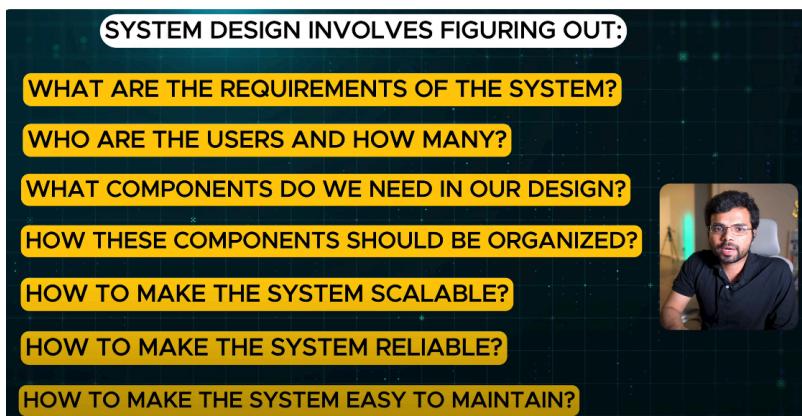
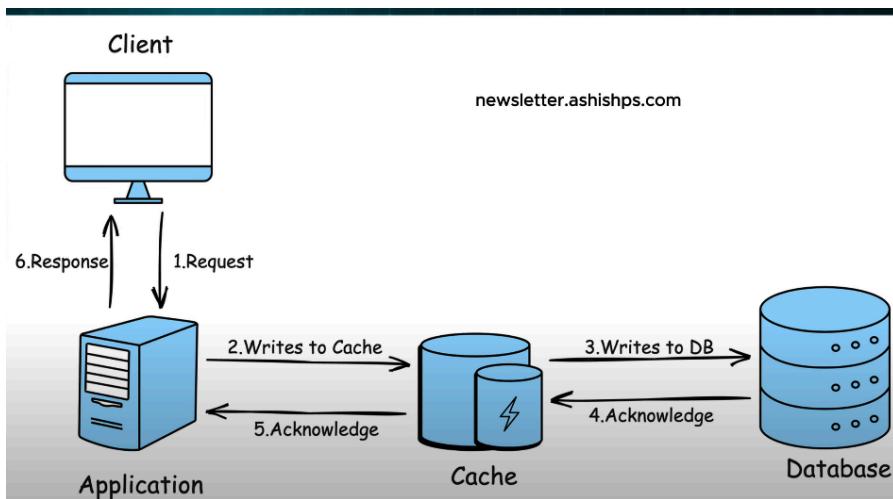


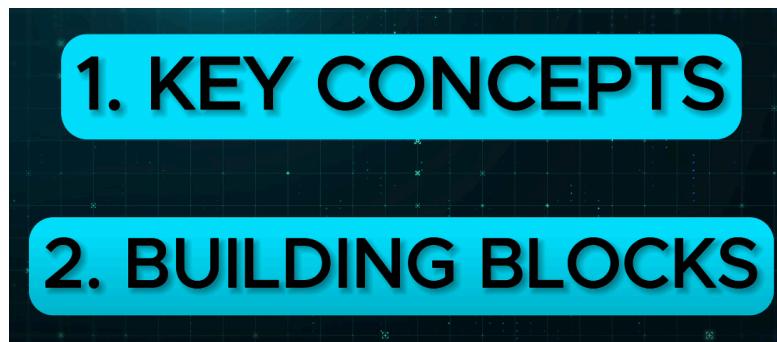
## System Design



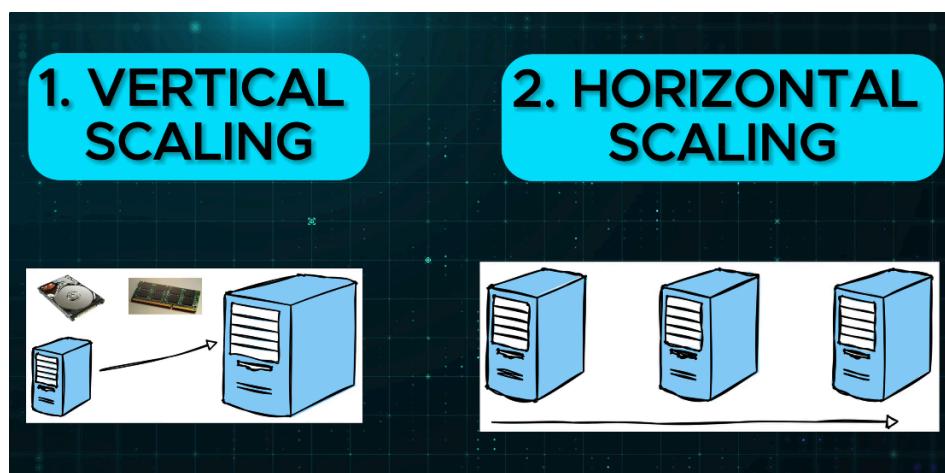
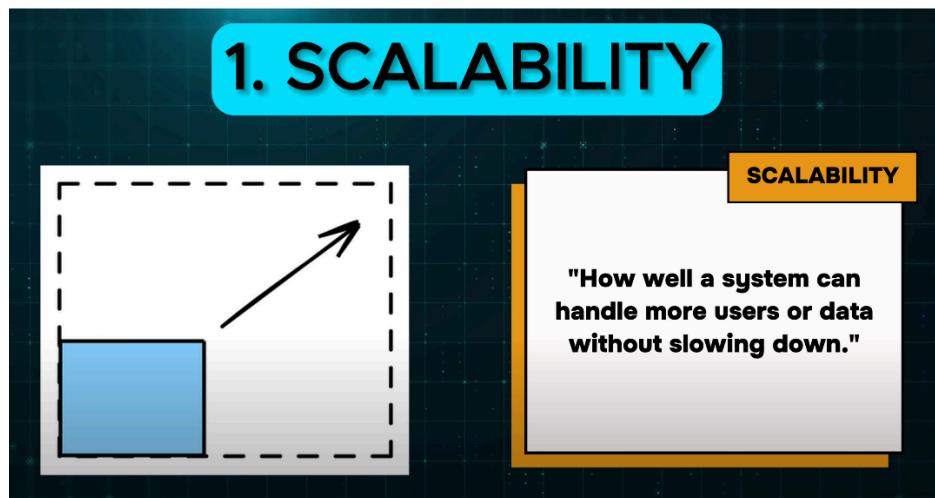
Not about writing the code. It is Big picture. It based on concept called Distributed system  
Distributed system means a group of computers work together to achieve common goal.



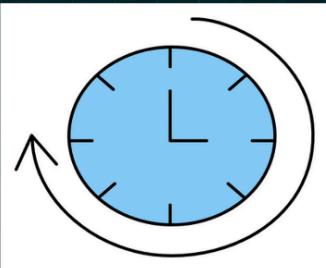
Fundamental concept of System Design:



Key concepts include 12 sub concept



## 2. PERFORMANCE



PERFORMANCE

"How fast your system works?"

### 1. LATENCY

LATENCY

"Time it takes for a single task."

### 2. THROUGHPUT

THROUGHPUT

"How many tasks your system can handle in a certain time."

## 3. AVAILABILITY

AVAILABILITY

"Your system is up and running when users need it without any significant downtime."

## 4. RELIABILITY

"Your system is doing what it's supposed to do even if things go wrong."

REPLICATION

REDUNDANCY

FAILOVER  
MECHANISMS

## 5. CONSISTENCY

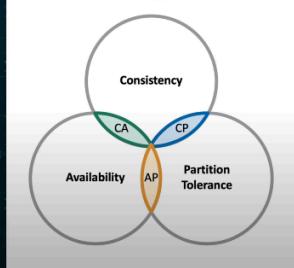
CONSISTENCY

"All users see the same data at the same time no matter which part of the system they interact with."

## EVENTUAL CONSISTENCY

**"The data may not be upto date immediately but it will be after a specific time."**

## 6. CAP THEOREM



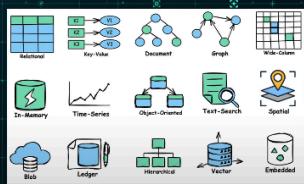
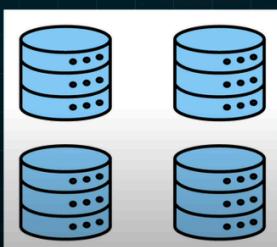
### CAP THEOREM

**"In a Distributed System you can only have 2/3:**

- 1. Consistency**
- 2. Availability**
- 3. Partition Tolerance"**

We need to trade-off which means selecting best out of available choices

## 7. DATA STORAGE AND RETRIEVAL



PARITIONING SHARDING REPLICATION

## 8. ACID TRANSACTIONS

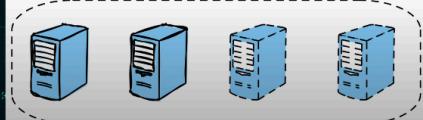
ACID

A - ATOMICITY  
C - CONSISTENCY  
I - ISOLATION  
D - DURABILITY

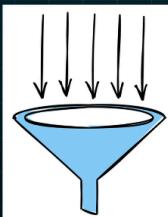
## 9. CONSISTENCE HASHING

"Used to spread data across a group of servers."

LOAD BALANCING SCALABILITY



## 10. RATE LIMITING



PREVENT ABUSE  
PROTECT AGAINST DDOS ATTACKS  
ENSURES FAIR USAGE OF RESOURCES

"Control the rate at which clients can make requests to a system."

## 11. NETWORKING AND COMMUNICATION

"How different parts of a system communicate?"

NETWORK PROTOCOLS

APIs

MESSAGE QUEUES

EVENT-DRIVEN ARCHITECTURE

## 12. SECURITY AND PRIVACY

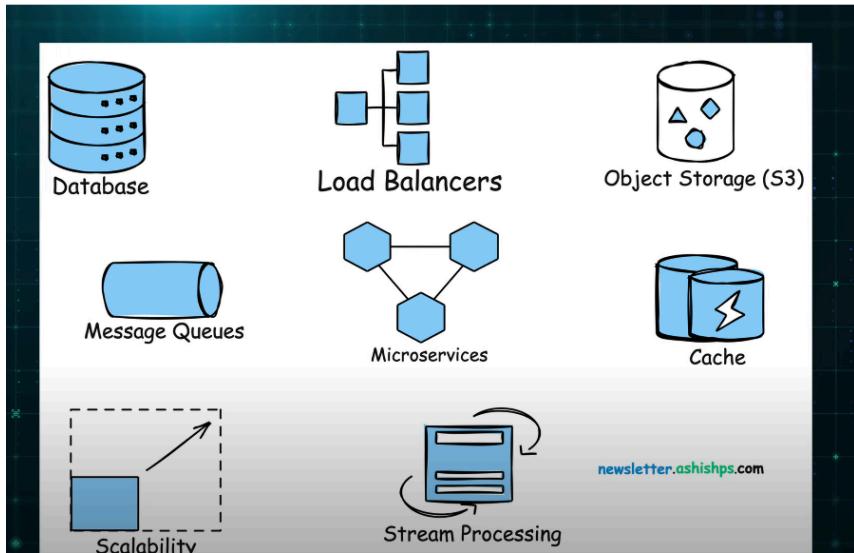
"Putting in place methods to keep important data safe and stop unwanted access."

AUTHENTICATION

AUTHORIZATION

ENCRYPTION

Building blocks of system Design



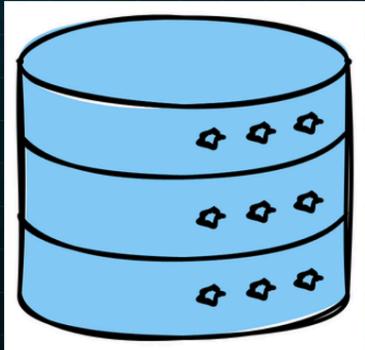
## 1. APPLICATION SERVERS

"Computers that handle the business logic and processing required by the application."

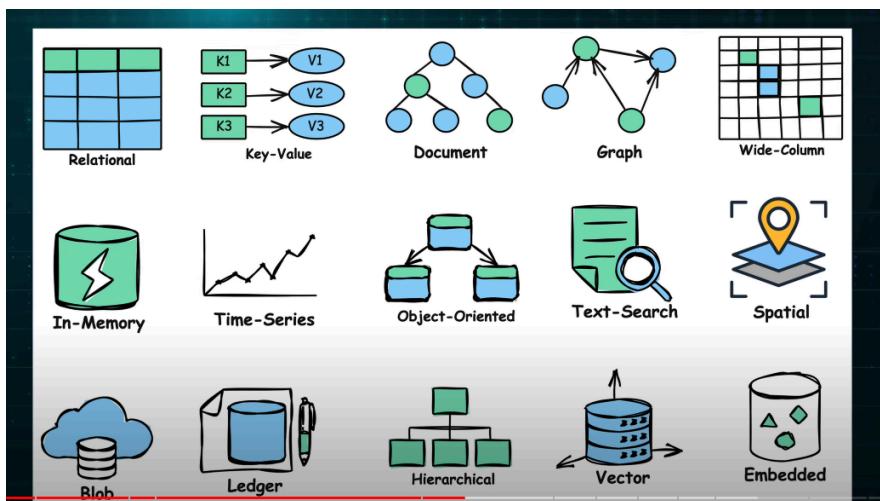
## 2. LOAD BALANCERS

"Distribute incoming requests to different servers to ensure no single server gets overwhelmed."

# 3. DATABASES



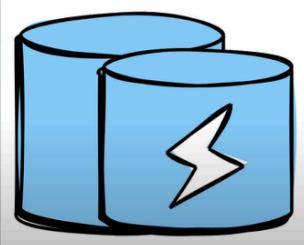
Different types of Databases



Most commonly used DBs

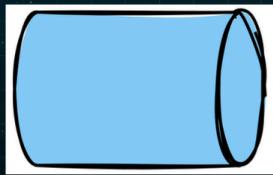


## 4. CACHING



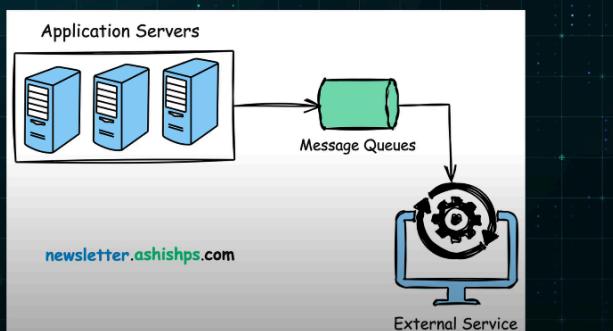
"Store frequently accessed data in a fast access storage to reduce the load on primary data source and improve response times."

## 5. MESSAGE QUEUES



"Enable asynchronous communication between system components"

## 5. MESSAGE QUEUES



## 6. STORAGE

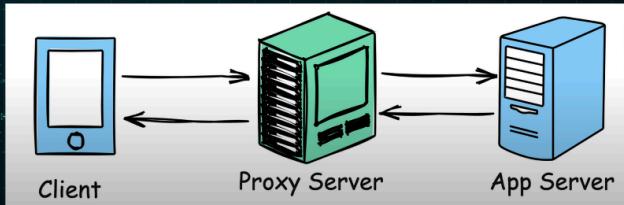


LOCAL FILE SYSTEMS  
DISTRIBUTED FILE SYSTEMS  
OBJECT STORAGE SYSTEMS

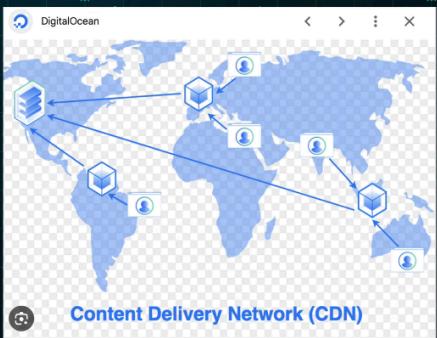
"Store and retrieve data such as files, images or videos."

## 7. PROXY SERVER

LOAD BALANCING CACHING SECURITY CONTENT FILTERING



## 8. CDN



## QUESTION?

# WHERE TO LEARN THESE CONCEPTS IN MORE DETAIL?

<https://github.com/ashishps1/awesome-system-design-resources>

Reading these links will give more insight.

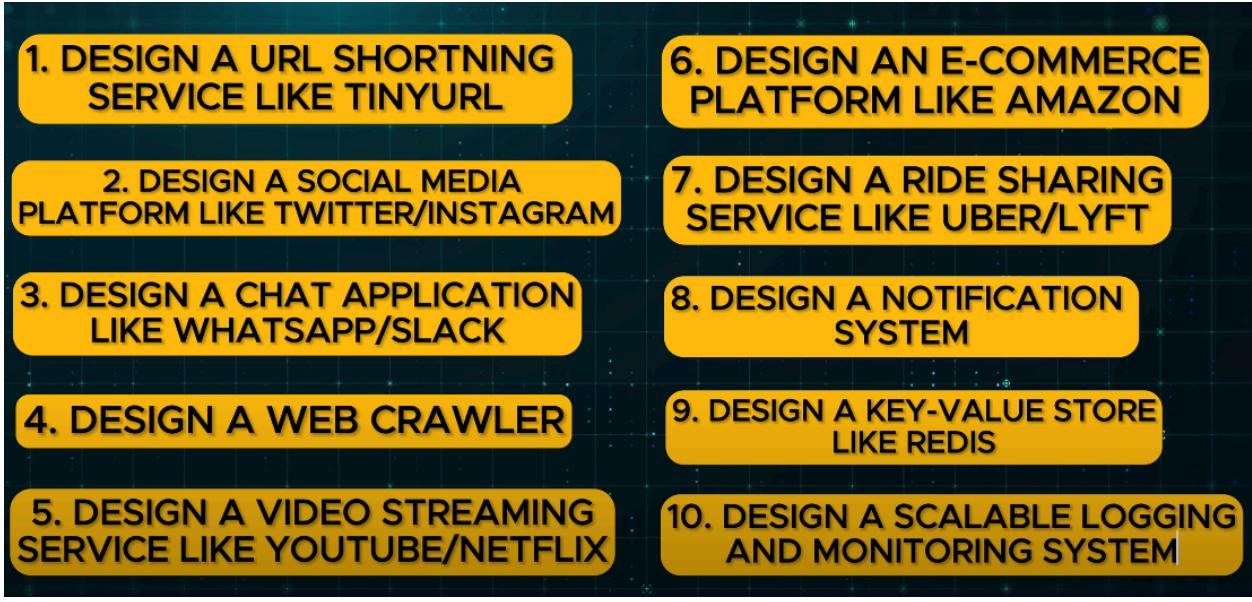
Study multiple times to get grip fastly and to understand better

- [CAP Theorem](#)
- [ACID Transactions](#)
- [Consistent Hashing](#)
- [Rate Limiting](#)
- [API Design](#)
- [Strong vs Eventual Consistency](#)
- [Synchronous vs. asynchronous communications](#)
- [REST vs RPC](#)
- [Batch Processing vs Stream Processing](#)
- [Fault Tolerance](#)
- [Consensus Algorithms](#)
- [Gossip Protocol](#)
- [Service Discovery](#)
- [Disaster Recovery](#)
- [Distributed Tracing](#)
- [Top 15 Tradeoffs](#)

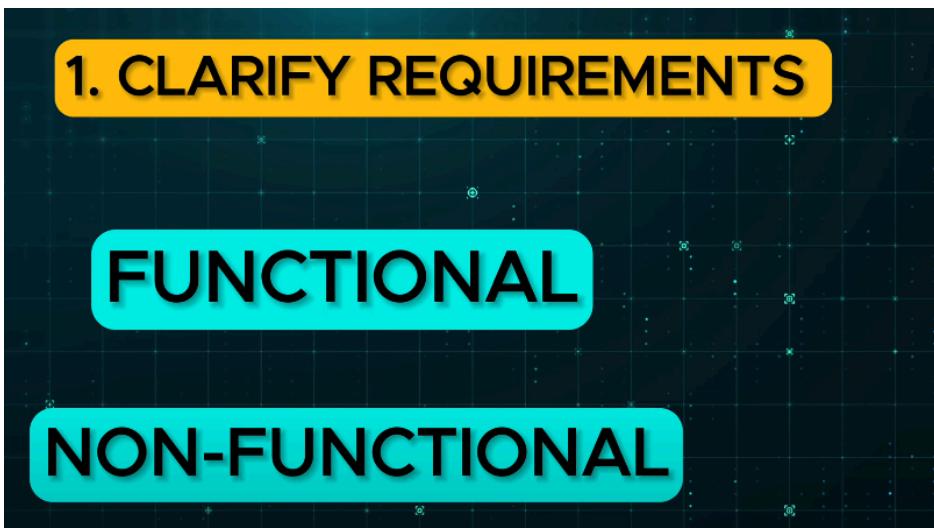
### ❖ System Design Building Blocks

- [Horizontal vs Vertical Scaling](#)
- [Databases](#)
- [Content Delivery Network \(CDN\)](#)
- [Domain Name System \(DNS\)](#)
- [Caching](#)
- [Distributed Caching](#)
- [Load Balancing](#)
- [SQL vs NoSQL](#)
- [Database Index](#)
- [Consistency Patterns](#)
- [HeartBeat](#)

Most common systems Designs

- 
1. DESIGN A URL SHORTNING SERVICE LIKE TINYURL
  2. DESIGN A SOCIAL MEDIA PLATFORM LIKE TWITTER/INSTAGRAM
  3. DESIGN A CHAT APPLICATION LIKE WHATSAPP/SLACK
  4. DESIGN A WEB CRAWLER
  5. DESIGN A VIDEO STREAMING SERVICE LIKE YOUTUBE/NETFLIX
  6. DESIGN AN E-COMMERCE PLATFORM LIKE AMAZON
  7. DESIGN A RIDE SHARING SERVICE LIKE UBER/LYFT
  8. DESIGN A NOTIFICATION SYSTEM
  9. DESIGN A KEY-VALUE STORE LIKE REDIS
  10. DESIGN A SCALABLE LOGGING AND MONITORING SYSTEM

The expectation from system design is



## 2. ESTIMATE THE CAPACITY

### Step 2: Capacity Estimation

After clarifying the requirements, it is important to estimate the capacity of the system you are going to design.

Estimating the scale upfront helps guide your design decisions and ensures that the system can meet the desired criteria.

This can include things like expected daily/monthly users, read/write requests per second, data storage and network bandwidth needs.



Users



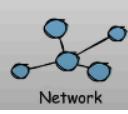
Traffic



Storage



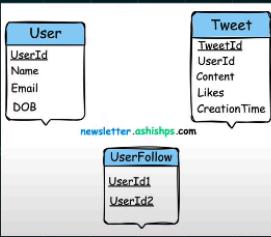
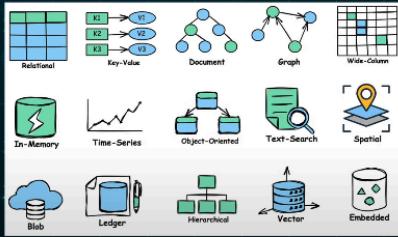
Memory



Network

newsletter.ashishps.com

## 3. CHOOSE THE RIGHT DATABASE



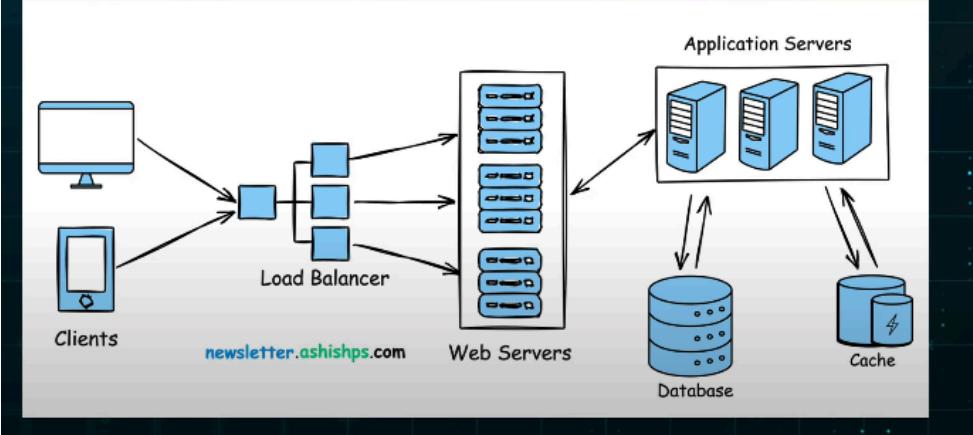
## 4. DESIGN APIs AND REQUEST/RESPONSE PATTERN

Define the API Endpoints:

```
createProfile(name, email, password)  
login(email, password)  
postTweet(userId, content, timestamp)  
followUser(userId1, userId2)
```

Example: Twitter APIs

## 5. SKETCH OUT A HIGH-LEVEL BLOCK DIAGRAM



## 6. DEEP DIVE INTO KEY COMPONENTS

### Common Areas for Deep Dives:

- **Databases:** How would you handle a massive increase in data volume? Discuss sharding (splitting data across multiple databases), replication (read/write replicas).
- **Web Servers/Application Servers:** How do you add more servers behind the load balancer for increased traffic?
- **Load Balancers:** Which Load Balancing techniques and algorithms to use (e.g., round-robin, least connections).
- **Caching:** Where do you add more cache layers (in front of web servers? in the application layer?), and how do you deal with cache invalidation?
- **Single Points of Failure:** Identify components whose failure would take down the system and discuss how to address it.
- **Authentication/Authorization:** How do you manage user access and permissions securely?
- **Rate Limiting:** How do you prevent excessive use or abuse of your APIs?

## 7. DISCUSS HOW THE SYSTEM WILL SCALE UNDER LOAD

**SHARDING**

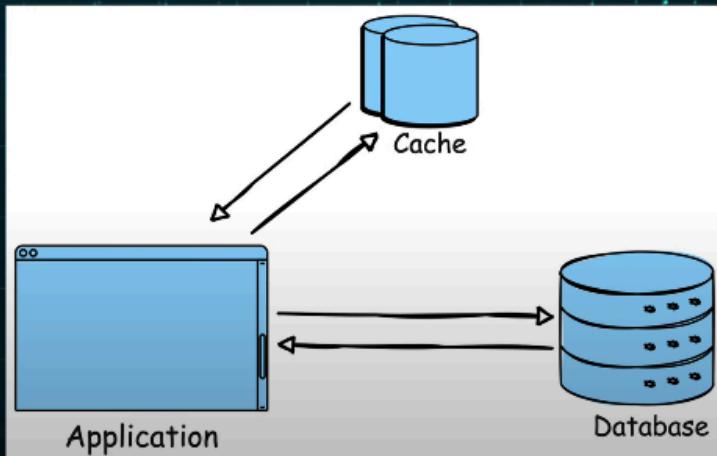
**REPLICATION**

**PARTITIONING**

## 8. DISCUSS TRADEOFFS

### SQL vs NoSQL

## 9. DISCUSS CACHING



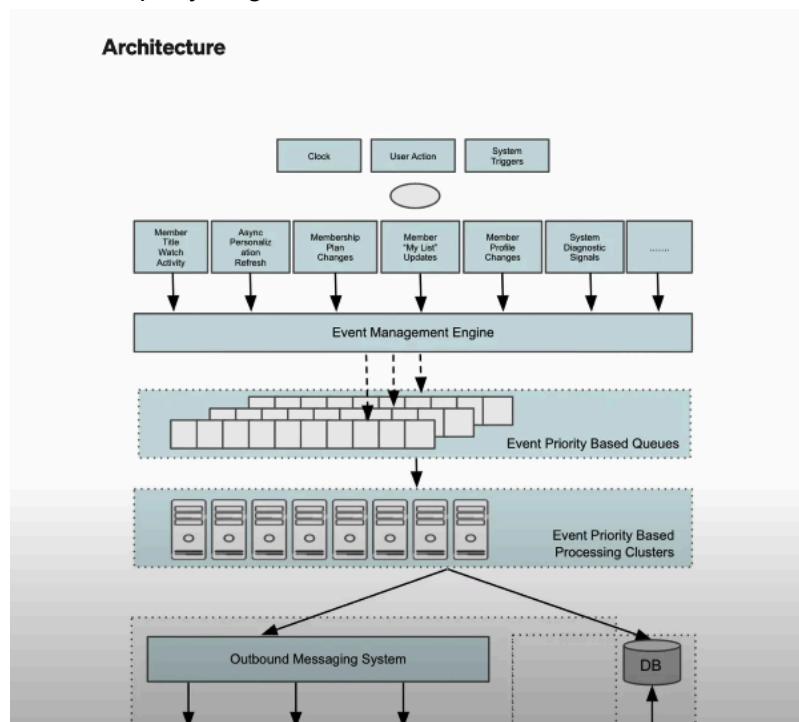
## 10. DISCUSS STRATEGIES FOR HANDLING FAILURES

REPLICAS

FALLBACKS

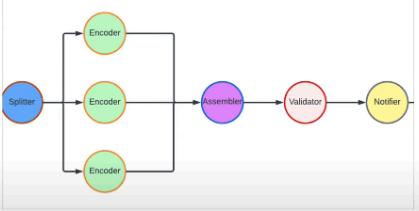
RETRIES

Read company Engineer blobs



**N THE NETFLIX TECH BLOG**

Following ▾



The Making of VES: the Cosmos Microservice for Netflix Video Encoding  
Liwei Guo, Vinicius Carvalho, Anush Moorthy, Aditya Mavankar, Lishan Zhu  
Netflix Technology Blog · Apr 9 · 11 min read

Latest

**Uber Blog** Explore ▾

Engineering

Engineering

The technology behind Uber Engineering



Migrating a Trillion Entries of Uber's Ledger Data from DynamoDB to LedgerStore  
April 11 / Global



How LedgerStore Supports Trillions of Indexes at Uber  
April 4 / Global



Scaling AI/ML Infrastructure at Uber  
March 29 / Global

Interested in joining Uber Eng?

**The Airbnb Tech Blog**  
A deeper look into how our Engineers and Data Scientists build a world of belonging

DATA AI & ML MOBILE WEB INFRASTRUCTURE OPEN SOURCE PEOPLE CAREERS

Following ▾



Introducing Trio | Part III  
Part three on how we built a Compose based architecture with Mavericks in the Airbnb Android app  
Eli Hart · Apr 11 · 10 min read