High-level-Design (Network - Protocols)

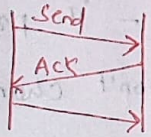
↓

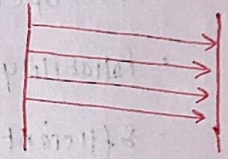
it defines the rules, so that two computers can communicate with each other

exg Languages.

As we learned → OSI ⇒

- Application ✓
- Presentation
- Session
- Transport ✓ (TCP/IP) (UDP) (fast) (Connection x)
- Network
- Data Link
- Physical





Application layer:-

↑ Connection Oriented

send → Recieving

(IMAP+POP)

Ordered x

fast ✓

Client server → HTTP, FTP, SMTP, Web sockets

sending

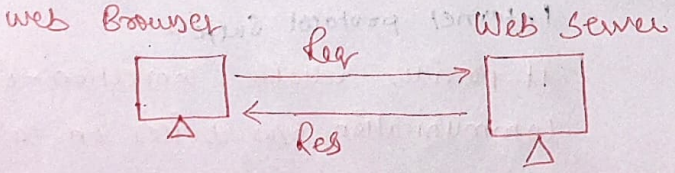
Peer to Peer → WebRTC

(Client can communicate with each other)

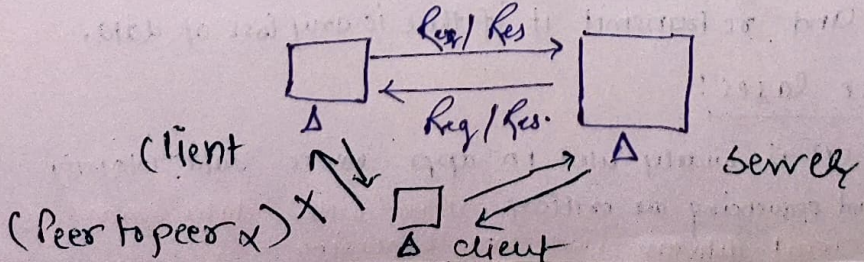
2 Connection

(Control Connection & Data Connection (Temp))

- Client server:-



- Web socket (Messaging App)



Network Protocols

it ensures efficient data transmissions ✓

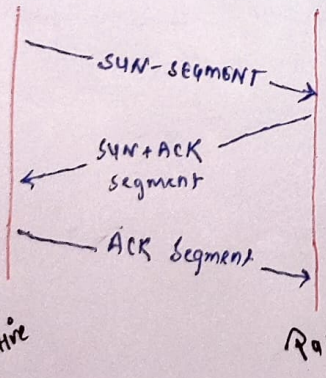
What ⇒ Network protocols are set of rules and conventions that initiates communication and data exchange between different devices o/w network, and network protocols define standards for data encoding, transmission and reception. ensuring that devices can understand and interpret each different's messages.

Commons are:- TCP/IP + HTTP

it impacts:-

- Interoperability :- it provides a way to share info.
- Reliability:- it won't changes by the time.
- Efficient :- it optimizes data transmission by minimizing latency and packet loss.
- Scalability:- Well designed protocols helps in scaling systems
- security:- SSL / TLS encrypt data during transmission.

1. Transmission Control Protocol :-



1. Role:- it's one of the core protocol of the Internet protocol suite.
it provides reliable, connection-oriented communication o/w devices on a network.
TCP operates at the transport layer of OSI

2. Functionality:- It establishes a connection earlier than data transfer, ensure the receipts of packets, and retransmit it if there is any loss of data.

3. Use Cases:-
TCP is widely used in app where data integrity and sequencing are critical along with data transfer, email-delivery and web browsing.

IP:-

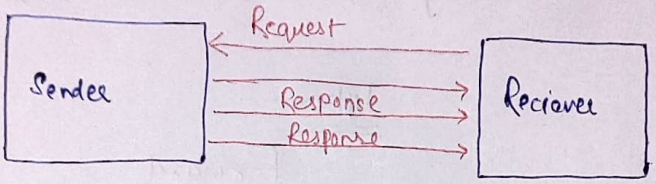
Role → IP is responsible for addressing and routing packets through networks.

Functionality:- it assigns a complete unique IP address to each device and determines the best suitable path for data packets to reach their stations

Use Cases:-

IP is fundamental to all networked system and is used together with other protocol for data transmission.

User Datagram Protocol:-



Role → it is same as TCP, but it's not reliable and connection-oriented UDP is simpler, connectionless protocol that minimal services

(Fire and forget Protocol) → cause it doesn't guarantee or order of packets

functionality:- it does not guagantee for packet delivery or sequencing

Use Cases:- UDP is typically used in real time app like VOIP, online gaming, and live video streaming, when low latency is important

HTTP and HTTPS

Role:- HTTP is used for tranfering hypertext files at WWW.

functionality:- it defines how messages are formatted and transmitted b/w different web server and clients.

Use Cases:- it is used in web browsers to retrieve and display web pages.

HTTPS adds a layer of security through encryption.

Secure Socket Layer

- it provide stable and secure connection.
- they encrypt information during transmission, making sure confidentiality and integ

File Transfer Protocol

- FTP is used for transferring documents between a client and server on CN.
- it allow user to upload, download and manipulate files on remote server

SMTP: Simple Mail Transfer Protocol and Post Office Protocol

SMTP is used for sending email and IMAP and POP3 are used for receiving email.

Proxy Servers:

A proxy server act as an intermediatory between client devices and servers.

Purpose:

- Content Filtering
- Privacy Anonymity
- Security and Access Protocol
- Load Balancing
- Caching

Types:

Forward
Reverse
Web Proxy
Public Proxy

Advantages:

- Enhanced security
- Improved Performance
- Content Control
- Load Balancing

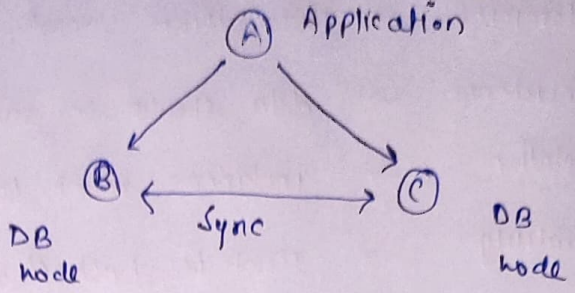
Disadvantage:

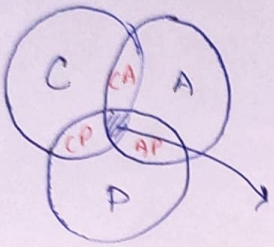
Latency
Configuration Complexity.

CAP Theorem:-

- Desirable property of Distributed System with Replicated data.

C - Consistency
A - Availability
P - Partition Tolerance.





In distributed system, we can't use all 3 properties

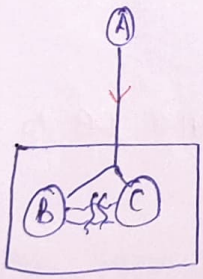
Cause →

1. if we make application consistent and available, so partition is not possible cause to make consistent the db instance must be connected
2. still you partitioned then either we should forget about consistency or availability.

Consistency:- B (a=4) and A writes on B (a=5), then B will replicate the same value in C (a=5), it means the changes must be done at each place in one shot.

Availability:- All nodes must be responsive, though data is inconsistent

Partition:-



Though B and C are not connected, still A is getting it's response it means it is fault tolerance.

"Always compromise b/w Consistency & Availability" - Brewer's Theorem

Based on Databases.

ACID

Atomicity → Transaction must be happen in one shot!

Consistency → data should be same on all node.

Isolation → instances must be work as indiriduatly.

Durability → must be backed up.

Base :- (AP) databases.

BA Basically Available

S → Soft State

E → Eventually consistent

PostGRES (CA)

- Uses SQL to interact with DB.
- it follows ACID and used in Banking, where consistency and Availability is crucial.
- it provides foreign key that allows multiple DB to communicate.

MongoDB (CP) :-

MongoDB is primary BASE oriented. but now it is adapting ACID it utilizes document to store data.

Cassandra (AP) :-

it sacrifice consistency at cost of availability

it is peer to peer

3- Microservices and it's patterns (Introduction) and Decomposition Pattern discussion.

Architechture

Monolithic

also known as legacy,

every functionality in Single App

DisAdvantage :-

- Overload IDE
- Scaling is Hard (Grow)
- Deployment and CI

Microservice

By this way we can make a service single responsible that makes.

- easy to scale
- easy to maintain
- easy to integrate and deploy

DisAdvantages.

if partition of services, not made correctly and still the distributed services are connected with each other then

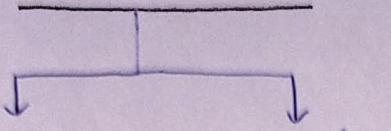
1- Latency will increase

2- Monitoring is difficult :-
(cause if anyone make it impact other services)

3- Transaction management

if any service Transaction got failed the complete microservice will be in consistent

Micro-Services



Micro → Small

Phase - I

To make micro there is a pattern, that we should follow →

1. Decomposition By Business Capability
2. De compose By Subdomain
3. Self Contained service
4. Service Per Team.

II.

Now after dividing the services the phase is to select database to keep data.

1- Database per service
2- Shared DB

III.

Now after selecting database, we will work on communication

1- API Compositions
2- Events Sourcing

IV

Now after all these we will integrate

1- API Gateway

Decomposition Pattern:

- Decompose By Business Capability
- Decompose By Subdomain. (DDD - Domain Driven Design)

- Decomposition By Business Capability:

Order - Online - Application →

Let's decompose by Business functionality.

1- Order Management
2- Product Management
3- Account Management
4- Login Management
5 - Billing Management
6 - Payment Management

- Decompose By Subdomain

- Order Management Domain - further divided into subdomain.
  - Order Tracking
  - Order Placing
  - Order Returning

- Payment Domain
  - Forward Payment
  - Reverse Payment

Microservices – II (Post Decompostion)

Handle Req & Transaction

1- STRANGLER 2- SAGA 3- CQRS

1- STRANGLER:-

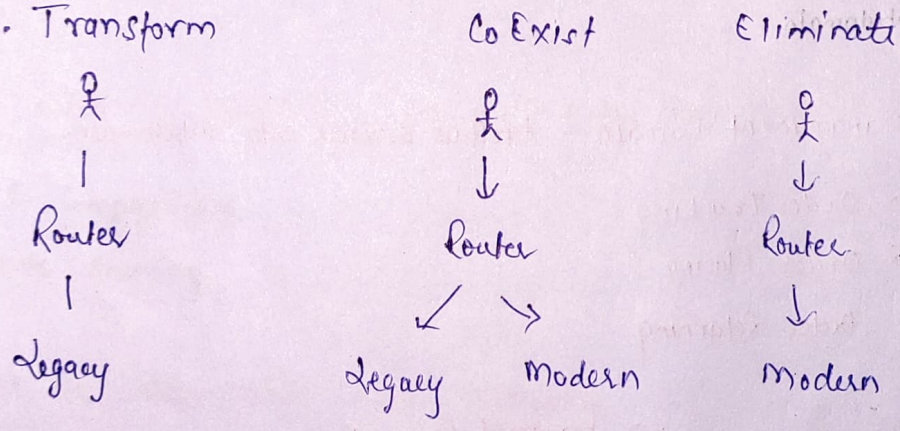
it solved a problem →

- How do you migrate a legacy monolithic app to microservices architechture ?

Modernizing an application by incrementally developing a new application, around the legacy app.

its an architectural approach employed during the migration from a monolithic app to a microservices-based architecture.

it derives ~~a way~~ its name from the way a vine slowly strangle a tree



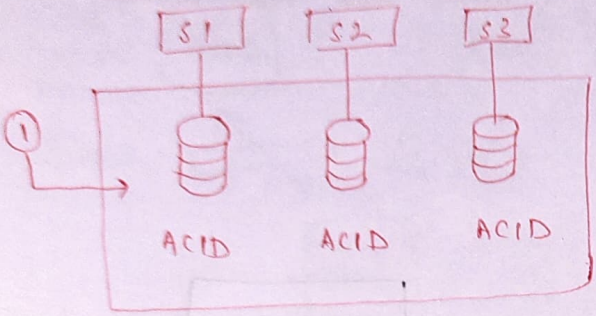
feature :-

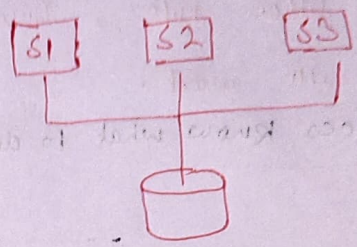
- Gradual Migration
- Co existence
- Stranging Behaviour.

SAGA:

Data Management in Micro Service.

- for Each individual service
- Shared Database



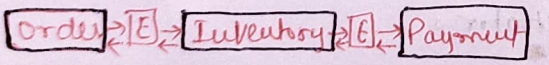


Let a order came and that will affect each db so is it possible to implement ACID on All together that if anywhere fail occur all db went to there earlier version

Solution → SAGA

Sequence of local Transactions

in SAGA we make events b/w local Transaction to keep track



whenever a transaction fail it sends a failure event for roll back.

1- Like if we have
- order
- inventory
- Payment

data at once place suddenly IM order came and we need to scale then we need to operate on single DB and that will impact other services down

2- Easy join ✓

3 → Transaction (ACID) ✓

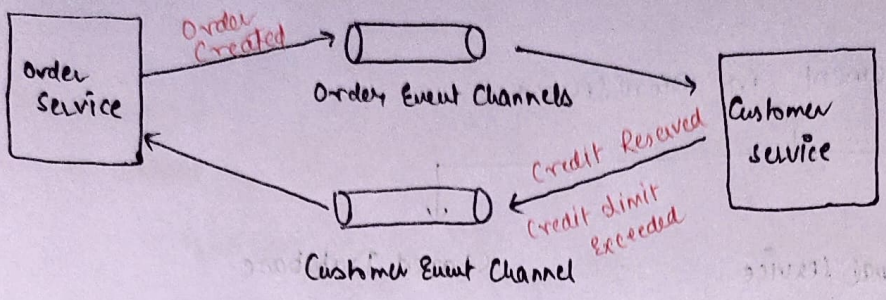
1 ↳ Choreography :-

→ each local transaction publishes domain events that trigger local transaction in other service

2 ↳ Orchestration → An orchestrator (Object) tells the participants what local transactions to execute

Choreography - based SAGA:-

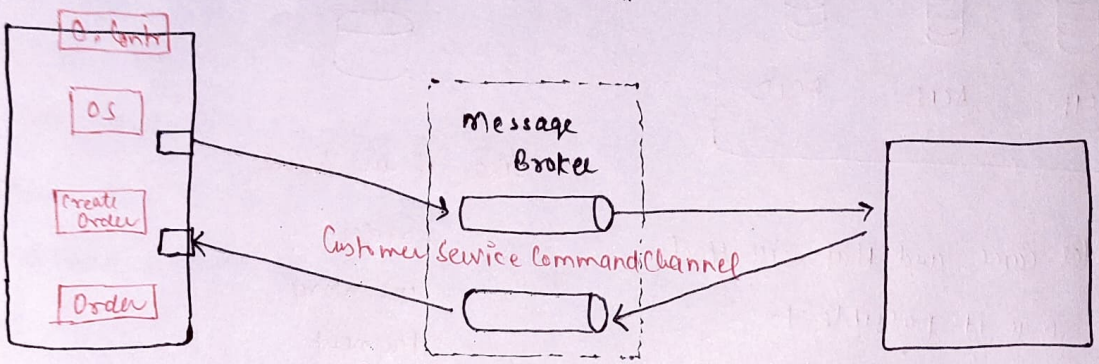
1.



One After Other Rule, Once will do its work then send message and other will work.

- Here services know what to do next

2.

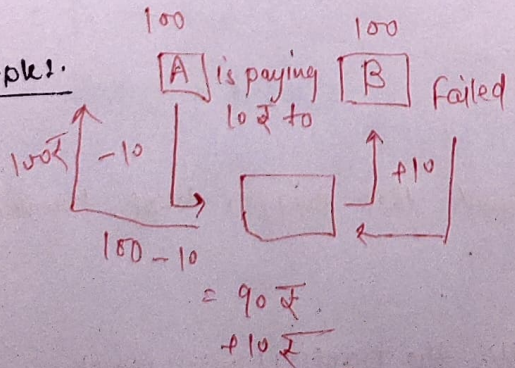


Here Order will go in pending state and a message sent to MB Once that message served by Customer service it sends the Response and on that basis the order will approve and reject

But the main thing is that Here Order Service just take care of its services it waits for commands by orcuastrator.

- Here Orcuasterator tells what is the process service to follow.

Example:-

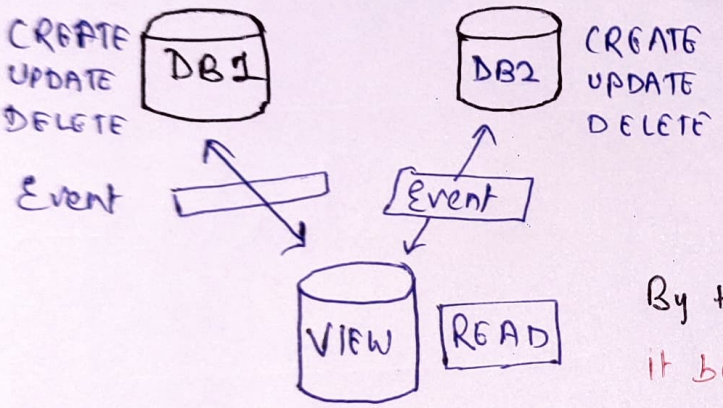


CQRS

Command query Request Segregation

(C.U.D) (Select)

To Segregate the Request of Command (Create, Update, delete) and Select (Read)

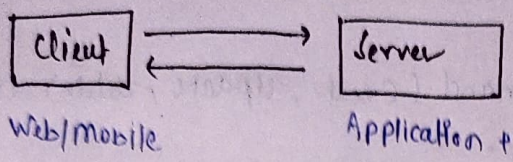


By this way there is no need of joining

It behaves like it joined but its independent.

Scale from Zero to Million Users in Detailed.

° Application is in starting phase when only developer uses the app. to test



} → College Projects

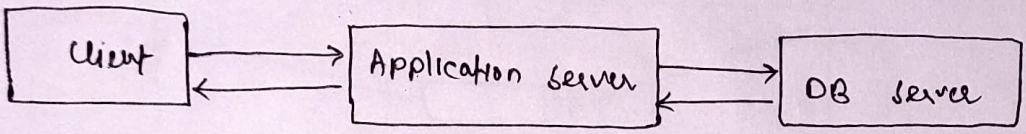
(This is also known as Client Server Architechture)

Steps:

1- Single Server - ✓ Make a server and deploy app on that server

2→ Application & DB seperation

(Now we distribute our application and db from a single server)

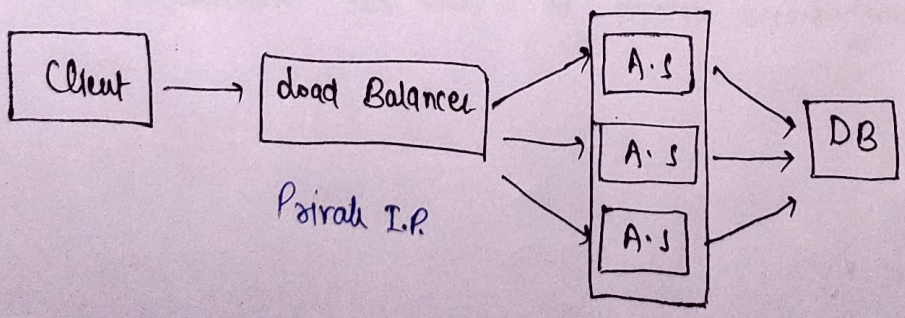


Mid-Tier

By this way we can scale our logic and db independently.

3→ Load Balancer + Multiple Application Server.

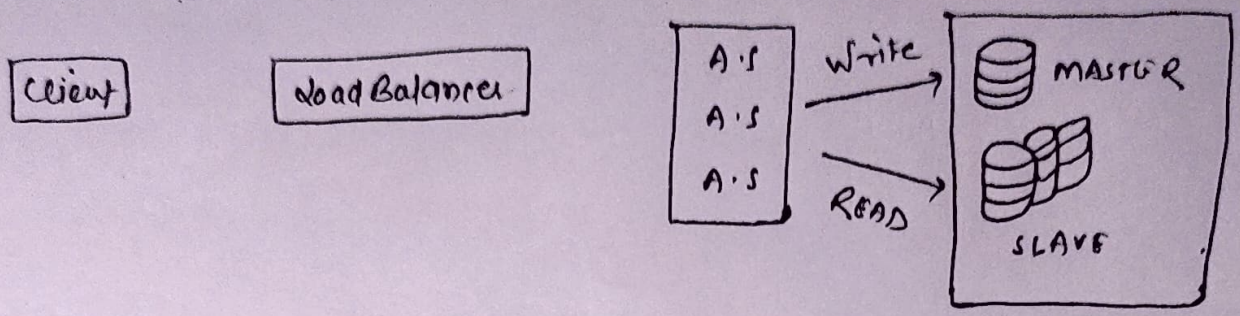
→ whenever the request are coming in bulk, then we need to handle traffic of application server by changing the mid-tier. which is we place multiple application server and to route the req to application servers.



So it improves security

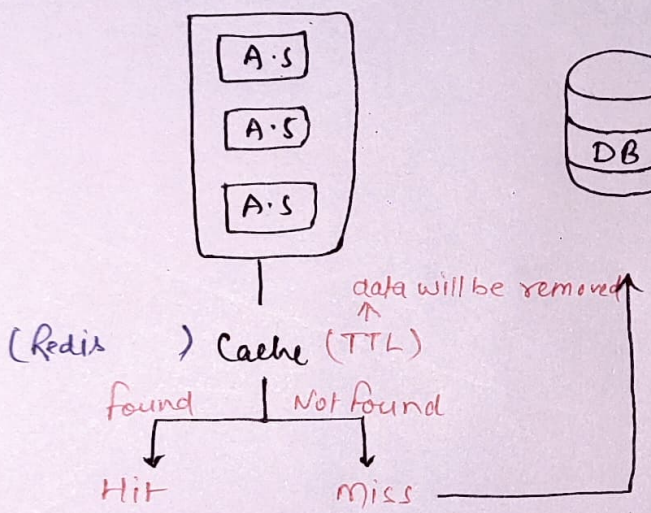
4. DataBase Replication

it's a Master - Slave Architecture DB.



5. Use of Cache

Cache is used to improve the time and performance, so here we add cache after A·S, and before DB to save DB calls.



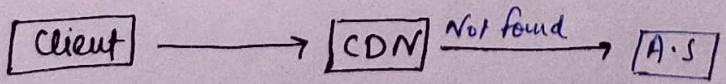
6. CDN:- (Content Delivery Network)

CDN does Caching Butt All those who do caching are not CDN.

it improves the latency for All over the world users..

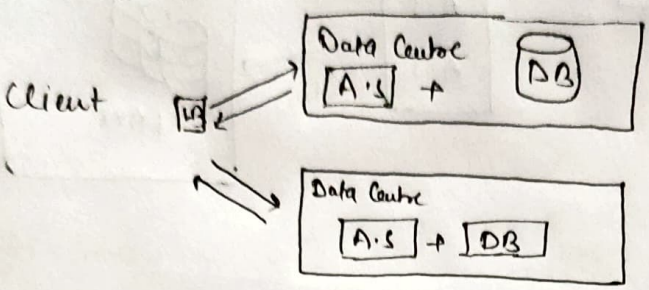
it stores static data at each specific CDN from main DataBase.

also it improves security cause direct access of db removed.

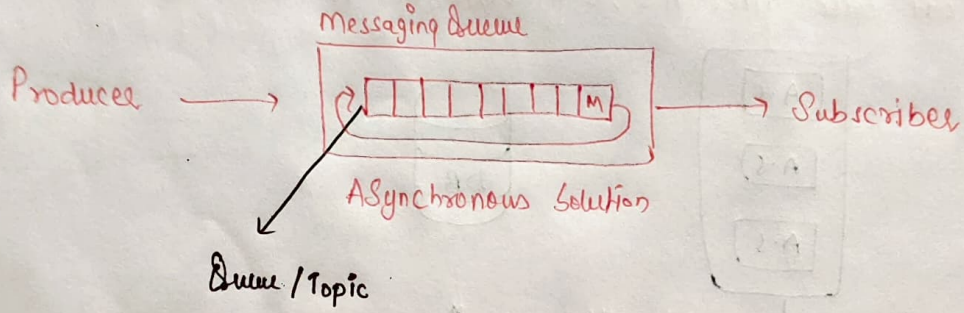


7. Data Centre :-

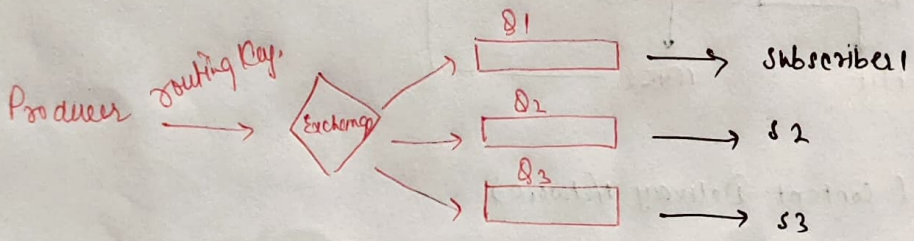
Data Centre is a way to place your db different places to make it easily available.



8. Messaging Queue :-



- Exchange :-



① Direct :- Routing Key == Binding Key.

② FanOut :- Send in All Queue.

③ Topic :- it can send in multiple Queue

9. DataBase Scaling :-

↳ Vertical + Horizontal

Vertical → increase hardware, increase limits of RAM + Rom + Processor

Horizontal ⇒ Add multiple Instances

↳ Implementation is Sharding

Horizontal	Vertical
① Divide into multiple Tables on the basis of Rows.	① Divide into multiple Tables by Column.
$T_1 \rightarrow A-P$	$T_1 \rightarrow C_1 - C_6$
$T_2 \rightarrow 8-2$	$T_2 \rightarrow C_7 - C_{10}$

it can go Hierarchical + there is no joining possible

To solve joining problem remove redundancy + dependency and make Table Normalized

Hierarchical → We can solve it by Consisten hashing

Consistent Hashing:-

It helps in Horizontal Sharding.

Hashing :-

hashing is a process to convert data in fixed-value, for fast data access we use it.

it convert arbitary length values to fixed length value

abcdefghij → Let Hashed value is 1451

Now there is also mod hashing to adjust data in a fixed place.

Let our array is length of 7 so the value for 1451 will be 907

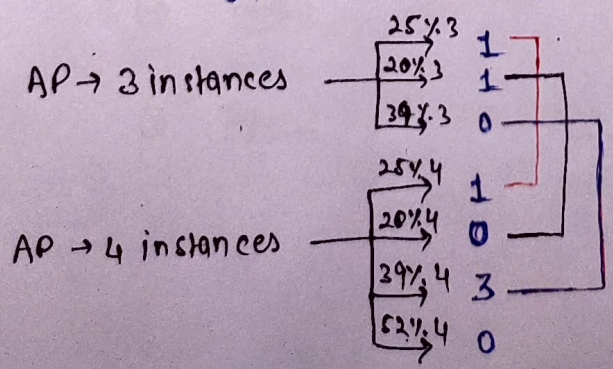
2 ✓

But Now the problem comes when our last mod value is not fixed cause in horizontal sharding there can be possibility of multiple instances.

we use hashing mainly on two places.

1- Load Balancing Application servers.
2- Load Balancing Database Sharding.

Now the problem is Rebalancing cause whenever we change the size of Array, we need to replace all the values to different node.



Now what we route on the basis of id and then Rebalancing → Million

The Solution is Consistent Hashing

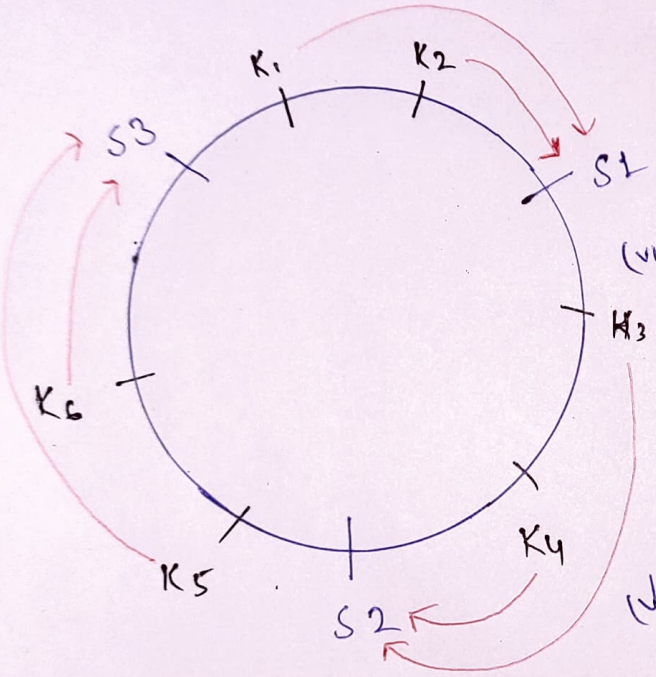
it scores $\frac{1}{n}$ % Rebalancing and that we consider as good.

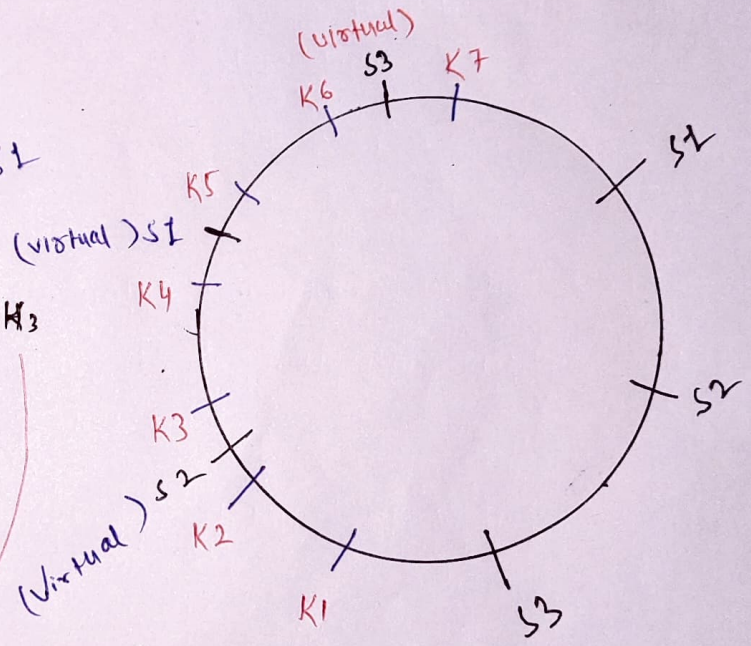
How Consistent Hashing Works ??

it works on Rebalancing, and even instances go down or up it make rebalancing ratio low.

① Take Virtual Ring

→ and place your server on mod hashing





→ in consistent hashing we serve keys by next servers.

Servers are continuous and this will be disadvantages, which can be saved by replication virtually

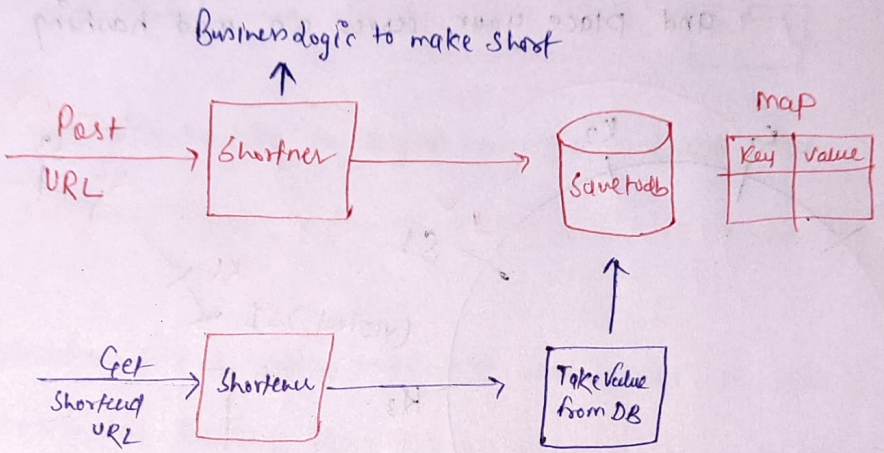
Design URL Shortner!

Here we will design a url shortner to remember any url easily and it is short.

So now what is our requirement ??

- So here we have to generate a short url by which my mainlink can accessed if there is a page which has long path and multiple param, so can short that url also

Rough Idea!



- Requirement analysis
  - what should be the length of shortened link? / How short
    - As much as you can.
      - ↳ now decide what is the traffic

        10M URL/day ⇒ 3650M URL/year

        and the duration of each link should be 100 year

        3650000M URL/year ⇒ 365×10^9 URL/year
    - now what are the character we use

      (0-9) | a-z | A-Z

      10 + 26 + 26 ⇒ (62)

      at a place we can use 62 character

      62^6 around 56B

      62^7 around 3.5 Trillion ⇒ we finalized 7 chars.

How to generate these hashed value -

↳ Use Hash function (MD5, SHA-1)
↳ Base 62

MD5 → 128 bit
↓
16 bytes
_ _ _ _

in hexadecimal we use 4 bits and
and 8 bit will generate only two hexadigits.

if one bytegenerate 2 hexadigit then 16 byte generate
16 × 2 = (32) hexadigits and it will take 32 chars
that is too much
and SHA-1 also uses 160 bits

→ So here we uses Base 62.

1. Id Generator
2. length could be different

Now here the problem is that we can't store all id's into single db.
So to overcome this we use 1. Ticket Server → single server was disadvantage
Single Point of failure

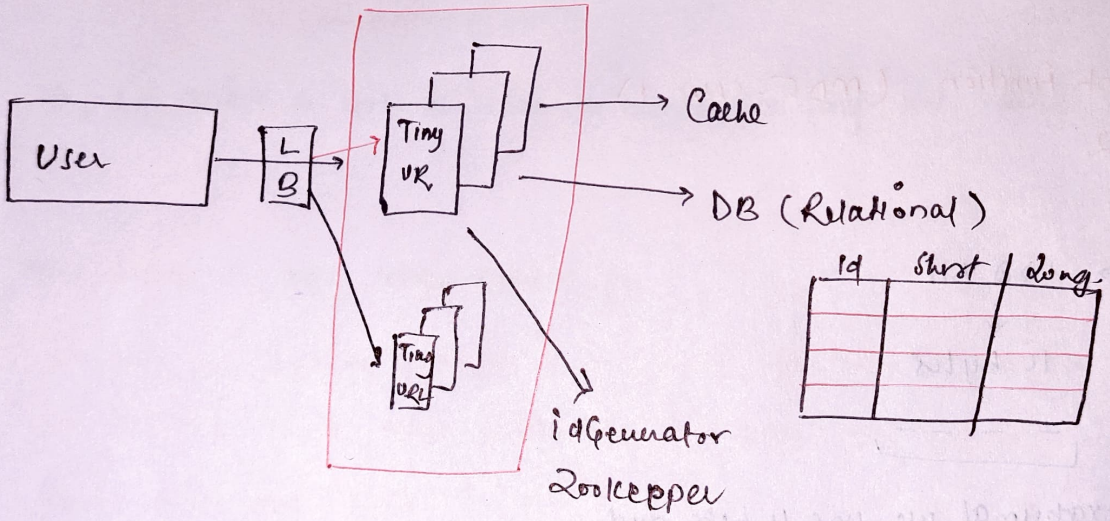
2. Snowflake:

Timestamp works →
Same Time was disadvantage.

Best solution

(3) Zookeeper:

Distributed Application can coordinate with each other reliably.
it put values in Range
Range1 → (1 to 1M)


User
LB
Tiny URL
Tiny URL
Cache
DB (Relational)
id
short
long
idGenerator
Zookeeper

Back-of-the-Envelope Estimation

Estimation of Facebook

- What is Back-of-Envelope?
  - → To know the cost/resource estimation of the components, and with this we come on our decision that components should be used or not.

- Consideration:
  - → Rough Estimation (T-shirt Size) (1)
  - → Don't spend too much Time (2)
  - → Keep the Assumption Value Simple (10, 100, 500)

- Cheat Sheet

	Traffic	Storage
3 Zero	Thousand	KB
6 Zero	Million	MB
9 Zero	Billion	GB
12 Zero	Trillion	TB
15 Zero	Quadrillion	PB (PegaByte)

Data

Character - 1 Byte (ASCII)
2 Bytes (Unicode)
Long/Double → 8 Byte
Image → 300 KB

→ No. of servers ??
→ RAM ??
→ Storage capacity
→ Trade off

formula

(1) XM users X YMB = XY TB Database
(2) 5M Users X 2KB ⇒ 10 GB Data-

Estimation of Facebook:

Traffic Estimation:

Total User :- 1 Billion ✓

Daily Active Users :- 25% of total user 250m.

- So Now we will calculate how many queries are coming

Assume → Every User → 5 Read & 2 Write (7 Query)

(250m users x 7 query) / ~~864~~00 Sec
(D.A.U)

≃ 100000 Sec.

10K query / 1second ✓

- Storage Assumption:

Every User 2 post (250 charaters each post)
10% of User Image (300kb Size)

1 Post is 250 char. ⇒ 250 x 2 bytes = 500 bytes
1 User → 2 post = 1000 bytes = 1KB

250m user ⇒ 1KB x 250m = 250 GB storage for Post only

for image ⇒ 25m (10% of 250m) x 1 image (300kb)

75000 GB

7.5 TB ≃ 7 TB Data storage for A day
to store ~~1 Post~~ 1 Image.

Total storage ⇒

(2000 day Apx.) 5 years ⇒ 2000 x 250 GB → Posts ⇒ ≃ 500TB
2000 x 0TB ⇒ Images ⇒ ≃ 16 PB.

RAM ESTIMATION

for each (D.AU) User last 5 post we are keeping it in cache

1 Post ⇒ 500 bytes

5 posts ⇒ 2500 bytes

Total Active D.A.U. = 25cm x 2500 byte

25x25 x 6B

625 GB Required for Memory Space

if One machine can hold 75 GB in-memory data.

then RAM ⇒ 10 Machines we need.

Latency ⇒ 95% of time ⇒ 500 ms.

10K Req per second

and One Server has 50 threads

in 1sec can server 2 request

so if a server has 50 threads then it can server

100 Req/second

$\frac{10K\ Req}{100\ Req}$ ⇒ 100 Servers

we need to accept Req

Number of Server ⇒ 100

RAM ⇒ 750 GB ⇒ 10 machines.

Storage ⇒ 17 PB Apx.

Now on the basis of this we decide (CAP) ✓

Here we compromise with consistency.

Design a Key Value DB || DynamoDB

Amazon use Dynamo DB in Add to Cart feature

To achieve Goals this database is designed

1- Scalability
2- Decentralization
3- Eventual Consistency

→ Steps

1- Partion
2- Replication
3- Get & put Operations
4- Data Versioning.
5. Gossip Protocol

1) Partition:

We think first thing that why not we can use mapping

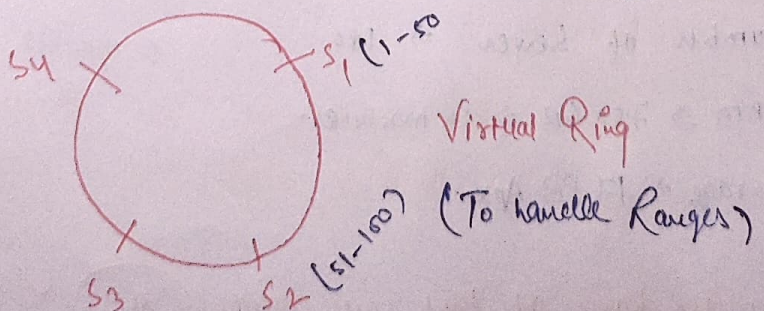
A person map his add to cart list

But the disadvantage here is how can we store this mapping in multiple systems.

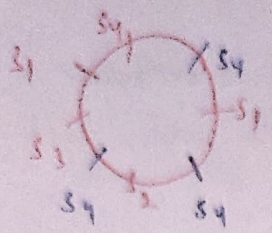
Now mapping made disadvantages for Scalability.

Cause when users will increase we can't handle all users data.

Sol^n → we do partioning of Request by consistent hashing



To prevent Bombading of only one Server → (Virtual Nodes)



To prevent of single point of failure.

we use Replication →

eg:

CAR - 45 served by S_1 (Coordinator)

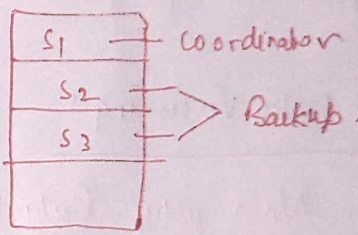
To maintain Availability we Replicate

(N-1) Servers ✓

we make copy of CAR in S_2 and S_3 ✓

→ Always Choose Different Datacentres.

Preference list



③ Get & Put Operation

1- Put Operation

Put(CAR) → find Key (45)

45 hrd in S_1 Keep it and now to replicate by asynchronous manner

(45-car) placed in S_1, S_2, S_3 → Async Manner

$R + W > N$

wait for success Res.

2. Get Operation:

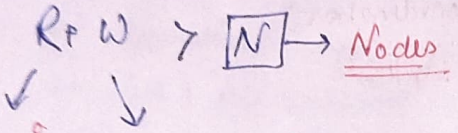
whenever we get Request it has 2B to get this response

2B → ① Generic ② Partition Aware

So, now whenever we use Generic dB, As it can go on any Server, now in this case our Preference list works and co-ordinator serves to that Req, if coordinator is down, then other nodes which are top of Preference list Take care.

Generic Load Balancer :- High latency & Simple to Implement

Partition Aware :- Low latency :



Get operation
Response successful.

Write Operation
Successful Res.

4. Data Versioning :-

Now why Data Versioning,

Let's Assume,

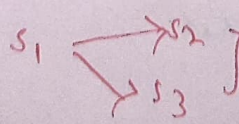
① put (CAR-45) → S_1 [45- CAR]
id
Replications in are S_2 S_3

② put (CART-45) → S_1 [45- CAR] & (Closed Now)
So it go to Preference list S_2

③ Now Server is up and you call get Req

S_1 has [CAR-45] Not updated

on this time

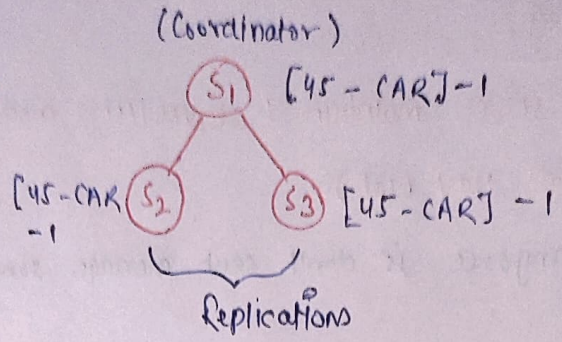


To implement this Vector Clock (Server, counter) static

1- Put Request

(45 - CAR)

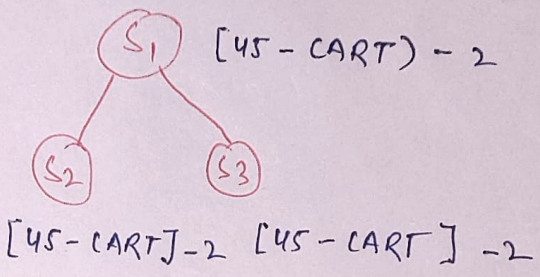
All Server is on



2- Put Request

(45 - CART)

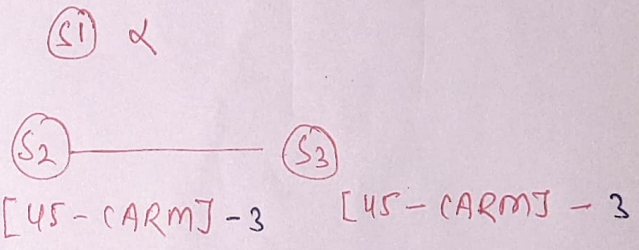
All Server is on



3- Put Request

(45 - CARM)

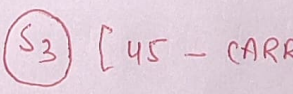
S_1 is down



4- Put Request

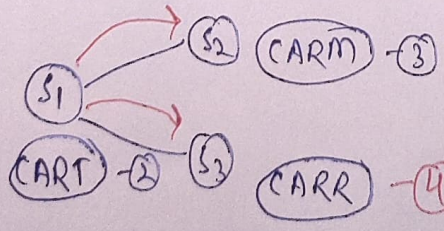
(45 - CARR)

on same time with (3)



5- Get Request

All Server is up



AP

6→ Eventual Consistency → cause here we can't compromise with Availability & Partition Tolerance so we compromise Consistency.

SQL or NoSQL ??

SQL Structured Query Language (RDBMS)

(Table (Row - Column) Relationships.)

Structure: →

Here we need to define predetermined schema

Nature:

Here we need to place complete data at a server only, we can't keep data at multiple servers,

- Centralized / Concentrated

Scalability:

- Horizontal (place data at multiple places, and this doesnot well support in SQL)
- Vertical (RAM↑, STORAGE↑)

Property:

A → Atomicity
C → Consistency
I → Isolated
D → Durability

→ Data Integrity / (Complete + Accurate)
Consistency (Same at every place)

NOSQL

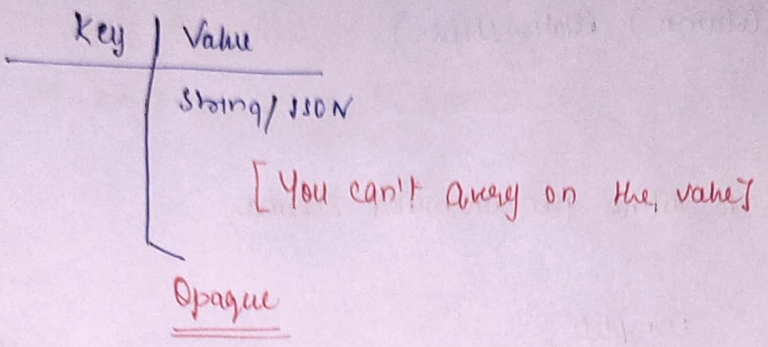
Non Relational / No SQL / Not Only SQL

- **Structure:** Unstructured data.

1- Key-Value
2- Document
3- Column Wise
4- Graph DB

(1) Key - Value Store:

Data stored in Key Value format where there is opaque nature of value we only find values by key only.



(2) Document DB

Key	JSON

Same as Key-Value, but here you can query through values.

Value > 30

3. Column Wise DB

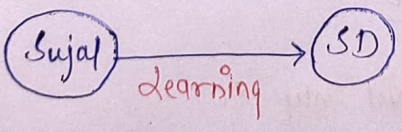
Here data is stored in Column-Value format like form.

Name : Sujal Sharma

4. Graph DB

Here we use Node an edge.

Relationship



Nature

It is distributed in nature, cause data we can distribute on multiple nodes

Scalability:-

Horizontal

Property:-

BASE

BASICally Available, Safe State, Eventual Consistency.

SQL	NoSQL
◦ Flexible Query functionality	◦ No complex Query.
◦ Relational	◦ Non Relational.
◦ Data Integrity (ACID)	◦ No consistency.
◦ Availibility is low. Just to provide consistent data.	◦ High Availability / Performance / Search Query / Less Consistency.

High level Design of Chat App.

Whatsapp, discord, Telegram, Slack, Fb messanger.

1- Requirement Gathering :-

Functional

① 1-1 Send/Recieve message (Text)

② Group Message Support

③ Last Seen Online/Offline

④ User login Authentication

Non - functional

① Scalability
↳ Huge Traffic

② low - latency

③ Availability

Back of the Envelope

Total Users ⇒ 2Billion

DAU ⇒ 50M

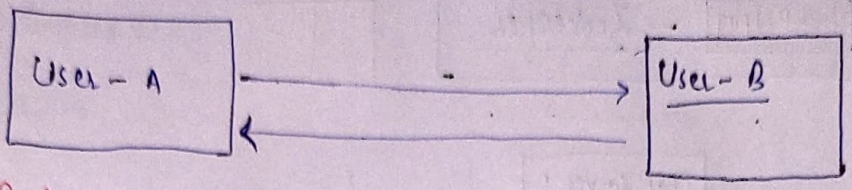
1 user can send 10 messages to 4 people in One day -

50M (users) x 40 messages 2000 M messages ≈ 2B messages per day

1 Message ⇒ 100 Byte

2B x 100 Bytes ≈ 200 x 10^9 Bytes ≈ 200 GB / day.

for 10 years ⇒ 2000 GB 2TB X 365 ⇒ 730 TB



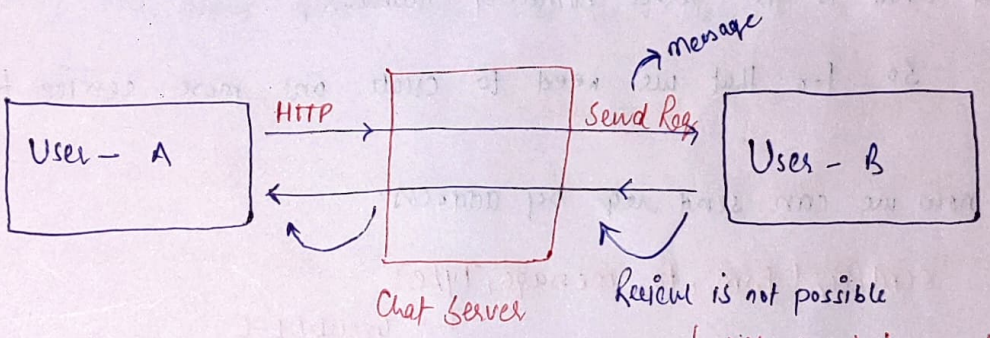
IP Address

IP Address

They can talk with each other directly by peer to peer, but there is no chat history.

- not scable
- not group
- not fully trailable

To provide this we need a common place which can hold this details and take care of us.



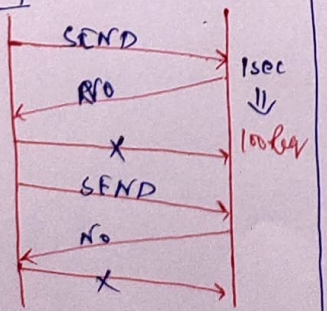
Protocol

HTTP fails — So normally for communication we use HTTP protocol, where we send a req and response came and terminate

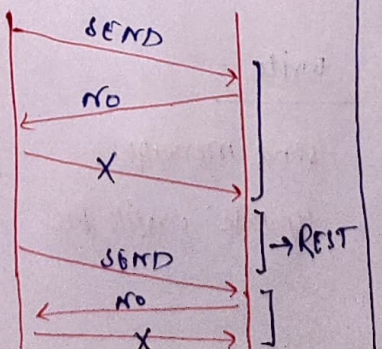
5. to reciev messages

we use improved protocols

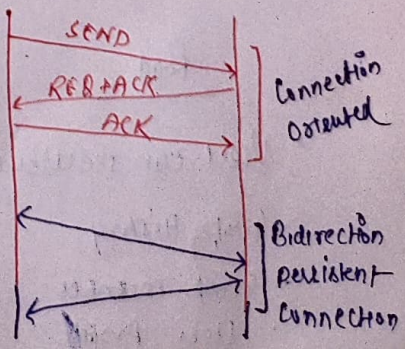
Polling

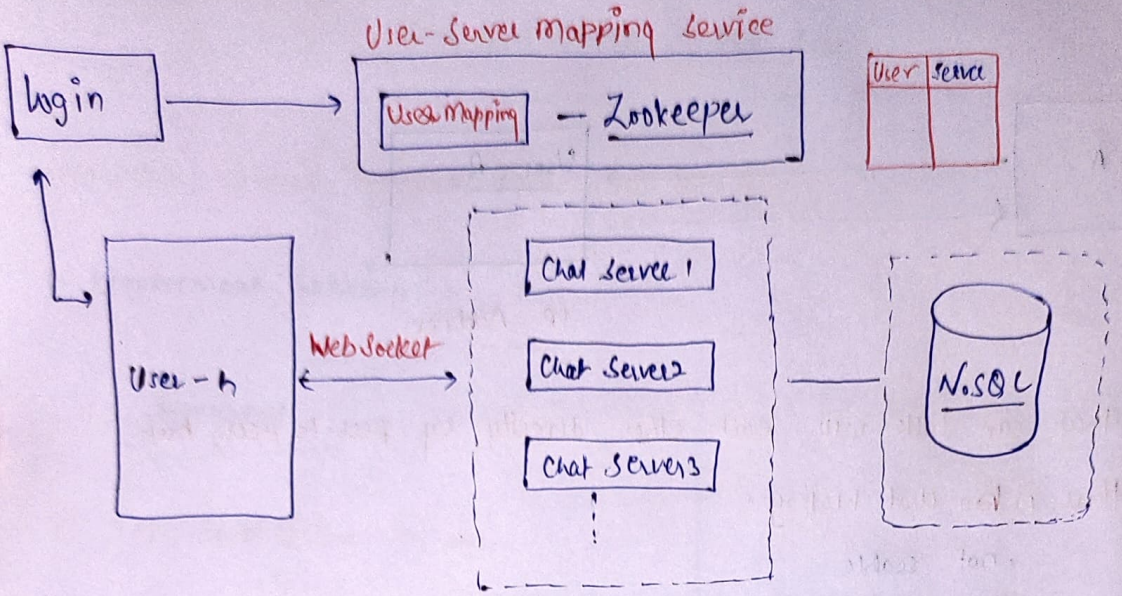


long-Pulling — Pushing



WebSocket (Bidirectional)





Now Any user can connect with any chat server then why ~~server~~ client sends req to any other client how send client server know that what is the server number of Reciever's,

So for that we need to create one more service for mapping

So now we can send req by address.

SendReq (from, to, message, Type)

↳ Group / 1-1

Now for Chat History:

To store any thing we need storage, that database.

Question is:

SQL or NoSQL ??

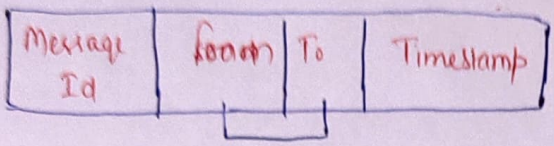
Read	Write
• Usesl can see user2	send message
• Group History	Update profile Pic
• Group member	
• User Profile	

Any Complex join? → No

Search → Low latency

} → No SQL

Multiple chat messaging services uses Column wise DB (Cassandra)



we can do partition based on From to To

ordering to place in nodes using message Id.

whenever there are multiple partition take care of unique id's

Now let's talk about when any chat server down. or user is offline

So whenever user login the login check with user mapping and if there is no server assigned, it assign a new one, and after mapping server will check that in db, there is some data for particular user.

Group

GroupId – Partition Key.

Message Id	group Id	User - Sender	Message	Time stamp

Last Seen!.

Presence Sys to check frequently user is online or not through Chat Servers.