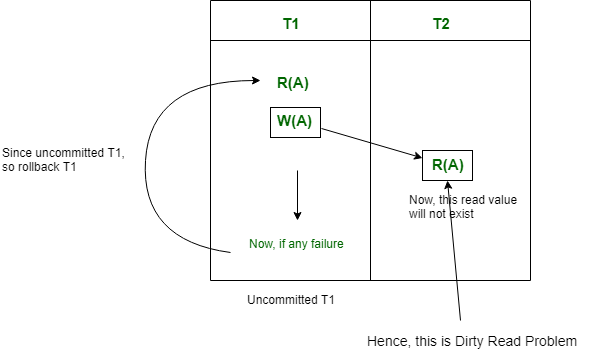
**[Dirty Read Problem](https://www.geeksforgeeks.org/dbms-dirty-read-in-sql/" \t "_blank):**  
When a Transaction reads data from uncommitted write in another transaction, then it is known as *Dirty Read*. If that writing transaction failed, and that written data may updated again. Therefore, this causes *Dirty Read Problem*.

In other words,

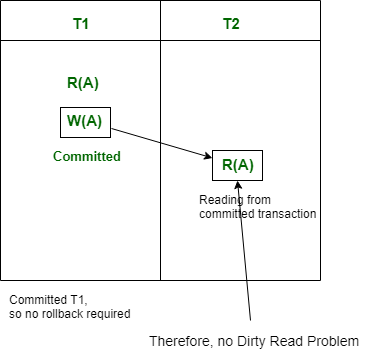
*Reading the data written by an uncommitted transaction is called as dirty read.*

It is called as dirty read because there is always a chance that the uncommitted transaction might roll back later. Thus, uncommitted transaction might make other transactions read a value that does not even exist. This leads to inconsistency of the database.

For example, let’s say transaction 1 updates a row and leaves it uncommitted, meanwhile, Transaction 2 reads the updated row. If transaction 1 rolls back the change, transaction 2 will have read data that is considered never to have existed.

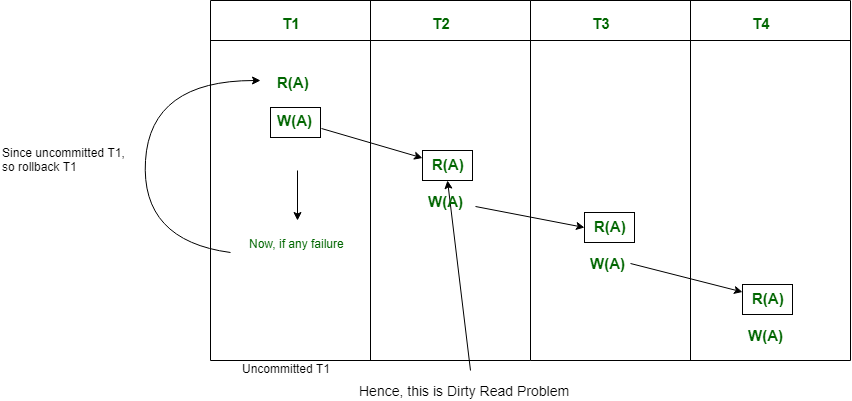


Note that there is no Dirty Read problem, is a transaction is reading from another committed transaction. So, no rollback required.



**Cascading Rollback:**  
If in a schedule, failure of one transaction causes several other dependent transactions to rollback or abort, then such a schedule is called as a Cascading Rollback or Cascading Abort or Cascading Schedule. It simply leads to the wastage of CPU time.  
These Cascading Rollbacks occur because of **Dirty Read problems**.

For example, transaction T1 writes uncommitted x that is read by Transaction T2. Transaction T2 writes uncommitted x that is read by Transaction T3.  
Suppose at this point T1 fails.  
T1 must be rolled back, since T2 is dependent on T1, T2 must be rolled back, and since T3 is dependent on T2, T3 must be rolled back.

  
  
  
Because of T1 rollback, all T2, T3, and T4 should also be rollback (Cascading dirty read problem).

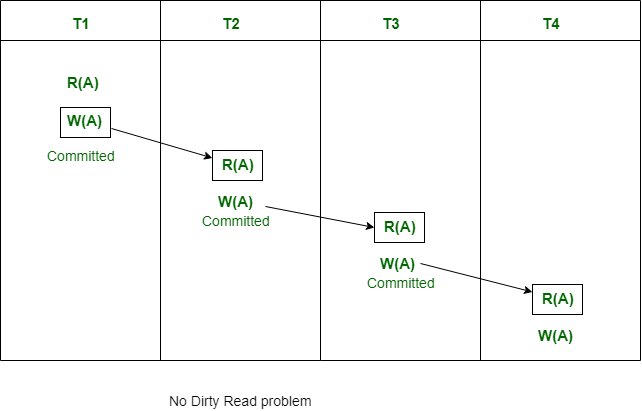
This phenomenon, in which a single transaction failure leads to a series of transaction rollbacks is called **Cascading rollback**.

**Cascadeless Schedule:**  
This schedule avoids all possible *Dirty Read Problem*.

In Cascadeless Schedule, if a transaction is going to perform read operation on a value, it has to wait until the transaction who is performing write on that value commits. That means there must not be **Dirty Read**. Because Dirty Read Problem can cause *Cascading Rollback*, which is inefficient.

Cascadeless Schedule avoids cascading aborts/rollbacks (ACA). Schedules in which transactions read values only after all transactions whose changes they are going to read commit are called cascadeless schedules. Avoids that a single transaction abort leads to a series of transaction rollbacks. A strategy to prevent cascading aborts is to disallow a transaction from reading uncommitted changes from another transaction in the same schedule.

In other words, if some transaction Tj wants to read value updated or written by some other transaction Ti, then the commit of Tj must read it after the commit of Ti.



Note that Cascadeless schedule allows only committed read operations. However, it allows uncommitted write operations.

Also note that Cascadeless Schedules are always [recoverable](https://www.geeksforgeeks.org/dbms-types-of-recoverability-of-schedules-and-easiest-way-to-test-schedule-set-2/), but all recoverable transactions may not be Cascadeless Schedule.