DATABASE TRANSACTION

Transaction set-

T1- User A Adding an item a and checking out the cart
T2-User B Adding an item a and checking out the cart

Now we will build schedules for the given set of transactions:

1) CONFLICTING SERIALIZABLE

T1	T2
WRITE(Cart, Item a)	
	Read(Cart,item a)
Read(product_quantity)	
	Write(order,item a)
Write(product_quantity)	
commit	
	commit
	Read(product_quantity)
	Write(product_quantity)
	commit

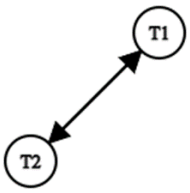
This type of conflict includes WR type of conflict from T1 to t2. And this is conflict serializable, as if we use the Precedence Graph and draw two nodes, T1 and T2, an arrow will be formed from T1 to T2. As the graph is acyclic, therefore it is conflict serializable.

2) NON CONFLICTING SERIALIZABLE

T1	T2
Read(cart,item A)	
	R(cart,item A)
Write(cart,item A)	
	W(cart,item A)
R(product_quantity)	R(product_quantity)
W(order)	W(order)
Commit	
W(product_quantity)	W(product_quantity)
	W(product_quantity)
	Commit

In this RW and WW type of conflict is taking place for user A.
This is a non-conflict serializable schedule because:
When we make the precedence graph for the two transactions, a loop is obtained in the graph hence this is a non-conflict serializable transaction whose transactions can't be swapped to obtain serializable scheduling

THERE IS A RW CONFLICT FROM T2 TO T1 AND A WW CONFLICT FROM T1 TO T2
HENCE A LOOP IS OBTAINED



We can solve this using strict two-phase locking:
S(a) shared lock
X(a) exclusive lock
U(a) unlock

T1	T2
S(A)	
Read(cart,item A)	
	R(cart,item A)
Write(cart,item A)	S(B)
X(A)	
	W(cart,item A)
R(product_quantity)	R(product_quantity)
W(order)	W(order)
U(A)	
Commit	
	Commit
W(product_quantity)	W(product_quantity)
	W(product_quantity)
	Commit

	U(B)