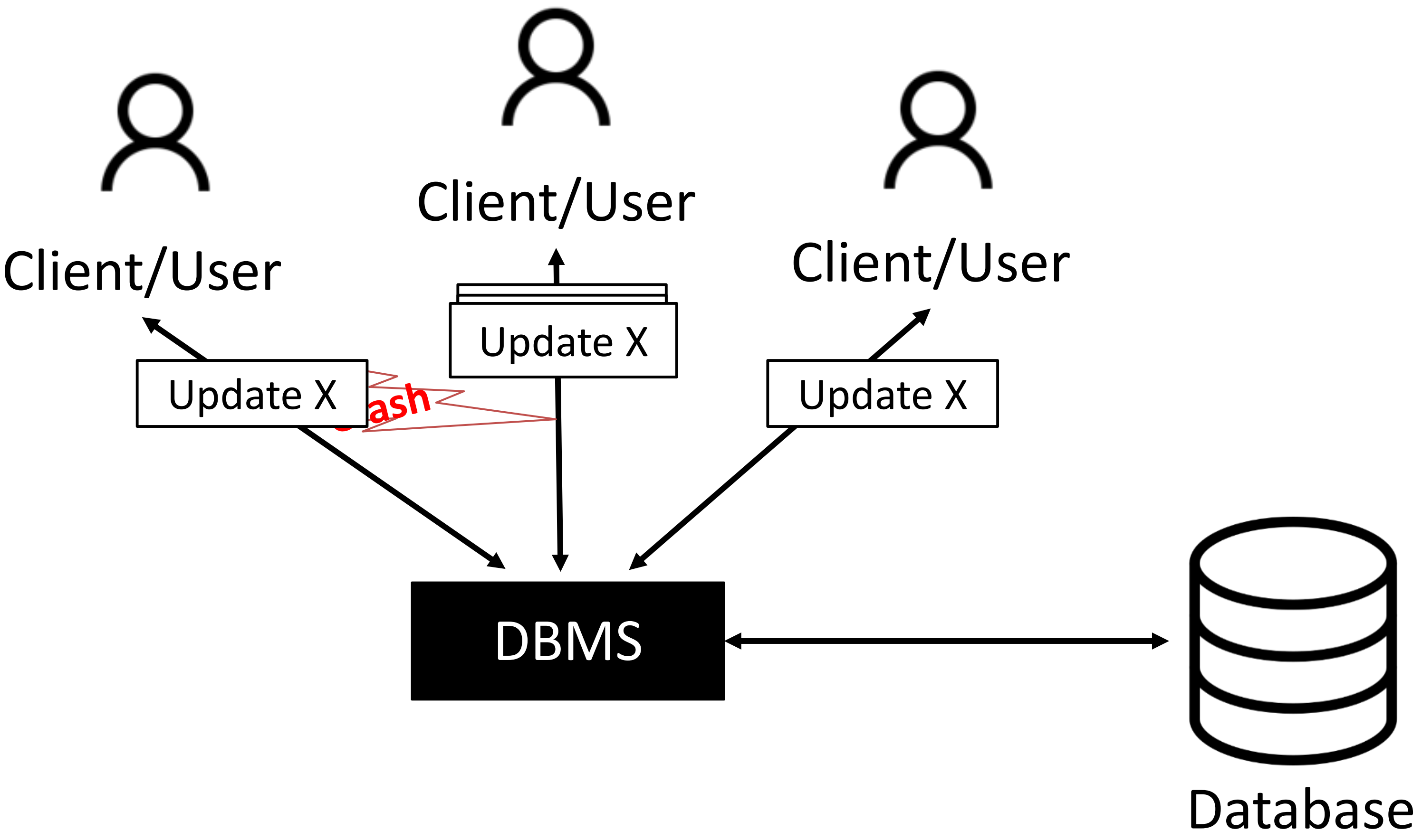
Introduction to Operating Systems and SQL for Data Science

Lecture 12 – Transactions

Previous lecture

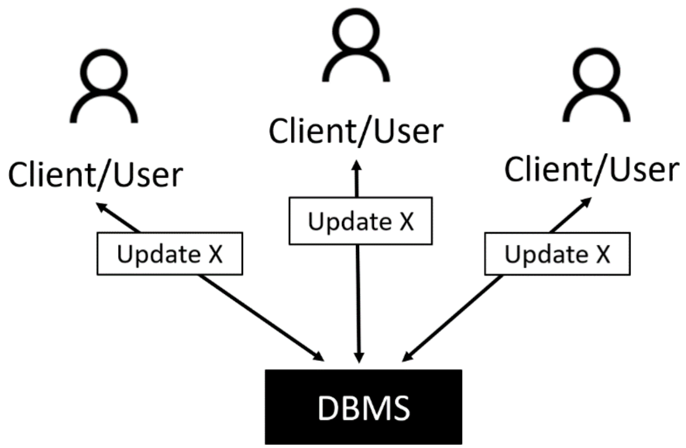
- ERD
- Entity
- attributes
- Connection between entities
- Connection types
- generalization

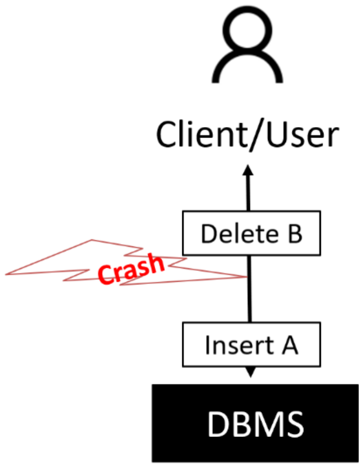
Challenges in a real database



Challenges in a real database

Challenges in a DB with **many users** that may **fail**





Transactions to save the day!

Lecture topics

- Problems in a multi-user DB that can fail
- **What is a transaction?**
  - What should a DBMS do with transactions?
- How DBMSs **implement transactions**?
- Trading data **consistency for speed**

Reichman University

Transactions

Example #1: Student Cleanup

Students

ID	Name	GPA	Status
1	Avi	82.5	Active
2	Benny	62.3	Rejected
3	Danna	91.2	Active
4	Ehud	65.3	Rejected

Archive

ID	Name	GPA	Status

Is it possible for a student to be in both tables?
(In a normal condition - **no!**)

Insert into Archive

Select * from Students Where Status = 'Rejected'

Delete From Students Where Status = 'Rejected'

Example #1: Student Cleanup – normal state

Students

ID	Name	GPA	Status
1	Avi	82.5	Active
3	Danna	91.2	Active

Archive

ID	Name	GPA	Status
2	Benny	62.3	Rejected
4	Ehud	65.3	Rejected

Insert into Archive

Select * from Students Where Status = ‘Rejected’

Delete From Students Where Status = ‘Rejected’

Reichman University

Example #1: Student Cleanup

Students

ID	Name	GPA	Status
1	Avi	82.5	Active
3	Danna	91.2	Active

Archive

ID	Name	GPA	Status
2	Benny	62.3	Rejected
4	Ehud	65.3	Rejected

What if the DBMS crashed after the INSERT?

Insert into Archive

Select * from Students Where Status = 'Rejected'

Delete From Students Where Status = 'Rejected'

Example #1: Student Cleanup

Students

ID	Name	GPA	Status
1	Avi	82.5	Active
2	Benny	62.3	Rejected
3	Danna	91.2	Active
4	Ehud	65.3	Rejected

Archive

ID	Name	GPA	Status

Insert into Archive

Select * from Students Where Status = ‘Rejected’

Delete From Students Where Status = ‘Rejected’

Reichman
versity

Example #1: Student Cleanup

Students

ID	Name	GPA	Status
1	Avi	82.5	Active
2	Benny	62.3	Rejected
3	Danna	91.2	Active
4	Ehud	65.3	Reject

Archive

ID	Name	GPA	Status
2	Benny	62.3	Rejected
4	Ehud	65.3	Rejected

Crash

Insert into Archive

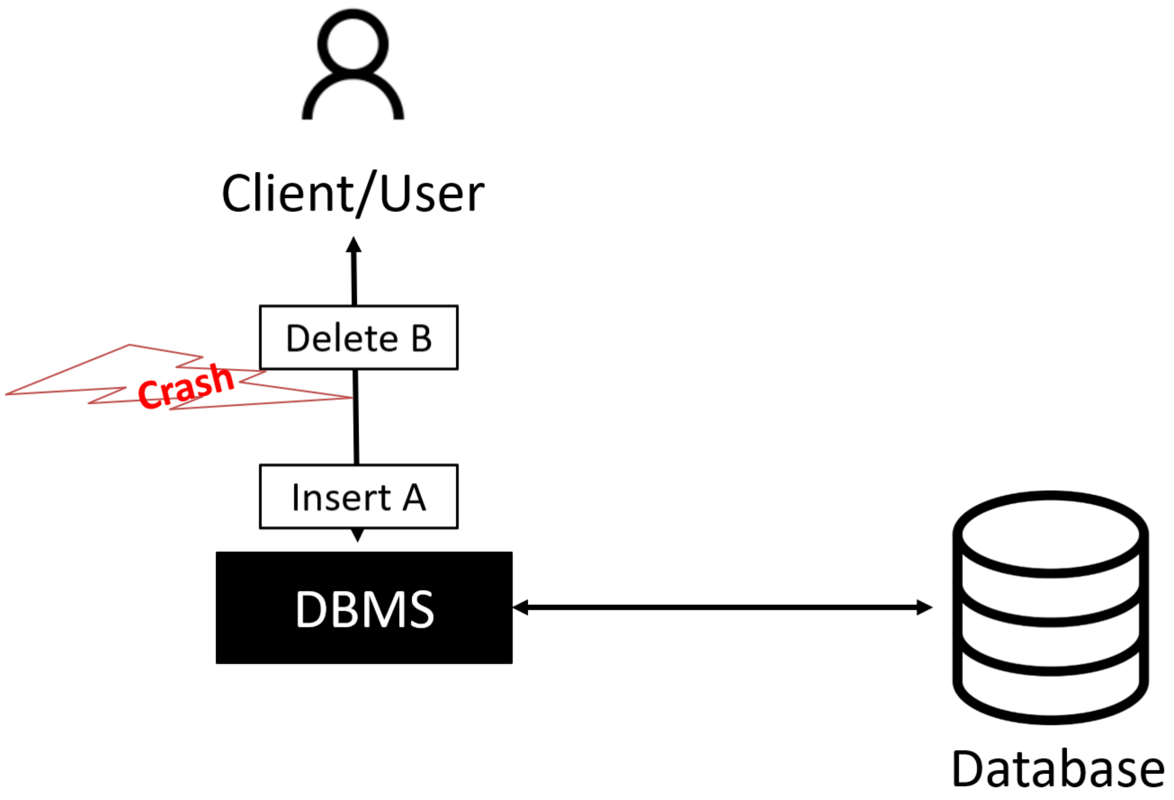
Select * from Students Where Status = ‘Rejected’

Delete From Students Where Status = ‘Rejected’

Example #1: Student Cleanup

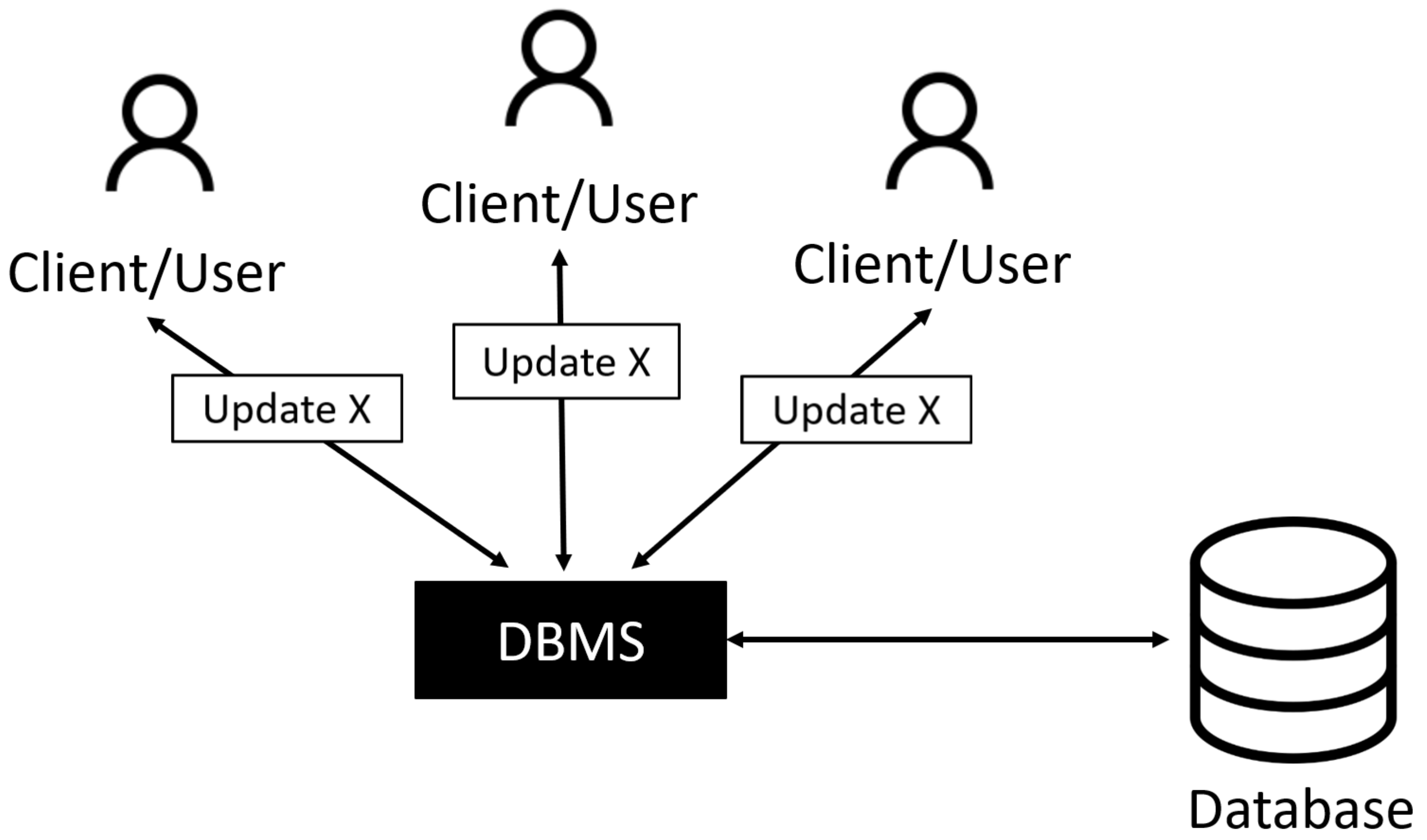
- There will be duplicate records both in Archive and Students tables
- If, we run again the query we get failure (duplicate records with the same primary key)
- We need a way to say execute **“all or nothing”**

Example #1: DBMS crash



Reichman University

Example #2: Multiple Users



Reichman University

How to preform an update

```
Update Employees
Set salary = salary + 1000
Where SSN= 7
```

Problem #1: Concurrent Updates

Update **Employees**
Set **salary = salary + 1000**
Where **SSN= 7**

Update **Employees**
Set **salary = salary + 1500**
Where **SSN= 7**

Concurrently

Is it 14,500?

Maybe 13,500 or 13,000???

Employees

SSN	Name	Salary
7	Danna	12,000

What is Danna's salary?

Reichman University

Problem #2: Concurrent Updates

Update **Employees**
Set **salary = salary + 2000**
Where **SSN= 7**

Update **Employees**
Set **salary = salary/2**
Where **SSN= 7**

Concurrently

Is it 7,000 or 8,000? (even worser than the previous example – no final stage)

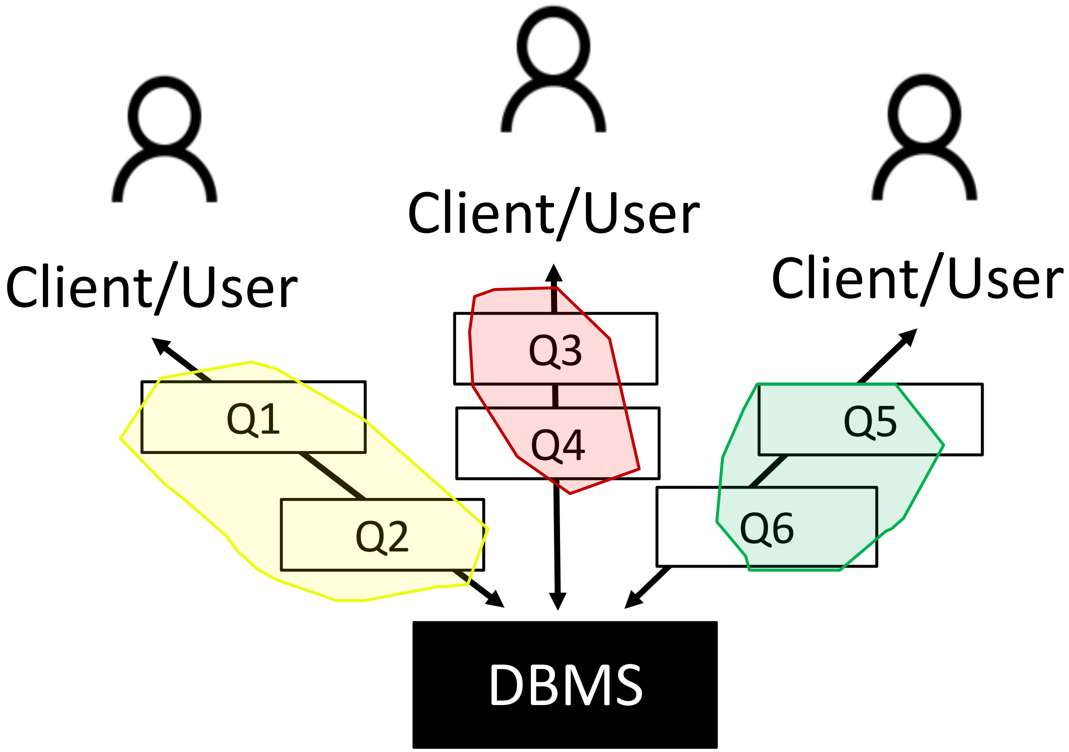
Employees

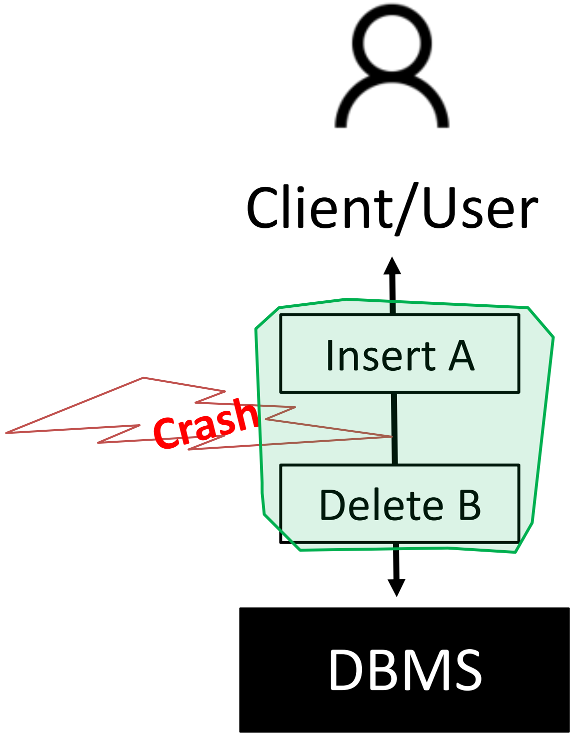
SSN	Name	Salary
7	Danna	12,000

Maybe 14,000 or 6,000???

What is Danna's salary?

Actions Need to be Grouped Together!





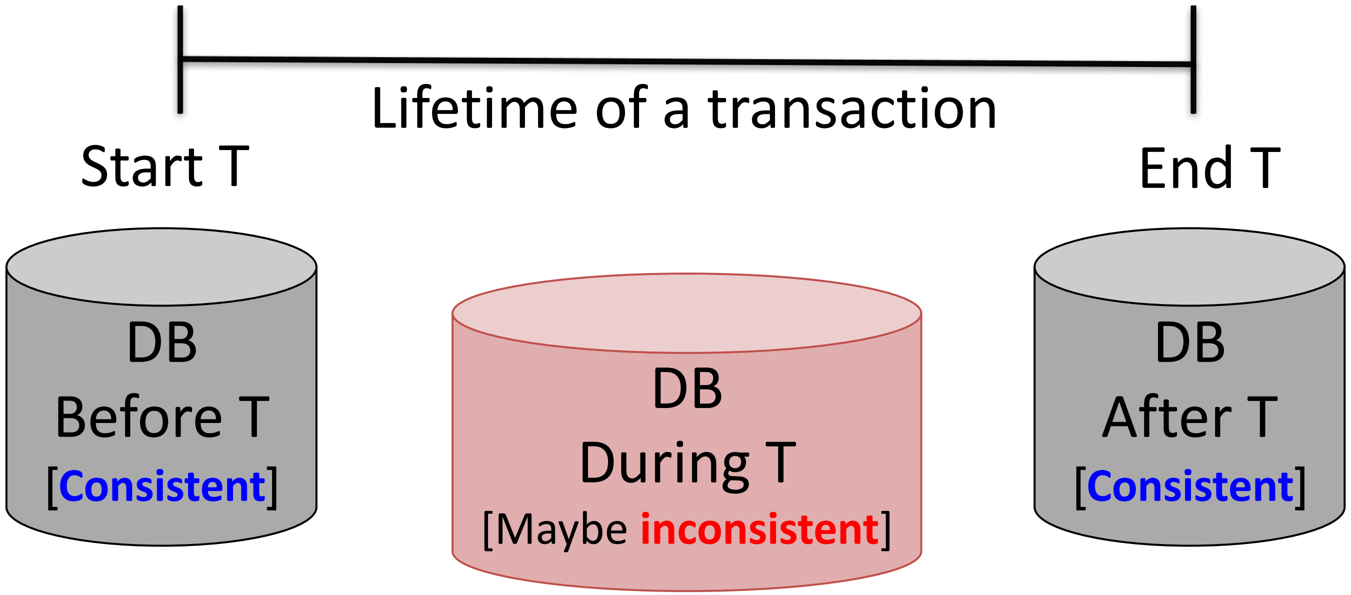
Reichman University

Transactions

Transactions to the Rescue!

Transaction: a **sequence of SQL operations** that are considered as a single unit

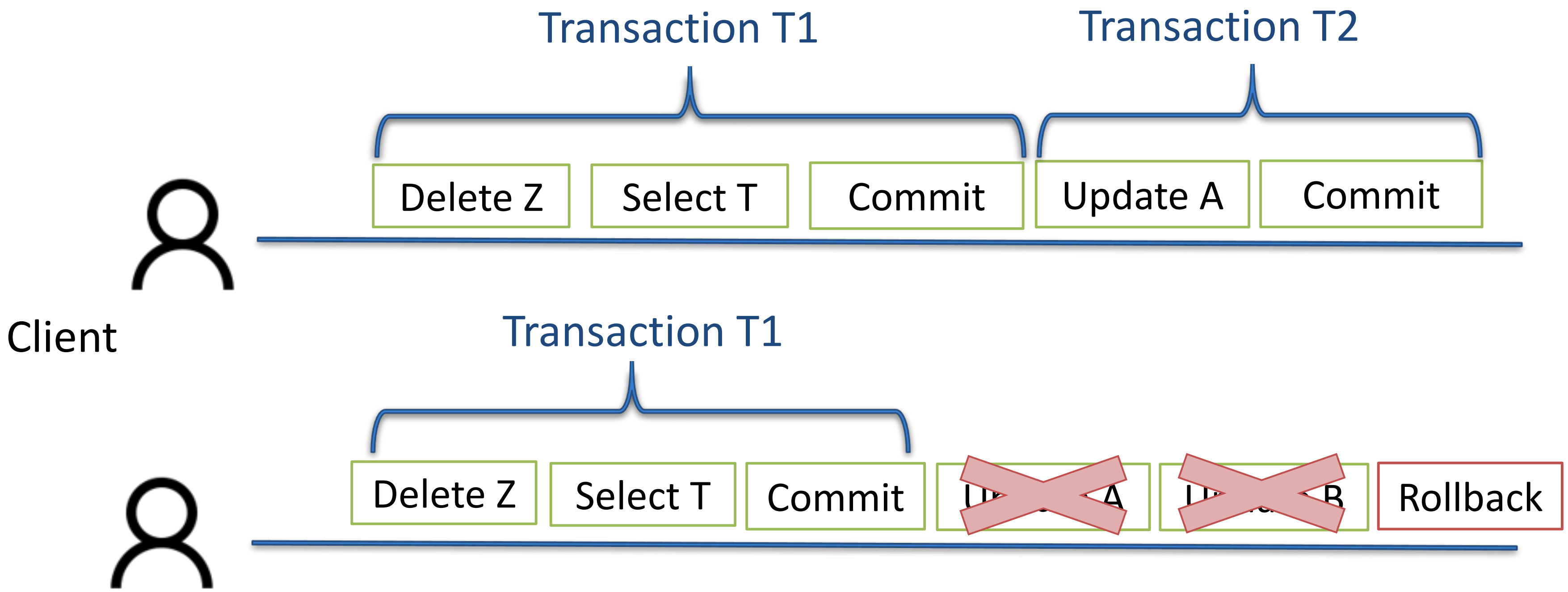
- In a transaction either **all actions are done or none**
- Transitions the DB from one consistent state to another



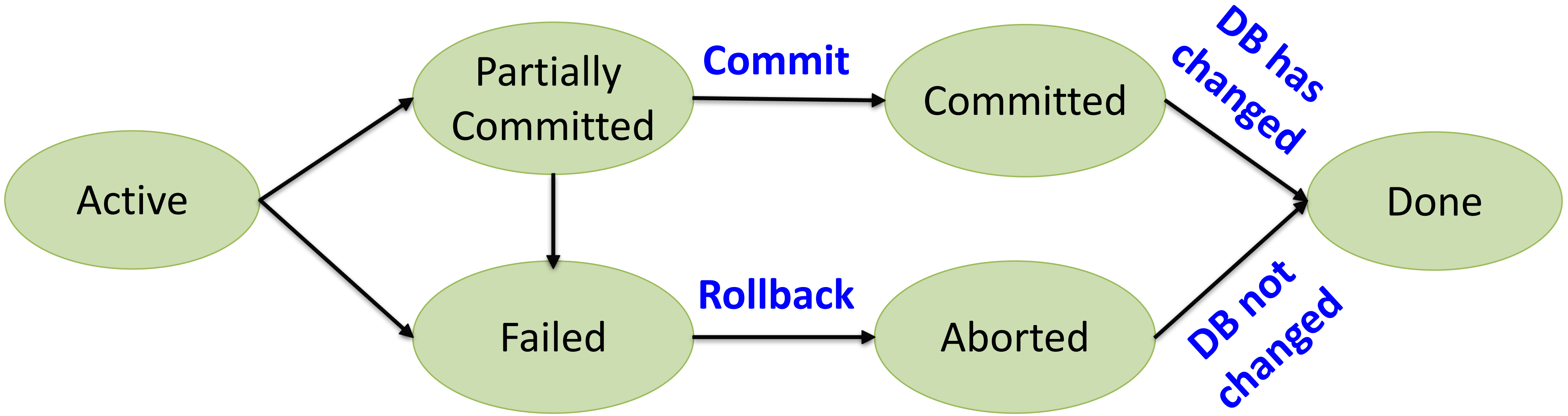
Transaction mechanism

- How we creates transactions?
  1. We apply SQL actions.
  2. When we want to “close” a Transaction we call this command **Commit**.
  3. If we want to “cancel” a started Transaction we do a **Rollback.**

- In many languages we can define auto commit



Transaction lifecycle



Reichman University

ACID

ACID: Desired Properties of Transactions

- Atomicity
- Consistency
- Isolation
- Durability

ACID: Atomicity

- Each transaction is “all-or-nothing” never left half done.
- What to do if an error occurs in the middle of a Transaction?
  - We perform a rollback to the Transaction to "clear" the changes

Reichman University

Atomicity

Students

ID	Name	GPA	Status
1	Avi	82.5	Active
2	Benny	62.3	Rejected
3	Danna	91.2	Active
4	Ehud	65.3	Reject

Archive

ID	Name	GPA	Status
2	Benny	62.3	Rejected
4	Ehud	65.3	Rejected

Crash

We solve this problem with atomicity as we return to the commit stage.

Insert into Archive

Select * from Students Where Status = ‘Rejected’

Delete From Students Where Status = ‘Rejected’

ACID: Consistency

Each client, each transaction:

- If all **constraints hold when transaction begins**
- Guarantee all **constraints hold when transaction ends**

- What kind of database constraints are there?

Referential integrity(e.g., cascade delete)

- What is an inconsistent state?

Reichman University

Atomicity vs. Consistency: are they the same?

- **Atomicity** - In an atomic transaction, a series of database operations either all occur, or nothing occurs. A guarantee of atomicity prevents updates to the database occurring only partially, which can cause greater problems than rejecting the whole series outright.

- **Consistency** - In database systems, a consistent transaction is one that does not violate any integrity constraints during its execution. If a transaction leaves the database in an illegal state, it is aborted, and an error is reported.

Reichman University

Atomicity vs. Consistency: are they the same? (Simple words)

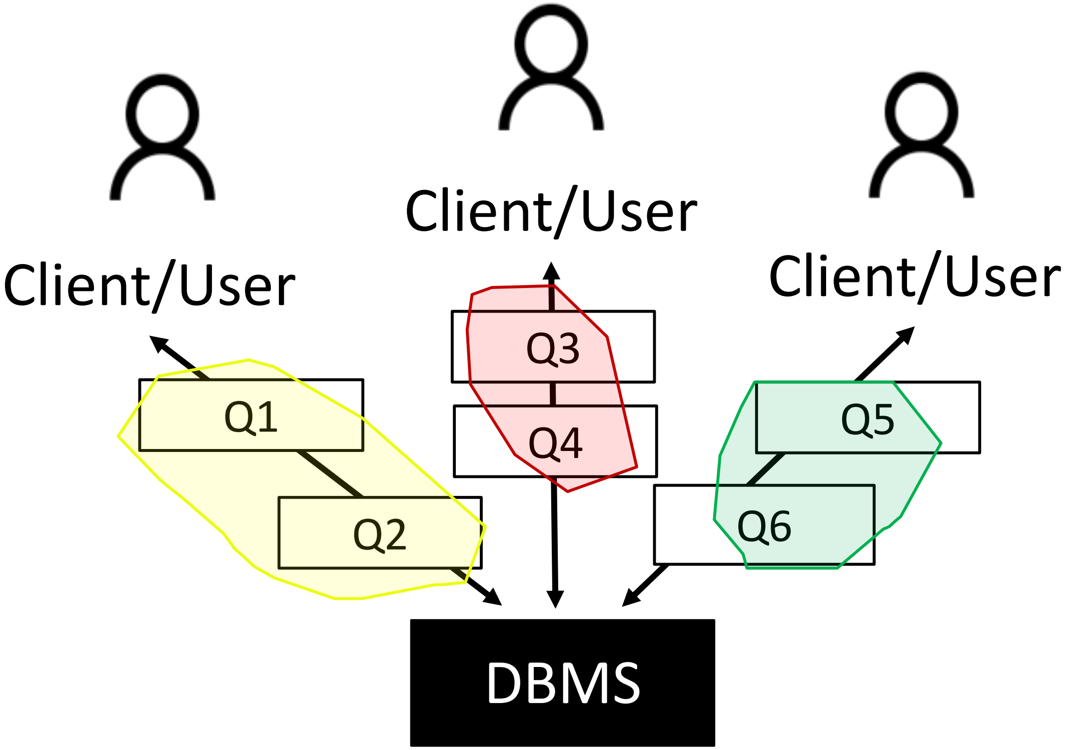
- Atomicity – all the transaction or nothing.

- Consistency – guarantee all constrains in the database. Such as, Referential integrity (e.g., cascade delete)

ACID: Isolation

Isolation

The transaction will behave as if it is the only operation being performed upon the database



Problem #1: Concurrent Updates

Update **Employees** Set **salary = salary + 1000** Where **SSN= 7**

Update **Employees** Set **salary = salary + 1500** Where **SSN= 7**

Concurrently

Is it 14,500?

Maybe 13,500 or 13,000???

Employees

SSN	Name	Salary
7	Danna	12,000

What is Danna's salary?

Problem #2: Concurrent Updates

Update **Employees**
Set **salary = salary + 2000**
Where **SSN= 7**

Update **Employees**
Set **salary = salary/2**
Where **SSN= 7**

Concurrently

Is it 7,000 or 8,000? (even worser than the previous example – no final stage)

Employees

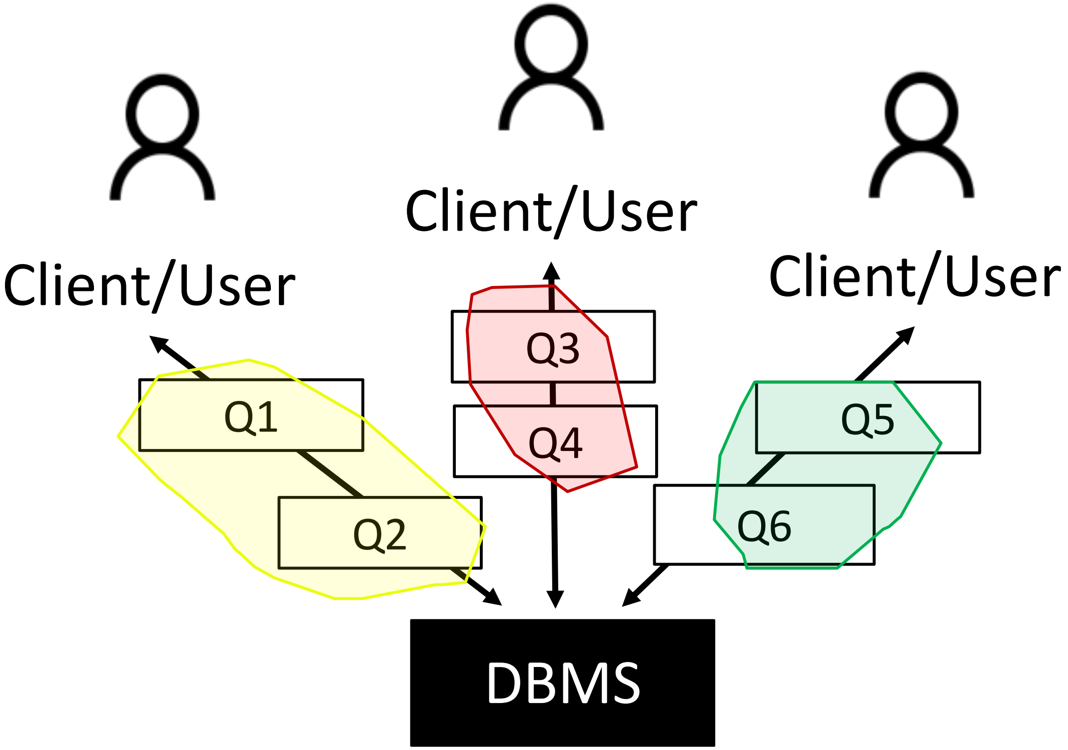
SSN	Name	Salary
7	Danna	12,000

Maybe 14,000 or 6,000???

What is Danna's salary?

ACID: Isolation

- Serializability - Execution must be equivalent to some sequential (serial) order of all transactions
- Often implemented by a locking mechanism



Problem #2: Concurrent Updates

- What is the right answer?

```
Update Employees
Set salary = salary + 2000
Where SSN= 7
```

```
Update Employees
Set salary = salary/2
Where SSN= 7
```

Concurrently

Employees

SSN	Name	Salary
7	Danna	12,000

What is Danna's salary?

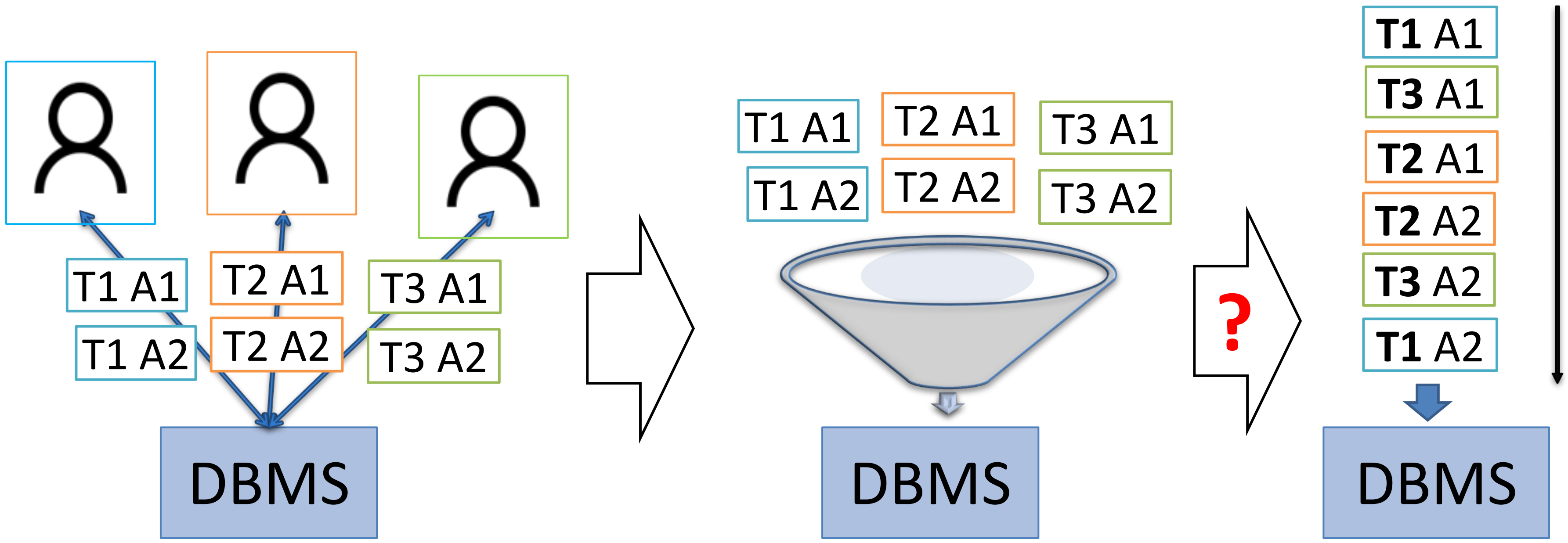
- The thing is there is no one answer…
- With Isolation we will get both 7,000 and 8,000 as valid answers but not 14,000 and 6,000.
- As we make sure that transaction preformed in a serialize manner.

Reichman University

Implementing Transactions on DMBS

Goal: maintaining Isolation

How to preserve Isolation? (while executing concurrently)



Scheduling

- Scheduling is the execution of several Transactions in parallel.
- Only one action is performed at each point in time.
- A reading / writing operation is performed in an atomic form.
- The timing represents the order on the timeline (the vertical axis) in which the actions are performed.

	Read(A)
Read(C)	
	Write(A)
	Read(B)
Write(C)	
Read(B)	
Write(B)	
	Write(B)

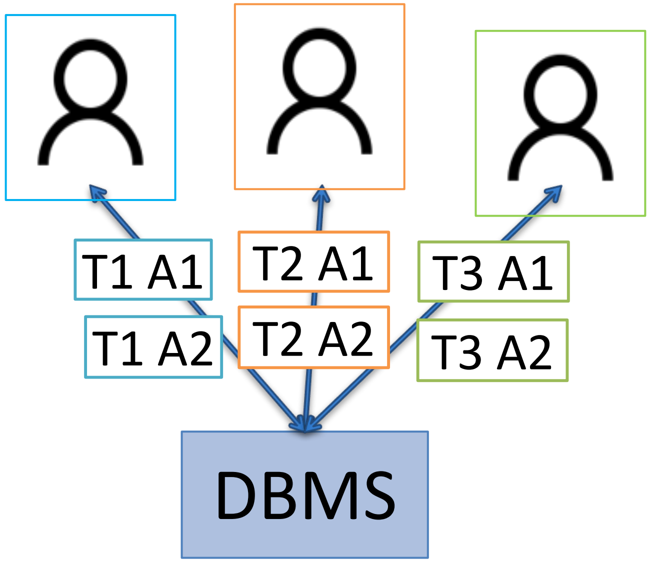
Serializable scheduling

- Scheduling where transactions are performed one after the other.
- Serial timing is correct. (serializable)

Read(A)	
Write(A)	
Read(B)	
Write(B)	
	Read(C)
	Write(C)
	Read(B)
	Write(B)

What is Isolation?

- What is a ***correct*** schedule?



Sequential schedules (s1, s2, s3) **?** (s4, s5)

s1	s2	s3	s4	s5
T1 A1	**T3** A1	**T2** A1	**T2** A2	**T1** A1
T1 A2	**T3** A2	**T2** A2	**T2** A1	**T2** A1
T2 A1	**T1** A1	**T3** A1	**T1** A2	**T2** A2
T2 A2	**T1** A2	**T3** A2	**T3** A1	**T3** A1
T3 A1	**T2** A1	**T1** A1	**T1** A1	**T1** A2
T3 A2	**T2** A2	**T1** A2	**T3** A2	**T3** A2

Serializable schedule

- A schedule is a serializable schedule if it is equivalent to any serial scheduling.
- Serializable schedule is a correct schedule.
- Need a way to determine that a schedule is serializable without knowing what calculations are being performed.
- That is, for every possible calculation, the timing should be equivalent to a serial scheduling.

Reichman University

Equivalent schedules

Two schedules are equivalent if:

- They are composed of the same actions.
- Both schedules have the same effect on the database, meaning that the database has the same values at the end of each of the schedules.
- Both schedules produce the same output to the user, i.e., present the user with the same values for each item they read.

Reichman University

How the DBMS Achieves Isolation?

Serializability

Operations may be interleaved, but execution must be equivalent to *some* sequential (serial) order of all transactions

How?

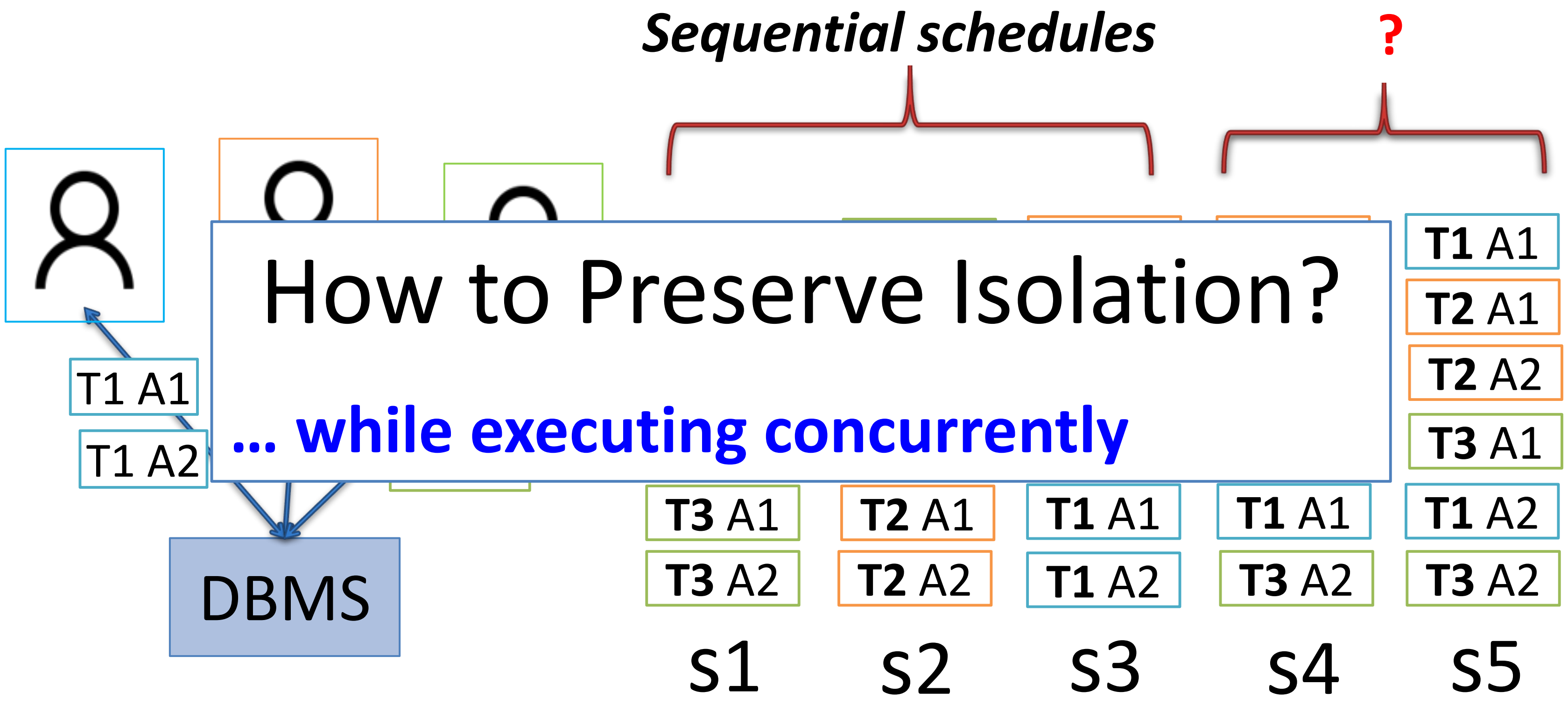
DBMS achieve isolation by

enforcing **all schedules to be serializable**

Reichman University

Implementing Isolation: Solution A

- Only allow sequential schedules



Reichman University

Implementing Isolation: Solution B

1. Generate a schedule
2. Check if it is serializable

How to check if a schedule is serializable?

Reichman University

Abstraction for computing Isolation

- How do we check if a parallel execution is correct?
  - It depends on the exact calculation of each program execution.
  - Too complicated* to analyze the exact calculation that each calculation performs.

- Thus, we refer only to the abstraction of the programs:
  - Reading operations from the database.
  - Writing actions into the database.
  - The sequence, on the timeline, in which the actions are performed.

Programs	
Read(A)	
A:=A-100	
	Read(C)
Write(A)	
	C:=C-100
Read(B)	
	Write(C)
B:=B+100	
	Read(B)
	B:=B+100
	Write(B)
Write(B)	

*In fact, it is not possible

Reichman University

Program abstraction

Actual program		Abstraction	
Read(A)		Read(A)	
A:=A-100			
			ad(C)
Write(A)			
Read(B)		Read(B)	
	Write(C)		Write(C)
B:=B+100			
	Read(B)		Read(B)
	B:=B+100		
	Write(B)		Write(B)
Write(B)		Write(B)	

We focus only on the read and write operations

eichman
ersity

How to Check if a Schedule is Serializable?

Approach: check ordering constraints

Schedule #1

T1	T2
Read(A)	
	Read(C)
Write(A)	
	Write(C)
	Read(B)
	Write(B)
Read(B)	
Write(B)	

Schedule #2

T1	T2
Read(A)	
	Read(C)
Write(A)	
Read(B)	
	Write(C)
	Read(B)
	Write(B)
Write(B)	

How to Check if a Schedule is Serializable?

Is it serializable?

Schedule #3

T1	T2
Read(A)	
	Read(C)
Write(A)	
Read(B)	
	Write(C)
	Read(B)
Write(B)	

Schedule #2

T1	T2
Read(A)	
	Read(C)
Write(A)	
Read(B)	
	Write(C)
	Read(B)
	Write(B)
Write(B)	

How to check if schedule is serializable?

- Definition: Precedence graph
  - Every transaction is a vertex.
  - There is an edge from T_i to T_j if and only if one of the following conditions is met:
    - T_i performs Read(x) before T_j performs Write (x)
    - T_i performs Write(x) before T_j performs Read(x)
    - T_i performs Write (x) before T_j performs Write(x)

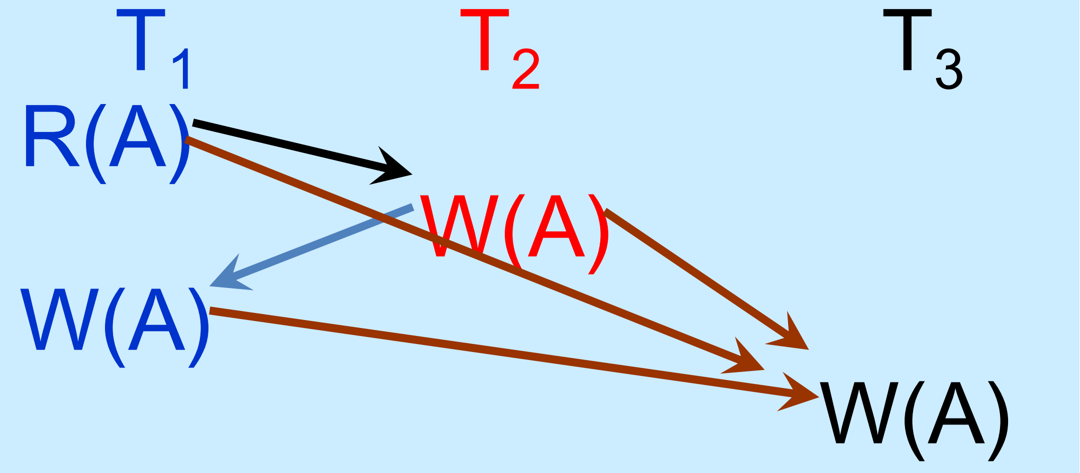
    - Note: read(X) read(X) do not add an edge

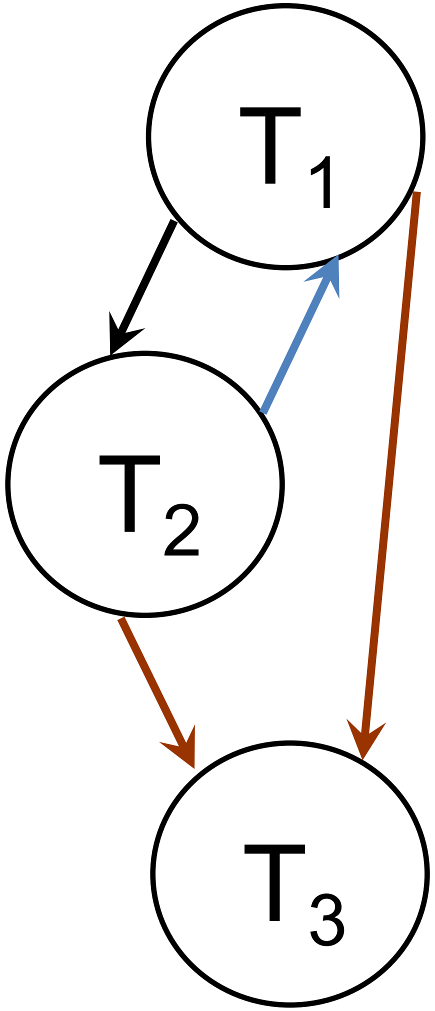
Reichman University

How to check if schedule is serializable?

- Scheduling is serializable if there are no circles in the precedence graph.
  - How to check if there are circuits (algorithm course)?
  - Is it possible to have a serializable schedule in a situation where there are circuits?

Precedence graph - illustration





Equivalent serial scheduling:

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

https://www.youtube.com/watch?v=U3SHusK80q0

Implementing Isolation: Solution C

- Goal: create serializable schedules

Approach #1: using **locks**

- What are we locking and why?
- What types of locks do we need?
- When to use which lock?

Exclusive and Shared Locks

- **Exclusive** lock
  - Only one transaction can hold it at time
  - When to use?

- **Shared** lock
  - Many transaction can hold this lock
  - When to use?

When we lock a key?

- To read item A you must first lock it in a shared lock.
  - It is possible to release the lock after reading.
- To write item A you must first lock it with an exclusive lock.
  - It is possible to release the lock after writing.
- If after reading need to write, then transaction can upgrade a shared lock to an exclusive lock.
- Requests for locks are directed to the lock manager.

Reichman University

When is it allowed to lock a key?

- Multiple transactions can simultaneously hold a shared lock on item A.
  - Provided there is no transactions holding an exclusive lock on A.
- Only one transaction can hold an exclusive lock on item A.
  - Then no other transaction has a shared or exclusive lock on A.
- The lock manager makes sure the rules are followed.

Reichman University

Lock manager

- The lock manager handles lock requests and lock releases.
- Holds a table that has an entry for each item A with:
  - List of transactions currently holding a lock on A.
  - Type of the lock.
  - Pointer to queue of lock requests on A.
- Receiving a lock and releasing a lock should be done as atomic operations.
- A common lock can be upgraded to an exclusive lock.

How and When to Lock?

Schedule #2	
T1	**T2**
Read(A)	
	Read(C)
Write(A)	
Read(B)	
	Write(C)
	Read(B)
	Write(B)
Write(B)	

Lock Type	Owner	Data item
Shared / Exclusive	T1	A
Shared / Exclusive	T2	C
Shared	T1	B
Shared / Exclusive	T2	B
Exclusive	T1	B

T1 must release its shared key on B before T2

T2 must release its exclusive key on B before T1

Reichman University

Concurrency Control with Locking

- Goal: **create serializable schedules**
- Approach: using locks

2 Phase Locking (2PL)!

Two-Phase Locking (2PL)

Two-Phase Locking (2PL)

- Transaction must get:
  - Shared lock before reading an item.
  - Exclusive lock before writing an item.

- Once a transaction releases a lock it cannot request additional locks.
- That is why the protocol is called "two-phase locking“
  - First a phase of making locks.
  - Followed by a phase of release of locks.

Reichman University

2PL Example

Is 2PL?	
T1	**T2**
S(A) Read(A)	
	S(C) Read(C)
X(A) Write(A)	
S(B) Read(B)	
	X(C) Write(C)
	S(B) Read(B)
	X(B) Write(B)
X(B) Write(B)	

Not a 2PL!

Is 2PL?	
T1	**T2**
S(A) Read(A)	
	S(C) Read(C)
X(A) Write(A)	
S(B) Read(B)	
X(B) Write(B)	
T1 release all its keys	X(C) Write(C)
	S(B) Read(B)
	X(B) Write(B)

2PL

chman University

The Downside of Using Locks Is …

Deadlocks

How handle deadlocks?

Detect & handle

Avoid

Deadlock

- A state of deadlock occurs when there is a circle of transactions, so that every transaction in the circle waits for a lock that is currently held by the next transaction in the circle.

- Two ways to deal with deadlock:
  - Prevention of deadlock.
  - Detection of deadlock and cancellation.

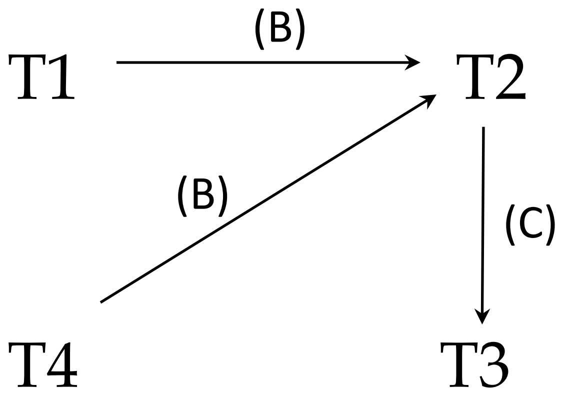
Reichman University

Is there a Deadlock here?

T1:	S(A), R(A),		S(B)					
T2:		X(B),W(B)				X(C)		
T3:				S(C), R(C)				X(A)
T4:					X(B)			

How to find a Deadlock?

T1:	S(A), R(A),		S(B)				
T2:		X(B),W(B)				X(C)	
T3:				S(C), R(C)			X(A)
T4:					X(B)		



Detection of deadlock – option 1

- Build a "wait for" graph
  - Node for every transaction.
  - from T_1 to T_2 if T_1 is waiting for a lock that T_2 currently holds.
- From time-to-time check if there is a circle in the graph (which means a state of deadlock)
- If there is a circle, cancel one of the transaction that is on the circle, and start it again.

Detection of deadlock – option 2

- Timeout method - after a certain amount of time without progress in making the transactions, it can be assumed that there is a Deadlock.

- Release from Deadlock is done by performing ABORT on at least one of the transactions.

Reichman University