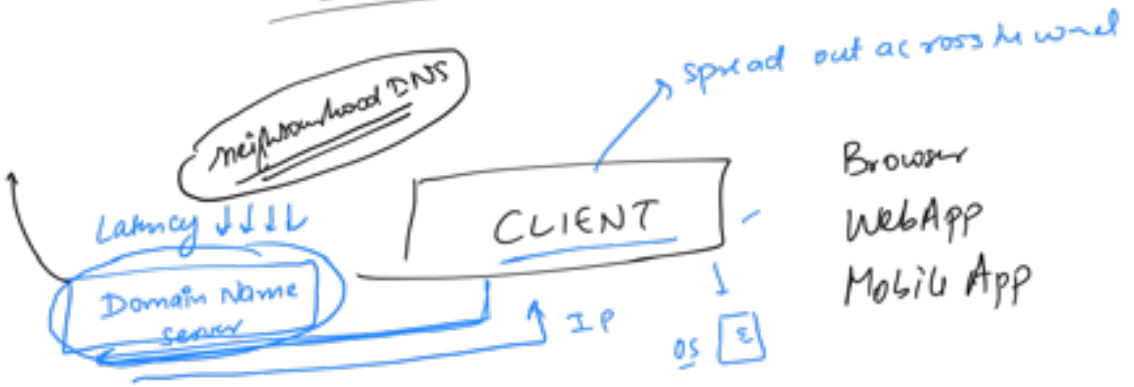
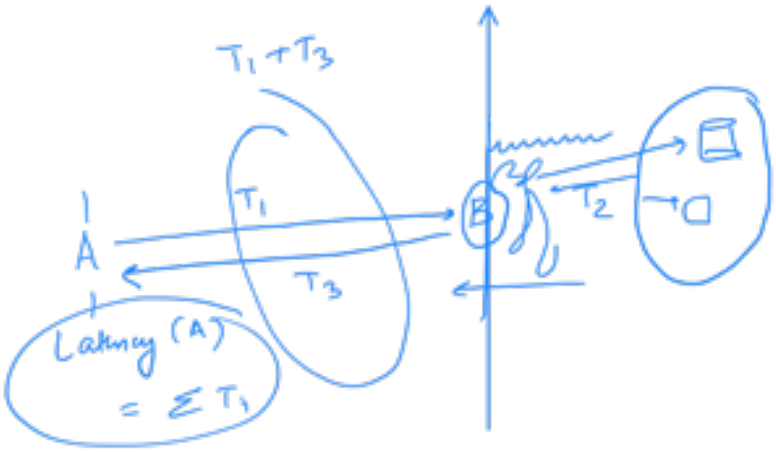
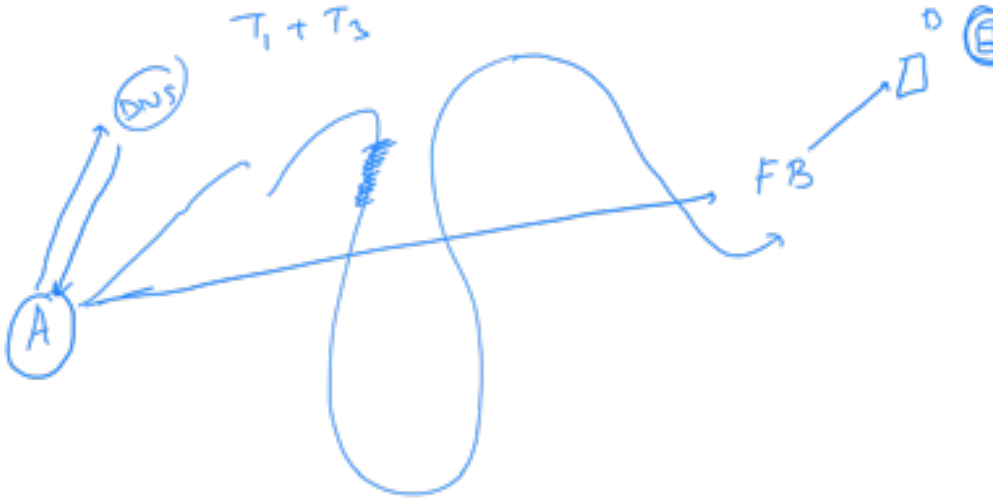
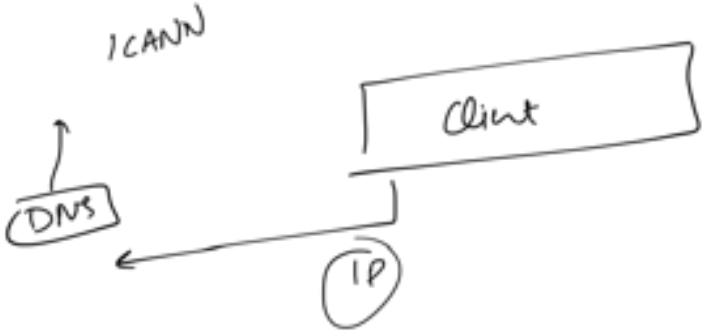
High Level Design

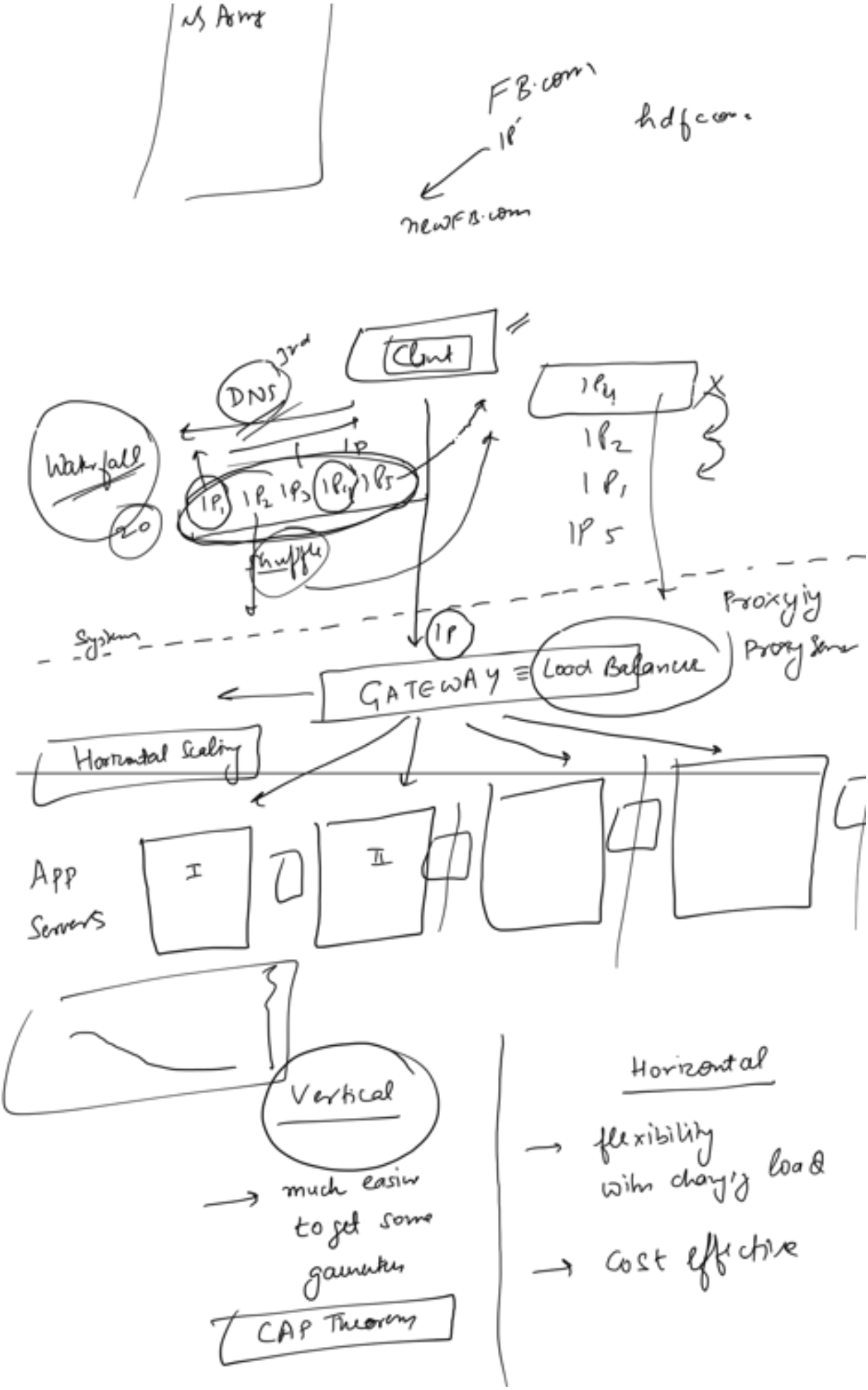


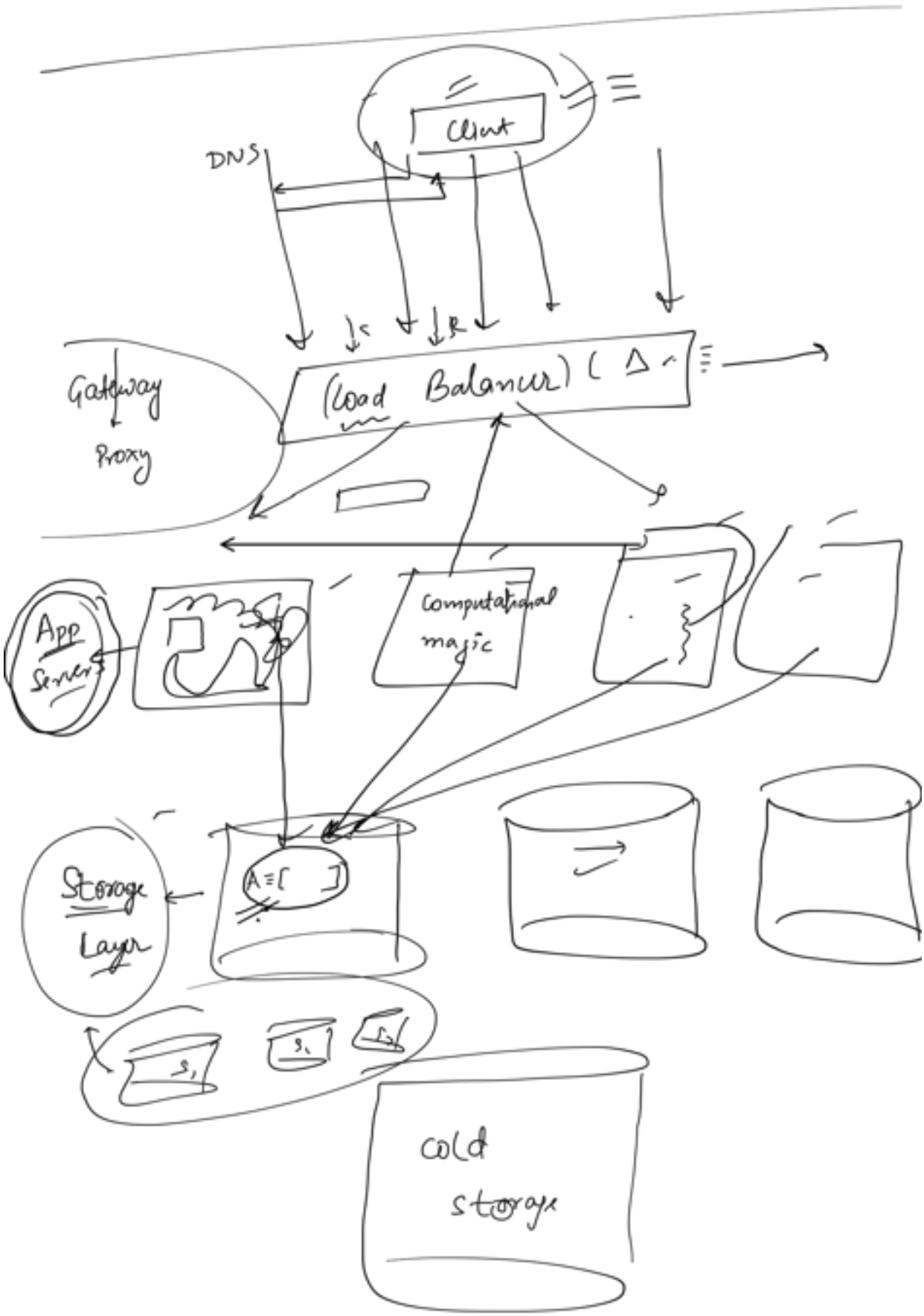




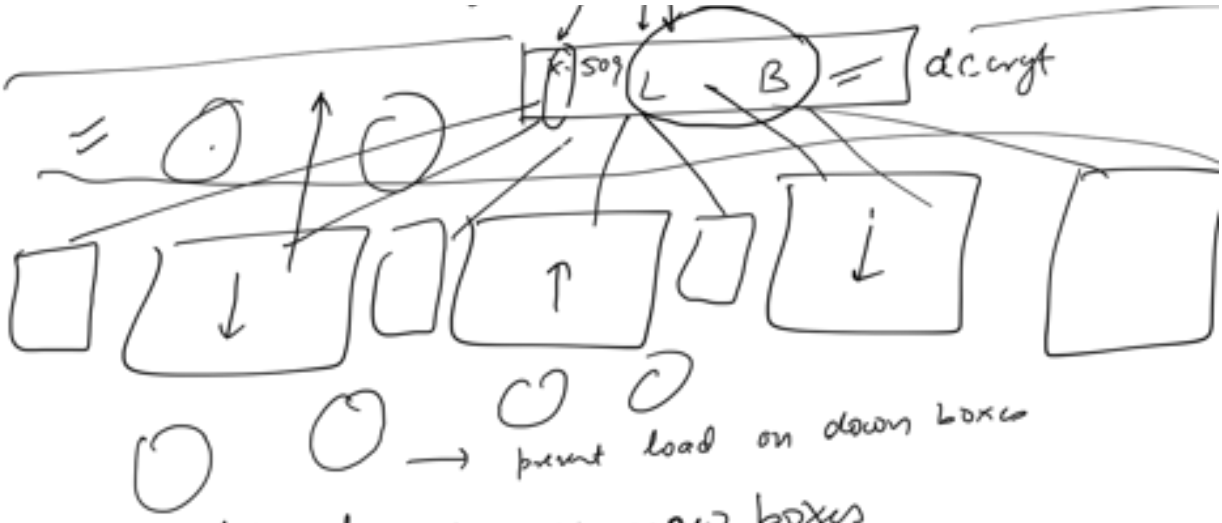


ISPs
Google
FB


FB.com
IP
newFB.com
hdfc.com
Client
DNS
3rd
Waterfall
IP1 IP2 IP3 IP4 IP5
Shuffle
IP4
IP2
IP1
IP5
System
IP
Proxying
GATEWAY = Load Balancer
Horizontal Scaling
App Servers
I
II
Vertical
much easier to get some guarantee
CAP Theorem
Horizontal
flexibility with changing load
cost effective


Client
DNS
Gateway
Proxy
Load Balancer
App Servers
Computational magic
Storage Layer
cold storage


internet
open internet



→ prevent load on down boxes

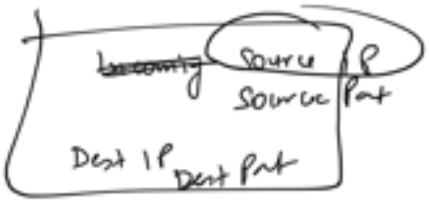
internet → register new boxes

→ balance out

— X.509 —

Layer 4 ≡ Transport layer

additional security



fast
simple

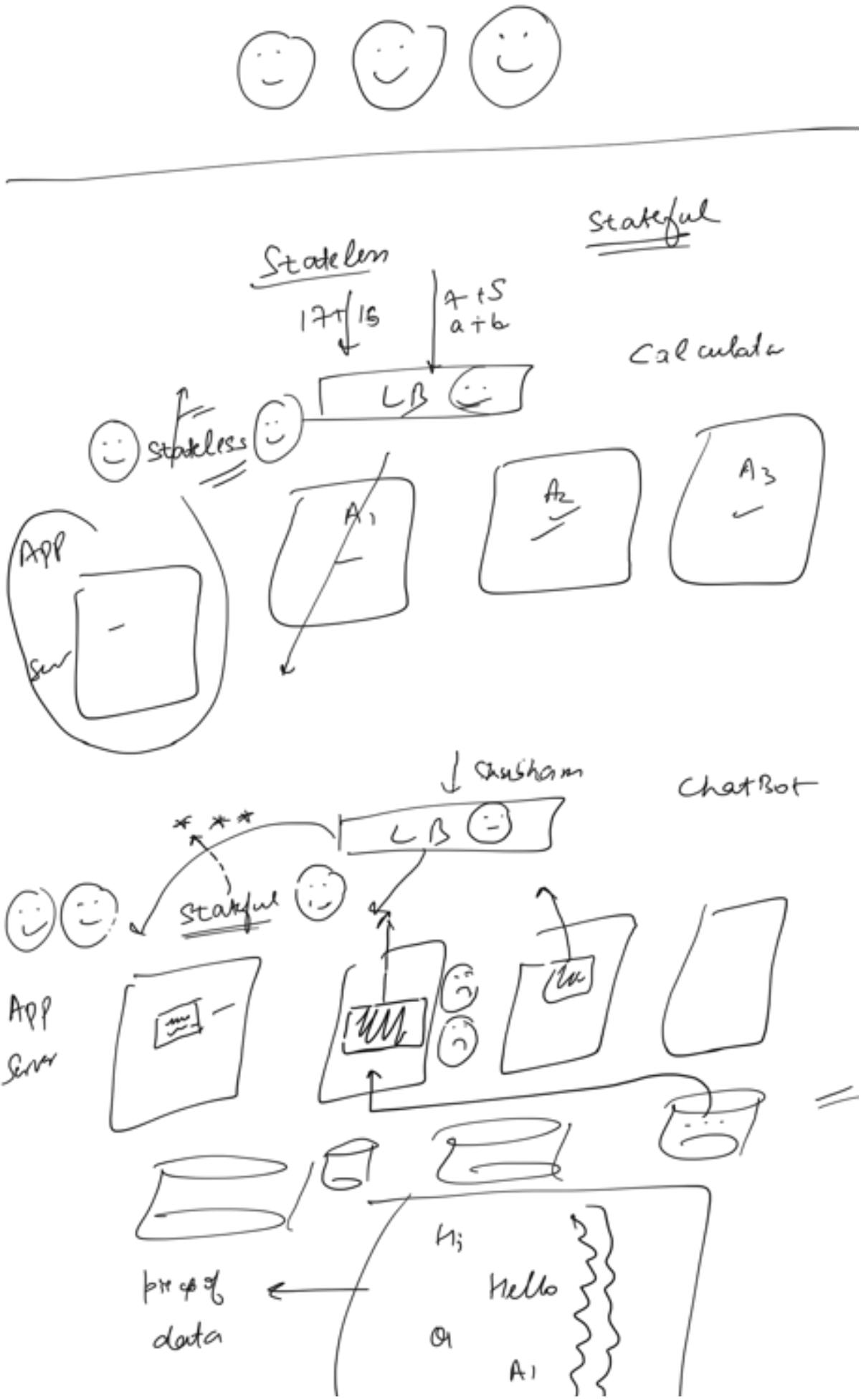
Layer 7 LB ≡ Application

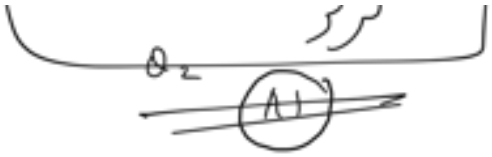
user id ≡ 707

uid = 107

much better flexibility

slower

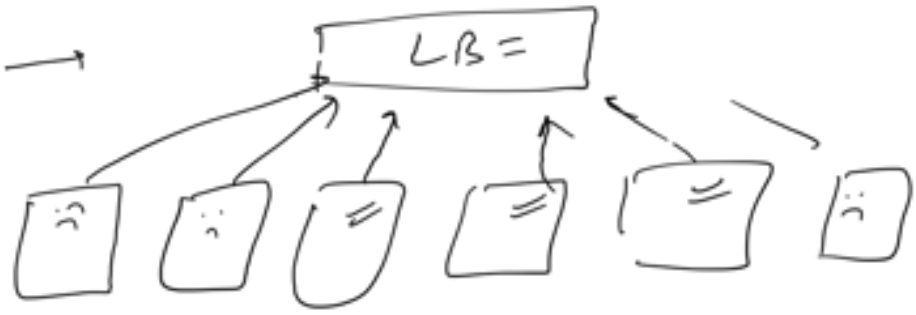

Stateful
Stateless
17+15
A+S
a+b
Calculate
LB
stateless
App
Ser
A1
A2
A3
Chatbot
LB
stateful
App
Ser
Hi
Hello
A1



Load Balancing

Stateless

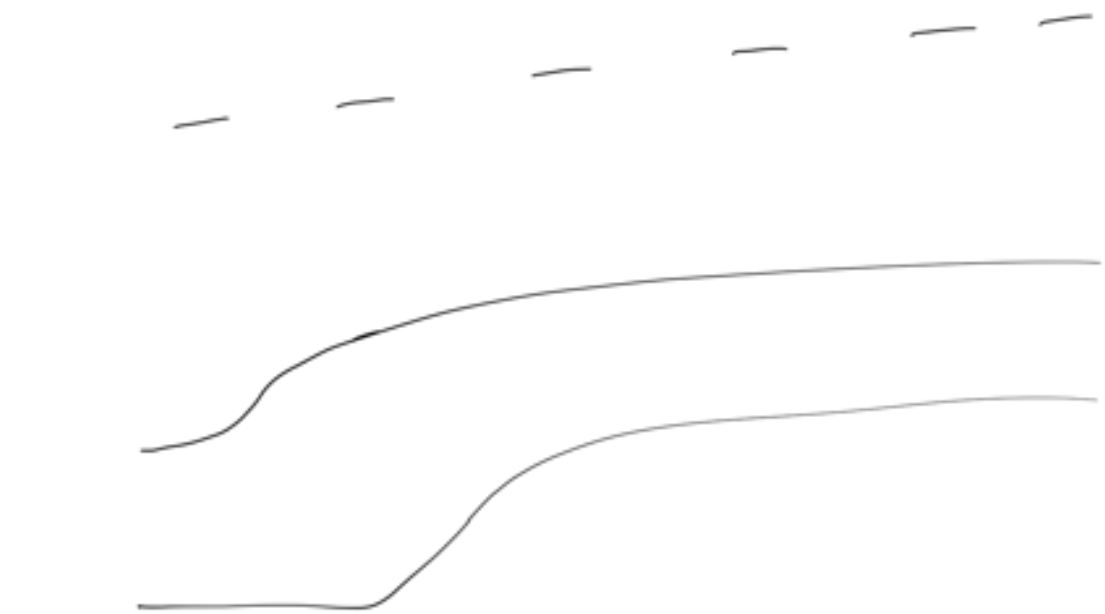
→ Round Robin

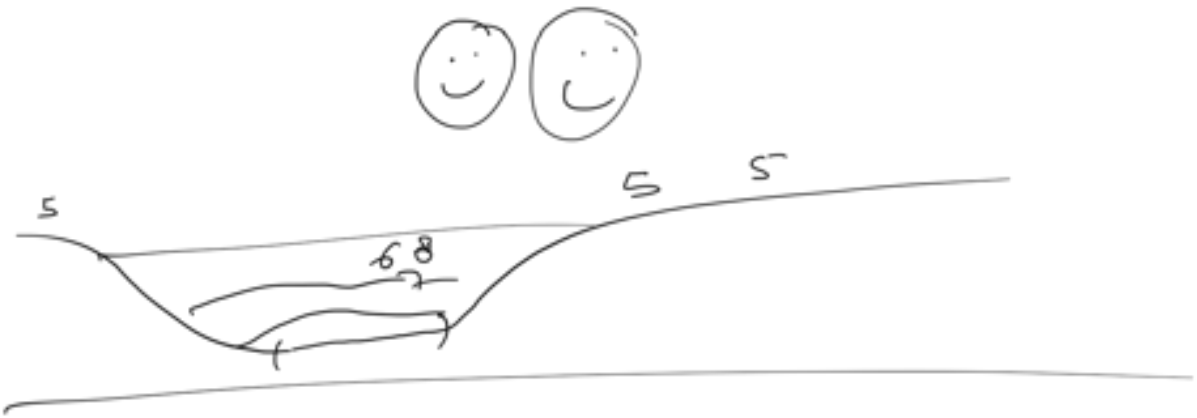


WRR

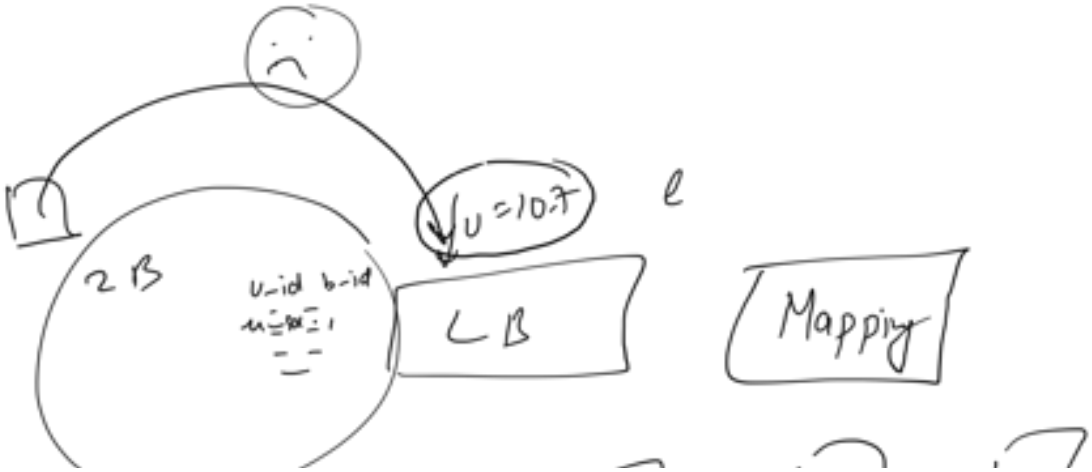
→ (Least Connection first)

→ WLK ARK





Load Balancing in Statfull



Sol (1) Maintain Map

BAD

map-size = order of key-size

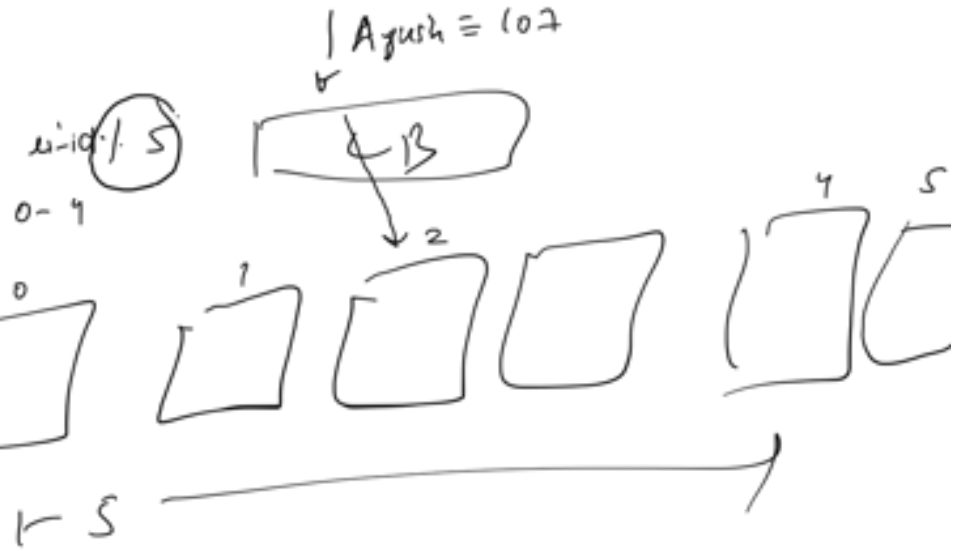
Load Balancing fence will slow down

Sol (2) Hash

BAD!!!

10 boxes ≡ 10 App Server

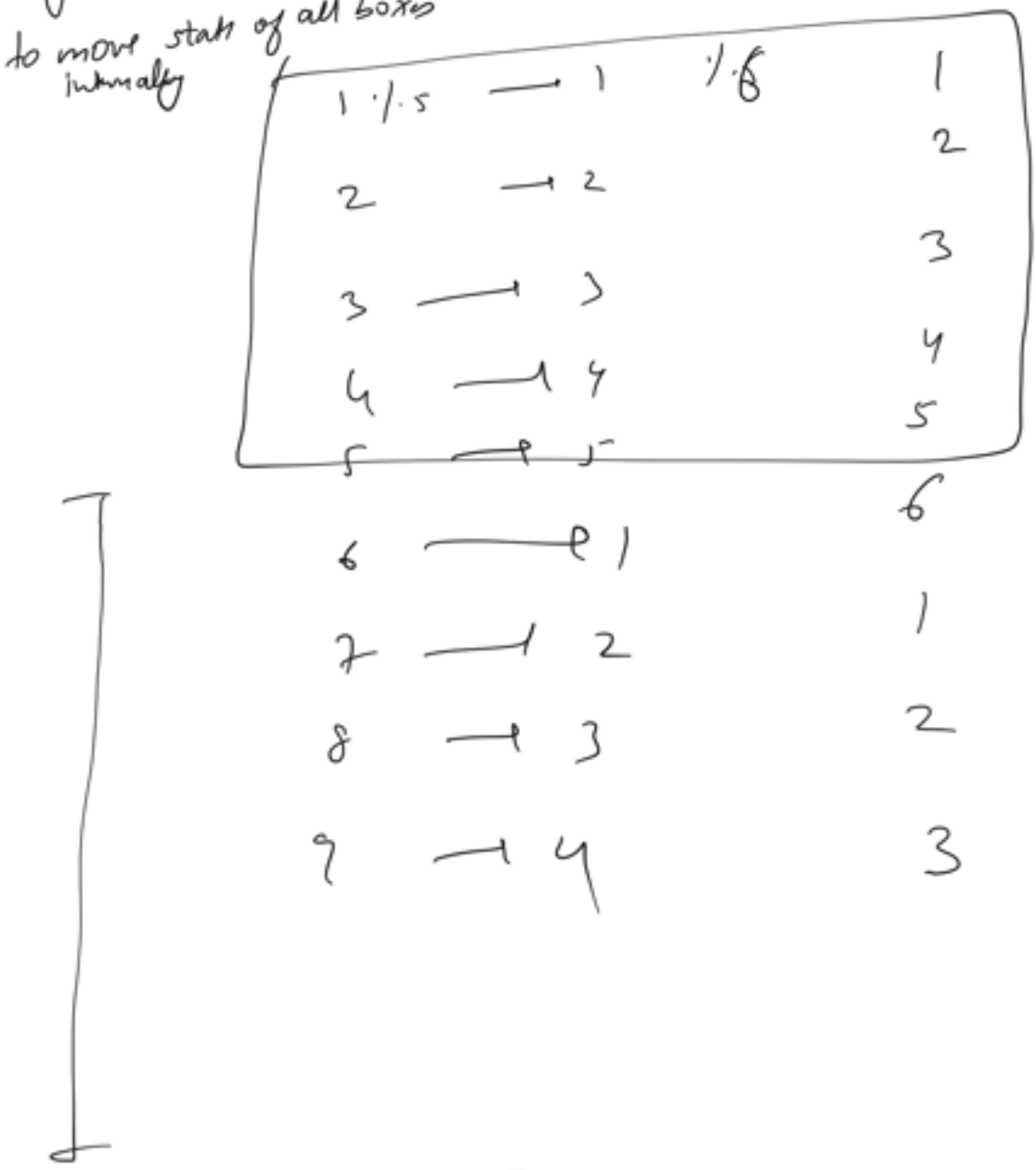
uid % 10 = 0-9



→ gaurantee of stickiness using hashing

→ LB ☺

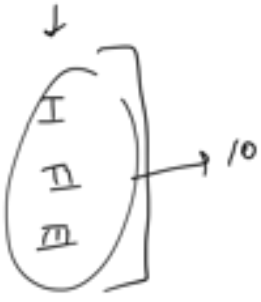
Downside
On server inc/dec
you will have ⅙
to move state of all boxes internally



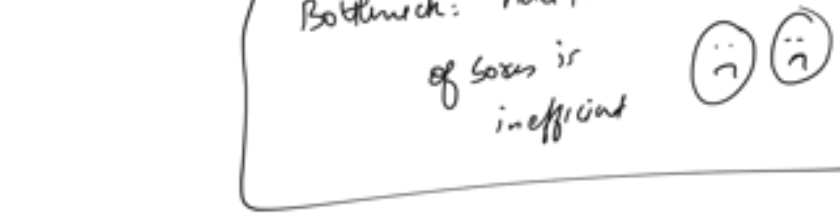
in LB

③ Ranges

1 - 1000
1001 - 5000
5001 - 10000



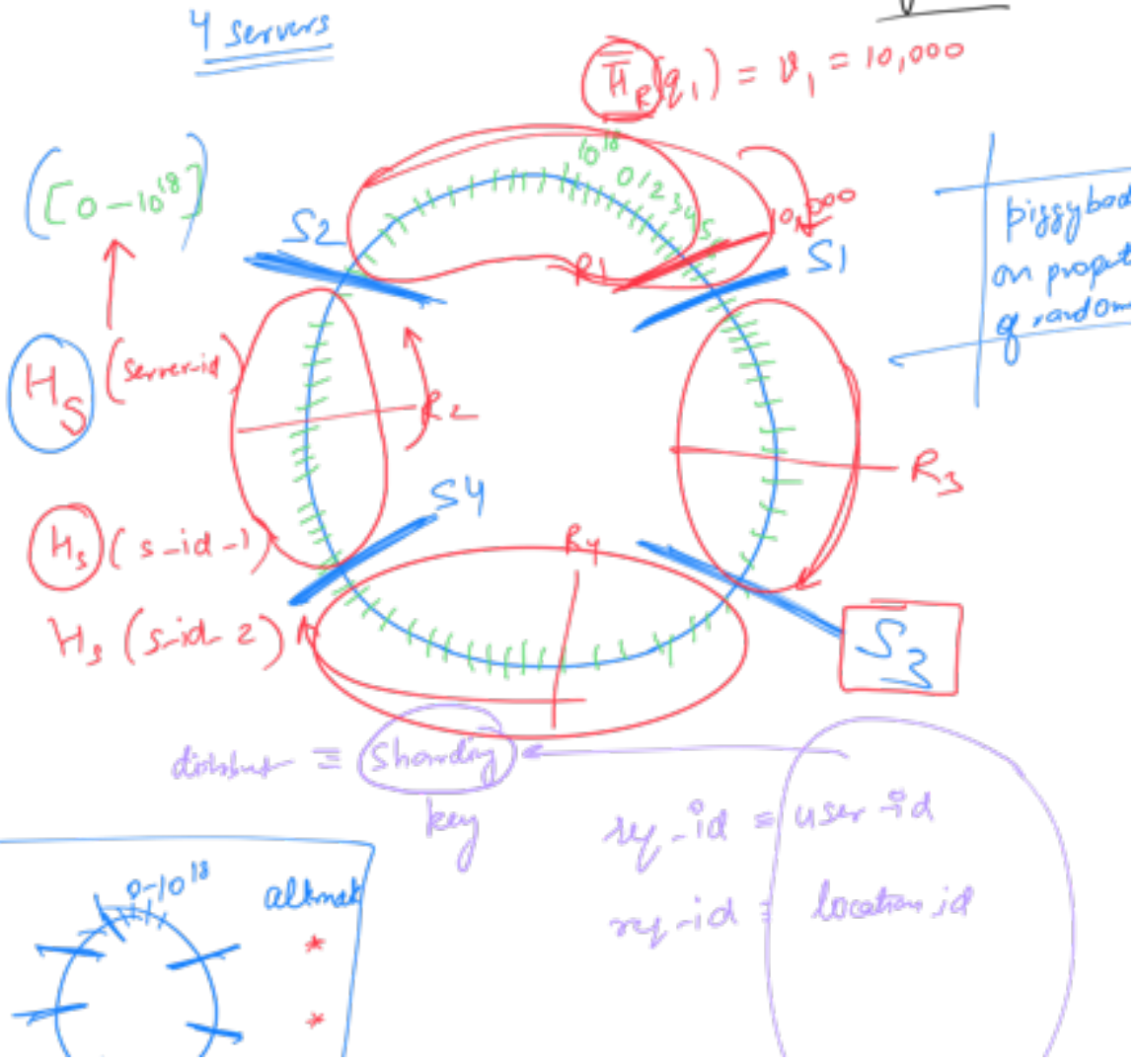
Bottleneck: Add / Removal of servers is inefficient



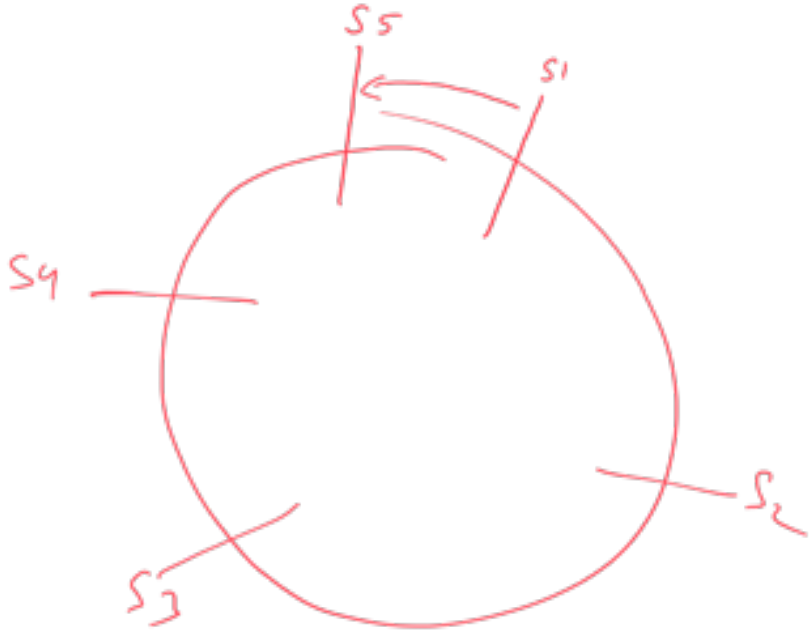
4 Consistent Hashing

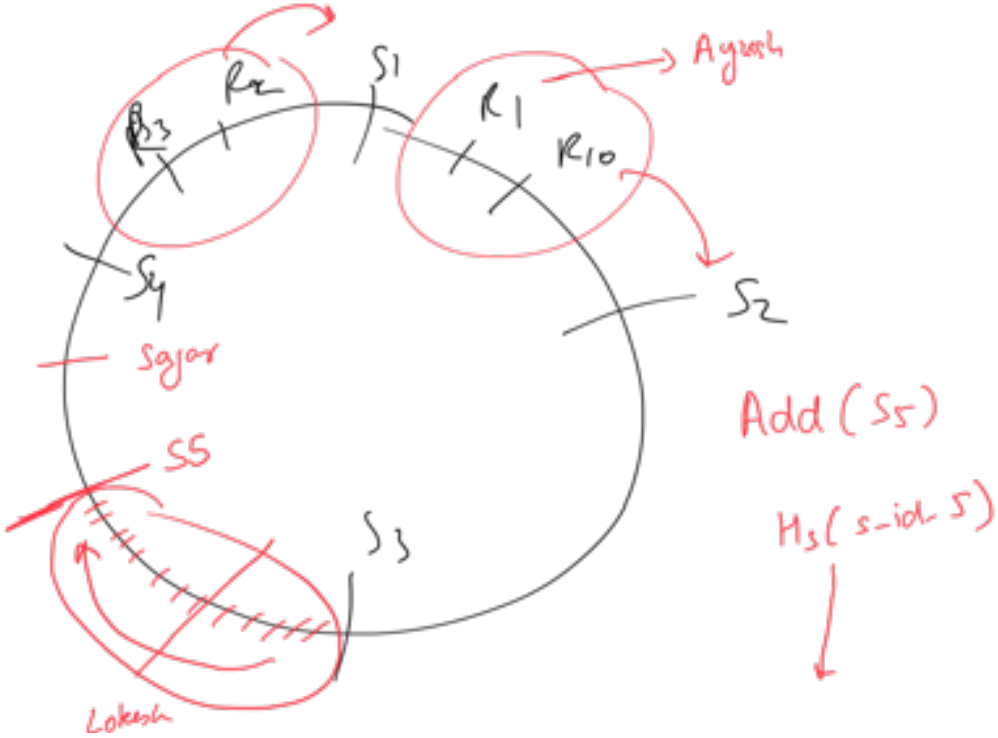
LB for Stateful Applications

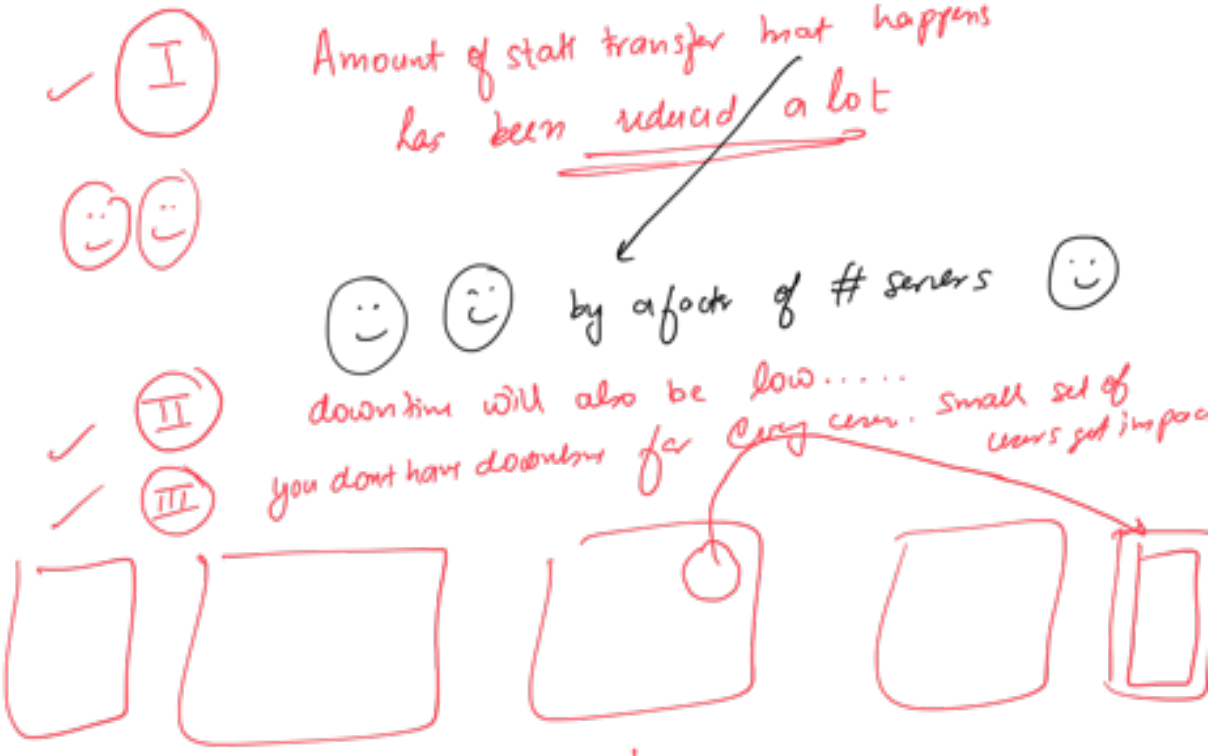
const addition/removal of servers




H_R (req-id)
[0 - 10^{18}]

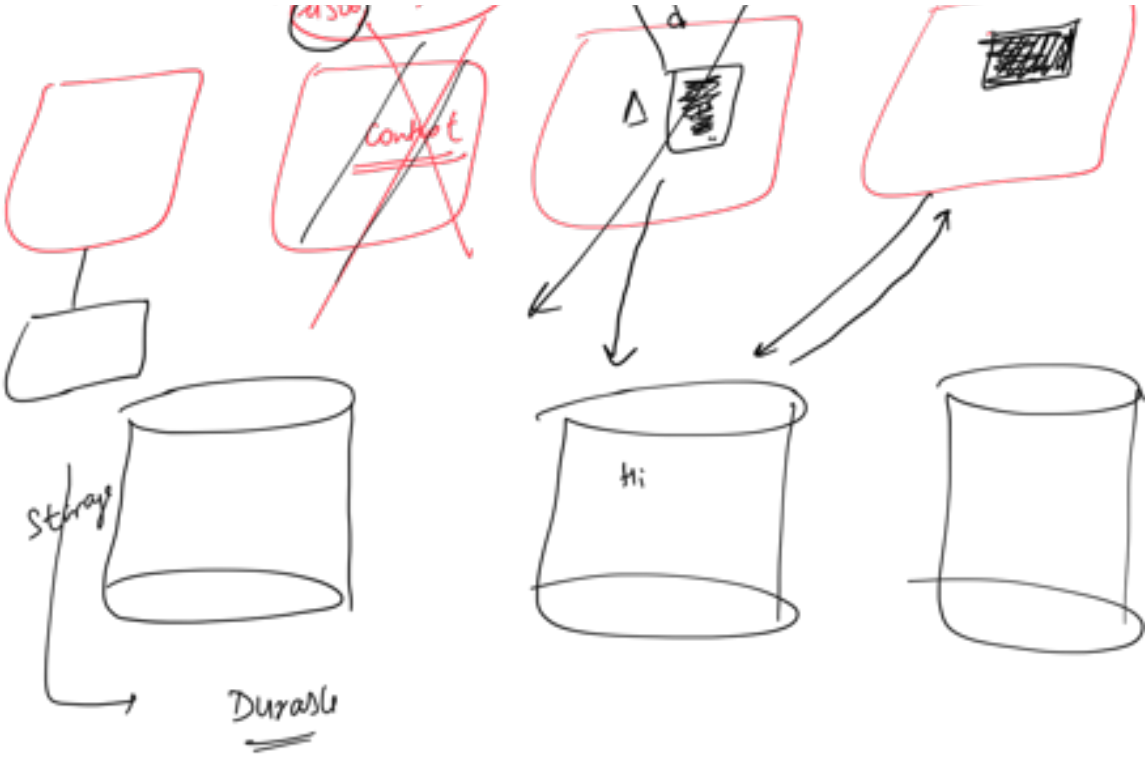

S5
S1
S4
S2
S3

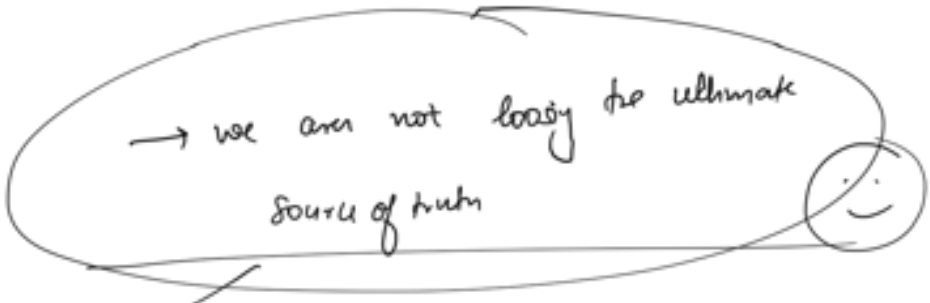

R2
S1
R1
Ayush
R3
R10
S4
S2
Sajar
Add (S5)
S5
S3
H_S(s-id, 5)
Lokesh


I
Amount of state transfer that happens has been reduced a lot
by a factor of # servers
II
III
downtime will also be low.....
you dont have downtime for every server. small set of users get impac


S1
S3
S4


μ500
μ1, μ100





(A) cold start ≡ request latency inc for some time

Hi

Hello

θ_1

A_1

θ_2

A_2

Online Learning Algo's ≡ model

(B) pre-emptive copying

storage layer → new App Server

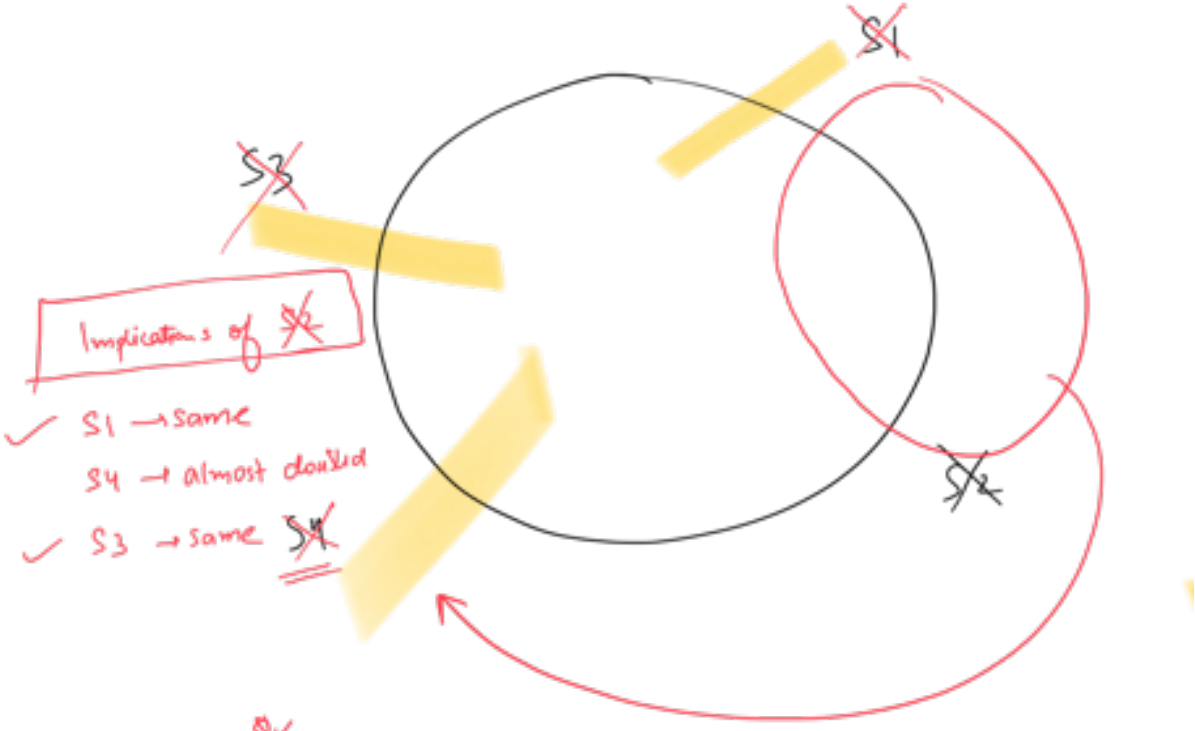
(I) ←

(II) ← Mslave

Solved problem

'impact of server addition or removal

'is minimized

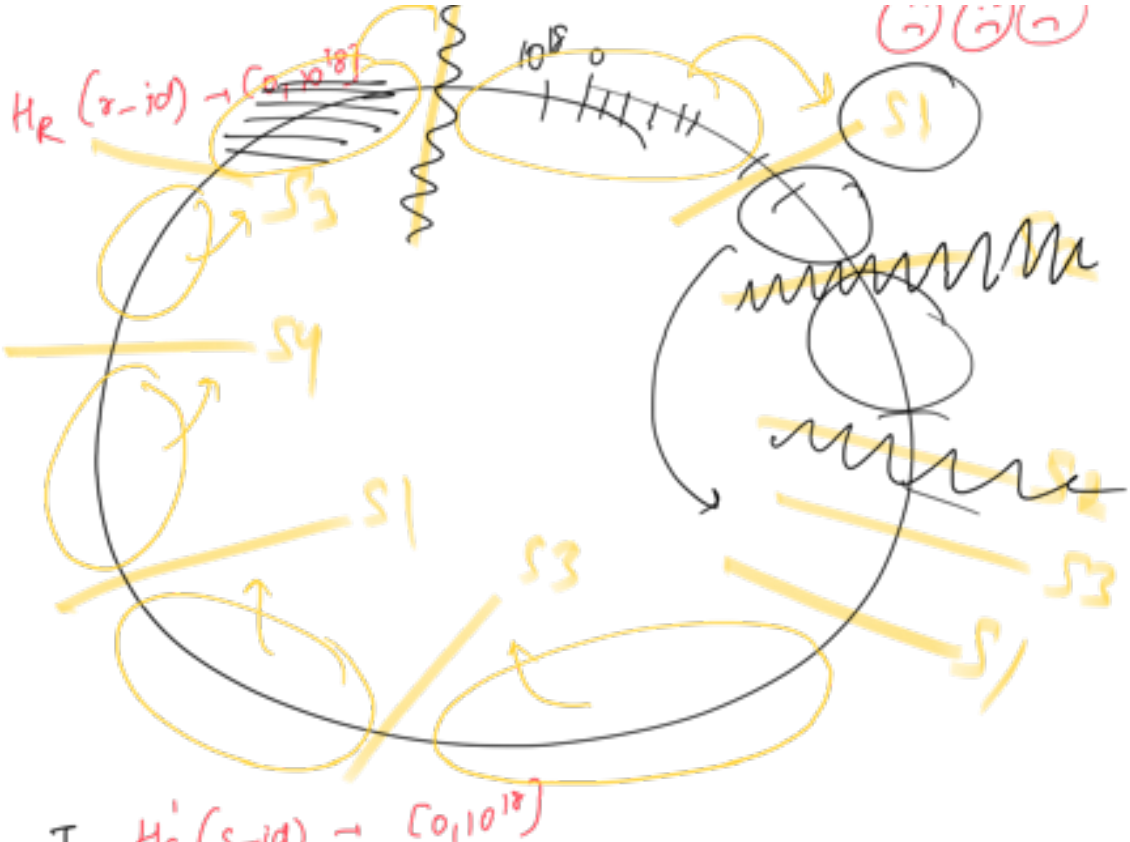


S1 ≡ same

S3 ≡ ∃X

"Cascadiy Failures"

S2 – evrgh will go dowon

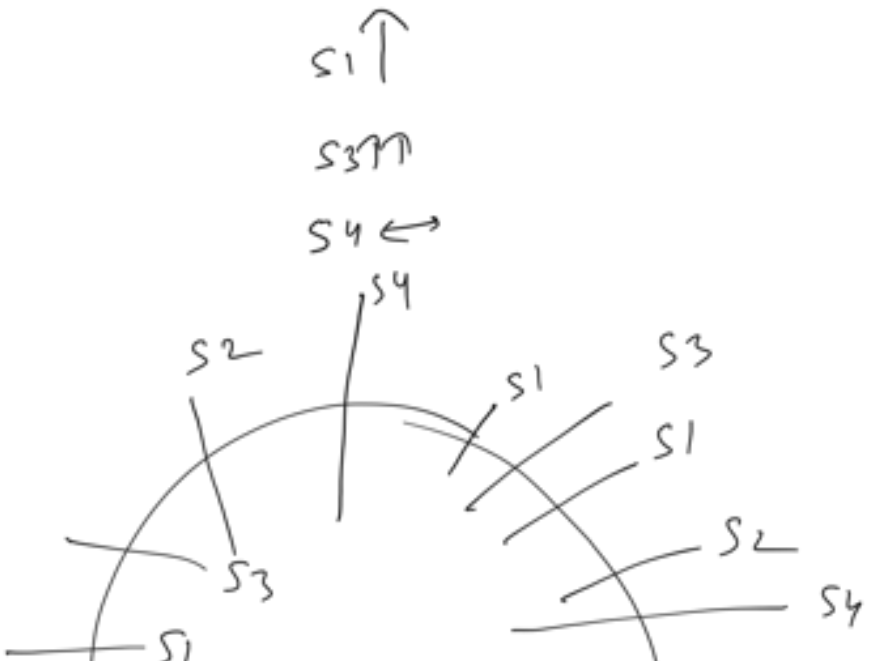


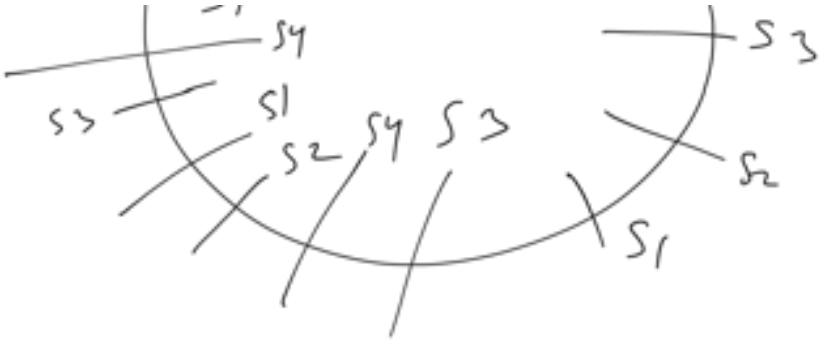
H_S^1 (s-id) → $[0, 10^{18}]$

H_S^2 (s-id)

H_S^3 (s-id)

|servers| X |H_S func|

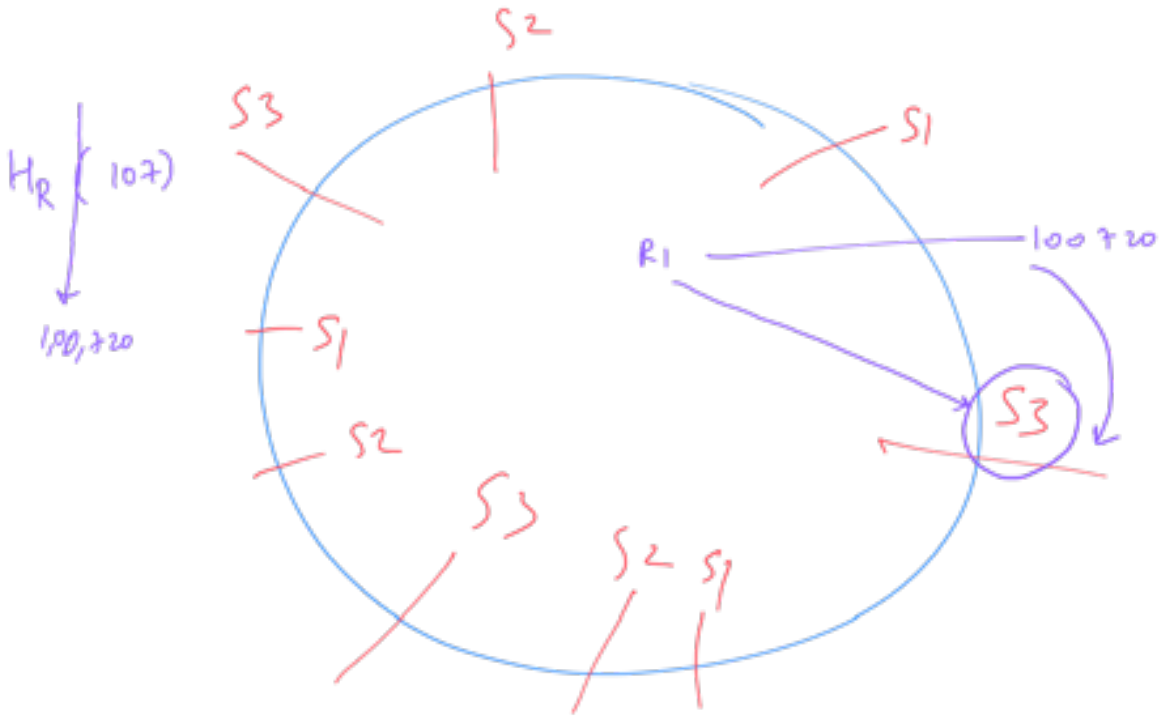




Solved

Cascading Failures

[Now, the load will get divided almost equally.



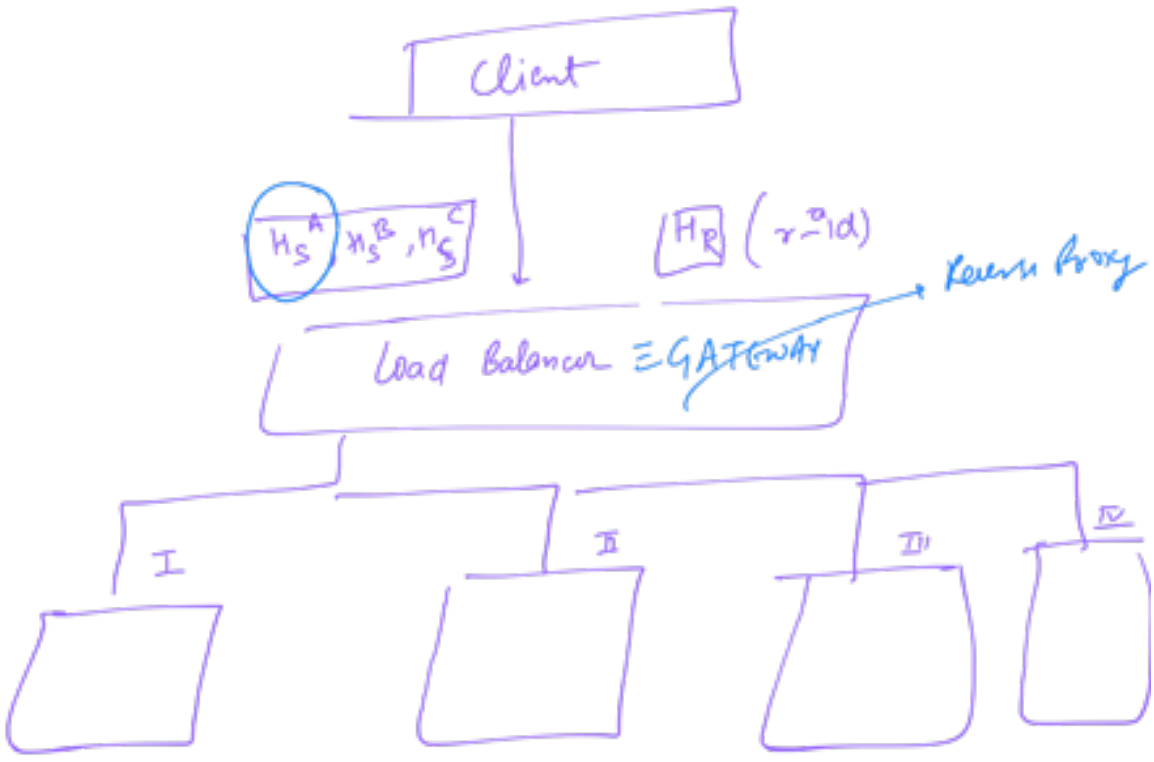
$|H_S| \uparrow\uparrow \equiv$ Randomness $\uparrow\uparrow$

$\downarrow$

Equitable Distribution $\uparrow\uparrow$

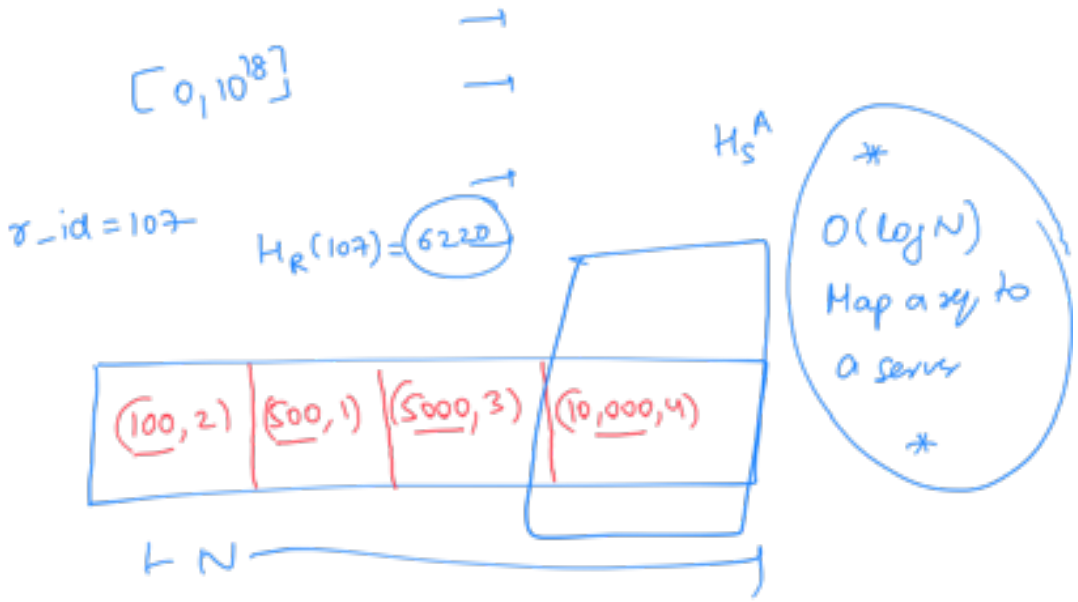
(H_S) TTT $\equiv$ time for Resolution / TTT

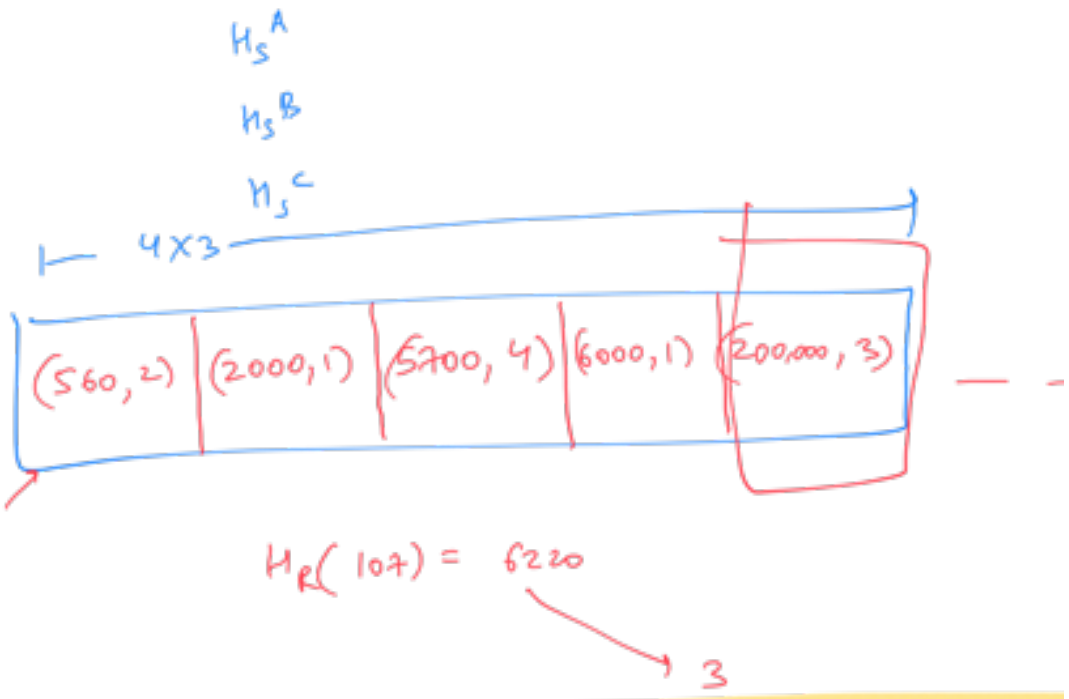
ASG



10 servers

H_S^A (i-id) ⟶

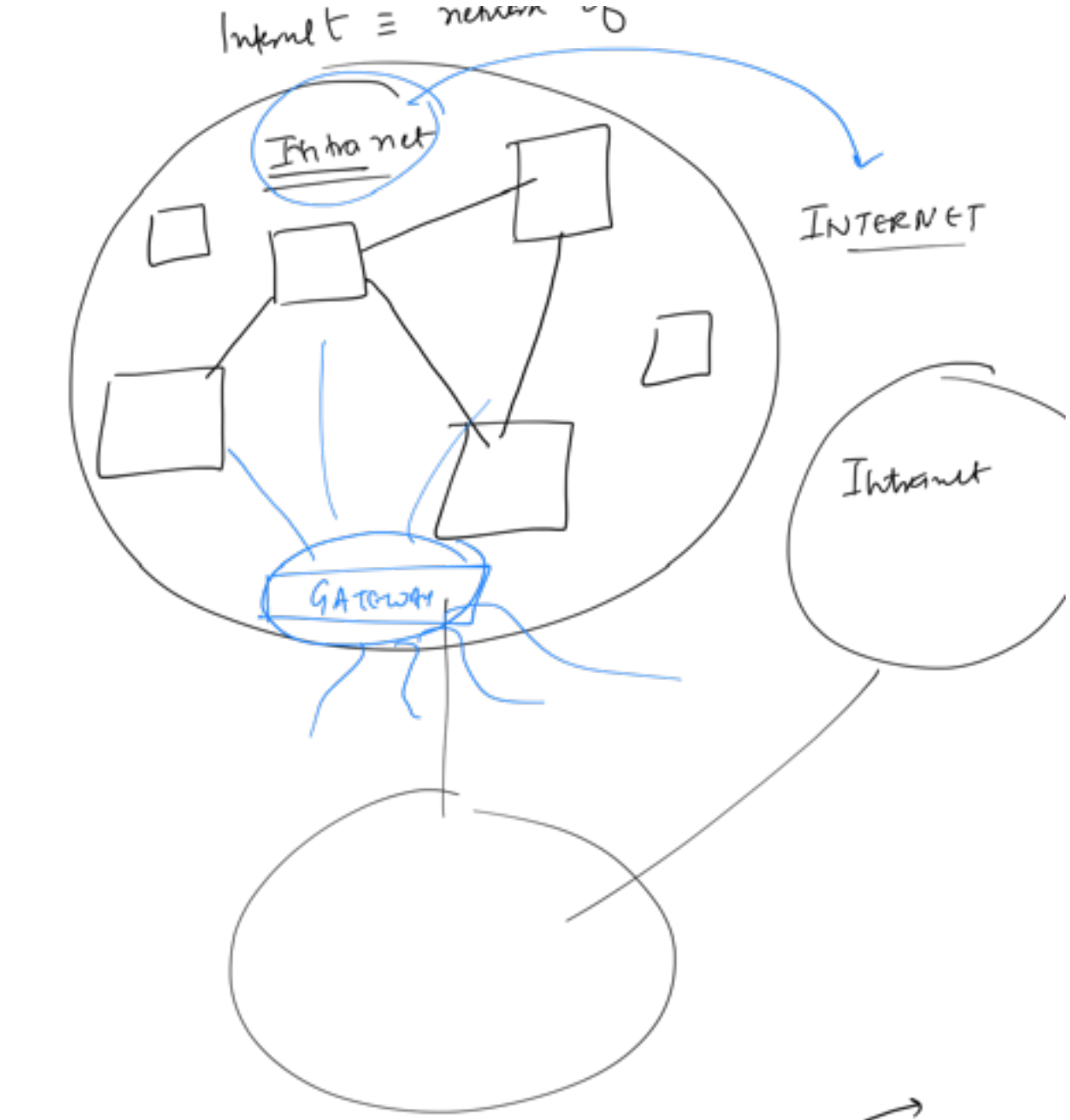


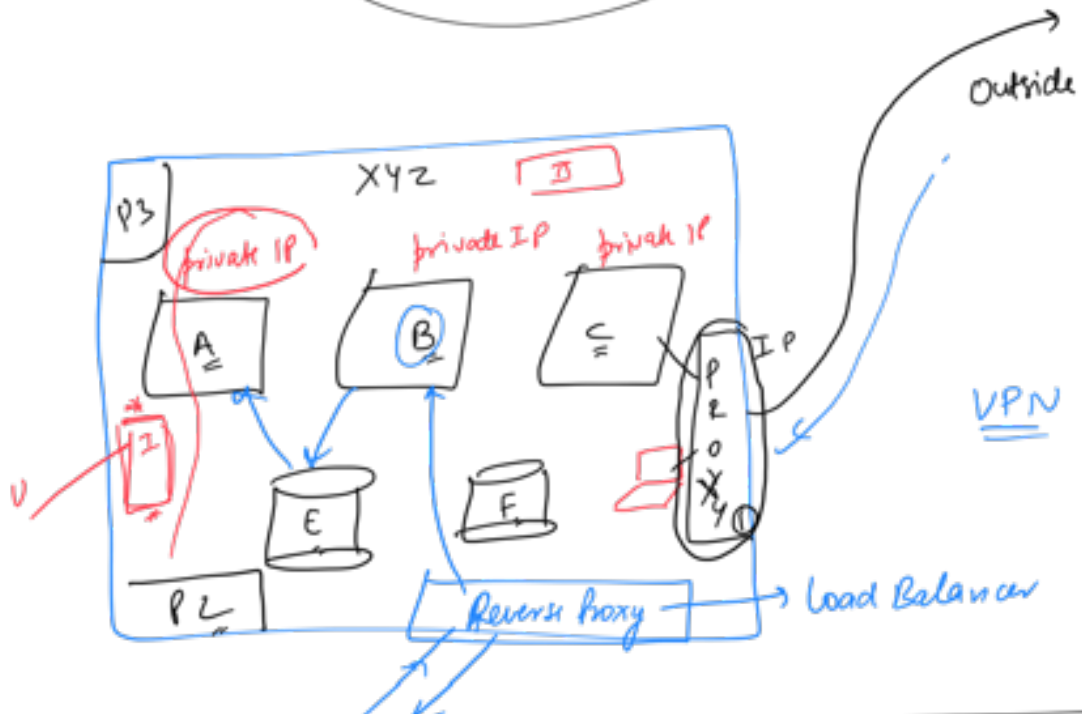


High Level Design

OSI Layers

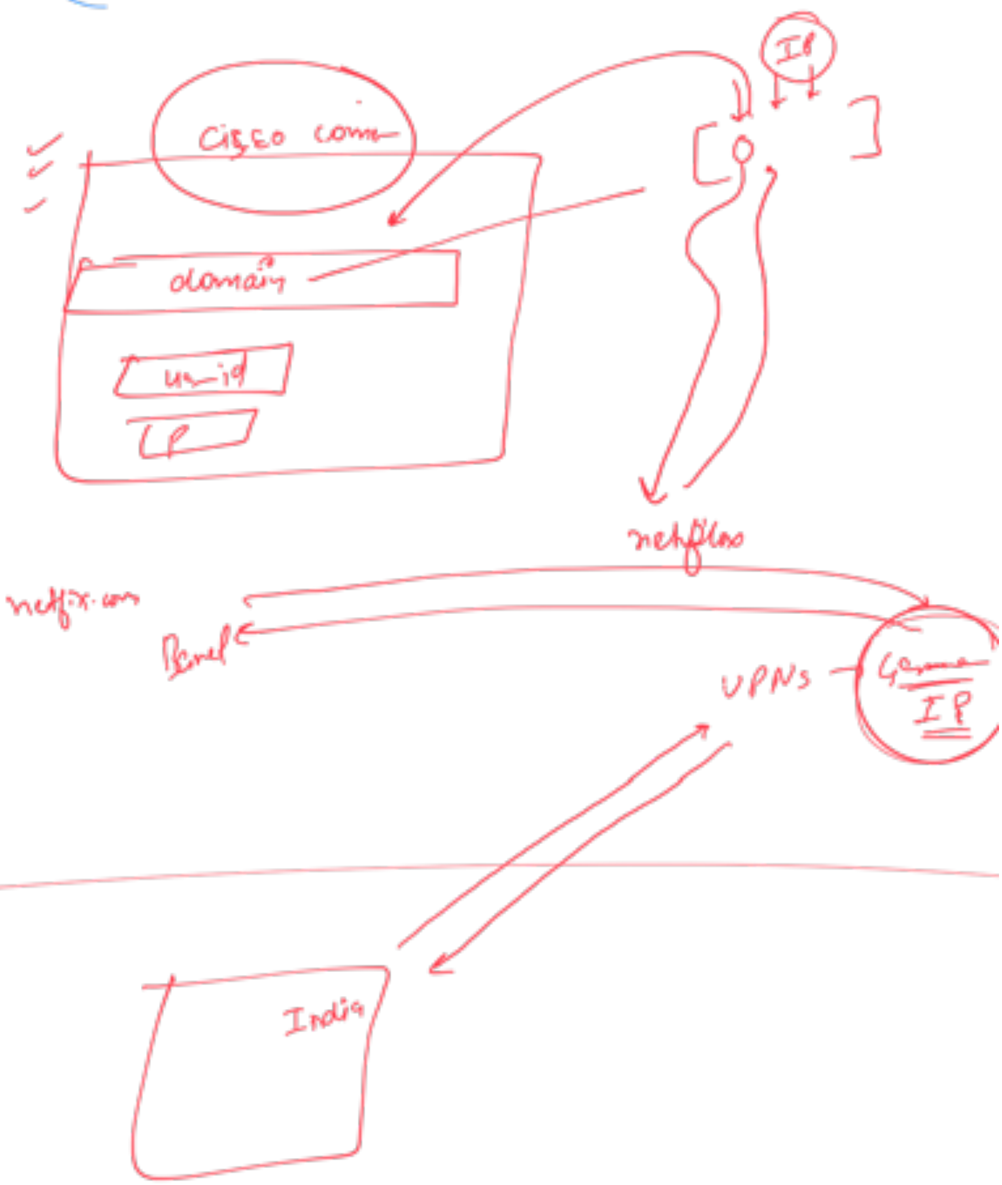
... of network

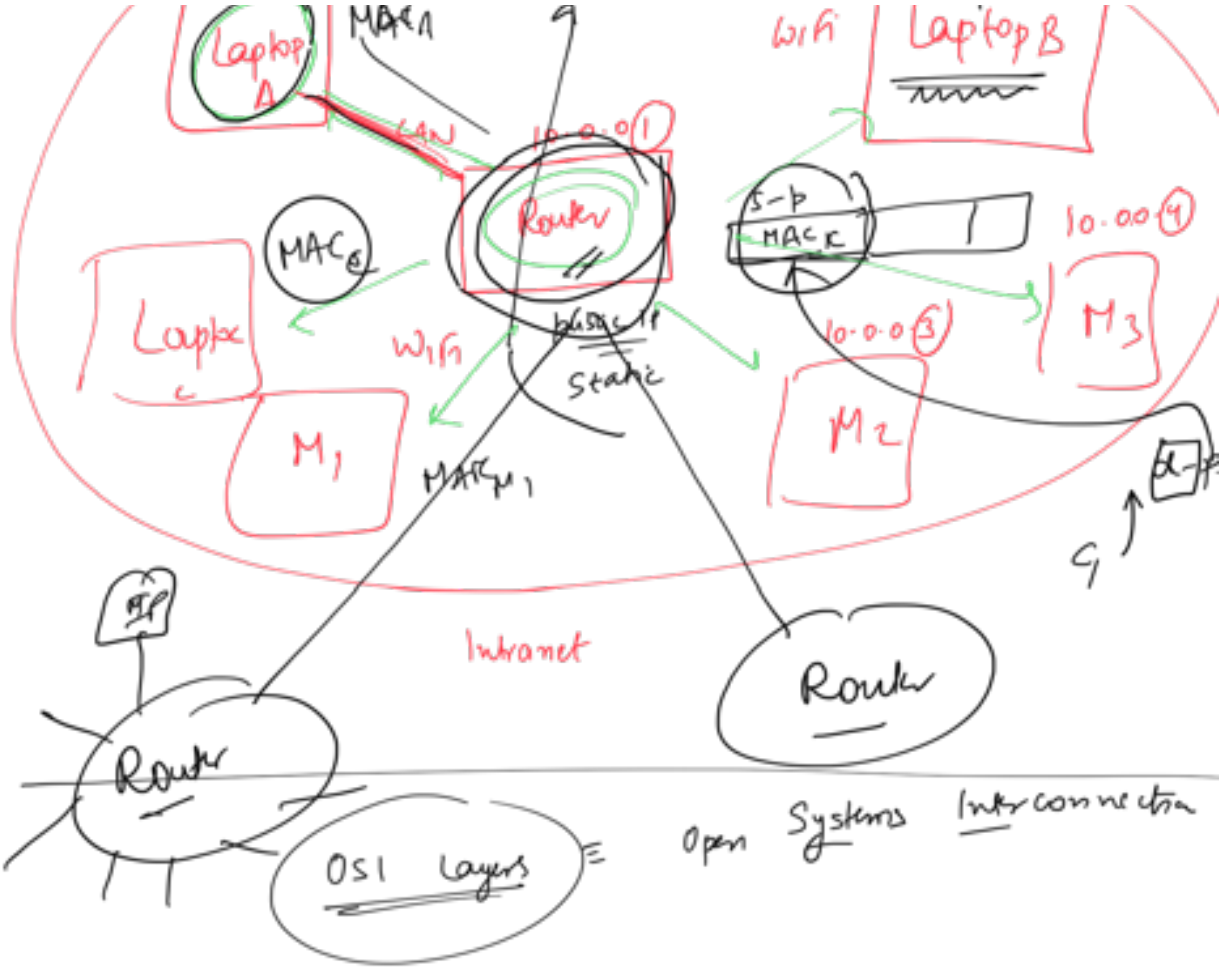

Intranet
INTERNET
Intranet
GATEWAY

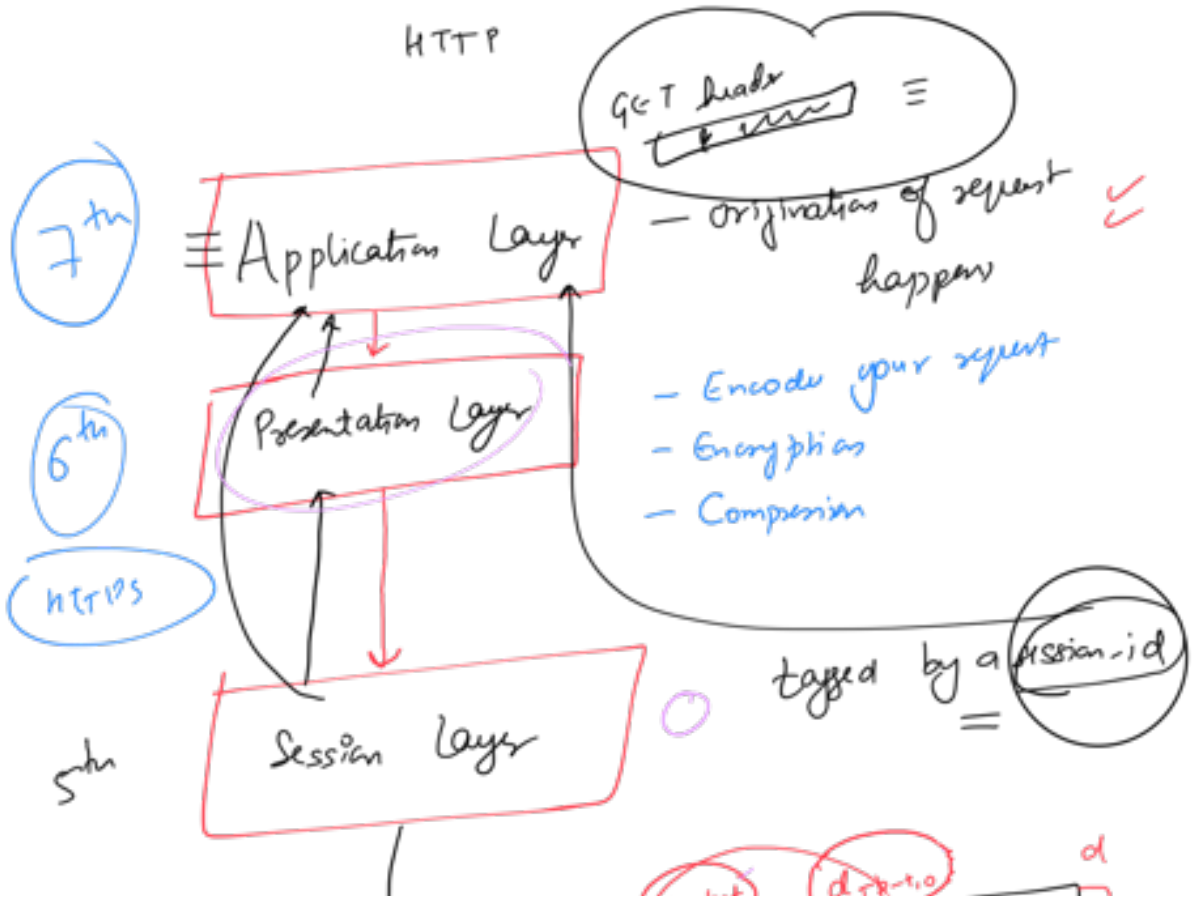

Outside
XYZ
P3
private IP
private IP
private IP
A
B
C
IP
PROXY
VPN
E
F
Reverse Proxy
Load Balancer

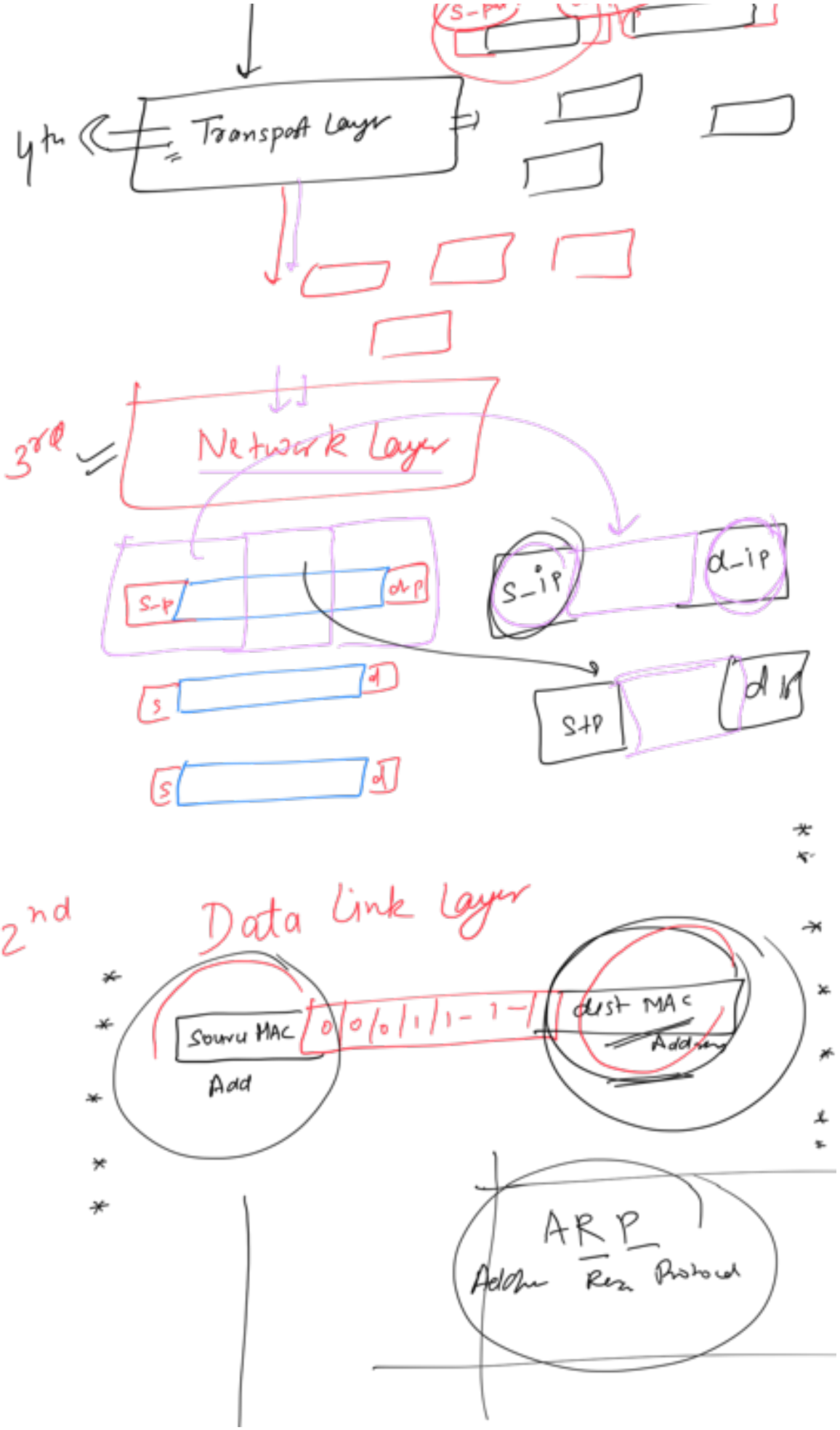
Gateway

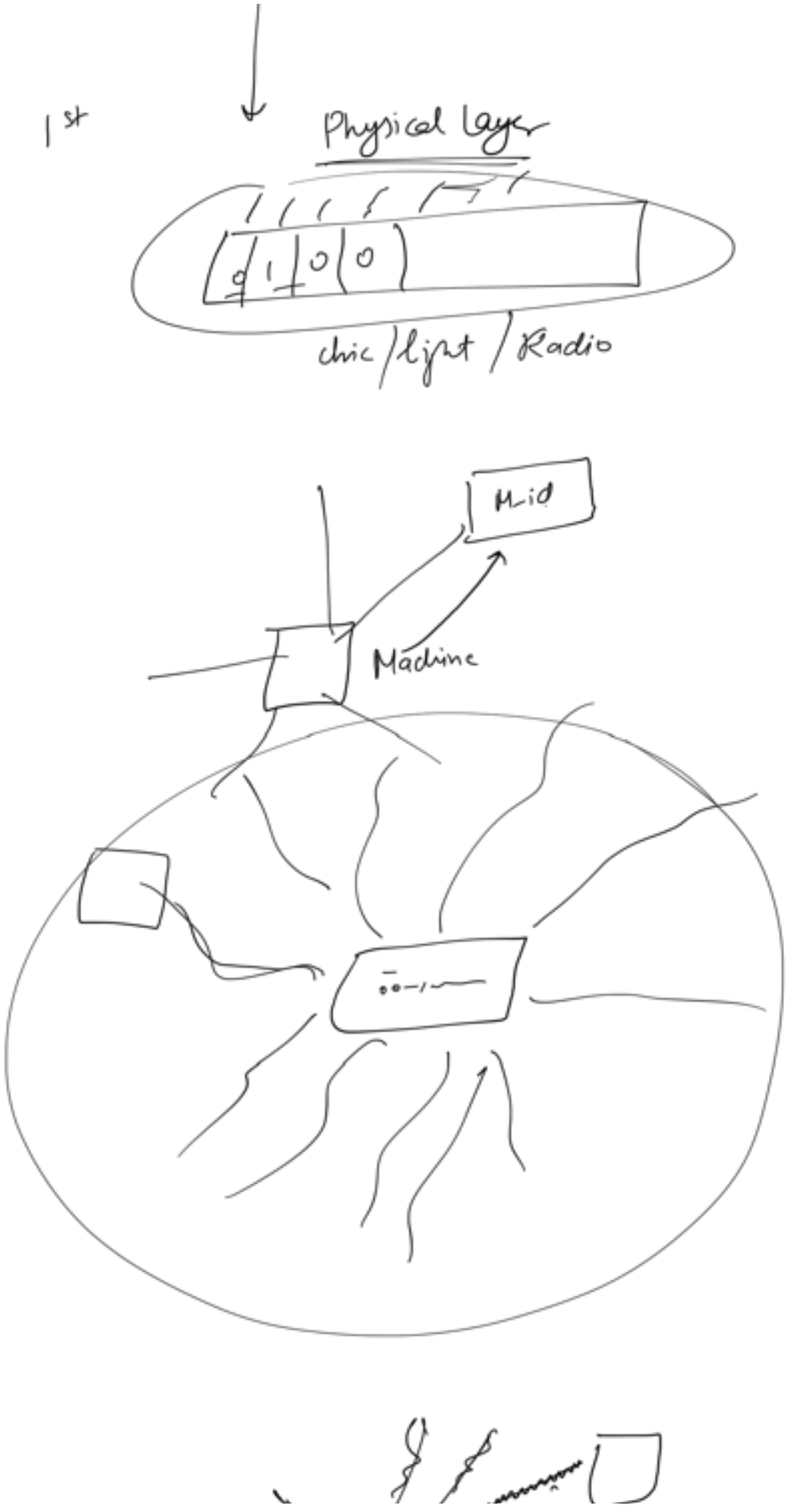
- Proxy ≡ Gateway to the outside world
- Reverse Proxy ≡ Gateway to the inside world

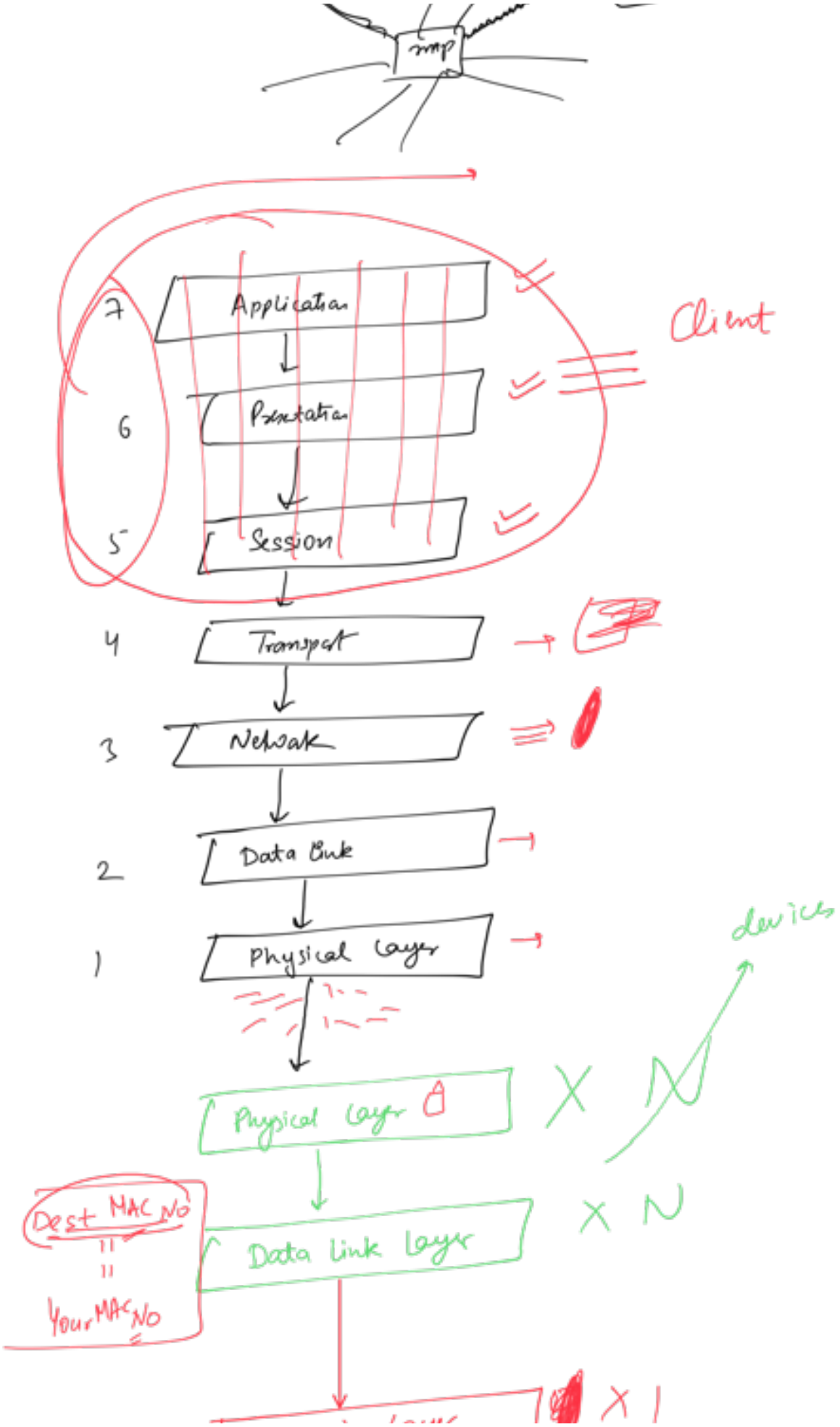


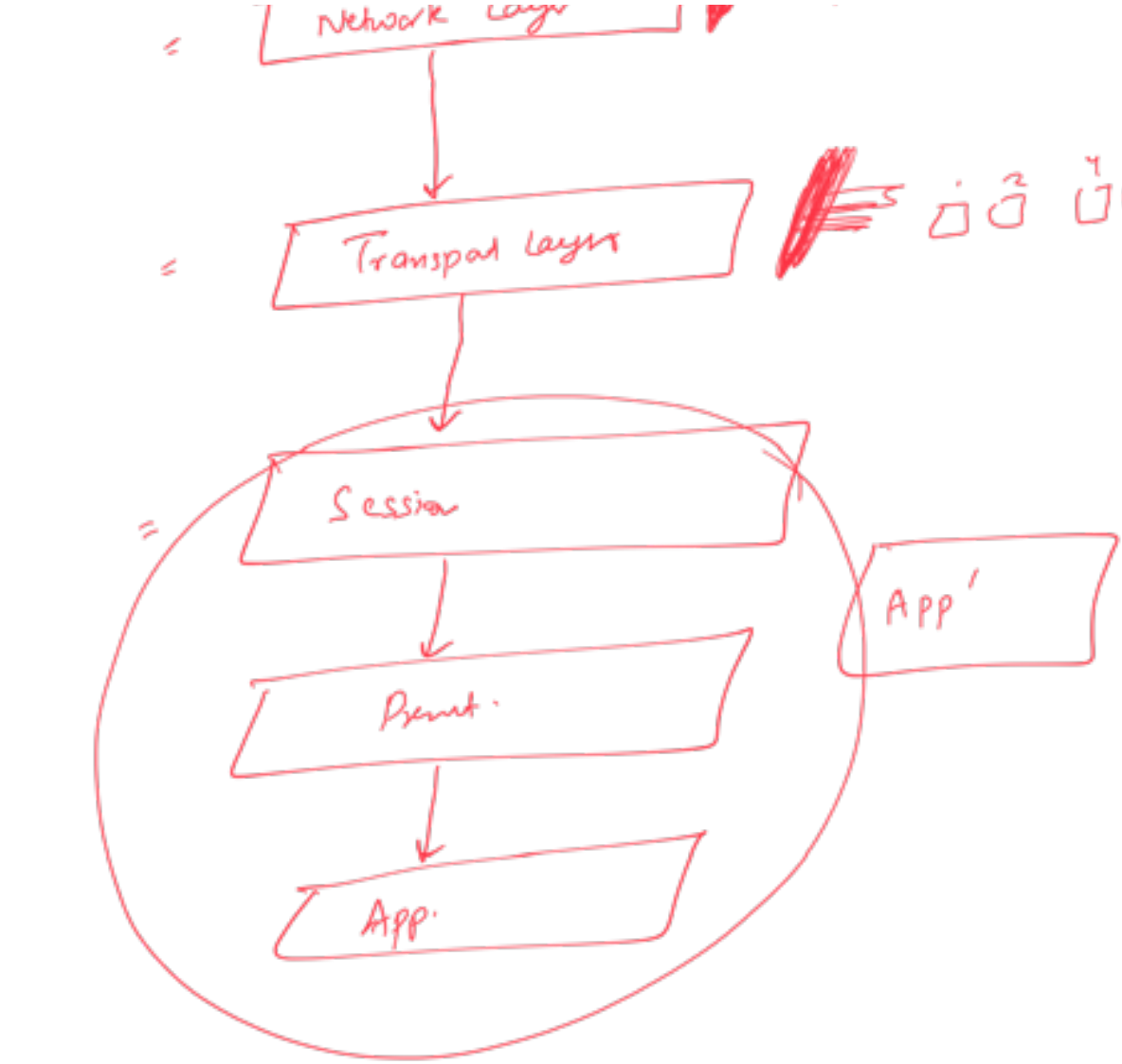

Laptop A
MAC A
LAN
10.0.0.1
Router
WiFi
Laptop B
MAC B
10.0.0.4
10.0.0.3
M3
M2
MAC C
Laptop C
MAC C
WiFi
M1
MAC M1
Static
Intranet
Router
Router
OSI Layers
Open Systems Interconnection

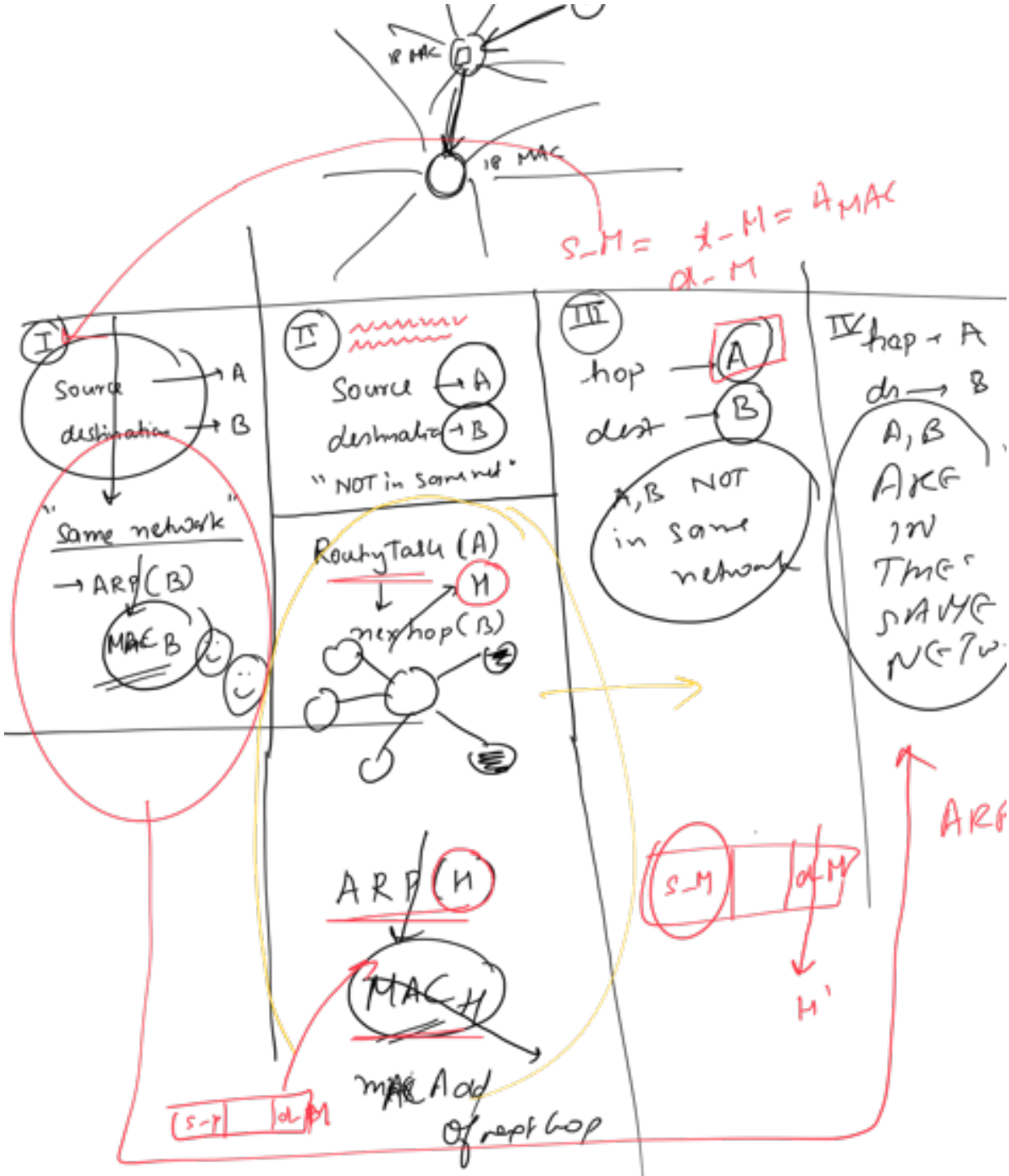

HTTP
GET /index
7th
Application Layer
originates of request happens
Presentation Layer
Encode your request
Encryption
Compression
6th
HTTPS
tagged by a session-id
Session Layer
5th


4th
Transport Layer
3rd
Network Layer
S-P
d-P
S-ip
d-ip
S-P
d-P
S
d
S
d
S-ip
d-ip
S-P
d-P
2nd
Data Link Layer
Source MAC
Add
dest MAC
ARP
Address Res Protocol


1st
Physical layer
1 1 0 0
chic / light / Radio
Mid
Machine


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


Network
Transport layer
Session
App'
Present.
App.


IP MAC
IP MAC
S_M = d_M = A MAC
I
Source
destination
A
B
"Same network"
ARP(B)
MAC B
II
Source
destination
A
B
"NOT in same net"
RoutyTalk (A)
M
nexthop(B)
ARP(M)
MAC M
MAC Add of nexthop
S_M
d_M
III
hop
dest
A
B
A,B NOT in same network
IV hop → A
dr → B
A, B ARE IN THE SAME NETW
S_M
d_M
M'
ARP


2-bit
110
x
y

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

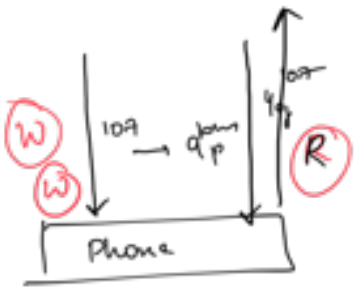
III Partition Tolerance

The system as a whole is allowed to have partitions

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

and the system as a whole will NOT stop functioning

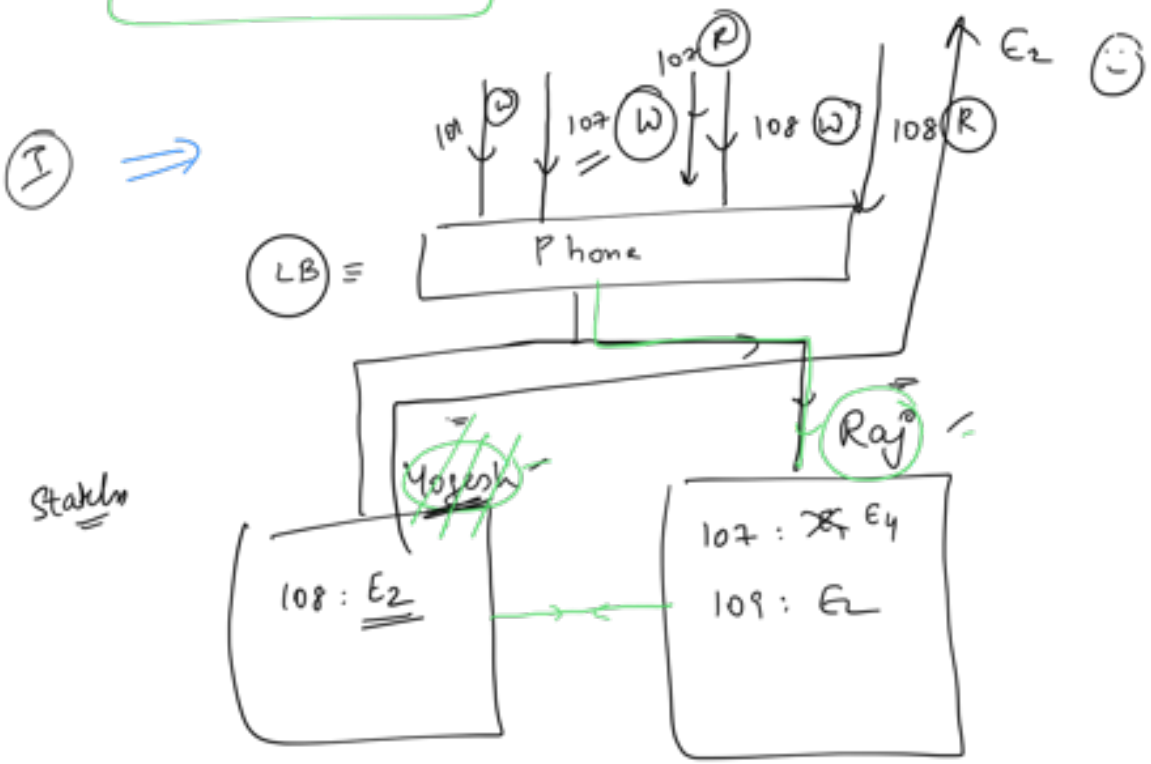
Event Reminder Service



DOES NOT Require Partition Tolerance

Consistent

Available



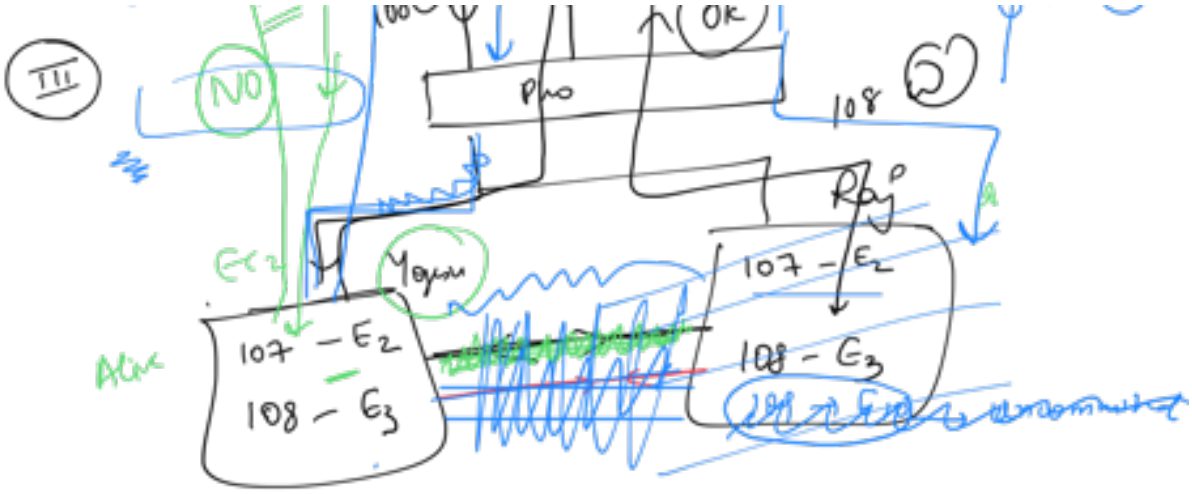
Partition Tolerance

NOT CONSISTENT

2/3

Available

NOT Available



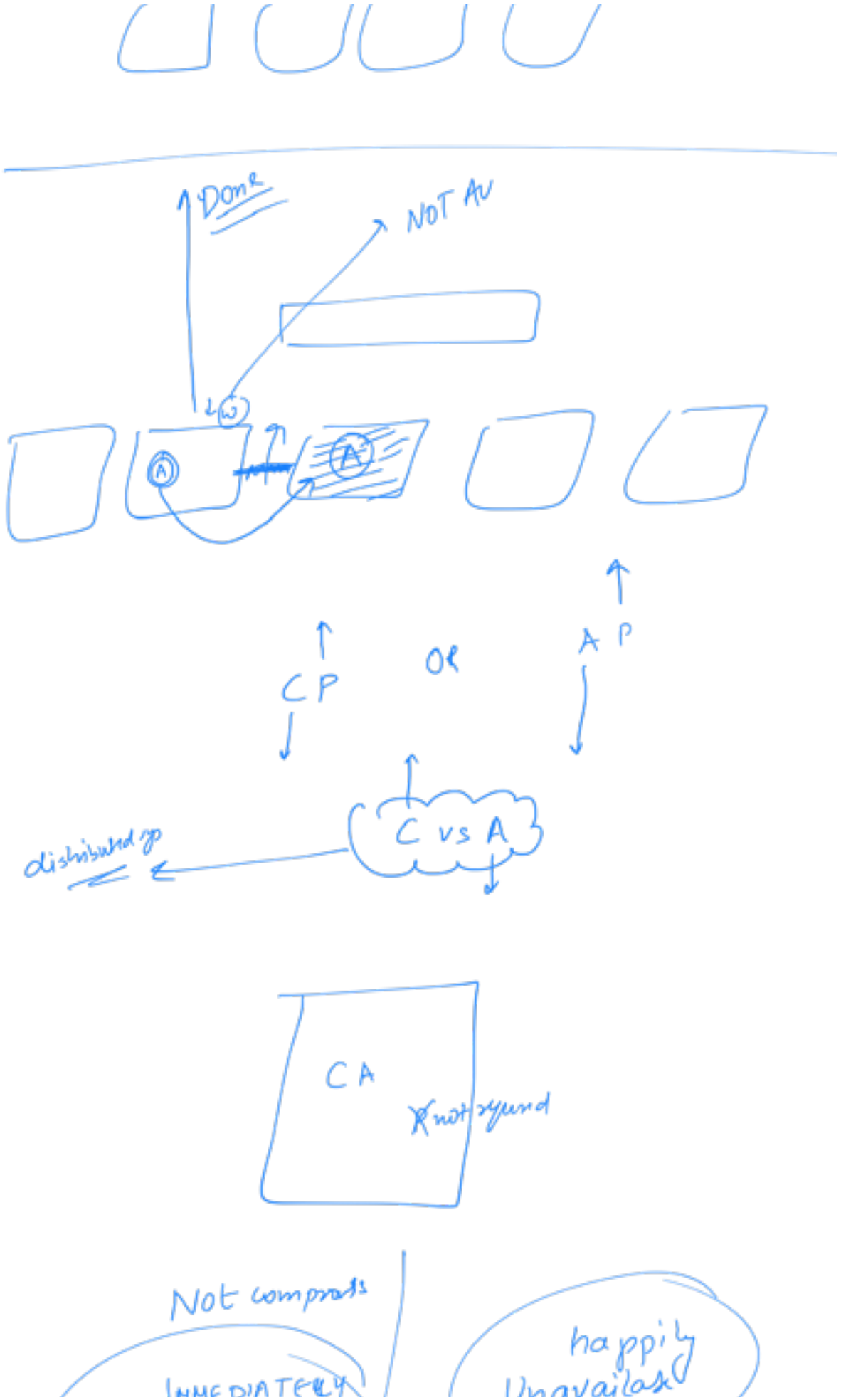
CONSISTENT

PARTION TOLERANT

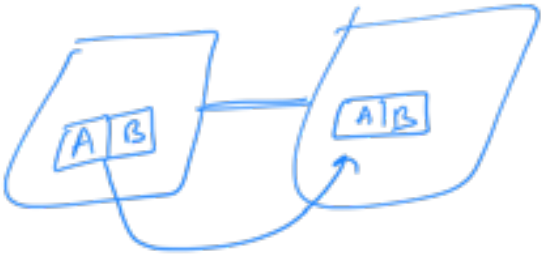
NOT AVAILABLE

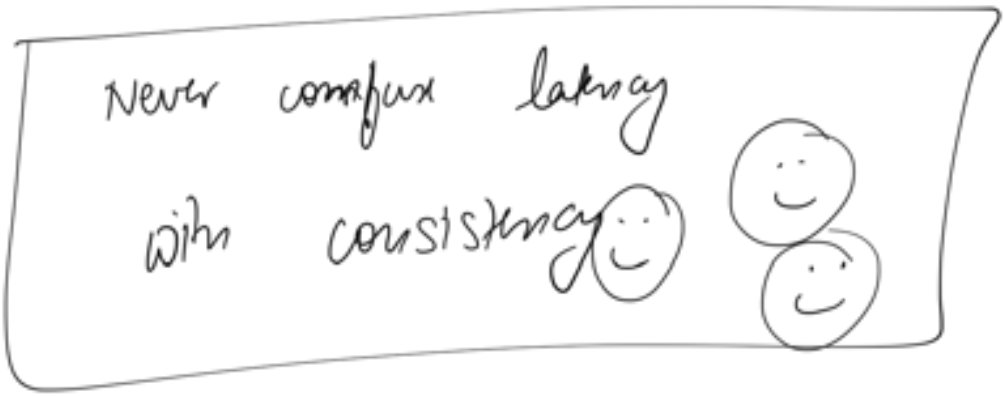
CP OR AP

- CA - if you are doing vertical scaling

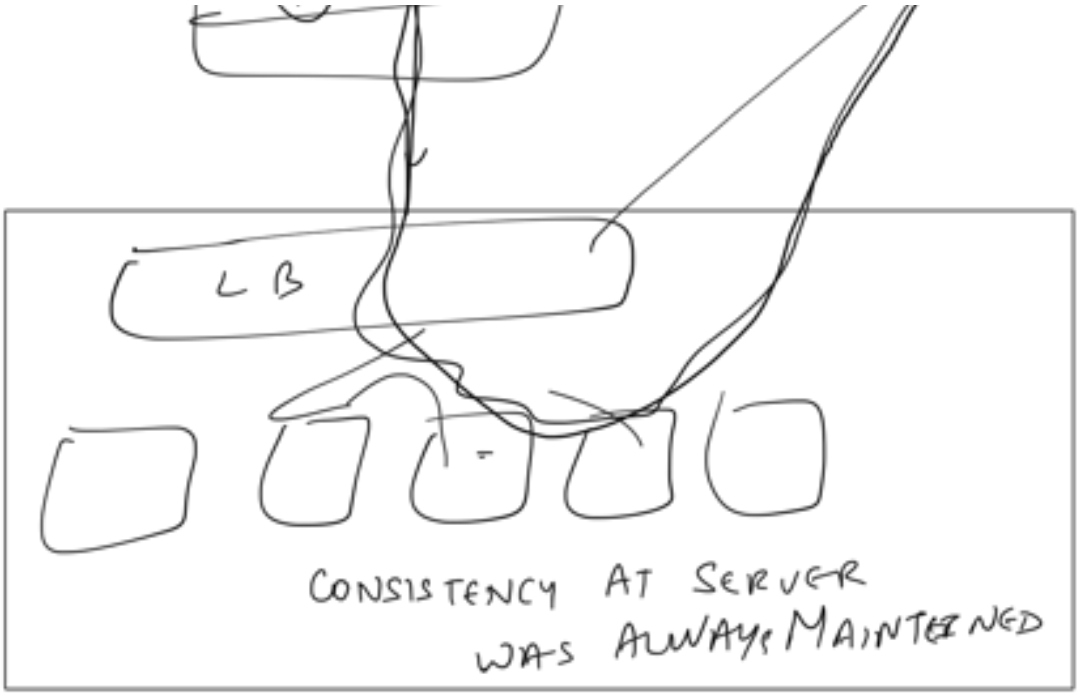

Done
NOT AV
W
A
A
CP
OR
AP
C vs A
distributed sp
CA
not required
Not compromise
IMMEDIATELY
happily
Unavailable

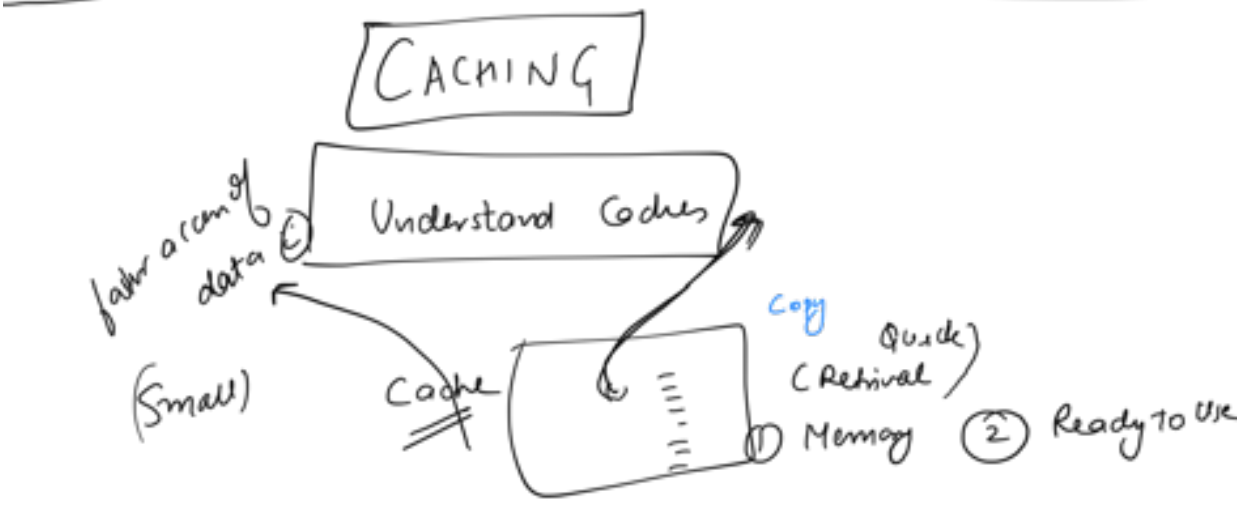
120mm
60mins

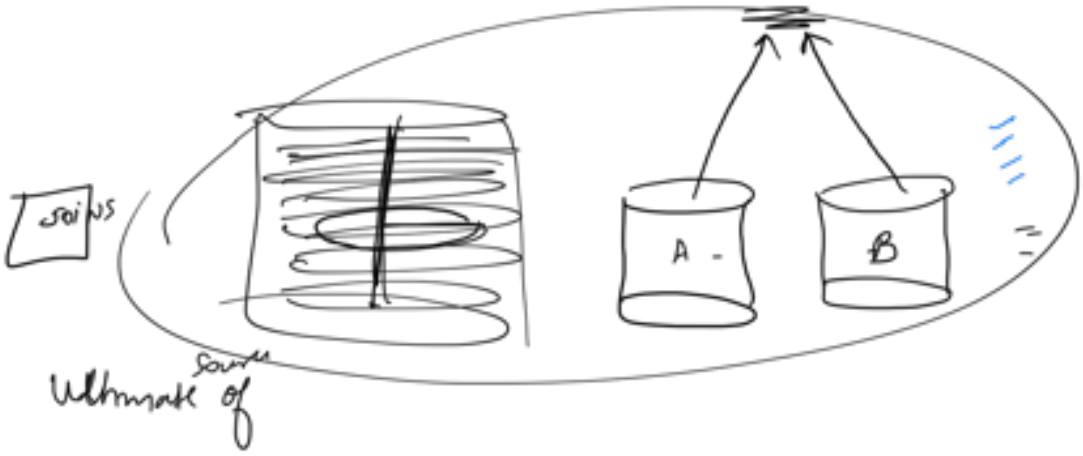


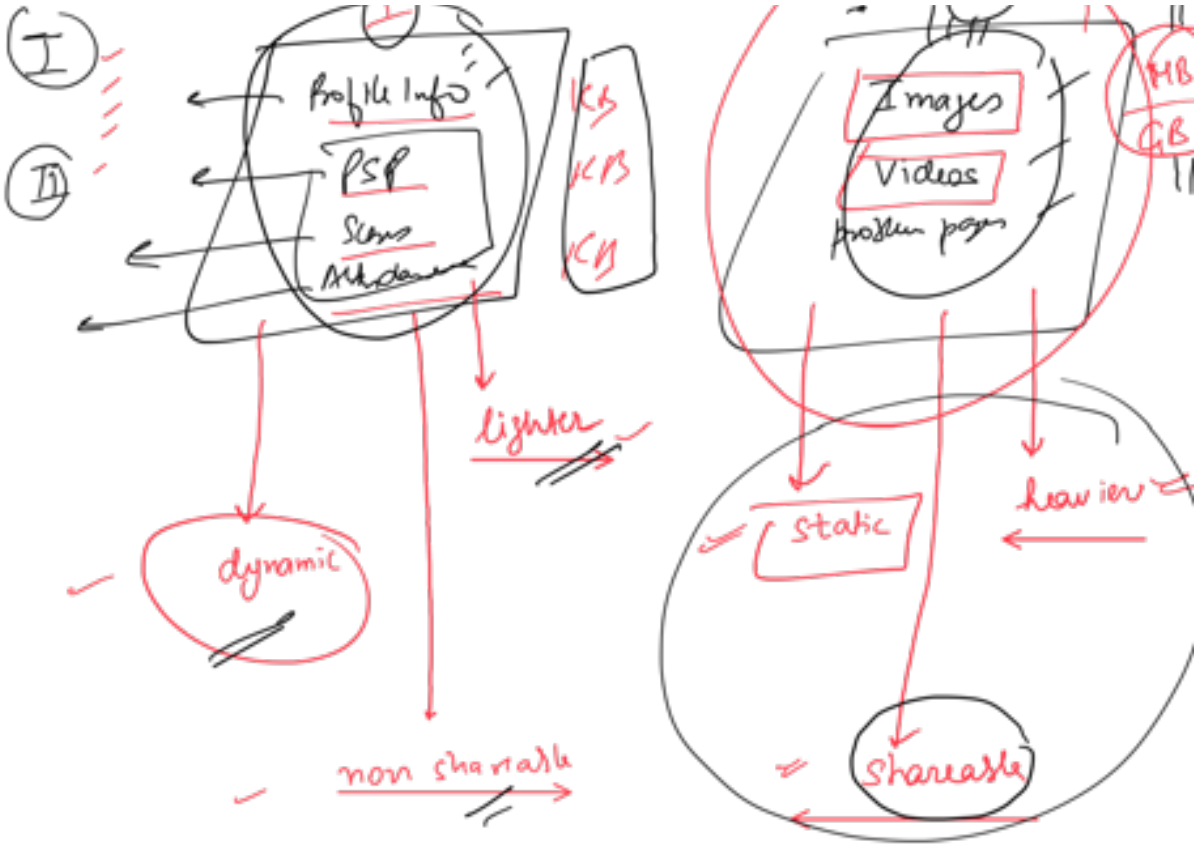


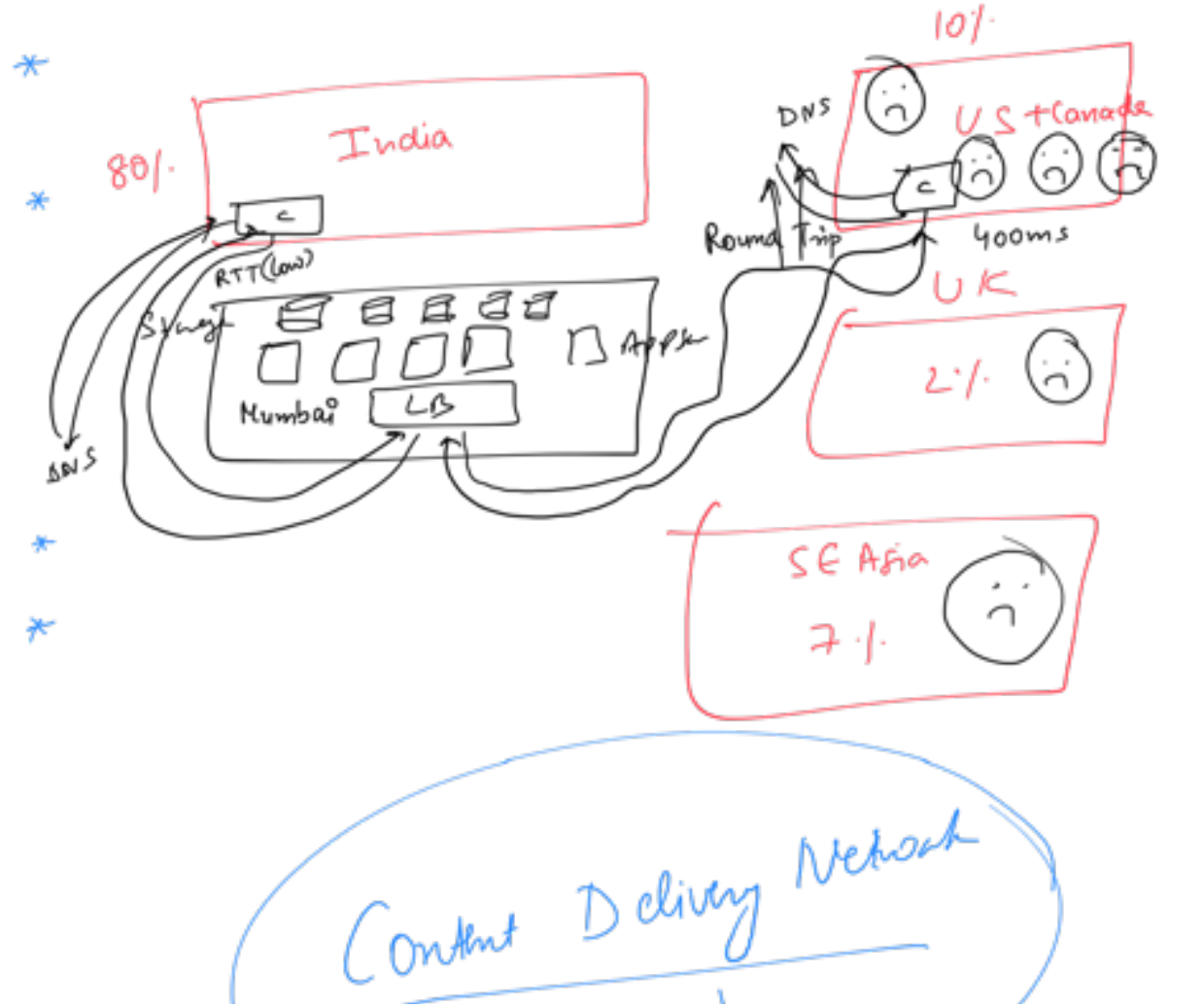
Mobile


LB
CONSISTENCY AT SERVER
WAS ALWAYS MAINTEINED

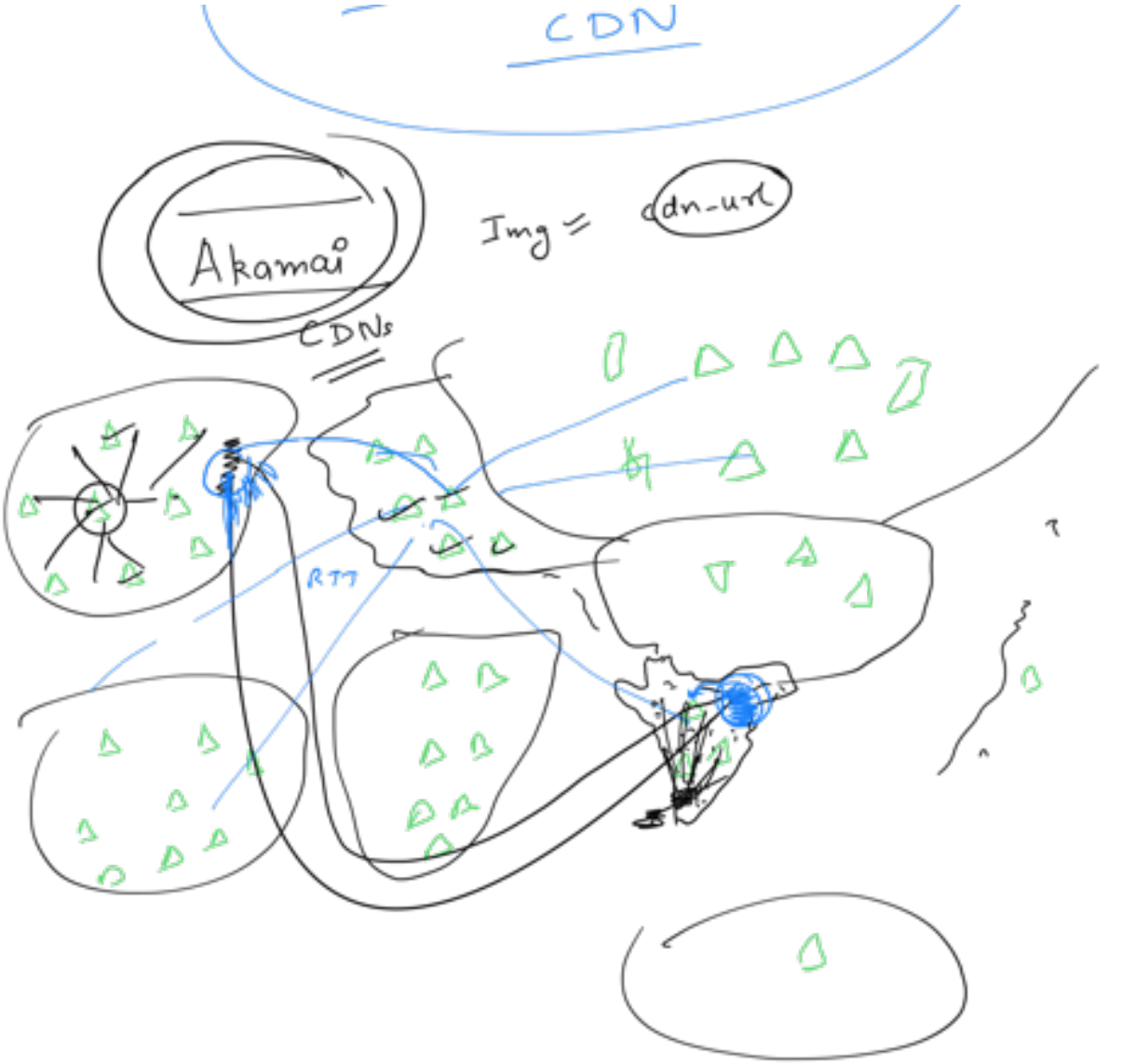

CACHING
Understand Codes
Copy
Quick
Retrival
(Small)
Cache
Memory
2
Ready to Use

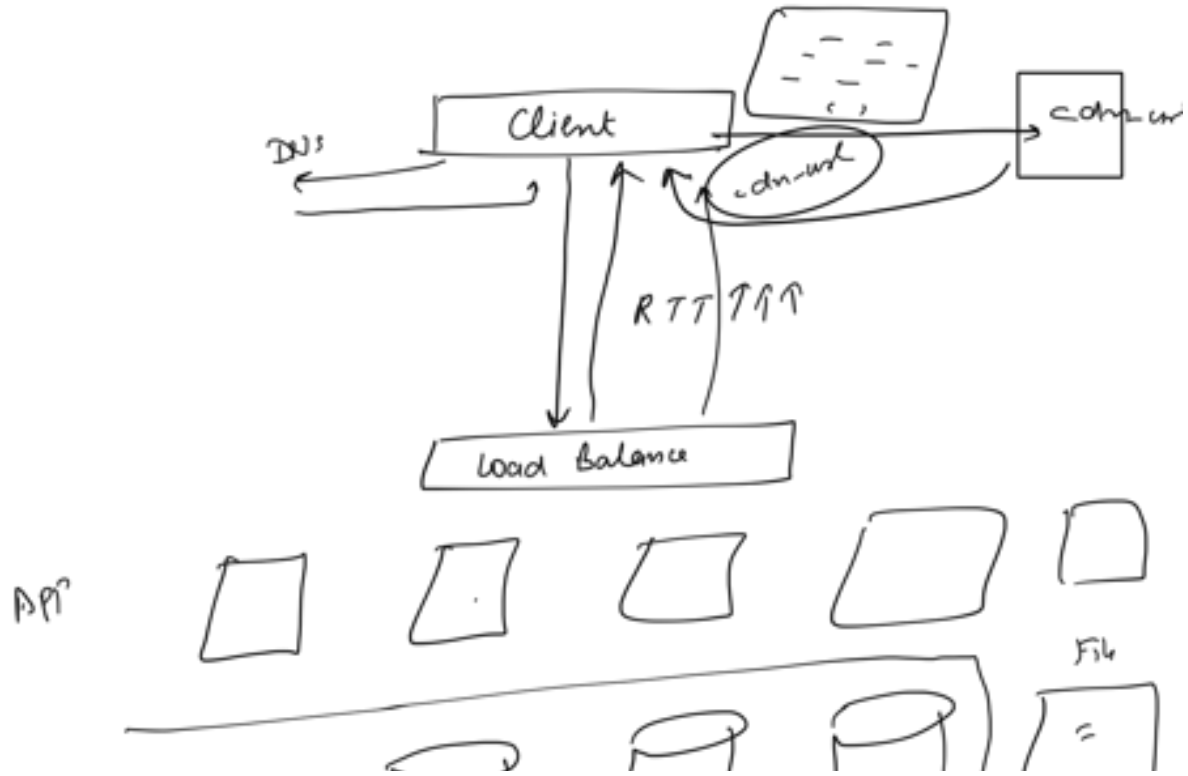

A
B





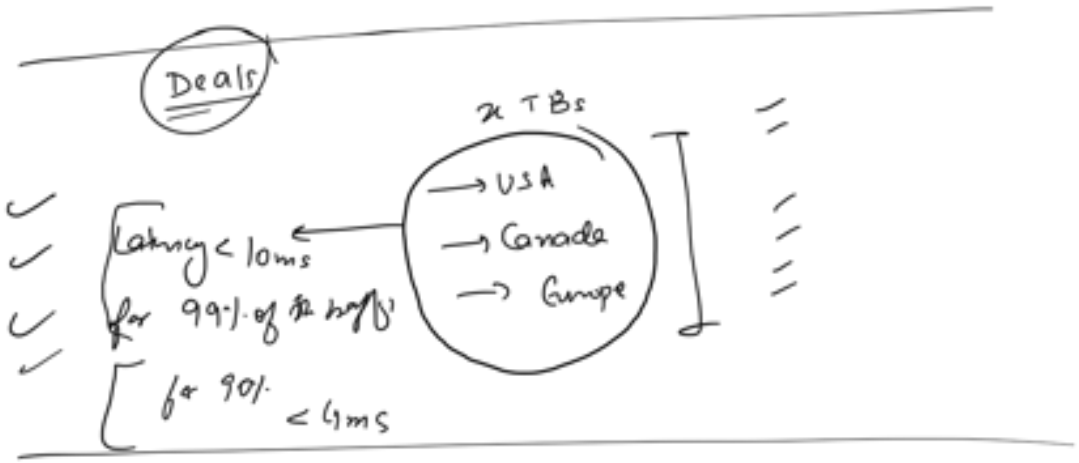
Content Delivery Network


CDN
Akamai
CDNs
Img =
dn-url
RTT


Client
DNS
cdn-url
RTT
Load Balance
API
File

img ⟷ cdn_url

- network is NOT getting checked
- latency solver

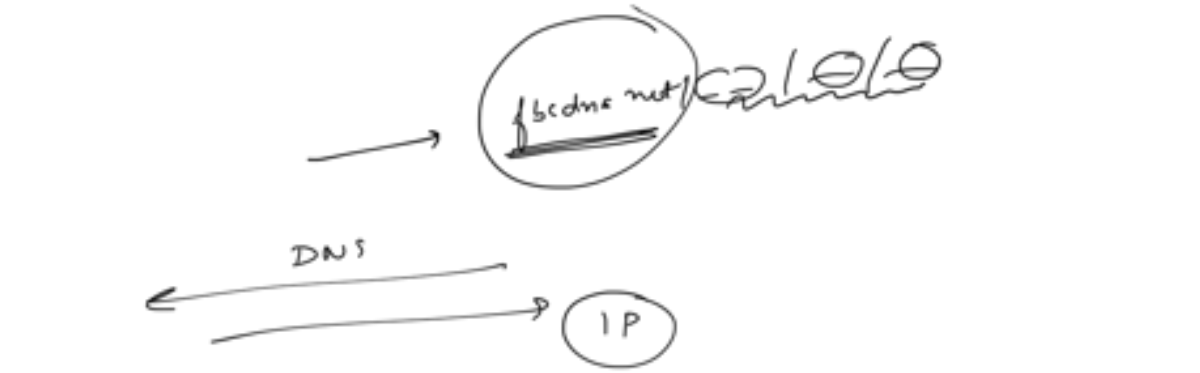


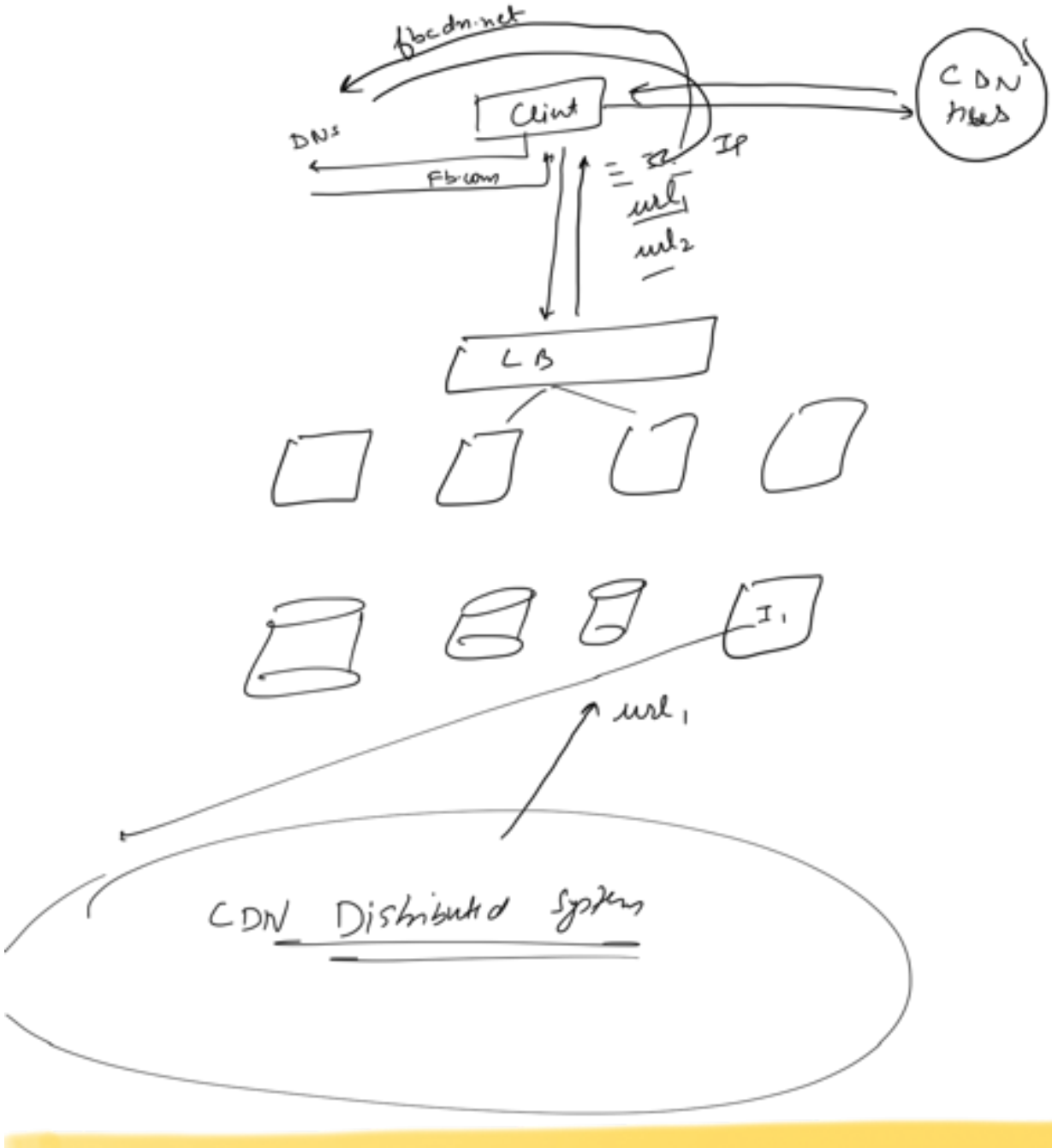
Company → CDN

CDN Hub → CDN systems

User → CDN

cdn


fbcdn.net
DNS
IP


fbcdn.net
Client
CDN
DNS
IP
LB
I1
url1
CDN Distributed System

Cache Invalidation

(A) → Time to Live (TTL)
= url should automatically become void after this time

(B) → Start sending a new URL

(C) → Versioning

Privacy in CDNs

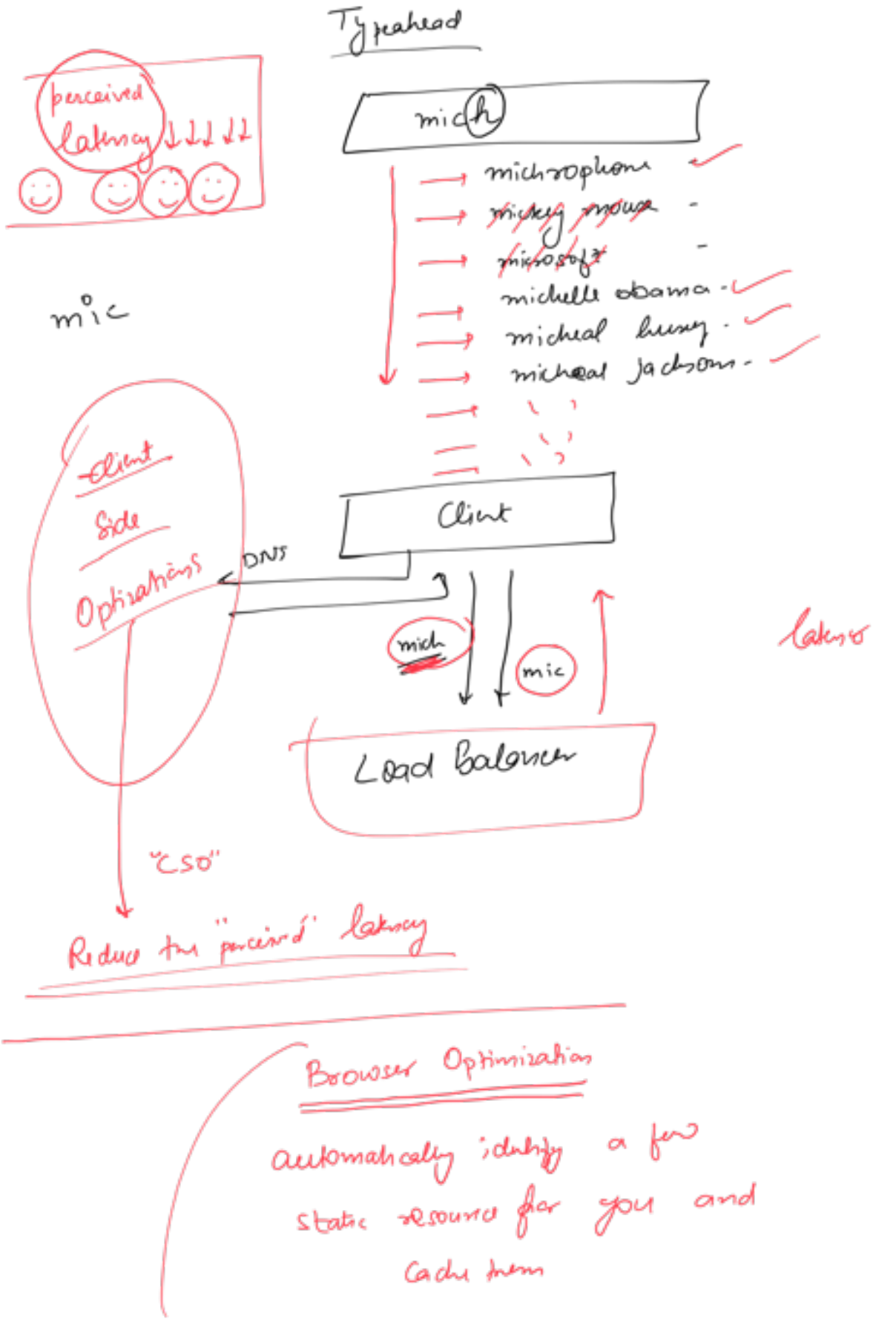
1. CDN urls are by design not guessable
2. encrypted format
3. Authentication

---

(2) Client Side Caching

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

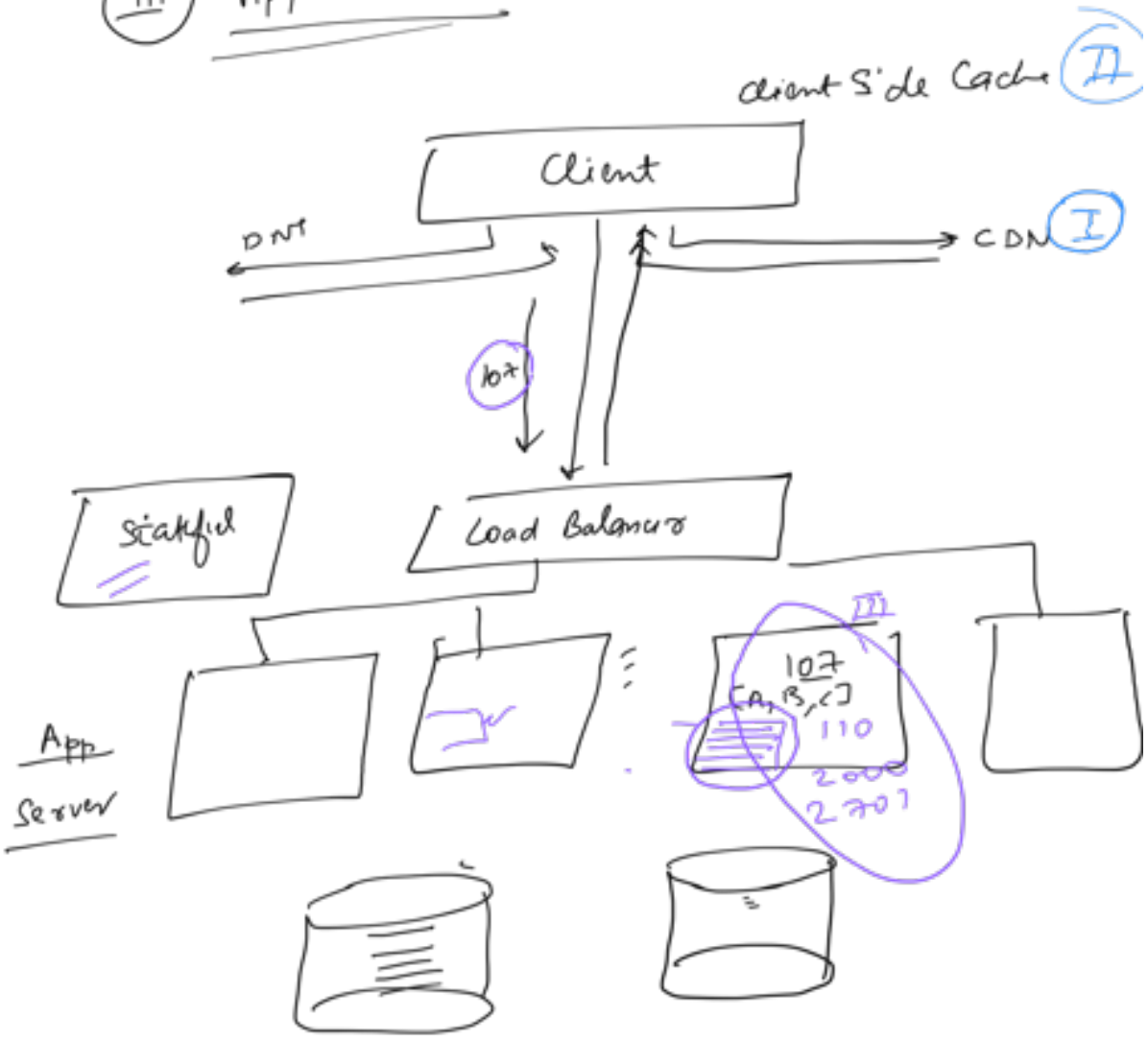
Browser
Mobile App
Desktop App


Typeahead
perceived latency
mich
microphone
michelle obama
micheal bluey
micheal jackson
mic
Client Side Optimizations
Client
DNS
mich
mic
latency
Load Balancer
"CSO"
Reduce the "perceived" latency
Browser Optimization
automatically identify a few static resource for you and cache them

Hard Reload / Forced Relod

'ctrl + shift + R'

III App Server Cache



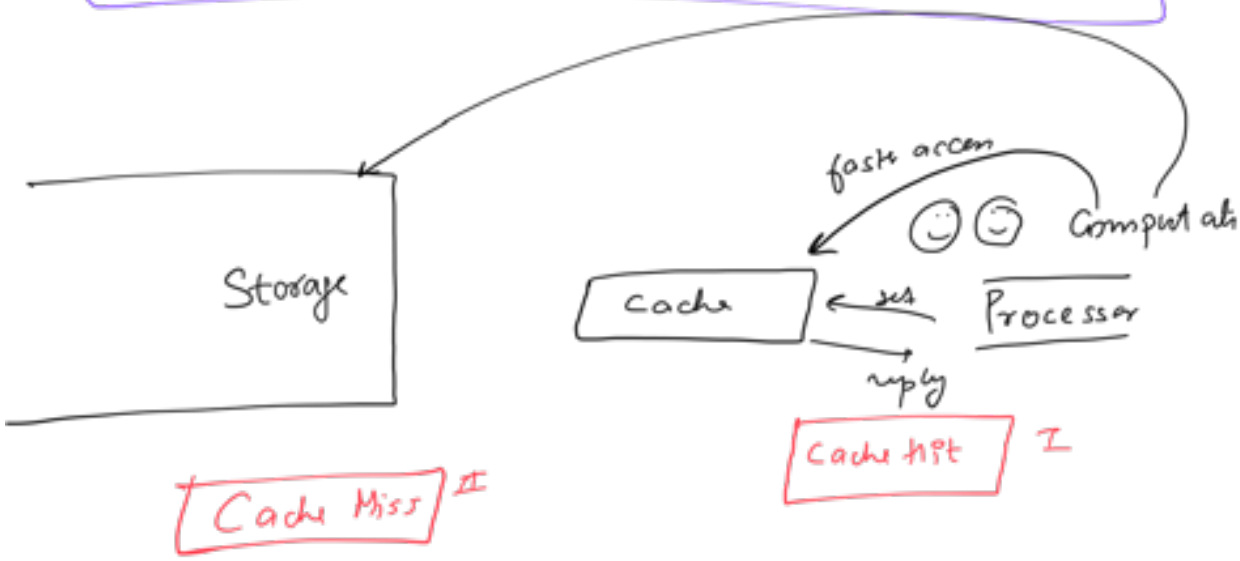
Cache Replacement

LRU → Least Recently Used

(A)

(B) MFU

(C)



Access Time for Cache = X

" " DB = 100X

k calls

Without Cache

Time = k X Avg DB Access Time + k X Avg Network Access Time

80-20

With Cache

70% C Hit Rate

[30% Cache Miss Rate]

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

+

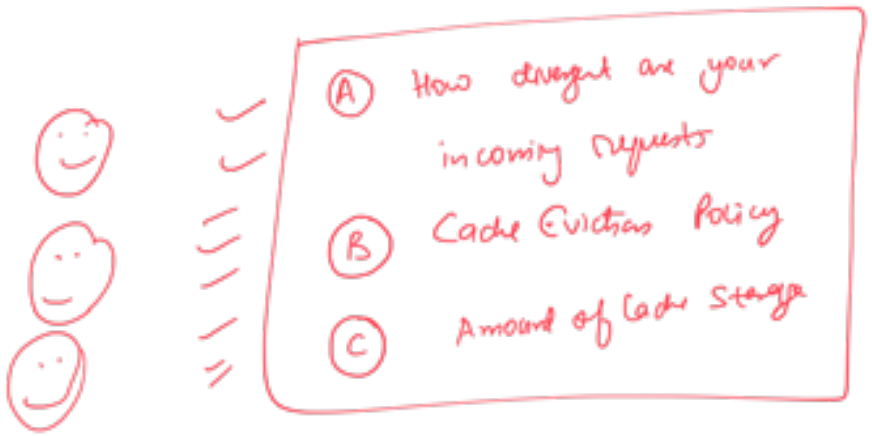
$\left[\frac{30}{100} \times k\right]$ X Cache Access Time

+

0.3k X DB Access Time

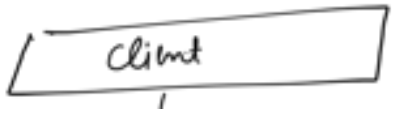
+

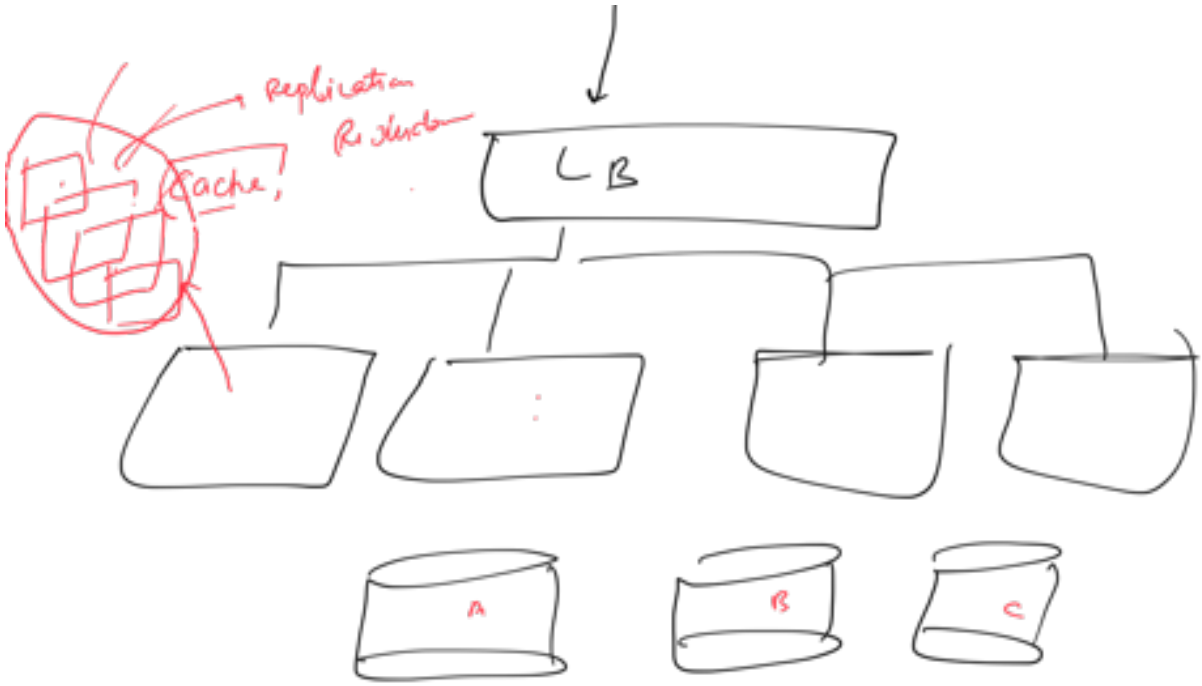
0.3k X Network Time

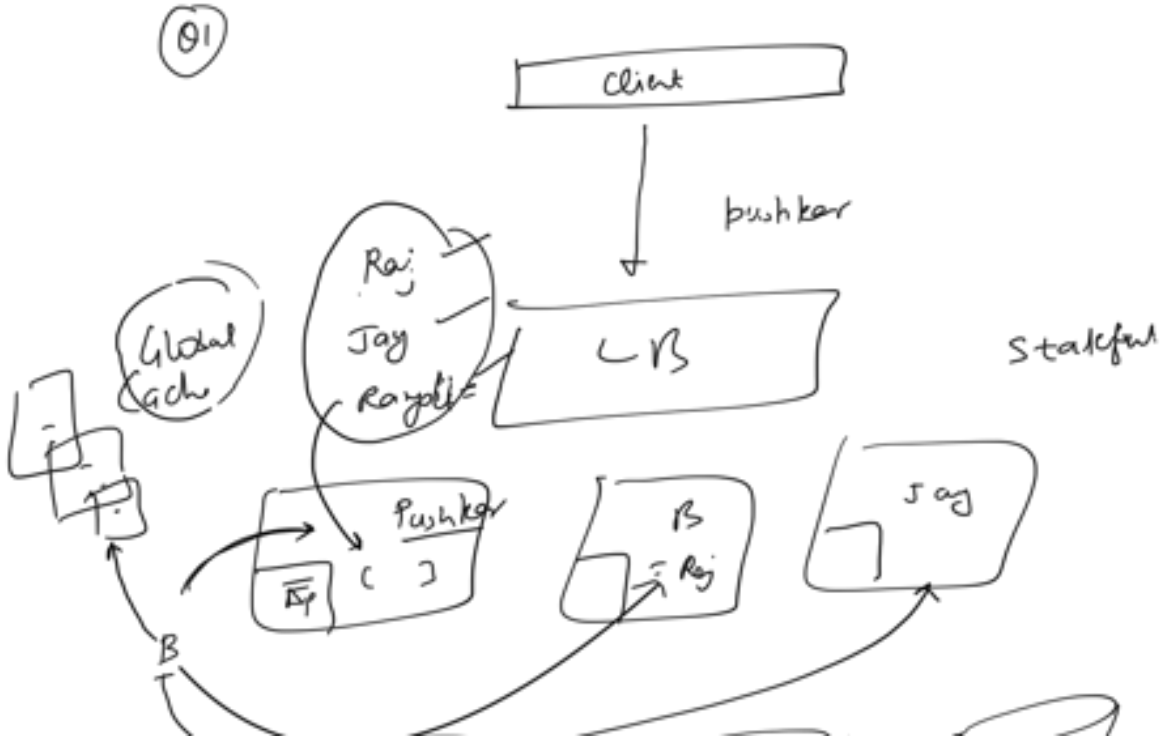


---

(4) Global Cache




Replication
Cache
LB
A
B
C


01
Client
pushkar
Raj
Jay
Global Cache
LB
Stateful
Pushkar
B
Jay
B