The 20 Concepts That Turned My System Desig Nightmare into Success

Why 90% of Engineers Bomb System Design Interviews (20 Concepts That Save You)

JAVINPAUL AND SOMA
AUG 31, 2025

Preparing for system design interviews can feel like climbing a mountain without a map. Unlike **coding interviews** where you can gain confidence by practicing data structures and algorithms on platforms like AlgoMonster, Exponent and LeetCode, system design questions demand a mix of breadth and depth — architecture principles, scalability patterns, trade-offs, and real-world application.

For me, this part of the interview loop was intimidating at first. I often felt lost in diagrams, unsure which concept to use where, and overwhelmed by the sheer vastn of distributed systems.

The turning point came when I started breaking the subject down into core concepts. Once I understood ideas like load balancing, caching, database sharding, CAP theorem, and message queues, everything else started to click into place.

Instead of memorizing solutions, I began recognizing patterns. That's when I realiz system design isn't about giving a "perfect" architecture, but about **reasoning throu trade-offs with clarity.**

What really accelerated my learning was leveraging structured resources. Books an visual explanations like **ByteByteGo's System Design Course** made the hardest concepts digestible with diagrams and case studies.

I also explored platforms such as Codemia.io and Bugfree.ai for hands-on interview prep and Exponent for mock interviews with engineers from top companies. Each helped me move from feeling clueless to confident, especially when facing open-en system design questions at FAANG-level interviews.

In this article, I'll share the **20 core concepts that completely changed how I appro system design interviews**. Mastering these will save you from confusion, help you build better mental models, and make those tough whiteboard sessions a lot less sc

Stop Failing System Design Interviews: Master These 20 Core Concepts First

Here are the 20 key concepts I learned and master by going through different Syste Design resources. Once you learn these concepts, half the battle is already one.

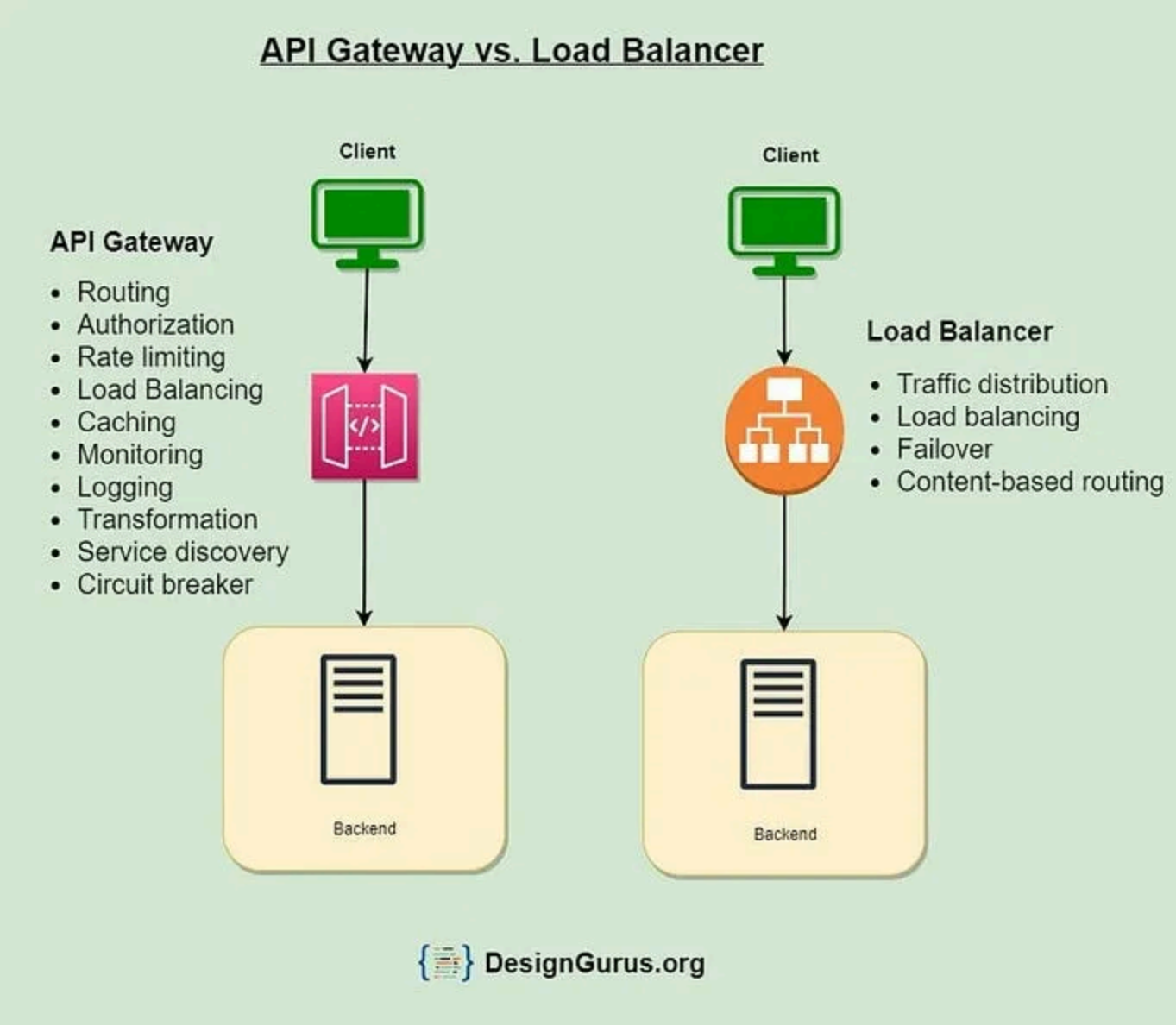
1. Load Balancing: The Traffic Director

Think of load balancers as smart traffic directors for your application. They distribu incoming requests across multiple servers to prevent any single server from becomi overwhelmed.

Key insight: There are different types — Layer 4 (transport layer) and Layer 7 (application layer). Layer 7 load balancers can make routing decisions based on content, while Layer 4 focuses on IP and port information.

Real-world example: When you visit Amazon, a load balancer decides which of the thousands of servers will handle your request.

Here is a nice diagram from designgurus.io which explains the load balancer conce along with API gateway which we will see in a couple of seconds.

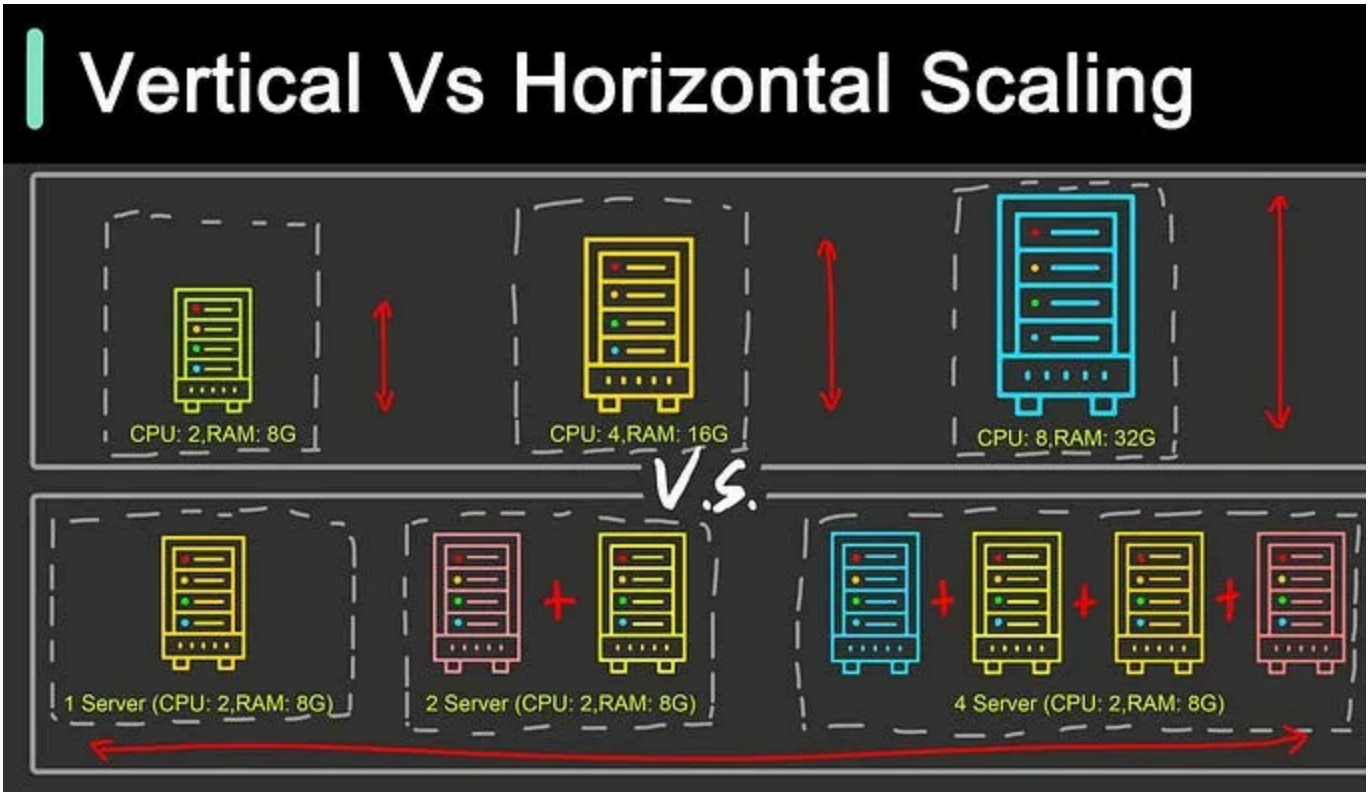


2. Horizontal vs Vertical Scaling: The Growth Strategies

- **Vertical Scaling (Scale Up):** Adding more power to existing machines
- **Horizontal Scaling (Scale Out):** Adding more machines to the pool

Game-changer moment: Understanding that horizontal scaling is almost always preferred for large systems because it's more cost-effective and provides better faul tolerance.

Here is a visual guide from ByteByteGo which makes this concept crystal clear



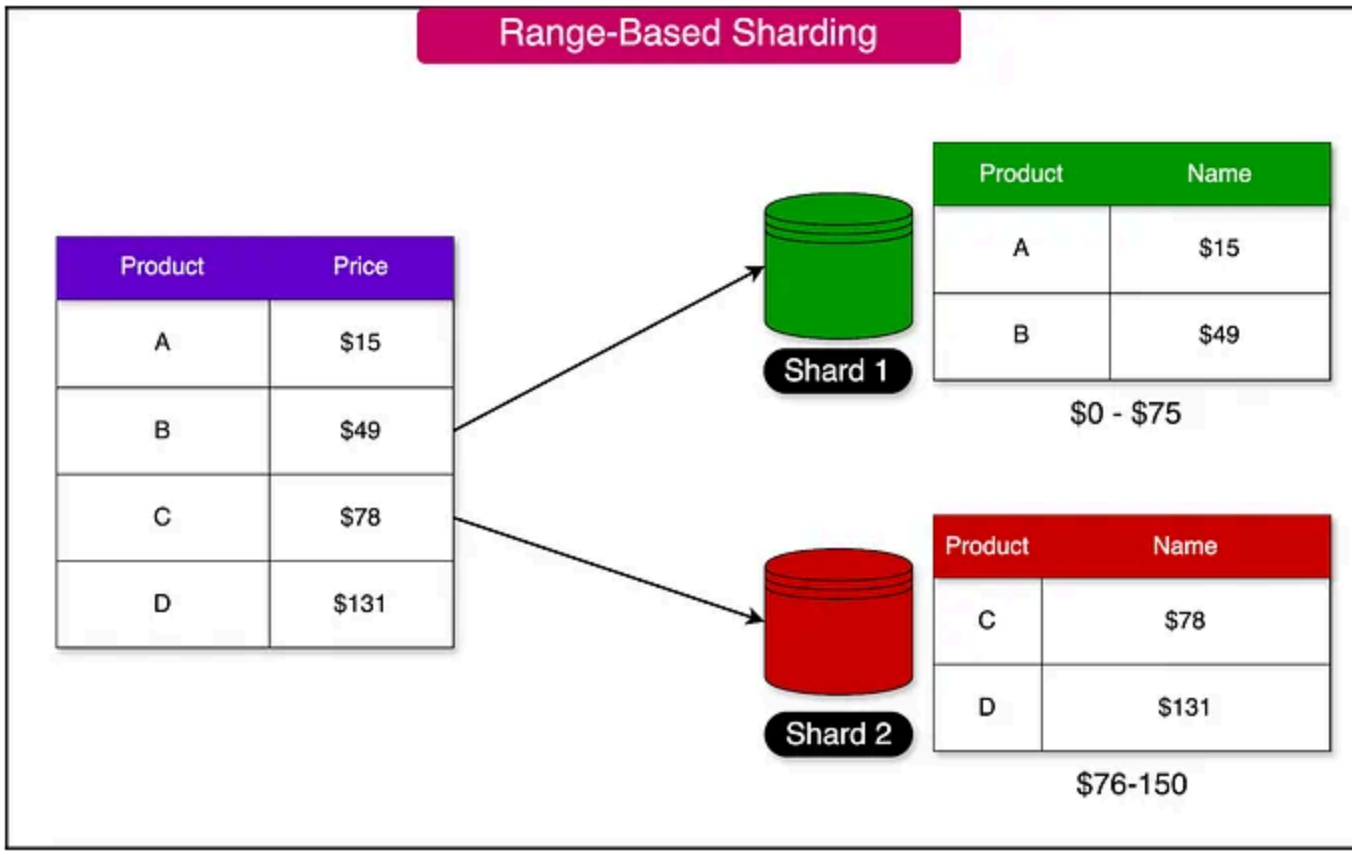
3. Database Sharding: Divide and Conquer

Sharding splits your database across multiple machines. Each shard contains a subs of your data.

The breakthrough: Learning about sharding keys and how poor sharding strategies can create hotspots that defeat the entire purpose.

Example: Instagram shards user data based on user ID, ensuring even distribution across databases.

Here is another great visual from ByteByteGo which explains Range based sharding



4. Caching Strategies: The Speed Multiplier

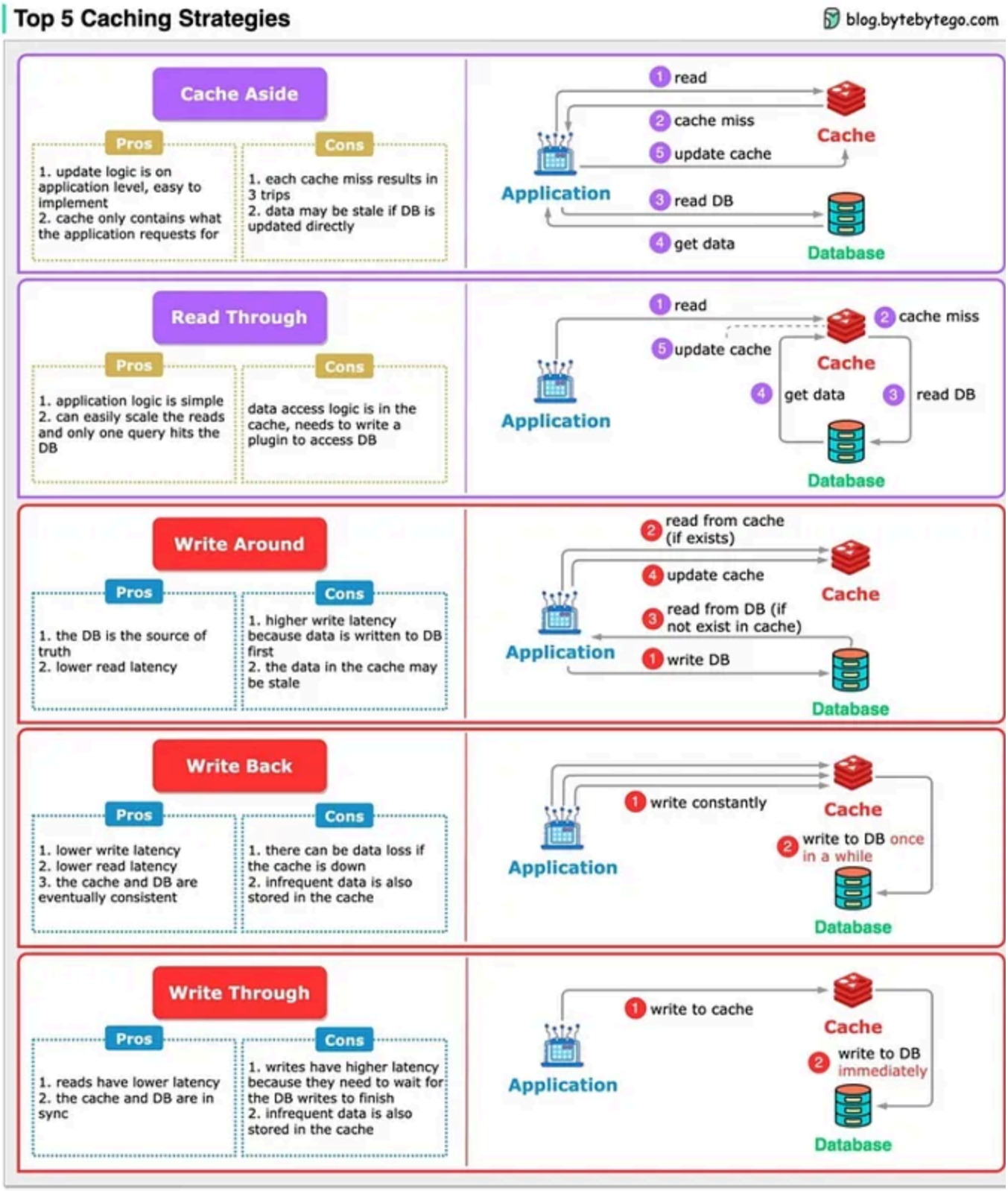
Caching is storing frequently accessed data in fast storage. The key is understandin different caching patterns:

- **Cache-aside (Lazy Loading):** Application manages cache
- **Write-through:** Write to cache and database simultaneously
- **Write-behind:** Write to cache immediately, database later

Pro tip: The cache invalidation problem is one of the hardest problems in computer science. Master cache eviction policies (LRU, LFU, FIFO).

A picture is worth thousand words and this visual from ByteByteGo proves that, int nicely explains all the caching strategies a senior developer should be aware of.

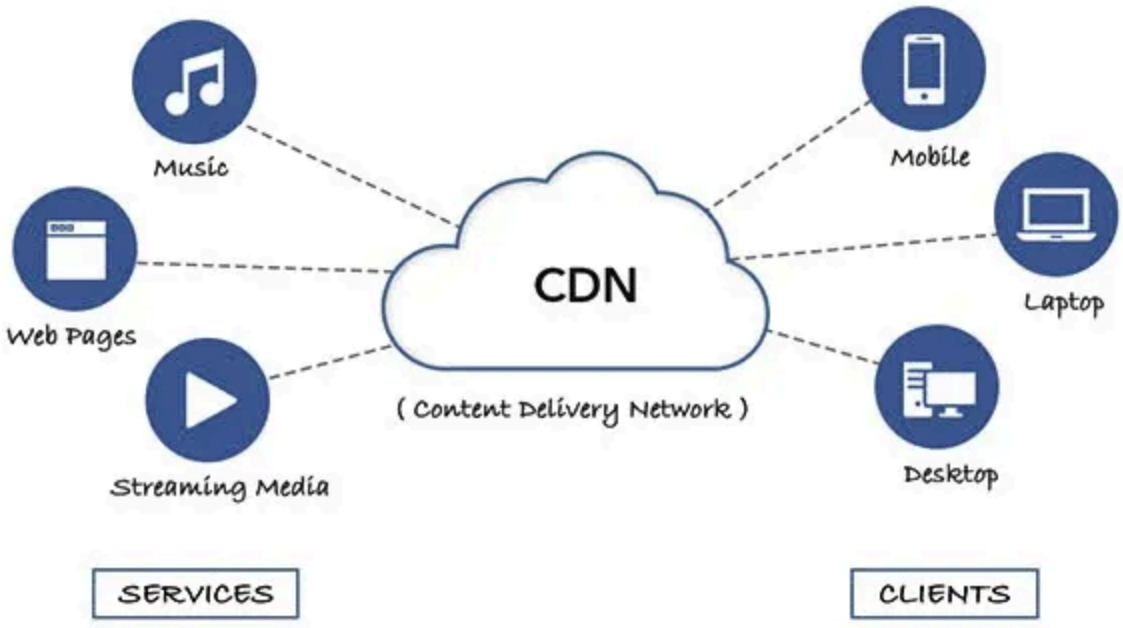
If you like visual learning, I highly recommend you to join ByteByteGo now as they offering **50% discount on their lifetime plan**. I have taken that as its just 2.5 times annual plan but provides most value.



5. Content Delivery Networks (CDN): Global Speed

CDNs cache your content at edge locations worldwide, reducing latency for users.

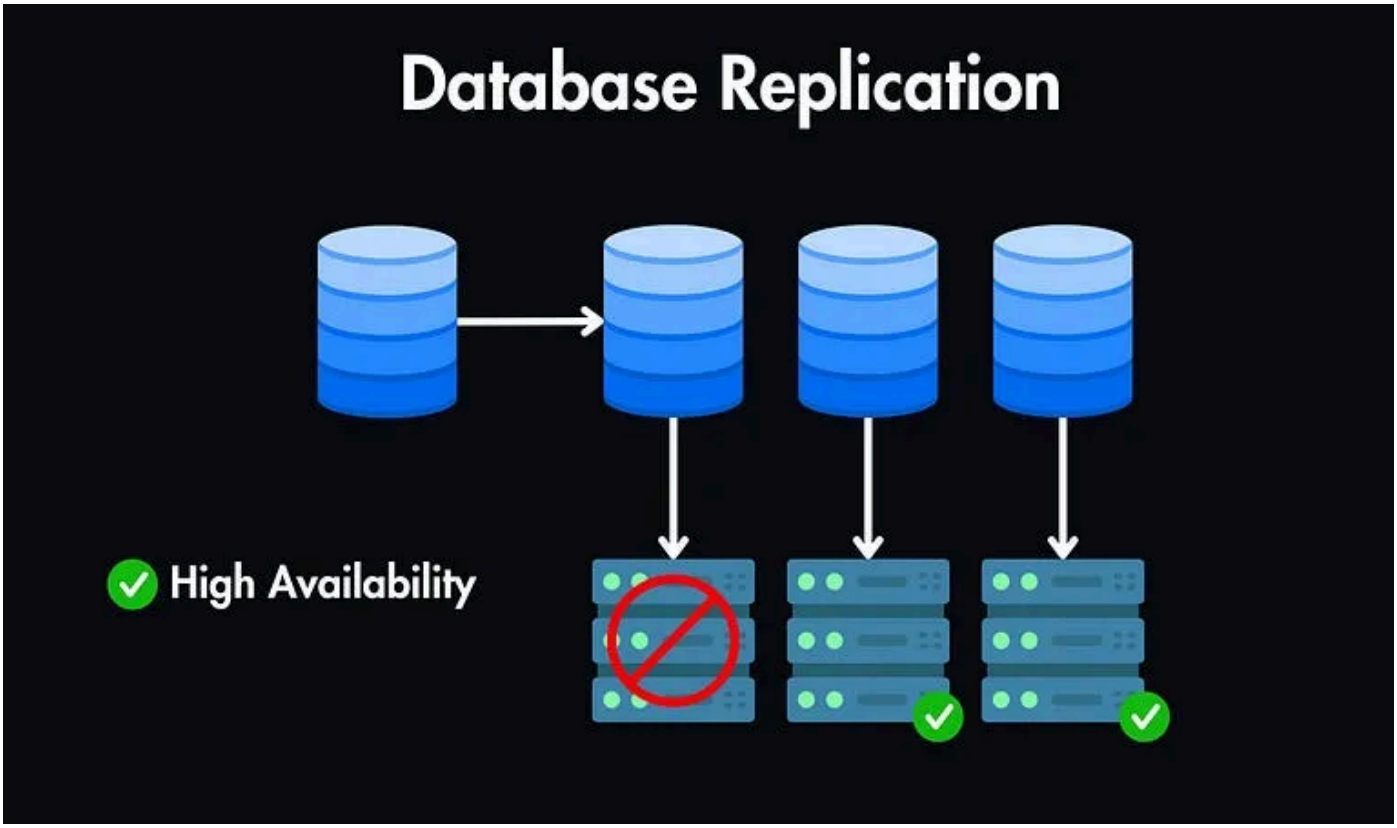
Aha moment: Realizing that CDNs don't just cache static content — modern CDNs can cache dynamic content and even run serverless functions at the edge.



6. Database Replication: The Backup Plan

- **Master-Slave:** One write node, multiple read nodes
- **Master-Master:** Multiple write nodes (more complex)

Critical insight: Understanding eventual consistency and how replication lag can affect your application logic.



7. Consistent Hashing: The Elegant Solution

Regular hashing breaks when you add/remove servers. Consistent hashing minimiz redistribution when the hash table is resized.

Why it matters: This is how systems like DynamoDB and Cassandra distribute data across nodes efficiently.

8. CAP Theorem: The Fundamental Trade-off

You can only guarantee two out of three:

- **Consistency:** All nodes see the same data simultaneously
- **Availability:** System remains operational
- **Partition Tolerance:** System continues despite network failures

Real impact: This guides every distributed system design decision you'll ever make.

9. Event-Driven Architecture: The Modern Approach

Systems communicate through events rather than direct calls. This creates loose coupling and better scalability.

Game-changer: Understanding that event sourcing can make your system audit-friendly and enable powerful debugging capabilities.

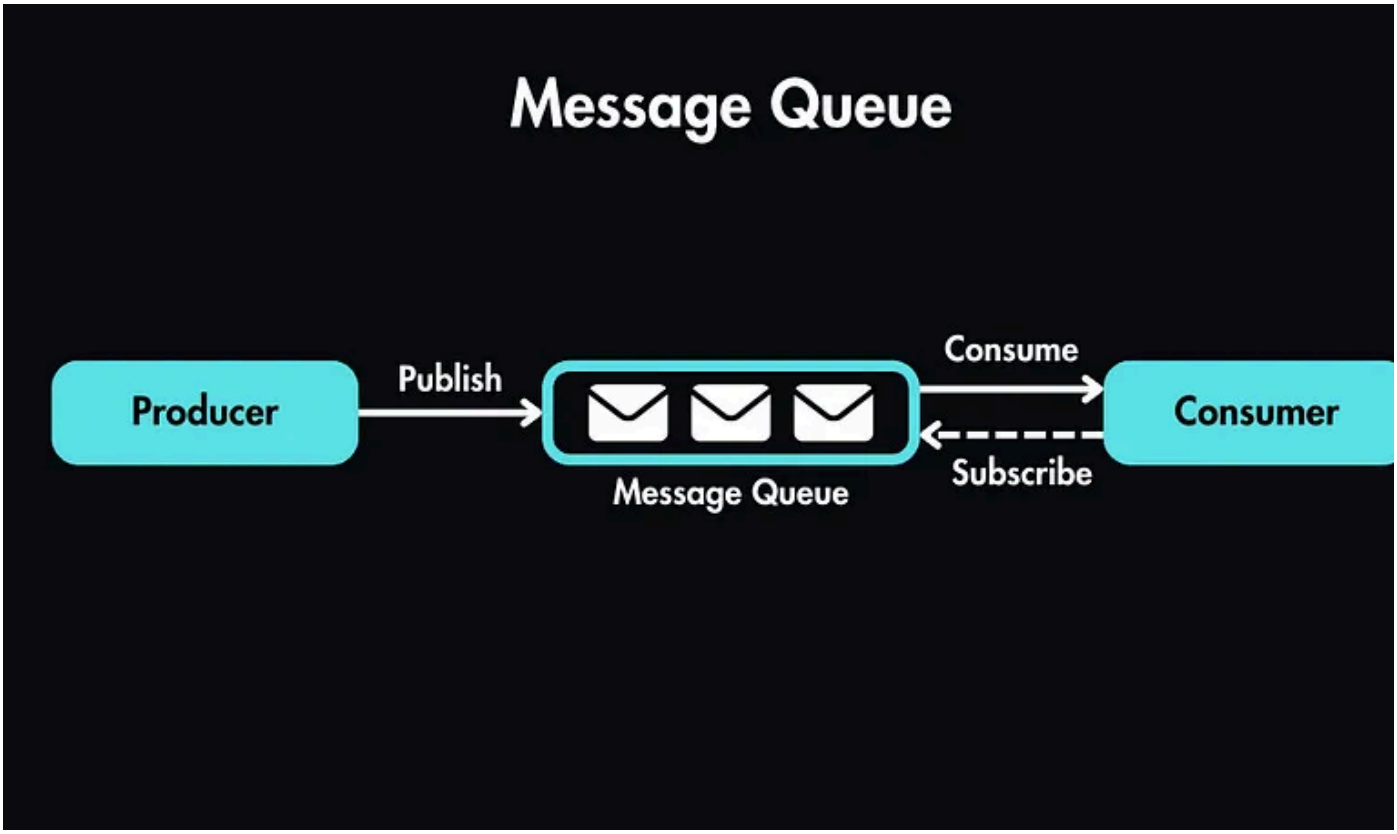
10. Message Queues: Asynchronous Communication

Queues decouple producers and consumers, enabling asynchronous processing.

Key patterns:

- Point-to-point (one consumer)
- Publish-subscribe (multiple consumers)

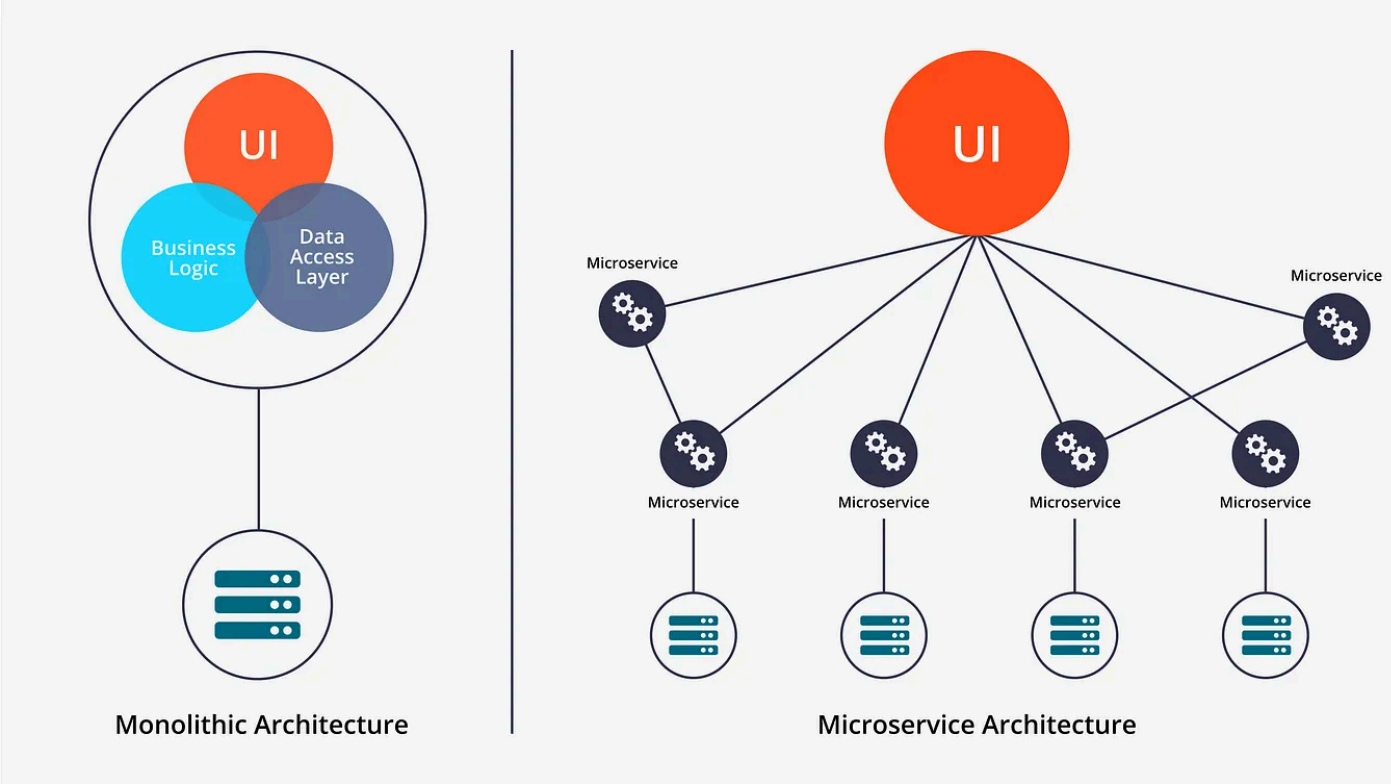
Example: When you upload a video to YouTube, it goes into a queue for processing rather than blocking your upload.



11. Microservices vs Monolith: The Architecture Debate

Monolith advantages: Simpler deployment, testing, debugging **Microservices advantages:** Independent scaling, technology diversity, fault isolation

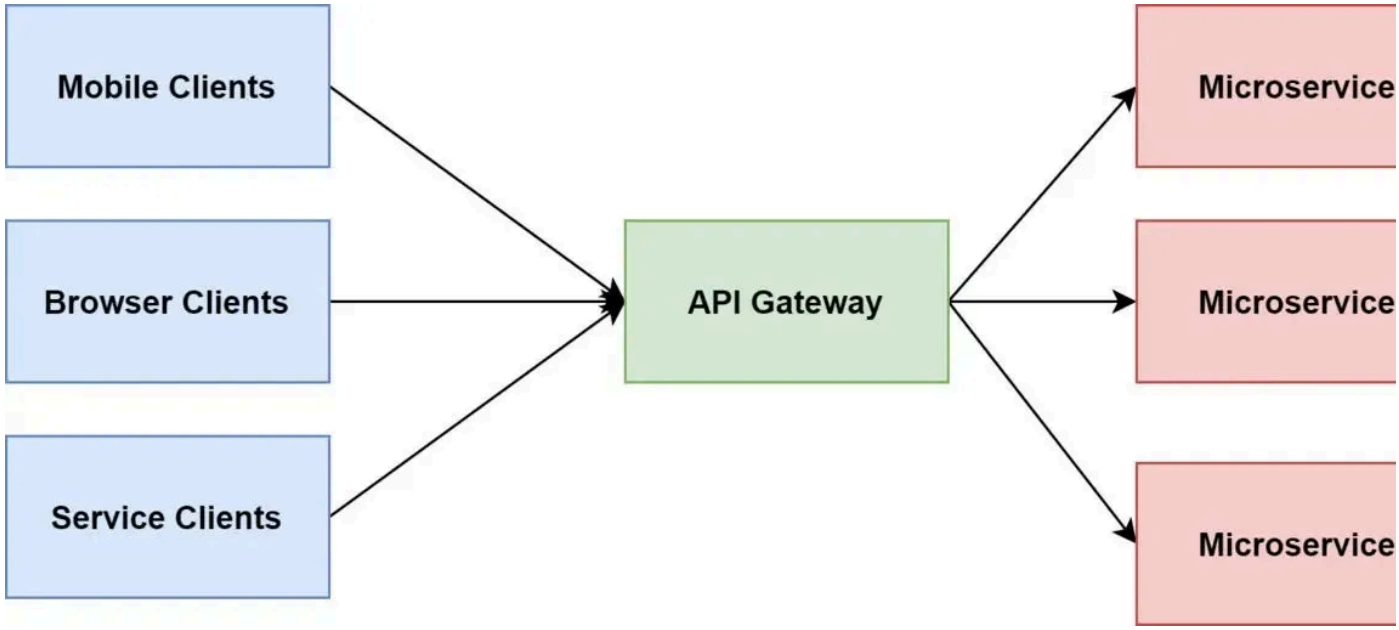
The insight: Start with a monolith, extract microservices when you have clear bounded contexts and team structure to support them.



12. API Gateway: The Single Entry Point

API gateways handle cross-cutting concerns like authentication, rate limiting, and request routing.

Why crucial: They prevent every microservice from implementing the same boilerplate code.



13. Database Indexing: Query Performance

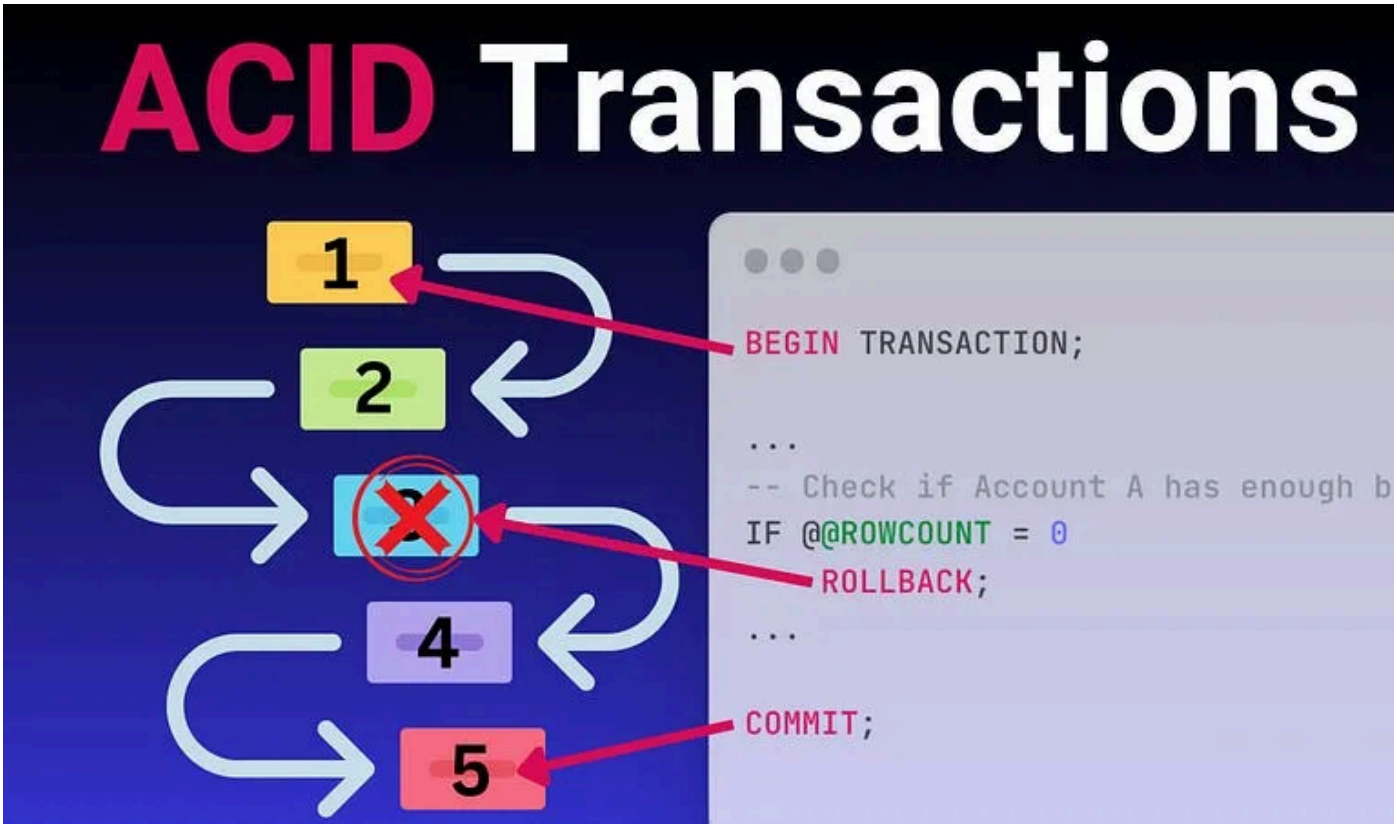
Indexes are data structures that improve query speed at the cost of storage and writ performance.

Advanced concept: Understand compound indexes, covering indexes, and when indexes actually hurt performance.

14. ACID vs BASE: Data Consistency Models

ACID: Atomicity, Consistency, Isolation, Durability (SQL databases) **BASE:** Basical Available, Soft state, Eventual consistency (NoSQL)

The decision framework: Use ACID for financial transactions, BASE for social me feeds.

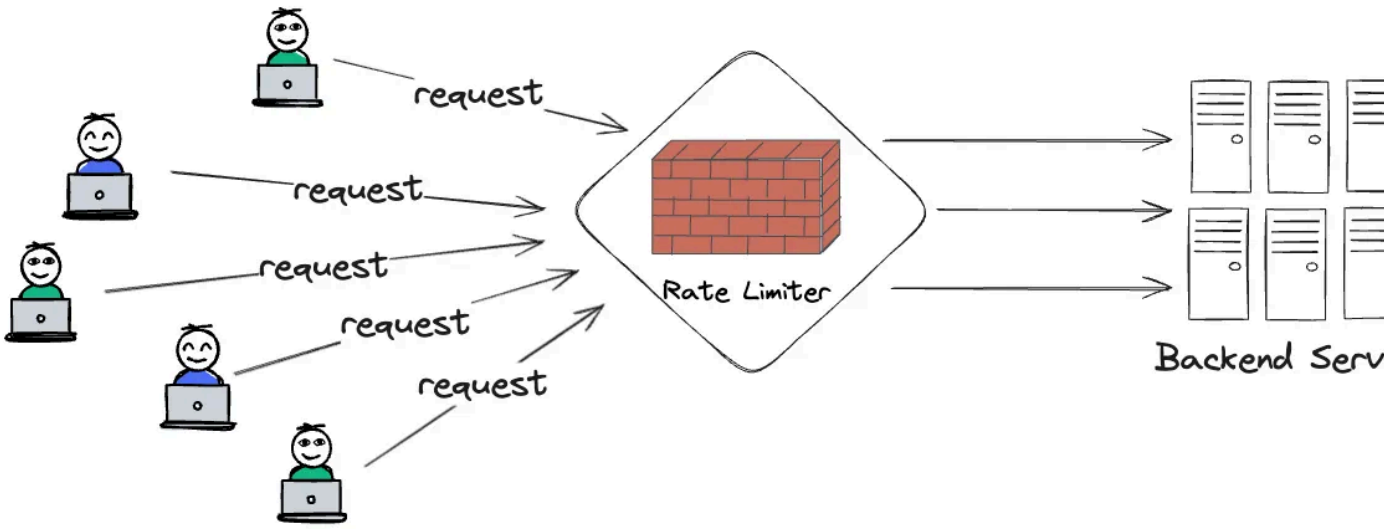


15. Rate Limiting: Protecting Your System

There are many different Rate limiting algorithms for controlling request rates:

- Token bucket
- Leaky bucket
- Fixed window
- Sliding window

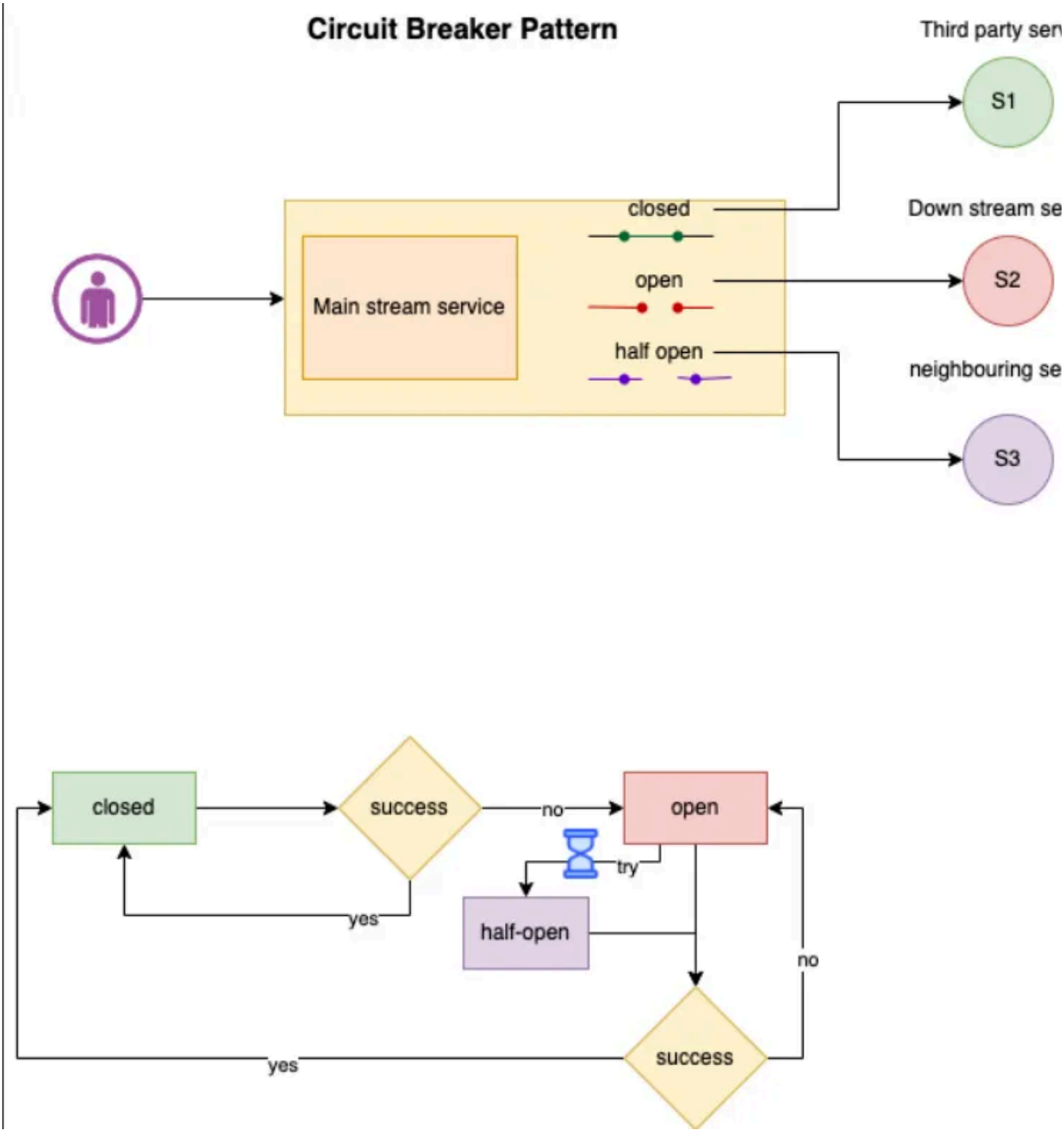
Real-world application: Twitter's rate limiting prevents spam and ensures fair usag across all users.



16. Circuit Breaker Pattern: Failure Resilience

This is a classic pattern which is asked multiple times on interview. This pattern Prevents cascade failures by temporarily stopping requests to a failing service.

States: Closed (normal), Open (failing), Half-open (testing recovery)



17. Distributed Consensus: Agreement in Chaos

Algorithms like Raft and Paxos help distributed systems agree on a single value eve with network partitions and node failures.

Why it matters: This is how systems like etcd (Kubernetes) and DynamoDB maintai consistency across replicas.

18. Eventual Consistency: The Distributed Reality

In distributed systems, achieving immediate consistency across all nodes is often impossible or impractical.

Examples:

- Social media likes (eventual consistency is fine)
- Bank transfers (strong consistency required)

19. Bloom Filters: Probabilistic Data Structures

Space-efficient data structure that tells you if an element is "definitely not in a set" "possibly in a set."

Use cases:

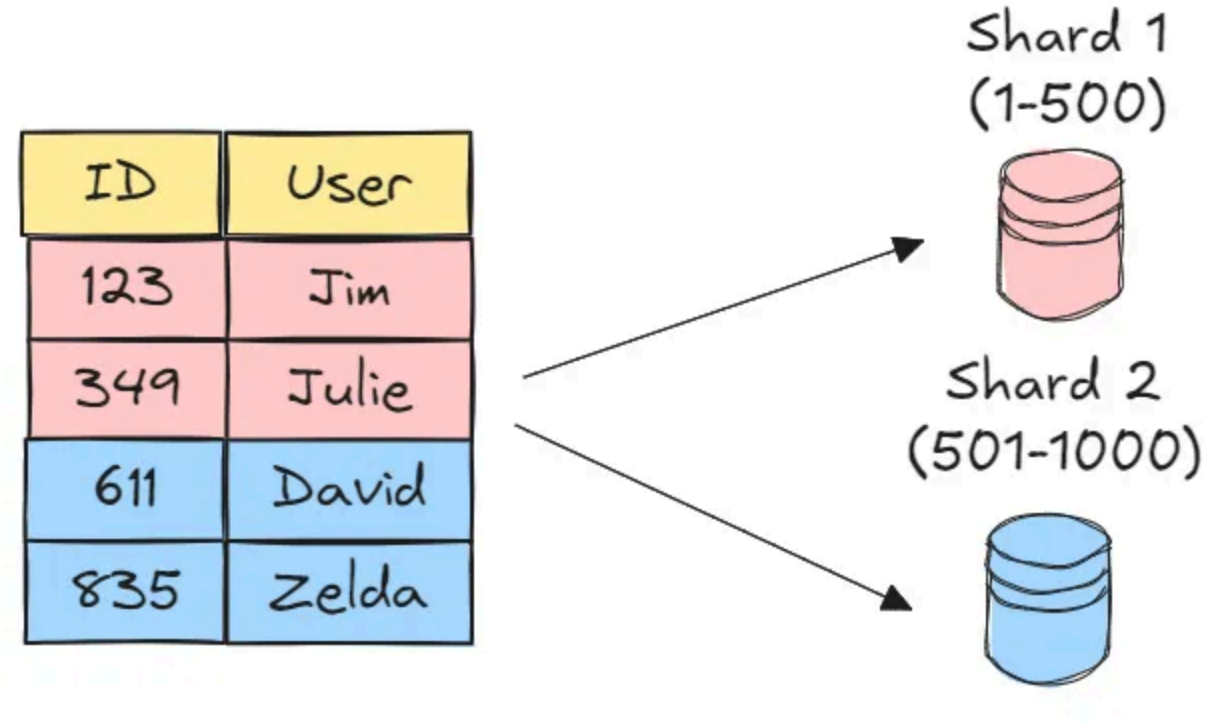
- Web crawlers avoiding duplicate URLs
- Databases checking if data exists before expensive disk reads

20. Data Partitioning Strategies

There are three main data partitioning strategies:

- **Vertical:** Split by features/columns
- **Horizontal:** Split by rows
- **Functional:** Split by service boundaries

Critical insight: Choosing the wrong partitioning key can create hotspots and unev load distribution.



How These Concepts Connected Everything?

The magic happened when I realized these concepts don't exist in isolation.

For example:

- **Netflix's architecture** combines CDNs for content delivery, microservices for different functions, event-driven architecture for recommendations, and sophisticated caching strategies.
- **WhatsApp's messaging** uses consistent hashing for user distribution, message queues for offline message delivery, and database sharding to handle billions of messages.

Once you understand how these concepts work together, you can design systems for any scale.

My Learning Strategy That Worked

Here's the exact approach I used to master these concepts:

1. Start with Fundamentals

I began with the basics using resources like ByteByteGo, which breaks down compl systems into digestible visual explanations. Their system design course was instrumental in building my foundation.

2. Practice with Real Examples

Sites like Codemia and System Design School provided excellent hands-on practice with real-world system design problems. The interactive approach helped me apply concepts immediately.

3. Deep Dive into Patterns

DesignGuru offered comprehensive coverage of system design patterns. Their Grokking the System Design Interview course became my bible.

4. Mock Interviews

Exponent and BugFree.ai provided peer to peer and AI-powered mock interviews tl helped me practice explaining my designs clearly and handling follow-up questions

5. Interactive Learning

Educative offered interactive courses that let me experiment with concepts in a har on environment.

6. Video Learning

You Tube and Udemy had comprehensive video courses that I could watch during commutes and lunch breaks.

7. Essential Reading

Designing Data-Intensive Applications became my go-to reference book. This book gold for understanding distributed systems deeply.

8. Open Source Learning

GitHub Repositories provided real-world examples and system design

Final Thoughts

Mastering system design doesn't happen overnight, but focusing on these **20 core concepts** will give you a strong foundation to tackle any interview with confidence.

If you want to go even deeper, I highly recommend resources like **ByteByteGo's System Design Interview course** and other structured programs that break down re world problems step by step. They are also offering 50% discount now on their lifeti plan.

With the right preparation and consistent practice, what once felt overwhelming w soon become second nature.

If you like this post, then don't forget to subscribe to **Soma's substack** where she sh more tips with System Design, Coding, Java, and React.js for developers.

React Java

A newsletter about Java, Programming, System Design, and React.js

By Soma

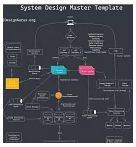
lsantos2000@gmail.com Subscribe

Other **System Design Articles** you may like

Stop Using HTTP for Everything: The API Protocol Guide That Sav Careers

JAVINPAUL • AUG 30

Read full story →



50 System Design Questions That Actually Prepare You for Real Interviews

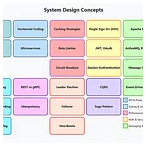
JAVINPAUL AND SOMA • AUG 29

Read full story →

Why 90% of APIs Fail (And How to Design Ones That Don't)

JAVINPAUL AND HAYK • AUG 26

Read full story →



How I Would Learn System Design in 2025 (If I Had To Start Over

JAVINPAUL AND SOMA • AUG 24

Read full story →

19 Likes · 4 Restacks

← Previous

A guest post by

Soma

Java and React Developer

Subscribe to Son

Discussion about this post

Comments Restacks

Write a comment...