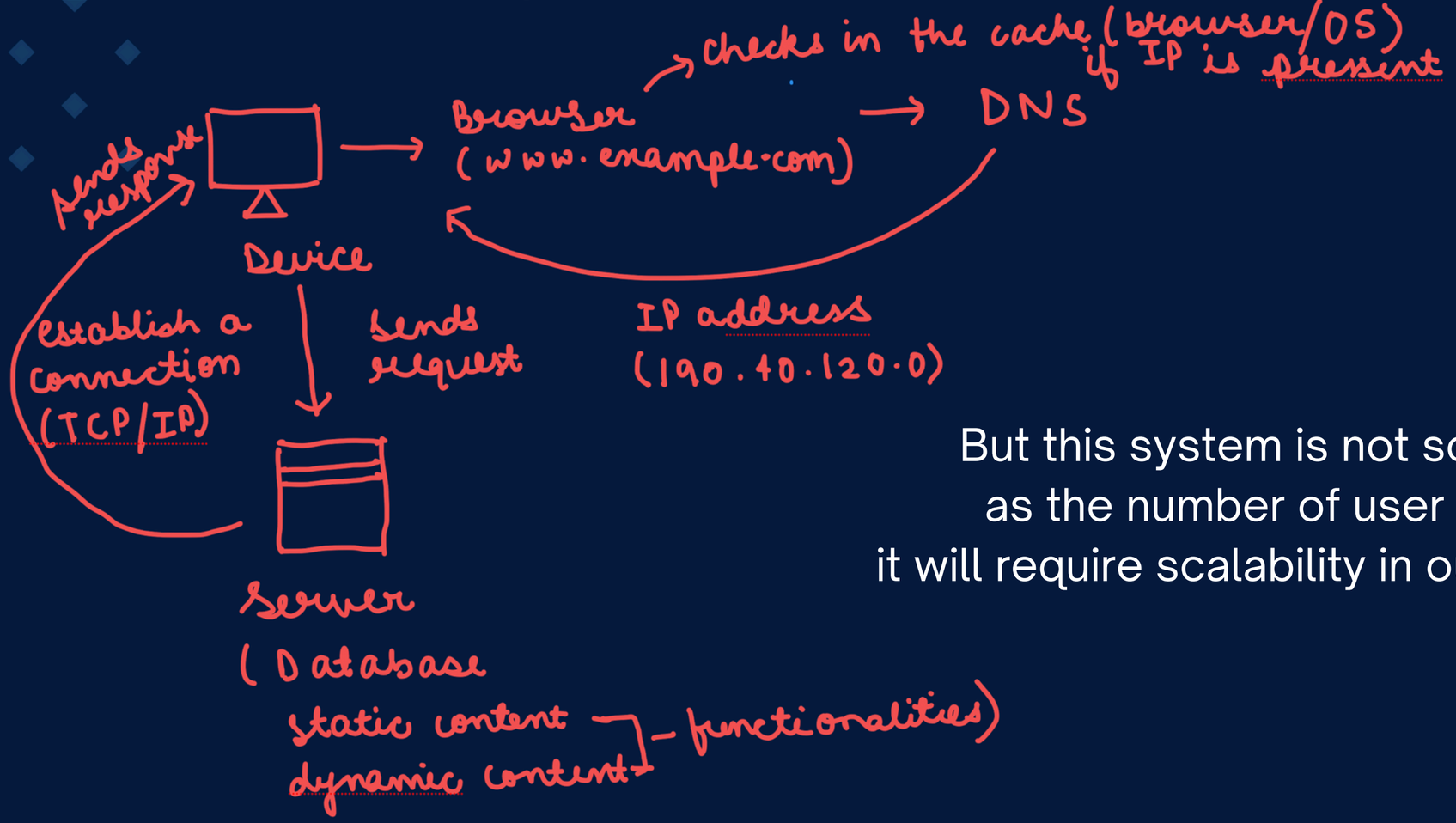
LET'S START WITH DBMS :)

Scaling in Databases

Process of user sending a request to access data on the internet

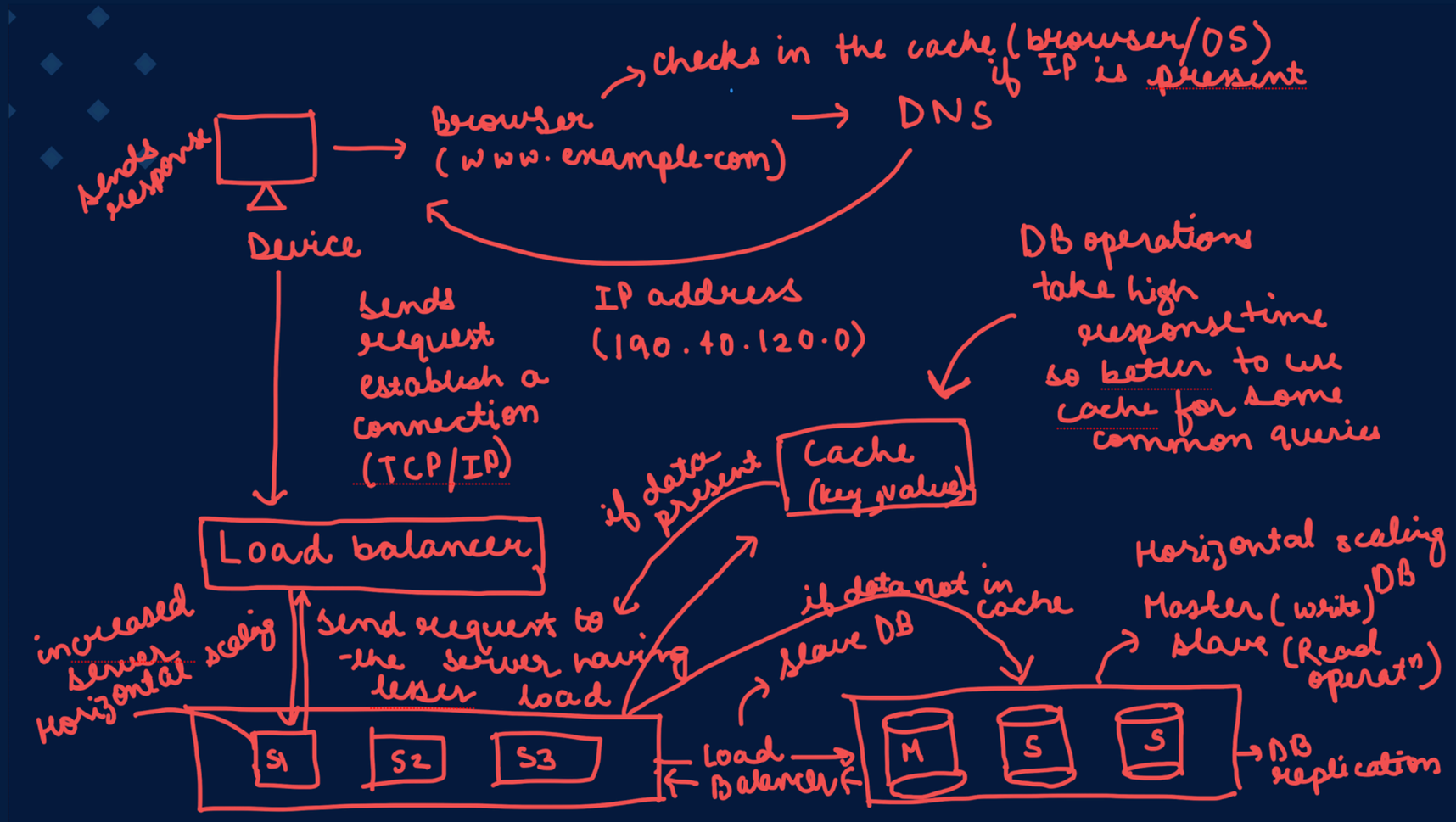


But this system is not scalable
as the number of user grows
it will require scalability in our systems

LET'S START WITH DBMS :)

Scaling in Databases

How to scale a Database?



LET’S START WITH DBMS :)

Scaling in Databases

User Journey

1. User Action: Sending a Request
2. Browser Parses the URL
3. DNS Lookup
4. Establishing a Connection
5. Sending the HTTP Request
6. Server Processing
7. Browser Receives the Response
8. Displaying the Web Page
9. Caching

LET’S START WITH DBMS :)

Scaling in Databases

The user sends a request by entering a URL or clicking a link. The browser then translates the domain name into an IP address through DNS lookup, establishes a connection with the server, sends an HTTP request, and receives a response.

The browser then processes this response, rendering the webpage and displaying it to the user. The process involves multiple layers of networking, data transfer, and client-server interaction, all working together to deliver the desired content to the user’s screen.

LET’S START WITH DBMS :)

Scaling in Databases

Scaling in databases is a crucial topic in development, mostly when applications grow and the volume of data increases.

- Vertical Scaling (Scaling Up)
- Horizontal Scaling (Scaling Out)
- Database Partitioning
- Replication
- Sharding
- Caching
- Load Balancing

LET’S START WITH DBMS :)

Increase the capacity of instances RAM,CPU

Scaling in Databases

Vertical Scaling (Scaling Up)

Vertical scaling involves increasing the capacity of a single database server.

- **Adding More Resources:** Upgrading the server by adding more RAM, faster processors, or larger and quicker storage. This helps the server handle more requests and store more data.

- **Improving Performance:** Tweaking the database settings, optimizing queries, and indexing tables to make better use of the hardware.

- **Upgrading to a Stronger Machine:** Moving the database to a more powerful server, like switching from a regular server to a high-performance cloud instance.

LET’S START WITH DBMS :)

Scaling in Databases

Vertical Scaling (Scaling Up)

Advantages:

- Easy to set up because it usually doesn't need changes to the app or database structure.
- Simple to manage since everything is kept on one server.

Disadvantages:

- Limited by the hardware's maximum capacity.
- Expensive, as high-end hardware can be very costly.
- Single point of failure: if the server fails, the entire database goes offline.

LET’S START WITH DBMS :)

add more instances

Scaling in Databases

Horizontal Scaling (Scaling Out):

Horizontal scaling involves spreading the database workload across multiple servers. This can be done in different ways:

a. Database Replication:
Replication is the process of copying data from one main server (master) to one or more backup servers (slaves).

- **Master-Slave Replication:** The master server manages all write operations, while read operations are distributed among the slave servers. This setup improves read performance and provides redundancy.

- **Multi-Master Replication:** Multiple servers handle both read and write operations, increasing availability and write capacity. However, it also introduces challenges in managing conflicts.

LET’S START WITH DBMS :)

Scaling in Databases

Divide the data between multiple tables in diff db instances

Horizontal Scaling (Scaling Out):

b. **Database Sharding**

Sharding involves partitioning the database into smaller, more manageable pieces (shards), with each shard stored on a separate server.

- Range-Based Sharding: Data is divided based on ranges of a particular key, such as date or user ID.
- Hash-Based Sharding: A hash function is applied to a key (e.g., user ID) to determine which shard the data should go to.

LET’S START WITH DBMS :)

Scaling in Databases

- **Caching**: Implement caching layers (e.g., Redis, Memcached) to store frequently accessed data in memory, reducing the load on the database.
- **Load Balancing**: Distribute database queries across multiple servers using load balancers to prevent any single server from becoming a bottleneck
- **Partitioning:** Divide large tables into smaller partitions that can be queried independently, improving query performance.

Partitioning : Divide the data into multiple table in same DB instance