1. What are transactions
2. Properties of transactions → ACID → Consistency, Durability, Atomicity, Isolation
3. How do transactions Work?
   - → Read
   - → Write
4. Commits and Rollback
5. Transaction Isolation Levels.
   - → Read Uncommitted
   - → Read Committed
   - → Repeatable Read
   - → Serializable

Mock Interview.

Platform

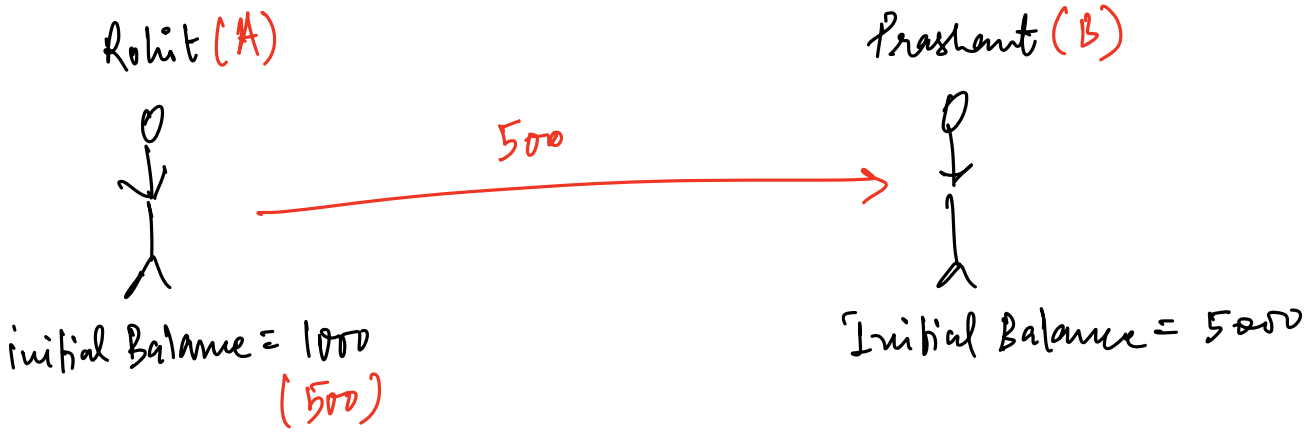
MySQL and	DSA, LLD
DSA and	LLD
HLD and	DSA

Rohit
DSA and LLD.
MySQL and DSA and LLD.
MySQL, DSA → none.

Transactions

select * from students where psp > 90; ✗ ⟶

Transferring money in a Bank.



accounts

id	name	balance	created_at	branch

CRUD

Steps

1. Get the balance of (DB call)
2. Check if the balance >= 500 (Not a DB call)
3. Reduce the balance of A by 500 (DB call)
4. Increase the balance of B by 500 (DB call)

```
transfer_money (from, to, amount) {
    // multiple SQL Queries with conditionals.
}
```

A → B
C → D
E → B
D → A

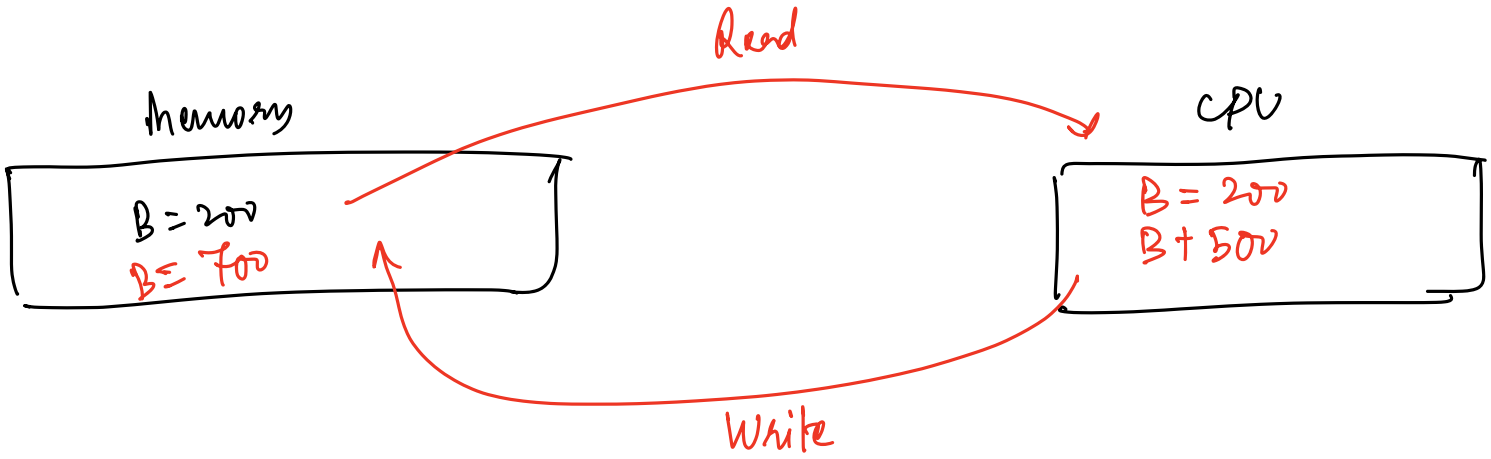
example:- A → B, C → B

B += 500 → B = B + 500 → executes in one operation?

① read the current value of B from memory (temp) (Read)
② Increase the value of temp by 500, temp = temp + 500
③ Write the temp value back to B. (Write)

```
transfer_money (a, b, amount) {
    read A → x (Reading from DB)
    if (x >= amount) {
        Write A ← x - amount (Write to DB)
        read B → x (Read from DB)
        x = x + amount
        Write B ← x ; (Write to DB)
    }
}
```

}



What could go wrong

#1

A → B

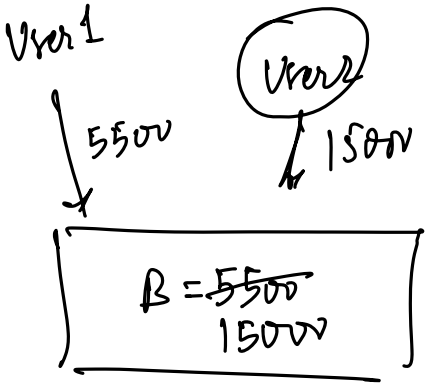
C → B

initially

A = 1000

B = 5000

C = 15000



A → B

a = 1000, b = 5000, amount = 500

C → B

C = 15000, b = 5000, amount = 10000

① a b

② C b

```
transfer_money (a, b, amount) {
    read A → X
    if (X >= amount) {
        write A ← X - amount
        read B → X
        X = X + amount
        write B ← X ;
    }
}
```

amount = 500

A → B

X = 1000

A = 500

X = 5000

X = 5500

B = 5500

amount = 10000

C → B

X = 15000

C = 5000

X = 5000

X = 15000

B = 15000

Before this transactions ⇒ a + b + c = 1000 + 5000 + 15000 = 21000

After ... ⇒ a + b + c = 500 + 15000 + 5000 = 20500

500 + 5500 + 5000

⇒ 15500

Loss of 500 rupees

#2

If DB machine goes down at line

A ← x-amount.

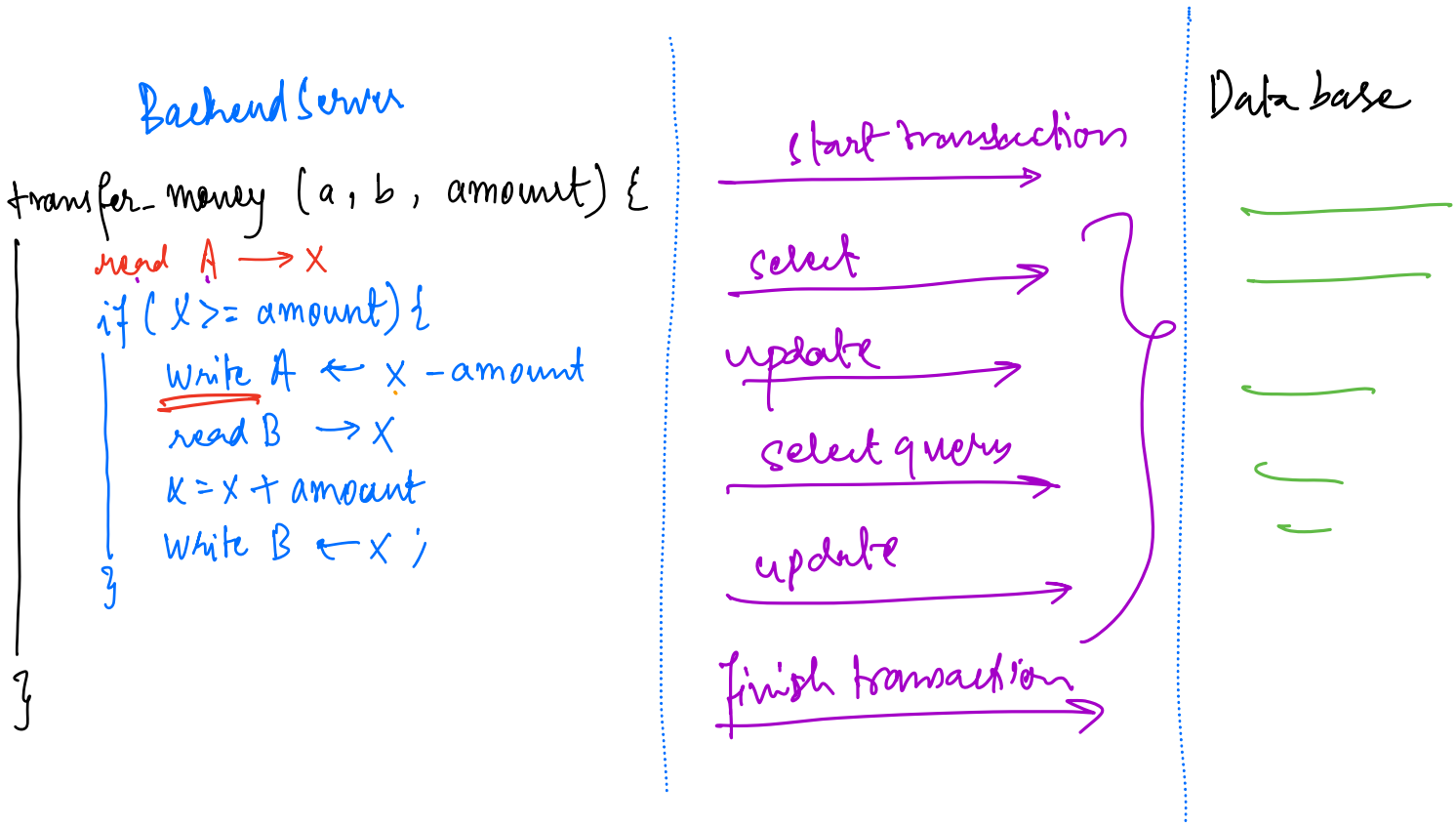
Money is deducted from A but never sent to B.

Transactions

① Inconsistent / Illogical State
② Complete operation may not execute

} → Transaction tries to solve these 2 problems.

Transactions is a set of DB operations logically grouped together to perform a task.


Backend Server
transfer_money (a, b, amount) {
read A → X
if (X >= amount) {
Write A ← X - amount
read B → X
X = X + amount
Write B ← X ;
}
}
start transaction
select
update
select query
update
finish transaction
Database

ACID Properties.

ACID properties may or may not be there in a transaction.

- → Atomicity (A)
- → Consistency (C)
- → Isolation (I)
- → Durability (D)

Atomicity

↓

indivisible unit

Consistency

In our Bank transfer example, Atomicity is taken care of.

In the Bank example, consistency is important.

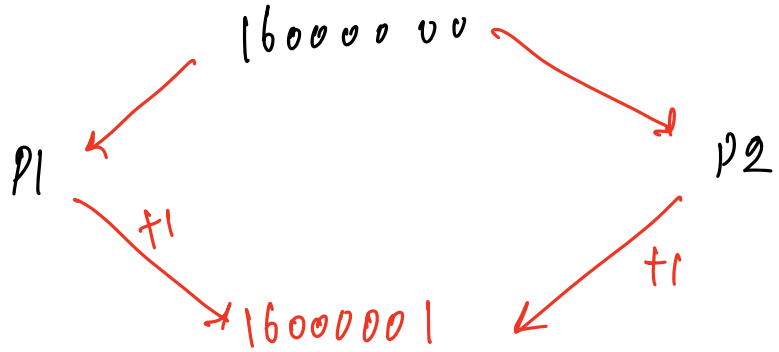
Stream_id	count
1	16000000

① select * from hotstar where stream_id=1

② update the count.

transaction

=



Consistency is something that might be expected, but not always. It depends on the usecase.

→ Bank example
→ Train Tickets
→ Book My Show

Isolation

Durability

Once a transaction is completed. We would want it's work to be persistant.

Whenever a transaction is complete, whatever the result was should be stored on disk.

① Saving to disk takes time
② By default, every DB may not update to the disk.

2 hours.

ACID Properties - Summarized.

Atomicity → Everything happens or nothing happens.

Consistency → Whatever happens should be accurate

Isolation → Transactions should not interfere

Durability → Once the outcome is calculated, it should be persisted

Commits and Rollbacks.

Students.

id	name	psp	batch
1	Alok	70	2
-	-	-	-
-	-	-	-

```
update students
SET psp = 80
where id = 1;

select * from students;
```

When I write a SQL query, it automatically gets updated to the DB.

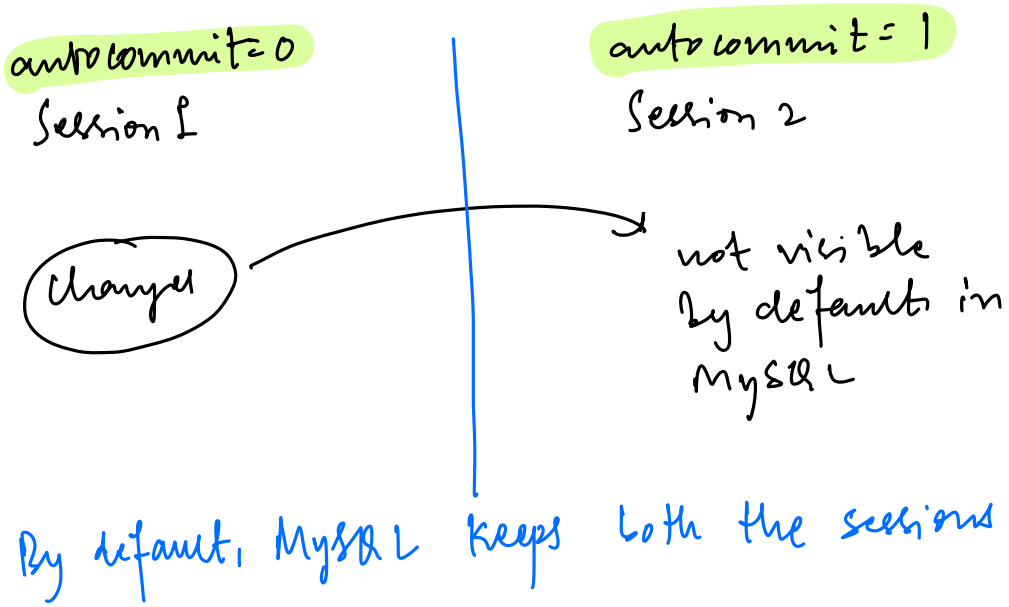
↓

autocommit

By default, autocommit = 1

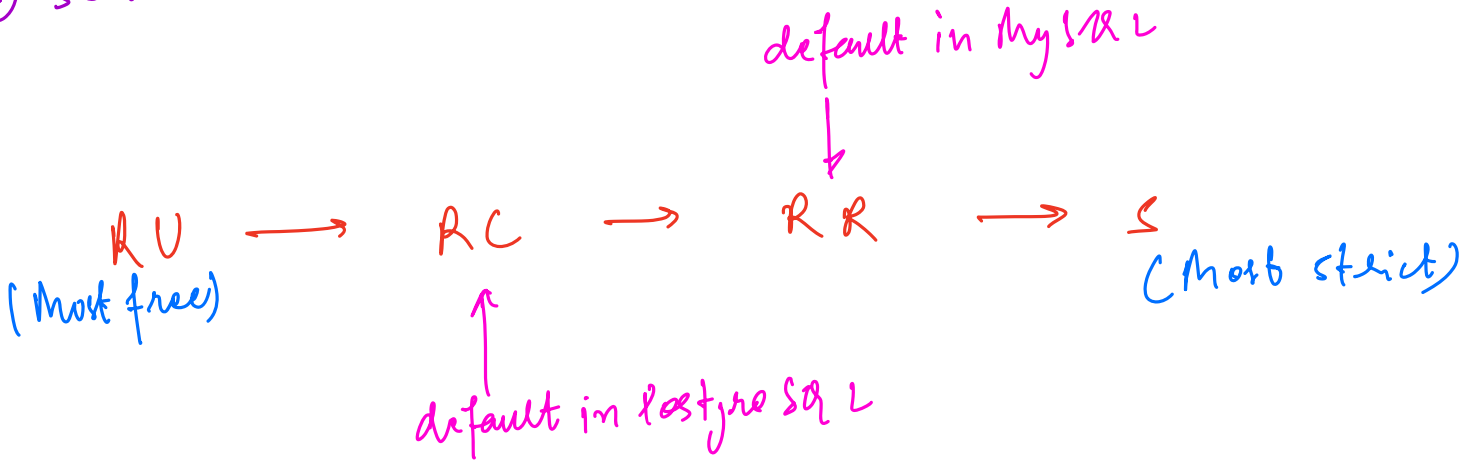
commit (important keyword) → used/executed to persist the results of a SQL query. (Durability)

Transaction Isolation Levels.



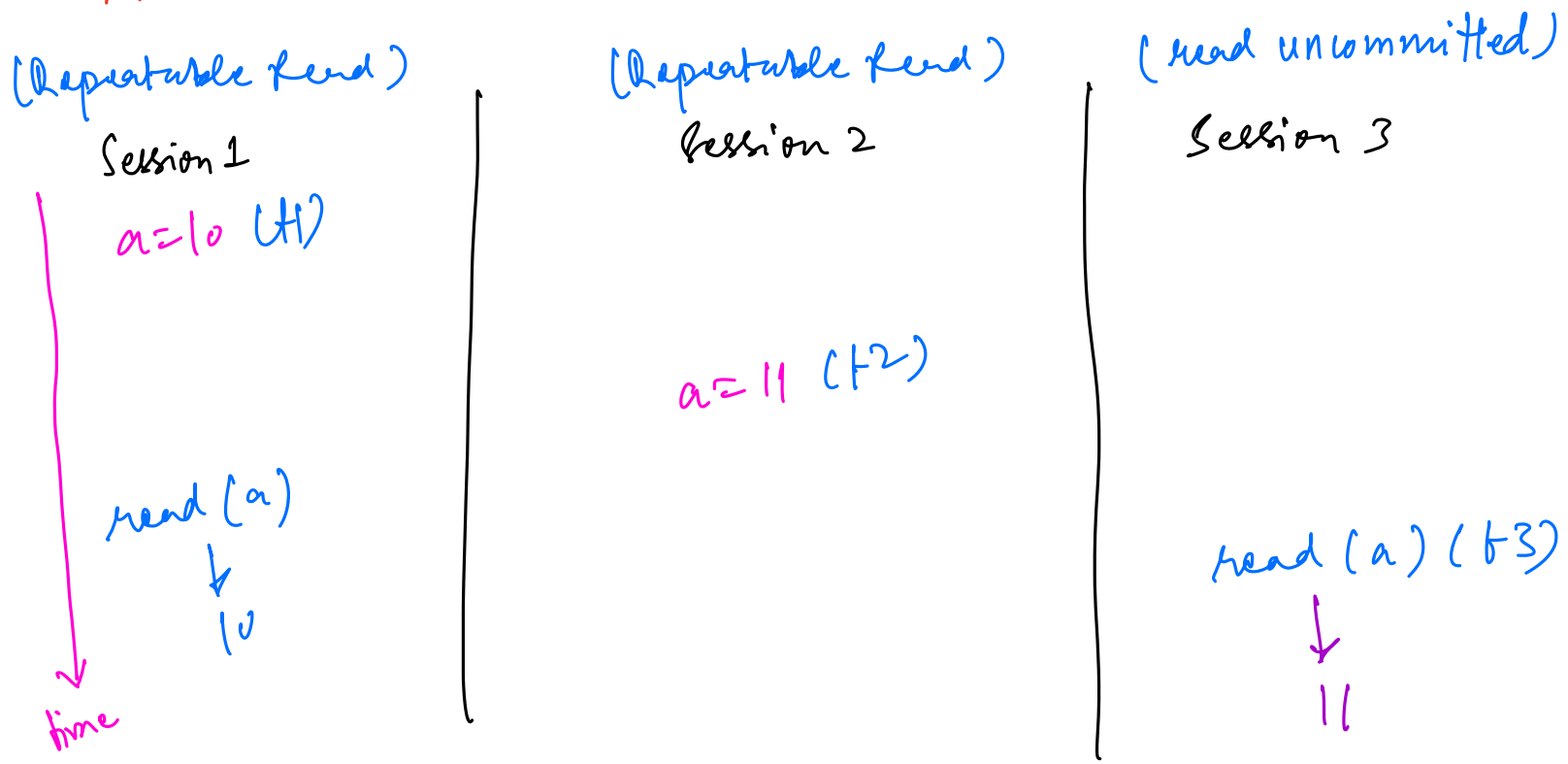
By default, MySQL keeps both the sessions isolated.

1. Read Uncommited (RU)
2. Read Commited (RC)
3. Repetable Read (RR)
4. Serializable (S)



Read Uncommited

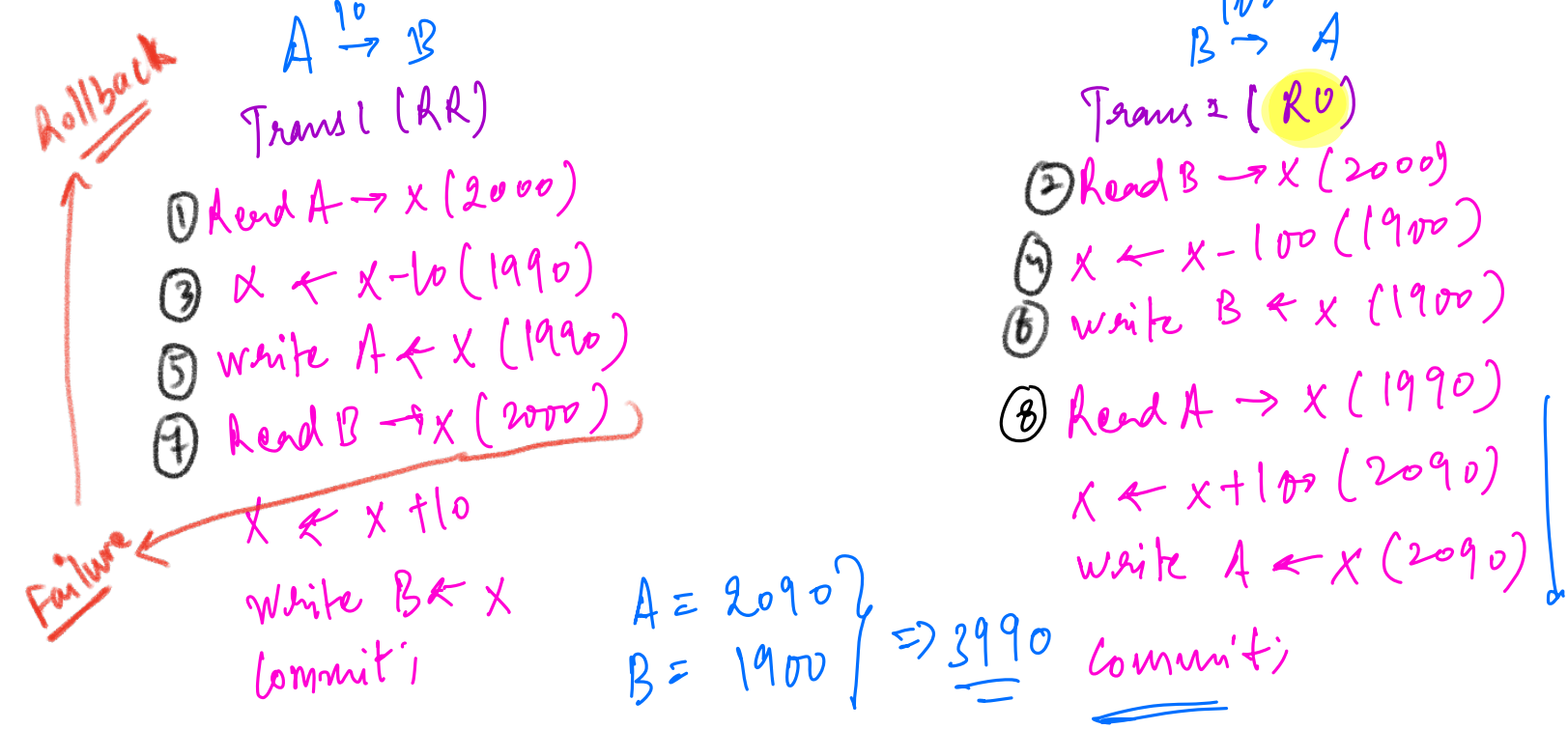
→ Reads latest data (whether committed or uncommitted)



Pros → Fast

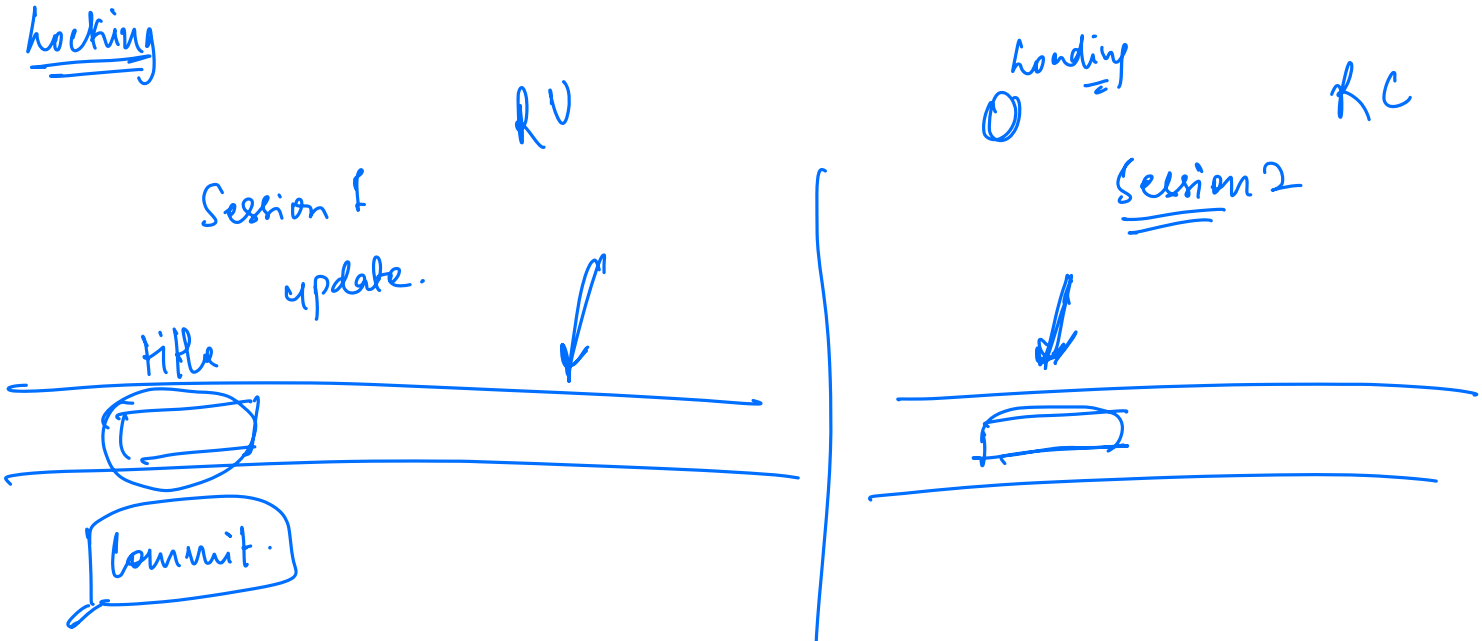
Cons → This may lead to inconsistency.

A = 2000 } → 4000
B = 2000 }



This is Dirty Read.

Read Committed.



There are 2 types of locking:-

1) Shared lock (Caused by a read query)

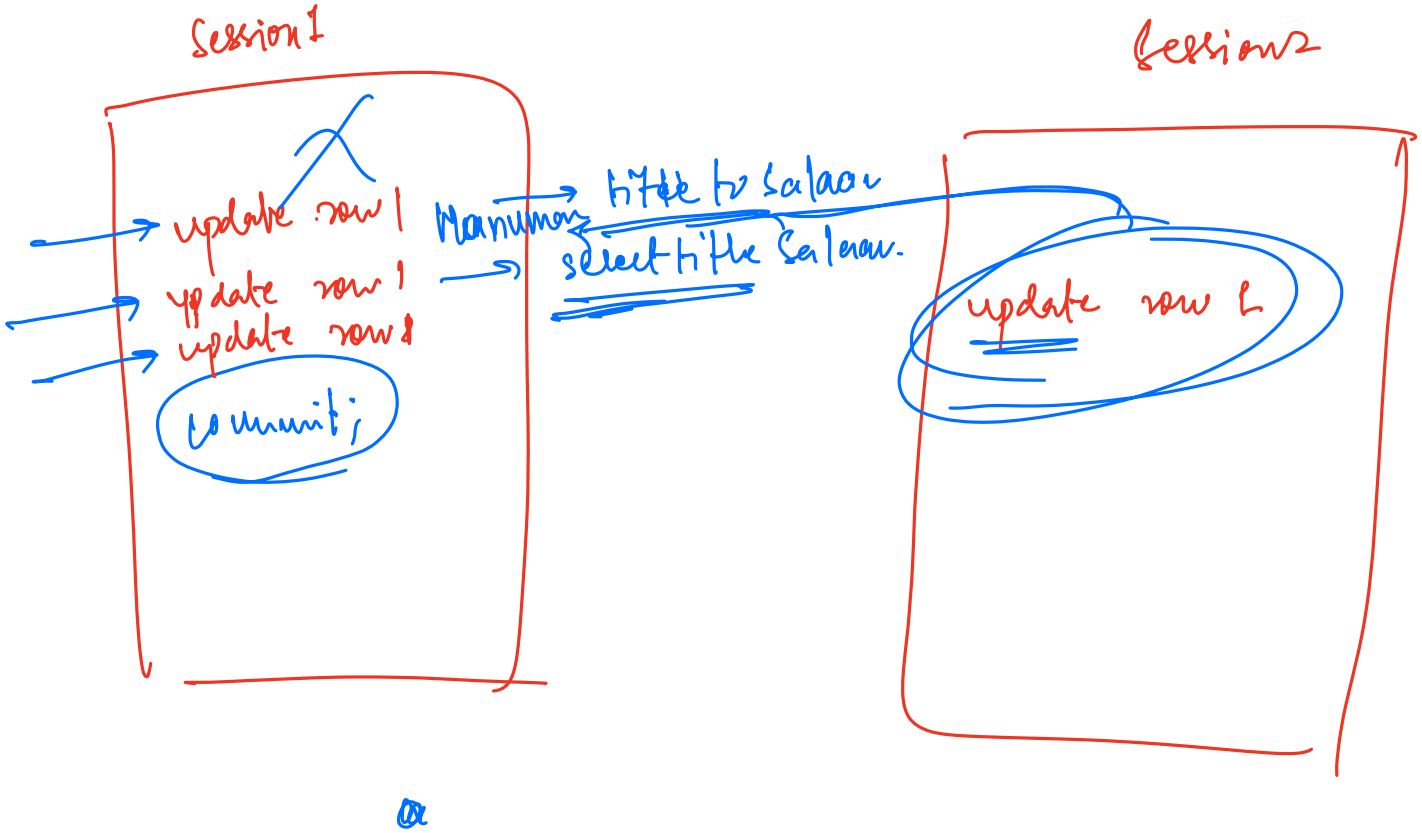
2) Exclusive lock (Caused by a write query)

Someone has	Someone else can
shared lock	read and write also on the same row
exclusive lock	read the same row but can't write

If one transaction has written on a row, no other transaction will be allowed to write on that row.

Summary (Read Committed)

1. A transaction will always read committed value
2. It will never read an uncommitted value
3. This means dirty reads will not happen.



Problems with Read Committed

Users

id	name	psp	email-sent
1	N	60	false → true
2	A	80 → 79	false
3	M	70	false → true
4	AB	95	false → true
5	XY	0	false
6	XYZ	39	

Phantoms

① Get list of users with psp < 80

list<user> = Select * from users where psp < 80;

② Send email to the obtained list of users (5 minutes)

③ Update the 'email-sent' column for users to whom the mail has been sent.

Update users
set email-sent = true
where psp < 80;

Sofia, Rahul, Rishi, Tanmay.
79 60 80 → 79. 69
↓ ↓ ↓ ↓
true true true true

→ this is called the problem of **non-repeatable read.**

Ideal Case?

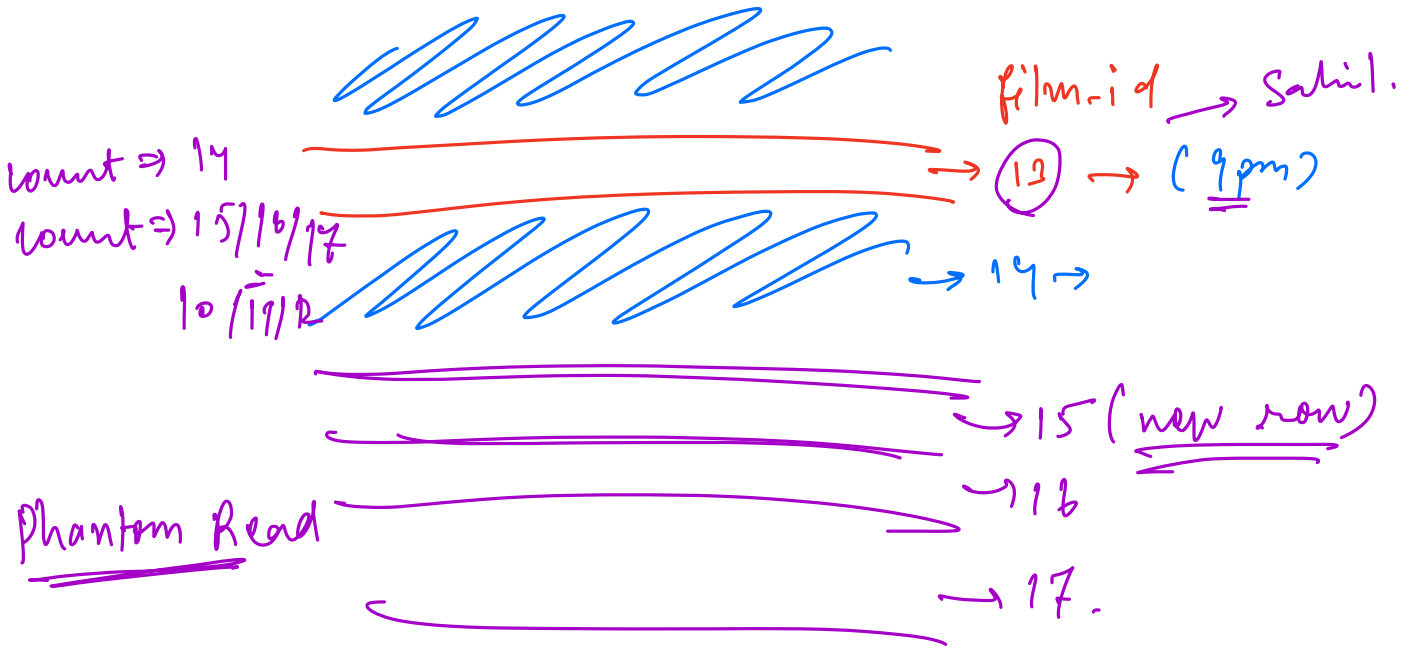
⇒ Within the same transaction, if I read a value twice it should must have the same value.

Repeatable Read

(i) For the very first time, it reads the latest committed value of the current row.

(ii) After that, until the transaction completes, it keeps on reading the same value it read the first time.

New Problem



How to solve? → Serializable

Serializable

You are only allowed to read rows that are not locked.