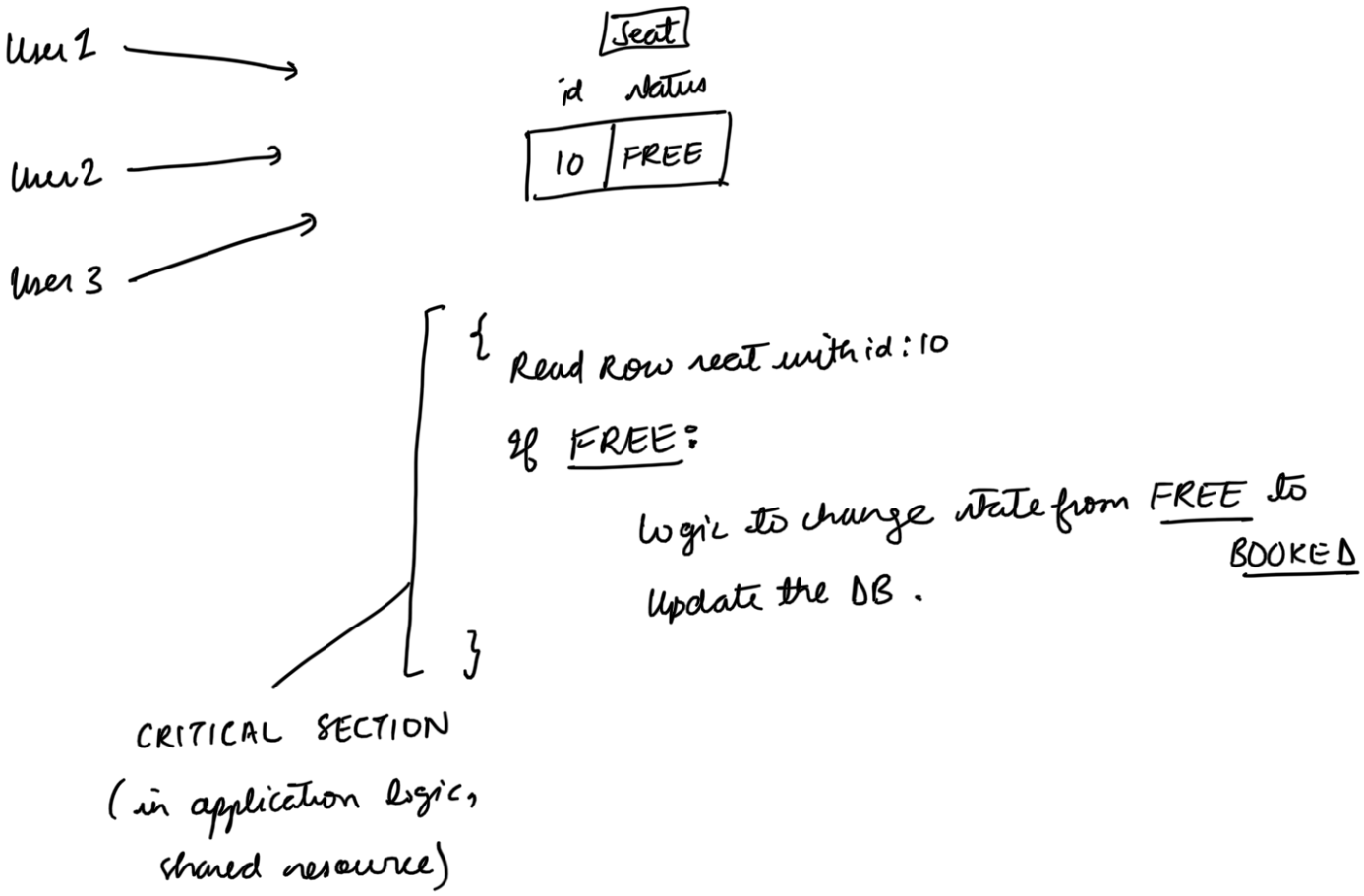
Concurrency control in Distributed systems

⊛ PROBLEM STATEMENT



Solution:

① Using 'SYNCHRONISED' for critical section.

U1 ✓
U2 ✗
U3 ✗

} won't work for distributed concurrency.

1 Process → n threads ✓ synchronised work

n Process ✗ synchronised won't work

(n machines)

② Using DISTRIBUTED CONCURRENCY CONTROL

OPTIMISTIC (OCC)

PESSIMISTIC (PCC)

Recap :-

① Usage of TRANSACTION ?

② DB LOCKING ?

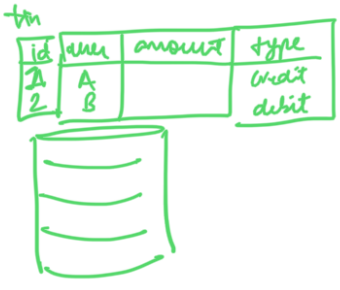
③ ISOLATION LEVEL present ?

we apply at application code level

we apply at DB level.

→ helps achieve INTEGRITY.

i.e. helps avoid INCONSISTENCY in DB.




```
tx.db.begin()
  - debit money from A
  - credit money to B
  if all writes ß
        COMMIT();
  else
```

tx.db.end()

TODO: know more about these work internally.

→ if txn fails, it will rollback all neo4j statements run as part of this txn.

DB LOCKING

LOCK
SHARED: (S)
EXCLUSIVE: (X)

→ makes sure that no other txn updates the locked resource.

LOCK TYPE	ANOTHER SHARED LOCK?	ANOTHER X LOCK?
HAVE SHARED LOCK	✓	X
HAVE (X) LOCK	X	X

(S):-
- is used for reads.
- in a shared lock, only read can happen
- restricts any other txn. to update record on which S is acquired

T_1 (S) — read — [13 | Free | .. | .....]

T_2 — read — ✓

(X):-
- is used by a txn. to WRITE to a record (update)

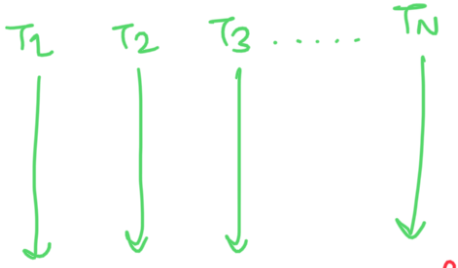
- other txns. cannot read/write to a record on which (X) is taken.

→ If a row has shared lock, first remove shared lock, then grant exclusive lock.

→ On exclusive lock, no other txn is allowed to take shared / exclusive lock.

ISOLATION LEVEL

- how much level of concurrency is allowed in your DATABASE.
- ACID
  ↓
  ISOLATION



these txn's feel like they are working alone.

ISOLATION LEVEL (solves problems →)	PROBLEMS: DIRTY READ	NON-REPEATABLE READ	PHANTOM READ
1) READ UNCOMMITTED	Yes	Yes	Yes
3) READ COMMITTED	NO	Yes	Yes
[illegible] READ	No	No	Yes

3) REPEATABLE READ	No	No	No
4) SERIALIZABLE			

① DIRTY READ PROBLEM :-

- if txn A is reading data written by txn B but not yet committed.
- if txn B does rollback(), then whatever data read by txn A is dirty read.

② NON-REPEATABLE READ :-

- if txn A reads same data / record multiple times, in a txn & same data does not get.
  ↳ other txn B has succesfully committed for that recorded. (commit())

③ PHANTOM READ :-

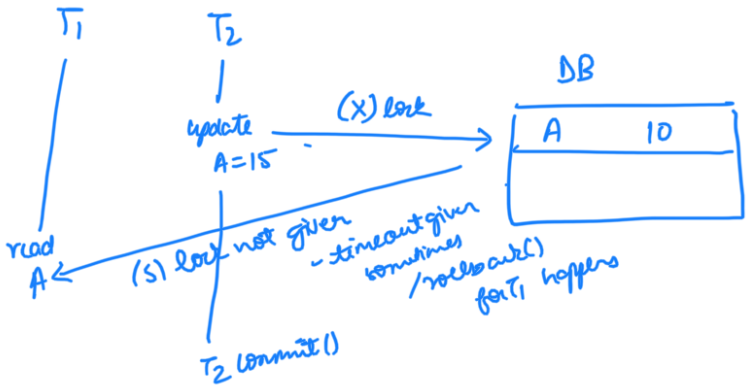
- txn A executes same query several times, & rows returned are different.

value ≥ 10 & value ≤ 15

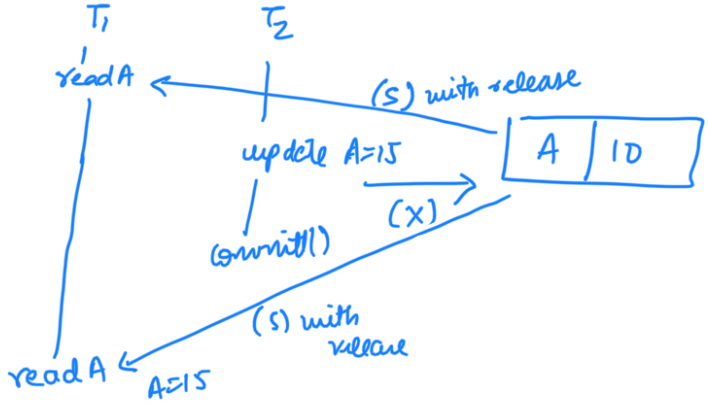
in this range ← txn B commits new rows to the database.

First read this, then go to problems table

ISOLATION LEVEL	CONCURRENCY ↑ High	LOCKING STRATEGY (S/X)
① READ UNCOMMITTED		Read: No lock Write: No lock (Very risky) (When you require only reading)
② READ COMMITTED		Read: Acquire S, Release as read is done. Write: Acquire X, & keep till end of txn. i.e. early commit()



∴ Dirty Read not possible



∴ Non-Repeatable Read possible

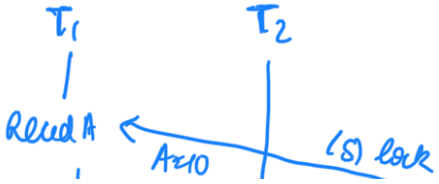
Phantom Read possible

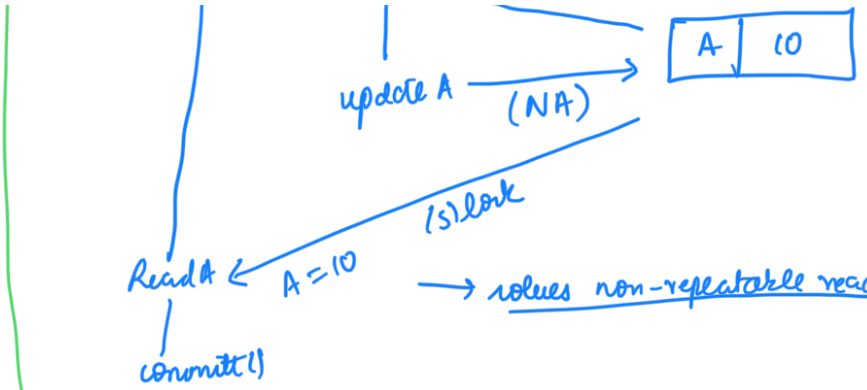
LESS CONCURRENT THAN READ UNCOMMITTED

③ REPEATABLE READ

Read:- (S) lock acquired, release only at end of txn.

Write:- (X) lock acquired, release only at end of txn.





→ allows non-repeatable read.

∴ Phantom Read Possible

④ SERIALIZABLE

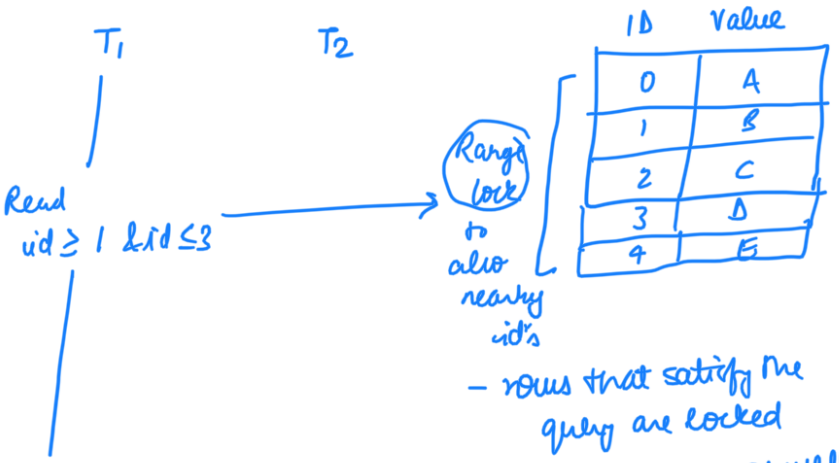
REPEATABLE READ STRATEGY

+

RANGE LOCK

(lock is released at the end of txn)

CONCURRENCY LOW



- rows that satisfy the query are locked
- & nearby rows as well.

∴ tries to avoid phantom reads.

we set it for every transaction else, default is selected for DB specific.

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

BEGIN TRANSACTION

select
```

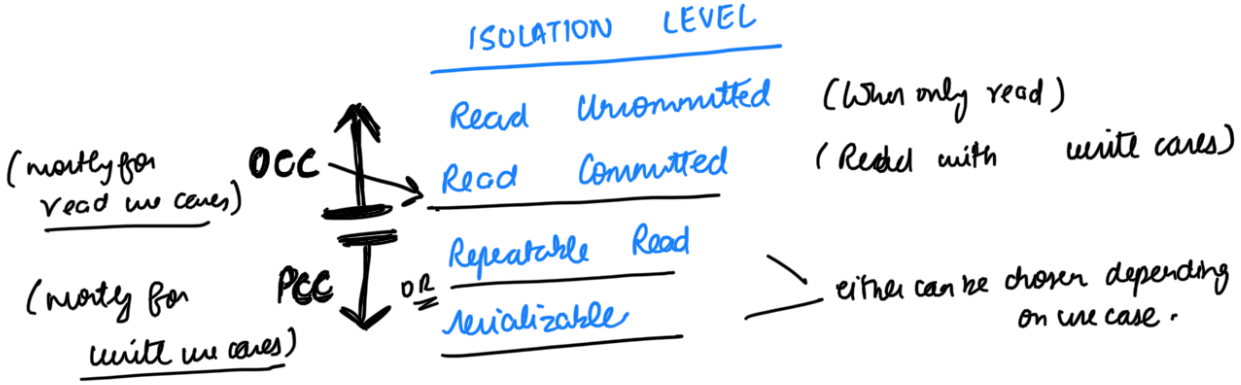
update

if success: COMMIT TRANSACTION

else: ROLLBACK TRANSACTION

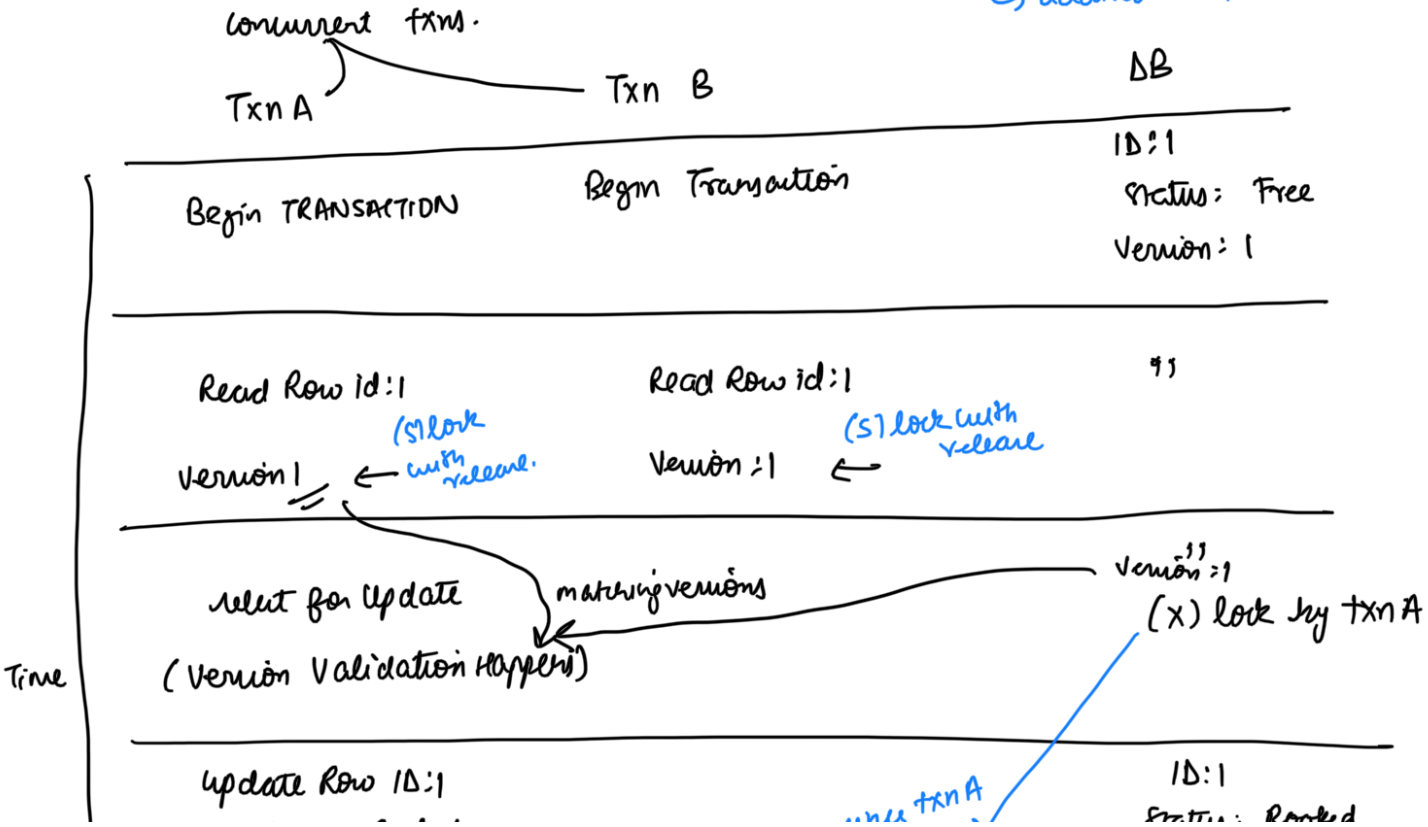
END TRANSACTION

OCC & PCC



OCC (Read Committed ISOLATION LEVEL)

- solves concurrency using version.
  - ↳ additional special row in DB



Status : Booked
Version : 2

has acquired (X) lock, txn B cannot do read / write till commit/abort

Version : 2
(X) lock by txn A

Commit
↳ Releases exclusive lock

ID : 1
Status : Booked
Version : 2

Select for Update (Version Validation)

version : 1

Version : 2

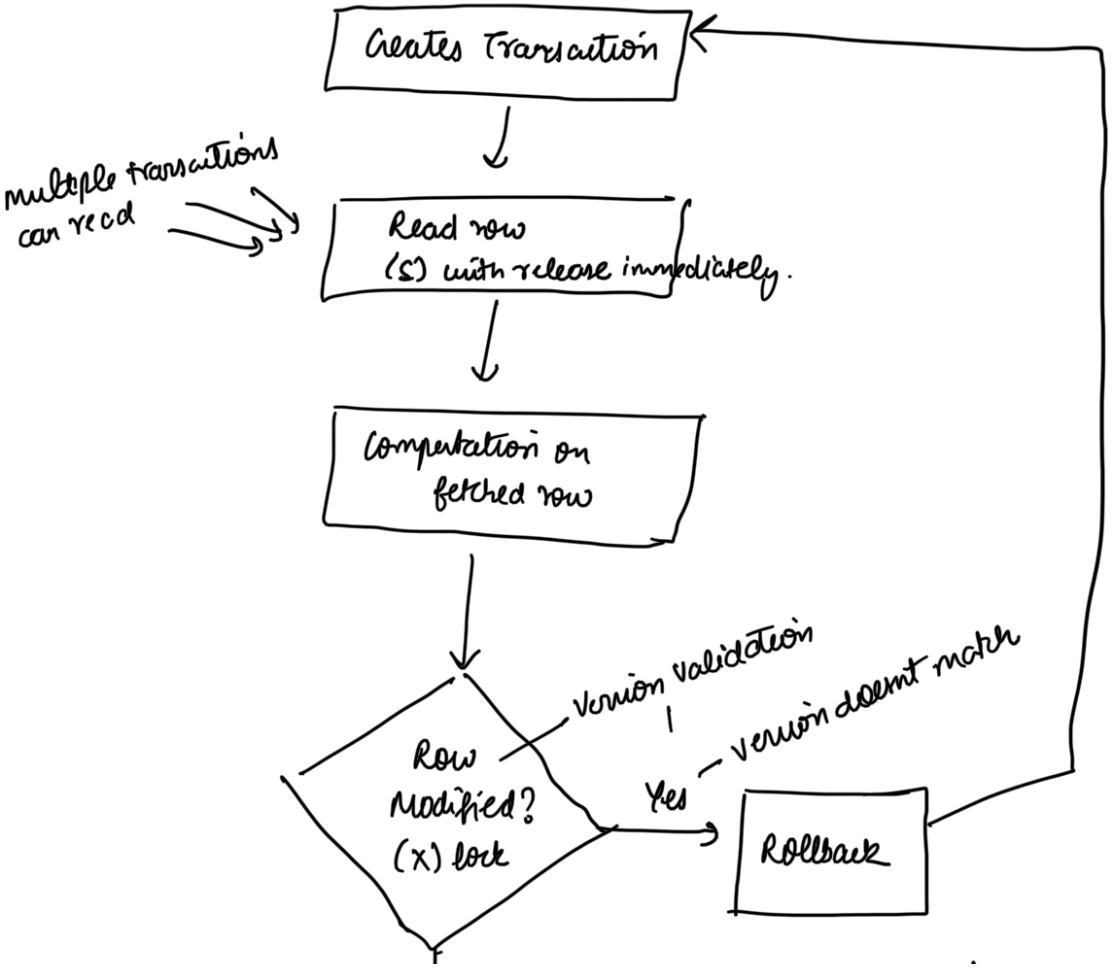
(X) lock applied.

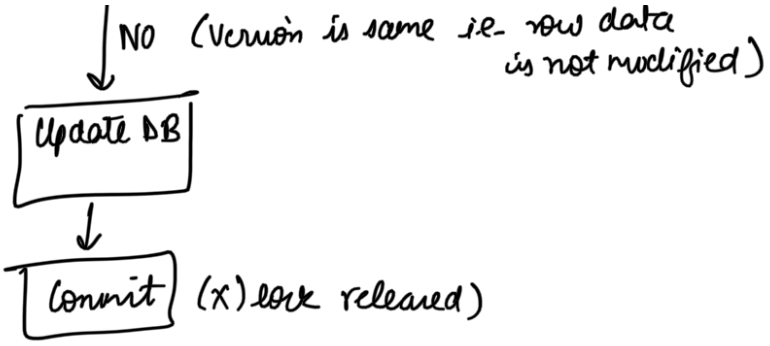
version mismatch

Rollback ()

(X) lock released

Flow for a Transaction :-





Mostly used in companies :-

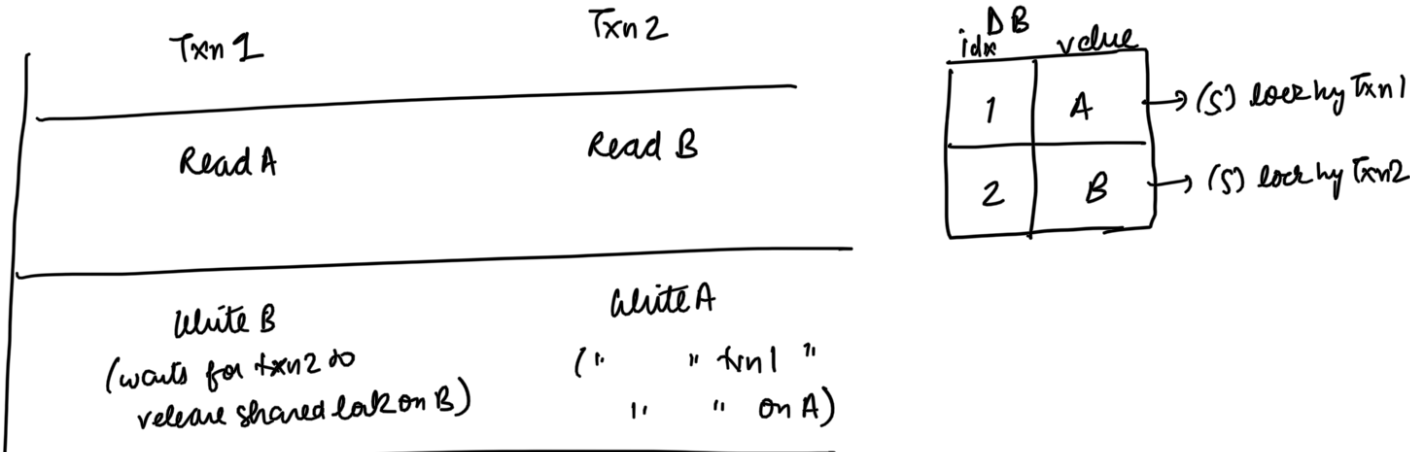
- very less sudden concurrency comes, OCC will handle.

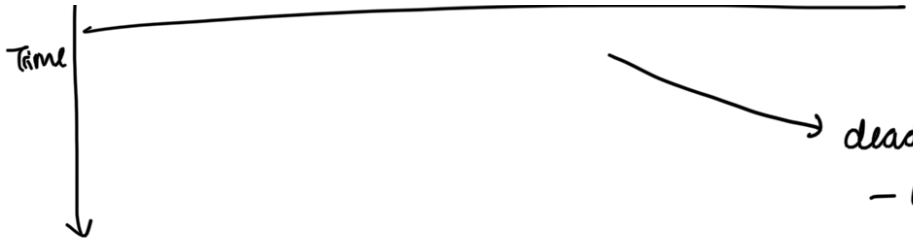
PCC

ISOLATION LEVEL

- REPEATABLE READ
- SERIALIZABLE

- brings serialization.
  (sequence because of locking)
- other txns would have to wait to update before one txn is done committing the update.
- this creates a problem of deadlocks in PCC





∴ OCC solves deadlock over PCC

OCC

* ISOLATION LEVEL

  Read :- Read Uncommitted

  ✓ Write :- Read Committed

* High Concurrency

* No chance of deadlock

* In case of conflict,

  ✓ overhead of transaction rollback (check versions)

  & retry logic is there

PCC

* Repeatable Read

  ✓ &

  Serializable

* Low concurrency compared to optimistic

* ✓ Deadlock is possible, txns stuck in deadlock are forced to rollback.

* ✓ Putting a long lock (isolation level duration) in case of long running txn

  ↓

  timeout issue happens & rollback needs to be done.