

Если лекция про
Шардирование
была так хороша, то где же сиквел?



Илья Кокорин
kokorin.ilya.1998@gmail.com

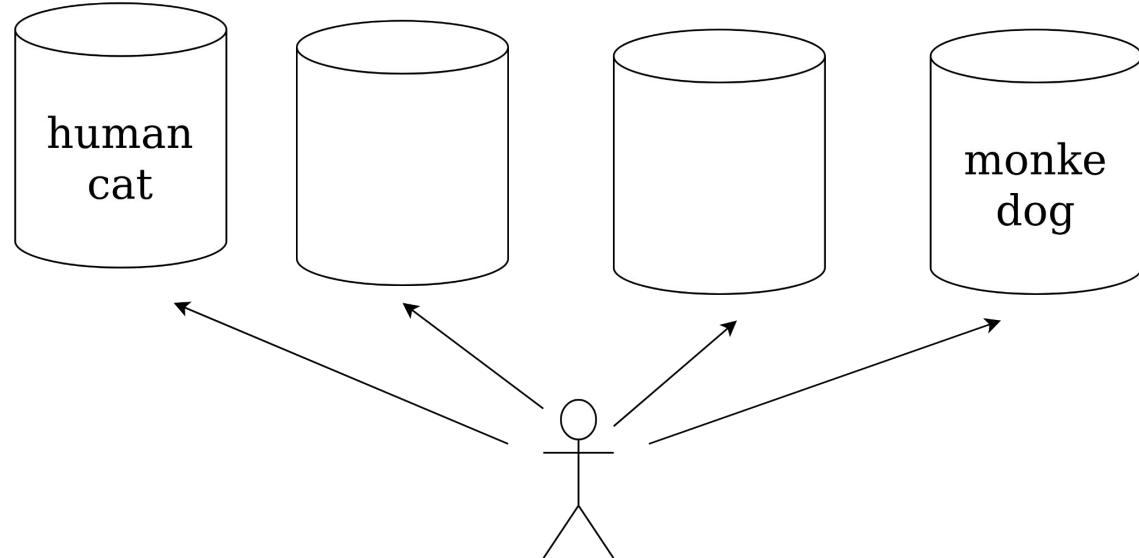
Иерархическое шардирование: мотивация

- Иногда ключи устроены иерархически
- Каждый ключ имеет вид a.b.c.d
- Представим, что мы делаем базу данных для биологов
- key = animals.chordata.mammals.primates.human
 - value = “Человек есть двуногое животное без перьев”
- key = animals.chordata.mammals.primates.monke
 - value =



Иерархические запросы

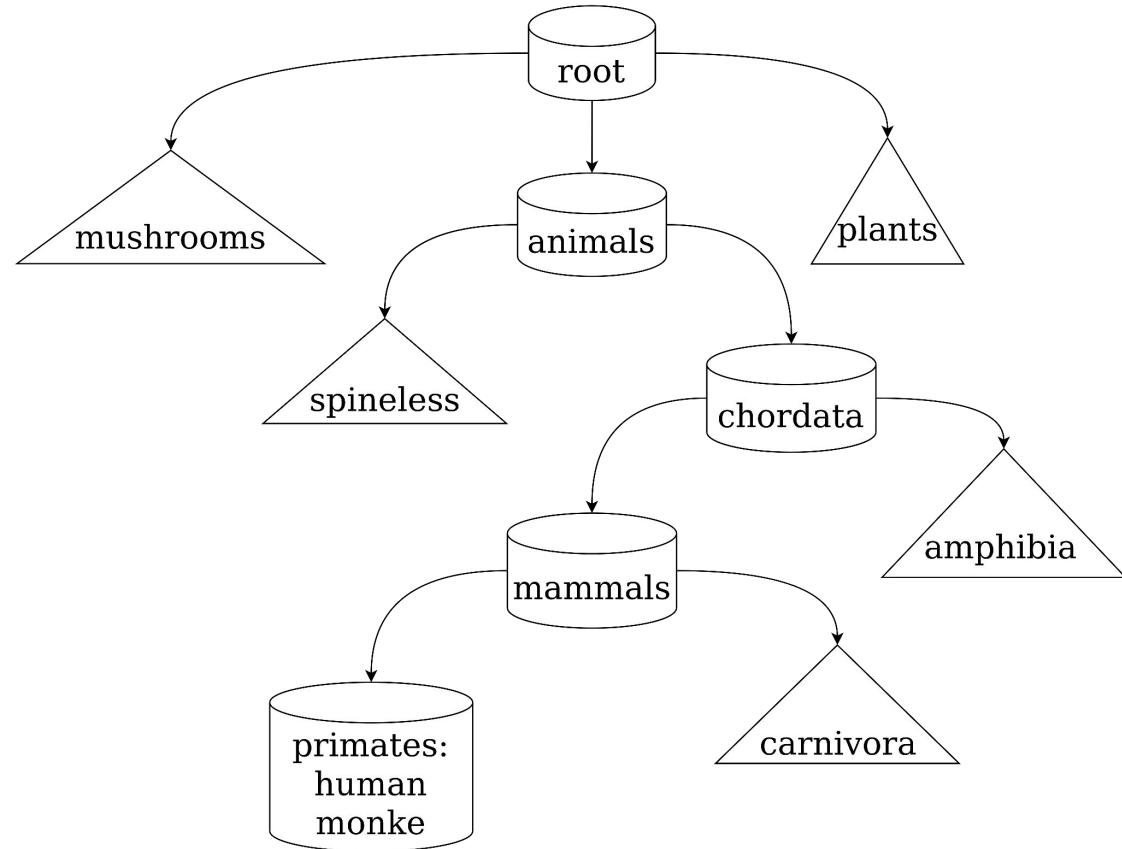
- Существуют запросы вида “дай мне информацию обо всех млекопитающих”
 - GET(animals.chordata.mammals.*)
- Если мы шардируем по хешу - нам придётся отправлять этот запрос на каждый шард



GET(animals.chordata.mammals.*)

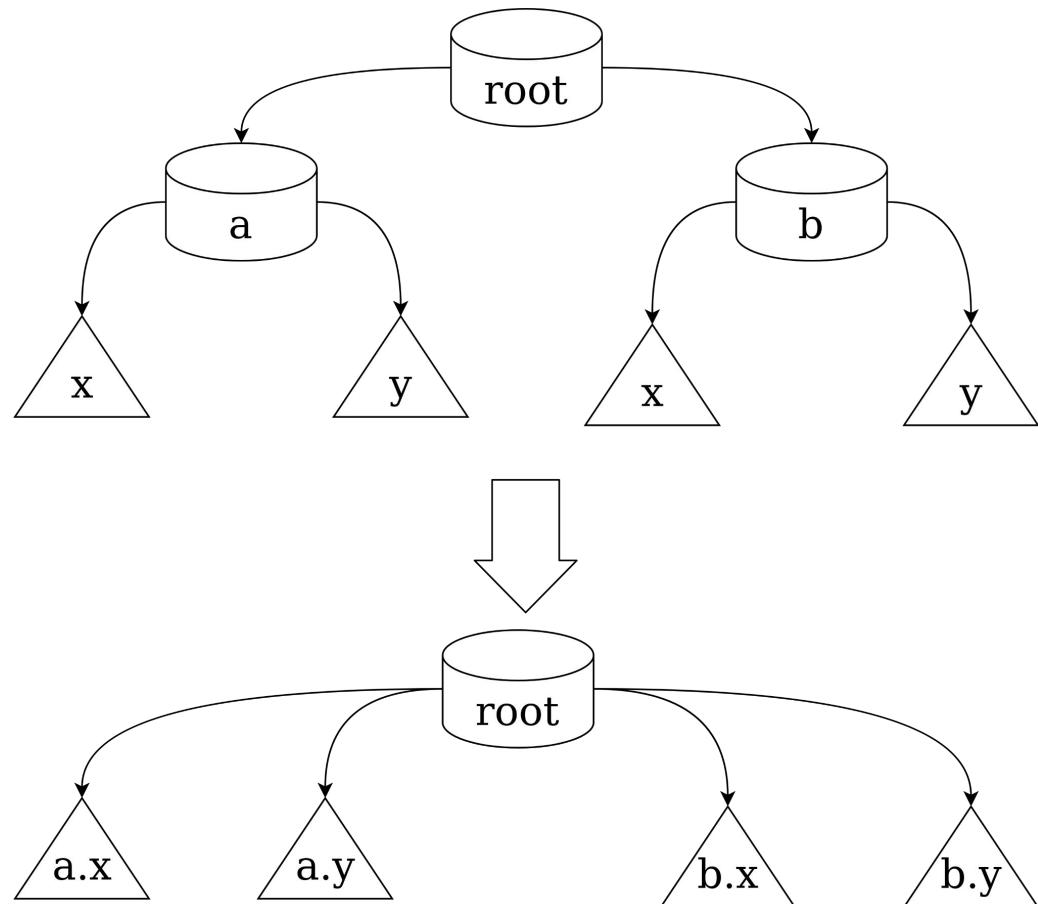
Иерархическое шардирование

- Раз ключи формируют дерево - то и шарды тоже должны формировать дерево
- Иерархический запрос обслуживается конкретным поддеревом



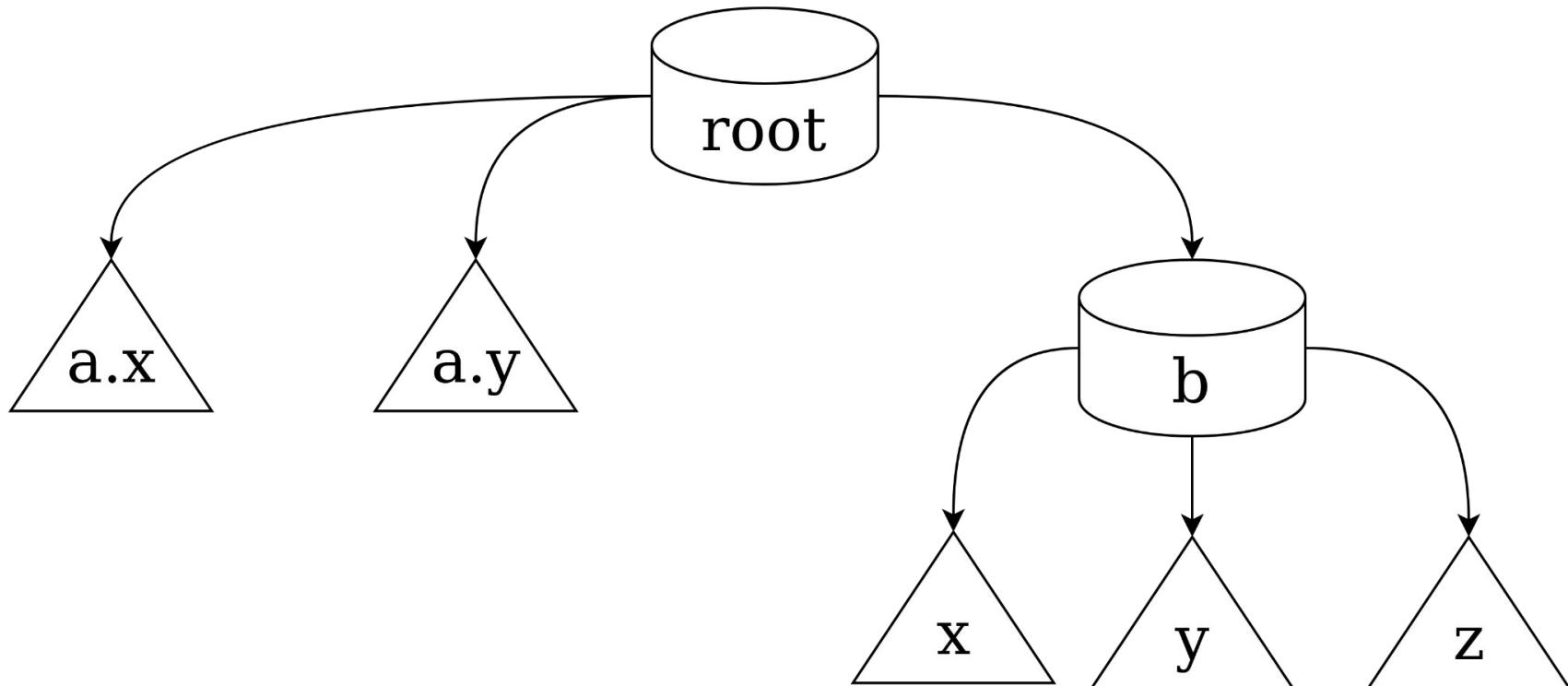
Иерархическое шардирование: сжатие

- Иногда на шарде хранится очень малое число записей
- Сделаем шард, который отвечает не за один уровень иерархии, а сразу за несколько



Иерархическое шардирование: сжатие

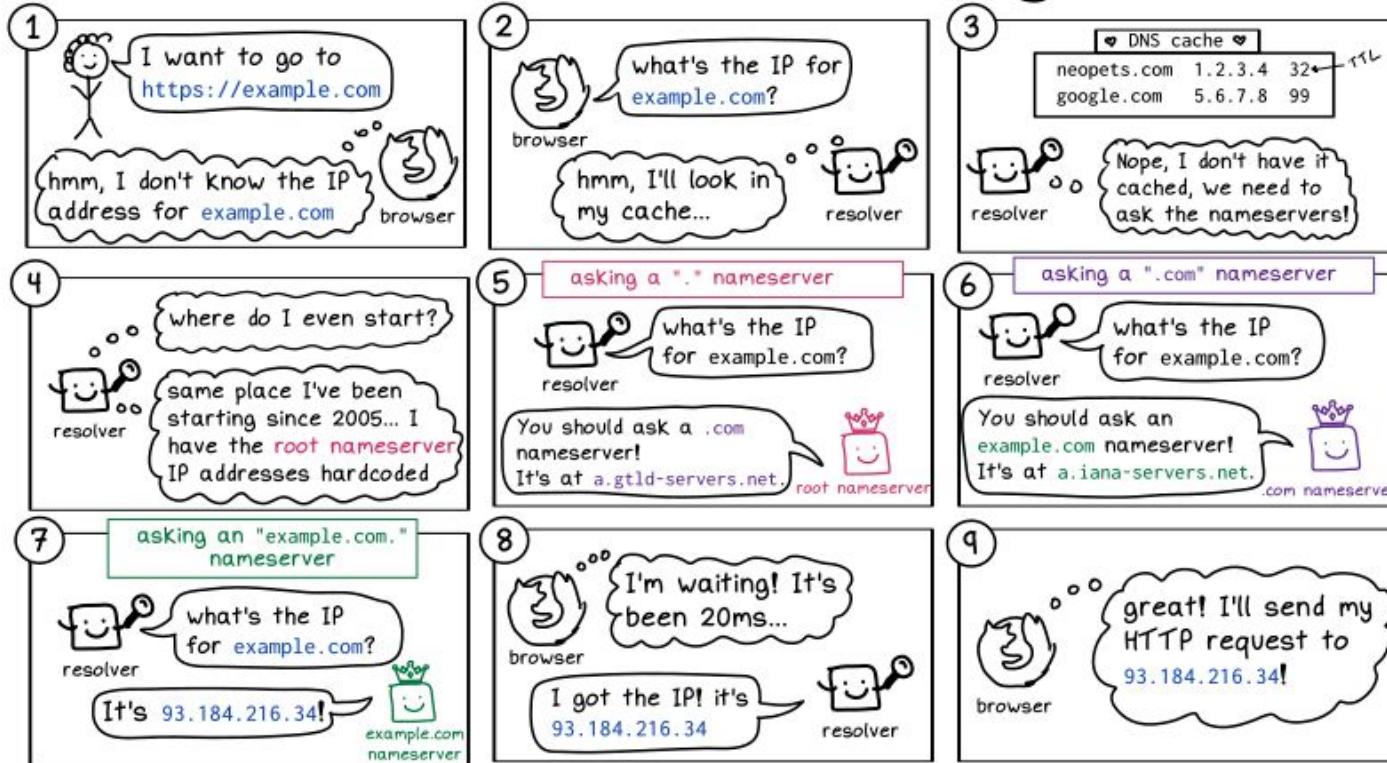
- Разбиение по уровням может быть произвольным



Иерархическое шардирование: DNS

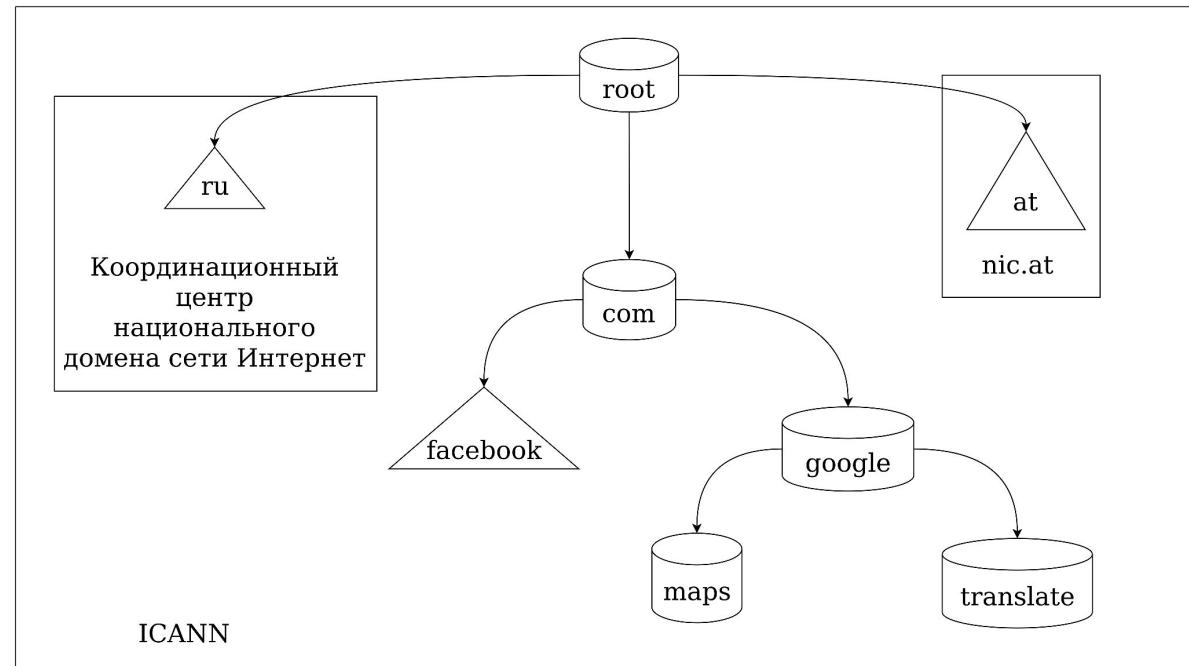
JULIA EVANS
@b0rk

life of a DNS query



Администрирование

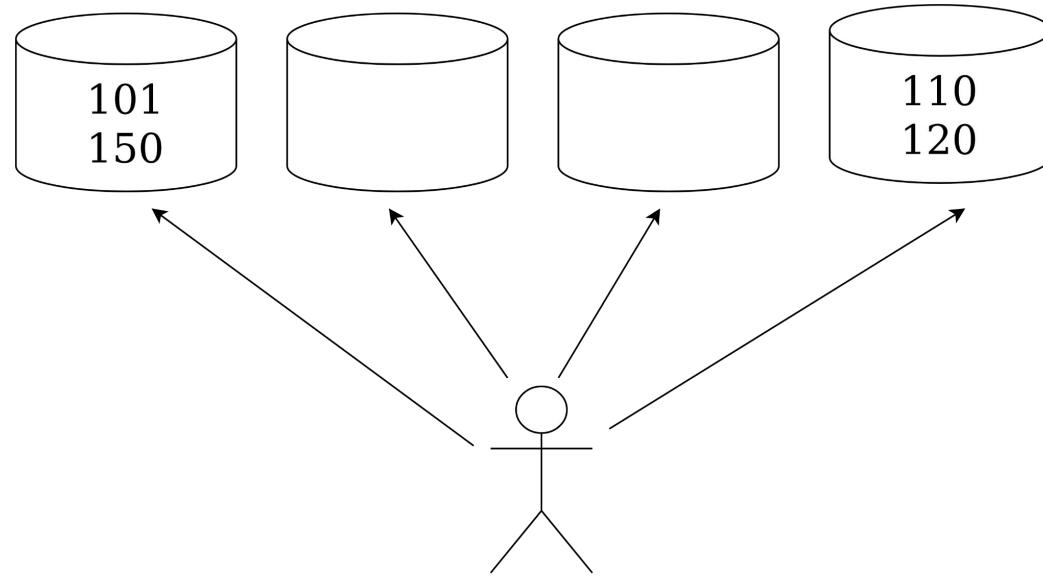
- Организация отвечает за целое поддерево шардов
- Делегирует другим организациям право администрировать свои поддеревья



Упорядоченное шардирование: мотивация

- Иногда ключи бывают упорядочены
- И мы можем делать по ним запросы, учитывающие порядок ключей

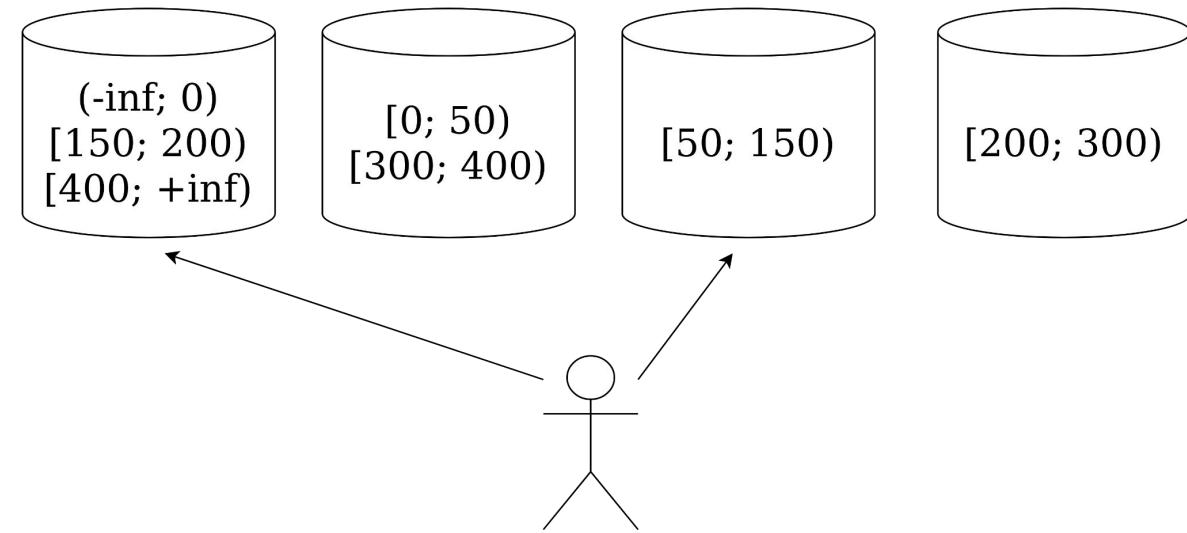
{key, value | key ∈ [a; b]}



SELECT key, value
FROM Table
WHERE 100 < key AND key < 200

Упорядоченное шардирование

- Храним ключи в упорядоченных блоках
- Для запроса можем сказать, в какие блоки нам нужно сделать запрос
- Делаем запрос только к серверам, хранящим нужные блоки

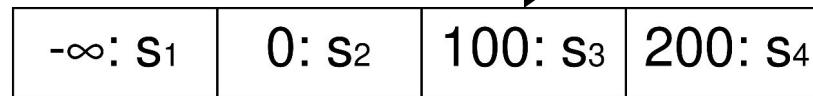


```
SELECT key, value  
FROM Table  
WHERE 100 < key AND key < 200
```

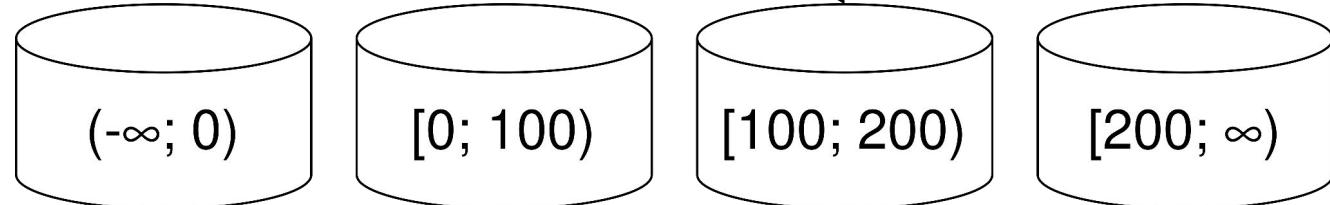
Упорядоченное шардирование: запрос

- Ищем блок, в который входит ключ
- Идём на сервер, хранящий этот блок

key = 150

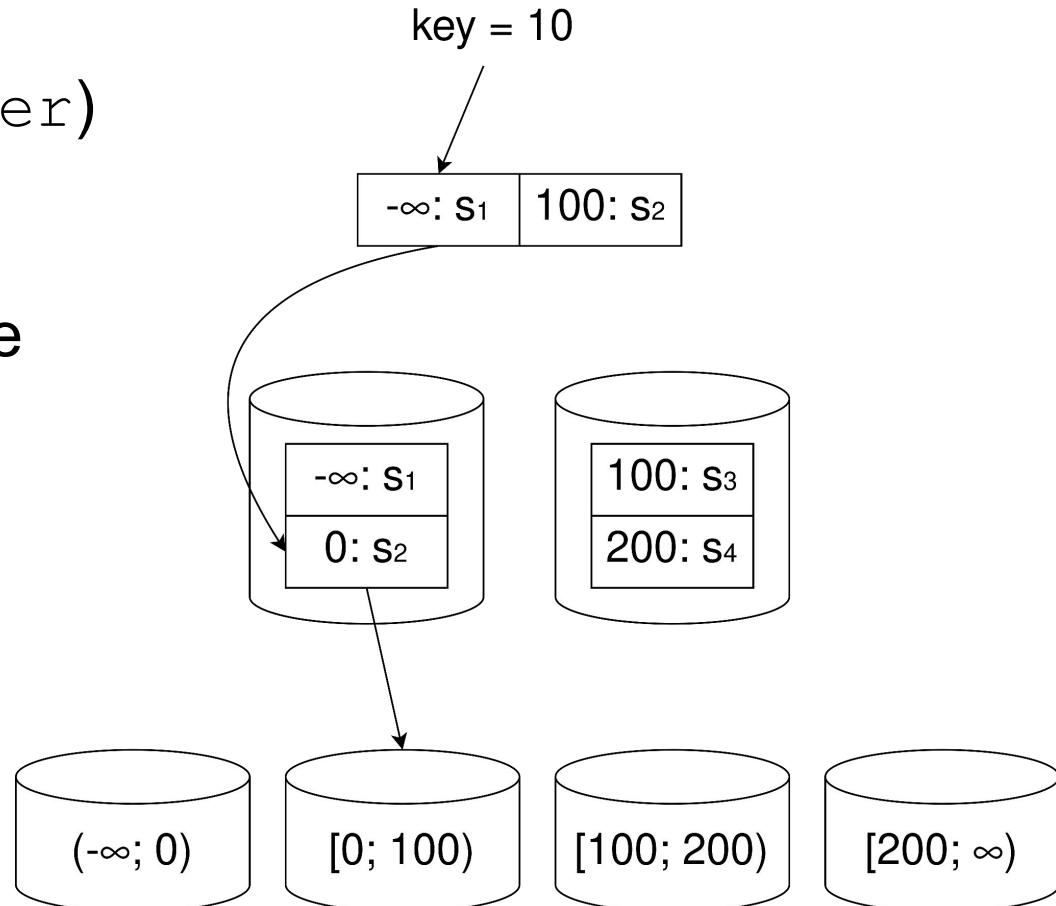


get(150)



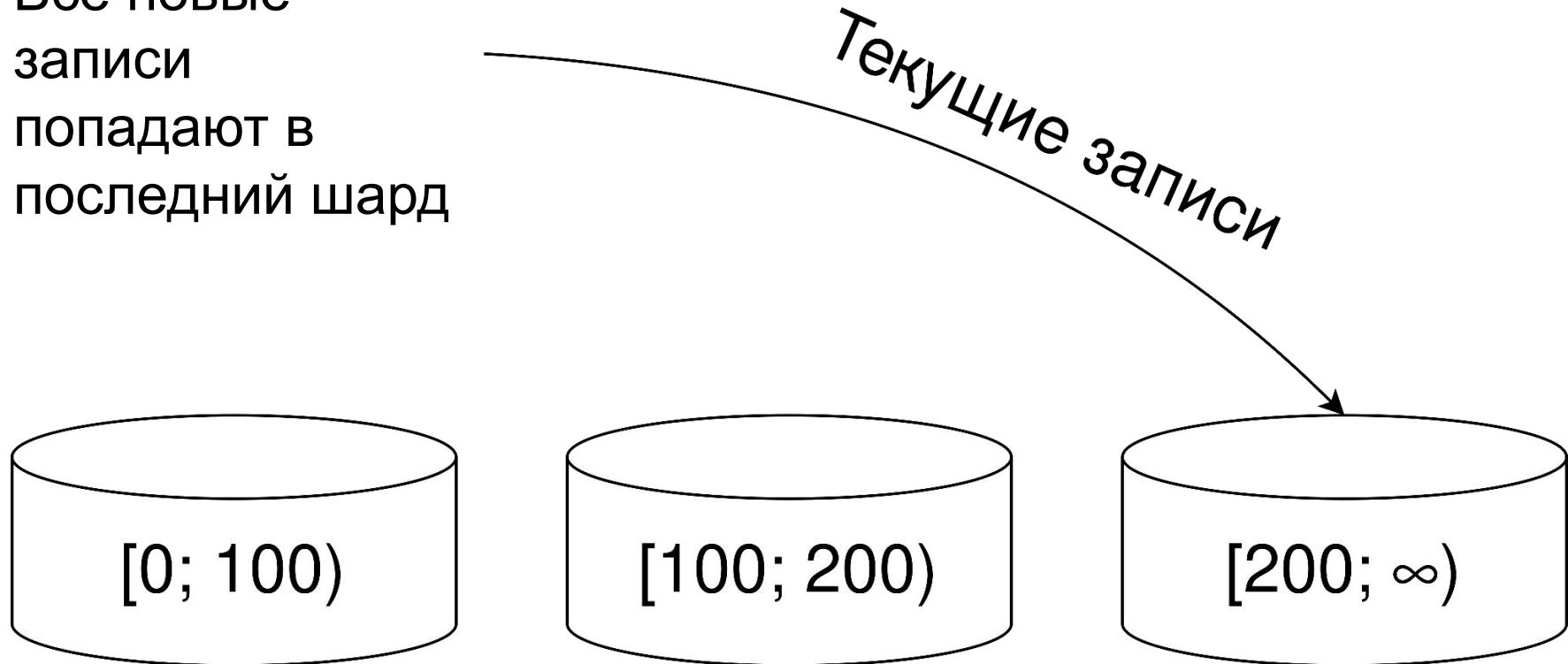
Многоуровневое шардирование

- Ключи вида
 $(\text{left_border} \rightarrow \text{server})$
также упорядоченно
шардируем
- Для этого нужно меньше
серверов
- Блоков меньше, чем
ключей
- Повторить, если нужно



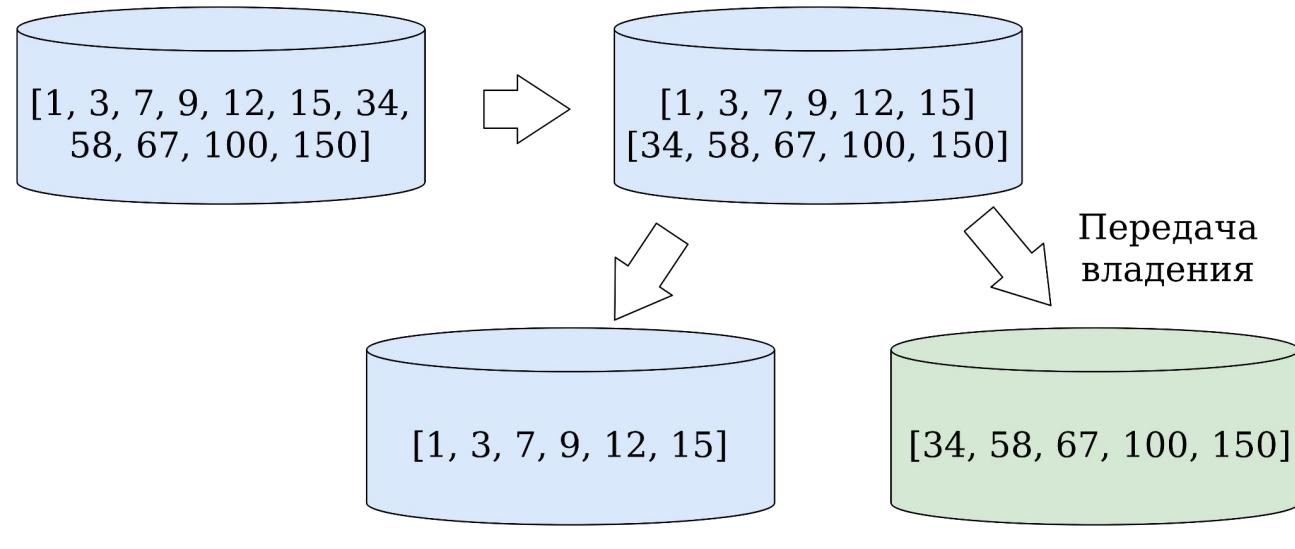
Неравномерность записи

- Записи шардируются по времени
- Все новые записи попадают в последний шард



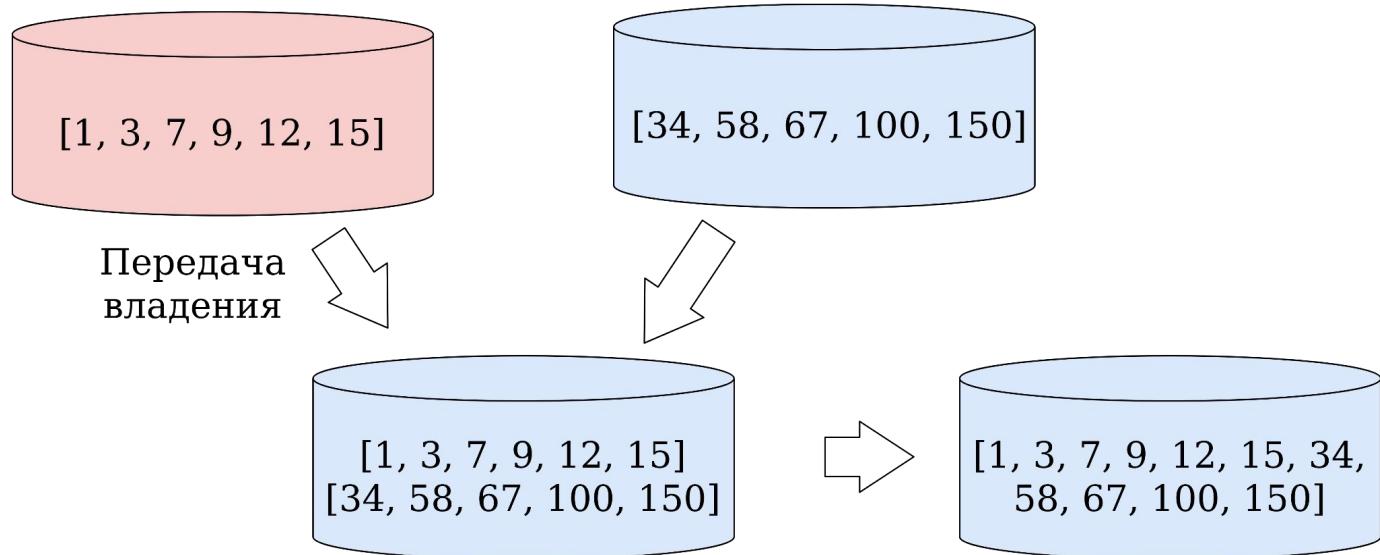
Разделение блоков

- Разделяем блоки когда
 - Его размер становится слишком большим
 - Нагрузка на него становится слишком большой
 - ...



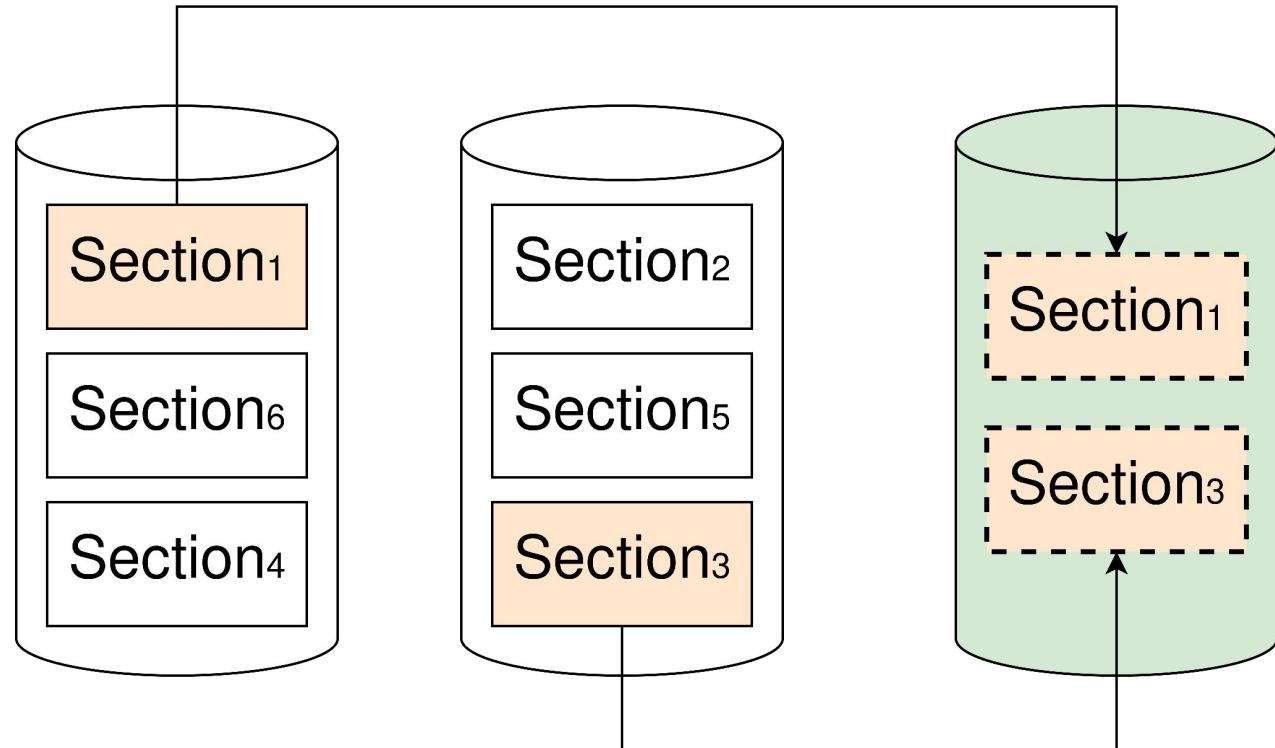
Слияние блоков

- Сливаем блоки когда
 - Размер двух соседних блоков становится слишком маленьким
 - Нагрузка на них становится слишком маленькой
 - ...



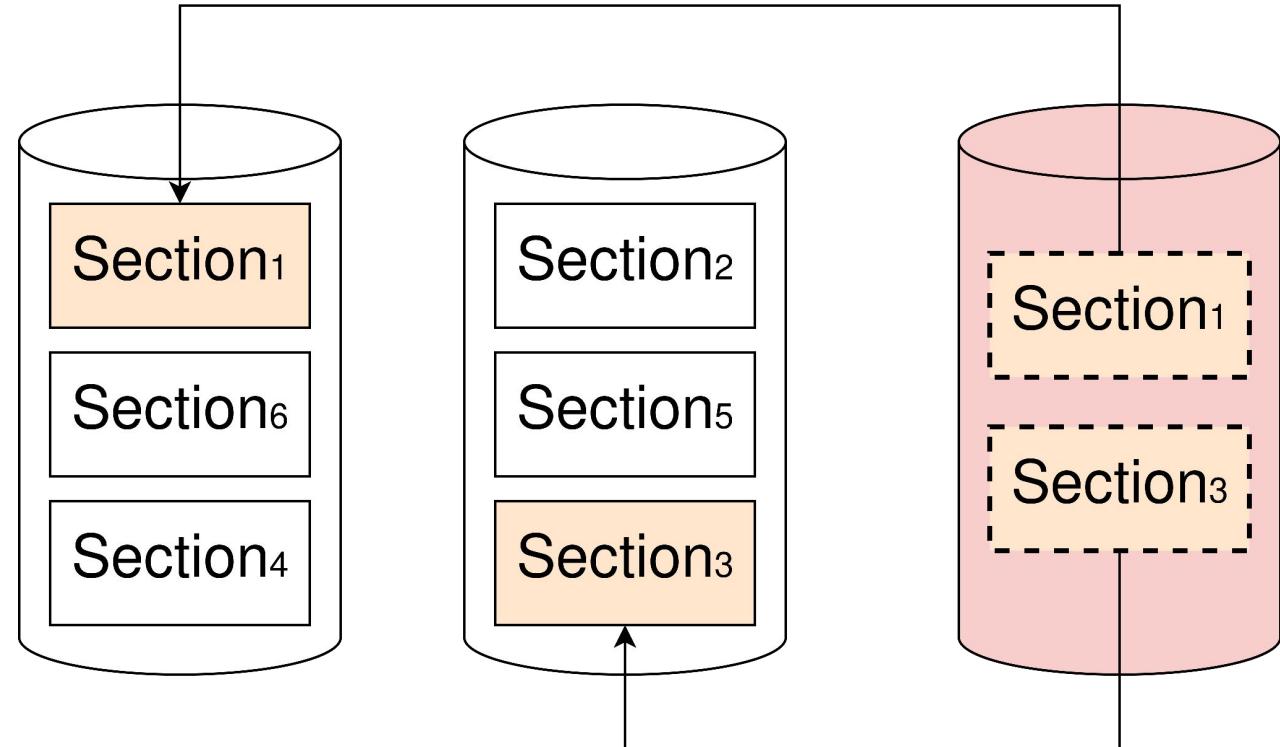
Добавление сервера

- Переносим часть блоков на новый сервер



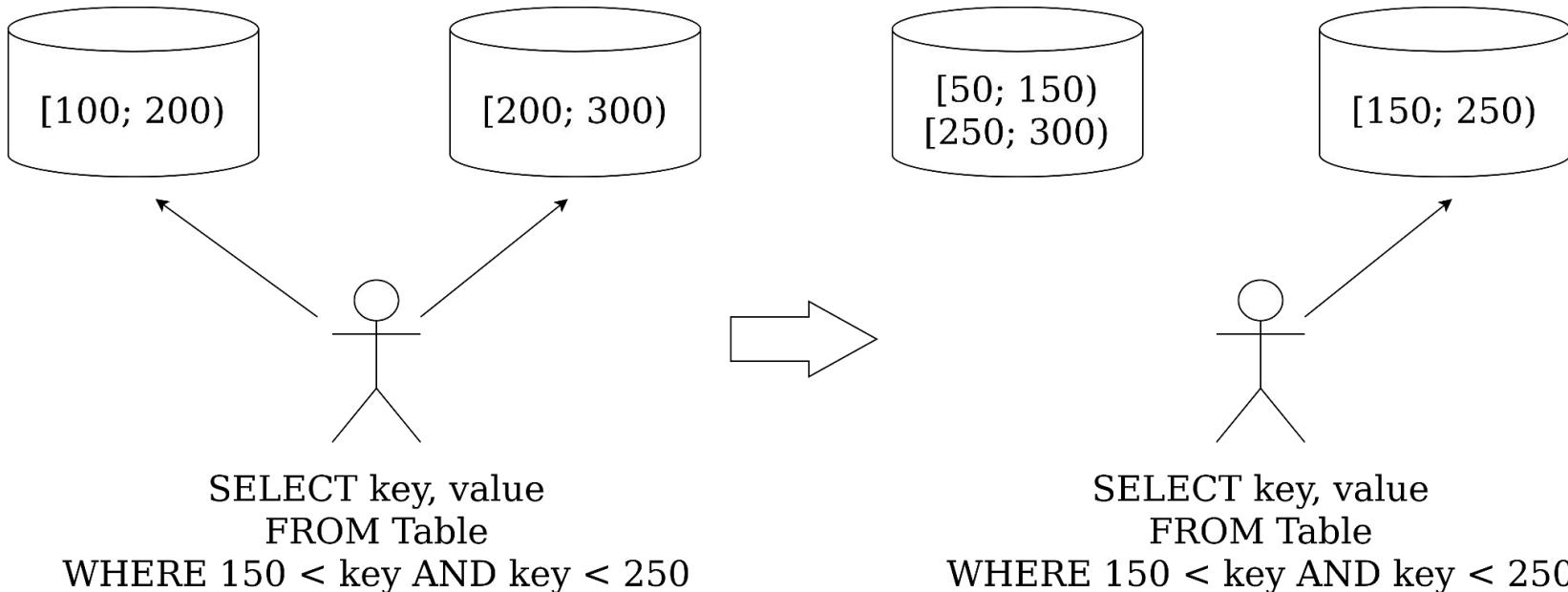
Удаление сервера

- Переносим блоки с удаляемого сервера на оставшиеся



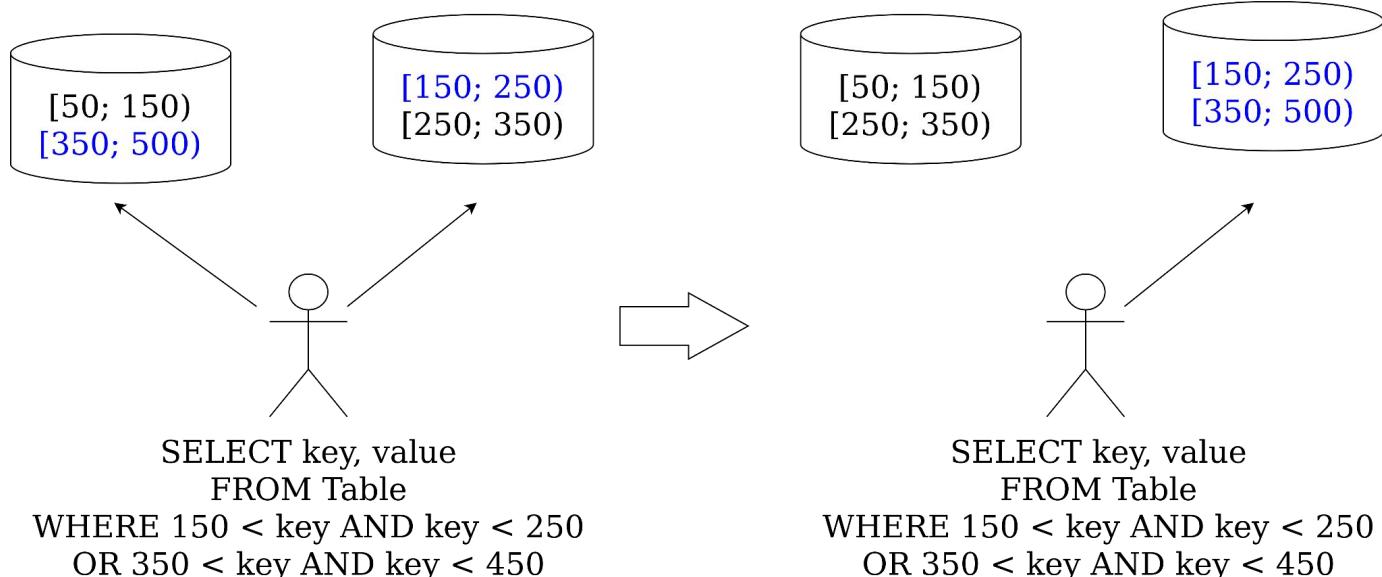
Изменение границ блоков

- Менять границы блоков можно в зависимости от запросов
- Чтобы популярные запросы затрагивали меньшее количество блоков (и, следовательно, серверов)



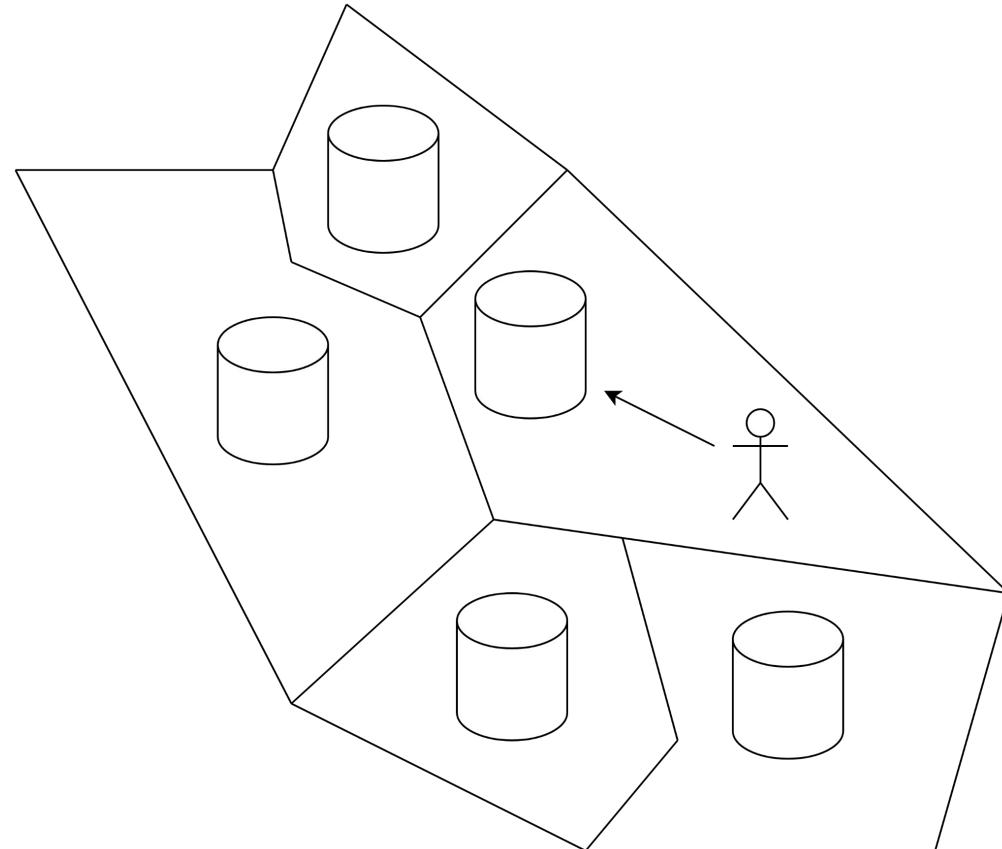
Коллокация блоков

- Менять расположение блоков можно в зависимости от запросов
- Чтобы популярные запросы, затрагивающие несколько несоседних блоков, затрагивали меньшее количество серверов



Гео-шардирование: мотивация

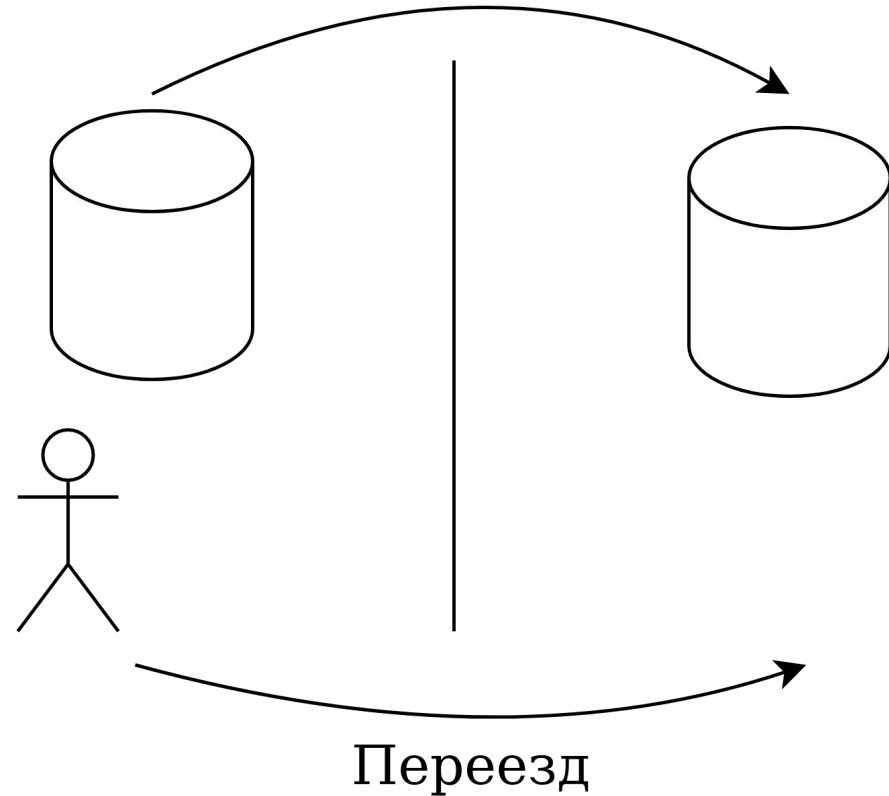
- Располагаем данные близко к пользователю, который будет запрашивать эти данные
 - Чтобы уменьшить задержку доступа
- По законодательным причинам



Переезд пользователя

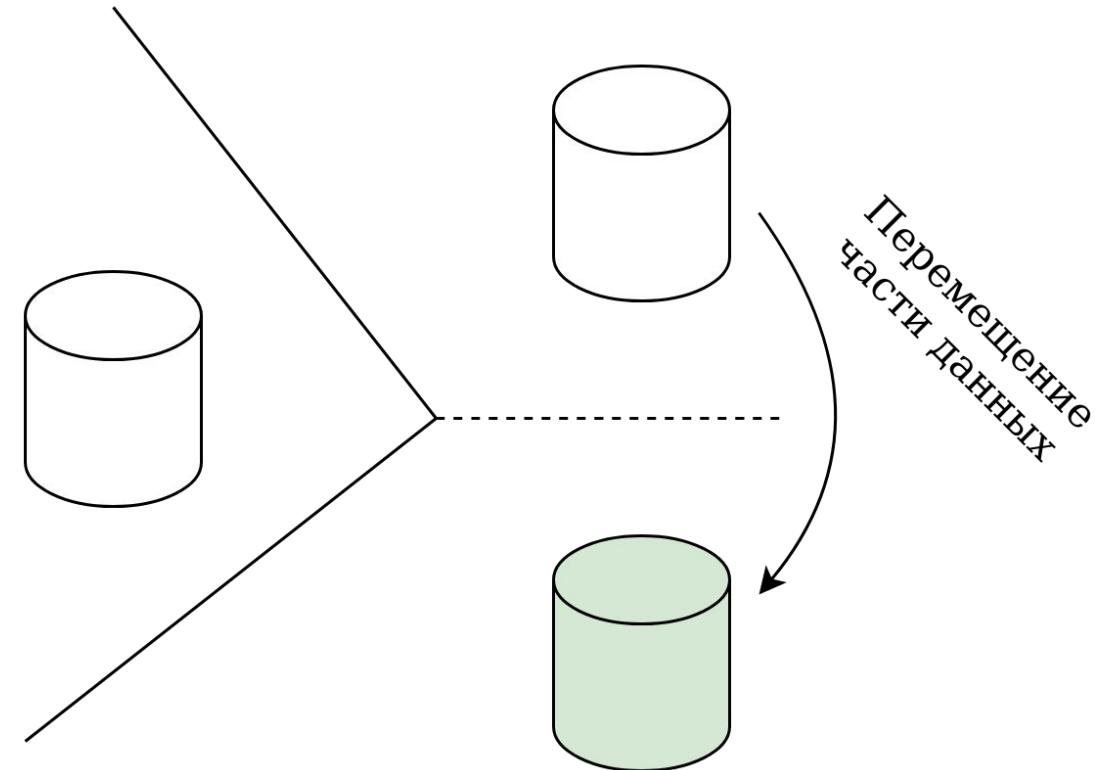
- Пользователь может переехать в другой регион
- Его данные должны переехать вслед за ним
 - Чтобы задержка продолжила оставаться низкой

Перемещение данных



Гео-шардирование: добавление серверов

- При добавлении новых шардов нам нужно переместить часть данных
 - Каких - зависит от географии
- А ещё могут меняться границы регионов



Полнотекстовый поиск: обратный индекс

- Входные данные:
корпус текстов

Full-text Search 101: The inverted index

User queries for “keeper”

1	The old night keeper keeps the keep in the town
2	In the big old house in the big old gown.
3	The house in the town had the big old keep
4	Where the old night keeper never did sleep.
5	The night keeper keeps the keep in the night
6	And keeps in the dark and sleeps in the light.

6 documents to index

- Итог: Map<Word, List<Docs>>
 - В которых это слово встречается

Term	Documents
and	<6>
big	<2> <3>
dark	<6>
did	<4>
gown	<2>
had	<3>
house	<2> <3>
in	<1> <2> <3> <5> <6>
keep	<1> <3> <5>
keeper	<1> <4> <5>
keeps	<1> <5> <6>
light	<6>
never	<4>
night	<1> <4> <5>
old	<1> <2> <3> <4>
sleep	<4>
sleeps	<6>
the	<1> <2> <3> <4> <5> <6>
town	<1> <3>
where	<4>

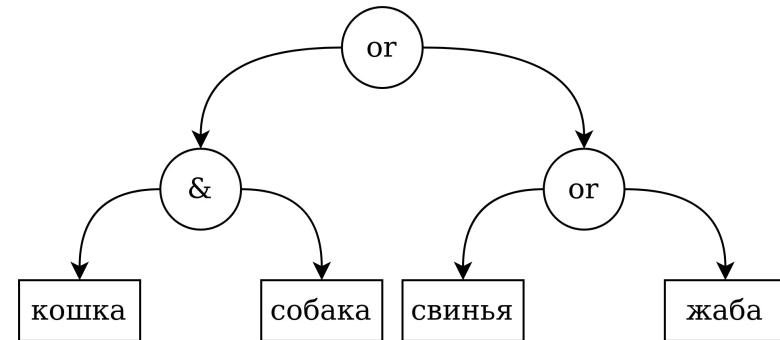
The index:

Dictionary and
posting lists

Булев поиск

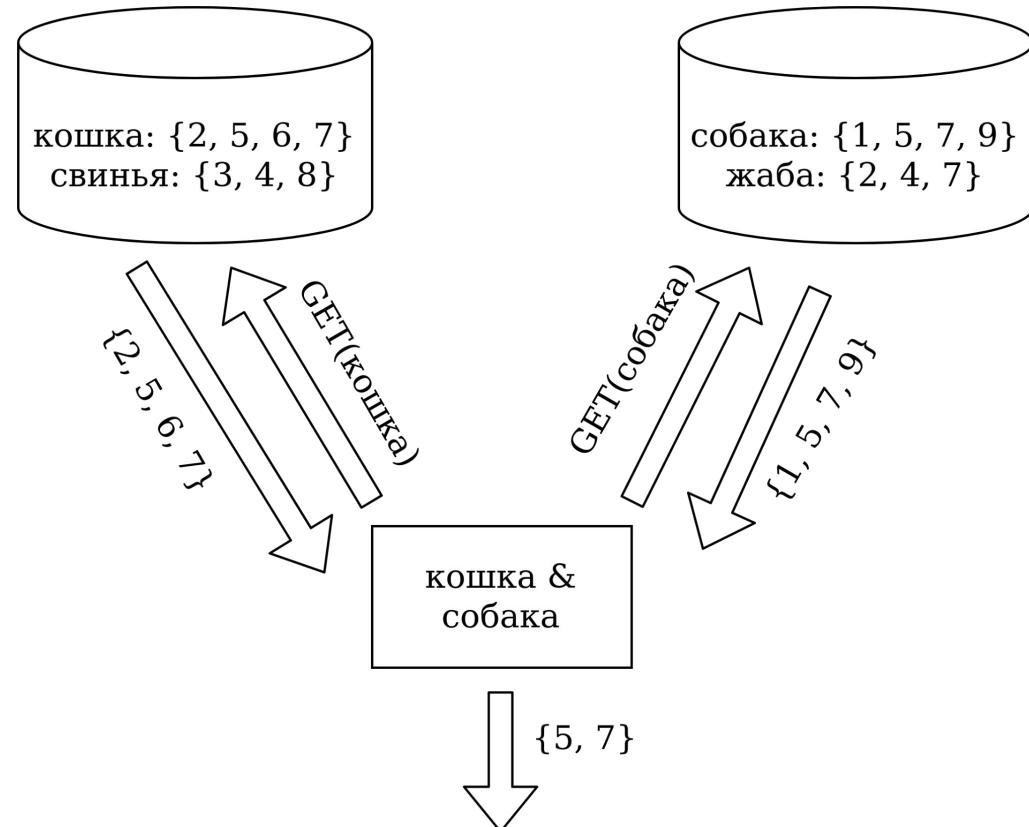
- Запросы вида (кошка & собака) | свинья | жаба

```
1 interface Query:  
2     fun get_result()  
3  
4 class WordQuery(word, index) implements Query:  
5     fun get_result():  
6         return index[word]  
7  
8 class AndQuery(query_a, query_b) implements Query:  
9     fun get_result():  
10        result_a = query_a.get_result()  
11        result_b = query_b.get_result()  
12        return result_a ∩ result_b  
13  
14 class OrQuery(query_a, query_b) implements Query:  
15    fun get_result():  
16        result_a = query_a.get_result()  
17        result_b = query_b.get_result()  
18        return result_a ∪ result_b
```



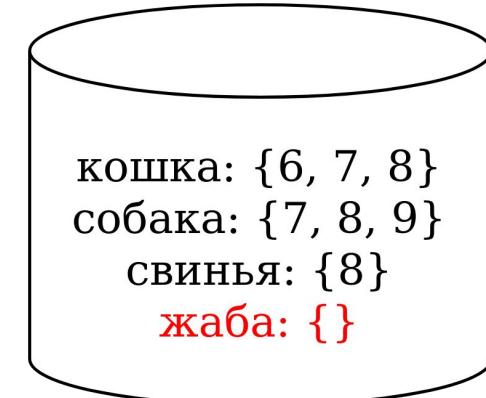
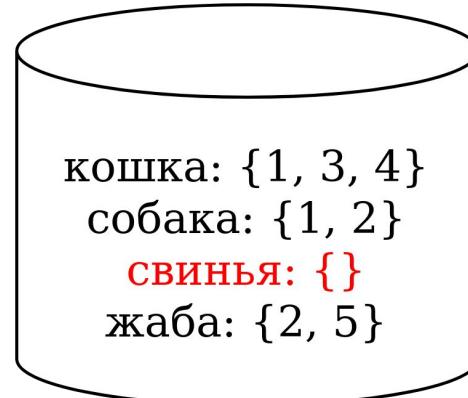
Шардирование по словам

- На каждом узле храним подмножество слов
- Для каждого слова локально храним список **всех** документов, где это слово встречается
- Не можем исполнить запрос локально



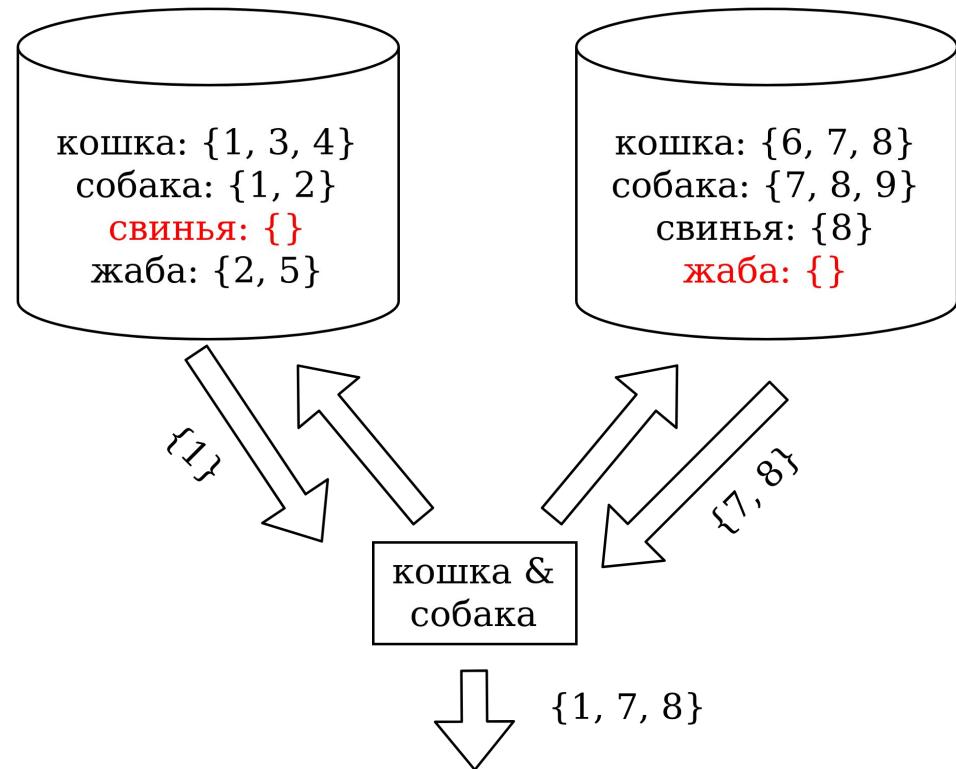
Шардирование по документам: хранение

- На каждом узле храним полный словарь
- Каждый узел отвечает за хранение только части документов
- Для каждого слова храним подмножество только тех документов, за которые отвечает этот сервер
- Если слово не содержится ни в одном из документов шарда, удалим его из словаря



Шардирование по документам: запрос

- Запрос может быть выполнен локально на каждом узле
- Так как весь список слов, встречающихся в документе хранится локально
- Для исполнения запроса нужно пойти на все шарды
 - В отличие от шардирования по словам



Boolean search at scale

Ways of Index Partitioning

- **By doc:** each shard has index for subset of docs
 - pro: each shard can process queries independently
 - pro: easy to keep additional per-doc information
 - pro: network traffic (requests/responses) small
 - con: query has to be processed by each shard
 - con: $O(K^*N)$ disk seeks for K word query on N shards

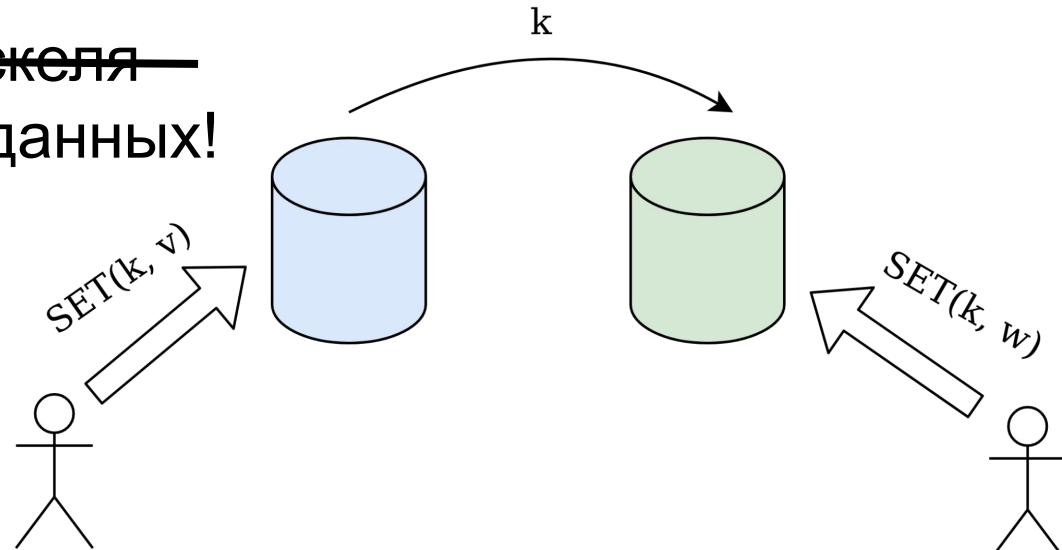
- **By word:** shard has subset of words for all docs
 - pro: K word query => handled by at most K shards
 - pro: $O(K)$ disk seeks for K word query
 - con: much higher network bandwidth needed
 - data about each word for each matching doc must be collected in one place
 - con: harder to have per-doc information



In our computing environment, **by doc** makes more sense

Решардинг: причина проблем

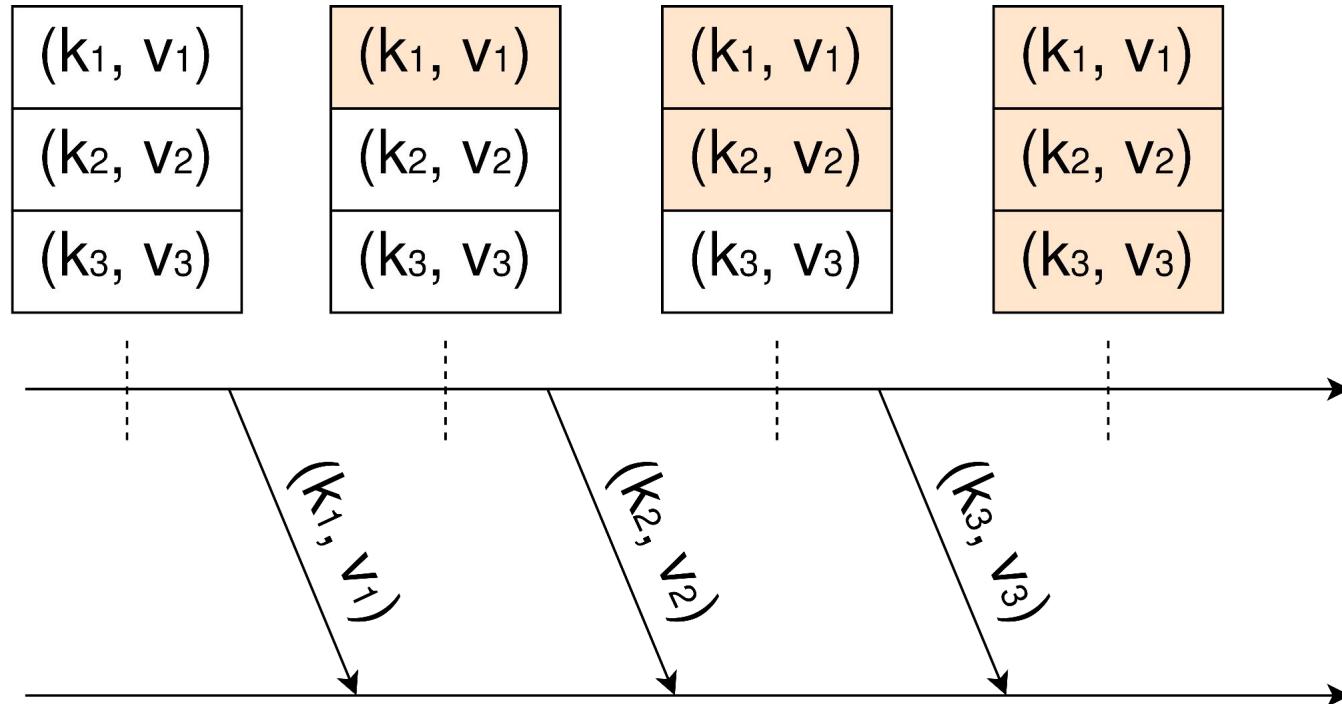
- Чего мы боимся?
 - ~~Темноты, пауков, хаскеля~~
 - Несогласованности данных!



- Перемещаем ключ K с узла N на узел M
- Клиент А считает, что ключ K ещё лежит на узле N
- Клиент В считает, что ключ K уже лежит на узле M
- Оба клиента делают запись на соответствующие узлы

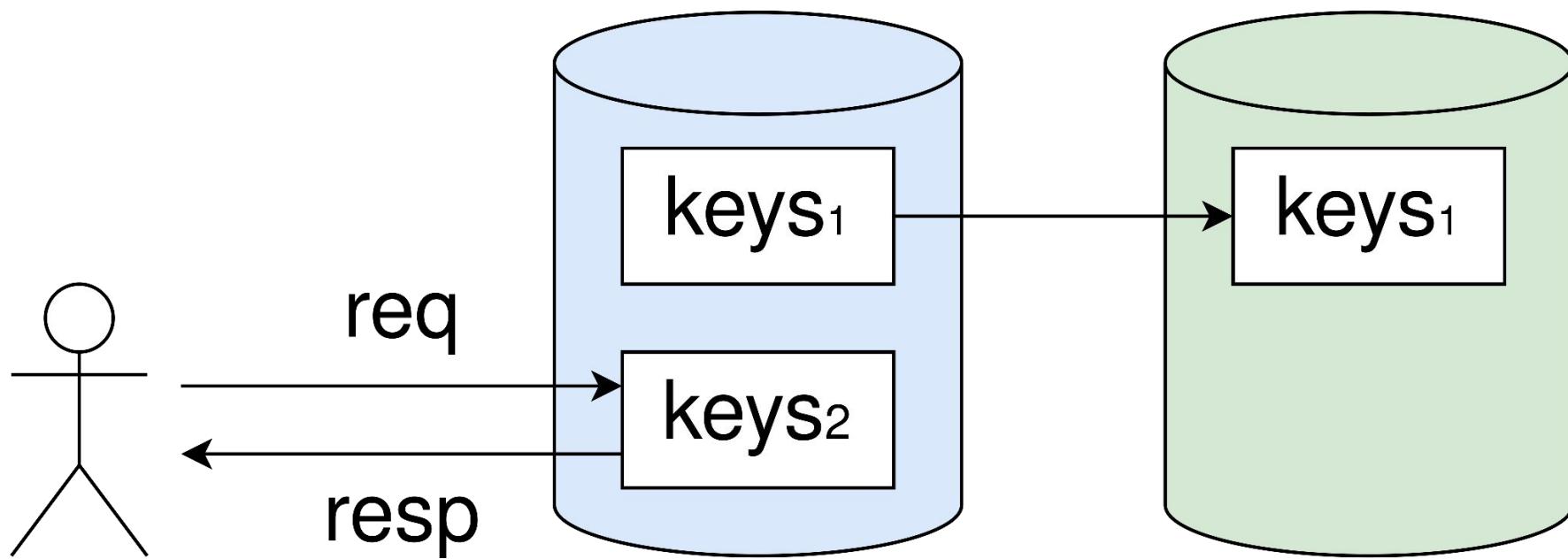
Решардинг

- Не можем перенести все ключи за раз
- Переносим небольшими пачками



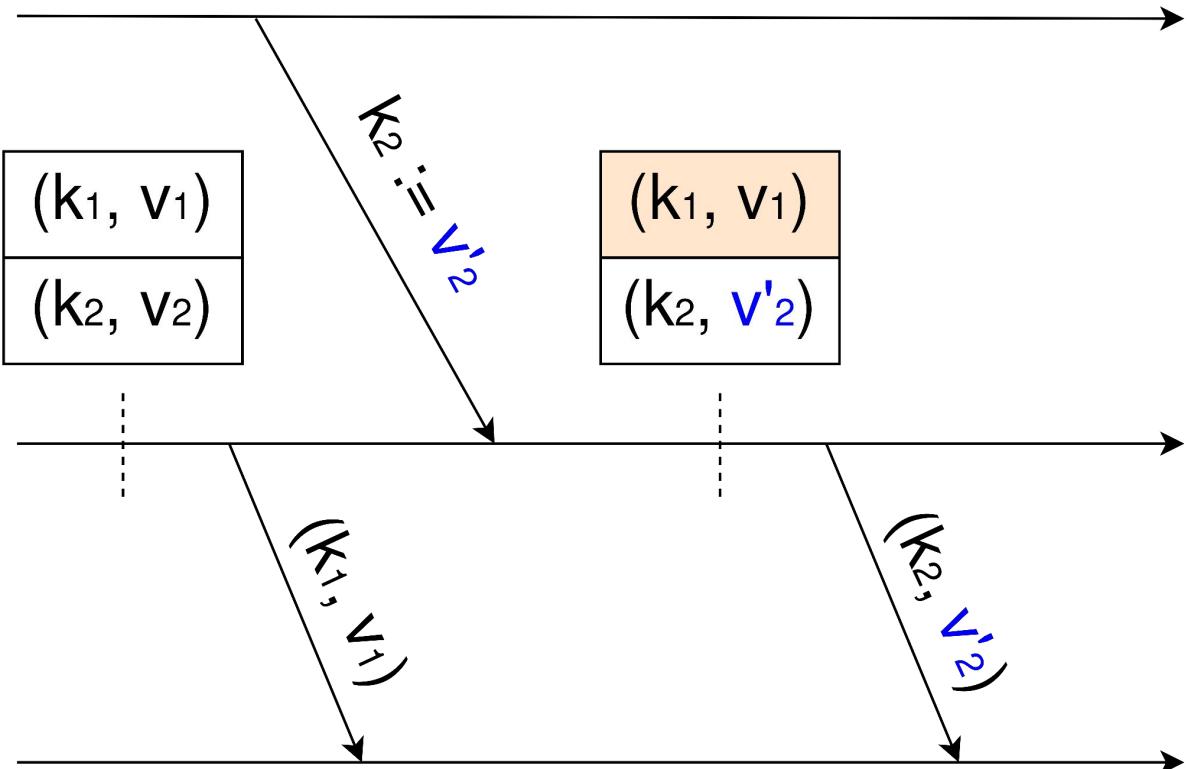
Решардинг

- Обычно с сервера мы переносим не все ключи
- По ключам, которые переносить не нужно, запросы исполняются как обычно



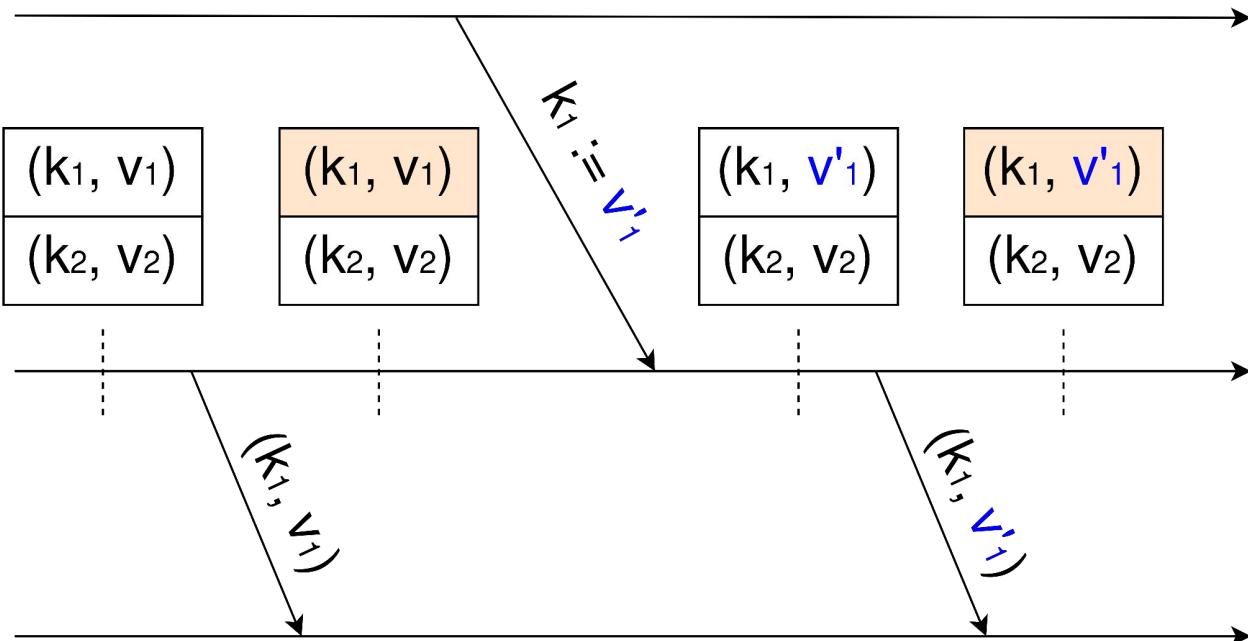
Решардинг: репликация изменений

- Старый хозяин обслуживает переносимые ключи
- В случае изменения ещё не перенесённого ключа, меняет его
- И переносит в свой черёд



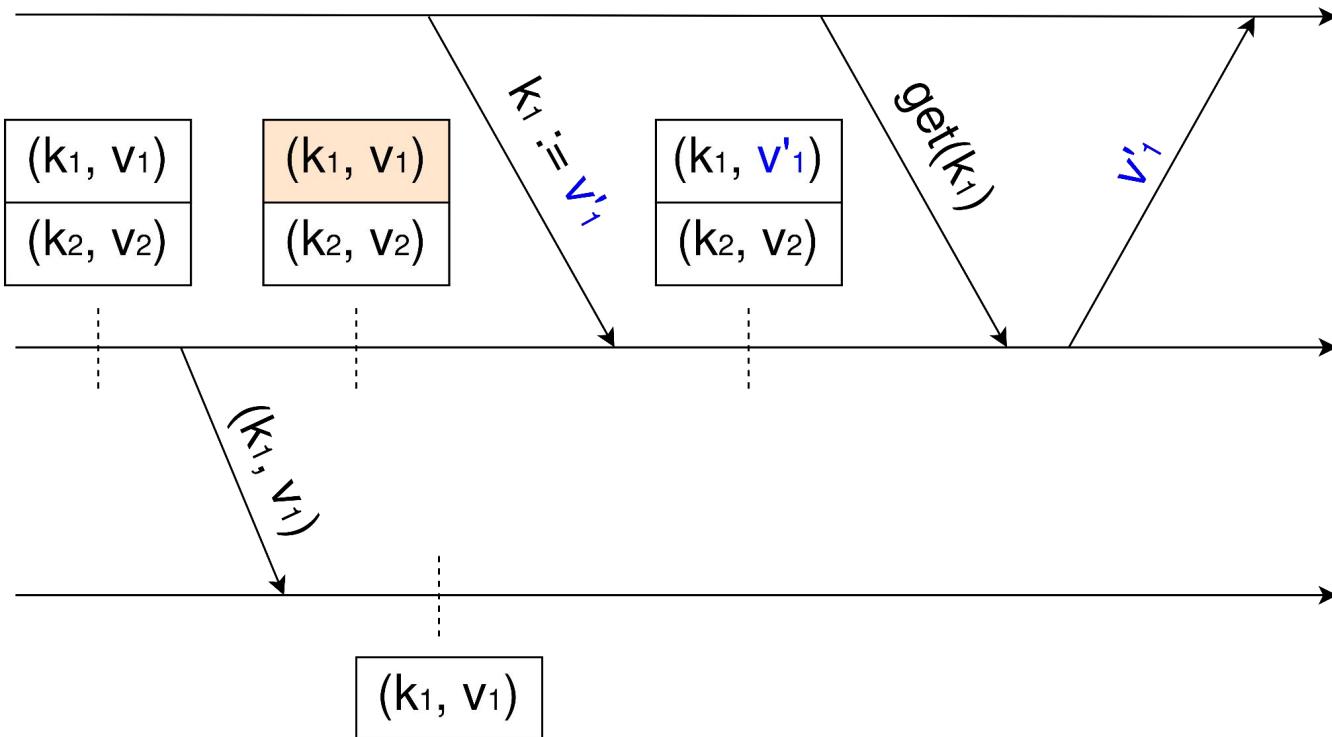
Решардинг: репликация изменений

- Старый хозяин обслуживает переносимые ключи
- В случае изменения перенесённого ключа, помечаем его как не перенесённый
- Потом переносит его ещё раз с новым значением



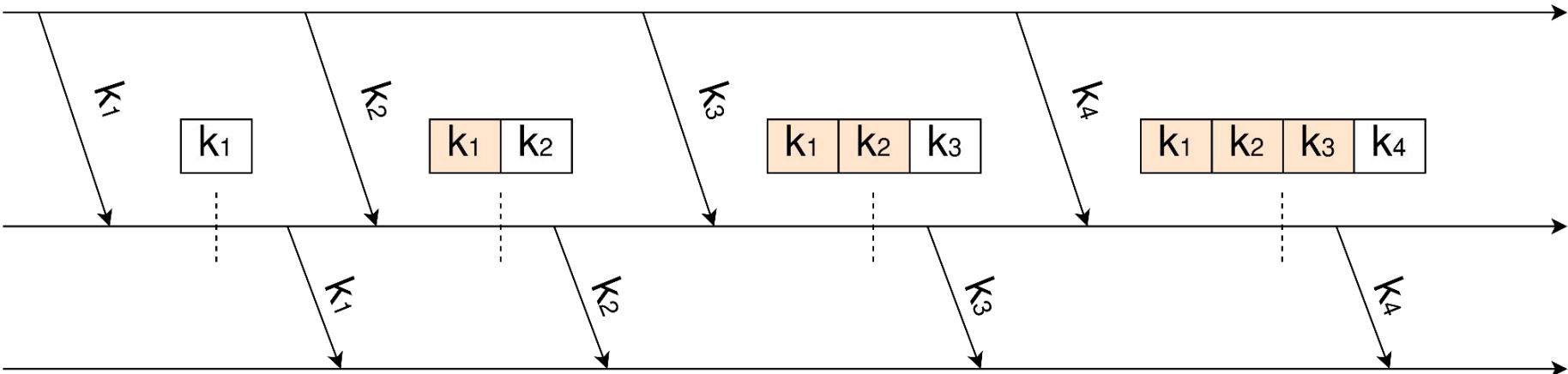
Решардинг: репликация изменений

- На читающие запросы отвечает старый владелец
- Его данные актуальные



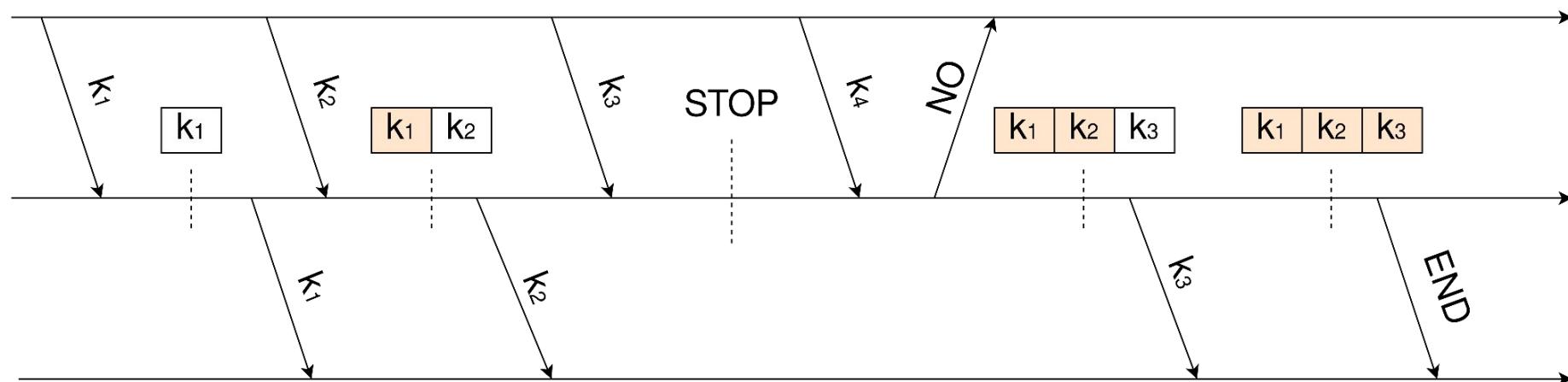
Решардинг: репликация изменений

- Может случиться так, что перенос ключей никогда не завершится
- Клиенты постоянно будут добавлять новые ключи



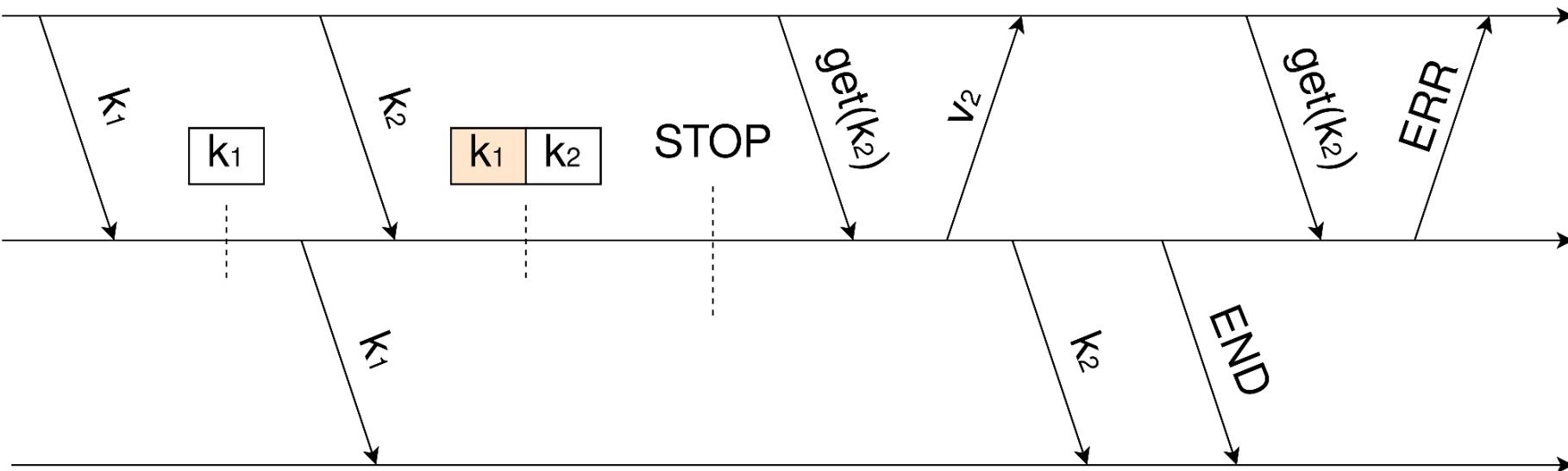
Решардинг: репликация изменений

- В какой-то момент старый владелец перестаёт принимать запросы на изменение
- Завершает перенос
- Передаёт владение
- На все запросы начинает отвечать новый владелец



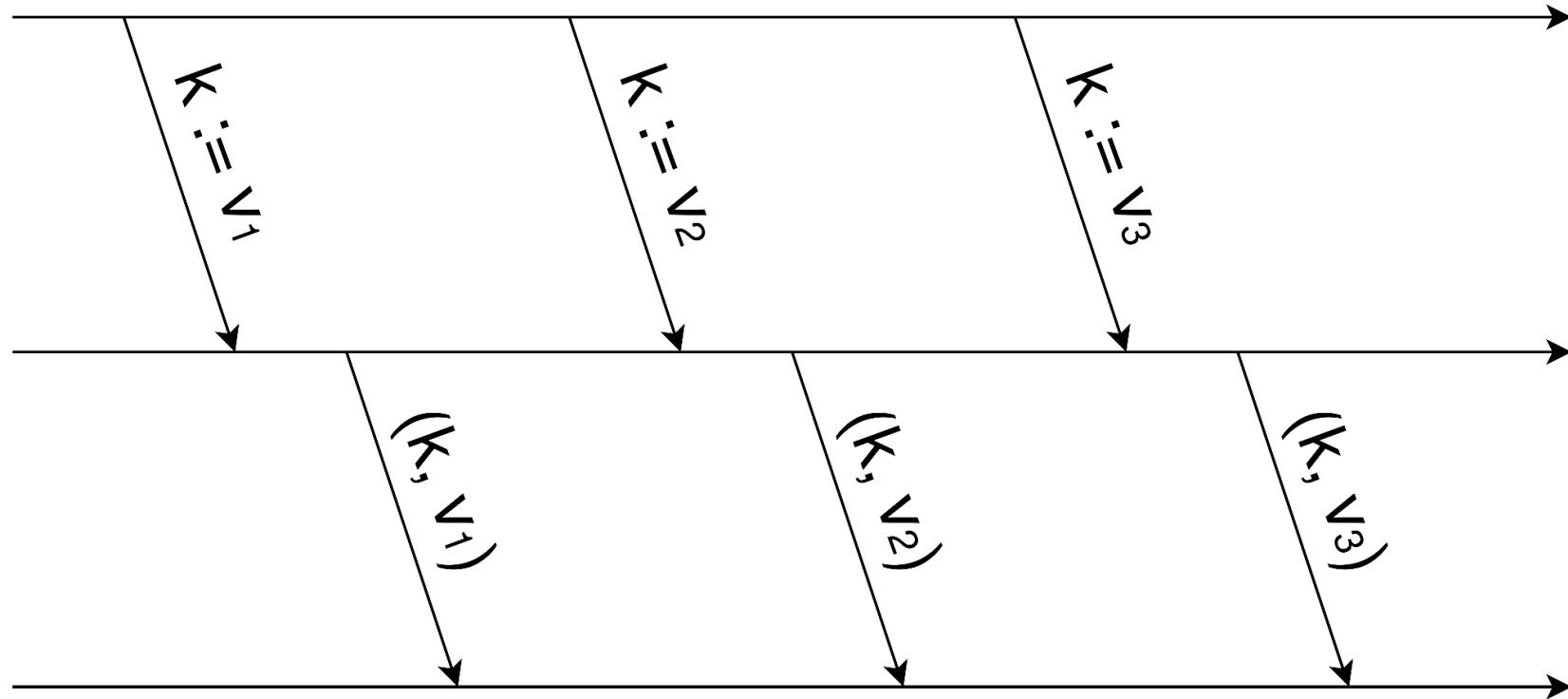
Решардинг: репликация изменений

- На читающие запросы можем отвечать даже после того, как прекратили отвечать на пишущие
- Но только до момента посылки сообщения о передаче ответственности



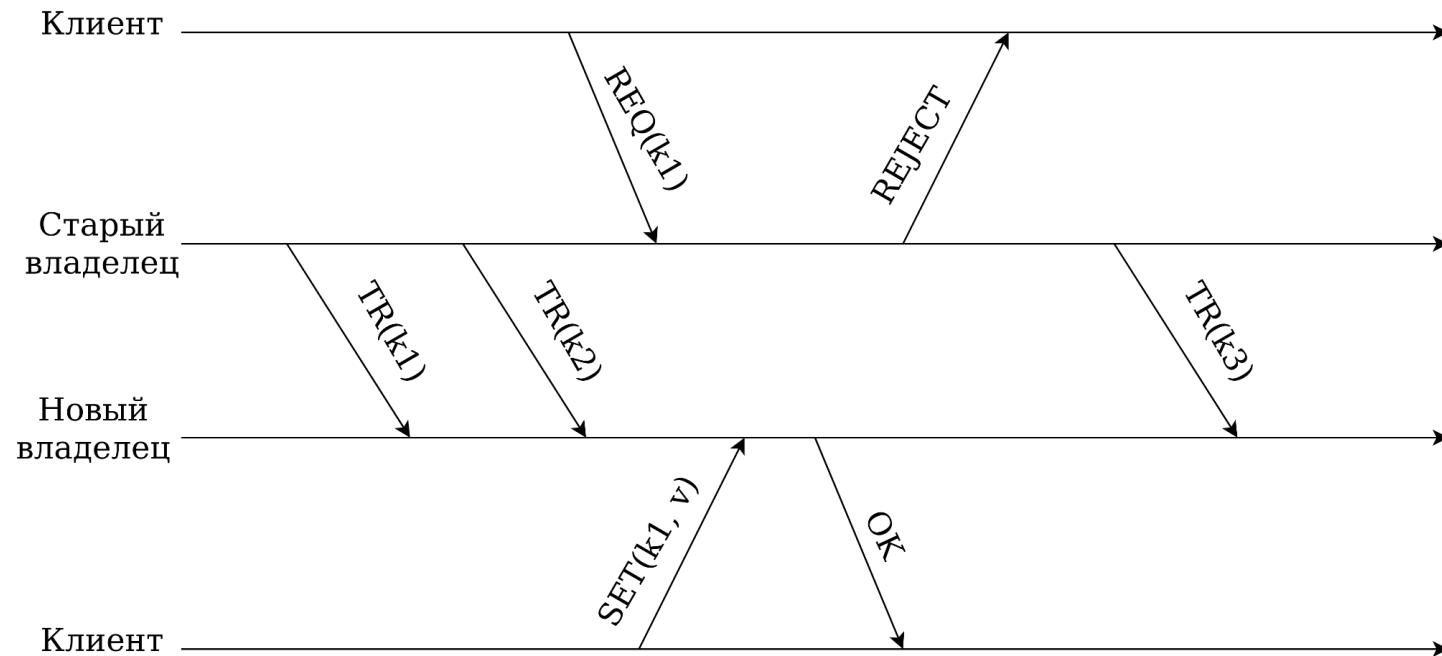
Репликация изменений: недостатки

- Если есть активный поток изменений, один и тот же ключ будем перемещать много раз



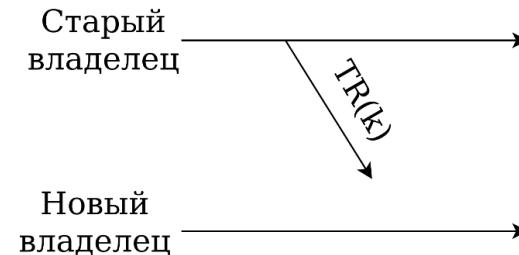
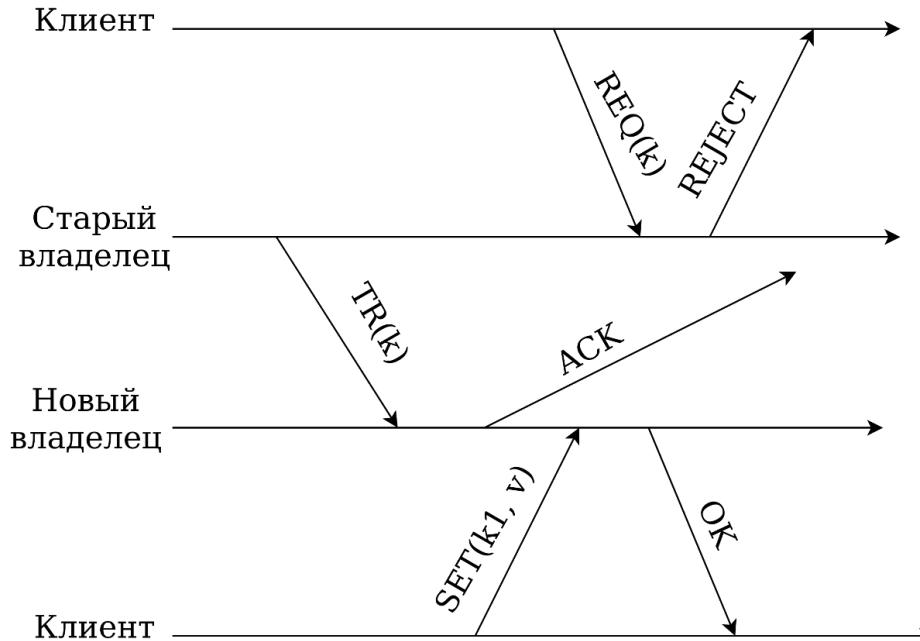
Перешардирование: оптимизация

- Помним, какие ключи мы уже передали новому владельцу
- Отказываемся исполнять запросы по таким ключам
- Запросы по ним исполняет новый владелец



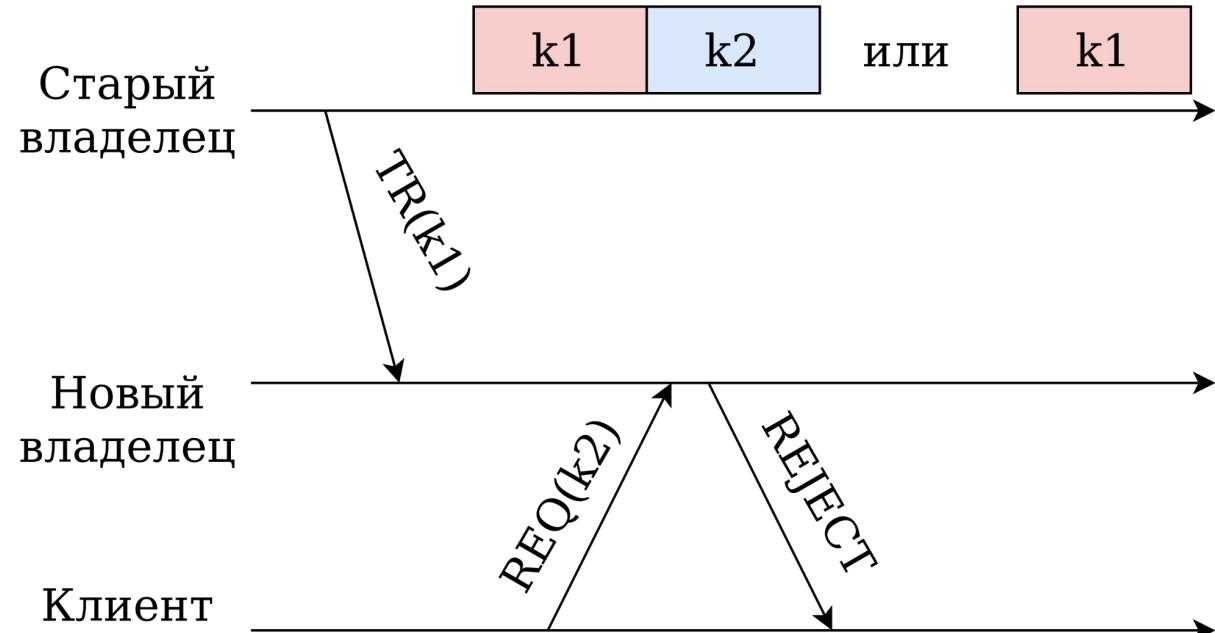
Перешардирование: ключи “в полёте”

- По некоторым ключам никто не может выполнять запросы: ни старый, ни новый владелец

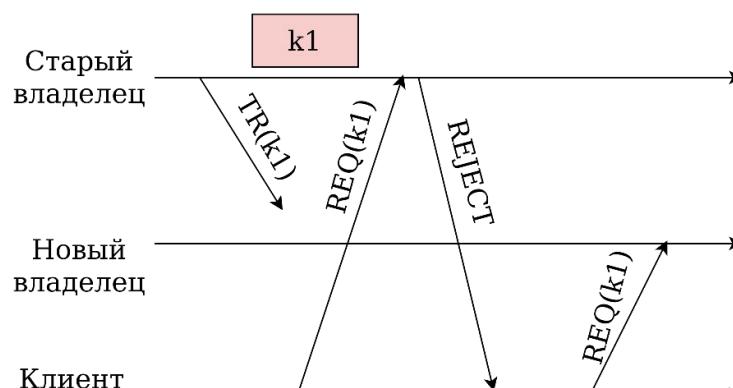
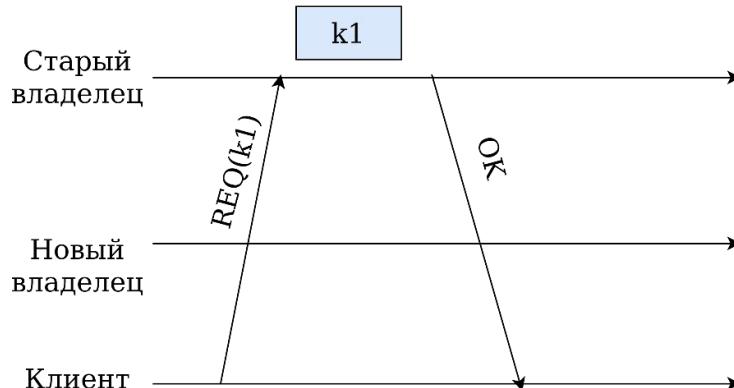
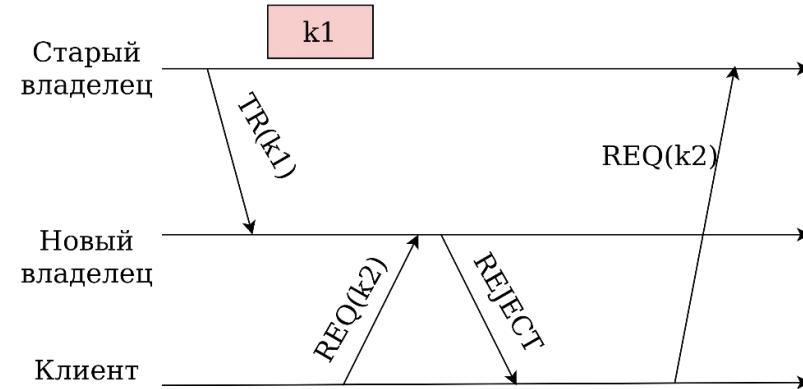
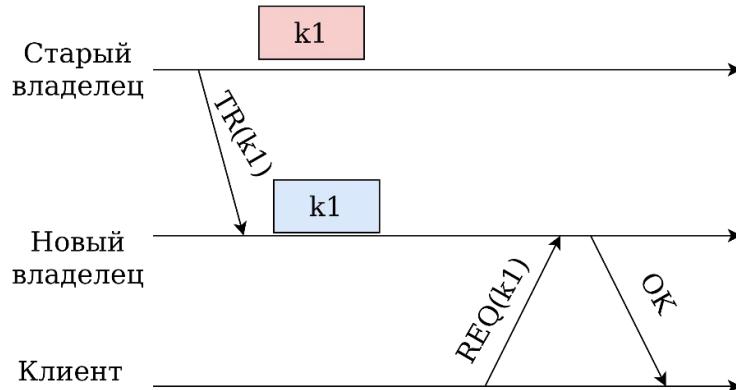


Перешардирование: несуществующие ключи

- Запросы по таким ключам не может принимать новый владелец
- Не может отличить несуществующий ключ от ключа, который ещё не перенесён

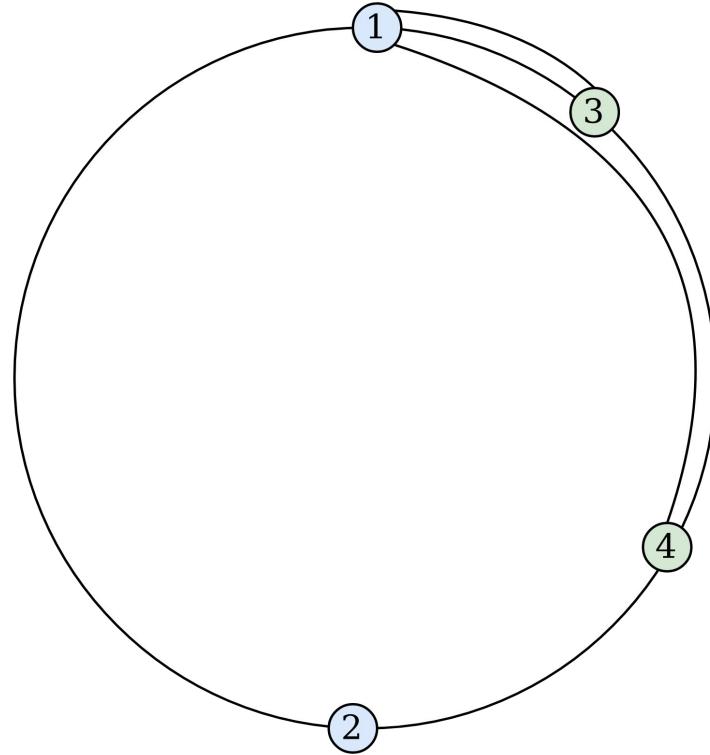


Перешардирование: подытожим



Упорядочивание перешардирований

- Одновременные перешардирования приводят к проблемам же диапазон $[node_1 + 1; node_3]$ может быть одновременно запрошен у второго узла двумя разными узлами (3 и 4)
- Перешардирования надо упорядочивать



Что почитать

- *Chang F. et al.* Bigtable: A distributed storage system for structured data
- *Huang D. et al.* TiDB: a Raft-based HTAP database
- *Corbett J. C. et al.* Spanner: Google's globally distributed database
- *Schütze H., Manning C. D., Raghavan P.* Introduction to information retrieval
- Ян Кисель. Индексация и булев поиск
- Data partitioning with chunks in MongoDB
- Live resharding in MongoDB
- Jeffrey Dean. Challenges in Building Large-Scale Information Retrieval Systems

Thanks for your attention

