

COSC344

Database Theory and Applications



Lecture 21 Transactions

Lecture 21 Transactions - Overview

- This Lecture
 - Transactions
 - **Source: Chapter 20**
- Next Lecture
 - Concurrency control
 - Source: Chapter 21
- Lecture After
 - Recovery
 - Source: Chapter 22

DBMS Component Modules

Where are we now?

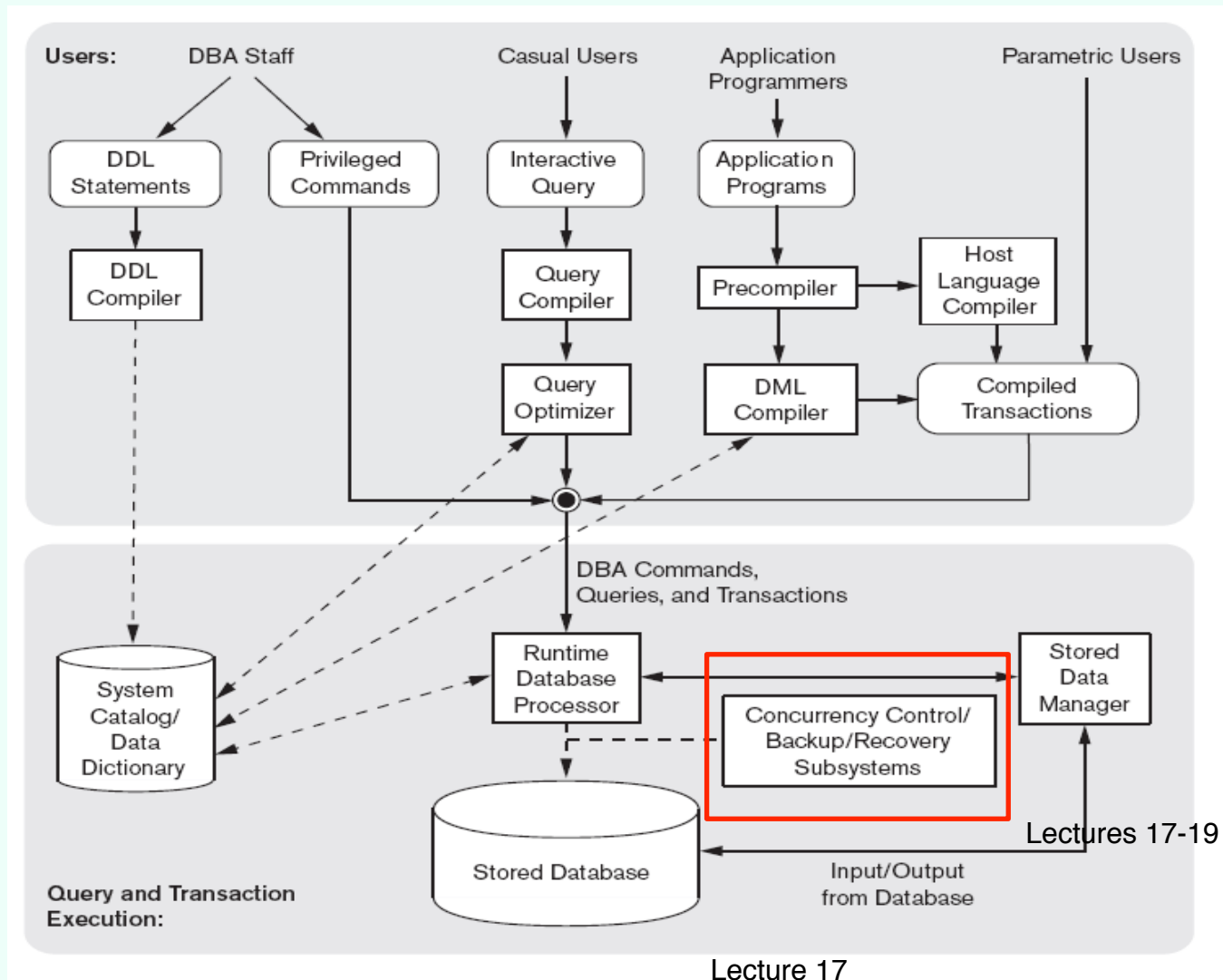


Figure 2.3
Component modules of a DBMS and their interactions.

Lecture 21 Transactions - Overview




Lecture 21 Transactions - Overview

Available rooms

Available rooms from **Sun, 31 May 2009** to **Mon, 01 Jun 2009**.



 **5 x Double bedded**

Persons 

Price **45.00** to **70.00** EUR


For the weekends the prices are modified by 50%

[More...](#)

82.50 EUR


Price for 1 night

Quantity ▼

 [Availability calendar](#)



 **10 x Single room**

Persons 

Price **40.00** to **45.00** EUR

[More...](#)

40.00 EUR

Price for 1 night

Quantity ▼

 [Availability calendar](#)

[Reserve >](#)

Lecture 21 Transactions - Overview

American Airlines | TYO to SFO | \$4530

willmoyer.com

SELECT		Average Price per Person - 4530.70 USD									
Carrier	Flight #	Departing		Arriving		Aircraft Type	Cabin	AA Flight Miles	Meals	Travel Time	
		City	Date & Time	City	Date & Time						
 AMERICAN AIRLINES OPERATED BY JAPAN AIRLINES	5816	HND Tokyo	Sep 01, 2012 12:05 AM	SFO San Francisco	Aug 31, 2012 05:40 PM	777	Economy		Meal Meal	9 hr 35 min	
Alert: Overnight flight or connection.											

American Airlines | TYO to SFO and JFK to HND | \$1449

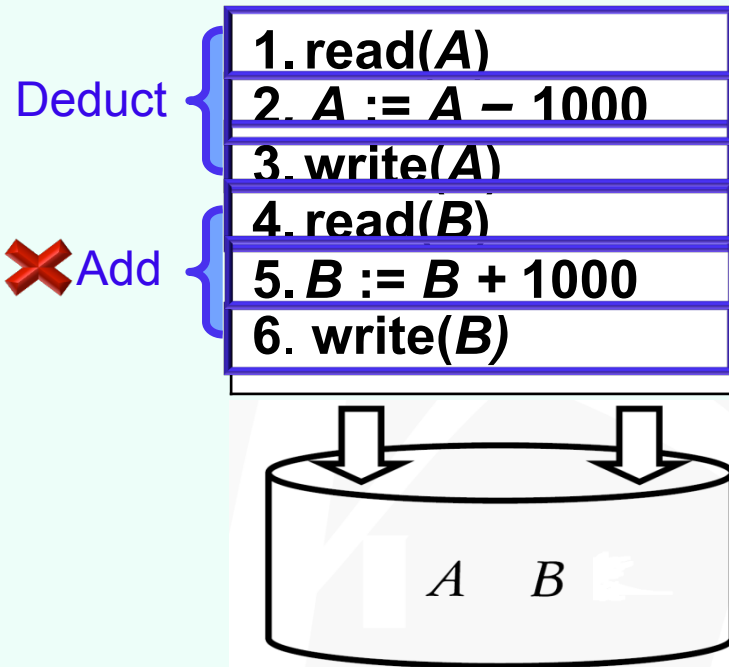
willmoyer.com

SELECT		Average Price per Person - 1449.40 USD									
Fare Alert: Nonrefundable fare.											
Carrier	Flight #	Departing		Arriving		Aircraft Type	Cabin	AA Flight Miles	Meals	Travel Time	
		City	Date & Time	City	Date & Time						
 AMERICAN AIRLINES OPERATED BY JAPAN AIRLINES	5816	HND Tokyo	Sep 01, 2012 12:05 AM	SFO San Francisco	Aug 31, 2012 05:40 PM	777	Economy		Meal Meal	9 hr 35 min	
Alert: Overnight flight or connection.											
 AMERICAN AIRLINES	135	JFK New York	Oct 01, 2012 07:10 PM	HND Tokyo	Oct 02, 2012 10:15 PM	777	Economy View Seats	6754	Lunch Dinner	14 hr 5 min	

A Money Transfer Example



- **Example:** Jane wants to transfer \$1000 from her checking account (A) to saving account (B)



	A	B
Original	1000	0
After 3	0	0
After 6	0	1000

- **Failures:**
 - System crash
 - Power failure
 - Software errors
 - User mistake ...

- **Question 1:** What happens if it fails after step 3 before 6.

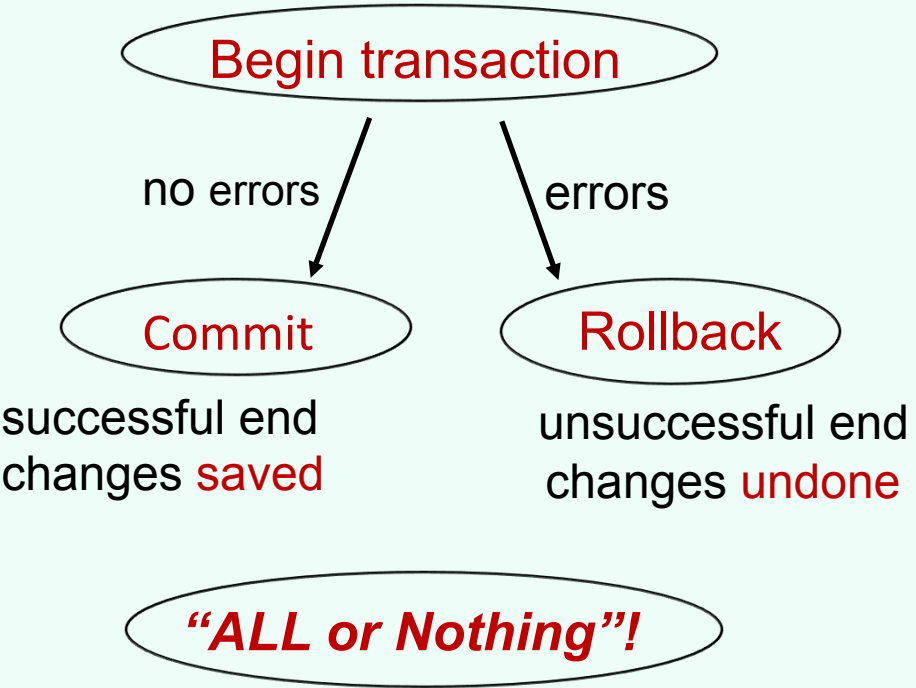


What is a Transaction?

- A transaction is a *sequence of operations* on a database, which is either *completely executed* or *not executed at all*

Transaction
BEGIN TRANSACTION
1. read (A)
2. $A := A - 1000$
3. write (A)
4. read (B)
5. $B := B + 1000$
6. write (B)
COMMIT TRANSACTION

ROLLBACK ↑
errors



Sample Transaction

All or Nothing



```
begin transaction
```

```
    UPDATE employee  
        SET salary = 1.1 * salary;
```

```
    UPDATE tot_sal  
        SET sal =  
            (SELECT SUM(salary)  
             FROM employee);
```

```
commit transaction
```

Transaction Processing

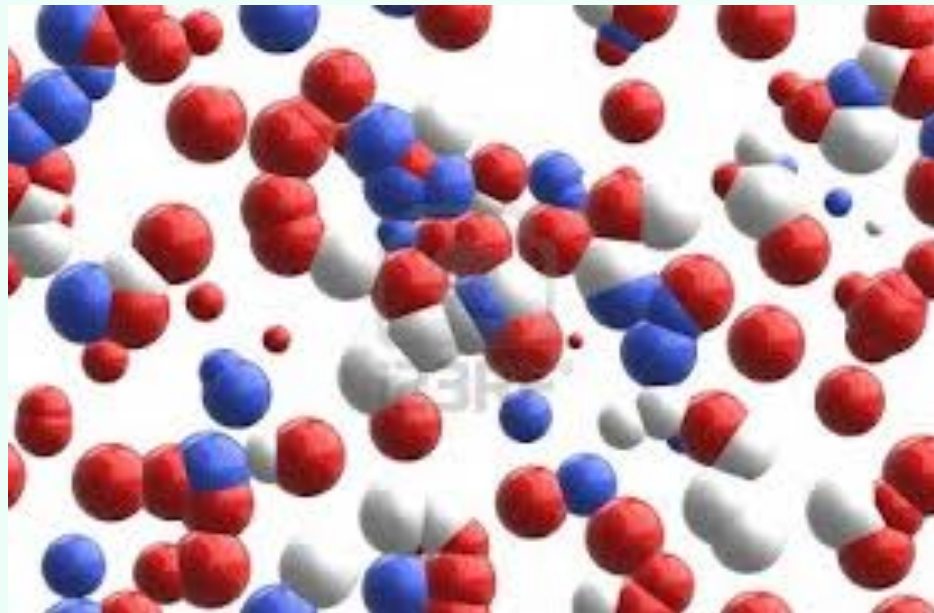
- Transaction processing guarantees that if a failure occurs before a transaction completes, then those updates will be undone.
- Either
 - Executes in its entirety
 - Is totally cancelled
- Transaction commit
 - Logical unit of work successfully completed
 - Database is or should be in a consistent state
 - All updates can be made permanent
- Transaction rollback
 - Something has gone wrong
 - Database may be in an inconsistent state
 - All updates must be undone or rolled back



ACID Properties

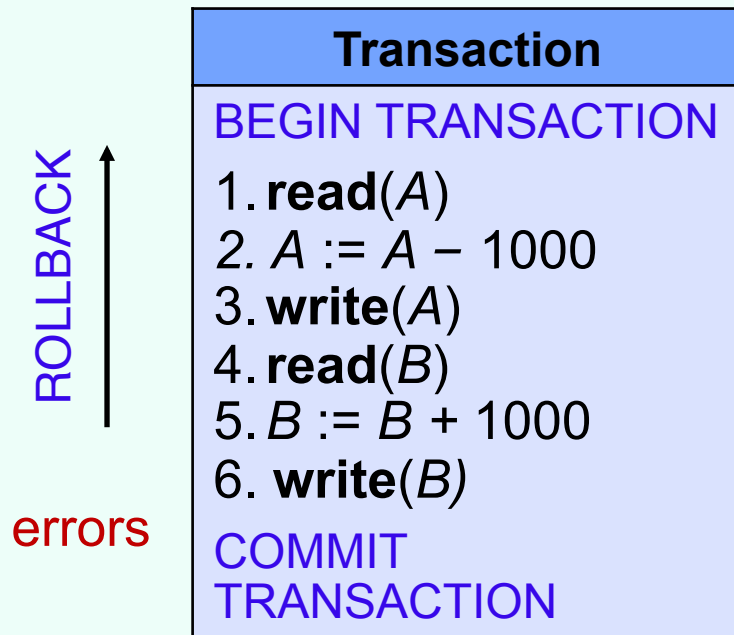
Atomicity, Consistency, Isolation, Durability

- Property 1: Atomicity



ACID Properties

Atomicity, Consistency, Isolation, Durability



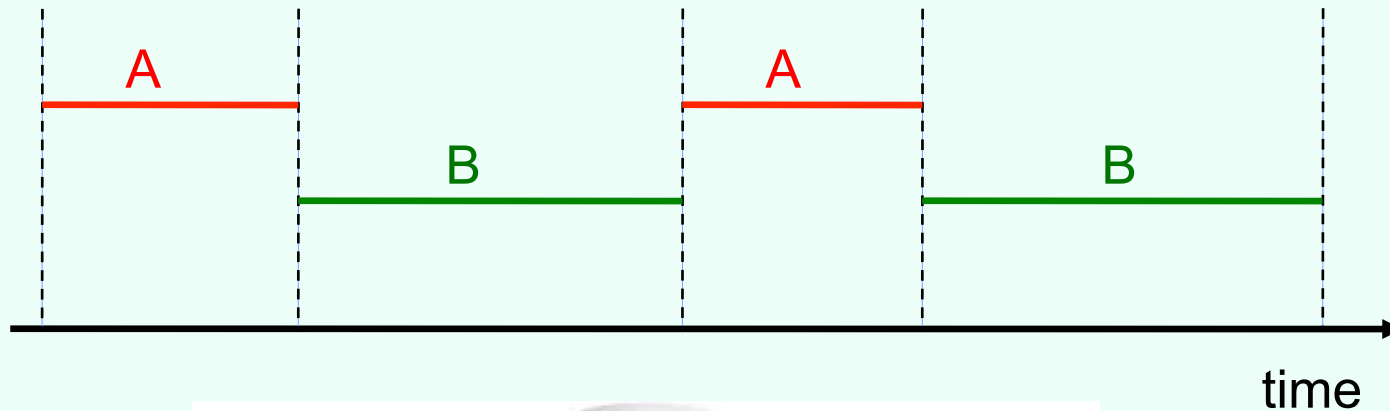
- Property 1: **Atomicity**
- A transaction is atomic
- Either all operations in the transaction have to be performed or none should be performed.
- Logical unit of work/recovery
- Failure recovery

“Atomicity” ensures “no money is taken from A without being given to B”

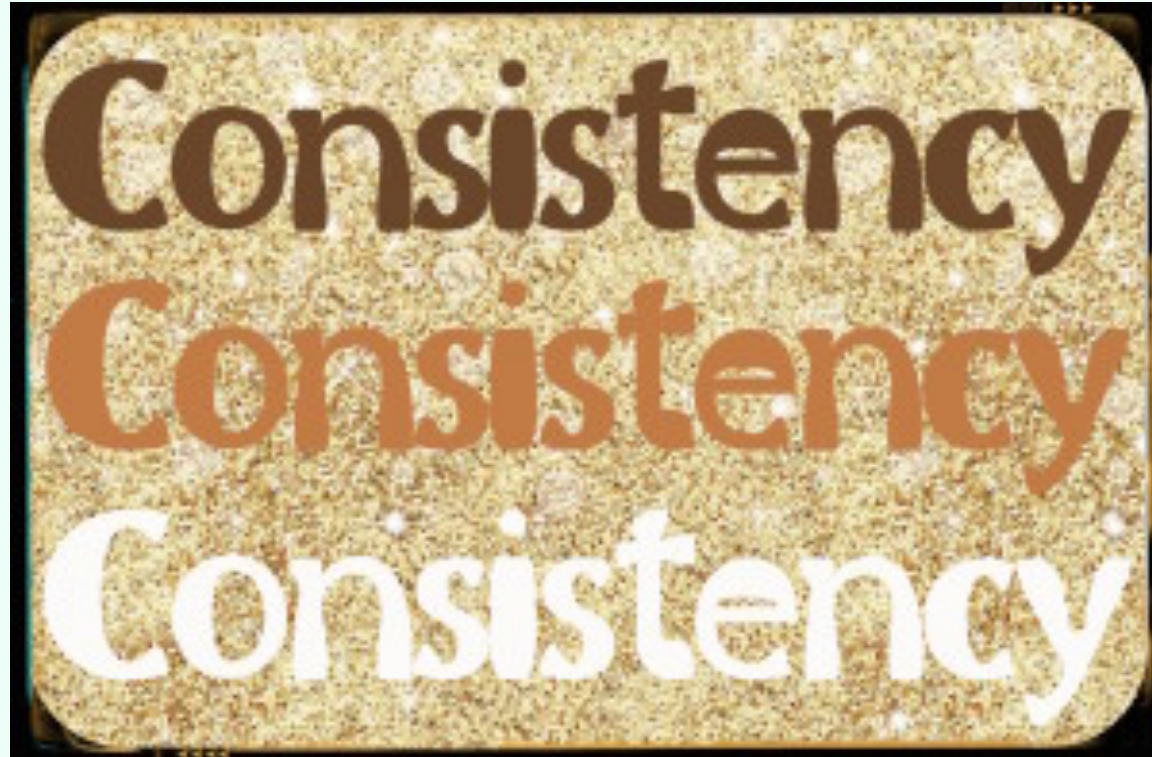


What Can Happen?

- What can go wrong if we allow multiple users simultaneous access to a database?
- Interleaving of operations



ACID Properties (Property 2: Consistency)



ACID Properties (Consistency)



• *Question 2 : What happens if transactions are executed simultaneously?*

- *T1 : \$1000 transfer from A to B*
- *T2 : \$2000 salary payment to A*

T1	T2
1. read(A) 1000	1. read(A) 1000
2. A := A - 1000	2. A := A + 2000
3. write(A) 0	3. write(A) 3000
4. read(B)	<i>Salary is lost!</i>
5. B := B + 1000	<i>update by T2 is overwritten by T1</i>
6. write(B)	<i>"Lost update problem!"</i>
	<i>Database is not consistent!</i>

	A	B
Original	1000	0
After T2. 3	3000	0
After T1. 6	0	1000

- *Property 2: **Consistency***
- A transaction transforms a consistent state of database into another consistent state
- Concurrency Control (next lecture)

"Consistency" makes sure that "money isn't lost or gained"



ACID Properties (Property 3: Isolation)



ACID Properties (Isolation)

Question 3 : What happens if the result of T1 is visible to T2 before T1 is completed?

- T1 : \$1000 transfer from A to B
- T2 : \$2000 salary payment to A

T1	T2
1. read(A) 1000	<i>Dirty Read Problem</i> T1: change A T2: read changed A T1: rollback T2: has read an invalid A
2. A := A - 1000	
3. write(A) 0	
4. read(B)	1. read(A) 0
5. B := B + 1000	2. A := A + 2000
6. write(B)	3. write(A) <i>Not valid</i>

ROLLBACK
↑
errors

	A	B
Original	1000	0
After T1. 3	0	0
T1	1000	0
rollback		

- Property 3: **Isolation**
 - Transactions are isolated from one another.
 - A transaction's updates are not visible to other transactions until it commits (or rollbacks).
 - Concurrency Control (next lecture)

"Isolation" enforces that "T2 can't see A change until T1 is completed"

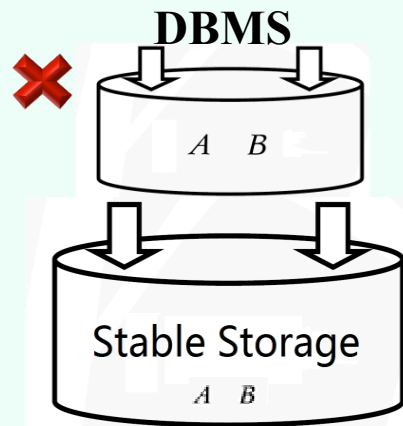
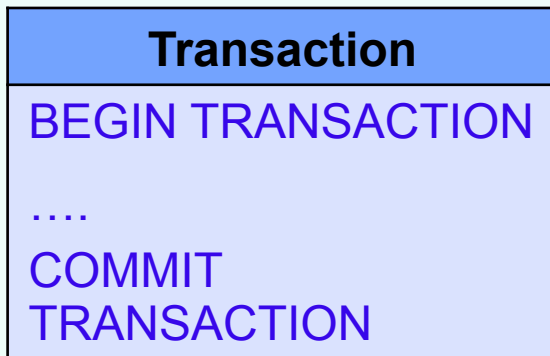


ACID Properties (Property 4: Durability)



ACID Properties (Durability)

- **Question 4:** What happens if database server crashes before the changed data is written to stable storage?



- A and B have been updated in the database – **Transferred!?**
- A and B have not been updated in stable storage – **Not Transferred!?**
- Property 4: **Durability**
 - once the user is notified that the transaction has completed, updates to the database by the transaction must be persistent despite failures.
 - Committed changes will not be reversed
 - Failure Recovery

“Durability” guarantees **“Once completed, money does not go back to A”**



Desirable Properties of Transactions

(ACID Properties)

- **Atomic** - performed in its entirety or not at all
- **Consistent** - a transaction transforms a consistent state of the database into another consistent state, *without necessarily preserving consistency at all intermediate points*
- **Isolated** - transactions are isolated from one another. The execution of a transaction should not be interfered with by any other transaction executing concurrently
- **Durable** - permanent. Once a transaction commits, its updates survive, even if there is a subsequent system crash
- **What are transactions used for?**
 - Banking Systems, Airline Reservation Systems, Hotel Systems
- **Two main issues to deal with:**
 - Failures (**A & D**): software failures and hardware failures
 - Concurrency (**C & I**): simultaneous execution of multiple transactions

Why Concurrency Control is Needed

- 3 Concurrency Problems
 - Lost update problem
 - Temporary update problem (uncommitted dependency)
 - Incorrect summary problem
- We assume
 - that operations of the two transactions can be interleaved by the operating system.
 - that the same database items are being accessed.



Two Example Transactions

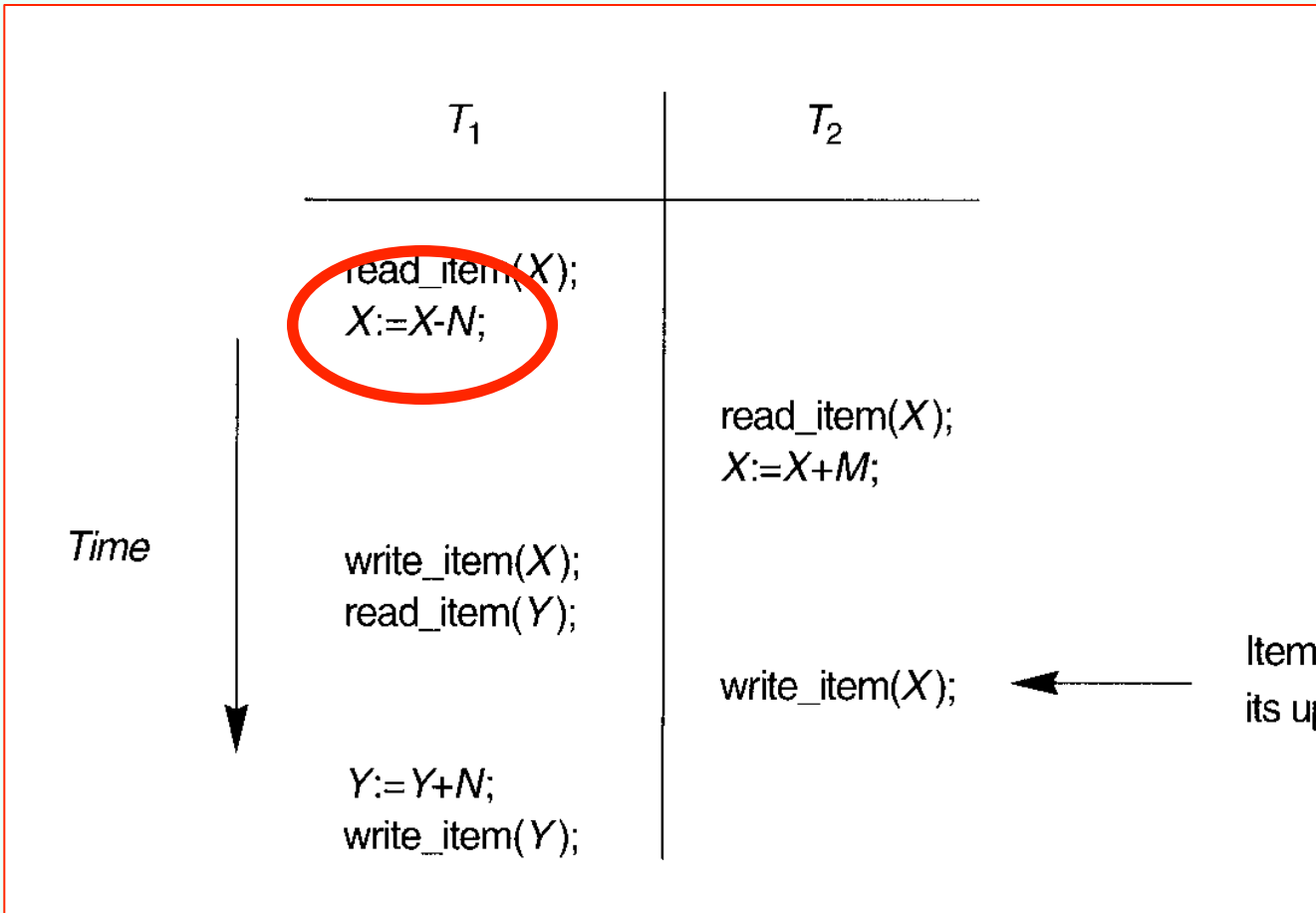
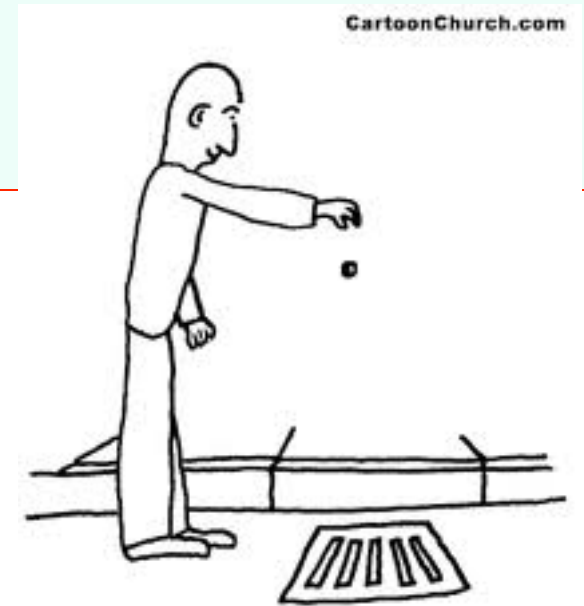
(a)	T_1	(b)	T_2
	<hr/>		<hr/>
	read_item (X);		read_item (X);
	X:=X-N;		X:=X+M;
	write_item (X);		write_item (X);
	read_item (Y);		
	Y:=Y+N;		
	write_item (Y);		

Note that read_item and write_item refer to reading and writing data to the database files

Figure 19.2 Two sample transactions. (a) Transaction T_1 . (b) Transaction T_2 .

If the transactions are run serially (NOT interleaved) then both answers are correct.

Lost Update Problem

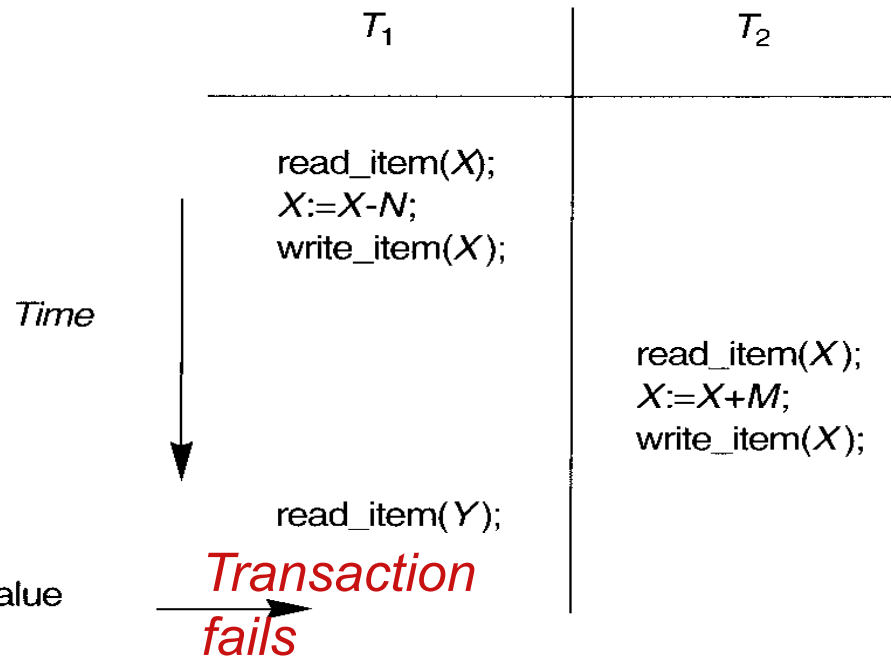


Item X has an incorrect value because its update by T_1 is "lost" (overwritten)

Elmasri & Navathe, 5th Ed, p614

Temporary Update Problem

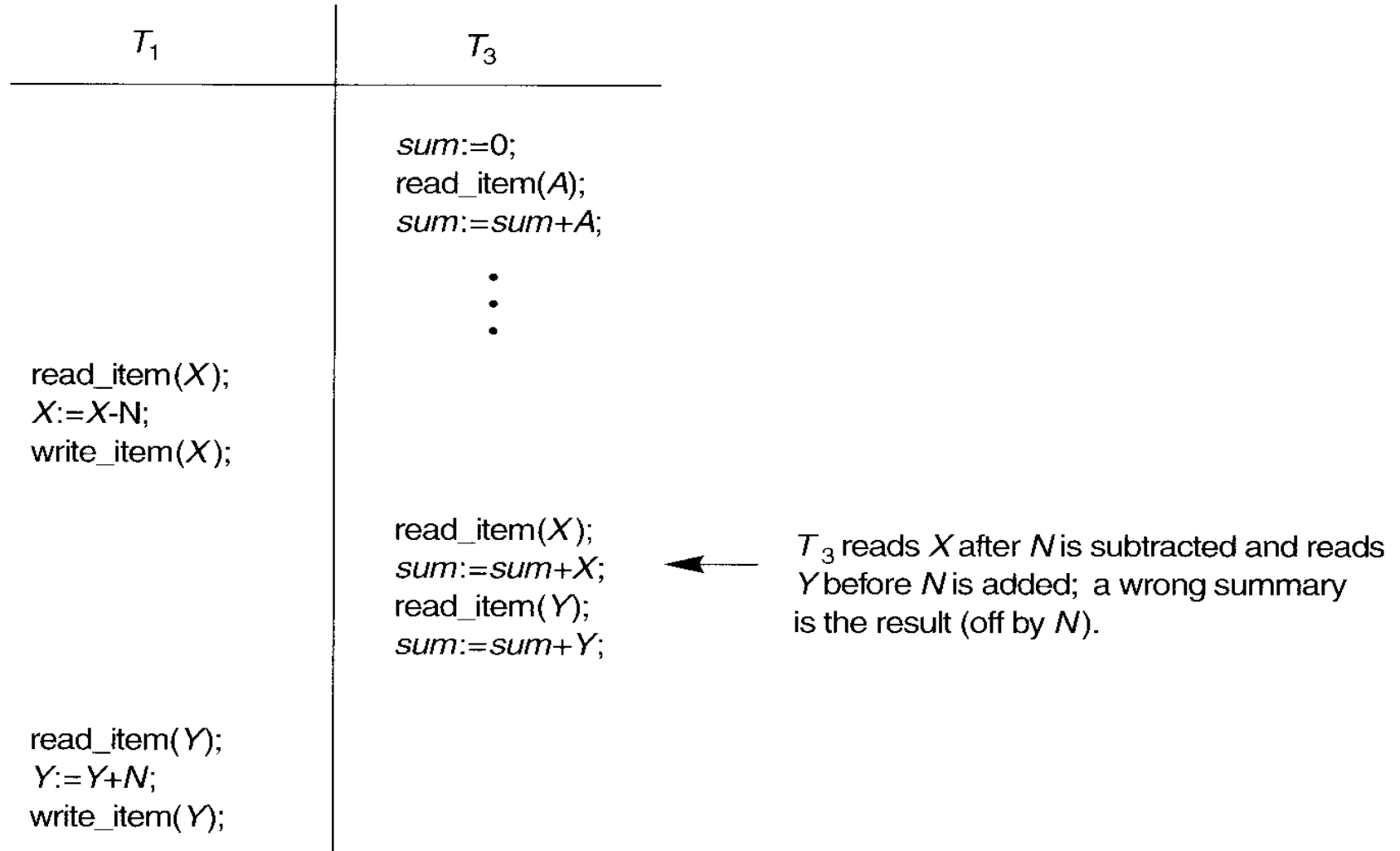
- Often called Uncommitted Dependency problem or Dirty Read



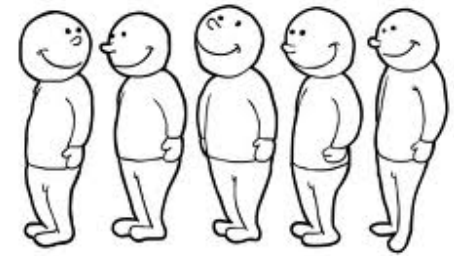
Transaction T_1 fails and must change the value of X back to its old value; meanwhile T_2 has read the "temporary" incorrect value of X .

Elmasri & Navathe, 5th Ed, p614

Incorrect Summary Problem

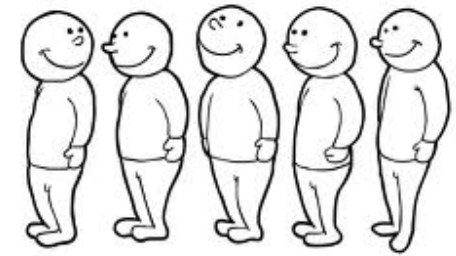


Schedules of Transactions

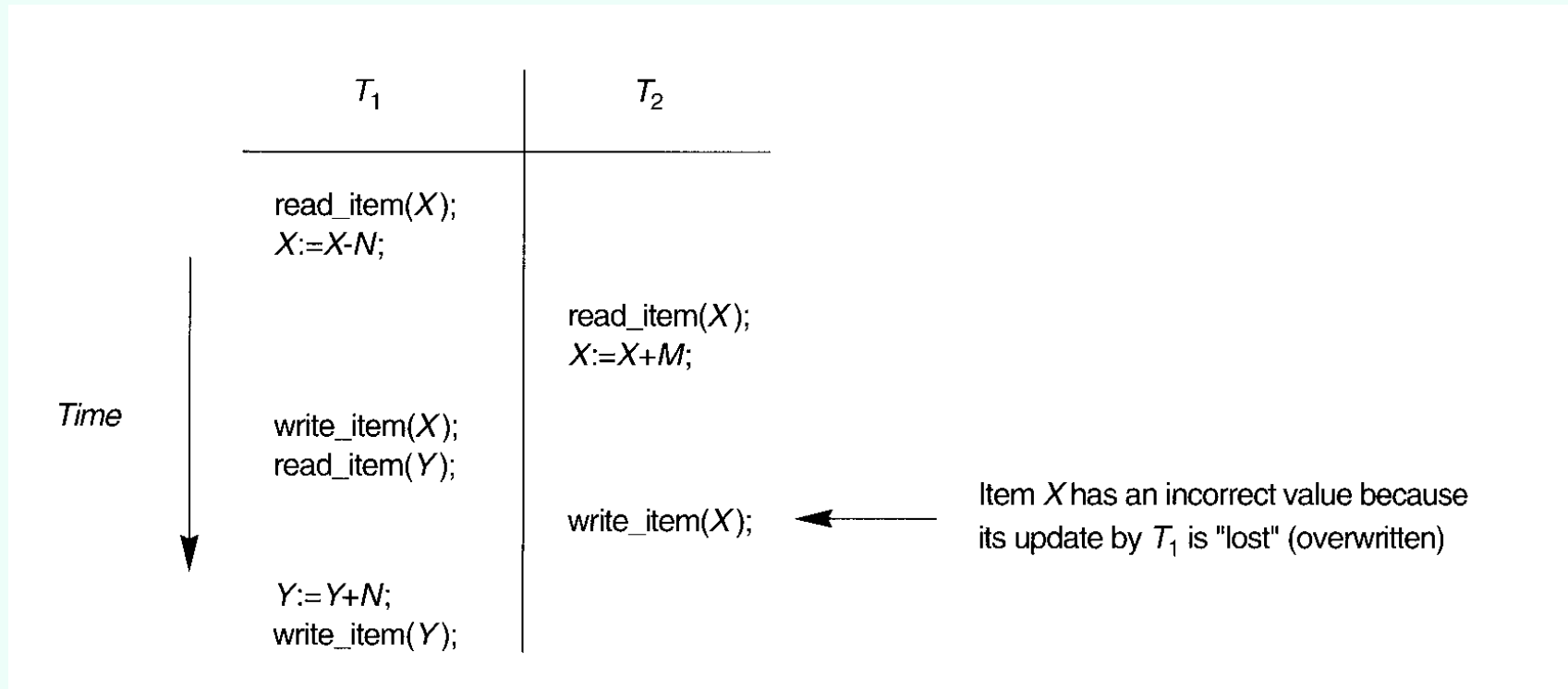


- Definition
 - A schedule S of transactions (T_1, T_2, \dots, T_N) is an ordering of the operations of the transactions subject to the constraint that the operations of each transaction must appear in the same order within the schedule that they appeared in the transaction
- Notations
 - r, w, c, a (read_item, write_item, commit, abort)
 - $r_1(X)$ (transaction 1 reads X)
- Example on Slide 23 can be written
 - $S_a: r_1(X), r_2(X), w_1(X), r_1(Y), w_2(X), w_1(Y)$
- Example on Slide 24 can be written
 - $S_b: r_1(X), w_1(X), r_2(X), w_2(X), r_1(Y), a_1$

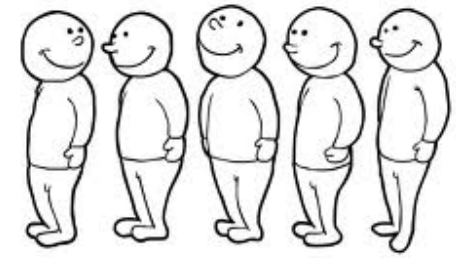
Schedules of Transactions



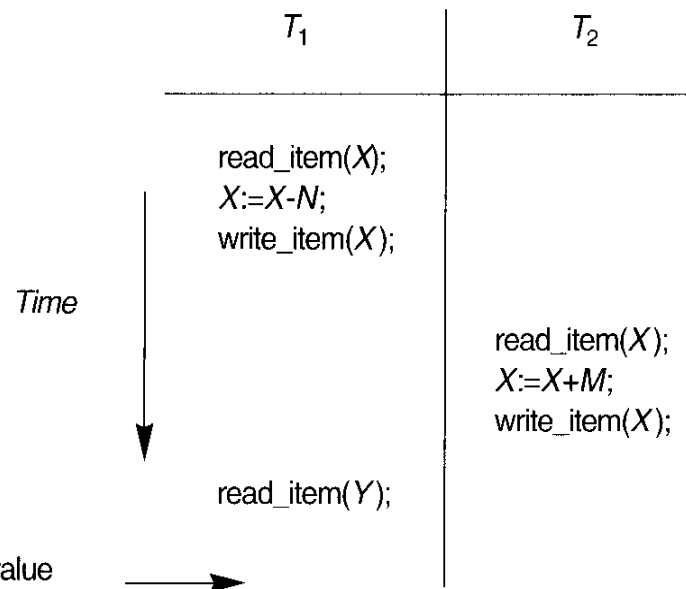
- Notations
 - r, w, c, a (read_item, write_item, commit, abort)
 - $r1(X)$ (transaction 1 reads X)
- Example on Slide 23 can be written
 - Sa: $r1(X), r2(X), w1(X), r1(Y), w2(X), w1(Y)$



Schedules of Transactions



- Notations
 - r, w, c, a
 - $r1(X)$
- Example on Slide 24 can be written
 - Sb: $r1(X), w1(X), r2(X), w2(X), r1(Y), a1$



Transaction T_1 fails and must change the value of X back to its old value; meanwhile T_2 has read the "temporary" incorrect value of X .

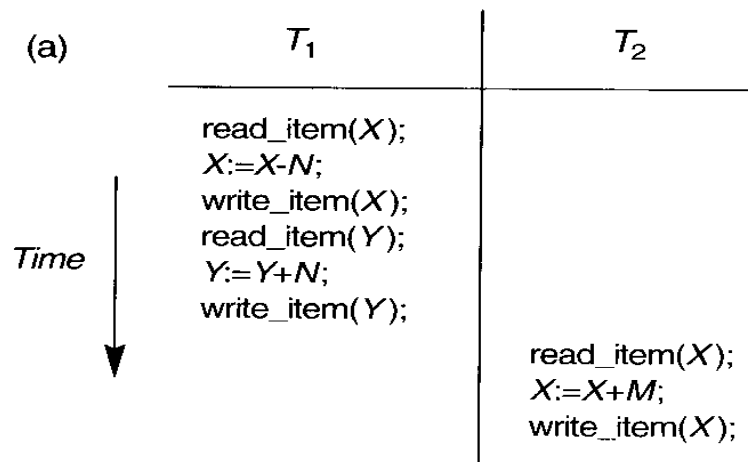
Conflict within a Schedule

- Two operations in a schedule are said to conflict if they satisfy **all** the following conditions
 - They belong to different transactions
 - They access the same item
 - At least one of the operations is a write
- Do the previous schedules have any conflicts?
- Sa: $r1(X), r2(X), w1(X), r1(Y), w2(X), w1(Y)$
- Sb: $r1(X), w1(X), r2(X), w2(X), r1(Y), a1$

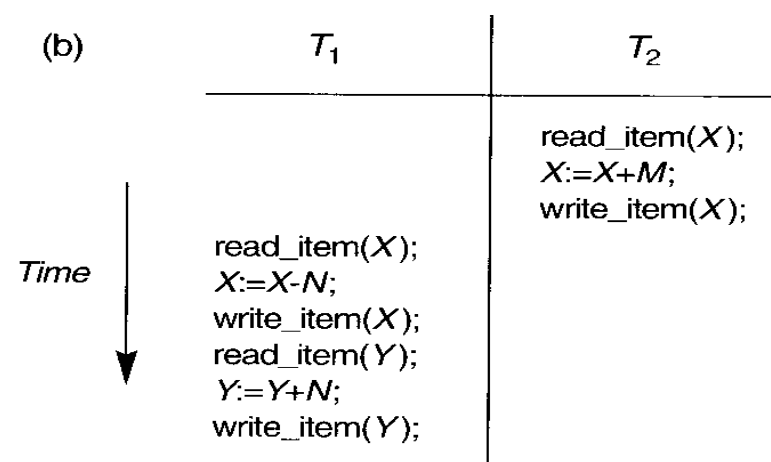


Serial Schedules

- A schedule is serial if, for every transaction T_i participating in the schedule, all the operations of T_i are executed consecutively.
- If the transactions are independent, every serial schedule is correct.



Schedule A



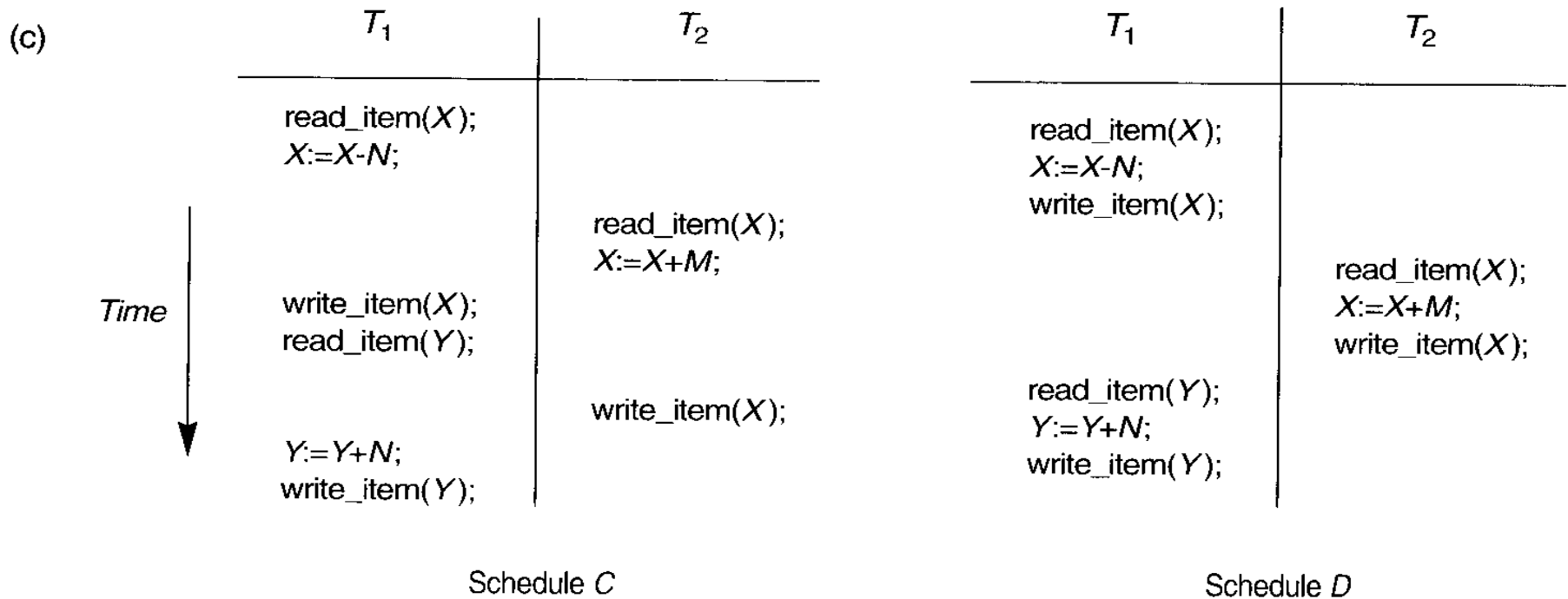
Schedule B

Y = 93 X = 89

Y = 93 X = 89

Nonserial Schedules

- Have interleaving
- Not all are correct



$Y = 93$

$X = 92$

$Y = 93$

$X = 89$

Serialisability

- A schedule of transactions is serialisable if it is equivalent to some serial schedule of the same transactions.
- What good is all this?
 - Can test for serialisability. Time consuming.
 - Design protocols that ensure serialisability (next lecture)

