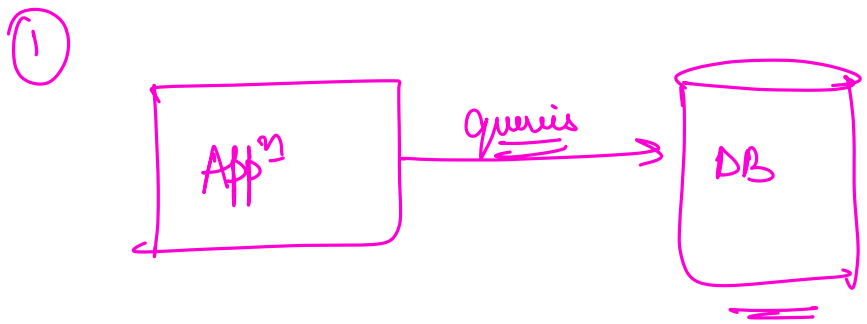
# Complete NoSQL

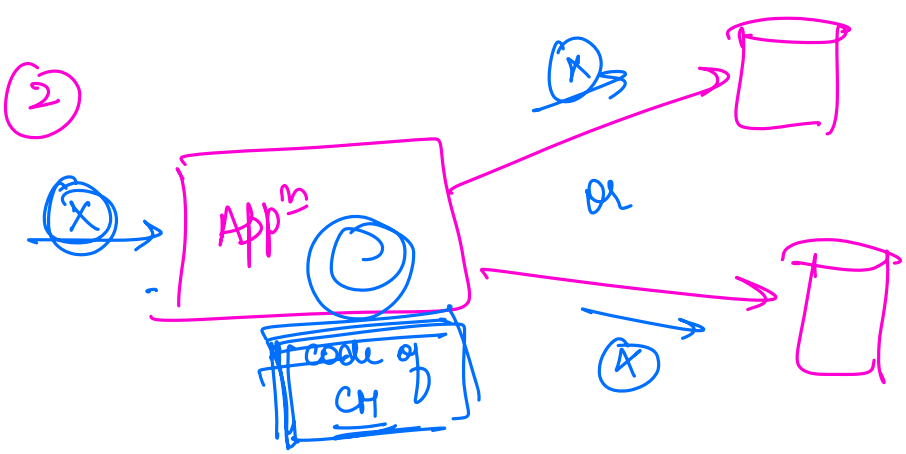
① Problems with Master Slave and Sharding
↳ Some amount of downtime

② Multimaster Architecture ⇐
→ Cassandra

③ Tunable Consistency
↳ Set some params C → consistent
A → available




bottleneck
(i) Size of data
(ii) # of queries



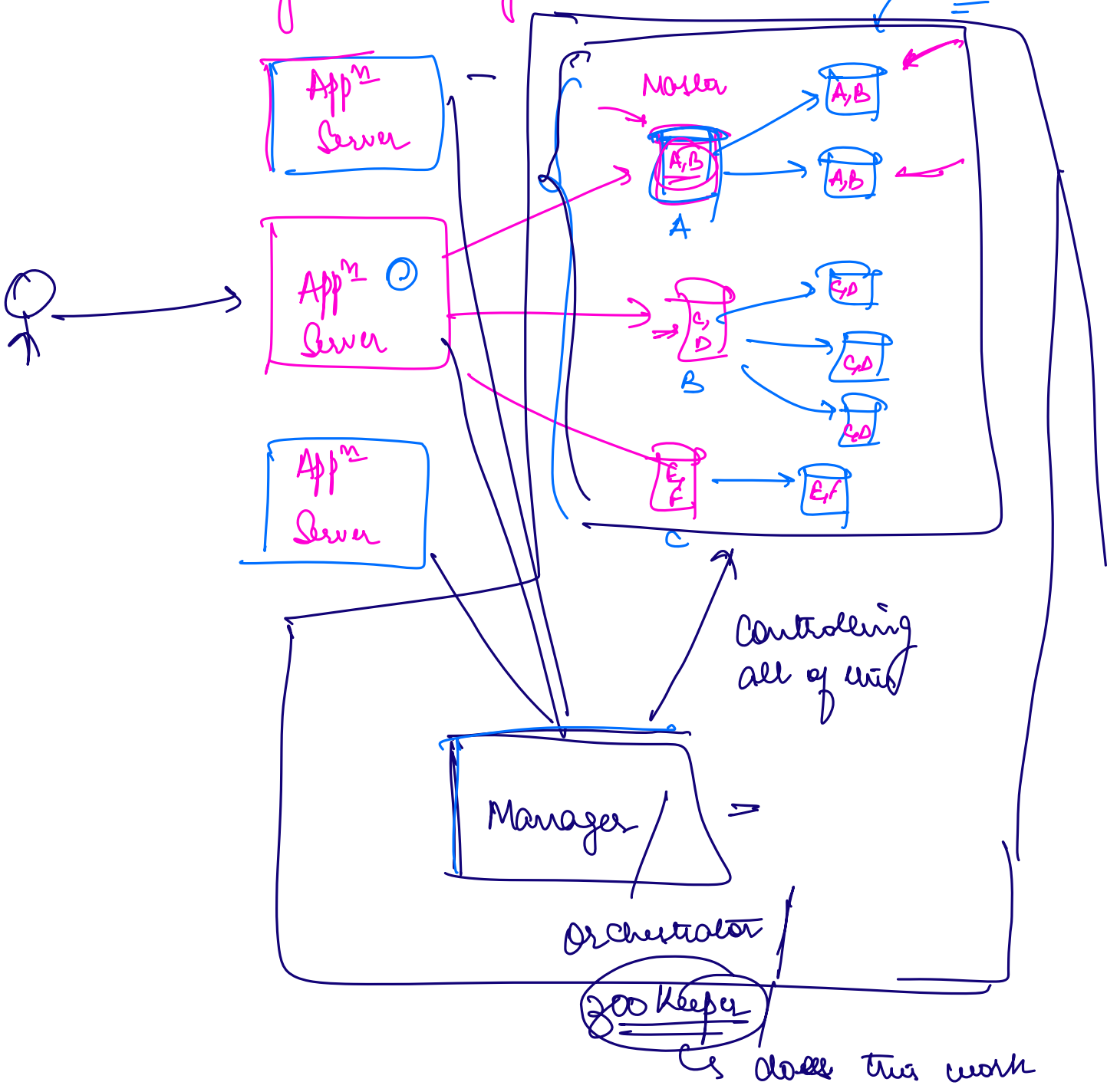
SQL DB by default don't support sharding / replication.

There has to be someone who does:

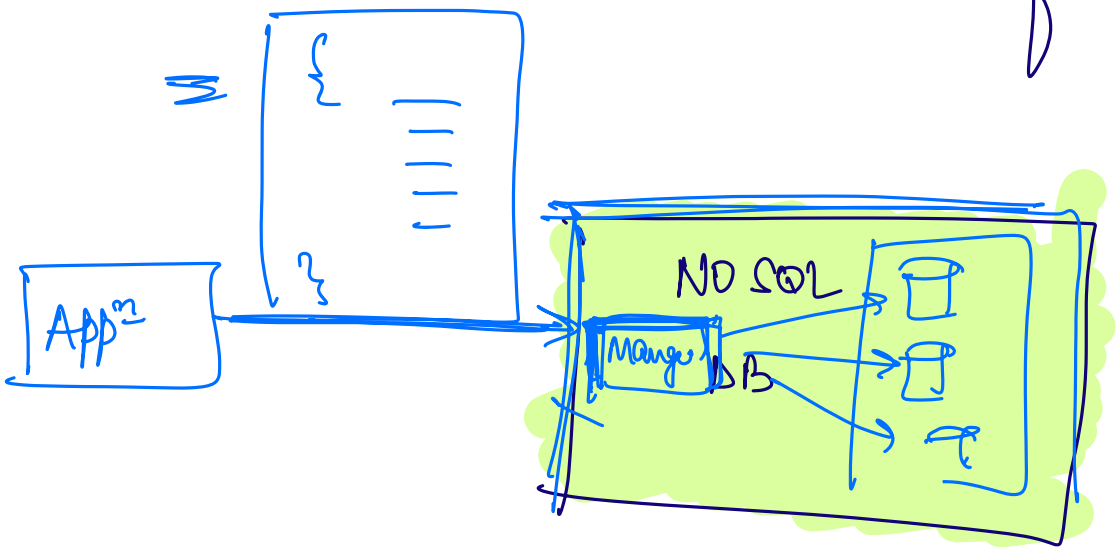
(i) Creating a new shard if needed.

(ii) Create another replica if needed

(iii) given a key, tell which shard has it



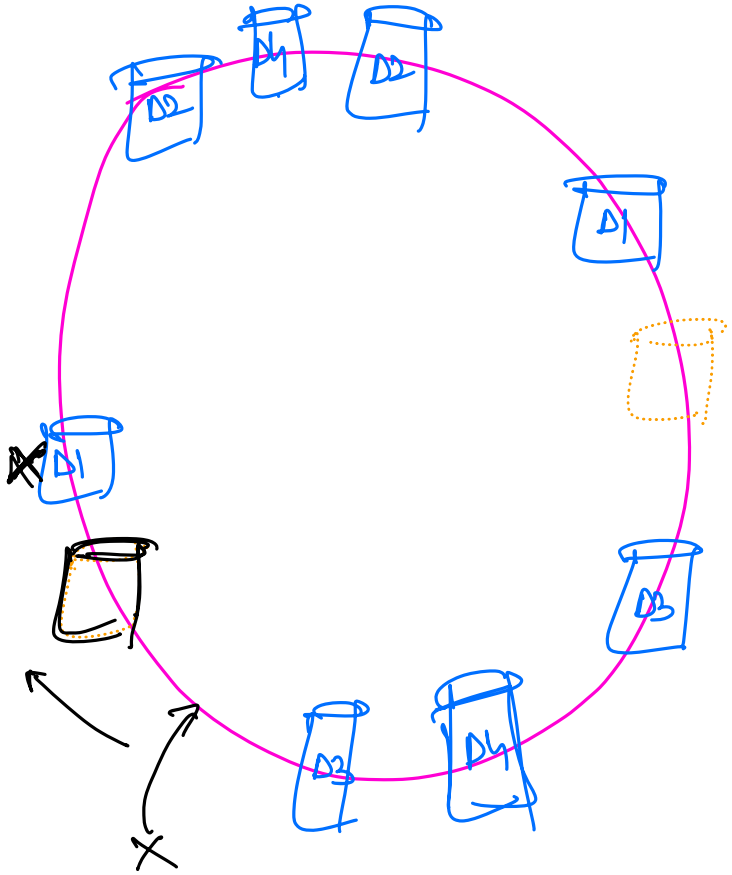
for Kafka




We consider a NoSQL DB as a blackbox where we only care about the f^n that DB offers.

How a DB Manager may work

1. Create new shards
2. more replicas of each shard
3. CH work




A machine is not able to handle all of data:

ADD another Shard.

① They bring machine to Chain right away.

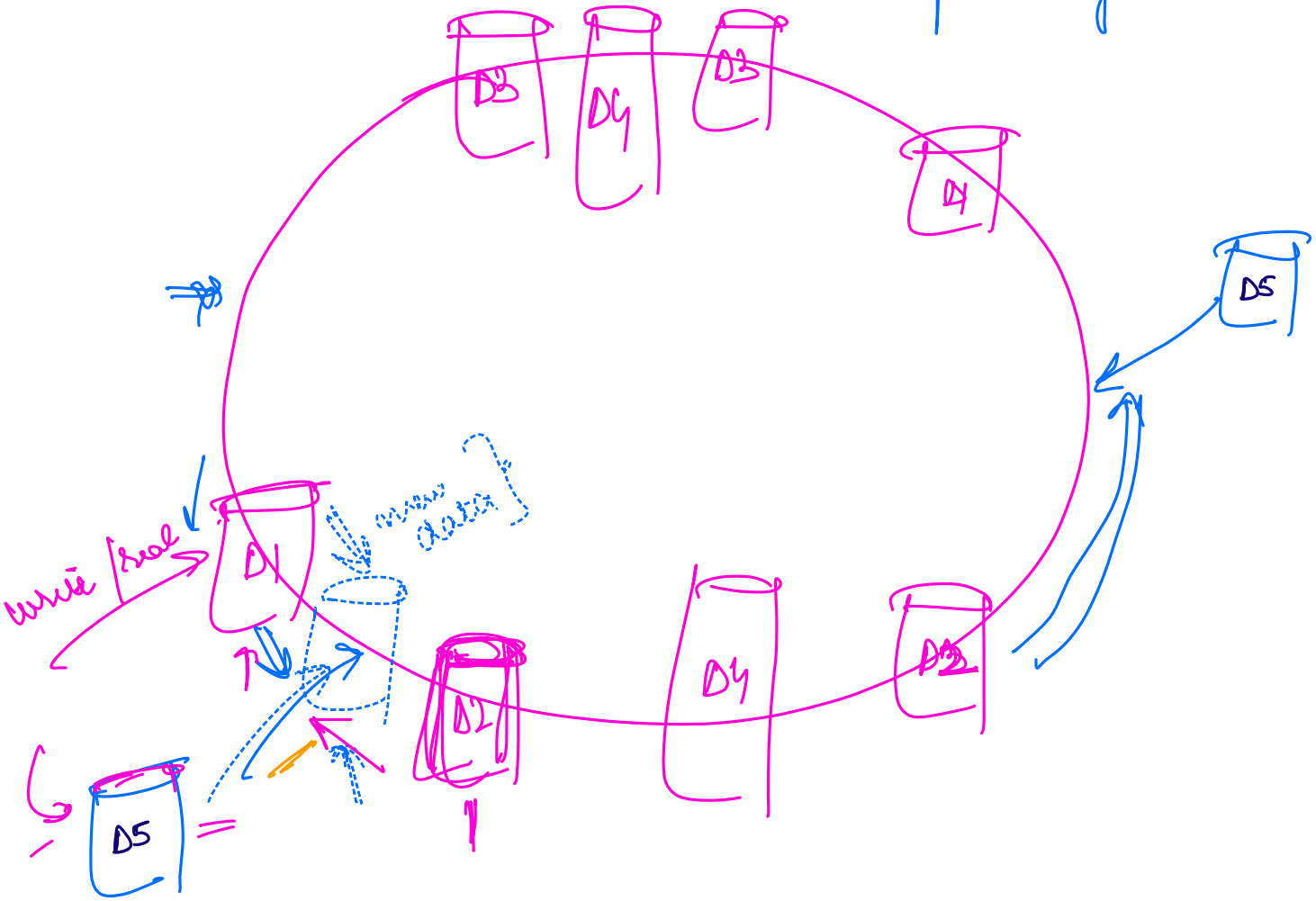
READ X ⇒ X doesn't exist

Available ⇒ ✓

Consistent ⇒ X

(2) First I fill the data into new shard and then bring to the chain.

Staging Time: when I am getting my new shard warmed up (ready)




Staging Time ⇒ Transferring data from older shards to new shards

$\frac{2}{3}$rd of peak capacity

Write

a. On new one:

Read

⇒ Old shard keeps on handling the

what if a read comes. The old one won't have value.

(b) On old one

⇒ Once the new shard is live, I will have to copy the newly created data.

↳ till not done, my reads can be inconsistent

(c) Write to both

↳ writes will become slow

---

When a new shard is going to come up, there will be a lot of work to be done, to get that shard ready.

⇒ At this time, my system may be slow.

① System is overloaded

② at that time, you are adding an additional overhead of Sharding

↓

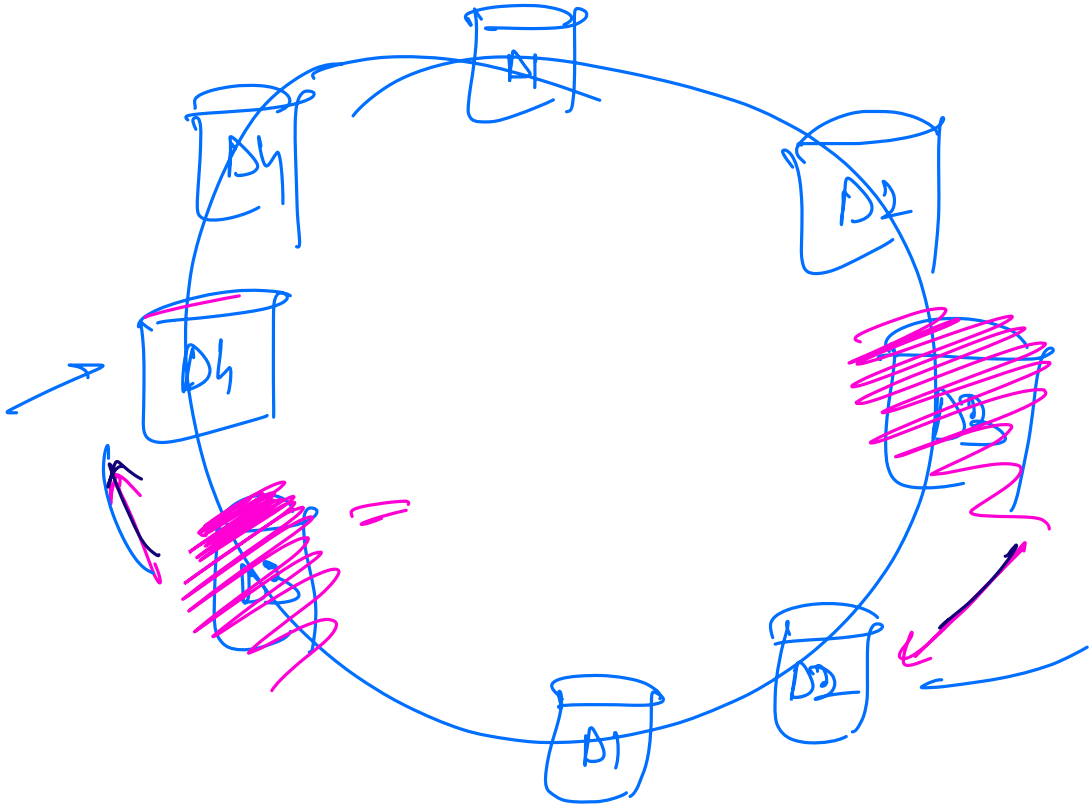
System may get impacted heavily on perf.

DELETE A SHARD

→ ① Load has reduced

→ ② Master machine of the shard may itself die.

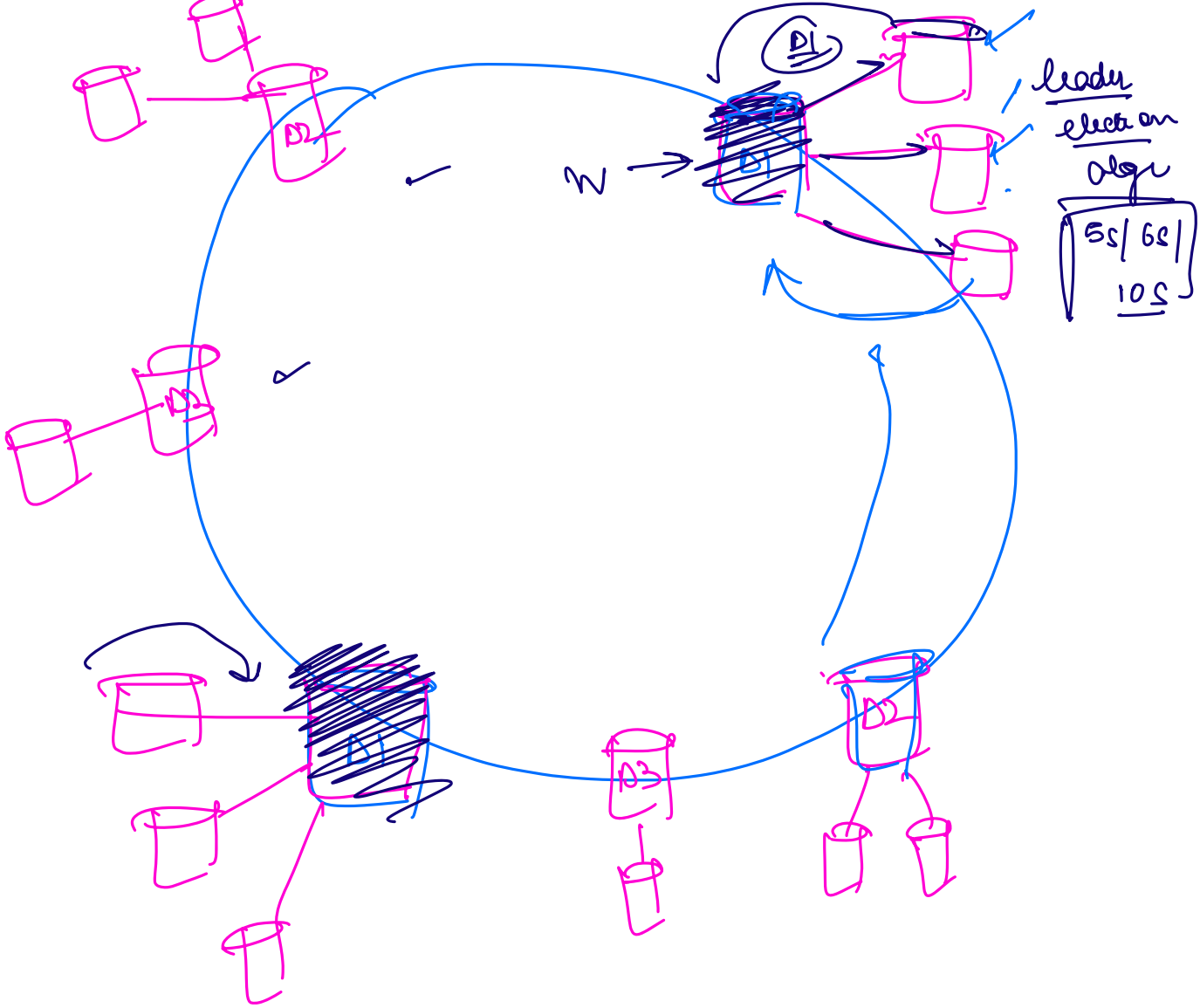
LOAD HAS REDUCED




(1) we will be transferring the data from (D3) to (D2) and (D4).

| Read | Writes |
| --- | --- |
| (1) new shard | (1) write to new shard. |

⇒ deal b/w consis and avail




⇒ Machine has died

- I don't need that many shards
- promote one of the replicas as master

## Write

① write to all replicas

↳ writes will become slow.

↳

② keep my write on hold

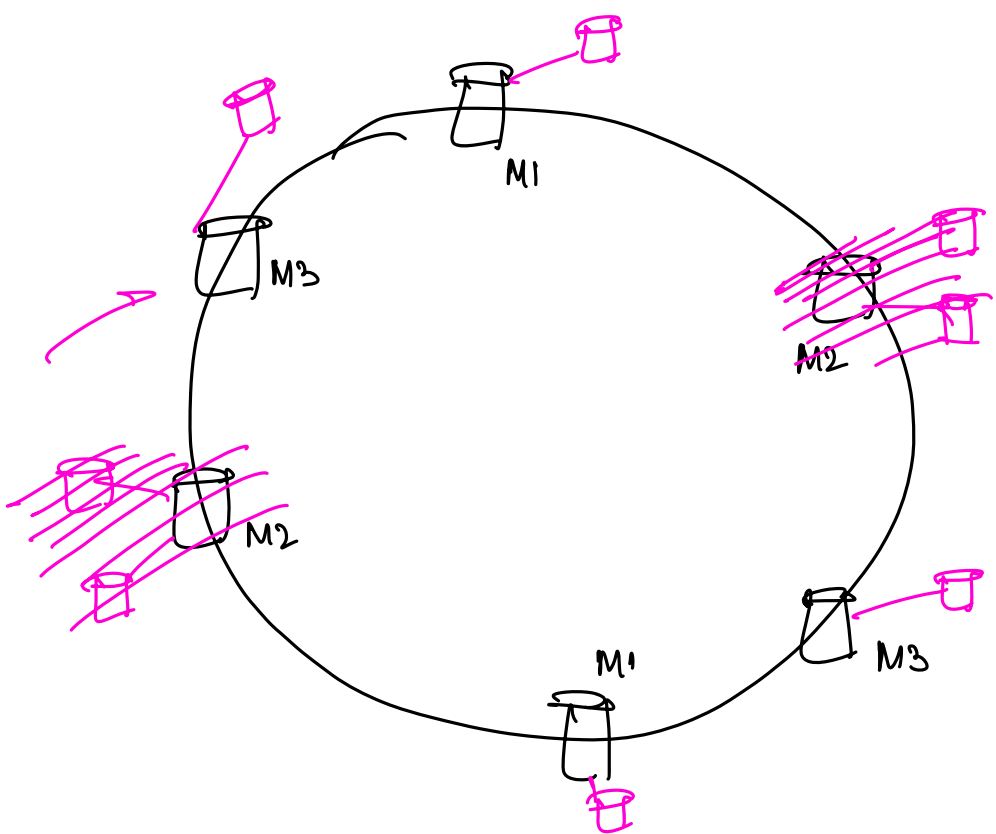
→ unavailable.

## Read

⇒ can still happen on replicas

---

Is there a cleaner sol^n



Till Now

for every key ⇒ there was a dedicated shard

then why did I need replica machines?

⇒ To handle SPOF
⇒ To handle more queries

100 GB

10

≈ 10 GB
+ 30% buffer
≈ 13 GB

⇒ if load inc, inc replicas.
⇒ if data sej, inc # of shards.

---

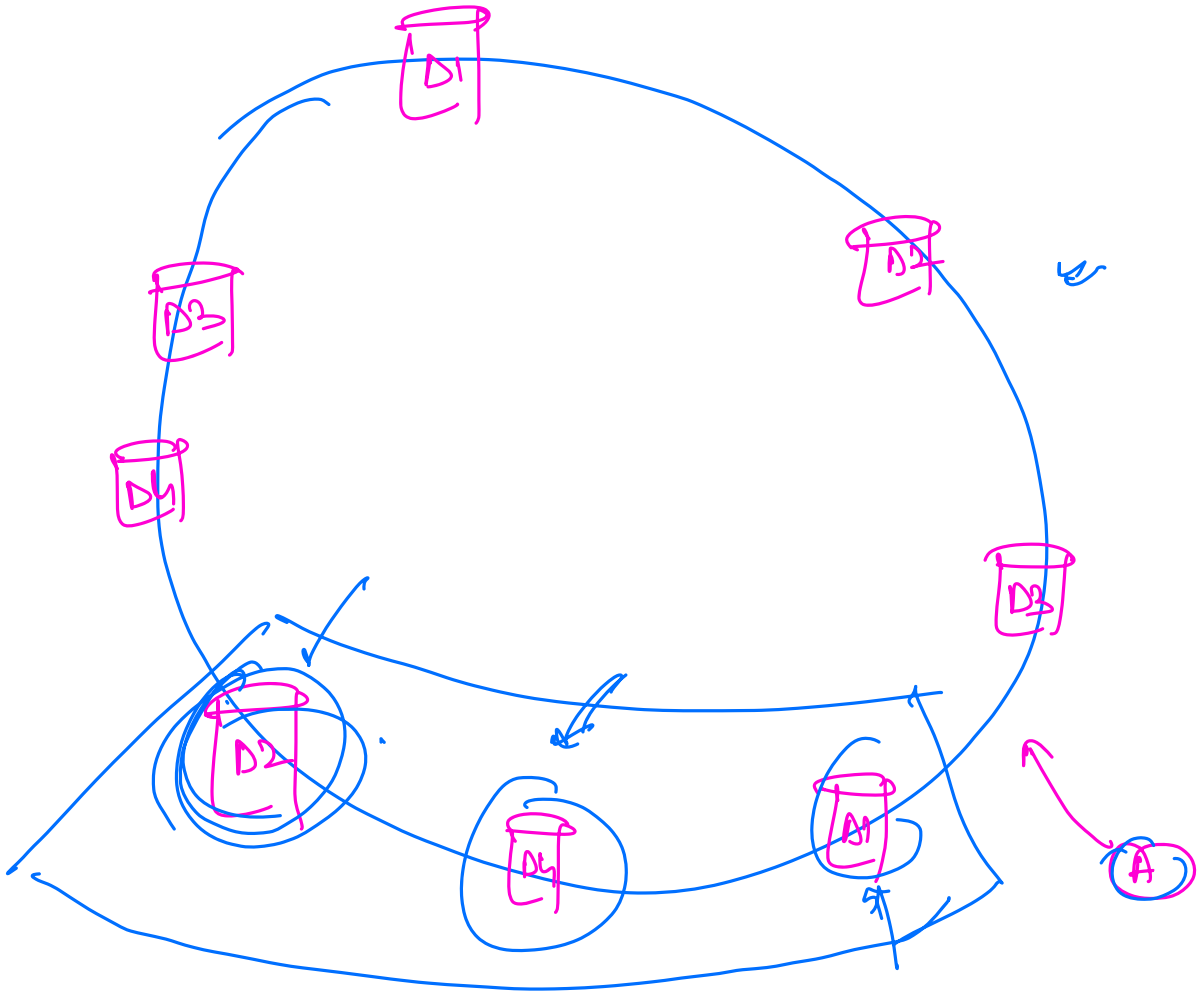
Multi Master Architecture

→ remove the need of replicas

⇒ # of machines have reduce

→ don't associate a key to only one shard, instead associate that to x shards.

(assume x = 3)

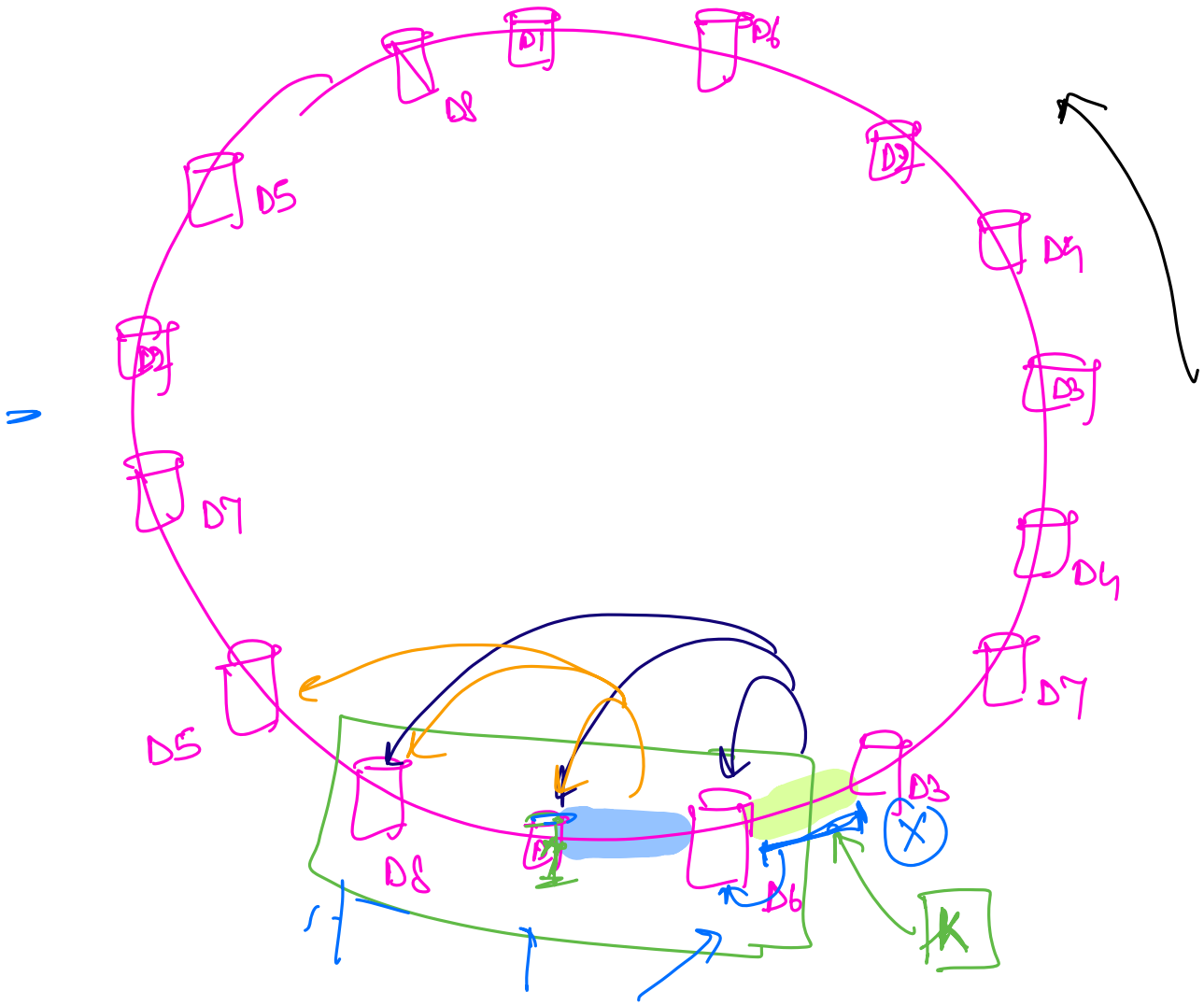



① Based on CH, Key (A)

G, D will write its value to next X machine

⇒ Now instead of 1 owner, there are X owners

⇒ No need of replicas. Because anyway data is being written at multiple places.




⇒ I have just made my clones also as threads.

$$X = 3$$

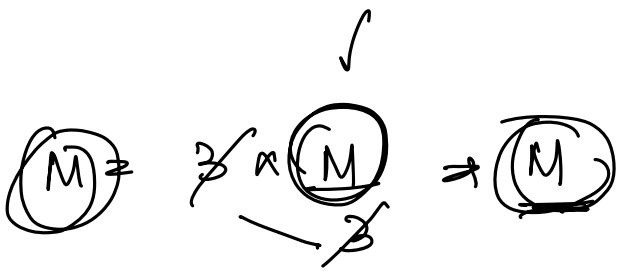
K ⇒ 3 owners: 6, 1, 8

Why we needed replicas earlier:

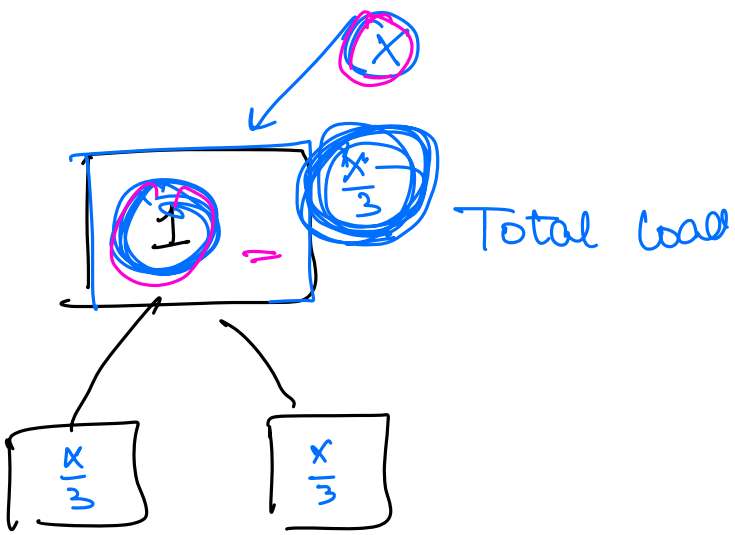
~~1 Backup~~

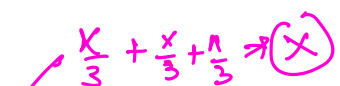
~~2 To share load of read queries.~~

→ distribute amongst all masters that own the key.

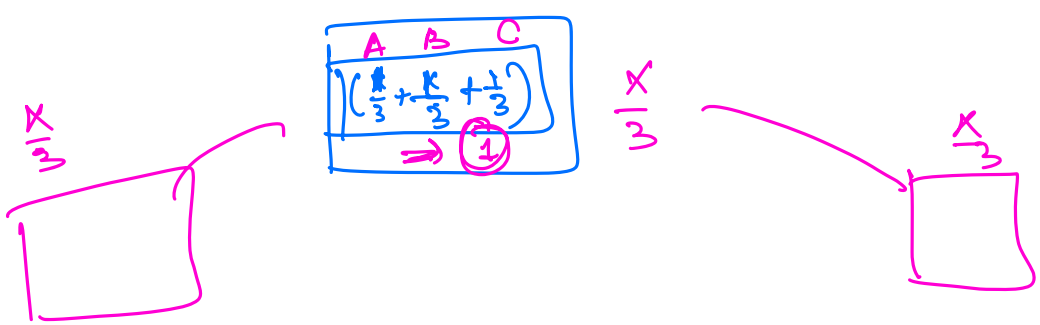



earlier →




now

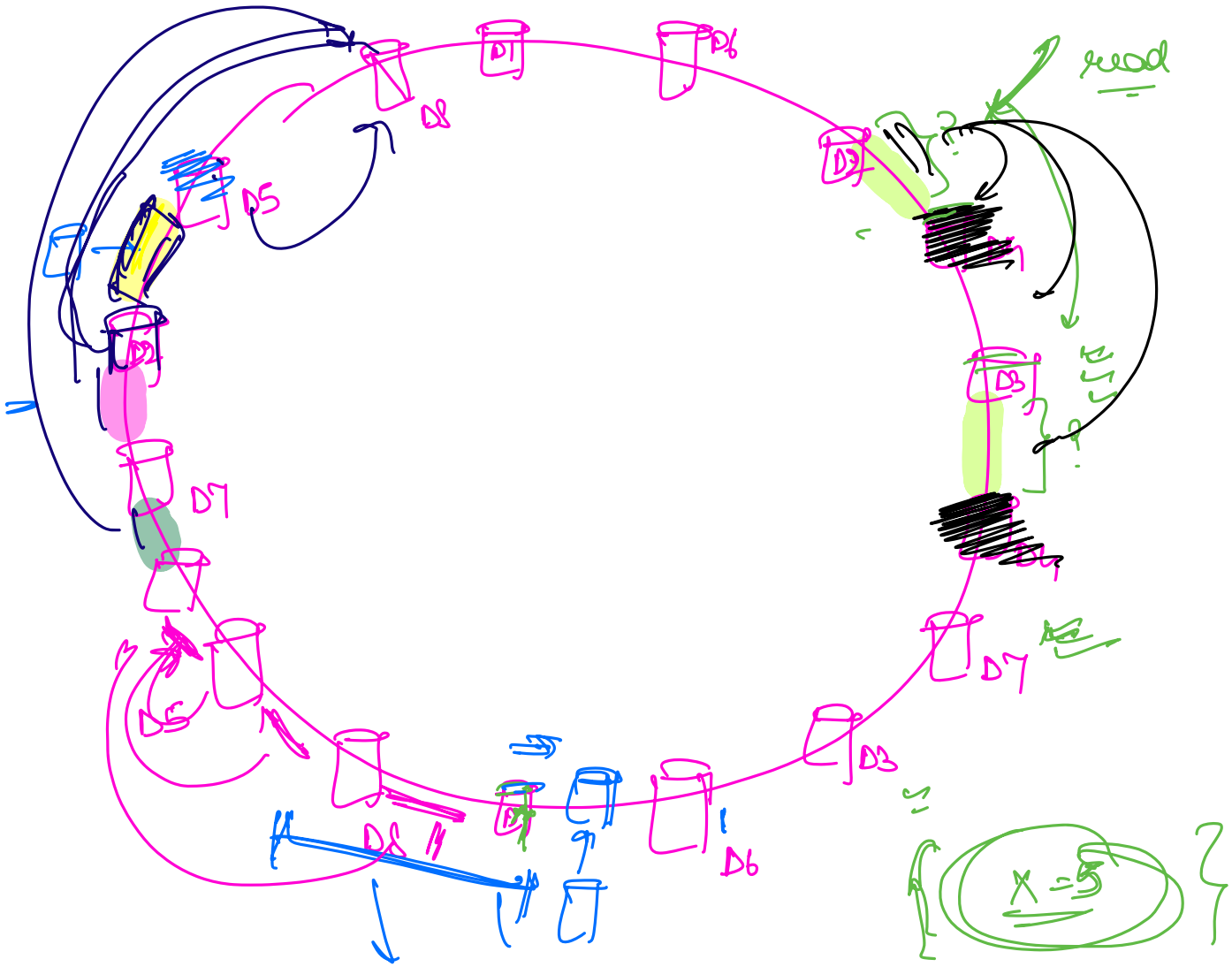


P

Problem with M-S ⇒ issues at time of adding/ removing a board.

for each machine, there will be x dummies

# DELETION OF A SHARD




for the Keys :

Read

→ no need of leader elec

→ just go to D3 :

Write

just write to next

X machine

adding a new shard

Multi Master Architecture

⇒ Algo : Cassandra

each key has
X masters

Master - Slave Architecture

each key has 1 master and
each master may have slave

In M-S

(1) Consistent

write to master as well as all replicas

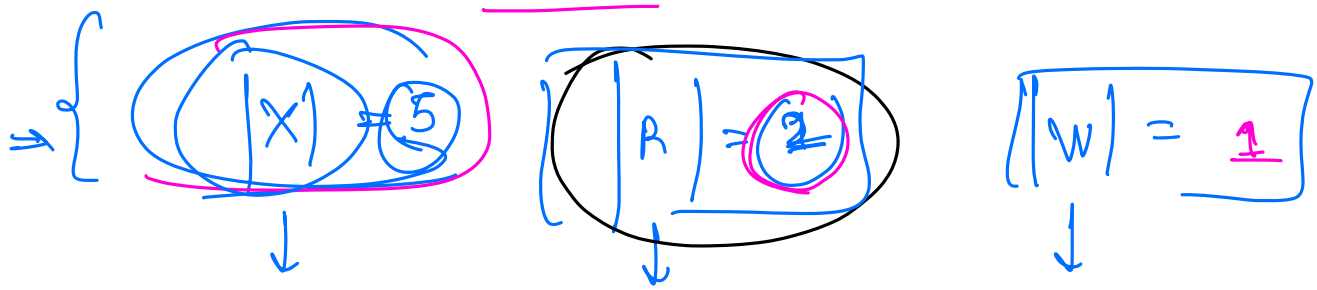
(2) Available

write to master and move on.

In M-M

↳ tunable consistency

↳ how much consistent / how much available




how many masters for every key

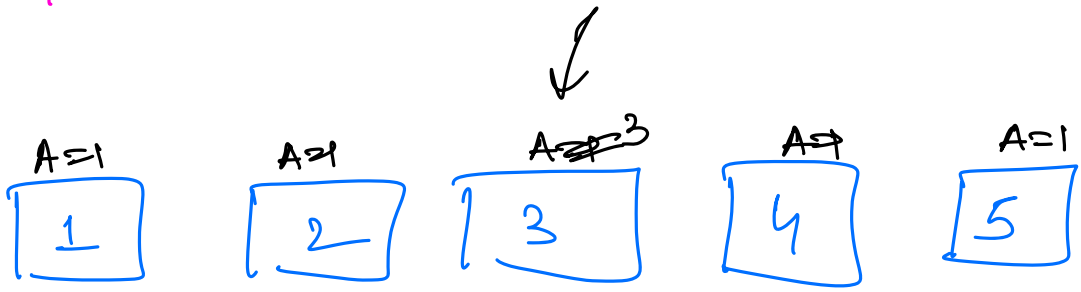
how many machines to read from before returning to client

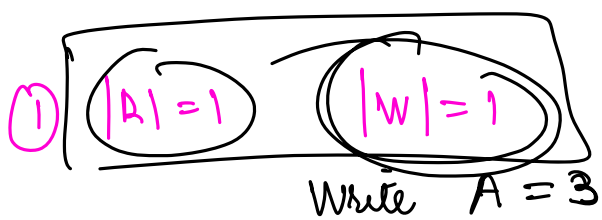
if any |R| of masters have same value return

how many machines to write to before returning success to client

later async I will sync other

K = |K| = 5



lower value of W ⇒ fast Writes

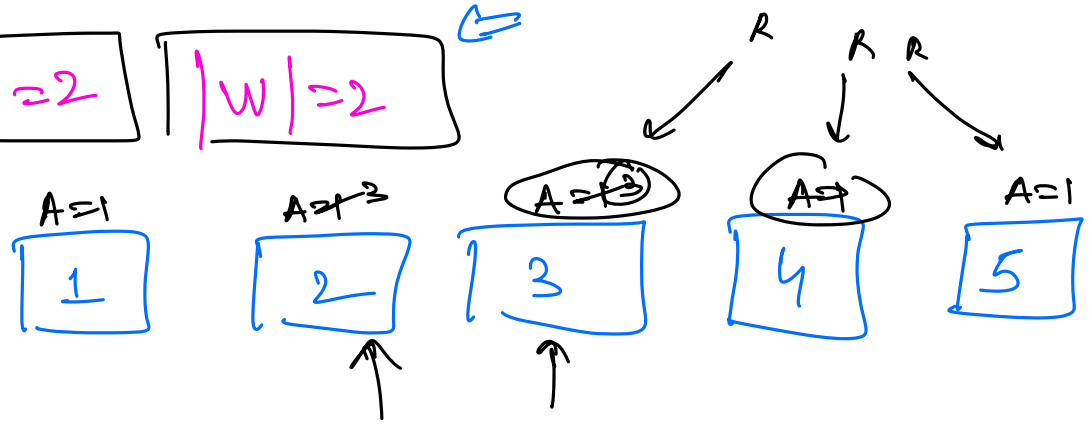
Read A ⟶ 1

Available → ✓

Consistent → ✗

less value of R ⇒ fast reads.

② |R| = 2 |W| = 2




Write A = 3 ⇒ higher latency than |W| = 1

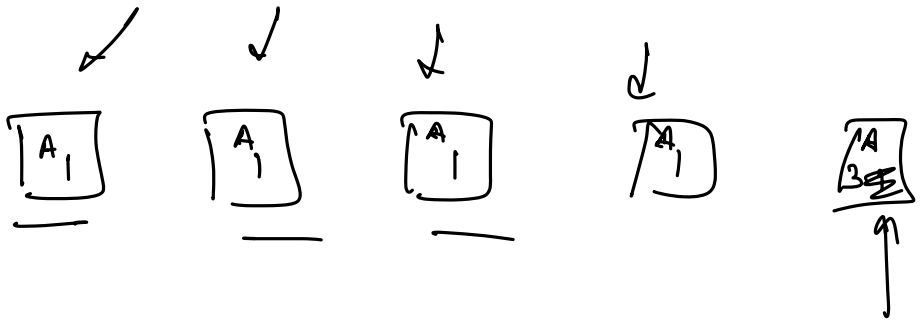
Read A ⇒ 3

Consistent but higher latency

1

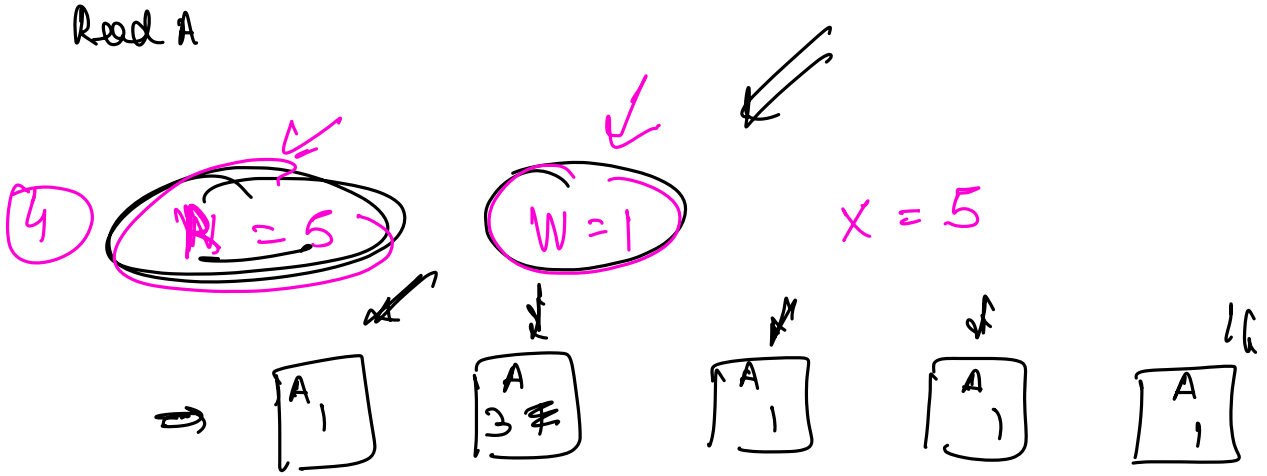
→ available but higher latency

③ |A| = 4 |W| = 1 |X = 5




Write A ⇒ 3

Read A



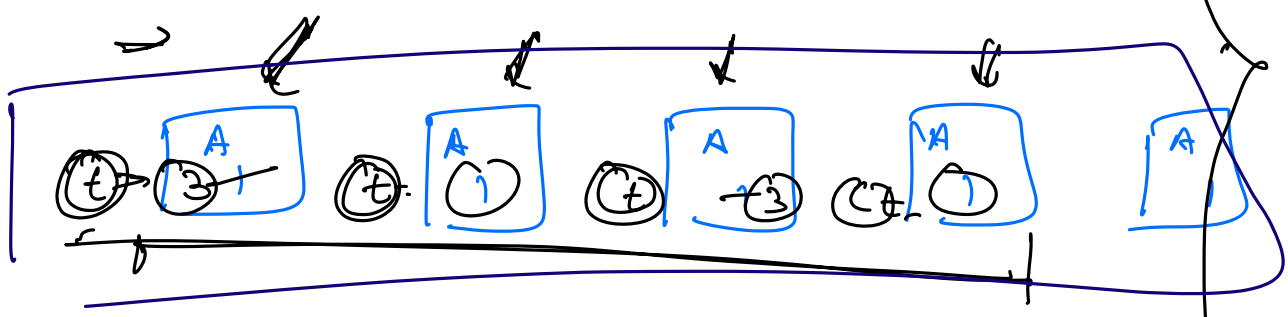

Write A = 3

Read A

⇒ Consistent ⇒ Available ⇒ ✗

Consistent ⇒ |R| + |W| > X

|W| = 2 |R| = 4 |X| = 5



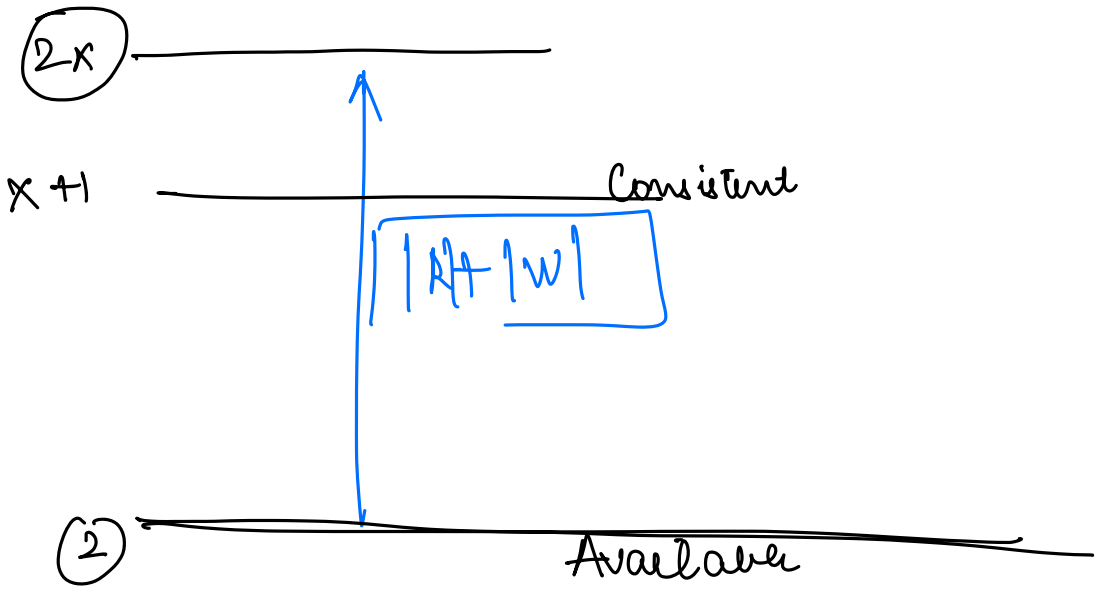

Write A = 3

Read A

Consistent ⇒ $|R| + |W| > |X|$

- If fast write ⇒ less W, big R
- If fast reads ⇒ less R, big W.

Available $|W| = 1$ $|R| = 1$




Cassandra ⇒ Fast Writes

⇒ Consistent

⇒ Slow Reads

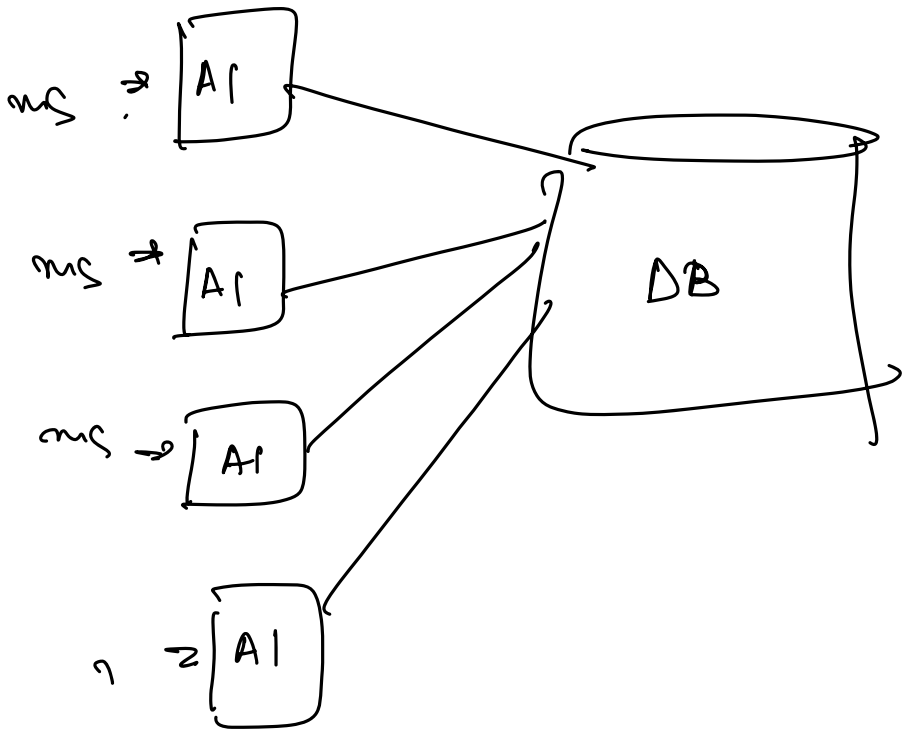
$|W| = 5$ $|R| \geq 1$

⇒ Consistent

⇒ Writes slow

⇒ Reads v. v. fast.

"NO GLOBAL CLOCK"

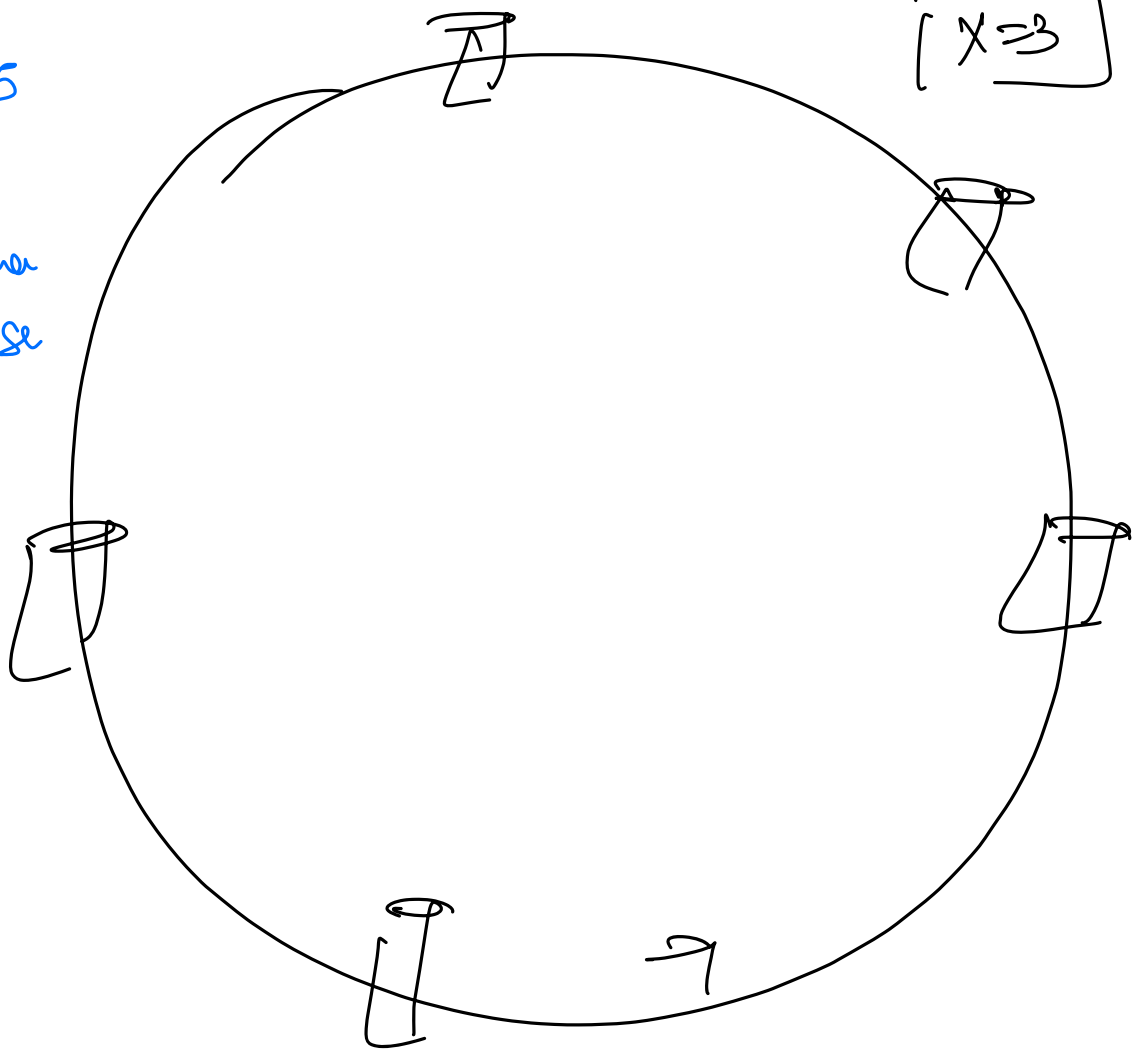


$X = 5$

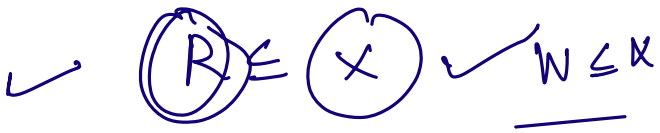
2

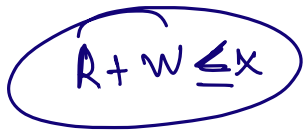
6 she

10 se

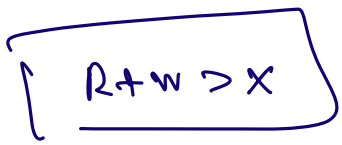


$X = 3$

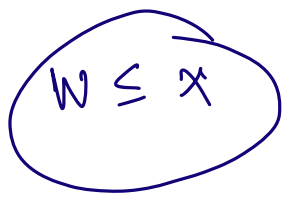


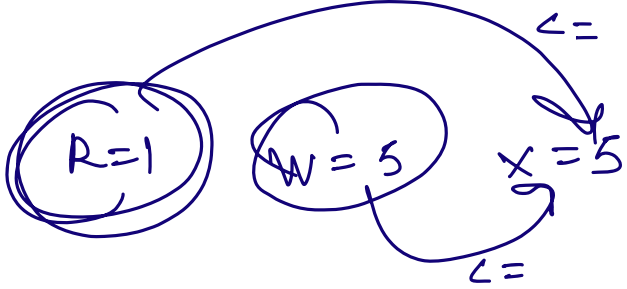




Avail ~
Consistn X




Consistn ~
Avail X




Consistent

W=1 R=5 x=5

Consis