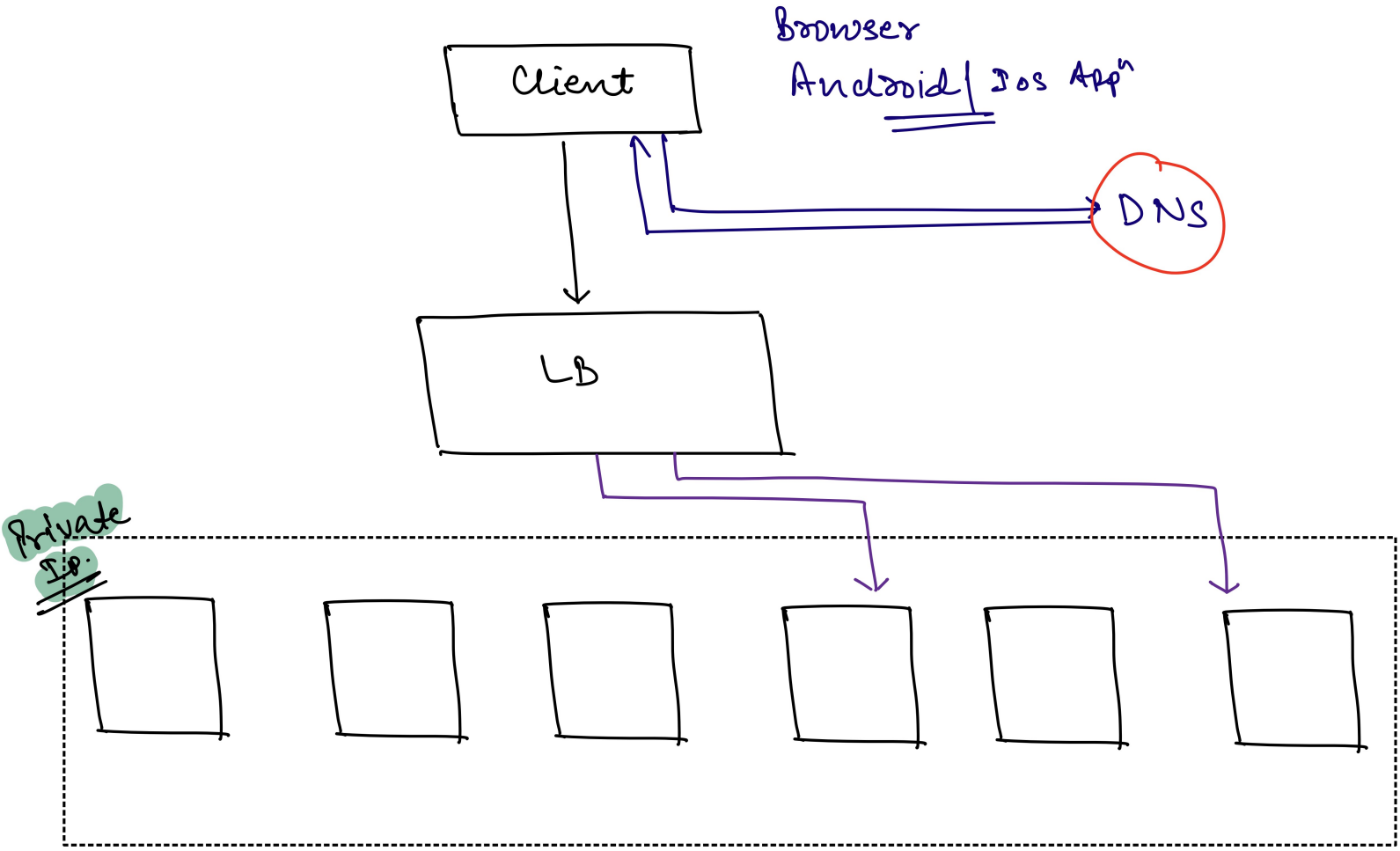
Agenda.

→ Consistent Hashing

↳ Load Balancing Algorithm.

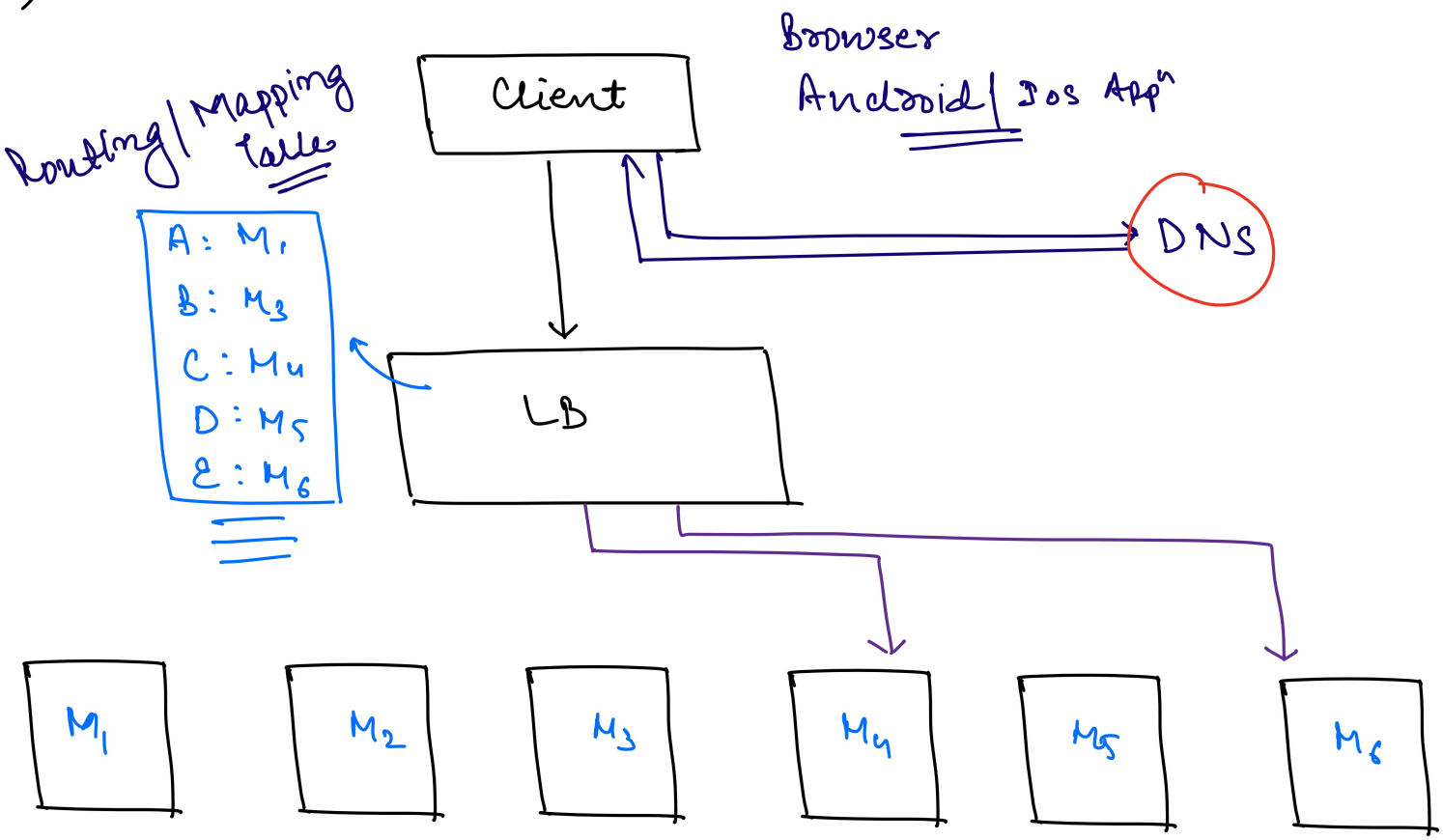
→ Introduction to Caching.



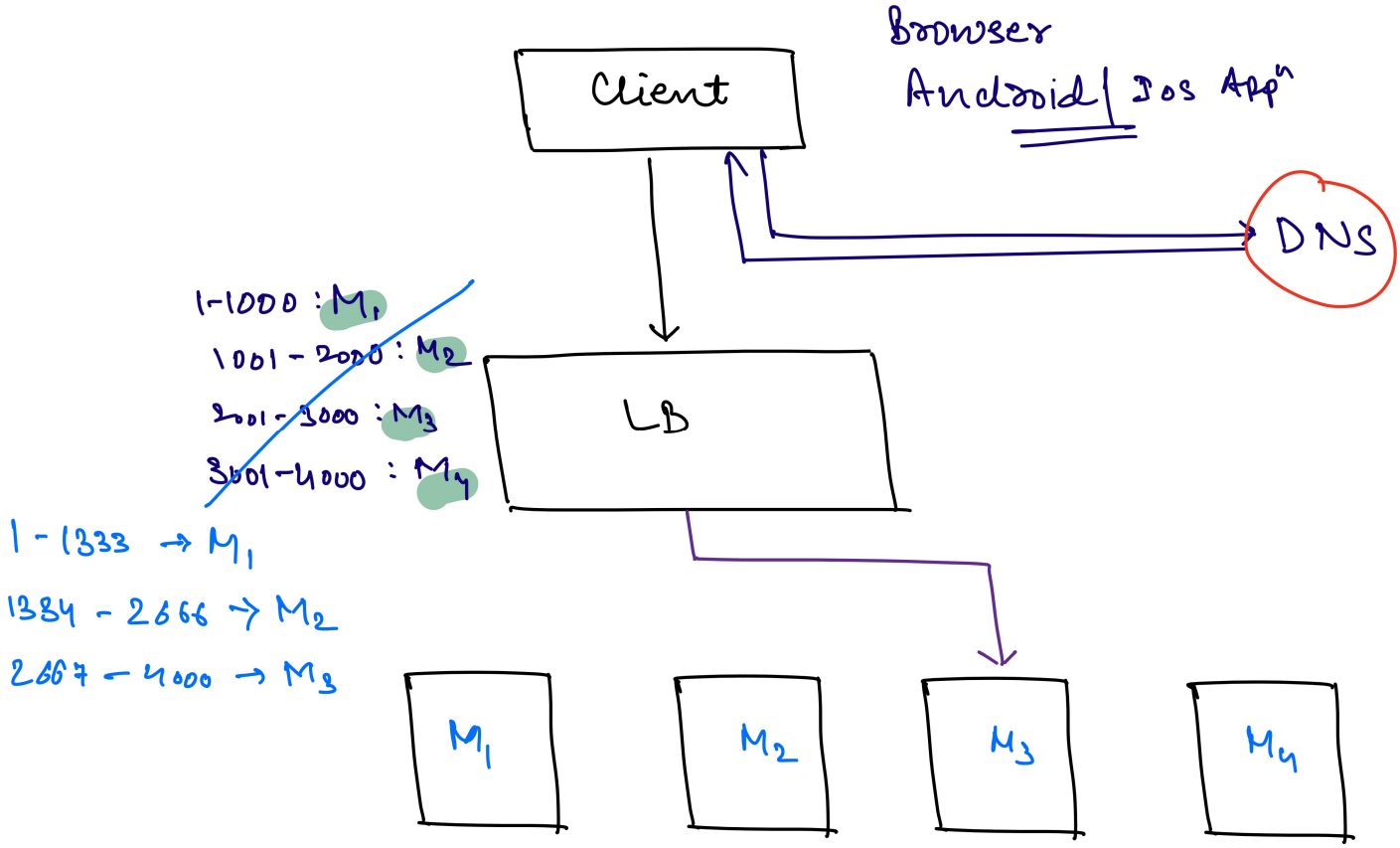
Load Balancing → Stateless → RoundRobin
Stateless → Least Response Time
Stateless → Least # of Connection
Load Balancing → Stateful.

#

I)



II) Range based Mapping



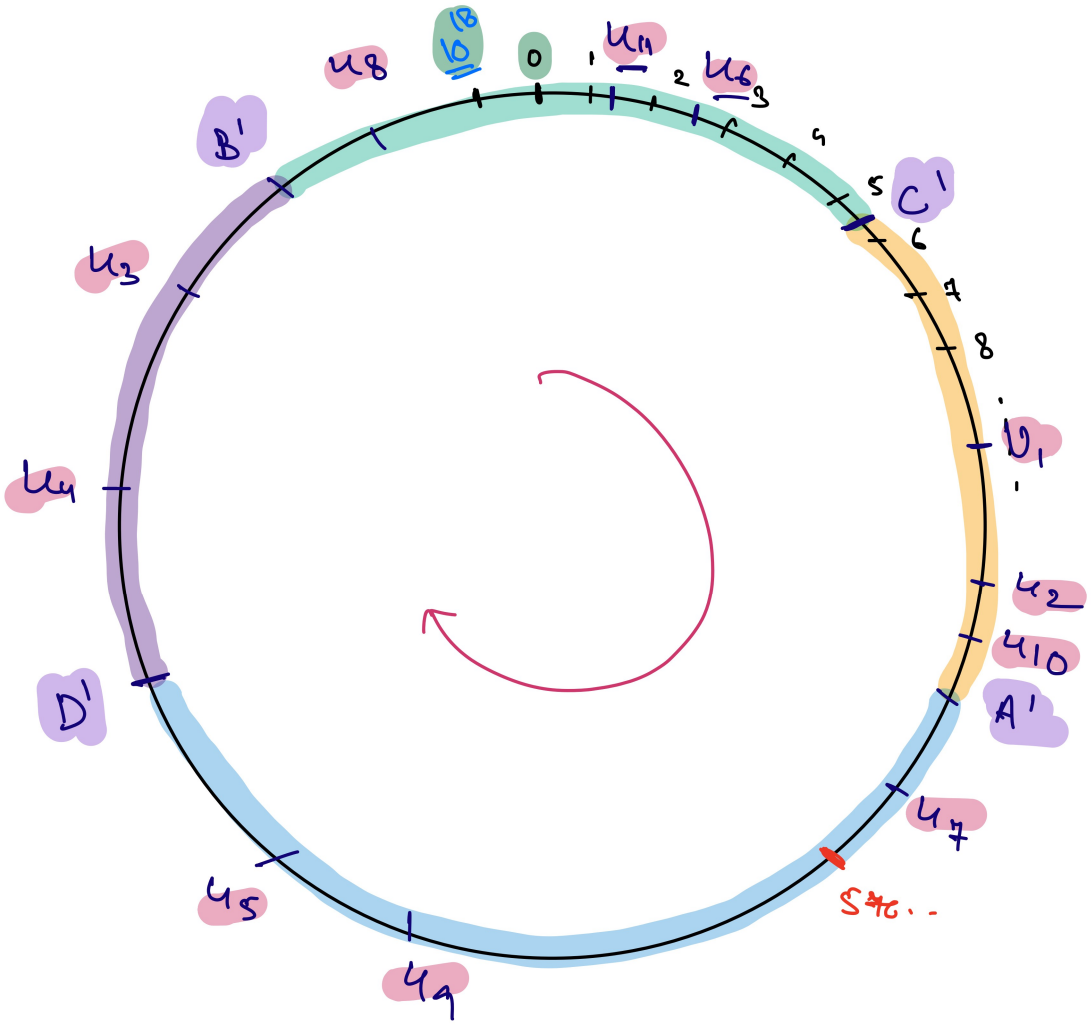
⇒ Lot of unnecessary data transfer will happen in this approach.

III) Modulo based routing.

IV) Consistent Hashing

⇒ Stateful Load Balancing Algorithm

⇒ Tries to minimize the data movement in case a m/c is going up (or) a m/c is going down.



⇒ Hash fun^

machines.

A
B
C
D

$H_s(A) = A'$

$H_s(B) = B'$

$H_s(C) = C'$

$H_s(D) = D'$

⇒ In consistent Hashing, we use a hash fun^ to distribute all the m/c across the ring.

⇒ H_u (user_id)

$H_u(107) = x$

$H_u(u_2) =$ —

$H_u(u_3) =$ —

$H_u(u_4) =$ - -

⇒ For a user, it's request will be allocated to the first server present in the clockwise direction.

⇒ Data for only the users who were mapped to a m/c which is going down needs to be reallocated.

⇒ No reallocation required for other users.

$H_s(A) = A' = 10^4$

$H_s(B) = B' = 10^{11}$

$H_s(C) = C' = 5$

$H_s(D) = D' = 10^9$

C'	A'	D'	B'
5	10^4	10^9	10^{11}

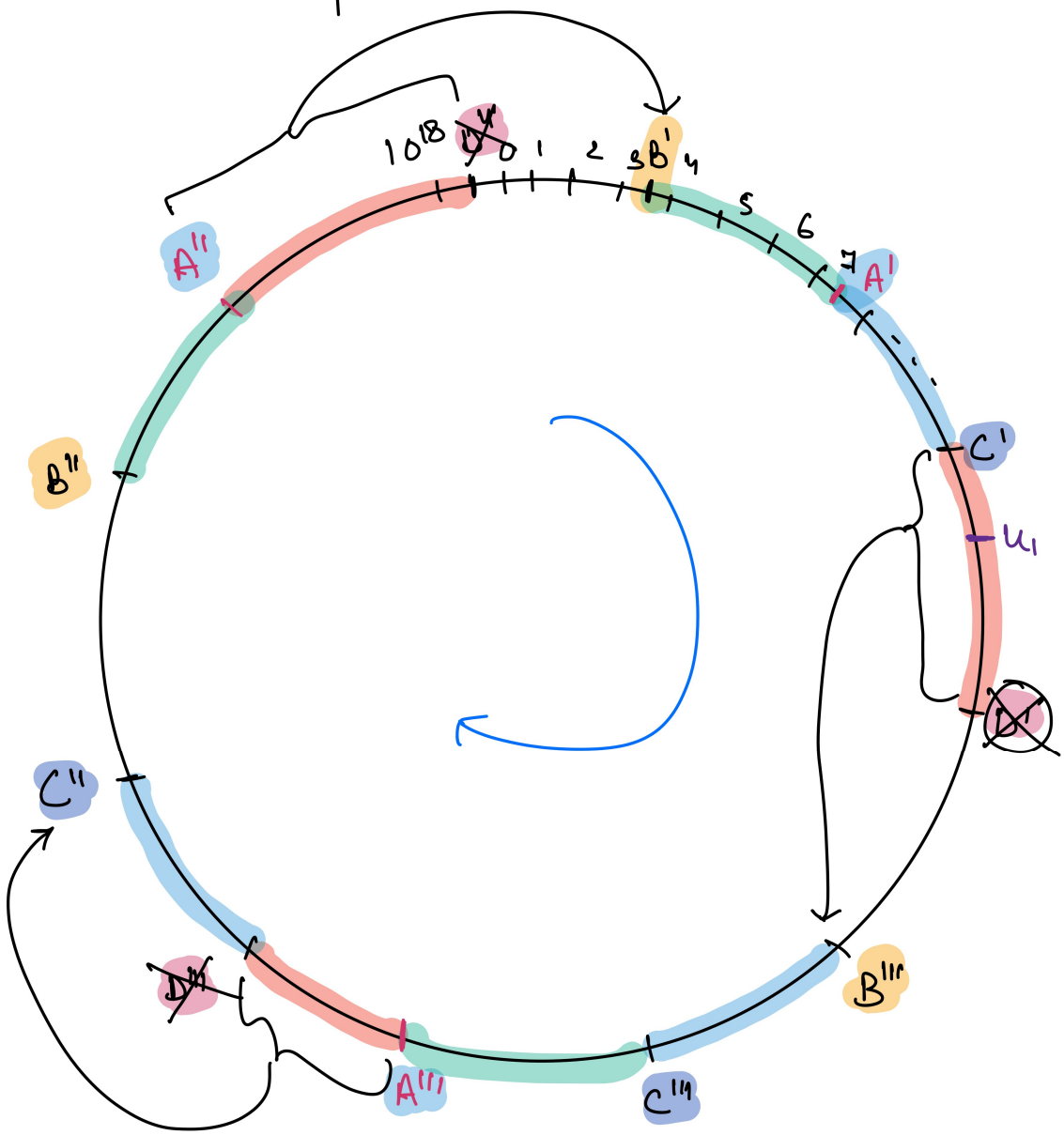
$H_u(107) = 576783$

Find the next greater no than this Number.

Cascading failure

Soln

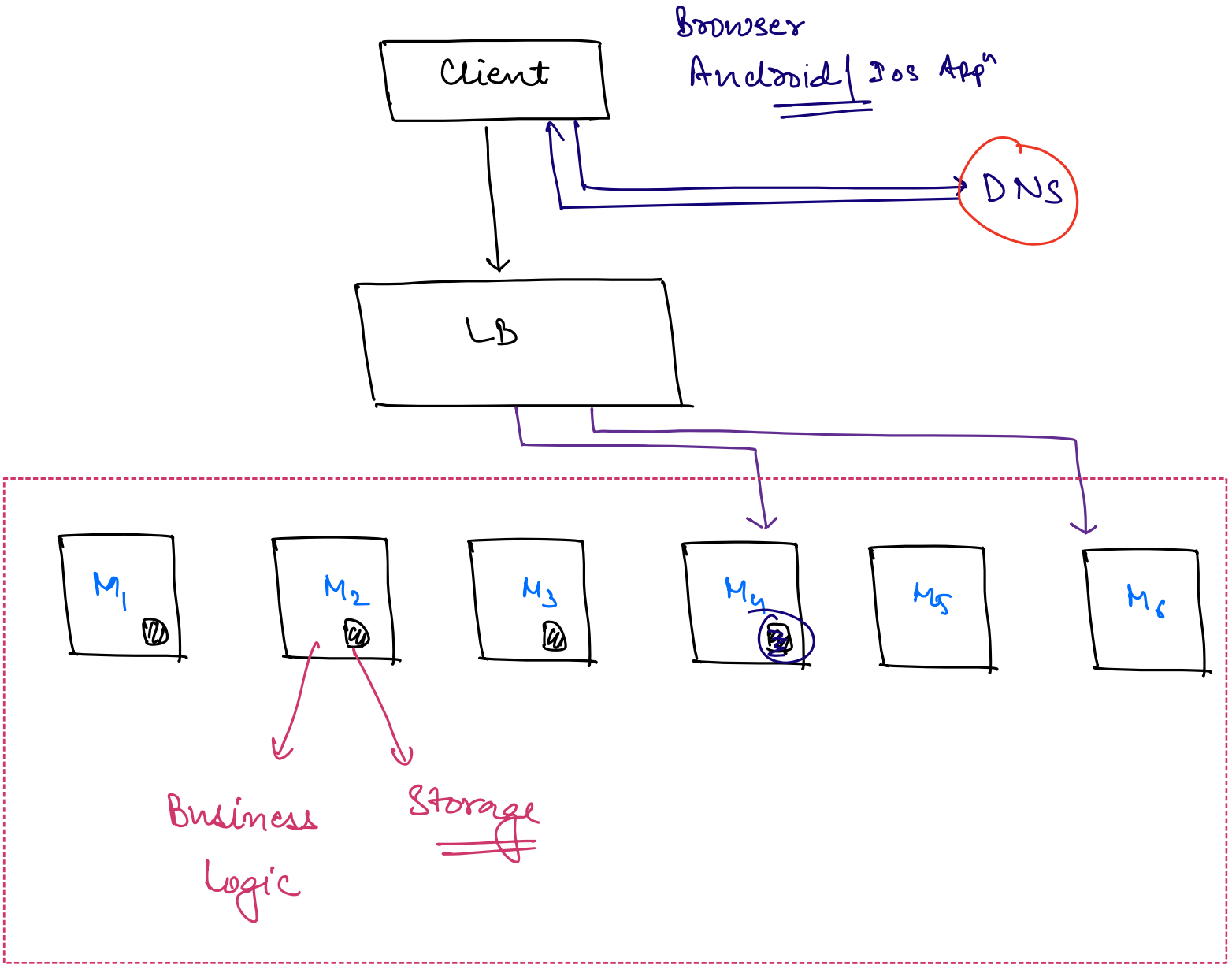
$H_{S1}(A) = A'$	$H_{S1}(B) = B'$
$H_{S2}(A) = A''$	$H_{S2}(B) = B''$
$H_{S3}(A) = A'''$	$H_{S3}(B) = B'''$



Hu(104)

$u_1 \Rightarrow D$.

Caching

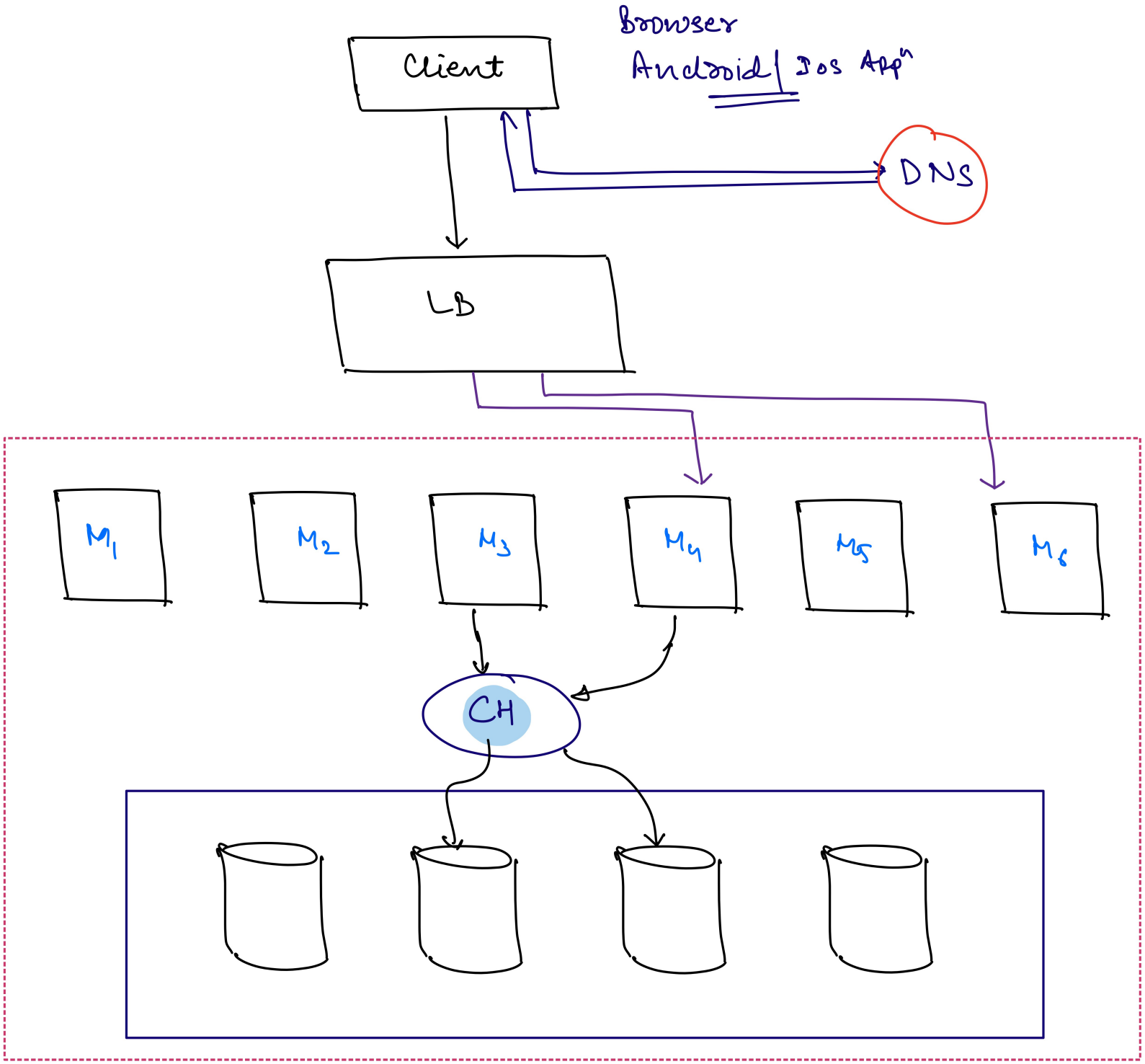


⇒ If App^n server and DB are present together in same m/c.

I) If app^n server goes down the DB also becomes inaccessible.

II) Because of continous deployments, App^n server goes down and during deployment DB also gets restarted.

III) Separate H/w requirements for App^n & DB servers.



⇒ In large scale app^n, storing entire data in one DB m|c won't be feasible, so we need to have Data distributed across multiple m|c.

↳ SHARDING.

HW: Sharding (vs) Partitioning.