In this scenario shows an involvement of concurrency control in a database system, where three transactions (T1, T2, and T3) interact with shared data. **Transaction T1 A Successful Transaction:**

Transaction T_1 begins by acquiring an exclusive lock (Lock-X) on data item A in step 1. The exclusive lock ensures that no other transaction can read or modify A until T_1 releases the lock. After securing the lock, T_1 proceeds to read A in step 2 and then writes to A in step 3, updating its value. This sequence of operations indicates that T_1 has modified the data item.

In step 4, T1 moves to acquire a shared lock (Lock-S) on data item B, reaching a lock point (LP). The lock point signifies a stage where T1 has locked all necessary resources, preventing conflicts with other transactions. After locking B, T1 reads the value of B in step 5 and then releases both locks on A and B in step 6. The successful unlocking of these resources indicates that T1 has completed its operations and can commit its changes, thereby maintaining data consistency and integrity.

Transaction T2 **dirty read and rollback**. Transaction T2 begins after T1 has completed its operations. In step 7, T2 attempts to acquire an exclusive lock on A. Once T2 secures the lock, it reads the value of A in step 8. However, this is where the problem of a dirty read arises. The value that T2 reads has been modified by T1, but the change has not yet been fully committed by T1. As a result, T2 reads uncommitted data.

In step 9, T2 tries to write to A, but the system detects that it has read uncommitted data, leading to a violation of concurrency control rules. To prevent data inconsistency, the system triggers a rollback for T2. The rollback ensures that any changes made by T2 are undone, and the system is reverted to a consistent state. This rollback highlights the importance of handling dirty reads and the role of concurrency control mechanisms in ensuring database reliability.

Transaction T3 **Another Dirty Read and Rollback**. Transaction T3 experiences a similar fate as T2. It begins by attempting to read the value of A after T1 has modified it. In step 11, T3 tries to acquire a shared lock (Lock-S) on A and subsequently reads the value of A. However, like T2, T3 is also reading the uncommitted value of A from T1, which leads to another instance of a dirty read. The system again detects the problem and triggers a rollback for T3 to prevent the propagation of inconsistent data.

Dirty Reads and Rollbacks: Key Issues. The term "dirty read" refers to a situation where a transaction reads data that has been modified by another transaction but not yet committed. This is a major issue in database concurrency because it can lead to inconsistencies if multiple transactions are allowed to operate on uncommitted data. In this scenario, both T2 and T3 fall victim to dirty reads caused by T1's uncommitted write to A.

To address this, the system enforces strict concurrency control mechanisms. When it detects that T2 and T3 have read uncommitted data, it triggers a rollback for each transaction. Rollbacks are a safeguard in database systems, ensuring that any transaction that has operated on faulty data is undone, thus preserving the consistency of the overall system.