1a. T2: R(Z), W(Z) With the new Z object in the new transaction there will be no conflicts because write actions by different transactions on different objects dont conflict

1b. T3: R(X), W(X)

T1	T3
R(X)	
W(X)	
	R(X)
	W(X)
	commit
R(Y)	
W(Y)	
Commit	

Write-read conflict here, T3 reads X but X was previosly modified by an uncommited transaction

2PL dissalows the above: T1: S(X), T1: R(X), T1: X(X), T1: W(X), next T3 wants to read X so it requests the shared lock but is blocked because T1 has an exclusive lock on X. This causes T2 to have to wait until T1 commits or abort.

With strict 2PL:

T1	T3
S(X)	
R(X)	
X(X)	
W(X)	
S(Y)	
R(Y)	
X(Y)	
W(Y)	
commit	
	S(X)
	R(X)
	X(X)
	W(X)
	commit

T3 tries to get the S(X) but is blocked by T1's X(X) so T3 is forced to wait

All locks held by T1 are released (S(X), S(Y), X(X), X(Y))

All locks held by T3 are released (S(X), X(X))

1c. T4: R(Y), W(Y)

T1	T4
R(X)	
W(X)	
R(Y)	
	R(Y)
	W(Y)
	commit
W(Y)	
commit	

A read-write conflict occurs here, when the actions of T4 go through this changes the value of Y so now it is not consistent throught T1 causing confusion for the dev

strict 2PL dissalows the above: T1: S(X), T1: R(X), T1: X(X), T1: W(X), T1: S(Y), T1: R(Y), T4: S(Y), T4: R(Y), next T4 wants to write to Y, the DBMS wants to put an exclusive lock on Y but its blocked because T1 has a shared lock on it. Next T1 tries to write to Y but it cant change the lock to an exclusive one because T4 also has a share lock on Y. T1 gets blocked and so does T4 causing a deadlock.

with strict 2PL:

T1	T4
S(X)	
R(X)	
X(X)	
W(X)	
S(Y)	
R(Y)	
X(Y)	
W(Y)	
commit	
	S(Y)
	R(Y)
	X(Y)
	W(Y)
	Commit

T1 is blocked when trying to upgrade the shared lock to exclusive This results in a deadlock

Here T4 tries to run, resulting in T4 being blocked when trying to put an exclusive lock on Y

All locks held by T1 are released (S(X), S(Y), X(X), X(Y))

All locks held by T4 are released (S(Y), X(Y))