Facebook

22z433 Santhosh TK
23z431 Dwarkesh
23z432 Kapil arif K
23z434 Praveen G
23z435 Sudharsan S
23z436
VasanthKumaar SB

Introduction to facebook

23z434

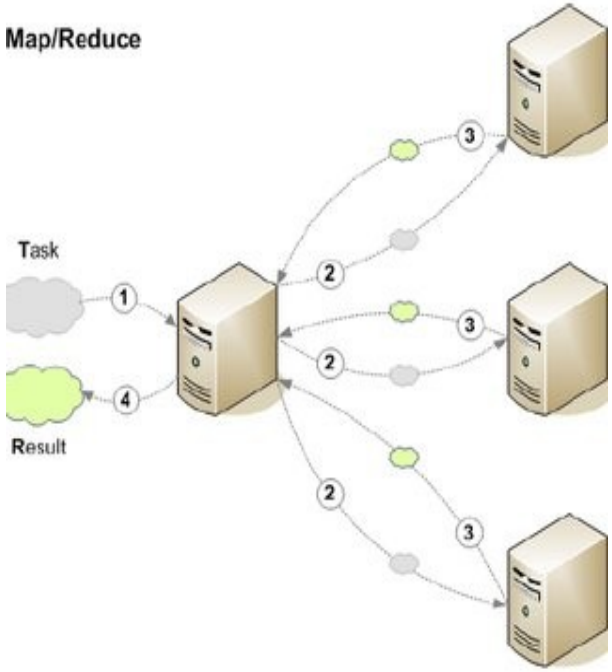
PRAVEEN G

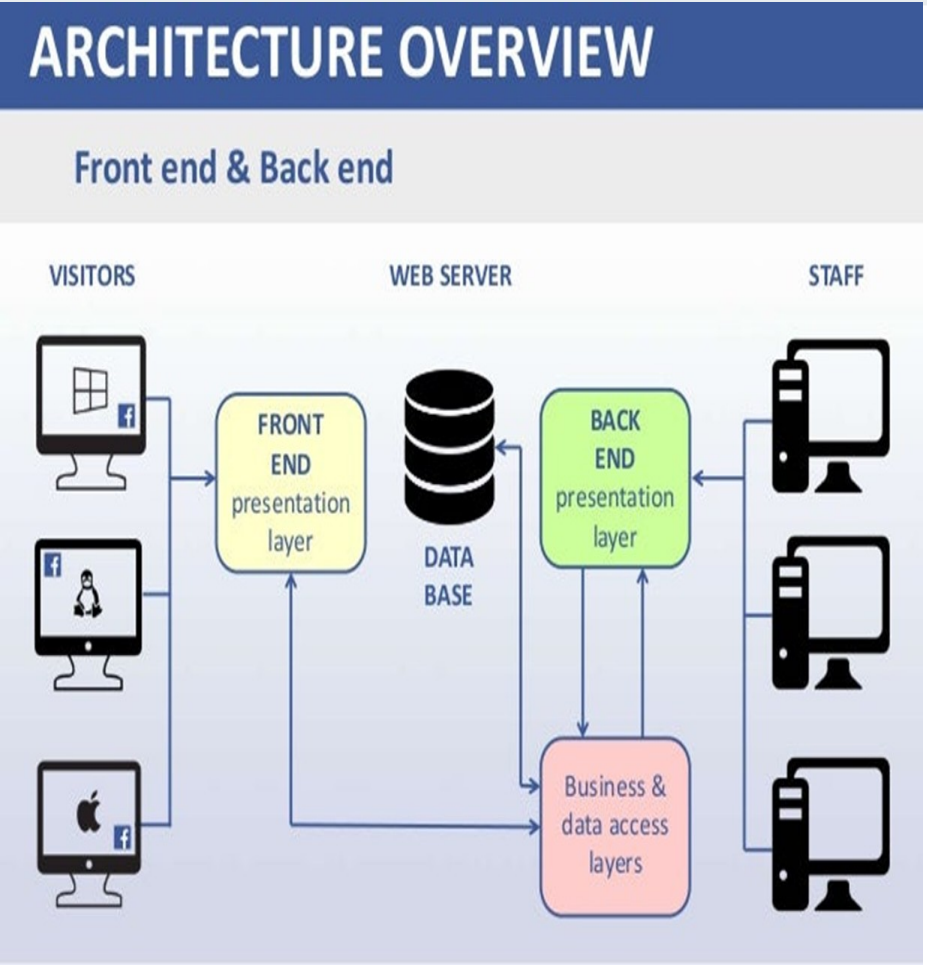
INTRODUCTION:

- Facebook is one of the largest social networking platforms, connecting over 3 billion users worldwide.
- Every second, Facebook processes a massive volume of data—from posts, messages, comments, to photos and videos.
- To handle this enormous scale efficiently, Facebook relies on distributed computing, which helps manage large data, ensure fast performance, and provide reliable services to users around the globe.

Why Facebook Needs Distributed Computing:

- **Massive Scale:** With billions of users, Facebook needs systems that can process huge amounts of data simultaneously.
- **Load Balancers:** Automatically redirect traffic if one server goes down, so users don’t experience downtime.
- **Backup Systems:** Regular backups ensure that data can be restored quickly in case of a failure.
- **Monitoring Tools:** Systems are constantly monitored to detect and fix issues before they affect users.
- **Distributed Systems:** Spreading data and services across multiple locations reduces the risk of a single point of failure.





Facebook's **frontend** is built using the **LAMP stack**.

BACKEND :TECHNOLOGY STACKS

Programming Languages: C++,Python,Java,Erlang

LAMP stack, which includes:

- **Linux** for server operations.
- **Apache** as the web server to handle incoming requests.
- **MySQL** for managing user data.
- **PHP** to create dynamic and interactive web content.

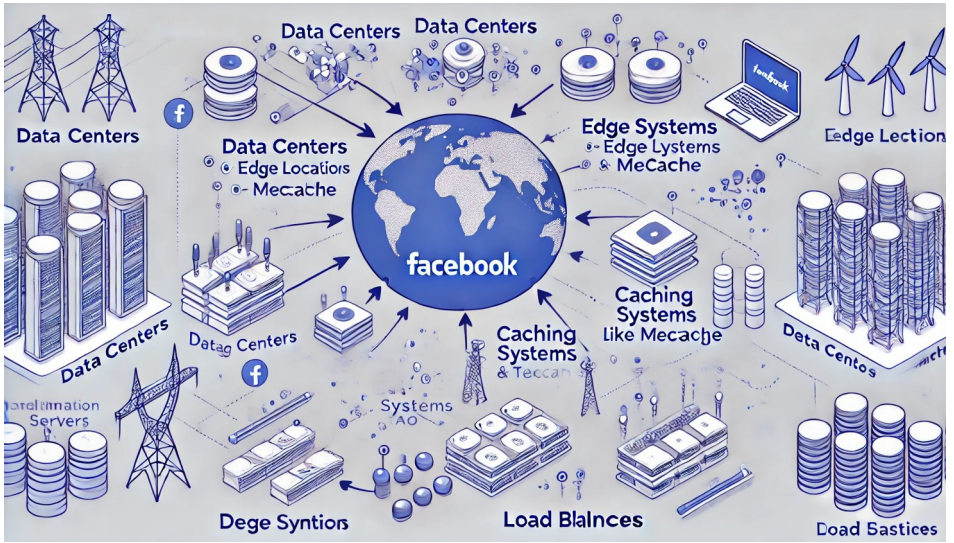
Why LAMP?

- It's **scalable** and can handle large traffic volumes.
- Being **open-source**, it allows flexibility and cost efficiency.
- Reliable for serving **dynamic content** quickly.

Introduction of Facebook’s Infrastructure

K . KAPIL
ARIF
23Z432

Facebook operates at an unprecedented scale, requiring highly efficient, distributed systems to handle massive amounts of data and billions of user interactions daily. Traditional database and storage architectures are not sufficient for such a large-scale system. Instead, Facebook has developed specialized solutions to handle different types of workloads efficiently.

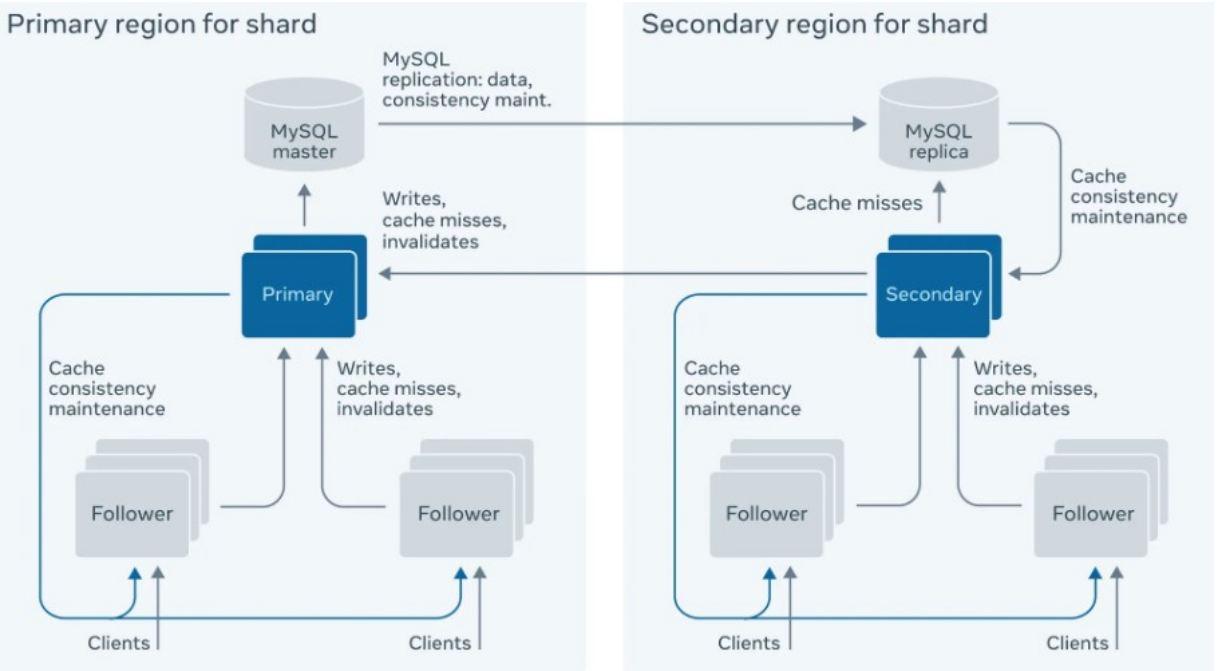


TAO (The Associations and Objects System)

TAO is Facebook's distributed graph storage system, designed to store and manage relationships between users, posts, and interactions efficiently.

Traditional relational databases struggle to handle such large-scale relationship queries efficiently.

TAO enables fast reads and writes, allowing users to retrieve and update their connections in real-time.



Primary Region for Shard:

- Clients interact with follower nodes that provide cached data.
- If the cache is missed, the request is forwarded to the Primary cache.
- If the data is still not found, a query is made to the MySQL master database.
- The response is cached and returned to the clients.
- Any writes, cache misses, or invalidations go through the Primary node to maintain data consistency.

Secondary Region for Shard:

- Similar to the Primary Region, but it uses a MySQL replica instead of the master database.
- Handles cache consistency maintenance by synchronizing with the Primary Region.
- Cache misses are forwarded to the Secondary cache and, if necessary, to the MySQL replica.

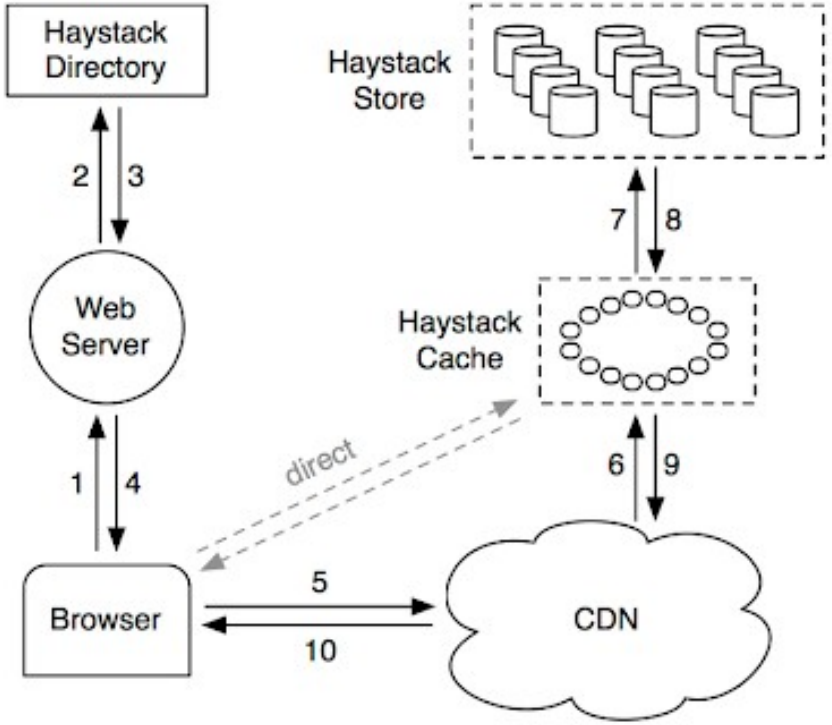
Haystack

Haystack is Facebook's high-performance photo storage system, designed to efficiently store and retrieve billions of small image files.

Facebook users upload over 350 million photos every day.

Storing images in traditional file systems results in slow access due to frequent disk operations.

Haystack minimizes disk access and retrieves images in a fraction of a second.



WORKING

1. Browser requests a photo → The request goes to the web server.
2. Web server queries the Haystack Directory to locate the requested photo.
3. Haystack Directory responds with the location of the photo.
4. Web server checks if the photo is available in cache/CDN(Content Delivery Network).
5. If the photo is already in the CDN, it is delivered instantly to the browser .
6. If not, the CDN requests the photo from Haystack Cache .
7. If the photo is in Haystack Cache, it is delivered to the CDN and then sent to the browser.
8. The CDN delivers the photo to the browser

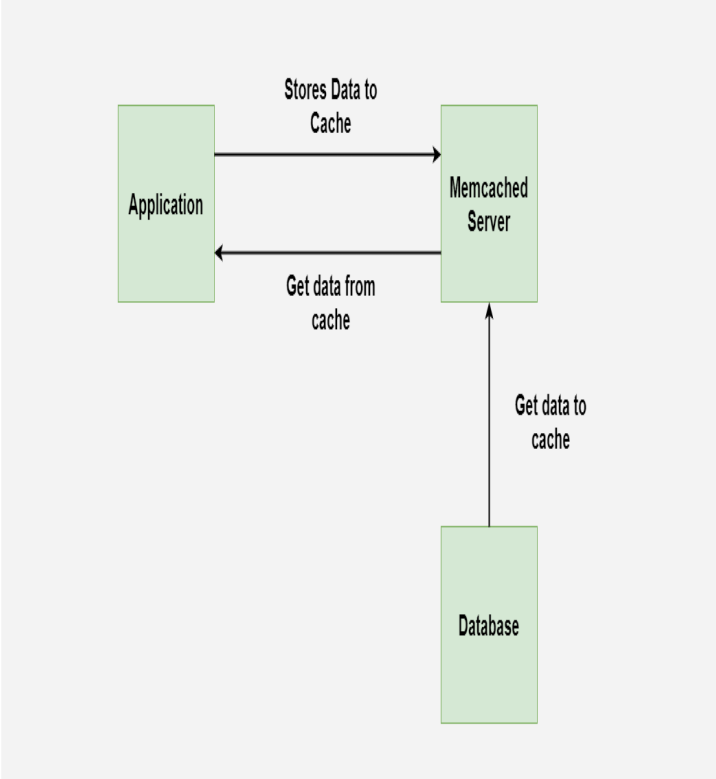
Memcached

Memcached is Facebook’s distributed caching system that stores frequently accessed data in memory to reduce database load and improve response times.

Facebook serves billions of requests per second.

If every request queried the database, it would cause slowdowns and high server costs.

Memcached stores frequently accessed data in RAM, making retrieval almost instantaneous



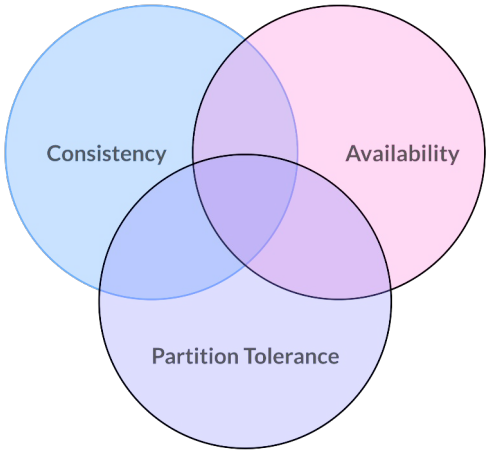
1. **Application Requests Data:**
    - When an application needs data, it first checks Memcached Server (cache) to see if the data is available.
2. **Cache Hit (Data Found in Memcached):**
    - If the data exists in Memcached, it is retrieved directly from the cache, avoiding a database query.
    - This speeds up the application's response time.
3. **Cache Miss (Data Not Found in Memcached):**
    - If the requested data is not in Memcached, the application queries the Database.
4. **Data Retrieval & Caching:**
    - The Database fetches the requested data.
    - The Memcached Server stores this data for future requests.
    - The Application receives the data from the database.
5. **Subsequent Requests:**
   The next time the same data is requested, it can be fetched directly from Memcached, reducing the load on the database.

Distributed System Concepts in Facebook infrastructure

Dwarkesh
23z431

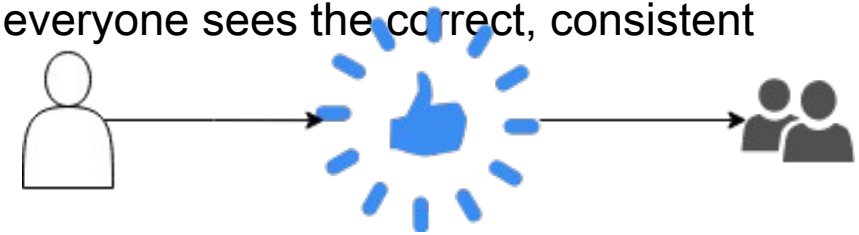
CAP Theorem

- Trade-offs between Consistency, Availability, and Partition Tolerance in distributed systems.
- Facebook often chooses availability and partition tolerance over strict consistency.
- For example, in Facebook's TAO distributed database, user actions like posting, liking, or commenting are always available, even if there are network partitions.
- Consistency might be temporarily relaxed so that the system stays responsive.



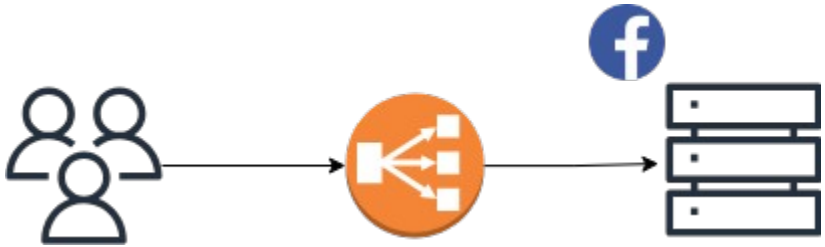
Eventual Consistency

- ➔ In distributed systems, data may not be immediately consistent, but it will sync eventually.
- ➔ When you ‘like’ a post on Facebook, it may show up instantly to you, but it might take a few seconds for that like to appear on your friend's feed.
- ➔ Facebook accepts temporary inconsistency to maintain speed and availability.
- ➔ Eventually, all replicas sync up, and everyone sees the correct, consistent state.



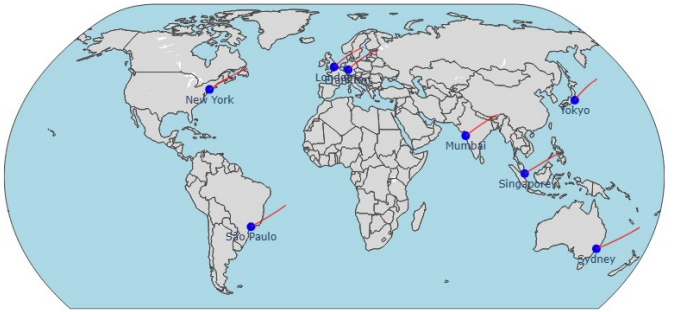
Load Balancing

- Load balancing is the process of distributing incoming network traffic across multiple servers to ensure no single server is overloaded.
- Facebook’s global infrastructure handles billions of user interactions per day, such as likes, comments, video plays, and messaging.
- Load balancers automatically distribute user requests to multiple backend servers to ensure fast response times and prevent crashes.



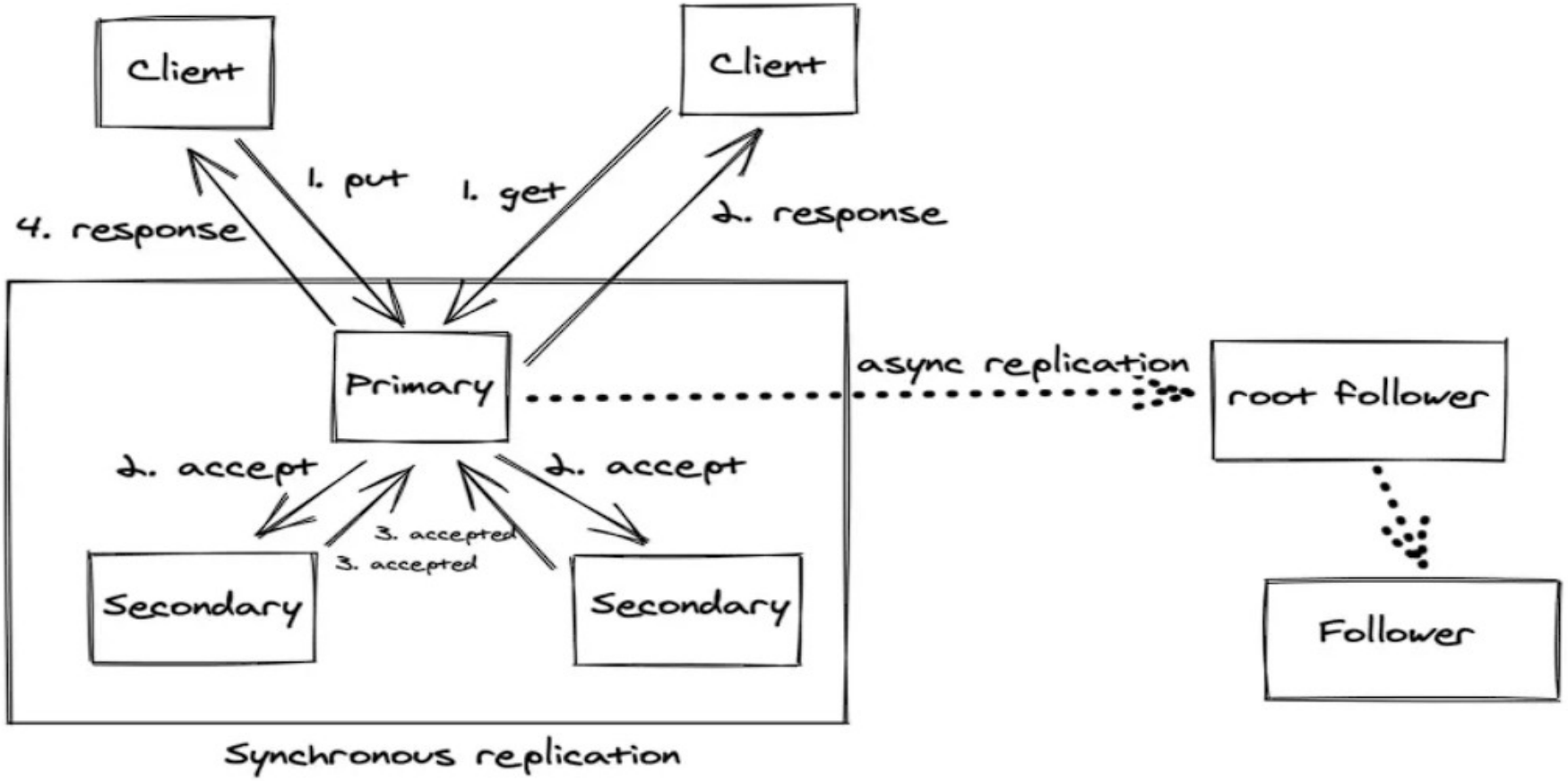
CDN (Content Delivery Network)

- A CDN (Content Delivery Network) is a group of distributed servers located worldwide (called edge servers) that deliver content to users from locations closer to them, reducing latency.
- Facebook's Edge Network caches images, videos, and static content (like profile pictures and stories) on edge servers near users.
- This speeds up the loading time for media-heavy content and saves bandwidth.



ZippyDB (Key-Value Store)

- Developed by Facebook as a distributed key-value storage system
- Purpose: Provides low-latency data access for features like comments, notifications, and user status updates
- Architecture: Sharded design with automatic failover capabilities
- Performance: Sub-millisecond read/write operations at massive scale


Client
Client
1. put
1. get
4. response
2. response
Primary
async replication
root follower
2. accept
2. accept
3. accepted
3. accepted
Secondary
Secondary
Follower
Synchronous replication

BigPipe (Dynamic Web Page Rendering)

- Problem: Traditional web pages load sequentially, creating user-perceived latency
- Solution: BigPipe breaks pages into "pagelets" that load independently and in parallel
- Implementation: Server streams content as it becomes available rather than waiting for everything

Benefits:

  - Significantly faster perceived page loads
  - Progressive rendering of the News Feed and other content-heavy features
  - Better user experience particularly on slower connections
- Legacy: Influenced modern web development frameworks and practices

HipHop for PHP (HHVM)

Challenge: PHP's interpreted nature created server efficiency bottlenecks
Solution: HipHop transforms PHP code into highly optimized C++ then compiles it
Evolution:

- Started as a PHP-to-C++ translator
- Evolved into HHVM (HipHop Virtual Machine) - a JIT compiler
- Led to the development of Hack programming language

Impact:

- Reduced server CPU usage by up to 50%
- Enabled massive cost savings in Facebook's infrastructure
- Improved response times across the platform

Scribe (Distributed Logging System)

Purpose: Aggregates and processes log data across Facebook's global infrastructure
Scale: Handles petabytes of log data daily from thousands of servers worldwide
Architecture:

- Distributed collection agents on every server
- Fault-tolerant message queuing
- Configurable storage backends (HDFS, S3, etc.)

Applications:

- Real-time monitoring and alerting
- Performance optimization
- User behavior analysis
- Security incident detection

Thrift (Cross-Language RPC Framework)

Thrift is an **open-source Remote Procedure Call (RPC) framework** that enables **different programming languages to communicate** efficiently. It was originally developed at **Facebook** to solve the problem of **cross-language service communication**.

Thrift solves this problem by:

- Defining a **common interface** for communication.
- Using an **efficient binary protocol** for fast data exchange.
- Supporting **multiple languages** (C++, Java, Python, PHP, Go, etc.).

XHP (Secure HTML Rendering)

- XHP **prevents XSS vulnerabilities** by treating HTML as structured objects.
- It **automatically escapes user input**, making PHP-HTML integration safer.
- Used by **Facebook** to enhance security in **large-scale applications**.

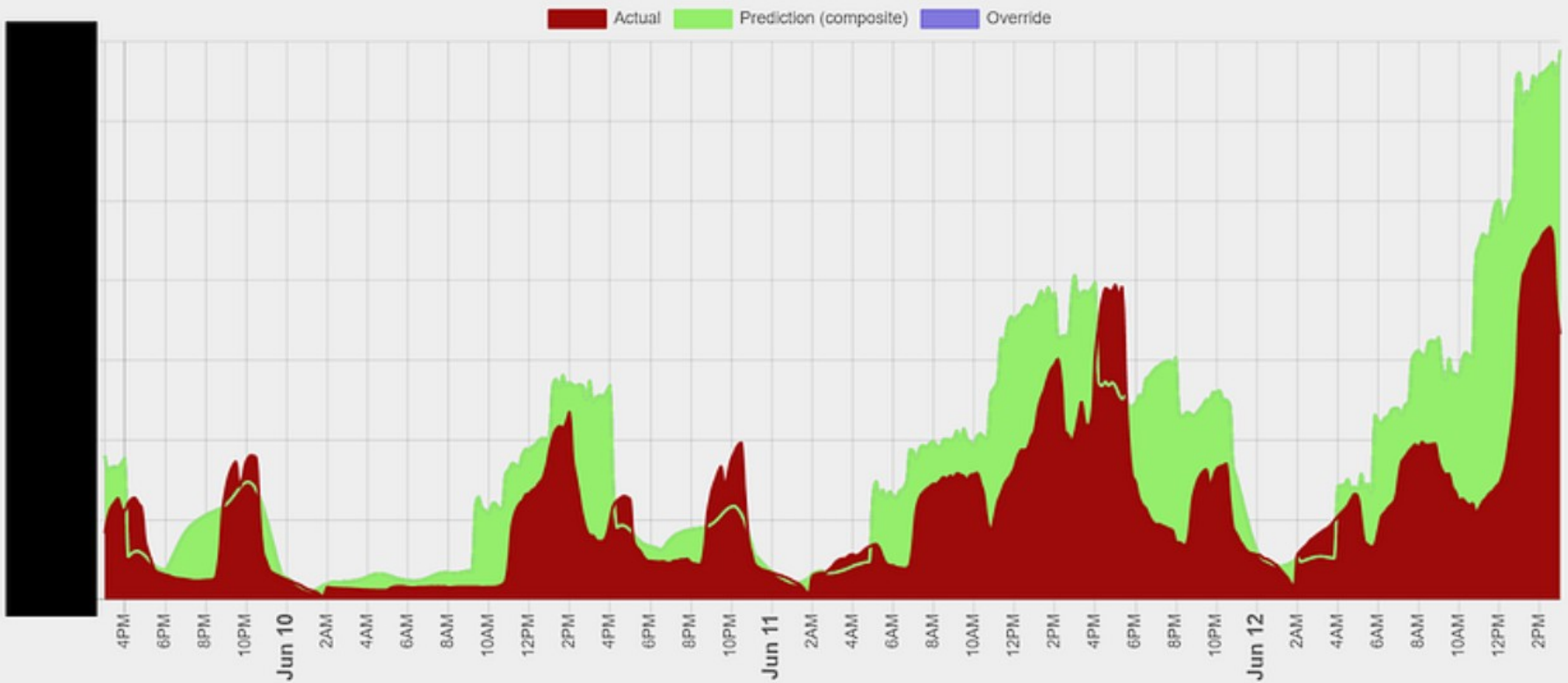
If a hacker enters <script>alert('Hacked!')</script> as the username.

```
$userName = htmlspecialchars($_POST['name']);
echo "<div>Hello, " . $userName . "!</div>";
```

Autoscale (Dynamic Scaling)

Autoscale is a system that **automatically adjusts the number of active servers** based on **real-time traffic demands**. Instead of keeping a fixed number of servers running at all times, **autoscaling dynamically increases or decreases resources** to ensure optimal performance and cost-efficiency.

1. **Run too many servers** all the time, wasting money.
2. **Run too few servers**, leading to slow response times or crashes during high traffic.

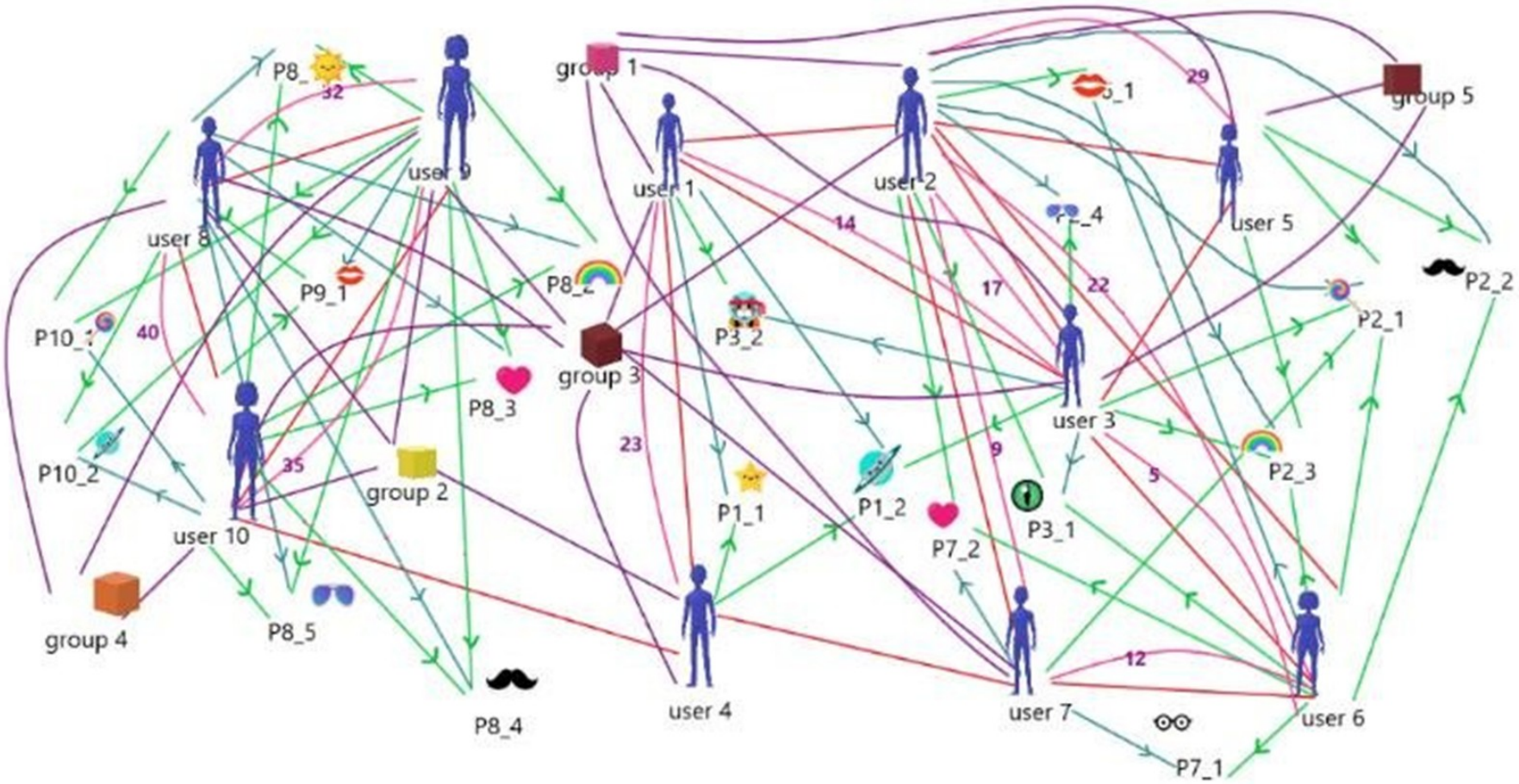

Actual
Prediction (composite)
Override
4PM
6PM
8PM
10PM
Jun 10
2AM
4AM
6AM
8AM
10AM
12PM
2PM
4PM
6PM
8PM
10PM
Jun 11
2AM
4AM
6AM
8AM
10AM
12PM
2PM
4PM
6PM
8PM
10PM
Jun 12
2AM
4AM
6AM
8AM
10AM
12PM
2PM

Social Graph

A **social graph** is a **digital map of relationships** between people,objects, and interactions in a social network. It represents how users are connected through **friendships, likes, comments, and other interactions**.

At **Facebook**, the social graph is stored and managed using **TAO**

A social graph consists of **nodes** and **edges.**

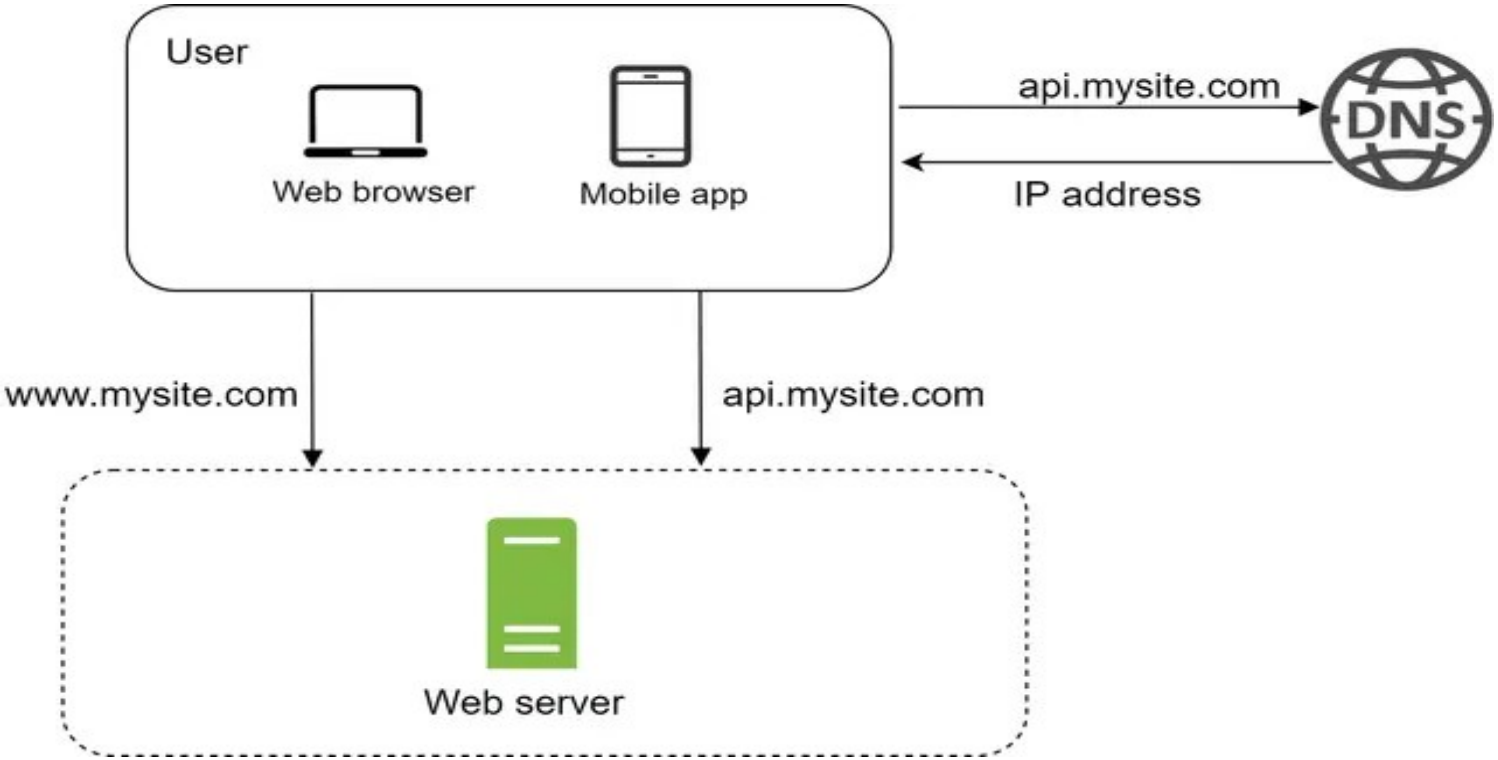

P8_
32
user 9
group 1
user 1
user 2
29
group 5
user 8
14
user 5
P9_1
P8_2
17
22
P2_2
P10_1
40
P3_2
P2_1
group 3
P8_3
user 3
P10_2
35
23
9
5
P2_3
group 2
P1_1
P1_2
P3_1
P7_2
user 10
group 4
P8_5
12
P8_4
user 4
user 7
user 6
P7_1

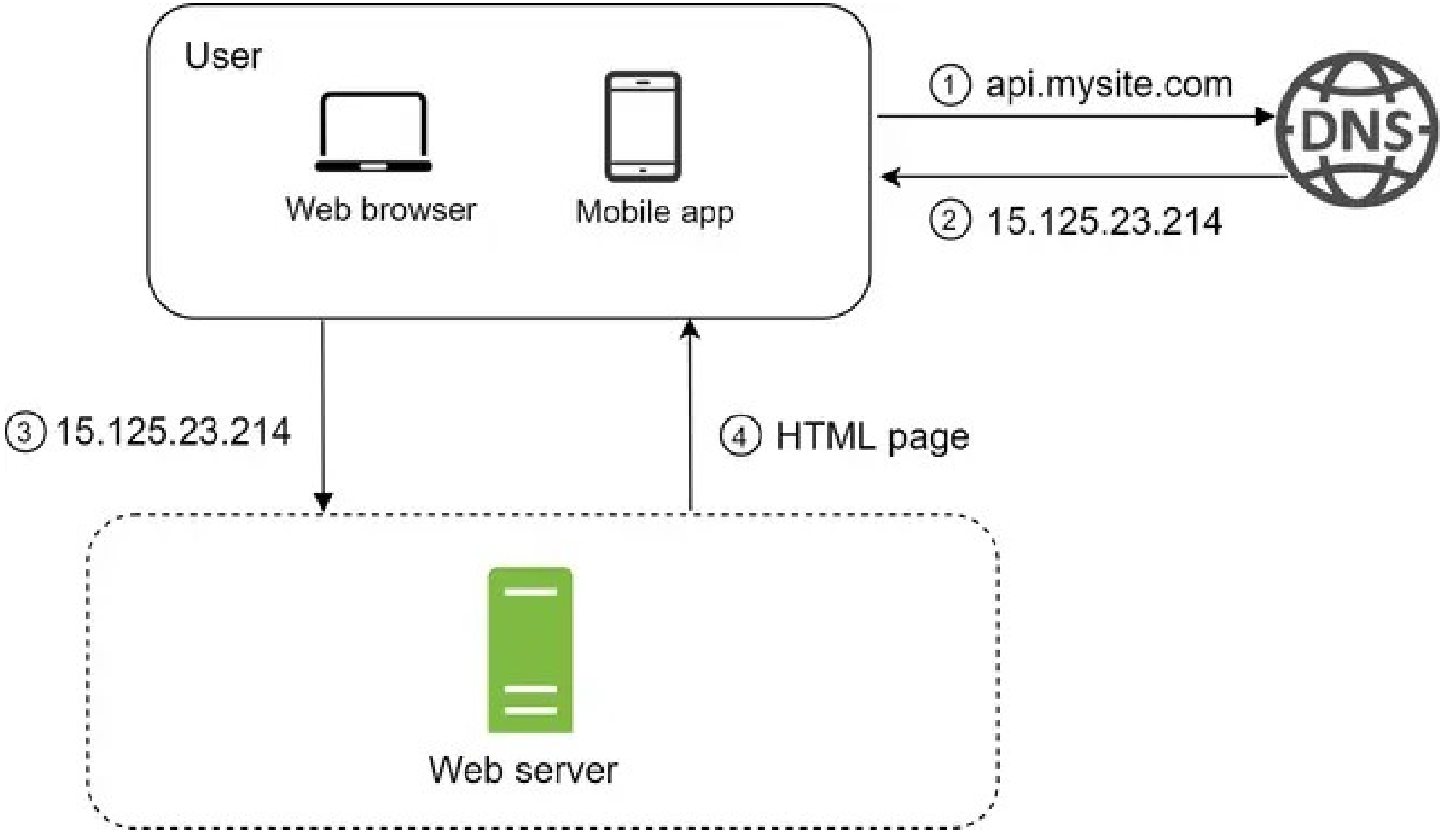
Scale From Zero To Millions Of Users

Designing a System like Facebook

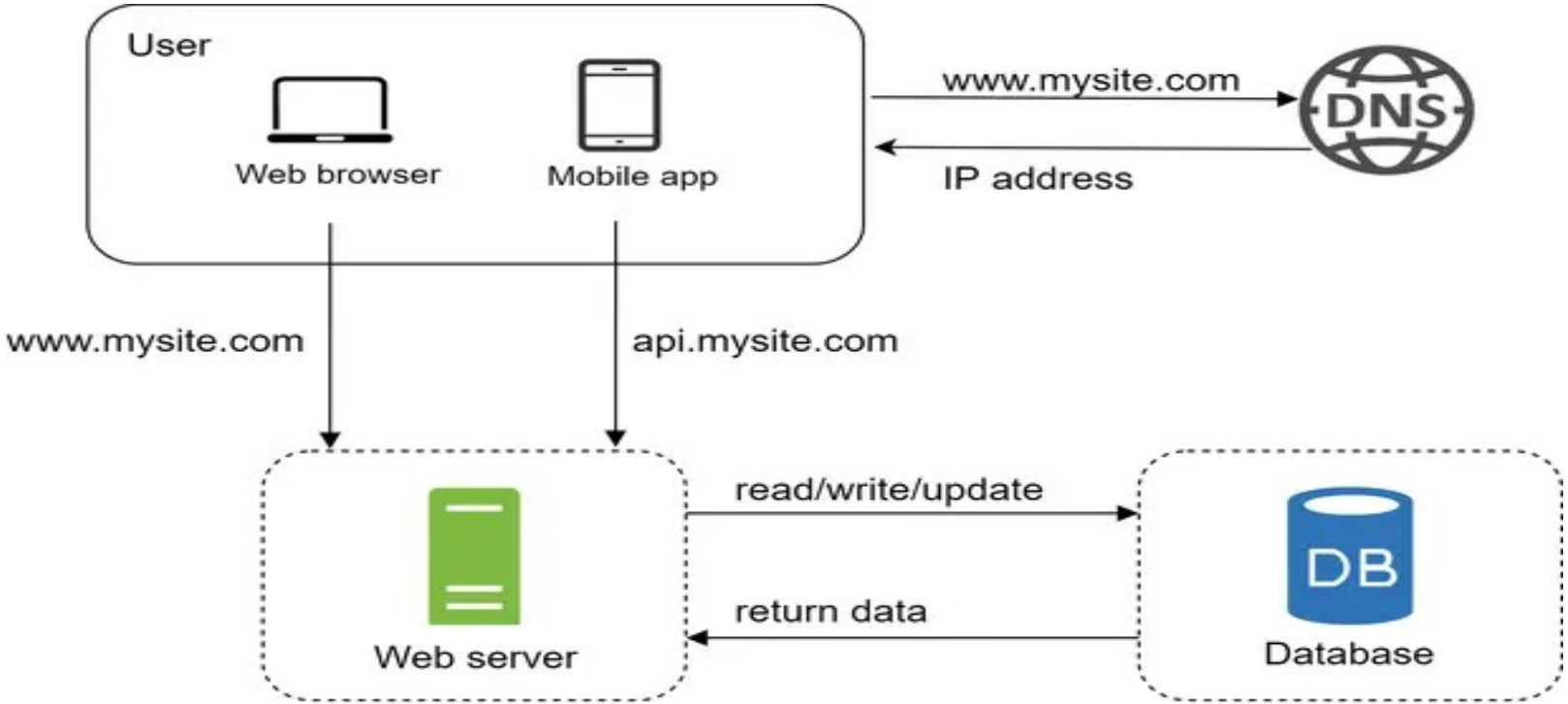
Santhosh T K
22z433

Single server setup




User
Web browser
Mobile app
① api.mysite.com
DNS
② 15.125.23.214
③ 15.125.23.214
④ HTML page
Web server

Database


User
Web browser
Mobile app
www.mysite.com
DNS
IP address
www.mysite.com
api.mysite.com
read/write/update
return data
Web server
DB
Database

Which DB to use Relational (or) Non-Relational ? (Design Decision)

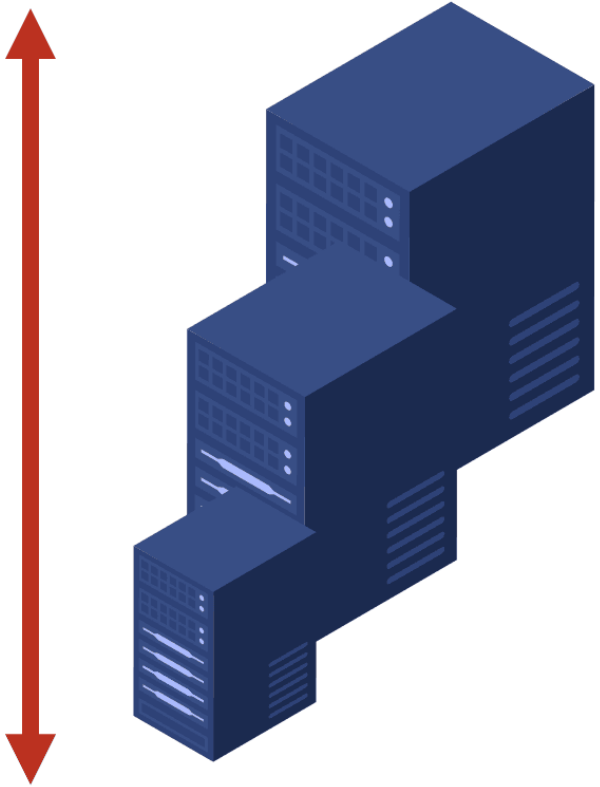
Go for Non-Relational if

- Need super low latency
- No relationship among data
- Need only serialization and deserialization
- Massive amount of data

If none of the above doesn’t matter to you, then go with relational.

Vertical Scaling

Increase or decrease the capacity of existing services/instances.

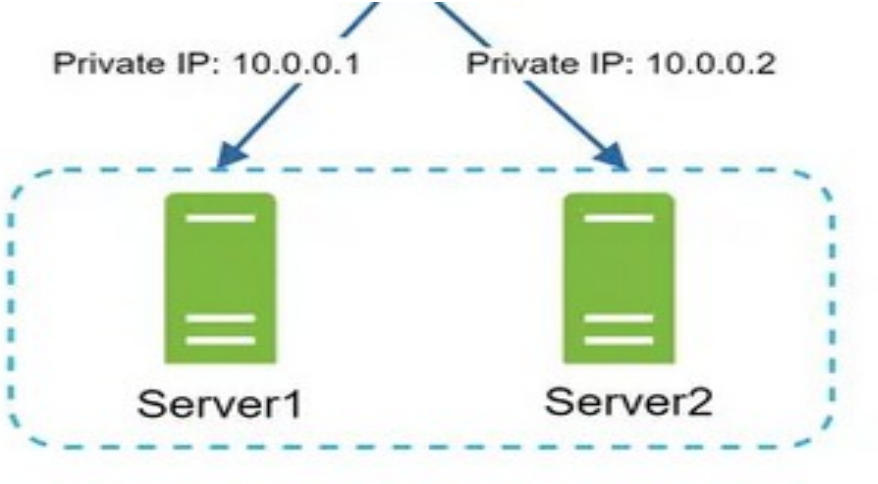


Horizontal Scaling

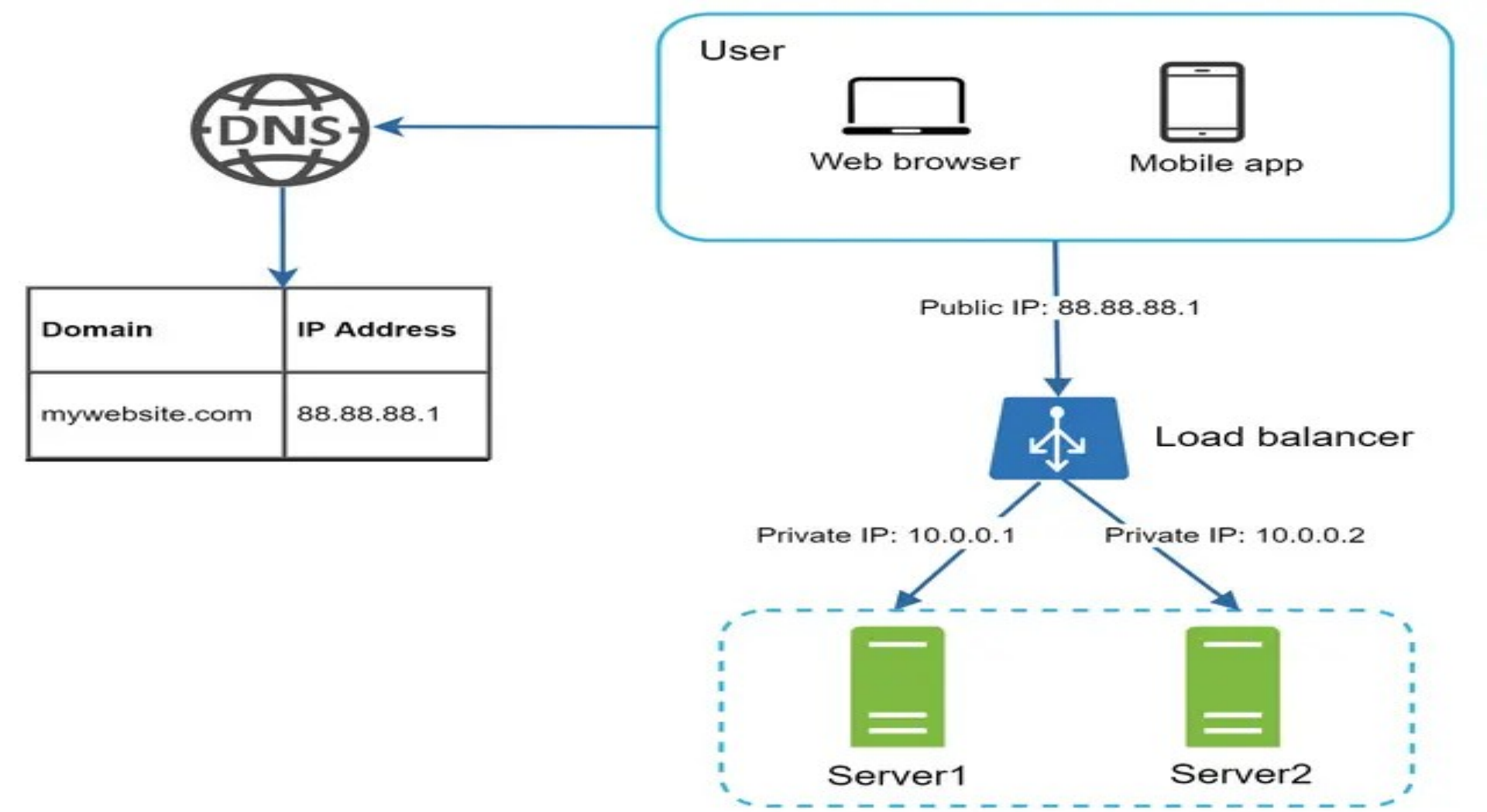
Add more resources like virtual machines to your system to spread out the workload across them.

Increased requests to the servers. What to do ?

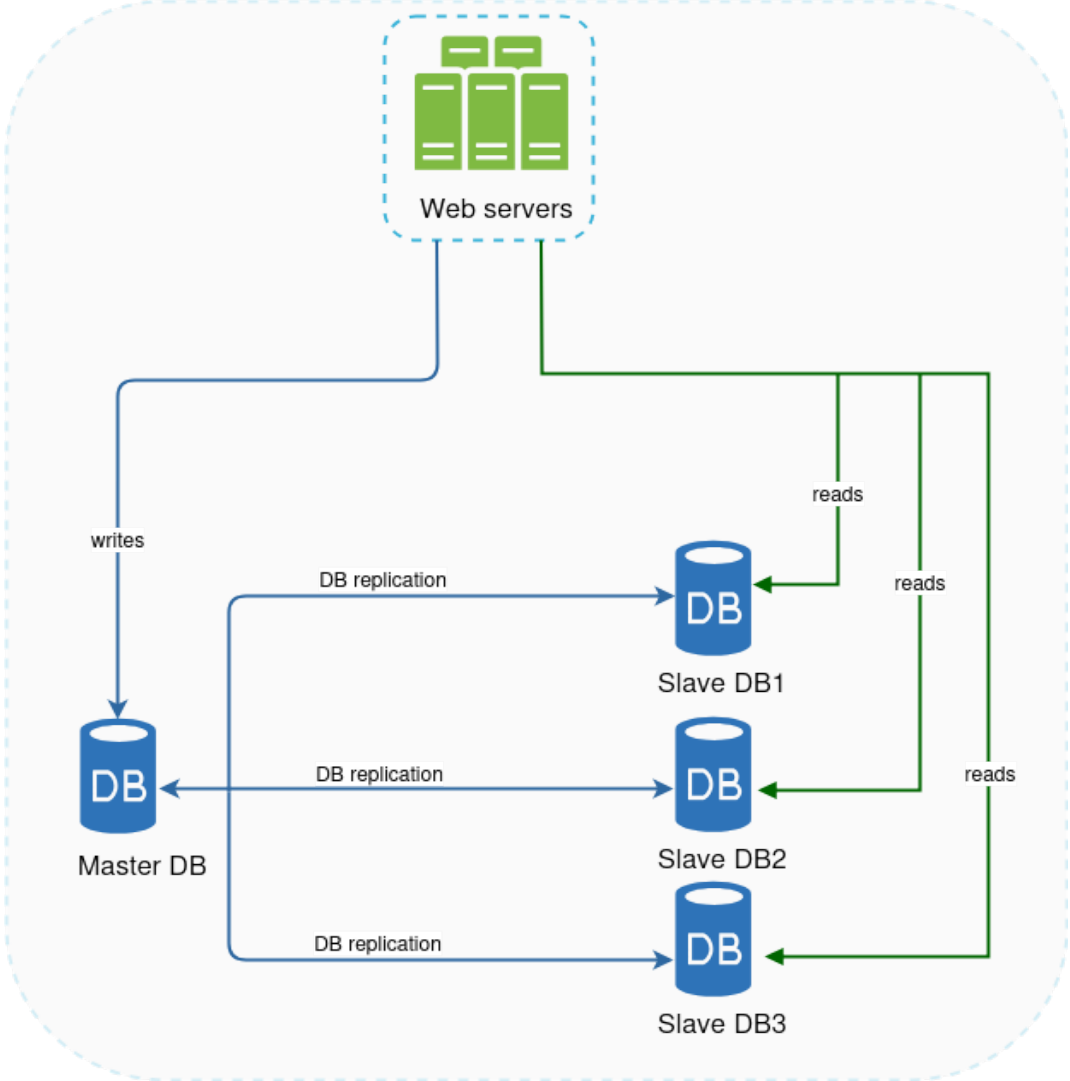
Design Decision : Horizontal Scaling

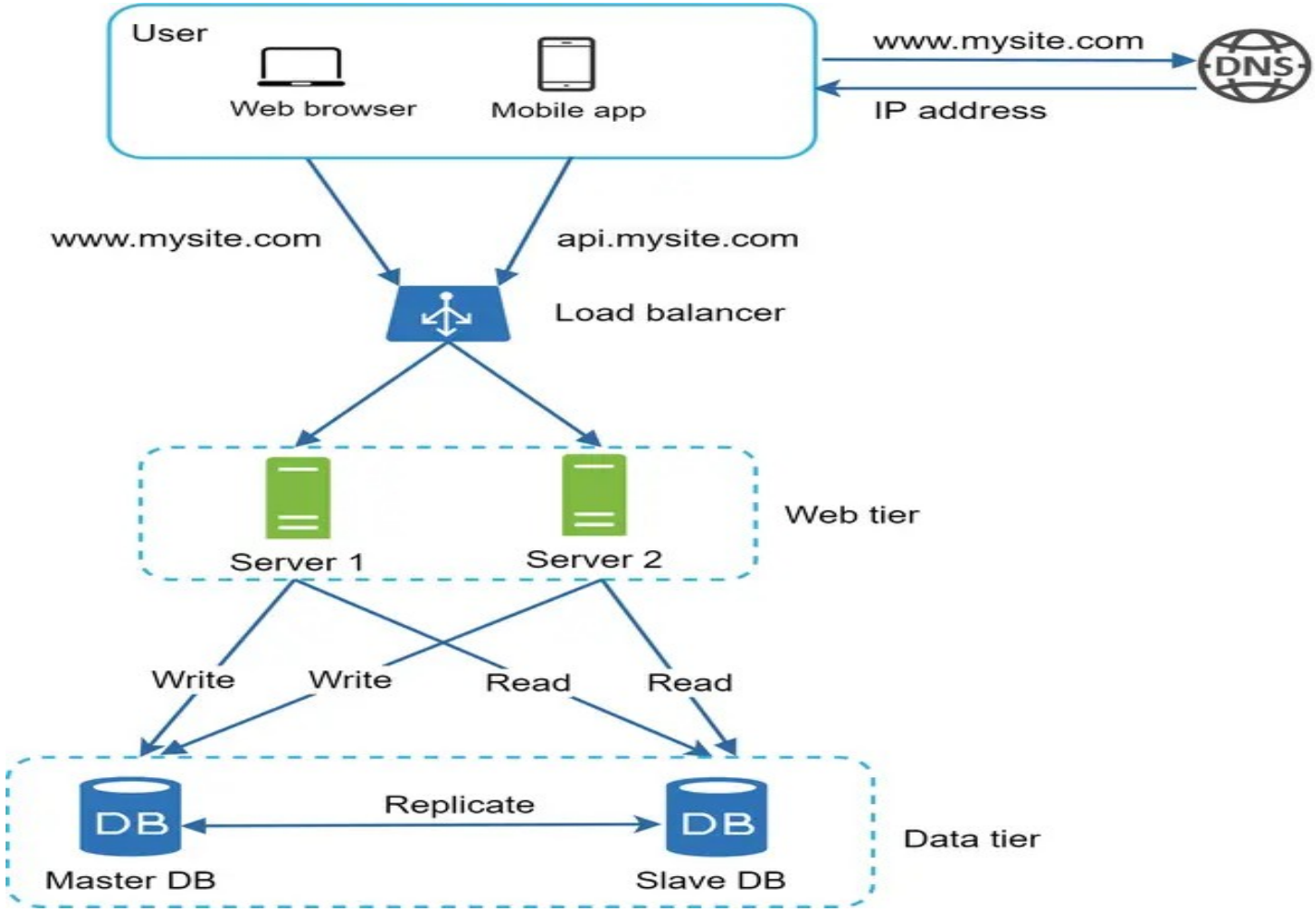


Which server should handle the request ?


User
Web browser
Mobile app
DNS
Domain
IP Address
mywebsite.com
88.88.88.1
Public IP: 88.88.88.1
Load balancer
Private IP: 10.0.0.1
Private IP: 10.0.0.2
Server1
Server2

Is single database enough ?


Web servers
writes
reads
reads
reads
DB replication
DB replication
DB replication
DB
DB
DB
DB
Master DB
Slave DB1
Slave DB2
Slave DB3


User
Web browser
Mobile app
www.mysite.com
DNS
IP address
www.mysite.com
api.mysite.com
Load balancer
Server 1
Server 2
Web tier
Write
Write
Read
Read
DB
Replicate
DB
Master DB
Slave DB
Data tier

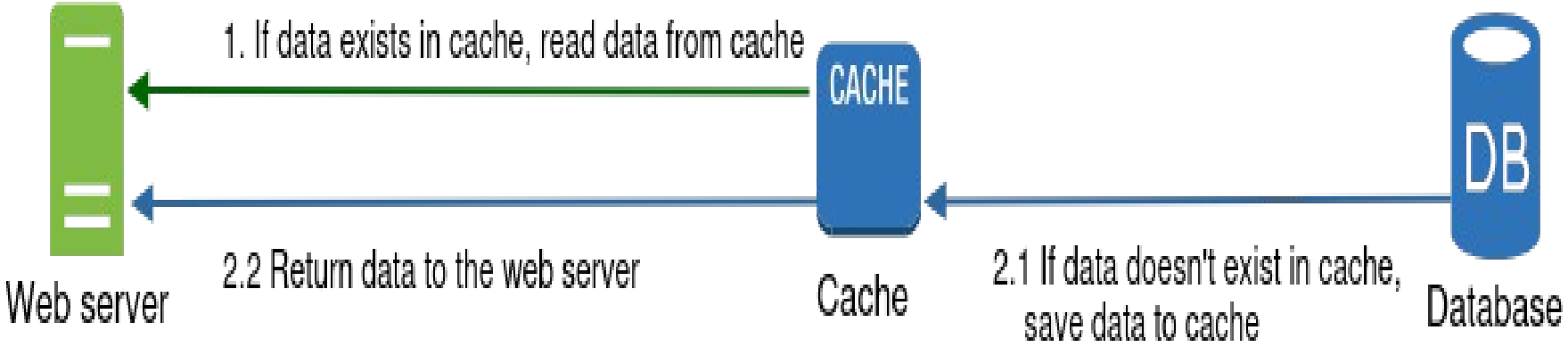
We need to add something, Guess what ?

Hint :

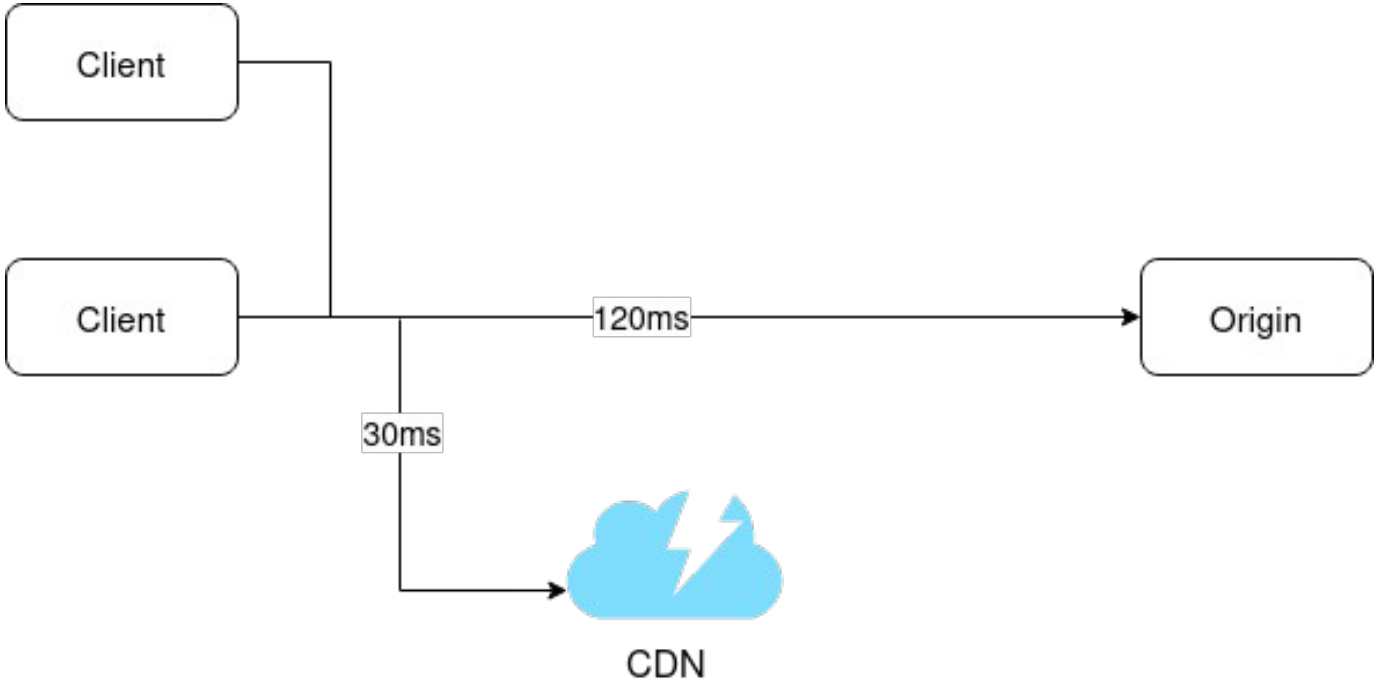
1. Trending Songs (frequent access to the same song by different users)
2. Breaking News (frequent access to the news)

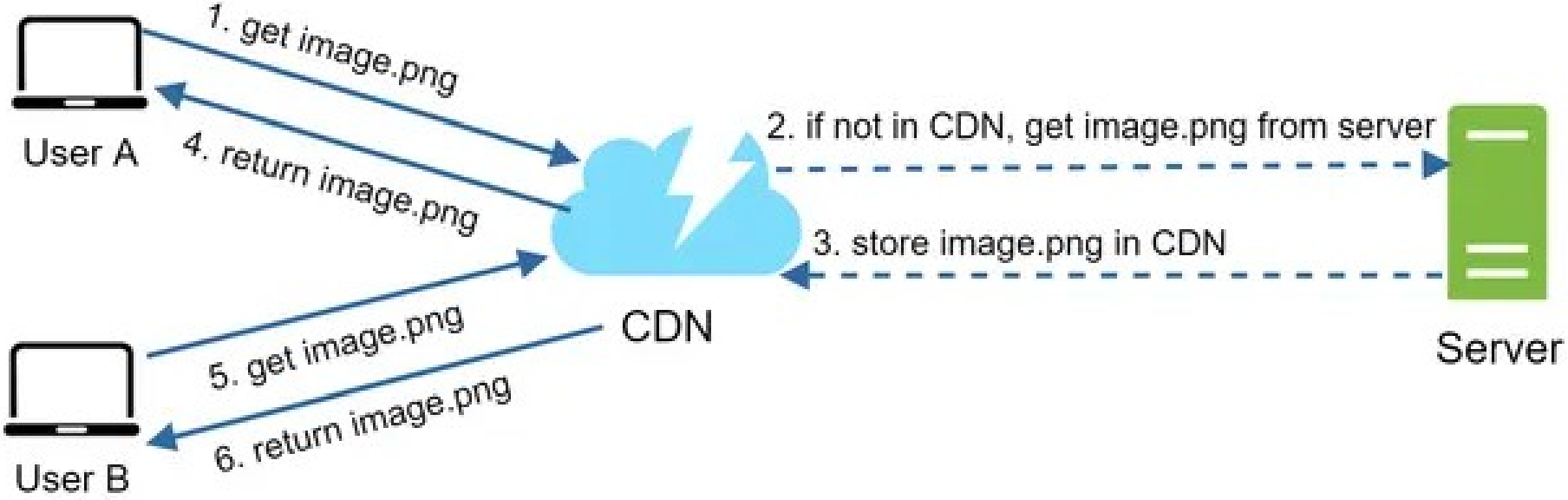
Got it right !!!

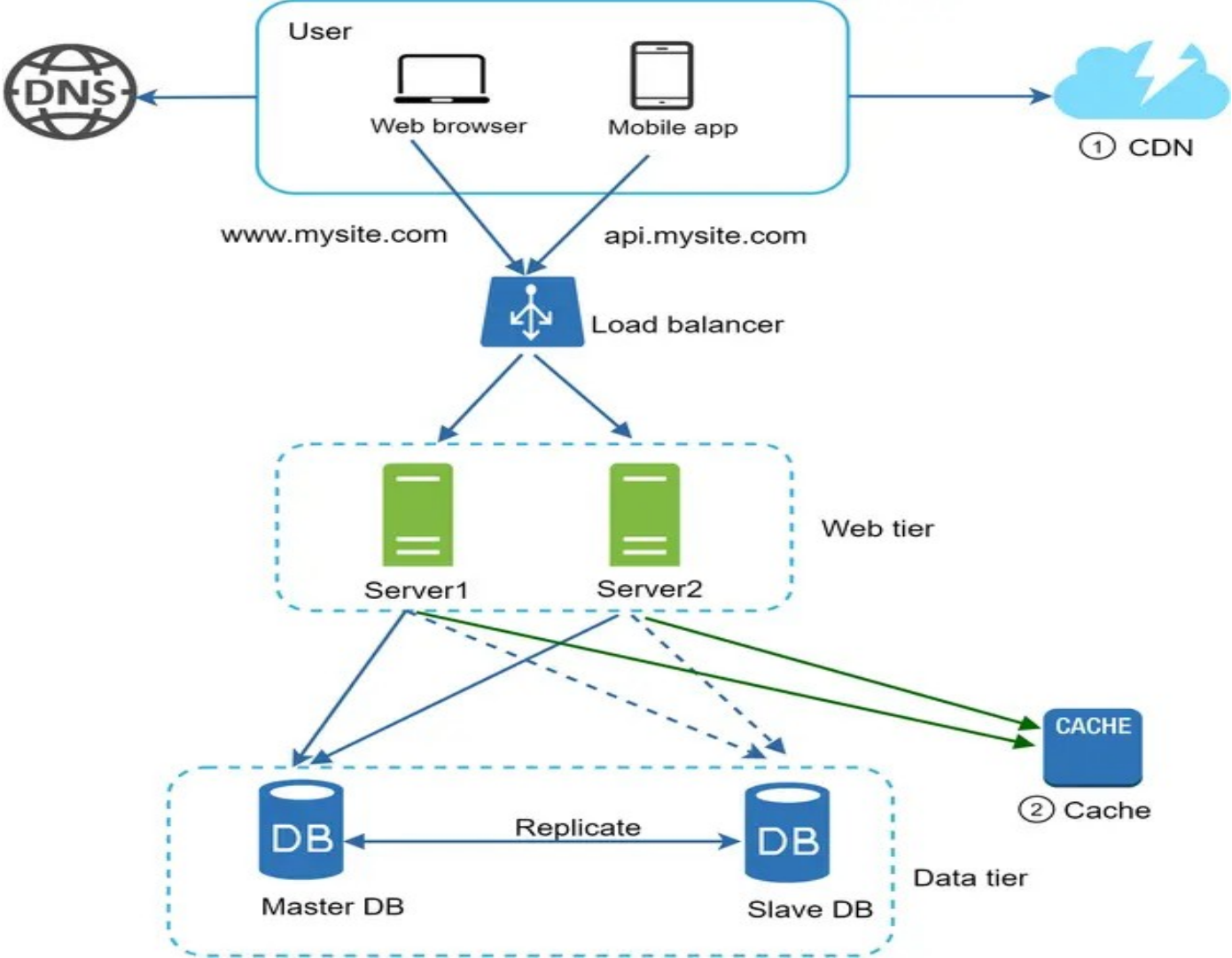
Cache tier


1. If data exists in cache, read data from cache
CACHE
DB
2.2 Return data to the web server
2.1 If data doesn't exist in cache, save data to cache
Web server
Cache
Database

Content delivery network (CDN)

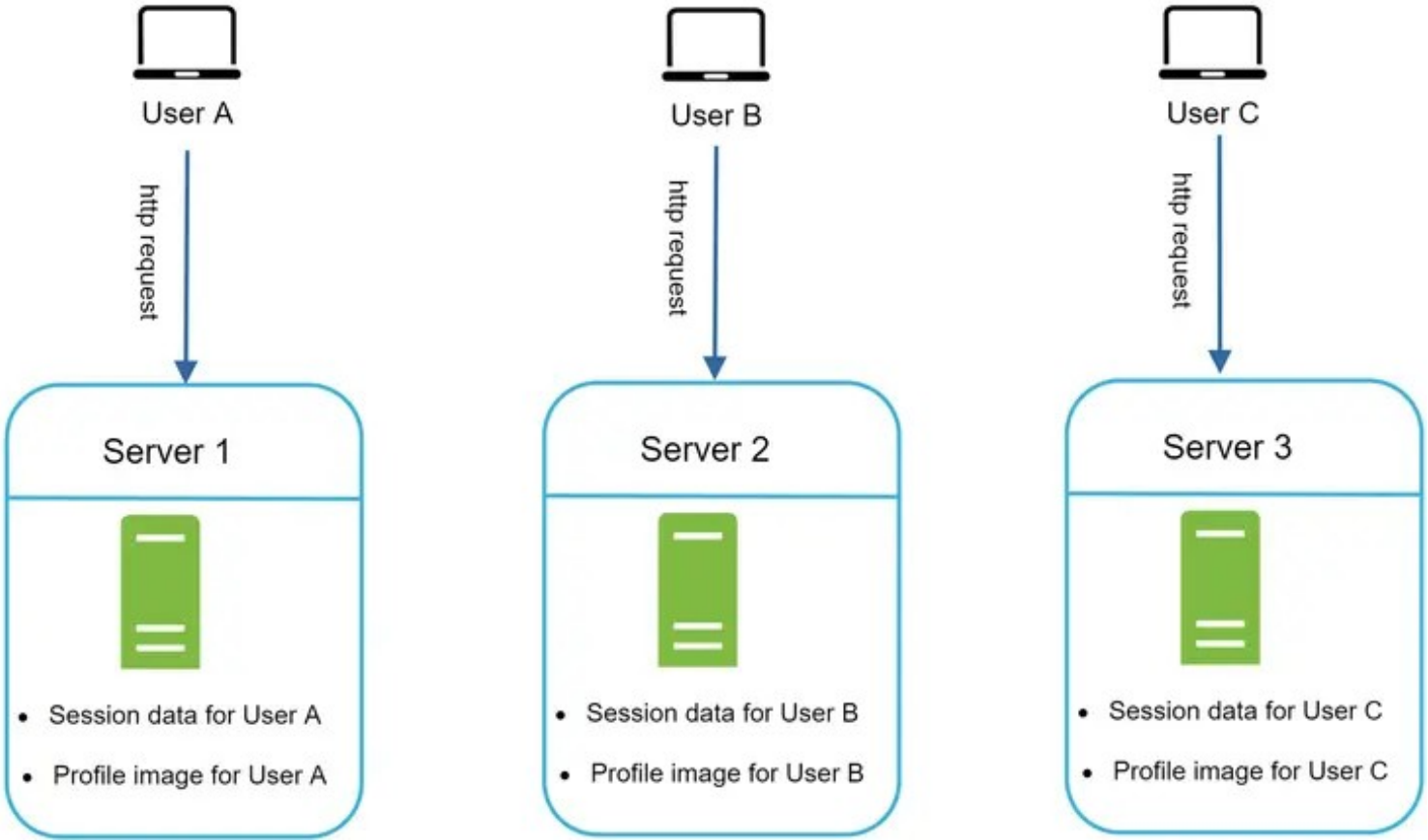



1. get image.png
User A
4. return image.png
2. if not in CDN, get image.png from server
3. store image.png in CDN
CDN
Server
5. get image.png
6. return image.png
User B

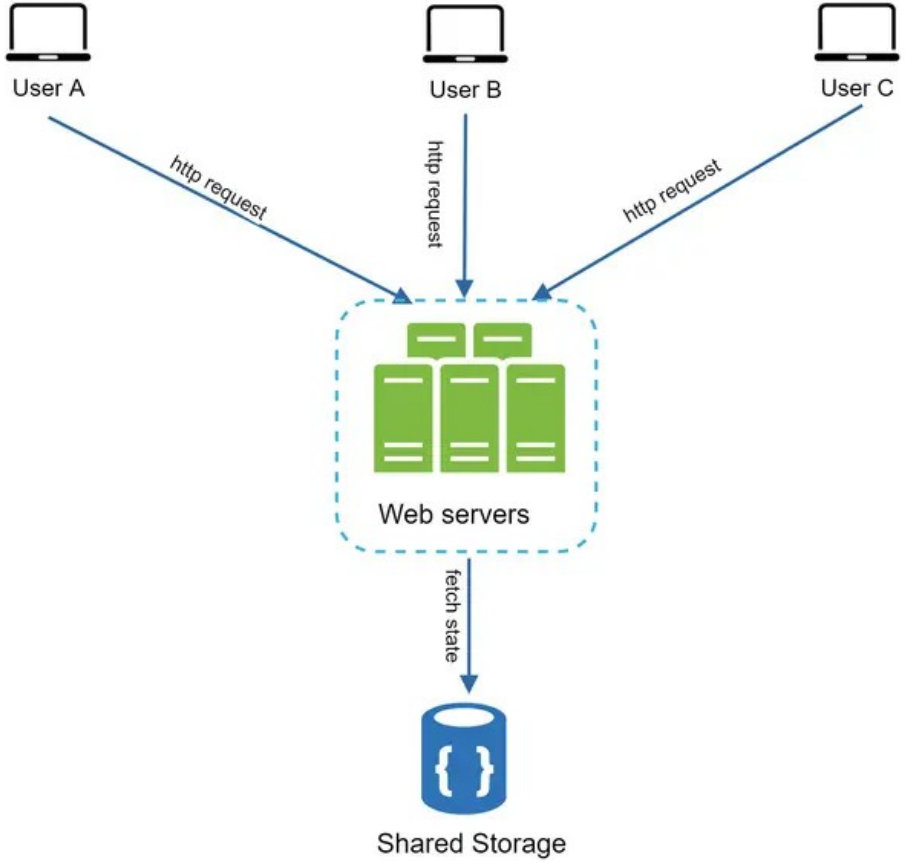

User
DNS
Web browser
Mobile app
1 CDN
www.mysite.com
api.mysite.com
Load balancer
Server1
Server2
Web tier
CACHE
2 Cache
DB
Replicate
DB
Master DB
Slave DB
Data tier

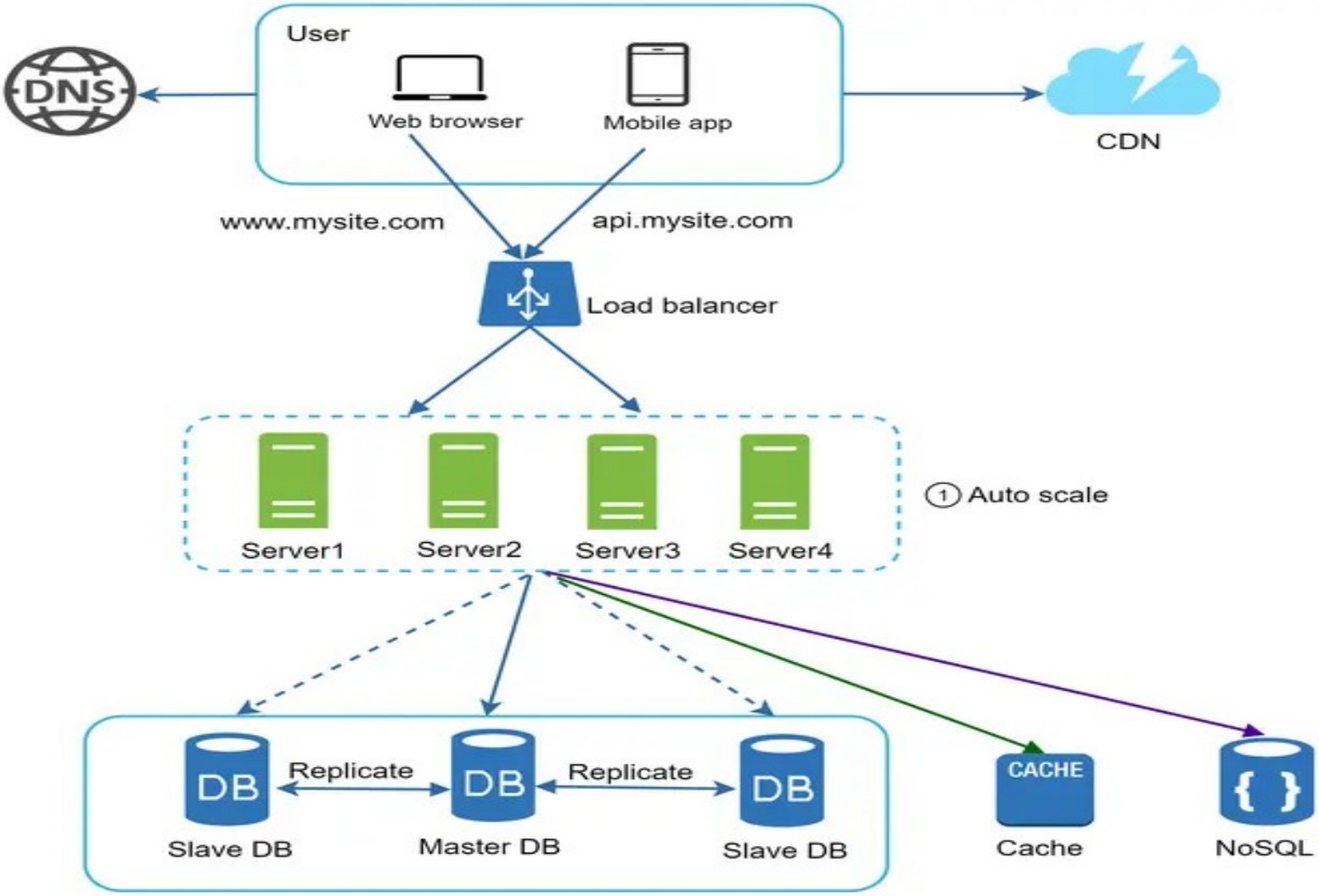
Stateless Web Tier

Stateful Architecture

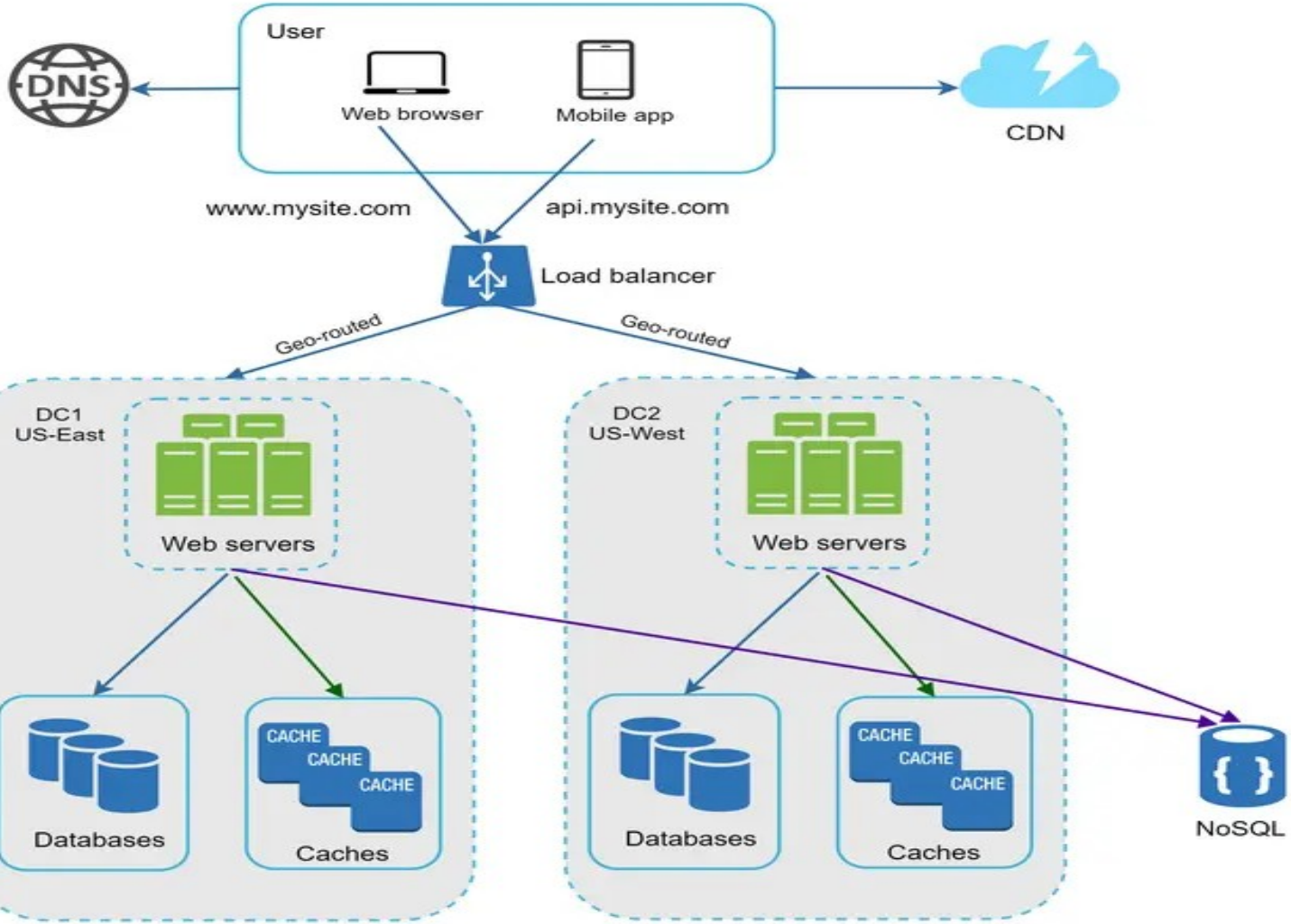


Stateless Architecture

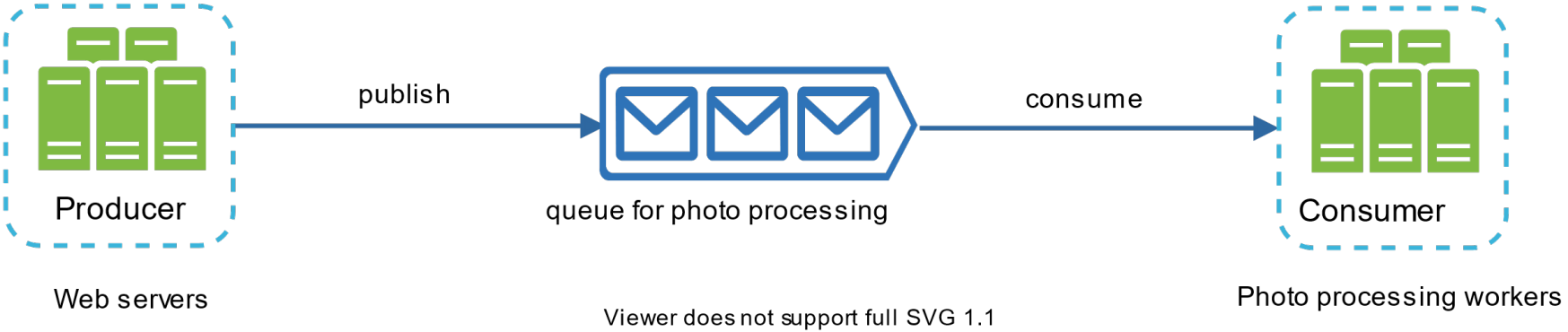


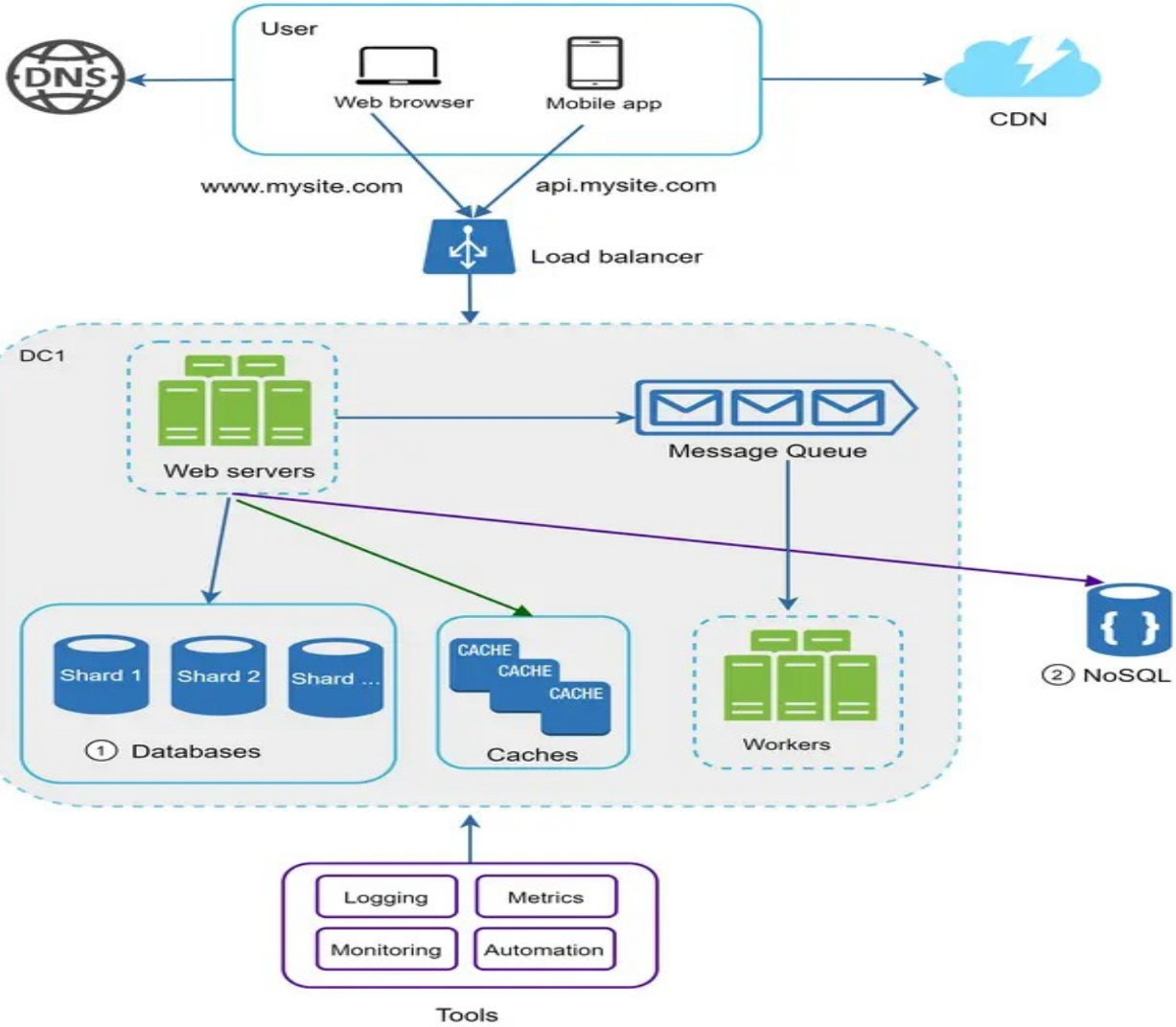

User
DNS
Web browser
Mobile app
CDN
www.mysite.com
api.mysite.com
Load balancer
Server1
Server2
Server3
Server4
① Auto scale
DB
Replicate
DB
Replicate
DB
Slave DB
Master DB
Slave DB
CACHE
Cache
{ }
NoSQL

Data Centers


User
DNS
Web browser
Mobile app
CDN
www.mysite.com
api.mysite.com
Load balancer
Geo-routed
Geo-routed
DC1
US-East
Web servers
DC2
US-West
Web servers
Databases
CACHE
CACHE
CACHE
Caches
Databases
CACHE
CACHE
CACHE
Caches
NoSQL

Message Queues for Async Communication




User
DNS
Web browser
Mobile app
CDN
www.mysite.com
api.mysite.com
Load balancer
DC1
Web servers
Message Queue
Shard 1
Shard 2
Shard ...
1 Databases
CACHE
CACHE
CACHE
Caches
Workers
2 NoSQL
Logging
Metrics
Monitoring
Automation
Tools

References

[1]. Scale From Zero To Millions Of Users: https://bytebytego.com/courses/system-design-interview/scale-from-zero-to-millions-of-users

[2]. R. Nishtala, "Facebook, Scaling Memcache at," 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI ’13).

[3]. What The Heck Are You Actually Using NoSQL For :http://highscalability.com/blog/2010/12/6/what-the-heck-are-you-actually-using-nosql-for.html


Thank
you!