Cassandra

«Использование баз данных»

Цесько Вадим Александрович
https://incubos.org
@incubos

ТЕХНОПОЛИС

24 мая 2017 г.

Содержание

History

- July 2008 — open-sourced by Facebook[1]
- March 2010 — graduated from the Apache Incubator
- Influenced by Amazon Dynamo
- Committers: Rackspace, Digg, Twitter, Amazon, Microsoft

[1] Facebook. Cassandra — A Decentralized Structured Storage System: `http://www.cs.cornell.edu/projects/ladis2009/papers/lakshman-ladis2009.pdf`

Features

- Decentralized — no SPoF, every node is identical
- Multi data center replication
- Elastic Scalability
- High Availability and Fault Tolerance
- Tunable consistency
- CQL

Data Model

- Based on Google Bigtable
- Hybrid: Key-value + Column-oriented
- Каждая строка содержит множество колонок
- Колонка (ячейка): имя + значение
- Разные строки могут иметь разный набор колонок
- Группировка колонок в семейства и суперсемейства

Table

A distributed multi dimensional map indexed by a key.

ACID

- Никаких JOIN-ов и внешних ключей
- *Атомарность* на уровне строк, нет rollback
- Можно получить ошибку при записи, но запись состоится
- Отметки времени от клиента при конфликтах
- Настраиваемая *консистентность* (количество реплик)
- *Изоляция* на уровне строк
- *Durability*: commit log + replication

CAP

- Tunable Consistency
- Tunable Availability
- Partition Tolerance

Пользователи

Самое популярное колоночное хранилище[2]:

- Cisco
- CERN
- Digg
- Facebook
- IBM
- Netflix
- Reddit
- SoundCloud
- Twitter
- Odnoklassniki

[2] http://db-engines.com/en/ranking/wide+column+store

Мотивация

If I had asked people what they wanted, they would have said faster horses.

Henry Ford

Что не так с RDBMS?

Chapter 1. "What's Wrong with Relational Databases?"in "Cassandra: The Definitive Guide"[a]

[a] Eben Hewitt. Cassandra: The Definitive Guide. 2010. ISBN 1449390412.

У нас есть сервис на RDBMS

Всё работает?

Не трогай!

С чем могут быть проблемы:

- (Distributed) Transactions[3]
- (Distributed) JOINs
- (Distributed) Schema evolution
- Sharding[4]
  - Functional segmentation
  - Key-based sharding
  - Lookup table

[3] Gregor Hohpe. Starbucks Does Not Use Two-Phase Commit: http://www.eaipatterns.com/ramblings/18_starbucks.html

[4] Michael Stonebraker. The Case for Shared Nothing. 1986

Use Case: RDBMS Scaling

LiveJournal

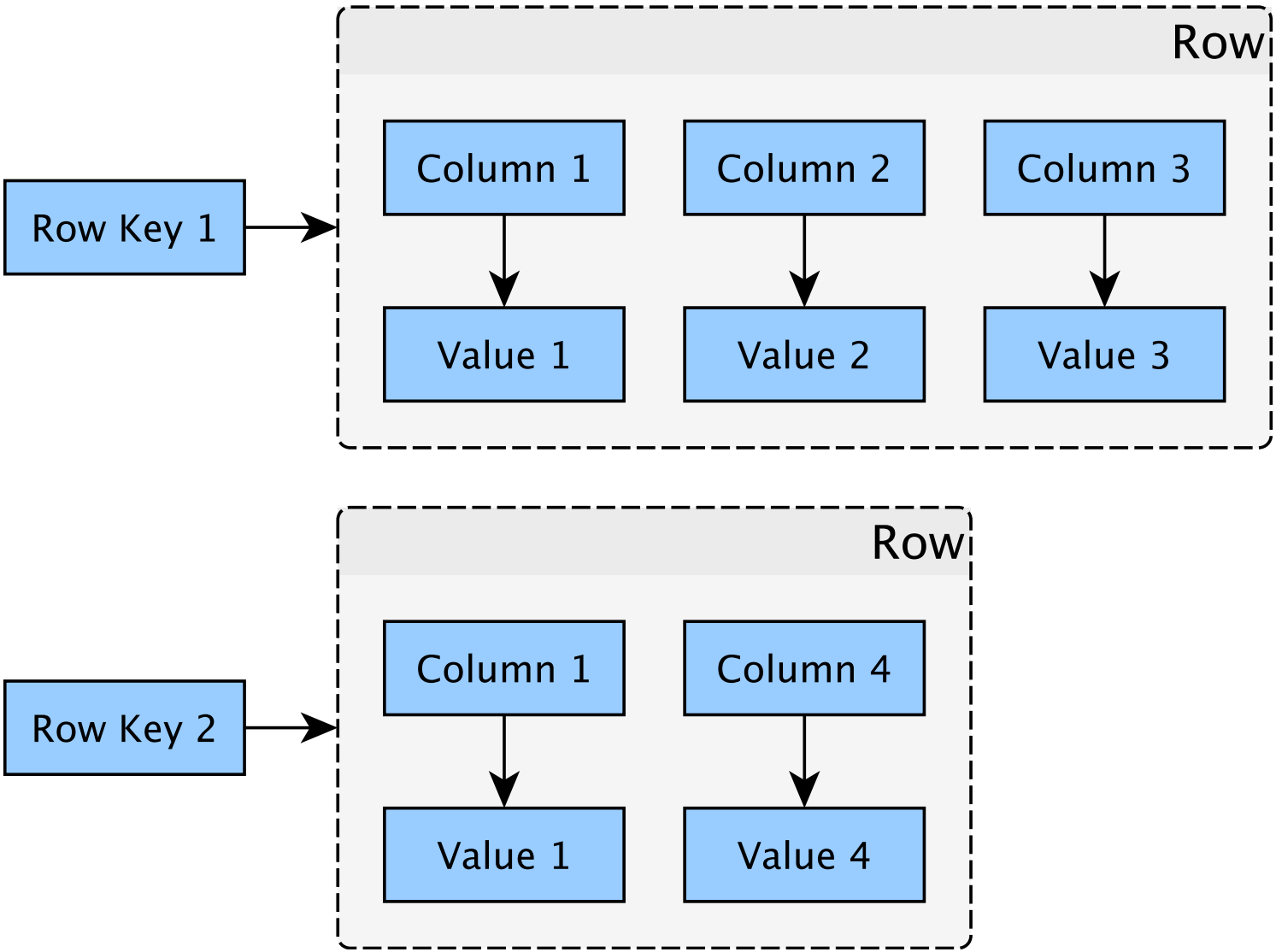
Brad Fitzpatrick. LiveJournal's Backend: A history of scaling[a]. August 2005.

[a] http://www.danga.com/words/2005_oscon/oscon-2005.pdf

Модель данных

- Cassandra — партиционированное хранилище рядов
- Исходим из запросов, а не из сущностей и связей
- Таблица — коллекция упорядоченных колонок
- Keyspace — группа таблиц (обычно per application)

Если глубже


Row
Row Key 1
Column 1
Column 2
Column 3
Value 1
Value 2
Value 3
Row
Row Key 2
Column 1
Column 4
Value 1
Value 4

Интерфейсы

- Thrift API
  - Исторически первый
  - Низкоуровневый
  - Довольно многословный
- **CQL**
  - SQL-подобный
  - (Почти насколько же) выразительный
  - Развивается

Пример

```
1 CREATE TABLE songs (
2   id uuid PRIMARY KEY,
3   title text,
4   album text,
5   artist text,
6   data blob
7  );
```

```
1 CREATE TABLE playlists (
2   id uuid,
3   song_order int,
4   song_id uuid,
5   title text,
6   album text,
7   artist text,
8   PRIMARY KEY (id, song_order));
```

Особенности

- Compound keys (clustering)
- Indices
- Collection types[5]: `set`, `list`, `map`
- `INSERT` = `UPDATE` = `UPSERT`
- `DELETE` для строки или набора ячеек
- TTL
- Counters
- См. спеку[6]

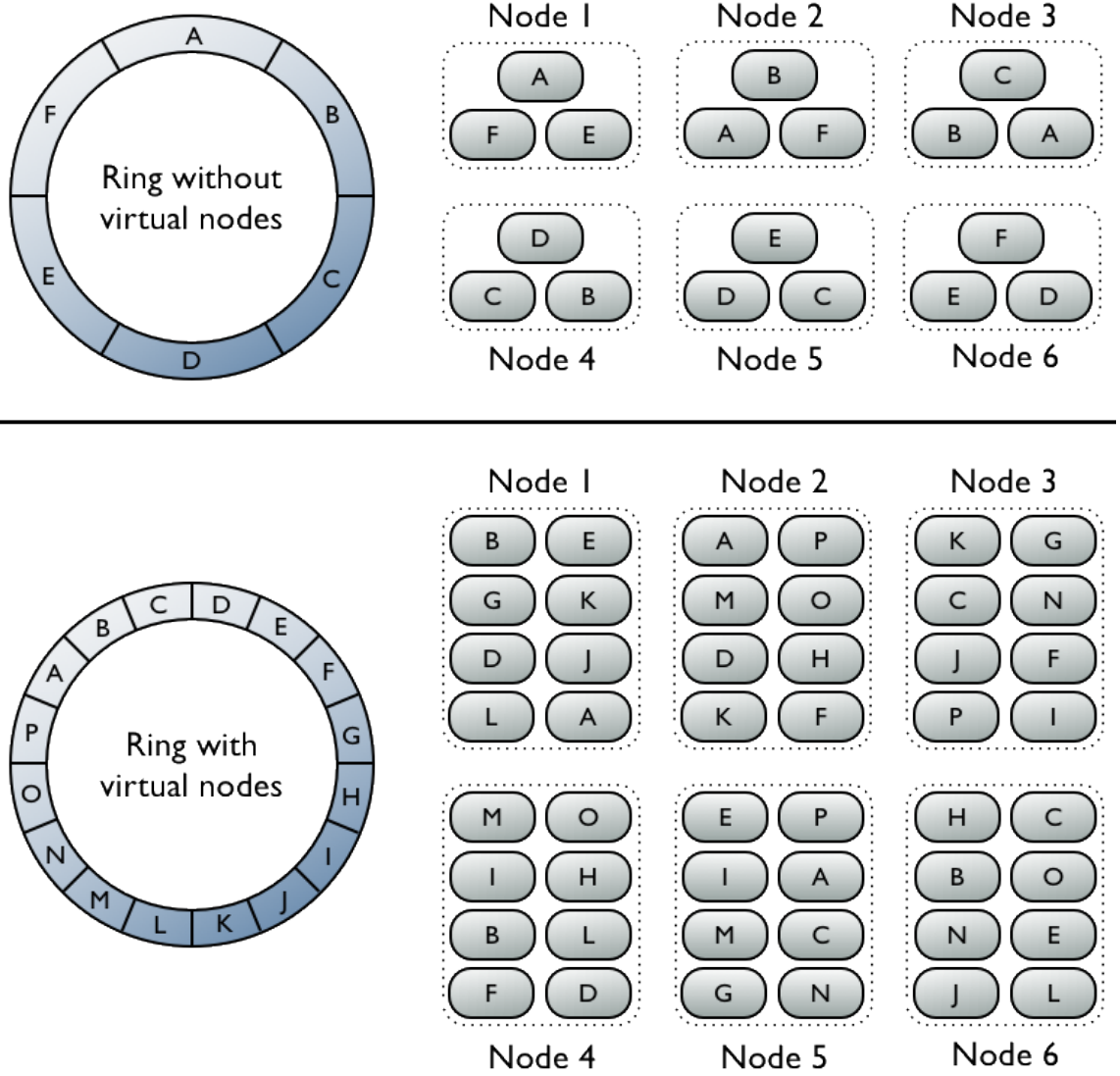
[5] `http://www.datastax.com/dev/blog/cql3_collections`

[6] `https://docs.datastax.com/en/dse/5.1/cql/index.html`

Компоненты

- **Consistent Hashing** + **VNodes** (см. лекцию 1)
- **Gossip** — состав и состояние узлов
- **Partitioner** — данные по узлам
- **Replica placement strategy** — реплики по узлам
- **Snitch** — топология (ДЦ и стойки)

Virtual Nodes



Gossip

- Узлы периодически обмениваются информацией о себе и других узлах
- Каждую секунду не больше чем с тремя узлами
- Все узлы быстро узнают информацио обо всех узлах
- Сообщения имеют версию — устаревшая информация затирается
- Seed nodes для bootstrap
- Gossip-информация хранится на каждой ноде персистентно
- Обнаружение сбоев узлов (+ dynamic snitch)[7]

[7] Naohiro Hayashibara, Xavier Défago, Rami Yared and Takuya Katayama. The ϕ Accrual Failure Detector. 2008

Partitioner

- Consistent Hashing[8]
- `Murmur3Partitioner` (-2^{63} to $+2^{63}$)
- `ByteOrderedPartitioner`
  - Лексикографически по байтам ключа
  - Сложная балансировка нагрузки (расчёт диапазонов партиций вручную)
  - Неравномерная балансировка для разных таблиц
  - Последовательная запись упирается в один узел

[8] `http://docs.datastax.com/en/dse/5.1/dse-arch/datastax_enterprise/dbArch/archDataDistributeHashing.html`

Replica Placement Strategy

- Replication Factor — количество реплик на кластер
- Каждая реплика на отдельном узле
- Все реплики равноправны
- `SimpleStrategy`:
  - 1 ДЦ (если планируете расширяться, не используйте)
  - Первая реплика на узел от Partitioner
  - Остальные — по часовой стрелке по кольцу
- `NetworkTopologyStrategy`:
  - Определяет количество реплик в каждом ДЦ
  - Узлы для реплик — идём по кольцу до следующей стойки
  - Часто стойки вырубаются целиком (питание, охлаждение, сеть, ...)

NetworkTopologyStrategy

Факторы

- Желательно чтение внутри ДЦ
- Учитываем типичные сбои

По 2 реплики в каждом ДЦ

Сбой одного узла — `ConsistencyLevel.ONE`

По 3 реплики в каждом ДЦ

- Сбой одного узла — `ConsistencyLevel.LOCAL_QUORUM`
- Сбой двух узлов — `ConsistencyLevel.ONE`

Snitch

- **Одинаковая конфигурация** на узлах
- Dynamic snitching wrapper[9]
  - По умолчанию и только на чтение
  - Данные с самой быстрой реплики, с остальных — контрольные суммы
  - Статистика Read Latency и текущие задачи узлов
- `SimpleSnitch`: никаких ДЦ и стоек
- `RackInferringSnitch`: (`110.dc.rack.node`)
- `PropertyFileSnitch`: адреса в ДЦ/стойки в файле
- `GossipingPropertyFileSnitch`: локальный ДЦ и стойка в файле на каждом узле + Gossip

[9] `http://www.datastax.com/dev/blog/dynamic-snitching-in-cassandra-past-present-and-future`

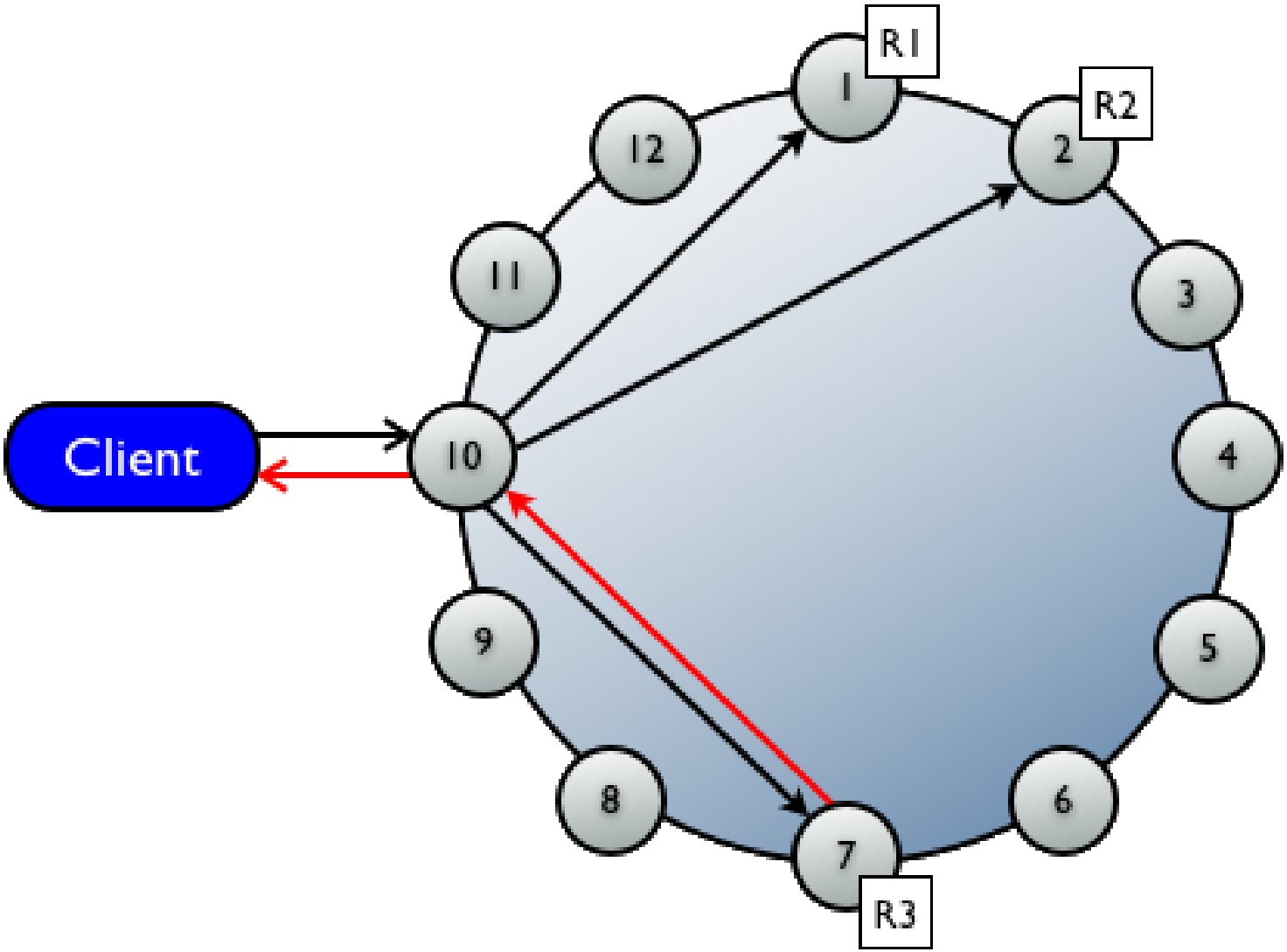
Клиентские запросы

- Можно обращаться к любому узлу
- Узел становится координатором для текущего запроса
- Координатор проксирует с учётом Partitioner и Replica Placement Strategy

Write

- Запрос всем репликам независимо от `ConsistencyLevel`
- `ConsistencyLevel` определяет, сколько реплик должно ответить для успеха

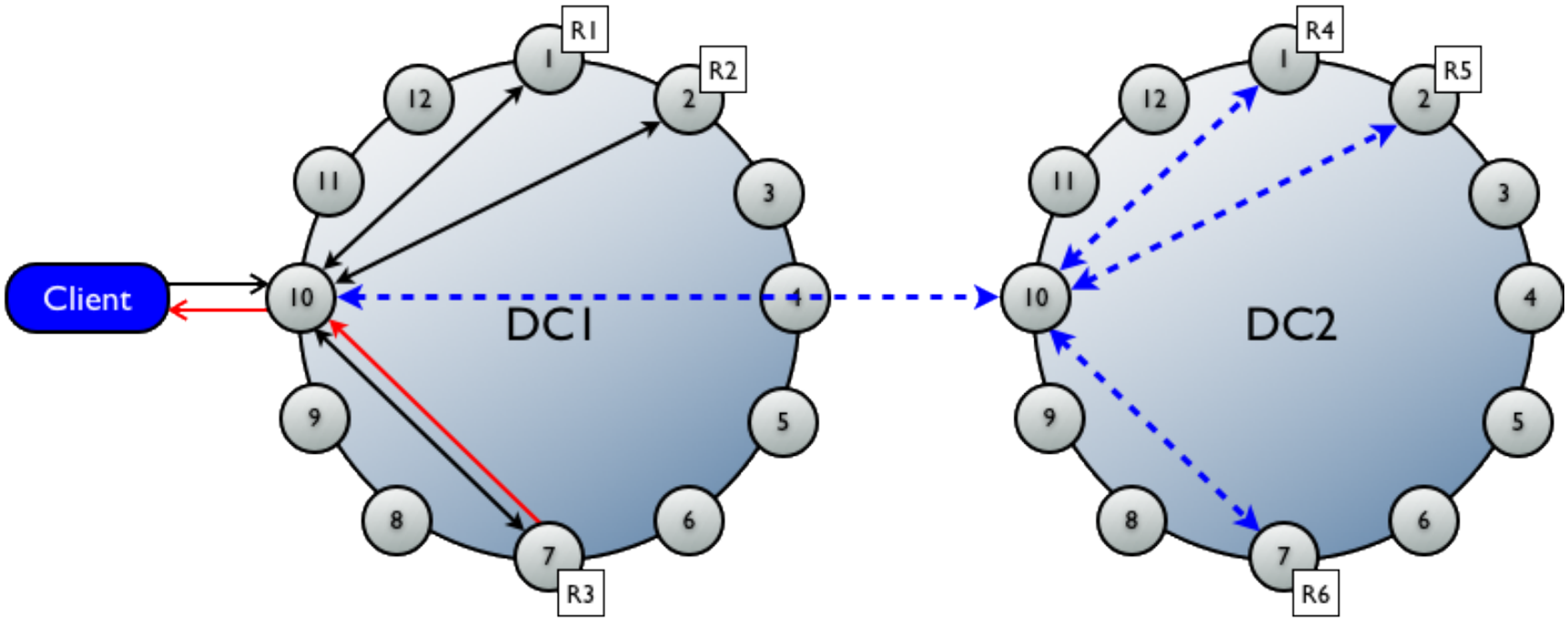
Write: Пример



MultiDC Write

- Оптимизация — один координатор в каждом удалённом ДЦ
- `ConsistencyLevel.LOCAL_ONE` или `ConsistencyLevel.LOCAL_QUORUM` — обязаны ответить только локальные узлы (география не влияет на задержку)

MultiDC Write: Пример



Write: ConsistencyLevel

- `ANY` — всегда успех (hinted handoff)
- `ONE` — в commit-log одного узла
- `TWO`
- `THREE`
- `QUORUM` — inter DC + нужен запас прочности
- `LOCAL_QUORUM` — быстрее, чем `QUORUM`
- `EACH_QUORUM` — выше консистентность
- `ALL` — WAT?

Не забываем

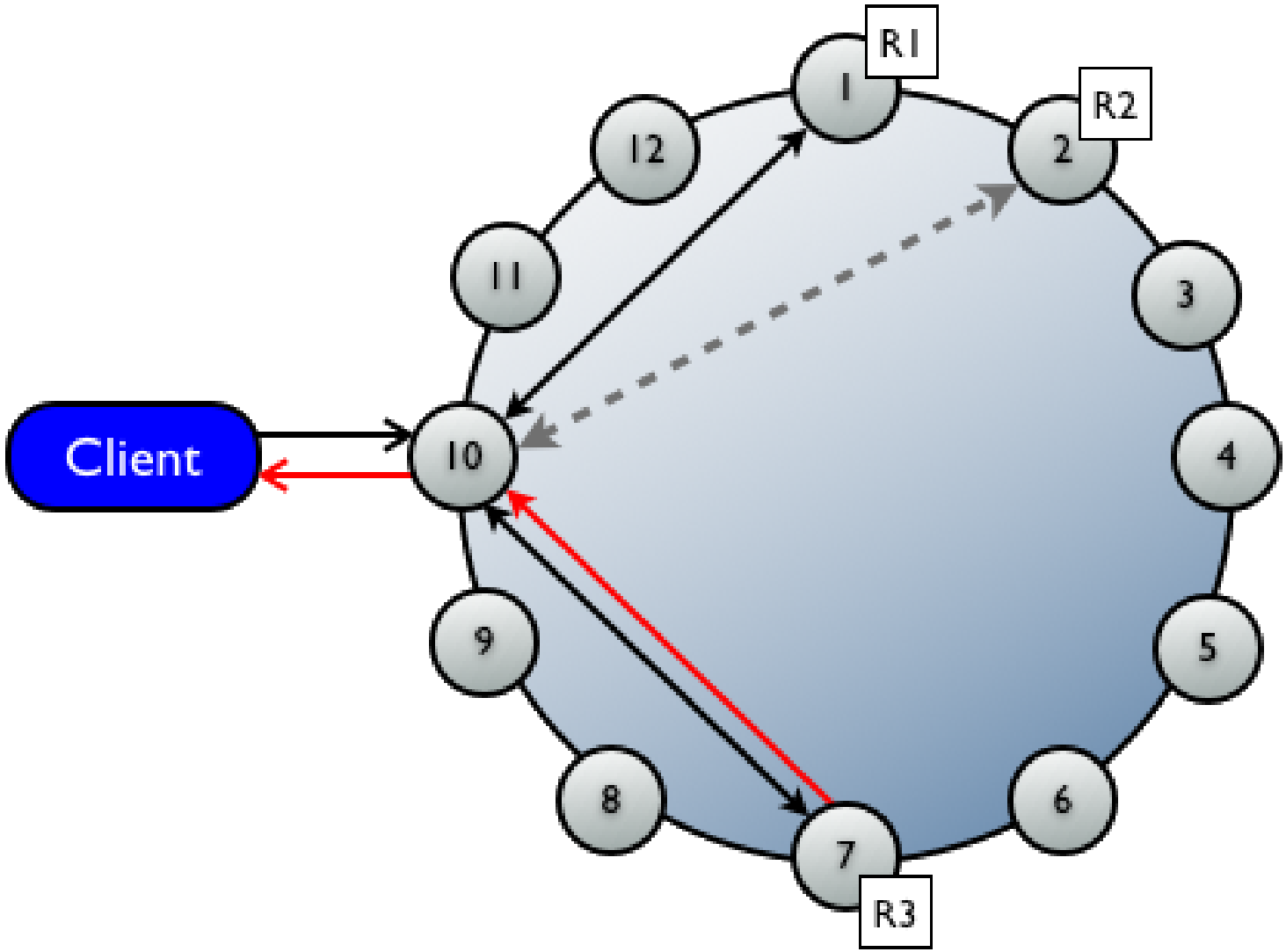
Даже при `ONE` и `LOCAL_QUORUM` запись пошлют всем репликам в т. ч. в других ДЦ

Read

Read-запросы от координатора репликам:

- Прямой запрос на чтение (в соответствии с `ConsistencyLevel`)
  - Самым «быстрым» репликам
  - Сравниваем ответы
  - Если не совпадают, то самая свежая (по timestamp) — клиенту
- Фоновый read repair (всем остальным репликам)
  - Для синхронизации «горячих» данных
  - `read_repair_chance` по умолчанию 0.1
  - Если не совпадают хэши, то перезаписываем

Read: Пример



Read ConsistencyLevel

- `ONE` — ближайшая реплика, возможно, устаревшие данные
- `TWO`
- `THREE`
- `QUORUM` — inter DC + нужен запас прочности
- `LOCAL_QUORUM` — быстрее, чем `QUORUM`
- `EACH_QUORUM` — выше консистентность
- `ALL` — WAT?

Интерпретация

Возвращаем самое свежее значение из требуемого множества реплик

Quorum

Quorum

$$Quorum = \lfloor \frac{RF}{2} + 1 \rfloor$$

- $RF = 2 \Rightarrow Quorum = 2$ — нет запаса
- $RF = 3 \Rightarrow Quorum = 2$ — в запасе 1 узел
- $RF = 4 \Rightarrow Quorum = 3$ — в запасе 1 узел
- $RF = 5 \Rightarrow Quorum = 3$ — в запасе 2 узла

Consistency

$$R + W > RF$$

- $R = 2 + W = 2 > RF = 3$ — в запасе 1 узел

Клиентские запросы

Проблема

- Узел сбойнул: железо или (чаще) сеть
- А мы хотим на него записать
- Что делать?

Hinted Handoff

- Пусть известно (Gossip), что узел лежит, или узел не отвечает
- Координатор запоминает hint локально
- Когда обнаружится, что узел поднялся (Gossip), ему перешлют накопленные hints

Hint

Содержимое

- Кому предназначается
- Значение ключа
- Данные

Внимание

- Hints хранятся ограниченное время (по умолчанию 3 часа)
- Официально рекомендуется периодический запуск `repair`

Ограничения

- Hint засчитывается не на всех уровнях консистентности[10]: 2 узла (1 мёртв) + RF=1 + `ConsistencyLevel.ONE`
- Но **всегда работает**, если не выключен
- `ConsistencyLevel.ANY` — extreme write availability
- Если умрёт машина с hints, то мы их потеряем

[10] `http://www.datastax.com/dev/blog/modern-hinted-handoff`

Anti-entropy

Проблема

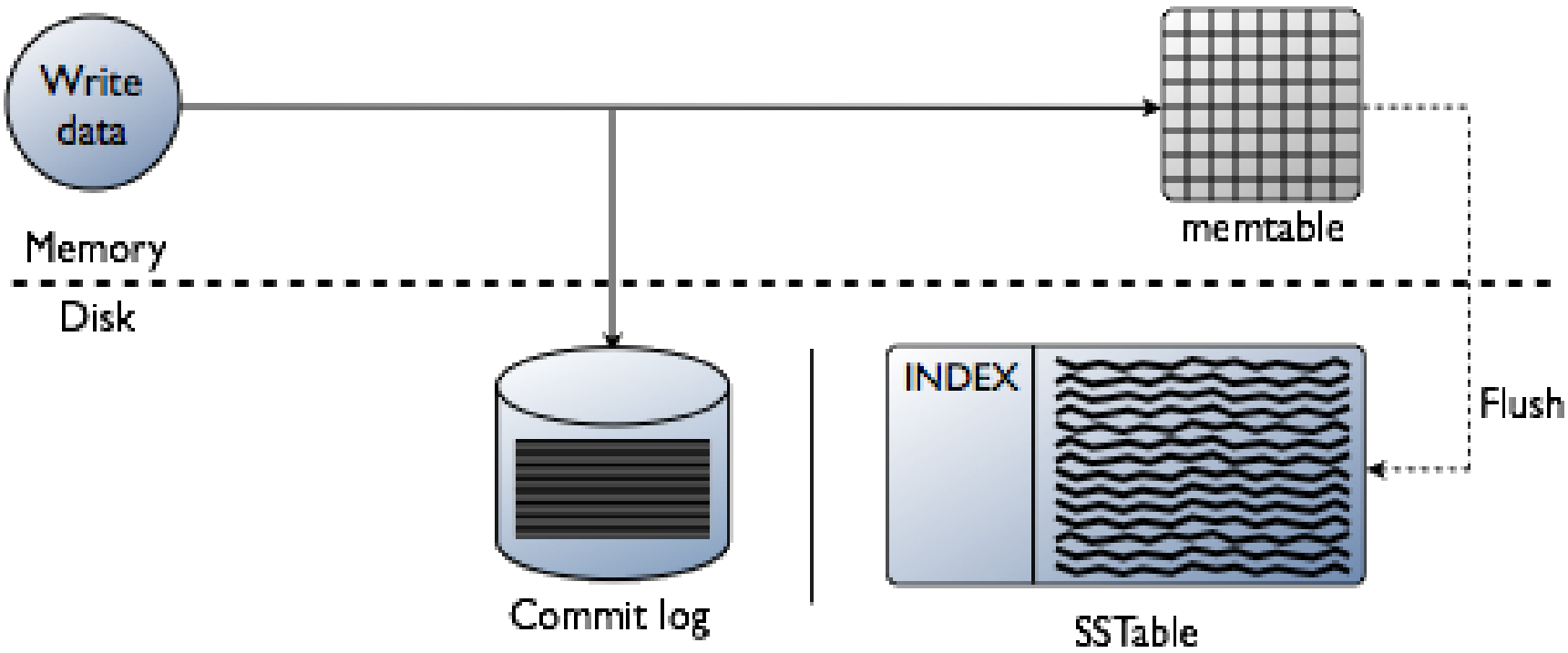
- Узел умер — пропустил удаление данных
- Вернулся к жизни — данные возродились

Anti-entropy

- Инициируем с помощью `nodetool repair`
- Запускаем readonly major compaction
- Строим Merkle Tree[a]
- Обмениваемся деревьями и ищем отличия
- Обмениваемся отличающимися сегментами

[a] `http://en.wikipedia.org/wiki/Merkle_tree`

Write Path


Write data
Memory
Disk
memtable
Flush
INDEX
Commit log
SSTable

Пояснения

- Memtables and SSTables per table
- Memtable — отсортирована
- **SSTable** — неизменяемая отсортированная ассоциативная таблица
- Обычно строка распределена по нескольким SSTable

Пример

Команды:

```
write (k1, c1:v1)
write (k2, c1:v1 c2:v2)
write (k1, c1:v4 c3:v3 c2:v2)
```

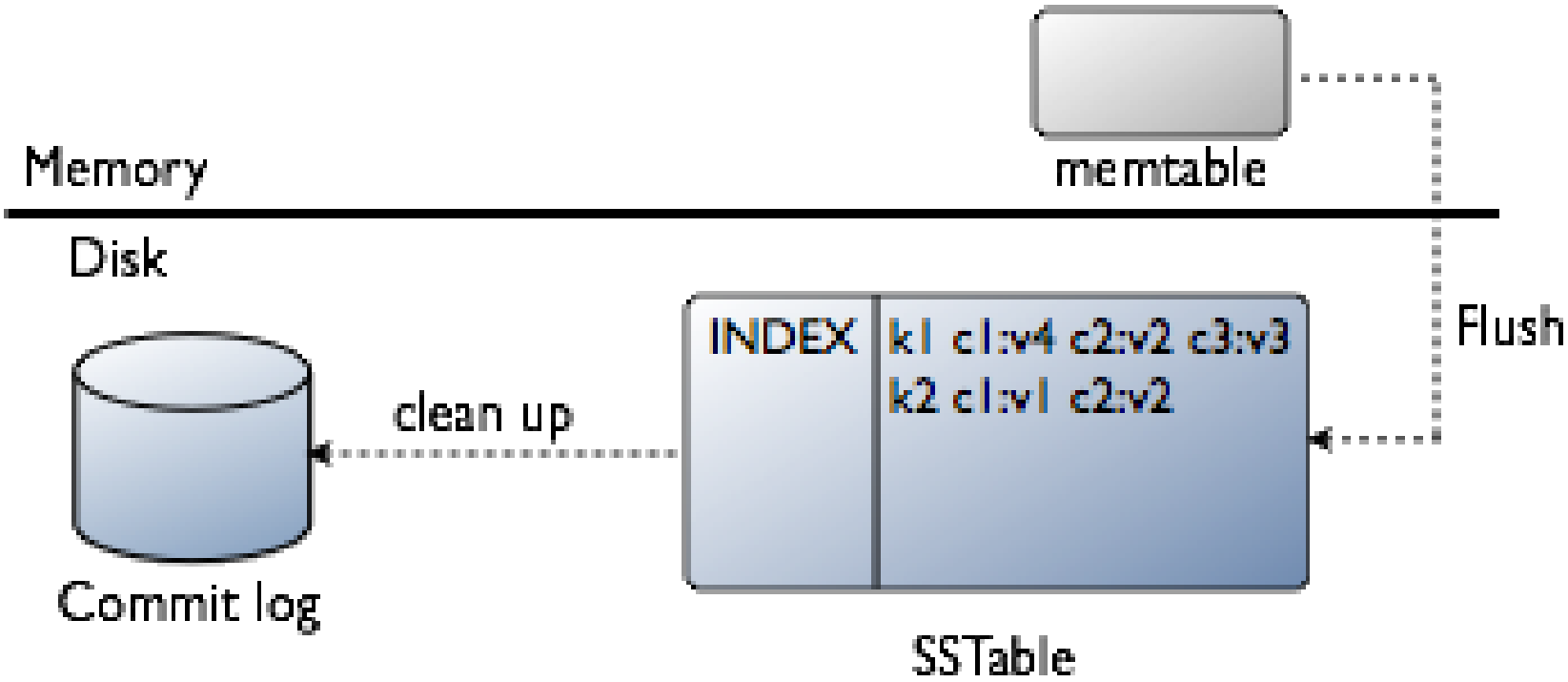
Commit-log:

```
k1, c1:v1
k2, c1:v1 c2:v2
k1, c1:v4 c3:v3 c2:v2
```

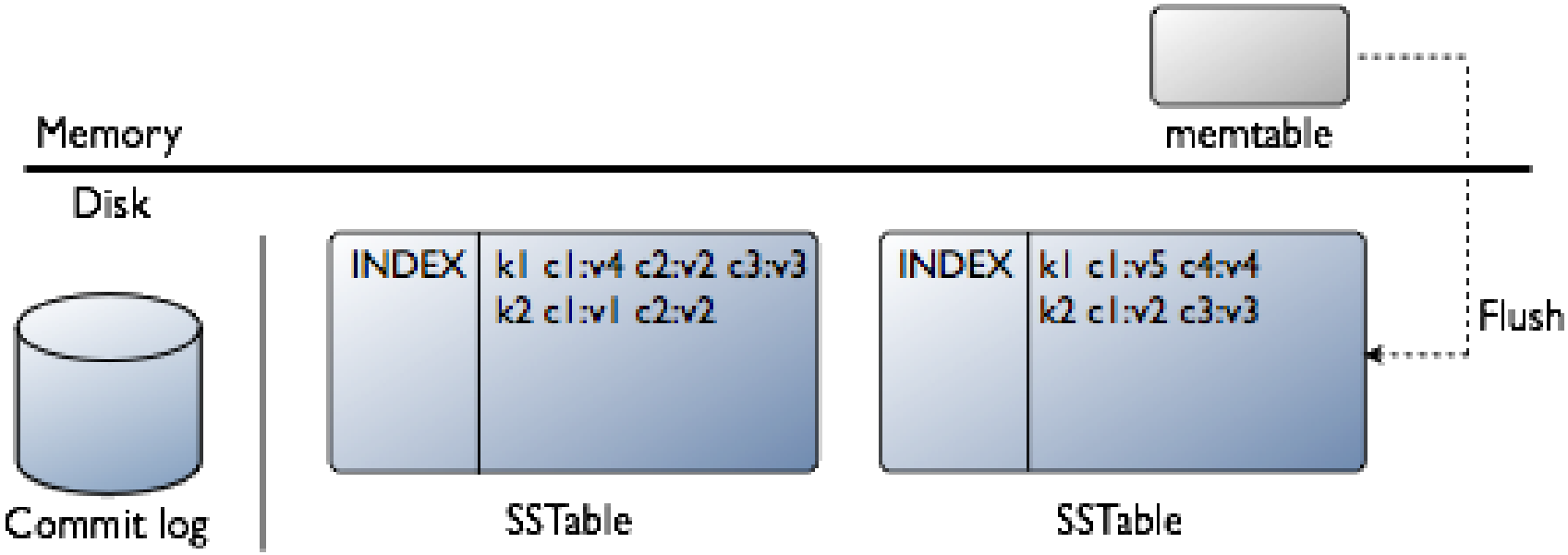
Memtable/SSTable:

```
k1 c1:v4 c2:v2 c3:v3
k2 c1:v1 c2:v2
```

Flush


memtable
Memory
Disk
Flush
INDEX
k1 c1:v4 c2:v2 c3:v3
k2 c1:v1 c2:v2
clean up
Commit log
SSTable

Update Path


memtable
Memory
Disk
INDEX
k1 c1:v4 c2:v2 c3:v3
k2 c1:v1 c2:v2
INDEX
k1 c1:v5 c4:v4
k2 c1:v2 c3:v3
Flush
Commit log
SSTable
SSTable

Delete

- SSTable неизменяема
- Delete — tombstone marker
- Реальное удаление по истечении `gc_grace_seconds` во время compaction
- Удалённые данные могут возродиться (см. Anti-entropy)

Compaction

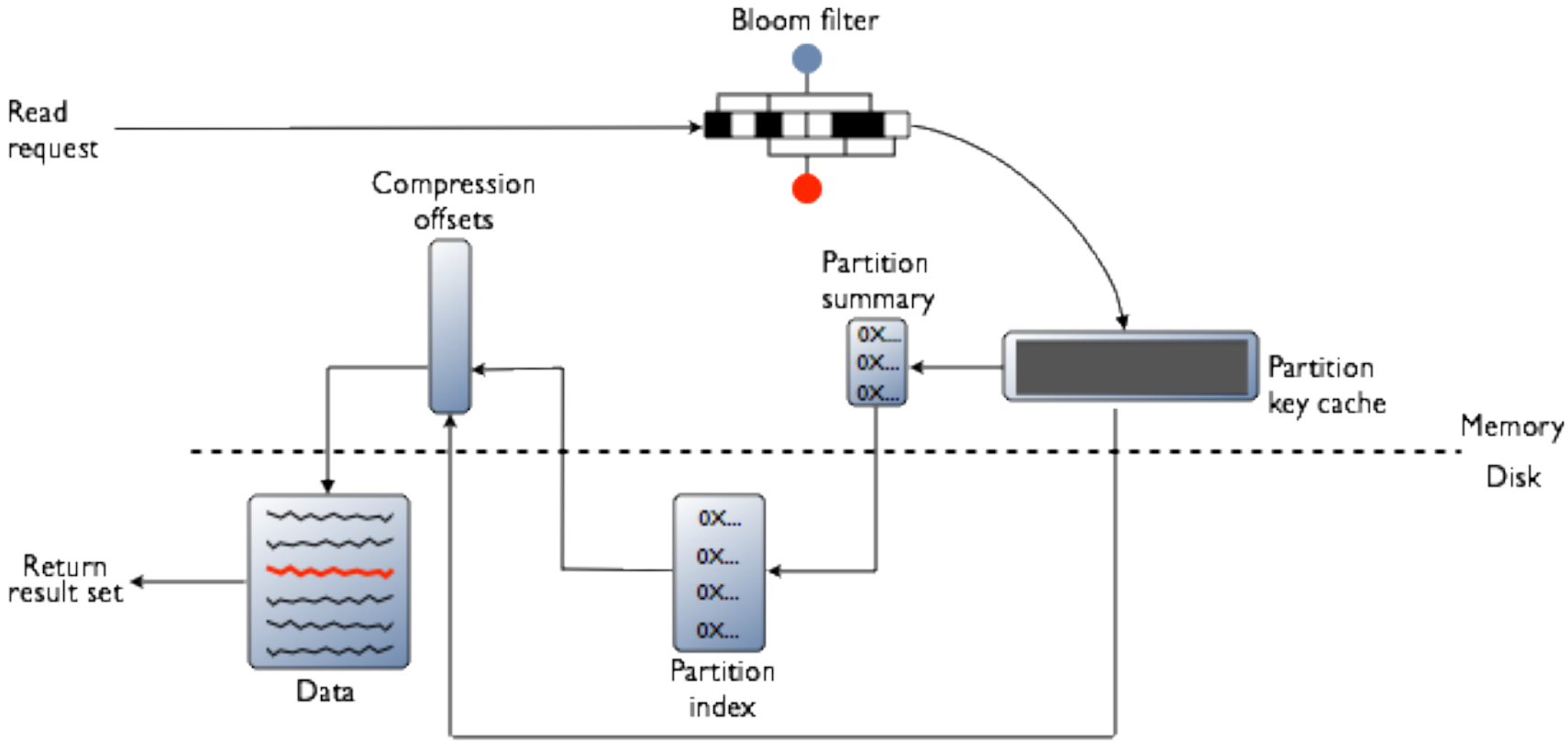
- Объединяет SSTable-файлы
  - Фрагменты строк
  - Протухшие tombstones
  - Перестраивает индексы
- SSTable отсортирована $\Rightarrow$ последовательный проход
- `SizeTieredCompactionStrategy`[11] vs `LeveledCompactionStrategy`[12]
- `DateTieredCompactionStrategy`[13]

[11] http://www.datastax.com/dev/blog/when-to-use-leveled-compaction

[12] http://www.datastax.com/dev/blog/leveled-compaction-in-apache-cassandra

[13] https://www.datastax.com/dev/blog/datetieredcompactionstrategy

Read


Bloom filter
Read request
Compression offsets
Partition summary
0X...
0X...
0X...
Partition key cache
Memory
Disk
Return result set
0X...
0X...
0X...
0X...
Data
Partition index

Комментарии

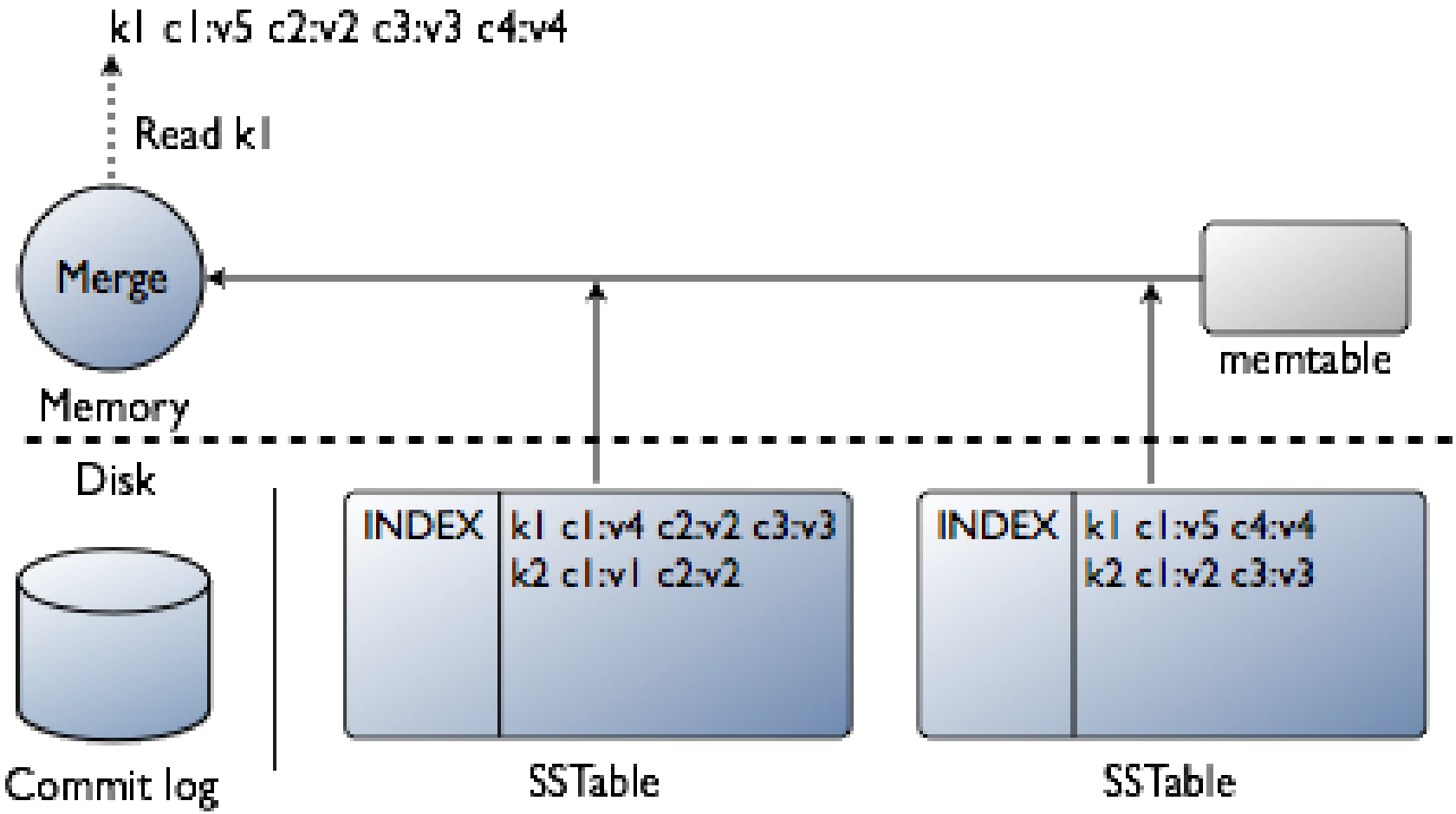
- Каждая SSTable имеет Bloom filter[14] — вероятность нахождения ключа в файле
- Если вероятность отлична от 0, идём в partition key cache
- Если нашли ключ в кэше, идём по смещению, находим сжатый блок и достаём данные
- Если не нашли ключ в кэше:
  - В partition summary примерно находим смещение на диске
  - Читаем последовательно блок с диска
  - Из compression offset map вынимаем индекс блока
  - Читаем сжатые данные и возвращаем клиенту

[14] http://en.wikipedia.org/wiki/Bloom_filter

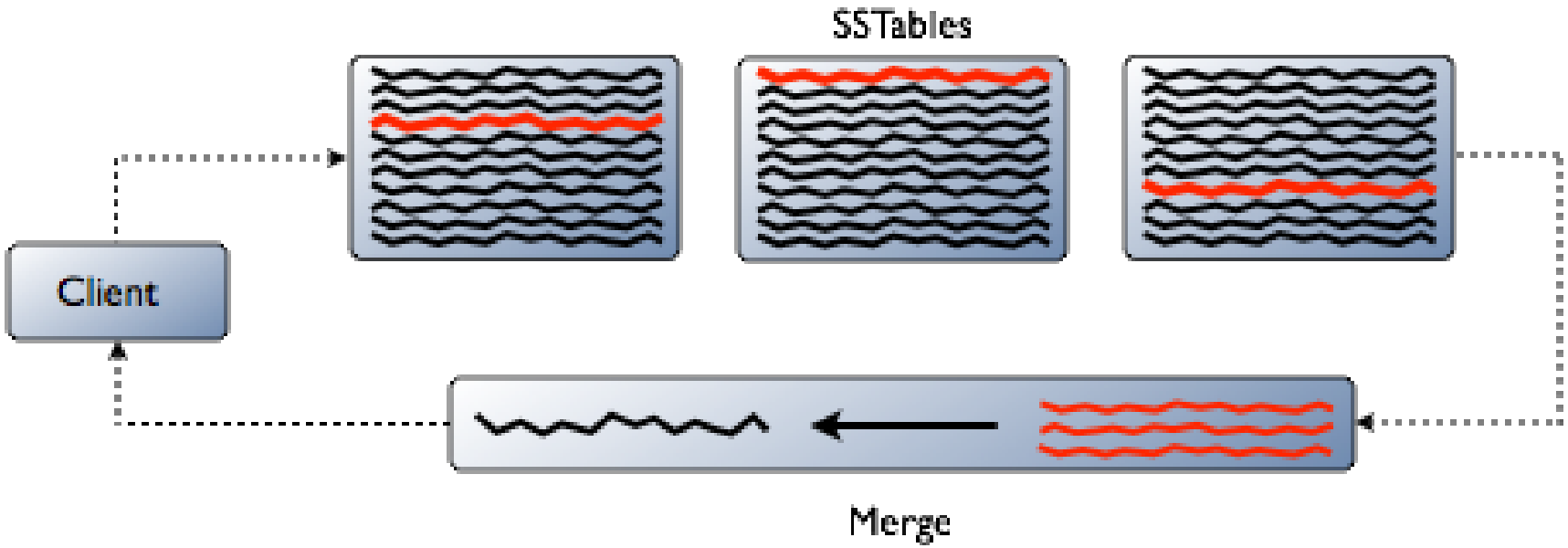
Дополнительные структуры

- Bloom filter — 1-2 ГБ / млрд. ключей
- Partition summary — по умолчанию шаг 128
- Compression offset map — 1-3 ГБ / 1 ТБ сжатых данных

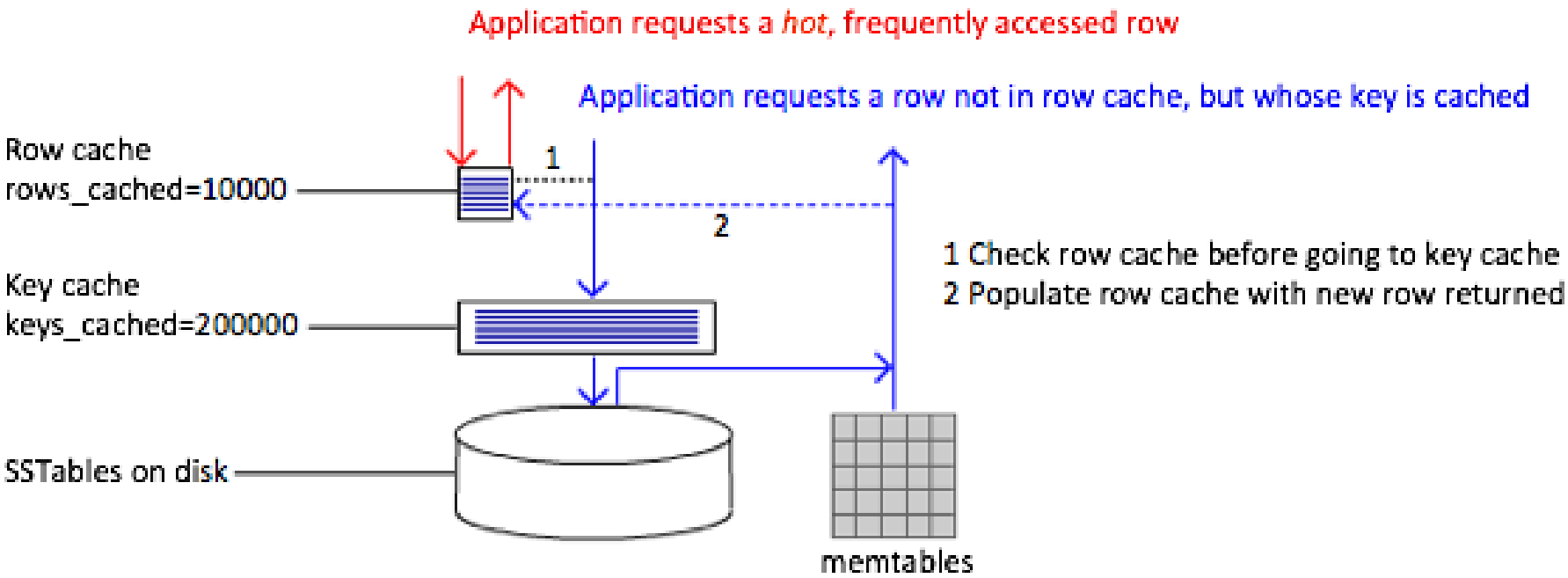
Read Path


k1 c1:v5 c2:v2 c3:v3 c4:v4
Read k1
Merge
memtable
Memory
Disk
INDEX
k1 c1:v4 c2:v2 c3:v3
k2 c1:v1 c2:v2
INDEX
k1 c1:v5 c4:v4
k2 c1:v2 c3:v3
Commit log
SSTable
SSTable

Write-through Row Cache



Key Cache + Row Cache



См. подробности[15].

[15] http://docs.datastax.com/en/cassandra/3.0/cassandra/operations/opsConfiguringCaches.html

CQL

```
1 CREATE TABLE comments (
2     article_id uuid,
3     posted_at timestamp,
4     author text,
5     karma int,
6     content text,
7     PRIMARY KEY (article_id, posted_at))
```

Колонки

620e8...	1340000000:	
	1340000000:author	Alice
	1340000000:content	Nice article, thanks
	1340000000:karma	4
	1340130000:	
	1340130000:author	Bob
	1340130000:content	I agree with Alice, +1
	1340130000:karma	0

См. подробности[16].

[16] http://www.datastax.com/dev/blog/thrift-to-cql3

Wide vs Skinny Rows

- Wide Rows
  - (+) Колонки отсортированы — возможны трюки[17]
  - (+/-) Атомарность
  - (-) Row cache (скорее всего) не работает
- Skinny Rows
  - (+) Row cache
  - (+/-) Неатомарно, но меньше contention
  - (-) Больше записей — больше индексы, хуже работает Bloom Filter

[17] Класс!ная Cassandra: `https://www.lektorium.tv/lecture/14531`

Осталось за кадром

- SEDA[18]
- Эксплуатация и тюнинг (JMX, `nodetool`)
- Масштабные примеры использования[19]
- См. блог DataStax[20]

[18] `http://en.wikipedia.org/wiki/Staged_event-driven_architecture`

[19] `https://www.datastax.com/customers`

[20] `http://www.datastax.com/dev/blog`

Вопросы?

- https://polis.mail.ru/blog/view/111/
- https://incubos.org/contacts/
- Не забудьте про ДЗ