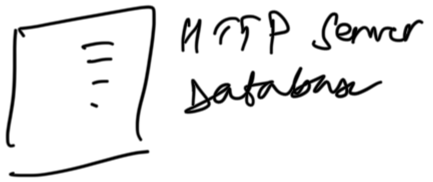
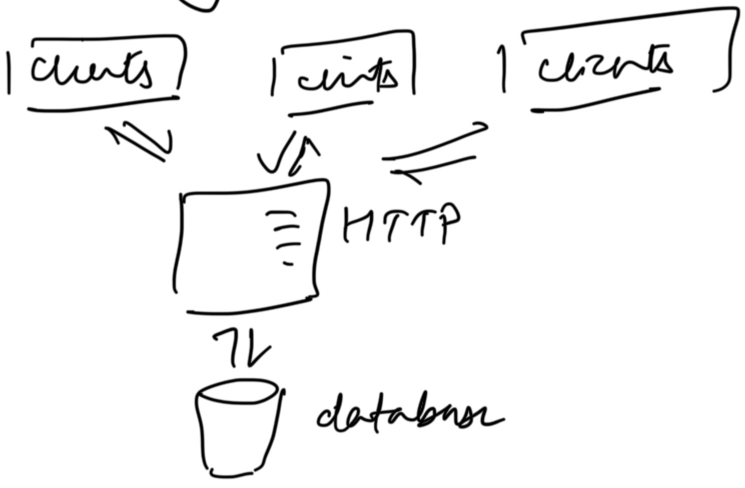
System Design Overview

Horizontal vs Vertical scaling

→ single server system
- is a single point of failure
- budget friendly and still has a place but less tolerant of failures

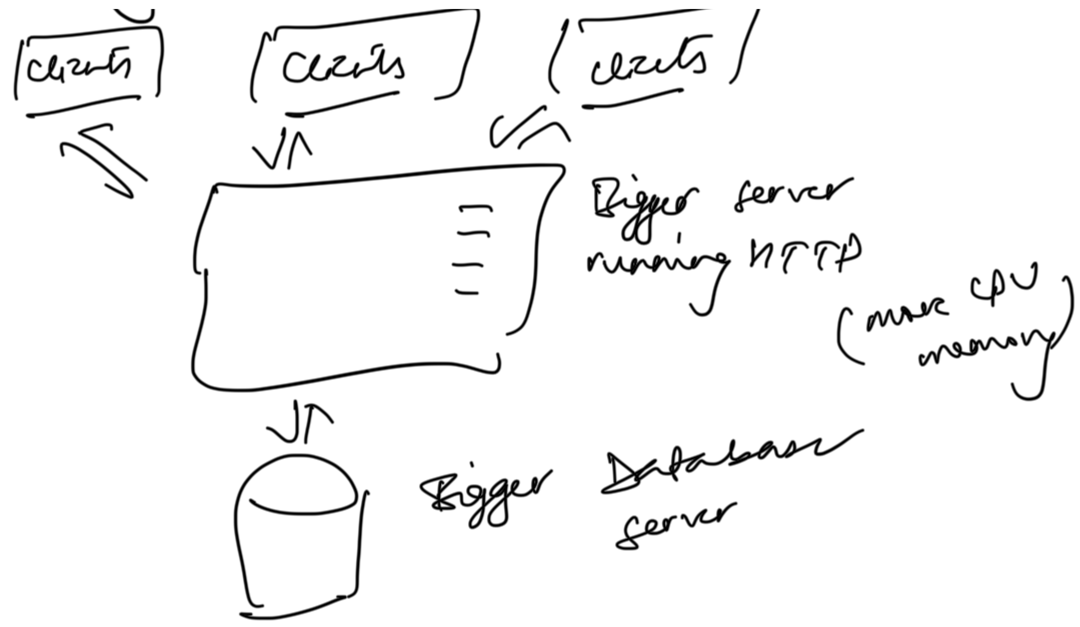


let us make the design a little bit more tolerant so that HTTP server and Database are hosted separately



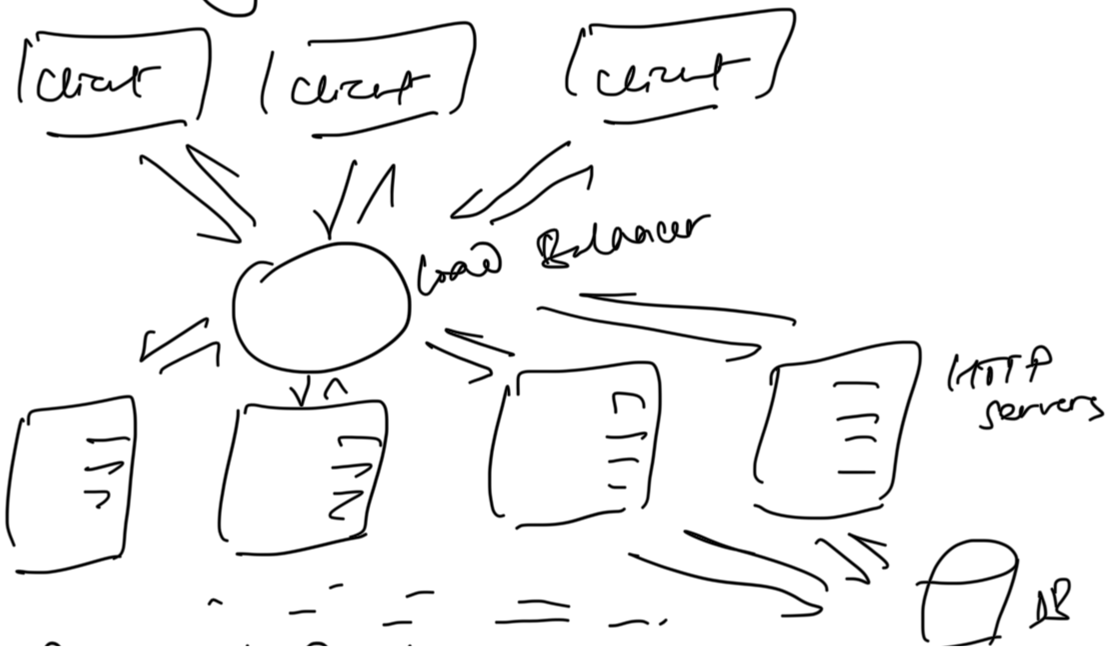
Little bit better because we can scale independently

Vertical scaling : use beefier (bigger) machine (CPU↑, memory↑)



- Still each machine is not tolerant to failures. If it goes down, service goes down
- Each machine can only be scaled so much i.e. there is a max of CPU and max of mem that you cannot scale beyond

Horizontal scaling: Add more servers



- needs a load balancer
- fault tolerant : one server goes down, request can be served still by other available servers
- can scale infinitely
- Downside : now we have more stuff to maintain

- Horizontal scaling is easier if the servers are stateless, meaning any subsequent request coming into the server should not rely on the server's state. (changes can be and will be made in database).
    - This means any next incoming request should be independent of earlier request
    - Because if they are dependent (and not reflected on the database), then there is no way to be certain which HTTP server the earlier request was served by and which server the next request will be routed to

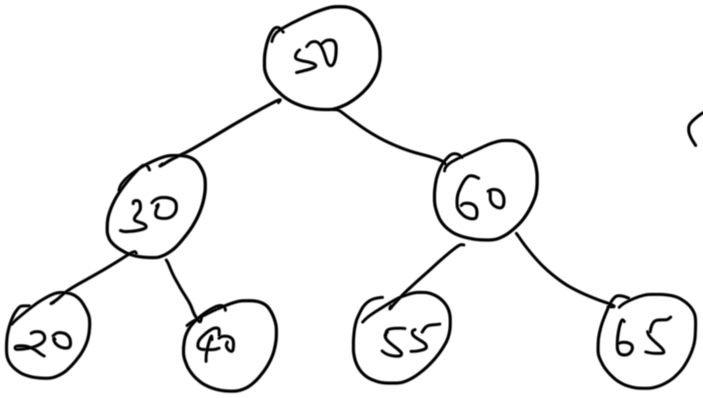
Binary Search Tree

- Binary Tree: each member has a left and a right child. each node has a most 2 children.
- Binary Search Tree: each left is less than

right child and parent. And right child is greater than parent.

- Rule applies to the entire tree, meaning any value to the left must be less and any value to the right must be higher

- Access in a BST is $O(\log n)$
  worst case is $O(n)$

- 

Binary Search Tree

Fail Over Strategies : Database

→ Servers can be
- hosted within your own company
- using cloud services
- using fully managed "serverless" services

→ How to scale DB
→ using " cold stand by server"
- create a periodic backup in other storage
- upsides:
  cheap

- cheap
- Downside
  - possible data loss
  - Downtime could be significant

→ using "warm standby server"
- create another database server that is a replication, and is always ready to handle request
- many databases allow replication. As a data is written/updated/deleted, operation is duplicated in both the places; may be after a delay
- There could still be a tiny window where you could lose data; but still a better option
- If the primary goes down, request will be directed to the backup/replication
- How is this handled in practice ??

→ using "Hot standby approach"
- Replication in real time
- thus its hot

→ using "Multi Primary"
- Writing simultaneously both both servers rather than relying on replication

Horizontal Scaling : Sharding