# QUESTION 1:

- The schedule is **serializable** as it is equivalent of the serial schedule T1 => T2

- The schedule is **conflict-serializable** because any serializable schedules are by definition conflict-serializable

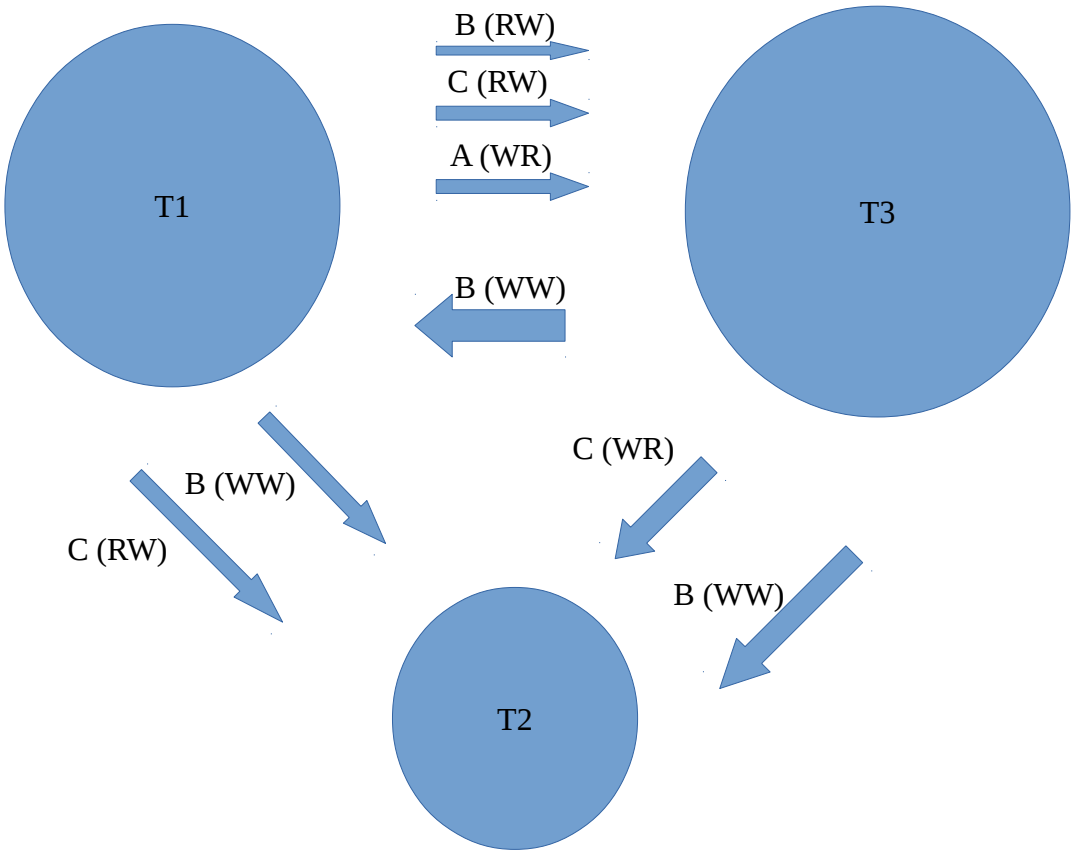
- The schedule is **recoverable** as no xact read value from another xact (T1 read the initial value A). All the Xact can then commit whenever.

- The schedule is **cascadeless** for the same reason as before. The only value to be read is the initial value of A

- The schedule is **not strict** as T2 writes A before T1 writes it.

# QUESTION 2:

(1) ***Precedence graph :***



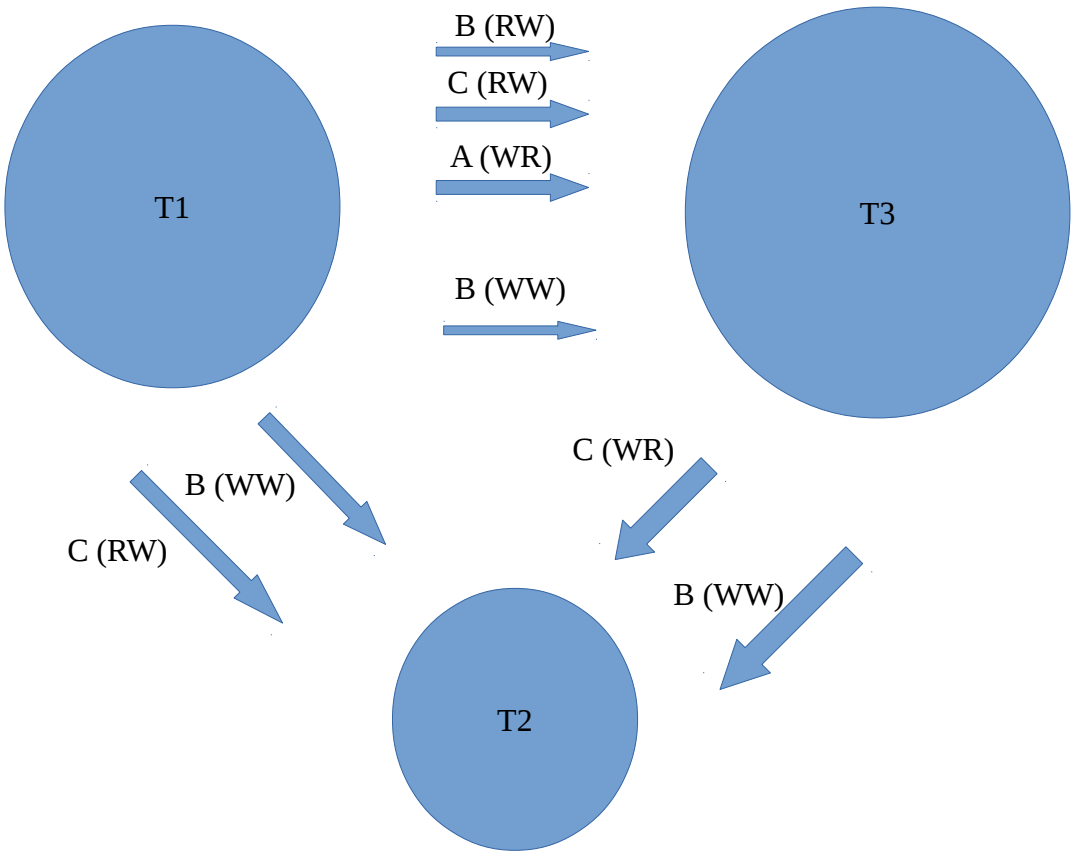
(2) **conflict-serializable? *:***

The schedule is not **conflict-serializable** because we can see from the precedence graph that ther is a cycle between T1 and T3.

(3) **reads from? *:***

T3 → T2 (actions 6 and 7)
T1 → T3 (actions 1 and 10)

(4) **New Schedule *:***




The precedence graph now has no cycle → The schedule is **confict serializable.**

To respect the conditions in question 3 we can see that the schedule can be serialize by first doing T1, then we do T3 and after T2 (the precedence graph gives the possibility to do that).

**<u>Serial schedule</u>** = T1 → T3 → T2

# QUESTION 3:

(1) ***Precedence graph :***

| Time Stamp | Action: |
|---|---|
| 1 | T1: S(A) |
| 2 | T1: R(A) |
| 3 | T2: X(A) |
| 4 | T2: blocked (If wait-die policy then T2 waits because he has a higher priority) |
| 5 | T3: X(B) |
| 6 | T3: W(B) |
| 7 | T1: X(B) |
| 8 | T1: Blocked (we assume it has a higher priority and wait-die policy) |
| 9 | T3: Commit |
| 10 | T3: Free locks (on B) |
| 11 | T1: X(B) |
| 12 | T1: W(B) |
| 13 | T1: Commit |
| 14 | T1: Free locks (on A and B) |
| 15 | T2: X(A) |
| 16 | T2: W(A) |
| 17 | T2: X(B) |
| 18 | T2: W(B) |
| 19 | T2: Commit |
| 20 | T2: Free locks (on A and B) |

# Question 4

***Question 1:***

In slide 12 we have the notation below (with the control info)

| LSN | PrevLSN | XID | Type | PageID | Length | Offset | Before-img | After-img |
|---|---|---|---|---|---|---|---|---|
| 0 | NULL | T3 | Start | 0 | | | - | - |
| 1 | 0 | T3 | Read | 0 | | | Tax=2 | tax=2 |
| (2 | 1 | T3 | ADD | 0 | | | - | 3) |
| 3 | Null | T1 | Start | 0 | | | - | - |
| 4 | 3 | T1 | Read | 0 | | | Salary=1 | Salary=1 |
| (5 | 4 | T1 | ADD | 0 | | | - | 2) |
| | | | | | | | | |
| 6 | 2 | T3 | Write | 0 | | | Tax=2 | Tax=3 |
| 7 | 6 | T3 | Commit | 0 | | | | |
| 8 | NULL | T2 | Start | 0 | | | - | - |
| 9 | 8 | T2 | Read | 0 | | | Tax=3 | Tax=3 |
| 11 | 9 | T2 | Read | 0 | | | Salary=1 | Salary=1 |
| 12 | 10 | T2 | Add | 0 | | | - | 5 |
| 13 | 11 | T2 | Write | 0 | | | Tax = 3 | Tax=4 |
| 14 | 12 | T2 | Commit | | | | | |
| 15 | | | | | | | | |
| 16 | 5 | T1 | Read | 0 | | | Tax=4 | Tax=4 |
| (17 | 16 | T1 | Add | 0 | | | 4+2 | 6) |
| 18 | 17 | T1 | Write | 0 | | | Tax=4 | Tax=6 |
| 19 | 18 | T1 | Write | 0 | | | Salary=1 | Salary=2 |
| | | | | | | | | |

***Question 2:***

The Transaction T1 has to be undone because it has failed

The Transaction T1 has to be redone (smallest recLSN)

# Question 5

***Question 1:***

undonextLSN is the PrevLSN of the record we are undoing

| LSN | PrevLSN | undonextLSN |
|---|---|---|
| 00 | - | - |
| 10 | 00 | - (T1 is not aborted) |
| 20 | - | - |
| 30 | - | - (T3 doesnt need to be undone) |
| 40 | 30 | - |
| 50 | 20 | 20 |
| 60 | 50 | 50 |
| 70 | 60 | - |

***Question 2:***

The actions taken are:
- write an Abort log record
- CLR: Undo T2 LSN 60 : undo update of p5
- CLR: Undo T2 LSN 50 : undo update of p5
- CLR: Undo T2 LSN 20 : undo update of p5
- T2 ends

***Question 3:***
The idea is to repeat all actions before the crash, then undo the xacts still active at crash time (T1)

| LSN | Type | Xact ID | Page ID | PrevLSN | undonextLSN |
|---|---|---|---|---|---|
| 80 | Abort | T1 | 0 | - | 10 |
| 90 | Undo | T1 | 0 | 00 | 00 |
| 100 | Undo | T1 | 0 | - | - |
| 110 | End | T1 | 0 | - | - |
| 120 | | | | | |
| 130 | | | | | |
| 140 | | | | | |