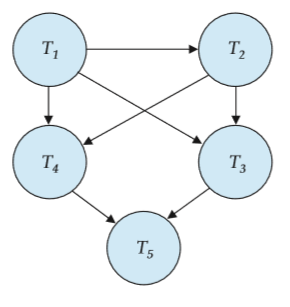
**14.4 Justify the following statement: Concurrent execution of transactions is more important when data must be fetched from (slow) disk or when transactions are long, and is less important when data are in memory and transactions are very short.**

If a transaction is very long or when it fetches data from a slow disk, it takes a long time to complete. In absence of concurrency, other transactions will have to wait for longer period of time. Average response time will increase. Also when the transaction is reading data from disk, CPU is idle. So resources are not properly utilized. Hence concurrent execution becomes important in this case. However, when the transactions are short or the data is available in memory, these problems do not occur.

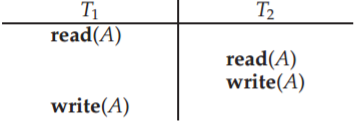
**14.6 Consider the precedence graph of Figure 14.16. Is the corresponding schedule conﬂict serializable? Explain your answer.**

There is a serializable schedule corresponding to the precedence graph below, since the graph is acyclic. A possible schedule is obtained by doing a topological sort, that is, T1, T2, T3, T4, T5.

**14.8 The lost update anomaly is said to occur if a transaction Tj reads a data item, then another transaction Tk writes the data item(possibly based on a previous read), after which Tj writes the data item. The update performed by Tk has been lost, since the updated one by Tj ignored the value written by Tk.**

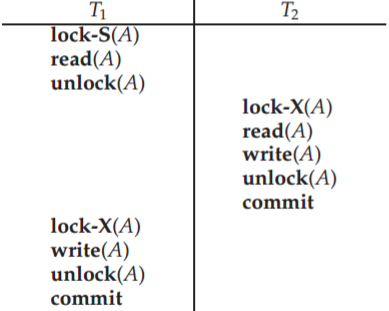


**a.Give an example of a schedule showing the lost update anomaly.**



In the above schedule, the value written by the transaction T2 is lost because of the write of the transaction T1.

**b.Give an example schedule to show that the lost update anomaly is possible with the read committed isolation level.**



The locking in the above schedule ensures the Read Committed isolation level. The value written by transaction T2 is lost due to T1’s write.

**c. Explain why the lost update anomaly is not possible with the repeatable read isolation level.**

Lost Update Anomaly is not possible in Repeatable Read isolation level. In repeatable read isolation level, a transaction T1 reading a data item X, holds a shared lock on X till the end. This makes it impossible for a newer transaction T2 to write the value of X (which requires X-lock) until T1 finishes. This forces the serialization order T1, T2 and thus the value written by T2 is not lost.

**14.14 Explain the distinction between the terms serial schedule and serializable schedule.**

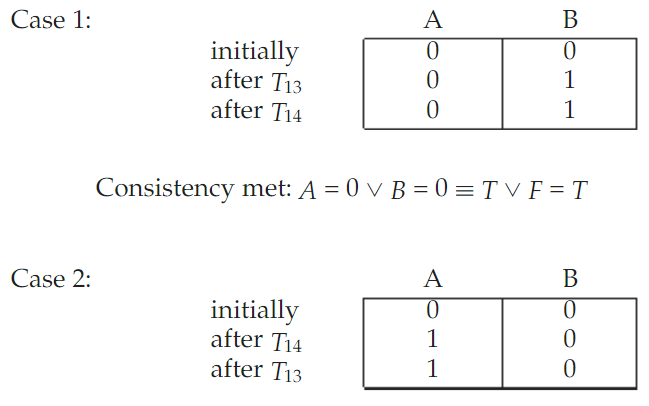
A schedule in which all the instructions belonging to one single transaction appear together is called a serial schedule. A serializable schedule has a weaker restriction that it should be equivalent to some serial schedule. There are two deﬁnitions of schedule equivalence–conﬂict equivalence and view equivalence. Both of the se are described in the chapter.

**14.15 Consider the following two transactions:**

**T13: read(A); read(B); if A = 0 then B := B + 1; write(B). T14: read(B); read(A); if B = 0 then A := A + 1; write(A). Let the consistency requirement be A = 0 ∨ B = 0, with A = B = 0 the initial values.**

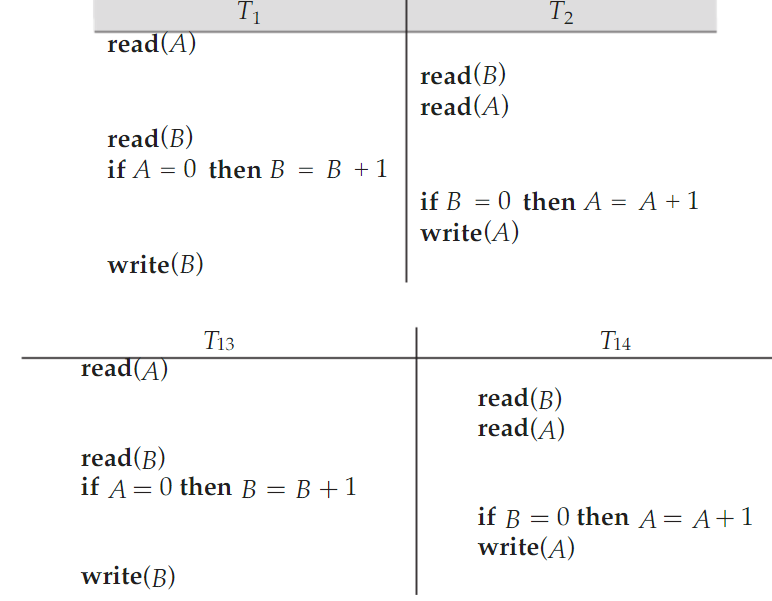
**a.Show that every serial execution involving these two transactions preserves the consistency of the database.**

There are two possible executions: T13 T14 and T14 T13



**b.Show a concurrent execution of T13 and T14 that produces an on serializable schedule.**

Any interleaving of T13 and T14 results in an on-serializable schedule.



**c. Is there a concurrent execution of T13 and T14 that produces a serializable schedule?**

There is no parallel execution resulting in a serializable schedule. From part a. we know that a serializable schedule results in A=0∨B=0. Suppose we start with T13 read (A). Then when the schedule ends, no matter when we run the steps of T2, B=1. Now suppose we start executing T14 prior to completion of T13. Then T2 read (B) will give B a value of 0. So when T2 completes, A=1. Thus B=1∧A=1→¬(A=0∨B=0). Similarly for starting with T14 read (B).

**14.19 Explain why the read-committed isolation level ensures that schedules are cascade-free.**

The read-committed isolation level ensures that a transaction reads only the committed data. A transaction Ti cannot read a data item X which has been modiﬁed by a yet uncommitted concurrent transaction Tj. This makes Ti independent of the success or failure of Tj. Hence, the schedules which follow read committed isolation level become cascade-free.