Domain Name Server using Distributed Hash Table

Submitted By: Team No 36

Objective

WE NEED TO IMPLEMENT
A DNS USING DHT.

DNS is a standard protocol that helps Internet users discover websites using human readable addresses. DNS lets you type the address of a website and automatically discover the Internet Protocol (IP) address for that website.

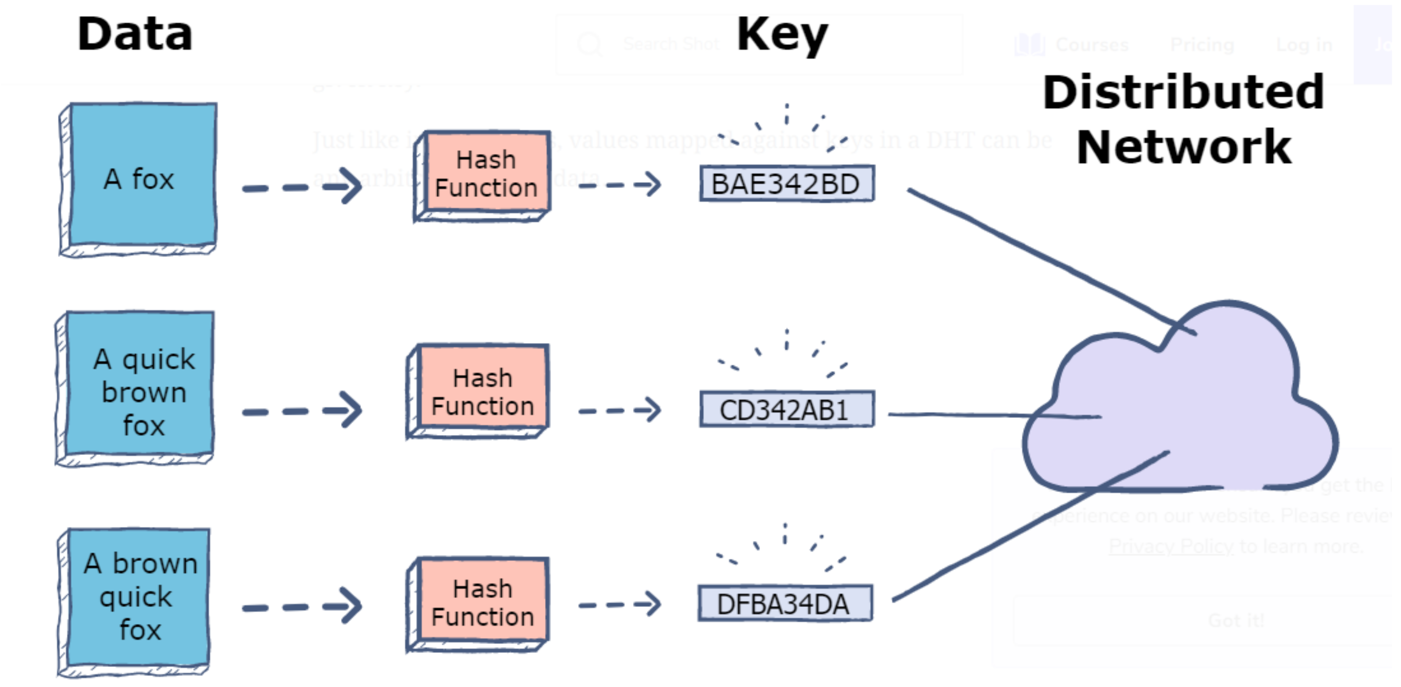
FOR THIS WE ARE USING
THE CHORD PROTOCOL.

Chord is based on consistent hashing, which assigns hash keys to nodes in a way that doesn't need to change much as nodes join and leave the system.

Distributed Hash Table

A DHT is a decentralized storage system that provides lookup and storage schemes similar to a hash table, storing key-value pairs.

Each node in a DHT is responsible for keys along with the mapped values. Any node can efficiently retrieve the value associated with a given key.



Just like in hash tables, values mapped against keys in a DHT can be any arbitrary form of data.

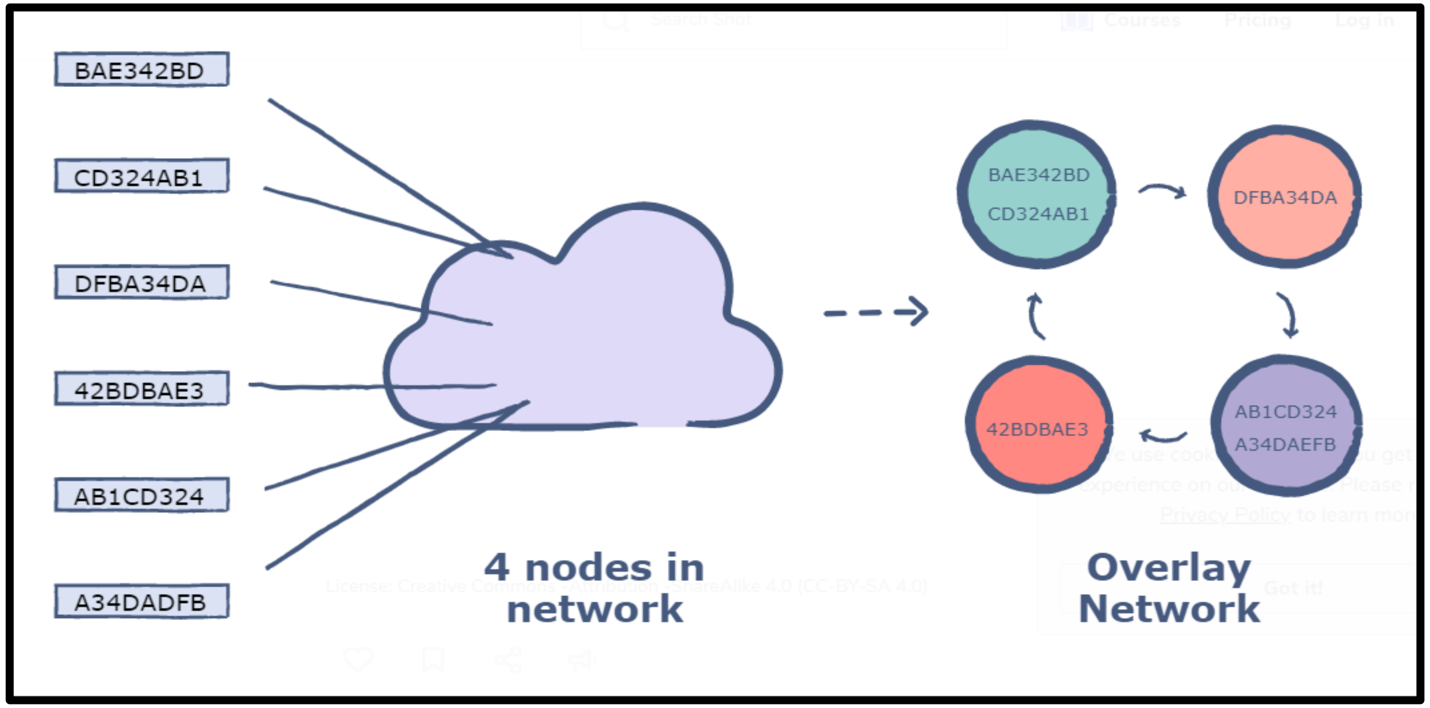
Distributed Hash Table

DHT's Properties

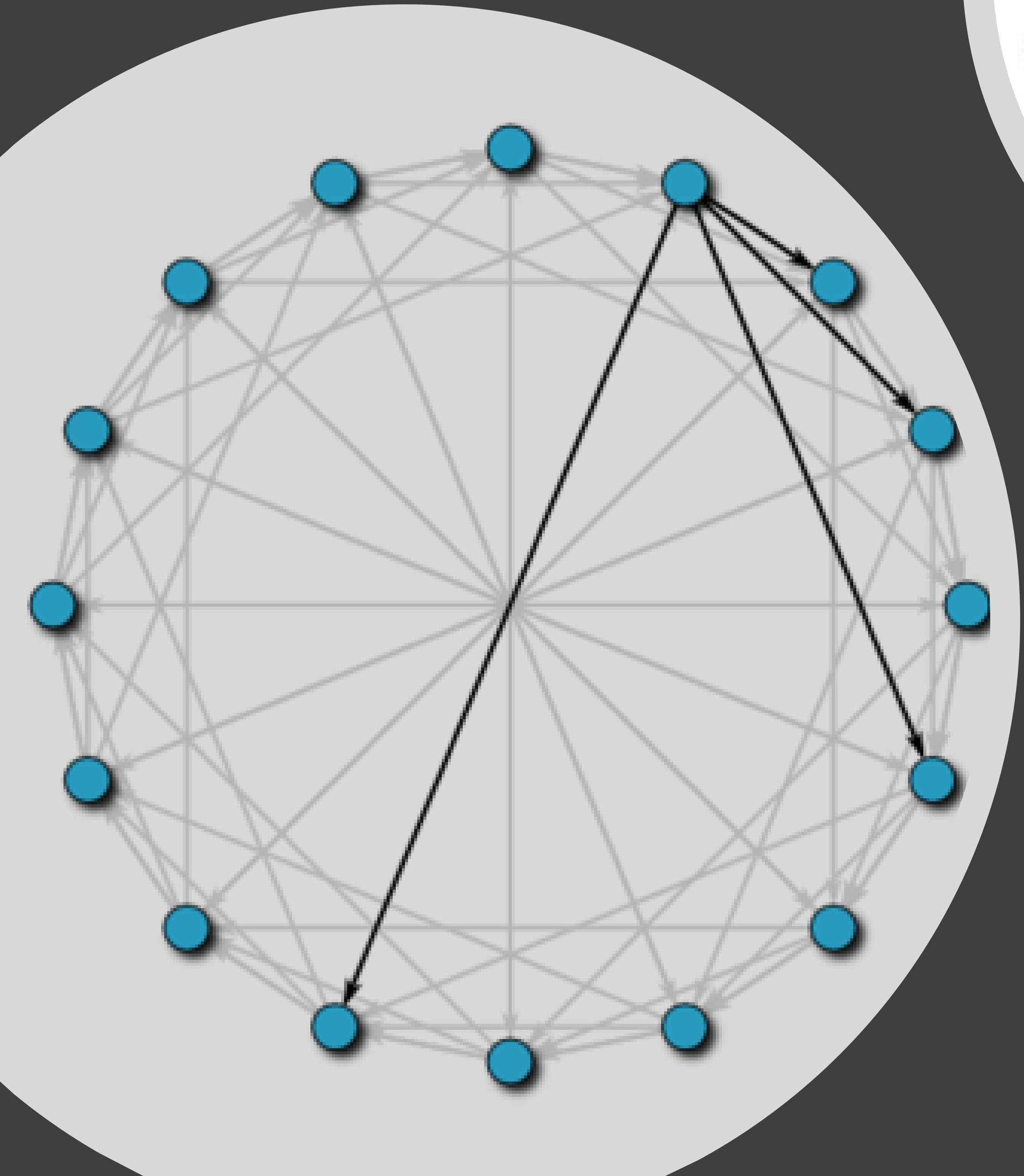
- **Decentralized & Autonomous**
- **Fault Tolerant**
- **Scalable**

DHT's Functions

- *put* (key, value)
- *get* (key)



The nodes in a DHT are connected together through an **overlay network** in which neighboring nodes are connected. This network allows the nodes to find any given key in the key-space.



CHORD PROTOCOL

Chord Protocol

Chord is based on consistent hashing, which assigns hash keys to nodes in a way that doesn't need to change much as nodes join and leave the system

Three features that distinguish Chord from many other peer-to-peer lookup protocols are its simplicity, provable correctness, and provable performance.

The Chord protocol supports just one operation: given a key, it will determine the node responsible for storing the key's value

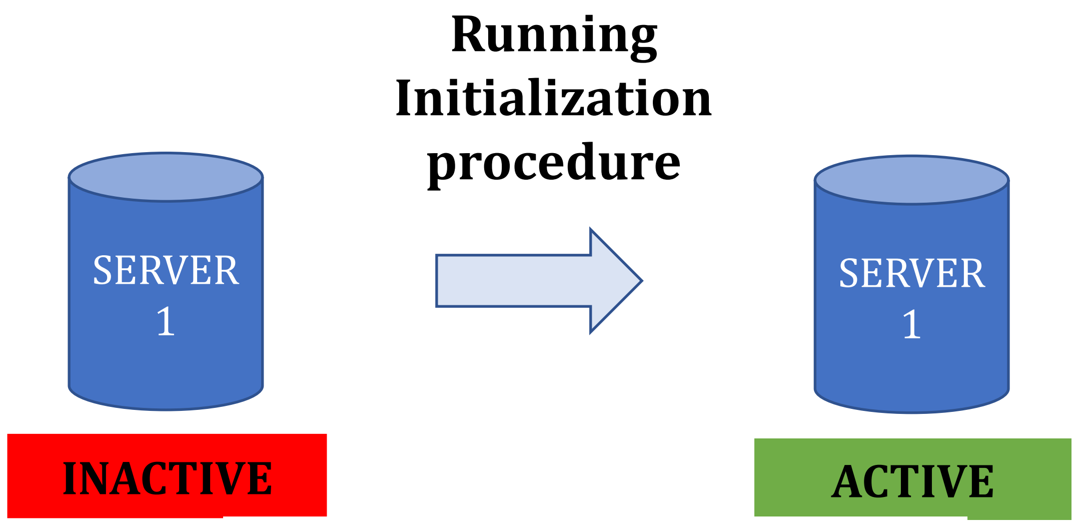
Efficient: O(log N) messages per lookup• N is the total number of servers• Scalable: O(log N) state per node• Robust: survives massive failures

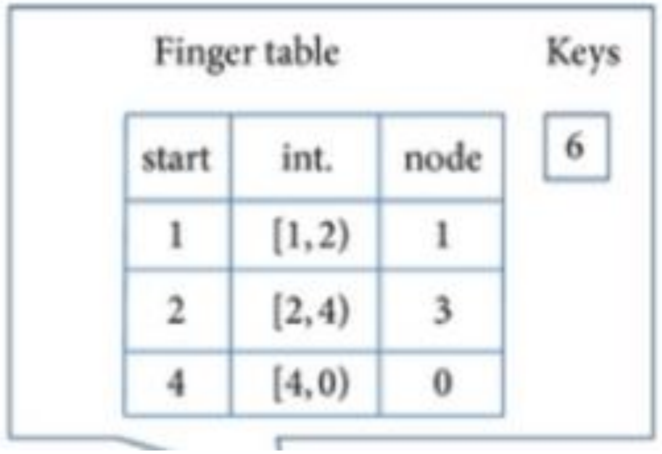
Our Implementation

Approach from Paper:

Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications.

PHASE 1: INITITATION MODULE





Finger table

start	int.	node
1	[1,2)	1
2	[2,4)	3
4	[4,0)	0

Keys

6

Operations Performed

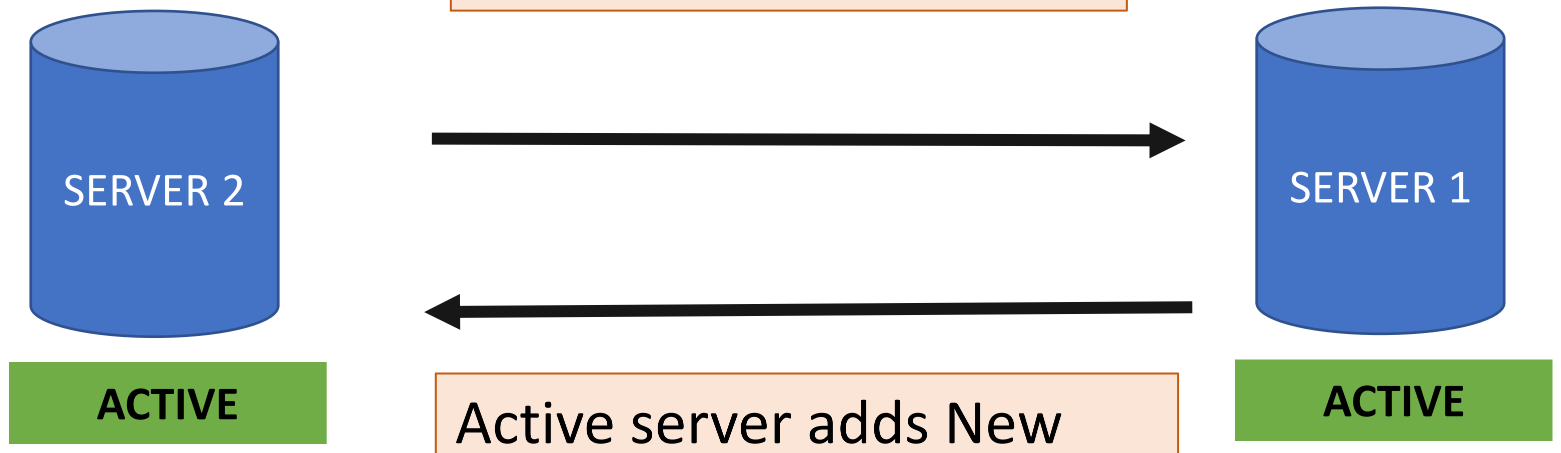
Setting Predecessor as NULL

Setting Successor as itself

Initializing Data Structures

- Updating Finger table
- Updating Key value Pair

PHASE 2: Node JOIN



New Server Sends Join request

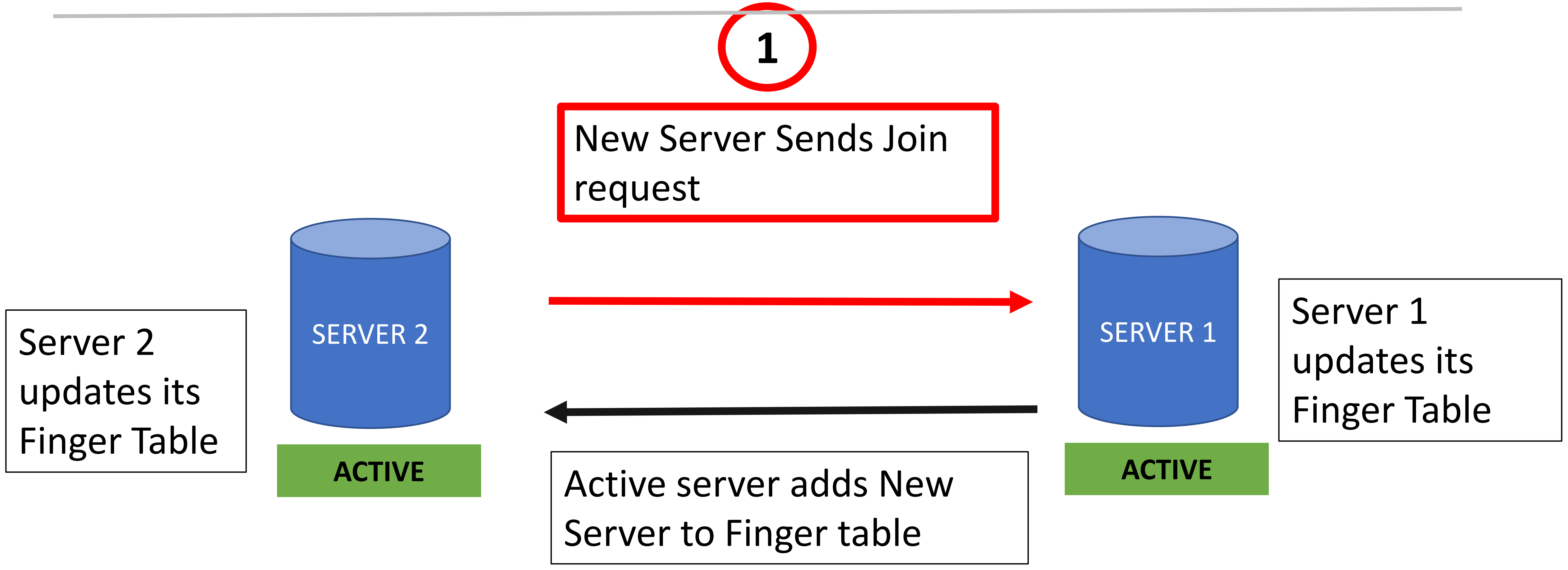
4.Server 2 updates its Finger Table

Server 1 updates its Finger Table

Active server adds New Server to Finger table

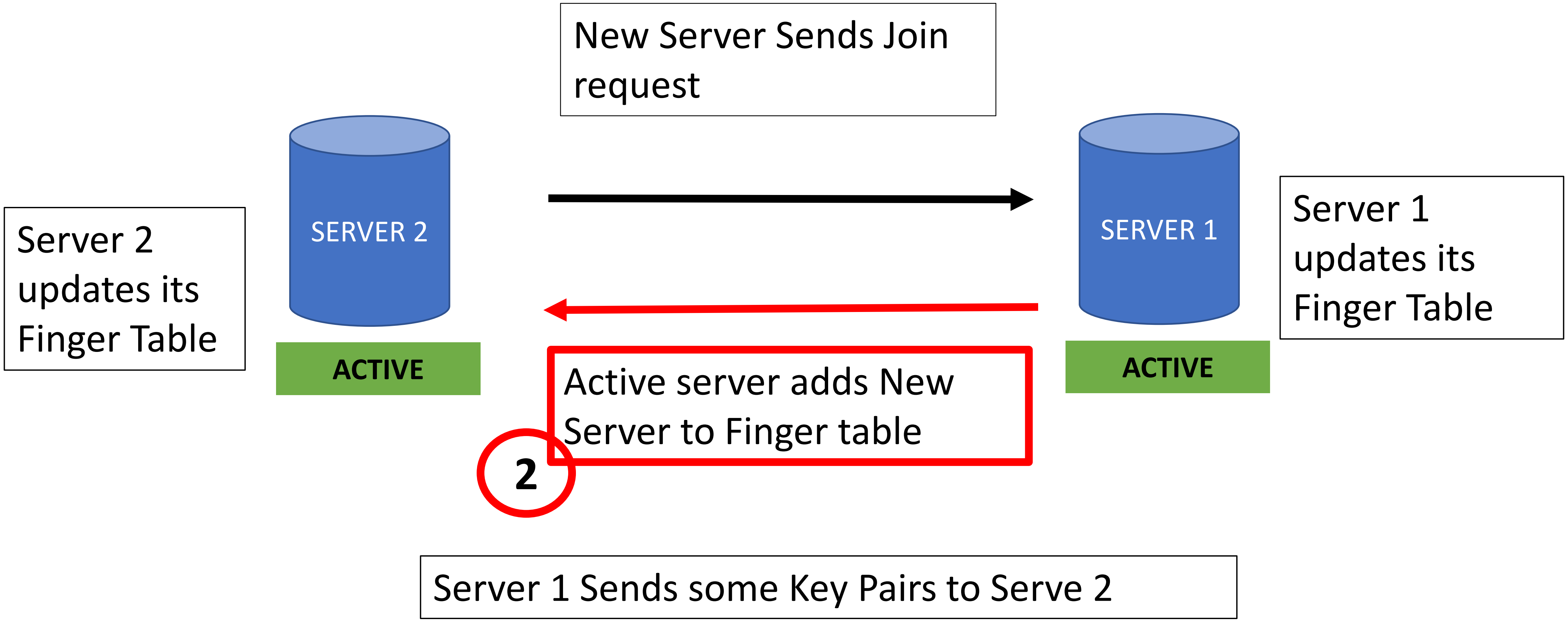
Server 1 Sends some Key Pairs to Serve 2

PHASE 2: Node JOIN


1
New Server Sends Join request
SERVER 2
SERVER 1
Server 2 updates its Finger Table
Server 1 updates its Finger Table
ACTIVE
ACTIVE
Active server adds New Server to Finger table

Server 1 Sends some Key Pairs to Serve 2

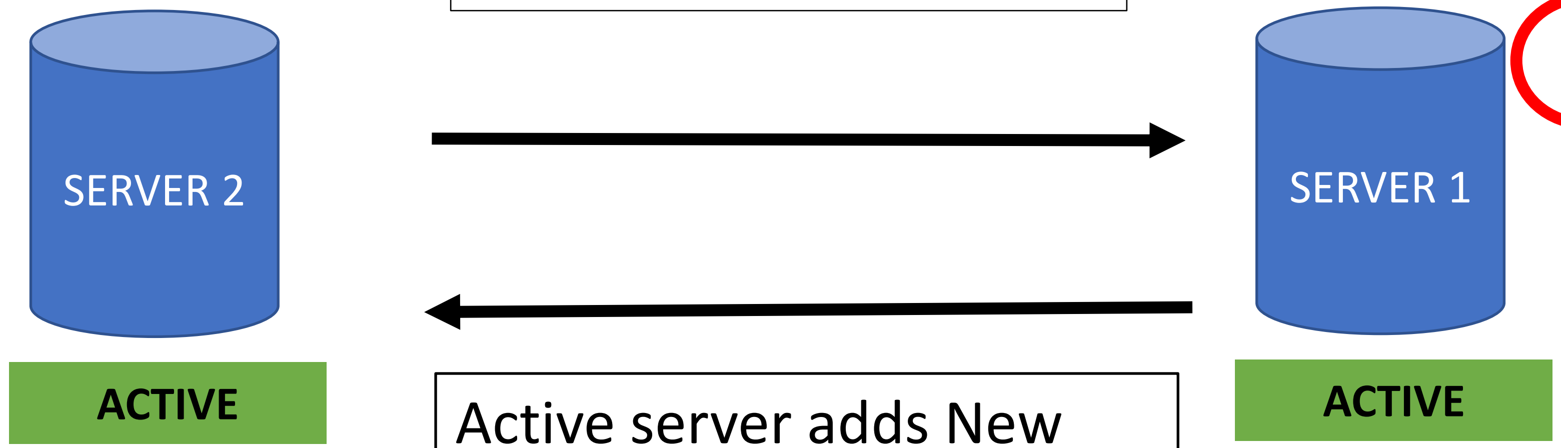
PHASE 2: Node JOIN


New Server Sends Join request
Server 2 updates its Finger Table
SERVER 2
ACTIVE
SERVER 1
ACTIVE
Server 1 updates its Finger Table
Active server adds New Server to Finger table
2
Server 1 Sends some Key Pairs to Serve 2

PHASE 2: Node JOIN

New Server Sends Join request

Server 2 updates its Finger Table



3

Server 1 updates its Finger Table

Active server adds New Server to Finger table

start	range	node
S2	A-g	S2

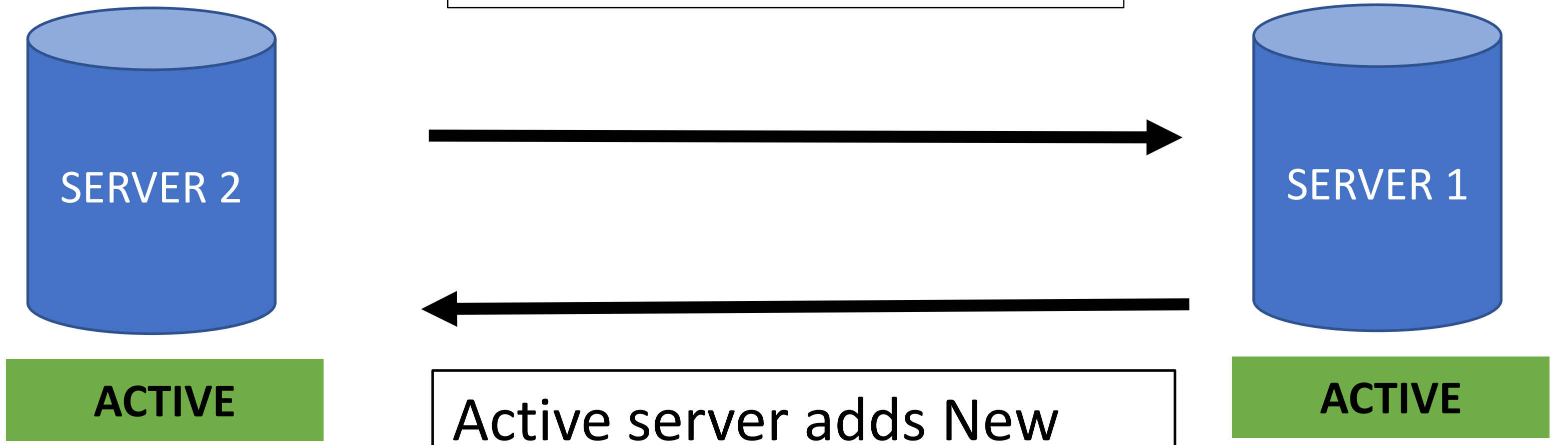
Server 1 Sends some Key Pairs to Serve 2

PHASE 2: Node JOIN

New Server Sends Join request

4

Server 2 updates its Finger Table



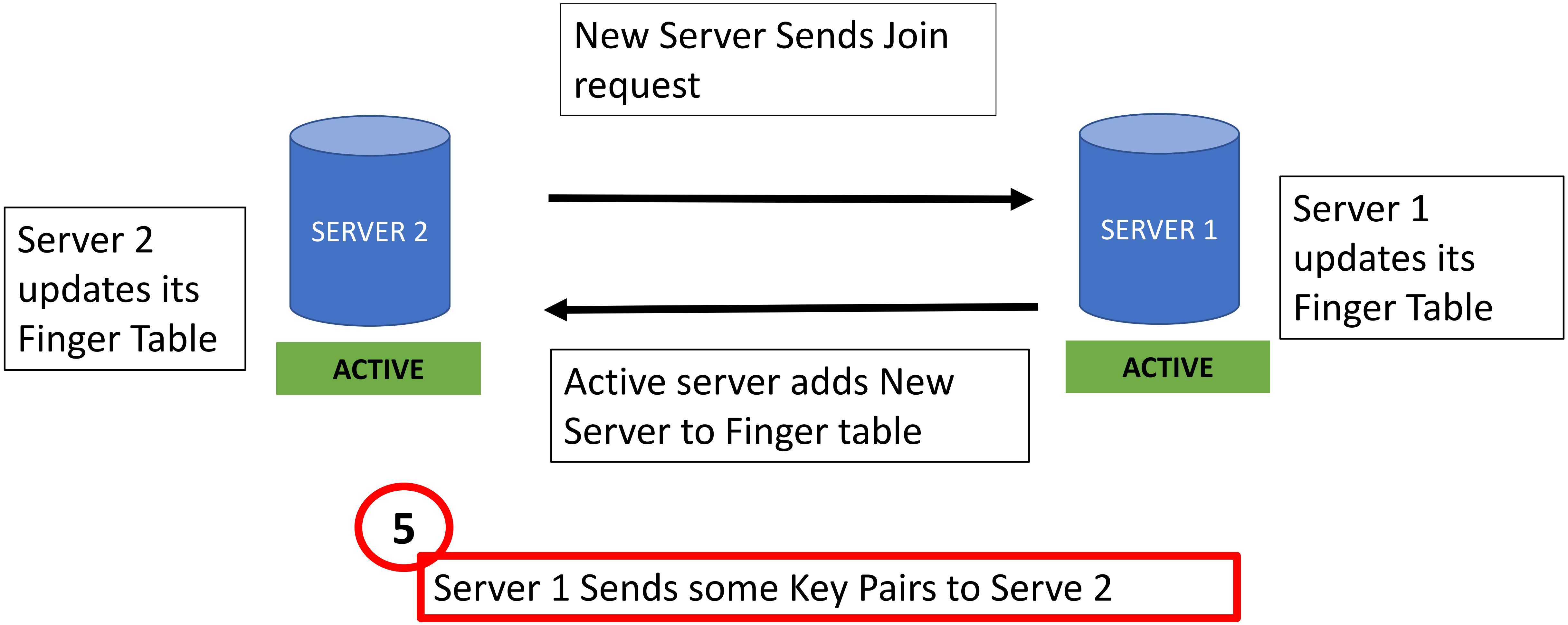
Server 1 updates its Finger Table

Active server adds New Server to Finger table

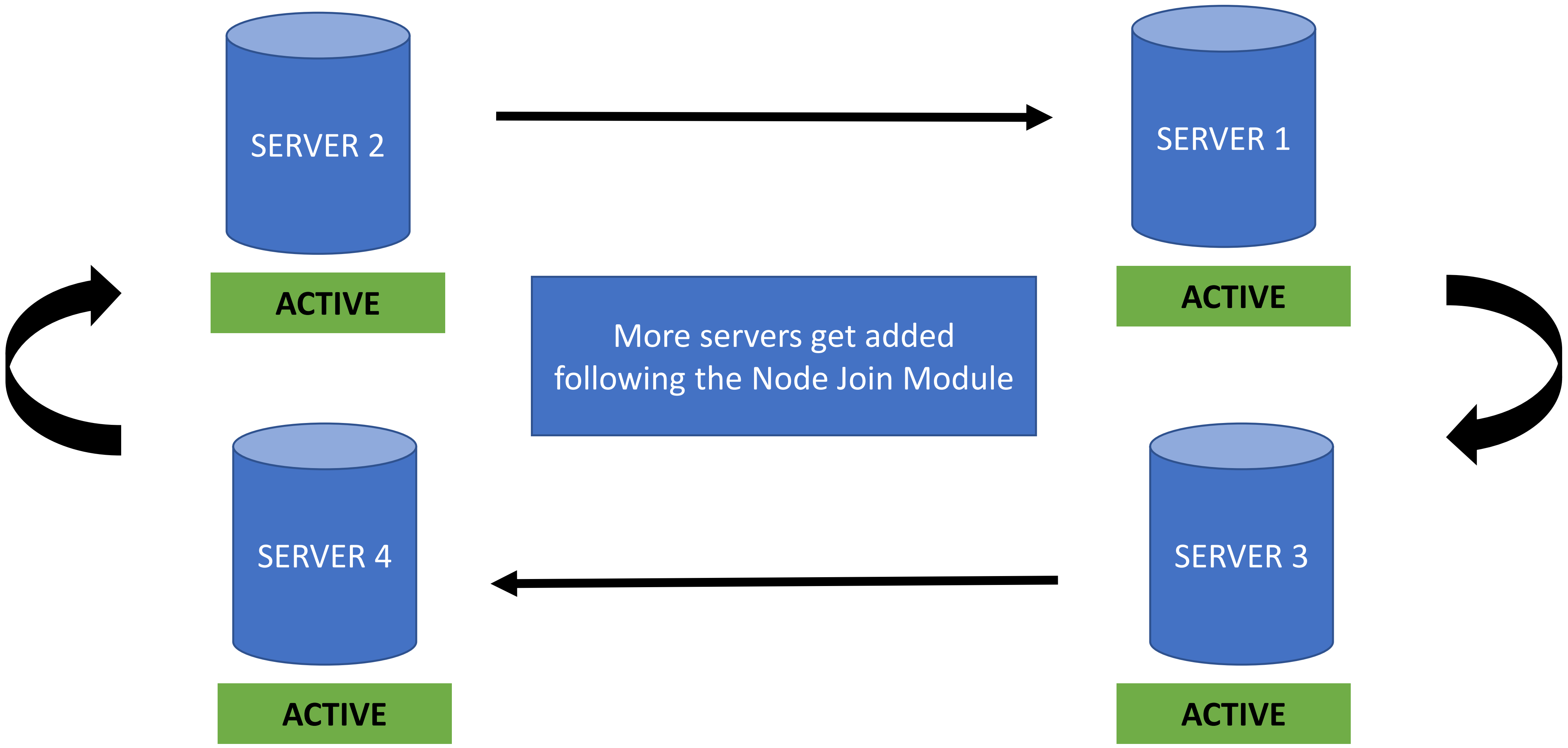
start	range	node
S1	g-z	S1

Server 1 Sends some Key Pairs to Serve 2

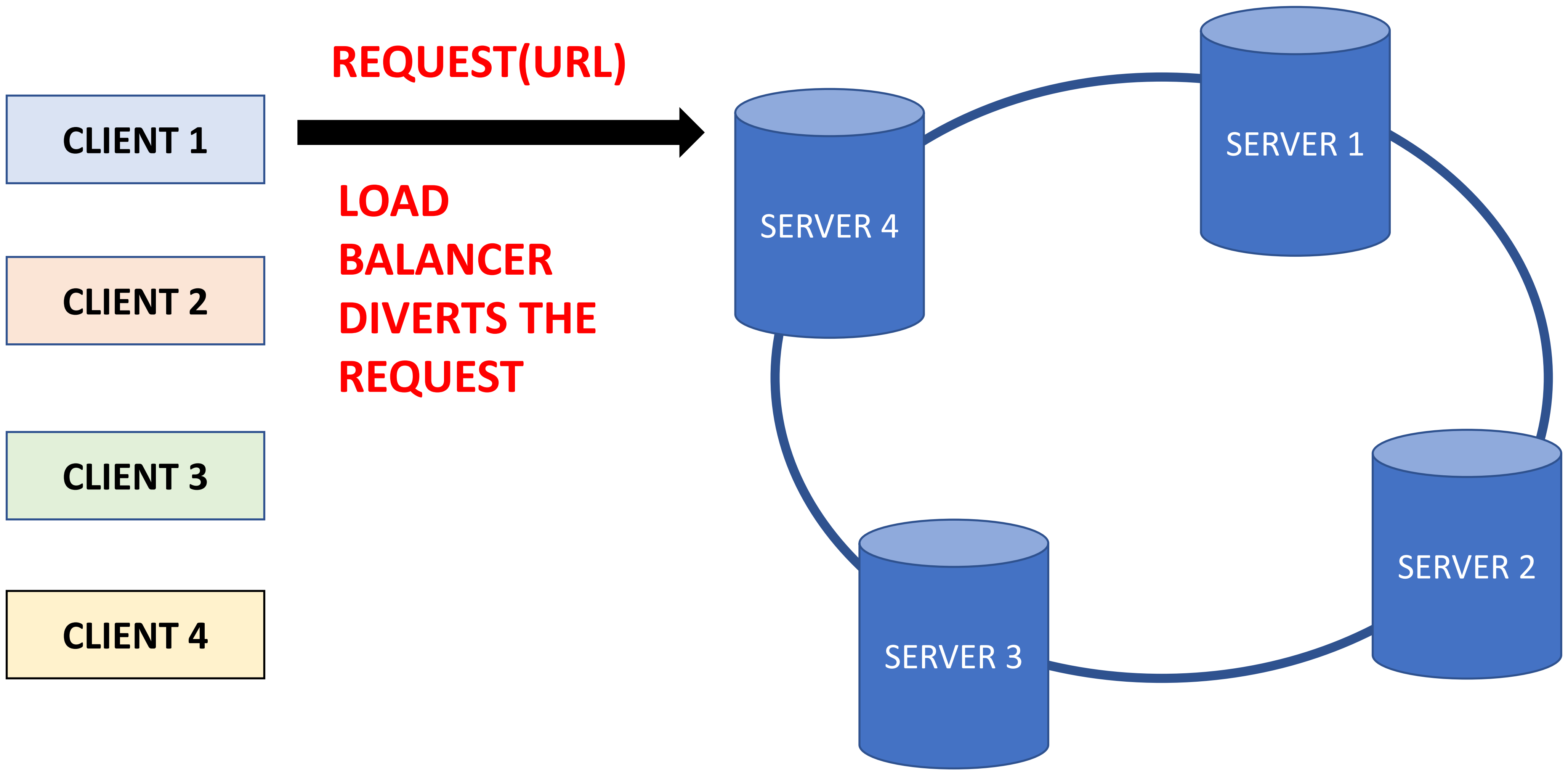
PHASE 2: Node JOIN


New Server Sends Join request
Server 2 updates its Finger Table
SERVER 2
ACTIVE
SERVER 1
ACTIVE
Server 1 updates its Finger Table
Active server adds New Server to Finger table
5
Server 1 Sends some Key Pairs to Serve 2

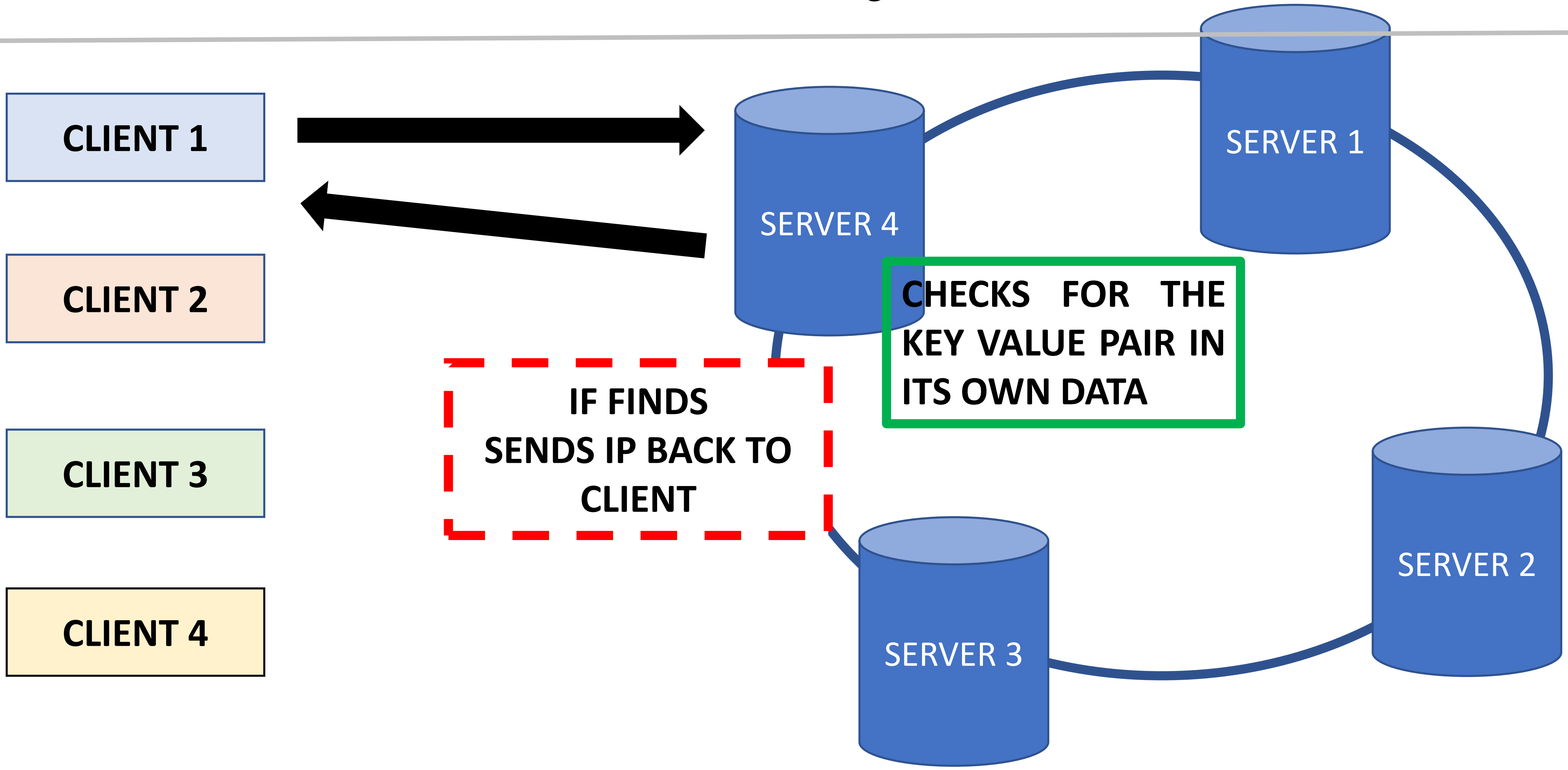
PHASE 2: Node JOIN



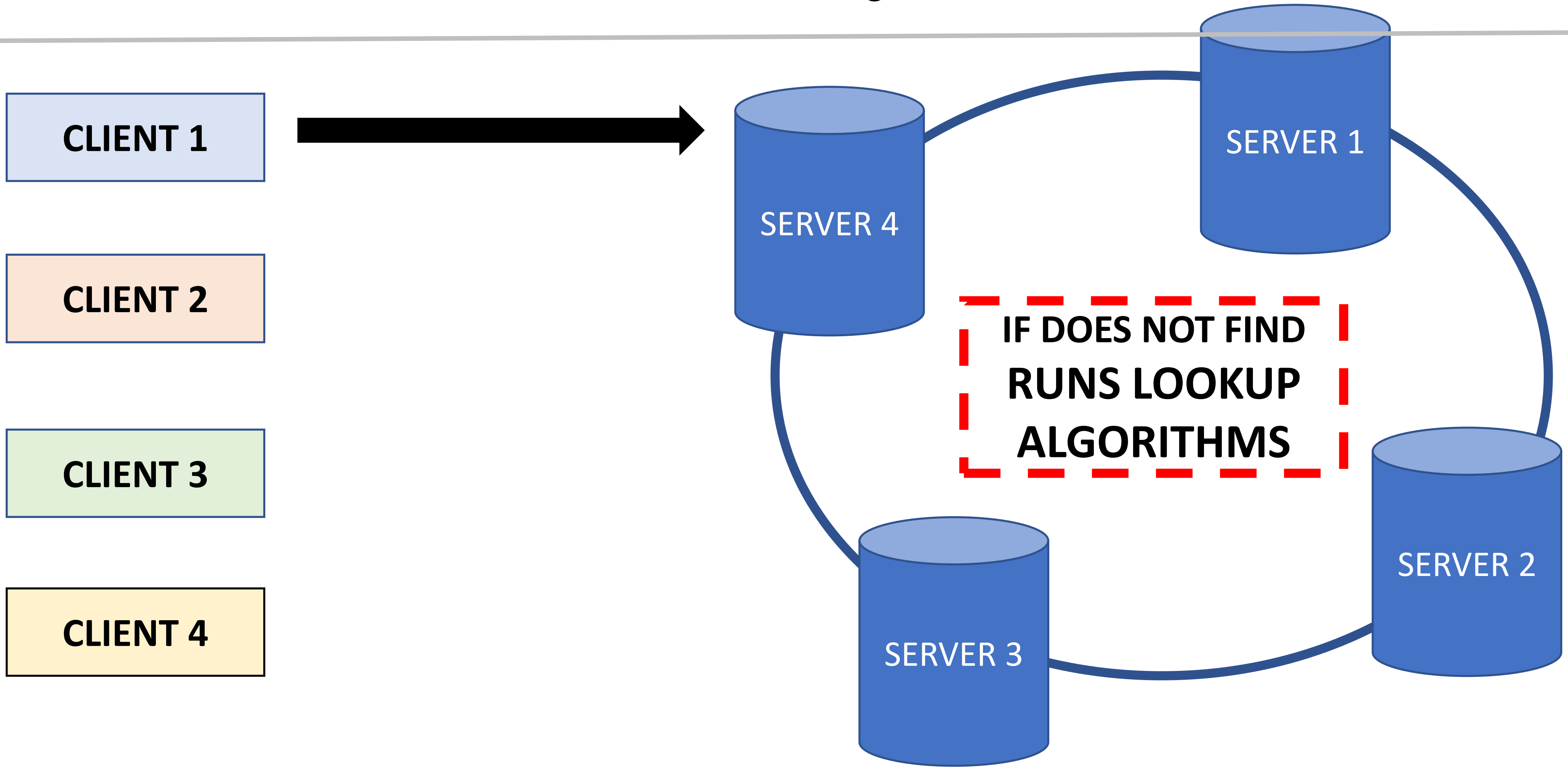
PHASE 3: CLIENT URL QUERY



PHASE 3: CLIENT URL QUERY


CLIENT 1
CLIENT 2
CLIENT 3
CLIENT 4
SERVER 4
SERVER 1
SERVER 2
SERVER 3
CHECKS FOR THE KEY VALUE PAIR IN ITS OWN DATA
IF FINDS SENDS IP BACK TO CLIENT

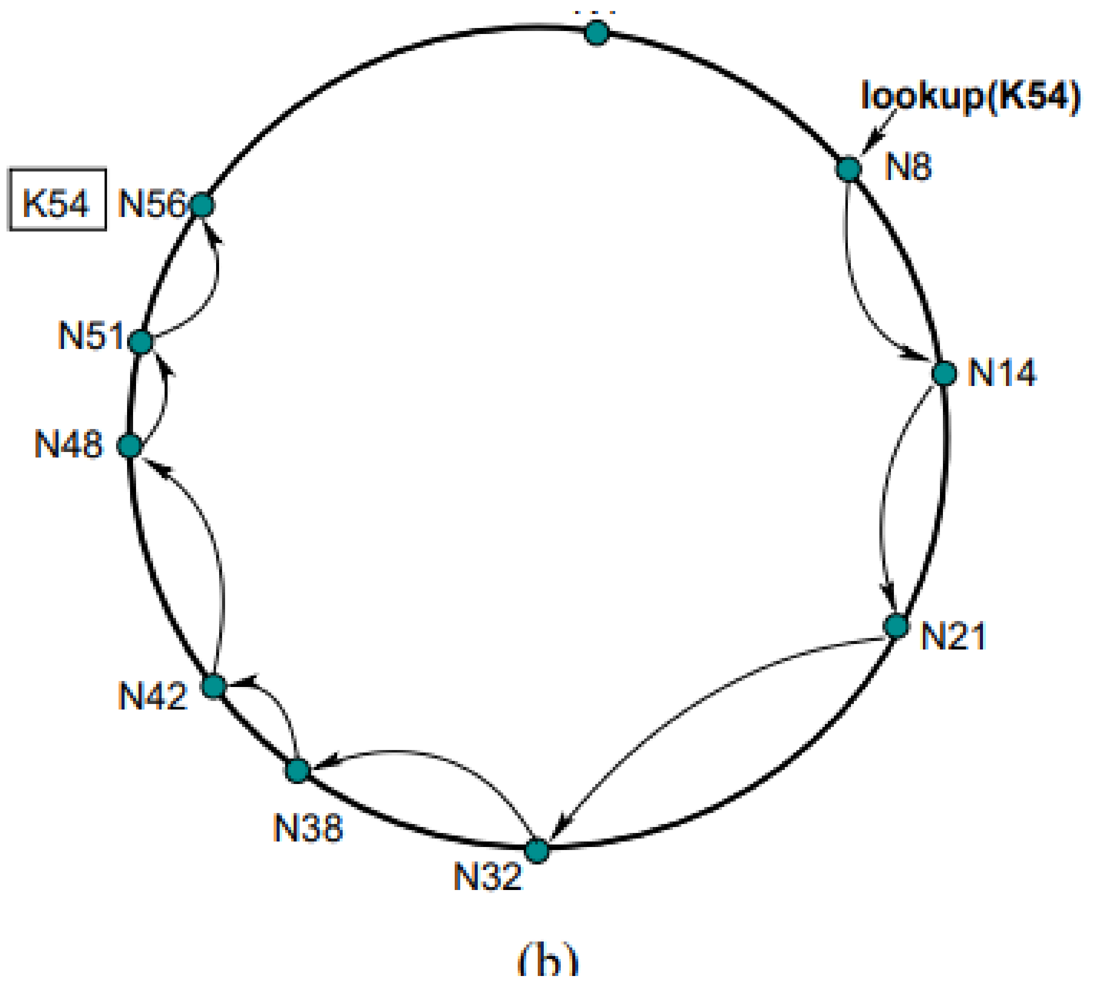
PHASE 3: CLIENT URL QUERY



PHASE 3: CLIENT URL QUERY

LOOKUP ALGORITHM:

1. Request with url to the load balancer
2. load balancer will forward the request to one of the servers in chord
3. node in the chord will calculate hashed value and search for the successor of the url
4. If hashed valued matches with any successor then request will be forwarded to that node otherwise it will be forwarded to the last node in its finger table.
5. Matched node will search the url in its own url_ip map and return it and if url is not present in that case it will return "Address not found"

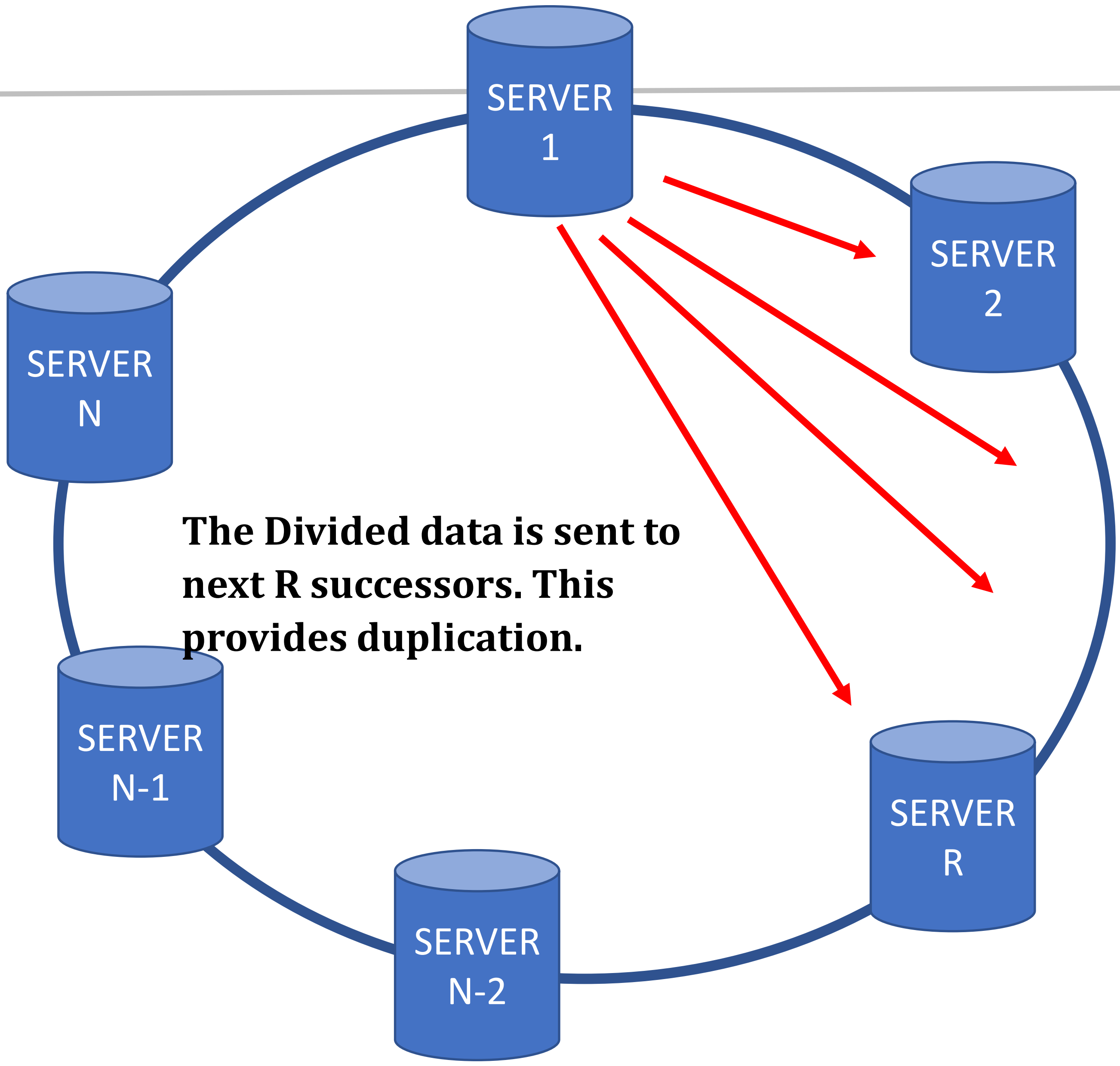


Fault Tolerance

⇒ When chord is implemented, every server is distributing its data to next R successors.

⇒ Now, if these R +1 nodes fail or crash simultaneously, Only then will our model fail.

⇒ Due to large Servers and low key ratio in every server, the data duplication is also less.


SERVER 1
SERVER 2
SERVER N
SERVER N-1
SERVER N-2
SERVER R
The Divided data is sent to next R successors. This provides duplication.

PROJECT DEMO