Agenda.

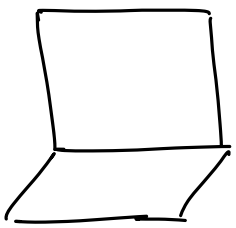
→ Load Balancing

→ Consistent Hashing.

Bookmarking App^n.

del.icio.us.

→ SPOF



2 Core CPU
40GB
512 MB RAM.

⇒ Vertical Scaling (Vs) Horizontal Scaling

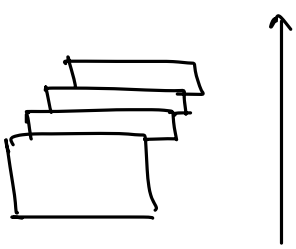
↓

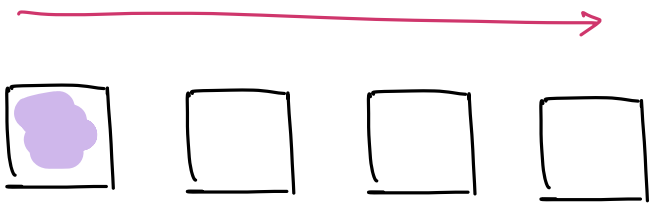
Adding m/c with more resources.

→ Limited Scaling

→ SPOF.

→ No Elasticity.

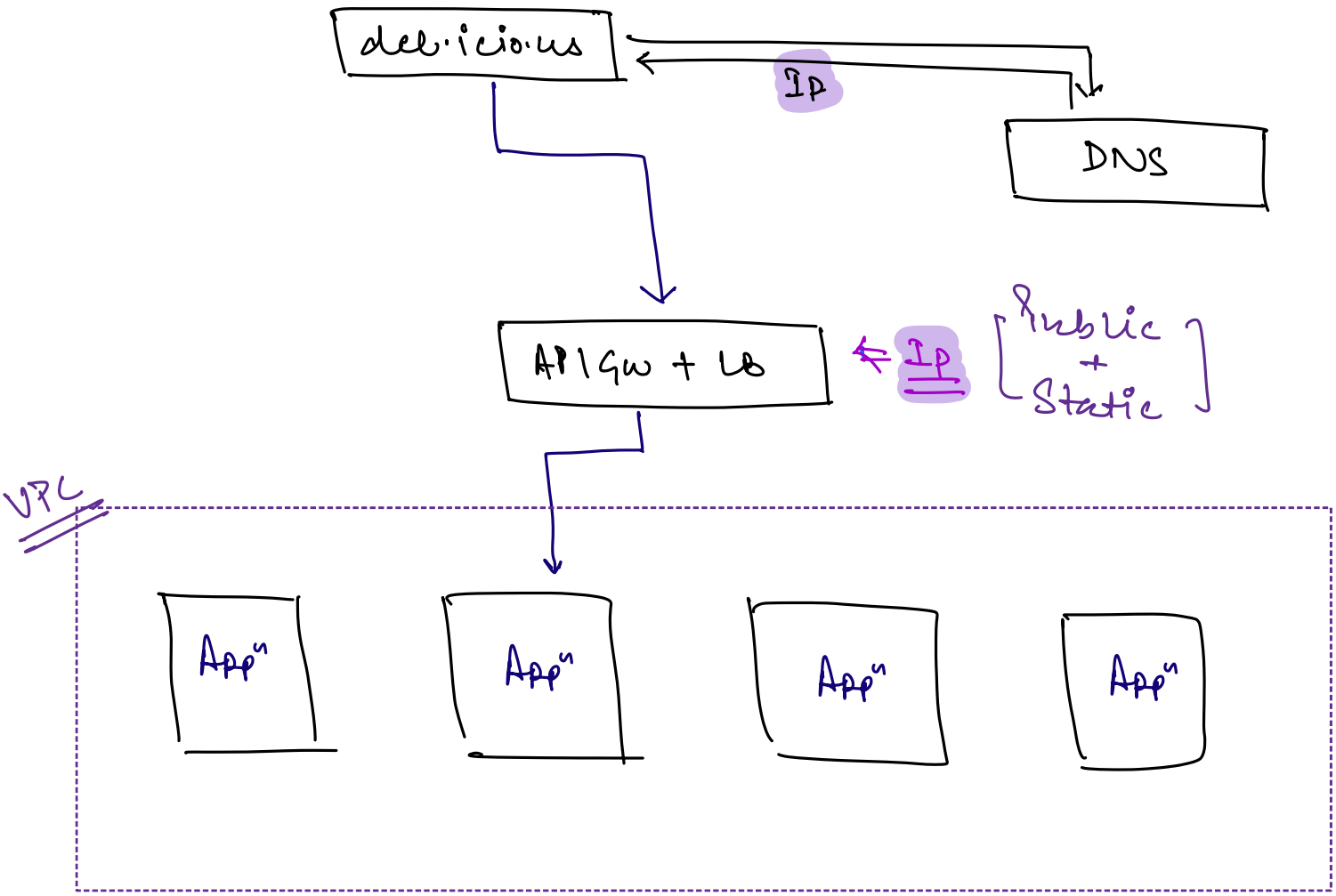




→ No SPOF

→ No limit on Scaling

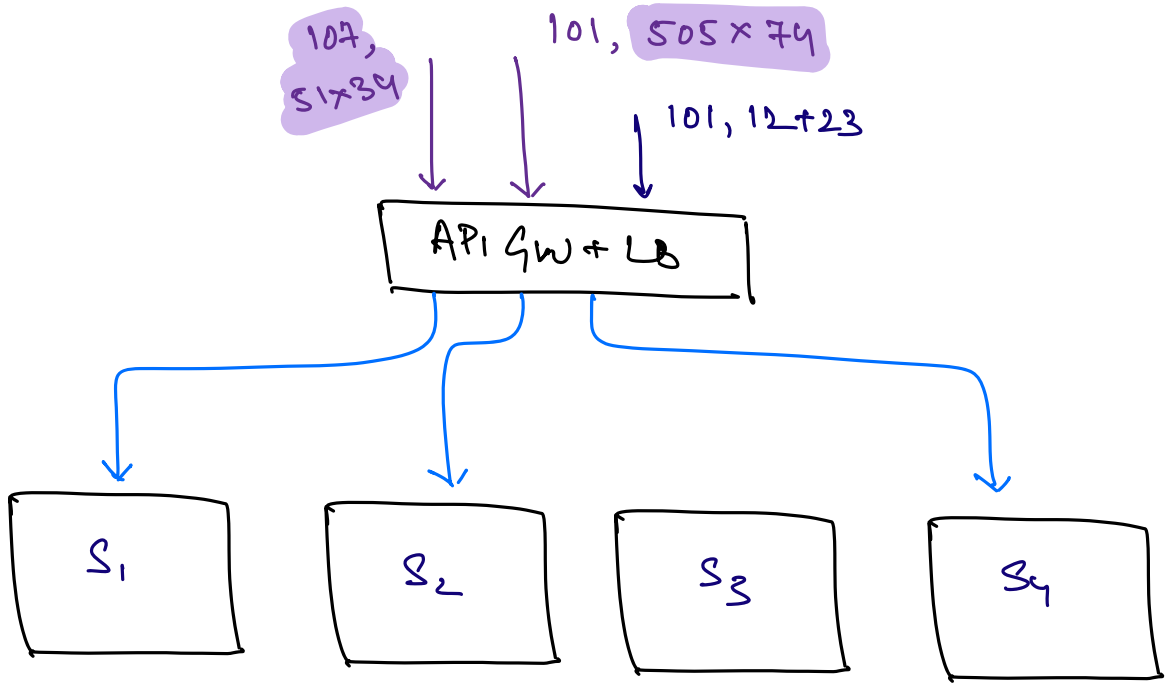
→ Elasticity.


delicious
Ip
DNS
APIGw + LB
Ip
Public
+
Static
VPC
App"
App"
App"
App"

Stateless (vs) Stateful load Balancing

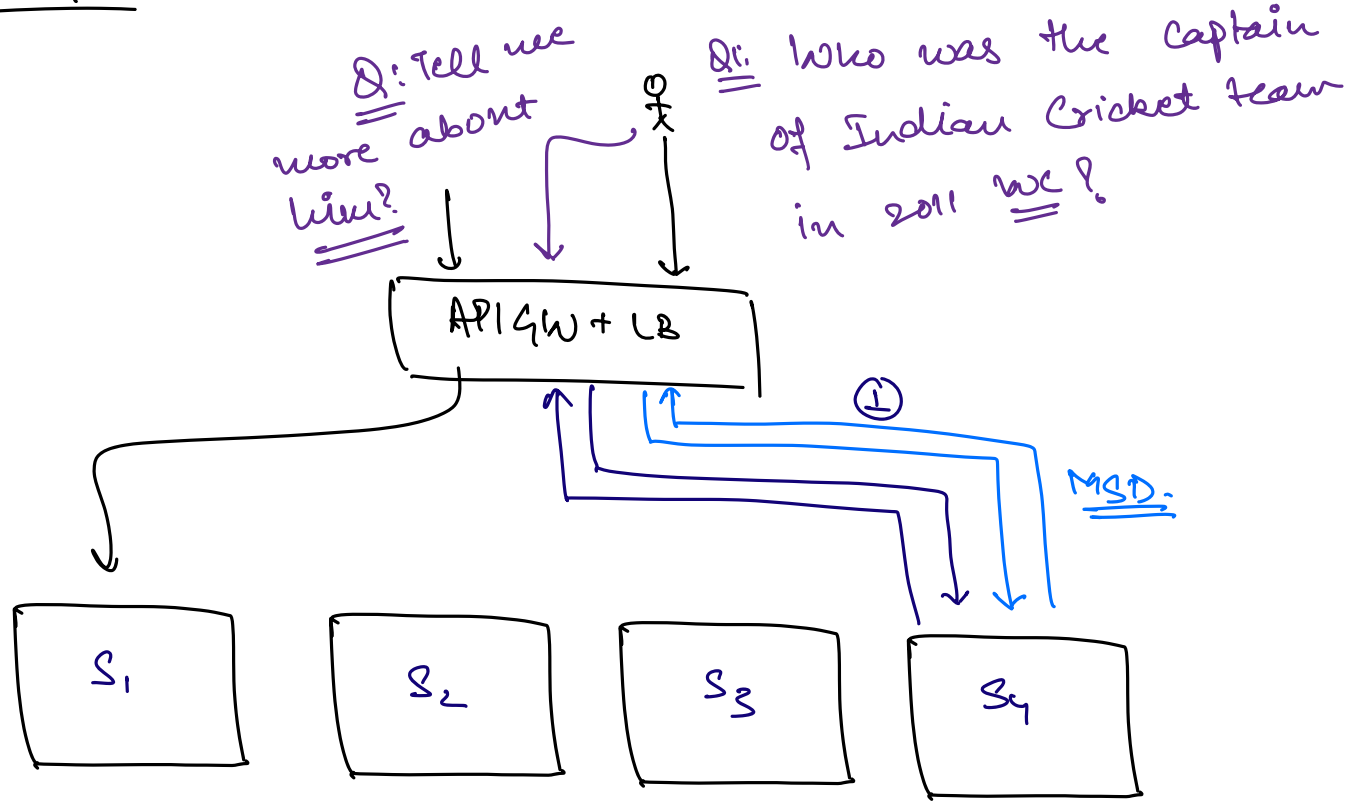
Calculator.

⇒ Stateless.



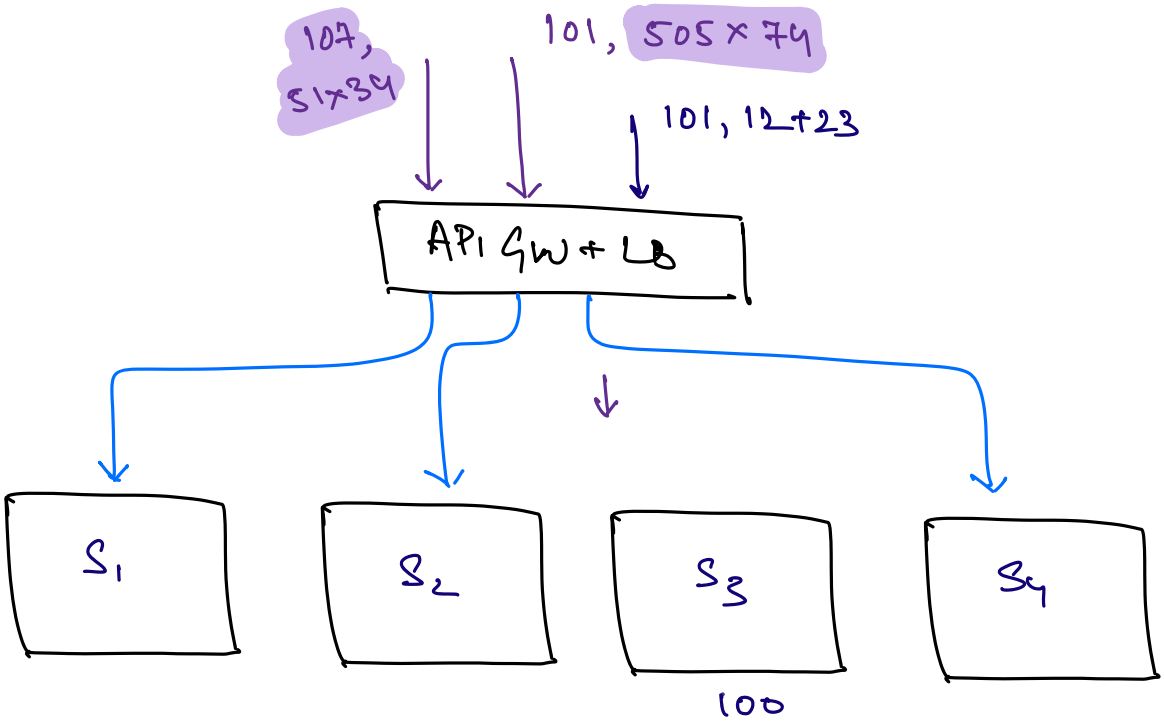
⇒ When requests are completely independent. We can route them to any server randomly.

ChatGPT



Stateful.

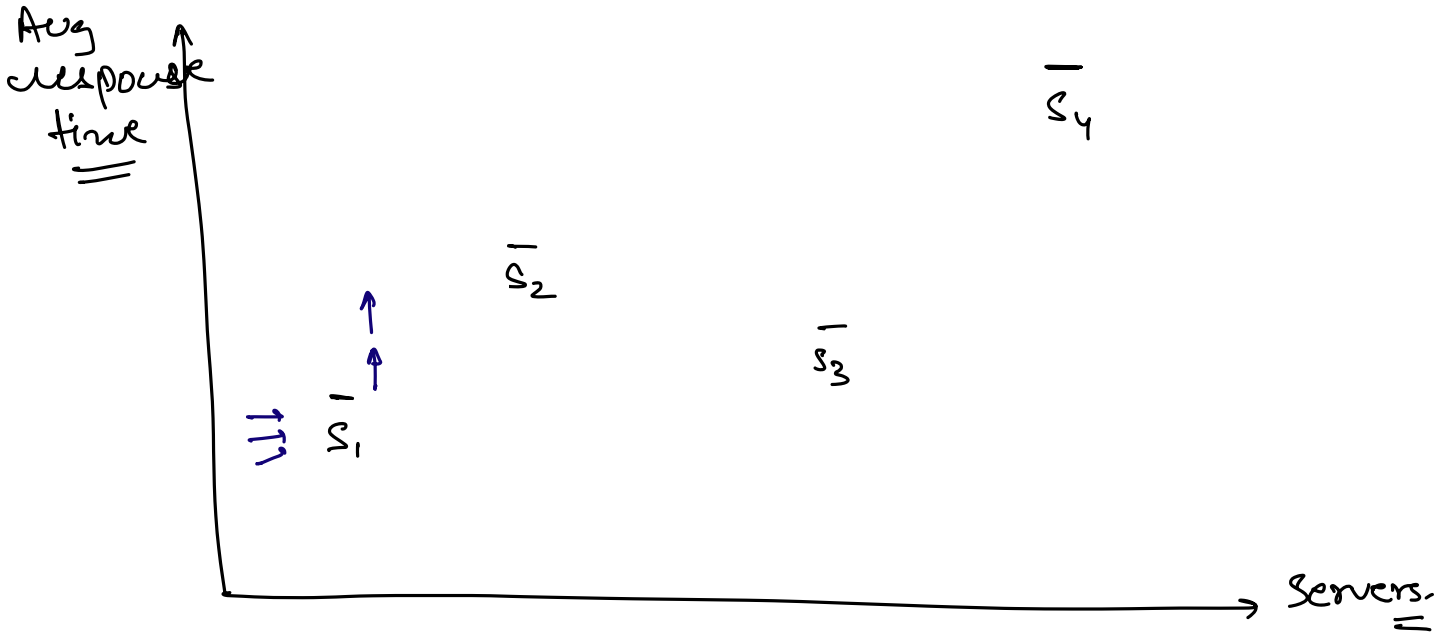
Stateless.



→ Round Robin

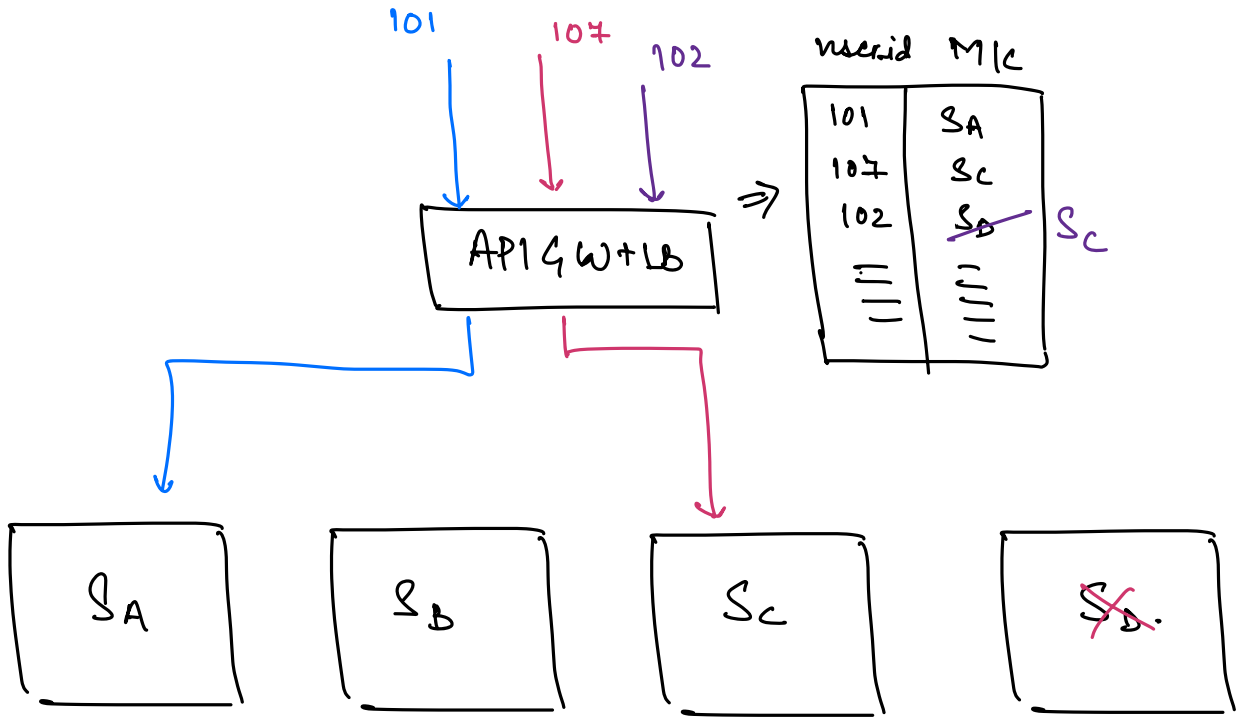
→ least Connections first.

→ least response time.



Stateful load balancing

Approach #1

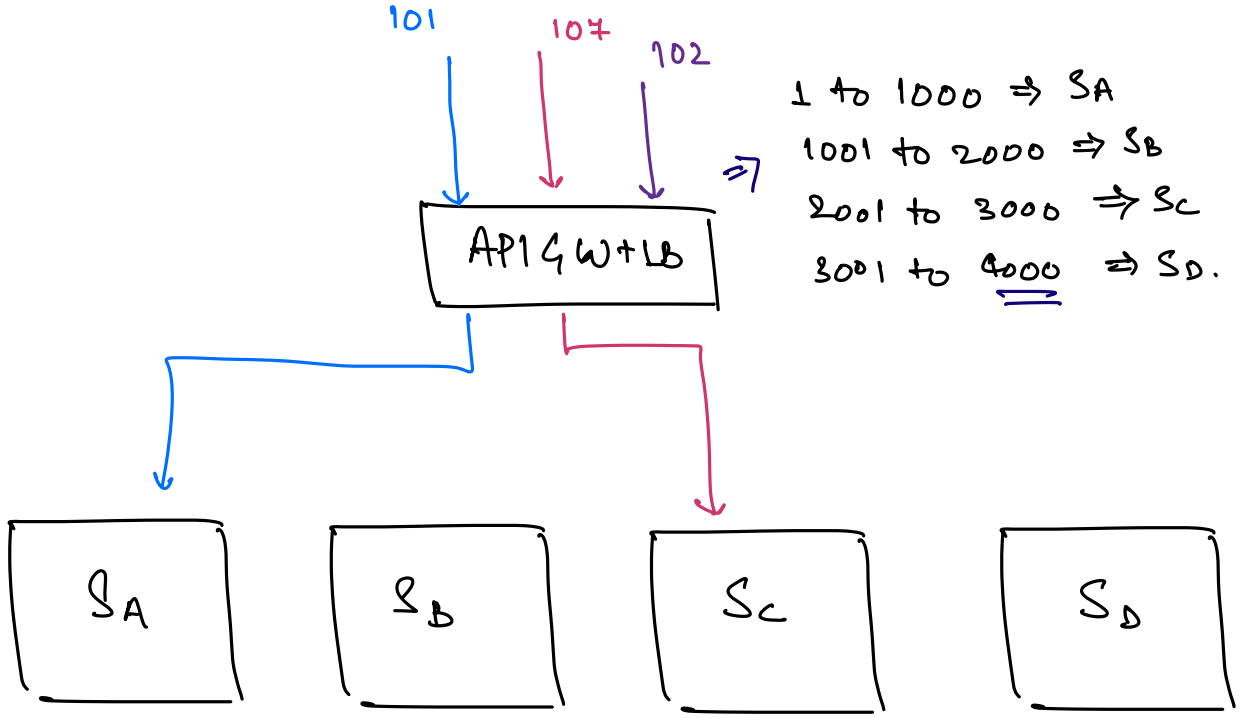


Size of the Map can be huge ≡ User Base.

Approach #2

↳ Range based mapping.

M/c ≡ Server ≡ Shard

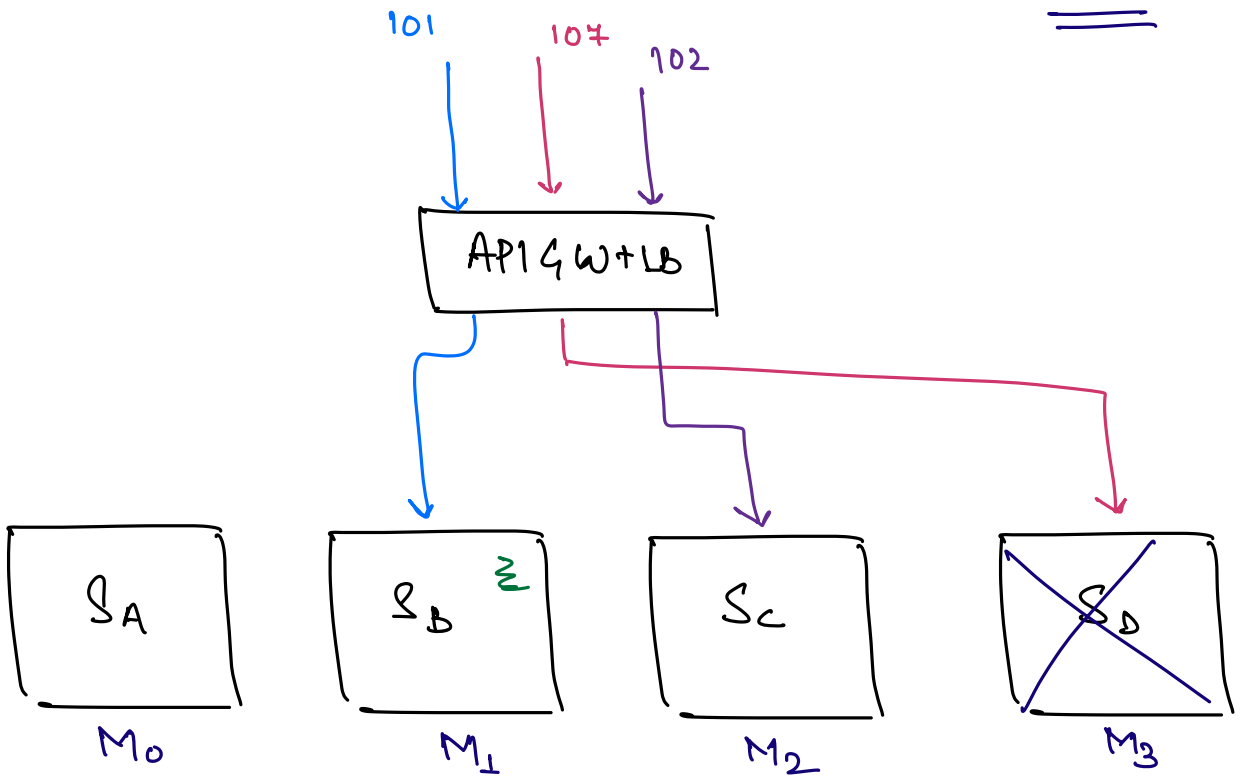


Approach#3.

↳ user_id % (N) ∈ [0, N-1]

No. of m/c

N=4



101 % 4 = 1 ⇒ 101 % 3 ⇒ (2)

102 % 4 = 2 101 % 6 ⇒ 5

103 % 4 = 3 107 % 6 ⇒ 5

104 % 4 = 0

Pro: Simple to implement.

No mapping required on LB.

Cons:

↳ Scale Up | Scale Down | M/c goes down.

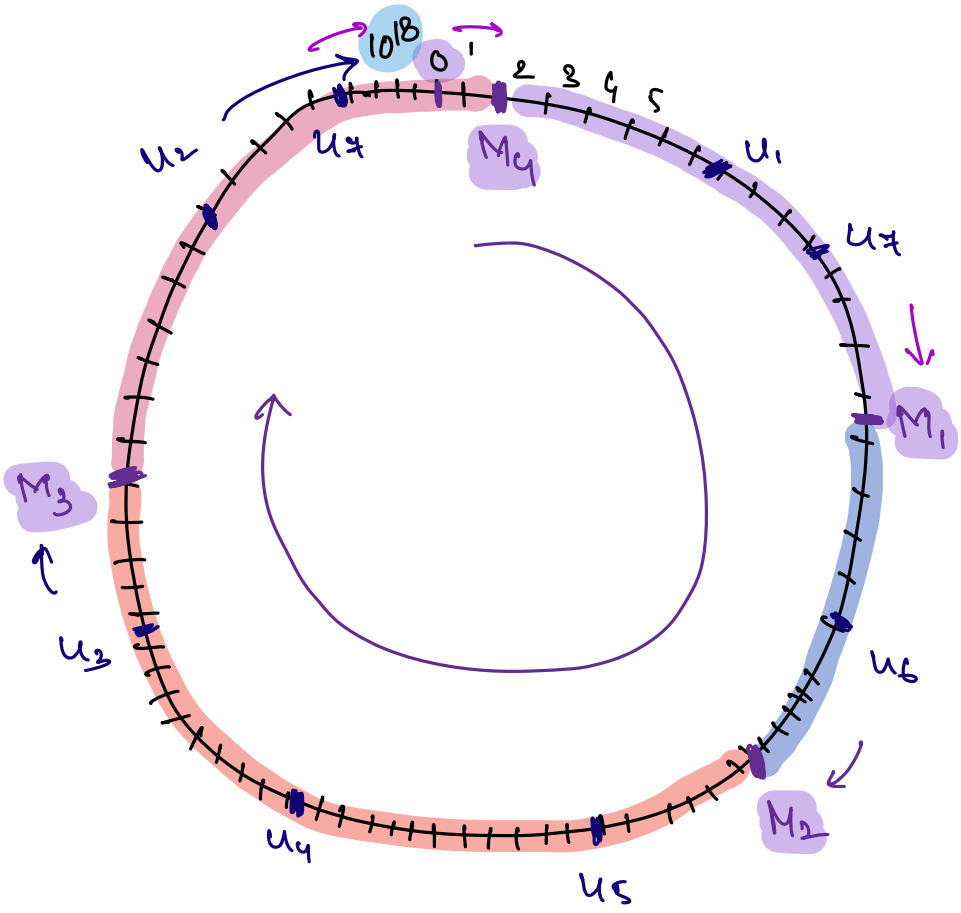
Challenging

Approach # 4. Consistent Hashing

1) Stateful load balancing.

2) We need to decrease the chaos that gets created because of Scaling Up | Scaling down & when a m/c goes down.

3) Easy to maintain at LB.



Logical Ring

$\Rightarrow$ 0 to 10^{18}

$H_{sem} \in [0, 10^{18}]$

H_s

H_u

H_s (server_id) =

hash

$H_s(S_1) = M_1 = 6,67,81,234$

$H_s(S_2) = M_2$

$H_s(S_3) = M_3$

$H_s(S_4) = M_4$

user_id

H_u(user_id) $\in [0, 10^{18}]$

$H_u[107]$

$H_u[102]$

$H_u[201]$

⇒ In CH, we use a Hs fun to place the servers on a Consistent Hashing ring.

⇒ Hu ⇒ To place users on a Consistent Hashing ring.

⇒ For every user on CH ring, we'll map to the first server coming in the clockwise direction.

⇒ Trade Off

—— * ——