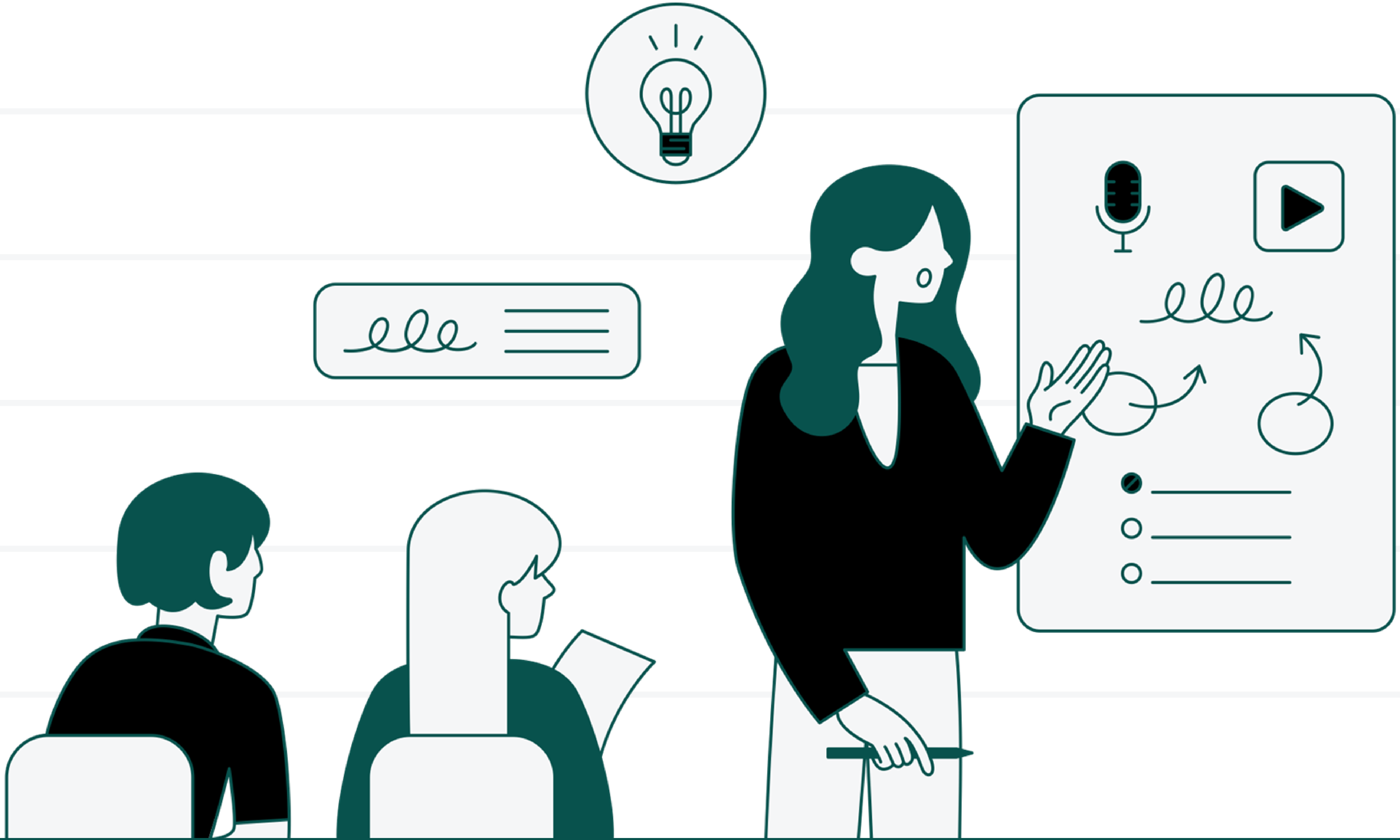
Software Design

Department of Computer Engineering
Sharif University of Technology
Maryam Ramezani
maryam.ramezani@sharif.edu



01

Introduction

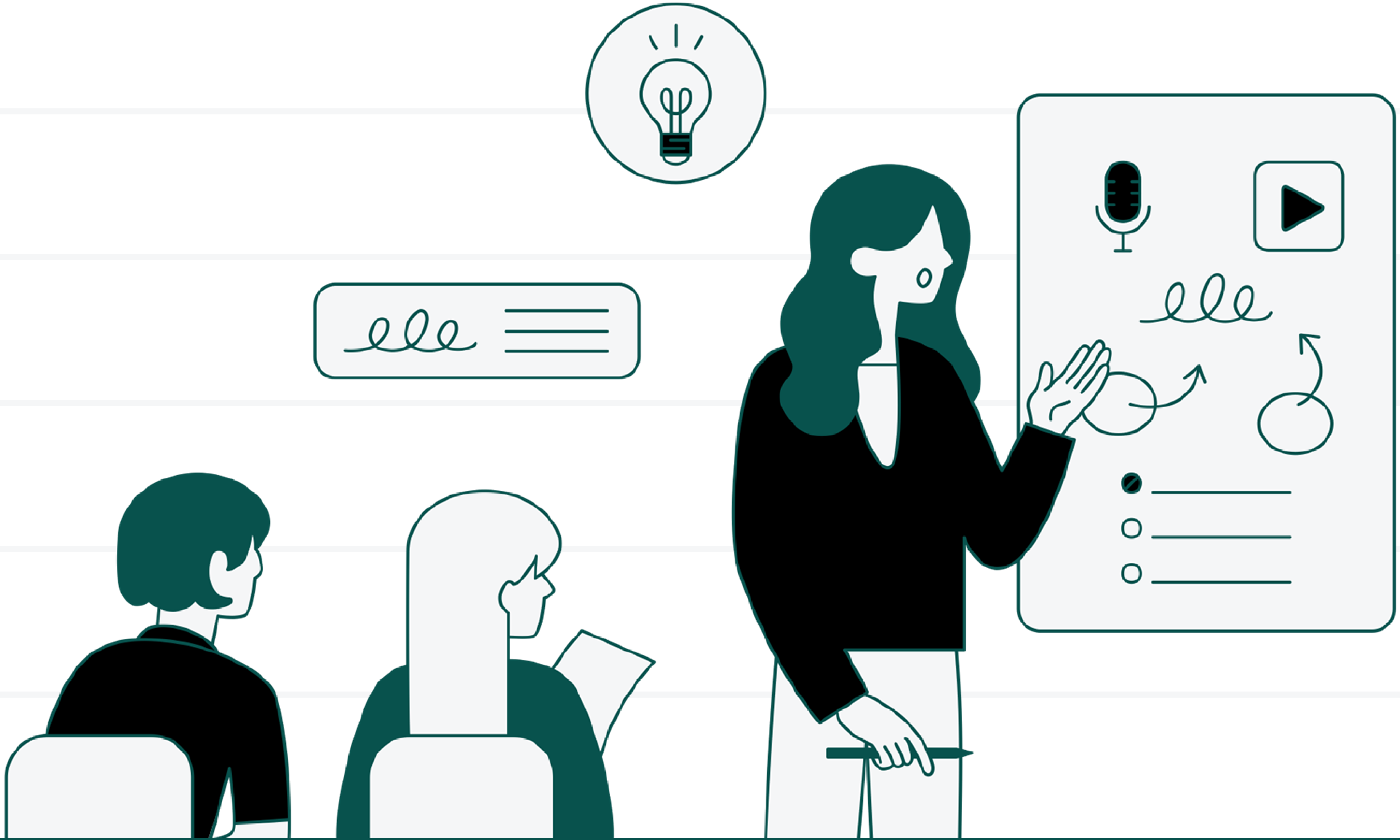
Types of Software and their Differences

- Classification 1 of the types of software
  - Custom software: developed to meet the specific needs of a particular customer and tends to be of little use to others. e.g. web sites, air-traffic control systems
  - Generic software/ (COTS)/ shrink-wrapped software: is designed to be sold on the open market, to perform functions that many people need, and to run on general purpose computers. e.g. word processors, spreadsheets, web browsers, computer games.
  - Embedded software: run specific hardware devices which are typically sold on the open market. e.g. washing machines, DVD players, and automobiles
- It is possible to take generic software and customize it and vise versa.

Type of software	**condition in market**
Embedded S/W	Highest number of copies in use
Generic S/W	Highest number of copies in use on general-purpose computers
Custom S/W	What most developers work on

Types of Software and their Differences -continue

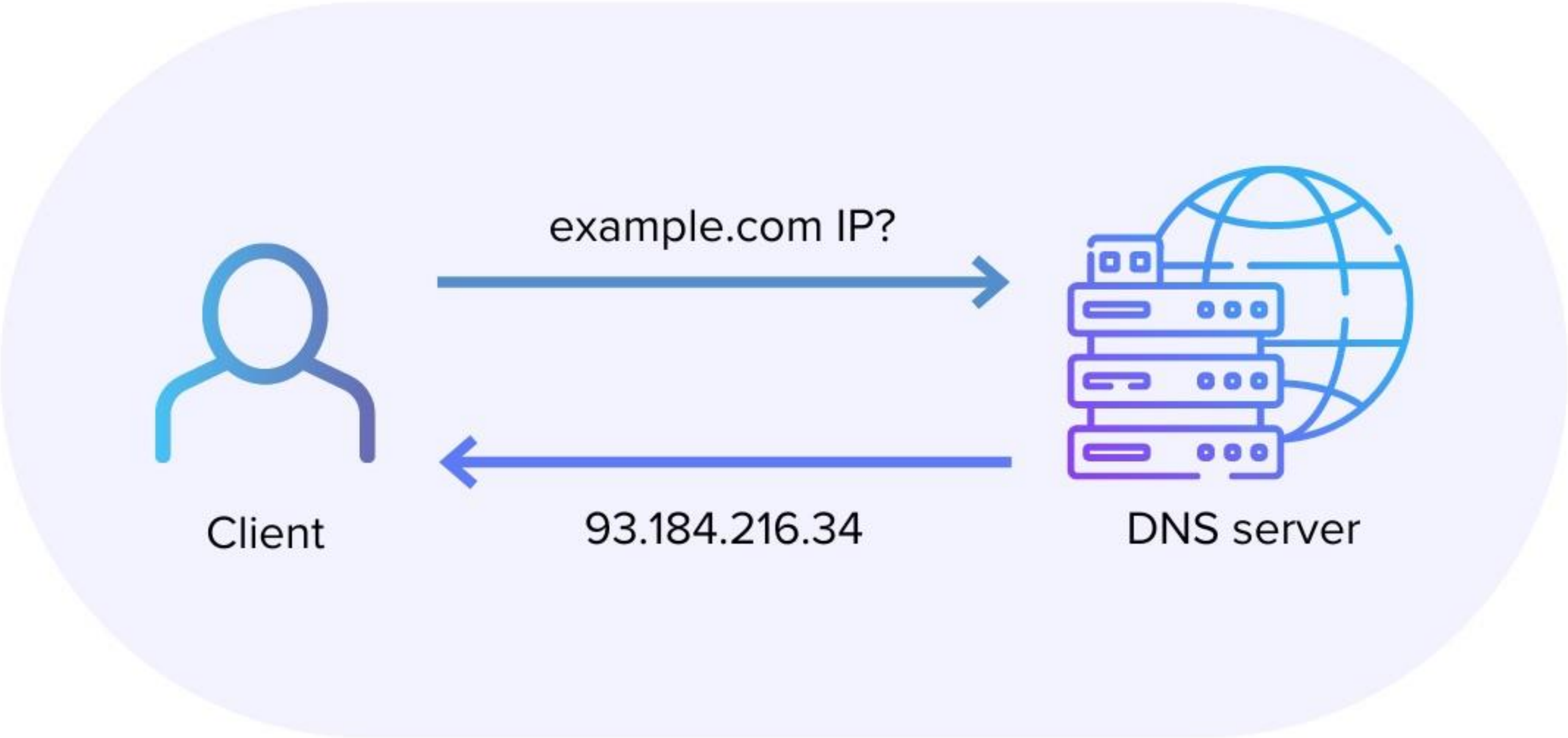
- Classification 2 of the types of software
  - Real-time software:
    - it has to react immediately (i.e. in real time) to stimuli from the environment (e.g. the pushing of a button, a signal from a sensor)
    - Responsiveness must always be guaranteed- safety is a key concern in their design
    - e.g. many of the embedded systems, custom systems that run industrial plants and telephone networks
  - Data processing software:
    - is used to run businesses.
    - It performs functions such as recording sales, managing accounts, printing bills etc.
    - The design concern here is how to organize the data and provide useful information gathered to the users so they can perform their work effectively
    - Accuracy and security of data are of major concern
    - In traditional data processing tasks, data is gathered together in batches to be processed later.
  - Some software has both real-time and data processing aspects.



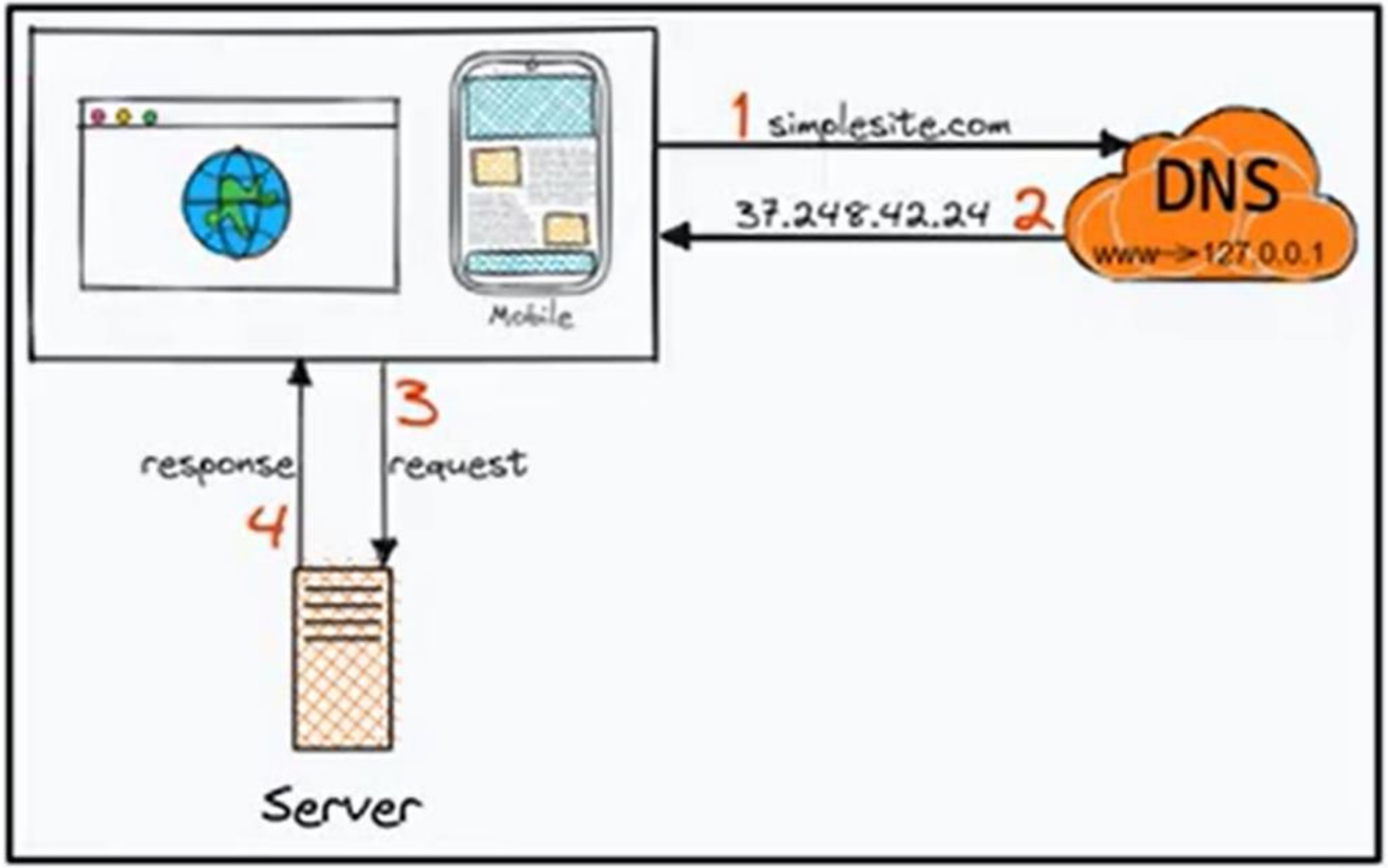
02 Domain Name System

DNS

- phonebook of the Internet



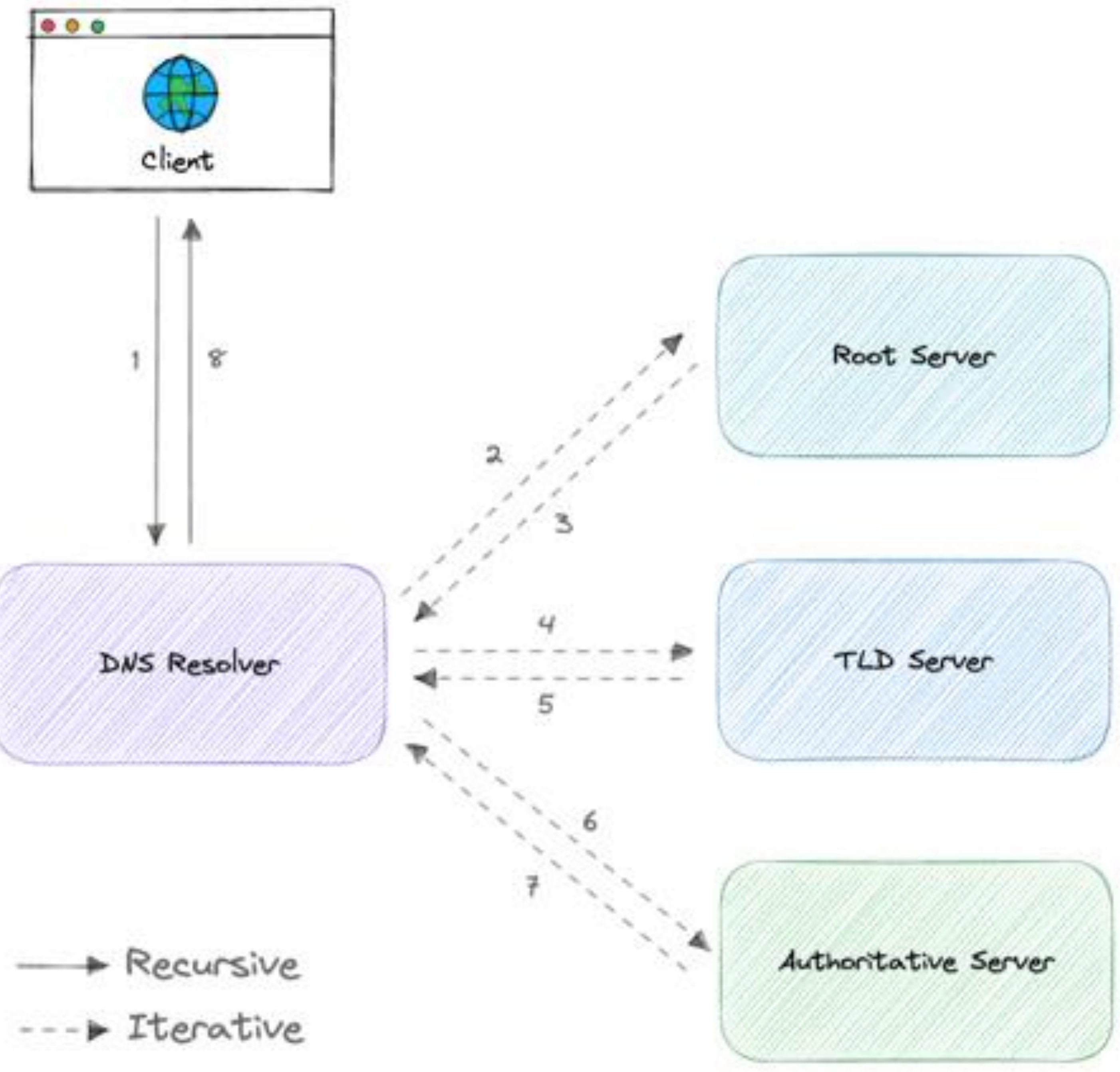
App-DNS-Server



How DNS works

Can have cache

Can have cache



How DNS works

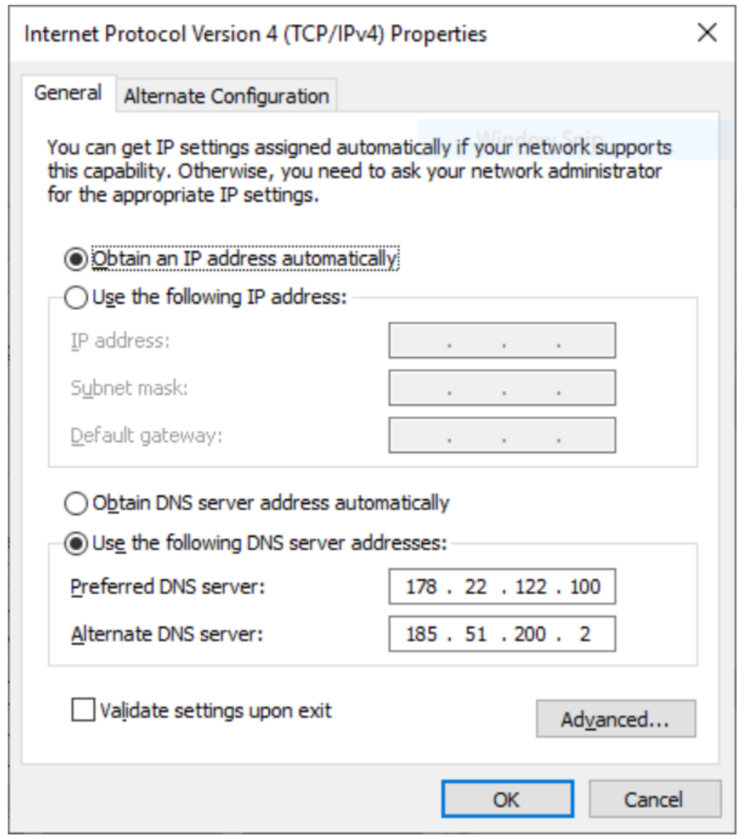
1. A client types example.com into a web browser,

   1-1 if browser or OS has cached the IP it will use it.

   1-2 Else: the query travels to the internet and is received by a DNS resolver.

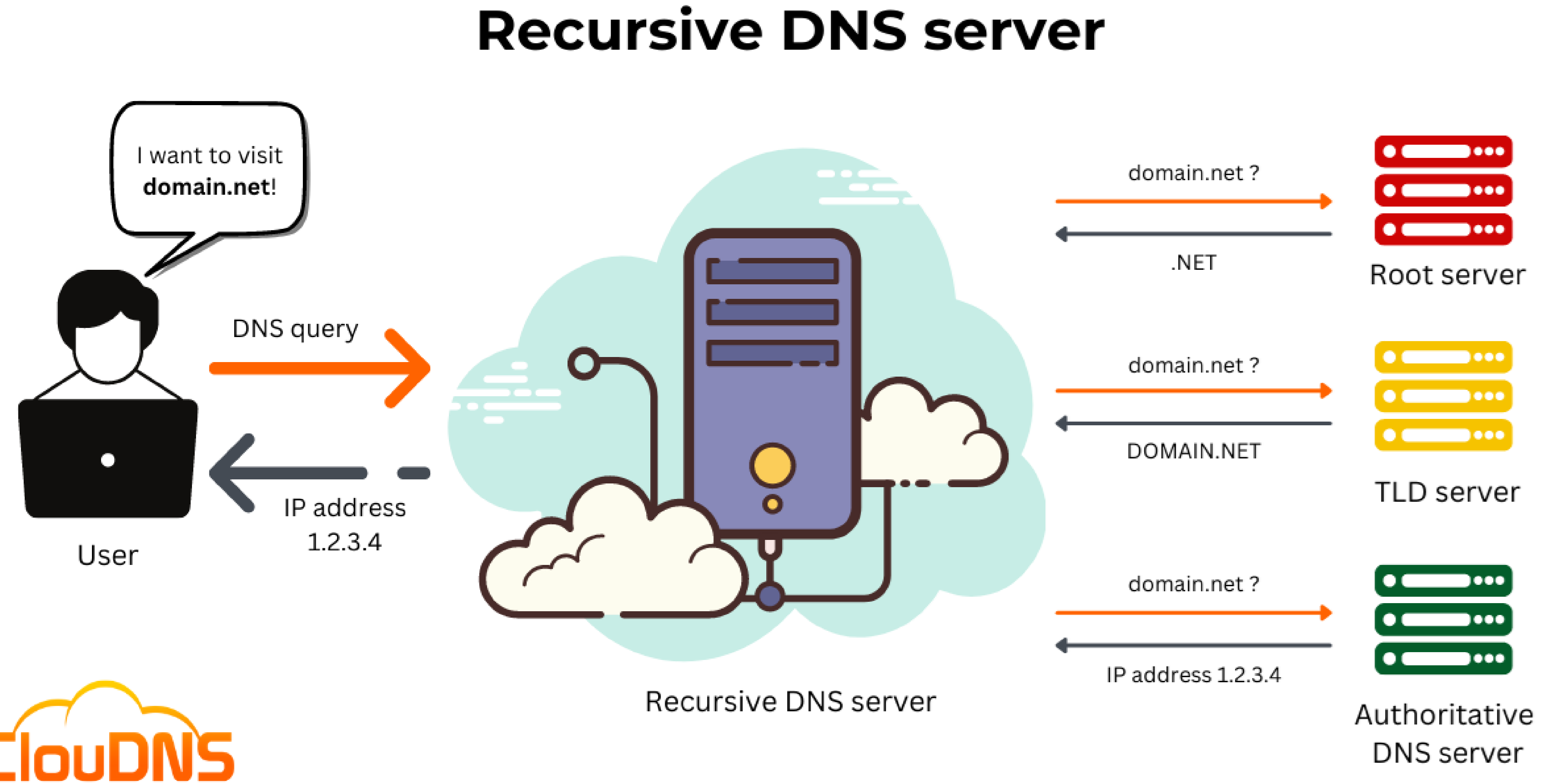
   Note: Who is DNS resolver?

   - ❑ Internet service provider (ISP) like Parsonline, Shatel , MCI, …
   - ❑ Who you have set in your OS like Google DNS: 8.8.8.8, Shecan!!!!

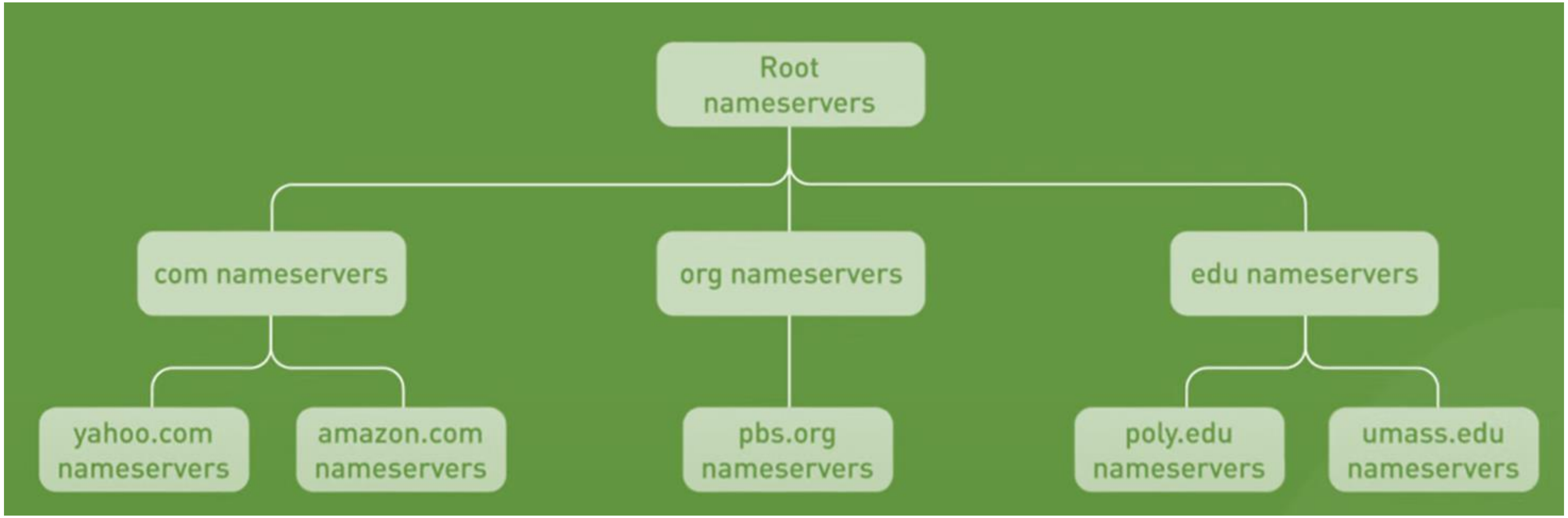


DNS Resolver

- Get Name and Give IP



- If it has cached the IP, will return it else it will send the name to Root Server!


Root nameservers
com nameservers
org nameservers
edu nameservers
yahoo.com nameservers
amazon.com nameservers
pbs.org nameservers
poly.edu nameservers
umass.edu nameservers

Root Server

- Like a Boss in company who knows the received request is related to each department!!
- Root server knows the related department! TLD Server based on the domain suffix like .com, .ir, .net, …

TLD Server

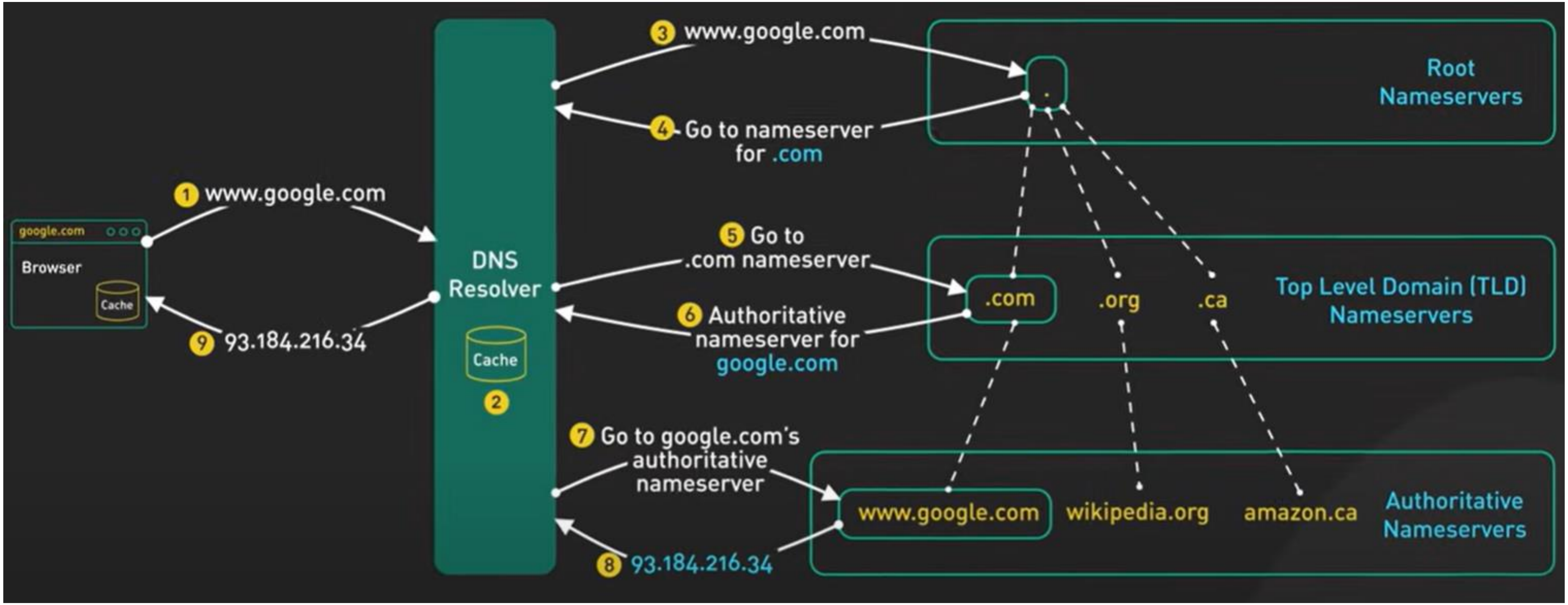
- Top-Level Domain
- Like a department manager who knows which employee in department named Authoritative Server knows the IP related to received request.

Authoritative Server

- Authoritative DNS information (DNS records) from its own store.
- In case it doesn’t know the answer, it is going to direct to another nameserver. For instance, the Root name server points to the responsible TLD (Top-Level Domain) server.
- An authoritative NXDOMAIN. It replies that the requested domain name doesn’t exist.
- An authoritative empty NOERROR (NODATA) answer. The requested domain name exists, but the particular queried DNS record does not.

CLOUDFLARE

Overview



3 www.google.com
4 Go to nameserver for .com
Root Nameservers
1 www.google.com
google.com
Browser
Cache
DNS Resolver
Cache
2
9 93.184.216.34
5 Go to .com nameserver
6 Authoritative nameserver for google.com
.com
.org
.ca
Top Level Domain (TLD) Nameservers
7 Go to google.com's authoritative nameserver
8 93.184.216.34
www.google.com
wikipedia.org
amazon.ca
Authoritative Nameservers

Conclusion

1. A client types example.com into a web browser,

   1-1 if browser or OS has cached the IP it will use it.

   1-2 Else: the query travels to the internet and is received by a DNS resolver.

   Note: Who is DNS resolver: Internet service provider (ISP) like Parsonline, Shatel , MCI, ...
2. The resolver then recursively queries a DNS root nameserver.
3. The root server responds to the resolver with the address of a Top Level Domain (TLD).
4. The resolver then makes a request to the .com TLD.
5. The TLD server then responds with the IP address of the domain's nameserver, example.com.
6. Lastly, the recursive resolver sends a query to the domain's nameserver.
7. The IP address for example.com is then returned to the resolver from the nameserver.
8. The DNS resolver then responds to the web browser with the IP address of the domain requested initially.

Common DNS Record Types

- 1. A (Address) Record: Maps a domain to an IPv4 address.

TTL (Time To Live): This value determines how long a DNS record remains in the cache (temporary storage). For faster changes, you can reduce the TTL, but for better performance, you can set it to **Automatic**.

Field	Value
Type	A
Name	sad
Value	203.0.113.10
TTL	Automatic

Common DNS Record Types

- 2. AAAA Record: Maps a domain to an IPv6 address.

Field	Value
Type	AAAA
Name	sad
Value	2001:0db8:85a3:0000:0000:8a2e:0370:7334
TTL	Automatic

Common DNS Record Types

- 3. CNAME (Canonical Name) Record: Creates an alias for another domain name.

Field	Value
Type	CNAME
Name	www.sad
Value	sad.sharif.edu
TTL	Automatic

This record makes www.sad.sharif.edu an alias for sad.sharif.edu. Any request to www.sad.sharif.edu will be directed to sad.sharif.edu, simplifying DNS management.

Common DNS Record Types

- 4. NS (Name Server) Record: Specifies the authoritative DNS servers for the domain.

Field	Value
Type	NS
Name	sharif.edu
Value	ns1.sharif.edu, ns2.sharif.edu

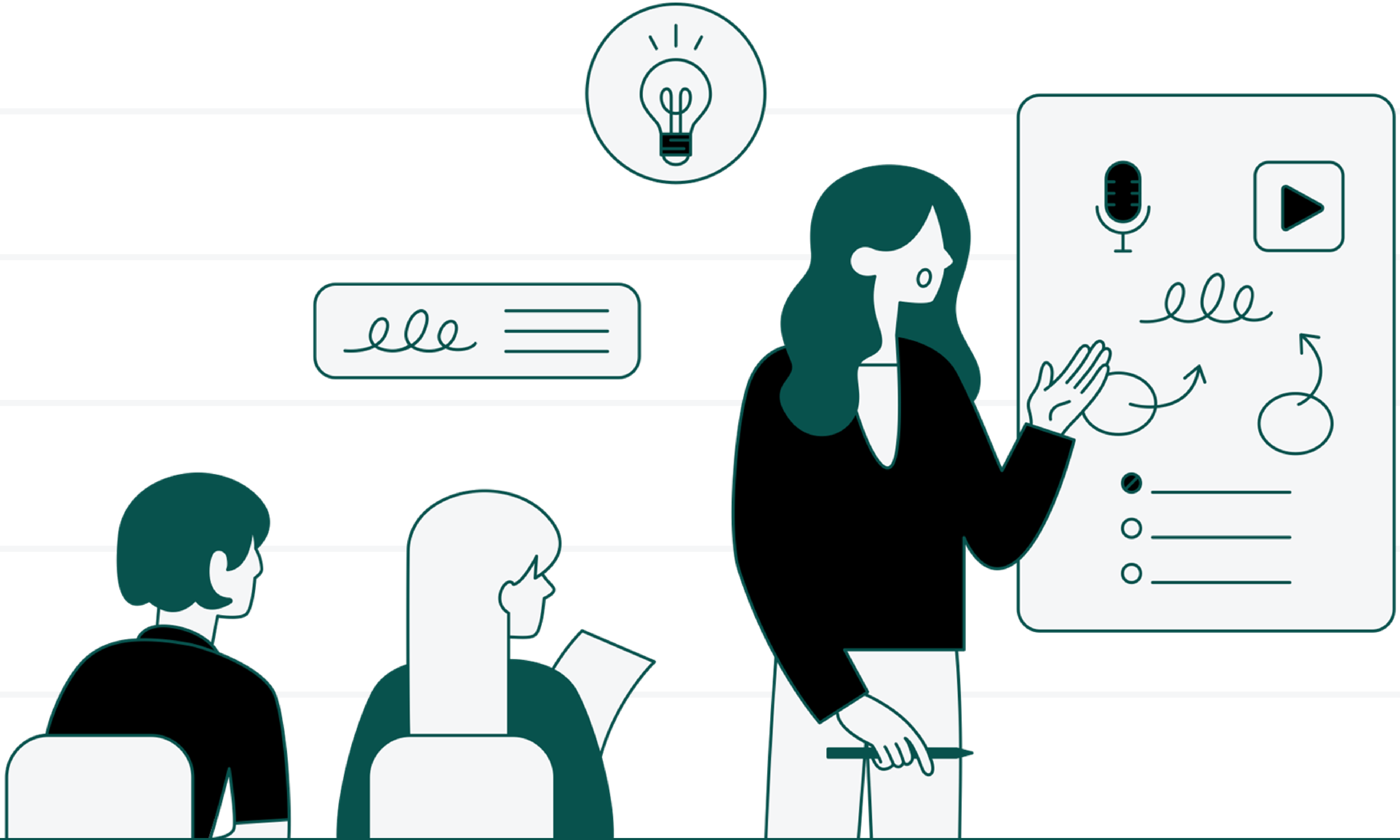
These records define ns1.sharif.edu and ns2.sharif.edu as the authoritative DNS servers responsible for managing DNS records for sharif.edu and its subdomains like sad.sharif.edu.

نوع	CDN	عنوان	مقدار	مقدار اعتبار
TXT	غیرفعال	aicloud.ir	v=spf۱ in"...	۲
NS	غیرفعال	aicloud.ir	.ns۱.xaas-cdn.com	۶۰
NS	غیرفعال	aicloud.ir	.ns۲.xaas-cdn.com	۶۰
NS	غیرفعال	aicloud.ir	.ns۳.xaas-cdn.com	۶۰

Common DNS Record Types

- 5. TXT (Text) Record: Stores text information for various purposes such as domain verification.

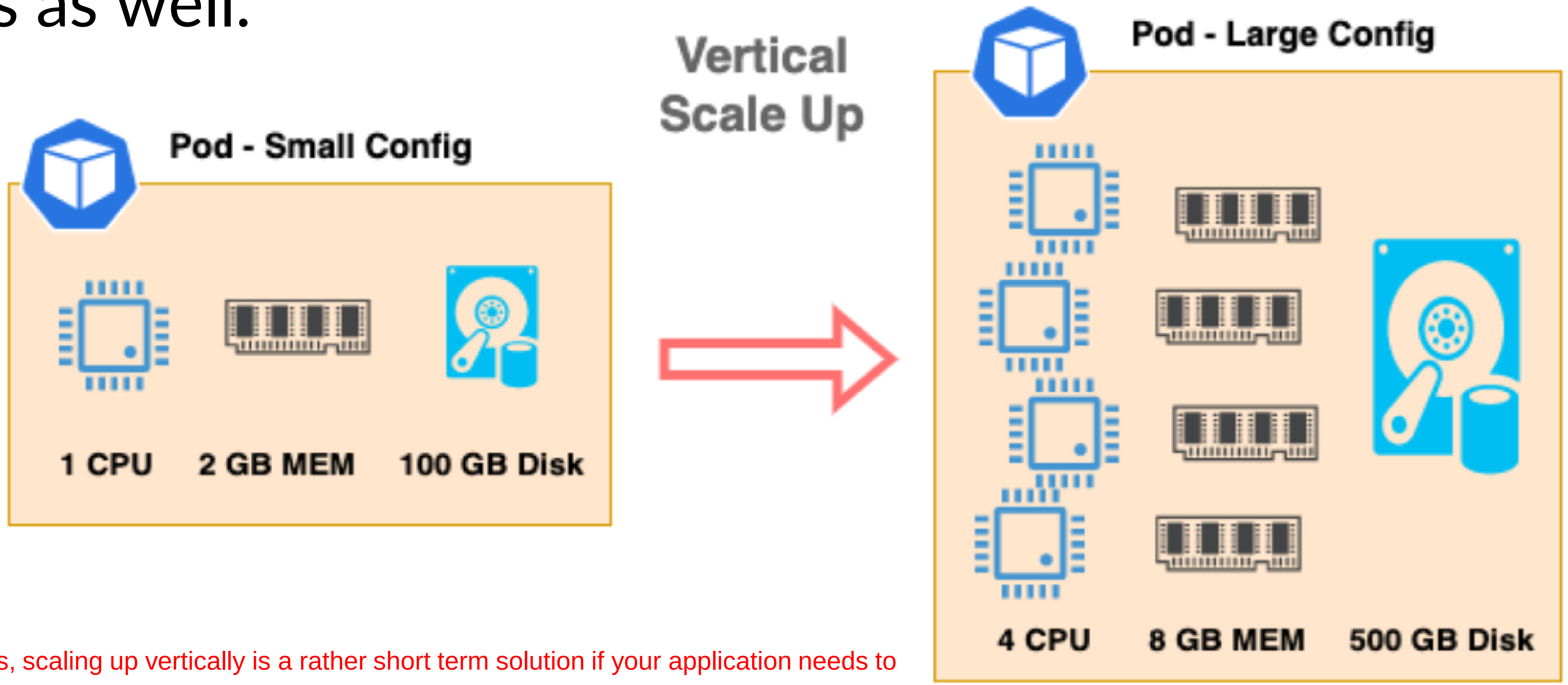
Field	Value
Type	TXT
Name	sad
Value	"v=spf1 include:sharif.edu ~all"
TTL	Automatic



03 Scaling

Scaling Up (Vertical Scaling)

- Scaling up (or vertical scaling) is adding more resources—like CPU, memory, and disk—to increase more compute power and storage capacity. This term applies to traditional applications deployed on physical servers or virtual machines as well.



With physical hardware limitations, scaling up vertically is a rather short term solution if your application needs to continue growing.

Scaling Up (Vertical Scaling)

- Advantages
  - It is simple and straightforward. For the applications with more traditional and monolithic architecture, it is much simpler to just add more compute resources to scale.
  - You can take advantage of powerful server hardware.

Scaling Up (Vertical Scaling)

- Disadvantages
  - Scaling up has limits. Even with today's powerful servers, as you continue to add compute resources to your application pod, you will still hit the physical hardware limitations sooner or later.
  - Down Time!
  - Bottlenecks develop in compute resources. As you add compute resources to a physical server, it is difficult to increase and balance the performance linearly for all the components, and you will most likely hit a bottleneck somewhere. For example, initially your server has a memory bottleneck with 100% usage of memory and 70% usage of CPU. After doubling the number of DIMMs, now you have 100% of CPU usage vs 80% of memory usage.
  - It may cost more to host applications. Usually the larger servers with high compute power cost more. If your application requires high compute resources, using these high-cost larger servers may be the only choice.

Design

- Design is the process of deciding how the requirements should be implemented using the available technology
- Some of the important activities during design: system engineering, determining the software architecture, detailed designs, user interface design, etc.
- For large systems, software engineers work on architectural design in conjunction with high-level requirements to effectively divide the system into subsystems
- For small systems, requirement precede design to avoid re-doing the design if requirements change

Scaling Out (Horizontal Scaling)

- Scaling out (or horizontal scaling) addresses some of the limitations of the scale up method. With horizontal scaling, the compute resource limitations from physical hardware are no longer the issue. In fact, you can use any reasonable size of server as long as the server has enough resources to run the pods

Scaling Out (Horizontal Scaling)

- **Advantages**
- **It delivers long-term scalability.** The incremental nature of scaling out allows you to scale your application for expected and long-term growth.
- **Scaling back is easy.** Your application can easily scale back by reducing the number of pods when the load is low. This frees up compute resources for other applications.
- **You can utilize commodity servers.** Normally, you don’t need large servers to run containerized applications. Since application pods scale horizontally, servers can be added as needed.

Scaling Out (Horizontal Scaling)

- **Disadvantages**
- **It may require re-architecting.** You will need to re-architect your application if your application is using monolithic architecture(s).

Which One Is Best: Scale-out or Scale-up?

- The answer depends on your particular needs and resources. Here are some questions to think about:

  - Are your needs long term or short term?
  - What's your budget? Is it big or small?
  - What type of workloads are you dealing with?
  - Are you dealing with a temporary traffic peak or constant traffic overload?

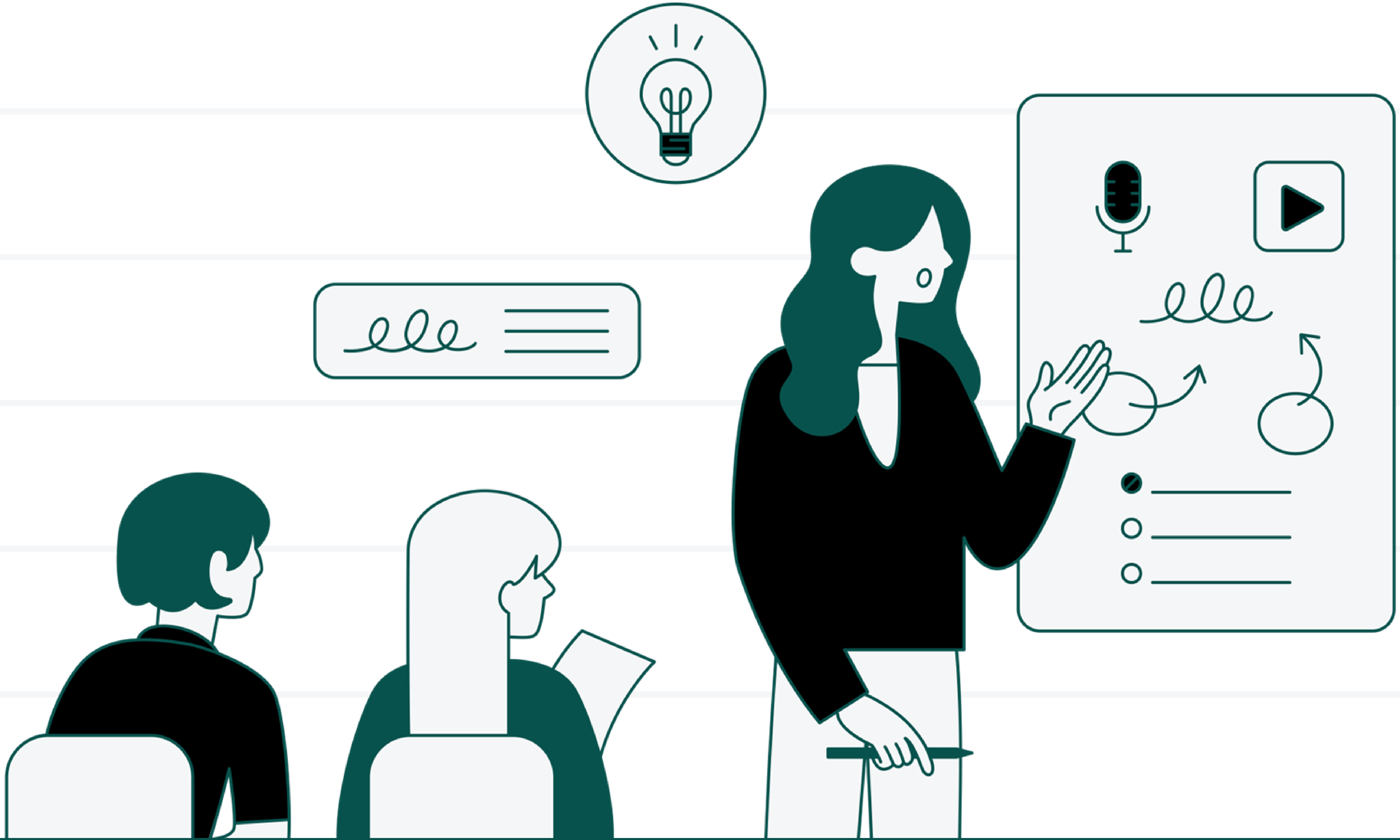
Which One Is Best: Scale-out or Scale-up?

- Once you've answered those questions, consider these factors:
  - Cost: Horizontal scaling is more expensive, at least initially, so if your budget is tight, then scaling up might be the best choice.
  - Reliability: Horizontal scaling is typically far more reliable than vertical scaling. If you're handling a high volume of transactional data or sensitive data, for example, and your downtime costs are high, you should probably opt for scaling out.
  - Geographic distribution: If you have, or plan to have, global clients, you'll be much better able to maintain your **SLA**s via scaling out since a single machine in a single location won't be enough for customers to access your services.
  - Future-proofing: Because scaling up uses a single node, it's tough to future-proof a vertical scaling-based architecture. With scaling out, it's much easier to increase the overall performance threshold of your organization by adding machines. If you're planning for the long term and operate in a highly competitive industry with lots of potential disruptors, scaling out would be the best option.

Which One Is Best: Scale-out or Scale-up?

- In short:

  - If you have a bigger budget and expect a steady and large growth in data over a long period of time and need to distribute an overstrained storage workload across several storage nodes, scaling out is the best option.
  - If you haven't yet maxed out the full potential of your current infrastructure and can still add CPUs and memory resources to it and you don't anticipate a meaningfully large growth in your data set over the next three to five years, then scaling up would likely be the best choice.

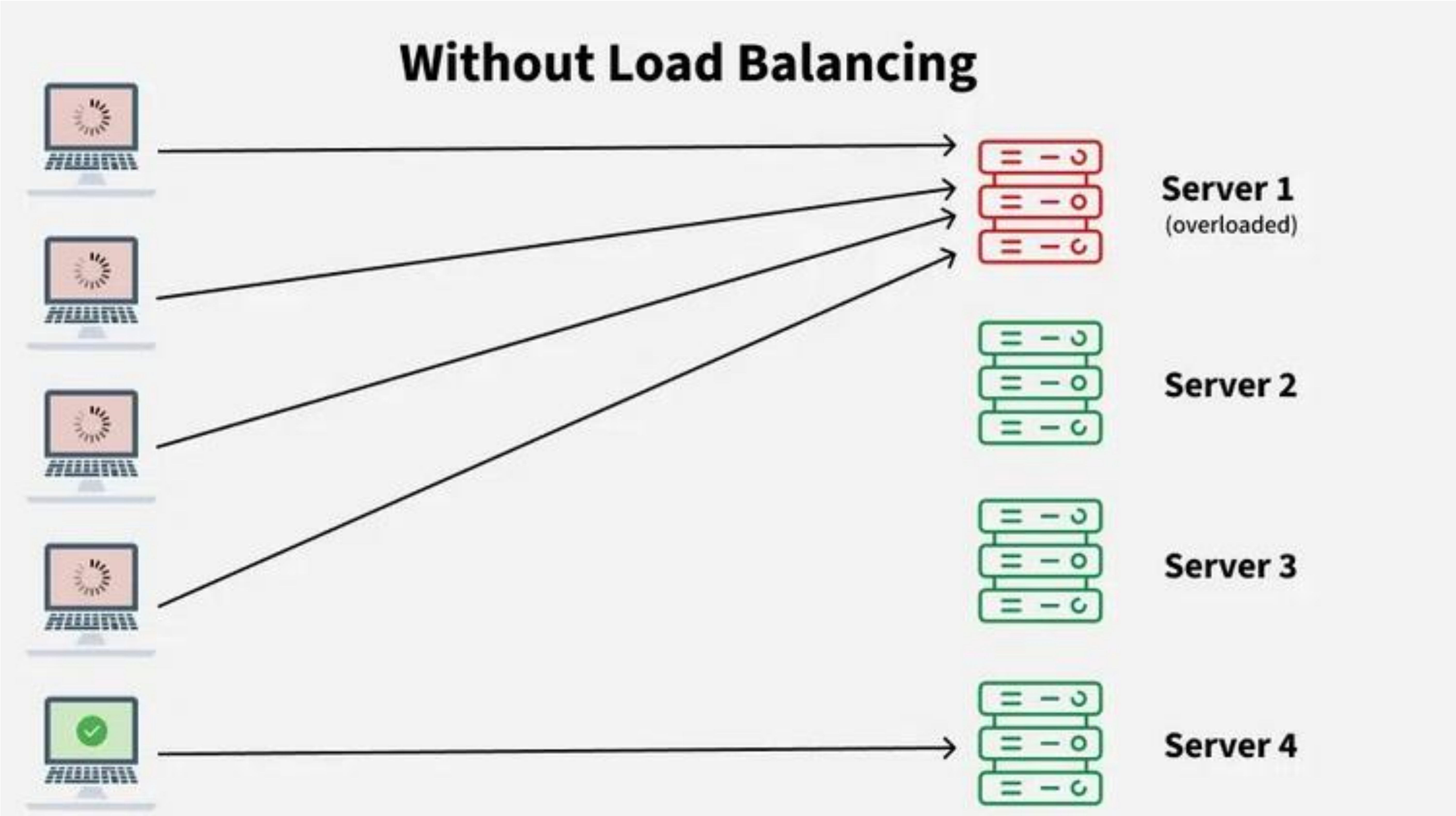


04 Load Balancing

What is a Load Balancer?

- A load balancer is a networking device or software application that distributes and balances the incoming traffic among the servers to provide high availability, efficient utilization of servers, and high performance. A load balancer works as a “traffic cop” sitting in front of your server and routing client requests across all servers
  - Load balancers are highly used in cloud computing domains, data centers, and large-scale web applications where traffic flow needs to be managed.
  - It simply distributes the set of requested operations effectively across multiple servers and ensures that no single server bears too many requests.

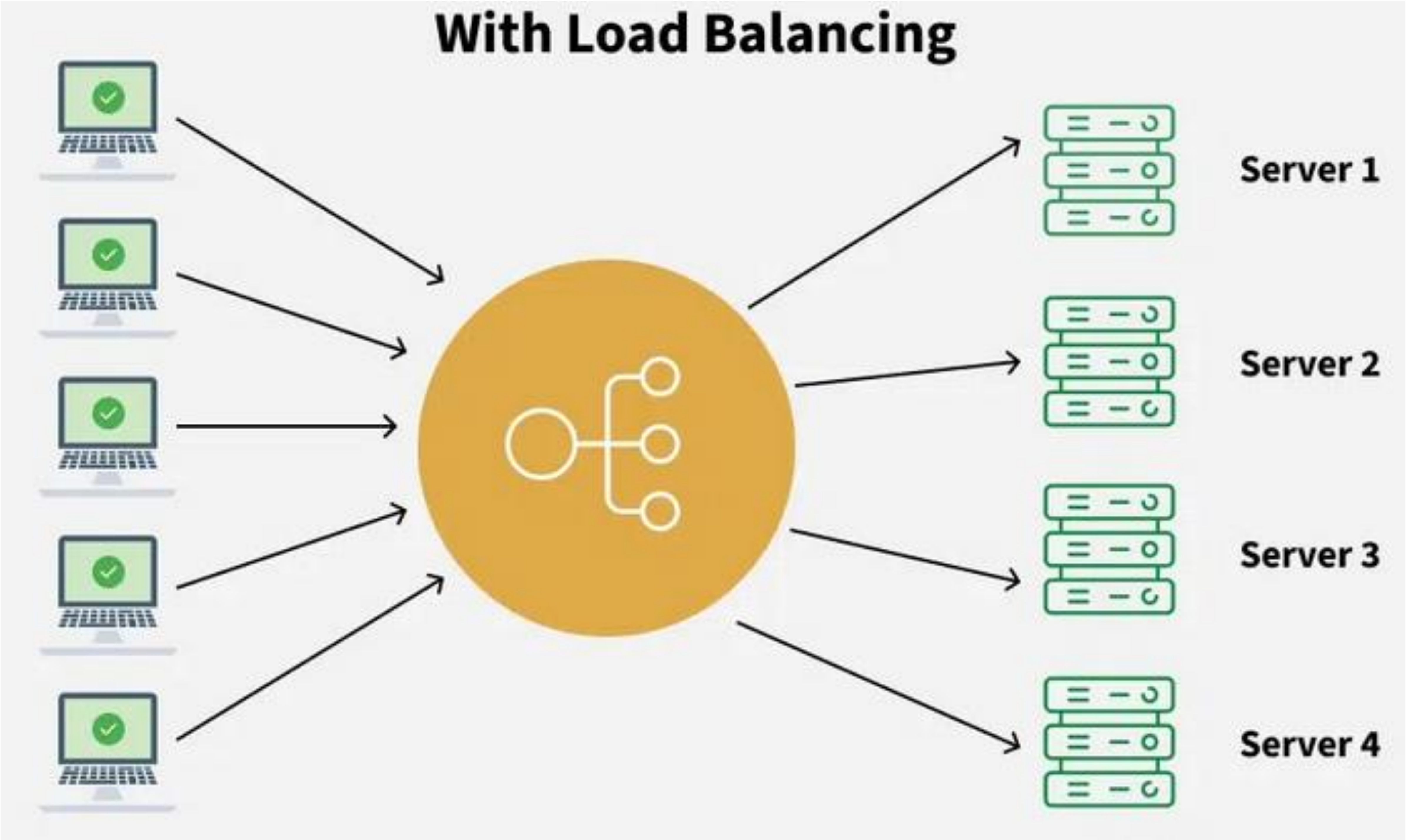
What will happen if there is NO Load Balancer?



Problems

- **Single Point of Failure:**
  - If the server goes down or something happens to the server the whole application will be interrupted and it will become unavailable for the users for a certain period. It will create a bad experience for users which is unacceptable for service providers.
- **Overloaded Servers:**
  - There will be a limitation on the number of requests that a web server can handle. If the business grows and the number of requests increases the server will be overloaded.
- **Limited Scalability**:
  - Without a load balancer, adding more servers to share the traffic is complicated. All requests are stuck with one server, and adding new servers won't automatically solve the load issue.

What will happen if there is Load Balancer?

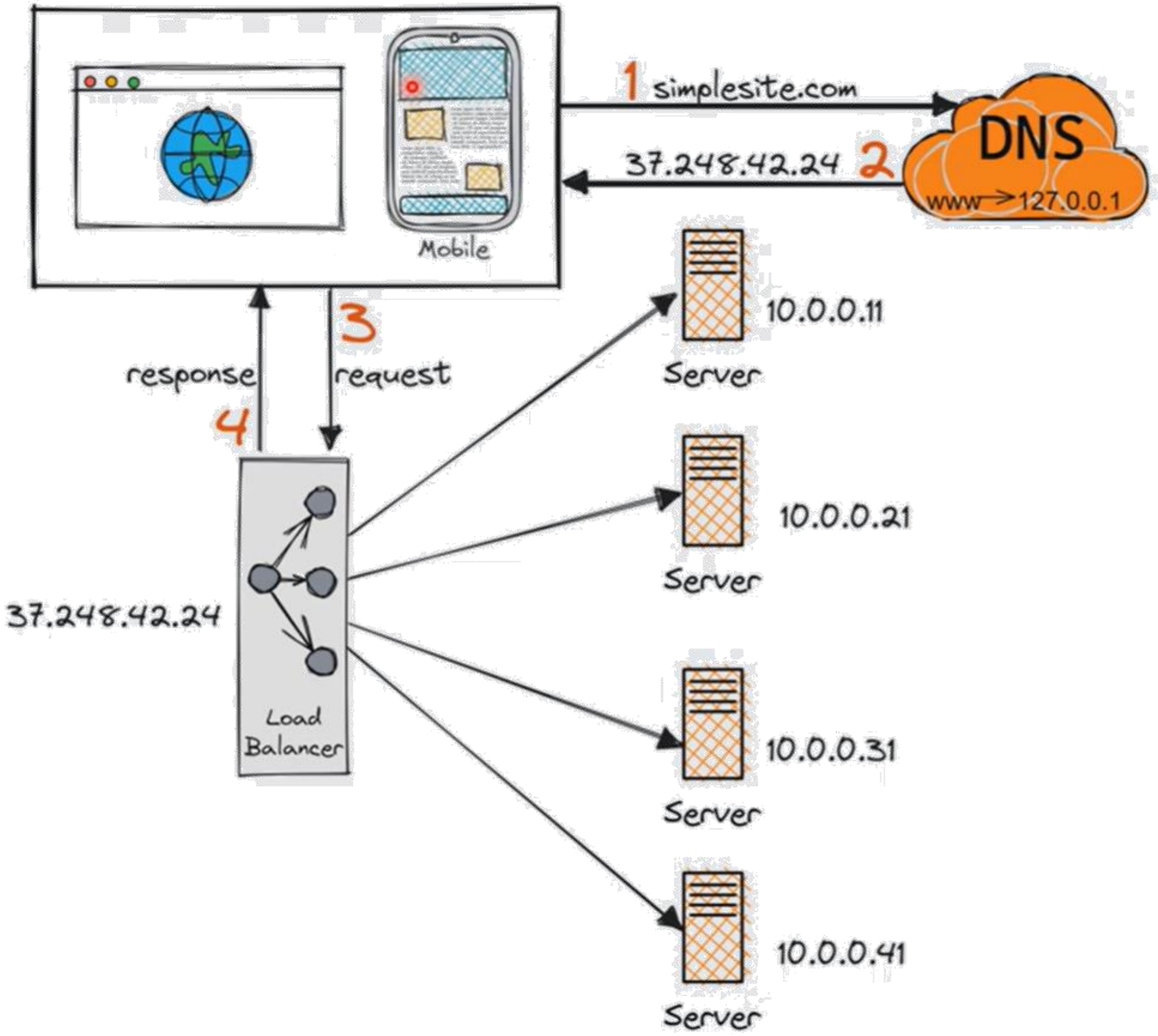


Key characteristics of Load Balancers

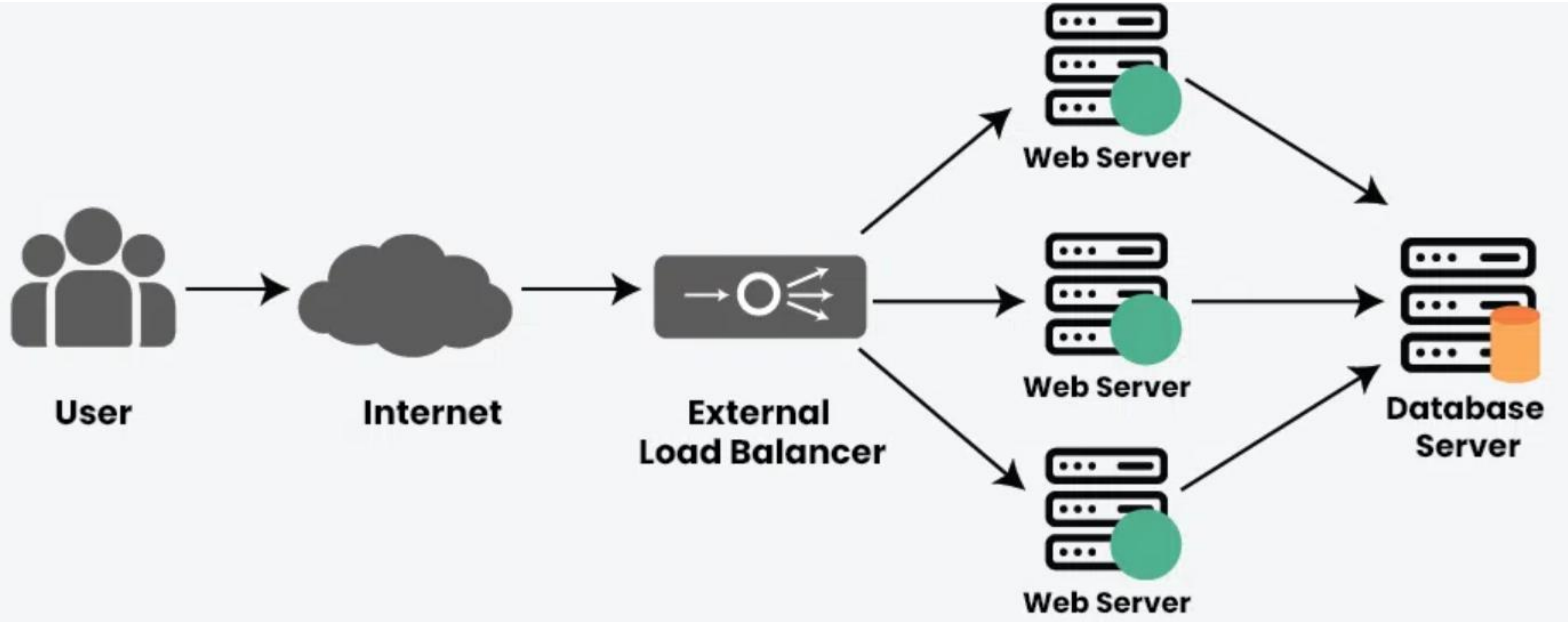
- **Traffic Distribution:** To keep any one server from becoming overburdened, load balancers divide incoming requests evenly among several servers.
- **High Availability:** Applications' reliability and availability are improved by load balancers, which divide traffic among several servers. The load balancer reroutes traffic to servers that are in good condition in the event that one fails.
- **Scalability:** By making it simple to add servers or resources to meet growing traffic demands, load balancers enable horizontal scaling.
- **Optimization:** Load balancers optimize resource utilization, ensuring efficient use of server capacity and preventing bottlenecks.
- **Health Monitoring:** Load balancers often monitor the health of servers, directing traffic away from servers experiencing issues or downtime.
- **SSL Termination:** Some load balancers can handle SSL/TLS encryption and decryption, offloading this resource-intensive task from servers.

What is a Load Balancer?

- ❑ Scaling Out (Horizontal Scaling) with load balancer!
- ❑ IP of load balancer set in DNS. So servers can be in private network. More Security!



How Load Balancer Works?



Types of Load Balancers

- **1. Hardware Load Balancers**

  These are real devices that are set up within a data center to control how traffic is distributed among servers. They are highly reliable and work well since they are specialized devices, but they are costly to purchase, scale, and maintain. They're often used by large companies with consistent, high traffic volumes.

- **2. Software Load Balancers**

  These are software or programs that divide up traffic among servers. They operate on pre-existing infrastructure (on-premises or in the cloud), in contrast to hardware load balancers.

- **3. Cloud Load Balancers**

  Cloud load balancers, which are offered as a service by cloud providers like AWS, Google Cloud, and Azure, automatically distribute traffic without requiring physical hardware. Users just pay for the resources they use, and they are very scalable. They are perfect for dynamic workloads since they can readily interface with cloud-based apps and adjust to traffic spikes.

Flow Example

```
+-----------------+       +--------------------+       +-----------------+
|     User's      |       |   Load Balancer    |       |     Backend     |
|     Browser     | <-----> |      (Nginx)       | <----> |     Servers     |
|   (client)      |       |   (Distributes     |       | (Handles Login, |
|                 |       |    Traffic)        |       | Auth, Database) |
+-----------------+       +--------------------+       +-----------------+
```

User accesses the website:

User enters sad.sharif.edu in the browser.

DNS resolves the domain to the IP address of the Load Balancer (Nginx).

Load Balancer (Nginx) receives the request:

The request reaches Nginx (Load Balancer).

Nginx forwards the request to one of the backend servers based on load balancing algorithm (e.g., Round Robin or Least Connections).

Backend Server serves the login page:

The selected backend server processes the request and serves the login page (HTML).

The server sends the login page back to Nginx, which then sends it to the user's browser.

Flow Example

```
+-----------------+          +--------------------+        +-----------------+
|     User's      |          |   Load Balancer    |        |     Backend     |
|     Browser     | <-----> |      (Nginx)       | <----> |     Servers     |
|   (client)      |          |   (Distributes     |        | (Handles Login, |
|                 |          |     Traffic)       |        | Auth, Database) |
+-----------------+          +--------------------+        +-----------------+
```

User enters credentials (username & password):

User fills in the login form and submits the credentials.

A POST request with the username and password is sent to sad.sharif.edu/login.

Load Balancer (Nginx) forwards the POST request:

Nginx receives the POST request and forwards it to one of the backend servers (based on load balancing).

Backend Server processes the login:

The backend server receives the POST request and begins processing the login logic.It connects to the Database to verify the username and password.

Flow Example

```
+----------------+         +-------------------+        +----------------+
|     User's     |         |   Load Balancer   |        |    Backend     |
|     Browser    | <-----> |      (Nginx)      | <----> |    Servers     |
|   (client)     |         |   (Distributes    |        |(Handles Login, |
|                |         |     Traffic)      |        |Auth, Database) |
+----------------+         +-------------------+        +----------------+
```

Backend Server queries the database:

The backend server queries the Database to check if the provided username exists and if the password matches.

If credentials are correct, it generates an authentication token.

Response sent from the backend server:

If the credentials are valid, the backend server creates an authentication token (or session ID) and sends it back to Nginx.

If invalid, the backend server sends an error response (e.g., 401 Unauthorized).

Flow Example

```
+----------------+        +-------------------+       +----------------+
|    User's      |        |   Load Balancer   |       |    Backend     |
|    Browser     | <----> |      (Nginx)      | <---> |    Servers     |
|  (client)      |        |   (Distributes    |       | (Handles Login,|
|                |        |    Traffic)       |       | Auth, Database)|
+----------------+        +-------------------+       +----------------+
```

Load Balancer (Nginx) sends response to the user:

Nginx sends the response to the user's browser.

- If successful, the response contains an authentication token or session cookie.
- If failed, an error message is returned to the user.

User is redirected to the dashboard or home page:

Upon successful login, the user is redirected to the main dashboard or homepage.

For subsequent requests, the authentication token (in the form of a cookie or header) is sent along with the request for authentication.

Challenges of using Load Balancers

1. **Single Point of Failure:** Load balancers might create a single point of failure even though they improve fault tolerance. Issues with the load balancer itself could cause traffic distribution to be disrupted.
2. **Complexity and Cost:** High-quality load balancing solutions may be expensive, and load balancer implementation and management can be complicated. This covers load balancers for both software and hardware.
3. **Configuration Challenges:** Configuring load balancers correctly can be challenging, especially when dealing with complex application architectures or diverse server environments.
4. **Potential for Overhead:** Depending on the load balancing technique and configuration, there may be additional overhead in the form of delay and processing time, even though modern load balancers are designed to lessen this effect.
5. **SSL Inspection Challenges:** When SSL termination is performed at the load balancer, it may introduce challenges related to SSL inspection and handling end-to-end encryption.

How can you prevent a load balancer from being a single point of failure (SPOF)?

- Load Balancer crash refers to a sudden failure of a load-balancing system that helps in distributing the network traffic across multiple servers and resources of a system.

1) Use multiple load balancers

- One of the simplest ways to avoid a load balancer from being a SPOF is to use more than one load balancer in your architecture. You can deploy two or more load balancers in parallel, with each one handling a portion of the traffic, or in failover mode, with one acting as a backup for the other. You can also use a load balancer cluster, which is a group of load balancers that work together as a single logical unit, sharing configuration and health information. A load balancer cluster can provide load balancing, redundancy, and scalability for your applications.

2) Monitor and optimize load balancer performance

- Another way to prevent a load balancer from being a SPOF is to monitor and optimize its performance regularly. You should use tools and metrics to track the load balancer's health, capacity, throughput, latency, errors, and availability. You should also perform load testing and benchmarking to identify and resolve any bottlenecks or issues that could affect the load balancer's performance. You should also tune the load balancer's settings and parameters to optimize its resource utilization, traffic distribution, and session management.

3) Implement load balancer security

- A third way to prevent a load balancer from being a SPOF is to implement load balancer security. You should protect your load balancer from unauthorized access, malicious attacks, and data breaches. You should use encryption, authentication, authorization, and firewall rules to secure the communication between the load balancer and the clients and servers. You should also use security patches, updates, and audits to keep the load balancer software up to date and compliant with the latest standards and regulations.

4) Leverage cloud-based load balancing services

- A fourth way to prevent a load balancer from being a SPOF is to leverage cloud-based load balancing services. Cloud-based load balancing services are offered by cloud providers as a managed service that can automatically scale, balance, and monitor the traffic for your applications. Cloud-based load balancing services can provide high availability, reliability, and performance for your load balancer, without requiring you to maintain or operate the load balancer hardware or software. You can also benefit from the cloud provider's global network, security, and support features.

5) Use DNS-based load balancing

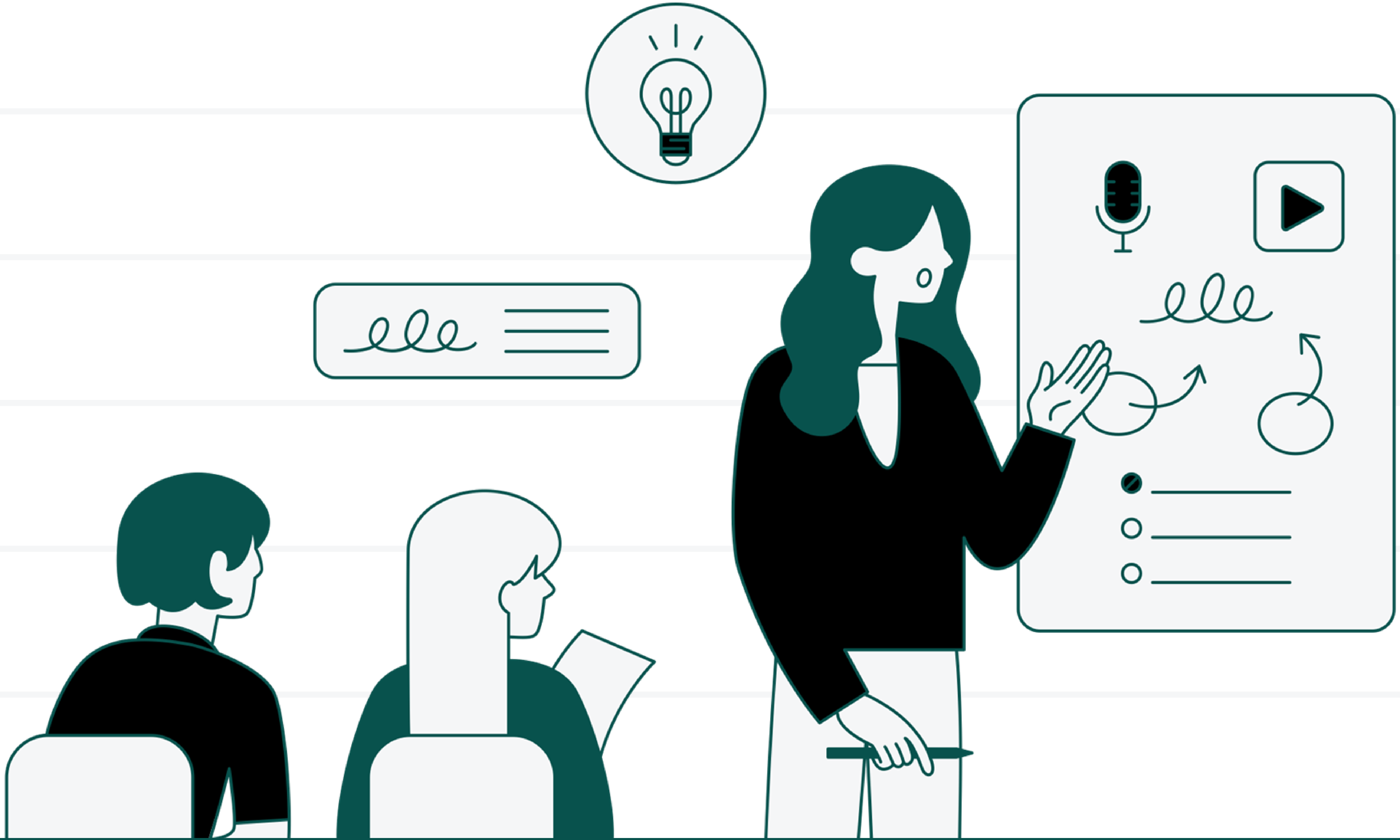
- A fifth way to prevent a load balancer from being a SPOF is to use DNS-based load balancing. DNS-based load balancing is a technique that uses the Domain Name System (DNS) to distribute the traffic across multiple load balancers or servers. DNS-based load balancing can provide failover, redundancy, and geo-distribution for your load balancer, by resolving the requests to different IP addresses based on the availability, proximity, and performance of the load balancers or servers. You can also use DNS-based load balancing to route the traffic to different regions, zones, or domains, depending on your business needs and preferences.

6) Combine multiple load balancing methods

- A sixth way to prevent a load balancer from being a SPOF is to combine multiple load balancing methods in your architecture. You can use a hybrid or multi-layered approach that integrates different types of load balancing techniques, such as hardware, software, cloud, and DNS-based load balancing. By combining multiple load balancing methods, you can achieve higher levels of availability, scalability, and performance for your load balancer, as well as greater flexibility and control over your traffic management. You can also use different load balancing algorithms, such as round robin, least connections, or weighted, to optimize the traffic distribution and load balancing efficiency.

Conclusion

- A load balancer enables elastic scalability which improves the performance and throughput of data. It allows you to keep many copies of data (redundancy) to ensure the availability of the system. In case a server goes down or fails you'll have the backup to restore the services.
- Load balancers can be placed at any software layer.
- Many companies use both hardware and software to implement load balancers, depending on the different scale points in their system.

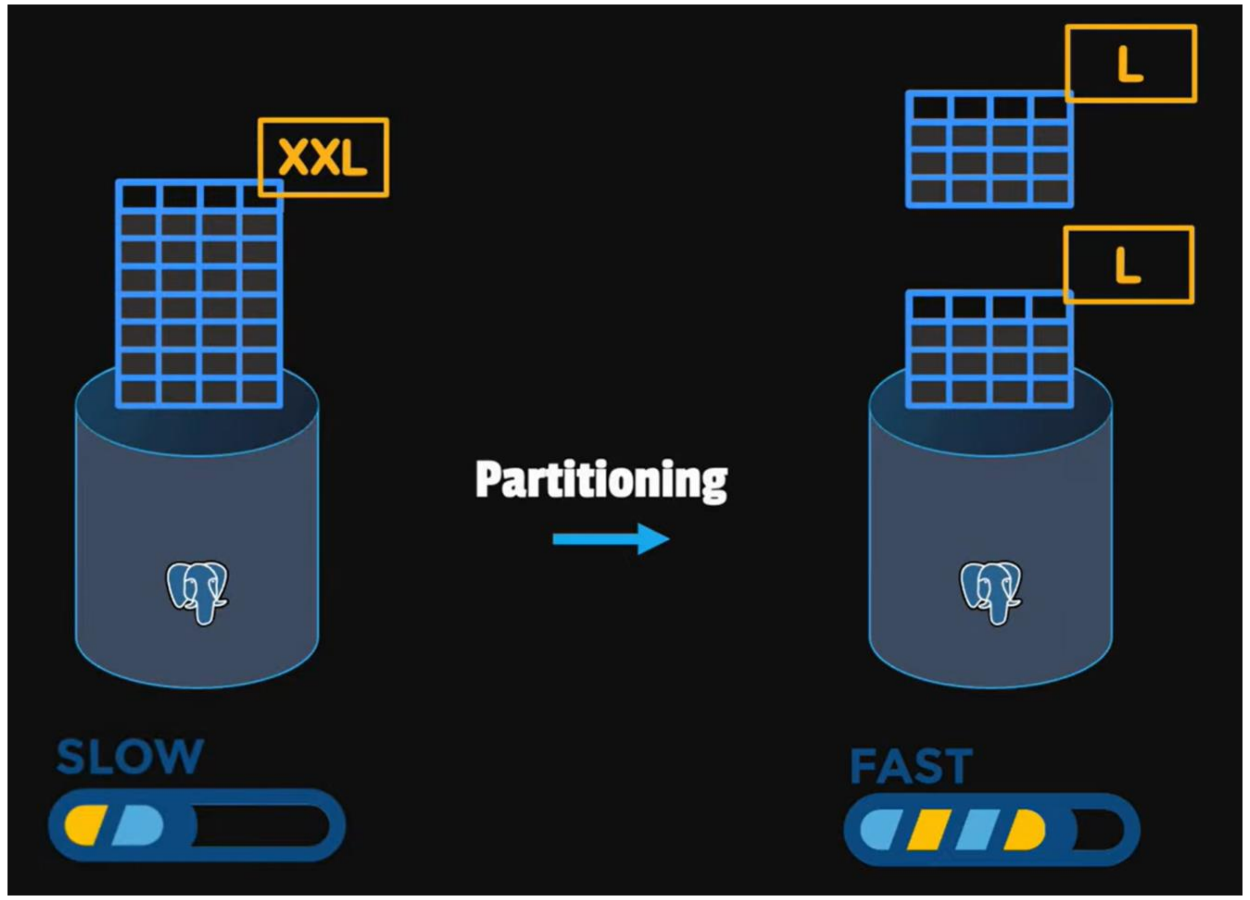


04 Database Sharding (Partitioning)

Partitioning

- When the table size grows over time, each operation cost on the table will increase as well.
- We can't increase the size of the table over 32GB in normal conditions. Before reaching this size performance issues may arise.

Good Solution: Partitioning

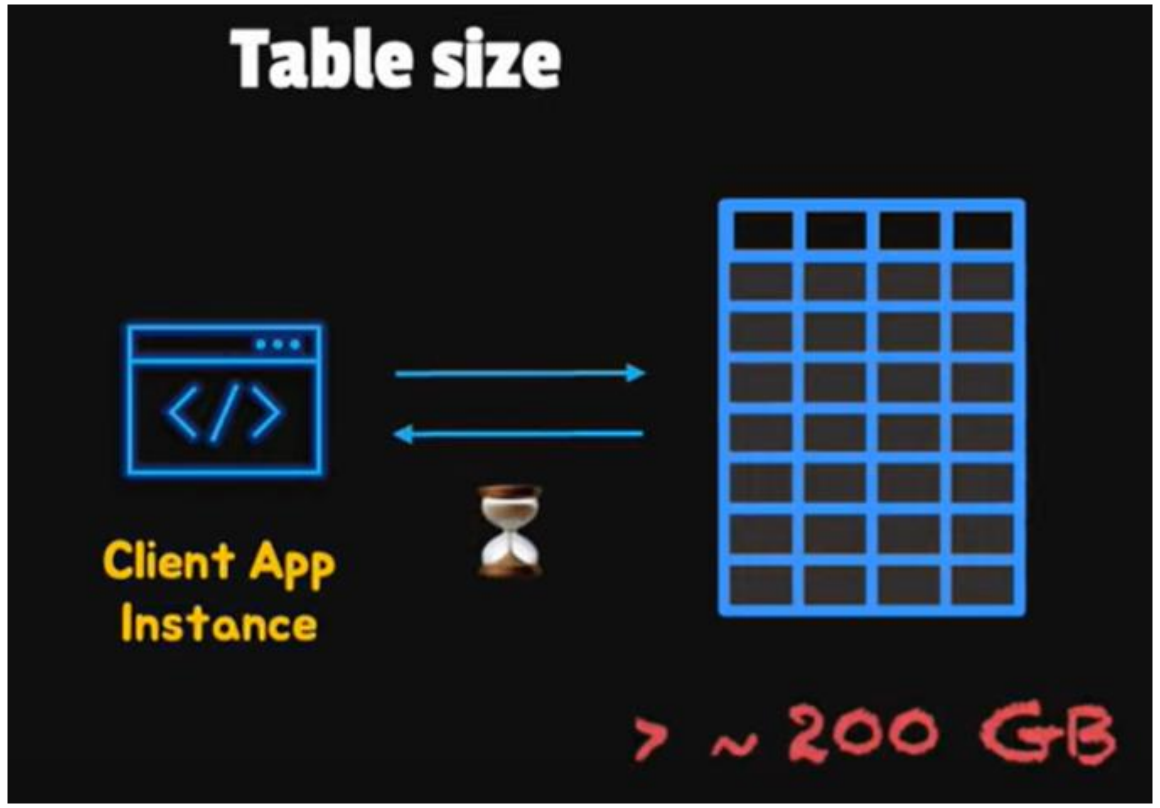


Add partitioning for a table?

- It shouldn’t be the first option to improve performance!!! Why?
  - It adds another level of complexity!!
  - Unlike other performance enhancing such as indexing, partitions are part of table definition so its difficult to change!!

Add partitioning for a table?

- Signs to check a table needs partitioning:
  1) Table Size: there is no rule! But encounter long responses time and table is larger than 200GB

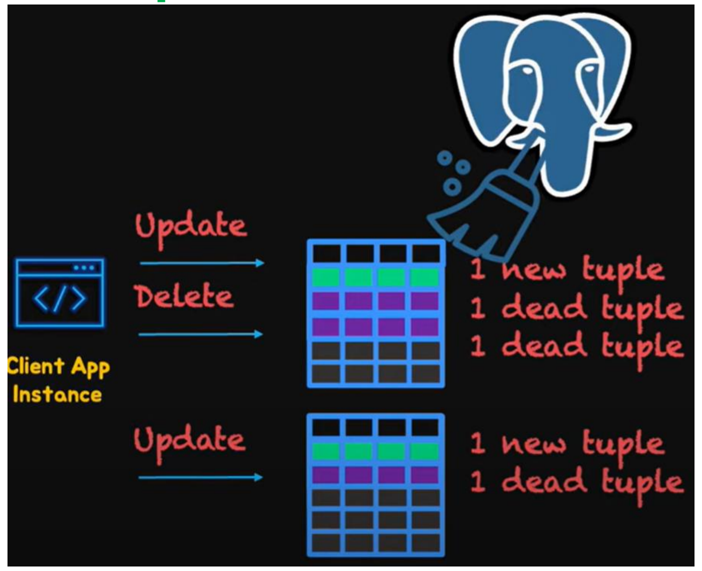


Add partitioning for a table?

2) Table Bloat: For a DELETE, it simply marks the row as unavailable for future transactions, and for UPDATE, under the hood it's a combined INSERT then DELETE, where the previous version of the row is marked unavailable.

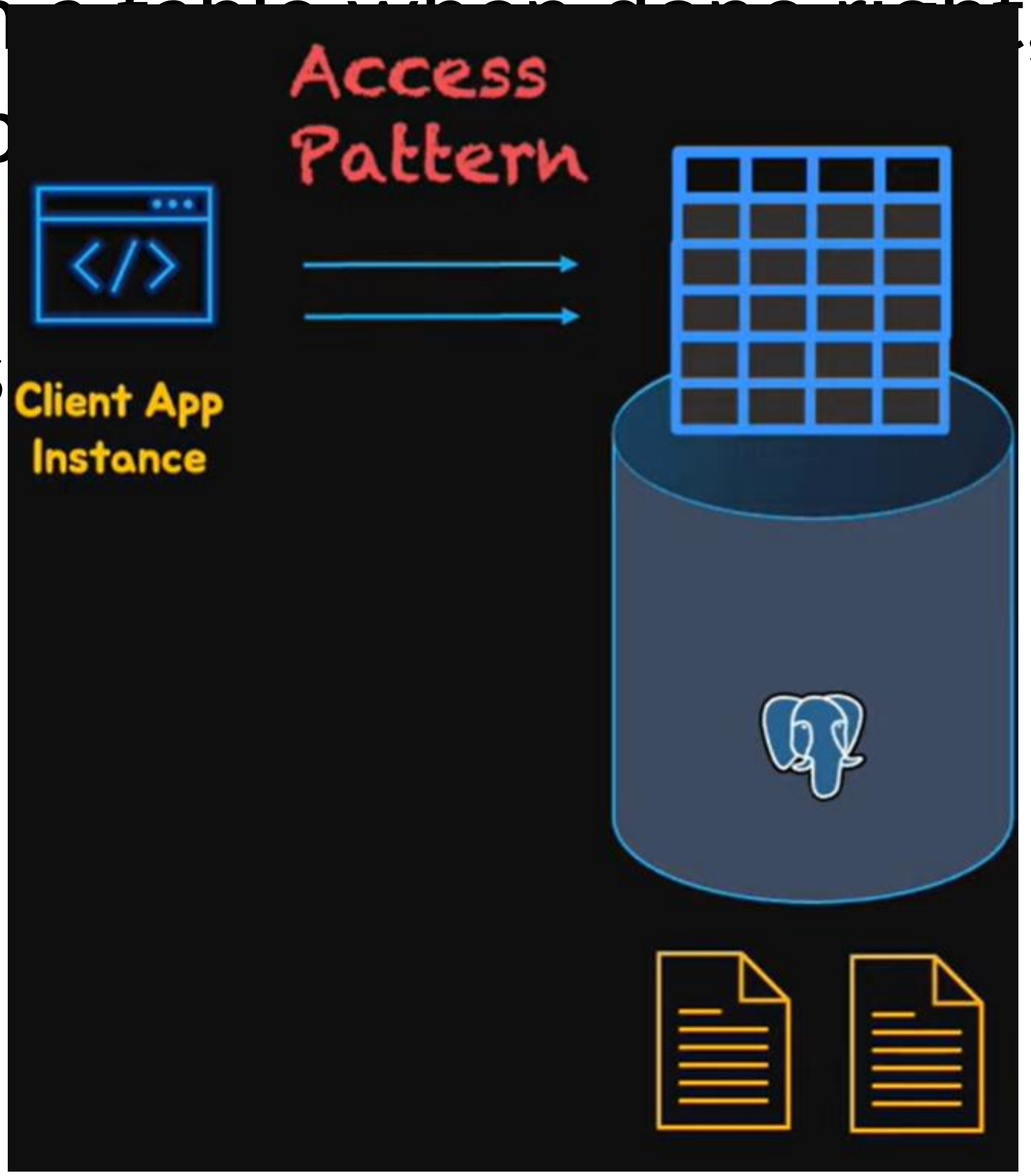
The space cannot be used. To then mark the space as available for use by the database, a vacuum process (manually or automatically) needs to come along behind the operations, and mark that space available for the database to use.

Vacuum process should scan all rows. If table is large vacuum process will take longer. Partitioning can help to make it faster with less CPU.

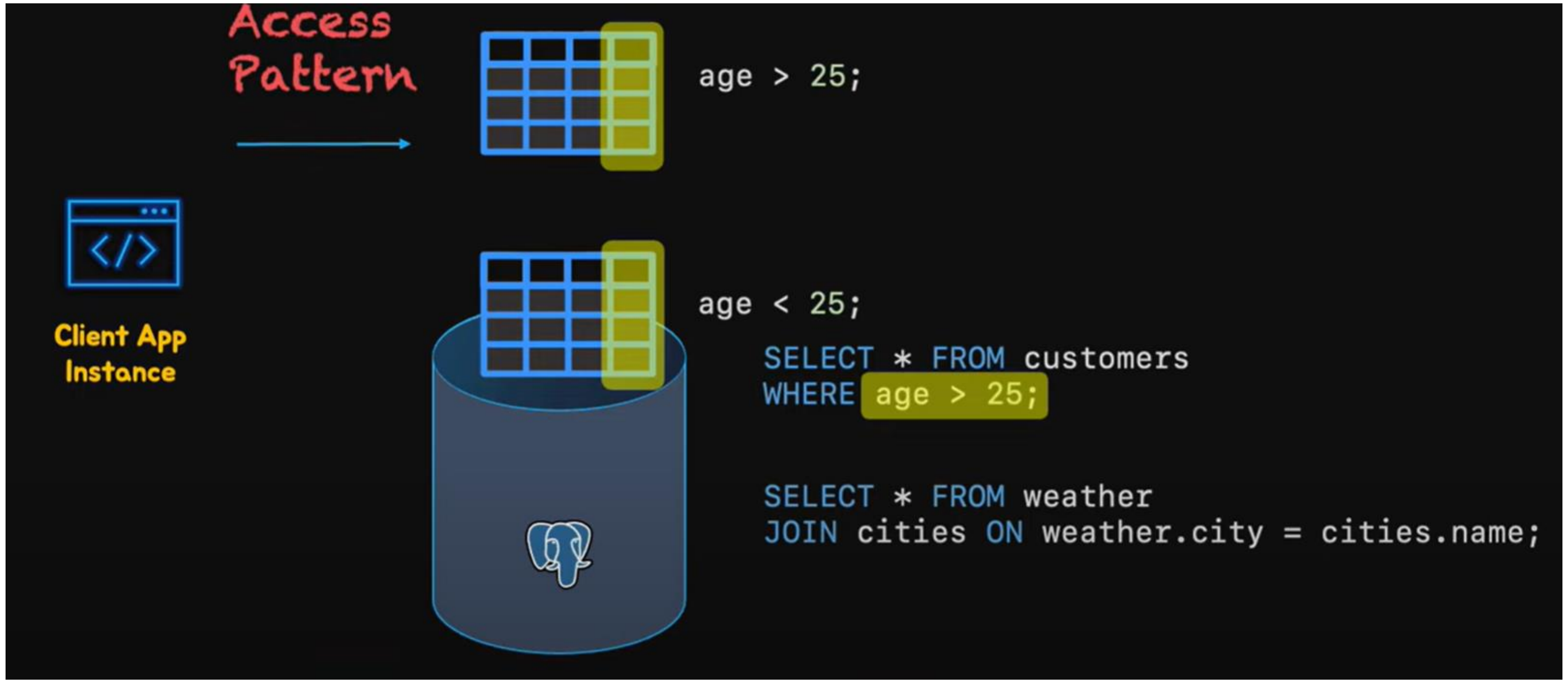


How should the Tables be partitioned?

- Partitioning can drastically improve performance on ... , but when not needed o done wrong can make the p... can make the database unstable.
- First look for access patterns for splitting the tables
  - By knowing the applications that access the database.
  - Monitoring the logs and generating reports.



How should the Tables be partitioned?



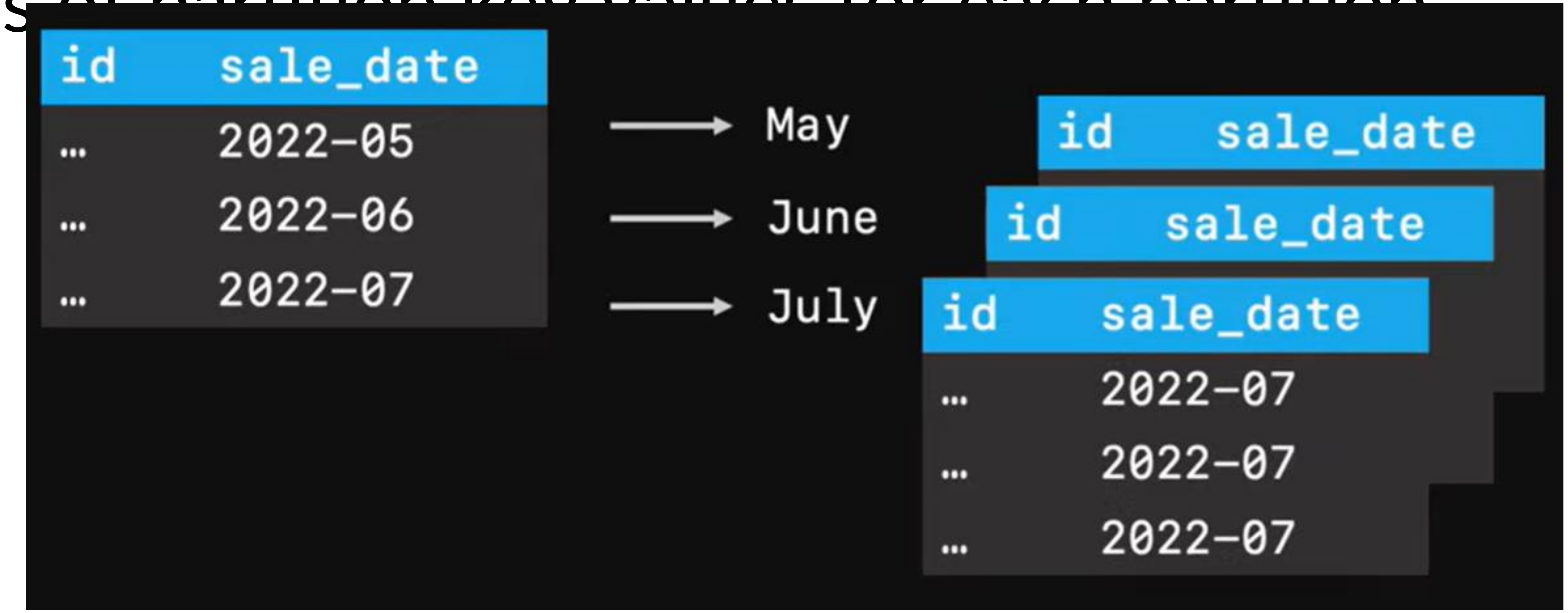
We look for columns that are either in **where** or in **join** conditions.
These will be the partition keys.
In a good design, we have a small subset of data rather than the whole

Partitioning Methods

- Range Partition
- List Partition
- Hash Partition
- Composite Partition

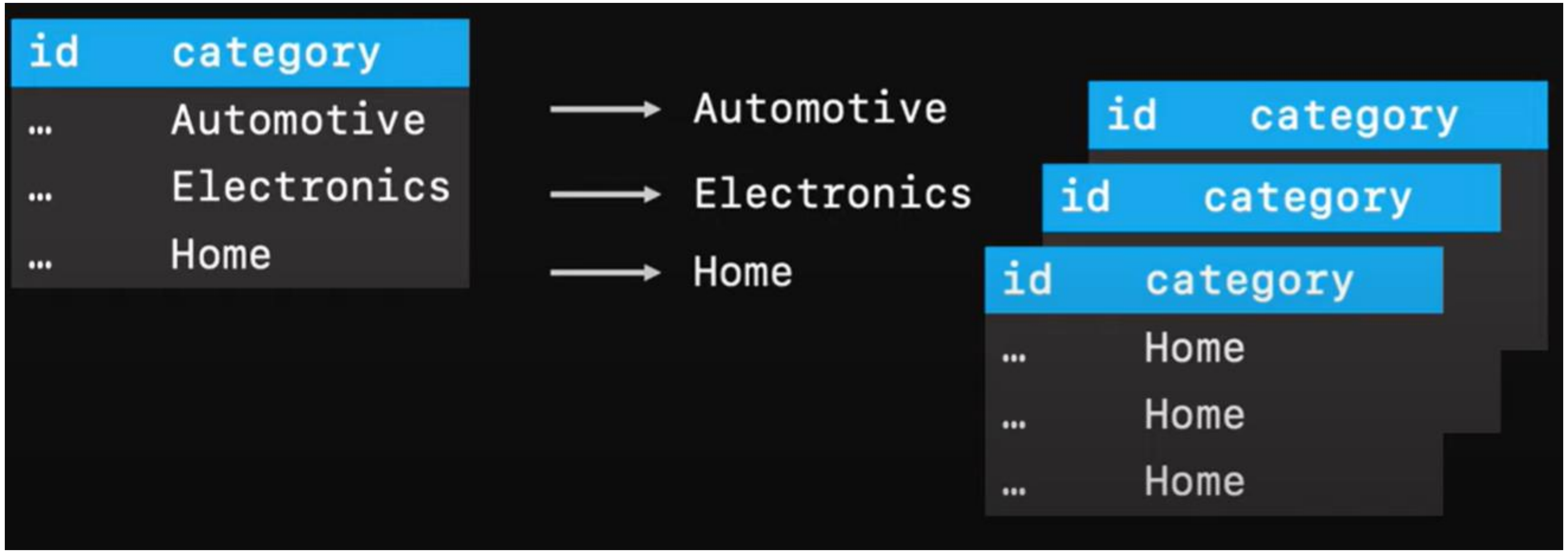
Partitioning Methods

- Range partitioning maps data to partitions on the basis of ranges of partition key values for each partition.



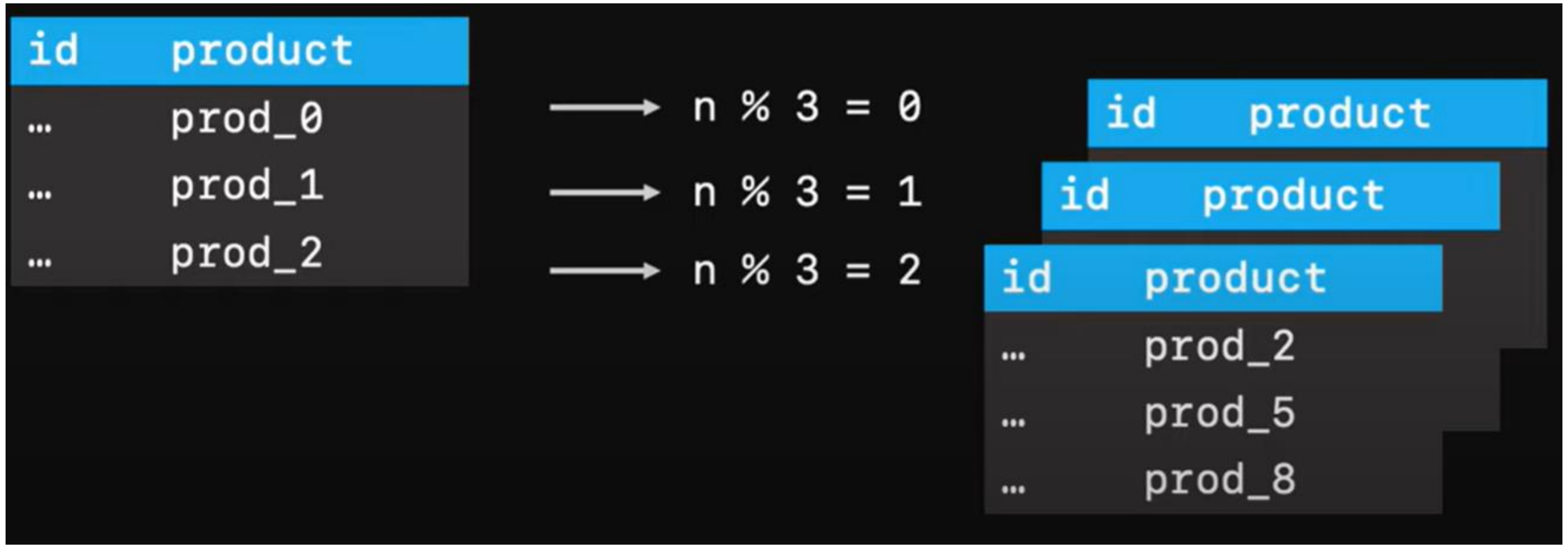
Partitioning Methods

- List partitioning maps rows to partitions by using a list of discrete values for the partitioning column.
  - Good when partition key is category value.



Partitioning Methods

- Hash partitioning maps data to partitions by using a hashing algorithm applied to a partitioning key.
  - Especially useful when there is no obvious way of diving data into logical groups.

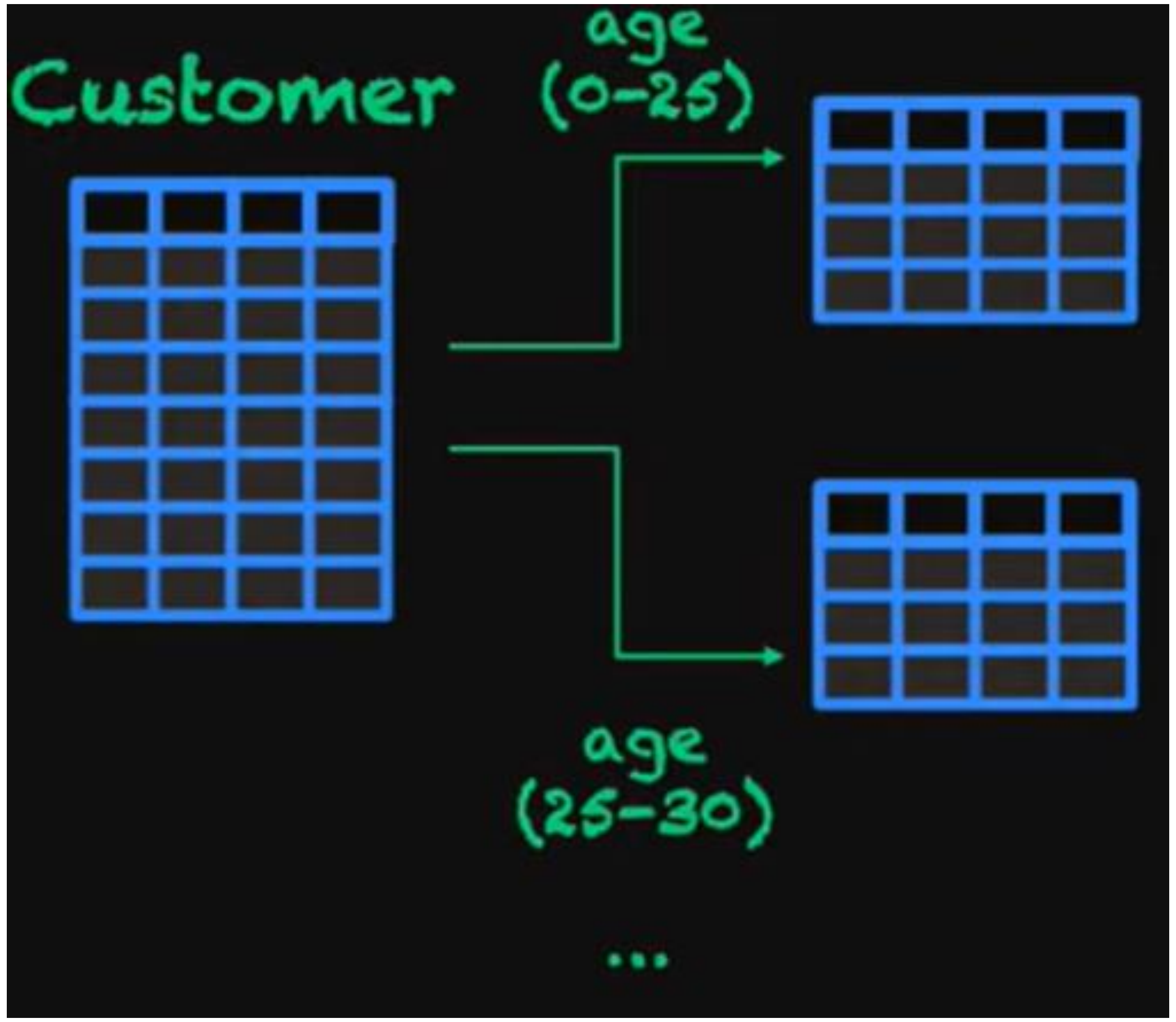


Partitioning Methods

- Composite partitioning:
  - Range-Hash sub partitions the range partitions using a hashing algorithm.
  - Range-List sub partitions the range partitions using an explicit list.

Range Partition - Example

- Consider following table with not null age attribute:



Range Partition- Example

- create table customers (id integer, name text, age numeric) partition by range(age)
- create table cust_young partition of customers for values from (MINVALUE) to (25)
- create table cust_medium partition of customers for values from (25) to (75)
- create table cust_old partition of customers for values from (75) to (MAXVALUE)
- insert into customers values (1,'Bob',20), (2,'Alice',20),(3,'Doe',38),(4,'Richard',80)
- select * from customers c
- select tableoid::regclass,* from customers c


https://amar.sabteahval.ir
Protected By CloudGuard.ir
شهروند گرامی
خدمات این درگاه تا اطلاع ثانوی از ساعت ۷ صبح الی ۲۰ می‌باشد.