

CC3201-1

BASES DE DATOS

OTOÑO 2019

Clase 11: Transacciones y ACID

Aidan Hogan

[aidhog@gmail.com](mailto:aidhog@gmail.com)

# Una cuenta bancaria ...

## Ingreso

<u>cuenta</u>	<u>comentario</u>	<u>fecha</u>	<u>hora</u>	<u>monto</u>	<u>saldo</u>	<u>id</u>
7873698669	Deposito inicial	2020-01-21	20:02:02	300000	300000	TRCXGU8JSHD
7873698669	C0°0°L Designs	2020-02-06	09:15:33	50000	325000	TRCCIA2J8A0

## Gasto

<u>cuenta</u>	<u>comentario</u>	<u>fecha</u>	<u>hora</u>	<u>monto</u>	<u>saldo</u>	<u>id</u>
7873698669	Electricidad	2020-02-02	20:00:01	8200	291800	TRCJASJDA9A
7873698669	Calefacción	2020-02-02	20:00:02	600	291200	TRC81KAQWAS
7873698669	Moviestar	2020-02-02	20:00:03	16200	275000	TRCK8J7JA8D
7873698669	Cajero	2020-02-08	16:05:02	100000	225000	TRCPM8A45AD

## Cuenta

<u>número</u>	<u>rut</u>	<u>tipo</u>	<u>saldo_clp</u>	<u>saldo_usd</u>
7873698669	32.000.273-K	Estacional	225000	344,42

## Divisa

<u>d1</u>	<u>d2</u>	<u>valor</u>
CLP	USD	0,0001533
USD	CLP	652,2750000

## Cliente

<u>rut</u>	<u>nombre</u>	<u>fono</u>	<u>dirección</u>
32.000.273-K	Kelvin	+56976698463	Campo de Hielo Sur, Depto 273

# Una cuenta bancaria ... integridad

## Ingreso

<u>cuenta</u>	<u>comentario</u>	<u>fecha</u>	<u>hora</u>	<u>monto</u>	<u>saldo</u>	<u>id</u>
7873698669	Deposito inicial	2020-01-21	20:02:02	300000	300000	TRCXGU8JSHD
7873698669	C0°0°L Designs	2020-02-06	09:15:33	50000	325000	TRCCIA2J8A0

## Gasto

<u>cuenta</u>	<u>comentario</u>	<u>fecha</u>	<u>hora</u>	<u>monto</u>	<u>saldo</u>	<u>id</u>
7873698669	Electricidad	2020-02-02	20:00:01	8200	291800	TRCJASJDA9A
7873698669	Calefacción	2020-02-02	20:00:02	600	291200	TRC81KAQWAS
7873698669	Moviestar	2020-02-02	20:00:03	16200	275000	TRCK8J7JA8D
7873698669	Cajero	2020-02-08	16:05:02	100000	225000	TRCPM8A45AD

## Cuenta

<u>número</u>	<u>rut</u>	<u>tipo</u>	<u>saldo_clp</u>	<u>saldo_usd</u>
7873698669	32.000.273-K	Estacional	225000	344,42

## Divisa

<u>d1</u>	<u>d2</u>	<u>valor</u>
CLP	USD	0,0001533
USD	CLP	652,2750000

## Cliente

<u>rut</u>	<u>nombre</u>	<u>fono</u>	<u>dirección</u>
32.000.273-K	Kelvin	+56976698463	Campo de Hielo Sur, Depto 273

# Restricciones sobre varias tablas (!!)

## Ingreso

cuenta	comentario	fecha	hora	monto	saldo	id
7873698669	Deposito inicial	2020-01-21	20:02:02	300000	300000	TRCXGU8JSHD
7873698669	C0°0°L Designs	2020-02-06	09:15:33	50000	325000	TRCCIA2J8A0

## Gasto

cuenta	comentario	fecha	hora	monto	saldo	id
7873698669	Electricidad	2020-02-02	20:00:01	8200	291800	TRCJASJDA9A
7873698669	Calefacción	2020-02-02	20:00:02	600	291200	TRC81KAQWAS
7873698669	Moviestar	2020-02-02	20:00:03	16200	275000	TRCK8J7JA8D
7873698669	Cajero	2020-02-08	16:05:02	100000	225000	TRCPM8A45AD

## Cuenta

número	rut	tipo	saldo_clp	saldo_usd
7873698669	32.000.273-K	Estacional	225000	344,94

```
CREATE TABLE Cuenta (  
  número INTEGER PRIMARY KEY,  
  rut VARCHAR (12) NOT NULL,  
  tipo VARCHAR (12) NOT NULL,  
  saldo_clp BIGINT NOT NULL,  
  saldo_usd FLOAT NOT NULL,  
  CHECK (  
    ( SELECT SUM(monto) FROM Ingreso WHERE cuenta=número )  
    - ( SELECT SUM(monto) FROM Gasto WHERE cuenta=número )  
    = saldo_clp ) )
```



¿A. P. A.? ...

INSERT INTO Ingreso ...

INSERT INTO Gasto ...

UPDATE Cuenta  
SET saldo\_clp ...

¿A. S.? ...

TRANSACCIONES

# Transacciones

Una **transacción** es

un **conjunto de operaciones** que

se ejecutan de **manera atómica**

(es decir, **como si fuera una sola operación**)

# Vacaciones ...



# Transacciones: START TRANSACTION/COMMIT

## Gasto

<u>cuenta</u>	<u>comentario</u>	<u>fecha</u>	<u>hora</u>	<u>monto</u>	<u>saldo</u>	<u>id</u>
7873698669	Electricidad	2020-02-02	20:00:01	8200	291800	TRCJASJDA9A
7873698669	Calefacción	2020-02-02	20:00:02	600	291200	TRC81KAQWAS
7873698669	Moviestar	2020-02-02	20:00:03	16200	275000	TRCK8J7JA8D
7873698669	Cajero	2020-02-08	16:05:02	100000	225000	TRCPM8A45AD
7873698669	Noruega	2020-02-12	02:14:20	400000	-175000	TRCLK9K24KS

## Cuenta

<u>número</u>	<u>rut</u>	<u>tipo</u>	<u>saldo_clp</u>	<u>saldo_usd</u>
7873698669	32.000.273-K	Estacional	-175000	-268,29

## Divisa

<u>d1</u>	<u>d2</u>	<u>valor</u>
CLP	USD	0,001533
USD	CLP	652,275000

## Cliente

<u>rut</u>	<u>nombre</u>	<u>fono</u>	<u>dirección</u>
32.000.273-K	Kelvin	+56976698463	Campo de Hielo Sur, Depto 273

```
START TRANSACTION;  
INSERT INTO Gasto VALUES  
  (7873698669, 'Noruega', '2020-02-12', '02:14:20', 400000, -175000, 'TRCLK9K24KS');  
UPDATE Cuenta SET saldo_clp=-175000, saldo_usd=-268.29 WHERE número=7873698669;  
COMMIT;
```

**START TRANSACTION** (o a veces **BEGIN**) inicia la transacción  
**COMMIT** realiza/guarda los cambios

# Transacciones (por defecto)

## Gasto

<u>cuenta</u>	<u>comentario</u>	<u>fecha</u>	<u>hora</u>	<u>monto</u>	<u>saldo</u>	<u>id</u>
7873698669	Electricidad	2020-02-02	20:00:01	8200	291800	TRCJASJDA9A
7873698669	Calefacción	2020-02-02	20:00:02	600	291200	TRC81KAQWAS
7873698669	Moviestar	2020-02-02	20:00:03	16200	275000	TRCK8J7JA8D
7873698669	Cajero	2020-02-08	16:05:02	100000	225000	TRCPM8A45AD
7873698669	Noruega	2020-02-12	02:14:20	400000	-175000	TRCLK9K24KS

## Cuenta

<u>número</u>	<u>rut</u>	<u>tipo</u>	<u>saldo_clp</u>	<u>saldo_usd</u>
7873698669	32.000.273-K	Estacional	-175000	-268,29

## Divisa

<u>d1</u>	<u>d2</u>	<u>valor</u>
CLP	USD	0,001533
USD	CLP	652,275000

## Cliente

<u>rut</u>	<u>nombre</u>	<u>fono</u>	<u>dirección</u>
32.000.273-K	Kelvin	+56976698463	Campo de Hielo Sur, Depto 273

```
INSERT INTO Gasto VALUES
```

```
(7873698669, 'Noruega', '2020-02-12', '02:14:20', 400000, -175000, 'TRCLK9K24KS');
```

```
-- COMMIT;
```

```
UPDATE Cuenta SET saldo_clp=-175000, saldo_usd=-268.29 WHERE número=7873698669;
```

```
-- COMMIT;
```

Si no hay una transacción explícita, por defecto, Postgres hace un **COMMIT** después de cada sentencia (pero se puede cambiar la configuración)

# Transacciones: ROLLBACK

## Gasto

<u>cuenta</u>	<u>comentario</u>	<u>fecha</u>	<u>hora</u>	<u>monto</u>	<u>saldo</u>	<u>id</u>
7873698669	Electricidad	2020-02-02	20:00:01	8200	291800	TRCJASJDA9A
7873698669	Calefacción	2020-02-02	20:00:02	600	291200	TRC81KAQWAS
7873698669	Moviestar	2020-02-02	20:00:03	16200	275000	TRCK8J7JA8D
7873698669	Cajero	2020-02-08	16:05:02	100000	225000	TRCPM8A45AD

## Cuenta

<u>número</u>	<u>rut</u>	<u>tipo</u>	<u>saldo_clp</u>	<u>saldo_usd</u>
7873698669	32.000.273-K	Estacional	225000	344,94

## Cliente

<u>rut</u>	<u>nombre</u>	<u>fono</u>	<u>dirección</u>
32.000.273-K	Kelvin	+56976698463	Campo de Hielo Sur, Depto 273

## Divisa

<u>d1</u>	<u>d2</u>	<u>valor</u>
CLP	USD	0,001533
USD	CLP	652,275000

```
START TRANSACTION;  
INSERT INTO Gasto VALUES  
(7873698669, 'Noruega', '2020-02-12', '02:14:20', 400000, -175000, 'TRCLK9K24KS');  
ROLLBACK;
```

**ROLLBACK** deshace/borra los cambios desde el inicio de la transacción

# Transacciones: SAVEPOINT

## Gasto

<u>cuenta</u>	<u>comentario</u>	<u>fecha</u>	<u>hora</u>	<u>monto</u>	<u>saldo</u>	<u>id</u>
7873698669	Electricidad	2020-02-02	20:00:01	8200	291800	TRCJASJDA9A
7873698669	Calefacción	2020-02-02	20:00:02	600	291200	TRC81KAQWAS
7873698669	Moviestar	2020-02-02	20:00:03	16200	275000	TRCK8J7JA8D
7873698669	Cajero	2020-02-08	16:05:02	100000	225000	TRCPM8A45AD
7873698669	Noruega	2020-02-12	02:14:20	400000	-175000	TRCLK9K24KS

## Cuenta

<u>número</u>	<u>rut</u>	<u>tipo</u>	<u>saldo_clp</u>	<u>saldo_usd</u>
7873698669	32.000.273-K	Estacional	-175000	-268,29

## Divisa

<u>d1</u>	<u>d2</u>	<u>valor</u>
CLP	USD	0,001533
USD	CLP	652,275000

## Ciente

<u>rut</u>	<u>nombre</u>	<u>fono</u>	<u>dirección</u>
32.000.273-K	Kelvin	+56976698463	Campo de Hielo Sur, Depto 273

```
START TRANSACTION;
```

```
INSERT INTO Gasto VALUES
```

```
(7873698669, 'Noruega', '2020-02-12', '02:14:20', 400000, -175000, 'TRCLK9K24KS');
```

```
UPDATE Cuenta SET saldo_clp=-175000, saldo_usd=-268.25 WHERE número=7873698669;
```

```
SAVEPOINT CompraNoruega;
```

```
INSERT INTO Gasto VALUES
```

```
(7873698669, 'BOSE', '2020-02-12', '20:00:01', 200000, -375000, 'TRCASD8PNAK');
```

```
ROLLBACK TO SAVEPOINT CompraNoruega;
```

```
COMMIT;
```

**ROLLBACK** puede deshacer/borrar los cambios desde un punto específico con **SAVEPOINT**

# Una transacción con valores dinámicos

## Gasto

<u>cuenta</u>	<u>comentario</u>	<u>fecha</u>	<u>hora</u>	<u>monto</u>	<u>saldo</u>	<u>id</u>
7873698669	Electricidad	2020-02-02	20:00:01	8200	291800	TRCJASJDA9A
7873698669	Calefacción	2020-02-02	20:00:02	600	291200	TRC81KAQWAS
7873698669	Moviestar	2020-02-02	20:00:03	16200	275000	TRCK8J7JA8D
7873698669	Cajero	2020-02-08	16:05:02	100000	225000	TRCPM8A45AD

## Cuenta

<u>número</u>	<u>rut</u>	<u>tipo</u>	<u>saldo_clp</u>	<u>saldo_usd</u>
7873698669	32.000.273-K	Estacional	225000	344,94

## Cliente

<u>rut</u>	<u>nombre</u>	<u>fono</u>	<u>dirección</u>
32.000.273-K	Kelvin	+56976698463	Campo de Hielo Sur, Depto 273

## Divisa

<u>d1</u>	<u>d2</u>	<u>valor</u>
CLP	USD	0,001533
USD	CLP	652,275000

```
START TRANSACTION;  
INSERT INTO Gasto VALUES  
  (7873698669, 'Noruega', '2020-02-12', '02:14:20', 400000, -175000, 'TRCLK9K24KS');  
UPDATE Cuenta SET saldo_clp=(saldo_clp-400000) WHERE número=7873698669;  
UPDATE Cuenta SET saldo_usd=(A.saldo_clp/valor)  
  FROM ( SELECT valor FROM Divisa WHERE d1='USD' AND d2='CLP') T,  
  ( SELECT saldo_clp FROM Cuenta WHERE número=7873698669 ) A  
  WHERE número=7873698669;  
COMMIT;
```

¿Valor final de `saldo_usd` en `Cuenta`?

# Una transacción con valores dinámicos

Gasto							
cuenta	comentario	fecha	hora	monto	saldo	id	
7873698669	Electricidad	2020-02-02	20:00:01	8200	291800	TRCJASJDA9A	
7873698669	Calefacción	2020-02-02	20:00:02	600	291200	TRC81KAQWAS	
7873698669	Moviestar	2020-02-02	20:00:03	16200	275000	TRCK8J7JA8D	
7873698669	Cajero	2020-02-08	16:05:02	100000	225000	TRCPM8A45AD	
7873698669	Noruega	2020-02-12	02:14:20	400000	-175000	TRCLK9K24KS	

Cuenta				
número	rut	tipo	saldo_clp	saldo_usd
7873698669	32.000.273-K	Estacional	-175000	-268,28

Divisa		
d1	d2	valor
CLP	USD	0,001533
USD	CLP	652,275000

Cliente			
rut	nombre	fono	dirección
32.000.273-K	Kelvin	+56976698463	Campo de Hielo Sur, Depto 273

```
START TRANSACTION;
INSERT INTO Gasto VALUES
(7873698669, 'Noruega', '2020-02-12', '02:14:20', 400000, -175000, 'TRCLK9K24KS');
UPDATE Cuenta SET saldo_clp=(saldo_clp-400000) WHERE número=7873698669;
UPDATE Cuenta SET saldo_usd=(A.saldo_clp/valor)
FROM ( SELECT valor FROM Divisa WHERE d1='USD' AND d2='CLP') T,
( SELECT saldo_clp FROM Cuenta WHERE número=7873698669 ) A
WHERE número=7873698669;
COMMIT;
```

¿Valor final de `saldo_usd` en Cuenta? **-268,28** (Se lee el valor actual de la misma transacción)

# Una transacción con CHECK

## Gasto

<u>cuenta</u>	<u>comentario</u>	<u>fecha</u>	<u>hora</u>	<u>monto</u>	<u>saldo</u>	<u>id</u>
7873698669	Electricidad	2020-02-02	20:00:01	8200	291800	TRCJASJDA9A
7873698669	Calefacción	2020-02-02	20:00:02	600	291200	TRC81KAQWAS
7873698669	Moviestar	2020-02-02	20:00:03	16200	275000	TRCK8J7JA8D
7873698669	Cajero	2020-02-08	16:05:02	100000	225000	TRCPM8A45AD

## Cuenta

<u>número</u>	<u>rut</u>	<u>tipo</u>	<u>saldo_clp</u>	<u>saldo_usd</u>
7873698669	32.000.273-K	Estacional	225000	344,94

## Ciente

<u>rut</u>	<u>nombre</u>	<u>fono</u>	<u>dirección</u>
32.000.273-K	Kelvin	+56976698463	Campo de Hielo Sur, Depto 273

## Divisa

<u>d1</u>	<u>d2</u>	<u>valor</u>
CLP	USD	0,001533
USD	CLP	652,275000

```
START TRANSACTION;
```

```
INSERT INTO Gasto VALUES
```

```
(7873698669, 'Noruega', '2020-02-12', '02:14:20', 400000, -175000, 'TRCLK9K24KS');
```

```
UPDATE Cuenta SET saldo_clp=(saldo_clp-400000) WHERE número=7873698669;
```

```
UPDATE Cuenta SET saldo_usd=(A.saldo_clp/valor)
```

```
FROM ( SELECT valor FROM Divisa WHERE d1='USD' AND d2='CLP') T,
```

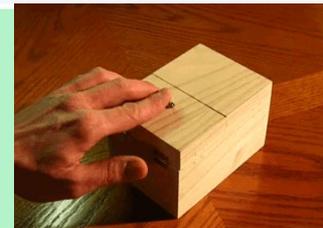
```
( SELECT saldo_clp FROM Cuenta WHERE número=7873698669 ) A
```

```
WHERE número=7873698669;
```

```
COMMIT;
```

```
CREATE TABLE Cuenta (...
```

```
CHECK ( saldo_clp = saldo_usd * (SELECT valor FROM Divisa WHERE d1='USD' AND d2='CLP') ) )
```



¿Funciona?

¡No!

# Transacciones / Restricciones: IMMEDIATE

Cuenta				
número	rut	tipo	saldo_clp	saldo_usd
7873698669	32.000.273-K	Estacional	225000	344,94

Banco de Chile

```
CREATE TABLE Cuenta (  
  ...,  
  CONSTRAINT Cuenta_PK  
    PRIMARY KEY (número)  
)
```

```
START TRANSACTION;  
  INSERT INTO Cuenta VALUES  
    (7873698669, '32.000.273-K', 'Estacional',  
     -175000, -268.29);  
  DELETE FROM Cuenta  
    WHERE número=7873698669 AND saldo_clp=225000;  
COMMIT;
```

Por defecto, se aplica la restricción inmediatamente después de cada sentencia

# Transacciones / Restricciones: DEFERRABLE

Cuenta				
<u>número</u>	rut	tipo	saldo_clp	saldo_usd
7873698669	32.000.273-K	Estacional	225000	344,94

Cuenta				
<u>número</u>	rut	tipo	saldo_clp	saldo_usd
7873698669	32.000.273-K	Estacional	-175000	-268,29

```
CREATE TABLE Cuenta (  
  ...,  
  CONSTRAINT Cuenta_PK  
    PRIMARY KEY (número)  
    DEFERRABLE  
)
```

```
START TRANSACTION;  
SET CONSTRAINT Cuenta_PK DEFERRED;  
INSERT INTO Cuenta  
  (7873698669, '32.000.273-K', 'Estacional',  
   -175000, -268.29);  
DELETE FROM Cuenta  
  WHERE número=7873698669 AND saldo_clp=225000;  
COMMIT;
```

**DEFERRABLE** define una restricción que se puede diferir hasta un **COMMIT**

**DEFERRED** difiere la restricción hasta el **COMMIT** en la transacción actual

# Transacciones / Restricciones: DEFERRABLE

Cuenta				
<u>número</u>	rut	tipo	saldo_clp	saldo_usd
7873698669	32.000.273-K	Estacional	225000	344,94

Cuenta				
<u>número</u>	rut	tipo	saldo_clp	saldo_usd
7873698669	32.000.273-K	Estacional	-175000	-268,29

```
CREATE TABLE Cuenta (  
    ...,  
    CONSTRAINT Cuenta_PK  
        PRIMARY KEY (número)  
        DEFERRABLE INITIALLY DEFERRED  
)
```

```
START TRANSACTION;  
    INSERT INTO Cuenta  
        (7873698669, '32.000.273-K', 'Estacional',  
         -175000, -268.29);  
    DELETE FROM Cuenta  
        WHERE número=7873698669 AND saldo_clp=225000;  
COMMIT;
```

**DEFERRABLE INITIALLY DEFERRED**

define una restricción que sea diferido por defecto hasta un **COMMIT**



# WARNING

Restricciones diferibles son estándares

Pero Postgres no permite diferir restricciones  
usando ni CHECK ni NOT NULL

Atomicidad, Consistencia, Aislamiento, Durabilidad  
(*Atomicity, Consistency, Isolation, Durability*)

# LAS GARANTÍAS DE ACID

No hay un solo usuario ...



... hay que tener cuidado con la concurrencia

# Una cuenta con varios usuarios



**Banco de Chilly**

N° Cuenta : 7873698669  
Saldo (CLP) : 225000  
Límite de crédito : 200000  
Disponible : 425000



```
CREATE TABLE Cuenta ( ..., CHECK ( saldo_clp > -200000 ) )
```

COMPRA(Islas de Caimán,300000)

```
INSERT INTO Gasto ...
```

```
UPDATE Cuenta ...
```

COMPRA(Noruega,400000)

```
INSERT INTO Gasto ...
```

```
UPDATE Cuenta ...
```

*¿Qué será el resultado final? ...*

# Caos



# Esta vez con transacciones ...



**Banco de Chilly**

N° Cuenta : 7873698669  
Saldo (CLP) : 225000  
Límite de crédito : 200000  
Disponible : 425000



```
CREATE TABLE Cuenta ( ..., CHECK ( saldo_clp > -200000 ) )
```

COMPRA(Islas de Caimán,300000)

COMPRA(Noruega,400000)

START TRANSACTION

INSERT INTO Gasto ...

UPDATE Cuenta ...

COMMIT;

START TRANSACTION

INSERT INTO Gasto ...

UPDATE Cuenta ...

COMMIT;

*¿Qué será el resultado final?*

Se rechazará una transacción.

# Garantías de ACID

- **Atomicidad:**
  - La ejecución de cada transacción es atómica:
    - Se realizan todas las acciones o no se realiza ninguna
- **Consistencia:**
  - Cada transacción debe preservar la integridad
    - La base de datos satisfacen todas las restricciones después de una transacción
- **Aislamiento (*Isolation*):**
  - Una transacción no puede afectar otra
- **Durabilidad:**
  - Una vez que haya un **COMMIT**, la base de datos debe persistir los cambios

# ACID: Un ejemplo más limpio

```
CREATE TABLE Balance (  
  cuenta BIGINT PRIMARY KEY,  
  total_gasto BIGINT,  
  total_ingreso BIGINT,  
  saldo BIGINT,  
  CHECK (total_ingreso - total_gasto = saldo)  
)
```

Usaré restricciones con CHECK porque dan ejemplos más claros pero es importante tener en cuenta que Postgres no soporte CHECKs diferibles.

# ACID: Atomicidad

```
CREATE TABLE Balance (  
  cuenta BIGINT PRIMARY KEY,  
  total_gasto BIGINT,  
  total_ingreso BIGINT,  
  saldo BIGINT,  
  CHECK (total_ingreso - total_gasto = saldo)  
)
```

```
START TRANSACTION  
  UPDATE Balance SET saldo=saldo-10 WHERE Cuenta=7873698669 ;  
  UPDATE Balance SET total_gasto=total_gasto+10 WHERE Cuenta=7873698669 ;  
COMMIT;
```

## Atomicidad

No se puede actualizar el saldo sin actualizar el gasto directamente después.  
(Si alguna actualización falla, ambas fallan.)

# ACID: Consistencia

```
CREATE TABLE Balance (  
  cuenta BIGINT PRIMARY KEY,  
  total_gasto BIGINT,  
  total_ingreso BIGINT,  
  saldo BIGINT,  
  CHECK (total_ingreso - total_gasto = saldo)  
)
```

```
START TRANSACTION  
  UPDATE Balance SET saldo=saldo-100 WHERE Cuenta=7873698669 ;  
  UPDATE Balance SET total_gasto=total_gasto+10 WHERE Cuenta=7873698669 ;  
COMMIT;
```



## Consistencia

Si el resultado de la transacción no satisface todas las restricciones, fallará.

# ACID: Aislamiento (*Isolation*)

```
CREATE TABLE Balance (  
  cuenta BIGINT PRIMARY KEY,  
  total_gasto BIGINT,  
  total_ingreso BIGINT,  
  saldo BIGINT,  
  CHECK (total_ingreso - total_gasto = saldo)  
)
```

```
START TRANSACTION T1  
  UPDATE Balance  
    SET saldo=saldo-10  
    WHERE Cuenta=7873698669 ; (1)  
  UPDATE Balance  
    SET total_gasto=total_gasto+100  
    WHERE Cuenta=7873698669 ; (3) ❌  
COMMIT;  
(4) ROLLBACK;
```

```
START TRANSACTION T2  
  UPDATE Balance  
    SET saldo=saldo+100  
    WHERE Cuenta=7873698669 ; (2)  
  UPDATE Balance  
    SET total_ingreso=total_ingreso+100  
    WHERE Cuenta=7873698669 ; (5)  
COMMIT; (6)
```

## Aislamiento

Una transacción no puede interferir con otra transacción.

En (4), hay que tener cuidado con el **ROLLBACK**: no se puede restaurar el valor de saldo antes del paso (1) porque el valor ya fue cambiado por (2).

# ACID: Durabilidad

```
CREATE TABLE Balance (  
  cuenta BIGINT PRIMARY KEY,  
  total_gasto BIGINT,  
  total_ingreso BIGINT,  
  saldo BIGINT,  
  CHECK (total_ingreso - total_gasto = saldo)  
)
```

```
START TRANSACTION  
  UPDATE Balance SET saldo=saldo-10 WHERE Cuenta=7873698669 ;  
  UPDATE Balance SET total_gasto=total_gasto+10 WHERE Cuenta=7873698669 ;  
COMMIT;
```



## Durabilidad

Una vez que haya un **COMMIT** exitoso, se persisten los cambios.

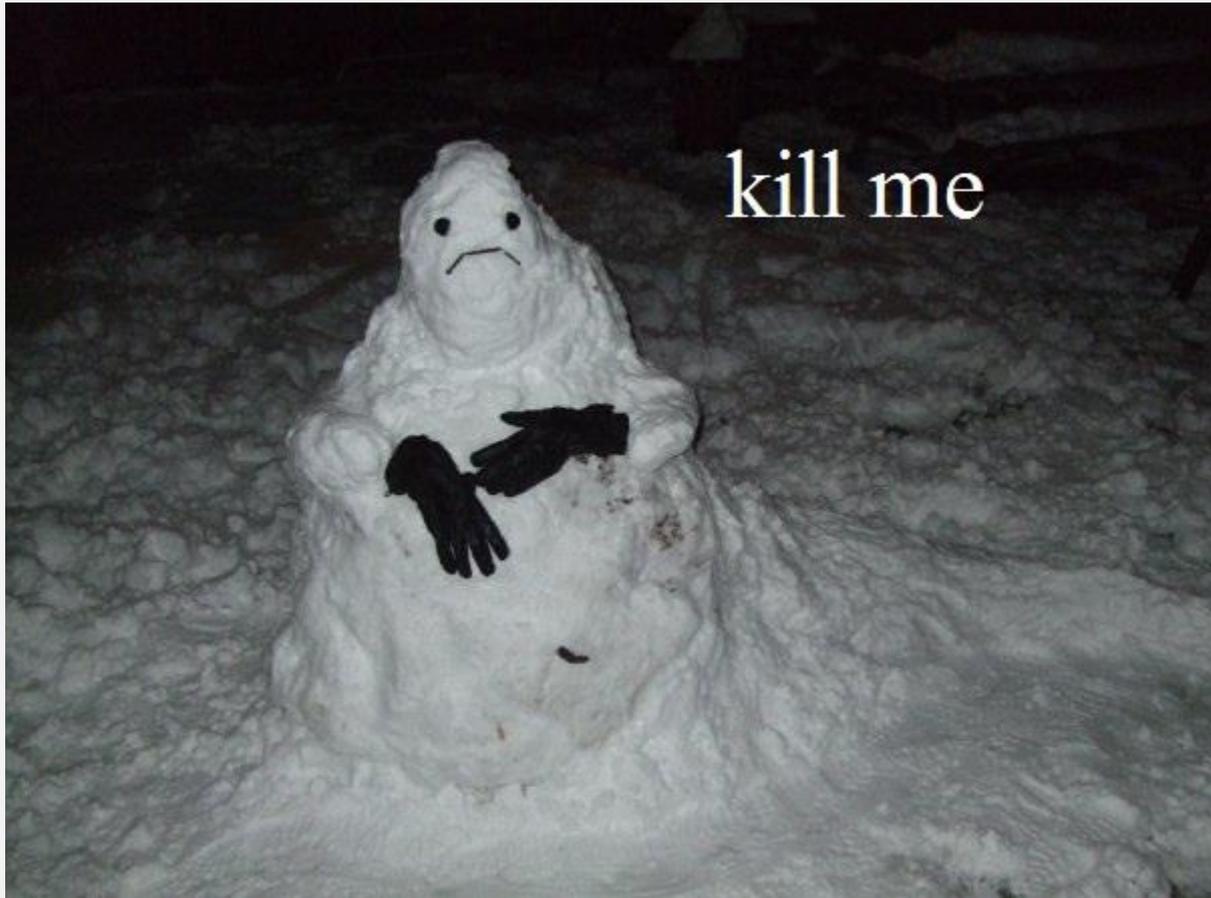
(Normalmente la persistencia aquí significa en el disco duro. Sin persistencia, en el caso de que la máquina falla y toda la evidencia de los cambios está en memoria principal, el sistema de base de datos **olvidará los cambios silenciosamente**.)

Entonces con las garantías de ACID ...



... todo está tranquilo.

... pero si uno tiene que *implementar* ACID



... es más difícil ...

# ¿Cuándo **no** tenemos ACID?

- **Atomicidad:**
  - × Una transacción se ejecuta solo a medias pero afecta el estado de la base de datos
- **Consistencia:**
  - × Al ejecutar la transacción, la base de datos no satisface las restricciones de integridad
- **Aislamiento (*Isolation*):**
  - × El resultado final de dos transacciones no es equivalente a correr cada transacción en serie
- **Durabilidad:**
  - × La base de datos se actualiza momentáneamente y luego vuelve al estado anterior.

# Modelando una transacción

**LEER**( $X$ ): leer un objeto  $X$  de la base de datos a memoria principal

**ESCRIBIR**( $X$ ): escribir un objeto de memoria principal a la base de datos

Un objeto  $X$  puede ser un valor, una fila, una tabla ...

*Ejemplo: Una transferencia bancaria ...*

**LEER**( $A$ )

$A \leftarrow A - 100$

**ESCRIBIR**( $A$ ) 

**LEER**( $B$ )

$B \leftarrow B + 100$

**ESCRIBIR**( $B$ ) 

T

← Dejamos el **COMMIT** implícito al fin de la transacción.

# ¿Cuándo **no** tenemos ACID?

- **Atomicidad:**
  - × Una transacción se ejecuta solo a medias pero afecta el estado de la base de datos
- **Consistencia:**
  - × Al ejecutar la transacción, la base de datos no satisface las restricciones de integridad
- **Durabilidad:**
  - × La base de datos se actualiza momentáneamente y luego vuelve al estado anterior.

# Modelando una transacción

- **Atomicidad:**
  - × Una transacción se ejecuta solo a medias pero afecta el estado de la base de datos

*Ejemplo: Una transferencia bancaria ...*

```
LEER(A) T  
A ← A - 100  
ESCRIBIR(A)   
LEER(B)  
B ← B + 100  
ESCRIBIR(B) 
```

No se pueden realizar dos escrituras *exactamente* al mismo tiempo

IMPLEMENTANDO ACID:  
REGISTROS (*LOGGING*)

# Mantener un registro de la transacción

*Ejemplo: Una transferencia bancaria ...*

LEER( $A$ )

$A \leftarrow A - 100$

ESCRIBIR( $A$ ) 

LEER( $B$ )

$B \leftarrow B + 100$

ESCRIBIR( $B$ ) 

T

```
T begin
```

```
T leer A 400
```

```
T escribir A 300
```

```
T leer B 200
```

```
T escribir B 300
```

```
T commit
```



```
./registro.log
```

# Si hay un problema, revertir el registro

La información en el registro debe bastar para revertir el estado de la base de datos sin ambigüedad

*Ejemplo: Una transferencia bancaria ...*

LEER( $A$ )

$A \leftarrow A - 100$

ESCRIBIR( $A$ ) 

LEER( $B$ )

$B \leftarrow B + 100$

ESCRIBIR( $B$ ) 

T

T begin

T leer A 400

T escribir A 300

T leer B 200

T escribir B 300

T rollback



./registro.log

# Registros ayudan con ...

- **Atomicidad:** 
  - × Una transacción se ejecuta solo a medias pero afecta el estado de la base de datos
- **Consistencia:** 
  - × Al ejecutar la transacción, la base de datos no satisface las restricciones de integridad
- **Aislamiento (*Isolation*):** 
  - × El resultado final de dos transacciones no es equivalente a correr cada transacción en serie
- **Durabilidad:** 
  - × La base de datos se actualiza momentáneamente y luego vuelve al estado anterior.

*¿Cuáles problemas podemos evitar con alguna forma de registro? ...*

# Concurrencia/Aislamiento: un problema abierto

```
CREATE TABLE Balance (  
  cuenta INTEGER PRIMARY KEY,  
  total_gasto BIGINT,  
  total_ingreso BIGINT,  
  saldo BIGINT,  
  CHECK (total_ingreso - total_gasto = saldo)  
)
```

```
START TRANSACTION T1  
  UPDATE Balance  
    SET saldo=saldo-10 (1)  
    WHERE Cuenta=7873698669 ;  
  UPDATE Balance  
    SET total_gasto=total_gasto+100  
    WHERE Cuenta=7873698669 ; (3) ❌  
COMMIT; (4) ROLLBACK;
```

```
START TRANSACTION T2  
  UPDATE Balance  
    SET saldo=saldo+100 (2)  
    WHERE Cuenta=7873698669 ;  
  UPDATE Balance  
    SET total_ingreso=total_ingreso+100  
    WHERE Cuenta=7873698669 ; (5)  
COMMIT; (6)
```

## Aislamiento

Una transacción no puede interferir con otra transacción.

En (4), hay que tener cuidado con el ROLLBACK: no se puede restaurar el valor de saldo antes del paso (1) porque el valor ya fue cambiado por (2).

IMPLEMENTANDO ACID:  
SECUENCIABILIDAD (*SERIALIZABILITY*)

# Registros no ayudan con ...

- Atomicidad:
  - × Una transacción se ejecuta solo a medias pero afecta el estado de la base de datos
- Consistencia:
  - × Al ejecutar la transacción, la base de datos no satisface las restricciones de integridad
- Aislamiento (*Isolation*):
  - × El resultado final de dos transacciones no es equivalente a correr cada transacción en serie
- Durabilidad:
  - × La base de datos se actualiza momentáneamente y luego vuelve al estado anterior.



... entonces que podemos hacer con respecto a aislamiento/concurrencia? ...

# La solución más simple: ejecución serial

*Ejemplo:* Dos transferencias bancarias entre cuentas  $A$  y  $B$   
(donde el valor  $A + B$  no debe cambiar,  $A$  inicia con 400,  $B$  con 200)

LEER( $A$ )

$A \leftarrow A - 100$

ESCRIBIR( $A$ ) 

LEER( $B$ )

$B \leftarrow B + 100$

ESCRIBIR( $B$ ) 

$T_1$

¿Cuánto es  $A + B$  después?

$$270 + 330 = 600$$

E  
J  
E  
C  
U  
C  
I  
Ó  
N

¿Hay un problema aquí?

¡Ejecución serial es lenta!

¿Se pueden ejecutar partes de  
 $T_1$  y  $T_2$  en paralelo? ...

LEER( $A$ )

$v \leftarrow A \times 0,1$

$A \leftarrow A - v$

ESCRIBIR( $A$ ) 

LEER( $B$ )

$B \leftarrow B + v$

ESCRIBIR( $B$ ) 

$T_2$

# ¿Ejecución paralela?

*Ejemplo:* Dos transferencias bancarias entre cuentas  $A$  y  $B$   
(donde el valor  $A + B$  no debe cambiar,  $A$  inicia con 400,  $B$  con 200)

LEER( $A$ )

$A \leftarrow A - 100$

ESCRIBIR( $A$ ) 

LEER( $B$ )

$B \leftarrow B + 100$

ESCRIBIR( $B$ ) 

$T_1$

LEER( $A$ )

$v \leftarrow A \times 0,1$

$A \leftarrow A - v$

ESCRIBIR( $A$ ) 

LEER( $B$ )

$B \leftarrow B + v$

ESCRIBIR( $B$ ) 

$T_2$

E  
J  
E  
C  
U  
C  
I  
Ó  
N

¿Cuánto es  $A + B$  después?

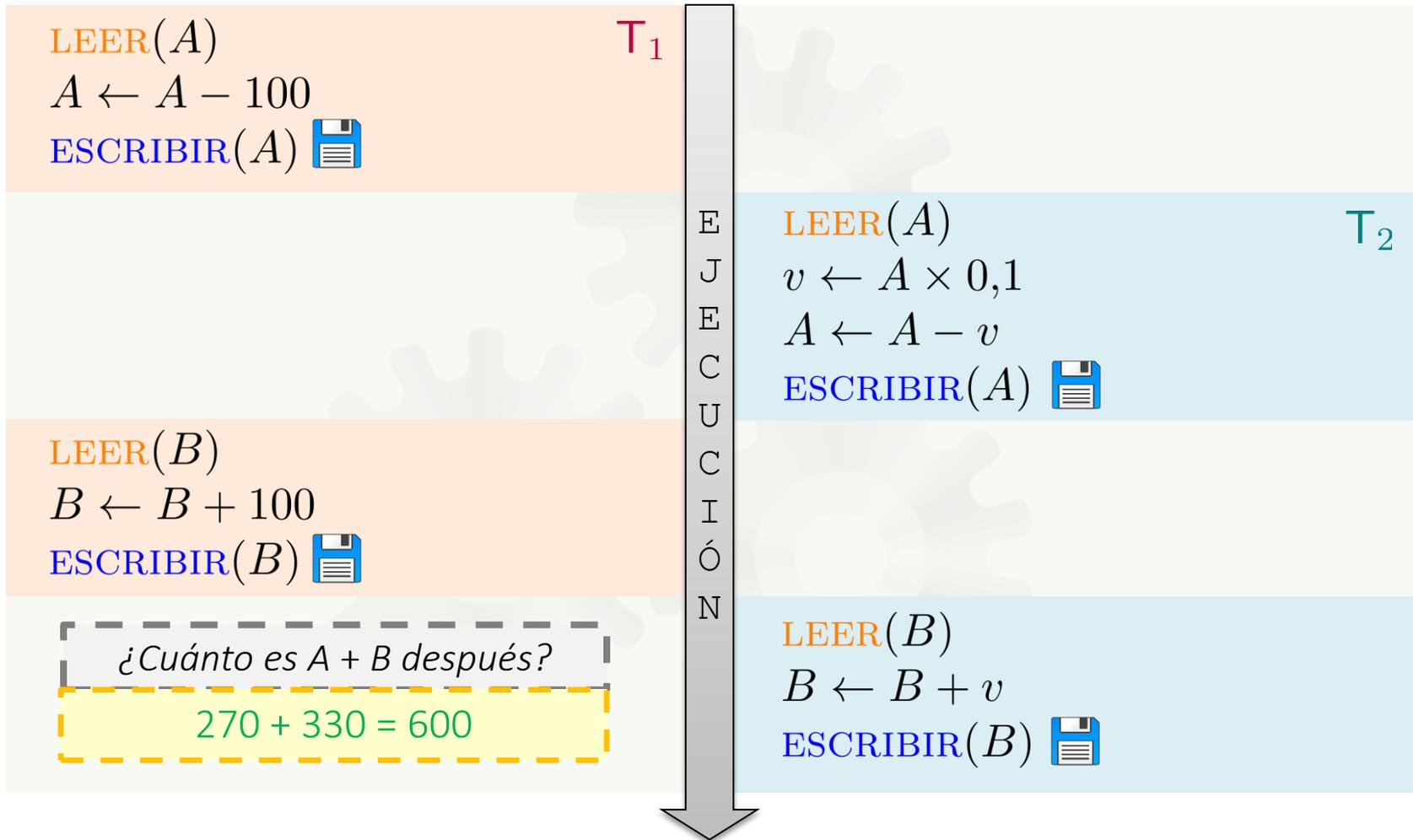
~\\_(\ツ)\\_/~

Por simplicidad asumiremos  
un orden de ejecución en el  
caso paralelo ...



# ¿Ejecución paralela?

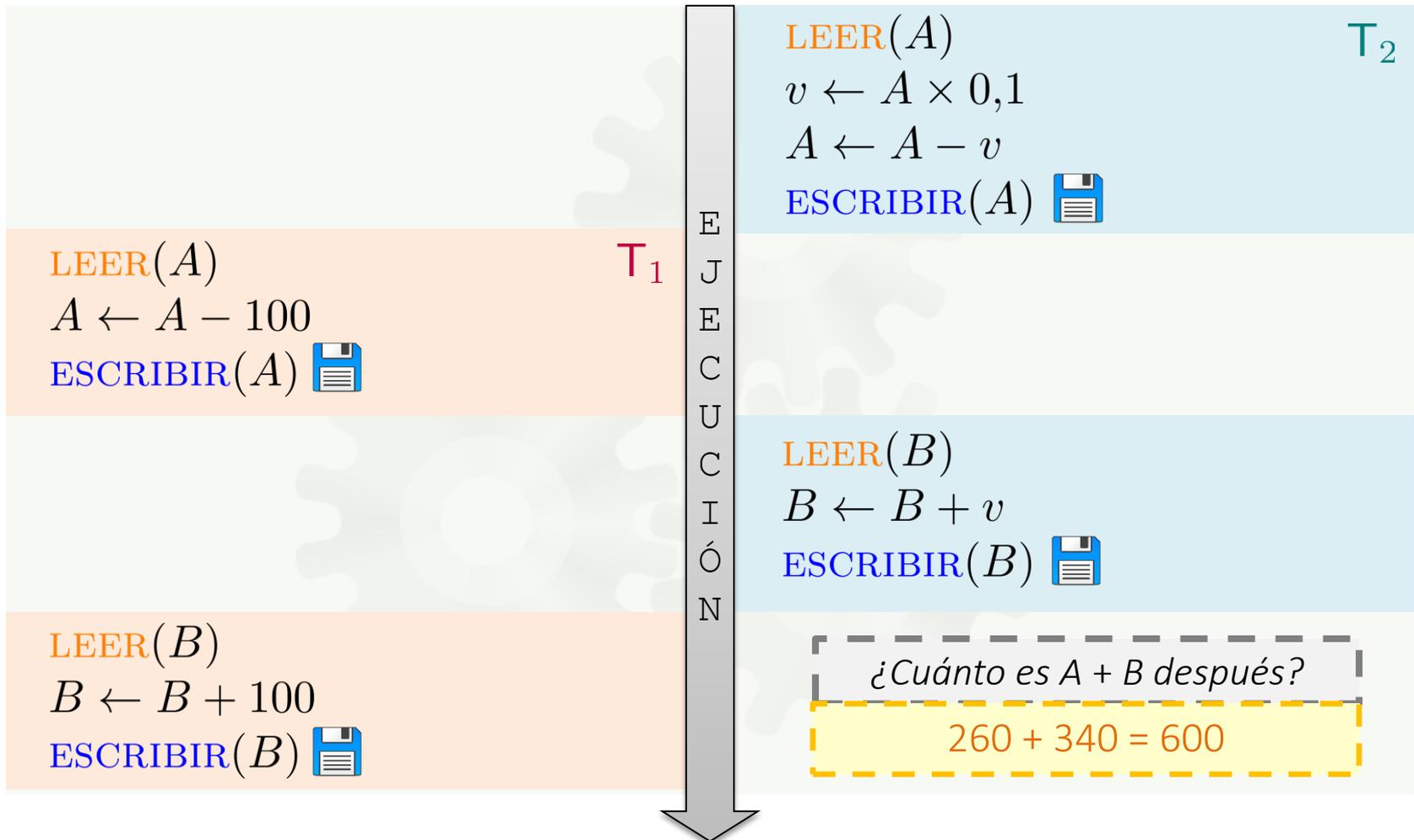
*Ejemplo:* Dos transferencias bancarias entre cuentas  $A$  y  $B$   
(donde el valor  $A + B$  no debe cambiar,  $A$  inicia con 400,  $B$  con 200)





# ¿Ejecución paralela?

*Ejemplo:* Dos transferencias bancarias entre cuentas  $A$  y  $B$   
(donde el valor  $A + B$  no debe cambiar,  $A$  inicia con 400,  $B$  con 200)





# ¡Cuidado cuando el orden importe!

Ejemplo: Dos transferencias bancarias entre cuentas  $A$  y  $B$   
(donde el valor  $A + B$  no debe cambiar,  $A$  inicia con 400,  $B$  con 200)

¿Cuánto es  $A + B$  después?

$$260 + 340 = 600$$

LEER( $A$ )

$A \leftarrow A - 100$

ESCRIBIR( $A$ ) 

LEER( $B$ )

$B \leftarrow B + 100$

ESCRIBIR( $B$ ) 

$T_1$

E  
J  
E  
C  
U  
C  
I  
Ó  
N

LEER( $A$ )

$v \leftarrow A \times 0,1$

$A \leftarrow A - v$

ESCRIBIR( $A$ ) 

LEER( $B$ )

$B \leftarrow B + v$

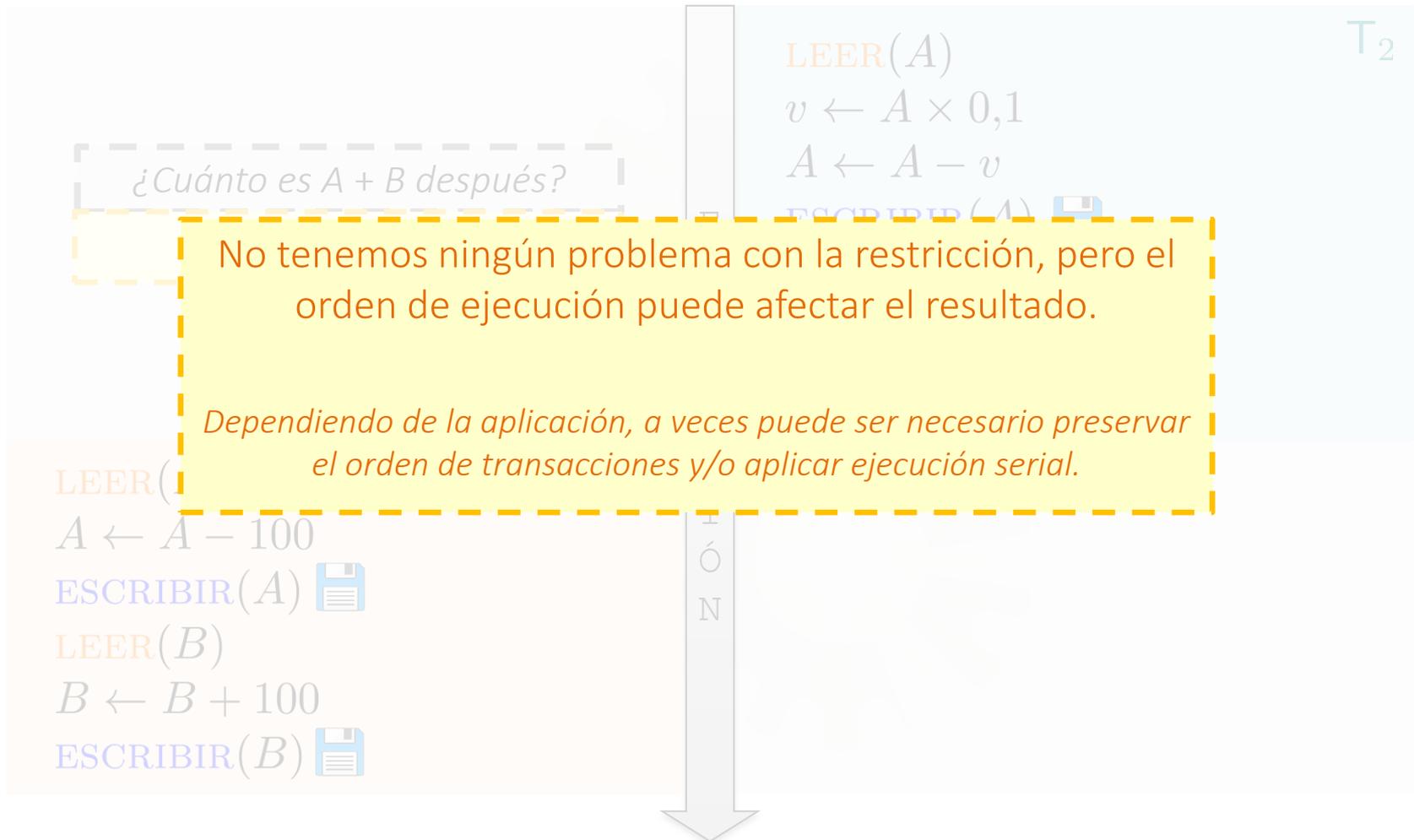
ESCRIBIR( $B$ ) 

$T_2$



# ¡Cuidado cuando el orden importe!

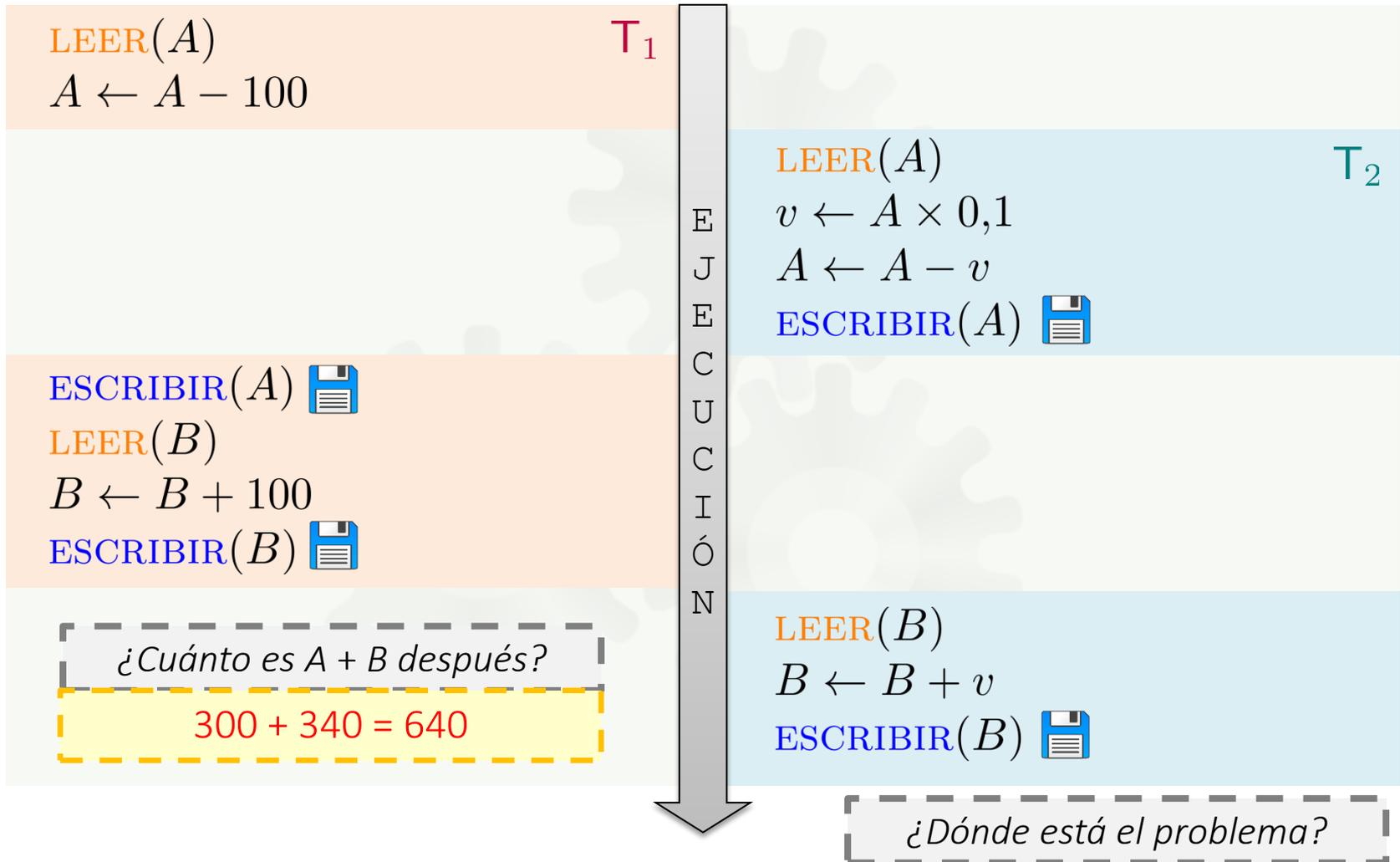
*Ejemplo:* Dos transferencias bancarias entre cuentas  $A$  y  $B$   
(donde el valor  $A + B$  no debe cambiar,  $A$  inicia con 400,  $B$  con 200)





# ¿Ejecución paralela?

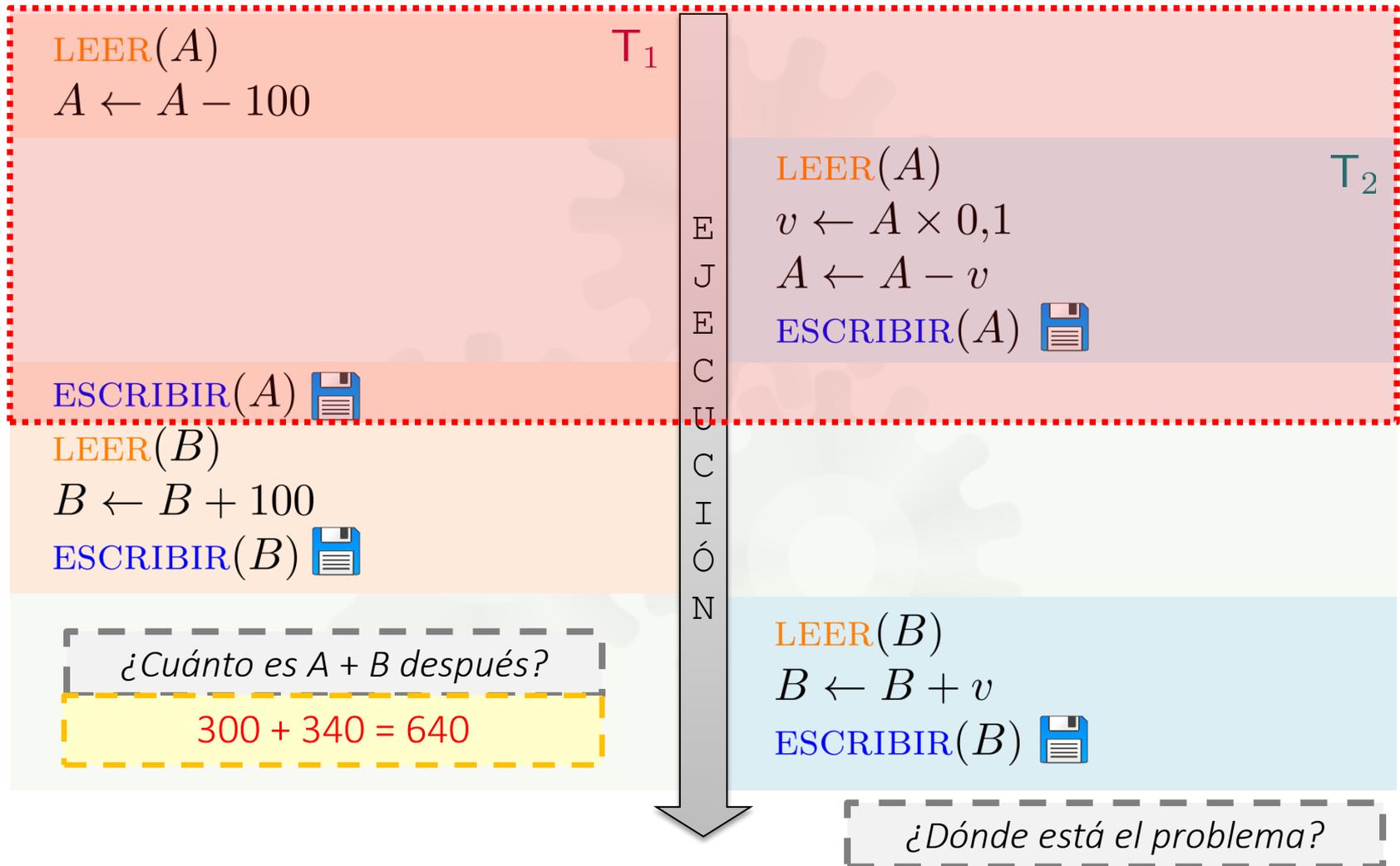
*Ejemplo:* Dos transferencias bancarias entre cuentas  $A$  y  $B$   
(donde el valor  $A + B$  no debe cambiar,  $A$  inicia con 400,  $B$  con 200)





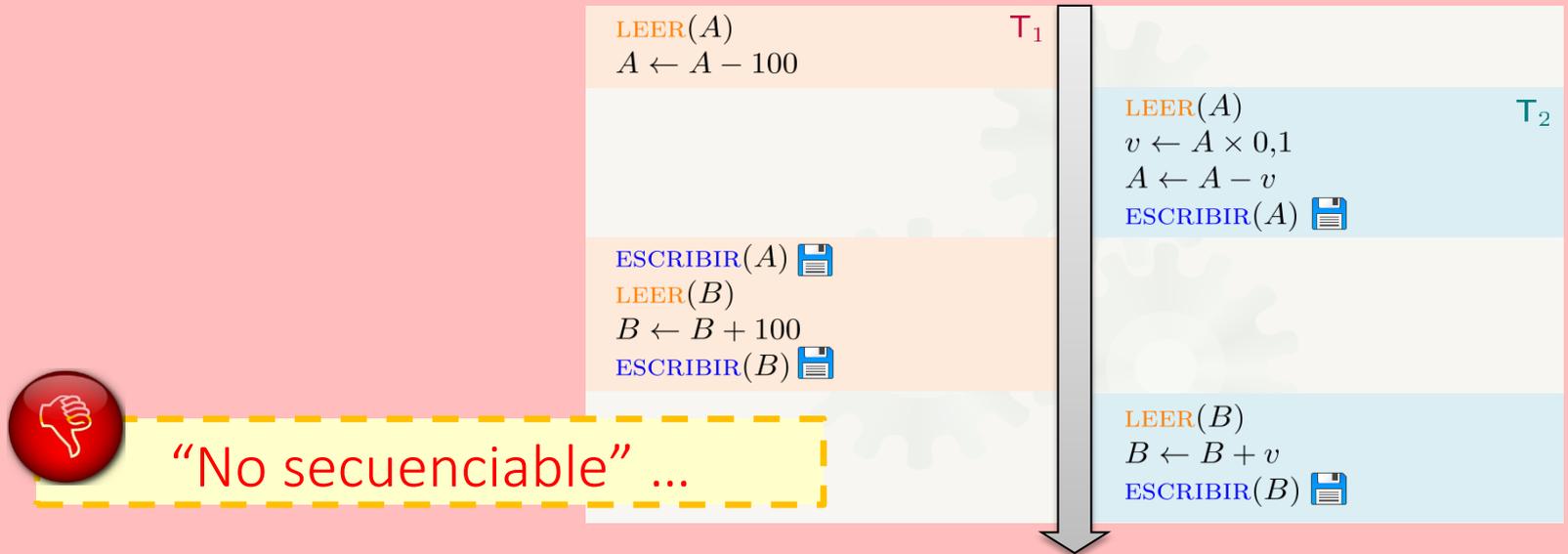
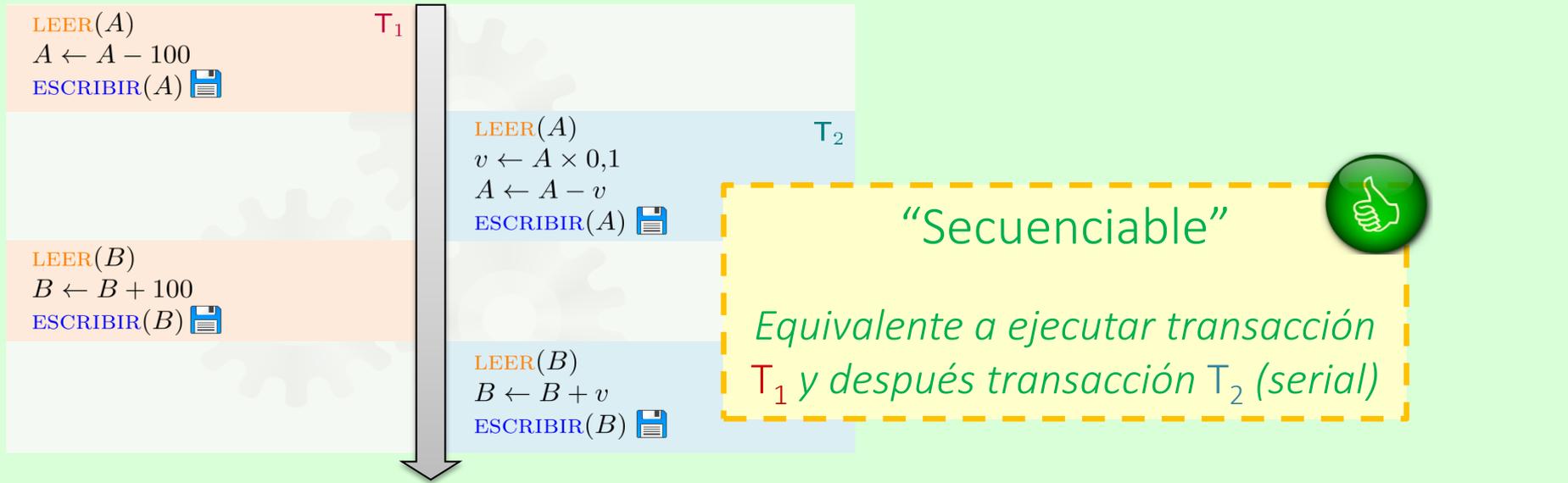
# ¿Ejecución paralela?

*Ejemplo:* Dos transferencias bancarias entre cuentas  $A$  y  $B$   
(donde el valor  $A + B$  no debe cambiar,  $A$  inicia con 400,  $B$  con 200)



# Planificaciones:

## Secuenciables vs. No Secuenciables



# Planificaciones:

## Secuenciables vs. No Secuenciables

LEER(A)  
 $A \leftarrow A - 100$   
ESCRIBIR(A)

$T_1$

Una **planificación** es una lista de acciones de un conjunto de transacciones en el orden de ejecución.

LEER(B)  
 $B \leftarrow B + 100$   
ESCRIBIR(B)

LEER(B)  
 $B \leftarrow B + v$

Equivalente a ejecutar transacción  $T_1$  y después transacción  $T_2$  (serial)

Una **planificación secuenciable** tendrá el mismo efecto que una planificación serial (en algún orden de las transacciones).

LEER(A)  
 $v \leftarrow A \times 0,1$   
 $A \leftarrow A - v$

$T_2$

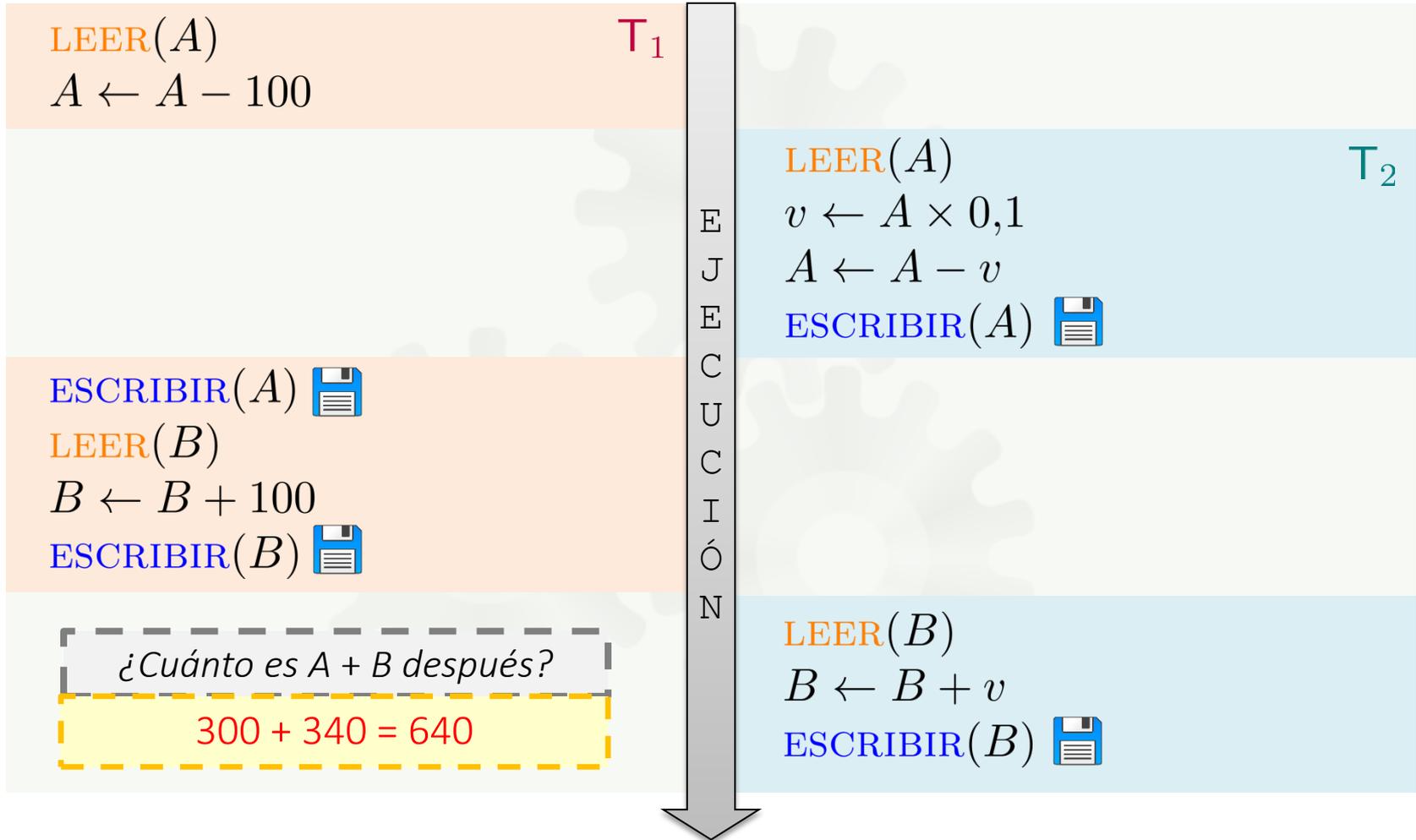
¿Cómo se puede identificar una planificación secuenciable?  
(sin ejecutarla y compararla con todas las posibles planificaciones seriales)? ...

Seguiremos con la pregunta inversa ... ¿cuándo **no se puede** garantizar que una planificación sea secuenciable?



# (1) Conflicto de Lectura–Escritura

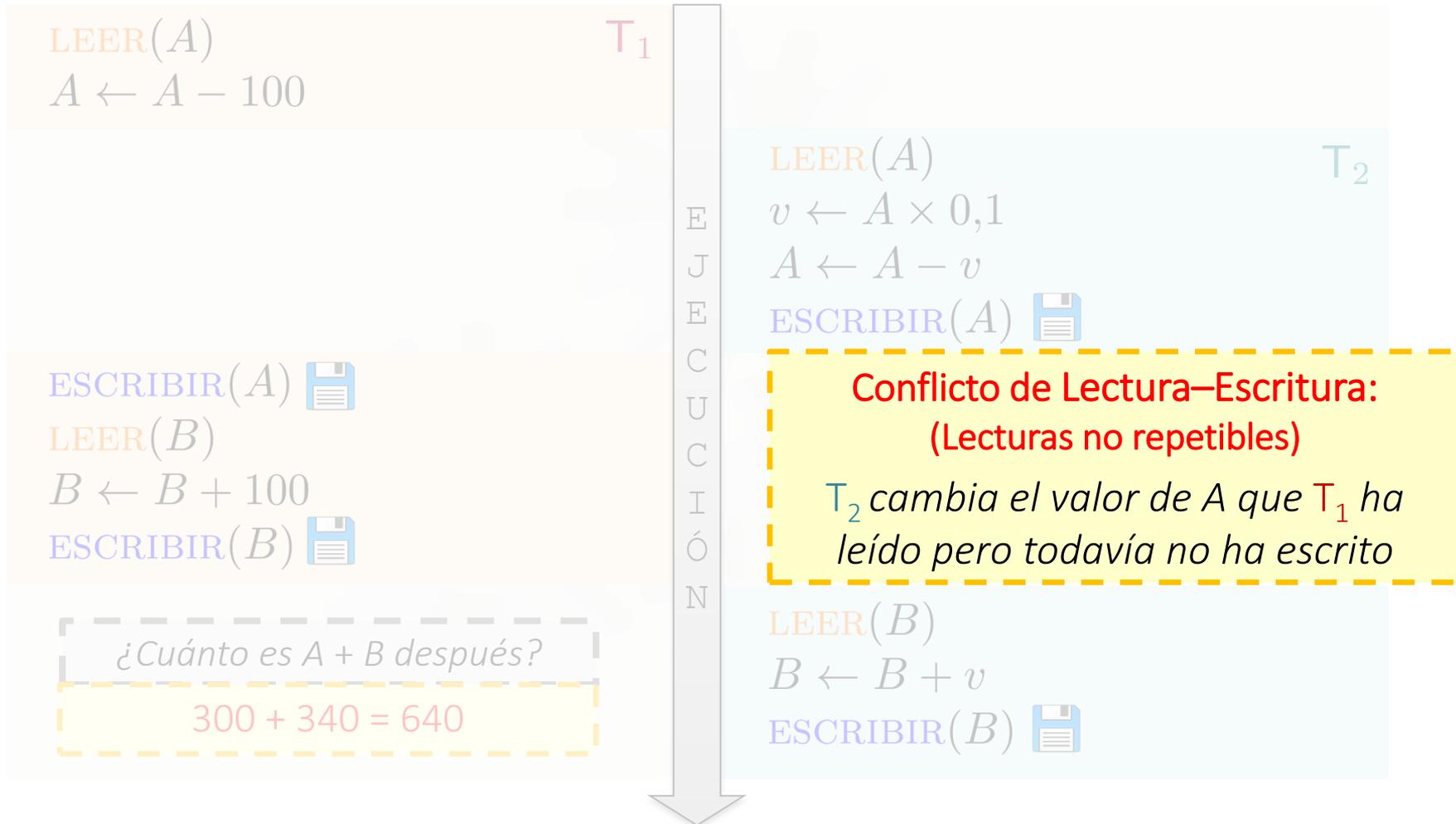
*Ejemplo 1:* Dos transferencias bancarias entre cuentas  $A$  y  $B$   
(donde el valor  $A + B$  no debe cambiar,  $A$  inicia con 400,  $B$  con 200)





# (1) Conflicto de Lectura–Escritura

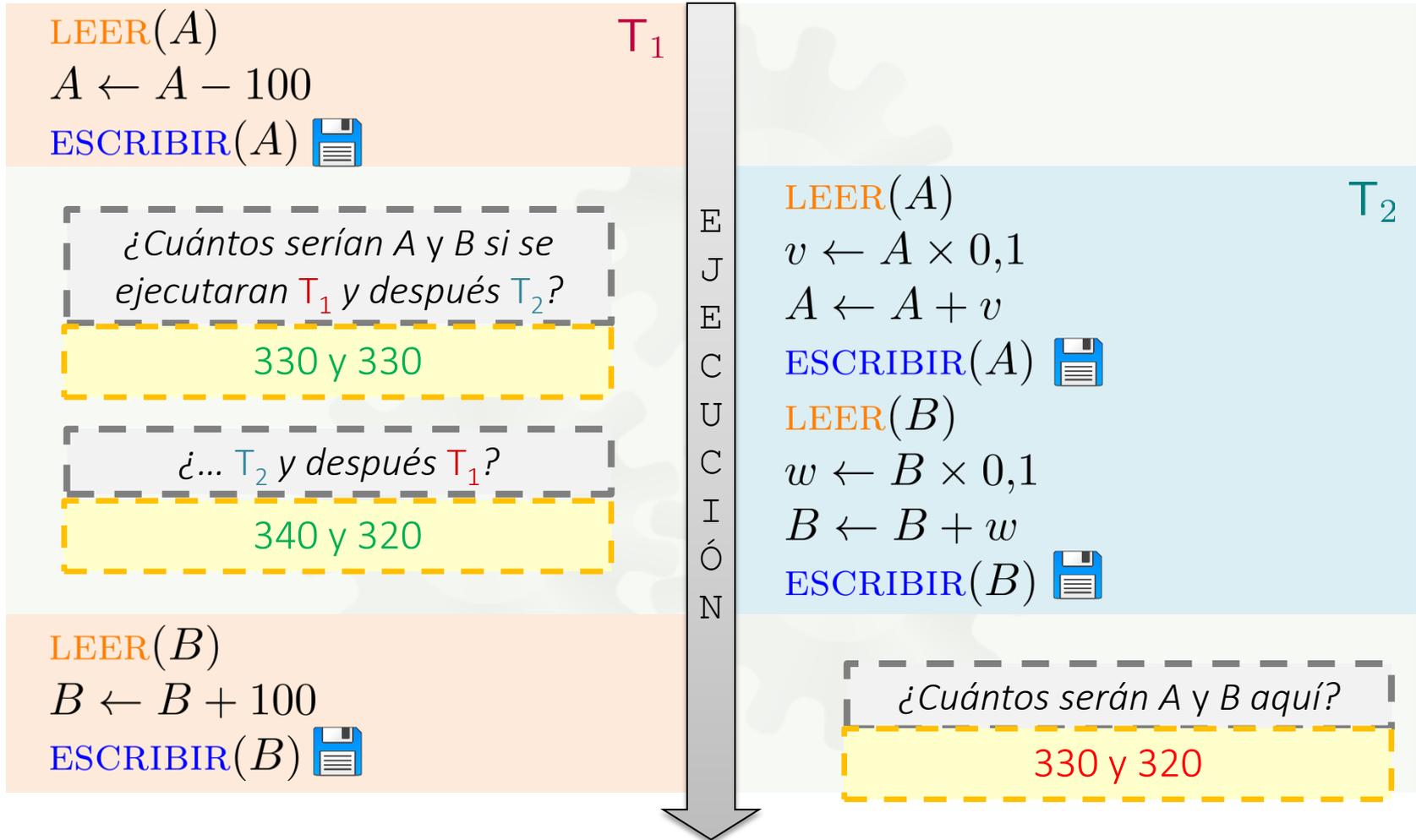
*Ejemplo 1:* Dos transferencias bancarias entre cuentas  $A$  y  $B$   
(donde el valor  $A + B$  no debe cambiar,  $A$  inicia con 400,  $B$  con 200)





## (2) Conflicto de Escritura–Lectura

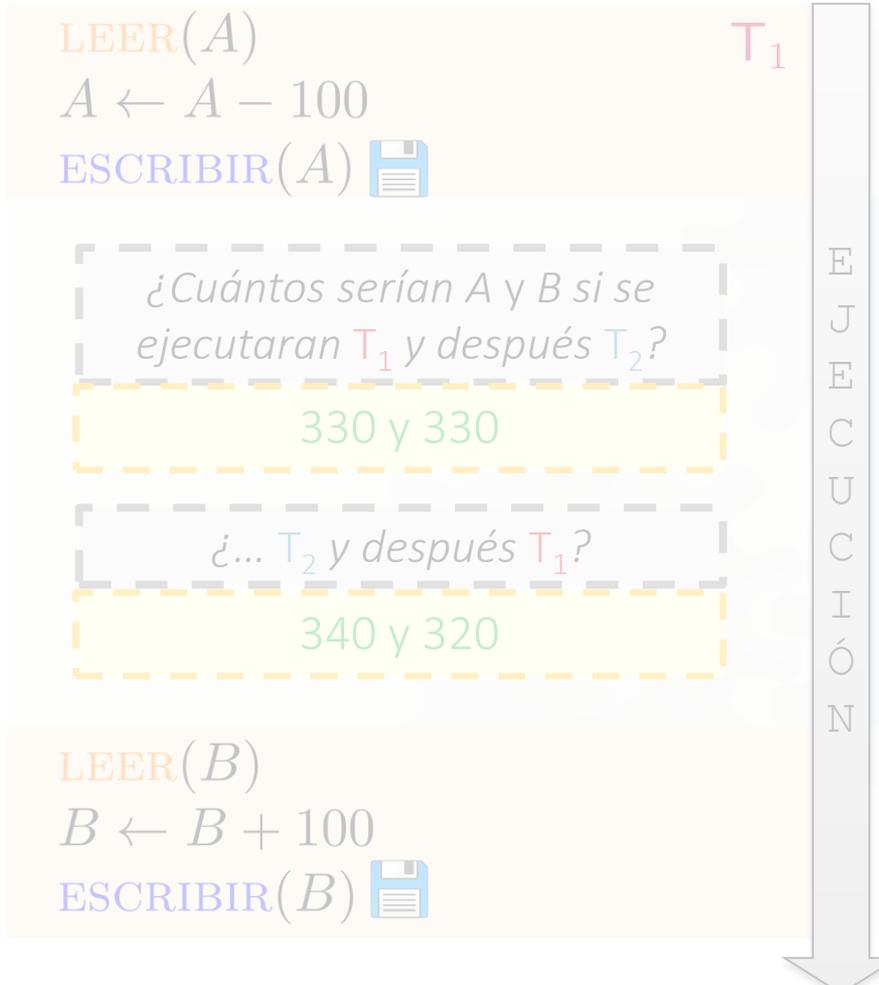
Ejemplo 2: Una transferencia y un pago de interés  
( $A$  inicia con 400,  $B$  con 200)



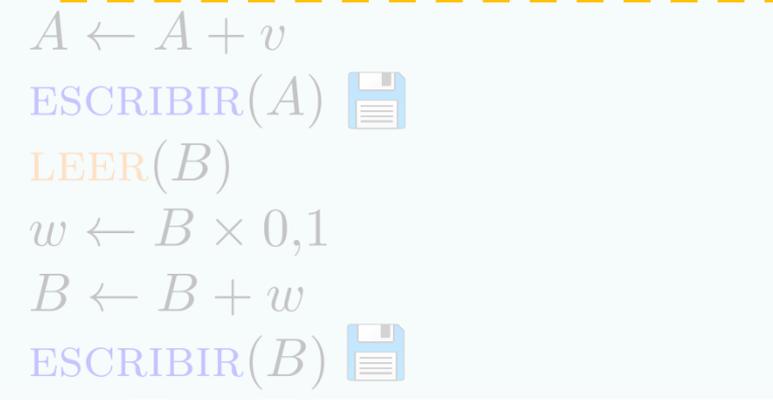


## (2) Conflicto de Escritura–Lectura

Ejemplo 2: Una transferencia y un pago de interés  
( $A$  inicia con 400,  $B$  con 200)



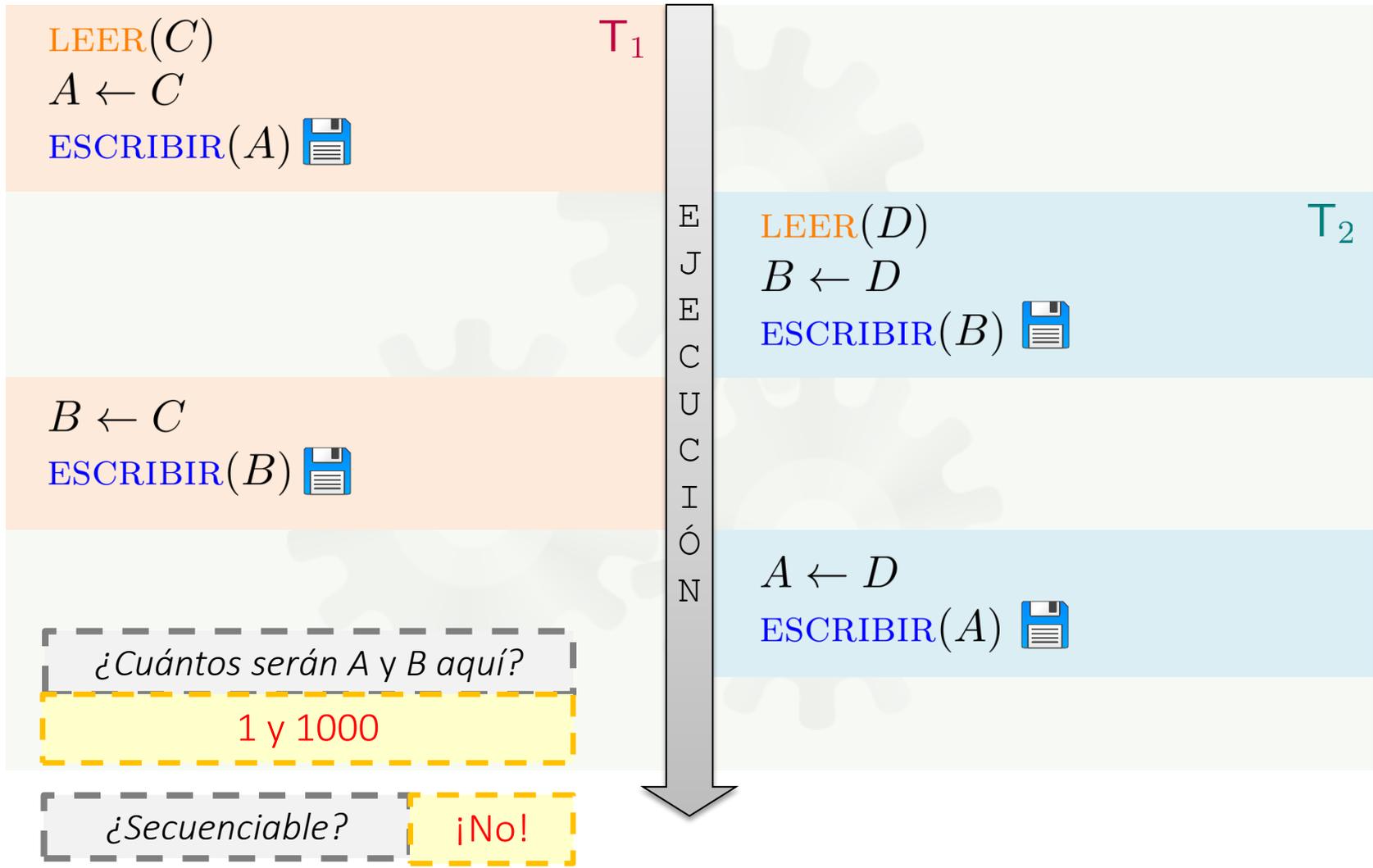
**Conflicto de Escritura–Lectura:**  
El orden de cambio de  $A$  es  $T_1/T_2$  mientras el orden de  $B$  es  $T_2/T_1$ , entonces el orden de ambos no es ni  $T_1/T_2$  ni  $T_2/T_1$





# (3) Conflicto de Escritura–Escritura

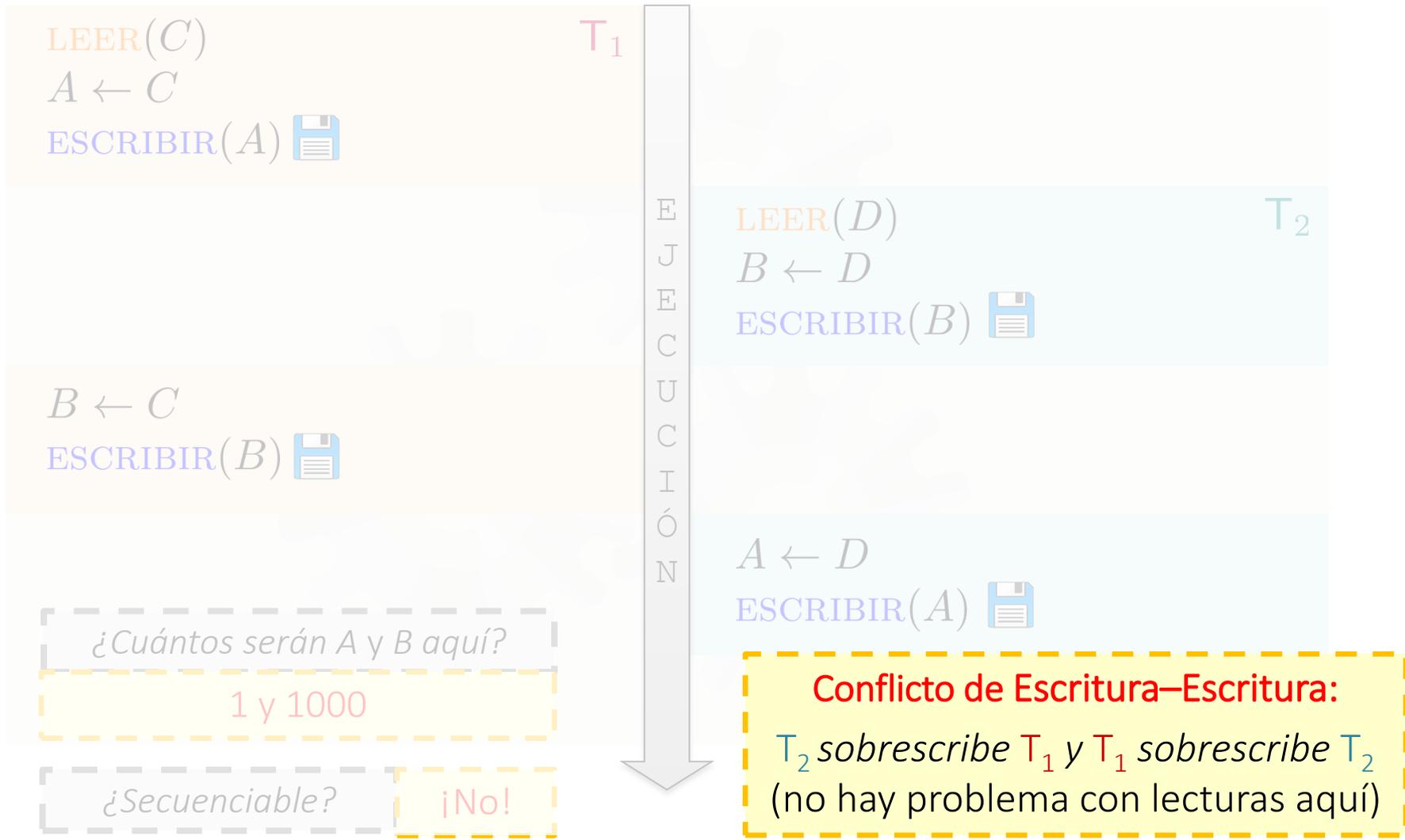
Ejemplo 3: Fijando los dos valores de las cuentas  $A$  y  $B$  desde  $C$  y  $D$   
( $A$  inicia con 400,  $B$  con 200,  $C$  con 1000,  $D$  con 1)

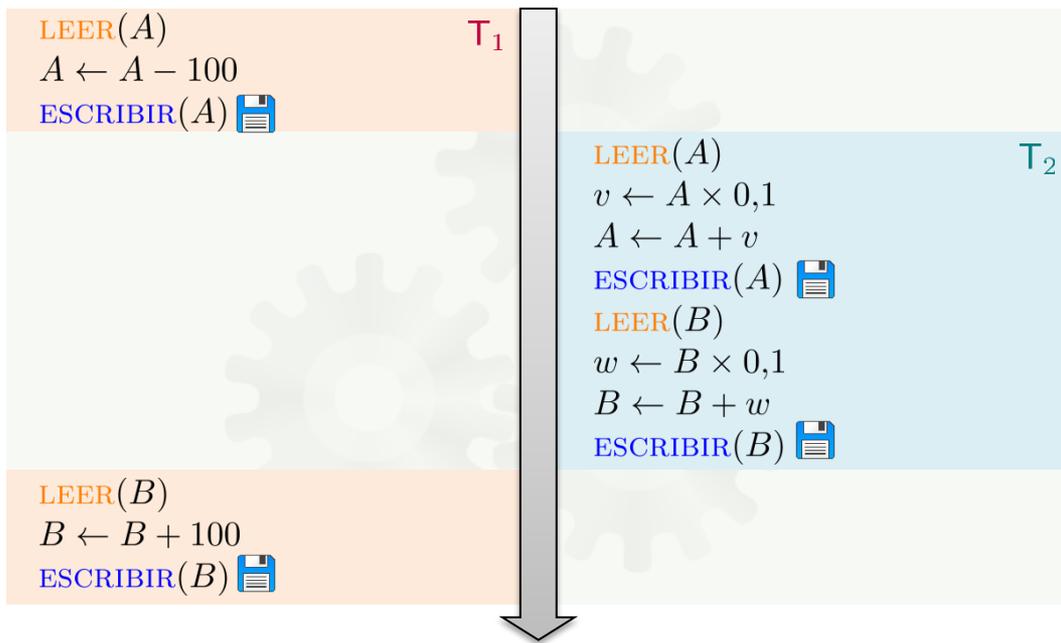




# (3) Conflicto de Escritura–Escritura

*Ejemplo 3: Fijando los dos valores de las cuentas A y B desde C y D*  
(A inicia con 400, B con 200, C con 1000, D con 1)





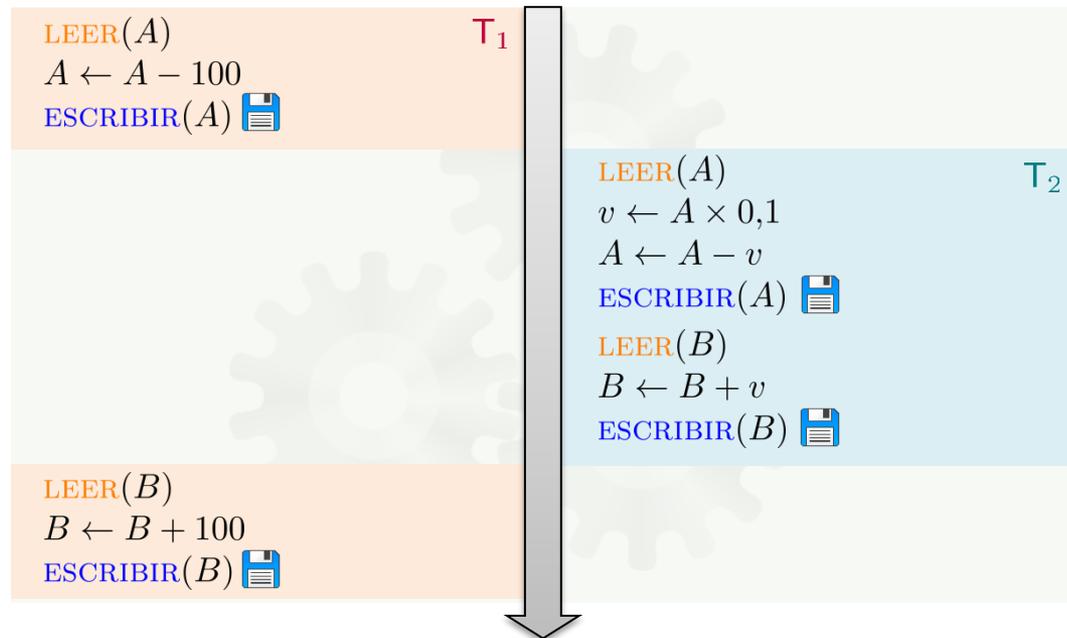
*¿Qué tipo de conflicto hay aquí?*

**Conflicto de Escritura–Lectura:**

A cambia con T<sub>1</sub> / T<sub>2</sub>  
 B cambia con T<sub>2</sub> / T<sub>1</sub>

*¿Qué tipo de conflicto hay aquí?*

**¡Es secuenciable!**  
 (No hay conflicto)



LEER(A)  
 $A \leftarrow A - 100$   
ESCRIBIR(A) 

$T_1$

LEER(A)  
 $v \leftarrow A \times 0,1$   
 $A \leftarrow A + v$   
ESCRIBIR(A)   
LEER(B)  
 $w \leftarrow B \times 0,1$   
 $B \leftarrow B + w$   
ESCRIBIR(B) 

$T_2$

LEER(B)  
 $B \leftarrow B + 100$   
ESCRIBIR(B) 

¿Qué tipo de conflicto hay aquí?

**Conflicto de Escritura–Lectura:**

A cambia con  $T_1 / T_2$

B cambia con  $T_2 / T_1$

*¡Los casos son parecidos pero uno es secuenciable y el otro no!*

$A \leftarrow A - 100$   
ESCRIBIR(A) 

LEER(A)  
 $v \leftarrow A \times 0,1$   
 $A \leftarrow A - v$   
ESCRIBIR(A)   
LEER(B)  
 $B \leftarrow B + v$   
ESCRIBIR(B) 

$T_2$

LEER(B)  
 $B \leftarrow B + 100$   
ESCRIBIR(B) 

¿Qué tipo de conflicto hay aquí?

**¡Es secuenciable!**

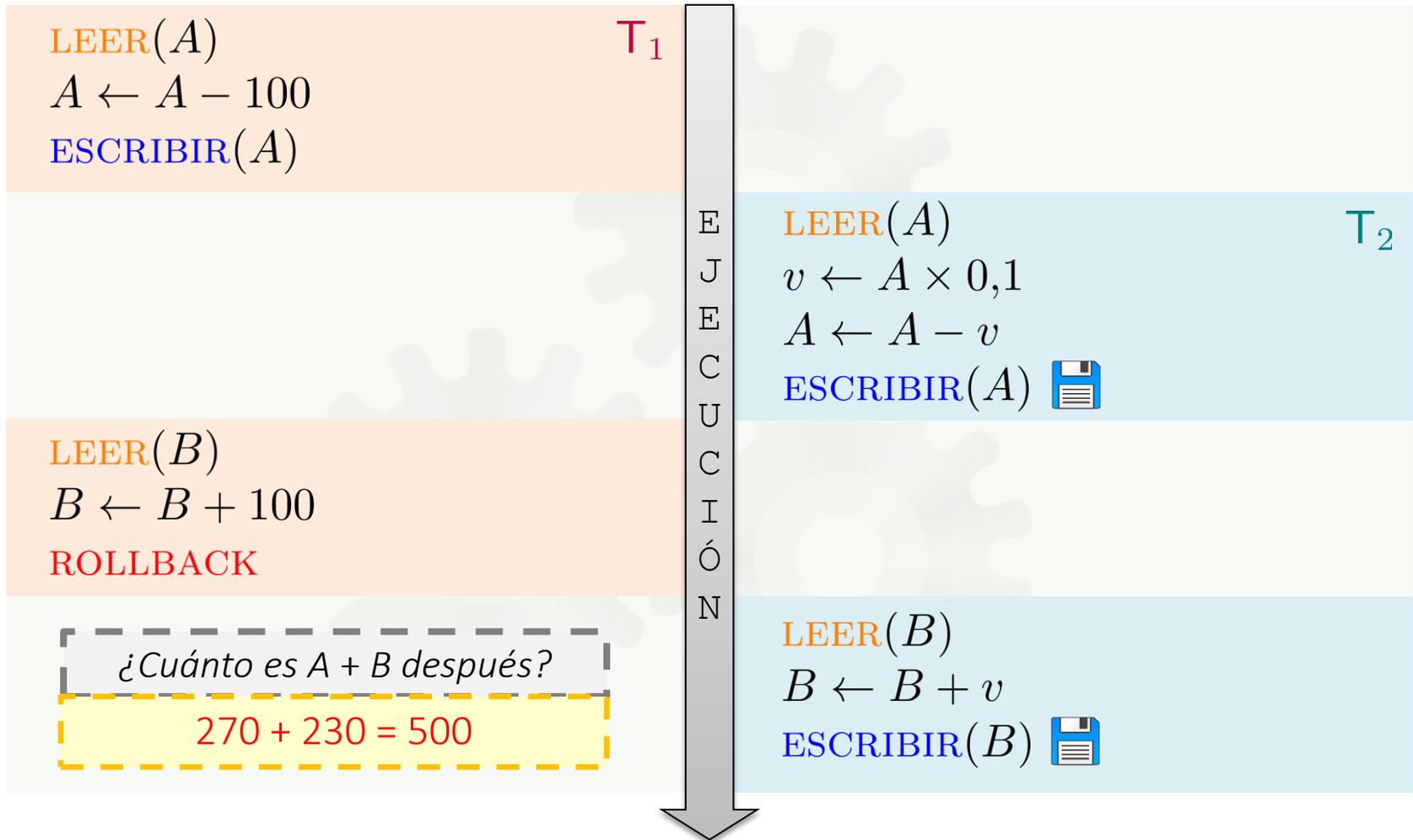
(No hay conflicto)





# (\* ) Conflictos parecidos con anulaciones

Ejemplo 1: Dos transferencias bancarias entre cuentas  $A$  y  $B$   
(donde el valor  $A + B$  no debe cambiar,  $A$  inicia con 400,  $B$  con 200)





# (\* ) Conflictos parecidos con anulaciones

*Ejemplo 1:* Dos transferencias bancarias entre cuentas  $A$  y  $B$

(donde el valor  $A + B$  no debe cambiar,  $A$  inicia con 400,  $B$  con 200)

LEER( $A$ )

$T_1$

- En una **planificación recuperable**, cada transacción  $T$  se compromete (COMMIT) solo después de que se comprometan a su vez todas las transacciones desde las cuáles  $T$  haya leído algo.
- Si cada transacción  $T$  solamente lee cambios de transacciones comprometidas, se pueden **evitar anulaciones en cascada** y mantener una **planificación recuperable**. De lo contrario, si  $T$  lee algo de una transacción no comprometida  $T'$  y hay que anular  $T'$ , entonces puede ser que haya que anular  $T$  también (una **anulación en cascada**) para mantener una **planificación recuperable**.

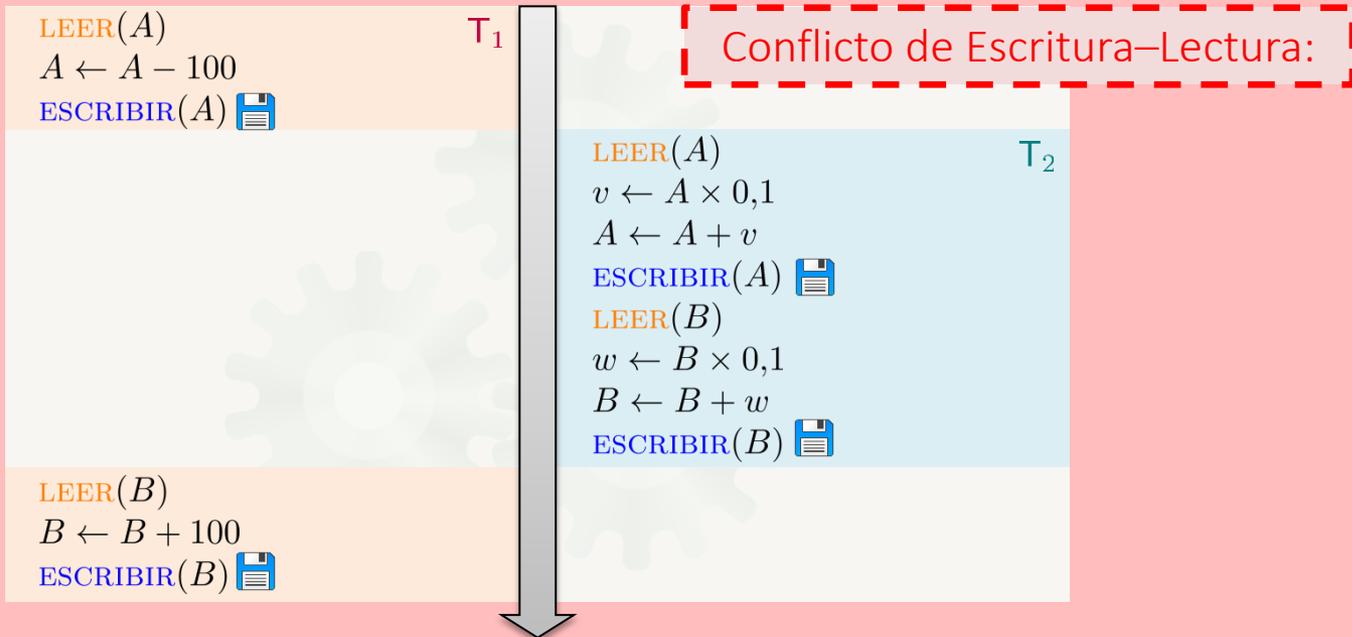
Para cumplir con ACID, un sistema de bases de datos debe garantizar que solamente se permitan **planificaciones recuperables**.





SOLUCIÓN: BLOQUEOS (*LOCKS*)

# El problema



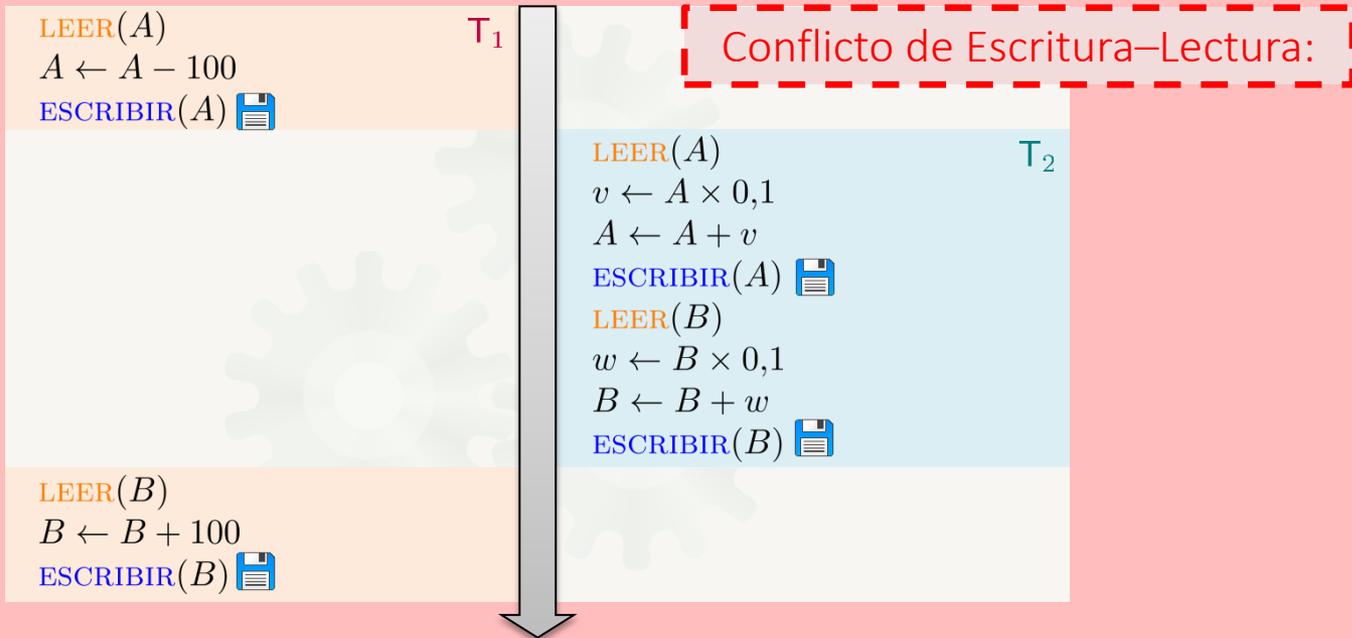
*¿Han visto un problema así antes de este curso?*

Multihilos (Multithreading)

*¿Y qué era la solución (en Java por ejemplo)?*

Bloques sincronizados

# Una solución en Java



```
public void trans(Cuenta A, Cuenta B, long val){
    synchronized(this.tablaCuentas) {
        long aAntiguo = tablaCuentas.leer(A);
        long aNuevo = aAntiguo - val;
        tablaCuentas.escribir(A,aNuevo);
        long bAntiguo = tablaCuentas.leer(B);
        long bNuevo = bAntiguo + val;
        tablaCuentas.escribir(B,bNuevo);
    }
}
```

```
public void interes(Cuenta[] Cs, float tasa){
    synchronized(this.tablaCuentas) {
        for(Cuenta C:Cs) {
            long cAntiguo = tablaCuentas.leer(C);
            long cNuevo = Math.round(cAntiguo*(1+tasa));
            tablaCuentas.escribir(C,cNuevo);
        }
    }
}
```

# Una solución en Java

LEER(A)

Ti

Conflicto de Escritura Lectura

*¿Cómo se podría mejorar el rendimiento del código abajo?*

Minimizar el código sincronizado tanto como sea posible

Usar objetos de bloqueos tan específico que sea posible

(por ejemplo, usar un valor de la tabla o una fila de la tabla como el objeto de bloqueo, no la entera tabla, si es posible)

ESCRIBIR(B)

LEER(B)

$B \leftarrow B + 100$

**¡Los mismos conceptos aplican a sistemas de bases de datos!**

Salvo que es el sistema, no el programador, que tiene que decidir (automáticamente) el nivel de sincronización, el nivel de bloqueo, etc.

Una transacción tiene que conseguir **bloqueos** para los objetos (valores, filas, vista, tablas) que quiere modificar.

Cuando haya terminado con el bloque, lo libera.

Un **protocolo de bloqueo** especifica las reglas que las transacciones tienen que seguir con respecto a bloqueos.

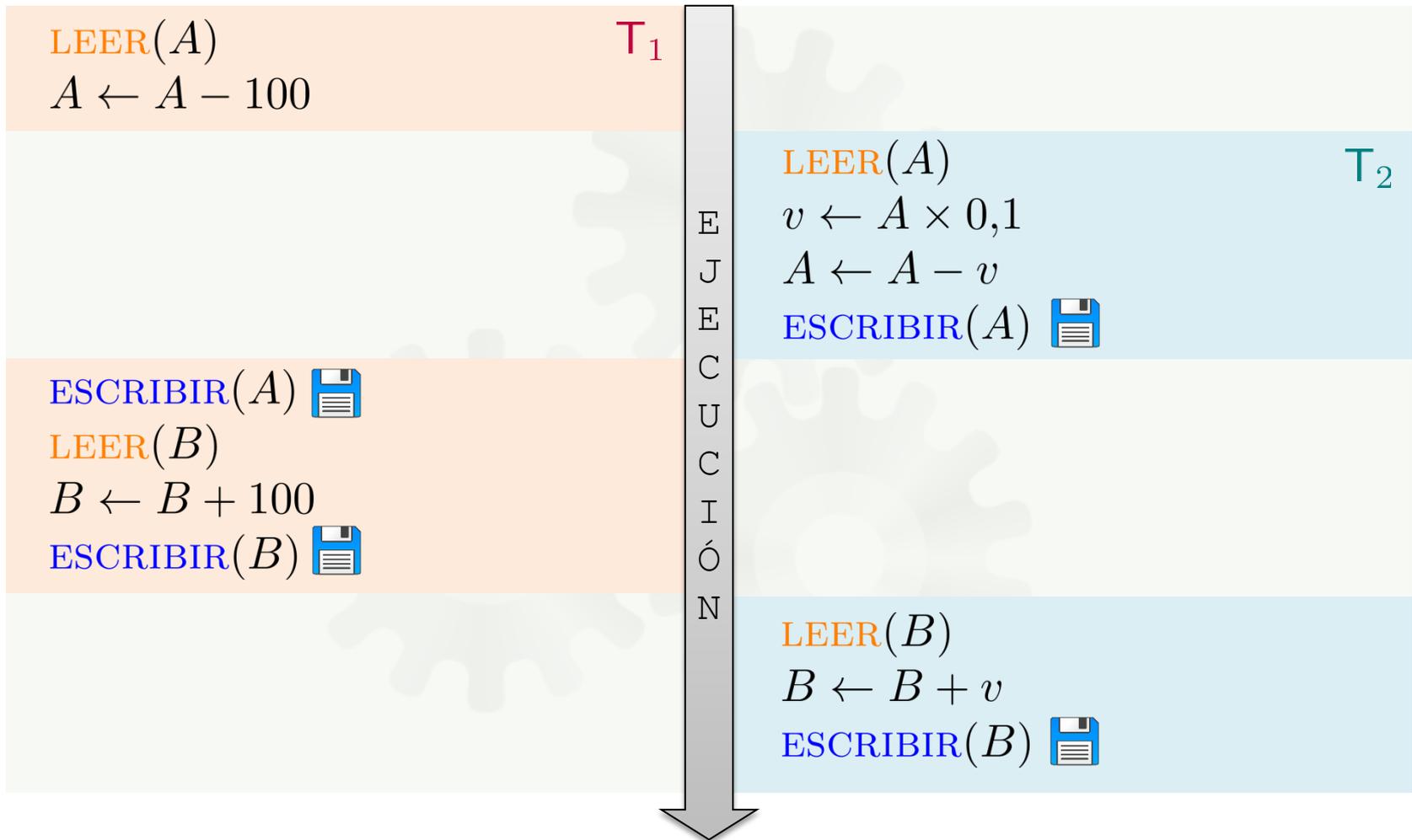
```
public void t
synchronize
long aAnt
long aNue
tablaCuen
long bAnt
long bNue
tablaCuen
}
```

```
tasa){
r(C);
o*(1+tasa));
```

# Bloqueos exclusivos y compartidos

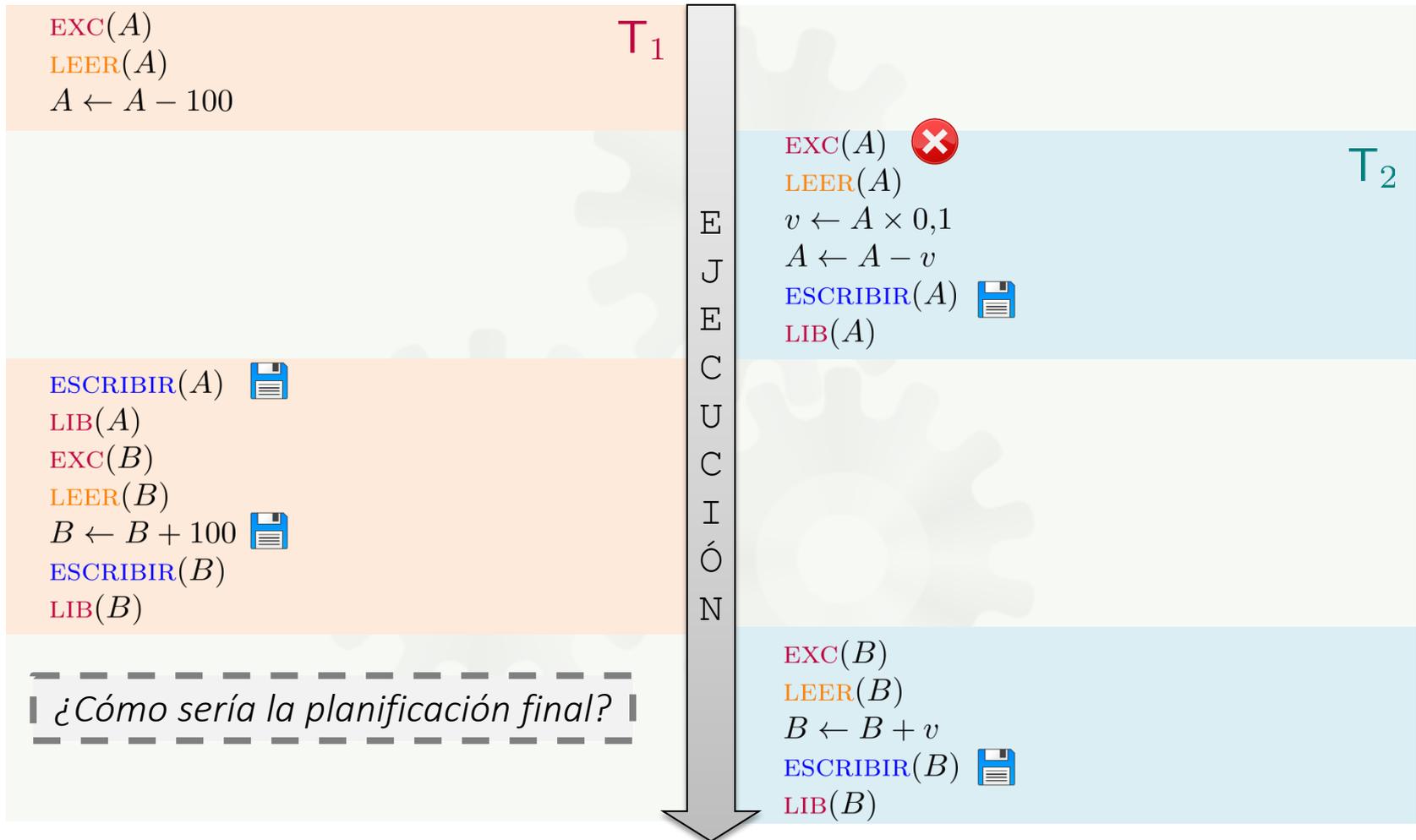
- Si una transacción T quiere modificar un objeto O, tiene que conseguir un **bloqueo exclusivo** sobre O
  - T puede leer o modificar O
  - Ninguna transacción (aparte de T) puede leer ni modificar O mientras que T tenga su bloqueo exclusivo sobre O
- Si una transacción T solo quiere leer un objeto O, puede conseguir un **bloqueo compartido** sobre O
  - T solo puede leer O
  - Varias transacciones pueden leer el mismo objeto al mismo tiempo
  - Un bloqueo exclusivo excluye bloqueos compartidos

# Conflicto de Lectura–Escritura



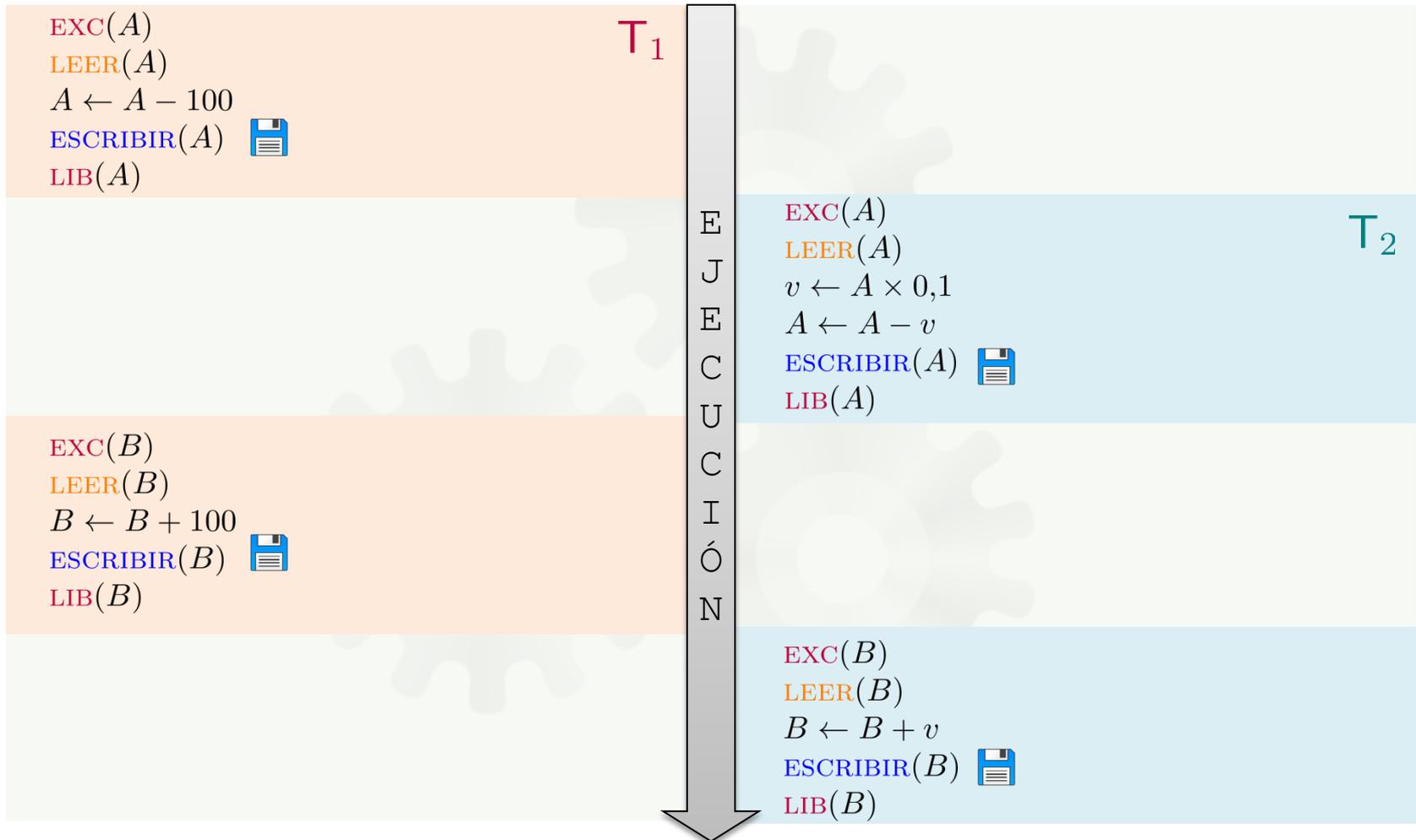
# ... con bloqueos exclusivos

EXC( $\cdot$ ) Bloqueo Exclusivo  
LIB( $\cdot$ ) Liberar

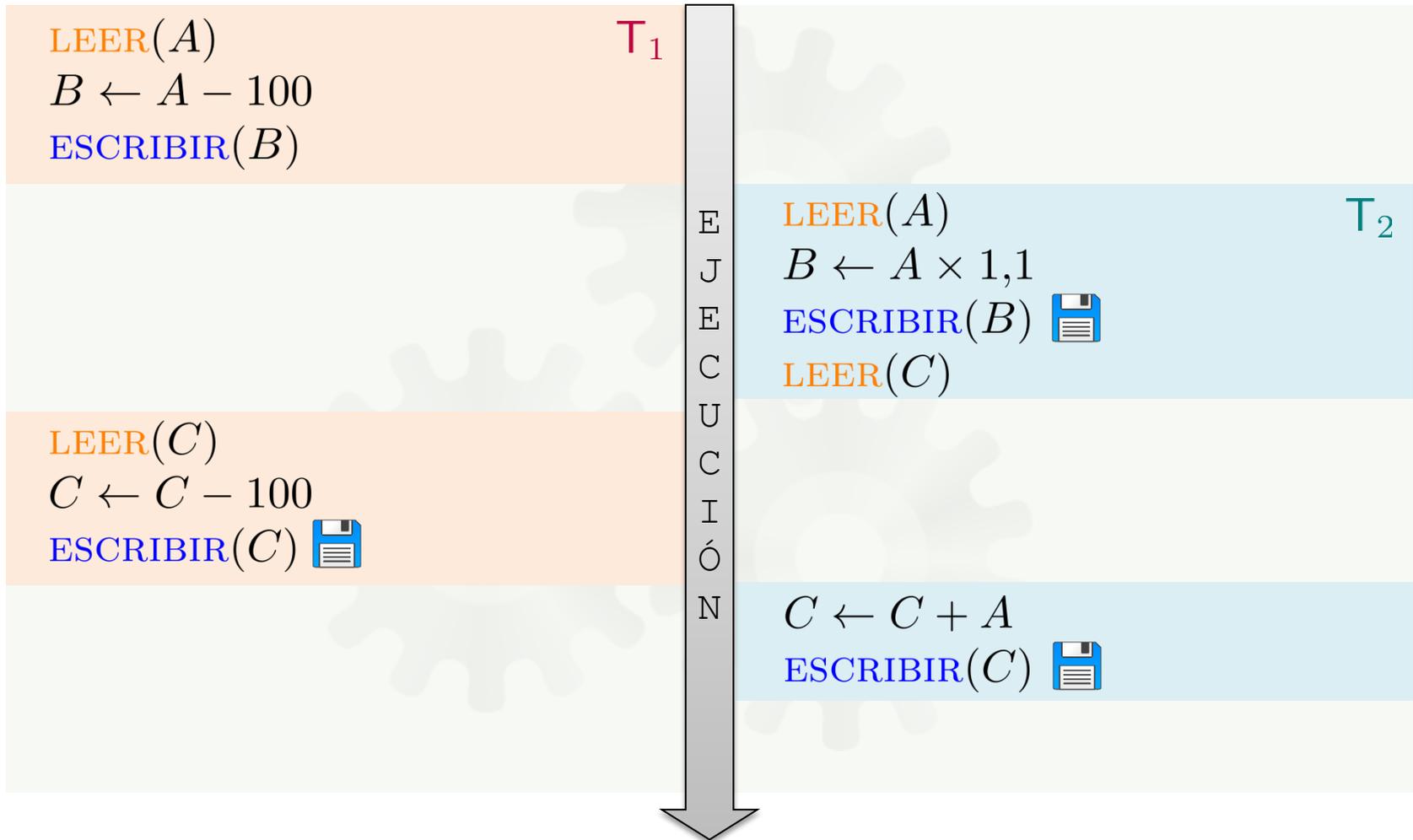


# ... con bloqueos exclusivos

EXC( $\cdot$ ) Bloqueo Exclusivo  
LIB( $\cdot$ ) Liberar

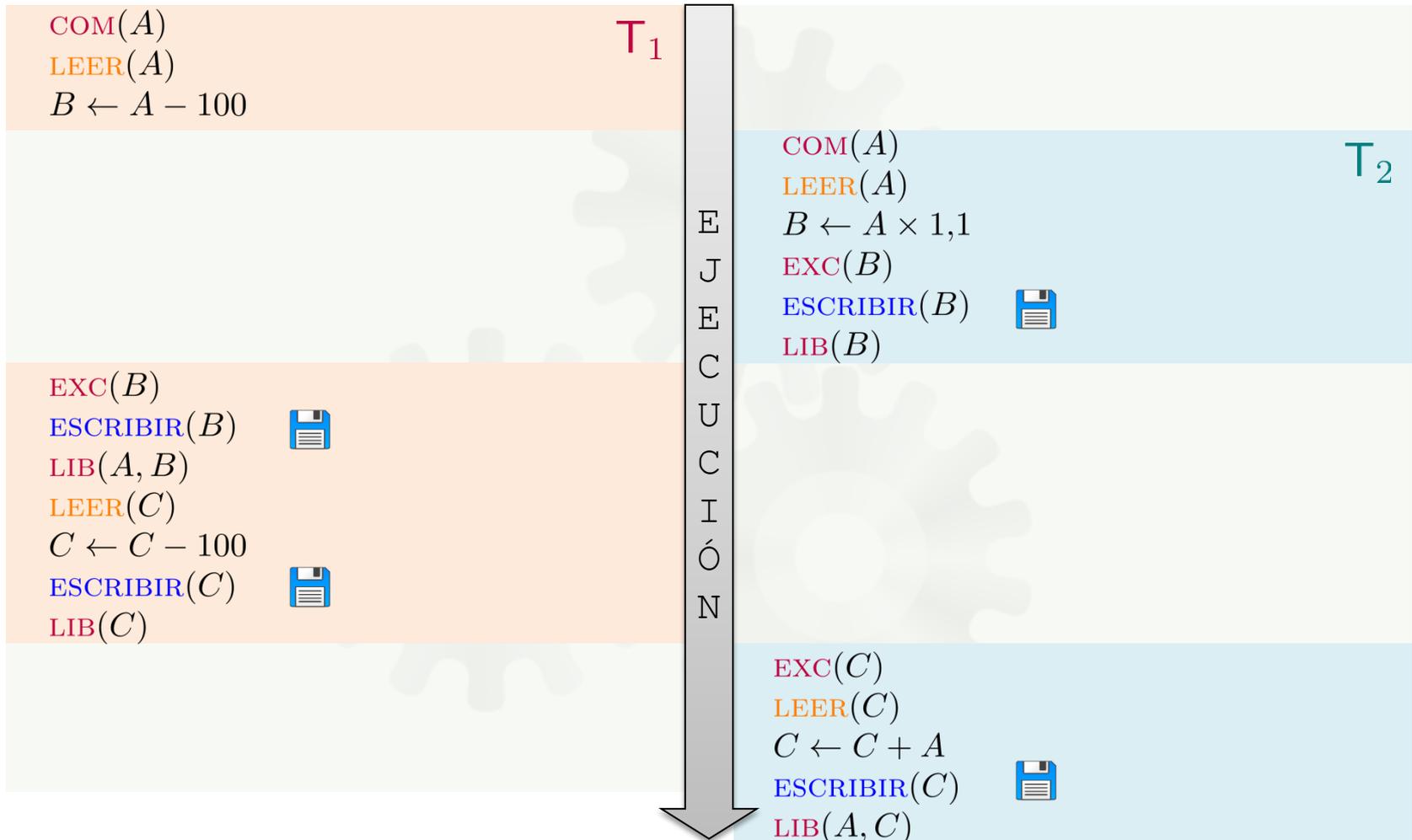


# Conflicto de Lectura–Escritura

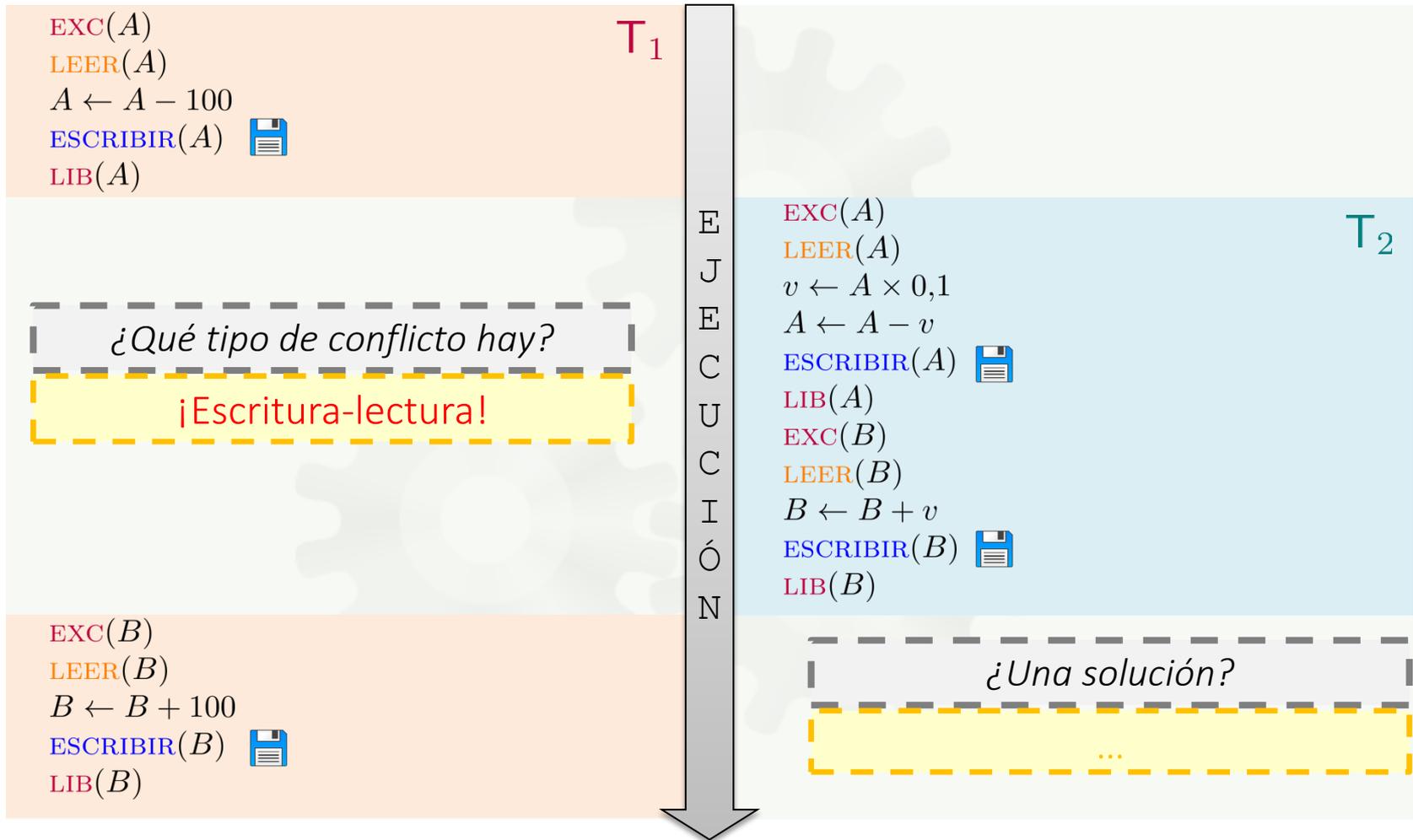


# ... con bloqueos exclusivos y compartidos

EXC( $\cdot$ ) Bloqueo **Ex**clusivo  
COM( $\cdot$ ) Bloqueo **Com**partido  
LIB( $\cdot$ ) **Lib**erar



# Bloqueo: Un problema



# Bloqueo en dos fases

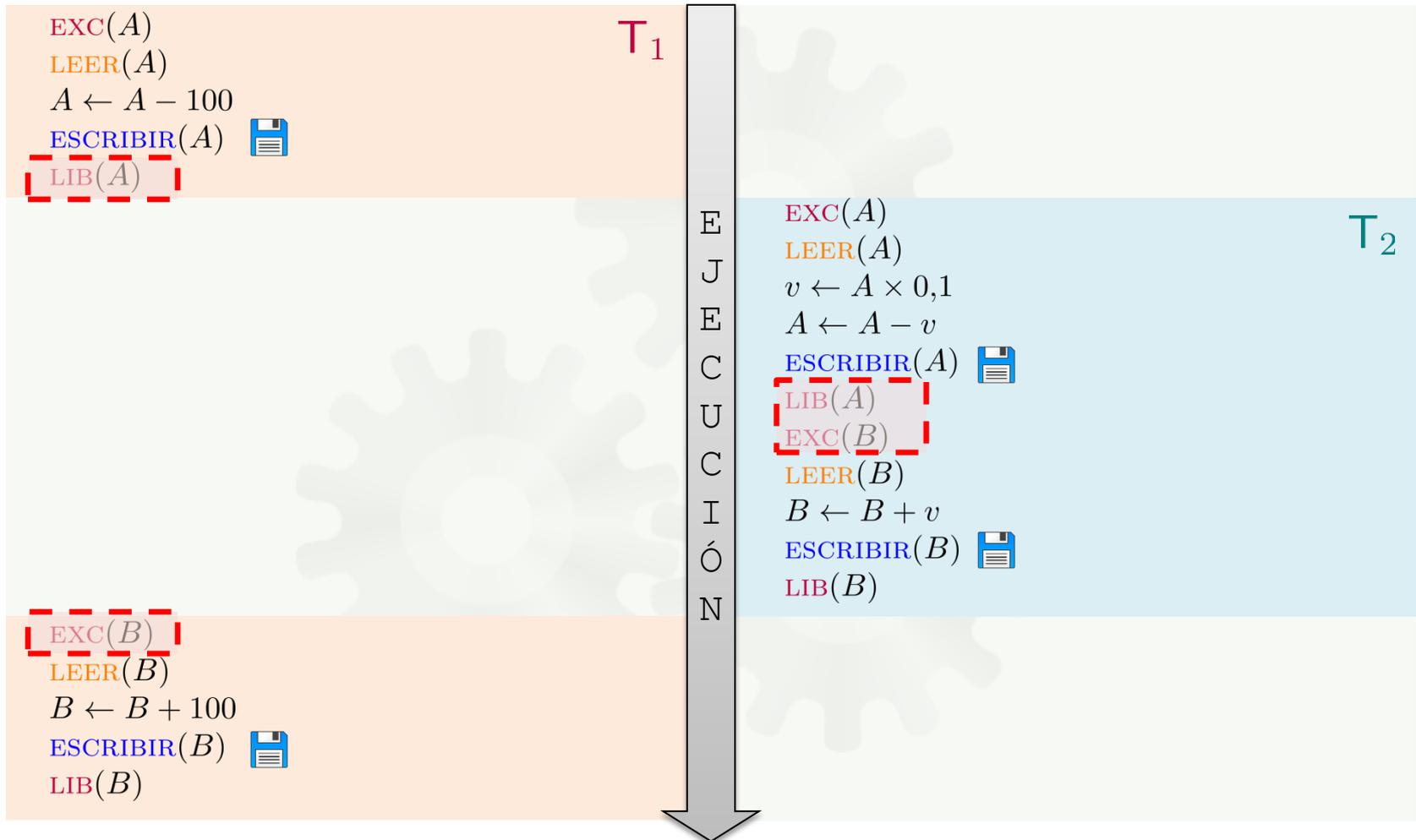
- Primera fase (de expansión):
  - Se adquieren bloqueos, pero no se liberan bloqueos
- Segunda fase (de reducción)
  - Se liberan bloques, pero no se adquieren bloqueos

No se pueden adquirir más bloqueos después de haber liberado algún bloqueo

# Bloqueo en dos fases

¿Bloqueo en dos fases?

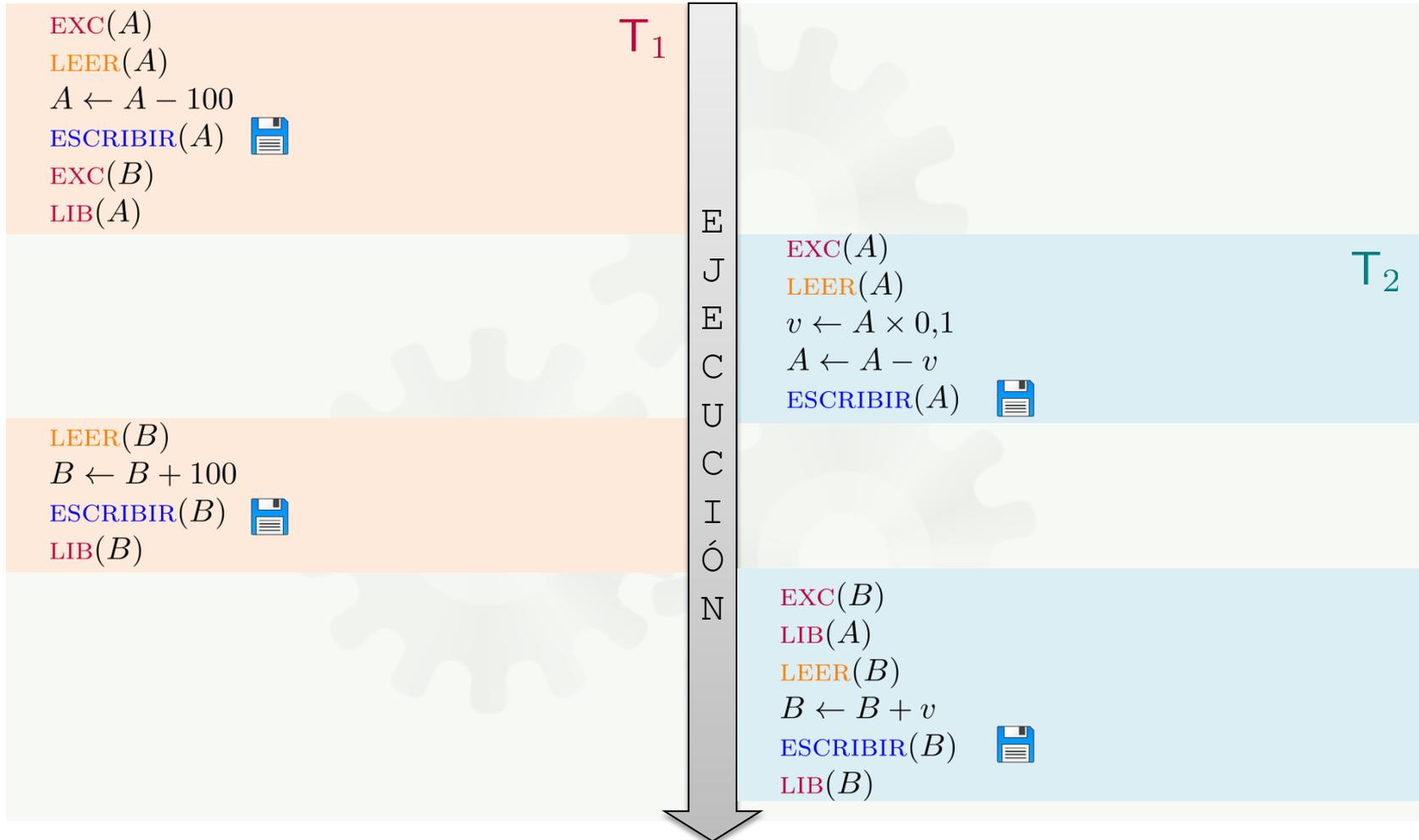
No ...



# Bloqueo en dos fases

¿Bloqueo en dos fases?

Sí



# Bloqueo en dos fases

- Primera fase (de expansión):
  - Se adquieren bloqueos, pero no se liberan bloqueos
- Segunda fase (de reducción)
  - Se liberan bloques, pero no se adquieren bloqueos

No se puede adquirir más bloqueos después de haber liberado algún bloqueo

¡Garantiza **secuenciabilidad** (que no habrá conflictos)!

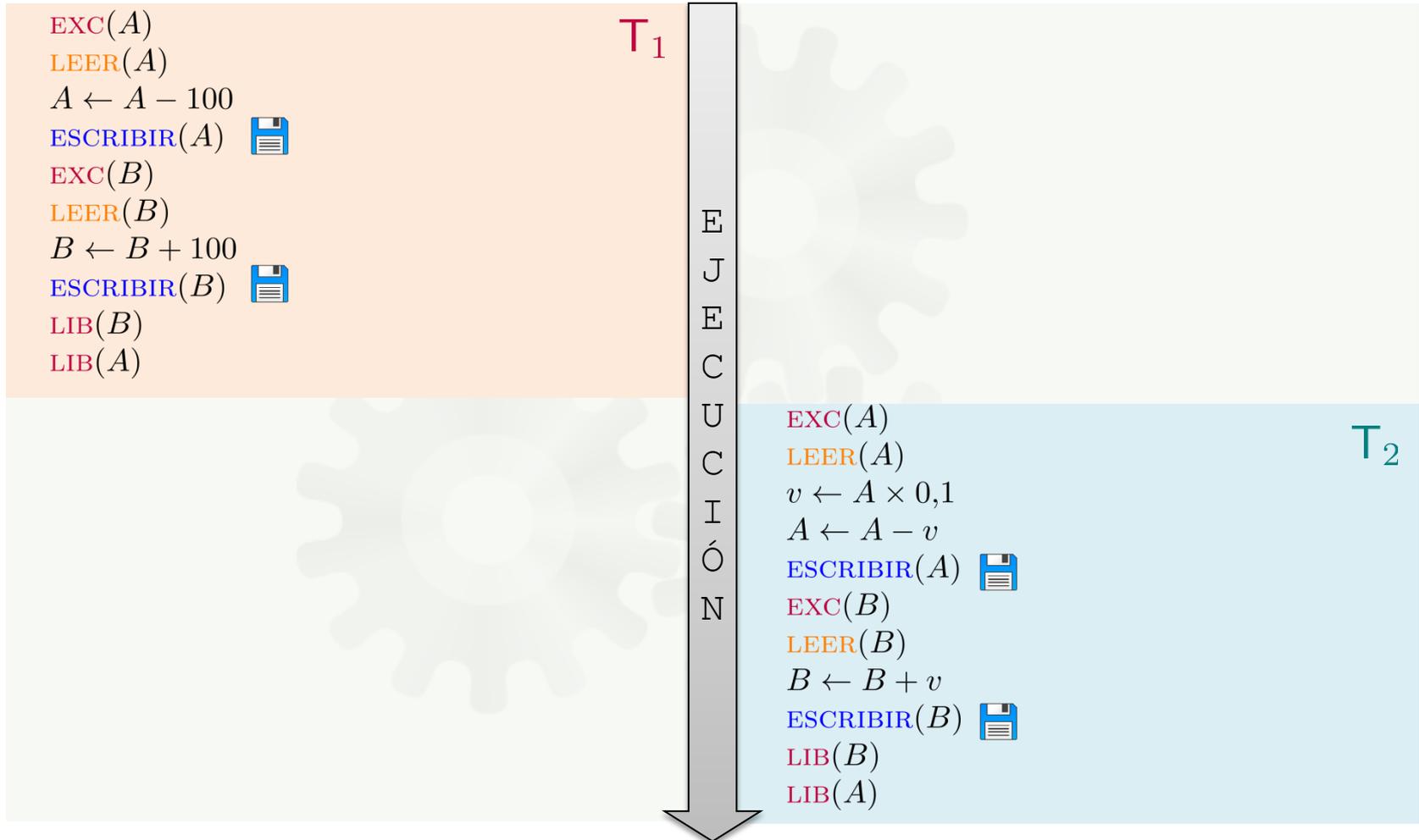
## Bloqueo en dos fases: **estricto**

- Bloqueo en dos fases **y**:
  - La transacción T liberará sus bloqueos exclusivos solo cuando haya terminado (al final)

# Bloqueo en dos fases: **estricto**

¿Bloqueo en dos fases estricto?

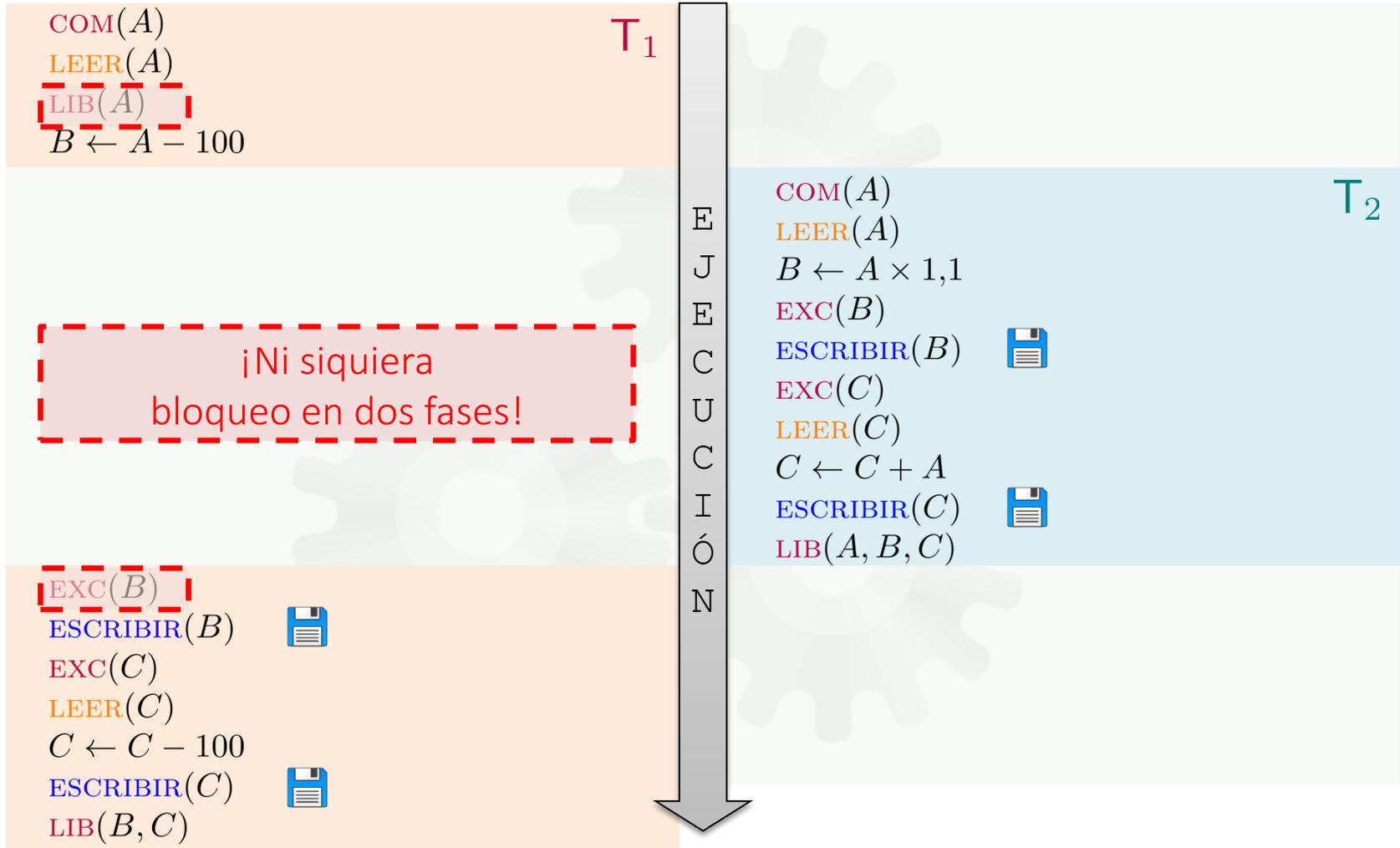
Sí



# Bloqueo en dos fases: **estricto**

¿Bloqueo en dos fases estricto?

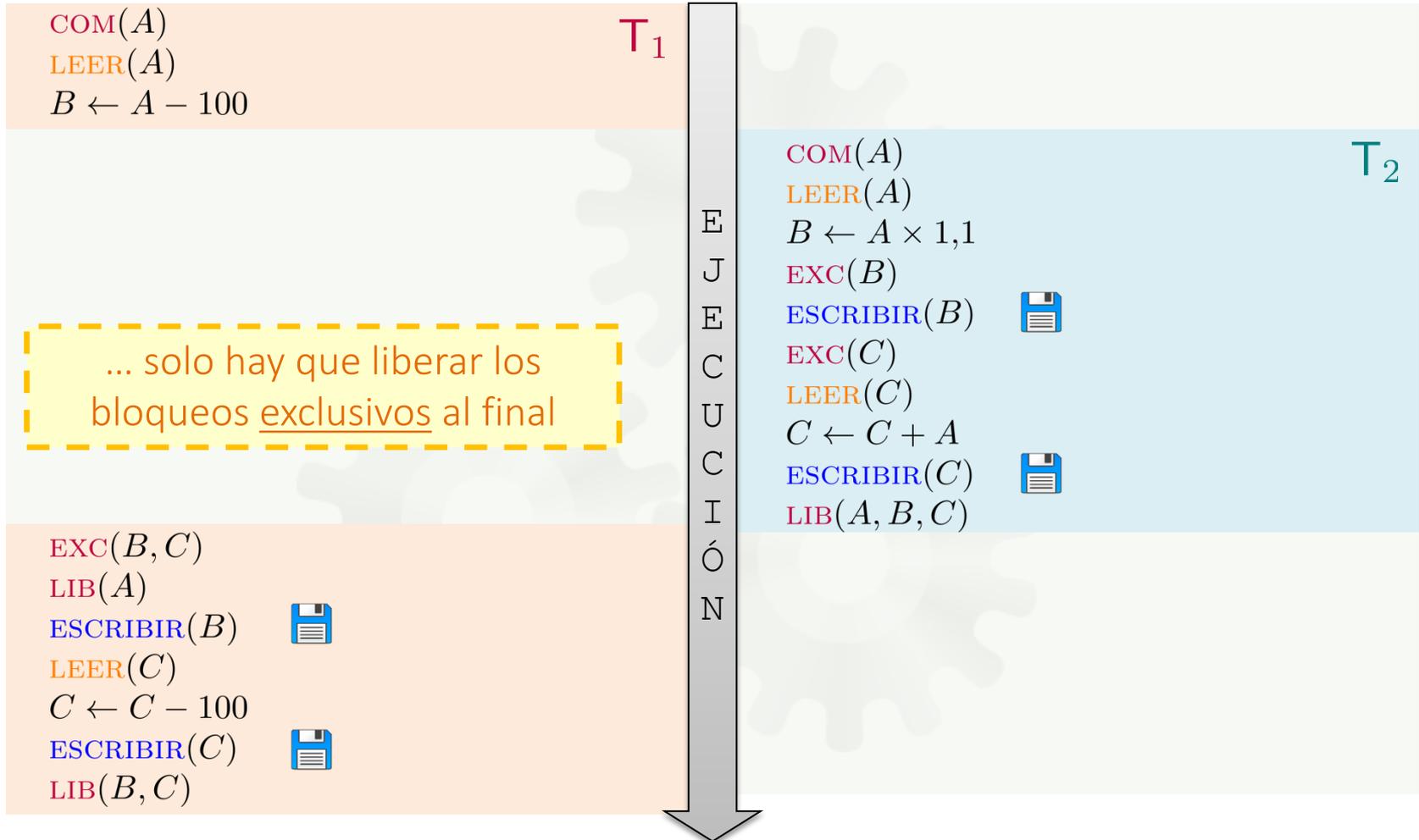
No ...



# Bloqueo en dos fases: **estricto**

¿Bloqueo en dos fases estricto?

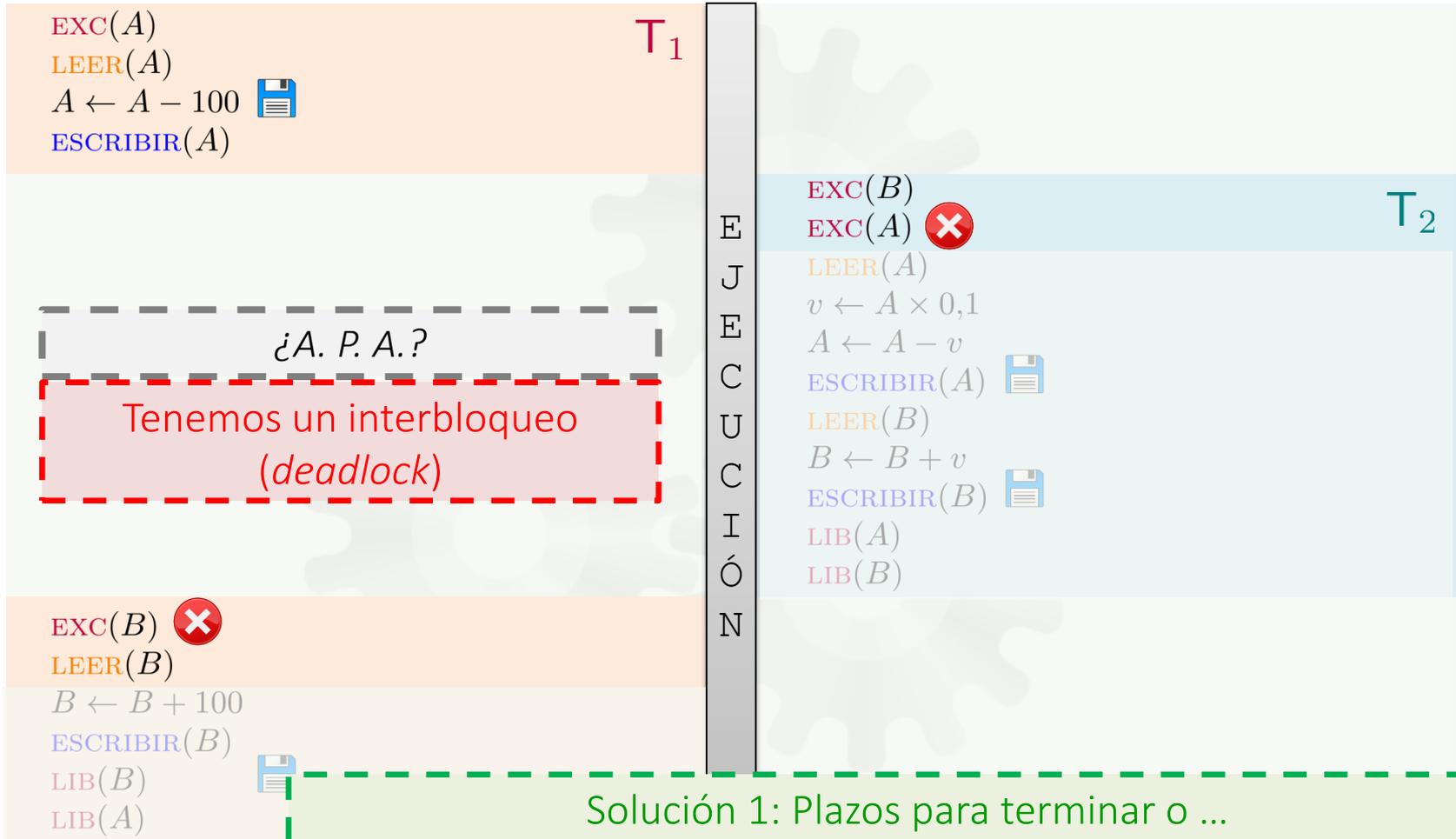
Sí



# Bloqueo en dos fases: **estricto**

¿Bloqueo en dos fases estricto?

Sí



Solución 1: Plazos para terminar o ...

Solución 2: Detectar ciclos en transacciones esperando o ...

# Bloqueo en dos fases: conservador

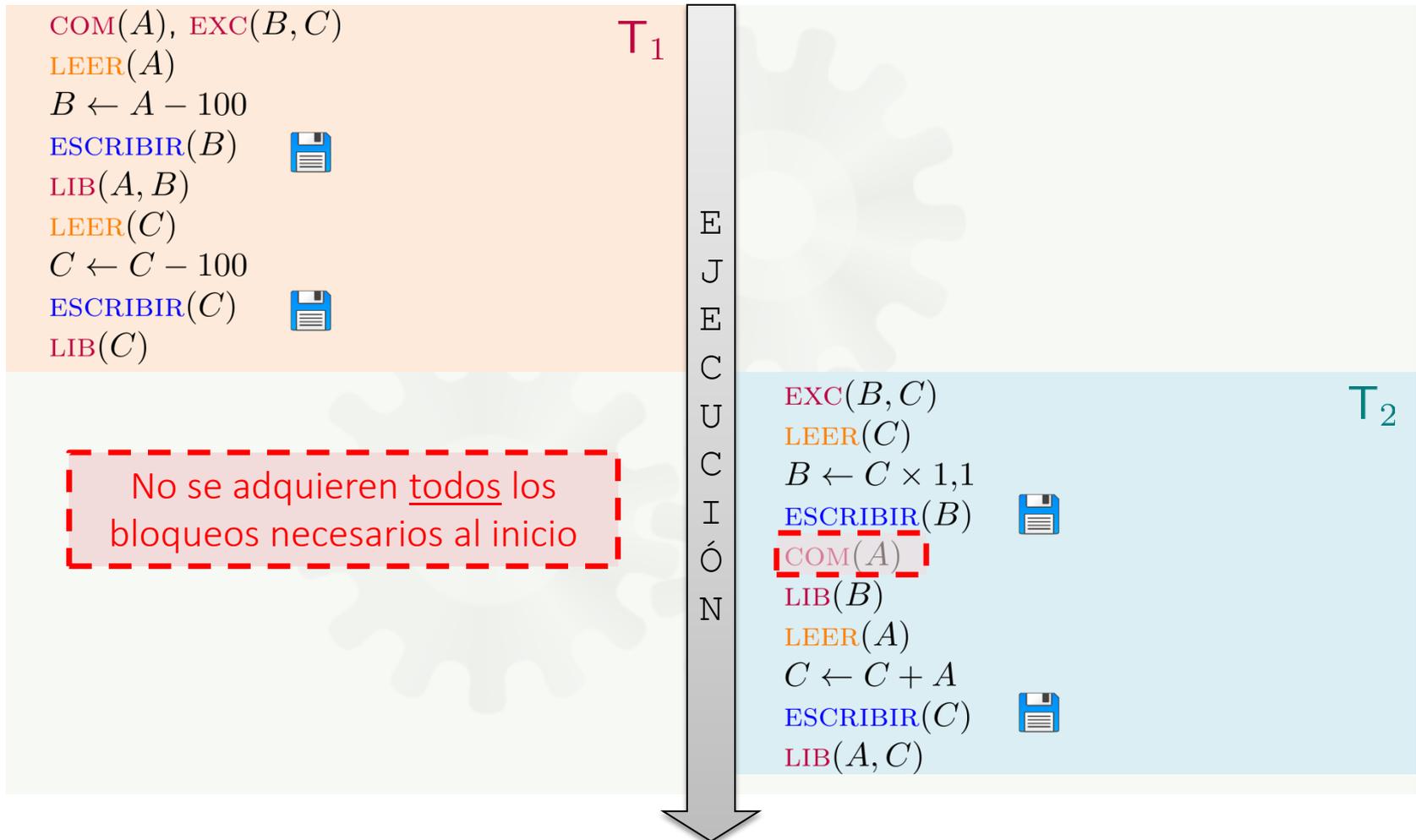
- Bloqueo en dos fases y:
  - La transacción adquieren todos los bloqueos necesarios al inicio y de una forma atómica

¡Evita interbloqueos!

# Bloqueo en dos fases: **conservador**

¿Bloqueo en dos fases conservador?

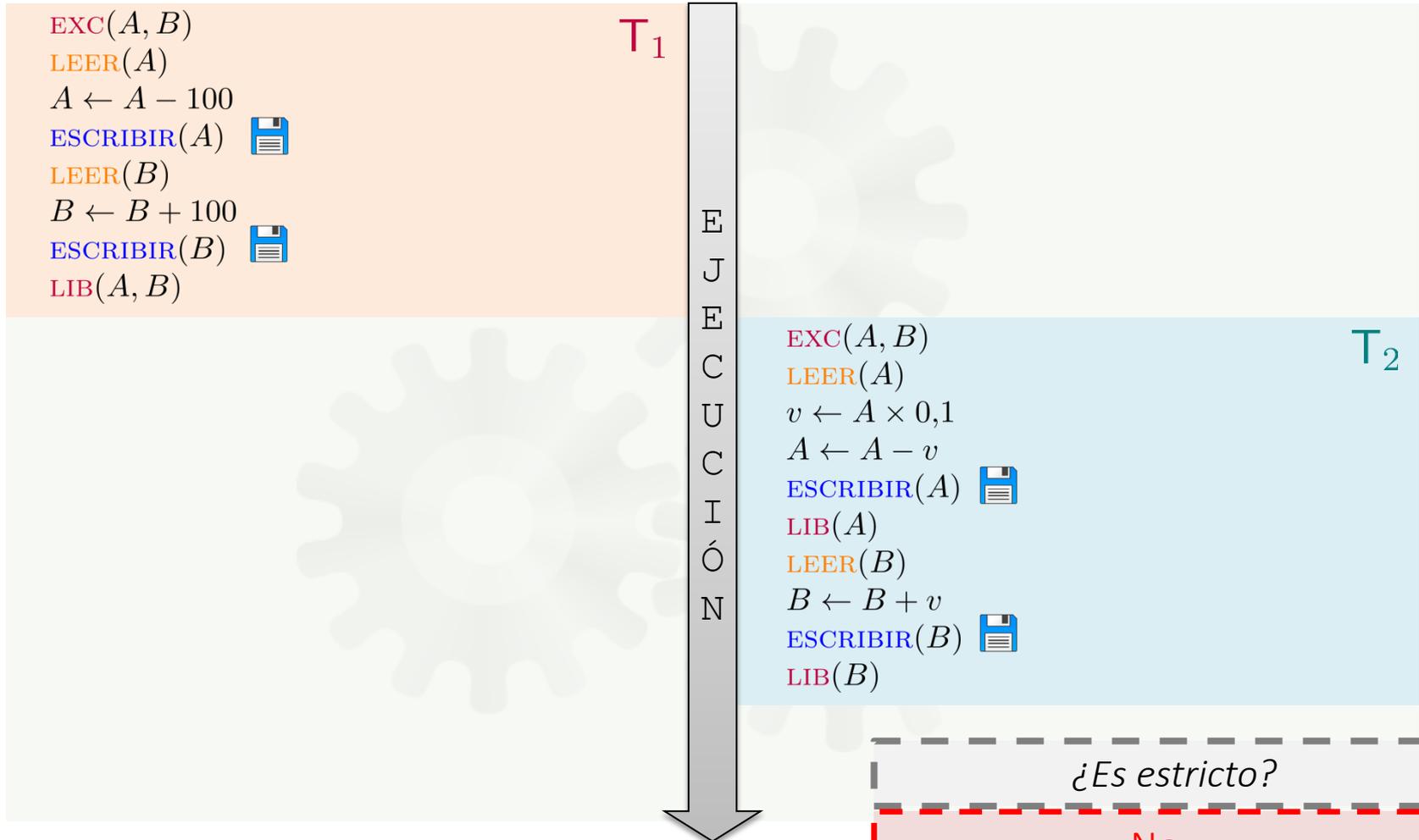
No ...



# Bloqueo en dos fases: **conservador**

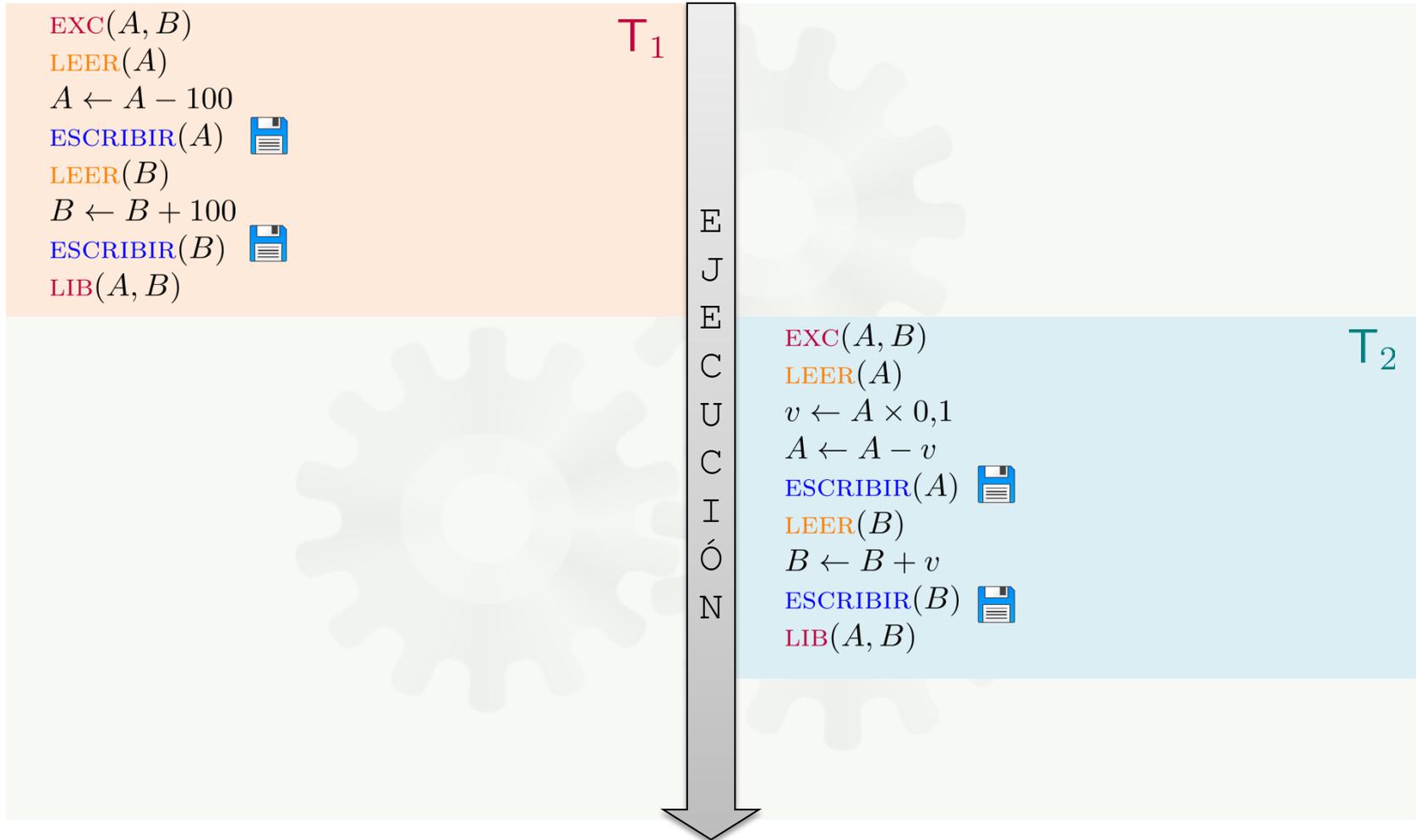
*¿Bloqueo en dos fases conservador?*

Sí



# Bloqueo en dos fases **estricto** y **conservador**

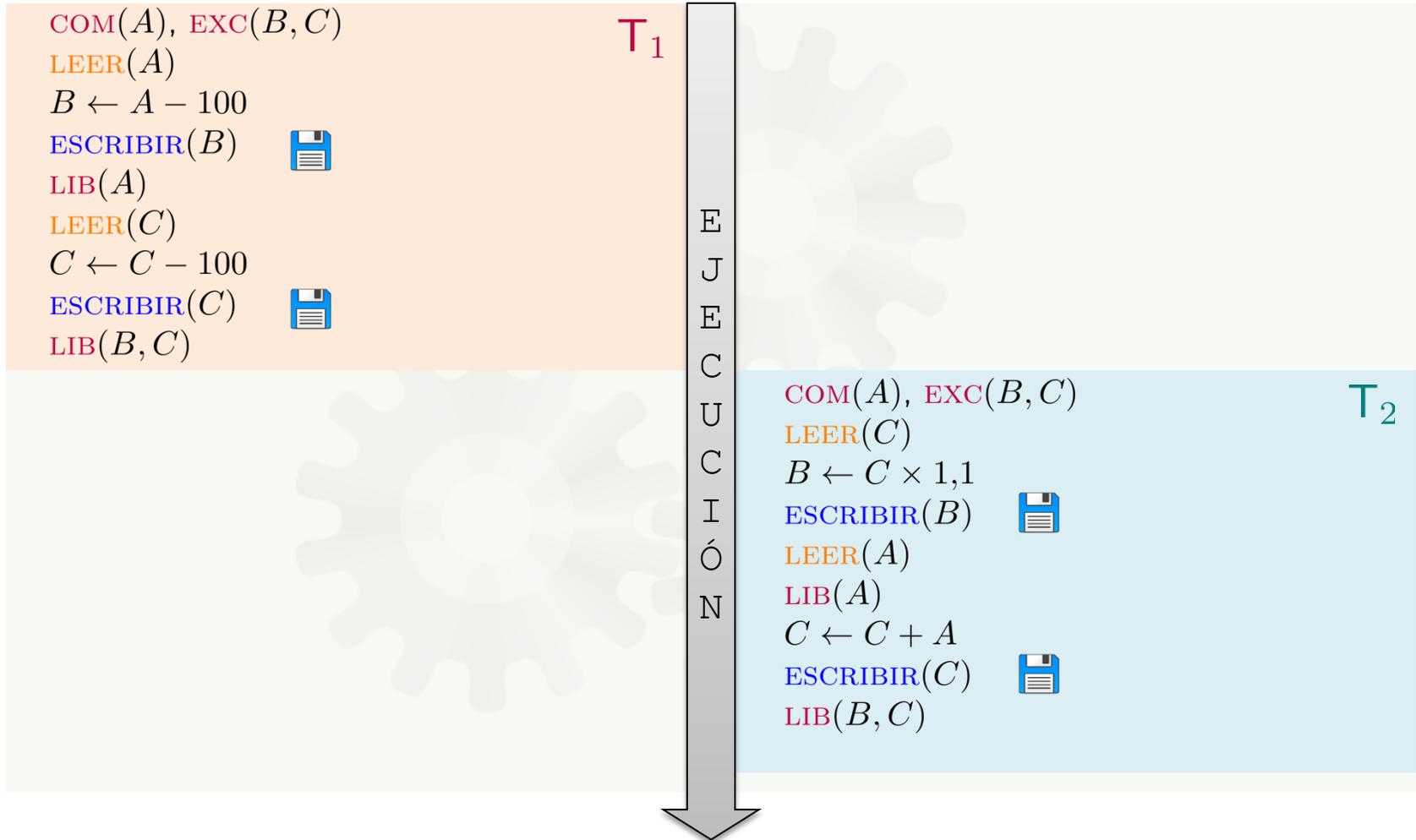
Paralelizar solo las transacciones que no compartan bloqueos exclusivos  
(lo más "fácil" pero lo menos paralelizable)

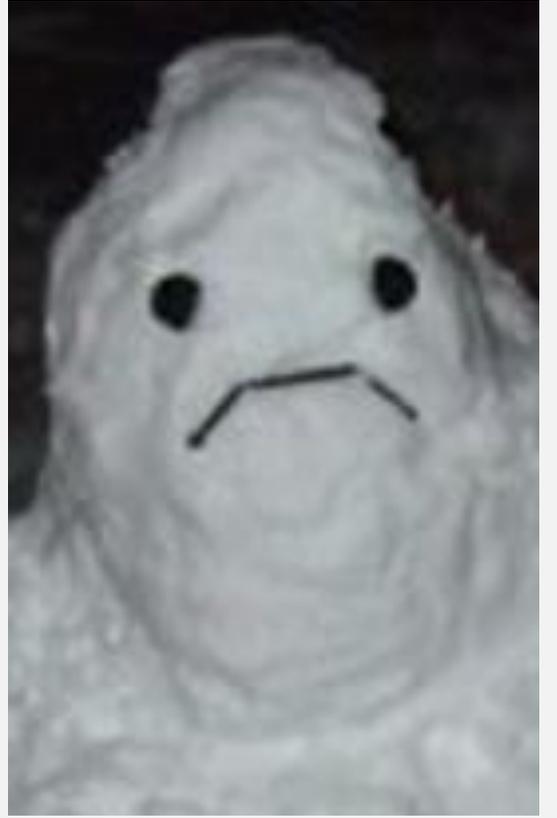


# Bloqueo en dos fases **estricto** y **conservador**

¿Bloqueo en dos fases estricto y conservador?

Sí





Pero para un usuario ...





Hemos terminado con bases de datos relacionales

Preguntas?

