The System Design Newsletter

Subscribe Sign in

11 System Design Concepts Explained, Simply

#88: Break Into System Design (9 Minutes)

NEO KIM AND DR. ASHISH BAMANIA
SEP 16, 2025

190 9 31 Share

This post outlines 11 must-know system design concepts.

- *Share this post & I'll send you some rewards for the referrals.*

Systems design is an important part of all software engineering tasks you will work on in your career.

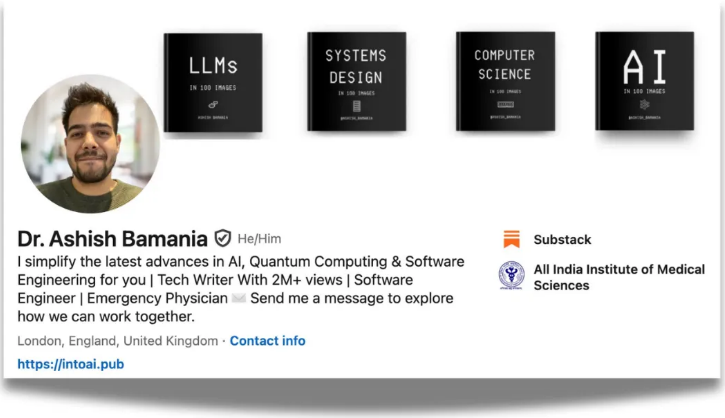
While junior developers may get by with writing functional code, senior engineers are valued for their ability to keep the bigger picture in mind.

> *How will this system handle a million users?*
>
> *What happens when a critical component fails?*
>
> *How do we maintain performance as users and their data grow exponentially?*

These are some questions that form the foundation of every 'effortlessly' running big tech app out there.

Onward.

I want to introduce Ashish Bamania as a guest author.



Dr. Ashish Bamania He/Him

I simplify the latest advances in AI, Quantum Computing & Software Engineering for you | Tech Writer With 2M+ views | Software Engineer | Emergency Physician ✉ Send me a message to explore how we can work together.

London, England, United Kingdom · Contact info

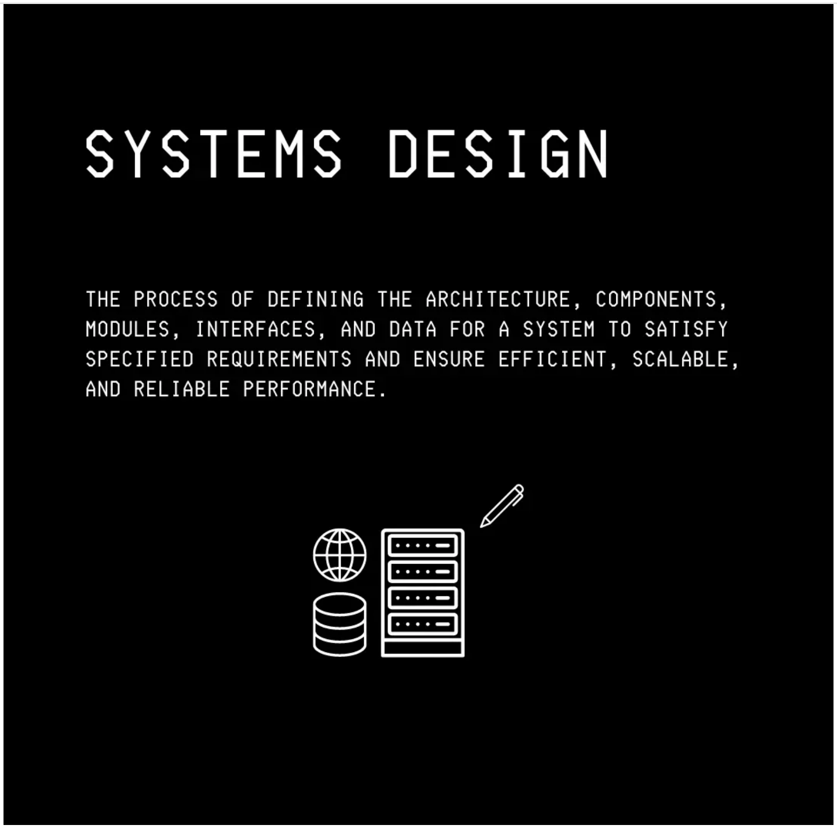
https://intoai.pub

Substack

All India Institute of Medical Sciences

He's a self-taught software engineer and works as an emergency physician.

If you're getting started with system design, I highly recommend getting his

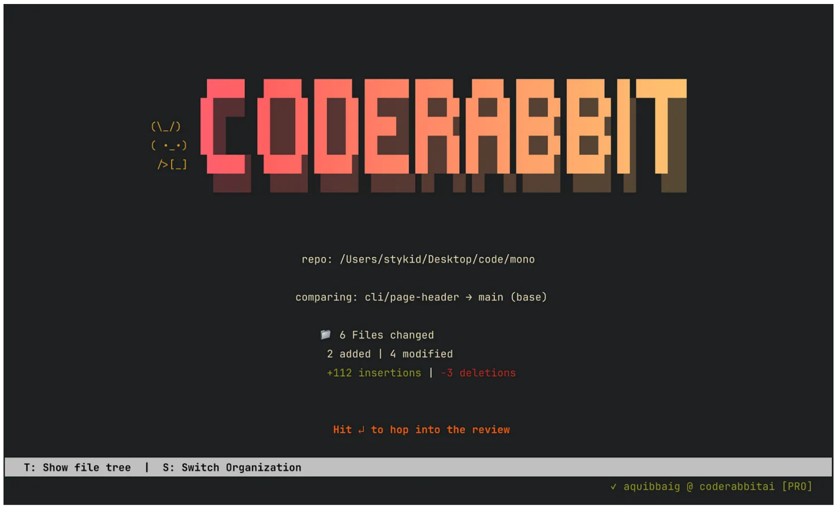


SYSTEMS DESIGN

THE PROCESS OF DEFINING THE ARCHITECTURE, COMPONENTS, MODULES, INTERFACES, AND DATA FOR A SYSTEM TO SATISFY SPECIFIED REQUIREMENTS AND ENSURE EFFICIENT, SCALABLE, AND RELIABLE PERFORMANCE.

Systems design teaches you how to answer those questions well. It is how you go from “*barely making things work*” to “*making things work at scale*”.

CodeRabbit: Free AI Code Reviews in CLI (Sponsor)



```
(\_/)
( •_•)
 />[_]   CODERABBIT

              repo: /Users/stykid/Desktop/code/mono

             comparing: cli/page-header → main (base)

                 📁 6 Files changed
                  2 added | 4 modified
                  +112 insertions | -3 deletions


                  Hit ↲ to hop into the review

 T: Show file tree  |  S: Switch Organization
                                        ✓ aquibbaig @ coderabbitai [PRO]
```

CodeRabbit CLI is an AI code review tool that runs directly in your terminal. It provides intelligent code analysis, catches issues early, and integrates seamlessly with AI coding agents like Claude Code, Codex CLI, Cursor CLI, and Gemini to ensure your code is production-ready before it ships.

- Enables pre-commit reviews of both staged and unstaged changes

changes, or entire commits based on your needs.

- Supports programming languages including JavaScript, TypeScript, Python, Java, C#, C++, Ruby, Rust, Go, PHP, and more.
- Offers free AI code reviews with rate limits so developers can experience senior-level reviews at no cost.
- Flags hallucinations, code smells, security issues, and performance problems.
- Supports guidelines for other AI generators, AST Grep rules, and path-based instructions.

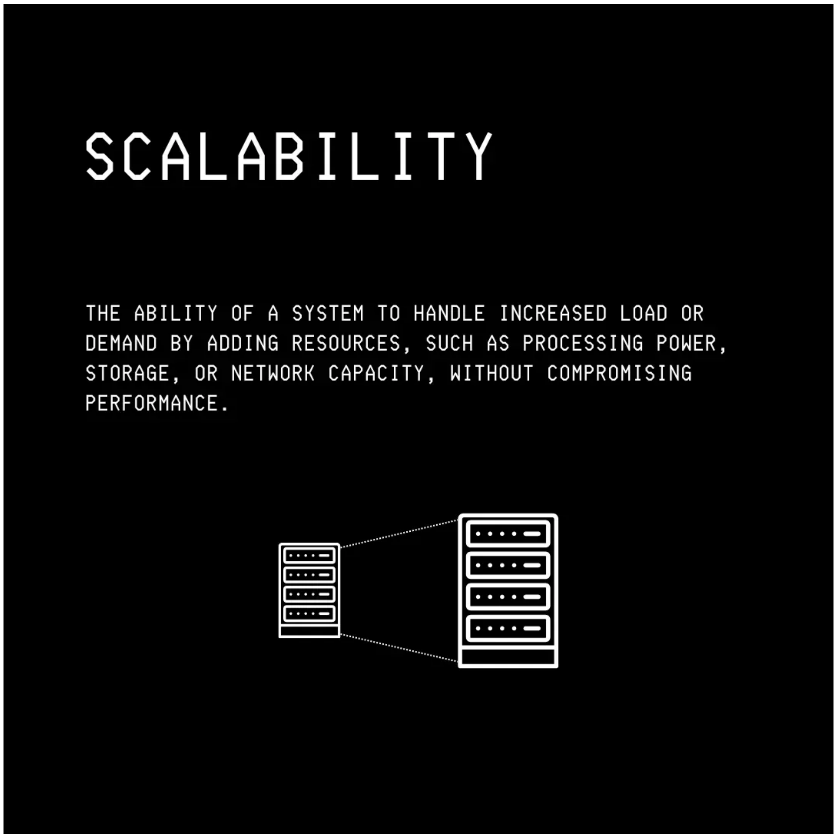
Install CodeRabbit CLI

System Design Concepts

Here are all the key concepts that you need to understand systems design and become a better engineer:

1. Scalability

Scalability is a system’s ability to handle increased load while maintaining acceptable performance.

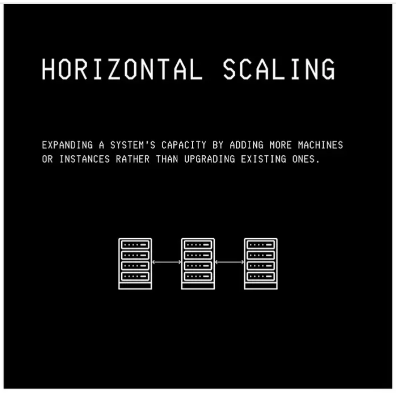


SCALABILITY

THE ABILITY OF A SYSTEM TO HANDLE INCREASED LOAD OR DEMAND BY ADDING RESOURCES, SUCH AS PROCESSING POWER, STORAGE, OR NETWORK CAPACITY, WITHOUT COMPROMISING PERFORMANCE.

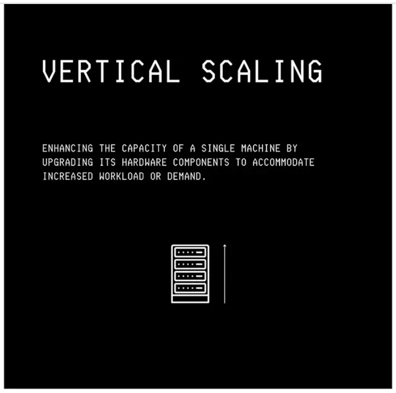
These are two ways to scale systems:

1. **Horizontal scaling**: adding more machines/ servers to handle increased load by distributing work across them.



HORIZONTAL SCALING

EXPANDING A SYSTEM'S CAPACITY BY ADDING MORE MACHINES OR INSTANCES RATHER THAN UPGRADING EXISTING ONES.



VERTICAL SCALING

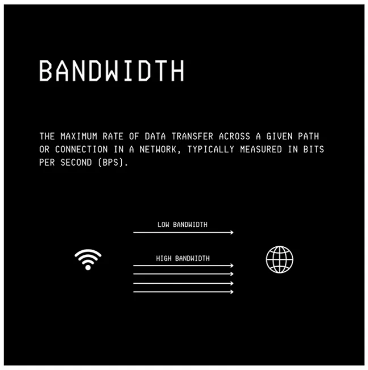
ENHANCING THE CAPACITY OF A SINGLE MACHINE BY UPGRADING ITS HARDWARE COMPONENTS TO ACCOMMODATE INCREASED WORKLOAD OR DEMAND.

2. Throughput & Bandwidth

Both terms are used interchangeably, but they have different meanings.

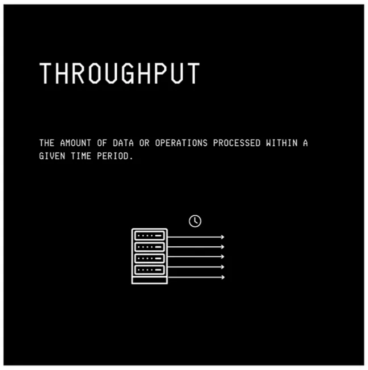
Bandwidth refers to the amount of data that can potentially travel through a network within a period.

Throughput refers to the amount of data that actually transfers during a specified period.



BANDWIDTH

THE MAXIMUM RATE OF DATA TRANSFER ACROSS A GIVEN PATH OR CONNECTION IN A NETWORK, TYPICALLY MEASURED IN BITS PER SECOND (BPS).



THROUGHPUT

THE AMOUNT OF DATA OR OPERATIONS PROCESSED WITHIN A GIVEN TIME PERIOD.

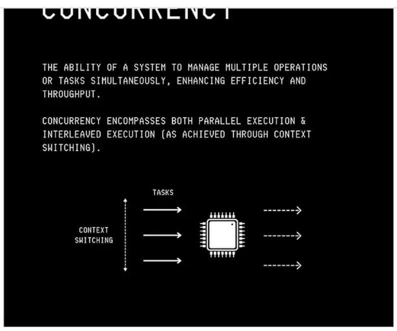
3. Concurrency vs. Parallelism

Many engineers confuse these terms, so it’s important to understand their meanings clearly.

Parallelism refers to executing multiple tasks simultaneously across different processor cores or machines.

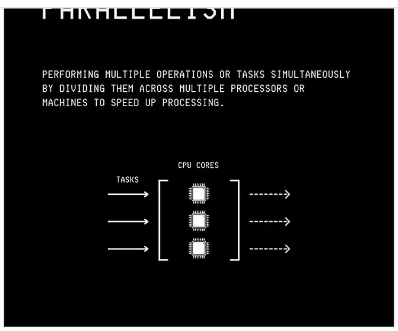
Concurrency means executing multiple tasks simultaneously, either by running them in parallel or by rapidly switching between them on the same processor core.

> *Parallelism is a subset of concurrency.*


CONCURRENCY

THE ABILITY OF A SYSTEM TO MANAGE MULTIPLE OPERATIONS OR TASKS SIMULTANEOUSLY, ENHANCING EFFICIENCY AND THROUGHPUT.

CONCURRENCY ENCOMPASSES BOTH PARALLEL EXECUTION & INTERLEAVED EXECUTION (AS ACHIEVED THROUGH CONTEXT SWITCHING).

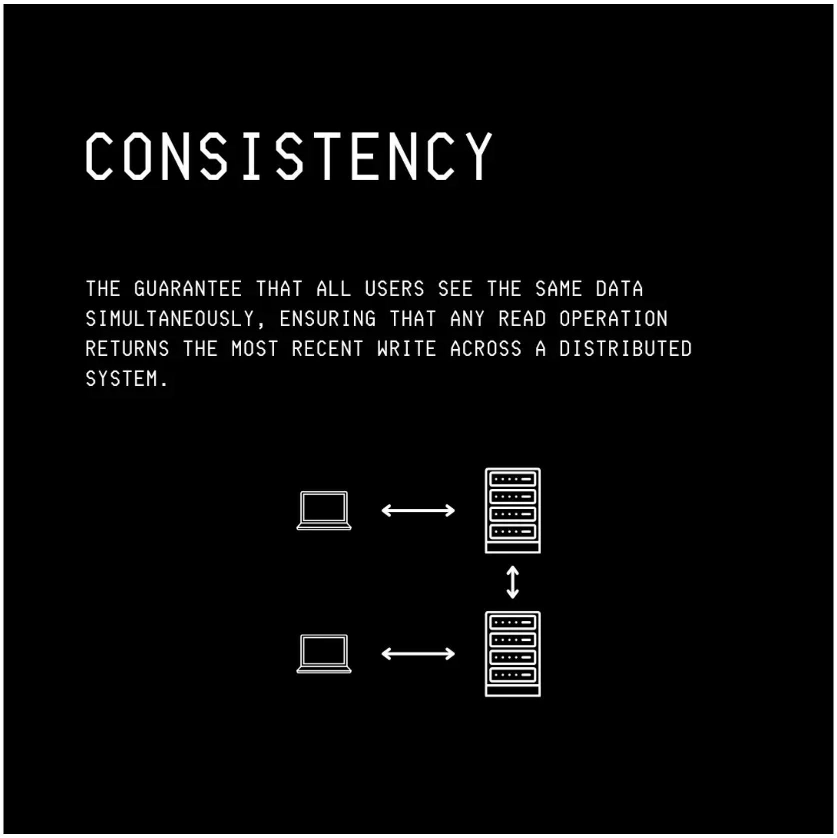

PARALLELISM

PERFORMING MULTIPLE OPERATIONS OR TASKS SIMULTANEOUSLY BY DIVIDING THEM ACROSS MULTIPLE PROCESSORS OR MACHINES TO SPEED UP PROCESSING.

4. Consistency, Availability & Partition Tolerance

These three terms are crucial for understanding the trade-offs and limitations when designing distributed systems.

- **Consistency**: The guarantee that a system of all machines connected in a distributed manner sees the same data at the same time.


CONSISTENCY

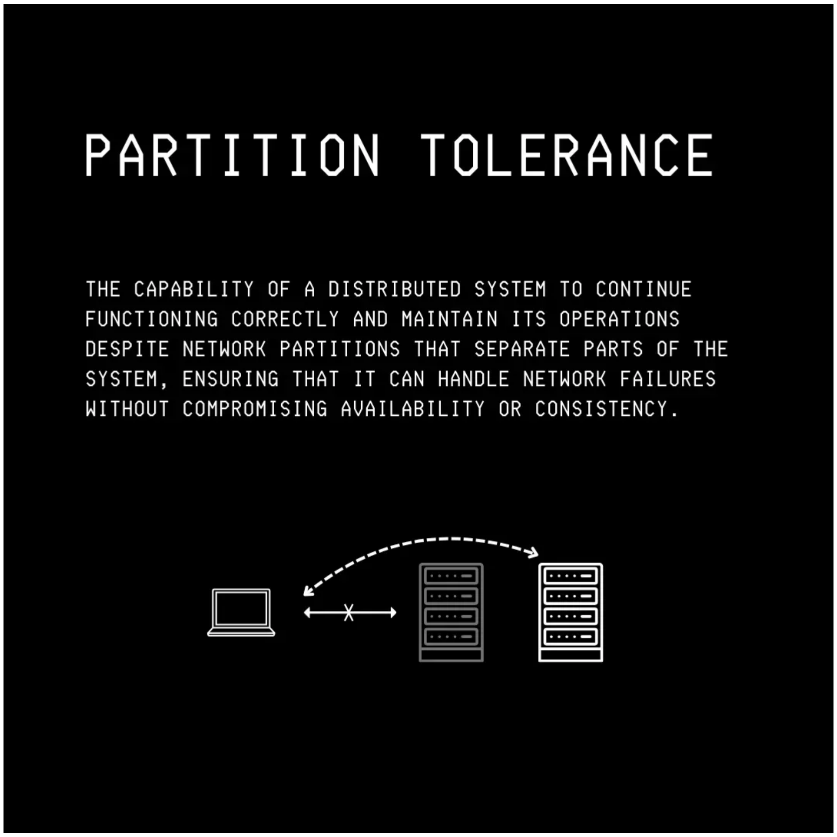
THE GUARANTEE THAT ALL USERS SEE THE SAME DATA SIMULTANEOUSLY, ENSURING THAT ANY READ OPERATION RETURNS THE MOST RECENT WRITE ACROSS A DISTRIBUTED SYSTEM.

- **Availability**: The degree to which a system remains operational and responsive to requests.

AVAILABILITY

THE DEGREE TO WHICH A SYSTEM OR COMPONENT IS OPERATIONAL AND ACCESSIBLE WHEN REQUIRED FOR USE.

- **Partition Tolerance:** The capability of a distributed system to continue operating despite network failures between the connected machines.

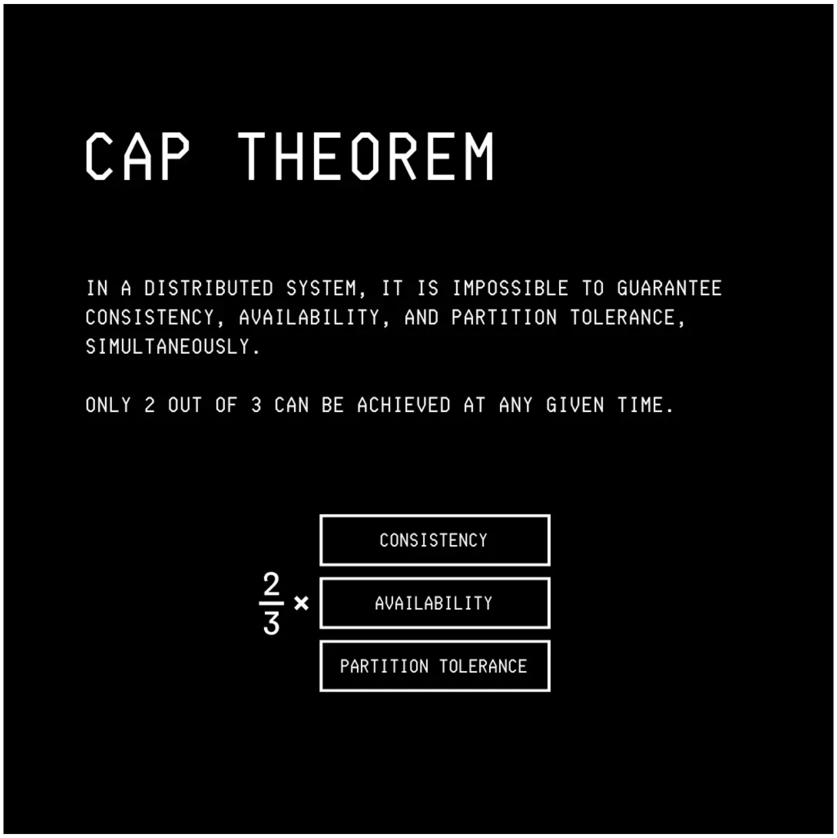


PARTITION TOLERANCE

THE CAPABILITY OF A DISTRIBUTED SYSTEM TO CONTINUE FUNCTIONING CORRECTLY AND MAINTAIN ITS OPERATIONS DESPITE NETWORK PARTITIONS THAT SEPARATE PARTS OF THE SYSTEM, ENSURING THAT IT CAN HANDLE NETWORK FAILURES WITHOUT COMPROMISING AVAILABILITY OR CONSISTENCY.

These three concepts are related to each other through the CAP theorem.

partition tolerance) but never all three simultaneously.



CAP THEOREM

IN A DISTRIBUTED SYSTEM, IT IS IMPOSSIBLE TO GUARANTEE CONSISTENCY, AVAILABILITY, AND PARTITION TOLERANCE, SIMULTANEOUSLY.

ONLY 2 OUT OF 3 CAN BE ACHIEVED AT ANY GIVEN TIME.

This leads to systems that either have:

- High Consistency and Partition Tolerance (for example, databases such as MongoDB and HBase).
- High Consistency and Availability (for example, traditional relational databases such as PostgreSQL that run on a single machine).
- High availability and Partition tolerance (for example, databases such as Cassandra and CouchDB).

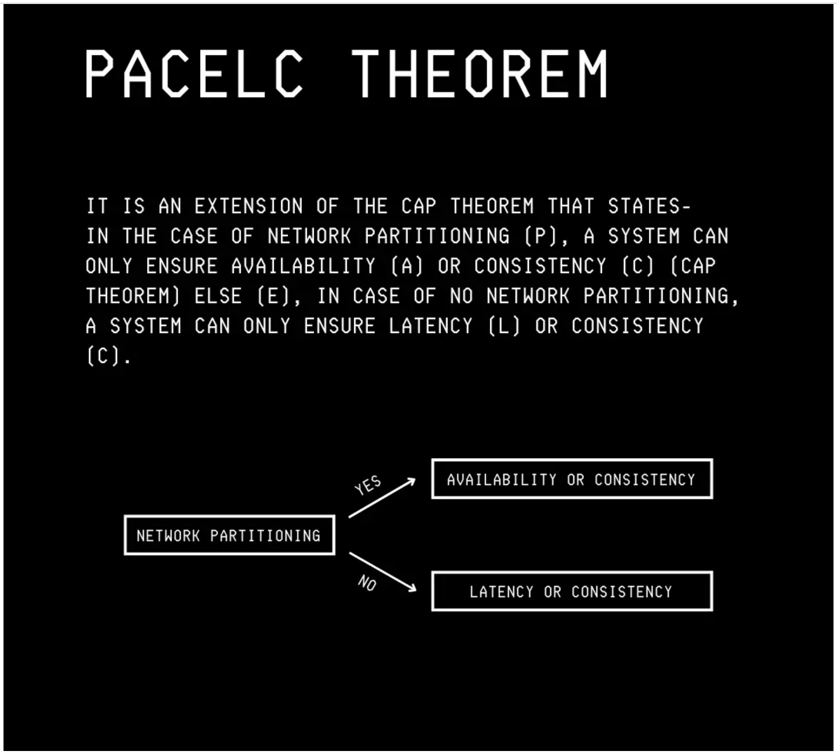
6. PACELC Theorem

The CAP theorem is further extended by the PACELC theorem.

While the CAP theorem states that with network partitioning, a distributed system must choose between availability and consistency.

The PACELC theorem adds to this by stating that, where there is no network partitioning, the distributed system still faces a trade-off between Latency and Consistency.

The acronym PACELC stands for Partition (P), Availability (A), Consistency (C), Else (E), Latency (L), Consistency (C).



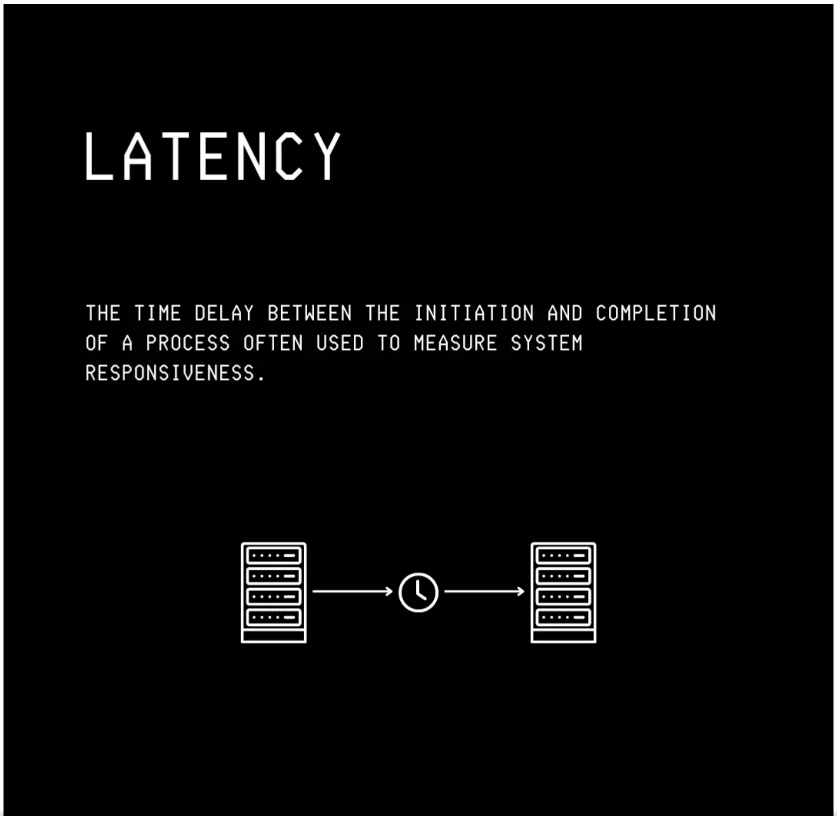
PACELC THEOREM

IT IS AN EXTENSION OF THE CAP THEOREM THAT STATES-
IN THE CASE OF NETWORK PARTITIONING (P), A SYSTEM CAN ONLY ENSURE AVAILABILITY (A) OR CONSISTENCY (C) (CAP THEOREM) ELSE (E), IN CASE OF NO NETWORK PARTITIONING, A SYSTEM CAN ONLY ENSURE LATENCY (L) OR CONSISTENCY (C).

We have not yet discussed what latency is, so let’s talk about it next.

7. Latency

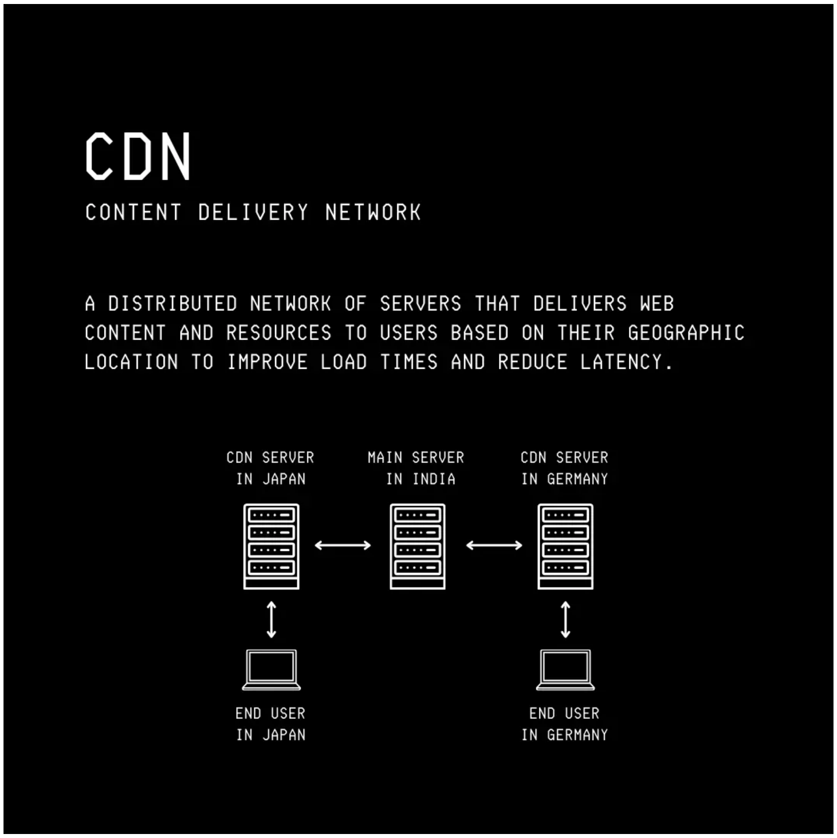
In distributed systems, Latency is the time it takes for a request to go through the system and return a response.



LATENCY

THE TIME DELAY BETWEEN THE INITIATION AND COMPLETION OF A PROCESS OFTEN USED TO MEASURE SYSTEM RESPONSIVENESS.

the users, as with Content Delivery Networks (CDNs).

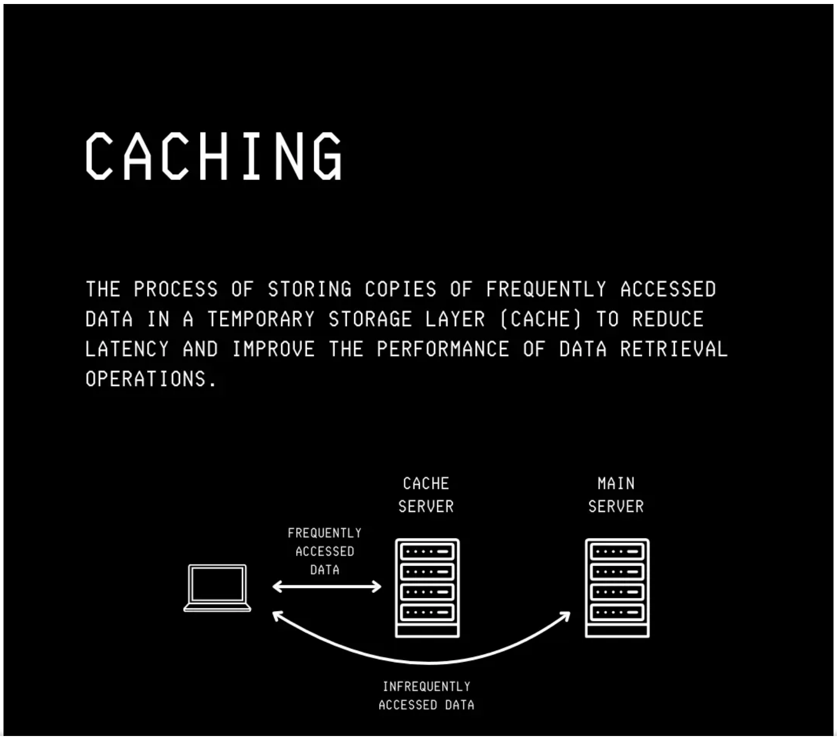


CDN

CONTENT DELIVERY NETWORK

A DISTRIBUTED NETWORK OF SERVERS THAT DELIVERS WEB CONTENT AND RESOURCES TO USERS BASED ON THEIR GEOGRAPHIC LOCATION TO IMPROVE LOAD TIMES AND REDUCE LATENCY.

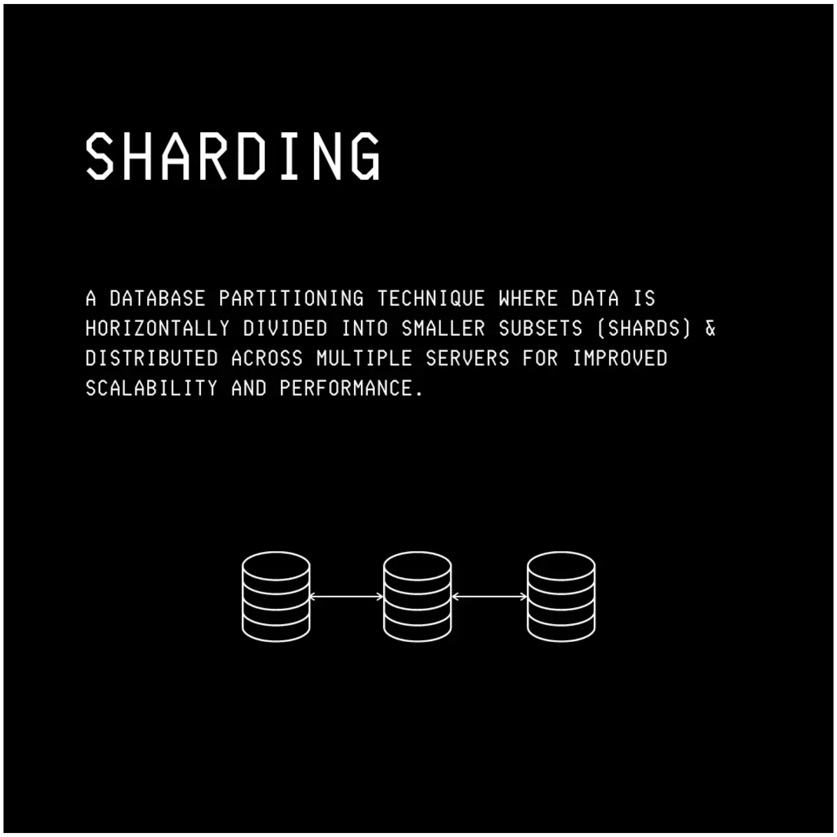
- **Caching**: a technique where frequently accessed results are stored in memory in a cache server rather than constantly querying the central database (a slow operation) for results.



CACHING

THE PROCESS OF STORING COPIES OF FREQUENTLY ACCESSED DATA IN A TEMPORARY STORAGE LAYER (CACHE) TO REDUCE LATENCY AND IMPROVE THE PERFORMANCE OF DATA RETRIEVAL OPERATIONS.

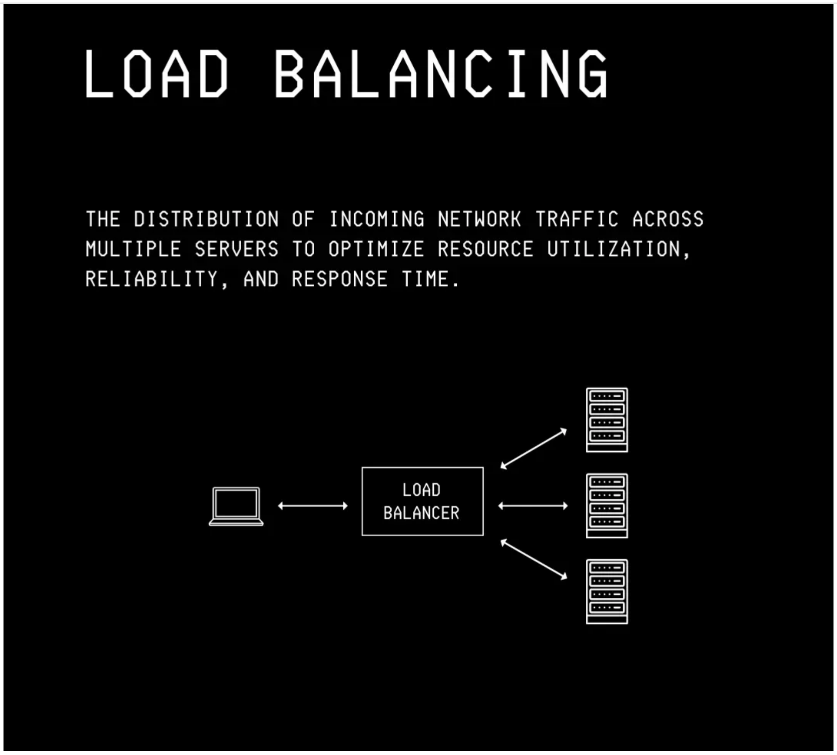
In this way, when the system is queried, each query only reaches the nodes that store the relevant shards, which process the request, reducing load and response time.



SHARDING

A DATABASE PARTITIONING TECHNIQUE WHERE DATA IS HORIZONTALLY DIVIDED INTO SMALLER SUBSETS (SHARDS) & DISTRIBUTED ACROSS MULTIPLE SERVERS FOR IMPROVED SCALABILITY AND PERFORMANCE.

- **Load balancing:** a technique of distributing incoming network requests across multiple servers to prevent any single server from becoming overloaded.



LOAD BALANCING

THE DISTRIBUTION OF INCOMING NETWORK TRAFFIC ACROSS MULTIPLE SERVERS TO OPTIMIZE RESOURCE UTILIZATION, RELIABILITY, AND RESPONSE TIME.

There are multiple algorithms for load balancing, and some popular ones are:

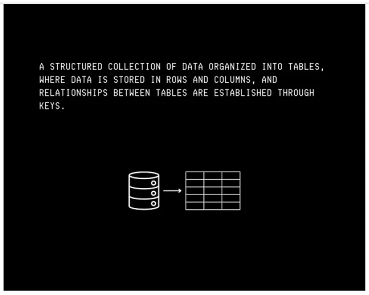
- **Round Robin:** Sending requests to servers in a fixed cyclical order.
- **Least Connections:** Sending requests to the server with the fewest active connections.
- **Least Response Time:** Sending requests to the server with the fastest response time.
- **Weighted Round Robin**: Sending requests to servers in a fixed cyclical order, but proportional to each server's capacity.

Next, let’s learn about databases.

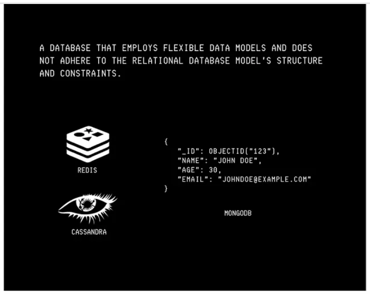
9. Relational vs. Non-Relational Databases

The two main types of databases are:

- **Relational/ SQL databases:** These databases organize data into tables with predefined schemas and use SQL (Structured Query Language) to query them. For example, PostgreSQL and MySQL.
- **Non-relational/NoSQL databases:** These databases do not rely on fixed table schemas and store and query data in flexible ways. For example, MongoDB, Redis, and Cassandra.



A STRUCTURED COLLECTION OF DATA ORGANIZED INTO TABLES, WHERE DATA IS STORED IN ROWS AND COLUMNS, AND RELATIONSHIPS BETWEEN TABLES ARE ESTABLISHED THROUGH KEYS.



A DATABASE THAT EMPLOYS FLEXIBLE DATA MODELS AND DOES NOT ADHERE TO THE RELATIONAL DATABASE MODEL'S STRUCTURE AND CONSTRAINTS.

```
{
  "_ID": OBJECTID("123"),
  "NAME": "JOHN DOE",
  "AGE": 30,
  "EMAIL": "JOHNDOE@EXAMPLE.COM"
}
```

10. Transactions & Their Types

The next concept that you must know about is a Transaction, which is a logical unit of work that groups one or multiple operations.

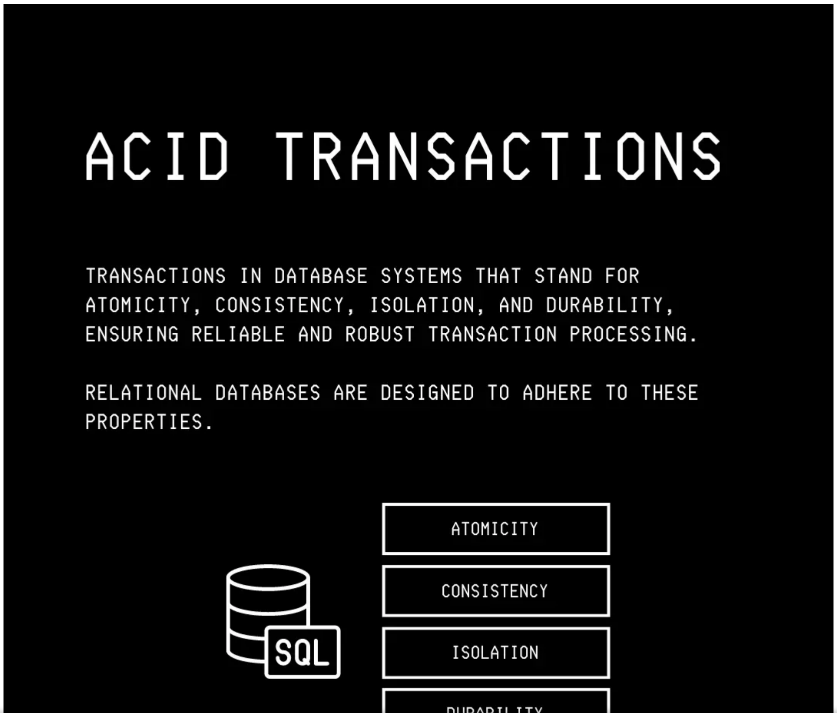
The two types of transactions in distributed systems are:

- ACID transactions
- BASE transactions

ACID Transactions

The acronym ACID stands for:

- **Atomicity**: A transaction either completes entirely or fails, and no partial transactions occur.
- **Consistency**: A system remains in a valid state before and after each transaction, as per all rules and constraints.
- **Isolation**: Transactions running in parallel do so as if they're the only ones executing. No two transactions interfere with each other's intermediate states.
- **Durability**: Once a transaction is committed, its changes are permanently saved and are unaffected by system failures.

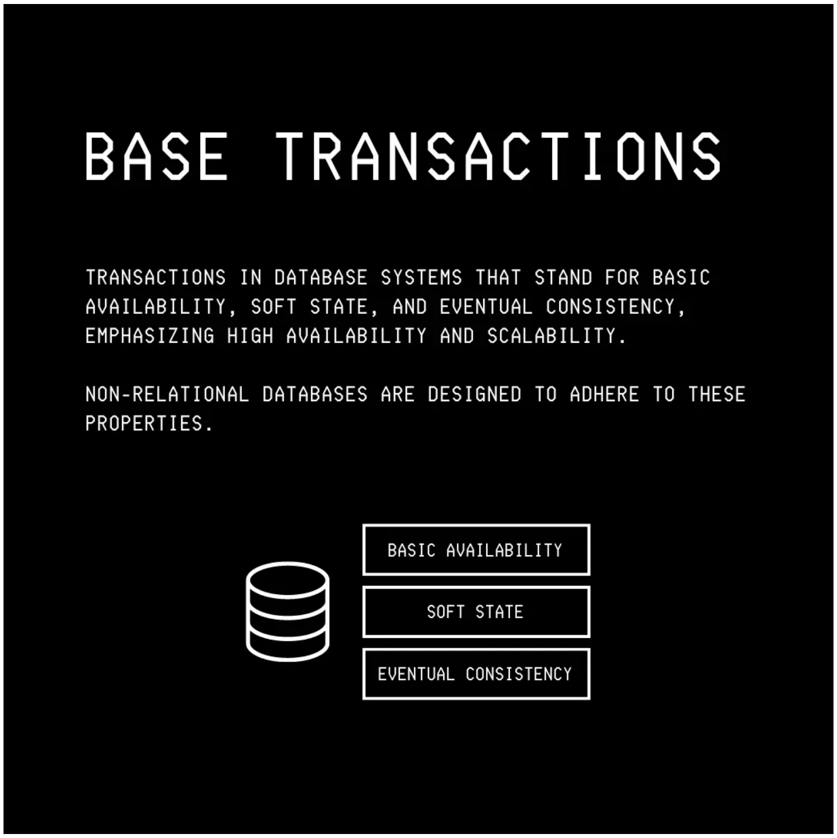


ACID TRANSACTIONS

TRANSACTIONS IN DATABASE SYSTEMS THAT STAND FOR ATOMICITY, CONSISTENCY, ISOLATION, AND DURABILITY, ENSURING RELIABLE AND ROBUST TRANSACTION PROCESSING.

RELATIONAL DATABASES ARE DESIGNED TO ADHERE TO THESE PROPERTIES.

- **Basically Available**: The system remains operational even when parts of it fail, although some data may be temporarily inaccessible.
- **Soft State**: Data can be temporarily inconsistent across different parts of the system. These temporary inconsistencies are resolved at a later time.
- **Eventual Consistency**: Given sufficient time, all parts of the system synchronize and eventually converge to a consistent state.



BASE TRANSACTIONS

TRANSACTIONS IN DATABASE SYSTEMS THAT STAND FOR BASIC AVAILABILITY, SOFT STATE, AND EVENTUAL CONSISTENCY, EMPHASIZING HIGH AVAILABILITY AND SCALABILITY.

NON-RELATIONAL DATABASES ARE DESIGNED TO ADHERE TO THESE PROPERTIES.

Relational/ SQL databases use ACID transactions to maintain strict data integrity and relationships.

NoSQL databases use BASE transactions to achieve high scalability and performance by relaxing consistency requirements.

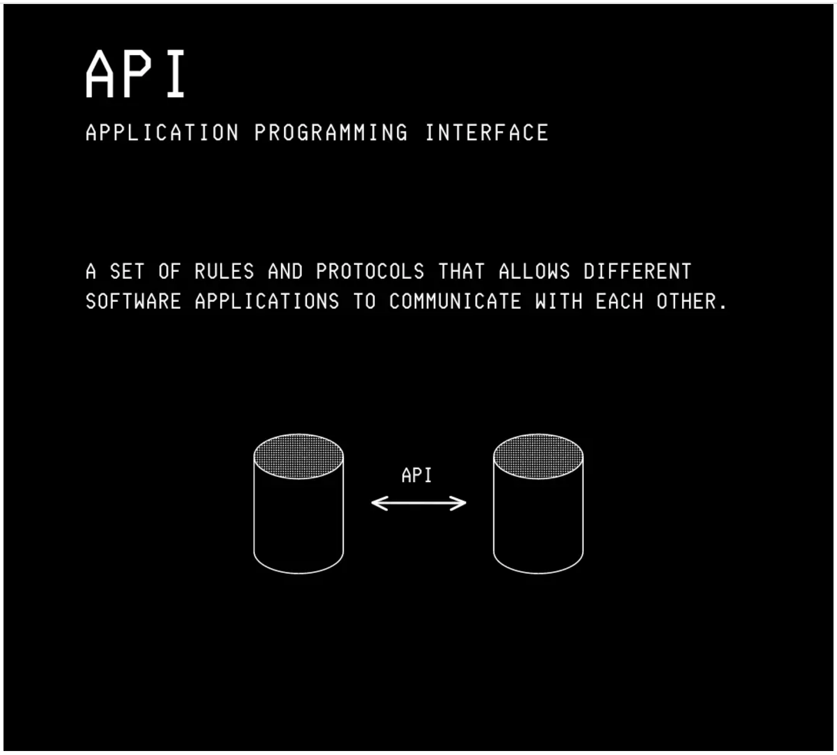
BASE transactions help large-scale applications, such as those from Amazon, Google, and Meta, work around the CAP/PACELC trade-offs by prioritizing availability and low latency over strict consistency.

11. APIs

The last important concept that we discuss in this article is Application Programming Interfaces (APIs).

These are sets of rules that allow different applications to communicate with each other.

Think of an API as a waiter in a restaurant. A client requests the waiter. The waiter (the API) takes the order to the kitchen (the Server), and gets back with the food for the client (the Response).



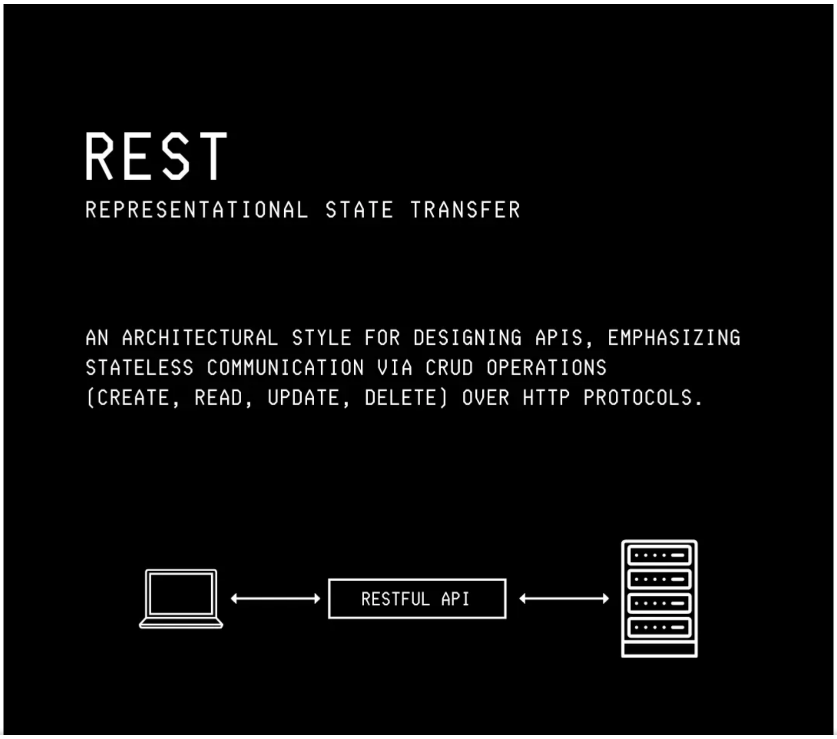
API

APPLICATION PROGRAMMING INTERFACE

A SET OF RULES AND PROTOCOLS THAT ALLOWS DIFFERENT SOFTWARE APPLICATIONS TO COMMUNICATE WITH EACH OTHER.

Three types of Web API architectural styles are popular and important to know about:

- **REST (Representational State Transfer):** This is the most common architectural style for designing web APIs. REST APIs use standard HTTP methods (`GET`, `POST`, `PUT`, `DELETE`), are stateless (so each request contains all the information for the server to process it), and usually return data in JSON format.

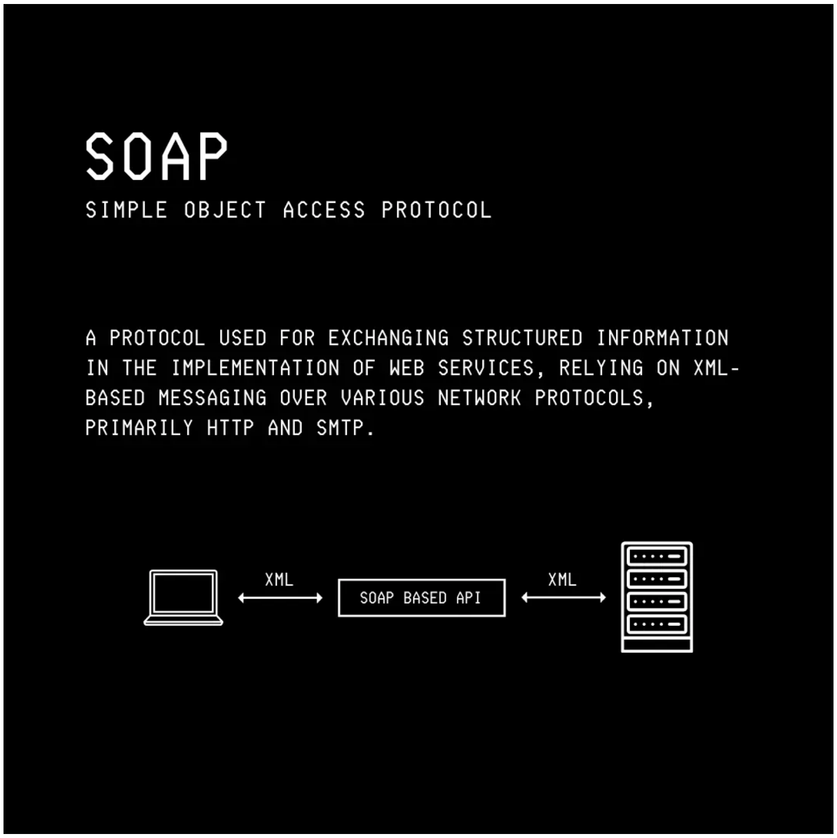


REST

REPRESENTATIONAL STATE TRANSFER

AN ARCHITECTURAL STYLE FOR DESIGNING APIS, EMPHASIZING STATELESS COMMUNICATION VIA CRUD OPERATIONS (CREATE, READ, UPDATE, DELETE) OVER HTTP PROTOCOLS.

enterprise environments because of its built-in error handling and security features.

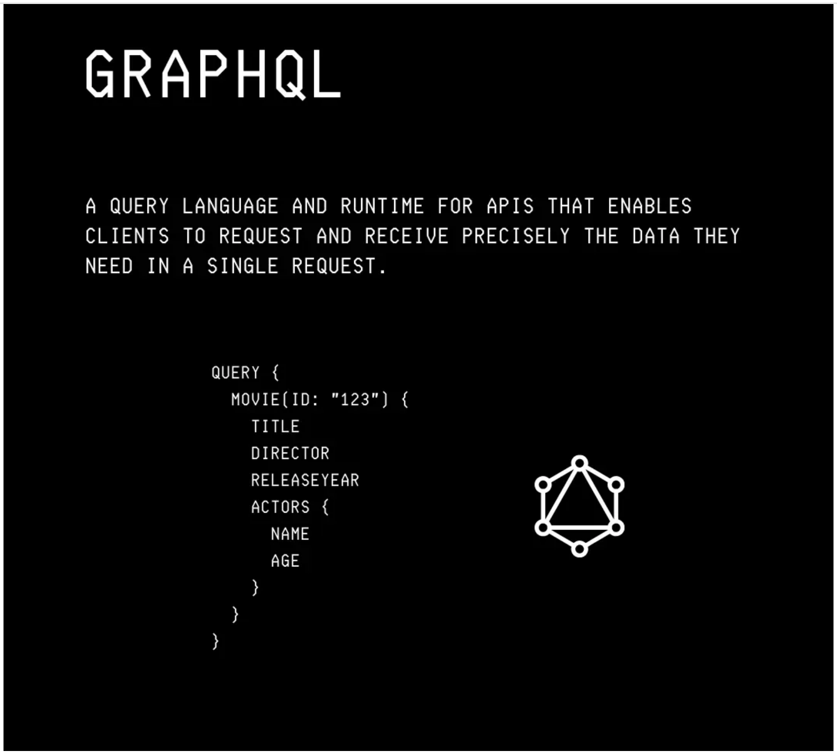


SOAP

SIMPLE OBJECT ACCESS PROTOCOL

A PROTOCOL USED FOR EXCHANGING STRUCTURED INFORMATION IN THE IMPLEMENTATION OF WEB SERVICES, RELYING ON XML-BASED MESSAGING OVER VARIOUS NETWORK PROTOCOLS, PRIMARILY HTTP AND SMTP.

- **GraphQL**: This is a newer API architectural style developed by Facebook (now Meta) that allows clients to request the exact data they need. Unlike REST, where multiple endpoints may be required, GraphQL uses a single endpoint to query for specific data requirements.



GRAPHQL

A QUERY LANGUAGE AND RUNTIME FOR APIS THAT ENABLES CLIENTS TO REQUEST AND RECEIVE PRECISELY THE DATA THEY NEED IN A SINGLE REQUEST.

```
QUERY {
  MOVIE(ID: "123") {
    TITLE
    DIRECTOR
    RELEASEYEAR
    ACTORS {
      NAME
      AGE
    }
  }
}
```

Now that you understand the basic concepts that are useful in designing systems that work at scale, it’s time to explore how they are applied.

Look at the apps you use every day and think about how they might work behind the scenes.

How does Substack handle millions of blogs?

How does WhatsApp deliver messages instantly worldwide?

How does your bank app handle millions of users without crashing?

Why doesn’t YouTube buffer when millions watch videos all at once?

After understanding the design principles behind these existing systems, start applying these concepts in your own work.

Practice redesigning small projects you’ve built before. Talk to other engineers to get their perspectives on how they would handle a problem.

Remember, every expert was once a beginner. I am sure that you will design systems that handle millions of users before you know it.

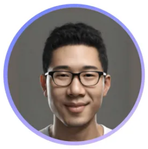
👋 I'd like to thank Ashish for writing this newsletter!

- Also try his newsletters: **Into AI** and **Into Quantum**.

Don't forget to **get his book** at a special 25% discount. It’ll help you understand system design fundamentals quickly.

Unlock access to every deep dive article by becoming a paid subscriber:

Type your email... | Subscribe



👋 Say hi on **LinkedIn** | **Twitter** | **Threads** | **Instagram**

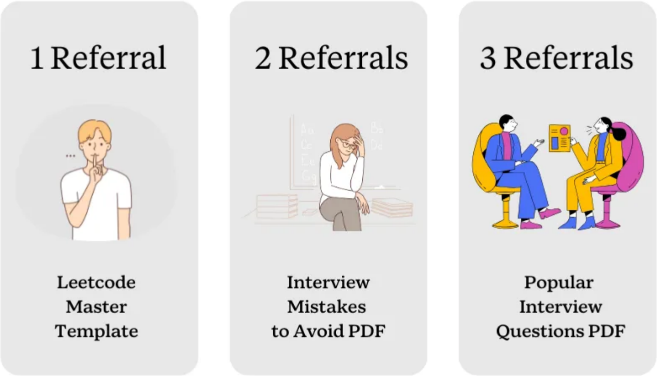
Want to advertise in this newsletter? 📰

If your company wants to reach a 170K+ tech audience, advertise with me.

Thank you for supporting this newsletter.

You are now 171,001+ readers strong, very close to 172k. Let’s try to get 172k readers by 20 September. Consider sharing this post with your friends and get rewards.

Y’all are the best.


1 Referral
Leetcode Master Template
2 Referrals
Interview Mistakes to Avoid PDF
3 Referrals
Popular Interview Questions PDF

Share

← Previous

Next →

A guest post by

Dr. Ashish Bamania

🍰 I simplify the latest advances in AI, Quantum Computing & Software Engineering for you | Software Engineer | Emergency Physician

Subscribe to Dr.

Discussion about this post

Comments Restacks

Write a comment...

Ellie Daw Sep 16

Liked by Neo Kim, Dr. Ashish Bamania

This was the perfect amount of detail - thank you!

LIKE (3) REPLY SHARE

2 replies by Neo Kim and others

Ishwarya Rajendrababu 2d

Liked by Neo Kim, Dr. Ashish Bamania

where do you see consistent hashing which provides fault tolerance in a distributed system fall into the above categories?

LIKE (2) REPLY SHARE

1 reply

7 more comments...

Top Latest Discussions

8 Reasons Why WhatsApp Was Able to Support 50 Billion Messages a Day With Only 32 Engineers

#1: Learn More - Awesome WhatsApp Engineering (6 minutes)

AUG 27, 2023 • NEO KIM

How PayPal Was Able to Support a Billion Transactions per Day With Only 8 Virtual Machines

#30: Learn More - Awesome PayPal Engineering (4 minutes)

DEC 26, 2023 • NEO KIM

How Stripe Prevents Double Payment Using Idempotent API

#45: A Simple Introduction to Idempotent API (4 minutes)

MAY 9, 2024 • NEO KIM