CC3201-1
Bases de Datos
Primavera 2016

Clase 12: Implementación de ACID

Aidan Hogan
aidhog@gmail.com

Transacciones

Una transacción es

un conjunto de operaciones que

se ejecutan de manera atómica

(es decir, como fueran una sola operación)

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 INTEGER PRIMARY KEY,
    total_gasto BIGINT,
    total_ingreso BIGINT,
    saldo BIGINT,
    CHECK (total_ingreso - total_gasto = saldo)
)
```

ACID: Atomicidad

```
CREATE TABLE Balance (
    cuenta INTEGER 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;
```

No se puede actualizar el saldo sin actualizar el gasto directamente después.

(Si alguna actualización falla, ambas fallan.)

ACID: Consistencia

```
CREATE TABLE Balance (
    cuenta INTEGER 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;
```

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

ACID: Aislamiento (*Isolation*)

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

T1

```
START TRANSACTION
  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;

T2

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

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 INTEGER 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;
```

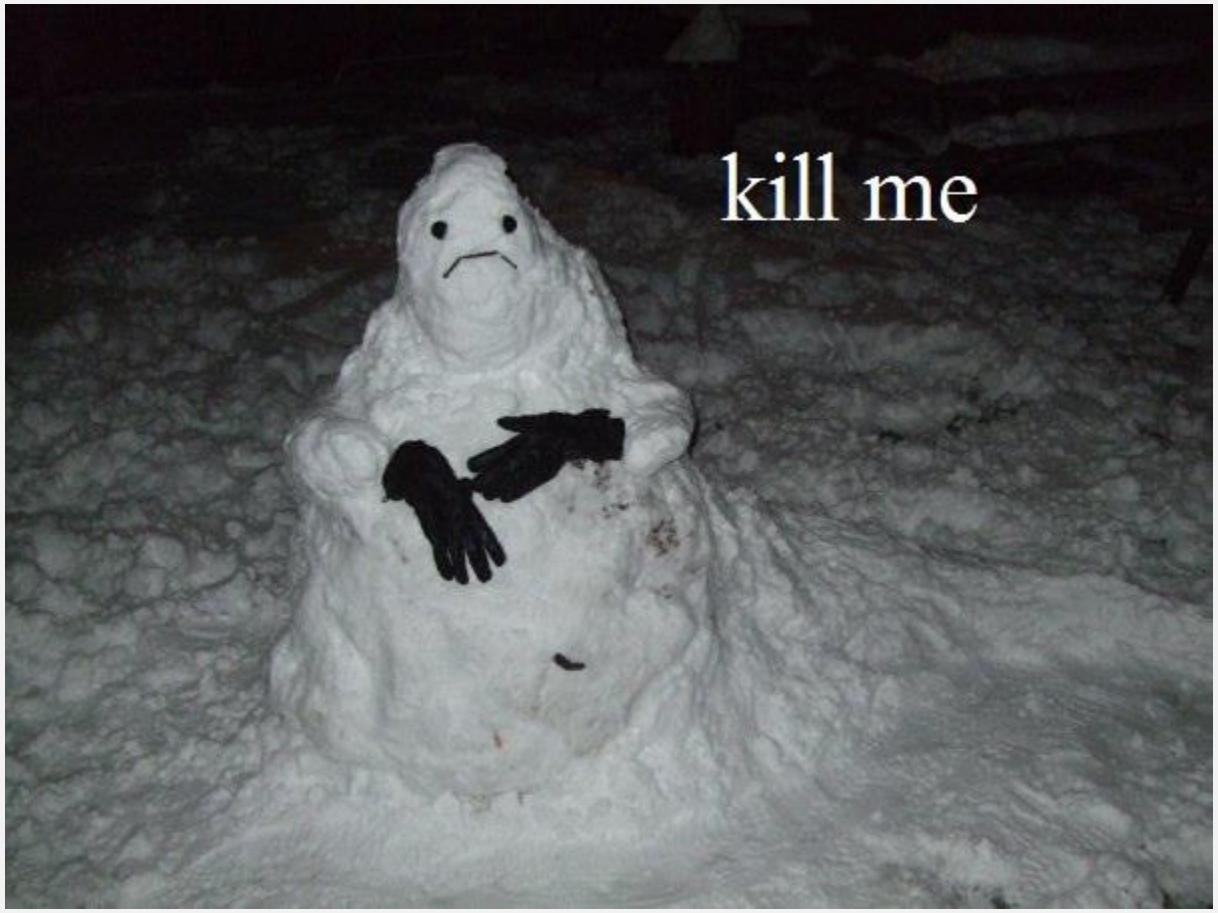
Una vez que haya un COMMIT exitoso, se persisten los cambios.

(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 olvidaré los cambios silenciosamente.)

Entonces con las garantías de ACID …

… todo está tranquilo para un usuario.

… pero si uno tiene que *implementar* ACID



… uff.

Cuando **no** tenemos ACID …

- Atomicidad:
  - × Una transacción se ejecutan a medidas 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 ...

T

LEER(A)
$A \leftarrow A - 100$
ESCRIBIR(A) 💾
LEER(B)
$B \leftarrow B + 100$
ESCRIBIR(B) 💾

← Dejamos el COMMIT implicito al fin de la transacción.

Cuando **no** tenemos ACID …

- Atomicidad:
  - × Una transacción se ejecutan a medidas 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.

¿Cómo puede una implementación evitar estas tipas de problemas? …

Modelando una transacción

- Atomicidad:
  - × Una transacción se ejecutan a medidas pero afecta el estado de la base de datos

Ejemplo: Una transferencia bancaria ...

T

$\text{LEER}(A)$
$A \leftarrow A - 100$
$\text{ESCRIBIR}(A)$ 💾
$\text{LEER}(B)$
$B \leftarrow B + 100$
$\text{ESCRIBIR}(B)$ 💾

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

IMPLEMENTANDO UNA BASE DE DATOS CON ACID: REGISTROS (*LOGGING*)

Mantener un registro de la transacción

Ejemplo: Una transferencia bancaria ...

T

$\text{LEER}(A)$
$A \leftarrow A - 100$
$\text{ESCRIBIR}(A)$ 💾
$\text{LEER}(B)$
$B \leftarrow B + 100$
$\text{ESCRIBIR}(B)$ 💾

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 ...

T

$\textsc{leer}(A)$
$A \leftarrow A - 100$
$\textsc{escribir}(A)$
$\textsc{leer}(B)$
$B \leftarrow B + 100$
$\textsc{escribir}(B)$

Registros ayudan con …

- Atomicidad:
  - × Una transacción se ejecutan a medidas 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)
)
```

T1

```
START TRANSACTION
  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;

T2

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

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).

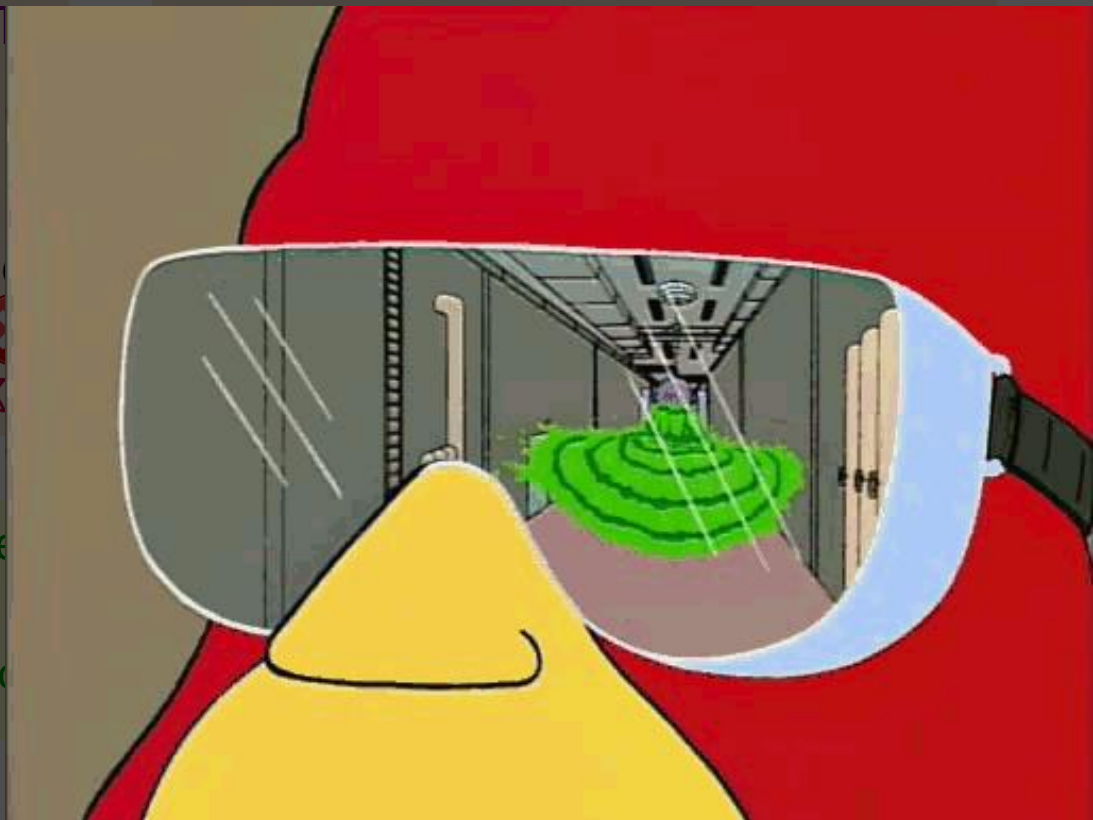
Concurrencia/Aislamiento: un problema abierto

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

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

Una transacción no pue

En (4), hay que tener cuidado con
saldo antes del paso (1) p



IMPLEMENTANDO UNA BASE DE DATOS CON ACID:
SECUENCIABILIDAD (*SERIALIZABILITY*)

Capítulo 8.3 (*es*)/16.3 (*en*) Database Management Systems,
Ramakrishnan / Gehrke (Third Edition)

Registros no ayudan con …

- Atomicidad:
  - × Una transacción se ejecutan a medidas 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)

T_1

$\text{LEER}(A)$
$A \leftarrow A - 100$
$\text{ESCRIBIR}(A)$
$\text{LEER}(B)$
$B \leftarrow B + 100$
$\text{ESCRIBIR}(B)$

EJECUCIÓN

¿Cuánto es A + B después?

270 + 330 = 600

T_2

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

¿Hay un problema aquí?

¡Ejecución serial es lenta!

¿Se pueden ejecutar partes de T_1 y T_2 en 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)

T_1

$\text{LEER}(A)$
$A \leftarrow A - 100$
$\text{ESCRIBIR}(A)$

$\text{LEER}(B)$
$B \leftarrow B + 100$
$\text{ESCRIBIR}(B)$

¿Cuánto es A + B después?

270 + 330 = 600

EJECUCIÓN

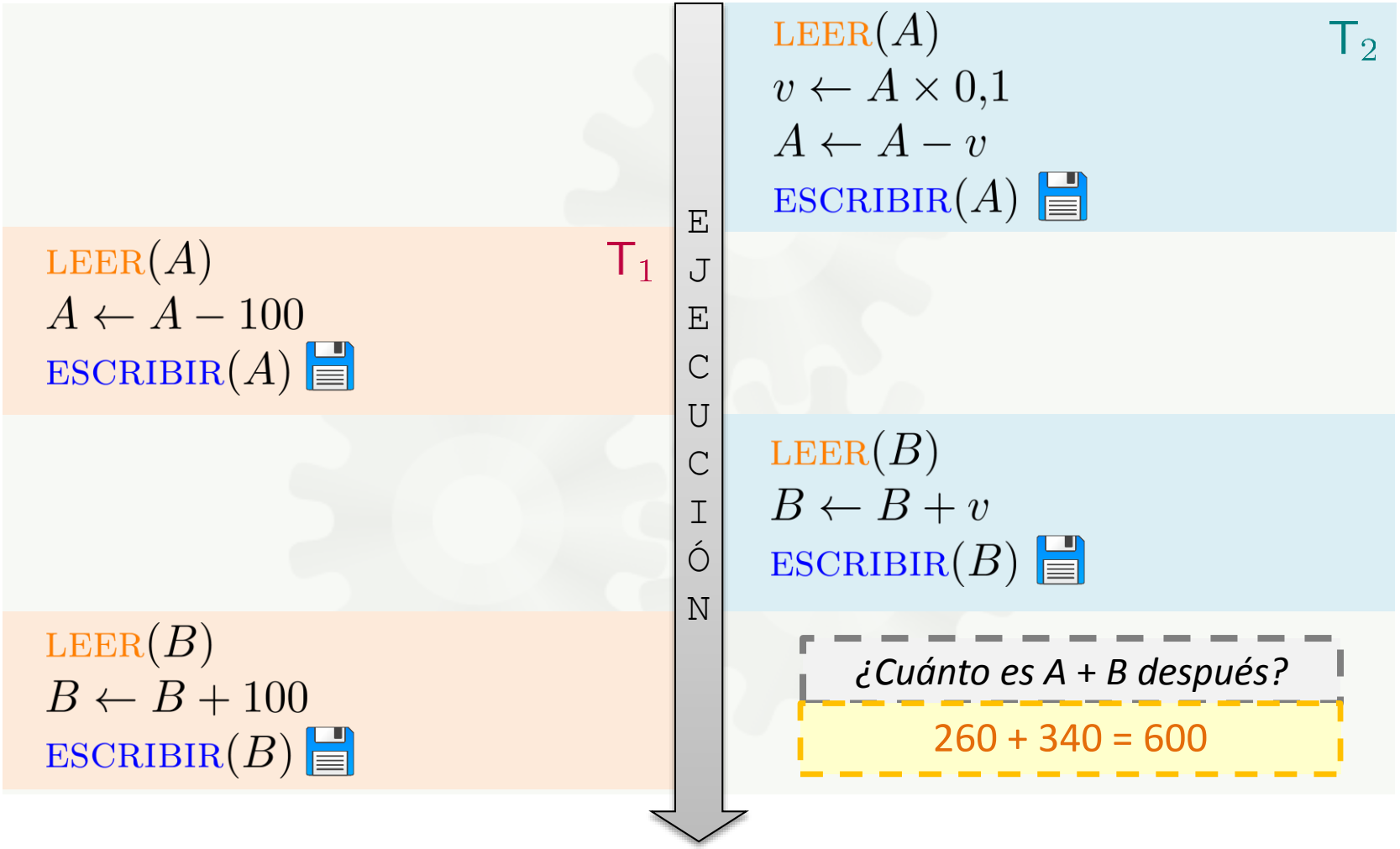
T_2

$\text{LEER}(A)$
$v \leftarrow A \times 0{,}1$
$A \leftarrow A - v$
$\text{ESCRIBIR}(A)$

$\text{LEER}(B)$
$B \leftarrow B + v$
$\text{ESCRIBIR}(B)$

¿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 + 330 = 600

EJECUCIÓN

T_1

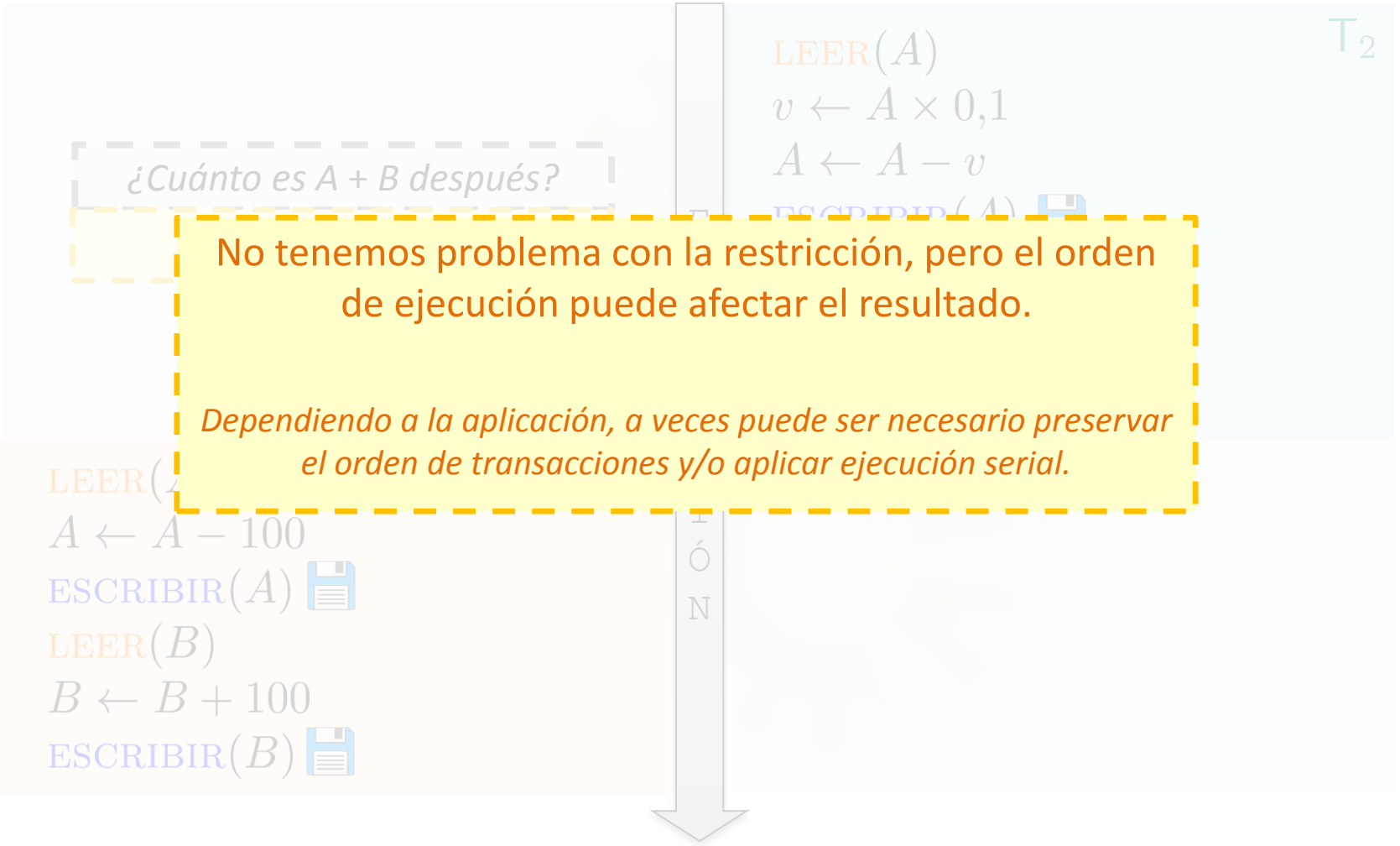
$\text{LEER}(A)$
$A \leftarrow A - 100$
$\text{ESCRIBIR}(A)$
$\text{LEER}(B)$
$B \leftarrow B + 100$
$\text{ESCRIBIR}(B)$

T_2

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

¡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)

T_1

$\text{LEER}(A)$
$A \leftarrow A - 100$

$\text{ESCRIBIR}(A)$
$\text{LEER}(B)$
$B \leftarrow B + 100$
$\text{ESCRIBIR}(B)$

¿Cuánto es A + B después?

300 + 340 = 640

EJECUCIÓN

T_2

$\text{LEER}(A)$
$v \leftarrow A \times 0{,}1$
$A \leftarrow A - v$
$\text{ESCRIBIR}(A)$

$\text{LEER}(B)$
$B \leftarrow B + v$
$\text{ESCRIBIR}(B)$

¿Dondé está el problema?

¿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)

T_1

$\text{LEER}(A)$
$A \leftarrow A - 100$

$\text{ESCRIBIR}(A)$

$\text{LEER}(B)$
$B \leftarrow B + 100$
$\text{ESCRIBIR}(B)$

EJECUCIÓN

T_2

$\text{LEER}(A)$
$v \leftarrow A \times 0{,}1$
$A \leftarrow A - v$
$\text{ESCRIBIR}(A)$

$\text{LEER}(B)$
$B \leftarrow B + v$
$\text{ESCRIBIR}(B)$

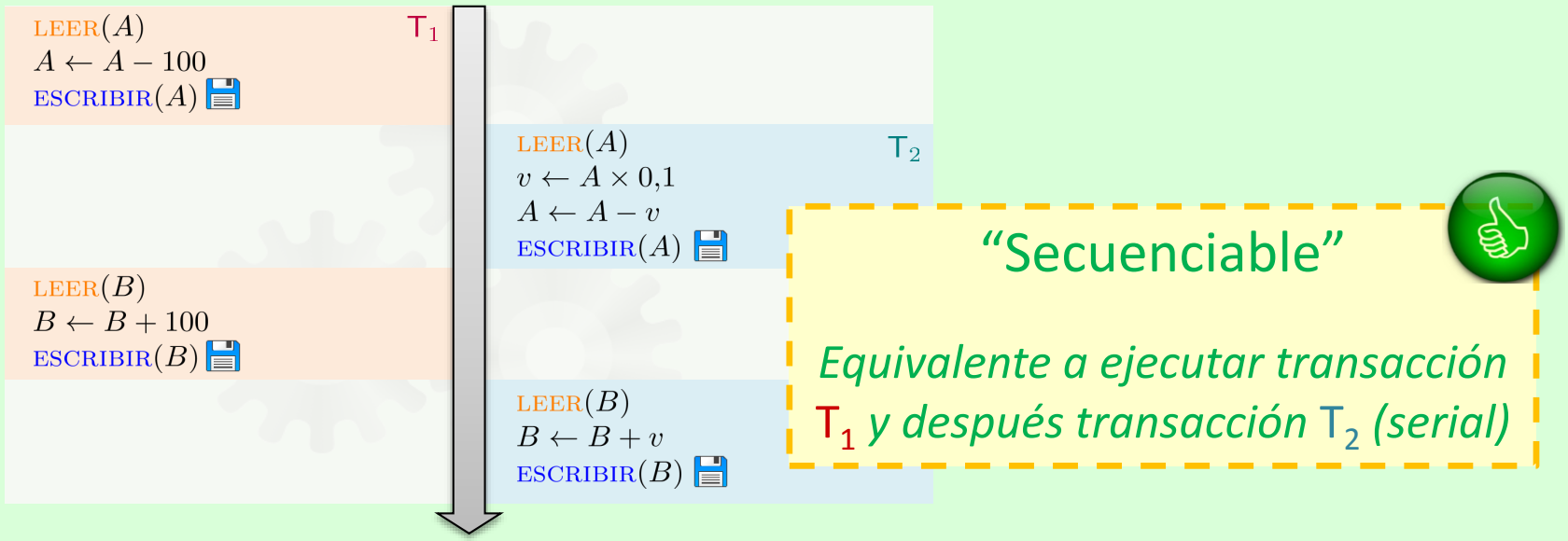
¿Cuánto es A + B después?

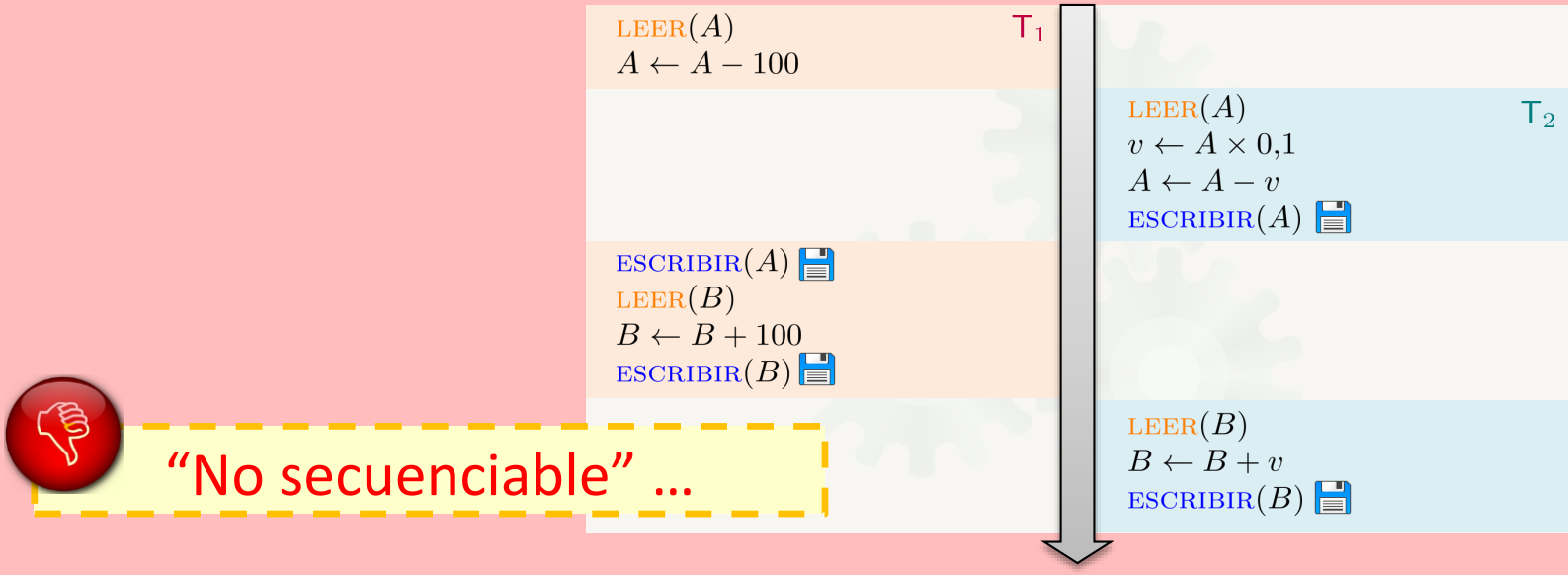
300 + 340 = 640

¿Dondé está el problema?

Planificaciones:

Secuenciables vs. No Secuenciables





Planificaciones:

Secuenciables vs. No Secuenciables

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

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

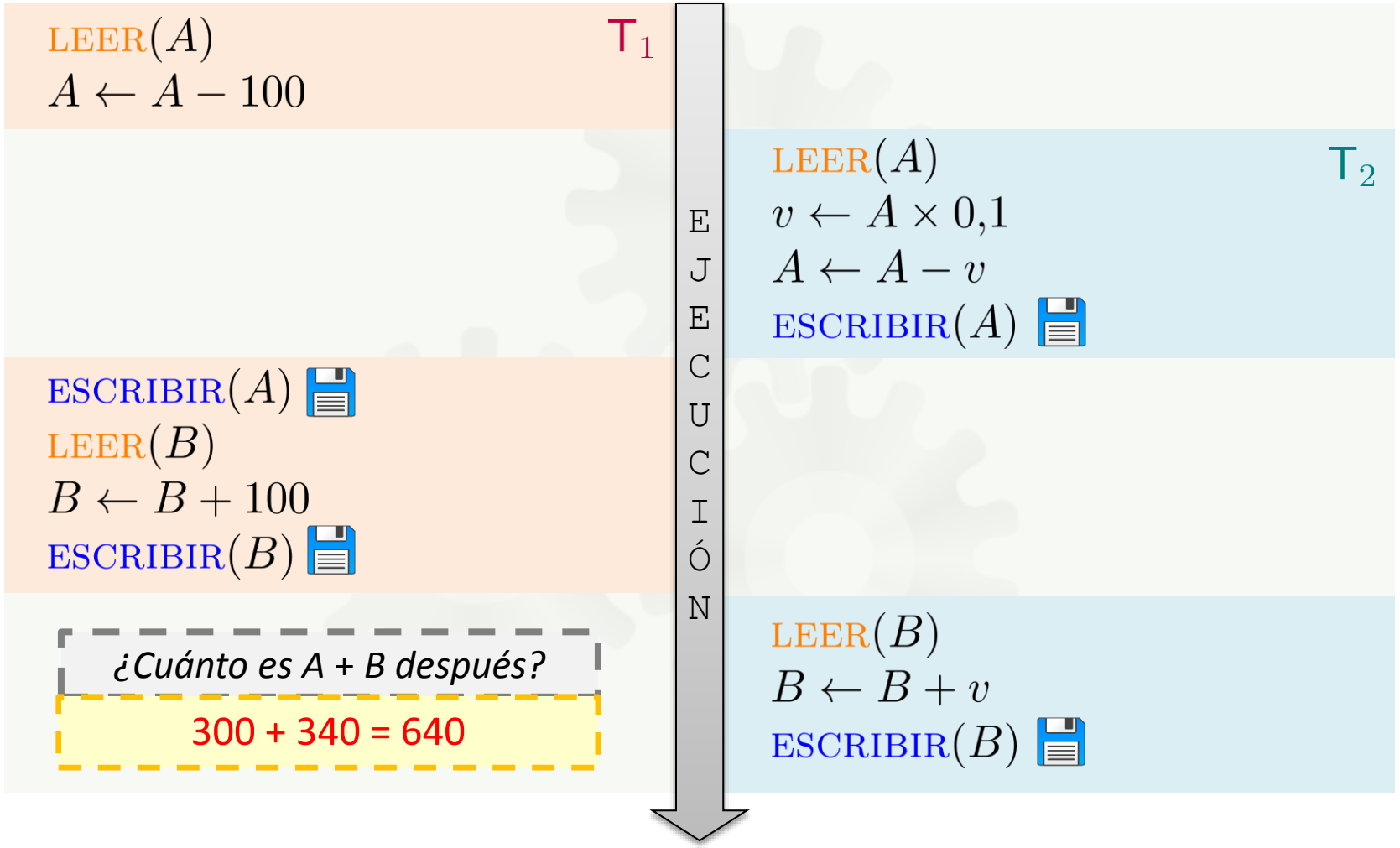
¿Cómo se puede identificar una planificación secuenciable?

(sin ejecutarla y compararla con todas las posibles planificaciones seriales)? …

Seguiremos con la pregunta inversa … cuando 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)

T_1

$\text{LEER}(A)$
$A \leftarrow A - 100$

$\text{ESCRIBIR}(A)$
$\text{LEER}(B)$
$B \leftarrow B + 100$
$\text{ESCRIBIR}(B)$

¿Cuánto es A + B después?

300 + 340 = 640

EJECUCIÓN

T_2

$\text{LEER}(A)$
$v \leftarrow A \times 0{,}1$
$A \leftarrow A - v$
$\text{ESCRIBIR}(A)$

$\text{LEER}(B)$
$B \leftarrow B + v$
$\text{ESCRIBIR}(B)$

Conflicto de Lectura–Escritura:
(Lecturas no repetibles)

T_2 cambia el valor de A que T_1 ha leído pero todavía no ha "usado"

(2) Conflicto de Escritura–Lectura

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

T_1

$\text{LEER}(A)$
$A \leftarrow A - 100$
$\text{ESCRIBIR}(A)$

¿Cuántos serían A y B si se ejecutaran T_1 y después T_2?

330 y 330

¿... T_2 y después T_1?

340 y 320

$\text{LEER}(B)$
$B \leftarrow B + 100$
$\text{ESCRIBIR}(B)$

EJECUCIÓN

T_2

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

¿Cuántos serán A y B aquí?

330 y 320

(2) Conflicto de Escritura–Lectura

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

T_1

$\text{LEER}(A)$
$A \leftarrow A - 100$
$\text{ESCRIBIR}(A)$

¿Cuántos serían A y B si se ejecutaran T_1 y después T_2?

330 y 330

¿... T_2 y después T_1?

340 y 320

$\text{LEER}(B)$
$B \leftarrow B + 100$
$\text{ESCRIBIR}(B)$

EJECUCIÓN

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

¿Cuántos serán A y B aquí?

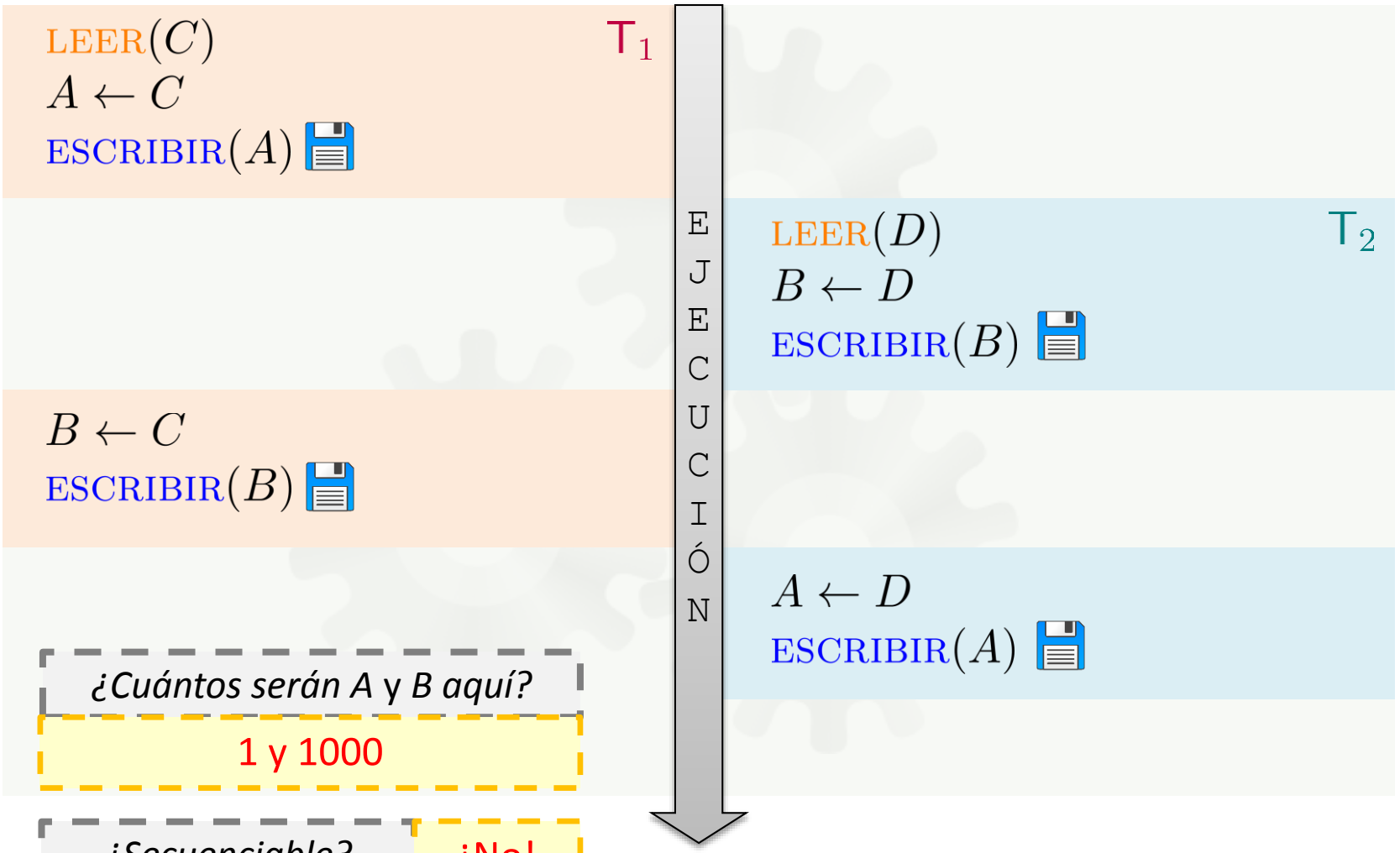
330 y 320

Conflicto de Escritura–Lectura:

El orden de cambio de A es T_1/T_2 mientras el orden de B 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)

T_1

$\text{LEER}(C)$
$A \leftarrow C$
$\text{ESCRIBIR}(A)$

T_2

$\text{LEER}(D)$
$B \leftarrow D$
$\text{ESCRIBIR}(B)$

EJECUCIÓN

$B \leftarrow C$
$\text{ESCRIBIR}(B)$

$A \leftarrow D$
$\text{ESCRIBIR}(A)$

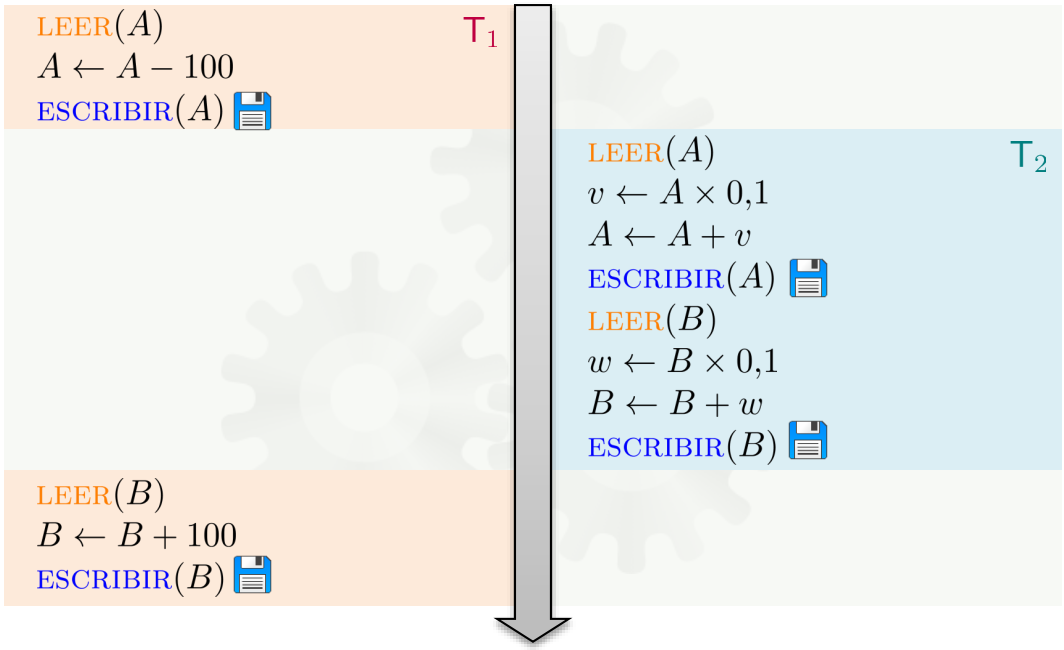
¿Cuántos serán A y B aquí?

1 y 1000

¿Secuenciable? ¡No!

Conflicto de Escritura–Escritura:

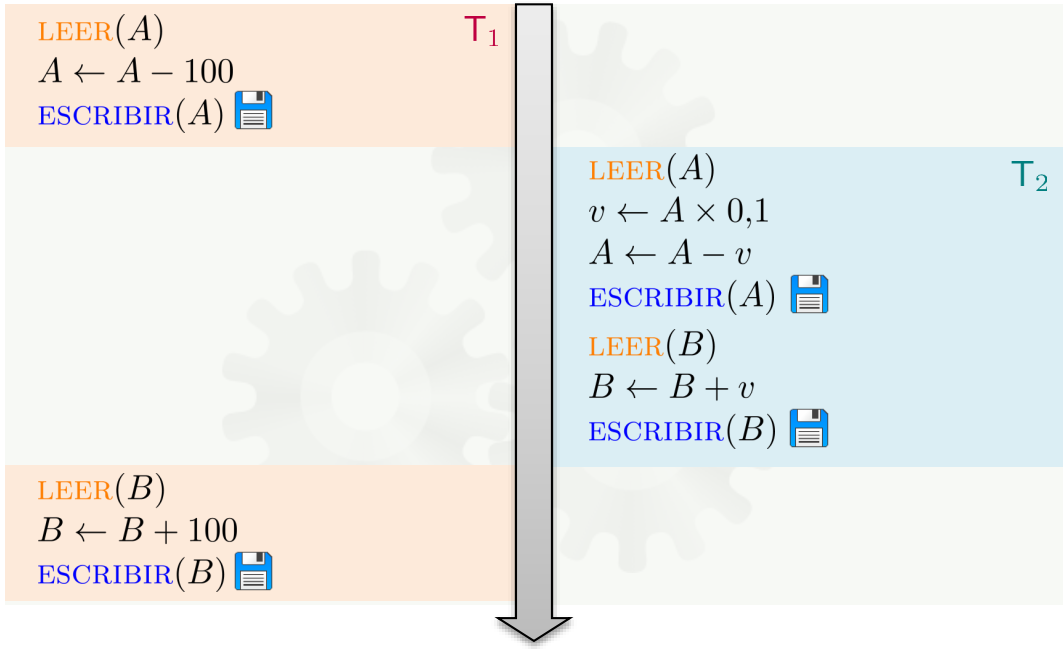
T_2 *sobrescribe* T_1 *y* T_1 *sobrescribe* T_2
(no hay problema con lecturas aquí)



¿Qué tipo de conflicto hay aquí?

Conflicto de Escritura–Lectura:

T_2 *lee el valor de A antes de que* T_1 *haya terminado sus cambios*



¿Qué tipo de conflicto hay aquí?

¡Es secuenciable!

(No hay conflicto)

T_1

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

T_2

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)

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

¿Qué tipo de conflicto hay aquí?

Conflicto de Escritura–Lectura:

T_2 lee el valor de A antes de que T_1 haya terminado sus cambios

¡No podemos <u>garantizar</u> que no haya conflictos!

(Asimismo, puede ser que no haya conflictos. Depende de lo que pasa en memoria.)

T_2

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

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)

T_1

$\text{LEER}(A)$
$A \leftarrow A - 100$
$\text{ESCRIBIR}(A)$

$\text{LEER}(B)$
$B \leftarrow B + 100$
ROLLBACK

¿Cuánto es A + B después?

270 + 230 = 500

EJECUCIÓN

T_2

$\text{LEER}(A)$
$v \leftarrow A \times 0{,}1$
$A \leftarrow A - v$
$\text{ESCRIBIR}(A)$

$\text{LEER}(B)$
$B \leftarrow B + v$
$\text{ESCRIBIR}(B)$

(*) 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)

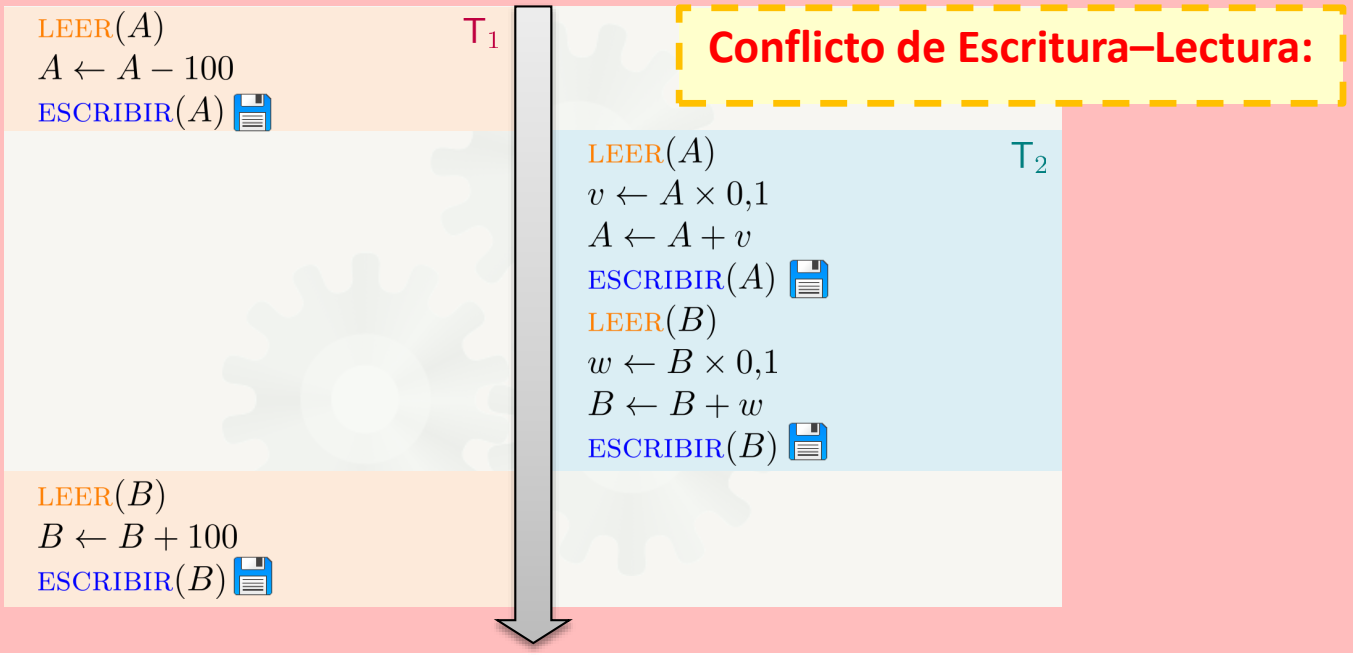
- En una **planificación recuperable**, cada transacción T se compromete (COMMIT) sólo después de que se comprometan a su vez todas las transacciones desde las cuáles T hayan leído algo.

- Si cada transacción T solamente lee cambios de transacciones comprometidas, **se puede 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 casada) 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



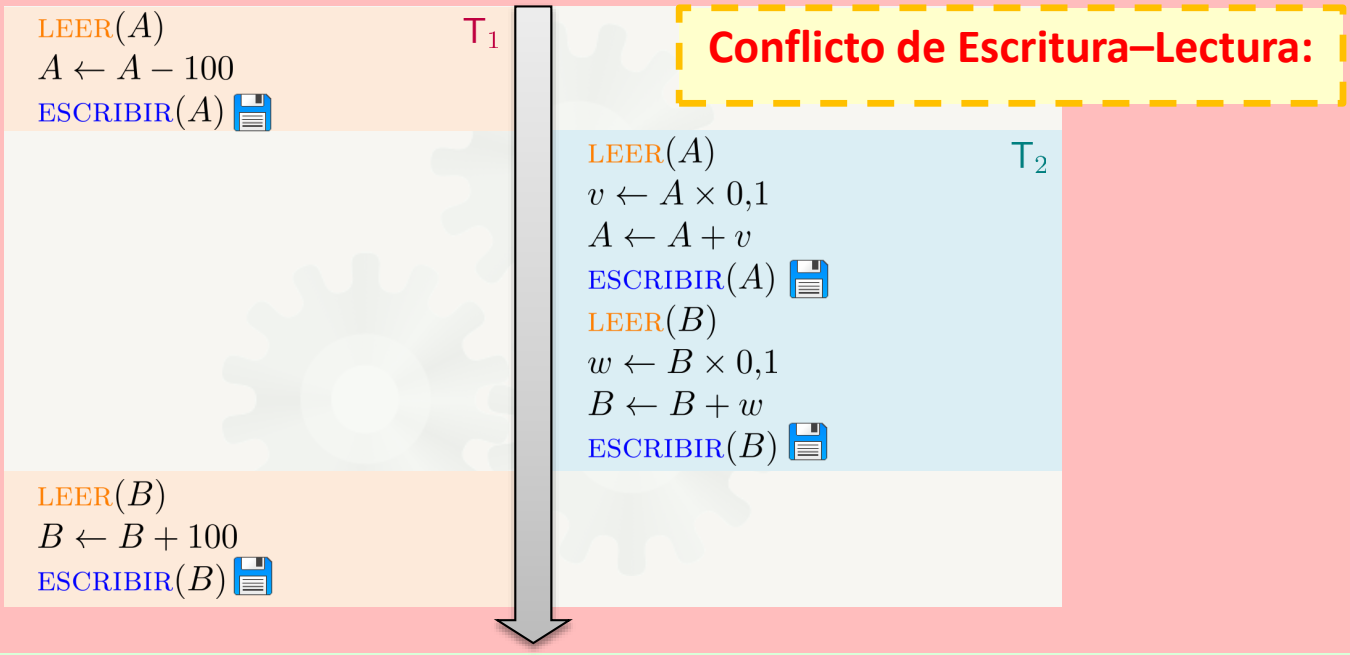
¿Han visto un problema así antes de este curso?

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){
  sychronized(this.tablaCuentas) {
    long aAntiguo = tablaCuentas.leer(A);
    long aNuevo = aAntiguo - monto;
    tablaCuentas.escribir(aNuevo);
    long bAntiguo = tablaCuentas.leer(B);
    long bNuevo = bAntiguo + monto;
    tablaCuentas.escribir(bNuevo);
  }
}
```

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

Una solución en Java

¿Cómo podrías mejorar el rendimiento del código abajo?

Minimizar el código sincronizado tanto como sea posible

Usar objetos de bloqueos tan especifico 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)

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

```
public void interés(Cuenta[] CS, float val){
  sychronized(this.tablaCuentas) {
    for(Cuenta C:Cs) {
      long cAntiguo = tablaCuentas.leer(C);
      long cNuevo = cAntiguo+=Math.round(monto*val);
      tablaCuentas.escribir(cNuevo);
    }
  }
}
```

¿Una solución en un sistema de b.d.d.?

¿Cómo podrías mejorar el rendimiento del código abajo?

Minimizar el código sincronizado tanto como sea posible

Usar objetos de bloqueos tan especifico 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)

¡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.

Un protocolo: Bloqueo en 2 fases estricto

- Si una transacción T quiere leer un objeto O, tiene que conseguir un bloqueo compartido sobre O
  - Varias transacciones pueden leer el mismo objeto al mismo tiempo
- Si una transacción T quiere modificar un objeto O, tiene que conseguir un bloqueo exclusivo sobre O
  - Un bloqueo exclusivo excluye bloques compartidos
  - T puede leer O (por supuesto)
  - Así nadie (aparte de T) puede ni leer ni modificar O mientras T tenga su bloqueo exclusivo sobre O
- Cuando T haya terminado con el bloqueo, la libera

Bloqueo en dos fases estricto

- Si una transacción T quiere leer un objeto O, tiene que conseguir un bloqueo compartido sobre O
  - Varias transacciones pueden leer el mismo objeto al mismo tiempo
- Si una transacción T quiere modificar un objeto O, tiene que conseguir un bloqueo exclusivo sobre O
  - Un bloqueo exclusivo excluye bloques compartidos
  - T puede leer O (por supuesto)
  - Así nadie (aparte de T) puede ni leer ni modificar O mientras T tenga su bloqueo exclusivo sobre O
- Cuándo T ha terminado con el bloqueo, la suelta

La forma más popular de garantizar secuenciabilidad.

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)

T_1

$\text{LEER}(A)$
$A \leftarrow A - 100$

$\text{ESCRIBIR}(A)$
$\text{LEER}(B)$
$B \leftarrow B + 100$
$\text{ESCRIBIR}(B)$

¿Cuánto es A + B después?

300 + 340 = 640

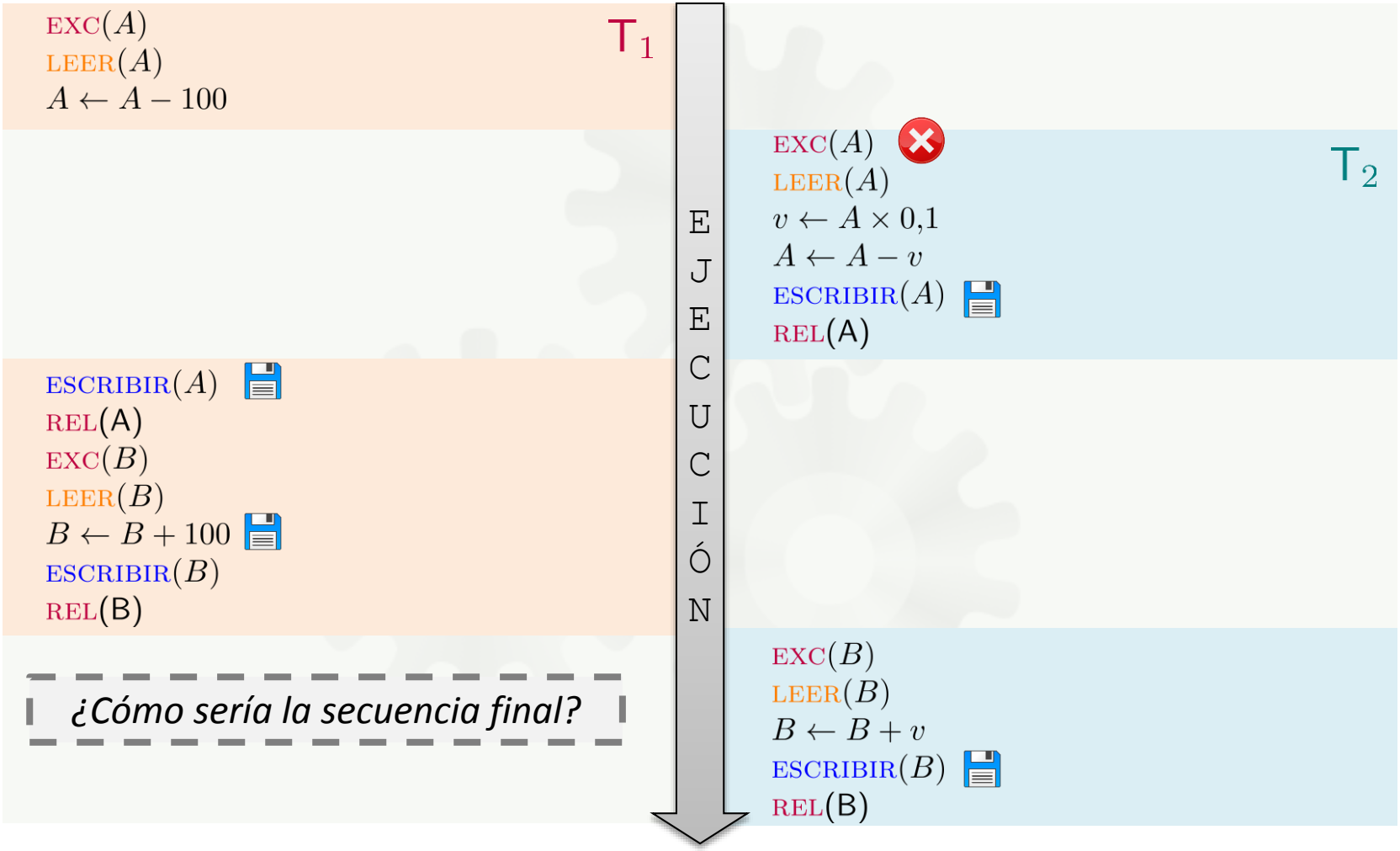
EJECUCIÓN

T_2

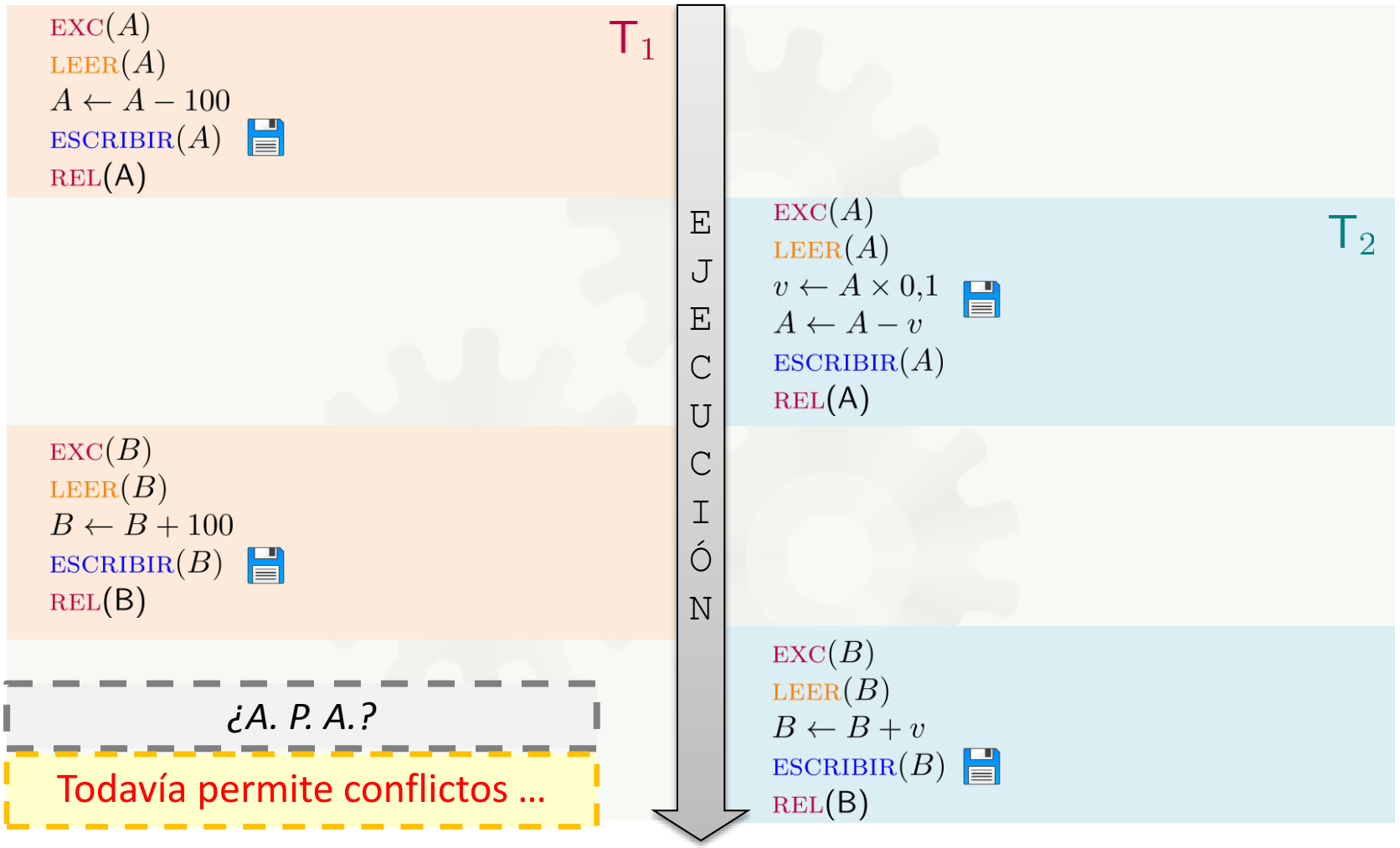
$\text{LEER}(A)$
$v \leftarrow A \times 0{,}1$
$A \leftarrow A - v$
$\text{ESCRIBIR}(A)$

$\text{LEER}(B)$
$B \leftarrow B + v$
$\text{ESCRIBIR}(B)$

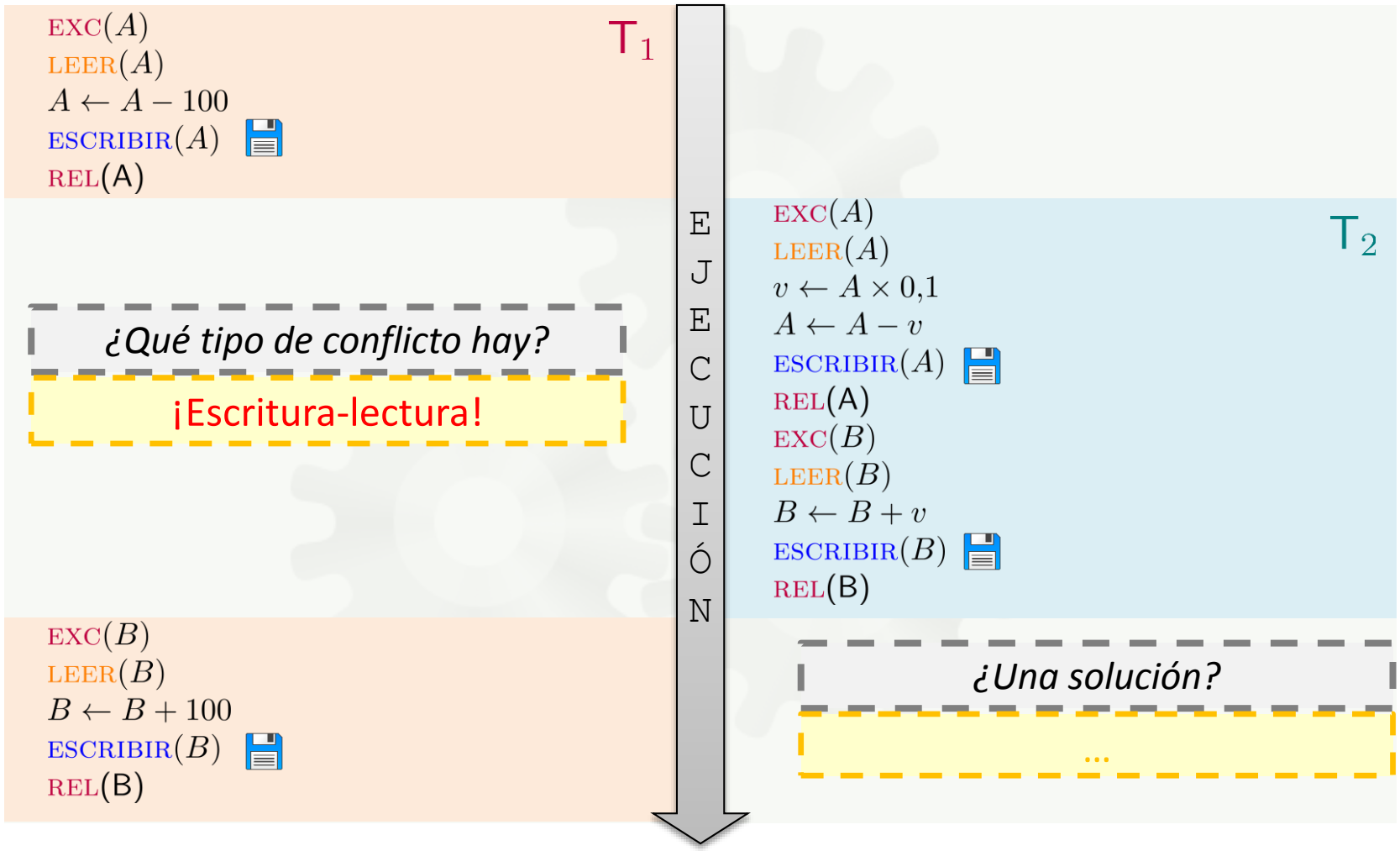
… con bloqueos



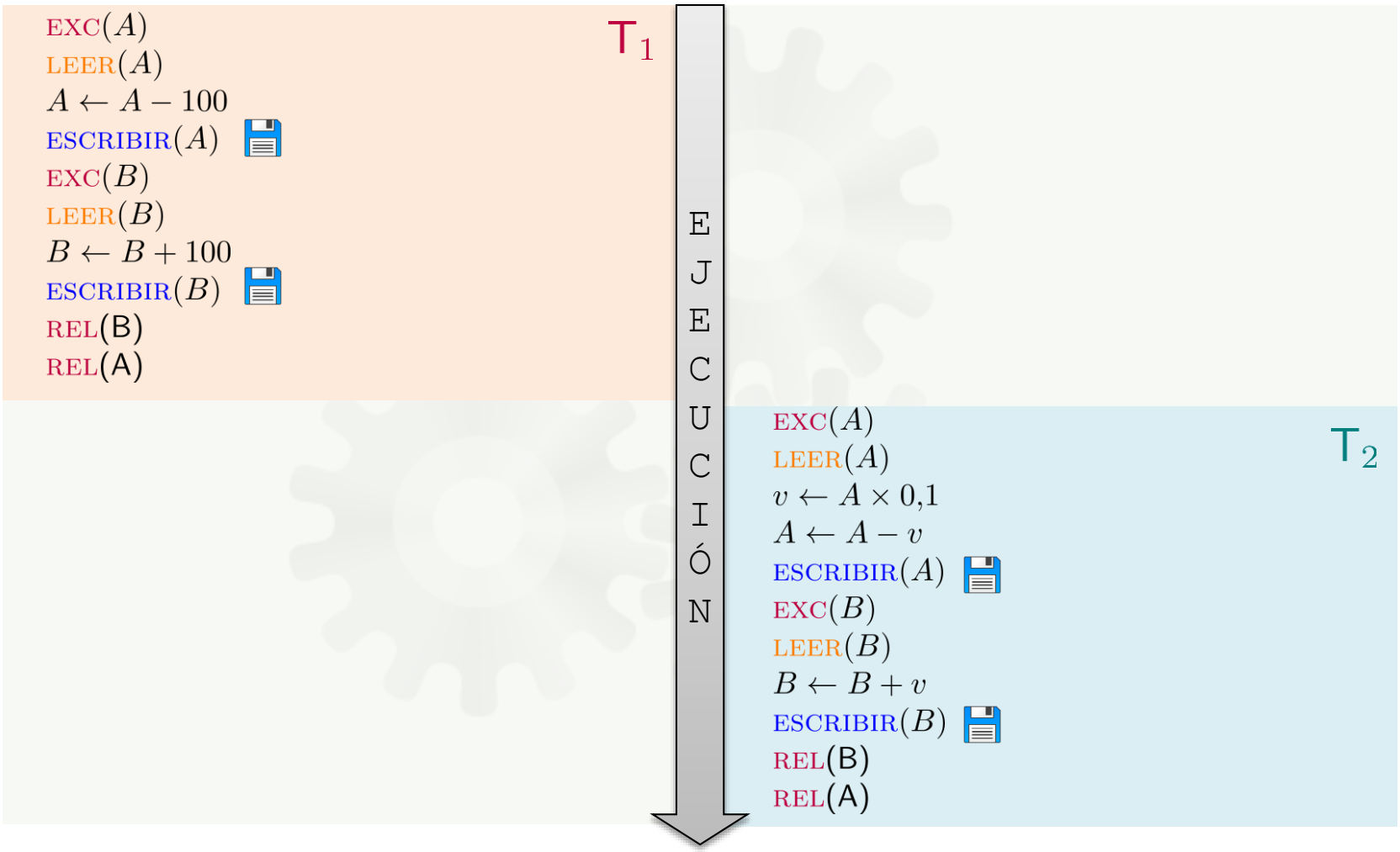
… con bloqueos



… conflicto


EXC(A)
LEER(A)
A ← A − 100
ESCRIBIR(A)
REL(A)
T1
EXC(A)
LEER(A)
v ← A × 0,1
A ← A − v
ESCRIBIR(A)
REL(A)
EXC(B)
LEER(B)
B ← B + v
ESCRIBIR(B)
REL(B)
T2
¿Qué tipo de conflicto hay?
¡Escritura-lectura!
EXC(B)
LEER(B)
B ← B + 100
ESCRIBIR(B)
REL(B)
EJECUCIÓN
¿Una solución?
…

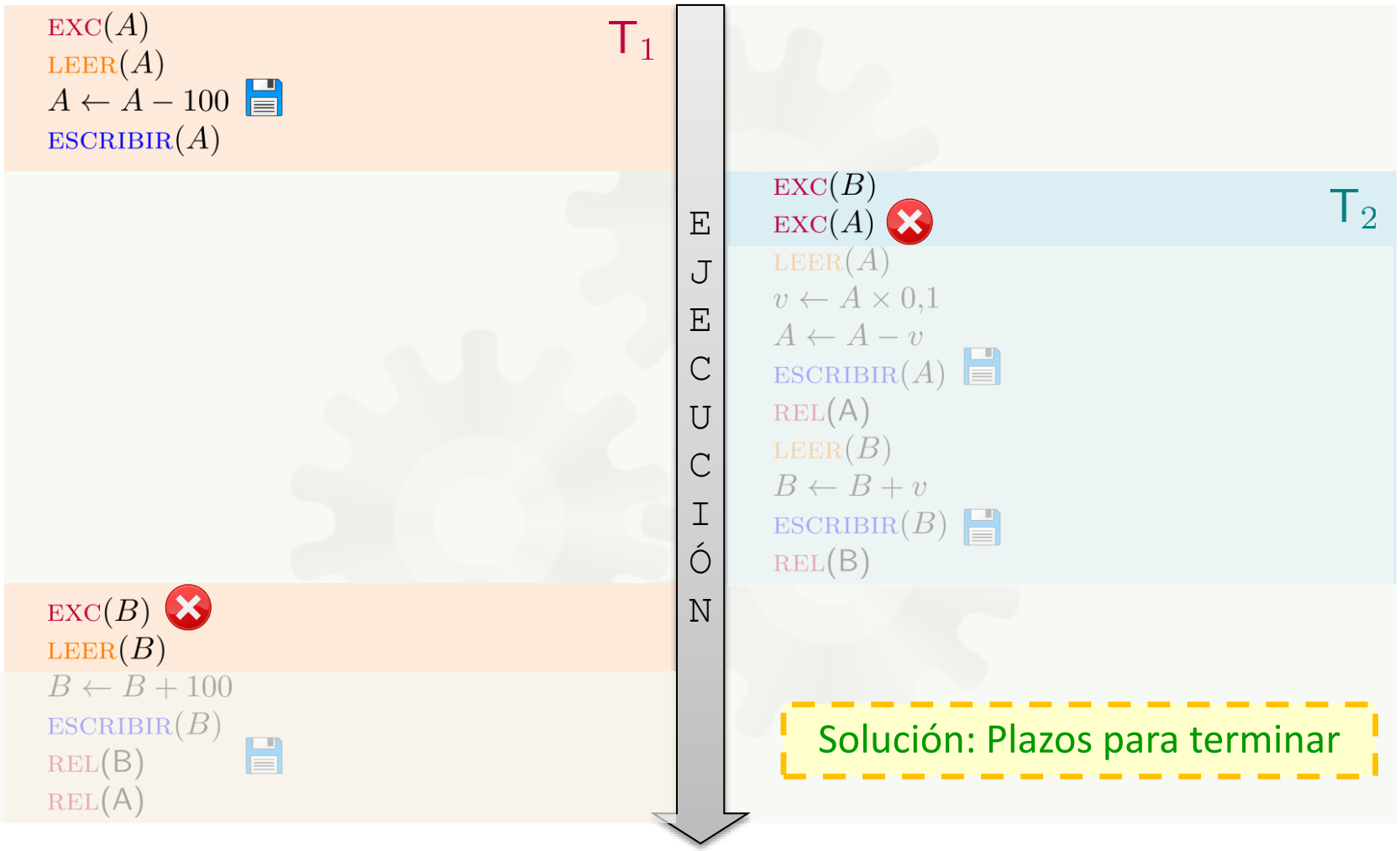
… lo más seguro (en este caso)



… pero cuidado!

¿A. P. A.?

Hay que evitar interbloqueos (*deadlocks*)



Solución: Plazos para terminar


2010
nd Minds

Hemos terminado con bases de datos relacionales

Preguntas?