Transactions

Schedules

A **schedule** S of n transactions T_1, T_2, ..., T_n is an ordering of the operations of the transactions.

T_1

read_item(X);
$X := X - N$;
write_item(X);
read_item(Y);
$Y := Y + N$;
write_item(Y);

T_2

read_item(X);
$X := X + M$;
write_item(X);

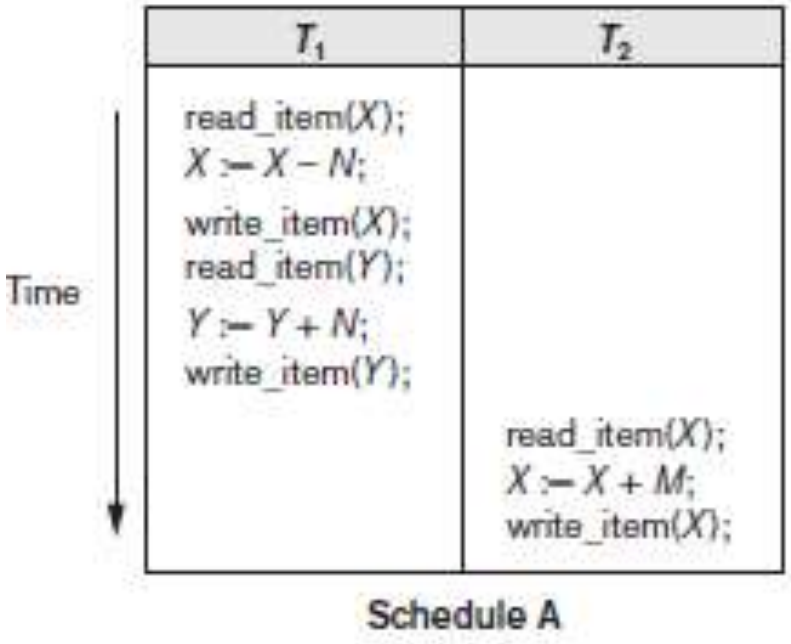
T_3

read_item(X);
$X := X - N$;
write_item(X);
read_item(Y);

Schedules

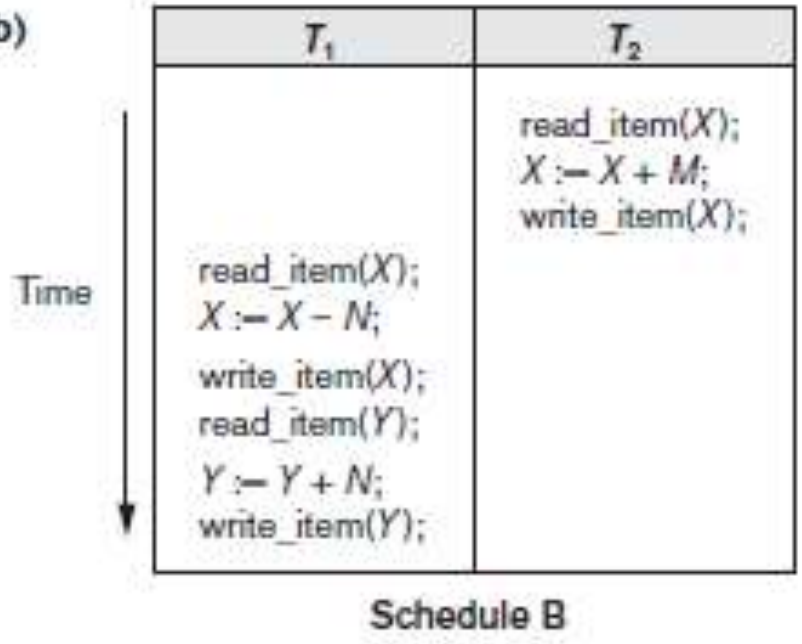
Serial schedule

- A schedule S is serial if, for every T in S, all the operations of T are executed consecutively in the schedule.



Schedule A

(b)



Schedule B

Schedules

Non-Serial Schedule

- A schedule S is non-serial if the operations from different transactions are **interleaved in *S*.**

T_1	T_2
read_item(X);	
$X := X - N$;	
	read_item(X);
	$X := X + M$;
write_item(X);	
read_item(Y);	
	write_item(X);
$Y := Y + N$;	
write_item(Y);	

Time ↓

The operations of each T_i in S must appear in the same order in which they occur in T_i.

$$S_a: r_1(X); r_2(X); w_1(X); r_1(Y); w_2(X); w_1(Y);$$

Schedules

Non-Serial Schedule

- A schedule S is non-serial if the operations from different transactions are **interleaved in *S*.**

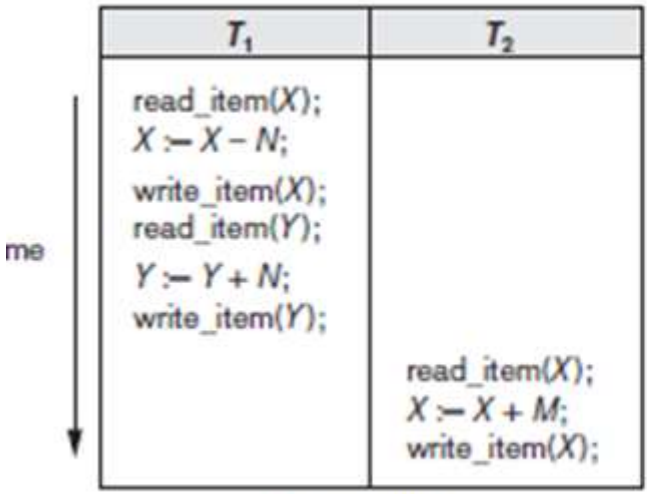
T_1	T_2
read_item(X);	
$X := X - N$;	
	read_item(X);
	$X := X + M$;
write_item(X);	
read_item(Y);	
	write_item(X);
$Y := Y + N$;	
write_item(Y);	

Time ↓

The operations of each T_i in S must appear in the same order in which they occur in T_i.

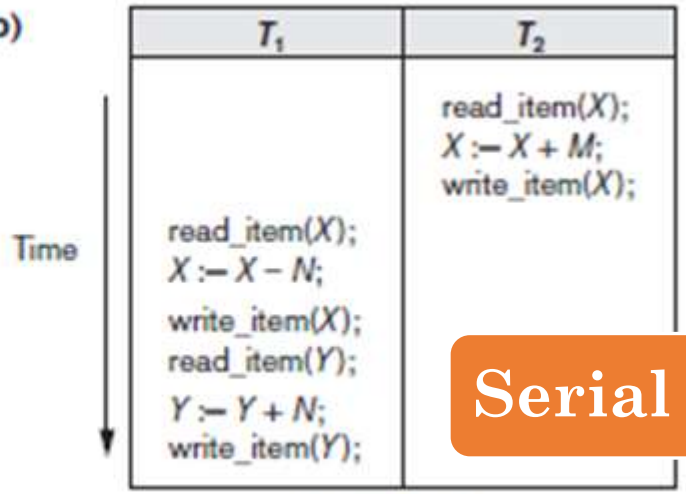
$$S_a: r_1(X); r_2(X); w_1(X); r_1(Y); w_2(X); w_1(Y);$$

Schedules Serializability



Schedule A

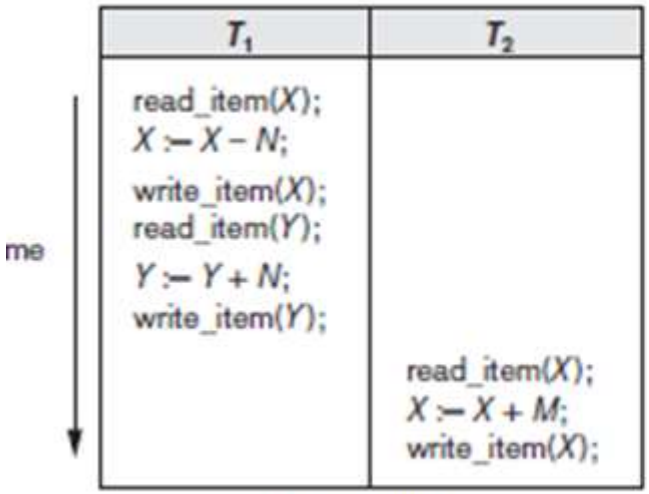
(b)



Schedule B

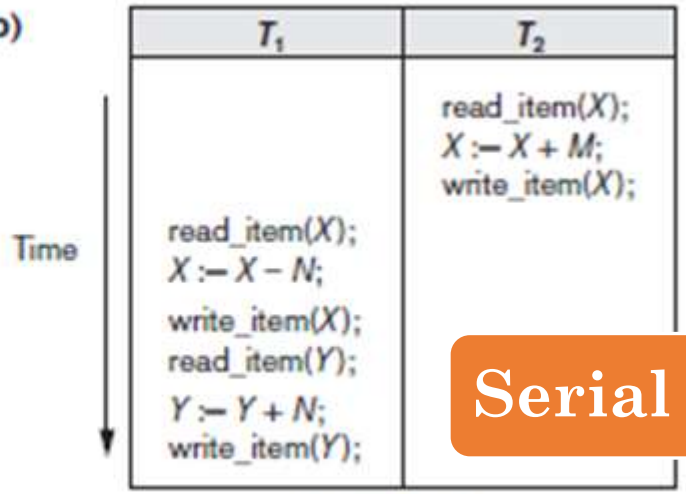
Serial schedule

Schedules Serializability



Schedule A

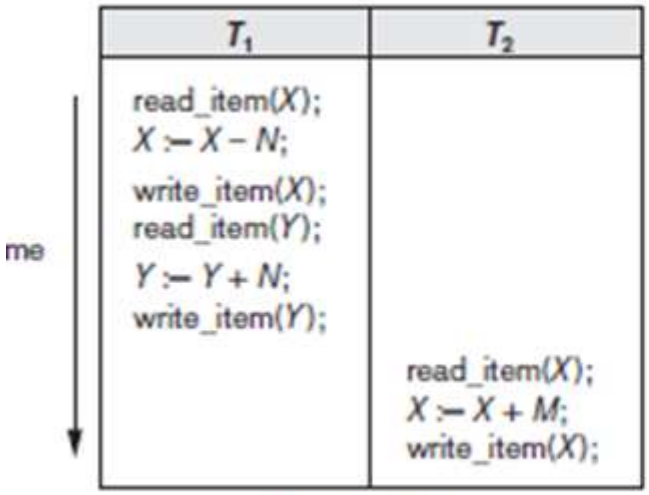
(b)



Schedule B

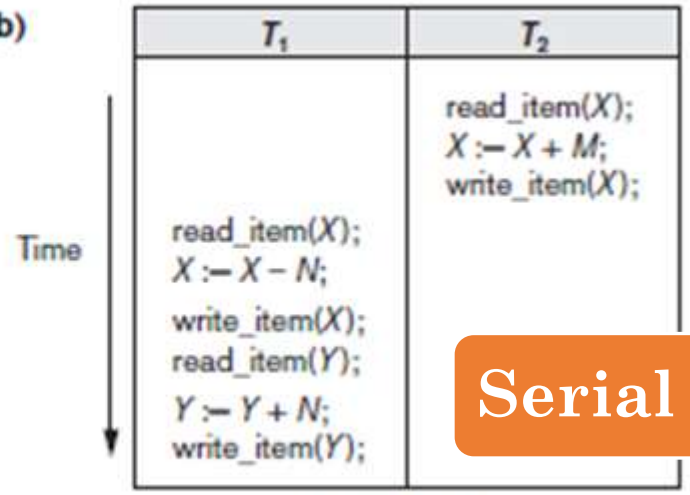
Serial schedule

Schedules Serializability



Schedule A

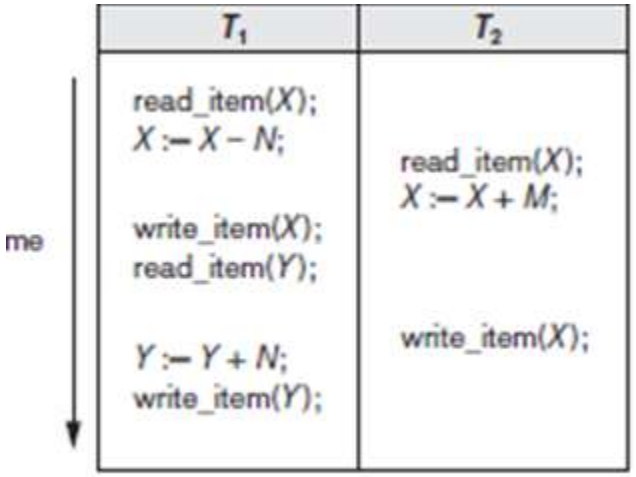
(b)



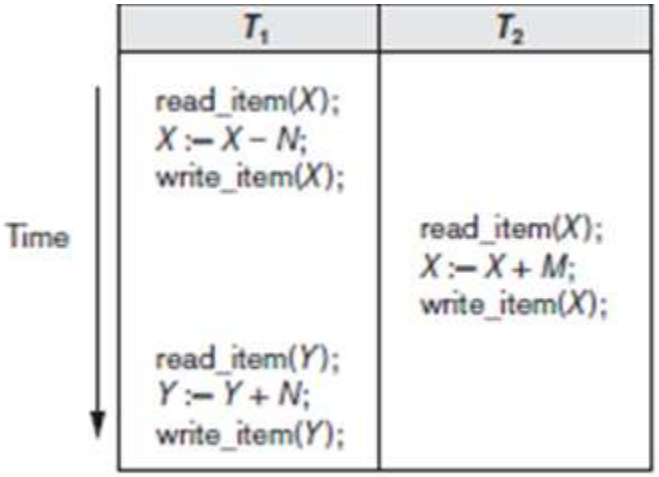
Schedule B

Serial schedule

Non-serial schedule



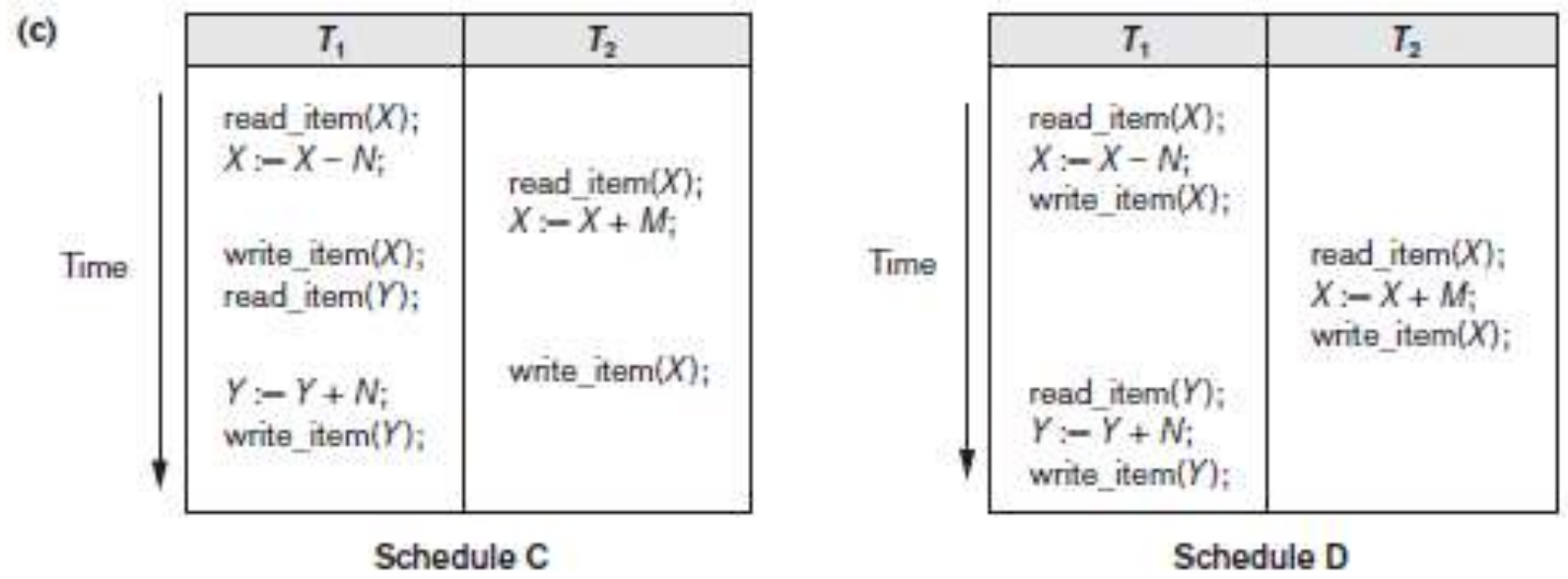
Schedule C



Schedule D

Characterizing Schedules based on Serializability

- **Non-Serial schedule**
  - **Which one is correct ?**



Schedule C gives an erroneous result because of the *lost update problem*
Schedule D gives correct results

Characterizing Schedules based on Recoverability

shutterstock.com • 1628971105

Recoverable Schedules

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

T1	T2
r(X)	
w(X)	
	r(X)
r(Y)	

Recoverable Schedules

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

Recoverable schedule:

- The following schedule is **not** recoverable **if**
  - T_2 **commits** immediately after the read

T1	T2
r(X)	
w(X)	
	r(X)
r(Y)	

  - If T_1 **abort**, T_2 have done a dirty read .

Recoverable Schedules

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

Recoverable schedule:

- if a transaction T_2 **reads** a data item *previously* **written** by T_1 .
- then the **commit** of T_1 should appears **before** the **commit** of T_2.

- The following schedule is **not** recoverable **if**
  - T_2 **commits** immediately after the read

T1	T2
r(X)	
w(X)	
	r(X)
r(Y)	

  - If T_1 **abort**, T_2 have done a dirty read .

Database must ensure that schedules are recoverable.

Characterizing Schedules based on Recoverability

In a recoverable schedule, no committed transaction ever needs to be rolled back

T1	T2
r(X)	
	r(X)
w(X)	
r(Y)	
	w(X)
	commit
w(Y)	
commit	

T1	T2
r(X)	
w(X)	
	r(X)
r(Y)	
	w(X)
	commit
abort	

Which one is recoverable ?

Cascading Rollbacks

- Consider the following schedule where:
  - **none** of the transactions has yet **committed**

T1	T2	T3
r(X)		
r(Y)		
w(X)		
	r(X)	
	w(X)	
		r(X)

Cascading Rollbacks

- Consider the following schedule where:
  - **none** of the transactions has yet **committed**
  - The schedule is **recoverable**

T1	T2	T3
r(X)		
r(Y)		
w(X)		
	r(X)	
	w(X)	
		r(X)

- If T_1 *fails*, T_2 and T_3 must also be *rolled back*.

Cascading Rollbacks

- **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**)

T1	T2	T3
r(X)		
r(Y)		
w(X)		
	r(X)	
	w(X)	
		r(X)

  - If T_1 *fails*, T_2 and T_3 must also be *rolled back*.

Can **lead** to *undo* of a *significant amount of work*

Characterizing Schedules based on Recoverability

Cascadeless schedule

- A schedule where every transaction reads only the items that are written by committed transactions.
- In other words: *No dirty Read*

Which schedule avoid cascade rollback

S1: r1(X) w1(X) r2(X) r1(Y) w2(X) w1(Y) c1 c2

S2: r1(X) w1(X) r2(X) r1(Y) w2(X) w1(Y) a1 a2

The $r_2(X)$ command in schedules *S1* and *S2* must be postponed until after T_1 has committed (or aborted),

This delays T_2 but ensuring no cascading rollback if T_1 aborts

Characterizing Schedules based on Recoverability

Sf: $w_1(X,5)$ $w_2(X,8)$ a_1

- *Sf* is cascadeless, but it can still have an issue !!!
  - It permits T_2 to write item X even though T_1 that last wrote X had not yet committed (or aborted).

Characterizing Schedules based on Recoverability

Sf: $w_1(X,5)$ $w_2(X,8)$ a_1

- *Sf* is cascadeless, but it can still have an issue !!!
  - It permits T_2 to write item X even though T_1 that last wrote X had not yet committed (or aborted).

If we have to undo a **write_item(*X*)** of an aborted transaction then we restore the **before image** (old_value) of *X*.

Characterizing Schedules based on Recoverability

Strict Schedule

A schedule in which a transaction can neither read or write an item X until the last transaction that wrote X has committed.

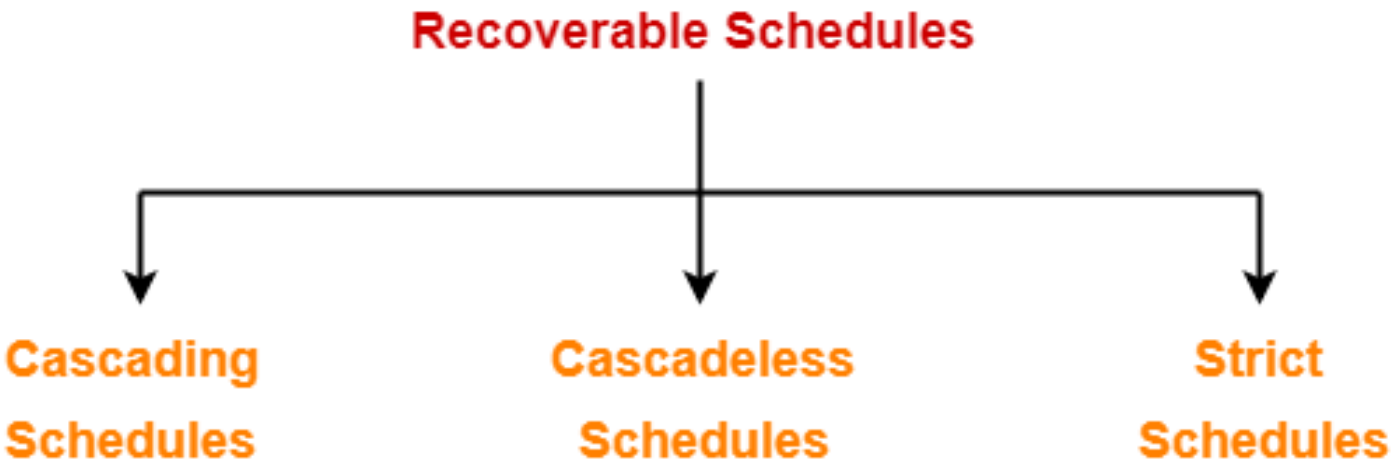
Sf: $w_1(X,5)$ $w_2(X,8)$ a_1

Strict schedules simplify the recovery process.

Characterizing Schedules based on Recoverability

Non-Recoverable schedule	• A transaction T_2 **reads** a data item **X** written by T_1 in S and commit before T_1
Recoverable schedule	• If a transaction T_2 **reads** a data item **X** written by T_1 in S and commit after T_1
Cascadeless schedule	• No dirty read
Strict Schedule	• A schedule in which a transaction can neither read or write an item X until the last transaction that wrote X has committed.

Characterizing Schedules based on Recoverability



Characterizing Schedules based on Recoverability

Every **strict** schedule is also **cascadeless**

Every **cascadeless** schedule is also **recoverable**

It is **desirable** to **restrict** the schedules to those that are **cascadeless**

Characterizing Schedules based on Recoverability