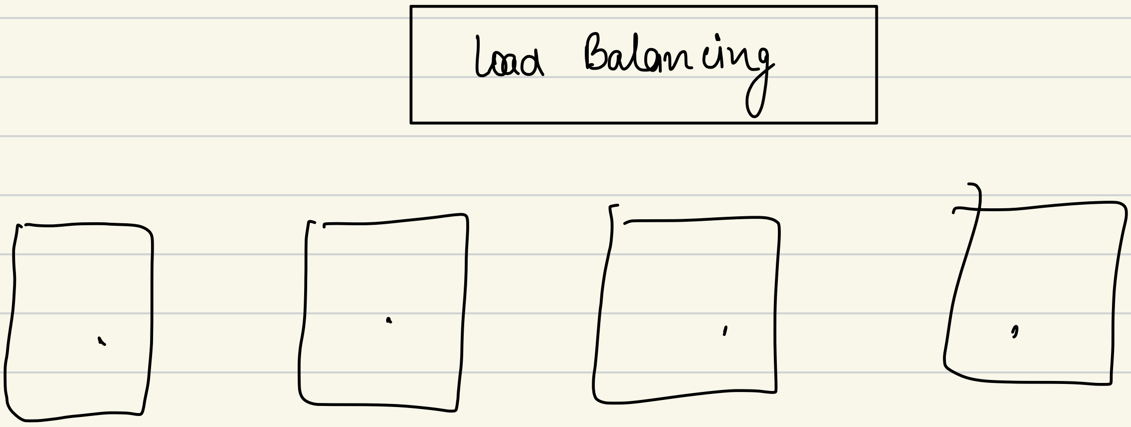
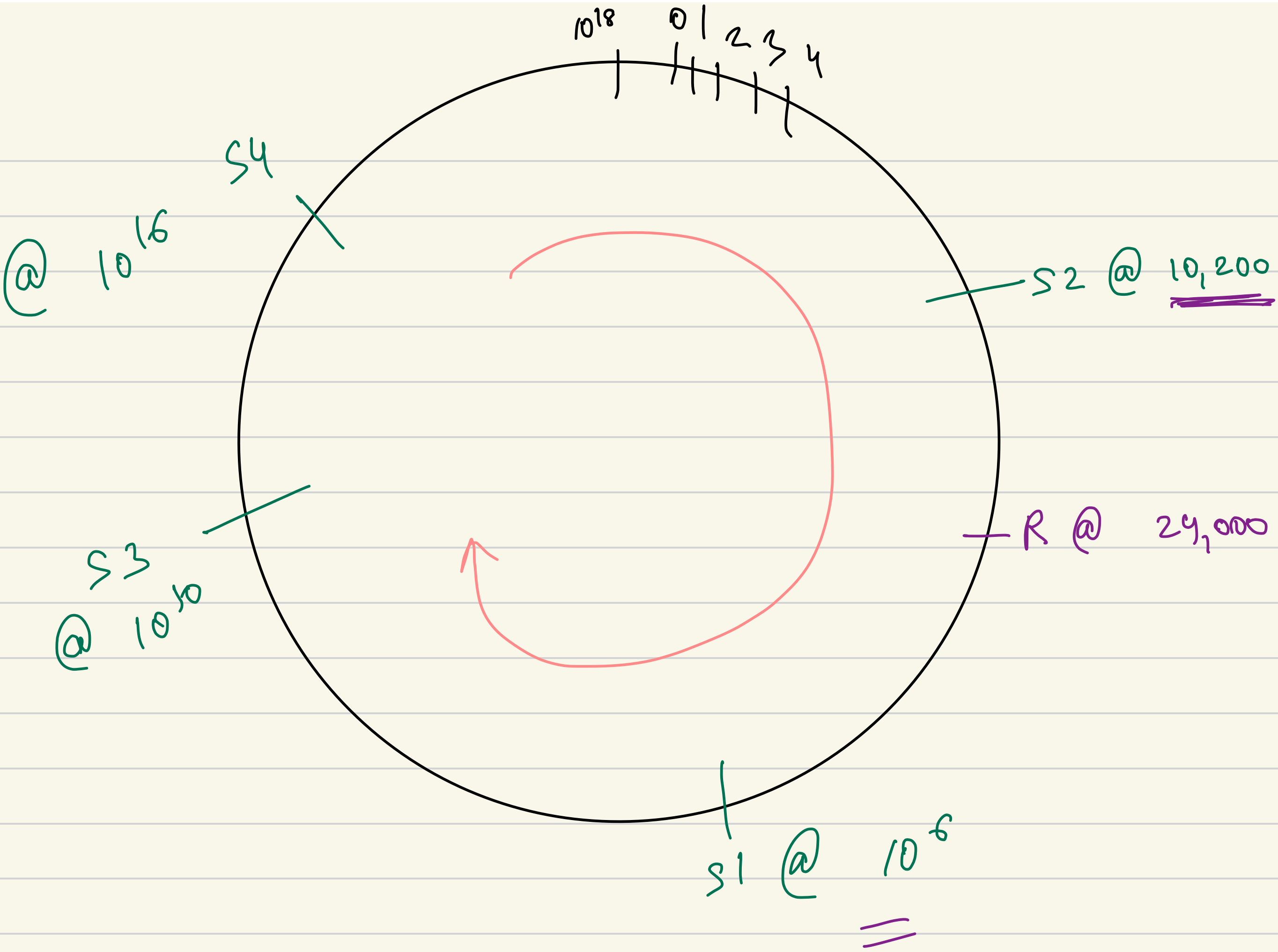
22/Nov/2023

Consistent hashing

Stateful LB

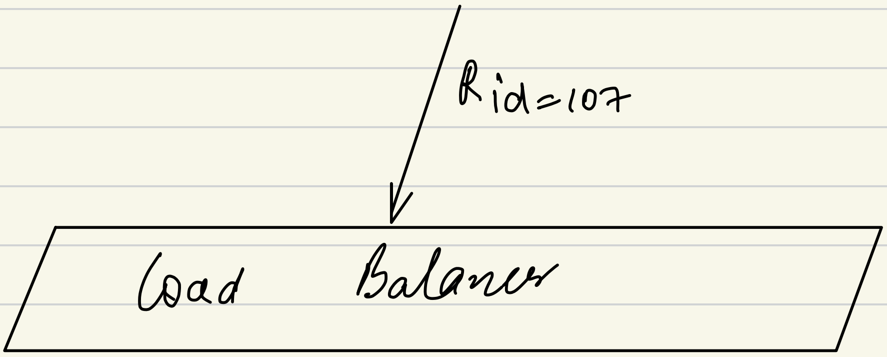




sorted array of server positions

A =

10,200	10^6	10^{10}	10^{16}
S2	S1	S3	S4

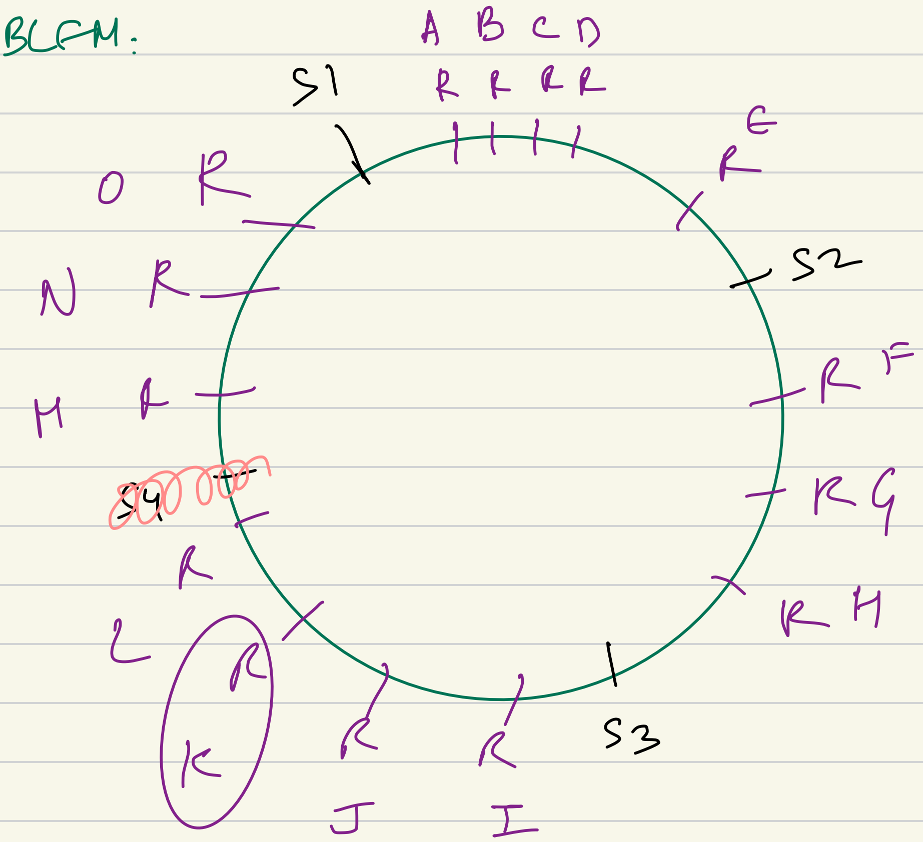


Step 1 : $H_R(107) = 24,000$

Step 2 : Binary Search (A) to find ceiling ☺ ☺

$\log(|S|)$ time ☺

PROBLEM:



S2 ↔

S3 ↔

S1 ↑↑

Now S4 dies ☹ ☹

With this approach of C.H, we have solved the problem of asking every state to move,

BUTTTTT....

We are being unfair with re-distribution

Now S1 dies ☹ ☹

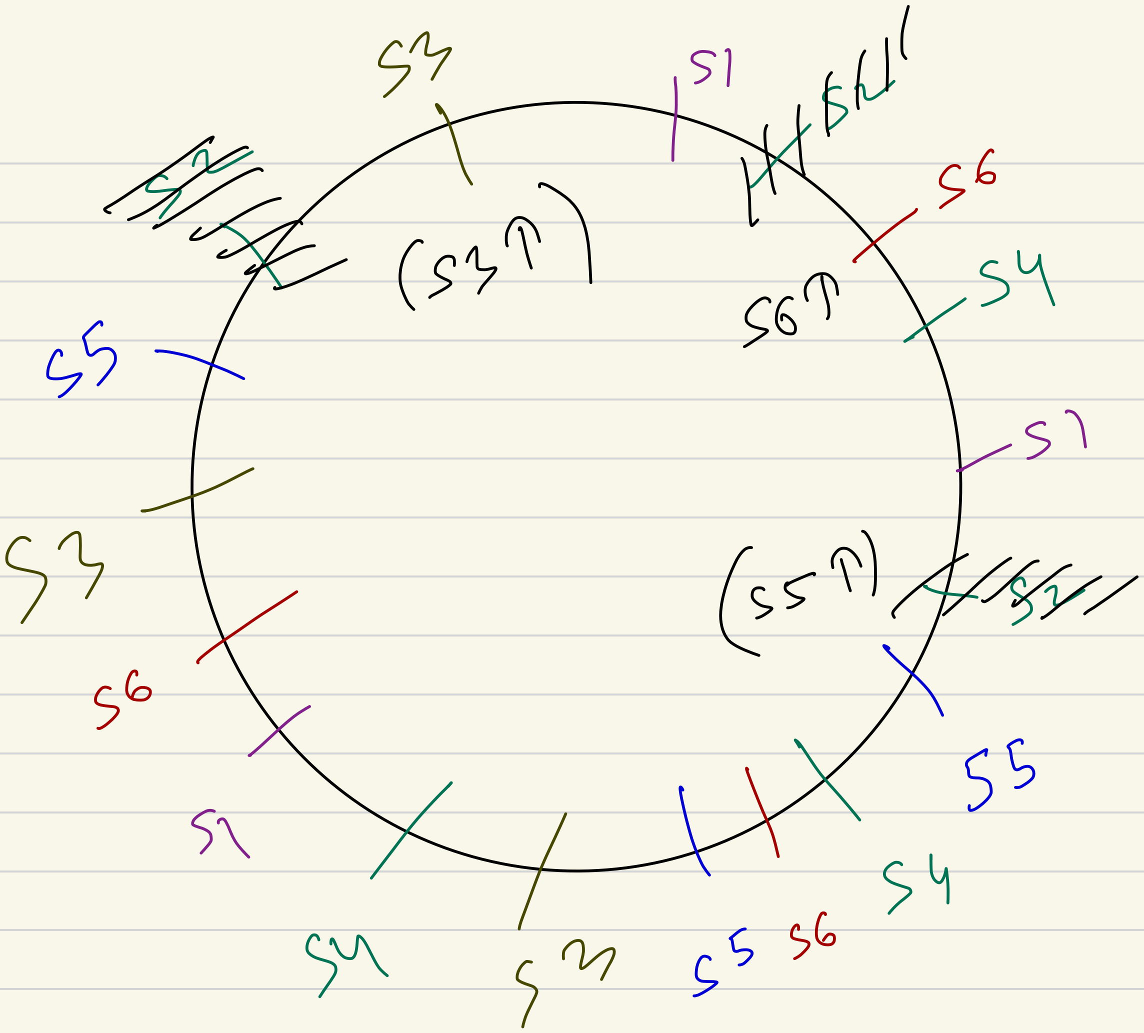
S2 ↑↑↑

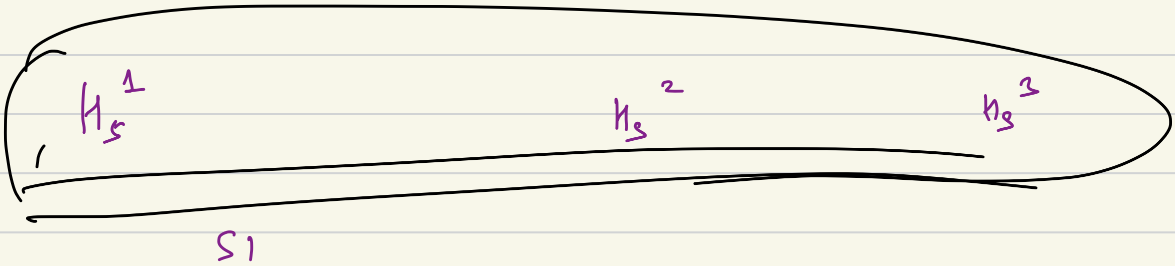
S3 ↔

Cascading Failure 😐 😐 😐

(domino falling)

ripple failure





$H_s^1(S1) = \text{value}$

$H_s^2(S_1) = \text{value 2}$

$H_s^3(S_1) = \text{valu3}$

10	20	32	48	60	72	80

NGI (49) ↓ ☆

$H_R(R_{107}) = 49$

C.H Redis

Kafka

etc

etc

$H_R(R_{111}) = 950$

$H_R(R_{107}) = \underline{650}$

NGI ⇒ S1

A =

100 S1	500 S2	600 S3	700 S1	800 S5	900 S4	1000 S1	2000 S7

S1 dies

$H_R(R_{107}) = 650$ NGI ⇒ S5

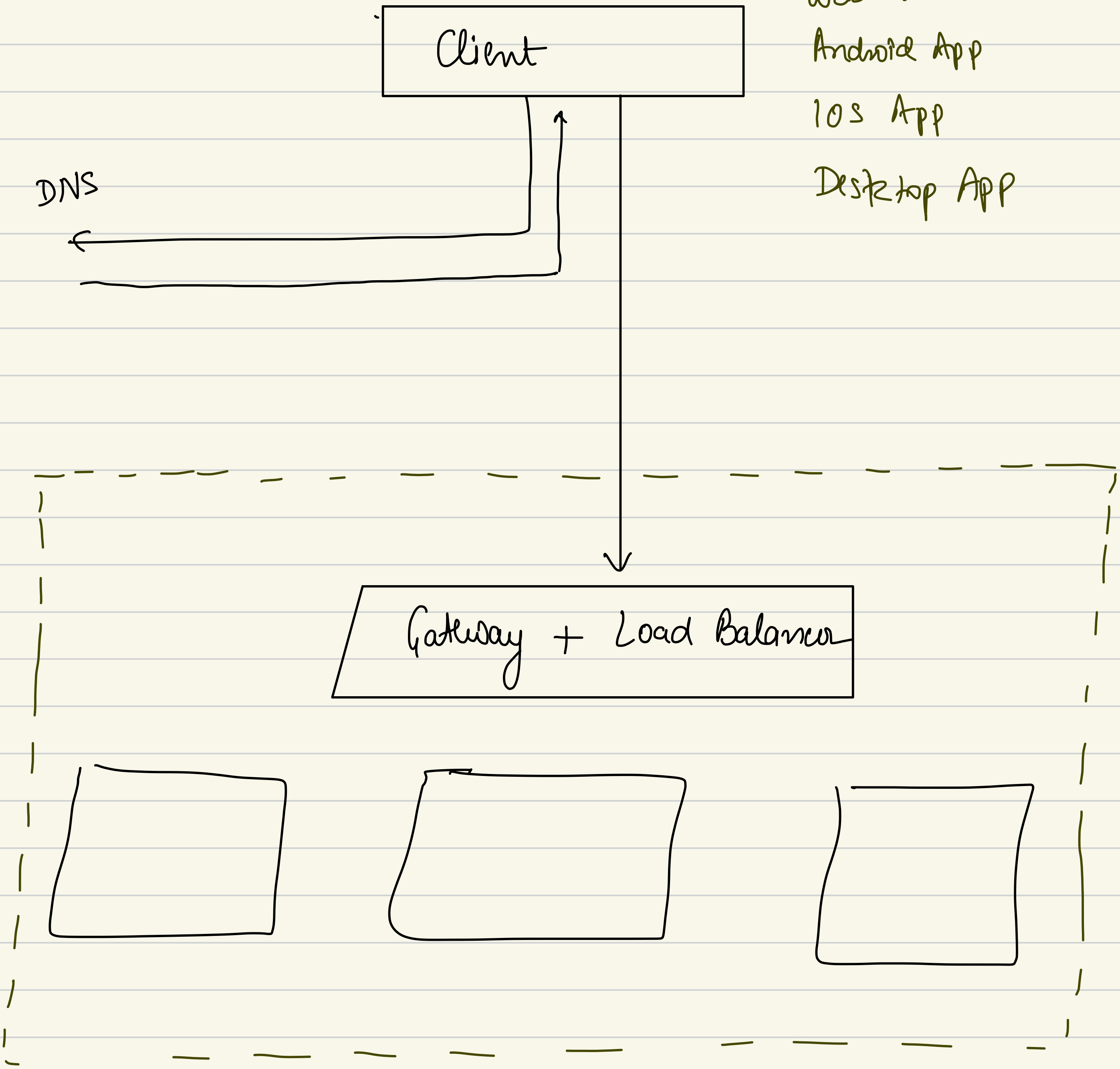
500 S2	600 S3		800 S5	900 S4		2000 S7

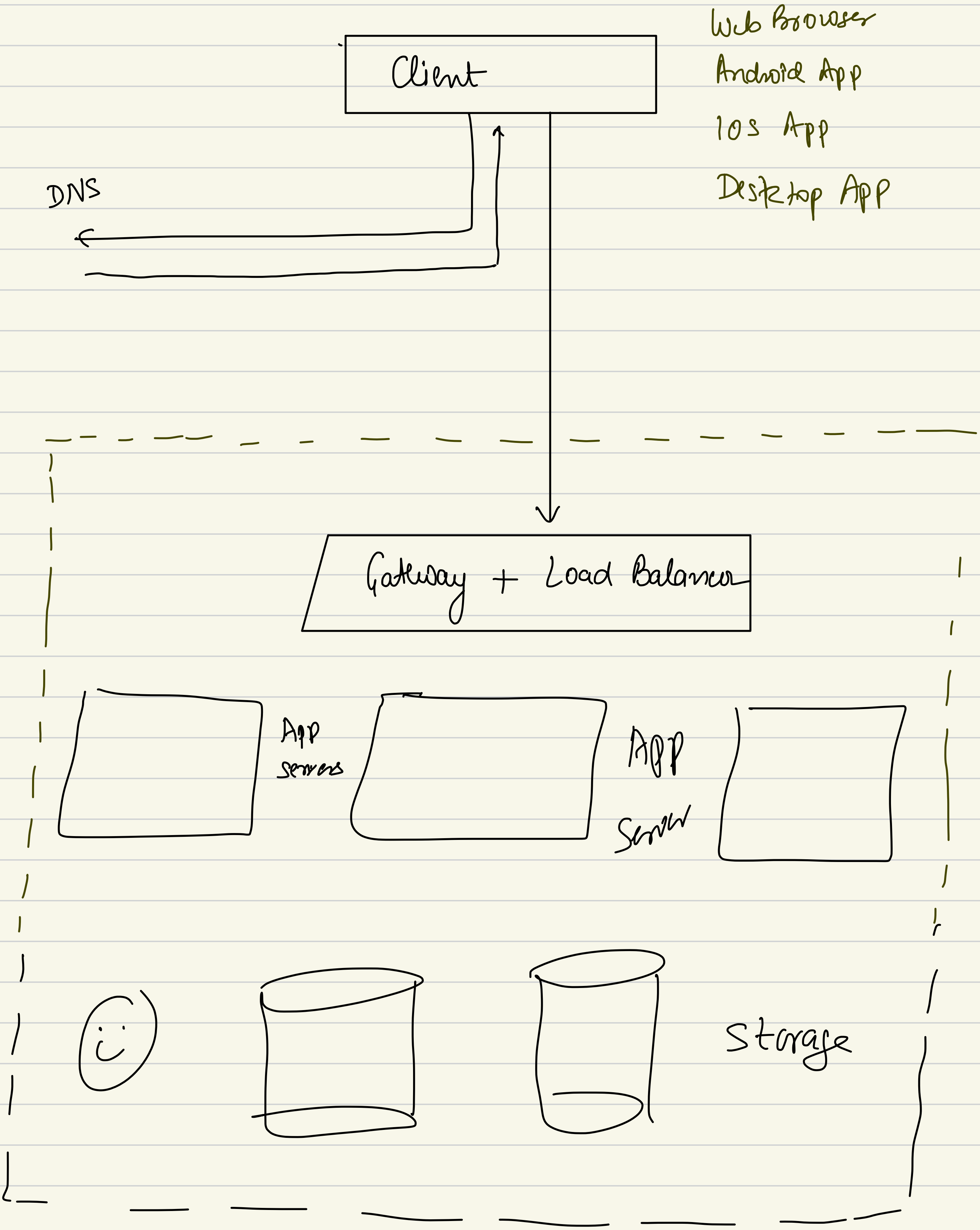
Web Browser

Android App

IOS App

Desktop App




Client
Web Browser
Android App
IOS App
Desktop App
DNS
Gateway + Load Balancer
App servers
App Server
Storage

In the backend, we don't just have one kind of machines doing everything for us

[Jack of all, master of none !!! 😜😛]

Instead,

we could want to employ different machines with different resources for different jobs.

For eg.

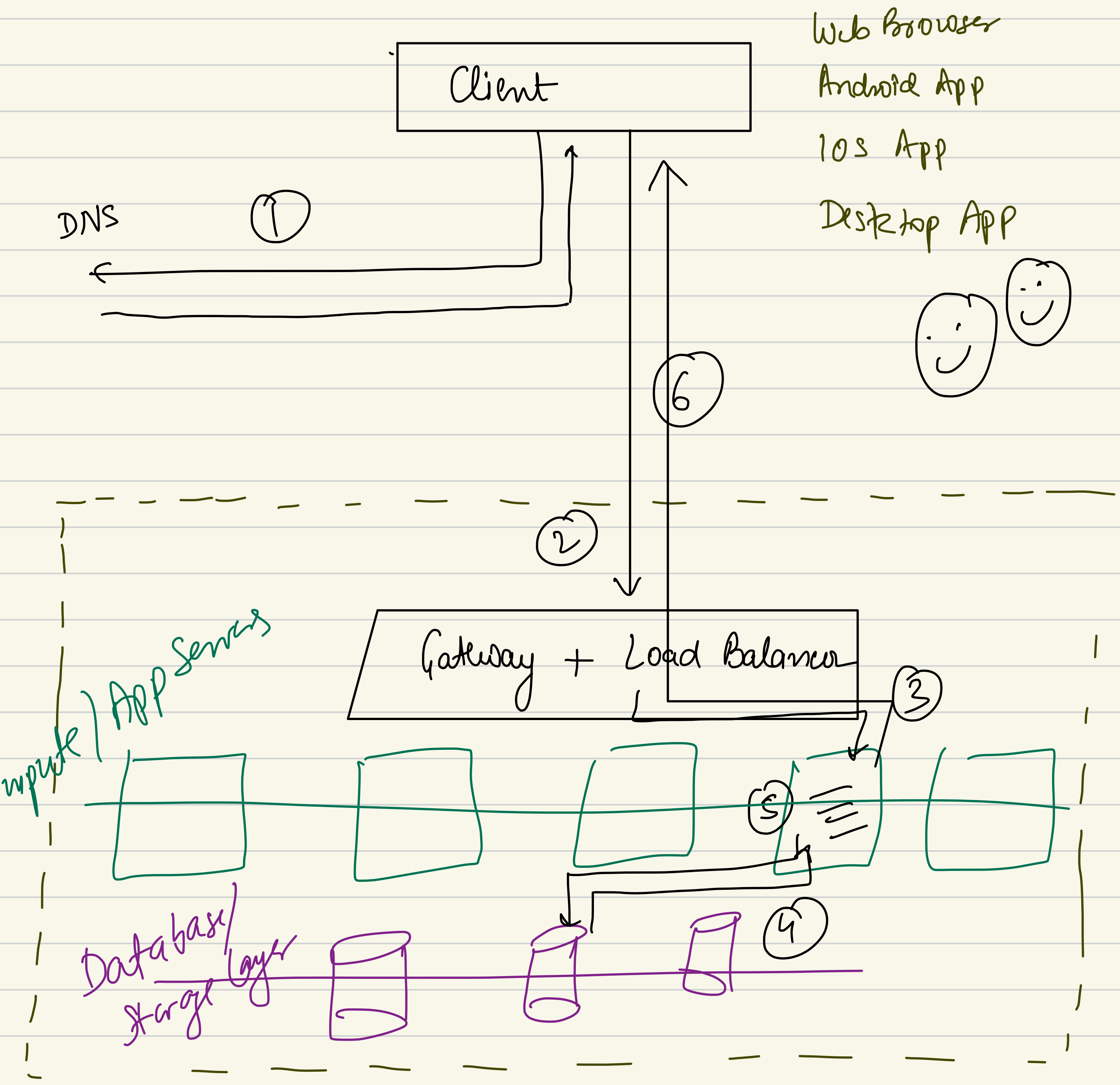
(1) compute layer | App Server layer

App Server / compute machines which are going to have good RAM, good CPU So that code and computations can run better.

② Storage Layer | DB Layer

HDD ↑↑

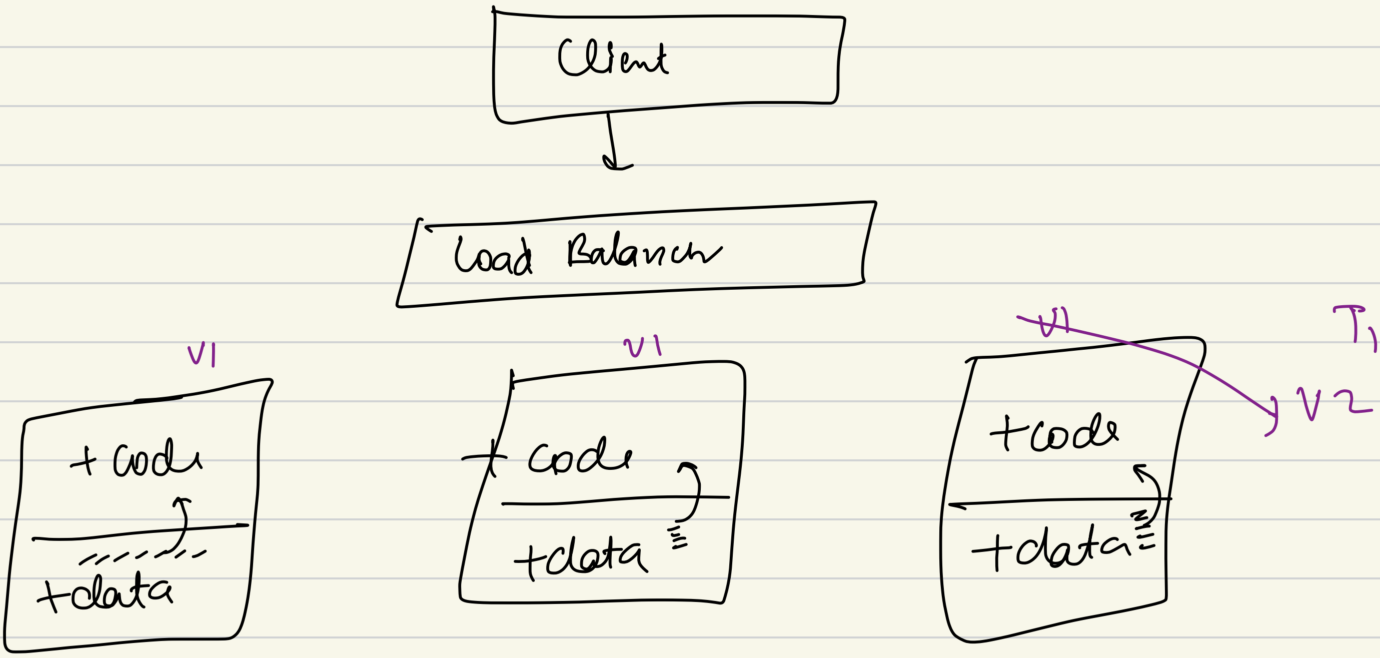
SSD ↑↑



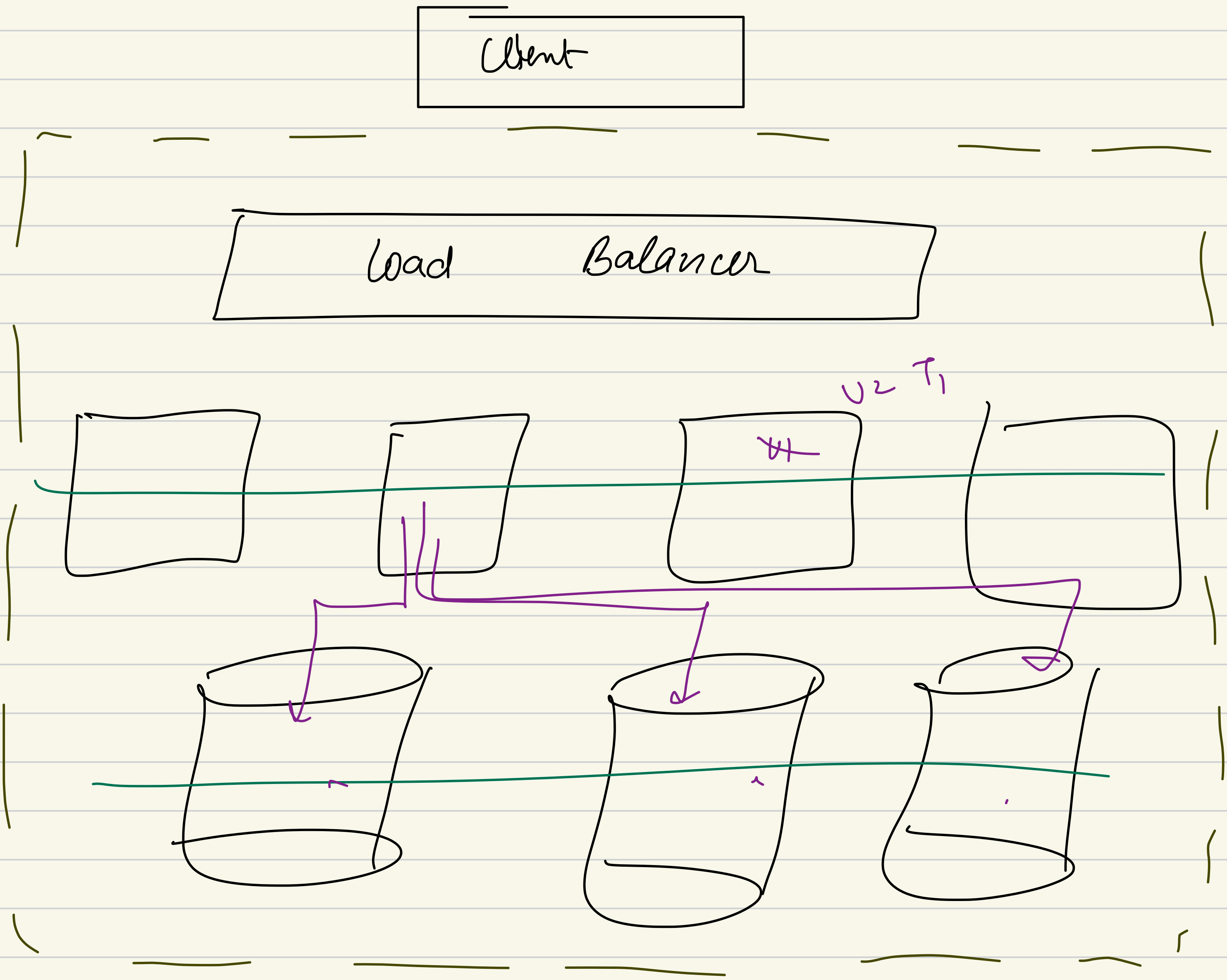
Why decoupling

1. Specialized resources.
2. Single Responsibility Principle ☺ ☺
3. Scale your systems better.
4. Code deployments become smoother

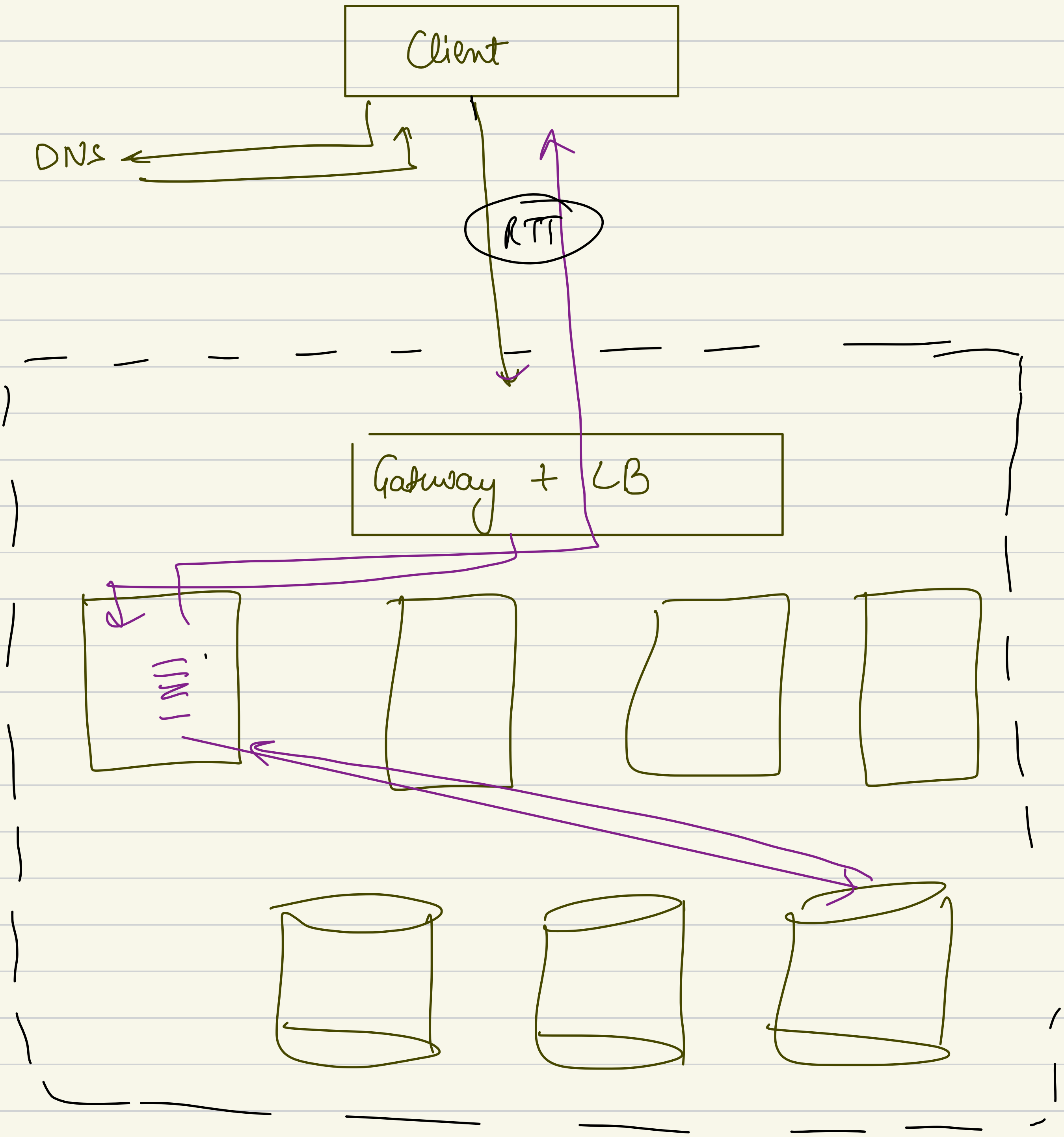
Case 1 : Coupled system 😐 😐 😐



Case 2: decoupling compute machine and storage machine



New Reality



Latency ≡ [Time at which the
request got completed

−

Time at which request
started]

[Turnaround Time]

User Latency

= [RTT (Round Trip Time]

+

Network Call b/w App server and DB

+

DB Access Time

+

Computation Time

hops

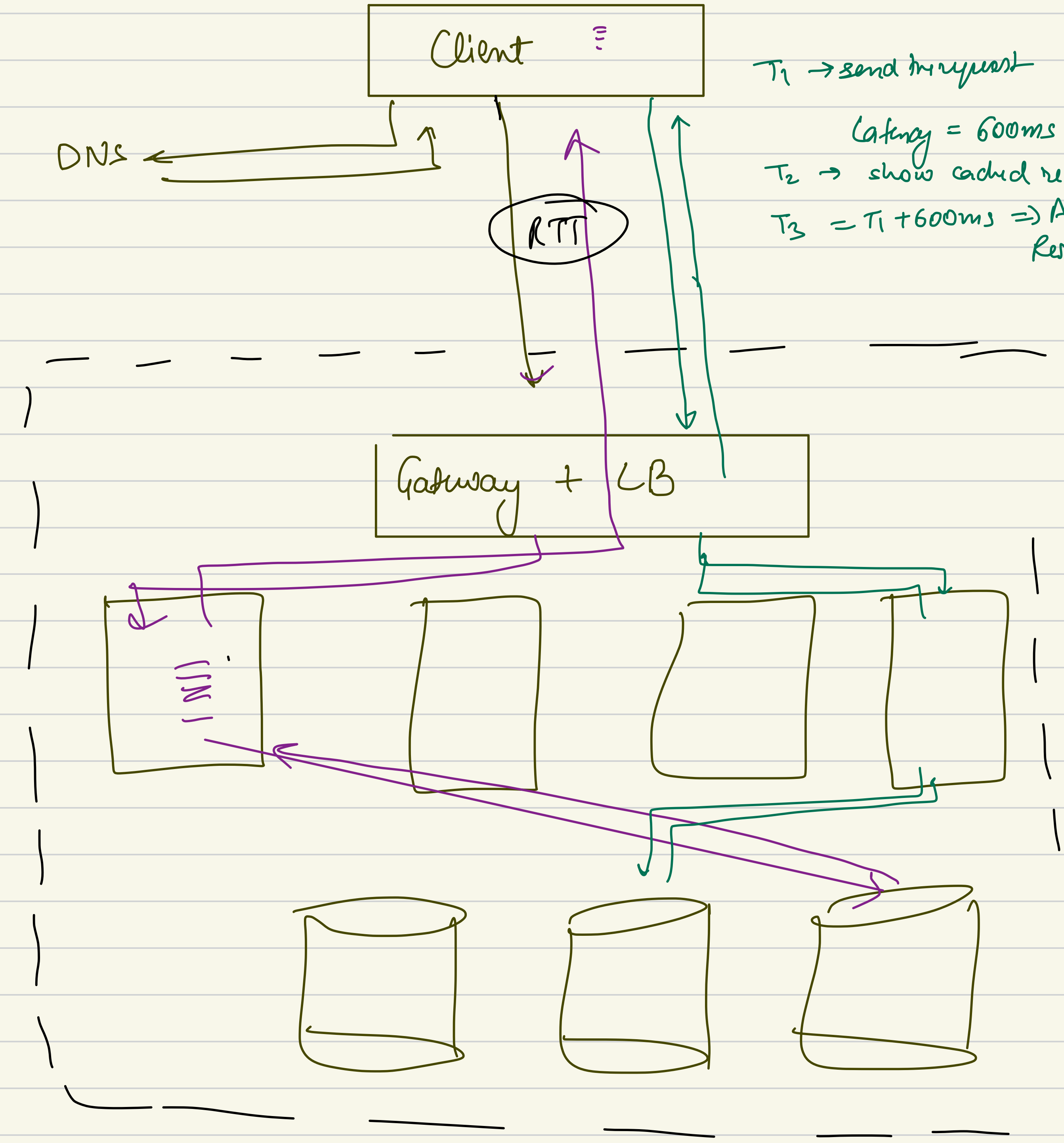
Caching: The idea of creating **partial copies** of underlying data so that we can access that data quickly. This is called caching

☺ ☺

for example in DNS

1. Browser Cache
2. OS Cache
3. Router Cache
4. Neighbourhood DNS Cache

① Browser | Client side cache



T1 → send the request

Latency = 600ms

T2 → show cached results

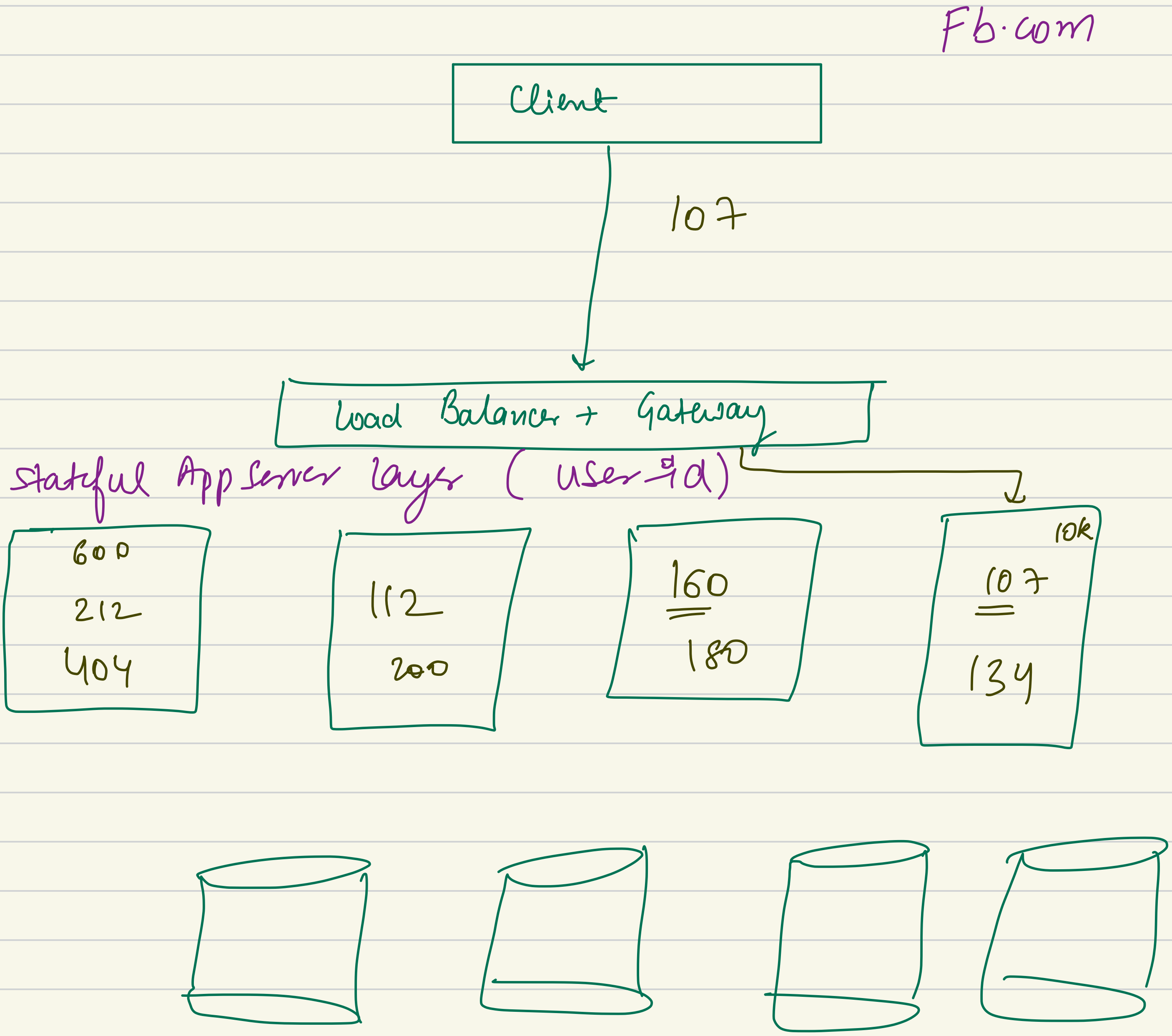
T3 = T1 + 600ms ⇒ Actual Results

Using client side caching (app data, browser cache)

The client engages the user by showing cached data, so that the perceived latency of the user reduces.

"Client side Optimizations"

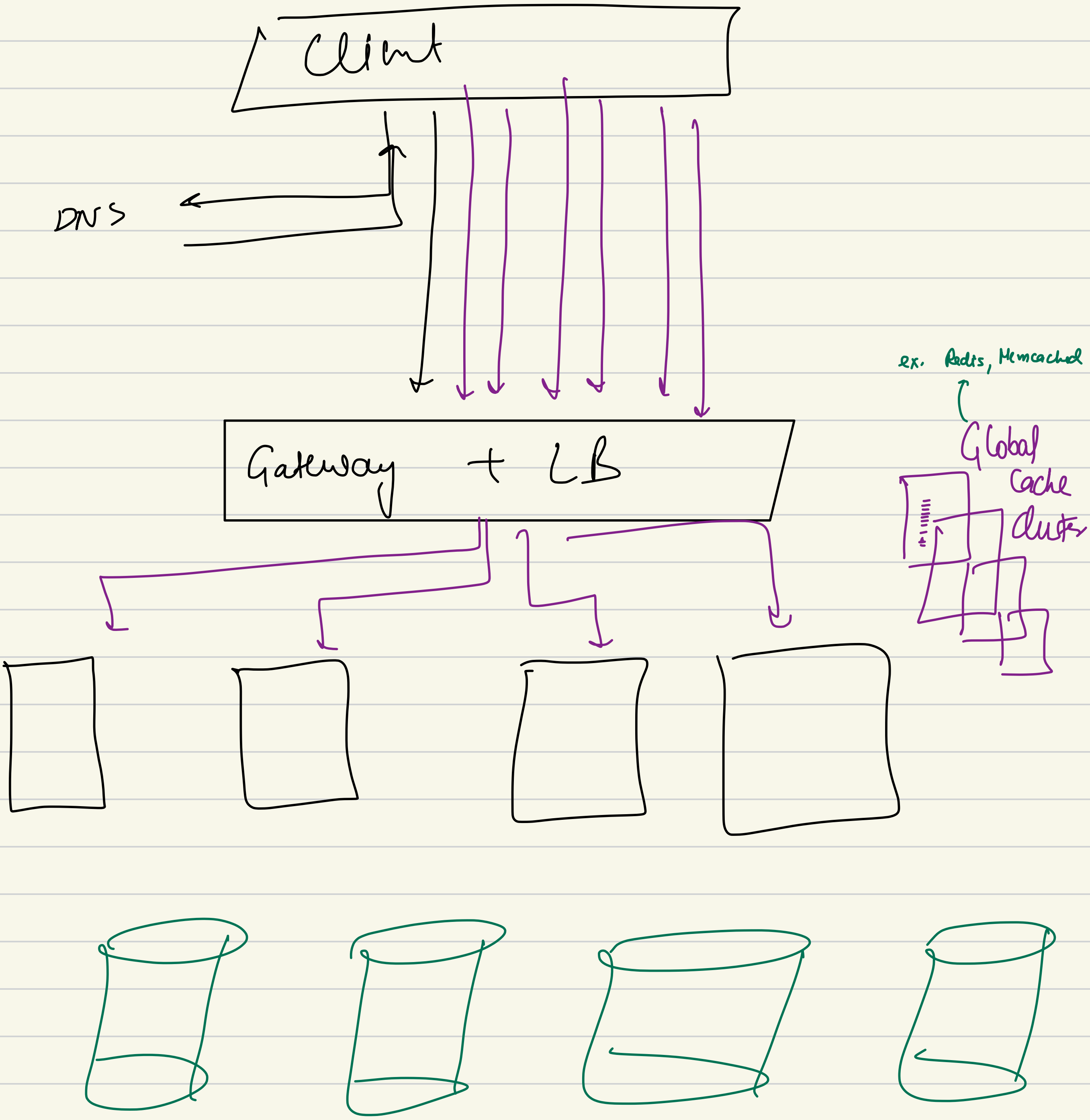
② App Server Cache



② App Server Caching | Local Caching ☺

Given we know that in stateful load balancing, a particular request is always going to land on the same machine, hence, we can make use of this and cache the relevant data on the relevant machines.

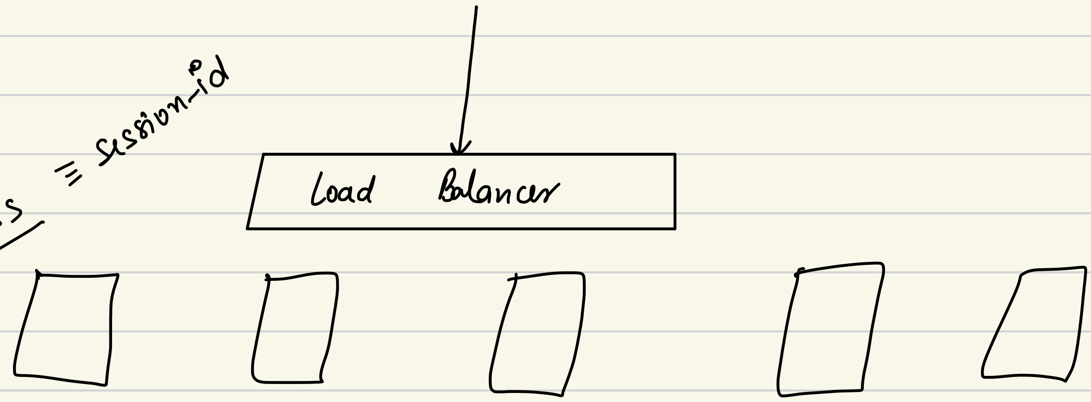
③ Global Caching

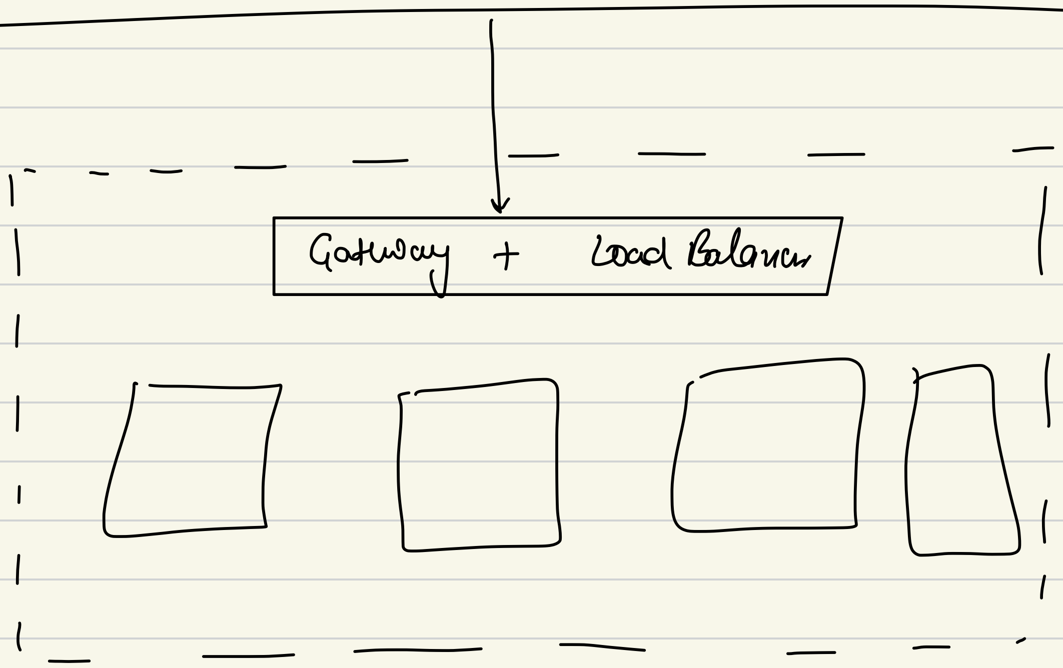


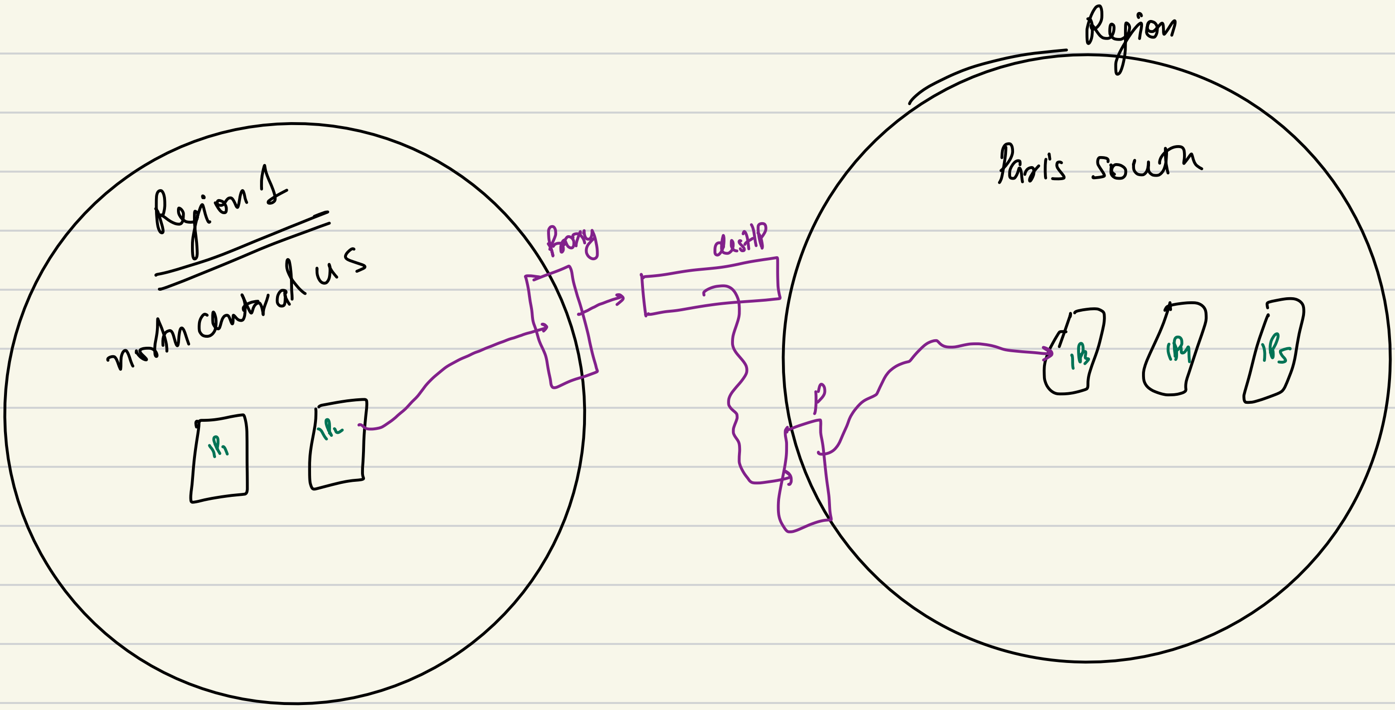
4 CDN

Questions

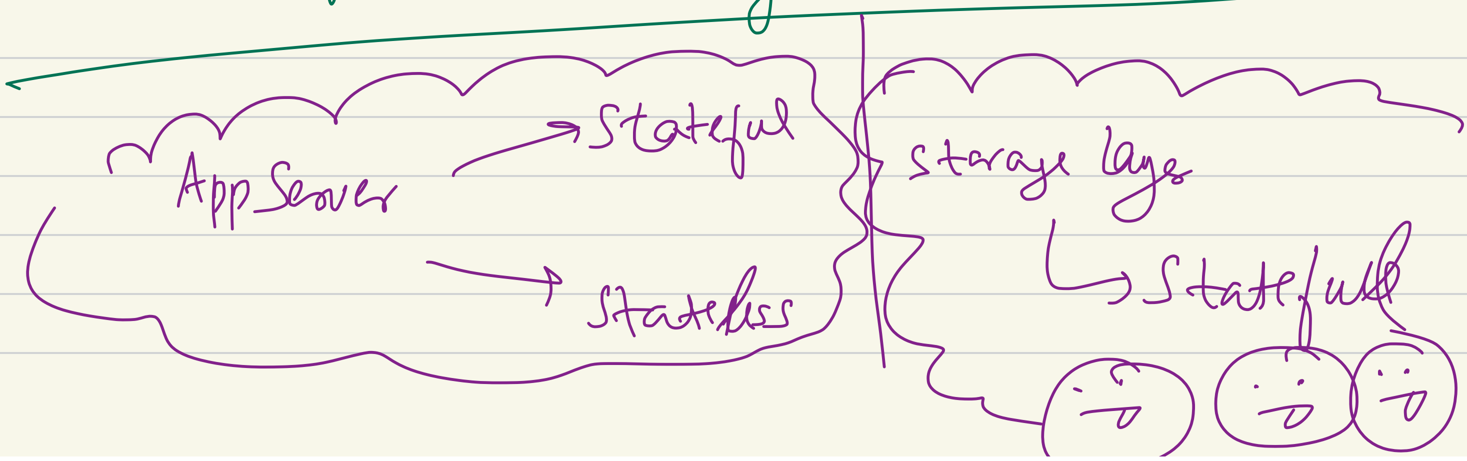
Session stickyness ≡ session-id







The same consistent hashing logic that we discussed for App Servers, the same is applicable for database layer as well; in fact the same is applicable for the caching cluster as well.


App Server
Stateful
Stateless
storage layer
Statefull