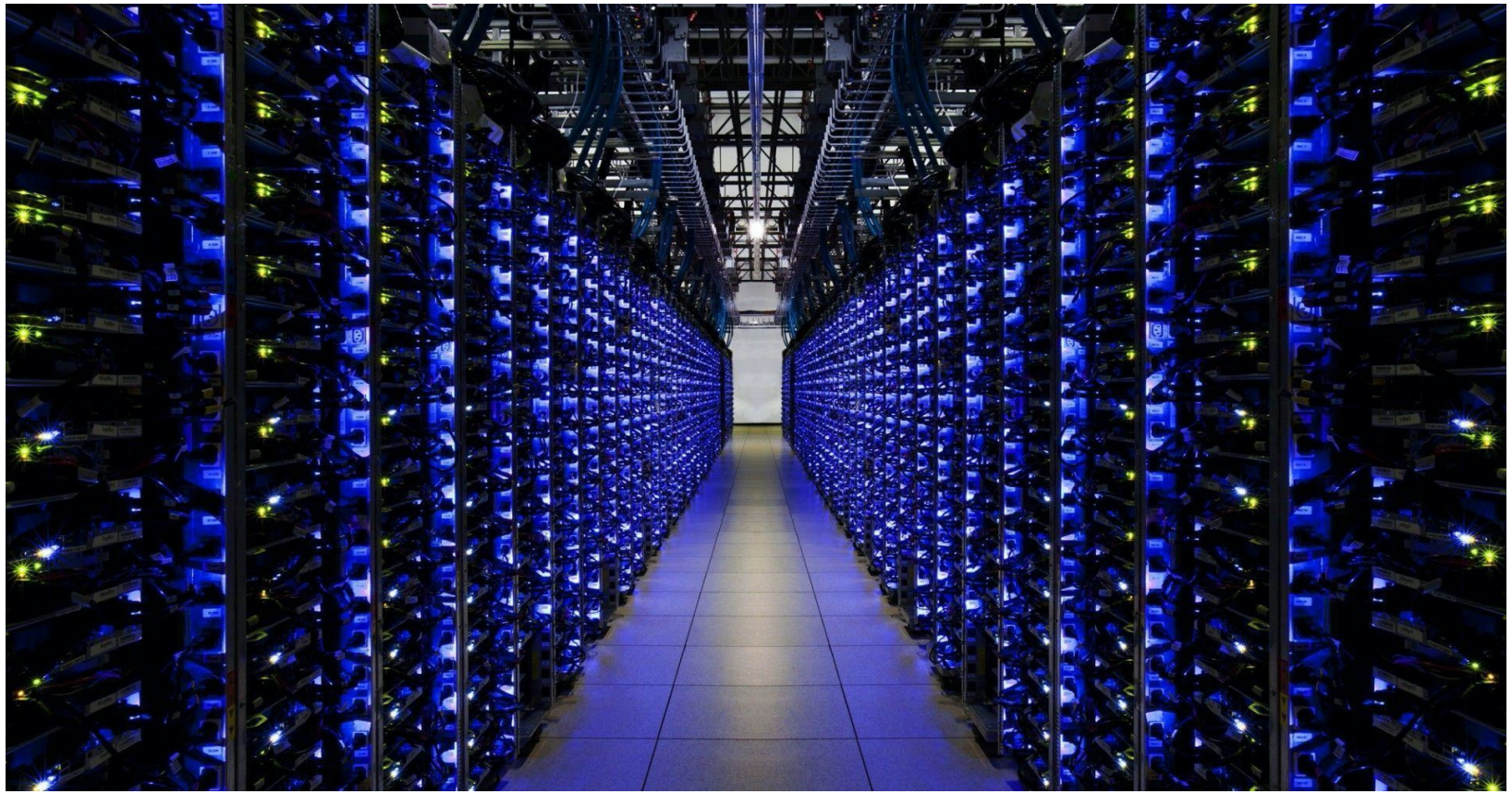
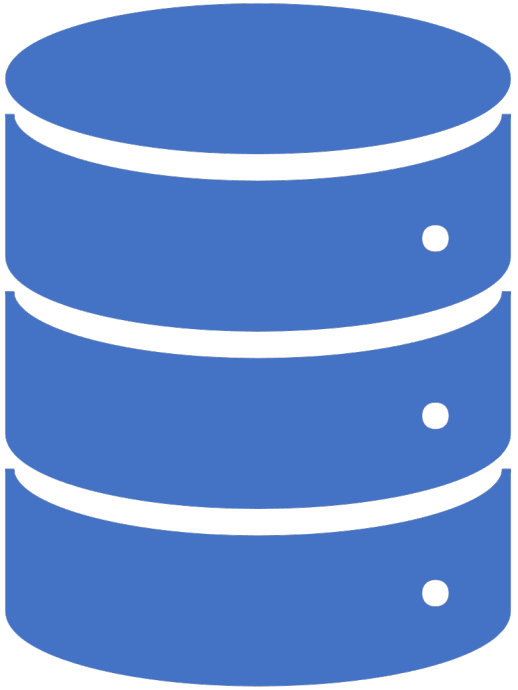
Database Scalability

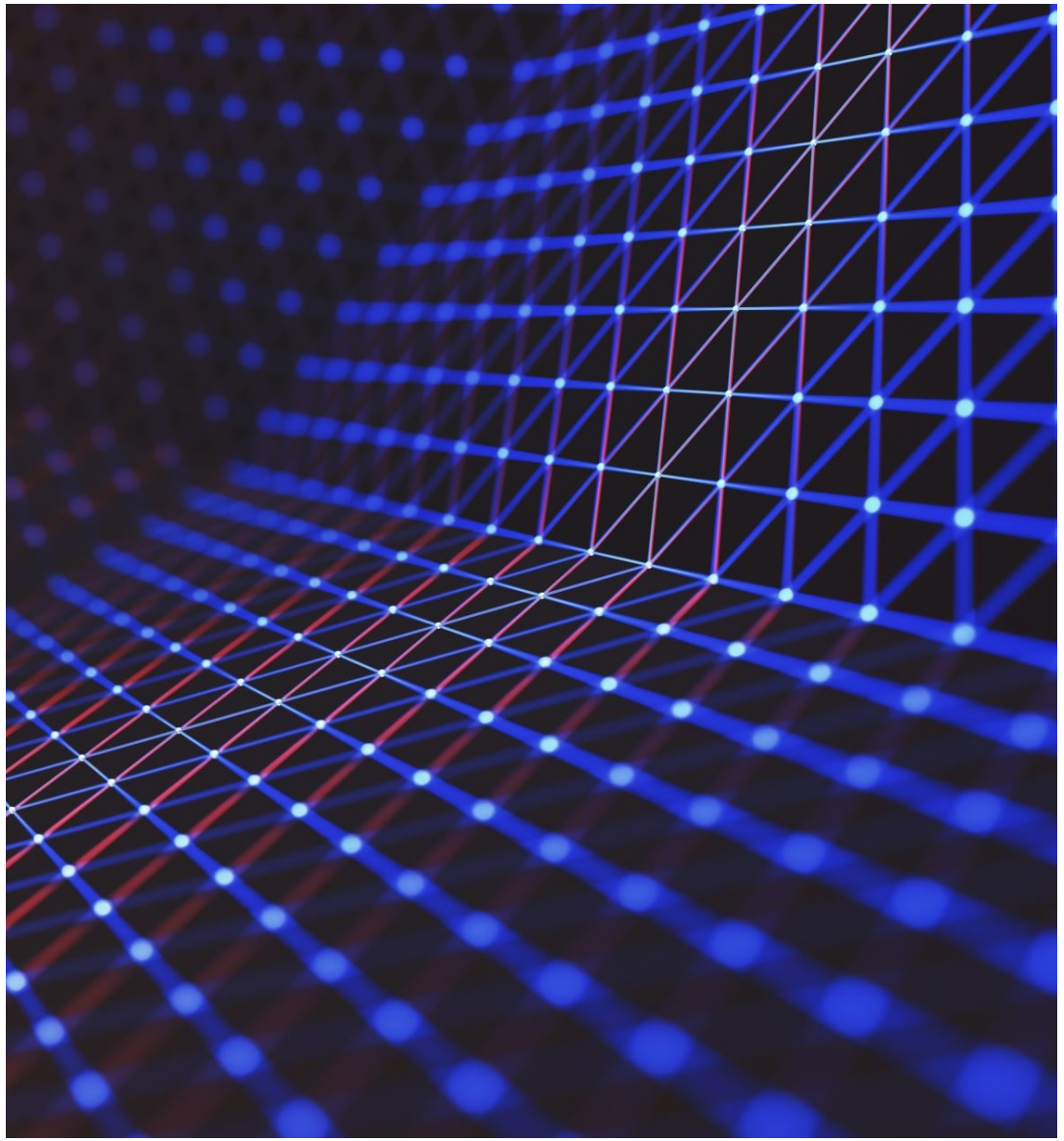
Huh, too much data!! Figurin where to keep em!!



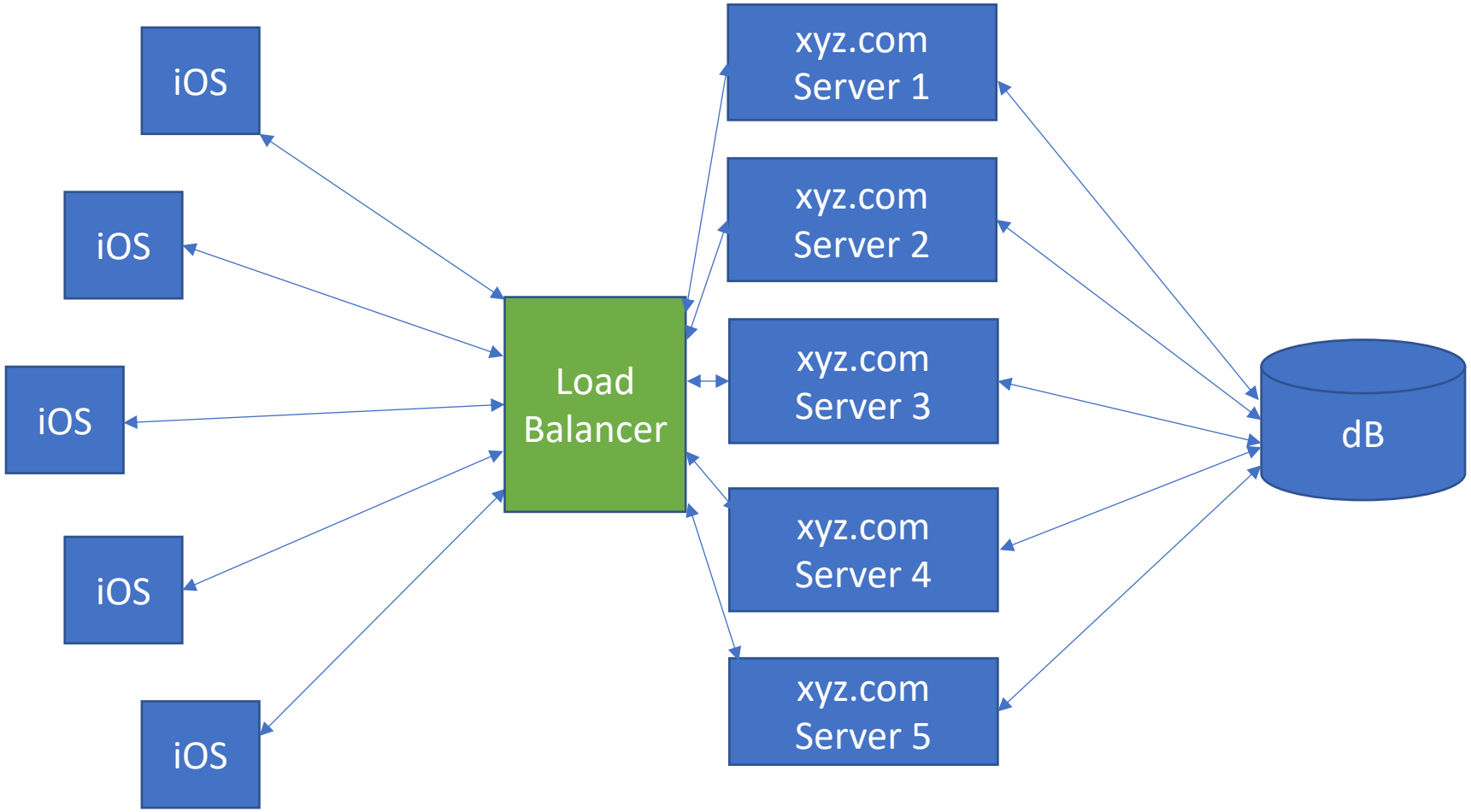


What we will be learning ?

1. Recap
2. Database Partitioning
3. Sharding
4. Database Replication



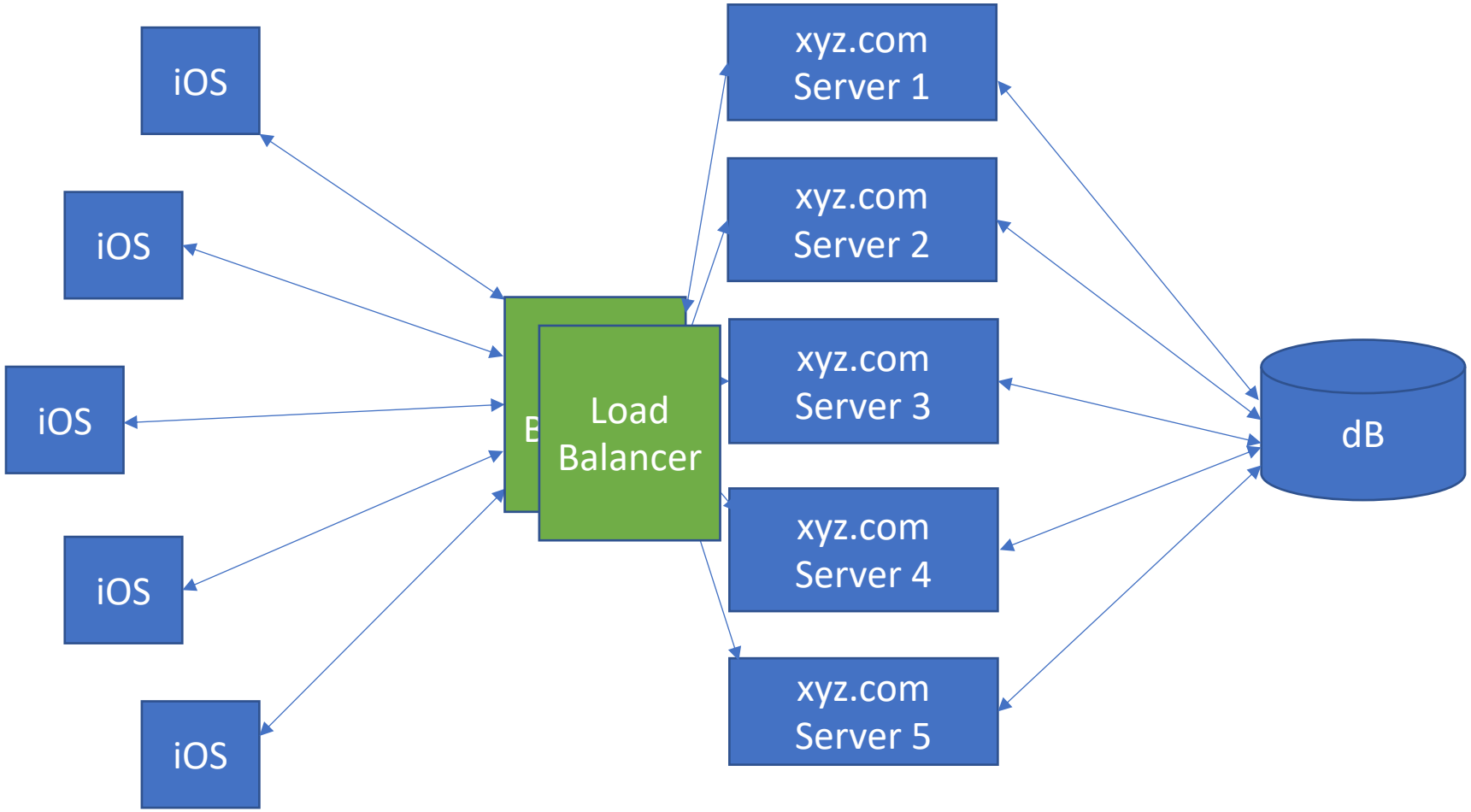
1. Recap


iOS
iOS
iOS
iOS
iOS
Load Balancer
xyz.com Server 1
xyz.com Server 2
xyz.com Server 3
xyz.com Server 4
xyz.com Server 5
dB

1. Recap

1. We spent some time trying to **scale up the servers** to handle client requests.
2. We discussed few **load balancing techniques**.
3. Also we touched upon **server Heat beat**(UDP packet at 5 sec interval) which every server sends to the Load balancer.
4. We don’t have a **single point of failures** now. **Really ?**
5. What if **LB fails** ? ☹

1. Recap


iOS
iOS
iOS
iOS
iOS
Load Balancer
xyz.com Server 1
xyz.com Server 2
xyz.com Server 3
xyz.com Server 4
xyz.com Server 5
dB

1. Concerns with existing dB

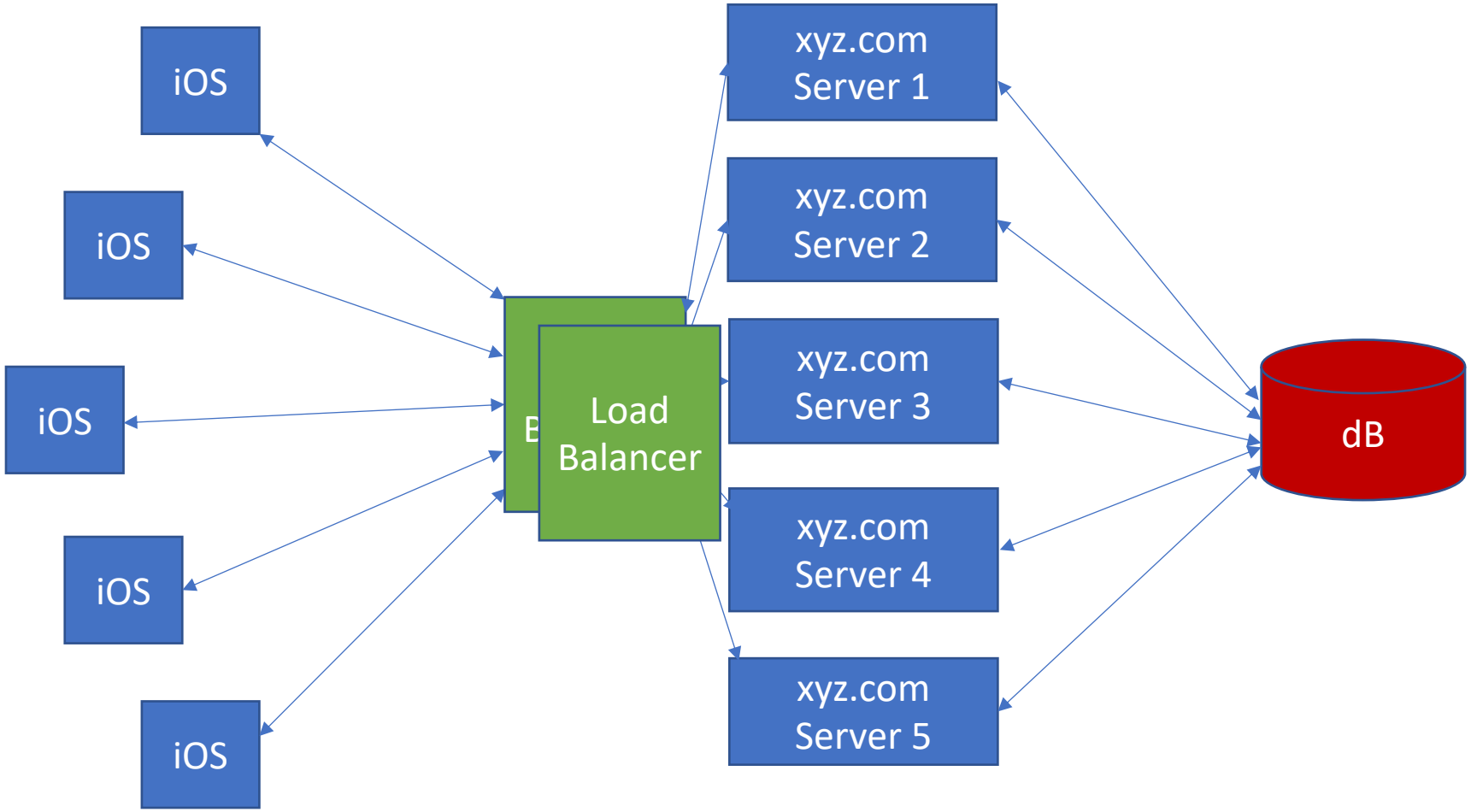
1. Lets try to execute a **SELECT** query into **Flights table** and analyze the execution time for our **postGres dB.**

2. **Execution time** for dB query **increases** as the number of rows/columns start to grow.

1. Concerns with existing dB

1. What happens if multiple servers **concurrently query** the same dB ?
2. What happens if we have **>1 Million rows/columns** in our dB table and we perform a **query operation** ?

1. Concerns with existing dB

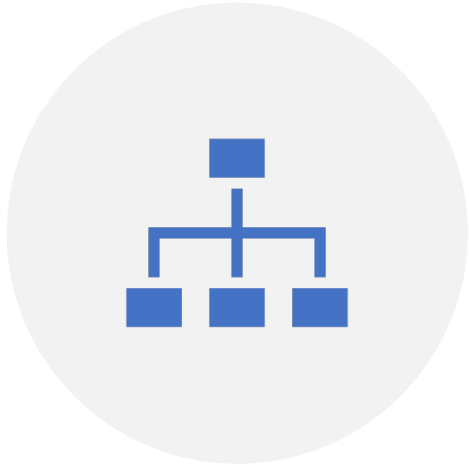


2. Database Partitioning

2. Database Partitioning

1. Trying to query information from single **large table** which is large in terms of no. of rows/columns gets **complicated** and **time-consuming.**
2. We split the table into more **manageable units**(e.g. tables) to have the queries on those units run more **efficiently** and **quickly**. This idea is **Database Partitioning.**

2. Database Partitioning Methods



1. VERTICAL PARTITIONING

2. HORIZONTAL PARTITIONING

2.1 Vertical Partitioning

2.1 Vertical Partitioning

1. The idea is to separate a dB table into multiple different tables by **decreasing the number of columns** in those tables based on **usage Pattern(**Slow Moving data/dynamic data).
2. Vertical Partitioning goes **beyond normalization** and splits even a normalized table.

2.1 Vertical Partitioning

PK=id	airline	origin	origin_code	destination	destination_code	duration(min)
1	JetBlue Airways	New York	JFK	London	LHR	415
2	Southwest Airlines	Shanghai	PVG	Paris	CDG	760
3	American Airlines	Istanbul	IST	Tokyo	NRT	700
4	Southwest Airlines	New York	JFK	Paris	CDG	435
5	Delta Airlines	Moscow	SVO	Paris	CDG	245
6	JetBlue Airways	Lima	LIM	New York	JFK	455
7	Delta Airlines	Lima	LIM	Atlanta	ATL	320
8	JetBlue Airways	Los Angeles	LAX	Atlanta	ATL	435
9	American Airlines	Chicago	ORD	Atlanta	ATL	651
10	Delta Airlines	Los Angeles	LAX	Boston	MA	632

2.1 Vertical Partitioning

1. We use the concept of **normalization** to vertically partition the dB table.
2. **Normalization**: The idea is to stop capturing redundant data in our table.
3. We split our table based on **functional dependencies** in our data.
4. **Functional dependency**: A **constraint** between two attributes of a table. If an attribute X is functionally dependent on attribute Y, it means for every X we have **exactly one** Y. $X \rightarrow Y$

2.1 Vertical Partitioning

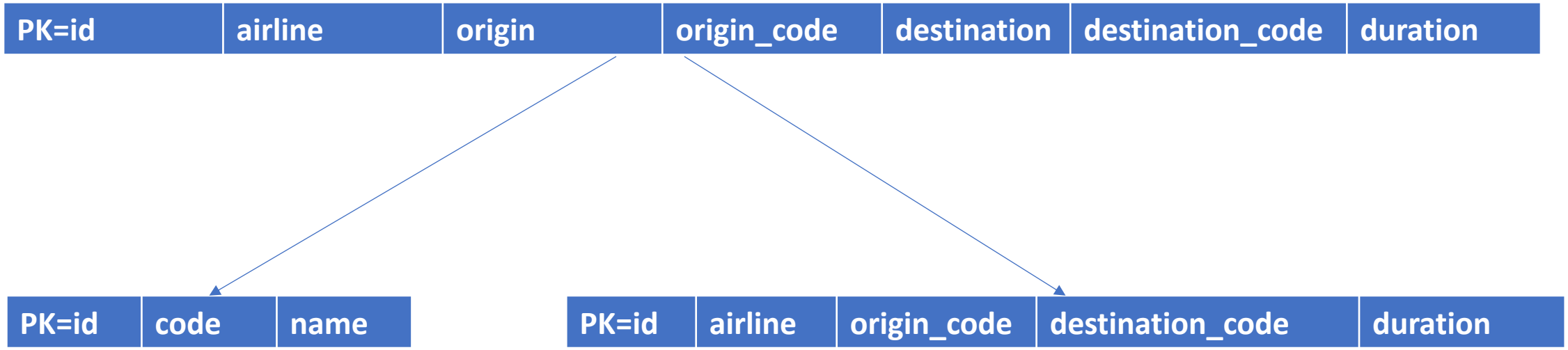
Step 1: We figure out **functional dependencies(single/Multi valued)** in our table data

a. origin_code → origin
b. destination_code → destination
c. Id → {airline, origin_code, destination_code, duration}

Step 2: Split the columns based on above functional dependencies

We use the idea of **Foreign keys** to vertically partition the table.

2.1 Vertical Partitioning



2.1 Vertical Partitioning

Locations(slow moving data)

PK=id	code	name
1	JFK	New York
2	PVG	Shanghai
3	IST	Istanbul
4	SVO	Moscow
5	LIM	Lima
6	LAX	Los Angeles
7	ORD	Chicago
8	MA	Boston
9	NRT	Tokyo
10	LHR	London
11	ATL	Atlanta
12	CDG	Paris

2.1 Vertical Partitioning

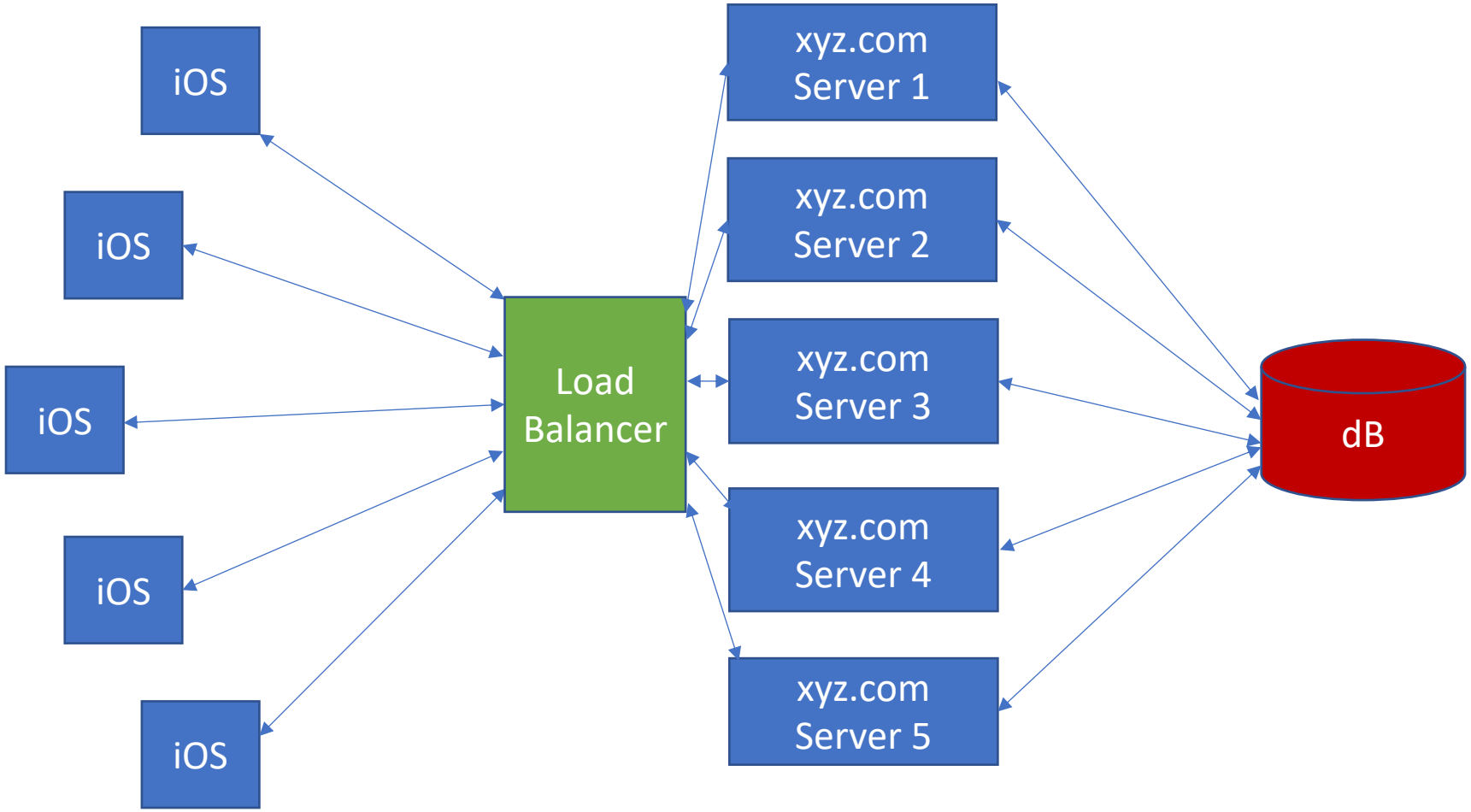
Flights(Dynamic Data)

PK=id	airline	origin_id	destination_id	duration(min)
1	JetBlue Airways	1	10	415
2	Southwest Airlines	2	12	760
3	American Airlines	3	9	700
4	Southwest Airlines	1	12	435
5	Delta Airlines	4	12	245
6	JetBlue Airways	5	1	455
7	Delta Airlines	5	11	320
8	JetBlue Airways	6	11	435
9	American Airlines	7	11	651
10	Delta Airlines	6	8	632

2.1 Vertical Partitioning

1. We have **split the columns based on usage patterns** and now queries are faster.
2. We have **reduced the amount of concurrent access to a table** that's needed because we have two tables now.

2.1 Concerns with Vertical Partitioning



2.1 Concerns with Vertical Partitioning

What happens if the **number of rows** in the table starts to **grow** ?

2.1 Concerns with Vertical Partitioning

Flights(Dynamic Data = 1 Million Rows)

PK=id	airline	origin_id	destination_id	duration(min)
1	JetBlue Airways	1	10	415
2	Southwest Airlines	2	12	760
3	American Airlines	3	9	700
4	Southwest Airlines	1	12	435
5	Delta Airlines	4	12	245
6	JetBlue Airways	5	1	455
7	Delta Airlines	5	11	320
..	JetBlue Airways	..	…	…
999999	American Airlines	7	11	651
1000000	Delta Airlines	6	8	632

2.2 Horizontal Partitioning

2.2 Horizontal Partitioning

1. The idea is to separate a dB table into multiple different tables by **decreasing the number of rows** in those tables.
2. We partition rows based on a column called **Partition Key.**
3. Also we sort each partition based on another column called **Sort Key.**

2.2 Horizontal Partitioning got some rules

1. Range Partitioning
2. List Partitioning
3. Hash partitioning

2.2.1 Range Partitioning

2.2.1 Range Partitioning

1. We split the dB table rows based on in which **interval** the **Partition Key(duration)** lies. Intervals don't overlap.

2. We also use **id(Sort Key)** to sort items in **each partition.**
3. Now we can **redirect query** to appropriate partition based on **Partition Key value**.

2.2.1 Range Partitioning

Flights(Dynamic Data)

PK=id	airline	origin_id	destination_id	duration(min)
1	JetBlue Airways	1	10	415
2	Southwest Airlines	2	12	760
3	American Airlines	3	9	700
4	Southwest Airlines	1	12	435
5	Delta Airlines	4	12	245
6	JetBlue Airways	5	1	455
7	Delta Airlines	5	11	320
8	JetBlue Airways	6	11	435
9	American Airlines	7	11	651
10	Delta Airlines	6	8	632

2.2.1 Range Partitioning

Partition 1

PK=id	airline	origin_id	destination_id	duration(min)
5	Delta Airlines	4	12	245

Partition 2

PK=id	airline	origin_id	destination_id	duration(min)
1	JetBlue Airways	1	10	415
4	Southwest Airlines	1	12	435
6	JetBlue Airways	5	1	455
7	Delta Airlines	5	11	320
8	JetBlue Airways	6	11	435

Partition 3

PK=id	airline	origin_id	destination_id	duration(min)
2	Southwest Airlines	2	12	760
3	American Airlines	3	9	700
9	American Airlines	7	11	651
10	Delta Airlines	6	8	632

2.2.2 List Partitioning

2.2.1 List Partitioning

1. We split the dB table rows by **explicitly listing which key value** appear in each partition and query the table based on **Partition Key.**
2. **Partition key = ‘airline’**
3. Partition1 Value = ‘JetBlue Airways’
4. Partition2 Value = ‘Southwest Airlines’
5. Partition3 Value = ‘American Airlines’
6. Partition4 Value = ‘Delta Airlines’
7. **Sort key = ‘id’,** We sort each partition based on id.

2.2.1 List Partitioning

Flights Table

PK=id	airline	origin_id	destination_id	duration(min)
1	JetBlue Airways	1	10	415
2	Southwest Airlines	2	12	760
3	American Airlines	3	9	700
4	Southwest Airlines	1	12	435
5	Delta Airlines	4	12	245
6	JetBlue Airways	5	1	455
7	Delta Airlines	5	11	320
8	JetBlue Airways	6	11	435
9	American Airlines	7	11	651
10	Delta Airlines	6	8	632

2.2.1 List Partitioning

Partition 1

PK=id	airline	origin_id	destination_id	duration(min)
1	JetBlue Airways	1	10	415
6	JetBlue Airways	5	1	455
8	JetBlue Airways	6	11	435

Partition 2

PK=id	airline	origin_id	destination_id	duration(min)
2	Southwest Airlines	2	12	760
4	Southwest Airlines	1	12	435

Partition 3

PK=id	airline	origin_id	destination_id	duration(min)
5	Delta Airlines	4	12	245
7	Delta Airlines	5	11	320
10	Delta Airlines	6	8	632

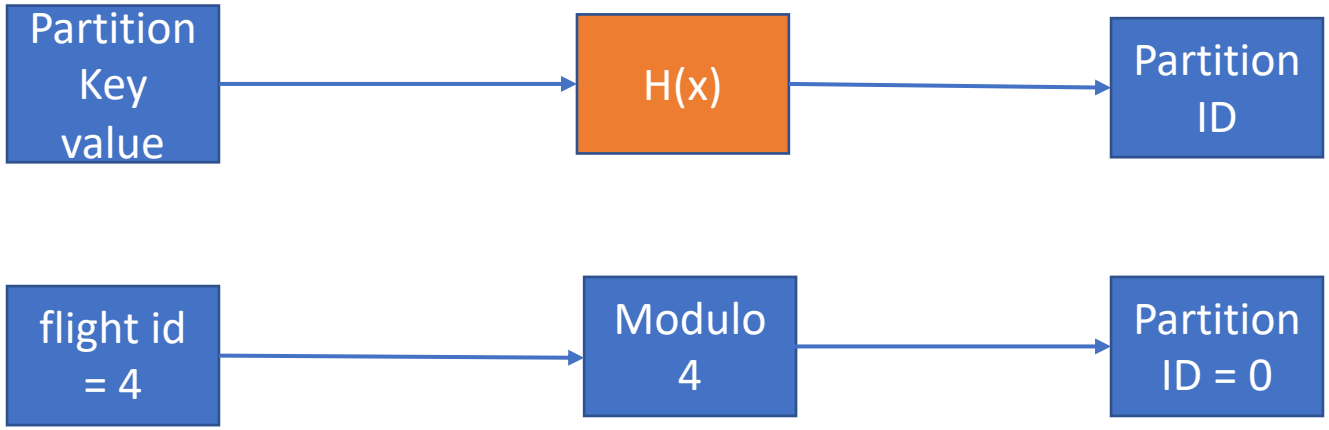
Partition 4

PK=id	airline	origin_id	destination_id	duration(min)
3	American Airlines	3	9	700
9	American Airlines	7	11	651

2.2.3 Hash Partitioning

2.2.3 Hash Partitioning

1. We split the dB table rows based on the hash value of the **Partition Key.**
2. We choose an appropriate Hash function to avoid collision
3. Queries get redirected based on hash value



2.2.3 Hash Partitioning

Flights Table

PK=id	airline	origin_id	destination_id	duration(min)
1	JetBlue Airways	1	10	415
2	Southwest Airlines	2	12	760
3	American Airlines	3	9	700
4	Southwest Airlines	1	12	435
5	Delta Airlines	4	12	245
6	JetBlue Airways	5	1	455
7	Delta Airlines	5	11	320
8	JetBlue Airways	6	11	435
9	American Airlines	7	11	651
10	Delta Airlines	6	8	632

2.2.3 Hash Partitioning

Partition 0

PK=id	airline	origin_id	destination_id	duration(min)
4	Southwest Airlines	1	12	435
8	JetBlue Airways	6	11	435

Partition 1

PK=id	airline	origin_id	destination_id	duration(min)
1	JetBlue Airways	1	10	415
5	Delta Airlines	4	12	245
9	American Airlines	7	11	651

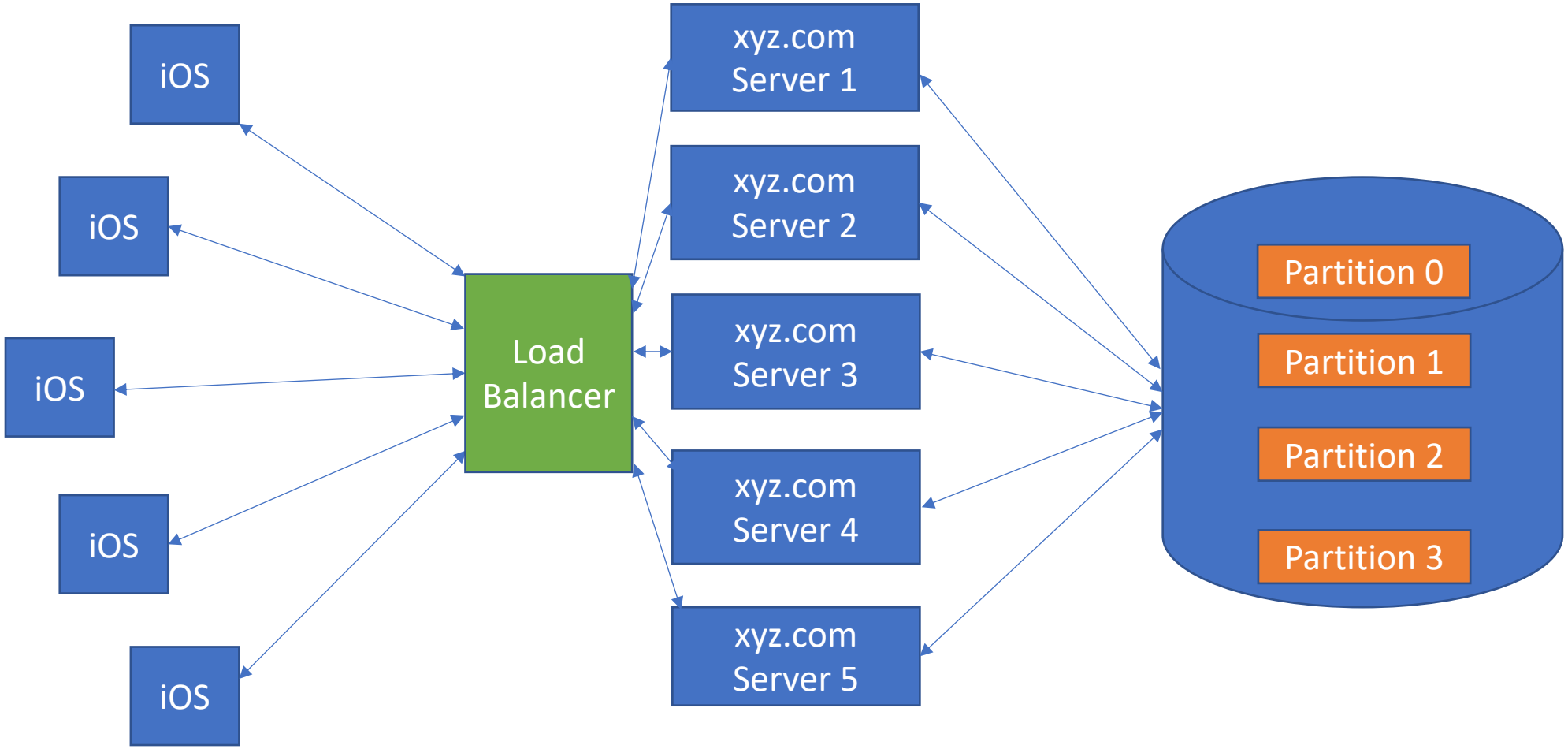
Partition 2

PK=id	airline	origin_id	destination_id	duration(min)
2	Southwest Airlines	2	12	760
6	JetBlue Airways	5	1	455
10	Delta Airlines	6	8	632

Partition 3

PK=id	airline	origin_id	destination_id	duration(min)
3	American Airlines	3	9	700
7	Delta Airlines	5	11	320

2.2.3 Hash Partitioning



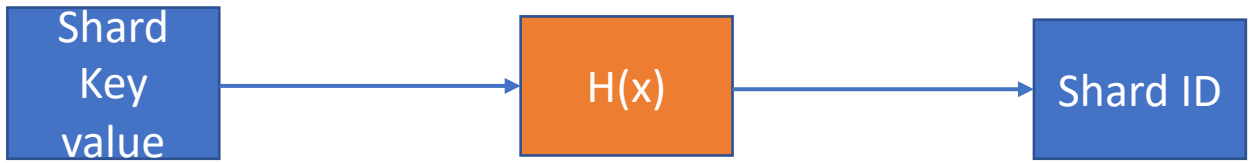
2.2 Concerns with Horizontal Partitioning

1. So far we have been **splitting the table** residing in the **same dB server.**
2. What happens if the table **query results don't fit** in the dB server memory(16 GB)?
3. What happens if the **number of records we can keep in a dB** reaches its limit ?
4. What happens if the CPU(2.9 GHz) is **not able to execute a dB query** within an acceptable time limit ?
5. We have a **single point of failure** with a single dB.
6. **HOTFIX: Vertically Scale the dB server ? Maybe.!!**

3. Sharding

3. Sharding

1. We keep **each horizontal partition(Table)** in a different **dB server** called a **Shard.**
2. We can **choose range/List/hash/composite** sharding scheme.
3. Application Server's query is routed to a Shard based on **Shard Key.**
4. Choose **Shard Key wisely**.!! We don't want too many JOINS(O(M+N)) over network.



3. Sharding: Using hash Partitioning

Shard 0

PK=id	airline	origin_id	destination_id	duration(min)
4	Southwest Airlines	1	12	435
8	JetBlue Airways	6	11	435

Shard 1

PK=id	airline	origin_id	destination_id	duration(min)
1	JetBlue Airways	1	10	415
5	Delta Airlines	4	12	245
9	American Airlines	7	11	651

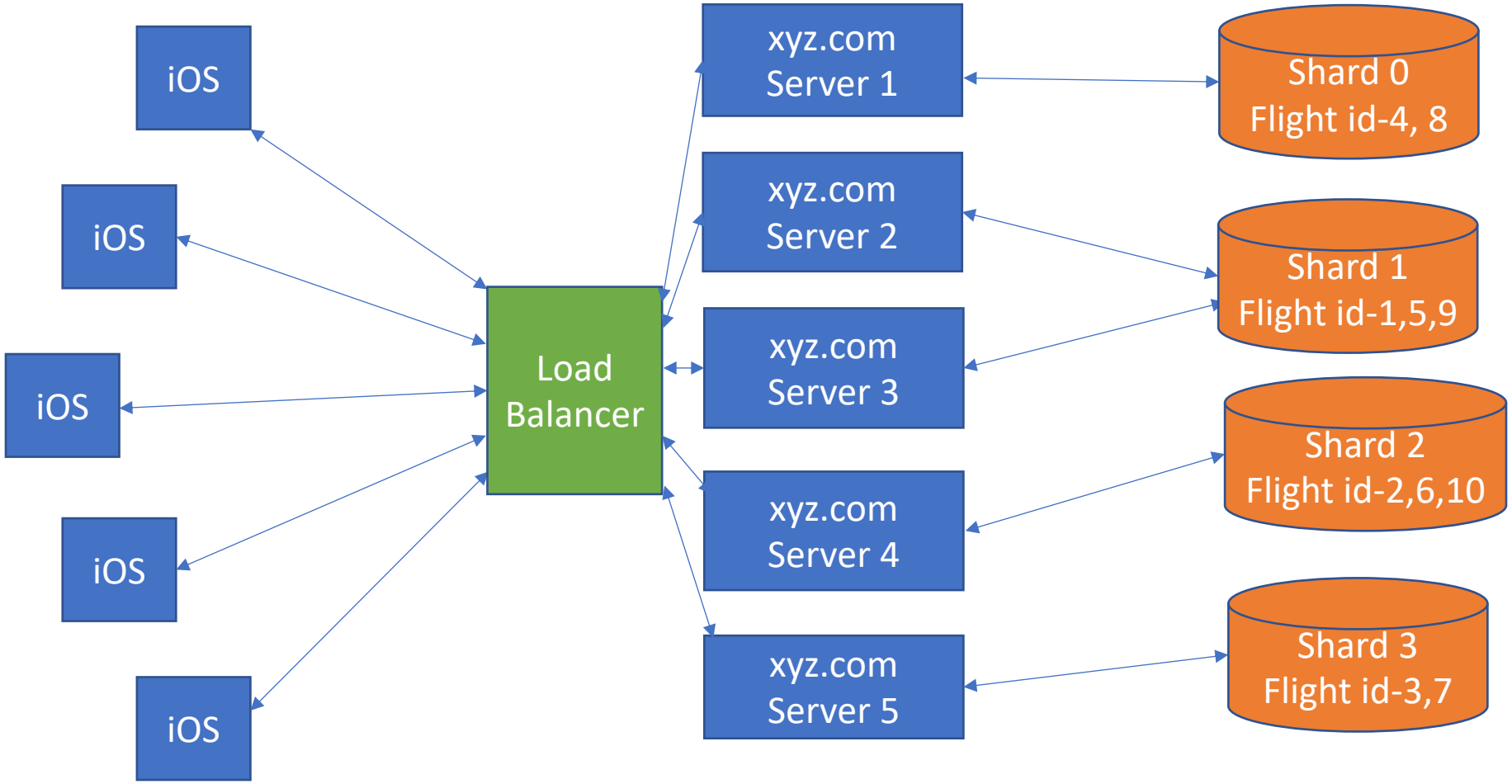
Shard 2

PK=id	airline	origin_id	destination_id	duration(min)
2	Southwest Airlines	2	12	760
6	JetBlue Airways	5	1	455
10	Delta Airlines	6	8	632

Shard 3

PK=id	airline	origin_id	destination_id	duration(min)
3	American Airlines	3	9	700
7	Delta Airlines	5	11	320

3. Sharding: Using hash Partitioning



Benefits of Sharding

1. Now each dB shard can **scale up well** as the **data size grows.**
2. Also now we have **concurrency, whoa:**

Time(Querying Table 1 on Shard 1 + Querying Table 2 on Shard 2) **<** Time(Querying Table 1 and Table 2 on same dB server)

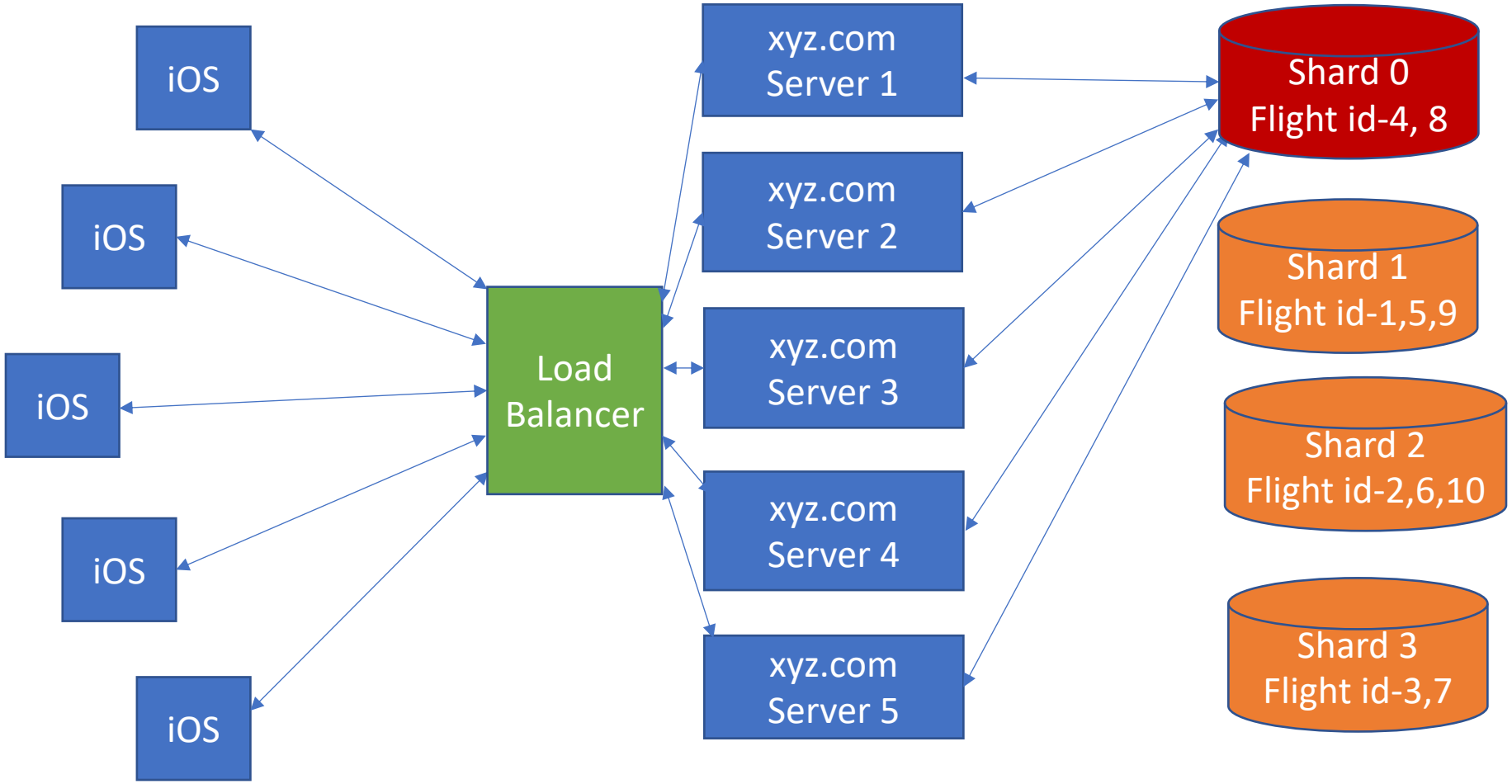
3. **Queries** are **faster** now. ☺

4. **Proudly,** We have solved the problem of **single point of failure**. If **one shard is down** we still have **other shards up** and running. **Really ?**

Concerns with Sharding

1. What if all **4 application servers** query **Shard-0** concurrently and overload **Shard-0** ?
2. Every shard is a **single point of failure**.

Concerns with Sharding

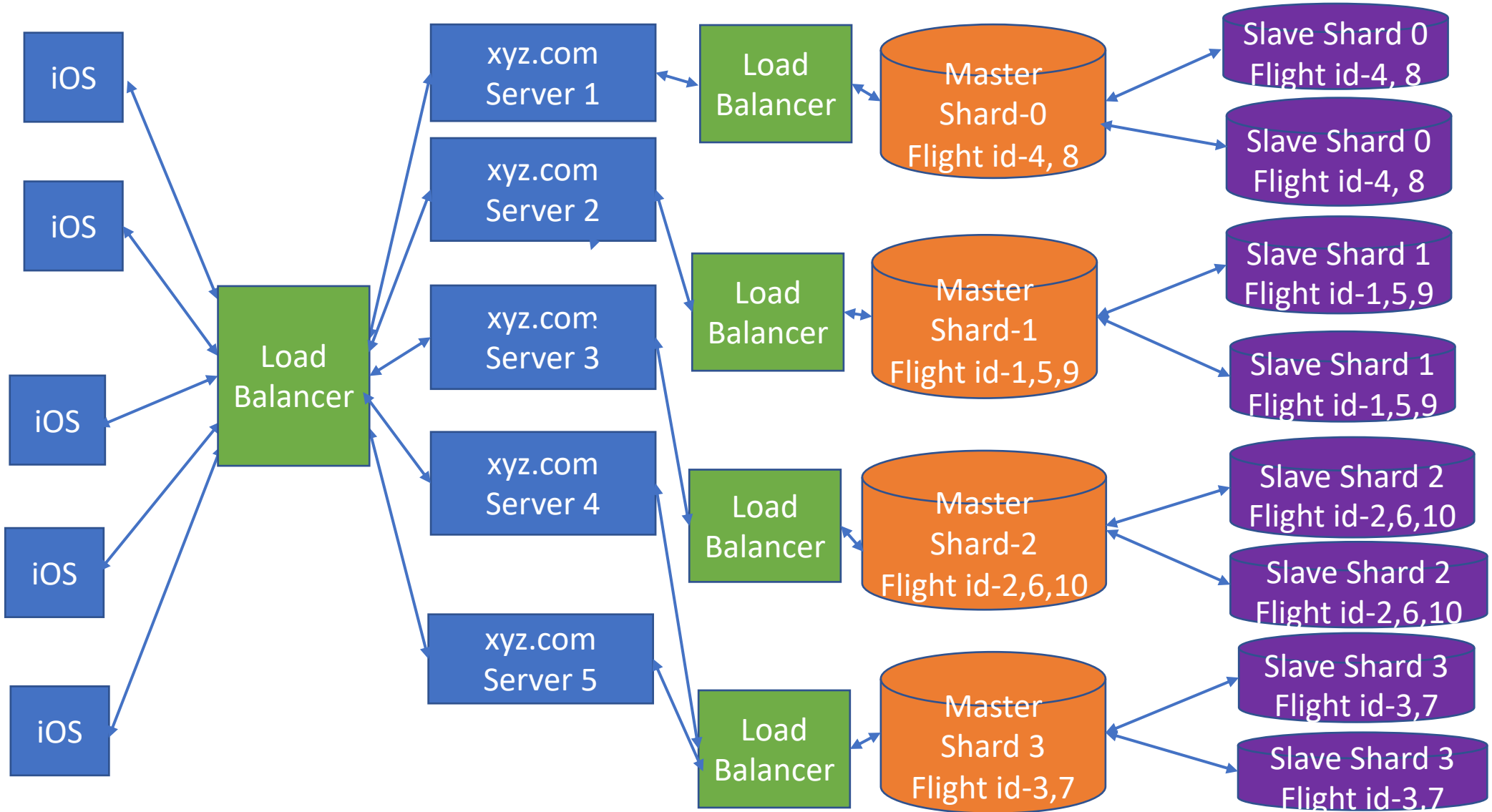


4. Database Replication

4. Database Replication

1. The idea is to **create multiple copies** of each Shard called a **Replica**.
2. Now we **distribute the load across replicas** using a **Load balancer.**
3. This solves the single point of failure issues having shard replicas.
4. Lets create **3 dB replicas** of each of the **Shards-0,1,2,3.**

4. Database Replication



4. Database Replication

1. Now that we have **3 dB replicas for each Shard** what happens if we update one shard Replica ?
2. How do we make sure that **updates are consistent across** shard replicas ?
3. How do we make sure **read/write operations are in sync** across all the 3 shard Replicas ?
4. How do the **dB replicas communicate** among themselves ?
5. We have few **dB replication models(Master-Slave, Master-Master)** to to fix this problem. We will be seeing those in the next session.

Stay Tuned for Next session on:

DB Replication Models

Consistency, Availability and Partition Tolerance

Caching

Questions