Scaling Out

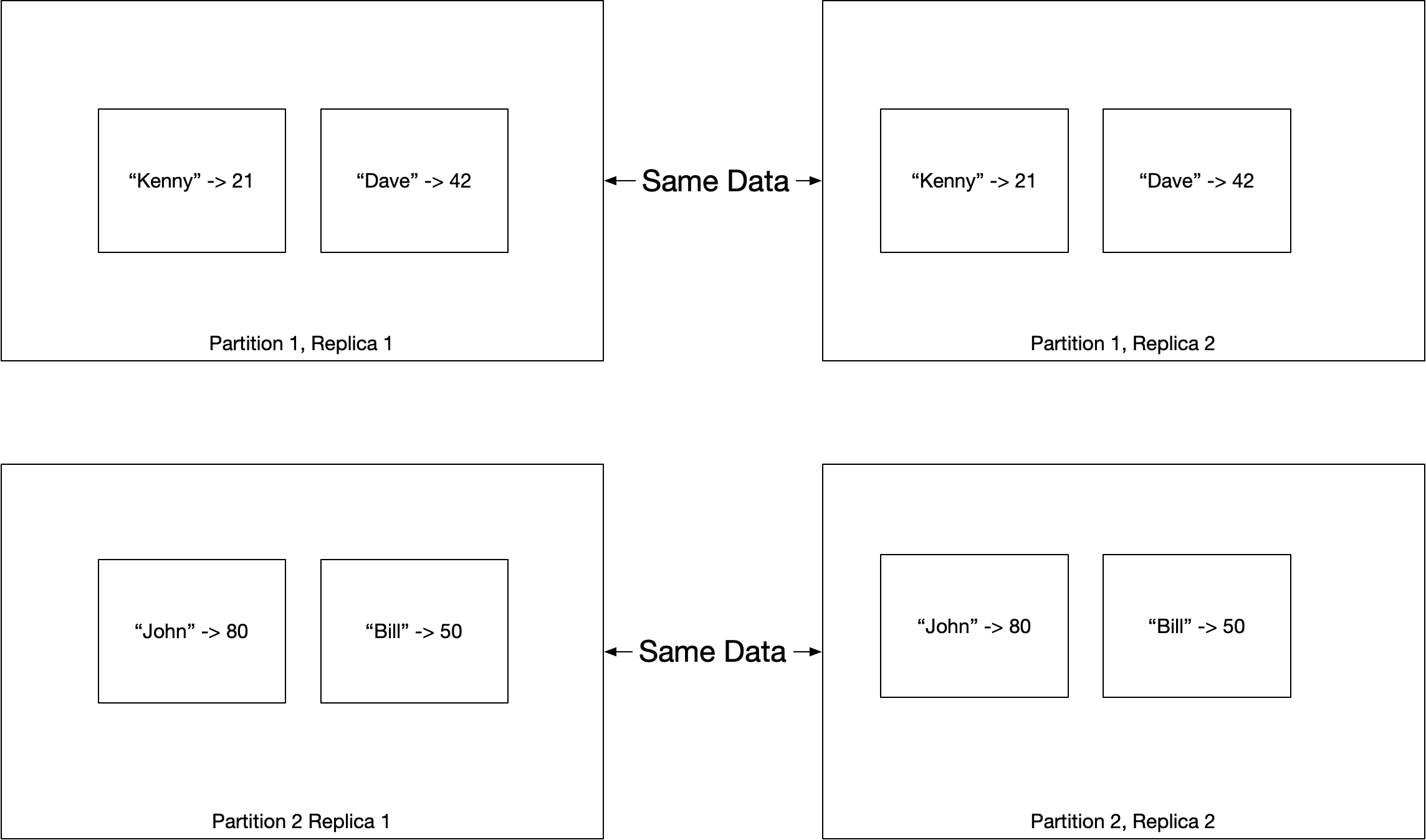
## Overview

We scale out in one of two ways.

* Replication – storing the same data on multiple nodes.
* Partitioning/Sharding – Splitting the data over multiple nodes.

The two approaches are not mutually exclusive and indeed often a system will combine both approaches.

Table 1 Partitioning and Replication



Replication can be used to increase fault tolerance and availability; increase read throughput and reduce latency by placing replicas close to clients. Partitioning is typically used to increase write throughput.

### Replication

The biggest problem with replication is keeping the data on each replica in sync. There are three popular methodologies which we will consider in turn.

* Single leader
* Multi leader
* Leaderless

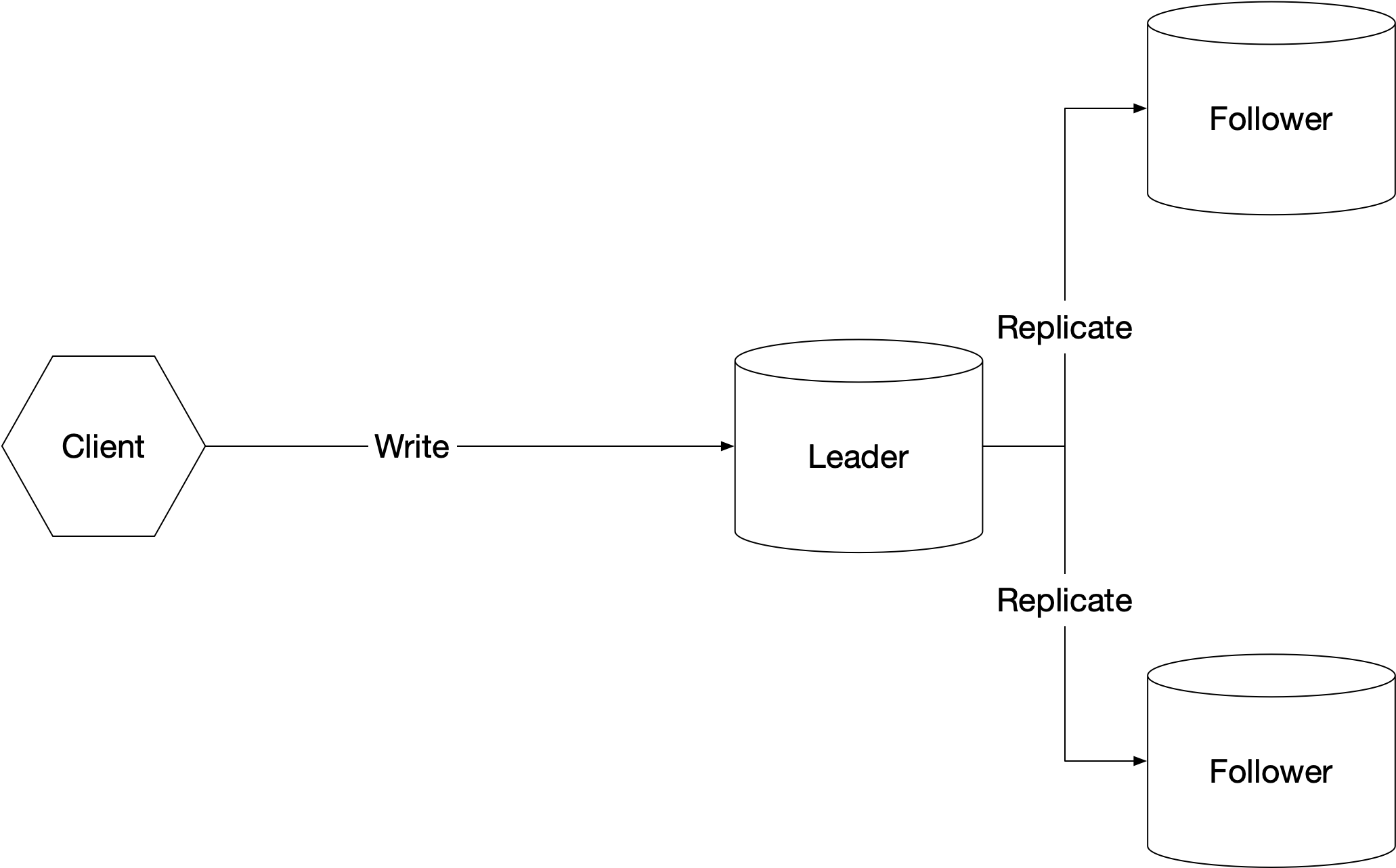
#### Single Leader

A single leader system is also known as leader-based replication, active/passive, or master/slave. This methodology is support by PostgreSQL, MySQL, Oracle, SQL Server, MongoDB, Kafka and RabbitMQ.

##### Updates

The algorithm works as follows.

1. All clients write to the leader (AKA master or primary)
2. The leader writes data to its local storage
3. The leader then updates the followers (AKA slaves or secondaries) by sending a replication log (AKA change stream)
4. Clients can read from the leader or any follower.



##### Synchronous Or Asynchronous Replication

Synchronous replication ensures the follower is fully consistent with the leader but if the follower is down the write cannot proceed. Typically, we update a single follower synchronously to ensure we have at least one fully consistent copy of the data and update all other followers asynchronously.

##### New Followers

When a new follower is added it takes a snapshot of the leader’s data and makes a note of the log sequence number of the most recent record in the snapshot. Once it is up and running it asks the leader for all updates that are more recent than its log sequence number.

##### Downtime of Follower

If a node is down, either due to failure or planned maintenance, when it comes back up it needs to update its state to reflect any updates that occurred when it was down. It does this by asking the leader to send all updates that have occurred since the log sequence number of the last update it received before going down.

##### Failover

When a leader goes down, we need to failover to a new leader. Typically, the new leader is chosen amongst the followers as the node with the most up to date data

##### Replication Logs

## Partitioning

Sharding splits data across multiple data stores in such a way that we can work out which information is on which host. For an in-depth discussion of partitioning see

[Azure Best Practices For Data Partitioning](https://docs.microsoft.com/en-us/azure/architecture/best-practices/data-partitioning)

Table 2 Benefits of Partitioning

|  |  |
| --- | --- |
| Scalability | Dividing data across multiple data stores prevents us being limited by the physical limits of a single store |
| Performance | Splitting data cross multiple data stores can lead to better performance as we need to search through a smaller amount of data on each partitioned store. |
| Flexibility | We can allocate different types of data to different types of data store. In this way the data store used is the one most appropriate for the type of data. |
| Availability |  |

There are several different ways of partitioning the data.

### Horizontal Partitioning / Sharding

Each partition is separate data store. All data stores have the same schema and holds a subset of the data. The following sections describe some strategies for allocating subsets to partitions

#### Range Strategy

### Vertical

Each partition holds a subset of the fields. Fields are divided according to how they are used e.g. frequently accessed fields might go into one partition.

#### Functional

The partitions are determined by the bounded contexts of the architectural solution. We might put orders in one partition and product definitions in another partition.

#### Key/Hash Based

Given N servers put the data on mod(key,n). As we add servers we need to repartition all the data which is expensive.

#### Directory based

Use a lookup table to prevent repartitioning as we add servers. The drawback of this approach is that the lookup can become a single source of failure and the extra level of performance can impact performance.