CSCU9YQ - NoSQL Databases
Lecture 3: Distribution Models

Gabriela Ochoa

Outline

- Recap
  - Characteristics of NoSQL DBs
  - Data models
  - The need to scale
- Distribution models
  - Replication
  - Sharding
  - Combining Sharding and replication

Characteristics of NoSQL Databases

- Not using the Relational Model
- Running well on clusters
- Open-source
- Flexible schemas (freely add fields to DB records, without needing to define a fixed schema first)
- Big data, web applications

Data Models

- An abstract representation organising elements of data and how they relate to one another an to properties of real world entities
- The dominant data model fore several decades has been the Relational Model
- The primary way in which non-tabular databases differ from relational databases is the data model
- Three main categories

Types of NoSQL DB by Data Model

Data Model	Key Characteristics
Document	• Store data in documents. Use structure like JSON • Intuitive way, closely aligned with OO programming (docs are objects) • Each record and its associated data are stored together, rather than distributed across tables.
Key-Value and Column-Family	• Most basic types of non-tabular DB • Data stored as an attribute name (key), with its value • Value is entirely opaque, data can only be queried by the key.
Graph	• Store data in graphs structures with nodes, edges and properties • Data modelled as a network of relationships between specific elements. (e.g. social networks)

Main Factor for No-SQL Emergence: Clusters

- Increase of scale (what is now called Big Data), produced a need of more computing resources
- Two options
  – Scaling up (vertical): bigger machines, more processors, disk, storage and memory. More expensive! (also limits)
  – Scaling out (horizontal): use a lots of small machines in a cluster. Much cheaper, and more resilient (keep going despite failures)
- RDB are not designed to run efficiently on clusters, while NoSQL have been designed to run on clusters

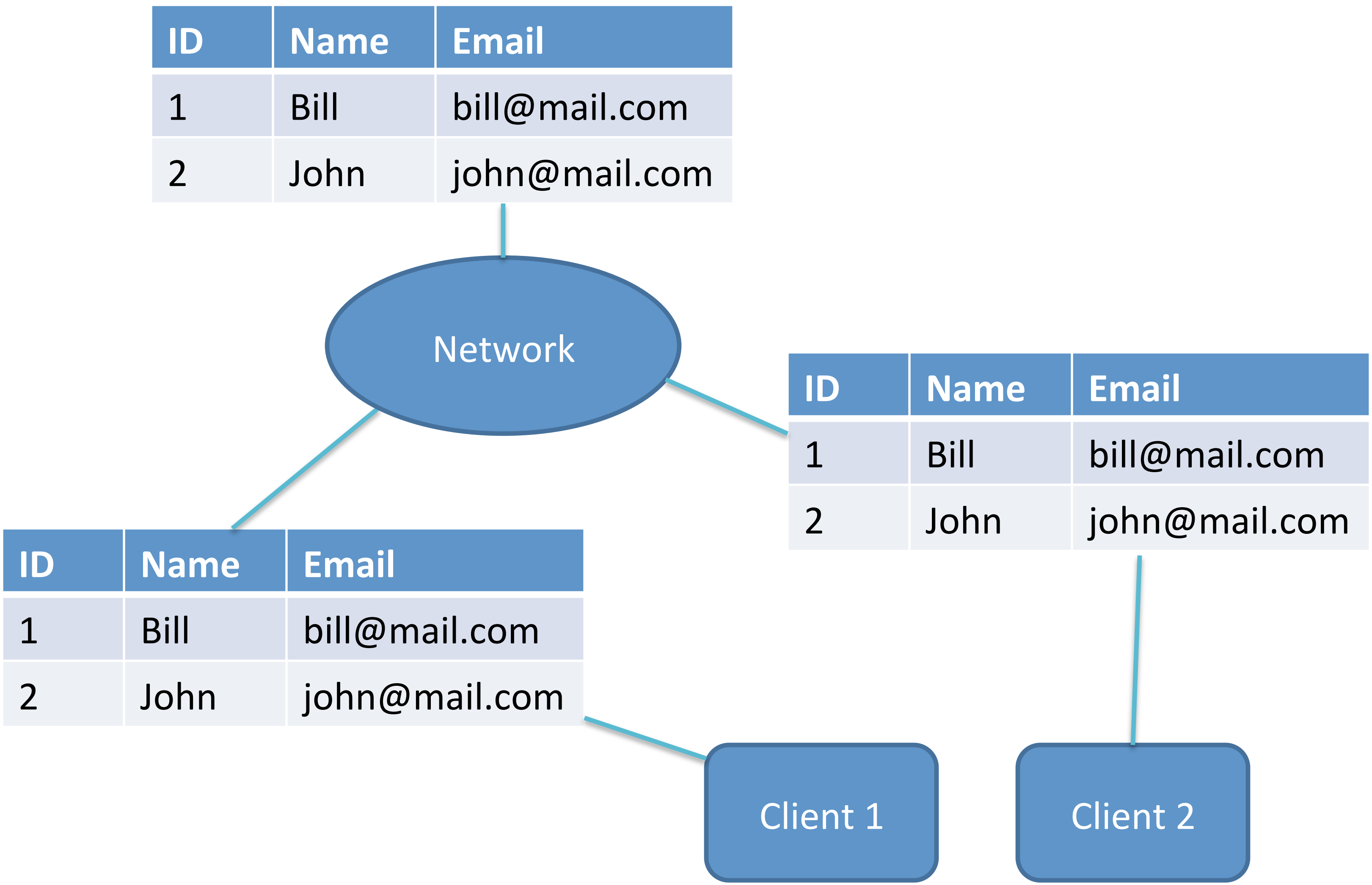
Distribution Models: Summary

- Replication
  - Copies data across multiple servers
  - Each data can be found in multiple places
  - Synchronous or Asynchronous
  - Asynchronous
    1. Primary-secondary
    2. Peer-to-pear
- Sharding
  - Distributes different data across multiple servers.
  - Each server acts as the single source for a subset of data
- Combining Replication with Sharding

Replication

- Replication provides redundancy and increases data availability.
- **Mainly** for resilience - in a big network, things WILL fail. Your system needs to continue as if nothing happened (Fault Tolerance)
- Also helps with performance
  - Reduce data distance travel by accessing local replica
  - Load balancing

Replication



Replication Resilience

ID	Name	Email
1	Bill	bill@mail.com
2	John	john@mail.com

Network

ID	Name	Email
1	Bill	bill@mail.com
2	John	john@mail.com

ID	Name	Email
1	Bill	bill@mail.com
2	John	john@mail.com

Client 1

Client 2

Types of Replication

- Synchronous
  - All replicas are updated on every write
  - Reads are guaranteed to be up to date, so you can read from any node
  - Maintains benefit of resilience, but can be too slow for some applications
  - Used if reads MUST be up to date
  - Works best if fewer writes are made

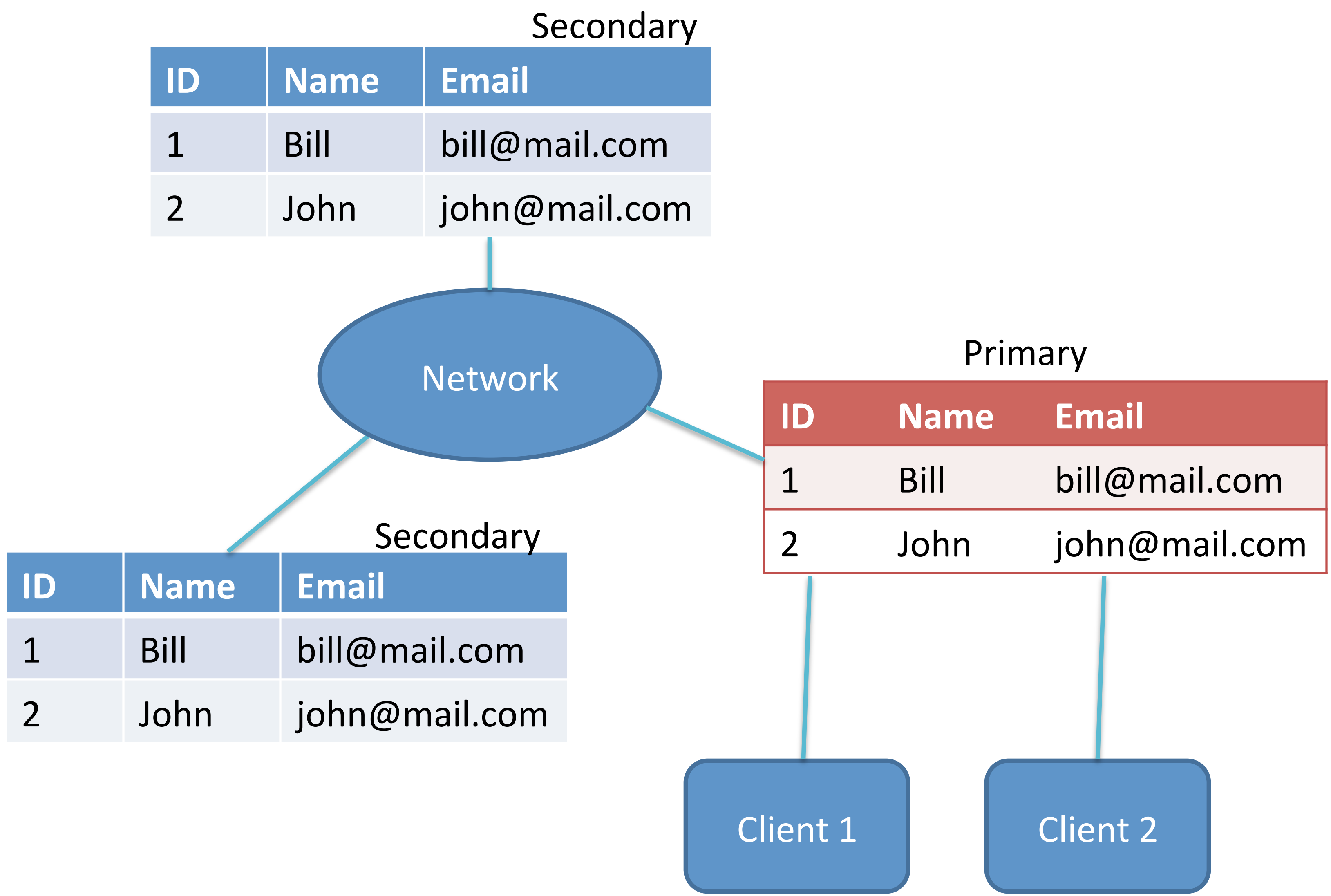
Types of Replication

- Asynchronous
  - Writes are propagated as soon as possible, but reads do not have to wait
  - Reads can be out of date
  - Eventual consistency
  - Works best if reads can be a little out of date
  - Methods include
    - Primary-secondary
    - Peer-to-peer

Primary-secondary Method

- A primary node, and 1 or more secondary nodes
- Not a SPOF (single point of failure) – if it fails, one of the secondaries will be promoted to primary
- Writes go only to the primary and are then propagated
- Secondaries can be read but not written
- Replication factor is the number of replicas made
- Generally, it is considered safe to have three copies in total

Primary – secondary replication



Primary in MongoDB

Secondary

ID	Name	Email
1	Bill	bill@mail.com
2	John	john@mail.com

Network

Primary

ID	Name	Email
1	Bill	bill@mail.com
2	John	john@mail.com

Client 1

Client 2

Consistency

- In a relational database, ACID consistency is handled by transactions, which maintain database integrity
- In NoSQL, consistency refers to whether or not reads reflect previous writes
  - Strict consistency – A read is guaranteed to get the most up to date data
  - Eventual consistency – Read data may be stale, but writes will catch up pretty quick

Back to MongoDB

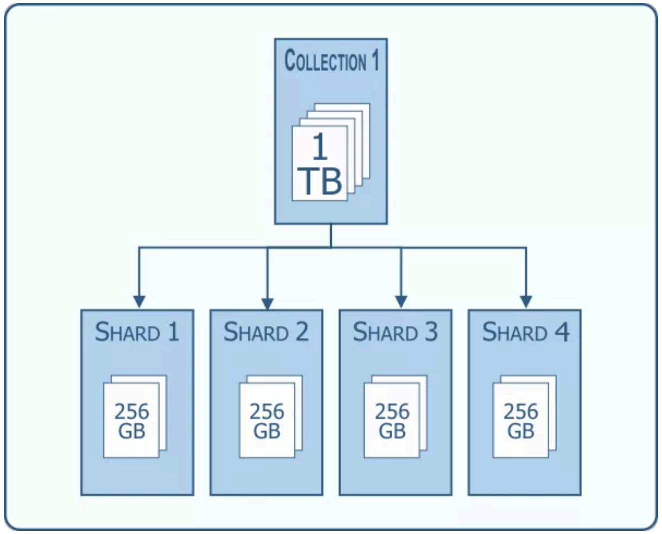
- Always reading from the primary gives strict consistency
- Reads from a secondary may be stale, so this is eventual consistency
- Why use eventual consistency?
  - For performance, reads can be spread across multiple secondary nodes
  - Some applications, e.g. Facebook don’t need strict consistency
  - Offline analytics should read from secondaries to avoid overloading primary

Peer-to-Peer Replication

- The primary node can be a write bottleneck
- In systems with a high write rate, this can cause latency
- Peer-to-peer replication allows all nodes to accept reads and writes (there is no primary node)
- Fast and resilient
- Problems with inconsistency – particularly write inconsistency where two peers receive conflicting updates

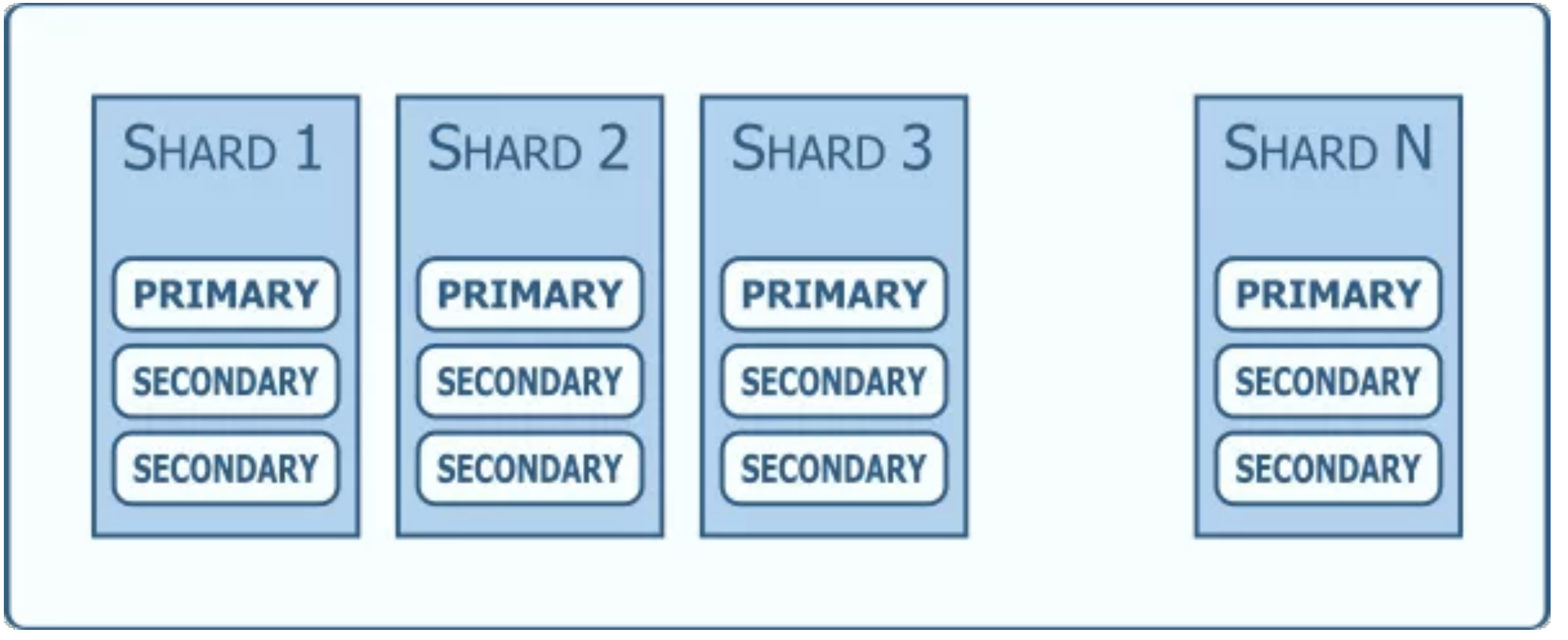
Sharding

- Spreads the data across nodes in the cluster
- Each node gets a subset of the data
- Might split the data by location (UK customer data in Ireland, American customer data in NY)
- Aggregate model helps sharding work



In MongoDB data is sharded at a collection level

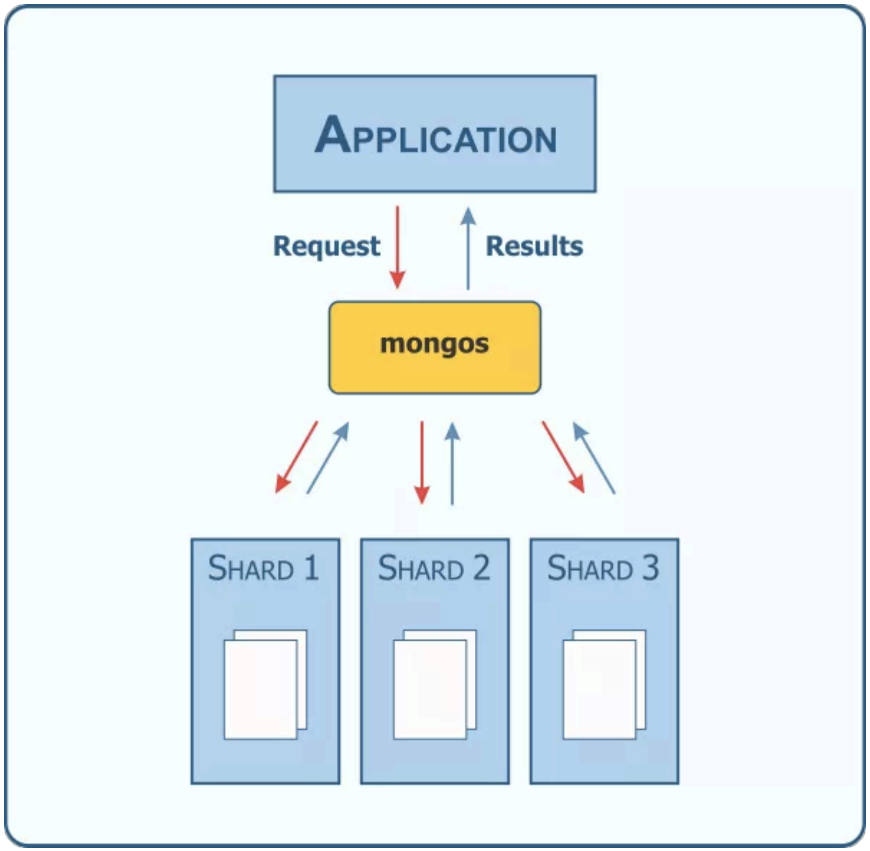
Combining Sharding with Relication



- A *replica set* is a group of mongod instances that maintain the same data set.
- Each cluster is, to keep redundancy, areplica sets with primary and secondary servers.

https://www.kenwalger.com/blog/nosql/mongodb/mongodb-horizontal-scaling-sharding/

Cluster configuration with MongoDB

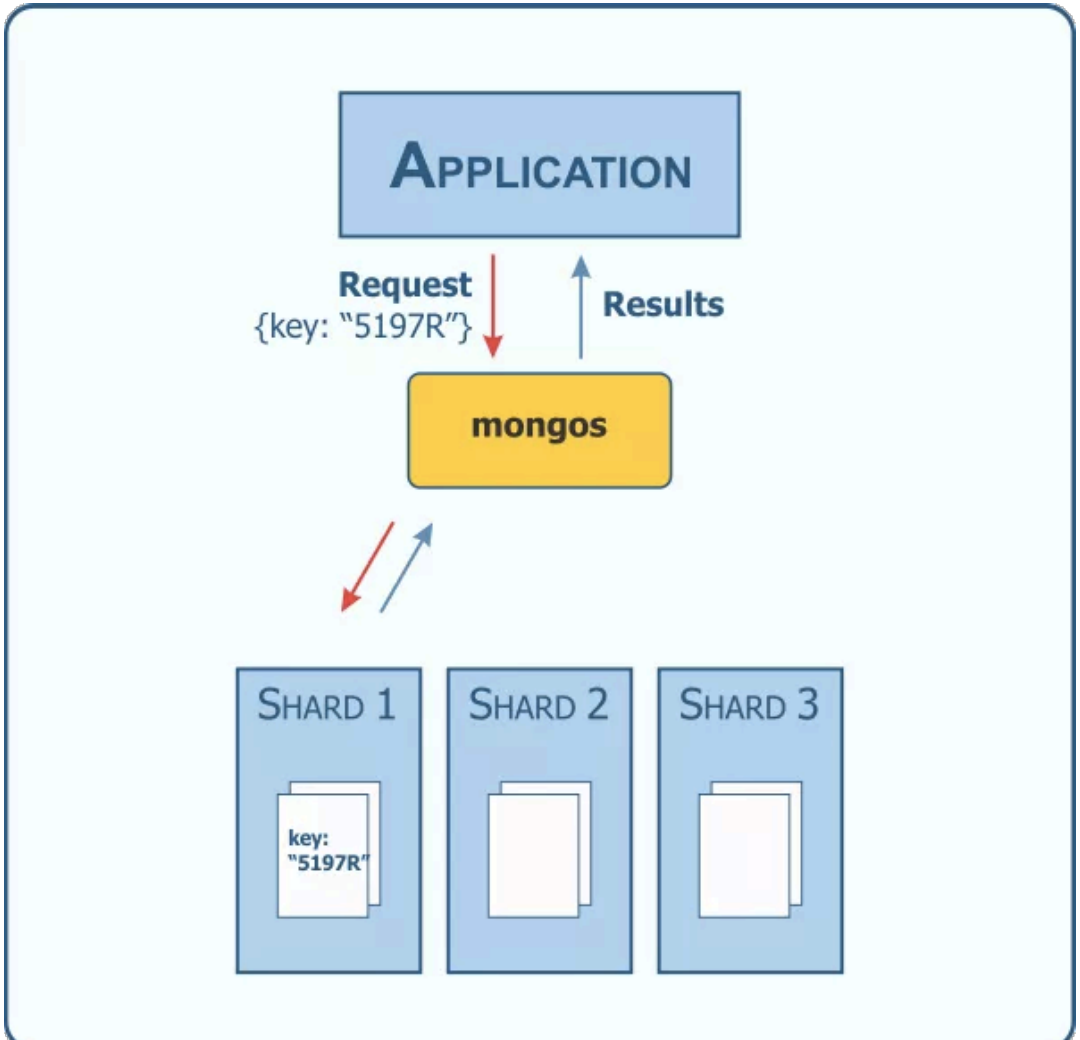


- mongos for “MongoDB Shard,” is a routing service for MongoDB shard configurations
- mongos acts as the interface between application and the data
- It routs queries and write operations to the appropriate shards
- An application access the data through mongos not touching the data directly
- Queries are directed to all shards unless it can be determined that the data resides on a particular shard

Shard Keys

- There has to be a better way than broadcasting that request to all shards
- Use a Key: to determine how documents in a collection are distributed across the shards.
- The Key is based on one or more chosen fields – much like a primary key in a RDB
- Shard keys can be chosen by hand but most systems offer auto-sharding (including resharding to maintain load balance)

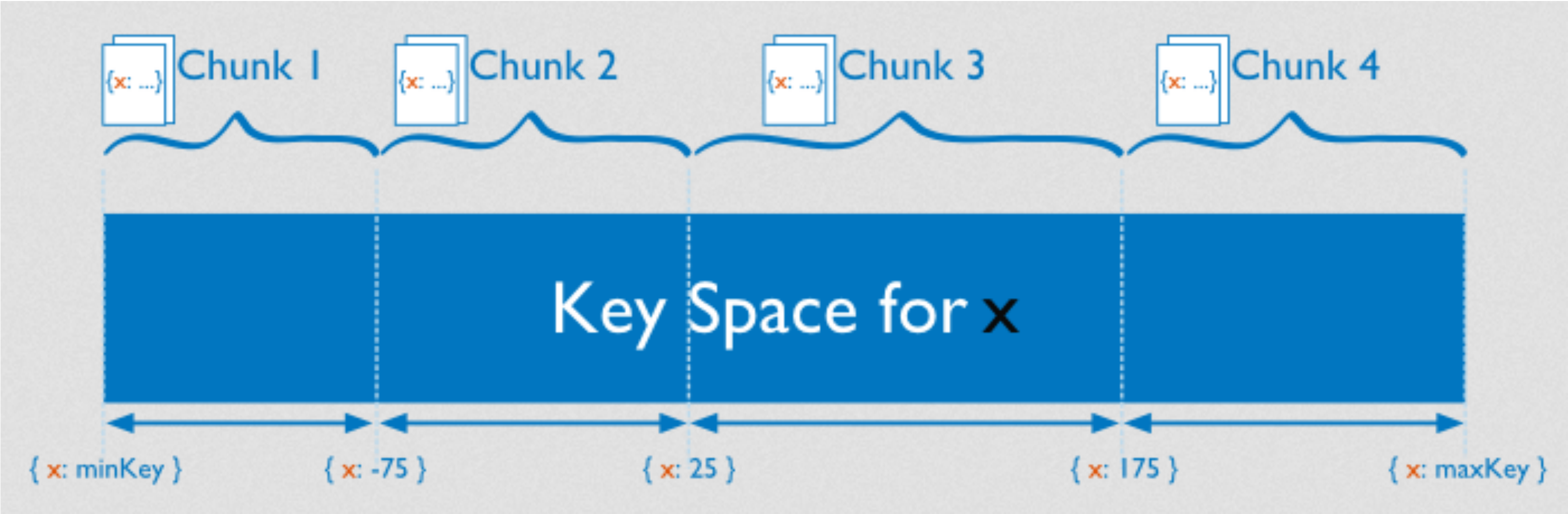
- If provided with a shard key during a query, the mongos knows how to route the request. More efficient!
- The choice of shard key affects the performance, efficiency, and scalability of a sharded cluster.



Choosing a Shard Key

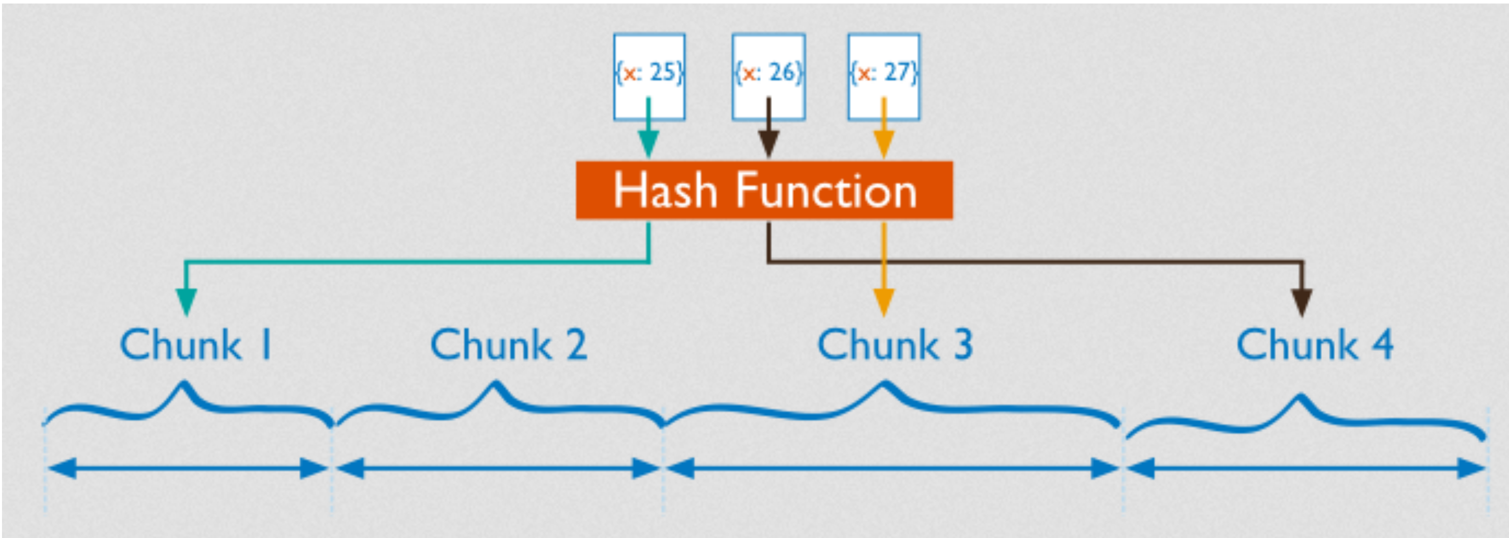
- Field(s) must appear in every record
- Easily divisible, with wide number of possible values
- Uniformly distributed across records and queries
- Heavily dependent on your application
- Think about fields that appear in common queries

Range Based Keys



- Divides the data set into ranges determined by the shard key values to provide **range based partitioning**.
- Consider a numeric shard key: If you visualize a line that goes from negative infinity to positive infinity, each value of the shard key falls at some point on that line.
- The line can be divided into smaller, non-overlapping ranges called **chunks** where a chunk is range of values from some minimum value to some maximum value.
- Documents with “close” shard key values are likely to be in the same chunk, and therefore on the same shard.

Hash Based Keys



- Compute a hash of a field's value, and then uses these hashes to create chunks (*hash based partitioning)*.
- Two documents with "close" shard key values are *unlikely* to be part of the same chunk.
- This ensures a more random distribution of a collection in the cluster.

Summary

- Replication
  - Copies data across multiple servers
  - Each data can be found in multiple places
  - Synchronous or Asynchronous
  - Asynchronous
    1. Primary-secondary
    2. Peer-to-pear
- Sharding
  - Distributes different data across multiple servers.
  - Each server acts as the single source for a subset of data
- Combining Replication with Sharding (MongoDB)