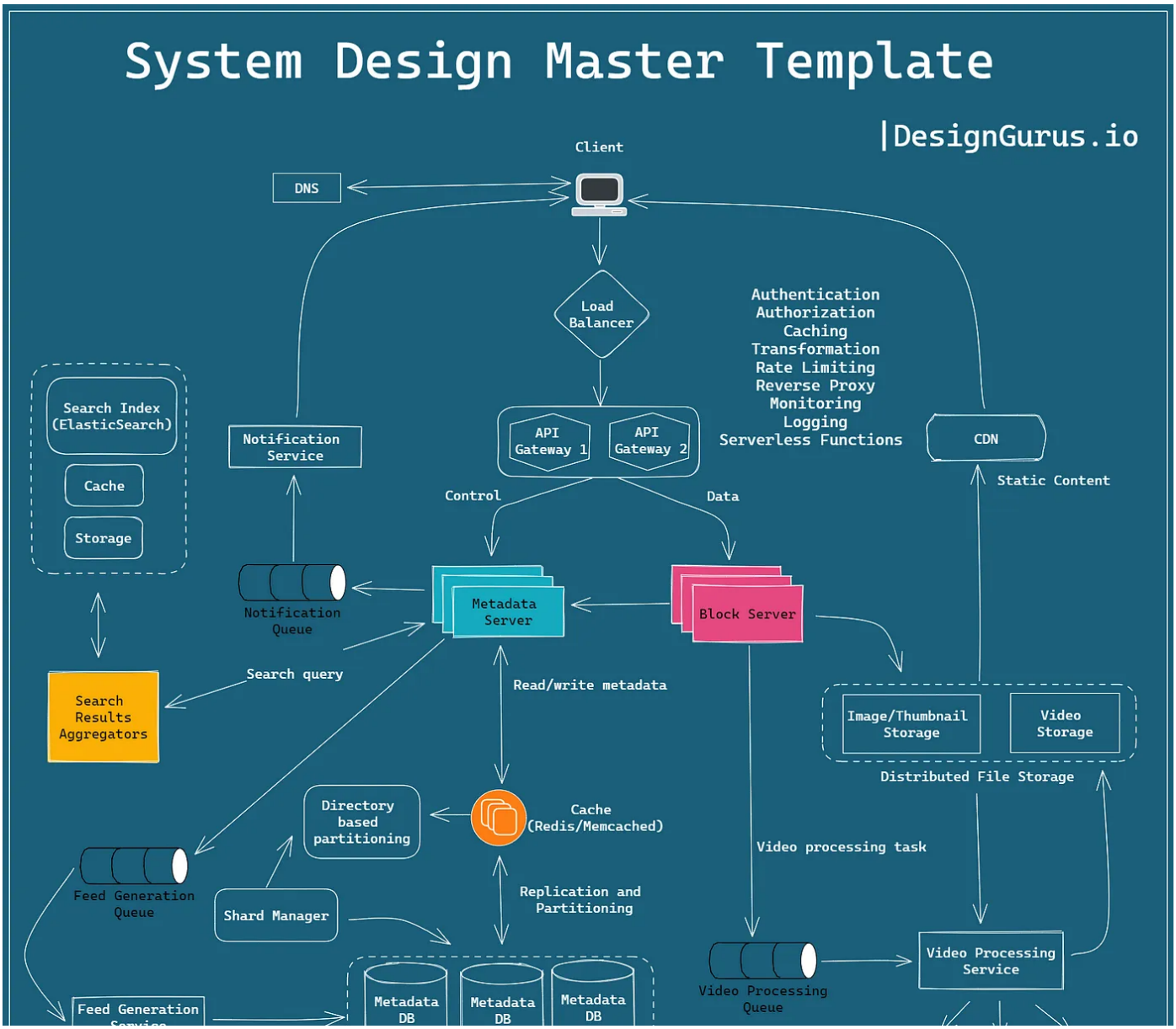
✦ Member-only story

16 System Design Concepts I Wish I Knew Before the Interview.

Arslan Ahmad · Follow
Published in Level Up Coding
13 min read · Apr 3

Listen Share

Mastering System Design Interview: Essential Concepts for Every Software Engineer



System Design Master Template

To excel in system design, one of the most crucial aspects is to develop a deep understanding of fundamental system design concepts such as **Load Balancing**, **Caching**, **Partitioning**, **Replication**, **Databases**, and **Proxies**.

Through my own experiences, I've identified 16 key concepts that can make a significant difference in your ability to tackle system design problems. These concepts range from understanding the intricacies of API gateway and mastering load-balancing techniques to grasping the importance of CDNs and appreciating the role of caching in modern distributed systems. By the end of this blog, you'll have a comprehensive understanding of these essential ideas and the confidence to apply them in your next interview.

System design interviews are unstructured by nature. During the interview, it is difficult to keep track of things and be sure that you have touched upon all the essential aspects of the design. To simplify this process, I have developed a **system design master template** that should guide you in answering any system design interview question. Take a look at the featured image to gain insight into the key components that may be involved in any system design.

Keeping this master template in mind, we will discuss the 16 essential system design concepts. Here is their brief description:

1. Domain Name System (DNS)
2. Load Balancer
3. API Gateway
4. CDN
5. Forward Proxy vs. Reverse Proxy
6. Caching
7. Data Partitioning
8. Database Replication
9. Distributed Messaging Systems
10. Microservices
11. NoSQL Databases
12. Database Index
13. Distributed File Systems
14. Notification System

1. Domain Name System (DNS)

Domain Name System (DNS) is a fundamental component of the internet infrastructure that translates human-friendly domain names into their corresponding IP addresses. It functions like a phonebook for the internet, allowing users to access websites and services by typing in easily memorable domain names, such as `www.designgurus.io` rather than the numerical IP addresses like “192.0.2.1” that computers use to identify each other.

When you enter a domain name into your web browser, the DNS is responsible for locating the associated IP address and directing your request to the correct server. The process begins with your computer sending a query to a recursive resolver, which then searches a series of DNS servers, starting with the root server, followed by the Top-Level Domain (TLD) server, and finally the authoritative name server. Once the IP address is found, the recursive resolver returns it to your computer, allowing your browser to establish a connection with the target server and access the desired content.

DNS Resolver

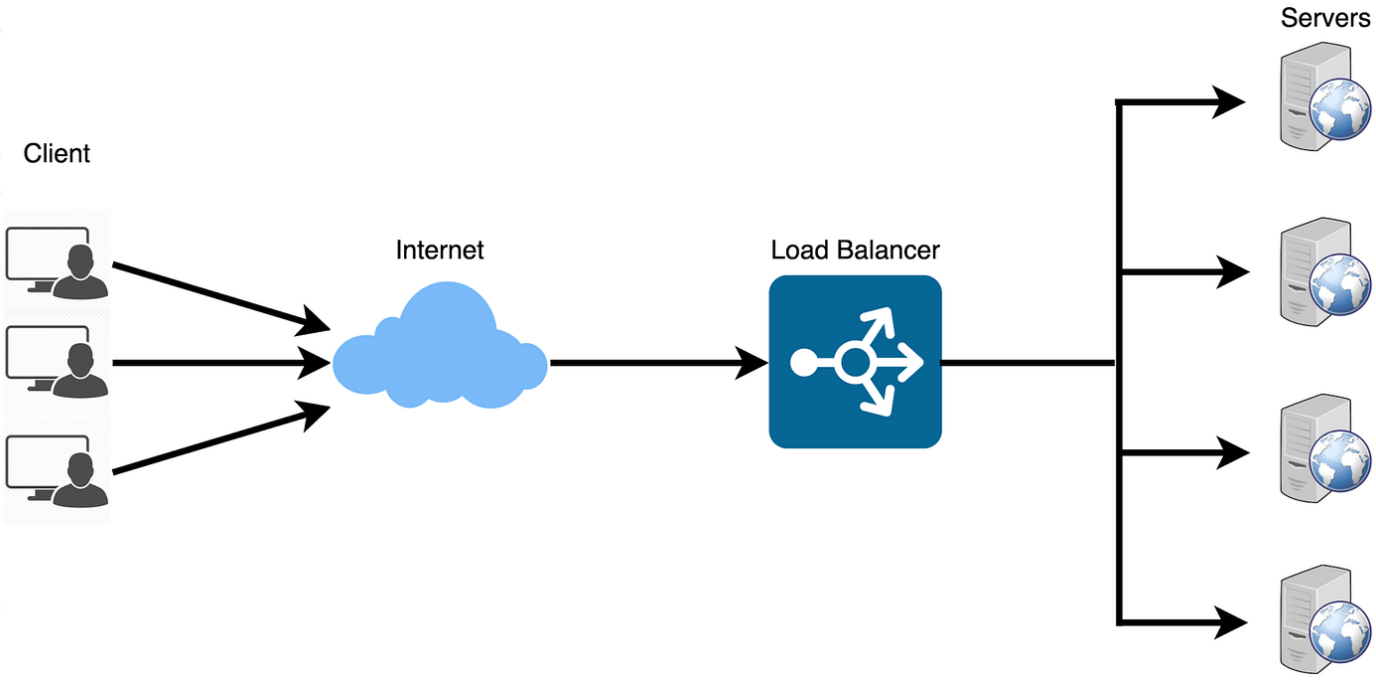
2. Load Balancer

A load balancer is a networking device or software that distributes incoming network traffic across multiple servers to ensure optimal resource utilization, reduce latency, and maintain high availability. It plays a vital role in scaling applications and managing server workloads efficiently, especially in situations where there is a sudden spike in traffic or uneven distribution of requests among servers.

Load balancers use different algorithms to determine how to distribute incoming traffic. Common algorithms include:

1. **Round Robin:** Requests are distributed sequentially and evenly across all available servers in a cyclical manner.
2. **Least Connections:** The load balancer assigns requests to the server with the fewest active connections, prioritizing less-busy servers.
3. **IP Hash:** The client's IP address is hashed, and the resulting value is used to determine which server the request should be directed to. This method ensures

that a specific client's requests are always routed to the same server, helping maintain session persistence.



Load Balancer

3. API Gateway

An API Gateway is a server or service that acts as an intermediary between external clients and the internal microservices or API-based backend services of an application. It is a crucial component in modern architectures, especially in microservices-based systems, where it simplifies the communication process and provides a single entry point for clients to access various services.

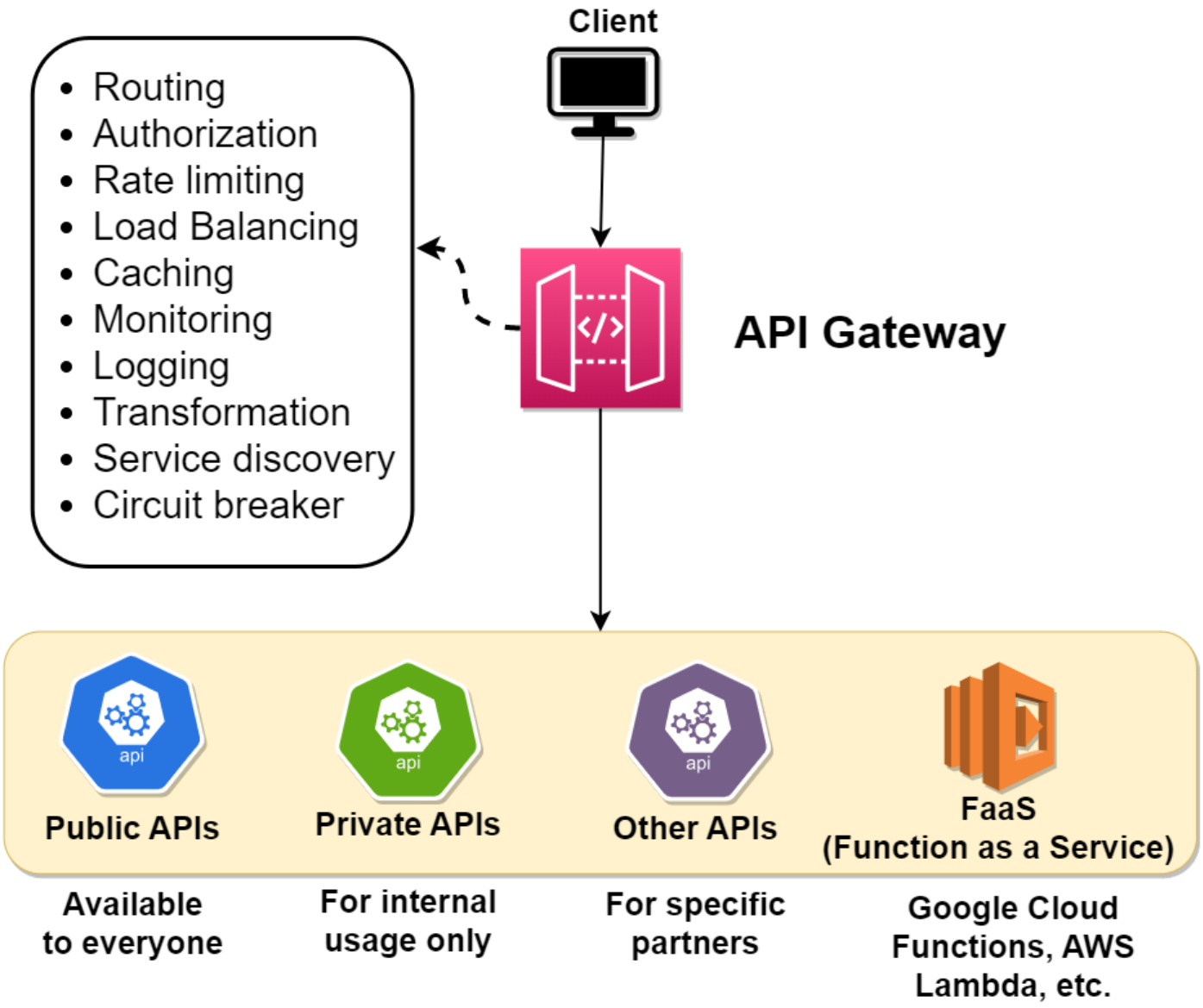
The main functions of an API Gateway include:

1. **Request Routing:** It directs incoming API requests from clients to the appropriate backend service or microservice, based on predefined rules and configurations.
2. **Authentication and Authorization:** The API Gateway can handle user authentication and authorization, ensuring that only authorized clients can access the services. It can verify API keys, tokens, or other credentials before routing requests to the backend services.

3. **Rate Limiting and Throttling:** To protect backend services from excessive load or abuse, the API Gateway can enforce rate limits or throttle requests from clients based on predefined policies.

4. **Caching:** To reduce latency and backend load, the API Gateway can cache frequently-used responses, serving them directly to clients without the need to query the backend services.

5. **Request and Response Transformation:** The API Gateway can modify requests and responses, such as converting data formats, adding or removing headers, or modifying query parameters, to ensure compatibility between clients and services.



API Gateway

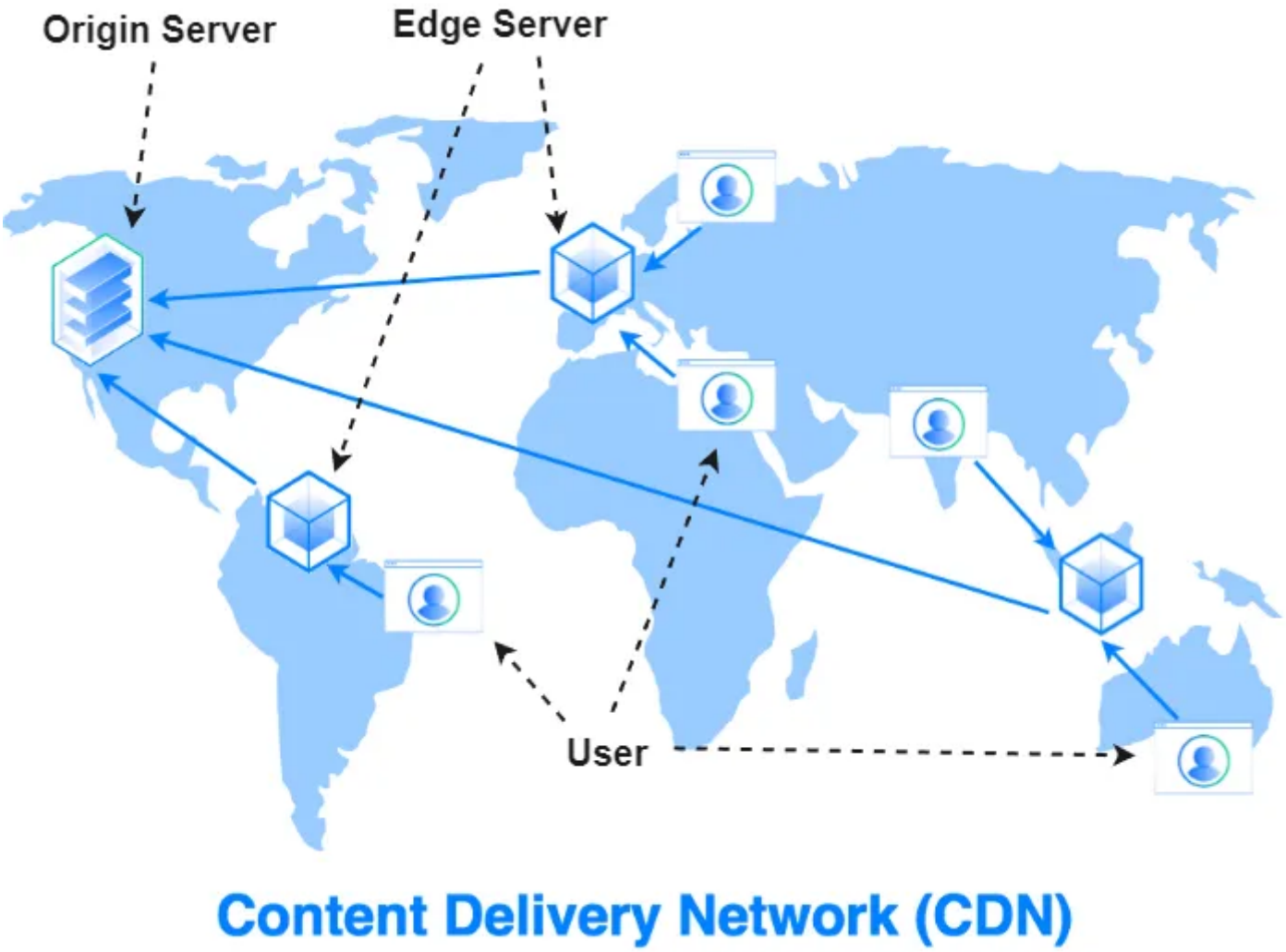
> *Check **Grokking the System Design Interview** for a list of common system design interview questions and basic concepts.*

4. CDN

A Content Delivery Network (CDN) is a distributed network of servers that store and deliver content, such as images, videos, stylesheets, and scripts, to users from geographically closer locations. CDNs are designed to improve the performance, speed, and reliability of content delivery to end-users, regardless of their location relative to the origin server.

Here's how a CDN works:

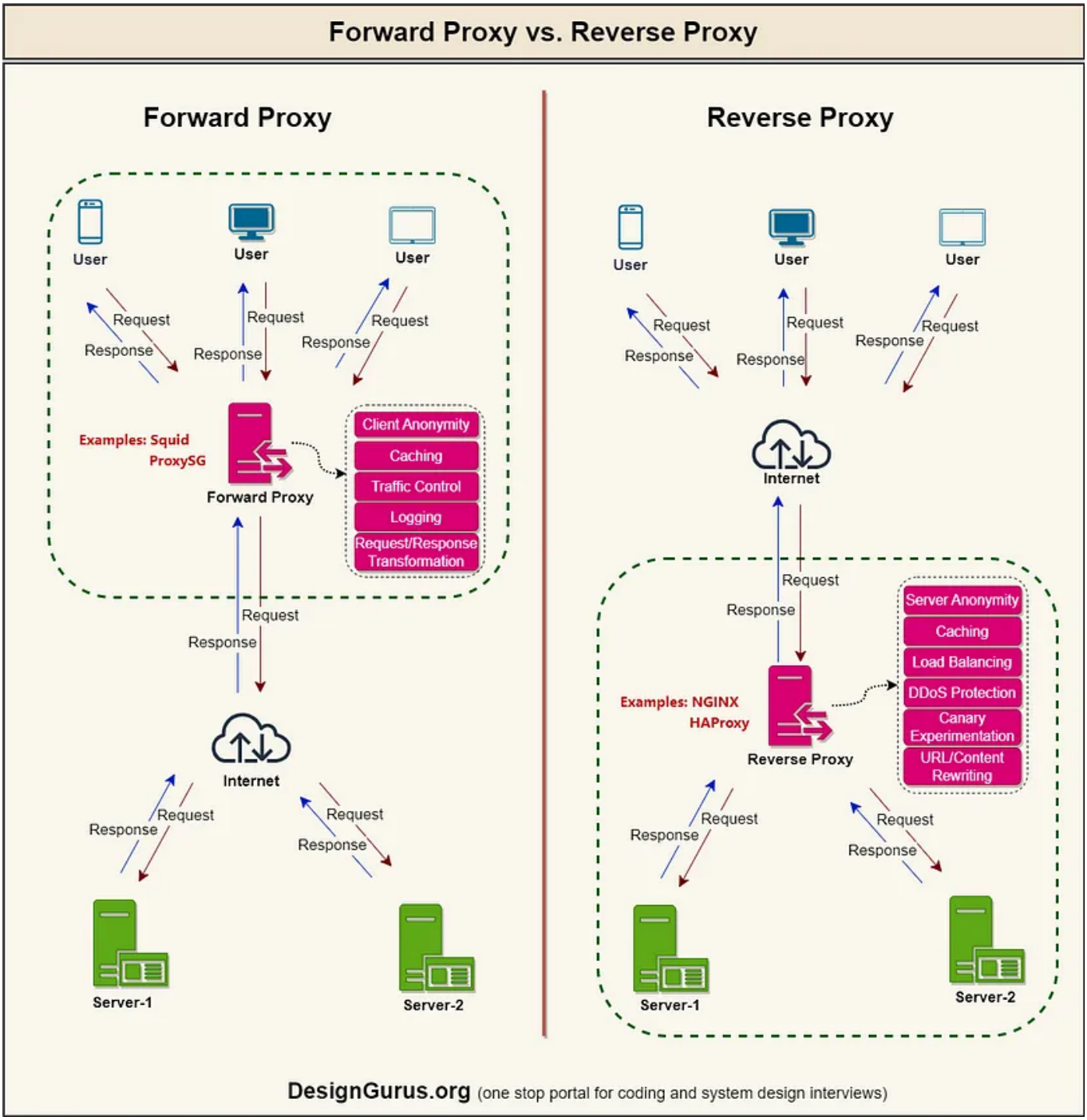
1. When a user requests content from a website or application, the request is directed to the nearest CDN server, also known as an edge server.
2. If the edge server has the requested content cached, it directly serves the content to the user. This reduces latency and improves the user experience, as the content travels a shorter distance.
3. If the content is not cached on the edge server, the CDN retrieves it from the origin server or another nearby CDN server. Once the content is fetched, it is cached on the edge server and served to the user.
4. To ensure the content remains up-to-date, the CDN periodically checks the origin server for changes and updates its cache accordingly.



5. Forward Proxy vs. Reverse Proxy

A **forward proxy**, also known as a "proxy server," or simply "proxy," is a server that sits in front of one or more client machines and acts as an intermediary between the clients and the internet. When a client machine makes a request to a resource on the internet, the request is first sent to the forward proxy. The forward proxy then forwards the request to the internet on behalf of the client machine and returns the response to the client machine.

A **reverse proxy** is a server that sits in front of one or more web servers and acts as an intermediary between the web servers and the Internet. When a client makes a request to a resource on the internet, the request is first sent to the reverse proxy. The reverse proxy then forwards the request to one of the web servers, which returns the response to the reverse proxy. The reverse proxy then returns the response to the client.



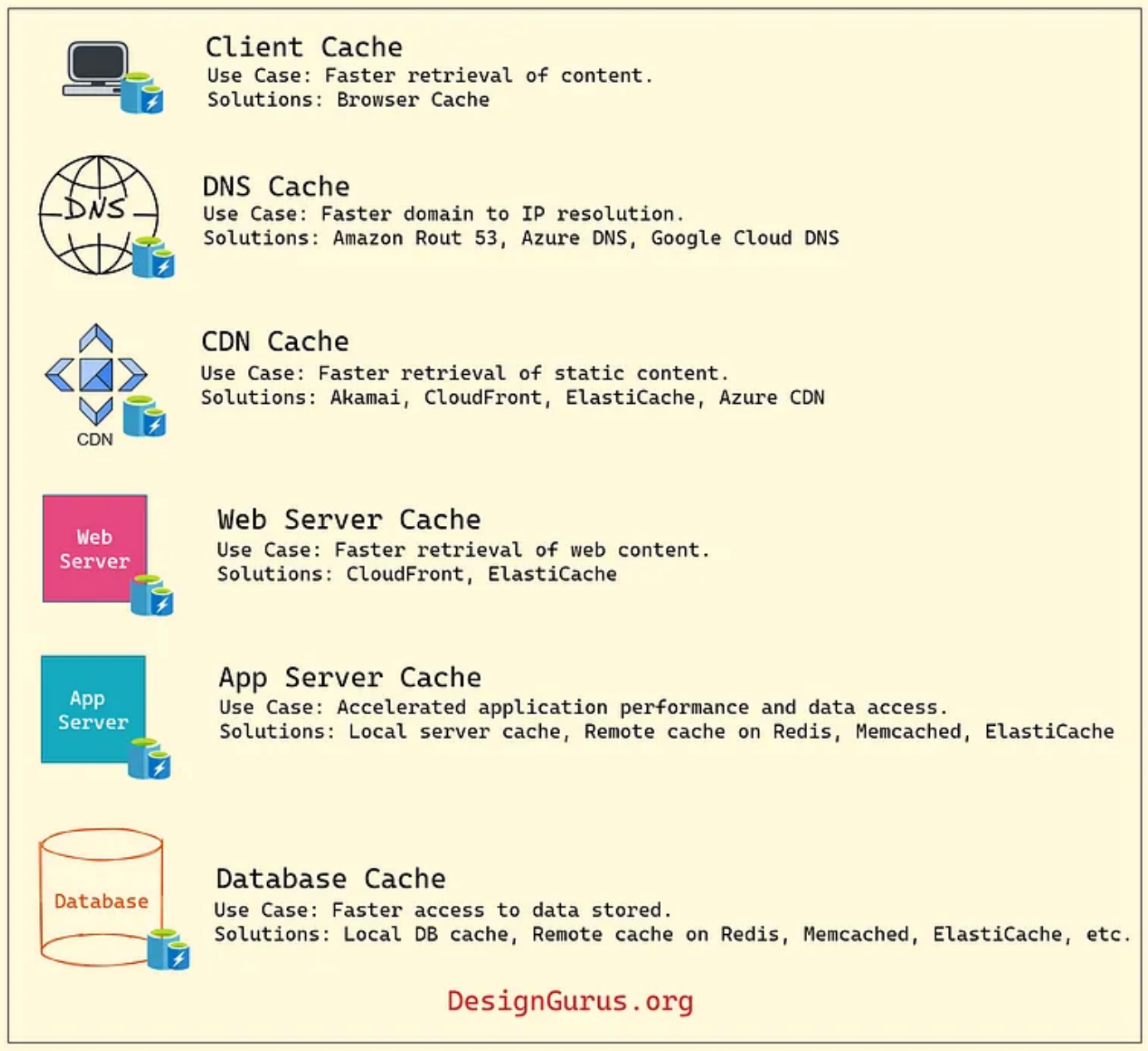
Forward Proxy vs. Reverse Proxy

> *Check **Grokking the Advanced System Design Interview** for architectural reviews of famous distributed systems.*

6. Caching

The cache is a high-speed storage layer that sits between the application and the original source of the data, such as a database, a file system, or a remote web service. When data is requested by the application, it is first checked in the cache. If the data is

found in the cache, it is returned to the application. If the data is not found in the cache, it is retrieved from its original source, stored in the cache for future use, and returned to the application. In a distributed system, caching can be done at multiple places for example, Client, DNS, CDN, Load Balancer, API Gateway, Server, Database, etc.



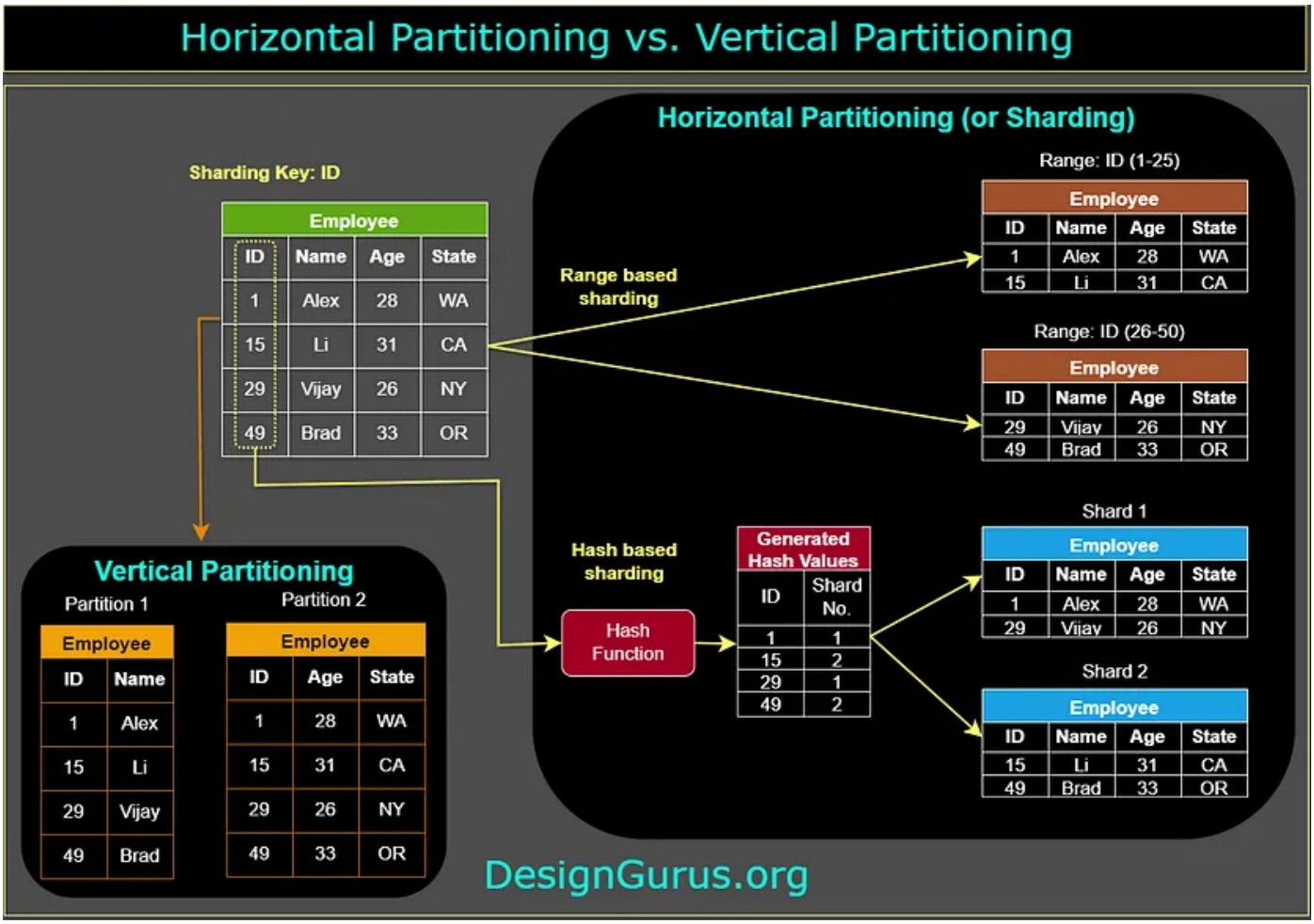
What are where to cache

7. Data Partitioning

In a database, **horizontal partitioning**, also known as **sharding**, involves dividing the rows of a table into smaller tables and storing them on different servers or database

instances. This is done to distribute the load of a database across multiple servers and to improve performance.

On the other hand, **vertical partitioning**, involves dividing the columns of a table into separate tables. This is done to reduce the number of columns in a table and to improve the performance of queries that only access a small number of columns.



Data partitioning

8. Database Replication

Database replication is a technique used to maintain multiple copies of the same database across different servers or locations. The primary purpose of database replication is to improve data availability, redundancy, and fault tolerance, ensuring that the system continues to function even in the case of hardware failures or other issues.

In a replicated database setup, one server acts as the primary (or master) database, while others function as replicas (or slaves). The process involves synchronizing data

between the primary database and replicas, so they all have the same up-to-date information. Database replication offers several benefits, including:

1. **Improved Performance:** By distributing read queries among multiple replicas, you can reduce the load on the primary database and improve query response times.
2. **High Availability:** In the event of a failure or downtime on the primary database, replicas can continue to serve data, ensuring uninterrupted access to the application.
3. **Enhanced Data Protection:** Having multiple copies of the database across different locations helps protect against data loss due to hardware failures or other disasters.
4. **Load Balancing:** Replicas can handle read queries, which allows for better load distribution and reduces the overall strain on the primary database.

9. Distributed Messaging Systems

Distributed messaging systems enable the exchange of messages between multiple, potentially geographically-dispersed applications, services, or components in a reliable, scalable, and fault-tolerant manner. They facilitate communication by decoupling the sender and receiver components, allowing them to evolve and operate independently. Distributed messaging systems are particularly useful in large-scale or complex systems, such as those found in microservices architectures or distributed computing environments. Examples of such systems are Apache Kafka and RabbitMQ.

10. Microservices

Microservices are an architectural style in which an application is structured as a collection of small, loosely-coupled, and independently deployable services. Each microservice is responsible for a specific piece of functionality or domain within the application, and communicates with other microservices through well-defined APIs. This approach is a departure from the traditional monolithic architecture, where an application is built as a single, tightly-coupled unit.

The main characteristics of microservices are:

1. **Single Responsibility:** Each microservice focuses on a specific functionality or domain, adhering to the Single Responsibility Principle. This makes the services

easier to understand, develop, and maintain.

2. **Independence:** Microservices can be developed, deployed, and scaled independently of one another. This allows for increased flexibility and agility in the development process, as teams can work on different services concurrently without impacting the entire system.

3. **Decentralized:** Microservices are typically decentralized, with each service owning its data and business logic. This encourages separation of concerns and enables teams to make decisions and choose technologies that best suit their specific requirements.

4. **Communication:** Microservices communicate with each other using lightweight protocols such as HTTP/REST, gRPC, or message queues. This promotes interoperability and makes it easier to integrate new services or replace existing ones.

5. **Fault Tolerance:** Since microservices are independent, a failure in one service does not necessarily cause the entire system to fail. This can help improve the overall resiliency of the application.

11. NoSQL Databases

NoSQL databases, or "Not Only SQL" databases, are non-relational databases designed to store, manage, and retrieve unstructured or semi-structured data. They offer an alternative to traditional relational databases, which rely on structured data and predefined schemas. NoSQL databases have become popular due to their flexibility, scalability, and ability to handle large volumes of data, making them well-suited for modern applications, big data processing, and real-time analytics.

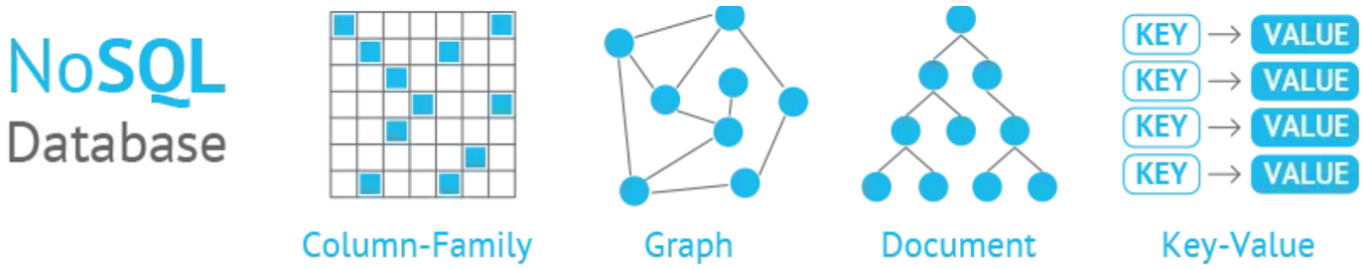
NoSQL databases can be categorized into four main types:

1. **Document-Based:** These databases store data in document-like structures, such as JSON or BSON. Each document is self-contained and can have its own unique structure, making them suitable for handling heterogeneous data. Examples of document-based NoSQL databases include MongoDB and Couchbase.

2. **Key-Value:** These databases store data as key-value pairs, where the key acts as a unique identifier, and the value holds the associated data. Key-value databases are highly efficient for simple read and write operations, and they can be easily partitioned and scaled horizontally. Examples of key-value NoSQL databases include Redis and Amazon DynamoDB.

3. **Column-Family:** These databases store data in column families, which are groups of related columns. They are designed to handle write-heavy workloads and are highly efficient for querying data with a known row and column keys. Examples of column-family NoSQL databases include Apache Cassandra and HBase.

4. **Graph-Based:** These databases are designed for storing and querying data that has complex relationships and interconnected structures, such as social networks or recommendation systems. Graph databases use nodes, edges, and properties to represent and store data, making it easier to perform complex traversals and relationship-based queries. Examples of graph-based NoSQL databases include Neo4j and Amazon Neptune.



Types of NoSQL databases

12. Database Index

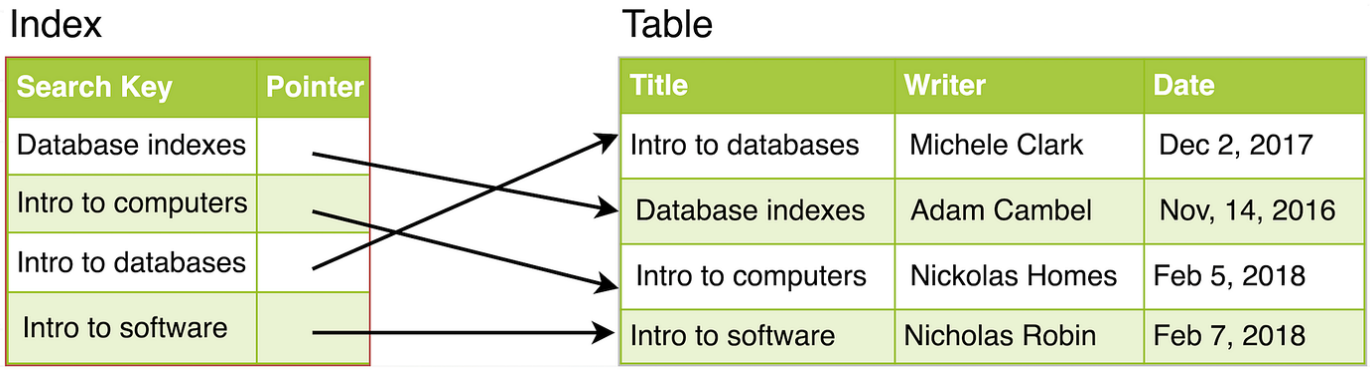
Database indexes are data structures that improve the speed and efficiency of querying operations in a database. They work similarly to an index in a book, allowing the database management system (DBMS) to quickly locate the data associated with a specific value or set of values, without having to search through every row in a table. By providing a more direct path to the desired data, indexes can significantly reduce the time it takes to retrieve information from a database.

Indexes are usually built on one or more columns of a database table. The most common type of index is the B-tree index, which organizes data in a hierarchical tree

structure, allowing for fast search, insertion, and deletion operations. There are other types of indexes, such as bitmap indexes and hash indexes, each with their specific use cases and advantages.

While indexes can significantly improve query performance, they also have some trade-offs:

1. **Storage Space:** Indexes consume additional storage space, as they create and maintain separate data structures alongside the original table data.
2. **Write Performance:** When data is inserted, updated, or deleted in a table, the associated indexes must also be updated, which can slow down write operations.



Database Index

13. Distributed File Systems

Distributed file systems are storage solutions designed to manage and provide access to files and directories across multiple servers, nodes, or machines, often distributed over a network. They enable users and applications to access and manipulate files as if they were stored on a local file system, even though the actual files might be physically stored on multiple remote servers. Distributed file systems are often used in large-scale or distributed computing environments to provide fault tolerance, high availability, and improved performance.

14. Notification System

These are used to send notifications or alerts to users, such as emails, push notifications, or text messages.

15. Full-text Search

Full-text search enables users to search for specific words or phrases within an app or website. When a user queries, the app or website returns the most relevant results. To do this quickly and efficiently, full-text search relies on an inverted index, which is a data structure that maps words or phrases to the documents in which they appear. An example of such systems is Elastic Search.

16. Distributed Coordination Services

Distributed coordination services are systems designed to manage and coordinate the activities of distributed applications, services, or nodes in a reliable, efficient, and fault-tolerant manner. They help maintain consistency, handle distributed synchronization, and manage the configuration and state of various components in a distributed environment. Distributed coordination services are particularly useful in large-scale or complex systems, such as those found in microservices architectures, distributed computing environments, or clustered databases. Examples of such service are Apache ZooKeeper, etcd, Consul.

Conclusion

Maximize your chances of acing system design interviews by using the aforementioned system design concepts and the template. Here is a list of common **system design interview questions**:

1. Designing a File-sharing Service Like Google Drive or Dropbox.
2. Designing a Video Streaming Platform
3. Designing a URL Shortening Service
4. Designing a Web Crawler
5. Designing Uber
6. Designing Facebook Messenger
7. Designing Twitter Search

Take a look at **Grokking the System Design Interview** for a detailed discussion of such system design interview questions.

Check **Grokking System Design Fundamentals** for a list of common system design concepts.

https://www.designgurus.io/blog/system-design-interview-fundamentals

To learn software architecture and practice advanced system design interview questions take a look at **Grokking the Advanced System Design Interview**.

Keep learn more on system design interviews:

10 System Design Interview Questions (With Answers) I Wished I Knew Before the Interview

Prepare these 10 system design questions before the interview to multiply your chances of success.

levelup.gitconnected.com

System Design Interview Survival Guide (2023): Preparation Strategies and Practical Tips

System Design Interview Preparation: Mastering the Art of System Design

levelup.gitconnected.com

16 System Design Concepts I Wish I Knew Before the Interview.

Mastering System Design Interview: Essential Concepts for Every Software Engineer

levelup.gitconnected.com

Thanks for reading

- View my content on Coding and System Design Interviews
- 🔔 Follow me: LinkedIn | Twitter | Newsletter

Technology Coding System Design Interview Data Science

Follow

Written by Arslan Ahmad

7.5K Followers · Writer for Level Up Coding

Founder www.designgurus.io | Formally a software engineer @ Facebook, Microsoft, Hulu, Formulatrix | Entrepreneur, Software Engineer, Writer.

More from Arslan Ahmad and Level Up Coding

 Arslan Ahmad in Level Up Coding

I Wish I Knew These 12 Algorithms and Their Applications Before the System Design Interview

How to Master Algorithmic Patterns for System Design Interviews?

✦ · 11 min read · Apr 25

592 4

Sanjay Priyadarshi in Level Up Coding

I Spent 14 Days Studying A Programmer Who Built a $167 B Company—Here Are His Weird Rules To...

Steal This Programmer Blueprint

✦ · 11 min read · Apr 23

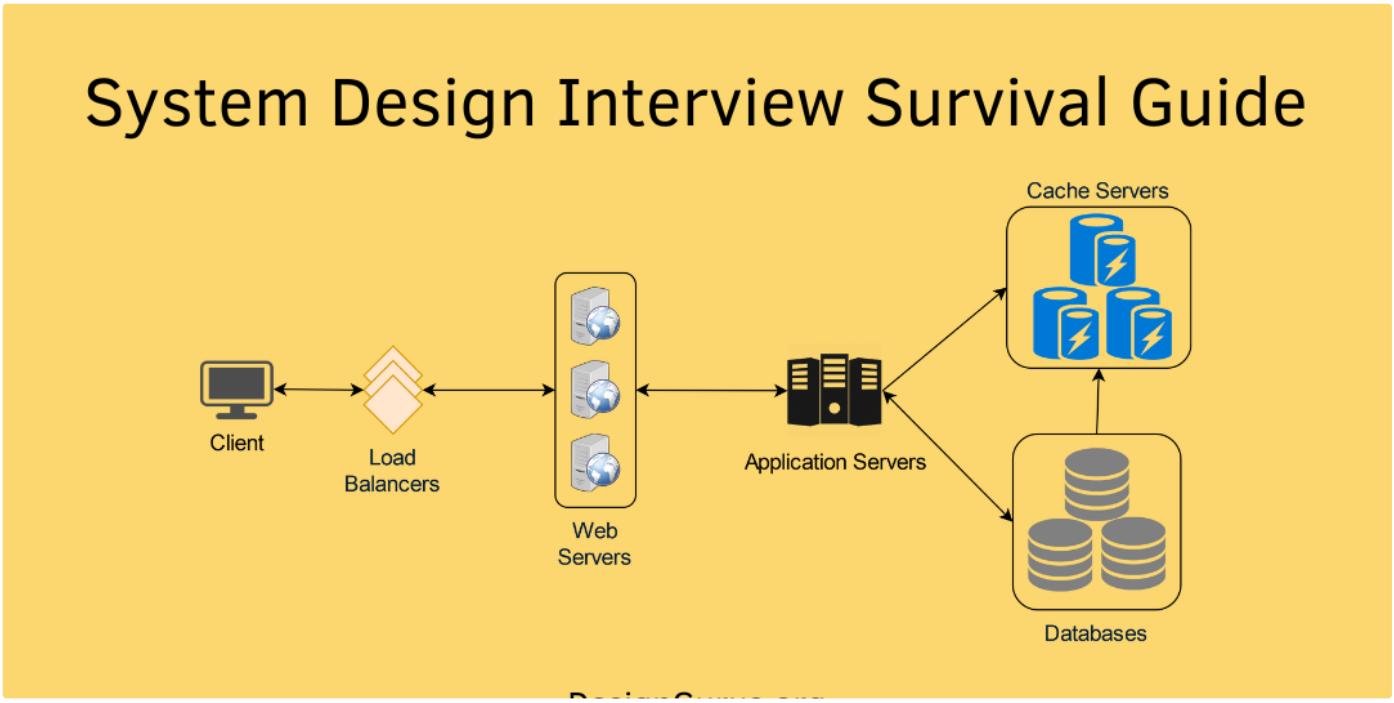
1.3K 16

Alexander Nguyen in Level Up Coding

Why I Keep Failing Candidates During Google Interviews...

They don't meet the bar.

· 4 min read · Apr 13



Arslan Ahmad in Level Up Coding

System Design Interview Survival Guide (2023): Preparation Strategies and Practical Tips

System Design Interview Preparation: Mastering the Art of System Design.

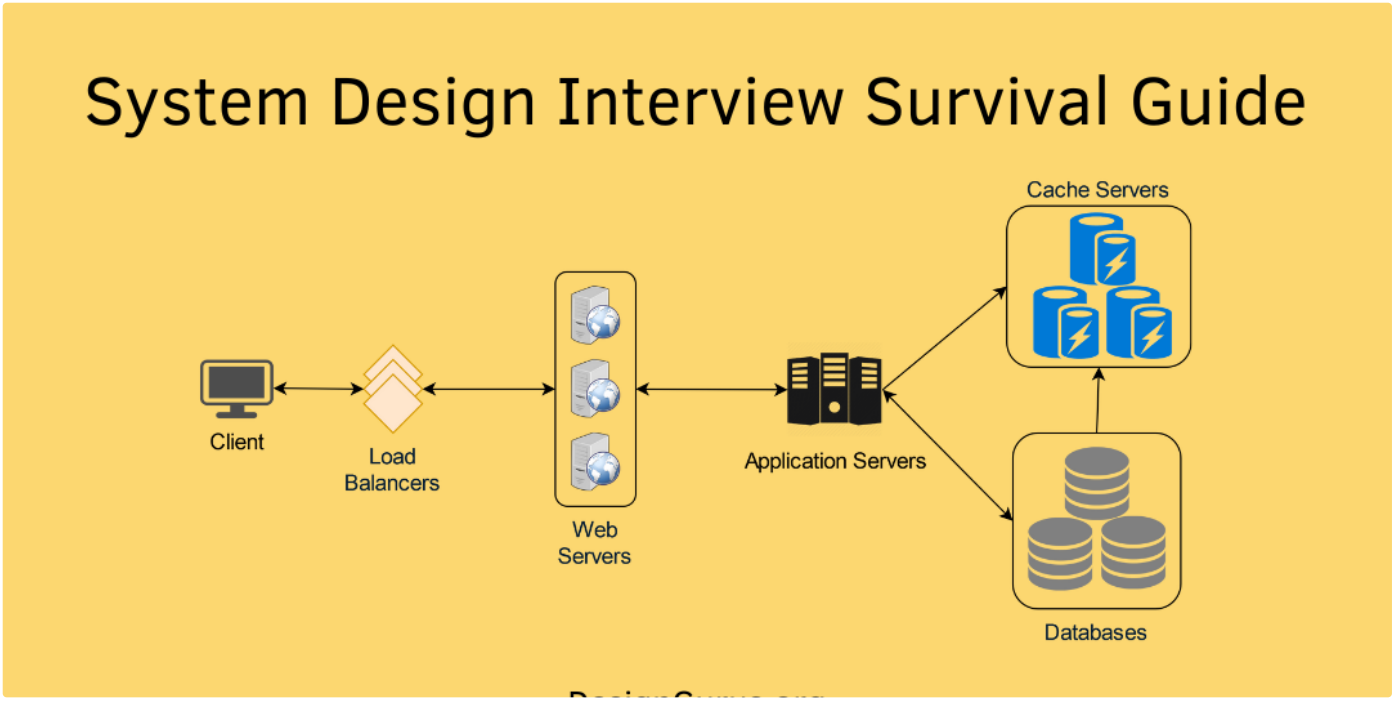
· 14 min read · Jan 19

1.4K 6

See all from Arslan Ahmad

See all from Level Up Coding

Recommended from Medium



Arslan Ahmad in Level Up Coding

System Design Interview Survival Guide (2023): Preparation Strategies and Practical Tips

System Design Interview Preparation: Mastering the Art of System Design.

✨ · 14 min read · Jan 19

1.4K 6

Alexander Nguyen in Level Up Coding

Why I Keep Failing Candidates During Google Interviews...

They don't meet the bar.

✨ · 4 min read · Apr 13

3.5K 108

Lists

Stories to Help You Grow as a Software Developer

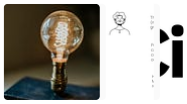
19 stories · 19 saves

What is ChatGPT?

9 stories · 21 saves

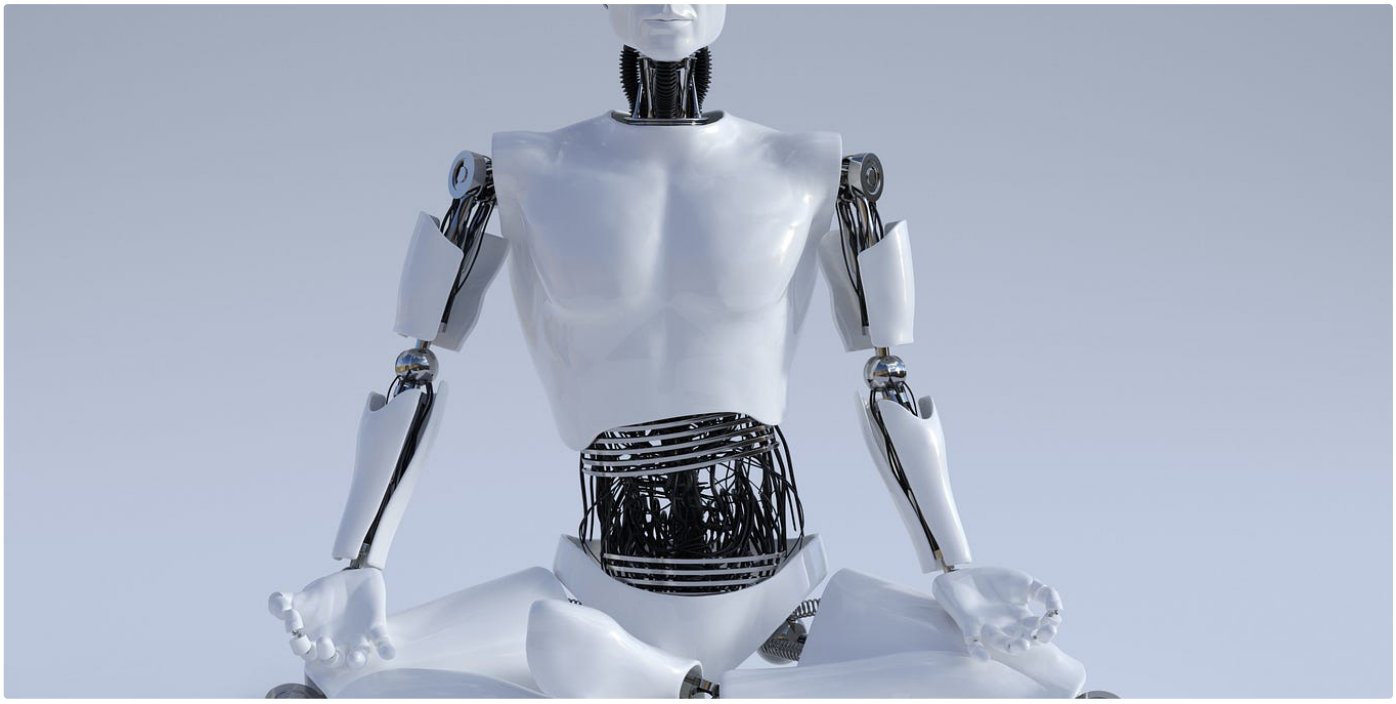
Stories to Help You Grow as a Designer

11 stories · 16 saves



Good Product Thinking

11 stories · 25 saves



The PyCoach in Artificial Corner

You're Using ChatGPT Wrong! Here's How to Be Ahead of 99% of ChatGPT Users

Master ChatGPT by learning prompt engineering.

· 7 min read · Mar 17

17.9K 326

RIPPLING

Santal Tech in Tech Pulse

$520K Offer from Rippling — Even After Failing a Coding Question?

I interviewed with Rippling in 2022 and received an offer of $520K (~$270K in base salary and ~$1M / 4 years in equity). I was surprised...

6 min read · Dec 31, 2022

413 9

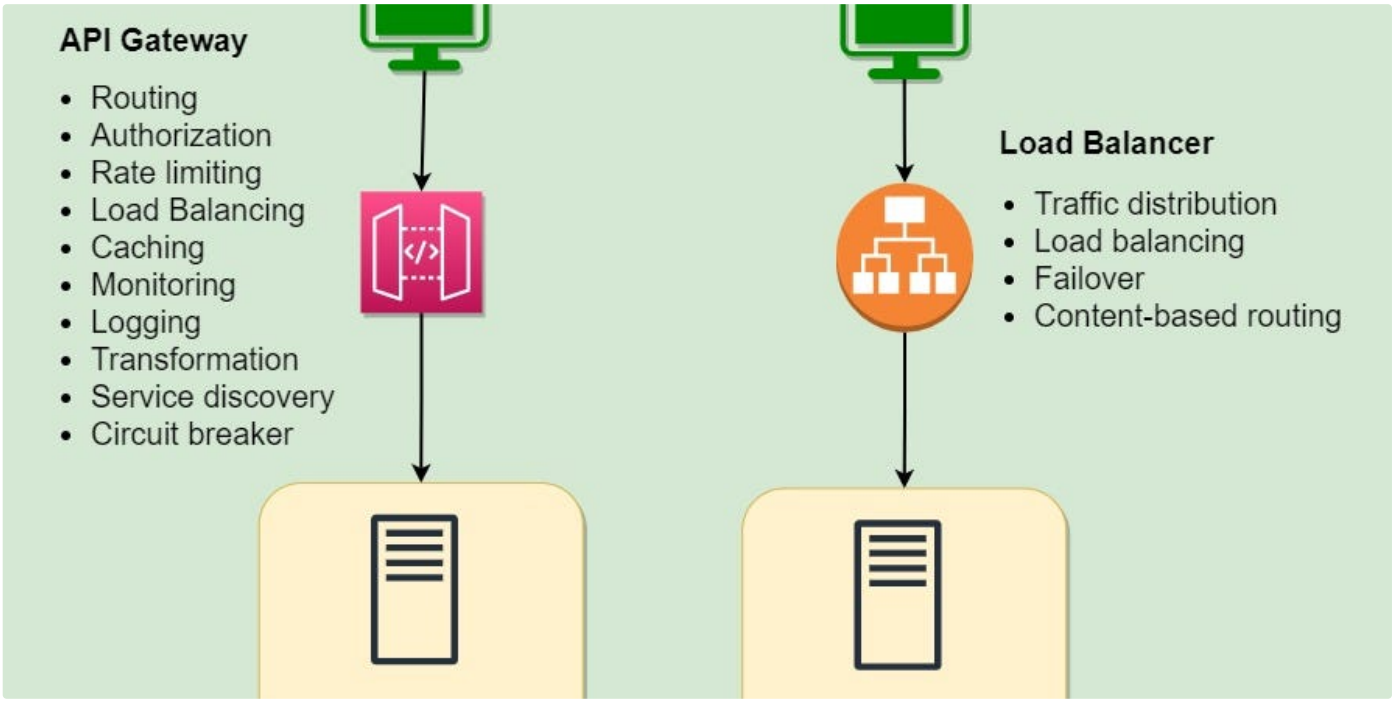
Hussein Nasser

How to Become a Good Backend Engineer (Fundamentals)

I have been a backend engineer for over 18 years and I witnessed technologies come and go but one thing always remain constant; The first...

· 11 min read · Dec 3, 2022

5.1K 36



Arslan Ahmad in Level Up Coding

System Design Interview Basics: Difference Between API Gateway and Load Balancer

Often, we come across software architectural components that are part of every system design and feel as though we don't have much...

· 7 min read · Dec 9, 2022

786 6