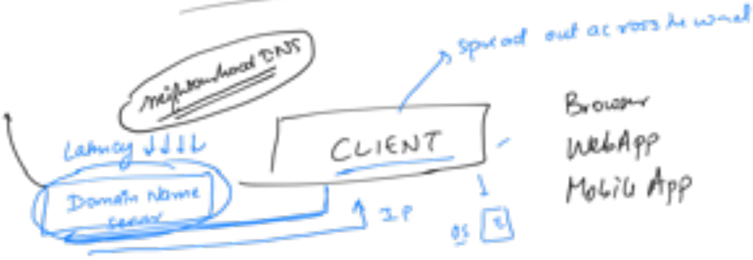
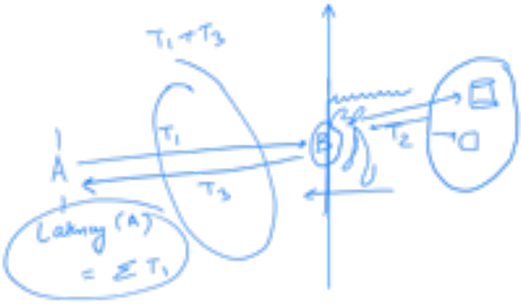
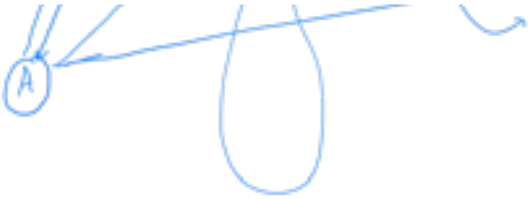
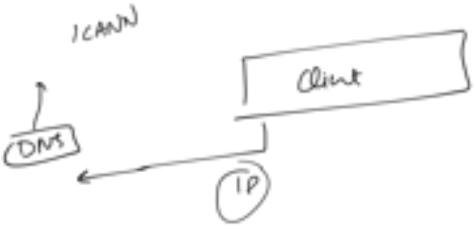
-

High Level Design

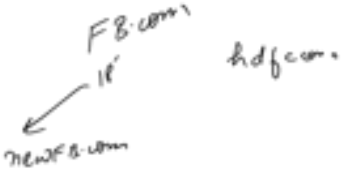


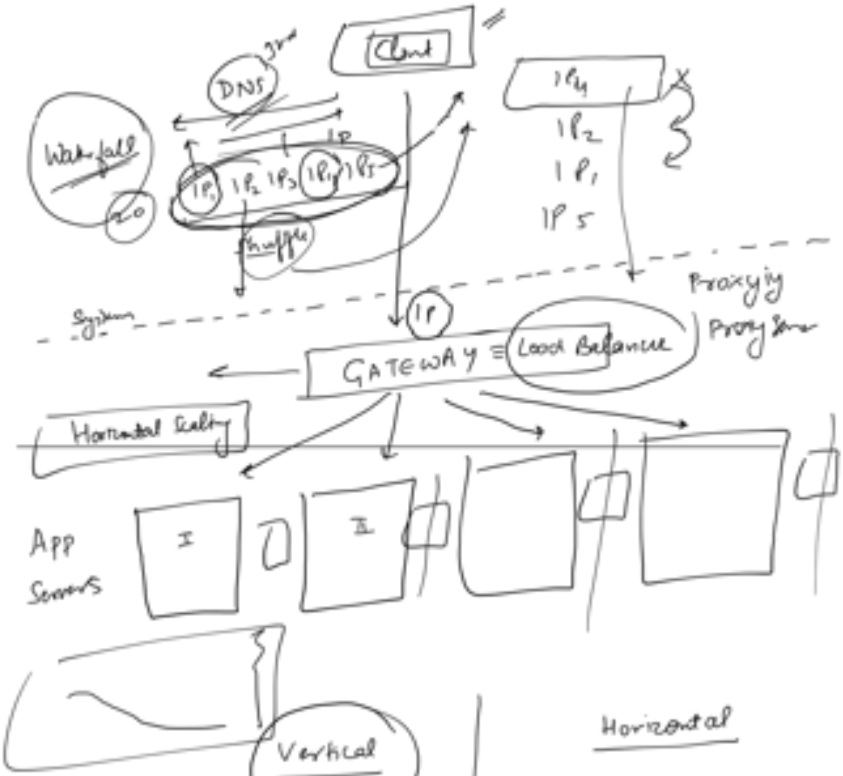






ISPs
Google
FB
US Army





Vertical

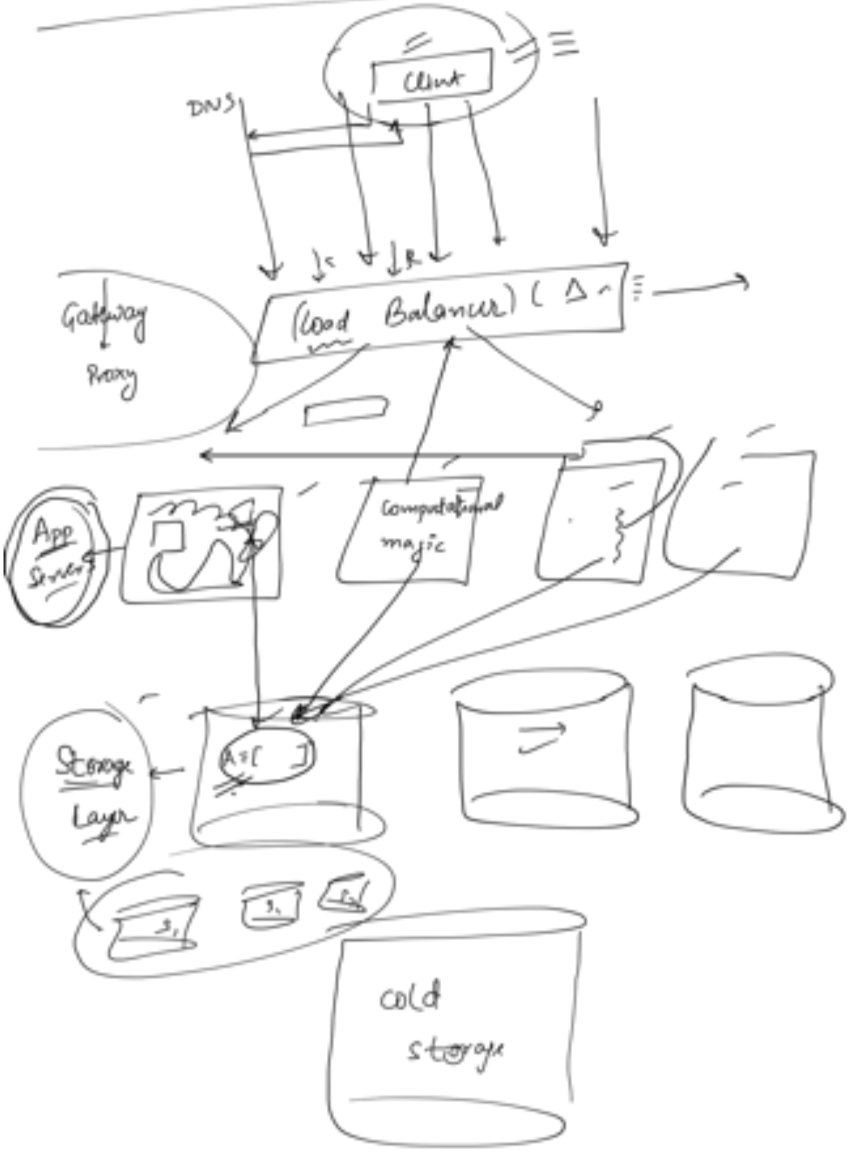
→ much easier to get some guarantee

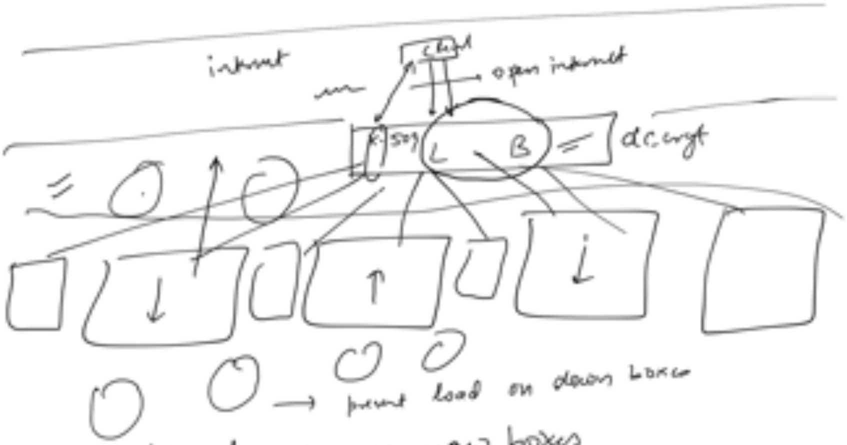
Horizontal

→ flexibility with chang'g load

→ Cost effective

CAP theory





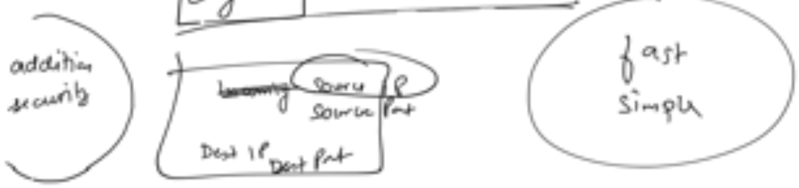
→ prevent load on down boxes

intranet → rgistr new boxes

→ balance out

— X.509 —

Layer 4 ≡ Transport Layer



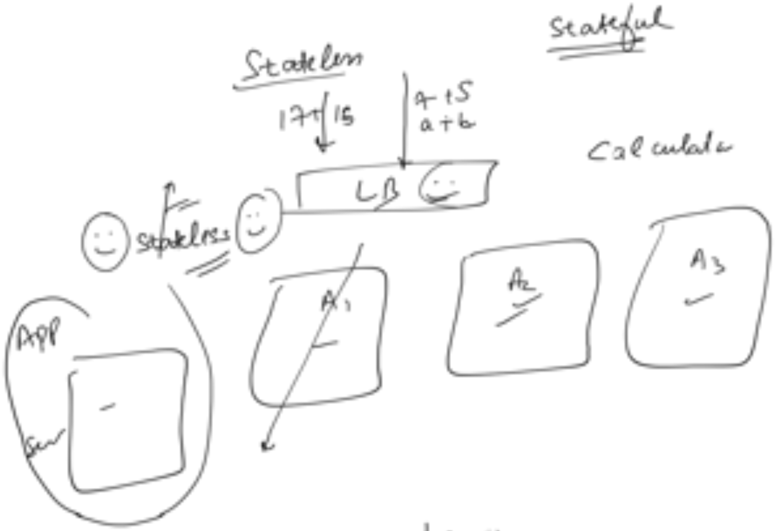
Layer 7 LB ≡ Aplication

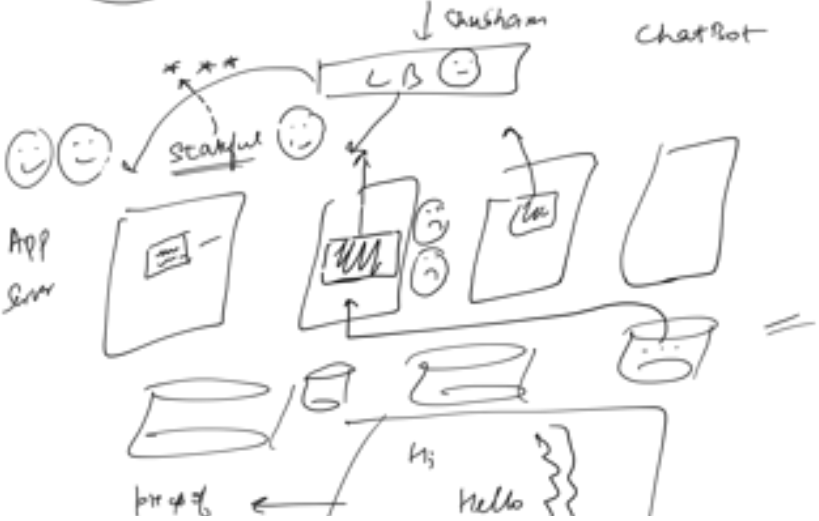
user id ≡ 707

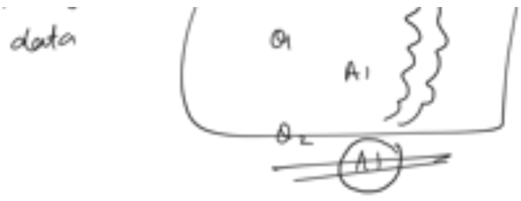
uid = 107

much better flexibility

Slowerd



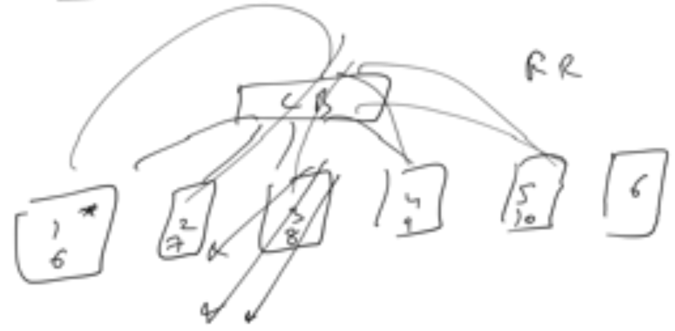




Load Balancing

Stateless

→ Round Robin



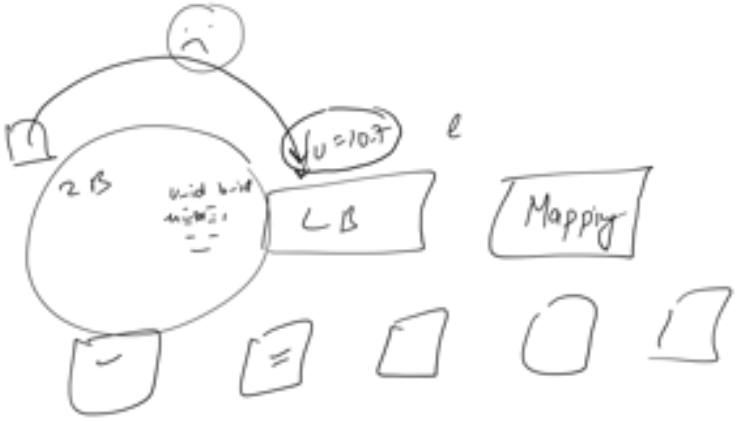
→

WRR

→ (Least Connection first)

→ WRR ARR

Load Balancing in Statefull



Sol ① Maintaing Map BAD

map-size = order of key-size

Load Balancing func will slow down

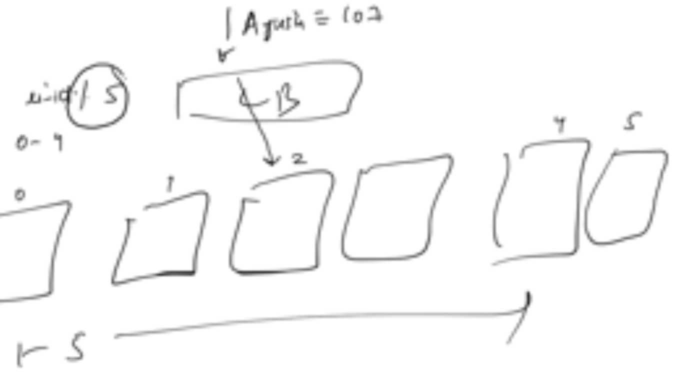
Sol ② Hash

BAD!!!

10 boxes ≡ 10 App Server

0 – 9

uid % 10 = □



→ gaurant of stickiness using hashing

→ LB ☺

Downside

On server inc/dec you will have to move state of all boxes internally

% 6

1 % 5	1	% 6	1
2	2		2
3	3		3
4	4		4
5	5		5
6	1		6
7	2		1
8	3		2
9	4		3

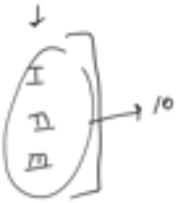
③ Ranges

in LB

1 - 1000

1001 - 5000

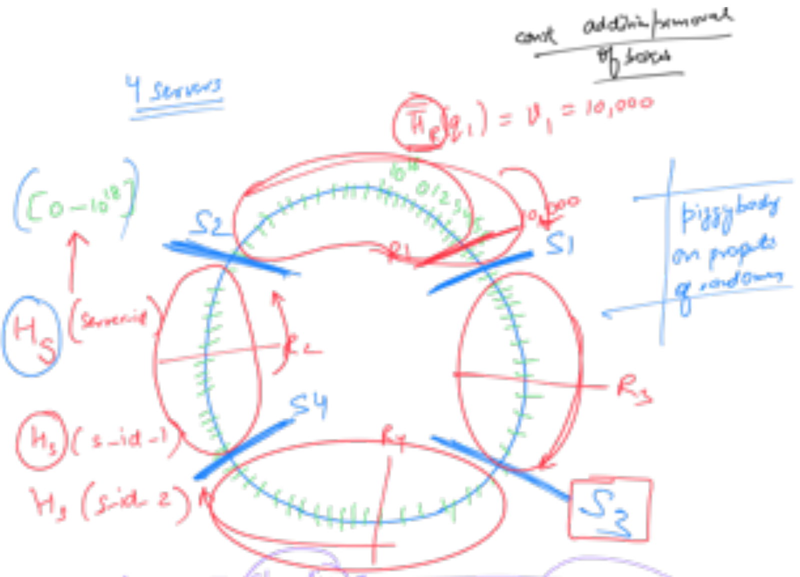
5001 - 10000



Bottleneck: Add / Removal of boxes is inefficient ☹ ☹

④ Consistent Hashing

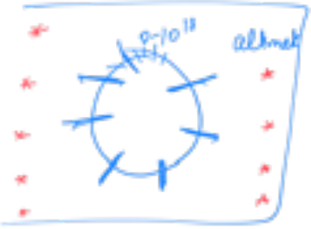
LB for Stateful Application



database ≡ Sharding key

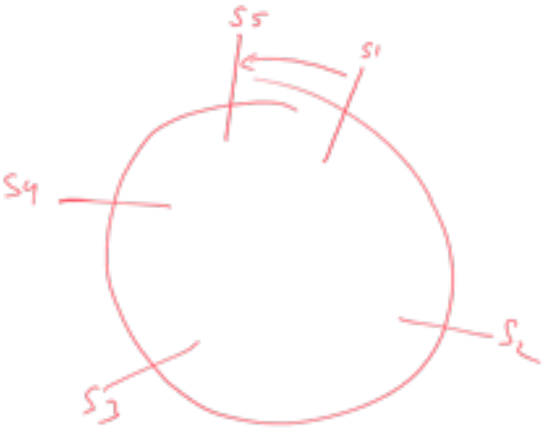
req-id = user id

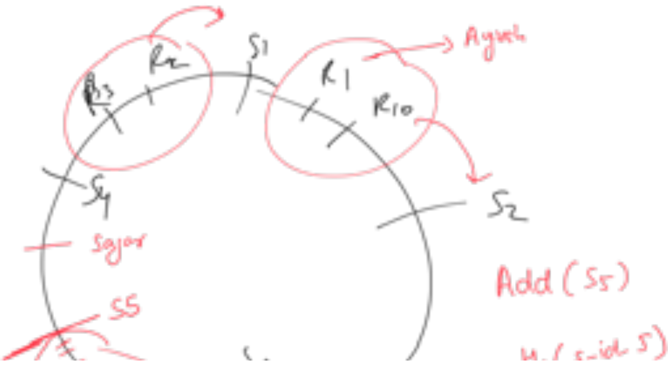
req-id = location id



H_R (req-id)

[0 - 10^{18}]



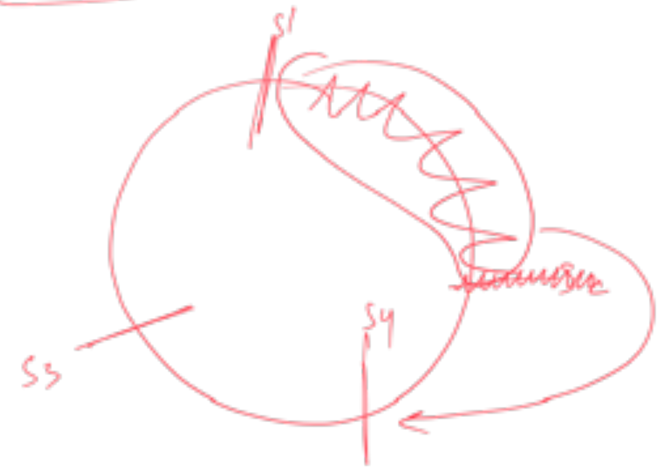


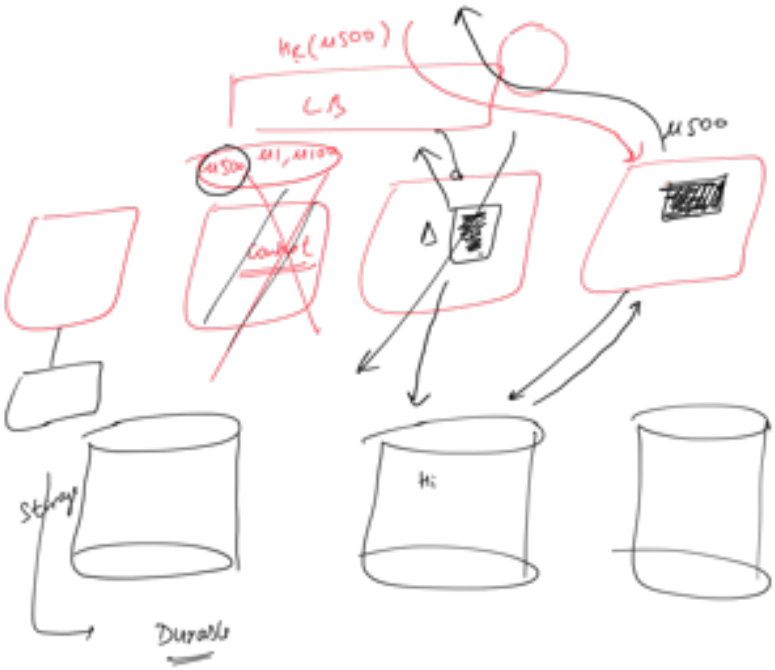
✓ (I) Amount of state transfer that happens has been reduced a lot

by a factor of # servers

✓ (II) down time will also be low.....

✓ (III) you don't have downtime for Eng com. small set of users get impacted





→ we are not losing the ultimate source of truth

Ⓐ cold starts ≡ requests latency inc for some time

Hi

nodes

O1

A1

O2

A2

Online Learning Algo's model =

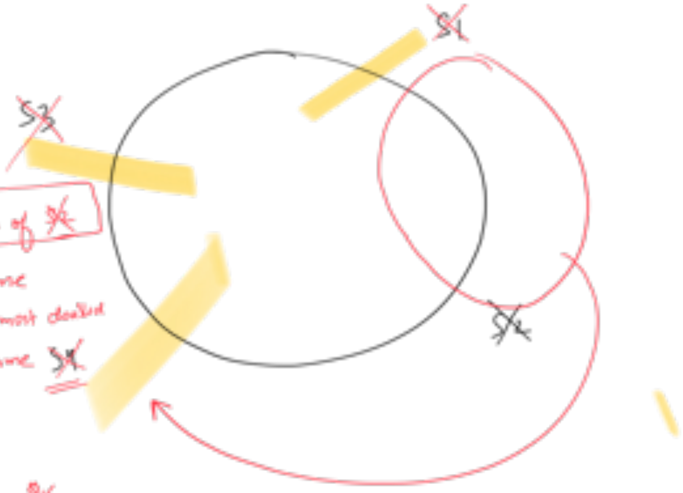
Ⓑ pre-emptive copying

Ⓘ ← Storage layer → new App Server

Ⓘ Ⓘ ← M S lave

Solved problem

impact of server addition or removal is minimized



Implications of ~~S4~~

✓ S1 → same

S4 → almost doubled

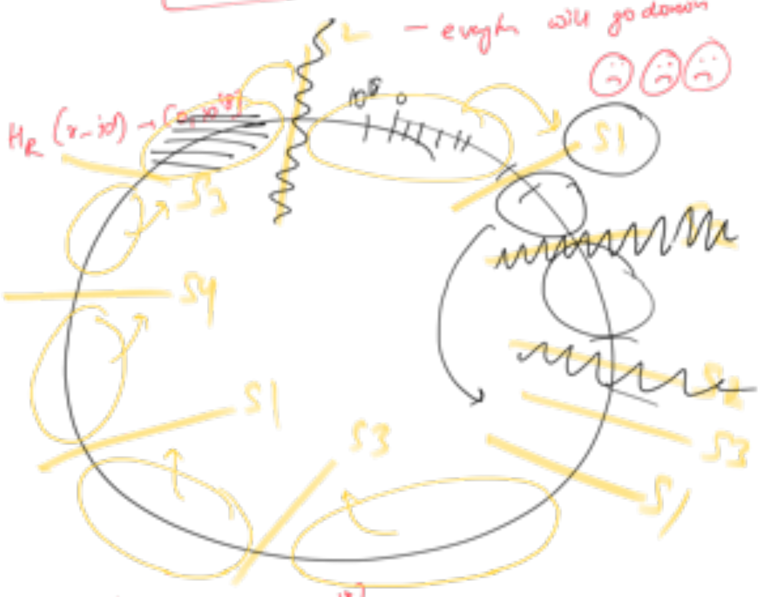
✓ S3 → same ~~S4~~

~~S4~~

S1 ≡ same

S3 ≡ 2X

Cascadiy Failurs



H_S^1 (s-id) → [0,10^18]

H_S^2 (s-id)

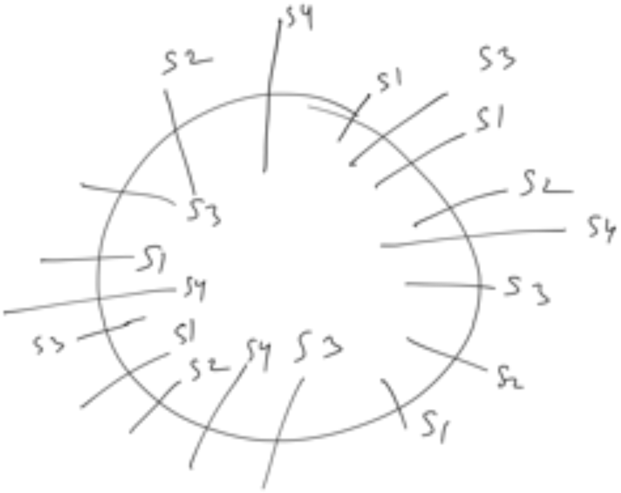
H_S^3 (s-id)

| servers | X | Hs func |

S1 ↑

S3 ↑↑

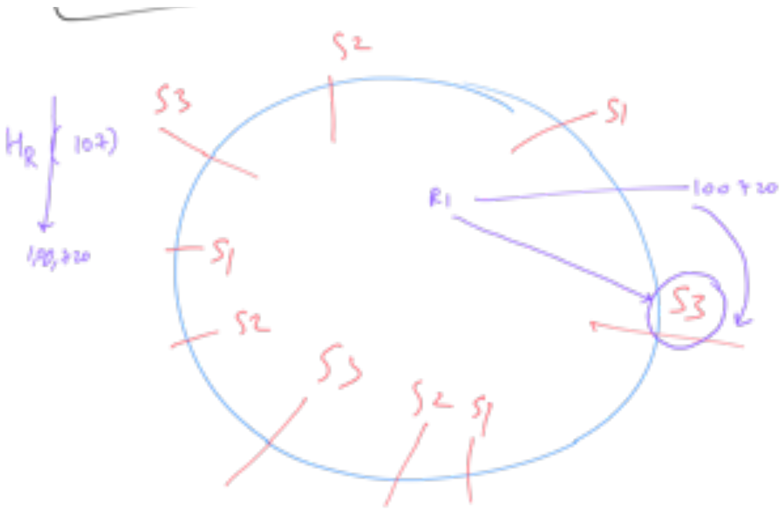
S4 ↔



solved

Cascading Failures

(Now, the load will get divided almost equally.



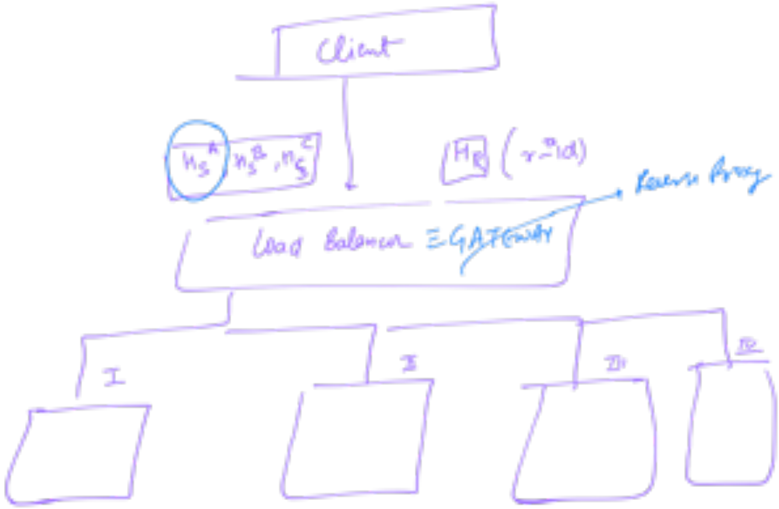
$|H_S|$ ↑↑ ≡ Randomness ↑↑

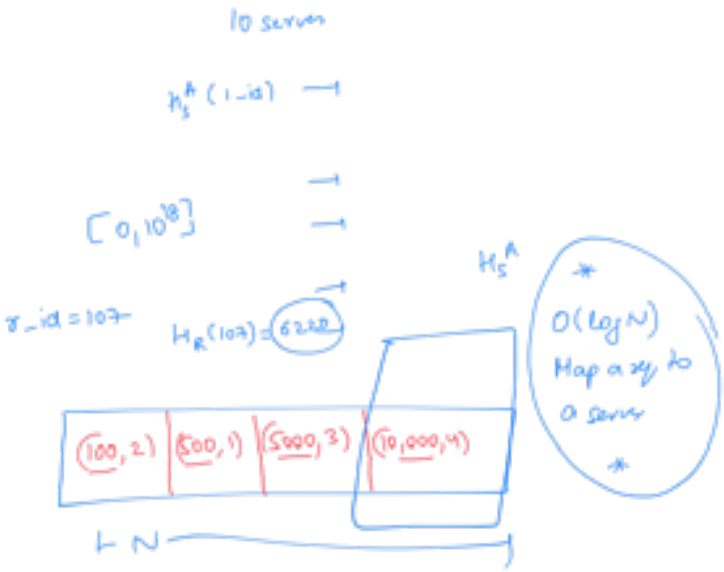
↓

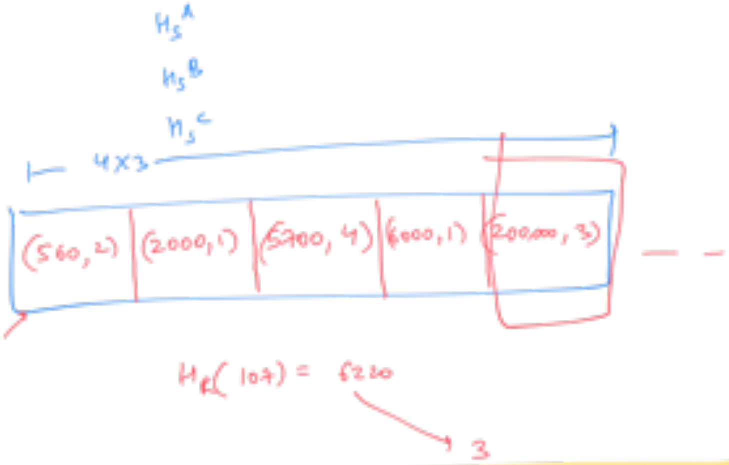
Equitable Distribution ↑↑

$|H_S|$ ↑↑↑ ≡ Time for Resolution ↑↑↑

ASG



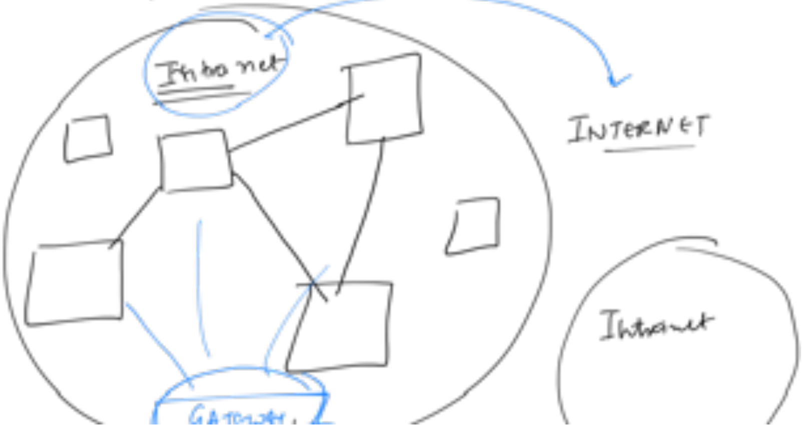


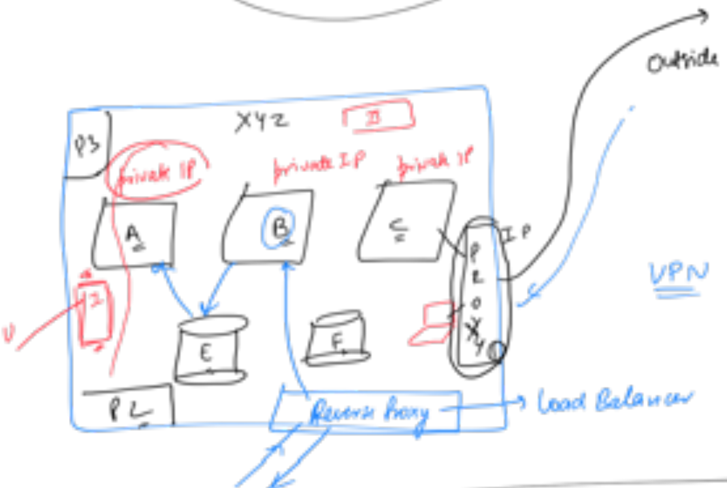


High Level Design

OSI Layers

Internet ≡ network of networks

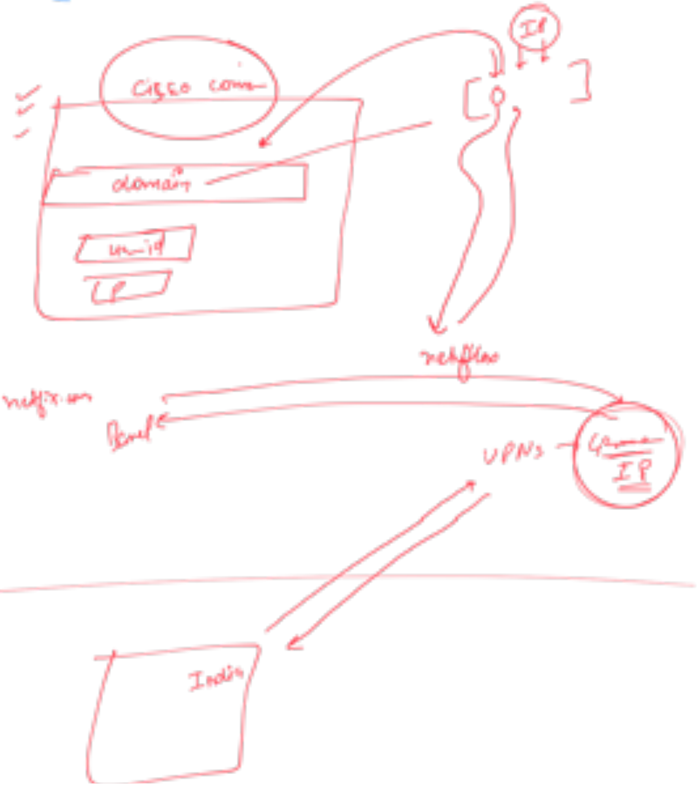


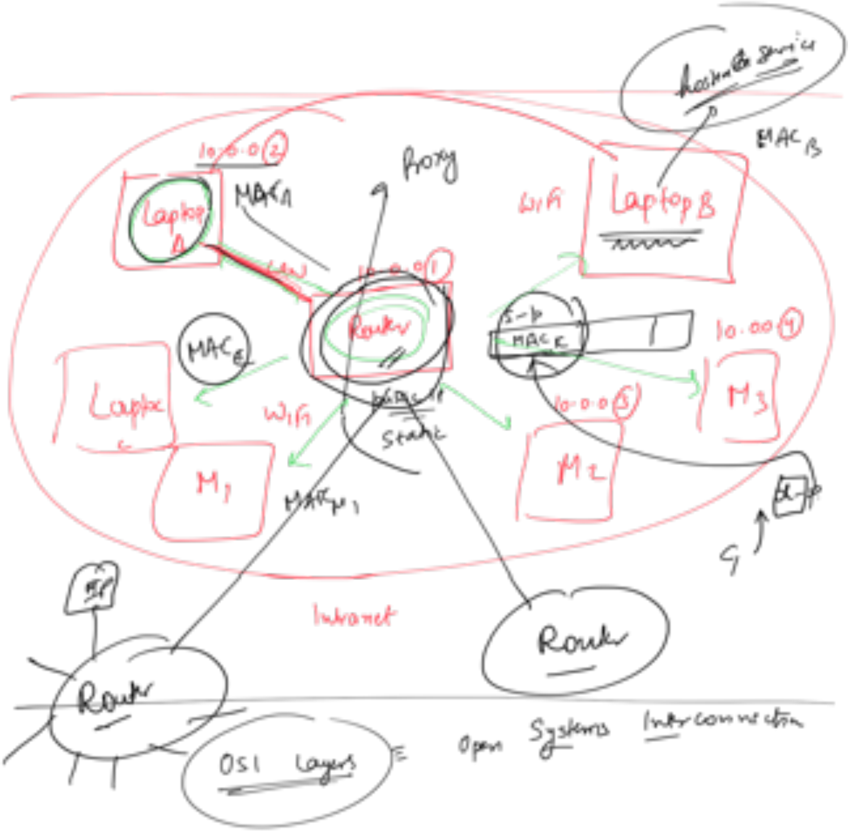


Gateway

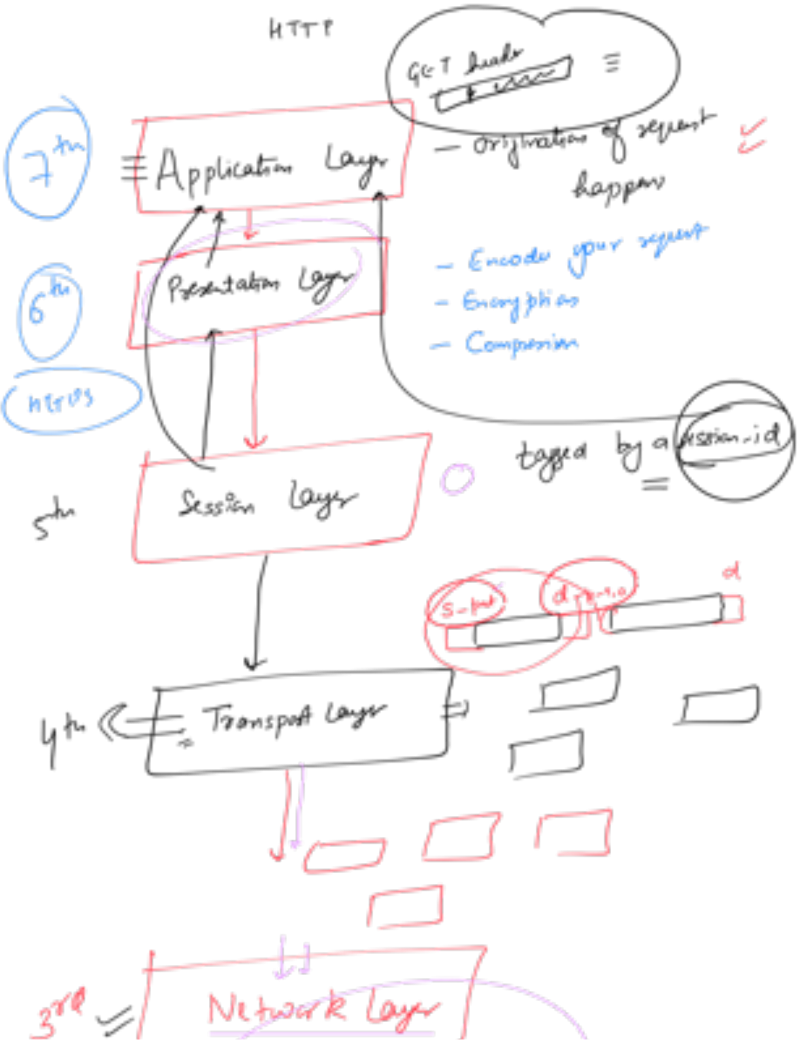
Proxy ≡ Gateway to the outside world

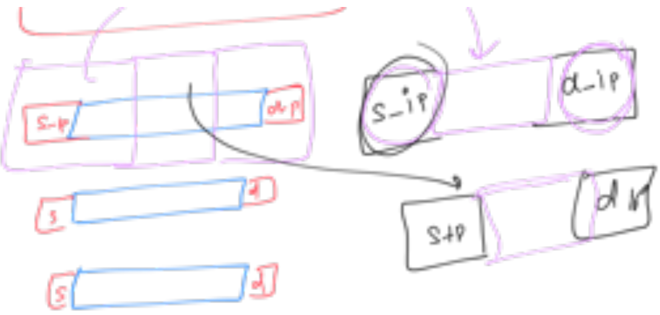
Reverse Proxy ≡ Gateway to the inside world



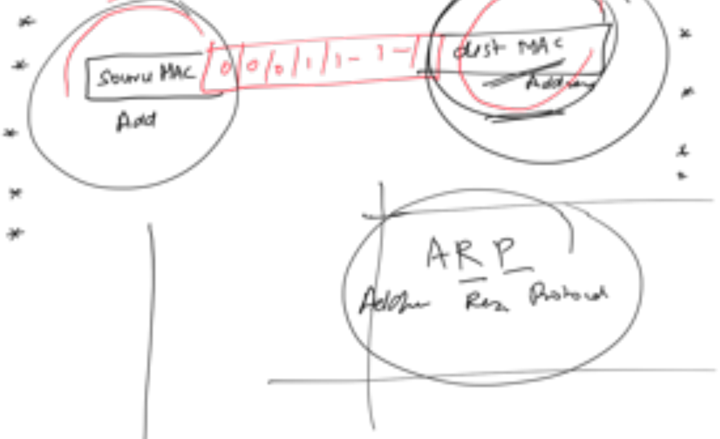


OSI Layers ≡ Open Systems Interconnection

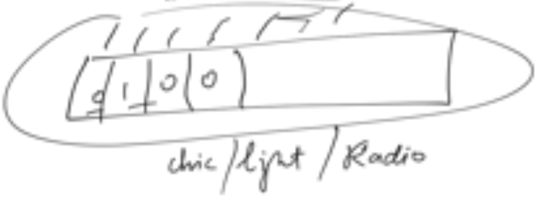


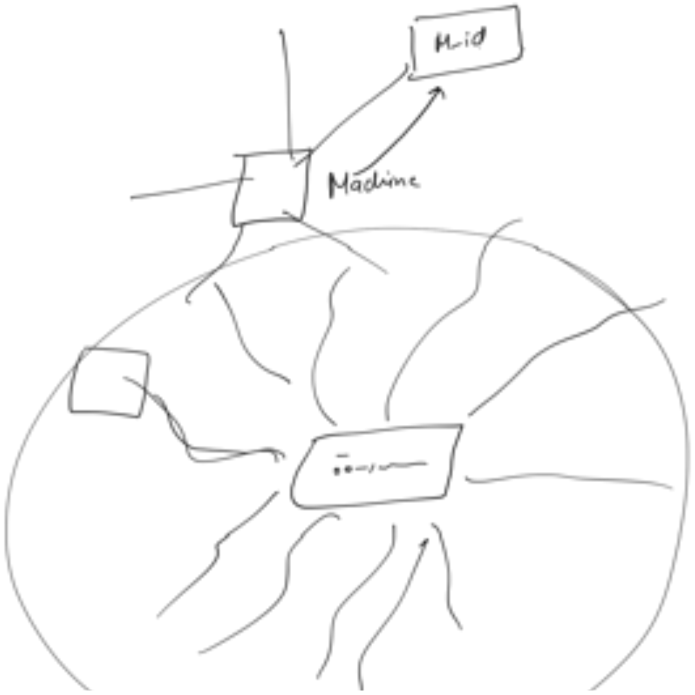


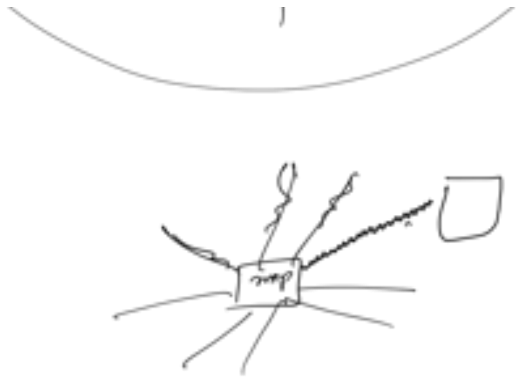
2nd Data Link Layer

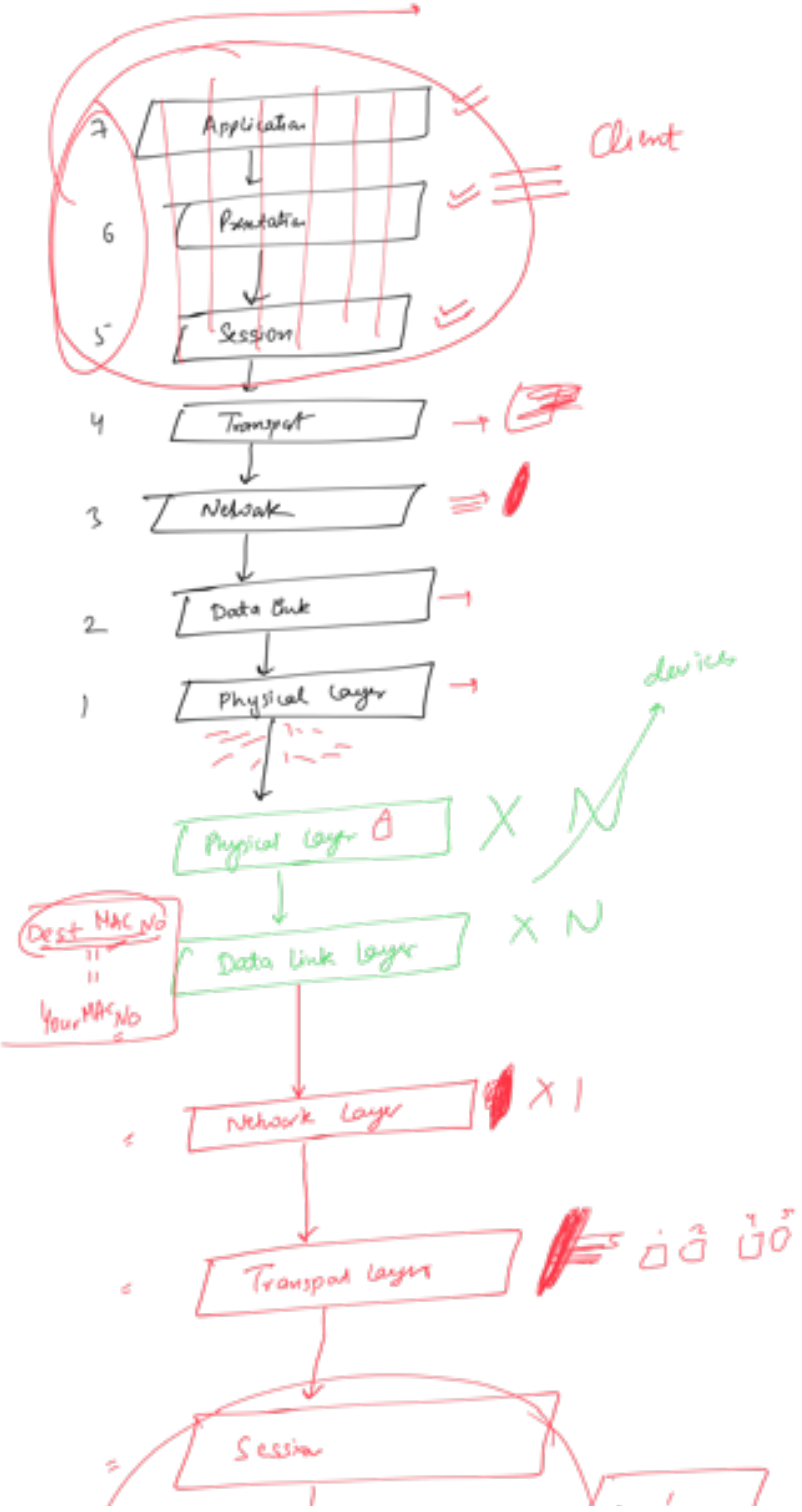


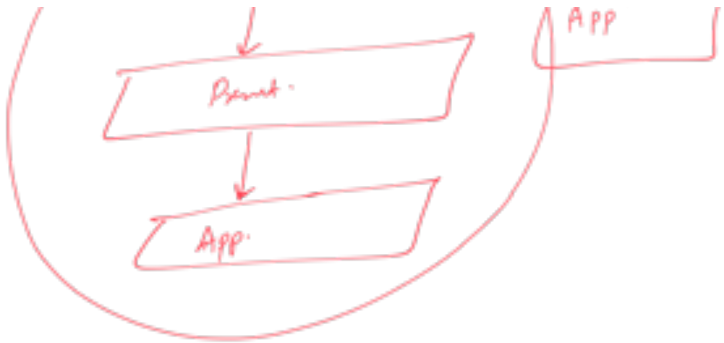
1st Physical Layer

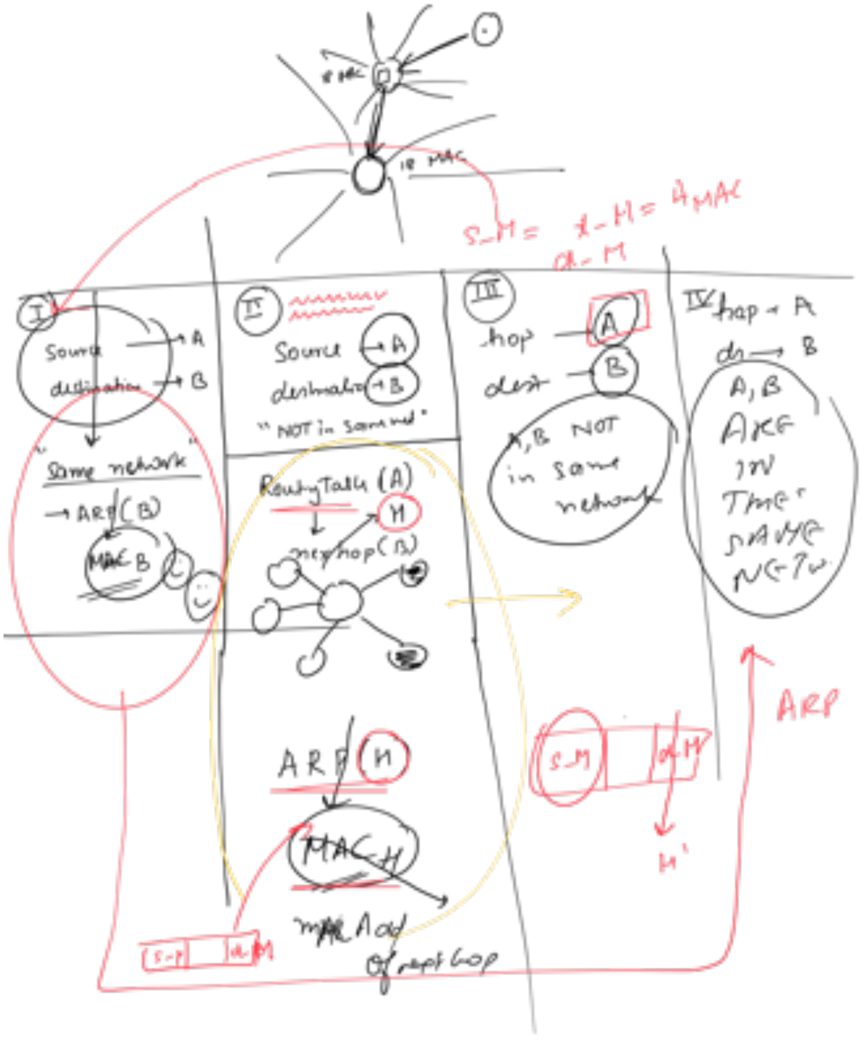







7
Application
6
Presentation
5
Session
4
Transport
3
Network
2
Data link
1
Physical layer
Client
devices
Physical layer
X N
Data Link layer
X N
Dest MAC No
=
Your MAC No
Network layer
X 1
Transport layer
Session


App
Present.
App.


IP MAC
IP MAC
S_M = ~~d~~_M = A_MAC
d_M
I
Source → A
destination → B
"Same network"
→ ARP (B)
MAC B
II
Source → A
destination → B
" NOT in Same net "
RoutingTable (A)
M
nexthop (B)
ARP (M)
MAC_M
~~MAC~~ Add of next hop
S_P
d_M
III
hop → A
dest → B
A,B NOT in Same network
S_M
d_M
M'
IV hop → A
d → B
A, B
ARE
IN
THE
SAME
NETW.
ARP

CAP THEOREM

[only guarantee $\frac{2}{3}$ properties simultaneously

C ≡ Consistency

A ≡ Availability

P ≡ Partition Tolerance

I Consistency → Immediate Consistency

Any read that you do, must return the value after the latest write

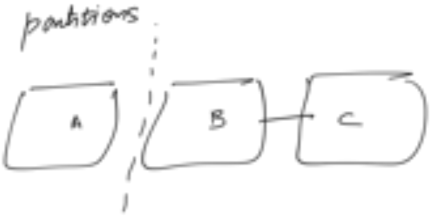
II Availability

Every and any request that goes to a non-dead server, must be responded to

III Partition Tolerance

The system as a whole is allowed to have partitions

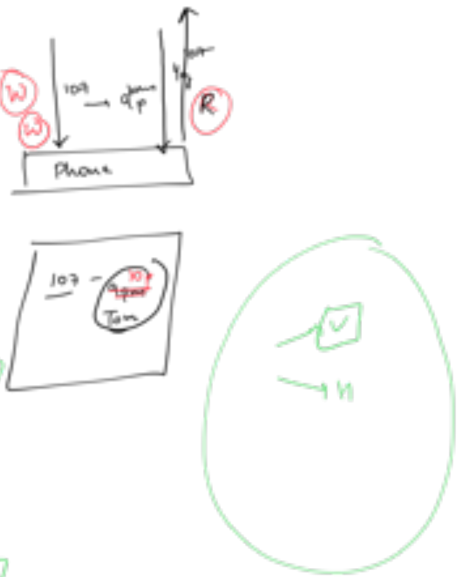
OR



You have accepted the fact that network partitions can happen at any time

and the system as a whole will NOT stop functioning

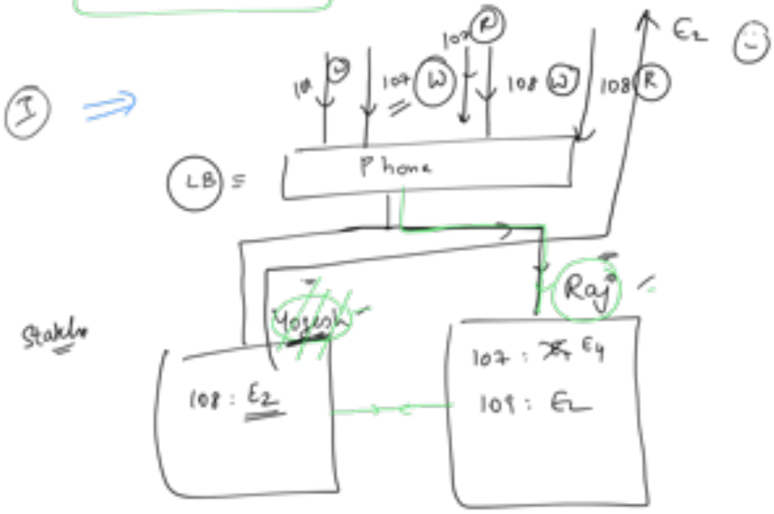
Event Reminder Service



Does NOT Require Partition Tolerance

Consistent

Available

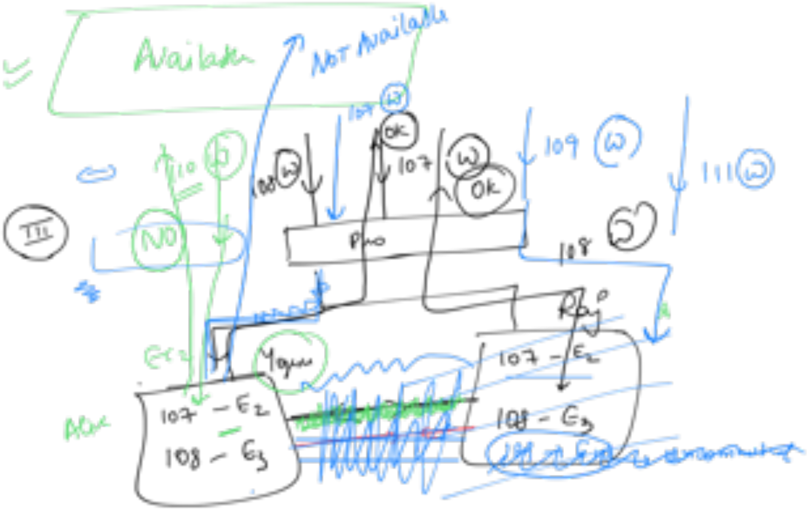


Partition Tolerance

NOT CONSISTENT

2/3

NOT CONSISTENT



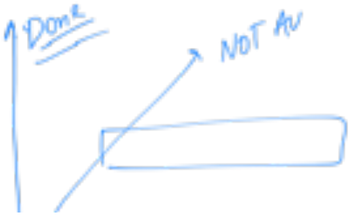
CONSISTENT

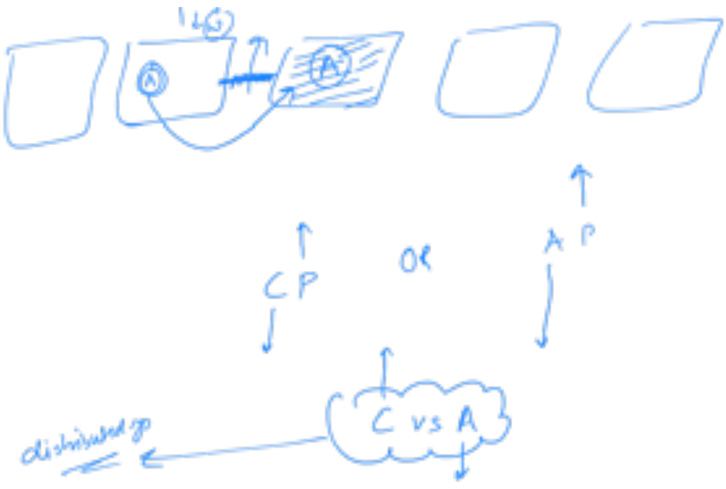
PARTITION TOLERANT

NOT AVAILABLE

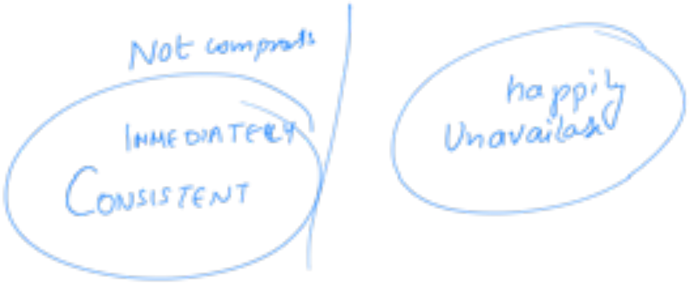
CP OR AP

— CA — if you are doing vertical scaling





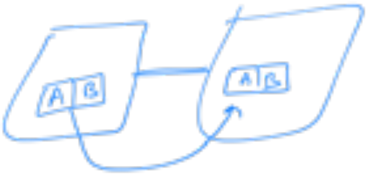
CA
X not required



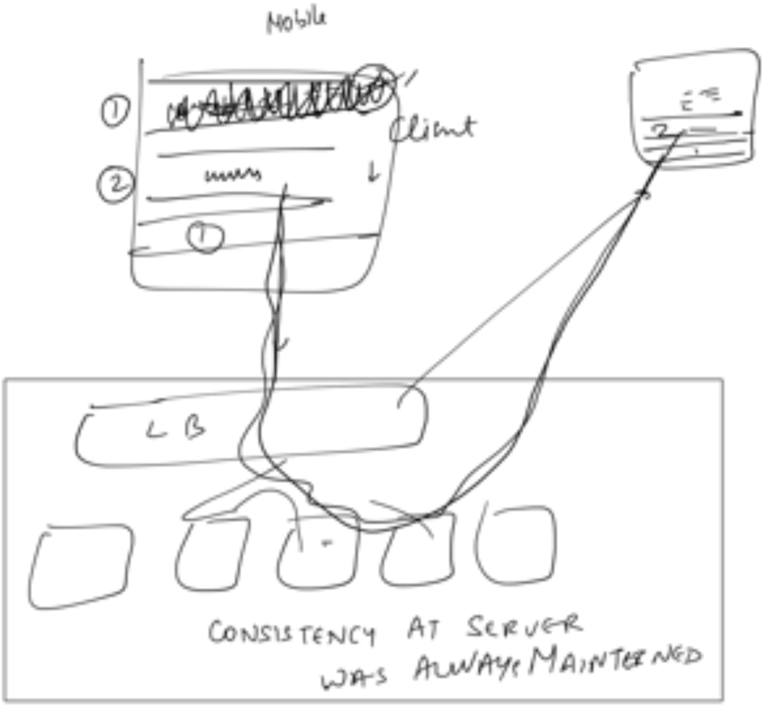
Eventually Consistent

120mins

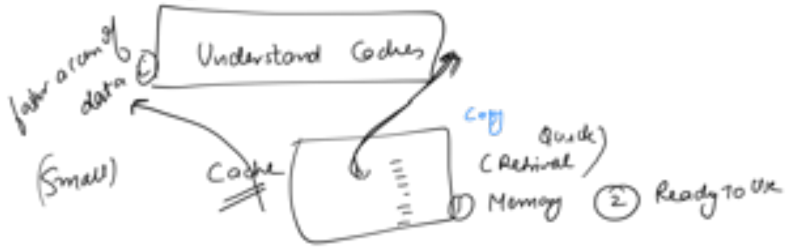
60mins

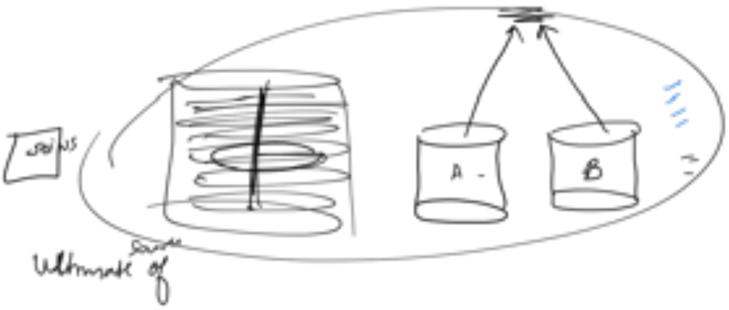


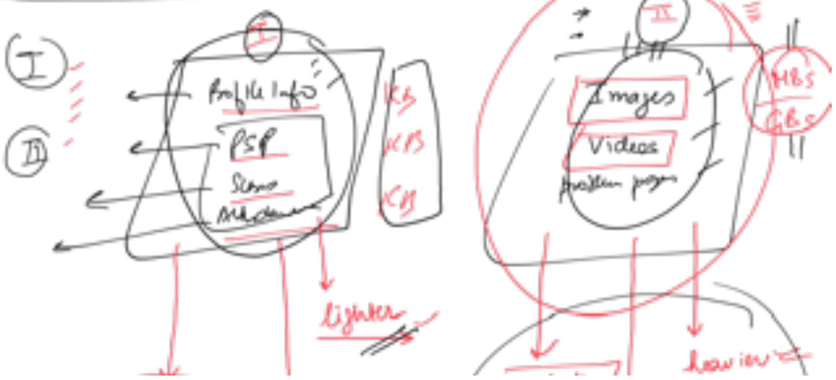
Never confuse latency with consistency

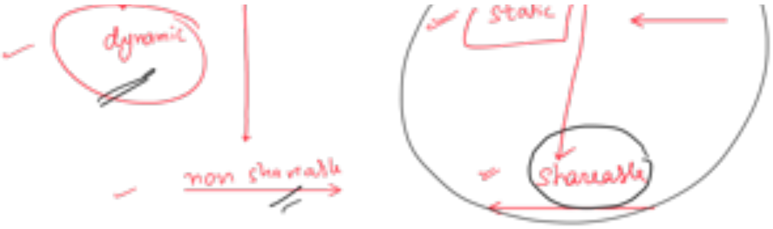

Mobile
1
2
1
Client
LB
CONSISTENCY AT SERVER WAS ALWAYS MAINTAINED

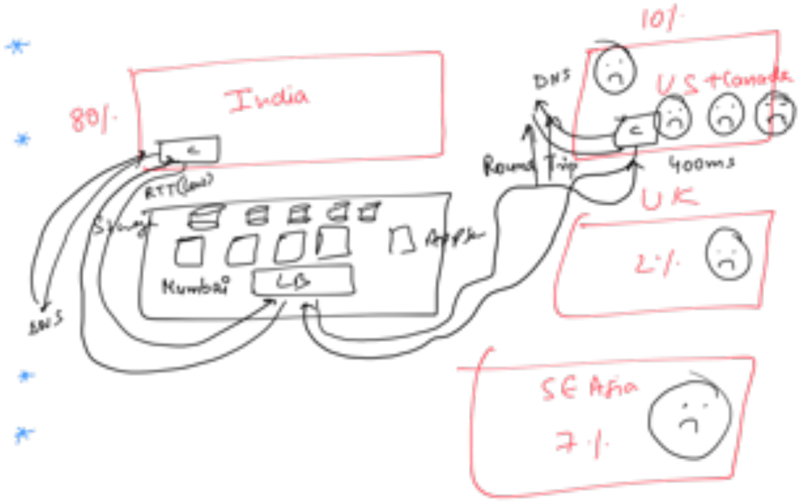
CACHING


Understand Caches
1
faster access of data
(Small)
Cache
Copy
Quick
(Retrival
1 Memory
2 Ready to Use


A
B
Ultimate source of

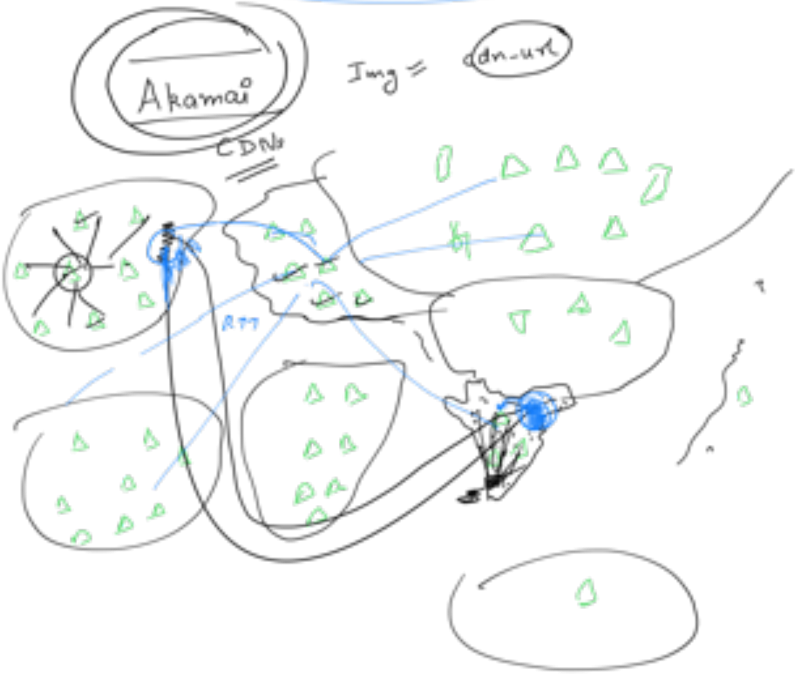

I
II
I
Profile Info
PSP
Scores
Attendance
KB
KBS
KB
II
Images
Videos
Problem pages
MBs
GBs
lighter
heavier

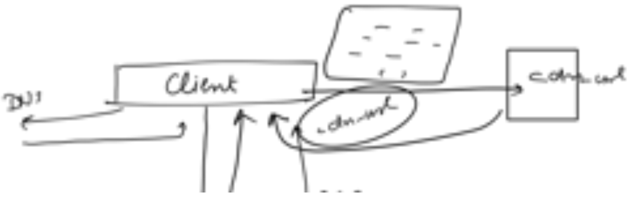


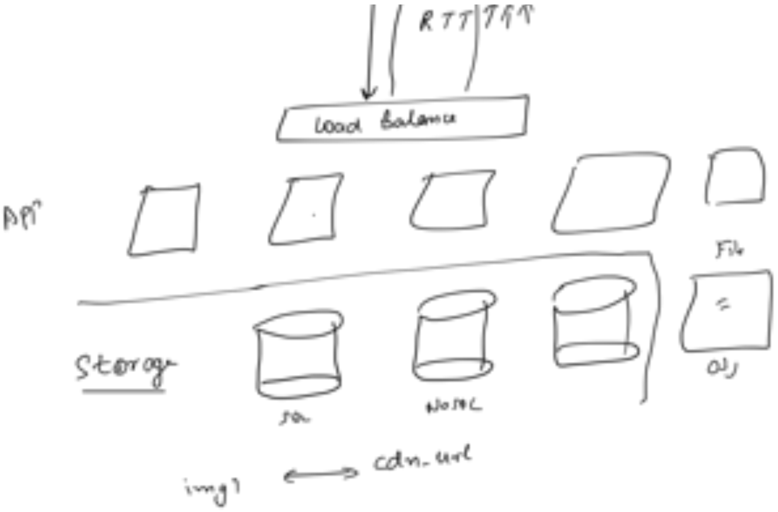


Content Delivery Network

CDN

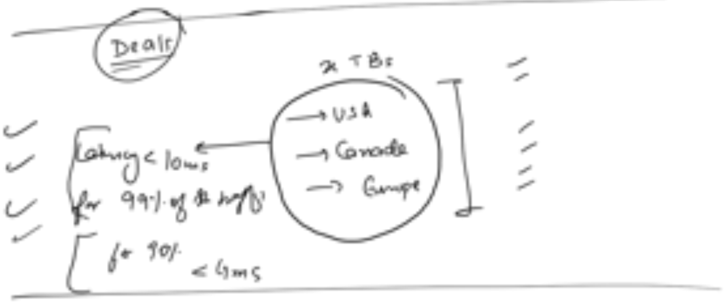






→ network is NOT getting clocked

→ latency solver

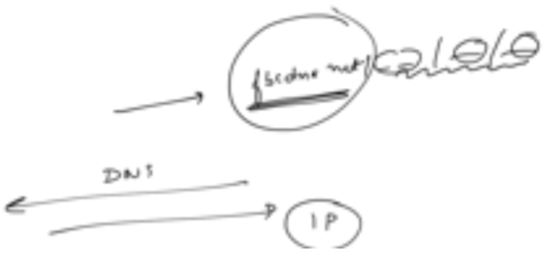


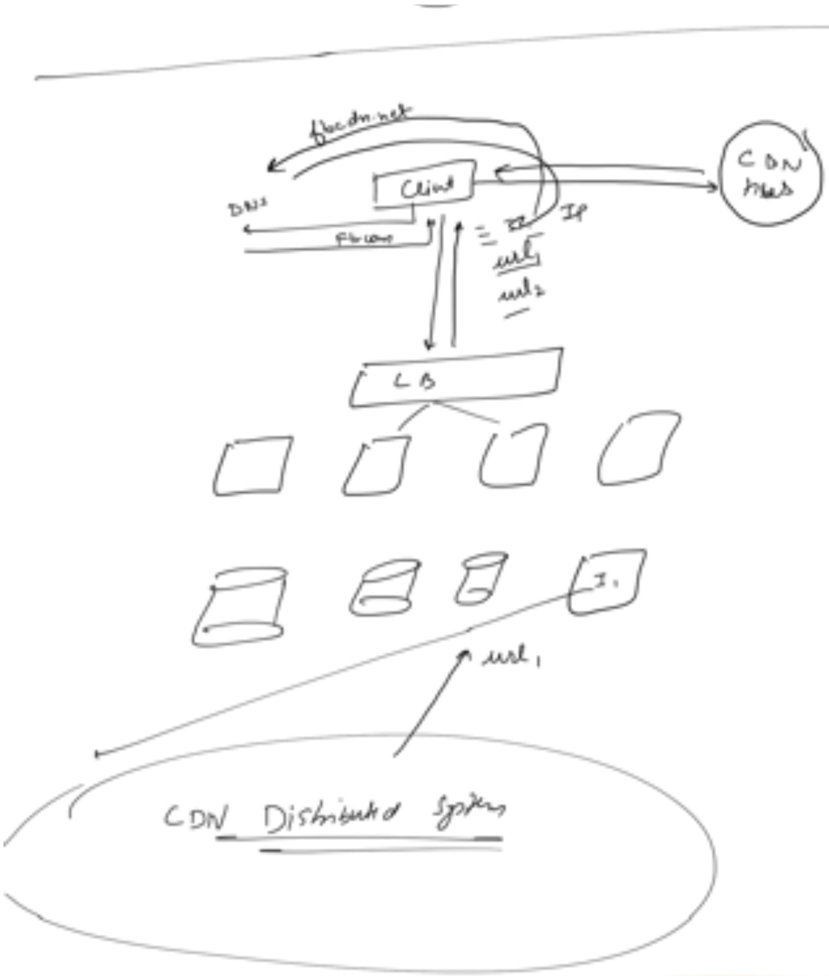
Company → CDN

CDN hub → CDN subhub

user → CDN

cdn





Cache Invalidation

Ⓐ → Time to live (TTL)

= url should automatically become void after this time

Ⓑ → Start sending a new URL

Ⓒ → Versioning

Privacy in CDNs

≠ ⇐ ① CDN urls are by design not guessable

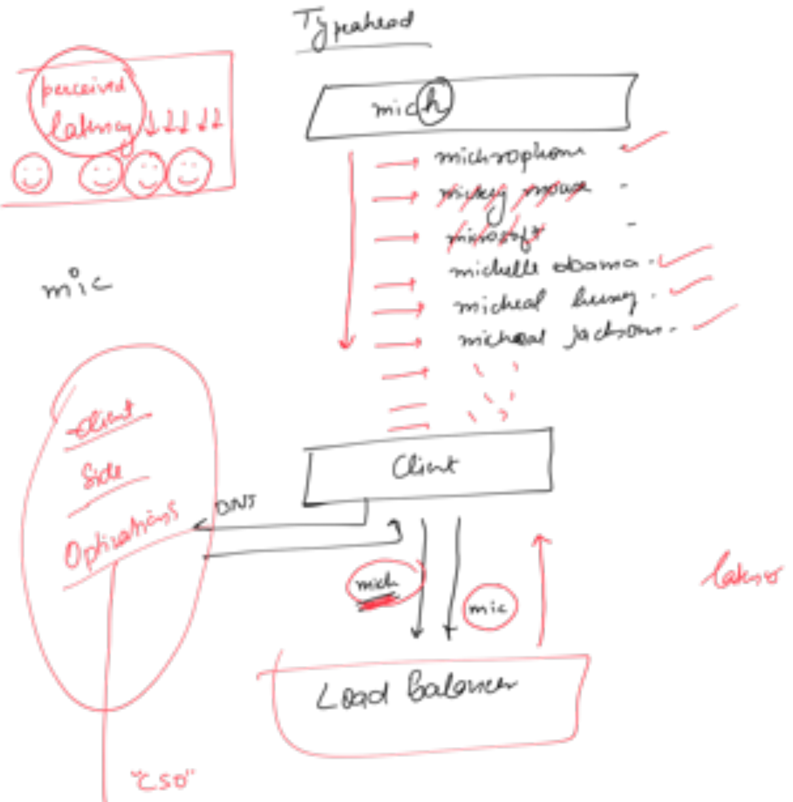
② encrypted format

③ Authentication

② Client Side Caching

Browser
Mobile App
Desktop App

You can choose to store some amount of "important" data on the client's side



Reduce the "perceived" latency

Browser Optimization

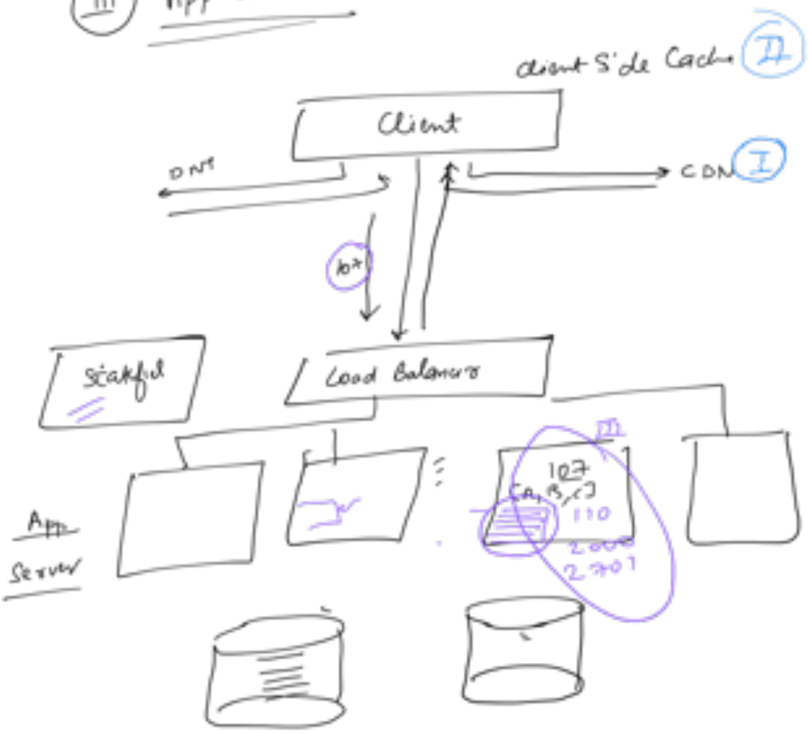
automatically identify a few static resources for you and cache them

Hard Reload / Forced Reload

'ctrl + shift + R'

☺ 😐

III App Server Cache

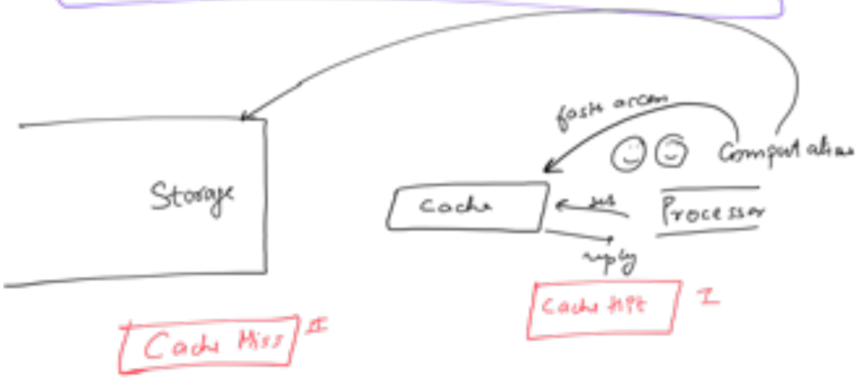


Cache Replacement

(A) LRU → Least Recently Used

(B) MFU

(C)



Access Time for Cache = x

" " DB = 100x

k caches

without cache

k × Avg DB Access Time

Time = [k × Avg
+
k × Avg Network Access Time]

80-20

With Cache

[70% C Hit Rate

30% Cache Miss Rate]

$$\text{Time} = \left[\frac{70}{100} \times k\right] \times \text{Cache Access Time}$$

$$+$$

$$\left[\frac{30}{100} \times k\right] \times \text{Cache Access Time}$$

$$+$$

$$0.3k \times \text{DB Access Time}$$

$$+$$

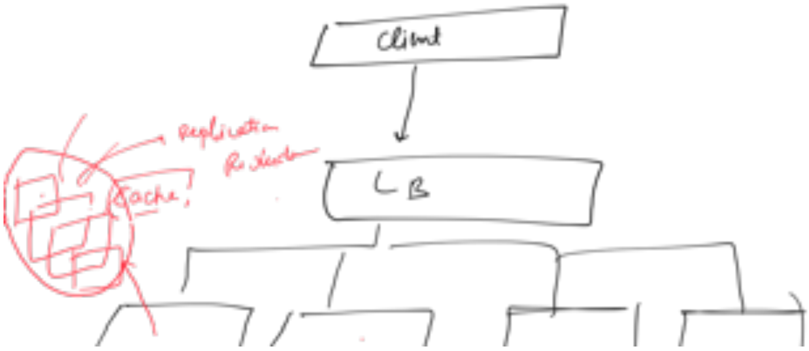
$$0.3k \ \text{Ne A. Time}$$

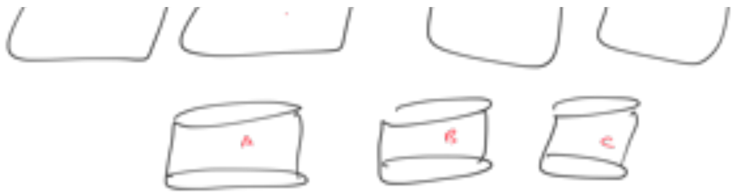
Ⓐ How deranged are your incoming requests

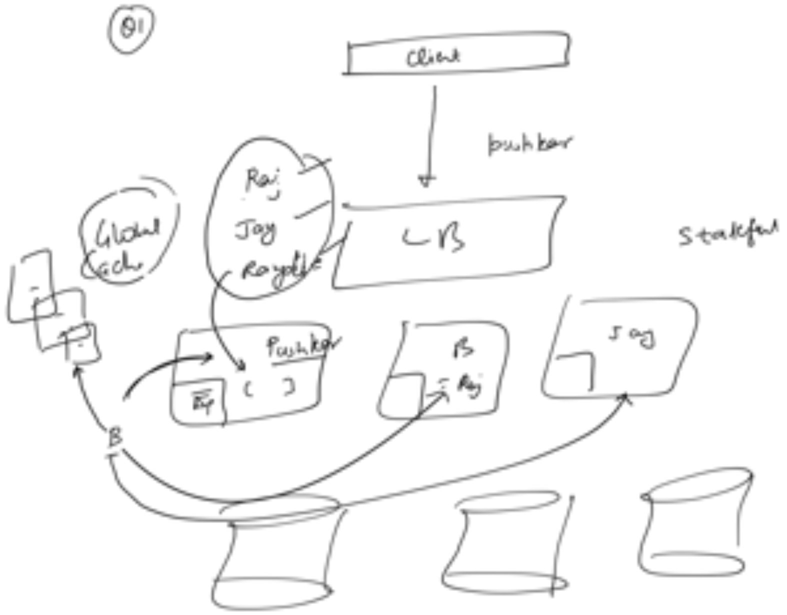
Ⓑ Cache Eviction Policy

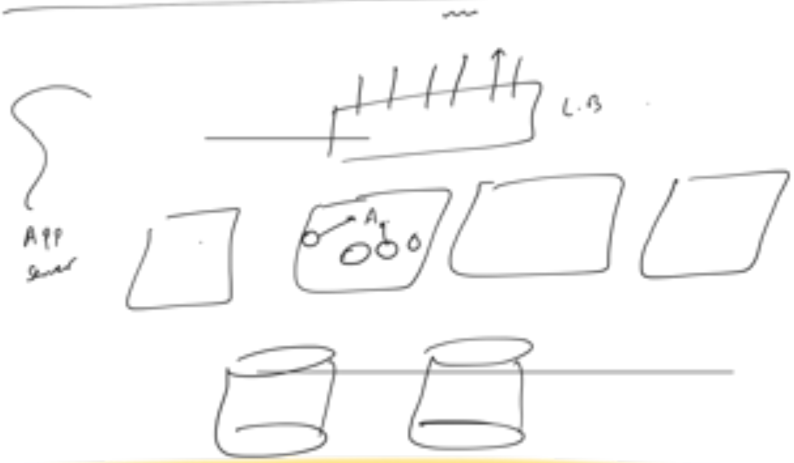
Ⓒ Amount of Cache Storage

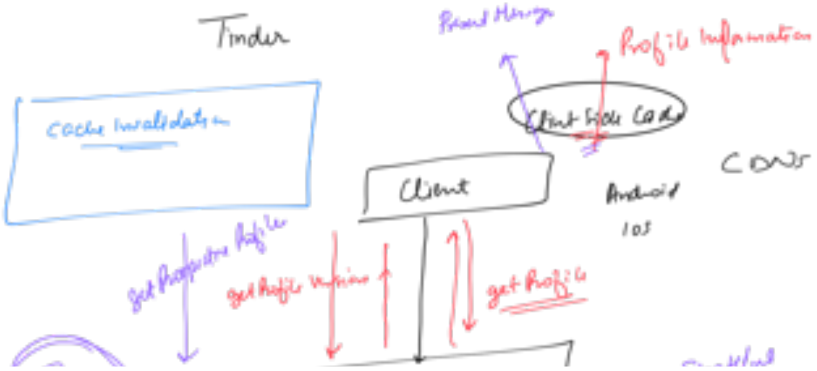
④ **Global Cache**

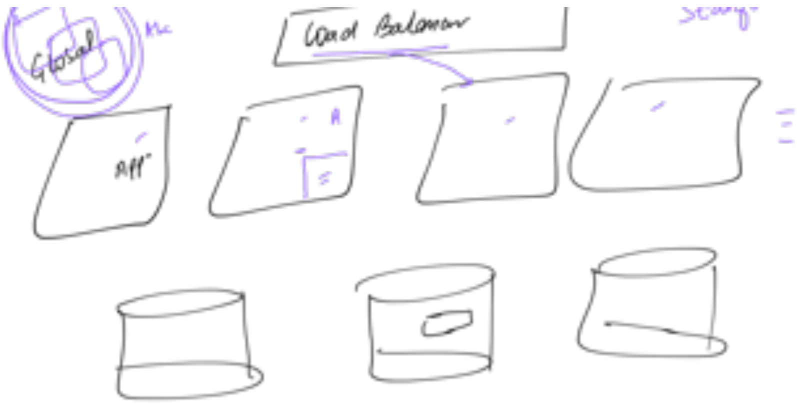



A
B
C


Q1
Client
pushkar
LB
Stateful
Global Cache
Raj
Jay
Raydle
Pushkar
B
Raj
Jay
Exp
B


L.B
App server
A


Tinder
Read Message
Profile Information
Cache invalidation
Client Side Cache
Cons
Client
Android
iOS
get Prospective Profile
get Profile version
get Profile
Stateful



1. Client side Cache = Profile Info

Timeline

Design a Facebook News Feed

1. Enlist all the features
   → Minimum Viable Product
2. Estimate the Scale
3. TradeOffs
4. Architecture Design
   → APIs
   → Data Storage
   → Data Flow

LinkedIn

Instagram

Quora

Twitter

I Ablity to view posts of interest

II Like/ Comment /share a post

III Multimedia Support → share Images /Videos

IV Bookmark a post

~~V~~ Making a friend / follow → ancillary feature

VI Suggesting friends → ancillary feature

VII Report a post

VIII Support hashtags → searching based on → hashtags

~~IX~~ Relevance → Time Decay
→ Connection Strength

X Ads

~~XI~~ Automatic Refresh

~~XII~~ Write a post

XIII Tagging users

B Estimation of scale

1. # users = 2 B →
2. DAU = 400 M → new feeds
3. # people who produce content or read a post = 40 M

4. # new posts / day = 2.5 × 40 M
= 100M

COMPUTE LOAD

A. writes / day = 100M

B. Reads / day = □ → × mul fact

by fetching news feed

= 400M × 25

10 000 M → 1,15000
= 10,000 M

reads / day

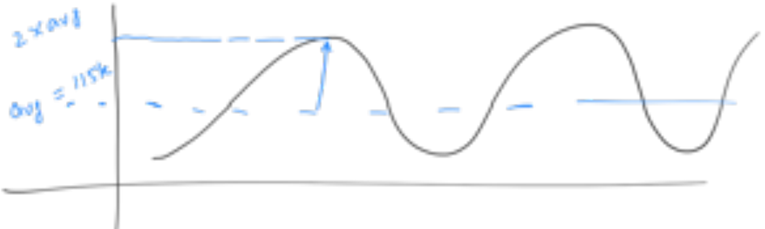
← 30M → $= \boxed{10B}$ reads/day

Read QPS $= \dfrac{1\cancel{0}\ B \ \cancel{10^5}\ 10^{\cancel{6}} \times 7}{\cancel{2}4 \times 6\cancel{0} \times 6\cancel{0}}$

$= 115000$ QPS

$= \boxed{115k \text{ QPS}}$

Write QPS $\dfrac{100M}{86400} = \boxed{1000 \text{ writes/sec}}$

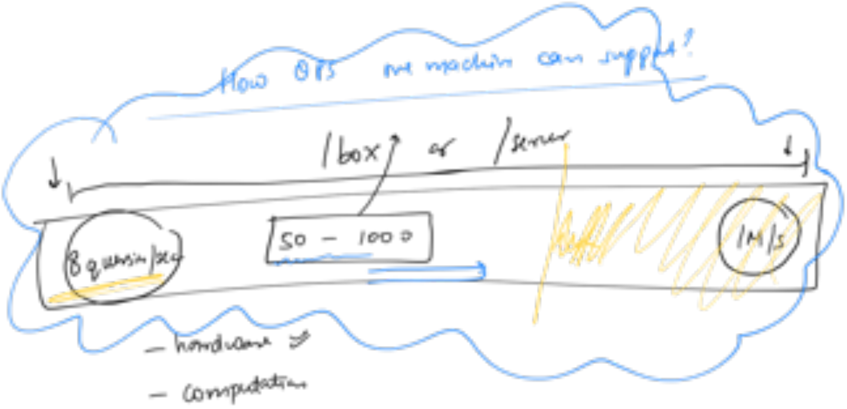


Read QPS = 115k × 2

$= \boxed{230k}$

Write QPS = 2000

How QPS one machine can support?



- hardware ≠
- computation
- data fetch

→ $\boxed{100 \text{ QPS / box}}$

1000

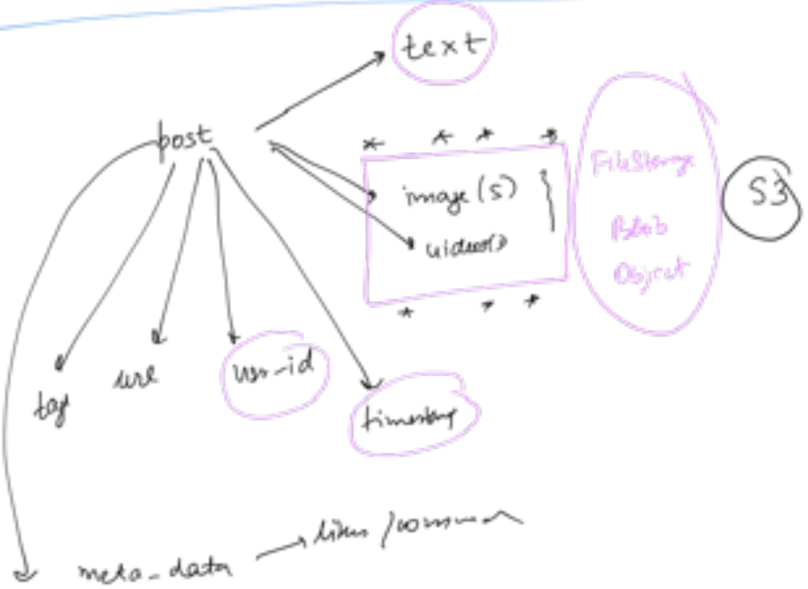
$$\boxed{\frac{230\text{k}}{100} = \#\text{ boxes}}$$

2300 boxes

Load Factor => 65% - 75%

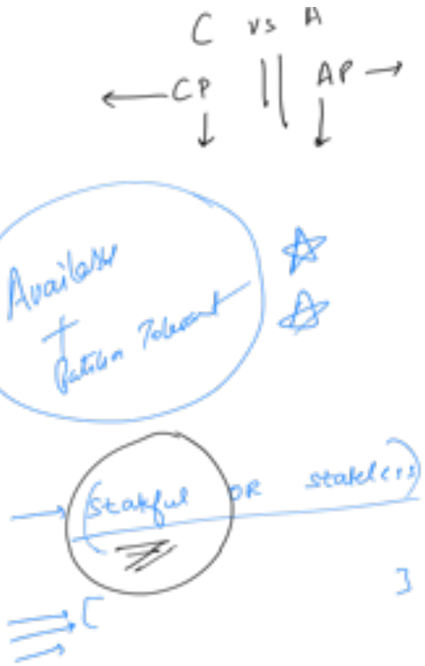
$$\frac{2300}{0.7} = \sim 4000$$

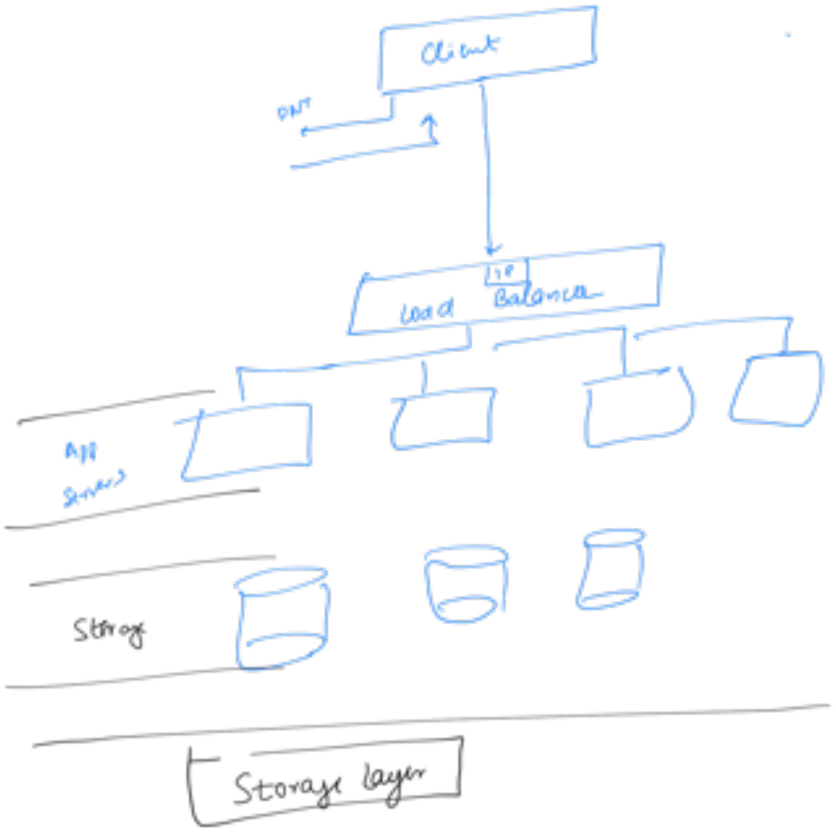
EC2 ≡ App Server



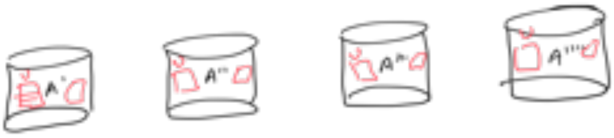
III Trade Off.

CAP Theorem

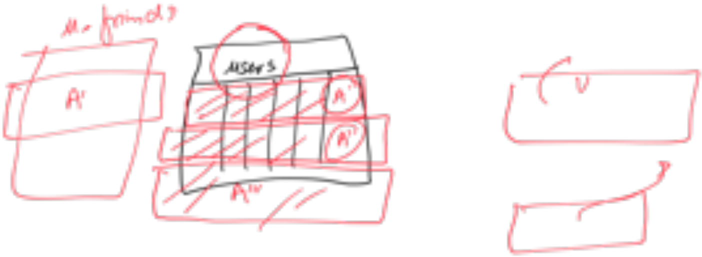

C vs A
CP
AP
Available
+
Partition Tolerant
Stateful
OR
stateless


Client
DNS
IP
Load Balancer
App Servers
Storage
Storage layer

2TB
MySQL



Sharding

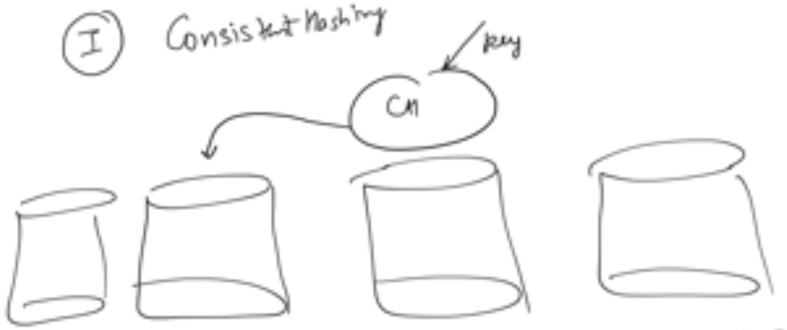


users

uid	f-name	l-name	age	score I	score II		

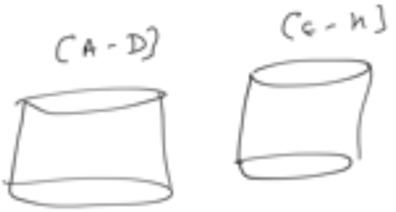
normalization

I Consistent Hashing

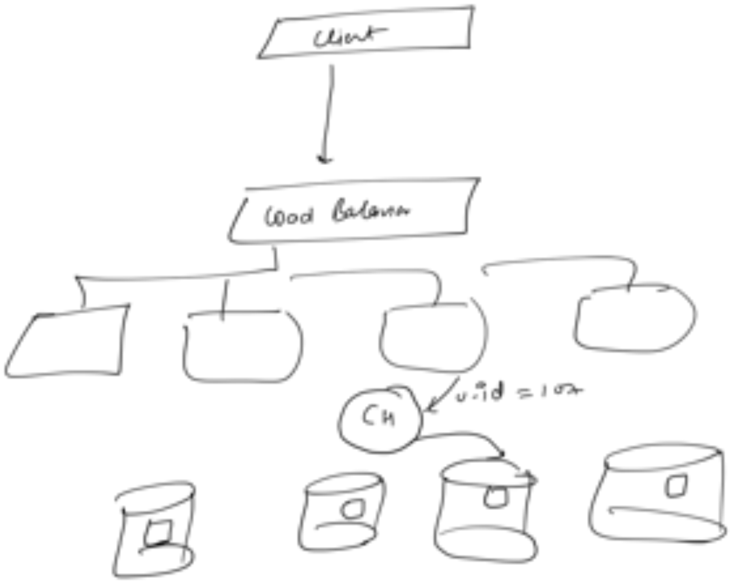


II Range based shard

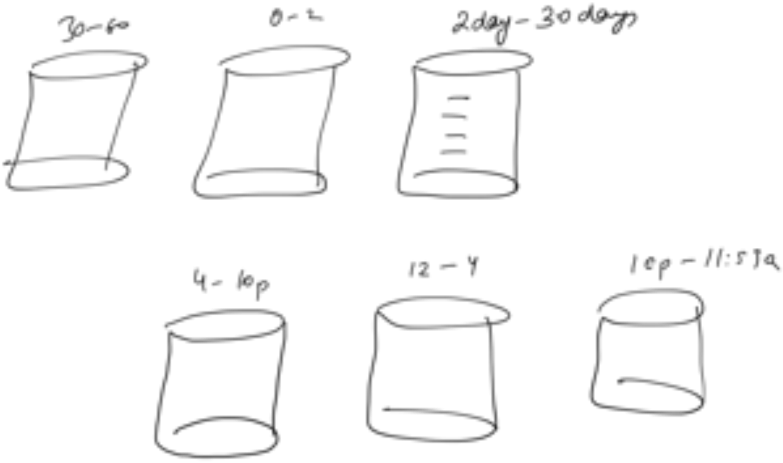
[90,000 - 98,000] [98,000 - 110,000]



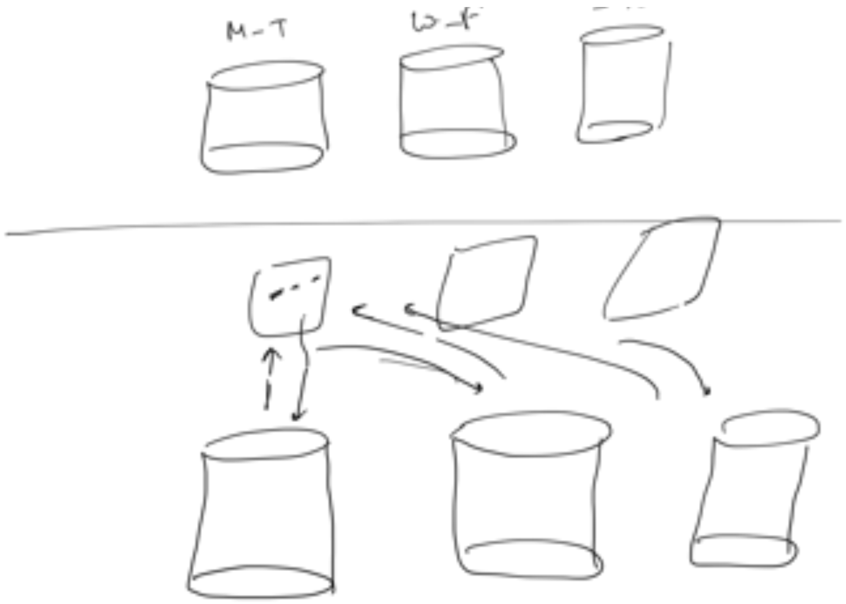
III Hot Shards

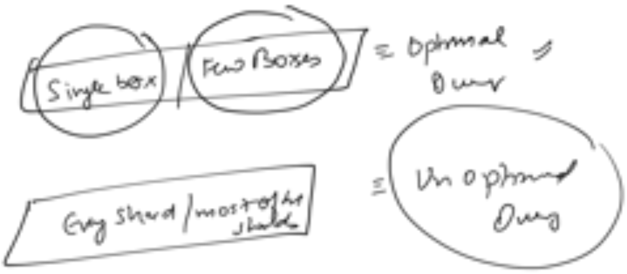


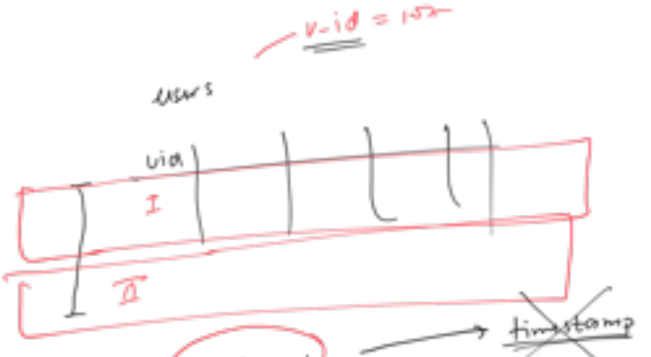
III Time Based Shardin



Sat





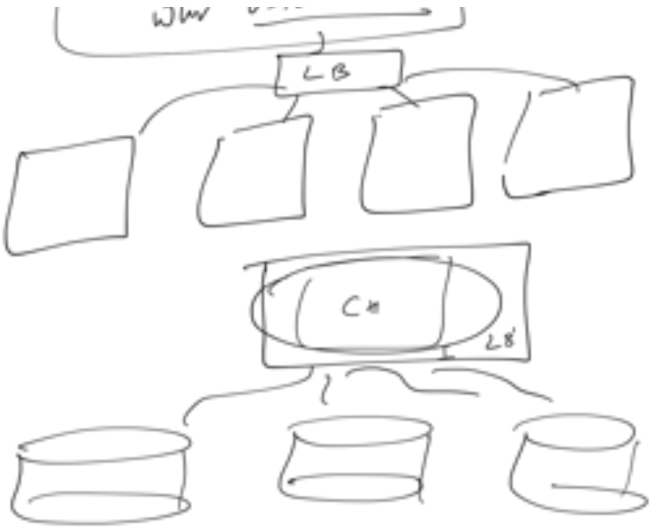


users_friend → ~~timestamp~~

u-id	f-id	timestamp
107	20	( )
107	22	( )
107	110	( )

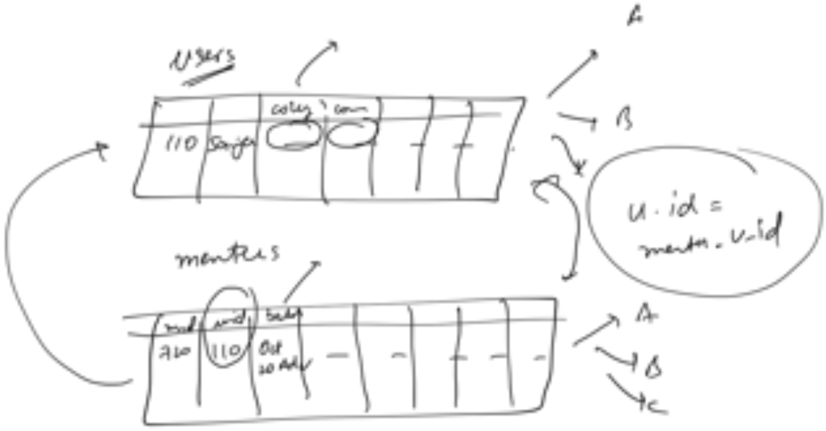
```
select *
from us-friends
      u-id = 107
```

X N


WWW
LB
CH
LB

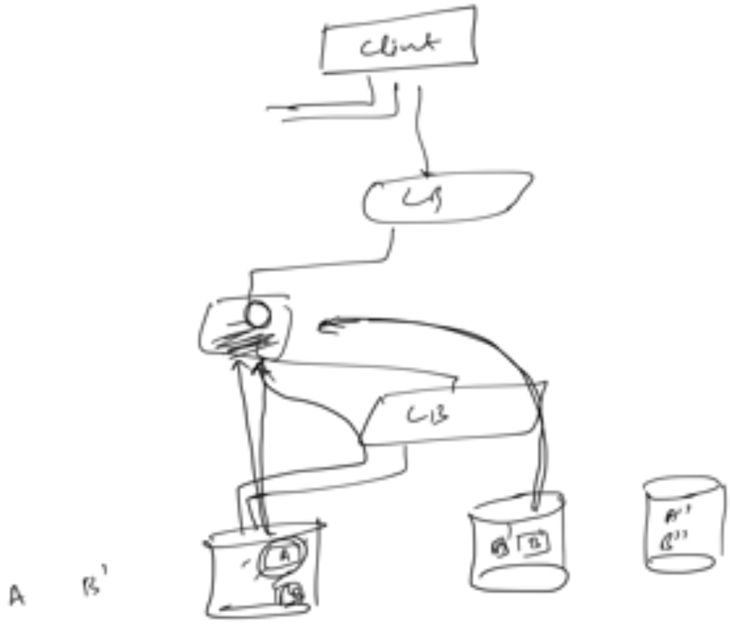
JOINS

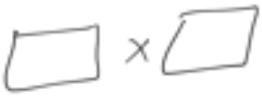
A
A B
A B


A
users
B
110 Sanjay
colly com
u.id = mentor.u_id
mentors
mid uid tada
710 110 Oct 20 AAV
A
B
C

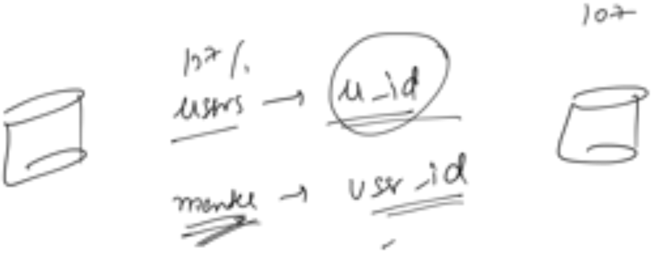
Intra Shard → JOIN

Inter shard JOIN

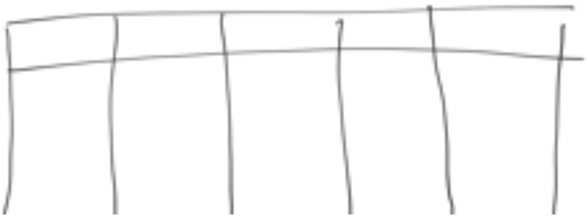


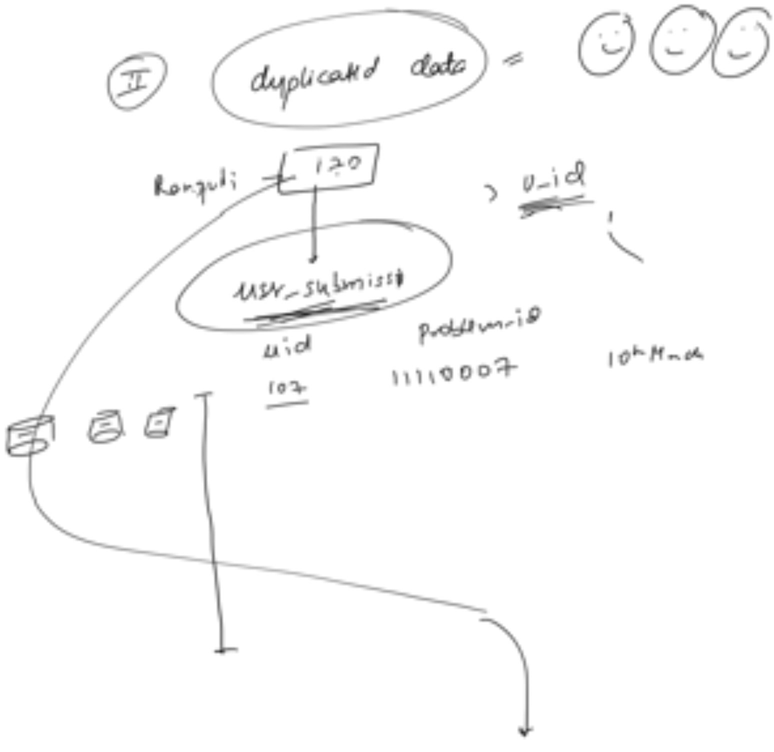


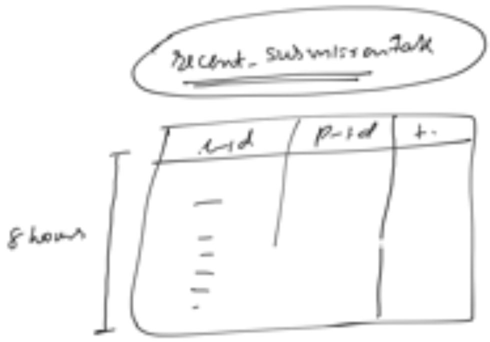
Inter shard JOINs Avoid



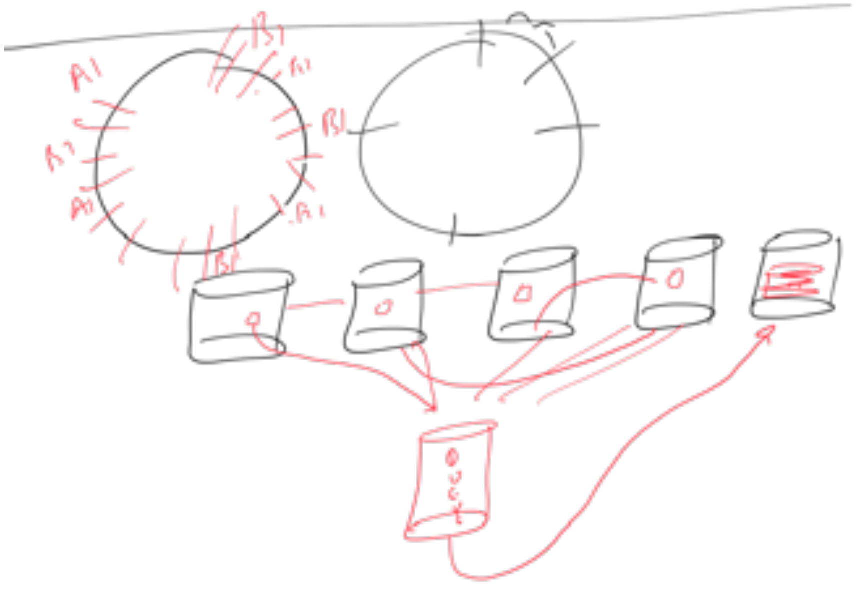
(I) denormalized way




II
duplicated data
=
Ranguli
120
u_id
user_submissi
uid
problem_id
10th March
107
11110007


recent_submission table
8 hours

uid	p_id	t.

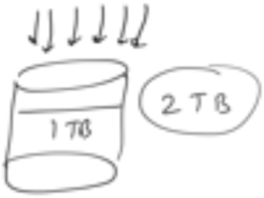

A1
B1
A1
B1
B1
A1
A1
B1

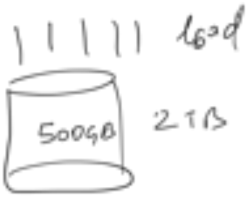
Sharding

500qps
125qps

→ scaled my storage
8 TB

→ scaled concurrent load

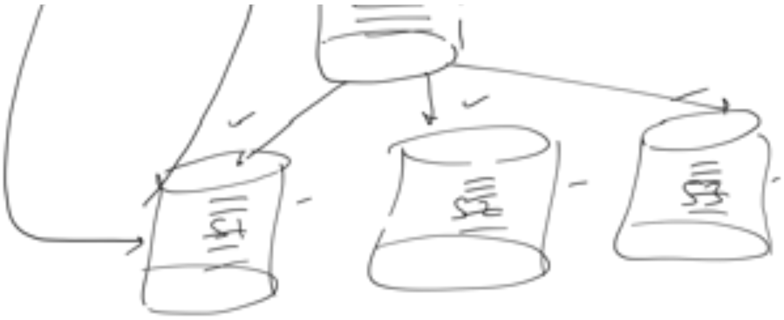




FAIL OVER ✓

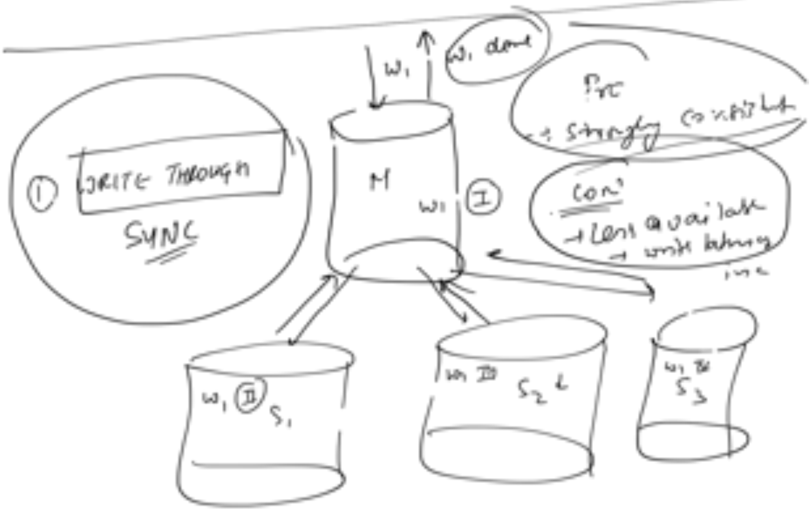
Master Slave Replication

A
Read
M
30 secs



MS Replication

- allows to scale READS
- requirement of backup has reduced
- increased READ Availabil

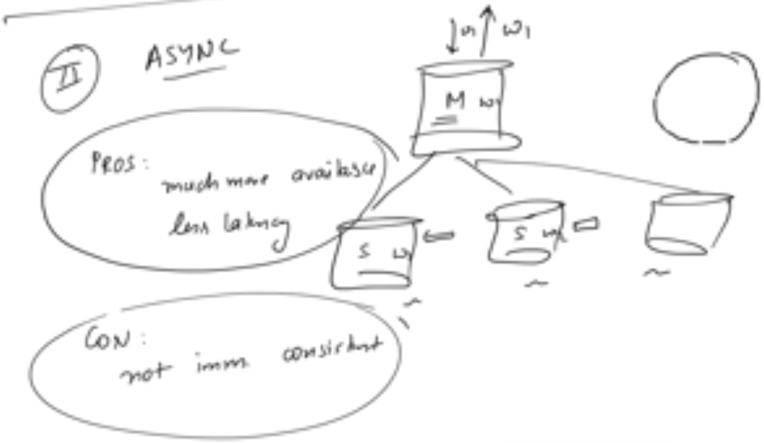


① WRITE THROUGH

SYNC

Pro
→ strongly consistent

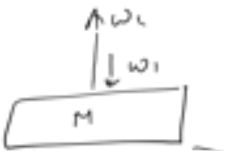
CON
→ Less available
→ write latency inc



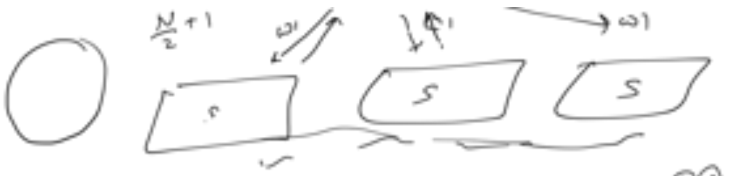
Ⅱ ASYNC

PROS: much more available
less latency

CON: not imm. consistent



Ⅲ QUORUM



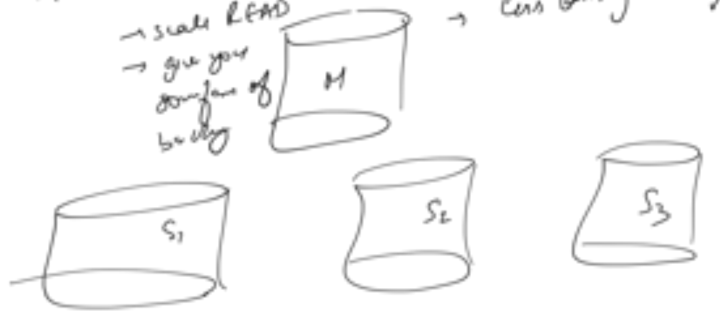
availabl
not consistent

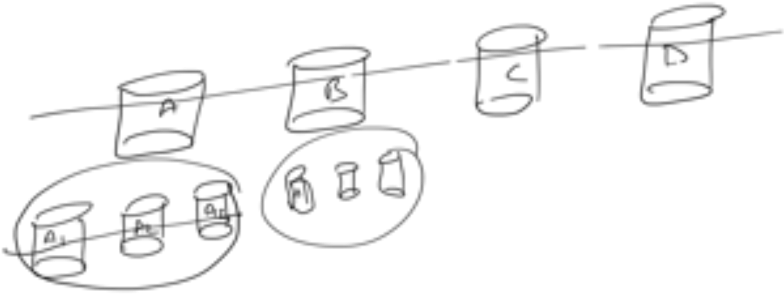
latency ↑↑

M-S Replication

→ scale READ
→ give your comfort of backup

READ
→ inc. availability
→ less latency latency





Master - Master Replication