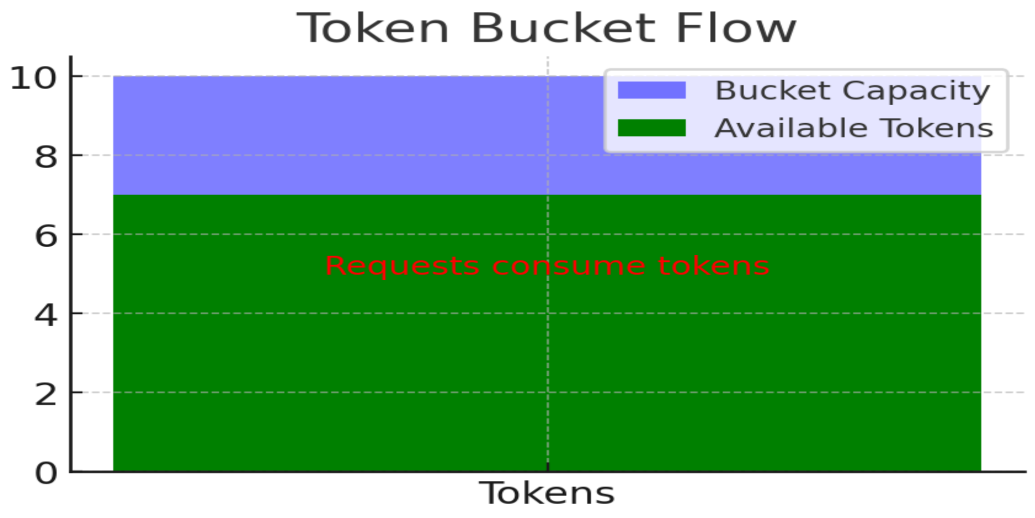
System Design Concepts - Illustrated & Explained

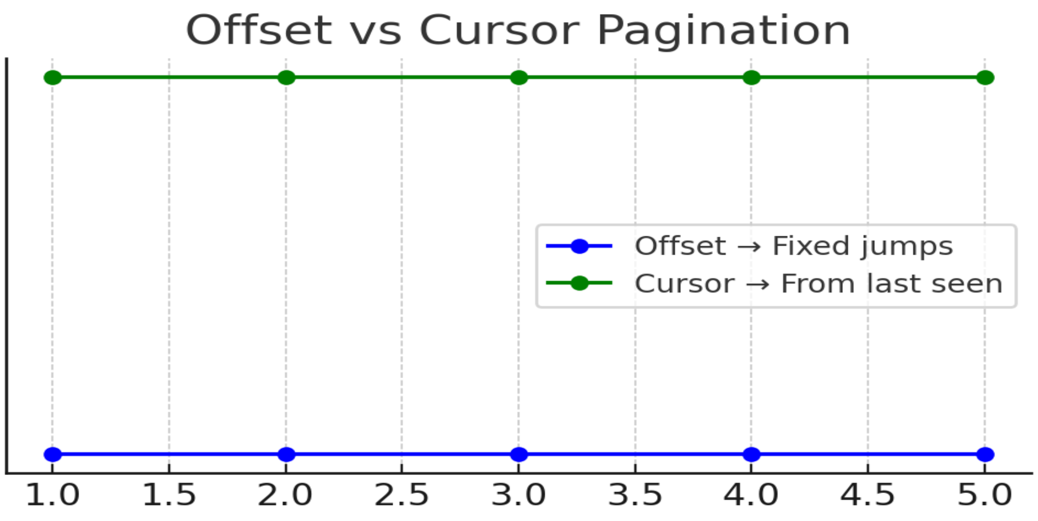
Token Bucket Algorithm

The Token Bucket algorithm is a rate-limiting mechanism widely used in distributed systems and networking. It works by maintaining a bucket that fills with tokens at a fixed rate (e.g., 10 tokens per second). Each request consumes one token. If tokens are available, the request is allowed; if not, it is delayed or rejected. This ensures fair usage of resources, prevents overload, and supports short bursts of traffic as long as tokens are available. It is often used in APIs, networking (QoS), and microservices to enforce throughput limits.



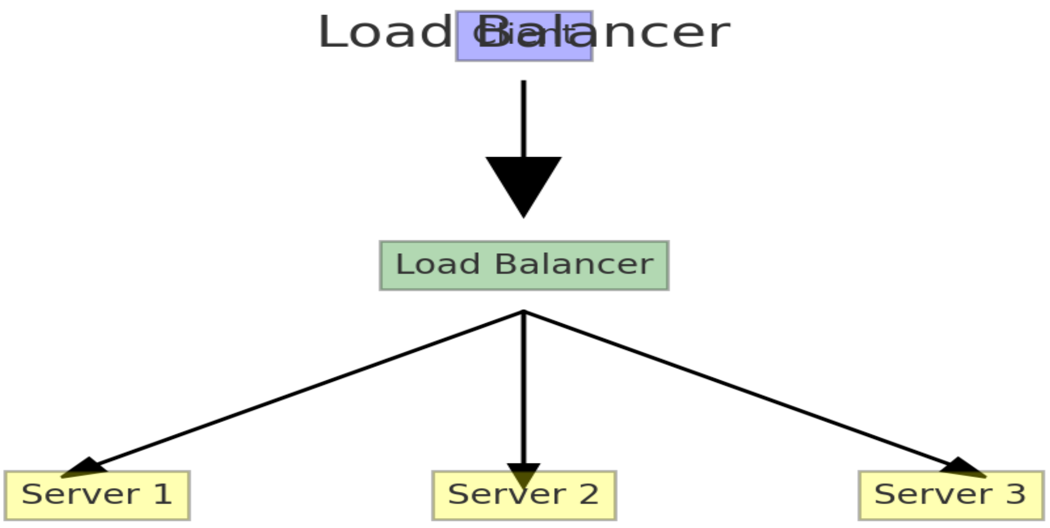
Offset vs Cursor Pagination

Offset pagination and cursor pagination are two methods of splitting large datasets into smaller chunks for efficient retrieval. Offset pagination relies on skipping a certain number of rows (e.g., LIMIT 10 OFFSET 50). It is easy to implement but becomes inefficient for large datasets because the database must scan and skip many rows. Cursor pagination, instead, uses a pointer (cursor) to the last retrieved record (e.g., an ID or timestamp). This allows the system to fetch subsequent pages efficiently without scanning. Cursor pagination is faster, avoids duplicates, and is more reliable for real-time applications (e.g., Twitter, Facebook feeds).



Load Balancers

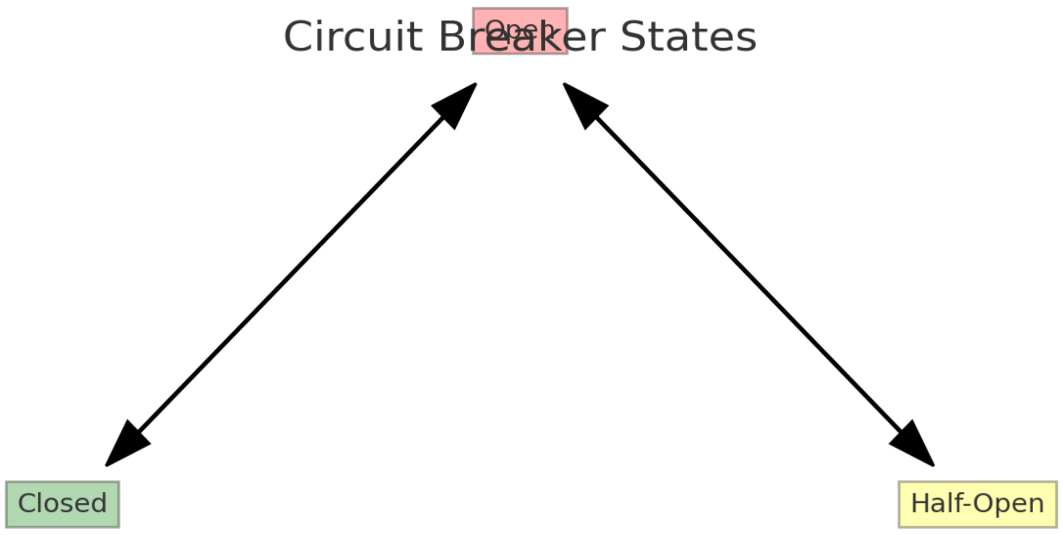
Load balancers distribute incoming traffic across multiple backend servers to ensure high availability, scalability, and fault tolerance. They can work at different layers: - Layer 4 (Transport Layer): Routes traffic based on IP and TCP/UDP ports. - Layer 7 (Application Layer): Routes based on URLs, cookies, headers. Load balancers can use algorithms like Round Robin, Least Connections, or IP Hash. They also provide health checks, SSL termination, and caching. In modern cloud setups, load balancers are critical for handling millions of concurrent requests while maintaining service reliability.



Circuit Breaker Pattern

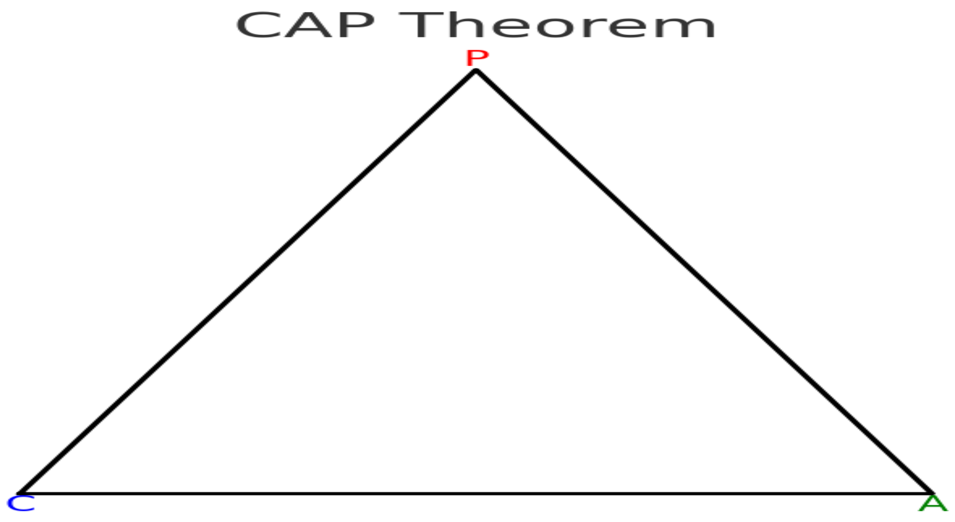
The Circuit Breaker pattern is a resilience design pattern that prevents cascading failures in distributed systems. When a downstream service starts failing (timeouts, errors), the circuit breaker "trips" and blocks further requests for a cooldown period. During this time, the system can quickly return fallback responses instead of waiting for failures. Once the cooldown expires, the circuit breaker transitions to a half-open state, allowing limited test requests. If successful, it closes again;

if not, it remains open. This prevents overloading failing services and improves overall system resilience.



CAP Theorem

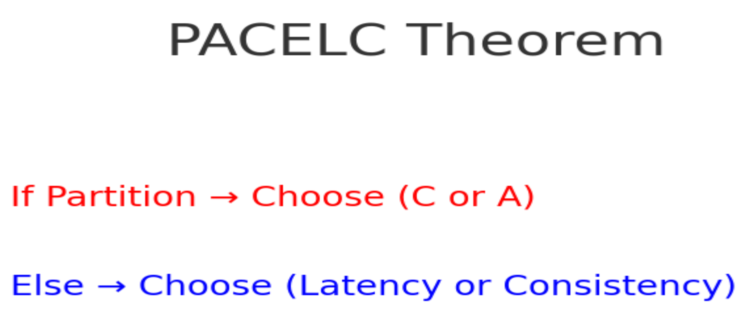
The CAP theorem (Consistency, Availability, Partition Tolerance) states that in the presence of a network partition, a distributed system can guarantee only two of the three properties: - Consistency: Every read gets the latest write or an error. - Availability: Every request gets a non-error response. - Partition Tolerance: The system continues functioning despite network splits. Since network partitions are inevitable in distributed systems, designers must choose between Consistency (CP systems, e.g., HBase) or Availability (AP systems, e.g., DynamoDB).



PACELC Theorem

The PACELC theorem extends CAP by stating that even in the absence of partitions, systems face a tradeoff: If Partition occurs (P), choose between Availability (A) or Consistency (C). Else (E), choose between Latency (L) or Consistency (C). For example: - DynamoDB is PA/EL (chooses

Availability under partition, Latency otherwise). - Spanner is PC/EC (chooses Consistency both under partition and normal conditions). This theorem provides a more realistic framework for evaluating real-world distributed databases.

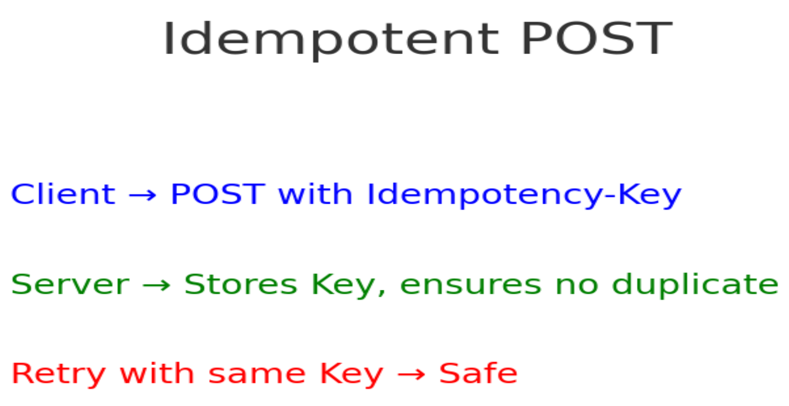

PACELC Theorem

If Partition → Choose (C or A)

Else → Choose (Latency or Consistency)

Idempotent POST

Normally, POST requests create new resources and are not idempotent (repeating them creates duplicates). However, in payment systems, booking APIs, or transactional workflows, we need retries to be safe. Idempotent POST is achieved using unique identifiers (Idempotency-Key). The client generates a unique key and sends it with the POST request. The server checks if a request with that key was already processed. If yes, it returns the previous response instead of creating a new resource. This prevents double payments or duplicate bookings.


Idempotent POST

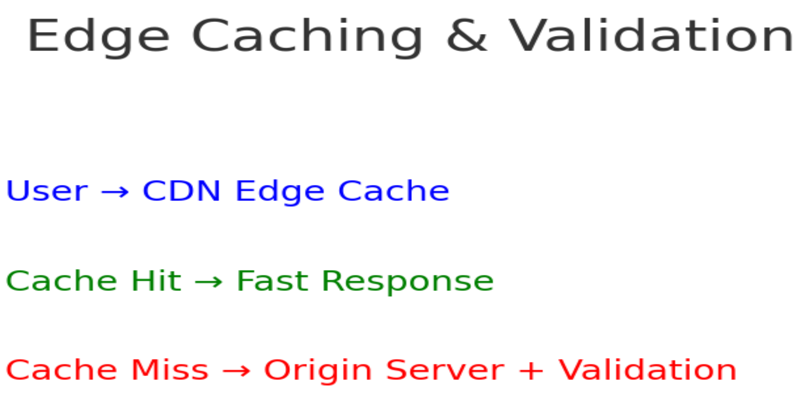
Client → POST with Idempotency-Key

Server → Stores Key, ensures no duplicate

Retry with same Key → Safe

Edge Caching & Validation

Edge caching improves performance by placing cached content closer to users at edge servers (e.g., CDNs like Cloudflare, Akamai). When a user requests content, the CDN serves it immediately if it's cached (cache hit). If the cache is stale or missing (cache miss), the CDN fetches from the origin server. To validate freshness, mechanisms like ETag or Last-Modified headers are used. The server can reply with 304 Not Modified if the content is unchanged, saving bandwidth and reducing latency. This is essential for global scale systems delivering web content, video streaming, and APIs.


Edge Caching & Validation
User → CDN Edge Cache
Cache Hit → Fast Response
Cache Miss → Origin Server + Validation