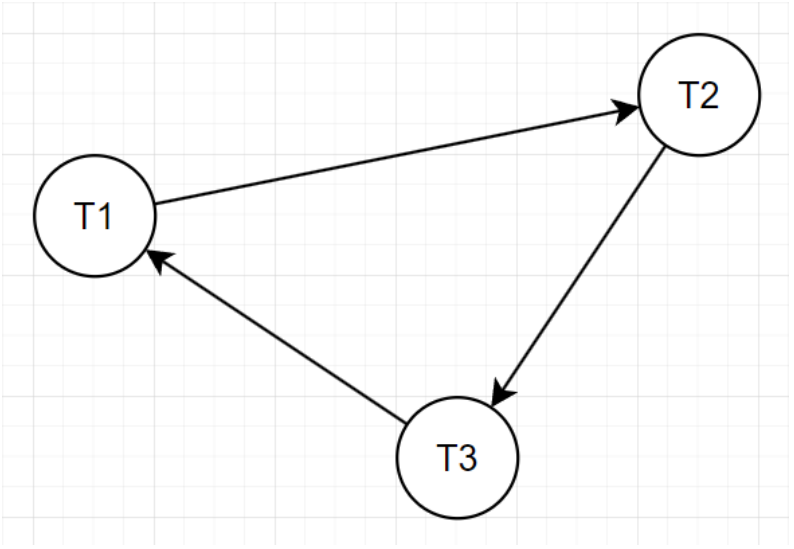
Parcialito de Concurrencia y Recuperación 2C24

Base de Datos 75.15/75.28/95.05/TA044 - Facultad de Ingeniería - UBA

nombre: Marcos Bianchi Fernández - **padrón**: 108921 - **fecha:** 25/10

1. Para las siguientes planificaciones:
   - Dibujar los grafos de precedencia
   - Listar los conflictos
   - Determinar cuales son serializables

   a) bT1; bT2; bT3; RT1(A); **WT1(A); RT2(A)**; WT2(A); RT3(B); **WT3(B); RT1(B)**; WT1(B); RT2(C); **WT2(C); RT3(C)**; WT3(C); cT1; cT2; cT3;



No es serializable

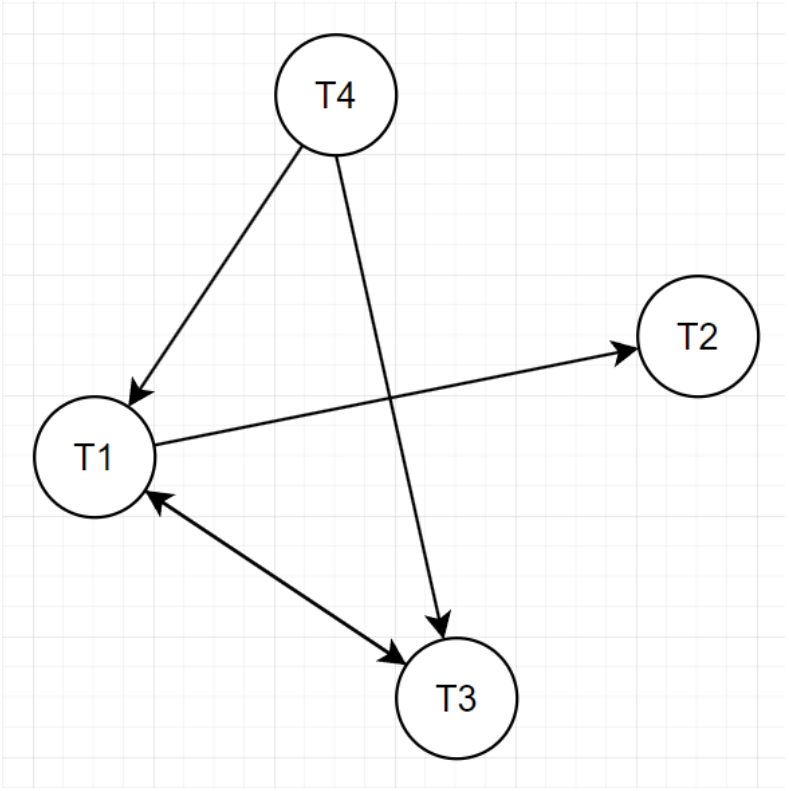
"WT1(A); RT2(A)": T1 → T2

"WT3(B); RT1(B)": T3 → T1

"WT2(C); RT3(C)": T2 → T3

Los demás conflictos no muestran ninguna precedencia nueva.

b) bT1; bT2; bT3; bT4; RT1(X);
WT1(X); RT2(X); WT2(X); RT3(Y); WT3(Y); RT4(Z); **WT4(Z); RT1(Z); RT3(Z)**; **WT1(Z); WT3(Z)**;
cT1; cT2; cT3; cT4;



No es serializable

"WT1(X); RT2(X)": T1 → T2

"WT4(Z); RT1(Z)": T4 → T1

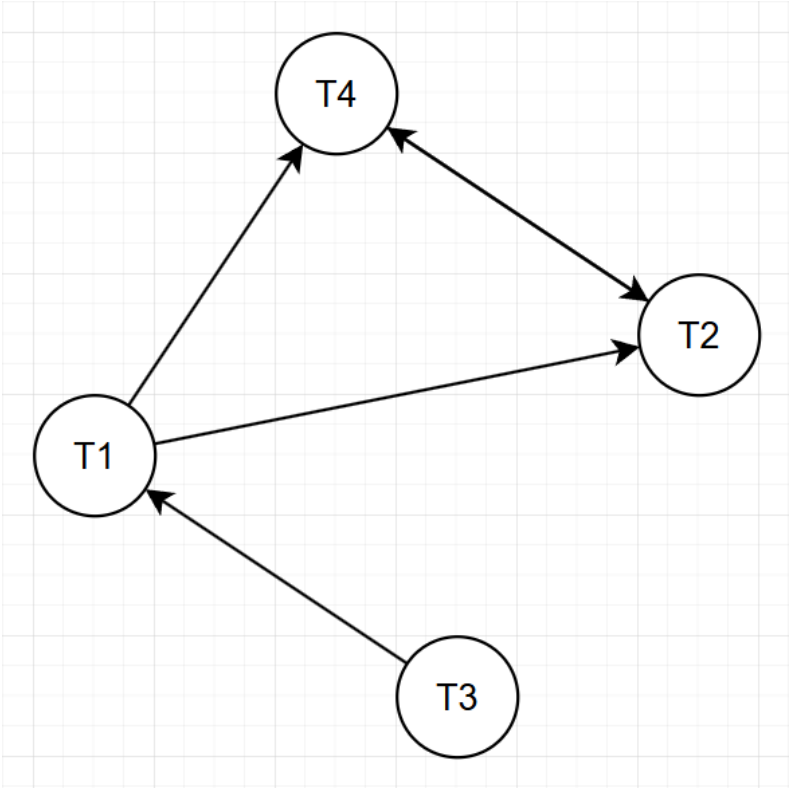
"WT4(Z); …; RT3(Z)": T4 → T3

"RT3(Z); WT1(Z)": T3 → T1

"WT1(Z); WT3(Z)": T1 → T3

Los demás conflictos no muestran una precedencia nueva.

c) bT1; bT2; bT3; bT4; RT1(A);
WT1(A); RT2(B); **WT2(B)**; RT3(C); **WT3(C)**; **RT4(A)**; **WT4(A)**; **RT2(A)**; WT2(A); **RT4(B)**; WT4(B);
RT1(C); WT1(C); cT1; cT2; cT3; cT4;



No es Serializable

"WT1(A); …; RT4(A)": T1 → T4

"WT1(A); …; RT2(A)": T1 → T2

"WT2(B); …; RT4(B)": T2 → T4

"WT3(C); …; RT1(C)": T3 → T1

"WT4(A); RT2(A)": T4 → T2

Los demás conflictos no muestran una precedencia nueva.

2. Dado el siguiente conjunto de transacciones:
   T1 : bT1 ; RT1(D); WT1(D); RT1(B); WT1(B); cT1
   T2 : bT2 ; RT2(A); WT2(A); RT2(D); WT2(D); cT2
   T3 : bT3 ; RT3(A); WT3(A); RT3(B); WT3(B); cT3
   Se pide:

   a. Coloque locks y unlocks a las transacciones de manera de respetar el Protocolo de Lock de 2 Fases, intentando a la vez minimizar el tiempo que las transacciones mantienen los locks sobre los recursos.

   bT1; bT2; bT3; **L(T1, D)**; **L(T2, A)**; RT1(D); RT2(A); WT1(D); WT2(A); **L(T1, B)**; **U(T1, D)**; **L(T2, D)**; **U(T2, A)**; **L(T3, A)**; RT3(A); WT3(A); RT1(B); WT1(B); **U(T1, B)**; **L(T3, B)**; **U(T3, A)**; RT2(D); WT2(D); **U(T2, D)**; RT3(B); WT3(B); **U(T3, B**); cT1; cT2; cT3

   b. Defina qué es recuperable. Indique si el solapamiento es recuperable, justificando su respuesta.

Un solapamiento es recuperable sii ninguna transacción T0 commitea previo a que todas las transacciones que escribieron datos antes de que T0 los leyera hayan commiteado.

Este solapamiento **es recuperable**, pues D es editado primero por T1, por lo que debe terminar antes que T2, A es editado primero por T2, por lo que debe terminar antes que T3 y B es editado primero por T1, por lo que debe terminar antes que T3. Entonces el orden de commits ascendente cumple con ser recuperable.

Los commits pueden suceder antes, creo que dejándolos al final lo hace un poco más legible.

3. Supongamos el siguiente log de un sistema que usa undo/redo logging. ¿Cuál es el valor de los ítems A, B, C, D, E, F, y G en disco después de la recuperación si la falla se produce en las siguientes situaciones:

Cree una tabla poniendo el valor de cada ítem para cada línea pedida.

```
01. <START T1>
 02. <T1, A, 10, 50>
 03. <START T2>
 04. <T1, B, 20, 60>
 05. <T1, A, 50, 70>
 06. <T2, C, 15, 25>
 07. <START CKPT(T1,T2)>
 08. <T1, B, 60, 80>
 09. <T2, D, 30, 40>
 10. <COMMIT T1>
 11. <T2, C, 25, 35>
 12. <START T3>
 13. <T3, E, 70, 90>
 14. <T2, D, 40, 60>
 15. <COMMIT T2>
 16. <T3, F, 50, 100>
 17. (ignorar)
 18. <T3, G, 80, 120>
 19. <START T4>
 20. <T4, E, 90, 110>
 21. <END CKPT>
 22. <T4, G, 120, 140>
 23. <COMMIT T3>
 24. <COMMIT T4>
```

a. Justo antes de la línea 18.

(REDO) Commiteados: T1, T2

(UNDO) No commiteados: T3

Arrancamos desde abajo, deshacemos todas las operaciones que encontremos de T3. Así hasta llegar al principio del log, ahora de arriba para abajo rehacemos todos los cambios de T1 y T2. Escribimos el ABORT de T3 y flusheamos el log.

A	B	C	D	E	F	G
70	80	35	60	70	50	80

b. Justo antes de la línea 22.

(REDO) Commiteados: T1, T2

(UNDO) No commiteados: T3, T4

Arrancamos desde abajo y vemos un END CKPT, mientras tanto deshacemos los cambios de T3 y T4, buscamos el próximo START CKPT y llegamos a la línea 7, su transacción más antigua es T1, por lo que vamos hasta la línea 1 y a partir de ahí bajamos rehaciendo T1 y T2. Al terminar escribimos el ABORT PARA T3 y T4 y flusheamos el log a disco.

A	B	C	D	E	F	G
70	80	35	60	70	50	80

c. Después de la línea 22.

Acá depende en que línea ocurra la falla, si es justo después de la 22 entonces el resultado es igual al anterior con la salvedad de que se deshace el cambio hecho por T4 sobre G, se comienzan a rehacer los cambios desde la línea 1. Se escribe ABORT de T3 y T4 y se flushea el log.

A	B	C	D	E	F	G
70	80	35	60	70	50	80

Si ocurre justo después de la línea 23 entonces los cambios hechos por T3 no se deshacen, sino que se rehacen, T4 se deshace y se arranca en la línea 2. Se escribe el ABORT de T4 y se flushea el log.

A	B	C	D	E	F	G
70	80	35	60	90	100	120

Si la falla ocurre al final del log, no hay cambios por deshacer, se rehacen los cambios de todas las transacciones desde la línea 1. No hay ABORTs, por lo que no hace falta flushear el log a disco.

A	B	C	D	E	F	G
70	80	35	60	110	100	140