ТЕХНОСФЕРА

Лекция 9

NoSQL, HBase

Евгений Чернов

Способы хранения данных: память

- \+ Широкий выбор структур данных
- \+ Возможность создавать свои типы данных
- \+ Быстрый доступ к данным: чтение, изменение, дополнение
- \+ **Высокая скорость**

- − Размер данных ограничен оперативной памятью
- − Данные существуют, пока жив процесс
- − **Низкая надежность**

Способы хранения данных: файлы

- \+ Существенно больший объем данных
- \+ Свобода в формате и структуре данных
- \+ Простые механизмы доступа к данным в файле
- \+ Отсутствие третьей стороны при работе с данными

- − Сложно вносить изменения в файл
- − Медленный доступ к данным
- − Тяжело организовать совместный доступ к данным
- − Отсутствие контроля целостности данных
- − Сложные механизмы доступа к данным, расположенным в нескольких файлах

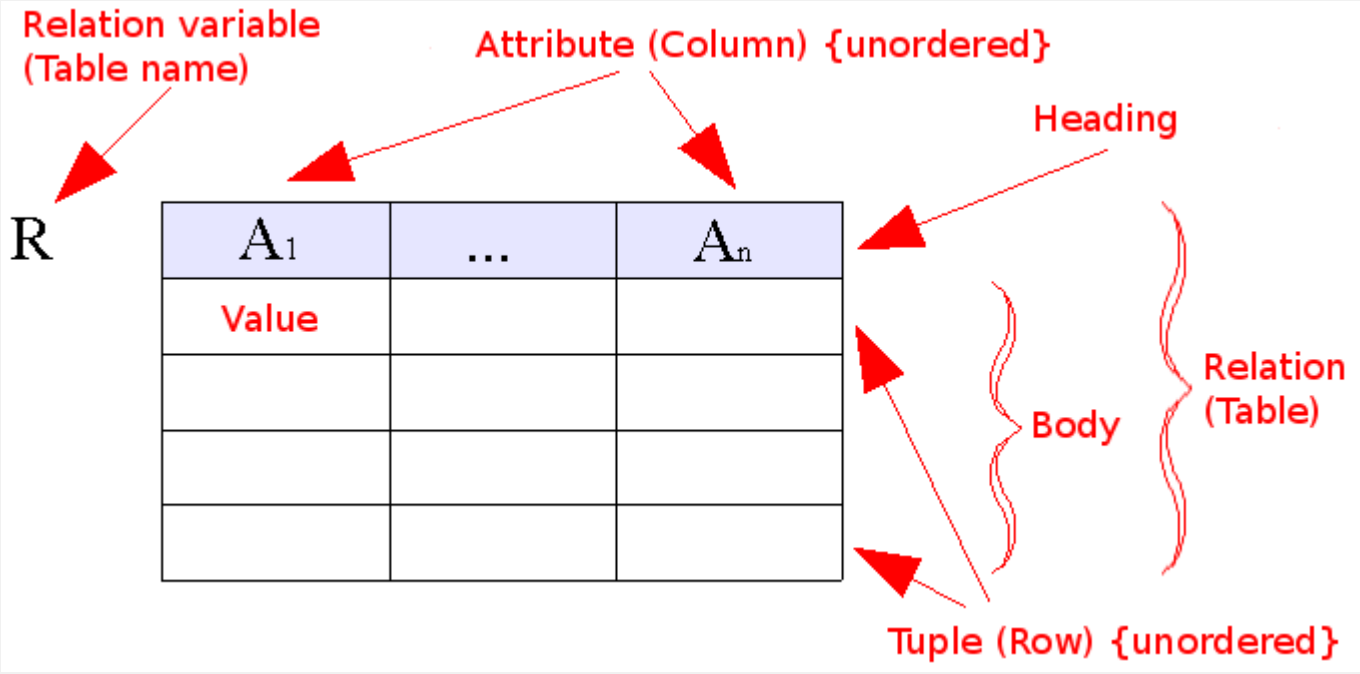
Реляционная Модель Данных

3 аспекта:

- Структурный аспект
- Аспект целостности
- Аспект обработки

Объекты:

- Отношение (relation)
- Кортеж (tuple)
- Атрибут (attribute)



Реляционные Базы Данных: ACID

Требования к транзакционной системе:

- **Atomicity** – Атомарность
  - Гарантия того, что транзакция не будет выполнена частично
- **Consistency** – Согласованность
  - Отсутствие нарушения целостности данных
  - Внутри транзакции согласованность может нарушаться
- **Isolation** – Изолированность
  - Другие транзакции не должны влиять на результат транзакции
- **Durability** – Надежность
  - Выполненные транзакции не должны пропасть после сбоя

Способы хранения данных: РБД

- \+ Универсальный доступ к данным (язык SQL)
- \+ Контроль за целостностью данных (ACID)
- \+ Одновременный доступ к данным
- \+ Повышенная безопасность

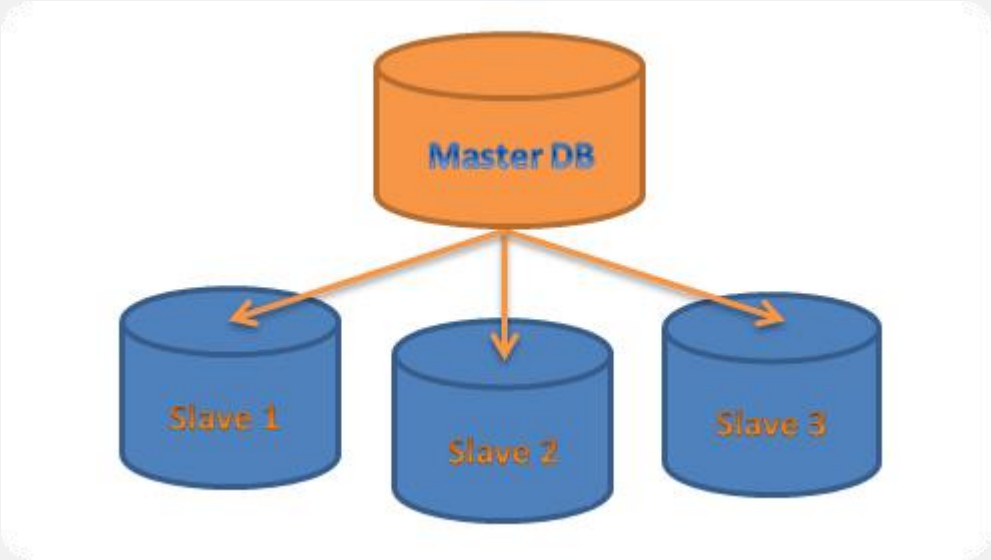
- \- Тяжело хранить иерархические данные
- \- Проблемы с маштабируемостью

Как хранятся данные?

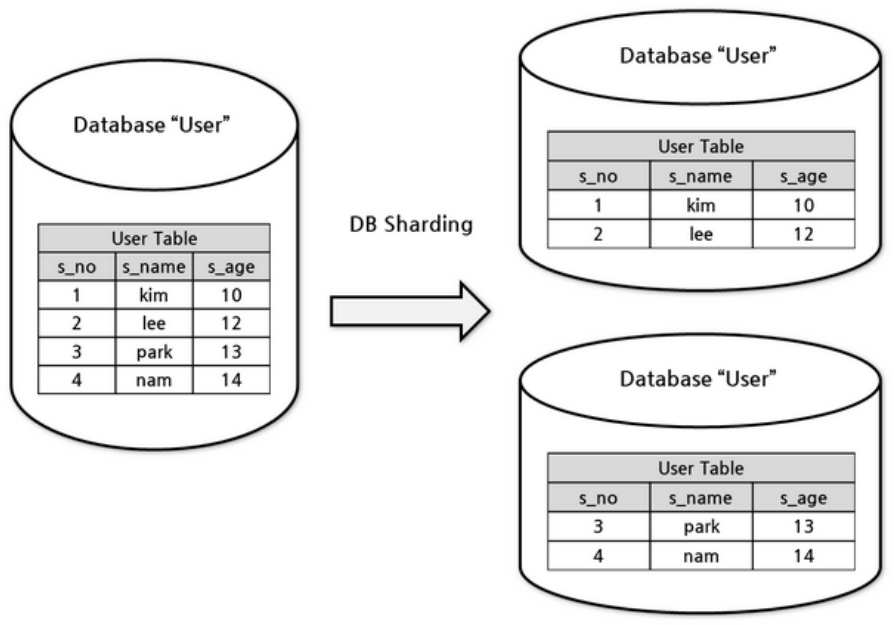
Системы	Хранение данных
Игры на ПК	Файлы
Социальные сети	Базы данных
Компилятор	Файлы, память
Поисковики	Файлы, базы данных
Игры на Dendy	Память
Online сервисы банков	Базы данных
Word, Excel	Файлы, память
ОС Windows (реестр)	Файлы
Интернет магазины	Базы данных

Масштабирование РБД – Master/Slave

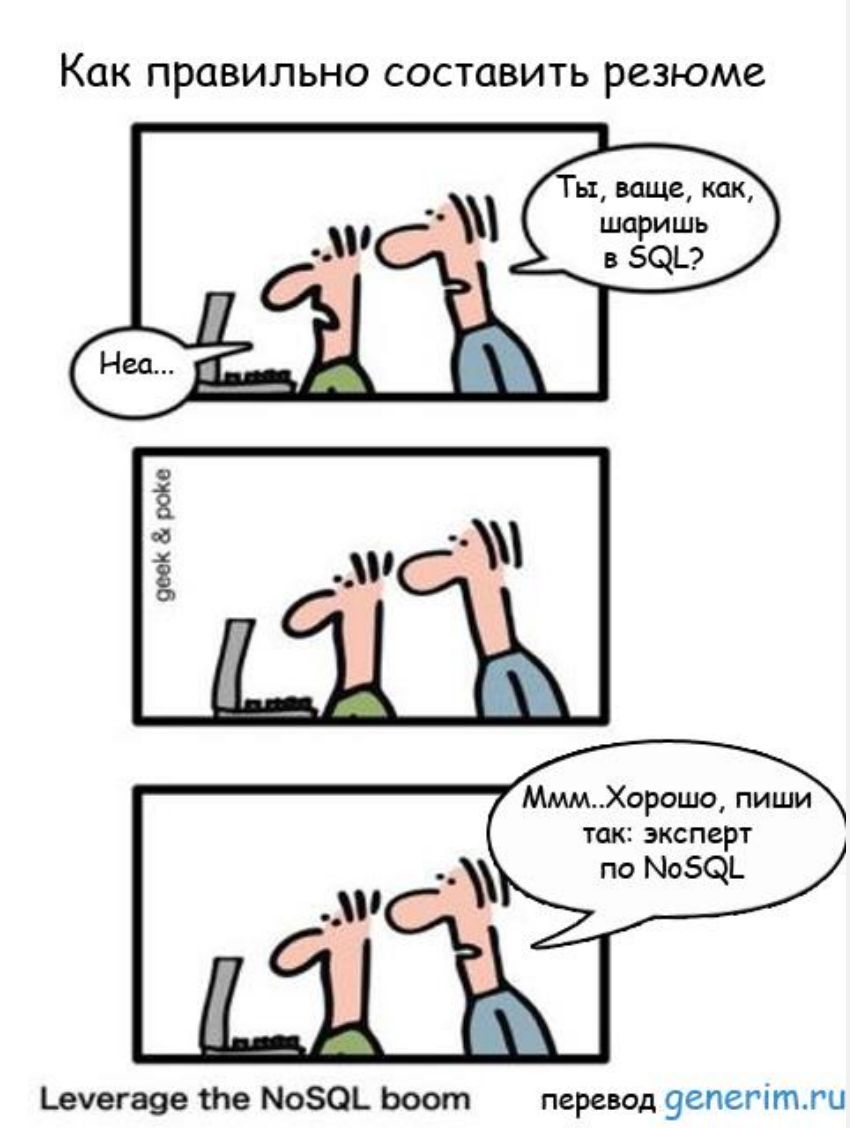
- Все операции записи идут на *Master nodes*
- Все операции чтения выполняются с реплицированных *Slave nodes*
- Критические операции чтения могут быть некорректны, т.к. данные еще неотреплицировались
- Большие объемы данные приносят большие проблемы из-за больших объемов репликации



Масштабируемость РБД – Sharding



- \+ Хорошо масштабируется для операций чтения и записи
- – Тяжело делать re-sharding
- – Нельзя делать join между шардами
- – На практике шардинг – это сложно


Как правильно составить резюме
Ты, ваще, как, шаришь в SQL?
Неа...
geek & poke
Ммм..Хорошо, пиши так: эксперт по NoSQL
Leverage the NoSQL boom
перевод generim.ru


«Не только SQL»
NoSQL
not only

Основные черты NoSQL?

- Применение различных типов хранилищ
- Возможность разработки БД без задания схемы
- Использование многопроцессорности
- Линейная маштабируемость
- Сокращение времени разработки

Базовые события в становлении NoSQL

BigTable (Google)

- *http://labs.google.com/papers/bigtable.html*

Dynamo (Amazon)

- *http://www.allthingsdistributed.com/2007/10/amazons_dynamo.html*
- *http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf*

CAP Theorem

CAP Theorem

Три основные свойства системы:

- ***Consistency*** - во всех вычислительных узлах в один момент времени данные не противоречат друг друг
- ***Availability*** - любой запрос к распределённой системе завершается корректным откликом
- ***Partitionability*** - расщепление распределённой системы на несколько изолированных секций не приводит к некорректности отклика от каждой из секций

Теорема: Можно иметь только два из трех свойств в любой *shared-data* системе

- Для горизонтального масштабирования надо применять *partitioning*
- Поэтому выбираем из *consistency* и *availability*
- Почти во всех случаях выбирается *availability* вместо *consistency*

Consistency Models

Определяют правила показа и порядка видимости обновлений данных:

- **Строгая (*strict consistency*)**
- Последовательная (*sequential consistency*)
- Причинная (*causal consistency*)
- Процессорная (*processor consistency*)
- Слабая (*weak consistency*)
- **Консистентность в конечном счете (*eventual*)**
- Консистентность по выходу (*release consistency*)
- Консистентность по входу (*entry consistency*)

Consistency Model: пример

- Данные по ключу **X** реплицируются на ноды **M** и **N**
- Клиент **A** пишет по ключу **X** на ноду **N**
- Проходит некоторый период времени ***t***
- Клиент **B** читает данные по ключу **X** с ноды **M**
- **Видит ли клиент B записанные данные от A?**
  - Strict Consistency: *Да*
  - Eventual consistency: *Может быть*

CAP теорема говорит: *Strict Consistency* не может быть достигнута одновременно с *availability* и *partition-tolerance*

BASE вместо ACID

BASE:

- ***B**asically **A**vailable* – базовая доступность
  - Каждый запрос гарантированно завершается
- ***S**oft State* – гибкое состояние
  - данные могут быть несогласованны какое-то время
- ***E**ventually Consistent* – согласованность в конечном счете

Если какое-то долгое время не происходит никаких операций обновления данных, то система будет в консистентном состоянии

Для данного *update* и данной ноды в итоге либо *update* достигнет ноды, либо нода будет удалена из кластера

Типы NoSQL

Key/Value (модель данных: *хеш-таблица*)
- Amazon S3 (Dynamo)
- Voldemort

Column-based (модель данных: *разряженная матрица*)
- HBase
- Cassandra

Document-based (модель данных: *дерево*)
- MongoDB
- OrientDB

Graph-based (модель данных: *граф*)
- Allegro
- InfiniteGraph

Key/Value

- \+ Высокая скорость
- \+ Хорошая масштабируемость
- \+ Простая модель данных
- \+ Горизонтально масштабируема

- − Многие структуры данных сложно представить в виде *KEY/VALUE*

Column-based

+ Более богатые структуры данных

- Меньшая согласованность данных
- Хуже масштабируется

HBase (Hadoop Data Base)

Особенности HBase

Распределенная база данных:

- Работает на кластере серверов
- Легко горизонтально маштабируется

NoSQL база данных:

- Не предоставляет SQL-доступ
- Не предоставляет реляционной модели

Особенности HBase

Column-Oriented хранилище данных

- нет фиксированной структуры колонок
- произвольное число колонок для ключа

Спроектирована для поддержки больших таблиц

- Миллиарды строк и миллионы колонок

Поддержка произвольных операций чтения/записи

Особенности HBase

- Основана на идеях Google BigTable
  - http://labs.google.com/papers/bigtable.html
- BigTable поверх GFS => HBase поверх HDFS
- Масштабируемость с помощью шардирования
- Автоматический fail-over
- Простой Java API
- Интеграция с MapReduce

Когда нужно использовать HBase

Подходит для больших объемов данных

- Если данных мало, то они все будут лежать на одной или нескольких нодах -> плохая утилизация кластера

Паттерн доступа к данным

- Выборка значений по заданному ключу
- Последовательный скан в диапазоне ключей

Свободная схема данных

- Строки могу существенно отличаться по своей структуре
- В схеме может быть множество колонок и большинство из них будет равно *null*

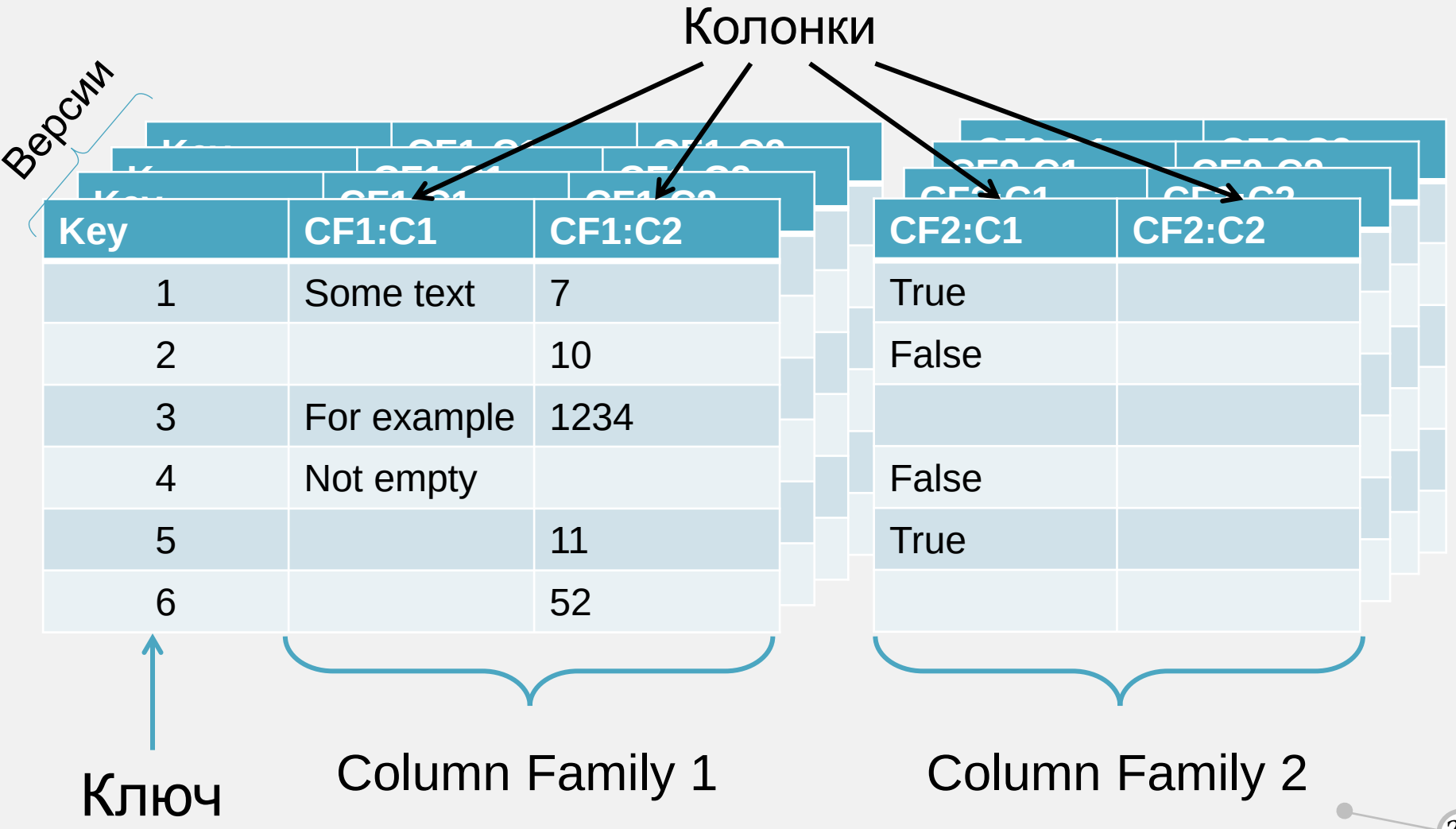
Когда НЕ нужно использовать HBase

Традиционный доступ к данным в стиле РБД

- Приложения с транзакциями
- Реляционная аналитика (*'group by', 'join‘ u 'where column like‘* и т.д.)

Плохо подходит для доступа к данным на основе текстовых запросов (*LIKE %text%*)

Модель данных Hbase



Модель данных Hbase

- Данные хранятся в таблицах (*table*)
- Таблицы содержат строки (*row*)
  - Доступ к строке осуществляется по уникальному ключу
    - Ключ – это байтовый массив
    - Все может быть ключом
  - Строки отсортированы в лексиграфическом порядке по ключам
- Строки группируются по колонкам в *column families*
- Данные хранятся в ячейках (*cells*) в виде байтового массива
- Доступ к ячейке: *row -> column-family -> column*

HBase Column Families

Column Family описывает общие свойства колонок:

- Сжатие
- Количество версий данных
- Время жизни (Time To Live)
- Опция хранения только в памяти (In-memory)
- Хранится в отдельном файле (HFile/StoreFile)

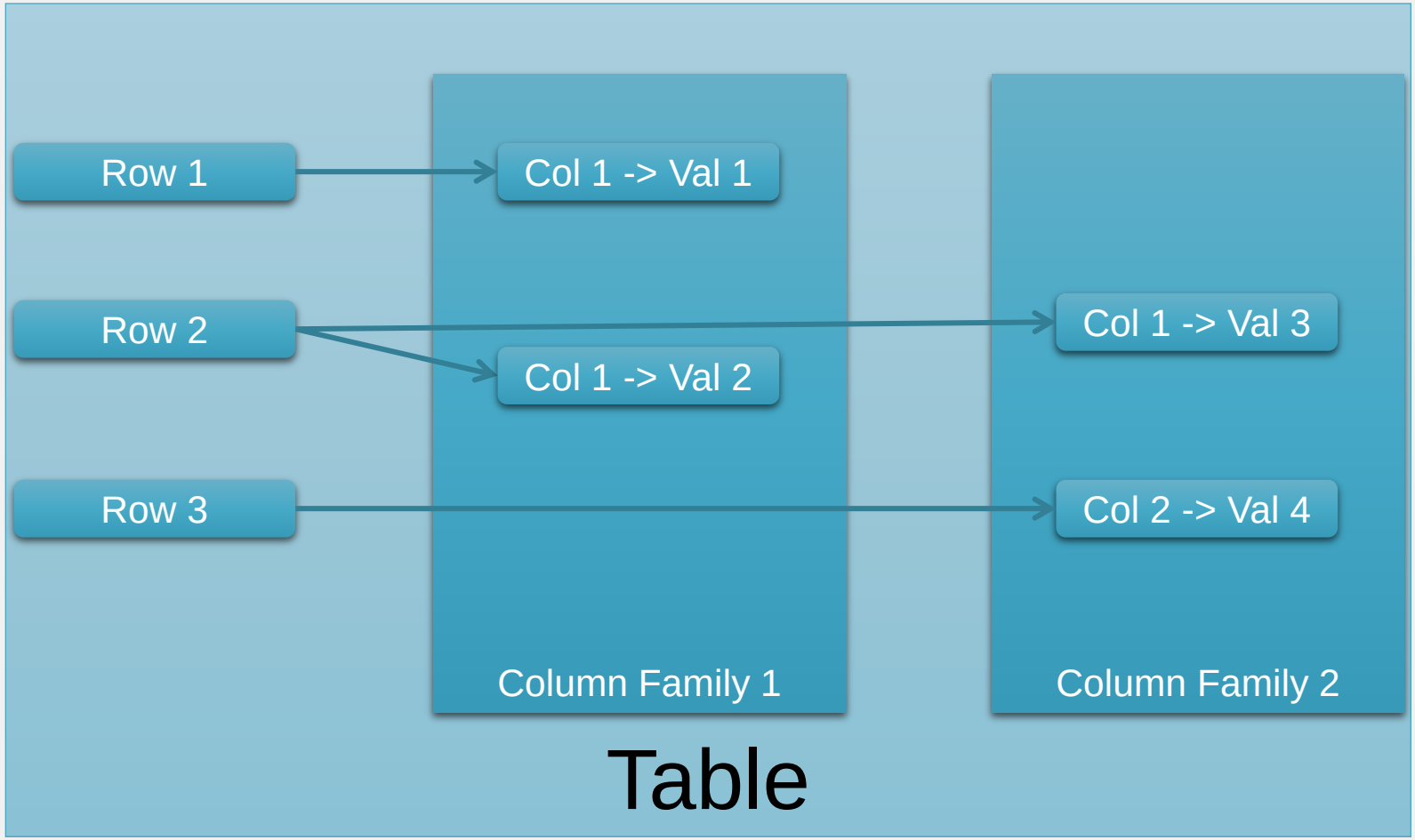
Конфигурация *CF* статична

- Задается в процессе создания таблицы
- Количество *CF* ограничено небольшим числом

Колонки наоборот НЕ статичны

- Создаются в runtime
- Может быть сотни тысяч для одной CF

HBase Column Families



HBase Timestamps

В ячейке может хранится несколько версий данных

- Настраивается в конфигурации CF
- 3 по-умолчанию
- Дополнительное измерение для данных

Timestamp данных

- Задается неявно при записи (текущим значением)
- Или явно указывается клиентом

Версии хранятся в убывающем порядке ts

- Последнее значение читается первым

Value = Table + RowKey + Family + Column + Timestamp

HBase Cells

Row Key	Timestamp	CF: “Data”		CF: “Meta”		
		Url	Html	Size	Date	Log
row1	t1	Mail.Ru				Log text 1
	t2				123456	Log text 2
	t3		<html>…	1234		Log text 3
	t4		<HTML>…	2345		Log text 4
row2	t1	OK.Ru			123765	Log text 1
	t2					Log text 2

HBase Cells

Row Key	Timestamp	CF: “Data”		CF: “Meta”		
		Url	Html	Size	Date	Log
row1	t1	**Mail.Ru**				Log text 1
	t2				**123456**	Log text 2
	t3		<html>…	1234		Log text 3
	t4		**<HTML>…**	**2345**		**Log text 4**
row2	t1	**OK.Ru**			**123765**	Log text 1
	t2					**Log text 2**

Архитектура

Column Family

Как хранить разряженную матрицу?

Column Family

Key	Column1	Column3	TimeStamp
Row1	value1		timestamp1
		value2	timestamp2
Row2	value4		timestamp1
Row3	value1	value6	timestamp1

HFile

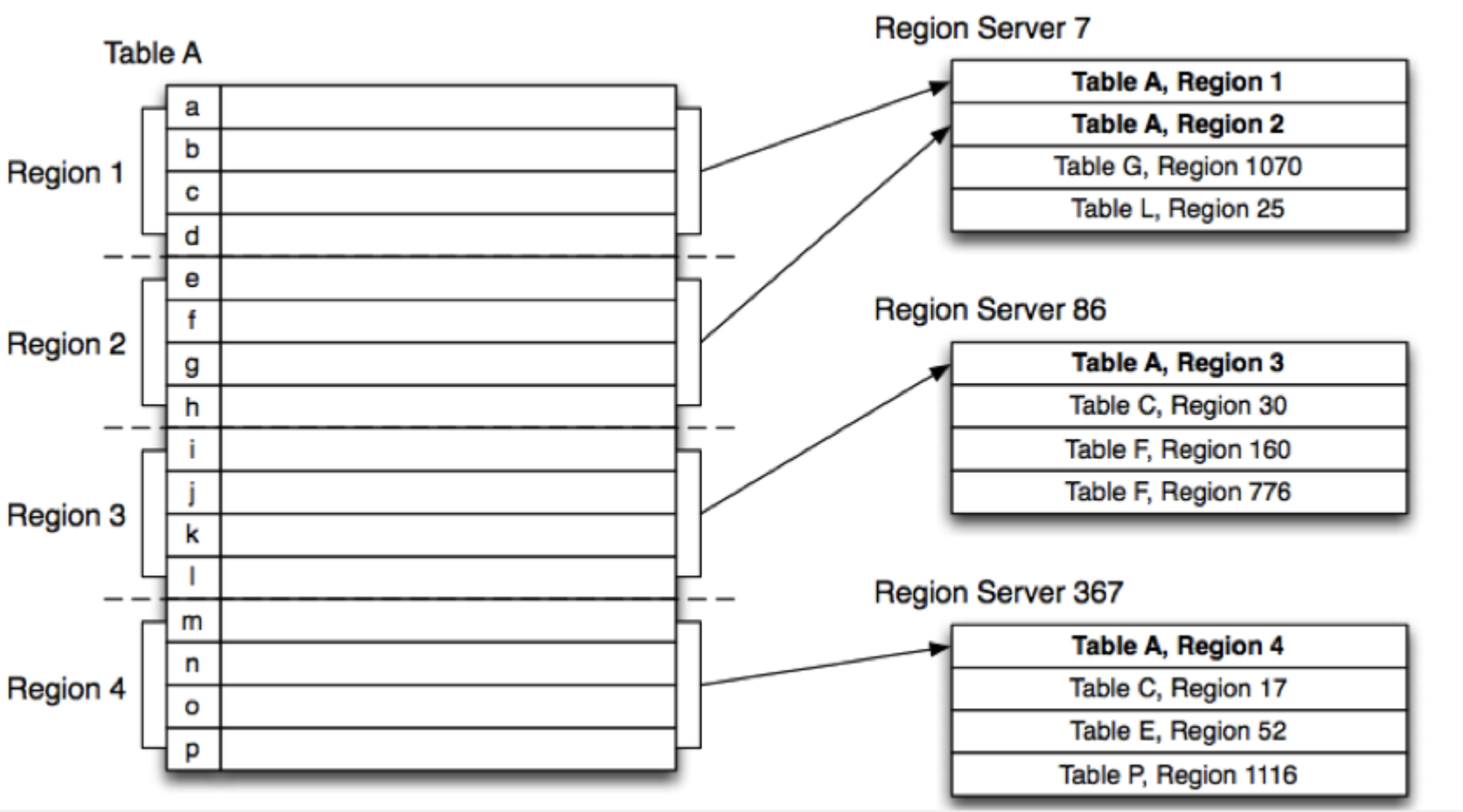
```
Row1:ColumnFamily:column1:timestamp1:value1
Row1:ColumnFamily:column3:timestamp2:value3
Row2:ColumnFamily:column1:timestamp1:value4
Row3:ColumnFamily:column1:timestamp1:value1
Row3:ColumnFamily:column3:timestamp1:value6
```

KeyValues

Масштабируемость в Hbase

- **Таблица** делится на регионы
- **Регион** – это группа строк, которые хранятся вместе
  - Единица шардинга
  - Динамически делится пополам, если становится большим
- **RegionServer** – демон, который управляет одним или несколькими регионами
  - Регион принадлежит только одному RS
- **MasterServer (HMaster)** – демон, который управляет всеми RS

Распределение ключей в RegionServer



Как осуществляется репликация регионов?

Hbase Regions

Регион – это диапазон ключей: [*Start Key; Stop Key)*

- *Start Key* включается в регион
- *Stop Key* не включается

- По-умолчанию есть только один регион
- Но можно предварительно задать кол-во регионов
- При привышении лимита размера региона, он разбивается (*split*) на 2 региона по среднему ключу

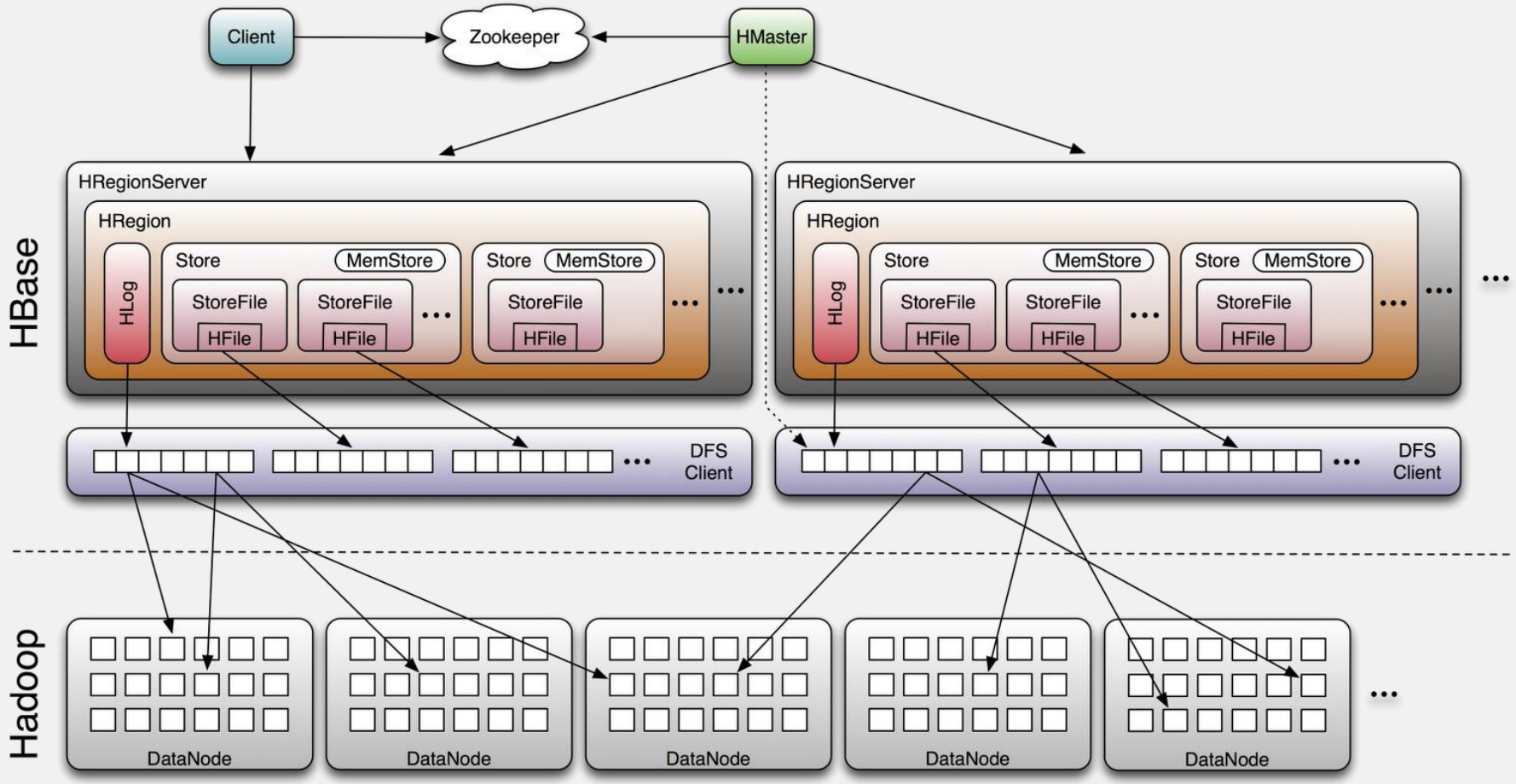
Количество регионов на RegionServer зависит от железа.
На практике:

- От 10 до 1000 регионов
- Размер региона до 2-3 Gb

Hbase Regions Split

- \+ Регионы более сбалансированны по размеру
- \+ Быстрое восстановление если регион зафейлился
- \+ Баланс нагрузки на RS
- \+ Split – это быстрая операция
  - Происходит в бэкграунде автоматически
  - Прозрачна для пользователей
- – На больших инсталляциях лучше производить в ручную

Архитектура Hbase



HBase Master

Управляет регионами:

- Назначает регион на RegionServer
- Перебалансирует их для распределения нагрузки
- Восстанавливает регион, если он становится недоступен

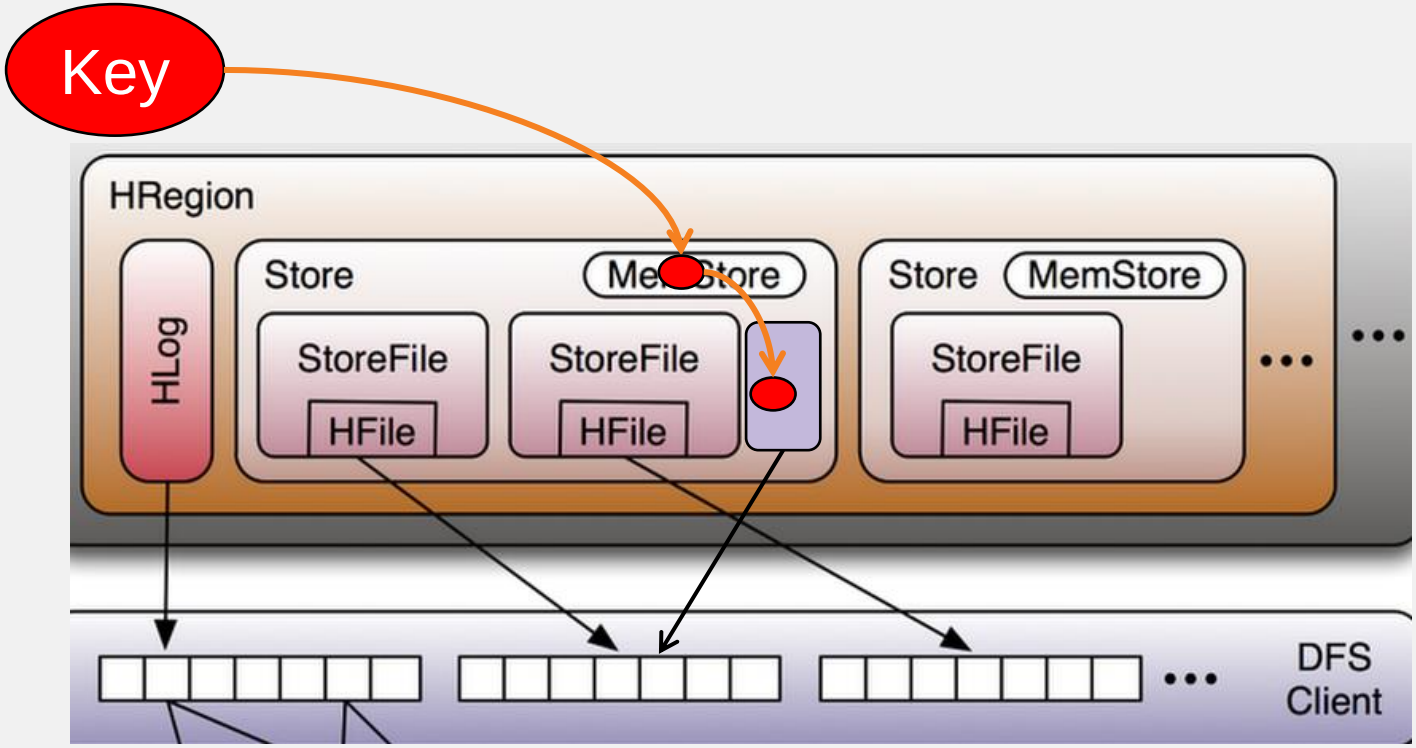
Не хранит данные

- Клиент взаимодействует напрямую с RS
- Обычно не сильно нагружен

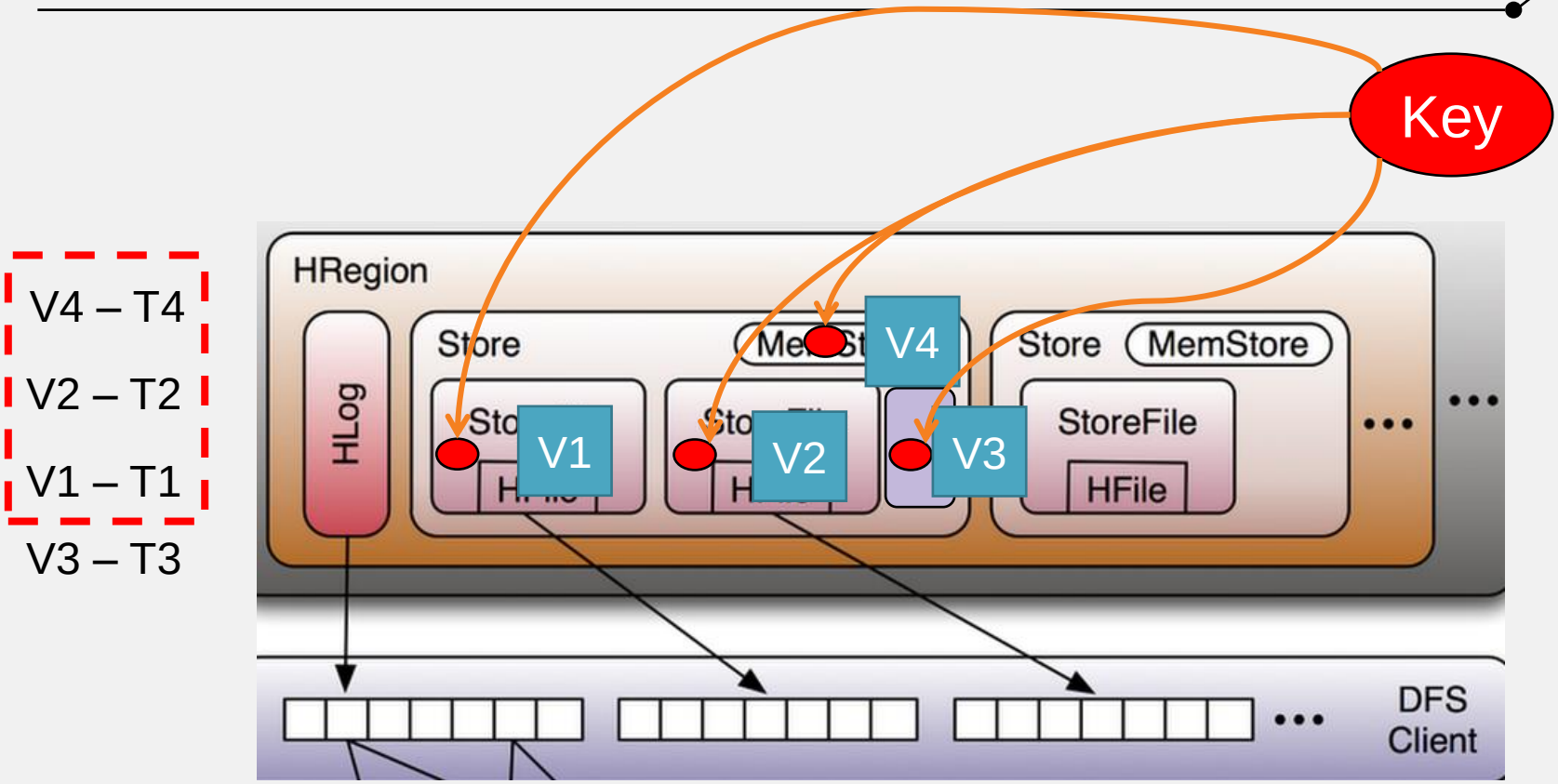
Отвечает за управление схемой таблиц и ее изменений

- Добавляет/удаляет таблицы и CF

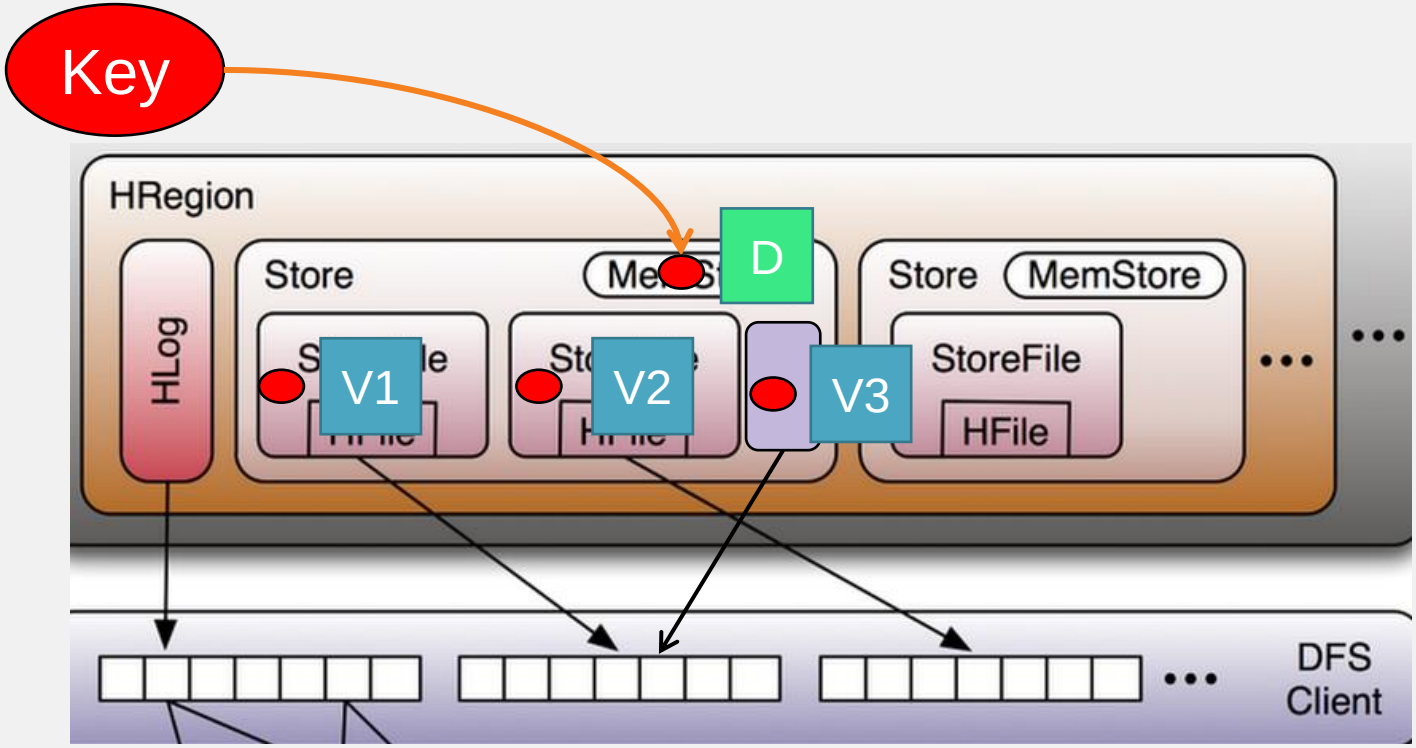
Data Storage: запись данных



Data Storage: чтение данных



Data Storage: удаление данных



Key -> {V1, V2, V3, D}

Data Storage: compaction

Minor Compaction

- Маленькие HFile'ы объединяются в бОльшие файлы
- Быстрая операция
- Маркеры удаления не применяются

Major Compaction

- Для каждого региона все файлы в рамках одной CF объединяются в один файл
- Используются маркеры удаления для того, чтобы не включать удаленные записи

Доступ к HBase

HBase Shell

Native Java API

Avro Server

- *Cross-language schema compiler*
- *http://avro.apache.org*
- Требует для запуска *Avro Server*

PyHBase

REST Server

Thrift

- *Cross-language schema compiler*
- *http://thrift.apache.org*
- Требует для запуска *Thrift Server*

Запрос данных из HBase

По RowId или набору RowIds

- Отдает только те строки, которые соответствуют заданным id
- Если не заданы дополнительные критерии, то отдаются все *cells* из всех *CF* заданной таблицы
  - Это означает слияние множества файлов из HDFS

По *ColumnFamily*

- Уменьшает количество файлов для загрузки

По *Timestamp/Version*

- Может пропускать полностью Hfile'ы, которые не содержат данный диапазон timestamp'ов

Запрос данных из HBase

По *Column Name/Qualifier*

- Уменьшает объем передаваемых данных

По *Value*

- Можно пропускать ячейки используя фильтры
- Самый медленный критерий выбора данных
- Будет проверять каждую ячейку

Фильтры могут применяться для каждого критерия выбора

- *rows, families, columns, timestamps u values*

Можно использовать множественные критерии выбора данных

ТЕХНОСФЕРА

Отмечайтесь и оставляйте отзыв

Спасибо за внимание!

Евгений Чернов

e.chernov@corp.mail.ru