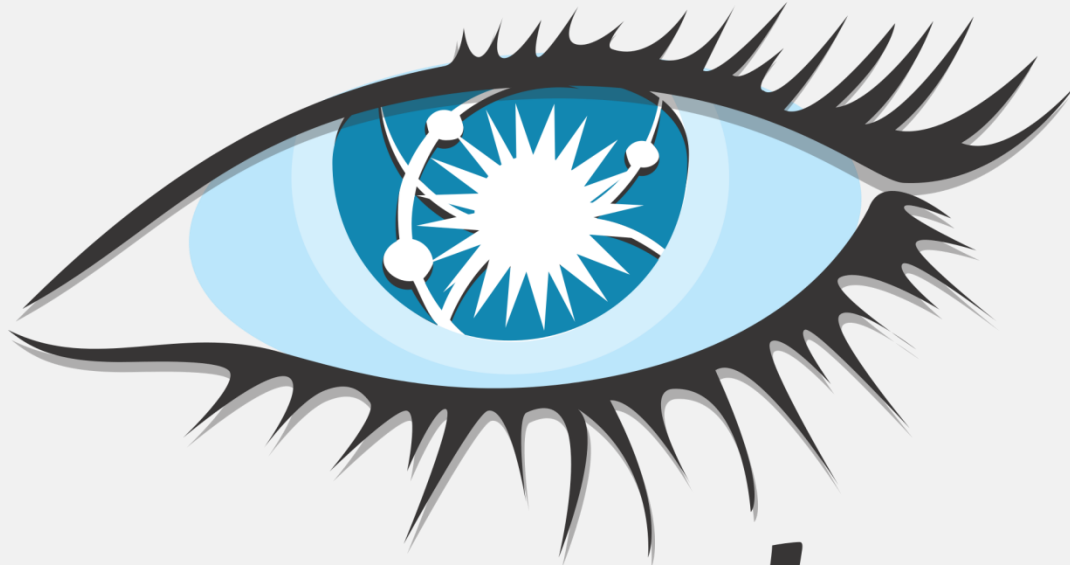


Лекция 11

Cassandra
MongoDB
Redis

Cassandra



cassandra



- Разработана в Facebook в 2008 году
- Следует модели Google BigTable
- Использует модель *Eventual Consistency*
- Проект Apache: <http://cassandra.apache.org/>
- Использует Apache Thrift для API
- Большой объем данных
- Устойчивость к нагрузкам и сбоям
- Легкая масштабируемость и репликация

Data Model



KeySpace

ColumnFamily

Row

Key

Column:

- name
- value
- timestamp



- **Column:** наименьший элемент данных, пара имя и значение
- **ColumnFamily:** структура для группировки *Columns*
- **Key:** постоянное имя записи
- **Keyspace:** самый внешний уровень организации данных (имя базы данных)

Super Column



Keyspace

ColumnFamily

Row key	Name 1	...	Name q
	Value 1		Value q
	time stamp		...

ColumnFamily 2

Row key	Name a	...	Name x
	Value a		Value x

ColumnFamily k

Row key	Name s	...	Name z
	Value s		Value z

SuperColumnFamily 1

Row key	Super column 1			
	Name 1	...	Name 5	
	Value 1		Value 5	
	time stamp		...	
	...			
	Super column 16			
	Name 1	...	Name 6	
	Value 1		Value 6	
	
	

SuperColumnFamily m

Row key	Super column 1			
	Name 1	...	Name 4	
	Value 1		Value 4	
	time stamp		...	
	...			
	Super column 9			
	Name 1	...	Name w	
	Value 1		Value w	
	
	

Column: **key -> string**
SuperColumn: **key -> columns**



Basic API

- *get(key)*
- *put(key, value)*
- *delete(key)*
- *execute(key, operation, parameters)*

Cassandra Query Language (CQL)



```
CREATE KEYSPACE MyKeySpace
WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : 3 };

USE MyKeySpace;

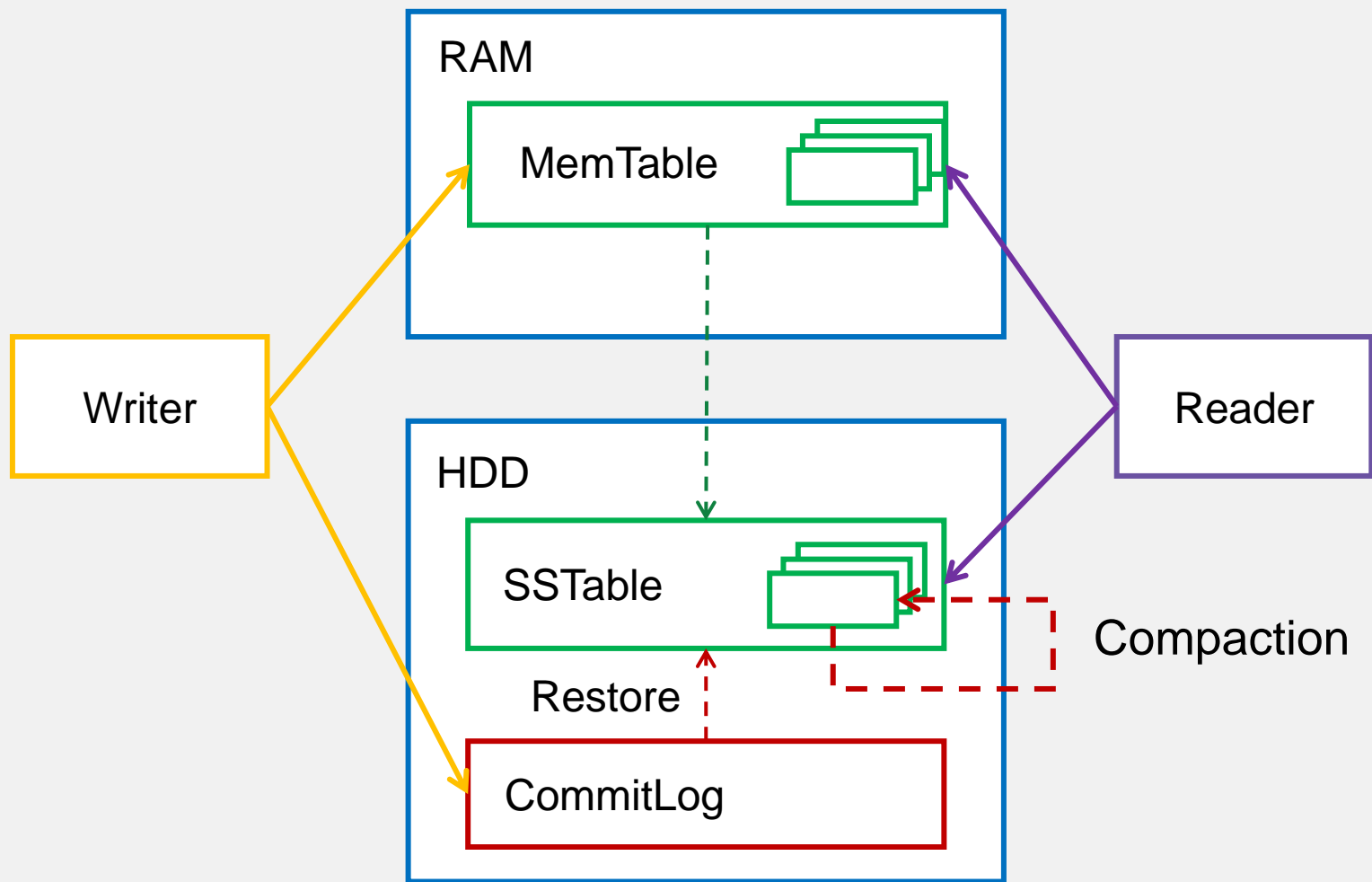
CREATE COLUMNFAMILY MyColumns (id text, Last text, First text, PRIMARY
KEY(id));

INSERT INTO MyColumns (id, Last, First) VALUES ('1', 'Doe', 'John');

SELECT * FROM MyColumns;
```

```
id   | first | last
----+-----+-----
1    | John  | Doe
(1 rows)
```


Запись и чтение данных



Cassandra и Consistency



- Eventual consistency
- Cassandra имеет программируемый *read/writable consistency*

Read

- **One:** возвращается ответ от самой первой ноды, которая отвечает
- **Quorum:** Запрос ко всем нодам и ответ от ноды, которая имеет самый последний таймстемп, когда большинство ответили
- **All:** Запрос ко всем нодам и ответ от ноды, которая имеет самый последний таймстемп, когда все ответили

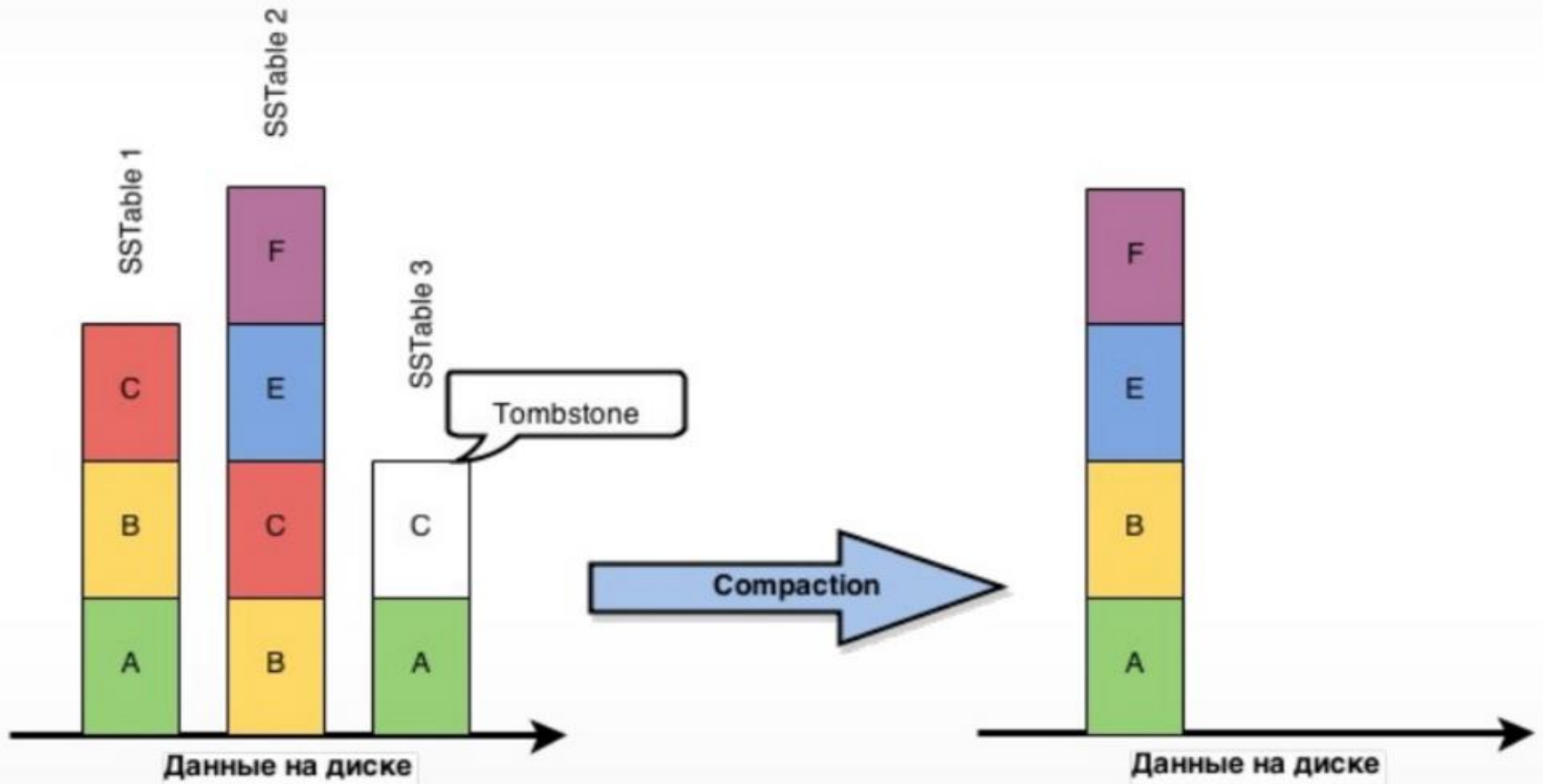
Cassandra и Consistency



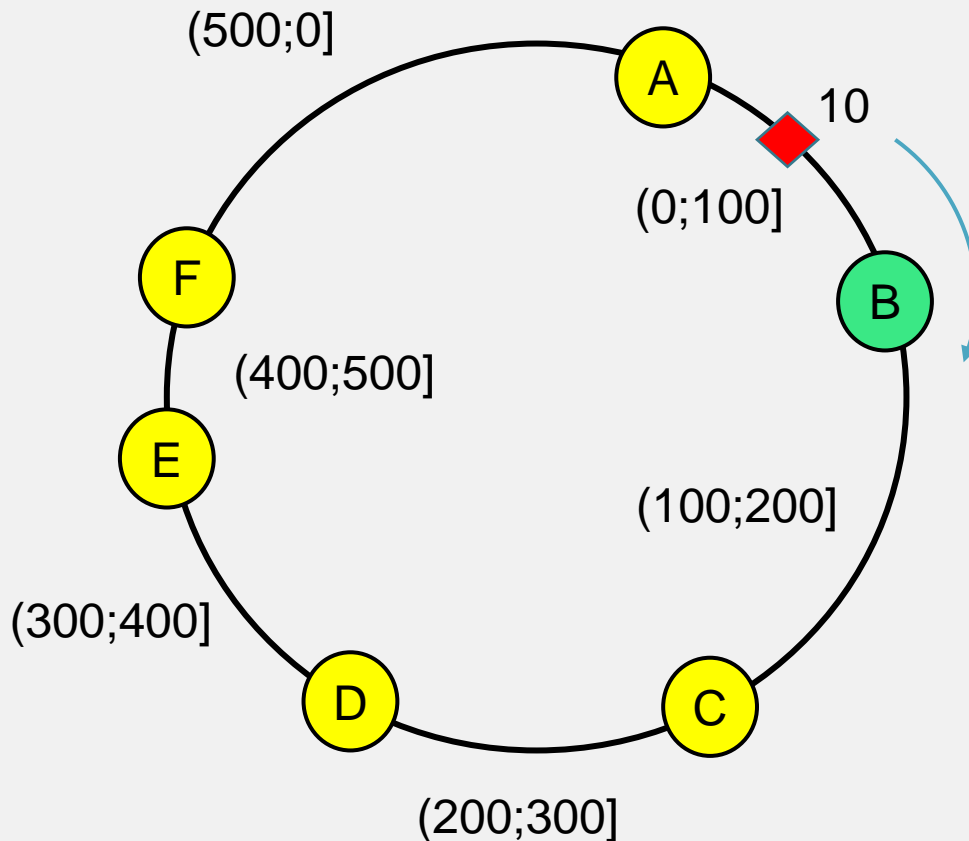
Write

- **Zero:** Ничего не гарантируется. Асинхронная запись в *background*
- **Any:** Гарантируется запись на, как минимум, одну ноду
- **One:** Гарантируется запись, как минимум, в один *commit log* и в *memory table*
- **Quorum:** Гарантируется, что запись будет выполнена на $N/2 + 1$ нод
- **All:** Гарантируется, что запись дойдет до всех нод

Compaction

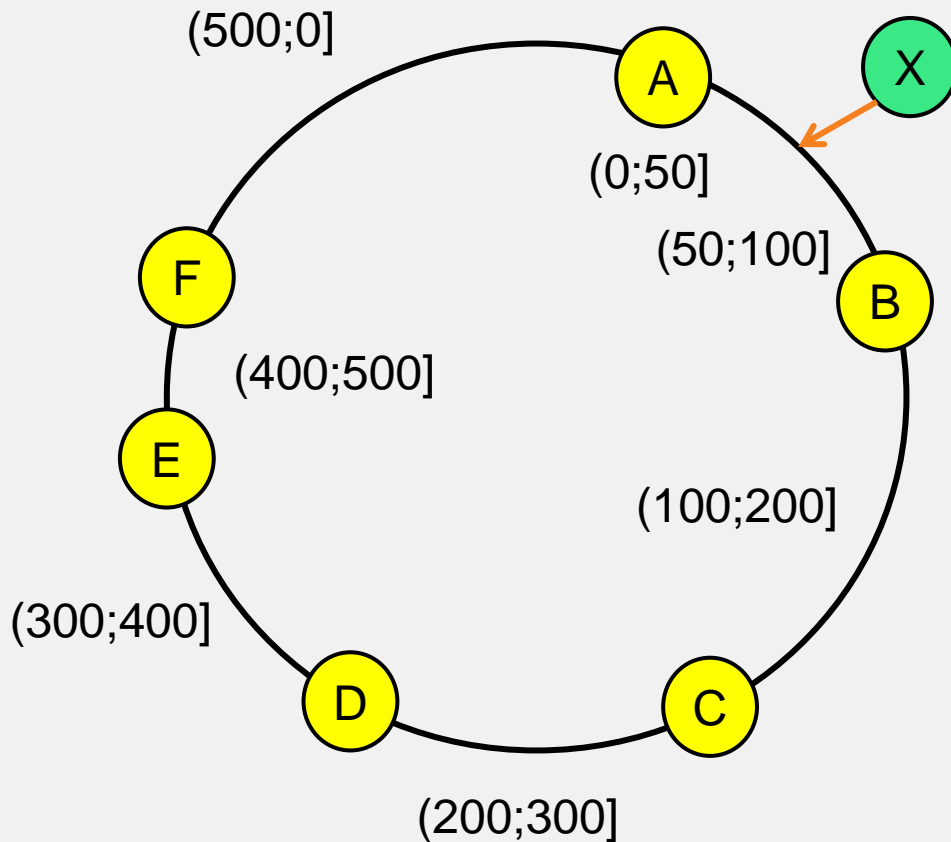


Распределение данных



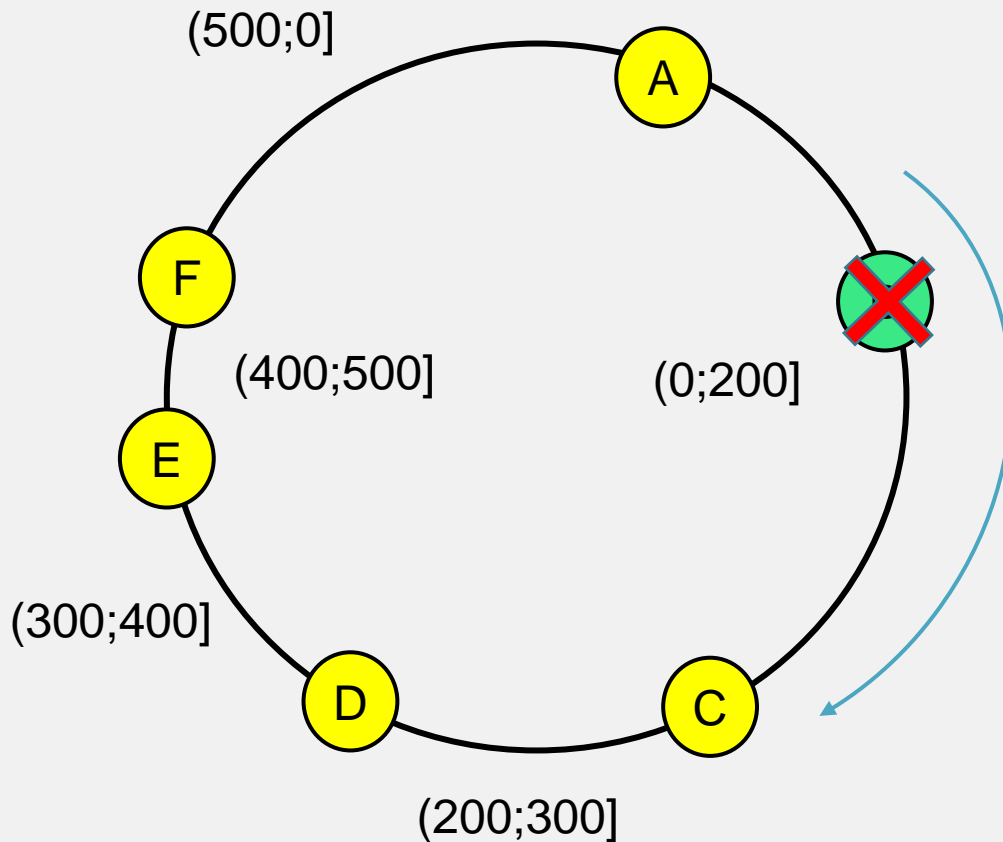
- Сервера расположены в кольце
- Каждый сервер отвечает за определенный диапазон ключей
- Обход кольца происходит по часовой стрелке

Добавление ноды



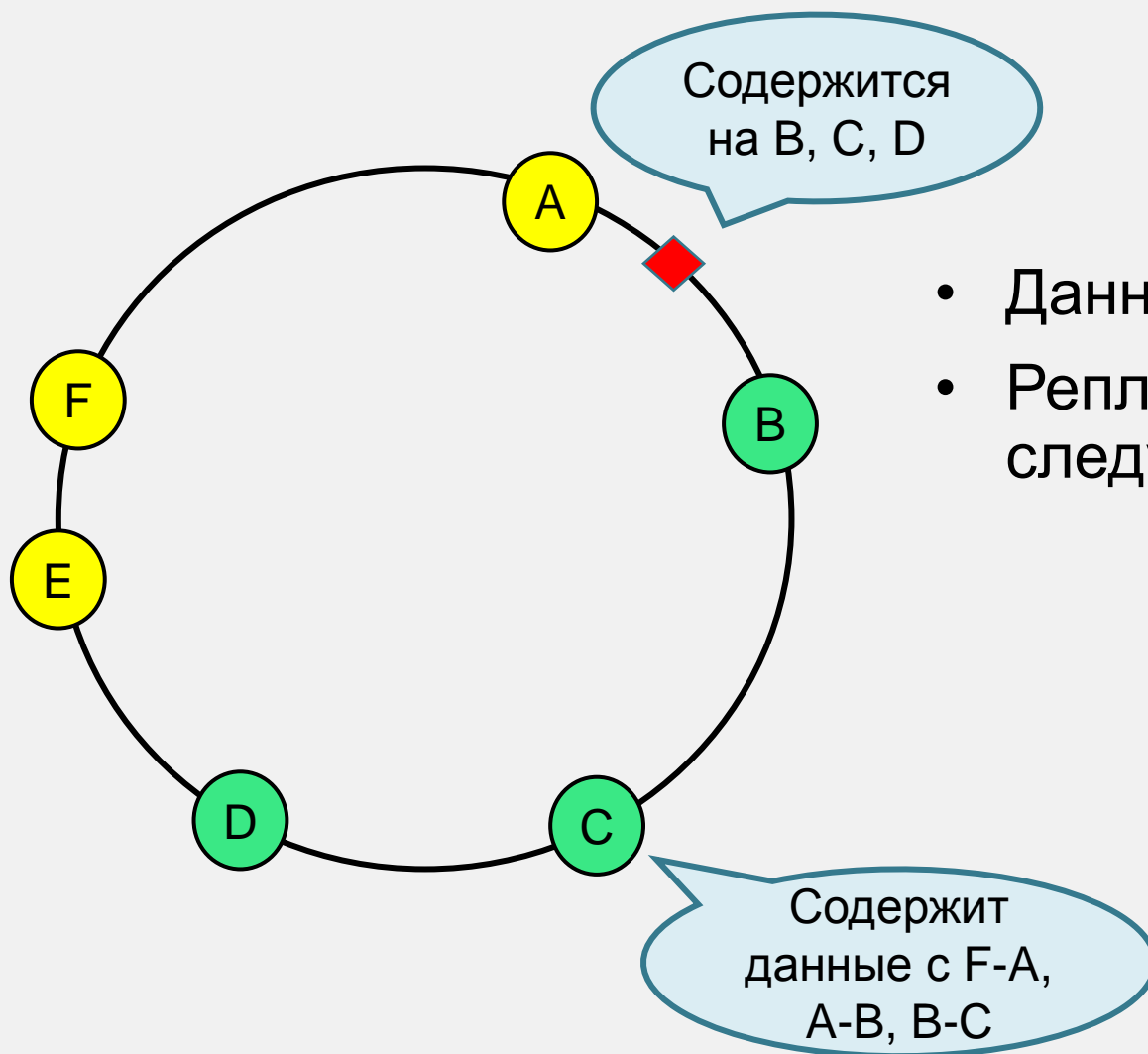
- При добавлении ноды текущий диапазон ключей делится пополам
- Нужные данные копируются с ноды B на X

Обработка падений



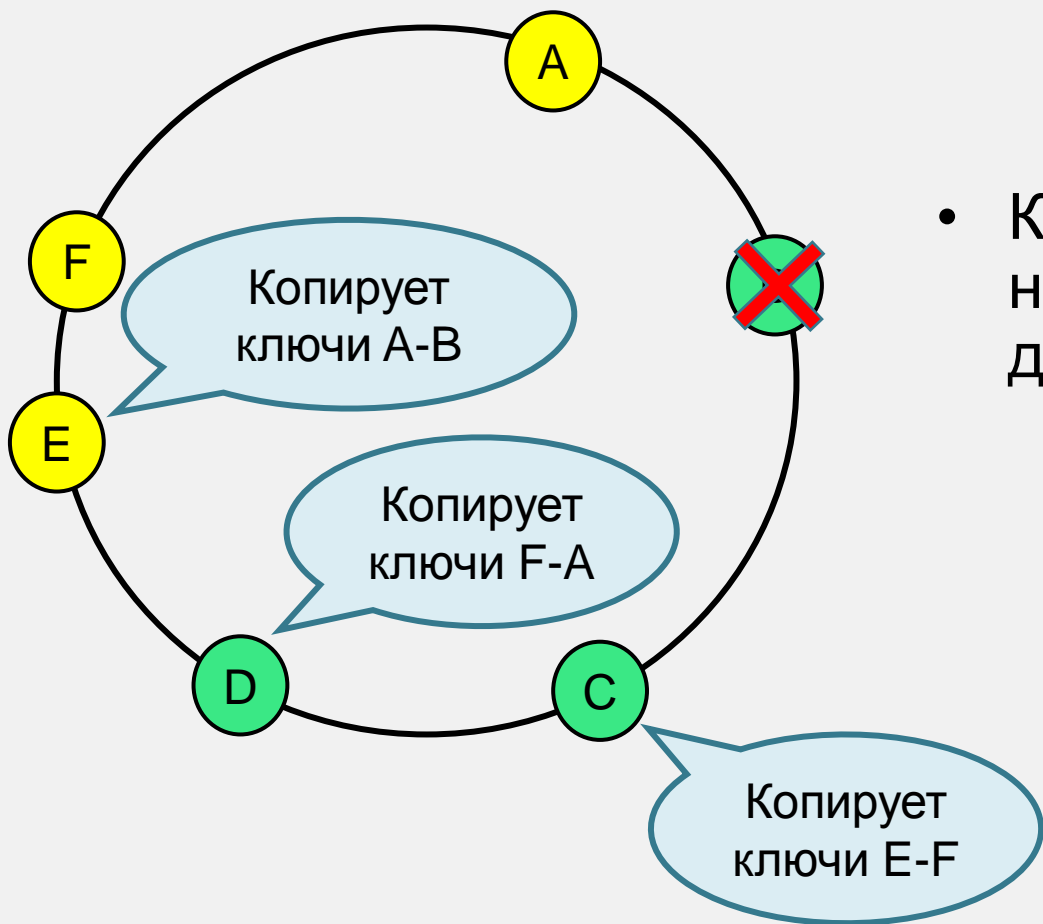
- При падении ноды, следующая за ней начинает отвечать за ее диапазон данных

Репликация данных



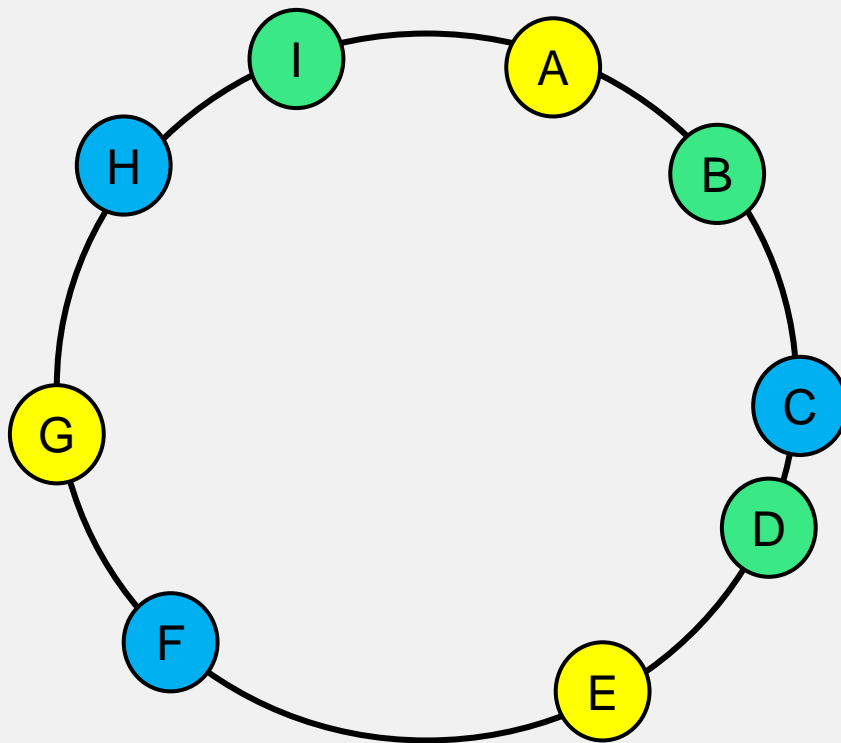
- Данные реплицируются
- Реплики хранятся на следующих серверах




Восстановление репликаций



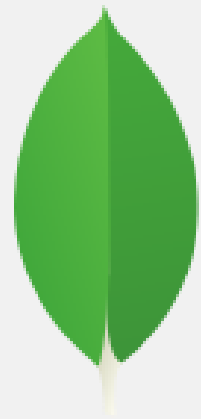
- Копируются необходимый минимум данных

Виртуальные ноды



-  Сервер 1: A, E, G
-  Сервер 2: B, D, I
-  Сервер 3: C, F, H





mongoDB®



- Разрабатывалась с 2007 года компанией 10gen (MongoDB Inc)
- Открытая кросс-платформенная документная база данных
- NoSQL DataBase:
 - Нет транзакций
 - Отсутствует изоляция
- Масштабируемость и отказоустойчивость
- Может работать в соответствии с парадигмой MapReduce



SQL	MongoDB
База данных	База данных
Таблица	Коллекция
Строка	Документ
Колонка	Поле
Индекс	Индекс
Join	-



Документ – JSON-подобный объект:

```
{  lesson: 1
  name: "Introduction"
  type: "lecture"
}
{  lesson: 2,
  name: "MapReduce",
  type: "Practice"
}
```

Модель данных



Не имеет фиксированной структуры:

```
{
    type: "good",
    name: "Television",
    price: 30000,
    features: {
        lcd: 1
        led: 0
    },
    categories: ["home", "tv"]
}

{
    type: "page",
    content: "Text"

    type: "news",
    date: "Text",
    header: "Title"
    teaser: "Snippet"
    content: "Text of
news"
}
```




Relational

Customer ID	First Name	Last Name	City
0	John	Doe	New York
1	Mark	Smith	San Francisco
2	Jay	White	Dallas
3	Meagan	White	London
4	Edward	Daniels	Boston

Phone Number	Type	DNC	Customer ID
1-212-555-1212	home	T	0
1-212-555-1213	home	T	0
1-212-555-1214	cell	F	0
1-212-777-1212	home	T	1
1-212-777-1213	cell	(null)	1
1-212-888-1212	home	F	2



MongoDB

```
{  vers: 1,
  customer_id : 1,
  name : {
    "f": "Mark",
    "l": "Smith"  },
  city : "San Francisco",
  phones: [ {
    number : "1-212-777-1212",
    dnc : true,
    type : "home"
  },
  {
    number : "1-212-777-1213",
    type : "cell"
  }
]
```



BSON – Binary JavaScript Object Notation

Типы:

- **string** – строка
- **int** – целое число
- **double** – число с плавающей точкой
- **DateTime** – дата
- **byte[]** – массив байт
- **bool** – булевый (True / False)
- **null**
- **BsonObject**
- **BsonObject[]**

Ключ объекта



- Присваивается автоматически:
_id: ObjectID("6kd9287vemd7hr02kd82s1q1")
- Задается пользователем при вставке

Create



SQL:

```
CREATE DATABASE vldc;
```

```
CREATE TABLE vldc.users('id' INT AUTO_INCREMENT  
PRIMARY KEY, 'author' VARCHAR(50));
```

```
INSERT INTO vldc.users SET first_name="Alex";
```

MongoDB:

```
use vldc
```

```
db.users.insert({first_name: "Alex"})
```

Select



Найти все документы:

```
db.users.find()
```

Найти только один документ:

```
db.users.findOne()
```

Все, у которых автор Alex:

```
db.users.find({author: "Alex"})
```

При этом выбрать только имя:

```
db.users.find({author: "Alex"}, {name: 1})
```

Функции агрегации `limit`, `count`, `sort`



Подсчитать количество:

```
db.users.find({author:"Alex"}).count()
```

Пропустить первые 2, вернуть следующие 3:

```
db.users.find().skip(2).limit(3)
```

При этом отсортировать по имени:

```
db.users.find().sort({name:1}).skip(2).limit(3)
```

Update



SQL:

```
UPDATE users SET title="Hadoop" WHERE author="Alex"
```

MongoDB (атомарная операция “нашел и обновил”):

```
db.users.update( {author="Alex"}, { $set: {title="Hadoop"} } )
```

```
> db.test.insert({'user':1}, true)
> doc = db.test.findOne()
{ "_id" : ObjectId("4e30aed2b2b4bfd1cbe90cc3"), "user" : 1 }
> doc.user = 2
> db.test.save(doc)
> db.test.find()
{ "_id" : ObjectId("4e30aed2b2b4bfd1cbe90cc3"), "user" : 2 }
```

Update операторы



```
{ $set : { x : 1 , y : 2 } }
{ $inc : { field : value } }
{ $unset : { field : 1 } }
{ $push : { field : value } }
{ $pushAll : { field : value_array } }
{ $addToSet : { field : value } }
{ $addToSet : { a : { $each : [ 3 , 5 , 6 ] } } }
{ $pop : { field : 1 } }
{ $pull : { field : _value } }
{ $pull : { field : { $gt : 3 } } } удаляем элементы больше 3
{ $pullAll : { field : value_array } }
{ $rename : { old_field_name : new_field_name } }
{ $bit : { field : { and : 5 } } }
```


Remove



SQL:

```
DELETE FROM users WHERE id=1
```

```
DELETE FROM users WHERE author="Alex"
```

MongoDB:

```
db.users.remove( { _id:ObjectID("6kd9287vemd7hr02kds1q1") } )
```

```
db.users.remove( { author:"Alex" } )
```

```
db.users.remove( { } ) – удаляем все
```

Атомарное удаление:

```
db.videos.remove( { rating: { $lt : 3.0 }, $atomic: true } )
```

Создание индексов



SQL:

```
ALTER TABLE 'users' ADD INDEX ('author')
```

MongoDB:

```
db.users.ensureIndex( {author:1} )    – по возрастанию
```

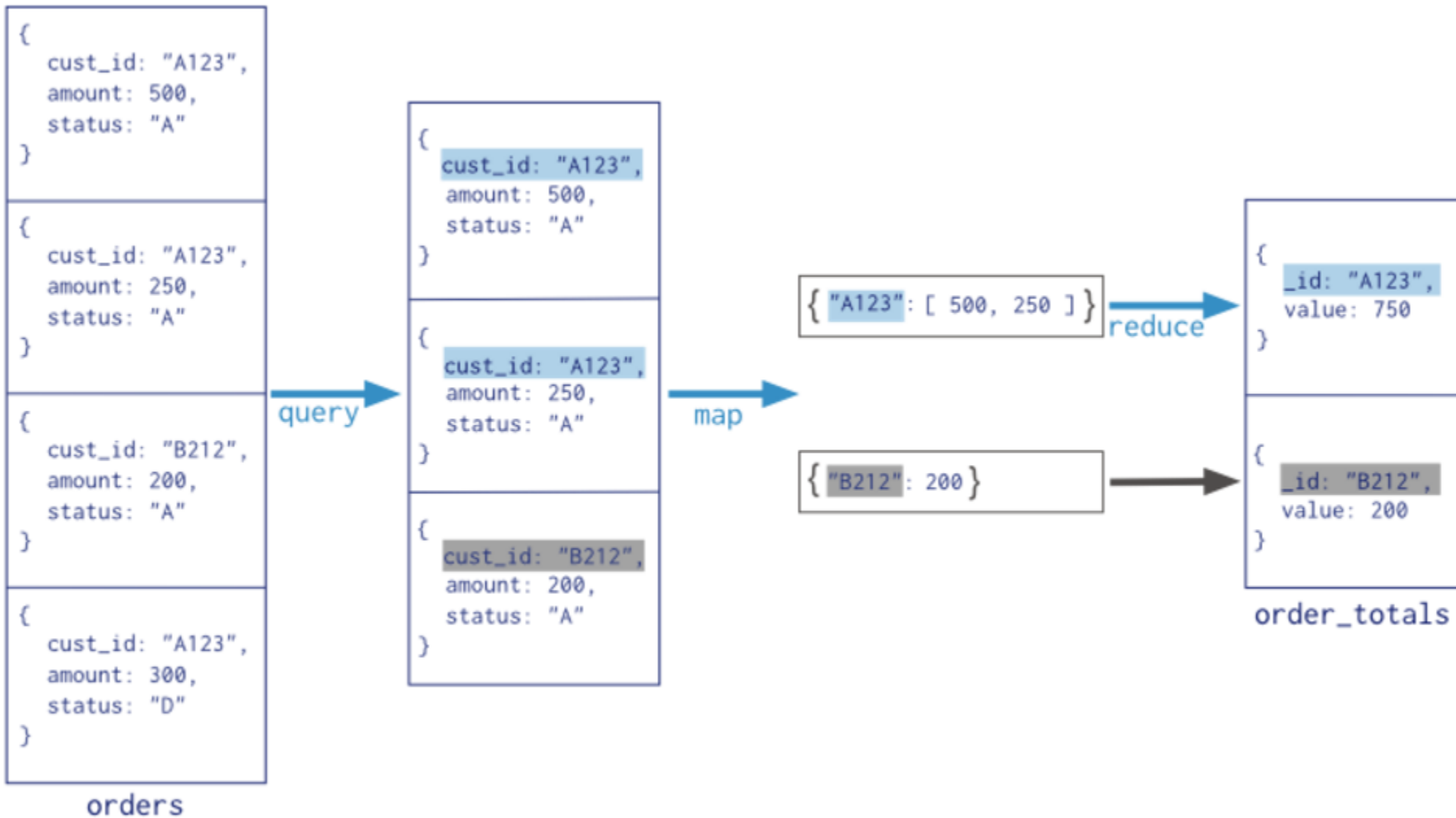
```
db.users.ensureIndex( {author:-1} )   – по убыванию
```

MapReduce



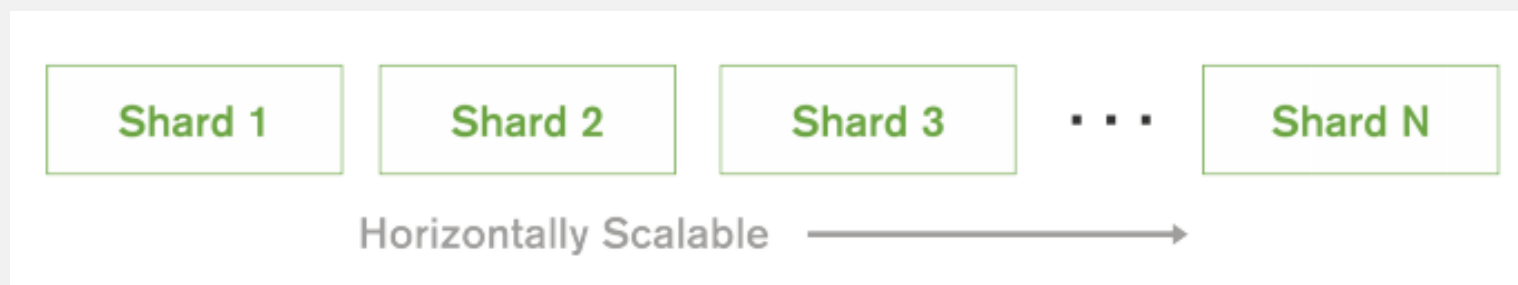
Collection
↓
db.orders.mapReduce(
 map → function() { emit(this.cust_id, this.amount); },
 reduce → function(key, values) { return Array.sum(values) },
 {
 query → { status: "A" },
 output → "order_totals"
 }
)

MapReduce





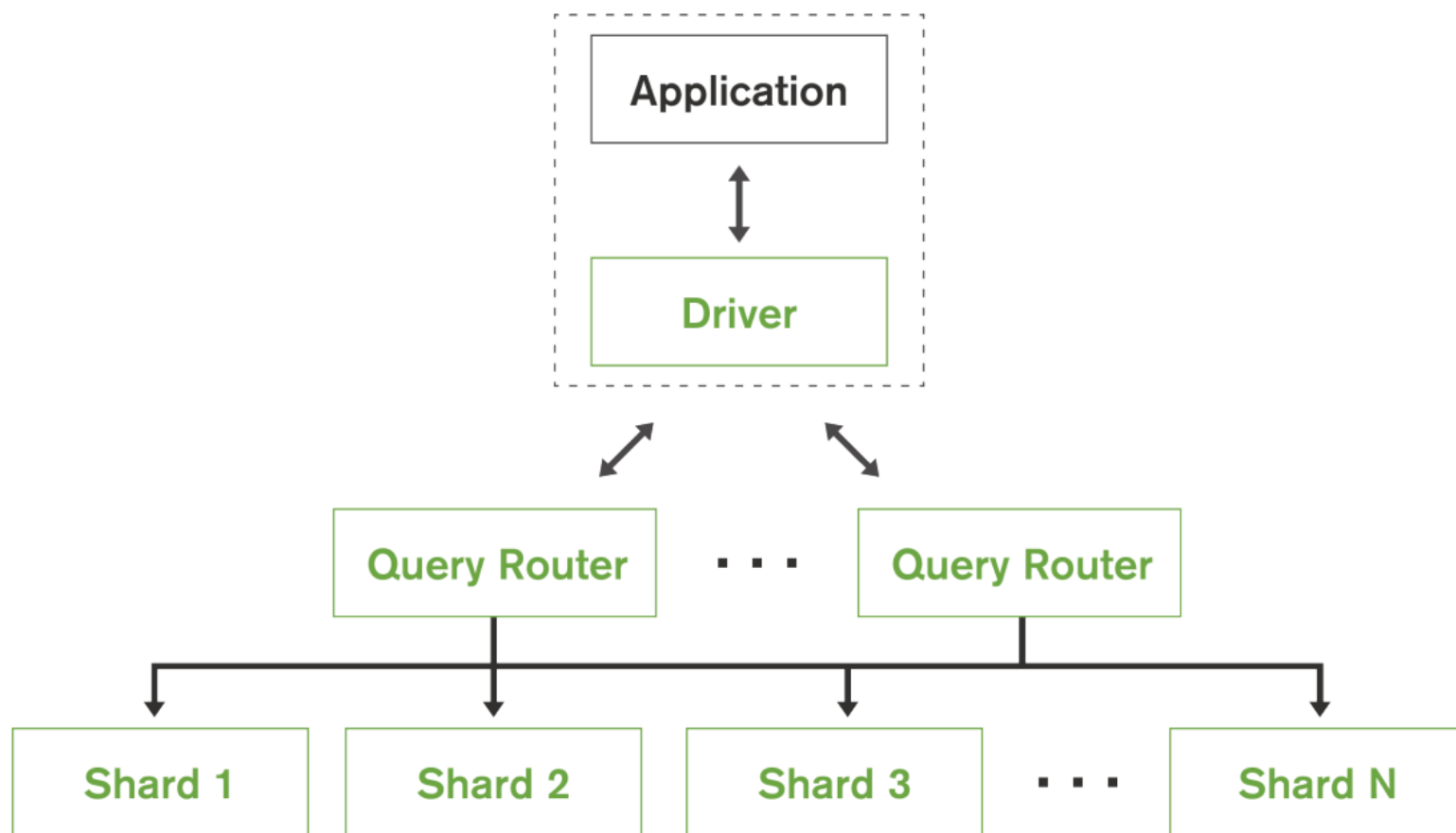
Горизонтальное масштабирование через sharding.



Несколько типов разбиения ключей:

- Range-based (близкие ключи лежат рядом)
- Hash-based (равномерное распределение по кластеру)
- Location-aware (пользовательская логика)

Обработка запроса. Query Router



Horizontally Scalable →

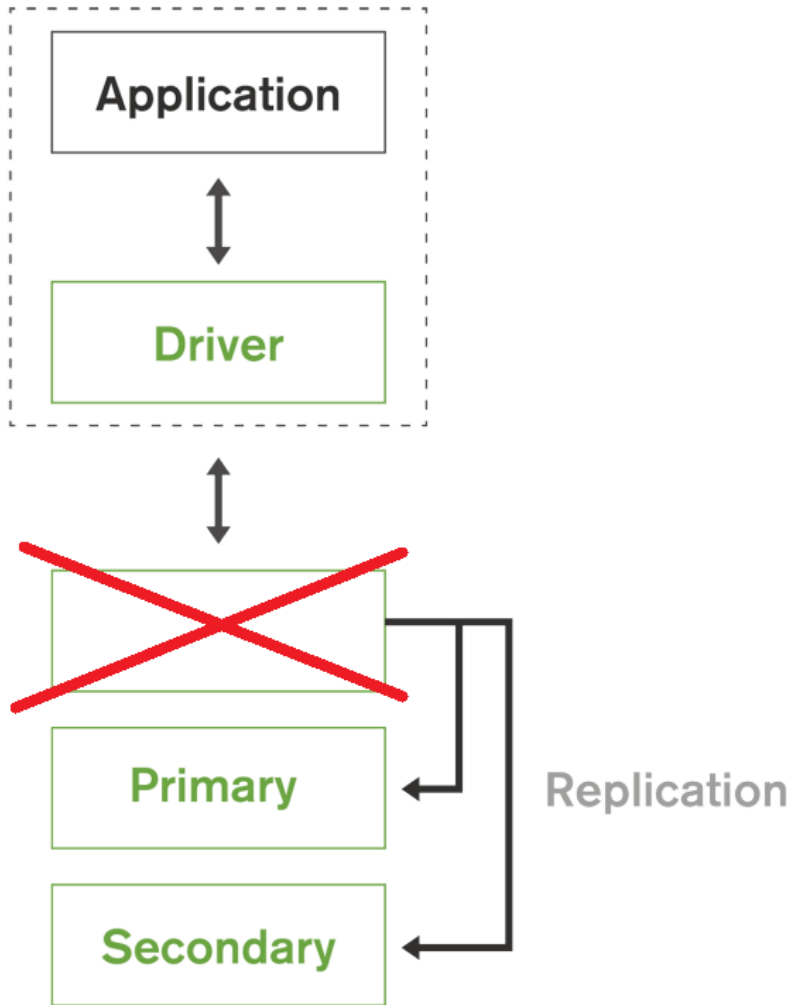


MongoDB поддерживает консистентность на уровне документа

Пользователь может задать уровень консистентности при записи данных:

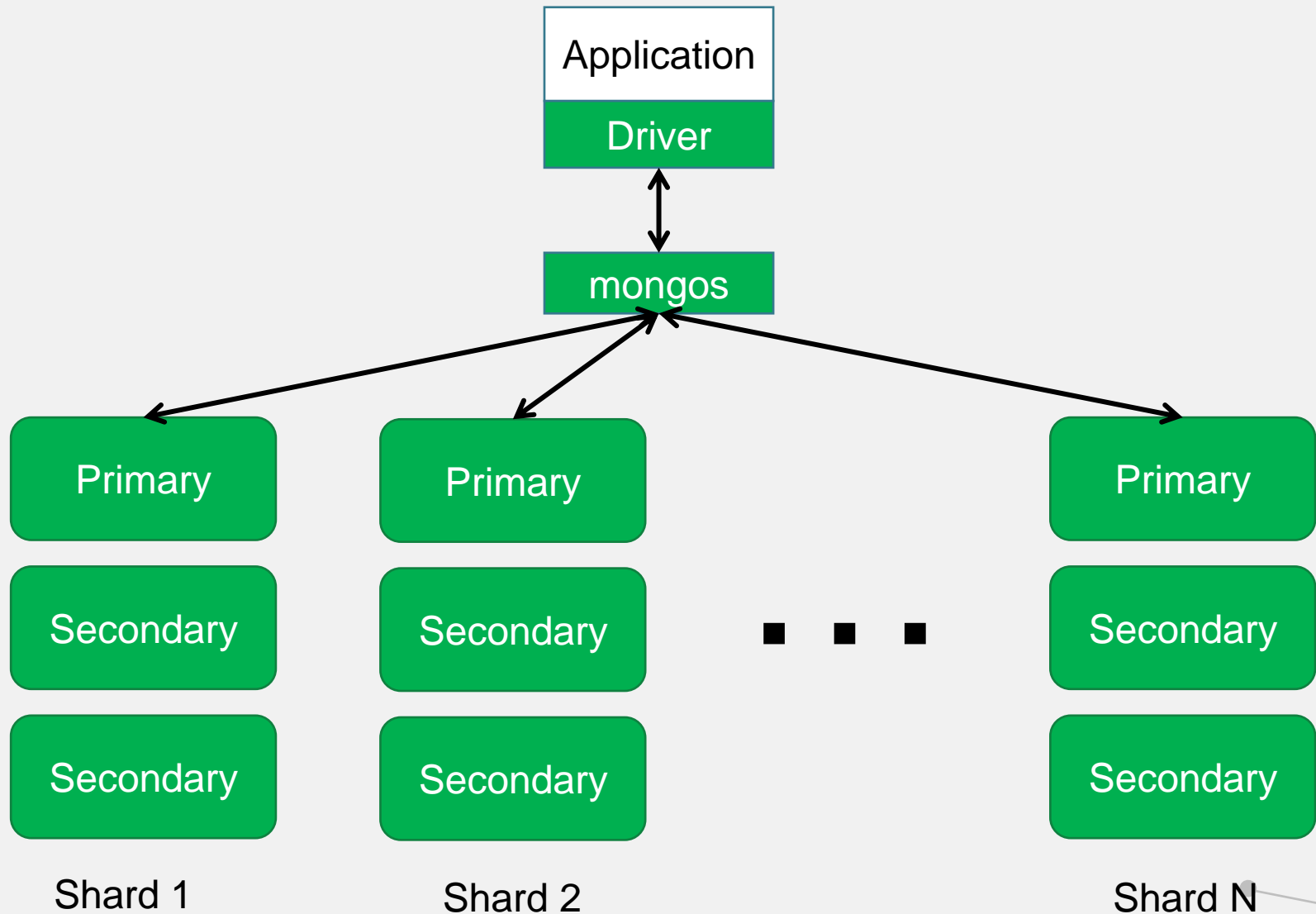
- ждать пока данные не будут записаны в лог
- ждать пока не будет записано несколько реплик
- ждать пока 2 реплики не будет записано в одном ДЦ, и 1 в другом

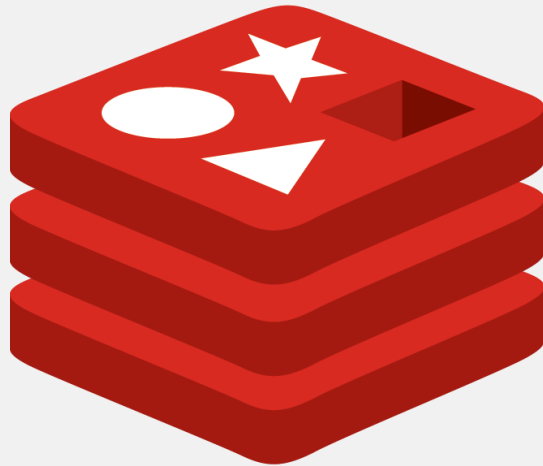
Репликация



- Данные реплицируются
- Количество реплик: 2–48
- Выделяется одна главная реплика
- Чтение и запись происходят в главную реплику
- В случае падения главной, выбирается новая

Архитектура MongoDB





redis

Особенности Redis



- NoSql база данных
- Key / value хранилище
- Value могут быть разных типов
- Работает в памяти, но сохраняет данные на диск
- Высокая производительность и масштабируемость
- Первая версия появилась в 2009 году
- Спонсировалась VMware



- Строки (get, set, incr, incrby)
- Списки (добавление $O(1)$, поиск по индексу $O(n)$, [push, pop, insert], [set, index, rem])
- Множества (операции $O(1)$, resize таблицы – блокирующий [sadd, srem, sdiff, sunion, sinter])
- Хеш-таблицы
- Упорядоченные множества
- Максимальный размер ключа: 512 Мб

Типы данных: Строки



```
127.0.0.1:6379> SET key somevalue
OK
127.0.0.1:6379> GET key
"somevalue"
127.0.0.1:6379> EXPIRE key 3600
(integer) 1
127.0.0.1:6379> TTL key
(integer) 3594
127.0.0.1:6379>
```

Типы данных: Хеши



```
127.0.0.1:6379> HMSET product:100 name "iPhone 5s" price 24900
OK
127.0.0.1:6379> HGETALL product:100
1) "name"
2) "iPhone 5s"
3) "price"
4) "24900"
127.0.0.1:6379>
```

Типы данных: Множества



```
127.0.0.1:6379> SADD plist:c:10 703 704
```

```
(integer) 2
```

```
127.0.0.1:6379> SMEMBERS plist:c:10
```

```
1) "703"
```

```
2) "704"
```

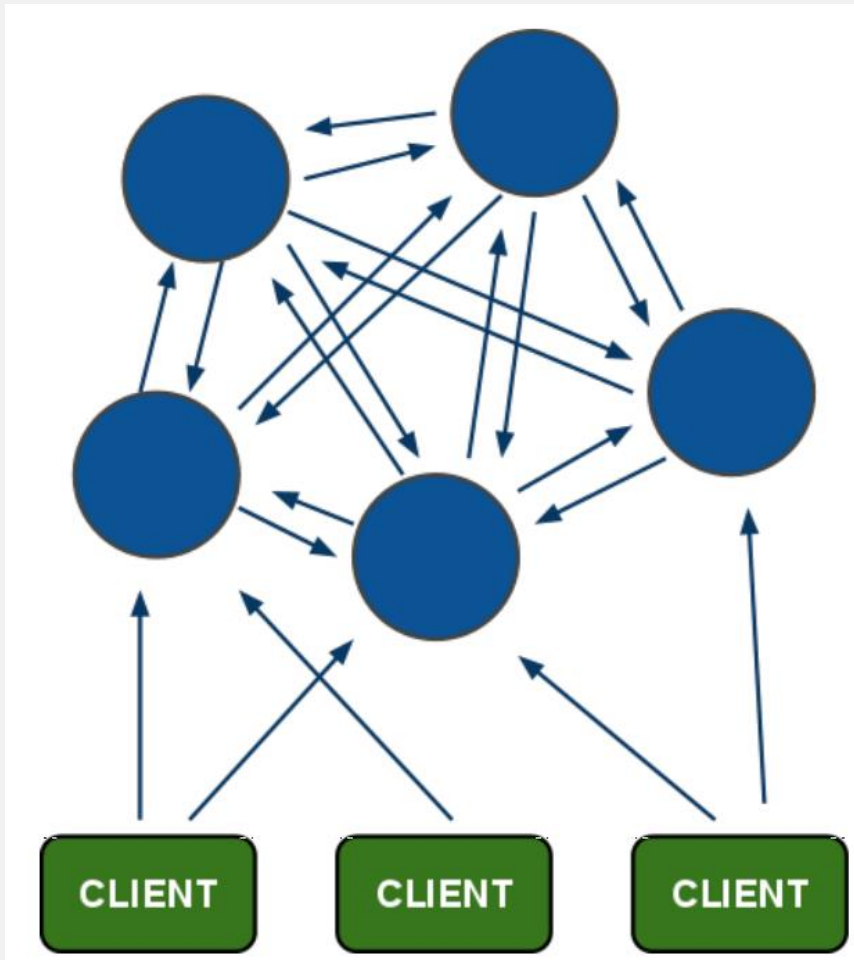
```
127.0.0.1:6379>
```

Типы данных: Сортированные множества



```
127.0.0.1:6379> ZADD comments:article:13 1414229007 "Great article"
(integer) 1
127.0.0.1:6379> ZADD comments:article:13 1414229017 "Thank you"
(integer) 1
127.0.0.1:6379> ZRANGE comments:article:13 0 -1
1) "Great article"
2) "Thank you"
127.0.0.1:6379> ZREVRANGE comments:article:13 0 -1
1) "Thank you"
2) "Great article"
127.0.0.1:6379>
```


Redis Cluster



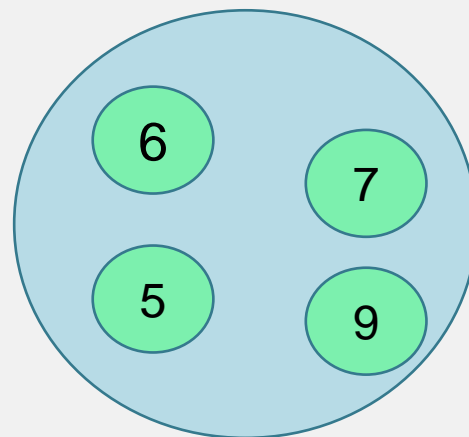
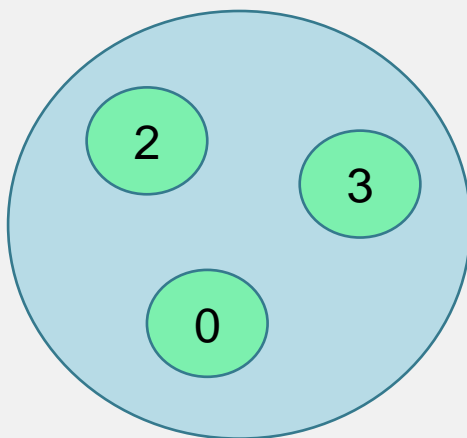
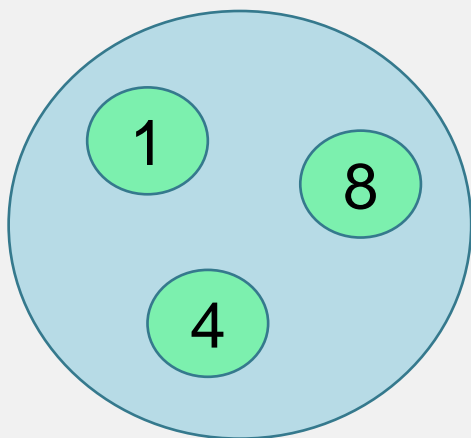
- Все ноды общаются между собой по специальному протоколу
- Ноды не проксируют запрос клиента
- Клиент работает с нодой как с обычным Redis

Распределение ключей



Все ключи делятся на 16384 частей:

$$\text{HASH_SLOT} = \text{CRC16}(\text{key}) \bmod 16384$$

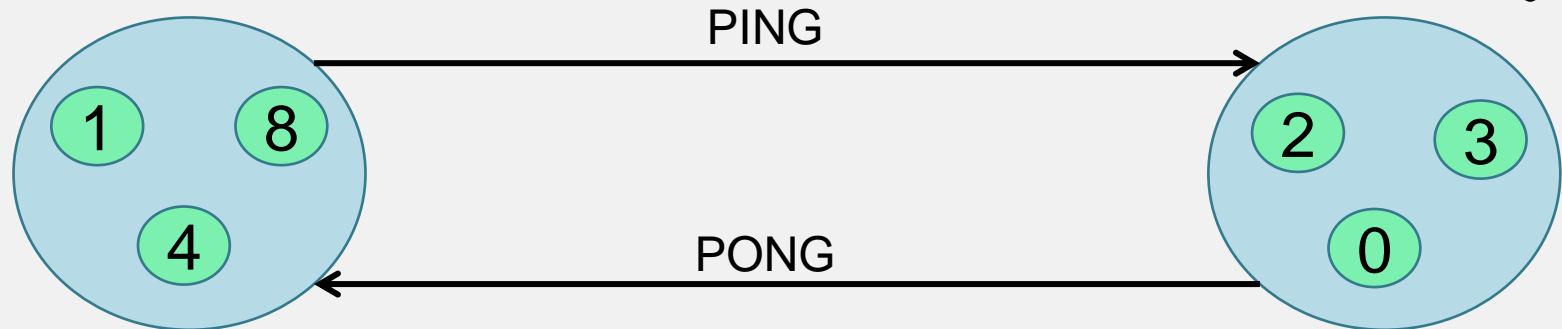


Kay hash tags



- Redis поддерживают множественные запросы
- Чтобы это не приводило к запросам по всему кластеру, используется механизм key tag:
- В ключе выделяется часть, по которой будет вычисляться слот:
 - `{user1000}.first` и `{user1000}.second` попадут в один слот

Сообщения между узлами



PING: Привет! Ты как?

Я мастер для слотов 1, 8, 4

Gossip: у меня есть некоторая информация про другие узлы:

- А отвечает на мой ping, думаю с ней все хорошо
- В не отвечает, возможно у нее проблемы

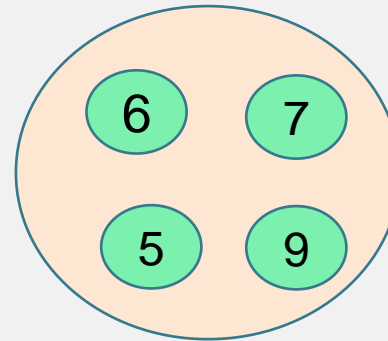
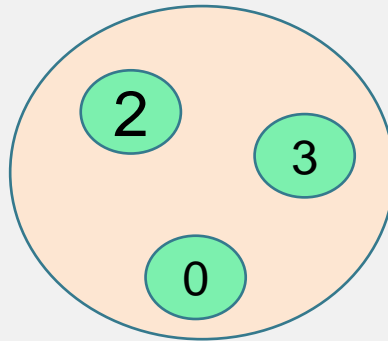
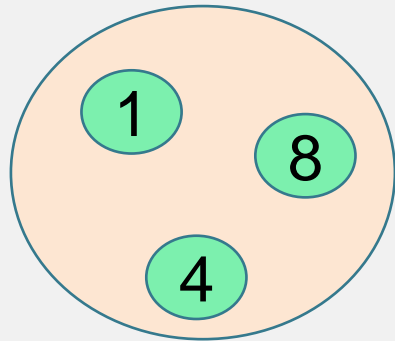
PONG: Привет! У меня все Ok!

Я мастер для слотов 2, 3, 0

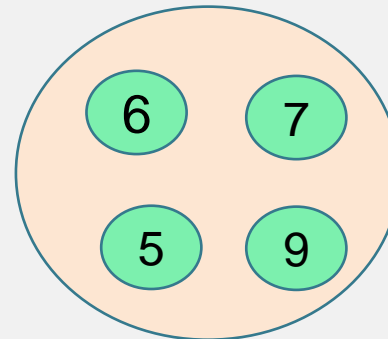
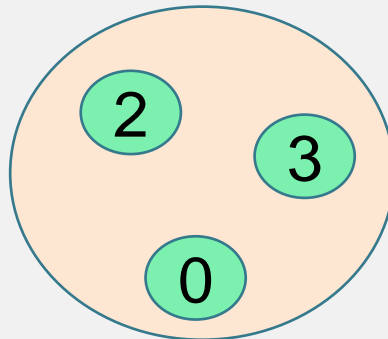
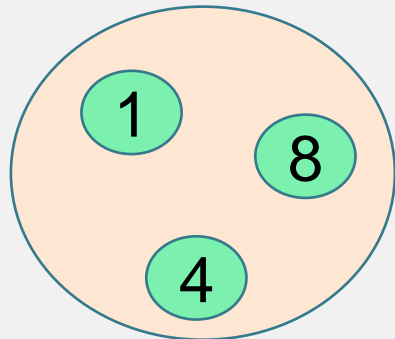
Gossip: а я поделюсь с тобой своей информацией:

- С и D в порядке, отвечают вовремя
- В мне тоже не отвечает. Значит она сломана.

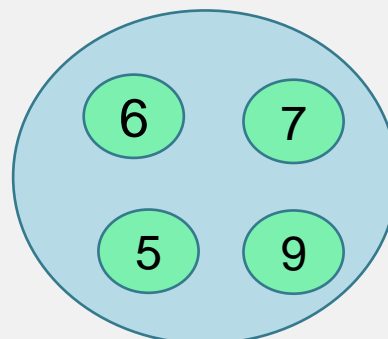
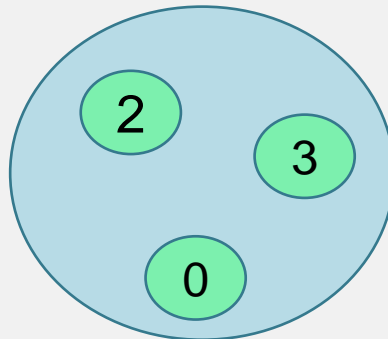
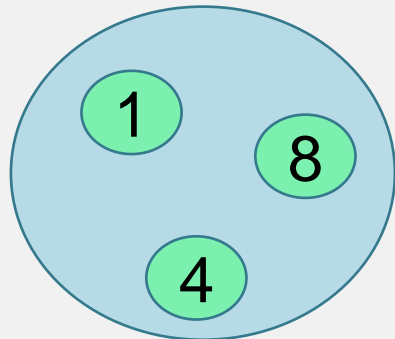
Master – Slaves



Slave



Slave



Master



- Репликация данных происходит асинхронно
 - нет задержки при записи
- Запись осуществляется только на мастер
- В случае выхода мастера из строя, выбирается другой мастер
- Считается, что актуальные данные есть только на мастере
 - нет мержа данных с разных нод



Для высокой производительности приходится жертвовать надежностью

Это приводит к тому, что:

- Данные находятся в неконсистентном состоянии:
 - клиент читает данные с необновившегося slave
- Записанные данные теряются

Потеря данных при записи

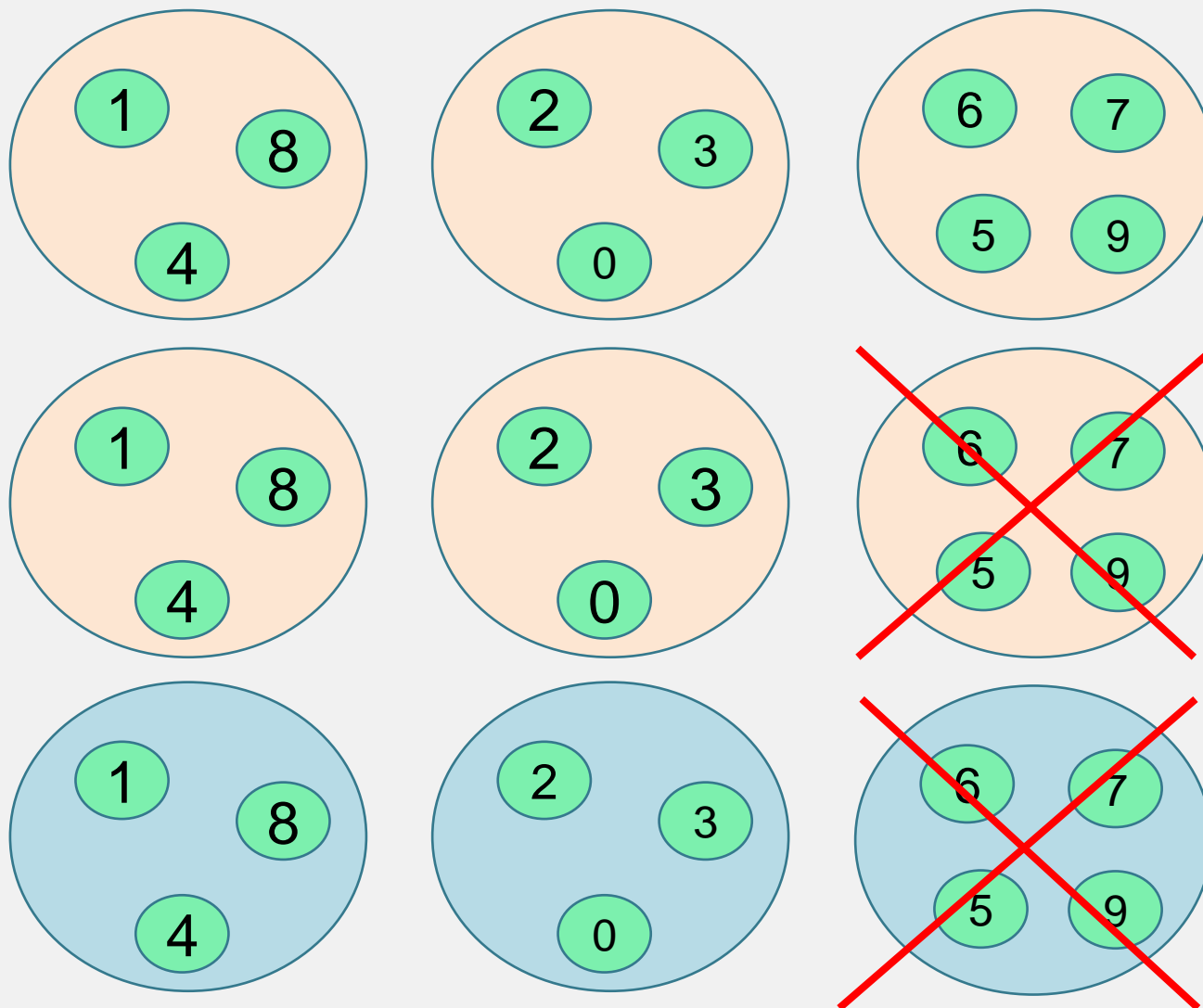


Сценарии потери данных:

1. Данные записываются на мастер, клиент получает сигнал об успешности записи, но данные не успевают отреплицироваться
2. Мастер становится недоступным, другой сервер берет на себя роль мастера, старый мастер возвращается, клиент пишет в старого мастера

Для консистентности используйте **WAIT** при записи

Выход из строя серверов



Выход из строя серверов



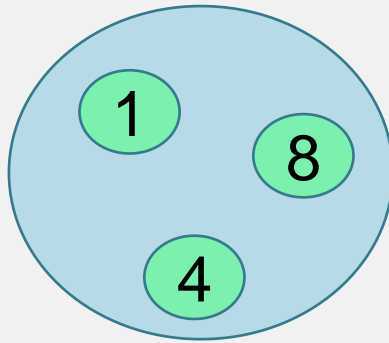
Пока есть хотя бы одна копия каждого слота, кластер считается рабочим

Какая вероятность, что при поломке 2-х произвольных серверов в кластере с 5 мастерами и двойной репликацией, произойдет потеря данных?

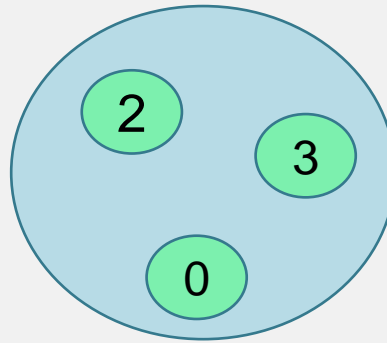
Ответ: $\sim 11,1\% \left(1 - 1/(2N-1) \right)$

Что будет в случае тройной репликации?

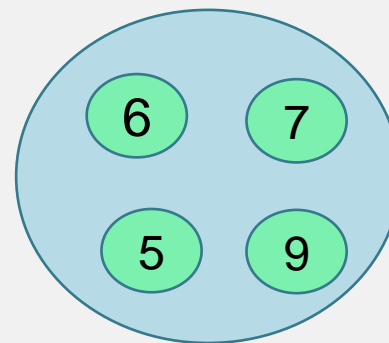
Простой клиент



A



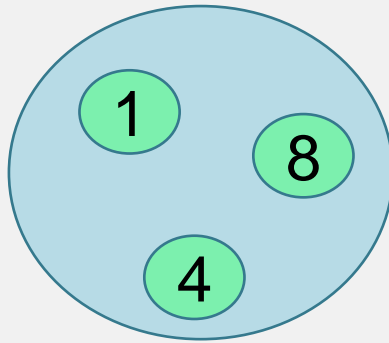
B



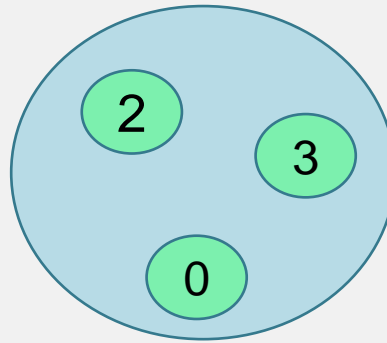
C

1. Client => A: **GET** foo
2. A => Client: **-MOVED 9** 192.168.5.21:6391
3. Client => B: **GET** foo
4. B => Client: "bar"

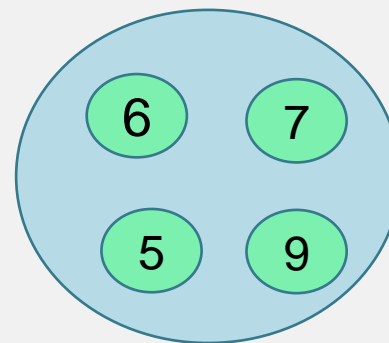
-MOVED 9 – ошибка: слот 9 лежит на другой ноде



A



B



C

1. Client => A: **CLUSTER HINTS**
2. A => Client: ... a map of hash slots -> nodes
3. Client => B: **GET foo**
4. B => Client: **"bar"**



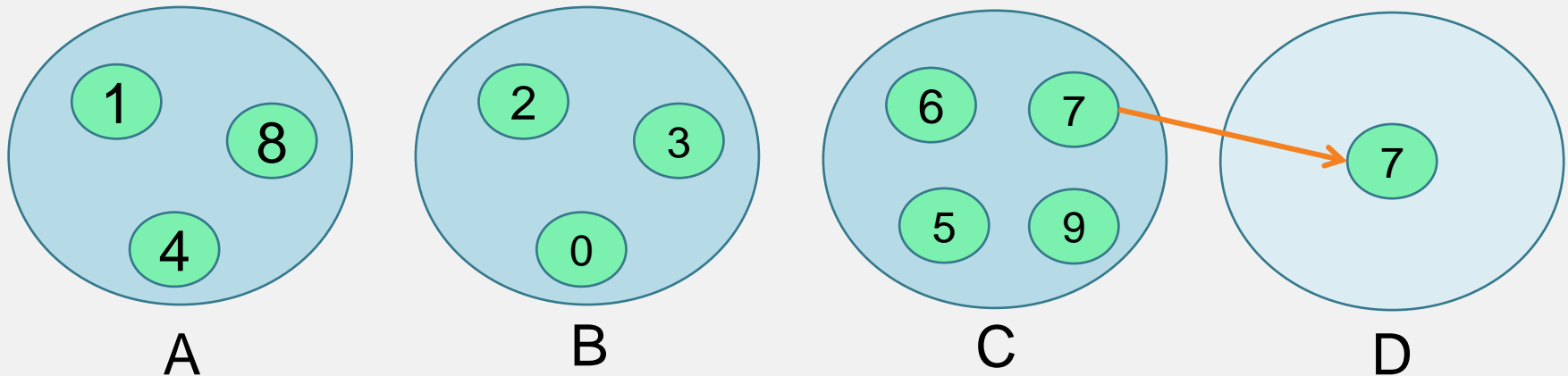
Простой клиент:

- Имеет соединение с 1 нодой
- Начинает чтение с произвольного сервера
- Поддерживает перенаправление

Умный клиент:

- Поддерживает соединения со всеми нодами
- Кеширует соответствие hash slot -> nodes
- Обновляет его при получении ошибки –MOVED

Масштабирование



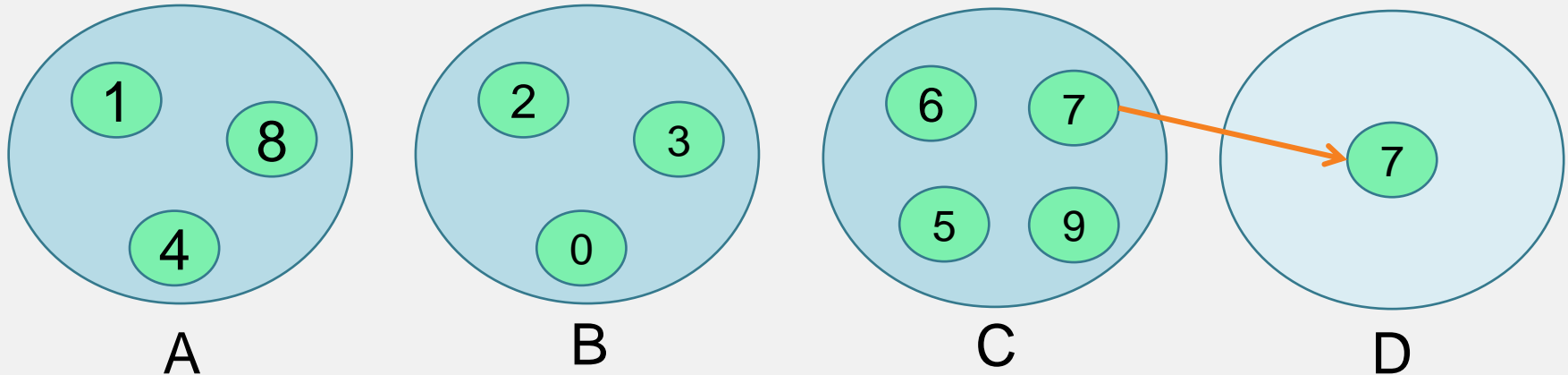
Нода C помечает слот 7 как “MOVING to D”

При запросе к C ключа из 7, происходит:

- возвращается значение, если ключ еще на C
- возвращается **–ASK D**, если ключ уже переехал

–ASK похож на –MOVED, но актуален только для этого запроса: умный клиент не должен обновлять свой конфиг

Масштабирование – перемещение данных

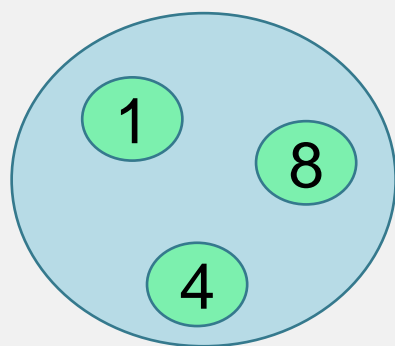


- Все новые ключи будут создаваться/обновляться на D
- Все старые ключи переносятся из C в D с помощью специальной утилиты: `redis-trib MIGRATE`
- `MIGRATE` – атомарная команда: переносит значение из C в D и удаляет на C только когда получит ОК с D
- Проблема: как получить следующий ключ в 7 эффективно

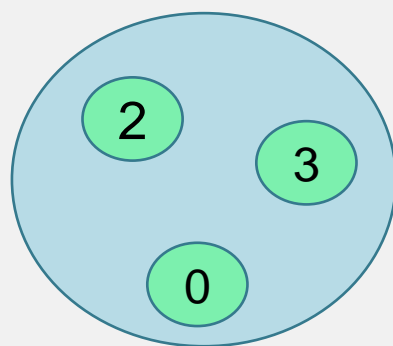
Масштабирование и падение ноды



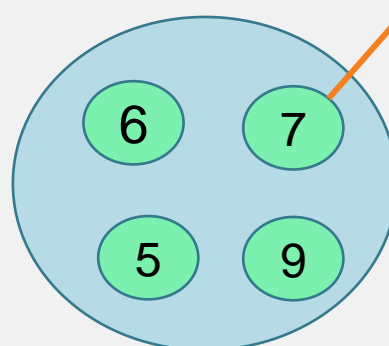
В случае падения ноды во время перемещения слота, запись продолжается на slave



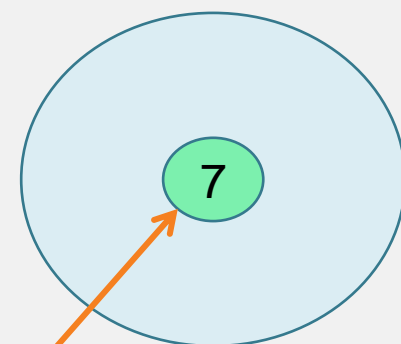
A



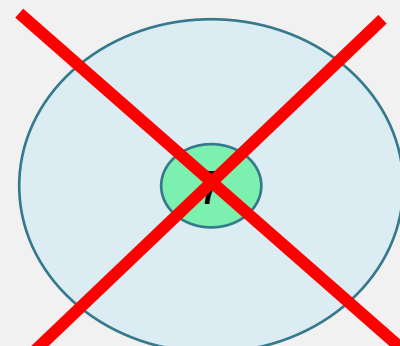
B



C



D-Slave

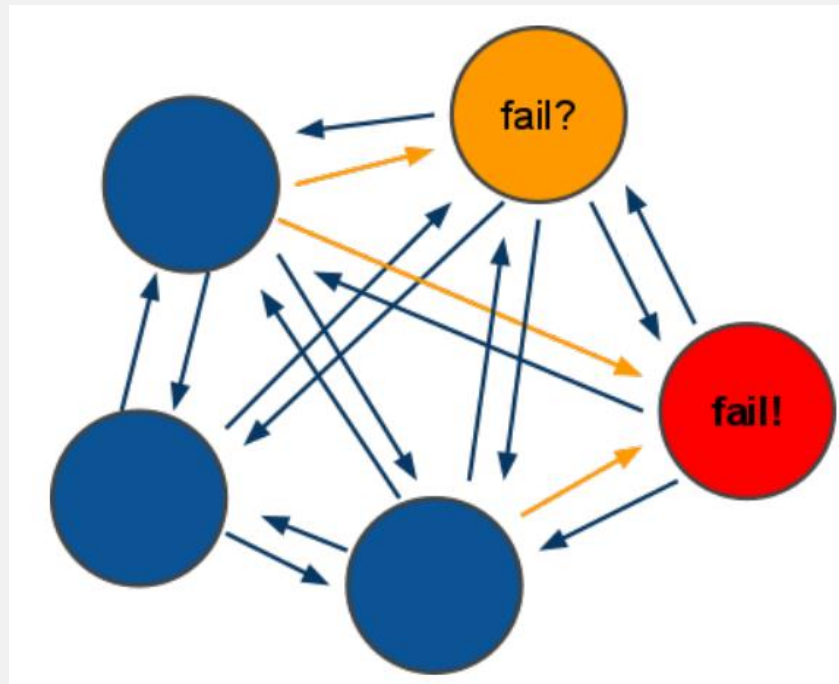


D

Fault tolerance



- Все ноды периодически посылают **PING** друг другу
- Нода маркируется “*возможно упавшей*”, если не отвечает в течении N секунд
- Каждый **PING** и **PONG** сигнал содержит секцию **gossip**: информацию о нодах, известную отправителю



Fault tolerance

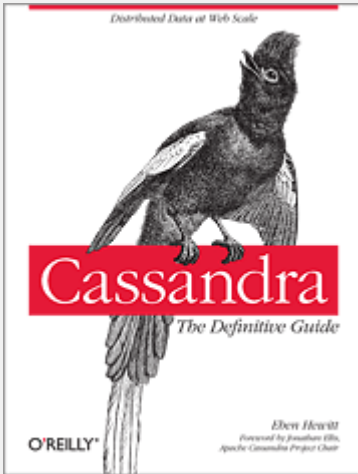


- **A** предполагает, что **B** упала, т.к. **PING** стаймаутил. Но **A** не предпринимает никаких действий пока.
- **C** посылает **A** **PONG** сигнал с секцией **gossip**. **C** тоже считает, что **B** не работает
- С этого момента **A** помечает **B** как “упавшая” и информирует об этом все остальные ноды кластера. Таким образом все ноды будут считать **B** недоступной
- Если **B** вернется, то при первом **PING-PONG** сигнале, она узнает, что **должна отключиться как можно быстрее**, чтобы отстающие клиенты не читали с нее

Сравнение

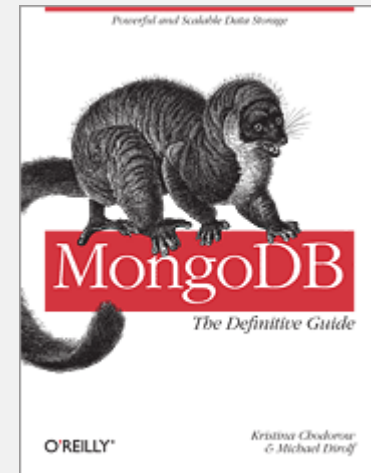


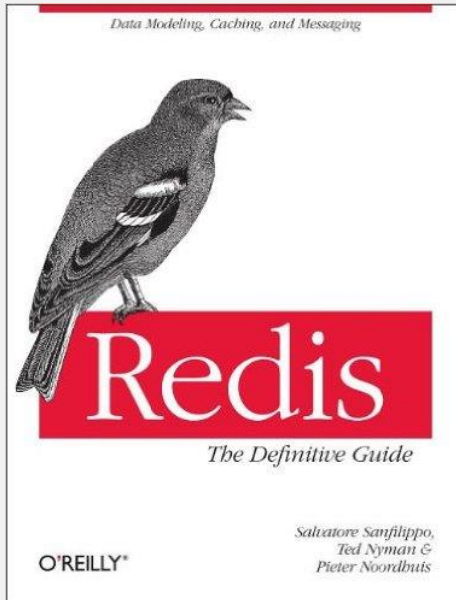
	HBase	Cassandra	MongoDB	Redis
Модель данных	Разреженная матрица	Разреженная матрица	JSON объекты	Словарь (map)
Типы данных	byte[]	string	BSON types	string, set, map, sorted set
Sharding	Диапазоны ключей	hash(key)->2 ⁿ диапазоны по кольцу	range-base hash-base location-aware	crc(key) % 16384
Master server	HMaster	любой сервер для каждого запроса	Query Router	нет главного
Репликация	HDFS	По кольцу	Master / Slave	Master / Slave
Consistency	Фиксированная	Можно управлять	Можно управлять + ACID для док	Можно управлять



Cassandra: The Definitive Guide Eben Hewitt (Author) O'Reilly Media; November 2010

MongoDB: The Definitive Guide Kristina Chodorow, Michael Dirolf





Redis: The Definitive Guide

Jay A. Kreibich_(Author)

O'Reilly Media; 2013

Очень хорошая документация на сайте: <http://redis.io/topics/cluster-spec>

Отмечайтесь и оставляйте отзыв

**Спасибо за
внимание!**

Евгений Чернов

e.chernov@corp.mail.ru