Database Management Systems

- Transactions

Doug Shook

Transactions

- How many people are using our database?
  - What problems can arise?

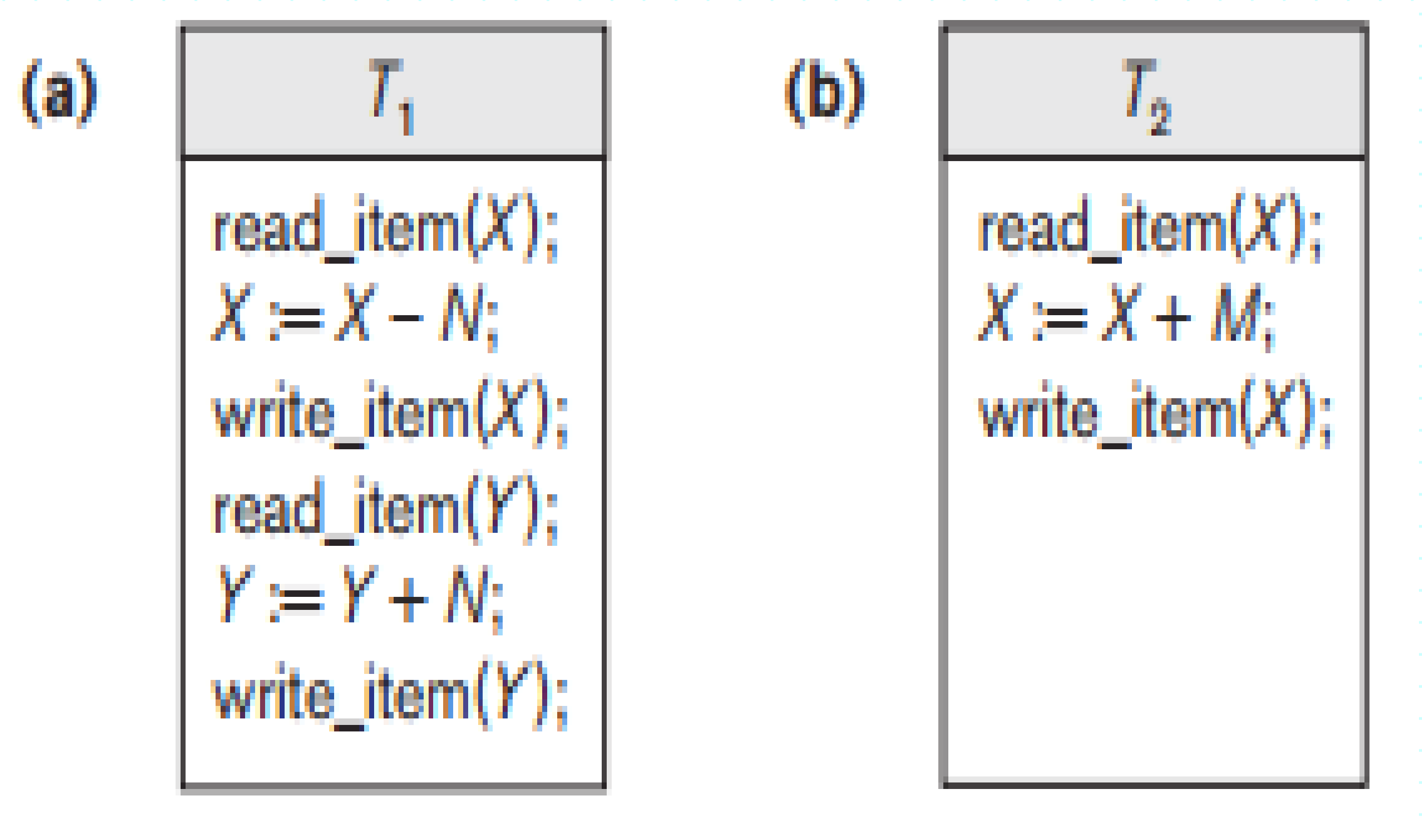
Data Items

- A transaction is a logical grouping of operations.

- What is the piece of data being monitored by a transaction?
  - A record?
  - A column?
  - A page?

- Two basic operations for each item: read and write

Transactions



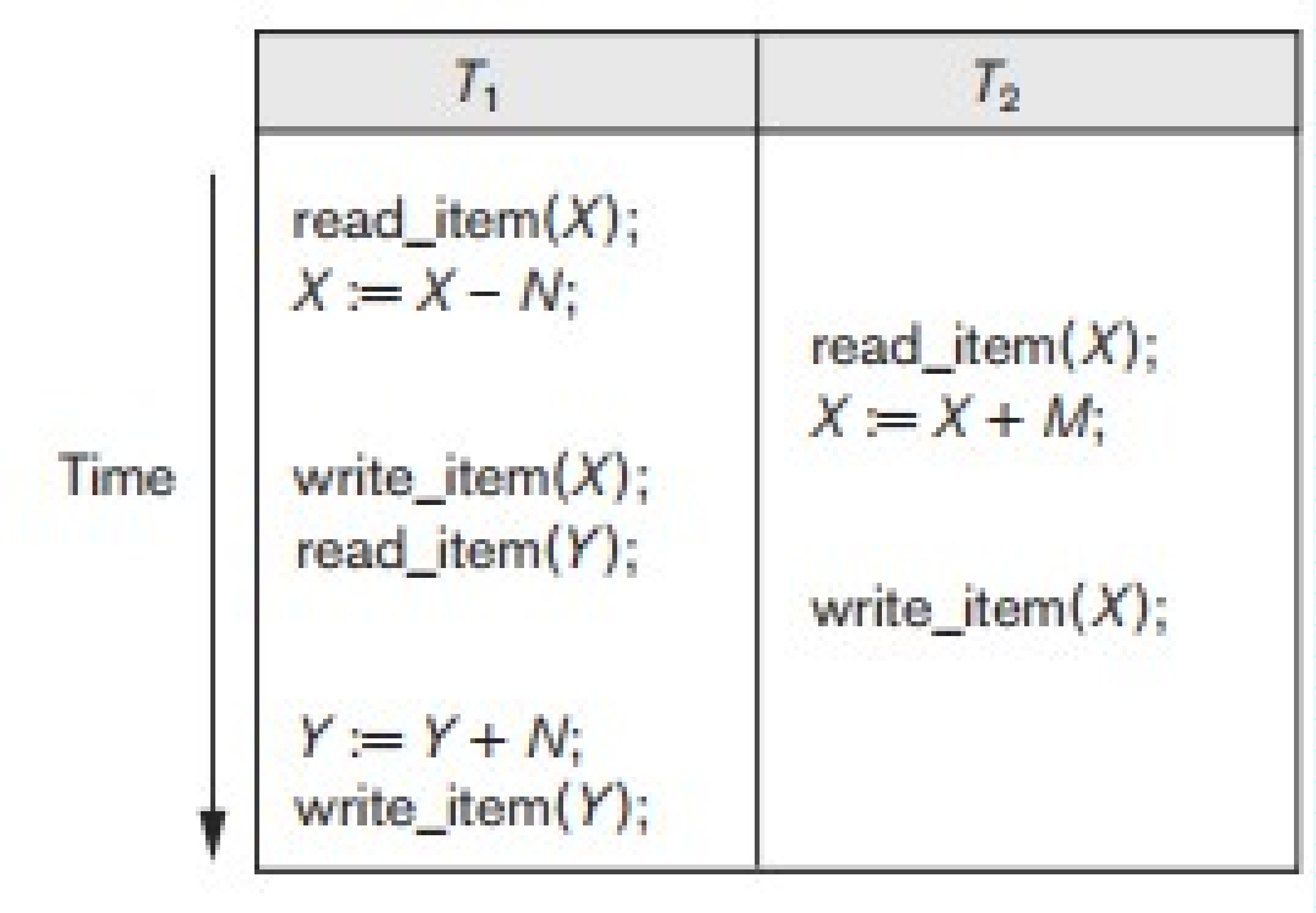
(a)

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

(b)

T_2
read_item(X);
$X := X + M$;
write_item(X);

Lost Update Problem



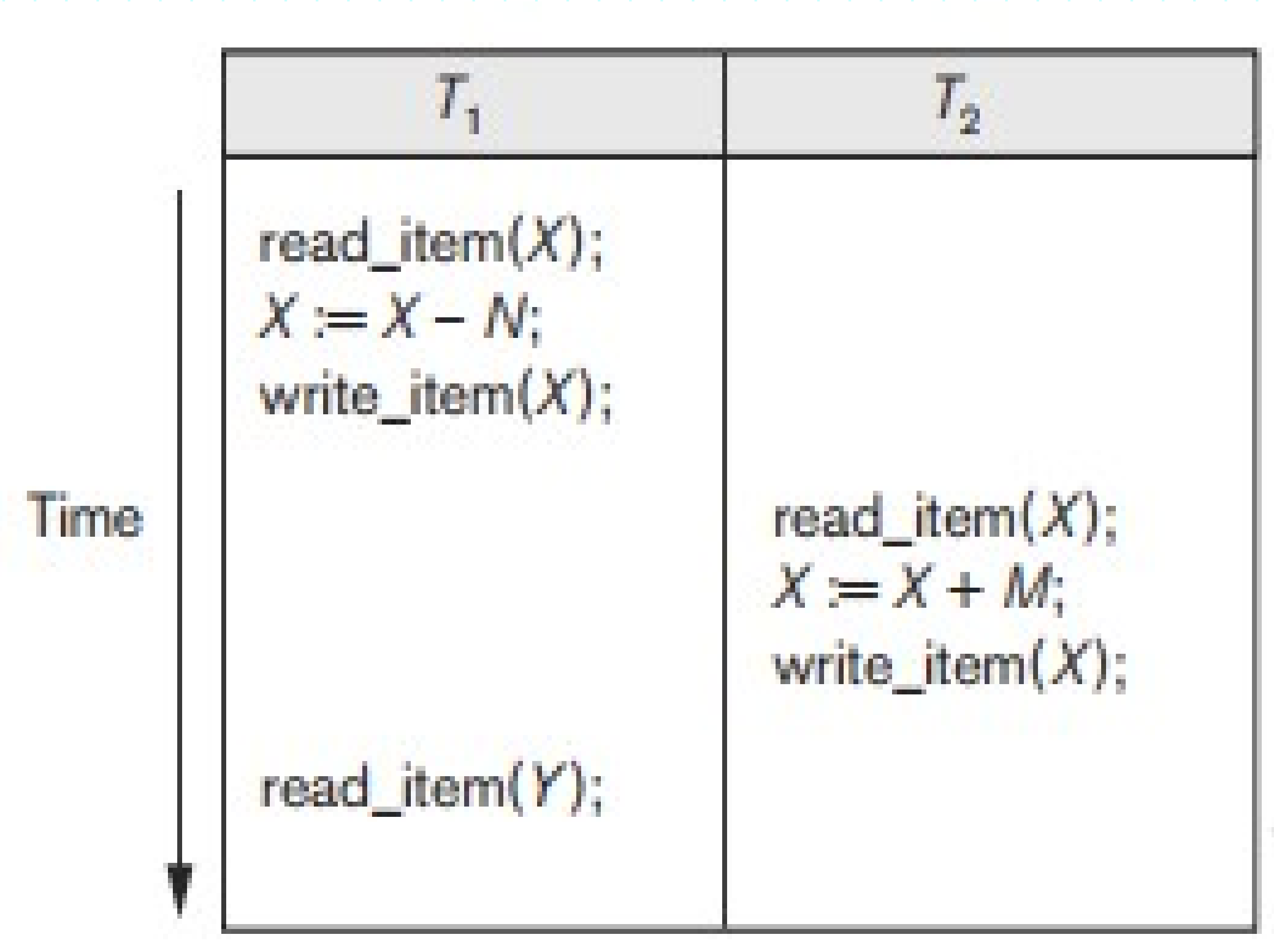
Time

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);	

Temporary Update Problem



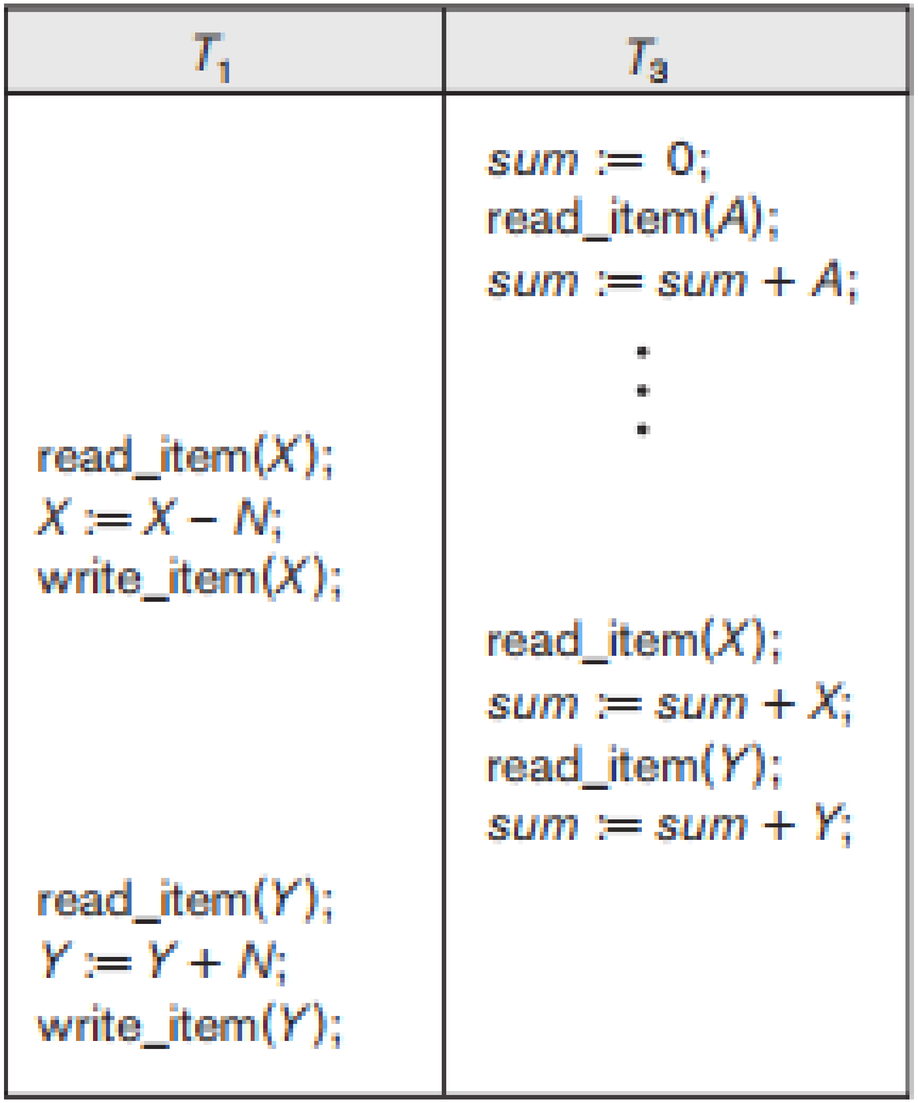
T_1	T_2

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

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

read_item(Y);	

Incorrect Summary Problem



T_1	T_3

	sum := 0;
	read_item(*A*);
	sum := *sum* + *A*;
	⋮

read_item(*X*);
X := *X* − *N*;
write_item(*X*);	

	read_item(*X*);
	sum := *sum* + *X*;
	read_item(*Y*);
	sum := *sum* + *Y*;

read_item(*Y*);
Y := *Y* + *N*;
write_item(*Y*);	

Unrepeatable Read Problem

- Occurs when same value is read twice, but modified in between
  - We expect the result to be the same for both reads, but it is not

Recovery

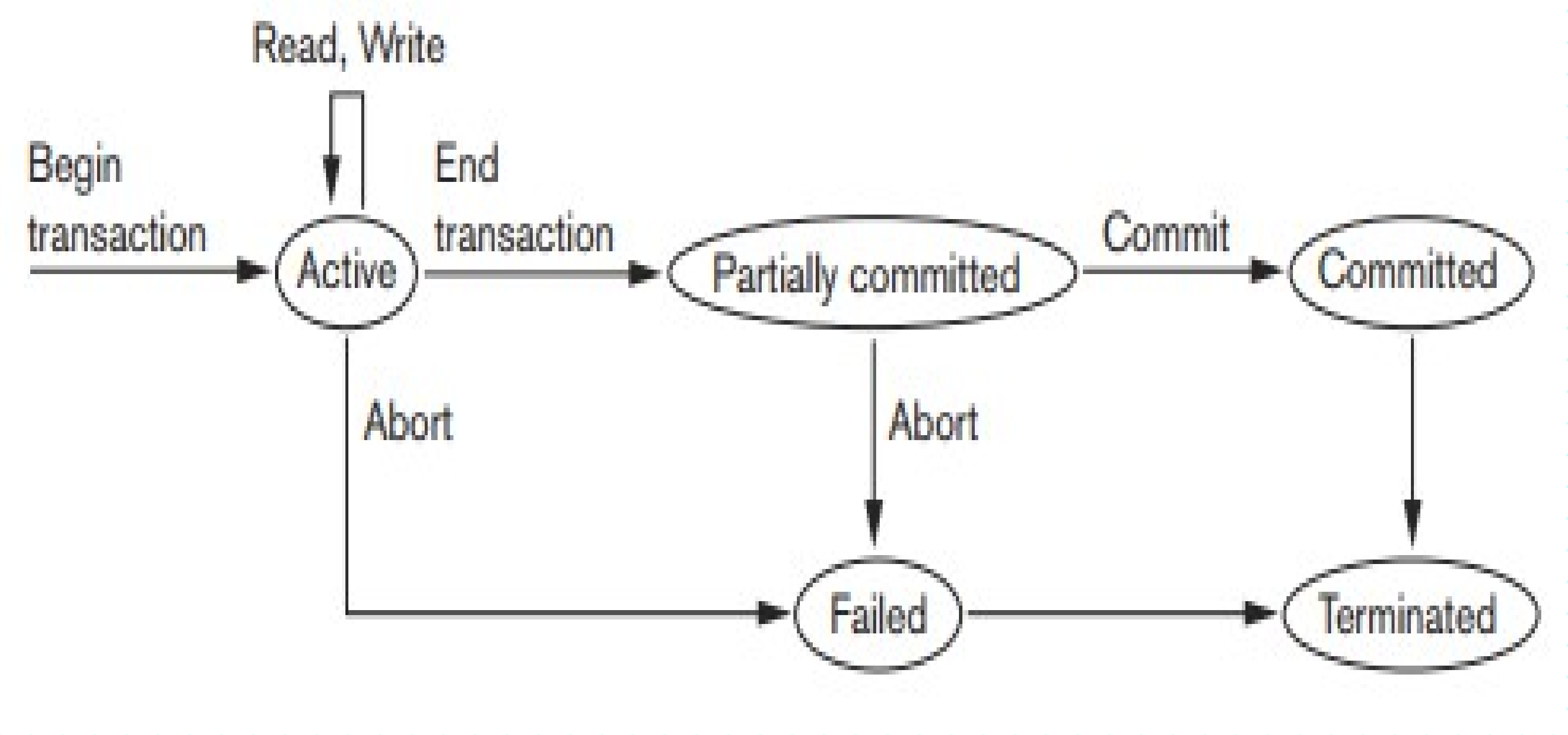
- All statements must execute within a transaction
  - Committed

- If a problem occurs that prevents a transaction from completing, we must undo our changes
  - What kinds of problems could occur?

Transaction State

- BEGIN TRANSACTION
- READ OR WRITE
- END TRANSACTION
- COMMIT TRANSACTION
- ROLLBACK

Transaction State



Log Files

- Sequential, append only list of transaction states

- Each entry contains:
  - Transaction ID
  - State
  - Any data modifications that were made

- This log is useful for
  - ROLLBACK
  - Recovery

Properties of Transactions

- Often called ACID properties:
  - Atomicity
  - Consistency Preservation
  - Isolation
  - Durability

Schedules

- A schedule is an ordering of the operations within a set of transactions
  - Operations from different transactions can be interleaved
  - Operations from a transaction must show up in the same order in the schedule

- Lost Update Schedule
  - Sa: r1(X); r2(X); w1(X); r1(Y); w2(X); w1(Y);

- Temporary Update Schedule
  - Sb: r1(X); w1(X); r2(X); w2(X); r1(Y); a1;

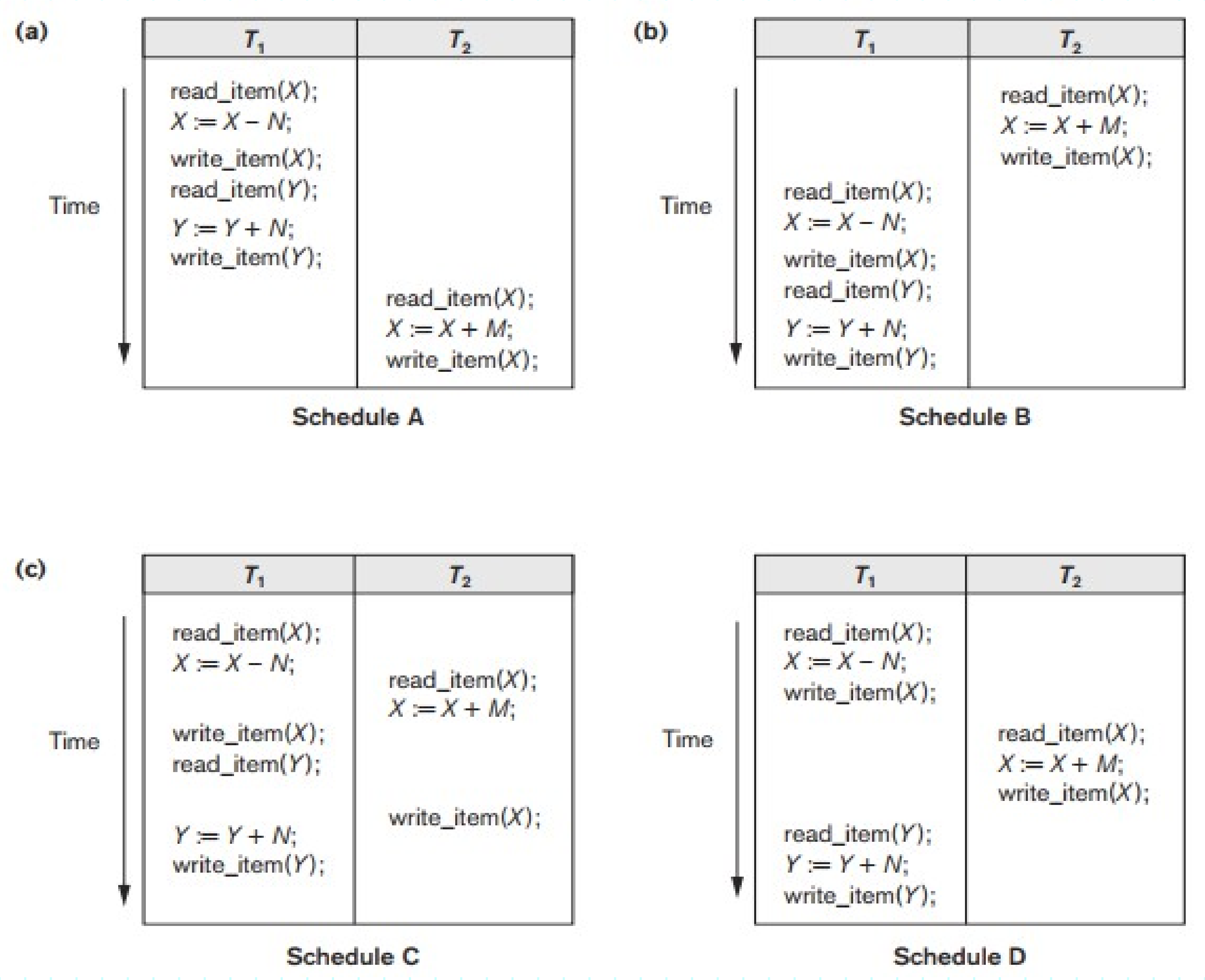
Conflicts

- Operations are said to be in conflict if:
  - They belong to different transactions
  - They access the same item
  - At least one of the operations is a write

- Two types
  - Read/write
  - Write/write

- We must be aware of conflicts when forming schedules

Serial vs. Nonserial Schedules



Time
Time
Time
Time

(a)

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

Schedule A

(b)

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

Schedule B

(c)

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);	

Schedule C

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

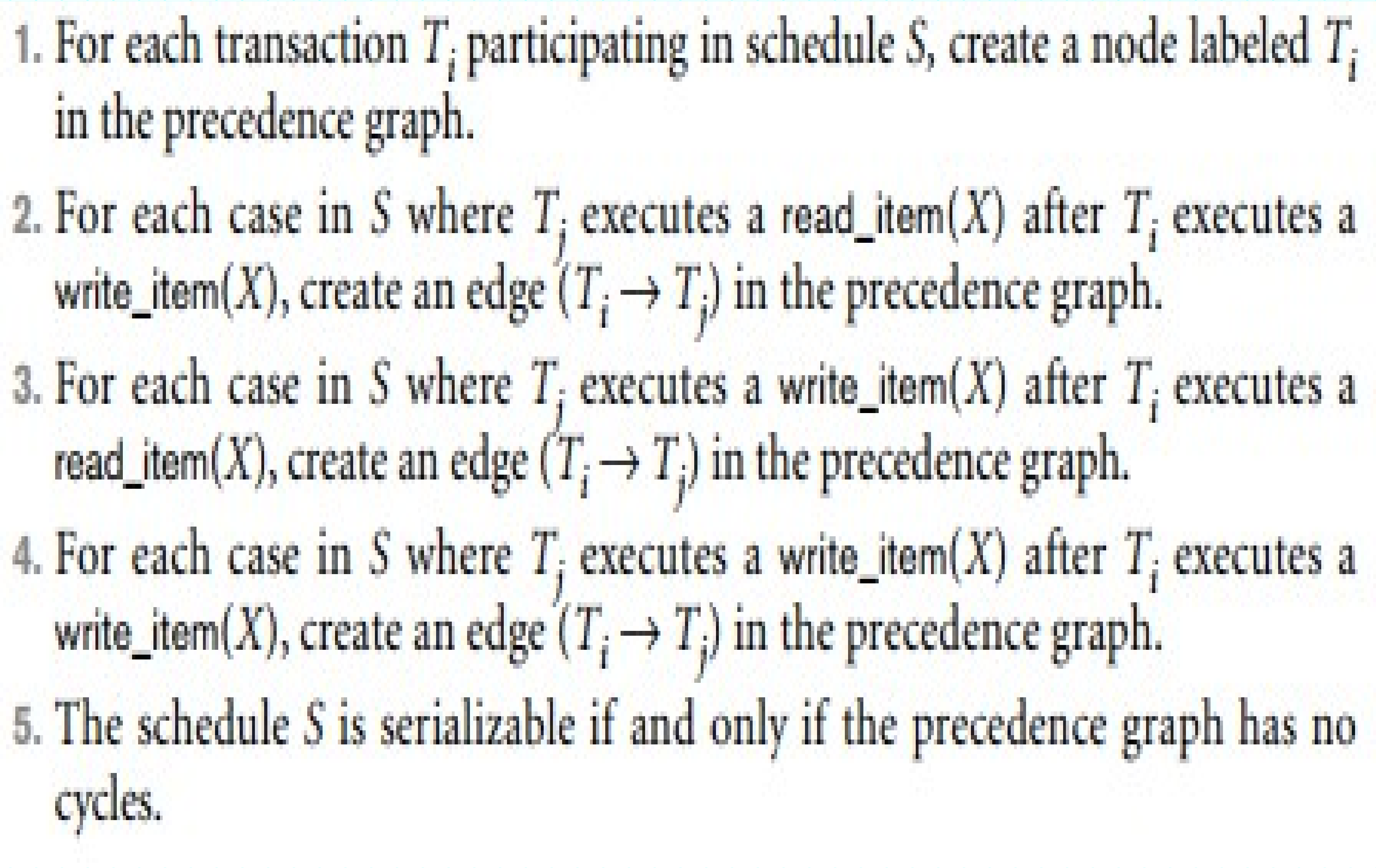
Schedule D

Serializable Schedules

- Serial Schedules are considered unnacceptable
  - Why?

- We wish to use non-serial schedules, but guarantee the same outcome
  - A serializable schedule is any schedule that is equivalent to a serial schedule
    - But what defines equivalence?

Conflict Serializable Test



1. For each transaction T_i participating in schedule S, create a node labeled T_i in the precedence graph.
2. For each case in S where T_j executes a read_item(X) after T_i executes a write_item(X), create an edge ($T_i \rightarrow T_j$) in the precedence graph.
3. For each case in S where T_j executes a write_item(X) after T_i executes a read_item(X), create an edge ($T_i \rightarrow T_j$) in the precedence graph.
4. For each case in S where T_j executes a write_item(X) after T_i executes a write_item(X), create an edge ($T_i \rightarrow T_j$) in the precedence graph.
5. The schedule S is serializable if and only if the precedence graph has no cycles.

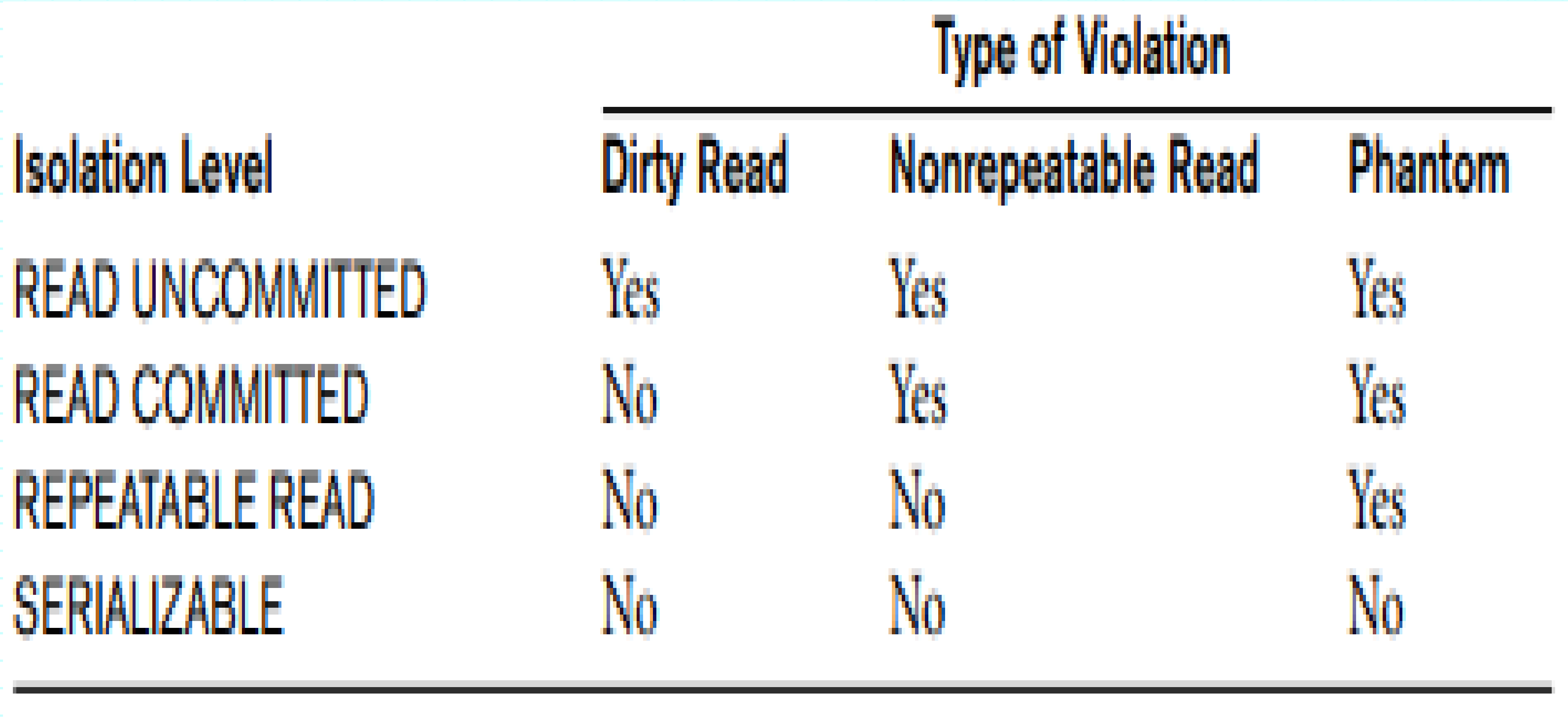
Practice Problem

- Run the conflict test for each schedule on slide 16

Transactions in SQL

- START TRANSACTION
- COMMIT
- ROLLBACK

ISOLATION LEVEL



	Type of Violation		
Isolation Level	Dirty Read	Nonrepeatable Read	Phantom
READ UNCOMMITTED	Yes	Yes	Yes
READ COMMITTED	No	Yes	Yes
REPEATABLE READ	No	No	Yes
SERIALIZABLE	No	No	No