Basics of System Design

1) Horizontal Scaling - increase more machines

2) vertical Scaling - increase capacity of a machine

3) Scalability - Handle more requests using 1 & 2

4) Load Balancing - Round Robin, Weighted RR,
Least connections, Weighted LC
IP hashing

5.) consistency - Eventual consistency (FB Likes)
Strong consistency (Bank Transactions)
Tunable " (e-commerce)
Serializable
Quorum Based (

6) Availability

7) CAP Theorem (consistency Availability
Partition Tolerance)
CP or AP

8) Latency

9) Single Point of failure, -) Redis to eliminate
Kafka or Zookeeper

10) ACID
(Atomicity, Consistency, Isolation,
- Durability

11) Consistent Hashing - IP Hashing (load Balancing
circular array
add Node Remove Node
virtual Node
To eliminate SPOF
using Redis

12) Rate Limiting:-

1) Token Buckcet
2) Leaky Buckcet
3) Fixed window counter
4) Sliding window log
5) Sliding window counter

13) Consensus Algorithm:

2/3rd of Nodes are honest then consensus is reached

14) Gossip Protocols: Nodes communicate to know the states

15) Robust

16) Service Discovery: Gateway checks the service name in Global Cache Distributed to identify the service name & ip. Service Discovery updates cache

17) Contracts

18) Heart beats, Monitoring (costly)

19) APIs: REST APIS (CRUD, POST, GET, PUT
PATCH, DELETE

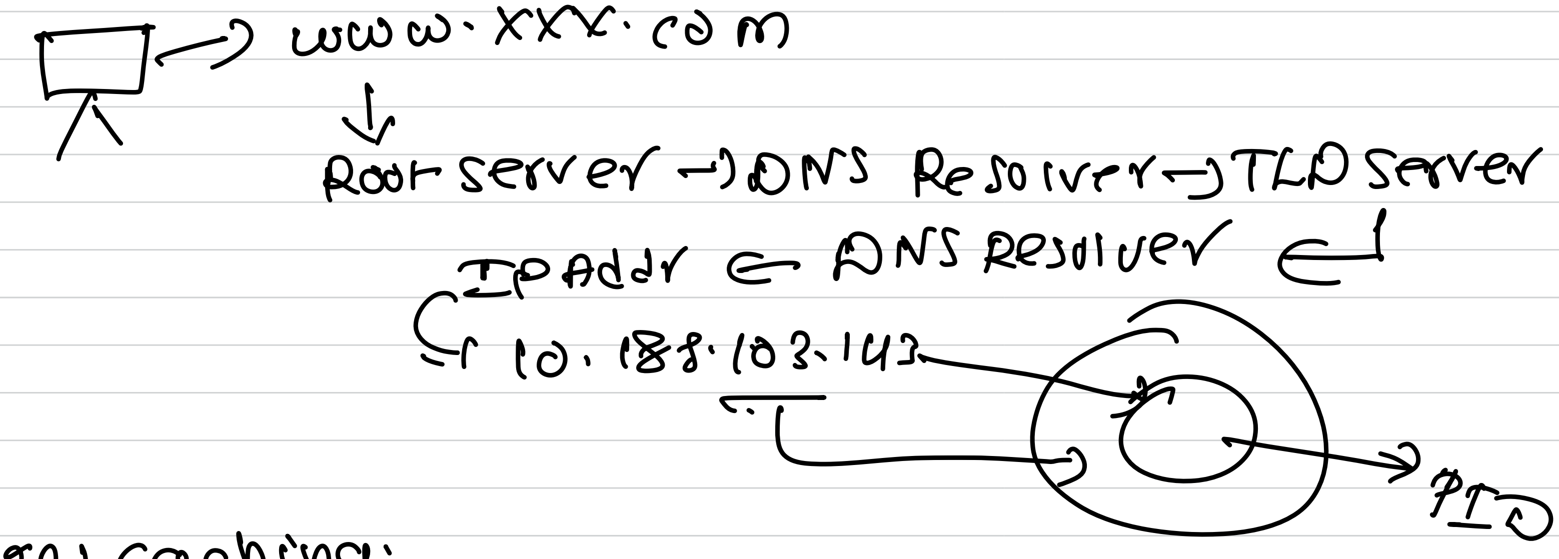
SOAP APIS (objects transferred)

GRAPHQL APIs
gRPC (google Remote procedure calls)

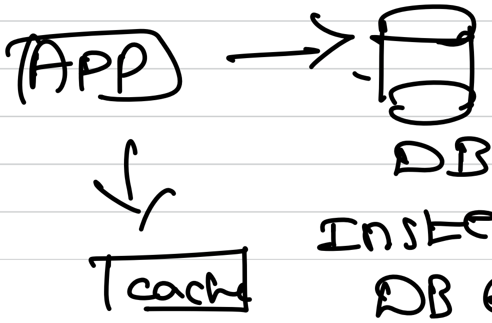
Public & private APIs

20) CDN:
Distribute Network in geographical locations.

21) DNS:



22) Caching:



Instead of looking into DB everytime, look into cache

1) IN-Memory cache (Redis)

IN RAM (sessions, frequentey used objects)

2) Distributed cache:

cache in multiple servers

Redis cluster

3) Client-side cache: cookies

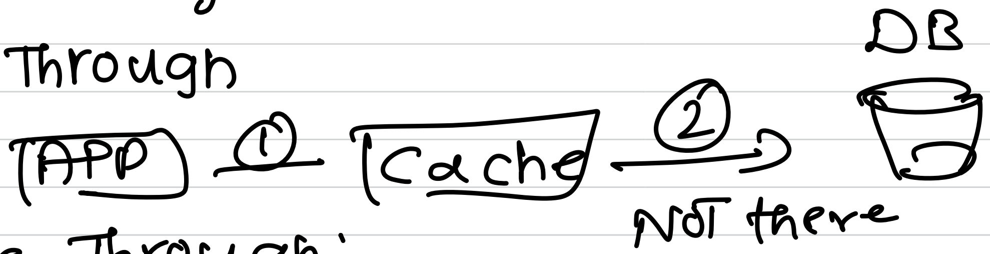
JToken

4) Data base cache:

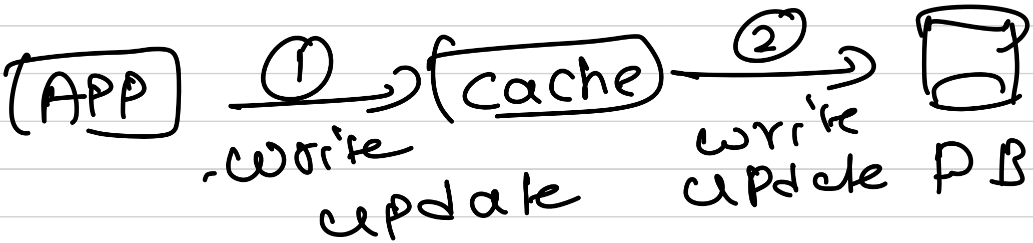
2>)

Caching strategies:

1) Read Through

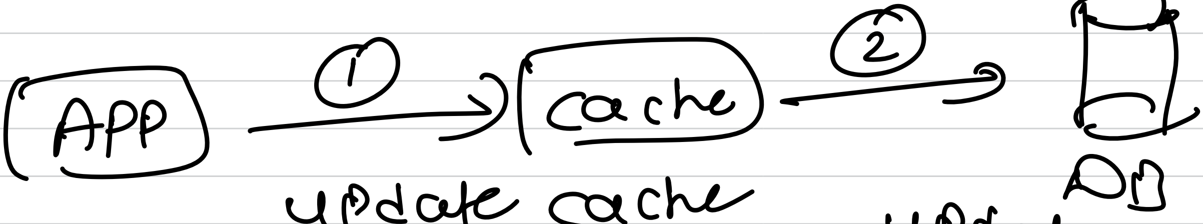


2) Write-Through:



update Both at a time (consistency strong)

3) Write Back cache:



update DB later

5) write Around:

update DB then eventually cache

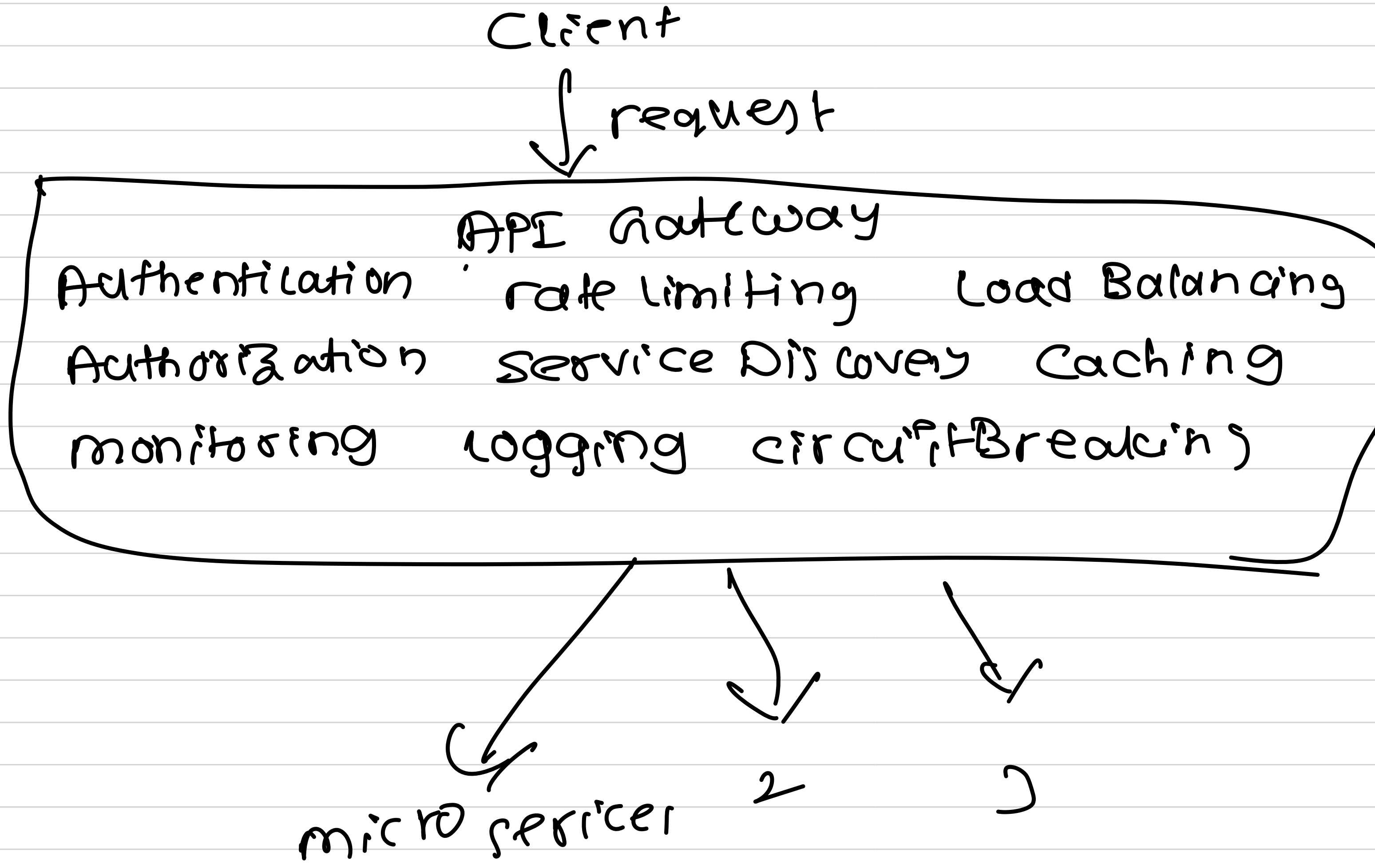
Eventual consistency

4) cache Aside: Read write both DB & cache

24) Cache Eviction Policies:

1) least recently used
2) least frequently used
3) First in First out
4) Time To Live

25] API Gateway:



Not tightly coupled mother fucicer

26) Load Balancing:

1) Round Robin (No.of requests servers in round robin fashion)

2) Weighted Round Robin

↑ Weight ↑ No.of requests

3) least connections

↓ connections ↑ No.of requests

4) Weight least connections

↑ weight ↑ No.of requests

5) least response time

↓ response time ↑ No.of requests

6) IP Hashing

consistent Hashing

Normal Hashing

27) Types of Databases:

1) SQL - ACID Transactions (RDMS)

2) Key-value store: Redis, Dynamo DB

Eg: session, cache

3) Document Databases:

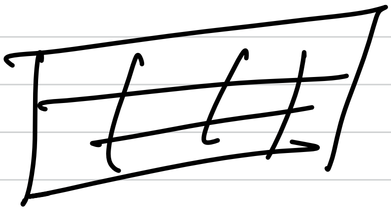
JSON, XML, BSON

Eg: MongoDB, CouchBase, Apache CouchDB

4) Graph DB: Social media recommendation system

Ex: Neo4J, Amazon Neptune

5) Wide-column stores:



Dynamic columns

Eventual consistency

high write throughput

Ex: Apache cassandra

Apache HBase

Google BigTable

6) In-Memory DB: online gaming, Bank Transactions,

In RAM costly

Ex: Redis, memcached

7) Time-Series DB: Financial Trades
Kite
Zerodha

Ex: prometheus,
Influx DB,
Timescale DB

8) object oriented DB:

Store & manipulate objects
OOPS following langs

Ex: Object DB, db40

9) Blob DB: (Binary Large Objects)
To store audios, videos, images

use cases: CDN, Backup,
Big Data Storage

Ex: Amazon S3
Azure Blob storage
HDFS

10) Ledger DB: Blockchain

cannot alter once updation or transcaction is Done.

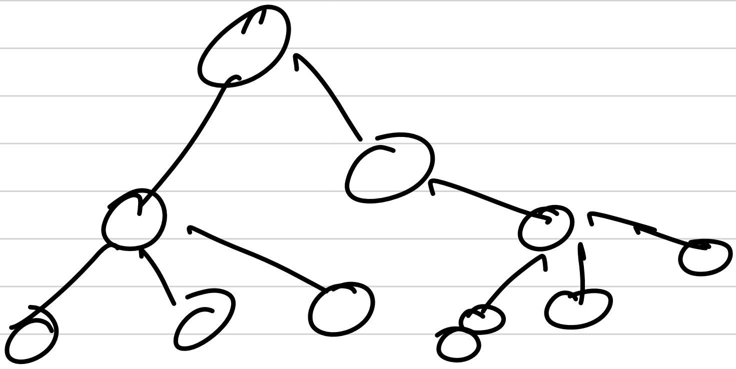
Eg: Amazon Quantum Ledger Database

usecase: supply chain

voting systems

12) Hieararchical DB:

Tree structure

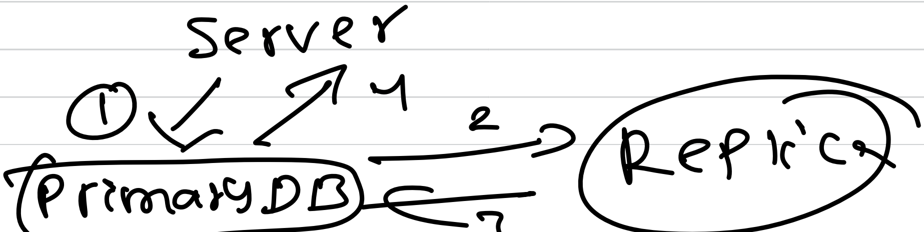


usecase: windows registry

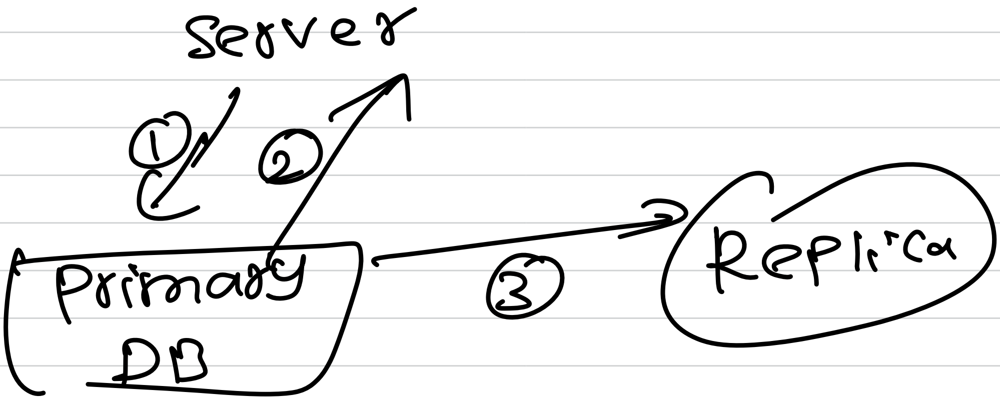
File system

28) consistency in Distributed systems:

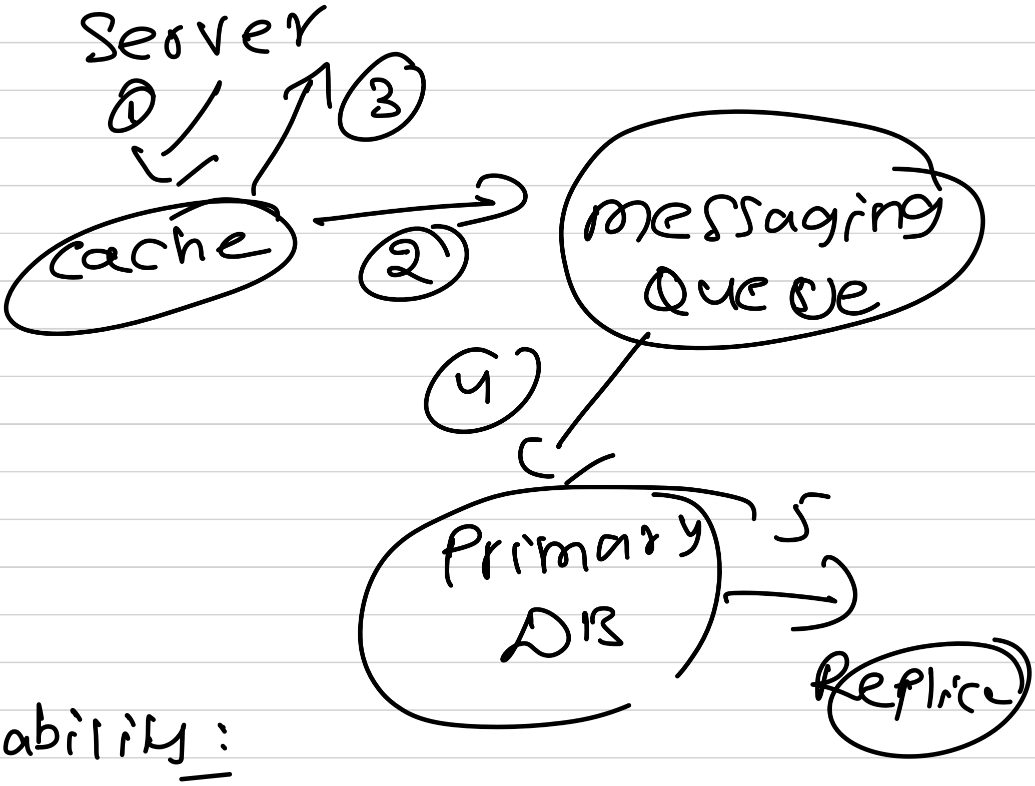
1) Strong consistency



2) Eventual consistency:



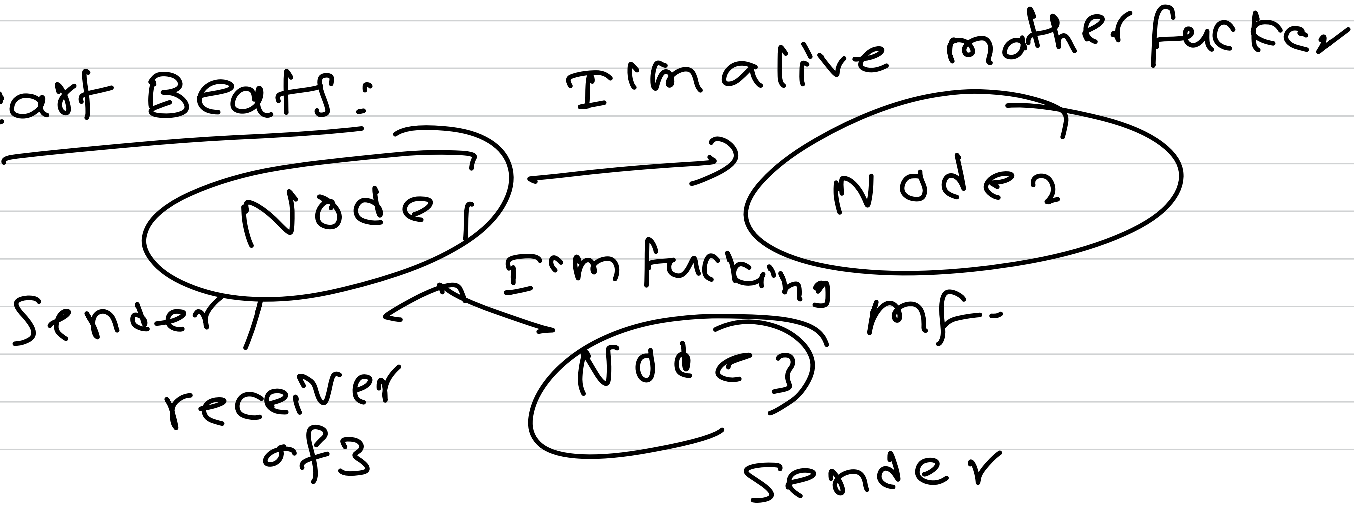
3) Weak consistency:



4) Linearizability:

5) causal consistency:

29) Heart Beats:



Sender / monitor

30) reliability: Data is consistent

31) Circuit Breaker: Break if more Load

32) Idempotency:

Matrix x matrix = matrix

Similarly

① get / xxxbabes = mia ①

② get / xxxbabes = mia ②

Same for 'n' no. of get calls

put

① put / xvideo = pang

② put / x vidoe = pang

Idempotent

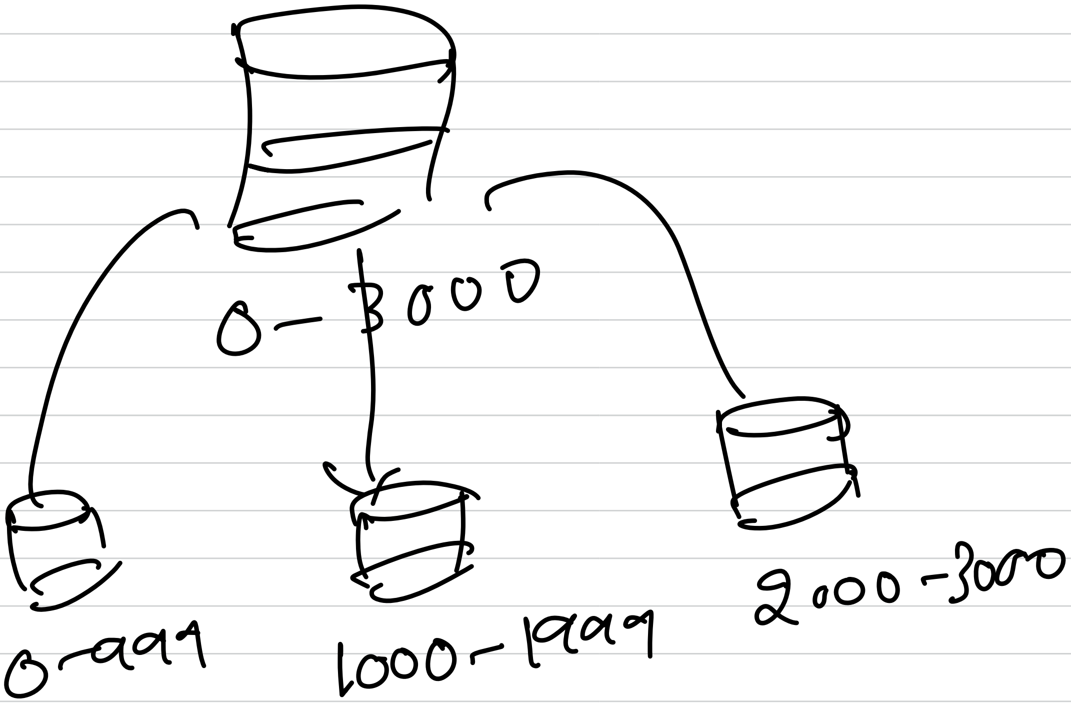
33) Database Scaling:

① Vertical Scaling (SQL)

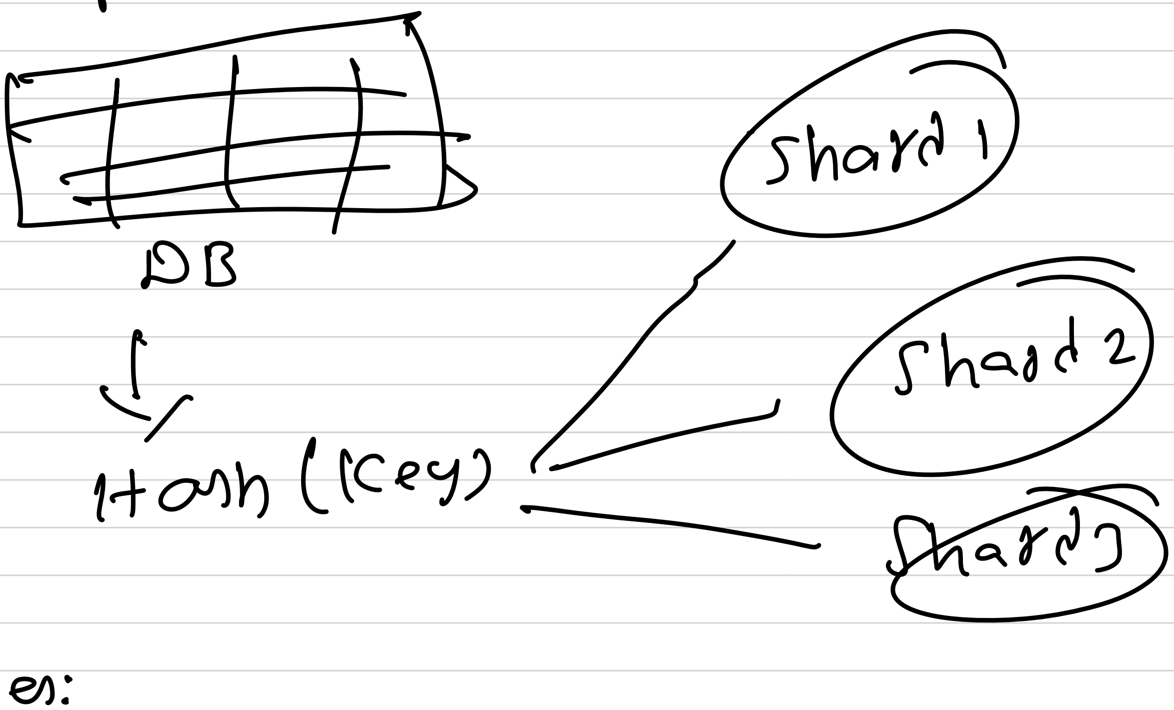
② Sharding

③ Indexing

(34) Database Sharding:



Split BigData into independent smaller pieces called shards



Types:

① Range Based Sharding

0-999 - shard 1
1000-2000 - shard 2 ...

⑨ Geo-Based Sharding:

India ∈ Shard 1 - Indian reels
America ∈ Shard 2 - American reels
Geo

(35) Data replication:

replicate data to ensure data is consistent accross all DBS (primary, slave)

Snapshot method

(36) Data redundancy:

Storing same data in multiple places

fault tolerance

(37) Failover: switch automatically to Backup

(38) Bloom fiters:

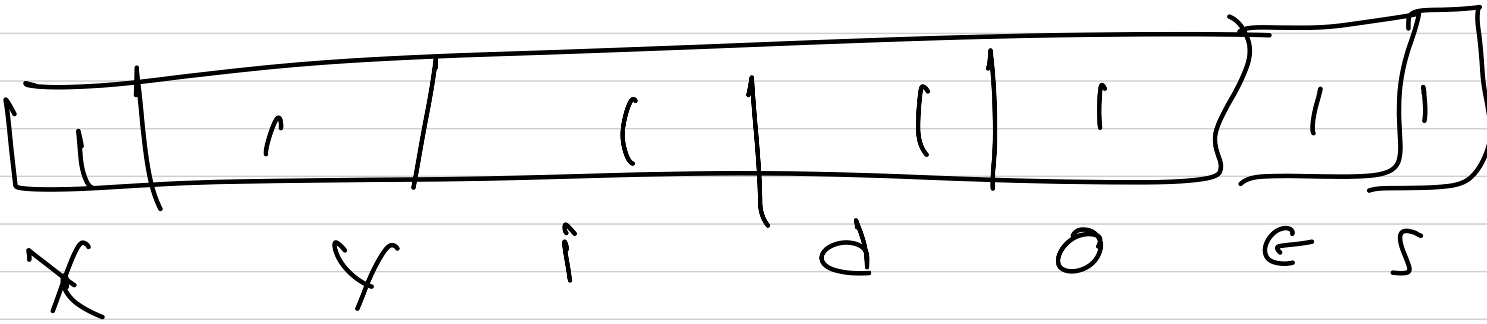
Searching data/video/audio gonna take O(N). Imagine in 1 Billion records.

Hash given data & store in

A-Z a-z 0-9

26 26 10 62 Base

62^7 = 1 Billion combinations



<hornycherry codes

↳ Hash → Xvidoes

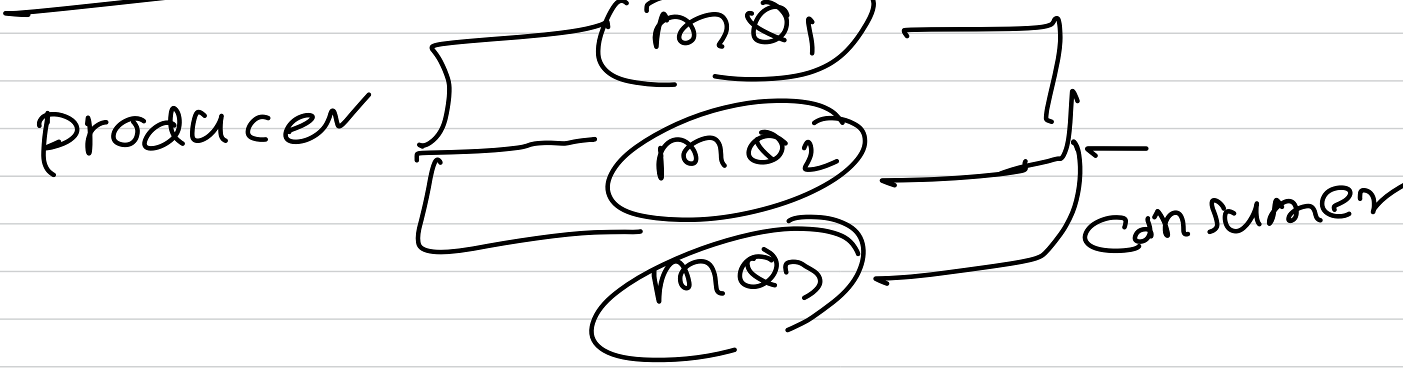
↳ convert to 62 Base then Hash & store

False positives ✓

False negatives ✗

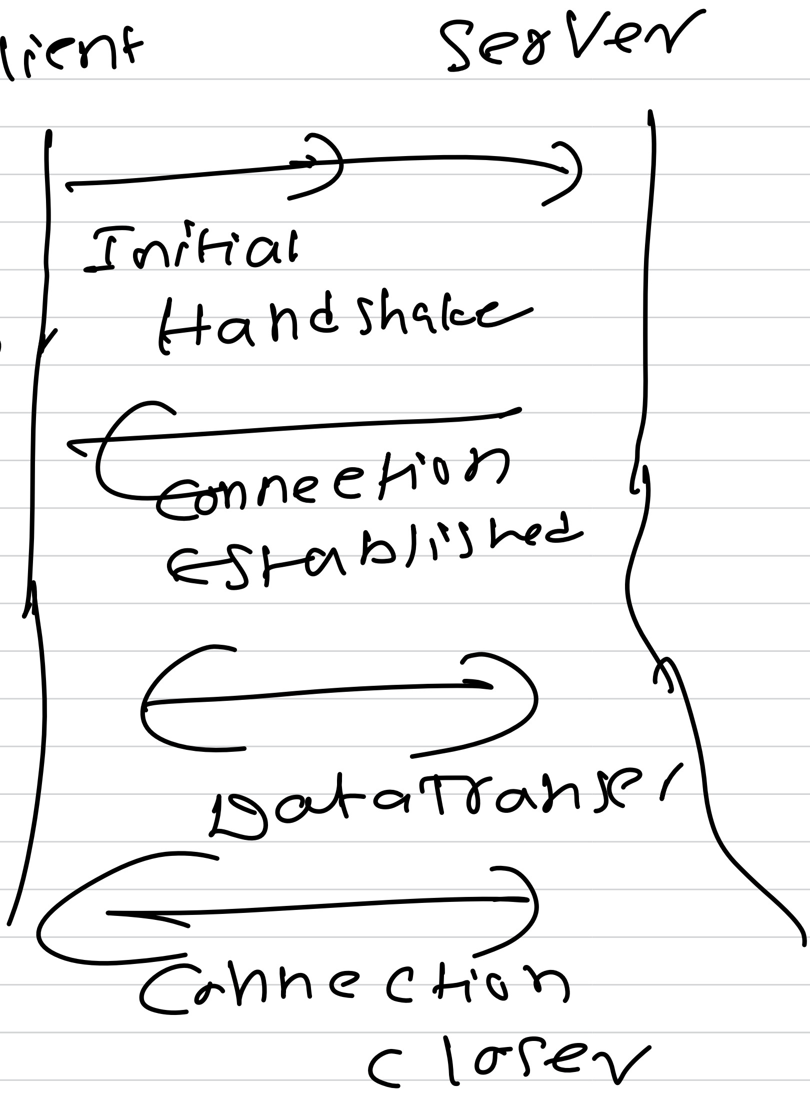
Hash1 && Hash2 && Hash 3 = True

(39) Message Queues:

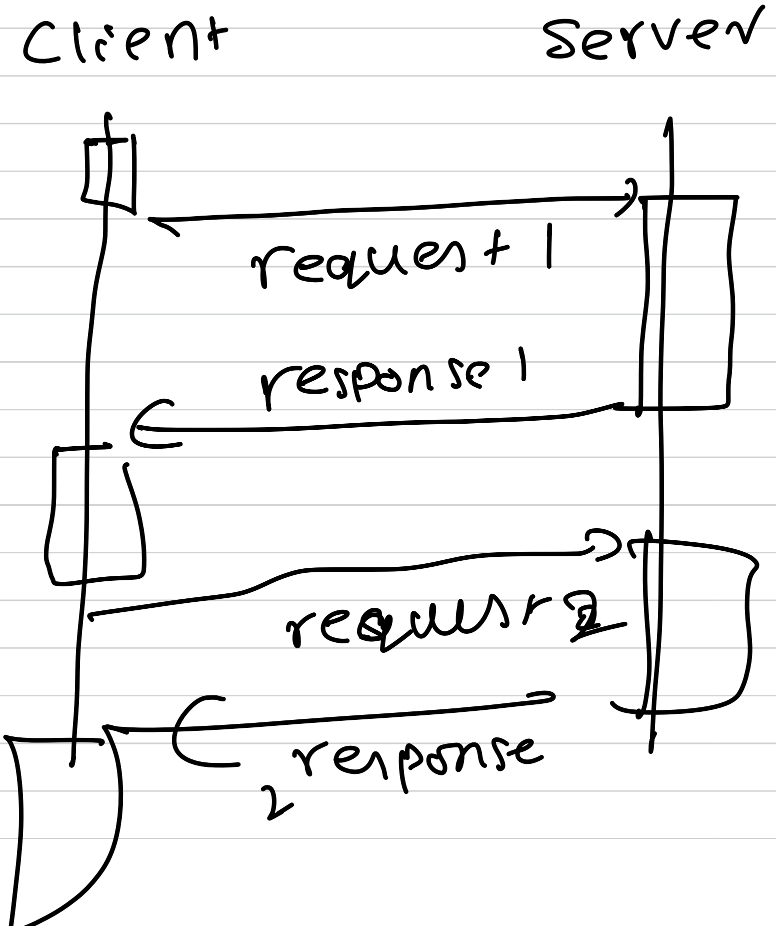


(40) Websockets: full-duplex
Bidirectional communicati
over single TCP connection

Whatsapp Base



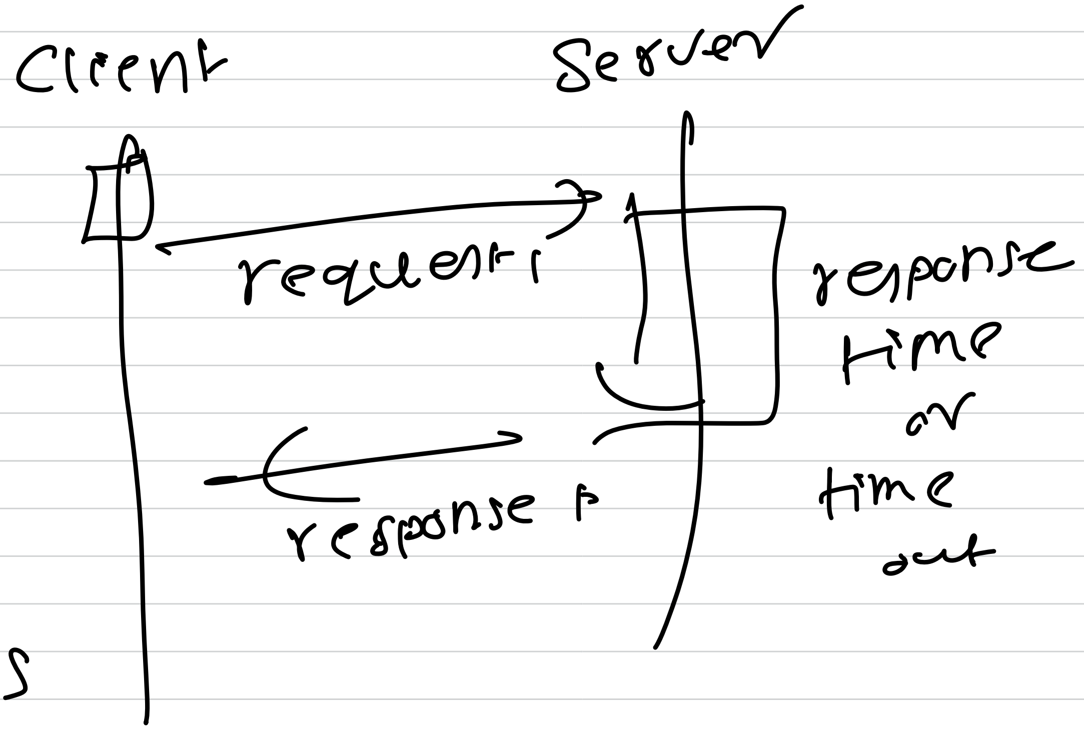
ful polling



client sends request to a server in fixed intervals of time

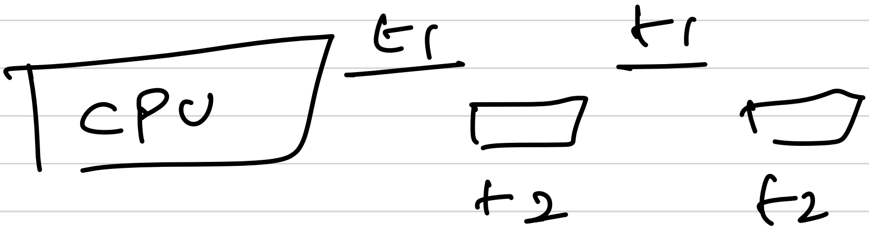
(42) Long polling

Connection remains until response is sent or timeover happens

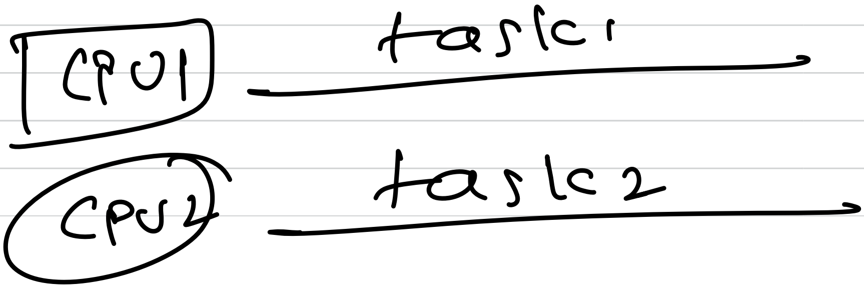


(43) concurrency vs parallelism

more than 1 task at a time by single cpu - concurrency

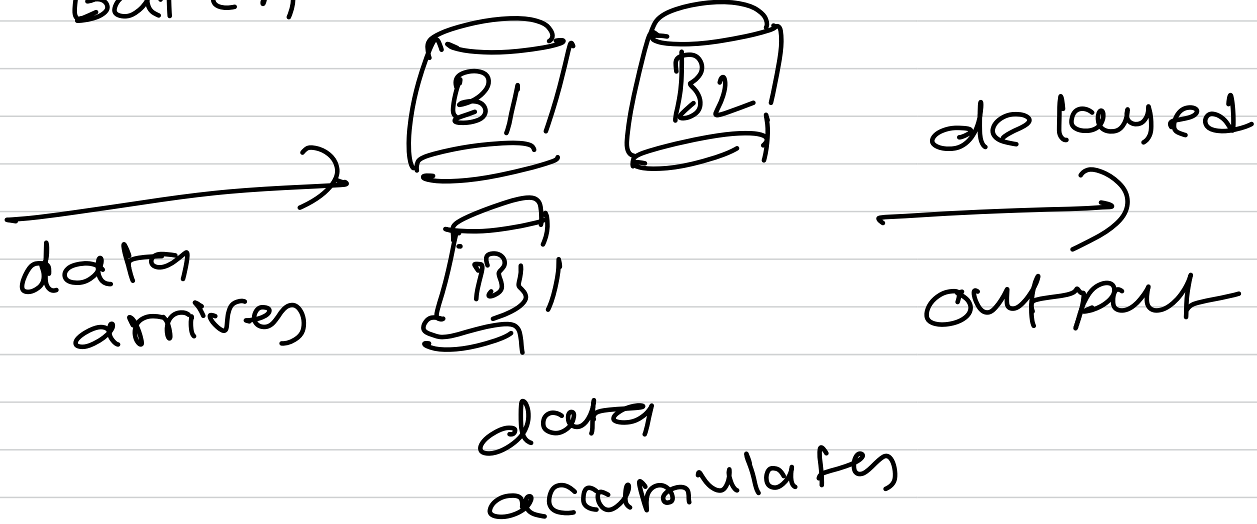


Multiple CPUs run multiple tasks at a time parallelism

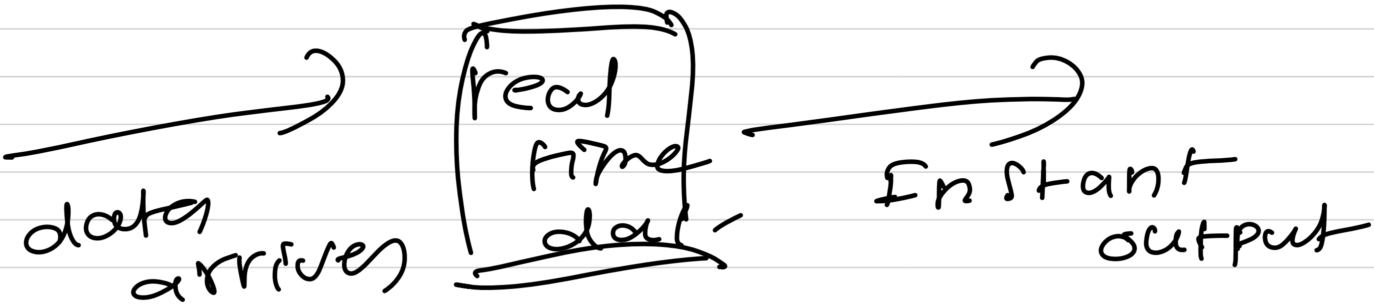


(44) Batch vs Stream Processing:

Batch



Stream



(45) Push vs pull architecture:

push → Any update

server notifies all clients

pull → clients pull needed information from the server

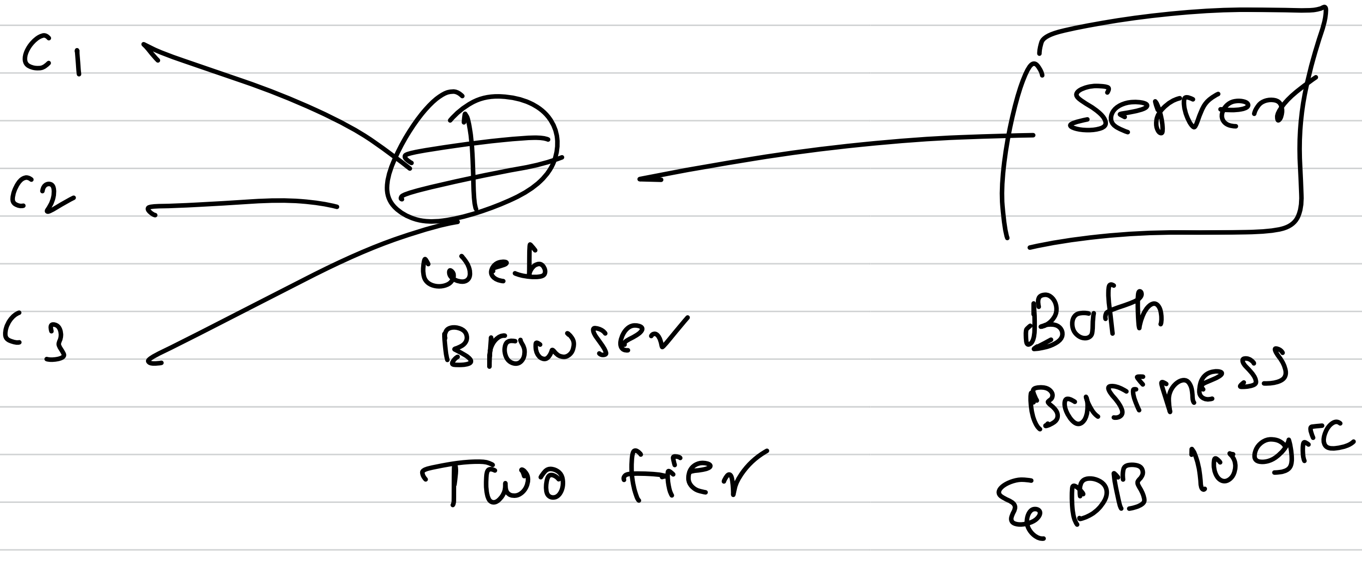
(46) **RPC:** Remote Procedure call

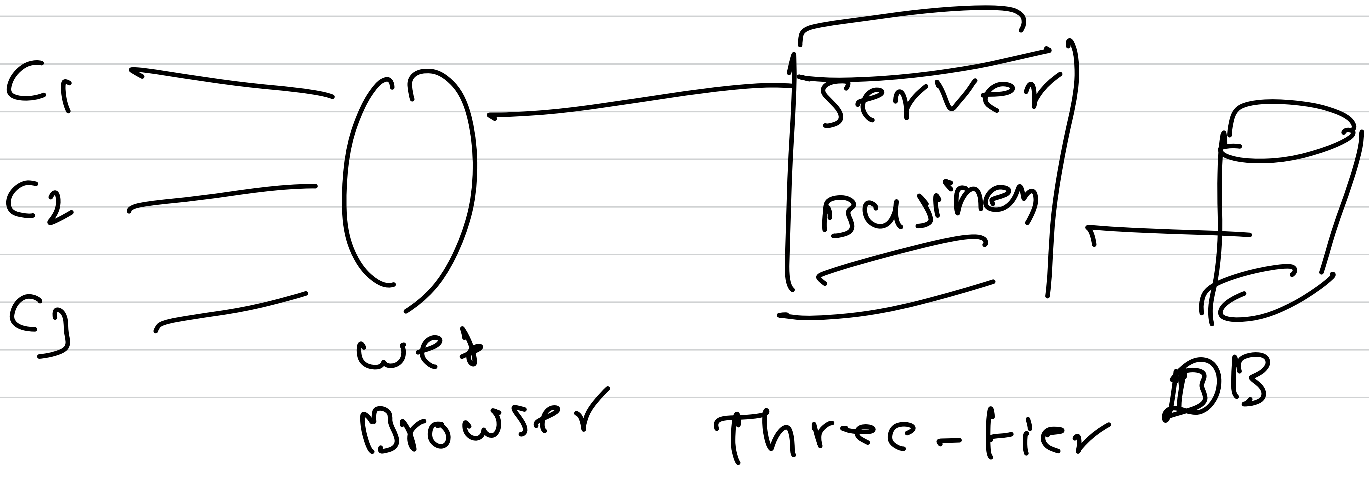
Program executes sub-routines in another address space or another physical machine

(47) **Throughput:**

$$\frac{\text{No.of operations performed}}{\text{Total time}}$$

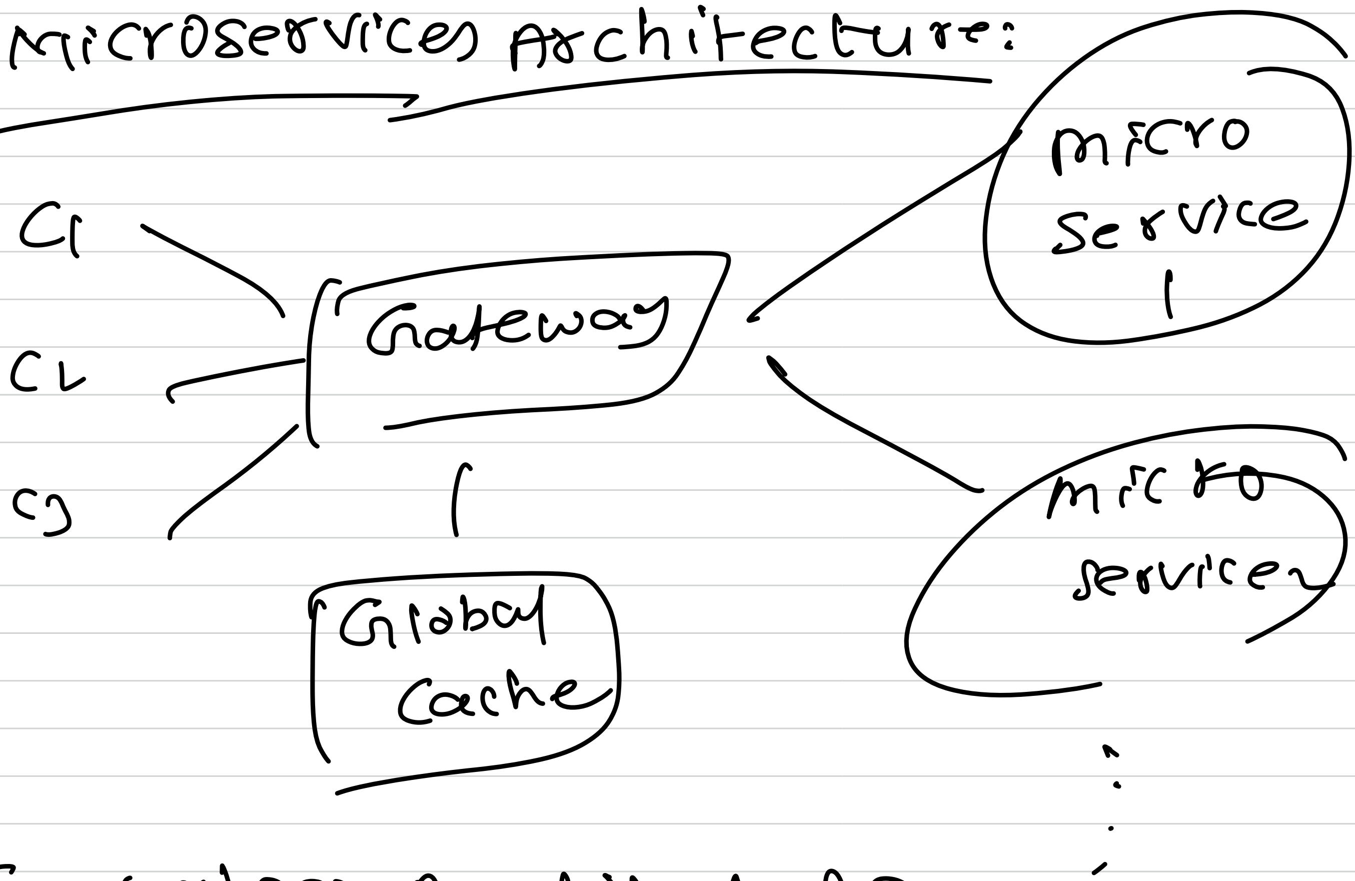
(48) **Client - server Architecture:**





N-tier

(49) Microservices Architecture:



(50) Serverless Architecture:

(51) Event Driven Architecture

(52) P2P Architecture: Block chain