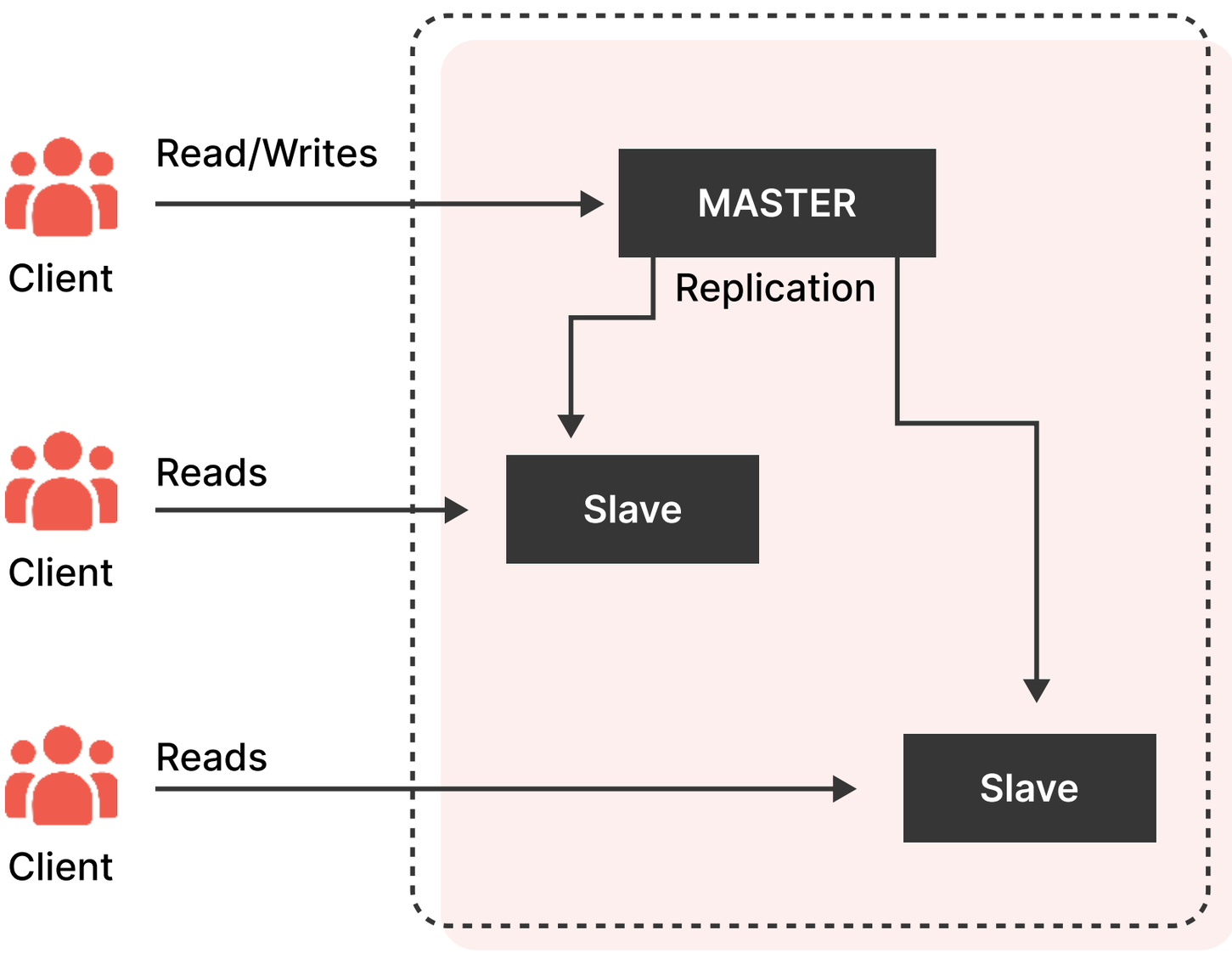
BOSSCODER ACADEMY

SYSTEM DESIGN

MOST ASKED

INTERVIEW QUESTIONS



Disclaimer

System Design is the most asked topic in tech interviews.

So, make sure you prepare it thoroughly.

Take the help of this doc and ace your System Design Interviews.

QUESTION-1

What is sharding in system design? EASY

Sharding is a technique to **horizontally scale a database** by splitting it into **multiple, independent servers** called **shards.**

🏅 Benefits:

- **Improved performance:** Each shard handles less data, leading to faster queries and processing.
- **Increased scalability:** Adding more shards easily expands the database capacity.

📁 Example:

Imagine a social media app with millions of users. Storing all data in one server would be overwhelming. We can shard based on user ID ranges:

- Shard 1: Users with IDs 1-1 million
- Shard 2: Users with IDs 1 million-2 million

Now:

- User queries are directed to the specific shard holding their data.
- Individual shards handle smaller workloads, improving performance.

- Adding more shards scales the database as the user base grows.

However, sharding also brings complexity:

- **Data distribution:** Determining the right sharding key and managing data movement across shards is crucial.
- **Joins and complex queries:** Combining data from multiple shards may require additional logic and can be slower.

QUESTION-2

What are bottlenecks in system design? EASY

Bottlenecks in system design are points or components that restrict overall performance. Identifying and addressing them is crucial for optimizing efficiency.

Common bottlenecks include:

1. **CPU Bottleneck:** When the CPU cannot process data fast enough to meet system demands, it becomes a limiting factor.
2. **Memory Bottleneck:** Insufficient RAM or slow memory access can hinder data storage and retrieval, slowing down the system.
3. **Storage Bottleneck:** Slow read/write speeds or limited storage capacity can impede system performance, especially in data-heavy applications.
4. **Database Bottleneck:** Inefficient database queries, poor indexing, or database contention can significantly affect application performance, particularly in database-dependent systems.

QUESTION-3

What is CAP Theorem? MEDIUM

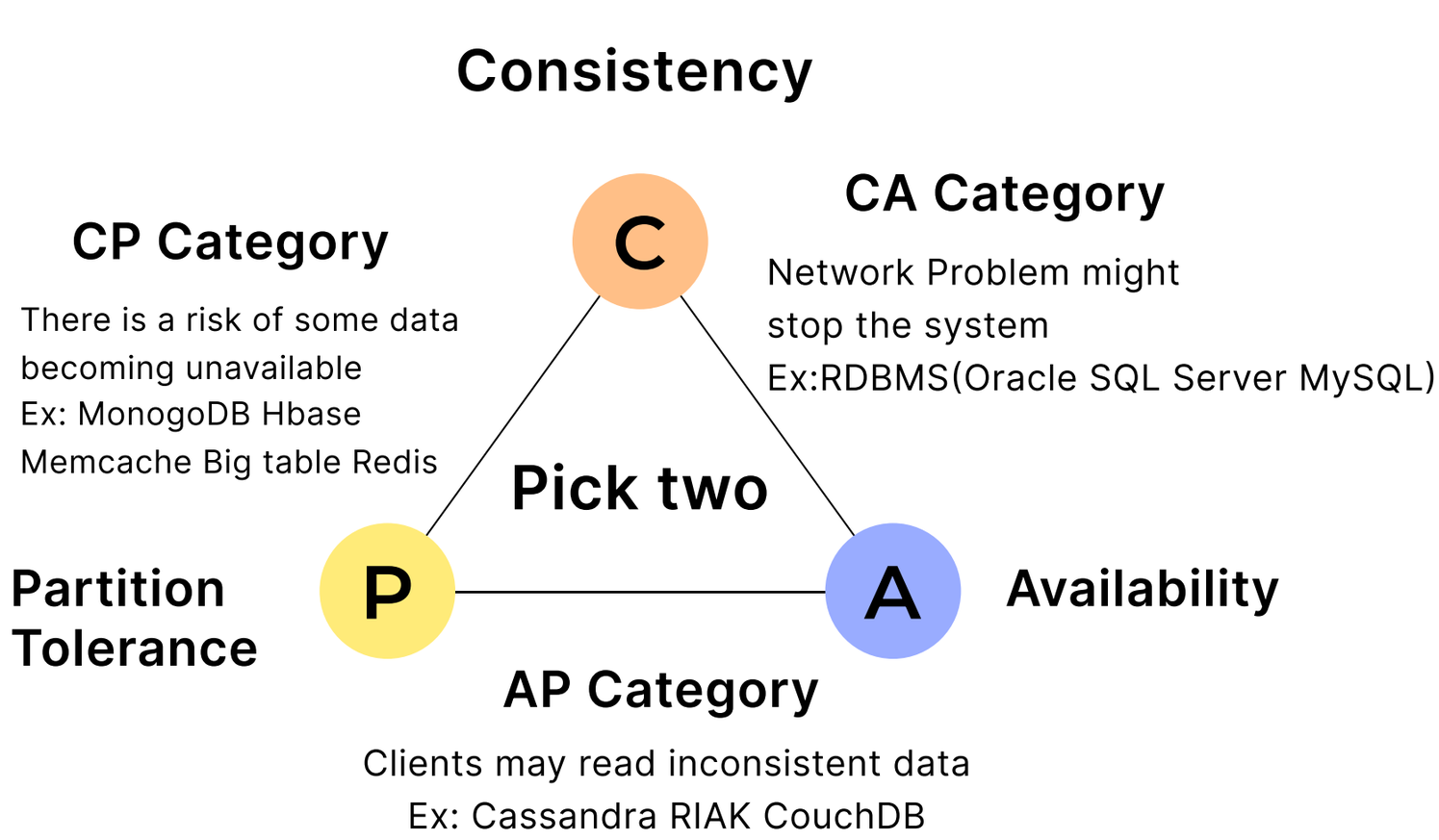
The **CAP theorem** (Brewer's theorem) states that a distributed system or database can provide only two out of the following three properties:

Consistency: Similar to ACID Properties, Consistency means that the state of the system before and after transactions should remain consistent.

Availability: This states that resources should always be available, there should be a non-error response.

Partition tolerance: Even when the network communication fails between the nodes in a cluster, the system should work well.

By the CAP theorem, all of these three properties cannot be achieved at the same time.



QUESTION-4

What is the difference between horizontal and vertical scaling? MEDIUM

Horizontal scaling involves adding more machines or nodes to a system to distribute the load and increase performance. **Vertical scaling** involves increasing the resources (CPU, RAM, storage, etc.) on a single machine to improve its performance.

Fundamentals	Horizontal Scaling	Vertical Scaling
Data Management	Scaling horizontally typically relies on data partitioning as each node only contains part of the data.	In vertical scaling, the data resides on a single node, and scaling is accomplished by multi-core, primarily by distributing the load among the machine's CPU and RAM resources.
Examples	Cassandra, MongoDB, Google Cloud Spanner	MySQL and Amazon RDS
Downtime possibility	You can scale with less downtime by adding additional computers to the existing pool because you are no longer constrained by the capacity of a single device.	There is an upper physical limit to vertical scaling, which is the scale of the current hardware.Vertical scaling is restricted to the capacity of one machine because expanding over that limit can result in downtime.

Fundamentals	Horizontal Scaling	Vertical Scaling
Concurrency Models	As it entails distributing jobs among devices over a network is known as distributed programming. Several patterns are connected to this model: MapReduce, Master/ Worker*, Blackboard, and many spaces.	It involves the Actor model: Multi-threading and in-process message forwarding are frequently used to implement concurrent programming on multi-core platforms.
Message Passing	Data sharing is more difficult in distributed computing because there isn't a shared address space. Since you will send copies of the data, it also increases the cost of sharing, transferring, or updating data.	Data sharing and message passing Can be accomplished by passing a reference in a multi-threaded scenario since it is reasonable to presume that there is a shared address space.

Choosing between horizontal and vertical scaling depends on your specific needs. Here are some general guidelines:

- Use horizontal scaling for:
  a. Handling high workloads and surges in traffic.
  b. Building highly resilient and available systems.
  c. Processing large datasets efficiently.

- Use vertical scaling for:
  a. Simple workloads and applications.
  b. Rapid deployment and testing.
  c. Cost-efficiency for low workloads.

QUESTION-5

Describe the RESTful API design principles. MEDIUM

1. **Uniform Interface:** Consistent resource naming and actions using HTTP methods (GET, POST, PUT, DELETE).
2. **Client-Server:** Separation of concerns between clients making requests and servers handling them.
3. **Statelessness:** Each request contains all information needed, servers don't "remember" past requests.
4. **Cacheable:** Resources can be cached by clients or intermediaries for better performance.
5. **Layered System:** Intermediaries can be placed between clients and servers without affecting communication.
6. **Code on Demand (Optional):** Servers can send executable code to clients to extend functionality.

These principles lead to well-designed, predictable, and scalable APIs.

QUESTION-6

How can you select which webservice to use between REST and SOAP? MEDIUM

When deciding between SOAP and REST for web services, consider the following factors:

1. **Nature of Data/Logic Exposure:**
   SOAP: Used for exposing business logic.
   REST: Used for exposing data.

2. **Formal Contract Requirement:**
   SOAP: Provides strict contracts through WSDL.
   REST: No strict contract requirement.

3. **Data Format Support:**
   SOAP: Limited support.
   REST: Supports multiple data formats.

4. **AJAX Call Support:**
   SOAP: No direct support.
   REST: Supports XMLHttpRequest for AJAX calls.

5. **Synchronous/Asynchronous Requests:**
   SOAP: Supports both sync and async.
   REST: Supports only synchronous calls.

6. **Statelessness Requirement:**
   SOAP: No.
   REST: Yes.

7. **Security Level:**
   SOAP: Preferred for high-security needs.
   REST: Security depends on underlying implementation.

8. **Transaction Support:**
   SOAP: Provides advanced support for transactions.
   REST: Limited transaction support.

9. **Bandwidth/Resource Usage:**
   SOAP: High bandwidth due to XML data overhead.
   REST: Uses less bandwidth.

10. **Development and Maintenance Ease:**
    SOAP: More complex.
    REST: Known for simplicity, easy development, testing, and maintenance.

QUESTION-7

What are Content Delivery Networks (CDN) ? MEDIUM

- A CDN is a geographically distributed network of servers strategically placed worldwide that cache and deliver static content (images, videos, JavaScript, CSS, etc.) to users, optimising performance and availability.
- Key concepts:

  a. **Edge servers:** Physically located at Points of Presence (PoPs) closer to users for faster content delivery.

  b. **Caching:** Stores frequently accessed content on edge servers, reducing origin server load and latency.

  c. **Routing:** Directs user requests to the nearest edge server with cached content.

  d. **Security:** Can provide DDoS protection, SSL/TLS offloading, and other security features.

Types of CDNs:

1. Pull CDNs:

- Edge servers fetch content from the origin server upon user request.
- Suitable for content with low update frequency or high demand spikes.

2. Push CDNs:

- Content proactively pushed to edge servers, ensuring immediate availability.
- Ideal for dynamic content that changes frequently or has high peak periods.

QUESTION-8

Explain the difference between data sharding and database partitioning? MEDIUM

Both partitioning and sharding are techniques for managing large datasets in databases. However, they differ in their scope and implementation:

Partitioning:

- **Definition:** Dividing a table into smaller logical segments based on specific criteria.
- **Location:** Within a single database server.
- **Types:** Vertical (split columns) and horizontal (split rows).
- **Benefits:** Improved query performance for specific types of queries.
- **Drawbacks:** Limited scalability, potential performance bottlenecks on a single server.

Sharding:

- **Definition:** A specific type of horizontal partitioning where data is distributed across multiple separate servers.

- **Location:** Each partition (shard) resides on a dedicated server.
- **Benefits:** Excellent scalability, high availability, faster query execution due to true parallelism.
- **Drawbacks:** Increased complexity, potential data consistency issues, higher infrastructure costs.

Choosing the right technique:

- For smaller datasets or non-critical applications, partitioning within a single server might be sufficient.
- For large, heavily accessed datasets requiring high scalability and performance, sharding across multiple servers is the ideal choice.

QUESTION-9

Explain availability, reliability, latency, performance, throughput in system design. MEDIUM

1. Availability: This refers to the percentage of time your system is up and running, accessible to users, and fulfilling its intended purpose. High availability systems aim for near-constant uptime, often measured in "nines" (e.g., 99.99% uptime).

2. Reliability: This builds on availability and focuses on the system's consistency in delivering correct and expected results. A reliable system performs as intended without errors or unexpected behavior, even under varying conditions.

3. Latency: This signifies the time it takes for a single request or operation to complete within the system. Think of it as the response time, measured in milliseconds (ms) or seconds (s). Low latency is crucial for real-time interactions and user experience.

4. Performance: This encompasses the overall responsiveness and efficiency of your system. It considers factors like latency, throughput, resource utilization, and scalability. A performant system handles requests quickly, utilizes resources effectively, and scales smoothly with increased demands.

5. Throughput: This measures the number of requests or operations your system can successfully process per unit of time. It's often expressed in requests per second (RPS) or transactions per second (TPS). High throughput ensures the system can handle large volumes of users or data.

QUESTION-10

What is consistency? What are its different types? MEDIUM

Consistency refers to the agreement between multiple copies of the same data stored across different servers. It ensures users have the same view of the data, regardless of their location.

There are three main types of consistency:

1. Eventual Consistency:

- Reads may not immediately reflect the latest write, but eventually (within milliseconds) will.
- Think of sending an email: the recipient won't see it instantly, but it will arrive eventually.
- Used in scalable systems like DNS and email due to its high availability and low cost.

2. Strong Consistency:

- Reads always reflect the latest write, meaning data is replicated synchronously across all servers.
- Provides strict consistency but sacrifices scalability and performance.

- Used in systems like databases and filesystems where data integrity is crucial.

3. Weak Consistency:

- Focuses on fast access, allowing reads to potentially see outdated data.
- No well-defined rules for data updates, so different servers might return different values.
- Used in real-time applications like VoIP and video chat where speed is prioritized over absolute data accuracy.

Choosing the right consistency model depends on your application's specific needs:

- **Prioritize accuracy?** Choose strong consistency.
- **Need high availability and scalability?** Consider eventual consistency.
- **Speed is essential?** Weak consistency might be acceptable.

QUESTION-11

What are some metrics for measuring system performance? MEDIUM

User Experience:

- **Apdex Score:** Blends response time and success rate for user satisfaction.
- **Average Response Time:** How long it takes the system to answer user requests.

System Health:

- **Throughput:** Number of requests handled per unit of time (helps determine scaling needs).
- **Availability/Uptime:** Percentage of time the system is operational.
- **Error Rates:** Frequency of errors encountered during operation.

Resource Utilization:

- **Latency & CPU:** Monitors data transfer speed and CPU usage to identify bottlenecks.
- **Garbage Collection:** Tracks memory management efficiency.

Others:

- **Database Performance:** Monitor key database metrics for optimal operation.
- **Network Performance:** Track bandwidth, packet loss, and latency for network reliability.
- **Security Metrics:** Monitor events and access controls for system security.

QUESTION-12

What is caching in system design? Explain types of caching based on data consistency. MEDIUM

Caching is a powerful optimization technique that stores frequently accessed data in a temporary location closer to the requesting process. This location, called a cache, is generally faster to access than the original data source, significantly improving system performance and scalability.

Based on Data Consistency:

Write-through:

- Writes happen to both cache and database simultaneously.
- Pros: Fast reads, high data consistency.
- Cons: High write latency due to double writes.

Write-around:

- Writes bypass the cache and go directly to the database.
- Pros: Potentially lower write latency.
- Cons: Increased cache misses, leading to higher read latency for frequently written and re-read data.

Write-back:

- Writes occur only in the cache, confirmed immediately. Data is asynchronously synced to the database later.
- Pros: Lower write latency, higher throughput for write-intensive workloads.
- Cons: Risk of data loss if the cache crashes before syncing. Improved reliability with multiple write acknowledgements.

QUESTION-13

What are message queues in system design? MEDIUM

Message queues are a form of service-to-service communication that facilitates asynchronous communication. They are used to effectively manage requests in large-scale distributed systems.

Here are some features of message queues:

- Push or Pull Delivery: Provides options for continuous querying (pull) or notification-based retrieval (push), with support for long-polling.
- FIFO (First-In-First-Out) Queues: Processes the oldest entry in the queue first.
- Schedule/Delay Delivery: Allows setting specific delivery times, including common delays using delay queues.
- At-Least-Once Delivery: Ensures message delivery by storing multiple copies and resending in case of failures.
- Exactly-Once Delivery: Filters out duplicates to guarantee single delivery.
- Dead-letter Queues: Stores unprocessable messages for further inspection without blocking.

- Ordering: Delivers messages in sender's order, ensuring each message is received at least once.
- Task Queues: Executes tasks in the background, supporting scheduling and intensive computations.

QUESTION-14

What is N-tier architecture in system design? MEDIUM

N-tier architecture divides an application into logical layers and physical tiers to manage dependencies and separate responsibilities.

Tiers in N-tier architecture are physically separated, possibly running on different machines. Communication between tiers can be direct or asynchronous, improving scalability and resilience while introducing latency due to network communication.

Two types of N-tier architectures exist:

- Closed-layer architecture restricts layers to call only the immediately lower layer.
- Open-layer architecture allows a layer to call any layer below it.

N-Tier architecture examples:

- 3-Tier architecture: Comprising Presentation, Business Logic, and Data Access layers.
- 2-Tier architecture: Involving client-side Presentation layer communicating directly with a data store.
- Single Tier (1-Tier) architecture: Simplest form where all components run on a single server or application.

QUESTION-15

What are virtual machines? What are the benefits of using virtual machines? MEDIUM

- A VM is a virtual environment mimicking a computer system with its own CPU, memory, network interface, and storage, hosted on physical hardware.
- A hypervisor separates hardware resources from the VM and manages them.

Benefits of Using VMs:

- **Server Consolidation:** Virtualizing servers enables efficient utilization of physical resources by hosting multiple VMs on one server.
- **Isolation:** VMs provide a segregated environment, preventing interference between VMs and host hardware.
- **Testing and Production Environments:** VMs are ideal for testing new applications or setting up production environments due to their isolated nature.
- **Specific Use Cases:** Single-purpose VMs can be employed to support particular functions.

QUESTION-16

What are containers? What are their advantages? MEDIUM

Containers package code and dependencies to ensure applications run consistently across different computing environments.

Need for Containers:

- **Separation of Responsibility:** Allows developers to focus on application logic while operations teams handle deployment.
- **Workload Portability:** Containers can run anywhere, simplifying development and deployment processes.
- **Application Isolation:** Virtualizes resources at the OS level, providing logical isolation for applications.
- **Agile Development:** Enables rapid development by eliminating concerns about dependencies and environments.
- **Efficient Operations:** Lightweight containers optimize resource usage, ensuring efficient computing.

QUESTION-17

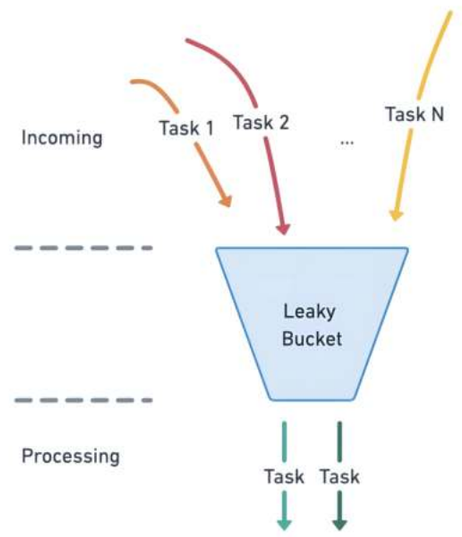
What is rate limiting? Explain a few algorithms used in API rate limiting. MEDIUM

Rate limiting is a crucial strategy in large-scale systems to prevent the frequency of operations from surpassing a defined threshold.

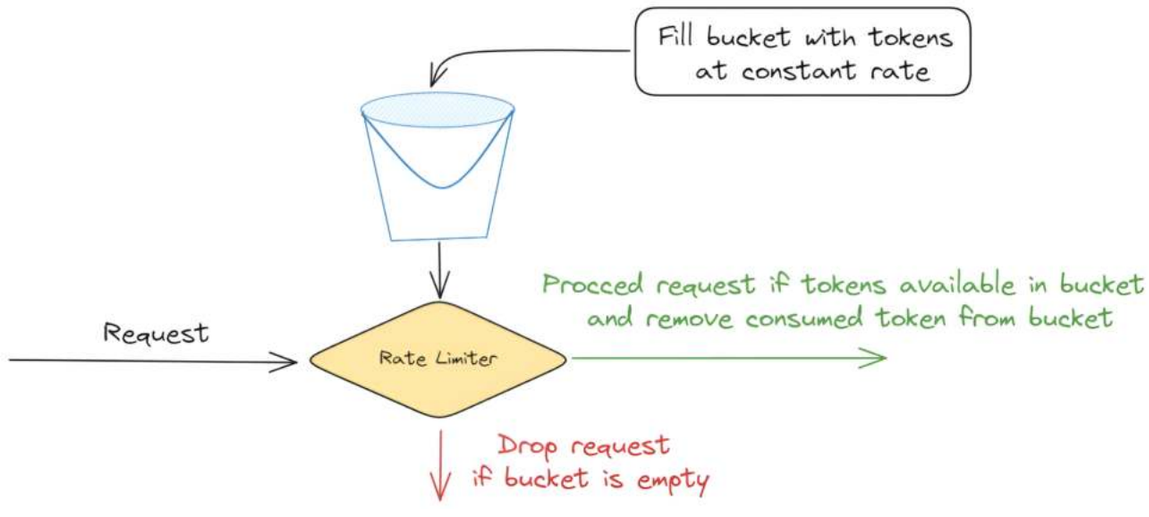
It safeguards our APIs from both unintentional and malicious overuse by constraining the quantity of requests permitted to access our API within a specified timeframe.

The following are few API Rate Limiting algorithms:

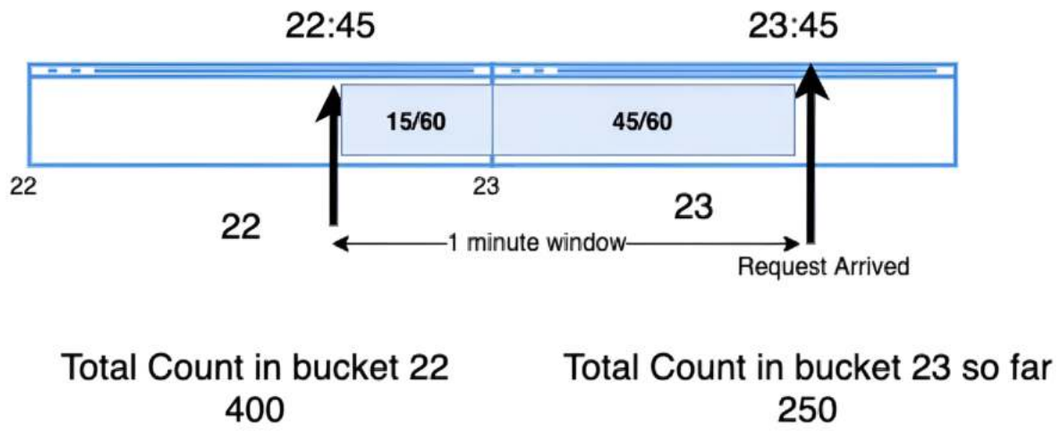
1. **Leaky Bucket:**
   Utilizes a queue system where requests are processed at a constant rate.
   Additional requests are discarded if the queue is full.



2. **Token Bucket:**
   Requests require tokens from a bucket for processng.
   Refuses requests if no tokens are available, refreshing the bucket over time.

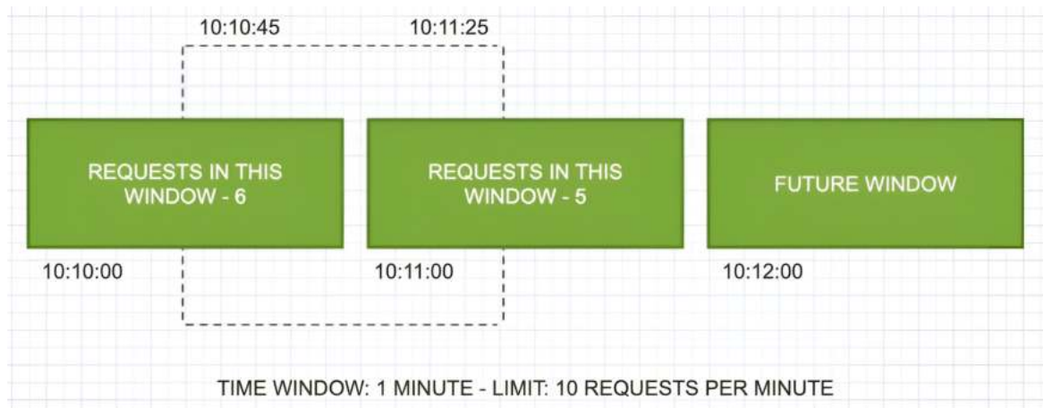


3. **Fixed Window:**
   Uses a fixed time window to track request rates.
   Requests exceeding a threshold within the window are discarded.



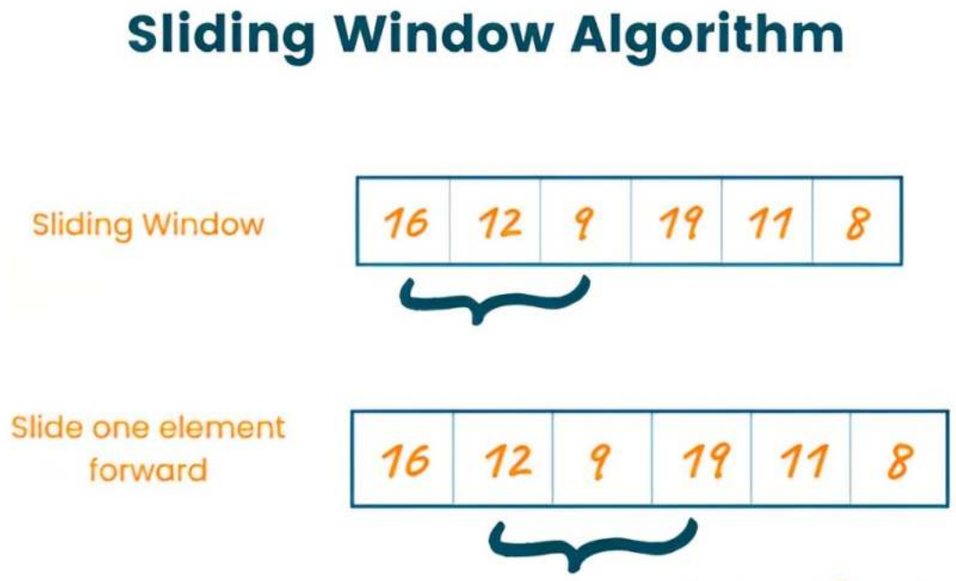
---

4. **Sliding Log:**
   Uses a fixed time window to track request rates.
   Requests exceeding a threshold within the window are discarded.



---

4. **Sliding Window:**
   Combines elements of fixed window and sliding log approaches.
   Tracks request rates for each fixed window, considering weighted values from previous windows for smoother traffic handling.



QUESTION-18

What is consistent hashing? MEDIUM

Consistent hashing is a strategy used to distribute keys/data among multiple servers in a distributed system.

It employs a hash function to map keys to points on a circular hash ring, where each server is associated with a point on the ring.

When a key request arrives, the hash function determines the corresponding point on the ring, and the server closest to that point handles the request.

Benefits of consistent hashing include:

- Minimizing key remapping when servers change.
- Allowing dynamic cluster resizing without significant overhead.
- Evenly distributing load across servers.

It's widely used in distributed systems like caching systems and content delivery networks to evenly distribute data and is integral to distributed hash table implementation.

QUESTION-19

What are the advantages and disadvantages of microservices architecture? HARD

Microservices are an architectural style that structures an application as a collection of small, loosely coupled, and independently deployable services.

Key Concepts:

- Independence: Each service has specific business function. Developed & scaled separately.
- Modularity: Breaking down a large, monolithic application into smaller, manageable pieces.

Advantages of Microservices:

- **Agility and Speed:** Faster development and deployment cycles due to independent services.
- **Scalability:** Individual services can be scaled up or down independently based on demand.
- **Resilience:** Failure of one service doesn't cripple the entire app.
- **Technology Choice:** Each service can use the best tool for the job without affecting others.

Disadvantages of Microservices:

- **Complexity:** Increased overhead in managing infrastructure, communication, and monitoring.
- **Testing:** Testing complex distributed systems can be challenging and time-consuming.
- **Debugging:** Identifying and fixing issues across services can be difficult.
- **Cost:** Initial setup and ongoing maintenance can be more expensive than monolithic.

QUESTION-20

When would you prefer to use an SQL database and when a NoSQL database? HARD

Use an SQL database if:

- **Structured data with well-defined relationships:** Your data fits neatly into tables with rows and columns, and there are clear connections between different tables. SQL excels at querying and manipulating such data.
- **Strong data consistency and ACID properties are crucial:** Transactions must be atomic, consistent, isolated, and durable. SQL adheres to ACID principles, ensuring data integrity for financial systems, regulatory applications, etc.
- **Complex queries and reporting are essential:** SQL's powerful query language enables intricate data analysis and report generation.

Use a NoSQL database if:

- **Unstructured or semi-structured data:** Your data doesn't adhere to a rigid structure, including documents, JSON, or graphs. NoSQL offers greater flexibility in storing and managing such data.

- **High scalability and performance are primary concerns:** You anticipate significant data growth and require rapid read/write operations. NoSQL often scales horizontally, adding more servers to handle increased load.
- **Data consistency trade-offs are acceptable:** Your application priorities might emphasize speed and availability over absolute data consistency.

BOSSCODER ACADEMY

WHY BOSSCODER?

- **750+** Alumni placed at Top Product-based companies.
- More than **136% hike** for every **2 out of 3** working professional.
- Average package of **24LPA.**

> The syllabus is most **up-to-date** and the list of problems provided covers **all important topics.**
>
> **Lavanya**
> Meta

> Course is very well **structured** and **streamlined** to crack any **MAANG** company
>
> **Rahul .**
> Google

EXPLORE MORE