

# Apache HBase

# Хранение данных на HDFS

- ✓ Удобно обрабатывать MapReduce
- ✓ Легко добавлять данные
- Отсутствует произвольный доступ
- Сложно обновлять

# Key-Value хранилище

Под Key-Value хранилищем подразумеваем “классические хранилища” – MemCache, Aerospike, Redis...

✓ Легко добавлять/обновлять данные

- Данные плохо структурированы
- Неудобно работать с последовательными данными
- Редко реализована поддержка MapReduce

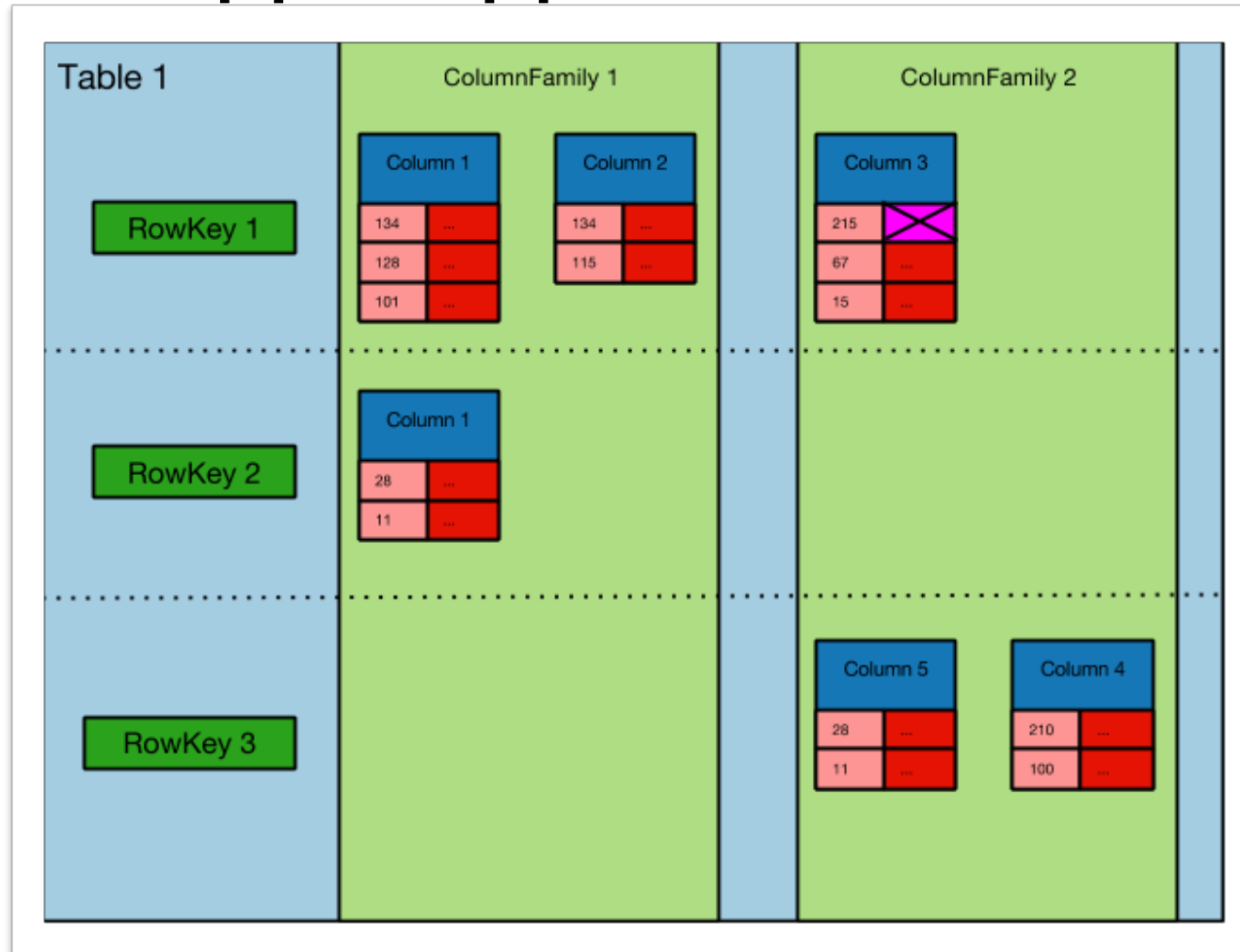
# Big Table

- Как и в случае с MapReduce придумано в Google:
  - Bigtable: A distributed storage system for structured data
  - <http://www.informatika.bg/resources/bigtable-osdi06.pdf>
- Hbase – открытая реализация принципов и идей изложенных в этой статье.

# Пакетная обработка vs произвольный доступ



# Модель данных HBase



Источник картинки: <http://www.ymc.ch/introduction-to-hbase>

# Модель данных Hbase

- Каждая запись в **таблице** проиндексирована при помощи первичного ключа, который называется **RowKey**
- Для каждого ключа **RowKey** может храниться неограниченное количество атрибутов, размещаемых в колонках (**columns**)
- **Колонки** не определяются схемой и могут быть добавлены "на лету"
- **Таблицы** являются разреженными. Пустые значения не требуют дополнительного места для хранения
- Для каждого атрибута может храниться несколько **версий**. Каждая **версия** имеет свой **Timestamp**.
- Записанное в Hbase **значение** не может быть изменено. Вместо этого необходимо добавить новую **версию** с более свежим **timestamp'ом**
- Для удаления записи помечаются специальным **маркером**
- **Колонки** группируются в **группы колонок(Column Family)**. **Группы колонок** определяются в момент создания таблицы и не могут быть изменены после
- Hbase является распределенной системой. Гарантируется что данные соответствующие одному **значению** и **группе колонок** хранятся вместе.

# Модель данных HBase

- Распределенное, многомерное, разреженное, сортированное отображение
- (Table, RowKey, ColumnFamily, Column, Timestamp) -> value
- OOP-way
- Table ← SortedMap<RowKey, Row>
- Row ← List<ColumnFamily>
- ColumnFamily ← SortedMap<Column, List<Entry>>
- Entry ← Tuple<Timestamp, Value>

Источник: <http://www.ymc.ch/introduction-to-hbase>



# Поддерживаемые операции

- **Get**
  - Получить все атрибуты для заданного ключа
- **Put**
  - Добавить новую запись в таблицу(если записи не было) или обновить(если запись была)
- **Scan**
  - Позволяет итерироваться по диапазону ключей
- **Delete**
  - Позволяет пометить запись как удаленную. Hbase не удаляет данные сразу, а ставит специальный маркер “могильный камень”(tombstone). Физическое удаление произойдет при следующем Major Compaction(см дальше).

# HBase shell

- Jruby среда, поддерживающая операции put, get, delete и scan

```
1. root@pdocker: ~ (ssh)
hbase(main):029:0> create 'users', {NAME => 'user_profile', VERSIONS => 5}, {NAME => 'user_posts', VERSIONS => 1231231231}
0 row(s) in 0.4140 seconds

=> Hbase::Table - users
hbase(main):030:0> put 'users', 'id1', 'user_profile:name', 'alexander'
0 row(s) in 0.0120 seconds

hbase(main):031:0> put 'users', 'id1', 'user_profile:second_name', 'alexander'
0 row(s) in 0.0060 seconds

hbase(main):032:0> get 'users', 'id1'
COLUMN                                CELL
user_profile:name                      timestamp=1458326489619, value=alexander
user_profile:second_name               timestamp=1458326493015, value=alexander
2 row(s) in 0.0100 seconds

hbase(main):033:0> put 'users', 'id1', 'user_profile:second_name', 'petrov'
0 row(s) in 0.0070 seconds

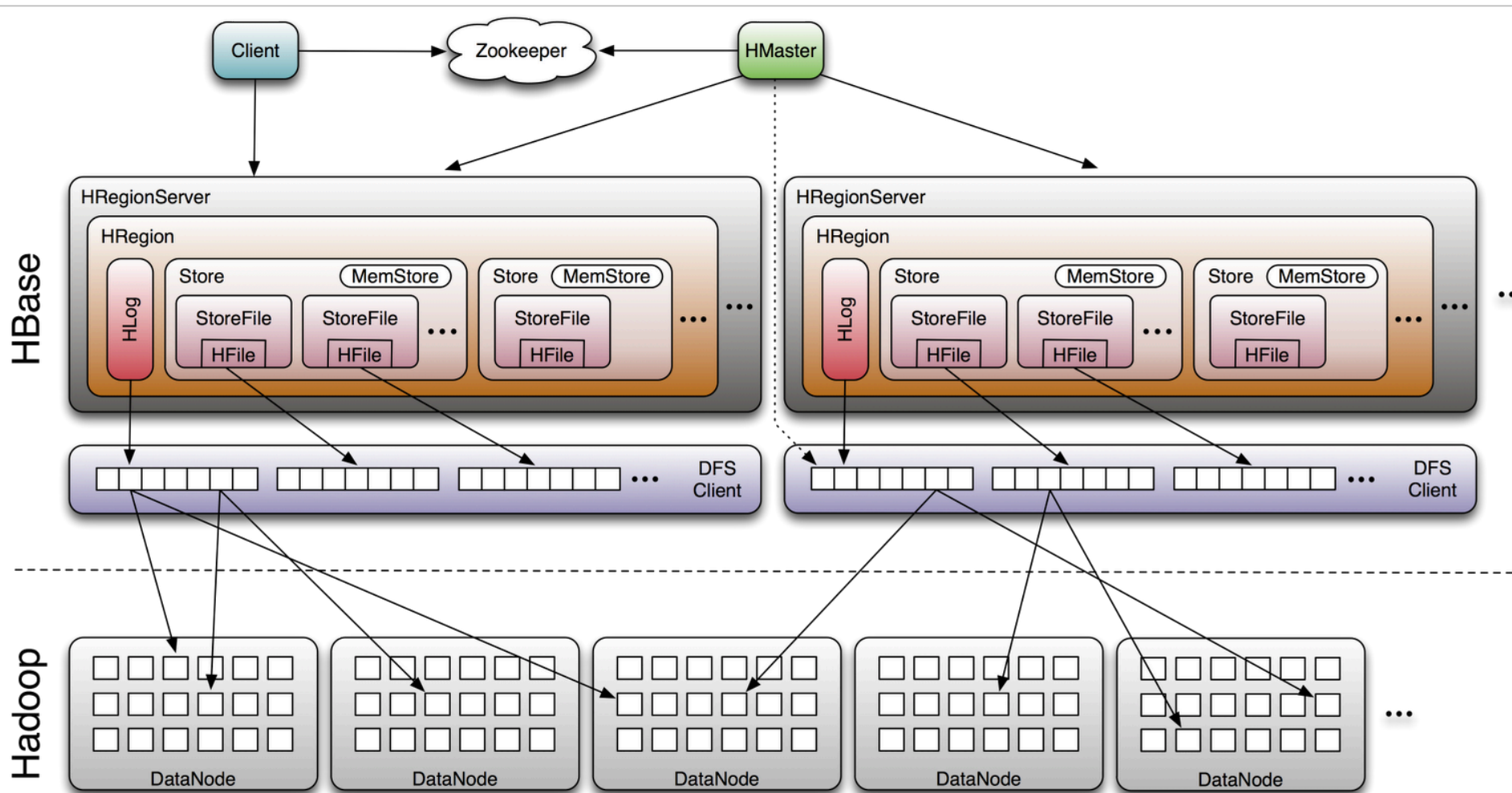
hbase(main):034:0> get 'users', 'id1'
COLUMN                                CELL
user_profile:name                      timestamp=1458326489619, value=alexander
user_profile:second_name               timestamp=1458326516165, value=petrov
2 row(s) in 0.0120 seconds

hbase(main):035:0> 
```

# Пример сессии в Hbase shell

- `create 'users', {NAME => 'user_profile', VERSIONS => 5}`
- `put 'users', 'id1', 'user_profile:name', 'alexander'`
- `put 'users', 'id1', 'user_profile:second_name', 'alexander'`
- `get 'users', 'id1'`
- `put 'users', 'id1', 'user_profile:second_name', 'petrov'`
- `get 'users', 'id1'`
- `get 'users', 'id1', {COLUMN => 'user_profile:second_name', VERSIONS => 5}`
- `put 'users', 'id2', 'user_profile:name', 'vasiliy'`
- `put 'users', 'id2', 'user_profile:second_name', 'ivanov'`
- `scan 'users', {COLUMN => 'user_profile:second_name', VERSIONS => 5}`
- `delete 'users', 'id1', 'user_profile:second_name'`
- `get 'users', 'id1'`

# Архитектура HBase

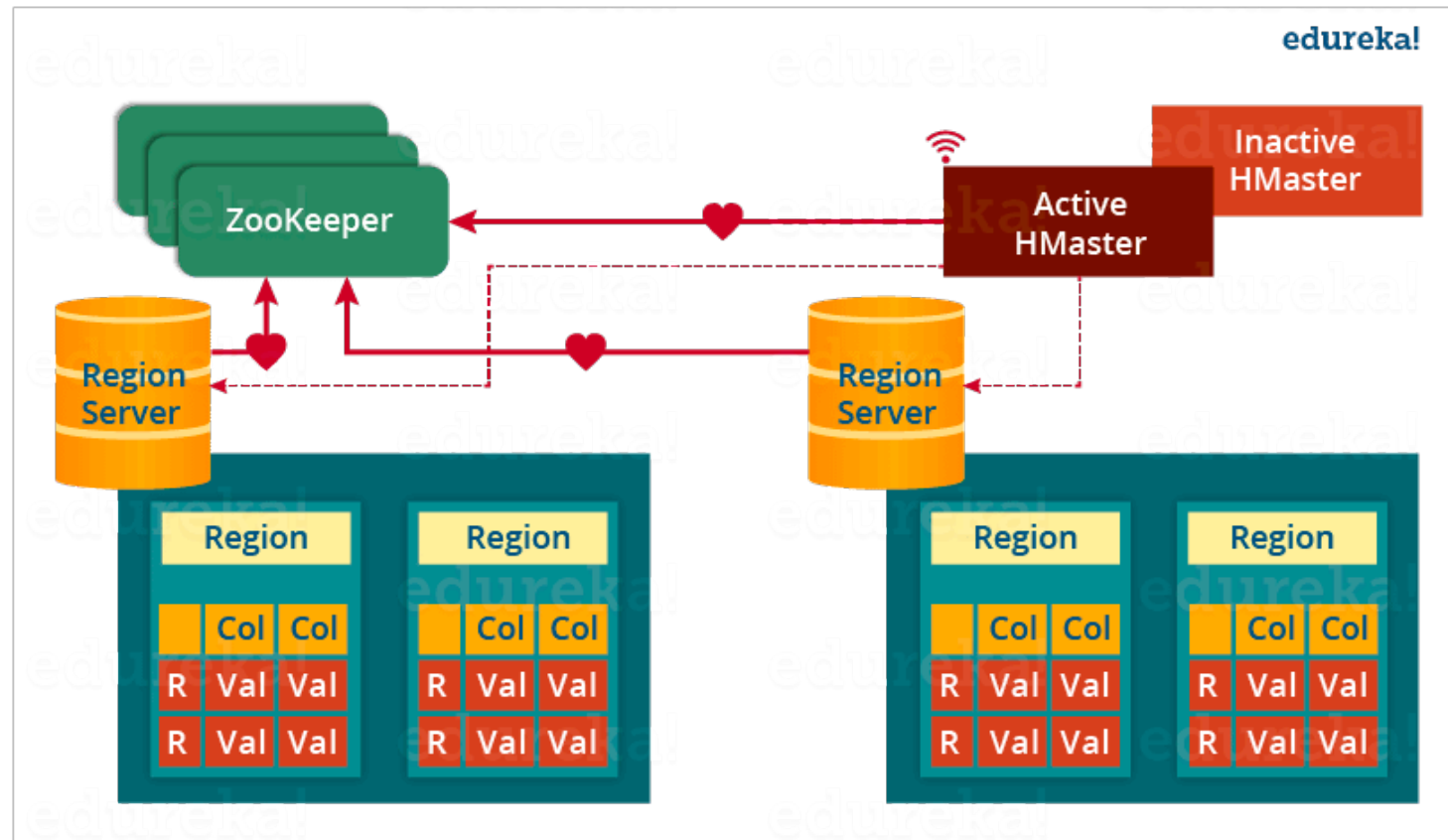


# Архитектура Hbase

- **ZooKeeper**

- Специальный сервис, используемый для координации сервисов. Представляет из себя очень "живучую" key-value базу данных, поддерживающая механизмы Pub/Sub
- Каждый Region Server и HMaster Server периодически отправляют heartbeat в Zookeeper и он проверяется статус. В случае потери инициирует сообщения о необходимости восстановления
- Активный HMaster отправляет сообщения в Zookeeper, inactive HMaster следит за активным. В случае падения сам становится активным.
- Если Region Server не отправляет уведомления HMaster запускает процесс восстановления.
- Zookeeper обслуживает путь до .META таблицы.

# Архитектура Hbase



# Архитектура Hbase

- **Master Server**

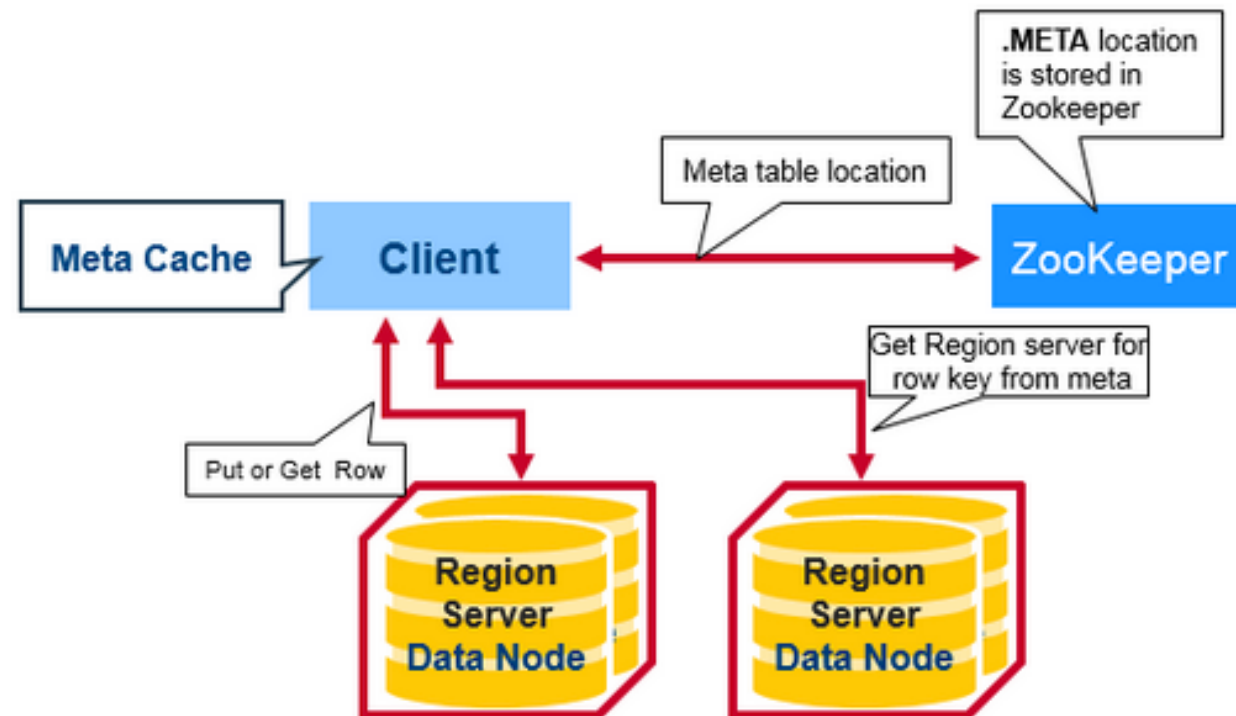
- Главный процесс в hbase, ведет реестр всех активных регионов
- Обслуживает DDL операции (create and delete tables), распределяет регионы по регион серверам.
- Управляет и координирует работу Region Server (как NameNode управляет DataNode в HDFS).
- Назначает Region в Region Servers при восстановлении и load balancing.
- Мониторит Region Servers (используя Zookeeper) и производит восстановление Region Server в случае падения.

# Архитектура Hbase

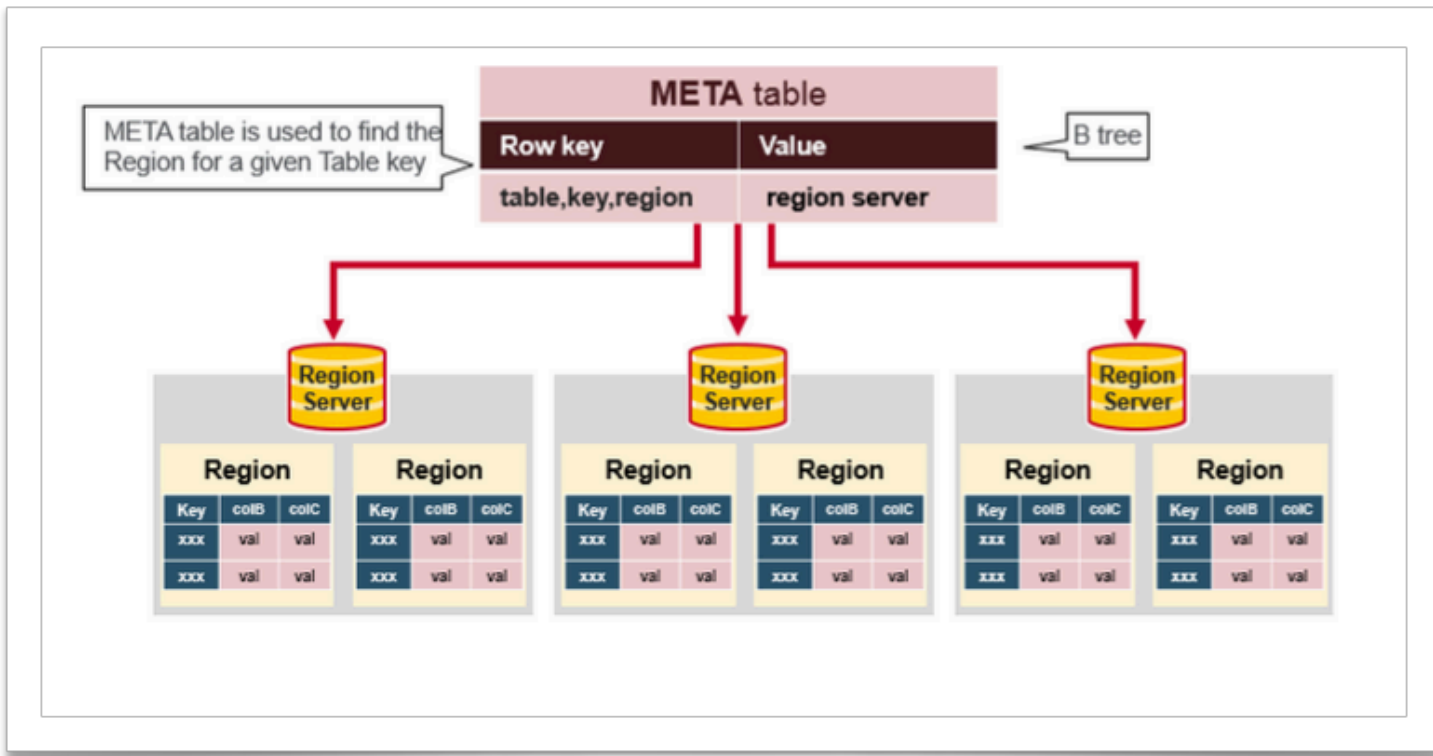
- **Region server**
  - Обслуживает один или несколько Region
- **Region**
  - Отсортированная часть строк таблицы
- **Wal (HLog)**
  - Журнал записей. Любая операция должна быть записана в WAL перед записью на диск
  - Append only
- **MemStore**
  - Кэш на запись. В MemStore хранятся которые еще не были записаны на диск (HFile).
  - Записи сортируются перед записью на диск.
  - 1 MemStore приходится на пару (Column Family, Region)
- **BlockCache**
  - LRU кэш на чтение. В нем хранятся часто используемые данные.
- **HFile**
  - Свой формат хранения
  - Отсортированный по RowKey набор значений в определенной ColumnFamily (дамп MemStore)



# Чтение



# .META.



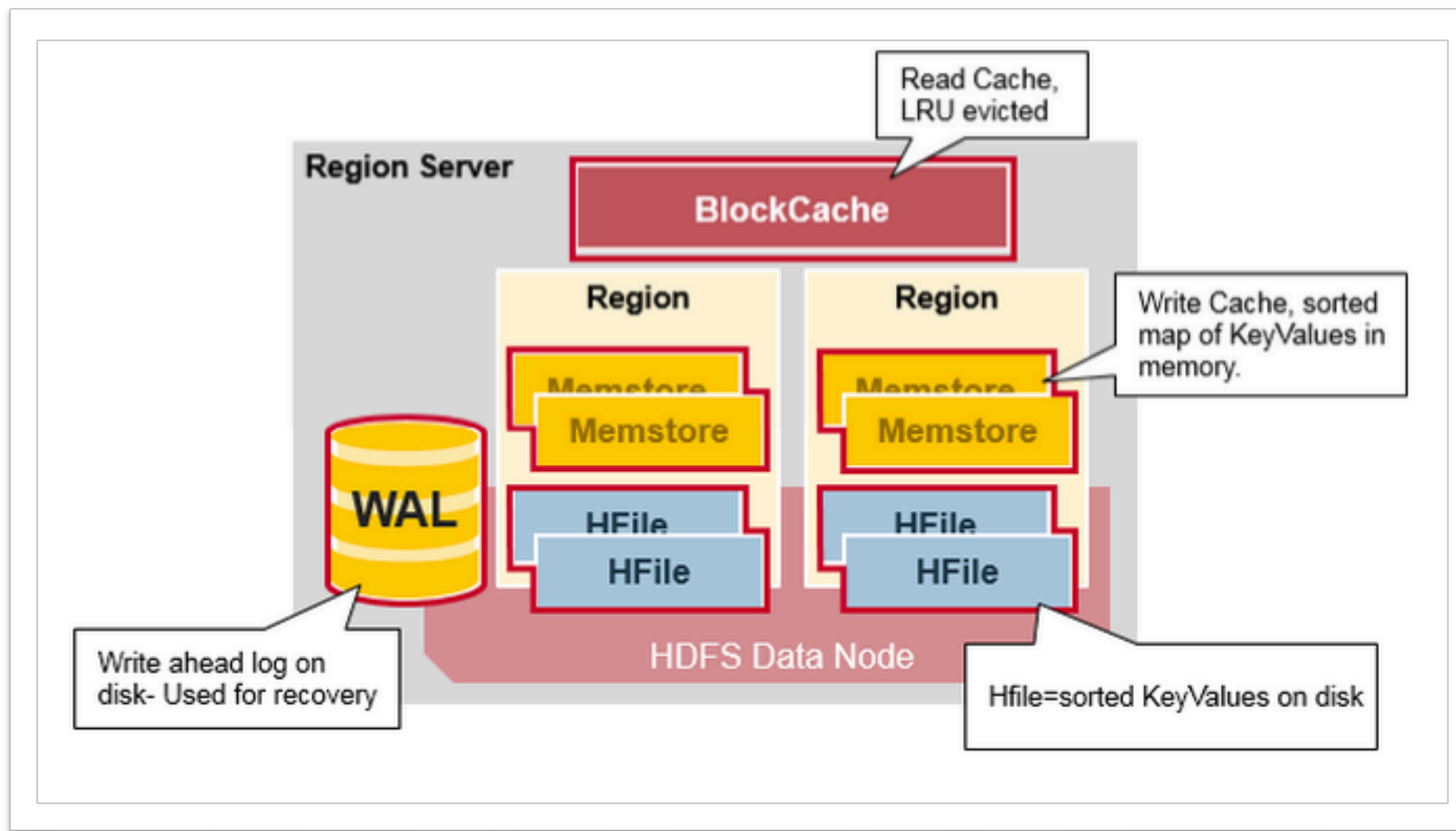
## Key

- ([table],[region start key],[region id])

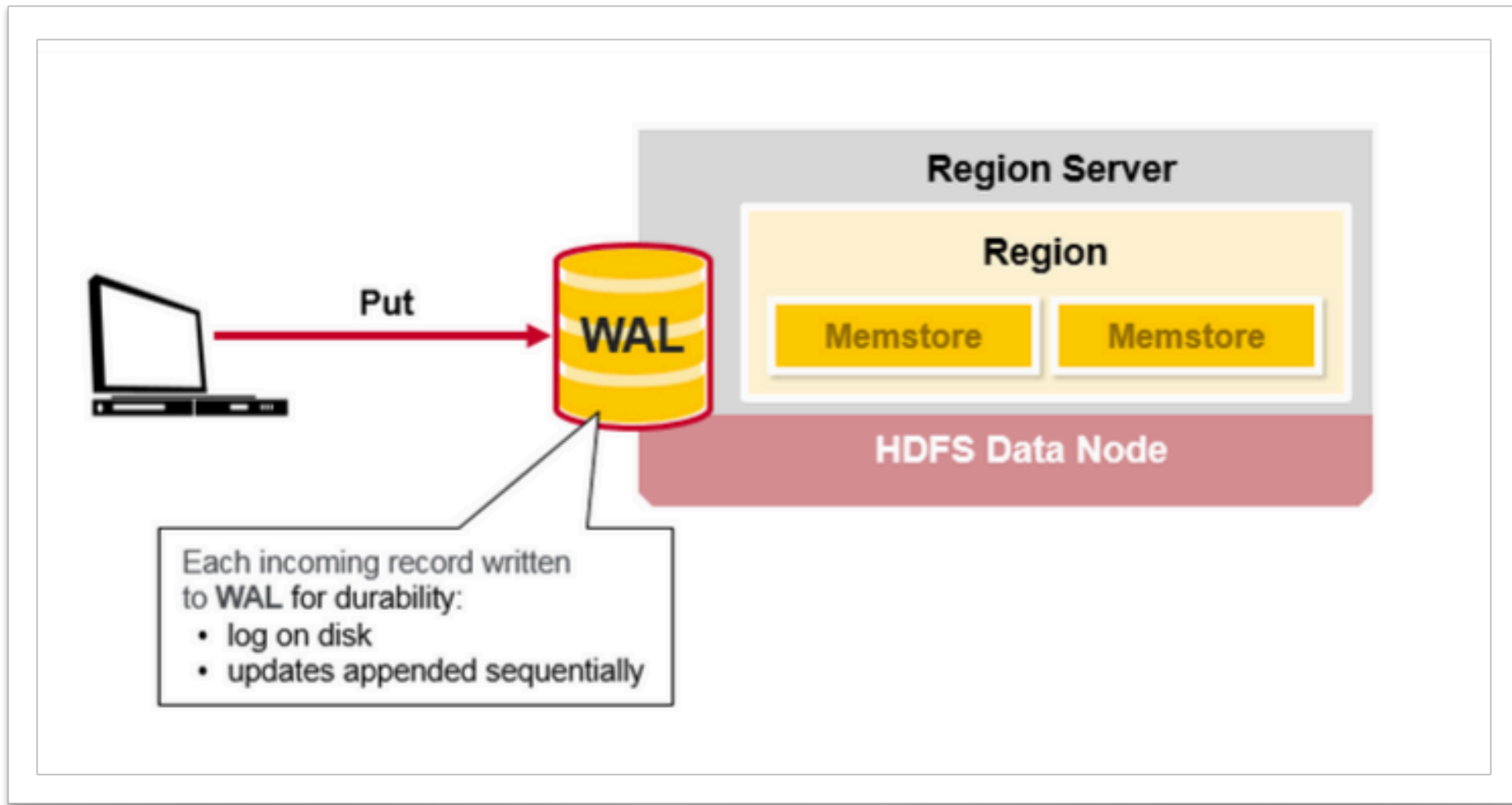
## Values

- info:regioninfo
- info:server (server:port RegionServer который обслуживает регион)
- info:serverstartcode (время запуска)

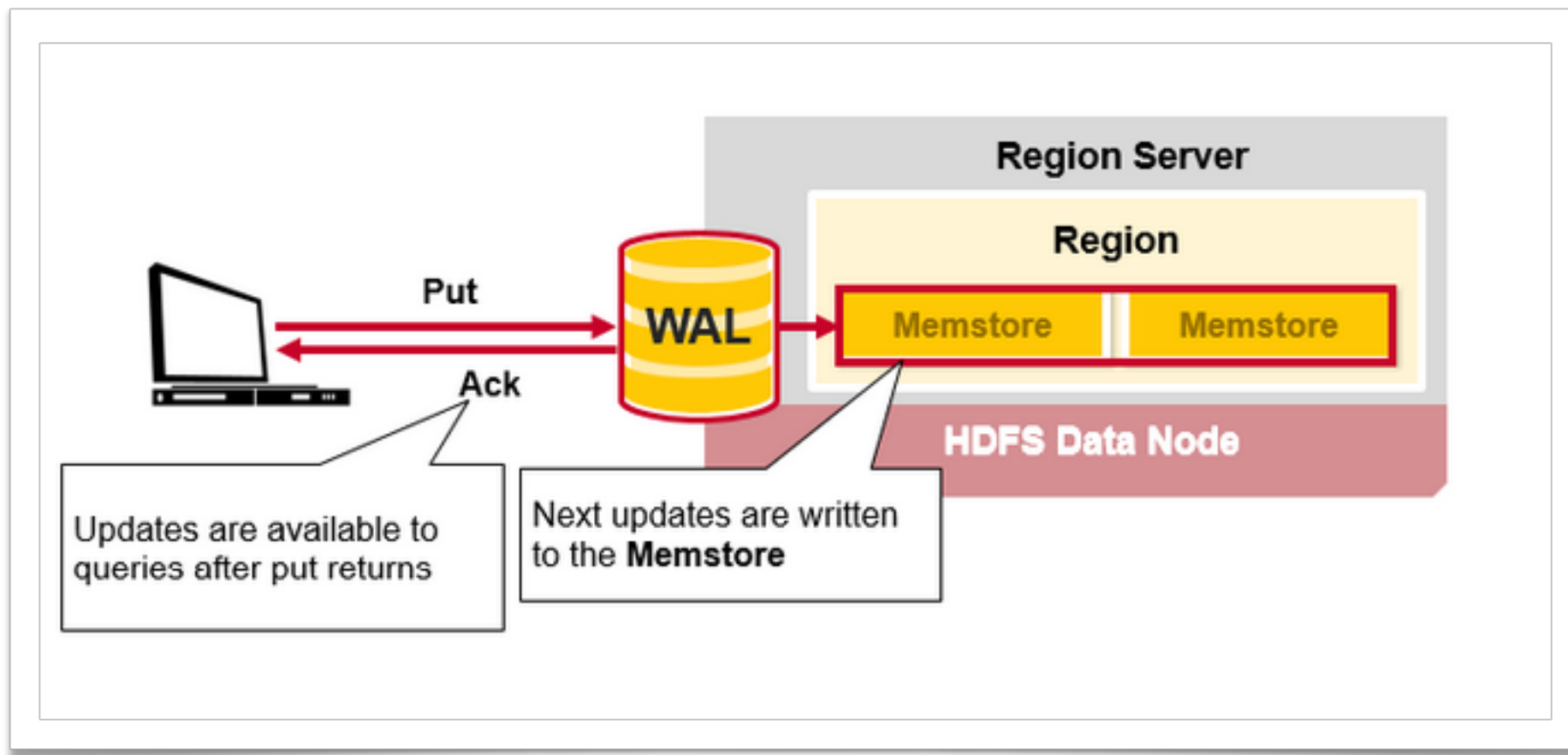
# Region Server



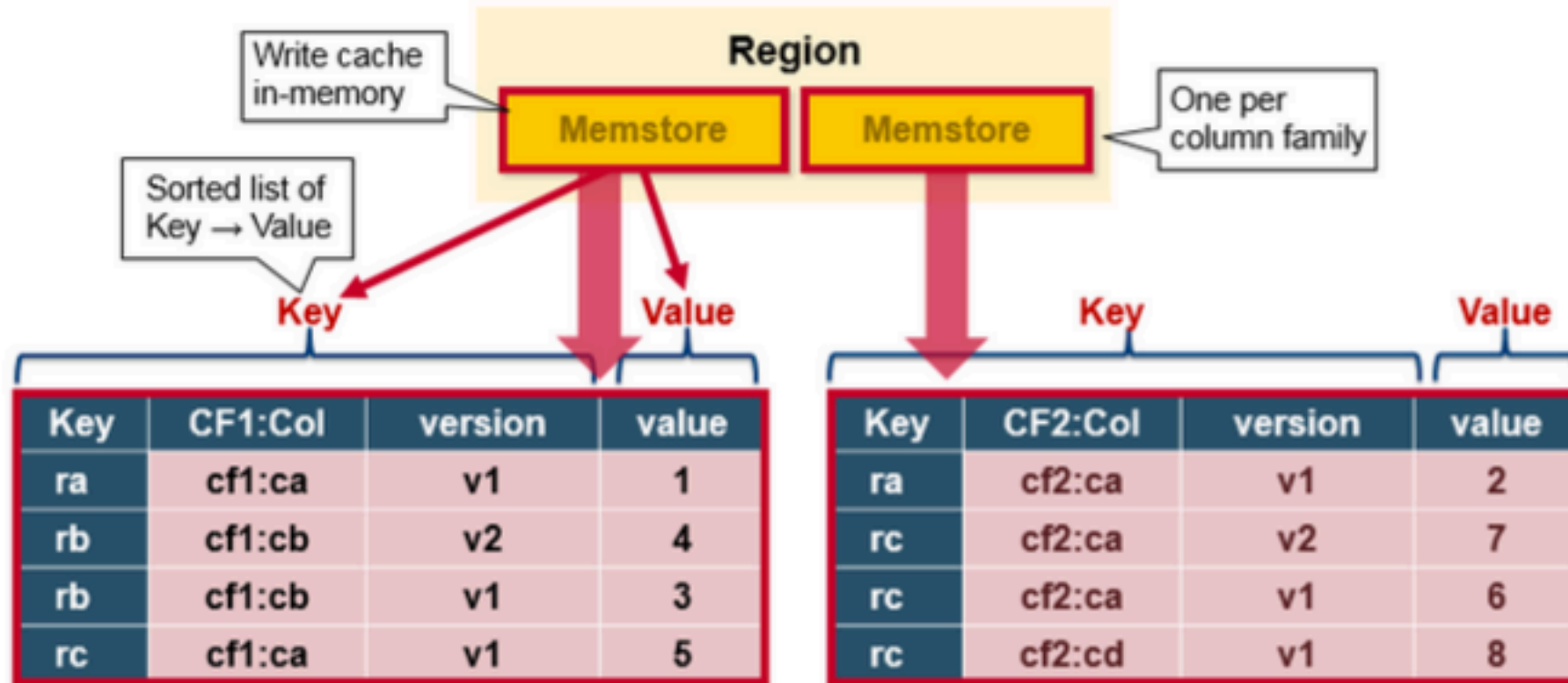
# Запись (шаг 1)



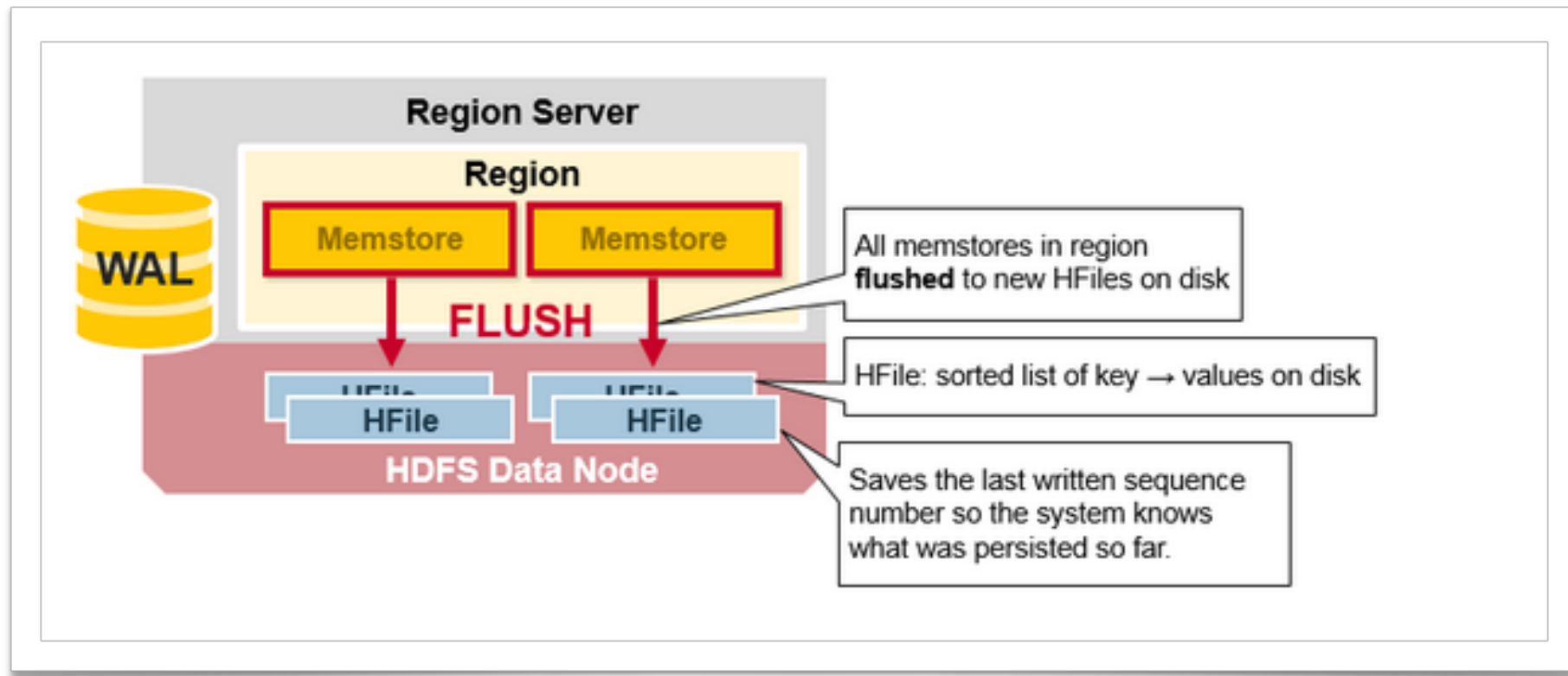
## Запись (шаг 2)



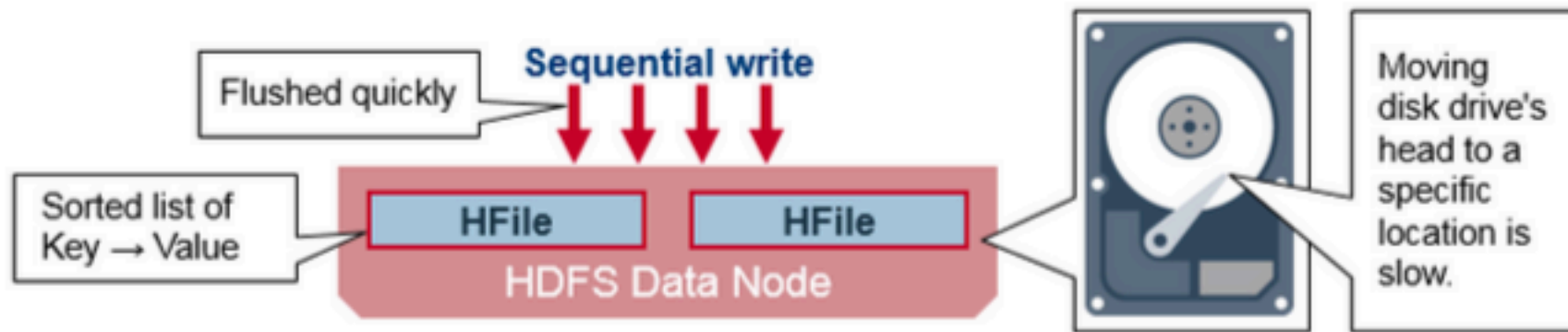
# MemStore



# Server Flush



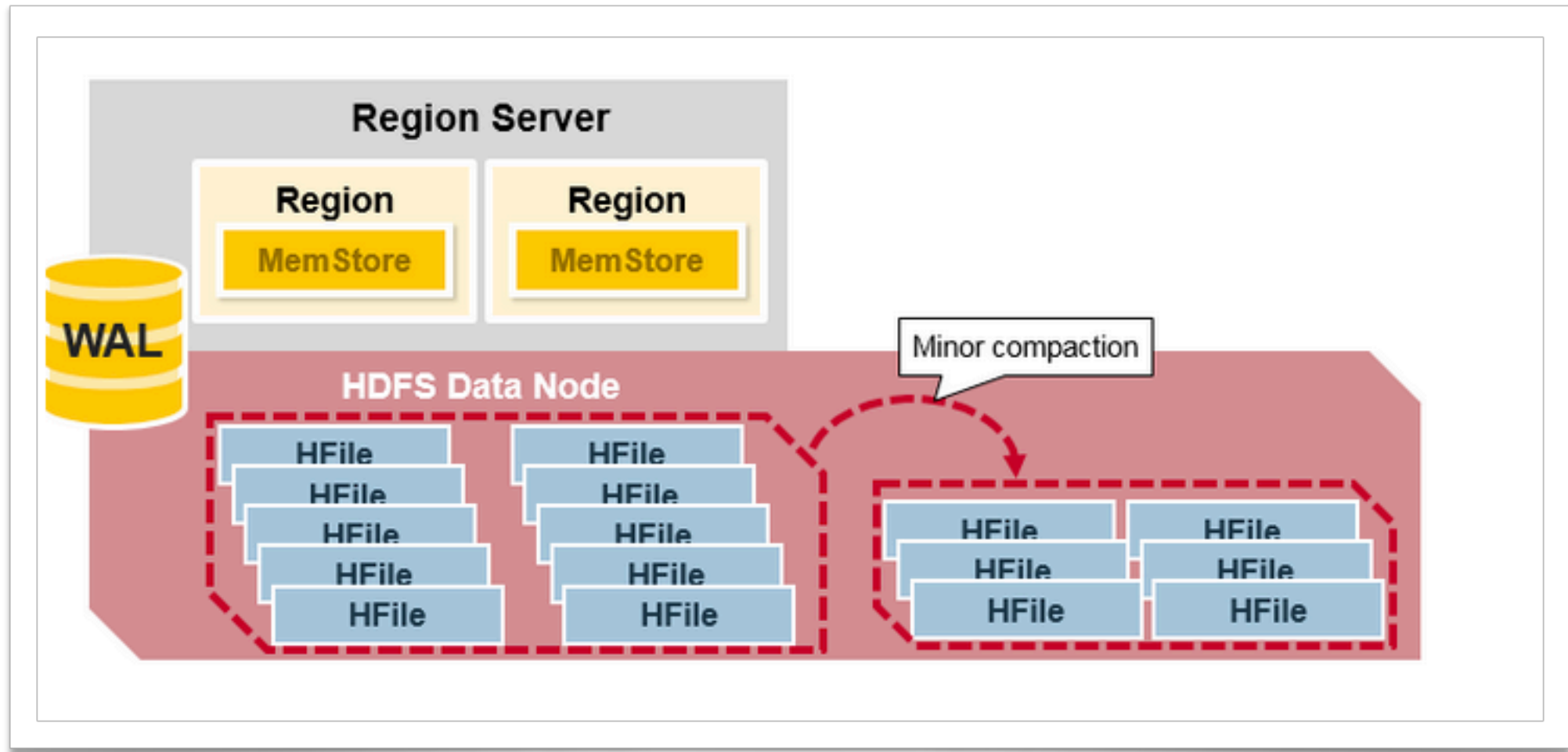
# HFile



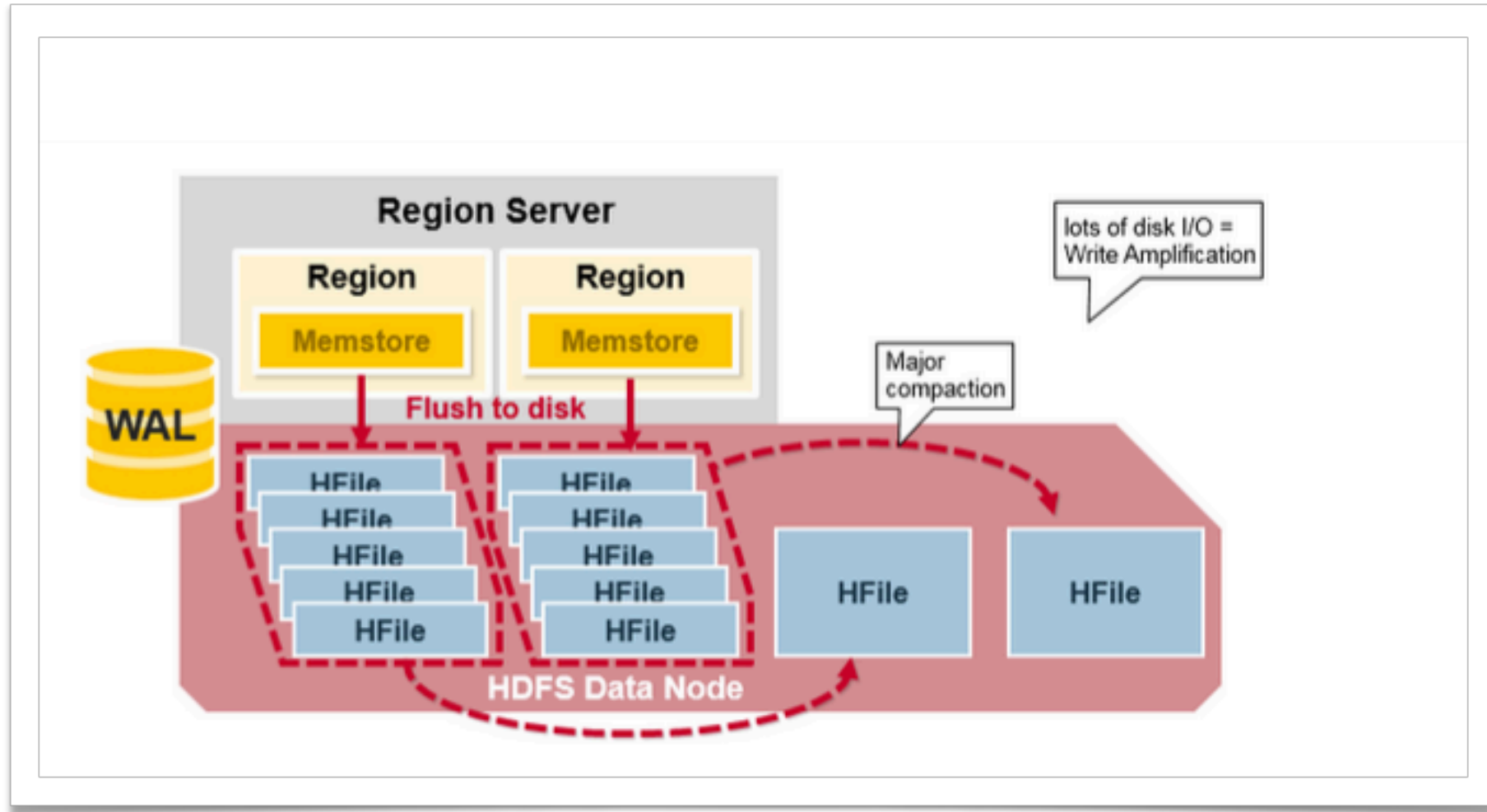
Key				Value			
Key	CF1:Col	version	value	Key	CF2:Col	version	value
ra	cf1:ca	v1	1	ra	cf2:ca	v1	2
rb	cf1:cb	v2	4	rc	cf2:ca	v2	7
rb	cf1:cb	v1	3	rc	cf2:ca	v1	6
rc	cf1:ca	v1	5	rc	cf2:cd	v1	8




# Minor compaction



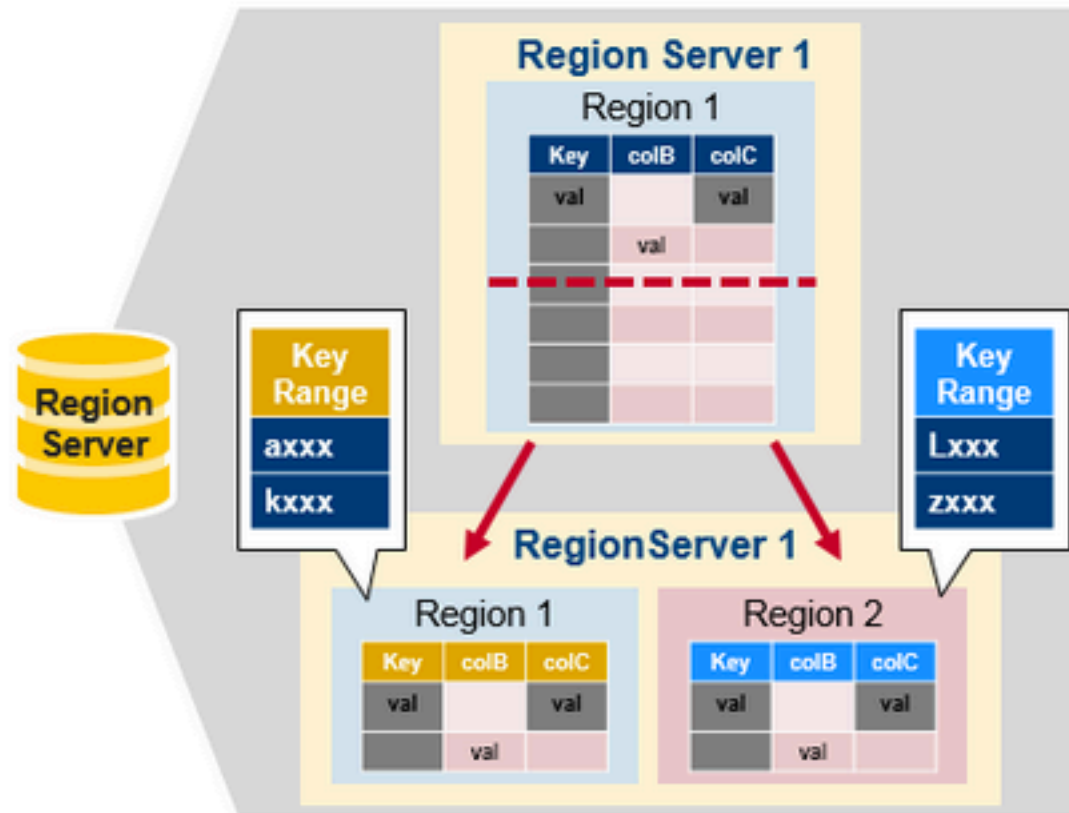
# Major compaction



# Compactions

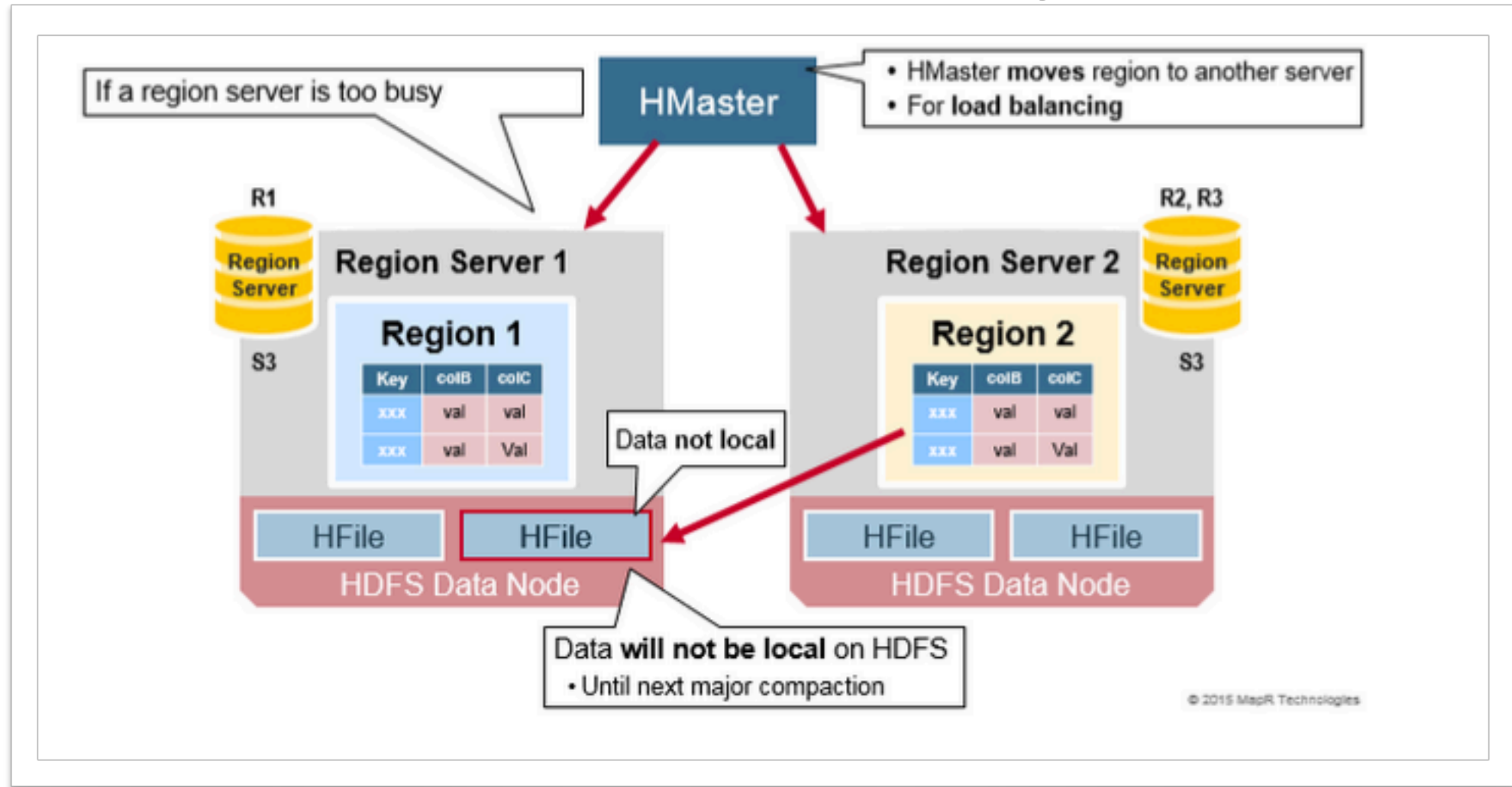
- Compaction – механизм слияния данных в hbase
  - Много маленьких HFile сливаются в один большой
- Minor Compaction
  - Происходят постоянно, автоматически, в фоне
  - Почти не снижают производительность
  - Не удаляют записи, только сливают несколько маленьких HFile в один большой
- Major Compaction
  - Запускаются вручную(можно настроить автозапуск по расписанию)
  - Снижение производительности в угоду слияния
- Борьба с compaction – то с чем сталкивается любой администратор HBase  


# Region split

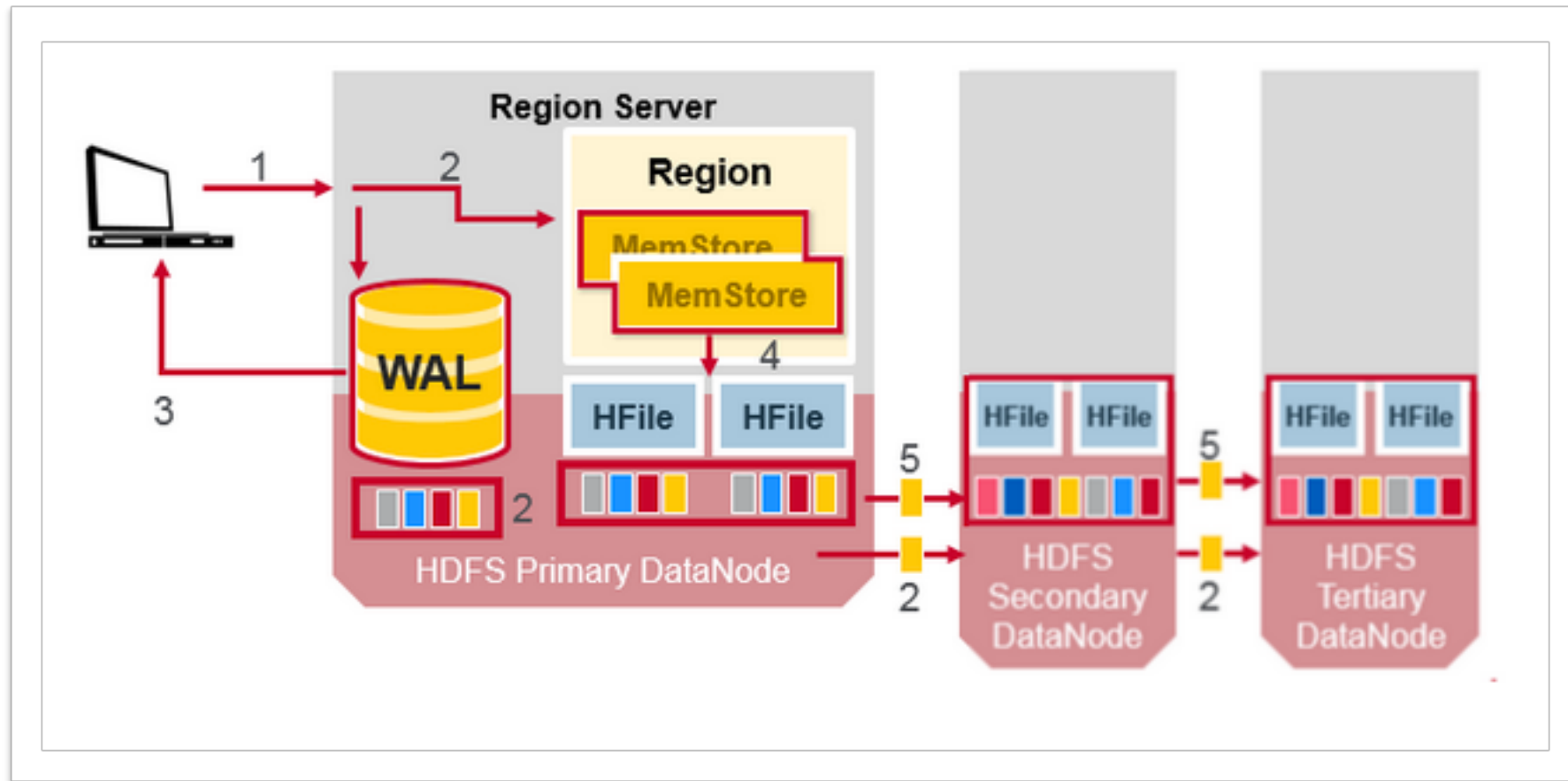


when region size >  
hbase.hregion.max.  
filesize → split

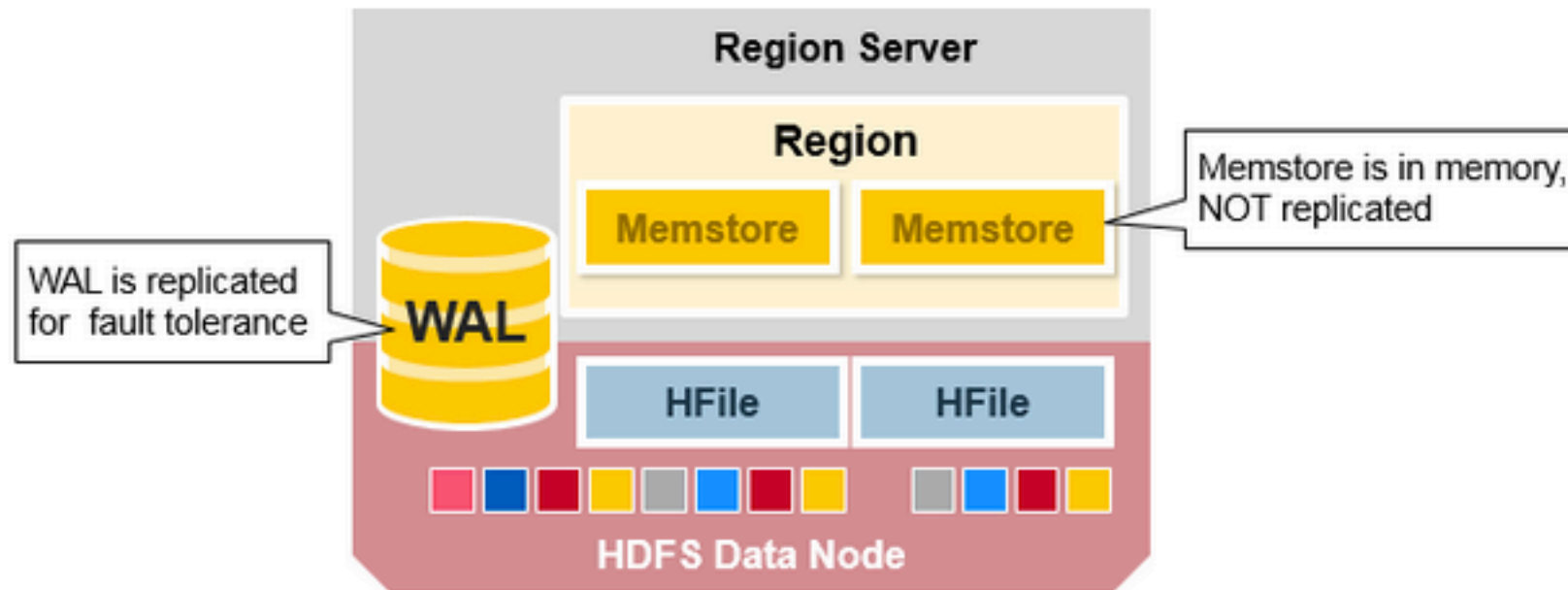
# Read balancing



# Репликация

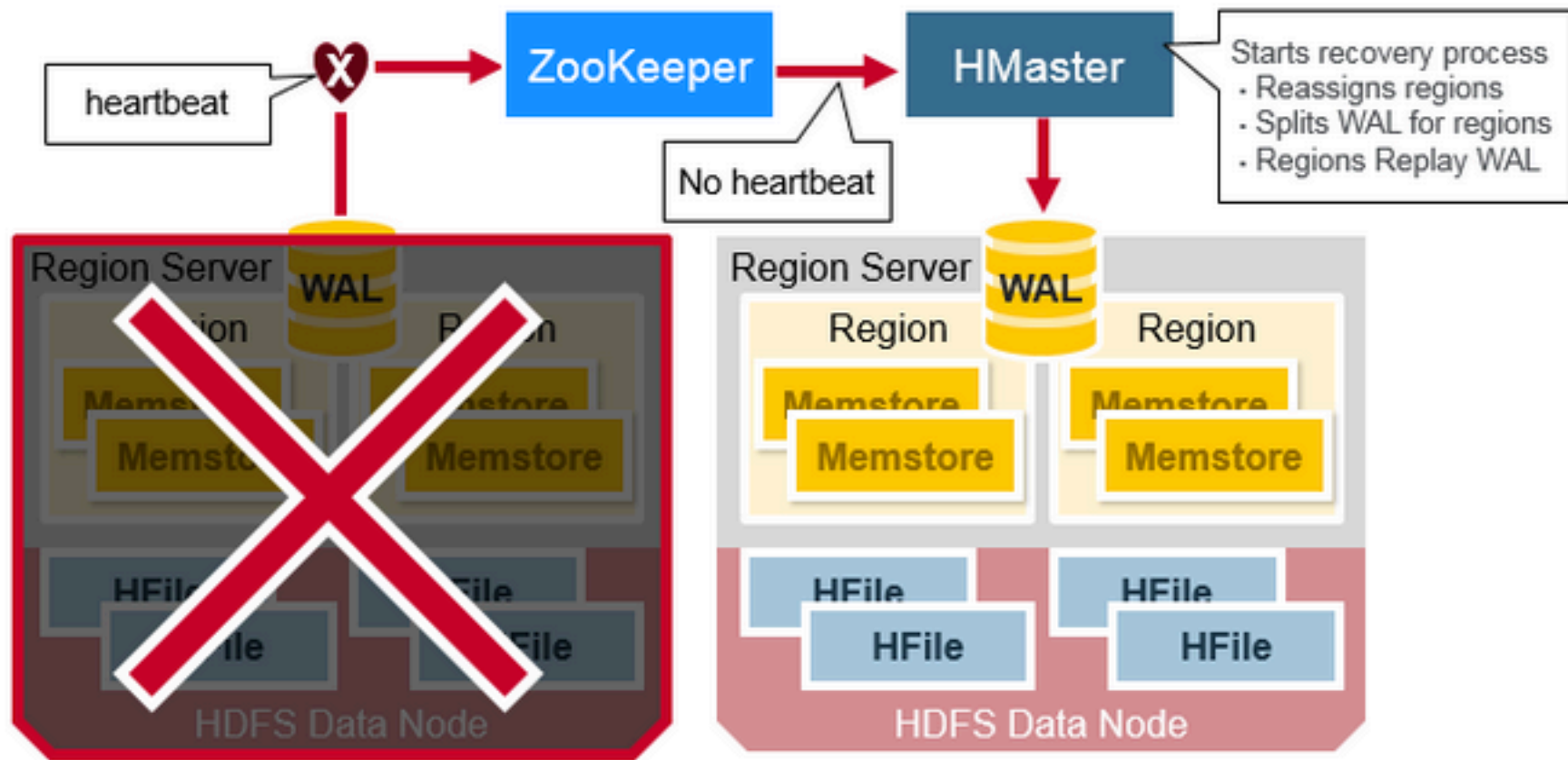


# Репликация



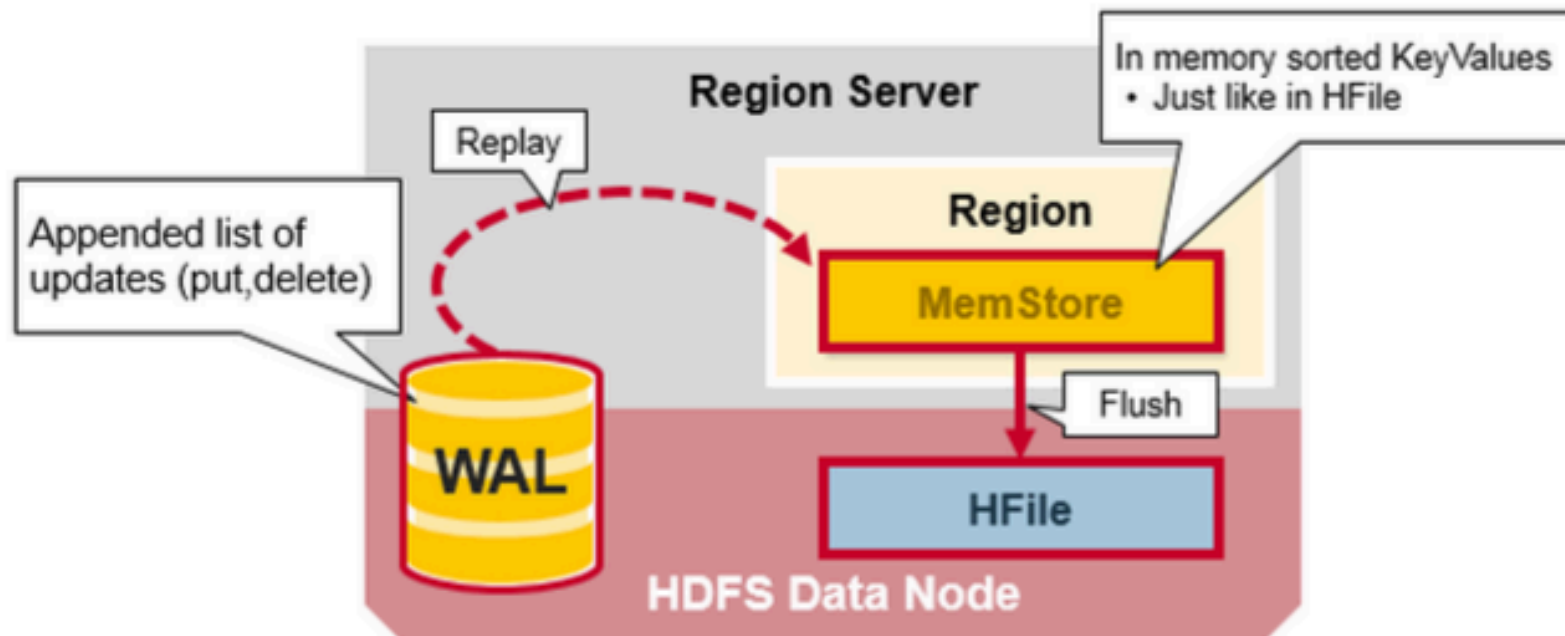
**How does HBase recover updates not persisted to HFiles?**

# Восстановление





# Восстановление



# Особенности HBase

- При разрастании регион автоматически разделяется
- Колоночная структура – данные можно читать независимо из разных колонок
- При записи данных с MemStore на HDFS формируется новый HFile
- Hbase оптимизирован в первую очередь под Troughput (в жертву latency)
- Параметры хранения данных(максимальное количество версий, алгоритм сжатия) задается на уровне **ColumnFamily**
- в hbase хранится специальная таблица 'meta', которая содержит информацию о всех блоках. Информация о расположении таблицы 'meta' хранится в zookeeper
- Мультиверсионность и TTL записей

# Важность RowKey

- **RowKey** выбирается один раз при заведении таблицы
- **RowKey** должен быть равномерным.
- При неправильном выборе RowKey может возникнуть "Hot Region" – регион нагрузка на который на порядок выше чем на другие регионы
- Очень сложно поменять RowKey в таблице с сотнями терабайт данных

# Примеры HBase

- **DMP:**
  - RowKey – User ID (UUID)
  - 2 column family:
    - Data: Количество версий – 2000, TTL 2 месяца
      - Data:urls – визиты пользователя
      - Data:sq – поисковые запросы
      - Data:ip – IP адрес...
    - LongData: Количество версий – 5, TTL 10 лет
      - LongData:Age
      - LongData:Bdate
      - LongData:Name

# Антипаттерн:

- Поисковый индекс
  - Ключ – домен в обратном порядке + URL (ru.yandex.www/index.html)
  - 1 Column Family
    - DATA:
      - Index
      - Text
      - HTML
      - ....
- Проблемы:
  - HotRegions (домены распределены не равномерно)
  - Все данные свалены ”в кучу” – часто нужна только малая часть данных

# Hbase и MapReduce

- Hbase “из коробки” имеет интеграцию с mapreduce
  - TableInputFormat
- Варианты использования Python.
  - Jython
  - Обвязка Java-кодом

# САР-теорема

- Целостность (Consistency)
- Доступность (Availability)
- Устойчивость к разделению (partition tolerance)

Выбери любые 2

HBASE принадлежит классу CP.

Разделение кластера HBASE на несколько частей может привести к некорректным ответам

# Альтернативы: реляционные СУБД

- Превосходят почти по всем параметрам если данных мало
- Очень плохо масштабируются



ORACLE®





# Альтернативы: Key-Value хранилища

- Быстрые
- Не обязательно консистентные
- Плохо с пакетной обработкой данных



# File-Based хранилища

- Текстовые файлы, parquet, sequence files
- Hive, pig – средства обработки
- Хорошо подходят для данных которые только добавляются
- Не подходят при обновляемых данных



# Cassandra и dynamodb

- Во многом похожа на Hbase
- Имеет column families
- SQL-like язык запросов



- Относится к классу AP (устойчива к расщеплению, не консистента)
  - Hbase относится к классу CP (не устойчива к расщеплению, консистента)

# Checklist Hbase

- ❑ много небольших Key-Value
- ❑ random access
- ❑ составной value, доступ к подмножествам колонок
- ❑ случайное обновление небольшого подмножества значений
- ❑ TTL, versions, filters

