1. First it will scan from bottom to top to find the most recent checkpoint, which is checkpoint (T_0)

We have transaction T_0 and T_1

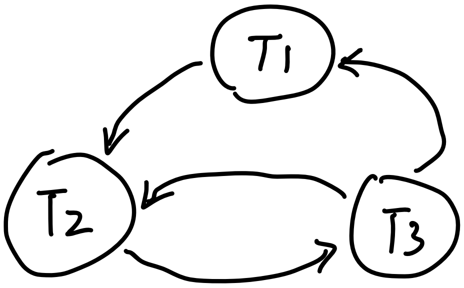
T_0 begins before checkpoint, but never committed or aborted ⇒ undo

T_1 begins after checkpoint and committed after checkpoint ⇒ redo

To undo T_0 : < T_0 , B, 2000 > < T_0, A , 1000 > < T_0 , abort >
will be written

To redo T_1: Z will be set to 150 , but no log will be writen

2、 For testing for conflict serializability



$T_1 \rightarrow T_2$: as T_1 read(A)
T_2 write(A) ∴ conflict

$T_2 \rightarrow T_3$: as T_2 read(B)
T_3 write(B) ∴ conflict

$T_3 \rightarrow T_1$: as T_3 read(A)
T_1 write(A) ∴ conflict

$T_3 \rightarrow T_2$: as T_3 read(A)
T_2 write(A) ∴ conflict

It is not acyclic ∴ not conflict serializable

3. a) Since in T_1 it will release the X-lock right after write(A). Then T_2 will have a chance to hold a S-lock on A, and to read the updated A by T_1. T_2 might potentially applied A to write B which then will be read by T_3. In this case, since update on A in T_1 is not committed, if T_1 abort, A will be rolled back to the state before any updates happens. However, T_2 & T_3 are already using updated version of A. Thus, it will result a cascading rollback of T_2 and T_3 as well, as they are using an invalid A for potential write.

b) Strict 2PL meaning T_1 will not release X-lock on A until abort. So that T_2 which accquires S-lock on A has to wait for T_1's X-lock to be released and T_3 will wait for T_2. In this case, only if T_1 is committed which means the update is valid or aborted, T_2 and T_3 can have a chance to read (A) and make modification further. Thus, T_2 and T_3 will never have a chance to use an invalid A to make updates. So, they don't need to be rolled back.

4. Cascading abort occur when one transaction (T_1) aborts after making some changes that other transactions $(T_2$ & $T_3)$ have already read. In this case If T_1 aborts, any other transactions that has read its uncommited changes must also abort to maintain consistency.

But for the case where all X-locks are held until after the transaction holding the lock committed or aborted, and while holding X-locks, S-locks and further X-locks are not allowed to be acquired on the same items. In this case, other transactions are prevented from accessing the affected/updated items before it is committed. This eliminate the possibility of reading uncommitted data to improve Isolation, atomicity and Consistency which eliminates the needs & risk of cascading aborts.