Chapter – 4

Distribution Models

Introduction

- Data volumes increase, it becomes more difficult and expensive to scale up—buy a bigger server.
- A more appealing option is to scale out—run the database on a cluster of servers.
- Running over a cluster introduces complexity so it's not something to do unless the benefits are compelling.
- Broadly, there are two paths to data distribution:

1. **Replication**
2. **Sharding**

Introduction

- ***Replication takes the same data and copies it over multiple nodes.***
- ***Sharding puts different data on different nodes.***
- Replication and sharding are orthogonal techniques: You can use either or both of them.
- Replication comes into two forms: **master-slave** and **peer-to-peer**.

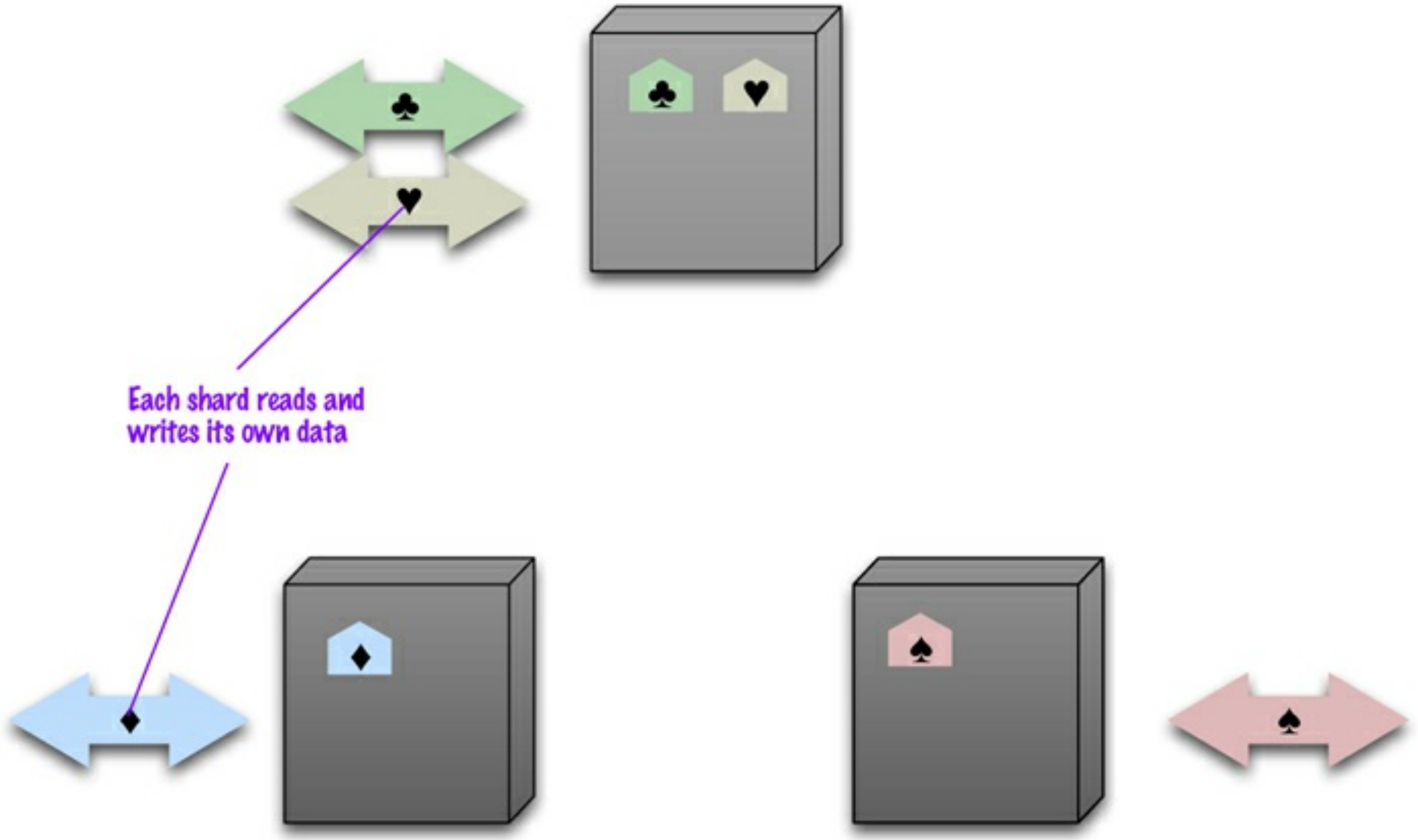
Single Server

- No distribution at all.
- Run the database on a single machine that handles all the reads and writes to the data store.
- Lot of NoSQL databases are designed to run on a cluster, it can make sense to use NoSQL with a single-server distribution model.
- **Graph databases** are the obvious category here—these work best in a single-server configuration.

Sharding

- A busy data store is busy because different people are accessing different parts of the dataset. In these circumstances we can support horizontal scalability by putting different parts of the data onto different servers—a technique that's called **Sharding.**

Sharding



Sharding puts different data on separate nodes, each of which does its own reads and writes.

Sharding

- In the ideal case, the load is balanced out nicely between servers—for example, if we have ten servers, each one only has to handle 10% of the load.
- For this, we have to ensure that data that's accessed together is clumped together on the same node and that these clumps are arranged on the nodes to provide the best data access.
- The first part of this question is how to clump the data up so that one user mostly gets her data from a single server. This is where aggregate orientation comes in really handy. The whole point of aggregates is that we design them to combine data that's commonly accessed together—so aggregates leap out as an obvious unit of distribution.

Sharding

- When it comes to arranging the data on the nodes, there are several factors that can help improve performance:

1. Physical Location
2. If some data tends to be accessed on certain days of the week—so there may be domain-specific rules you'd like to use.
3. Aggregates are evenly distributed across all the nodes.
4. Put aggregates together if you think they may be read in sequence.

Sharding

- ***Sharding can be done as part of application logic.***
- You might put all customers with surnames starting from A to D on one shard and E to G on another.
- This complicates the programming model, as application code needs to ensure that queries are distributed across the various shards.
- Rebalancing the sharding means changing the application code and migrating the data.
- **Auto-sharding**, where the database takes on the responsibility of allocating data to shards and ensuring that data access goes to the right shard.

Sharding

- Sharding can improve both read and write performance.
- Although the data is on different nodes, ***a node failure makes that shard's data unavailable*** just as surely as it does for a single-server solution.
- Only the users of the data on that shard will suffer.
- It's not good to have a database with part of its data missing.
- Clusters usually try to use less reliable machines, and you're more likely to get a node failure. So in practice, sharding alone is likely to decrease resilience.

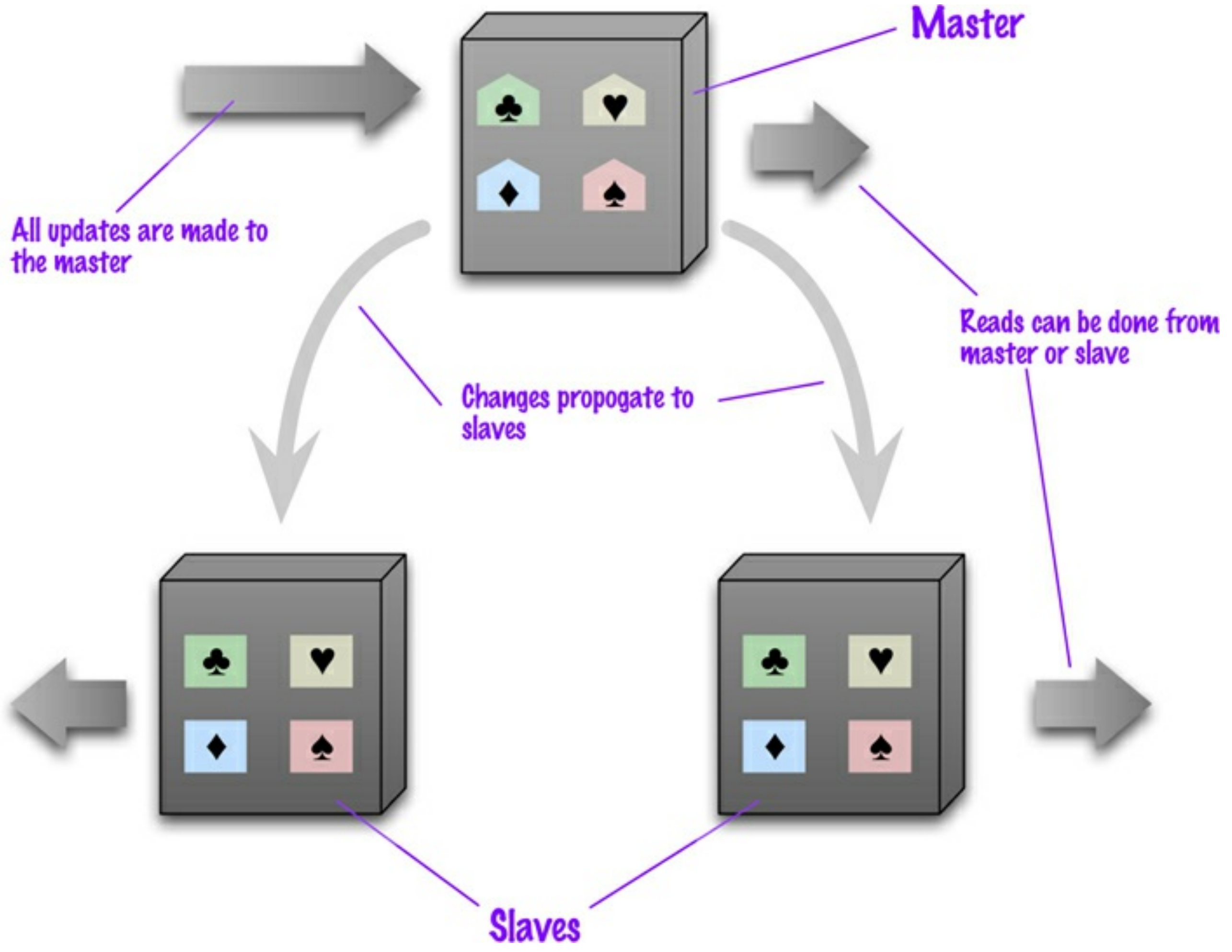
Sharding

- Some databases are intended from the beginning to use sharding, in which case it's wise to run them on a cluster from the very beginning of development, and certainly in production.
- Other databases use sharding as a deliberate step up from a single-server configuration, in which case it's best to start single-server and only use sharding once your load projections clearly indicate that you are running out of headroom.

Master Slave Replication

- With master-slave distribution, you replicate data across multiple nodes.
- One node is designated as the master, or primary. This master is the authoritative source for the data and is usually responsible for processing any updates to that data.
- The other nodes are slaves, or secondaries. A replication process synchronizes the slaves with the master.

Master Slave Replication


Master
All updates are made to the master
Reads can be done from master or slave
Changes propogate to slaves
Slaves

Data is replicated from master to slaves. The master services all writes; reads may come from either master or slaves.

Master Slave Replication

- With master-slave distribution, you replicate data across multiple nodes.
- One node is designated as the master/primary and is responsible for processing any updates to that data.
- The other nodes are slaves/secondaries. A replication process synchronizes the slaves with the master.
- Master-slave replication is helpful for read-intensive dataset.
- You can scale horizontally to handle more read requests by adding more slave nodes and ensuring that all read requests are routed to the slaves.
- It isn't such a good scheme for datasets with heavy write traffic

Master Slave Replication

- Advantage of master-slave replication: ***read resilience***: Should the master fail, the slaves can still handle read requests.
- The failure of the master does eliminate the ability to handle writes until either the master is restored or a new master is appointed.
- Slaves as replicates of the master speeds up recovery after failure - a slave can be appointed a new master very quickly.
- ***Masters can be appointed manually or automatically.***
- Manual appointing – when you configure your cluster, you configure one node as the master.

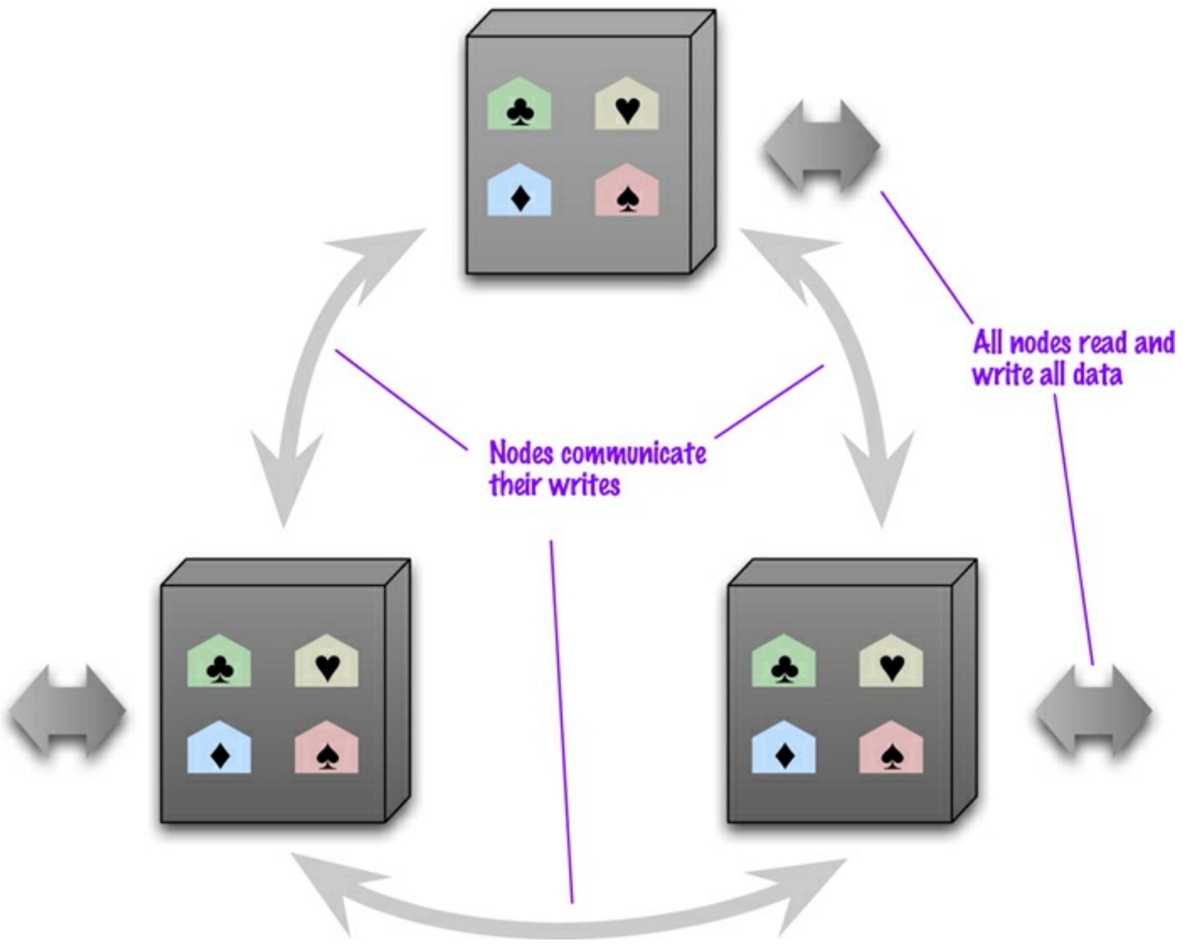
Master Slave Replication

- ***Automatic appointment*** - you create a cluster of nodes and they elect one of themselves to be the master.
- Advantage of automatic appointment: the cluster can automatically appoint a new master when a master fails, reducing downtime.
- ***Replication results into— inconsistency.*** You have the danger that different clients, reading different slaves, will see different values because the changes haven't all propagated to the slaves.
- In the worst case, that can mean that a client cannot read a write it just made.
- If the master fails, any updates not passed on to the backup are lost.

Peer-to-Peer Replication

- Master-slave replication helps with read scalability but doesn't help with scalability of writes.
- The master is a single point of failure.
- Peer-to-peer replication attacks these problems by not having a master. ***All the replicas have equal weight, they can all accept writes, and the loss of any of them doesn't prevent access to the data store.***
- ***Biggest complication: Consistency.*** When you can write to two different places, you run the risk that two people will attempt to update the same record at the same time—***a write-write conflict***.
- Inconsistencies on read lead to problems but at least they are relatively transient. Inconsistent writes are forever.

Peer-to-Peer Replication


All nodes read and write all data
Nodes communicate their writes

Peer-to-peer replication has all nodes applying reads and writes to all the data.

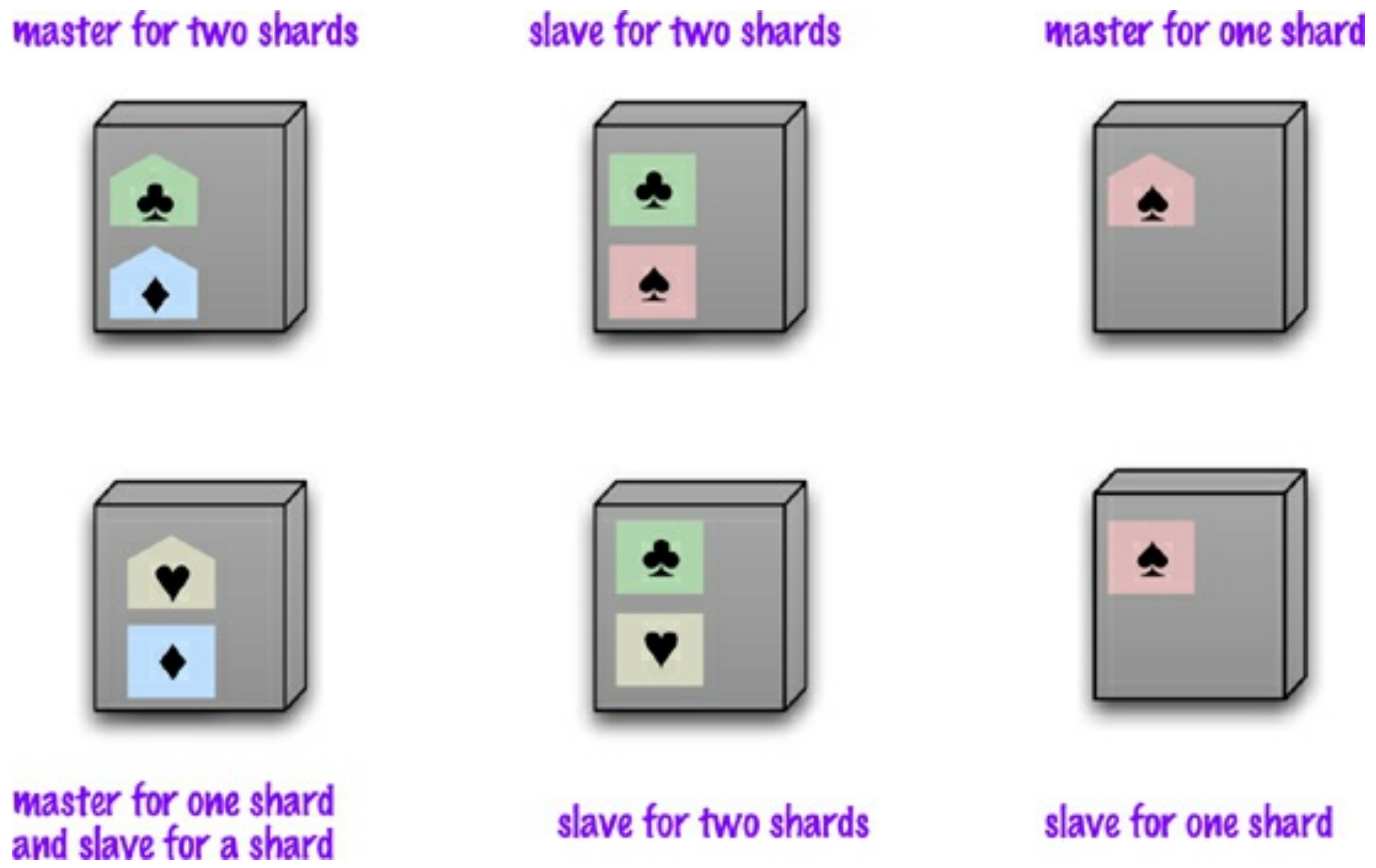
Peer-to-Peer Replication

- At one end, we can ensure that whenever we write data, the replicas coordinate to ensure we avoid a conflict.
- We don't need all the replicas to agree on the write, just a majority, so we can still survive losing a minority of the replica nodes.
- At the end ***we can come up with policy to merge inconsistent writes.***

Combining Sharding and Replication

- Replication and sharding are strategies that can be combined.
- If we use both master-slave replication and sharding, this means that we have multiple masters, but each data item only has a single master.
- Depending on your configuration, you may choose a node to be a master for some data and slaves for others, or you may dedicate nodes for master or slave duties.

Combining Sharding and Replication

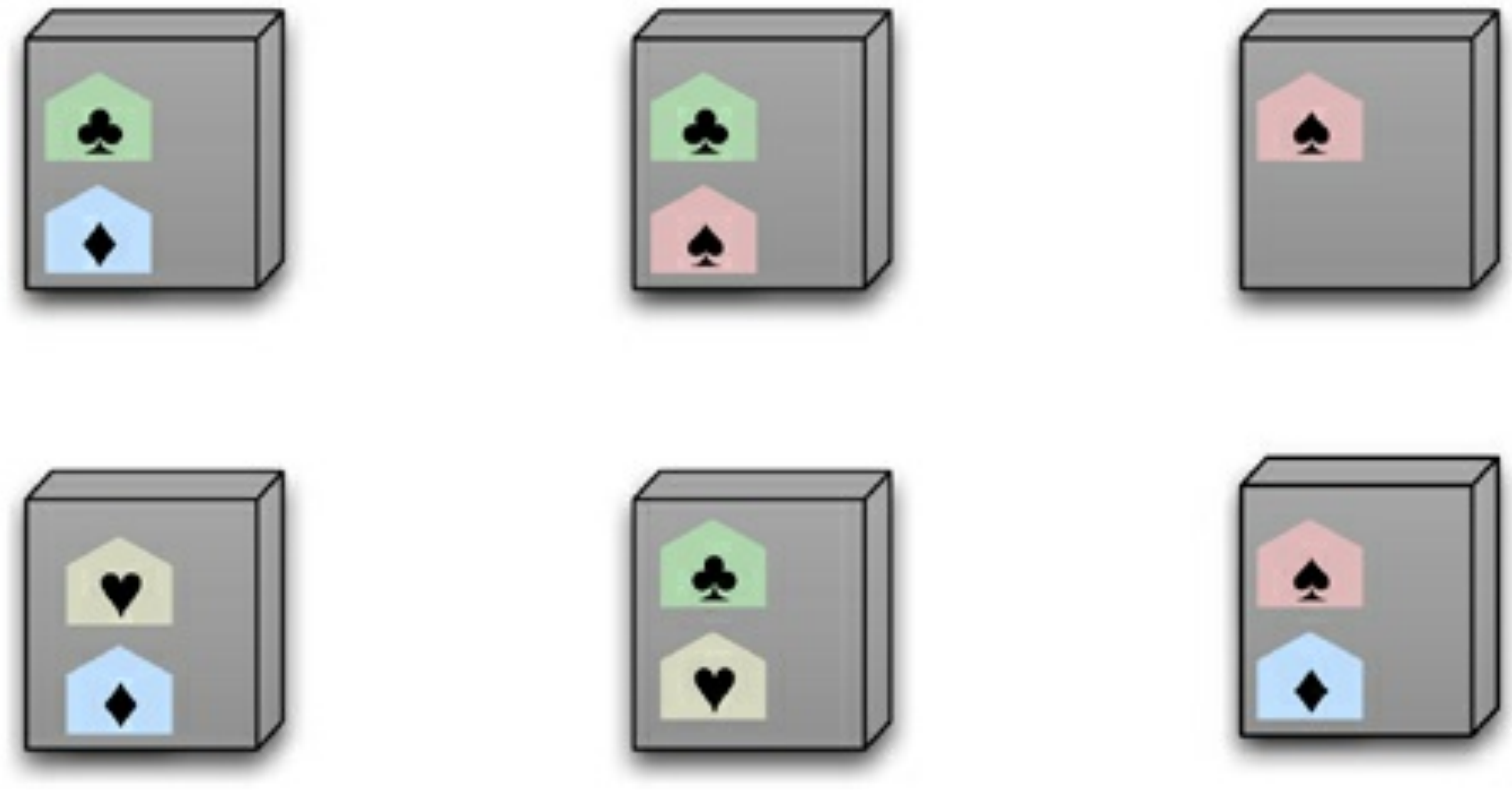


Using master-slave replication together with sharding

Combining Sharding and Replication

- Using peer-to-peer replication and sharding is a common strategy for column-family databases.
- A good starting point for peer-to-peer replication is to have a replication factor of 3, so each shard is present on three nodes.
- Should a node fail, then the shards on that node will be built on the other nodes.

Combining Sharding and Replication



Using peer-to-peer replication together with sharding

THANK YOU