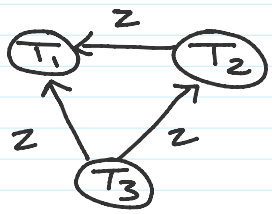
e.g. Schedule S_{11}

T_1	T_2	T_3
R(x);		
	R(Y); ✓	
w(x);		
	w(Y); ✓	
		R(z); ✓
		w(z);
	R(z); ✓	
	w(z); ✓	
w(z); ✓		



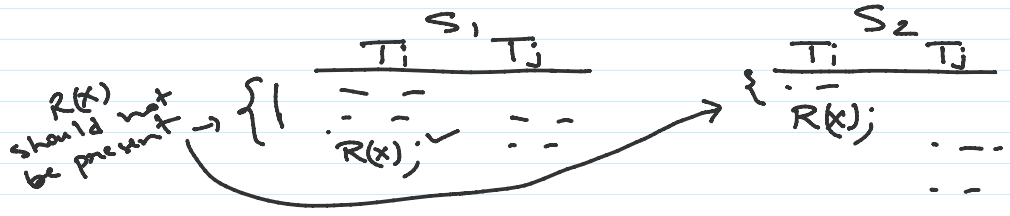
Schedule S_{11} is a conflict serializable schedule & serial schedule equivalent to it is

$T_3 \rightarrow T_2 \rightarrow T_1$

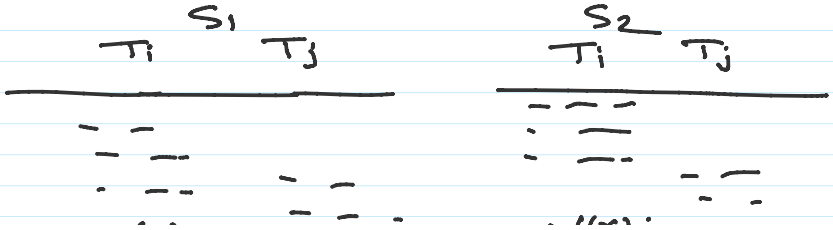
* View Serializability

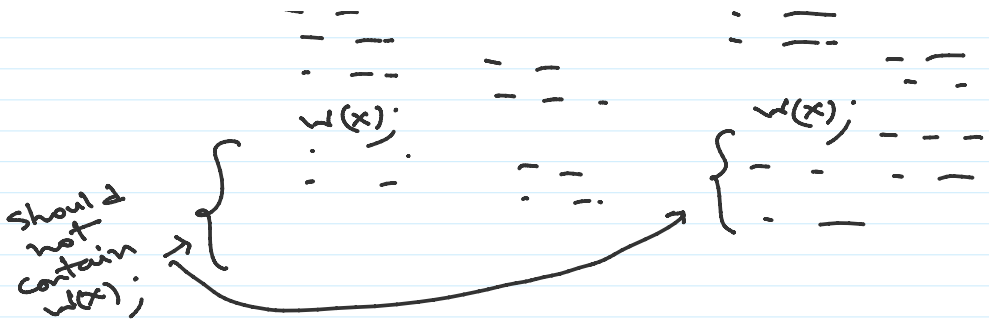
** view equivalent: let there are two schedules S_1 & S_2. S_1 & S_2 are view equivalent of each other when

(i) Initial read: If data item 'x' is initialy read by transaction T_i in schedule S_1, then same transaction T_i should also read initial value of data item 'x' in schedule S_2.

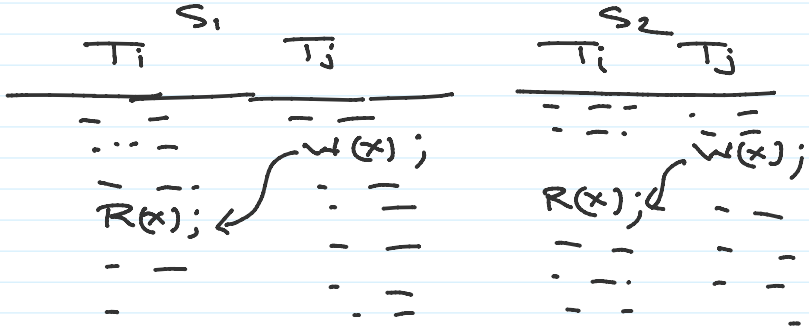


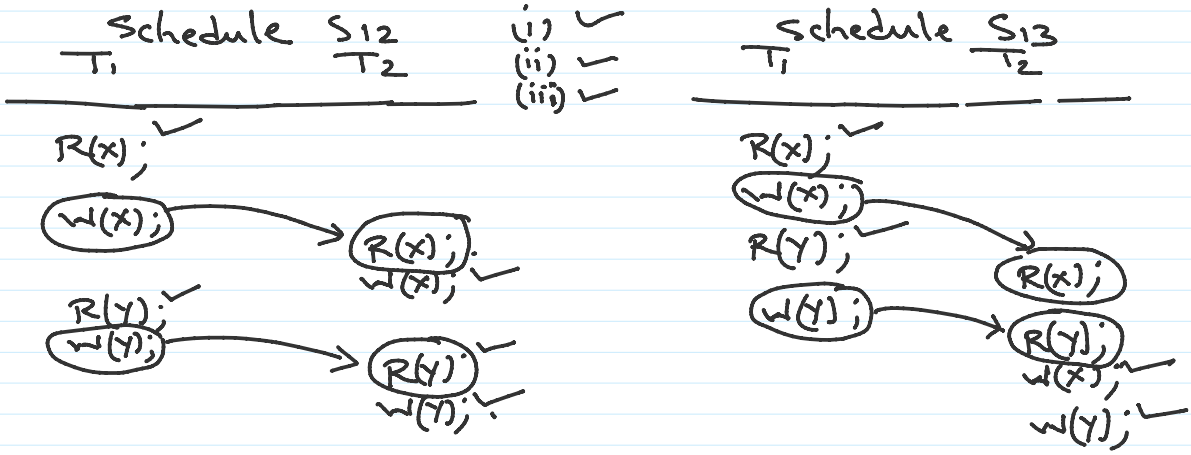
(ii) Final write: If transaction T_i writes final value of data 'x' in schedule S_1 then transaction T_i of schedule S_2 should also write final value of data item 'x'.





(iii) Read after write : If transaction T_i reads the value of data item 'x' which is written by transaction T_j in schedule S_1 then transaction T_i of schedule S_2 should also read the value of data item 'x' that is written by transaction T_j.

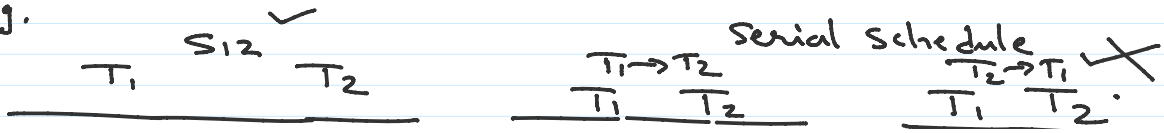


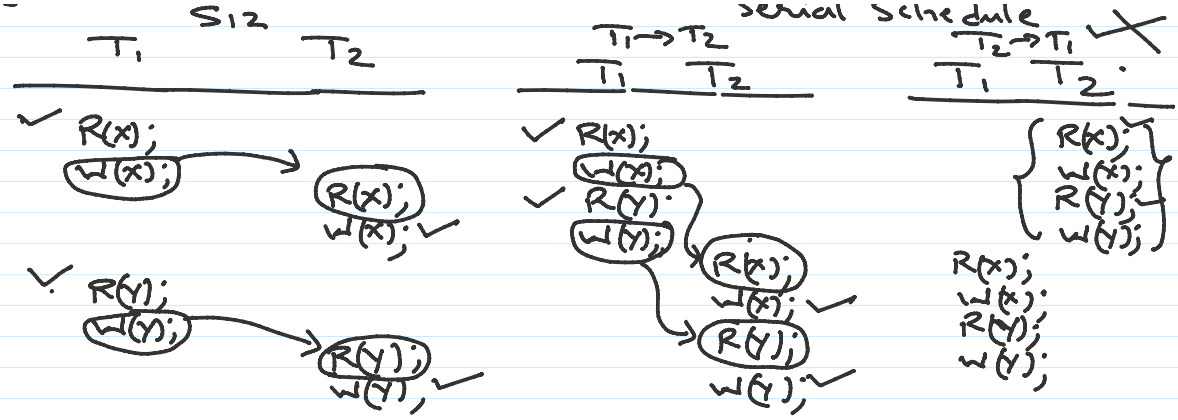


Schedule S_{12} & S_{13} are view equivalent.

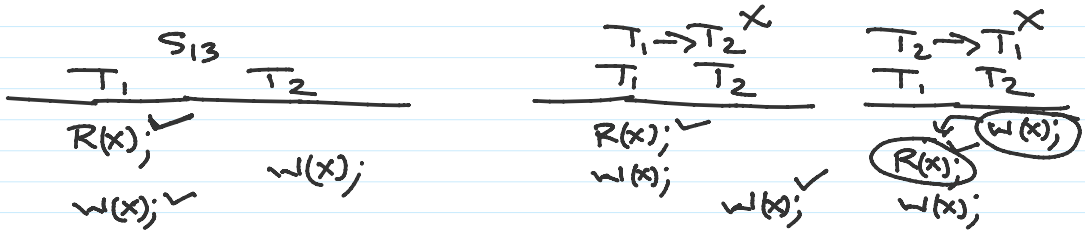
* View serializability : If any schedule 'S' is in view equivalent to any of its serial schedule then 'S' is called view serializable.

e.g.





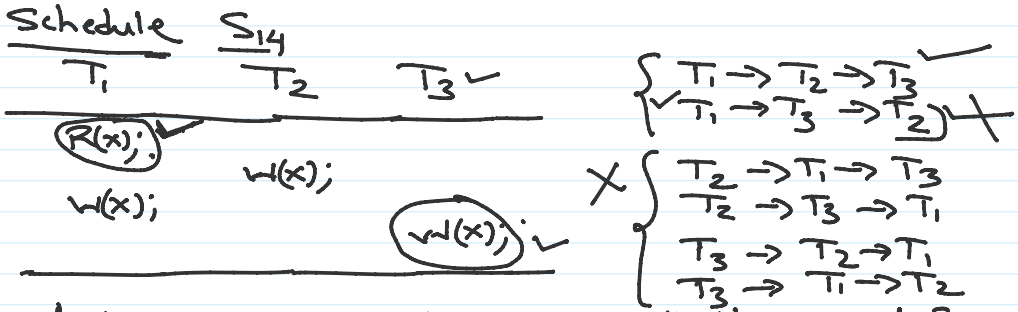
Schedule S_{12} is view serializable.



* find view serializability.

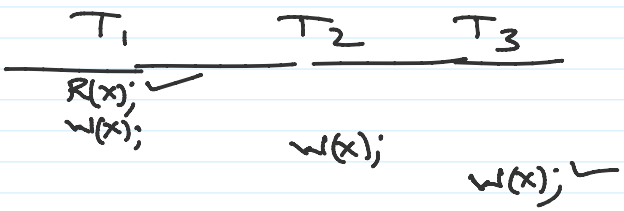
S_{13} is not a view serializable.

Schedule S_{14}



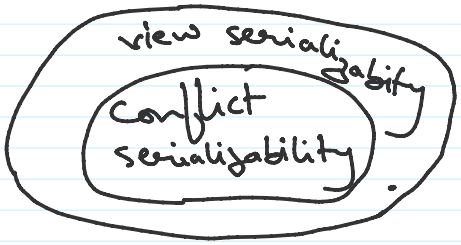
* find Schedule S_{14} is view serializable or not?

$T_1 \rightarrow T_2 \rightarrow T_3$



Schedule S_{14} is a view serializable schedule & serial schedule equivalent to it is $T_1 \rightarrow T_2 \rightarrow T_3$.

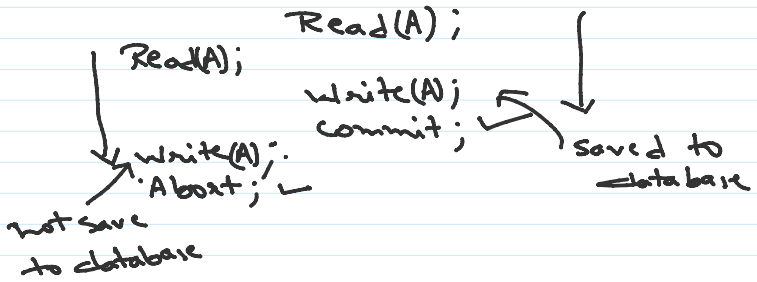
* Every conflict serializable schedule is also a view seriable schedule but its reverse is not true.



21/01/2021

Transaction isolation & atomicity :

e.g. Schedule S15



(i) **Recoverable schedule** : let transaction T_i is reading a data item 'x' which is written by transaction T_j, then if commit operation of T_j appears before commit operation of T_i then such schedule is recoverable otherwise not recoverable.

Recoverable

S

T_i	T_j
	write(x);
read(x);	
	commit;
commit;	

t_m t_n — Time line

$t_m < t_n$

S

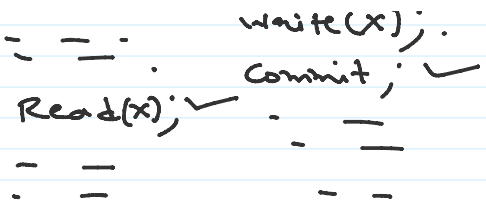
T_i	T_j
	write(x);
Read(x);	
commit;	
	Abort;

Non recoverable.

(ii) **cascadeless schedule** : A schedule is cascadeless schedule when any transact^n T_i is reading data item 'x' which is written by transact^n T_j then the commit operation of T_j must appear before read(x) operation of T_i.

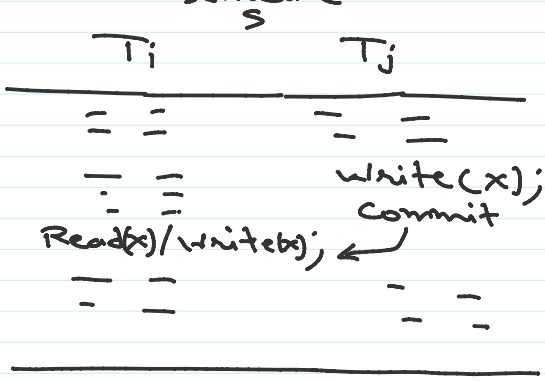
Schedule S

T_i	T_j
	write(x);
	commit;

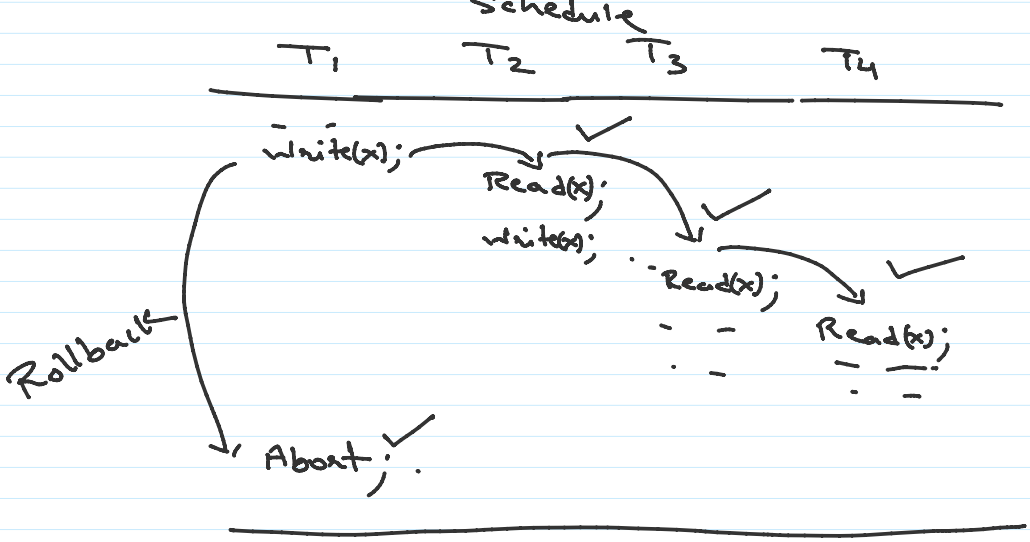


→ Every cascadeless schedule is also a recoverable schedule.

(iii) strict schedule : A schedule is a strict schedule when any transaction T_i wants to perform either of the operation (Read or write) on data item 'x' which is written by any transaction T_j then commit operation of T_j must appear before either of the operation of T_i.



(iv) cascading rollback : Due a single transaction failure, series of transaction rollback. It is called cascading rollback.



Concurrency Control

If we want to execute transaction concurrently

Concurrency control

* We want to execute transaction concurrently.
* After concurrent execution, we have to verify consistency of database. Is it possible that every concurrent execution preserve consistency ?
* NO, we can not go for concurrent execution of transactions
* We have to execute transaction concurrently with some control on them.
* If transactions are running concurrently then isolation property may no longer be preserved.

Concurrency control protocols

(a) Lock based protocol : (Concept is similar to process synchronizath^n)

(i) Binary lock : A lock with two states is called binary lock.

States → 1 locked
→ 0 unlocked

→ At a time only one transaction can hold a lock on data item for performing either of the operation (Read / write). { Lock(x) = 1 ; }

→ once the transaction completes its operation, it unlocks data item. { lock(x) = 0 ; }

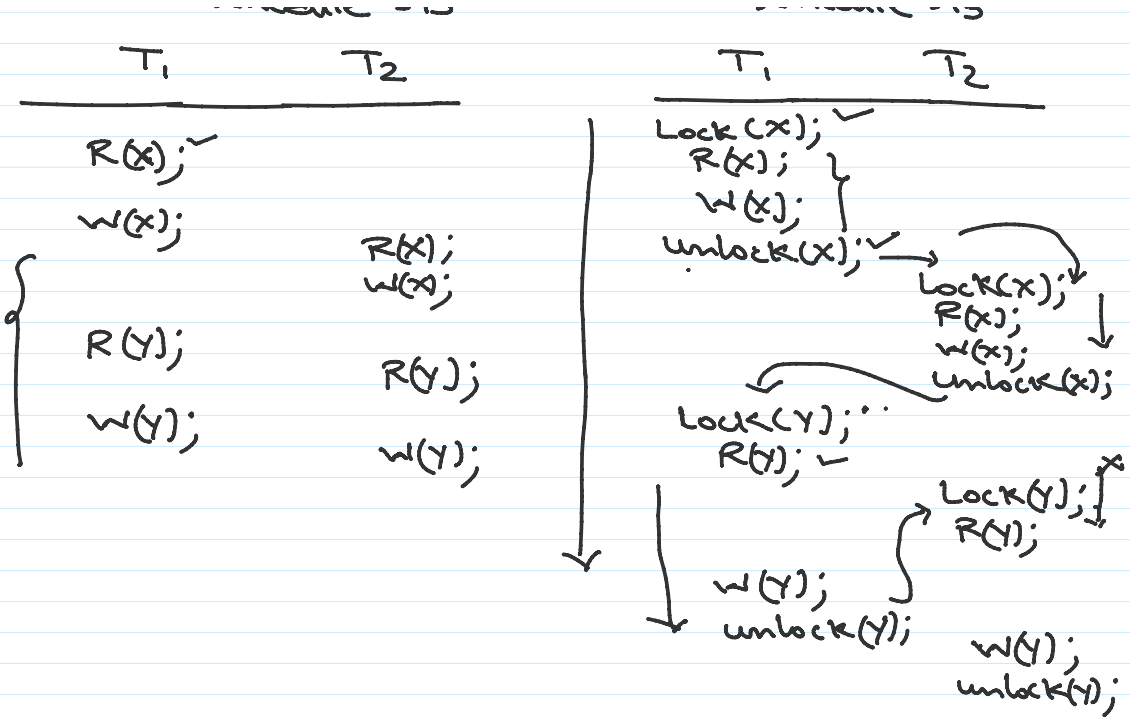
(wait) Lock_item(x)	(signal) Unlock_item(x)
L: If lock(x) = 0 then lock(x) ⇐ 1 ; else wait until lock(x) = 1 ; Block & add in list ; when lock(x) = 1 goto L ;	Lock(x) ⇐ 0 unblock one transaction from list of blocked transaction.

e.g. Schedule S_{15}

T_1	T_2

Schedule S_{15}

T_1	T_2



22/01/2021 :

The main disadvantage of Binary lock protocol is that it treats both read & write to a data item same & lock the data item.

If data item is used for read purpose only then we can allow morethan one transactions to read same data item parallelly.

(ii) Shared & Exclusive lock:

1. Shared (Read) lock: If a transaction Ti has obtained a shared lock on data item 'X', then Ti can read the value of data item 'X' but not write on 'X'.

   Lock_S(X) : it means shared lock on data item 'X'.

   more than one transactions can obtain shared lock on any data item at the same time.

2. Exclusive (write) lock: If a transaction Ti has obtained an exclusive lock on data item 'X' then It can both read from & write on 'X'.

   Lock_X(X) : it means exclusive lock on data item 'X'.

   At a time only one transaction can obtain exclusive lock on any data item.

   lock compitibility matrix :

   Tj

lock compitibility matrix :

T_i \ T_j	lock_s(x)	lock_x(x)
lock_s(x)	✓	✗
lock_x(x)	✗	✗

(Reader write problem of process synchronizatn)

lock_s(x) : shared lock

```
If lock_s(x) = "unlocked" then
    lock_s(x) = "Read_locked";
    no_of_reads(x) ← 1;
else if lock_s(x) = "Read_locked" then
    no_of_reads(x) ← no_of_reads(x) + 1
else
    wait until lock(x) = "unlocked"
    & add transaction to a waiting list.
```

lock_x(x) : exclusive lock

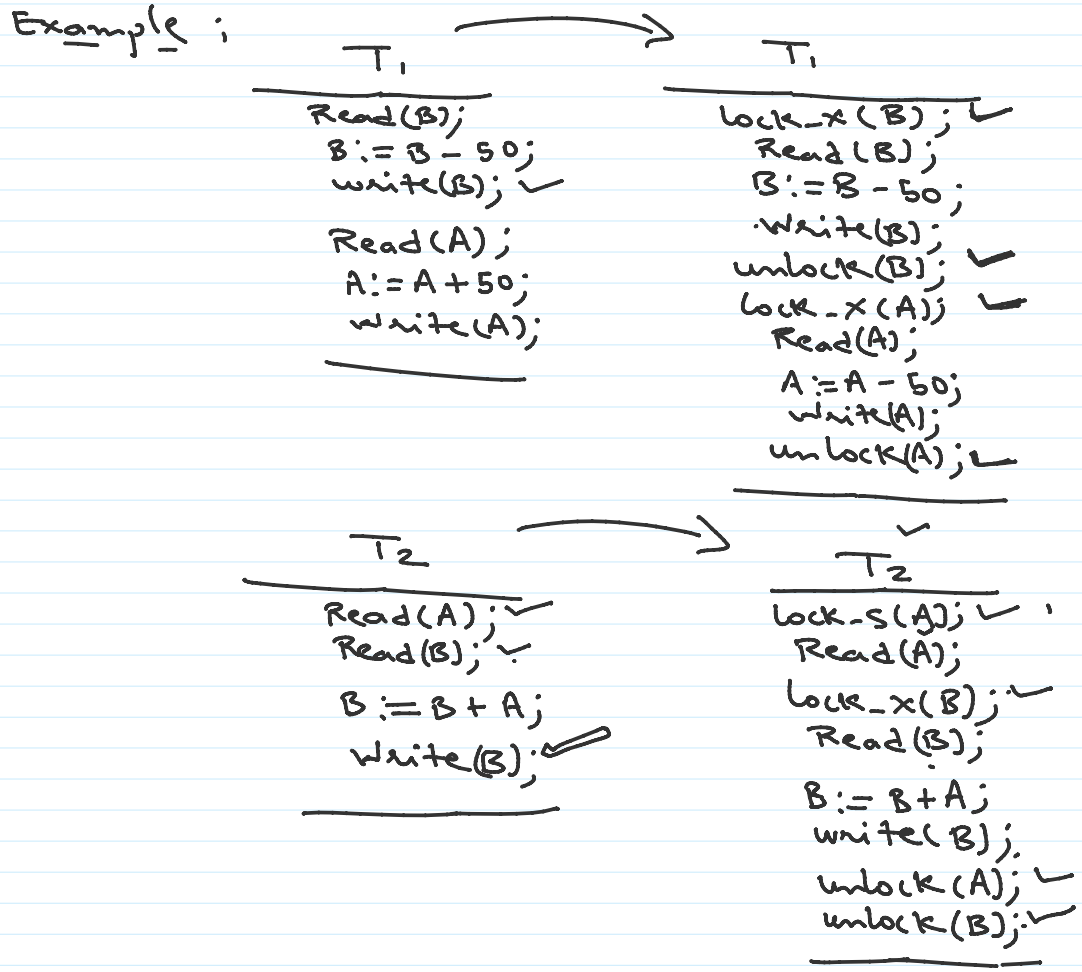
```
If lock_x(x) = "unlocked" then
    lock_x(x) = "write_locked";
else
    wait until lock(x) = "unlocked"
    & add transaction to a waiting list.
```

unlock(x) : It unlocks either of the locks from data item 'x'.

```
If lock(x) = "write_locked" then
    lock(x) = "unlocked";
    wakeup one of the waiting transactn
    from waiting list;
else if lock(x) = "Read_locked" then
    no_of_reads(x) = no_of_reads(x) - 1;
    if no_of_reads(x) = 0 then
        lock(x) = "unlocked";
        wakeup one of the waiting transactn
        from waiting list;
```

wakeup one of the waiting transact" from waiting list;

Example :

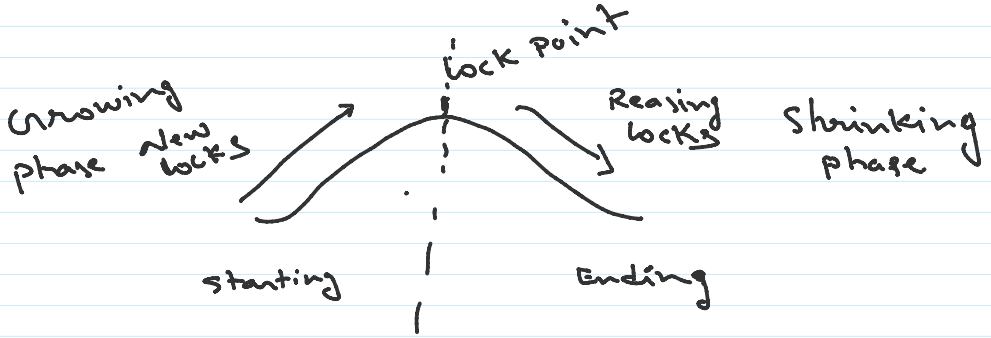


* Two-phase locking protocol :

The protocol works in two phases namely: Growing phase & shrinking phase.

(a) Growing phase : In growing phase, a transaction may obtain newlocks by may not release any lock.

(b) Shrinking phase : In shrinking phase, a transaction may release locks but may not obtain new locks.



If progress is not in concurrent execution, it means deadlock. (No process execution)