Partitioning

Systems Analysis & Design

Learning Objectives

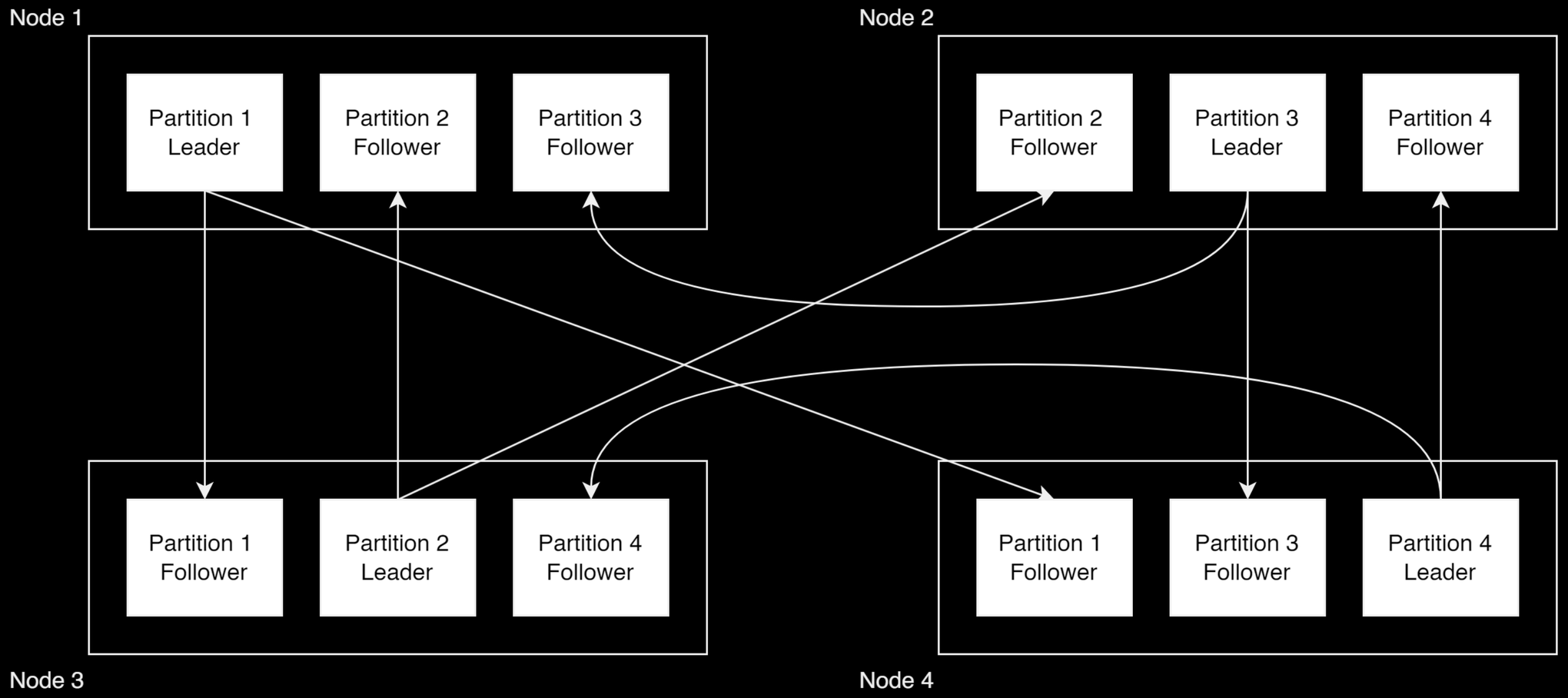
By the end of this session, you will have acquired the following information:

- Scalability through Partitioning
- Partitioning by Key Range
- Partitioning by Hash of Key
- Partitioning and Secondary Indexes

Scalability through Partitioning

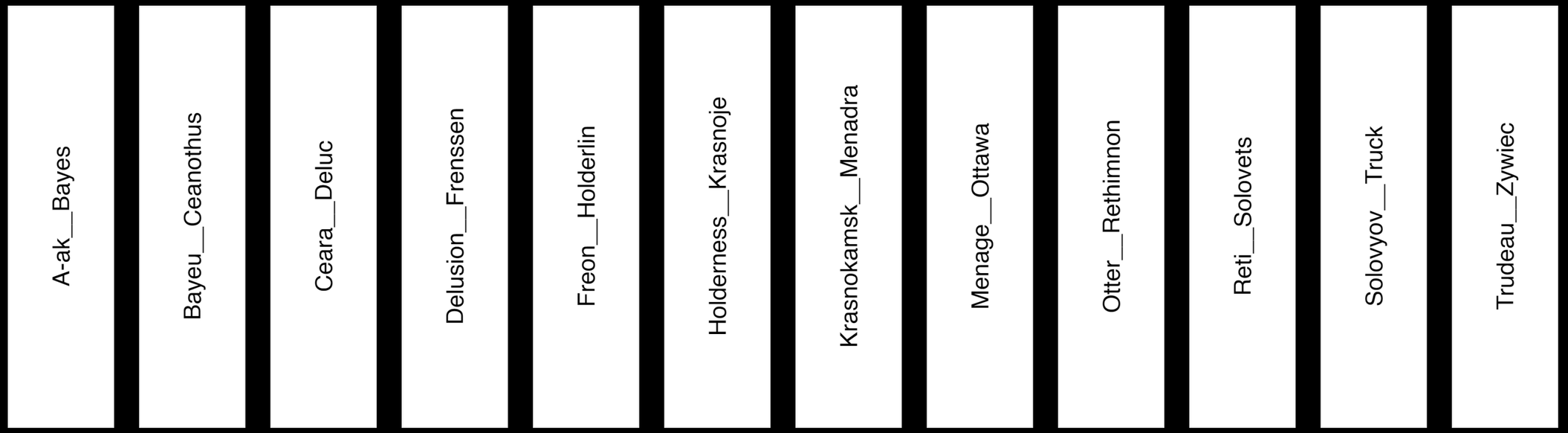
- Replication is insufficient for very large datasets or high query throughput.
- Data needs to be divided into partitions, also known as *sharding*.
- Each piece of data is assigned to exactly one partition in the defined partitions.
- The main reason for wanting to partition data is scalability.

- Partitioning is usually combined with replication so that copies of each partition are stored on multiple nodes.
- Each node serves as a leader for some partitions and as a follower for others.



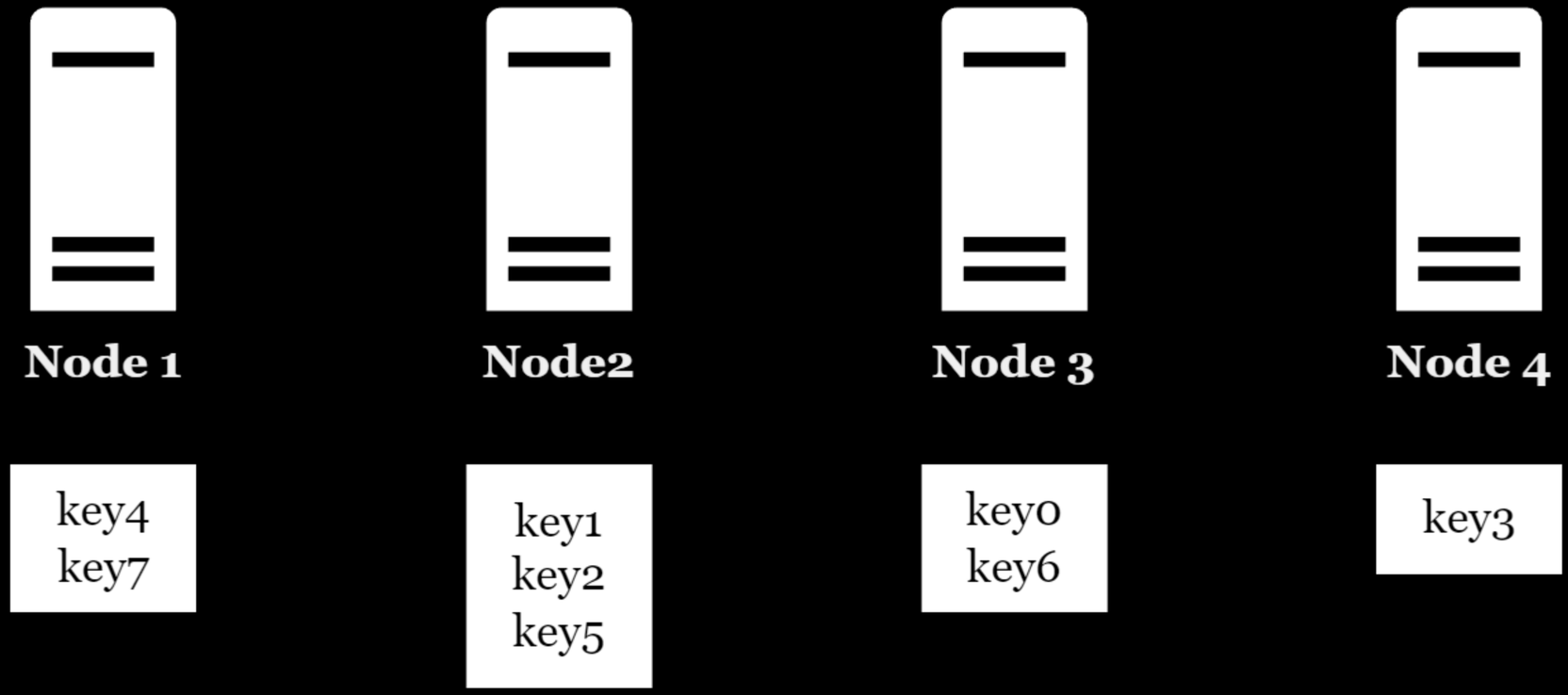
Partitioning by Key Range

- Partitioning assigns a continuous range of keys to each partition, enabling direct requests to the appropriate node if partition-node assignments are known.
- The ranges of keys are not necessarily evenly spaced, because your data may not be evenly distributed.
- The downside of key-range partitioning is that it can lead to hot spots due to certain access patterns.



Partitioning by Hash of Key

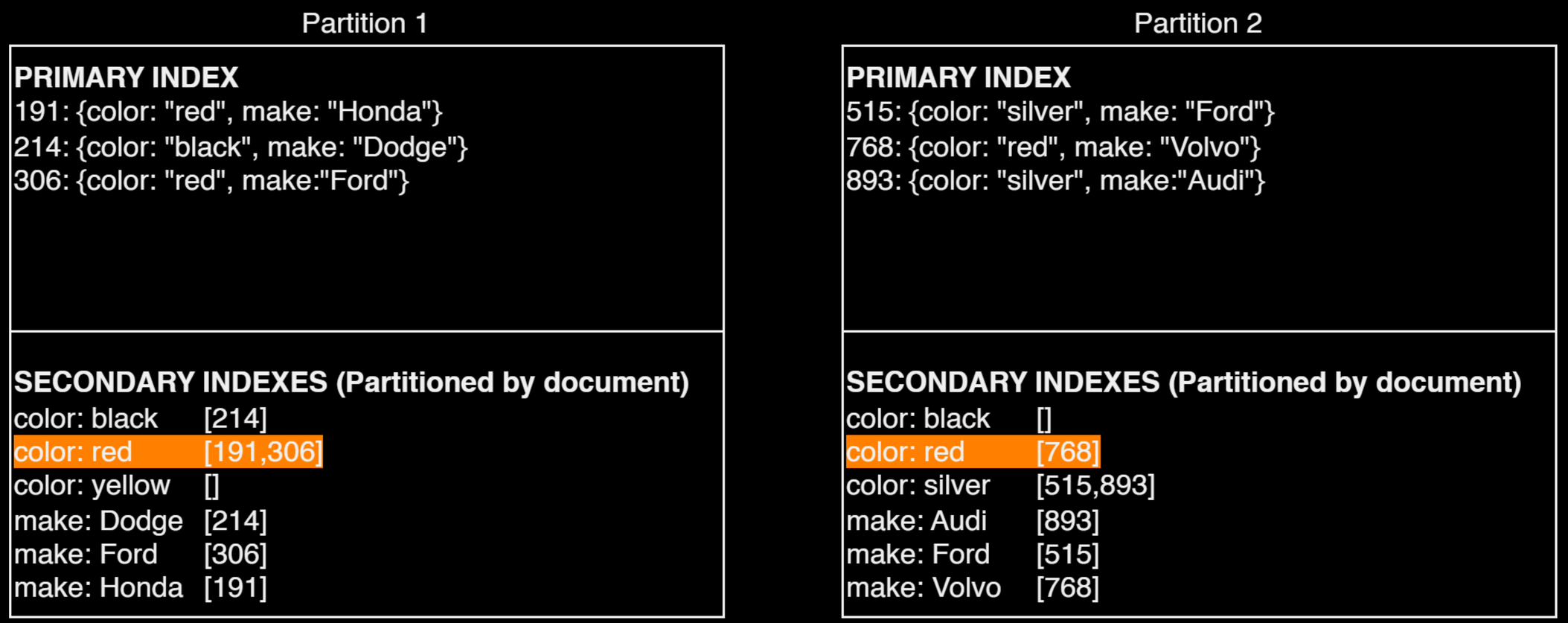
- Many distributed datastores use a hash function to determine the partition for a given key.
- By using the hash of the key for partitioning, we lose a beneficial property of key-range partitioning: the ability to perform efficient range queries.



Partitioning and Secondary Indexes

- If records are only ever accessed via their primary key, we can determine the partition from that key and use it to route read and write requests to the partition responsible for that key.
- Secondary indexes complicate the situation as they search for specific value occurrences, not unique record identification.
- **Examples**
  - find all actions by user *aaghamohammadi*
  - find all articles containing the word *Alireza*
  - find all cars whose color is *red*

- Each partition maintains own secondary indexes, covering only the documents in that partition
- The approach to querying a partitioned database is known as *scatter/gather*



- Instead of each partition having its own secondary index (a local index), a global index that covers data across all partitions can be constructed.
- While a global index must also be partitioned, its partitioning can differ from that of the primary key index.

Partition 1

PRIMARY INDEX
191: {color: "red", make: "Honda"}
214: {color: "black", make: "Dodge"}
306: {color: "red", make:"Ford"}

SECONDARY INDEXES (Partitioned by term)

color: black	[214]
color: red	[191,306,768]
make: Audi	[893]
make: Dodge	[214]
make: Ford	[306,515]

Partition 2

PRIMARY INDEX
515: {color: "silver", make: "Ford"}
768: {color: "red", make: "Volvo"}
893: {color: "silver", make:"Audi"}

SECONDARY INDEXES (Partitioned by term)

color: silver	[515,893]
color: yellow	[]
make: Honda	[191]
make: Volvo	[768]

Further Resources

- Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems (pages 221-241)