1. Types of NoSQL DB
2. Challenges wrt creating a NoSQL DB
   - (i) Storage
   - (ii) Sharding
3. How NoSQL DB shard data
   - (i) M-S architecture
   - (ii) Multi Master architecture

Types of NoSQL DB

→ everything other than SQL

Based upon how they store data, NoSQL DB are of Multiple Types (and each type has Pros and Cons)

- (i) Key-Value DB
- (ii) Document DB
- (iii) Column Family
- (iv) Graph DB
- (v) TimeSeries DB
- (vi) Hierarchical DB

KEY VALUE DB

(Map)

→ Store data very much like a Map.
↳ k-v pairs

→ Store data only By Key

→ Fetch data only by Key

dbs. put (k, v)

dbs. get (k)

persistence

Eg: 1.) Dynamo DB
2.) Redis

no persistence

1.) k-v DBs often don't have any constraint reg the value
↳ value is nothing but a byte[]
↳ they can't have

reads basic attrs of value.

[0110101110100]

JSON / Integer / String

for the appⁿ to decide.

db.put(1, "hello")

1 → [1011101110001001001]

db.get(1) ⇒ Byte[]

(int) a = (int) db.get(1)

a = (String) db.get(1)

Animal c = (Animal) db.get(1)

create table something (
key int,
value byte[] .
)

↓

0110110010010000

↓ Developer of DB

```
void put (String key byte[] value){

}
```

→ when only op^n we need are based on key

↳ (k)-v DB are most performant

dynamo.put (1, "Hello")

dh.put (1, "Hello")

Hashable

int, String

Byte[]

```
void put (Object key, Object value) {

}
```

(2) DOCUMENT DB

⇒ also K-V DB

⇒ But one diff

↳ value is a JSON doc which can have any set of attrs.

↳ also allow you to query on those attributes.

db.put (1) { id: ____ ,
title: ____ ,
price: ____
rom ____
}

db.put (2) { id: ____
fabric: ____
title: ____
}

RDB. get ({ fabric: "cotton" })

.get ({ fabric: "cotton" })

→ ACID

⇒ eg : 1.) MongoDB
2.) Elastic Search.

When to use:

⇒ entities with diff set of attr
⇒ and we want to fetch basis those attributes.

eg : E-COM Websites

Products

1 { id: ___ , name: ___ , colour: ___ }

2 { id: ___ , colour: ___ , fabric: ___ , brand: }

③ COLUMN FAMILY

SQL

How is data stored

@431

id	name	81	93	2	Amit	31
14	3					

Row 1 Row 2 Row 3 Row 4

Multiple Disk Access

SQL

Students

id	name	marks	psp.

Q^n give me marks of latter 3 Students

RAM

array[3][5] = { 1, 2, 3, 4, 5
6, 7, 8, 9, 10
11, 12, 13, 14, 15 }

Code 1 (Faster)

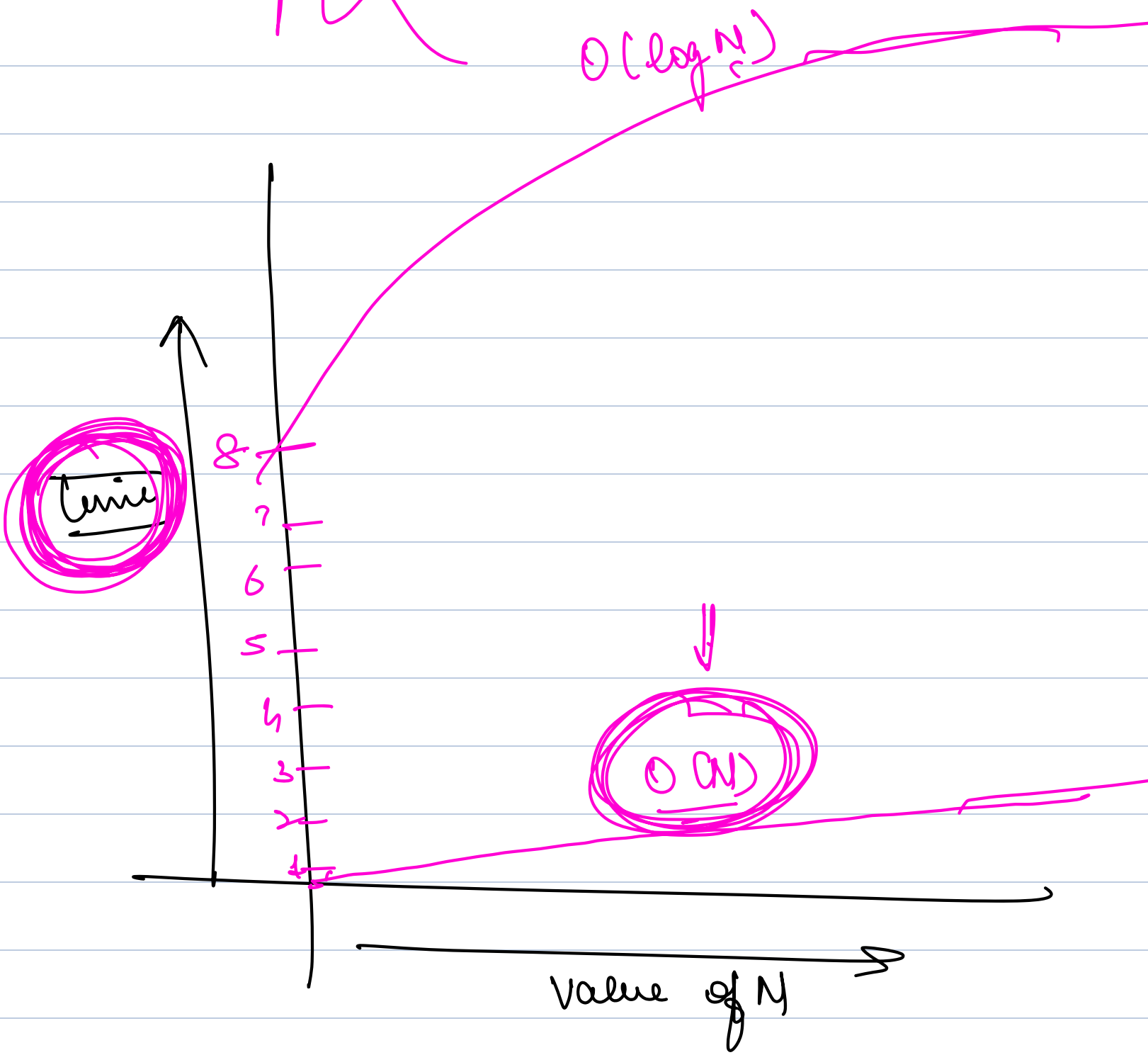
```
for (i = 0 ; i < # rows ; ++i) {
    for (j=0; j < # cols ; ++j) {
        print (arr [i][j])
    }
}
```

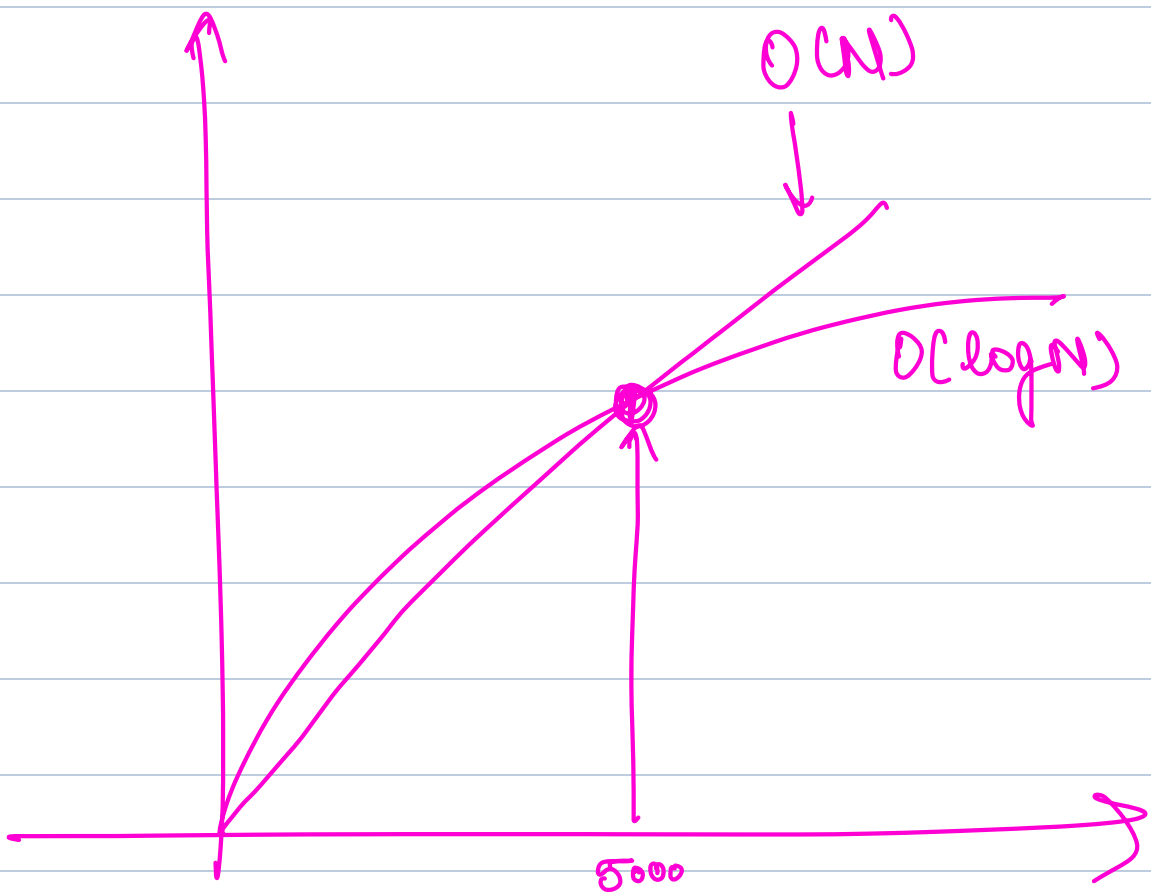
Code 2

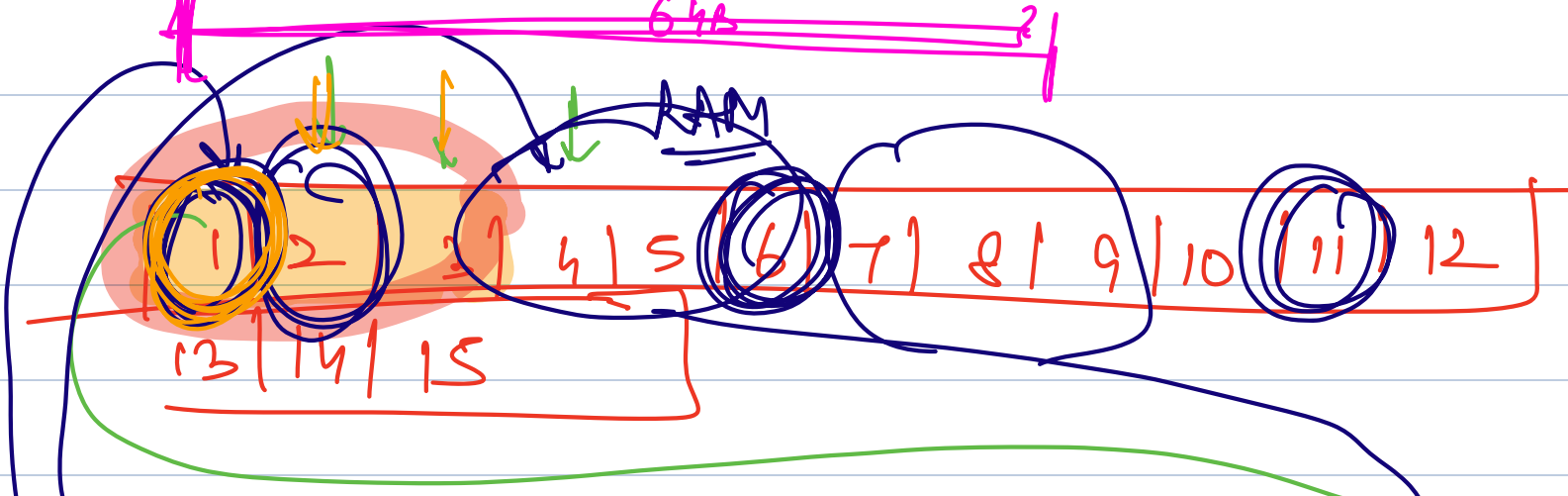
```
for (i = 0; i < # cols ; ++i) {
    for (j=0; j < # rows ; ++j) {
        print (arr [i][j])
    }
}
```

~~Same~~ =

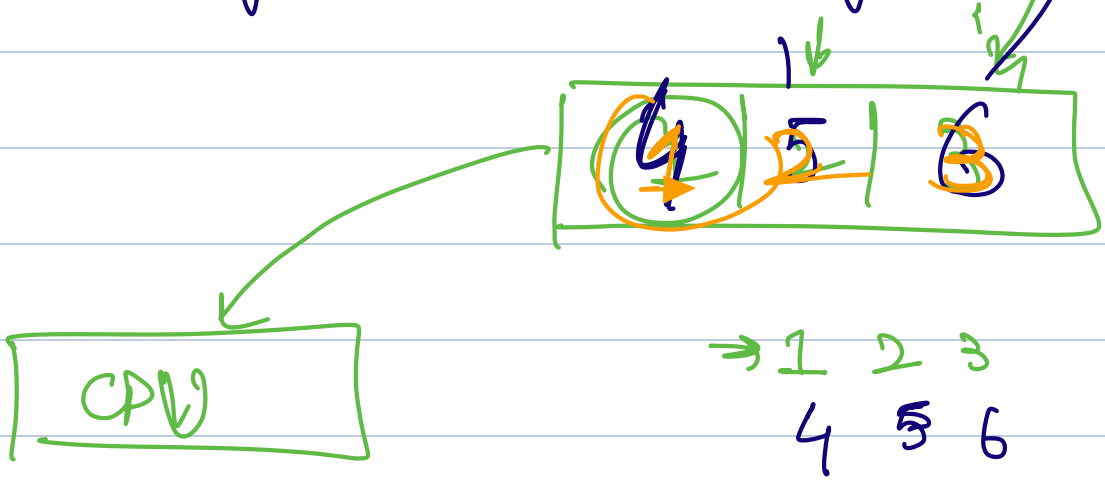
~~depend~~

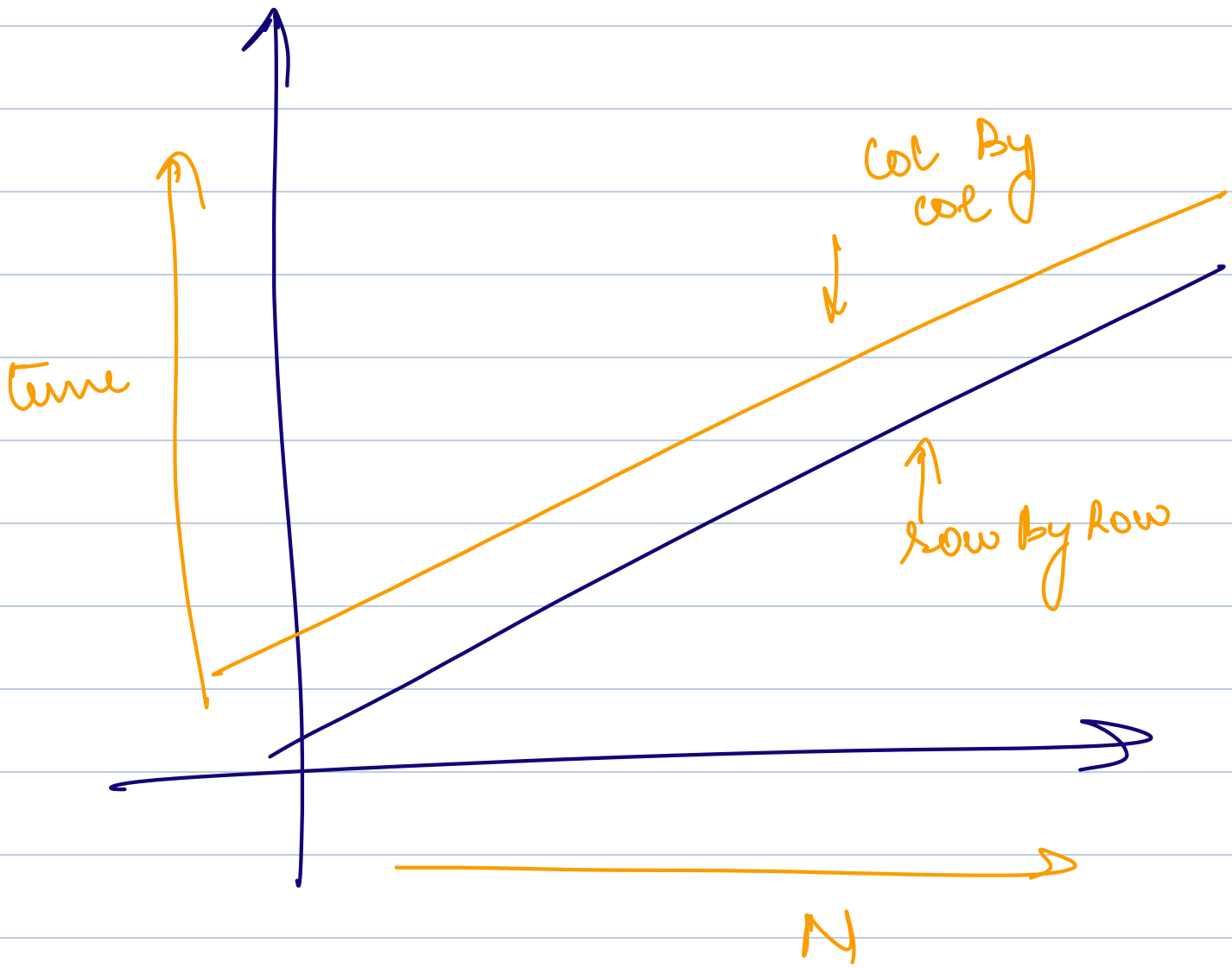



O(N)
O(logN)
5000



access : comp. gets that add and nearby add to a register.

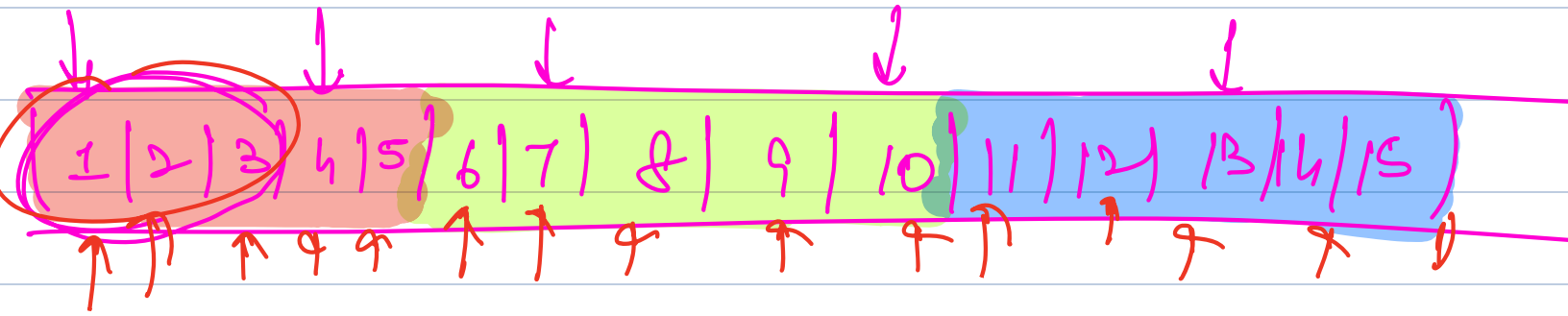




"Locality of Reference"

Computer when you ask them to access a particular add from RAM they bring the data in nearby add as well to register just in case you

might need it



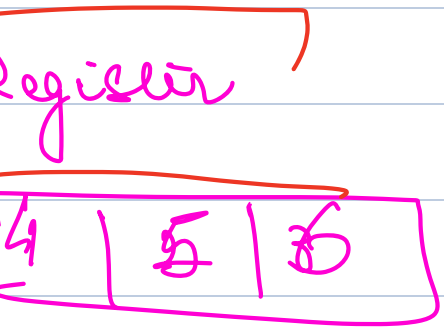
Size of Register = 12B

Row By Row

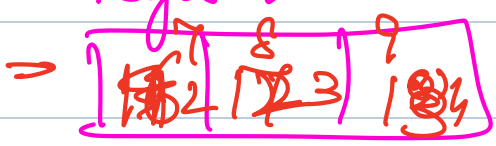
5 Mem Accesses

Col By Col

Register



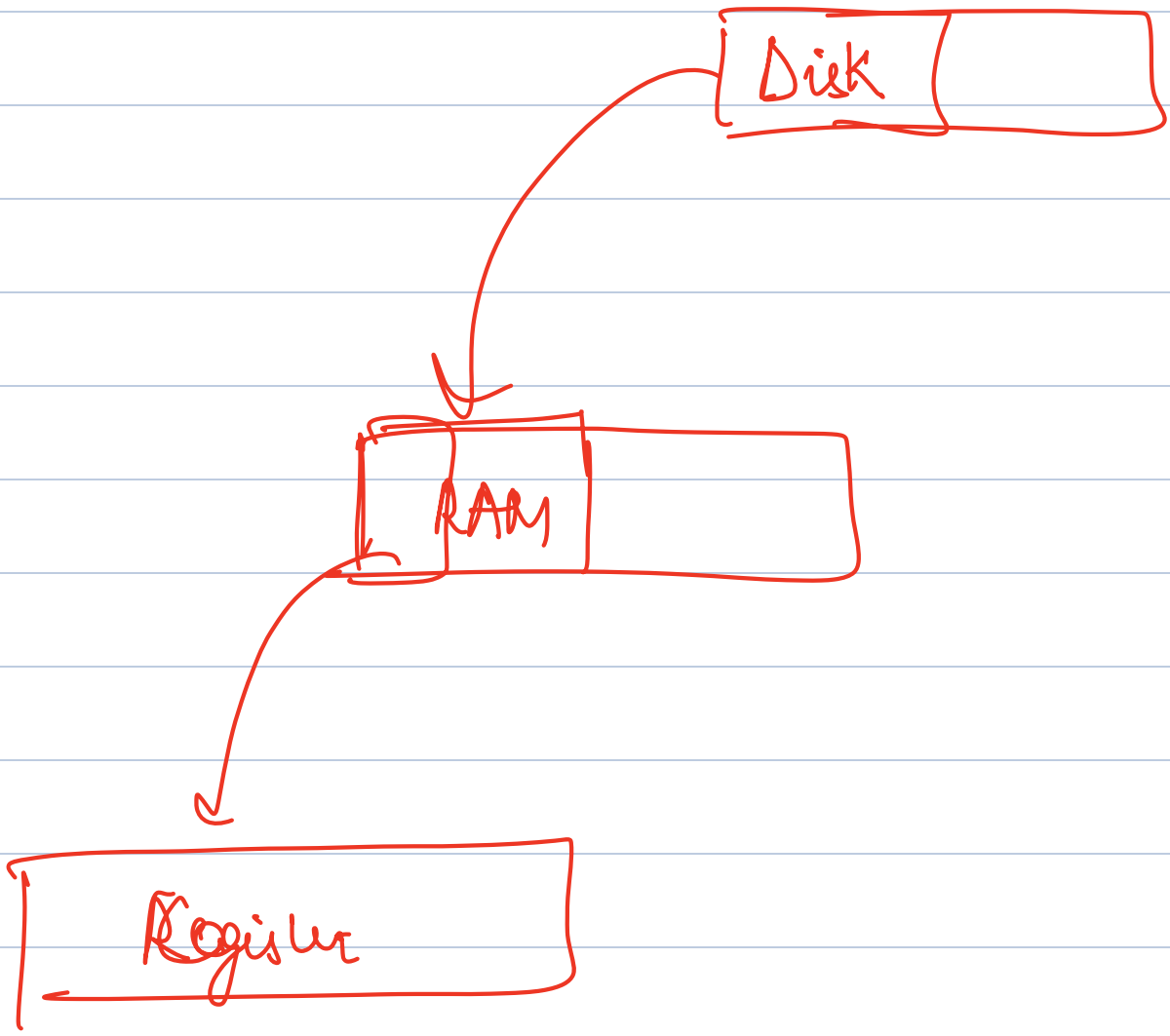
Register



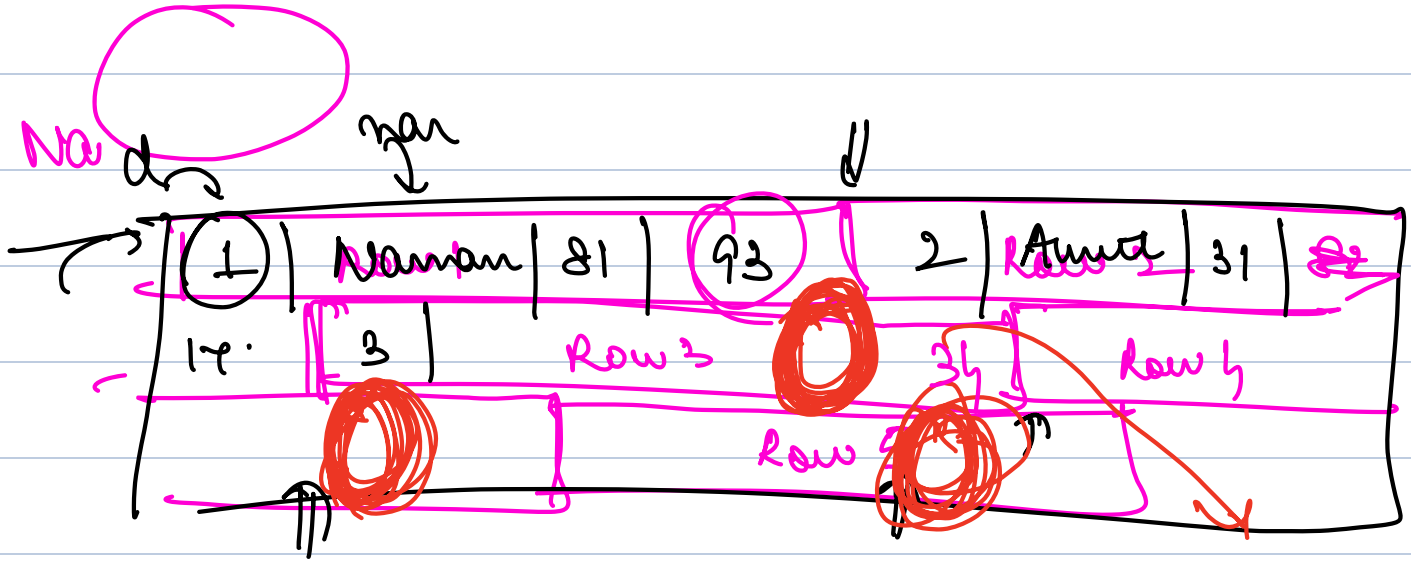
12 13 14

10 years ago

4999 → 14 hrs screen on



What if data was stored like this:



1	2	3	4	5

Naman	Amit	vicky	—	—

81	31	97	—

93	87	34	?

get Marks of latest 3 Students

⇒ Store data col By Col → RAM

⇒ only allow to append values

tweets

civis Pak

⇒ twitter uses a col family DB.

⇒ They have a col family of every hashtag.

Each col^n has a family

Trend Vs Pole → [_ , _ , _ , _]

Row of a Table

Table

Ban EVM [_ , _ , _]

#INDIA [_ , _ , _ , _ , 0]

whenever a tweet is created,

its value is appended to ~~every~~ column of every hashtag it contains.

Today is a huge day #indVSpak #INDIA

db.get ('#indVSpak', 5)

Ind Vs Pak = [

{ user_name, ___, time1 ___, content ___ },

{ ___, ___, ___ },

{ ___, ___, ___ }

]

Think of a CF DB as a KV DB where K is the name of col^n family and V is a list. Where you can append anything

#IndVSPak	[0, 0, 0, 0, 0, 0, 0, 0, 0]
#NVDA	[ ]

db1 = new Cassandra( )

db2 = new MySQL( )

No 2. execute( )

Q^n → live score of a cricket match

123 ⇒ current-score

K-V DB

avg

102, 106, 108, 108, 109, 110,

Col Family

Q^n Current Loc^n of User Data

K-V DB

Q^n Products info in Ecom Web

→ Document DB

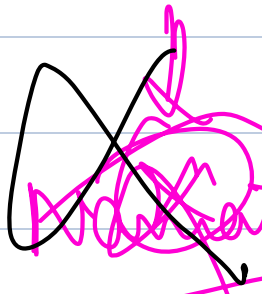
Eg Cassandra, HBase

↓

Col Family

Internals of NoSQL

- → Storage
- → Shard



→ Take Turn

→ Compete Everyone

NO Partition

P → Partition Tolerance

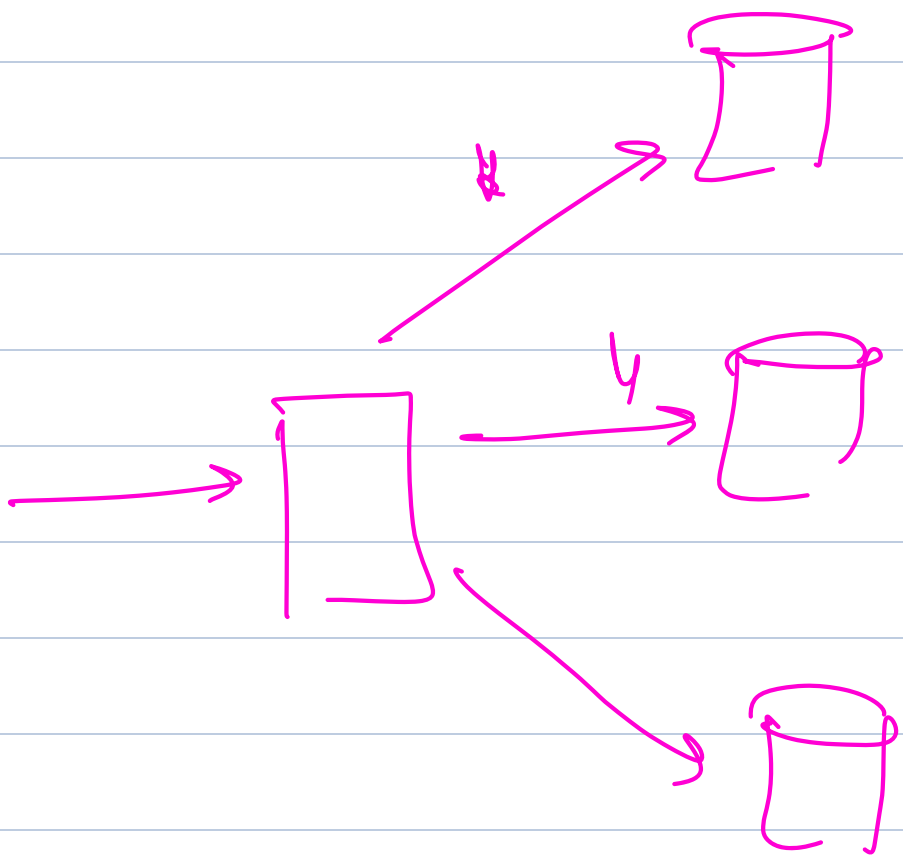
A or C

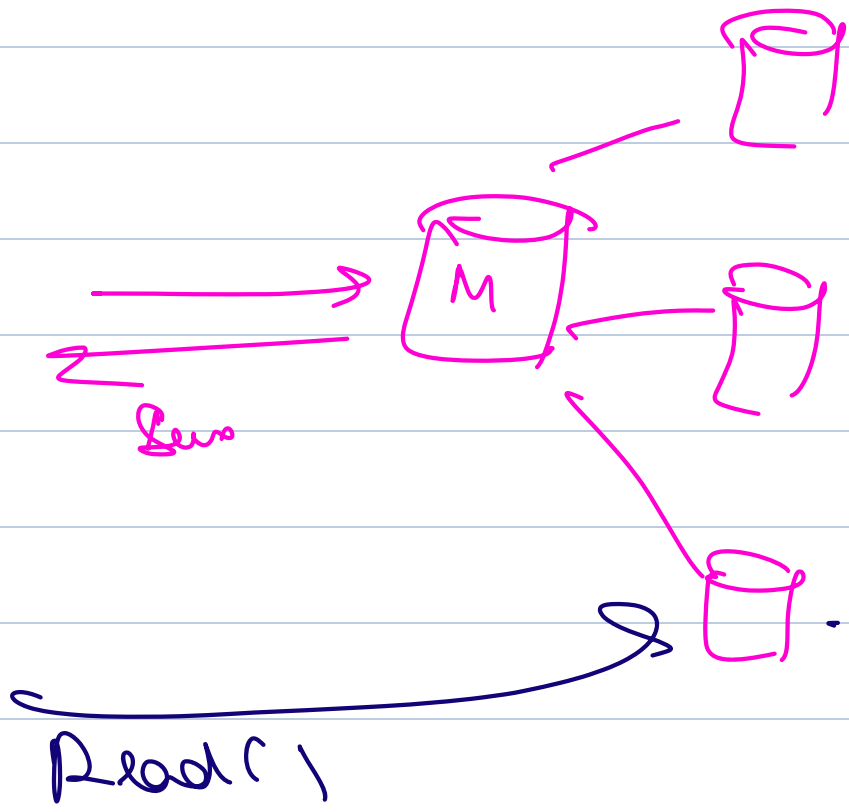
E L C

Consistent

Low latency

Availability is guaranteed




M
Send
Read( )

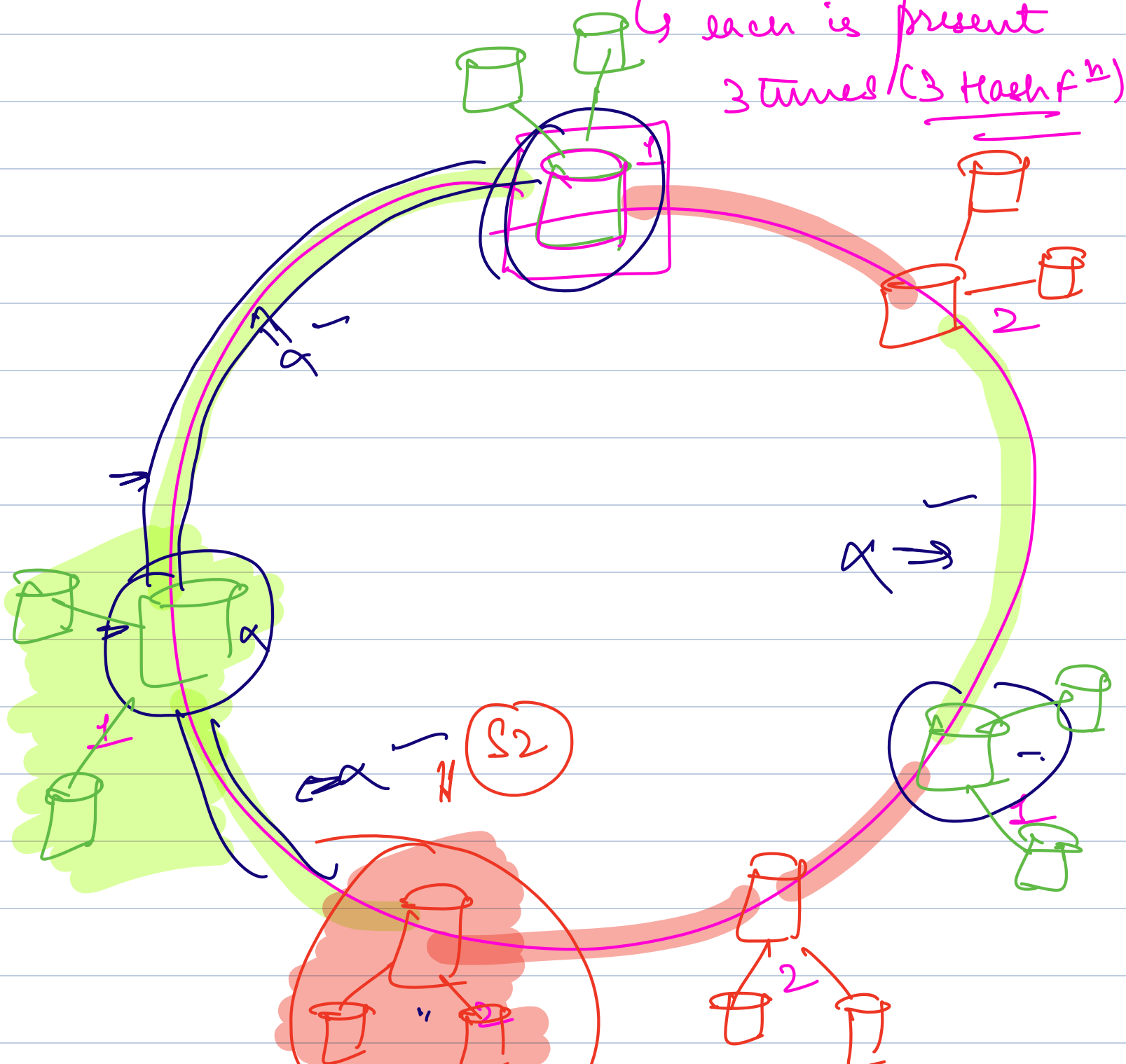
How MI Architecture and Sharding Works

Sharding

↳ logic: Consistent Hashing

↳ 2 Shards

↳ each is present 3 times (3 Hash f^n)



→ SPOF

6 Machines

CONS.
HASHING
+
M-S Architecture

[(W2), (B1)]

S_1 → master_ip =
slaves_ips = [ ]

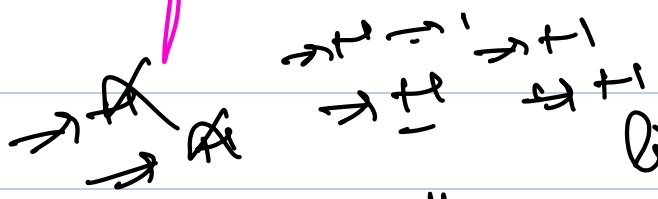
MACHINE GOES DOWN

- Master
- Slave

Slave goes down

Read
- ⇒ one less slave to read from
- ⇒ machine will become overloaded
- ⇒ in parallel we will ramp up a new machine and put it there.

Write



live - viewers

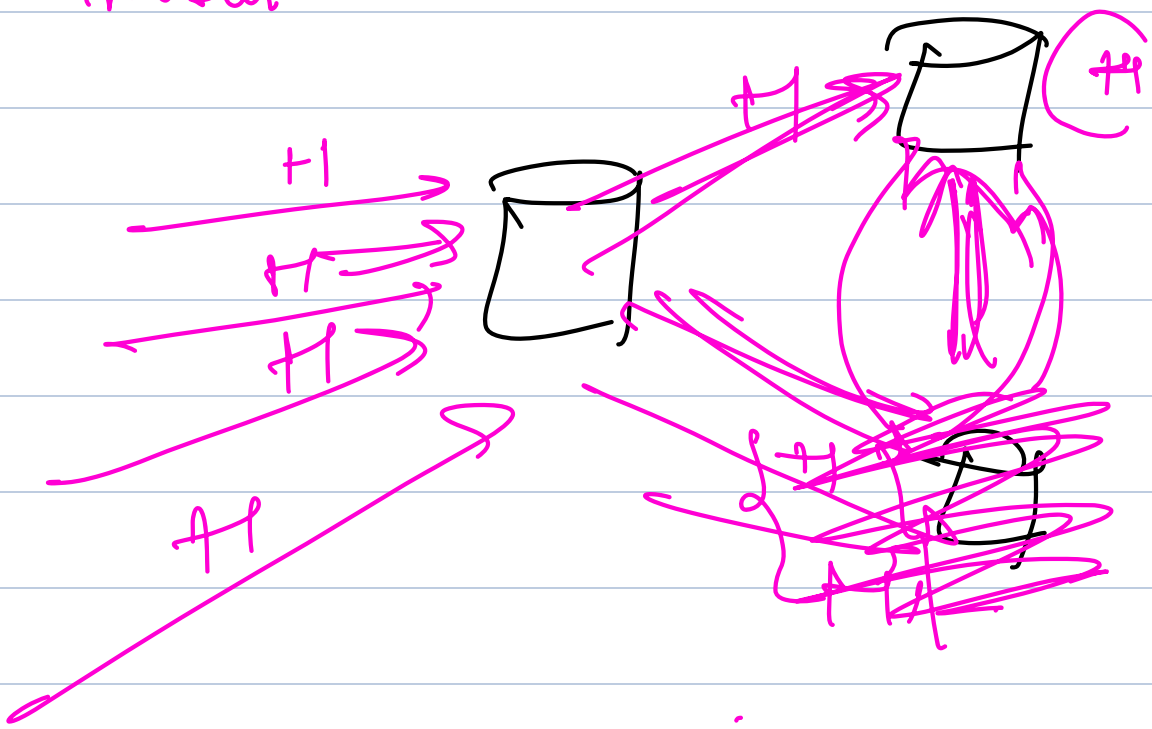
analytics

- Available and Never Consistent
  - ⇒ NO IMPACT
- Available and eventually consistent
  - ⇒ NO IMPACT
- Completely core system
  - ⇒ DOWNTIME FOR WRITES

TILL NEW MACHINE SETS UP

Master goes down

⇒ Leader Election Happens (one of the other machines is chosen master)



⇒ A new machine is also created to ensure our shard has count of machines as decided.

Read CS

→ Can still be served

WRITE(C)

→ COMPLETE DOWNTIME

↳ all writes go to master

↳ downtime till new master is chosen + slave added.

⇒ M-S Architecture

+

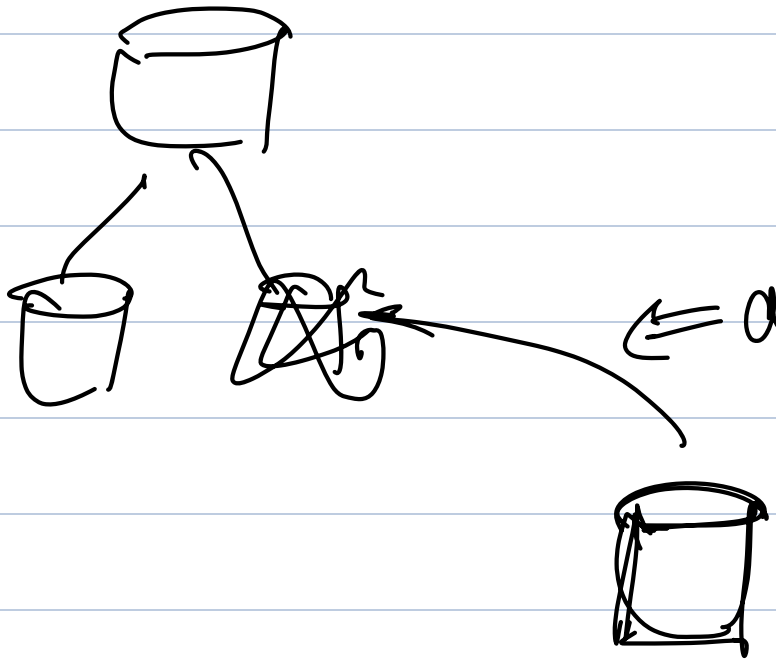
MACHINE DOWN ⇒ DOWNTIME

+

CONSISTENT

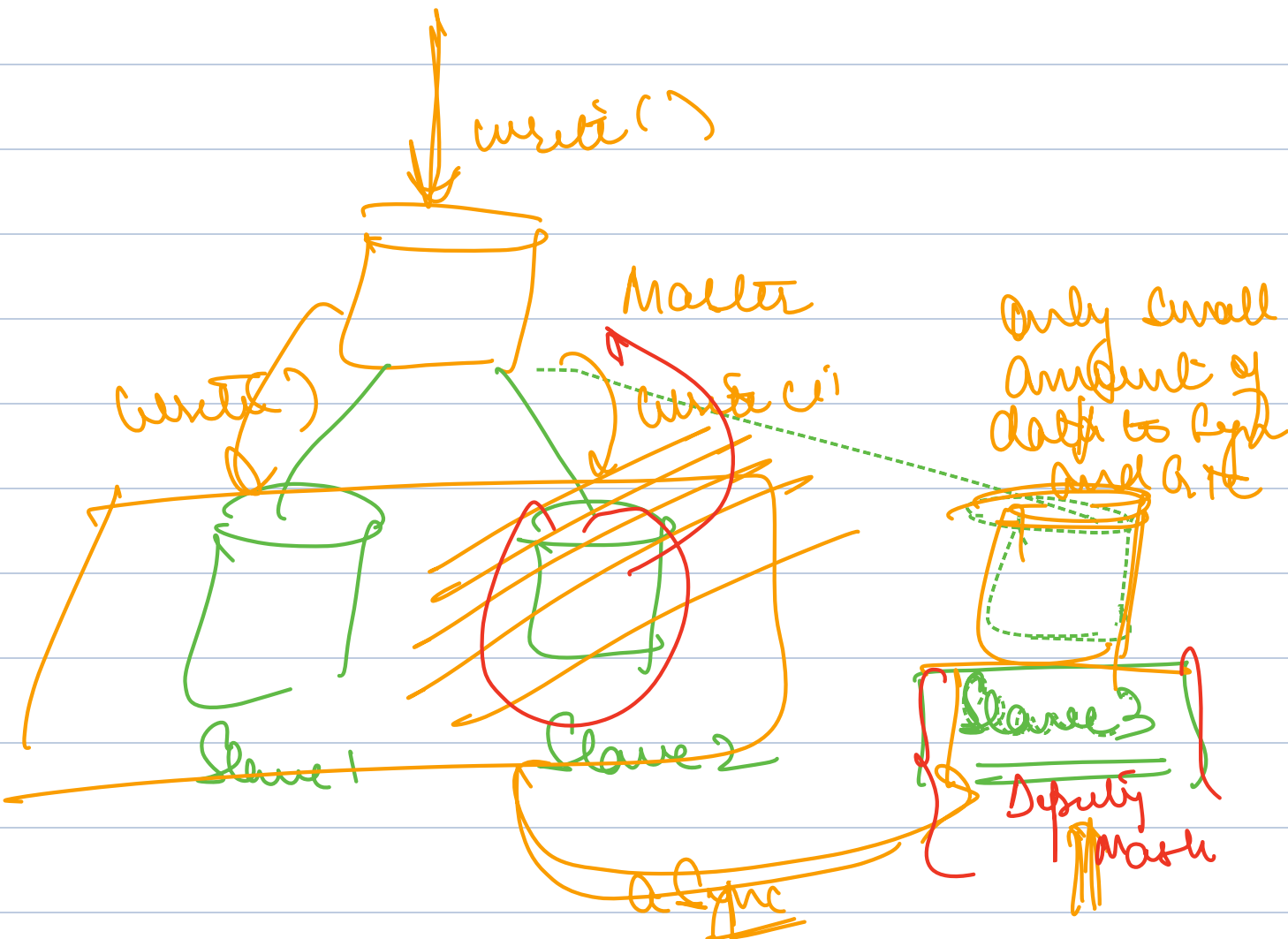
↓

downtime till another machine is set up with all data



⇐ downtime will come to fill all data in this machine.

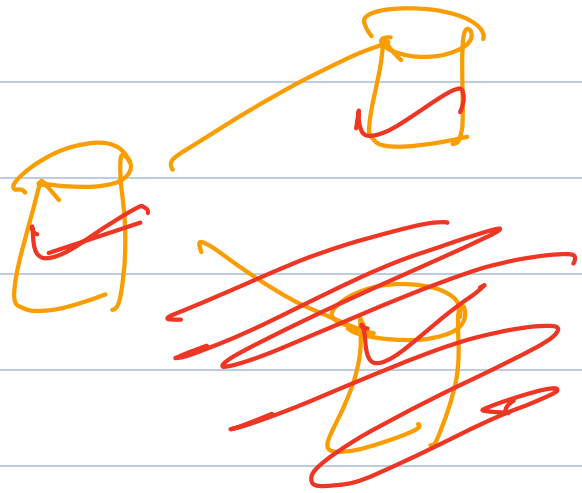
⇒ what if I keep an additional slave for every shard.

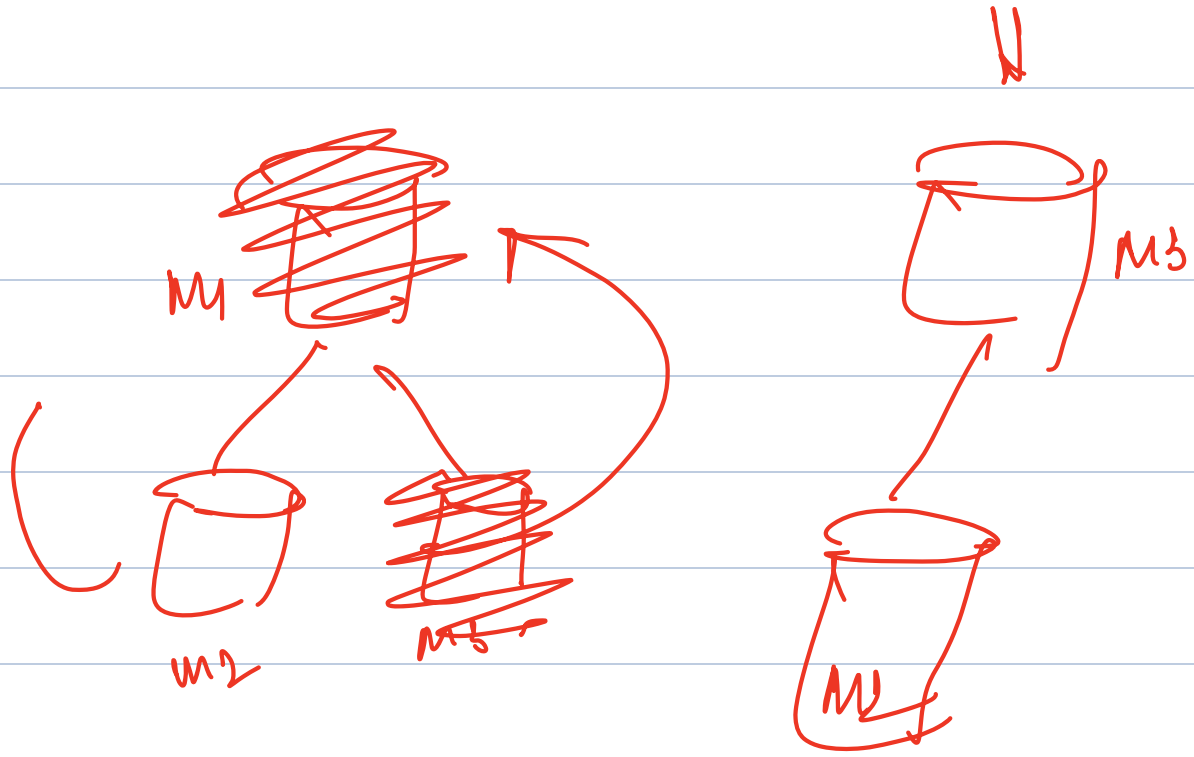


Lower down costs

BUT → additional cost of keeping 1 extra machine per shard running.

extra burden
↳ cost
↳ maintenance
↳ syncing.

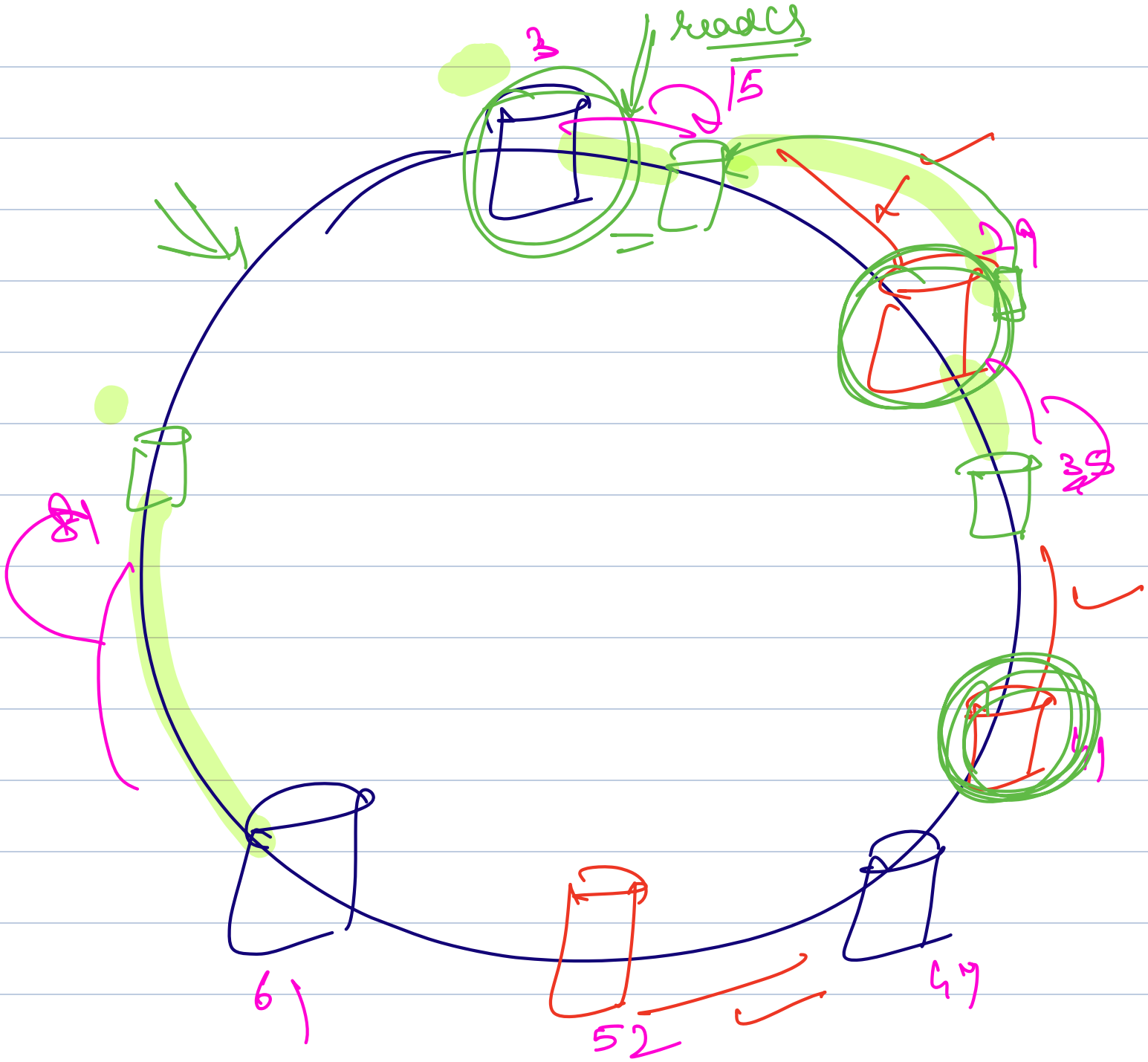




Adding of a new Shard

2 Shard

3 copies



How should new shard be added?

where? → decided By Hash F^n

when? =

~~and~~ How

(15, S3) (35, S3)

array =

[(3, S1) ~~(15, S2)~~ (29, S2) (41, S2) (47, S2)
(52, S2) (61, S4)]

(81, S3)

Should I add directly → ✗

NO New shard will not have data

→ Reads will become unavailable.

→ writes will also be unavailable.

For some portion of data → DOWNTIME

A new shard is added in 2 phases:

1 STAGING PHASE

decide where the new shard will go. w/o putting shard there

(same delta)

we know where shard will go.

copy completes

GTC

decide to add shard.

start copying

copy the data that is going to be unpacked.

downtime

we will take downtime

original shard keeps on handling req.

DOWNTIME

copy the delta (changes since I started copying)

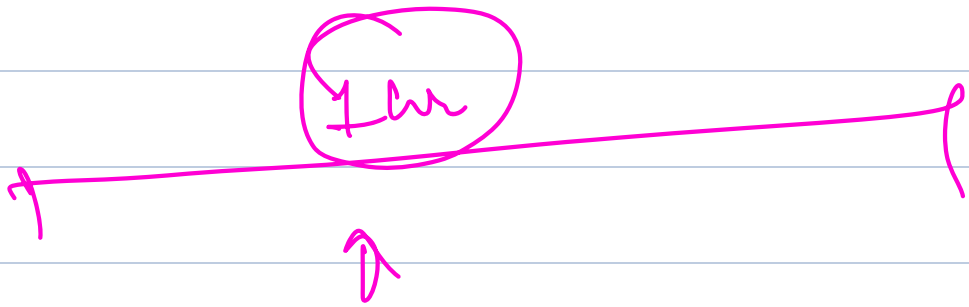
WAL → write ahead log / binlog / commit log

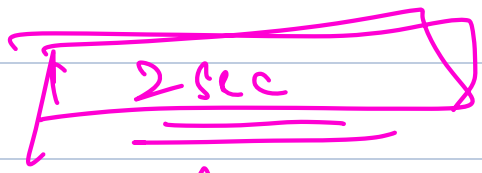
exact thing that Cloves also use to sync

we can't add a new shard w/o downtime

MULTIMACTER ARCHITECTURE

(Break till 10:15)



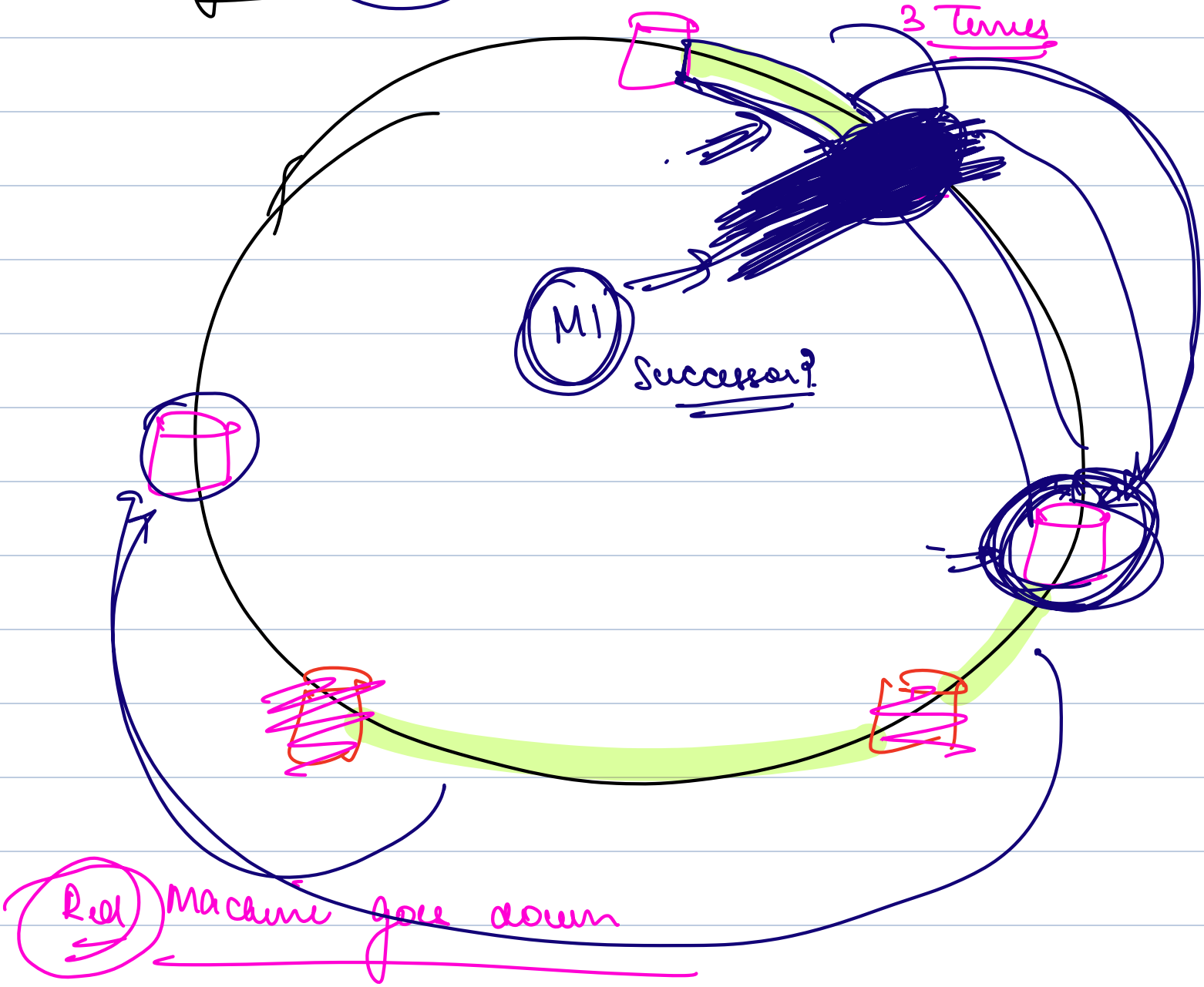


only for the
[illegible] impacter

Let's ignore SPOF

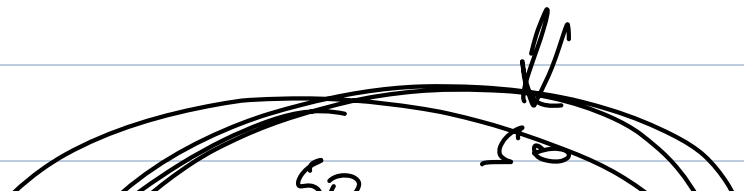
2 Shards

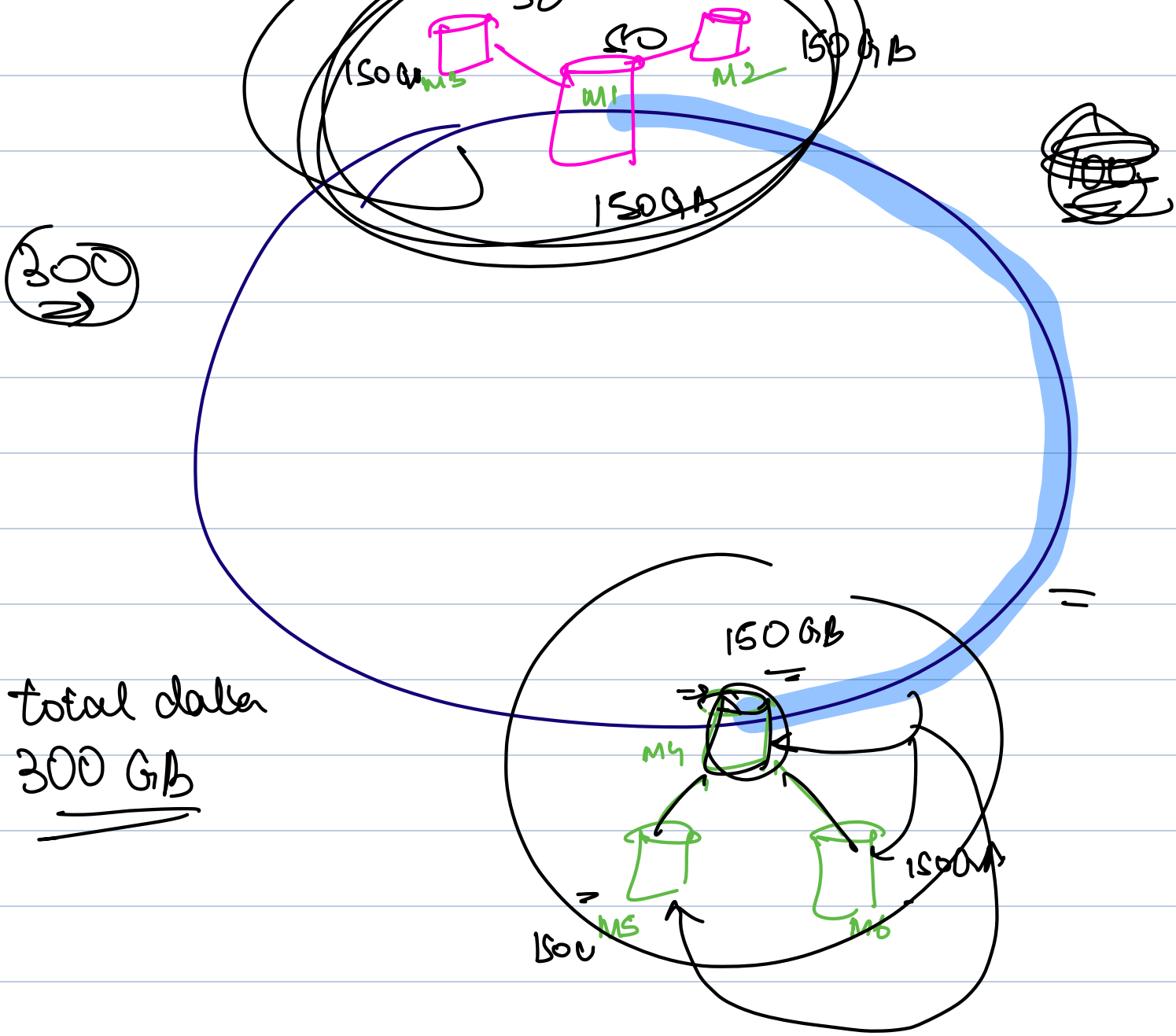
3 Times



⇒ Remove concept of Master Slaves

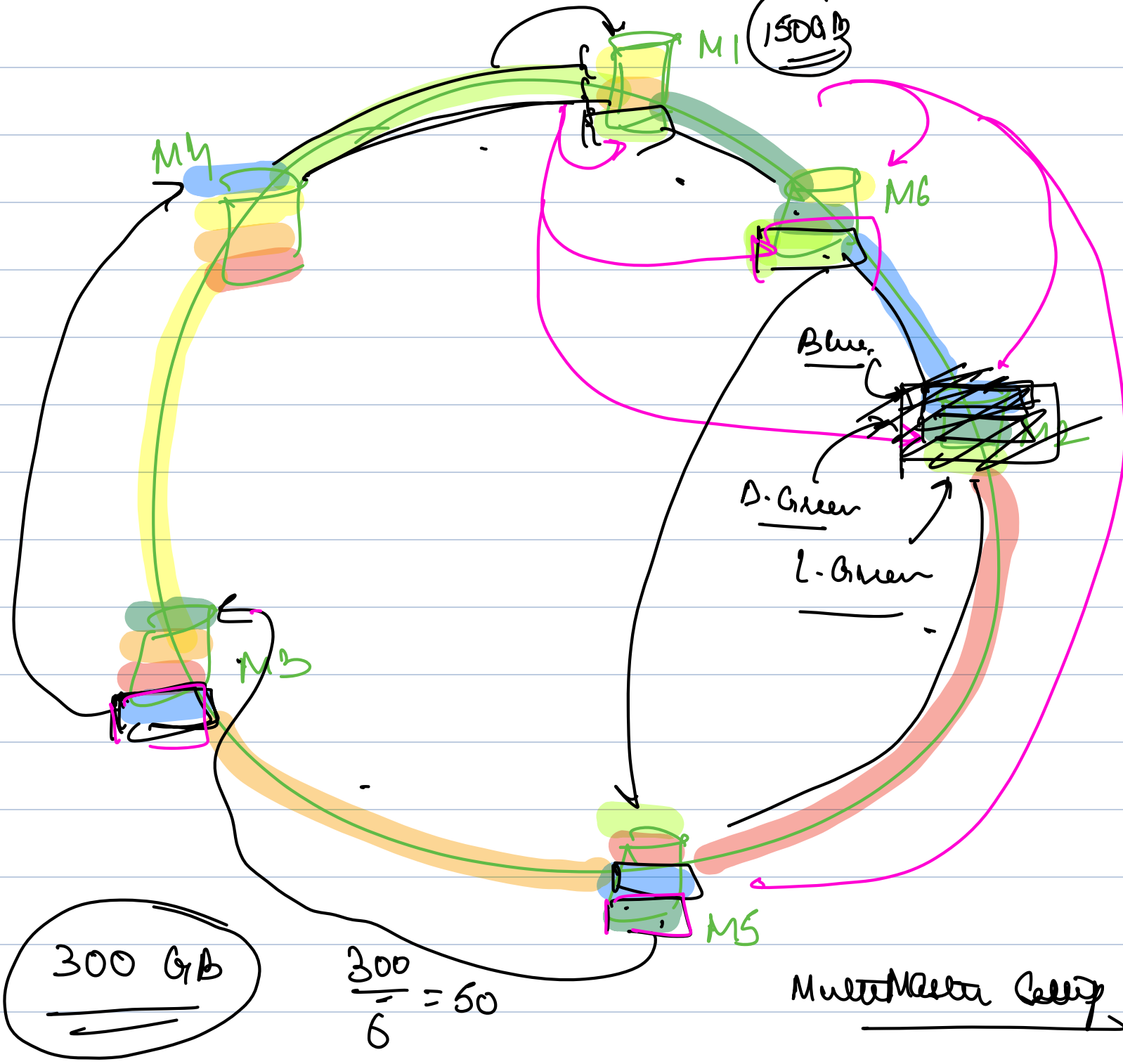
⇒ Everyone is a Master





total data
300 GB

⇒ Put each machine on cons. hashing ring

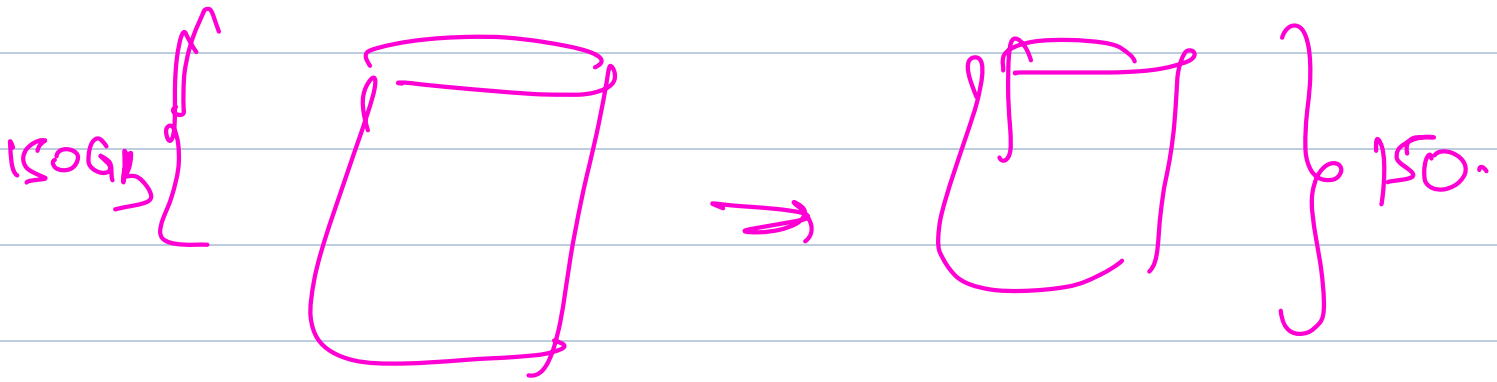


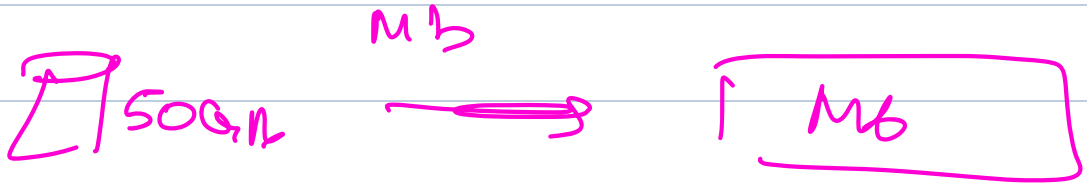
amount of data in each

Same → Yes

Diff → No

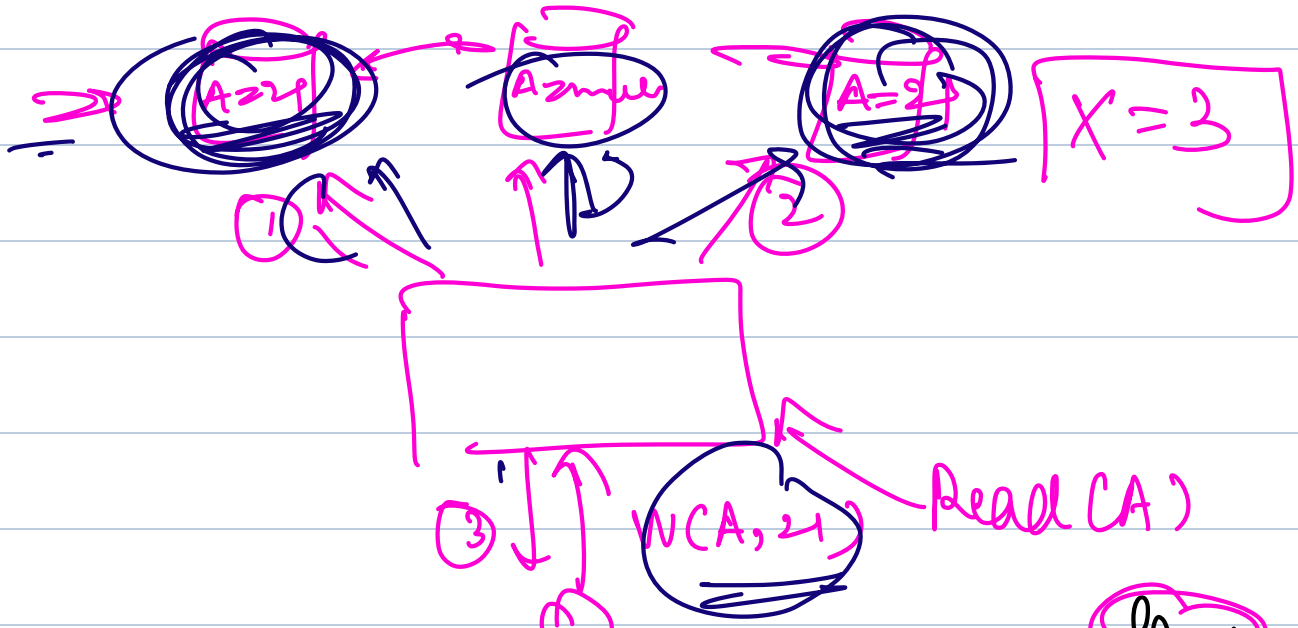
→ Same amount of data
→ Same backups





Tunable Consistency

X (# of machines where data is stored) $= 3$



R ⇒ # of machines to read the same data value from before considering read to be successful

W ⇒ # of machines to write the value to before considering write successful

R = 2

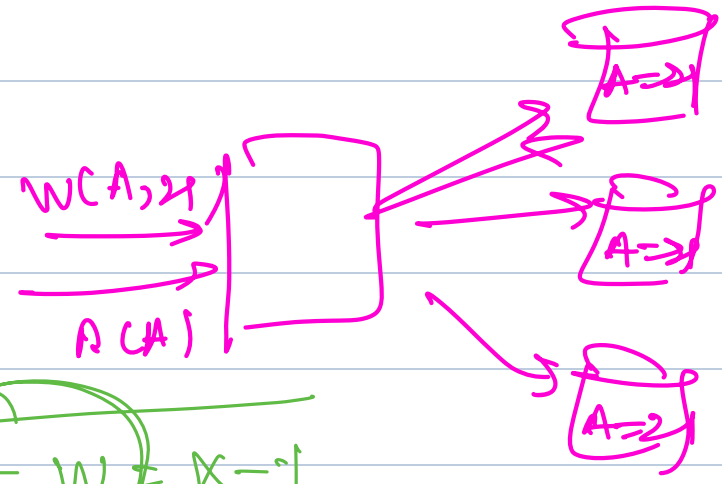
W = 2

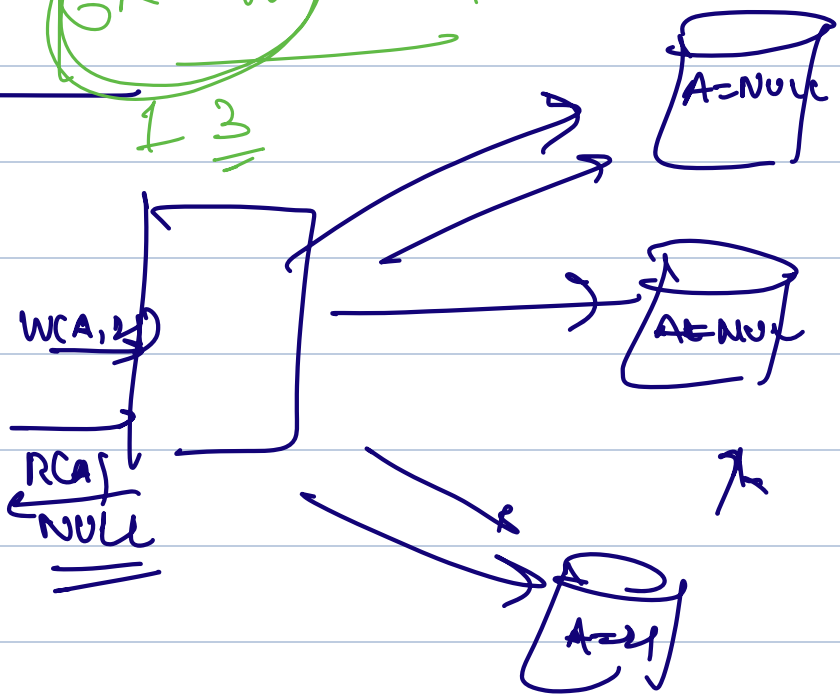
X = 3

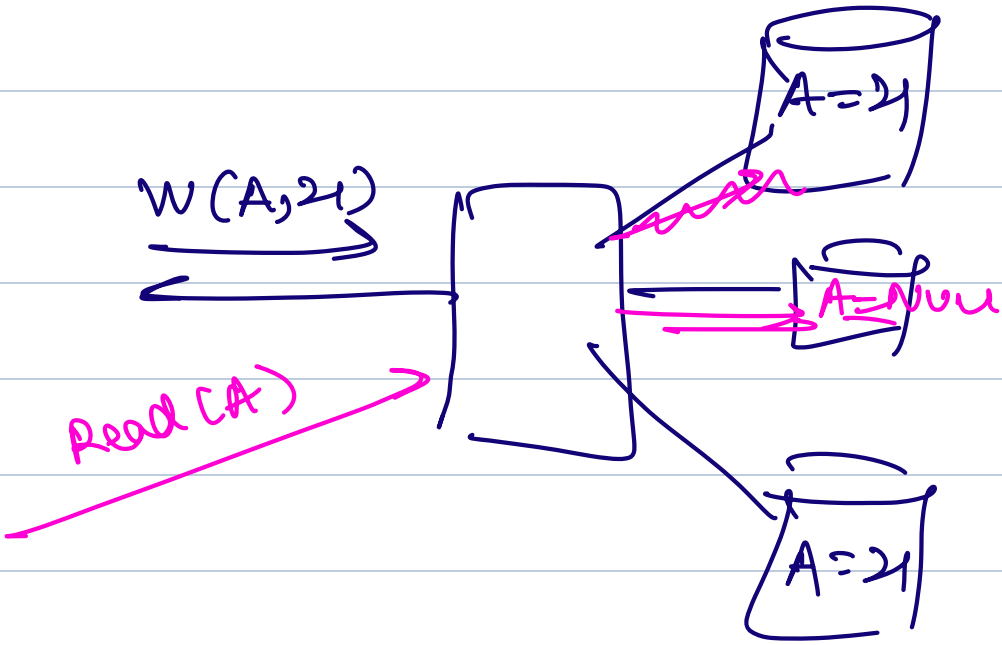
R	W	Cons?
1	1	X
1	2	X
1	3	✓
2	1	X
2	2	✓
2	3	✓
3	1	
3	2	
3	3	

R = W = X − 1

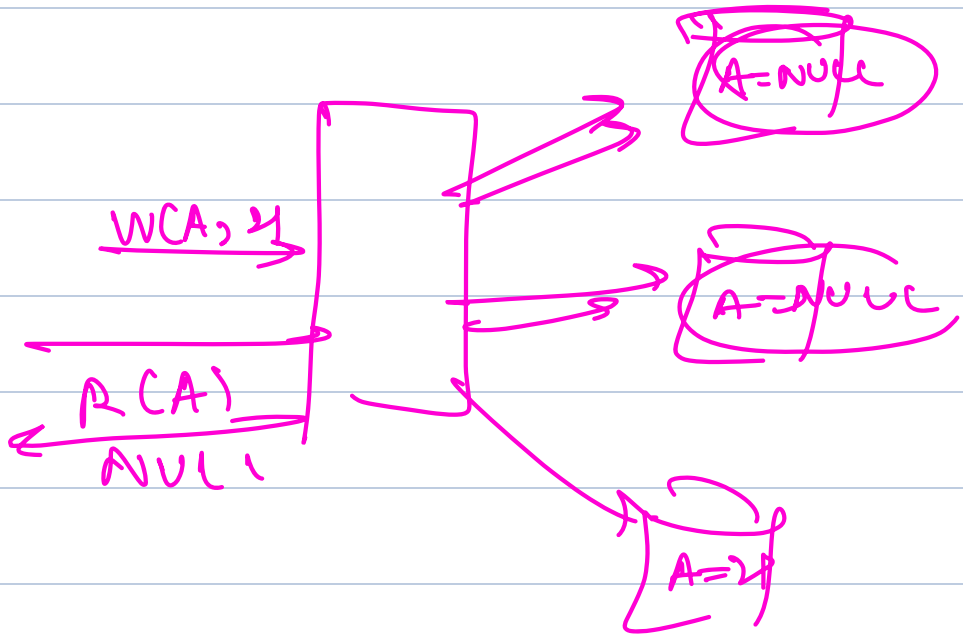
1 3



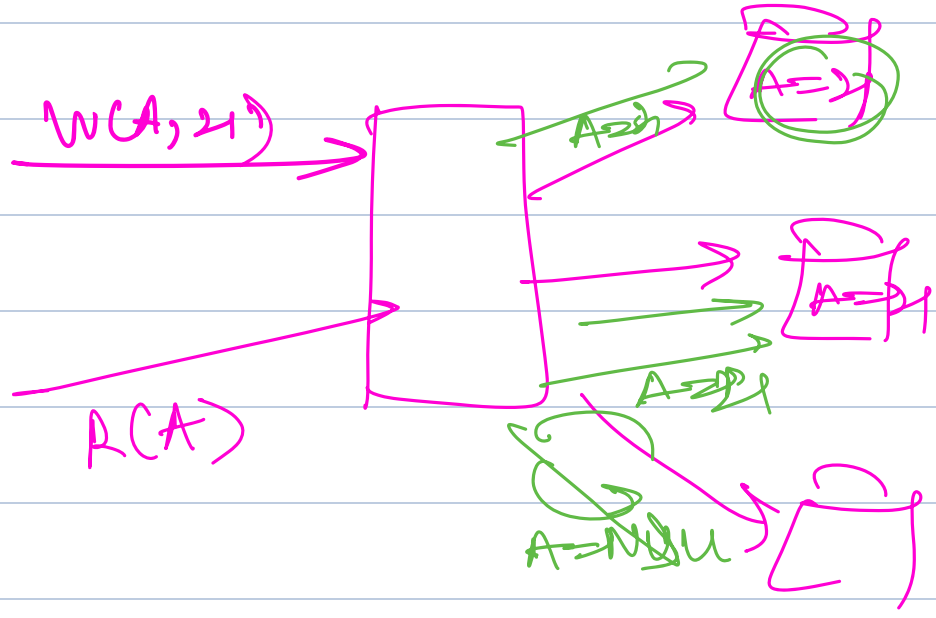




Consistent ⇒ $R + W > X$



(2, 2)



Consistent

(N = 3)

R = 3 W = 1 ⇒ Cons and Fast Writes But Slow Reads

R = 1 W = 3 ⇒ Cons and Fast Reads But Slow Writes

R = 2 W = 2 ⇒ Cons and Inter Reads and Writes Speed.
medi-

⇓

Availabe and v. v. Low latency

R = 1 W = 1

Pros

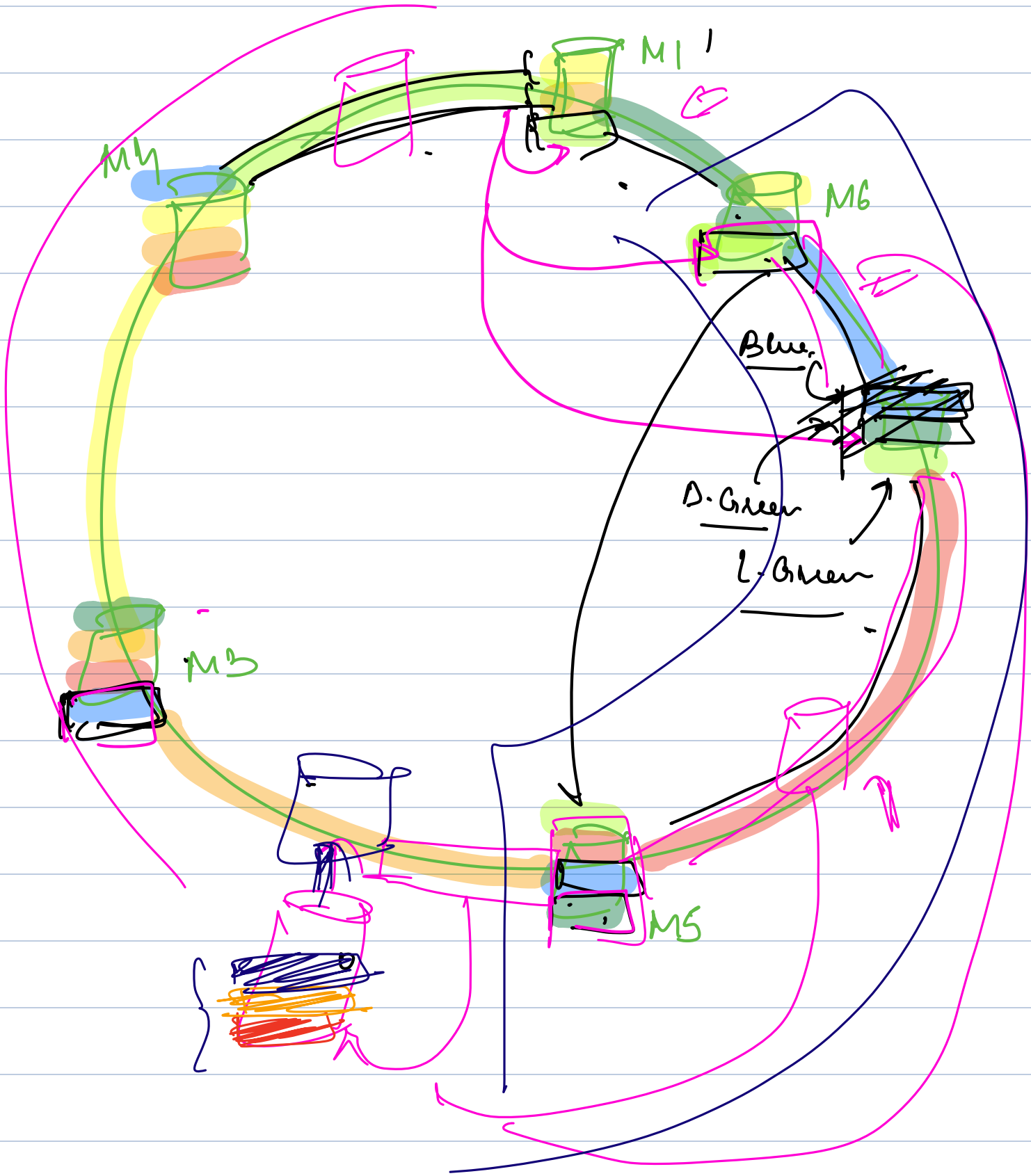
1. Tunable Consistency
2. No reserve machines
   - ↳ cost saving

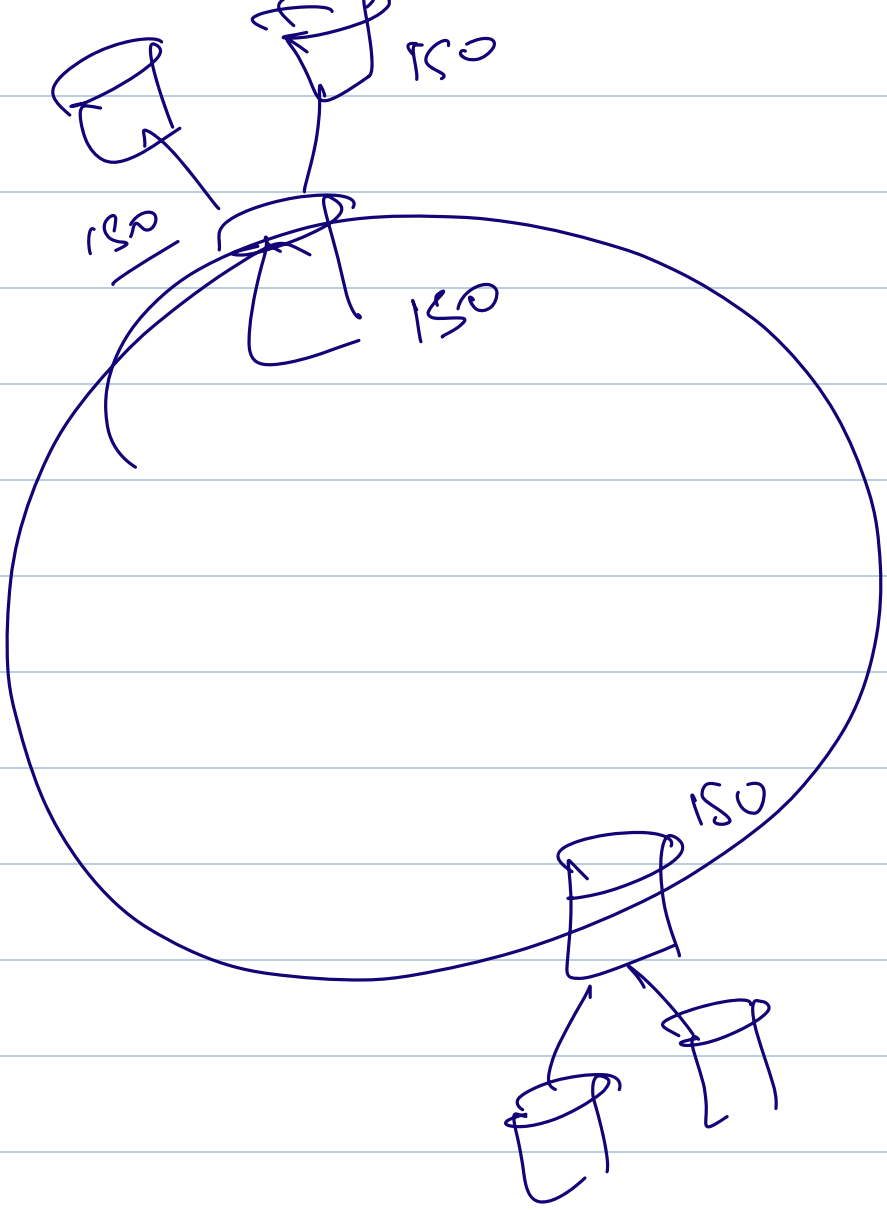
Eg Cassandra, HBase

↳ FB Messenger, Discord

↳ lot of writes + consistent

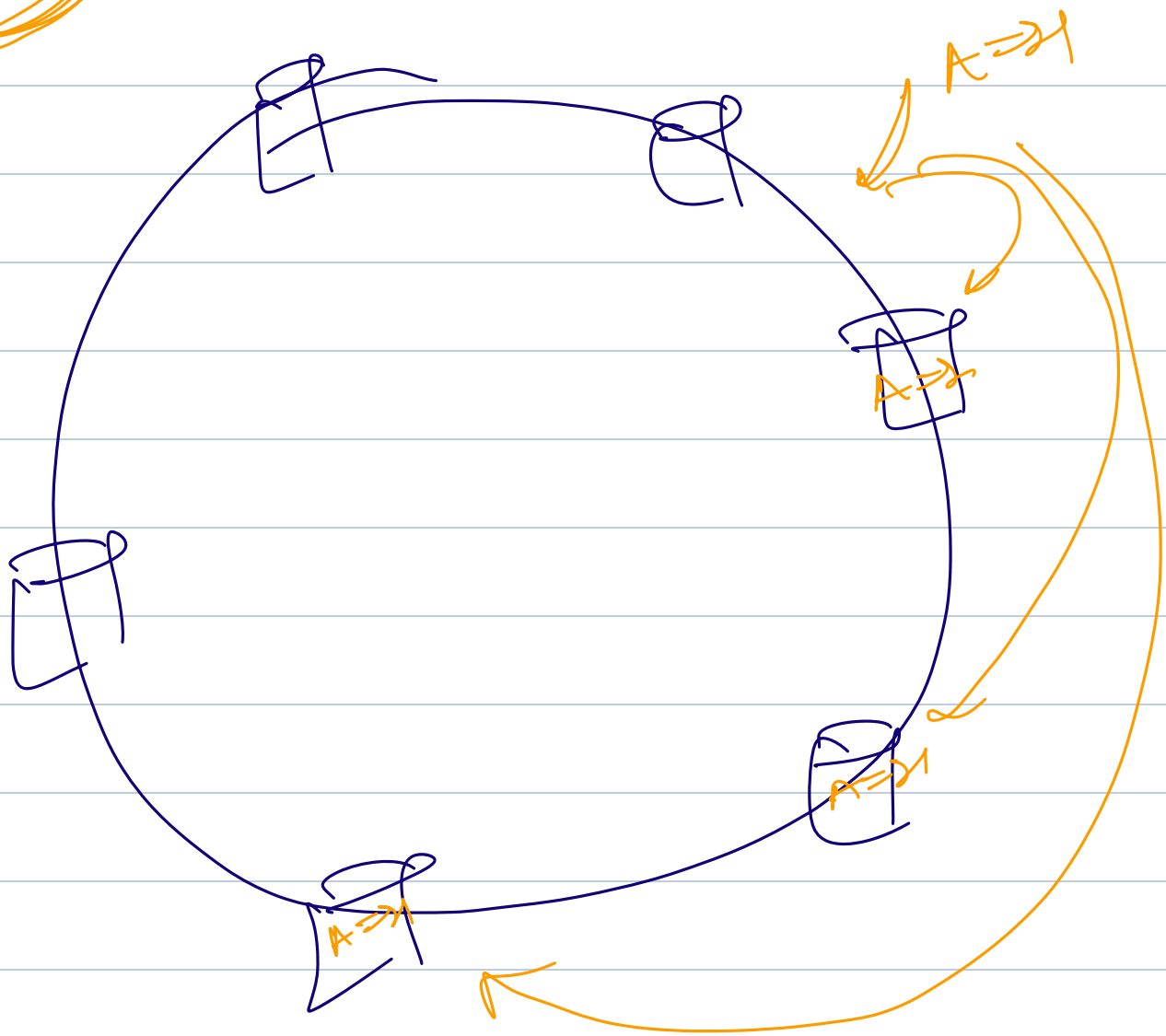
R=2 W=2


M1
M4
M6
Blue
D. Green
L. Green
M3
M5



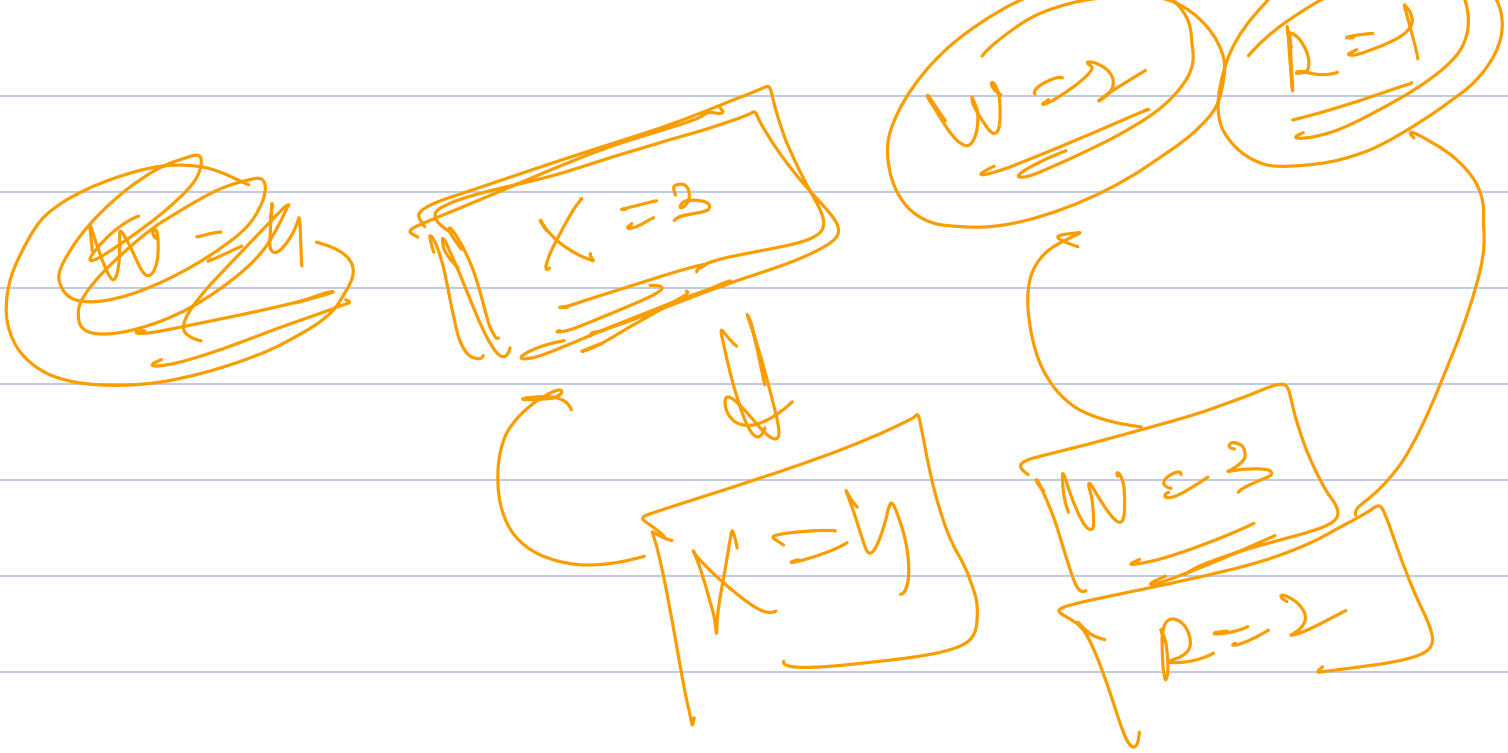
A=31

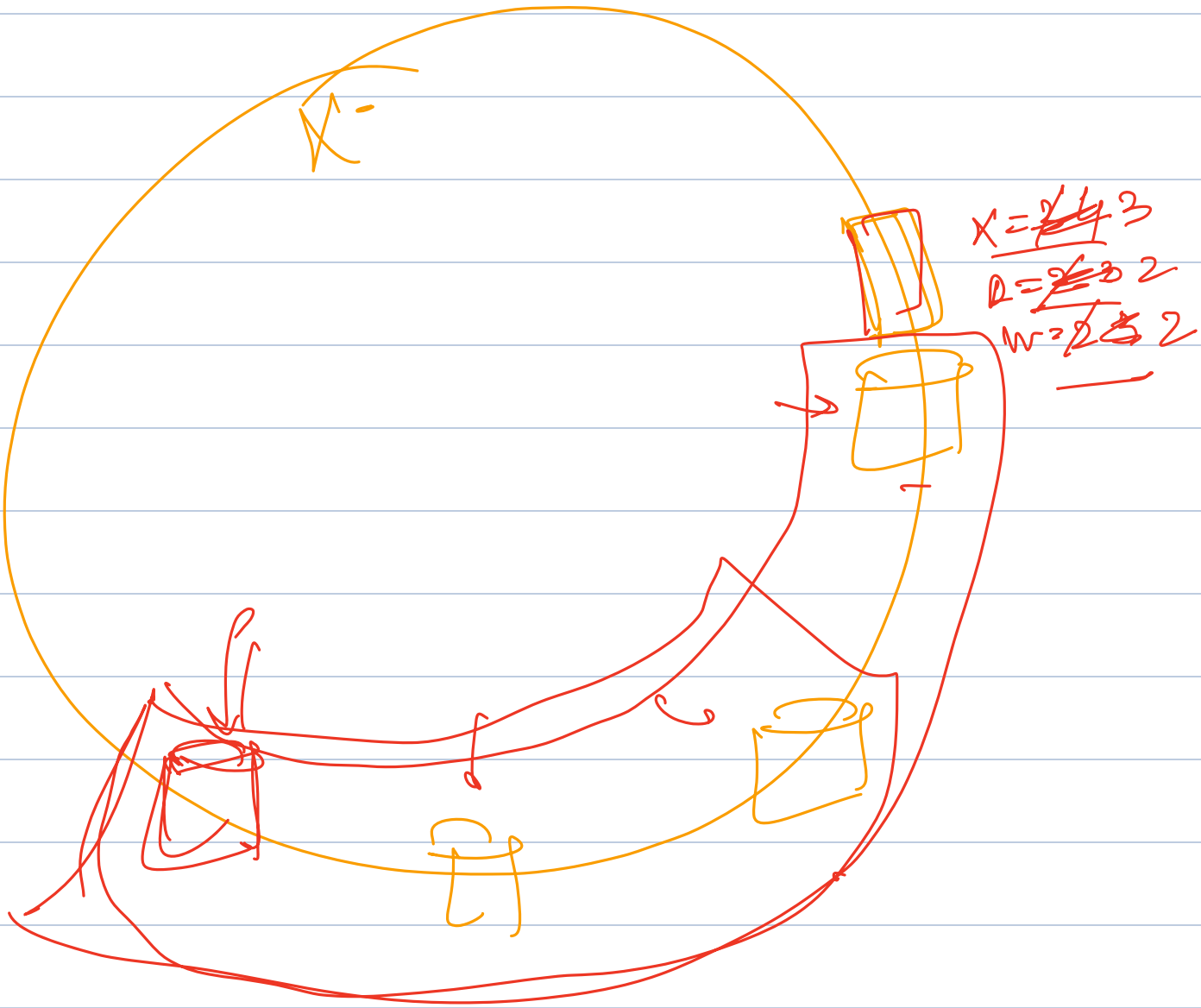
$X \Rightarrow 3$

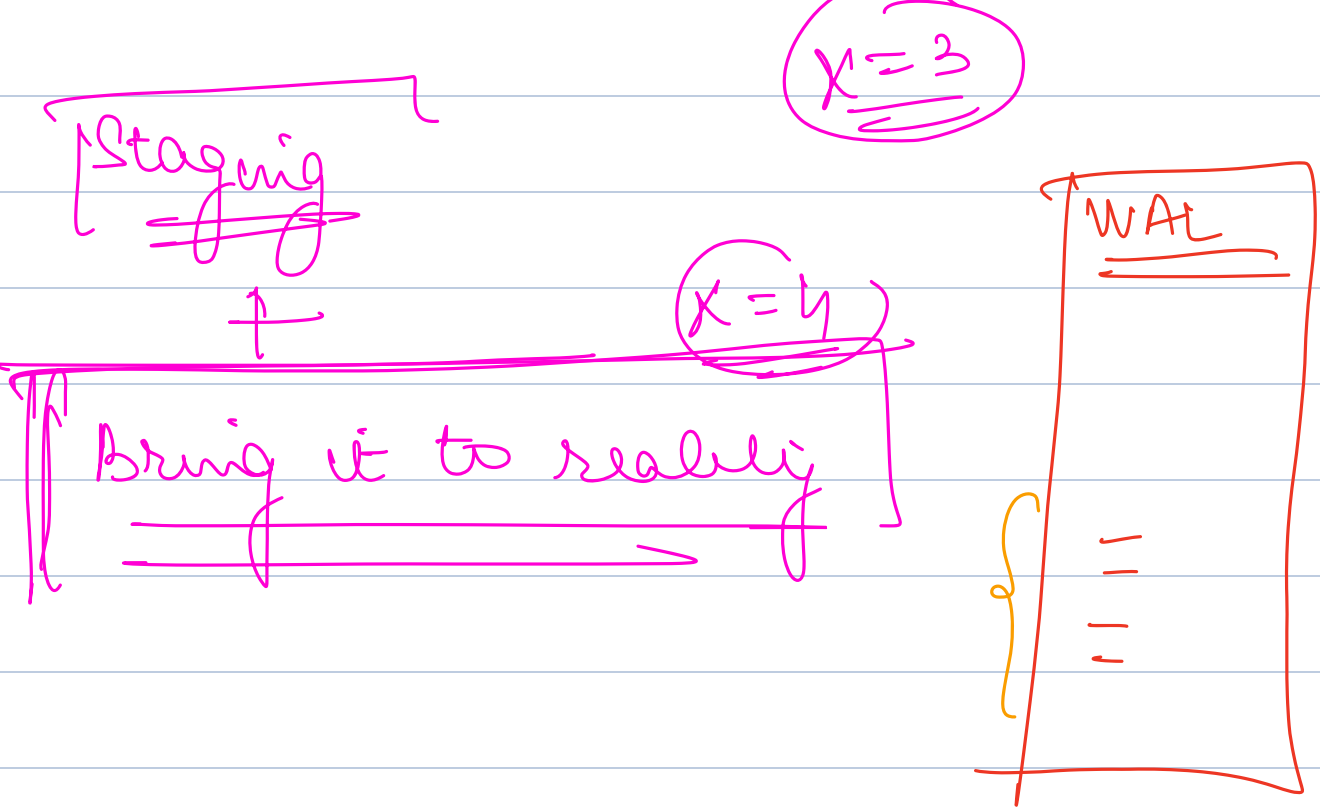


$W \leq X$

$R \leq X$


W = 1
X = 3
W = 2
R = 1
X = 4
W = 3
R = 2


X =
X = 3
R = 2
W = 2


x = 3
Staging
x = 4
Bring it to reality
WAL