# Schemaless Databases

When you want to store data in a relational database, you first have to define a schema—a defined structure for the database which says what tables exist, which columns exist, and what data types each column can hold. Before you store some data, you have to have the schema defined for it.

With NoSQL databases, storing data is much more casual. A key-value store allows you to store any data you like under a key. A document database effectively does the same thing, since it makes no restrictions on the structure of the documents you store. Column-family databases allow you to store any data under any column you like. Graph databases allow you to freely add new edges and freely add properties to nodes and edges as you wish.

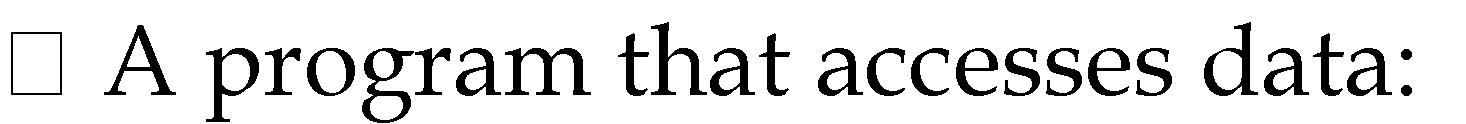
**why schemaless?**

* A schemaless store also makes it easier to deal with nonuniform data
* When starting a new development project you don't need to spend the same amount of time on up-front design of the schema.
* No need to learn SQL or database specific stuff and tools.
* The rigid schema of a relational database (RDBMS) means you have to absolutely follow the schema. It can be harder to push data into the DB as it has to perfectly fit the schema. Being able to add data directly without having to tweak it to match the schema can save you time
* Minor changes to the model and you will have to change both your code and the schema in the DBMS. If no schema, you don't have to make changes in two places. Less time consuming
* With a NoSql DB you have fewer ways to pull the data out
* Less overhead for DB engine
* Less overhead for developers related to scalability
* Eliminates the need for Database administrators or database experts -> fewer people involved and less waiting on experts
* Save time writing complex SQL joins -> more rapid development **Pros and cons of** Schemaless data

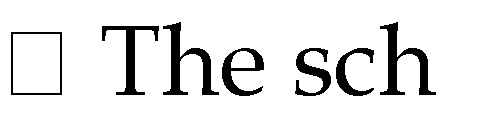
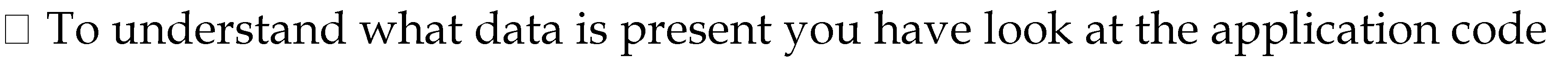
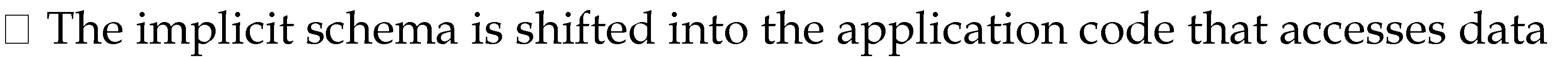
**Pros:**

* More freedom and flexibility
* you can easily change your data organization
* you can deal with nonuniform data

**Cons**:

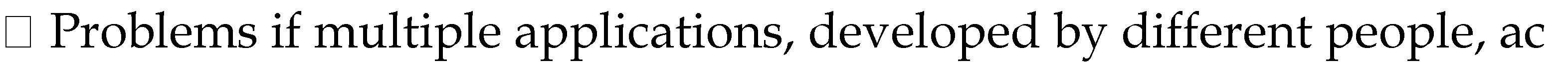


* almost always relies on some form of implicit schema
* it assumes that certain fields are present
* carry data with a certain meaning

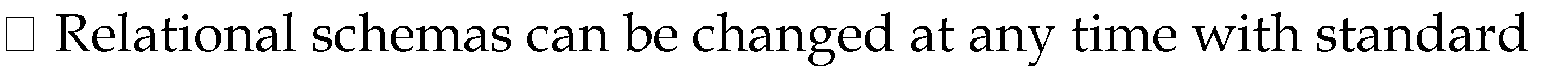


ema cannot be used to:

* decide how to store and retrieve data efficiently  ensure data consistency

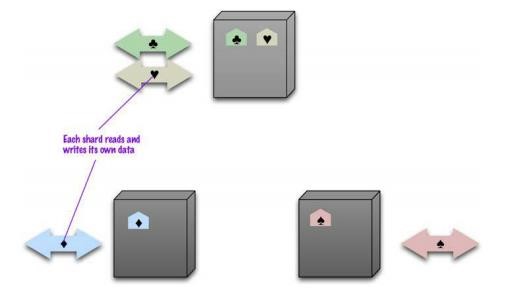
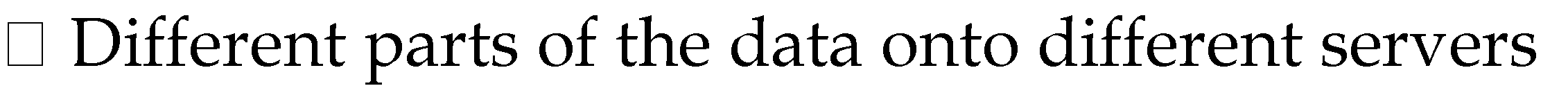
cess the same

database.

SQL commands .

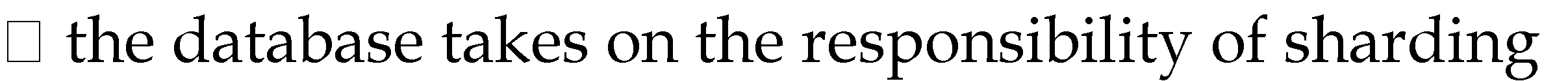
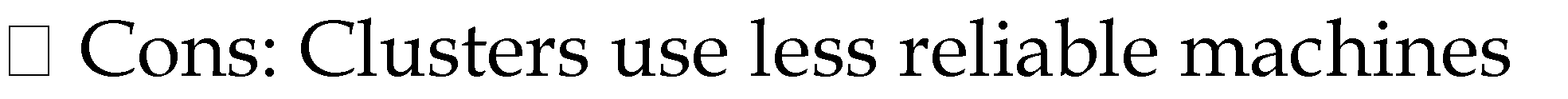
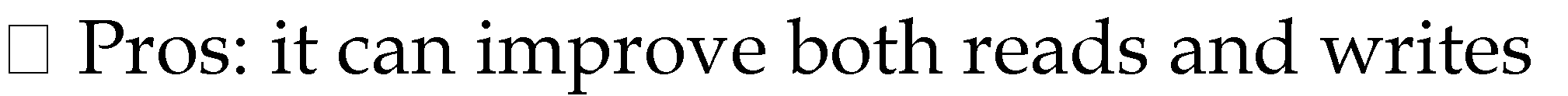
**Sharding:** DB Sharding is nothing but horizontal partitioning of data. Different people are accessing different parts of the dataset. In these circumstances we can support horizontal scalability by putting different parts of the data onto different servers—a technique that’s called sharding.

A table with billions of rows can be partitioned using “Range Partitioning”. If the customer transaction date, for an example, based partitioning will partition the data vertically. So irrespective which instance in a Real Application Clusters access the data, it is “not” horizontally partitioned although Global Enqueue Resources are owning certain blocks in each instance but it can be moving around. But in “db shard” environment, the data is horizontally partitioned. For an example: United States customer can live in one shard and European Union customers can be in another shard and the other countries customers can live in another shard but from an access perspective there is no need to know where the data lives. The DB Shard can go to the appropriate shard to pick up the data.



* Horizontal scalability
* Ideal case: different users all talking to different server nodes  Data accessed together on the same node aggregate unit!̶

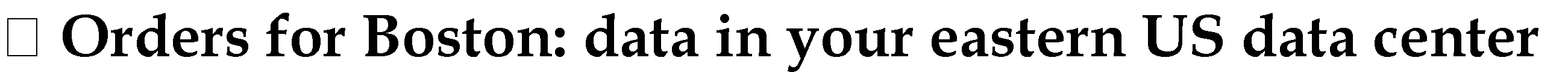
resilience decreases̶ Many NoSQL databases offer auto-sharding



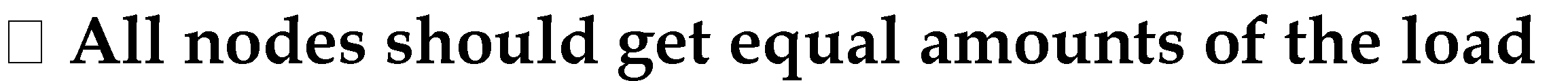
**Improving performance:**

Main rules of sharding:

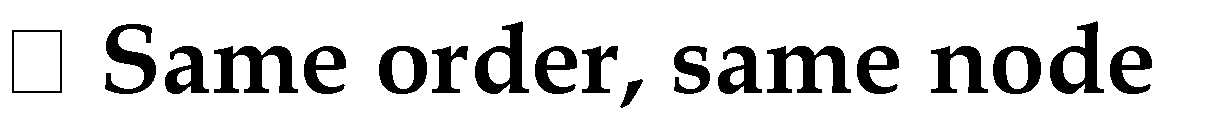
## 1. Place the data close to where it’s accessed



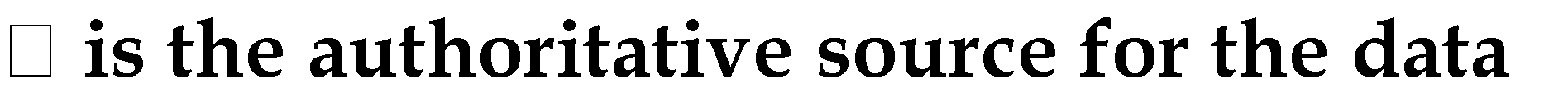
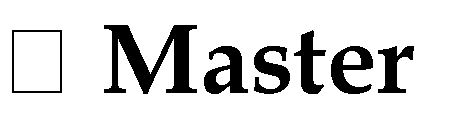
## 2. Try to keep the load even



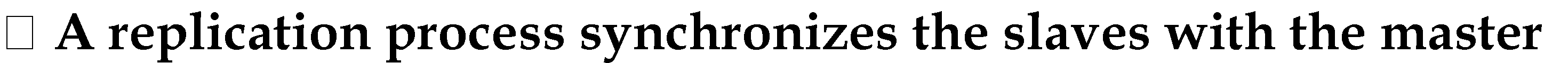
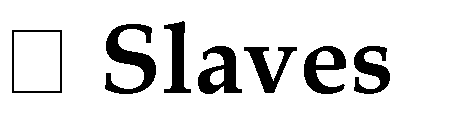
## 3. Put together aggregates that may be read in sequence

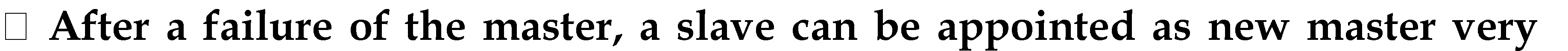


# Master-Slave Replication

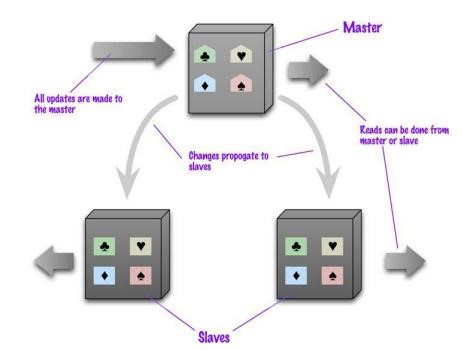


# es to that data ically

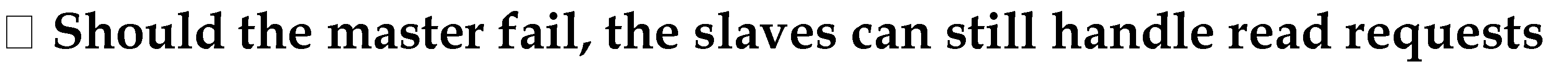
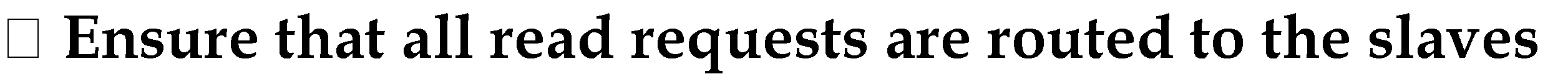
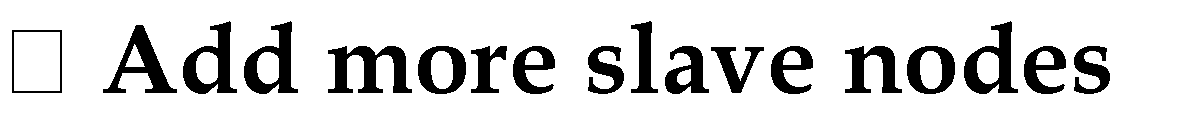
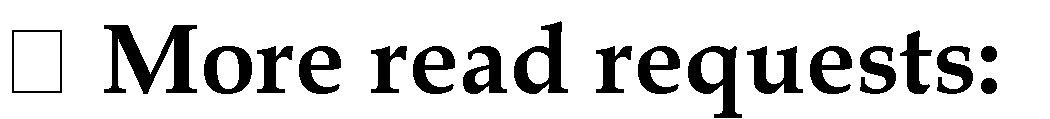
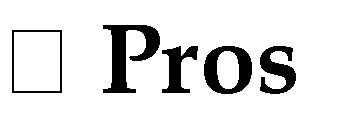


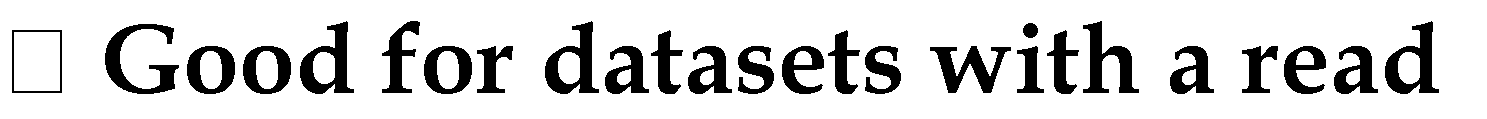


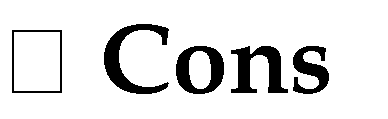
# quickly



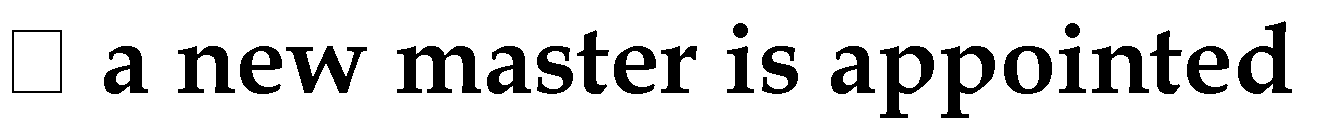
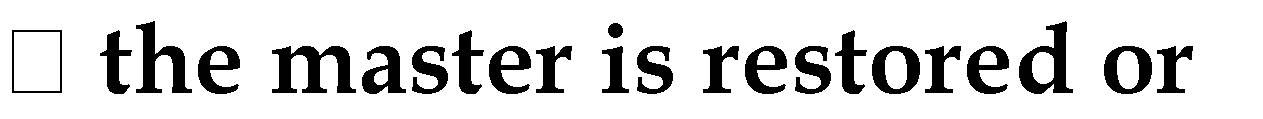
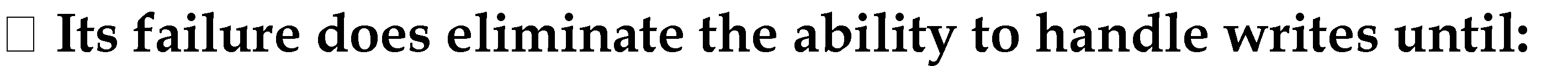
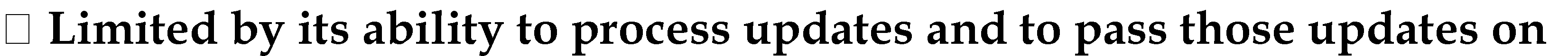
# Pros and cons of Master-Slave Replication

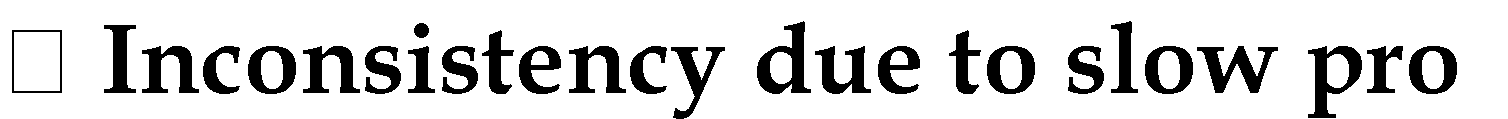


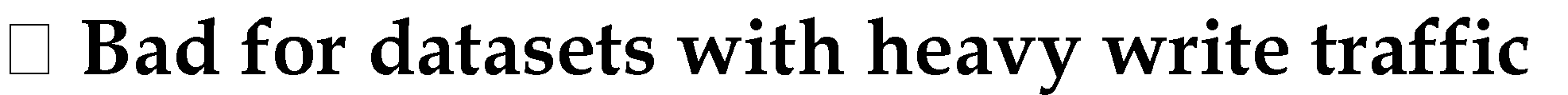
**-intensive dataset**



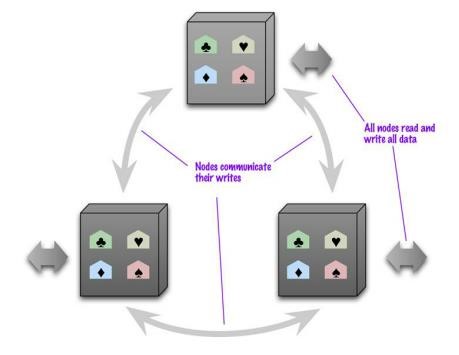
# ter is a bottleneck



**pagation of changes to the slaves**

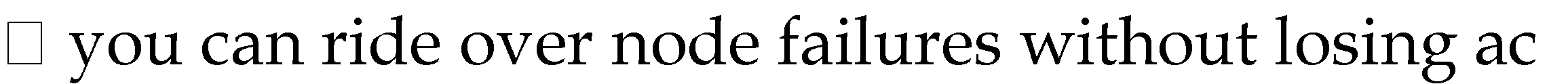
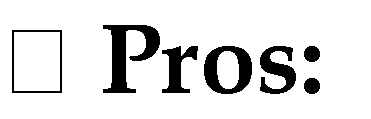


# Peer-to-Peer Replication

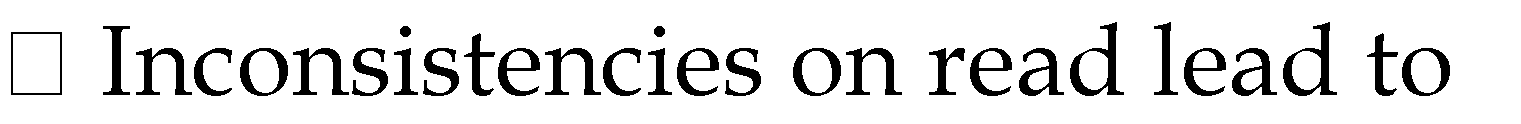
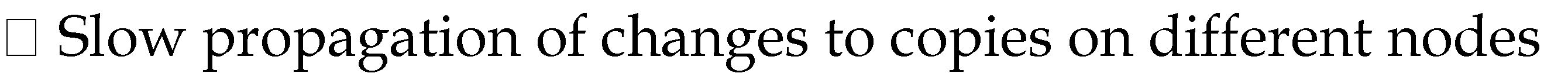
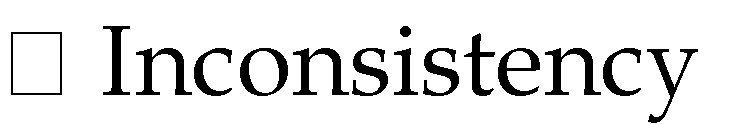
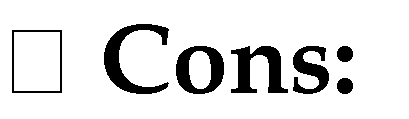
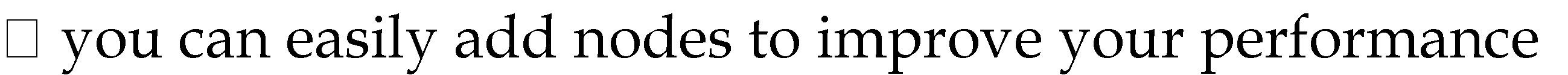


 All the replicas have equal weight, they can all accept writes  The loss of any of them doesn’t prevent access to the data store.

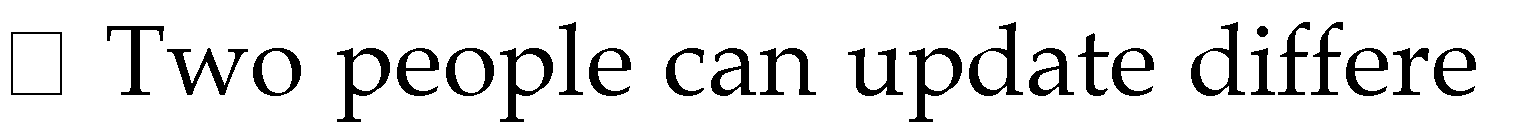
# Pros and cons of peer-to-peer replication



cess to data

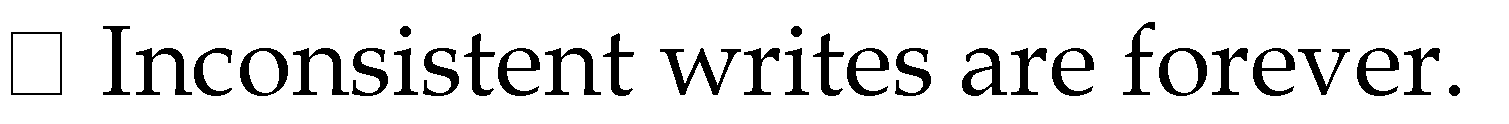


problems but are relatively transient



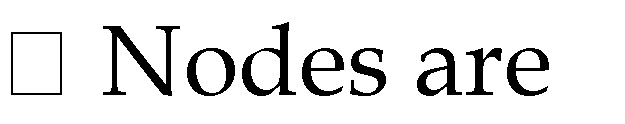
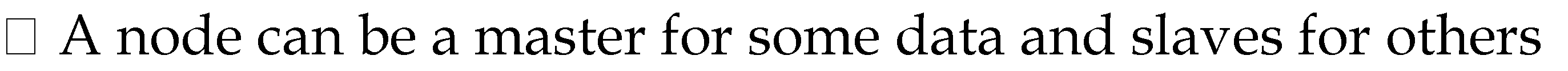
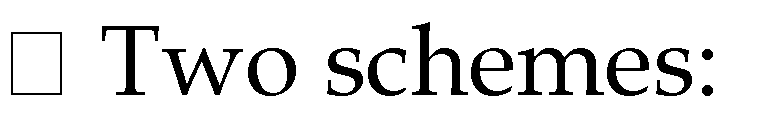
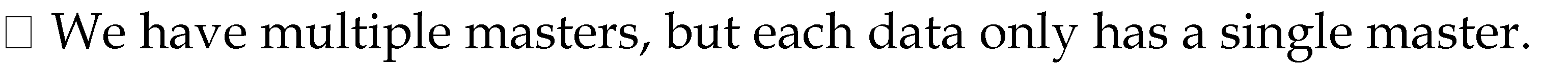
nt copies of the same record stored on different

nodes at the same time - a write-write conflict.

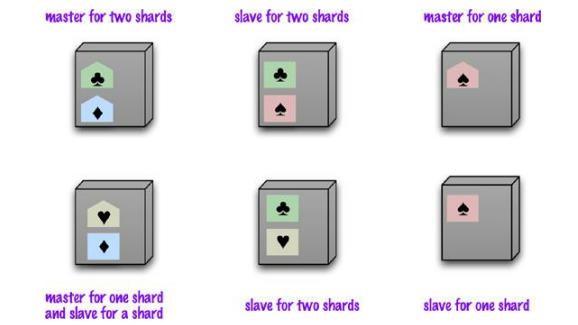


# Sharding and Replication on Master-Slave

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.

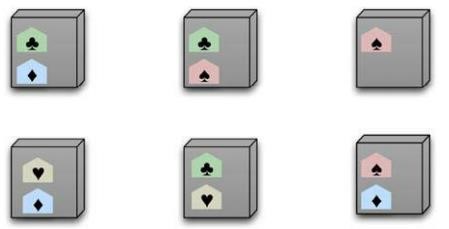


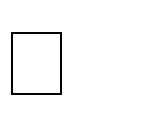
dedicated for master or slave duties

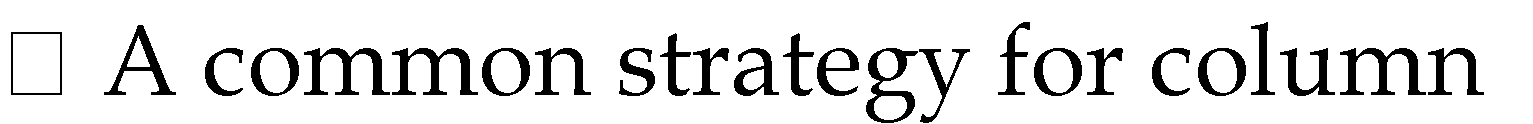


# Sharding and Replication on P2P

Using peer-to-peer replication and sharding is a common strategy for column family databases. In a scenario like this you might have tens or hundreds of nodes in a cluster with data sharded over them. 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. (See following figure)



Usually each shard is present on three nodes

-family databases

# Key Points

* There are two styles of distributing data:
* Sharding distributes different data across multiple servers, so each server acts as the single source for a subset of data.
* Replication copies data across multiple servers, so each bit of data can be found in multiple places.

A system may use either or both techniques.

* Replication comes in two forms:
* Master-slave replication makes one node the authoritative copy that handles writes while slaves synchronize with the master and may handle reads.
* Peer-to-peer replication allows writes to any node; the nodes coordinate to synchronize their copies of the data.

Master-slave replication reduces the chance of update conflicts but peer to-peer replication avoids loading all writes onto a single point of failure.