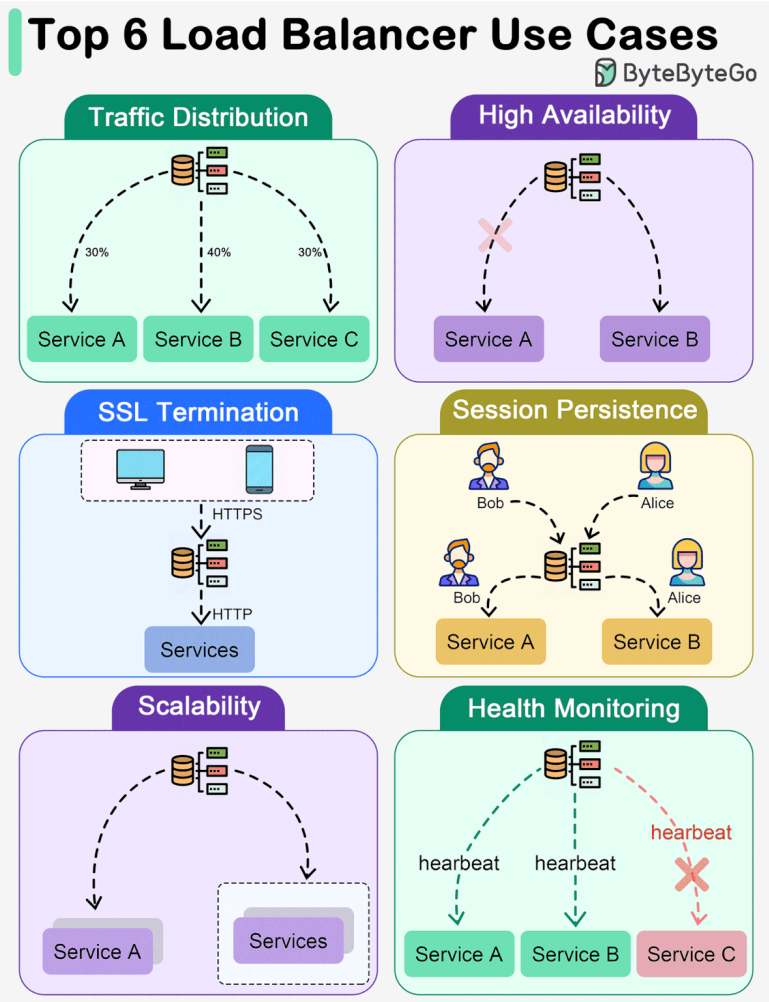
Key Use Cases for Load Balancers

The diagram below shows top 6 use cases where we use a load balancer.



- Traffic Distribution
  Load balancers evenly distribute incoming traffic among multiple servers, preventing any single server from becoming overwhelmed. This helps maintain optimal performance, scalability, and reliability of applications or websites.

- High Availability
  Load balancers enhance system availability by rerouting traffic away from failed or unhealthy servers to healthy ones. This ensures uninterrupted service even if certain servers experience issues.

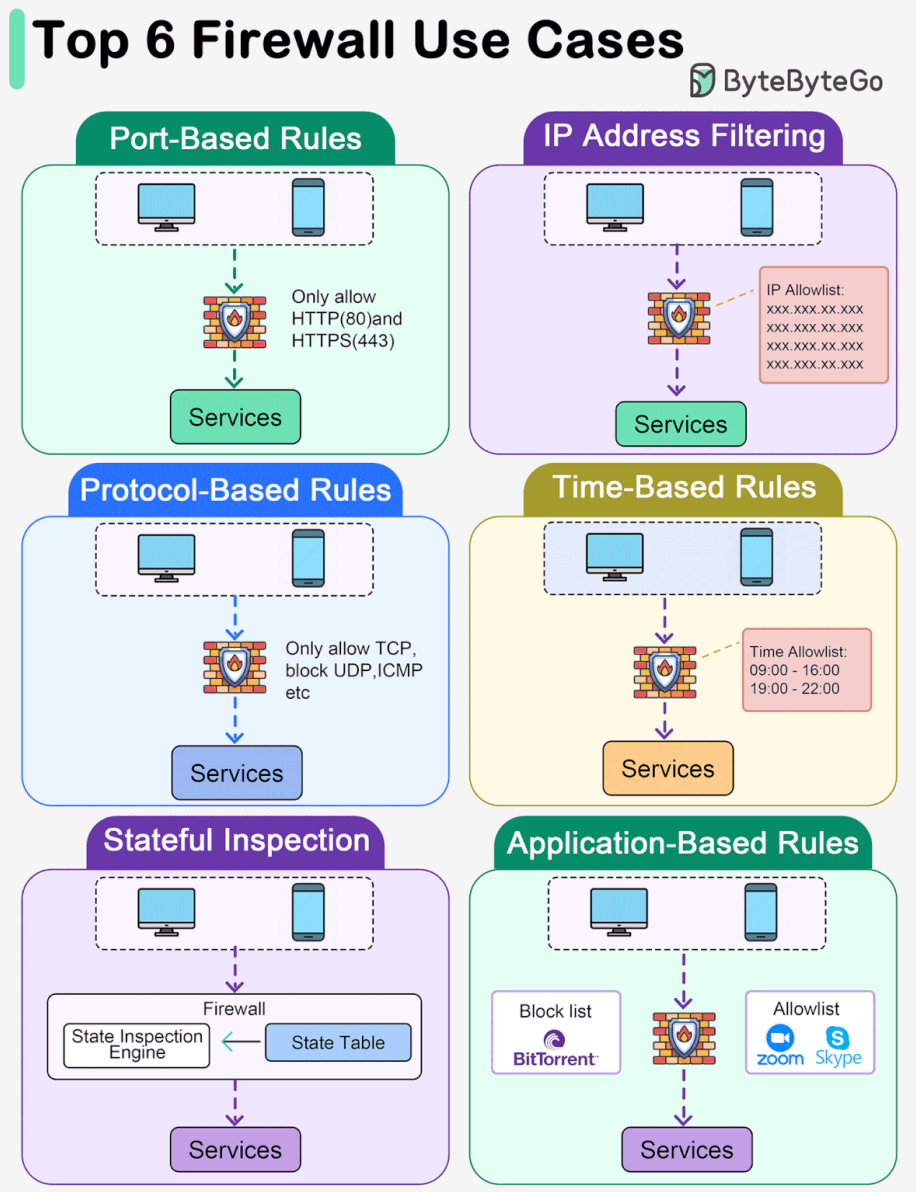
- SSL Termination
  Load balancers can offload SSL/TLS encryption and decryption tasks from backend servers, reducing their workload and improving overall performance.

- Session Persistence
  For applications that require maintaining a user's session on a specific server, load balancers can ensure that subsequent requests from a user are sent to the same server.

- Scalability
  Load balancers facilitate horizontal scaling by effectively managing increased traffic. Additional servers can be easily added to the pool, and the load balancer will distribute traffic across all servers.

- Health Monitoring
  Load balancers continuously monitor the health and performance of servers, removing failed or unhealthy servers from the pool to maintain optimal performance.

Top 6 Firewall Use Cases



- Port-Based Rules
  Firewall rules can be set to allow or block traffic based on specific ports. For example, allowing only traffic on ports 80 (HTTP) and 443 (HTTPS) for web browsing.

- IP Address Filtering
  Rules can be configured to allow or deny traffic based on source or destination IP addresses. This can include whitelisting trusted IP addresses or blacklisting known malicious ones.

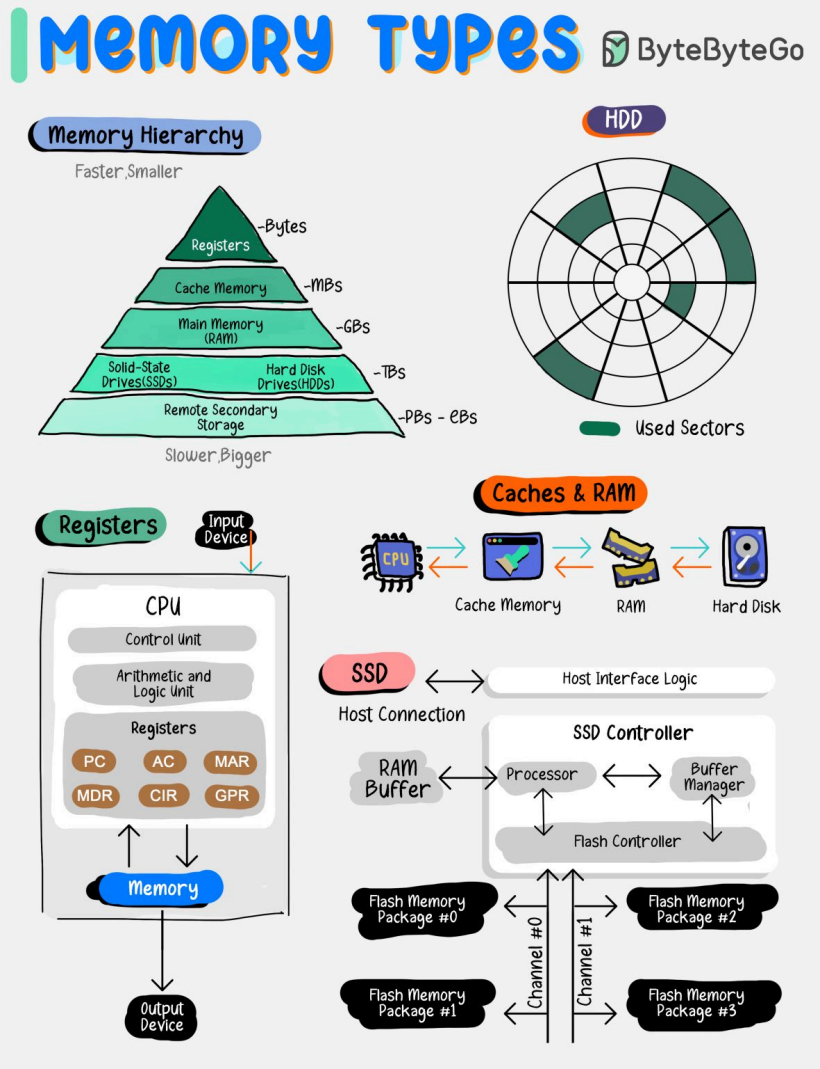
- Protocol-Based Rules
  Firewalls can be configured to allow or block traffic based on specific network protocols such as TCP, UDP, ICMP, etc. For instance, allowing only TCP traffic on port 22 (SSH).

- Time-Based Rules
  Firewalls can be configured to enforce rules based on specific times or schedules. This can be useful for setting different access rules during business hours versus after-hours.

- Stateful Inspection
  Stateful Inspection: Stateful firewalls monitor the state of active connections and allow traffic only if it matches an established connection, preventing unauthorized access from the outside.

- Application-Based Rules
  Some firewalls offer application-level control by allowing or blocking traffic based on specific applications or services. For instance, allowing or restricting access to certain applications like Skype, BitTorrent, etc.

Types of memory. Which ones do you know?



Memory types vary by speed, size, and function, creating a multi-layered architecture that balances cost with the need for rapid data access.

By grasping the roles and capabilities of each memory type, developers and system architects can design systems that effectively leverage the strengths of each storage layer, leading to improved overall system performance and user experience.

Some of the common Memory types are:

1. Registers:
   Tiny, ultra-fast storage within the CPU for immediate data access.

2. Caches:
   Small, quick memory located close to the CPU to speed up data retrieval.

3. Main Memory (RAM):
   Larger, primary storage for currently executing programs and data.

4. Solid-State Drives (SSDs):
   Fast, reliable storage with no moving parts, used for persistent data.

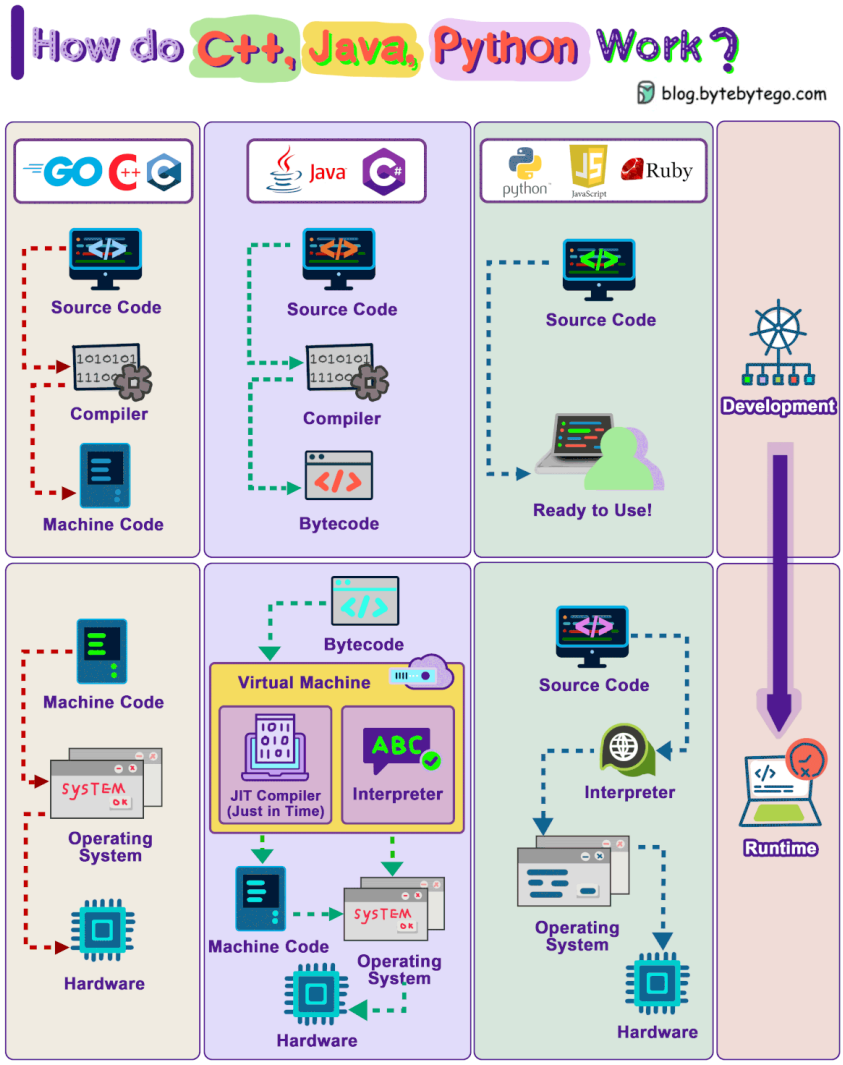
5. Hard Disk Drives (HDDs):
   Mechanical drives with large capacities for long-term storage.

6. Remote Secondary Storage:
   Offsite storage for data backup and archiving, accessible over a network.

Over to you: Which memory type resonates most with your tech projects and why? Share your thoughts!

How Do C++, Java, Python Work?

The diagram shows how the compilation and execution work.



Compiled languages are compiled into machine code by the compiler. The machine code can later be executed directly by the CPU. Examples: C, C++, Go.

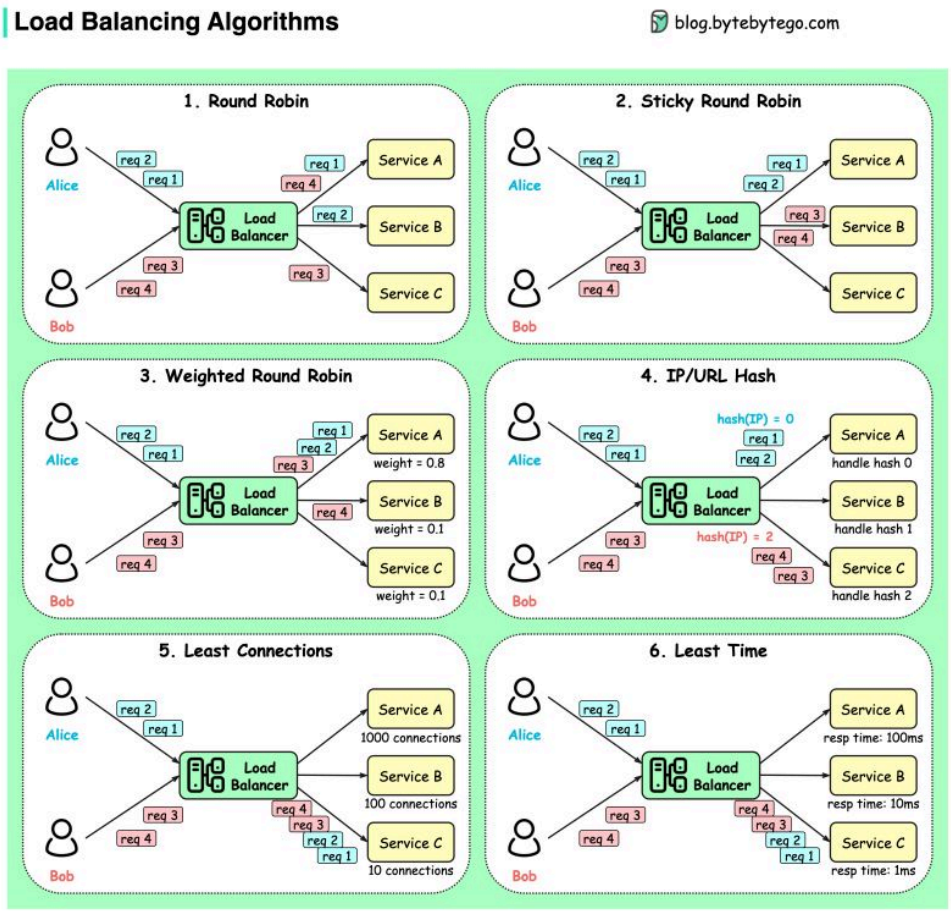
A bytecode language like Java, compiles the source code into bytecode first, then the JVM executes the program. Sometimes the JIT (Just-In-Time) compiler compiles the source code into machine code to speed up the execution. Examples: Java, C#

Interpreted languages are not compiled. They are interpreted by the interpreter during runtime. Examples: Python, Javascript, Ruby

Compiled languages in general run faster than interpreted languages.

Over to you: which type of language do you prefer?

Top 6 Load Balancing Algorithms



- **Static Algorithms**

1. Round robin
   The client requests are sent to different service instances in sequential order. The services are usually required to be stateless.

2. Sticky round-robin
   This is an improvement of the round-robin algorithm. If Alice's first request goes to service A, the following requests go to service A as well.

3. Weighted round-robin
   The admin can specify the weight for each service. The ones with a higher weight handle more requests than others.

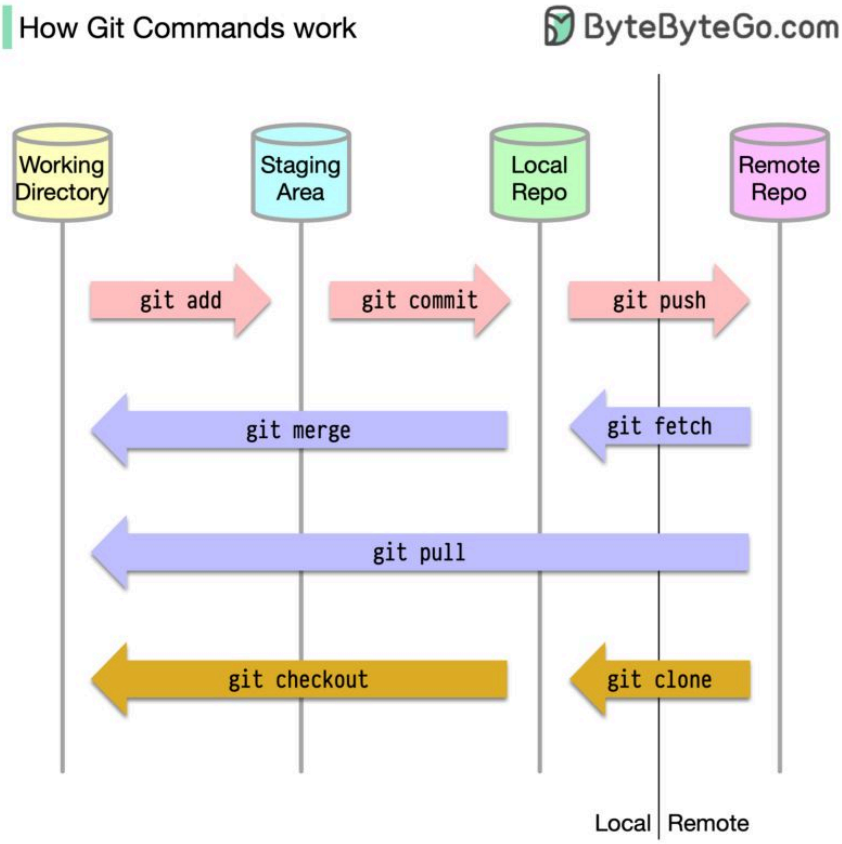
4. Hash
   This algorithm applies a hash function on the incoming requests' IP or URL. The requests are routed to relevant instances based on the hash function result.

- **Dynamic Algorithms**

5. Least connections
   A new request is sent to the service instance with the least concurrent connections.

6. Least response time
   A new request is sent to the service instance with the fastest response time.

How does Git work?



To begin with, it's essential to identify where our code is stored. The common assumption is that there are only two locations - one on a remote server like Github and the other on our local machine. However, this isn't entirely accurate. Git maintains three local storages on our machine, which means that our code can be found in four places:

- Working directory: where we edit files
- Staging area: a temporary location where files are kept for the next commit
- Local repository: contains the code that has been committed
- Remote repository: the remote server that stores the code

Most Git commands primarily move files between these four locations.

Over to you: Do you know which storage location the "git tag" command operates on? This command can add annotations to a commit.

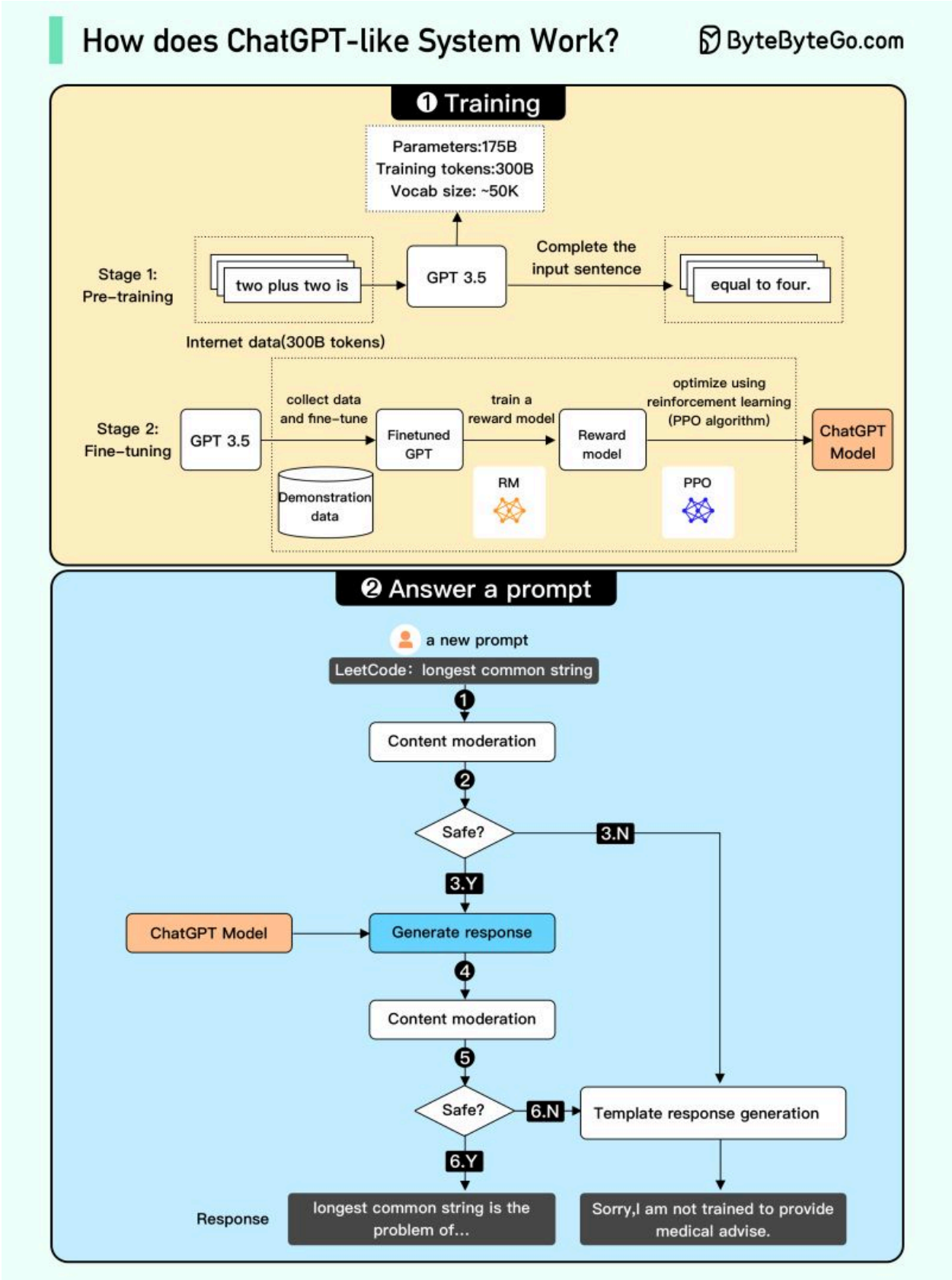
HTTP Cookies Explained With a Simple Diagram

HTTP, the language of the web, is naturally "stateless." But hey, we all want that seamless, continuous browsing experience, right? Enter the unsung heroes - Cookies!

So, here's the scoop in this cookie flyer:

1. HTTP is like a goldfish with no memory - it forgets you instantly! But cookies swoop in to the rescue, adding that "session secret sauce" to your web interactions.
2. Cookies? Think of them as little notes you pass to the web server, saying, "Remember me, please!" And yes, they're stored there, like cherished mementos.
3. Browsers are like cookie bouncers, making sure your cookies don't party crash at the wrong website.
4. Finally, meet the cookie celebrities - SameSite, Name, Value, Secure, Domain, and HttpOnly. They're the cool kids setting the rules in the cookie jar!

How does a ChatGPT-like system work?



We attempted to explain how it works in the diagram below. The process can be broken down into two parts.

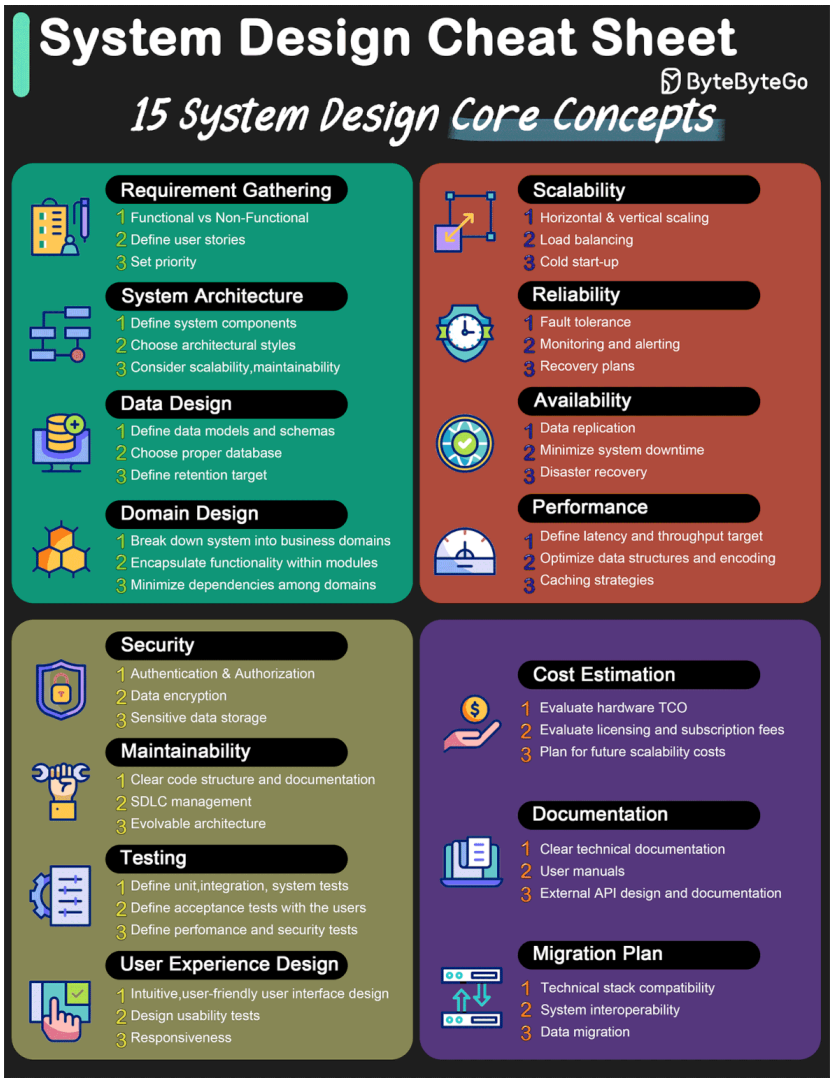
1. Training. To train a ChatGPT model, there are two stages:
    - Pre-training: In this stage, we train a GPT model (decoder-only transformer) on a large chunk of internet data. The objective is to train a model that can predict future words given a sentence in a way that is grammatically correct and semantically meaningful similar to the internet data. After the pre-training stage, the model can complete given sentences, but it is not capable of responding to questions.
    - Fine-tuning: This stage is a 3-step process that turns the pre-trained model into a question-answering ChatGPT model:

        1. Collect training data (questions and answers), and fine-tune the pre-trained model on this data. The model takes a question as input and learns to generate an answer similar to the training data.
        2. Collect more data (question, several answers) and train a reward model to rank these answers from most relevant to least relevant.
        3. Use reinforcement learning (PPO optimization) to fine-tune the model so the model's answers are more accurate.

2. Answer a prompt
    - Step 1: The user enters the full question, “Explain how a classification algorithm works”.
    - Step 2: The question is sent to a content moderation component. This component ensures that the question does not violate safety guidelines and filters inappropriate questions.
    - Steps 3-4: If the input passes content moderation, it is sent to the chatGPT model. If the input doesn’t pass content moderation, it goes straight to template response generation.
    - Step 5-6: Once the model generates the response, it is sent to a content moderation component again. This ensures the generated response is safe, harmless, unbiased, etc.
    - Step 7: If the input passes content moderation, it is shown to the user. If the input doesn’t pass content moderation, it goes to template response generation and shows a template answer to the user.

A cheat sheet for system designs

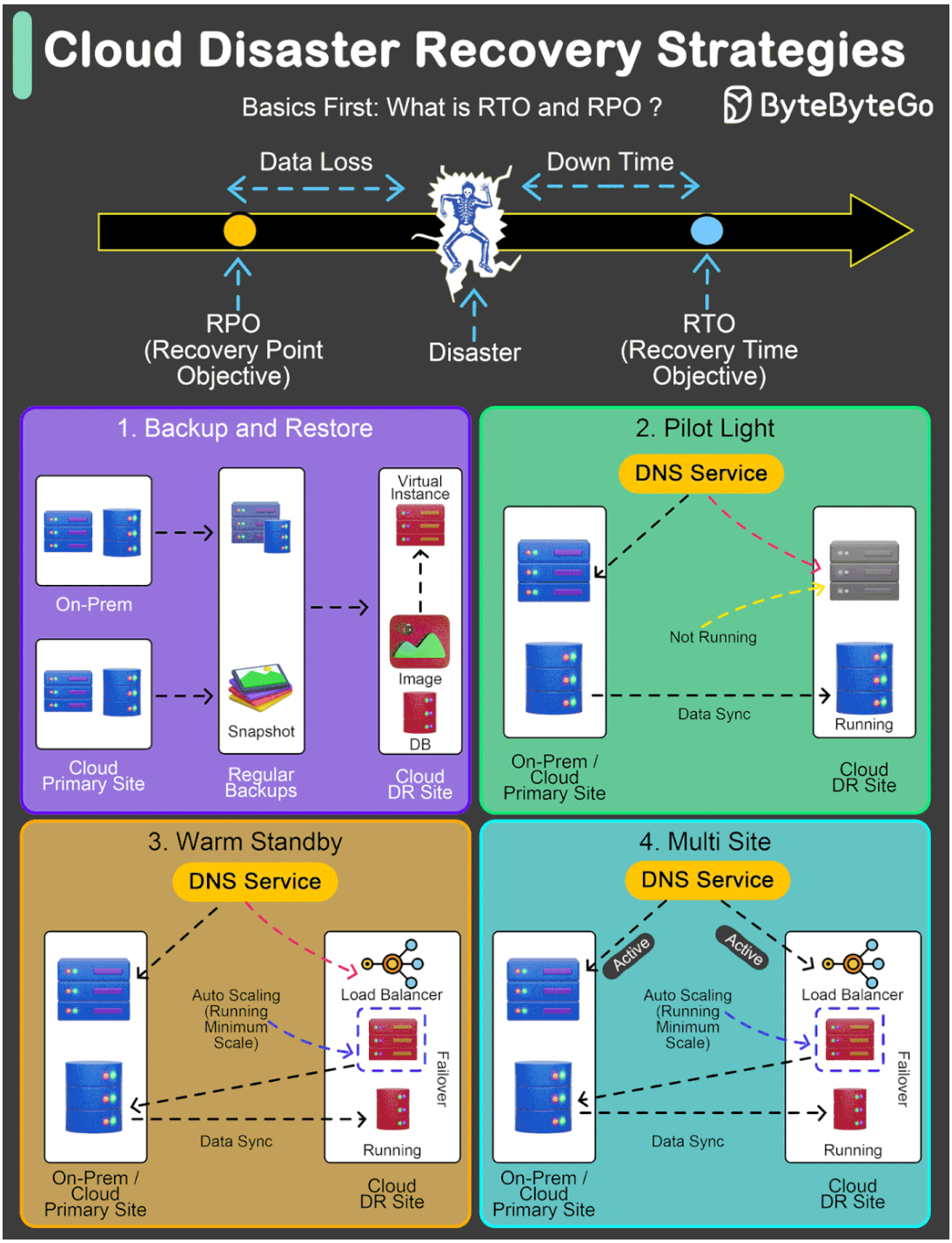
The diagram below lists 15 core concepts when we design systems. The cheat sheet is straightforward to go through one by one. Save it for future reference!



- Requirement gathering
- System architecture
- Data design
- Domain design
- Scalability
- Reliability
- Availability
- Performance
- Security
- Maintainability
- Testing

- User experience design
- Cost estimation
- Documentation
- Migration plan

Cloud Disaster Recovery Strategies



An effective Disaster Recovery (DR) plan is not just a precaution; it's a necessity.

The key to any robust DR strategy lies in understanding and setting two pivotal benchmarks: Recovery Time Objective (RTO) and Recovery Point Objective (RPO).

- Recovery Time Objective (RTO) refers to the maximum acceptable length of time that your application or network can be offline after a disaster.
- Recovery Point Objective (RPO), on the other hand, indicates the maximum acceptable amount of data loss measured in time.

Let's explore four widely adopted DR strategies:

1. Backup and Restore Strategy:

   This method involves regular backups of data and systems to facilitate post-disaster recovery.
   - Typical RTO: From several hours to a few days.
   - Typical RPO: From a few hours up to the time of the last successful backup.

2. Pilot Light Approach:

   Maintains crucial components in a ready-to-activate mode, enabling rapid scaling in response to a disaster.
   - Typical RTO: From a few minutes to several hours.
   - Typical RPO: Depends on how often data is synchronized.

3. Warm Standby Solution:

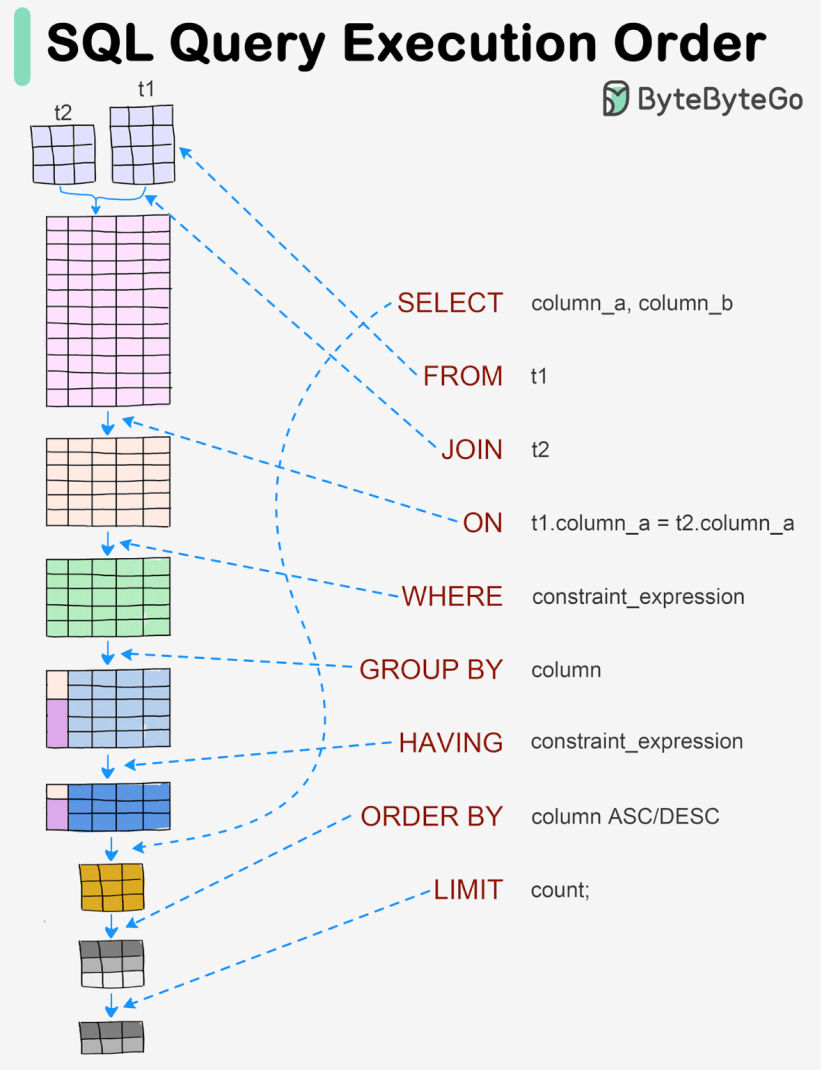
   Establishes a semi-active environment with current data to reduce recovery time.
   - Typical RTO: Generally within a few minutes to hours.
   - Typical RPO: Up to the last few minutes or hours.

4. Hot Site / Multi-Site Configuration:

   Ensures a fully operational, duplicate environment that runs parallel to the primary system.
   - Typical RTO: Almost immediate, often just a few minutes.
   - Typical RPO: Extremely minimal, usually only a few seconds old.

Over to you: What factors would influence your decision to choose a DR strategy?

Visualizing a SQL query



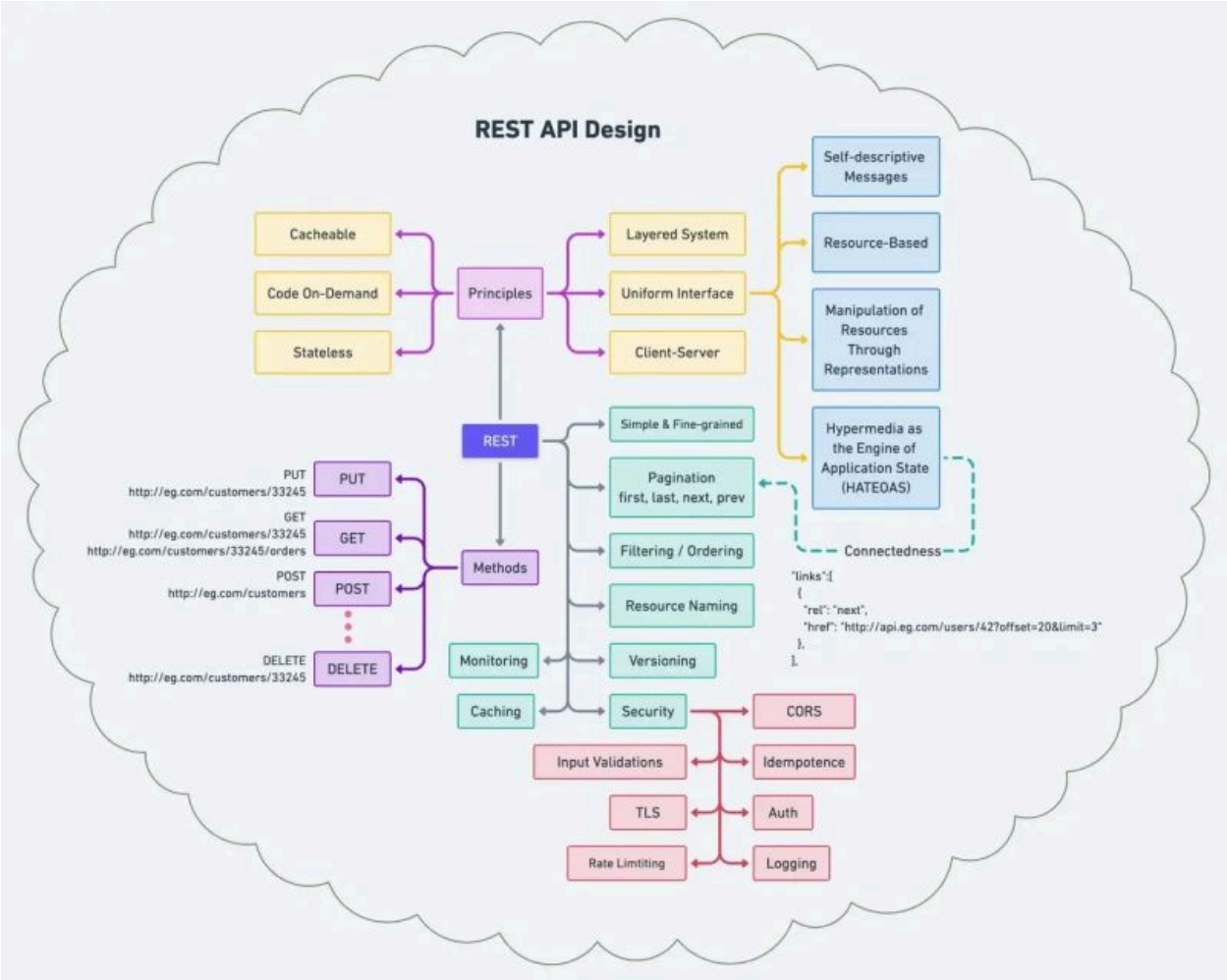
SQL statements are executed by the database system in several steps, including:

- Parsing the SQL statement and checking its validity
- Transforming the SQL into an internal representation, such as relational algebra
- Optimizing the internal representation and creating an execution plan that utilizes index information
- Executing the plan and returning the results

How does REST API work?

What are its principles, methods, constraints, and best practices?

I hope the diagram below gives you a quick overview.



Explaining 9 types of API testing