Unidad 4

Transacciones

Agenda

- Conceptos y propiedades
- Control de concurrencia
- Recuperación ante fallos
- Transacciones en SQL

Conceptos y propiedades

Transacción de negocio vs Transacción de datos

Realizar venta

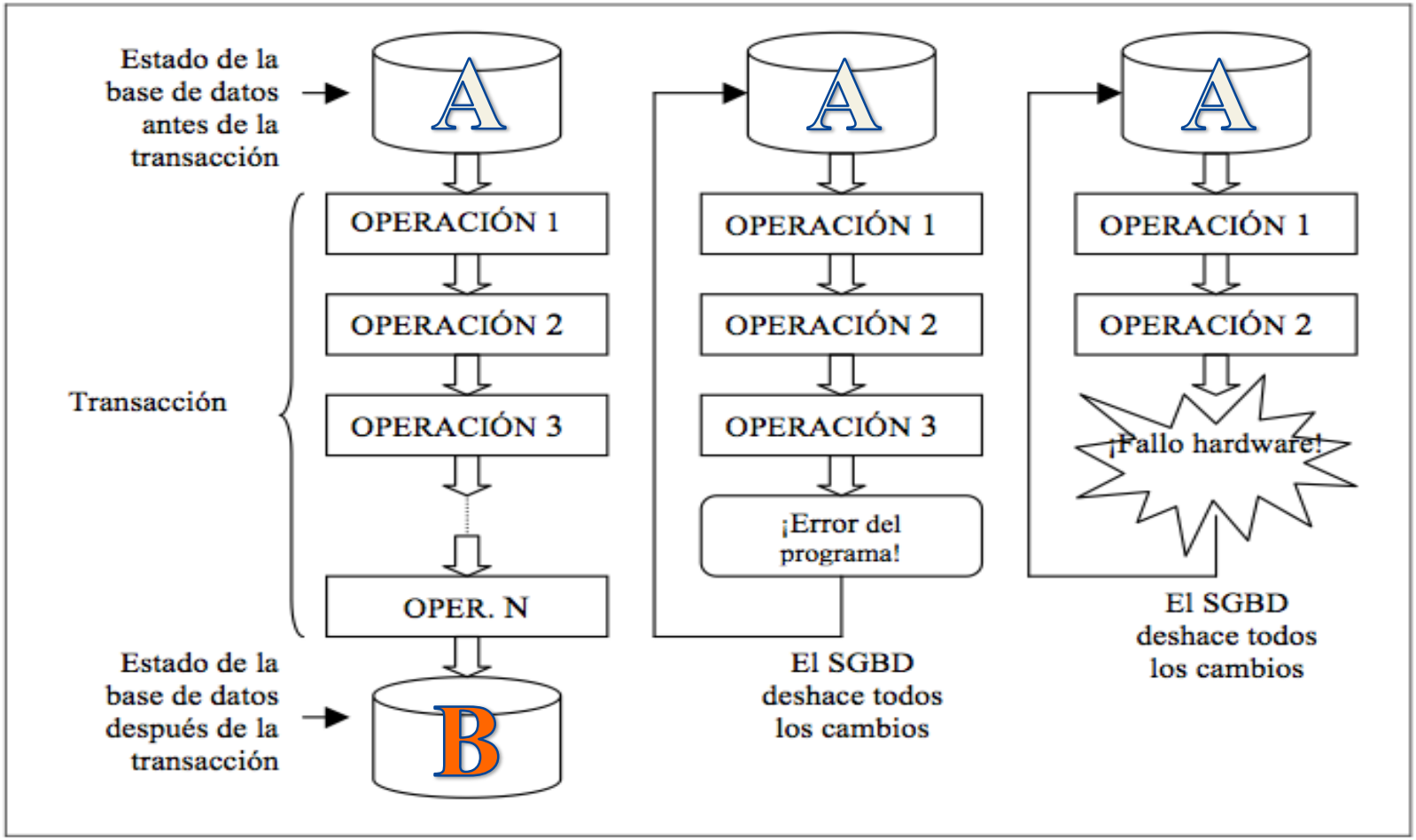
- Seleccionar productos
- Calcular valor a pagar
- Realizar el pago
- Emitir factura
- Entregar productos

Registrar venta

- Registrar datos generales de la factura
- Registrar ítems facturados
- Actualizar inventario
- Registrar pago

Conceptos y propiedades

¿Qué es una transacción de datos?



Conceptos y propiedades

¿Qué es una transacción de base de datos?

- Definición

  Conjunto de operaciones de manipulación de datos que se ejecutan como una sola unidad lógica de trabajo.

- Ejemplo

  transferencia de fondos de una cuenta corriente a una cuenta de ahorros

```
begin transaction
  debitar de la cuenta corriente
  acredidar en la cuenta de ahorros
end transaction
```

asegura la consistencia de la base de datos

Conceptos y propiedades

¿Qué es una transacción de datos?

Ejemplo "Transferir $50 de la cuenta A a la cuenta B del cliente Juan P."

Estado inicial de los datos

Tabla CUENTAS

cuenta	cliente	saldo
CA	Juan P.	1000
CB	Juan P.	2000

Saldo **INICIAL** Juan P.: $ **3000**

Transacción

```
A:= SELECT saldo
    FROM cuentas c
    WHERE c.cuenta = "CA";

A:= A - 50;

UPDATE cuentas
SET saldo = A
WHERE c.cuenta = "CA";

B:= SELECT saldo
    FROM cuentas c
    WHERE c.cuenta = "CB";

B:= B + 50;

UPDATE cuentas
SET saldo = B
WHERE c.cuenta = "CB";
```

Estado final de los datos

Tabla CUENTAS

cuenta	cliente	saldo
CA	Juan P.	**950**
CB	Juan P.	**2050**

Saldo **FINAL** Juan P.: $ **3000**

Conceptos y propiedades

Riesgo si falla la transacción

Ejemplo "Transferir $50 de la cuenta A a la cuenta B del cliente Juan P."

Estado inicial de los datos

Tabla CUENTAS

cuenta	cliente	saldo
CA	Juan P.	1000
CB	Juan P.	2000

Saldo **INICIAL** Juan P.: $ **3000**

Transacción

```
A:= SELECT saldo
    FROM cuentas c
    WHERE c.cuenta = "CA";

A:= A - 50;

UPDATE cuentas
SET saldo = A
WHERE c.cuenta = "CA";

B:= SELECT saldo
    FROM cuentas c
    WHERE c.cuenta = "CB";

B:= B + 50;

UPDATE cuentas
SET saldo = B
WHERE c.cuenta = "CB";
```

Estado final de los datos

Tabla CUENTAS

cuenta	cliente	saldo
CA	Juan P.	950
CB	Juan P.	2000

Saldo **FINAL** Juan P.: $ **2950**

Conceptos y propiedades

Gestión del fallo por el SGBD: **Deshace la transacción**

Ejemplo "Transferir $50 de la cuenta A a la cuenta B del cliente Juan P."

Estado inicial de los datos

Tabla CUENTAS

cuenta	cliente	saldo
CA	Juan P.	1000
CB	Juan P.	2000

Saldo **INICIAL** Juan P.: $ **3000**

Transacción

```
A:= SELECT saldo
    FROM cuentas c
    WHERE c.cuenta = "CA";

A:= A - 50;

UPDATE cuentas
SET saldo = A
WHERE c.cuenta = "CA";

B:= SELECT saldo
    FROM cuentas c
    WHERE c.cuenta = "CB";

B:= B + 50;

UPDATE cuentas
SET saldo = B
WHERE c.cuenta = "CB";
```

Estado final de los datos

Tabla CUENTAS

cuenta	cliente	saldo
CA	Juan P.	1000
CB	Juan P.	2000

Saldo **FINAL** Juan P.: $ **3000**

Conceptos y propiedades

Operaciones que controla una transacción

- A efectos de gestionar la transacciones, al SGBD solo le interesan dos tipos de operaciones: ***leer*** y ***escribir***
- Leer (**read**) corresponde a consulta de datos (SELECT en SQL)
- Escribir (**write**) corresponde a actualización de datos (INSERT, UPDATE y DELETE en SQL)

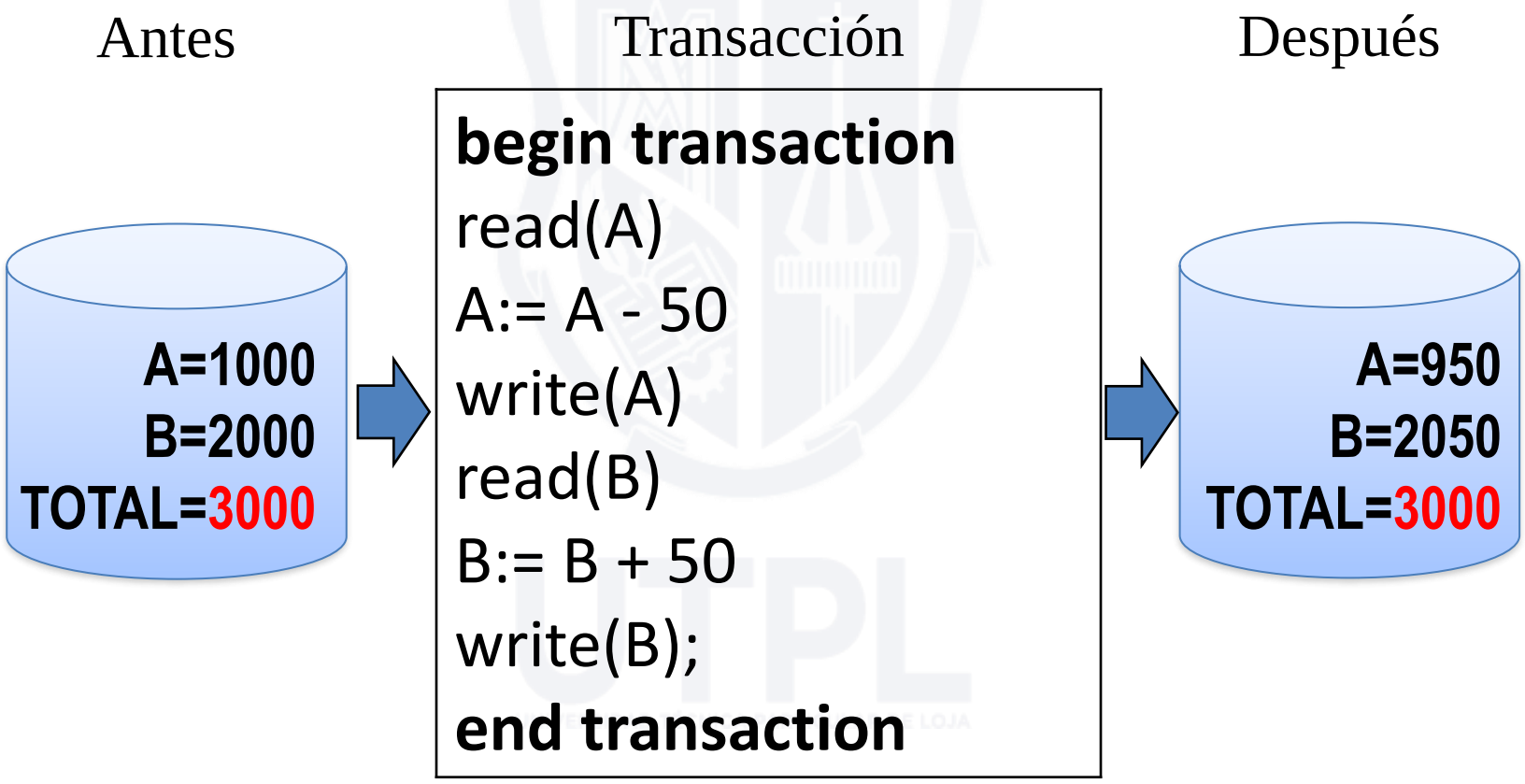
- **`read`**`(X)`

- **`write`**`(X)`

Conceptos y propiedades

Operaciones que controla una transacción

Ejemplo "Transferir $50 de la cuenta A a la cuenta B del cliente Juan P."



Conceptos y propiedades

Propiedades ACID

- En bases de datos, toda transacción cumple con 4 propiedades:
  1. **A**tomicity (Atomicidad)
  2. **C**onsistency (Consistencia)
  3. **I**solation (Aislamiento)
  4. **D**urability (Durabilidad)

Conceptos y propiedades

Atomicidad

- Una transacción es un *unidad de trabajo indivisible*
- Por tanto, **todas las operaciones de la transacción son ejecutadas completamente o ninguna**
- El ejecutar solo un subconjunto de operaciones, contradice el objetivo de una transacción

Conceptos y propiedades

Consistencia

- Una transacción es una *unidad de integridad*
- Por tanto, **la ejecución de una transacción debe preservar la consistencia de los datos**
- En el ejemplo de la transferencia bancaria: A+B debe permanecer invariable después de la ejecución de la transacción

Conceptos y propiedades

Aislamiento

- Una transacción es una *unidad de aislamiento*
- Implica que **la ejecución de una transacción no debe afectar la ejecución de otras**
- Permite la ejecución concurrente de transacciones, sin el riesgo de que las operaciones de una transacción afecte el resultado de otra (**secuencialidad**)

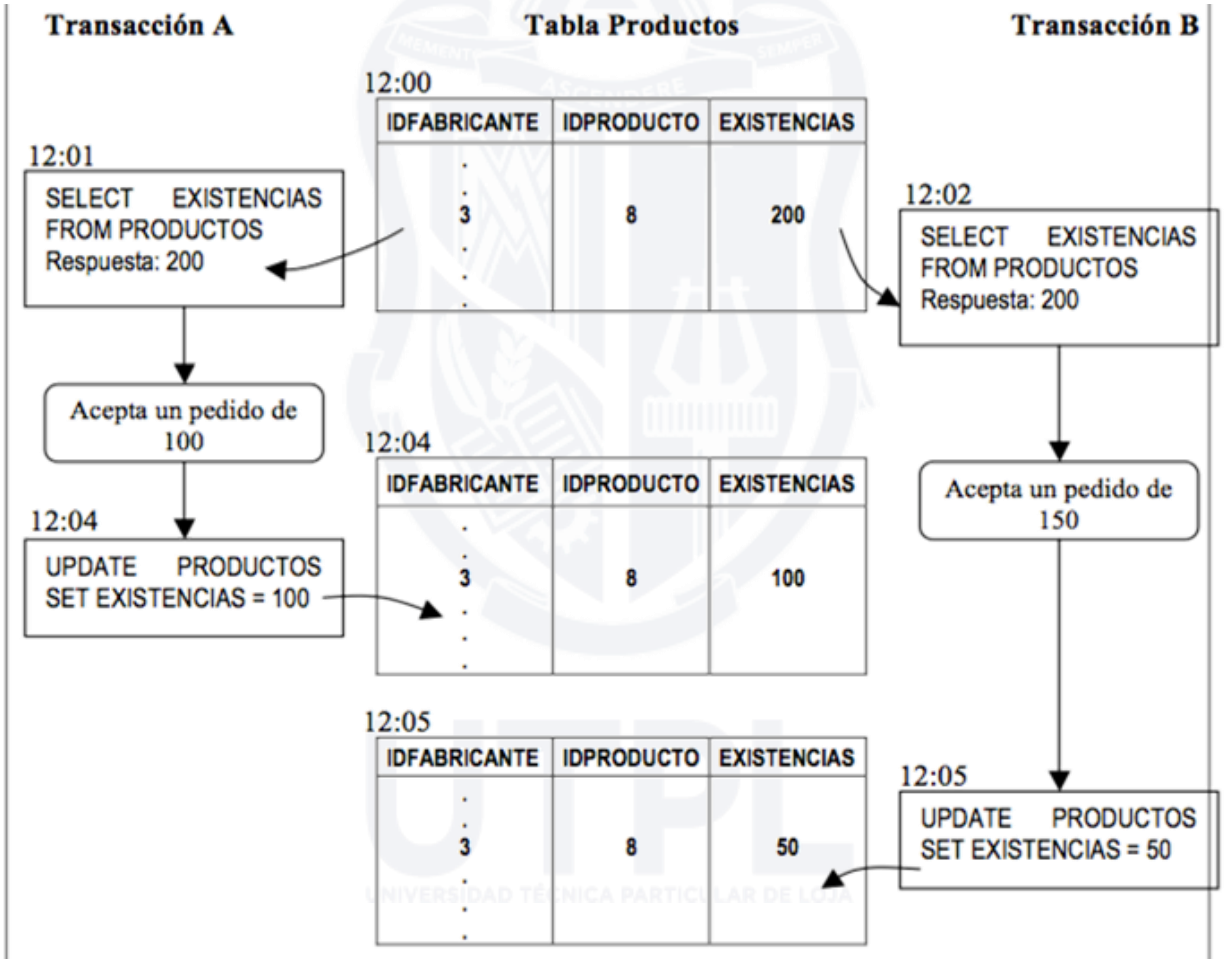
Conceptos y propiedades

Durabilidad

- Una transacción es una *unidad de recuperación*
- Si una transacción culmina exitosamente **el DBMS debe garantizar que los cambios sean permanentes**.
- Aun cuando el equipo falle luego de finalizada la transacción; al reiniciar los cambios deben estar reflejados íntegramente.

Control de concurrencia

El riesgo de la ejecución concurrente de transacciones



Control de concurrencia

Ejecuciones concurrentes

- El SGBD ejecuta un conjunto de transacciones de manera concurrente preservando la consistencia de los datos.
- Esquemas de control de concurrencia
  - Mecanismos para controlar la interacción entre las transacciones concurrentes para prevenir la destrucción de la consistencia de la base de datos
- Ventajas
  - Incremento de la utilización del procesador y del disco
  - Reducción del *tiempo promedio de respuesta* para la transacciones

Control de concurrencia

Ejecuciones concurrentes

- Para realizar una ejecución concurrente el SGBD diseña una planificación.
- **Planificación**:
  - Secuencia que indica el orden cronológico en el cuál las instrucciones de un conjunto de transacciones son ejecutadas.
  - Compuesta de todas las instrucciones de las transacciones
  - Debe preservar el orden en el que las instrucciones aparecen en cada transacción individual

Control de concurrencia

Ejecuciones concurrentes

- Ejemplo

  T_1 transfiere \$50 de A a B, y T_2 transfiere 10% del saldo (*balance*) de A a B.

```
T1:  read(A);
     A := A-50;
     write(A);
     read(B);
     B := B+50;
     write(B).
```

```
T2:  read(A);
     temp := A*0.1;
     A := A-temp;
     write(A);
     read(B);
     B := B+temp;
     write(B).
```

Control de concurrencia

Planificación secuencial

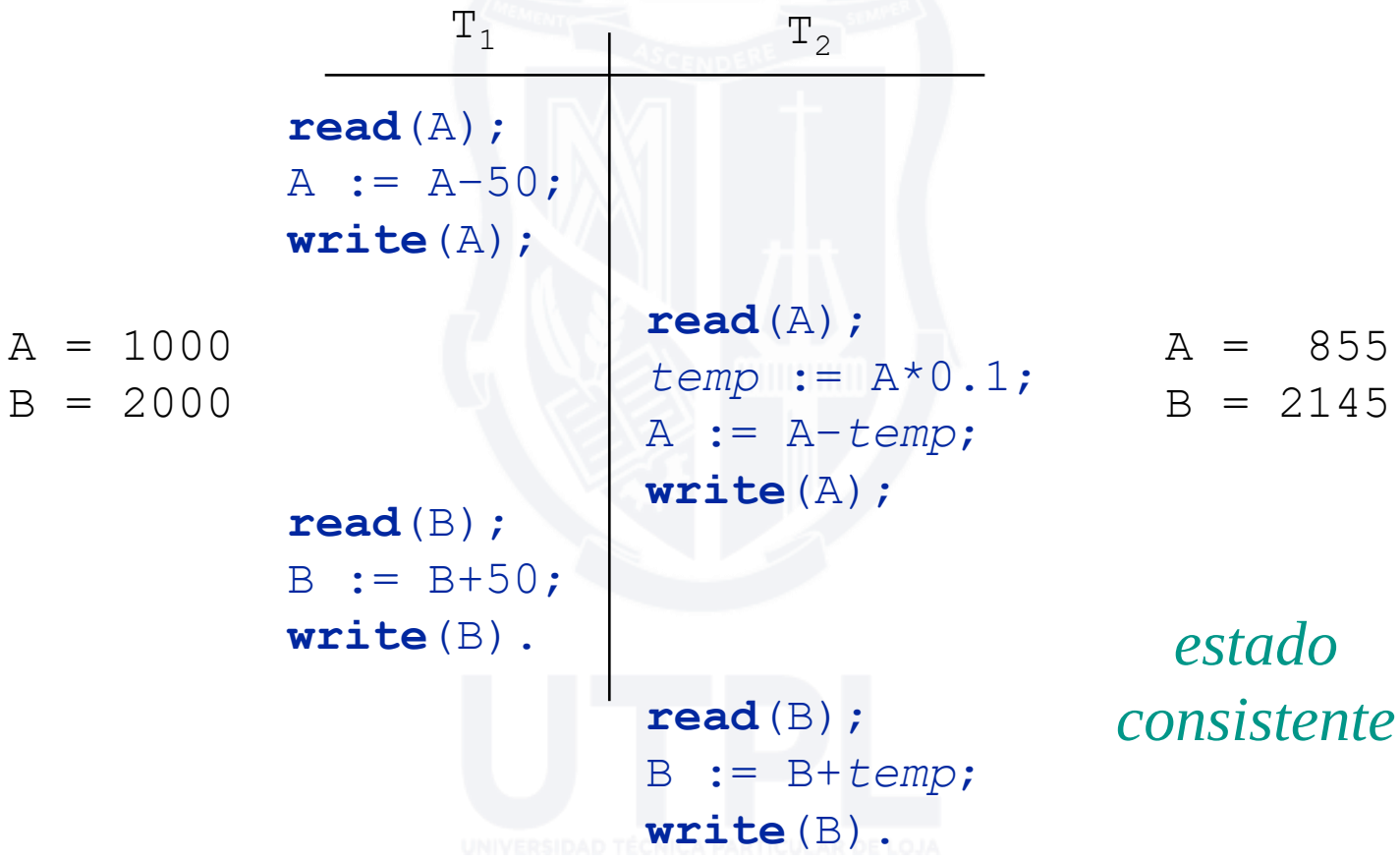
A = 1000
B = 2000

T_1	T_2
read(A);	
A := A-50;	
write(A);	
read(B);	
B := B+50;	
write(B).	
	read(A);
	temp := A*0.1;
	A := A-*temp*;
	write(A);
	read(B);
	B := B+*temp*;
	write(B).

A = 855
B = 2145

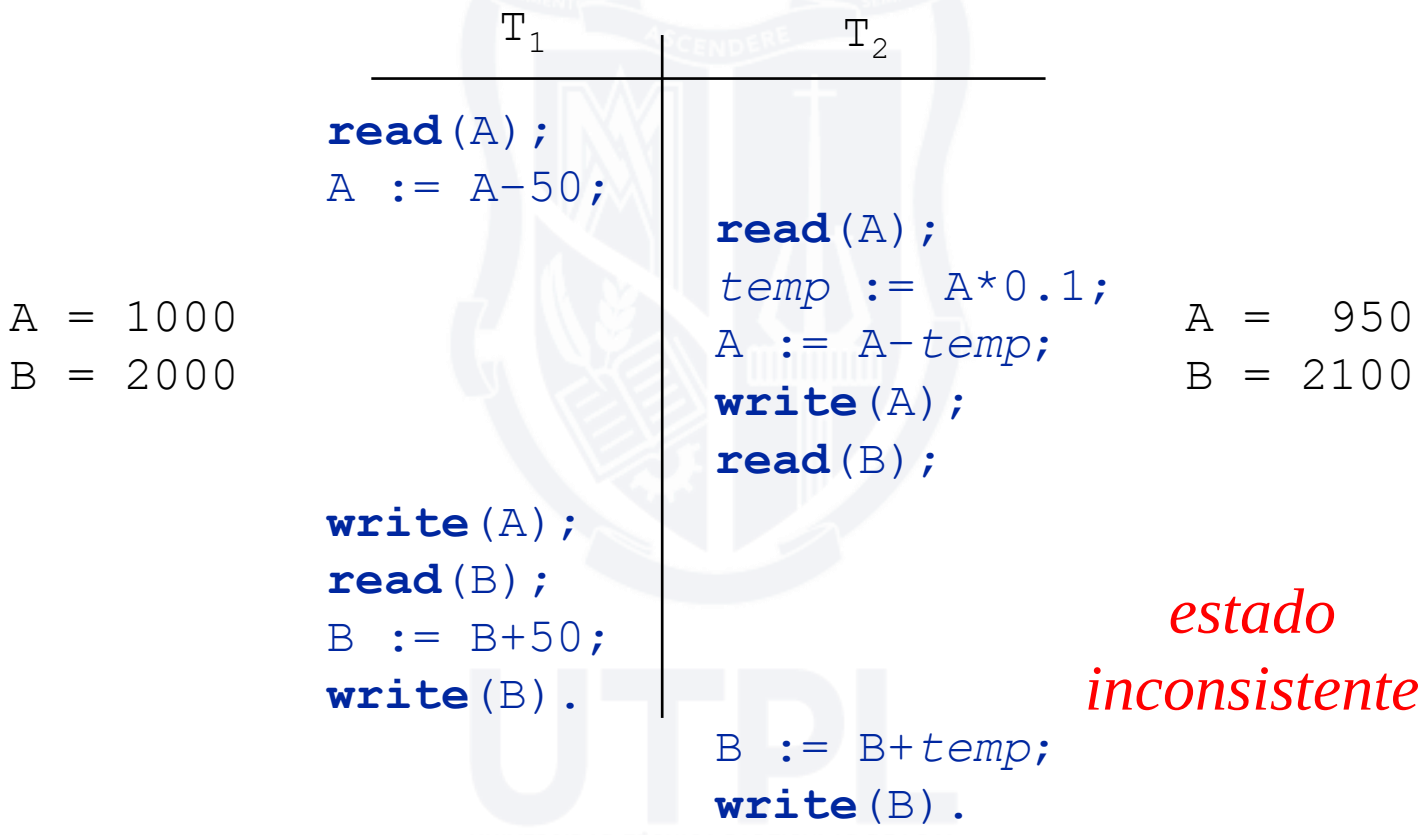
Control de concurrencia

Planificación concurrente secuenciable



Control de concurrencia

Planificación concurrente no secuenciable



Control de concurrencia

Secuencialidad

- La secuencialidad es la propiedad que una planificación concurrente debe cumplir con el fin de asegurar la consistencia de los datos, debiendo ser equivalente a una planificación secuencial.

Control de concurrencia

Planificaciones equivalentes

- Dos planificaciones son equivalentes si ambas llevan a la base de datos desde un mismo estado inicial a mismo estado final
- Una planificación secuenciable es **equivalente** a una planificación secuencial
- Por tanto, si una planificación secuencial lleva a la base de datos de un estado consistente a otro, la planificación secuenciable también lo hará

Control de concurrencia

Secuencialidad

- Existen dos formas de secuencialidad:
  - *Secuencialidad en cuanto a conflictos*
  - *Secuencialidad en cuanto a vistas*

Control de concurrencia

Secuencialidad en cuanto a conflictos

- *S*: planificación secuencial
- I_i, I_m: instrucciones
- T_i, T_m: transacciones
- i≤m
- Si I_i e I_m hacen referencia a datos diferentes, entonces es posible intercambiar I_i e I_m sin afectar el resultado
- Si I_i e I_m hacen referencia al mismo dato *Q*, entonces el orden de estas instrucciones puede importar

Control de concurrencia

Secuencialidad en cuanto a conflictos

casos:

1. $I_i = \mathbf{read}(Q)$, $I_m = \mathbf{read}(Q)$

   el orden de I_i e I_m no importa

2. $I_i = \mathbf{read}(Q)$, $I_m = \mathbf{write}(Q)$

   Si I_i antecede a I_m, entonces T_i no lee el valor de Q escrito por T_m en la instrucción I_m

   Si I_m antecede a I_i, entonces T_i lee el valor de Q escrito por T_m en la instrucción I_m

   Por lo tanto, el orden importa

Control de concurrencia

Secuencialidad en cuanto a conflictos

3. $I_i = \mathbf{write}(Q)$, $I_m = \mathbf{read}(Q)$

   el orden de I_i e I_m importa por las mismas razones que el caso anterior

4. $I_i = \mathbf{write}(Q)$, $I_m = \mathbf{write}(Q)$

   si importa el orden porque afecta al resultado final de Q

Control de concurrencia

Secuencialidad en cuanto a conflictos

- conflicto entre I_i and I_j
  - Se da cuando son operaciones consecutivas de diferentes transacciones sobre el mismo elemento de dato y al menos una de estas instrucciones es una operación de escritura

Control de concurrencia

Secuencialidad en cuanto a conflictos

- supongamos que I_i e I_m son instrucciones consecutivas de la planificación S.
- si I_i e I_m son instrucciones de diferentes transacciones y *no hay conflicto* entre ellas, entonces podemos intercambiar el orden de I_i e I_m para obtener una nueva planificación S´
- Se puede decir que S es equivalente a S´ dado que todas las instrucciones aparecen en el mismo orden en ambas planificaciones excepto para I_i e I_m, cuyo orden no tiene importancia

Control de concurrencia

Secuencialidad en cuanto a conflictos

T_1	T_2
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A
	read(B)
	write(B)

*Planificación **S** (**secuencial**)*

Control de concurrencia

Secuencialidad en cuanto a conflictos

T_1	T_2
read(A);	
write(A);	
	read(A);
	write(A);
read(B);	
write(B).	
	read(B);
	write(B).

*Planificación **S'** **(secuenciable en términos de conflictos)***

Control de concurrencia

Secuencialidad en cuanto a vistas

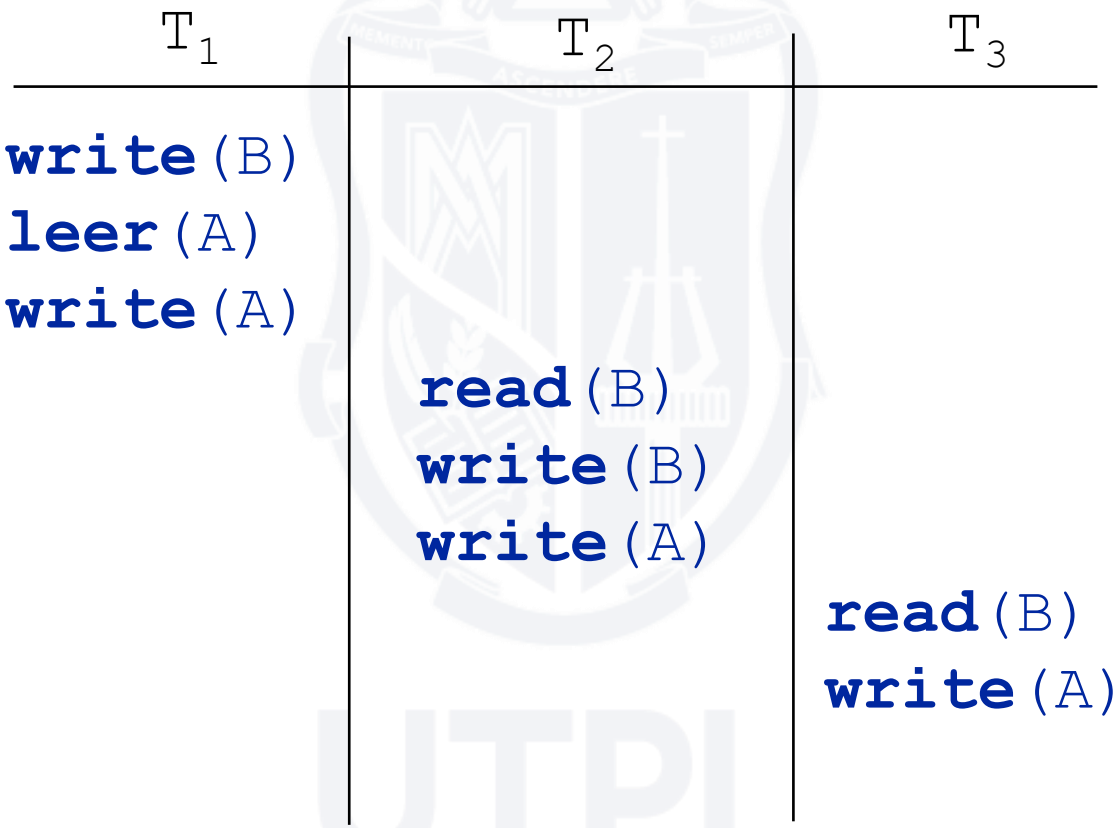
a) Si la transacción Ti lee el valor inicial de Q en **S**, debería hacerlo en **S'**.

b) Si la transacción Ti ejecuta leer(Q) y el valor lo ha producido en **S** la transacción Tj (si hay), debería pasar lo mismo en **S'**.

c) La transacción (si hay) que realice la última operación escribir(Q) en **S** debería hacerlo en **S'**.

Control de concurrencia

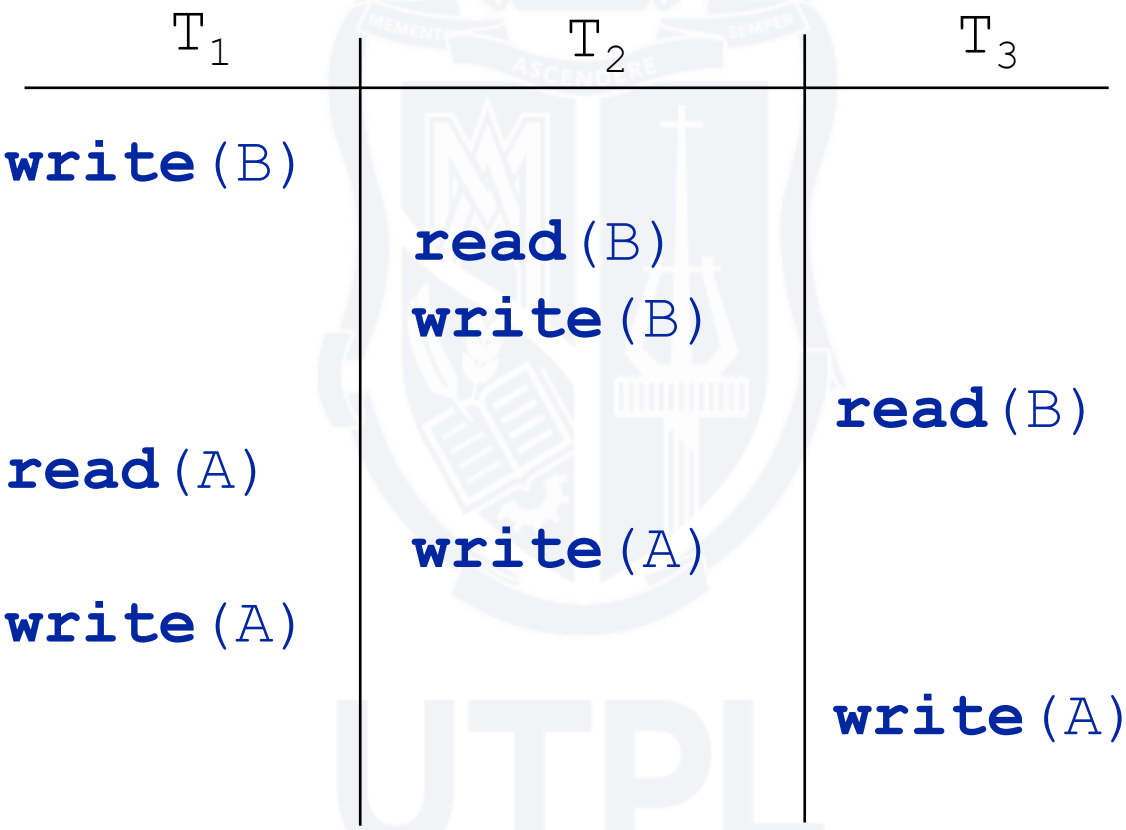
Secuencialidad en cuanto a vistas



Planificación secuencial

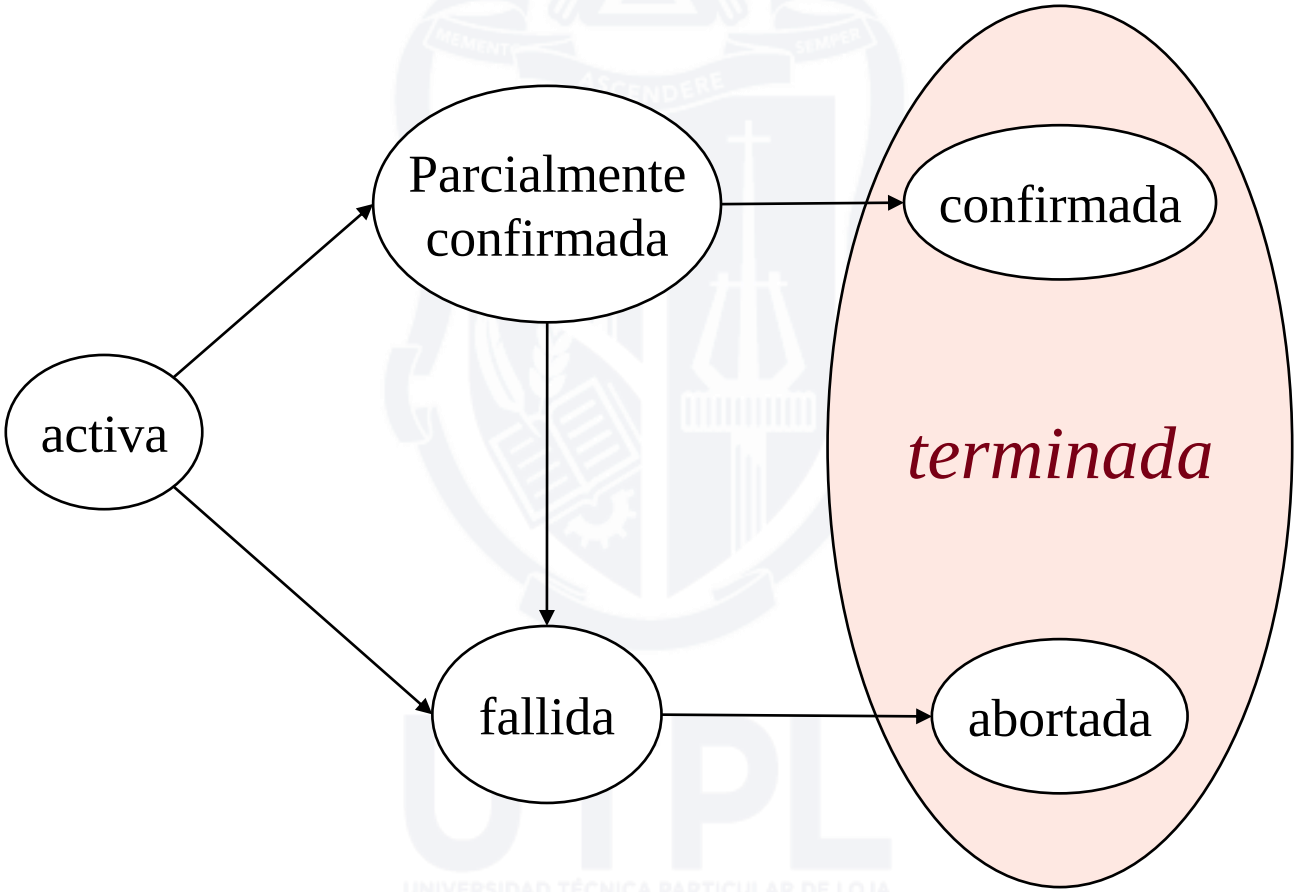
Control de concurrencia

Secuencialidad en cuanto a vistas



Planificación secuenciable en cuanto a vistas

Estados de una transacción



Estados de una transacción

Estados de una transacción

- *activa*
  - estado inicial
  - la transacción permanece en este estado mientras está ejecutándose
- *parcialmente confirmada*
  - después que la última instrucción de la transacción ha sido ejecutada
- *fallida*
  - después de descubrir que la ejecución normal no puede proseguir

Conceptos y propiedades

Estados de una transacción

- *abortada*
  - después que la transacción ha sido “deshecha” y la base de datos es restaurada a un estado anterior al inicio de la transacción
    - reiniciar la transacción
    - suprimir la transacción
- *confirmada*
  - después de haber completado exitosamente la transacción

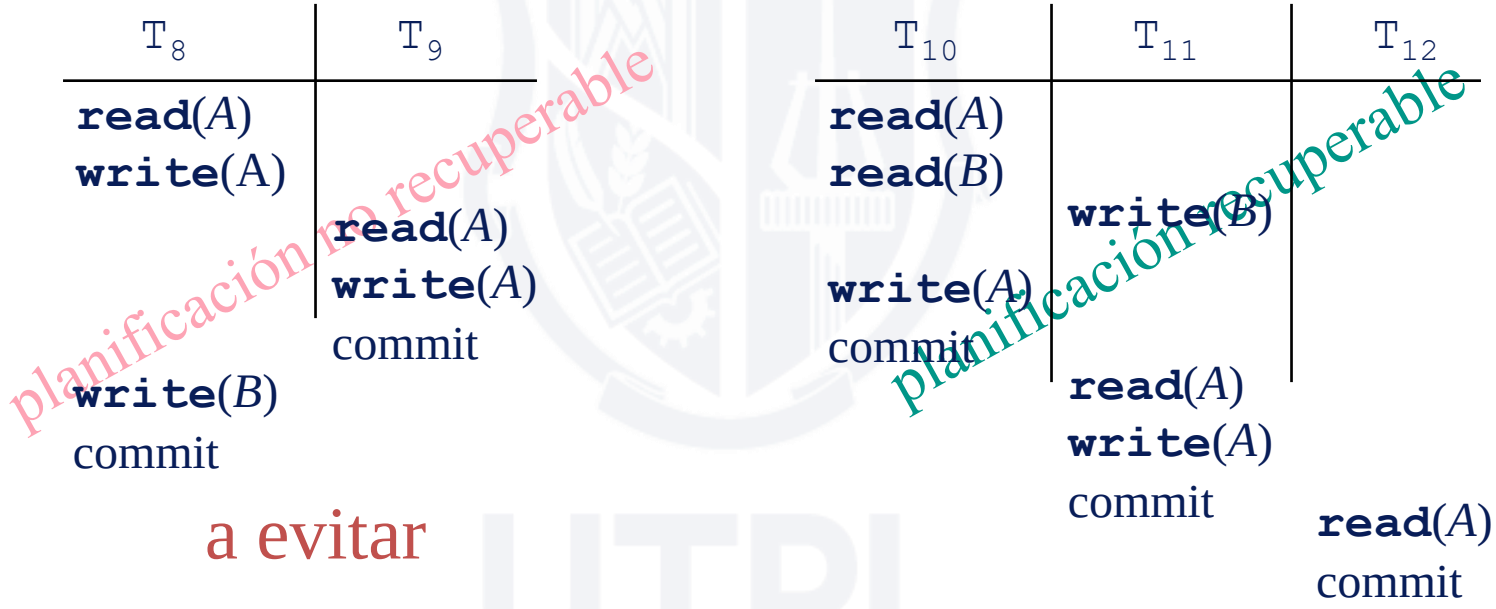
Recuperación ante fallos

- en caso de fallas de las transacciones
  - si una transacción T_i falla, por cualquier razón, es necesario deshacer el efecto de esta transacción para asegurar la atomicidad
  - en ejecuciones concurrentes, es necesario asegurar que cualquier transacción T_m dependiente de T_i es también abortada

Recuperación ante fallos

Planificación recuperable

Si una transacción T_m lee datos previamente escritos por una transacción T_i , el **commit** de T_i debe aparecer antes del **commit** de T_m



Toda planificación secuencial es recuperable

Recuperación ante fallos

Retrocesos en cascada (cascading rollback)

Una falla de una transacción simple puede conducir a una serie de *transaction rollbacks*

T_1	T_2	T_3
read(A)		
read(B)		
write(A)		
	read(A)	
	write(A)	
		read(A)
read(C)		
commit		
	write(C)	
	commit	
		write(C)
		commit

Planificación secuenciable

conduce a deshacer una cantidad significativa de trabajo

¿deseable cascading rollback?

Recuperación ante fallos

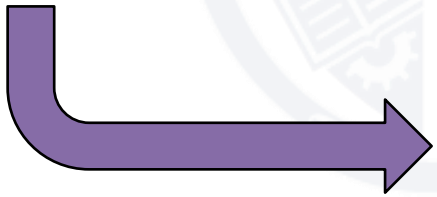
Retrocesos en cascada (cascading rollback)

- **rollback**: operación que regresa la base de datos a su estado previo
- es deseable restringir las planificaciones a aquellas donde ***retrocesos en cascada*** no pueden ocurrir
- para cada par de transacciones T_i y T_m tal que T_m lee un dato previamente escrito por T_i, el **`commit`** de T_i aparece antes del **`read`** de T_m
- toda *planificación en cascada* es recuperable

Recuperación ante fallos

Ejemplo

Transacciones		
T1	T2	T3
read(A)	read(B)	write(B)
read(B)	write(B)	read(A)
write(A)	read(A)	write(A)
read(C)	write(A)	commit
commit	write(C)	
	commit	



Planificación Secuencial		
T1	T2	T3
read(A)		
read(B)		
write(A)		
read(C)		
commit		
	read(B)	
	write(B)	
	read(A)	
	write(A)	
	write(C)	
	commit	
		write(B)
		read(A)
		write(A)
		commit

Recuperación ante fallos

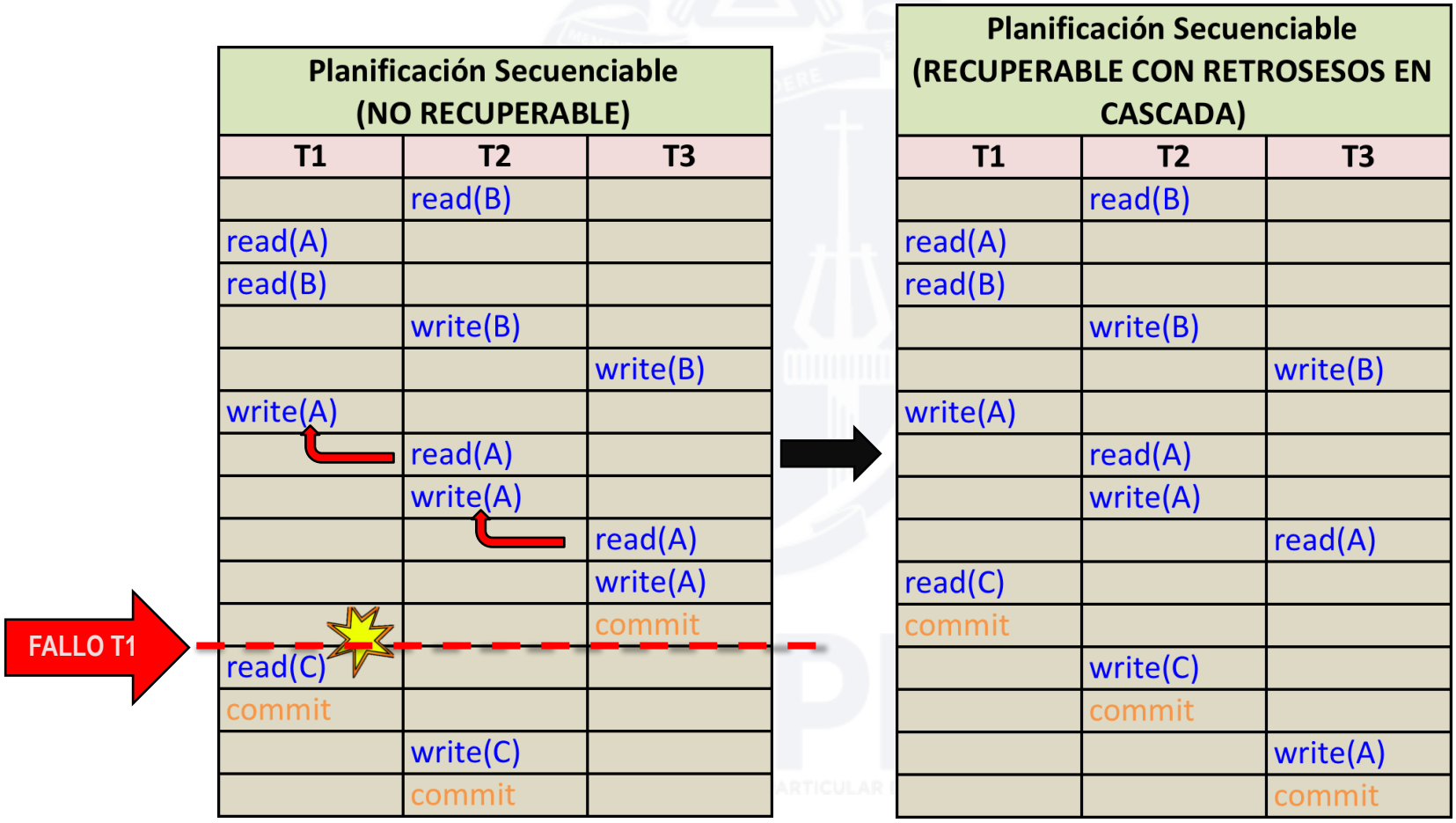
Ejemplo

Planificación Secuencial		
T1	T2	T3
read(A)		
read(B)		
write(A)		
read(C)		
commit		
	read(B)	
	write(B)	
	read(A)	
	write(A)	
	write(C)	
	commit	
		write(B)
		read(A)
		write(A)
		commit

Planificación Secuenciable (NO RECUPERABLE)		
T1	T2	T3
	read(B)	
read(A)		
read(B)		
	write(B)	
		write(B)
write(A)		
	read(A)	
	write(A)	
		read(A)
		write(A)
		commit
read(C)		
commit		
	write(C)	
	commit	

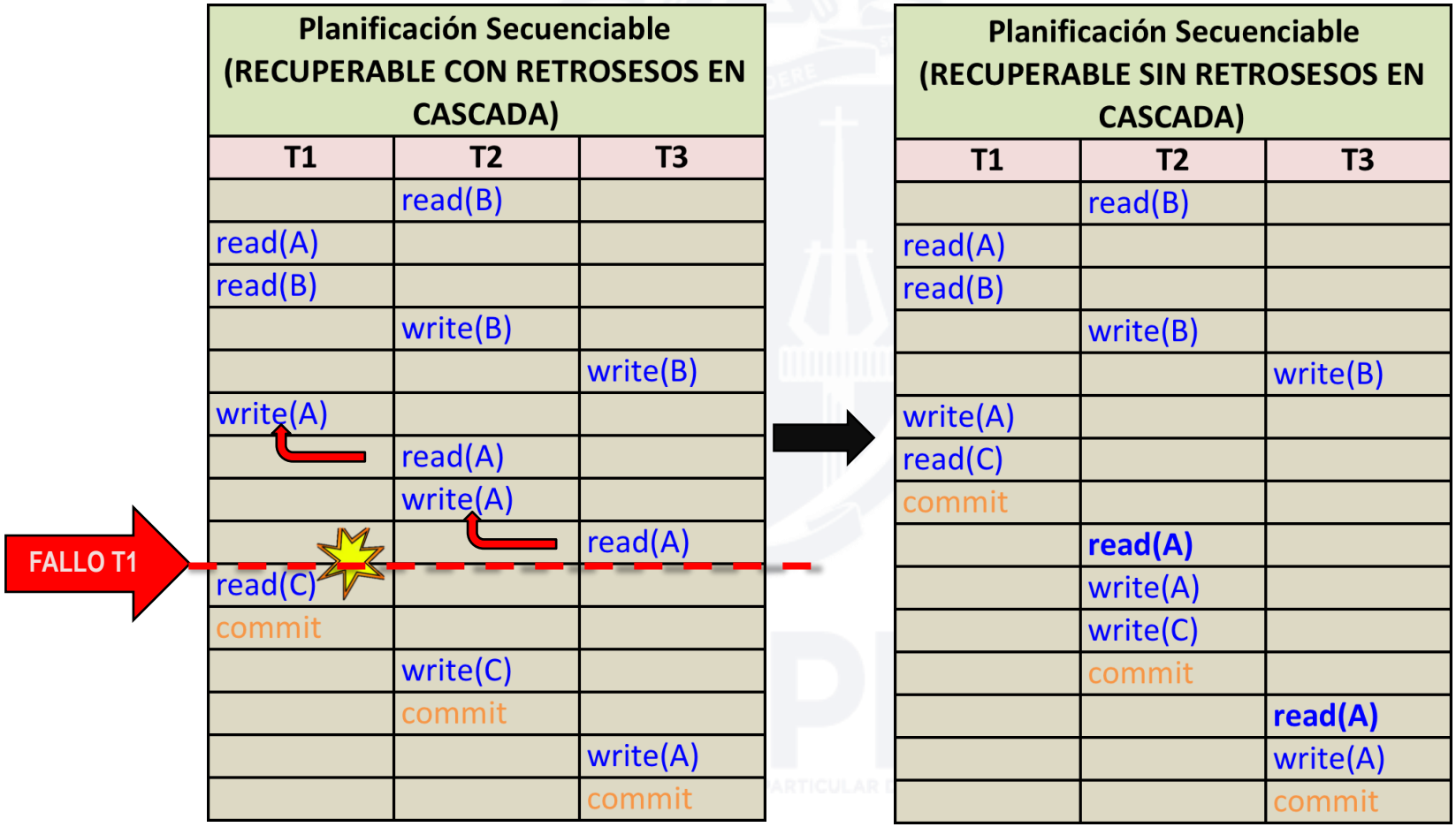
Recuperación ante fallos

Ejemplo



Recuperación ante fallos

Ejemplo



Planificación Secuenciable (RECUPERABLE CON RETROSESOS EN CASCADA)

T1	T2	T3
	read(B)	
read(A)		
read(B)		
	write(B)	
		write(B)
write(A)		
	read(A)	
	write(A)	
		read(A)
read(C)		
commit		
	write(C)	
	commit	
		write(A)
		commit

Planificación Secuenciable (RECUPERABLE SIN RETROSESOS EN CASCADA)

T1	T2	T3
	read(B)	
read(A)		
read(B)		
	write(B)	
		write(B)
write(A)		
read(C)		
commit		
	read(A)	
	write(A)	
	write(C)	
	commit	
		read(A)
		write(A)
		commit

Recuperación ante fallos

- Para materializar la ejecución de un rollback, lo motores relacionales usan dos métodos alternativos:
  - Rollback por bitácora (log-based rollback)
  - Rollback por modificación diferida (deferred modification)
- La mayoría de motores usan el método log-based rollback.