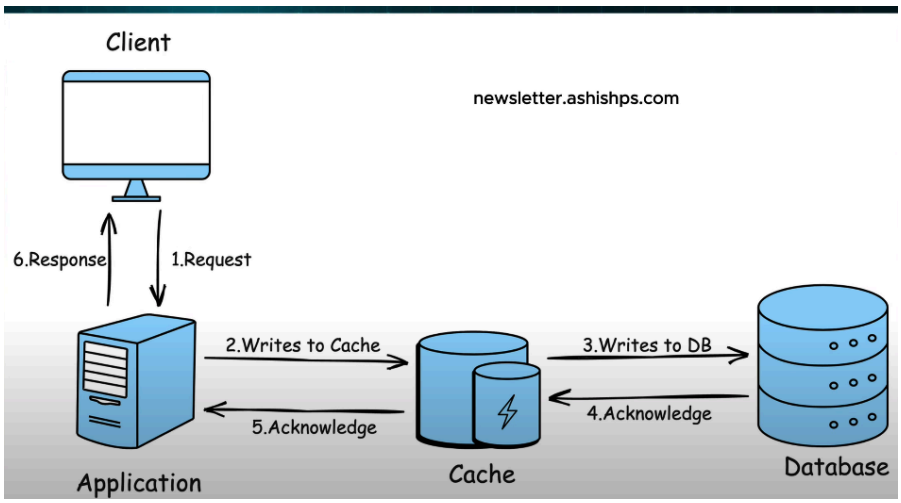
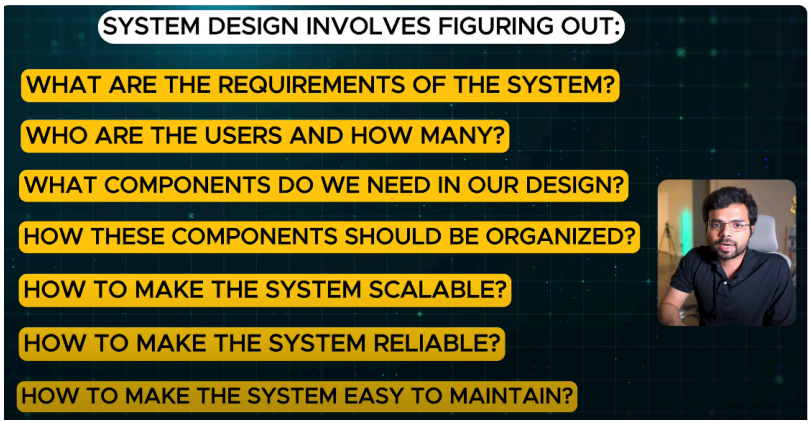
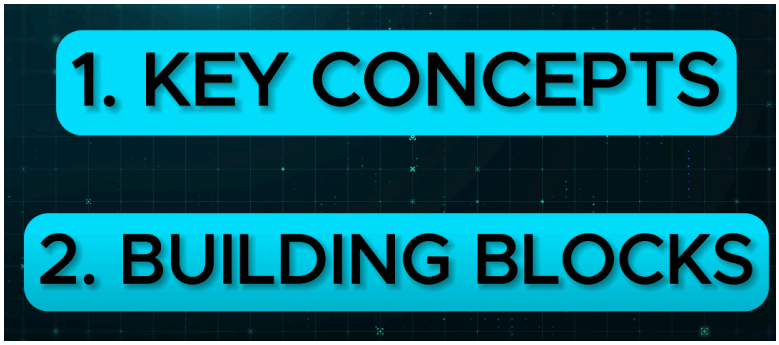
System Design

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.

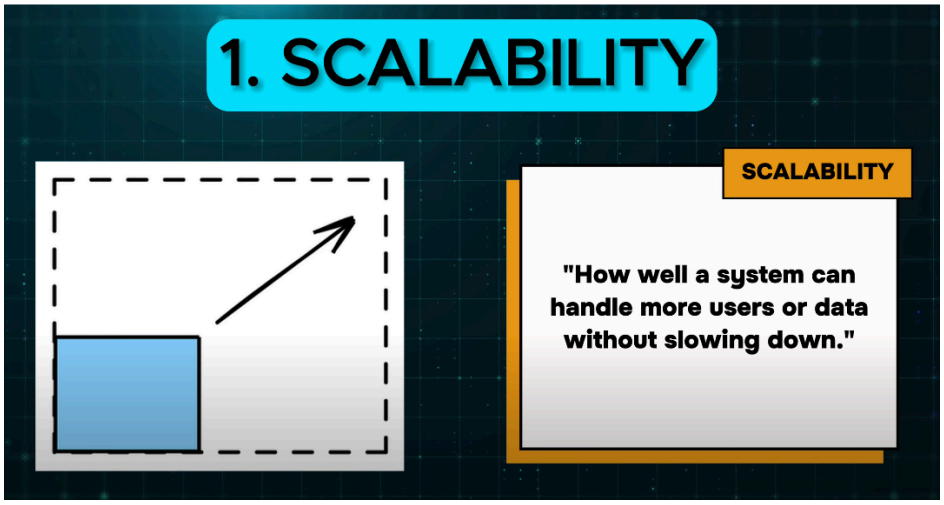


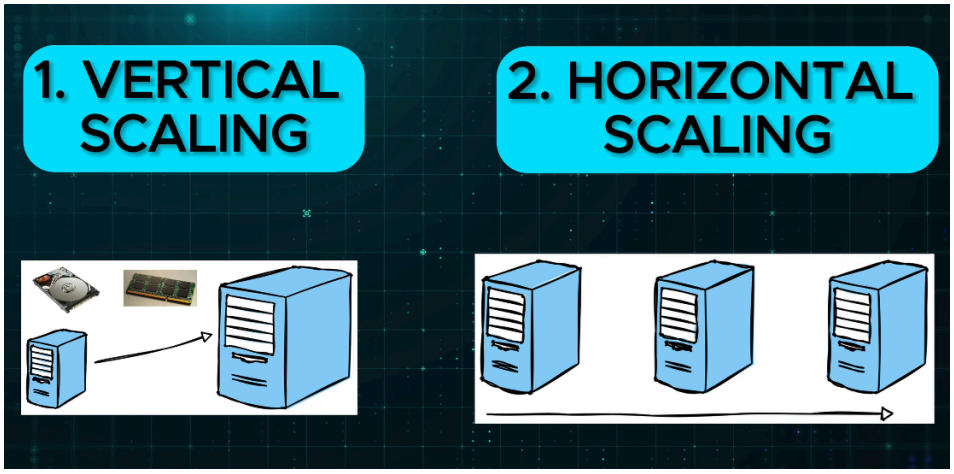


Fundamental concept of System Design:

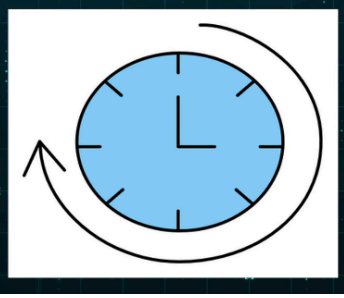


Key concepts include 12 sub concept





2. PERFORMANCE



PERFORMANCE

"How fast your system works?"

1. LATENCY

LATENCY

"Time it takes for a single task."

2. THROUGHPUT

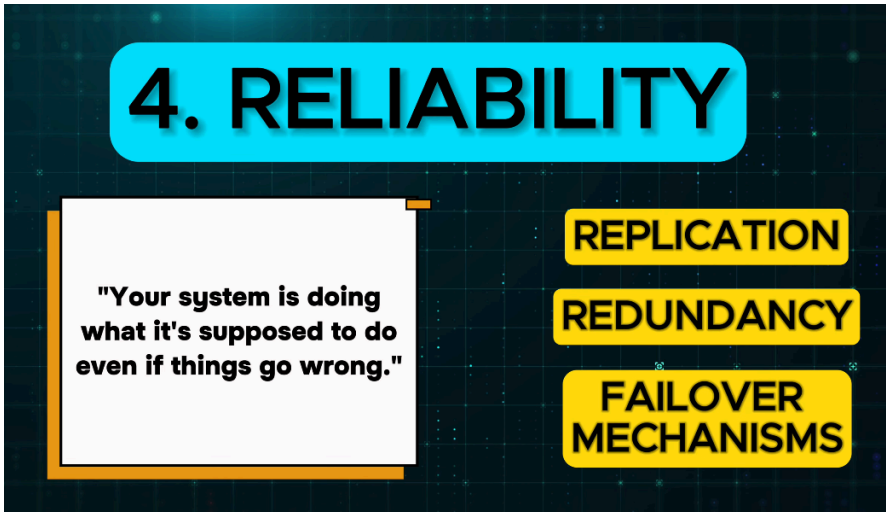
THROUGHPUT

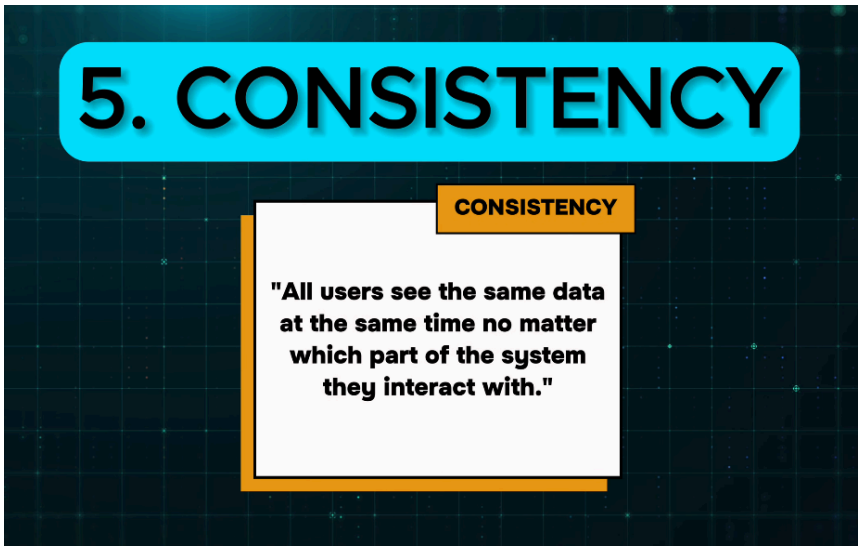
"How many tasks your system can handle in a certain time."

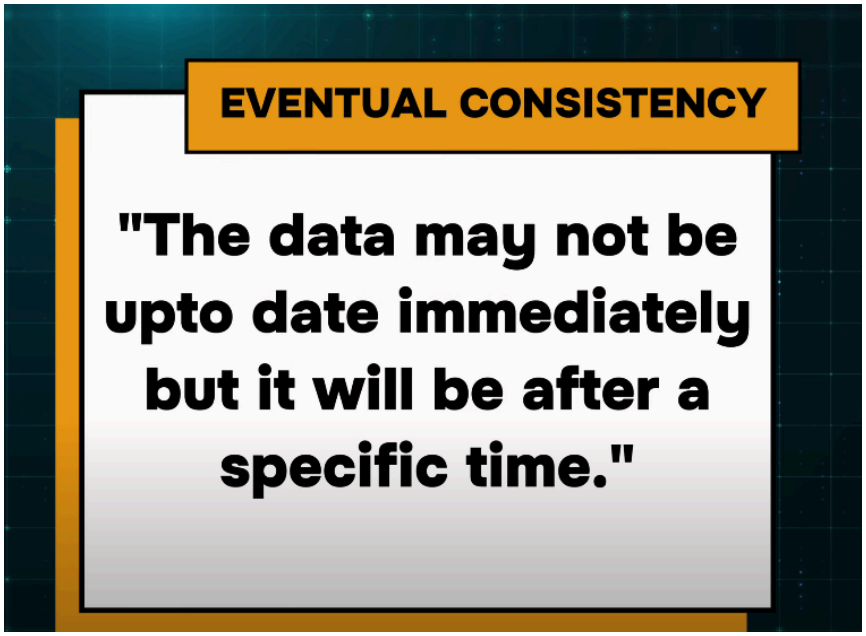
3. AVAILABILITY

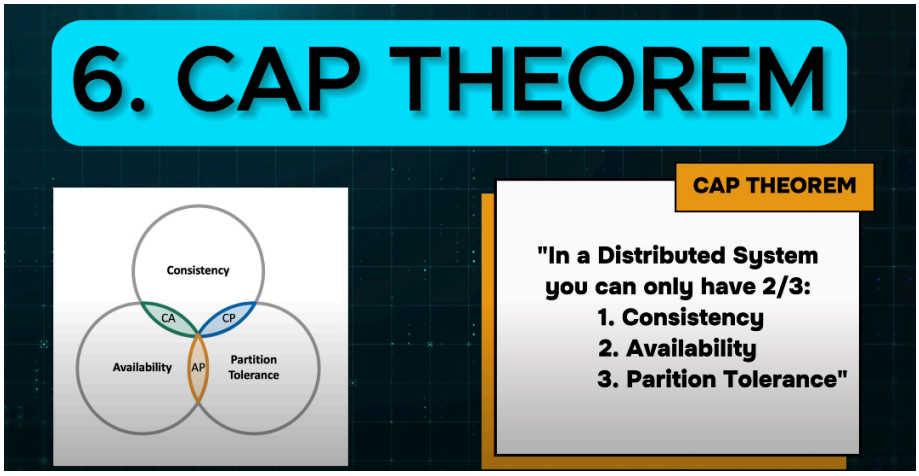
AVAILABILITY

"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

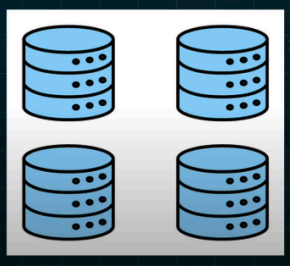

5. CONSISTENCY
CONSISTENCY
"All users see the same data at the same time no matter which part of the system they interact with."

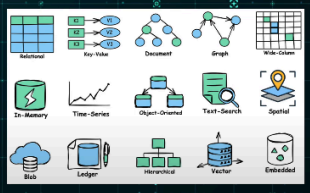


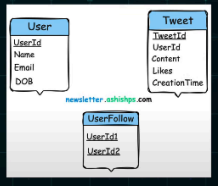


We need to trade-off which means selecting best out of available choices

7. DATA STORAGE AND RETRIEVAL







PARITIONING SHARDING REPLICATION

8. ACID TRANSACTIONS

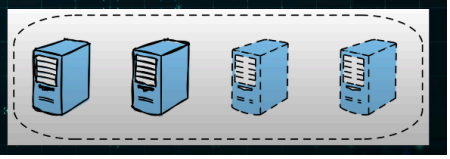
ACID

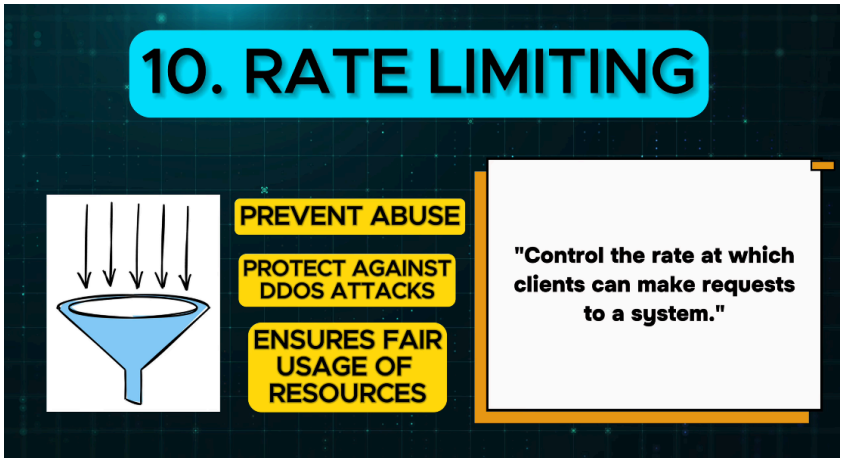
A - ATOMICITY
C - CONSISTENCY
I - ISOLATOIN
D - DURABILITY

9. CONSISTENCE HASHING

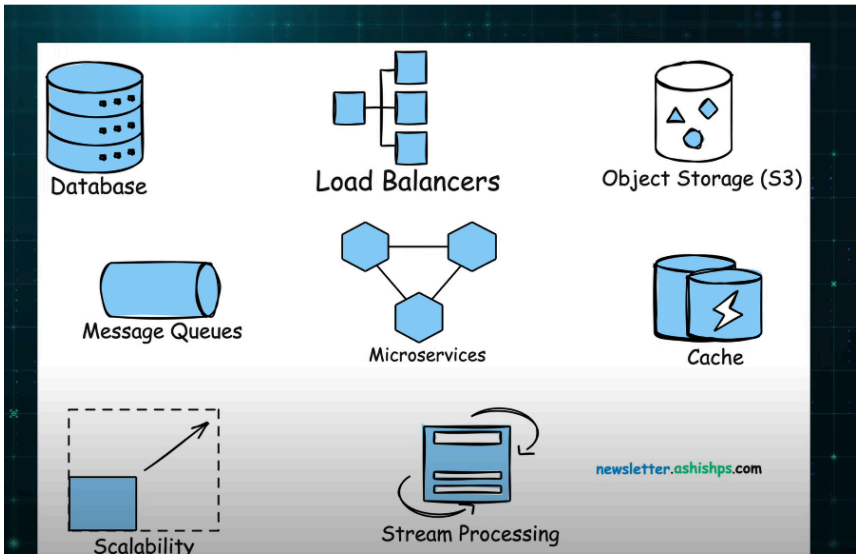
"Used to spread data across a group of servers."

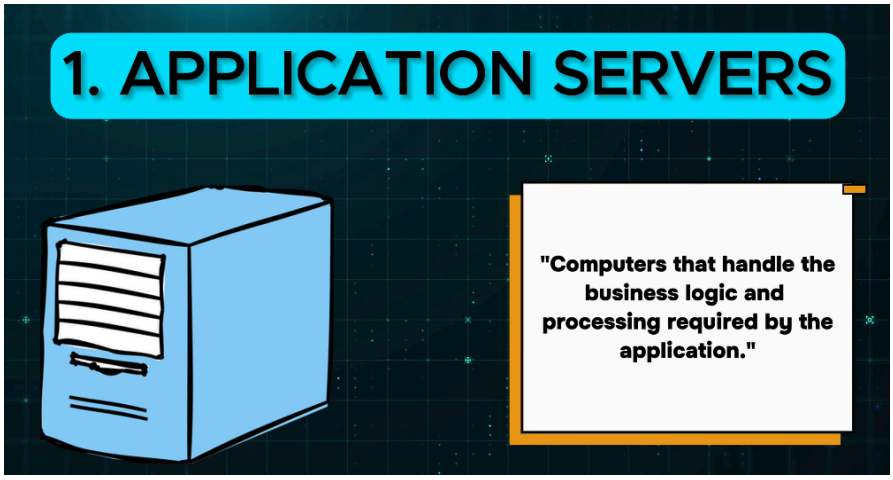
LOAD BALANCING SCALABILITY

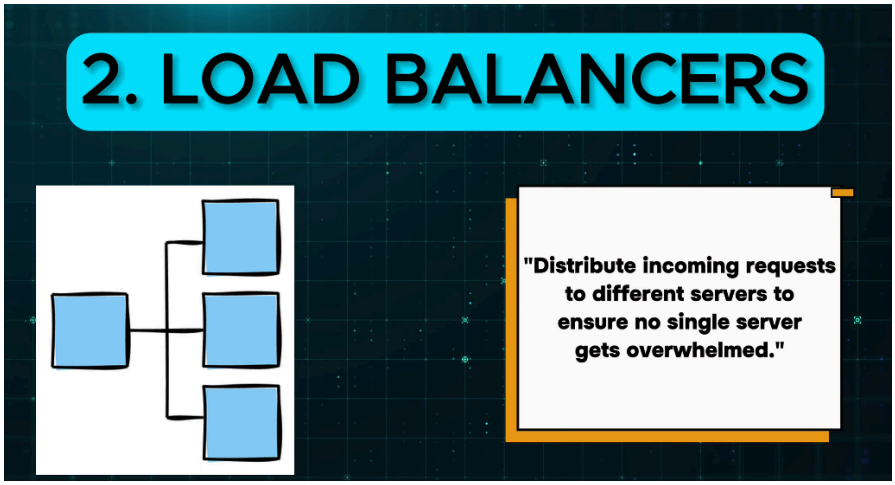




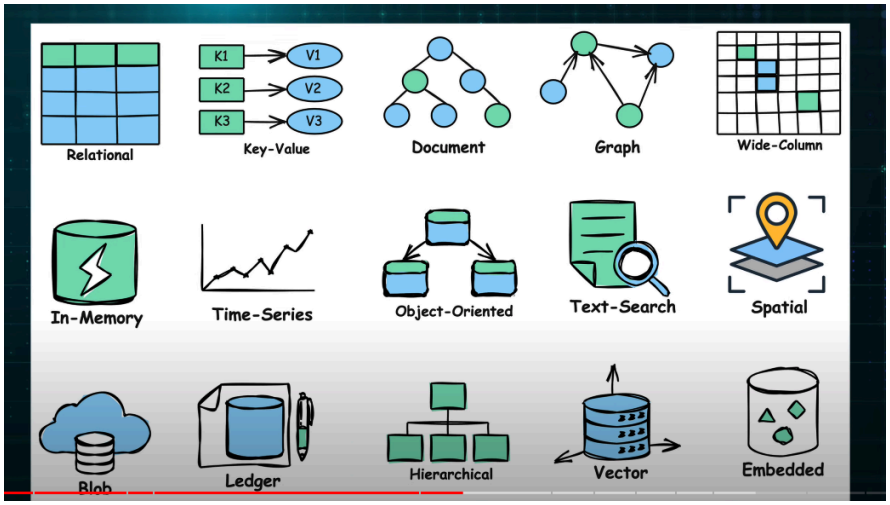
Building blocks of system Design


Database
Load Balancers
Object Storage (S3)
Message Queues
Microservices
Cache
Scalability
Stream Processing
newsletter.ashishps.com


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

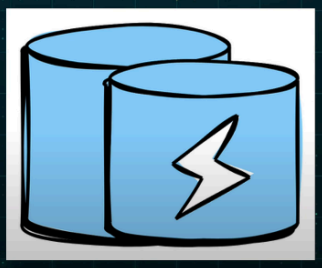

2. LOAD BALANCERS
"Distribute incoming requests to different servers to ensure no single server gets overwhelmed."

Different types of Databases



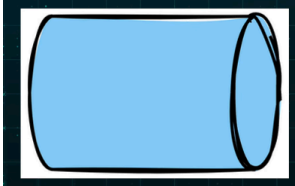
Most commonly used DBs

4. CACHING



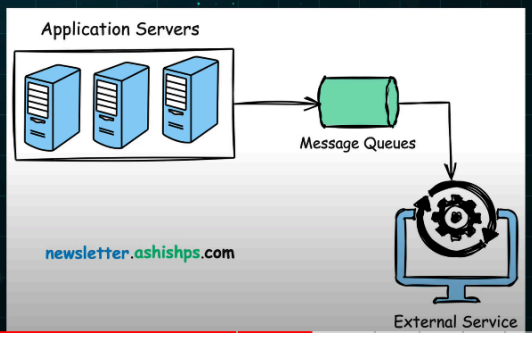
"Store frequently accessed data in a fast access storage to reduce the load on primary data source and improve response times."

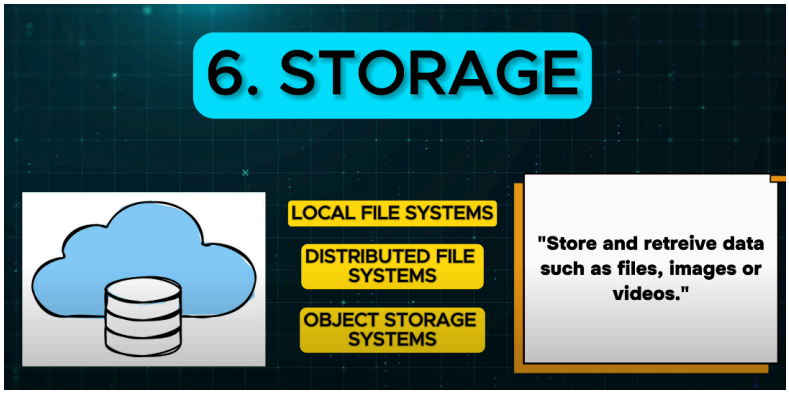
5. MESSAGE QUEUES

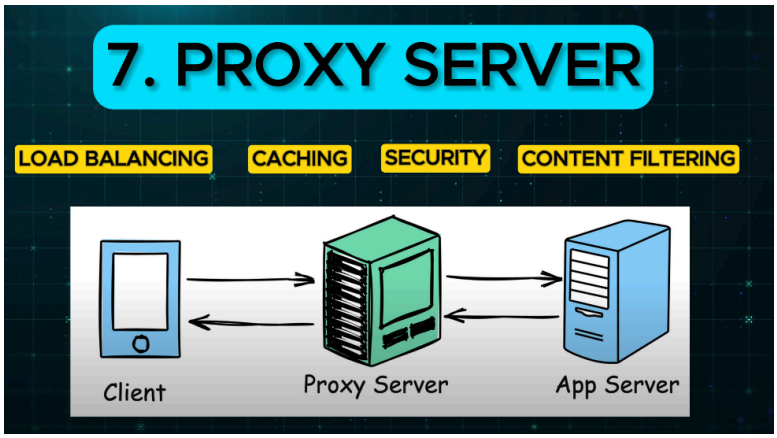


"Enable asynchronous communication between system components"

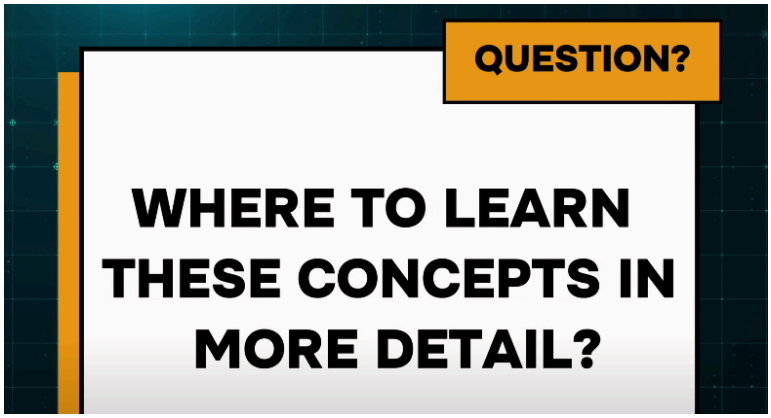
5. MESSAGE QUEUES




6. STORAGE
LOCAL FILE SYSTEMS
DISTRIBUTED FILE SYSTEMS
OBJECT STORAGE SYSTEMS
"Store and retreive data such as files, images or videos."

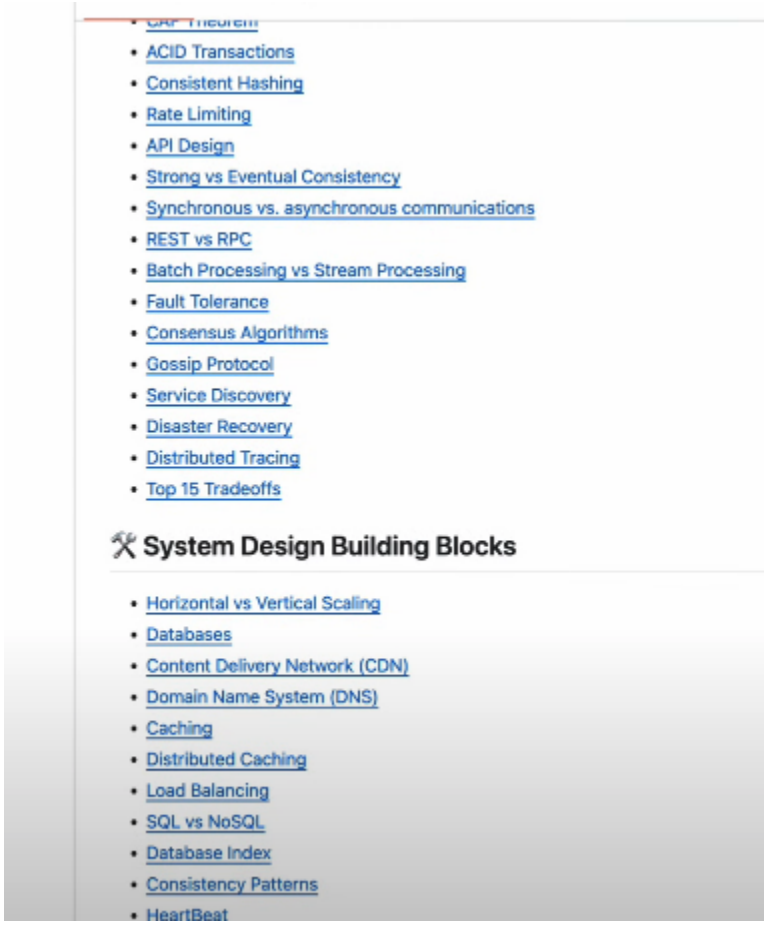

7. PROXY SERVER
LOAD BALANCING
CACHING
SECURITY
CONTENT FILTERING
Client
Proxy Server
App Server


8. CDN
DigitalOcean
Content Delivery Network (CDN)

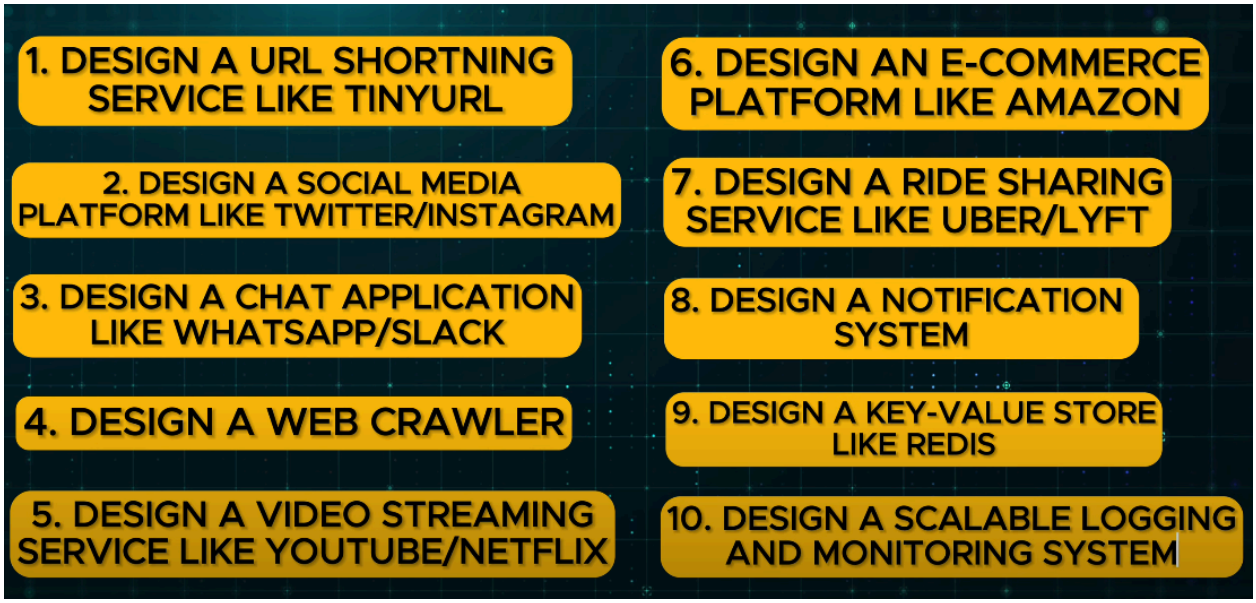


https://github.com/ashishps1/awesome-system-design-resources

Reading these links will give more insight.
Study multiple times to get grip fastly and to understand better



Most common systems Designs



The expectation from system design is

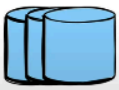
2. ESTIMATE THE CAPACITY

Step 2: Capacity Estimation

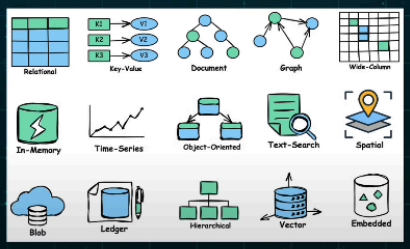
After clarifying the requirements, it is important to estimate the capacity of the system you are going to design.

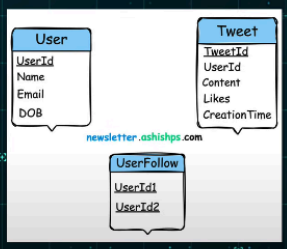
Estimating the scale upfront helps guide your design decisions and ensures that the system can meet the desired criteria.

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



3. CHOOSE THE RIGHT DATABASE





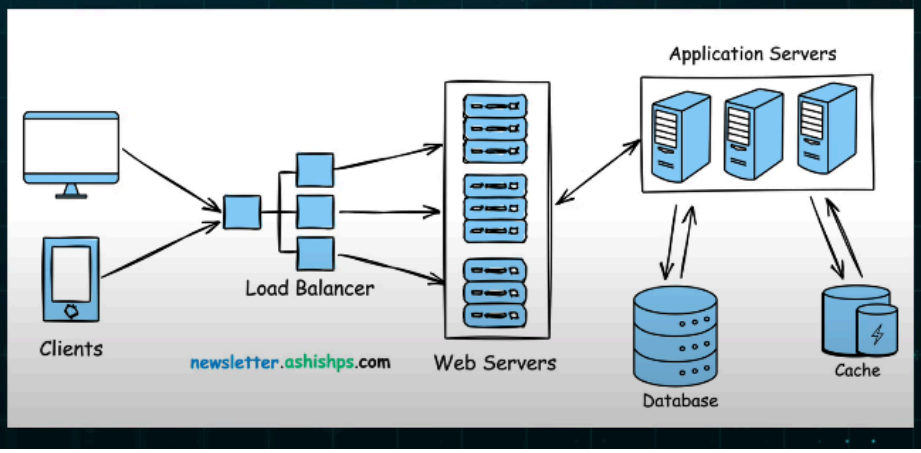
4. DESIGN APIS AND REQUEST/RESPONSE PATTERN

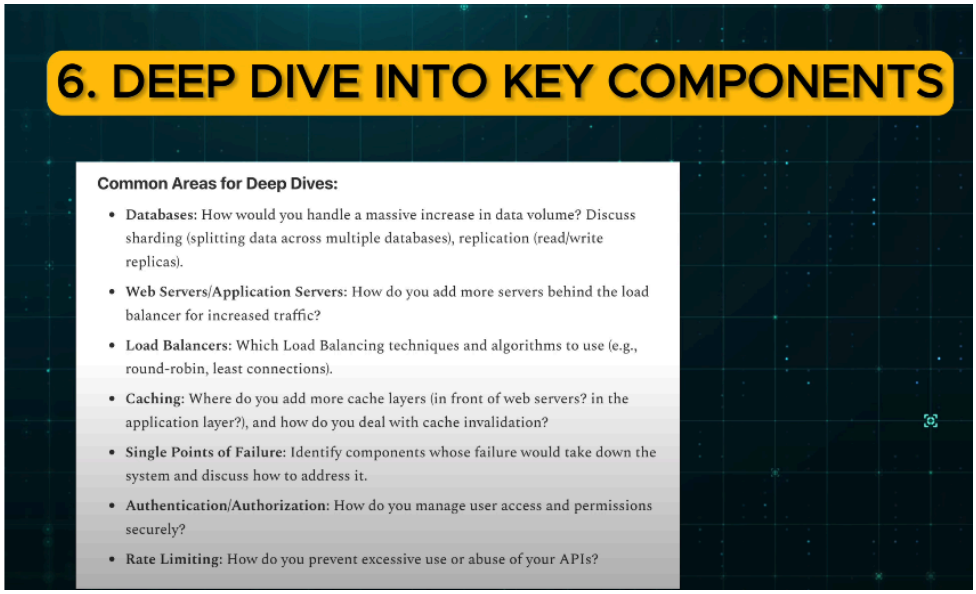
Define the API Endpoints:

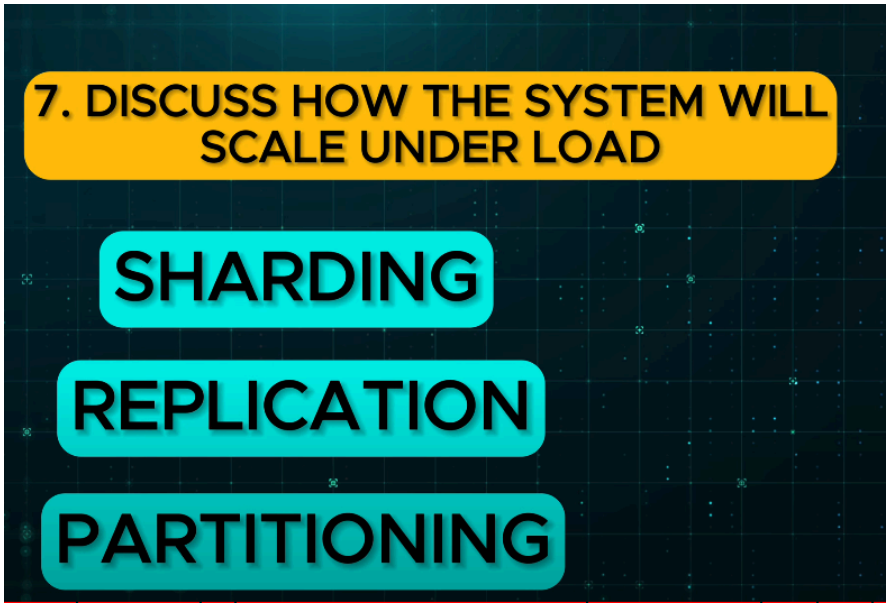
createProfile(name, email, password)
login(email, password)
postTweet(userId, content, timestamp)
followUser(userId1, userId2)

Example: Twitter APIs

5. SKETCH OUT A HIGH-LEVEL BLOCK DIAGRAM



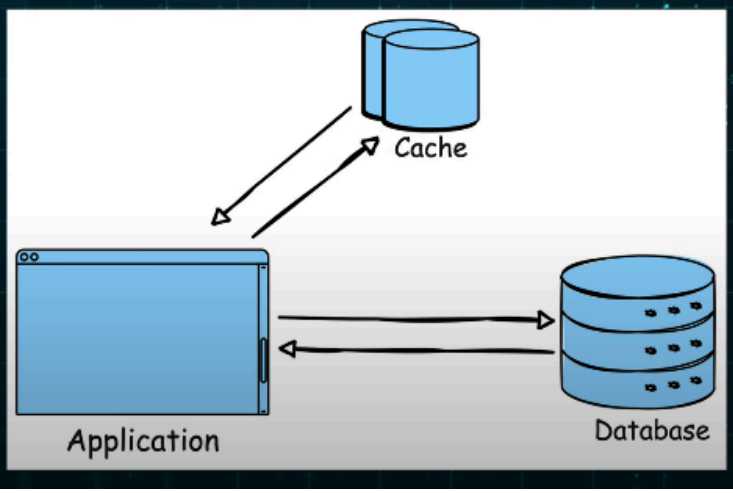

6. DEEP DIVE INTO KEY COMPONENTS
Common Areas for Deep Dives:
Databases: How would you handle a massive increase in data volume? Discuss sharding (splitting data across multiple databases), replication (read/write replicas).
Web Servers/Application Servers: How do you add more servers behind the load 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

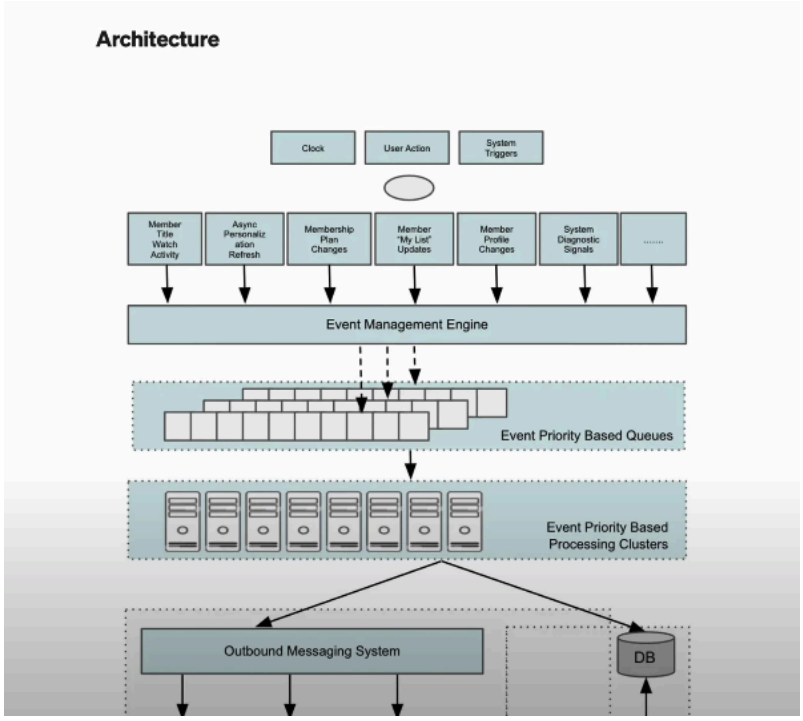
8. DISCUSS TRADOFFS

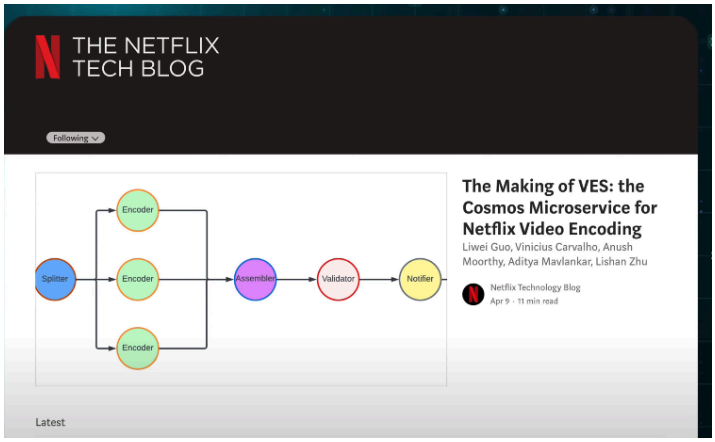
SQL vs NoSQL

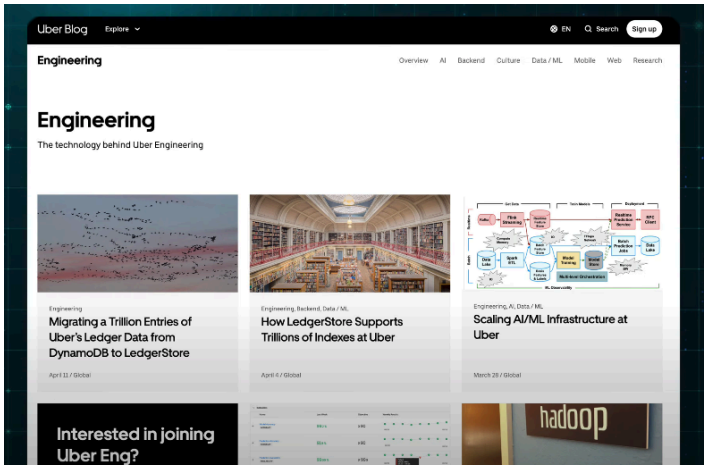
9. DISCUSS CACHING

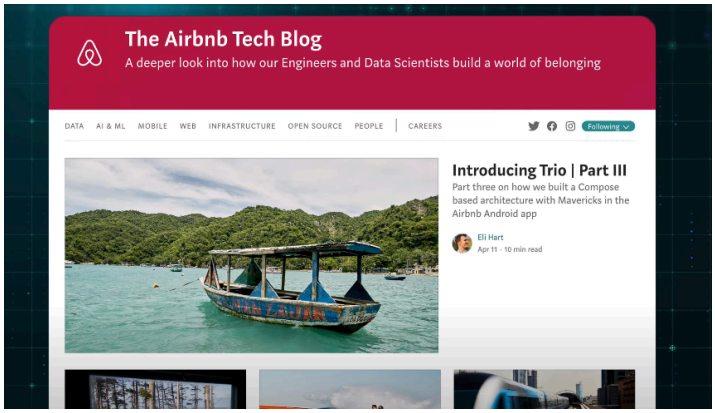


Read company Engineer blobs




THE NETFLIX
TECH BLOG
Following
Encoder
Encoder
Encoder
Splitter
Assembler
Validator
Notifier
The Making of VES: the Cosmos Microservice for Netflix Video Encoding
Liwei Guo, Vinicius Carvalho, Anush Moorthy, Aditya Mavlankar, Lishan Zhu
Netflix Technology Blog
Apr 9 · 11 min read
Latest


Uber Blog
Explore
EN
Search
Sign up
Engineering
Overview
AI
Backend
Culture
Data / ML
Mobile
Web
Research
Engineering
The technology behind Uber Engineering
Migrating a Trillion Entries of Uber's Ledger Data from DynamoDB to LedgerStore
How LedgerStore Supports Trillions of Indexes at Uber
Scaling AI/ML Infrastructure at Uber
Interested in joining Uber Eng?
hadoop


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