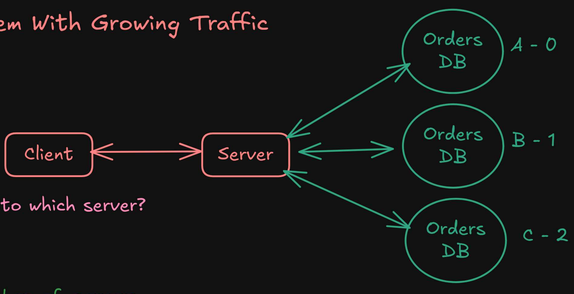
Example Setup: Orders System With Growing Traffic



How do you decide which order goes to which server?

You need a routing logic.
That's where hashing comes in.

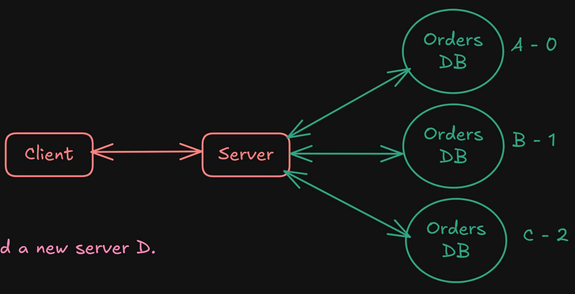
`server_index = hash(order_id) % number_of_servers`

hash(101) = 287 → 287 % 3 = 2 → Server C
(because 287 ÷ 3 = 95 remainder 2)

hash(205) = 193 → 193 % 3 = 1 → Server B
(193 ÷ 3 = 64 remainder 1)

hash(309) = 300 → 300 % 3 = 0 → Server A
(300 ÷ 3 = 100 remainder 0)

Problem: Rehashing When Servers Change



Now let's say traffic increases again and you add a new server D.
Number of servers changes from 3 → 4.
Now the formula changes:

`server_index = hash(order_id) % 4`

Before (3 servers): 5 % 3 = 2 → Server C

After (4 servers): 5 % 4 = 1 → Server B (moved)

Almost every order will move to a different server, causing:

Huge data migration
Cache miss storm
Downtime
Network overload
Operational headaches

Even removing one server breaks everything again.
This is the fundamental problem of normal hashing.

Introducing Consistent Hashing

Consistent Hashing was invented to solve this exact problem.
Main idea:

When servers are added or removed,

only a very small portion of keys should move instead of remapping everything.

Consistent Hashing

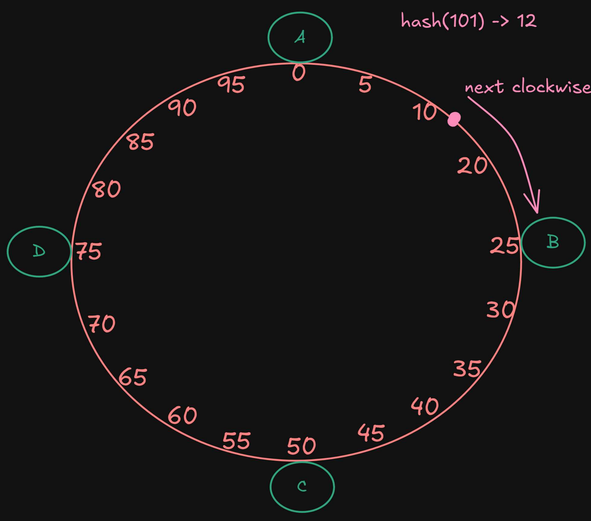
Consistent hashing is a hashing strategy designed so that minimum data moves when nodes/servers are added or removed.

Classic hashing scatters keys evenly but reshuffles everything when cluster size changes.
Consistent hashing fixes that headache.

Why We Even Need It ?

Normal hashing (like server = hash(key) % N) works fine until N changes.
If:
you add a node → almost all keys remap
you remove a node → almost all keys remap

Consistent hashing ensures only approximately (1/n) of the total keys move



Core Idea

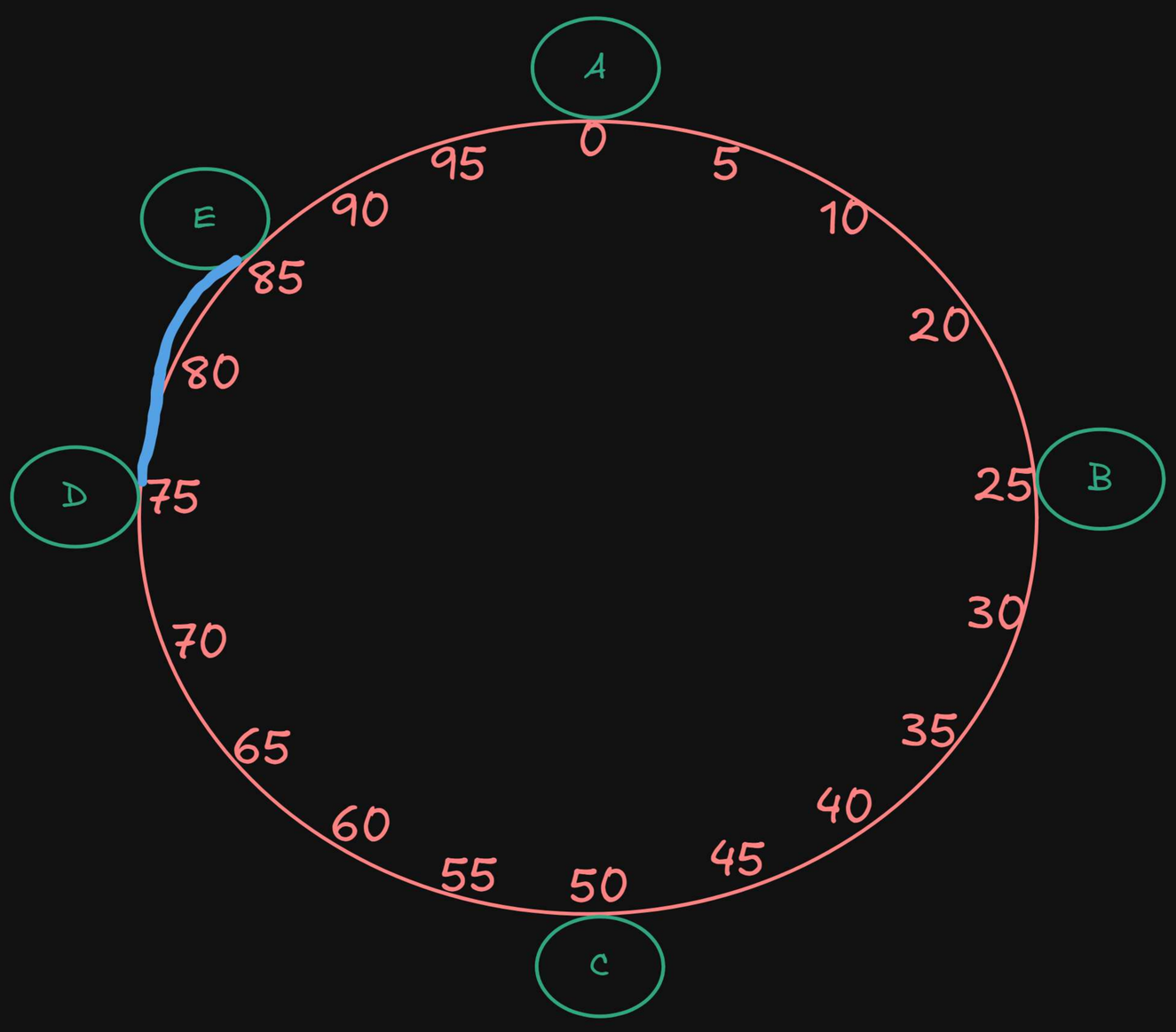
Imagine:

you hash keys
you hash servers
place both on a circular ring (0 to $2^{32} - 1$)

Each key goes to the next server clockwise.
If the hash goes beyond max, wrap to 0.

So the ring solves mapping without depending on N.

What Happens When Nodes Change?

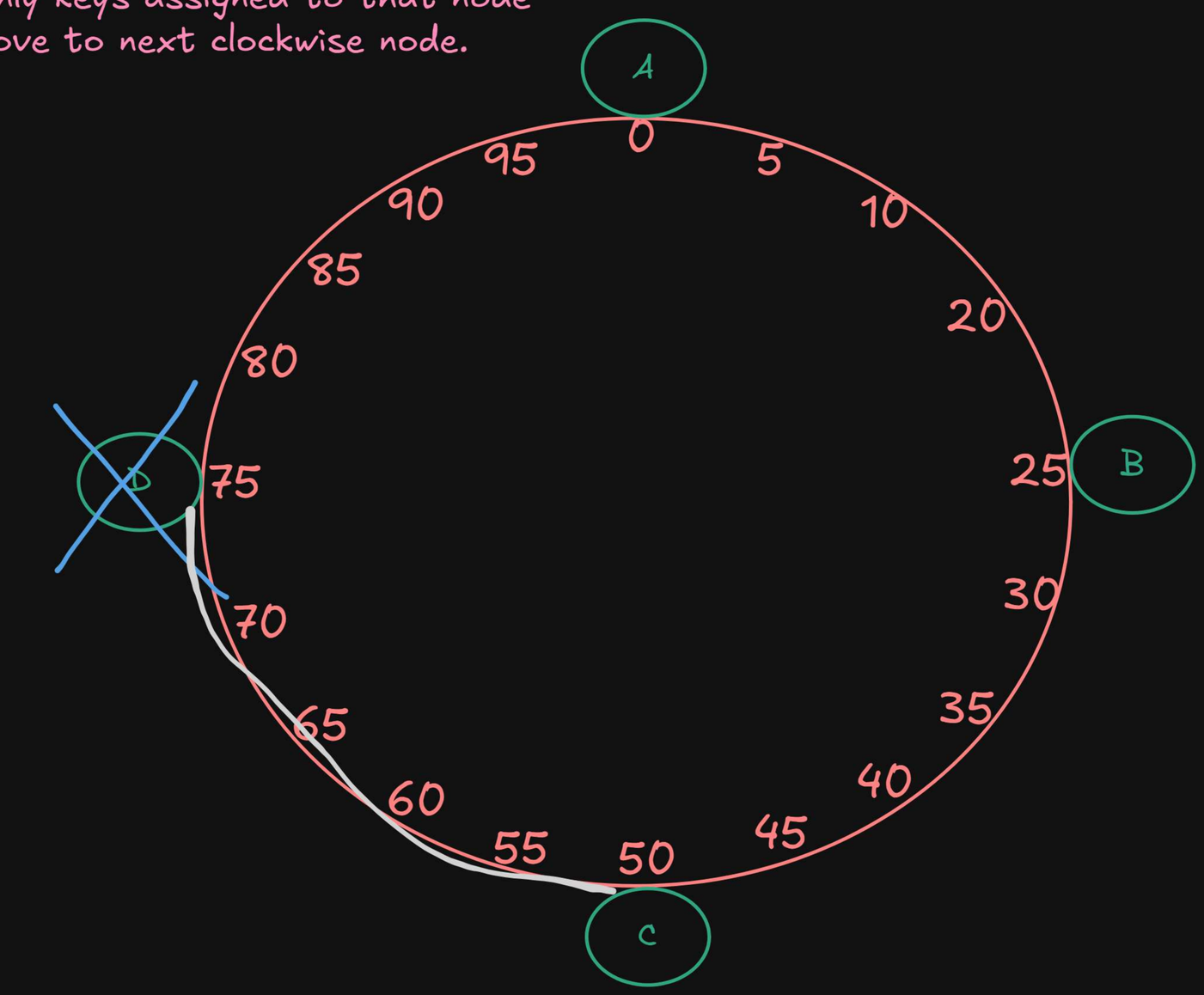


Add a node

Only keys that would now map to this new node
(i.e., keys between prev server and new node) move.

Remove a node

Only keys assigned to that node
move to next clockwise node.



The Problem: Uneven Distribution

Hashing nodes directly to the ring can
lead to imbalance:

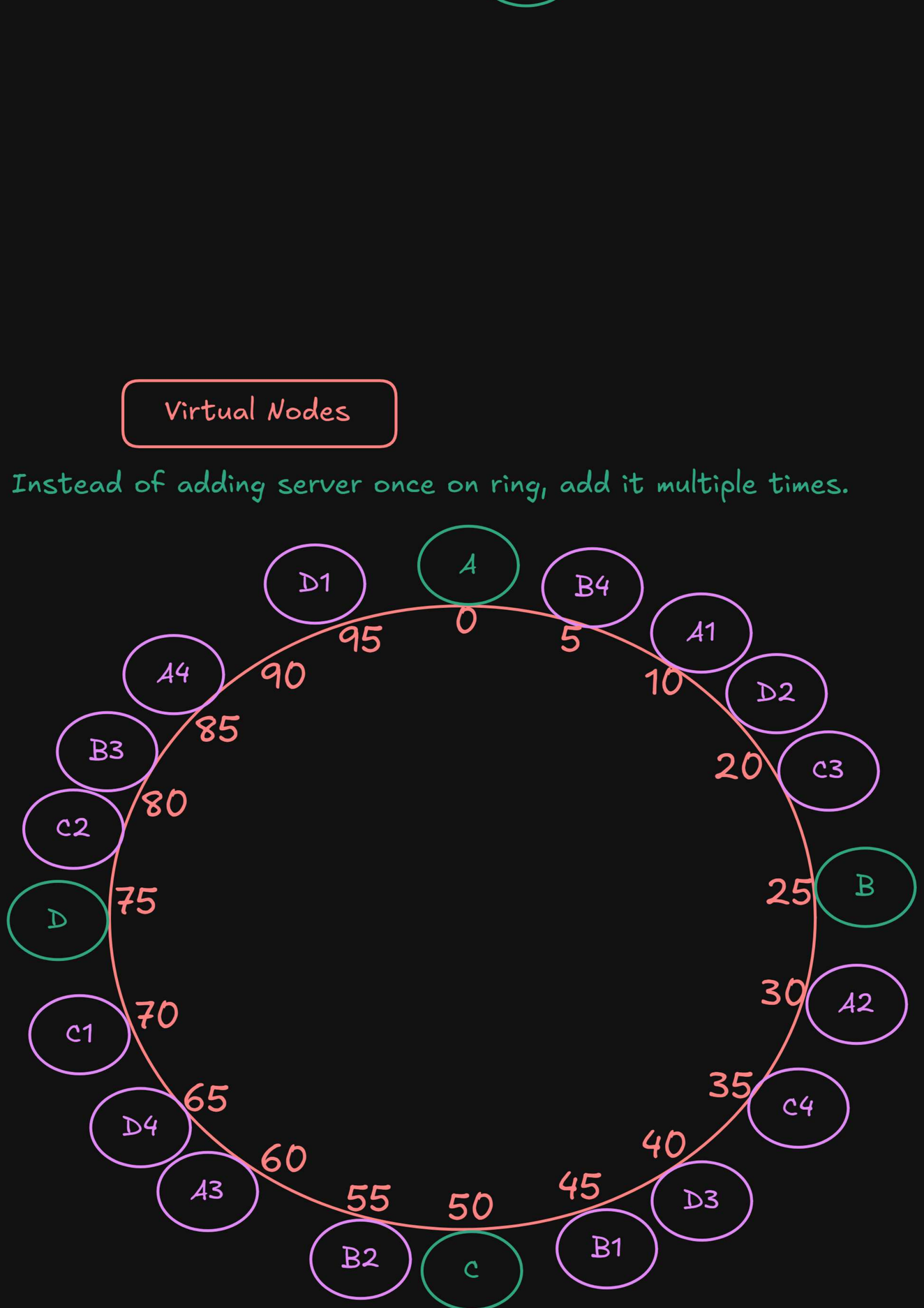
Some servers end up with too many keys.

Others get too few.

Solution: Virtual Nodes (VNodes)

Virtual Nodes

Instead of adding server once on ring, add it multiple times.



Each server handles many small segments
instead of one big chunk

Typical vnode count:

100–200 per server
Cloudflare uses 256 or 512
DynamoDB-style systems use thousands sometimes

Data Movement Math

Without vnodes: ~1/N
With vnodes: even smoother distribution.

In virtual nodes, how do we put a server at multiple points?
Random or systematic?

NOT random.
If it were random, every machine would have inconsistent mappings → disaster.

Virtual Nodes (VNodes) are generated in a deterministic repeatable way.

Server name + index → hash
Example for server A:

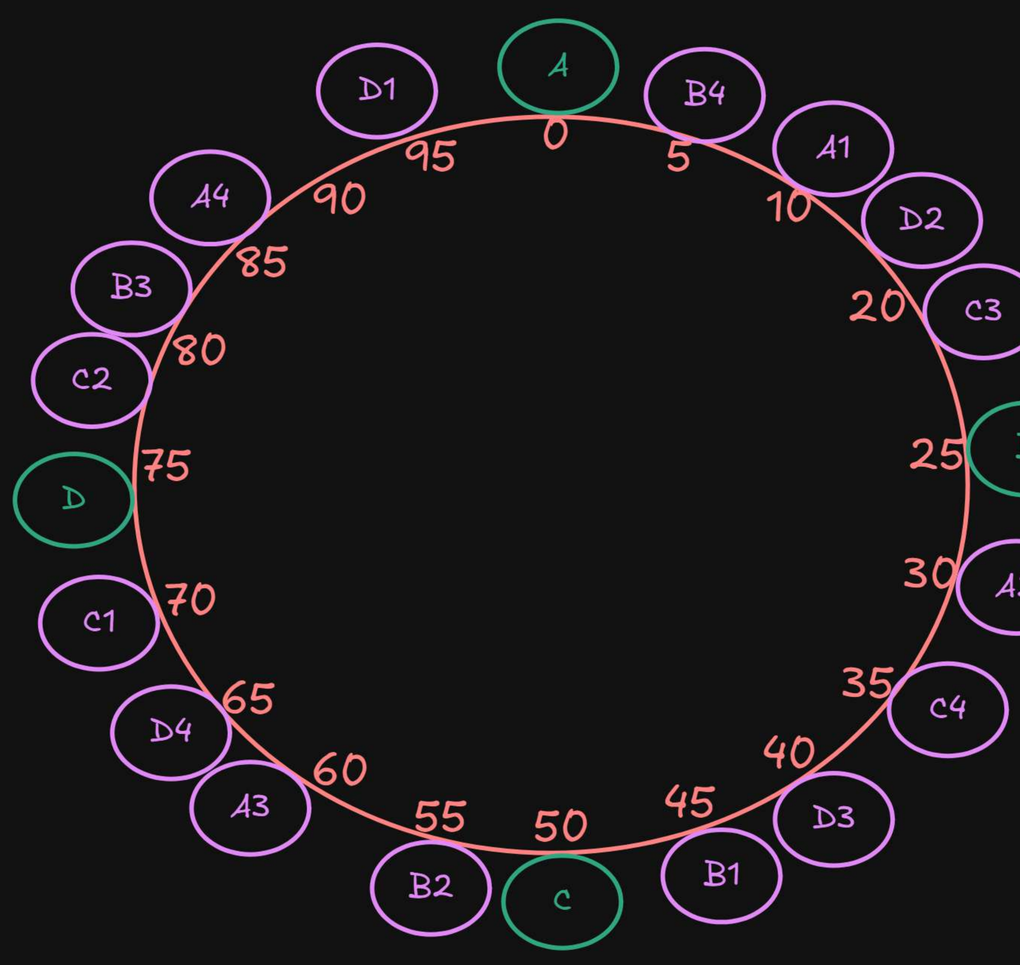
Hash("A#1")
Hash("A#2")
Hash("A#3")
...
Each hash produces a position on the ring.
This ensures:

Deterministic

Every client will place A's virtual nodes at the exact same points.

Good distribution

Because each hash is different, the hash values spread across ring positions.



Why not random?

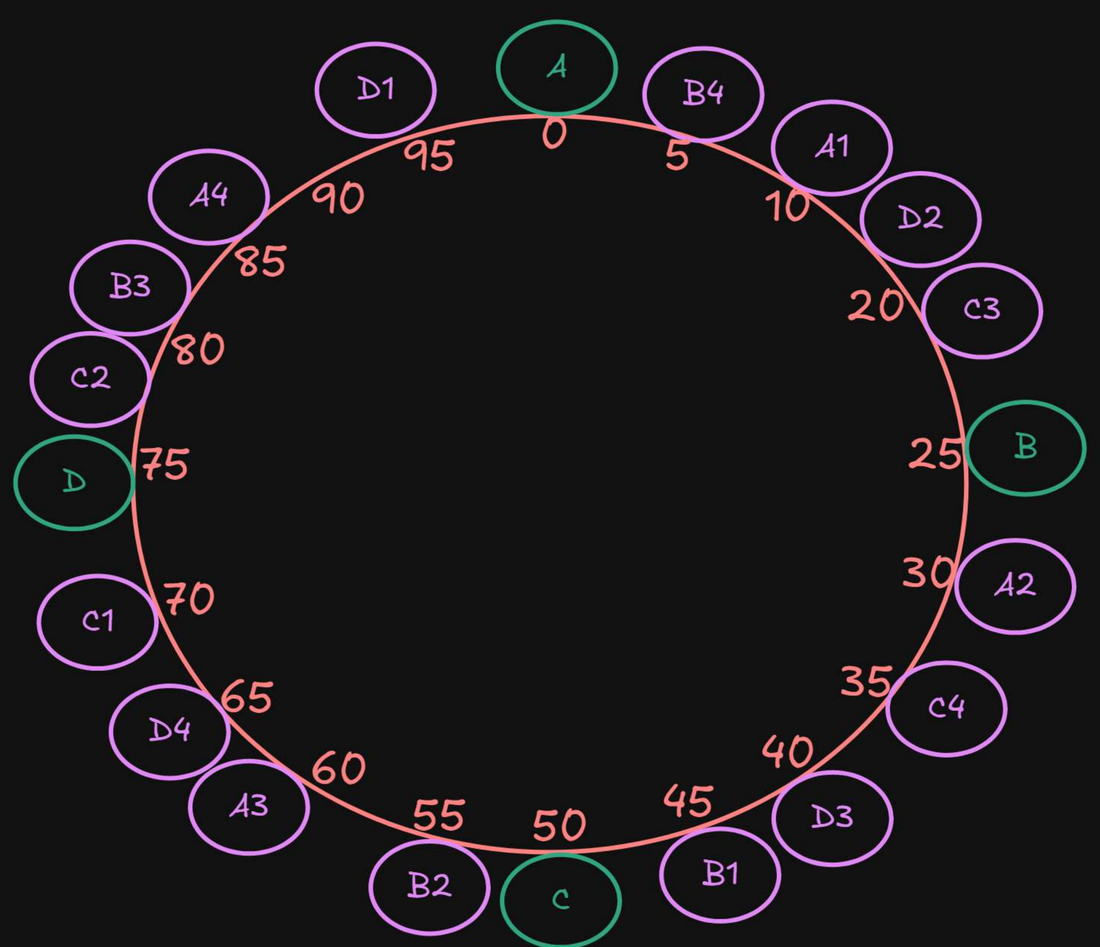
API Server 1

API Server 2

If two API servers
(or two microservices) both
generate the ring, both must
land on the same node placements.

If they used random positions, rings would differ → inconsistent mapping → data loss.

So virtual nodes must be deterministic.



WHAT IF A SERVER IS MORE POWERFUL?

Just give it more virtual nodes.

Example:
Server A (small machine) → 100 vnodes
Server B (big machine) → 300 vnodes

So Server B handles ~3× traffic.

This is how Cassandra, DynamoDB,
Redis Cluster scale heterogenous hardware.

Consistent Hashing WITH REPLICATION

(Also called consistent hashing with R replicas)

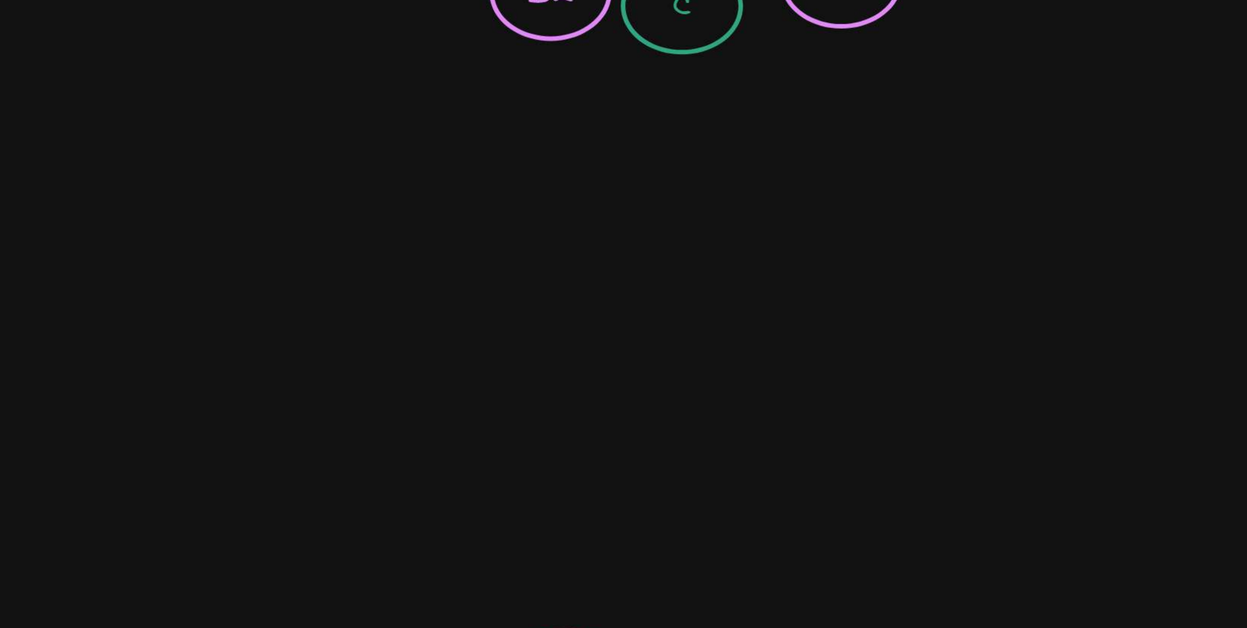
Goal:
If a server fails, data should not be lost.
So, each key is stored on next R servers clockwise.

Let R = 3 replicas.

Replica 1 (primary): A
Replica 2: B
Replica 3: C

What happens if A fails ?

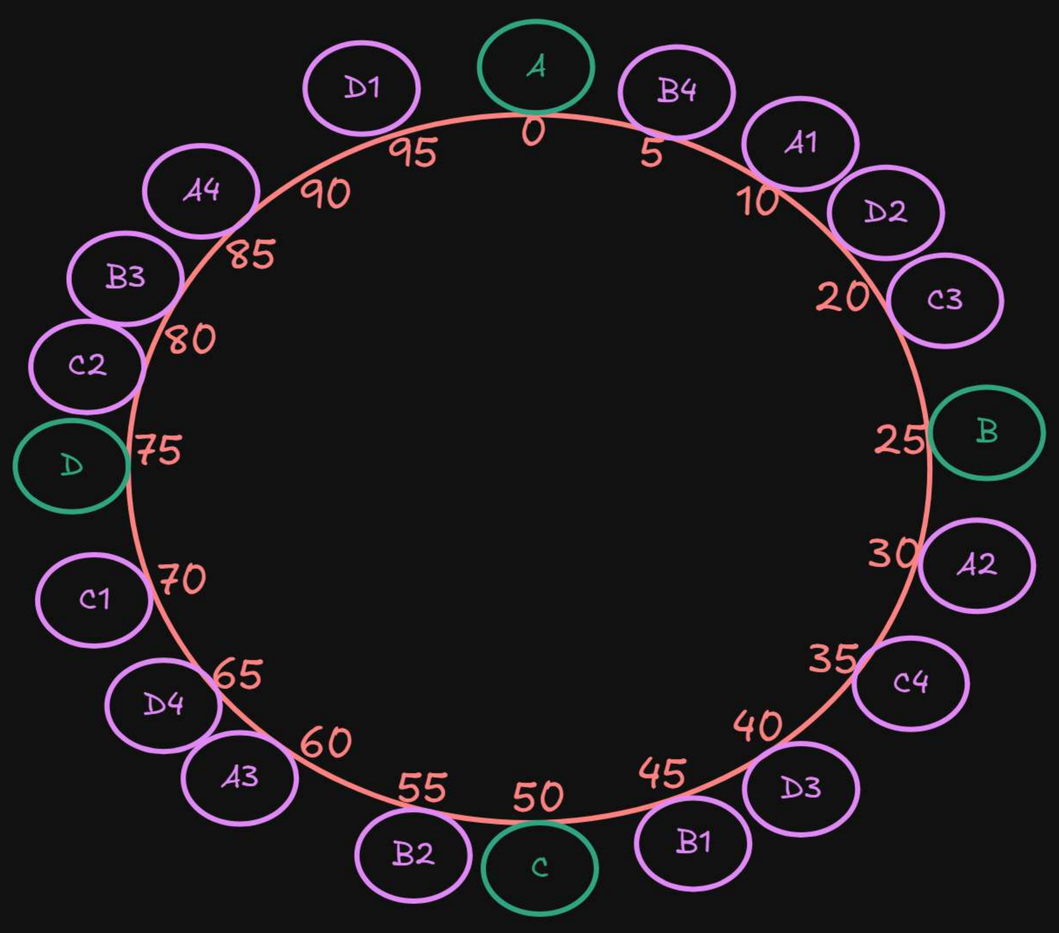
The next available replica is
B → B becomes primary automatically.



Replication with Virtual Nodes

With virtual nodes + replication,
failover does NOT follow the ring to the next vnode.
It follows the replica chain.

So if B and C are replicas and A fails,
the key at 9° goes to the first vnode of B,
not to C3 and not to D2.



Popular Implementations of Consistent Hashing

Consistent hashing isn't limited to databases alone.
Any system that needs to spread data or workload across
multiple machines can benefit from it — whether that cluster
consists of caches, message brokers, storage nodes,
or even application servers.
This technique shows up in many large-scale production systems, such as:

Apache Cassandra – distributes partitions across nodes using a hashing ring

Amazon DynamoDB – relies on consistent hashing internally

CDNs – Uses hashing to map users to edge servers

It's a foundational concept behind many modern, highly scalable distributed architectures.

When should you mention consistent hashing in an interview?

Honestly, not as often as people think. Most modern distributed systems
already handle data distribution for you.
If you're using tools like DynamoDB, Cassandra, Redis Cluster,
or even sharded databases, you can simply mention that these
systems rely on consistent hashing or a variation of it to
manage scaling and sharding internally.

Where it does matter is in interviews that dive deep into system internals.
If the interviewer asks you to build a distributed datastore,
cache, or message queue from scratch, that's when
consistent hashing becomes essential. You'll want to walk through:

Why simple % N sharding breaks as servers change
How consistent hashing fixes this
How virtual nodes smooth traffic across the cluster
How to handle node additions/failures
How to deal with hot keys and rebalance gracefully

Most system design rounds won't require all of this detail
— but infrastructure interviews absolutely will.