Lecture 4

List the conflict operations in the following schedule

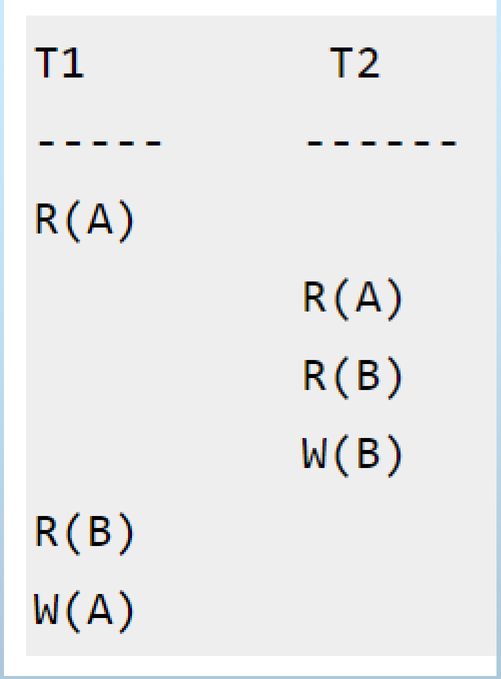
- S_a : $r_1(x)$, $r_2(x)$, $w_1(x)$, r_1 (y), $w_2(x)$, $w_1(y)$

$r_1(x)$	$w_2(x)$	T_1 -> T_2
$r_2(x)$	$w_1(x)$	T_2 -> T_1
$w_1(x)$	$w_2(x)$	

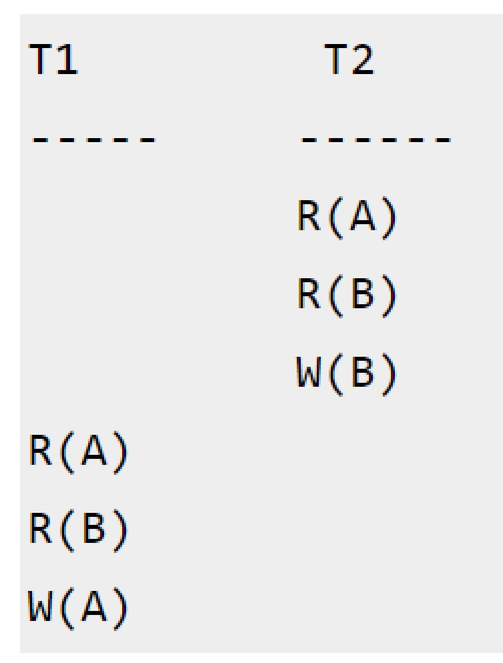
Non conflict serializable

Example

- Is this conflict serializable??



```
T1          T2
-----      ------
R(A)
           R(A)
           R(B)
           W(B)
R(B)
W(A)
```



```
T1          T2
-----      ------
           R(A)
           R(B)
           W(B)
R(A)
R(B)
W(A)
```

conflict serializable :T_2 –T_1

View Serializability

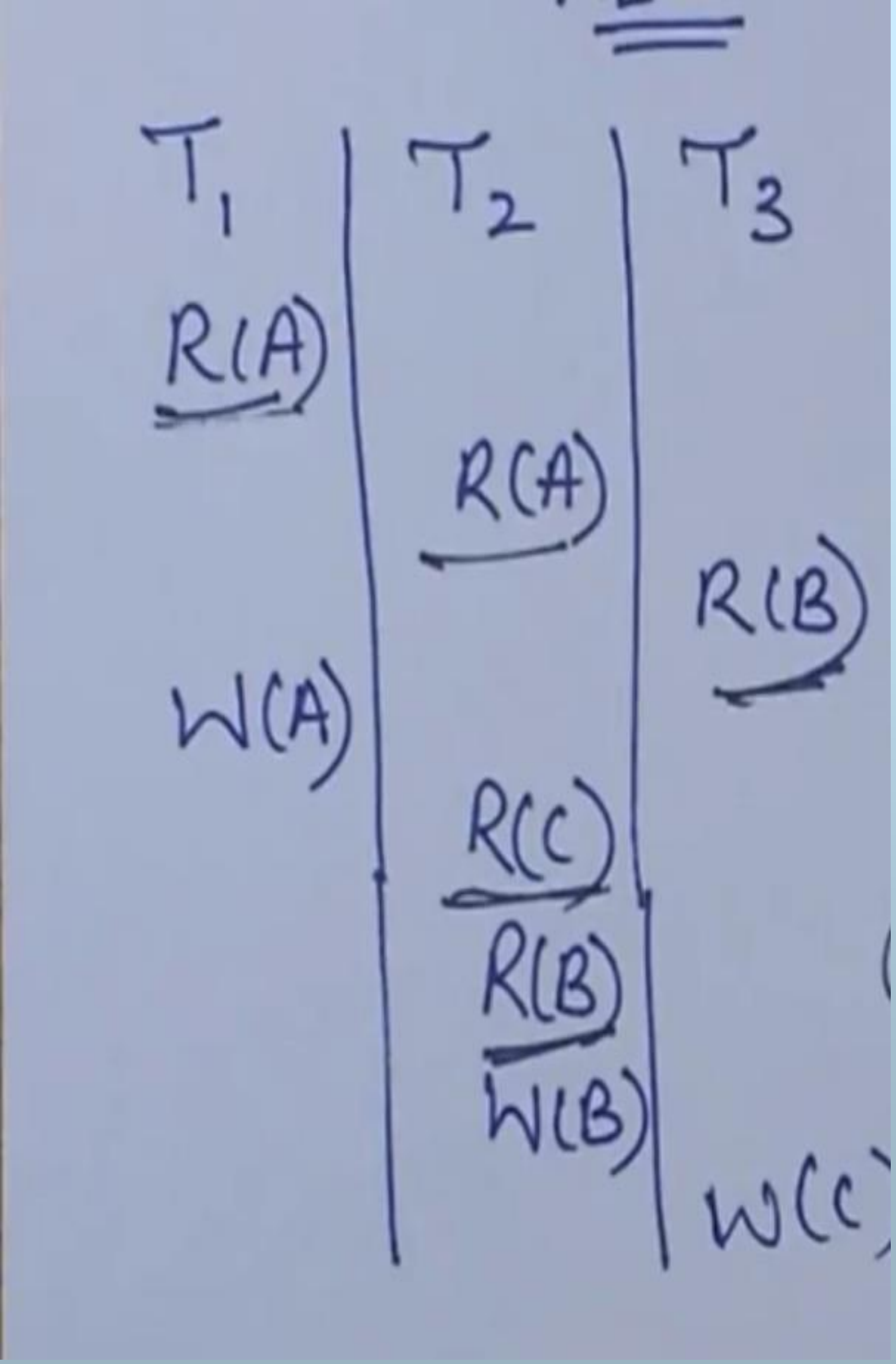
- Let *S* and *S´* be two schedules with the same set of transactions. *S* and *S´* are **view equivalent** if the following three conditions are met:

  1. (**Initial Read**) For each data item *Q,* if transaction T_i reads the initial value of *Q* in schedule *S,* then transaction T_i must, in schedule *S´*, also read the initial value of *Q.*

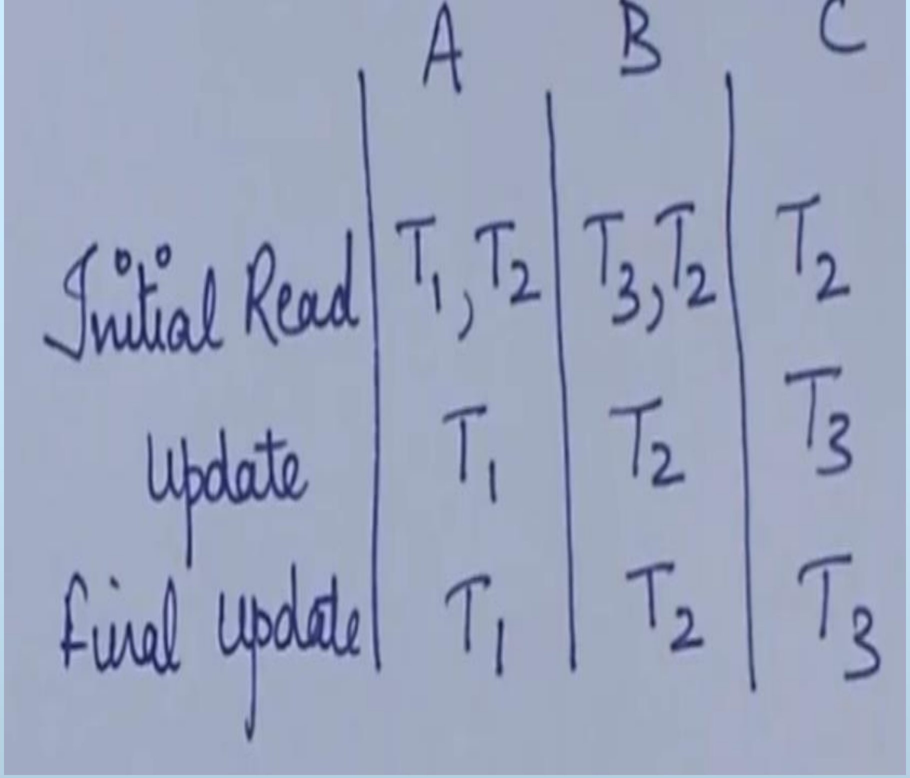
  2. (**Update read**) For each data item *Q* if transaction T_i executes **read**(*Q)* in schedule *S*, and that value was produced by transaction T_j (if any), then transaction T_i must in schedule *S´* also read the value of *Q* that was produced by transaction T_j .

  3. .(**final write**) For each data item *Q*, the transaction (if any) that performs the final **write**(*Q*) operation in schedule *S* must perform the final **write**(*Q*) operation in schedule *S´*.

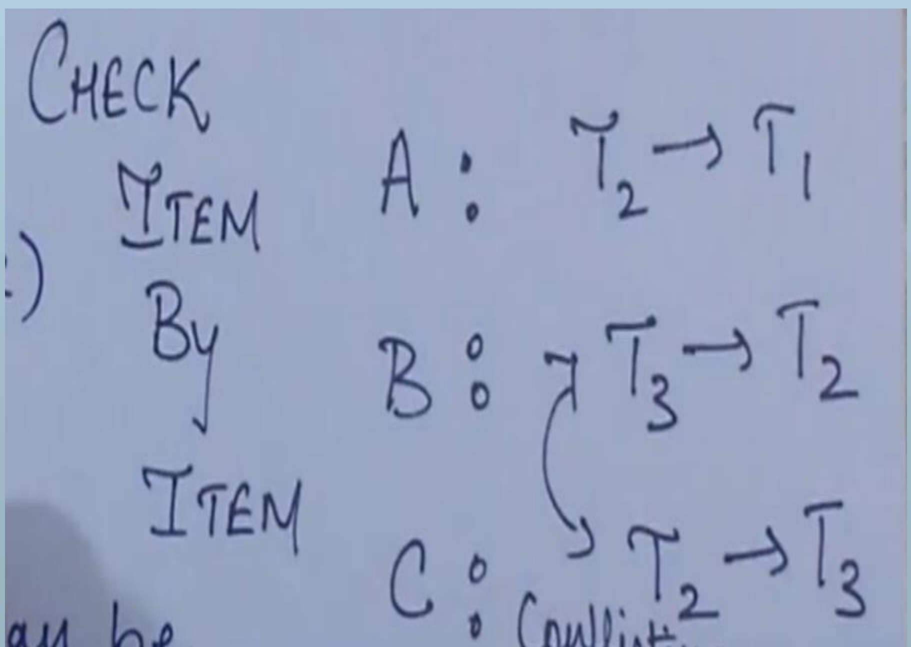
As can be seen, view equivalence is also based purely on **reads** and **writes** alone.



T_1	T_2	T_3
R(A)		
	R(A)	
		R(B)
W(A)		
	R(C)	
	R(B)	
	W(B)	
		W(C)



	A	B	C
Initial Read	T_1, T_2	T_3, T_2	T_2
update	T_1	T_2	T_3
final update	T_1	T_2	T_3



CHECK

ITEM By ITEM

A : $T_2 \rightarrow T_1$

B : $T_3 \rightarrow T_2$

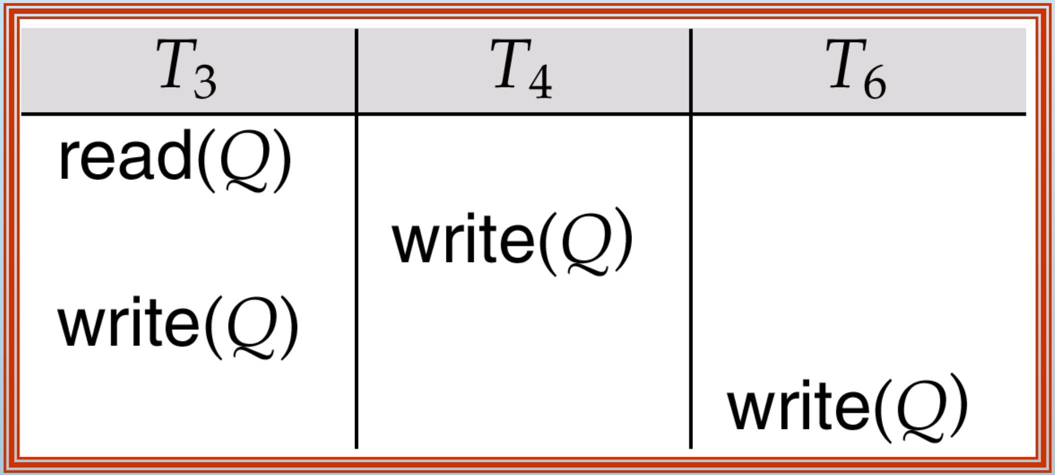
C : $T_2 \rightarrow T_3$

Conflict

an be

View Serializability (Cont.)

- A schedule *S* is **view serializable** it is view equivalent to a serial schedule.
- Every conflict serializable schedule is also view serializable.
- Schedule 9— a schedule which is **view-serializable** (it is equivalent to serial schedule <T_3,T_4,T_6>), but *not* conflict serializable.

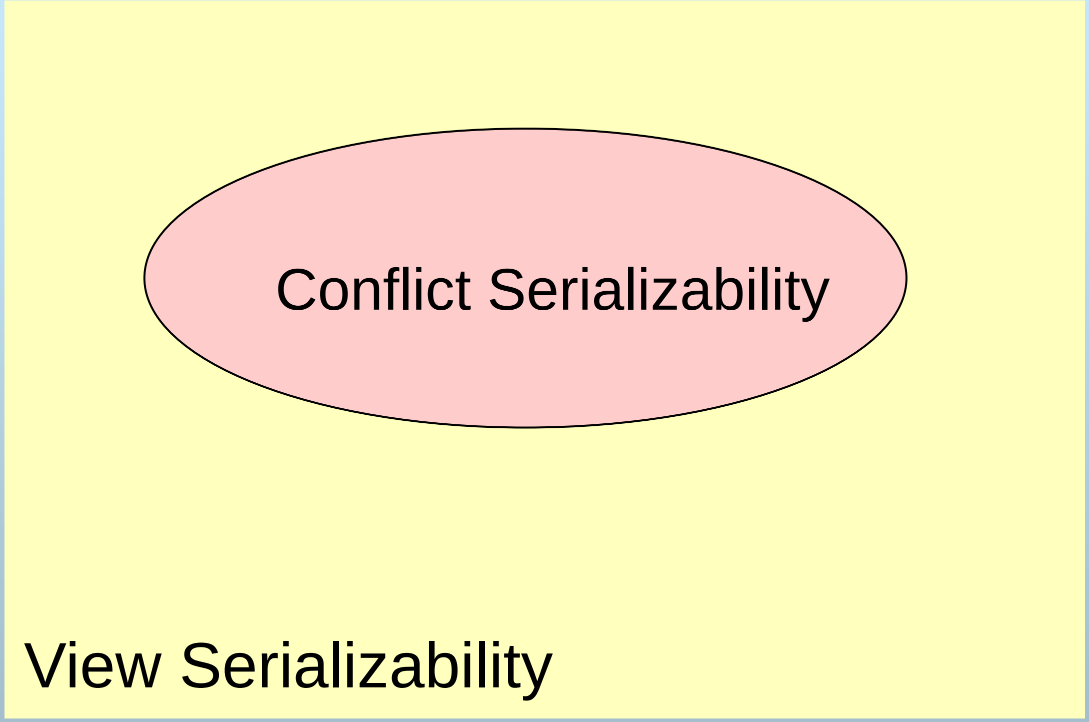


T_3	T_4	T_6
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

- Every view serializable schedule that is not conflict serializable has **blind writes (write variable without reading it).**

Note: Blind writes appear in any view-serializable schedule that is not conflict serializable.

Every conflict-serializable schedule is also view serializable, but there are view serializable schedules that are not conflict serializable.



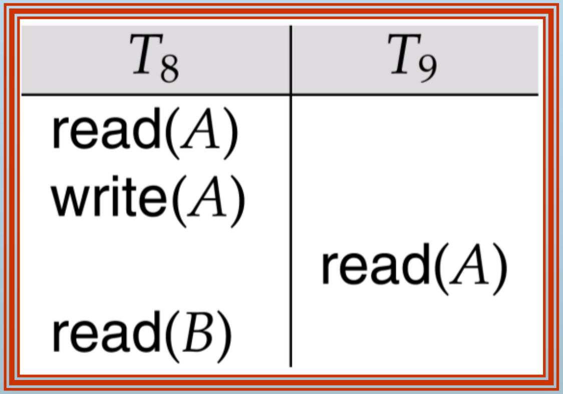
Conflict Serializability

View Serializability

Recoverability

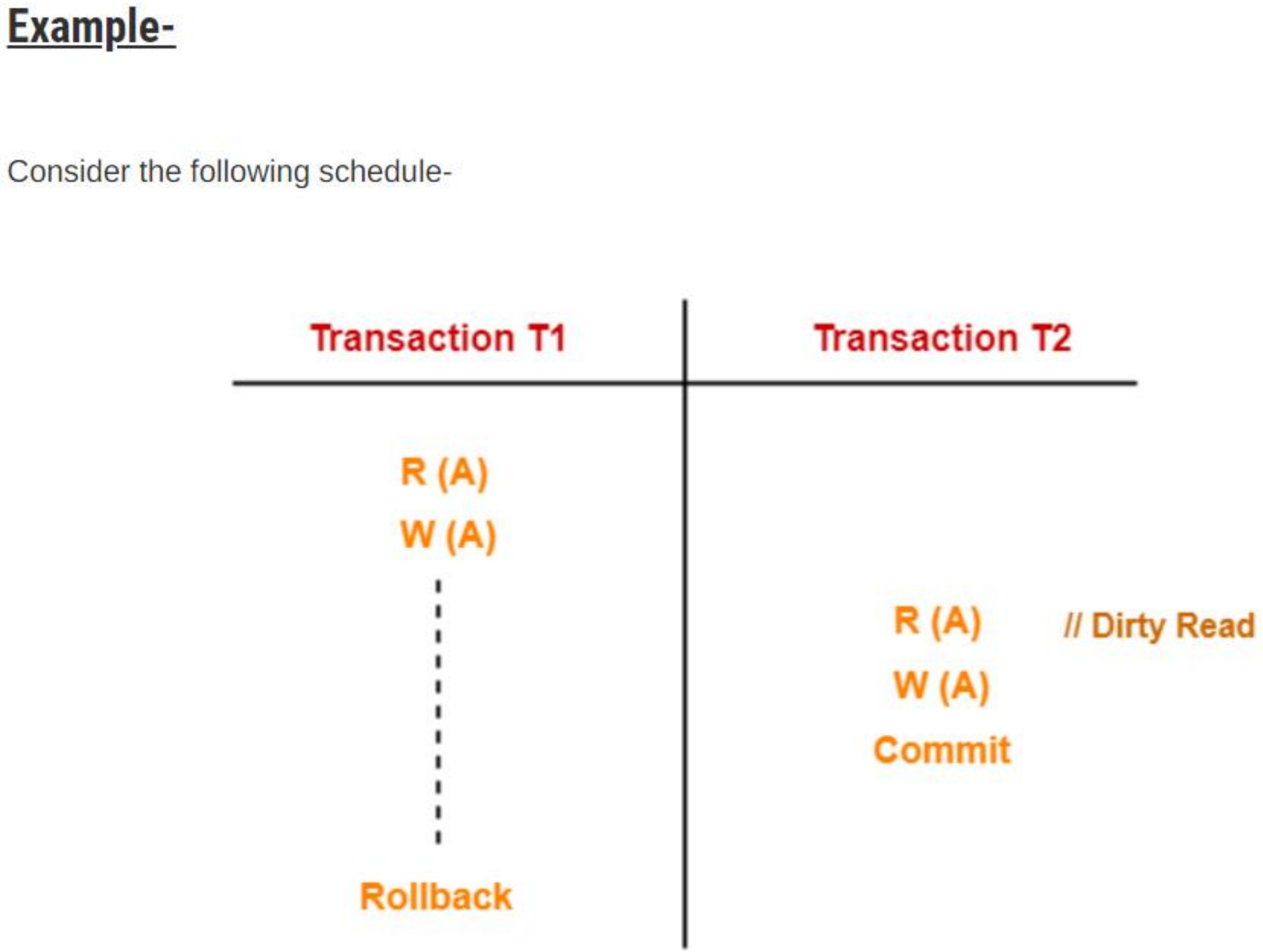
Need to address the effect of transaction failures on concurrently running transactions.

- **Recoverable schedule** — if a transaction T_j reads a data items previously written by a transaction T_i , the commit operation of T_i appears before the commit operation of T_j.
- Is the following schedule recoverable or not ??



T_8	T_9
read(A)	
write(A)	
	read(A)
read(B)	

- If T_8 should abort, T_9 would have read (and possibly shown to the user) an inconsistent database state. Hence database must **ensure that schedules are recoverable**.



Example-

Consider the following schedule-

Transaction T1	Transaction T2
R (A)	
W (A)	
	R (A) // Dirty Read
	W (A)
	Commit
Rollback	

Irrecoverable

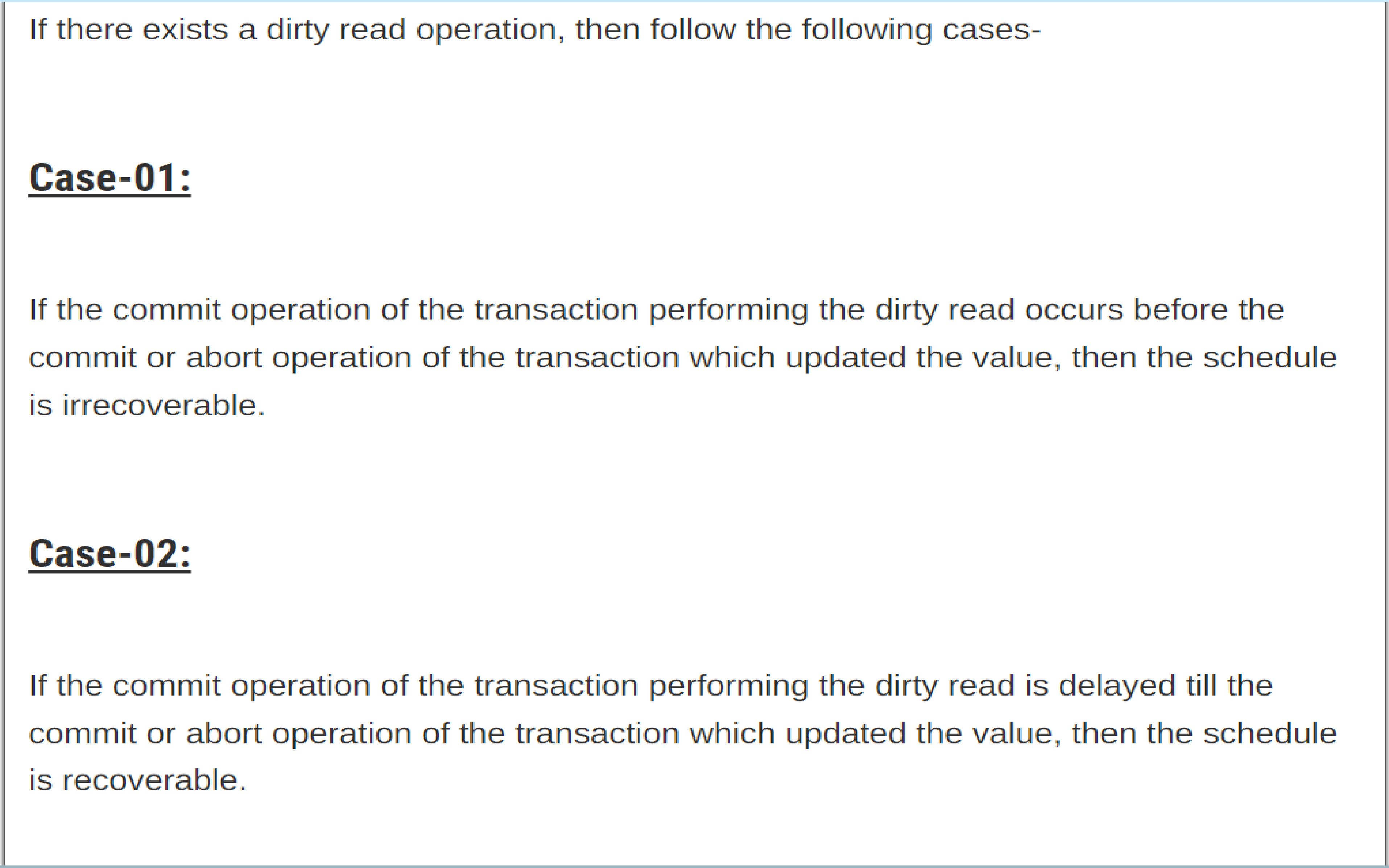
Example-

Consider the following schedule-

Transaction T1	Transaction T2
R (A)	
W (A)	
	R (A) // Dirty Read
	W (A)
Commit	
	Commit // Delayed

recoverable

Method to check recoverability



If there exists a dirty read operation, then follow the following cases-

Case-01:

If the commit operation of the transaction performing the dirty read occurs before the commit or abort operation of the transaction which updated the value, then the schedule is irrecoverable.

Case-02:

If the commit operation of the transaction performing the dirty read is delayed till the commit or abort operation of the transaction which updated the value, then the schedule is recoverable.

Examples

$S_a : r_1(x), w_1(x), r_1 (y), r_2(x), w_2(x) , C_2, w_1(y) , C_1$

Non recoverable

$S_b : r_1(x), w_1(x), r_2(x), r_1 (y), w_2(x) , C_2, a_1$

Non recoverable

$S_c : r_1(x), w_1(x), r_2(x), r_1 (y), w_2(x) , w_1(y) , C_1, C_2$

Recoverable

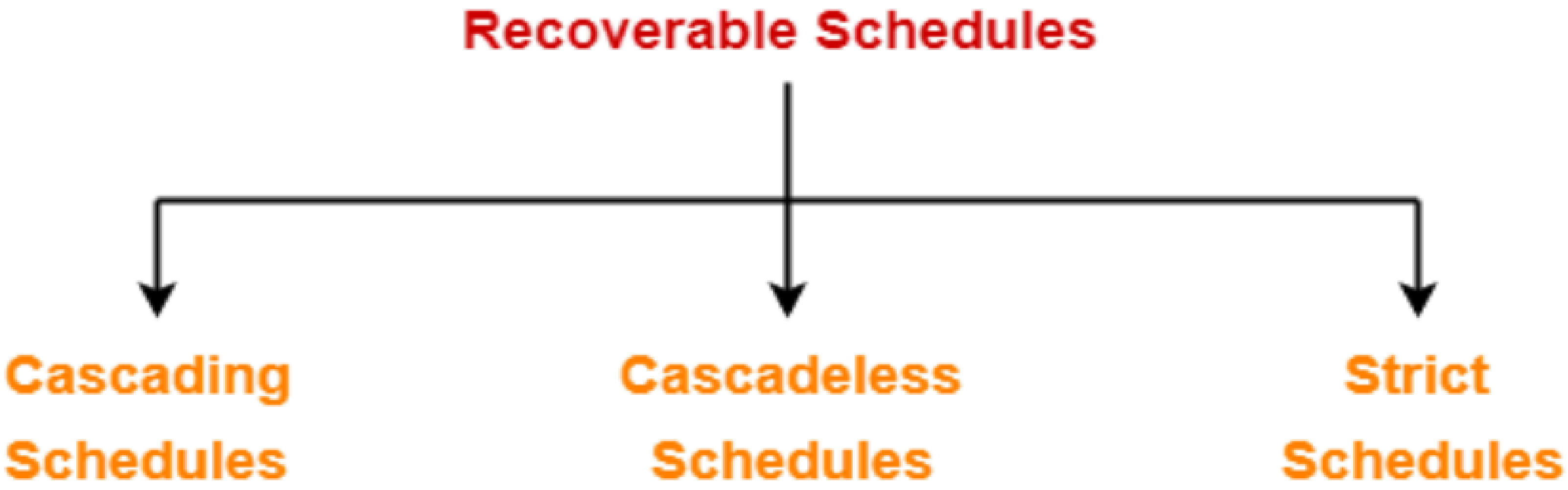
$S_c : r_1(x), w_1(x), r_2(x), r_1 (y), w_2(x) , w_1(y) , a_1, a_2$

Recoverable

none of the transactions has yet committed (so the schedule is recoverable)

Types of Recoverable Schedules-

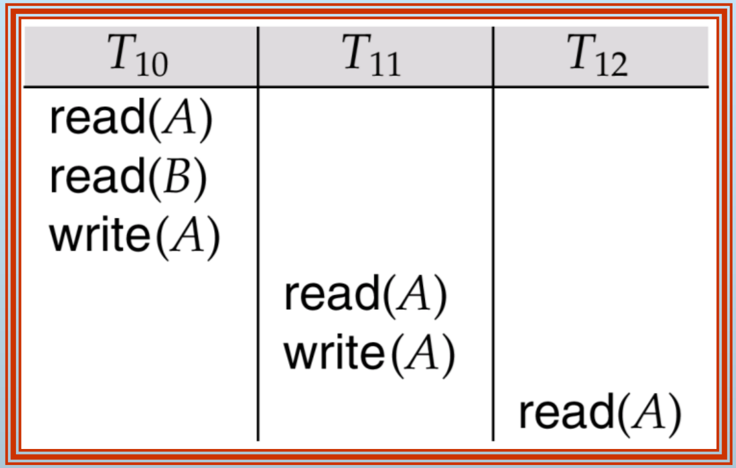
A recoverable schedule may be any one of these kinds-



1. Cascading Schedule
2. Cascadeless Schedule
3. Strict Schedule

Cascading rollback (Cascading abort)

- **Cascading rollback** – a single transaction failure leads to a series of transaction rollbacks.
- Consider the following schedule where none of the transactions has yet committed (so the schedule is recoverable)

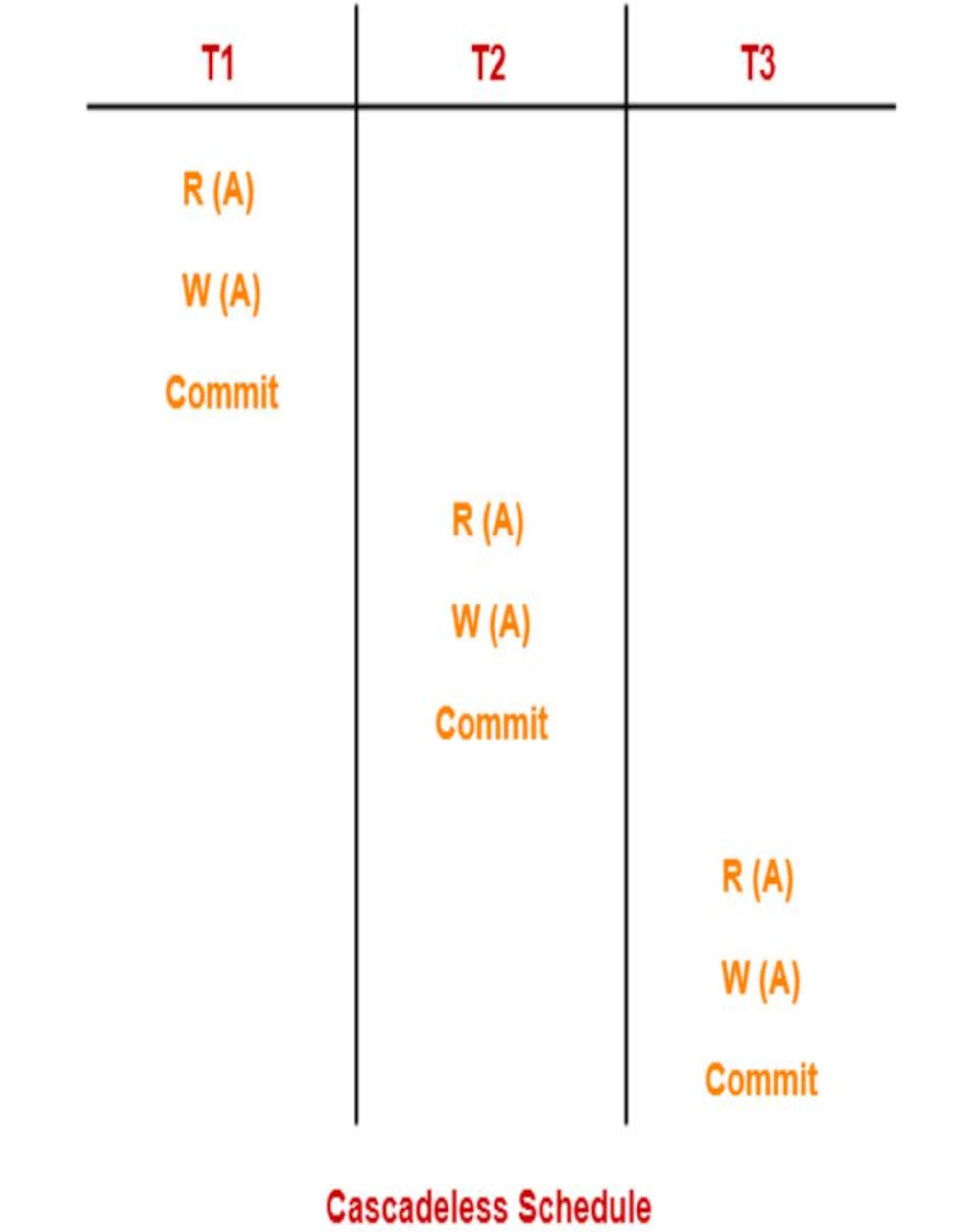


T_{10}	T_{11}	T_{12}
read(A)		
read(B)		
write(A)		
	read(A)	
	write(A)	
		read(A)

- If T_{10} fails, T_{11} and T_{12} must also be rolled back.
- Can lead to the undoing of a significant amount of work

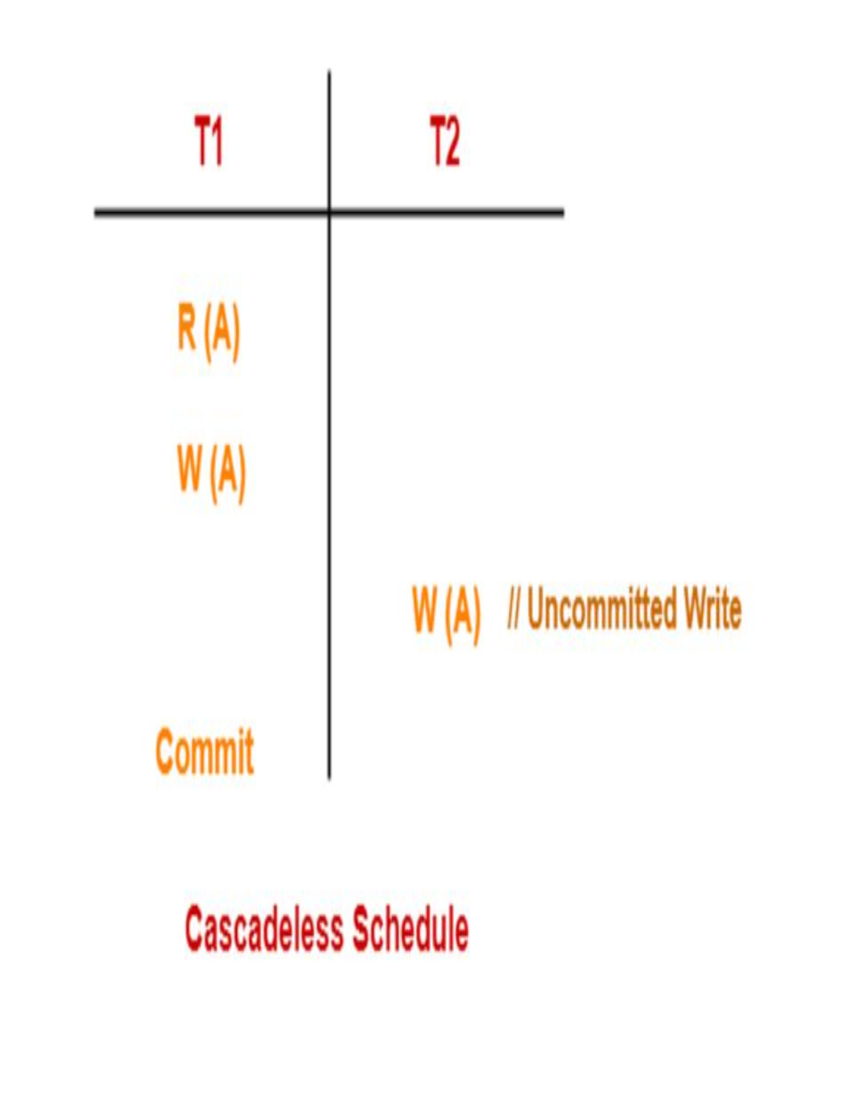
Cascadeless schedules (avoid cascading rollback)

- **Cascadeless schedules** — cascading rollbacks cannot occur; for each pair of transactions T_i and T_j such that T_j reads a data item previously written by T_i, the commit operation of T_i appears before the read operation of T_j.
- Every cascadeless schedule is also recoverable
- It is desirable to restrict the schedules to those that are cascadeless



T1	T2	T3
R (A)		
W (A)		
Commit		
	R (A)	
	W (A)	
	Commit	
		R (A)
		W (A)
		Commit

Cascadeless Schedule

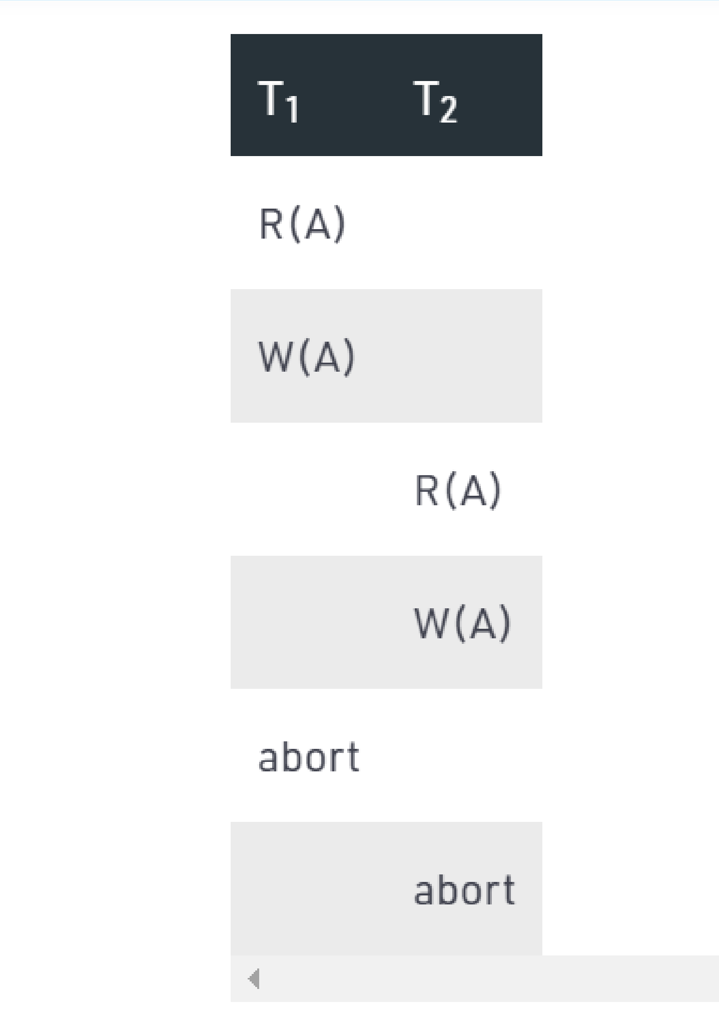


T1	T2
R (A)	
W (A)	
	W (A) // Uncommitted Write
Commit	

Cascadeless Schedule

Examples on cascadeless schedules

Example

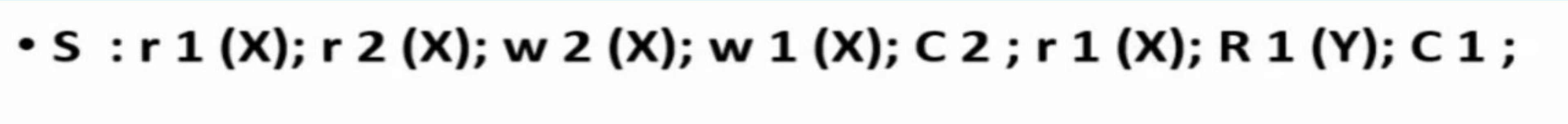


T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)
abort	
	abort

This schedule is recoverable schedule or cascadeless??

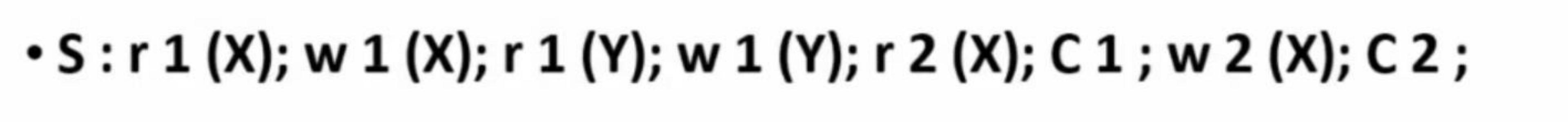
recoverable schedule (because no commit exist)
but not cascadeless

Examples



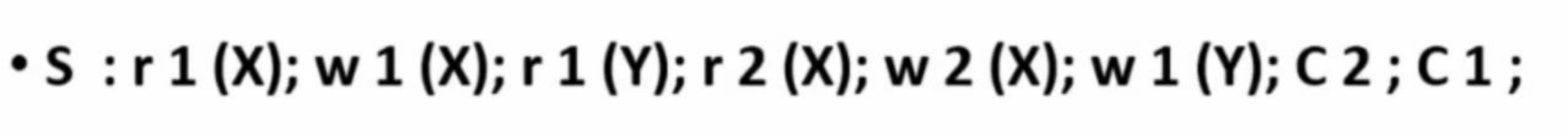
- S : r 1 (X); r 2 (X); w 2 (X); w 1 (X); C 2 ; r 1 (X); R 1 (Y); C 1 ;

Cascadeless (trivially recoverable)



- S : r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); r 2 (X); C 1 ; w 2 (X); C 2 ;

Recoverable (not cascadeless)



- S : r 1 (X); w 1 (X); r 1 (Y); r 2 (X); w 2 (X); w 1 (Y); C 2 ; C 1 ;

Non recoverable (sure non recoverable)

Strict Schedule

- T_j can read or write updated or written value of T_i only after T_i commits/aborts.

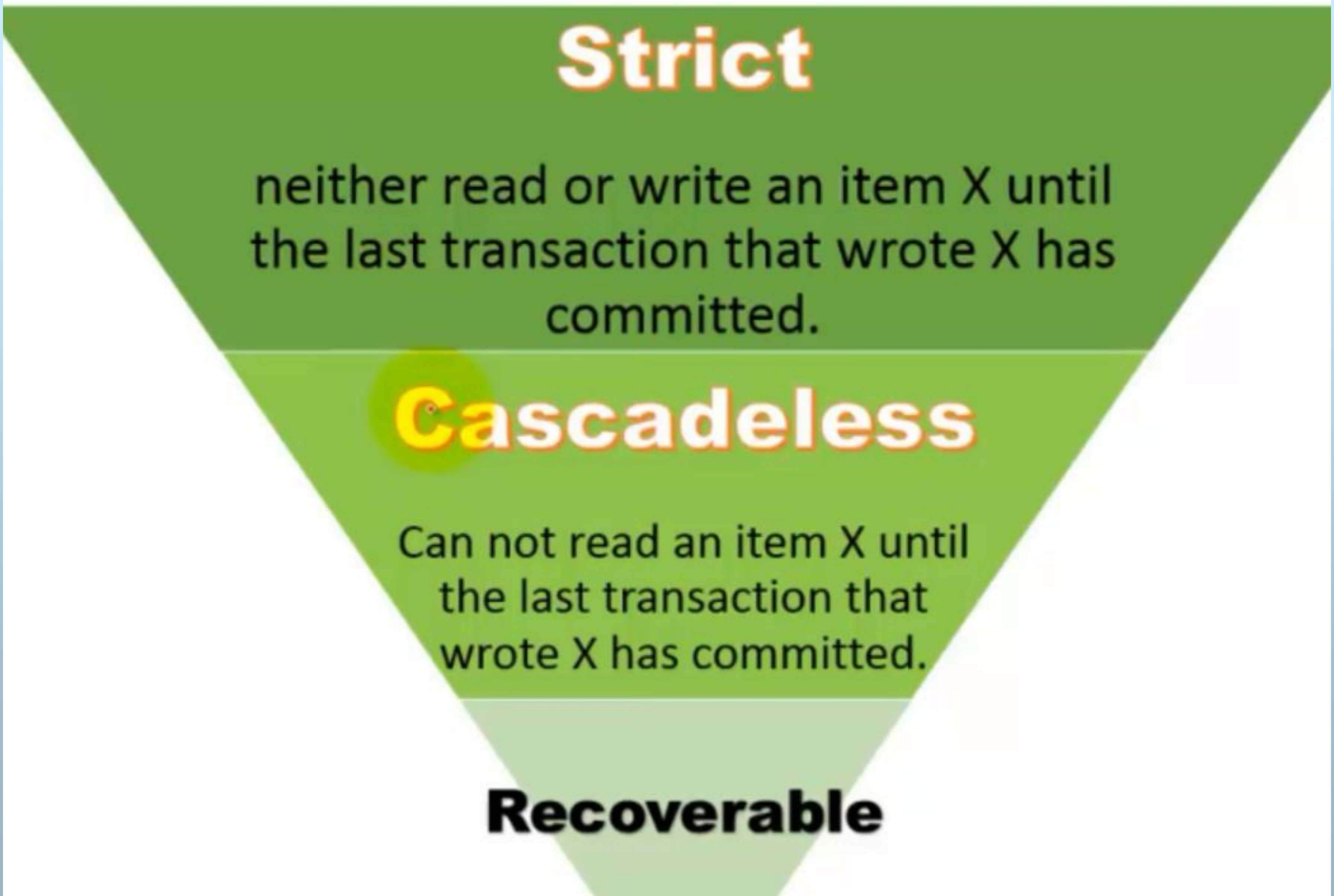
T_1	T_2
R(A)	
	R(A)
W(A)	
commit	
	W(A)
	R(A)
	commit

strict schedule

T_1	T_2
R(A)	
	R(A)
W(A)	
	W(A)
Commit	
	R(A)
	commit

Cascadeless schedule

Types of Schedules



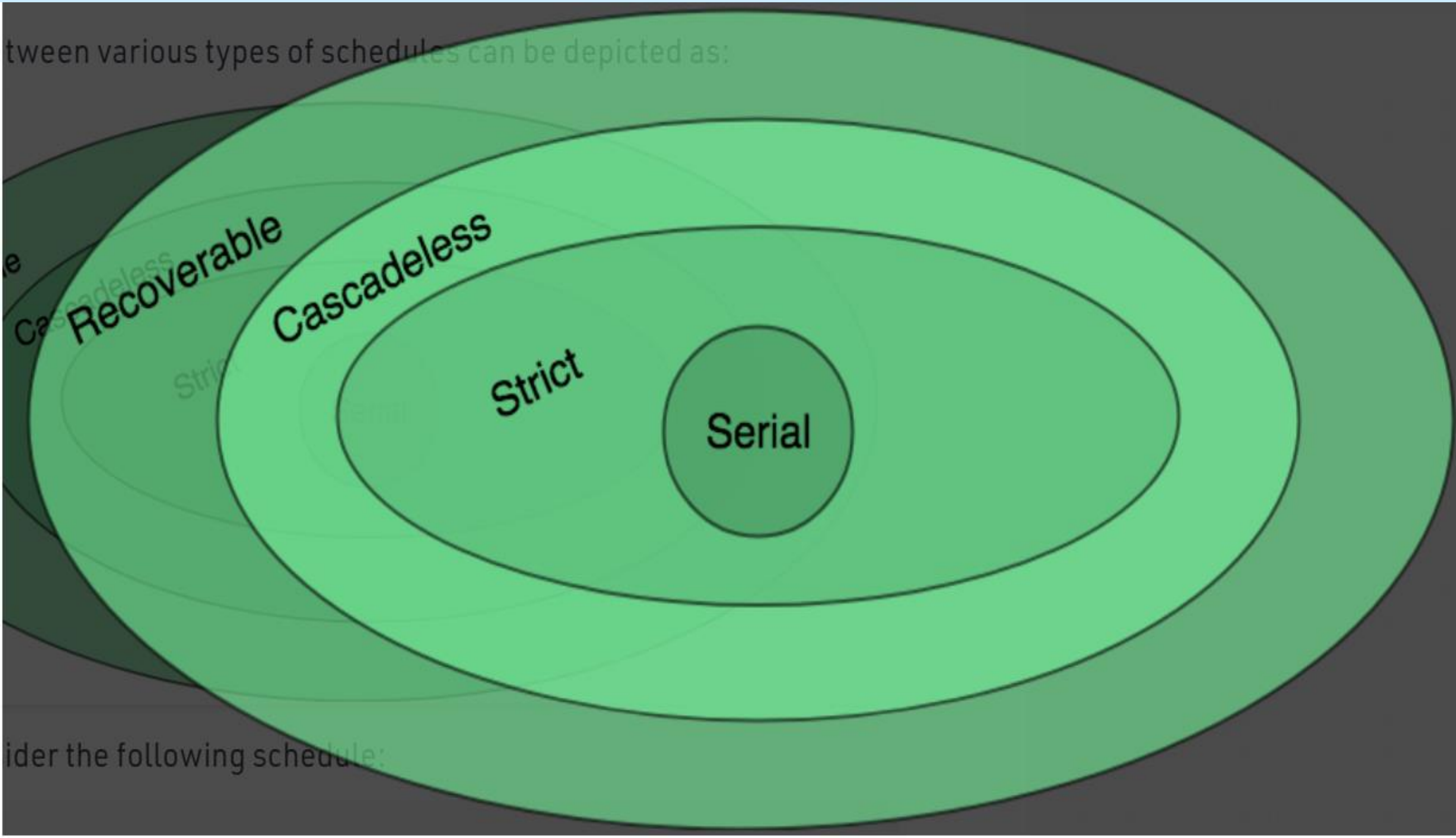
Strict

neither read or write an item X until the last transaction that wrote X has committed.

Cascadeless

Can not read an item X until the last transaction that wrote X has committed.

Recoverable

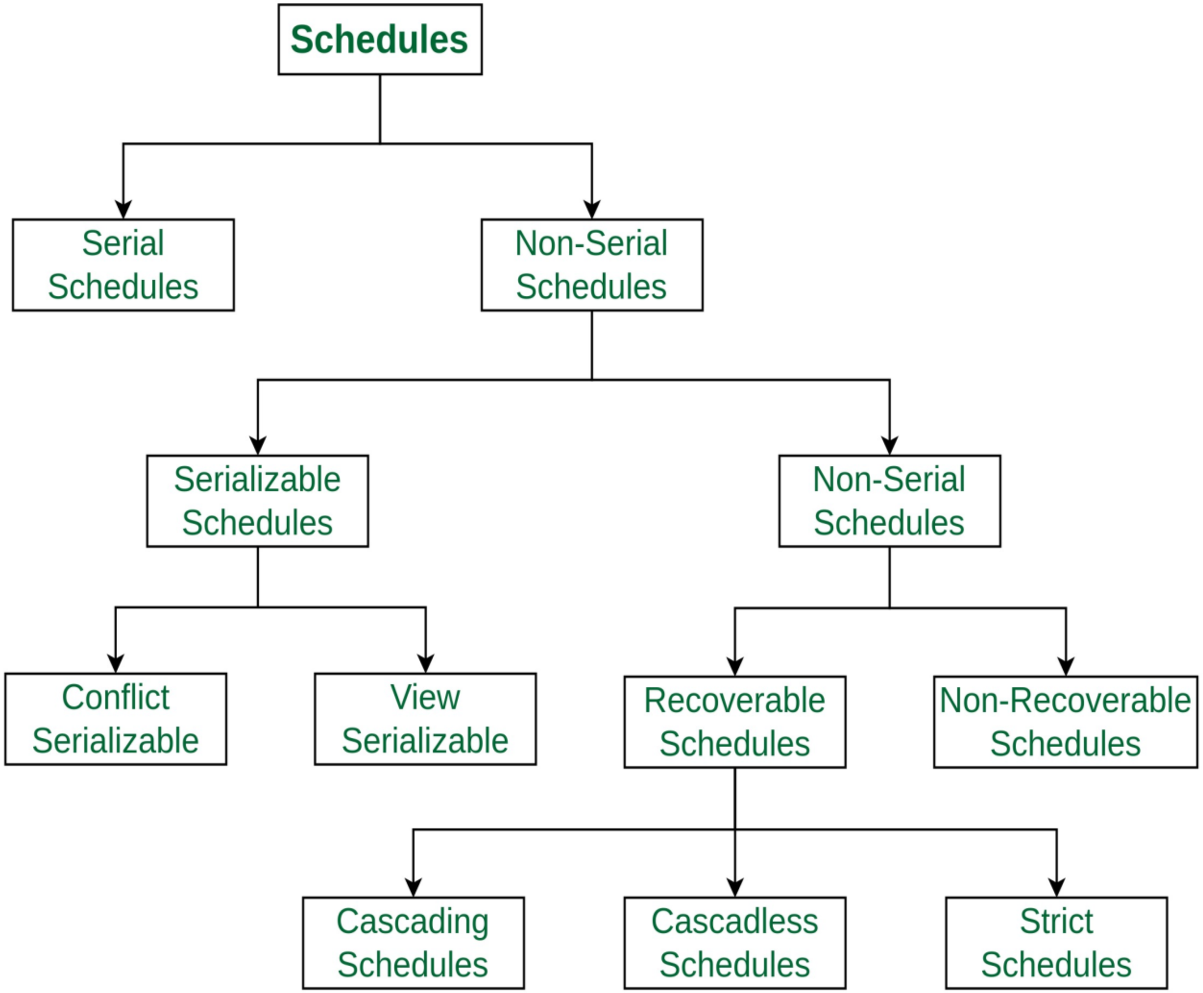


tween various types of schedules can be depicted as:

ider the following schedule:

The relation between various types of schedules

Types of schedules in DBMS



Schedules

Serial Schedules

Non-Serial Schedules

Serializable Schedules

Non-Serial Schedules

Conflict Serializable

View Serializable

Recoverable Schedules

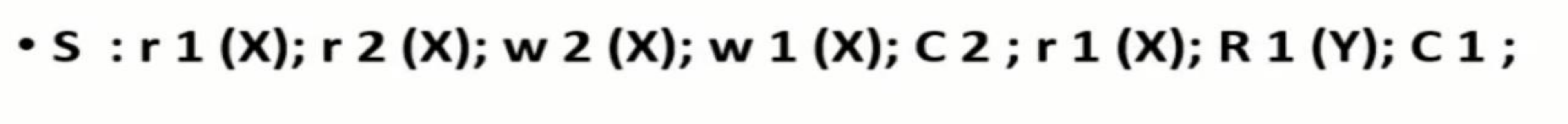
Non-Recoverable Schedules

Cascading Schedules

Cascadless Schedules

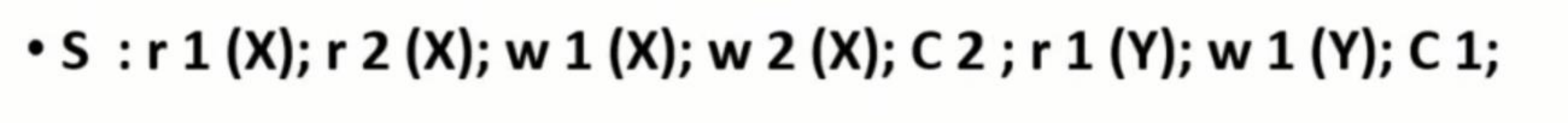
Strict Schedules

Examples Cont.

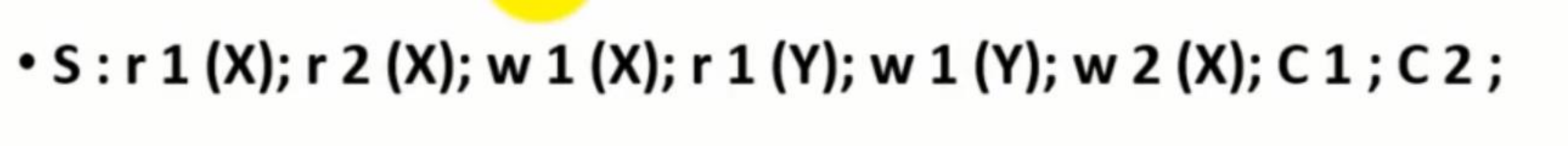


- S : r 1 (X); r 2 (X); w 2 (X); w 1 (X); C 2 ; r 1 (X); R 1 (Y); C 1 ;

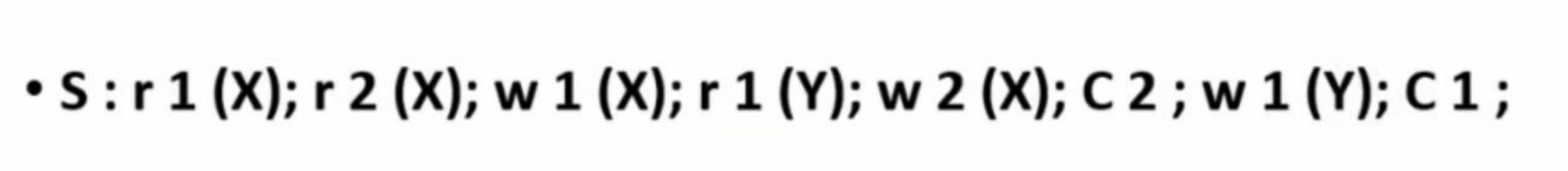
Cascadeless



- S : r 1 (X); r 2 (X); w 1 (X); w 2 (X); C 2 ; r 1 (Y); w 1 (Y); C 1;

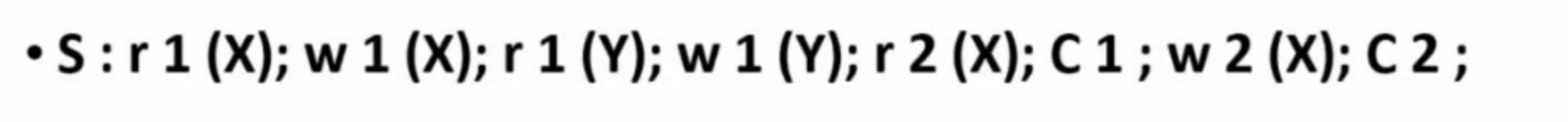


- S : r 1 (X); r 2 (X); w 1 (X); r 1 (Y); w 1 (Y); w 2 (X); C 1 ; C 2 ;



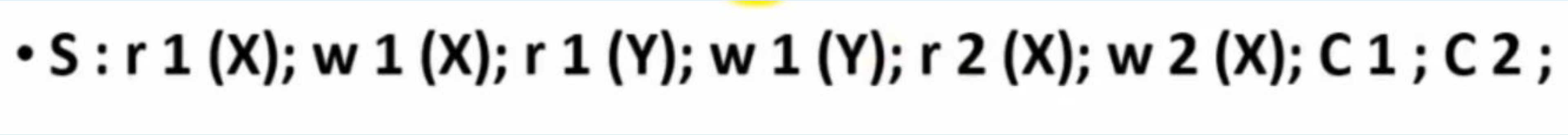
- S : r 1 (X); r 2 (X); w 1 (X); r 1 (Y); w 2 (X); C 2 ; w 1 (Y); C 1 ;

Examples Cont.

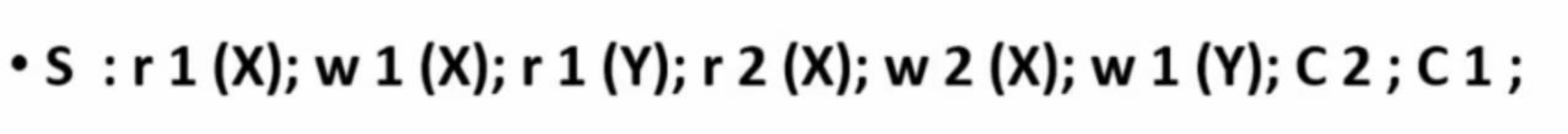


- S : r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); r 2 (X); C 1 ; w 2 (X); C 2 ;

recoverable

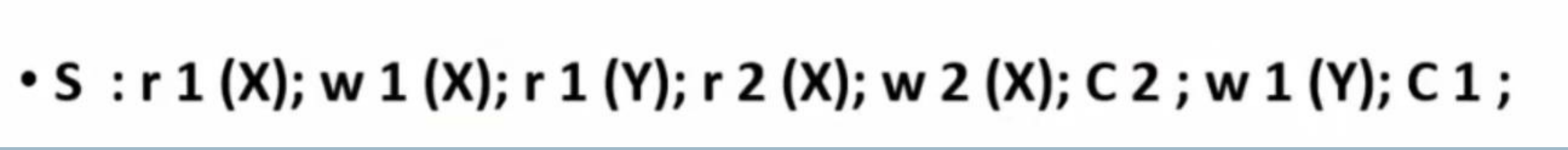


- S : r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); r 2 (X); w 2 (X); C 1 ; C 2 ;



- S : r 1 (X); w 1 (X); r 1 (Y); r 2 (X); w 2 (X); w 1 (Y); C 2 ; C 1 ;

Non recoverable



- S : r 1 (X); w 1 (X); r 1 (Y); r 2 (X); w 2 (X); C 2 ; w 1 (Y); C 1 ;