COMP207
Database Development

Lecture 4

Transaction Management:
Serial and Concurrent Schedules

Labs start next week

- Lab start next week

- If you are not assigned to a session, tell me please

Team introduction

Shagufta Scanlon

In charge of assignments and labs

Demonstators

Micheal Abaho

Satya Chavali

Ameneh Irankhah

Mingxuan Liu

Udhayabanu Natarajan

Gautam Pal

Lukasz Przybyla

Kortext drop in session

- Kortext (the guys giving the free e-books) will answer questions you have **Wednesday 2nd October** at **1pm** in **Lecture Theatre C** on the first floor of the **Central Teaching Hub**

Review

- Previous lecture:
  - Transactions
  - How transactions help to solve problems in practice
  - **ACID** properties
  - Which components of a DBMS are responsible for **ACID**
- Can you recall these?

Transactions

- Sequences of operations

  1. read(staffNo=1234, salary)
  2. salary=salary*1.1
  3. write(staffNo=1234, salary)

- Goals:
  - *Atomicity*: execute as a whole or not at all
  - *Serialisability*: the effect of executing it should be the same as if all transactions were executed serially

- In practice, avoids problems due to:
  - *Concurrency*: SQL queries that overlap in time
  - *Partial execution* of SQL statements (e.g., due to failures)

ACID Properties

A: Atomicity

C: Consistency

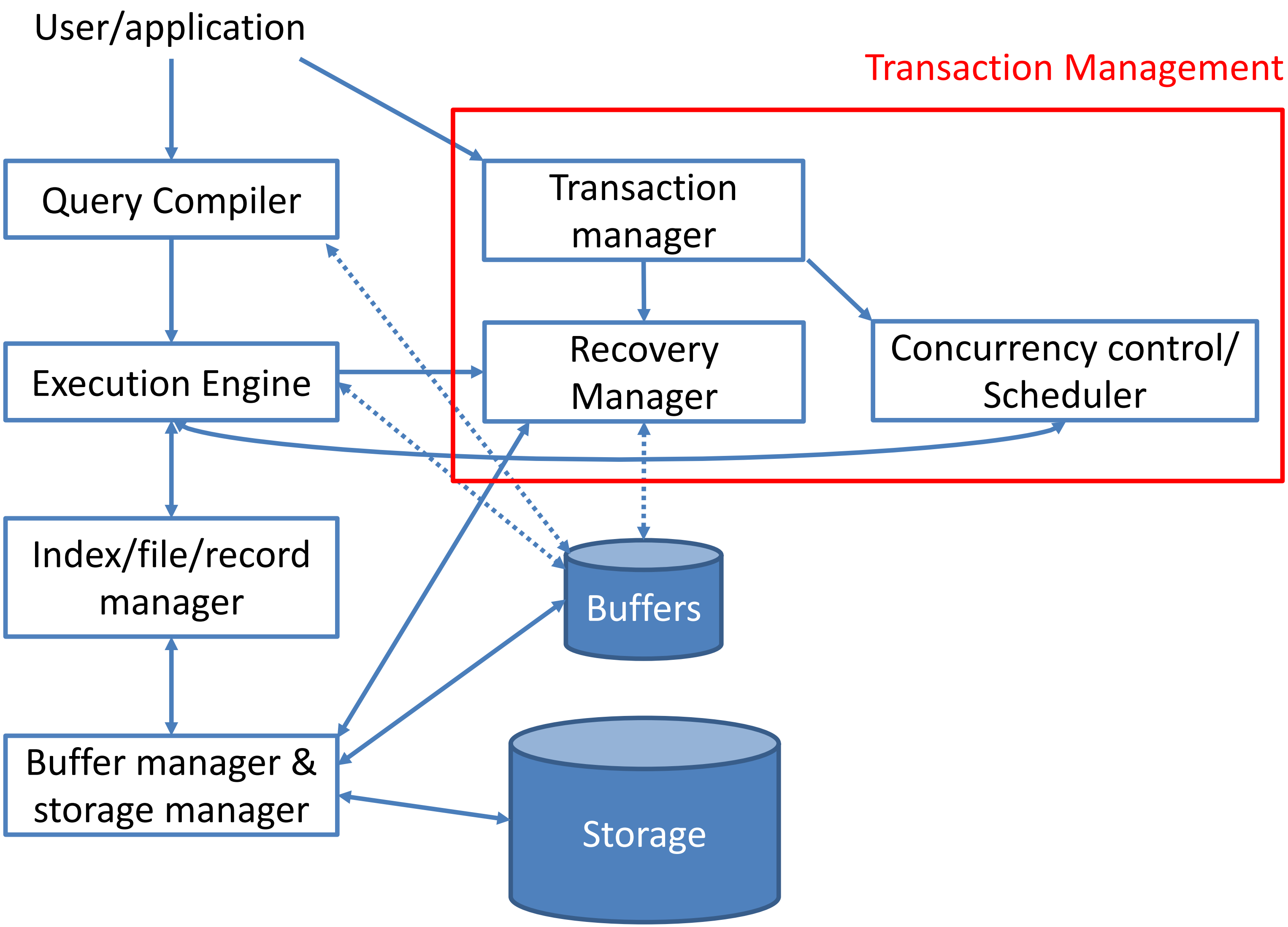
I: Isolation

D: Durability

Can you think of any situation where *Durability* might be violated?

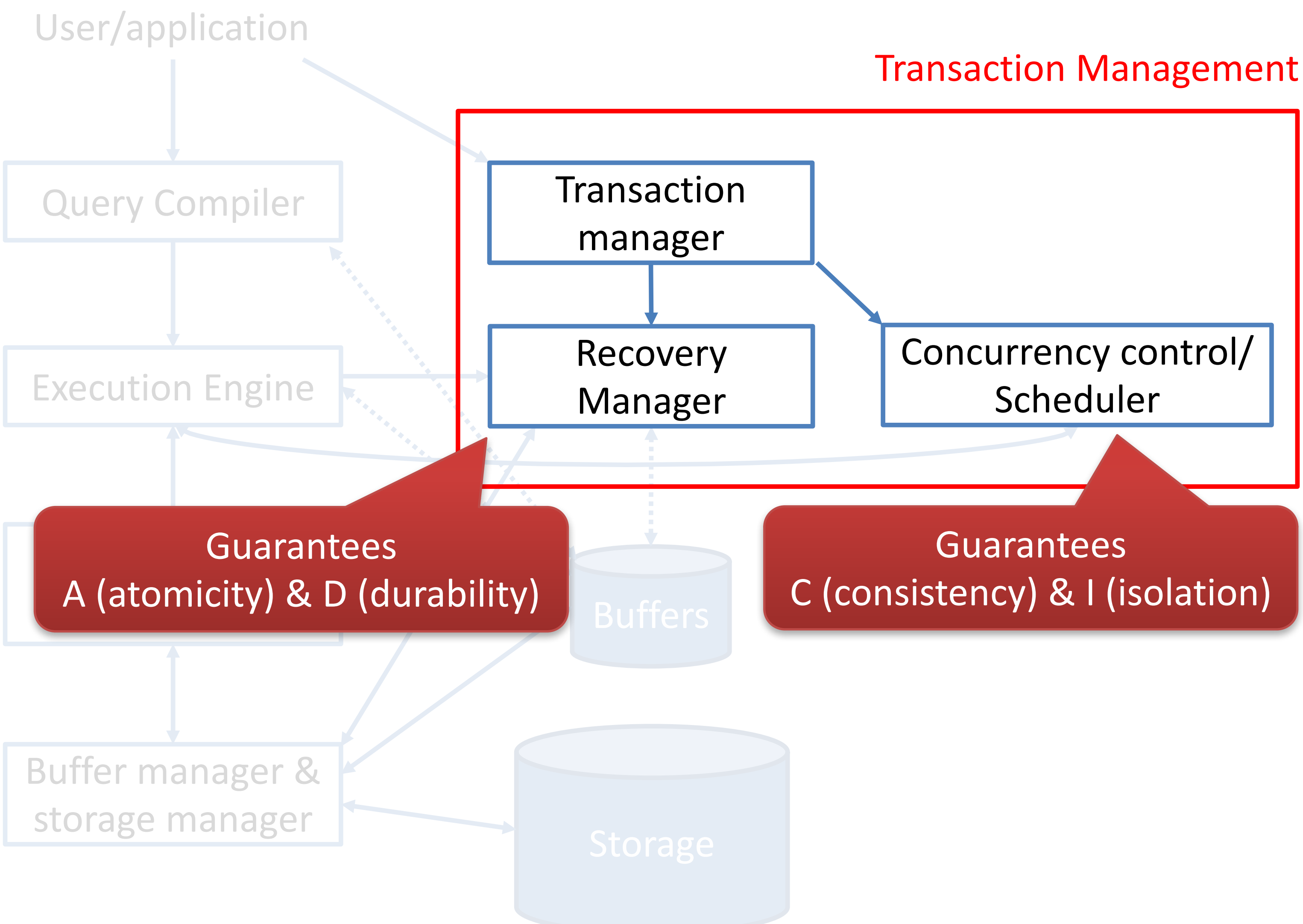
Relational DBMS Components

(Simplified, from Lecture 1)



Relational DBMS Components

(Simplified, from Lecture 1)



Outline

- Today:
  - Concurrency control – how to guarantee **C**onsistency and **I**solation?
  - Focus on basic concepts

Basic Operations of Transactions

- **read_item(X):** Reads a database item X into a program variable (also named X, for simplicity)
  - Find the address of the disk block (page) that contains item X
  - Copy that disk block into a buffer in main memory
    - if that disk block is not already in some main memory buffer
  - Copy item X from the buffer to the program variable X

- **write_item(X):** Writes the value of program variable X into the database item named X
  - Find the address of the disk block (page) that contains item X.
  - Copy that disk block into a buffer in main memory
    - if that disk block is not already in some main memory buffer.
    - Copy item X from the program variable X into its correct location in the buffer
    - Store the updated block from the buffer back to disk either immediately or at some later point in time.

Two Sample Transactions

T_1

```
Begin
read_item(X);
X := X + 100;
write_item(X);
read_item(Y);
Y := Y + 50;
write_item(Y);
commit;
End
```

Begin/End will often be omitted.

T_2

```
Begin
read_item(X);
read_item(Y);
X := X + Y;
write_item(X);
End
```

A Serial Transaction

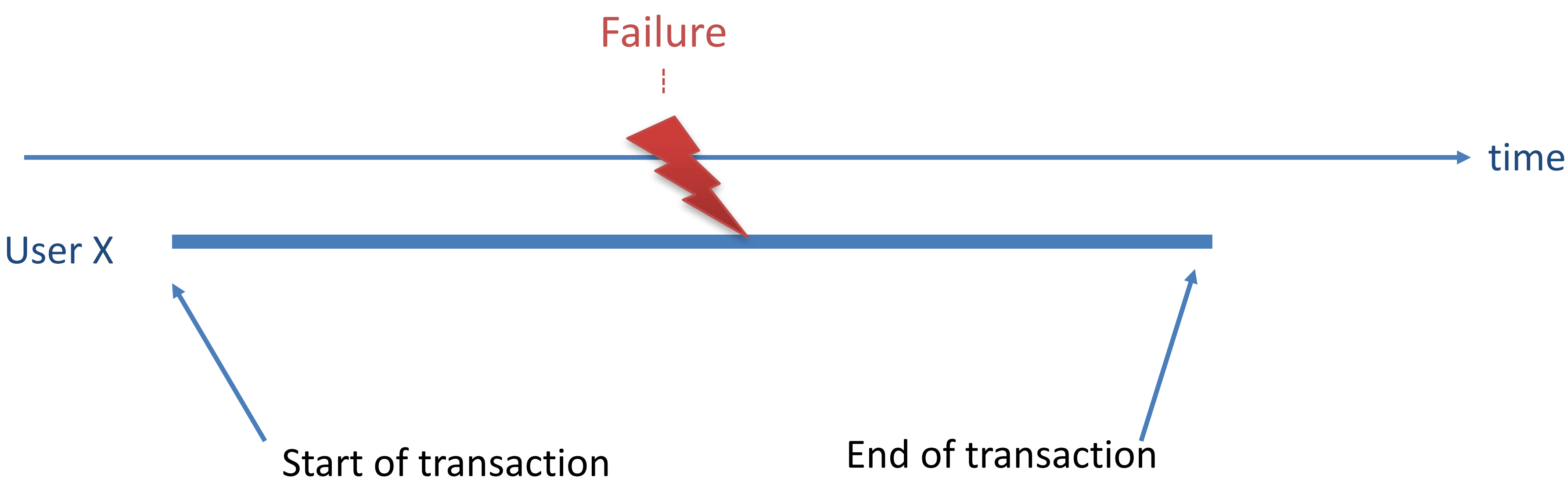
- These are run on their own – they execute serially and give correct results so they maintain consistency in the database

T_3

Begin
read_item(X);
X := X + 100;
write_item(X);
commit;
End

Time	T_3	X
t0		100
t1	Begin	100
t2	read_item(X)	100
t3	X = X + 100	200
t4	write_item(X)	200
t5	commit	200
t6	End	200

Atomicity is broken

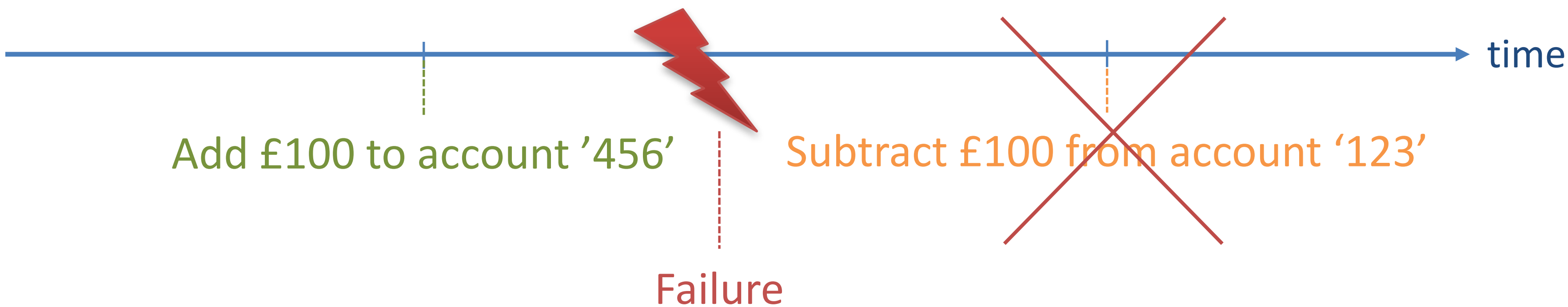


Example 2

(from Lecture 3)

Accounts(accountNo, accountHolder, balance)

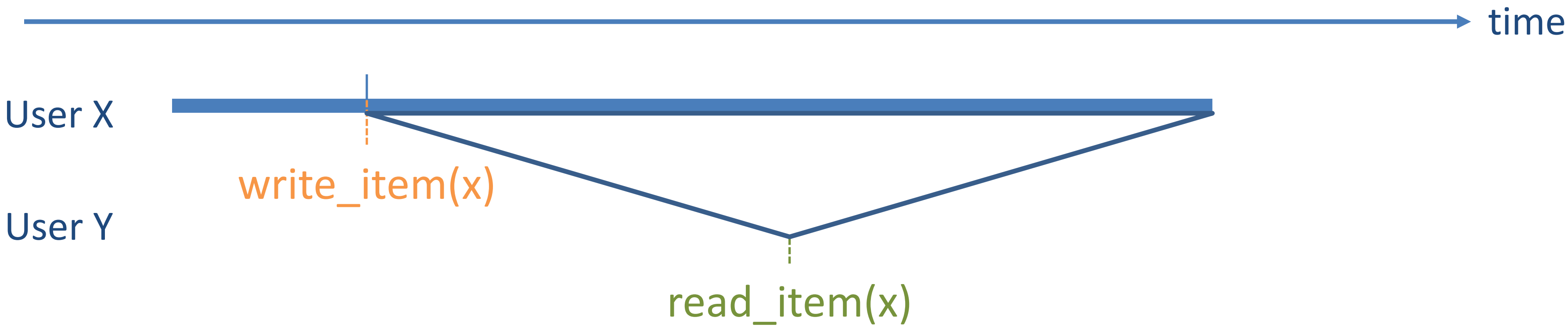
Goal: Transfer £100 from account ‘123’ to account ‘456’




```
UPDATE Accounts
SET    balance = balance + 100
WHERE  accountNo = 456;
```

```
UPDATE Accounts
SET    balance = balance - 100
WHERE  acountNo = 123;
```

Isolation is broken



Example 1

(from Lecture 3)

Flights(flightNo, date, seatNo, seatStatus)

time

User 1

Which seats on flight ‘123’ are still available?

Book seat ‘14B’

User 2

Which seats on flight ‘123’ are still available?

Book seat ‘14B’

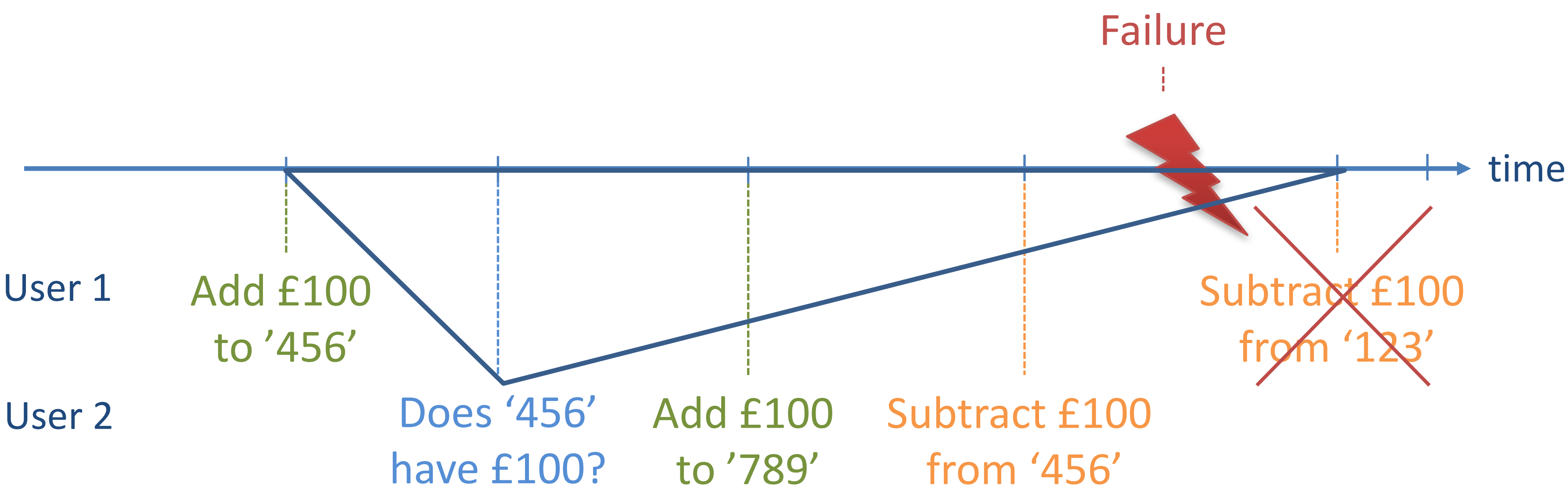
```
SELEC
FROM
WHERE
    AN
    AN
```

```
upied’
8’
```

No violation of isolation, because no triangle!

Example 3

(from Lecture 3)



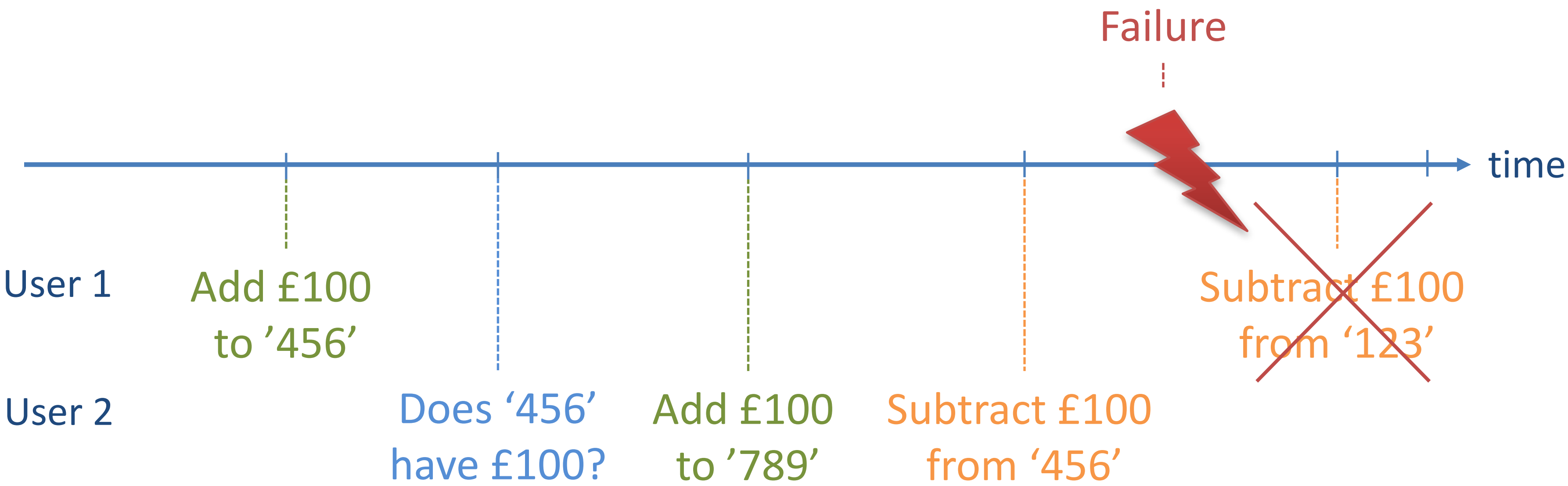
- Which of the ACID properties does this violate?
  - *Atomicity*
  - *Isolation*

Durability is broken

- Two cases:
    1. System and every backup breaks at the same time
    2. We rollback a completed transactions (e.g. to follow some other property)

Example 3

(from Lecture 3)



• W

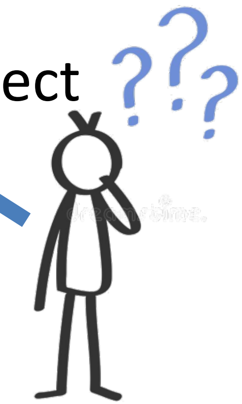
–

–

Must roll user 2's transaction back to satisfy isolation, but that breaks durability!

Consistency is broken

- From definition:
  - It should correctly transform the database state to reflect the effect of a real world event
  - Transactions may not violate integrity constraints



Easy!

- That said, we assume that consistency is satisfied for serial transactions (or equivalent) – see next slide!

Schedules

- Schedules hold many transactions for execution
- **Serial Schedule**
  - holds transactions, one after another
  - preserves the order of the operations of transactions
- Simplest form of schedule that guarantees *Consistency* and *Isolation* properties
- No interleaving of operations of the transactions (this would make it a ‘concurrent schedule’ → later)

A Serial Schedule

```
Begin (T1)
    read_item(X);
    X := X + 100;
    write_item(X);
    read_item(Y);
    Y := Y + 50;
    write_item(Y);
    commit;
End (T1)
Begin (T2)
    read_item(X);
    read_item(Y);
    X := X + Y;
    write_item(X);
    commit;
End (T2)
```

- Executes all operations in transaction T1, then all operations in transaction T2.
- As we are concerned with maintaining ACID properties, we will often ignore non-database operations such as X := X + 100 in future notation

Shorthand Notation for Schedules

Begin (T1)

    read_item(X);
    X := X + 100;
    write_item(X);
    read_item(Y);
    Y := Y + 50;
    write_item(Y);
    commit;

End (T1)

Begin (T2)

    read_item(X);
    read_item(Y);
    X := X + Y;
    write_item(X);
    commit;

End (T2)

- Shorthand notation for this schedule:

  $\mathbf{S_a\text{: } r_1(X); w_1(X); r_1(Y); w_1(y); c_1; r_2(X); r_2(Y); w_2(X); c_2}$

- Symbols:
  - $\mathbf{S}_{id}$ = schedule (*id* is the schedule ID)
  - $\mathbf{r}_i\mathbf{(X)}$ = read_item(X) in transaction *i*
  - $\mathbf{w}_i\mathbf{(X)}$ = write_item(X) in transaction *i*
  - $\mathbf{c}_i$ = commit in transaction *i*
  - $\mathbf{a}_i$ = abort (“rollback”) in transaction *i*

- Will be used much more in the next lectures

Another Example

Time	S_b	X
t0		100
t1	read_item(X)	100
t2	X = X - 10	90
t3	write_item (X)	90
t4	commit	90
t5	read_item(X)	90
t6	X = X + 100	190
t7	write_item(X)	190
t8	commit	190

What is the shorthand notation for this schedule?

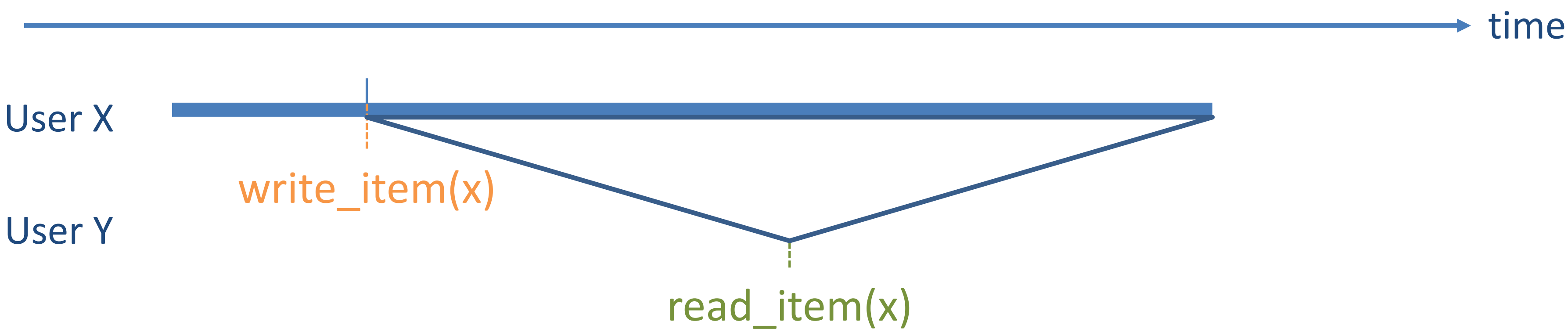
Another Example

Time	S_b	X
t0		100
t1	read_item(X)	100
t2	X = X - 10	90
t3	write_item (X)	90
t4	commit	90
t5	read_item(X)	90
t6	X = X + 100	190
t7	write_item(X)	190
t8	commit	190

What is the shorthand notation for this schedule?

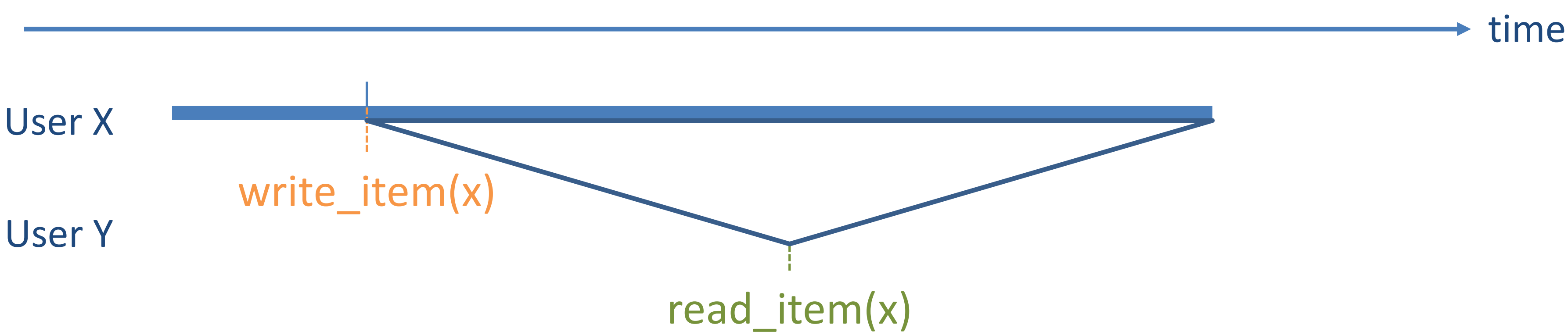
S_b: $r_1(x)$; $w_1(x)$; c_1; $r_2(x)$; $w_2(x)$; c_2

Isolation is broken



What is the shorthand notation for when isolation is broken?

Isolation is broken



What is the shorthand notation for when isolation is broken?

S_c: $w_x(x)$; … $r_y(x)$; … c_x; or
S_d: $w_x(x)$; … $r_y(x)$; … a_x; for $x \neq y$

Order Matters

Time	Schedule	X
t0		0
t1	read_item(X)	0
t2	X = X + 100	100
t3	write_item(X)	100
t4	commit	100
t5	read_item(X)	100
t6	X = X * 2	200
t7	write_item(X)	200
t8	commit	200

vs

Time	Schedule	X
t0		0
t1	read_item(X)	0
t2	X = X * 2	0
t3	write_item(X)	0
t4	commit	0
t5	read_item(X)	0
t6	X = X + 100	100
t7	write_item(X)	100
t8	commit	100

Concurrent Schedule

- Serial schedules waste time:
  - While executing a transaction, other transactions have to wait (assumes a single processor system).
  - Transactions may take a long time.
- *More efficient:* interleave operations of different transactions to maximize CPU usage
- **Concurrent Schedule:**
  - Executes operations from several transactions *concurrently* in an *interleaved* fashion
  - Operations of an individual transaction T appear in the same order in which they occur in T

A Serial Schedule

$r_1(x); w_1(x); r_1(y); w_1(y); c_1; r_2(x); w_2(x); c_2$

Time	Schedule	
t0		
t1	read_item(X)	
t2	X := X - N	
t3	write_item(X)	
t4	read_item(Y)	
t5	Y := Y + N	
t6	write_item(Y)	
t7	commit	
t8		read_item(X)
t9		X := X + M
t10		write_item(X)
t11		commit

A Concurrent Schedule

$r_1(x); r_2(x); w_1(x); r_1(y); w_2(x); c_2; w_1(y); c_1$

Time	Schedule	
t0		
t1	read_item(X)	
t2		read_item(X)
t3		X := X + M
t4	X := X - N	
t5	write_item(X)	
t6	read_item(Y)	
t7		write_item(X)
t8		commit
t9	Y := Y + N	
t10	write_item(Y)	
t11	commit	

Is this schedule always equivalent to the original one?

A Concurrent Schedule

$r_1(x); r_2(x); w_1(x); r_1(y); w_2(x); c_2; w_1(y); c_1$

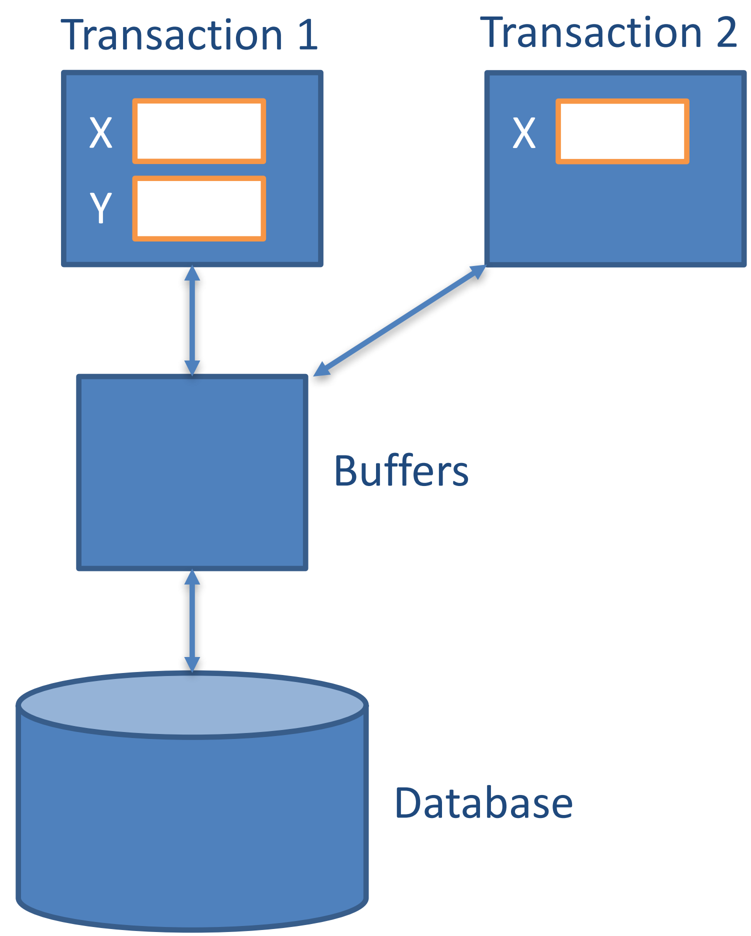
Time	Schedule	
t0		
t1	read_item(X)	
t2		read_item(X)
t3		X := X + M
t4	X := X - N	
t5	write_item(X)	
t6	read_item(Y)	
t7		write_item(X)
t8		commit
t9	Y := Y + N	
t10	write_item(Y)	
t11	commit	

Is this schedule always equivalent to the original one?

No!

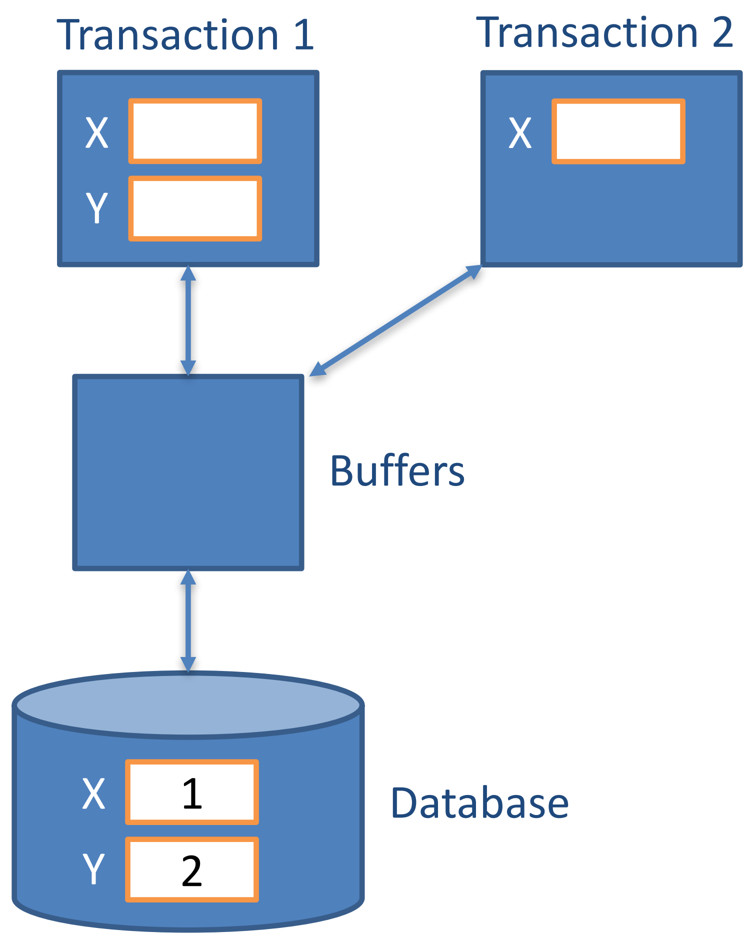
Concurrent Schedules Do Not Guarantee Consistency

Time	Schedule	
t0		
t1	read_item(X)	
t2		read_item(X)
t3		X := X + M
t4	X := X + N	
t5	write_item(X)	
t6	read_item(Y)	
t7		write_item(X)
t8		commit
t9	Y := Y + N	
t10	write_item(Y)	
t11	commit	



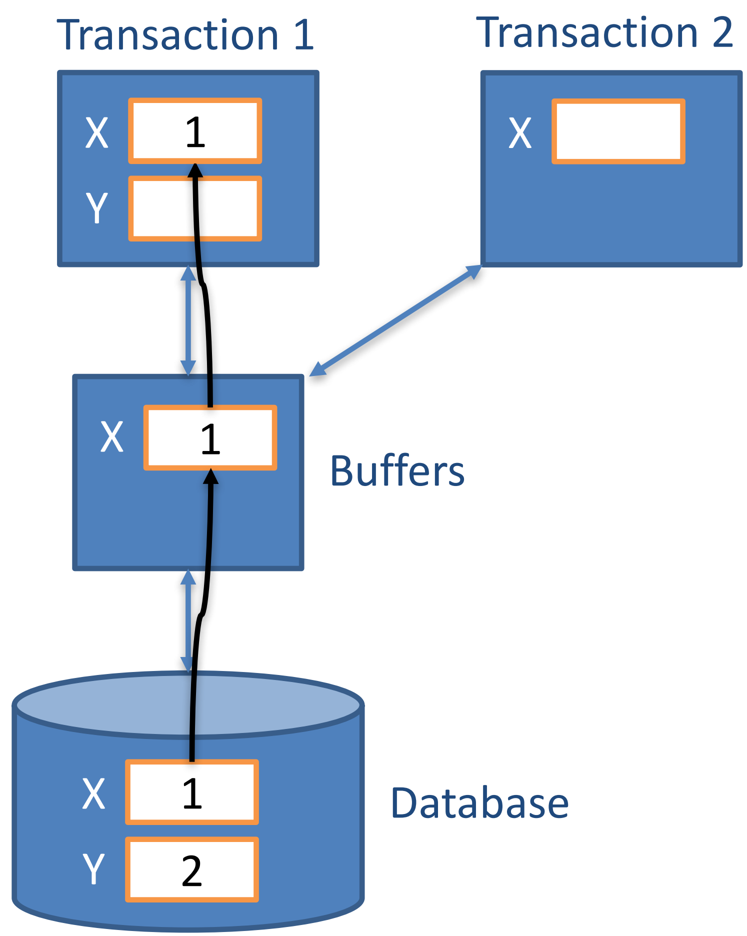
Concurrent Schedules Do Not Guarantee Consistency

Time	Schedule	
t0		
t1	read_item(X)	
t2		read_item(X)
t3		X := X + M
t4	X := X + N	
t5	write_item(X)	
t6	read_item(Y)	
t7		write_item(X)
t8		commit
t9	Y := Y + N	
t10	write_item(Y)	
t11	commit	



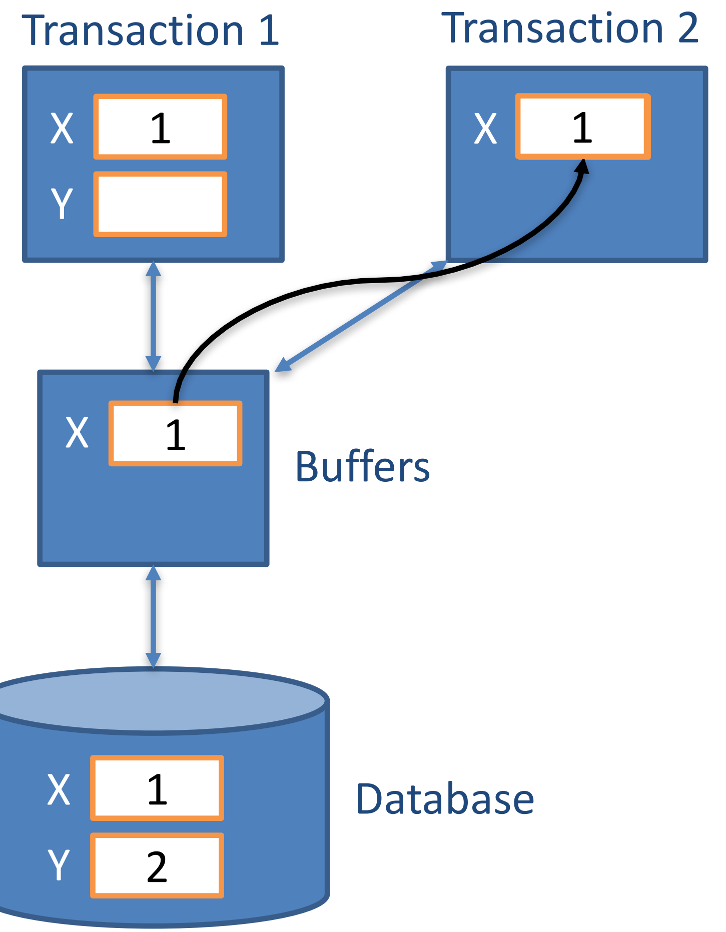
Concurrent Schedules Do Not Guarantee Consistency

Time	Schedule	
t0		
t1	read_item(X)	
t2		read_item(X)
t3		X := X + M
t4	X := X + N	
t5	write_item(X)	
t6	read_item(Y)	
t7		write_item(X)
t8		commit
t9	Y := Y + N	
t10	write_item(Y)	
t11	commit	



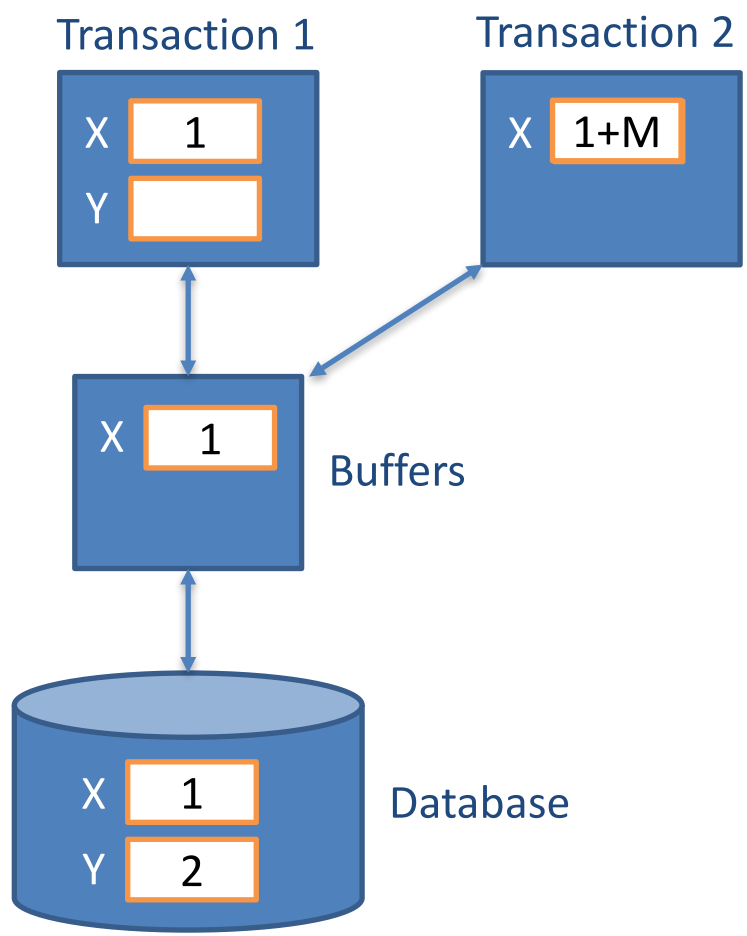
Concurrent Schedules Do Not Guarantee Consistency

Time	Schedule	
t0		
t1	read_item(X)	
t2		read_item(X)
t3		X := X + M
t4	X := X + N	
t5	write_item(X)	
t6	read_item(Y)	
t7		write_item(X)
t8		commit
t9	Y := Y + N	
t10	write_item(Y)	
t11	commit	



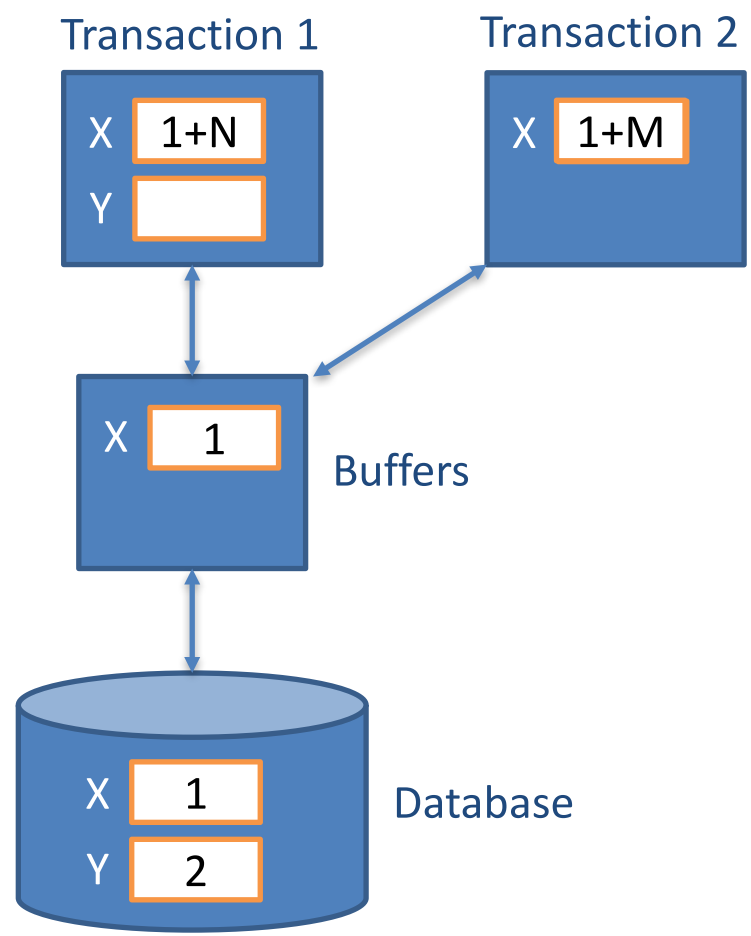
Concurrent Schedules Do Not Guarantee Consistency

Time	Schedule	
t0		
t1	read_item(X)	
t2		read_item(X)
t3		X := X + M
t4	X := X + N	
t5	write_item(X)	
t6	read_item(Y)	
t7		write_item(X)
t8		commit
t9	Y := Y + N	
t10	write_item(Y)	
t11	commit	



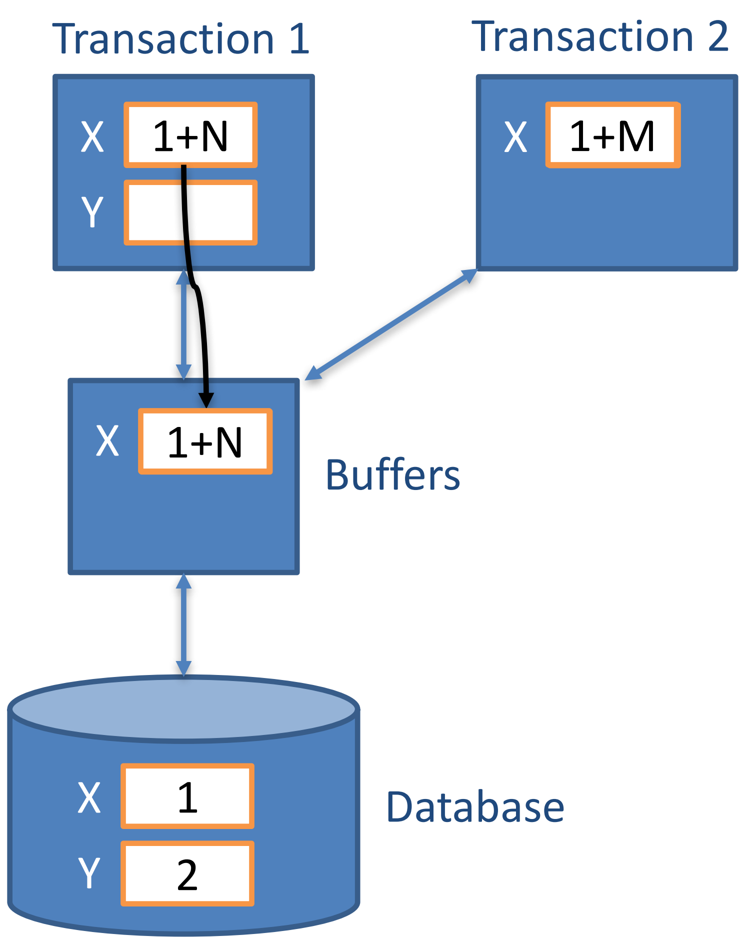
Concurrent Schedules Do Not Guarantee Consistency

Time	Schedule	
t0		
t1	read_item(X)	
t2		read_item(X)
t3		X := X + M
t4	X := X + N	
t5	write_item(X)	
t6	read_item(Y)	
t7		write_item(X)
t8		commit
t9	Y := Y + N	
t10	write_item(Y)	
t11	commit	



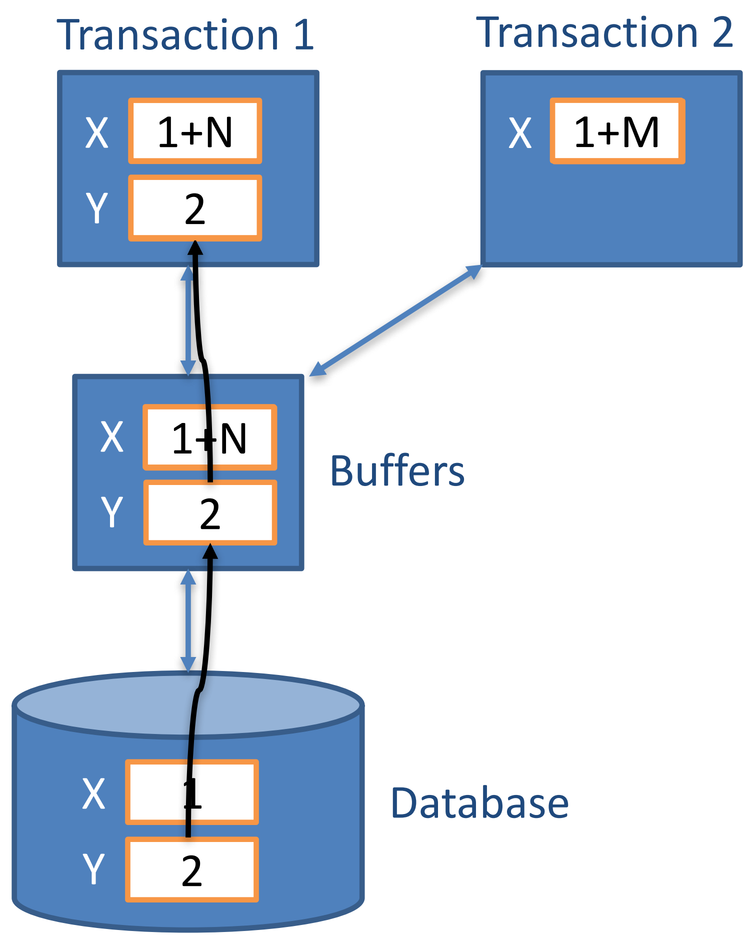
Concurrent Schedules Do Not Guarantee Consistency

Time	Schedule	
t0		
t1	read_item(X)	
t2		read_item(X)
t3		X := X + M
t4	X := X + N	
t5	write_item(X)	
t6	read_item(Y)	
t7		write_item(X)
t8		commit
t9	Y := Y + N	
t10	write_item(Y)	
t11	commit	



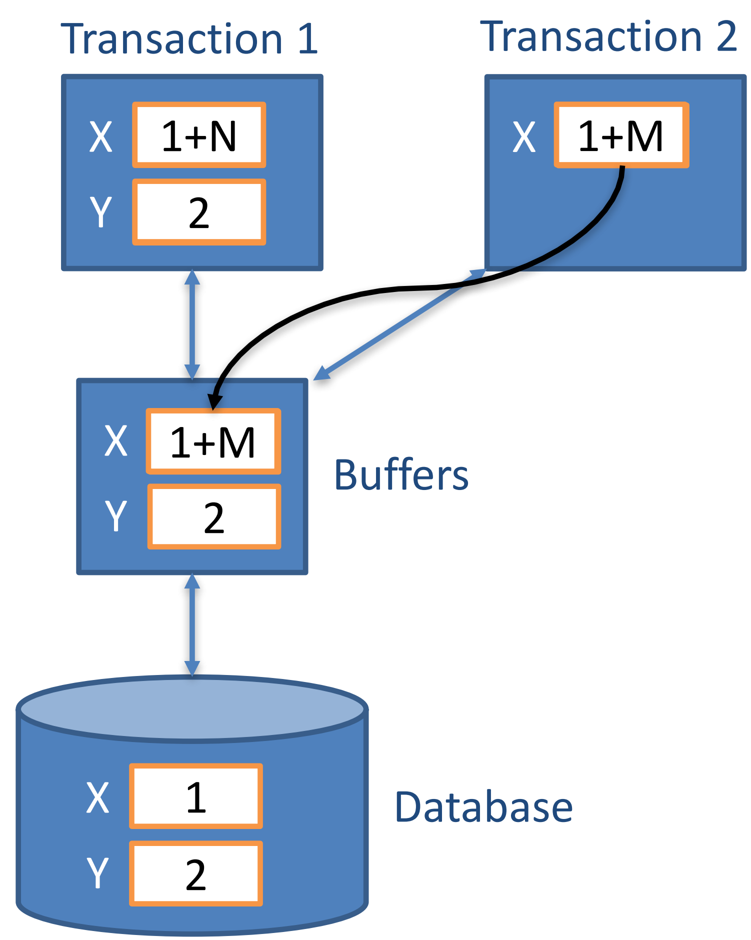
Concurrent Schedules Do Not Guarantee Consistency

Time	Schedule	
t0		
t1	read_item(X)	
t2		read_item(X)
t3		X := X + M
t4	X := X + N	
t5	write_item(X)	
t6	read_item(Y)	
t7		write_item(X)
t8		commit
t9	Y := Y + N	
t10	write_item(Y)	
t11	commit	



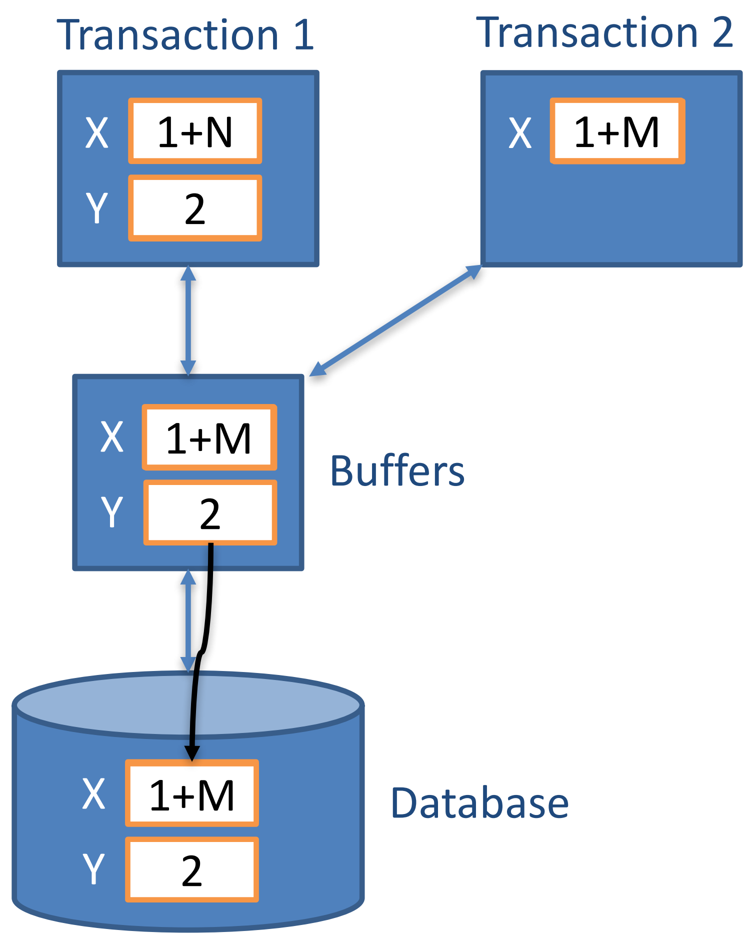
Concurrent Schedules Do Not Guarantee Consistency

Time	Schedule	
t0		
t1	read_item(X)	
t2		read_item(X)
t3		X := X + M
t4	X := X + N	
t5	write_item(X)	
t6	read_item(Y)	
t7		write_item(X)
t8		commit
t9	Y := Y + N	
t10	write_item(Y)	
t11	commit	



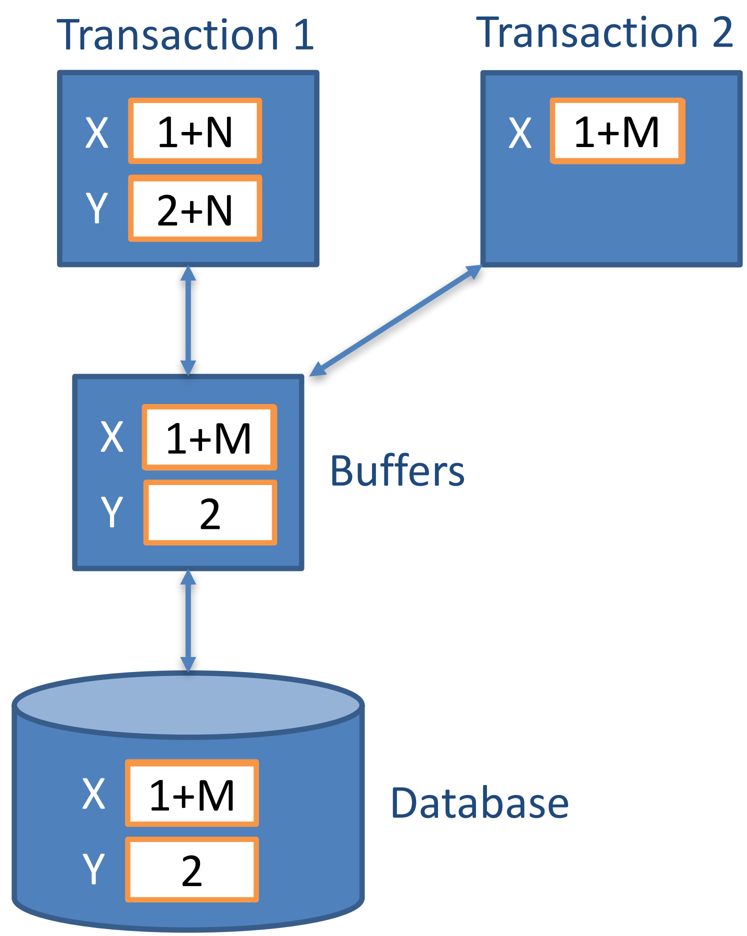
Concurrent Schedules Do Not Guarantee Consistency

Time	Schedule	
t0		
t1	read_item(X)	
t2		read_item(X)
t3		X := X + M
t4	X := X + N	
t5	write_item(X)	
t6	read_item(Y)	
t7		write_item(X)
t8		commit
t9	Y := Y + N	
t10	write_item(Y)	
t11	commit	



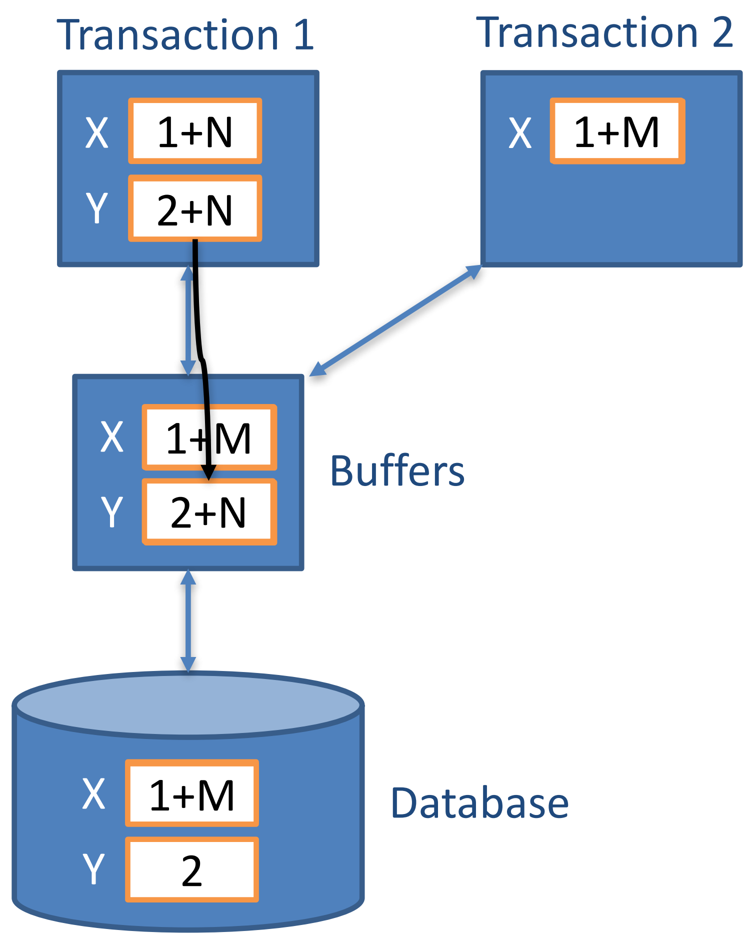
Concurrent Schedules Do Not Guarantee Consistency

Time	Schedule	
t0		
t1	read_item(X)	
t2		read_item(X)
t3		X := X + M
t4	X := X + N	
t5	write_item(X)	
t6	read_item(Y)	
t7		write_item(X)
t8		commit
t9	Y := Y + N	
t10	write_item(Y)	
t11	commit	



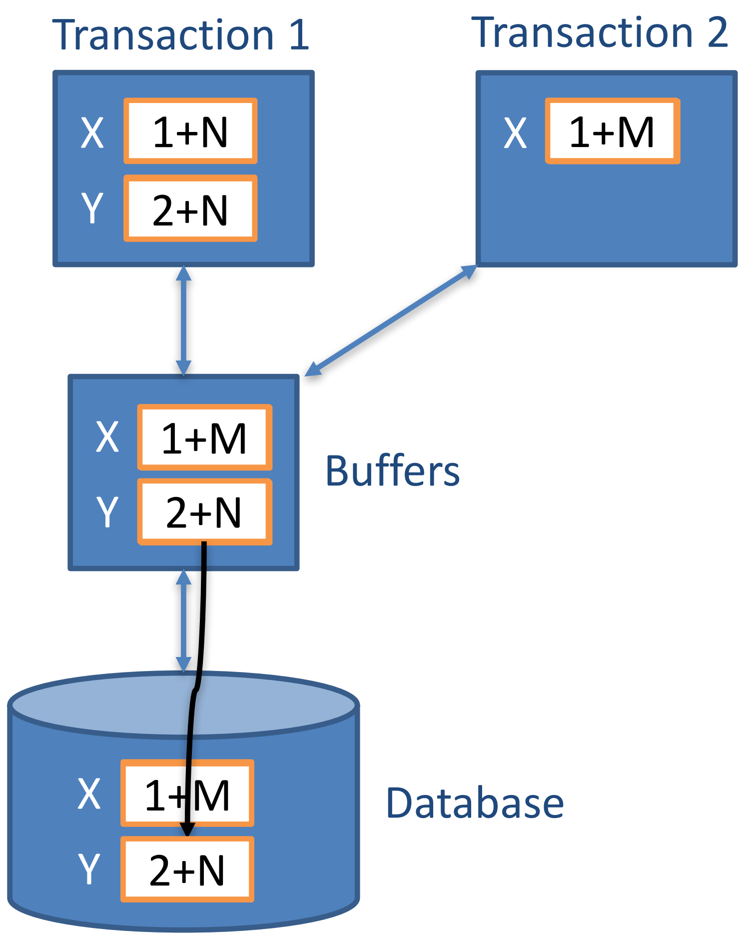
Concurrent Schedules Do Not Guarantee Consistency

Time	Schedule	
t0		
t1	read_item(X)	
t2		read_item(X)
t3		X := X + M
t4	X := X + N	
t5	write_item(X)	
t6	read_item(Y)	
t7		write_item(X)
t8		commit
t9	Y := Y + N	
t10	write_item(Y)	
t11	commit	



Concurrent Schedules Do Not Guarantee Consistency

Time	Schedule	
t0		
t1	read_item(X)	
t2		read_item(X)
t3		X := X + M
t4	X := X + N	
t5	write_item(X)	
t6	read_item(Y)	
t7		write_item(X)
t8		commit
t9	Y := Y + N	
t10	write_item(Y)	
t11	commit	



Concurrent Schedules Do Not Guarantee Consistency

Time	Schedule	
t0		
t1	read_item(X)	
t2		read_item(X)
t3		X := X + M
t4	X := X + N	
t5	write_item(X)	
t6	read_item(Y)	
t7		write_item(X)
t8		commit
t9	Y := Y + N	
t10	write_item(Y)	
t11	commit