DBMS - Transactions and Indexing

Agenda -

① Transactions

- First transactions
- Isolation levels

② Indexes

- disk access
- Types
- How to create

Isolation

↳ concurrent transactions should be independent

↳ independent

↳ slow

↳ Isolation levels allow us to fine tune isolation / how concurrency isimplemented.

→ Read uncomitted

→ Read comitted

→ Repeatable reads

increasing order of

→ $\boxed{\text{Serializable}}$

↓ strictness

$\boxed{X = 2000}$

T1	T2
T = READ(X) ⇒ 2000	
	T = READ(X) ⇒ 2000
T = T + 500	
WRITE(X, T)	

↓

2500

T = READ(X)

⇒ 2500

COMMIT

T = READ(X)

⇒ $\boxed{2500}$

① Read uncomitted

↓

comitted

T1 x = 2000	T2
T = READ(x)	
T = T + 500	
WRITE(x, T)	
⋮	← T = READ(x)
	⇒ $\boxed{2000}$
COMMIT	

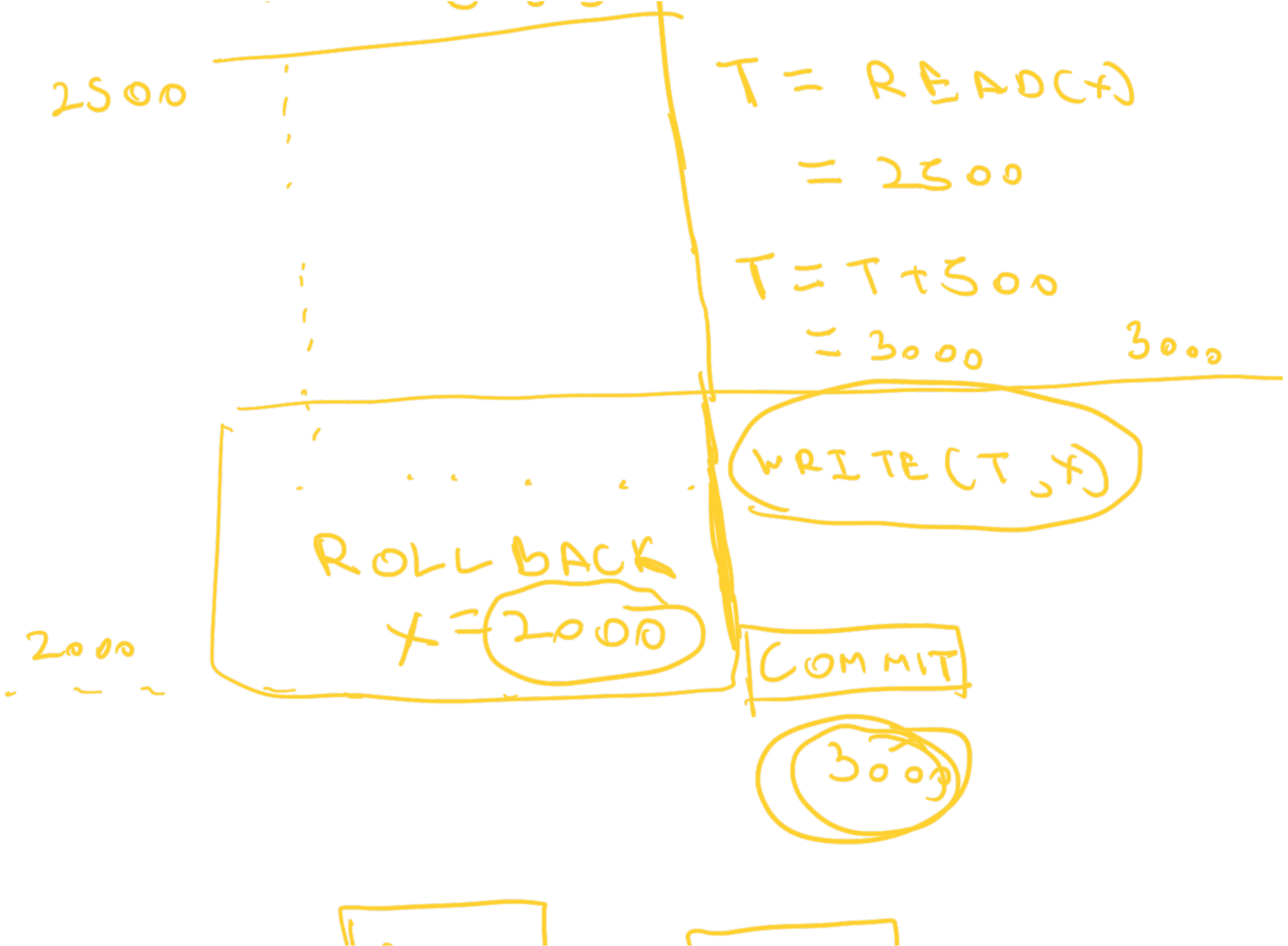
Read commited values on not

Read   Read

Committed	Uncommitted
→ Slower	→ Faster
→ Consistent	→ Dirty reads

Uncommitted

T1	T2
2000	
T = READ(X)	
T = T+500	
WRITE(X, T)	


2500
T = READ(X)
= 2500
T = T+500
= 3000
3000
WRITE(T,X)
ROLLBACK
X = 2000
2000
COMMIT
3000

|2000| vs |3000| $

un comitted

Dirty read

→ read uncomitted

Read comitted

– no dirty reads

Read uncommited

→ Eventual consistency

→ Sending emails

CAP

→ availability

→ partition tolerance

→ consistency

analytics

repeatable reads

I want to send emails | coupons to all my students

T1 | T2

→ Get all students (100)

→ Generate 100 coupons

Create a new student (101)

COMMIT;

→ Read all students (101)

Error

non-repetable recods

→ Every time I read from DB
I get a new value

① Repeatable recods

② Serializable

Dirty read

- uncommitted value

- T1 T2

- T1 → x+50

- T2 → x+50

= T1 → Rollback

Read comitted

Lost update

T1 2000	T2
T = READ(x)	
T = T+500	
	T = READ(x)
	→ 2000

WRITE(Y)

COMMIT

T = T + 500

= 2500

WRITE(2500)

2300 - 3000

Lost update

Non-repetable read

→ Emails

→ Read () — $\boxed{100}$ T1

→ Coupons

→ Creates a new student T2

— 101

— COMMIT

→ Read () — $\boxed{101}$

→ Inconsistent state

Lost update — repetable read

Serializable — strictest

↳ sequential transactions

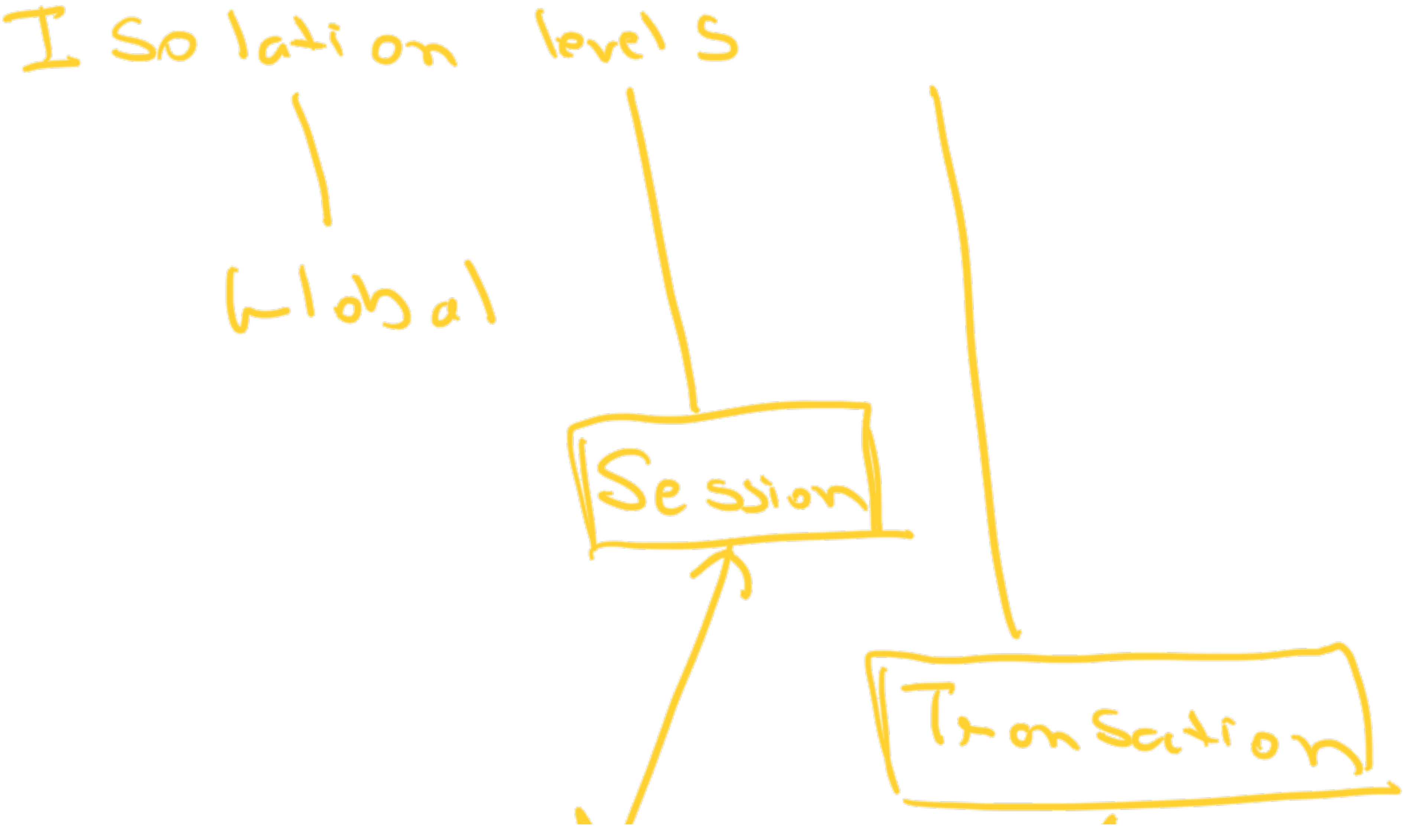
↳ Slow — not as efficient

↳ Highly consistent

↳ Banking

5:58 - 6:02

10:32



Developer

Application

Indexes — glossary

CRS Index

DP - 1, 20, 100

Graph - 10, 20, 100

→ Easy way to search for values

Index → reduce disk

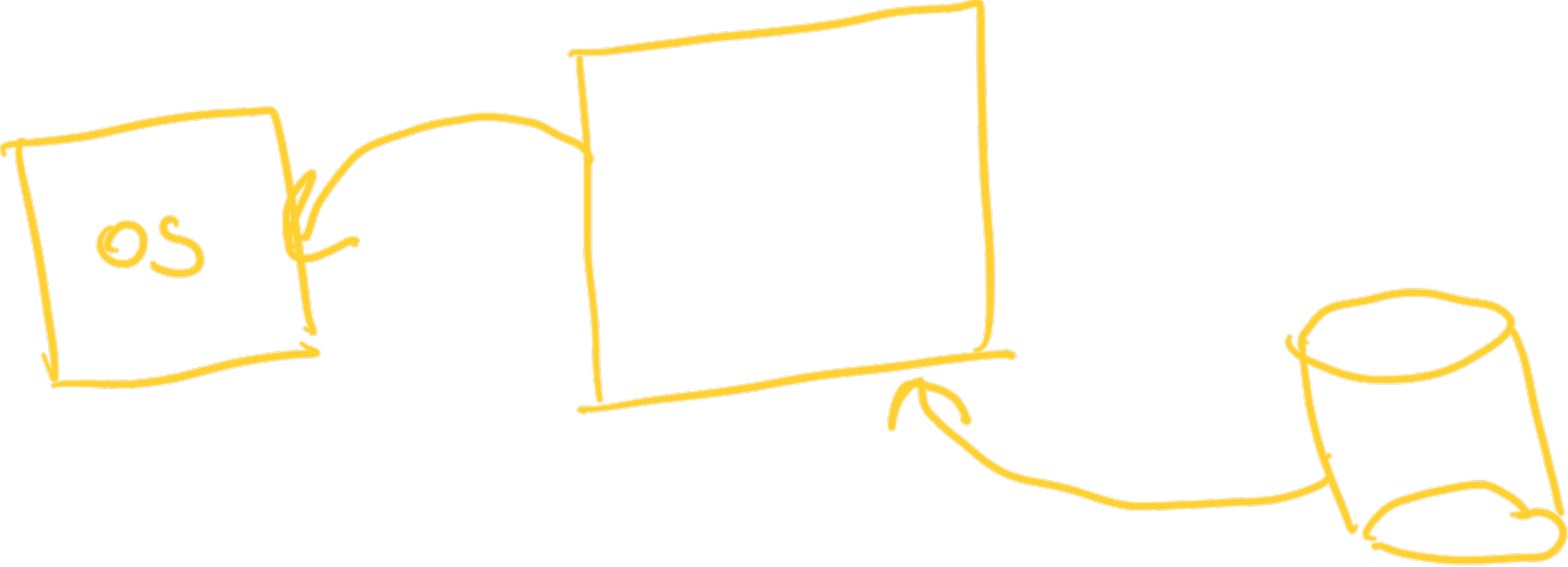
[illegible] reduce disk access

Database - persistent
- disk

→ Disk is very slow

→ CPU is fast

Memory



Students

→ ⇒

id	name	phone	email	batchid
1	A	1	A	2

✗

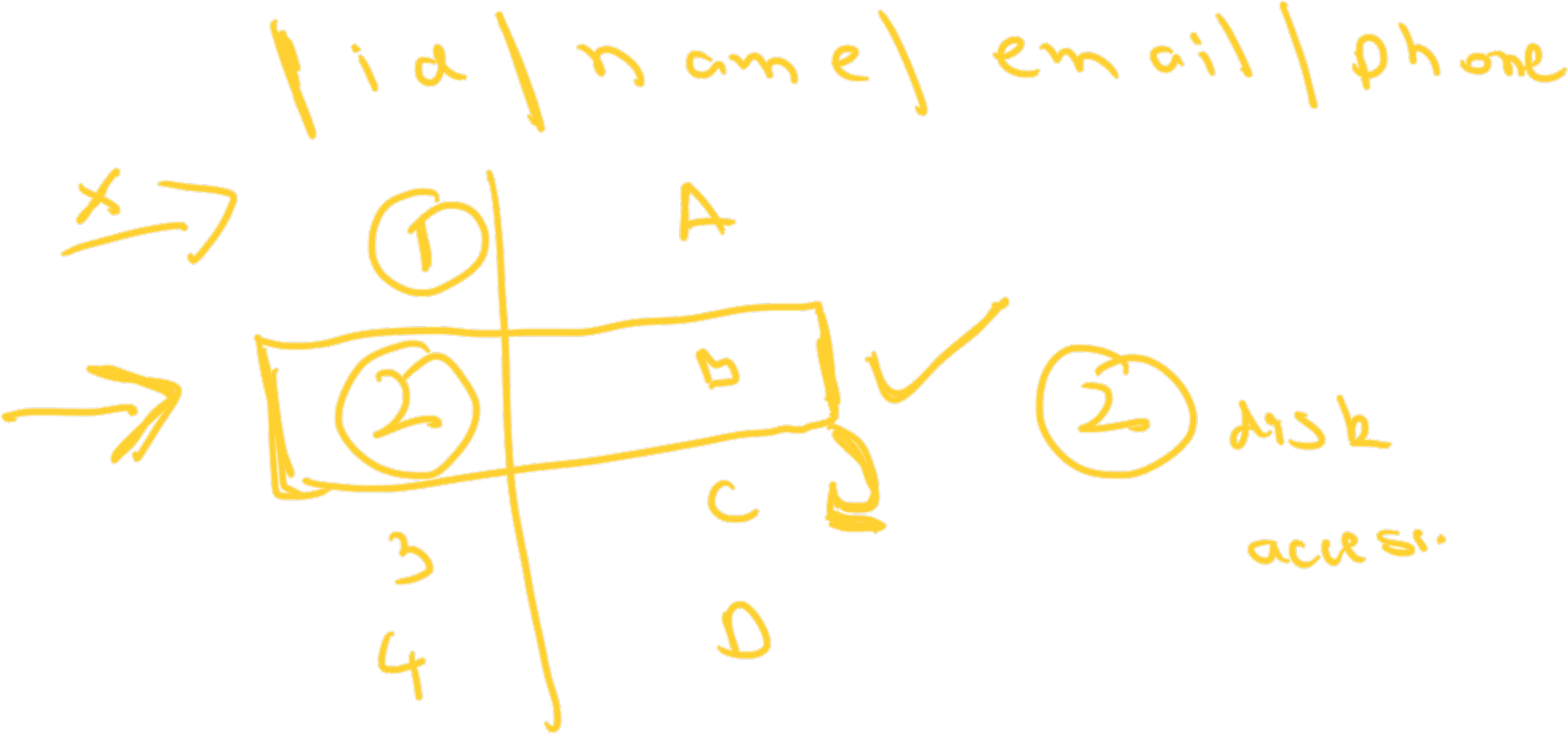
4

→	2	b	2	b	3	X
→	3	c	3	c	1	X
→	4	D	4	D	4	X

```
Select * from students
         WHERE   batch_id = 4
```

10 M — 10 M

Students



id	name	email	phone
1	A		
2	B		
3	C		
4	D		

2 disk

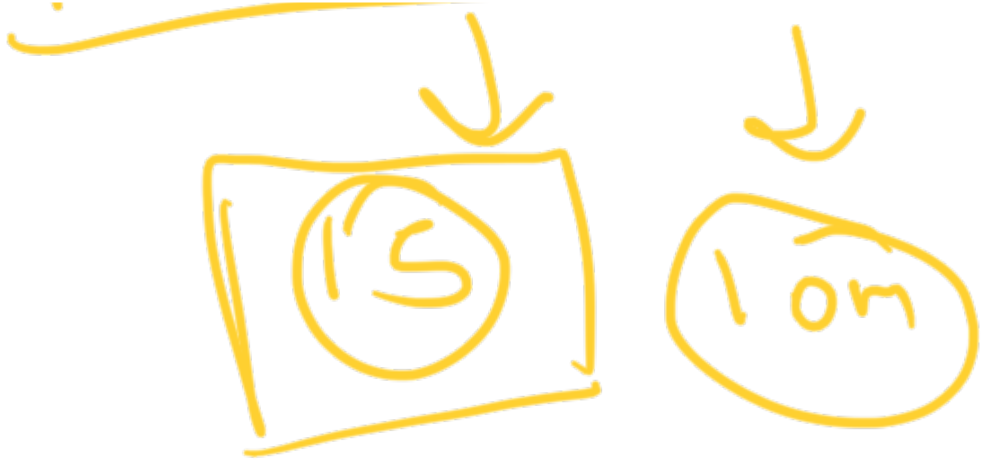
acces.

from

Select students

where id = 2;

10M → Select id = 15



Student

	id	name	phone
A01	1	A	+91
A02	2	B	+44
A03	3	C	+90

unsorted

→ AD4 4 D +1 |

Select * where ph = +1

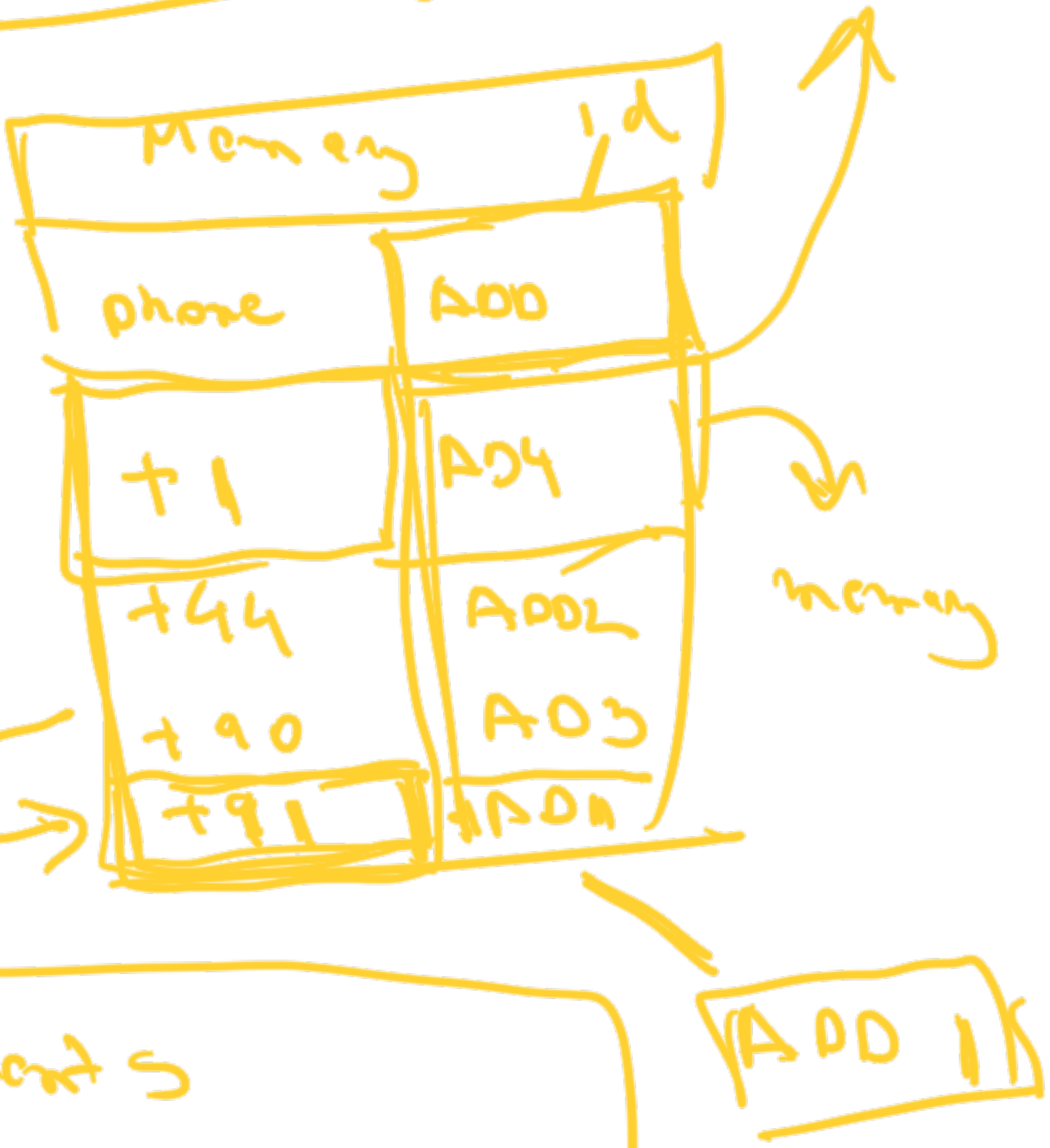
index table

→ Auxilary space

→ Sort ?

binary search

hash map



Memory id

phone	ADD
+1	AD4
+44	ADD2
+90	AD3
+91	ADD1

memory

ADD 1

Select * from students

where phone = +91

1 disk access

Reduce disk access

→ unsorted, non-unique

O(N) — N

→ Sorting

→ ID 10 → out of 10M

→ $\boxed{10}$

→ Sorting + separate table

→ Disk access → ①

Indexes

→ way to map values to address.

val → ADD.

→ Sort your values

+ address

→ When does this map get updates?

→ Upade| insert → index

→ Insert queries slower

When to use indexes?

→ Find rows onthe basis of a where clause. frequently

PK – indexes

phone, email — indexes

- Access patterns
- If pattern is obvious
    - email, phone — index
- APM — application monitoring tool
    - slow queries
        - WHERE

— index

Composite indexes

when not to create indexes

→ small tables — 10,000 1k
— 1mb

→ If where clause
returns a lot of rows

no index

→ NULL values

→ Colums that are change often

→ UPDATE →

  - → Remove index
  - → update
  - → create index