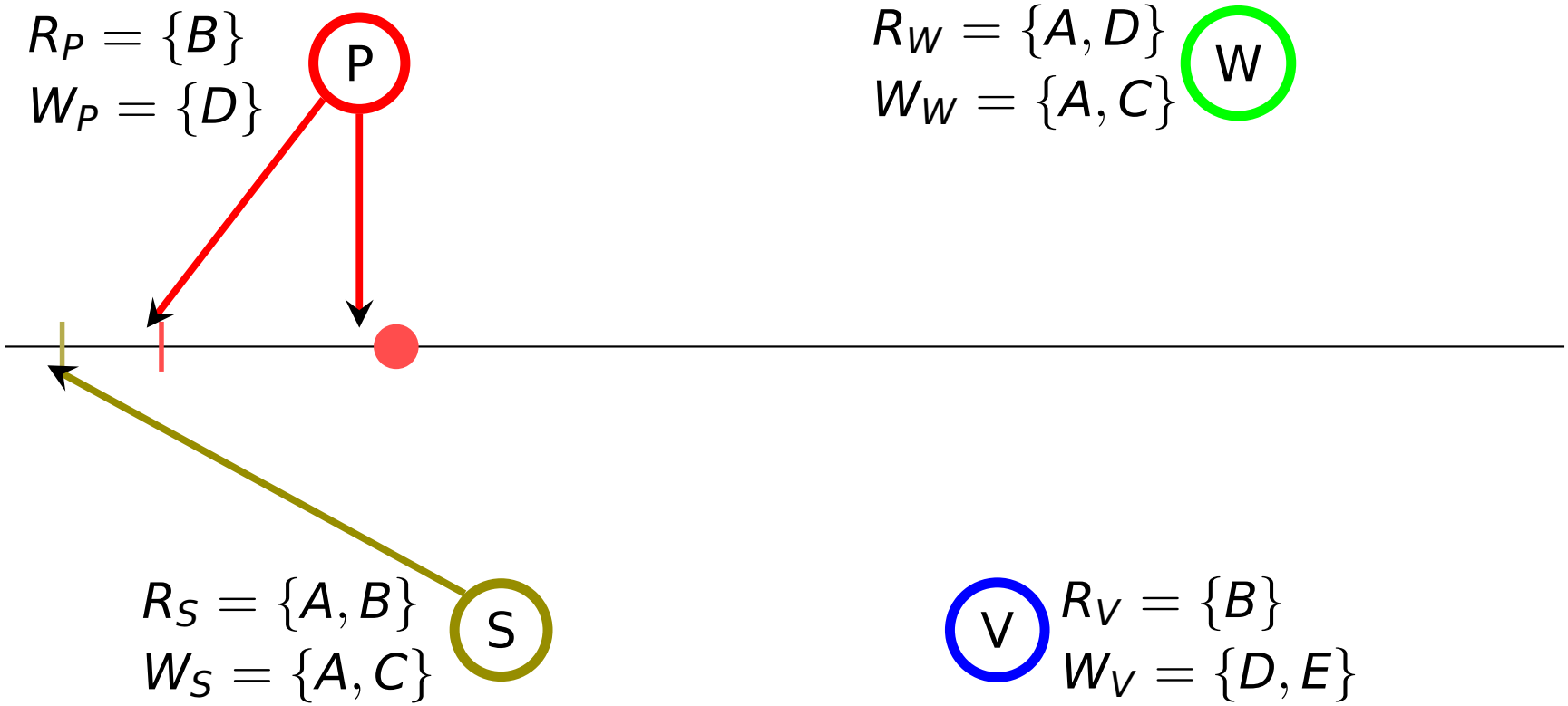
- Verificar que $\boldsymbol{R_T} \cap \boldsymbol{W_U}$ es vacío para cualquier transacción $\boldsymbol{U}$ validada previamente y que no finalizo antes de que $\boldsymbol{T}$ comenzara.
- Verificar $\boldsymbol{W_T} \cap \boldsymbol{W_U}$ es vacío para cualquier transacción $\boldsymbol{U}$ validada previamente y que no finalizo antes de que $\boldsymbol{T}$ sea validada.

Ejemplo de Planificador con Validación



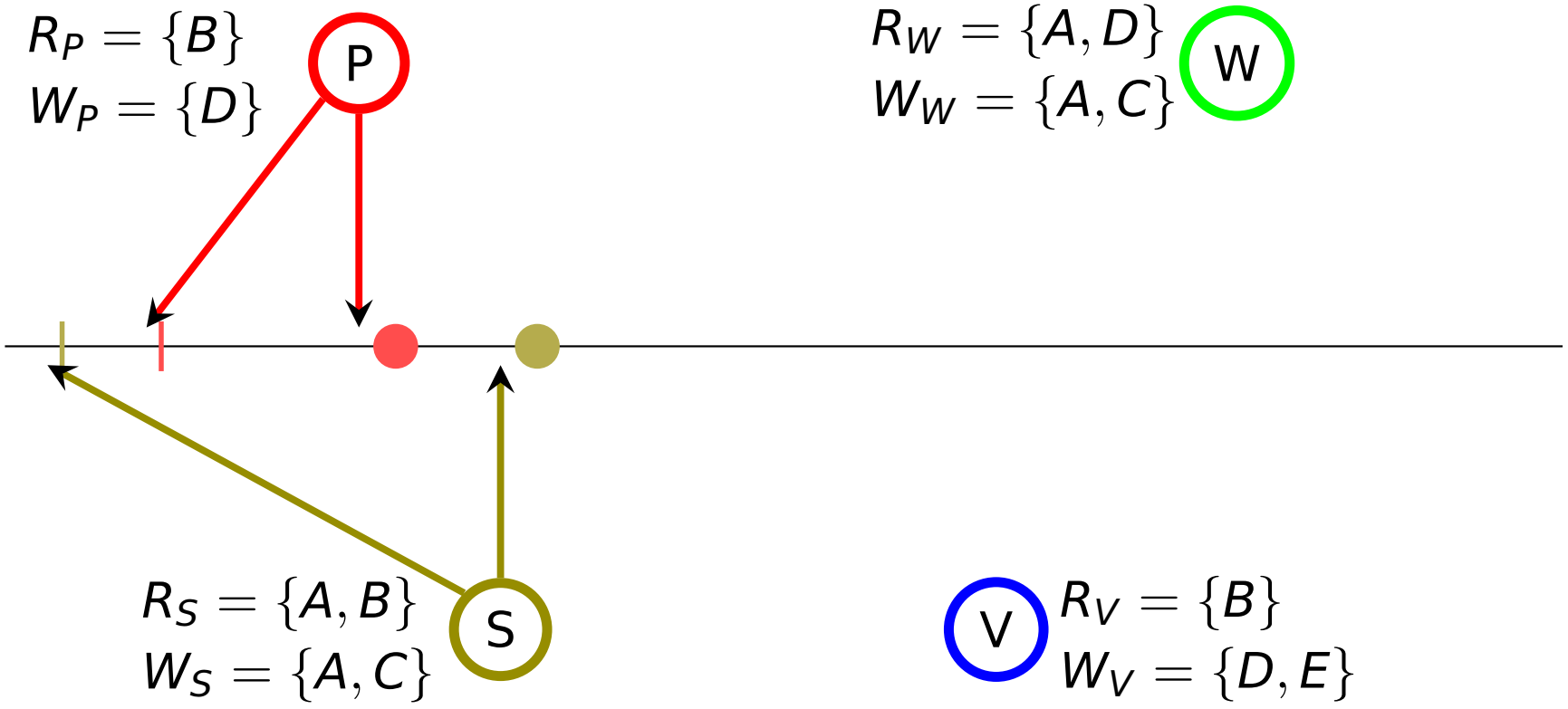
- Verificar que $R_T \cap W_U$ es vacío para cualquier transacción U validada previamente y que no finalizo antes de que T comenzara.
- Verificar $W_T \cap W_U$ es vacío para cualquier transacción U validada previamente y que no finalizo antes de que T sea validada.

Ejemplo de Planificador con Validación



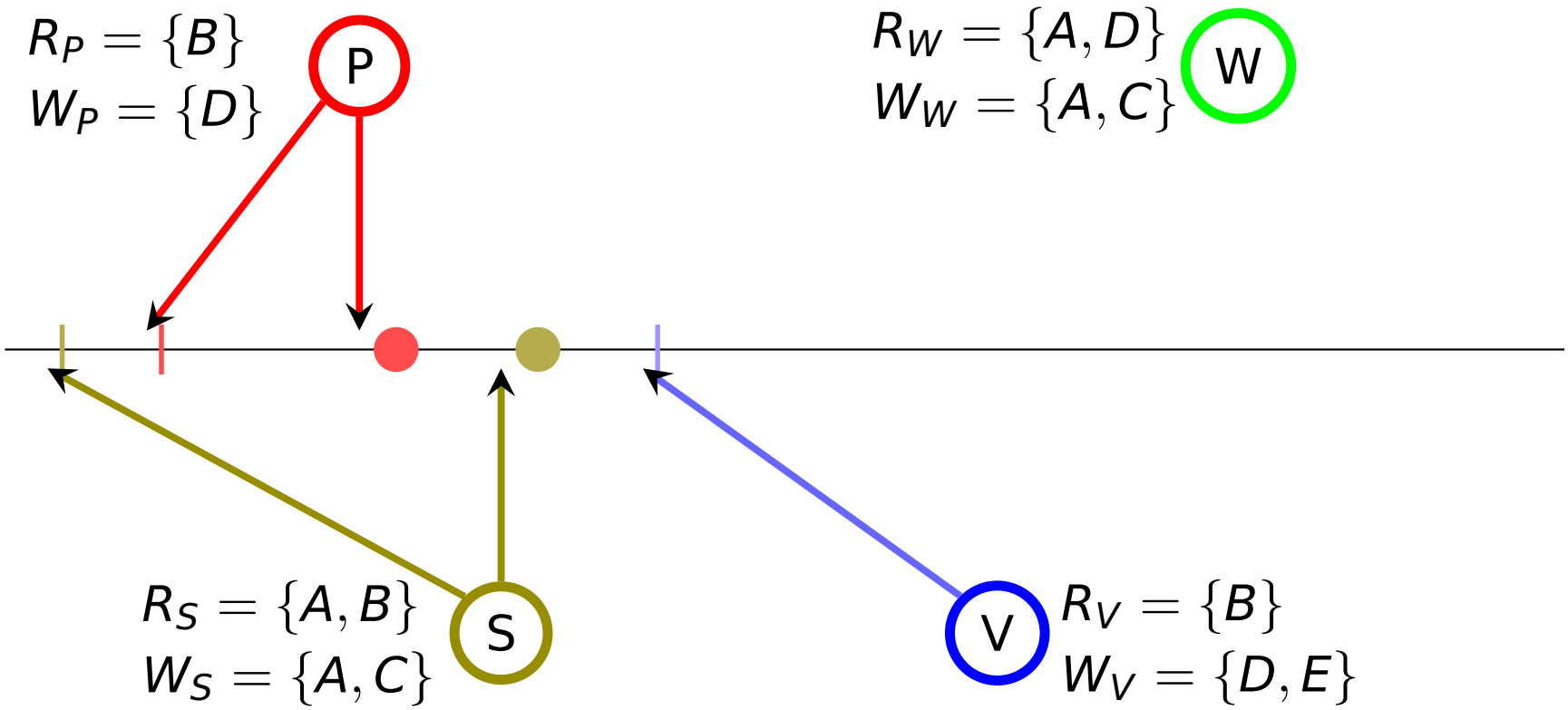
- Verificar que $R_T \cap W_U$ es vacío para cualquier transacción U validada previamente y que no finalizo antes de que T comenzara.
- Verificar $W_T \cap W_U$ es vacío para cualquier transacción U validada previamente y que no finalizo antes de que T sea validada.

Ejemplo de Planificador con Validación



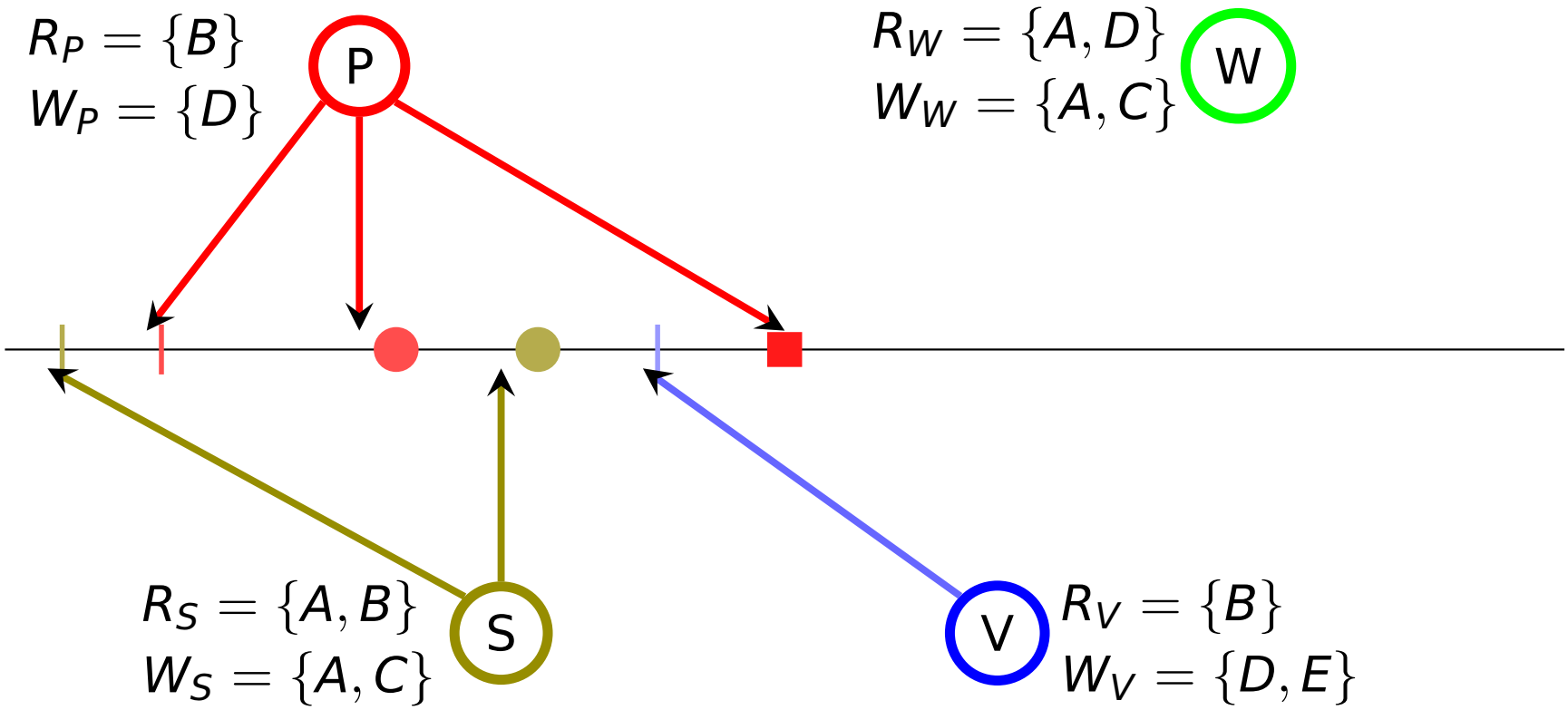
- Verificar que $R_T \cap W_U$ es vacío para cualquier transacción U validada previamente y que no finalizo antes de que T comenzara.
- Verificar $W_T \cap W_U$ es vacío para cualquier transacción U validada previamente y que no finalizo antes de que T sea validada.

Ejemplo de Planificador con Validación



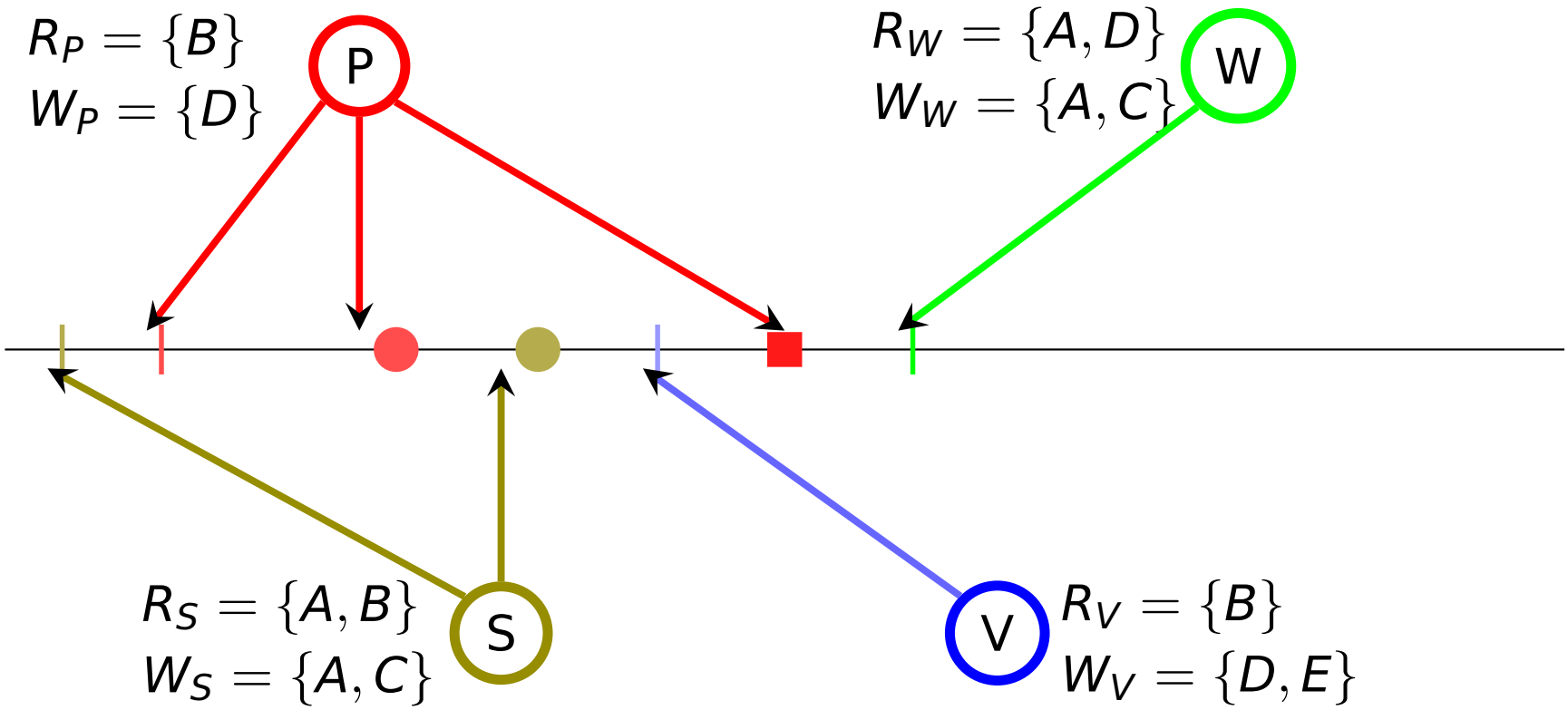
- Verificar que $R_T \cap W_U$ es vacío para cualquier transacción U validada previamente y que no finalizo antes de que T comenzara.
- Verificar $W_T \cap W_U$ es vacío para cualquier transacción U validada previamente y que no finalizo antes de que T sea validada.

Ejemplo de Planificador con Validación



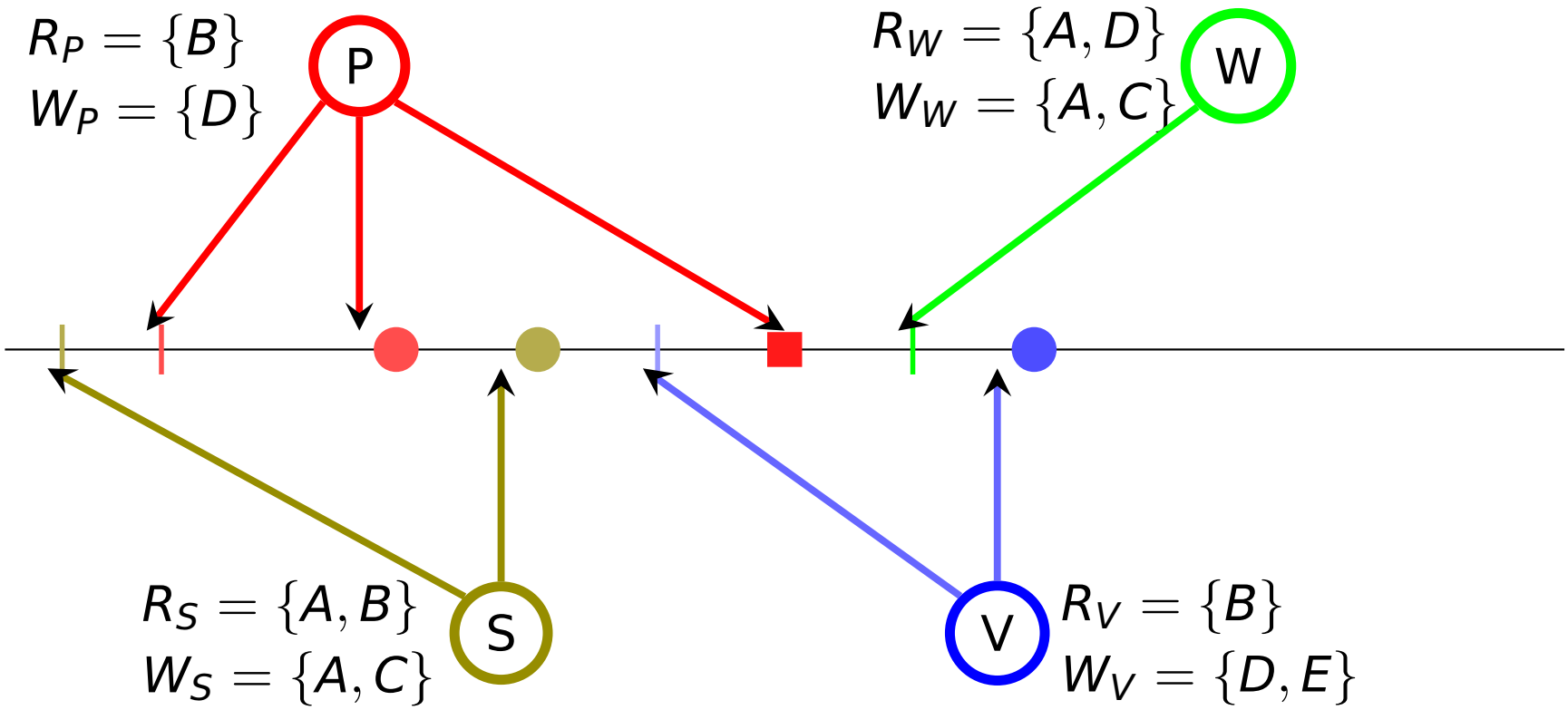
- Verificar que $R_T \cap W_U$ es vacío para cualquier transacción U validada previamente y que no finalizo antes de que T comenzara.
- Verificar $W_T \cap W_U$ es vacío para cualquier transacción U validada previamente y que no finalizo antes de que T sea validada.

Ejemplo de Planificador con Validación



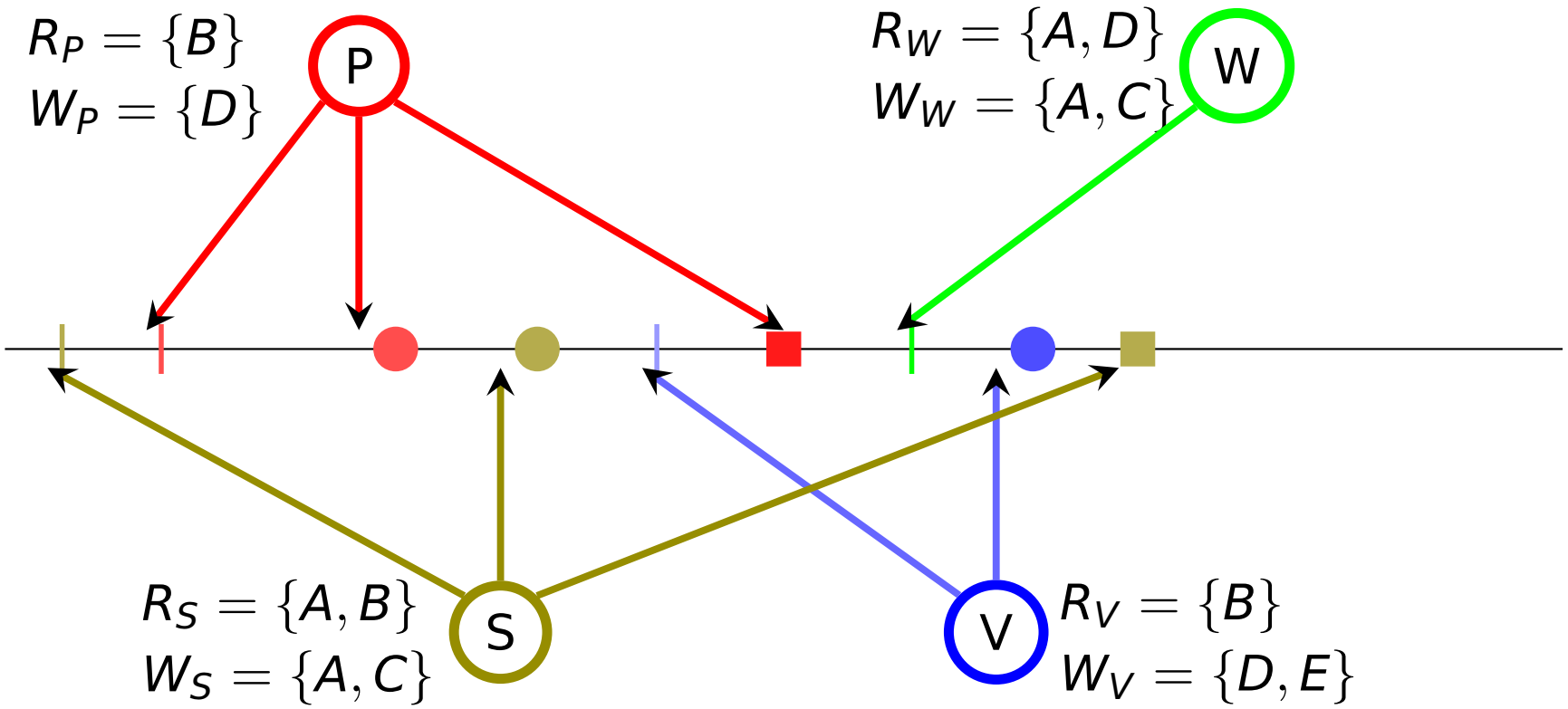
- Verificar que $R_T \cap W_U$ es vacío para cualquier transacción U validada previamente y que no finalizo antes de que T comenzara.
- Verificar $W_T \cap W_U$ es vacío para cualquier transacción U validada previamente y que no finalizo antes de que T sea validada.

Ejemplo de Planificador con Validación



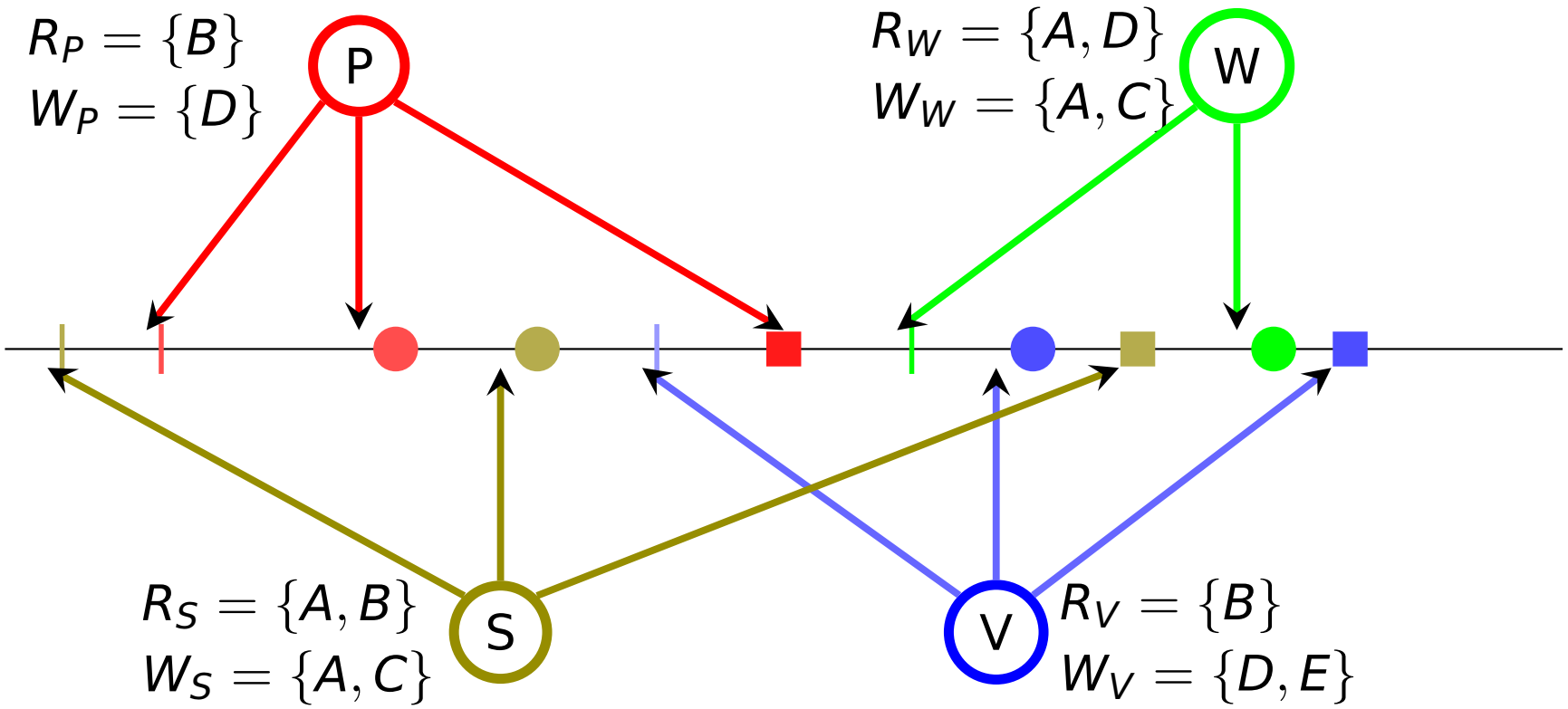
- Verificar que $R_T \cap W_U$ es vacío para cualquier transacción U validada previamente y que no finalizo antes de que T comenzara.
- Verificar $W_T \cap W_U$ es vacío para cualquier transacción U validada previamente y que no finalizo antes de que T sea validada.

Ejemplo de Planificador con Validación



- Verificar que $R_T \cap W_U$ es vacío para cualquier transacción U validada previamente y que no finalizo antes de que T comenzara.
- Verificar $W_T \cap W_U$ es vacío para cualquier transacción U validada previamente y que no finalizo antes de que T sea validada.

Ejemplo de Planificador con Validación



- Verificar que $R_T \cap W_U$ es vacío para cualquier transacción U validada previamente y que no finalizo antes de que T comenzara.
- Verificar $W_T \cap W_U$ es vacío para cualquier transacción U validada previamente y que no finalizo antes de que T sea validada.